

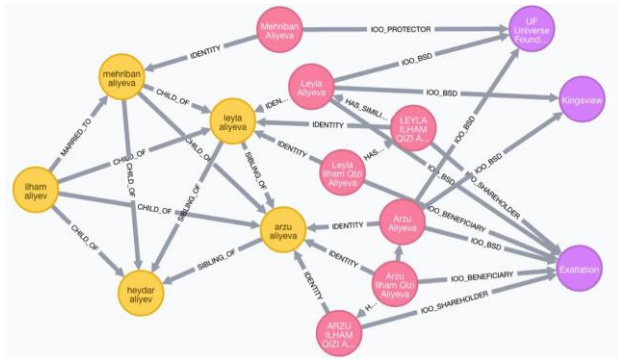
Freeway: Maximizing MLP for Slice-Out-of-Order Execution

Rakesh Kumar, Mehdi Alipour, David Black-Schaffer

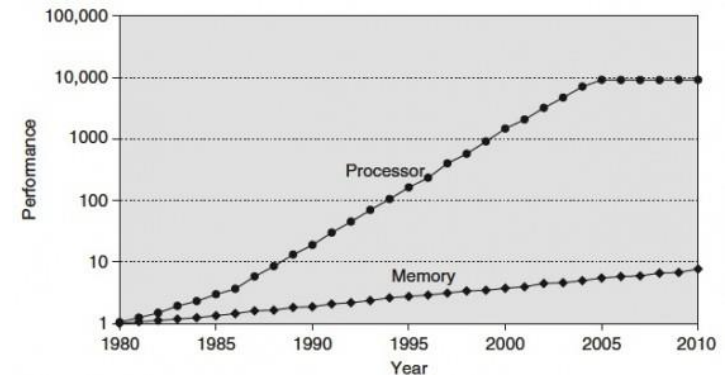


The Memory Bottleneck

Growing data sets



Memory is slow

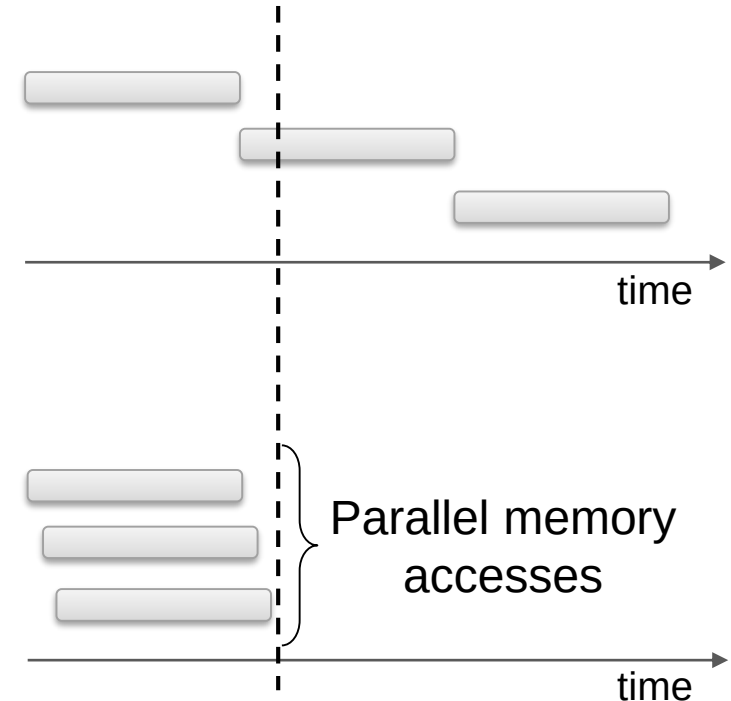
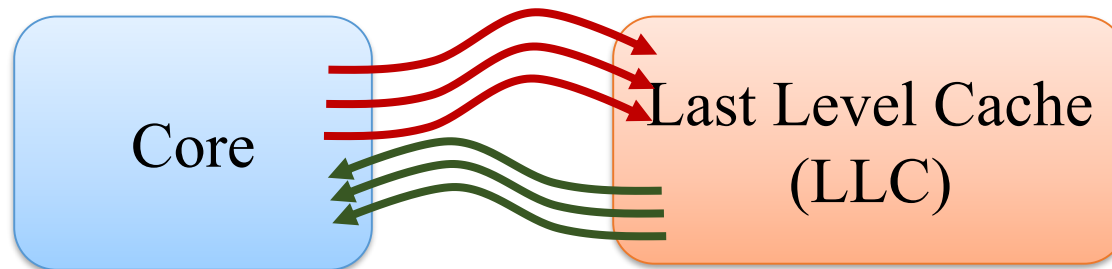
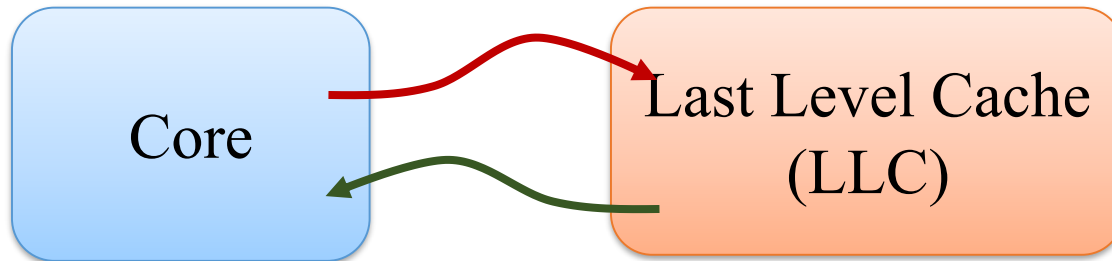


Memory pressure is growing:

- Multiple cores
- Accelerators

Memory subsystem dictates the overall system performance

Overcoming the Bottleneck



Memory Level Parallelism (MLP) is effective
in hiding memory latency

MLP in Out-of-Order Cores

OoO Core

I0 ld r1=M[r7]
I1 add r2=r1+1
I2 add r3=r3+1
I3 ld r4=M[r3]



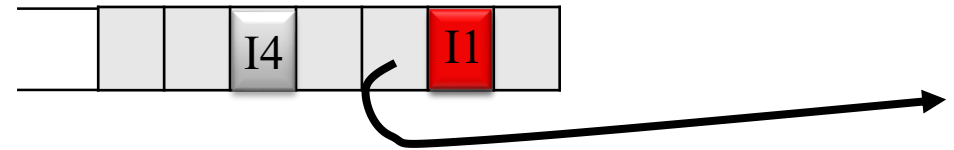
Instruction Queue (CAM)



MLP in Out-of-Order Cores

OoO Core

Instruction Queue (CAM)



I0 ld r1=M[r7]
I1 add r2=r1+1
I2 add r3=r3+1
I3 ld r4=M[r3]

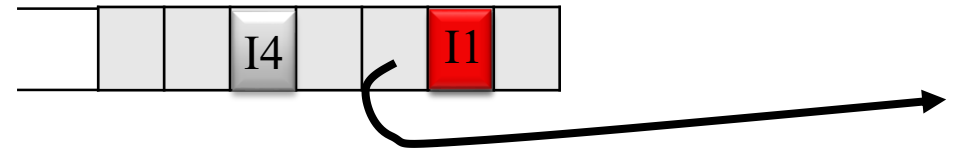
OoO cores issue ready instructions
by skipping the stalled one

MLP in Out-of-Order Cores

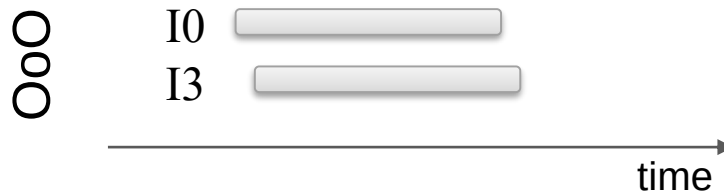
I0 ld r1=M[r7]
 I1 add r2=r1+1
 I2 add r3=r3+1
 I3 ld r4=M[r3]

OoO Core

Instruction Queue (CAM)



MLP



Power



MLP in In-Order Cores

IO Core

I0 ld r1=M[r7]
I1 add r2=r1+1
I2 add r3=r3+1
I3 ld r4=M[r3]



Instruction Queue (FIFO)



MLP in In-Order Cores

IO Core

I0 ld r1=M[r7]
I1 add r2=r1+1

I2 add r3=r3+1
I3 ld r4=M[r3]

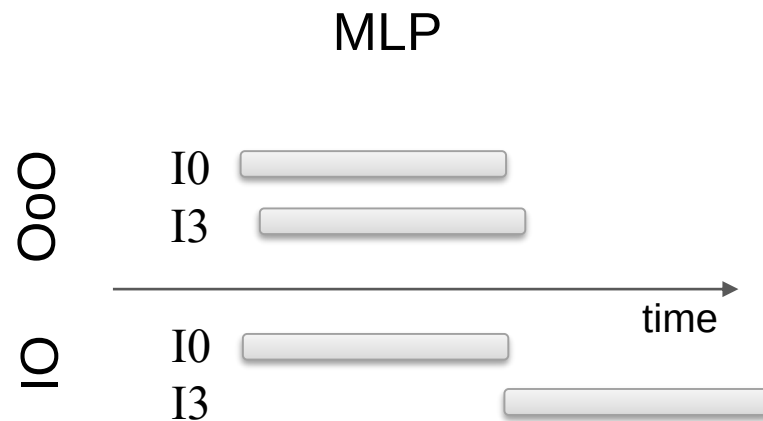
Instruction Queue (FIFO)



Ready instructions behind the first stalled instruction are also stalled (not issued)

MLP in Modern Cores

I0 ld r1=M[r7]
I1 add r2=r1+1
I2 add r3=r3+1
I3 ld r4=M[r3]



MLP in Modern Cores

The Big Question

Can we achieve MLP benefits of an OoO core with the power cost of an IO core?



