Automatic Generation of Schedulings for Improving the Test Coverage of Systems-on-a-Chip

Claude Helmstetter
and Florence Maraninchi
and Laurent Maillet-Contoz
and Matthieu Moy

Verimag &
ST Microelectronics

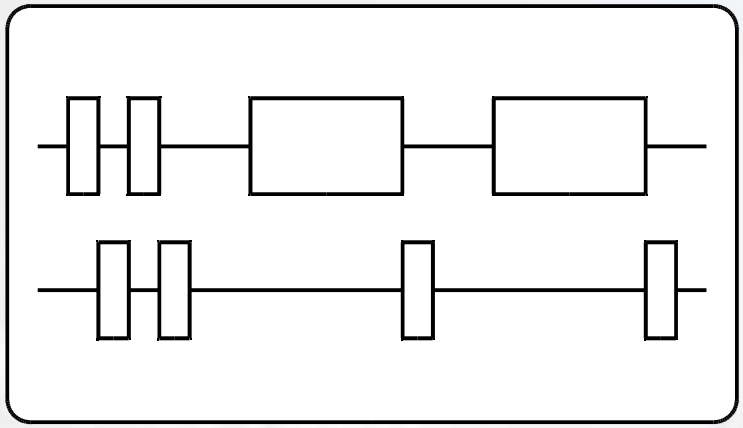
FMCAD'06

Context: Transaction Level Models

fastest

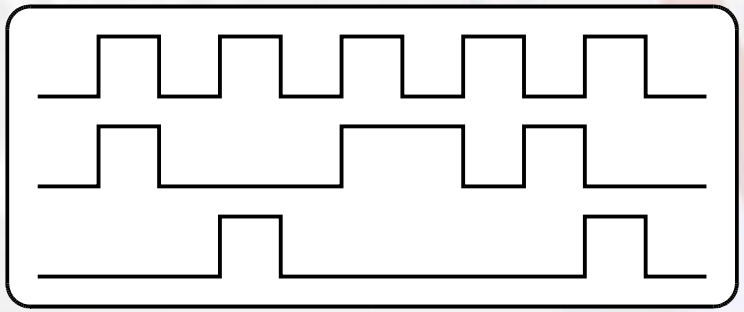
Early simulation of the embedded software

