

Operating Systems

Virtual Memory, x86, and Page Replacement

Daniel Gruss

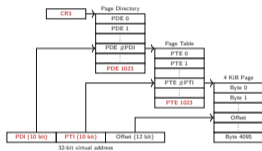
2023-10-05



1. Efficient Address Translation
2. Booting
3. Memory Layout
4. Page Replacement

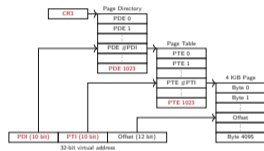
Efficient Address Translation

Efficient Address Translation



x86-32

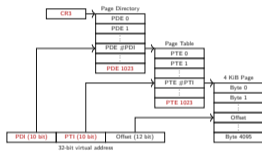
- What about speed?



x86-32



- What about speed?
- How many actual memory accesses per intended memory access?

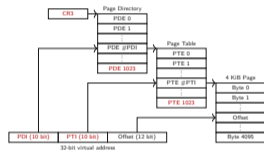


x86-32



Efficient Address Translation

- What about speed?
- How many actual memory accesses per intended memory access?

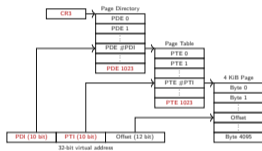


x86-32

- 1 access into page directory



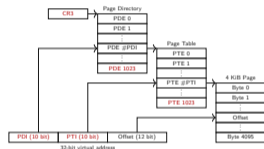
- What about speed?
- How many actual memory accesses per intended memory access?



x86-32

- 1 access into page directory
- 1 access into the page table page

- What about speed?
- How many actual memory accesses per intended memory access?



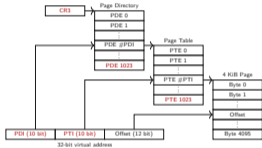
x86-32

- 1 access into page directory
- 1 access into the page table page
- 1 access into memory

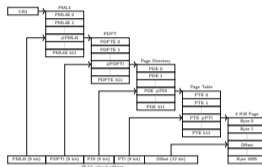


Efficient Address Translation

- What about speed?
- How many actual memory accesses per intended memory access?



x86-64

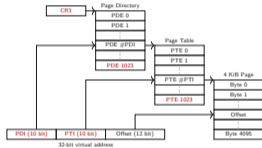


x86-32

- 1 access into page directory
- 1 access into the page table page
- 1 access into memory

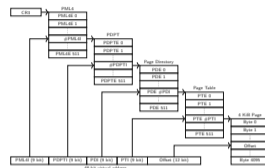
Efficient Address Translation

- What about speed?
- How many actual memory accesses per intended memory access?



x86-32

- 1 access into page directory
- 1 access into the page table page
- 1 access into memory

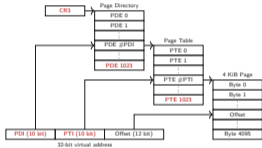


x86-64

- 1 access into page-map-level-4

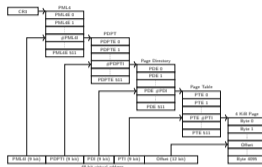
Efficient Address Translation

- What about speed?
- How many actual memory accesses per intended memory access?



x86-32

- 1 access into page directory
- 1 access into the page table page
- 1 access into memory

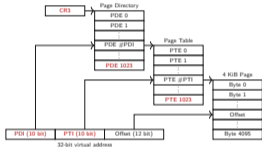


x86-64

- 1 access into page-map-level-4
- 1 access into the page-directory pointer

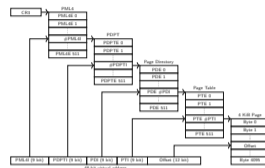
Efficient Address Translation

- What about speed?
- How many actual memory accesses per intended memory access?



x86-32

- 1 access into page directory
- 1 access into the page table page
- 1 access into memory

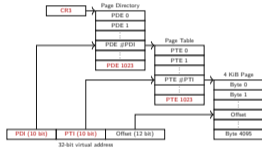


x86-64

- 1 access into page-map-level-4
- 1 access into the page-directory pointer
- 1 access into page directory

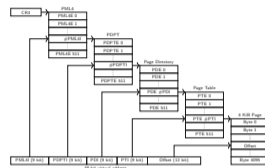
Efficient Address Translation

- What about speed?
- How many actual memory accesses per intended memory access?



x86-32

- 1 access into page directory
- 1 access into the page table page
- 1 access into memory

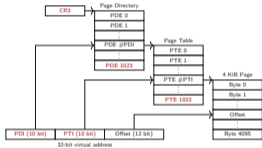


x86-64

- 1 access into page-map-level-4
- 1 access into the page-directory pointer
- 1 access into page directory
- 1 access into the page table page

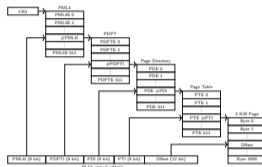
Efficient Address Translation

- What about speed?
- How many actual memory accesses per intended memory access?



x86-32

- 1 access into page directory
- 1 access into the page table page
- 1 access into memory



x86-64

- 1 access into page-map-level-4
- 1 access into the page-directory pointer
- 1 access into page directory
- 1 access into the page table page
- 1 access into memory



- Translation Look-aside Buffer (TLB)

- Translation Look-aside Buffer (TLB)
- Cache recent virtual \rightarrow physical translations



- Translation Look-aside Buffer (TLB)
- Cache recent virtual \rightarrow physical translations
- TLB: A page-table-entry cache

- Translation Look-aside Buffer (TLB)
- Cache recent virtual \rightarrow physical translations
- TLB: A page-table-entry cache
 - Cache hit: use translation

- Translation Look-aside Buffer (TLB)
- Cache recent virtual \rightarrow physical translations
- TLB: A page-table-entry cache
 - Cache hit: use translation
 - Cache miss: walk multi-level page table

- Translation Look-aside Buffer (TLB)
- Cache recent virtual \rightarrow physical translations
- TLB: A page-table-entry cache
 - Cache hit: use translation
 - Cache miss: walk multi-level page table
- TLB entry

- Translation Look-aside Buffer (TLB)
- Cache recent virtual \rightarrow physical translations
- TLB: A page-table-entry cache
 - Cache hit: use translation
 - Cache miss: walk multi-level page table
- TLB entry
 - virtual page number

- Translation Look-aside Buffer (TLB)
- Cache recent virtual \rightarrow physical translations
- TLB: A page-table-entry cache
 - Cache hit: use translation
 - Cache miss: walk multi-level page table
- TLB entry
 - virtual page number
 - physical page frame number

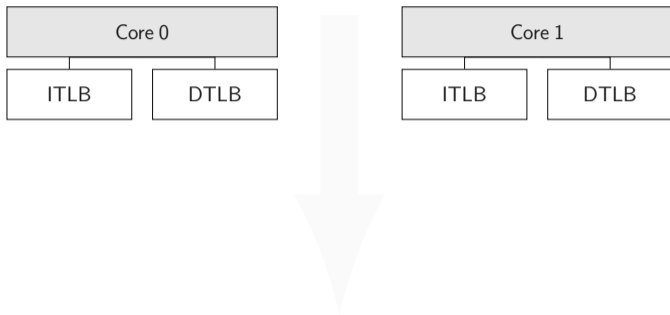


- Translation Look-aside Buffer (TLB)
- Cache recent virtual \rightarrow physical translations
- TLB: A page-table-entry cache
 - Cache hit: use translation
 - Cache miss: walk multi-level page table
- TLB entry
 - virtual page number
 - physical page frame number
 - access permissions

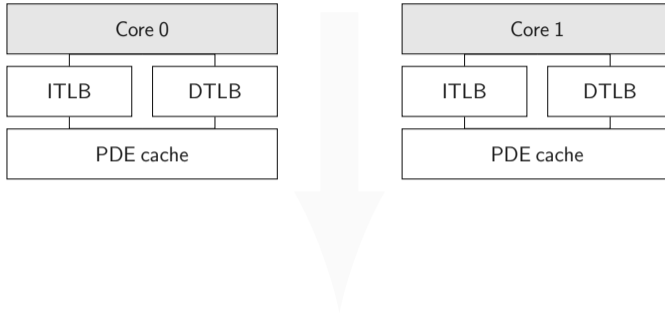
How many memory accesses per cache miss?



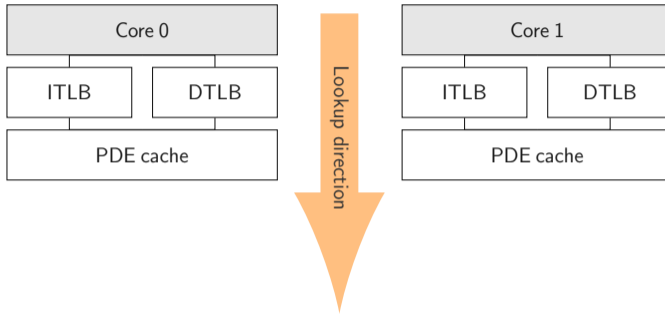
How many memory accesses per cache miss?



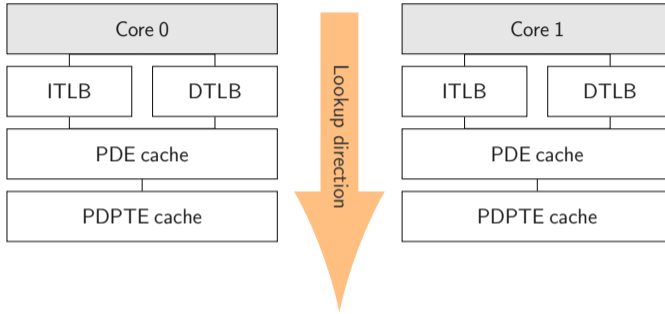
How many memory accesses per cache miss?



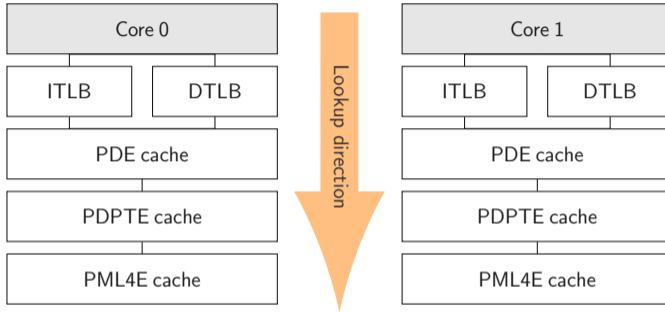
How many memory accesses per cache miss?



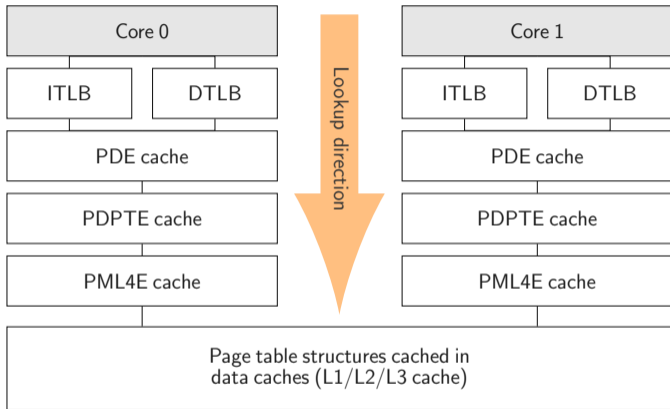
How many memory accesses per cache miss?



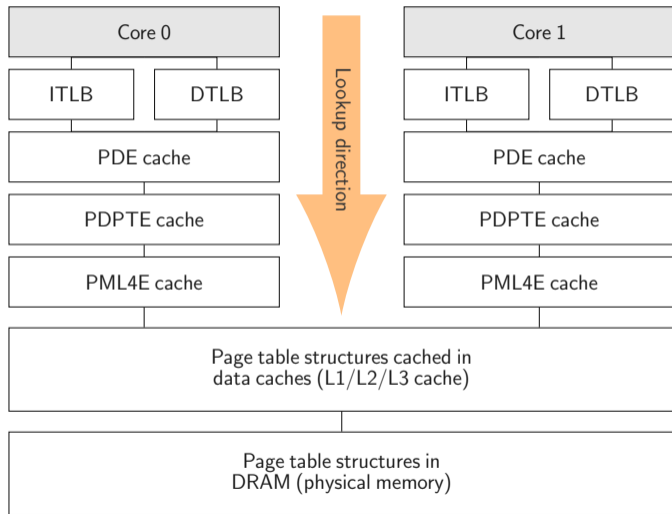
How many memory accesses per cache miss?



How many memory accesses per cache miss?



How many memory accesses per cache miss?








- Why does caching help?




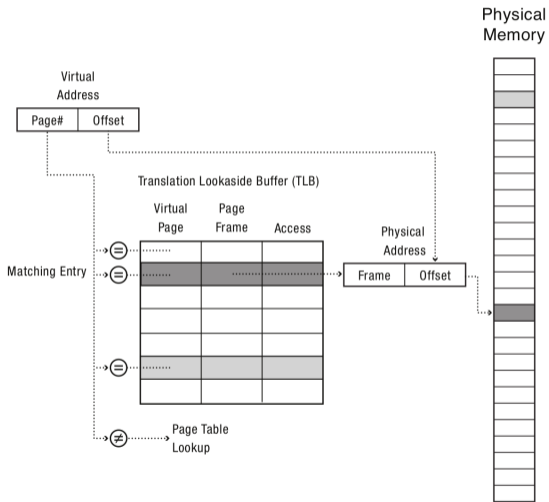
- Why does caching help?

- Why does caching help?
- Principle of locality 

- Why does caching help?
- Principle of locality 
 - If a memory address is accessed, likely nearby addresses are referenced in the future

- Why does caching help?
- Principle of locality 
 - If a memory address is accessed, likely nearby addresses are referenced in the future
 - Nearby: same page, uses identical address translation (without offset)

- Why does caching help?
- Principle of locality 
 - If a memory address is accessed, likely nearby addresses are referenced in the future
 - Nearby: same page, uses identical address translation (without offset)
 - High degree of locality: almost all page translations from TLB



When Do TLBs Work/Not Work?

- Video Frame Buffer: $32 \text{ bits} \times 1\text{K} \times 1\text{K} = 4\text{MB}$



When Do TLBs Work/Not Work?

- Video Frame Buffer: $32 \text{ bits} \times 1\text{K} \times 1\text{K} = 4\text{MB}$
- redraw screen - processor may touch every pixel



When Do TLBs Work/Not Work?

- Video Frame Buffer: $32 \text{ bits} \times 1\text{K} \times 1\text{K} = 4\text{MB}$
- redraw screen - processor may touch every pixel
- 1024 TLB entries required



Set of contiguous pages in physical memory that map a contiguous region of virtual memory

- e.g. 2 MB superpage consists of 512 regular pages (4 KB)

→ fewer TLB-Entries needed

Set of contiguous pages in physical memory that map a contiguous region of virtual memory

- e.g. 2 MB superpage consists of 512 regular pages (4 KB)
- aligned to lie on a 2 MB boundary

→ fewer TLB-Entries needed

How long does the TLB stay valid?



Context switches:

How long does the TLB stay valid?



Context switches:

- Do we have to invalidate the entire TLB?

How long does the TLB stay valid?



Context switches:

- Do we have to invalidate the entire TLB?

How long does the TLB stay valid?



Context switches:

- Do we have to invalidate the entire TLB?

Solution: Tagged TLB

Context switches:

- Do we have to invalidate the entire TLB?

Solution: Tagged TLB

- Each TLB entry has a tag (PID or CR3 or ...)

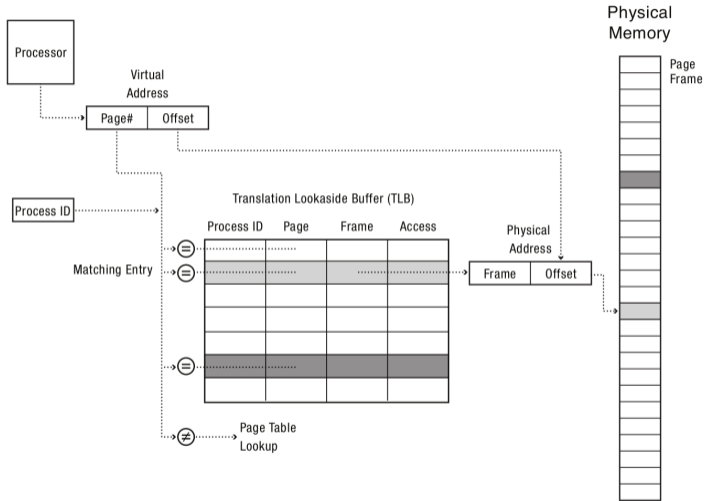
Context switches:

- Do we have to invalidate the entire TLB?

Solution: Tagged TLB

- Each TLB entry has a tag (PID or CR3 or ...)
- TLB hit only if tag matches current register state

Tagged TLB



How long does the TLB stay valid? (2)



What happens when OS changes permissions on a page?



What happens when OS changes permissions on a page?

- demand paging (zero on reference)
- copy on write

How long does the TLB stay valid? (2)



What happens when OS changes permissions on a page?

- demand paging (zero on reference)
- copy on write

TLB may contain old information

How long does the TLB stay valid? (2)



What happens when OS changes permissions on a page?

- demand paging (zero on reference)
- copy on write

TLB may contain old information

- OS must ask hardware to purge TLB entry

How long does the TLB stay valid? (2)



What happens when OS changes permissions on a page?

- demand paging (zero on reference)
- copy on write

TLB may contain old information

- OS must ask hardware to purge TLB entry

On a multicore: TLB shutdown

How long does the TLB stay valid? (2)



What happens when OS changes permissions on a page?

- demand paging (zero on reference)
- copy on write

TLB may contain old information

- OS must ask hardware to purge TLB entry

On a multicore: TLB shutdown

- OS must ask each CPU to purge TLB entry

Booting

- 16 bit mode



- 16 bit mode
- Address space: 1 MB

- 16 bit mode
- Address space: 1 MB
- How is that possible?



- 16 bit mode
- Address space: 1 MB
- How is that possible?
- CS register has a 20-bit base address

- 16 bit mode
- Address space: 1 MB
- How is that possible?
- CS register has a 20-bit base address
 - actually only 4 bit, but shifted by 16 bits to the left

- 16 bit mode
 - Address space: 1 MB
 - How is that possible?
 - CS register has a 20-bit base address
 - actually only 4 bit, but shifted by 16 bits to the left
- 4 bits (base/prefix) + 16 bits (address/offset) = 20 bit address

9.1.4 First Instruction Executed

The first instruction that is fetched and executed following a hardware reset is located at physical address FFFFFFF0H. This address is 16 bytes below the processor's uppermost physical address. The EPROM containing the software-initialization code must be located at this address.

- Address: `0xFFFFFFFF0`

- Address: `0xFFFFFFFF0`
- How is that possible?

- Address: `0xFFFFFFFF0`
- How is that possible?
 - CS register also has a 32-bit base address (initialized to `0xFFFF0000`)

- Address: `0xFFFFFFFF0`
- How is that possible?
 - CS register also has a 32-bit base address (initialized to `0xFFFF0000`)
- What if I have < 4 GB RAM?

- Address: `0xFFFFFFFF0`
- How is that possible?
 - CS register also has a 32-bit base address (initialized to `0xFFFF0000`)
- What if I have < 4 GB RAM?
 - physical address space \neq RAM directly mapped

```
00000000-007fffffff (prio 0, RW): alias ram-below-4g (this is our RAM)
000a0000-000bffff (prio 1, RW): vga-lowmem (remember for later)
000c0000-000dffff (prio 1, RW): pc.rom
000e0000-000ffffff (prio 1, R-): alias isa-bios
fd000000-fdffffff (prio 1, RW): vga.vram
febc0000-febdffff (prio 1, RW): e1000-mmio
febf0400-febf041f (prio 0, RW): vga ioports remapped
febf0500-febf0515 (prio 0, RW): bochs dispi interface
febf0600-febf0607 (prio 0, RW): qemu extended regs
fffc0000-ffffffff (prio 0, R-): pc.bios (ahhh!)
...
```



- BIOS initializes hardware platform

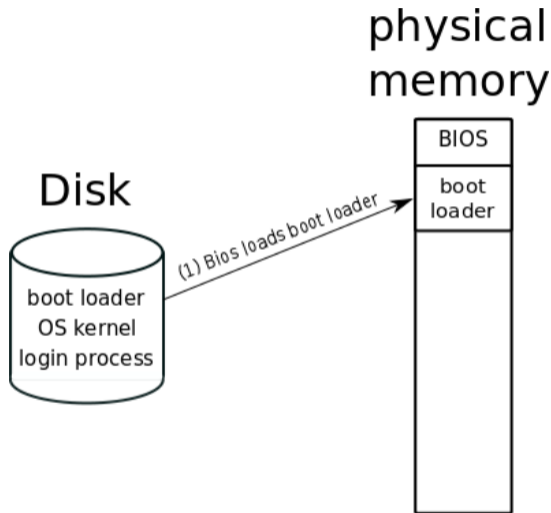


- BIOS initializes hardware platform
- Switch to protected mode (32 bit)

- BIOS initializes hardware platform
- Switch to protected mode (32 bit)
- Select a device to boot from

- BIOS initializes hardware platform
- Switch to protected mode (32 bit)
- Select a device to boot from
- Load MBR from device into memory

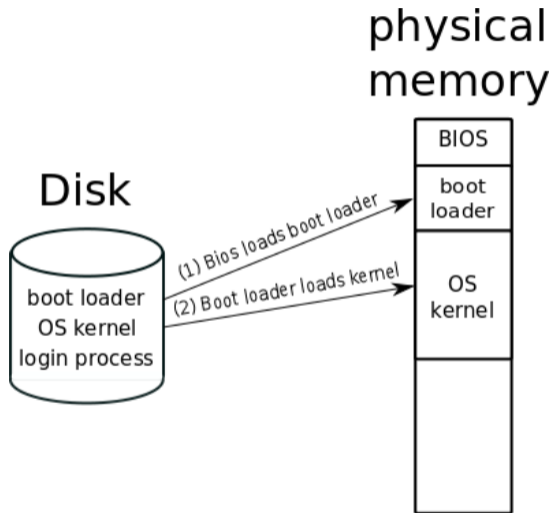
- BIOS initializes hardware platform
- Switch to protected mode (32 bit)
- Select a device to boot from
- Load MBR from device into memory
- Execute code from MBR



- Boot loader for Linux, SWEb, ...
- Loads the OS image from disk and starts OS

GRUB

One of the important features in GRUB is flexibility; GRUB understands file-systems and kernel executable formats, so you can load an arbitrary operating system the way you like, without recording the physical position of your kernel on the disk. Thus you can load the kernel just by specifying its file name and the drive and partition where the kernel resides.



- Prepare hardware



- Prepare hardware
- Start device drivers and initialize devices

- Prepare hardware
- Start device drivers and initialize devices
- Start initial processes (e.g. *init*-process)



Kernel is a compiled binary (e.g. an ELF binary)

Kernel is a compiled binary (e.g. an ELF binary)

```
% readelf -a kernel.x | grep Entry  
Entry point address:          0x801001ba
```


Kernel is a compiled binary (e.g. an ELF binary)

```
% readelf -a kernel.x | grep Entry
Entry point address:          0x801001ba
```

```
% objdump -S kernel.x | less
801001ba <entry>:
801001ba:      55                push   %ebp
801001bb:      89 e5            mov    %esp,%ebp
801001bd:      83 ec 10        sub   $0x10,%esp
801001c0:      89 1d 00 90 14 00  mov   %ebx,0x149000
```

Kernel is a compiled binary (e.g. an ELF binary)

```
% readelf -a kernel.x | grep Entry
Entry point address:          0x801001ba
```

```
% objdump -S kernel.x | less
801001ba <entry>:
801001ba:      55                push   %ebp
801001bb:      89 e5             mov    %esp,%ebp
801001bd:      83 ec 10          sub   $0x10,%esp
801001c0:      89 1d 00 90 14 00 mov   %ebx,0x149000
```

Wait, that's C-Code!

```
extern "C" void entry()
{
    asm("mov %ebx,multi_boot_structure_pointer - BASE");

    PRINT("Booting...\n");
}
```

```
PRINT("Clearing Framebuffer...\n");  
memset((char*) 0xB8000, 0, 80 * 25 * 2);  
  
PRINT("Clearing BSS...\n");  
char* bss_start = TRUNCATE(&bss_start_address);  
memset(bss_start, 0, TRUNCATE(&bss_end_address) - bss_start);  
  
PRINT("Initializing Kernel Paging Structures...\n");  
//...
```

```
PRINT("Enable PSE and PAE...\n");
asm("mov %cr4,%eax\n"
    "or $0x20, %eax\n"
    "mov %eax,%cr4\n");

PRINT("Setting CR3 Register...\n");
asm("mov %[pd],%%cr3" : : [pd]"r"(TRUNCATE(kernel_page_map_level_4)));

PRINT("Enable EFER.LME and EFER.NXE...\n");
asm("mov $0xC0000080,%ecx\n"
    "rdmsr\n"
    "or $0x900,%eax\n"
    "wrmsr\n");
//...
PRINT("Enable Paging...\n");
asm("mov %cr0,%eax\n"
    "or $0x80000001,%eax\n"
    "mov %eax,%cr0\n");
```

```
PRINT("Setup TSS...\n");
TSS* g_tss_p = (TSS*) TRUNCATE(&g_tss);
g_tss_p->ist0_h = -1U;
g_tss_p->ist0_l = (uint32) TRUNCATE(boot_stack) | 0x80004000;
g_tss_p->rsp0_h = -1U;
g_tss_p->rsp0_l = (uint32) TRUNCATE(boot_stack) | 0x80004000;
```

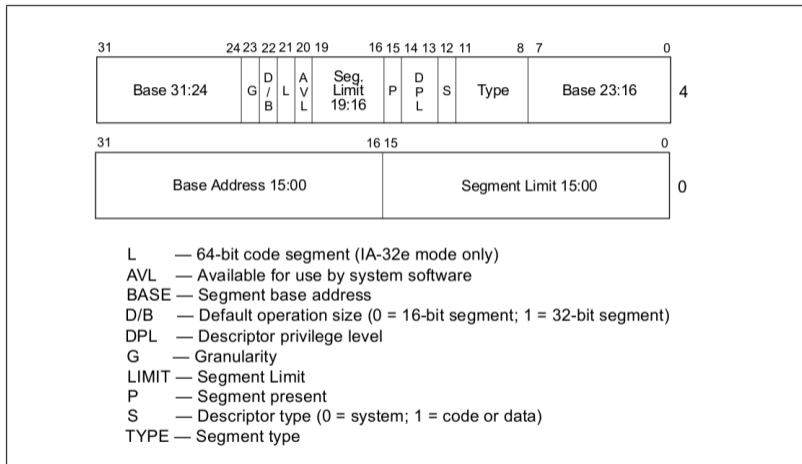


Figure 3-8. Segment Descriptor

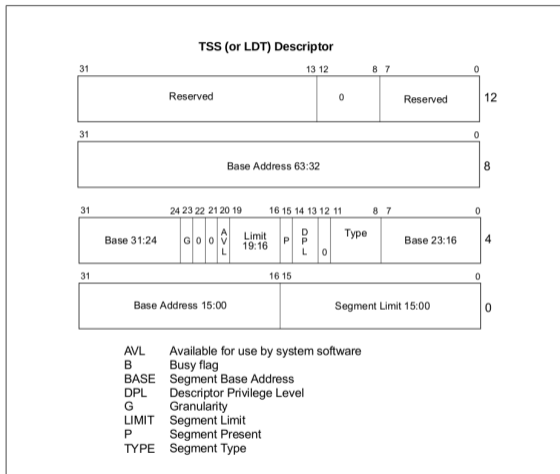


Figure 7-4. Format of TSS and LDT Descriptors in 64-bit Mode


```
static void setSegmentDescriptor(uint32 index, uint32 baseH, uint32 baseL, uint32
    limit, uint8 dpl, uint8 code, uint8 tss);

PRINT("Setup Segments...\n");
setSegmentDescriptor(1, 0, 0, 0, 0, 1, 0);
setSegmentDescriptor(2, 0, 0, 0, 0, 0, 0);
setSegmentDescriptor(3, 0, 0, 0, 3, 1, 0);
setSegmentDescriptor(4, 0, 0, 0, 3, 0, 0);
setSegmentDescriptor(5, -1U, (uint32) TRUNCATE(&g_tss) | 0x80000000,
    sizeof(TSS) - 1, 0, 0, 1);

PRINT("Loading Long Mode GDT...\n");

struct GDT32Ptr gdt32_ptr;
gdt32_ptr.limit = sizeof(gdt) - 1;
gdt32_ptr.addr = (uint32) TRUNCATE(gdt);
asm("lgdt %[gdt_ptr]" : : [gdt_ptr]"m"(gdt32_ptr));
// ...
```

```
PRINT("Setting Long Mode Segment Selectors...\n");
asm("mov %%ax, %%ds\n"
    "mov %%ax, %%es\n"
    "mov %%ax, %%ss\n"
    "mov %%ax, %%fs\n"
    "mov %%ax, %%gs\n"
    : : "a"(KERNEL_DS));

PRINT("Calling entry64()...\n");
asm("ljmp %[cs], $entry64-BASE\n" : : [cs]"i"(KERNEL_CS));

PRINT("Returned from entry64()? This should never happen.\n");
asm("hlt");
}
```

```
PRINT("Setting Long Mode Segment Selectors...\n");
asm("mov %%ax, %%ds\n"
    "mov %%ax, %%es\n"
    "mov %%ax, %%ss\n"
    "mov %%ax, %%fs\n"
    "mov %%ax, %%gs\n"
    : : "a"(KERNEL_DS));

PRINT("Calling entry64()...\n");
asm("ljmp %[cs], $entry64-BASE\n" : : [cs]"i"(KERNEL_CS));

PRINT("Returned from entry64()? This should never happen.\n");
asm("hlt");
}
```

```
extern "C" void entry64()
{
    PRINT("Parsing Multiboot Header...\n");
    parseMultibootHeader();
    PRINT("Initializing Kernel Paging Structures...\n");
    initialisePaging();
    PRINT("Setting CR3 Register...\n");
    asm("mov %%rax, %%cr3" : : "a"(VIRTUAL_TO_PHYSICAL_BOOT(ArchMemory::
        getRootOfKernelPagingStructure())));
    PRINT("Switch to our own stack...\n");
    asm("mov %[stack], %%rsp\n"
        "mov %[stack], %%rbp\n" : : [stack]"i"(boot_stack + 0x4000));
}
```

```
PRINT("Loading Long Mode Segments...\n");

gdt_ptr.limit = sizeof(gdt) - 1;
gdt_ptr.addr = (uint64)gdt;
asm("lgdt (%rax)" : : "a"(&gdt_ptr));
asm("mov %%ax, %%ds\n"
    "mov %%ax, %%es\n"
    "mov %%ax, %%ss\n"
    "mov %%ax, %%fs\n"
    "mov %%ax, %%gs\n"
    : : "a"(KERNEL_DS));
asm("ltr %%ax" : : "a"(KERNEL_TSS));
PRINT("Calling startup()...\n");
asm("jmp *[startup]" : : [startup]"r"(startup));
while (1);
}
```

```
extern "C" void startup()
{
    writeLine2Bochs("Removing Boot Time Ident Mapping...\n");
    removeBootTimeIdentMapping();
    system_state = BOOTING;

    PageManager::instance();
    writeLine2Bochs("PageManager and KernelMemoryManager created \n");

    main_console = ArchCommon::createConsole(1);
    writeLine2Bochs("Console created \n");
    // ...
}
```

```
Scheduler::instance();

//needs to be done after scheduler and terminal, but prior to enableInterrupts
kprintf_init();

debug(MAIN, "Threads init\n");
ArchThreads::initialise();
debug(MAIN, "Interupts init\n");
ArchInterrupts::initialise();

ArchInterrupts::setTimerFrequency(IRQ0_TIMER_FREQUENCY);
```

```
ArchCommon::initDebug();

vfs.initialize();
debug(MAIN, "Mounting DeviceFS under /dev/\n");
DeviceFSType *devfs = new DeviceFSType();
vfs.registerFileSystem(devfs);
default_working_dir = vfs.root_mount("devicefs", 0);

debug(MAIN, "Block Device creation\n");
BDManager::getInstance()->doDeviceDetection();
debug(MAIN, "Block Device done\n");

for (BDVirtualDevice* bdvd : BDManager::getInstance()->device_list_)
{
    debug(MAIN, "Detected Device: %s :: %d\n", bdvd->getName(), bdvd->
        getDeviceNumber());
}
```



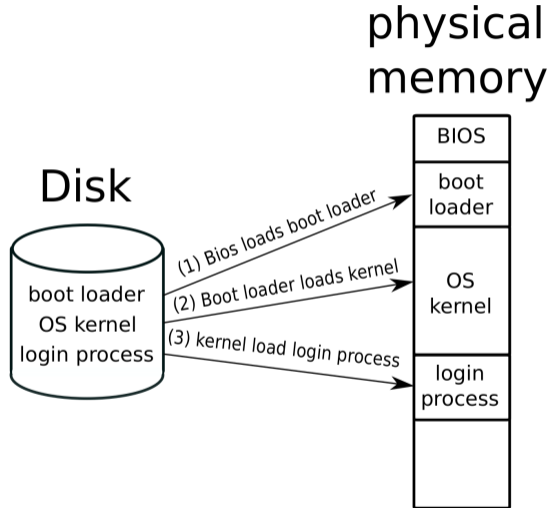
```
// initialise global and static objects
extern ustl::list<FileDescriptor*> global_fd;
new (&global_fd) ustl::list<FileDescriptor*>();
extern Mutex global_fd_lock;
new (&global_fd_lock) Mutex("global_fd_lock");
// ...
debug(MAIN, "Timer enable\n");
ArchInterrupts::enableTimer();
KeyboardManager::instance();
ArchInterrupts::enableKBD();
```

```
debug(MAIN, "Adding Kernel threads\n");
Scheduler::instance()->addNewThread(main_console);
Scheduler::instance()->addNewThread(new ProcessRegistry(new FileSystemInfo(*
    default_working_dir), user_progs /*see user_progs.h*/));
Scheduler::instance()->printThreadList();

kprintf("Now enabling Interrupts...\n");
system_state = RUNNING;

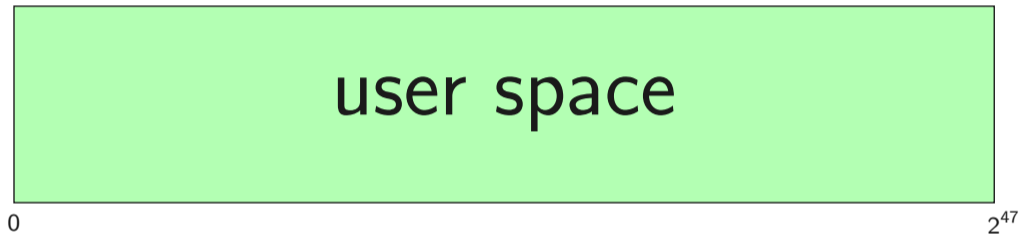
ArchInterrupts::enableInterrupts();

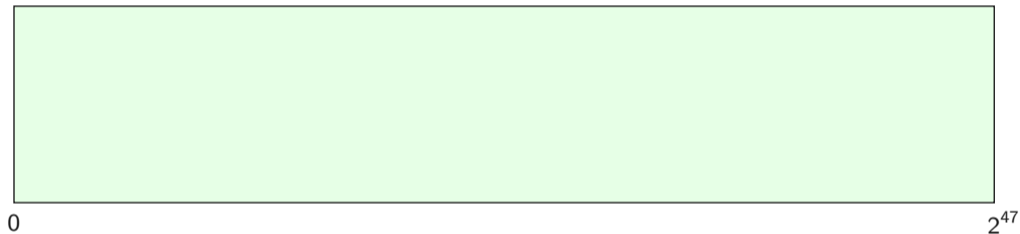
Scheduler::instance()->yield();
//not reached
assert(false);
}
```

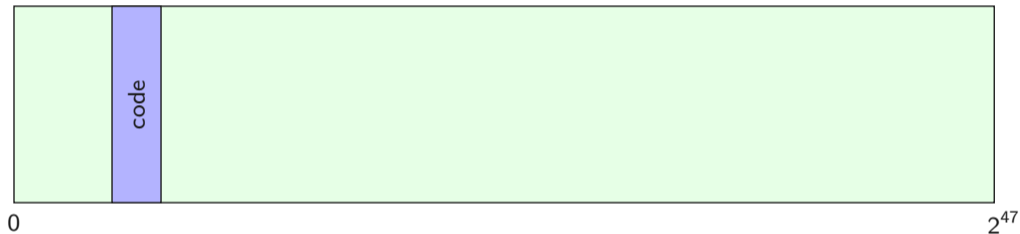


Memory Layout

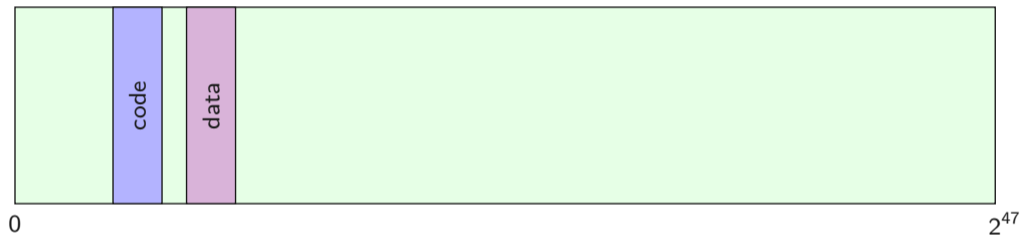
- Parse binary (headers)
- different binary formats
 - .COM - program always starts at byte 256 (also used in CP/M)
 - a.out
 - COFF
 - **Executable and Linking Format (ELF)**



