The Load Slice Core Microarchitecture

Trevor E. Carlson, Uppsala University

Wim Heirman, Intel

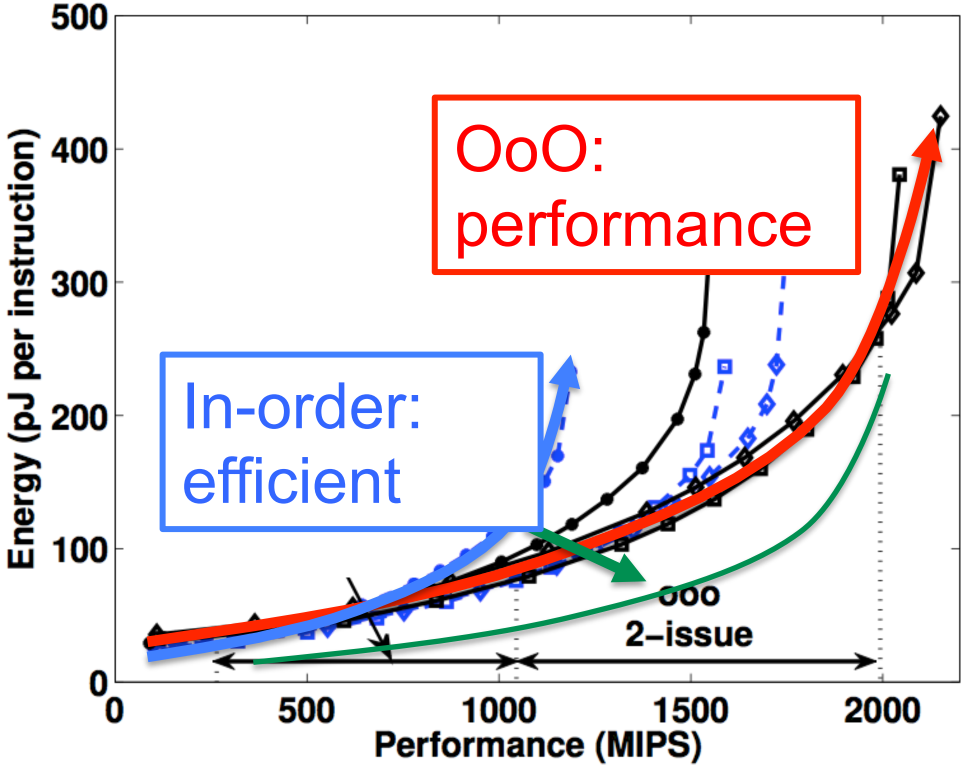
Osman Allam, Ghent University

Stefanos Kaxiras, Uppsala University

Lieven Eeckhout, Ghent University

LSC: Improving Energy Efficiency

- All systems are power-limited
- OoO cores are inefficient



OoO: performance

In-order: efficient

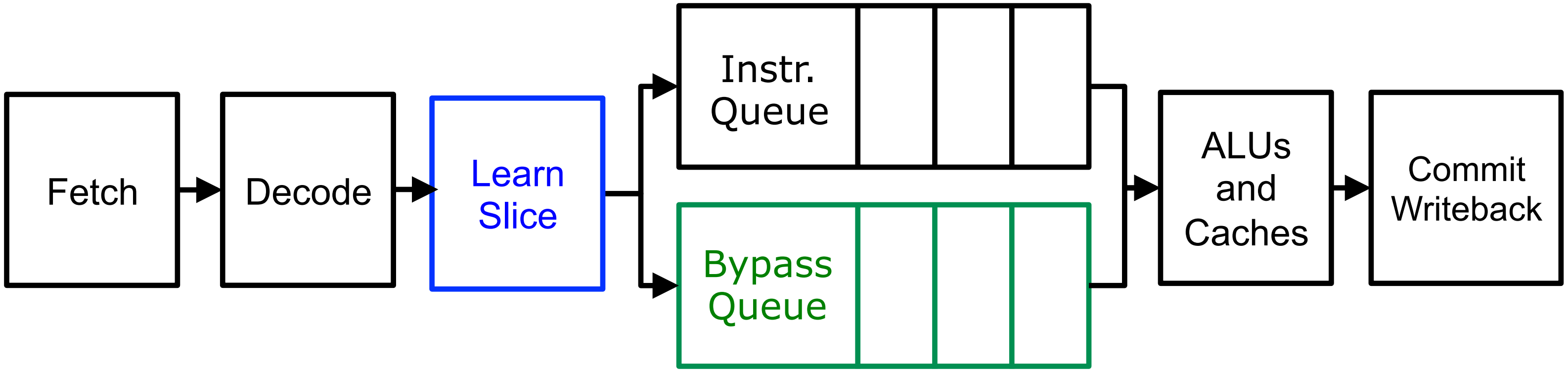
Goal:
OoO performance at in-order efficiency

