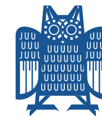
Challenges for Timing Analysis of Multi-Core Architectures

Jan Reineke @ SAARLAND UNIVERSITY



COMPUTER SCIENCE

DICE-FOPARA, *Uppsala, Sweden*

April 22, 2017

The Context: Hard Real-Time Systems

Safety-critical applications:

- Avionics, automotive, train industries, manufacturing

Side airbag in car
Reaction in < 10 msec

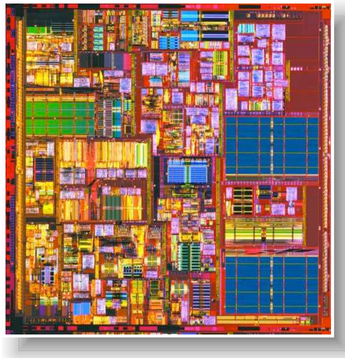
Crankshaft-synchronous tasks
Reaction in < 45 microsec

- Embedded software must
  - compute correct control signals,
  - within time bounds.

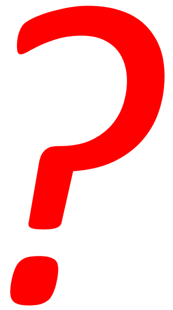
The Timing Analysis Problem

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

Set of Software Tasks



Microarchitecture



Timing Requirements

"Standard Approach" for Timing Analysis

Two-phase approach:

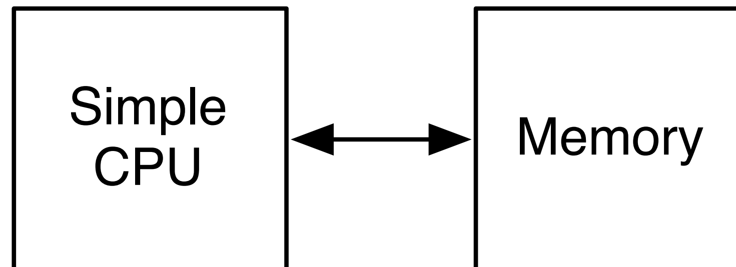
1. Determine WCET (worst-case execution time) bounds for each task on microarchitecture
2. Perform response-time analysis

Simple interface between WCET analysis and response-time analysis: WCET bounds

What does the execution time depend on?

SAARLAND UNIVERSITY
COMPUTER SCIENCE

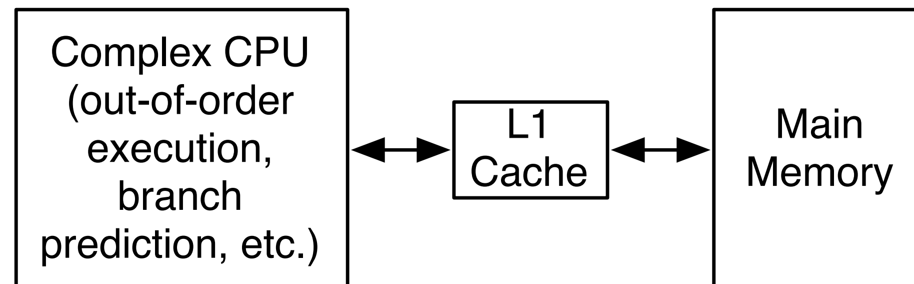
- The input, determining which path is taken through the program.



What does the execution time depend on?

SAARLAND UNIVERSITY
COMPUTER SCIENCE

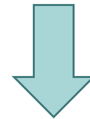
- The input, determining which path is taken through the program.
- The state of the hardware platform:
  - Due to caches, pipelining, speculation, etc.



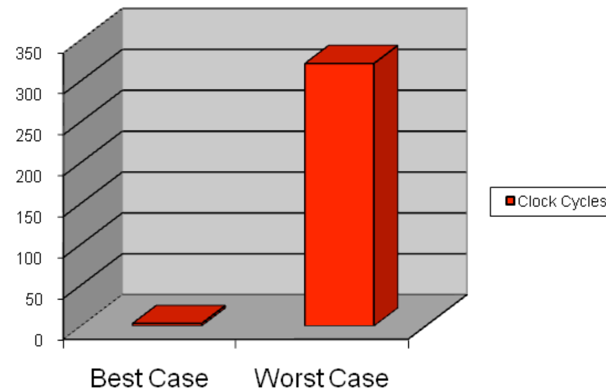
Example of Influence of Microarchitectural State

x=a+b; →

```
LOAD   r2, _a
LOAD   r1, _b
ADD    r3,r2,r1
```



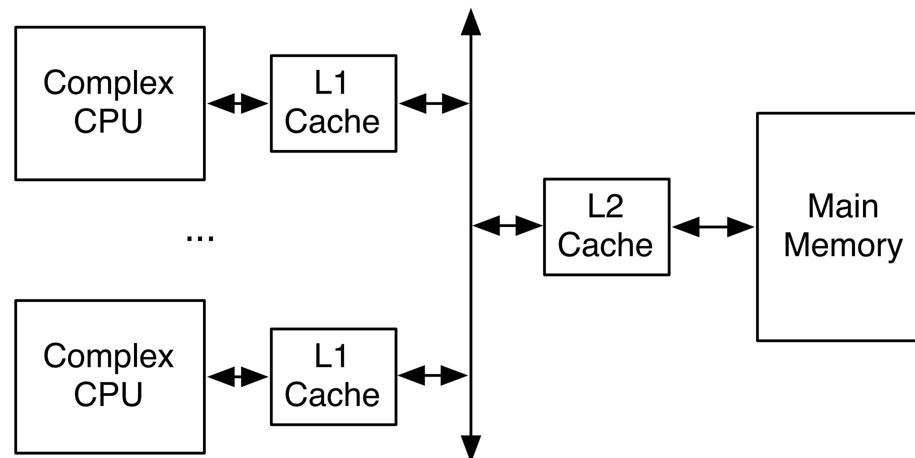
PowerPC 755



What does the execution time depend on?

