GASICS

Games for Analysis and Synthesis of Interactive Computational Systems

Jean-François Raskin
Université Libre de Bruxelles
Belgium

LogICCC kick-off meeting, Prague, Oct. 6, 2008

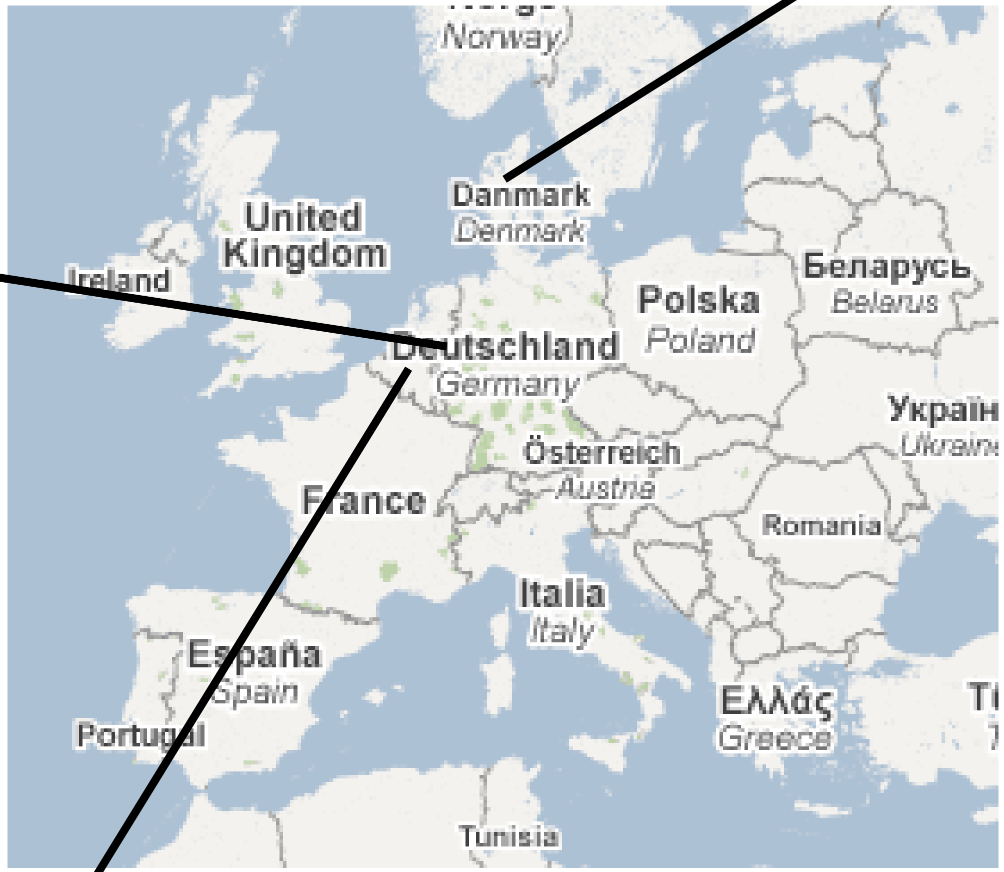
Plan of the talk

- Overview of the consortium
- Interactive computational systems (aka Reactive Systems) and motivations for “formal” methods
- Verification and synthesis
- Why synthesis can be reduced to a game problem ?
- Examples of important open problems in the area
- Why is our approach innovative ?
- Why is our project “exciting” ?

The GASICS consortium

U Aalborg
(FNU)

RWTH Aachen
(DFG)

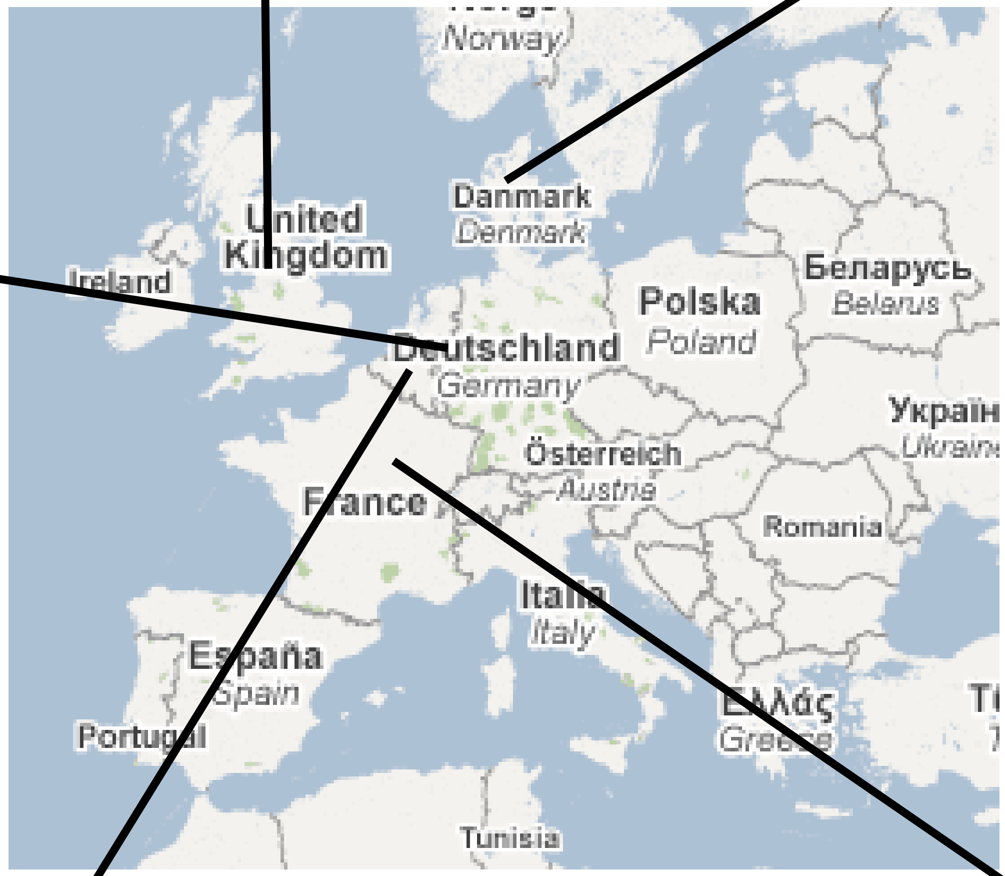


Université Libre de Bruxelles
(FNRS)

U Warwick

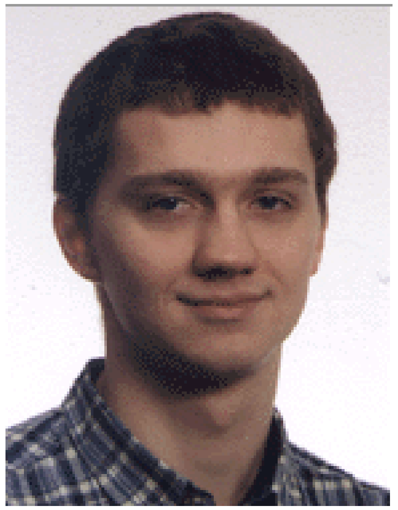
U Aalborg
(FNU)

RWTH Aachen
(DFG)



Université Libre de Bruxelles
(FNRS)

U Paris 7
ENS Cachan



U Warwick

Prof. Marcin Jurdzinski

U Aalborg

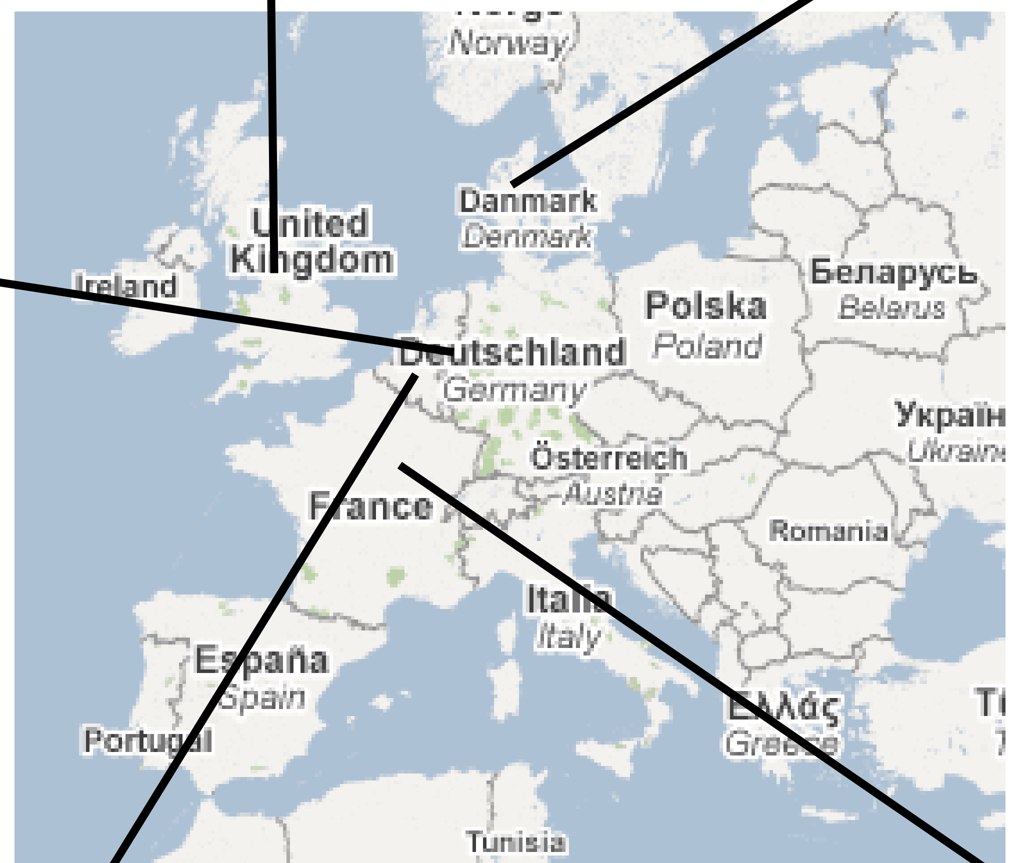
(FNU)

Prof. Kim Larsen

RWTH Aachen

(DFG)

Prof. Wolfgang Thomas.



Université Libre de Bruxelles

(FNRS)

Prof. Jean-François Raskin

U Paris 7

ENS Cachan

Dr. Jean-Eric Pin

Dr. Nicolas Markey

What are Interactive Computational Systems (aka reactive systems) ?

