Hardware-Software Interface Issues in Heterogeneous Systems: Design, Verification, and Programming.

Margaret Martonosi
Princeton University

http://www.istc-cc.cmu.edu/

Acknowledgements & Collaborators

- Students on this project:
  - Dan Lustig, Princeton
    - 2013 Intel Graduate Fellow
  - Caroline Trippel, Princeton
- Collaboration with Dr. Michael Pellauer, Intel VSSAD, Hudson, MA

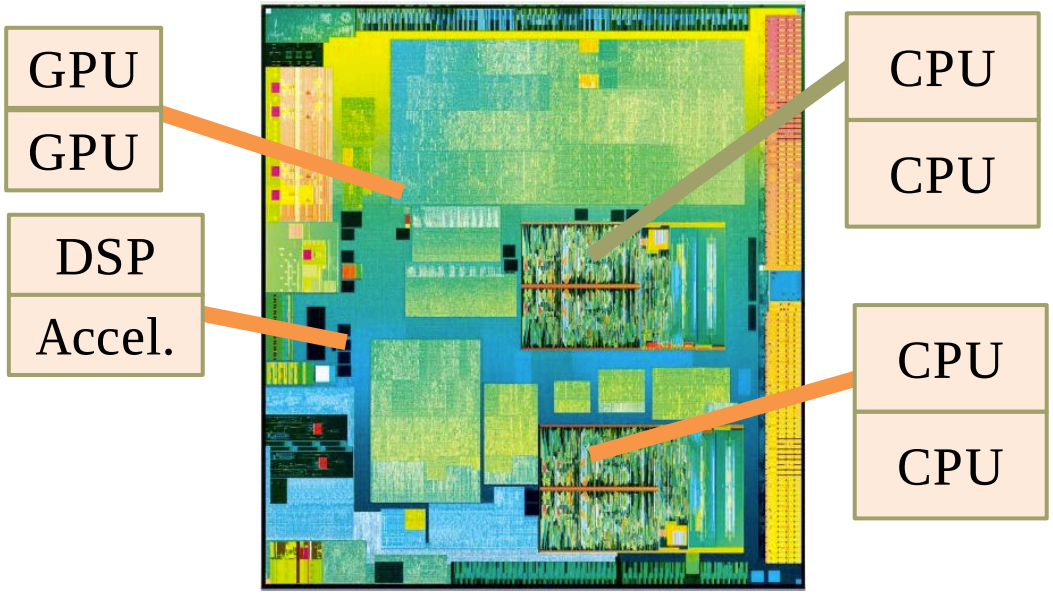
- Related papers:
  - "PipeCheck: Specifying and Verifying Microarchitectural Enforcement of Memory Consistency Models" To Appear. MICRO 2014.
  - Other work in submission.

- Other ISTC-CC students
  - Elba Garza, Tae Jun Ham, Wenhao Jia, Logan Stafman, Ozlem Bilgir Yetim, Yavuz Yetim.

Heterogeneity: HW/SW Challenges

Heterogeneous Parallelism: Widespread, clear benefits:

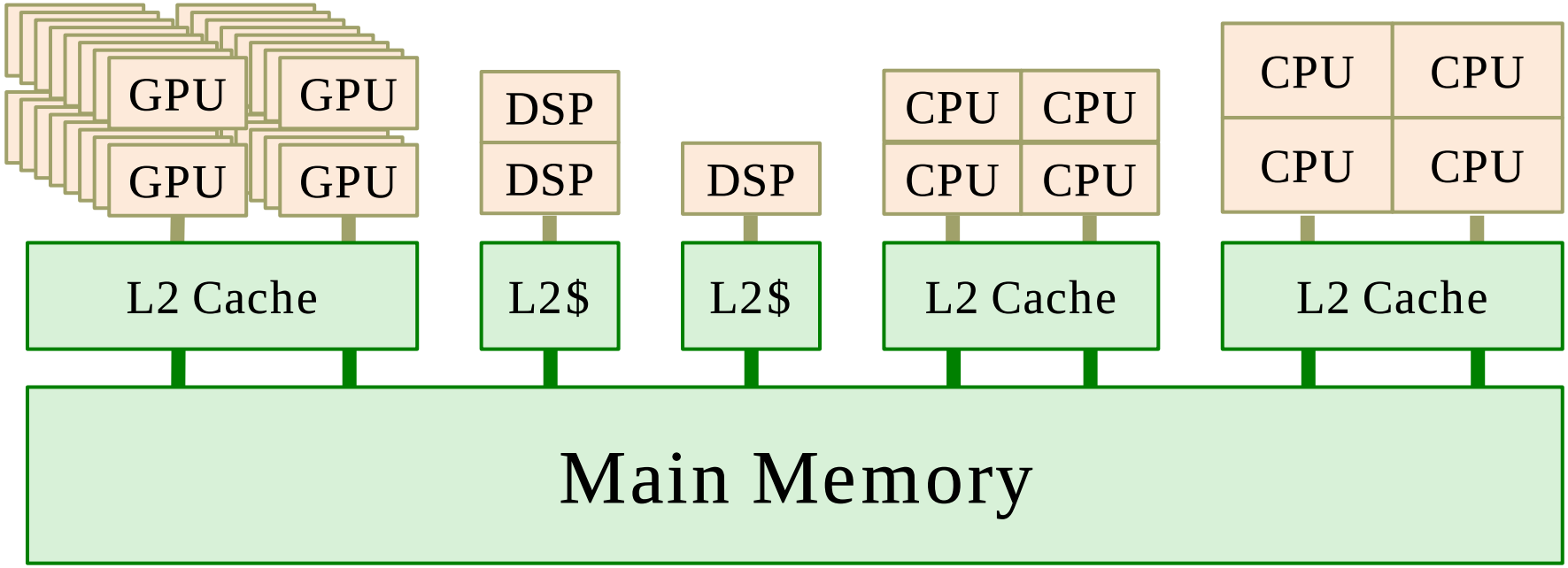
- Specialization => high perf-per-watt
- On-chip, On-Device, Datacenters, Cloud...