SAARLAND UNIVERSITY
COMPUTER SCIENCE
SAARLAND UNIVERSITY
COMPUTER SCIENCE

- 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.



Example of Influence of Corunning Tasks in Multicores

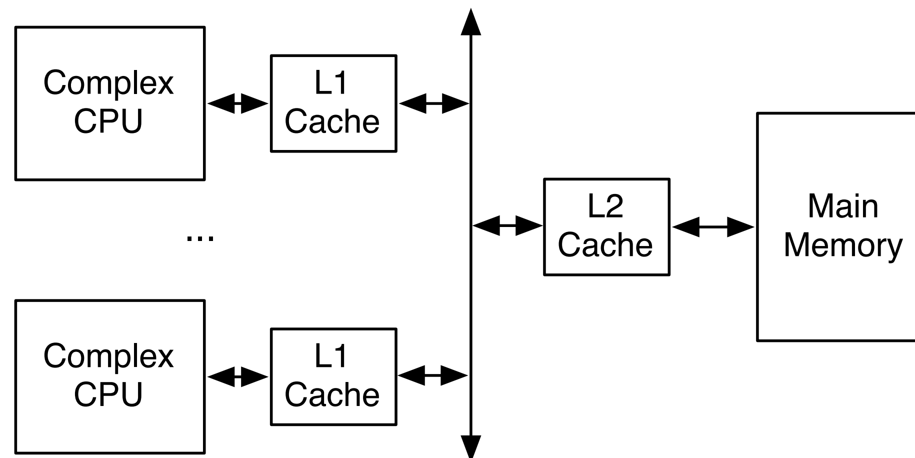
Radojkovic et al. (ACM TACO, 2012) on Intel Atom and Intel Core 2 Quad:

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

What does the execution time depend on?

SAARLAND UNIVERSITY
COMPUTER SCIENCE

- 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.



Three Challenges:

Modeling

How to obtain sound timing models?

Analysis

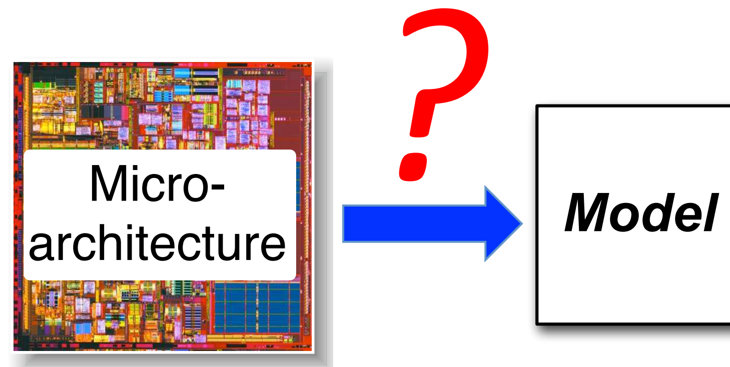
How to precisely & efficiently bound the WCET?

Design

How to design microarchitectures that enable precise & efficient WCET analysis?

The Modeling Challenge

Predictions about the future behavior of a system are always based on models of the system.



All models are wrong, but some are useful.

George Box (Statistiker)

The Need for Timing Models

The ISA only **partially** defines the behavior of microarchitectures: it **abstracts from timing**.

How to obtain **timing models**?

- Hardware manuals
- Manually devised microbenchmarks
- Machine learning

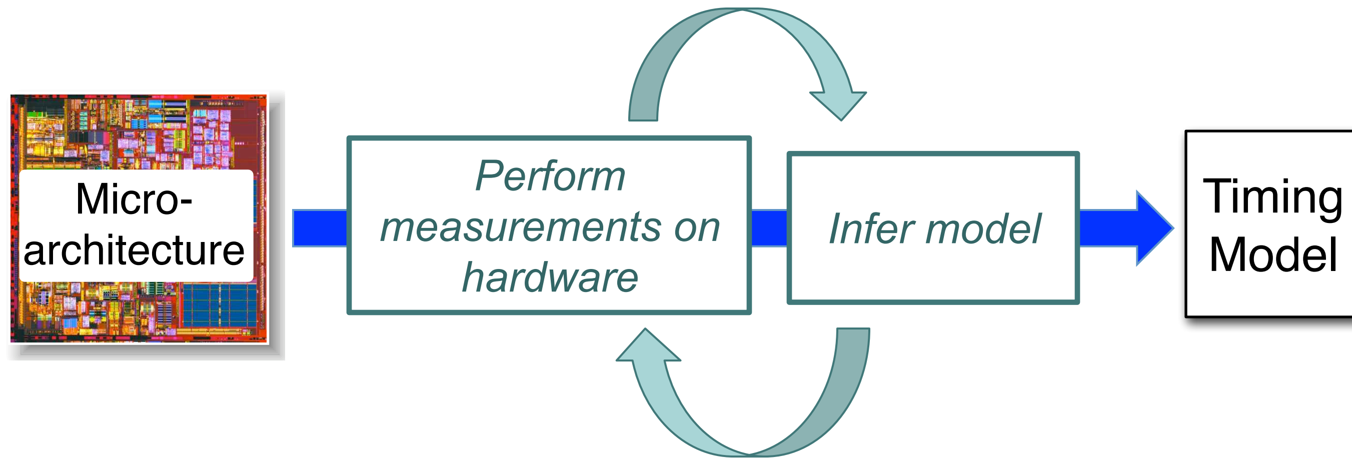
Challenge*: Introduce HW/SW contract to capture timing behavior of microarchitectures.*

