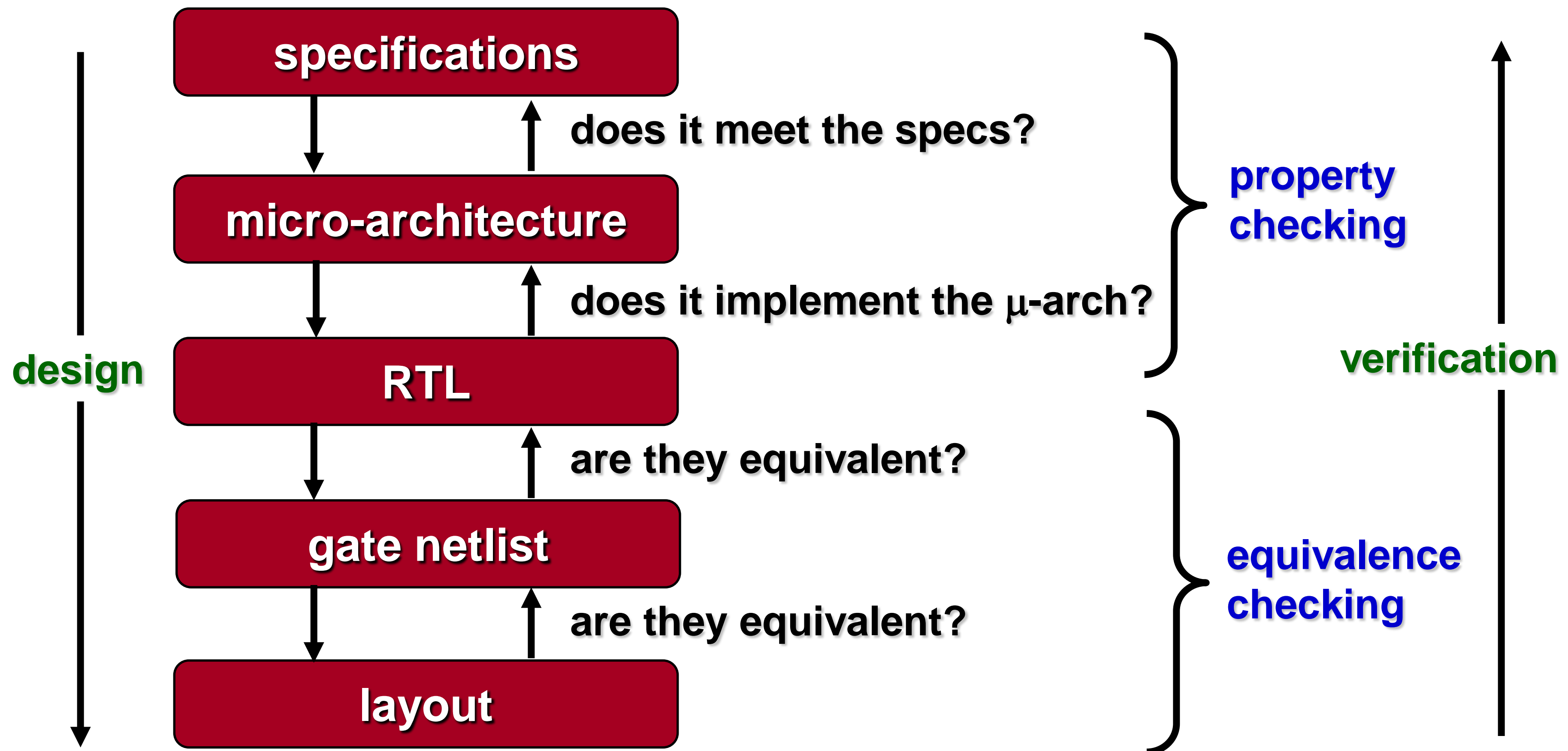
Design Verification – *An Overview*

Testing and Verification of Circuits (CS60089)

Dept. of CSE, IIT Kharagpur

Dr. Aritra Hazra
Assistant Professor,
Department of Computer Science & Engineering,
Indian Institute of Technology Kharagpur,
Paschim Medinipur, West Bengal, India – 721302.

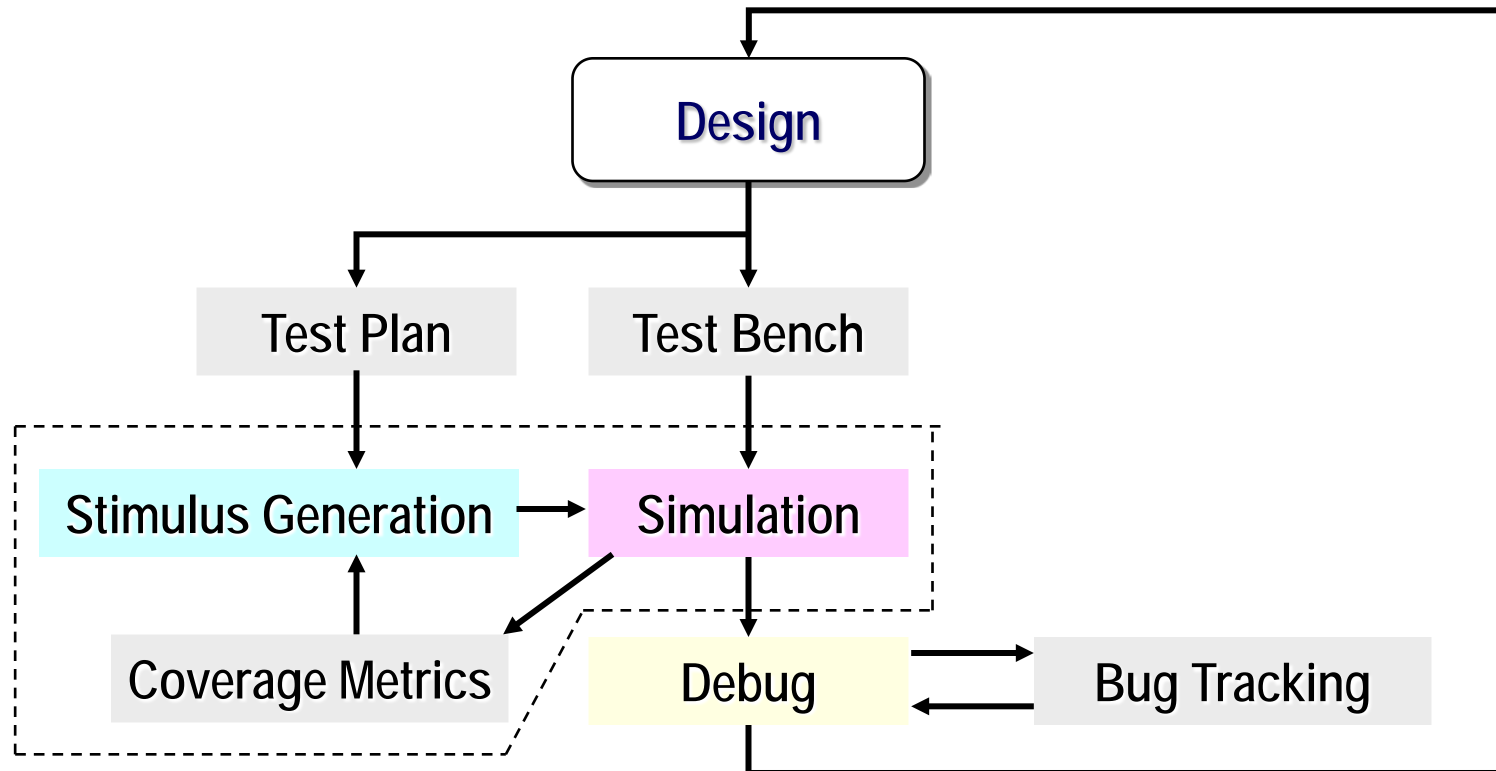
Design and Verification



Functional Verification Challenge

- **Is the implementation correct?**
  - How do we define *correct*?
    - Classical: Simulation result matches with golden output
    - Formal: Equivalence with respect to a golden model
    - Property verification: Correctness properties (assertions) expressed in a formal language
      - Formal: Model checking
      - Semi-formal: Assertion-based verification
  - Trade-off between computational complexity and exhaustiveness

