Timing Anomalies and Timing Compositionality

Jan Reineke @ SAARLAND UNIVERSITY
COMPUTER SCIENCE

TACLe: Timing Analysis on Code Level

Prague, January 20, 2016

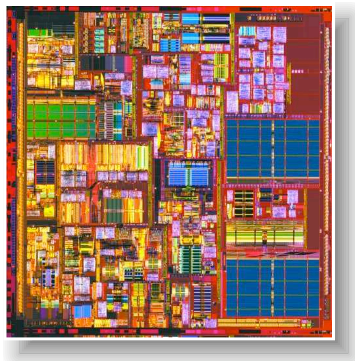
Outline

The 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

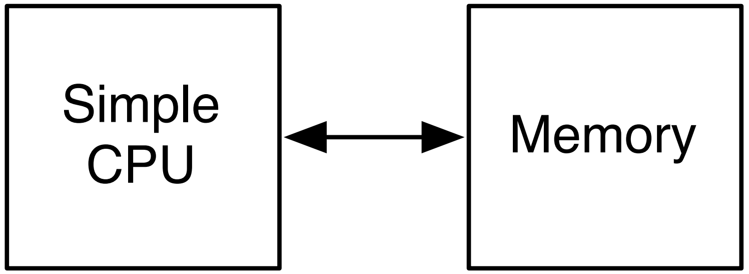
?

$\Rightarrow$

Timing Requirements

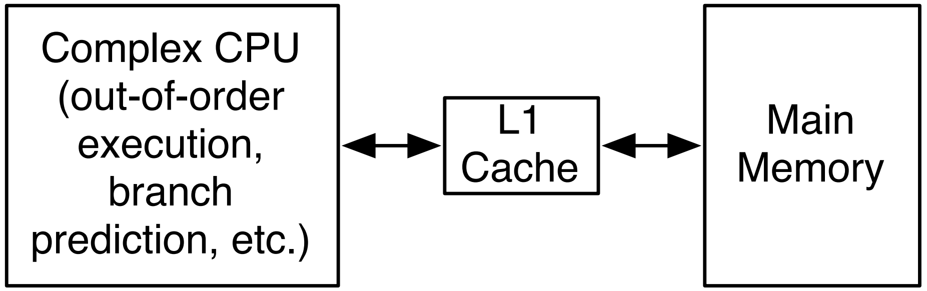
What does the execution time depend on?

- 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.



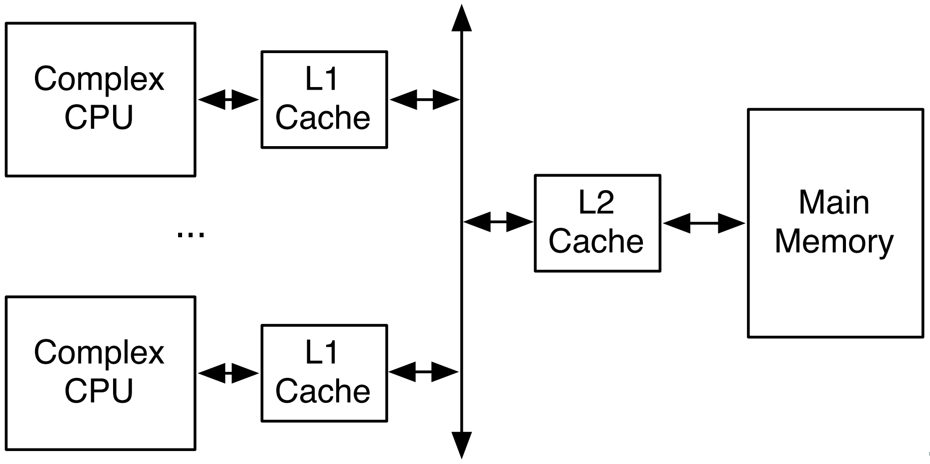
What does the execution time depend on?

- 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.

