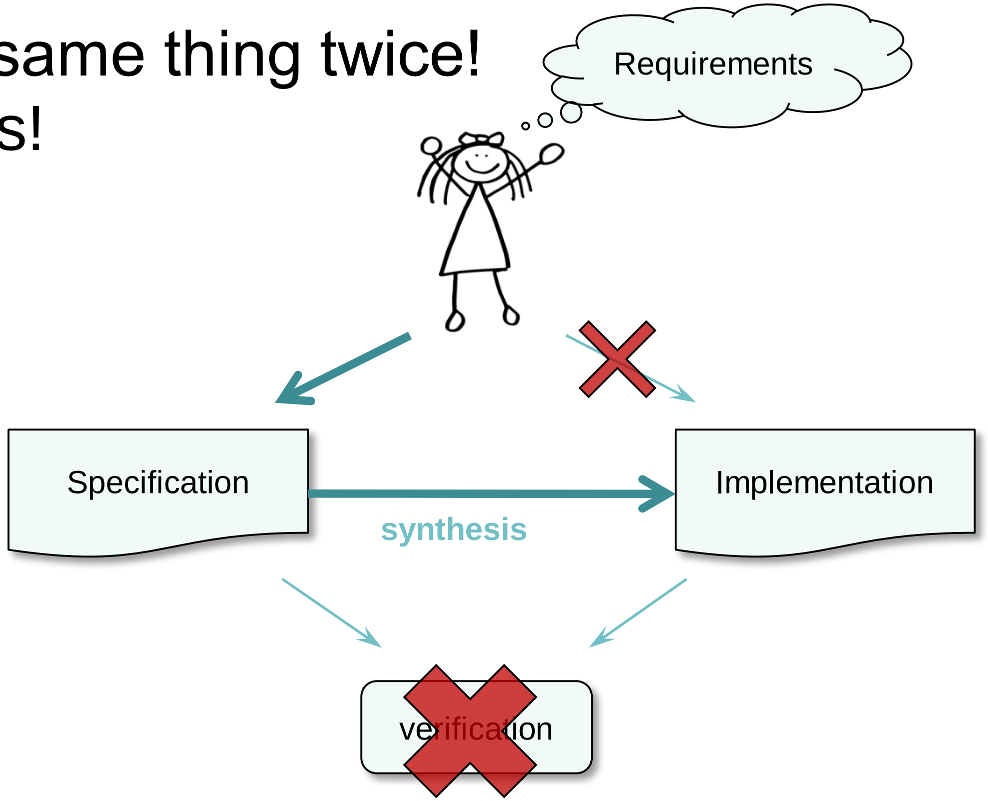
# Synthesis

# Construct Correct Systems Automatically

Don’t do the same thing twice!
Use synthesis!




**reactive synthesis**

# What Theory Will We Use?

- Games
- Automata
- Logic

# Games Examples

- A new Cola market?
  - Coke is ahead of Pepsi – makes first decision
  - Market cannot bear two competitors
  - Assumption: game is not repeated
  - Payoff matrix (Coke profit / Pepsi profit)

| | Coke enters | No Coke |
|---|---|---|
| Pepsi | ~~-5/-~~ | 0/20* |
| No Pepsi | 10/0* | ~~0/0~~ |

- **Who will enter the market?**

coke enters
pepsi doesn't

In synthesis, we will look at *graph games*

# Games Examples

- A new Cola market?
  - Coke is ahead of Pepsi – makes first decision
  - Market cannot bear two competitors
  - Assumption: game is not repeated
  - Payoff matrix (Coke profit / Pepsi profit)

| | Coke | No Coke |
|---|---|---|
| Pepsi | -5/-5 | 0/20 |
| No Pepsi | 10/0 | 0/0 |

  - **Who will enter the market?**

In synthesis, we will look at *graph games*

# Prisoners Dilemma

P1 confess p1 deny

P2 conf

p2 deny

| | Player 1 Confess | Player 1 Deny |
|---|---|---|
| **Player 2 Confess** | -6/-6 | 0/-10 |
| **Player 2 Deny** | -10/0 | ~~1/1~~ -1/-1 |

P1 payoff / P2 payoff

| | | P1 | |
|---|---|---|---|
| | | conf | deny |
| P2 | conf | -6/-6 | -10/0 |
| | deny | 0/-10 | -1/-10 |

monkey cucumber.

# Satisfiability & Realizability

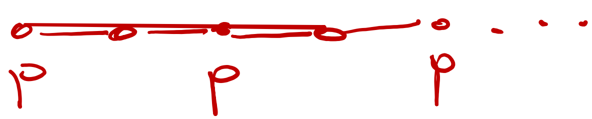
**Two problems:**

1. **Satisfiability**: Is there a *trace* that satisfies spec?
2. **Realizability**: Is there a *system* that satisfies spec?

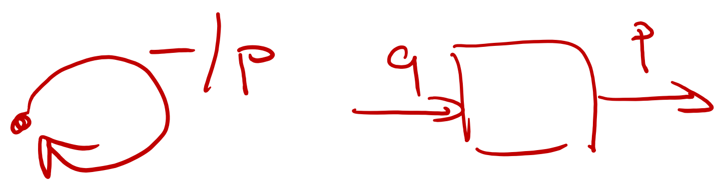
$\varphi = GF p$

$\varphi$ LTL

satisfiable:




realizable



# Satisfiability & Realizability

Satisfiability: Is there a trace that satisfies the spec?

Realizability: Is there a system that satisfies the spec?

Realizability $\neq$ Satisfiability

**always( (req1 → grant1) ∧ (req2 → grant2))**

**never ( grant1 ∧ grant2 )**

φ = G ¬

π =

| | 0 | 1 | 2 | 3 |
|---|---|---|---|---|
| r1 | F | F | F | F |
| g1 | F | F | F | F |
| r2 | F | F | F | F |
| g2 | F | F | F | F |

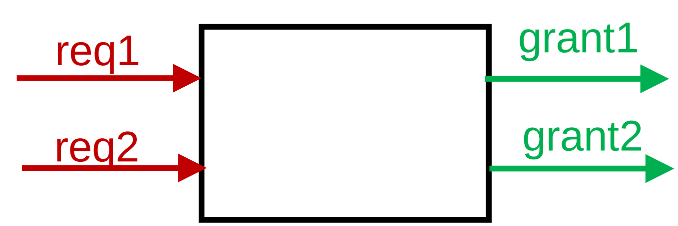
$\pi \models \phi$

SATISFIABLE

Satisfiable? Realizable?

Satisfiable, but functionally impossible!

Distinguish inputs from outputs!



# Satisfiability & Realizability

Satisfiability: Is there a trace that satisfies the spec?

Realizability: Is there a system that satisfies the spec?

Realizability $\neq$ Satisfiability

$\varphi$ **always( ~~grant1~~ ↔ next req1 )**

grant1

| | 0 | 1 | . . . |
|---|---|---|---|
| g1 | F | F | |
| r1 | F | F | |

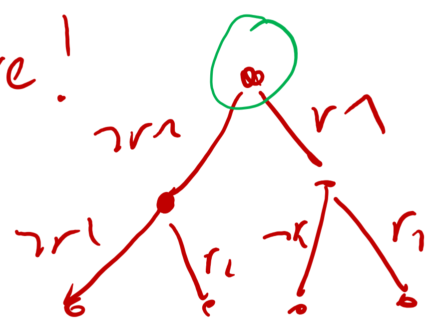
$\forall$ input sequence $\exists$ outseq $\models \varphi$

Satisfiable? Realizable?

Satisfiable, but not realizable: clairvoyant!

Take time into account!

not realizable! looks into the future!




# Formal Verification

Given:

System provides outputs

A specification


One Player: (not a game!)

- Environment provides inputs

System is good if it fulfills the spec **for all possible inputs**

# Synthesis is a Game

Given:

~~System provides outputs~~

A specification


**Two** Players (a game!)

- Environment provides inputs
- **System provides outputs**

System is good if it fulfills the spec **for all possible inputs**

# Our Setting

## Reactive Systems

- Constant interaction
- No Termination
- E.g. Cell phones, Operating Systems, Powerpoint

## Finite State


Non-terminating, finite systems are graphs with loops

- Not our focus: functions
  - “Create a function that computes sqrt(2)”

# Example I: Chess


- Environment determines black moves
- System determines white movers
- Winning condition:
  - If all black moves are legal, then all white moves are legal and eventually, white reaches checkmate

Easy to specify!