Azizi, et al., Energy-Performance Tradeoffs in Processor Architecture and Circuit Design: A Marginal Cost Analysis, ISCA 2010

Performance Through MHP

- Goal
  - Out-of-order-like performance with in-order efficiency
- Opportunity for in-order processors:
  - Applications wait for the memory hierarchy
  - Stalls in-order processors
- How to fix and keep efficiency?
  - Identify **Memory Hierarchy Parallelism** (MHP)
  - Prioritize MHP-critical instructions

The Load Slice Core



- Restricted out-of-order core
- Learn critical instruction slices
  - Iterative Backwards Dependency Analysis (IBDA) to find loads and address generating instructions
- Bypass critical instructions
  - Expose MHP for performance

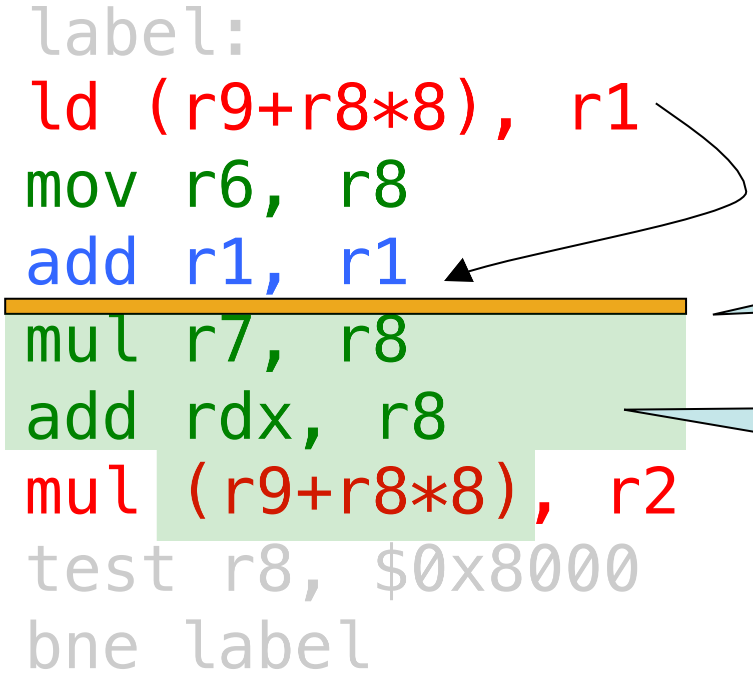
- Prior work
  - Dyn./spec. precomp., Continual flow, slipstream: OoO as a starting point
  - Complexity effective: focuses on ILP, not MHP
  - SLTP, iCFP, flea-flicker two-pass: use extensive structures for slices
  - Runahead execution: re-executes instructions
  - DAE, braid, OUTRIDER, flea-flicker multi-pass: require recompilation
- LSC: hardware-only, does not re-execute

Optimization Example

```
label:
ld (r9+r8*8), r1
mov r6, r8
add r1, r1
mul r7, r8
add rdx, r8
mul (r9+r8*8), r2
test r8, $0x8000
bne label
```

- SPEC CPU2006 leslie3d
- Two load instructions are long-latency
- First use by add
- Key address generating instructions
- Branch instructions left out for clarity