Simulation



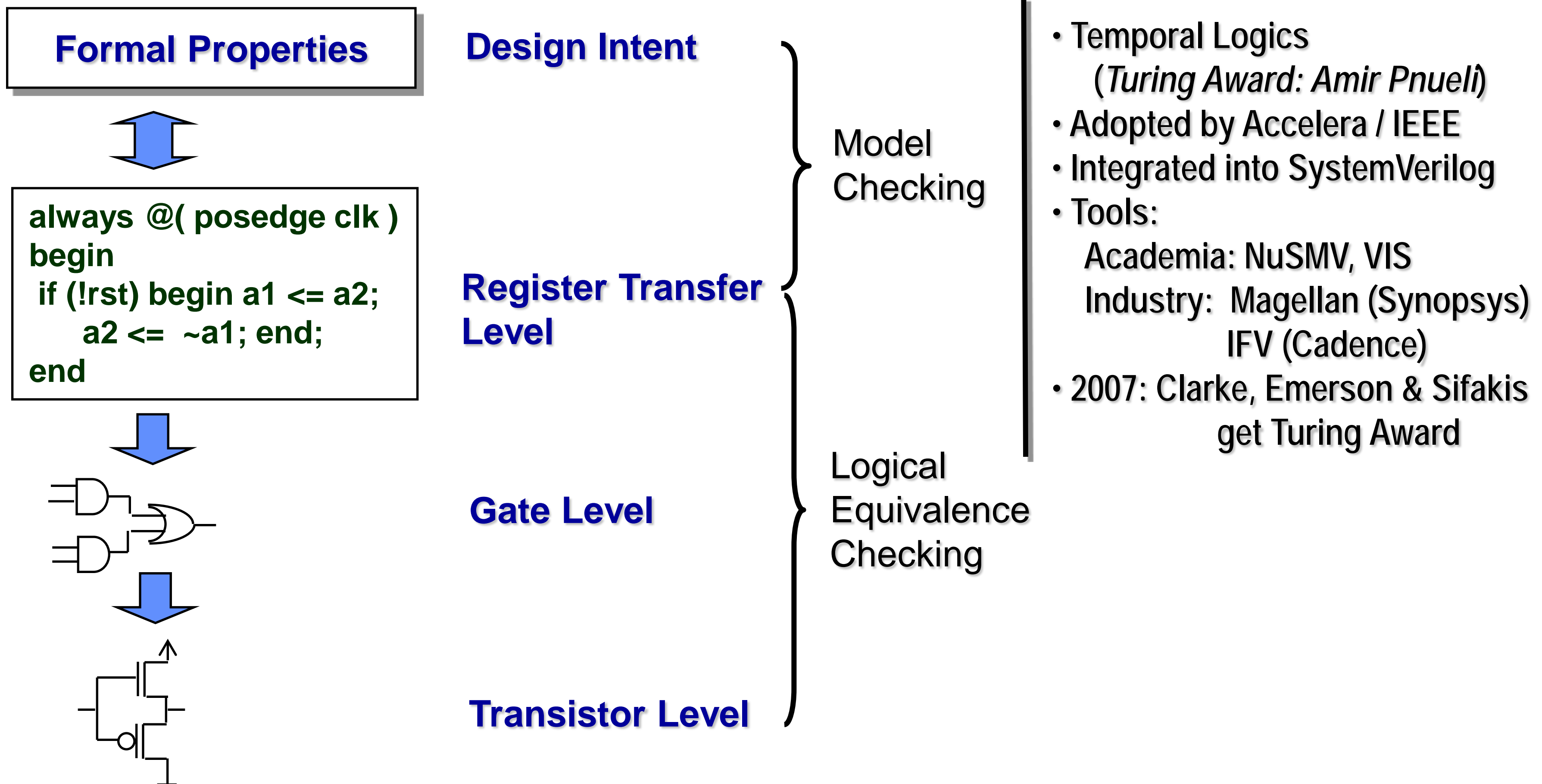
Advances:

- Test bench languages are richer (such as SystemVerilog)
- Coverage monitors and assertions
- Layered test benches and Transaction Level Modelling

Advent of Formal Methods in EDA

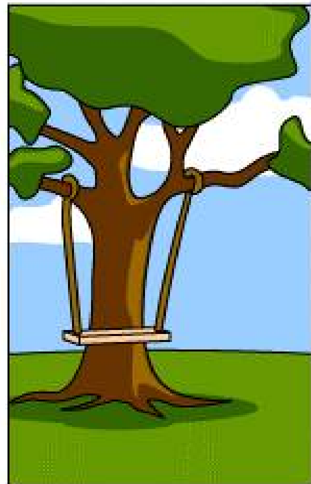
Goal: *Exhaustive verification of the design intent within feasible time limits*

Philosophy: *Extraction of formal models of the design intent and the implementation and comparing them using mathematical / logical methods*



Why do we need formal specifications?

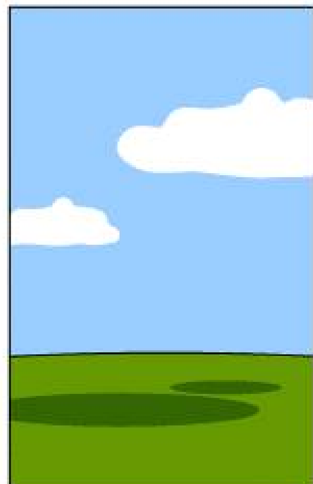
How the customer explained it

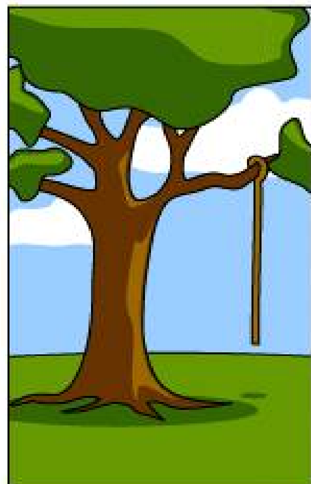

How the Project Leader understood it

How the Analyst designed it

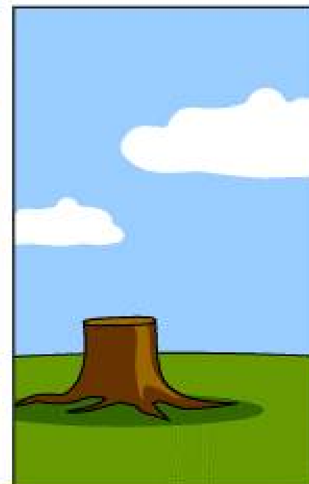
How the Programmer wrote it

How the Business Consultant described it


How the project was documented


What operations installed

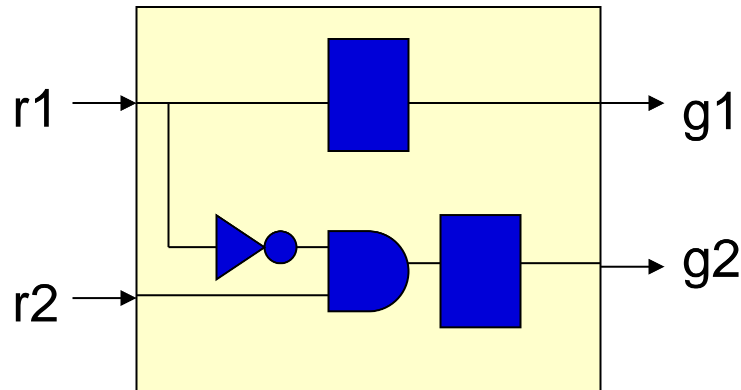
How the customer was billed


How it was supported

What the customer really needed

Source: http://lore.ua.ac.be/Teaching/SE3BAC/SoftwareSpecCartoon2.jpg

Toy Example: Priority Arbiter



- Either g1 or g2 is always false (mutual exclusion)