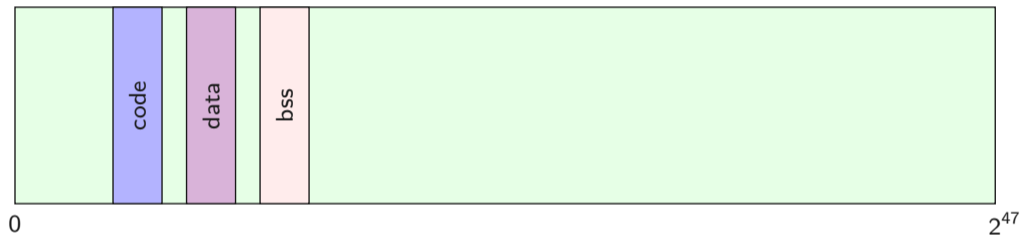




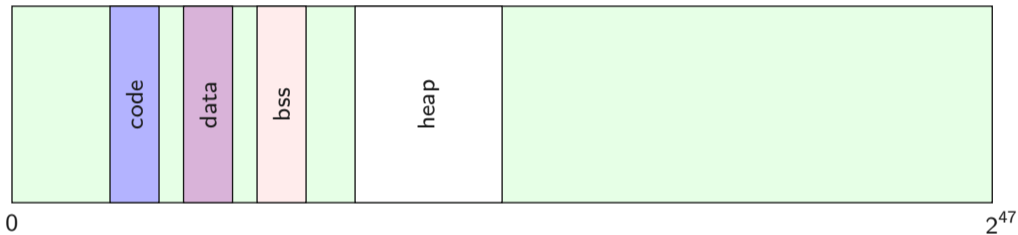
Memory Layout User Space on



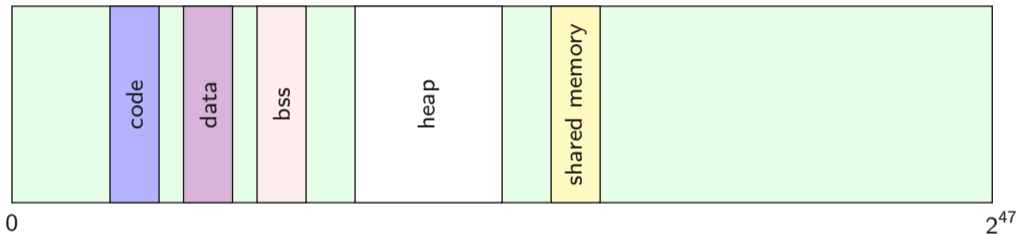
Memory Layout User Space on



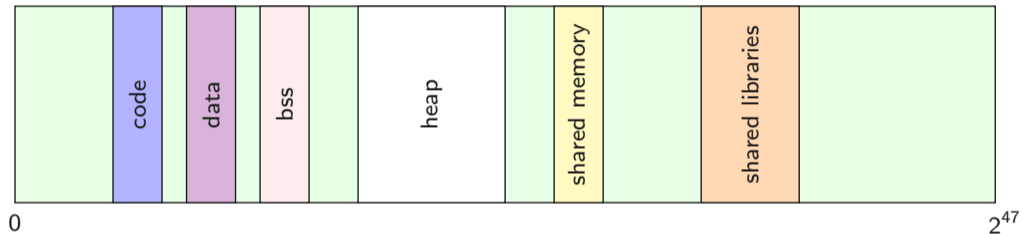
Memory Layout User Space on



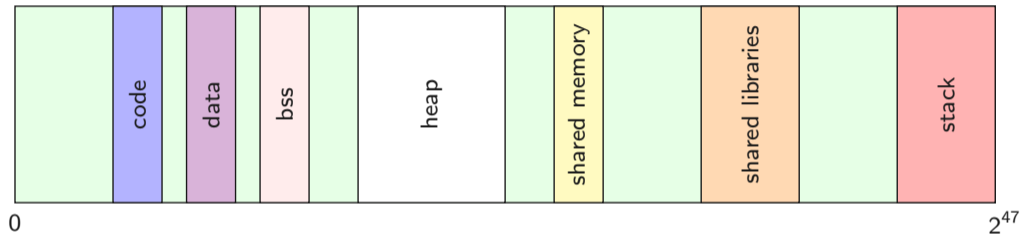
Memory Layout User Space on

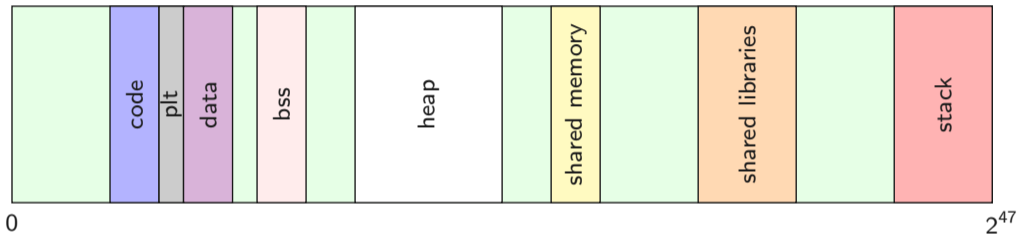


Memory Layout User Space on



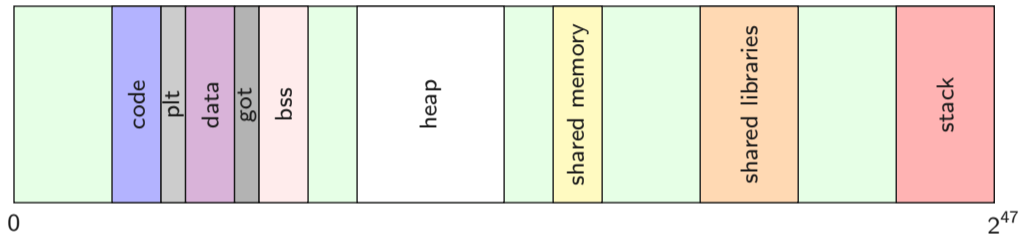
Memory Layout User Space on





- PLT: Procedure Linkage Table

Memory Layout User Space on



- PLT: Procedure Linkage Table
- GOT: Global Object Table



- Executable
- Usually readable
- Usually not writable

```
me@nux:~$ ./mini
me@nux:~$ echo $?
42
```

```
0 1 2 3 4 5 6 7 8 9 A B C D E F
00: 7F .E .L .F 01 01 01
10: 02 00 03 00 01 00 00 00 60 00 00 08 40 00 00 00
20:
34 00 20 00 01 00
40: 01 00 00 00 00 00 00 00 00 00 08 00 00 00 08
50: 70 00 00 00 70 00 00 00 05 00 00 00
60: BB 2A 00 00 00 B8 01 00 00 00 CD 80
```

ELF HEADER
IDENTIFY AS AN ELF TYPE
SPECIFY THE ARCHITECTURE

PROGRAM HEADER TABLE
EXECUTION INFORMATION

CODE

FIELDS	VALUES
e_ident	
EI_MAG	0x7F, "ELF"
EI_CLASS, EI_DATA	1ELFCLASS32, 1ELFDATA2LSB
EI_VERSION	1EV_CURRENT
e_type	2ET_EXEC
e_machine	3EM_386
e_version	1EV_CURRENT
e_entry	0x8000060
e_phoff	0x0000040
e_ehsize	0x0034
e_phentsize	0x0020
e_phnum	0001
p_type	1PT_LOAD
p_offset	0
p_vaddr	0x8000000
p_paddr	0x8000000
p_filesz	0x0000070
p_memsz	0x0000070
p_flags	5PF_R PF_X

```
X86 ASSEMBLY      EQUIVALENT C CODE
mov ebx, 42
mov eax, 1          SC_EXIT
int 80h            return 42;
```

- object files (compiled code)

- object files (compiled code)
- dynamic libraries

- object files (compiled code)
- dynamic libraries
- static libraries

- What about the stack?

- What about the stack?
- Size? Address?

- What about the stack?
- Size? Address?
 - Locate a suitable address area for the stack

- What about the stack?
- Size? Address?
 - Locate a suitable address area for the stack
 - Define the initial size of the area

- What about the stack?
- Size? Address?
 - Locate a suitable address area for the stack
 - Define the initial size of the area
- Load on demand

- What about the stack?
- Size? Address?
 - Locate a suitable address area for the stack
 - Define the initial size of the area
- Load on demand
 - Data from binary

- What about the stack?
- Size? Address?
 - Locate a suitable address area for the stack
 - Define the initial size of the area
- Load on demand
 - Data from binary
 - Zeros (security!)

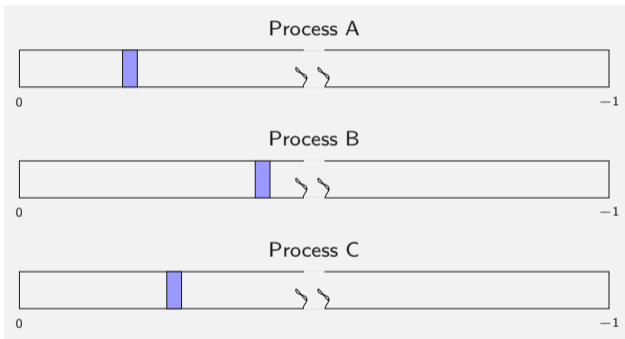


Every program start, use different
random offsets for



Every program start, use different random offsets for

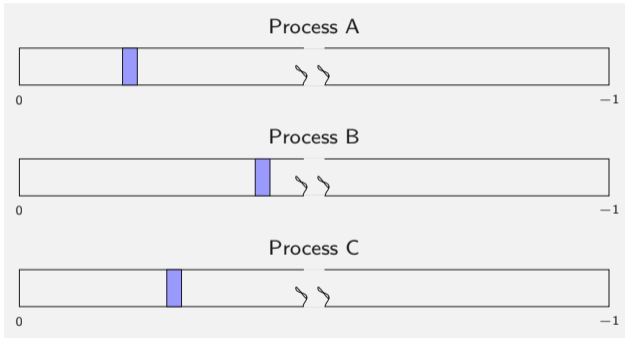
- program sections





Every program start, use different random offsets for

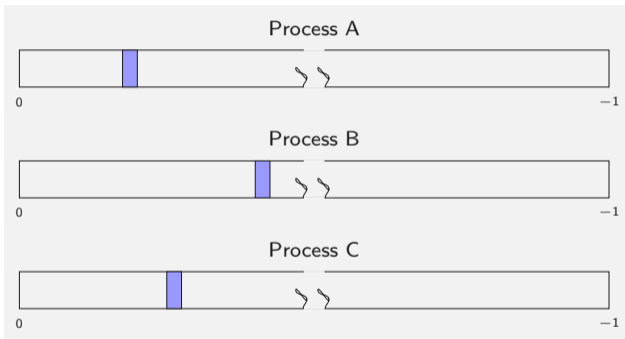
- program sections
- libraries





Every program start, use different random offsets for

- program sections
- libraries
- heap

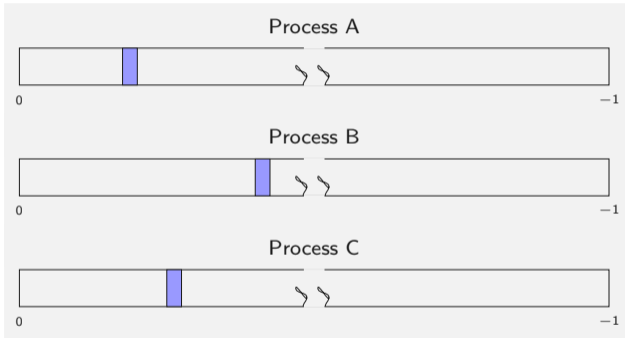




Every program start, use different random offsets for

- program sections
- libraries
- heap
- stacks

→ Addresses are unpredictable for an attacker

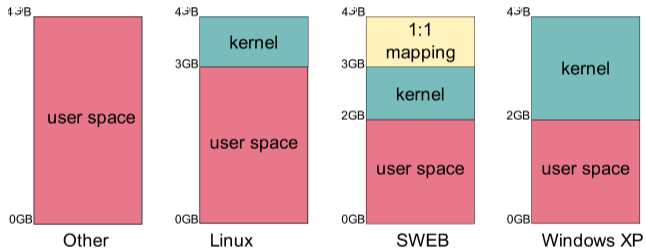


- OS has to layout the linear memory for a process

- OS has to layout the linear memory for a process
- only addresses can be accessed that are mapped into the process address space via the page table mechanism

- OS has to layout the linear memory for a process
- only addresses can be accessed that are mapped into the process address space via the page table mechanism
- decision: do we also map the kernel into the process address space?

Memory layout



- 32-bit addresses: memory locations between 0 GB and 4 GB

- 32-bit addresses: memory locations between 0 GB and 4 GB
- x86 requires a minimal region of the kernel to be mapped (for context switches)



- 32-bit addresses: memory locations between 0 GB and 4 GB
- x86 requires a minimal region of the kernel to be mapped (for context switches)
- typically a large part of the linear address space is reserved for the kernel





- 32-bit addresses: memory locations between 0 GB and 4 GB
- x86 requires a minimal region of the kernel to be mapped (for context switches)
- typically a large part of the linear address space is reserved for the kernel
- inaccessible due to userspace permission bit (set to 0 for kernel pages)


```
0000000000000000-00007fffffffffffffff (=47 bits) user space
ffff800000000000-ffff87fffffffffffffff (=43 bits) hypervisor
ffff880000000000-ffffc7fffffffffffffff (=64 TB) identity mapping
ffffc90000000000-ffffe8fffffffffffffff (=45 bits) vmalloc/ioremap space
ffffea0000000000-fffffeafffffffffffffffff (=40 bits) virtual memory map
fffffec0000000000-fffffbfffffffffffffff (=44 bits) KASAN shadow memory
fffffff0000000000-fffffff7fffffffffffffff (=39 bits) ESP fixup stacks
fffffffef000000000-fffffffeffffffffffffffff (=64 GB) EFI region mappings
fffffffff80000000-fffffffff9fffffffffffff (=512 MB) kernel code/data
ffffffffffa0000000-ffffffffffff5fffff (=1526 MB) kernel modules
ffffffffffff600000-ffffffffffffdfffff (=8 MB) vsyscalls
```



Page Replacement


- At some point in time, physical memory will become full


- At some point in time, physical memory will become full
- We need to make space available → throw out (= evict ) a page


- At some point in time, physical memory will become full
- We need to make space available → throw out (= evict ) a page
 - Unmodified code and data could be reloaded from binary


- At some point in time, physical memory will become full
- We need to make space available → throw out (= evict ) a page
 - Unmodified code and data could be reloaded from binary
 - What about other memory contents (modified from disk or generated)?

- At some point in time, physical memory will become full
- We need to make space available → throw out (= evict ) a page
 - Unmodified code and data could be reloaded from binary
 - What about other memory contents (modified from disk or generated)?
- When do we perform page replacement?

- At some point in time, physical memory will become full
- We need to make space available → throw out (= evict ) a page
 - Unmodified code and data could be reloaded from binary
 - What about other memory contents (modified from disk or generated)?
- When do we perform page replacement?

- At some point in time, physical memory will become full
- We need to make space available → throw out (= evict ) a page
 - Unmodified code and data could be reloaded from binary
 - What about other memory contents (modified from disk or generated)?
- When do we perform page replacement? For now:
 - When not a single page is available, and

- At some point in time, physical memory will become full
- We need to make space available → throw out (= evict ) a page
 - Unmodified code and data could be reloaded from binary
 - What about other memory contents (modified from disk or generated)?
- When do we perform page replacement? For now:
 - When not a single page is available, and
 - a thread T tries to allocate a page.

- At some point in time, physical memory will become full
- We need to make space available → throw out (= evict ) a page
 - Unmodified code and data could be reloaded from binary
 - What about other memory contents (modified from disk or generated)?
- When do we perform page replacement? For now:
 - When not a single page is available, and
 - a thread T tries to allocate a page.→ We evict a page, clear it, and return it (the now free page) directly to thread T .

What to do with modified pages?





Swap Out

Swap Out

- Reserve a special area on the disk

Swap Out

- Reserve a special area on the disk
 - swap file

Swap Out

- Reserve a special area on the disk
 - swap file
 - swap partition

Swap Out

- Reserve a special area on the disk
 - swap file
 - swap partition
 - swap disk

Swap Out

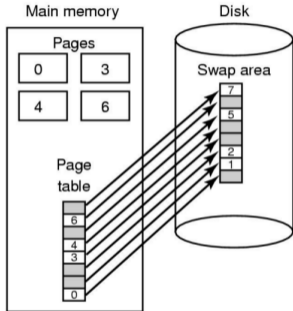
- Reserve a special area on the disk
 - swap file
 - swap partition
 - swap disk
- Write modified page there

Swap Out

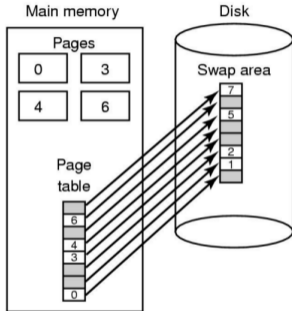
- Reserve a special area on the disk
 - swap file
 - swap partition
 - swap disk
- Write modified page there
- Evict it from RAM

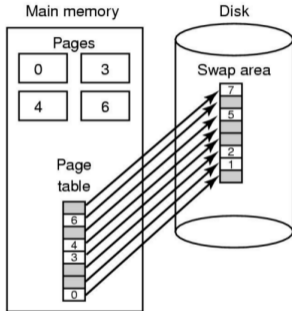


Swapping

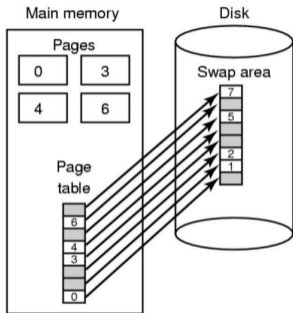


Swapping

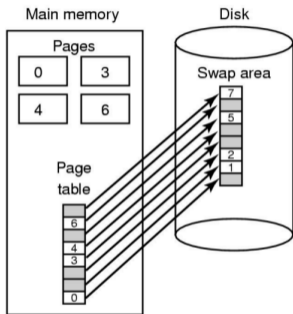




- static assignment

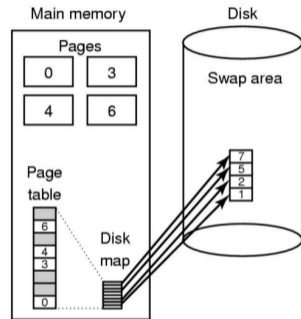
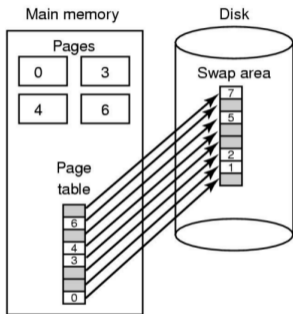


- static assignment
- low overhead

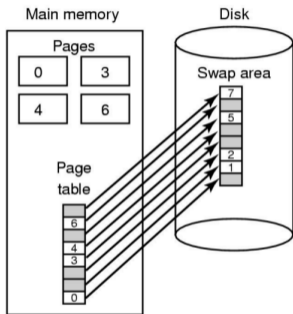


- static assignment
- low overhead
- not “on demand”: waste of disk space

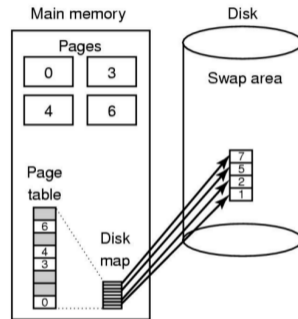
Swapping



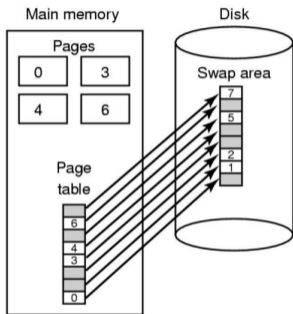
- static assignment
- low overhead
- not “on demand”: waste of disk space



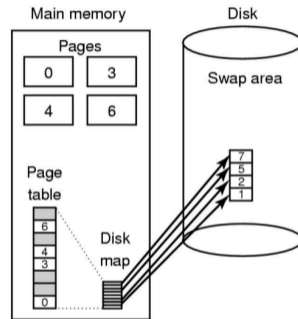
- static assignment
- low overhead
- not "on demand": waste of disk space



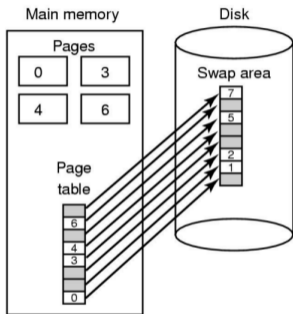
- dynamic assignment



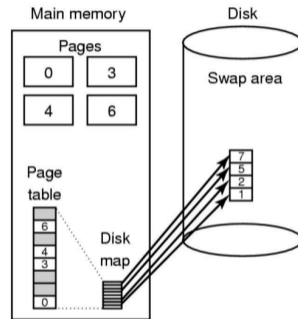
- static assignment
- low overhead
- not “on demand”: waste of disk space



- dynamic assignment
- larger overhead



- static assignment
- low overhead
- not "on demand": waste of disk space



- dynamic assignment
- larger overhead
- on demand: no wasted disk space

The Most Simple Page Replacement Algorithm (PRA): Random



- Simply evict a random page, any page.



- Simply evict a random page, any page.
- How good is Random PRA?



- Simply evict a random page, any page.
- How good is Random PRA?



- Simply evict a random page, any page.



- How good is Random PRA? It's a good start...
 - Often used as replacement algorithm in caches (ARM processors)

- Simply evict a random page, any page.



- How good is Random PRA? It's a good start...
 - Often used as replacement algorithm in caches (ARM processors)
- Source of randomness?

- Simply evict a random page, any page.



- How good is Random PRA? It's a good start...
 - Often used as replacement algorithm in caches (ARM processors)
- Source of randomness?

- Simply evict a random page, any page.



- How good is Random PRA? It's a good start...
 - Often used as replacement algorithm in caches (ARM processors)
- Source of randomness? Not that important, e.g., `rdtsc`





Pros:



Pros:

Pros:

- Very simple to implement in software or hardware

Pros:

- Very simple to implement in software or hardware
- No state, no precomputations, fast decision, tiny code base

Pros:

- Very simple to implement in software or hardware
- No state, no precomputations, fast decision, tiny code base
- May perform better than several more complex PRAs

Pros:

- Very simple to implement in software or hardware
- No state, no precomputations, fast decision, tiny code base
- May perform better than several more complex PRAs

Cons:

Pros:

- Very simple to implement in software or hardware
- No state, no precomputations, fast decision, tiny code base
- May perform better than several more complex PRAs

Cons:

Pros:

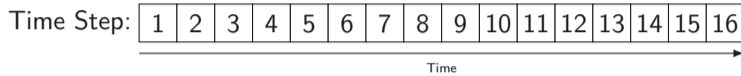
- Very simple to implement in software or hardware
- No state, no precomputations, fast decision, tiny code base
- May perform better than several more complex PRAs

Cons:

- PRA could use more information to not evict pages which are frequently used / required in the near future

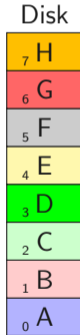
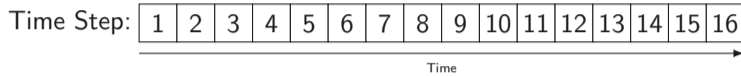
2 hits, 14 misses → rather bad





2 hits, 14 misses → rather bad





2 hits, 14 misses → rather bad

Time Step:

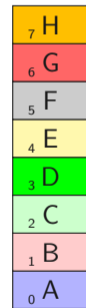
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time 

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Disk



2 hits, 14 misses → rather bad

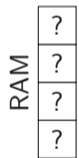
Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

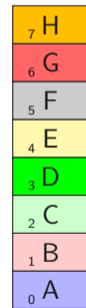
Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



Time

Disk



2 hits, 14 misses → rather bad

Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

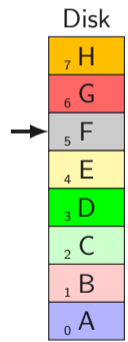
Time

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



Time



2 hits, 14 misses → rather bad

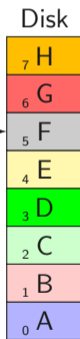
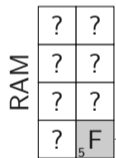
Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time →

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



Time →

2 hits, 14 misses → rather bad



Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time →

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

RAM

?	?	?
?	?	?
?	?	C ₂
?	F ₅	F ₅

Time →

Disk

7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A

2 hits, 14 misses → rather bad

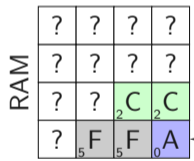
Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time →

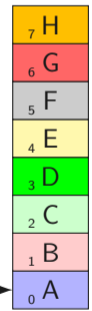
Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

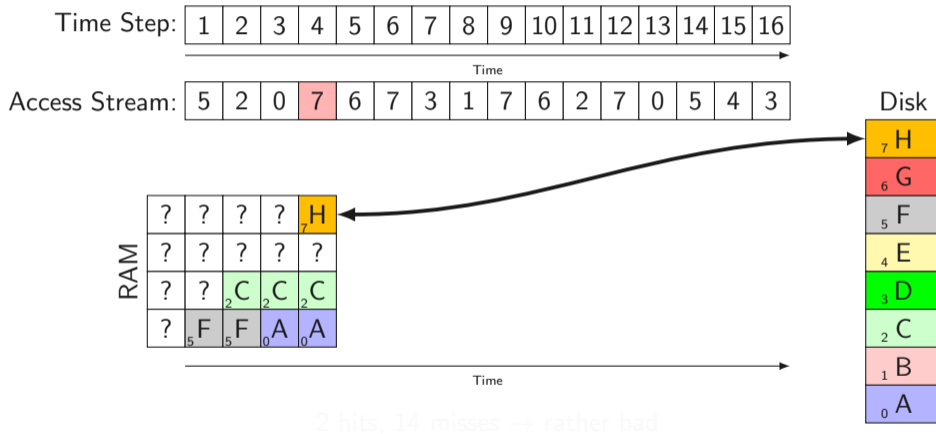


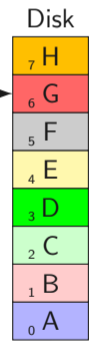
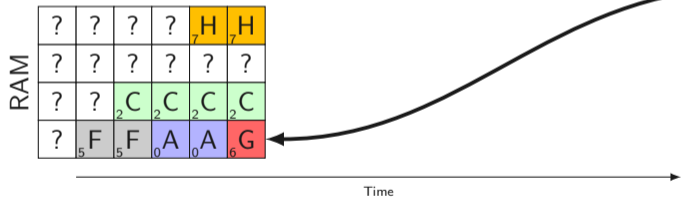
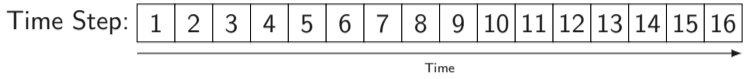
Time →

Disk



2 hits, 14 misses → rather bad





2 hits, 14 misses → rather bad



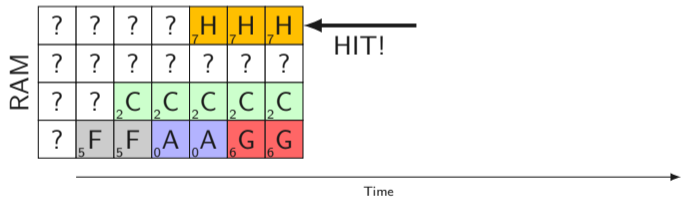
Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

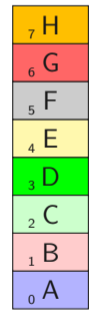
Time →

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



Disk



2 hits, 14 misses → rather bad

Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time →

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

RAM

?	?	?	?	H ₇	H ₇	H ₇	H ₇
?	?	?	?	?	?	?	?
?	?	C ₂	C ₂	C ₂	C ₂	C ₂	C ₂
?	F ₅	F ₅	A ₀	A ₀	G ₆	G ₆	D ₃

Time →

Disk

7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A

2 hits, 14 misses → rather bad

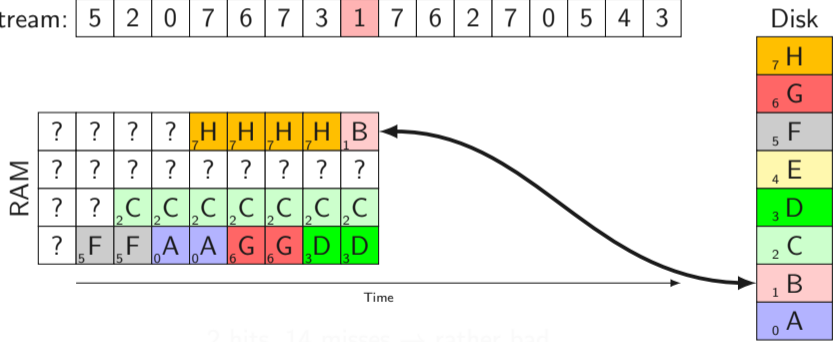
Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time →

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time →

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

RAM

?	?	?	?	H ₇	H ₇	H ₇	H ₇	B ₁	H ₇
?	?	?	?	?	?	?	?	?	?
?	?	C ₂	C ₂	C ₂	C ₂	C ₂	C ₂	C ₂	C ₂
?	F ₅	F ₅	A ₀	A ₀	G ₆	G ₆	D ₃	D ₃	D ₃

Time →

Disk

7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A

2 hits, 14 misses → rather bad

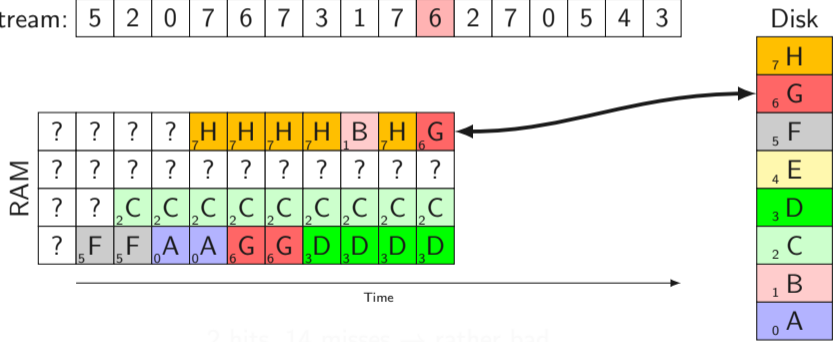
Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time →

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



2 hits, 14 misses → rather bad

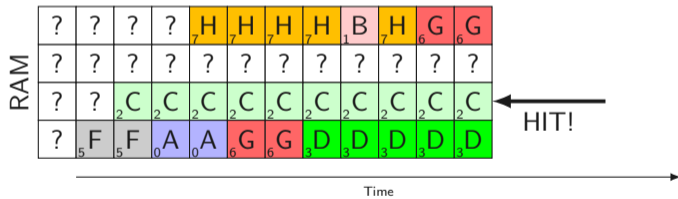
Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

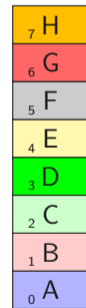
Time →

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



Disk



2 hits, 14 misses → rather bad

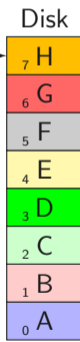
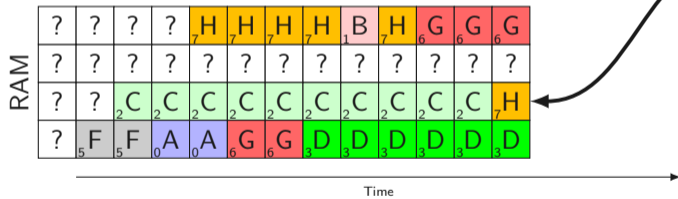
Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time →

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



2 hits, 14 misses → rather bad

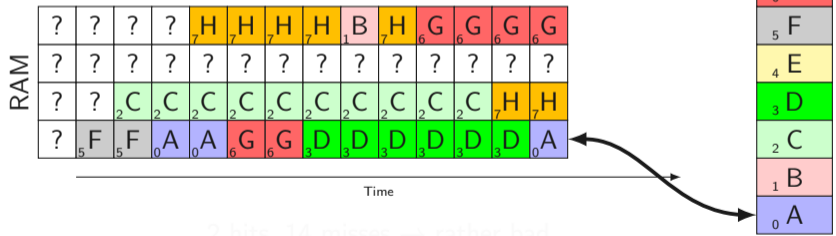
Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time →

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



Time Step:

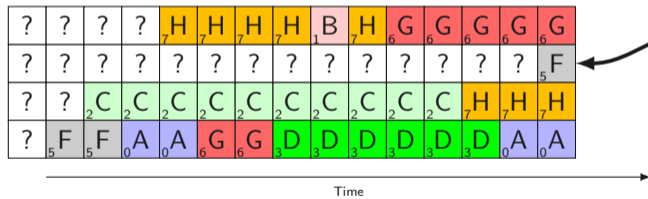
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time →

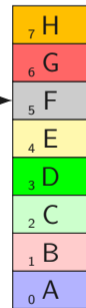
Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

RAM



Disk



2 hits, 14 misses → rather bad

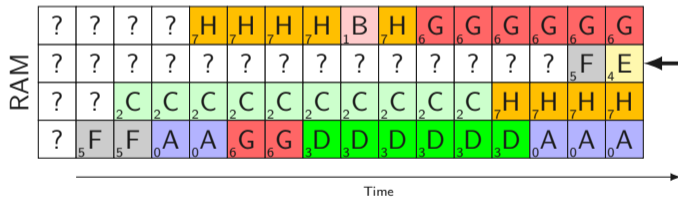
Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



2 hits, 14 misses → rather bad

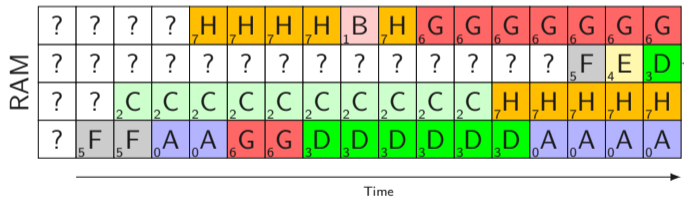
Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

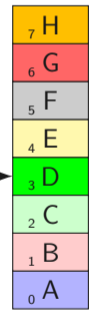
Time

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



Disk



2 hits, 14 misses → rather bad

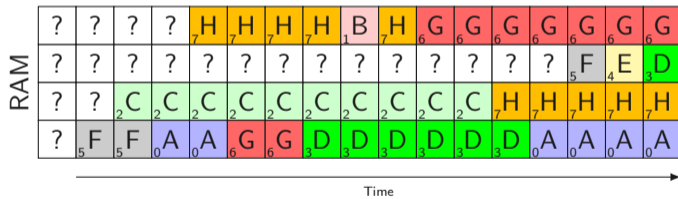
Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

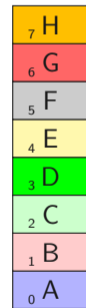
Time →

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



Disk



2 hits, 14 misses → rather bad

Let's assume, we can predict the future



Let's assume, we can predict the future



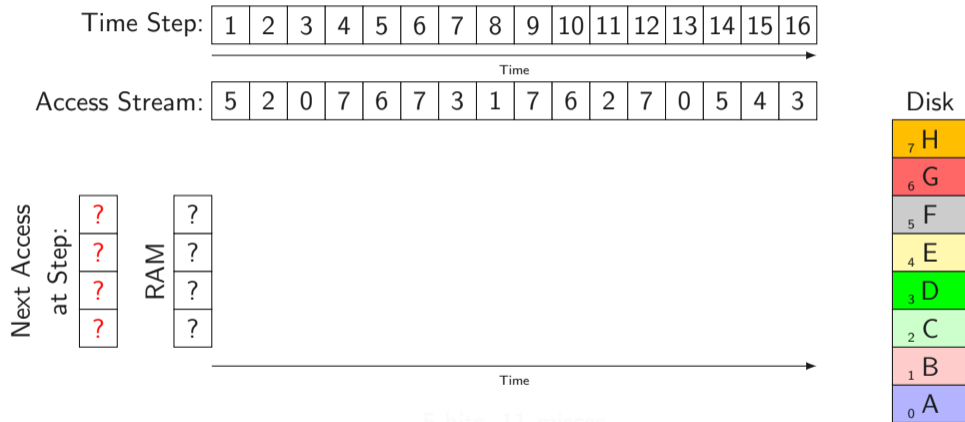
1. Store number of steps until next access (per page)