Current Process of Deriving Timing Models

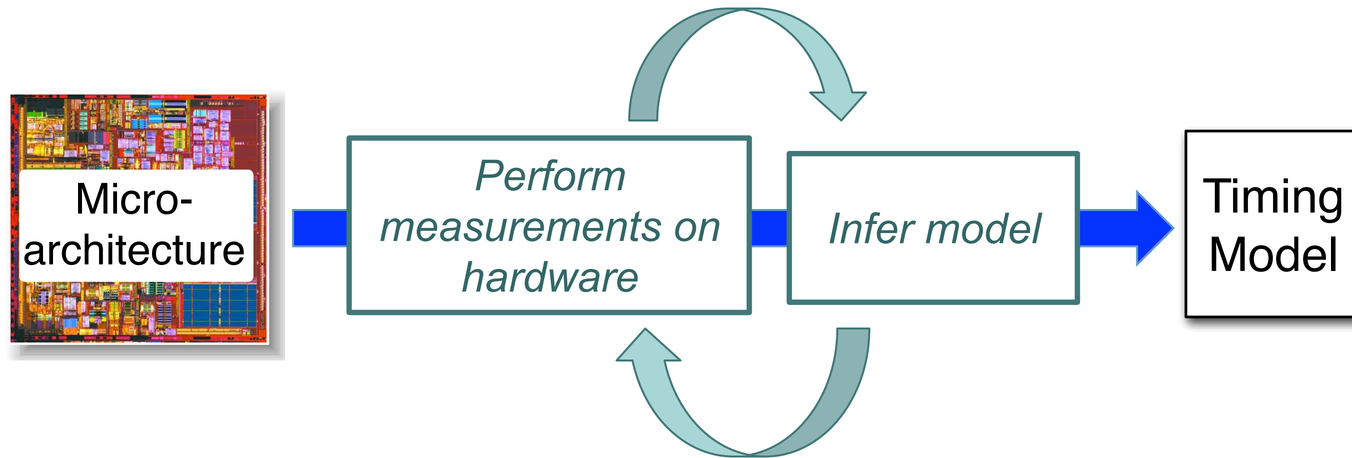
→ Time-consuming, and

→ error-prone.

Can We Automate the Process?



Can We Automate the Process?



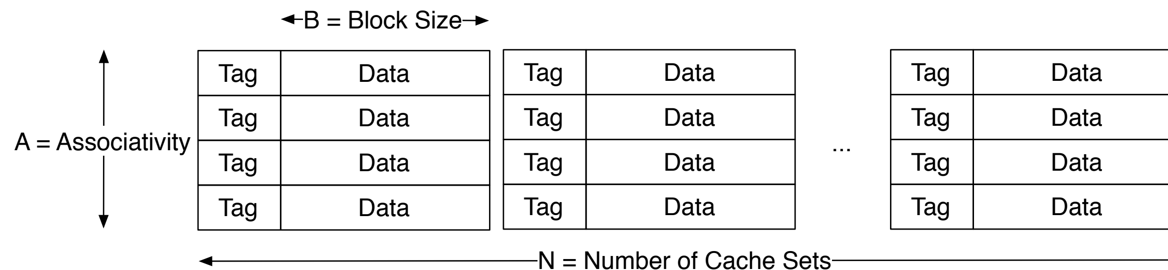
Derive timing model automatically from measurements on the hardware using methods from automata learning.

→ *No manual effort, and*

→ *(under certain assumptions) provably correct.*

Proof-of-concept:
Automatic Modeling of the Cache Hierarchy

- Can be characterized by a few parameters:
  - ABC: associativity, block size, capacity
  - Replacement policy: finite automaton



chi [Abel and Reineke, RTAS] derives all of these parameters fully automatically including previously undocumented replacement policies.

Modeling Challenge: Ongoing and Future Work

1. Extend automata learning techniques to account for prior knowledge
   [NASA Formal Methods Symposium, 2016]
2. Apply approach to other parts of the microarchitecture:
   - Translation lookaside buffers, branch predictors
   - Shared caches in multicores including their coherency protocols
   - **Contemporary out-of-order cores**

Analysis and Design Challenges

Design for Predictability

How to design hardware to allow for precise and efficient analysis without sacrificing performance?

Precise & Efficient Timing Analysis

How to precisely and efficiently account for caches, pipelining, speculation, etc.?

The Analysis Challenge: State of the Art

SAARLAND UNIVERSITY
COMPUTER SCIENCE

Private Caches

Precise & efficient abstractions, for

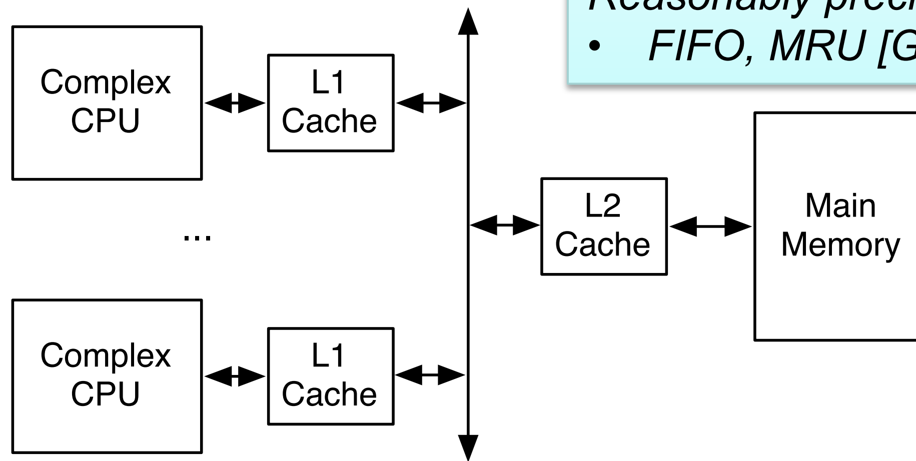
- *LRU [Ferdinand, 1999]*

Not-as-precise but efficient abstractions, for

- *FIFO, PLRU, MRU [Grund and Reineke, 2008-2011]*

Reasonably precise quantitative analyses, for

- *FIFO, MRU [Guan et al., 2012-2014]*



Shared Resources on Multicores

Major challenge: interference on shared resources

→ execution time depends on corunning tasks

→ need timing compositionality

Complex Pipelines

Precise but very inefficient analyses; little abstraction

Major challenge: timing anomalies

Contributions to Analysis and Design Challenges

Analysis

- Caches [SIGMETRICS 08, SAS 09, WCET 10, ECRTS 10, CAV 17]
- Branch Target Buffe… [RTCSA 09, JSA 10]
- Preemption Cost [… 09, LCTES 10, RTNS 16]
- Architecture-Parametric Timing Ana… [RTAS 14]
- Multi-Core Timing Analysis [RTNS 15, DA… RTNS 16]