# Checkers and Systems

**Checkers** are passive

Judge if given behavior is allowed (satisfiability)

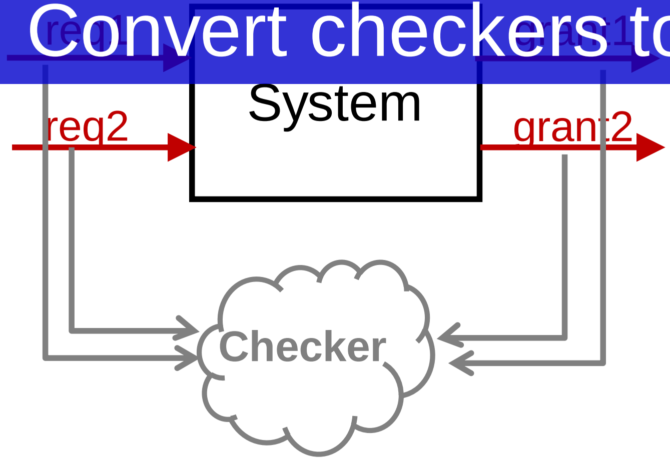
Used in verification

**Systems** are active

Construct correct behavior (realizability)

Result of synthesis

Synthesis: Convert checkers to systems




# Synthesis

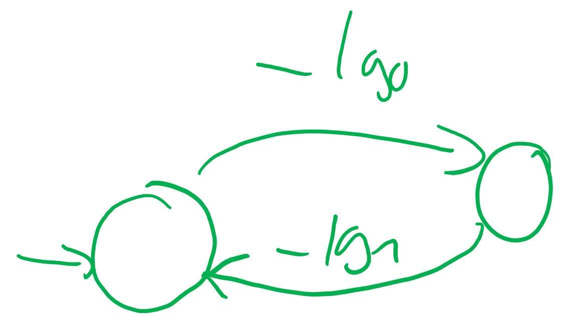
1. Specify
2. Create Game
3. Solve Game
4. Create System

# Example II: Arbiter

1. **Specify**
2. Create Game
3. Solve Game
4. Create System


Input: r0, r1

Output: g0, g1




**What is the specification?**

$\boxed{G(r_0 \to F g_0)}$

$G(r_1 \to F g_1)$

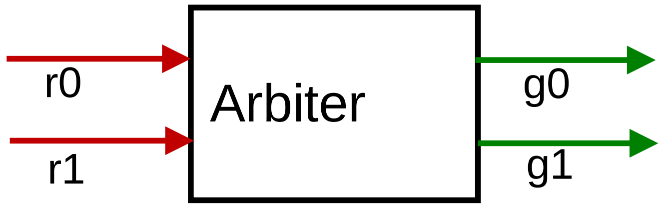
$G \neg(g_0 \wedge g_1)$

"grants come relatively quickly"

"no unnecessary grants"

# Example II: Arbiter

1. **Specify**
2. Create Game
3. Solve Game
4. Create System



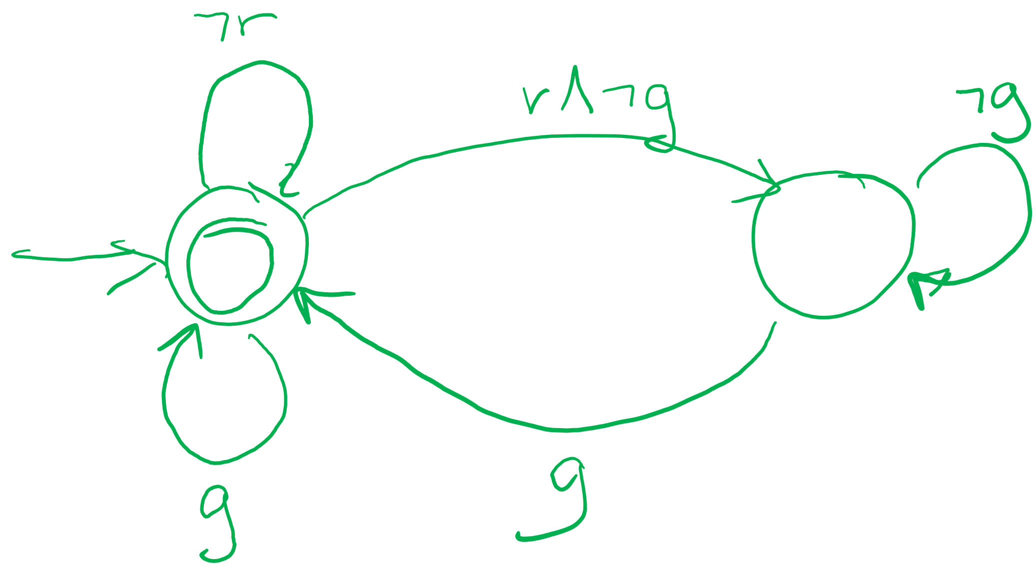

Input: r0, r1

Output: g0, g1

$G(r0 \rightarrow Fg0)$

$G(r1 \rightarrow Fg1)$

$G(\neg g0 \vee \neg g1)$



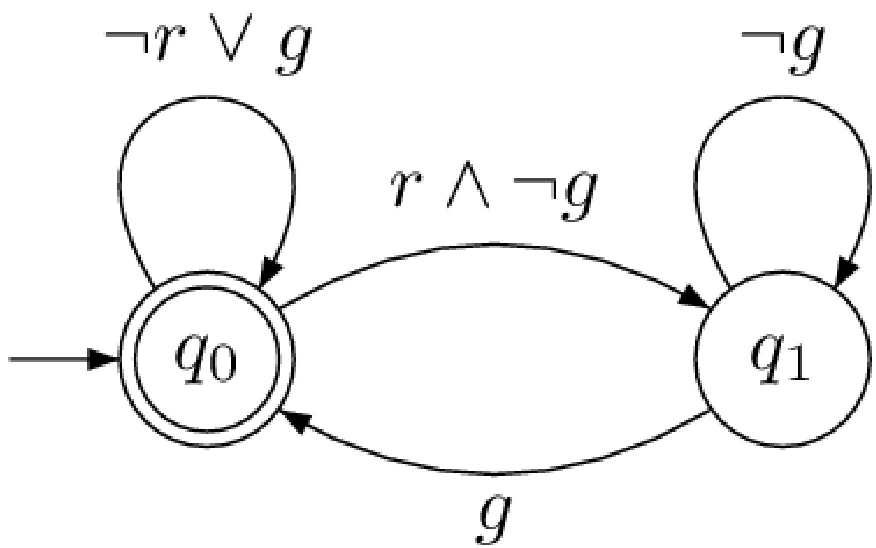
# Arbiter Specification

1. **Specify**
2. Create Game
3. Solve Game
4. Create System

Deterministic *Büchi automaton* for

G(r → Fg)

Accepting states must be visited infinitely often



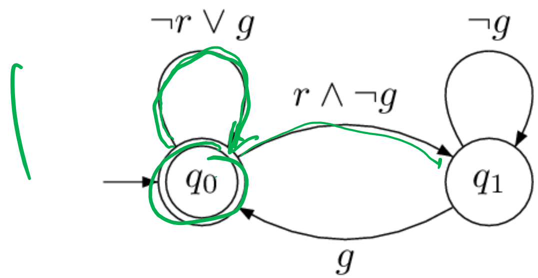

**Don't get stuck here!**

# Arbiter Game

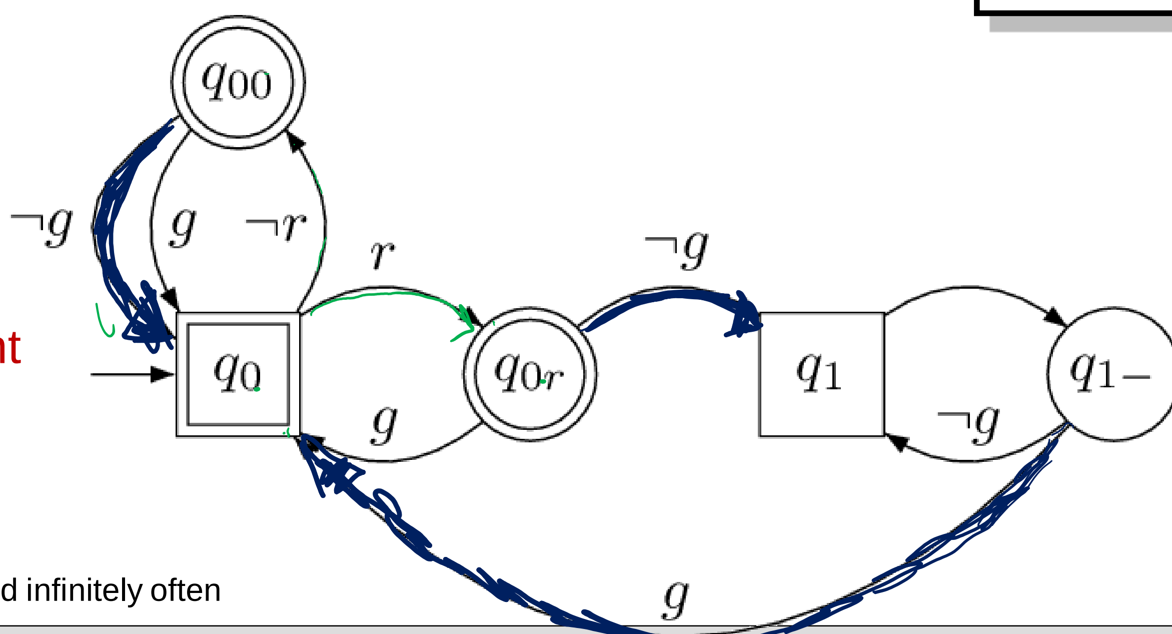
1. Specify
2. **Create Game**
3. Solve Game
4. Create System

Game for

$G(r \rightarrow Fg)$








Box: environment

Circle: system

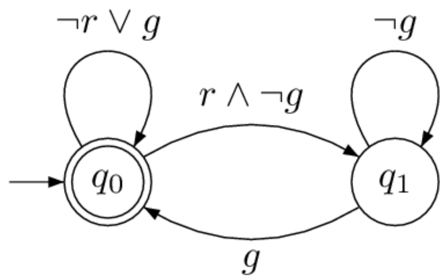
Accepting states must be visited infinitely often

