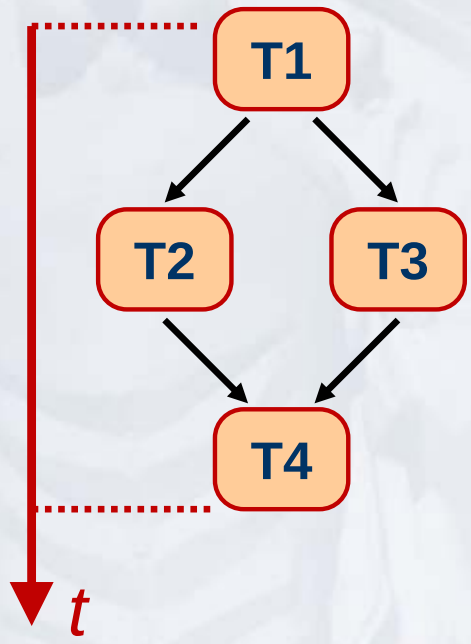
Timing Analysis of Real-Time Software

Raimund Kirner

Vienna University of Technology
Austria

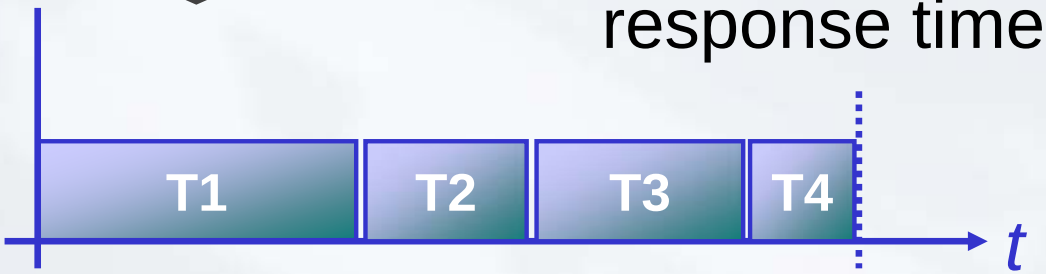
This is joint work with Peter Puschner and the CoSTA and ForTAS project teams.

From RTS Design to Implementation



Task set with precedence constraints and deadline

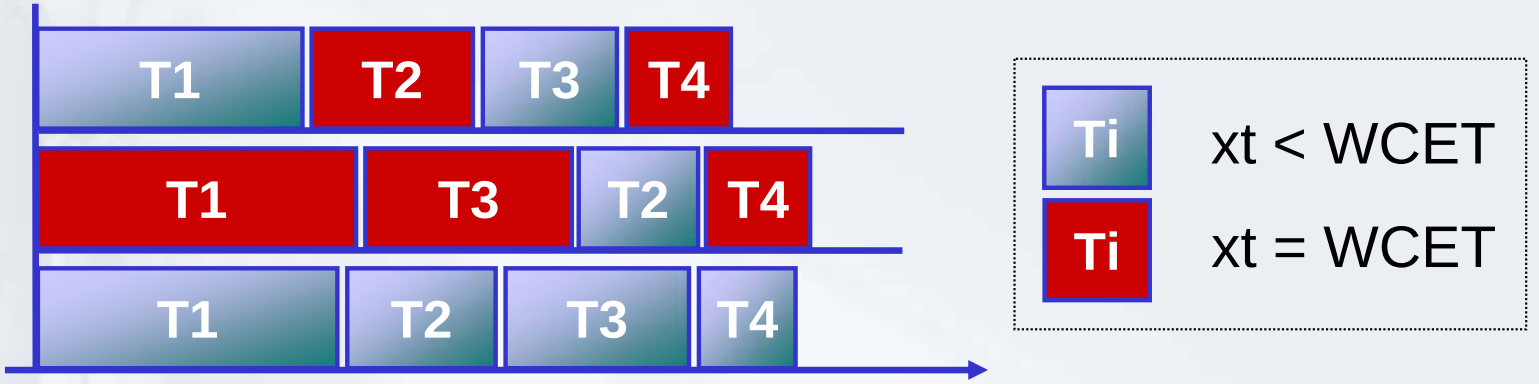
Task sequence: execution times, response time



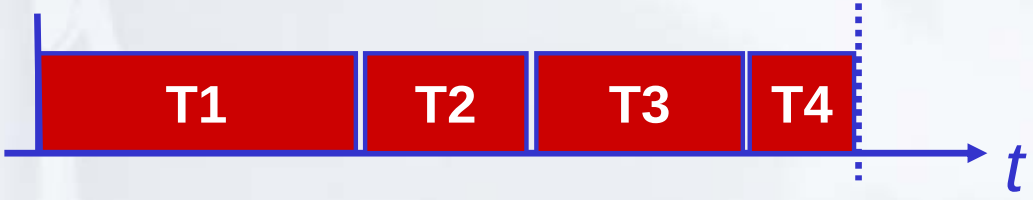
Can we guarantee that: response time < deadline?

WCET: Timing Analysis Abstraction

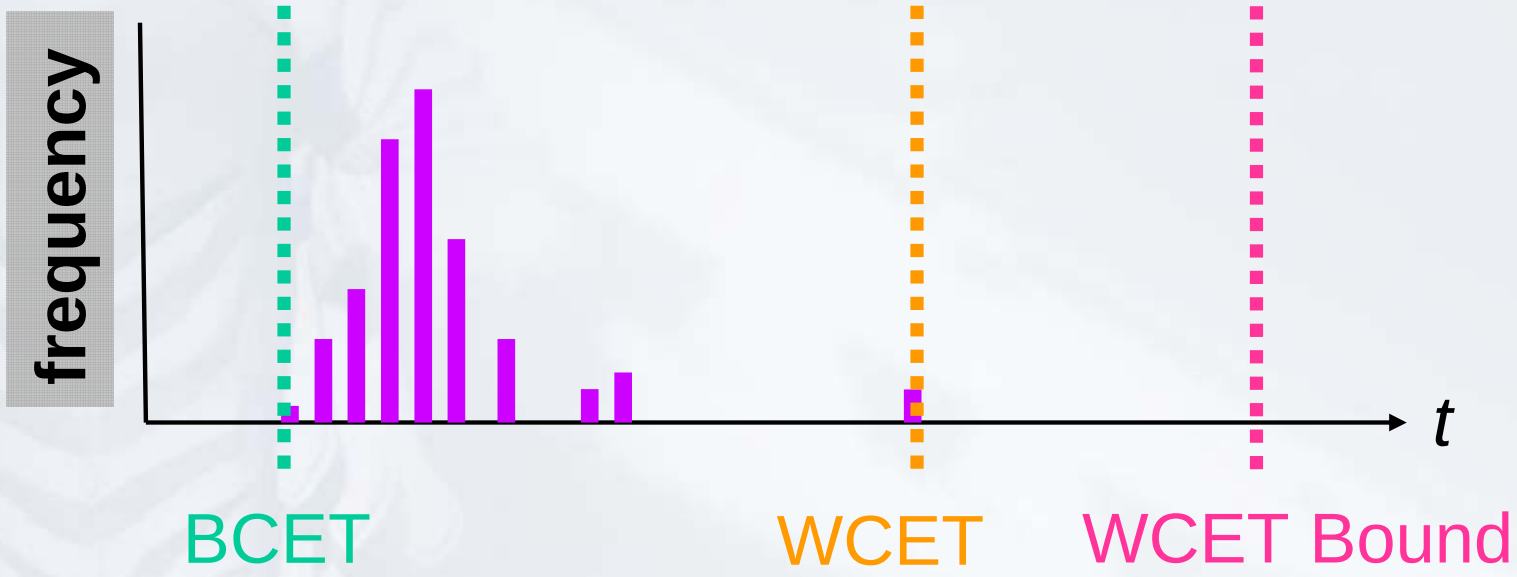
In general it is infeasible to model all possible execution scenarios and combinations of task execution times



WCET analysis abstracts the different execution times of each task to one single value ⇨ WCET bound



WCET vs. calculated WCET bound



Remarks on WCET Analysis

- Computes upper bounds
- Bounds are application-dependent
- Assesses time that processor is actually executing the code
- WCET result is hardware-dependent

⇨ WCET bounds must be safe

⇨ WCET bounds should be tight

Requirements on WCET Analysis Tools

Find feasible abstractions and analysis methods such that:

1. development effort of WCET tool is affordable
2. the calculated WCET estimates are sufficiently precise
3. analysis problems are tractable with acceptable resource requirements, and
4. the WCET tool is easy to use

Acceptance depends on application domain…

Tradeoffs required: there is no single WCET analysis technique that is well-suited for all application domains!!

