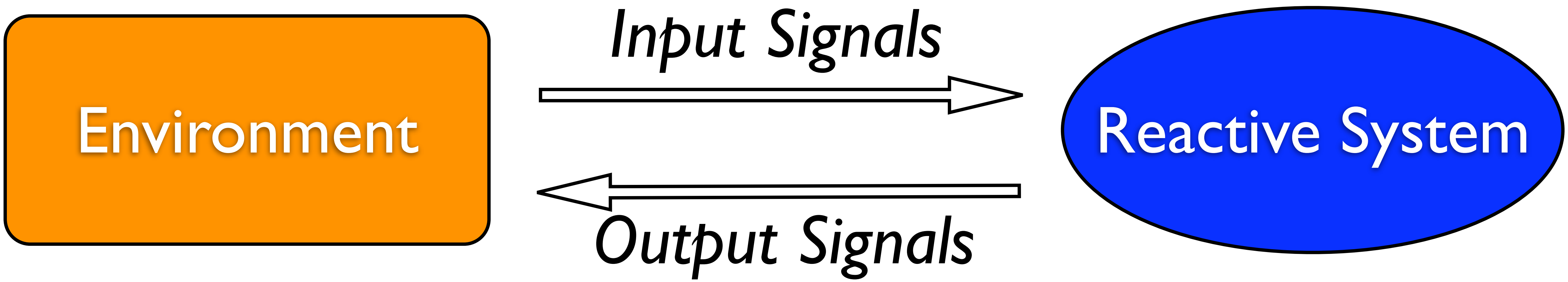
Towards Efficient Synthesis of LTL Specifications

Emmanuel Filiot
joint with Naiyong Jin and Jean-François Raskin

Université Libre de Bruxelles

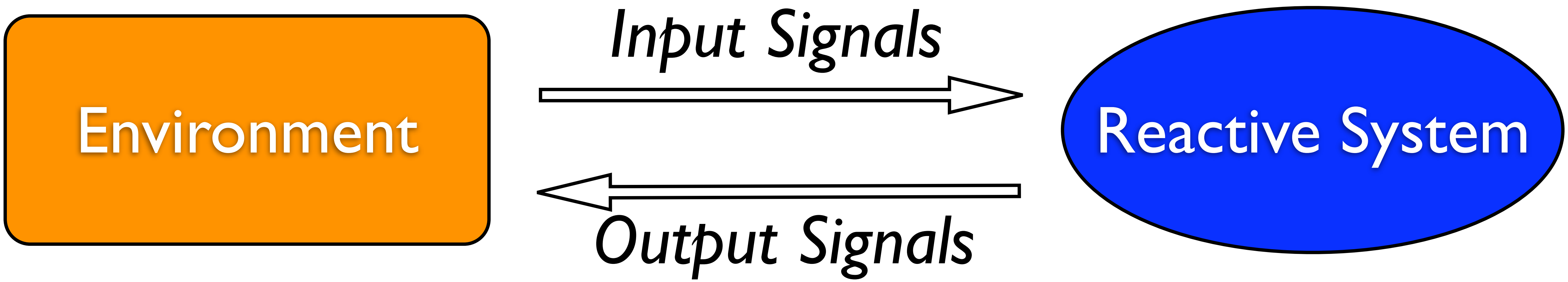
FNRS contact day

Reactive Systems



- continuous interaction with their environment
- non-terminating
- have to respect real-time properties (e.g. safety properties)
- have to cope with the uncontrollable behavior of their environment