# Arbiter Strategy

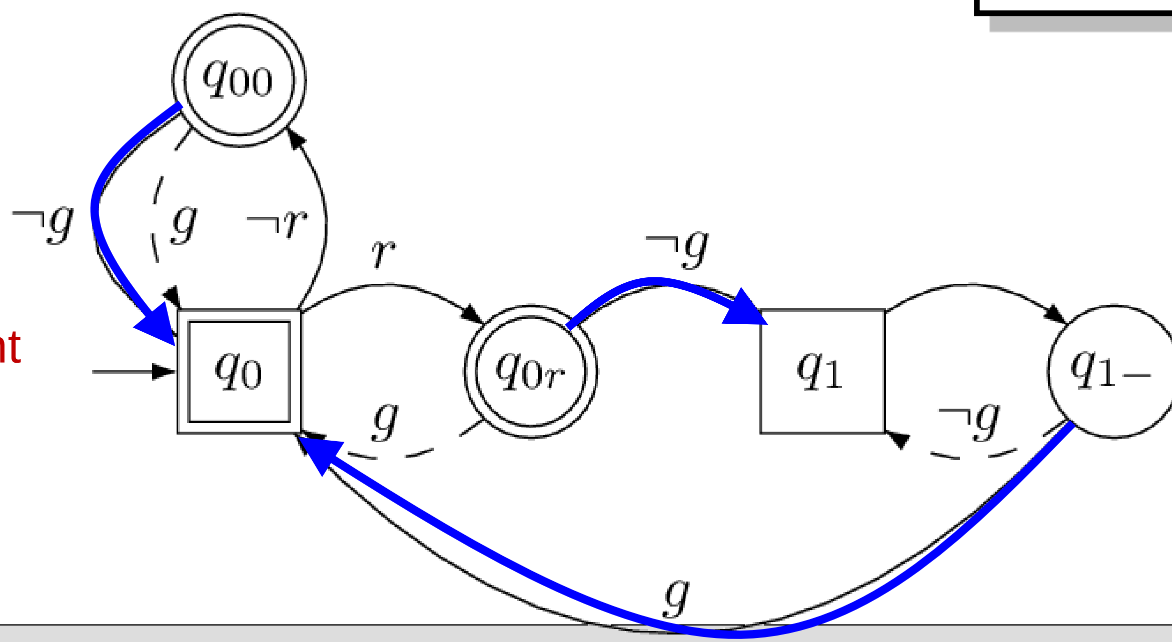
1. Specify
2. Create Game
3. **Solve Game**
4. Create System

Game for

G(r → Fg)








Box: environment

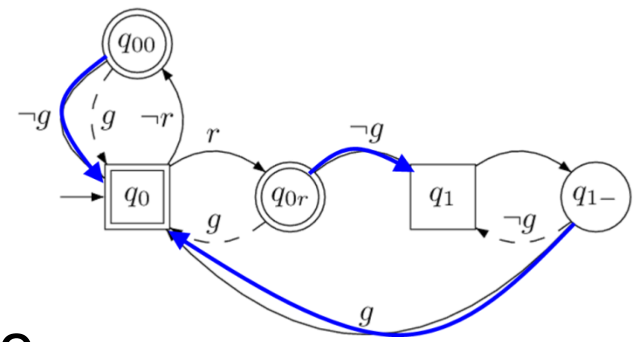
Circle: system

# Arbiter Strategy

1. Specify
2. Create Game
3. Solve Game
4. **Create System**

Game for

$G(r \rightarrow Fg)$





```
initial state = q0
while(){
  r = getinput();
  if(state==q0 && r==0) {g=0; state=q0}
  if(state==q0 && r==1) {g=0; state=q1}
  if(state==q1         ) {g=1; state=q0}
}
```

# Games

$G = (V_0, V_1, E, F)$.

$V_0$: Player 0 states (circles)

$V_1$: Player 1 states (squares) $V = V_0 \cup V_1$

$E \subseteq (V_0 \times V_1) \cup (V_1 \times V_0)$ edges

$F \subseteq (V_0 \cup V_1)^\omega$ winning condition

We want to know from whether (and how!) Player 0 can force a play in $F$.

# Games

$G = (V_0, V_1, E, F).$

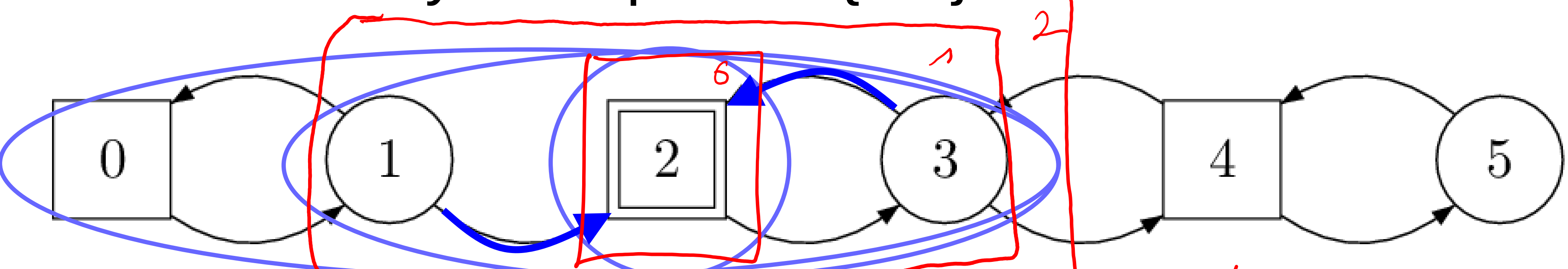
$V_0$: Player 0 states (circles)

$V_1$: Player 1 states (squares) $\quad V = V_0 \cup V_1$

$E \subseteq (V_0 \times V_1) \cup (V_1 \times V_0)$ edges

$F \subseteq (V_0 \cup V_1)^\omega$ winning condition

Reachability example: F= {0-5}*.2



# Reachability Game

$$pre(C) = \{q \mid q \in V_0 \wedge \exists q'.(q,q') \in E \wedge q' \in C\} \vee \{q \mid q \in V_1 \wedge \forall q'.(q,q') \in E \rightarrow q' \in C\}$$

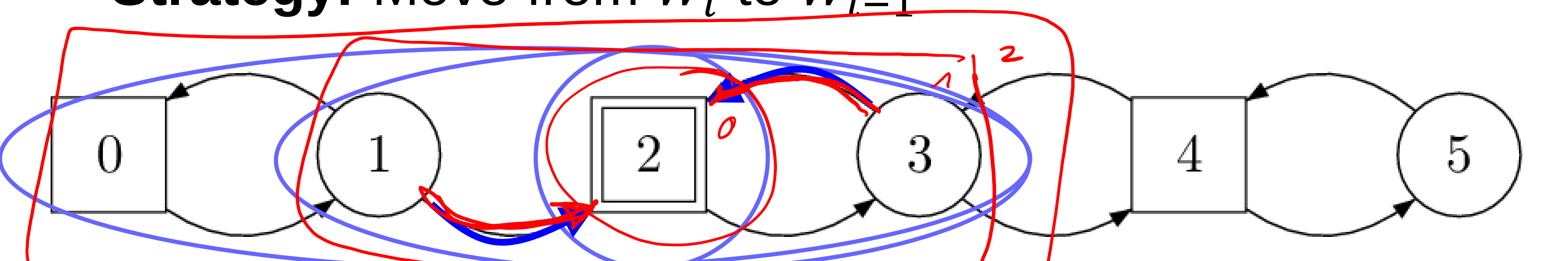
Reachability game with goal F:

$$\begin{aligned} W_0 &= F \\ W_i &= W_{i-1} \cup pre(W_{i-1}) \\ W &= \bigcup W_i \end{aligned}$$

EF

**Let's call this $rch(C) = W$**

**Strategy:** Move from $W_i$ to $W_{i-1}$



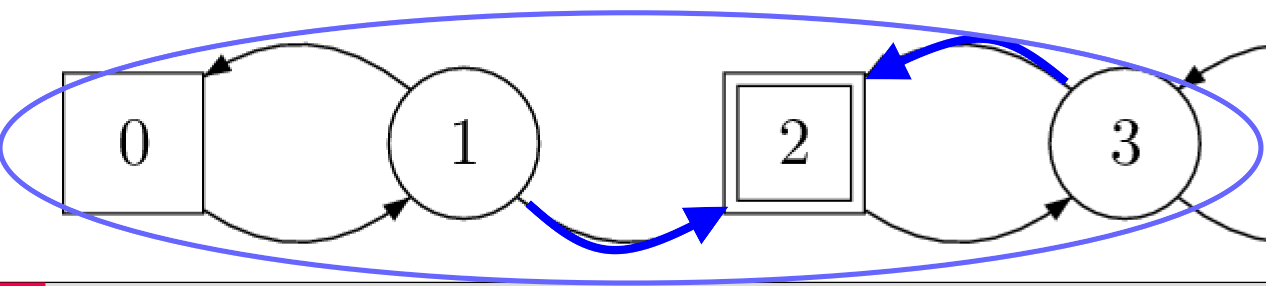
# Terminology

**Play** Sequence $\pi = q0q1q2\ldots \in V^{\omega}$ s.t. $(qi,qi+1) \in T$

**Play** is winning if $\pi \in F$.

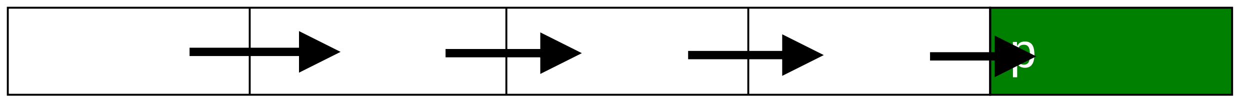
**Strategy** Function $\sigma$: $V^{*} \cdot V0 \rightarrow V$