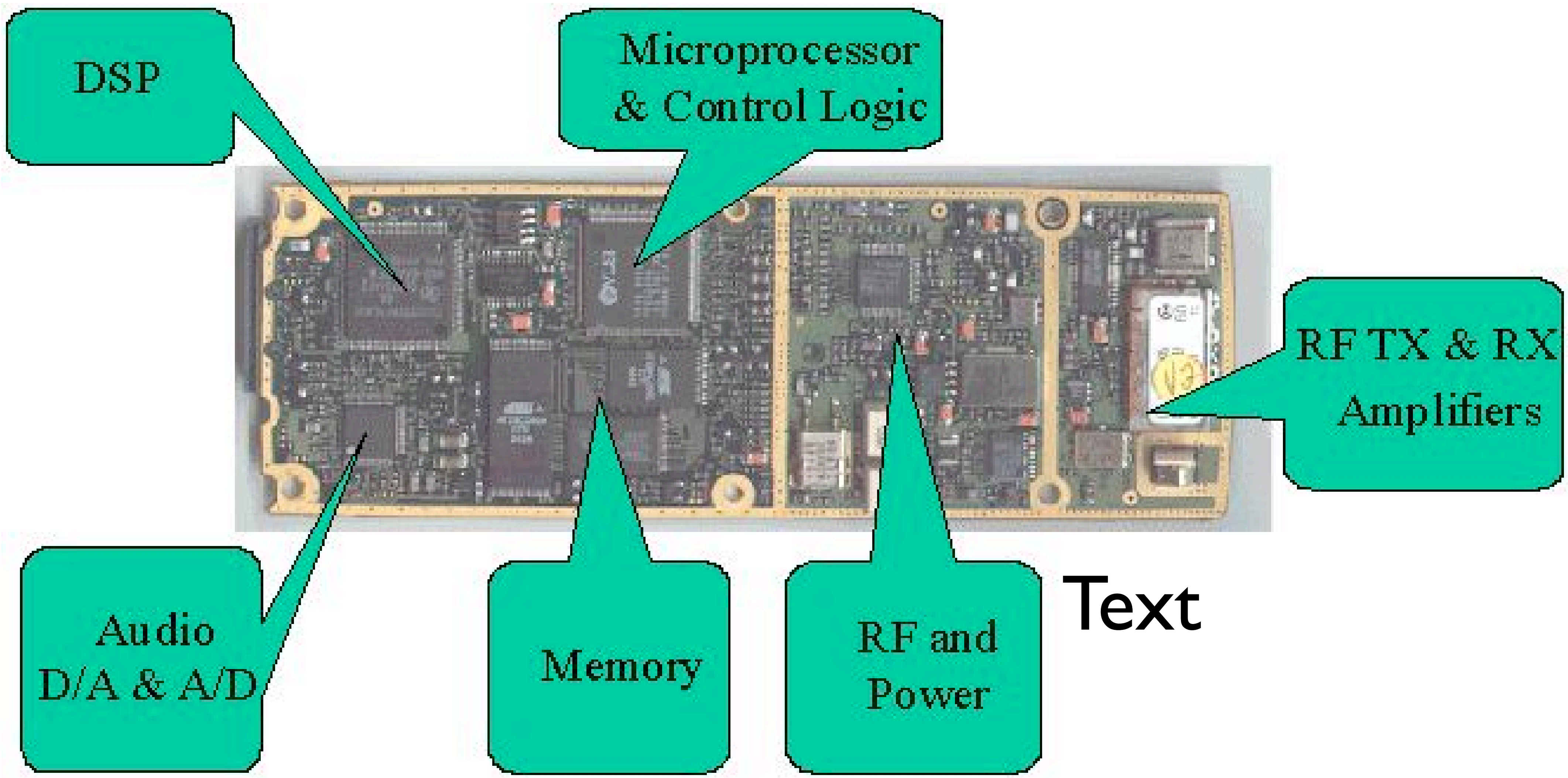

777
BOEING 777-300

300 horses power

100 processors

Cellular Phone

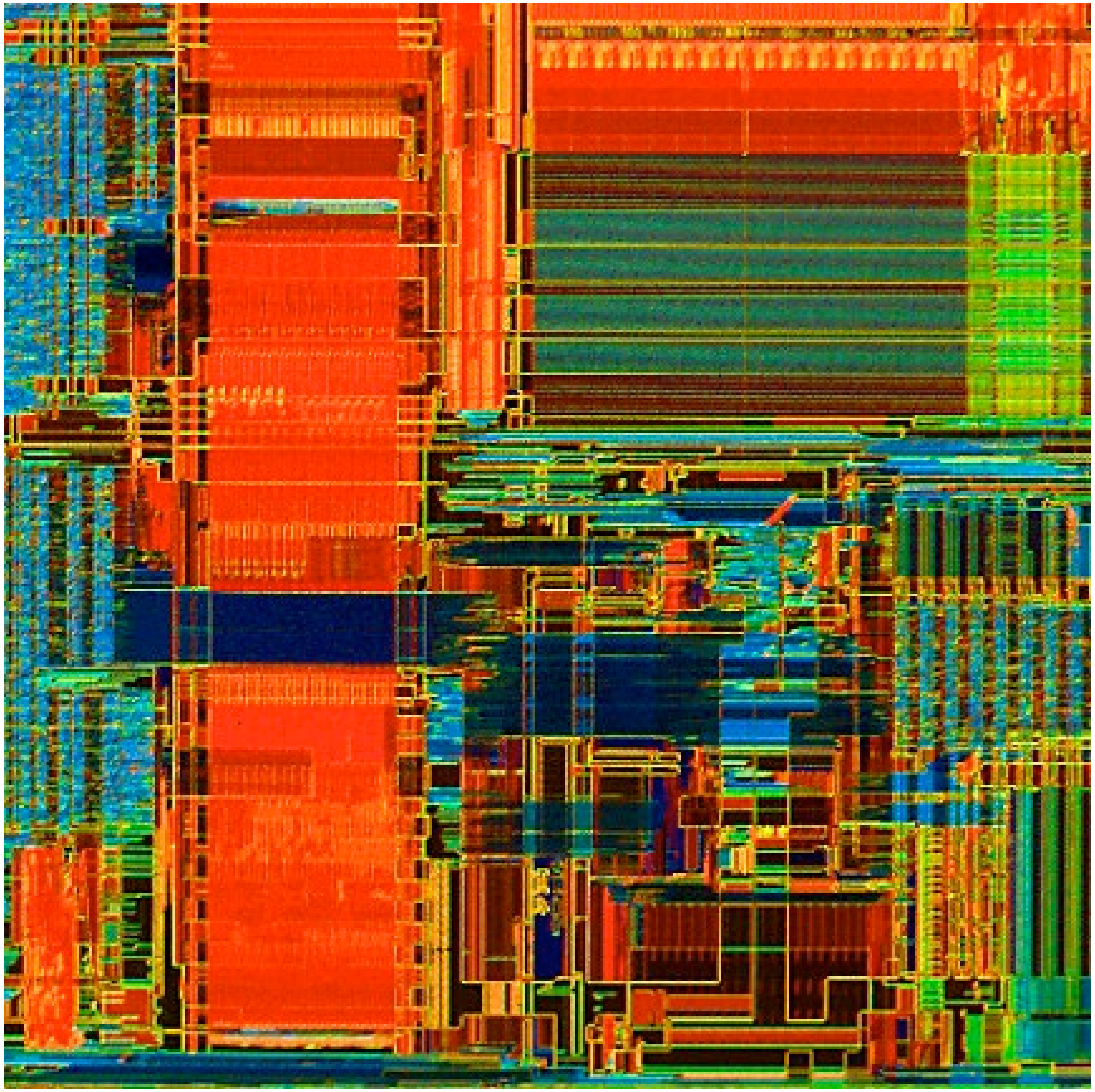
more and more software

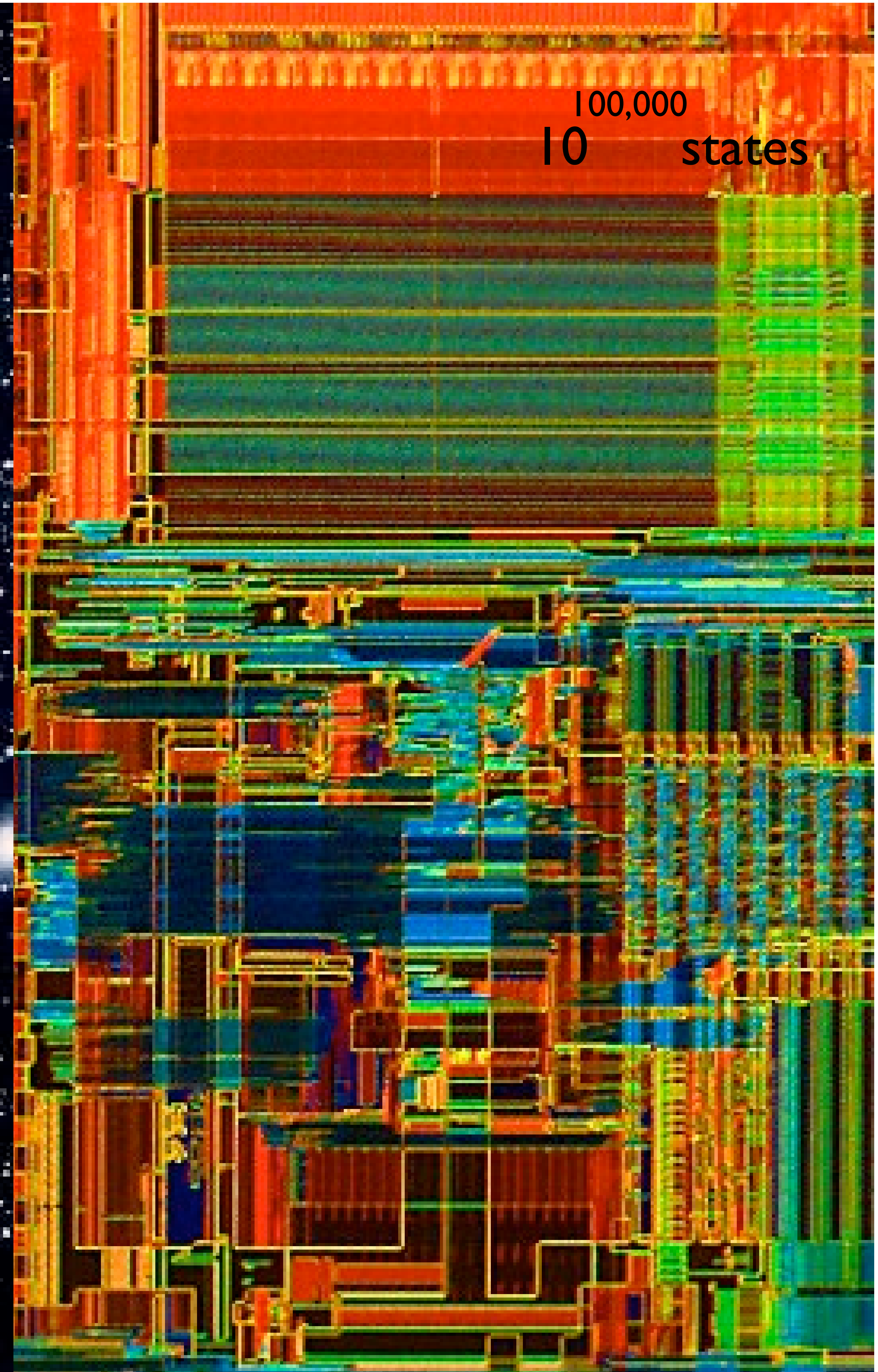


Text

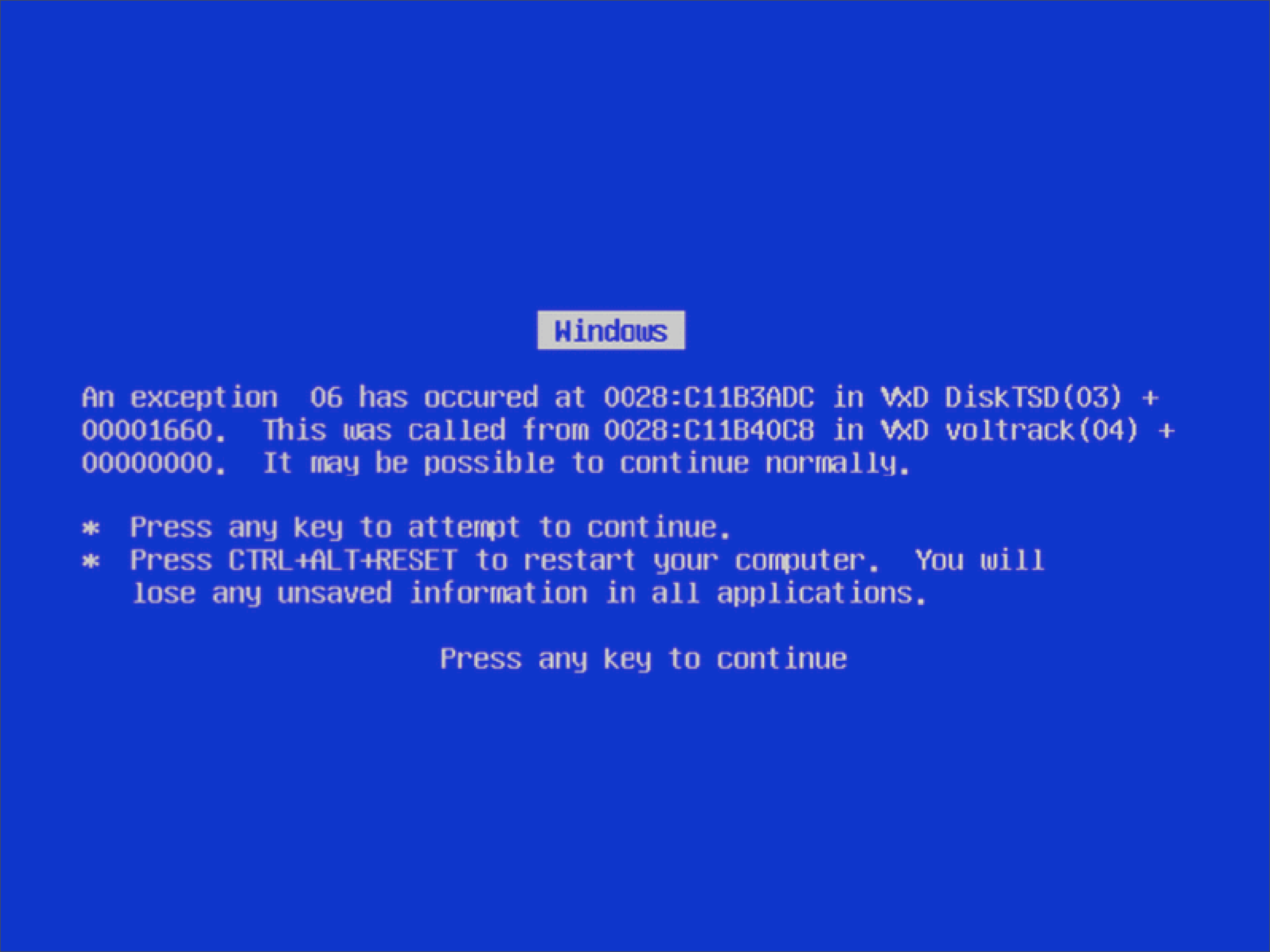
Concurrency: several hardware and software components
Heterogeneity: digital (discrete time) and analog (continuous time)
Uncertainty: environment, exceptions handling

Concurrency : 300 000 logical gates




$10^{100,000}$ states


French Guniea, june 4, 1996


Windows
An exception 06 has occured at 0028:C11B3ADC in VxD DiskTSD(03) + 00001660. This was called from 0028:C11B40C8 in VxD voltrack(04) + 00000000. It may be possible to continue normally.
* Press any key to attempt to continue.
* Press CTRL+ALT+RESET to restart your computer. You will lose any unsaved information in all applications.
Press any key to continue

Reactive systems
Interactive computational systems

- Reactive systems are systems that maintain a continuous **interaction** with their environment, and they usually have several of the following properties:

  - they are non-terminating systems (processes);

  - they have to respect or enforce real-time properties;

  - they have to cope with concurrency (several processes are executing concurrently);

  - they are often embedded into an complex and safety critical environments.

- ... as a result: the specifications that have to meet RS are often **very complex** and as a result RS are **difficult to design correctly !**

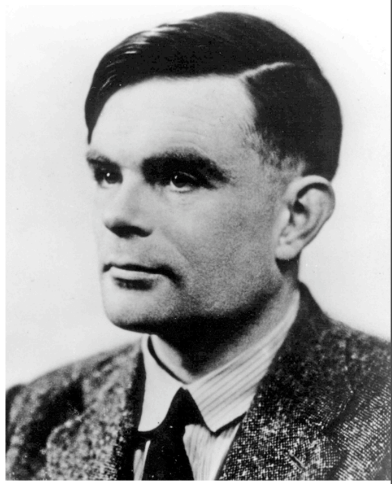
Need for verification

- ... as they are difficult to develp correctly !
- ... and often safety critical !

⇒ we should verify them !

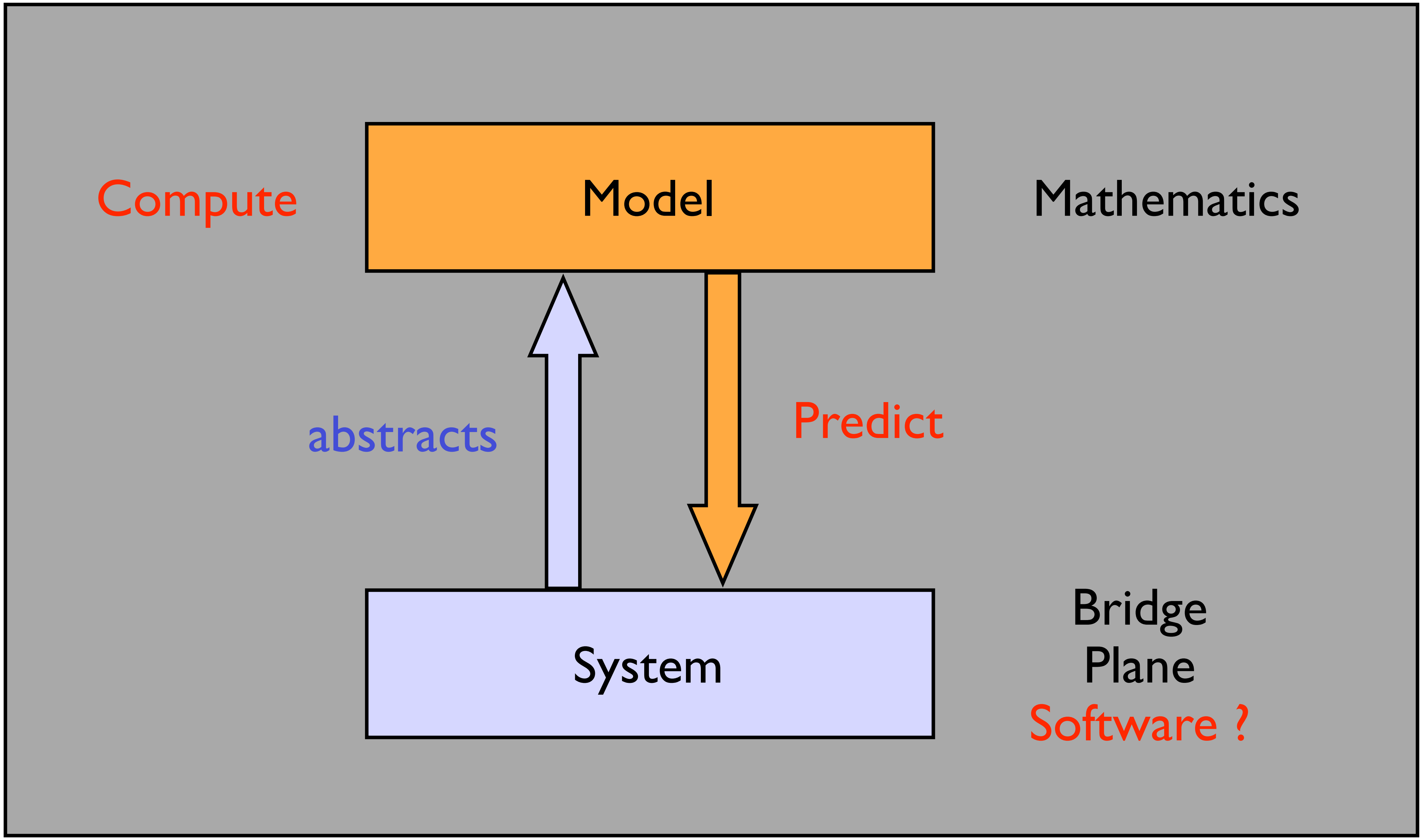
⇒ or construct them in a way that ensures their **correctness** !

