

Provably Timing-Predictable Microarchitectures

Jan Reineke

joint work with Sebastian Hahn and Johannes Kahlen



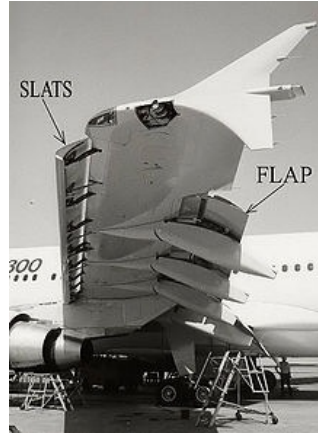
UNIVERSITÄT
DES
SAARLANDES

SIC Saarland Informatics
Campus

Context: Hard Real-Time Systems



*Airbag
Reaction in < 10 ms*

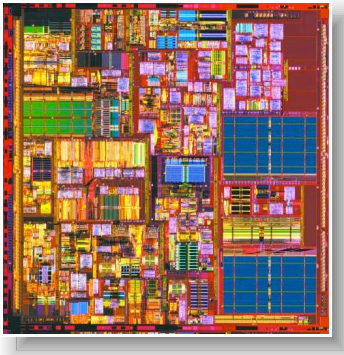


- Embedded software must
- deliver **correct** control signals,
 - within **fixed time bounds**.

Timing Analysis Problem

```
// Perform the convolution.  
for (int i=0; i<10; i++) {  
    x[i] = a[i]*b[j-i];  
    // Notify listeners.  
    notify(x[i]);  
}
```

Embedded Software



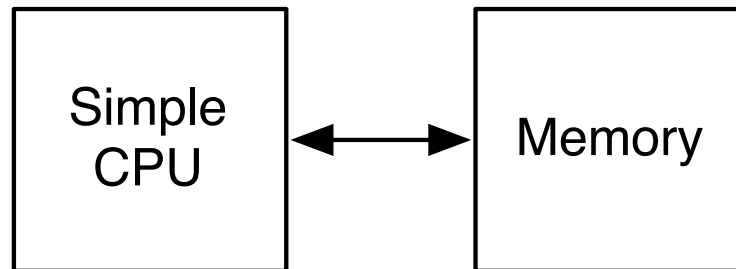
Microarchitecture



Timing Requirements

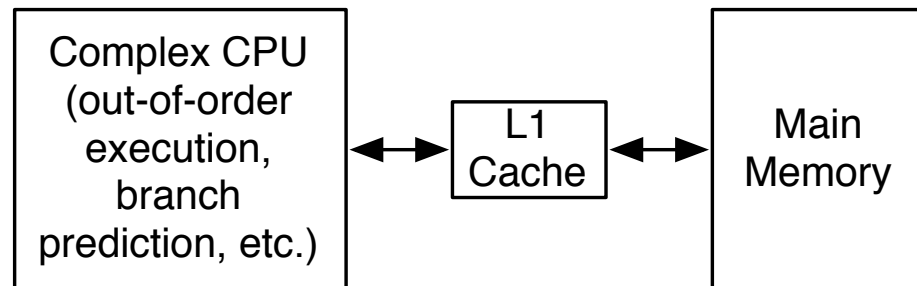
What does the execution time depend on?

- The **input**, determining which path is taken through the program.

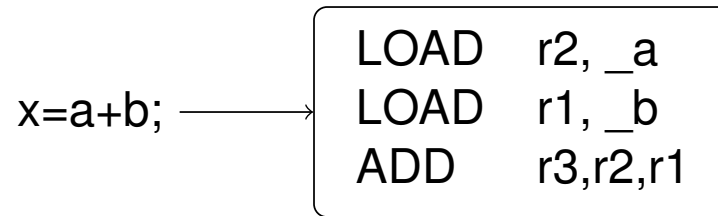


What does the execution time depend on?

- The input, determining which path is taken through the program.
- The **state of the hardware platform**:
 - Due to caches, pipelining, speculation, etc.



Influence of Microarchitectural State



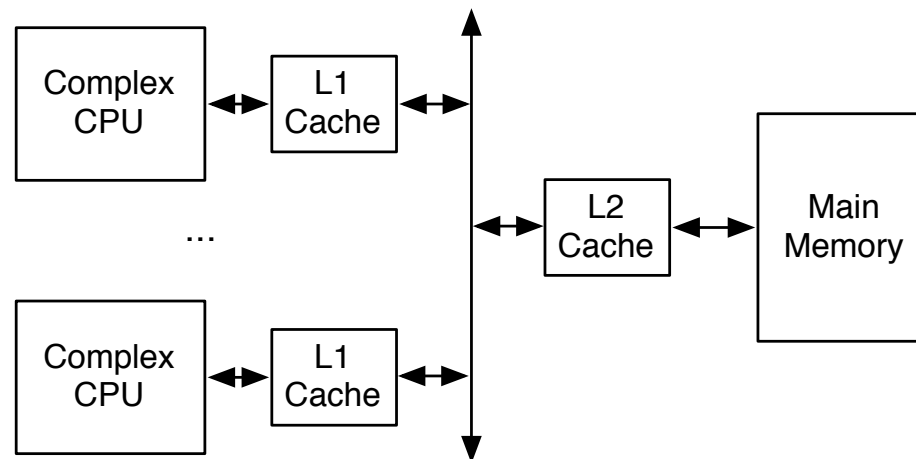
PowerPC 755

Execution Time (Clock Cycles)



What does the execution time depend on?