**Play** adheres to $\sigma$ if for all prefixes $q0\ldots qi$ with $qi \in V0$, we have $qi+1 = \sigma(q0,\ldots,qi)$

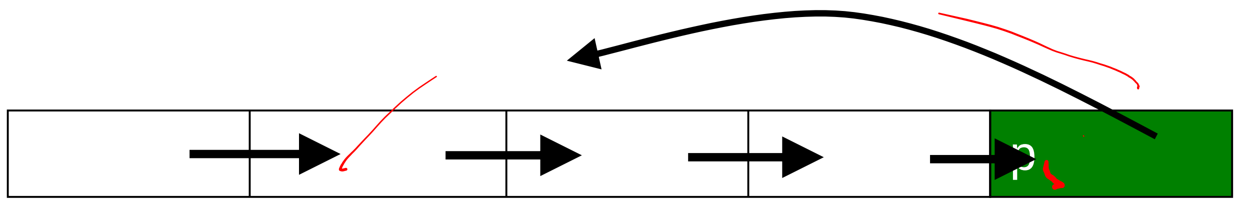


# More General Games

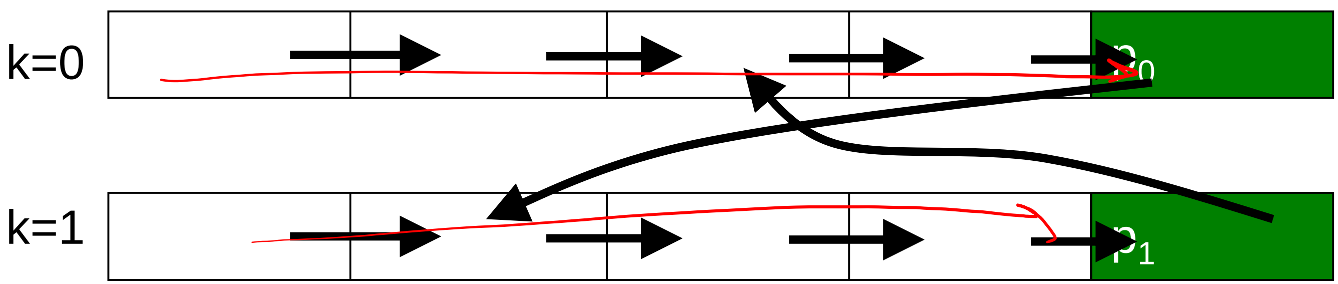
- Reachability: Fp




- Büchi: GF p




- Generalized Büchi:
- GF$p_0$ ∧ GF $p_1$ ∧ …



# Symbolic Games

# Symbolic Games

$X = \{x_1 \dots x_n\}$

$\cancel{W = \{w_1, \dots, w_n\}}$ system variables

$V_0$ is a Boolean formula over $\cancel{W}$ $X$: system states

$V = \neg V_0$: environment states

$R$ is a Boolean formula over $W$ and $W'$

$W \longrightarrow W'$

$$pre(C) = \{q \mid q \in V_0 \wedge \exists q'.(q,q') \in E \wedge q' \in C\} \vee$$
$$\{q \mid q \in V_1 \wedge \forall q'.(q,q') \in E \rightarrow q' \in C\}$$

$pre(C)(\cancel{W}\, X) =$ $V_0(x) \wedge \exists x'. R(x,x') \wedge C(x') \vee$
$V_1(x) \wedge \forall x'. R(x,x') \rightarrow C(x')$

BDDs

# Symbolic Games

$X = \{x_1, \dots, x_n\}$ system variables

$V_0$ is a Boolean formula over $X$: system states

$V = \neg V_0$: environment states

$R$ is a Boolean formula over $X$ and X$'$

$$pre(C) = \{q \mid q \in V_0 \wedge \exists q'.(q,q') \in E \wedge q' \in C\} \vee$$
$$\{q \mid q \in V_1 \wedge \forall q'.(q,q') \in E \rightarrow q' \in C\}$$

$$pre(C)(X) = V_0 \wedge \exists X'.C(W') \wedge R(X,X') \vee$$
$$V_1 \wedge \forall X'.R(X,X') \rightarrow C(X')$$

# Symbolic Game

Reachability game with goal F:

W = F;

while W changes do

    $W = W \vee pre(W)$

What kind of solver do we use?

# Selecting One Implementation

Specification = Set of sequential circuits

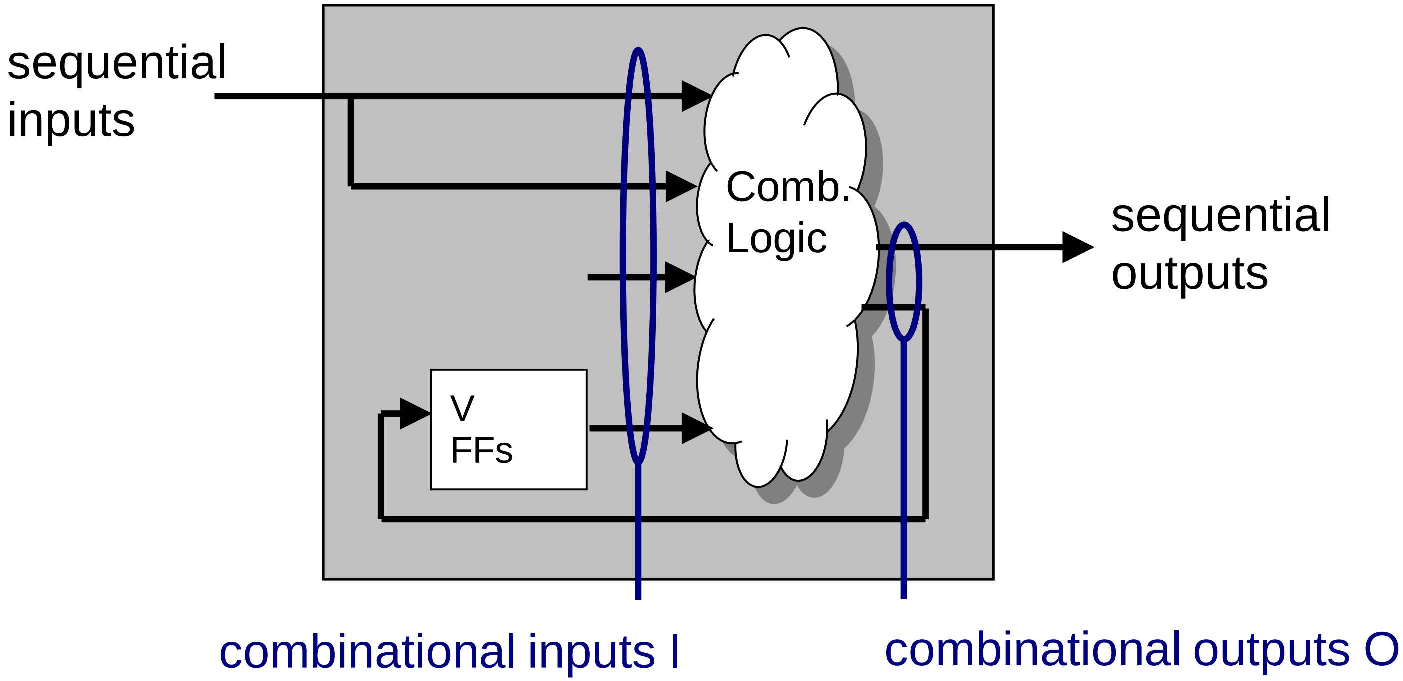
Strategy = Set of combinational circuits

Construction of circuit

One combinational circuit

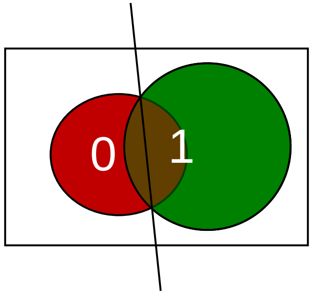
Less freedom
Fewer circuits
More complexity

# Constructing Circuit




- Spec is given in terms of sequential inputs and outputs
- Flipflops keep track of state of specification automata (state space of game)
- Strategy is relation between combinational inputs and combinational outputs: $R \subseteq I \times O$
- A circuit is a function f: $I \rightarrow O$

# From BDD to Circuit




## Relation Solving

**Given**: Strategy R: I x O

**Find**: function f: I $\to$ O such that
if f(i) = o then (i,o) $\in$ R

Multiple possibilities lead to wildly different sizes in circuits

# Back to Theory

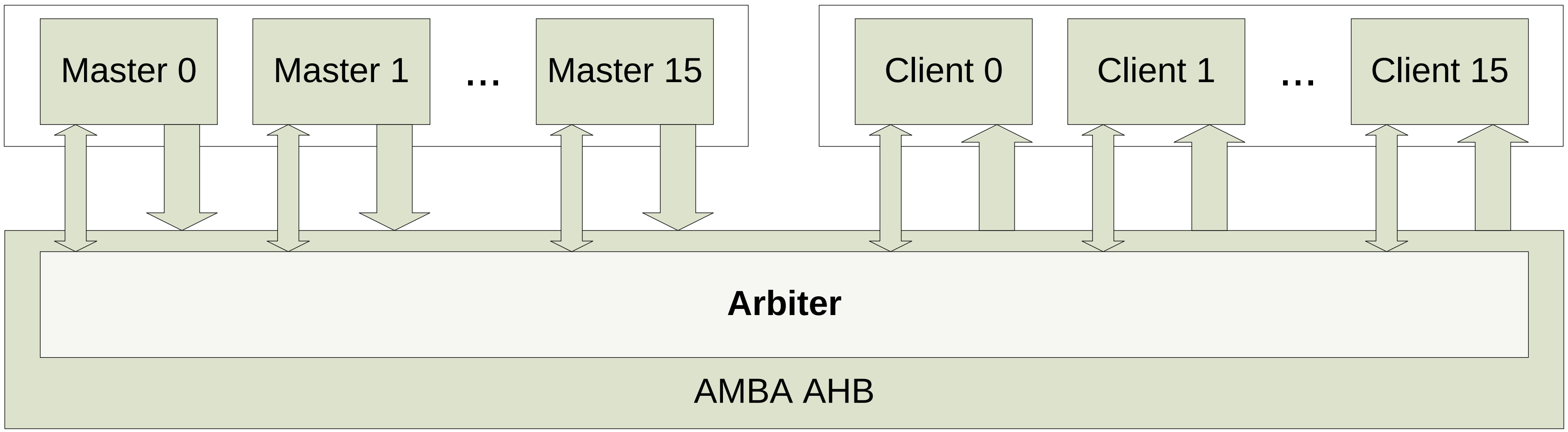
- Automata Theory
  - For Games
- Logics
  - For the Spec: Temporal Logic
  - For the Solution: Automatic Solvers
  - Quantifiers?