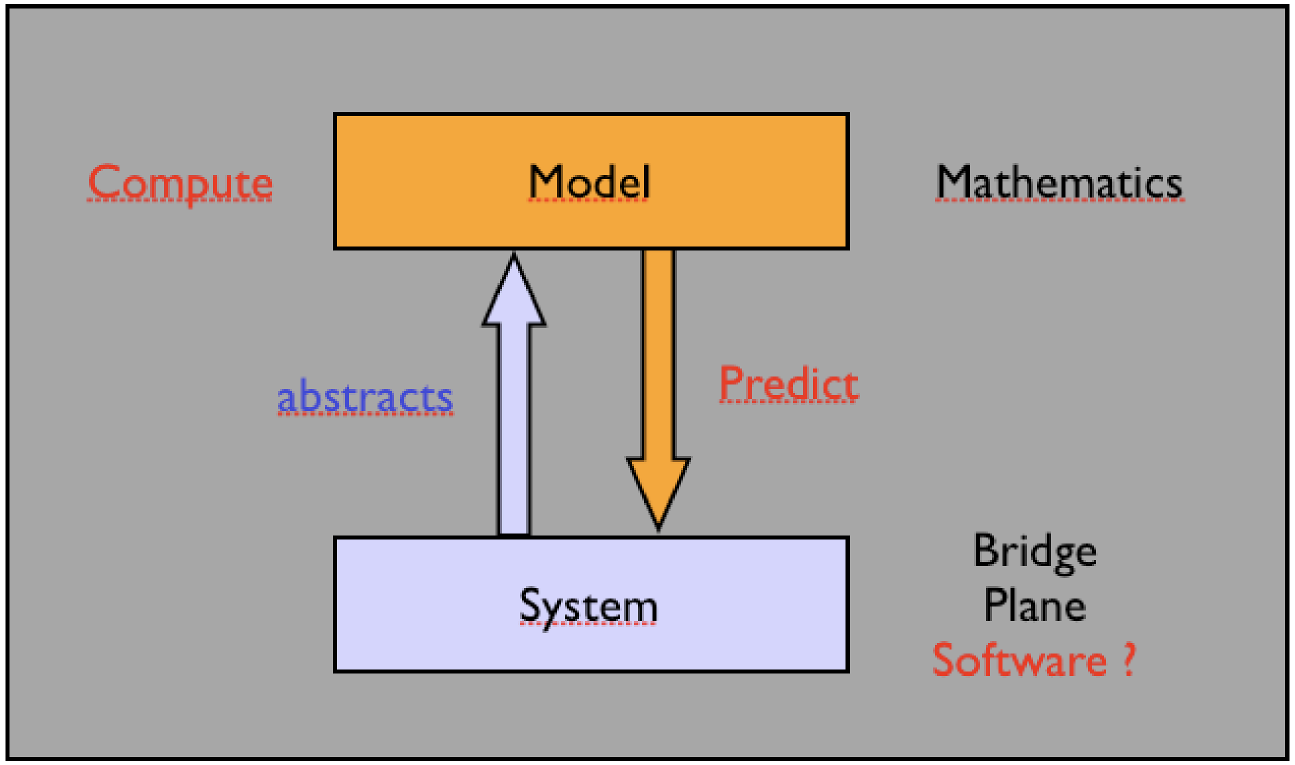
The old impossible dream of computer scientists



- As soon as 1936, Turing has shown that **fully automatic verification** of programs is impossible
(a.o. program termination is **undecidable**).
- Are programs or computer systems **too complex** to be analyzed using automated tools ? Yes, ...
- and **no** ...

How do we cope with complexity in science ?





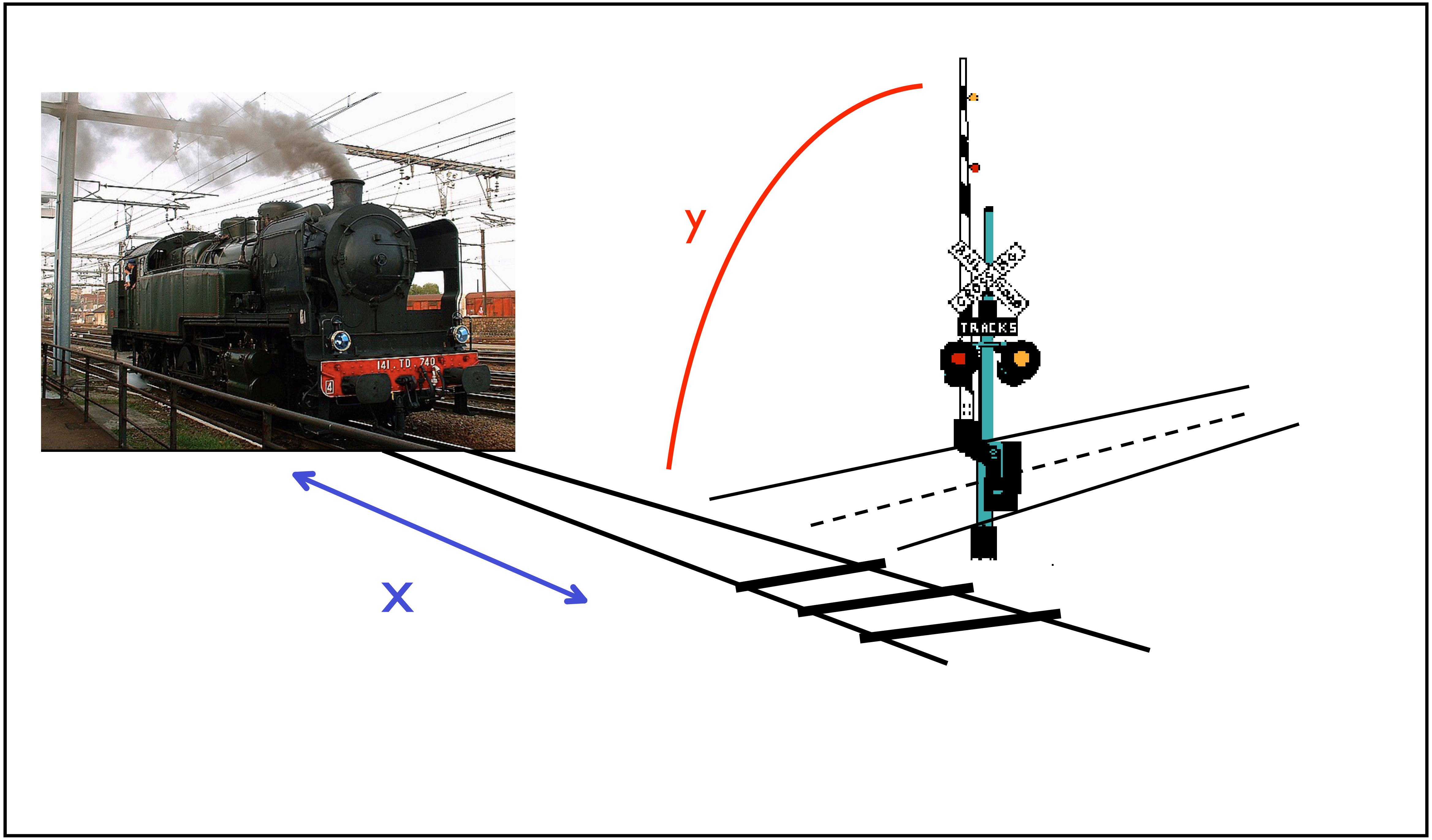
- Model construction: capture the essential aspects of the system (sometimes automatically);

- Model verification: algorithms to analyze models.

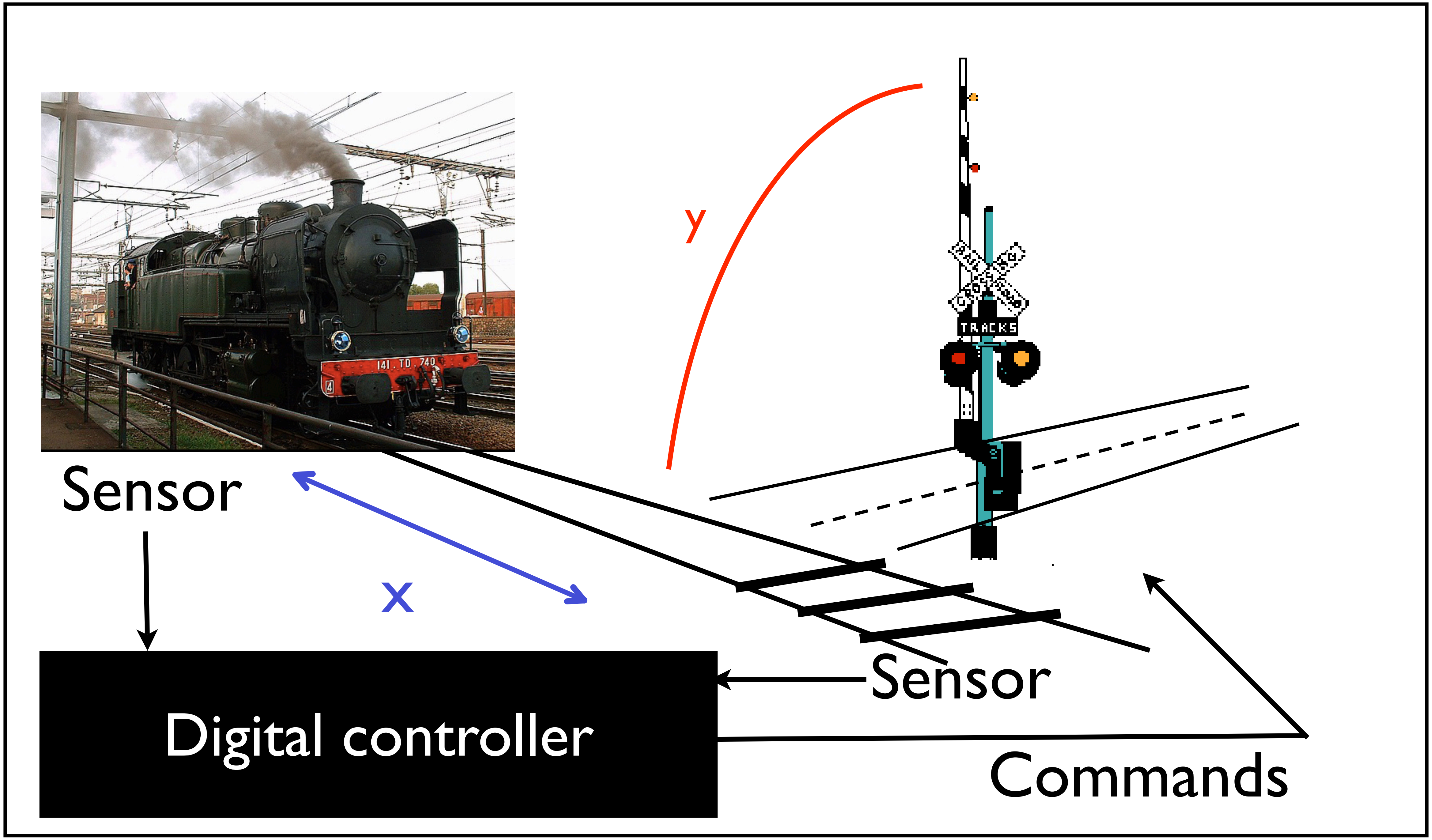
(Clarke, Emerson, and Sifakis received the **2008-ACM Turing Award** for their seminal works in that area).

Formal models of reactive systems

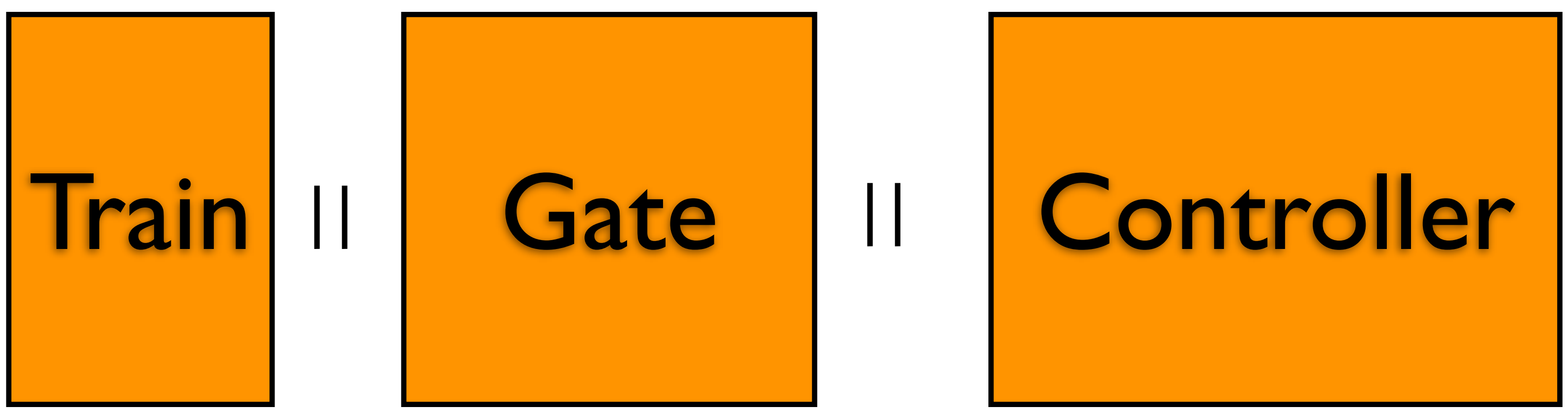
A toy example



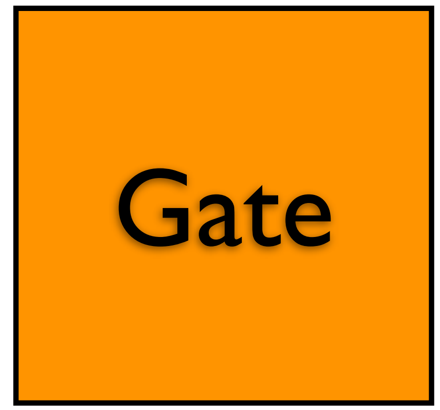
A toy example



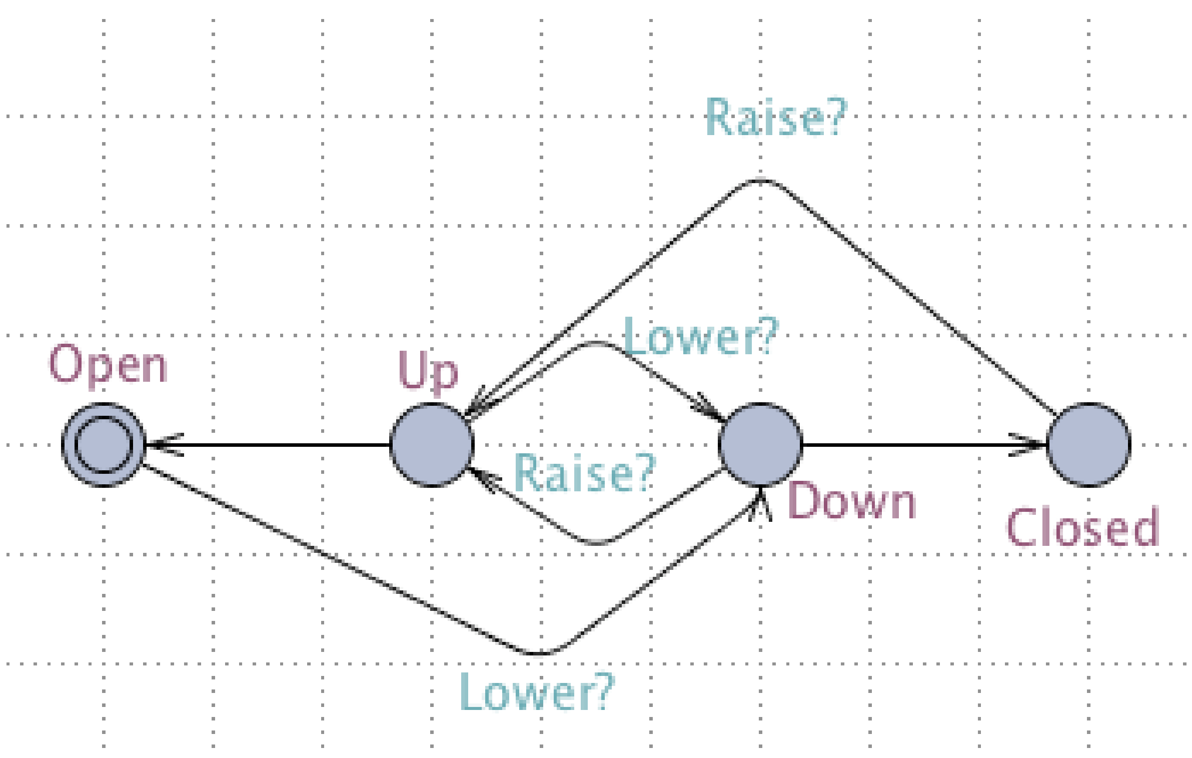
Models of reactive systems



Models of reactive systems



The model gathers information about the possible **states** of the gate, and the possible **evolutions** (triggered by events) of the states along time.