The Path Problem

- The path problem: calculate a description or enumeration of the (in)feasible paths of a program
- Any brute-force approaches like executing or simulating the program with all possible input data are intractable (the different values of input data are even more than different paths exist)
- Problem is shifted to the user: request for manual path descriptions (requires experts, high effort, is error-prone)
- Program analyzes with right abstractions help to reduce the needed manual code annotations.

Program Annotations for WCET Analysis

- Explicit flow information required to guide WCET analysis:
  - intractable program complexity
  - description of execution modes or input data

```
scope
{
    for (i=0; i<N; i++)
    {
      maximum N iterations;
      for (j=0; j<i; j++)
      {
        maximum N iterations;
        marker m1;
            …
      }
    }
  restriction m1 == N * (N+1) / 2;
}
```

loop bound

loop bound

flow variable

linear flow constraint

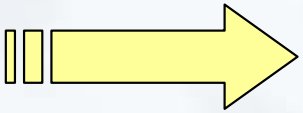
$$1 \bullet f_{m1} == [N \bullet (N+1)/2] \bullet f_{SCOPE}$$

Challenge: Correctness of Flow Information after Code Optimization

```
for (i=0; i<N; i++)
maximum 10 iterations
{
   f(i);
}
```

additional knowledge assumed: N≤10 (flow fact given as code annotation)

loop unrolling (unrolling factor 3)



```
for (i=0; i<(N-2); i=i+3)
{
   f(i); f(i+1); f(i+2);
}
for (;i<N; i++)
{
 f(i);
}
```

Q: what flow information is known of the transformed code?

Challenge: Correctness of Flow Information after Code Optimization

```
for (i=0; i<N; i++)
maximum 10 iterations
{
   f(i);
}
```

additional knowledge assumed: N≤10 (flow fact given as code annotation)

loop unrolling (unrolling factor 3)

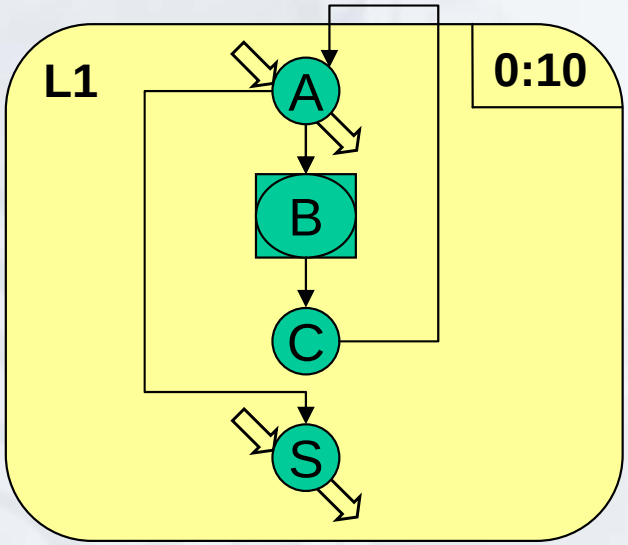
```
scope {
   for (i=0; i<(N-2); i=i+3)
   maximum 3 iterations                    loop bound
   {
      marker m1;                           flow variable
      f(i);  f(i+1);  f(i+2);
   }
   for (;i<N; i++)
   maximum 2 iterations                    loop bound
   {
      marker m2;                           flow variable
      f(i);
   }
   restriction 3*m1 + m2 <= 10;
}                                          linear flow constraint
```

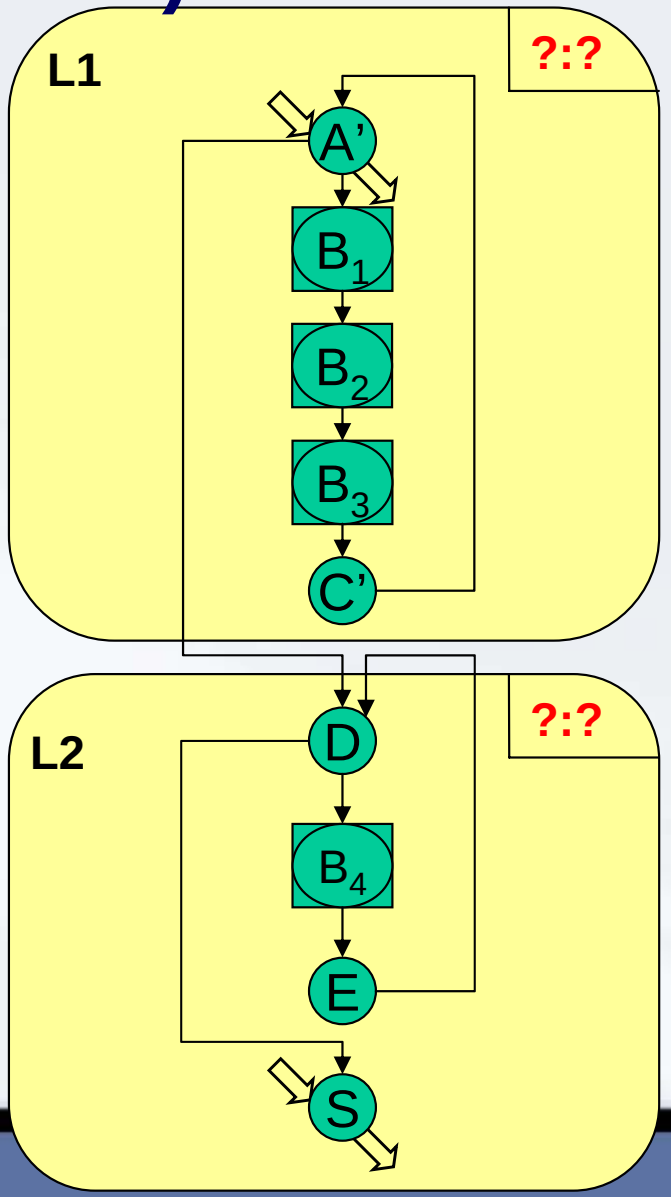
Automatic update of flow information in parallel to code transformation

Universal Flow-Information Update

- We developed a framework that allows to update flow information for *arbitrary code transformations* using rules composed of the following operations:
  - Update of loop bound information ($\xrightarrow{L}$):
    (create, modify, or delete loop bound information)
  - Update of flow constraints ($\xrightarrow{R}$):
    (transform terms of the form "const • flowvariable" of a linear flow constraint into a new term respectively a sum of terms, creation of new flow constraints)