A Step Forward

Load Slice Core [ISCA'15]

Slice-Out-of-Order Execution: Execute *slices* of MLP
generating instruction out-of-order only w.r.t. the rest of instructions

I0	ld	r1=M[r7]	} Slice 1
I1	add	r2=r1+1	

I2	add	r3=r3+1	} Slice 2
I3	ld	r4=M[r3]	

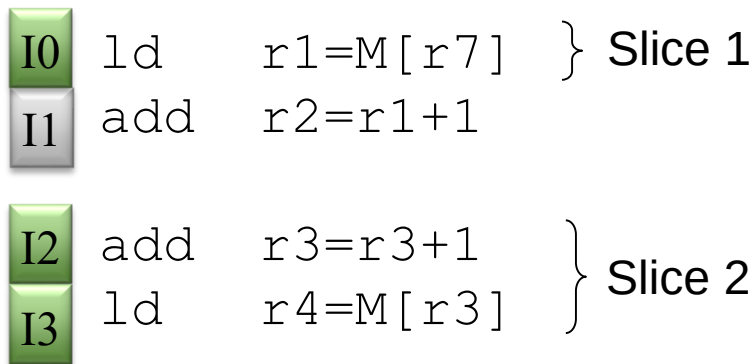
Slice: sequence of address generating instructions leading up to a load or store

A Step Forward

Load Slice Core [ISCA'15]

Slice-Out-of-Order Execution: Execute *slices* of MLP
generating instruction out-of-order only w.r.t. the rest of instructions

Load Slice Core (LSC)



Instruction Queue (FIFO)



Bypass Queue (FIFO)

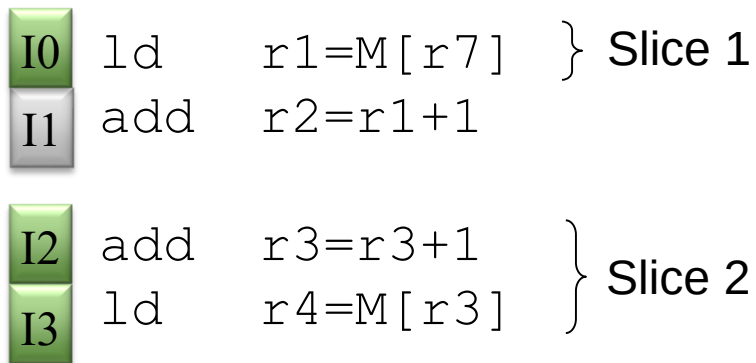


A Step Forward

Load Slice Core [ISCA'15]

Slice-Out-of-Order Execution: Execute *slices* of MLP
generating instruction out-of-order only w.r.t. the rest of instructions

Load Slice Core (LSC)



Instruction Queue (FIFO)



Bypass Queue (FIFO)

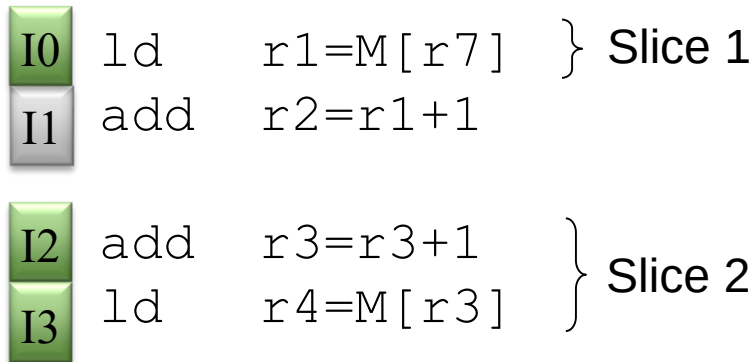


A Step Forward

Load Slice Core [ISCA'15]

Slice-Out-of-Order Execution: Execute *slices* of MLP
generating instruction out-of-order only w.r.t. the rest of instructions

Load Slice Core (LSC)



Instruction Queue (FIFO)



Bypass Queue (FIFO)

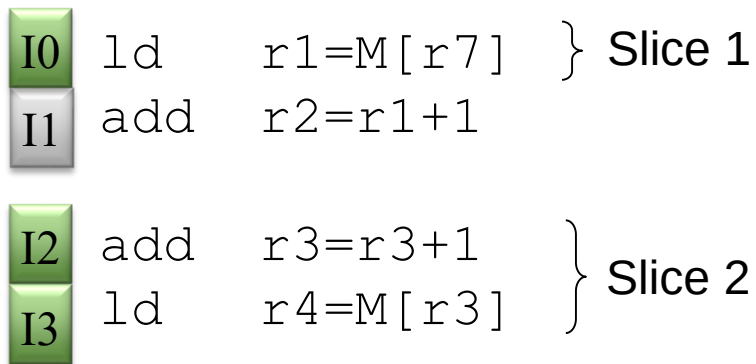


A Step Forward

Load Slice Core [ISCA'15]

Slice-Out-of-Order Execution: Execute *slices* of MLP
generating instruction out-of-order only w.r.t. the rest of instructions

Load Slice Core (LSC)



Instruction Queue (FIFO)



Bypass Queue (FIFO)

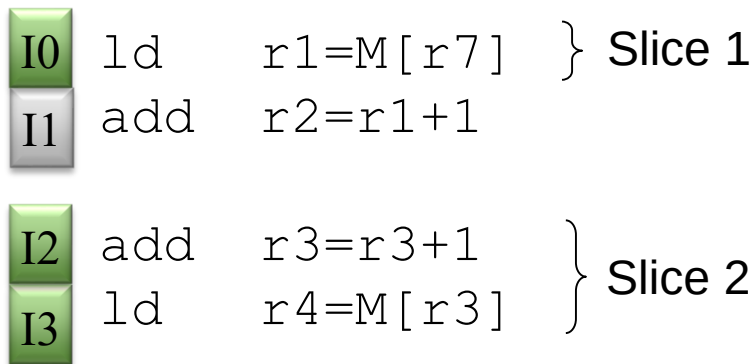


A Step Forward

Load Slice Core [ISCA'15]

Slice-Out-of-Order Execution: Execute *slices* of MLP
generating instruction out-of-order only w.r.t. the rest of instructions

Load Slice Core (LSC)



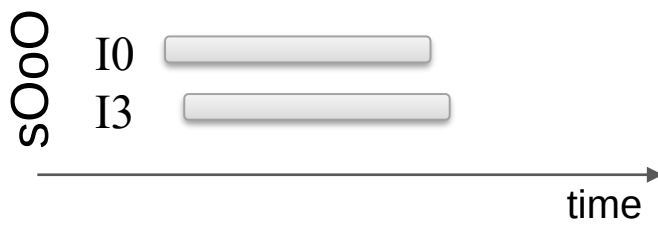
Instruction Queue (FIFO)



Bypass Queue (FIFO)



MLP



Power

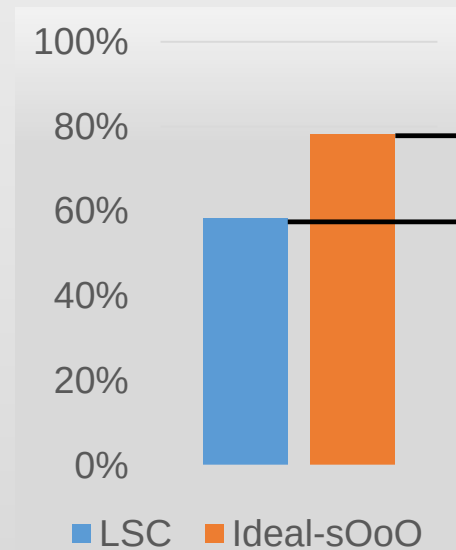


sOoO (LSC) Evaluation

Overhead over in-order core

Area : 15% Power: 22%

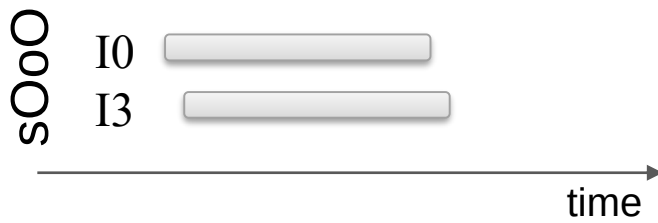
Performance



19% performance opportunity

MLP

Power



Summarizing State-of-the-Art

	<u>MLP</u>	<u>Power</u>
In-order cores		
Out-of-order cores		
Slice-out-of-order cores		

Our Goal

MLP

Power

In-order cores



Out-of-order cores

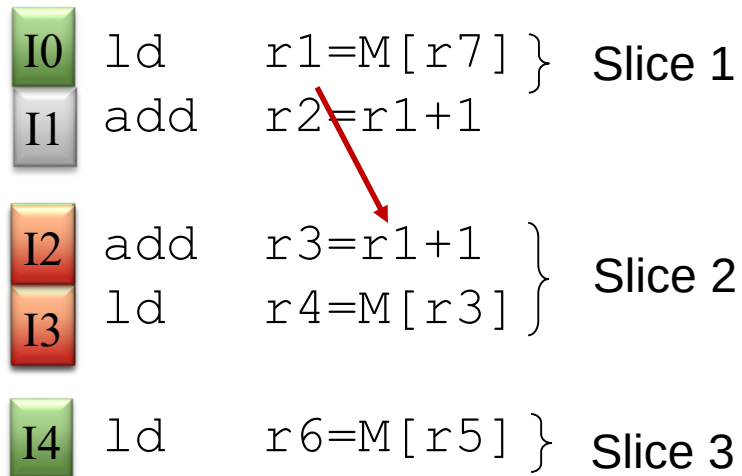


Slice-out-of-order cores



Analyzing Load Slice Core

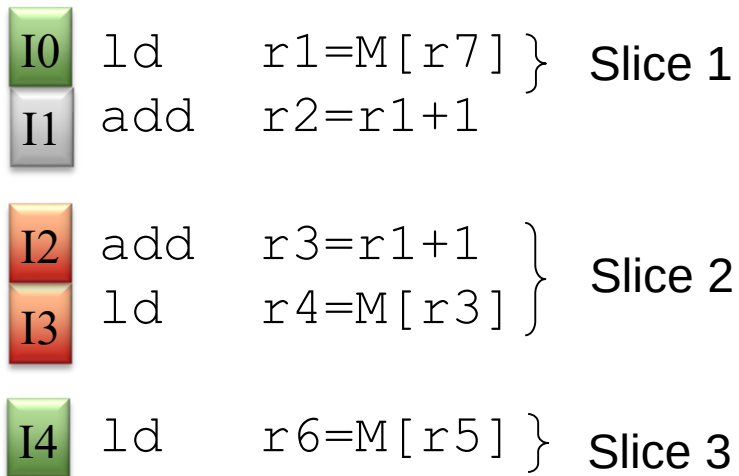
Observation: Dependent slices hurt MLP.