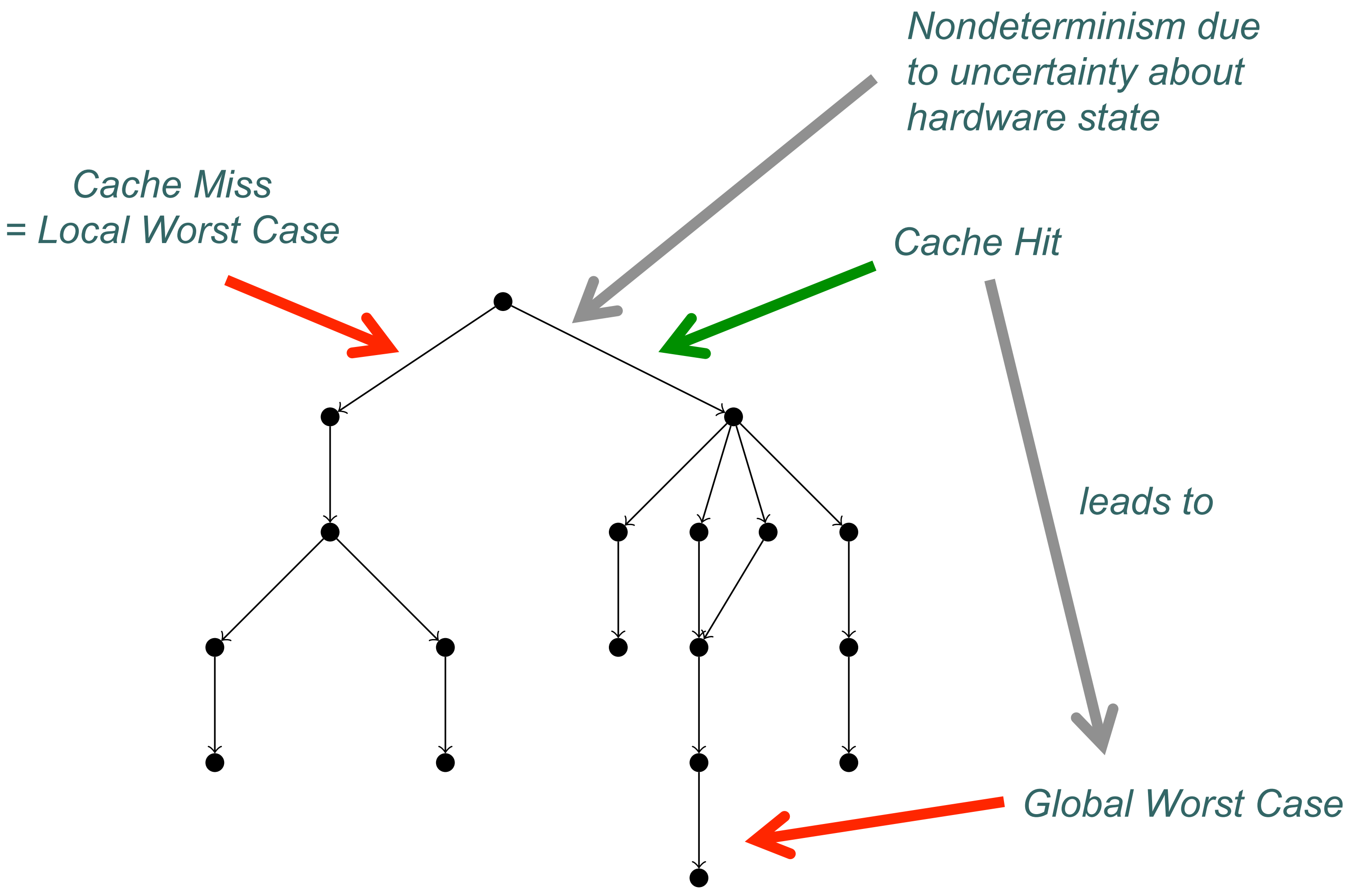

Complex CPU (out-of-order execution, branch prediction, etc.)
L1 Cache
Main Memory

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.