Let's assume, we can predict the future

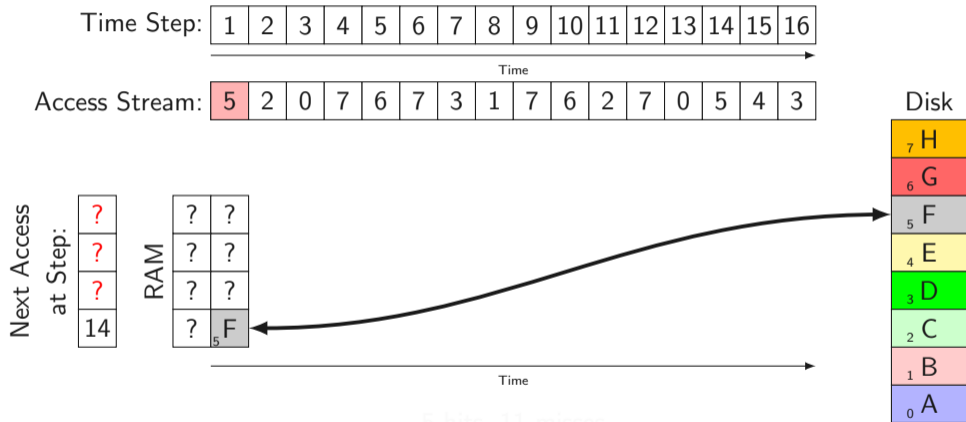


1. Store number of steps until next access (per page)
2. Remove page with largest number

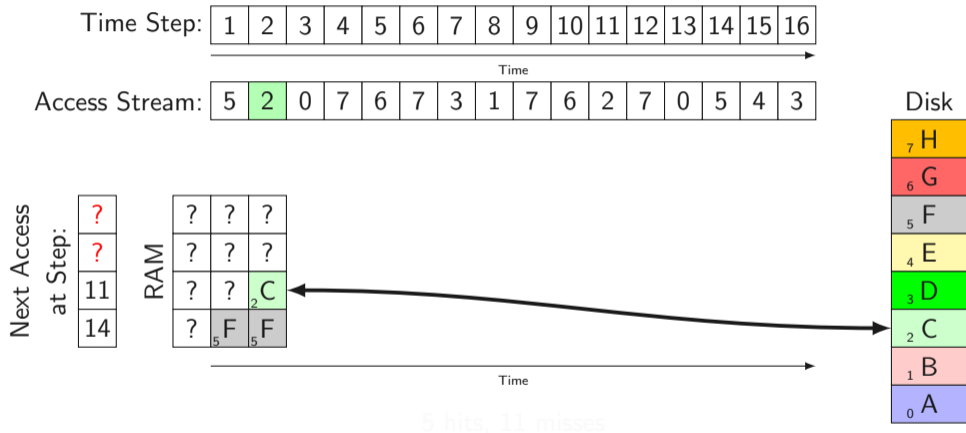
Optimal PRA



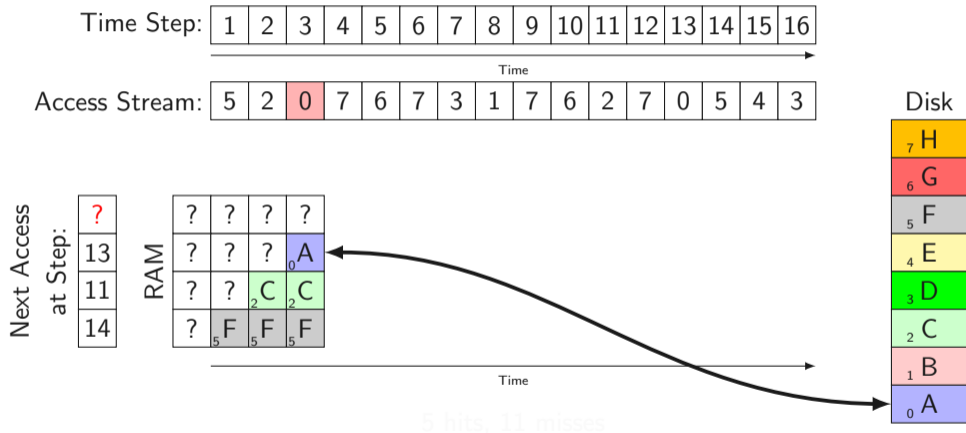
Optimal PRA



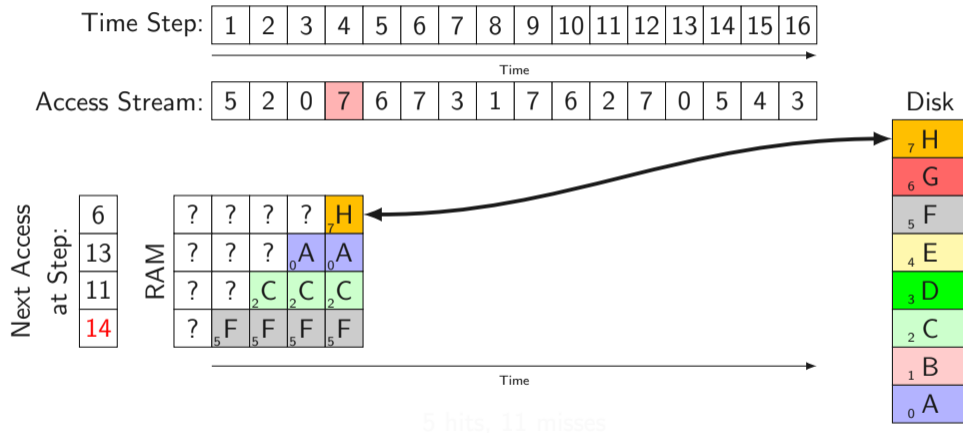
Optimal PRA



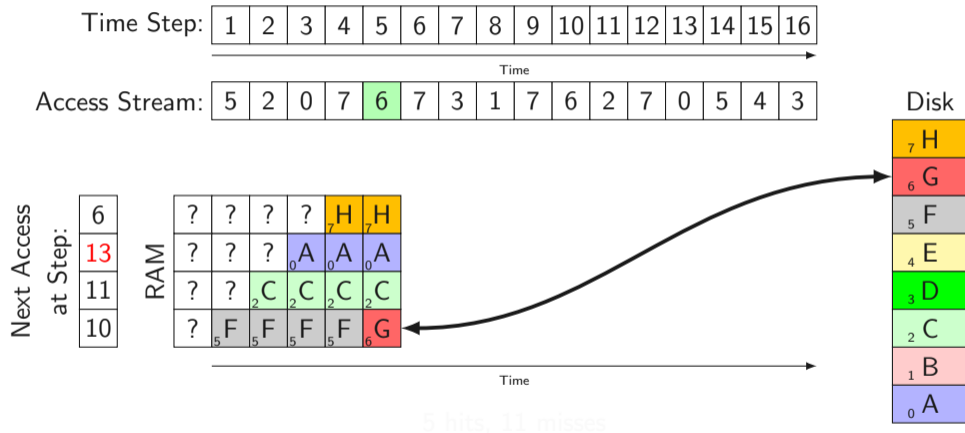
Optimal PRA



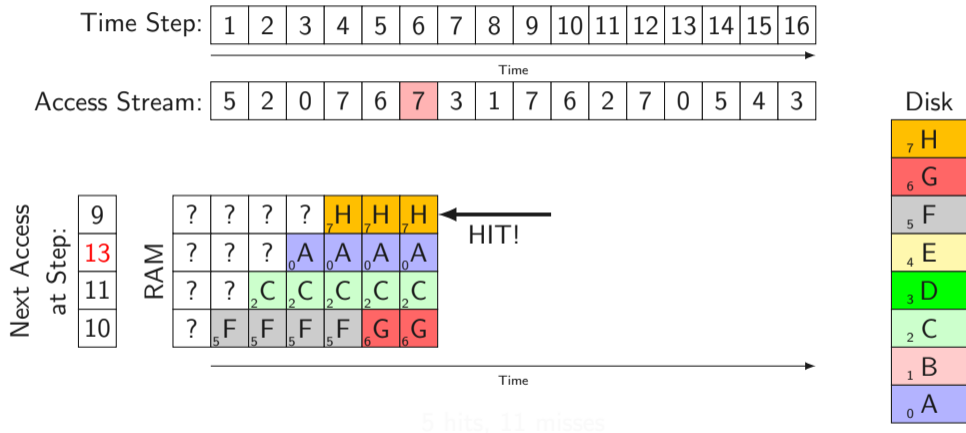
Optimal PRA



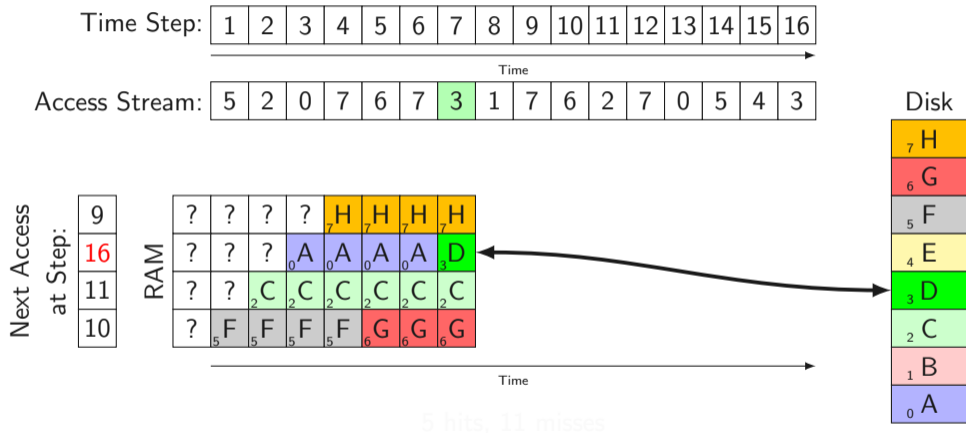
Optimal PRA



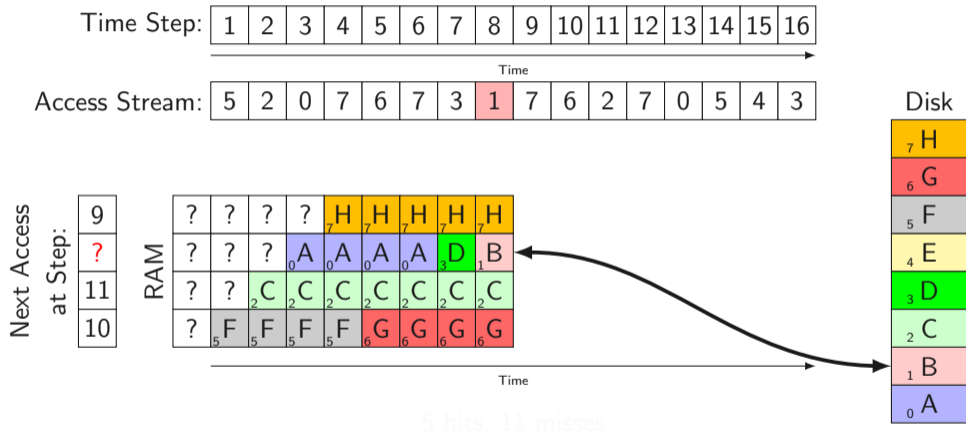
Optimal PRA



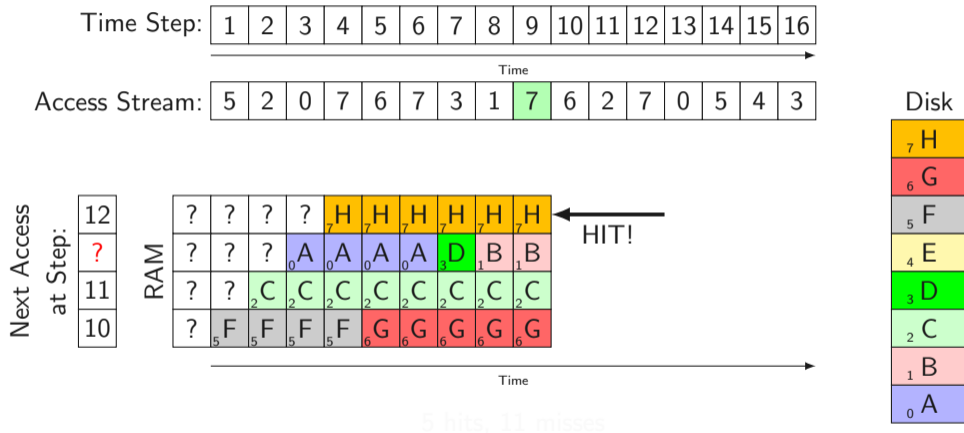
Optimal PRA



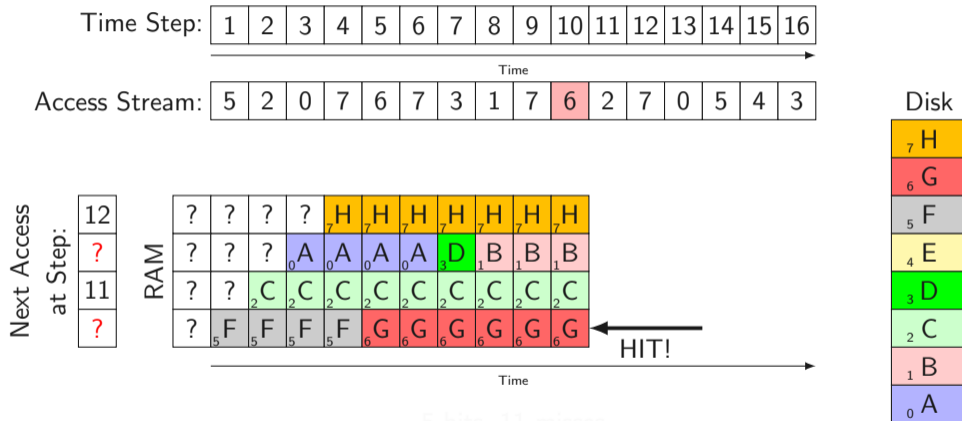
Optimal PRA



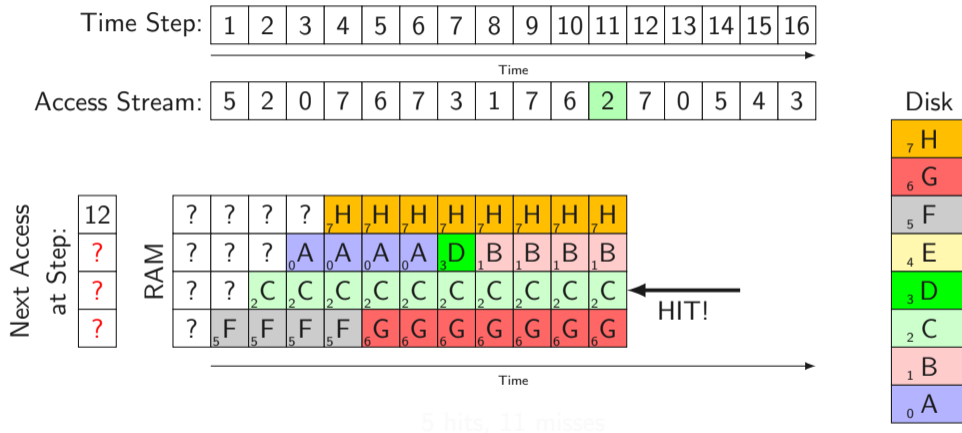
Optimal PRA



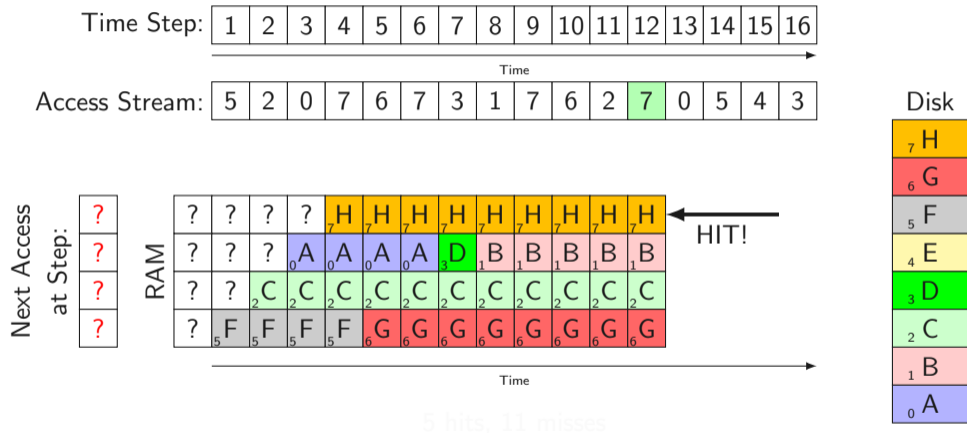
Optimal PRA



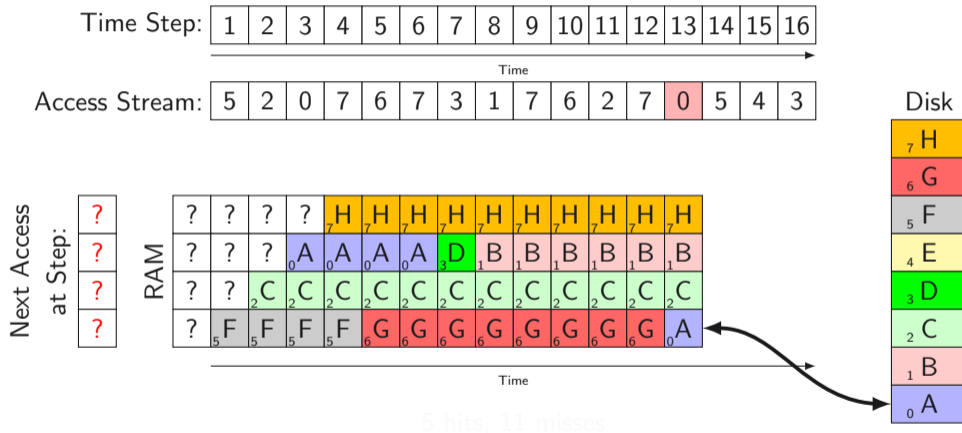
Optimal PRA



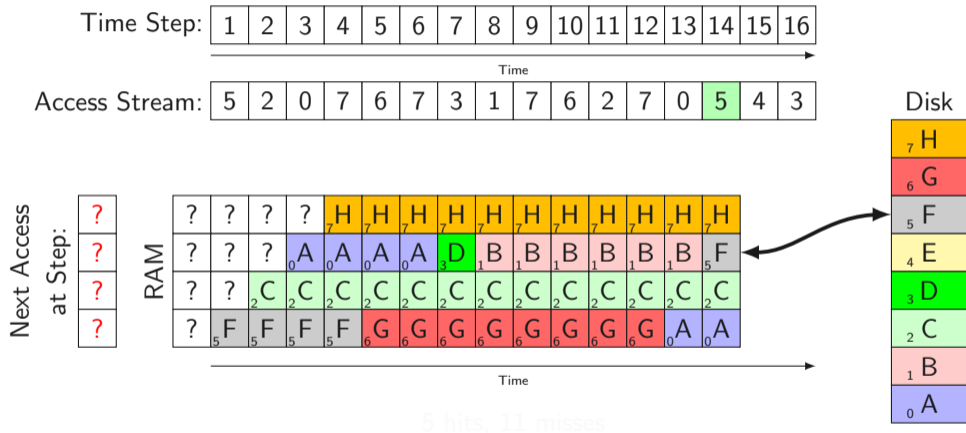
Optimal PRA



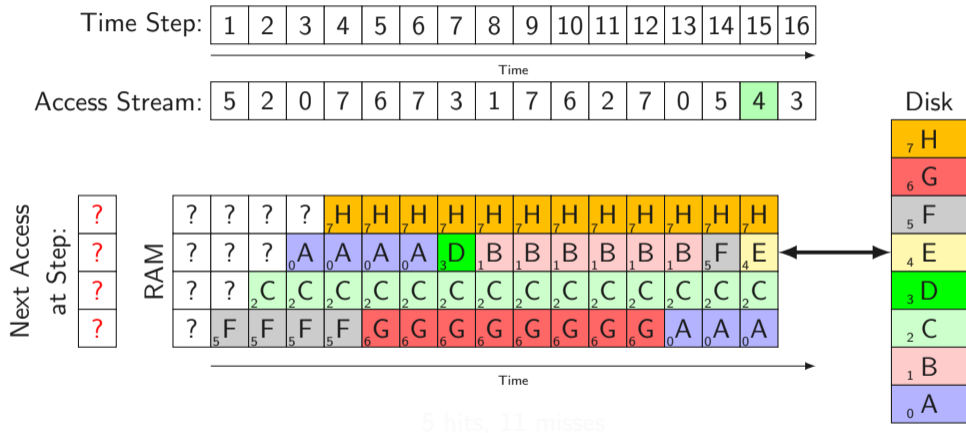
Optimal PRA



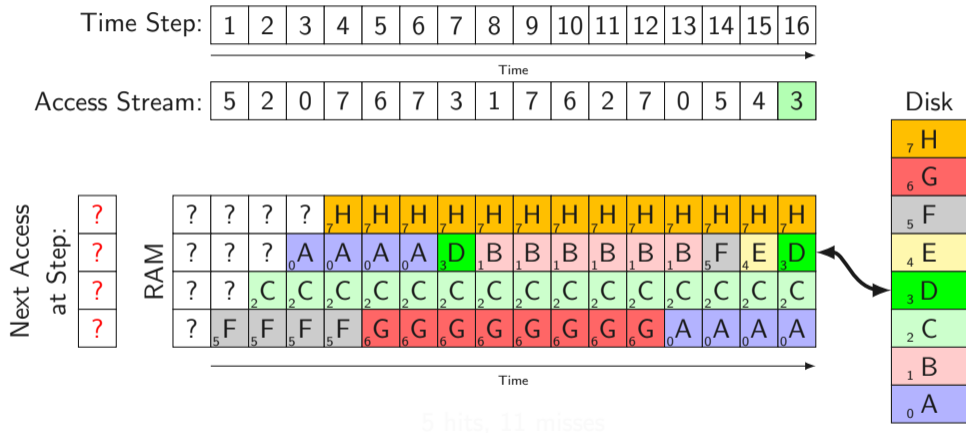
Optimal PRA



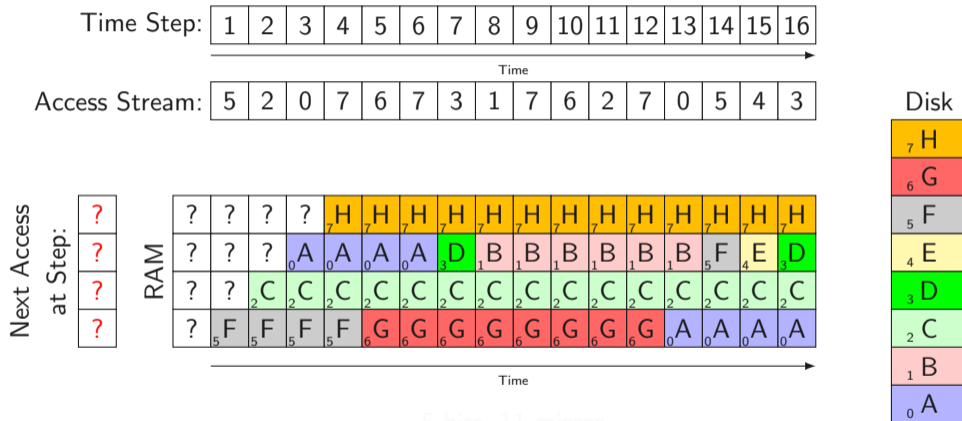
Optimal PRA



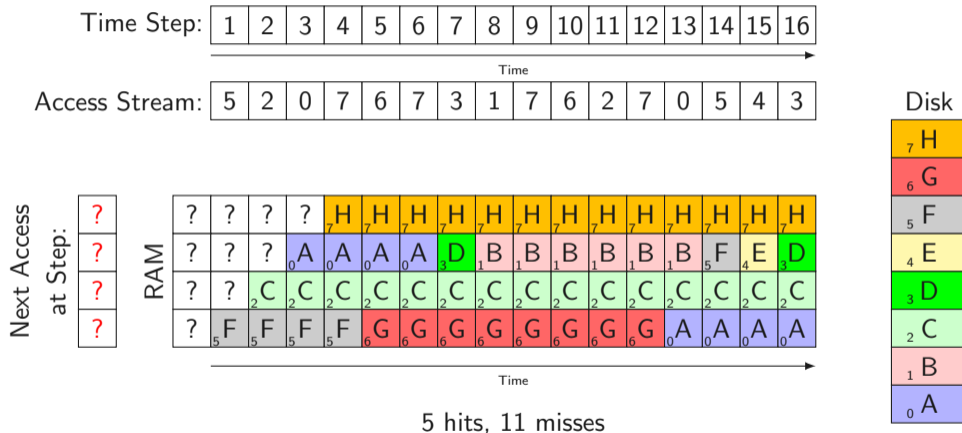
Optimal PRA






Optimal PRA







Optimal PRA







- We can't look into the future... 

- We can't look into the future... 
- Principle of locality 

- We can't look into the future... 
- Principle of locality 
 - future memory access might be near past memory accesses

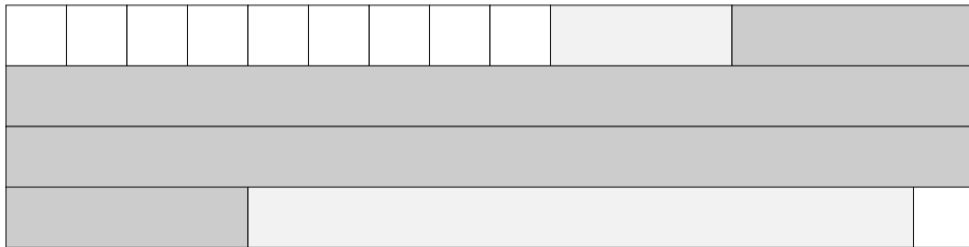
- We can't look into the future... 
- Principle of locality 
 - future memory access might be near past memory accesses
 - design idea of virtually all sophisticated PRAs

- We can't look into the future... 
- Principle of locality 
 - future memory access might be near past memory accesses
 - design idea of virtually all sophisticated PRAs

- We can't look into the future... 
- Principle of locality 
 - future memory access might be near past memory accesses
 - design idea of virtually all sophisticated PRAs

→ How do we learn past memory accesses?

How to detect past memory read and write accesses?





How to detect past memory read and write accesses?



How to detect past memory read and write accesses?



How to detect past memory read and write accesses?



How to detect past memory read and write accesses?

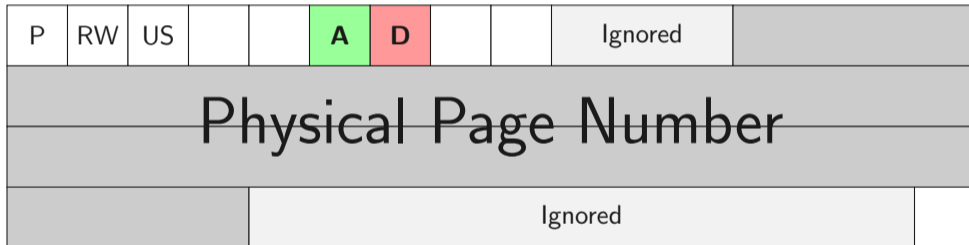


How to detect past memory read and write accesses?



P	RW	US			A				Ignored	
Physical Page Number										
				Ignored						

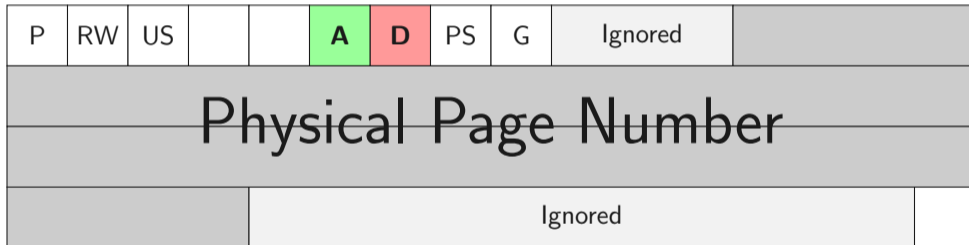
How to detect past memory read and write accesses?



How to detect past memory read and write accesses?



How to detect past memory read and write accesses?



How to detect past memory read and write accesses?



P	RW	US			A	D	PS	G	Ignored	
Physical Page Number										
					Ignored					X

How to detect past memory read and write accesses?



P	RW	US	WT		A	D	PS	G	Ignored	
Physical Page Number										
				Ignored						X

How to detect past memory read and write accesses?

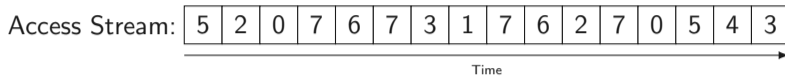


P	RW	US	WT	PCD	A	D	PS	G	Ignored	
Physical Page Number										
				Ignored						X

Problem: 1 bit of information is not a detailed trace of past memory accesses

How do we get the information we need?

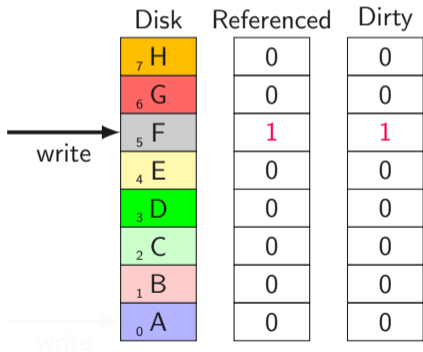
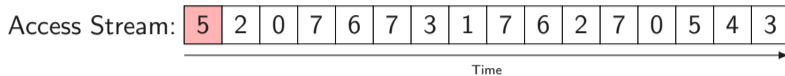
Detecting reads and writes



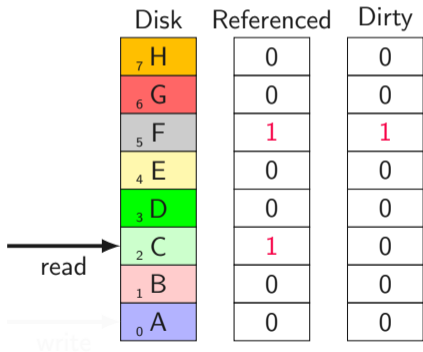
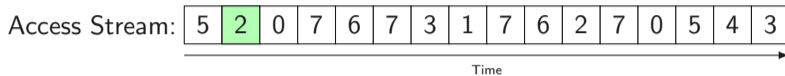
Disk	Referenced	Dirty				
<table border="1"><tr><td>7</td><td>H</td></tr></table>	7	H	<table border="1"><tr><td>0</td></tr></table>	0	<table border="1"><tr><td>0</td></tr></table>	0
7	H					
0						
0						
<table border="1"><tr><td>6</td><td>G</td></tr></table>	6	G	<table border="1"><tr><td>0</td></tr></table>	0	<table border="1"><tr><td>0</td></tr></table>	0
6	G					
0						
0						
<table border="1"><tr><td>5</td><td>F</td></tr></table>	5	F	<table border="1"><tr><td>0</td></tr></table>	0	<table border="1"><tr><td>0</td></tr></table>	0
5	F					
0						
0						
<table border="1"><tr><td>4</td><td>E</td></tr></table>	4	E	<table border="1"><tr><td>0</td></tr></table>	0	<table border="1"><tr><td>0</td></tr></table>	0
4	E					
0						
0						
<table border="1"><tr><td>3</td><td>D</td></tr></table>	3	D	<table border="1"><tr><td>0</td></tr></table>	0	<table border="1"><tr><td>0</td></tr></table>	0
3	D					
0						
0						
<table border="1"><tr><td>2</td><td>C</td></tr></table>	2	C	<table border="1"><tr><td>0</td></tr></table>	0	<table border="1"><tr><td>0</td></tr></table>	0
2	C					
0						
0						
<table border="1"><tr><td>1</td><td>B</td></tr></table>	1	B	<table border="1"><tr><td>0</td></tr></table>	0	<table border="1"><tr><td>0</td></tr></table>	0
1	B					
0						
0						
<table border="1"><tr><td>0</td><td>A</td></tr></table>	0	A	<table border="1"><tr><td>0</td></tr></table>	0	<table border="1"><tr><td>0</td></tr></table>	0
0	A					
0						
0						

write →

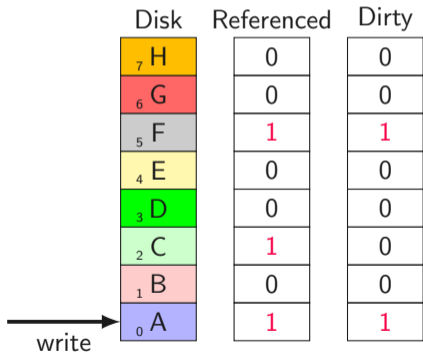
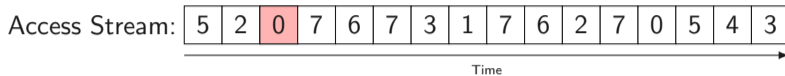
Detecting reads and writes



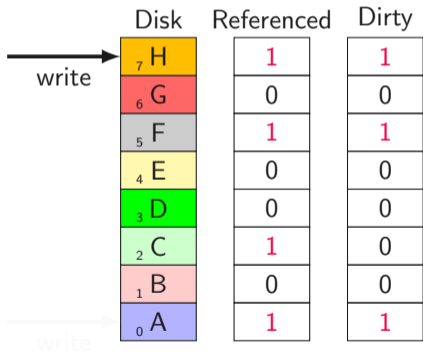
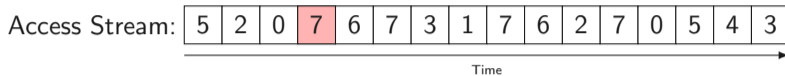
Detecting reads and writes



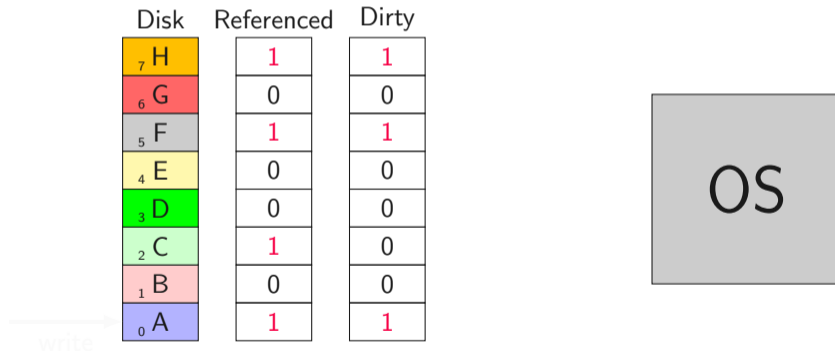
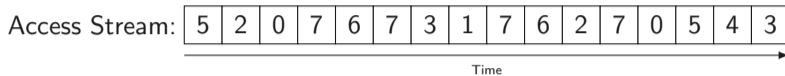
Detecting reads and writes



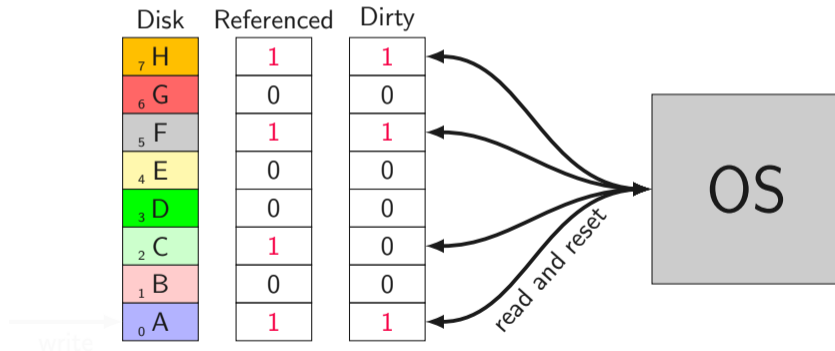
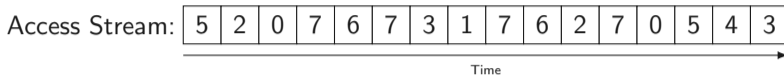
Detecting reads and writes