    $\forall i\, \exists o.\ \varphi$
  - Non-Boolean logics?

# Example III: AMBA

# AMBA Bus

- Industrial standard
- ARM's AMBA AHB bus
  - High performance on-chip bus
  - Data, address, and control signals (pipelined)
  - Arbiter part of bus (determines control signals)
  - Up to 16 masters and 16 clients



# AMBA Bus

- Master initiates transfer. Signals:
  - HBUSREQi - Master i wants the bus
  - HLOCKi - Master i wants an uninterruptible access
  - HBURST - This access has length 1/4/incr
  - address & data lines
- The arbiter decides access
  - HGRANTi - Next transfer for master i
  - HMASTER[..] - Currently active master
  - HMASTLOCK - Current access is uninterruptible
- The clients synchronize the transfer
  - HREADY - Ready for next transfer
- Sequence for master
  - Ask; wait for grant; wait for hready; state transfer type & start transfer

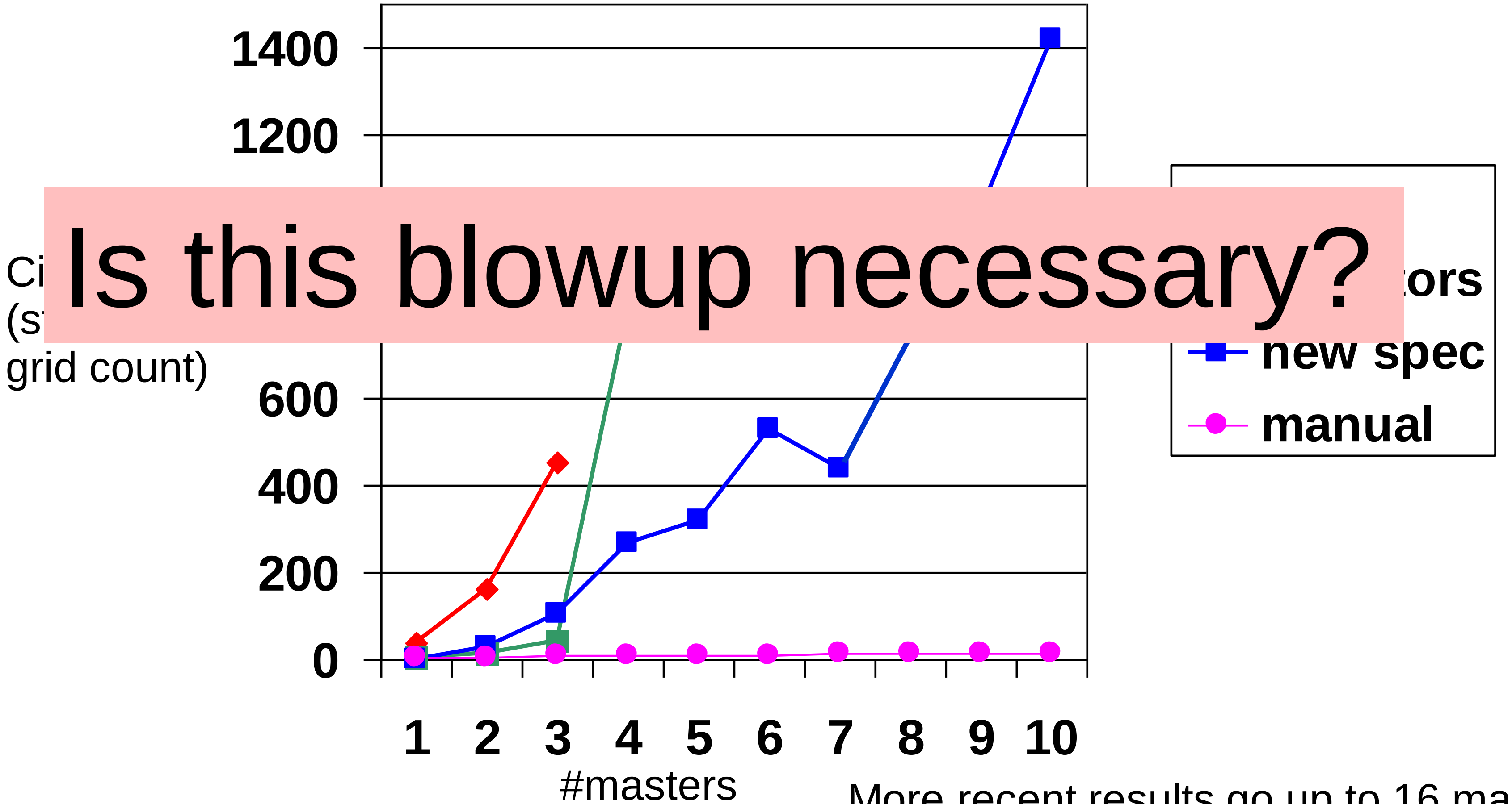
# AMBA Arbiter

- Specification
- 3 Assumptions, 12 Guarantees.
- Example:

*“When a locked unspecified length burst starts, new access does not start until current master (i) releases bus by lowering HBUSREQi.”*

$$\bigwedge_i G(\text{HMASTLOCK} \wedge \text{HBURST=INCR} \wedge \text{HMASTER=i} \wedge \text{START} \rightarrow$$
$$X(\neg\text{START}\ U\ \neg\text{HBUSREQi}))$$