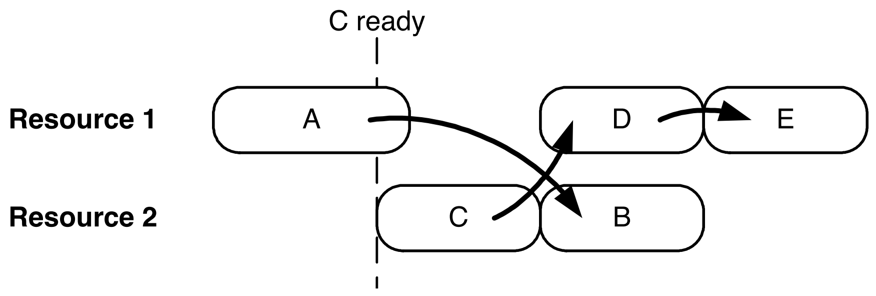
Timing Anomalies


Nondeterminism due to uncertainty about hardware state
Cache Miss = Local Worst Case
Cache Hit
leads to
Global Worst Case

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

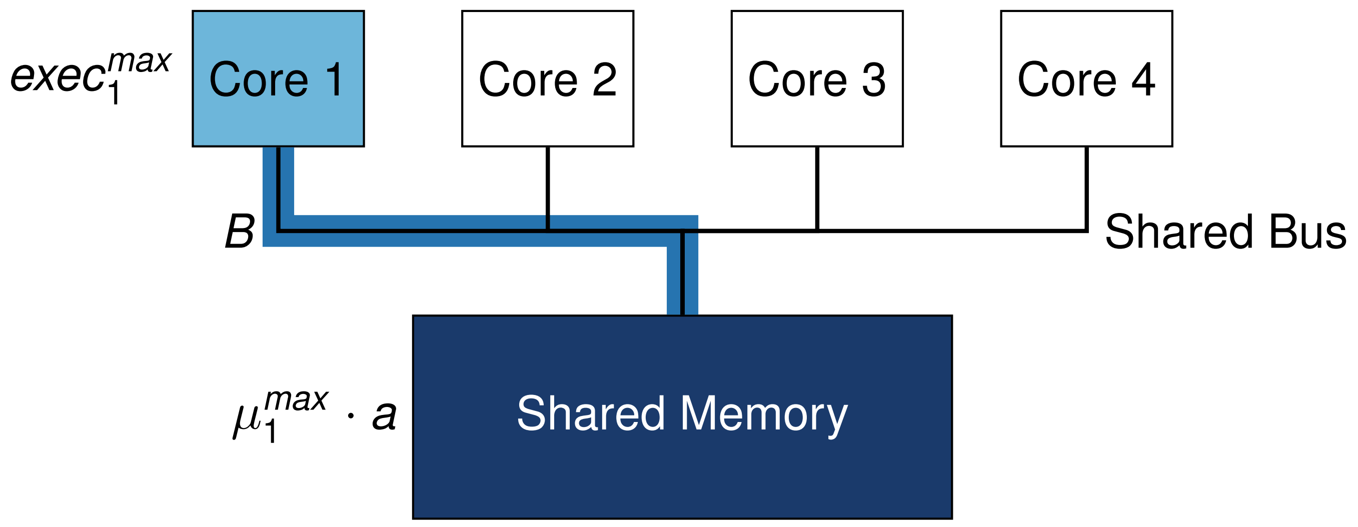
Timing Anomalies: Example

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).