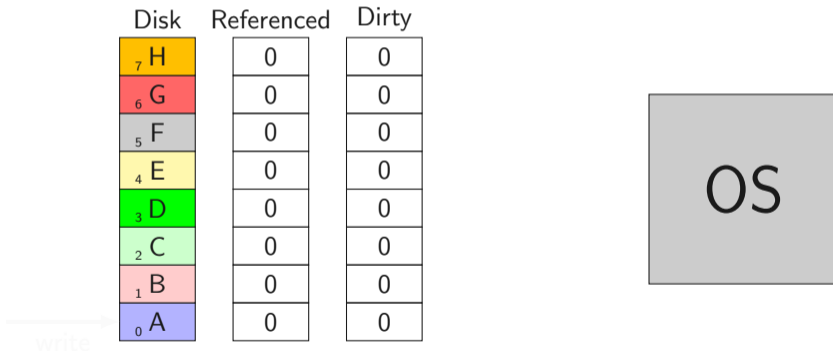
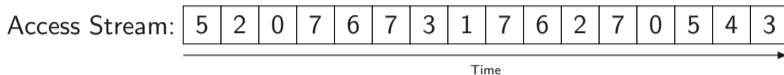
Detecting reads and writes



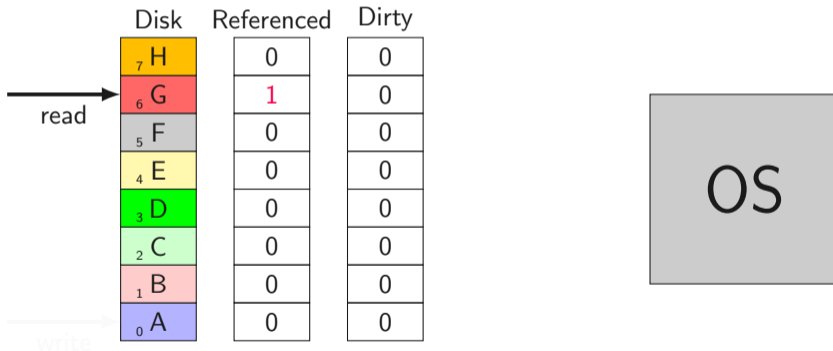
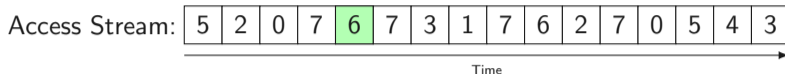
Detecting reads and writes



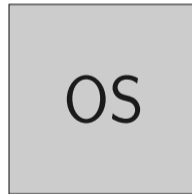
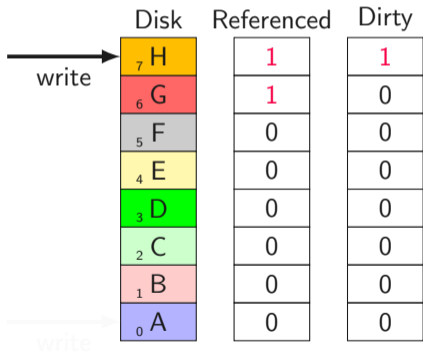
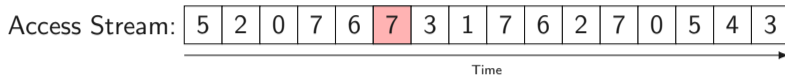
Detecting reads and writes



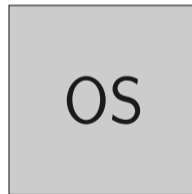
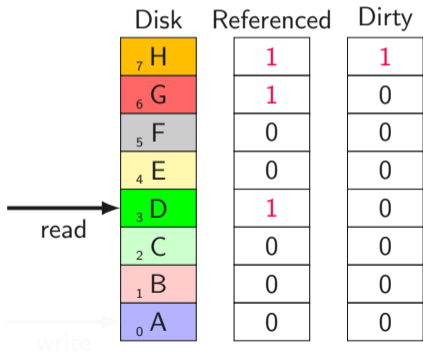
Detecting reads and writes



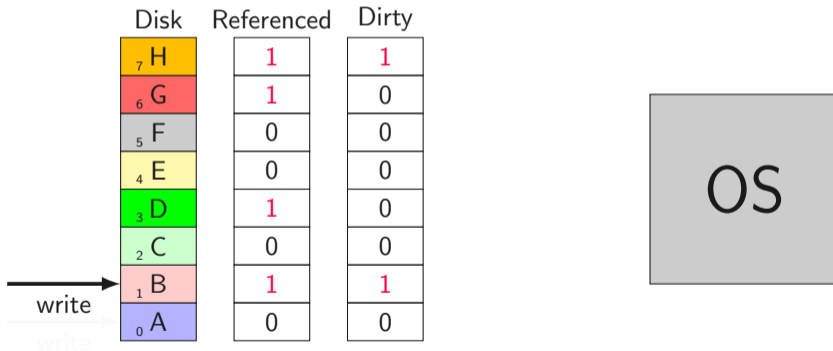
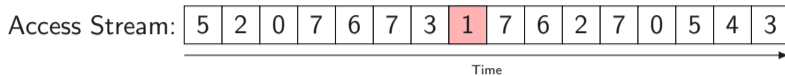
Detecting reads and writes



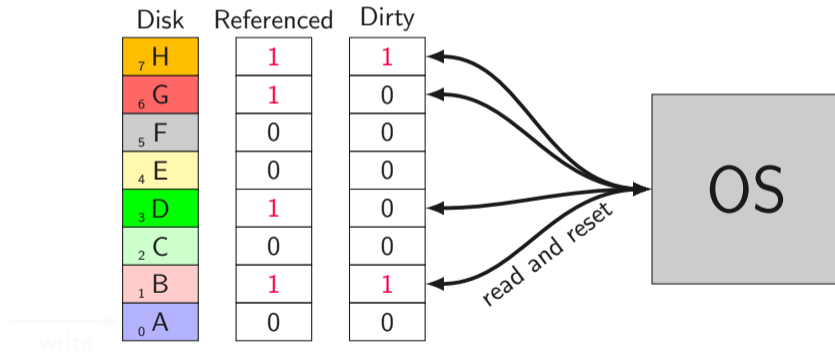
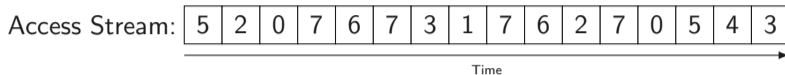
Detecting reads and writes



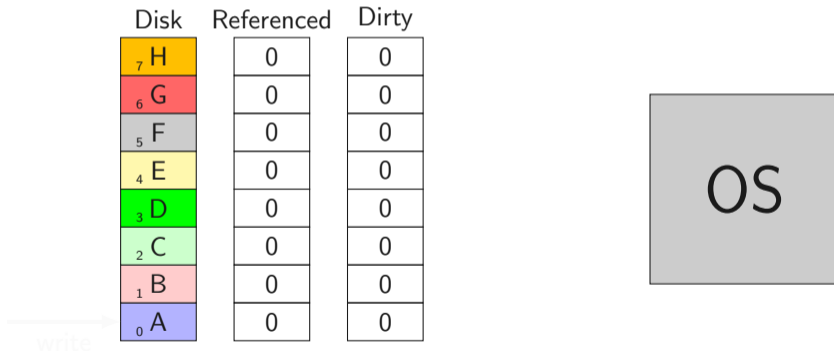
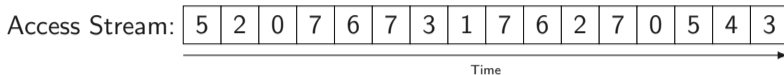
Detecting reads and writes



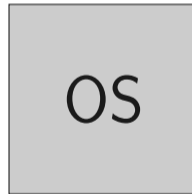
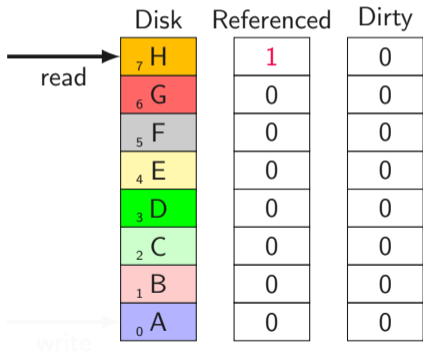
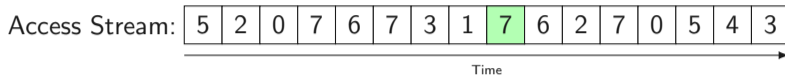
Detecting reads and writes



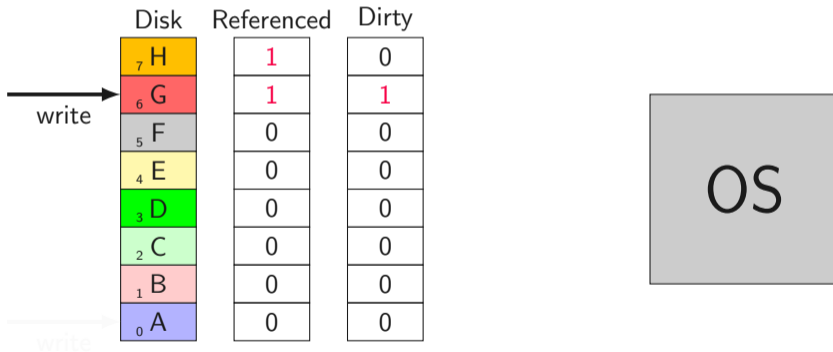
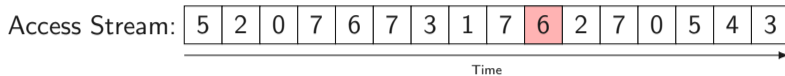
Detecting reads and writes



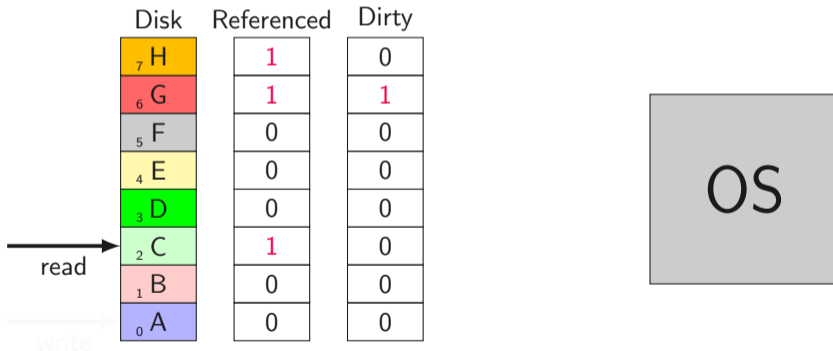
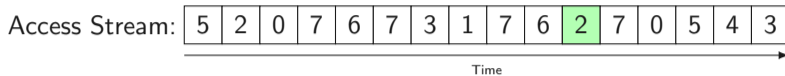
Detecting reads and writes



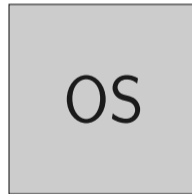
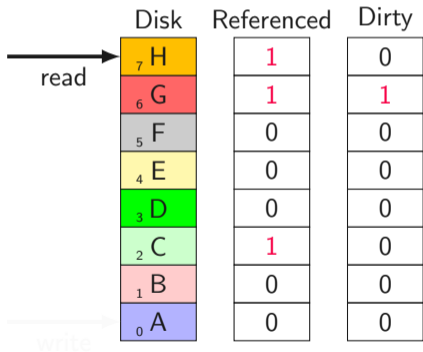
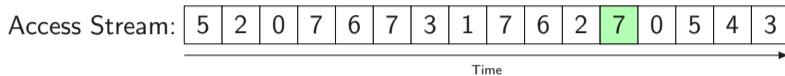
Detecting reads and writes



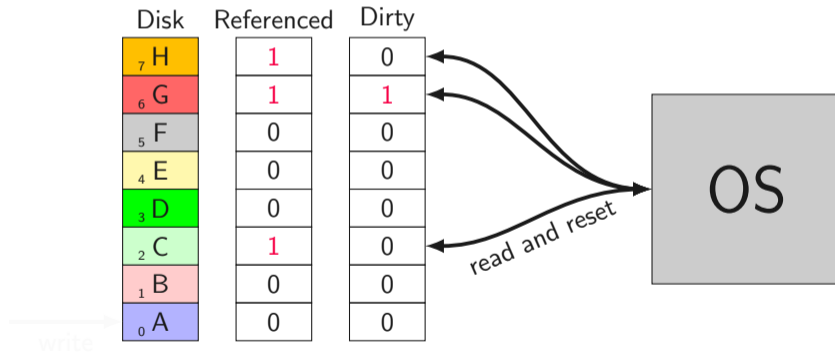
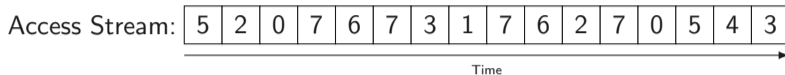
Detecting reads and writes



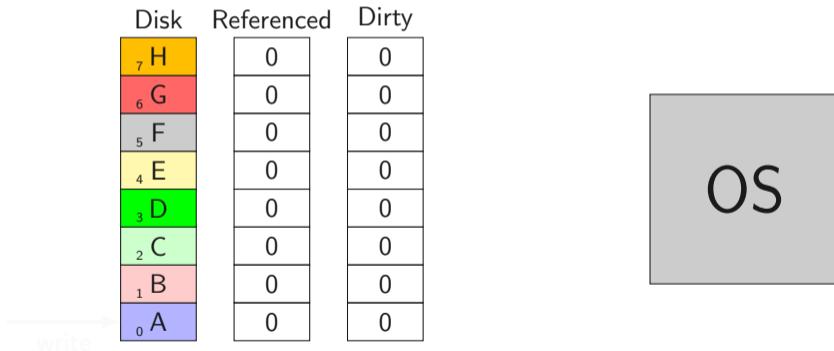
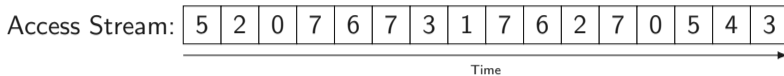
Detecting reads and writes



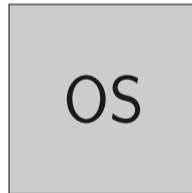
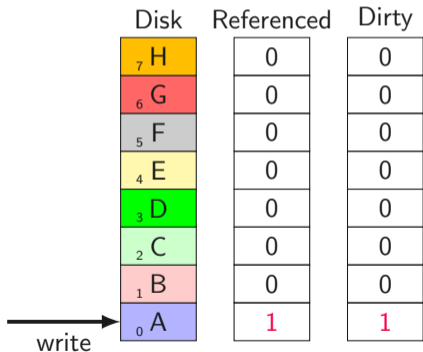
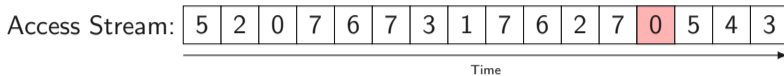
Detecting reads and writes



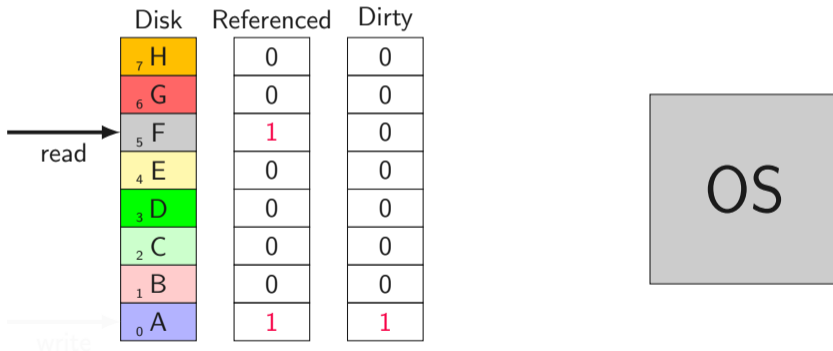
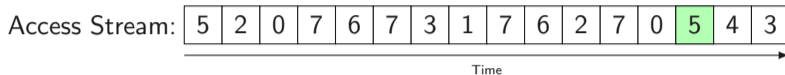
Detecting reads and writes



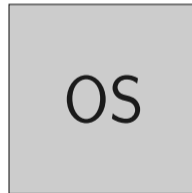
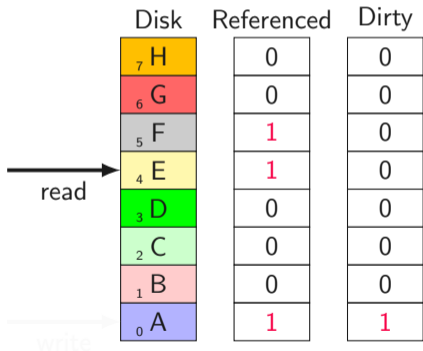
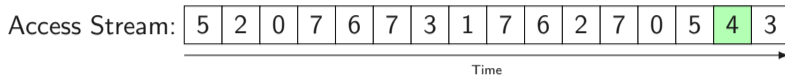
Detecting reads and writes



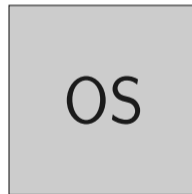
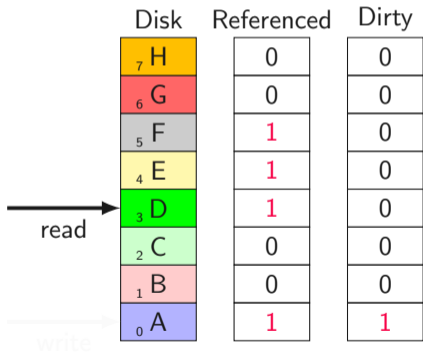
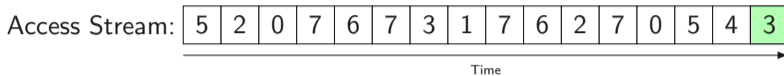
Detecting reads and writes



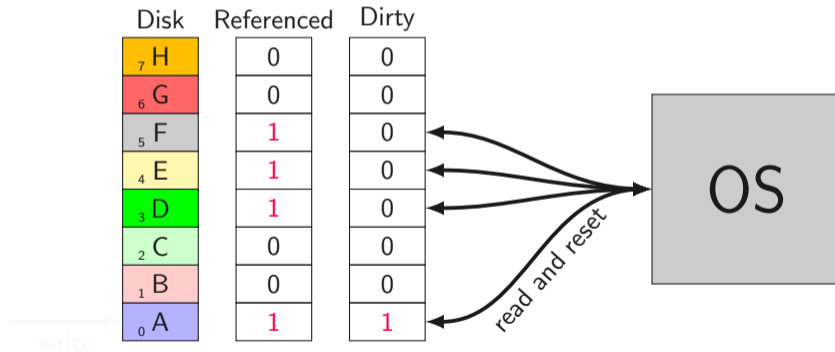
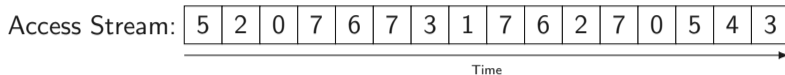
Detecting reads and writes



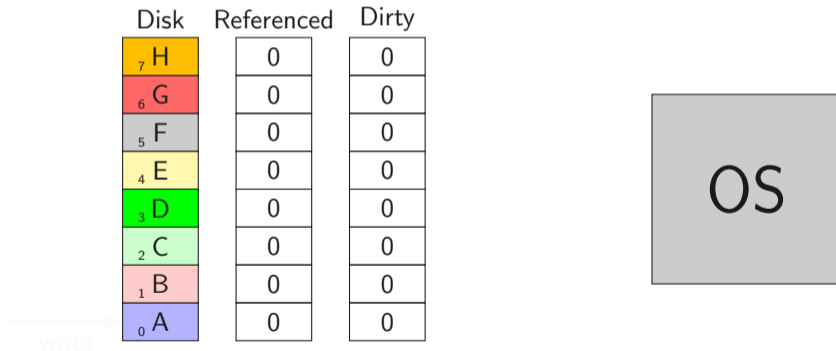
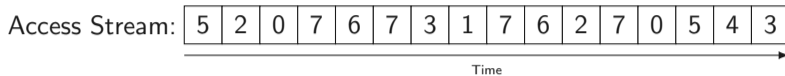
Detecting reads and writes



Detecting reads and writes



Detecting reads and writes



Which is the best class to choose pages from for replacement?

Class	Referenced	Dirty	Properties
0	0	0	
1	0	1	
2	1	0	
3	1	1	

Dirty = it's not stored identically on the disk

Which is the best class to choose pages from for replacement?

Class	Referenced	Dirty	Properties
0	0	0	Not used in a while and not modified → just evict
1	0	1	
2	1	0	
3	1	1	

Dirty = it's not stored identically on the disk

Which is the best class to choose pages from for replacement?

Class	Referenced	Dirty	Properties
0	0	0	Not used in a while and not modified → just evict
1	0	1	Not used in a while but modified → write back, then evict
2	1	0	
3	1	1	

Dirty = it's not stored identically on the disk

Which is the best class to choose pages from for replacement?

Class	Referenced	Dirty	Properties
0	0	0	Not used in a while and not modified → just evict
1	0	1	Not used in a while but modified → write back, then evict
2	1	0	Recently used but not modified → prefer eviction of other pages
3	1	1	

Dirty = it's not stored identically on the disk

Which is the best class to choose pages from for replacement?

Class	Referenced	Dirty	Properties
0	0	0	Not used in a while and not modified → just evict
1	0	1	Not used in a while but modified → write back, then evict
2	1	0	Recently used but not modified → prefer eviction of other pages
3	1	1	Recently used and modified → only evict as a last resort

Dirty = it's not stored identically on the disk



- Basically: Random PRA with classes (0-3)



- Basically: Random PRA with classes (0-3)
- Performs better than Random PRA



- Basically: Random PRA with classes (0-3)
- Performs better than Random PRA
- Design Decision: How far does “recently” go?

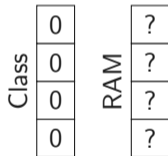
Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

Access Stream:

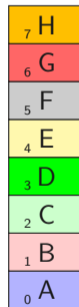
5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

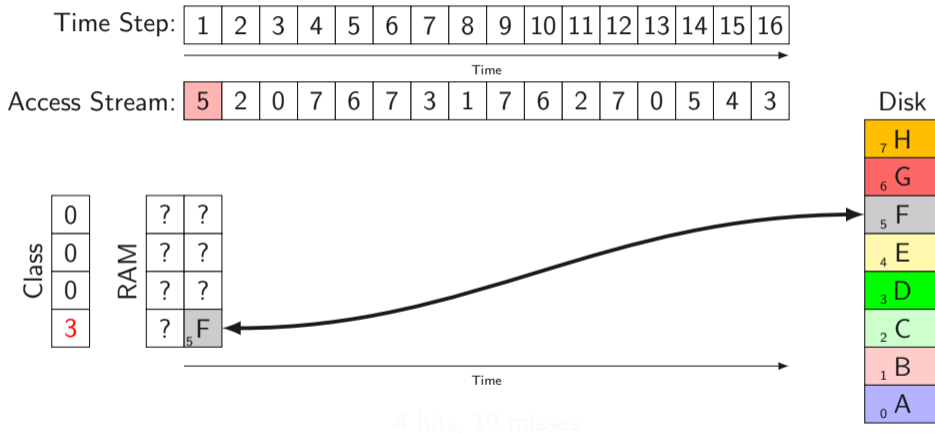


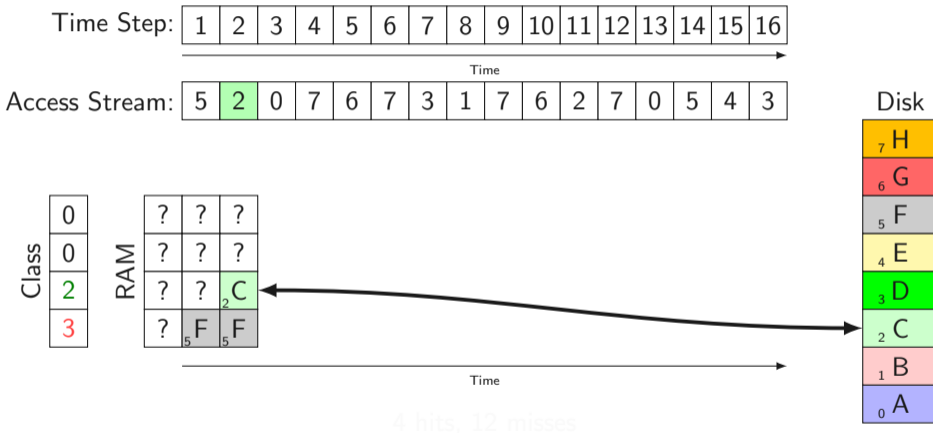
Time

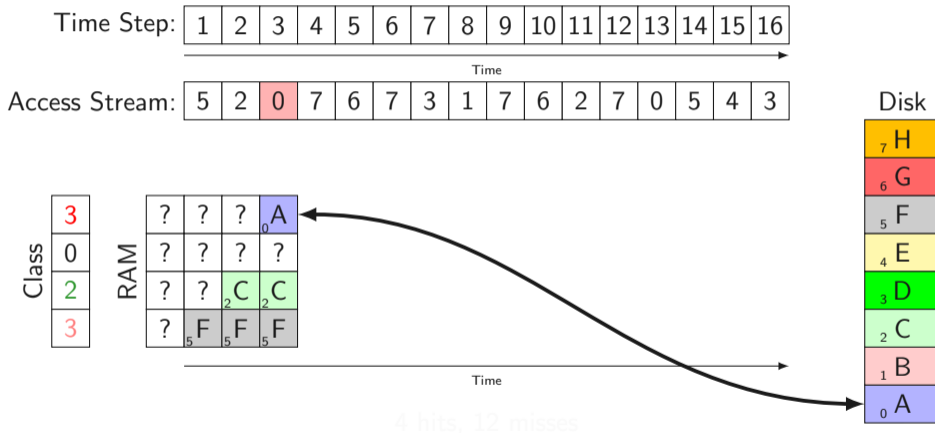
4 hits, 12 misses

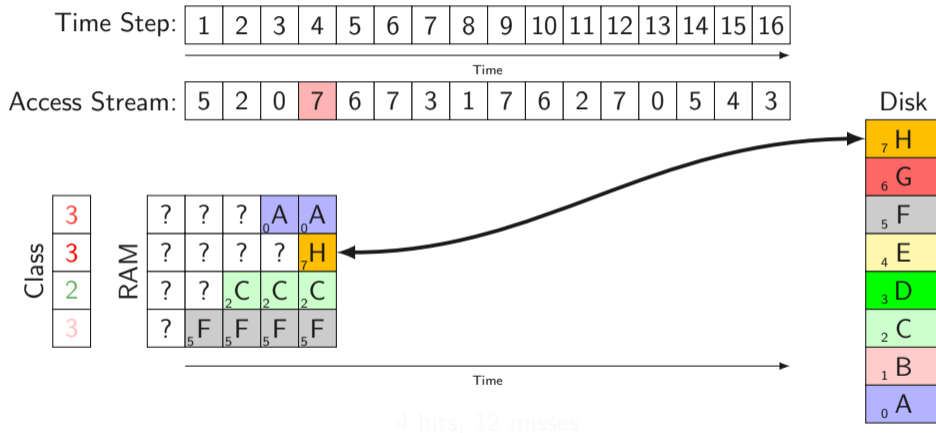
Disk

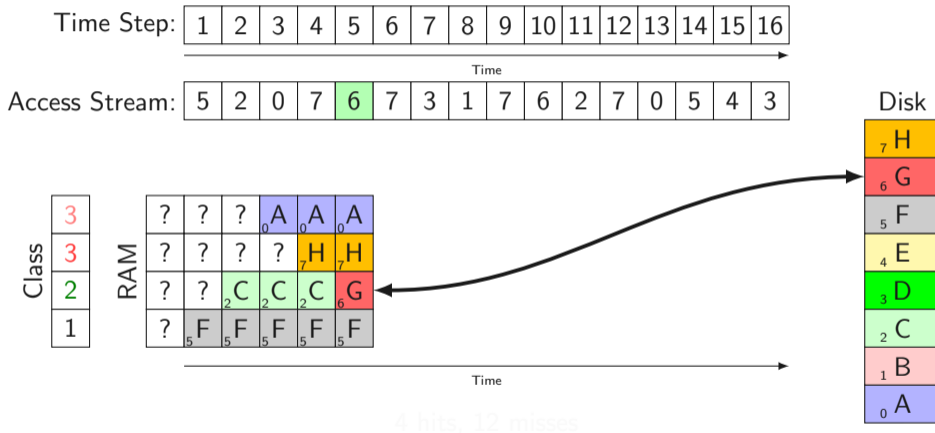












Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Class

3
3
2
1

RAM

?	?	?	A ₀	A ₀	A ₀	A ₀
?	?	?	?	H ₇	H ₇	H ₇
?	?	C ₂	C ₂	C ₂	G ₆	G ₆
?	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅

HIT!

Time

Disk

7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A

4 hits, 12 misses



Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

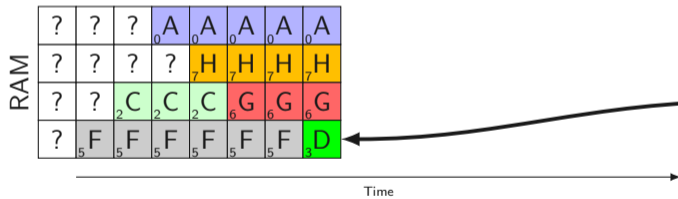
Time →

Access Stream:

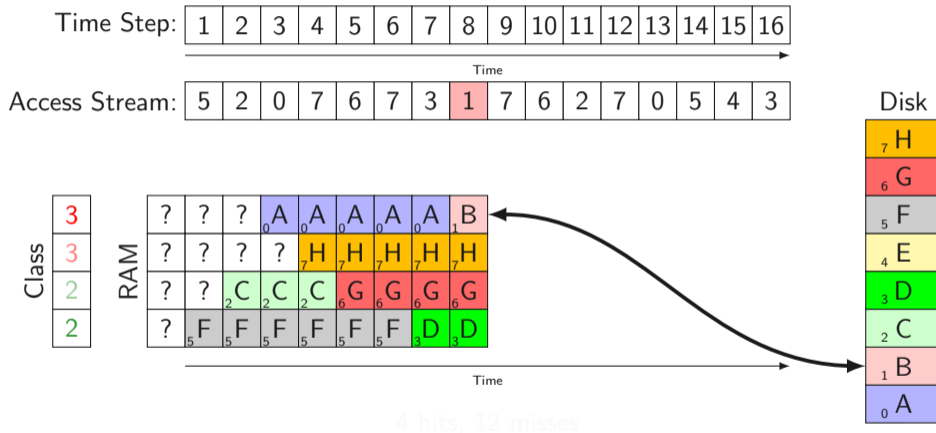
5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Class

1
3
2
2



4 hits, 12 misses





Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

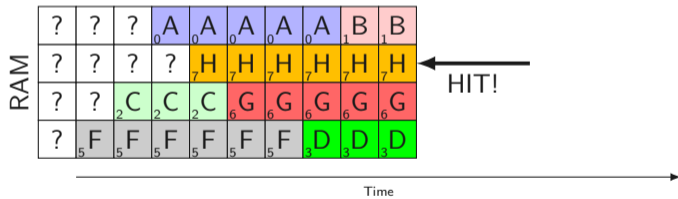
Time →

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Class

3
3
0
2



HIT!

4 hits, 12 misses





Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time →

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Class

3
3
3
2

RAM

?	?	?	A ₀	A ₀	A ₀	A ₀	A ₀	B ₁	B ₁	B ₁
?	?	?	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇
?	?	C ₂	C ₂	C ₂	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆
?	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	D ₃	D ₃	D ₃	D ₃

HIT!

Time →

4 hits, 12 misses

Disk

7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A



Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time →

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Class

3
3
3
2

RAM

?	?	?	A ₀	A ₀	A ₀	A ₀	A ₀	B ₁	B ₁	B ₁	B ₁
?	?	?	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇
?	?	C ₂	C ₂	C ₂	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆
?	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	D ₃	D ₃	D ₃	D ₃	C ₂

Time →

Disk

7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A

4 hits, 12 misses



Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

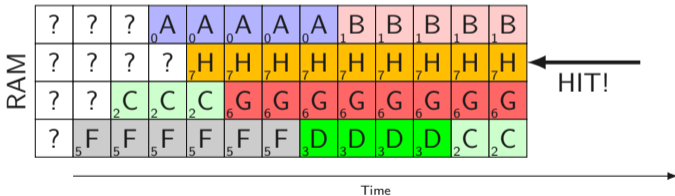
Time →

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Class

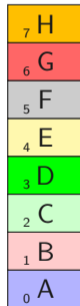
1
3
3
2

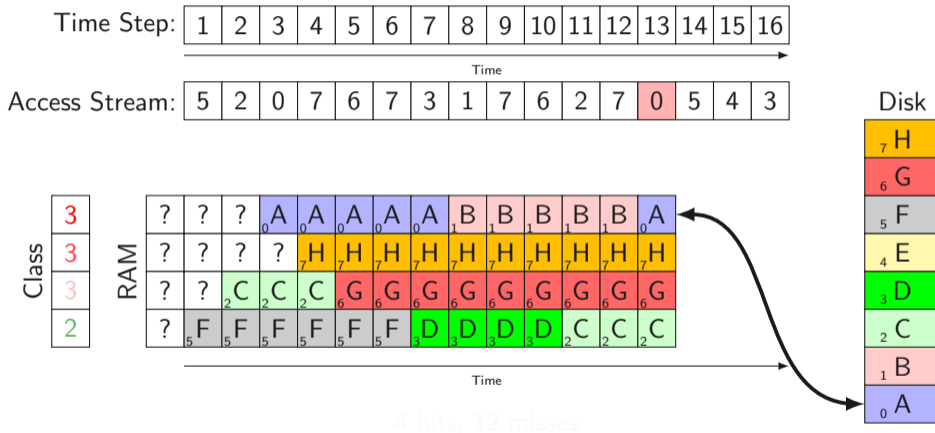


HIT!

4 hits, 12 misses

Disk





Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Class

3
3
1
2

RAM

?	?	?	A ₀	A ₀	A ₀	A ₀	A ₀	A ₁	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁	A ₀	A ₀	
?	?	?	?	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	
?	?	C ₂	C ₂	C ₂	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	
?	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	D ₃	D ₃	D ₃	D ₃	D ₃	D ₃	C ₂	C ₂	C ₂	F ₅

Time

Disk

7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A

4 hits, 12 misses

Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Class

3
3
2
2

RAM

?	?	?	A ₀	A ₀	A ₀	A ₀	A ₀	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁	A ₀	A ₀	A ₀
?	?	?	?	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇
?	?	C ₂	C ₂	C ₂	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	E ₄
?	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	D ₃	D ₃	D ₃	D ₃	D ₃	D ₃	C ₂	C ₂	C ₂

Time

Disk

7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A

4 hits, 12 misses

Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Class

3
1
2
2

RAM

?	?	?	A ₀	A ₀	A ₀	A ₀	A ₀	B ₁	B ₁	B ₁	B ₁	B ₁	A ₀	A ₀	A ₀	A ₀	
?	?	?	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	
?	?	C ₂	C ₂	C ₂	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	E ₄	E ₄	
?	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	D ₃	D ₃	D ₃	D ₃	D ₃	C ₂	C ₂	C ₂	F ₅	F ₅	D ₃

Time

4 hits, 12 misses

Disk

7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A



Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time →

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Class

3
1
2
2

RAM

?	?	?	A ₀	A ₀	A ₀	A ₀	A ₀	B ₁	B ₁	B ₁	B ₁	B ₁	A ₀	A ₀	A ₀	A ₀
?	?	?	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇
?	?	C ₂	C ₂	C ₂	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	E ₄	E ₄
?	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	D ₃	D ₃	D ₃	D ₃	D ₃	C ₂	C ₂	C ₂	F ₅

Time →

4 hits, 12 misses

Disk

H ₇
G ₆
F ₅
E ₄
D ₃
C ₂
B ₁
A ₀



- Queue/List of all pages (e.g. `std::queue`)



- Queue/List of all pages (e.g. `std::queue`)
- Load a page: `push_back`



- Queue/List of all pages (e.g. `std::queue`)
- Load a page: `push_back`
- Page to replace: `pop_front`




- Queue/List of all pages (e.g. `std::queue`)
- Load a page: `push_back`
- Page to replace: `pop_front`
- Very simple algorithm




- Queue/List of all pages (e.g. `std::queue`)
- Load a page: `push_back`
- Page to replace: `pop_front`
- Very simple algorithm
- Rarely used in practice

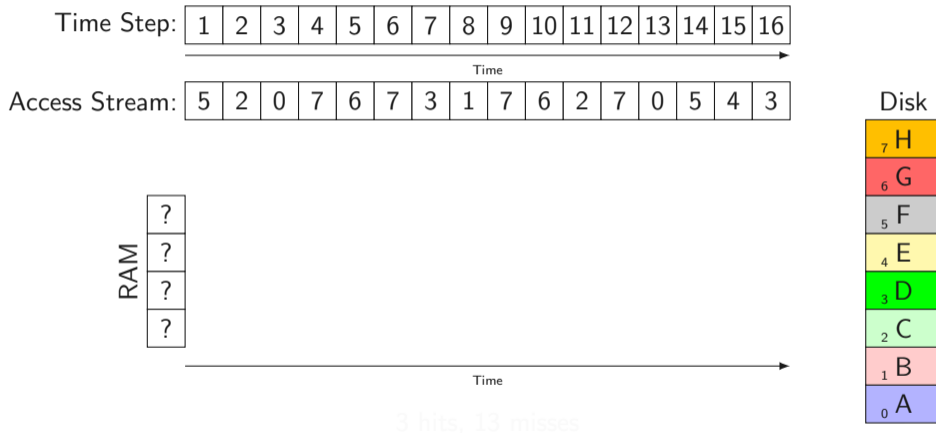


- Queue/List of all pages (e.g. `std::queue`)
- Load a page: `push_back`
- Page to replace: `pop_front`
- Very simple algorithm
- Rarely used in practice
- Performance can even be worse than Random PRA(!) 