- The input, determining which path is taken through the program.
- The state of the hardware platform:
 - Due to caches, pipelining, speculation, etc.
- **Interference** from the environment:
 - External interference as seen from the analyzed task on shared busses, caches, memory.

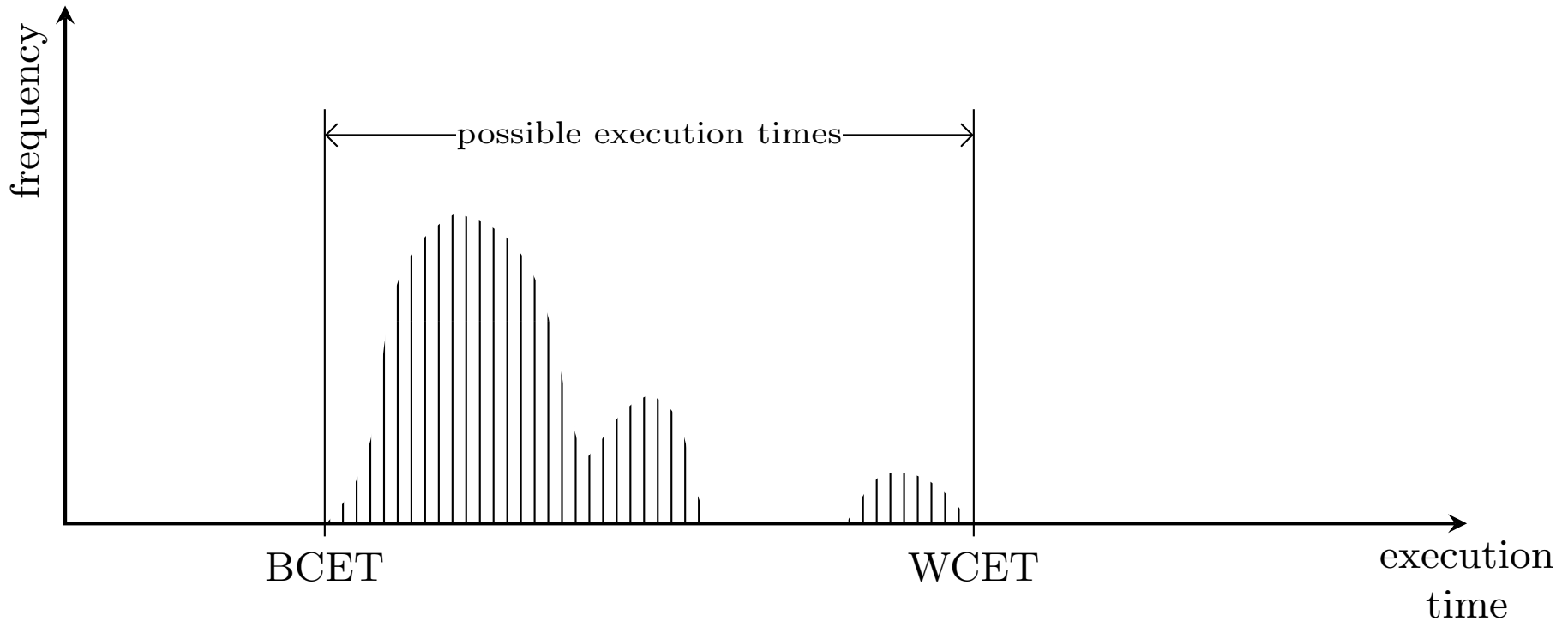


Influence of Corunning Tasks in Multicores

Radojkovic et al. (ACM TACO, 2012) on
Intel Atom and Intel Core 2 Quad:

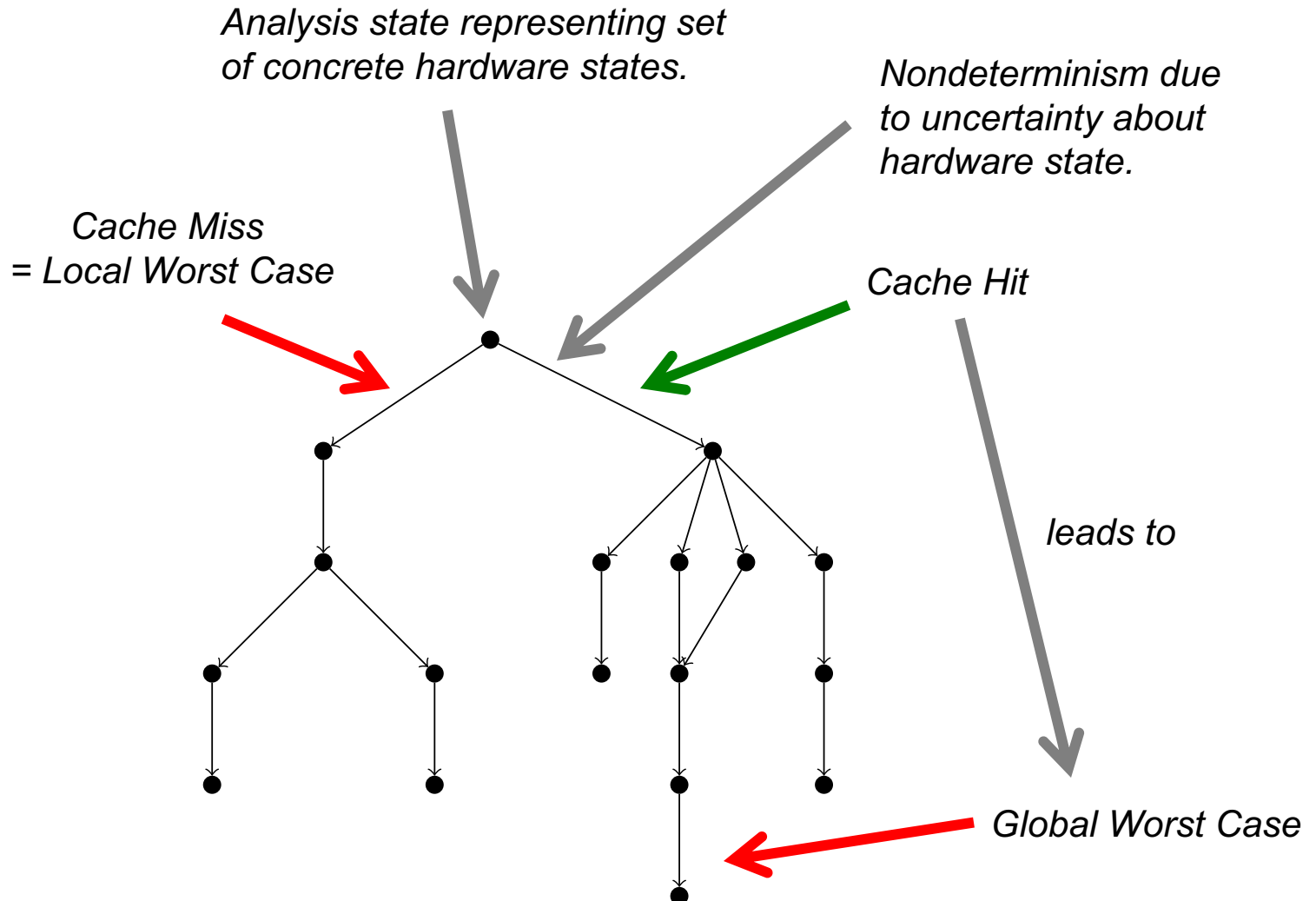
up to **14x slow-down** due to interference
on **shared L2 cache** and **memory controller**

Two Schools of Thought



1. Predictable = $\frac{WCET}{BCET}$ is small \approx deterministic timing
2. Predictable = WCET can be efficiently approximated

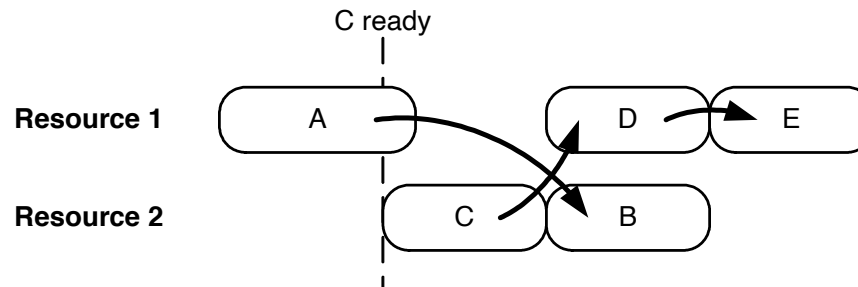
Timing Anomalies



Timing Anomalies in Dynamically Scheduled Microprocessors
T. Lundqvist, P. Stenström – RTSS 1999