Optimization Example

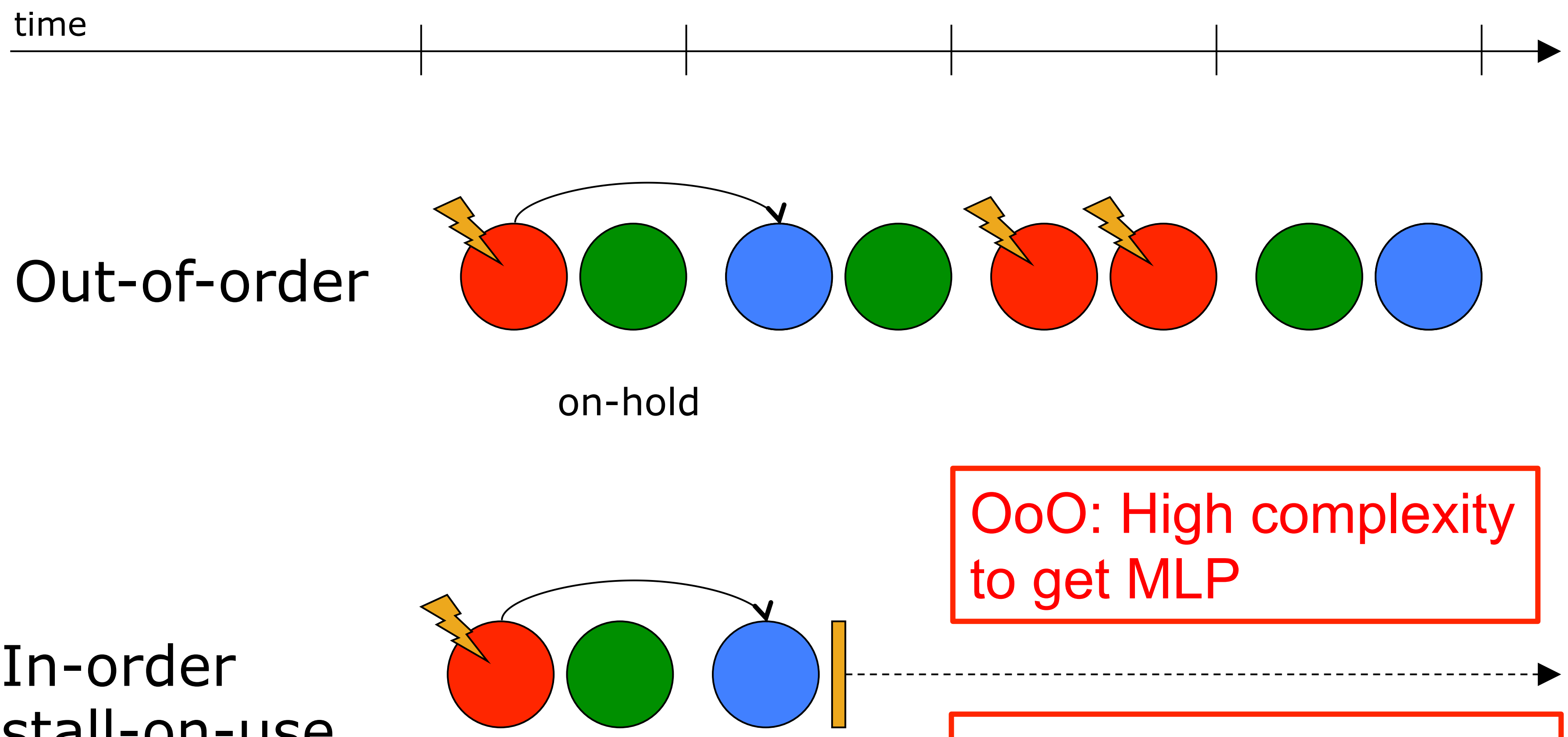


```
label:
ld (r9+r8*8), r1
mov r6, r8
add r1, r1
mul r7, r8
add rdx, r8
mul (r9+r8*8), r2
test r8, $0x8000
bne label
```

- SPEC CPU2006 leslie3d
- Two load instructions
- First use by add
- ing
- Branch instructions left out for clarity