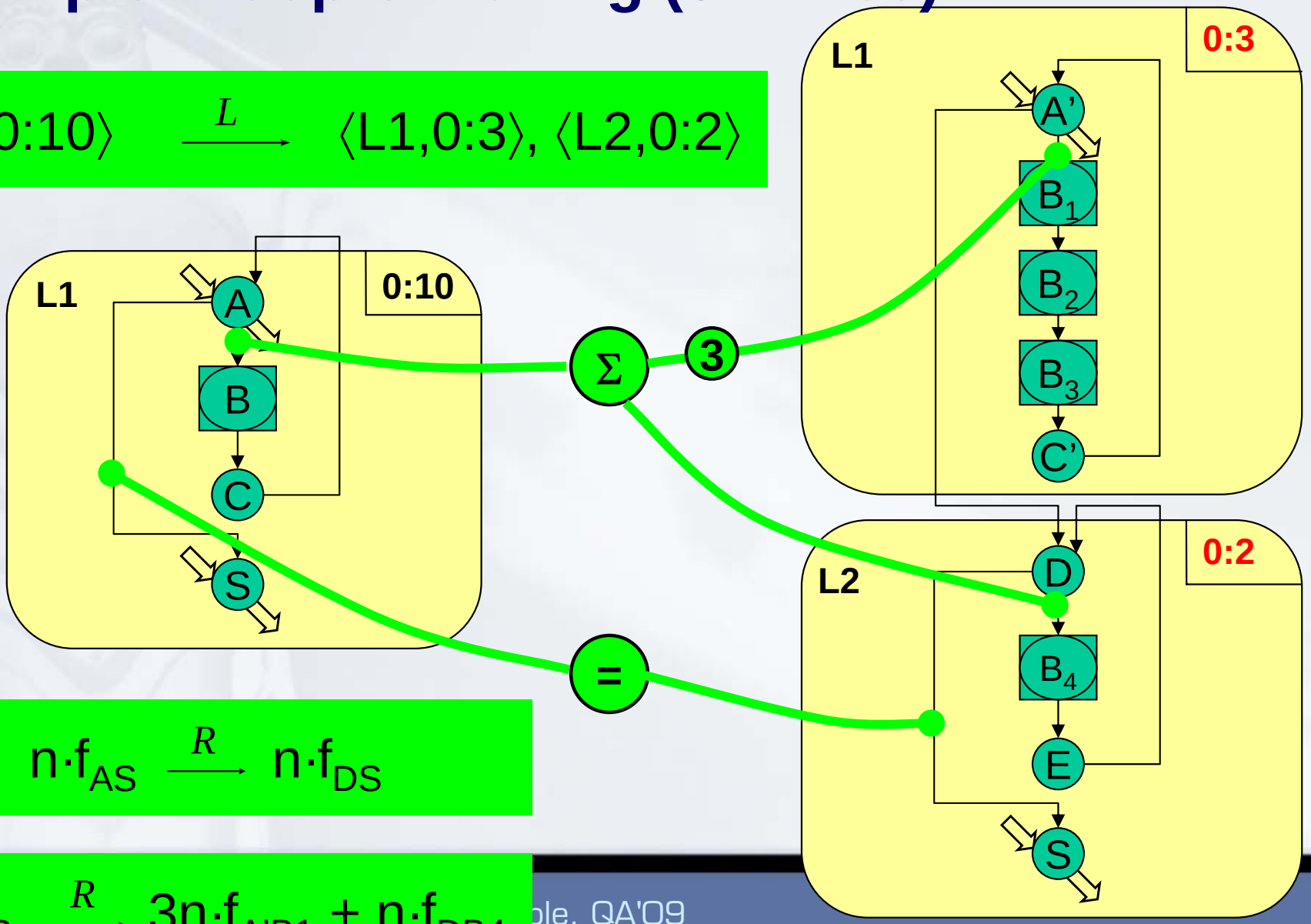
Example: Loop Unrolling (3 Times)





Example: Loop Unrolling (3 Times)

$\langle L1,0:10\rangle \xrightarrow{L} \langle L1,0:3\rangle, \langle L2,0:2\rangle$



$n \cdot f_{AS} \xrightarrow{R} n \cdot f_{DS}$

$n \cdot f_{AB} \xrightarrow{R} 3n \cdot f_{A'B1} + n \cdot f_{DB4}$

Universal Flow-Information Update

Advantages:

- Manual code annotations:
  Reduced cognitive complexity
  (no need anymore to annotate machine code)

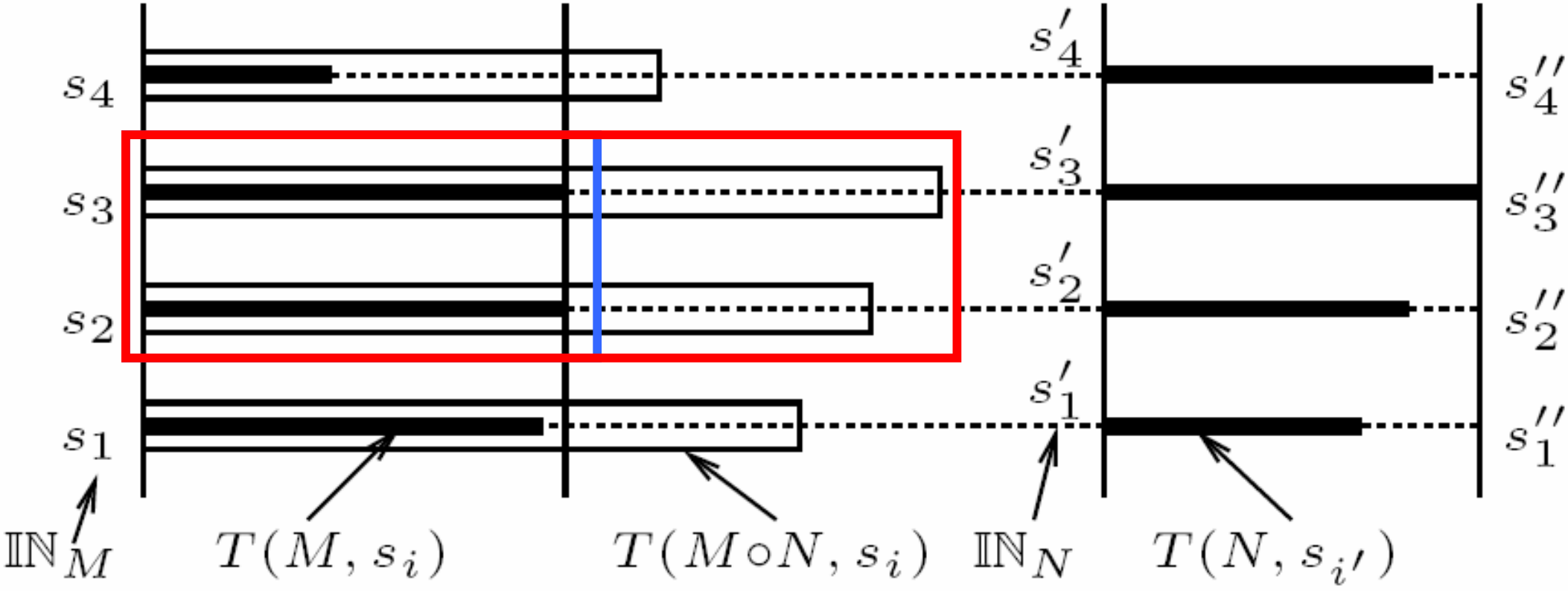
- Automatic calculation of flow information:
  Platform-independent calculation with reduced effort
  (information more explicit available at source code)

The State Explosion Problem

- Processor behavior modeling faces the problem of state explosion
  - instruction timing depends on context (execution history)
  - caches, pipelines, etc.
  - even, when using the timing relevant dynamic processor state (TRDPS)
- The desired solution:
  decompose the timing analysis problem using "divide and conquer"