Design

Hardware:

- Shared DRAM Controller [CODES+ISSS 11]
- Preemption-aware Cache [RTAS 14]
- Smooth Shared Caches [WAOA 15]
- Anomaly-free Pipelines [Correct Sys. Des. 15]

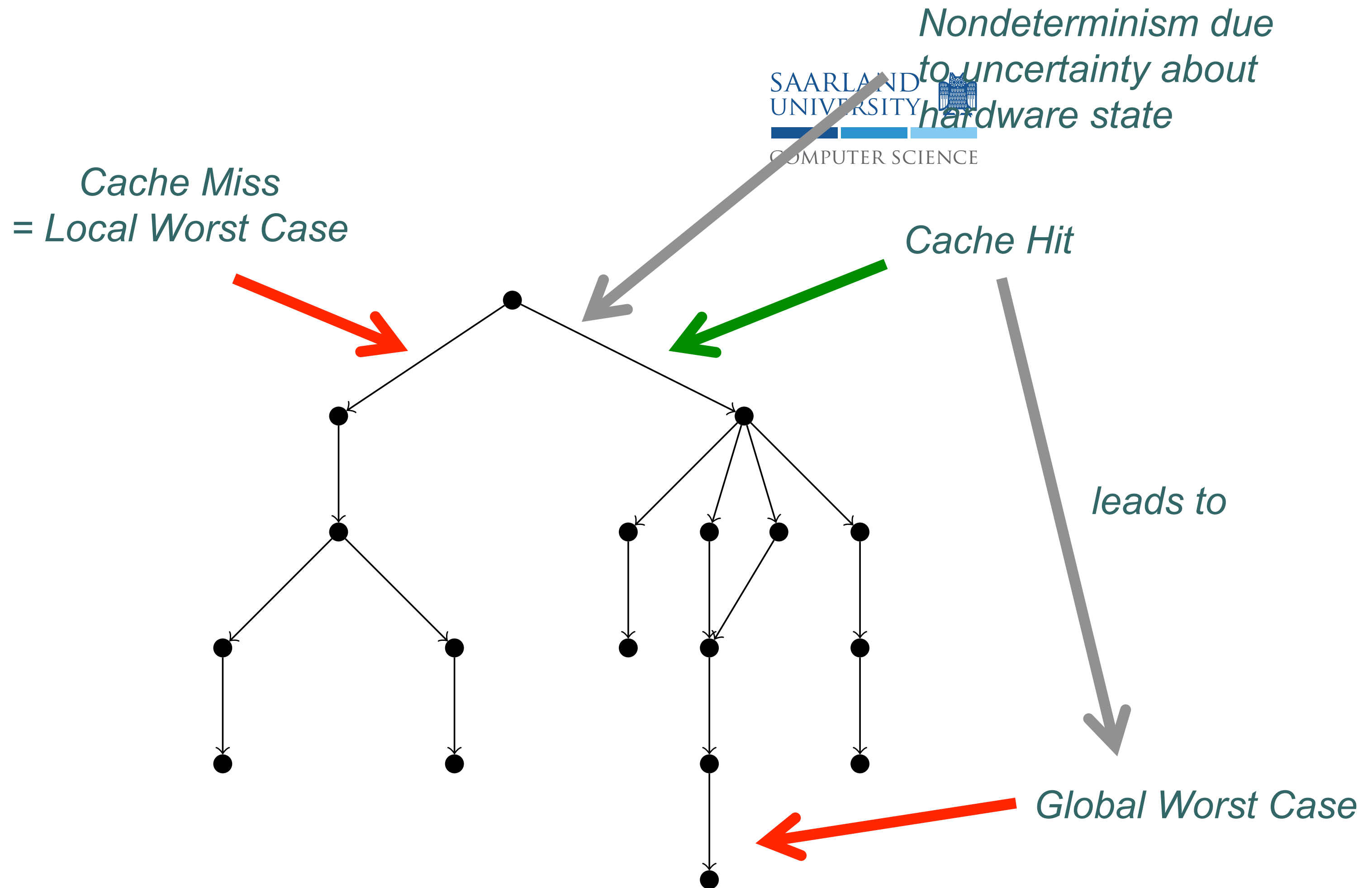
Software:

- Predictable Memory Allocation [ECRTS 11]
- Compilation for Predictability [RTNS 14]

Predictability Assessment

- (Randomized) Caches [RTS 07, TECS 13, LITES 14, WAOA 15]
- Branch Target Buffers [JSA 10]
- Pipelines and Buses [TCAD 09]
- Load/Store-Unit [WCET 12]
- Timing Anomalies [WCET 06]
- Timing Compositionality [CRTS 13]

Timing Anomalies

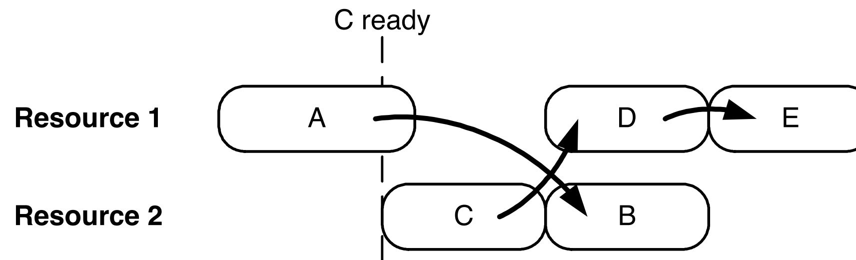


Timing Anomalies in Dynamically Scheduled Microprocessors
T. Lundqvist, P. Stenström – RTSS 1999

Timing Anomalies

Timing Anomaly = Counterintuitive scenario in which the "local worst case" does not imply the "global worst case".