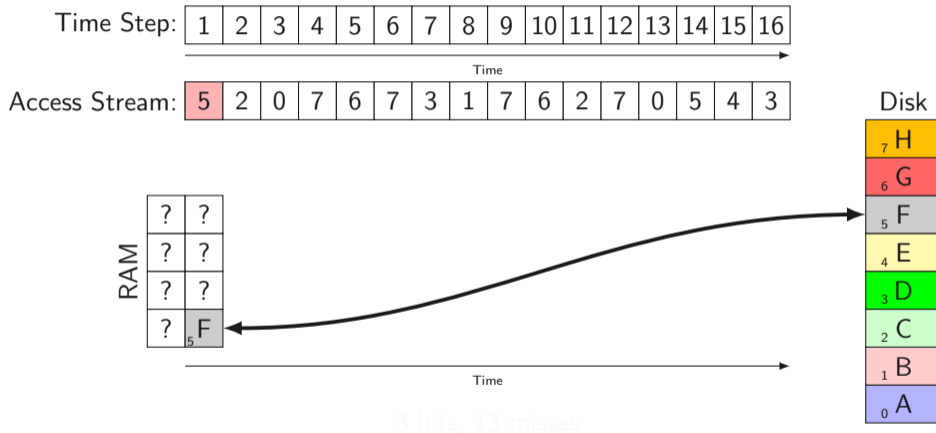


- Queue/List of all pages (e.g. `std::queue`)
- Load a page: `push_back`
- Page to replace: `pop_front`
- Very simple algorithm
- Rarely used in practice
- Performance can even be worse than Random PRA(!) 
- + FIFO anomaly / Belady's anomaly: increasing memory size can reduce performance

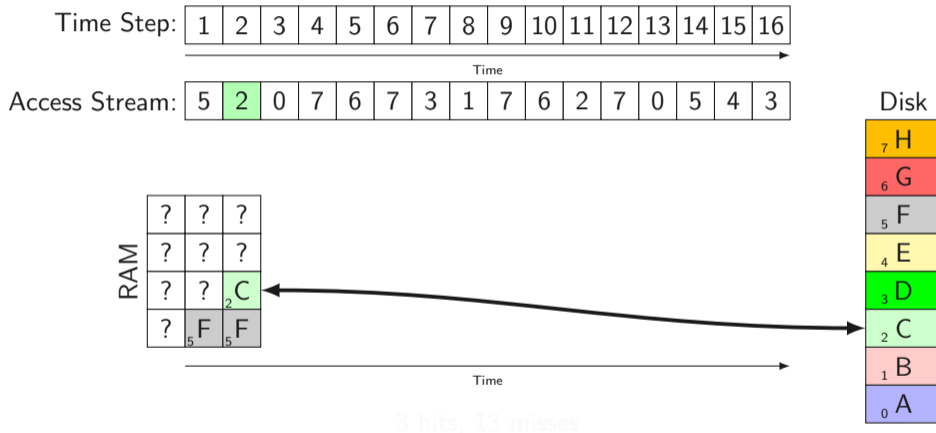
FIFO PRA



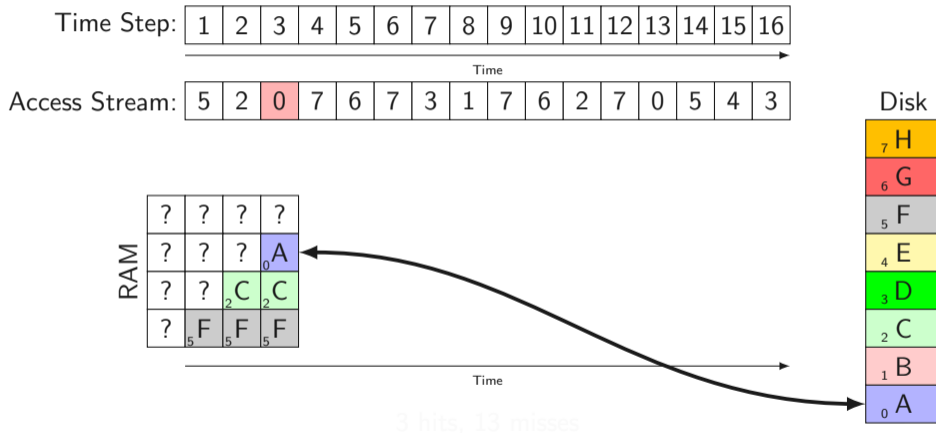
FIFO PRA



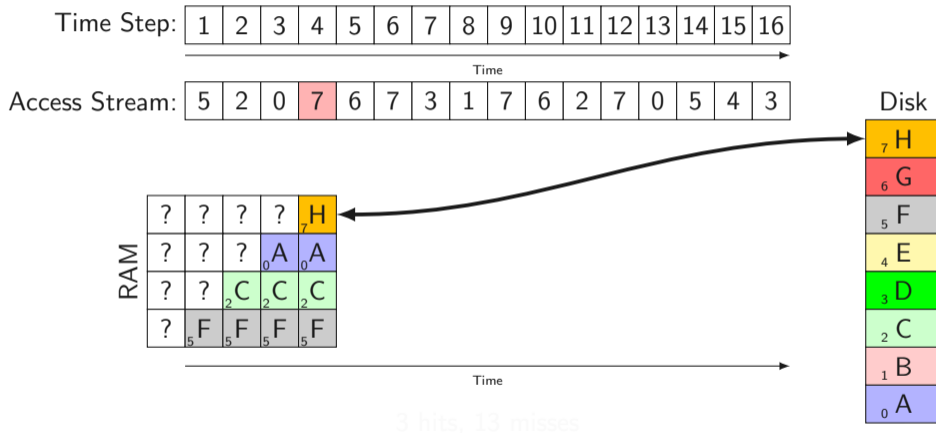
FIFO PRA



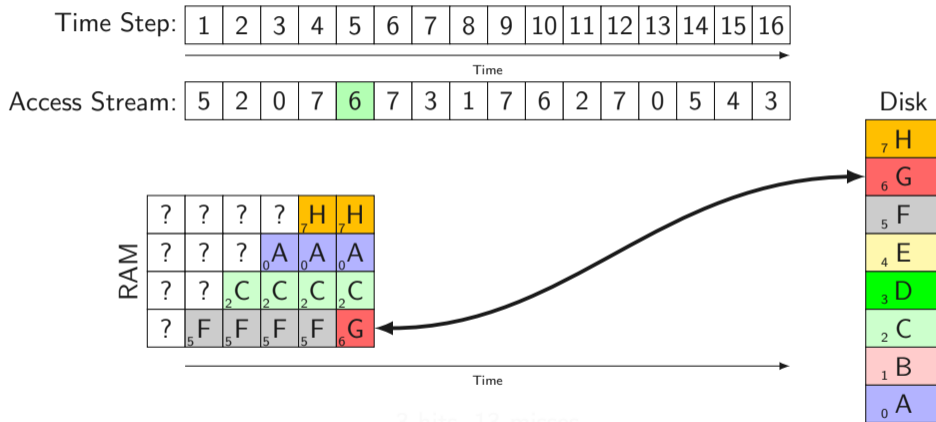
FIFO PRA



FIFO PRA



FIFO PRA

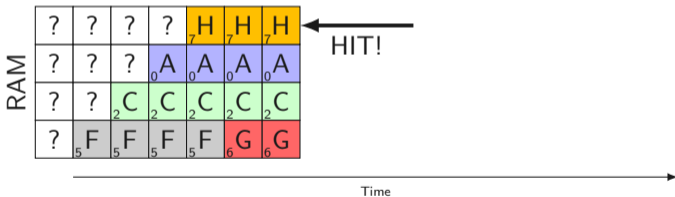


FIFO PRA

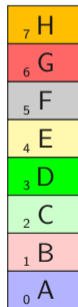
Time Step: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Time

Access Stream: 5 2 0 7 6 7 3 1 7 6 2 7 0 5 4 3

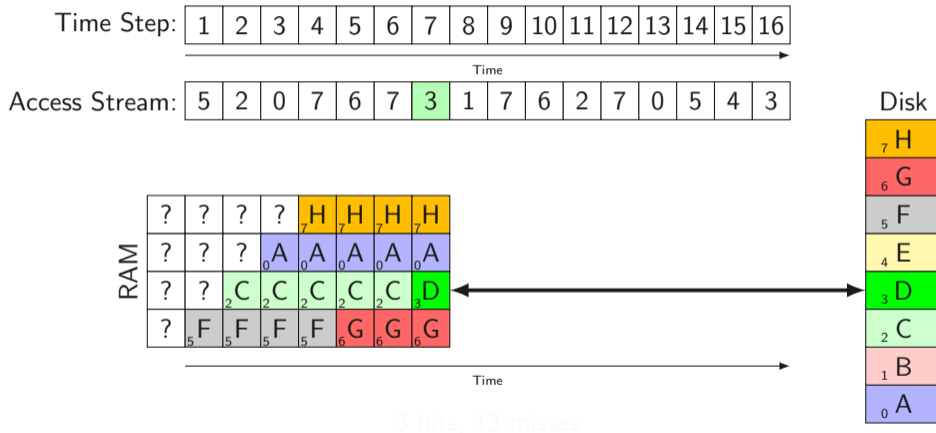


Disk

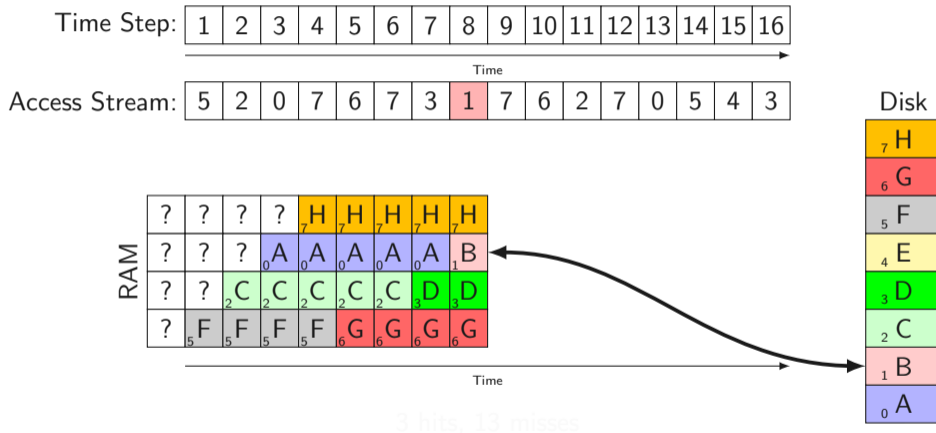


3 hits, 13 misses

FIFO PRA



FIFO PRA



FIFO PRA

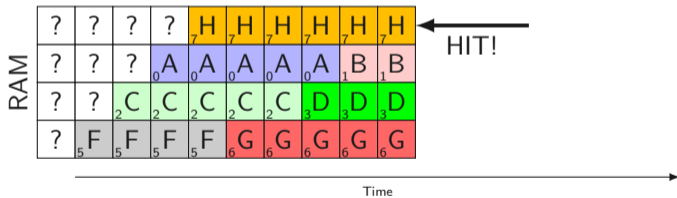
Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

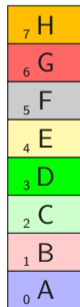
Time

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



Disk



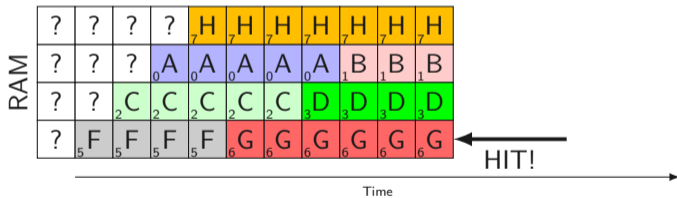
3 hits, 13 misses

FIFO PRA

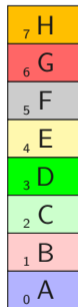
Time Step: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Time

Access Stream: 5 2 0 7 6 7 3 1 7 6 2 7 0 5 4 3

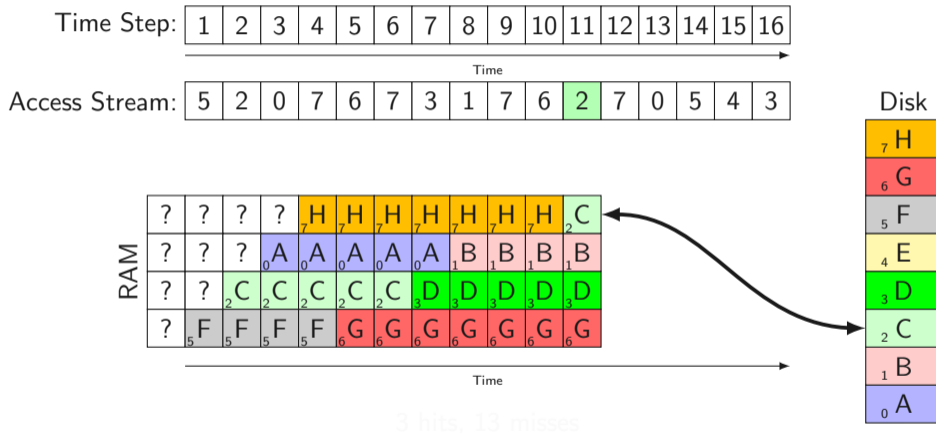


Disk

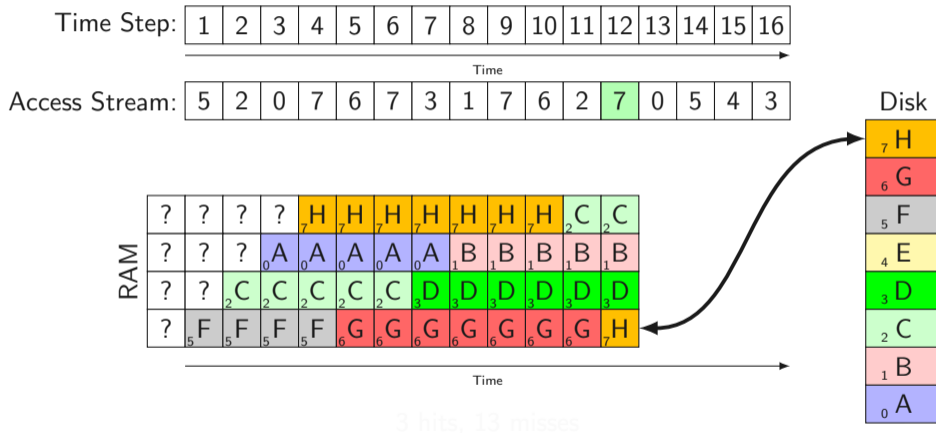


3 hits, 13 misses

FIFO PRA



FIFO PRA



FIFO PRA

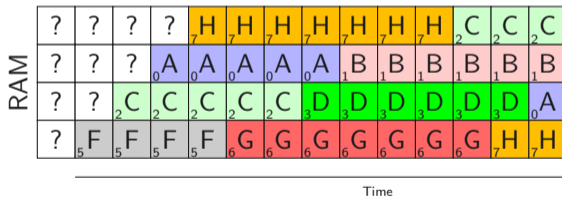
Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

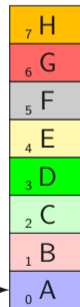
Time →

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



Disk



3 hits, 13 misses

FIFO PRA

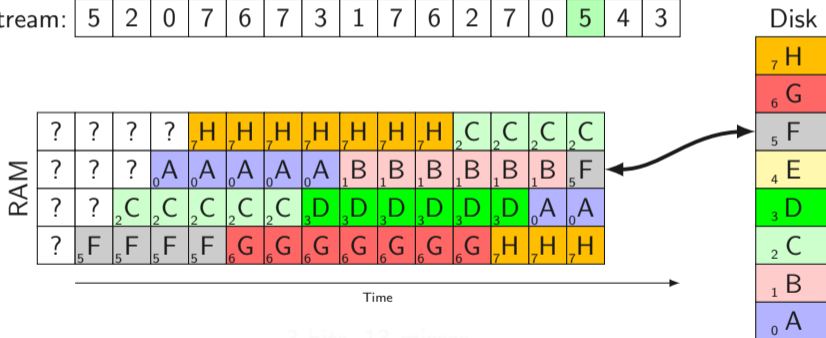
Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

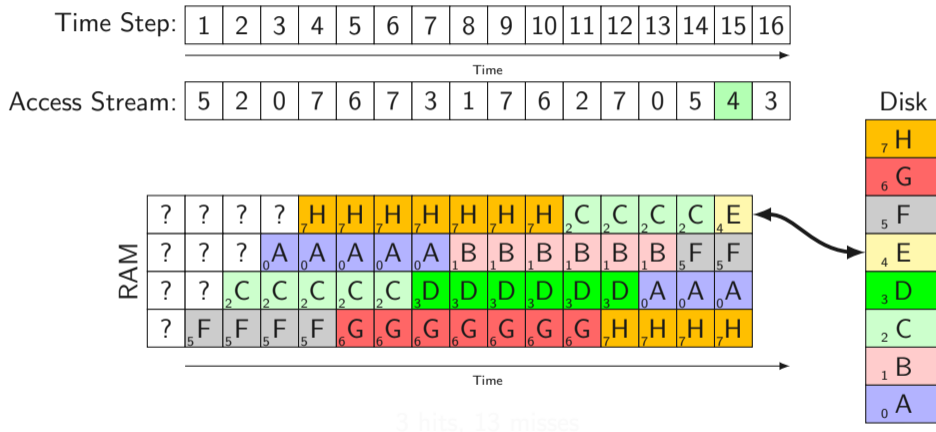
Time

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---



FIFO PRA

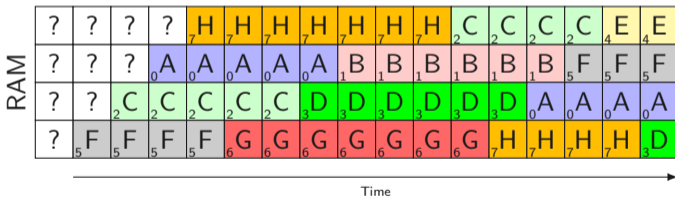


FIFO PRA

Time Step: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Time

Access Stream: 5 2 0 7 6 7 3 1 7 6 2 7 0 5 4 3



Disk



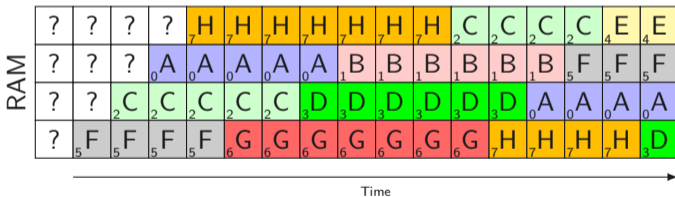
3 hits, 13 misses

FIFO PRA

Time Step: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

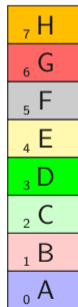
Time

Access Stream: 5 2 0 7 6 7 3 1 7 6 2 7 0 5 4 3



3 hits, 13 misses

Disk





- Idea: Make FIFO great again!



- Idea: Make FIFO great again!
 - We could call it FI(ANR)FO: “First-in-and-not-referenced first-out”



- Idea: Make FIFO great again!
 - We could call it FI(ANR)FO: “First-in-and-not-referenced first-out”
- Check “referenced” -bit:



- Idea: Make FIFO great again!
 - We could call it FI(ANR)FO: “First-in-and-not-referenced first-out”
- Check “referenced”-bit:
 - $R = 0$? evict

- Idea: Make FIFO great again!
 - We could call it FI(ANR)FO: “First-in-and-not-referenced first-out”
- Check “referenced”-bit:
 - $R = 0$? evict
 - $R = 1$? set $R = 0$ and go to next page

- Idea: Make FIFO great again!
 - We could call it FI(ANR)FO: “First-in-and-not-referenced first-out”
- Check “referenced”-bit:
 - $R = 0$? evict
 - $R = 1$? set $R = 0$ and go to next page
- Performance may degenerate to FIFO PRA (\rightarrow which may be worse than Random PRA)





- Virtually identical to Second Chance!




- Virtually identical to Second Chance!
- Only difference is the data structure




- Virtually identical to Second Chance!
- Only difference is the data structure
 - Second Chance: List + List Operations (`push_back`, `pop_front`)

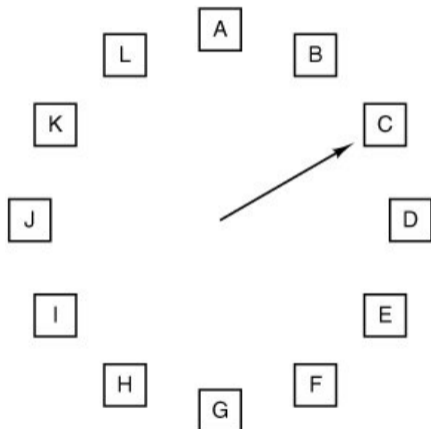


- Virtually identical to Second Chance!
- Only difference is the data structure
 - Second Chance: List + List Operations (`push_back`, `pop_front`)
 - Clock: Linked List + Pointer 

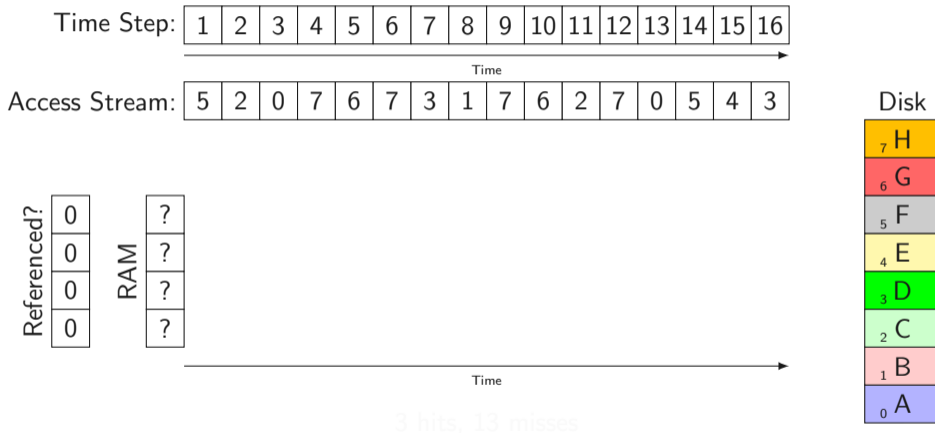


- Virtually identical to Second Chance!
- Only difference is the data structure
 - Second Chance: List + List Operations (`push_back`, `pop_front`)
 - Clock: Linked List + Pointer 
- Performance may degenerate to FIFO PRA (→ which may be worse than Random PRA)

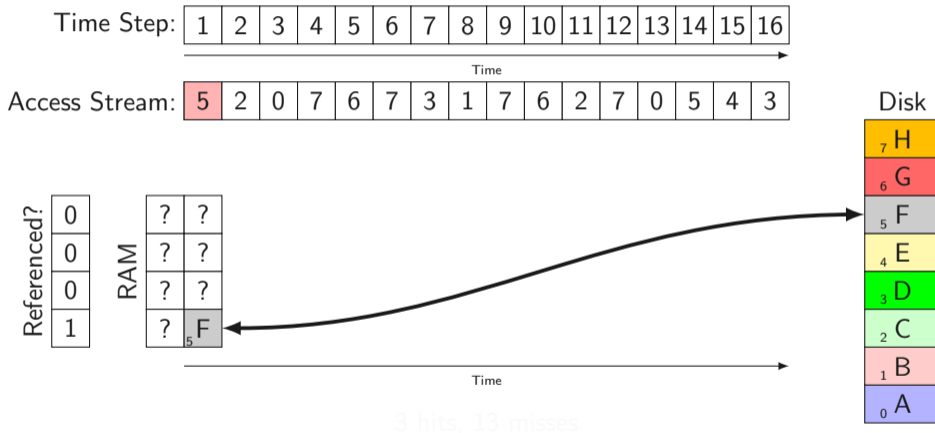




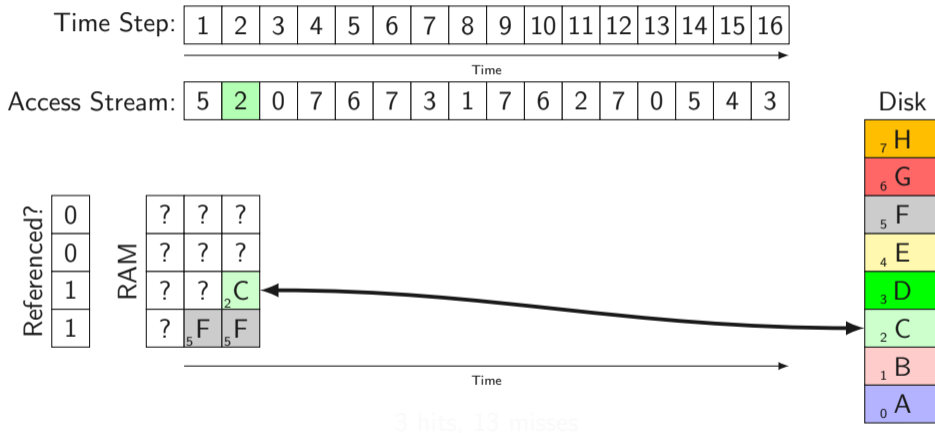
Second Chance PRA / Clock PRA



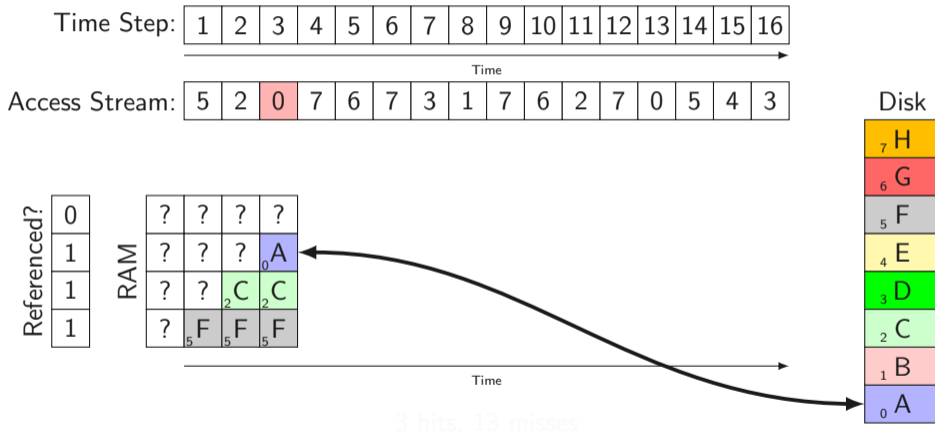
Second Chance PRA / Clock PRA



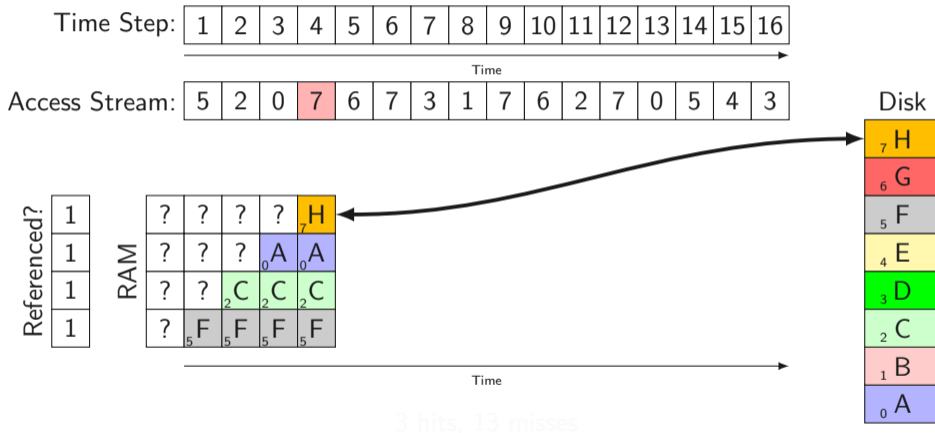
Second Chance PRA / Clock PRA



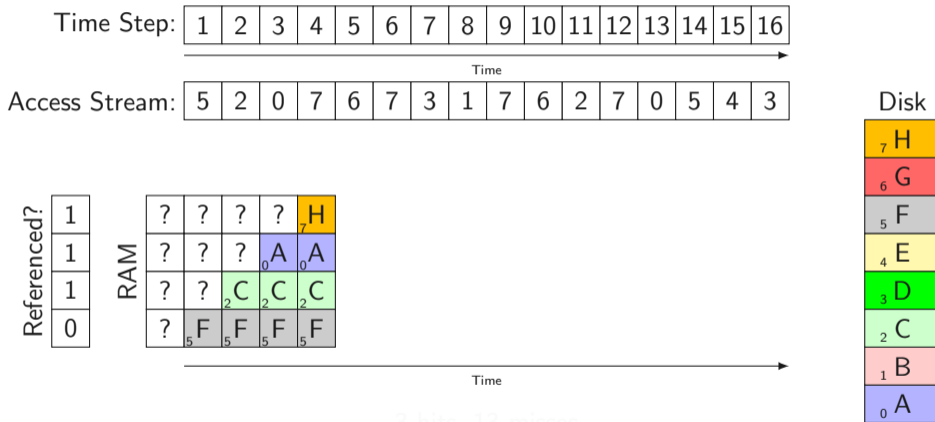
Second Chance PRA / Clock PRA



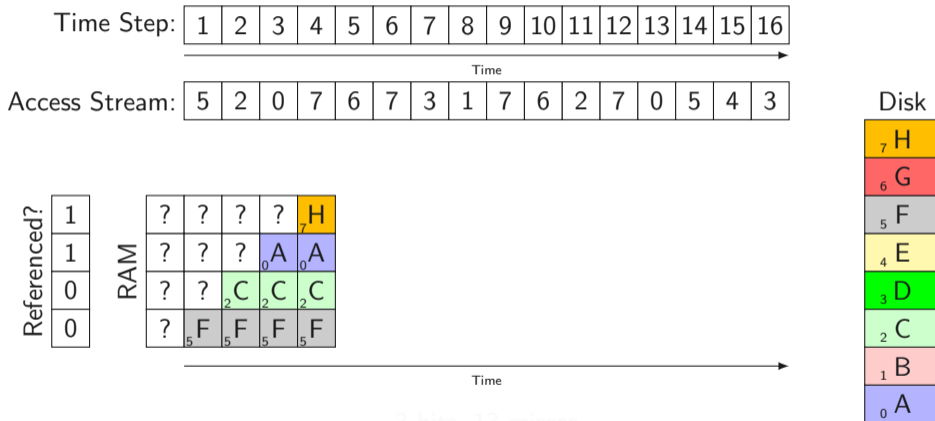
Second Chance PRA / Clock PRA



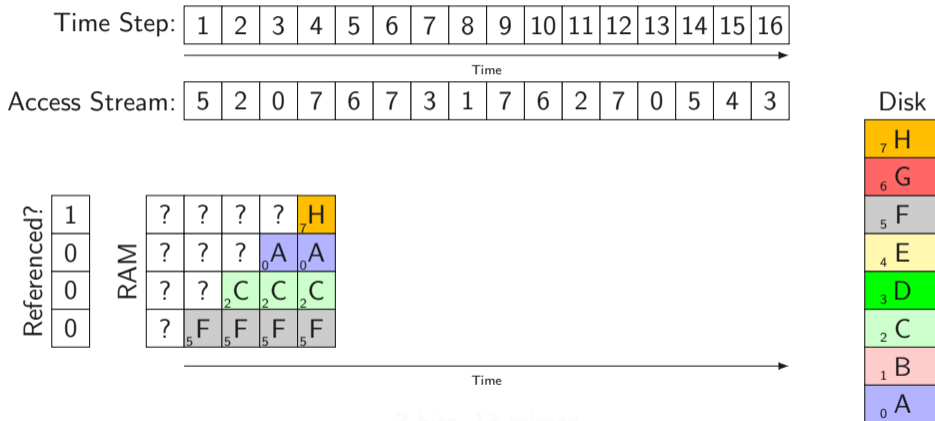
Second Chance PRA / Clock PRA



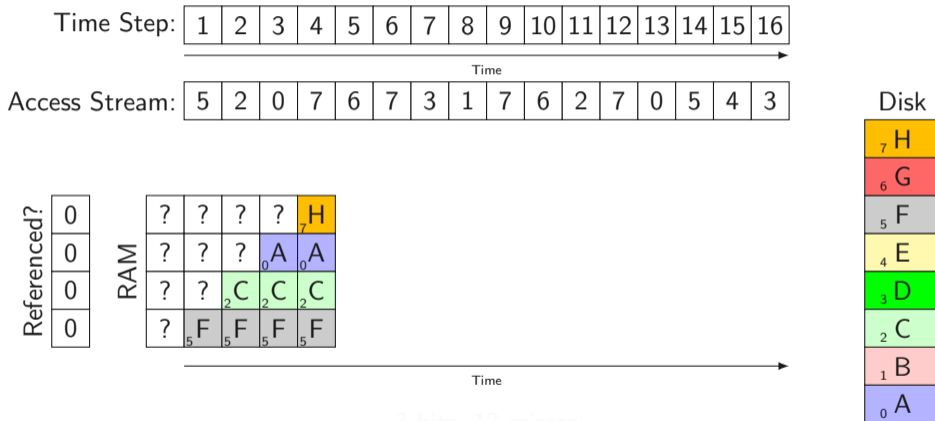
Second Chance PRA / Clock PRA



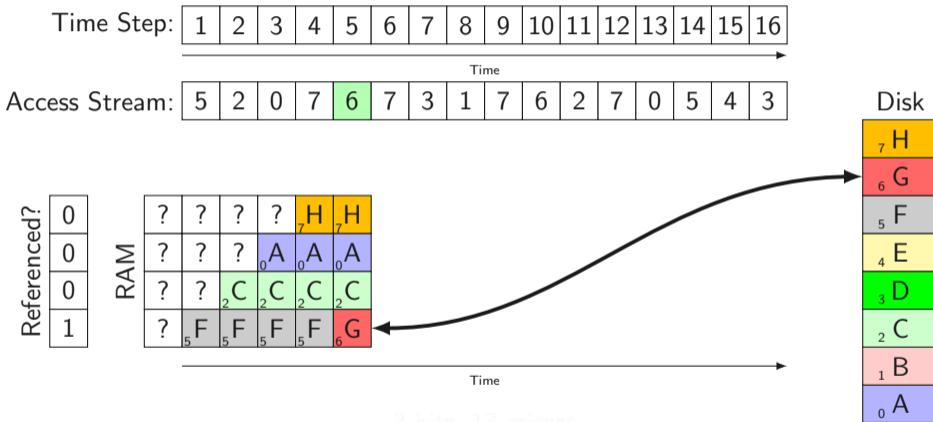
Second Chance PRA / Clock PRA



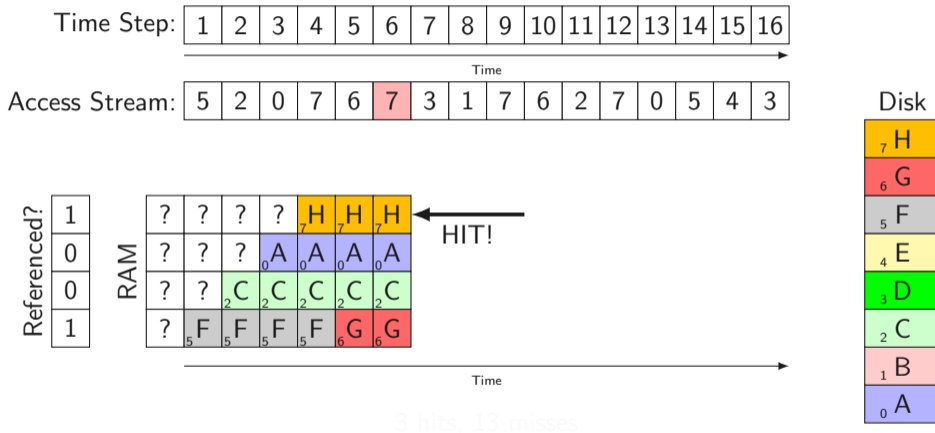
Second Chance PRA / Clock PRA



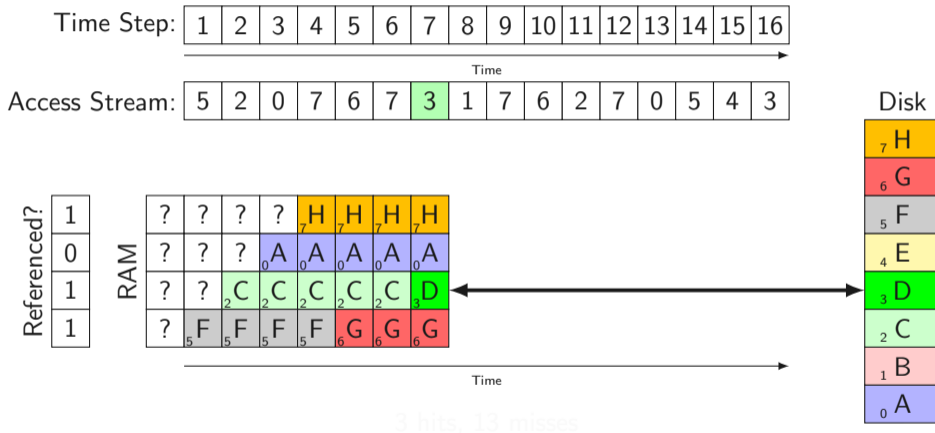
Second Chance PRA / Clock PRA



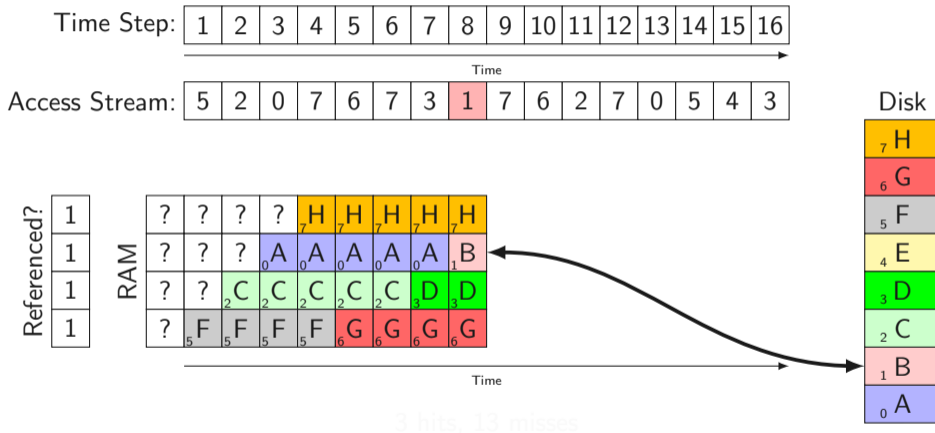
Second Chance PRA / Clock PRA



Second Chance PRA / Clock PRA



Second Chance PRA / Clock PRA



Second Chance PRA / Clock PRA

Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Referenced?