$$\text{always}[\neg g1 \vee \neg g2]$$

- Whenever r1 is asserted, g1 is given in the next cycle

$$\text{always}[\ r1 \Rightarrow \text{next } g1\ ]$$

- When r2 is the sole request, g2 comes in the next cycle

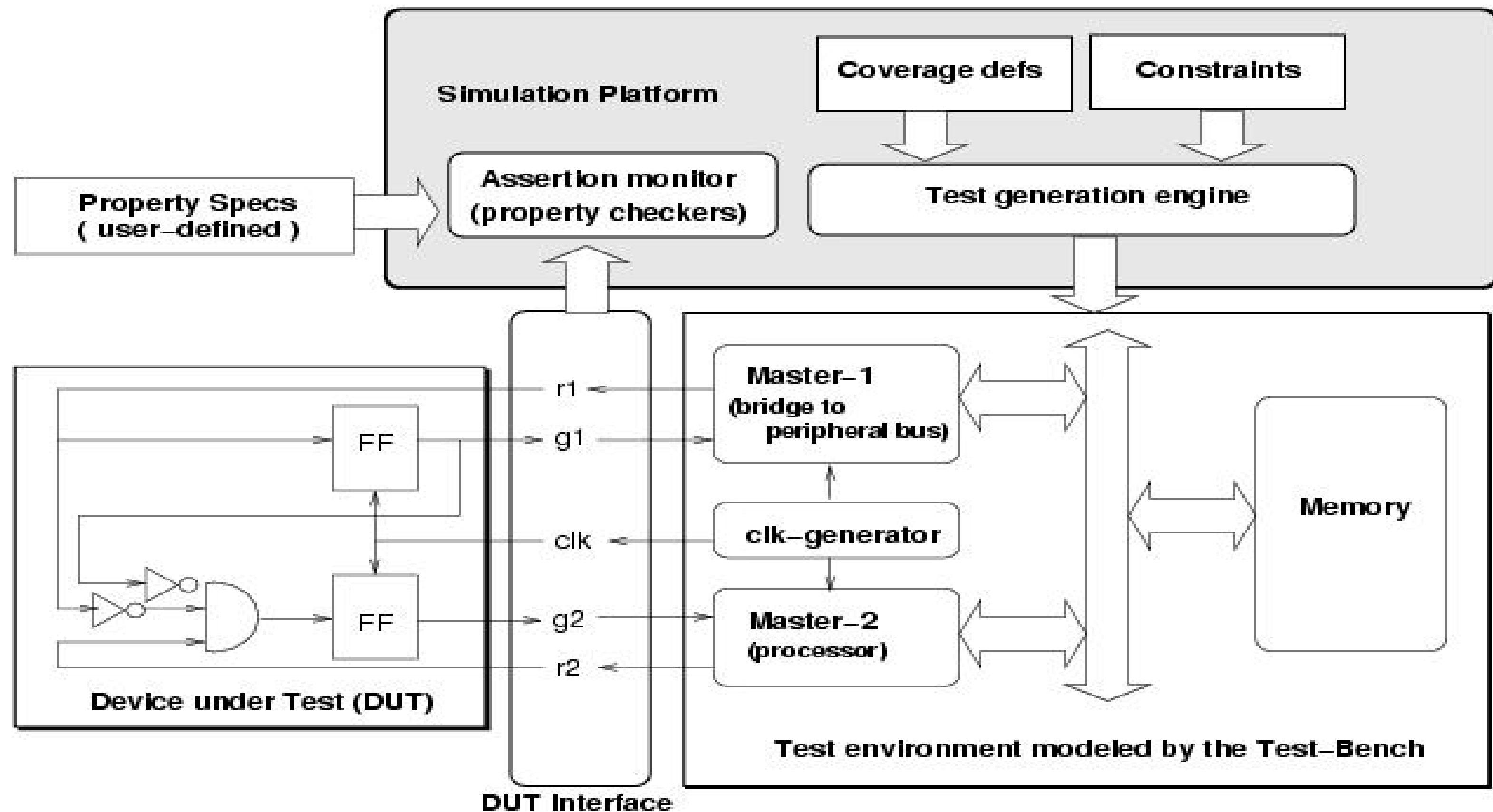
$$\text{always}[\ (\neg r1 \wedge r2) \Rightarrow \text{next } g2\ ]$$

- When none are requesting, the arbiter parks the grant on g2

$$\text{always}[\ (\neg r1 \wedge \neg r2) \Rightarrow \text{next } g2\ ]$$

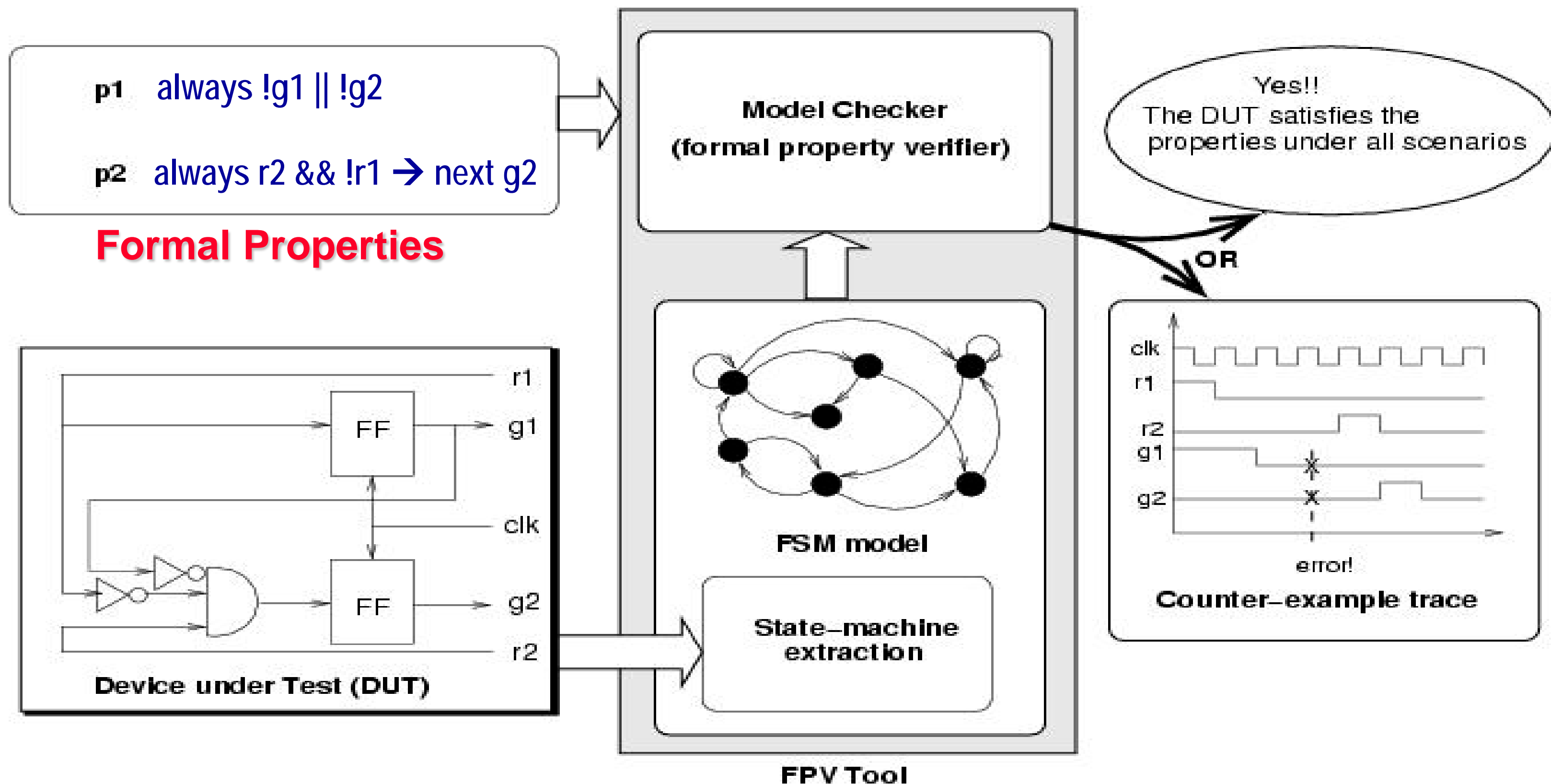
Violation!!