e.g., Intel Bay Trail SoC

- But programming model is weak:
  - "1 CPU + peripheral devices" is nowhere near rich enough for today's complex processors and systems.
- Our Overall Focus: Programming models and execution optimizations for Heterogeneous CMPs.
- One Specific Issue: Memory Consistency Models

Diverse processing resources share data in memory…



Memory consistency model: the set of rules on the ordering and visibility of memory accesses

The Challenge: Memory Consistency models are as diverse as the processors employing them!

Consistency Models & Preserved Program Order

- Memory Consistency Models (MCMs) set the rules regarding which loads and stores may be reordered relative to each other.
  - E.g., Total Store Order (TSO) used by x86
- Other architectures make different (often much weaker) ordering promises.
  - E.g. GPUs make very weak ordering promises
  - Even other CPU ISAs (IBM, ARM) make fairly weak promises
- If “default” promise is weak, programs/compilers use fence instructions to provide ordering guarantees required by the program

Our Research Goals

- How to verify the implementation of a memory consistency model in a given processor pipeline?
- How to dynamically translate executing code from one consistency model to another at runtime?
  - Dynamic Consistency Model Translation ≈ Dynamic Binary Translation
- Seamless design of memory models with “black-box” IP Blocks?
- Dynamic optimization regarding migration and optimization of resources to use?

Our Research Goals

- How to verify the implementation of a memory consistency model in a given processor pipeline?
- How to dynamically translate executing code from one consistency model to another at runtime?
  - Dynamic Consistency Model Translation ≈ Dynamic Binary Translation
- Seamless design of memory models with “black-box” IP Blocks?
- Dynamic optimization regarding migration and optimization of resources to use?

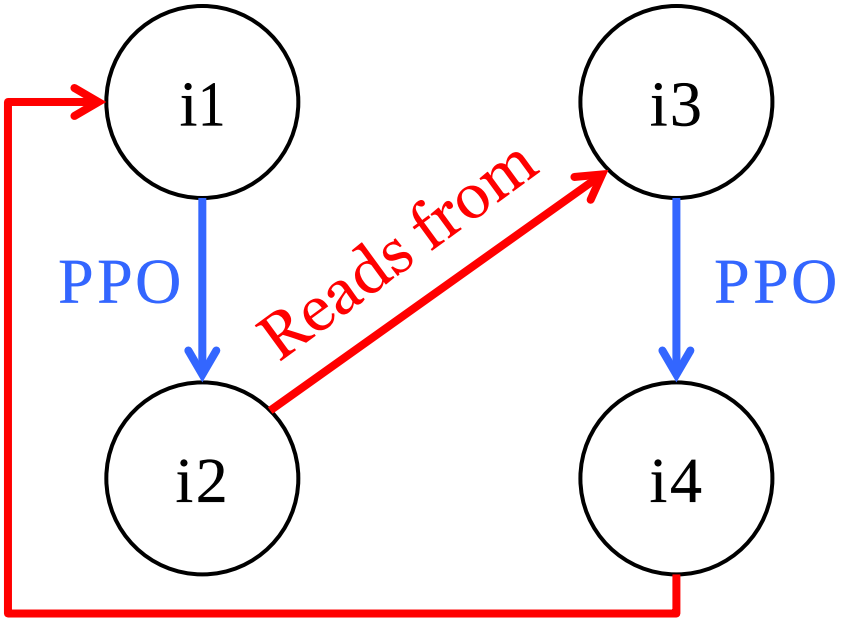
Architecture-Level Analysis: Happens-before Graphs

Litmus test:

Core 0	Core 1
(i1) st [x], 1 (i2) st [y], 2	(i3) ld [y] → r1 (i4) ld [x] → r2

TSO: Forbid r1=2, r2=0

Analysis:



Reads from earlier ("from-reads")

Generally: A cycle implies that execution is forbidden

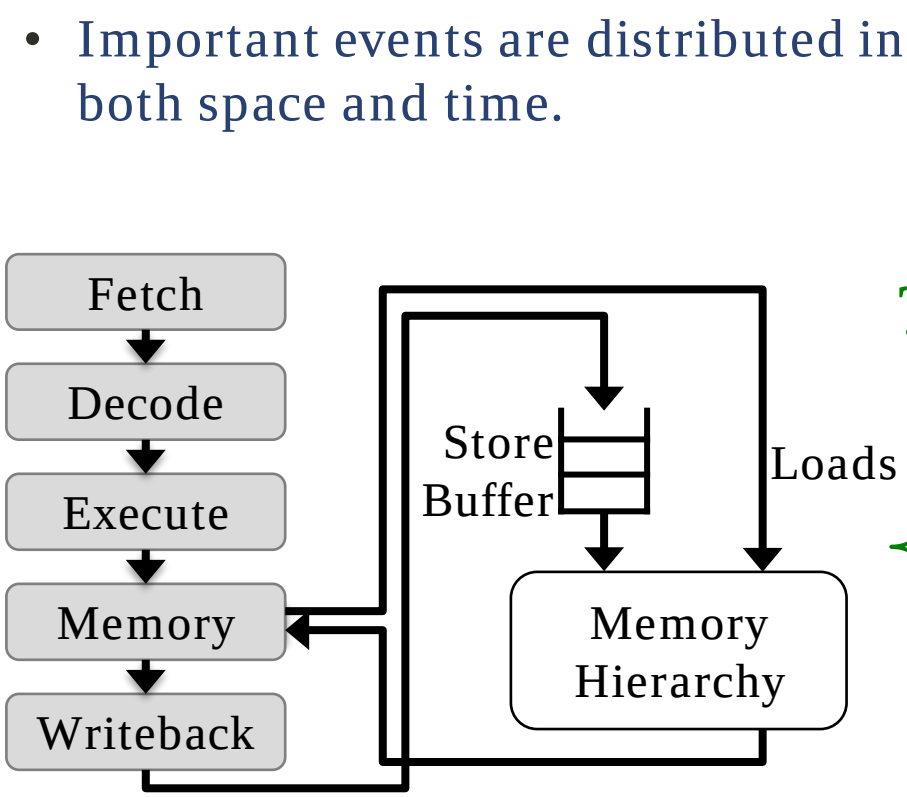
(Intuition: instruction can't happen before itself)

[Alglave, FMSD '09]

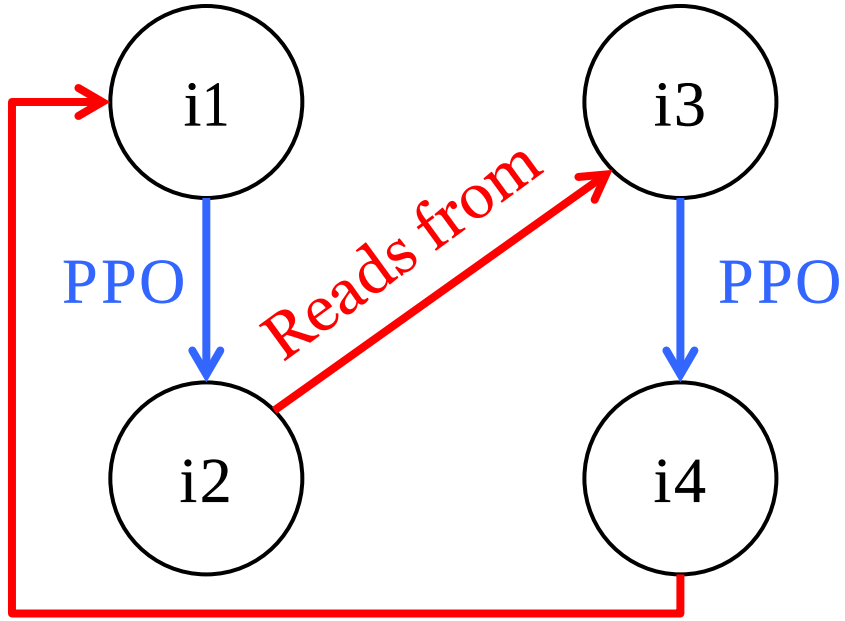
Architecture vs. Implementation…

How is the consistency model enforced at the **microarchitecture level**?

- Important events are distributed in both space and time.



?