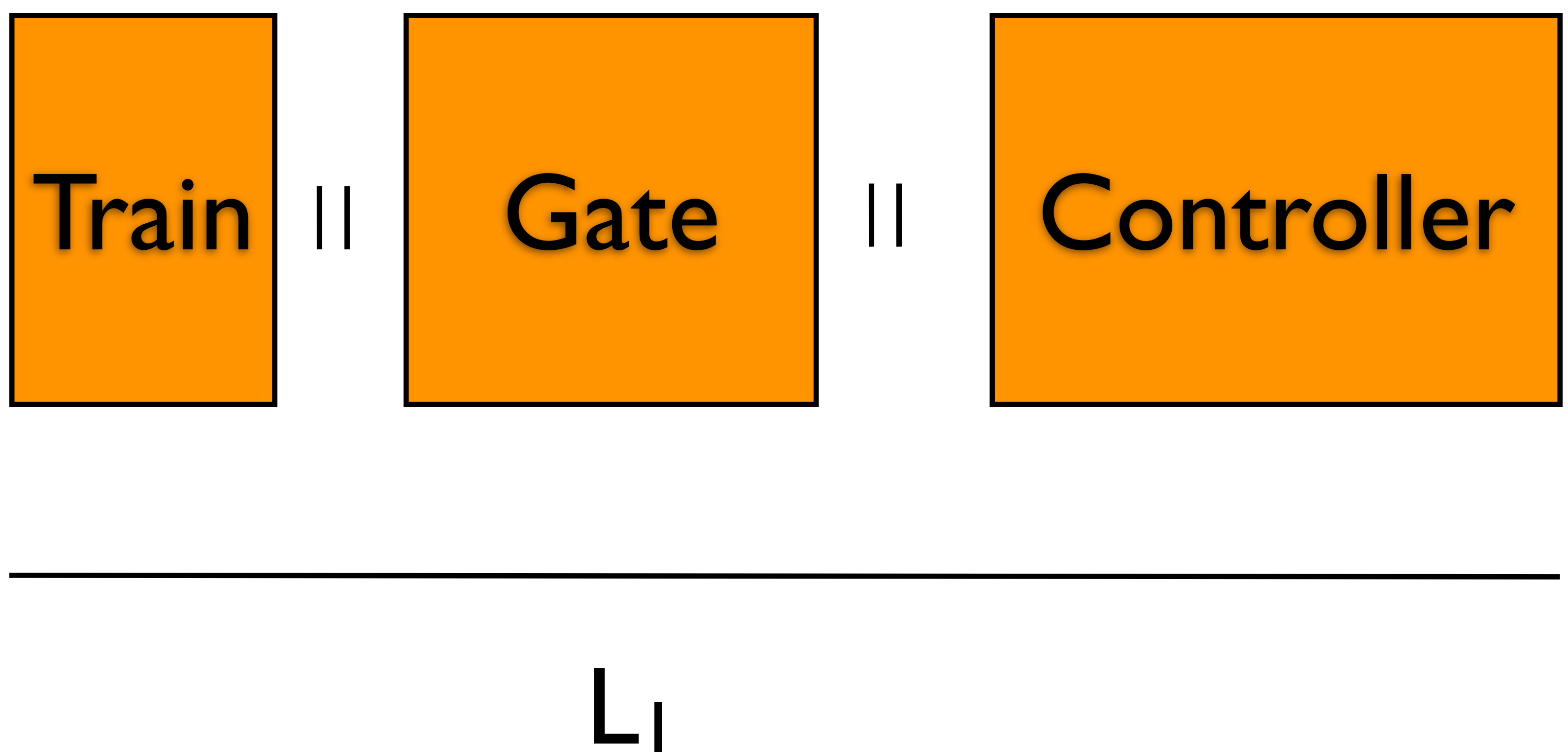
Defines sequences of states-events

Open —Lower?→ Down —ε→ Closed —Raise?→ Up ...
Open —Lower?→ Down —Raise?→ Up —Lower?→ Down ...
...

The **language** of the gate is:

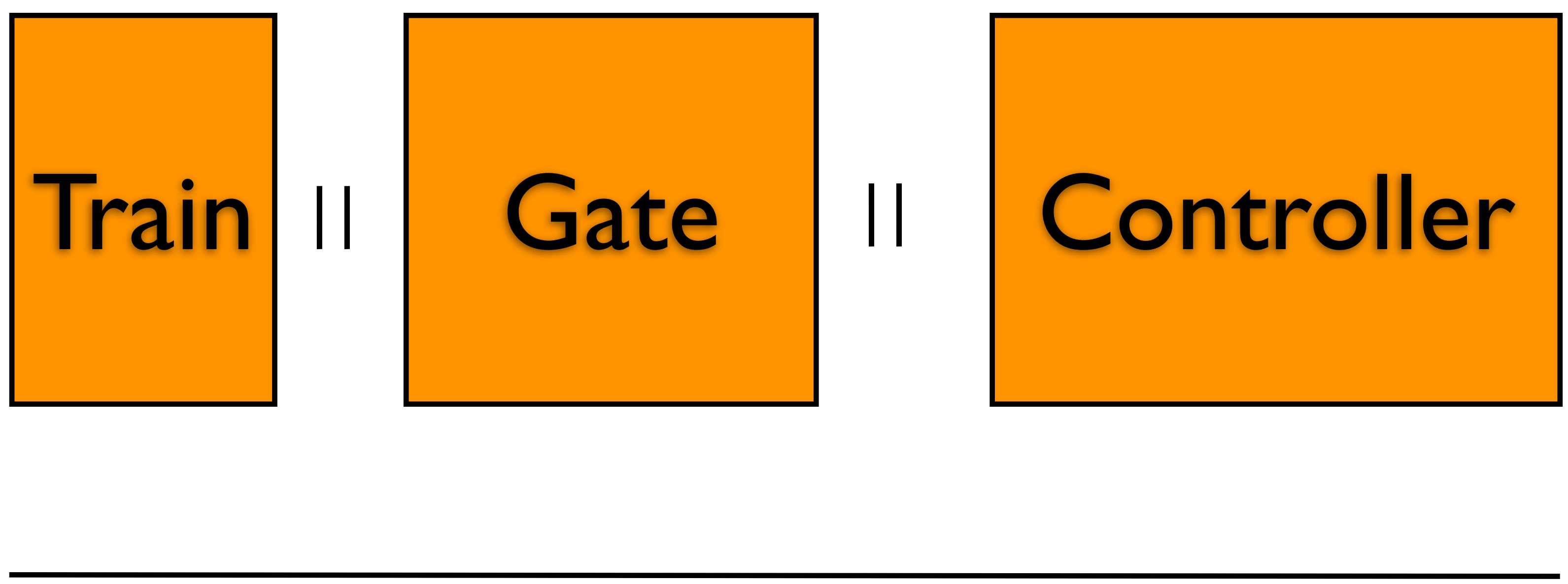
{Open —Lower?→ Down —ε→ Closed —Raise?→ Up ...,
Open —Lower?→ Down —Raise?→ Up —Lower?→ Down ...
...}

Models of reactive systems



Train, Gate and Controller are modeled as automata that synchronize on common events. The resulting model is a (huge) **graph** whose paths are the possible behaviors of the system.

Models of reactive systems



L_1

Compact representation of a language
(=infinite set of infinite traces, aka words).

Properties

An example of a property for our toy system:

“in all traces ***t***, on all states of the trace ***t*** if the train is within the crossing, the gate is closed”.

Properties

An example of a property for our toy system:

"in all traces ***t***, on all states of the trace ***t*** if the train is within the crossing, the gate is closed".

☛ Property = set of traces.

Properties

An example of a property for our toy system:

“in all traces ***t***, on all states of the trace ***t*** if the train is within the crossing, the gate is closed”.

☛ Property = set of traces.

☛ Formalized as an **automaton**, or a formula in a **temporal logic**.

$$\Box\,(\, In \rightarrow Closed \,)$$

Systems and properties

Train || Gate || Controller

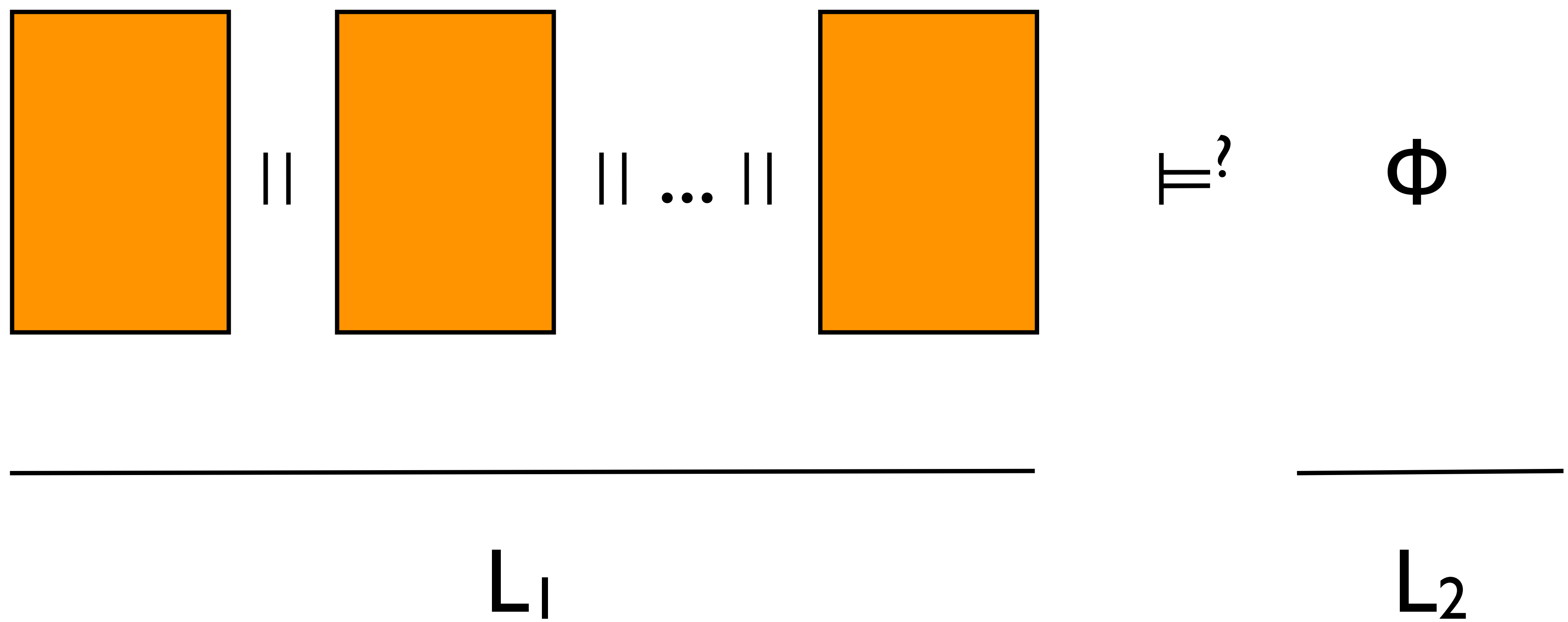
$\square\,(\mathit{In} \rightarrow \mathit{Closed})$