Dynamic Property Verification (DPV)



[Source: *A Roadmap for Formal Property Verification, Springer, 2006*]

Formal Property Verification (FPV)

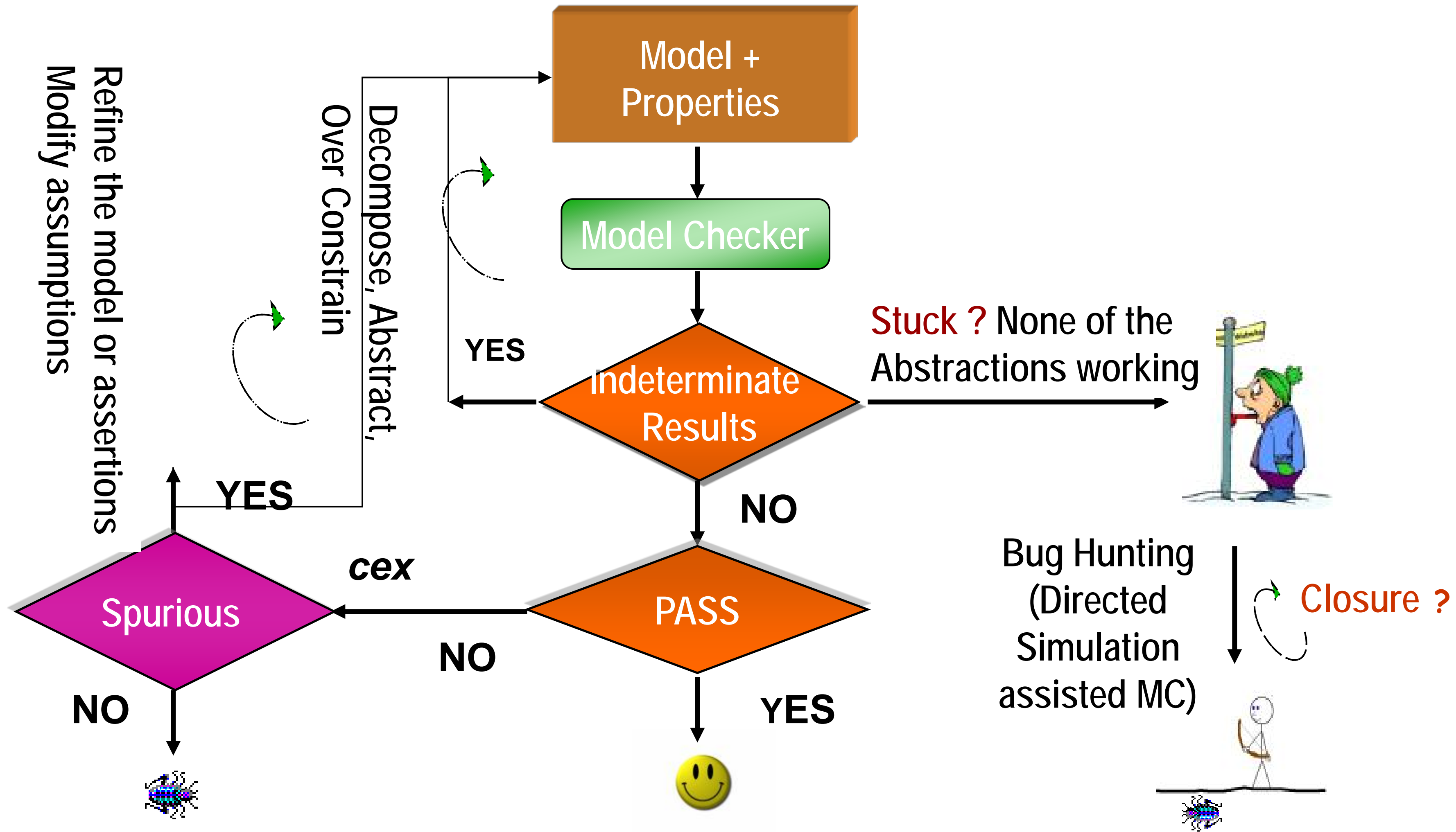


Temporal Logics (Timed / Untimed, Linear Time / Branching Time): *LTL, CTL*

Early Languages: *Forspec (Intel), Sugar (IBM), Open Vera Assertions (Synopsys)*

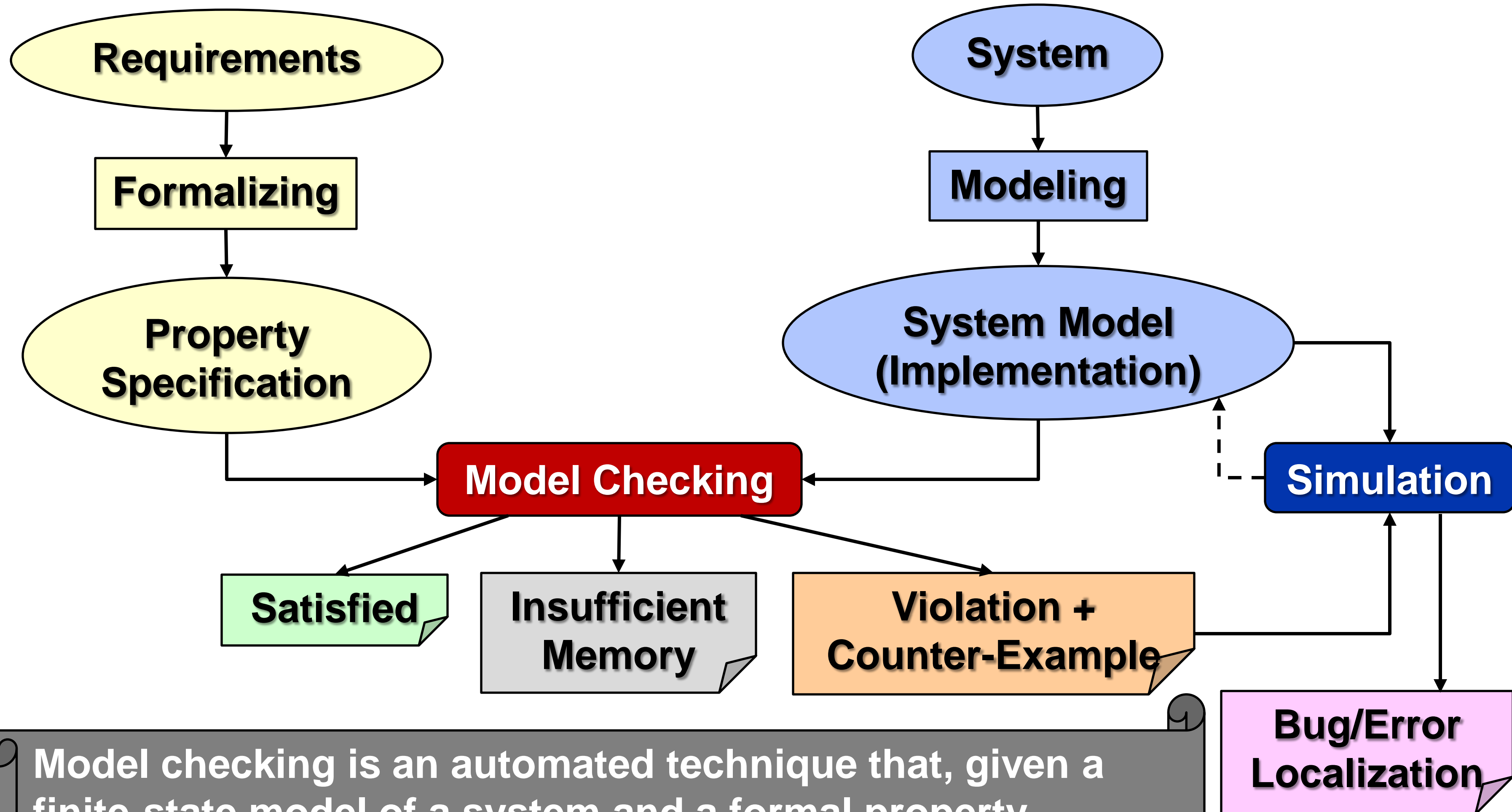
Current IEEE Standards: *SystemVerilog Assertions (SVA),*
Property Specification Language (PSL)