Conventional Wisdom

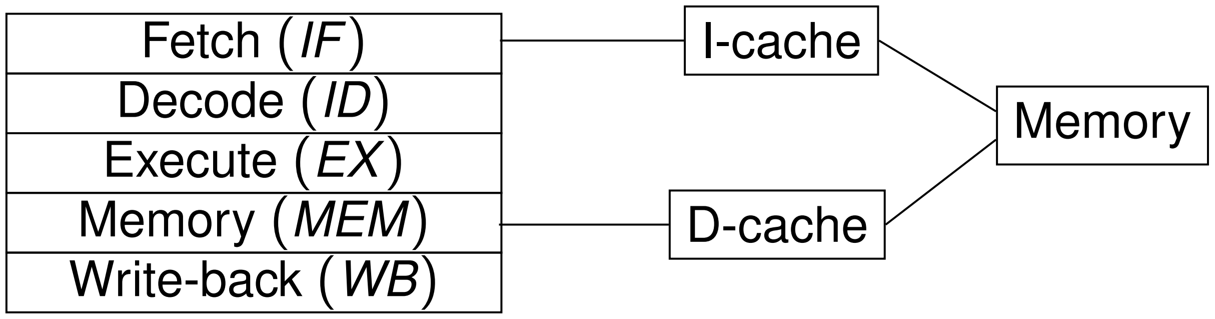
Simple in-order pipeline + LRU caches

→ no timing anomalies

→ timing-compositional

$\boxed{\text{False!}}$

Bad News: In-order Pipelines



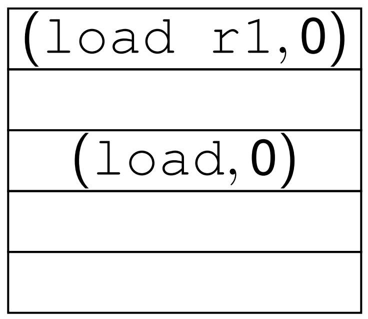
We show such a pipeline has timing anomalies:

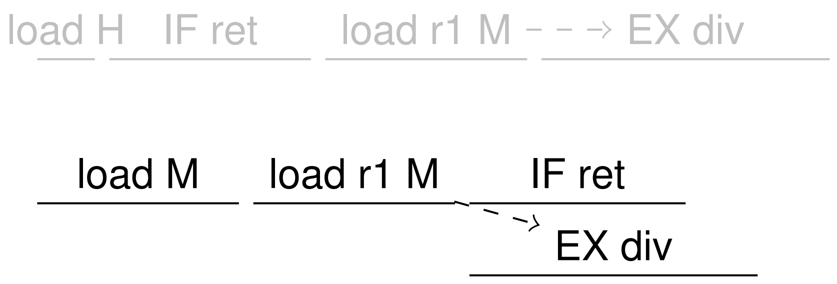
Toward Compact Abstractions for Processor Pipelines

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

A Timing Anomaly

```
load ...
nop
load r1, ...
div ..., r1
------------
ret
```





Hit case:
- Instruction fetch starts before second load becomes ready
- S

Intuitive Reason:

Progress in the pipeline influences order of instruction fetch and data access

Mis

- S
- Second load is prioritized over instruction fetch
- Loading before fetching suits subsequent execution

Timing Compositionality of In-order Pipeline

Maximal cost of an additional cache miss?

Intuitively: cache miss penalty

Unfortunately: ~ 2 times cache miss penalty
- ongoing instruction fetch may block load
- ongoing load may block instruction fetch

Good News

Two approaches to solve problem:

1. Stall entire processor upon „timing accidents“
2. Strictly in-order pipeline

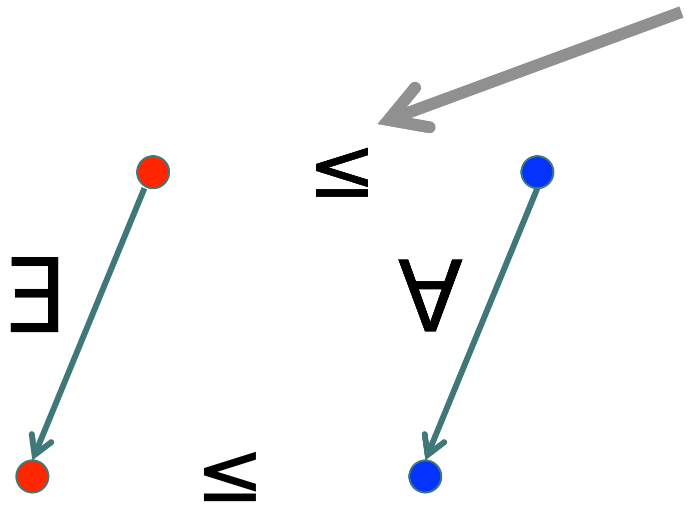
Strictly In-Order Pipelines: Definition

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

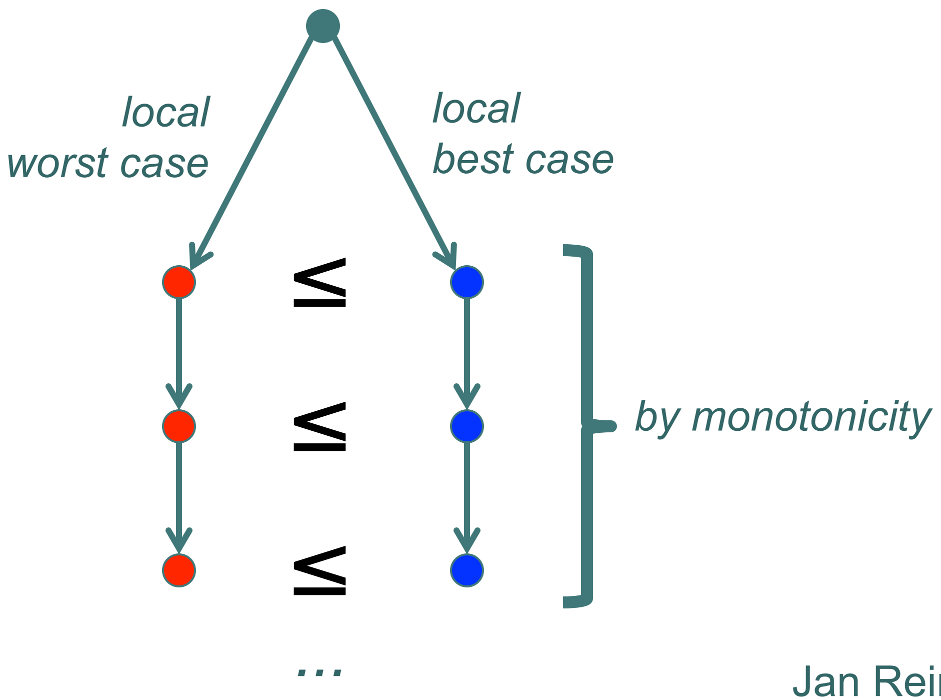
Theorem 1 (Monotonicity):
In the strictly in-order pipeline progress of an instruction is monotone in the progress of other instructions.



In the blue state, each instruction has the same or more progress than in the red state.

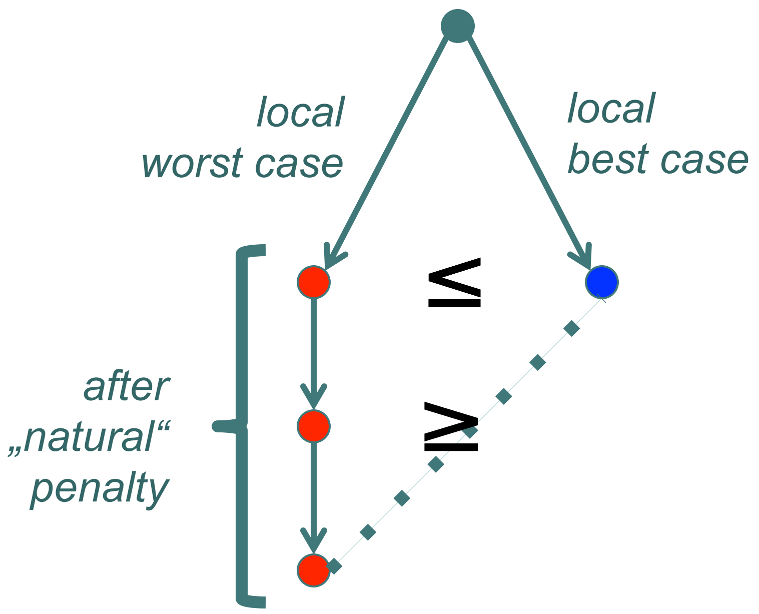
Strictly In-Order Pipelines: Properties

Theorem 2 (Timing Anomalies):
The strictly in-order pipeline is free of timing anomalies.



Strictly In-Order Pipelines: Properties

Theorem 3 (Timing Compositionality):
The strictly in-order pipeline admits „compositional analysis with intuitive penalties.“



Conclusions

Timing compositionality enabler for

- CRPD-aware response-time, and
- multi-core timing analysis

Not provided even by simple, in-order pipelines.

Strictly in-order pipeline is free of timing anomalies and provides timing compositionality.

Future Work

Evaluate impact on analysis efficiency, average-case performance, and predictable performance.

Extensions:

- Can we extend the approach to more complex, e.g. even out-of-order pipelines?

Thank you for your attention!

References

Towards Compositionality in Execution Time Analysis - Definition and Challenges

S. Hahn, J. Reineke, and R. Wilhelm. In CRTS, 2013.

Toward Compact Abstractions for Processor Pipelines

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