Slices that depends on the load instruction of an older slice

Analyzing Load Slice Core

Observation: Dependent slices hurt MLP.



Instruction Queue (FIFO)

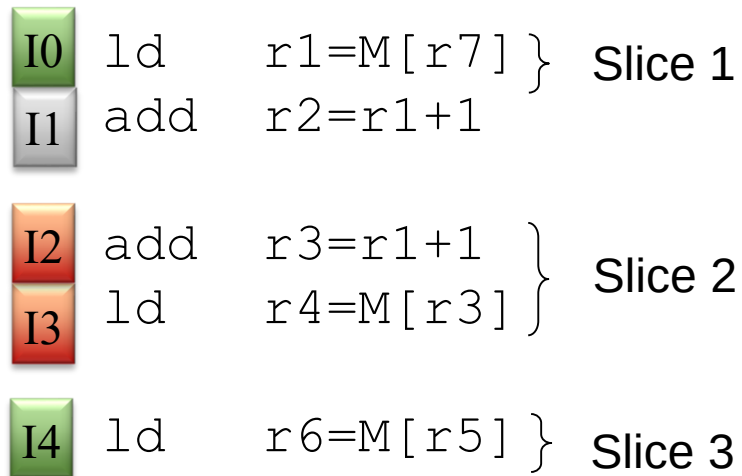


Bypass Queue (FIFO)



Analyzing Load Slice Core

Observation: Dependent slices hurt MLP.



Instruction Queue (FIFO)

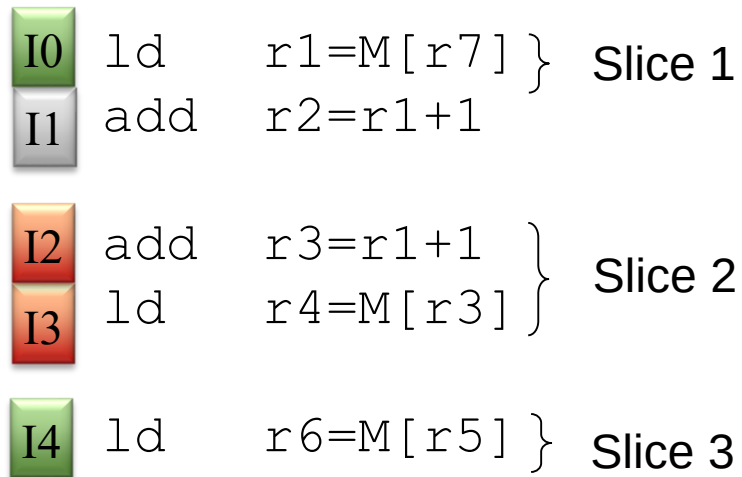


Bypass Queue (FIFO)



Analyzing Load Slice Core

Observation: Dependent slices hurt MLP.



Instruction Queue (FIFO)



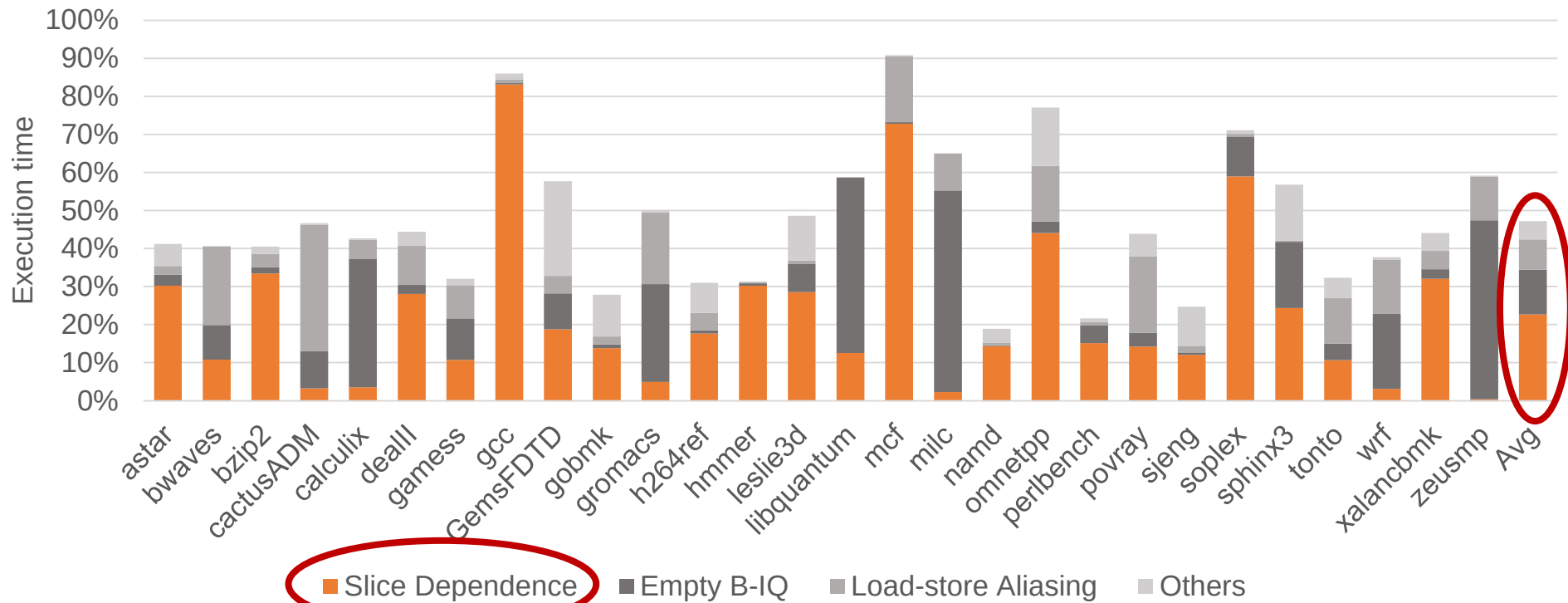
Bypass Queue (FIFO)



Blocked independent slice

Load Slice Core Stalls Quantification

Sources of stalls in the bypass queue



65% of the avoidable stalls are caused by dependent slices

Towards Unobstructed MLP Extraction

Problem: Dependent slices block subsequent independent slices and limit MLP

Idea: Steer dependent slices to a FIFO yielding queue (Y-IQ).

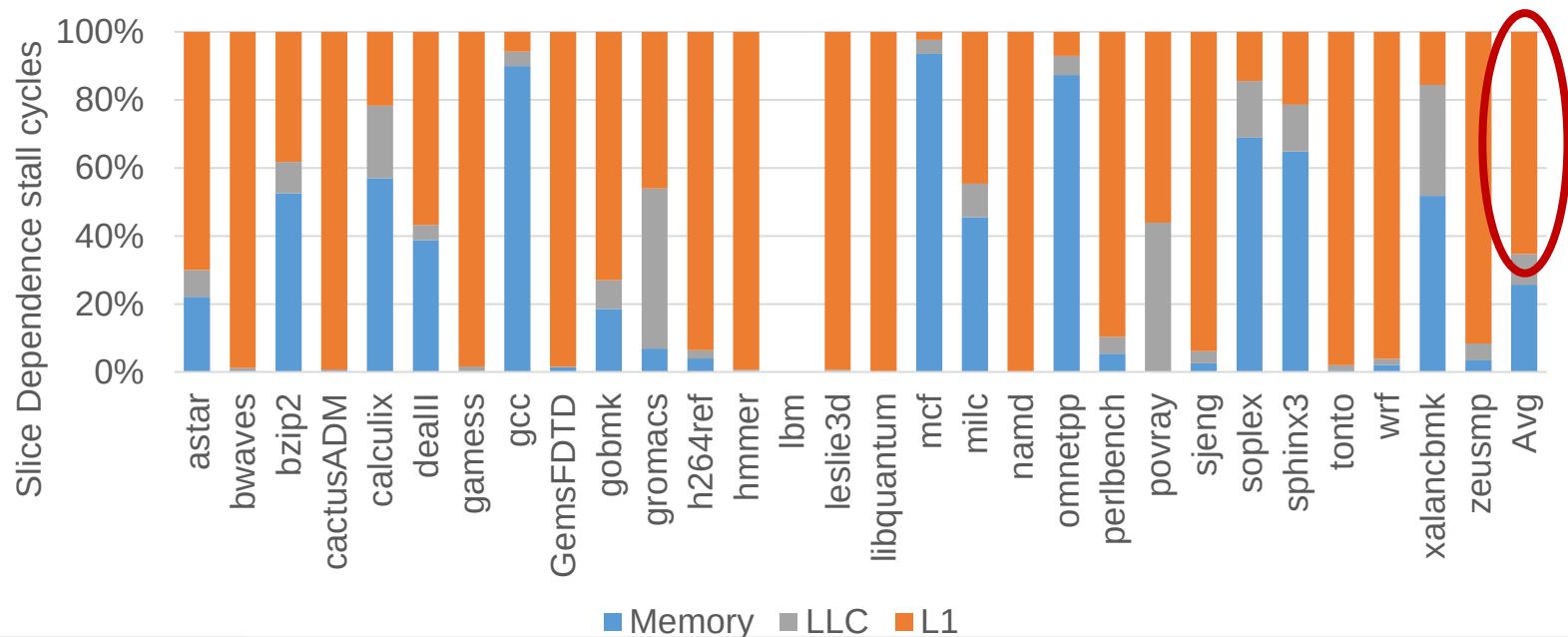
Challenges:

- Which dependent slices to steer: all, only a subset?
- Is a single Y-IQ enough: wouldn't dependent slices block each other?