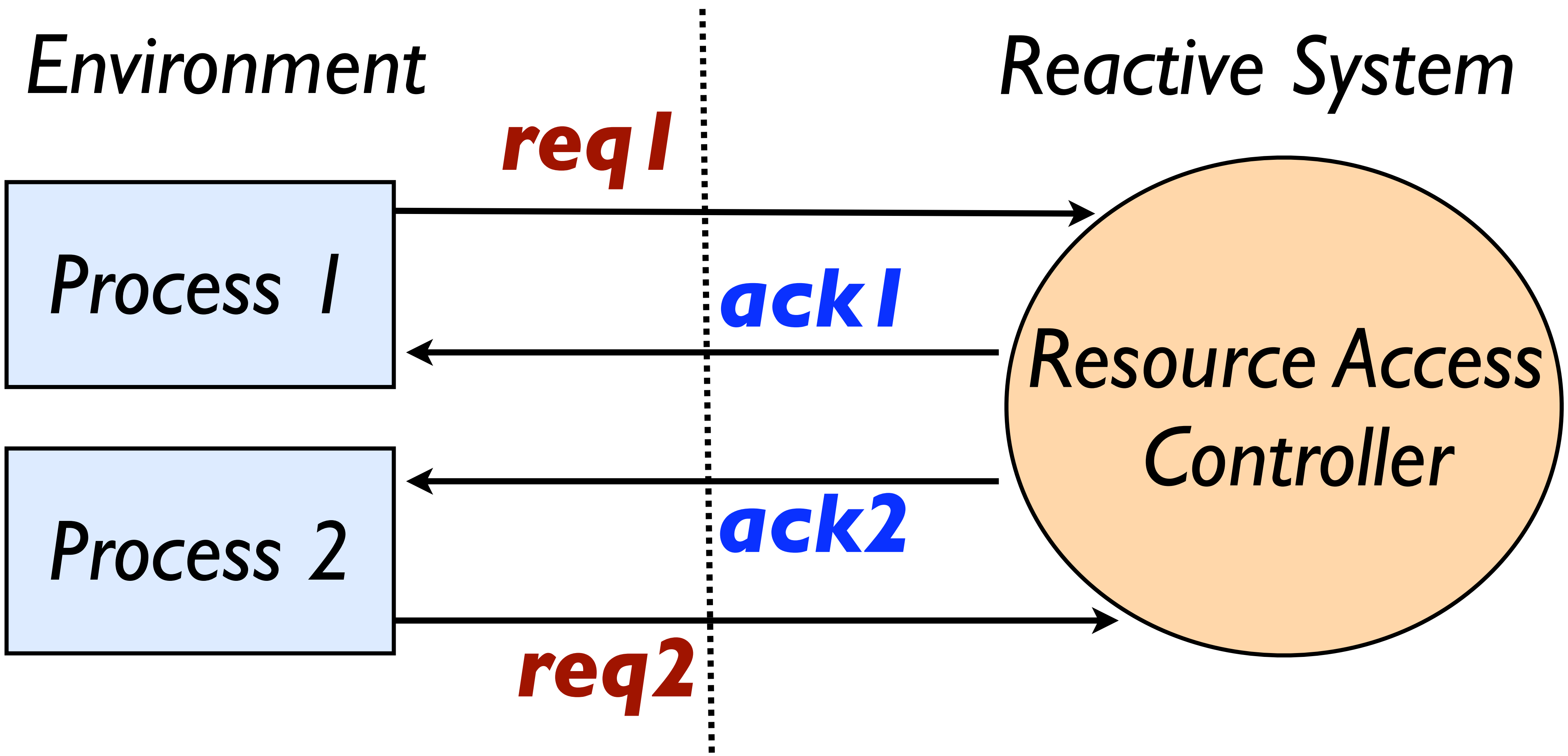
Reactive Systems



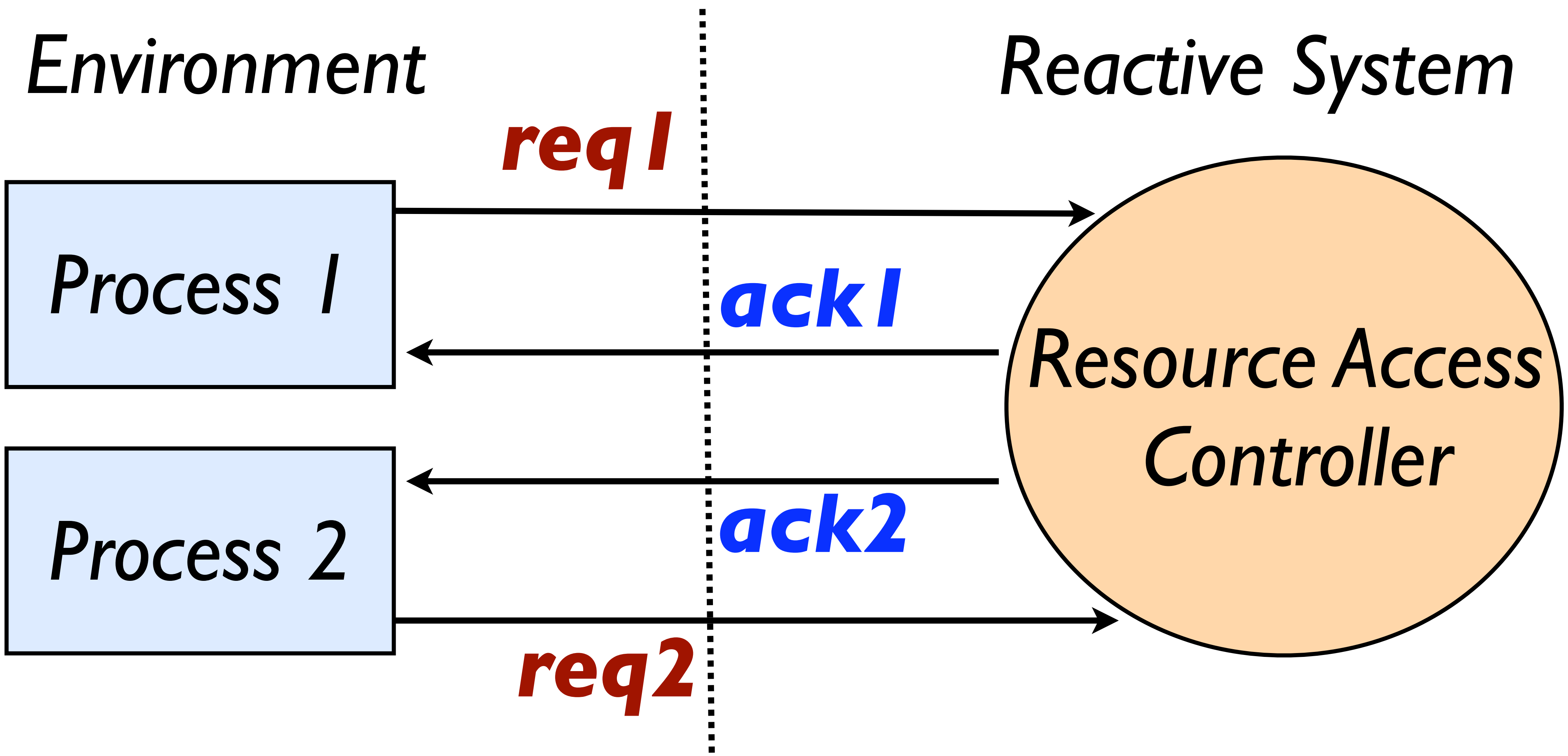
- continuous interaction with their environment
- non-terminating
- have to respect real-time properties (e.g. safety properties)
- have to cope with the uncontrollable behavior of their environment

Hard to design, needs **synthesis** from specification!

Example


Environment
Reactive System
req1
Process 1
ack1
Resource Access Controller
Process 2
ack2
req2

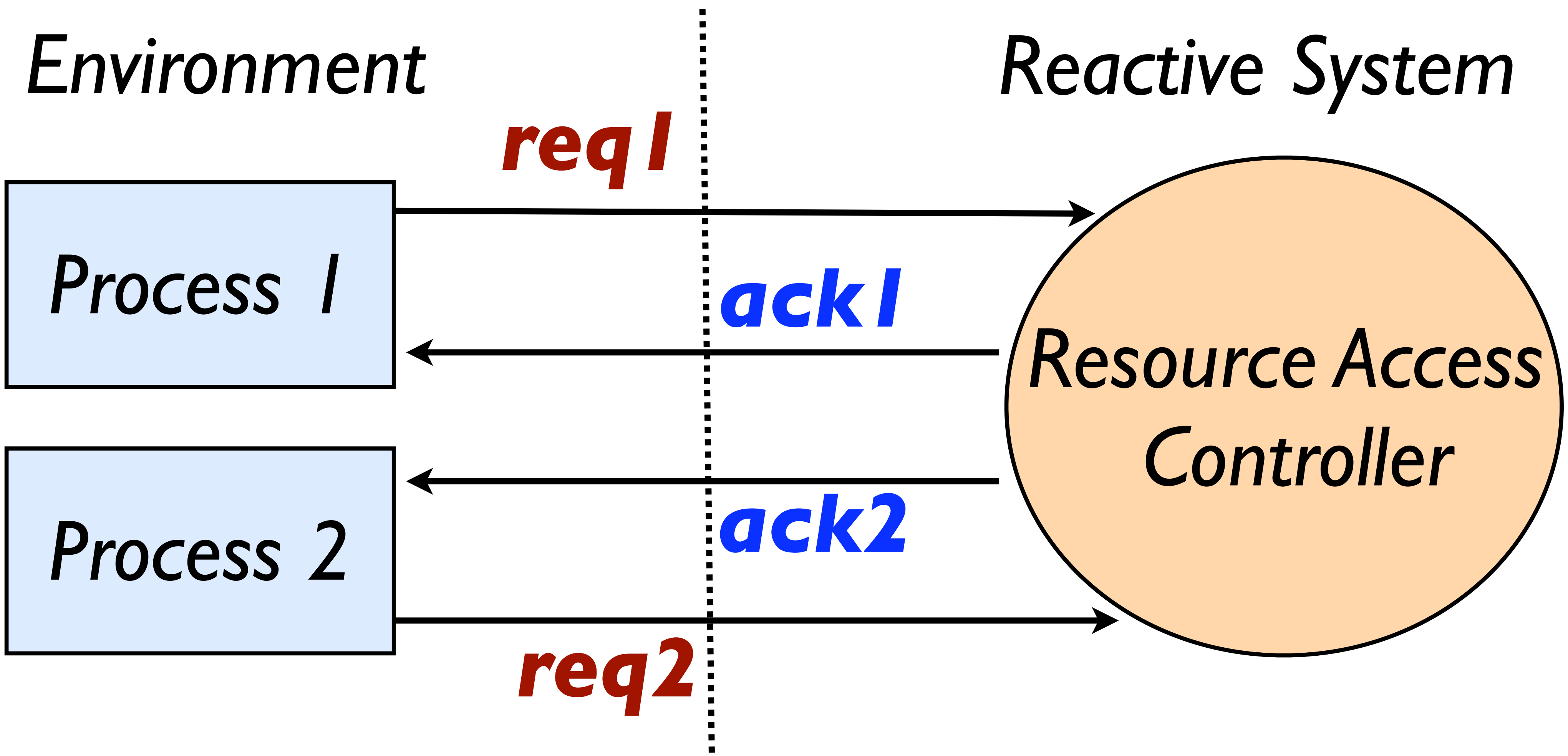
Example



Executions: infinite sequences of sets of signals

{***req1***, ***req2***} {***ack1***} {***req1***} {***ack2***} {***req1***} {***ack1***} **...**

Example



Properties we would like to ensure

Liveness property: G (**reqi** -> F **acki**) ***i=1,2***

Safety property: G (¬**ack1** ∨ ¬**ack2**)