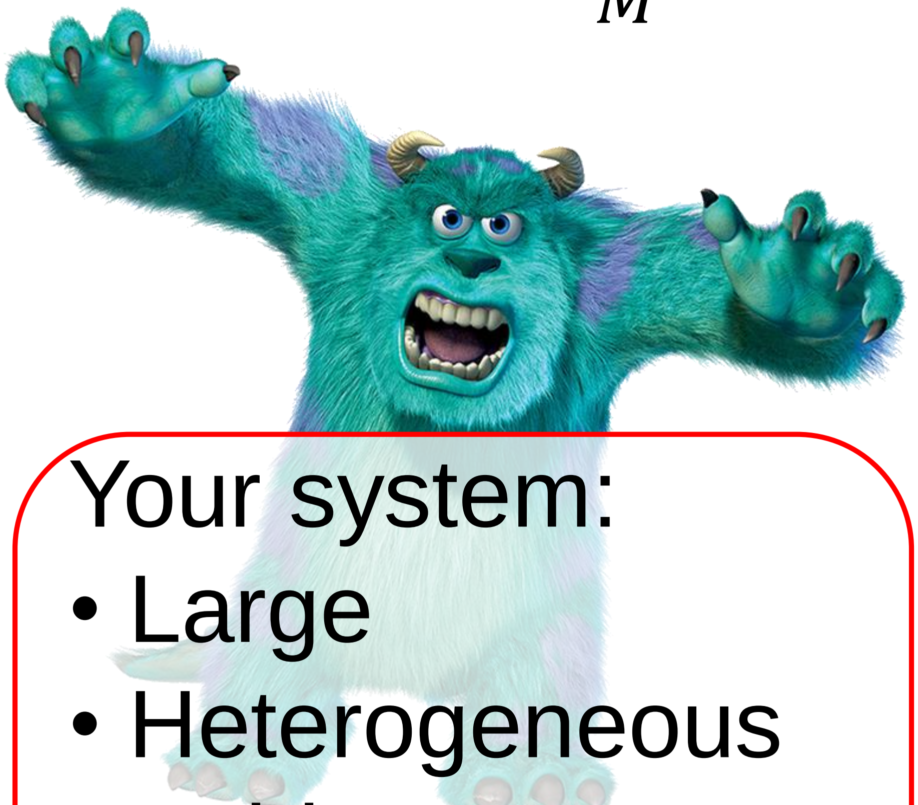
# New Spec

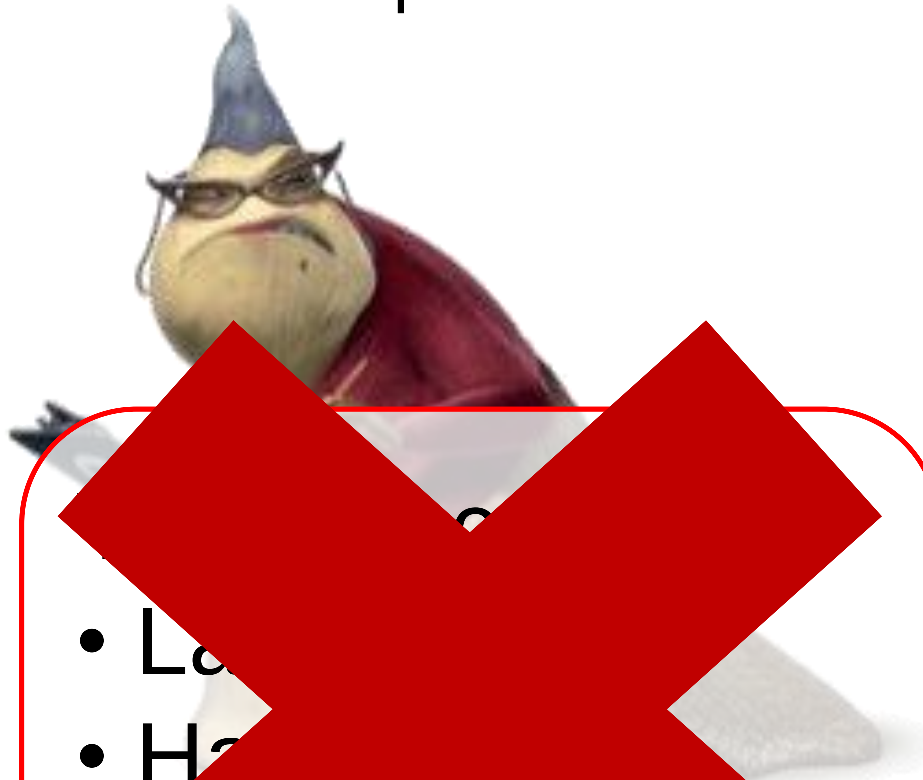



More recent results go up to 16 masters

# Application Example: Shields

$$M \quad \models \quad \phi$$

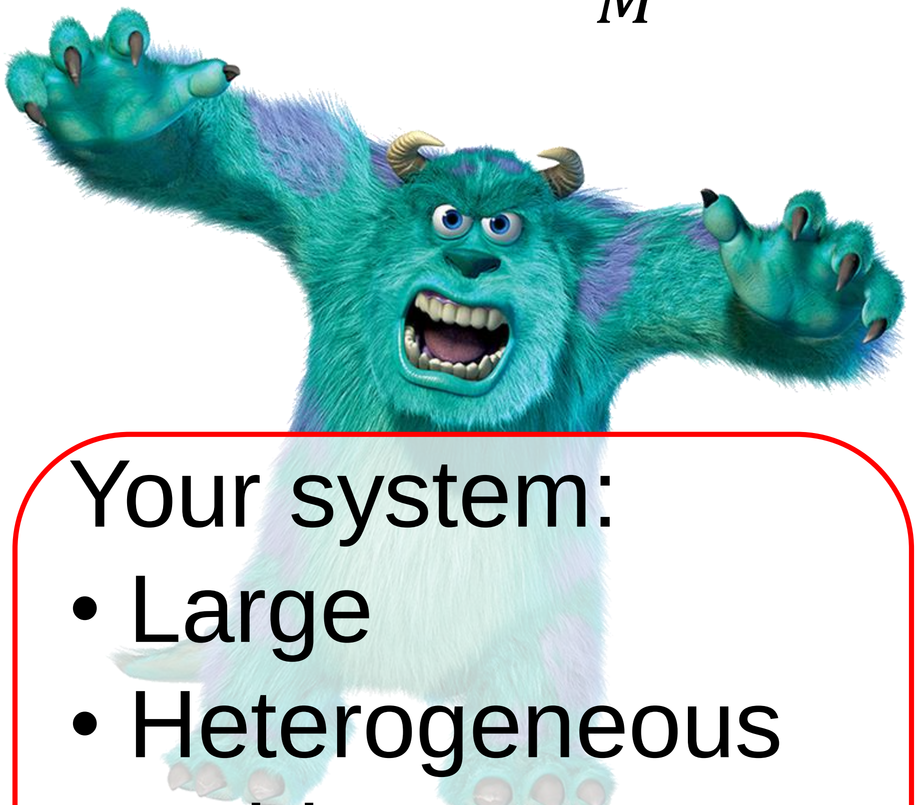




Your system:

- Large
- Heterogeneous
- Subject to failures
- Not available

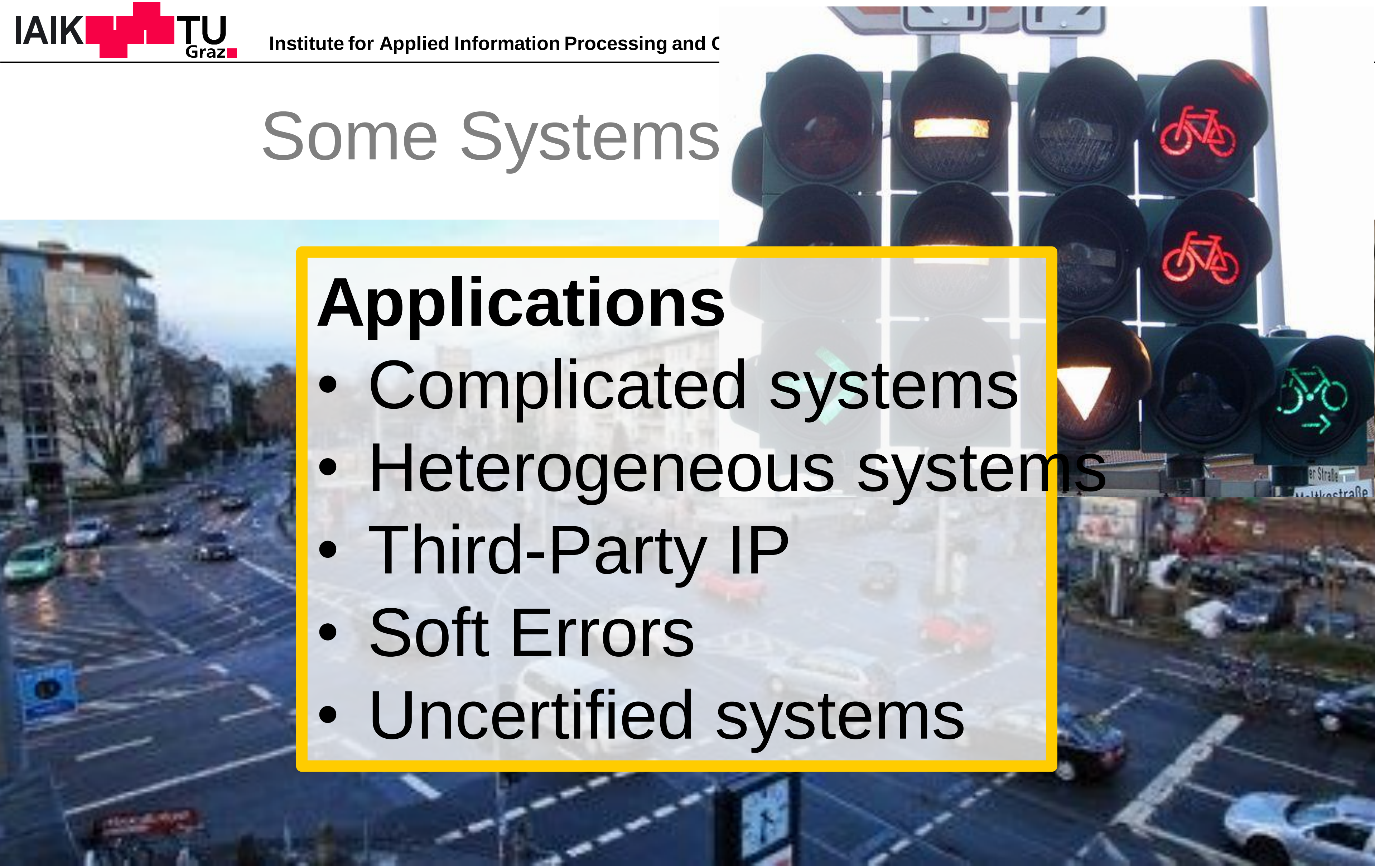
$$M \qquad \vDash \qquad \phi$$



Your system:
- Large
- Heterogeneous
- Subject to failures
- Not available

**Critical** spec:
- Critical aspects only
- Small & sweet

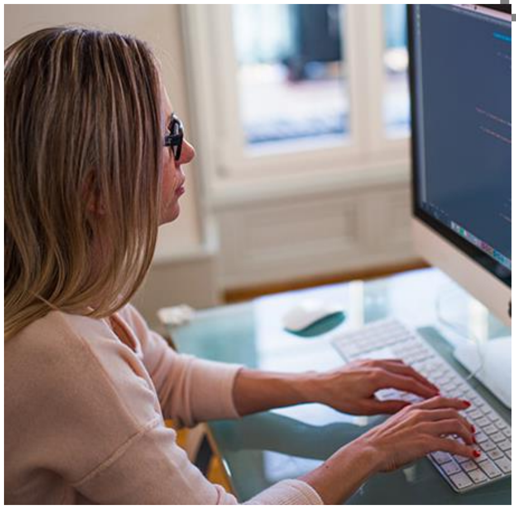
# Some Systems



**Applications**

- Complicated systems
- Heterogeneous systems
- Third-Party IP
- Soft Errors
- Uncertified systems