Golden model for RTL validation



TLM

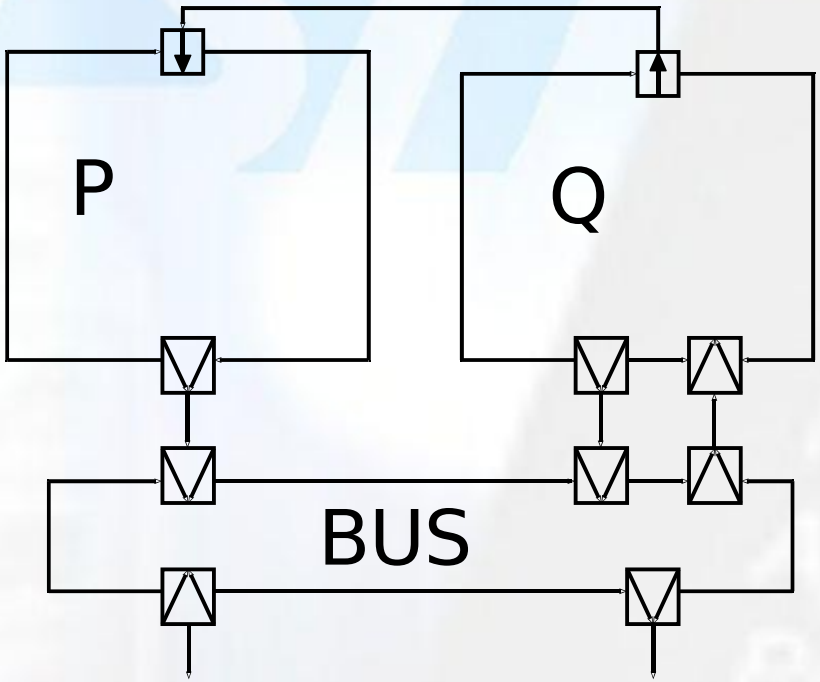
System-on-a-Chip (SoC) synthesis



RTL

most precise

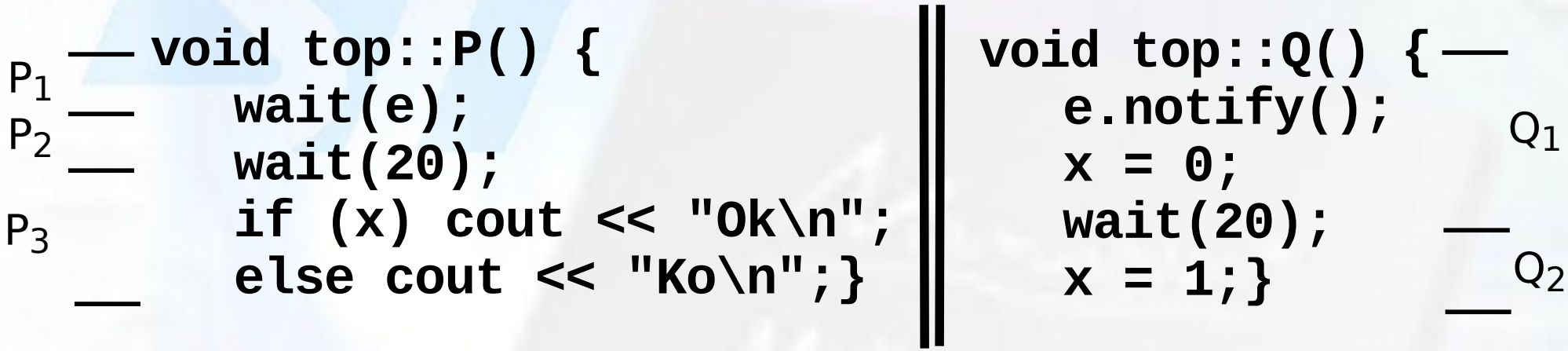
Context: SystemC, a C++ Library




```
...
unsigned x;
sc_event e;
SC_HAS_PROCESS(top);
top(sc_module_name name):
        sc_module(name) {
  SC_THREAD(P);
  SC_THREAD(Q);
}
void top::P() {
  wait(e);
  ...
```

Elaboration phase, non-premptive scheduling, simulated time.

Example of Scheduling Dependencies



- **3** possible schedulings: (TE=Time Elapse)
  - $P_1;Q_1;P_2;[TE];Q_2;P_3$: **Ok**
    default OSCI scheduler choice, if **P** declared before **Q and** if ...
  - $P_1;Q_1;P_2;[TE];P_3;Q_2$: **Ko**
  - $Q_1;P_1;[TE];Q_2$: **“dead-lock”**

The Coverage Problem

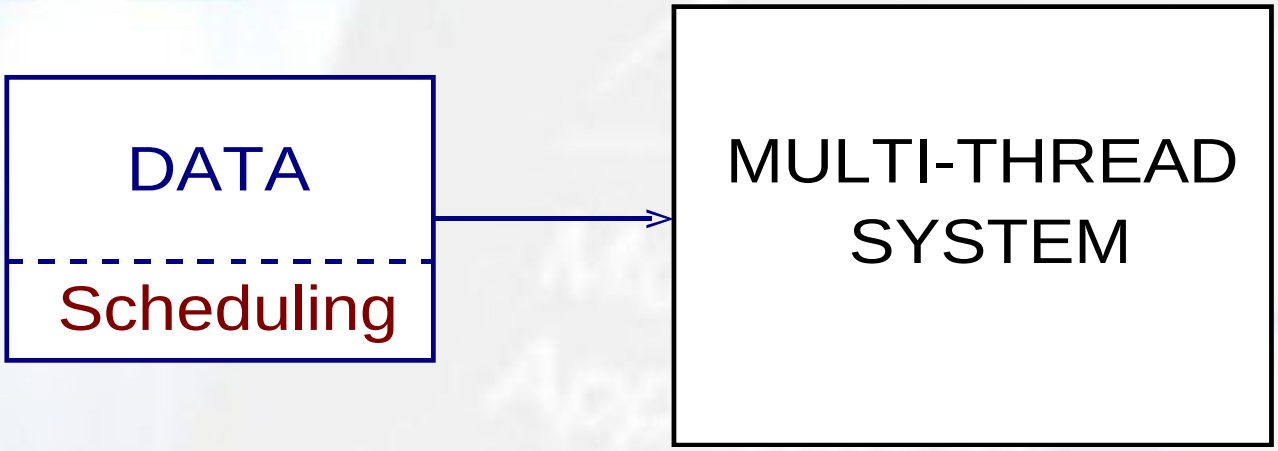
- Even if data is fixed
  - The SystemC LRM allows many schedulings
  - Some implementations are not deterministic
- For the validation of SoC models:
  - 1 execution => very poor coverage
  - Random schedulings => uncertain coverage, lots of useless executions
  - Test with all possible schedulings => unrealistic
- Our goal: **test only executions that may lead to different final states**

Outline

- TLM, SystemC and the Coverage Problem
- **Principle of the Technique Applied**
- Implementation and Results
- Conclusion