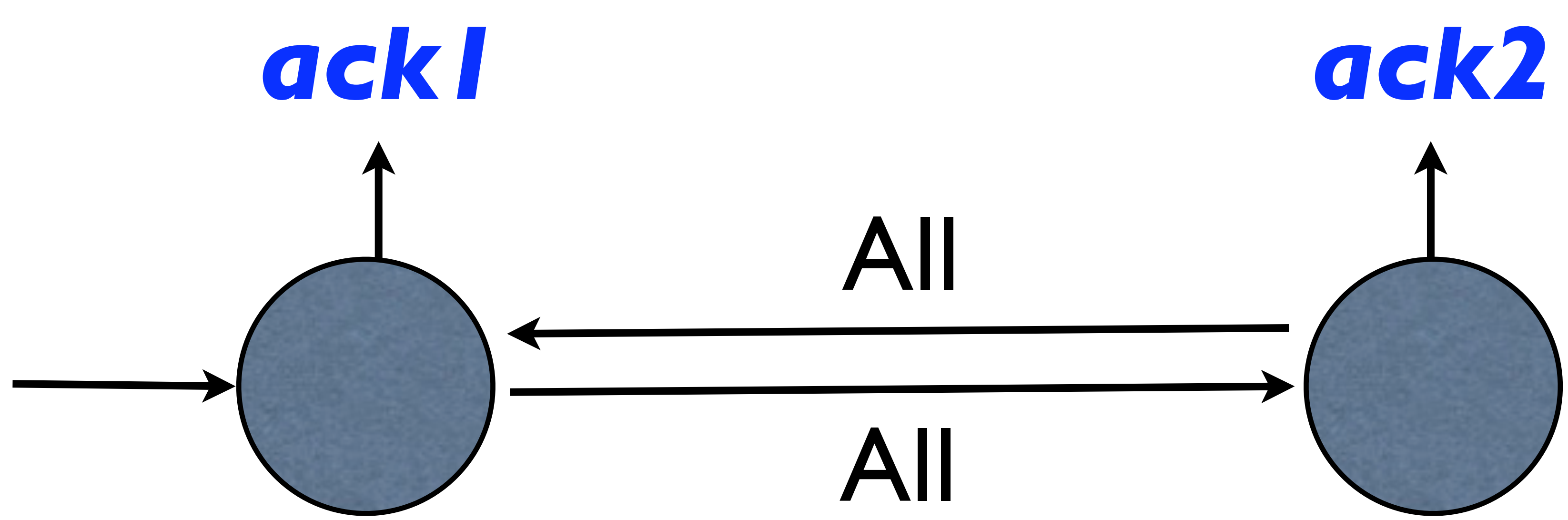
LTL Synthesis

LTL Spec

Liveness property: G (***reqi*** -> F ***acki***) ***i=1,2***

Safety property: G (¬***ack1*** ∨ ¬***ack2***)

generate a RS that realizes the spec



LTL Synthesis

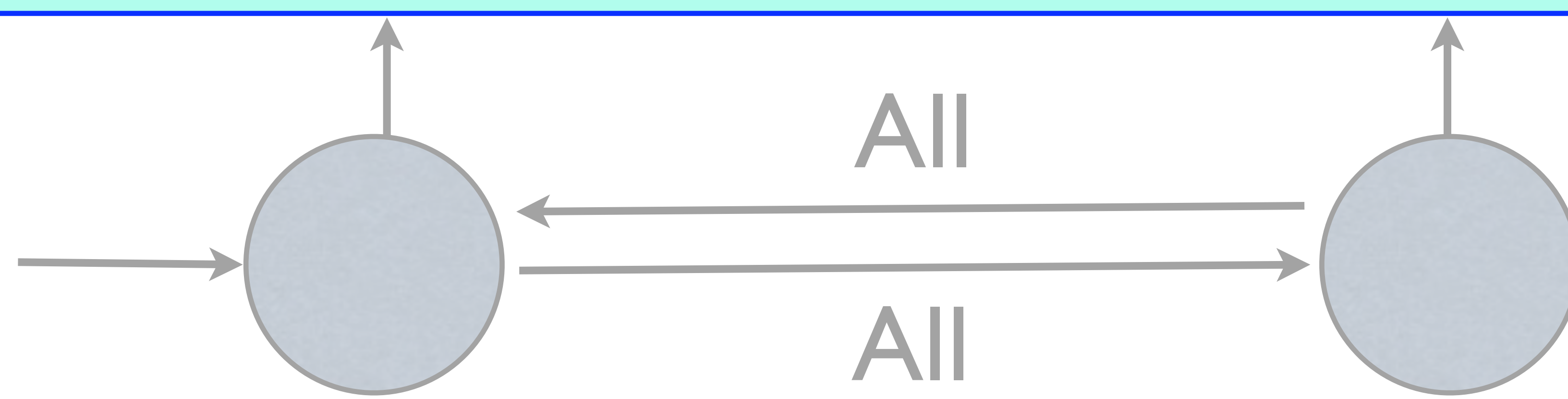
LTL Spec

Liveness property: G (***reqi*** -> F ***acki***) ***i=1,2***

Safety property: G (¬***ack1*** ∨ ¬***ack2***)

Realizability

Given an LTL spec, does there exists a RS such that all its executions (whatever the environment does) satisfy the spec ?