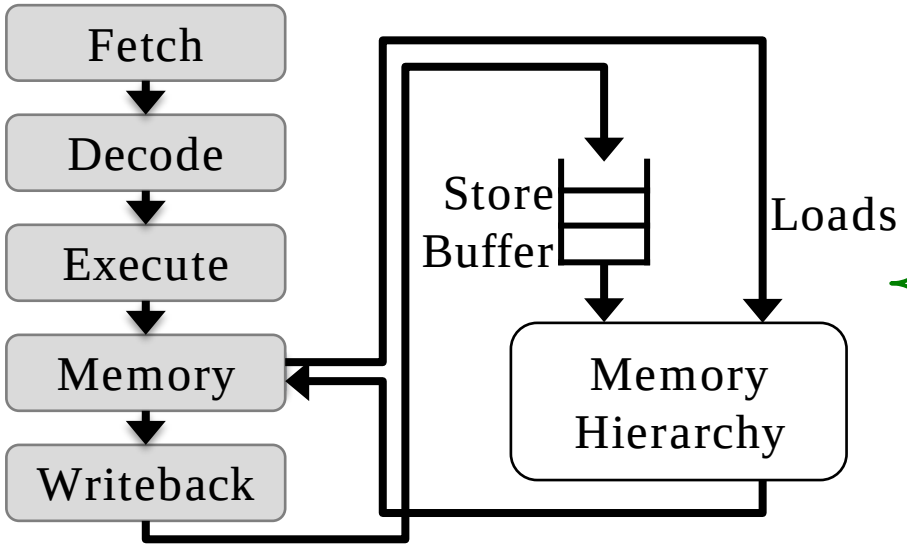


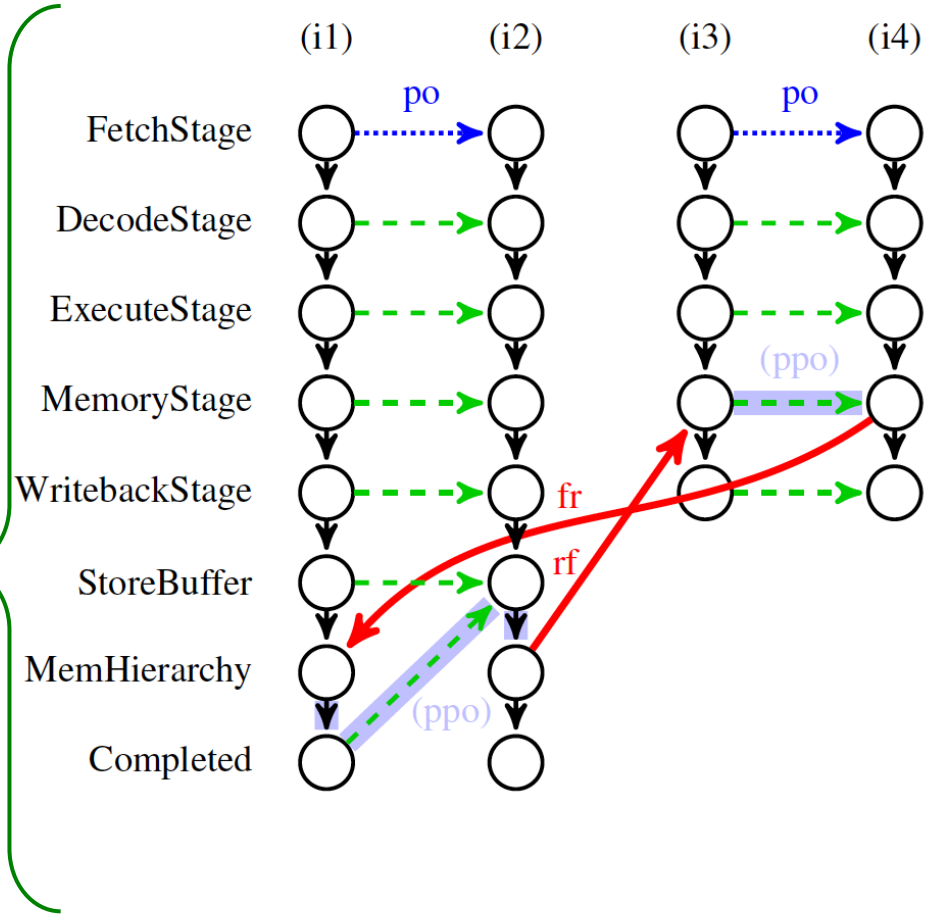
Reads from earlier ("from-reads")