In-order will stall here

Could get more MHP if we prioritize these instead



time

Out-of-order

on-hold

In-order
stall-on-use

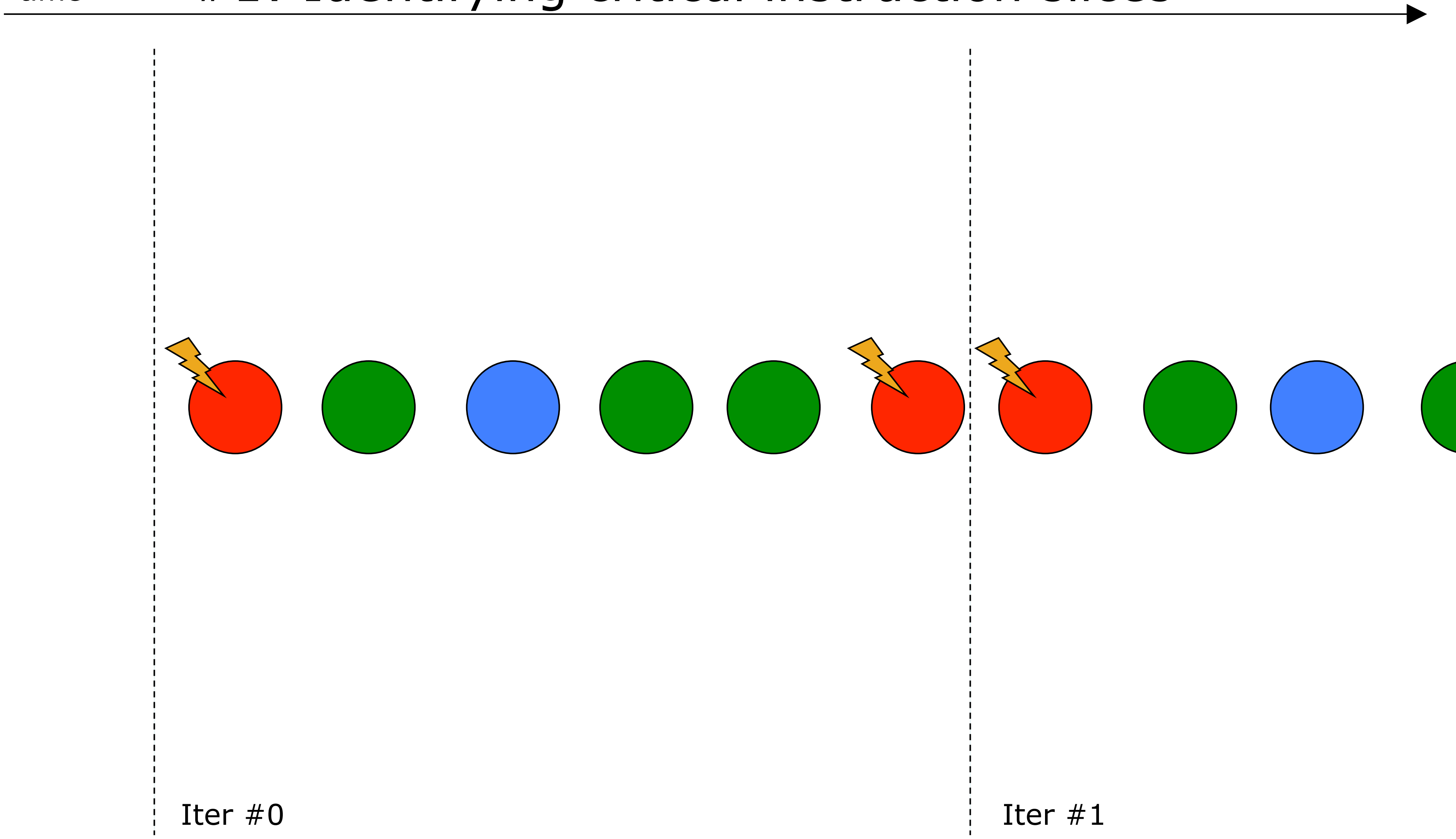
OoO: High complexity to get MLP

IO: Stalls on first use

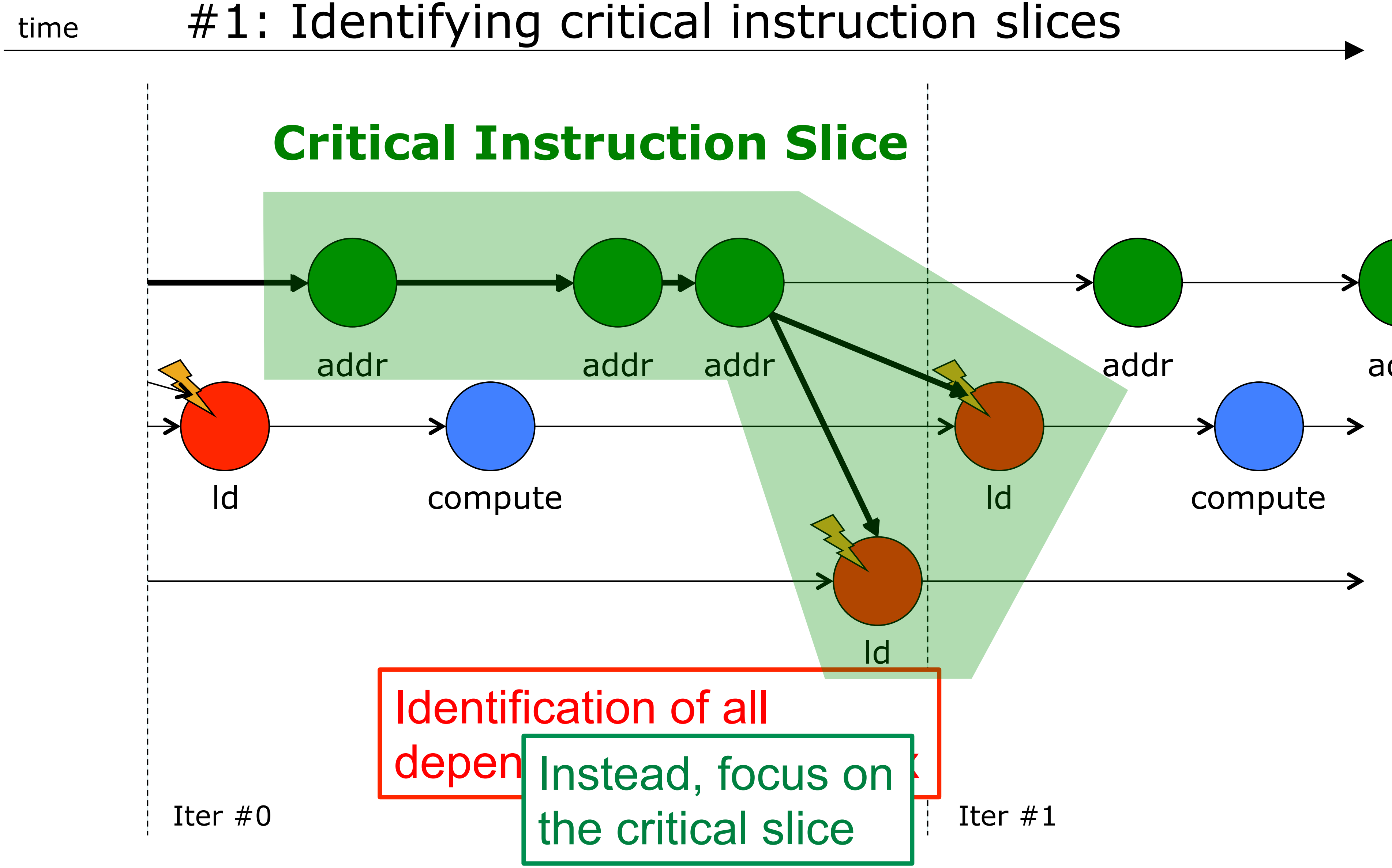
Two key LSC techniques:

1) Identify critical instruction slices
2) Bypass to increase MHP and performance

#1: Identifying critical instruction slices



#1: Identifying critical instruction slices



Identification of all
depen

Instead, focus on the critical slice

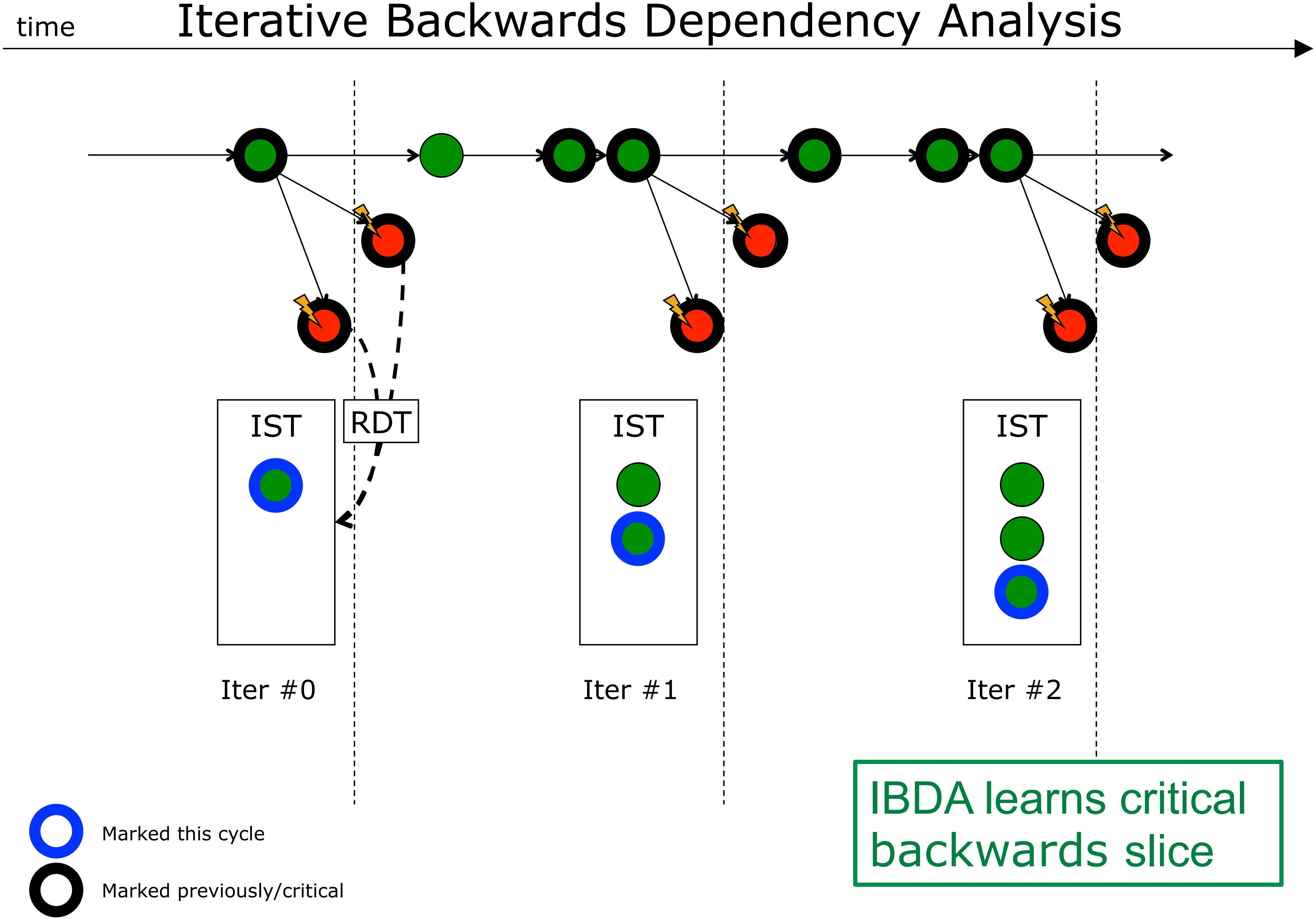
Iterative Backward Dependency Analysis

- Learning critical slices
  1. Start with load and store addresses
  2. IBDA to learn address generating instructions