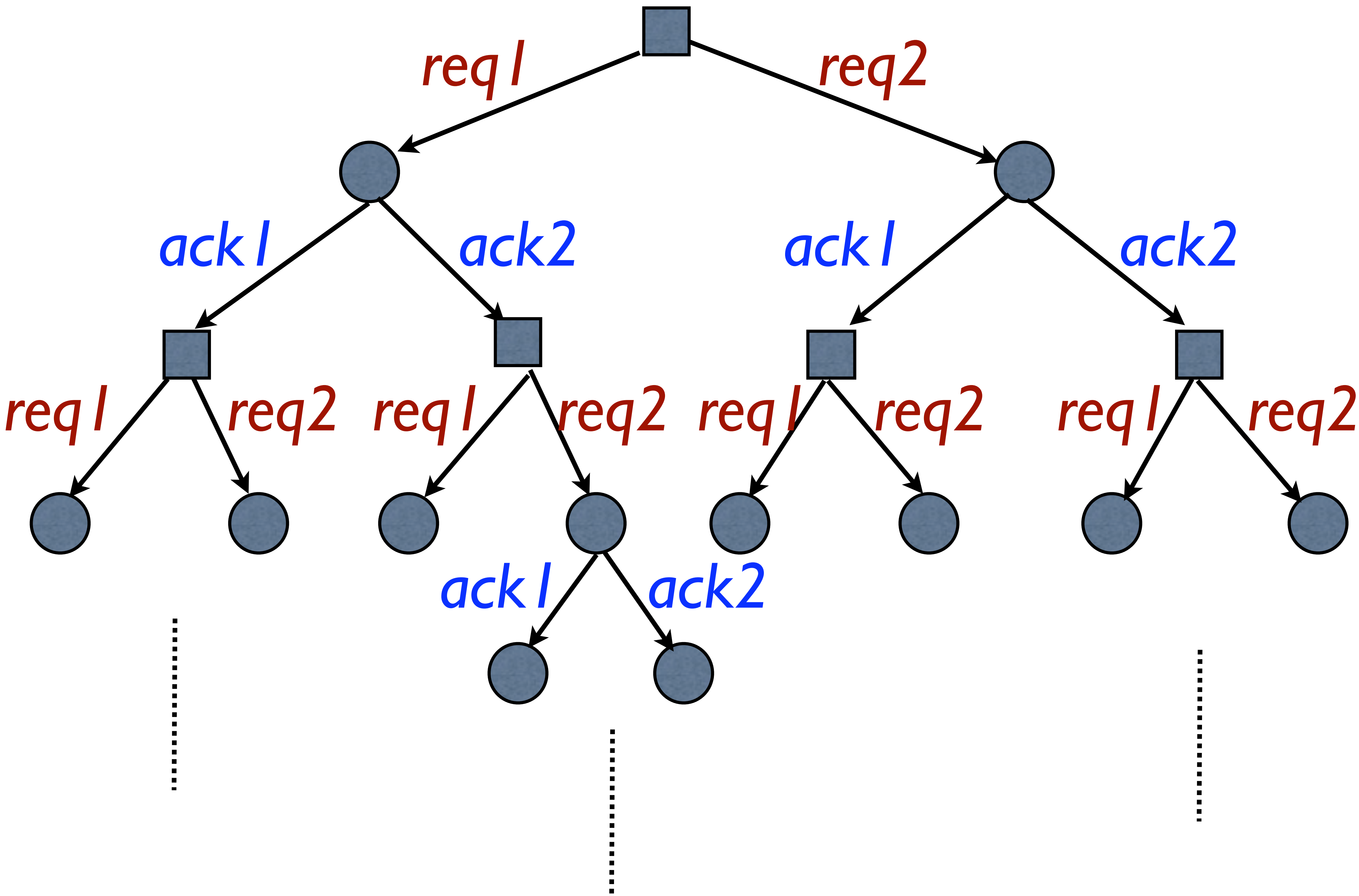


Unrealizable Spec

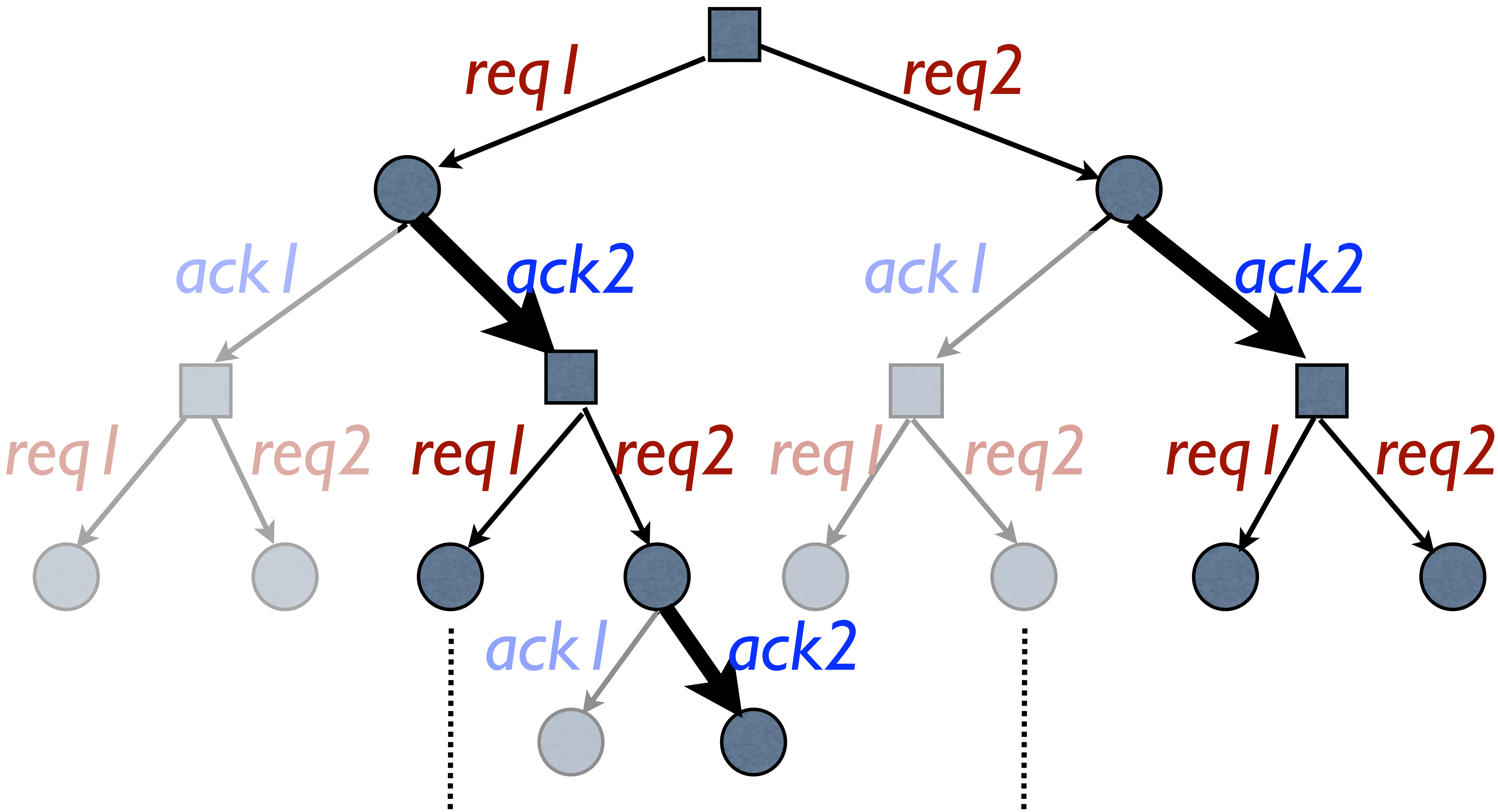
G (ack1 -> F req1)

"Each time the system acknowledge, the environment eventually sends a request"

Synthesis as Game



Reactive System as a Strategy



All the infinite paths have to satisfy the spec

Existing Procedures

- if a spec is realizable, it is realizable by a finite-steate strategy
- 2ExpTime-Complete [Rosner, 92]
- "classical" procedure [Pnueli, Rosner, 89]

LTL $\longrightarrow$ **Rabin Game**

Existing Procedures

- if a spec is realizable, it is realizable by a finite-steate strategy
- 2ExpTime-Complete [Rosner, 92]
- "classical" procedure [Pnueli, Rosner, 89]

Needs Safra's Determinization !

Existing Procedures

- if a spec is realizable, it is realizable by a finite-steate strategy
- 2ExpTime-Complete [Rosner, 92]
- "classical" procedure [Pnueli, Rosner, 89]

Needs Safra's Determinization !

- Safraless procedure [Kupferman, Vardi, 05]

LTL $\longrightarrow$ **Büchi Game**

Existing Procedures

- if a spec is realizable, it is realizable by a finite-steate strategy
- 2ExpTime-Complete [Rosner, 92]
- "classical" procedure [Pnueli, Rosner, 89]

Needs Safra's Determinization !

- Safraless procedure [Kupferman, Vardi, 05]

Implemented in Lily [Jobstmann, Bloem]

Existing Procedures

- if a spec is realizable, it is realizable by a finite-steate strategy
- 2ExpTime-Complete [Rosner, 92]
- "classical" procedure [Pnueli, Rosner, 89]

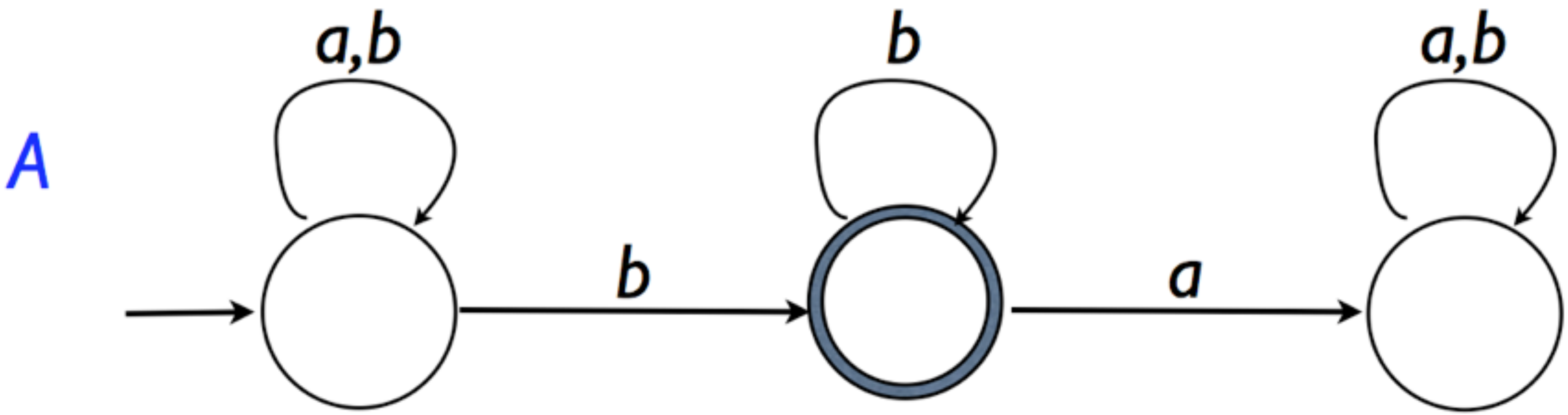
Needs Safra's Determinization !