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 Anomalies
Consequences for Timing Analysis

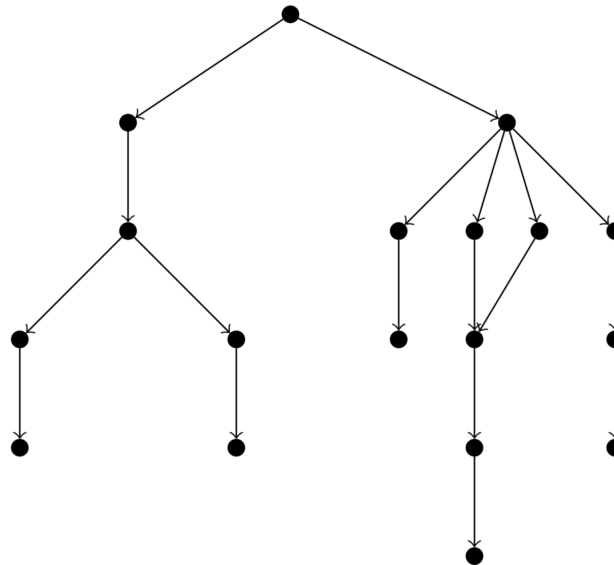
Cannot exclude cases “locally”:

→ Need to consider all cases

→ May yield “State explosion problem”

SAARLAND UNIVERSITY

COMPUTER SCIENCE



Conventional Wisdom

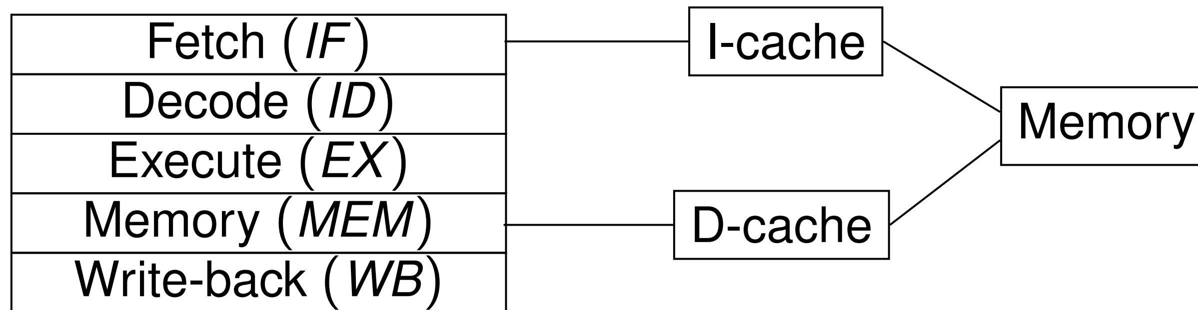
Simple in-order pipeline + LRU caches

→ no timing anomalies

→ timing compositional

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

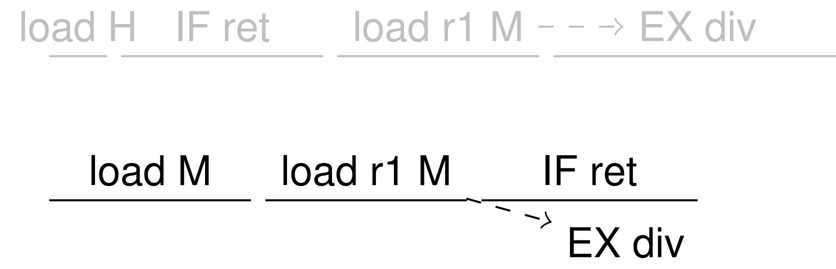
SAARLAND UNIVERSITY
COMPUTER SCIENCE

Program:

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

Pipeline State:

Stage	State
IF	(load r1, 0)
ID	
EX	(load, 0)
MEM	
WB	



Hit case:

- Instruction fetch starts before second load becomes ready
- S

Mis

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

> *Intuitive Reason:*
> Progress in the pipeline influences order of instruction fetch and data access

Good News: Strictly In-Order Pipelines

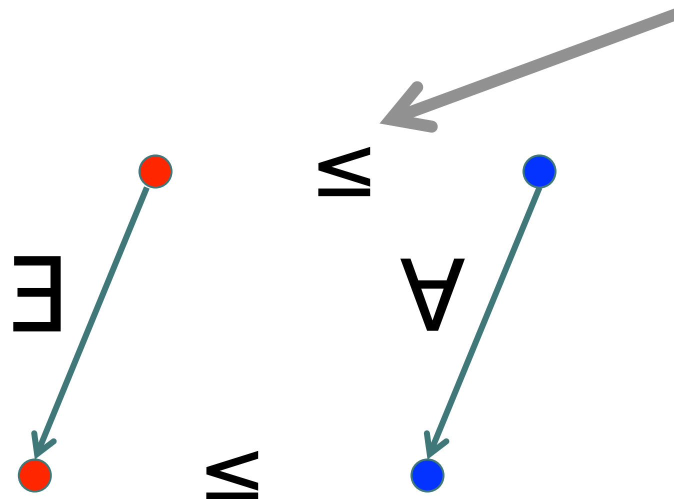
> *Definition (Strictly In-Order):*
> We call a pipeline *strictly in-order* if each *resource* processes the instructions in program order.