- IST – Instruction Slice Table
  - Tracks critical instructions
  - Enables bypassing for MHP

- RDT – Register Dependency Table
  - Maps registers to instruction producers
  - Enables backwards dependency analysis

Iterative Backwards Dependency Analysis



IBDA learns critical backwards slice

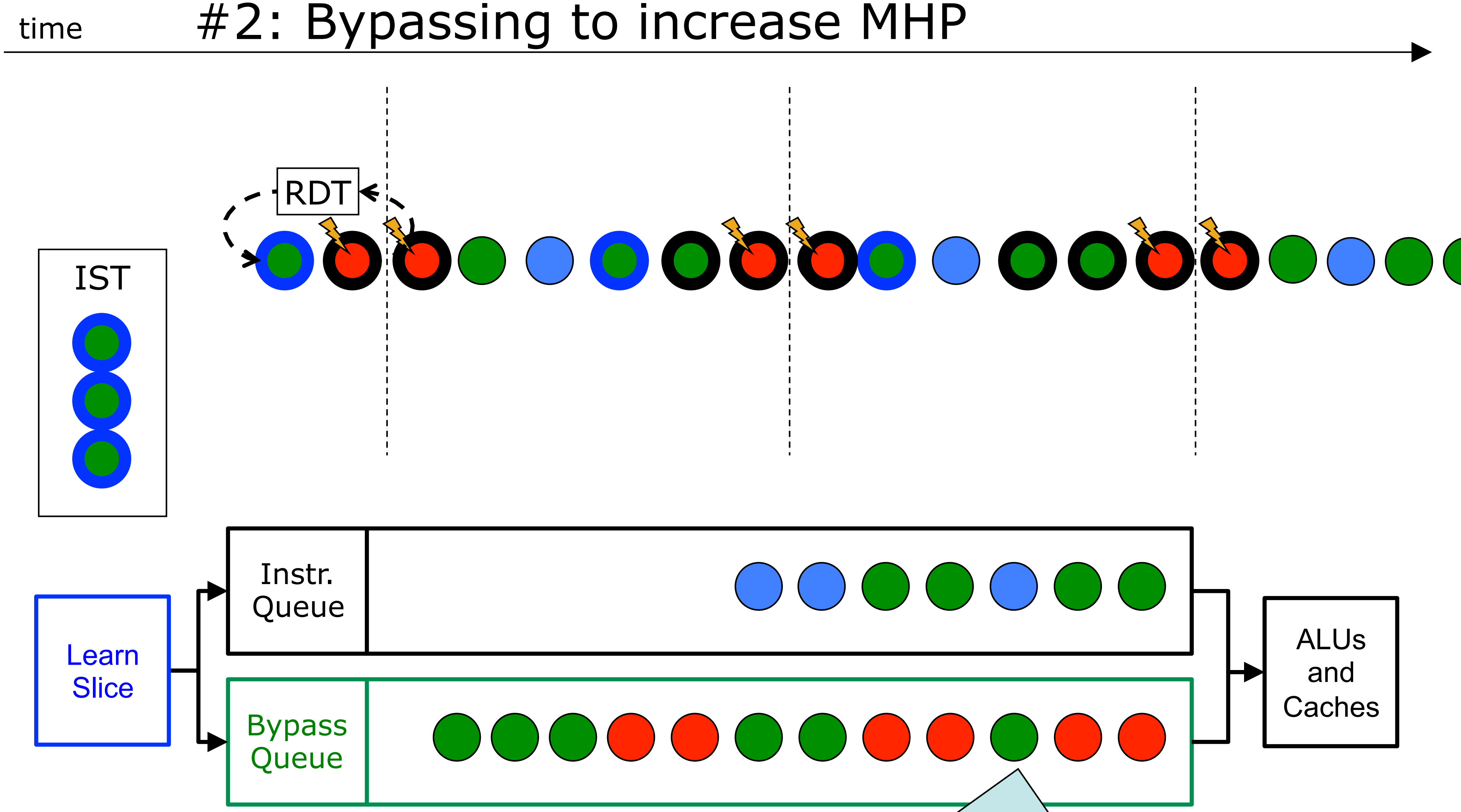
Bypassing

- Restricted out-of-order core

- Bypass queue:
  - Execute critical slice instructions earlier
  - Out-of-order with respect to regular queue
  - **In-order** within each queue
  - Loads can bypass store data (great for MHP)