Assertion Based Verification Flow



[Source: Raj Mitra, TI]

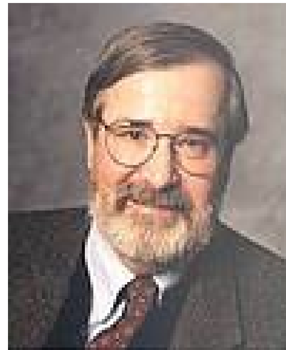
Model Checking Overview



Model checking is an automated technique that, given a finite-state model of a system and a formal property, systematically checks whether this property holds for (a given state in) that model.

Recognitions and Awards

- Paris Kanellakis Theory and Practice Award 1998



Randal Bryant | **Edmund Clarke** | **E. Allen Emerson** | **Ken McMillan**

For their invention of "symbolic model checking", a method of formally checking system designs, which is widely used in the computer hardware industry and starts to show significant promise also in software verification and other areas.

Recognitions and Awards (contd…)

- Gödel Prize 2000

Moshe Vardi

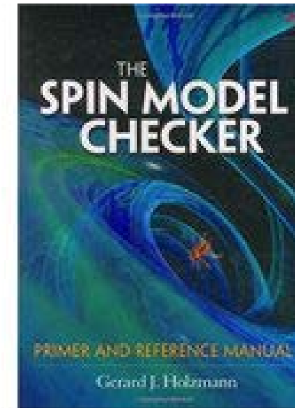
Pierre Wolper

"For work on model checking with finite automata"

Recognitions and Awards (contd…)

- ACM System Software Award 2001

Gerard J. Holzmann



SPIN Book

SPIN is a popular open-source software tool, used by thousands of people worldwide, that can be used for the formal verication of distributed software systems.

Recognitions and Awards (contd…)