L_1

L_2

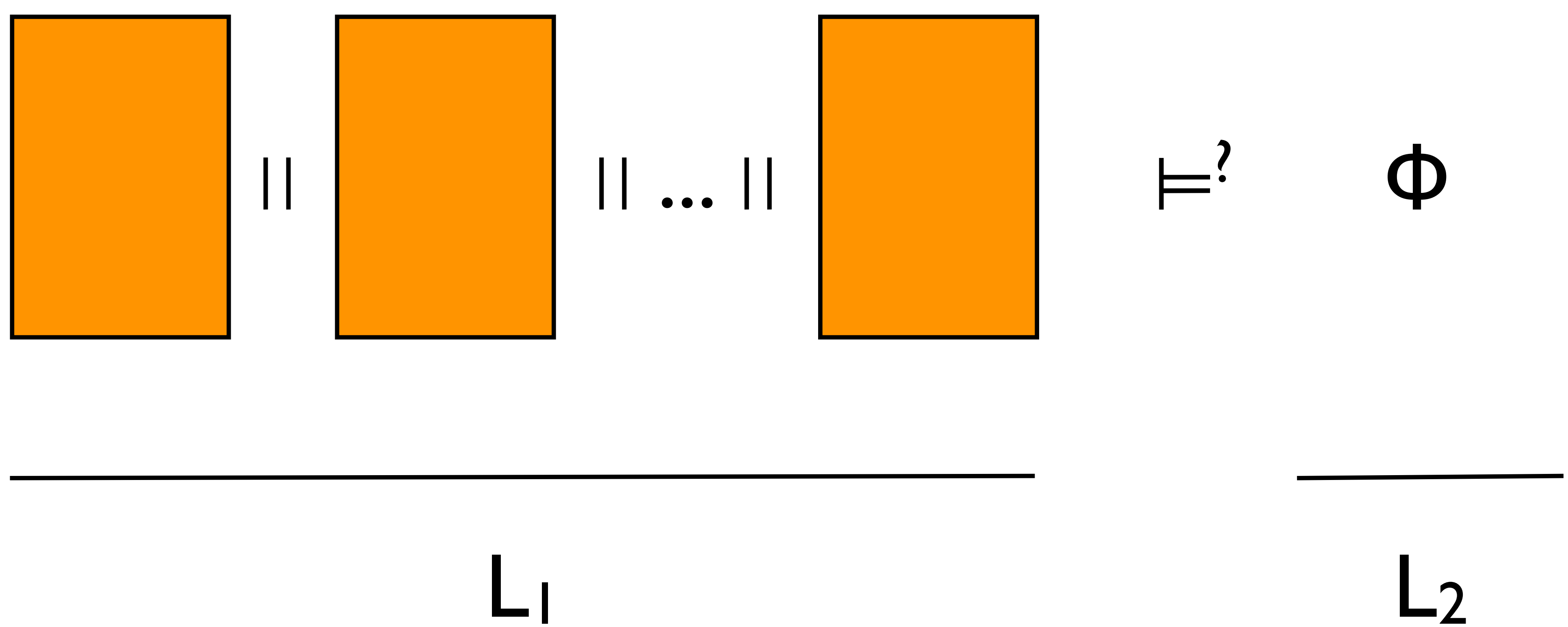
Verification and Synthesis

Verification



Question: L1 $\subseteq^?$ L2

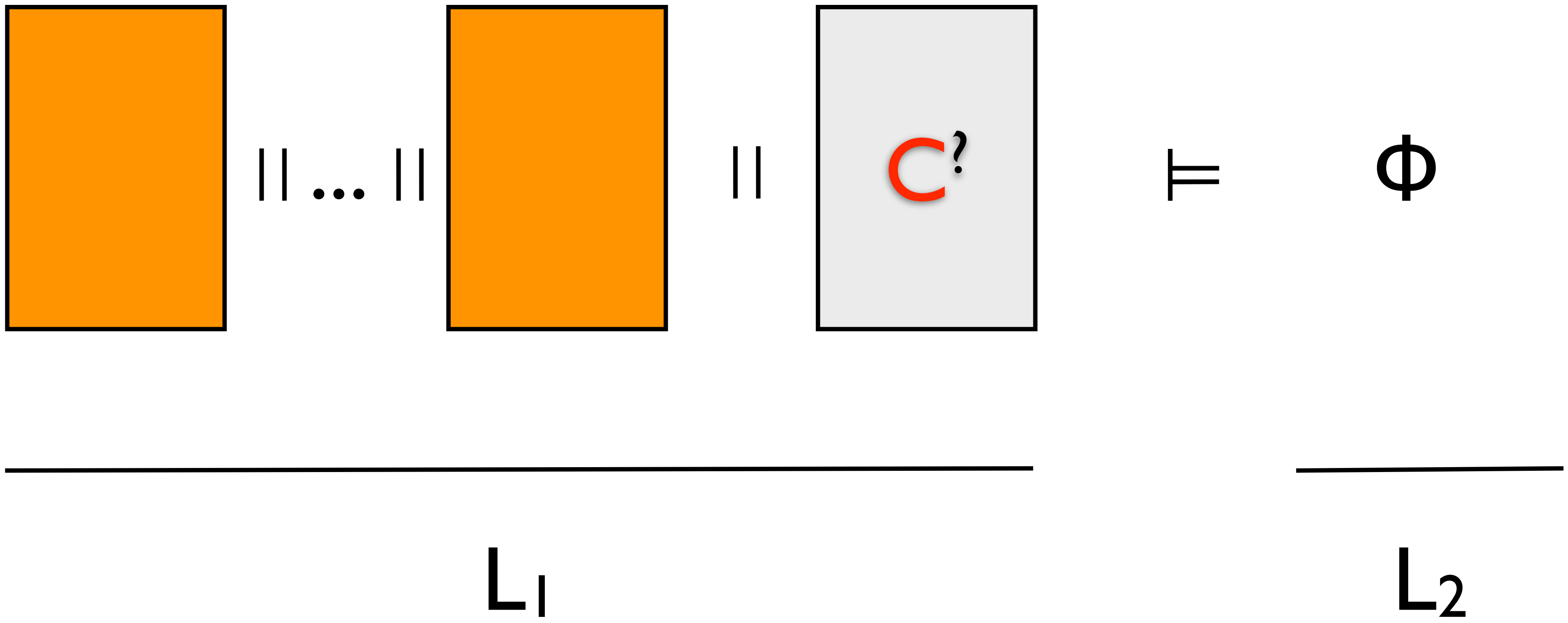
Verification



Question: L1 $\subseteq^?$ L2

Usually PSpace-hard

Synthesis



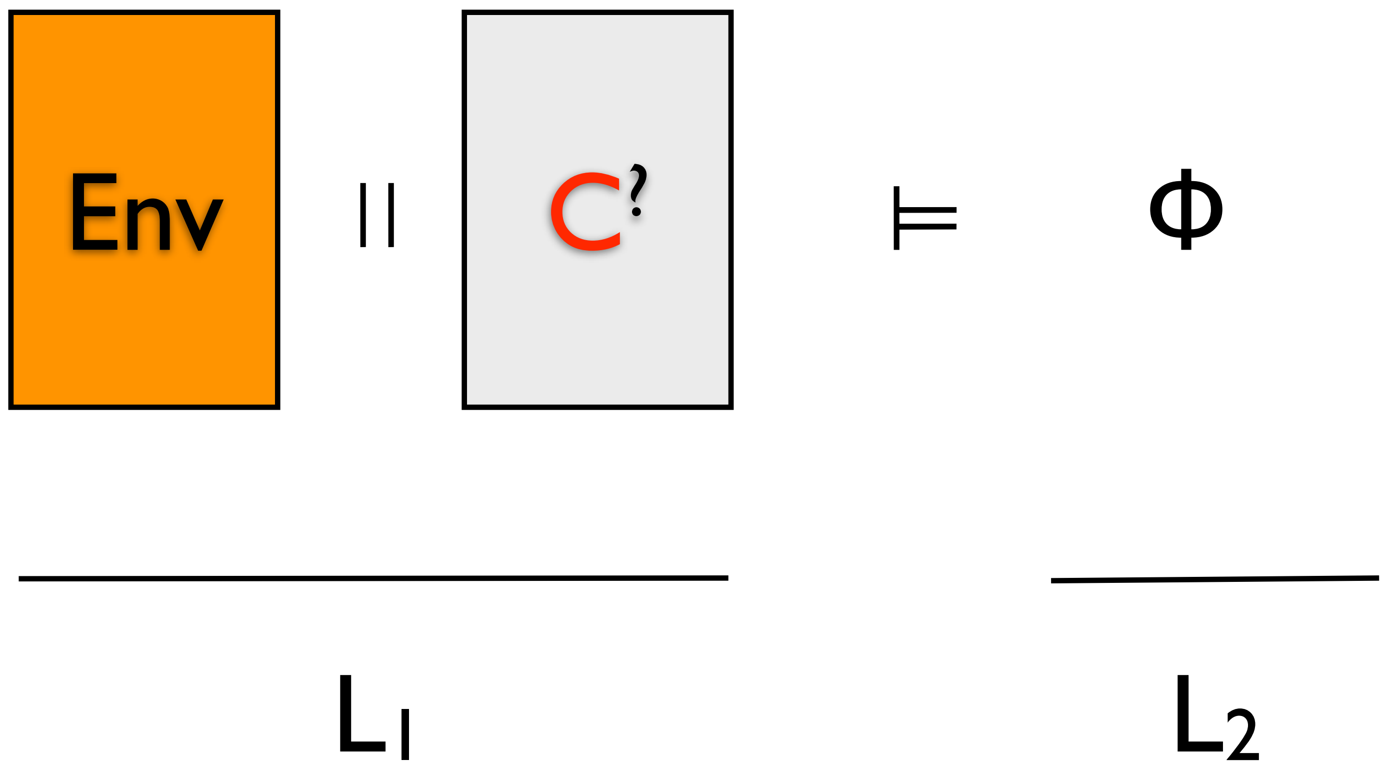
Question: Find C such that L1 ⊆ L2

Main research efforts in Computer Aided Verification and Synthesis

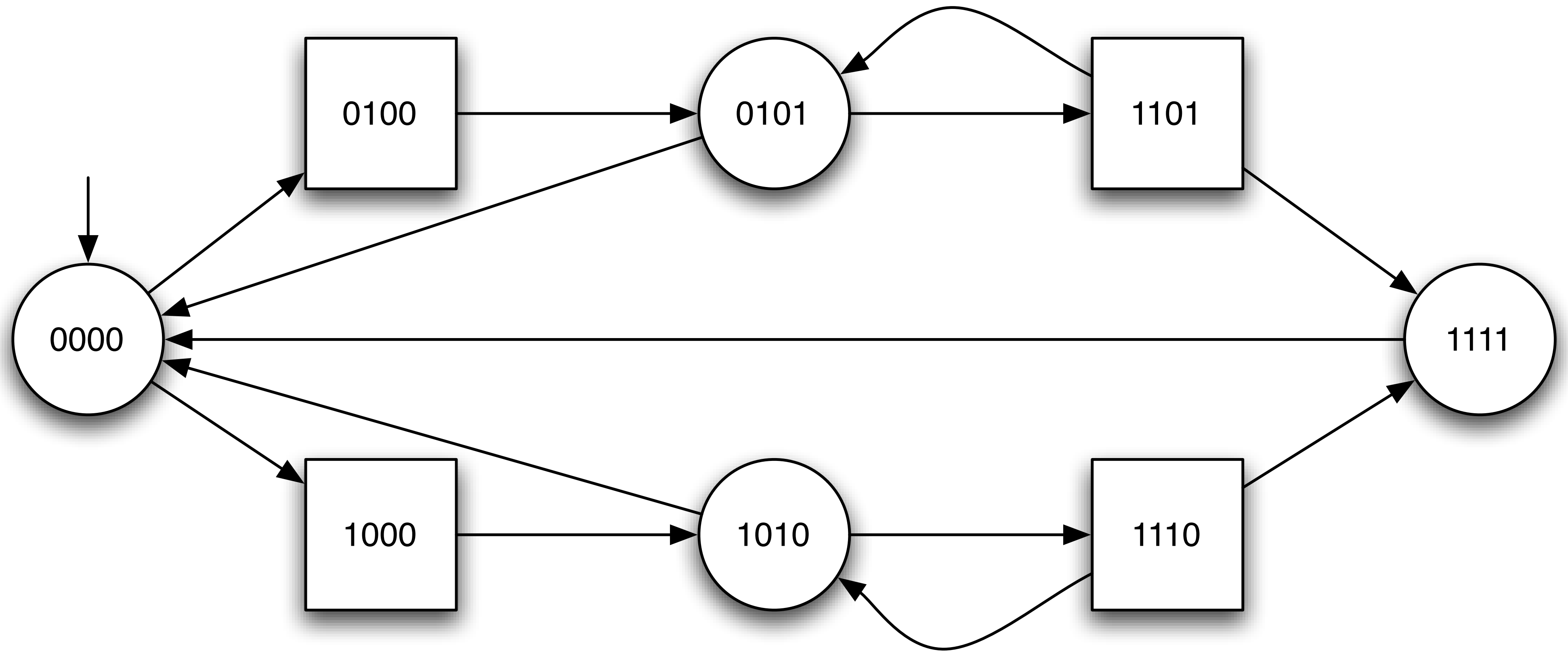
- Find good models for modeling reactive systems (automata and **extensions,** e.g. real-time);
- Study the **complexity** of verification and synthesis problems;
- Find algorithms to **verify** correctness of design models against properties;
- Find algorithms to **synthesize** components from specifications.

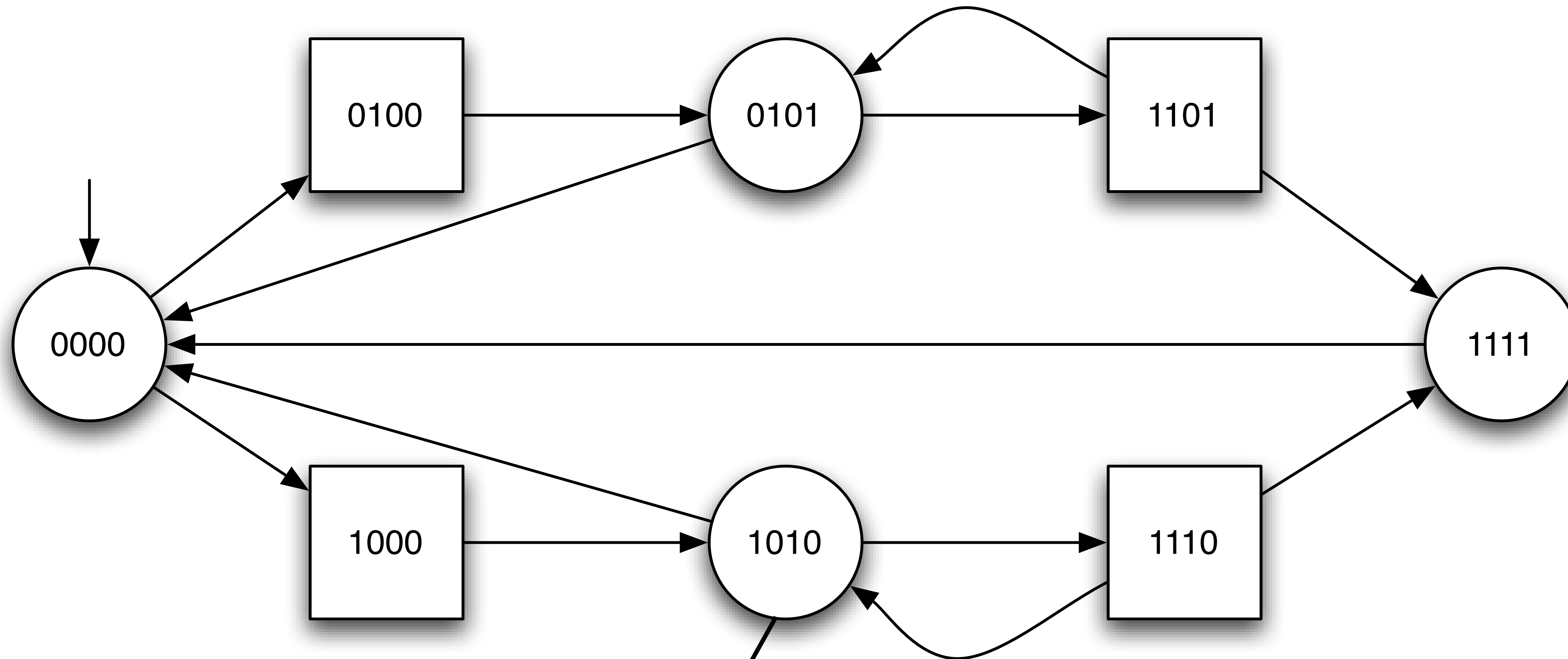
The synthesis problem reduced to a **game** problem