Timing Anomalies: Example

Scheduling Anomaly

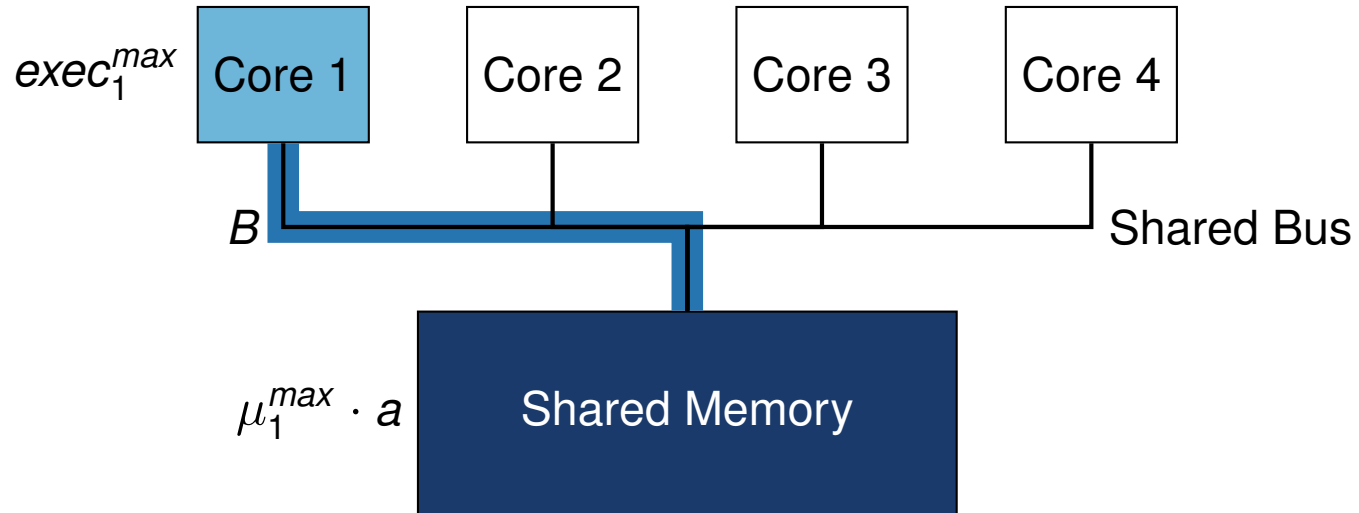


Bounds on multiprocessing timing anomalies

RL Graham - SIAM Journal on Applied Mathematics, 1969 – SIAM

(<http://epubs.siam.org/doi/abs/10.1137/0117039>)

Timing Compositionality: By Example

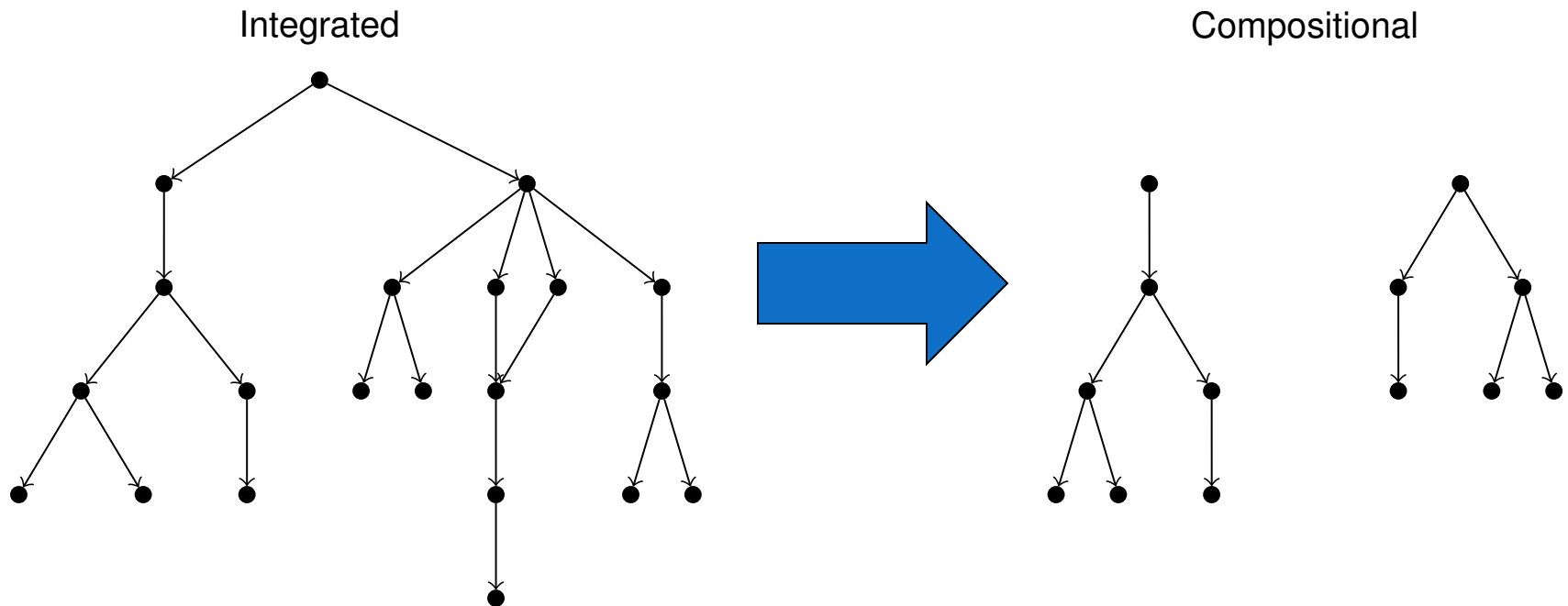


Timing Compositionality =

Ability to simply sum up timing contributions by different components

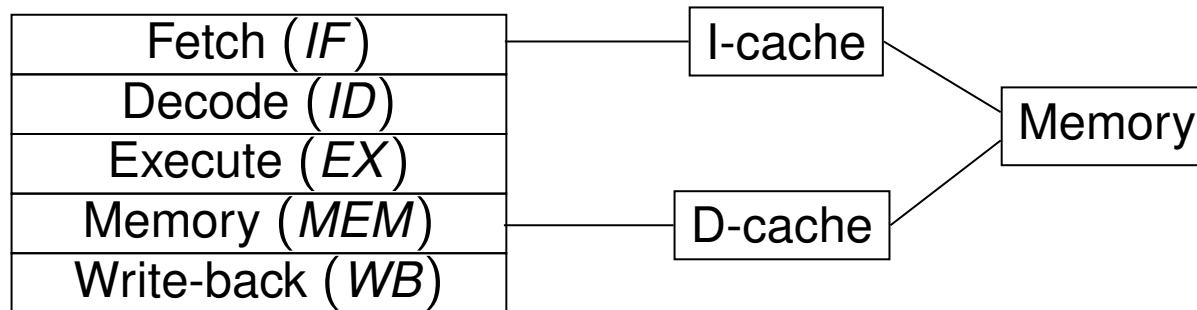
Implicitly or explicitly assumed by (almost) all approaches to timing analysis for multi cores and cache-related preemption delays (CRPD).