- Safraless procedure [Kupferman,Vardi, 05]

Implemented in Lily [Jobstmann, Bloem]

- in this talk: LTL $\longrightarrow$ Safety Game

K-CoBüchi Automata



An (infinite) word is accepted

iff

all its runs visits **at most *K*** accepting states

In this example: words with at most K symbols b

Automata as a spec

- K-co-Büchi automata specify infinite words
- they can be used as RS specifications

Theorem

For any LTL specification Φ, one can construct a K-co-Büchi automaton A such that:

Φ is realizable

iff

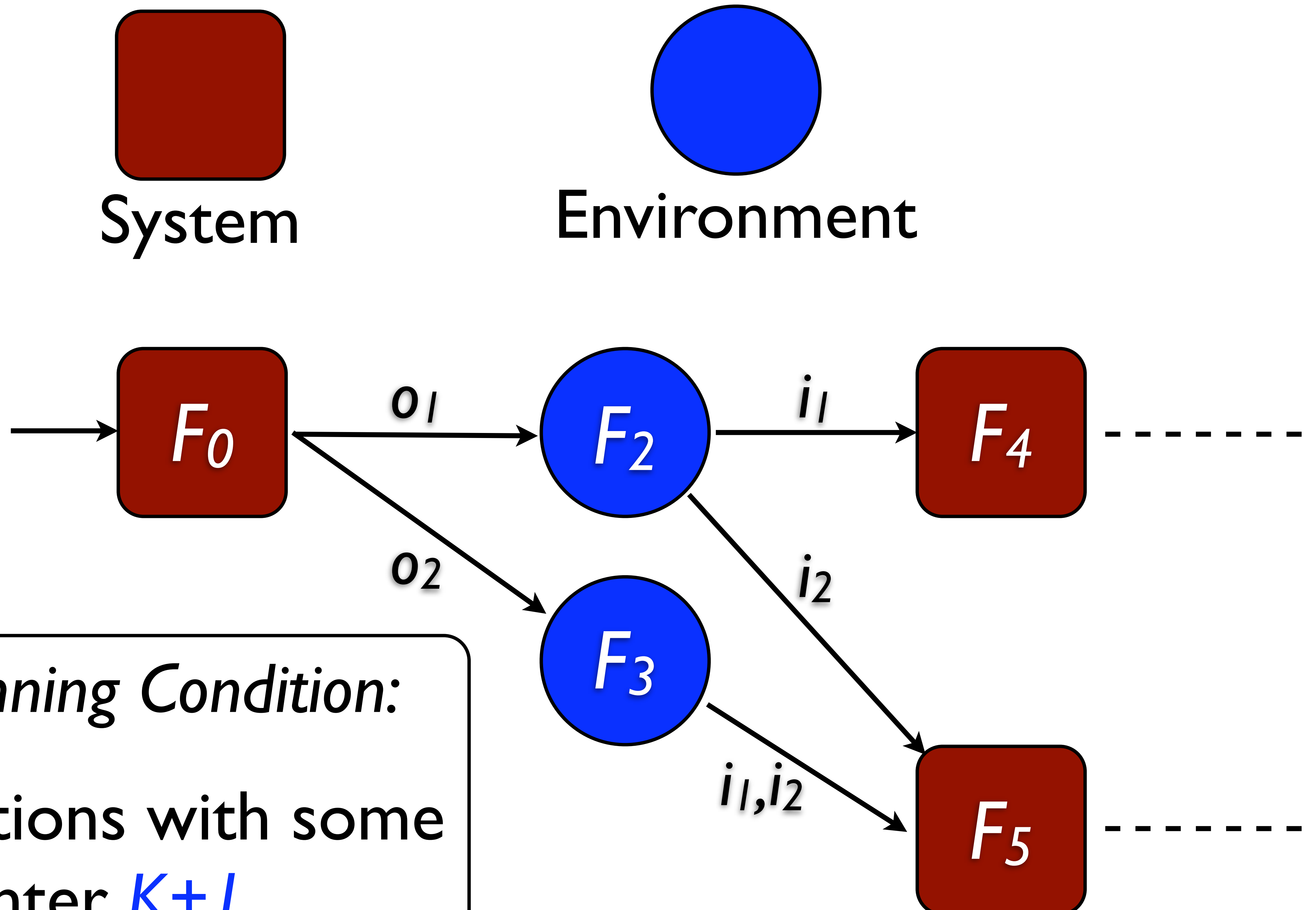
A is realizable

Construction: exptime and $K = O(2^{|\Phi|})$

Determinization

- *K*-Co-Büchi automata are easily determinizable
- extend subset construction with counters (up to $K+1$)
- states: Functions $F: Q \longrightarrow \{0,1,\ldots,K+1\}$

Playing on automata



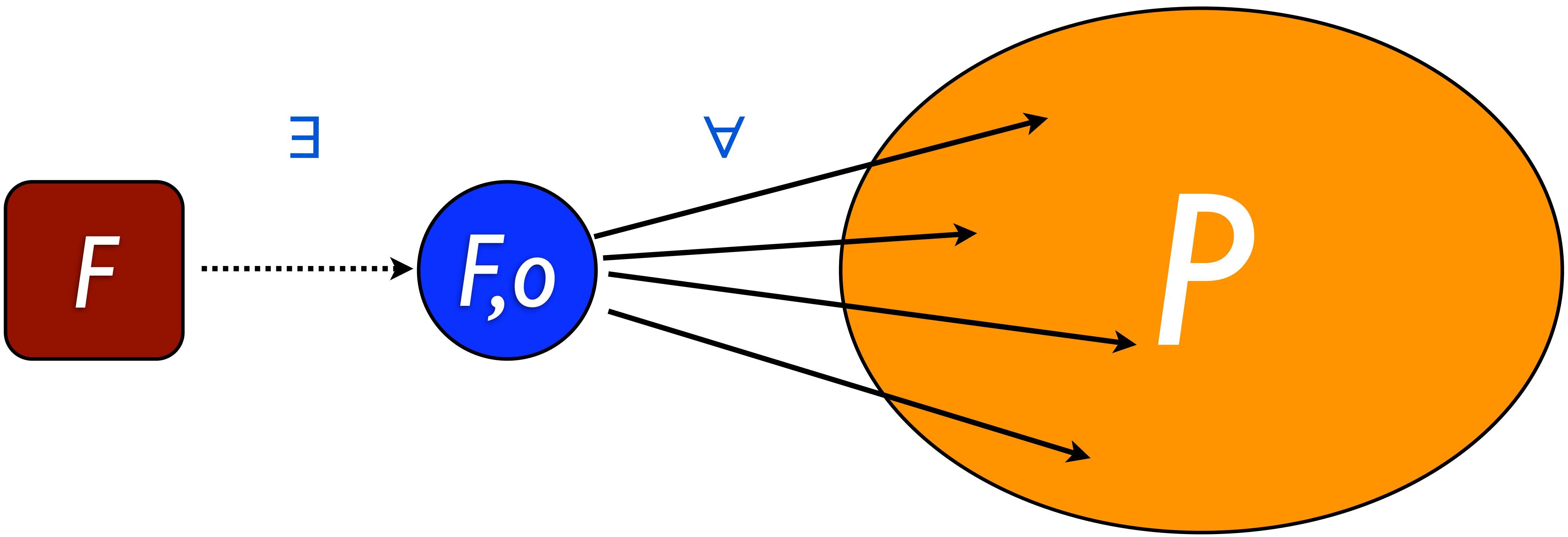
Safety Winning Condition:

Avoid functions with some counter $K+1$

Controllable Predecessors

- $P \subseteq \mathbb{F}$: subset of system positions

- safe controllable predecessors of P
  $Pre(P) = \{ F \mid \exists o \subseteq O, \forall F', ((F,o),F') \in T \Rightarrow F' \in P \}$



- greatest fixpoint Pre^* = *winning region for System*

Controllable Predecessors

1. partial order on counting functions:

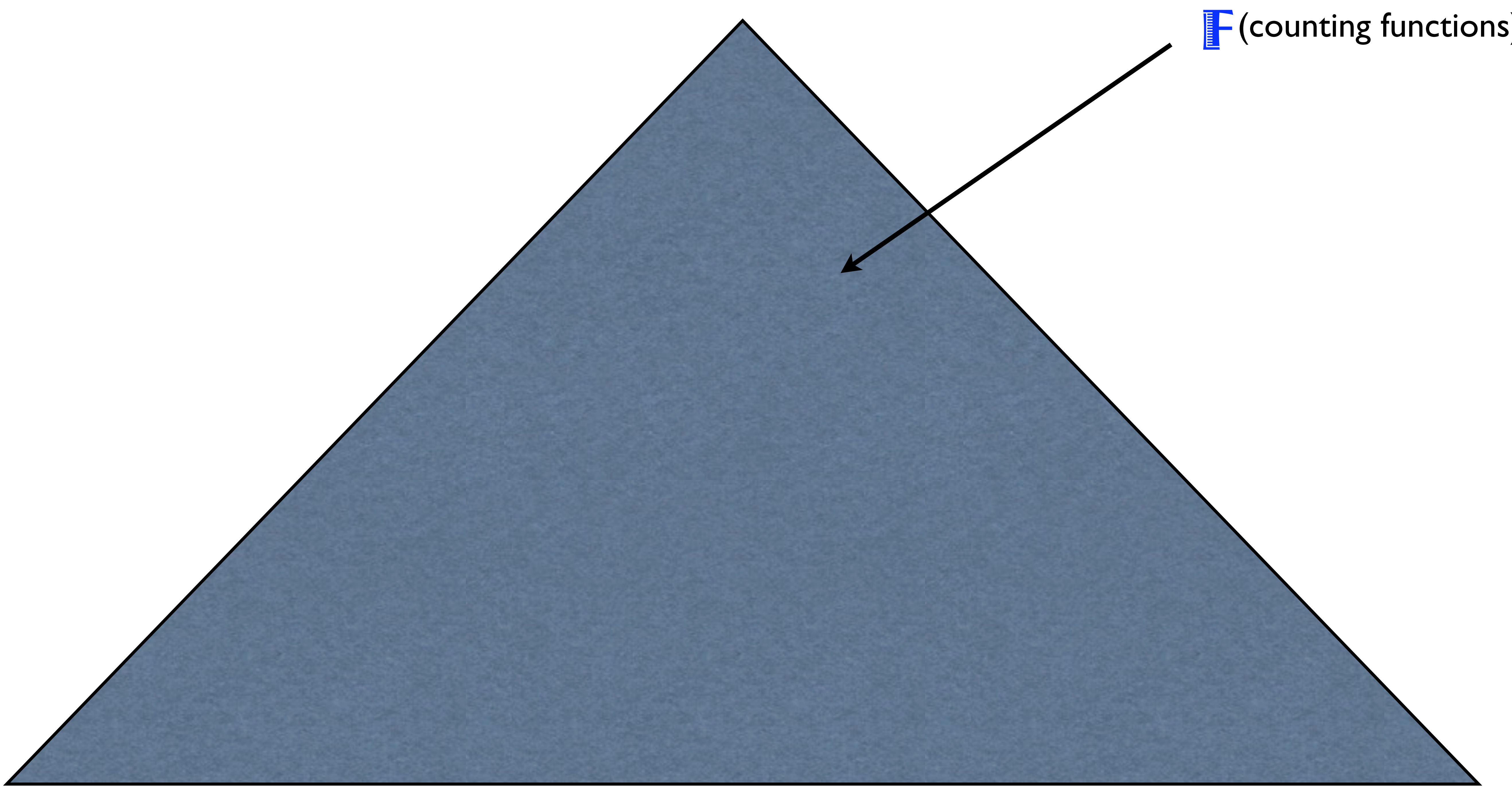
$$F \leq_d F' \text{ if } \forall q\text{: } F(q) \leq F'(q)$$

2. if System wins from *F'*, she also wins from

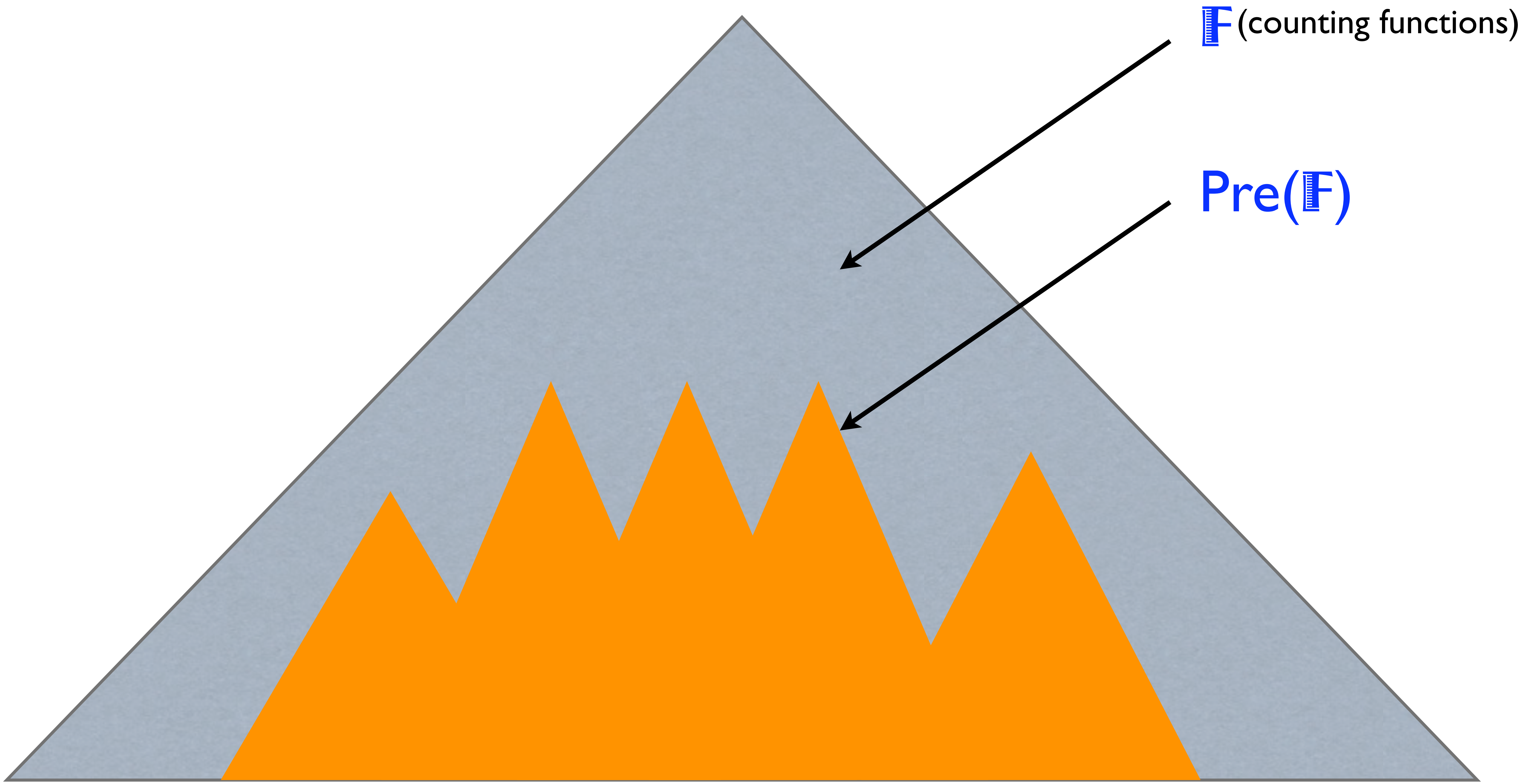
3. *Pre(.)* preserves *downward*-closed sets

4. represent each (downward) set of the fixpoint computation by its maximal elements