The synthesis problem

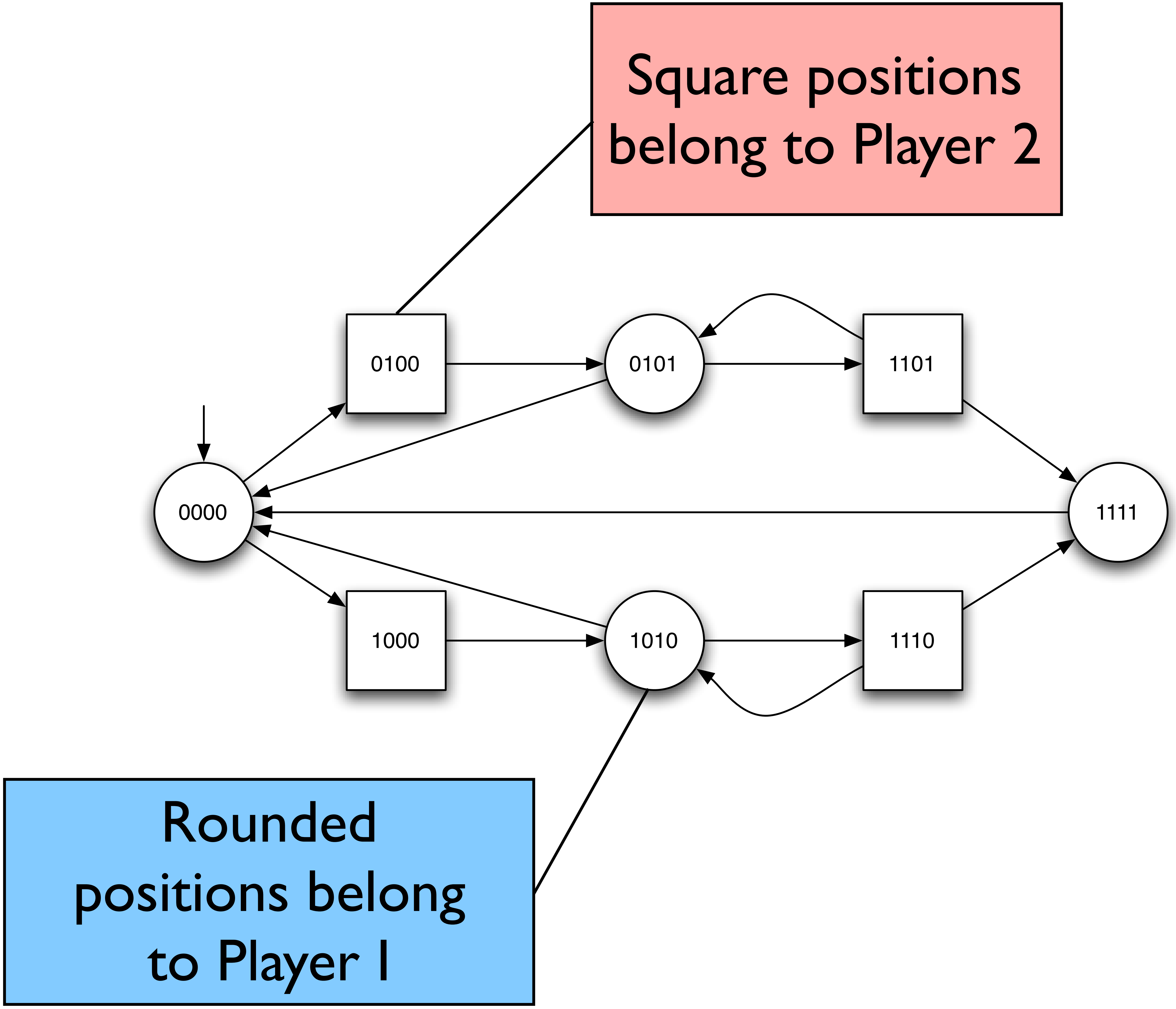


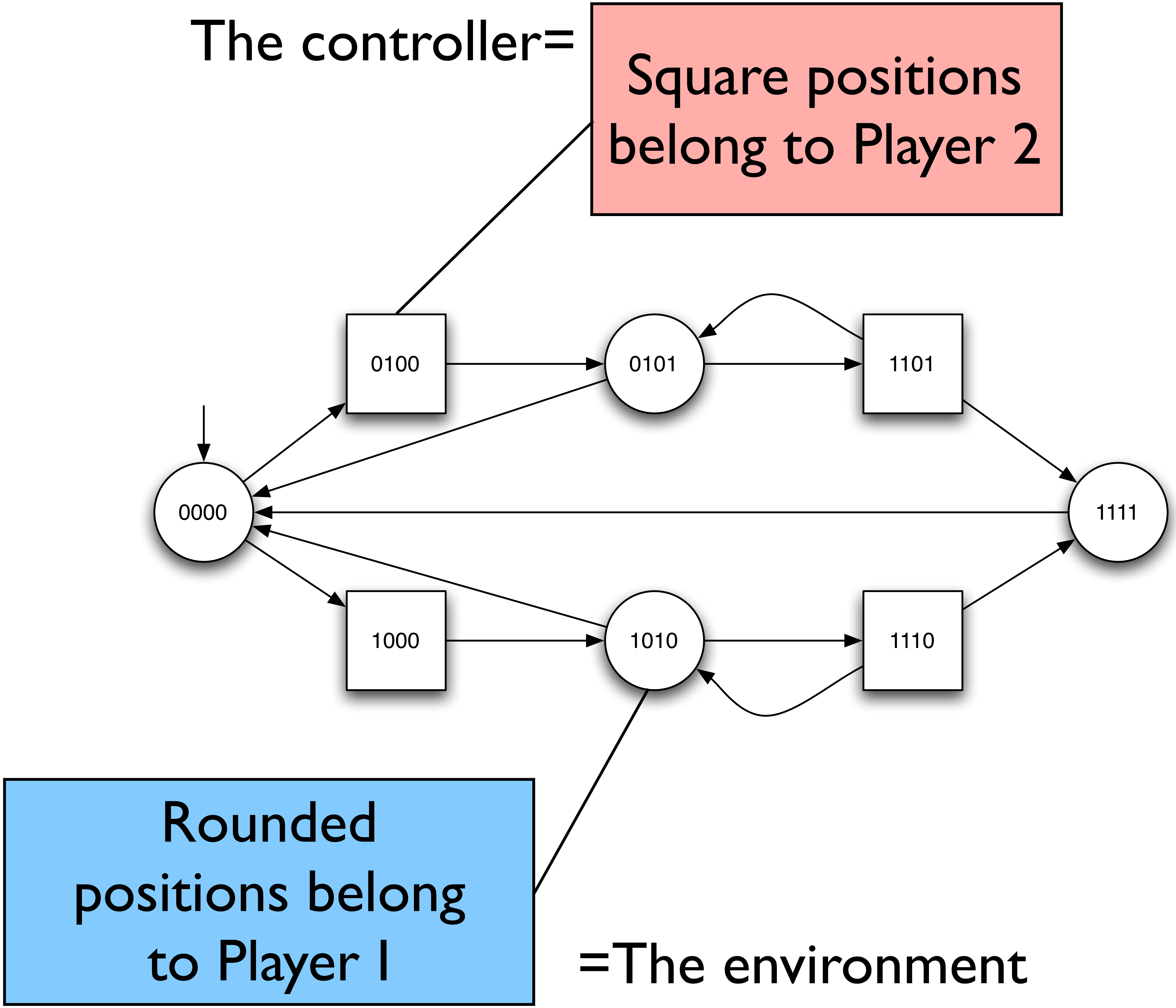
Question: Find C such that L1 ⊆ L2


0100
0101
1101
0000
1111
1000
1010
1110

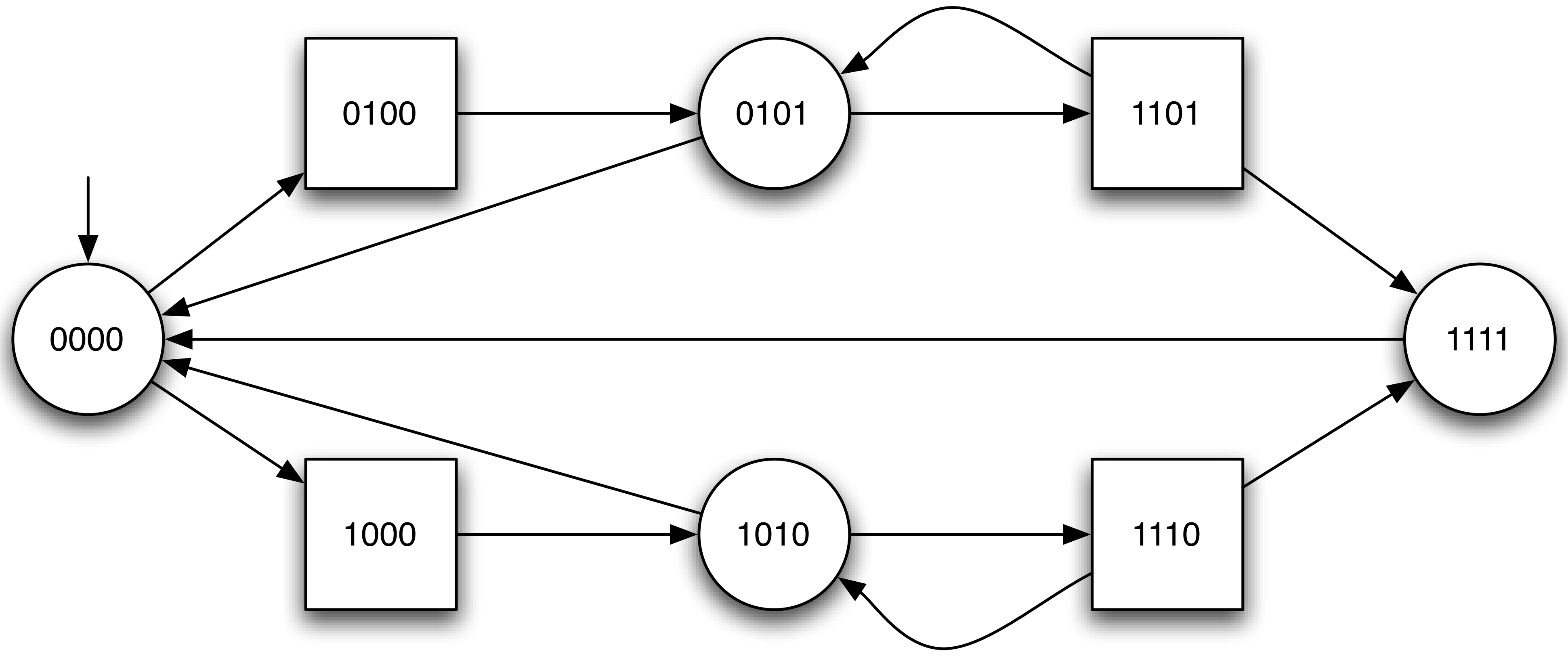


Rounded positions belong to Player 1


Square positions belong to Player 2
0100
0101
1101
0000
1111
1000
1010
1110
Rounded positions belong to Player 1

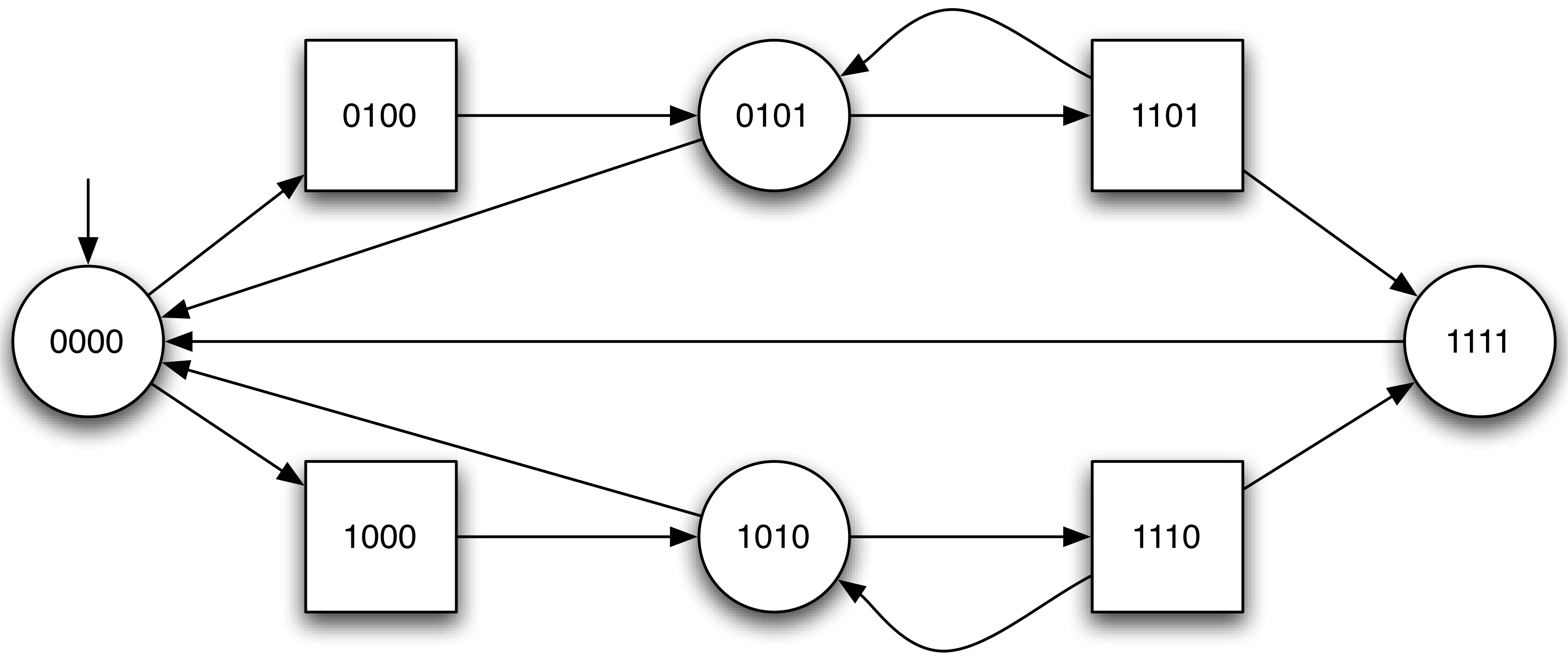

The controller=
Square positions belong to Player 2
0100
0101
1101
0000
1111
1000
1010
1110
Rounded positions belong to Player I
=The environment

Rounded positions belong to Player 1
Square positions belong to Player 2



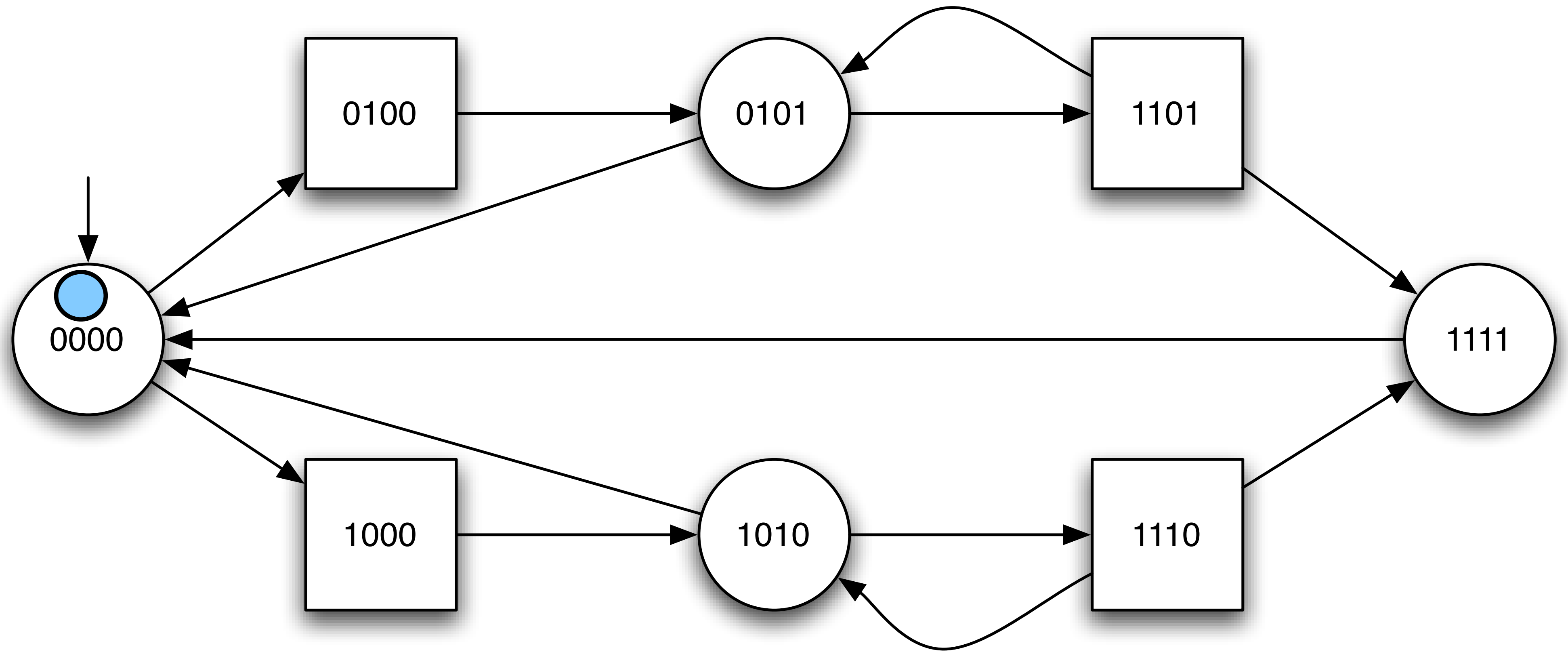
A game is played as follows: in each **round**, the game is in a **position**, if the game is in a rounded position, Player 1 resolves the **choice** for the next state, if the game is in a square position, Play 2 resolves the choice. The game is played for an **infinite number of rounds**.

Player 1 = Environment
Player 2 = Controller

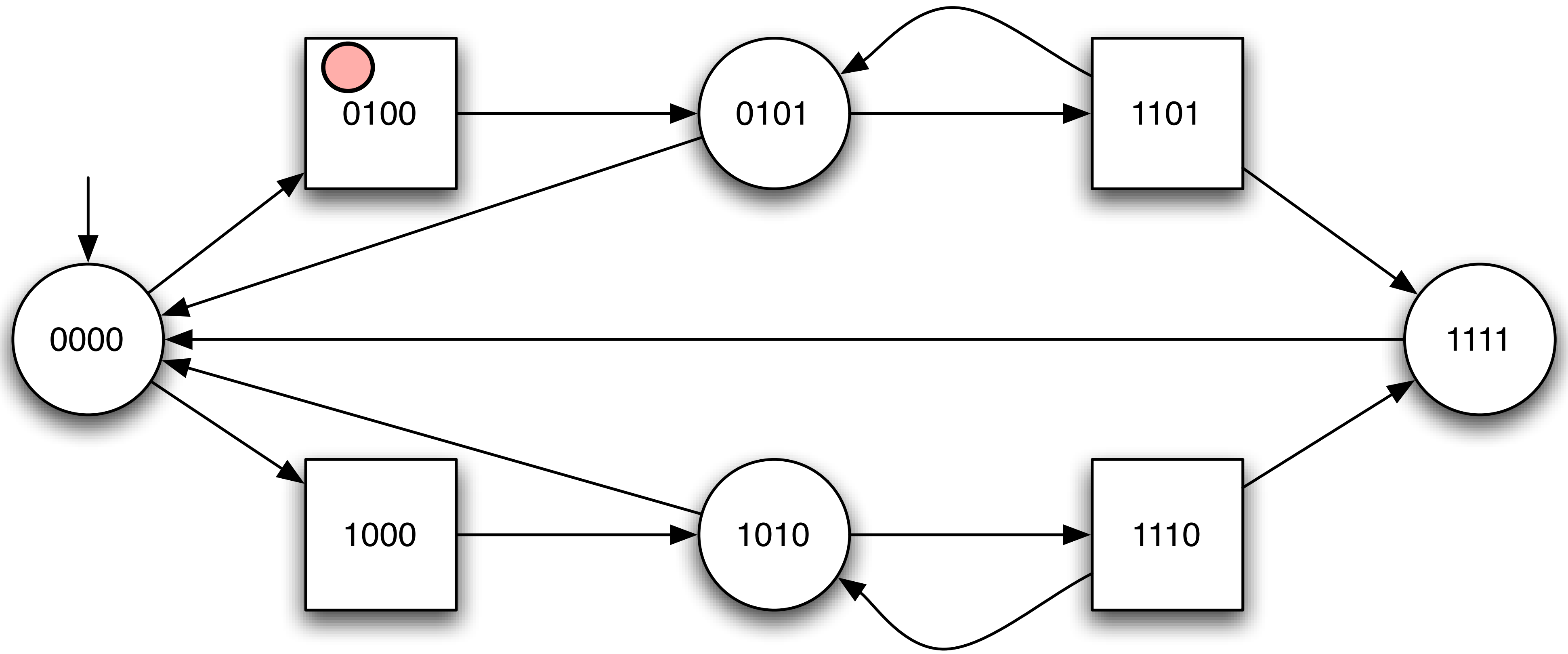


☛ The **choices** of the controller are to be interpreted as **decisions** that are to be taken to control the environment.