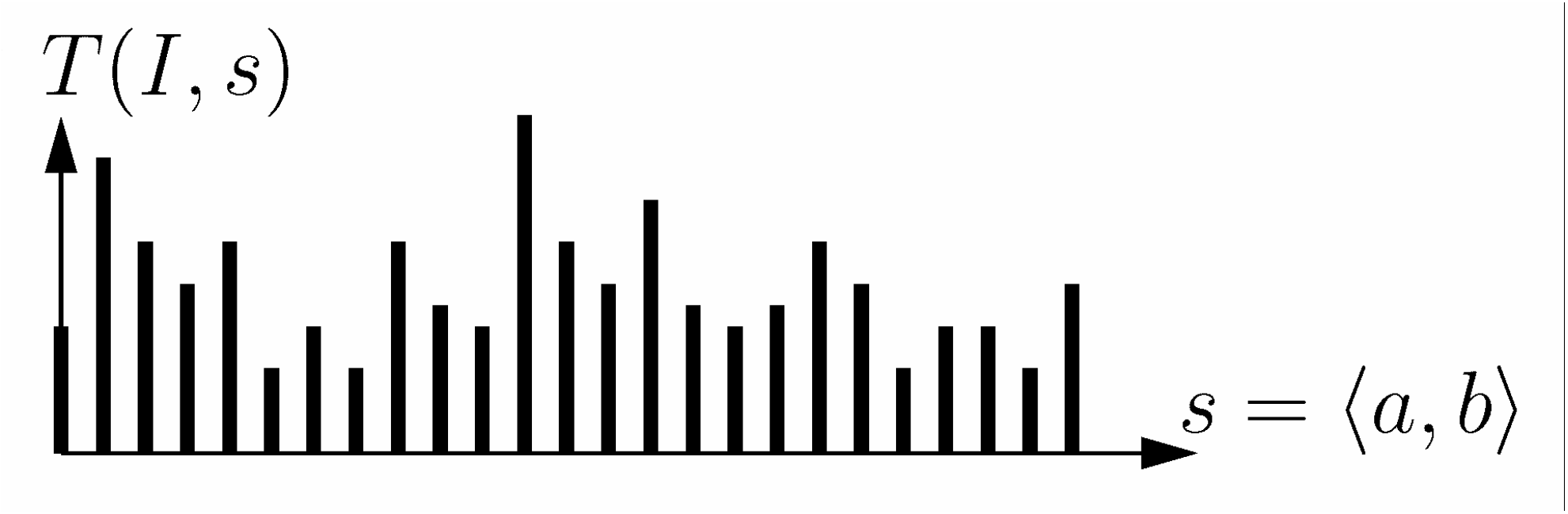
Series Decomposition

- Analysis on control-flow graphs instead on the set of execution traces

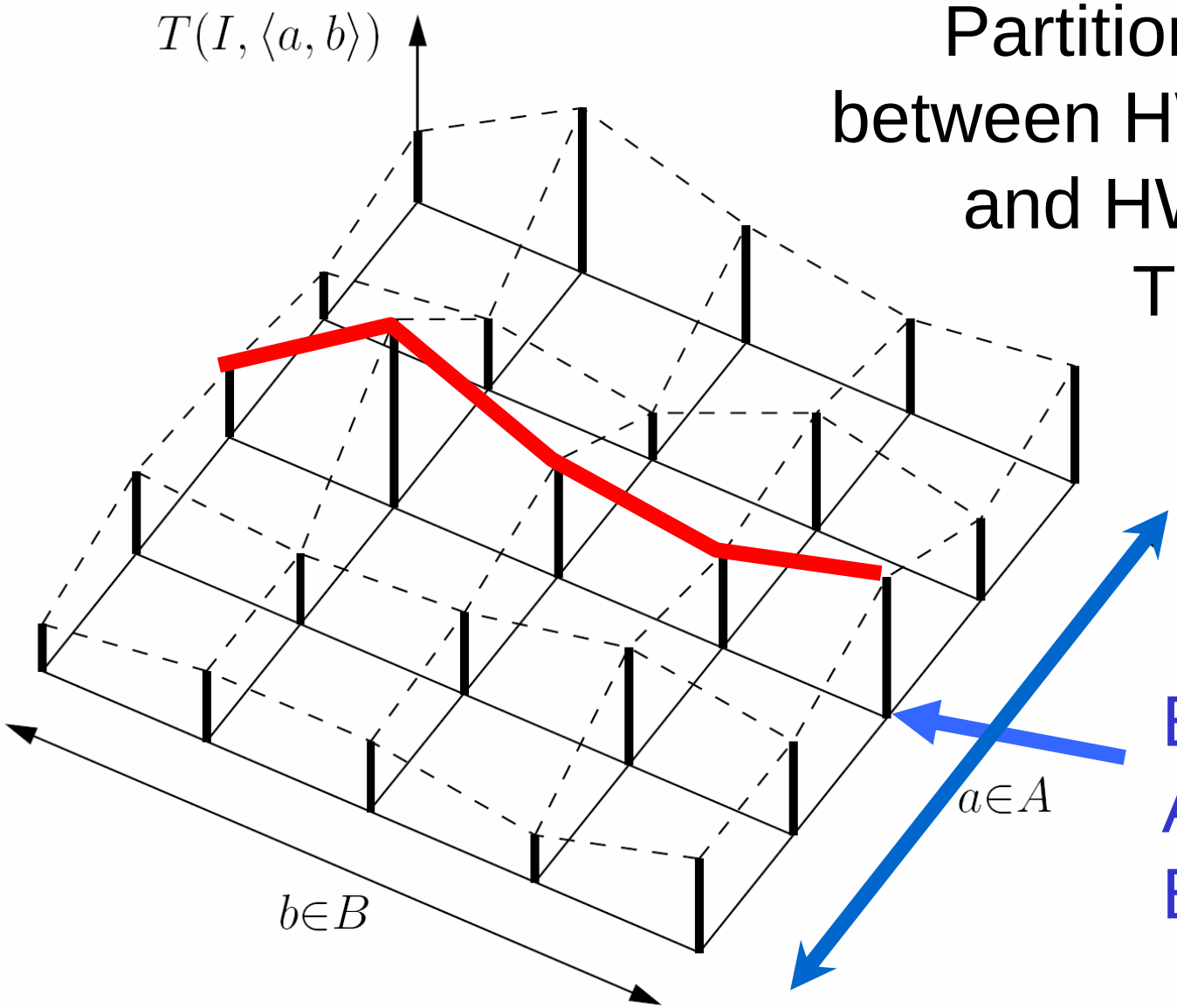


Parallel Composition

- The execution time *T(I,s)* of an instruction sequence *I* depends on the TRDPS *s*:



Parallel Composition (TRDPS Partitioning)



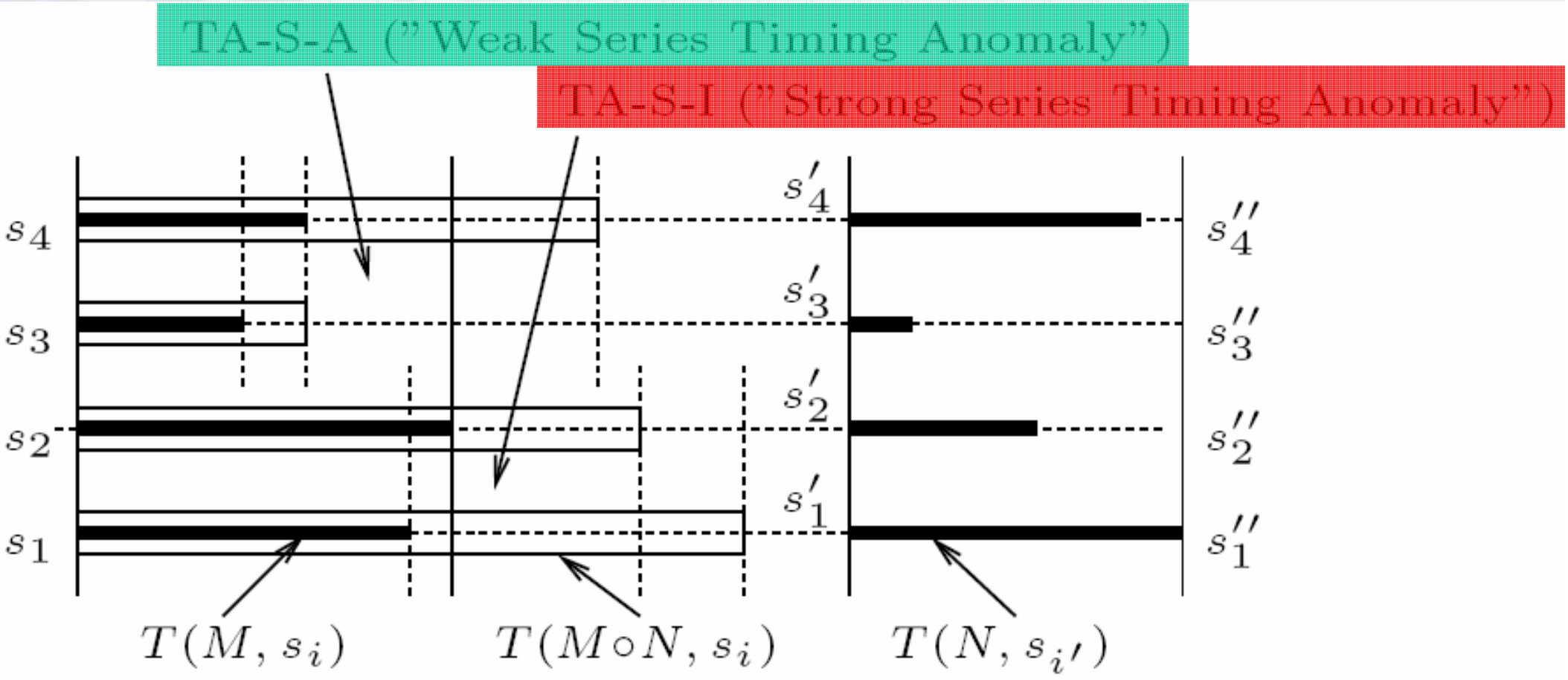
Partitioning the TRDPS between HW component A and HW component B: TRDPS: A × B