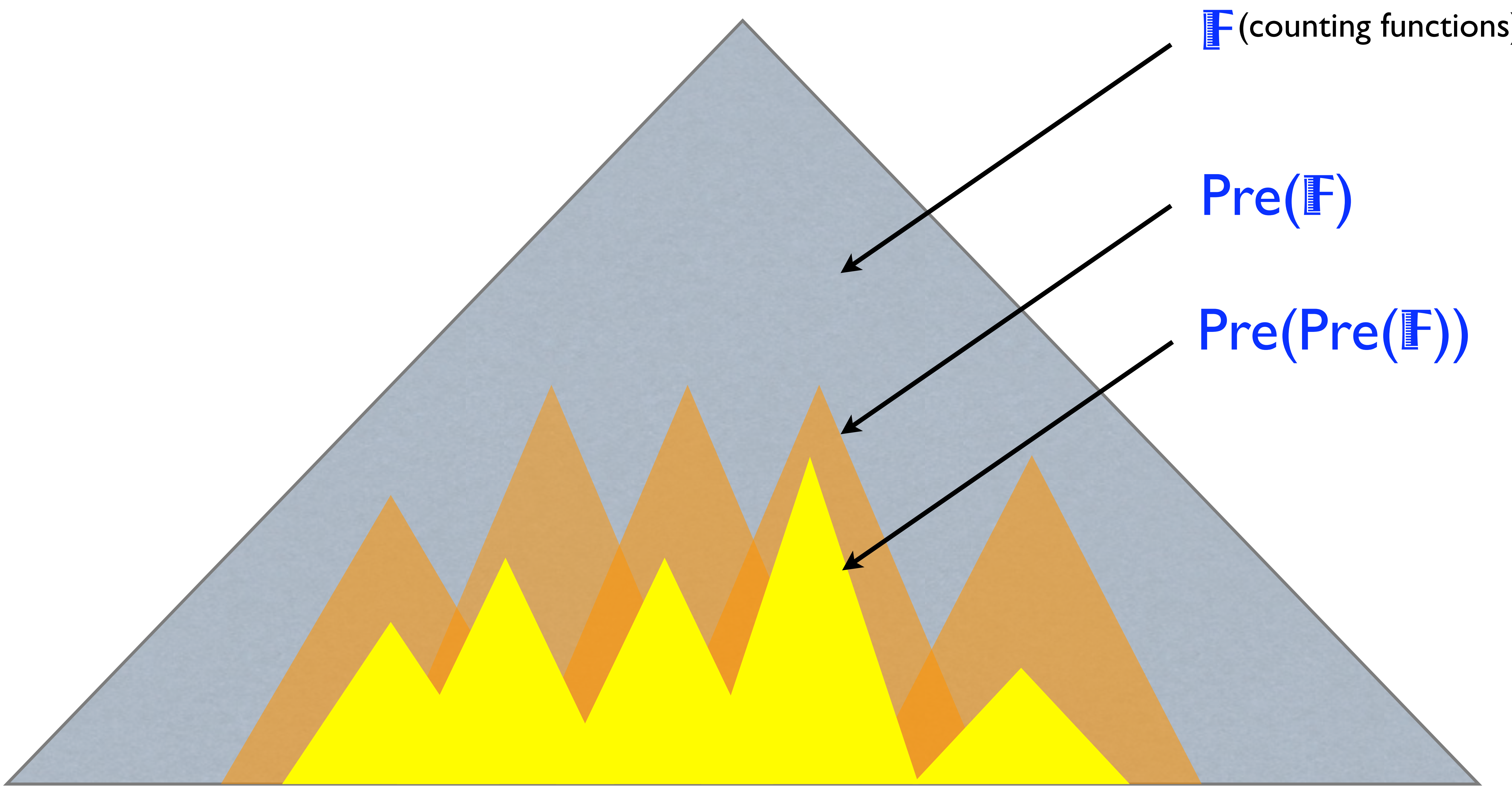
Symbolic Fixpoint Computation



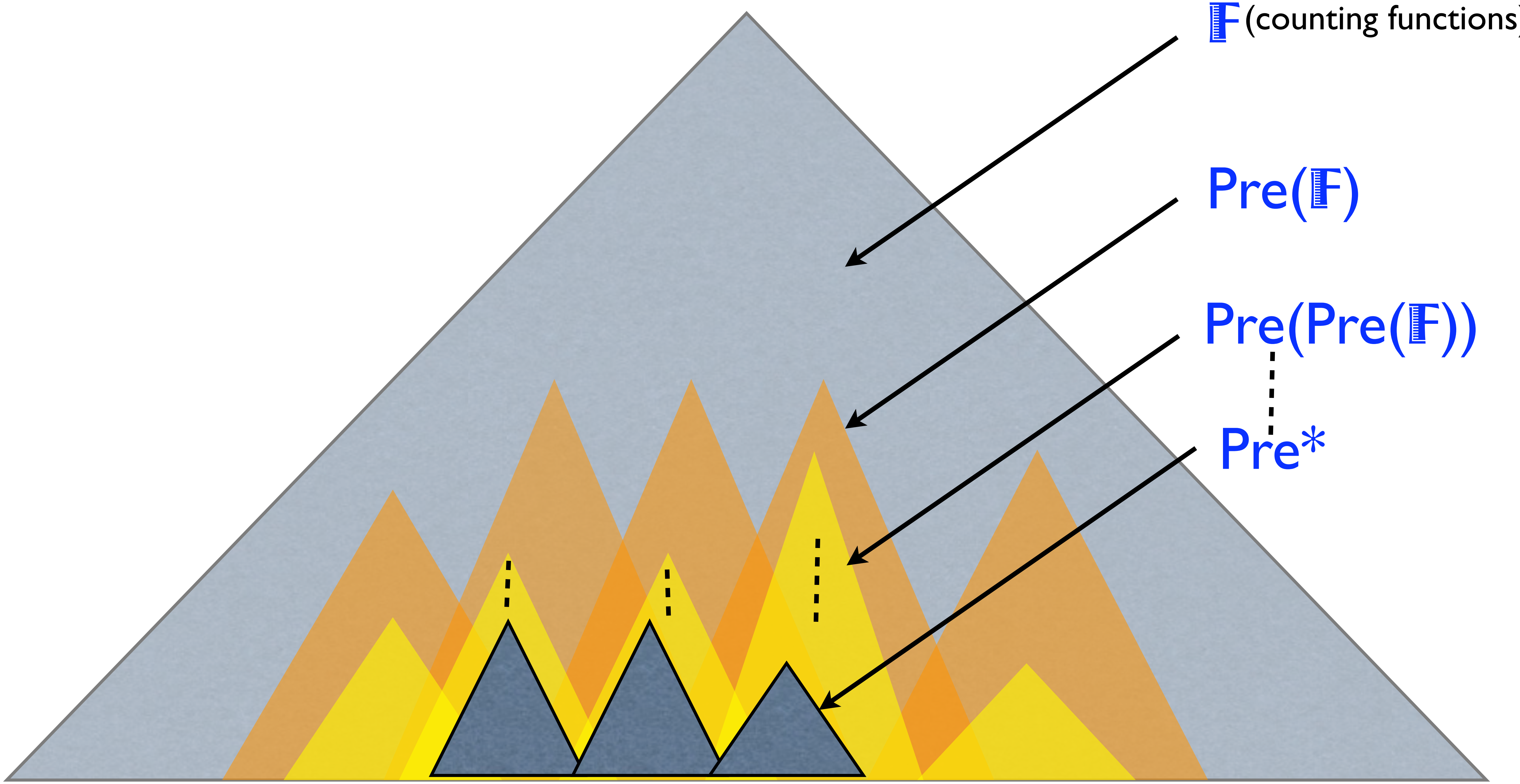
Symbolic Fixpoint Computation



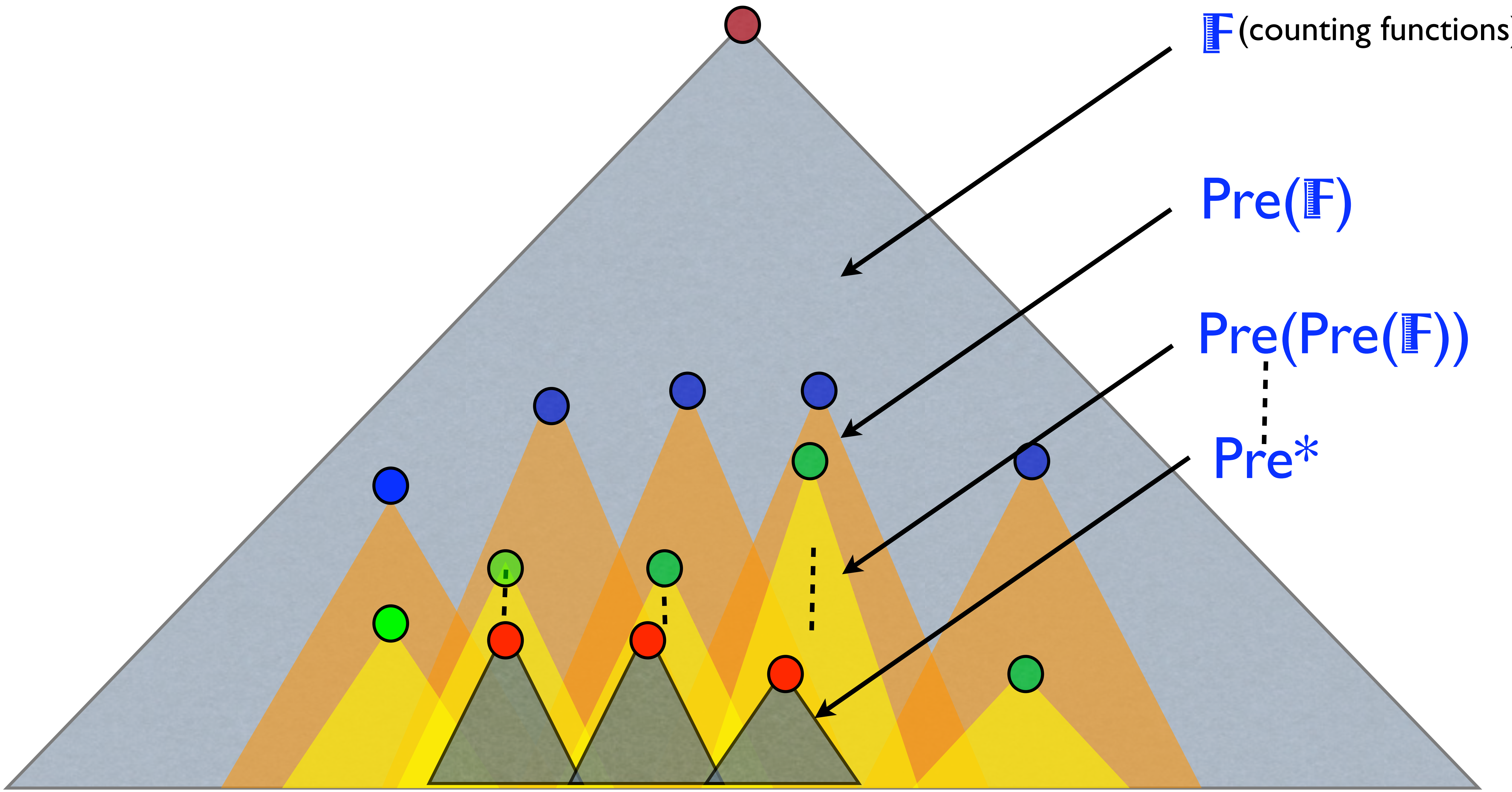
Symbolic Fixpoint Computation



Symbolic Fixpoint Computation



Symbolic Fixpoint Computation



Incremental Algorithm

- the bound **K** is very big (doubly exponential)
- if the spec is realizable with a "small" bound, it is realizable with a "big" bound
- iterate over $k=0,1,\ldots,$**K**

Incremental Algorithm

- the bound **K** is very big (doubly exponential)
- if the spec is realizable with a "small" bound, it is realizable with a "big" bound
- iterate over *k=0, 1,...,*,**K**

Not reasonable for unrealizable specifications

Incremental Algorithm

- the
- if t
  rea
- ite

Not

ions

But by Martin's determination theorem:

ϕ is unrealizable for the System iff $\neg\phi$ is realizable for the Environment.

Experiments

- implementation in Perl (as Lily)
- if the spec is realizable, output a Moore machine that realizes it
- formula to automata construction borrowed from Lily (based on Wring [Somenzi, Bloem])
- significantly faster on all realizable Lily's examples
- bottleneck: formula to automaton construction

Example

```
assume StoB_REQ1=0;

assume G((StoB_REQ0=1 * BtoS_ACK0=0) -> X(StoB_REQ0=1));
assume G((StoB_REQ1=1 * BtoS_ACK1=0) -> X(StoB_REQ1=1));

assume G(BtoS_ACK0=1 -> X(StoB_REQ0=0));
assume G(BtoS_ACK1=1 -> X(StoB_REQ1=0));


(BtoS_ACK0=0 *
 G( (StoB_REQ0=0 * X(StoB_REQ0=1)) -> X(BtoS_ACK0=0 * X(F(BtoS_ACK0=1))) ) *
 G( (BtoS_ACK0=0 * X(StoB_REQ0=0)) -> X(BtoS_ACK0=0) ) *
 G(BtoS_ACK0=0 + BtoS_ACK1=0)
);


(BtoS_ACK1=0 *
 G( (StoB_REQ1=0 * X(StoB_REQ1=1)) -> X(BtoS_ACK1=0 * X(F(BtoS_ACK1=1))) ) *
 G( (BtoS_ACK1=0 * X(StoB_REQ1=0)) -> X(BtoS_ACK1=0) ) *
 G(BtoS_ACK0=0 + BtoS_ACK1=0)
);

assume RtoB_ACK0=0;
assume RtoB_ACK1=0;