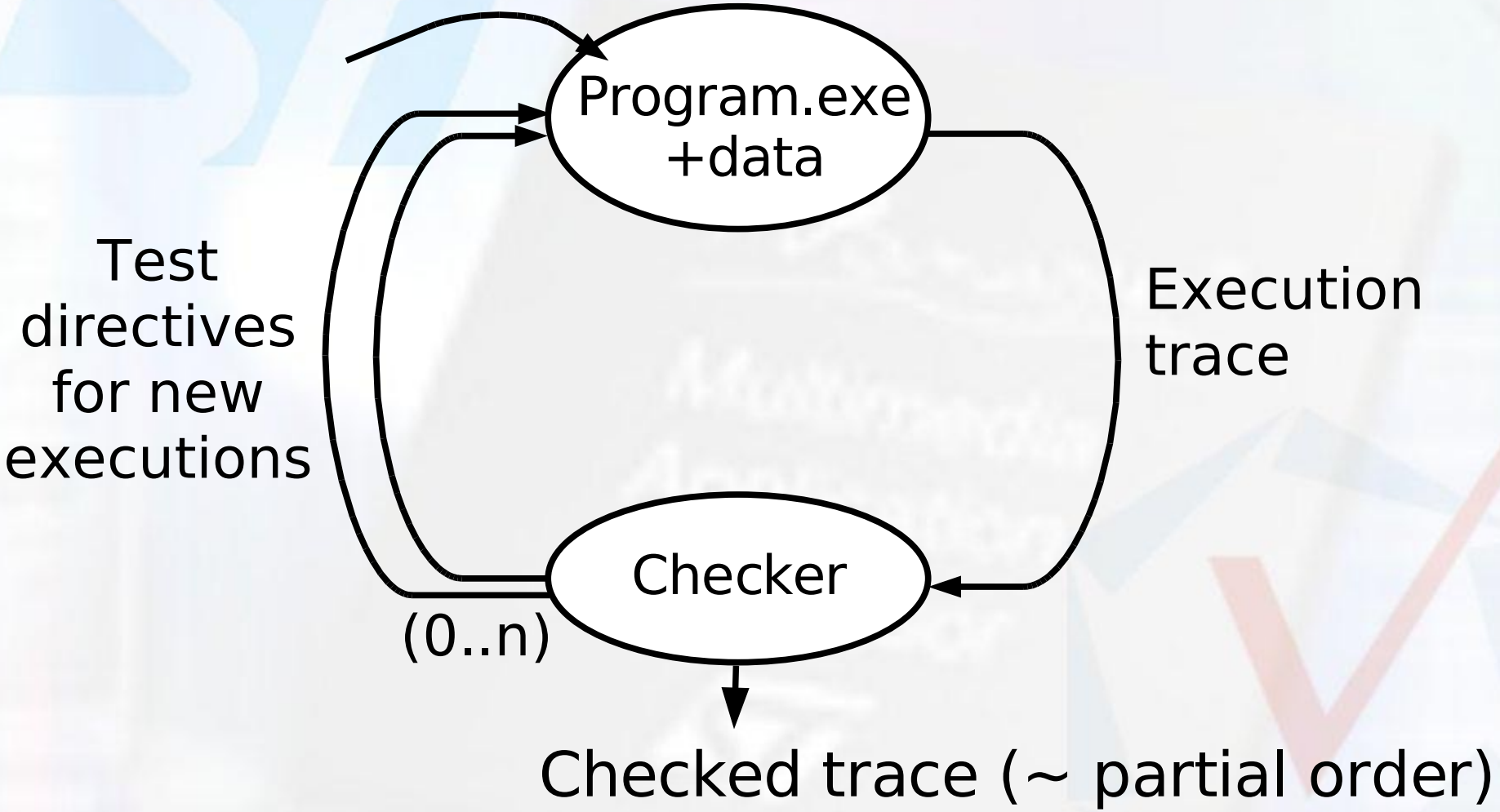
Principle of the Approach

Data is **fixed**; we generate schedulings

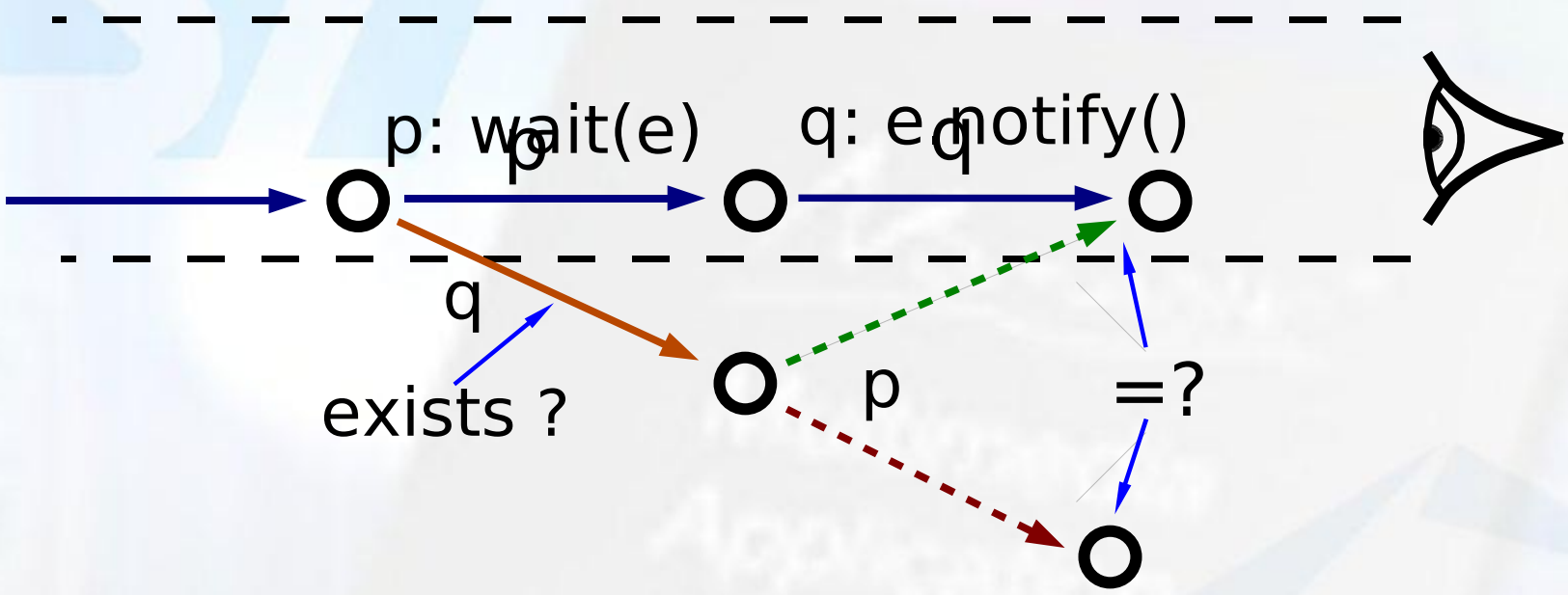


Use of Dynamic Partial Order Reductions