Timing Compositionality: Benefit

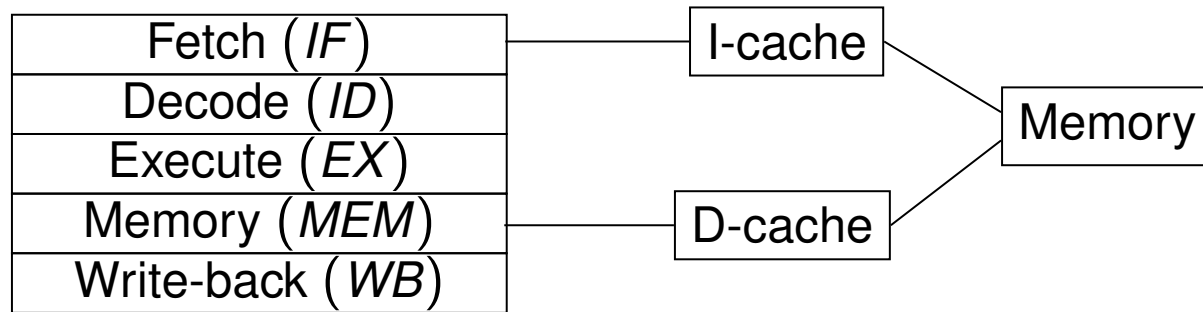


Achieving Timing Compositionality

Textbook in-order pipeline + LRU caches



Bad News I: Timing Anomalies



We show such a pipeline has timing anomalies:

Toward Compact Abstractions for Processor Pipelines

S. Hahn, J. Reineke, and R. Wilhelm. In Correct System Design, 2015.

Bad News II: Timing Compositionality

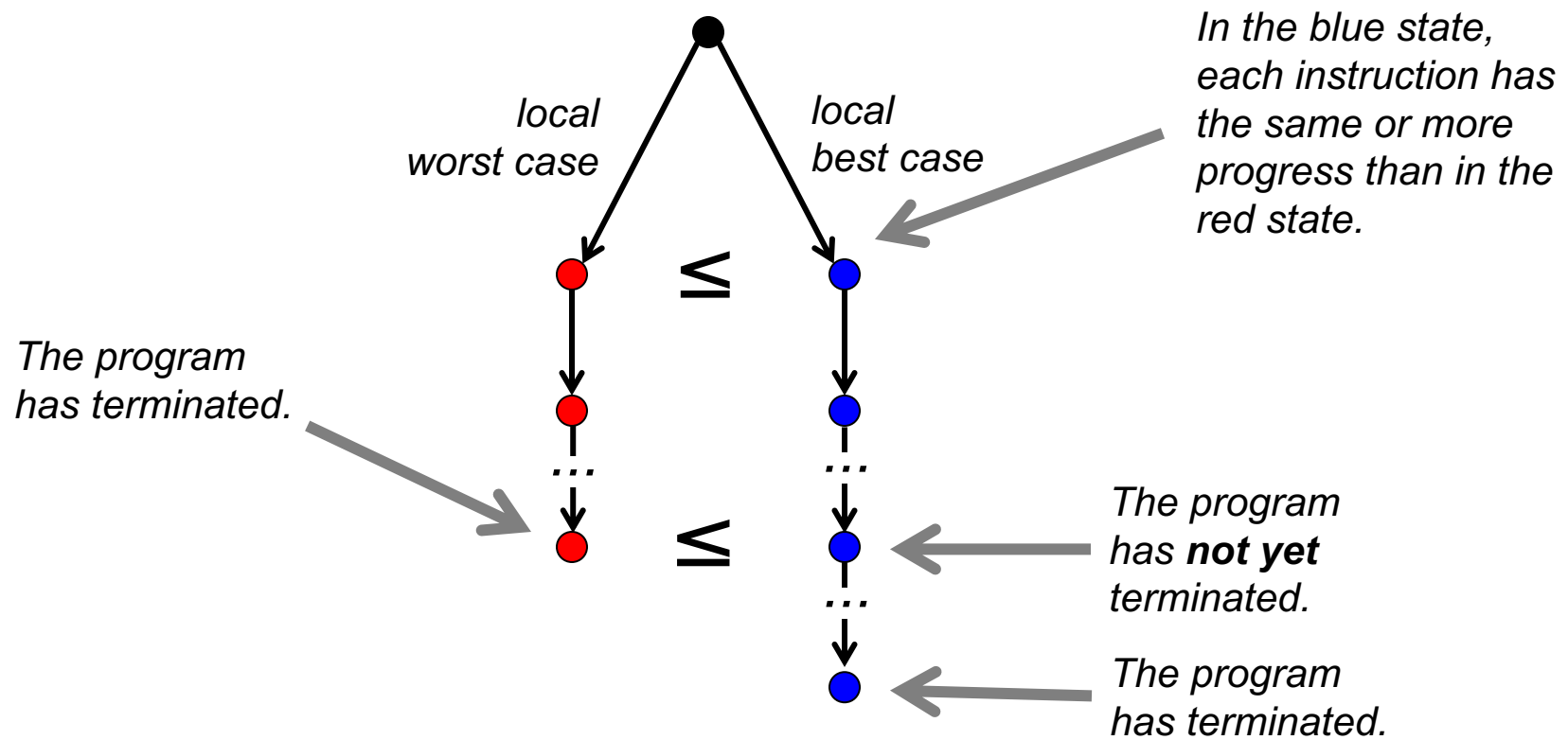
Maximal cost of an additional cache miss?