Example:
A … cache state
B … pipeline state

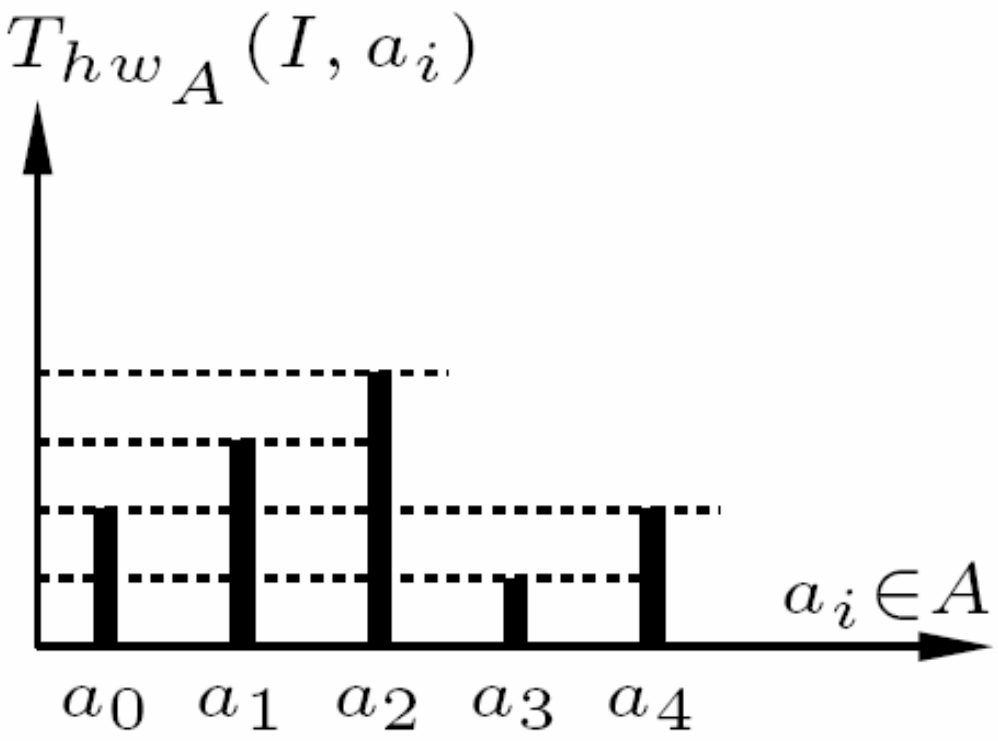
Is it that simple?

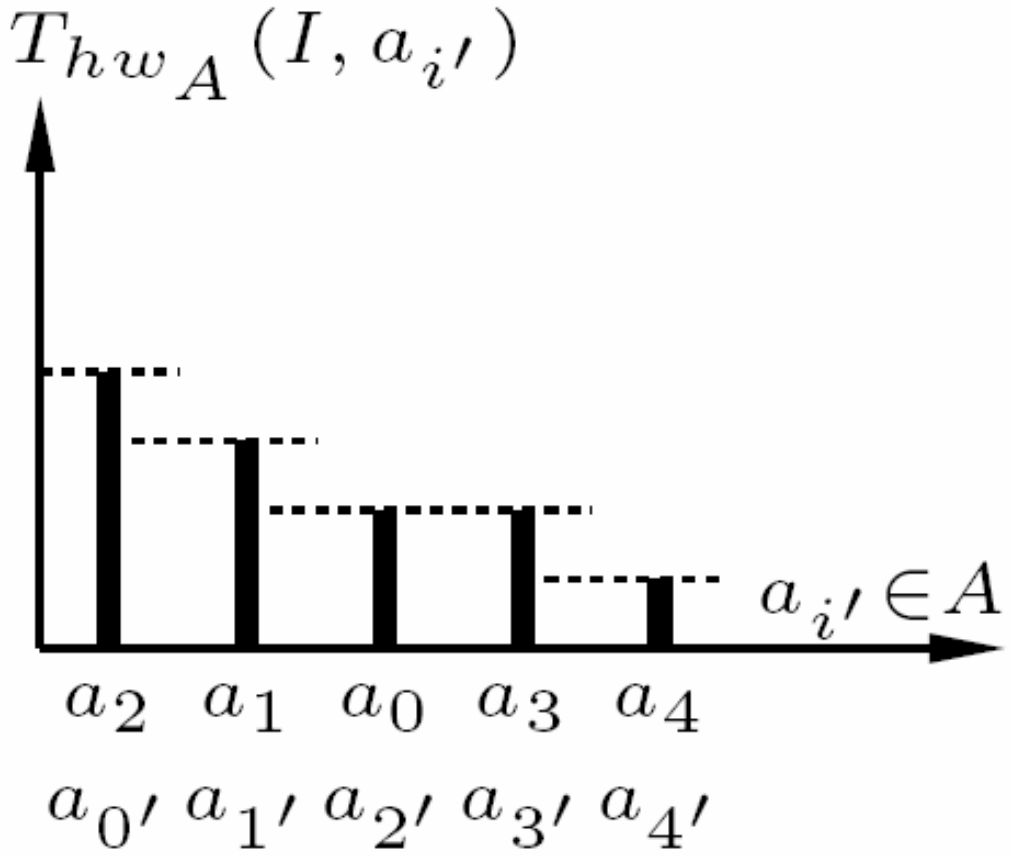
Using Newton's world view together with an inadequate observation system may cause underestimation of the timing effects at a local system!