Which Dependent Slices Go To Y-IQ?

Intuitively: Slices that block for long intervals

- i.e. producers hit in LLC or memory

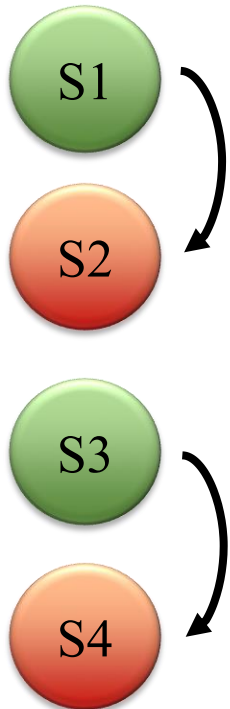


Most stalls due to short L1 hits: need to steer all the dependent slices to Y-IQ.

Is a Single Yielding Queue Enough?

Problem: Y-IQ will block if younger slices become ready before the older slices.

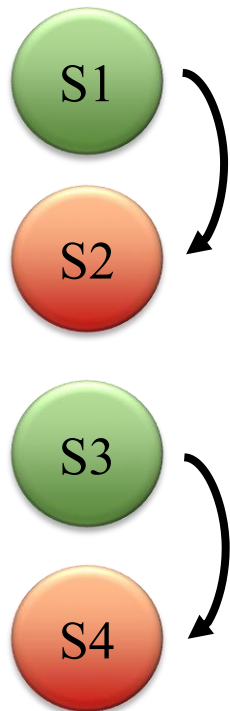
Scenario:



Is a Single Yielding Queue Enough?

Problem: Y-IQ will block if younger slices become ready before the older slices.

Scenario:



Instruction Queue (FIFO)



Bypass Queue (FIFO)



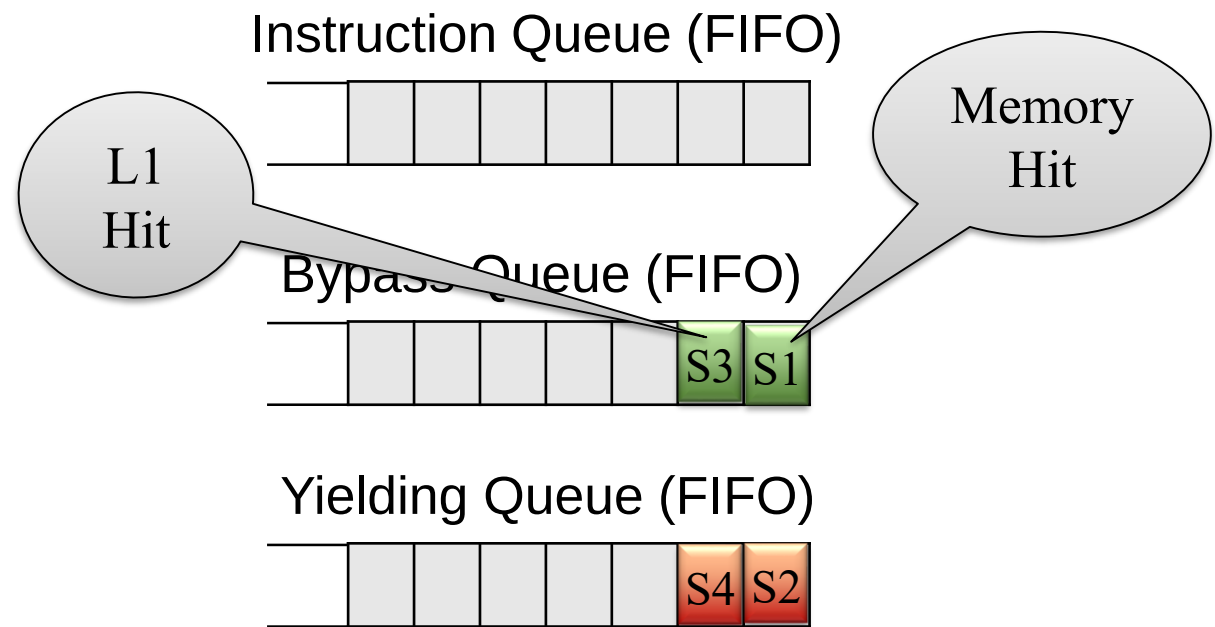
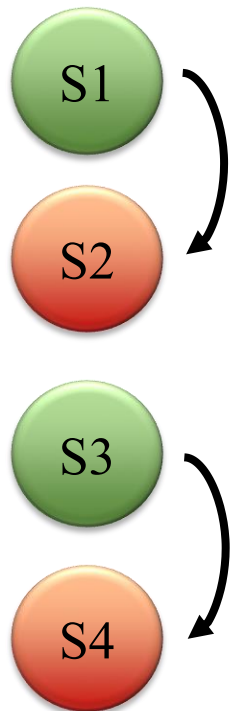
Yielding Queue (FIFO)



Is a Single Yielding Queue Enough?

Problem: Y-IQ will block if younger slices become ready before the older slices.

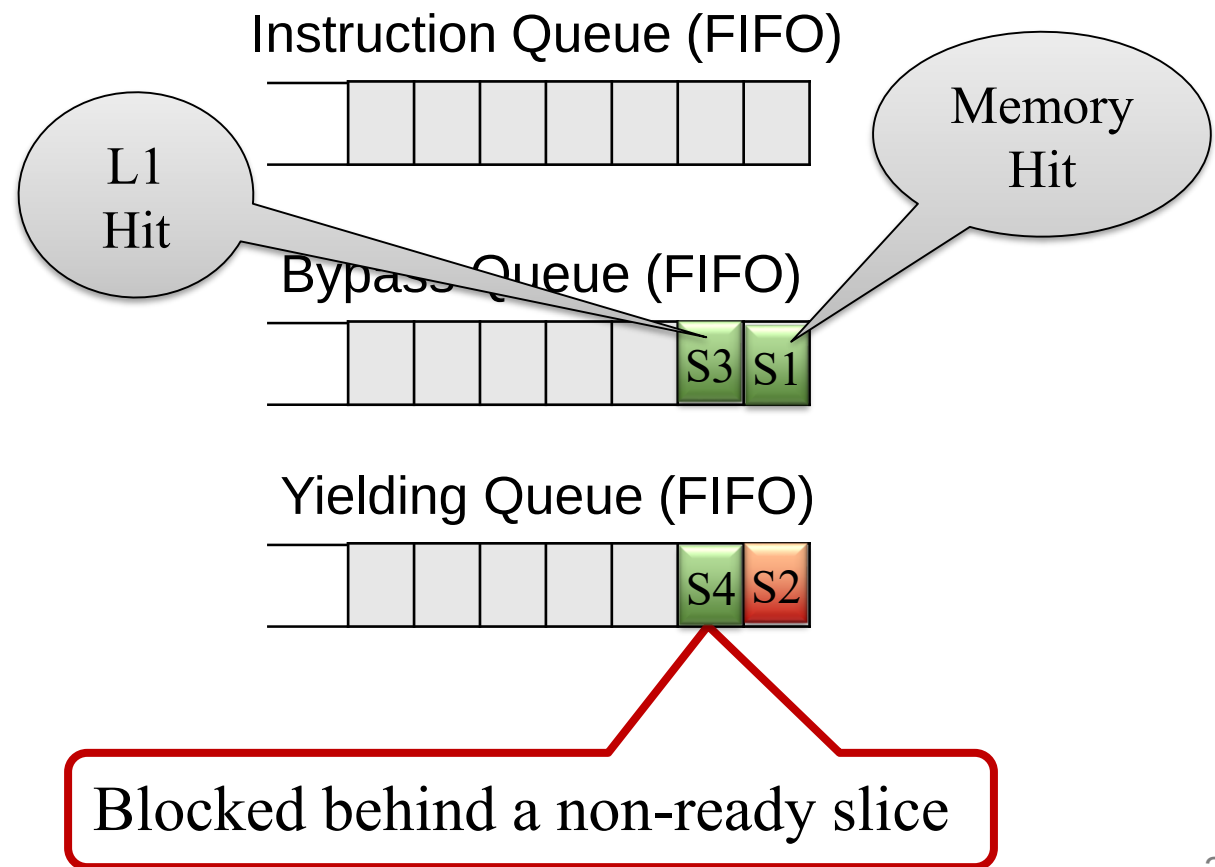
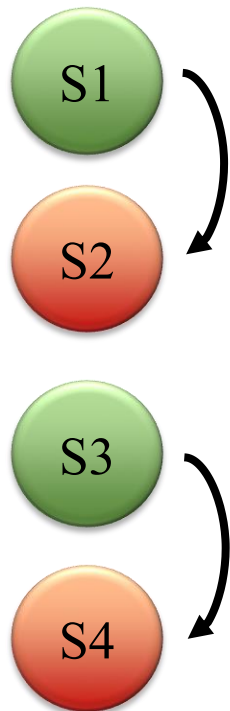
Scenario:



Is a Single Yielding Queue Enough?