- ACM Turing Award 2007

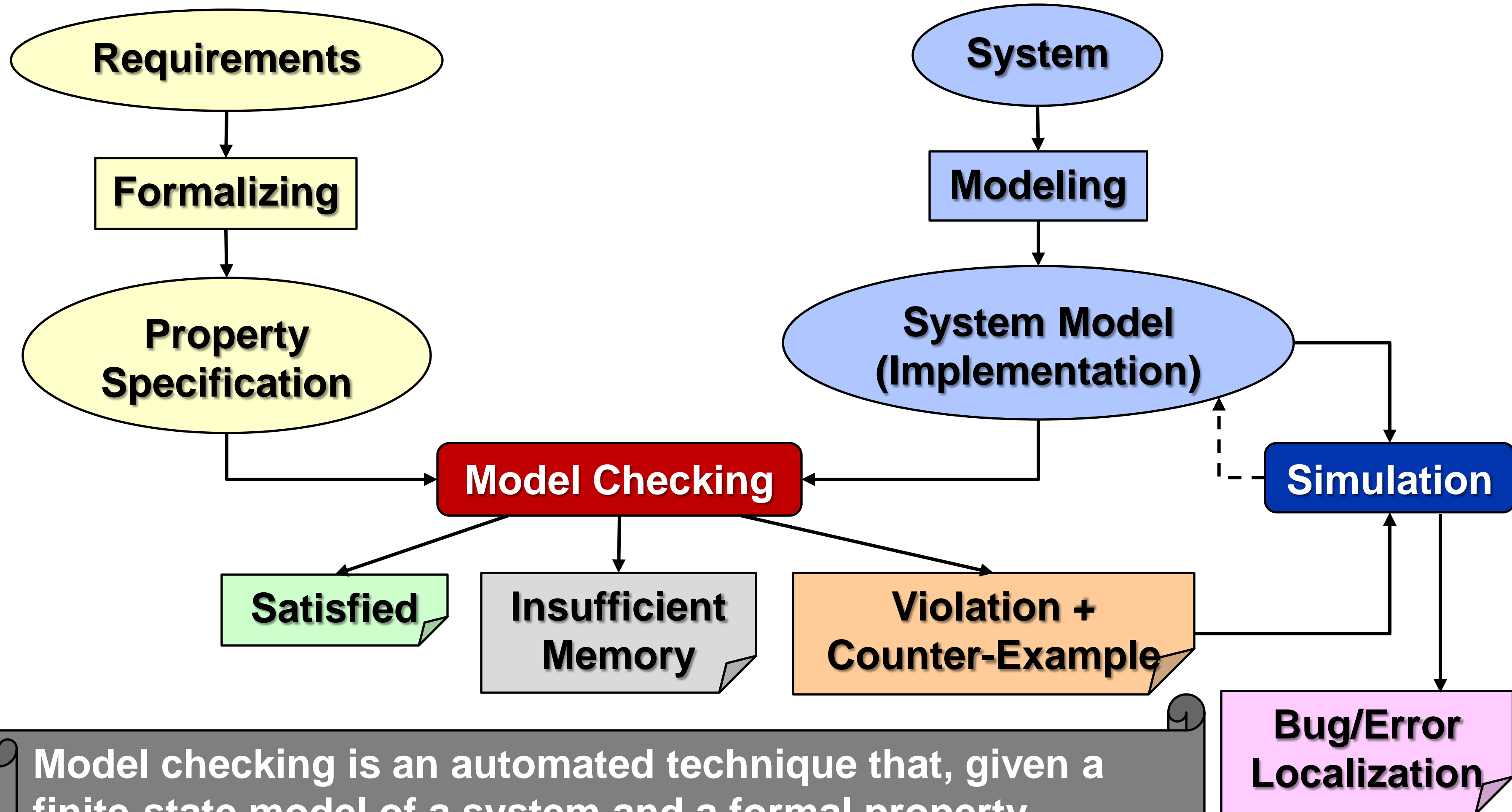
Edmund Clarke

E. Allen Emerson

Joseph Sifakis

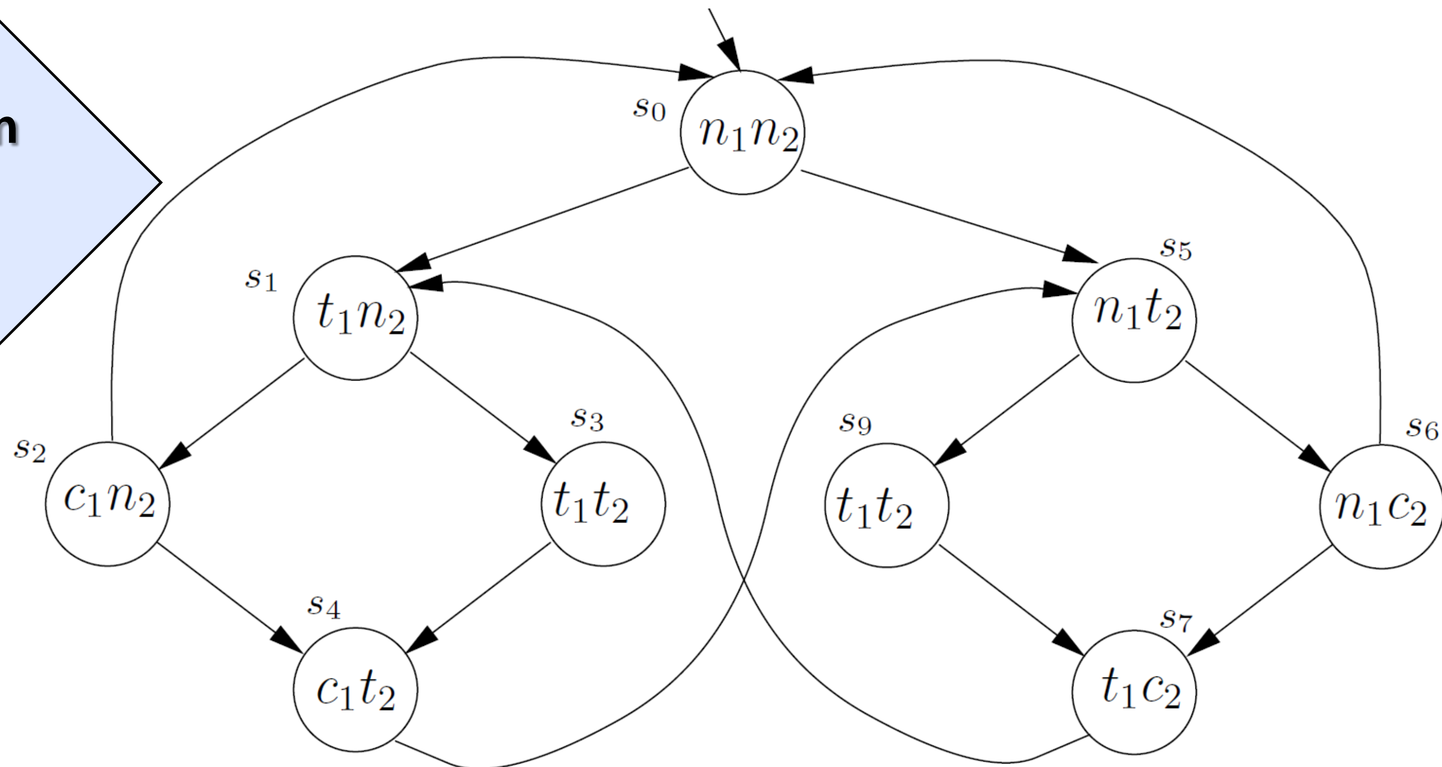
"For their role in developing Model-Checking into a highly effective verification technology, widely adopted in the hardware and software industries."

Model Checking Overview



Model checking is an automated technique that, given a finite-state model of a system and a formal property, systematically checks whether this property holds for (a given state in) that model.

What are Models?



- **Hardware Circuits as Transition Systems?**
  - States labelled with basic propositions
  - Transition relation between states
  - Action-labelled transitions to facilitate composition

What are Properties?

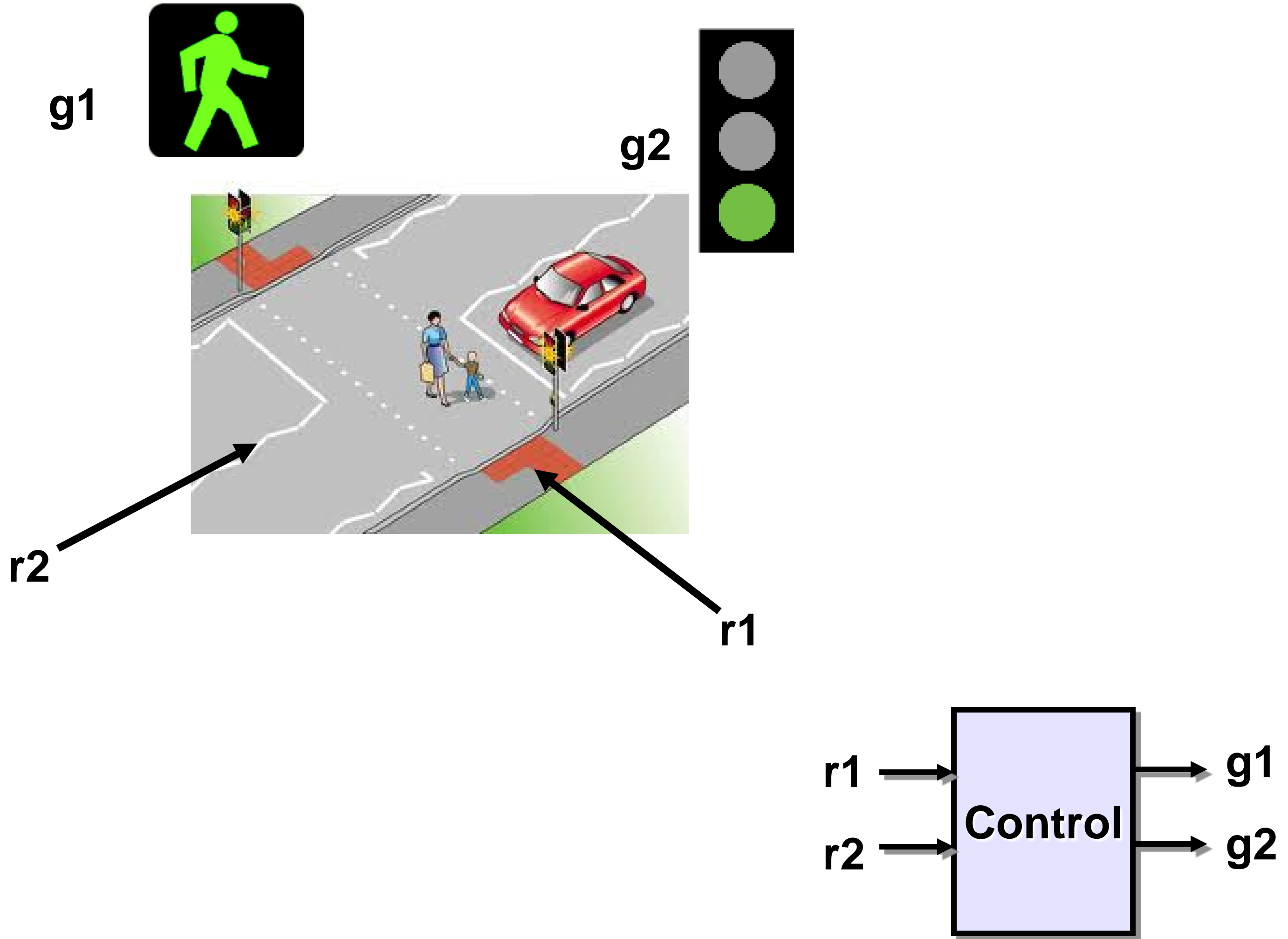
❑ Example Properties

- Can the system reach a deadlock situation?
- Can two processes ever be simultaneously in a critical section?
- On termination, does a program provide the correct output?