Pitfall: Series Timing Anomalies

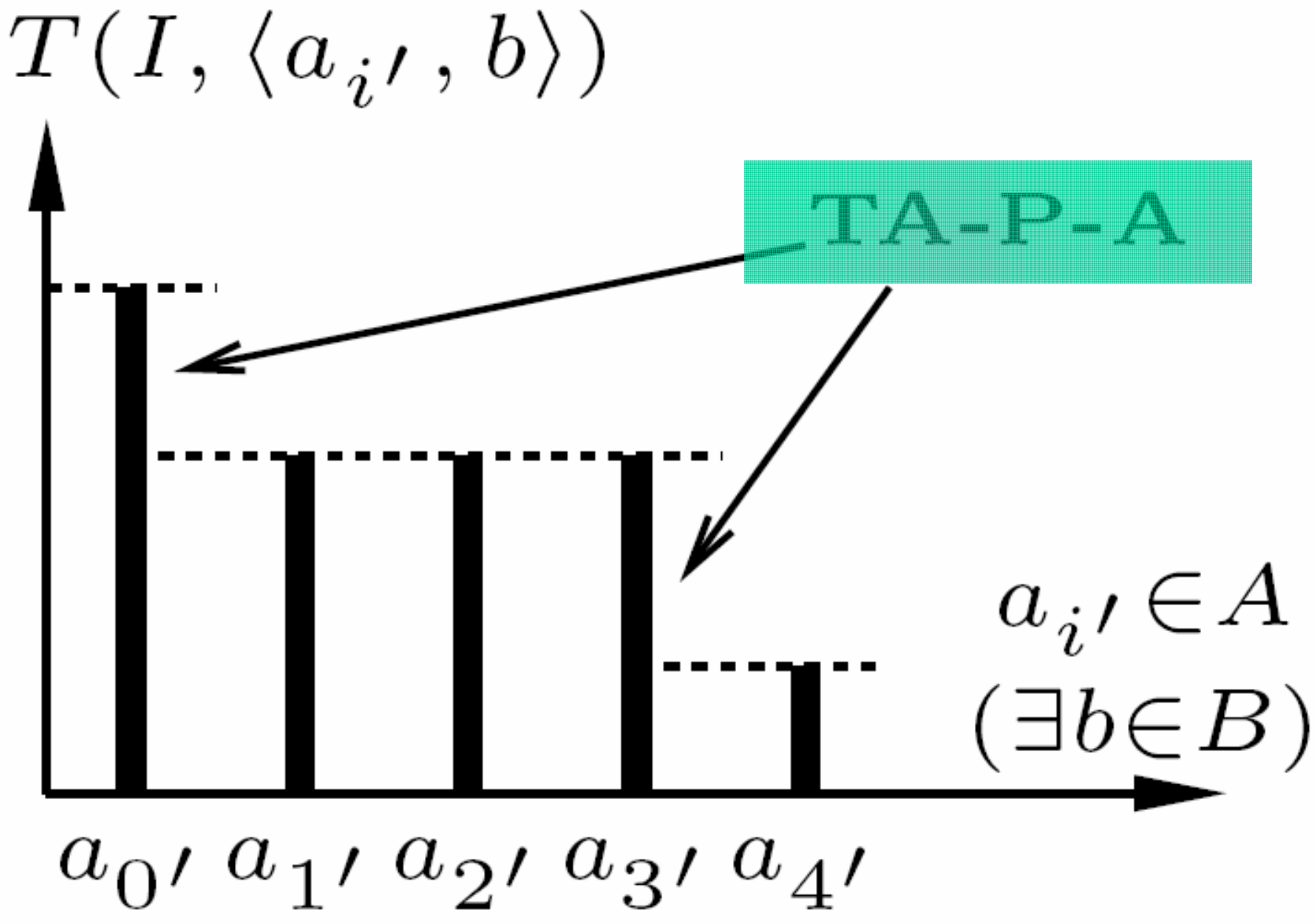


Pitfall: Parallel Timing Anomalies

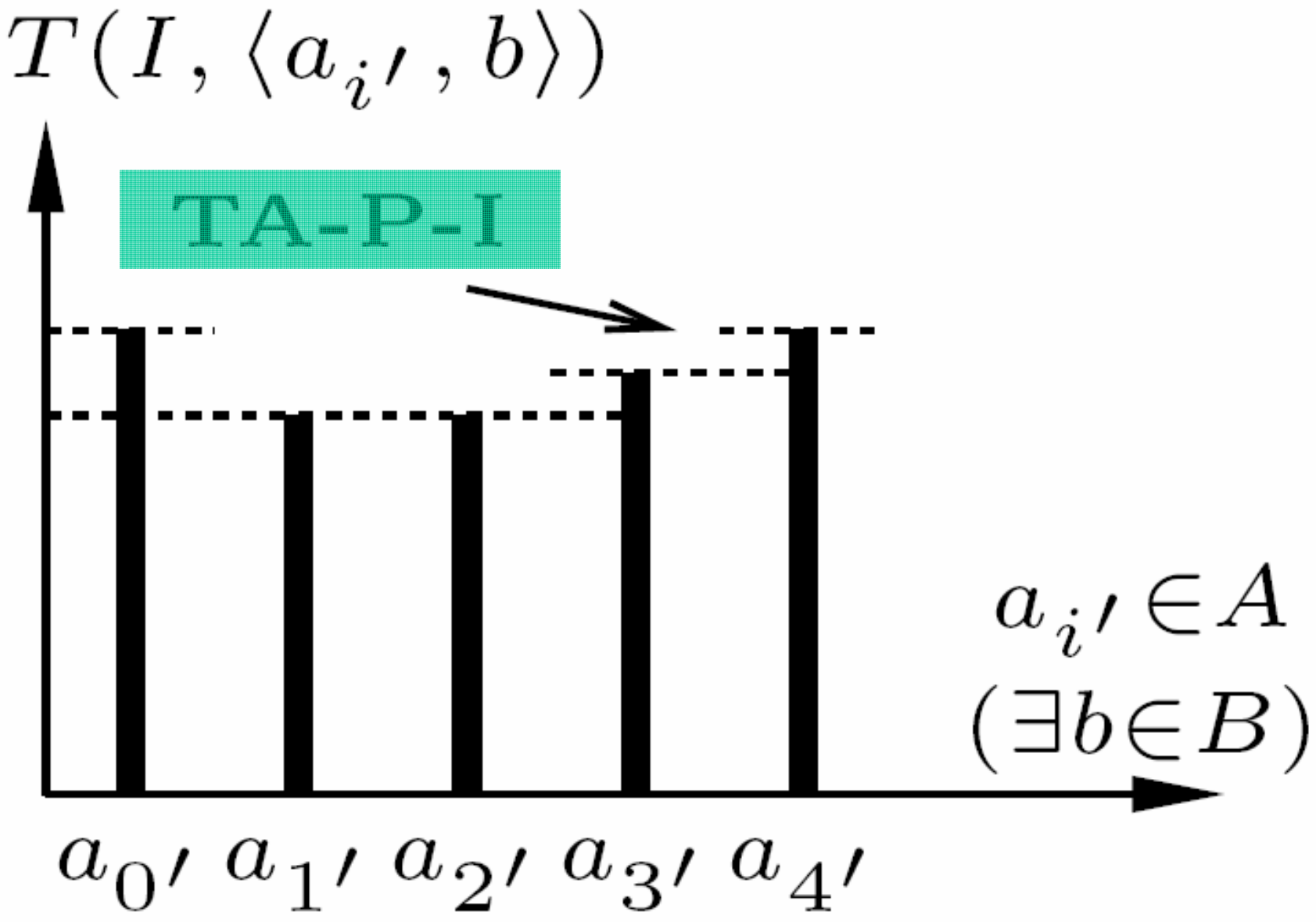




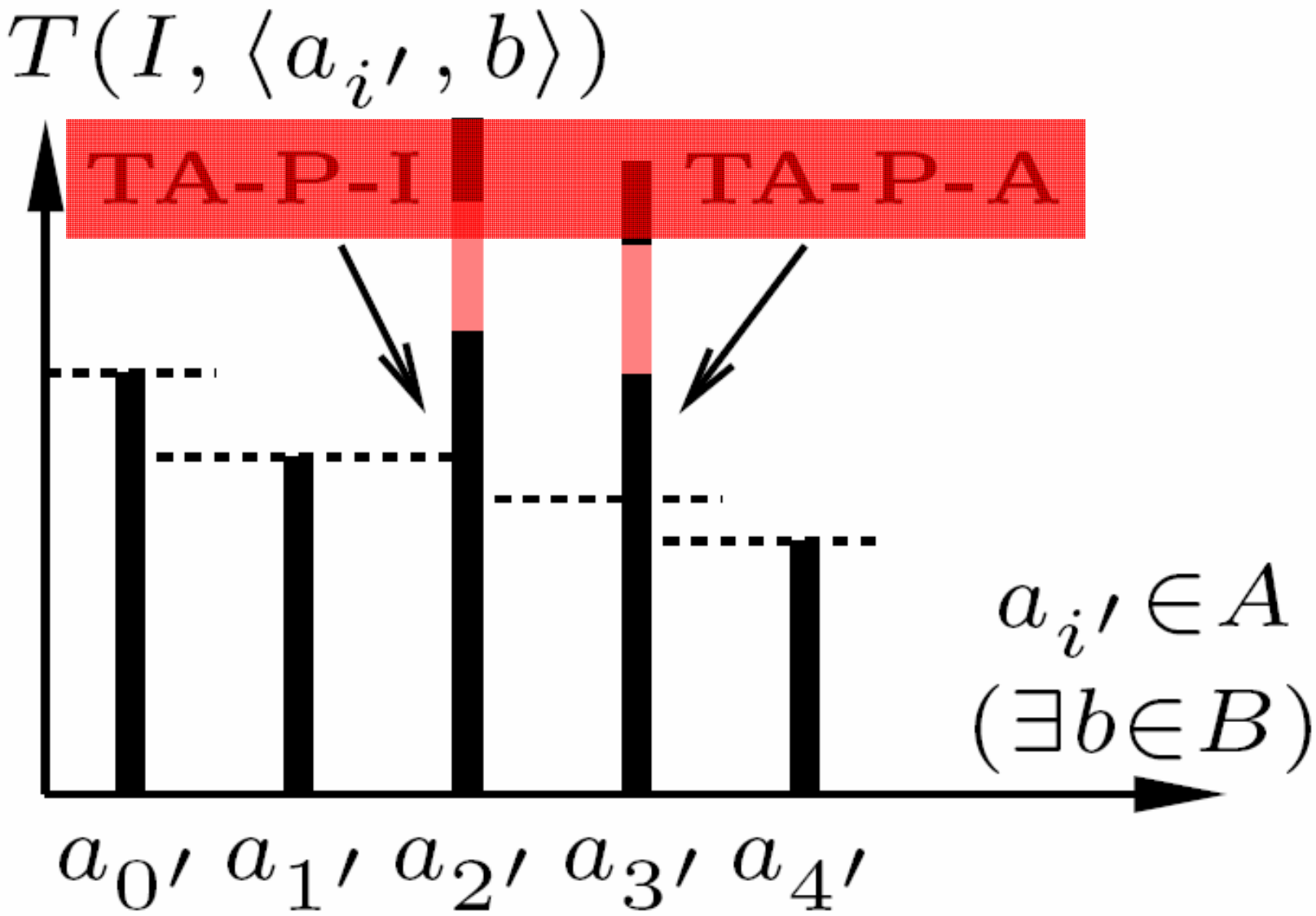
Pitfall: Parallel Timing Anomalies