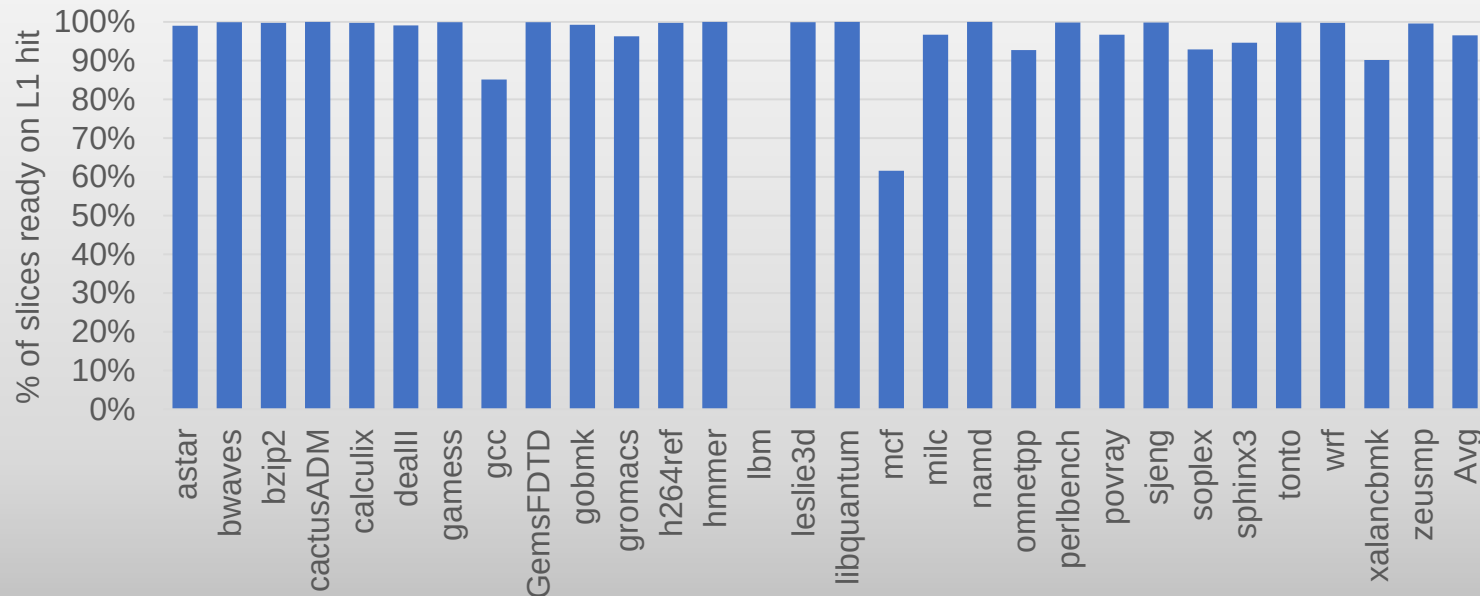
Problem: Y-IQ will block if younger slices become ready before the older slices.

Scenario:



Is a Single Yielding Queue Enough?

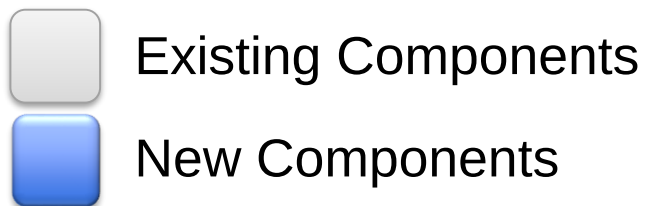
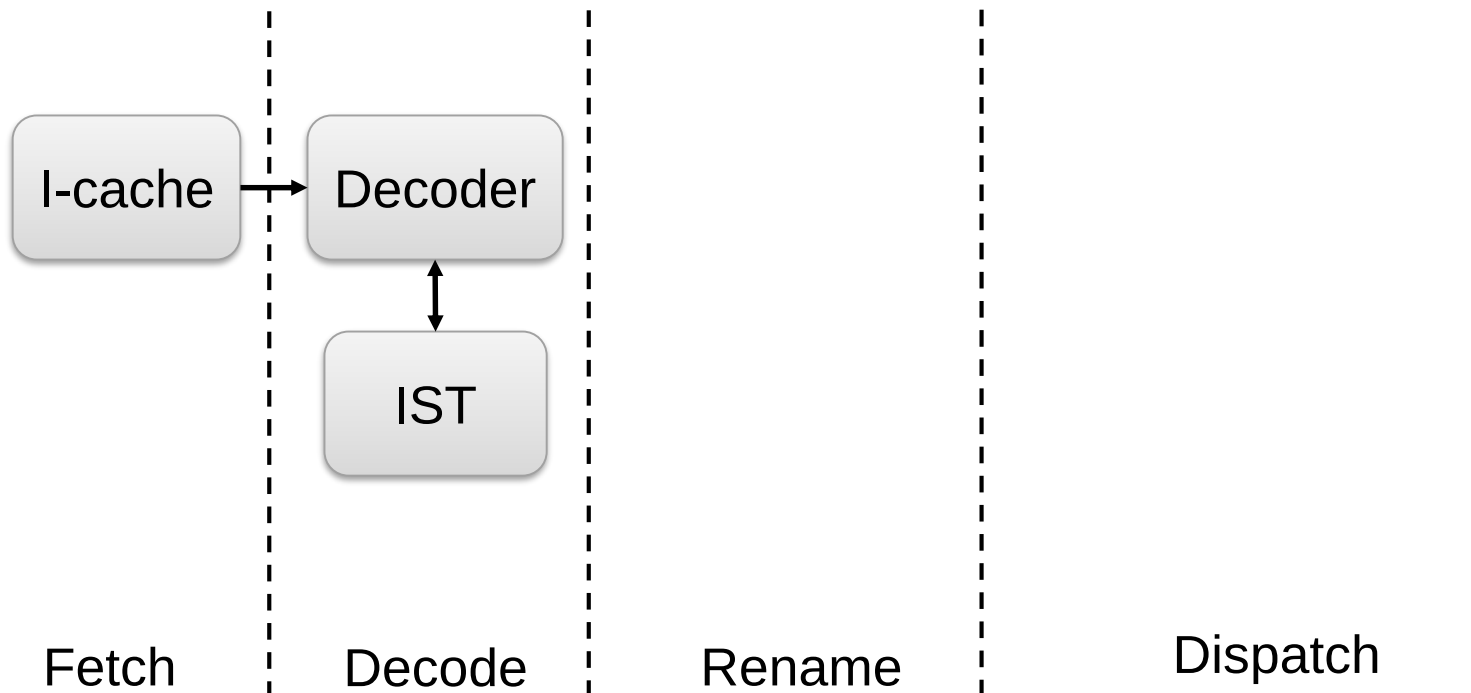
Independent slice hit location



Majority of independent slices hit at same cache level (L1), dependent slices usually become ready in program order

Single Y-IQ is sufficient as dependent slices usually don't block each other

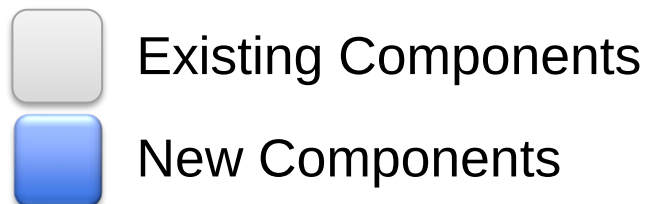
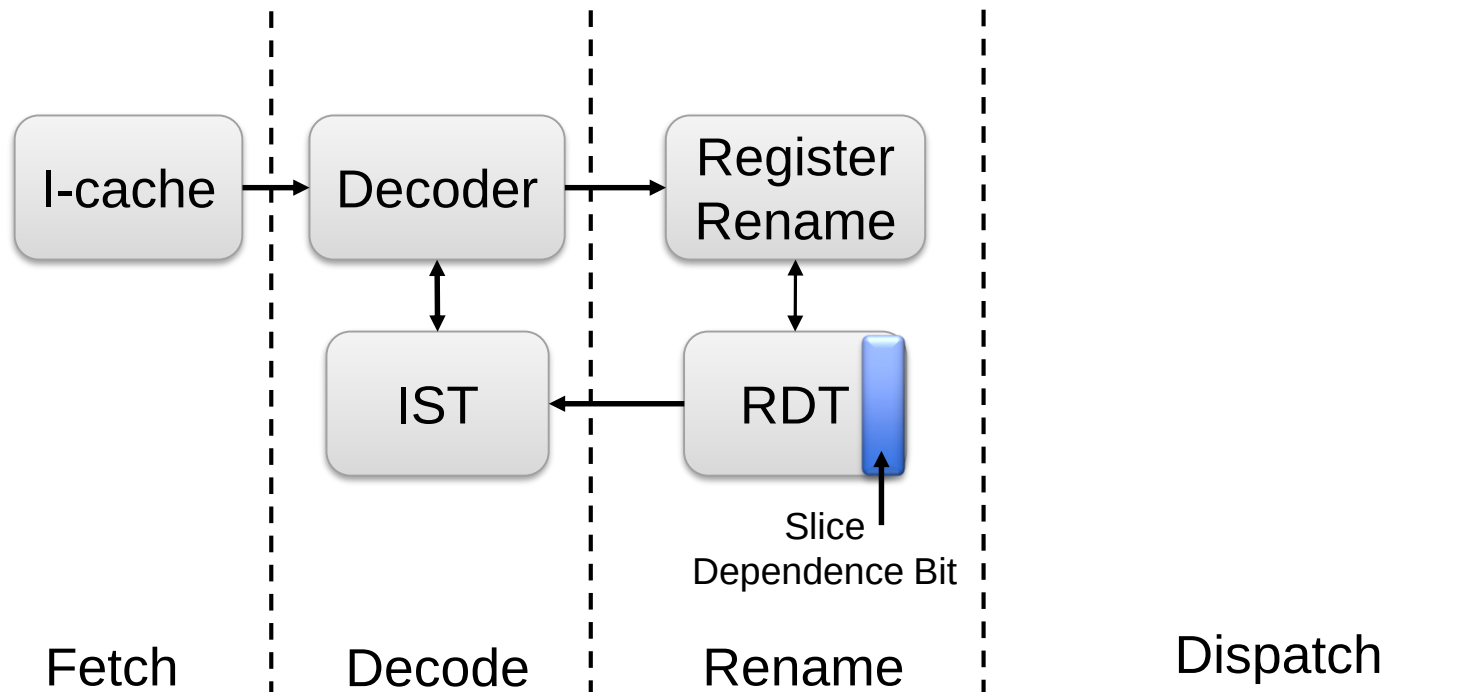
Freeway Microarchitecture



IST (Instruction Slice Table)

- Tracks memory slices
- Identifies if a (pre-)decoded instruction belongs to a memory slice