❑ Temporal Logic

- Propositional logic
- Temporal operators such as next, future, always, until
- Interpreted over state sequences (linear)
- Or over infinite trees of states (branching)

Example: *Simple Pedestrian Crossing Control*



Example: *A Simple Traffic Control*

Properties:

1. Request line r1 has higher priority than request line r2. Whenever r1 goes high, g1 must be asserted for the next two cycles

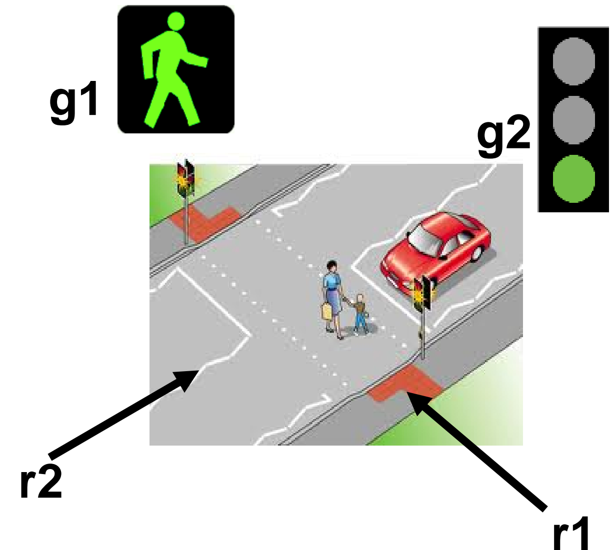
   $$\textbf{always}\ [\ r1 \Rightarrow \textbf{next}\ g1 \wedge \textbf{next next}\ g1\ ]$$

2. When none of the request lines are high, the control parks the grant on g2 in the next cycle

   $$\textbf{always}\ [\ \neg r1 \wedge \neg r2 \Rightarrow \textbf{next}\ g2\ ]$$

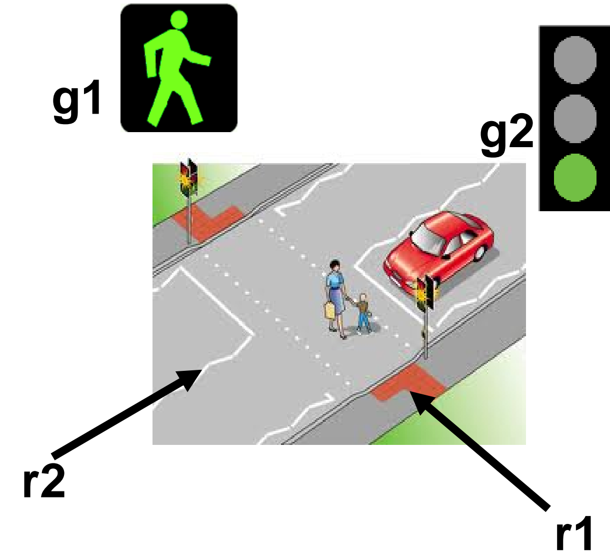
3. The grant lines g1 and g2 are mutually exclusive

   $$\textbf{always}\ [\ \neg g1 \vee \neg g2\ ]$$



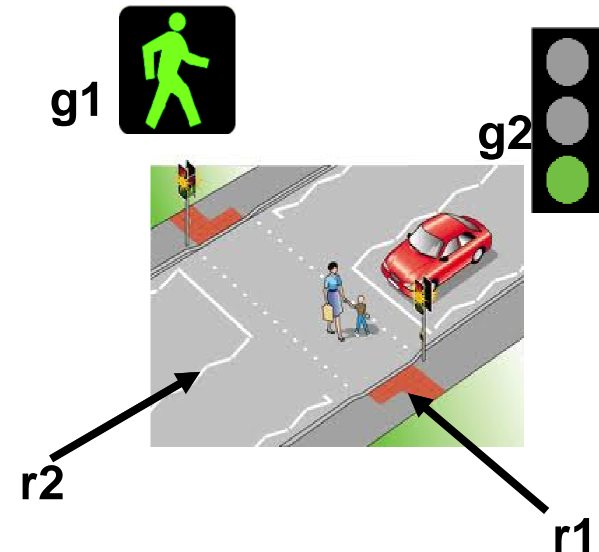
Is the specification correct?

1. always [r1 $\Rightarrow$ next g1 $\wedge$ next next g1]
2. always [$\neg$r1 $\wedge$ $\neg$r2 $\Rightarrow$ next g2]
3. always [$\neg$g1 $\vee$ $\neg$g2]



- Consider the case when r1 is high at time t and low at time t+1, and r2 is low at both time steps.
  - The first property forces g1 to be high at time t+2
  - The second property forces g2 to be high at time t+2
  - The third property says g1 and g2 cannot be high together
  - We have a conflict !!
  - Lets go back to the specification

Pedestrian Crossing: Revised Specs



Properties:

1. Request line r1 has higher priority than request line r2. Whenever r1 goes high, the grant line g1 must be asserted for the next two cycles

   always [r1 $\Rightarrow$ next g1 $\wedge$ next next g1]

2. When none of the request lines are high, the control parks the grant on g2 in the next cycle

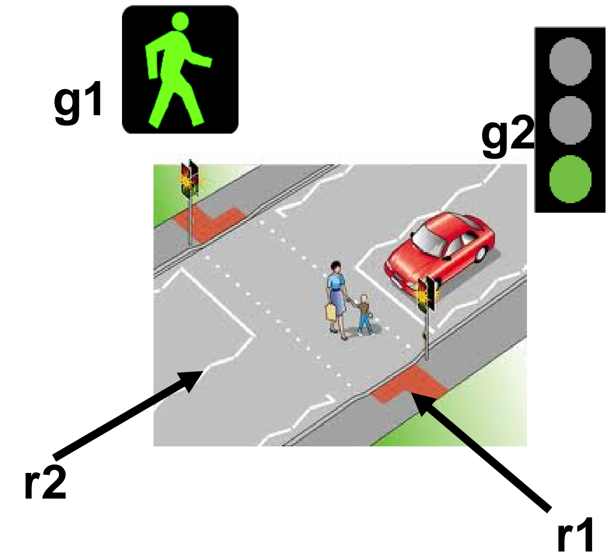
   ~~always [$\neg$r1 $\wedge$ $\neg$r2 $\Rightarrow$ next g2]~~ revised to always [$\neg$g1 $\Rightarrow$ g2]

3. The grant lines g1 and g2 are mutually exclusive

   always [$\neg$g1 $\vee$ $\neg$g2]

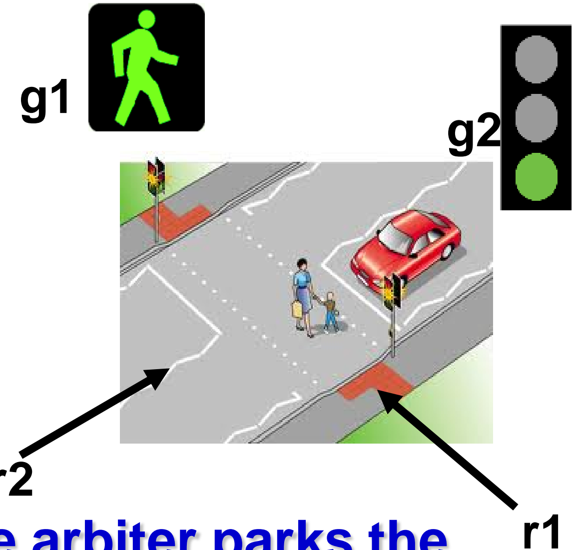
Pedestrian Crossing: Is the specs complete?

1. always [r1 $\Rightarrow$ next g1 $\wedge$ next next g1]
2. always [$\neg$g1 $\Rightarrow$ g2]
3. always [$\neg$g1 $\vee$ $\neg$g2]



- **Observation:** *We can satisfy the specification by designing a control which always asserts g1 and never asserts g2!!*
  - We need to add either of the following types of properties:
    - Ones which specify when g2 should be high, or
    - Ones which specify when g1 should be low
  - Lets go back to the specification

Pedestrian Crossing: Revised specs



Properties:

1. Request line r1 has higher priority than request line r2. Whenever r1 goes high, the grant line g1 must be asserted for the next two cycles

   always [r1 $\Rightarrow$ next g1 $\wedge$ next next g1]

2. When none of the request lines are high, the arbiter parks the grant on g2 in the next cycle

   always [$\neg$g1 $\Rightarrow$ g2]

3. When r1 is low for consecutive cycles, then g1 should be low in the next cycle

   always [$\neg$r1 $\wedge$ next $\neg$r1 $\Rightarrow$ next next $\neg$ g1]

   New!!