PipeCheck Verification: Overview

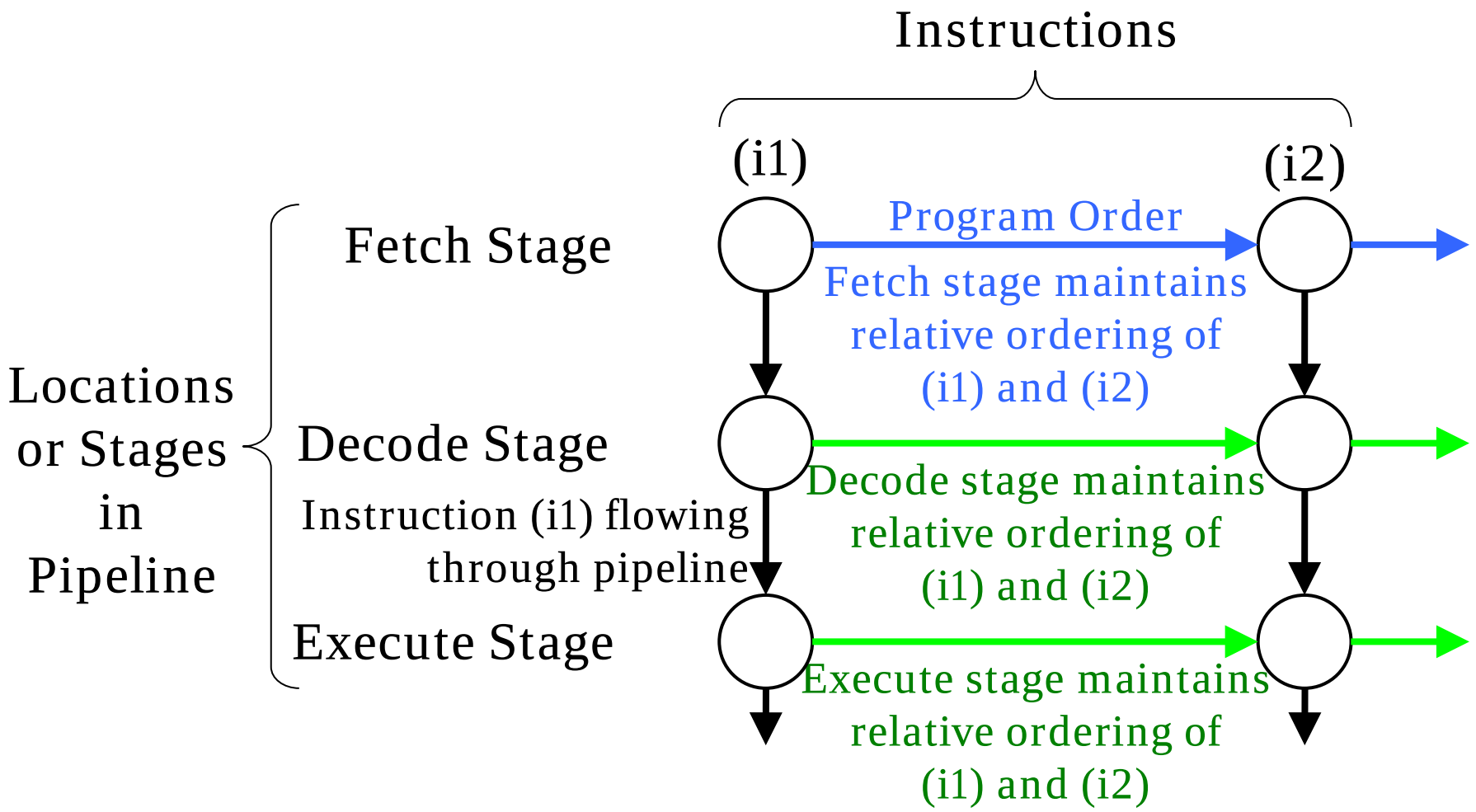
- **Model Specification +**
- **Automatic Analysis**