- 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

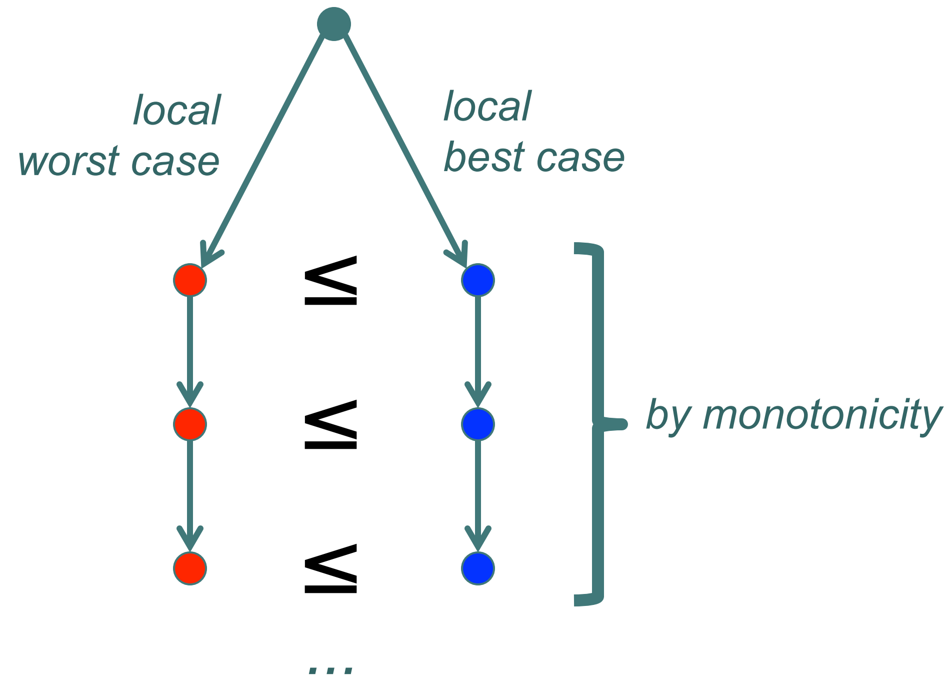
Theorem 1 (Monotonicity):
In the strictly in-order pipeline progress of an instruction is monotone in the progress of other instructions.

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



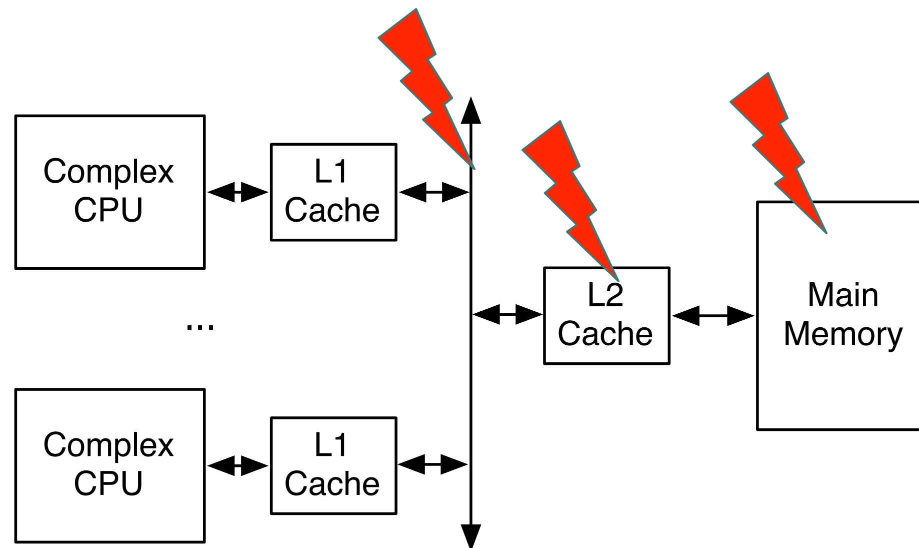
Strictly In-Order Pipelines: Properties

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



Multi-Core Timing Analysis

Execution time depends strongly on execution context due to interference on shared resources



“Standard Approach” for Timing Analysis

Two-phase approach:

1. Determine WCET (worst-case execution time) bounds for each task on platform
2. Perform response-time analysis

Simple interface between WCET analysis and response-time analysis: WCET bounds

Still adequate in case of multi cores?

Three Approaches to Timing Analysis for Multi- and Many-Cores

1. Murphy Approach

Maintain standard two-phase approach:

1. Determine context-independent WCET bound
2. Perform response-time analysis

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

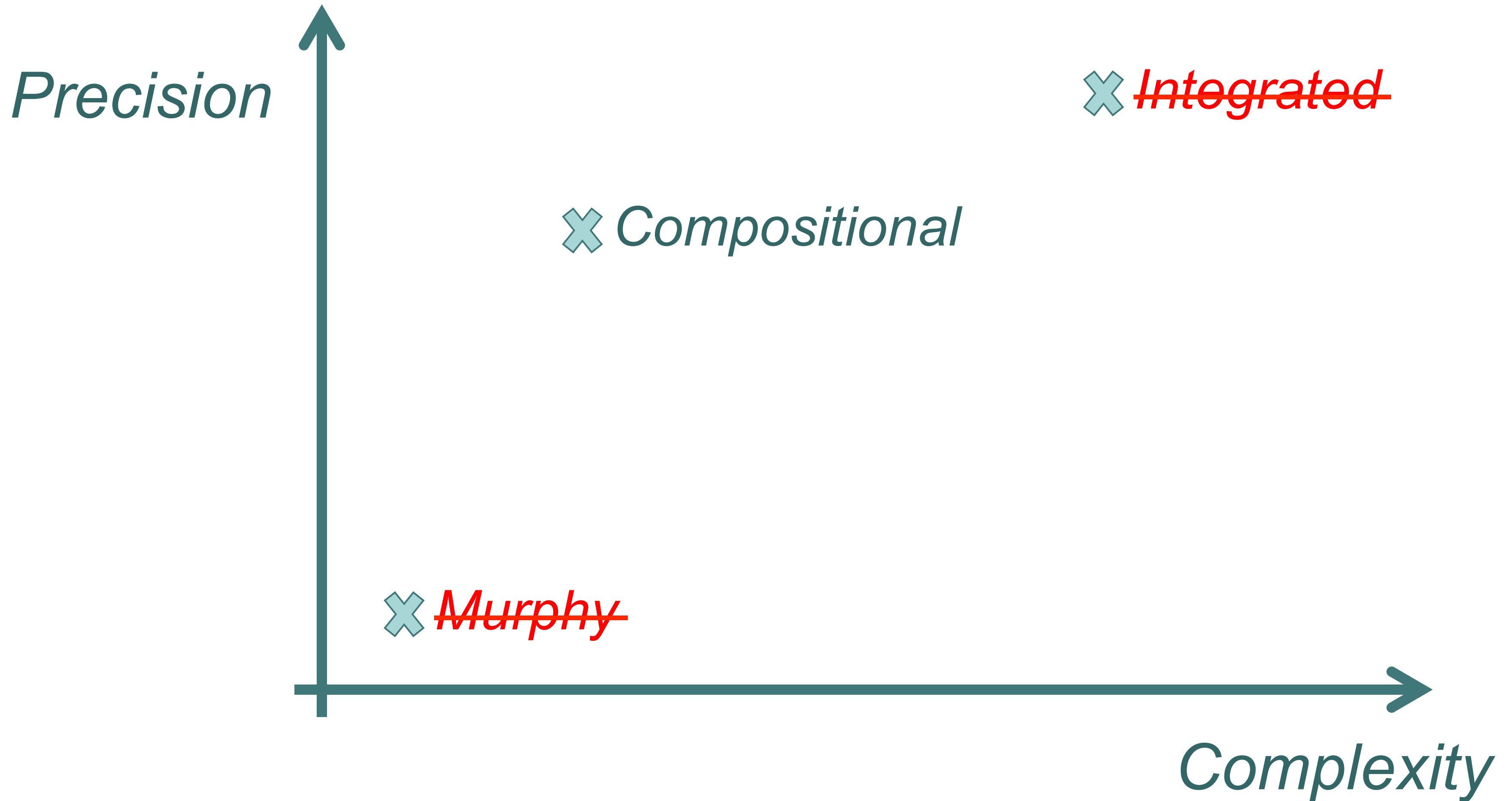
→ Results will be extremely pessimistic

2. Integrated Analysis Approach

Analyze entire task set at once in a combined WCET and response-time analysis

→ Infeasible even for the analysis of two co-running tasks

Three Approaches to Timing Analysis for Multi- and Many-Cores

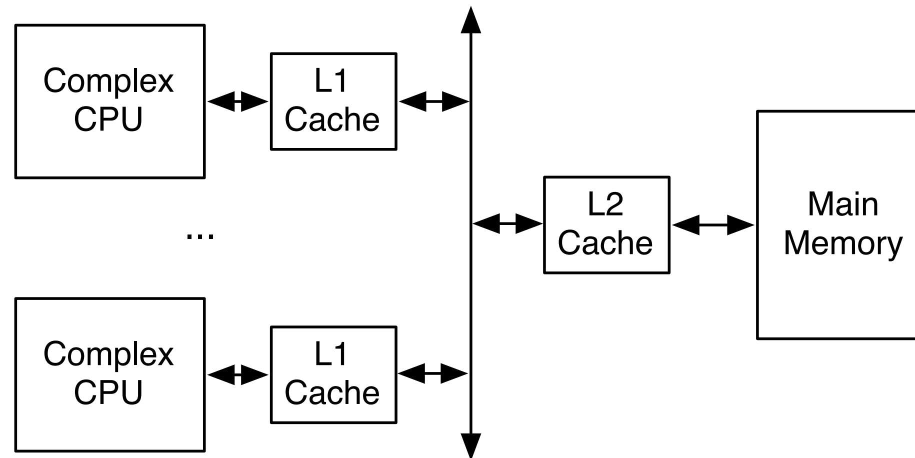


3. Compositional Approach

1. "WCET Analysis": for each task:
   a) Compute WCET bound assuming no interference
   b) Compute maximal interference generated by task on each shared resource