# Humans & Formal



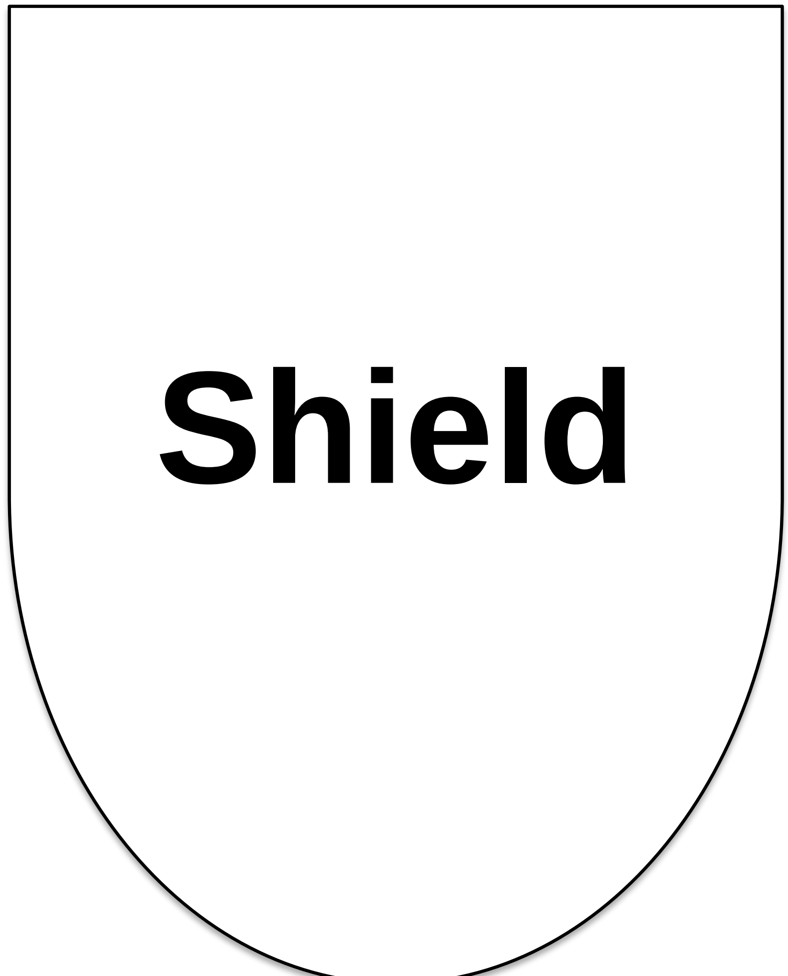
or machine learning

**Humans** should take care of excellent average case behavior

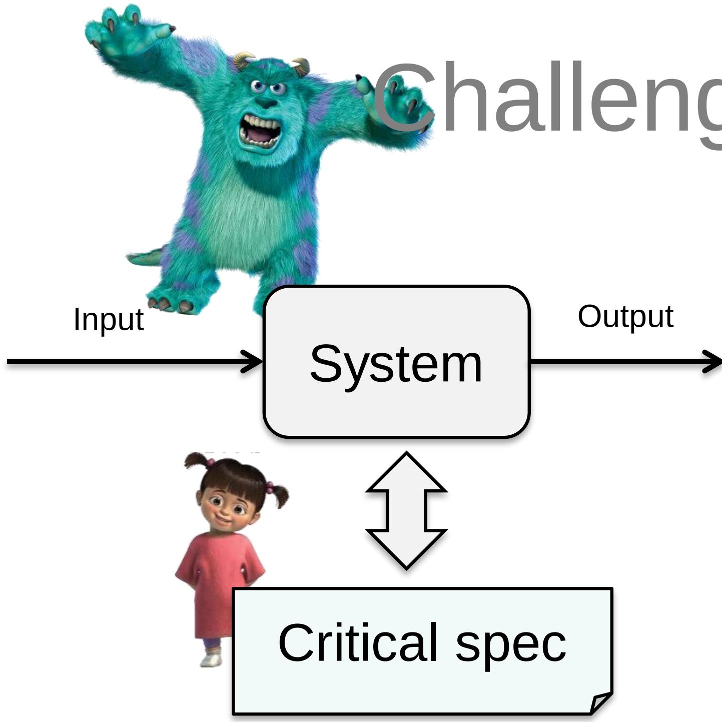
**Humans write programs**

**Formal** should take care of acceptable worst case

**Formal makes sure they are right at runtime**


Shield

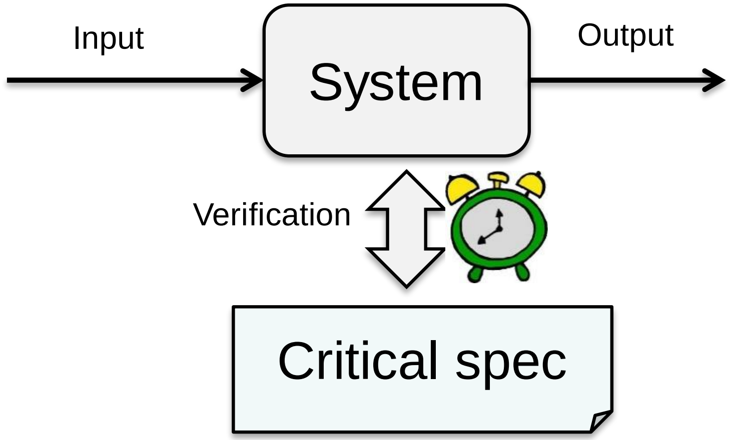
# Challenge




**System:** complicated

**Critical specification:** simple safety property

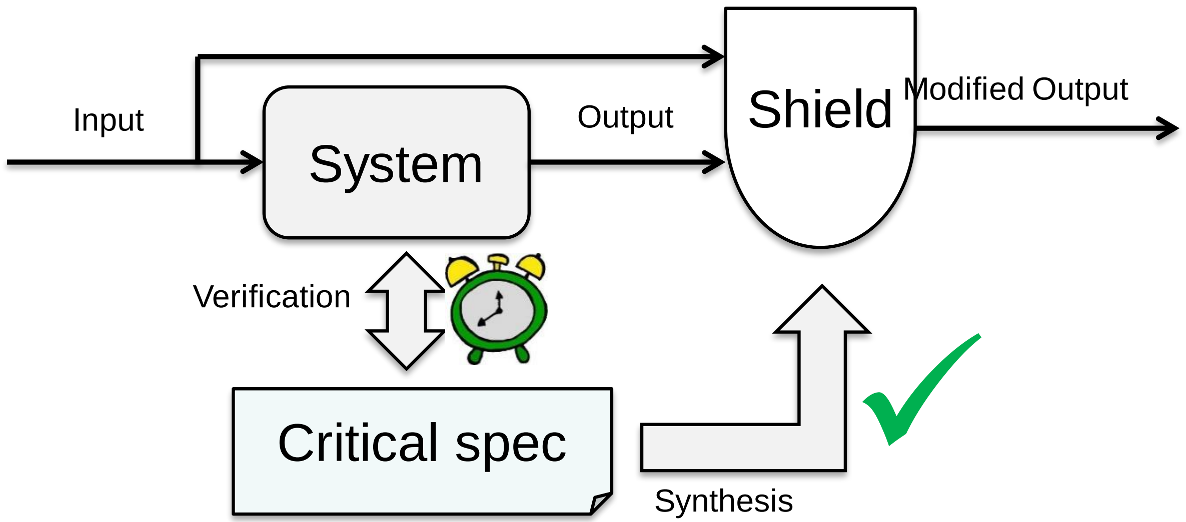
# Challenge




**Verification inconclusive:**

- System too complicated
- … but we need absolute certainty of critical spec

# Challenge




## Synthesize a "safety shield"

Scalable because critical spec is small

# Trafictec CMU Conflict Monitor Unit

**Monitors selected traffic controller channels for conflicts.**


**Goals:**

1. Generate automatically
2. Independent of system
3. Smarter than going to failsafe

## General

**The Trafictec CMU Conflict Monitor Unit**, which is optional for use with the SBC-2400 Traffic Control System, monitors selected traffic signal controller channels for conflicts. There are 8 input lines and 6 CMU channels. Two sets of input lines are tied together, forming a single signal group (i.e., Green and Walk) that can be monitored as one channel. When two channels are in conflict, the CMU will activate a conflict signal. This signal is read by the traffic controller, which will then place the intersection in flash. The conflict signal is latched and can only be cleared by a manual reset. The latching relay maintains the fault status during a power outage.

http://www.zwiesler.com/sbc2400/cmu_conflict_monitor.htm

# Crucial Properties

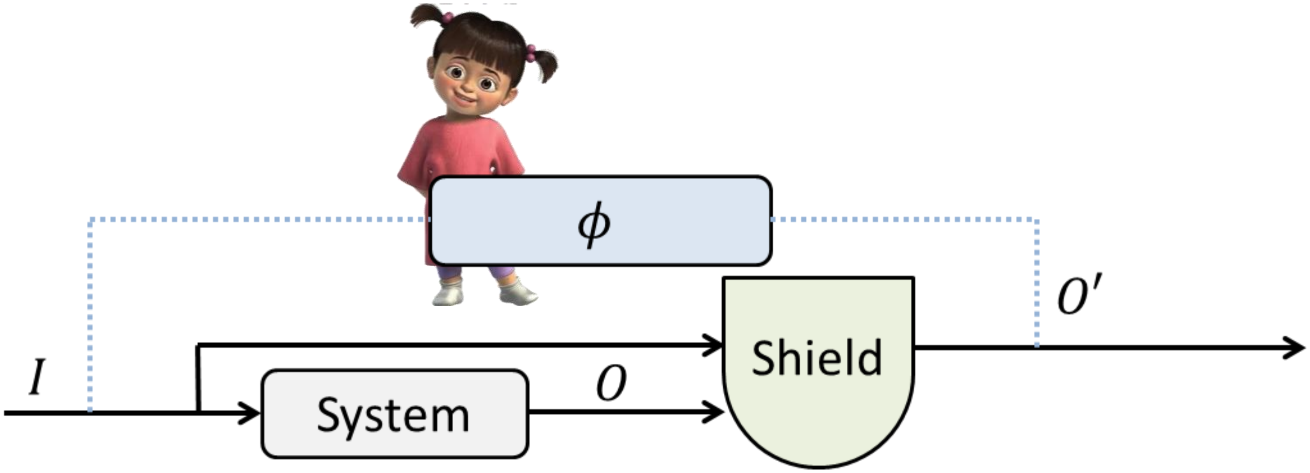
1. Correct

   Shielded system satisfies specification

2. Generic

   Shield does not depend the system implementation

3. Reacts on-the-fly, no delay



# Crucial Properties

1. Correct

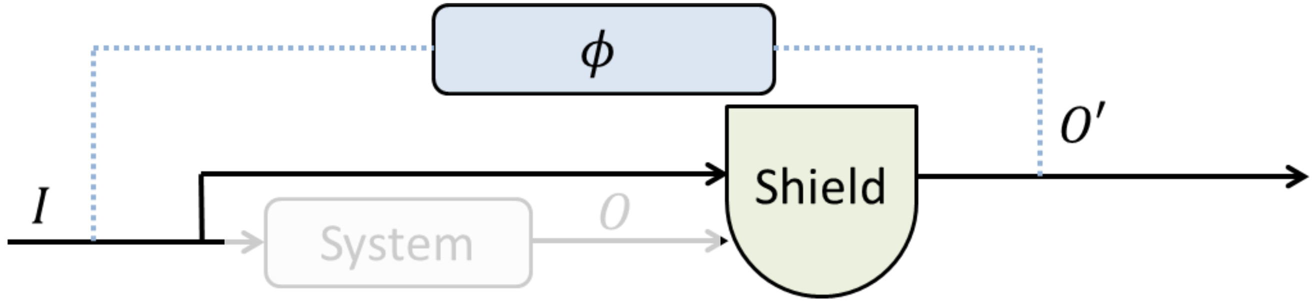
   Shielded system satisfies specification