4. The grant lines g1 and g2 are mutually exclusive

   always [$\neg$g1 $\vee$ $\neg$g2]

The Model Checking Process

- **Modelling phase**
  - model the system under consideration
  - as a first sanity check, perform some simulations
  - formalise the property to be checked

- **Running phase**
  - run the model checker
  - check the validity of the property in the model

- **Analysis phase**
  - property satisfied? → check next property (if any)
  - property violated? →
    - analyze generated counter-example by simulation
    - refine the model, design, or property … and repeat the entire procedure
  - out of memory? → try to reduce the model and try again

The Merits/Demerits of Model Checking

The Pros of Model Checking

- widely applicable (hardware, software, protocol systems, ...)
- allows for partial verification (only most relevant properties)
- potential "push-button" technology (hw/sw-tools)
- rapidly increasing industrial interest
- in case of property violation, a counterexample is provided sound and interesting mathematical foundations
- not biased to the most possible scenarios (such as testing)

The Cons of Model Checking

- main focus on control-intensive applications (less data-oriented)
- model checking is only as "good" as the system model
- no guarantee about completeness of results
- impossible to check generalisations (in general)

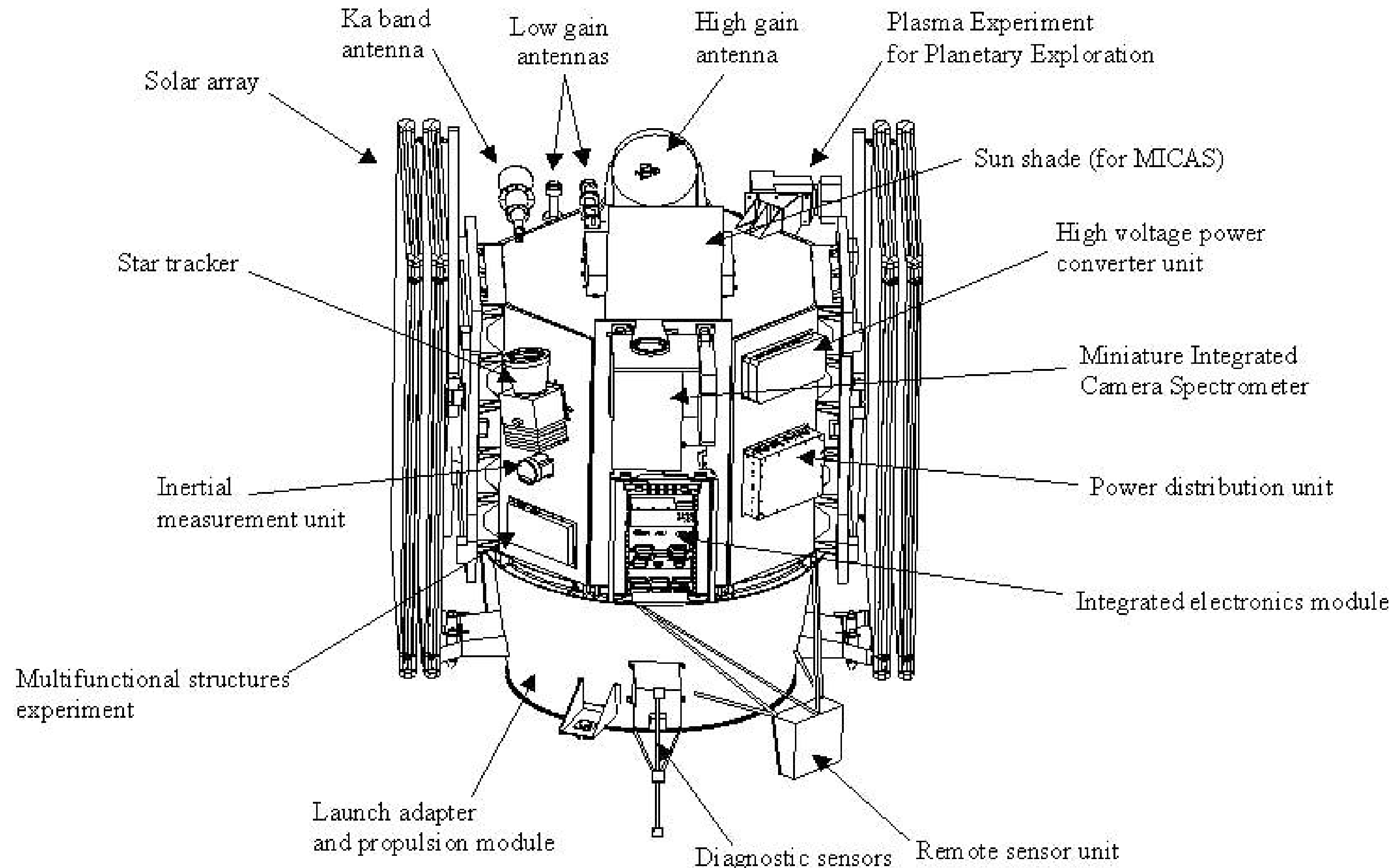
Striking Model Checking Examples

- Security: Needham-Schroeder encryption protocol
  - error that remained undiscovered for 17 years unrevealed
- Transportation systems
  - train model containing 10^{476} states
- Model checkers for C, Java and C++
  - used (and developed) by Microsoft, Digital, NASA
  - successful application area: device drivers
- Dutch storm surge barrier in Nieuwe Waterweg
- Software in current/next generation of space missiles
  - NASA's Mars Pathfinder, Deep Space-1, JPL LARS group

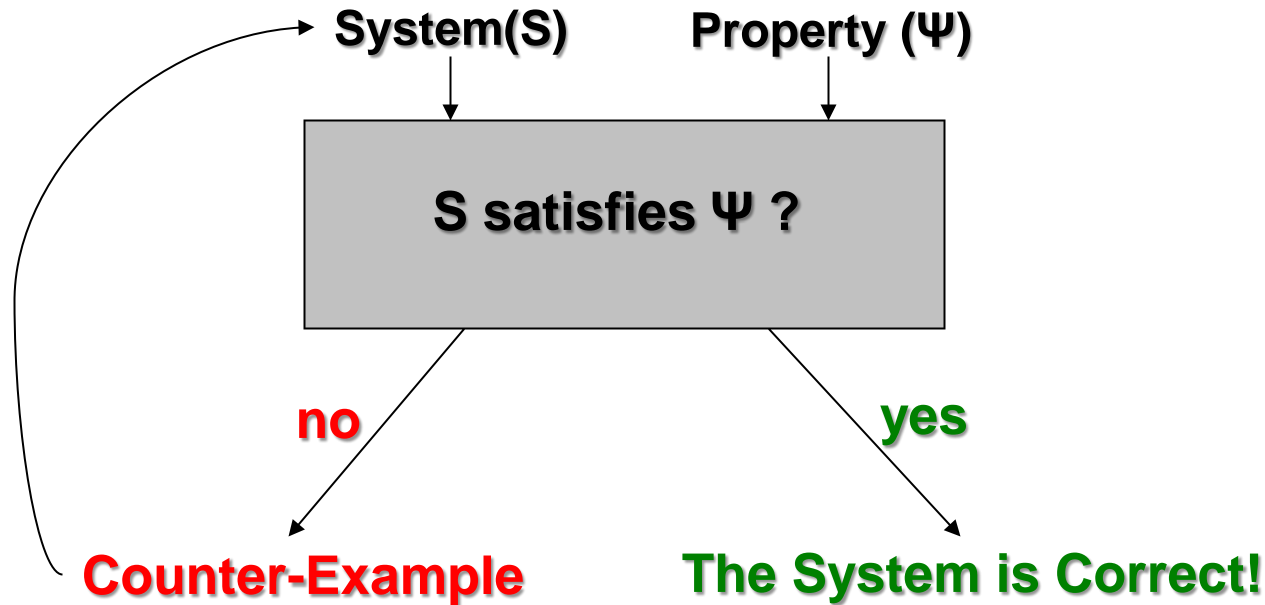
Model Checking Examples (contd…)

NASA's Deep Space-1 Spacecraft: (Launched in October 1998)

Model checking applied to several modules of this spacecraft



Design Verification Recap …



S satisfies Ψ