(presented by C.Flanagan, P.Godefroid
at POPL'05)

Cyclic Generation



Checker: Observing Traces



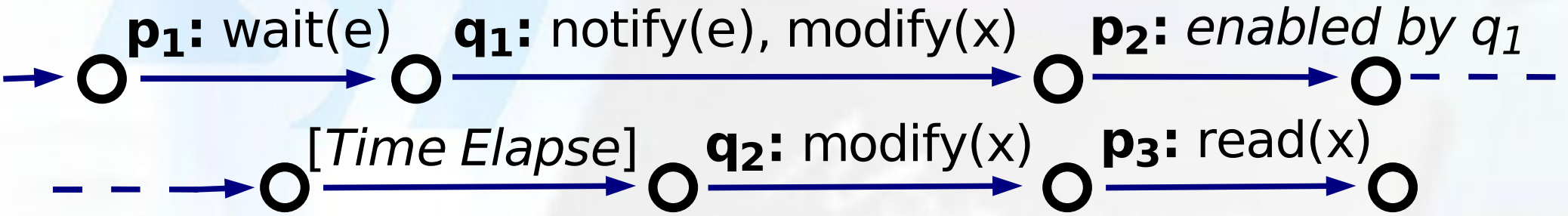
Goal:
Guess if transitions are dependent by observation of their behavior