2. Generic

   Shield does not depend the system implementation

3. Reacts on-the-fly, no delay

**Shield could ignore system!**



# Desirable Properties

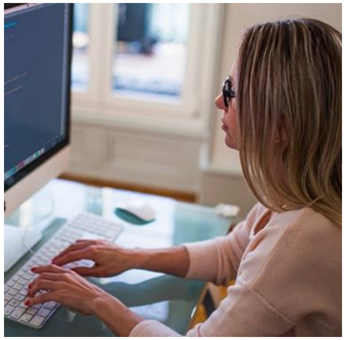
Minimum Interference:

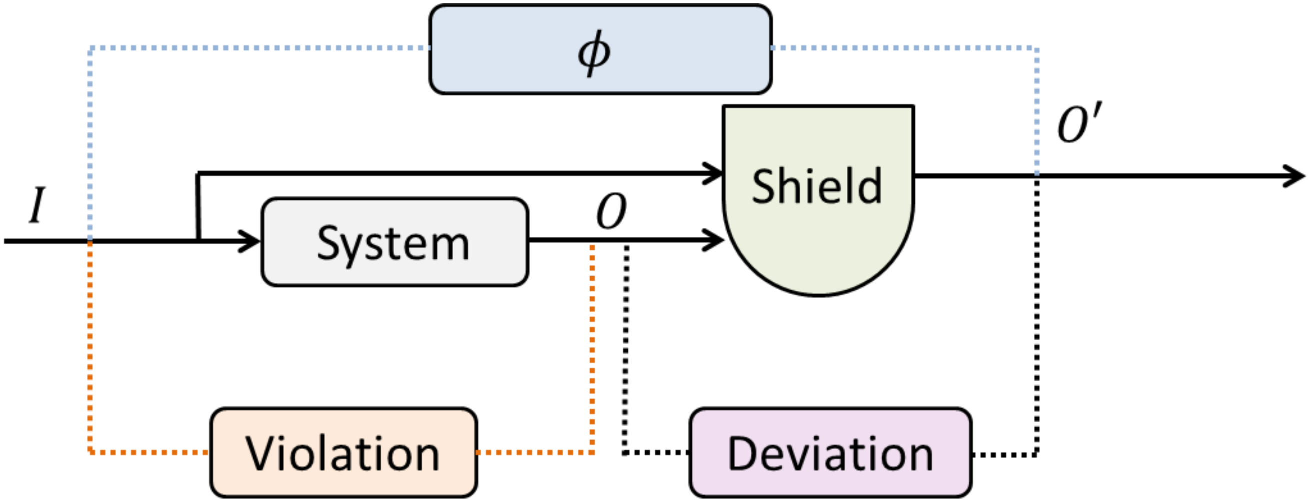
1. Deviate only if necessary

   Retain non-critical properties if system OK

2. Deviate as little as possible

   likely retain non-critical properties if system fails

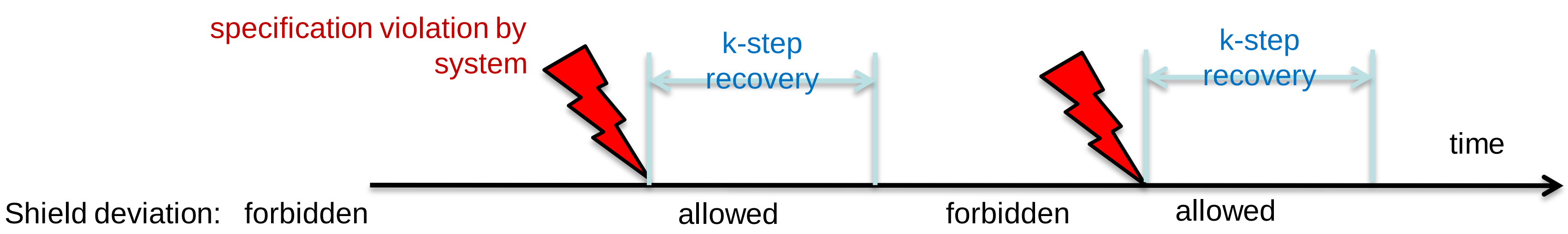




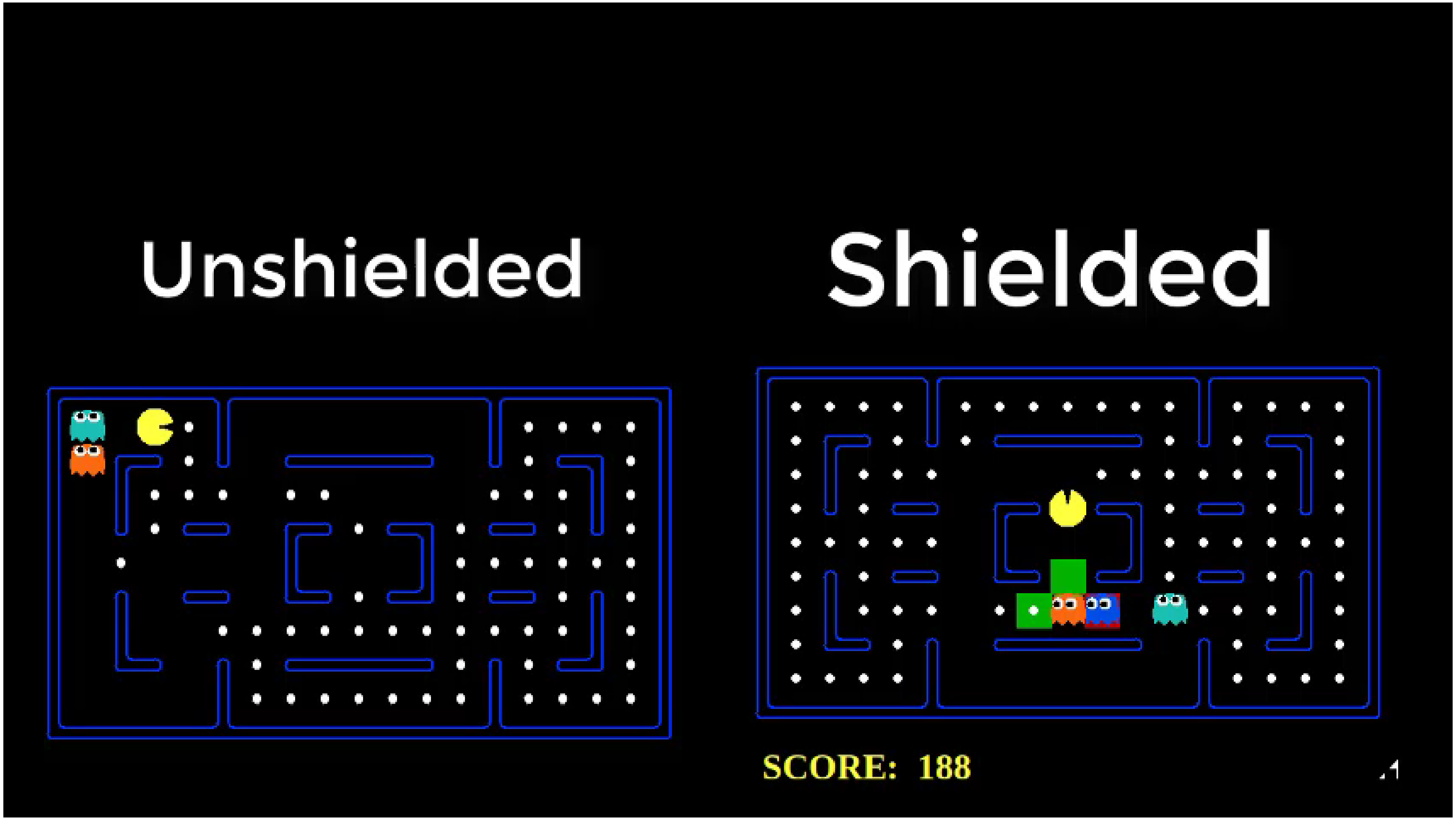
# Deviate as Little as Possible

## k-stabilization

Deviation allowed up to k steps after violation




*What does it mean to have multiple failures?*


Unshielded
Shielded
SCORE: 188

# Organizational

| Number | Date | Lecturer | Chapter |
|---|---|---|---|
| 1 | 2020-03-10 | Roderick Bloem | Introduction |
| 2 | 2020-03-17 | Bettina Könighofer | Bounded Synthesis, safety only |
| 3 | 2020-03-24 | Vedad Hadzic | Introduction SMT (Z3) |
| 4 | 2020-03-31 | Bettina Könighofer | SMT – Theory |
| 5 | 2020-04-21 | Masoud Ebrahimi | LTL |
| 6 | 2020-04-28 | All SCOS members | **Presentations Exercise 1 by Students** |
| 7 | 2020-05-05 | Benedikt Marderbacher | Omega Autmomata |
| 8 | 2020-05-12 | Benedikt Marderbacher | Bounded Synthesis |
| 9 | 2020-05-19 | Bettina Könighofer | Synthesis via Games, Games 1 |
| 10 | 2020-05-26 | Bettina Könighofer | Synthesis via Games, Games 2 |
| 11 | 2020-06-09 | Roderick Bloem | Relation Determinization |
| 12 | 2020-06-16 | All SCoS Members | Research |
| 13 | 2020-06-23 | All SCoS Members | **Presentations Exercise 2 by Students** |

# Time Line

- Lecture: 120 minutes
  - 10:00-11:00
  - 15 minute break
  - 11:15-12:15
  - Or not??

- Interactive!
- Exam will be waived for those who participate actively

- Uebung
  - Pencil and Paper take-home exercises

# Corona?

- Webex