1
1
1
1

RAM

?	?	?	?	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇
?	?	?	A ₀	A ₀	A ₀	A ₀	A ₀	B ₁	B ₁
?	?	C ₂	C ₂	C ₂	C ₂	C ₂	D ₃	D ₃	D ₃
?	F ₅	F ₅	F ₅	F ₅	G ₆	G ₆	G ₆	G ₆	G ₆

HIT!

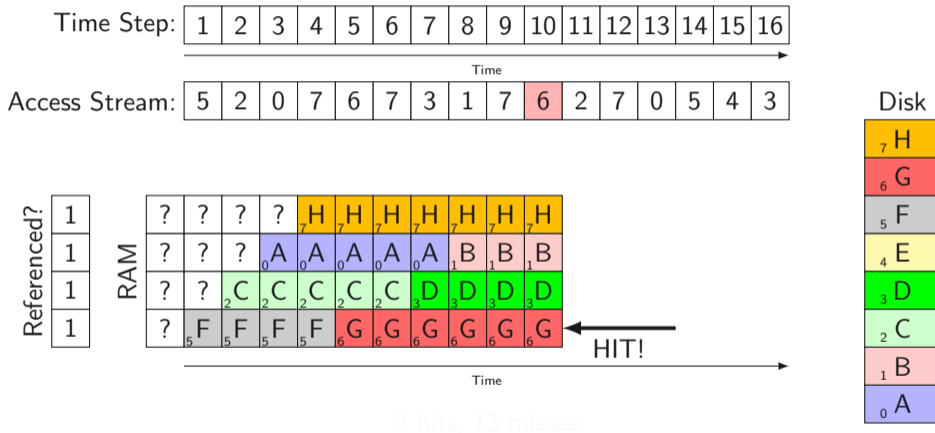
Time

3 hits, 13 misses

Disk

7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A

Second Chance PRA / Clock PRA



Second Chance PRA / Clock PRA

Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Referenced?

0
1
1
1

RAM	?	?	?	?	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇
	?	?	?	A ₀	A ₀	A ₀	A ₀	A ₀	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁
	?	?	C ₂	C ₂	C ₂	C ₂	C ₂	C ₂	D ₃	D ₃	D ₃	D ₃	D ₃	D ₃	D ₃
	?	F ₅	F ₅	F ₅	F ₅	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆

Time

Disk

7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A

3 hits, 13 misses

Second Chance PRA / Clock PRA

Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time →

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Referenced?
0
1
1
0

RAM	?	?	?	?	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇
	?	?	?	A ₀	A ₀	A ₀	A ₀	A ₁	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁
	?	?	C ₂	C ₂	C ₂	C ₂	C ₂	D ₃	D ₃	D ₃	D ₃	D ₃	D ₃	D ₃	D ₃
	?	F ₅	F ₅	F ₅	F ₅	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆

Time →

3 hits, 13 misses

Disk

7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A

Second Chance PRA / Clock PRA

Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

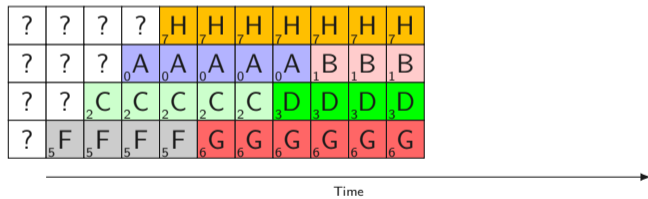
Time →

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

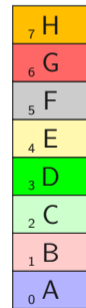
Referenced?
0
1
0
0

RAM



3 hits, 13 misses

Disk



Second Chance PRA / Clock PRA

Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Referenced?
0
0
0
0

RAM

?	?	?	?	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇
?	?	?	A ₀	A ₀	A ₀	A ₀	A ₀	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁
?	?	C ₂	C ₂	C ₂	C ₂	C ₂	C ₂	D ₃	D ₃	D ₃	D ₃	D ₃	D ₃	D ₃	D ₃
?	F ₅	F ₅	F ₅	F ₅	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆

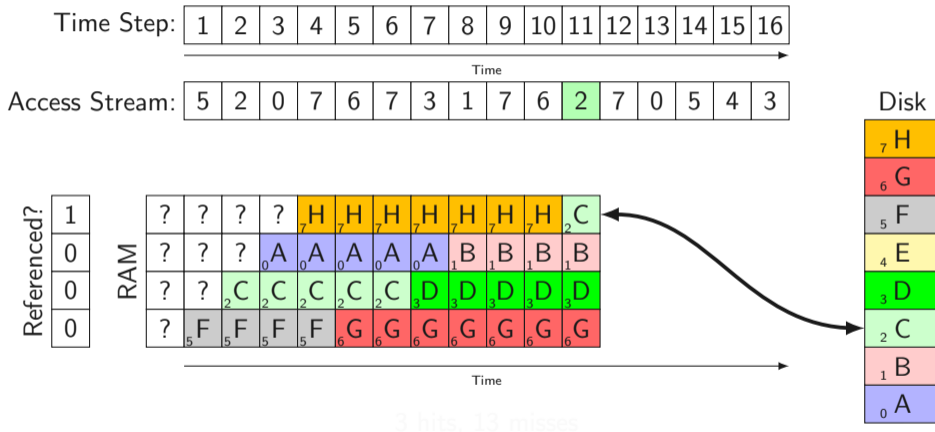
Time

Disk

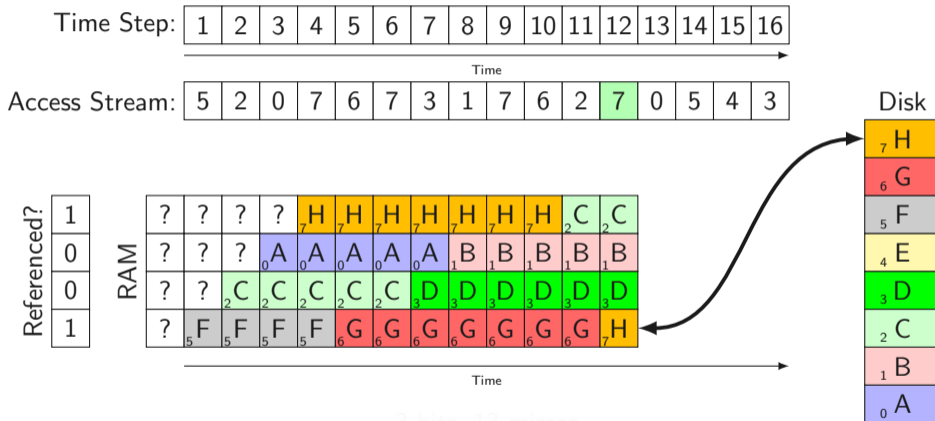
7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A

3 hits, 13 misses

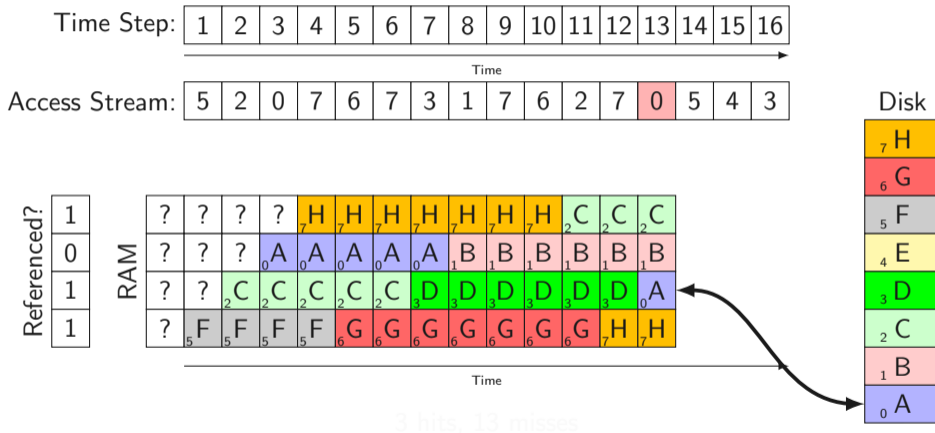
Second Chance PRA / Clock PRA



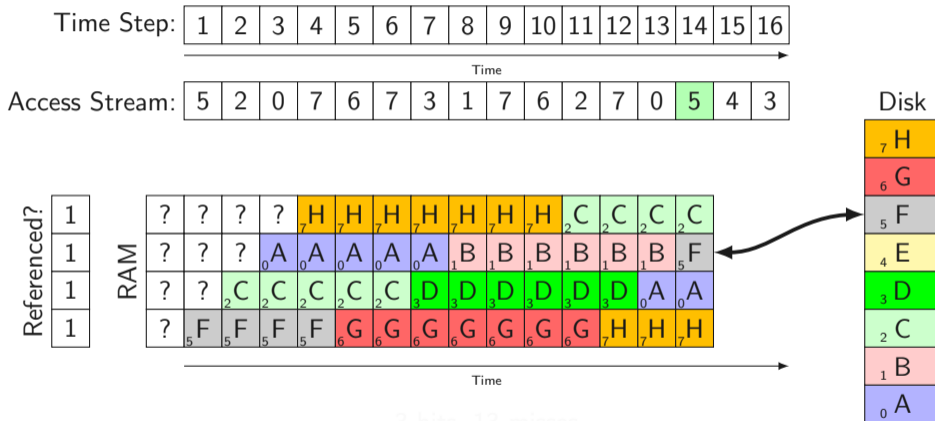
Second Chance PRA / Clock PRA



Second Chance PRA / Clock PRA



Second Chance PRA / Clock PRA



Second Chance PRA / Clock PRA

Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Referenced?

0
1
1
1

RAM

?	?	?	?	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	C ₂	C ₂	C ₂	C ₂
?	?	?	A ₀	A ₀	A ₀	A ₀	A ₀	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁	F ₅
?	?	C ₂	C ₂	C ₂	C ₂	C ₂	C ₂	D ₃	D ₃	D ₃	D ₃	D ₃	D ₃	A ₀	A ₀
?	F ₅	F ₅	F ₅	F ₅	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	H ₇	H ₇

Time

Disk

7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A

3 hits, 13 misses

Second Chance PRA / Clock PRA

Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time →

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Referenced?
0
1
1
0

RAM

?	?	?	?	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	C ₂	C ₂	C ₂	C ₂
?	?	?	A ₀	A ₀	A ₀	A ₀	A ₀	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁	B ₅	F ₅
?	?	C ₂	C ₂	C ₂	C ₂	C ₂	C ₂	D ₃	D ₃	D ₃	D ₃	D ₃	D ₃	A ₀	A ₀
?	F ₅	F ₅	F ₅	F ₅	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	H ₇	H ₇

Time →

Disk

7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A

3 hits, 13 misses

Second Chance PRA / Clock PRA

Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time →

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Referenced?
0
1
0
0

RAM

?	?	?	?	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	C ₂	C ₂	C ₂	C ₂
?	?	?	A ₀	A ₀	A ₀	A ₀	A ₀	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁	F ₅
?	?	C ₂	C ₂	C ₂	C ₂	C ₂	C ₂	D ₃	D ₃	D ₃	D ₃	D ₃	D ₃	A ₀	A ₀
?	F ₅	F ₅	F ₅	F ₅	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	H ₇	H ₇

Time →

Disk

7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A

3 hits, 13 misses

Second Chance PRA / Clock PRA

Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

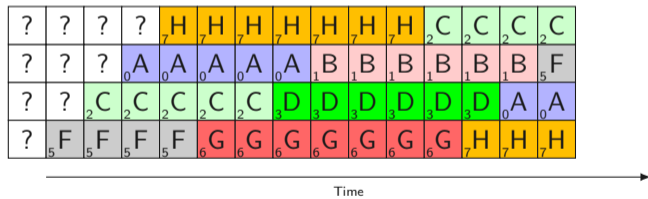
Time →

Access Stream:

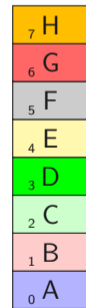
5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Referenced?
0
0
0
0

RAM

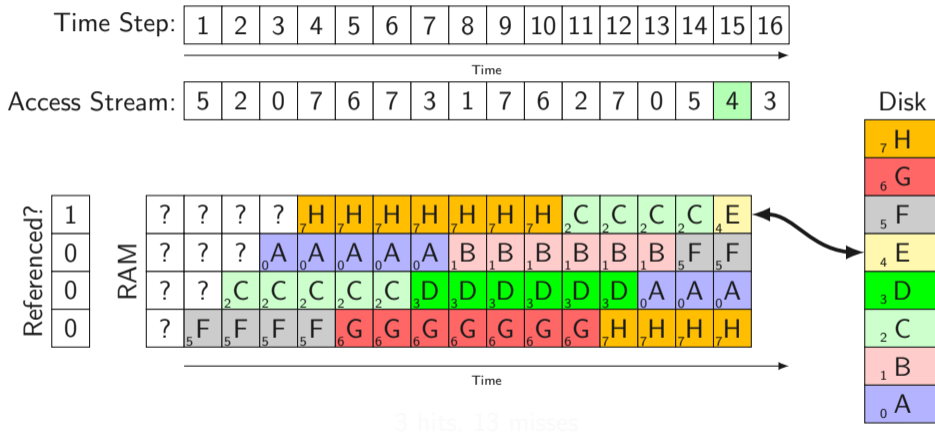


Disk

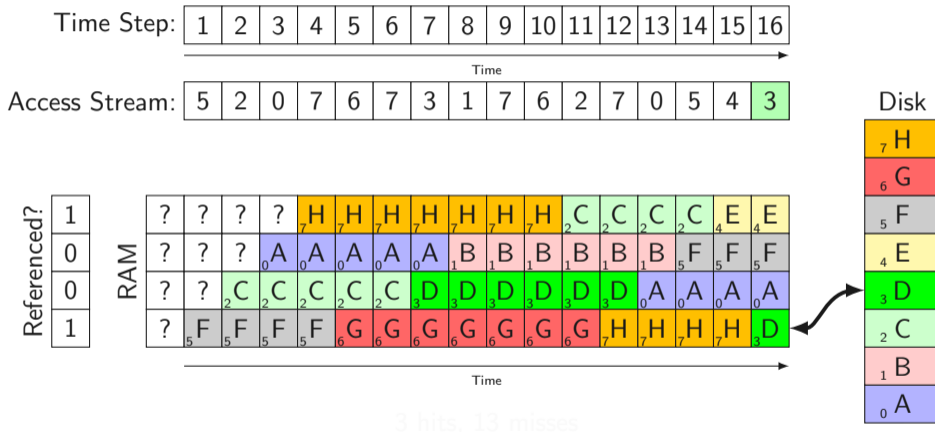


3 hits, 13 misses

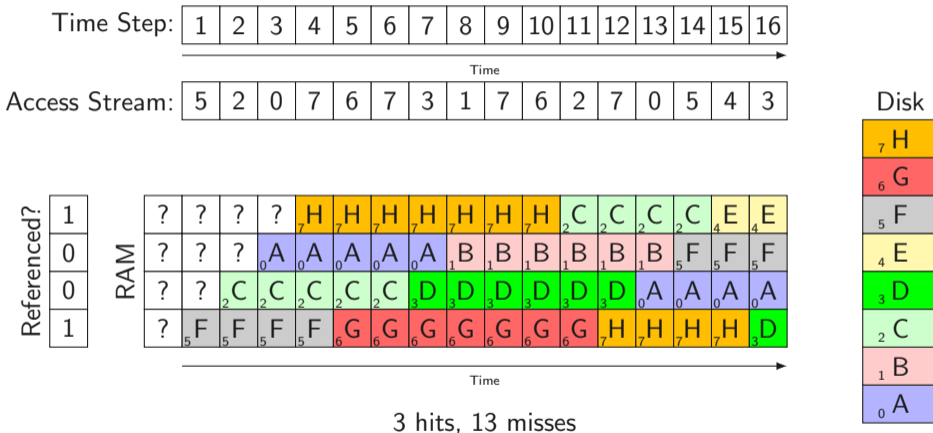
Second Chance PRA / Clock PRA



Second Chance PRA / Clock PRA



Second Chance PRA / Clock PRA



- Principle of Locality: Pages that were recently accessed will more likely be accessed again





- Principle of Locality: Pages that were recently accessed will more likely be accessed again
- Idea: Evict the page that was least recently accessed (used)





- Principle of Locality: Pages that were recently accessed will more likely be accessed again
- Idea: Evict the page that was least recently accessed (used)
- How do we find this page?


- LRU data structure:

- LRU data structure:
 - (Linked) list of all pages 



- LRU data structure:
 - (Linked) list of all pages 
 - Upon access: Move page to end of list


- LRU data structure:
 - (Linked) list of all pages 
 - Upon access: Move page to end of list
 - Page to evict? `pop_front`

- LRU data structure:
 - (Linked) list of all pages 
 - Upon access: Move page to end of list
 - Page to evict? `pop_front`
- Can this be done in software?

- LRU data structure:
 - (Linked) list of all pages 
 - Upon access: Move page to end of list
 - Page to evict? `pop_front`
- Can this be done in software?
 - Only with extreme performance penalty (enforce every memory access to cause a page fault)




- LRU data structure:
 - (Linked) list of all pages 
 - Upon access: Move page to end of list
 - Page to evict? `pop_front`
- Can this be done in software?
 - Only with extreme performance penalty (enforce every memory access to cause a page fault)
- 
- Can this be done in hardware?


- LRU data structure:
 - (Linked) list of all pages
 - Upon access: Move page to end of list
 - Page to evict? `pop_front`
- Can this be done in software?
 - Only with extreme performance penalty (enforce every memory access to cause a page fault)
- 
- Can this be done in hardware?
 - Reordering large data structures of variable size in hardware is difficult


- Global data structure for physical page “ages”

- Global data structure for physical page “ages”
 - Related: Where do you store the reference count for CoW-pages?

- Global data structure for physical page “ages”
 - Related: Where do you store the reference count for CoW-pages?
- Upon access to a page: Store current value of `rdtsc` (cycle counter)

- Global data structure for physical page “ages”
 - Related: Where do you store the reference count for CoW-pages?
- Upon access to a page: Store current value of `rdtsc` (cycle counter)
- Page replacement: Search data structure for lowest stored `rdtsc` value 

- Global data structure for physical page “ages”
 - Related: Where do you store the reference count for CoW-pages?
- Upon access to a page: Store current value of `rdtsc` (cycle counter)
- Page replacement: Search data structure for lowest stored `rdtsc` value 

- Global data structure for physical page “ages”
 - Related: Where do you store the reference count for CoW-pages?
- Upon access to a page: Store current value of `rdtsc` (cycle counter)
- Page replacement: Search data structure for lowest stored `rdtsc` value 

Can we implement this?

- Same trick as before:

- Same trick as before:
 - Poll page tables: read and reset referenced bits

- Same trick as before:
 - Poll page tables: read and reset referenced bits
 - Store `rdtsc` value as age in the global data structure

- Same trick as before:
 - Poll page tables: read and reset referenced bits
 - Store `rdtsc` value as age in the global data structure
- When do we do this?

- Same trick as before:
 - Poll page tables: read and reset referenced bits
 - Store `rdtsc` value as age in the global data structure
- When do we do this?
 - A thread continuously running and checking

- Same trick as before:
 - Poll page tables: read and reset referenced bits
 - Store `rdtsc` value as age in the global data structure
- When do we do this?
 - A thread continuously running and checking
 - Upon de-scheduling

- Same trick as before:
 - Poll page tables: read and reset referenced bits
 - Store `rdtsc` value as age in the global data structure
 - When do we do this?
 - A thread continuously running and checking
 - Upon de-scheduling
- = LRU PRA (which is actually pseudo-LRU)

- Same trick as before:
 - Poll page tables: read and reset referenced bits
 - Store `rdtsc` value as age in the global data structure
 - When do we do this?
 - A thread continuously running and checking
 - Upon de-scheduling
- = LRU PRA (which is actually pseudo-LRU)

- Same trick as before:
 - Poll page tables: read and reset referenced bits
 - Store `rdtsc` value as age in the global data structure
 - When do we do this?
 - A thread continuously running and checking
 - Upon de-scheduling
- = LRU PRA (which is actually pseudo-LRU)

Performance?

- Same trick as before:
 - Poll page tables: read and reset referenced bits
 - Store `rdtsc` value as age in the global data structure
 - When do we do this?
 - A thread continuously running and checking
 - Upon de-scheduling
- = LRU PRA (which is actually pseudo-LRU)

Performance? You have 8 MB RAM and loop over a 8.1 MB array → very bad performance





Time Step: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Time

Access Stream: 5 2 0 7 6 7 3 1 7 6 2 7 0 5 4 3

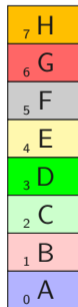
Last Used
?
?
?
?

RAM

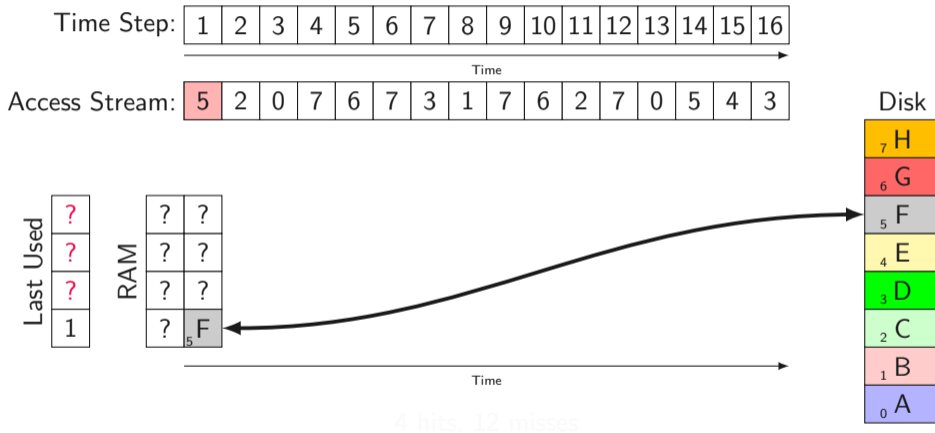
?
?
?
?

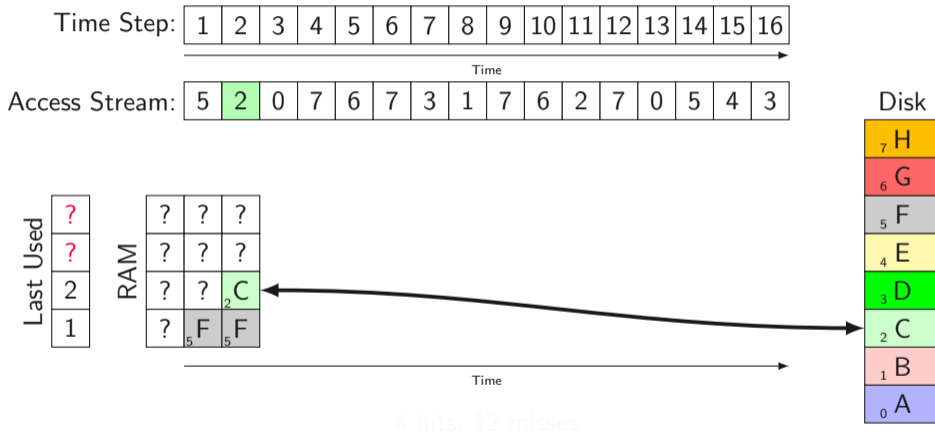
Time

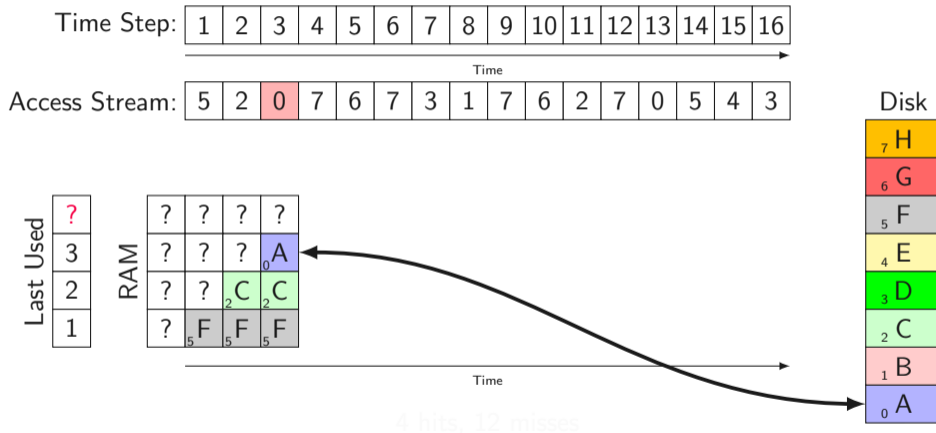
Disk

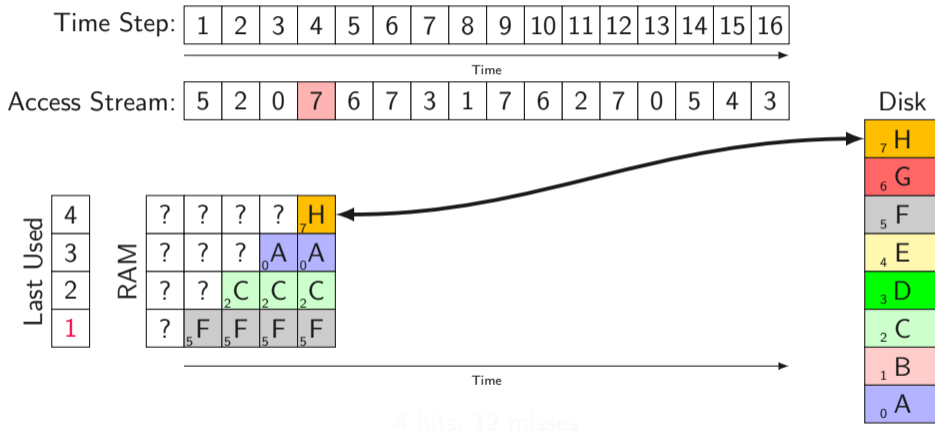


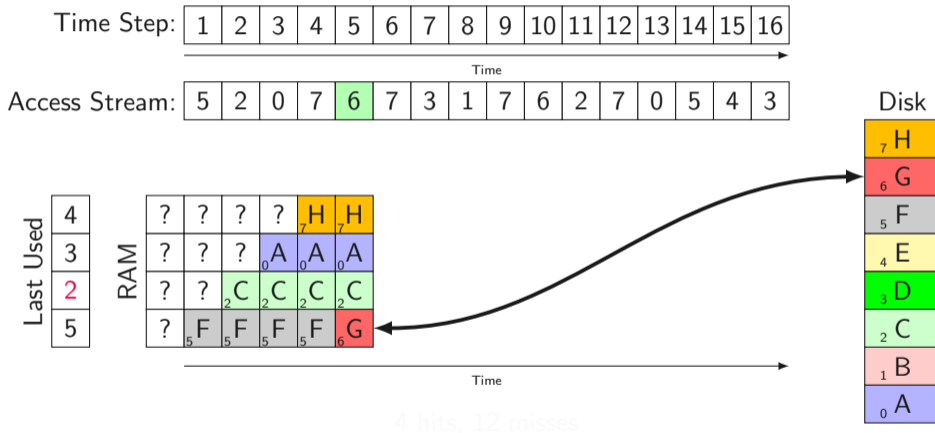
4 hits, 12 misses













Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time →

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Last Used

6
3
2
5

RAM

?	?	?	?	H ₇	H ₇	H ₇
?	?	?	A ₀	A ₀	A ₀	A ₀
?	?	C ₂	C ₂	C ₂	C ₂	C ₂
?	F ₅	F ₅	F ₅	F ₅	G ₆	G ₆

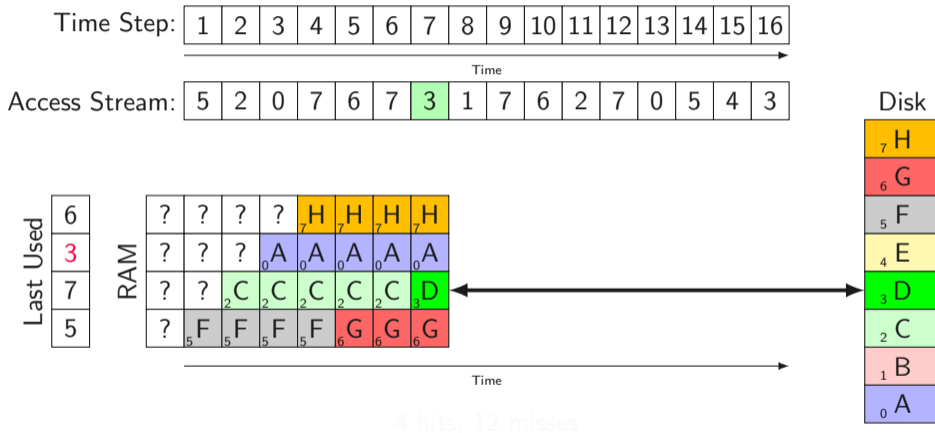
HIT!

Time →

4 hits, 12 misses

Disk

7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A



Time Step:

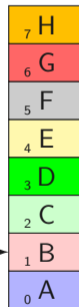
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time →

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Disk



Last Used

6
8
7
5



Time →

4 hits, 12 misses



Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time →

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Last Used

9
8
7
5

RAM

?	?	?	?	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇
?	?	?	A ₀	A ₀	A ₀	A ₀	A ₀	B ₁	B ₁
?	?	C ₂	C ₂	C ₂	C ₂	C ₂	D ₃	D ₃	D ₃
?	F ₅	F ₅	F ₅	F ₅	G ₆	G ₆	G ₆	G ₆	G ₆

HIT!

Time →

4 hits, 12 misses

Disk

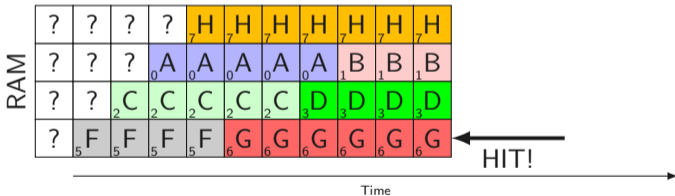
7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A

Time Step: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

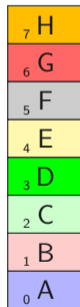
Time

Access Stream: 5 2 0 7 6 7 3 1 7 6 2 7 0 5 4 3

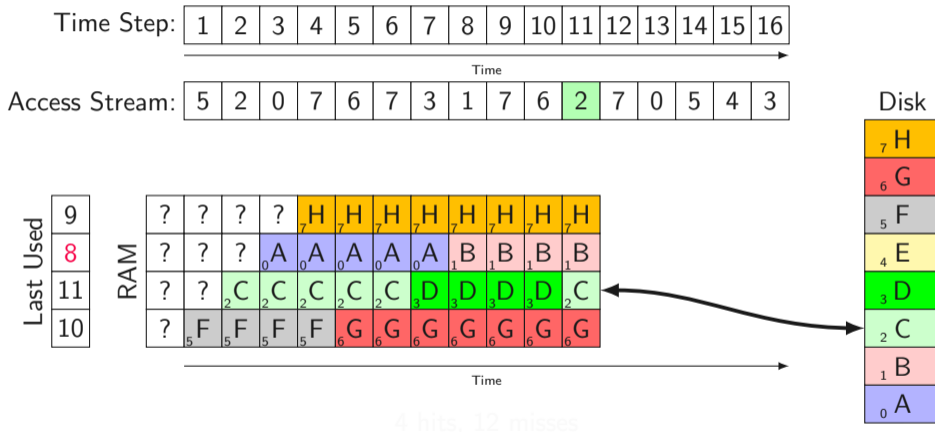
Last Used
9
8
7
10



Disk



4 hits, 12 misses



Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Last Used

12
8
11
10

RAM

?	?	?	?	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇
?	?	?	A ₀	A ₀	A ₀	A ₀	A ₀	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁
?	?	C ₂	C ₂	C ₂	C ₂	C ₂	C ₂	D ₃	D ₃	D ₃	D ₃	D ₃	C ₂	C ₂	C ₂
?	F ₅	F ₅	F ₅	F ₅	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆

HIT!