Checker: Action Dependencies

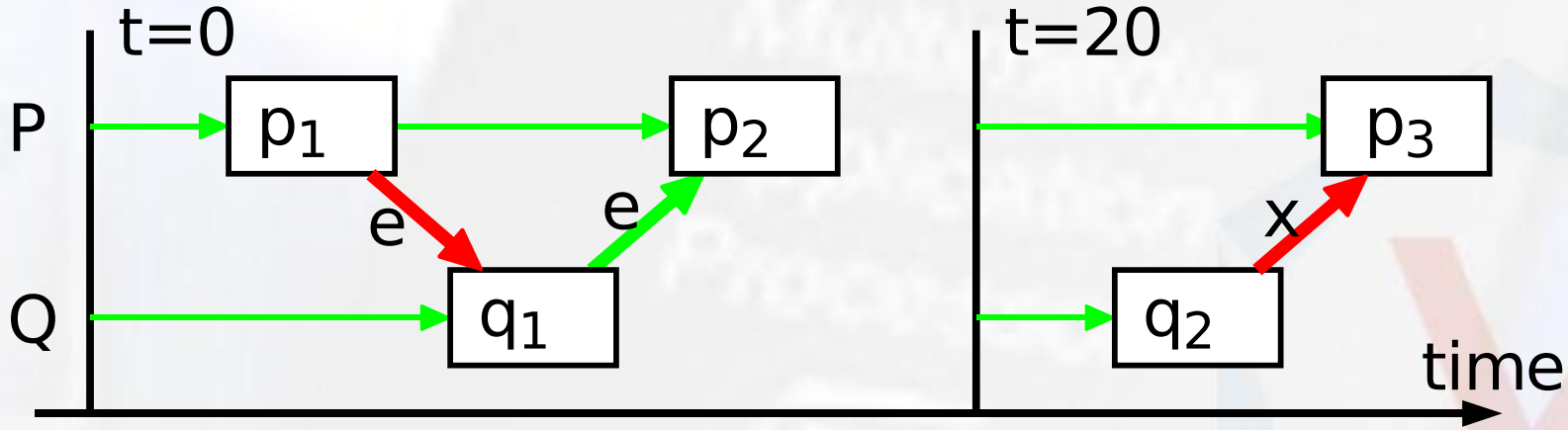
- Independent <=> order is irrelevant
- Dependency cases for SystemC:
  - Variables (or memory locations):
    - Two **`write`** (`T[12]=1` and `T[12]=2`)
    - One **`write`** and one **`read`** (`x=1` and `f(x)`)
  - Events:
    - One **`notify`** and one **`wait`**
    - In some cases: two **`notify`**
      (consequences on the computed partial order)

Checker: Dynamic Dependency Graph

Execution Trace:



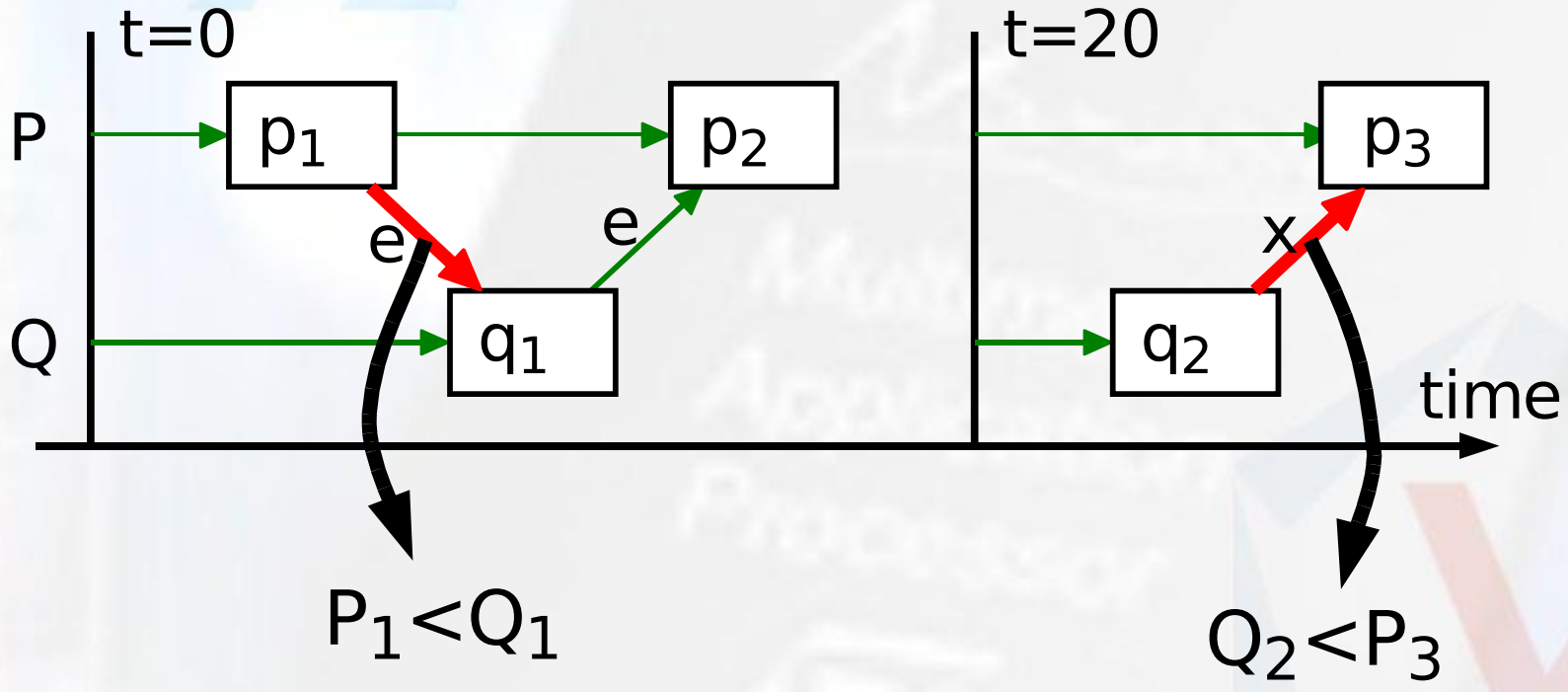
Dynamic Dependency Graph:



Green arrows: dependent but **not permutable**
Red arrows: dependent and **permutable**

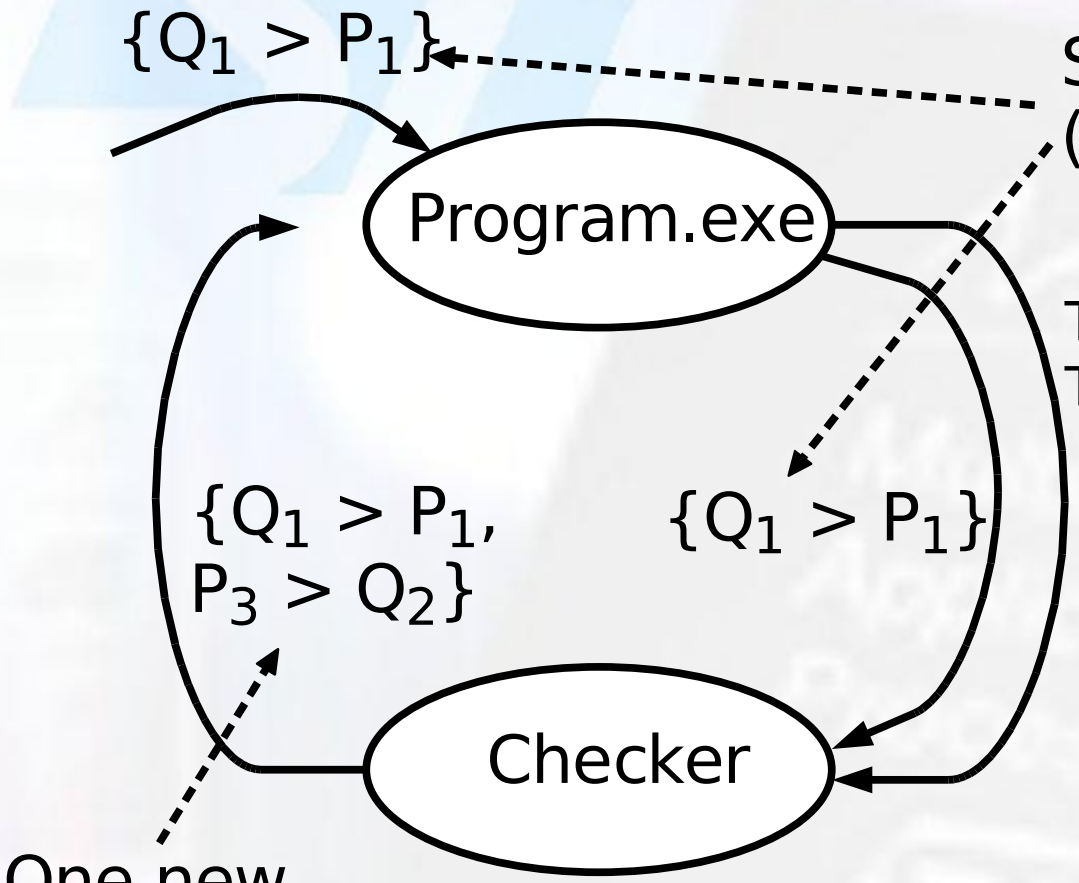
Checker: Scheduling Constraint

Generation of 1 new test directive for each red arrows



p_i<q_j: **i**-th execution of process **p** before **j**-th execution of process **q**

Cyclic Generation with Scheduling Constraints



Set of inherited constraints (from previous checking)