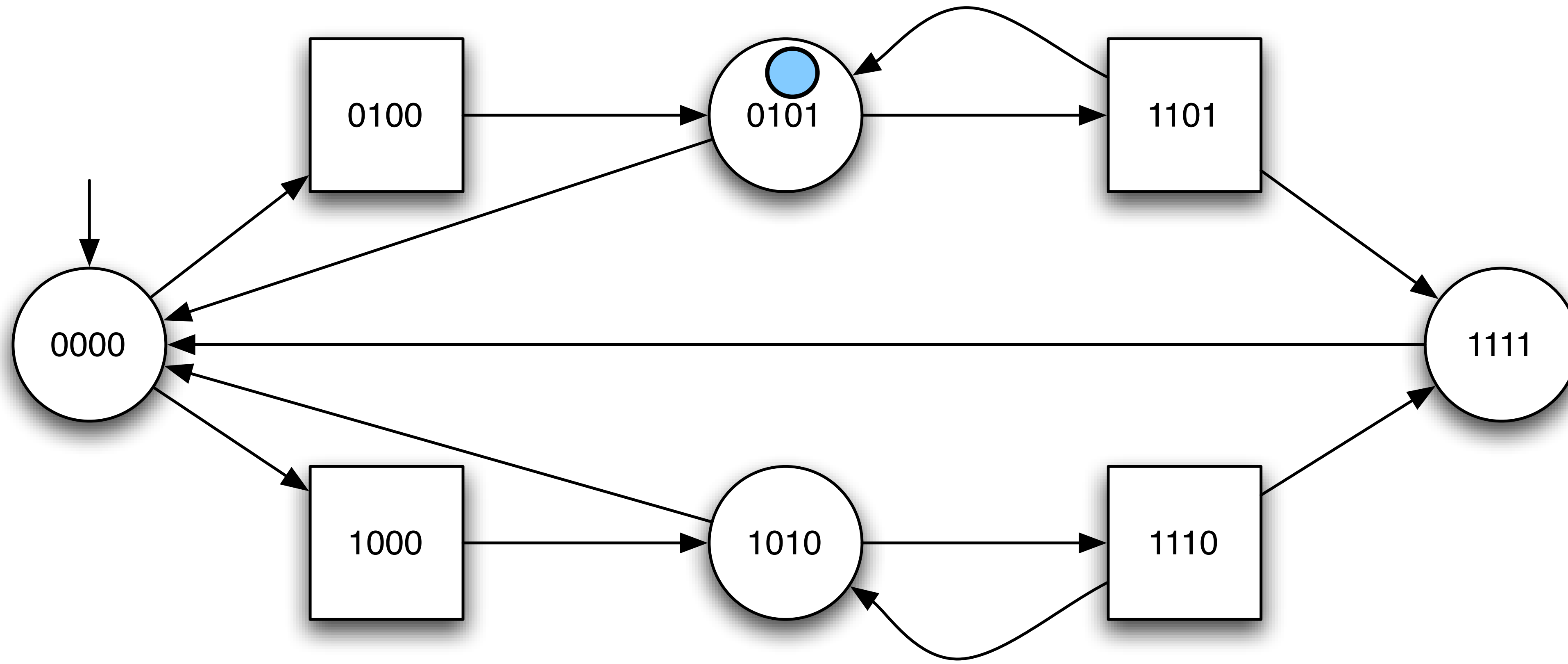
☛ The **choices** of the environment are beyond the control of the designer of the system and they must be interpreted as **adversarial**.



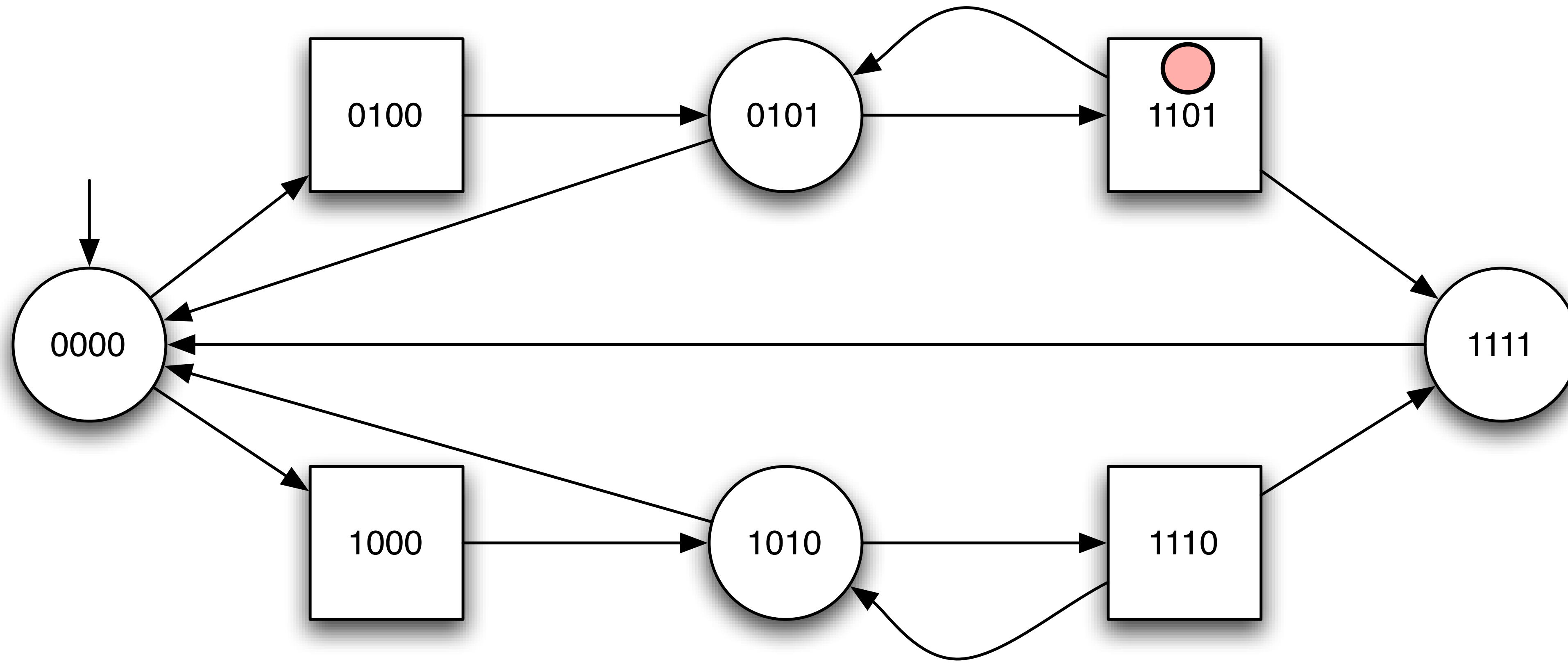
Play : 0000



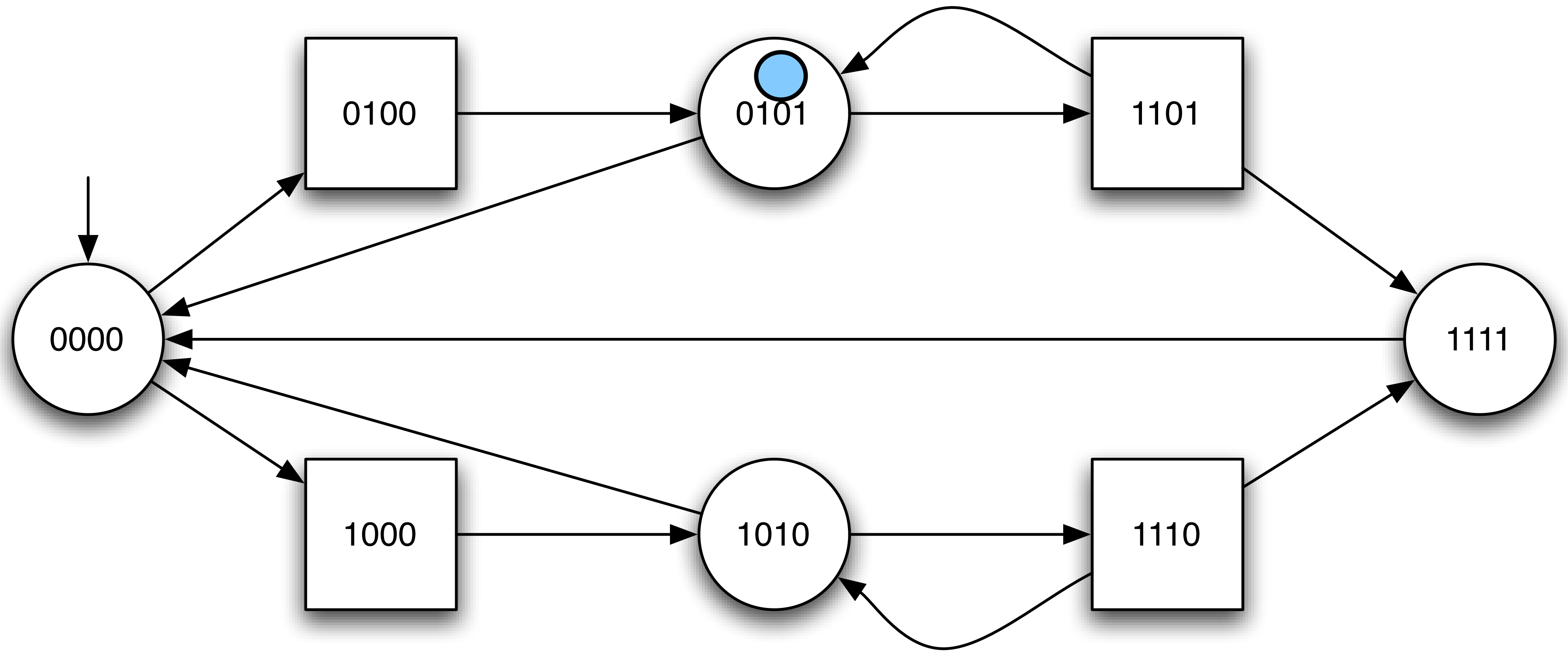
Play : 0000 0100



Play : 0000 0100 0101

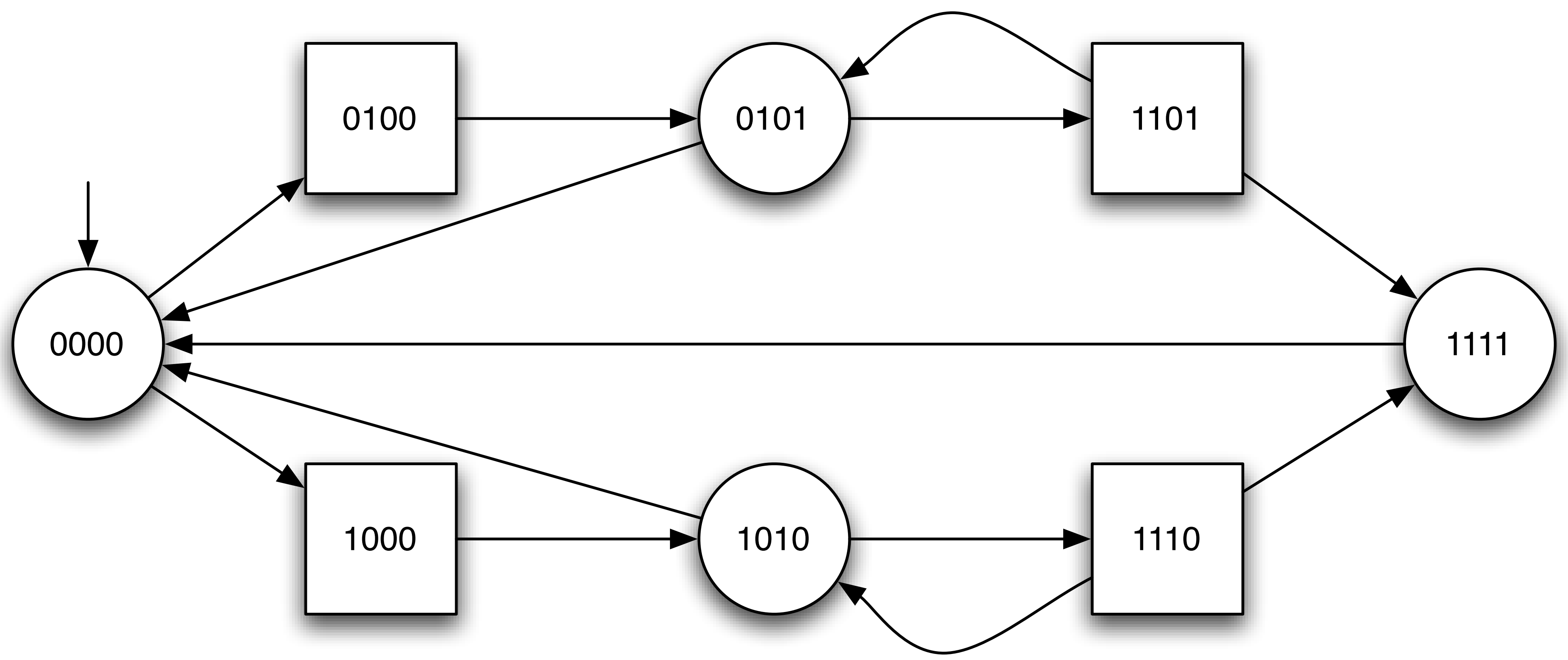


Play : 0000 0100 0101 1101



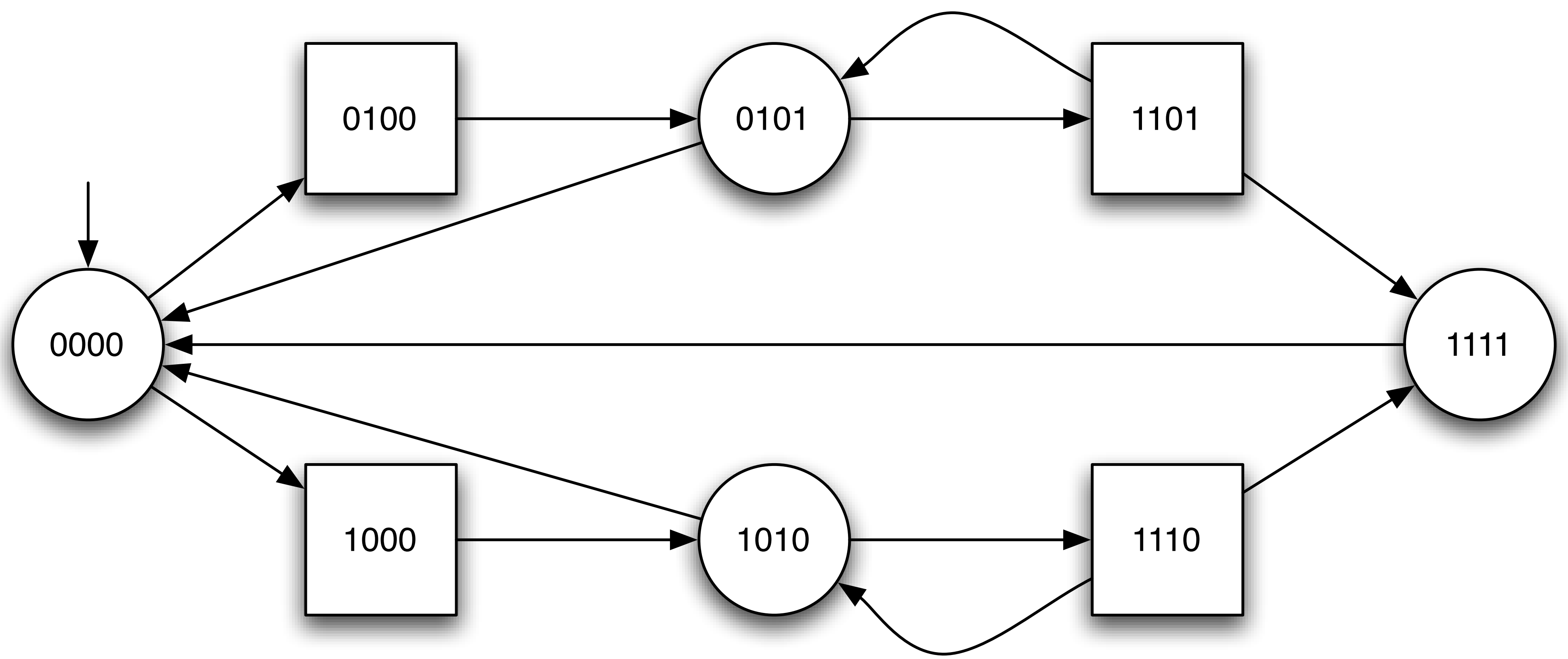
Play : 0000 0100 0101 1101 ...