- Do we have memory dependence violations?
  - Address computations always marked for bypass
  - Address computations **execute in program order**
  - Guarantees correct memory ordering
    (store buffer knows addresses)

#2: Bypassing to increase MHP



○ Marked this cycle

● Marked previously/critical

The critic
the comp

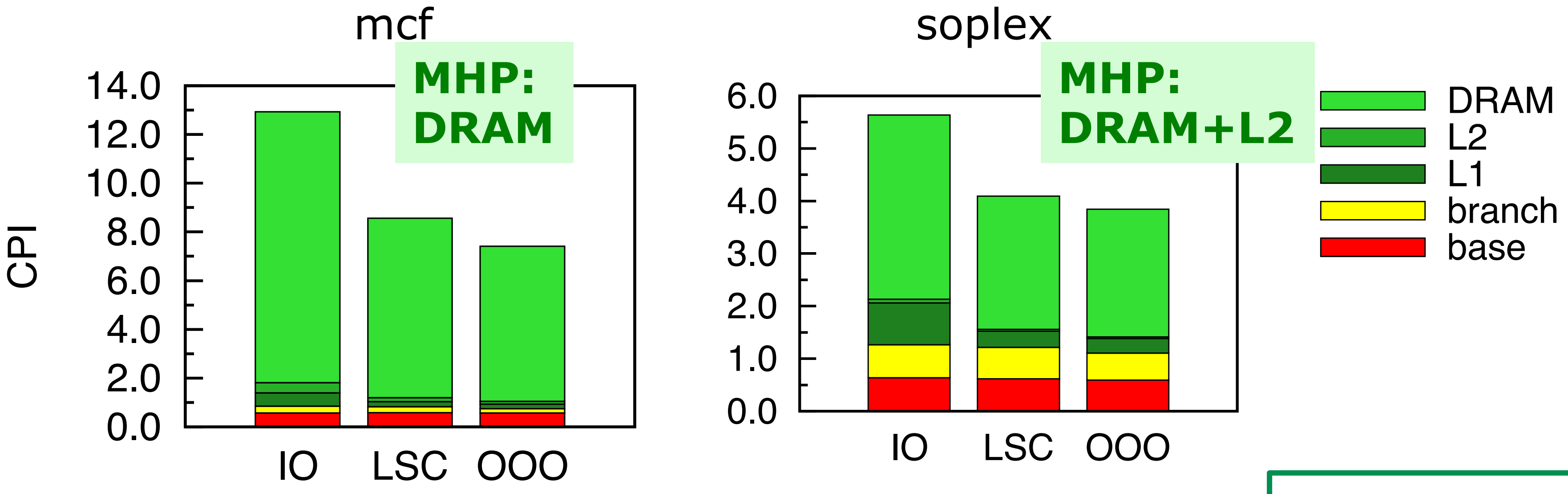
First learned address gen. in bypass

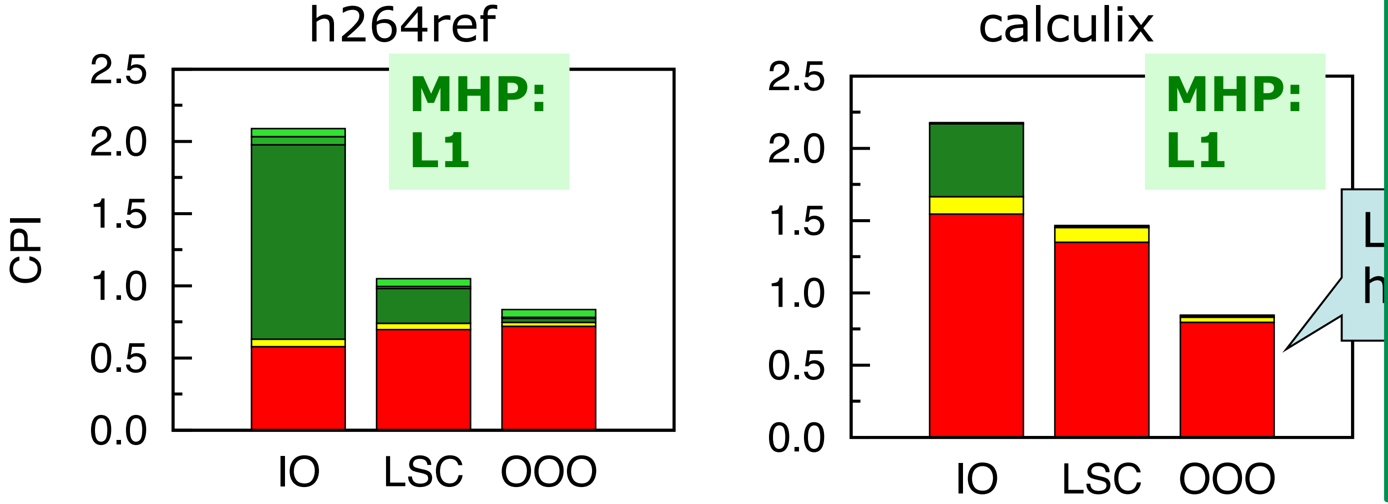
Experimental Setup

- Sniper multi-core simulator
  - ARM Cortex-A7-like configuration
  - 32KB L1s, 512KB L2, L1D prefetcher
  - 28nm (CACTI 6.5), 2.0GHz
- SPEC CPU2006 representative 750M instruction SimPoints and SPEC OMP and NPB representatives used