Intuitively: main memory latency

Unfortunately: ~ 2 times main-memory latency

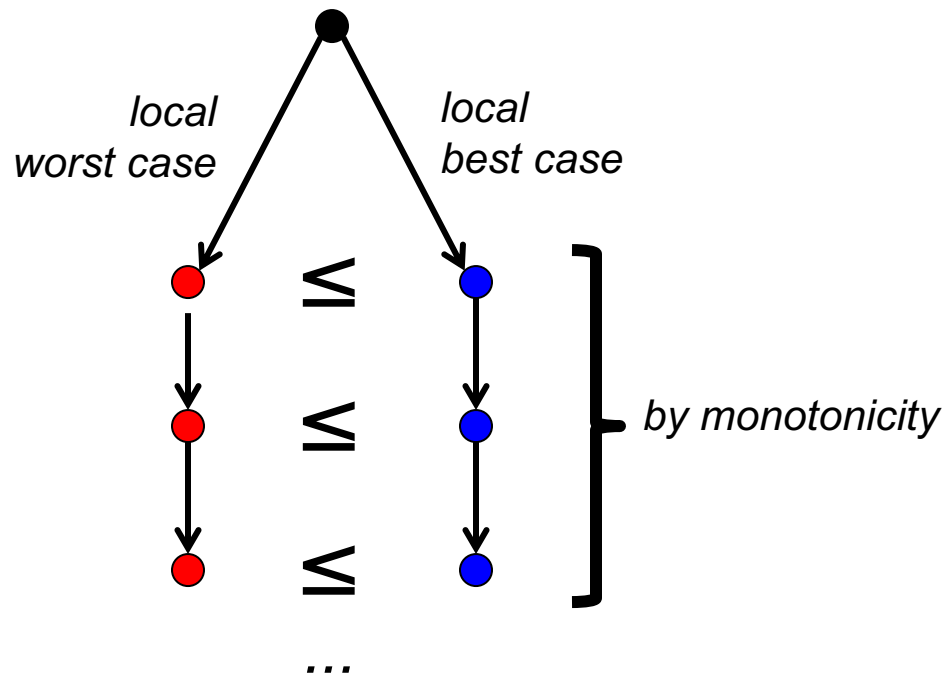
- ongoing instruction fetch may block load
- ongoing load may block instruction fetch

Key Insight: Anomalies Require Non-Monotonicity



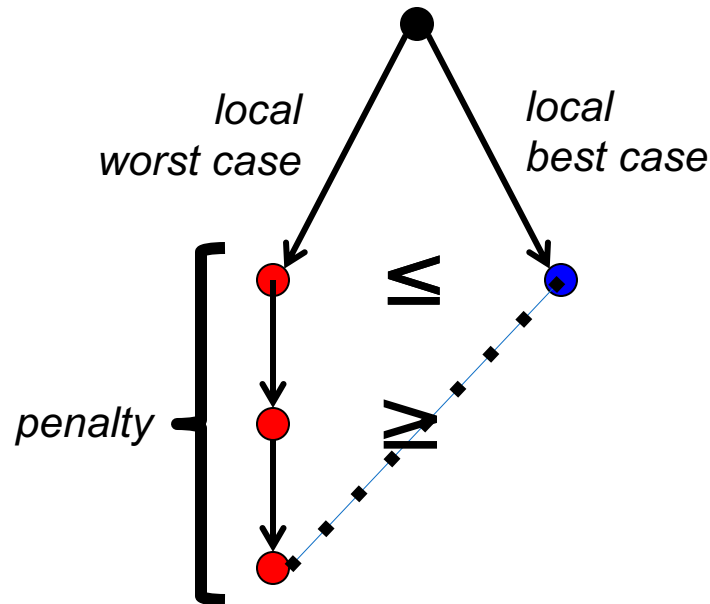
Monotonicity Enables Predictability 1/2

Theorem 1 (Timing Anomalies):
Monotonicity implies absence of timing anomalies.