Who is winning ?



Play : 0000 0100 0101 1101 ...

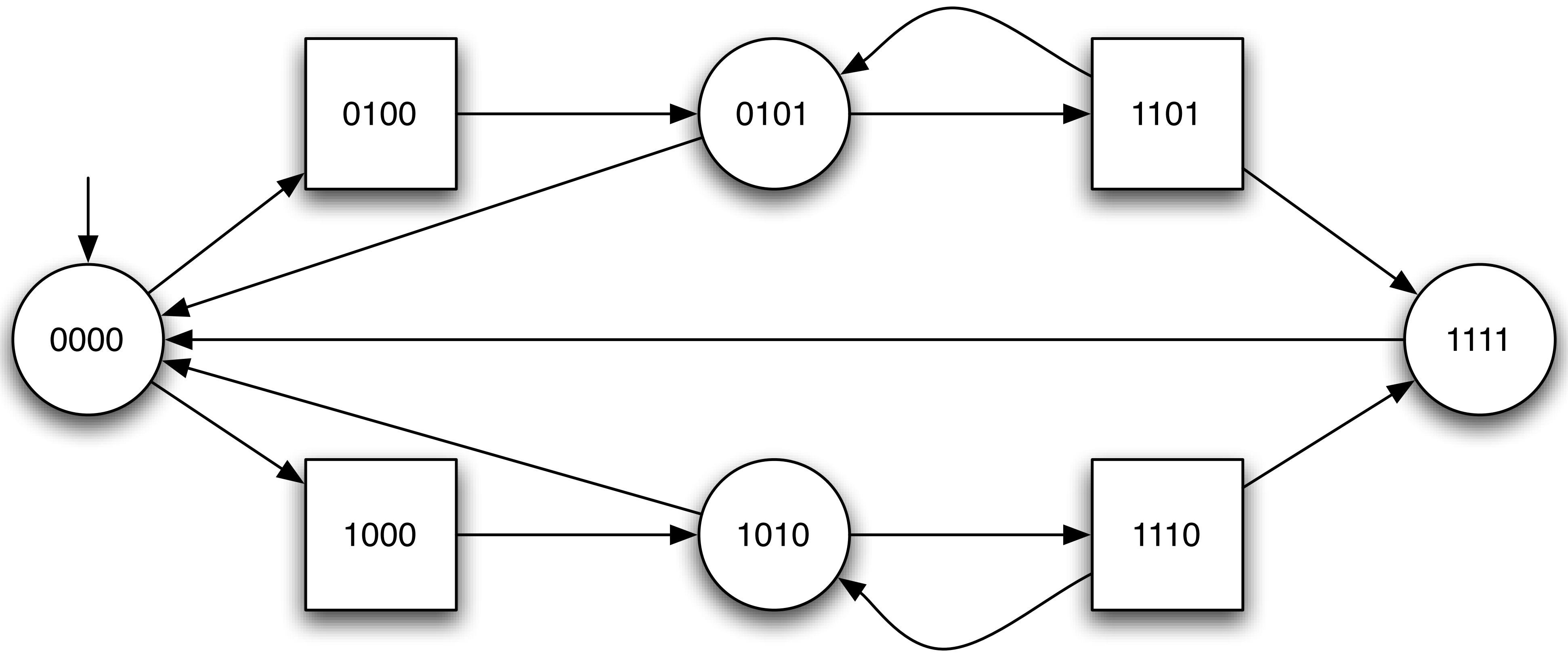
Who is winning ?



Play : 0000 0100 0101 1101 ...

=Trace, behavior of the system under design

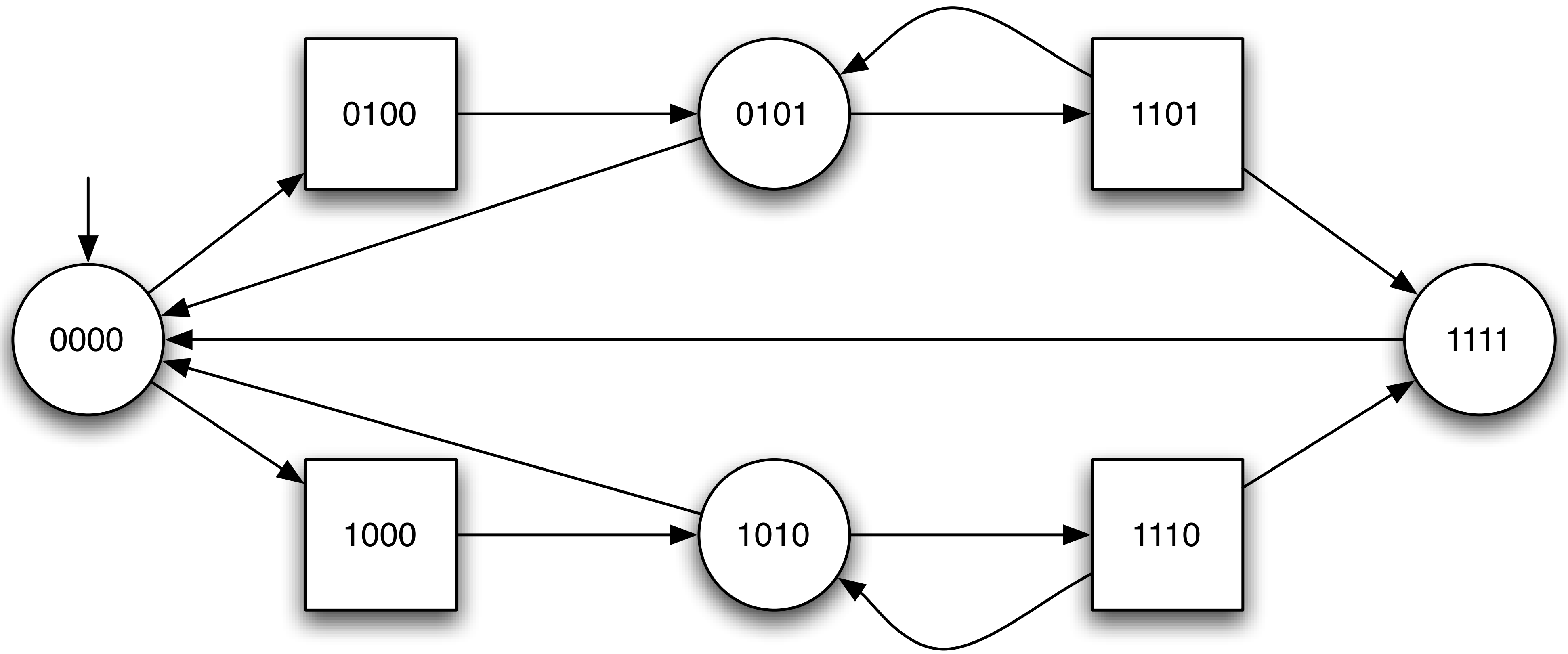
Who is winning ?



Play : 0000 0100 0101 1101 ...

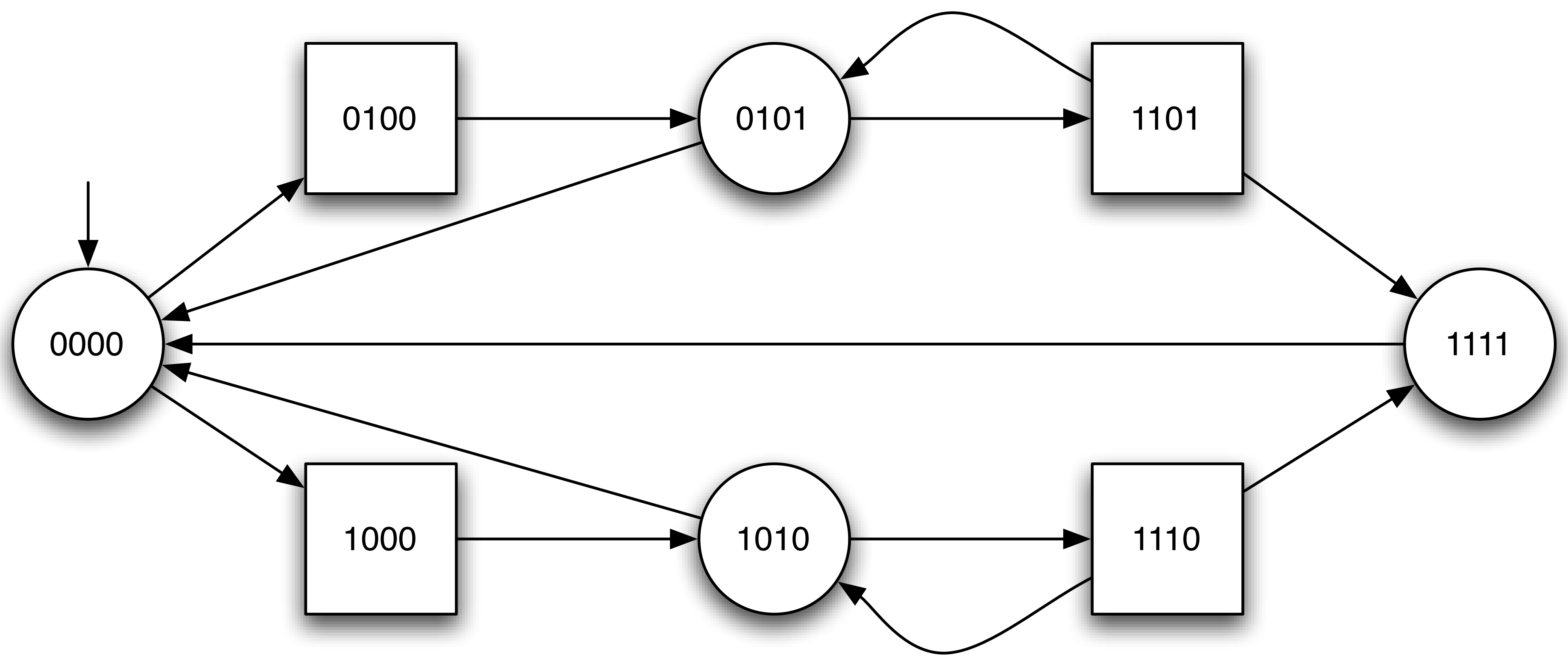
Is this a **good** or a **bad** play for **Player 2** ?

Who is winning ?



A **winning condition** (for Player 2)
is a set of plays
$$W \subseteq (Q_1 \cup Q_2)^\omega$$

Who is winning ?



A **winning condition** (for Player 2)
is a set of plays

A property !

Game

=

Two-player game structure

+

Winning condition for Player 2

Game

=

Two-player game structure

+

Winning condition for Player 2

The specification !

Strategies

Players are playing **according** to **strategies.**

A **Player *k*** (=1,2) **strategy** is a function that given the positions visited so far prescribes the next move to play.

A **Player *k*** (=1,2) **strategy** is winning for objective *W* if when player k plays according to the strategy the resulting play is within *W*, no matter what Player 3-k is playing.

Winning strategies

=

Controllers that enforce winning plays

Examples of open problems in the area

Non zero sum games played on graphs

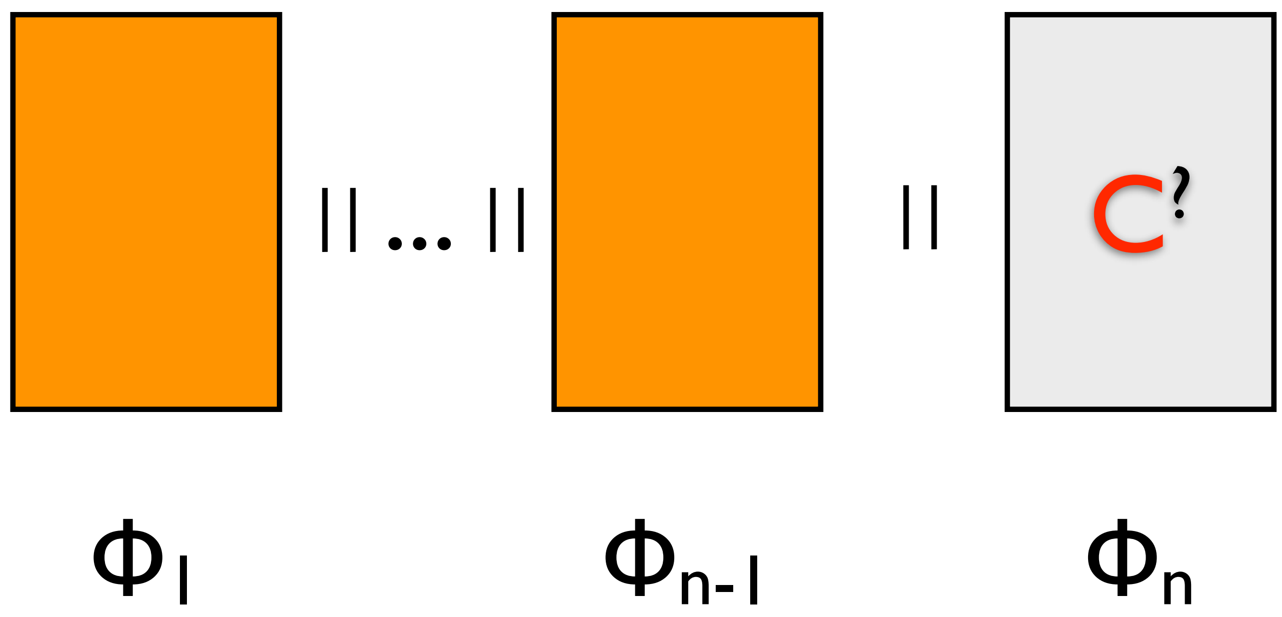
- In the example of game problems defined so far, we have given **fully antagonist** objectives to players:

  Player 2 is winning if the resulting play is in W,
  Player 1 is winning if the resulting play is not in W.

  This is a very conservative view (the environment is demoniac).

- ... very often we would like to synthesize systems where each component has its own objectives. Those objectives are **not** necessarily fully antagonist.

Non zero sum games played on graphs



Can we use concepts like **Nash equilibria**, **subgame perfect equilibria**, **secure equilibria** to synthesize complex reactive systems ?

Efficient synthesis for LTL objectives

- Two-player zero sum game played on a graph with winning conditions defined using a LTL formula are **2-ExpTime Complete**.
- A **theoretically optimal** procedure is known since 1989 but this procedure is not usable in practice. The doubly exponential almost always shows up.
- We still need to better understand the **structure** of this problem in order to study heuristics based on **new strong theoretical arguments**.

From Qualitative to Quantitative

- A large number of results that are known in the field of games played on graphs are for **qualitative** objectives (boolean objectives);
- We need to study variants of those problems where the objectives are **quantitative**. This new results will be the theoretical basis for **optimal controller synthesis**.

Why is our approach innovative ?

- The current state of the art in computer system design is **still very ad-hoc**;
- The theoretical basis for a **modern system theory** are still largely to be defined;
- **Game theoretic formalisms** are well suited to model systems build from several components as are modern computer systems.

Why is our project exciting ?

- Synthesis is a very **ambitious** goal: this would help designer to concentrate on important high level aspects of systems (their specification) and allow to avoid low level errors that are often very difficult to find.
- **Game theory** is a very **elegant** piece of mathematics. Games played on graphs are strongly related to **important problems** in logic and automata theory.

Research axes

- Axis 1. Adapted notions of games for synthesis of complex interactive computational systems.

  Non zero sum games, solution concepts, ...

- Axis 2. Games played on complex and infinite graphs.
  Timed systems, recursive graphs, pushdown automata, games with counters, ...

- Axis 3. Games with quantitative objectives.
  Stochastic games, games with costs, ...

- Axis 4. Game with incomplete information and over dynamic structures.
  Dynamic networks, observation based strategies, need for randomized strategies, ...

- Axis 5. Heuristics for efficient game solving.
  Better understand the structure of problems to fight prohibitive worst case complexities, tools implementation.

Possible cross-fertilization with other CRPs

- LINT: Game foundations of interactions.
- CFSC: computational complexity issues, compact representations.
- ... ?