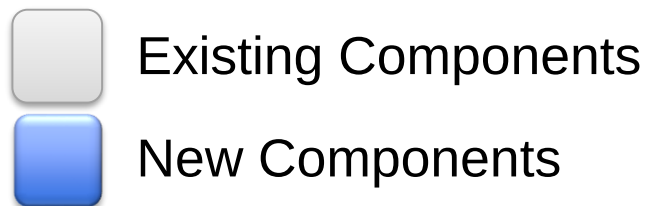
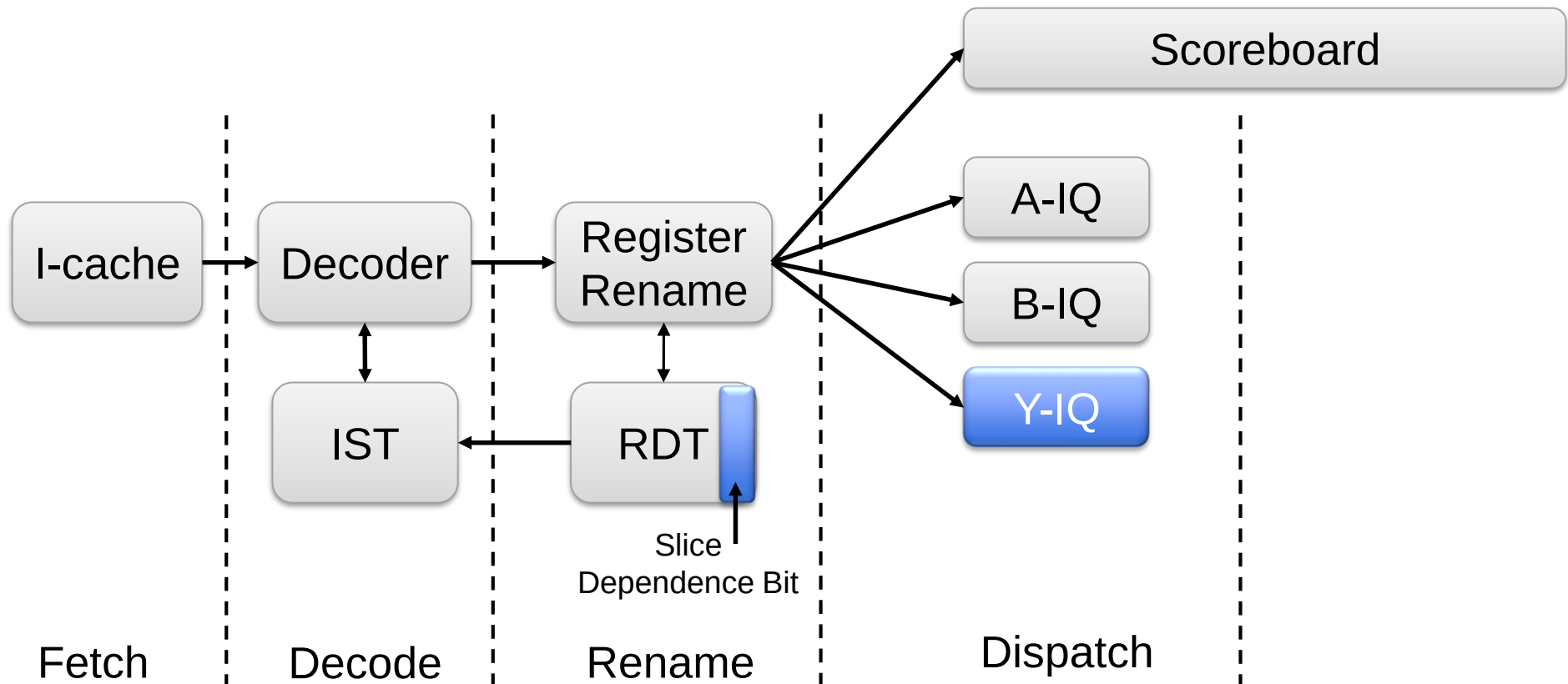
Freeway Microarchitecture



RDT (Register Dependence Table)

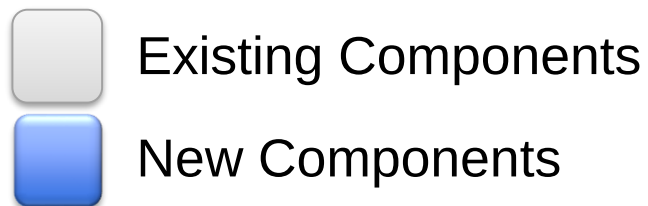
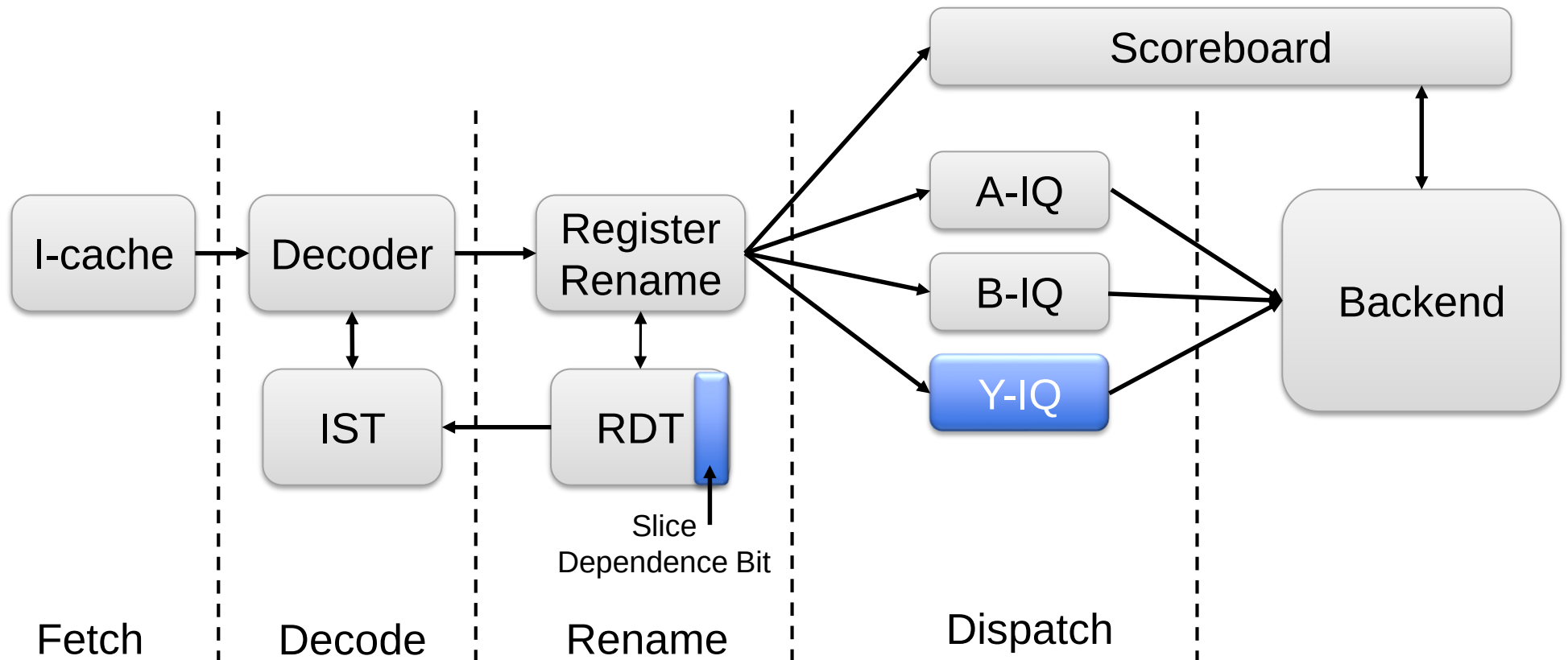
- Extended to propagate slice dependency information

Freeway Microarchitecture



Slice	Dependent	Queue
No	-	A-IQ
Yes	No	B-IQ
Yes	Yes	Y-IQ

Freeway Microarchitecture



Slice	Dependent	Queue
No	-	A-IQ
Yes	No	B-IQ
Yes	Yes	Y-IQ

Identifying Memory Slices

Iterative Backwards Dependence Analysis (IBDA) LSC[ISCA'15]

Decode

Rename

Instruction slice Table (IST)

Register Dependence Table (RDT)

Tag	
Tag	
Tag	
Tag	
Tag	

p1	
p2	
p3	
p4	
...	

I0 add r3=r2+1
I1 ld r4=M[r3]



Decode Stage:

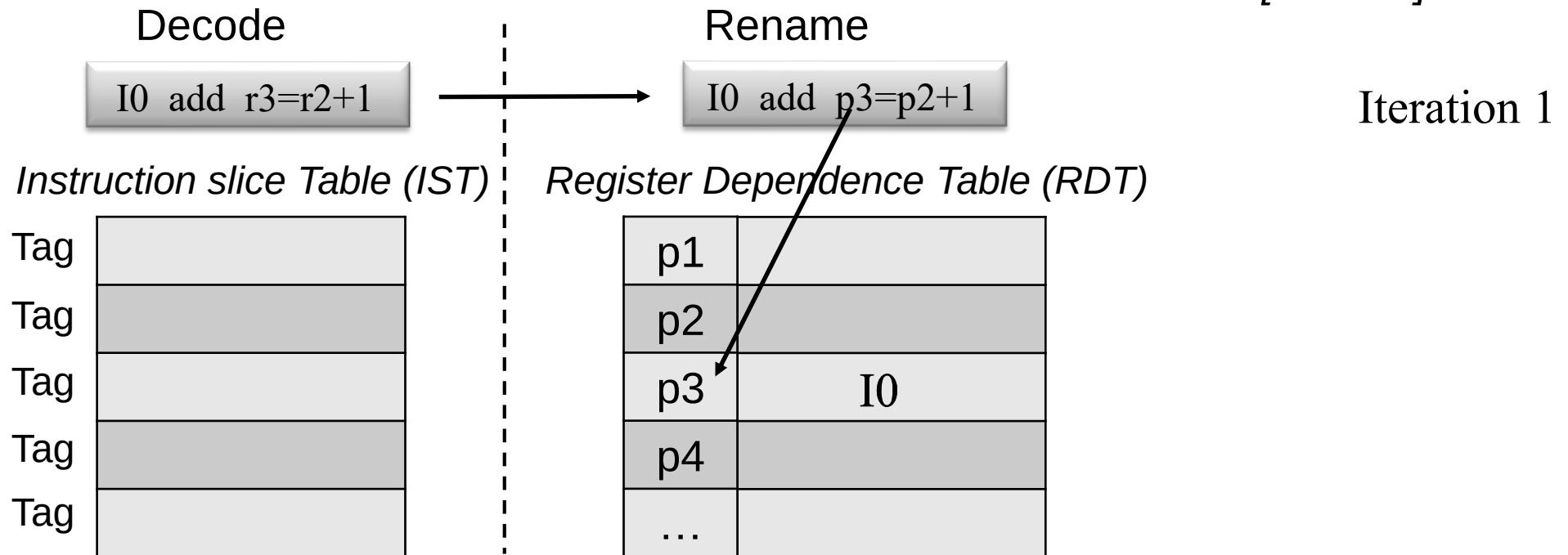
- PC in IST? Yes: Inst Slice; No: Not a slice