2. Perform extended response-time analysis

3. Compositional Approach: Response-time Analysis [RTNS 15, DAC 16]



Response time of a task = Execution time in isolation

+ Interference on its Core

+ Interference on Caches

+ Interference on Bus

+ Interference on Memory

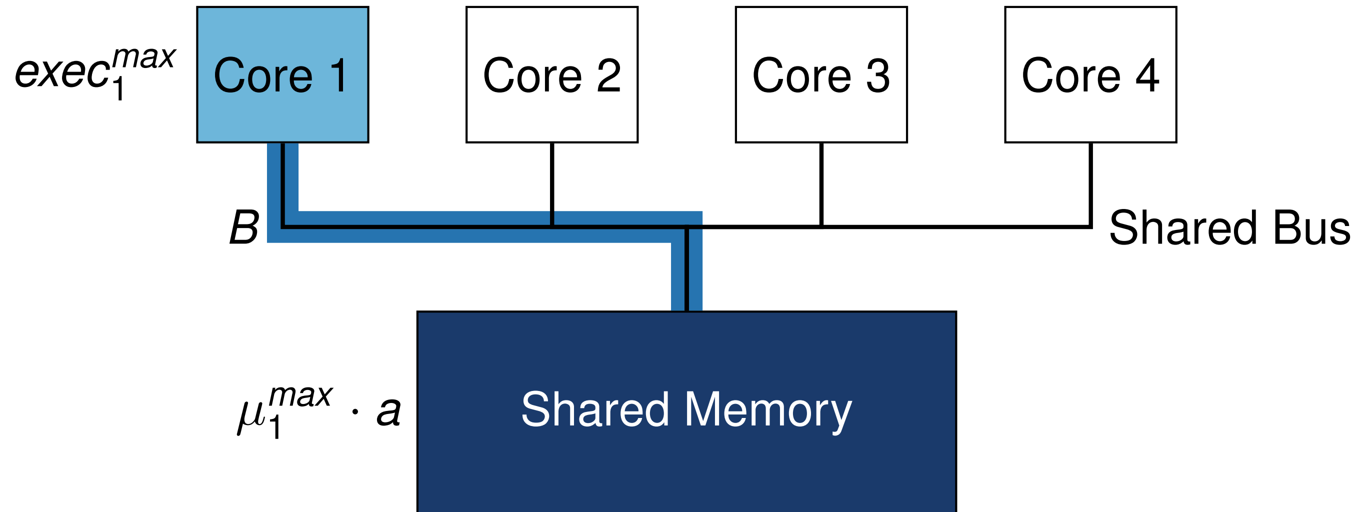
3. Compositional Approach: Challenges

What are good interference characterizations?

→ Want precision and analysis efficiency

Approaches usually rely on timing compositionality.

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 of Conventional In-order Pipeline

Maximal cost of an additional cache miss?

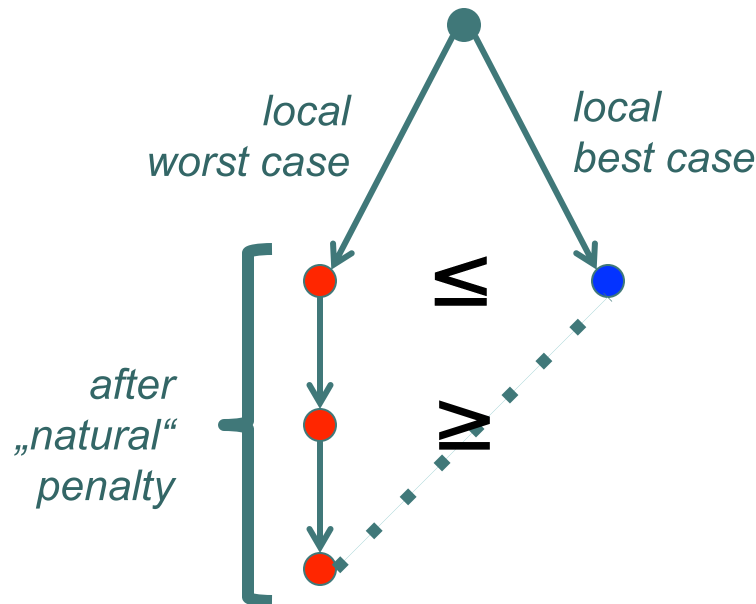
Intuitively: cache miss penalty

Unfortunately:

- Common case: less than cache miss penalty
- But worst case: ~ 2 times cache miss penalty
  - ongoing instruction fetch may block load
  - ongoing load may block instruction fetch

Strictly In-Order Pipelines: Properties

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



Conclusions

Modeling Timing analysis needs timing models; models can be obtained by machine learning

Analysis Multicores require rethinking interface between WCET analysis and response-time analysis

Design
- Simple, in-order pipelines do not fulfill assumptions of state-of-the-art analyses
- *Strictly in-order pipeline* is free of timing anomalies and timing-compositional

  → Component of future predictable multi-cores!?

Thank you for your attention!

Some References

Gray-box Learning of Serial Compositions of Mealy Machines
A. Abel and J. Reineke. In NASA Formal Methods Symposium, 2016.

MIRROR: Symmetric Timing Analysis for Real-Time Tasks on Multicore Platforms with Shared Resources
W.-H. Huang, J.-J. Chen, and J. Reineke. In DAC, 2016.

A Generic and Compositional Framework for Multicore Response Time Analysis
S. Altmeyer, R.I. Davis, L.S. Indrusiak, C. Maiza, V. Nelis, and J. Reineke. In RTNS, 2015.

Toward Compact Abstractions for Processor Pipelines
S. Hahn, J. Reineke, and R. Wilhelm. In Correct System Design, 2015.

A Compiler Optimization to Increase the Efficiency of WCET Analysis
M. A. Maksoud and J. Reineke. In RTNS, 2014.

Architecture-Parametric Timing Analysis
J. Reineke and J. Doerfert. In RTAS, 2014.

Selfish-LRU: Preemption-Aware Caching for Predictability and Performance
J. Reineke, S. Altmeyer, D. Grund, S. Hahn, C. Maiza. In RTAS, 2014.

Towards Compositionality in Execution Time Analysis - Definition and Challenges
S. Hahn, J. Reineke, and R. Wilhelm. In CRTS, 2013.

Impact of Resource Sharing on Performance and Performance Prediction: A Survey
A. Abel, F. Benz, J. Doerfert, B. Dörr, S. Hahn, F. Haupenthal, M. Jacobs, A. H. Moin, J. Reineke, B. Schommer, and R. Wilhelm. In CONCUR, 2013.

Measurement-based Modeling of the Cache Replacement Policy
A. Abel and J. Reineke. In RTAS, 2013.

A PRET Microarchitecture Implementation with Repeatable Timing and Competitive Performance
I. Liu, J. Reineke, D. Broman, M. Zimmer, and E.A. Lee. In ICCD, 2012.

PRET DRAM Controller: Bank Privatization for Predictability and Temporal Isolation
J. Reineke, I. Liu, H.D. Patel, S. Kim, and E.A. Lee. In CODES+ISSS, 2011.