assume G(BtoR_REQ0=0 -> X(RtoB_ACK0=0));
#assume G((BtoR_REQ0=1 * RtoB_ACK0=1) -> X(RtoB_ACK0=1));

assume G(BtoR_REQ1=0 -> X(RtoB_ACK1=0));
#assume G((BtoR_REQ1=1 * RtoB_ACK1=1) -> X(RtoB_ACK1=1));


assume G(BtoR_REQ0=1 -> X(F(RtoB_ACK0=1)));
assume G(BtoR_REQ1=1 -> X(F(RtoB_ACK1=1)));


(BtoR_REQ0=0 *
 G(RtoB_ACK0=1 -> X(BtoR_REQ0=0)) *
 G((BtoR_REQ0=1 * RtoB_ACK0=0) -> X(BtoR_REQ0=1)) *
 G((BtoR_REQ0=1 * X(BtoR_REQ0=0)) -> X( BtoR_REQ0=0 U (BtoR_REQ0=0 * BtoR_REQ1=1))) *
 G(F(BtoR_REQ0=1)) *
 G((BtoR_REQ0=0) + (BtoR_REQ1=0))
);

(BtoR_REQ1=0 *
 G(RtoB_ACK1=1 -> X(BtoR_REQ1=0)) *
 G((BtoR_REQ1=1 * RtoB_ACK1=0) -> X(BtoR_REQ1=1)) *
 G((BtoR_REQ1=1 * X(BtoR_REQ1=0)) -> X( BtoR_REQ1=0 U (BtoR_REQ1=0 * BtoR_REQ0=1))) *
 G(F(BtoR_REQ1=1)) *
 G((BtoR_REQ0=0) + (BtoR_REQ1=0))
);
```

```
(BtoS_ACK0=0 *
 G( (StoB_REQ0=0 * X(StoB_REQ0=1)) -> X(BtoS_ACK0=0 * X(F(BtoS_ACK0=1))) ) *
 G( (BtoS_ACK0=0 * X(StoB_REQ0=0)) -> X(BtoS_ACK0=0) ) *
 G(BtoS_ACK0=0 + BtoS_ACK1=0)
);


(BtoS_ACK1=0 *
 G( (StoB_REQ1=0 * X(StoB_REQ1=1)) -> X(BtoS_ACK1=0 * X(F(BtoS_ACK1=1))) ) *
 G( (BtoS_ACK1=0 * X(StoB_REQ1=0)) -> X(BtoS_ACK1=0) ) *
 G(BtoS_ACK0=0 + BtoS_ACK1=0)
);



(BtoR_REQ0=0 *
 G(RtoB_ACK0=1 -> X(BtoR_REQ0=0)) *
 G((BtoR_REQ0=1 * RtoB_ACK0=0) -> X(BtoR_REQ0=1)) *
 G((BtoR_REQ0=1 * X(BtoR_REQ0=0)) -> X( BtoR_REQ0=0 U (BtoR_REQ0=0 * BtoR_REQ1=1))) *
 G(F(BtoR_REQ0=1)) *
 G((BtoR_REQ0=0) + (BtoR_REQ1=0))
);

(BtoR_REQ1=0 *
 G(RtoB_ACK1=1 -> X(BtoR_REQ1=0)) *
 G((BtoR_REQ1=1 * RtoB_ACK1=0) -> X(BtoR_REQ1=1)) *
 G((BtoR_REQ1=1 * X(BtoR_REQ1=0)) -> X( BtoR_REQ1=0 U (BtoR_REQ1=0 * BtoR_REQ0=1))) *
 G(F(BtoR_REQ1=1)) *
 G((BtoR_REQ1=0) + (BtoR_REQ1=1))
);
```

Future Work ...

- compositionnality
- avoid automata construction to handle larger formulas

Future Work ...

- compositionnality
- avoid automata construction to handle larger formulas

... Thank You