Rename Stage:

- Associate PhyReg to PC
- For slice inst, insert the producers' PC to IST

Identifying Memory Slices

Iterative Backwards Dependence Analysis (IBDA) LSC[ISCA'15]



I0 add r3=r2+1
I1 ld r4=M[r3]

Decode Stage:

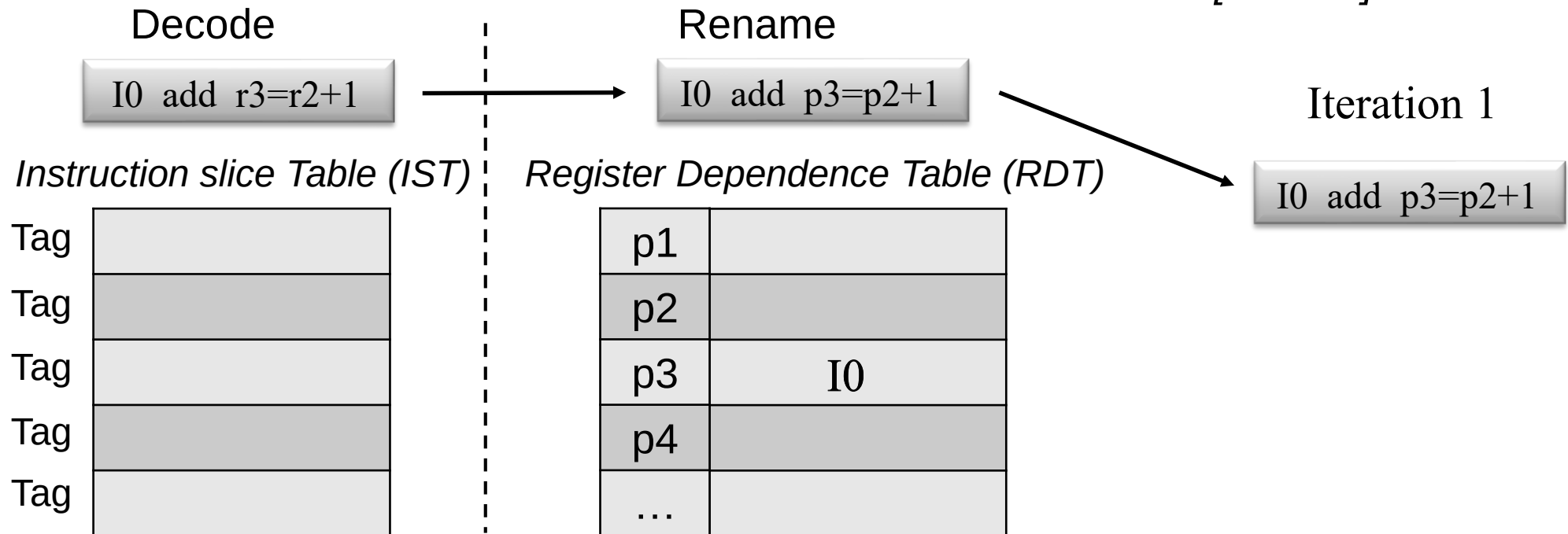
- PC in IST? Yes: Inst Slice; No: Not a slice

Rename Stage:

- Associate PhyReg to PC
- For slice inst, insert the producers' PC to IST

Identifying Memory Slices

Iterative Backwards Dependence Analysis (IBDA) LSC[ISCA'15]



I0 add r3=r2+1
I1 ld r4=M[r3]

Decode Stage:

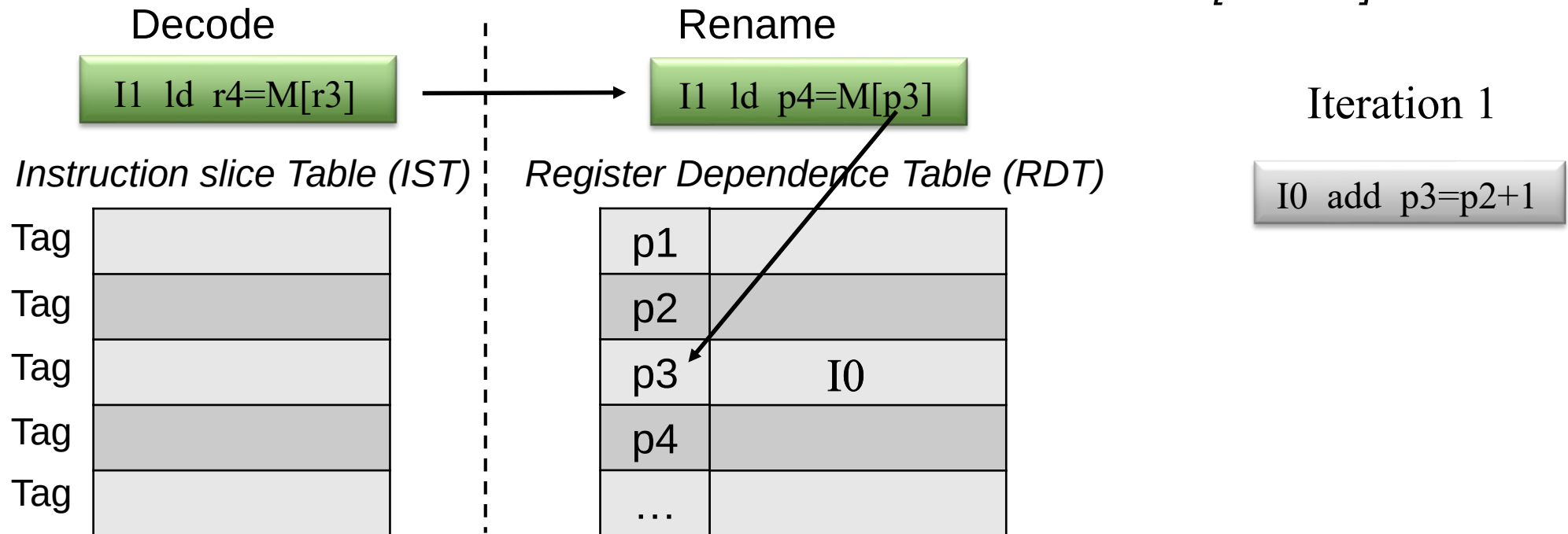
- PC in IST? Yes: Inst Slice; No: Not a slice

Rename Stage:

- Associate PhyReg to PC
- For slice inst, insert the producers' PC to IST

Identifying Memory Slices

Iterative Backwards Dependence Analysis (IBDA) LSC[ISCA'15]



Decode Stage:

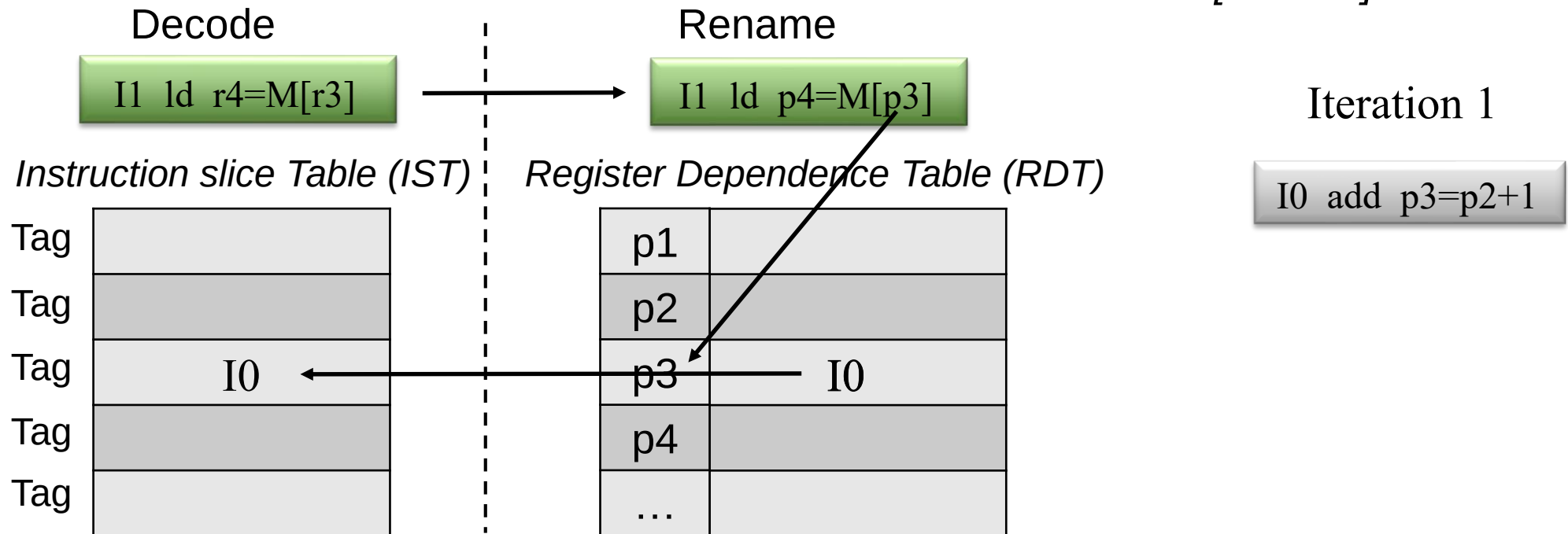
- PC in IST? Yes: Inst Slice; No: Not a slice