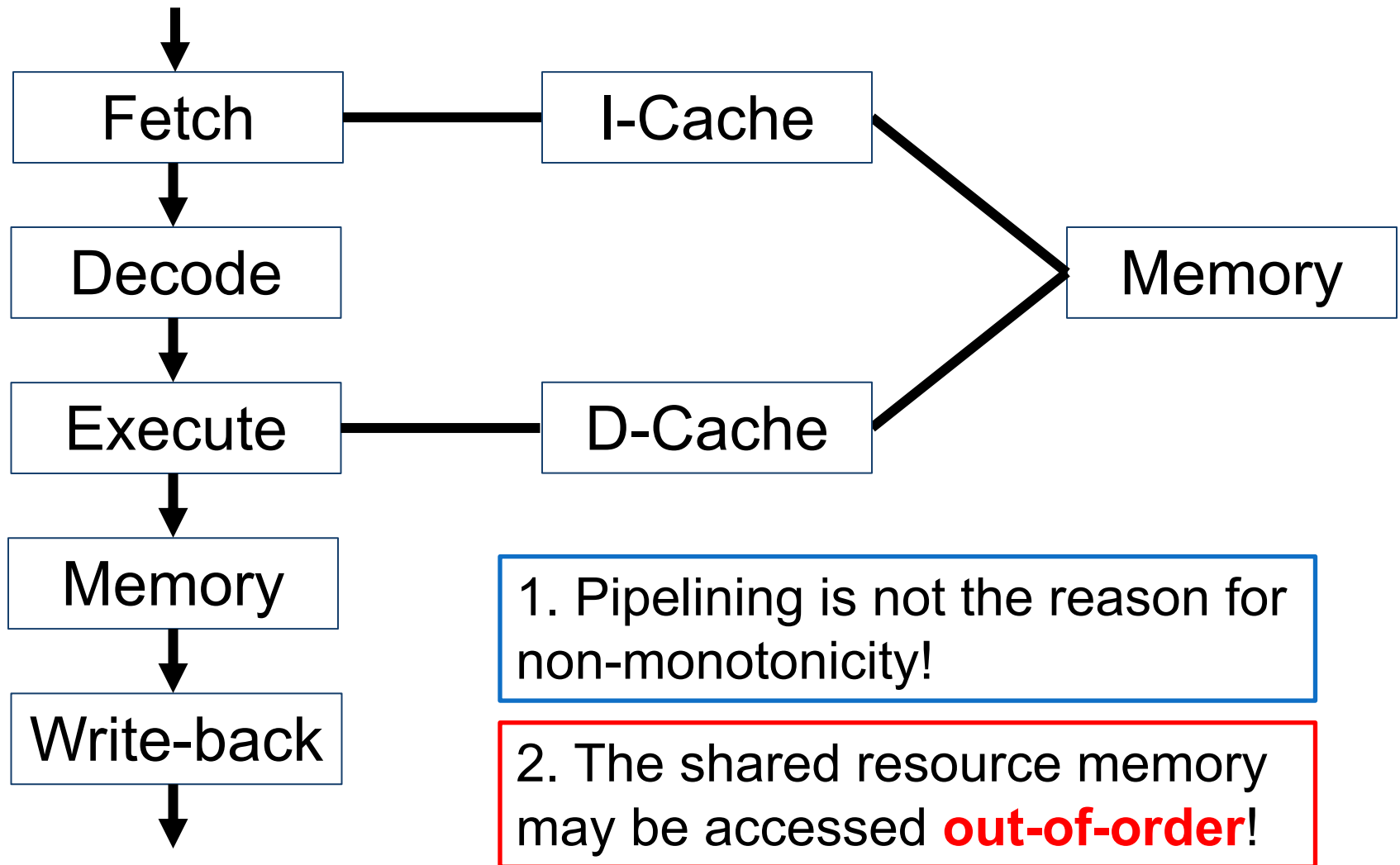


Monotonicity Enables Predictability 1/2

Theorem 2 (Timing Compositionality):
Monotonicity enables the derivation of sound penalties.



How to Achieve Monotonicity?



Strictly In-Order Pipelines: Definition

Definition (Strictly In-Order):

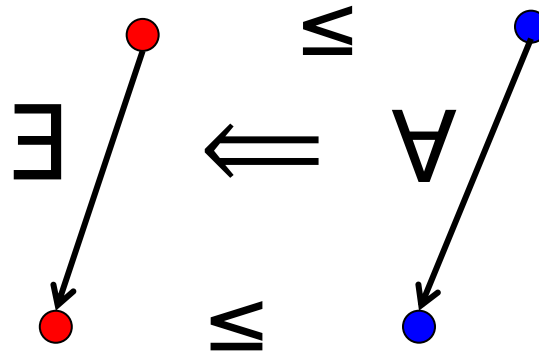
We call a pipeline *strictly in-order* if each *resource* processes the instructions in program order.

- Enforce memory operations (instructions and data) in-order (common memory as resource)
- Block instruction fetch until no potential data accesses in the pipeline

Strictly In-Order Pipelines: Properties

Theorem 1 (Monotonicity):

In the strictly in-order pipeline progress of an instruction is monotone in the progress of other instructions.



Corollary (Timing Anomalies and Timing Compositionality):

In the strictly in-order pipeline

- does not have timing anomalies, *and*
- admits compositional analysis with natural penalties.

Experimental Evaluation

Performance:

Strictly in-order pipeline is about 6% slower than regular in-order pipeline.

→ Preserves most of the benefits of pipelining.