Pitfall: Parallel Timing Anomalies

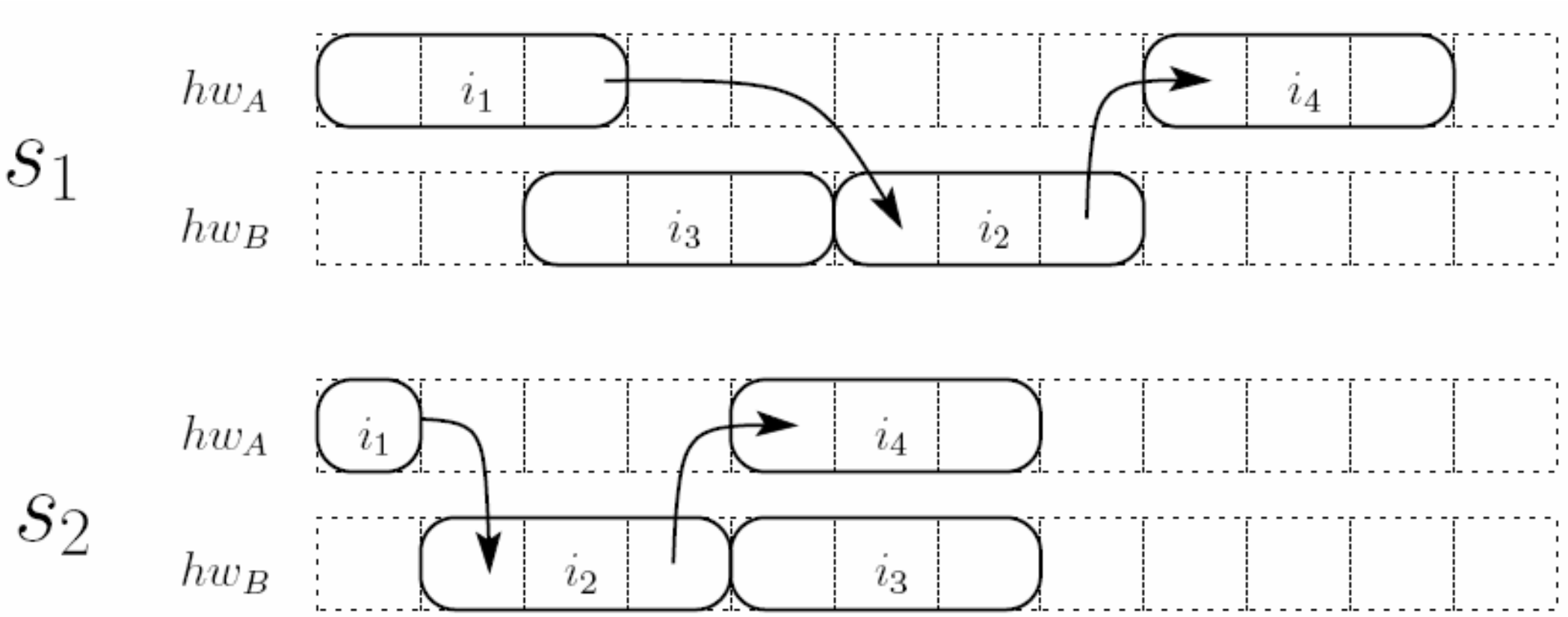


Pitfall: Parallel Timing Anomalies



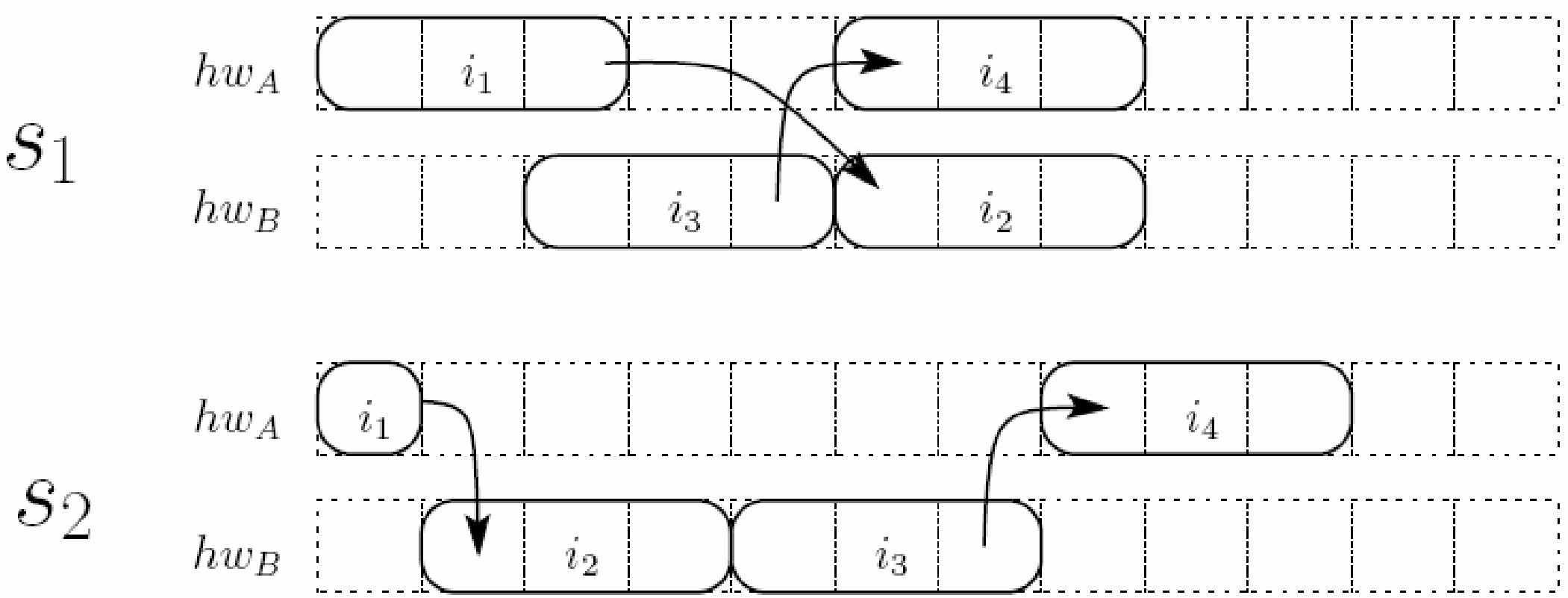
Example of TA-S-A and TA-P-A

out-of-order pipeline + cache + data dependencies:



Example of TA-S-I and TA-P-I

out-of-order pipeline + cache + data dependencies:



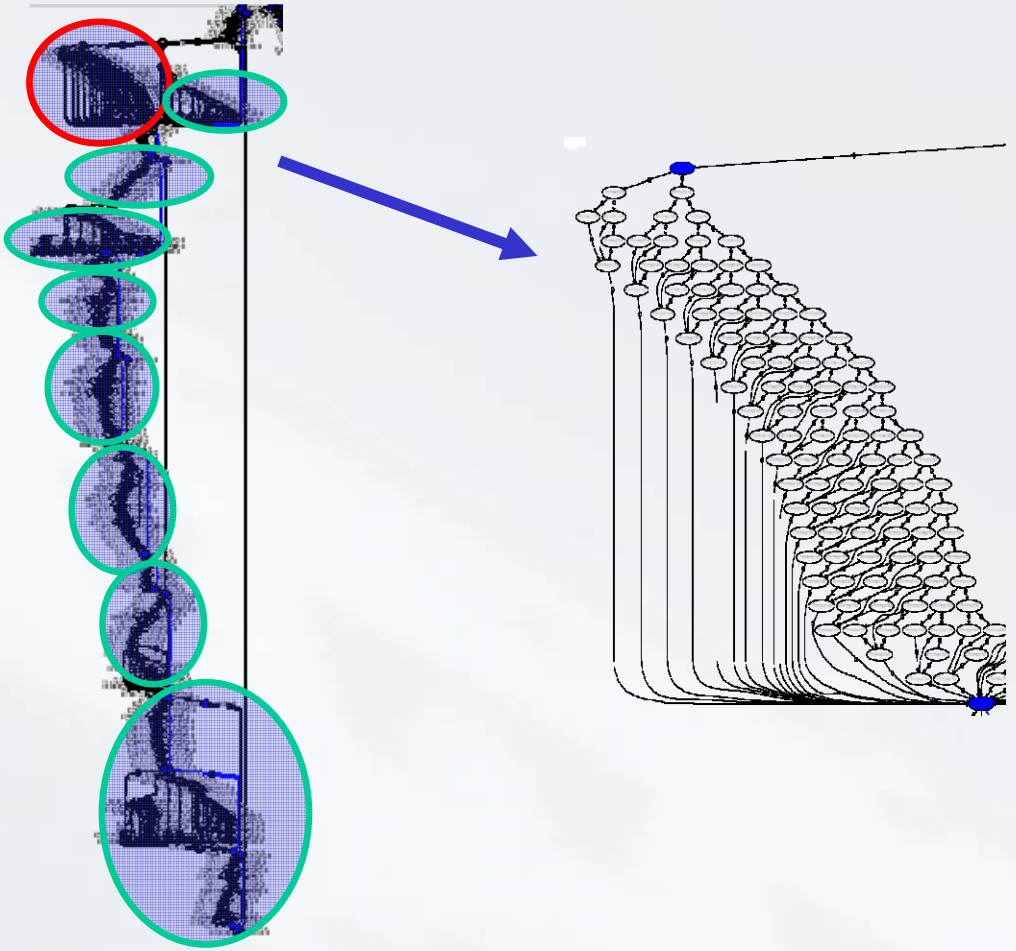
Our Contribution so far against the State Explosion Problem

- Providing a precise definition of Timing Anomalies (formal but without unnecessary details)
- Formalizing the different types of decomposition techniques for WCET analysis
- Proofs of which types of TAs are incompatible with which type of decomposition technique

[Kirner,TR-01-2009], [Kirner,ECRTS'09]

Measurement-Based Timing Analysis

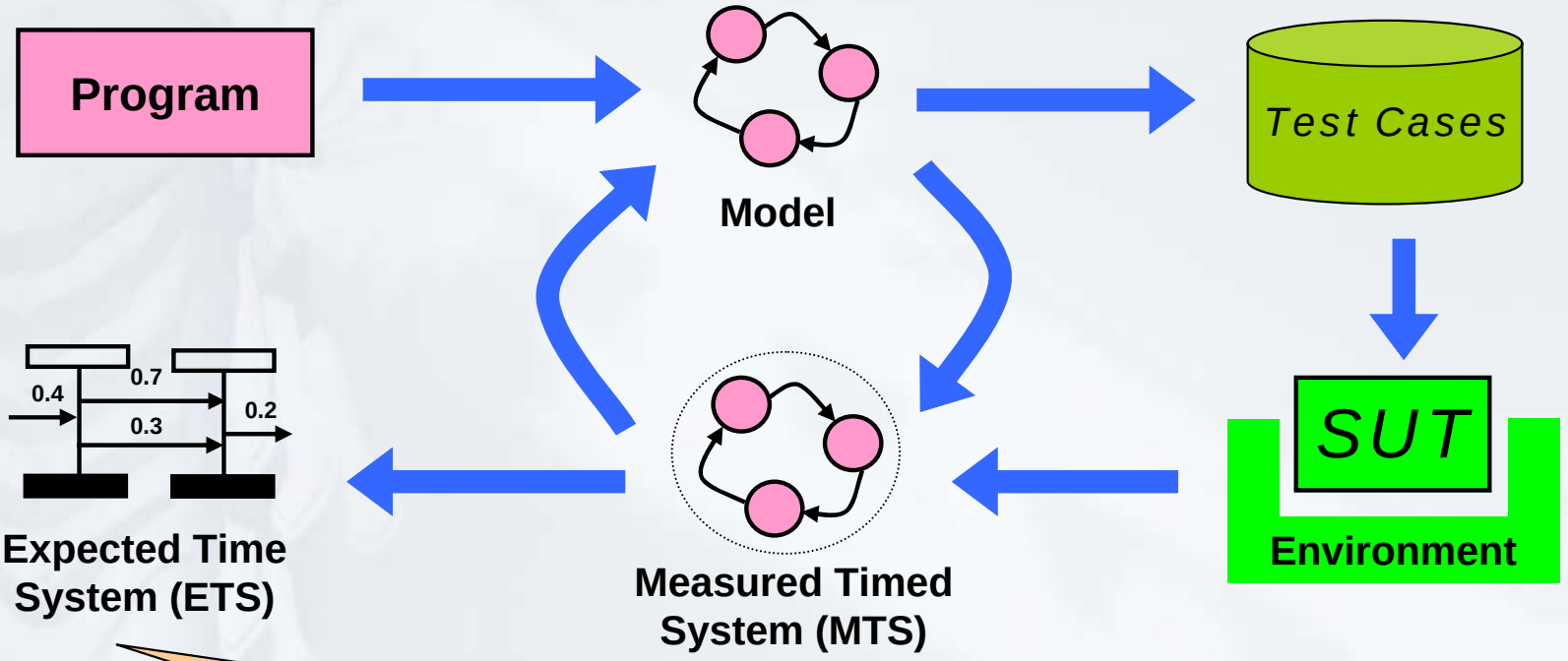
- Research project: ForTAS (Formal Timing Analysis Suite)



- Cooperation with TU Darmstadt
- Learning the Hardware Timing Model by systematic execution time measurements

The ForTAS Refinement Loop

- Testing for Timing Analysis



Richer timing information than just WCET: probabilistic timing model

The Next Steps: Getting Out of the Complexity Mess

- Take a pro-active approach to advance timing analysis: construction of embedded systems that support:
  - Predictability (with reasonable margins)
  - Composability
  - Scalability
- Predictable access to shared resources
- Timing-predictability requires a HW/SW co-design ("patterns of predictability").

Thank You!

http://costa.tuwien.ac.at

http://www.fortastic.net