=> Microarchitectural equivalent of "happens-before" graphs

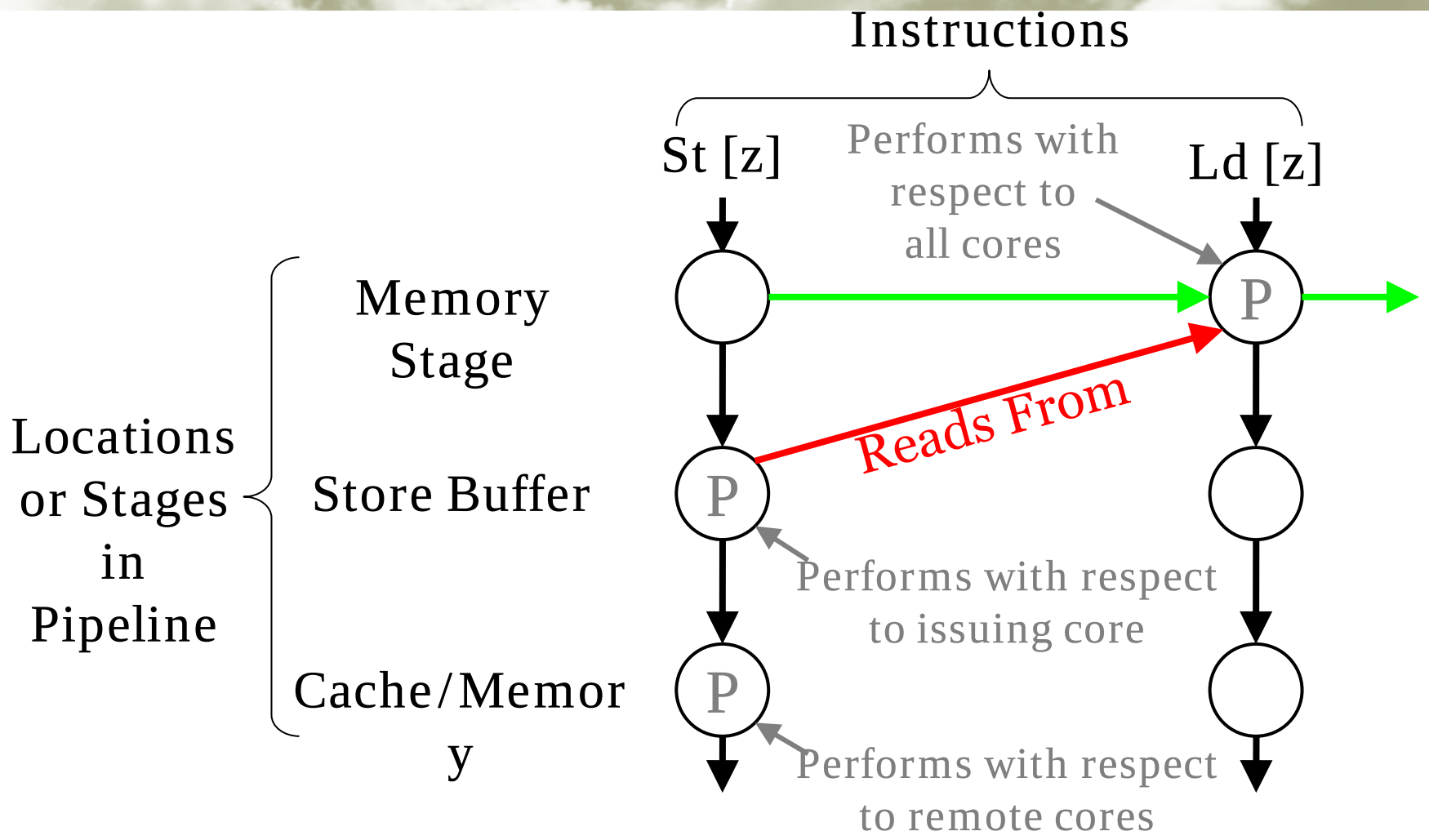




μ-Happens-Before Graphs

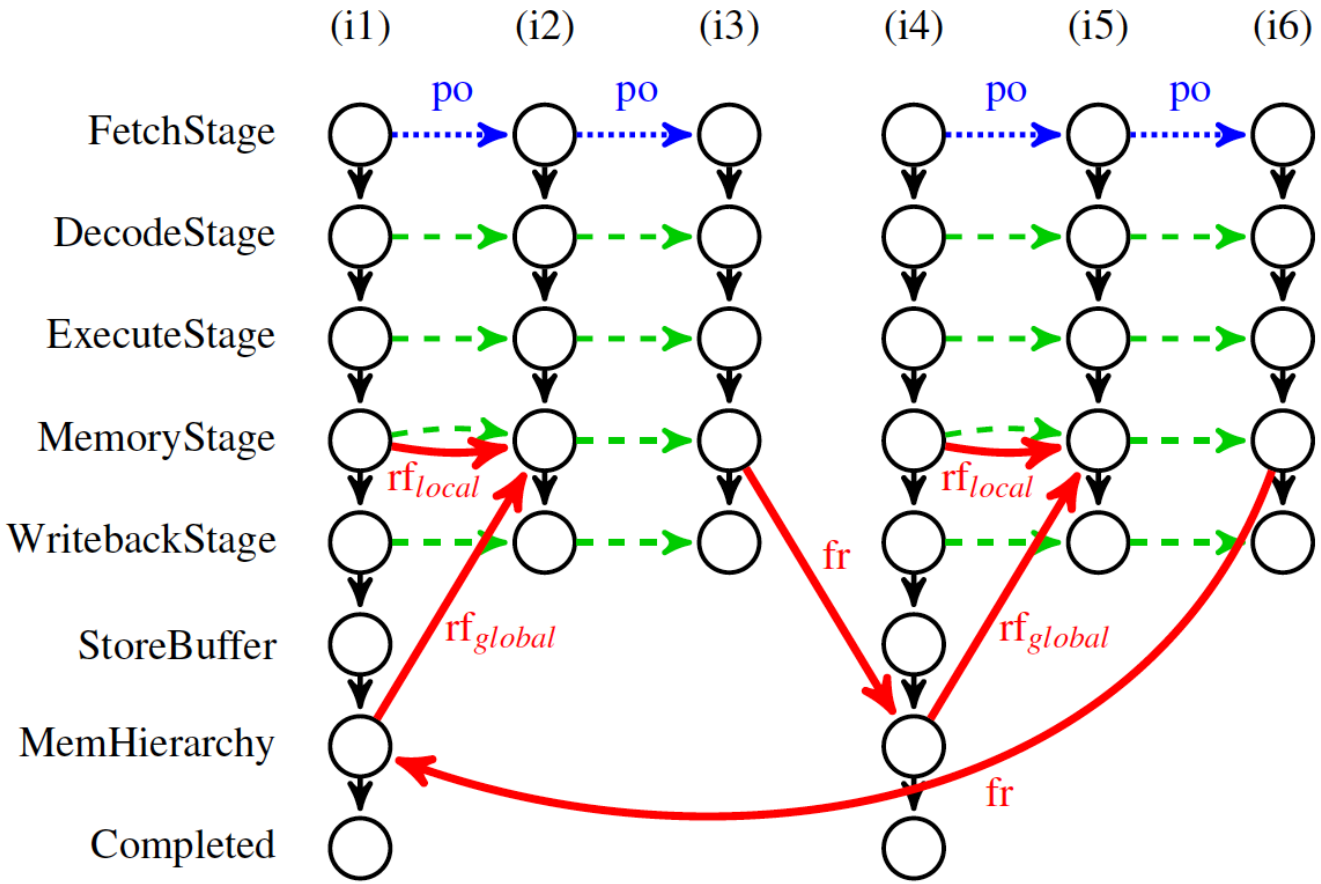


μ-Happens-Before Graphs



More Complex Cases

With a µarch-level view, special exceptions are no longer needed



PipeCheck Implementation Summary

- Pipelines modeled by specifying:
  - list of stages
  - list of possible paths through the pipeline
  - set of “non-local edges” (details in poster)
  - list of “performing locations”

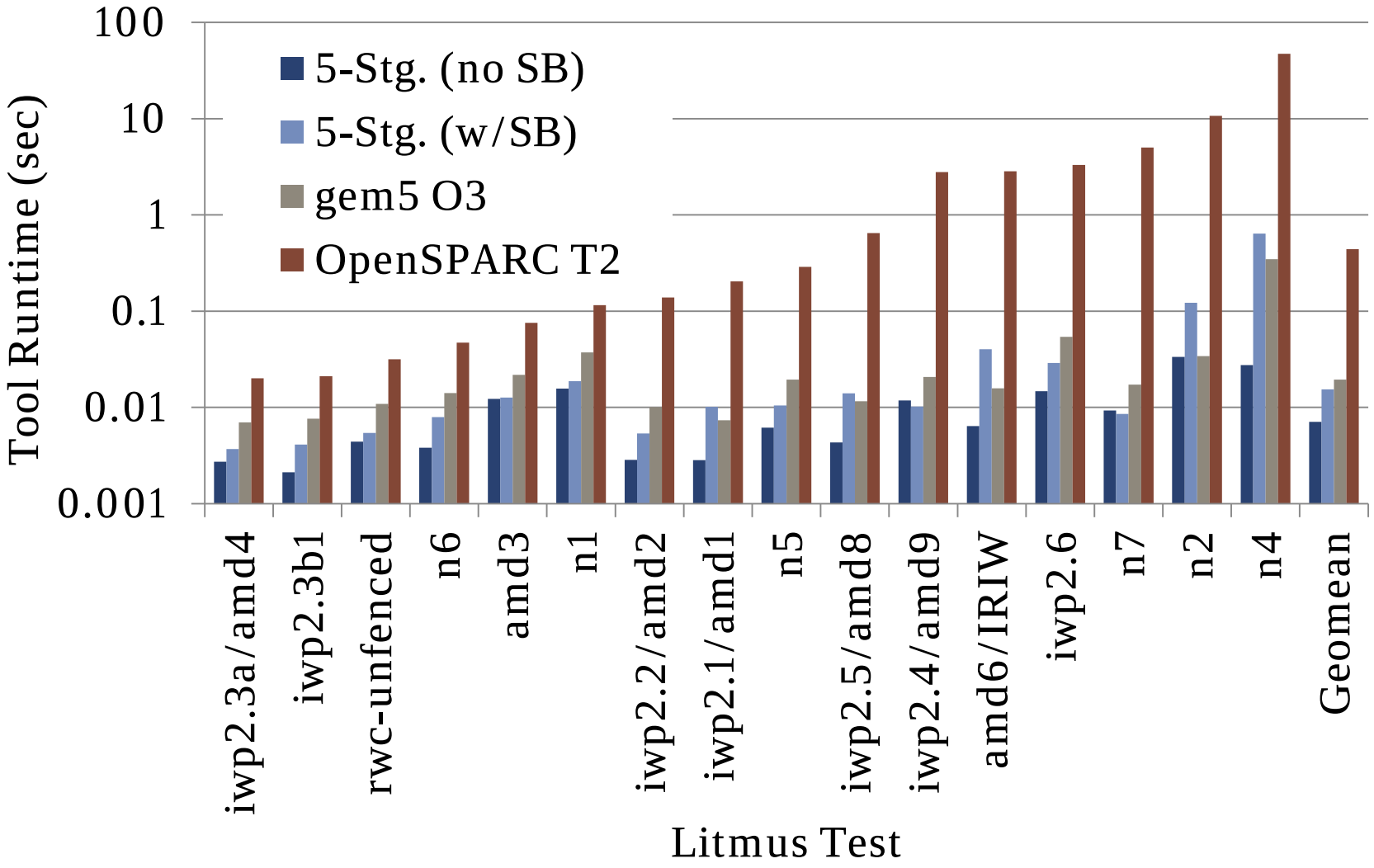
Pipeline	Lines of Code
Classic 5-Stage Pipeline without Store Buffer	37
Classic 5-Stage Pipeline with Store Buffer	62
gem5 O3 CPU Model	106
OpenSPARC T2	115