Predictability:

~4x faster single-core analysis

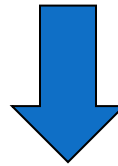
~32x faster multi-core analysis

Automating the Predictability Proofs

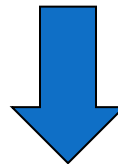
Formalization
of Processor

+

Formalization
of Property



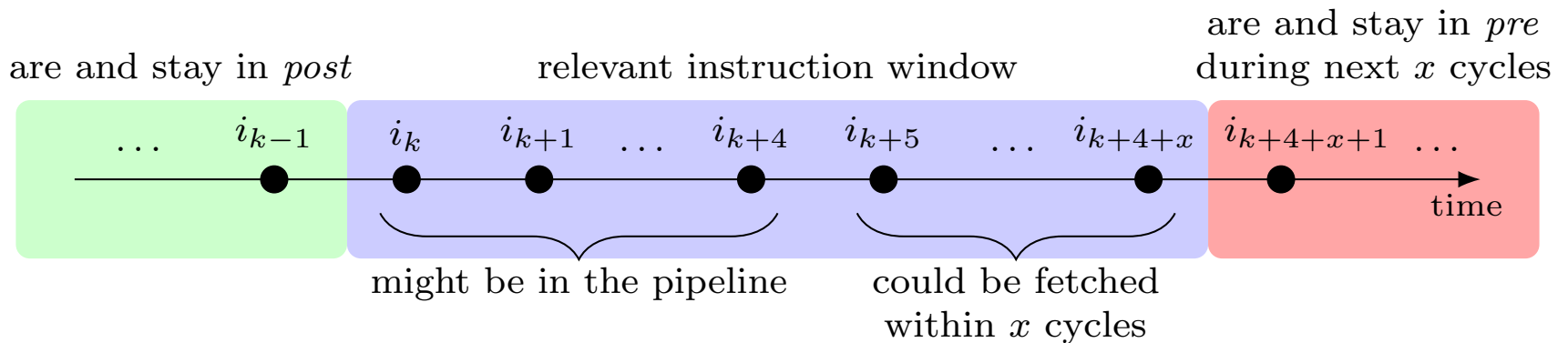
SMT Solver



Challenges in Proof Automation

Processor states map dynamic instruction instances to their progress → **infinite state space**

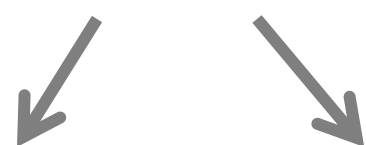
Suffices to consider a finite window:



Proof of Monotonicity: First Attempt

Define formula that is **unsatisfiable**,
if *transition relation* is **monotonic**.

*Captures
transition
relation*



$$p_1 \sqsubseteq p_2 \wedge \mathbf{cycle}(p_1, p'_1) \wedge \mathbf{cycle}(p_2, p'_2) \wedge \neg p'_1 \sqsubseteq p'_2$$

However, the formula is **satisfiable**!

Need to capture reachable states...

Proof of Monotonicity: Second Attempt

Define formula that is **unsatisfiable**,
if *transition relation* is **monotonic**.

$\text{validPipelineState}(p_1) \wedge \text{validPipelineState}(p_2) \wedge$
 $p_1 \sqsubseteq p_2 \wedge \text{cycle}(p_1, p'_1) \wedge \text{cycle}(p_2, p'_2) \wedge \neg p'_1 \sqsubseteq p'_2$



*proved correct
via separate
SMT queries*

This formula is indeed **unsatisfiable**!

Anomaly Freedom and Timing Compositionality

See paper:

Sebastian Hahn, Jan Reineke:

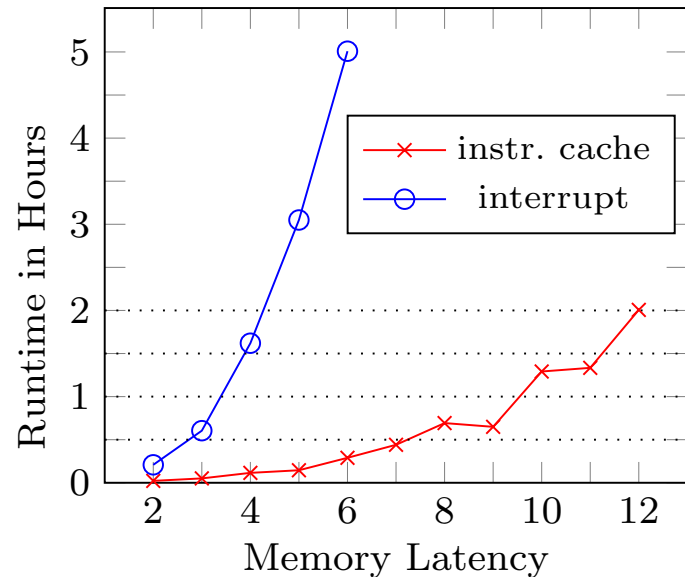
Design and analysis of SIC: a provably timing-predictable pipelined processor core.

Real-Time Systems, November 2019

Efficiency of SMT Proofs

Proof	Z3 Runtime
monotonicity of SIC	7s
non-monotonicity of textbook in-order	< 1s
anomaly-freedom w.r.t. cache	< 1s

*Compositionality w.r.t.
instruction-cache and
interrupts:*



Conclusions and Future Work

Key Insight:

Monotonicity enables Timing Predictability

Strictly in-order pipeline is monotonic

Predictability proofs can be automated

- Translation of model to SMT still manual
- Need to capture relevant invariants manually
- Can we automate the process further?

References

Sebastian Hahn, Jan Reineke:

Design and analysis of SIC: a provably timing-predictable pipelined processor core.

RTSS 2018 (best student paper award)

Sebastian Hahn, Jan Reineke:

Design and analysis of SIC: a provably timing-predictable pipelined processor core.

Real-Time Systems, November 2019