Time

4 hits, 12 misses

Disk

7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A

Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time →

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Last Used

12
13
11
10

RAM

?	?	?	?	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇
?	?	?	A ₀	A ₀	A ₀	A ₀	A ₀	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁	A ₀	A ₀
?	?	C ₂	C ₂	C ₂	C ₂	C ₂	C ₂	D ₃	D ₃	D ₃	D ₃	C ₂	C ₂	C ₂	C ₂
?	F ₅	F ₅	F ₅	F ₅	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆

Time →

Disk

7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A

4 hits, 12 misses

Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Last Used

12
13
11
14

RAM

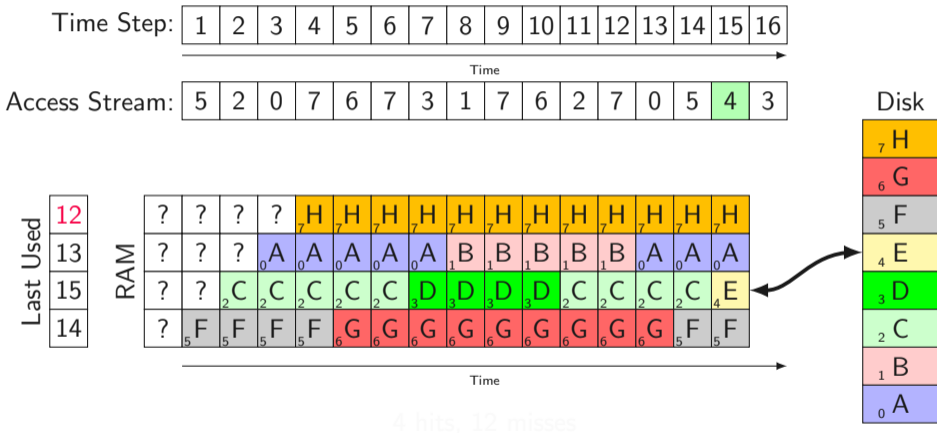
?	?	?	?	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇
?	?	?	A ₀	A ₀	A ₀	A ₀	A ₀	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁	A ₀	A ₀
?	?	C ₂	C ₂	C ₂	C ₂	C ₂	D ₃	D ₃	D ₃	D ₃	C ₂	C ₂	C ₂	C ₂	
?	F ₅	F ₅	F ₅	F ₅	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	F ₅

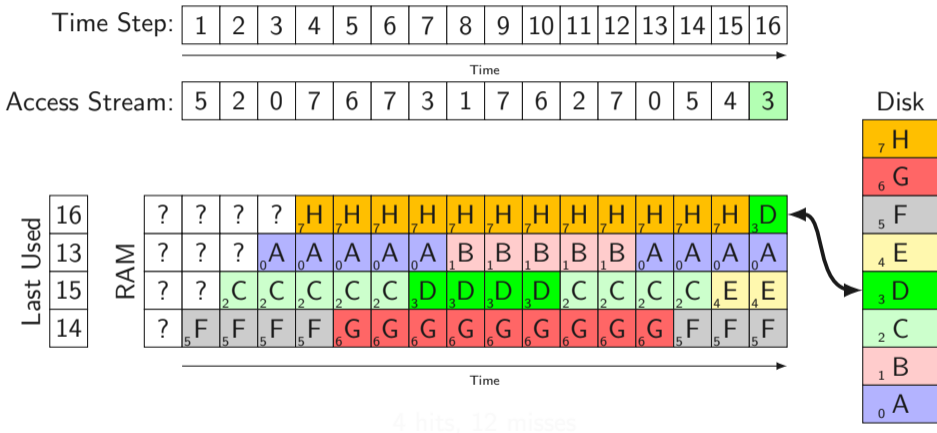
Time

Disk

7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A

4 hits, 12 misses





Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time →

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Last Used

16
13
15
14

RAM

?	?	?	?	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	D ₃
?	?	?	A ₀	A ₀	A ₀	A ₀	A ₁	B ₁	B ₁	B ₁	B ₁	B ₁	A ₀	A ₀	A ₀	A ₀
?	?	C ₂	C ₂	C ₂	C ₂	C ₂	D ₃	D ₃	D ₃	D ₃	C ₂	C ₂	C ₂	C ₂	E ₄	E ₄
?	F ₅	F ₅	F ₅	F ₅	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	F ₅	F ₅


Time →

4 hits, 12 misses


Disk


7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A

- Again: Principle of Locality 

- Again: Principle of Locality 
- Idea: Record frequency of accesses and evict page with lowest access frequency



- Again: Principle of Locality 
- Idea: Record frequency of accesses and evict page with lowest access frequency
- Approximate frequency by access count


- Again: Principle of Locality 
- Idea: Record frequency of accesses and evict page with lowest access frequency
- Approximate frequency by access count
- How do we obtain the access count?





- Global data structure for physical page access frequency



- Global data structure for physical page access frequency
- Upon access to a page: Increment access counter

- Global data structure for physical page access frequency
- Upon access to a page: Increment access counter
- Page replacement: Search for lowest counter value 

- Global data structure for physical page access frequency
- Upon access to a page: Increment access counter
- Page replacement: Search for lowest counter value 

- Global data structure for physical page access frequency
- Upon access to a page: Increment access counter
- Page replacement: Search for lowest counter value 

Can we implement this?

- Same trick as before:



- Same trick as before:
 - Poll page tables: read and reset referenced bits

- Same trick as before:
 - Poll page tables: read and reset referenced bits
 - Increment access counter in the global data structure

- Same trick as before:
 - Poll page tables: read and reset referenced bits
 - Increment access counter in the global data structure
- When do we do this?

- Same trick as before:
 - Poll page tables: read and reset referenced bits
 - Increment access counter in the global data structure
- When do we do this?
 - A thread continuously running and checking

- Same trick as before:
 - Poll page tables: read and reset referenced bits
 - Increment access counter in the global data structure
- When do we do this?
 - A thread continuously running and checking
 - Upon de-scheduling

- Same trick as before:
 - Poll page tables: read and reset referenced bits
 - Increment access counter in the global data structure
- When do we do this?
 - A thread continuously running and checking
 - Upon de-scheduling

= NFU PRA

- Same trick as before:
 - Poll page tables: read and reset referenced bits
 - Increment access counter in the global data structure
- When do we do this?
 - A thread continuously running and checking
 - Upon de-scheduling

= NFU PRA

- Performance?

- Same trick as before:
 - Poll page tables: read and reset referenced bits
 - Increment access counter in the global data structure
- When do we do this?
 - A thread continuously running and checking
 - Upon de-scheduling

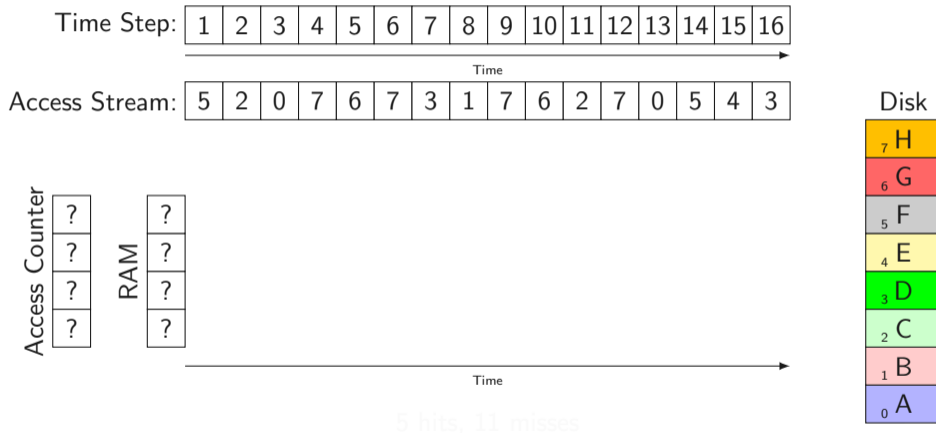
= NFU PRA

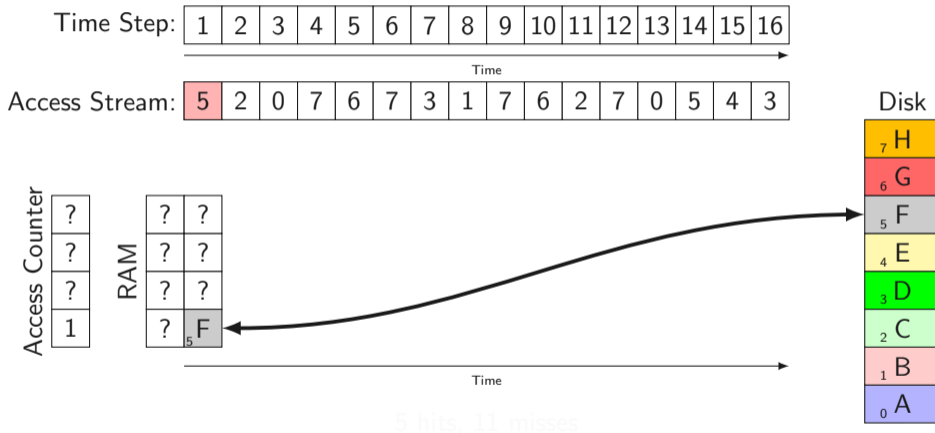
- Performance?

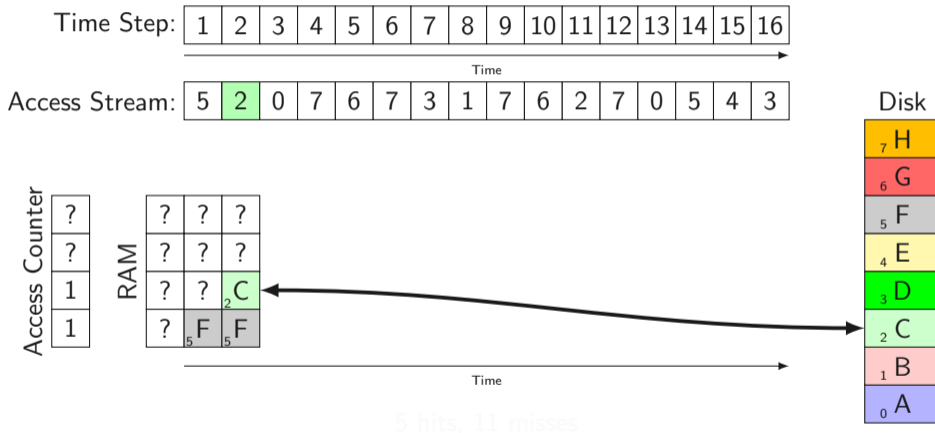
- Same trick as before:
 - Poll page tables: read and reset referenced bits
 - Increment access counter in the global data structure
- When do we do this?
 - A thread continuously running and checking
 - Upon de-scheduling

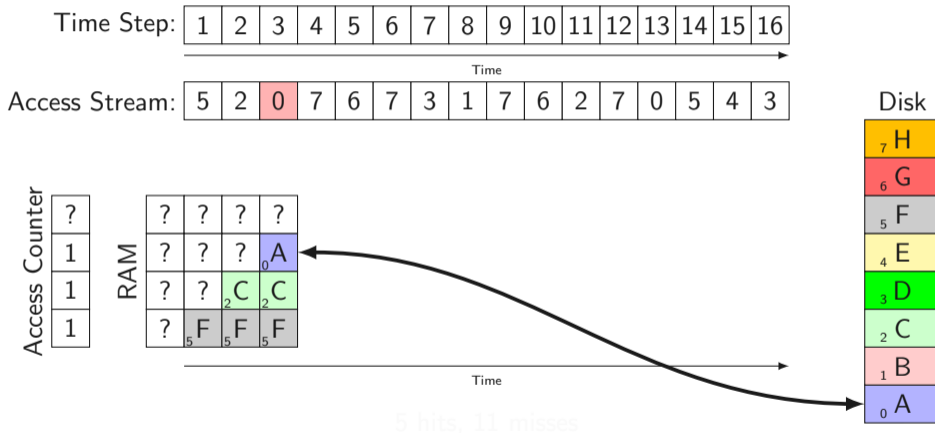
= NFU PRA

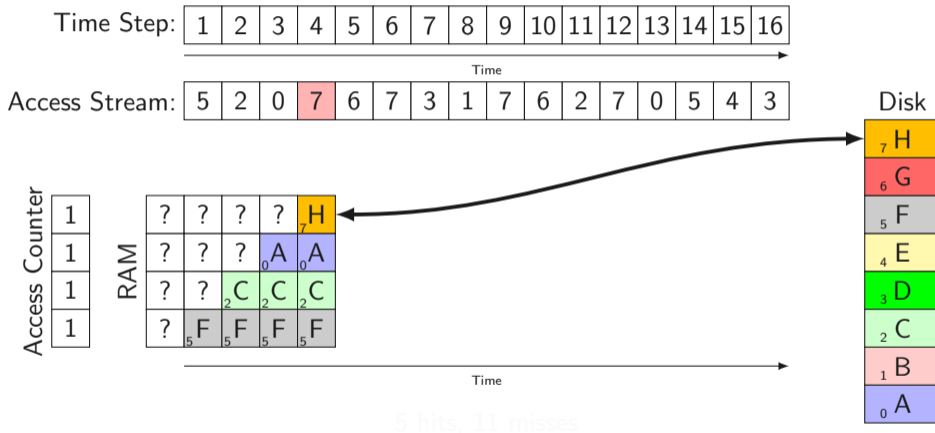
- Performance? Boot code very unlikely to be swapped (because it was used a lot during boot up)

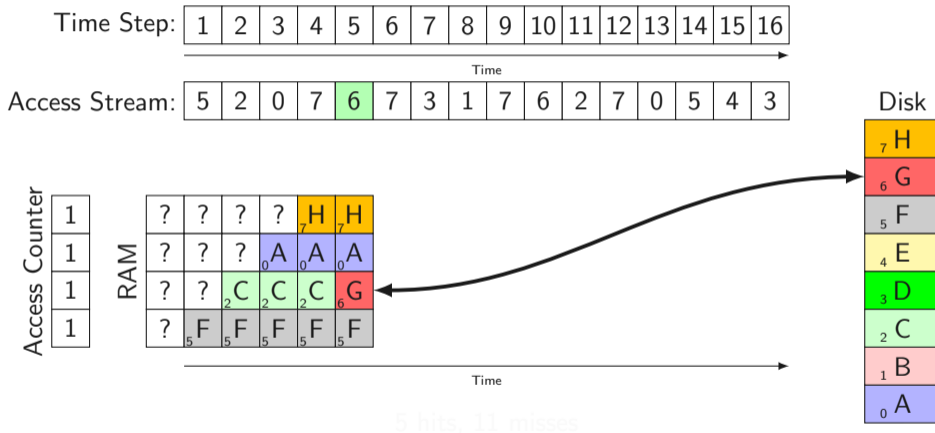












Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time →

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Access Counter

2
1
1
1

RAM

?	?	?	?	7	7	7
?	?	?	0	0	0	0
?	?	2	2	2	6	6
?	5	5	5	5	5	5

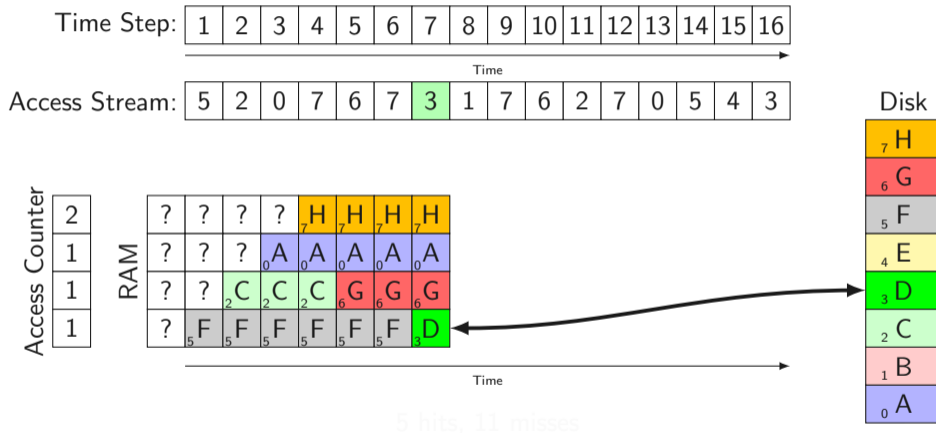
HIT!

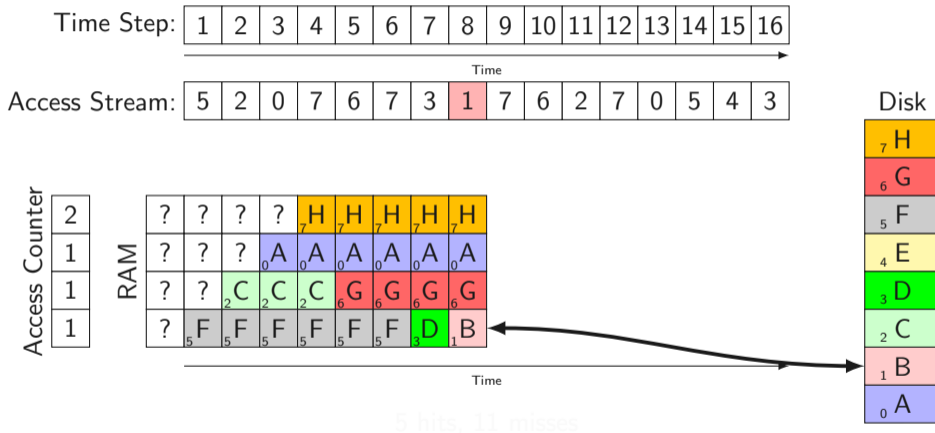
Time →

5 hits, 11 misses

Disk

7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A







Time Step:

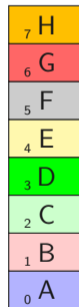
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time →

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Disk



Access Counter

3
1
1
1

RAM



HIT!

Time →

5 hits, 11 misses



Time Step:

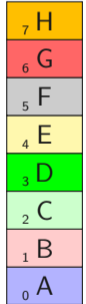
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time →

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Disk



Access Counter

3
1
2
1

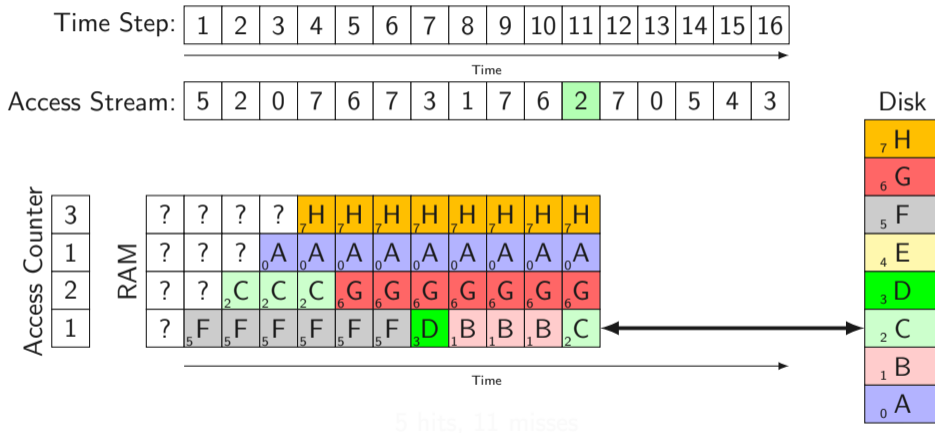
RAM



← HIT!

Time →

5 hits, 11 misses



Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Access Counter

4
1
2
1

RAM

?	?	?	?	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇
?	?	?	A ₀	A ₀	A ₀	A ₀	A ₀	A ₀	A ₀	A ₀	A ₀	A ₀	A ₀
?	?	C ₂	C ₂	C ₂	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆
?	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	D ₃	B ₁	B ₁	B ₁	B ₁	C ₂	C ₂

HIT!

Time

Disk

7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A

5 hits, 11 misses

Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Access Counter

4
2
2
1

RAM

?	?	?	?	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇
?	?	?	A ₀	A ₀	A ₀	A ₀	A ₀	A ₀	A ₀	A ₀	A ₀	A ₀	A ₀	A ₀	A ₀
?	?	C ₂	C ₂	C ₂	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆
?	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅

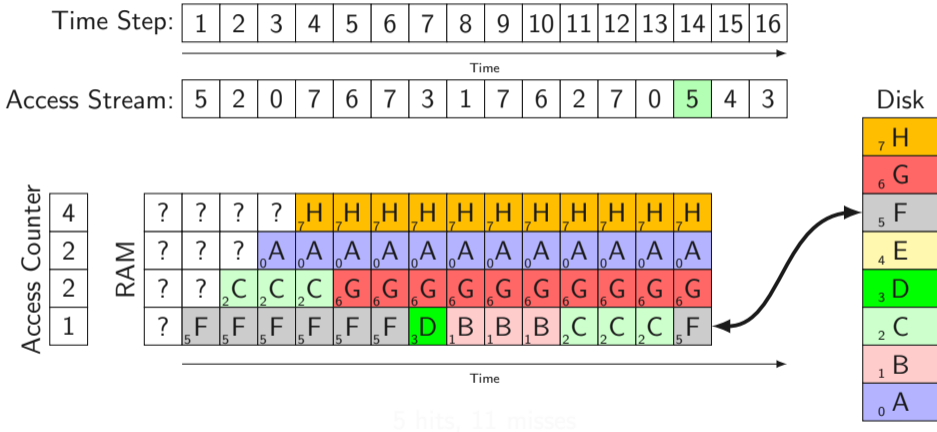
HIT!

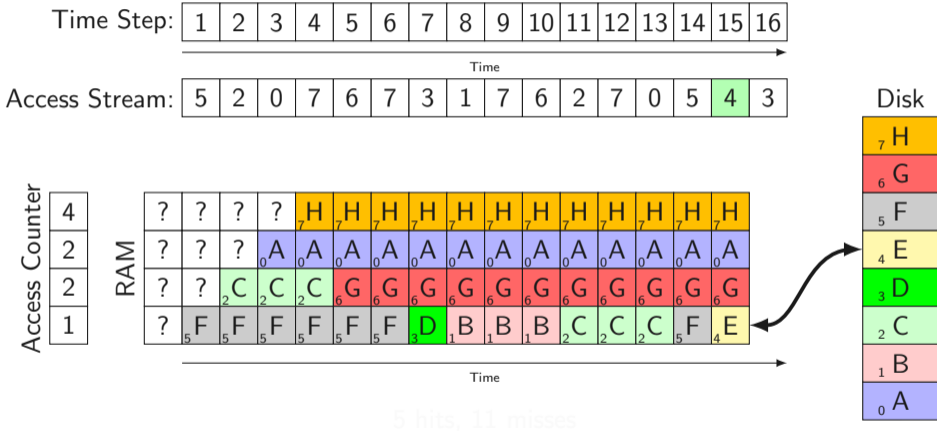
Time

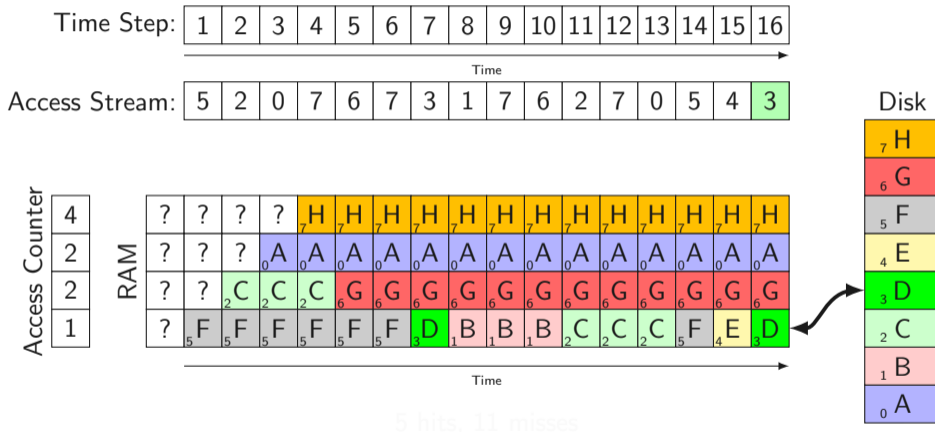
Disk

7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A

5 hits, 11 misses







Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Access Counter

4
2
2
1

RAM

?	?	?	?	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇
?	?	?	A ₀	A ₀	A ₀	A ₀	A ₀	A ₀	A ₀	A ₀	A ₀	A ₀	A ₀	A ₀	A ₀	A ₀
?	?	C ₂	C ₂	C ₂	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆
?	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	D ₃	B ₁	B ₁	B ₁	C ₂	C ₂	C ₂	F ₅	E ₄	D ₃

Time

5 hits, 11 misses

Disk

7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A

- NFU has problems because it never forgets (cf. human brain)



- NFU has problems because it never forgets (cf. human brain)
- Idea: Make NFU's memories slowly fade away



- NFU has problems because it never forgets (cf. human brain)
 - Idea: Make NFU's memories slowly fade away
- Let access information age over time




- NFU has problems because it never forgets (cf. human brain)
 - Idea: Make NFU's memories slowly fade away
- Let access information age over time
- How do we observe an access to a page?





- Global data structure for age

- Global data structure for age
- Upon access to a page: Set most-significant bit to 1 (e.g. 1000)

- Global data structure for age
- Upon access to a page: Set most-significant bit to 1 (e.g. 1000)
- In a constant frequency: Age all pages by shifting value in global data structure to the right (e.g. 1000 \rightarrow 0100)

- Global data structure for age
- Upon access to a page: Set most-significant bit to 1 (e.g. 1000)
- In a constant frequency: Age all pages by shifting value in global data structure to the right (e.g. 1000 \rightarrow 0100)
- Page replacement: Search for lowest numerical value (=highest age) 

- Global data structure for age
- Upon access to a page: Set most-significant bit to 1 (e.g. 1000)
- In a constant frequency: Age all pages by shifting value in global data structure to the right (e.g. 1000 \rightarrow 0100)
- Page replacement: Search for lowest numerical value (=highest age) 

- Global data structure for age
- Upon access to a page: Set most-significant bit to 1 (e.g. 1000)
- In a constant frequency: Age all pages by shifting value in global data structure to the right (e.g. 1000 \rightarrow 0100)
- Page replacement: Search for lowest numerical value (=highest age) 

Can we implement this?

- Same tricks as before:

- Same tricks as before:
 - Poll page tables: read and reset referenced bits

- Same tricks as before:
 - Poll page tables: read and reset referenced bits
 - Set most-significant bit in the global data structure

- Same tricks as before:
 - Poll page tables: read and reset referenced bits
 - Set most-significant bit in the global data structure
- When do we do this?

- Same tricks as before:
 - Poll page tables: read and reset referenced bits
 - Set most-significant bit in the global data structure
- When do we do this?
 - Before aging (shifting)

- Same tricks as before:
 - Poll page tables: read and reset referenced bits
 - Set most-significant bit in the global data structure
- When do we do this?
 - Before aging (shifting)
 - Upon de-scheduling

- Same tricks as before:
 - Poll page tables: read and reset referenced bits
 - Set most-significant bit in the global data structure
- When do we do this?
 - Before aging (shifting)
 - Upon de-scheduling
- When do we age (shift) the values?

- Same tricks as before:
 - Poll page tables: read and reset referenced bits
 - Set most-significant bit in the global data structure
- When do we do this?
 - Before aging (shifting)
 - Upon de-scheduling
- When do we age (shift) the values?
 - Set up a dedicated periodic interrupt

- Same tricks as before:
 - Poll page tables: read and reset referenced bits
 - Set most-significant bit in the global data structure
- When do we do this?
 - Before aging (shifting)
 - Upon de-scheduling
- When do we age (shift) the values?
 - Set up a dedicated periodic interrupt
 - Upon every n -th timer interrupt

- Same tricks as before:
 - Poll page tables: read and reset referenced bits
 - Set most-significant bit in the global data structure
- When do we do this?
 - Before aging (shifting)
 - Upon de-scheduling
- When do we age (shift) the values?
 - Set up a dedicated periodic interrupt
 - Upon every n -th timer interrupt

= Aging PRA

- Same tricks as before:
 - Poll page tables: read and reset referenced bits
 - Set most-significant bit in the global data structure
- When do we do this?
 - Before aging (shifting)
 - Upon de-scheduling
- When do we age (shift) the values?
 - Set up a dedicated periodic interrupt
 - Upon every n -th timer interrupt

= Aging PRA

- Performance?

- Same tricks as before:
 - Poll page tables: read and reset referenced bits
 - Set most-significant bit in the global data structure
- When do we do this?
 - Before aging (shifting)
 - Upon de-scheduling
- When do we age (shift) the values?
 - Set up a dedicated periodic interrupt
 - Upon every n -th timer interrupt

= Aging PRA

- Performance?

- Same tricks as before:
 - Poll page tables: read and reset referenced bits
 - Set most-significant bit in the global data structure
 - When do we do this?
 - Before aging (shifting)
 - Upon de-scheduling
 - When do we age (shift) the values?
 - Set up a dedicated periodic interrupt
 - Upon every n -th timer interrupt
- = Aging PRA
- Performance? One of the most widely used PRAs in practice

Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Ages

000000
000000
000000
000000

RAM

?
?
?
?

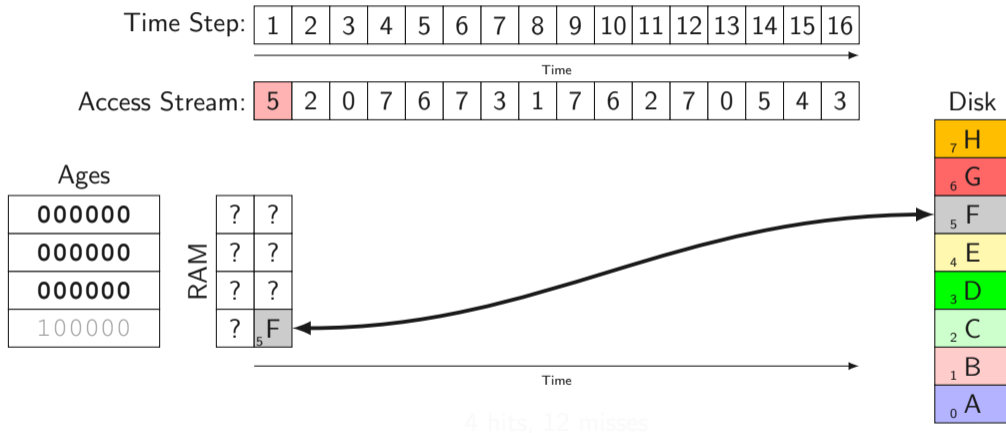
Time

4 hits, 12 misses

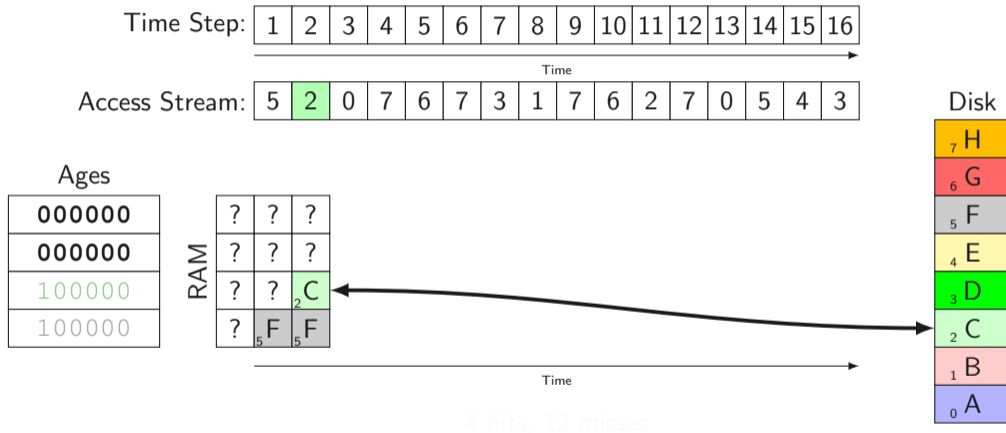
Disk

7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A

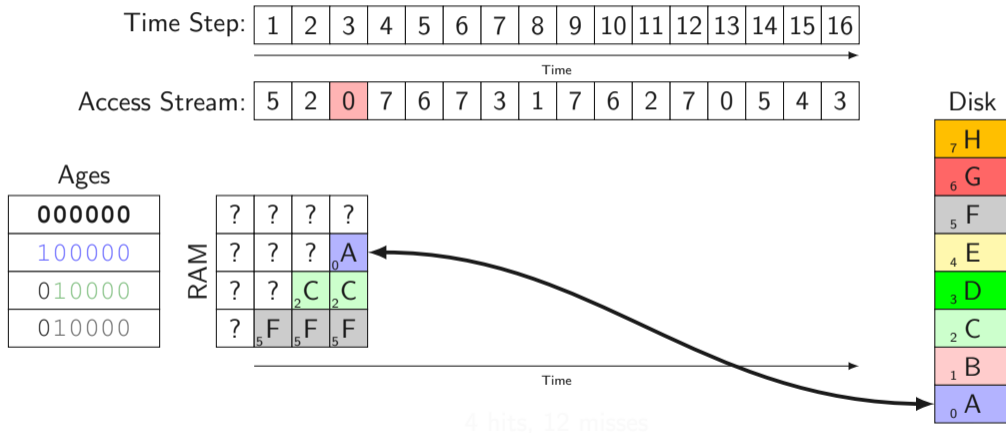
Aging PRA



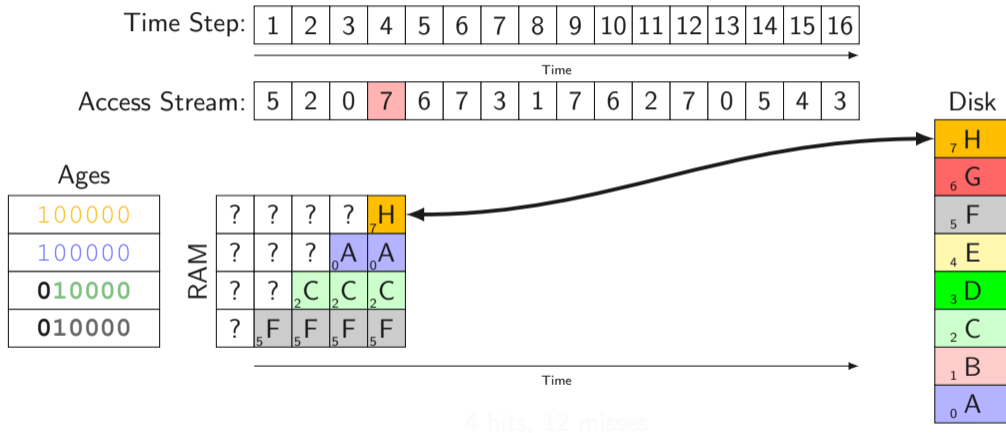
Aging PRA



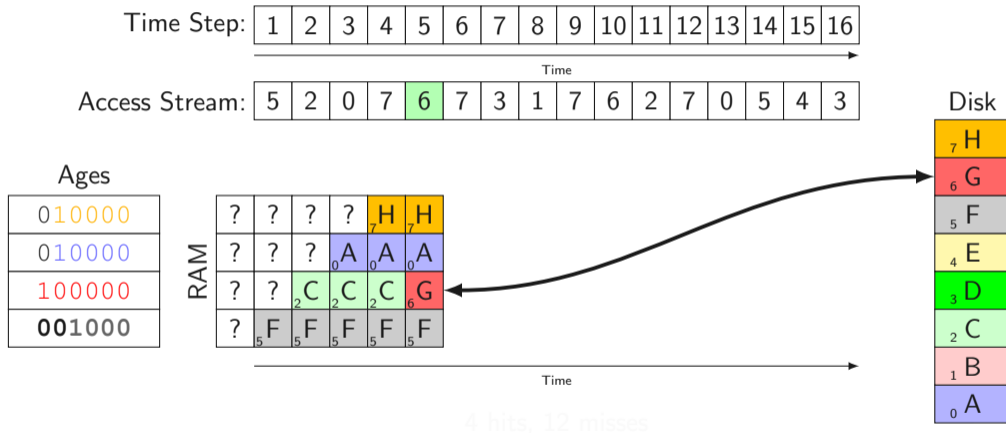
Aging PRA



Aging PRA



Aging PRA



Aging PRA

Time Step: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Time

Access Stream: 5 2 0 7 6 7 3 1 7 6 2 7 0 5 4 3

Ages

110000
010000
100000
001000

RAM	?	?	?	?	H ₇	H ₇	H ₇	← HIT!
	?	?	?	A ₀	A ₀	A ₀	A ₀	
	?	?	C ₂	C ₂	C ₂	G ₆	G ₆	
	?	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	

Time

4 hits, 12 misses

Disk

7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A

Aging PRA

Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Ages

0	1	1	0	0	0
0	0	0	1	0	0
0	1	0	0	0	0
1	0	0	0	0	0

RAM

?	?	?	?	H ₇	H ₇	H ₇	H ₇
?	?	?	A ₀	A ₀	A ₀	A ₀	A ₀
?	?	C ₂	C ₂	C ₂	G ₆	G ₆	G ₆
?	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	D ₃

Time

Disk

7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A

4 hits, 12 misses

Aging PRA

Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Ages

0	1	1	0	0	0
1	0	0	0	0	0
0	1	0	0	0	0
1	0	0	0	0	0

RAM

?	?	?	?	H ₇	H ₇	H ₇	H ₇	H ₇
?	?	?	A ₀	A ₀	A ₀	A ₀	A ₀	B ₁
?	?	C ₂	C ₂	C ₂	G ₆	G ₆	G ₆	G ₆
?	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	D ₃	D ₃

Time

Disk

7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A

4 hits, 12 misses

Aging PRA

Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Ages

101100
010000
001000
010000

?	?	?	?	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇
?	?	?	A ₀	A ₀	A ₀	A ₀	A ₀	B ₁	B ₁
?	?	C ₂	C ₂	C ₂	G ₆	G ₆	G ₆	G ₆	G ₆
?	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	D ₃	D ₃

HIT!

Time

4 hits, 12 misses

Disk

7 H
6 G
5 F
4 E
3 D
2 C
1 B
0 A

Aging PRA

Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Ages

101100
010000
101000
010000

RAM

?	?	?	?	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇
?	?	?	A ₀	A ₀	A ₀	A ₀	A ₀	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁
?	?	C ₂	C ₂	C ₂	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆
?	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	D ₃	D ₃	D ₃	D ₃	D ₃	D ₃	D ₃	D ₃

HIT!