TRACE

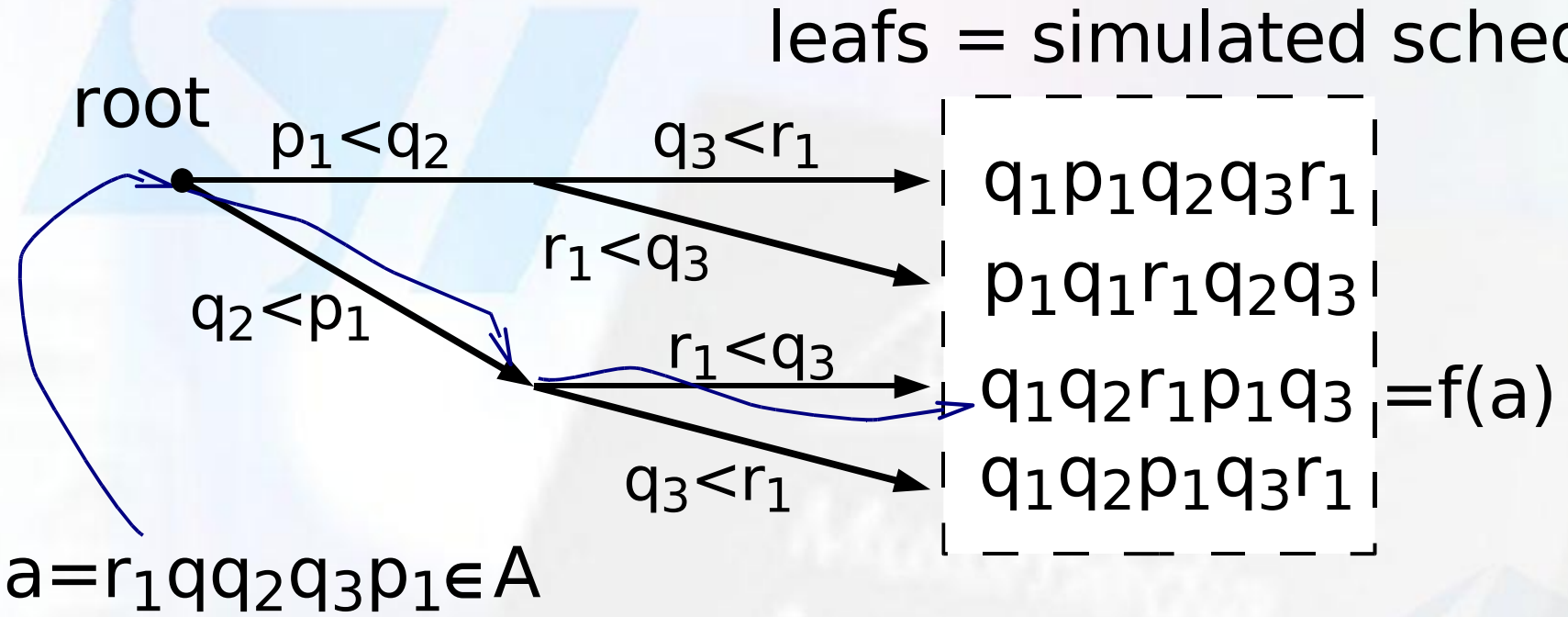
Transition	Actions
P_1	wait(e)
Q_1	notify(e), modify(x)
P_2	enabled by Q_1
TE	
P_3	read(x)
Q_2	modify(x)

Property Guaranteed by this Method

- **A:** Set of all possible executions (for one data)
- **G:** Set of generated executions (for the same data)

- **Property:** For all ***a*** in **A**, there exists **g** in **G** that differs only by the order of independent transitions.
- **Consequences on coverage:**
  - Full code accessibility for each process
  - All Dead-locks found