PipeCheck Implementation Summary

- Tool written in Coq and extracted to OCaml
  - Open to future formal verification
- PipeCheck software verifies each pipeline against suite of litmus tests and PPO tests
  - Automatically enumerate all possible executions (i.e. all possible graphs) for each pipeline model/ test pair

	Observable	**Not Observable**
Permitted	OK	OK (μarch stricter than necessary)
Forbidden	**Pipeline bug!!**	OK

Verification Time



Litmus Test Results

Litmus Test	Expected	5-Stage w/o St. Buf	5-Stage w/ St. Buf	gem5 O3	OpenSPARC T2
iwp2.1/amd1	Forbid	✔	✔	**Observed** [2]	✔
iwp2.2/amd2	Forbid	✔	✔	✔	✔
iwp2.3a/amd4	Permit	Not obs.[1]	✔	✔	✔
iwp2.3b	Permit	✔	✔	✔	✔
iwp2.4/amd9	Permit	Not obs. [1]	✔	✔	✔
iwp2.5/amd8	Forbid	✔	✔	**Observed** [2]	✔
iwp2.6	Forbid	✔	✔	✔	✔
amd3	Permit	Not obs. [1]	✔	✔	✔
amd6	Forbid	✔	✔	**Observed** [2]	✔
n1	Permit	Not obs. [1]	✔	✔	✔
n2	Forbid	✔	✔	**Observed** [2]	✔
n4	Forbid	✔	✔	✔	✔
n5	Forbid	✔	✔	✔	✔
n6	Permit	✔	✔	✔	✔
n7	Permit	Not obs. [1]	✔	✔	✔
rwc-unfenced	Permit	Not obs. [1]	✔	✔	✔

1. Permitted results not observable: pipeline stronger than necessary

2. Forbidden results observed: Found bugs in pipeline!

Other Work…

- ArMOR: Automatic translation from one memory consistency model to another.
  - Binary translation for *consistency model* differences, not just ISA differences.
  - Supports agile migration from one ISA+MCM to another
- Application-Aware Data Motion optimizations.
  - Communication accelerators, not just computation accelerators.
- Design Space Exploration
  - Fast regression-based methods for estimating optimal hw and sw design parameter choices in heterogeneous design spaces.

Conclusions

- Heterogeneous parallelism is here and growing.
- Heterogeneity affects every aspect of our ability to specify, verify, and building performance-optimized parallel systems.
- Recent work: Specification + Automatic Analysis
  - PipeCheck: Fast, automatic verification of consistency implementations against their higher-level architectural abstractions.
  - ArMOR: Fast binary translation of executables from one consistency model to another.
- Future: CPU + accelerator, cross-data-center, ...

Acknowledgements & Collaborators

- Students on this project:
  - Dan Lustig, Princeton
    - 2013 Intel Graduate Fellow
  - Caroline Trippel, Princeton
- Collaboration with Dr. Michael Pellauer, Intel VSSAD, Hudson, MA

- Related papers:
  - "PipeCheck: Specifying and Verifying Microarchitectural Enforcement of Memory Consistency Models" To Appear. MICRO 2014.
  - Other work in submission.

- Other ISTC-CC students
  - Elba Garza, Tae Jun Ham, Wenhao Jia, Logan Stafman, Ozlem Bilgir Yetim, Yavuz Yetim.

Hardware-Software Interface Issues in Heterogeneous Systems: Design, Verification, and Programming.

Margaret Martonosi
Princeton University

http://www.istc-cc.cmu.edu/