Time

4 hits, 12 misses

Disk

H ₇
G ₆
F ₅
E ₄
D ₃
C ₂
B ₁
A ₀

Aging PRA

Time Step: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Time

Access Stream: 5 2 0 7 6 7 3 1 7 6 2 7 0 5 4 3

Ages

0	1	0	1	1	0
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	0	0

RAM

?	?	?	?	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇
?	?	?	A ₀	A ₀	A ₀	A ₀	A ₀	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁
?	?	C ₂	C ₂	C ₂	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆
?	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	D ₃	D ₃	D ₃	D ₃	D ₃	D ₃	C ₂	C ₂

Time

Disk

7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A

4 hits, 12 misses

Aging PRA

Time Step: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Time

Access Stream: 5 2 0 7 6 7 3 1 7 6 2 7 0 5 4 3

Ages

110110
001000
010100
100000

RAM

?	?	?	?	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇
?	?	?	A ₀	A ₀	A ₀	A ₀	A ₀	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁
?	?	C ₂	C ₂	C ₂	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆
?	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	D ₃	D ₃	D ₃	D ₃	D ₃	D ₃	C ₂	C ₂	C ₂

HIT!

Time

4 hits, 12 misses

Disk

7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A

Aging PRA

Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Ages

0	1	1	0	1	1
1	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	0

RAM

?	?	?	?	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇
?	?	?	A ₀	A ₀	A ₀	A ₀	A ₀	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁	A ₀	A ₀
?	?	C ₂	C ₂	C ₂	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆
?	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	D ₃	D ₃	D ₃	D ₃	D ₃	C ₂	C ₂	C ₂

Time

Disk

7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A

4 hits, 12 misses

Aging PRA

Time Step: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Time

Access Stream: 5 2 0 7 6 7 3 1 7 6 2 7 0 5 4 3

Ages

011011
100000
100000
010000

RAM

?	?	?	?	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇
?	?	?	A ₀	A ₀	A ₀	A ₀	A ₀	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁	A ₀	A ₀	A ₀
?	?	C ₂	C ₂	C ₂	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	F ₅
?	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	D ₃	D ₃	D ₃	D ₃	D ₃	C ₂	C ₂	C ₂	C ₂

Time

Disk

H ₇
G ₆
F ₅
E ₄
D ₃
C ₂
B ₁
A ₀

4 hits, 12 misses

Aging PRA

Time Step: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Time

Access Stream: 5 2 0 7 6 7 3 1 7 6 2 7 0 5 4 3

Ages

001101
010000
010000
100000

RAM

?	?	?	?	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	
?	?	?	A ₀	A ₀	A ₀	A ₀	A ₀	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁	A ₀	A ₀	A ₀	
?	?	C ₂	C ₂	C ₂	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	F ₅	F ₅	F ₅	
?	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	D ₃	D ₃	D ₃	D ₃	D ₃	C ₂	C ₂	C ₂	C ₂	E ₄

Time

Disk

H ₇
G ₆
F ₅
E ₄
D ₃
C ₂
B ₁
A ₀

4 hits, 12 misses

Aging PRA

Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Ages

100000
010000
010000
100000

RAM

?	?	?	?	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	D ₃
?	?	?	A ₀	A ₀	A ₀	A ₀	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁	A ₀	A ₀	A ₀	A ₀
?	?	C ₂	C ₂	C ₂	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	F ₅	F ₅	F ₅
?	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	D ₃	D ₃	D ₃	D ₃	C ₂	C ₂	C ₂	C ₂	E ₄	E ₄

Time

Disk

7 H
6 G
5 F
4 E
3 D
2 C
1 B
0 A

4 hits, 12 misses

Aging PRA

Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Ages

100000
010000
010000
100000

RAM

?	?	?	?	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	D ₃
?	?	?	A ₀	A ₀	A ₀	A ₀	A ₀	B ₁	B ₁	B ₁	B ₁	B ₁	B ₁	A ₀	A ₀	A ₀
?	?	C ₂	C ₂	C ₂	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	G ₆	F ₅	F ₅	F ₅
?	F ₅	F ₅	F ₅	F ₅	F ₅	F ₅	D ₃	D ₃	D ₃	D ₃	C ₂	C ₂	C ₂	C ₂	E ₄	E ₄

Time

4 hits, 12 misses

Disk

H ₇
G ₆
F ₅
E ₄
D ₃
C ₂
B ₁
A ₀



- No age difference between pages in same aging cycle



- No age difference between pages in same aging cycle
- Limited number of bits:

- No age difference between pages in same aging cycle
- Limited number of bits:
 - if counter = 0, no difference if unused since 10 or 100 ticks

- No age difference between pages in same aging cycle
 - Limited number of bits:
 - if counter = 0, no difference if unused since 10 or 100 ticks
- more bits is better (but also uses more space)




- So far we completely ignored processes...


- So far we completely ignored processes...
- Can we measure how **fair** PRAs are (wrt. processes)?




- So far we completely ignored processes...
- Can we measure how **fair** PRAs are (wrt. processes)?
 - Process performance? Difficult to compare...



- So far we completely ignored processes...
- Can we measure how **fair** PRAs are (wrt. processes)? 
 - Process performance? Difficult to compare...
 - Same amount of memory for every process? Tiny shell vs. 3D game

- So far we completely ignored processes...
- Can we measure how **fair** PRAs are (wrt. processes)? 
 - Process performance? Difficult to compare...
 - Same amount of memory for every process? Tiny shell vs. 3D game→ Same page faults per second (= page fault frequency)!

- So far we completely ignored processes...
- Can we measure how **fair** PRAs are (wrt. processes)? 
 - Process performance? Difficult to compare...
 - Same amount of memory for every process? Tiny shell vs. 3D game
→ Same page faults per second (= page fault frequency)!
- How do we make every process have the same number of page faults per second?

- Thrashing: system deals more with page faults and swapping than with work




- Thrashing: system deals more with page faults and swapping than with work




- Processes need more RAM than exists : always too many page faults




- Thrashing: system deals more with page faults and swapping than with work
- Processes need more RAM than exists : always too many page faults
- Page fault frequency too high?




- Thrashing: system deals more with page faults and swapping than with work
- Processes need more RAM than exists : always too many page faults
- Page fault frequency too high?




- Thrashing: system deals more with page faults and swapping than with work
- Processes need more RAM than exists : always too many page faults
- Page fault frequency too high? → not enough RAM



- Thrashing: system deals more with page faults and swapping than with work
 - Processes need more RAM than exists : always too many page faults
 - Page fault frequency too high? → not enough RAM
- Swap out entire processes until page fault frequency decreases



- Thrashing: system deals more with page faults and swapping than with work
 - Processes need more RAM than exists : always too many page faults
 - Page fault frequency too high? → not enough RAM
- Swap out entire processes until page fault frequency decreases
- Only schedule processes where all required pages are in RAM





Peter Denning, 1968, abbreviated:

We define the working set $W(t, \tau)$ of a process at time t to be the collection of information referenced by the process during the process time interval $(t - \tau, t)$.



Peter Denning, 1968, abbreviated:

We define the working set $W(t, \tau)$ of a process at time t to be the collection of information referenced by the process during the process time interval $(t - \tau, t)$.

Peter Denning, 1968, abbreviated:

We define the working set $W(t, \tau)$ of a process at time t to be the collection of information referenced by the process during the process time interval $(t - \tau, t)$.

- $\tau =$ the *working set parameter*

Peter Denning, 1968, abbreviated:

We define the working set $W(t, \tau)$ of a process at time t to be the collection of information referenced by the process during the process time interval $(t - \tau, t)$.

- τ = the *working set parameter*
- $\omega(t, \tau)$ = number of pages in $W(t, \tau)$

Peter Denning, 1968, abbreviated:

We define the working set $W(t, \tau)$ of a process at time t to be the collection of information referenced by the process during the process time interval $(t - \tau, t)$.

- τ = the *working set parameter*
- $\omega(t, \tau)$ = number of pages in $W(t, \tau)$

Peter Denning, 1968, abbreviated:

We define the working set $W(t, \tau)$ of a process at time t to be the collection of information referenced by the process during the process time interval $(t - \tau, t)$.

- τ = the *working set parameter*
- $\omega(t, \tau)$ = number of pages in $W(t, \tau)$

More ideas:

- Prepaging: preload all pages in the working set before scheduling

Peter Denning, 1968, abbreviated:

We define the working set $W(t, \tau)$ of a process at time t to be the collection of information referenced by the process during the process time interval $(t - \tau, t)$.

- τ = the *working set parameter*
- $\omega(t, \tau)$ = number of pages in $W(t, \tau)$

More ideas:

- Prepaging: preload all pages in the working set before scheduling
- PRA: only swap pages which are in no working set

Peter Denning, 1968, abbreviated:

We define the working set $W(t, \tau)$ of a process at time t to be the collection of information referenced by the process during the process time interval $(t - \tau, t)$.

- τ = the *working set parameter*
- $\omega(t, \tau)$ = number of pages in $W(t, \tau)$

More ideas:

- Prepaging: preload all pages in the working set before scheduling
- PRA: only swap pages which are in no working set
- Adaptive τ !

Peter Denning, 1968, abbreviated:

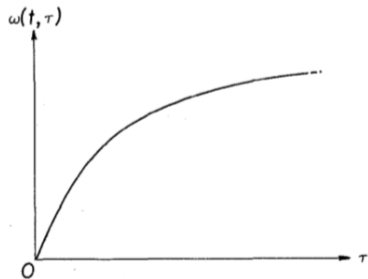
We define the working set $W(t, \tau)$ of a process at time t to be the collection of information referenced by the process during the process time interval $(t - \tau, t)$.

- $\tau =$ the *working set parameter*
- $\omega(t, \tau) =$ number of pages in $W(t, \tau)$

More ideas:

- Prepaging: preload all pages in the working set before scheduling
- PRA: only swap pages which are in no working set
- Adaptive τ !

Behavior of $\omega(t, \tau)$:



Define working set size by

- Time: All pages younger than τ are in the working set. (suggested by Denning)

Define working set size by

- Time: All pages younger than τ are in the working set. (suggested by Denning)
- Huge Shift Register: shift in page number upon access. (difficult to implement)

Define working set size by

- Time: All pages younger than τ are in the working set. (suggested by Denning)
- Huge Shift Register: shift in page number upon access. (difficult to implement)
- Page Count: The N youngest pages are in the working set.

Define working set size by

- Time: All pages younger than τ are in the working set. (suggested by Denning)
- Huge Shift Register: shift in page number upon access. (difficult to implement)
- Page Count: The N youngest pages are in the working set.

Define working set size by

- Time: All pages younger than τ are in the working set. (suggested by Denning)
- Huge Shift Register: shift in page number upon access. (difficult to implement)
- Page Count: The N youngest pages are in the working set.

Page fault frequency too high?

Define working set size by

- Time: All pages younger than τ are in the working set. (suggested by Denning)
- Huge Shift Register: shift in page number upon access. (difficult to implement)
- Page Count: The N youngest pages are in the working set.

Page fault frequency too high?

- Globally reduce τ , or the size of the shift register, or N respectively



- Prepaging not common



- Prepaging not common
- Working Set is no PRA ...



- Prepaging not common
- Working Set is no PRA ...
- ... but commonly used to form a process-aware PRA

- Prepaging not common
- Working Set is no PRA ...
- ... but commonly used to form a process-aware PRA
- Same approximations as in other PRAs:

- Prepaging not common
- Working Set is no PRA ...
- ... but commonly used to form a process-aware PRA
- Same approximations as in other PRAs:
 - polling referenced bits



- Prepaging not common
- Working Set is no PRA ...
- ... but commonly used to form a process-aware PRA
- Same approximations as in other PRAs:
 - polling referenced bits
 - storing information in a global data structure





Working Set:



Working Set:

- Every process has a working set size N



Working Set:

- Every process has a working set size N
- Every process has M mapped pages



Working Set:

- Every process has a working set size N
- Every process has M mapped pages
- Each page has a timestamp

Working Set:

- Every process has a working set size N
- Every process has M mapped pages
- Each page has a timestamp
 - not real time, process time, `clock()`



Working Set:

- Every process has a working set size N
- Every process has M mapped pages
- Each page has a timestamp
 - not real time, process time, `clock()`
- Working Set: The N youngest pages

Working Set:

- Every process has a working set size N
- Every process has M mapped pages
- Each page has a timestamp
 - not real time, process time, `clock()`
- Working Set: The N youngest pages

Working Set:

- Every process has a working set size N
- Every process has M mapped pages
- Each page has a timestamp
 - not real time, process time, `clock()`
- Working Set: The N youngest pages

Process-aware PRA:

- Any page in **no** working set (of any process) is **swappable**

Working Set:

- Every process has a working set size N
- Every process has M mapped pages
- Each page has a timestamp
 - not real time, process time, `clock()`
- Working Set: The N youngest pages

Process-aware PRA:

- Any page in **no** working set (of any process) is **swappable**
- Use global PRA on **swappable** pages

Working Set:

- Every process has a working set size N
- Every process has M mapped pages
- Each page has a timestamp
 - not real time, process time, `clock()`
- Working Set: The N youngest pages

Process-aware PRA:

- Any page in **no** working set (of any process) is **swappable**
- Use global PRA on **swappable** pages
 - e.g., Clock \rightarrow WSClock

Working Set:

- Every process has a working set size N
- Every process has M mapped pages
- Each page has a timestamp
 - not real time, process time, `clock()`
- Working Set: The N youngest pages

Adaptive process-aware PRA:

Process-aware PRA:

- Any page in **no** working set (of any process) is **swappable**
- Use global PRA on **swappable** pages
 - e.g., Clock \rightarrow WSClock

Working Set:

- Every process has a working set size N
- Every process has M mapped pages
- Each page has a timestamp
 - not real time, process time, `clock()`
- Working Set: The N youngest pages

Process-aware PRA:

- Any page in **no** working set (of any process) is **swappable**
- Use global PRA on **swappable** pages
 - e.g., Clock \rightarrow WSClock

Adaptive process-aware PRA:

- Update N upon certain occasions

Working Set:

- Every process has a working set size N
- Every process has M mapped pages
- Each page has a timestamp
 - not real time, process time, `clock()`
- Working Set: The N youngest pages

Process-aware PRA:

- Any page in **no** working set (of any process) is **swappable**
- Use global PRA on **swappable** pages
 - e.g., Clock \rightarrow WSClock

Adaptive process-aware PRA:

- Update N upon certain occasions
- Set $N = N - 1$ for **all** processes to reduce memory pressure

Working Set:

- Every process has a working set size N
- Every process has M mapped pages
- Each page has a timestamp
 - not real time, process time, `clock()`
- Working Set: The N youngest pages

Process-aware PRA:

- Any page in **no** working set (of any process) is **swappable**
- Use global PRA on **swappable** pages
 - e.g., Clock \rightarrow WSClock

Adaptive process-aware PRA:

- Update N upon certain occasions
- Set $N = N - 1$ for **all** processes to reduce memory pressure
 - e.g. when trying to swap a page but none are swappable

Working Set:

- Every process has a working set size N
- Every process has M mapped pages
- Each page has a timestamp
 - not real time, process time, `clock()`
- Working Set: The N youngest pages

Process-aware PRA:

- Any page in **no** working set (of any process) is **swappable**
- Use global PRA on **swappable** pages
 - e.g., Clock \rightarrow WSClock

Adaptive process-aware PRA:

- Update N upon certain occasions
- Set $N = N - 1$ for **all** processes to reduce memory pressure
 - e.g. when trying to swap a page but none are swappable
- Set $N = N + 1$ for a process P to adjust for increasing memory usage

Working Set:

- Every process has a working set size N
- Every process has M mapped pages
- Each page has a timestamp
 - not real time, process time, `clock()`
- Working Set: The N youngest pages

Process-aware PRA:

- Any page in **no** working set (of any process) is **swappable**
- Use global PRA on **swappable** pages
 - e.g., Clock \rightarrow WSClock

Adaptive process-aware PRA:

- Update N upon certain occasions
- Set $N = N - 1$ for **all** processes to reduce memory pressure
 - e.g. when trying to swap a page but none are swappable
- Set $N = N + 1$ for a process P to adjust for increasing memory usage
 - e.g. when P experiences a pagefault


Working Set:

- Every process has a working set size N
- Every process has M mapped pages
- Each page has a timestamp
 - not real time, process time, `clock()`
- Working Set: The N youngest pages

Process-aware PRA:

- Any page in **no** working set (of any process) is **swappable**
- Use global PRA on **swappable** pages
 - e.g., Clock \rightarrow WSClock

Adaptive process-aware PRA:

- Update N upon certain occasions
 - Set $N = N - 1$ for **all** processes to reduce memory pressure
 - e.g. when trying to swap a page but none are swappable
 - Set $N = N + 1$ for a process P to adjust for increasing memory usage
 - e.g. when P experiences a pagefault
- \rightarrow Page fault frequency will settle to the same value for every process 


Working Set:

- Every process has a working set **parameter τ**
- Every process has M mapped pages
- Each page has a timestamp
 - not real time, process time, `clock()`
- Working Set: **All pages younger than τ**

Process-aware PRA:

- Any page in **no** working set (of any process) is **swappable**
- Use global PRA on **swappable** pages
 - e.g., Clock \rightarrow WSClock

Adaptive process-aware PRA:

- Update τ upon certain occasions
 - **Decrease τ slightly** to reduce memory pressure
 - e.g. when trying to swap a page but none are swappable
 - **Increase τ slightly** to adjust for increasing memory usage
 - e.g. when P experiences a pagefault
- \rightarrow Page fault frequency will settle to the same value for every process 

Time Step:

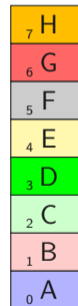
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

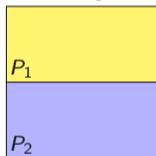
Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

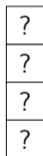
Disk



Working Set



RAM



Time

P₁: 1 hits, 7 misses P₂: 2 hits, 6 misses

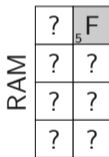
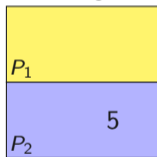
Time Step: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Time

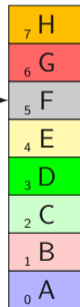
Access Stream: 5 2 0 7 6 7 3 1 7 6 2 7 0 5 4 3

Disk

Working Set



RAM



Time

P_1 : 1 hits, 7 misses P_2 : 2 hits, 6 misses

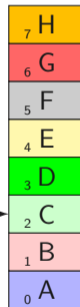
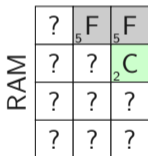
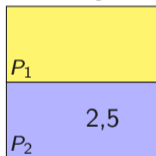
Time Step: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

Time

Access Stream: 5 2 0 7 6 7 3 1 7 6 2 7 0 5 4 3

Disk

Working Set



Time

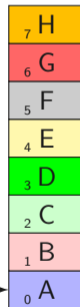
P_1 : 1 hits, 7 misses P_2 : 2 hits, 6 misses

Time Step: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

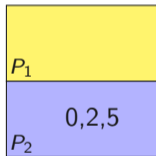
Time

Access Stream: 5 2 0 7 6 7 3 1 7 6 2 7 0 5 4 3

Disk



Working Set



Time

P_1 : 1 hits, 7 misses P_2 : 2 hits, 6 misses

Time Step:

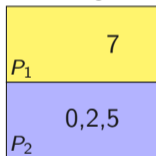
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

Access Stream:

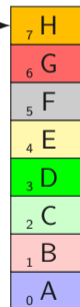
5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Working Set



Time

Disk



P_1 : 1 hits, 7 misses P_2 : 2 hits, 6 misses

Time Step:

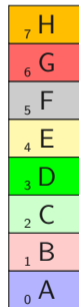
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

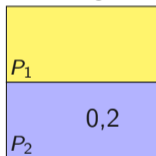
Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Disk



Working Set



Time

P₁: 1 hits, 7 misses P₂: 2 hits, 6 misses

Time Step:

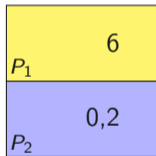
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

Access Stream:

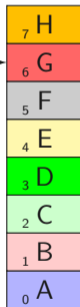
5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Working Set



Time

Disk



P_1 : 1 hits, 7 misses P_2 : 2 hits, 6 misses

Time Step:

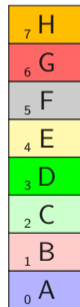
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

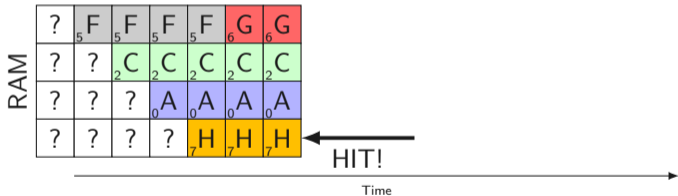
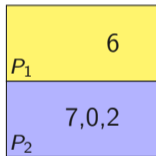
Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Disk



Working Set



P_1 : 1 hits, 7 misses P_2 : 2 hits, 6 misses

Time Step:

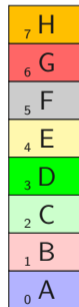
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

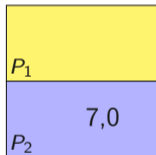
Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Disk



Working Set



Time

P₁: 1 hits, 7 misses

P₂: 2 hits, 6 misses

Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

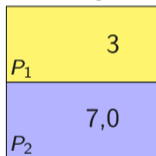
Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Disk



Working Set



Time

P_1 : 1 hits, 7 misses

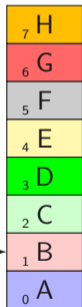
P_2 : 2 hits, 6 misses

Time Step: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

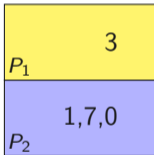
Time

Access Stream: 5 2 0 7 6 7 3 1 7 6 2 7 0 5 4 3

Disk



Working Set



Time

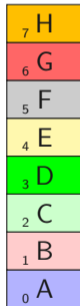
P_1 : 1 hits, 7 misses P_2 : 2 hits, 6 misses

Time Step: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

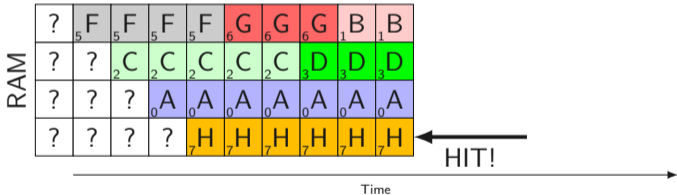
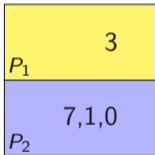
Time

Access Stream: 5 2 0 7 6 7 3 1 7 6 2 7 0 5 4 3

Disk



Working Set



P_1 : 1 hits, 7 misses P_2 : 2 hits, 6 misses

Time Step:

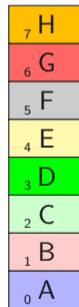
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

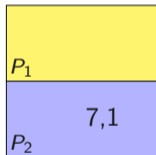
Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Disk



Working Set



Time

P₁: 1 hits, 7 misses P₂: 2 hits, 6 misses

Time Step:

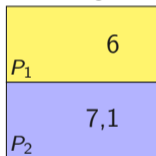
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Working Set

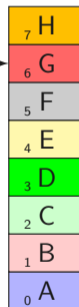


RAM



Time

Disk



P_1 : 1 hits, 7 misses P_2 : 2 hits, 6 misses

Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Working Set

P_1	2,6
P_2	7,1

RAM

	?	F ₅	F ₅	F ₅	F ₅	G ₆	G ₆	G ₆	B ₁	B ₁	B ₁	B ₁
?	?	C ₂	C ₂	C ₂	C ₂	D ₃	D ₃	D ₃	G ₆	G ₆		
?	?	?	A ₀	A ₀	A ₀	A ₀	A ₀	A ₀	A ₀	A ₀	C ₂	
?	?	?	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇	H ₇

Time

Disk

7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A

P_1 : 1 hits, 7 misses P_2 : 2 hits, 6 misses

Time Step:

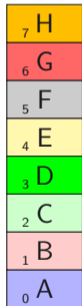
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

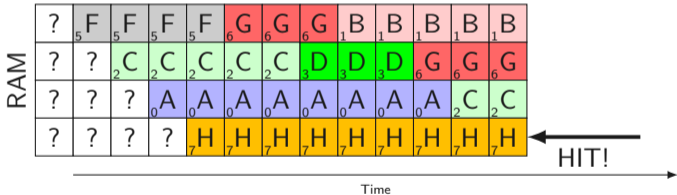
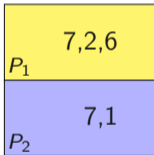
Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Disk



Working Set



P_1 : 1 hits, 7 misses P_2 : 2 hits, 6 misses

Time Step:

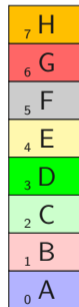
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

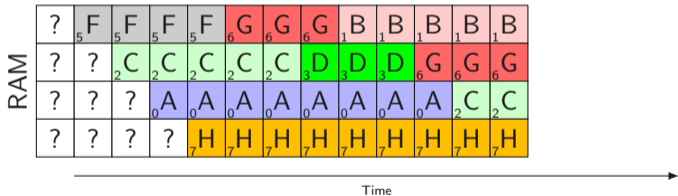
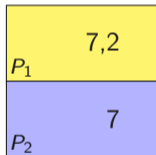
Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Disk



Working Set



P_1 : 1 hits, 7 misses P_2 : 2 hits, 6 misses

Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Working Set

P_1	7,2
P_2	0,7

RAM	?	₅ F	₅ F	₅ F	₅ F	₆ G	₆ G	₆ G	₁ B	₁ B	₁ B	₁ B	₁ B	₁ B	₁ B
	?	₂ C	₂ C	₂ C	₂ C	₃ D	₃ D	₃ D	₆ G	₆ G	₆ G	₆ G	₀ A		
	?	?	₀ A	₀ A	₀ A	₀ A	₀ A	₀ A	₀ A	₀ A	₂ C	₂ C	₂ C		
	?	?	?	₇ H	₇ H	₇ H	₇ H	₇ H	₇ H	₇ H	₇ H	₇ H	₇ H	₇ H	₇ H

Time

Disk

7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A

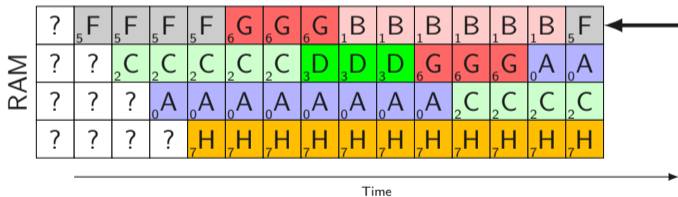
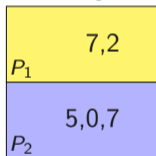
P_1 : 1 hits, 7 misses P_2 : 2 hits, 6 misses

Time Step: 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16

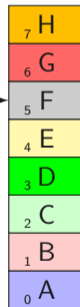
Time

Access Stream: 5 2 0 7 6 7 3 1 7 6 2 7 0 5 4 3

Working Set



Disk



P_1 : 1 hits, 7 misses P_2 : 2 hits, 6 misses

Time Step:

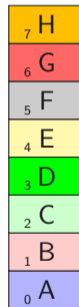
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

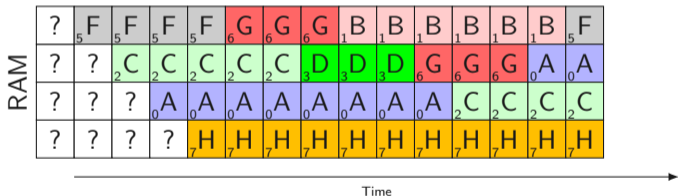
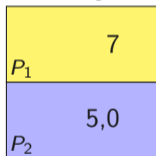
Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Disk



Working Set



P_1 : 1 hits, 7 misses P_2 : 2 hits, 6 misses

Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Working Set

P_1	4,7
P_2	5,0

RAM	?	₅ F	₅ F	₅ F	₅ F	₆ G	₆ G	₆ G	₁ B	₁ B	₁ B	₁ B	₁ B	₅ F	₅ F
	?	?	₂ C	₂ C	₂ C	₂ C	₃ D	₃ D	₃ D	₆ G	₆ G	₆ G	₀ A	₀ A	₀ A
	?	?	?	₀ A	₀ A	₀ A	₀ A	₀ A	₀ A	₀ A	₂ C	₂ C	₂ C	₂ C	₄ E
	?	?	?	?	₇ H	₇ H	₇ H	₇ H	₇ H	₇ H	₇ H	₇ H	₇ H	₇ H	₇ H

Time

Disk

7	H
6	G
5	F
4	E
3	D
2	C
1	B
0	A

P_1 : 1 hits, 7 misses P_2 : 2 hits, 6 misses

Time Step:

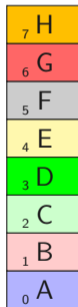
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

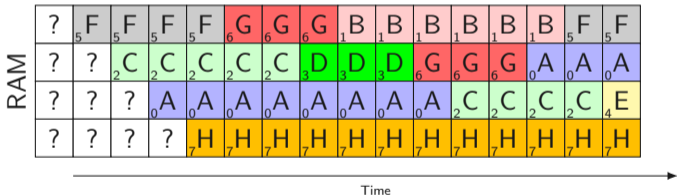
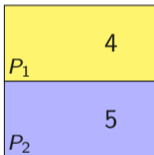
Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Disk



Working Set



Time

P_1 : 1 hits, 7 misses P_2 : 2 hits, 6 misses

Time Step:

1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

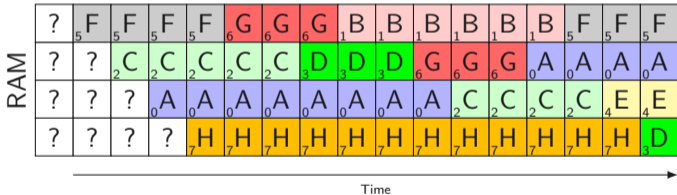
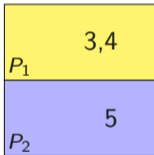
Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Disk



Working Set



Time

P_1 : 1 hits, 7 misses P_2 : 2 hits, 6 misses

Time Step:

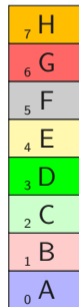
1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
---	---	---	---	---	---	---	---	---	----	----	----	----	----	----	----

Time

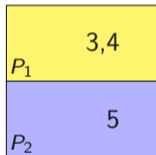
Access Stream:

5	2	0	7	6	7	3	1	7	6	2	7	0	5	4	3
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

Disk



Working Set



Time

P_1 : 1 hit, 7 misses

P_2 : 2 hits, 6 misses






PRA selects page for eviction ...

PRA selects page for eviction ...



- **local:** ... from the same process





PRA selects page for eviction ...

- **local:** ... from the same process 
 - Is the working set size fixed or adaptive?



PRA selects page for eviction ...

- **local:** ... from the same process 
 - Is the working set size fixed or adaptive?
- **global:** ... from any process 

PRA selects page for eviction ...


- **local:** ... from the same process 
 - Is the working set size fixed or adaptive?
- **global:** ... from any process 


PRA selects page for eviction ...

- **local:** ... from the same process 
 - Is the working set size fixed or adaptive?
- **global:** ... from any process 

Working set algorithms are inherently global

PRA selects page for eviction ...


- **local:** ... from the same process 
 - Is the working set size fixed or adaptive?


- **global:** ... from any process 

Working set algorithms are inherently global

Global strategies usually perform better:

PRA selects page for eviction ...

- **local:** ... from the same process 
 - Is the working set size fixed or adaptive?


- **global:** ... from any process 


Working set algorithms are inherently global

Global strategies usually perform better:

- Process needs more pages:

PRA selects page for eviction ...

- **local:** ... from the same process 
 - Is the working set size fixed or adaptive?



- **global:** ... from any process 

Working set algorithms are inherently global

Global strategies usually perform better:

- Process needs more pages:
 - Thrashing although other processes might have spare pages

PRA selects page for eviction ...


- **local:** ... from the same process 
 - Is the working set size fixed or adaptive?
- **global:** ... from any process 


Working set algorithms are inherently global

Global strategies usually perform better:

- Process needs more pages:
 - Thrashing although other processes might have spare pages
- Process needs fewer pages:

PRA selects page for eviction ...

- **local:** ... from the same process 
 - Is the working set size fixed or adaptive?

- **global:** ... from any process 

Working set algorithms are inherently global

Global strategies usually perform better:

- Process needs more pages:
 - Thrashing although other processes might have spare pages
- Process needs fewer pages:
 - Memory waste despite possible thrashing in another processes

- Page allocation latency crucial for performance

- Page allocation latency crucial for performance
- Bad Latency when going through a lot of steps:

- Page allocation latency crucial for performance
- Bad Latency when going through a lot of steps:
 1. No free physical page

- Page allocation latency crucial for performance
- Bad Latency when going through a lot of steps:
 1. No free physical page
 2. No clean pages

- Page allocation latency crucial for performance
- Bad Latency when going through a lot of steps:
 1. No free physical page
 2. No clean pages
 3. Swap out page (wait for disk)

- Page allocation latency crucial for performance
- Bad Latency when going through a lot of steps:
 1. No free physical page
 2. No clean pages
 3. Swap out page (wait for disk)
 4. Return released page to user

- Page allocation latency crucial for performance
- Bad Latency when going through a lot of steps:
 1. No free physical page
 2. No clean pages
 3. Swap out page (wait for disk)
 4. Return released page to user
- Better: don't let it get this far

- Page allocation latency crucial for performance
- Bad Latency when going through a lot of steps:
 1. No free physical page
 2. No clean pages
 3. Swap out page (wait for disk)
 4. Return released page to user
- Better: don't let it get this far
 - How realistic is that?



Some classes are cheaper for swapping than others:

Some classes are cheaper for swapping than others:

Class	Referenced	Dirty	Properties
0	0	0	Not used in a while and not modified → just evict

A Paging Daemon doing Pre-Swapping

A Paging Daemon doing Pre-Swapping

- Paging Daemon mostly inactive

A Paging Daemon doing Pre-Swapping

- Paging Daemon mostly inactive
- Checks regularly: Evictable/unused page frames below threshold?

A Paging Daemon doing Pre-Swapping

- Paging Daemon mostly inactive
- Checks regularly: Evictable/unused page frames below threshold?
 - Swap a dirty page

A Paging Daemon doing Pre-Swapping

- Paging Daemon mostly inactive
- Checks regularly: Evictable/unused page frames below threshold?
 - Swap a dirty page
 - Keep it in RAM

A Paging Daemon doing Pre-Swapping

- Paging Daemon mostly inactive
- Checks regularly: Evictable/unused page frames below threshold?
 - Swap a dirty page
 - Keep it in RAM
 - Set dirty-bit to 0

A Paging Daemon doing Pre-Swapping

- Paging Daemon mostly inactive
- Checks regularly: Evictable/unused page frames below threshold?
 - Swap a dirty page
 - Keep it in RAM
 - Set dirty-bit to 0


→ Pre-swapped pages are evictable pages

A Paging Daemon doing Pre-Swapping

- Paging Daemon mostly inactive
 - Checks regularly: Evictable/unused page frames below threshold?
 - Swap a dirty page
 - Keep it in RAM
 - Set dirty-bit to 0
- Pre-swapped pages are evictable pages
- Evictable pages are as good as unused pages (performance-wise)



Maybe a page is required to stay in RAM?

Maybe a page is required to stay in RAM? → Pinning 

Scenario:

1. A process requests I/O (e.g. `read(FD, buffer, nrBytes)`) and blocks

Maybe a page is required to stay in RAM? → Pinning



Scenario:

1. A process requests I/O (e.g. `read(FD, buffer, nrBytes)`) and blocks
2. Other processes raise page faults

Maybe a page is required to stay in RAM? → Pinning



Scenario:

1. A process requests I/O (e.g. `read(FD, buffer, nrBytes)`) and blocks
2. Other processes raise page faults
 - This might replace the destination page

Maybe a page is required to stay in RAM? → Pinning



Scenario:

1. A process requests I/O (e.g. `read(FD, buffer, nrBytes)`) and blocks
2. Other processes raise page faults
 - This might replace the destination page→ DMA transfer would go to wrong location

Maybe a page is required to stay in RAM? → Pinning



Scenario:

1. A process requests I/O (e.g. `read(FD, buffer, nrBytes)`) and blocks
2. Other processes raise page faults
 - This might replace the destination page→ DMA transfer would go to wrong location

Maybe a page is required to stay in RAM? → Pinning



Scenario:

1. A process requests I/O (e.g. `read(FD, buffer, nrBytes)`) and blocks
2. Other processes raise page faults
 - This might replace the destination page
 - DMA transfer would go to wrong location

Avoiding this scenario:

Maybe a page is required to stay in RAM? → Pinning



Scenario:

1. A process requests I/O (e.g. `read(FD, bufferm, nrBytes)`) and blocks
2. Other processes raise page faults
 - This might replace the destination page
 - DMA transfer would go to wrong location

Avoiding this scenario:

→ Page must be locked in memory (= excluded from PRA)

Maybe a page is required to stay in RAM? → Pinning



Scenario:

1. A process requests I/O (e.g. `read(FD, bufferm, nrBytes)`) and blocks
2. Other processes raise page faults
 - This might replace the destination page
 - DMA transfer would go to wrong location

Avoiding this scenario:

- Page must be locked in memory (= excluded from PRA)
- Alternatively: use (non-evictable) kernel buffers