Proof Hint: Constraint Trees

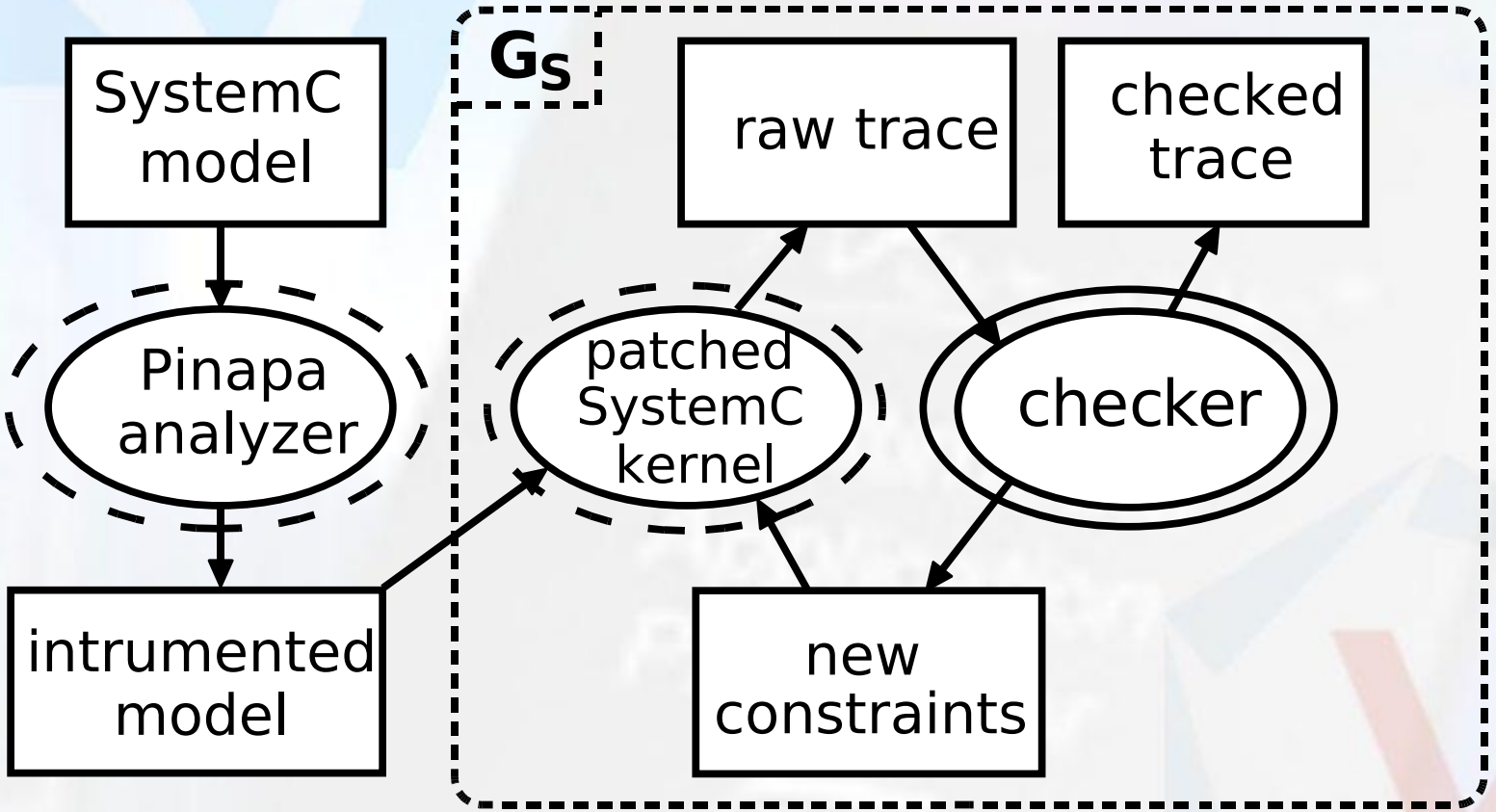


- Define a function f from A to G
- a and f(a) differ only by the order of independent transitions.

Outline

- TLM, SystemC and the Coverage Problem
- Principle of the Technique Applied
- **Implementation and Results**
- Conclusion & Demo

The Tool Chain



Industrial Case Study: LCMPEG

- Part of a Set-Top Box, from STM
- 5 components, runs of 150 transitions, with long sections of sequential code (~50klines)

- 32 generated schedulings (=*card(G)*), 13 sec
- 9.5 sec for the 32 simulations / 3.5 sec of overhead (time spent in checker)
- At least 2^40 possible schedulings (=*card(A)*)

Extension: Validation of SoC models in the presence of loose timings

- New instruction: **`lwait(42±12)`** for representing unprecise timings
  - basic implementation: random in [30,54]
  - better: DPOR + Linear Programming
- Results : LCMPEG with delays ± 20% => 3584 simulations, 35 min 11 sec.
- Presented at FMICS'06

Conclusion

- Already works on medium-sized industrial case studies
- Should work on larger case studies with some improvements
- Well adapted to abstract TLM models which are asynchronous
- Light tool: no explicit extraction of an abstract formal model, no state comparison, ...

Further Works

- Validation of SoC models in the presence of loose timings (presented at FMICS'06)
- Possible optimizations:
  - higher level synchronization mechanisms (persistent events)
  - remove useless branches of constraint trees
  - parallelization of the prototype
- Parallelization of the SystemC engine (based on dependency analysis too)

Thank you for your attention.

Persistent Events

- Process A: v = 1; e.notify();
- Process B: if (!v) wait(e); v = 0;

- Consequence: useless simulations
- Solution:
  - new class pevent with methods wait, notify and reset
  - extending dependency analysis
- Result: from 128 to 32 generated schedulings for the LCMPEG