In-order	**LSC**	**OOO**
Stall on use	Restricted out-of-order	Full out-of-order
16-entry queue	32-entry IQ bypass queue, scoreboard	32-entry ROB and scheduler/issue queue
2-wide issue	2-wide issue	2-wide dispatch
ARM Cortex-A7-like	15% area overhead	155% area overhead (ARM Cortex-A9-like)

LSC Performance





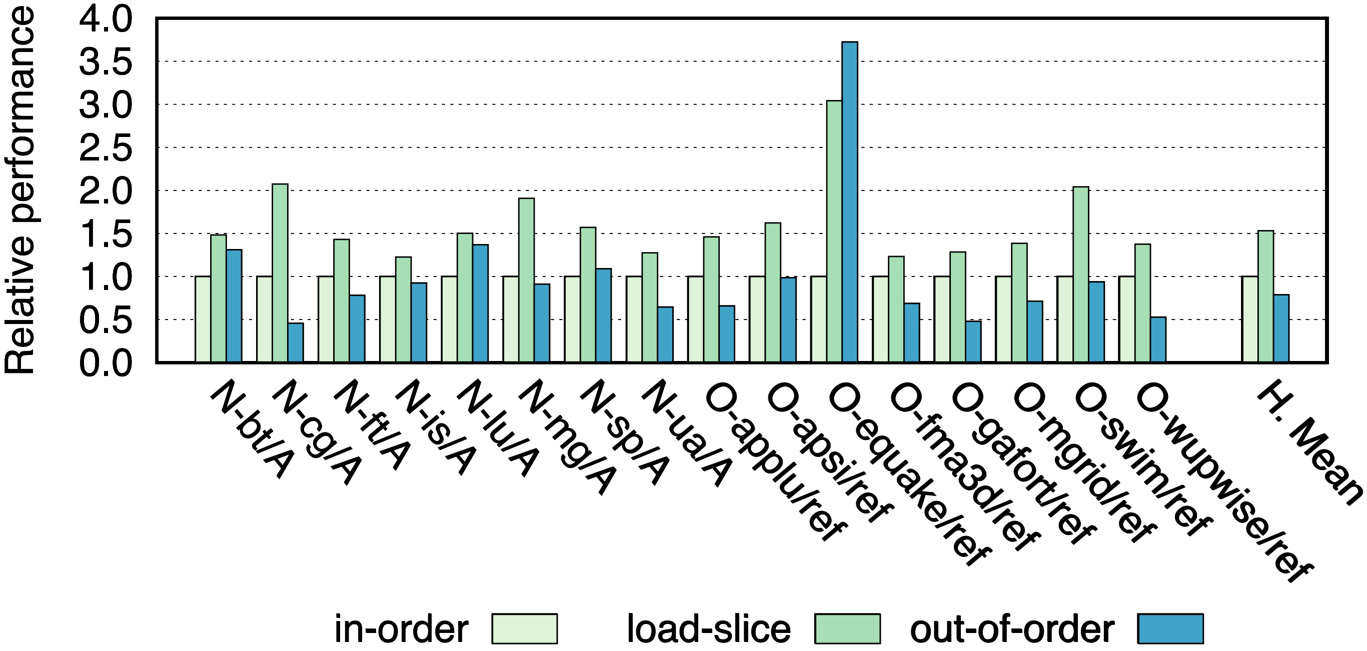
Average: 53% improvement in-order, within 25% of OoO;

Better energy/ area efficiency overall

LSC Many-Core Performance

	Power (W)	Area (mm^2)	Cores
Max	**45.0**	**350**	-
In-order	25.5	344	105
LSC	25.3	322	98
Out-of-Order	44.0	140	32

LSC has almost a 2x performance benefit over an out-of-order design



Conclusion

- MHP: an opportunity for better in-order performance

Load Slice Core

- Identify critical slices:
  - Backwards with IBDA
  - Learn across iterations
- Bypass critical instructions
  - Simple queue

- More performance through increased MHP:
  - Single-core: within 25% of OoO
  - Multicore: nearly 2x for area/power-limited designs

The Load Slice Core Microarchitecture

Trevor E. Carlson, Uppsala University

Wim Heirman, Intel

Osman Allam, Ghent University

Stefanos Kaxiras, Uppsala University

Lieven Eeckhout, Ghent University