Rename Stage:

- Associate PhyReg to PC
- For slice inst, insert the producers' PC to IST

Identifying Memory Slices

Iterative Backwards Dependence Analysis (IBDA) LSC[ISCA'15]



I0 add r3=r2+1
 I1 ld r4=M[r3]

Decode Stage:

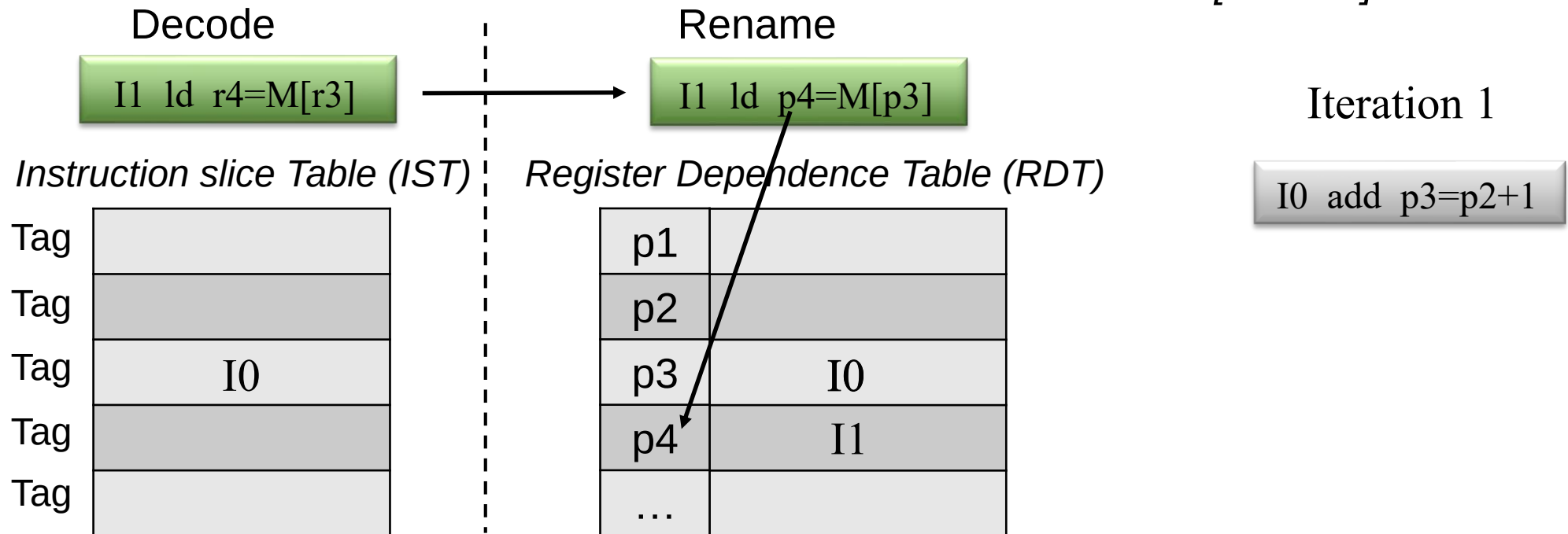
- PC in IST? Yes: Inst Slice; No: Not a slice

Rename Stage:

- Associate PhyReg to PC
- For slice inst, insert the producers' PC to IST

Identifying Memory Slices

Iterative Backwards Dependence Analysis (IBDA) LSC[ISCA'15]



I0 add r3=r2+1
 I1 ld r4=M[r3]

Decode Stage:

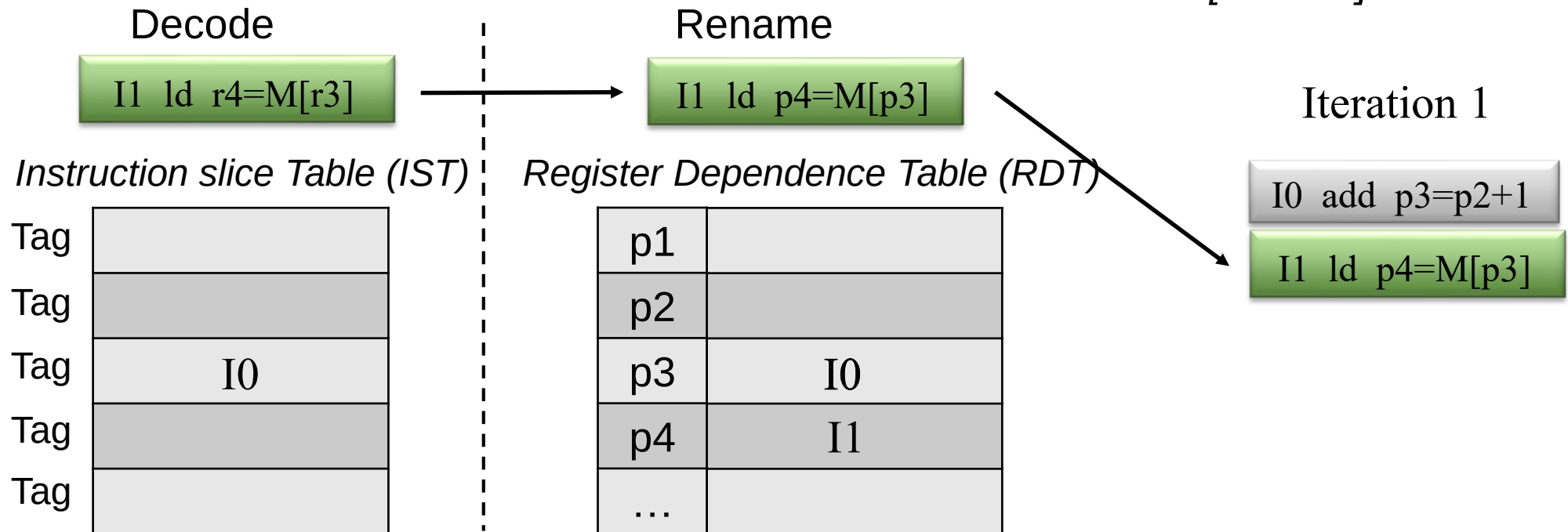
- PC in IST? Yes: Inst Slice; No: Not a slice

Rename Stage:

- Associate PhyReg to PC
- For slice inst, insert the producers' PC to IST

Identifying Memory Slices

Iterative Backwards Dependence Analysis (IBDA) LSC[ISCA'15]



I0 add r3=r2+1
I1 ld r4=M[r3]

Decode Stage:

- PC in IST? Yes: Inst Slice; No: Not a slice

Rename Stage:

- Associate PhyReg to PC
- For slice inst, insert the producers' PC to IST

Identifying Memory Slices

Iterative Backwards Dependence Analysis (IBDA) LSC[ISCA'15]

Decode

I0 add r3=r2+1

Instruction slice Table (IST)

Tag	
Tag	
Tag	I0
Tag	
Tag	

Rename

Register Dependence Table (RDT)

p1	
p2	
p3	I0
p4	I1
...	

Iteration 1

I0 add p3=p2+1

I1 ld p4=M[p3]

Iteration 2

I0 add r3=r2+1
I1 ld r4=M[r3]



Identifying Memory Slices

Iterative Backwards Dependence Analysis (IBDA) LSC[ISCA'15]

Decode

I0 add r3=r2+1

Instruction slice Table (IST)

Tag	
Tag	
Tag	I0
Tag	
Tag	

Rename

Register Dependence Table (RDT)

p1	
p2	
p3	I0
p4	I1
...	

Iteration 1

I0 add p3=p2+1

I1 ld p4=M[p3]

Iteration 2

I0 add r3=r2+1
I1 ld r4=M[r3]

Identifying Memory Slices

Iterative Backwards Dependence Analysis (IBDA) LSC[ISCA'15]

Decode

I0 add r3=r2+1

Instruction slice Table (IST)

Tag	
Tag	
Tag	I0
Tag	
Tag	

Rename

I0 add p3=p2+1

Register Dependence Table (RDT)

p1	
p2	
p3	I0
p4	I1
...	

Iteration 1

I0 add p3=p2+1

I1 ld p4=M[p3]

Iteration 2

I0 add p3=p2+1

I0 add r3=r2+1
I1 ld r4=M[r3]

Freeway: Identifying Dependent Slices

Rename

Register Dependence Table

p1		
p2		
p3		
p4		
p5		

Instruction reading this physical register
belongs to a dependent slice or not?

Freeway: Identifying Dependent Slices

Rename

Register Dependence Table

p1		0
p2		0
p3		0
p4		0
p5		0

I0	ld	p1=M[p2]
I1	add	p2=p1+1
I2	add	p3=p1+1
I3	ld	p4=M[p3]
I4	ld	p2=M[p5]

Freeway: Identifying Dependent Slices

Rename

Register Dependence Table

p1		0
p2		0
p3		0
p4		0
p5		0

0: Independent Slice

I0	ld	p1=M[p2]
I1	add	p2=p1+1
I2	add	p3=p1+1
I3	ld	p4=M[p3]
I4	ld	p2=M[p5]

Freeway: Identifying Dependent Slices

Rename

Register Dependence Table

p1		1
p2		0
p3		0
p4		0
p5		0

p1 consumers would be
dependent slice instructions

I0	ld	p1=M[p2]
I1	add	p2=p1+1
I2	add	p3=p1+1
I3	ld	p4=M[p3]
I4	ld	p2=M[p5]

Freeway: Identifying Dependent Slices

Rename

Register Dependence Table

p1		1
p2		0
p3		0
p4		0
p5		0

1: Dependent Slice

I0	ld	p1=M[p2]
I1	add	p2=p1+1
I2	add	p3=p1+1
I3	ld	p4=M[p3]
I4	ld	p2=M[p5]

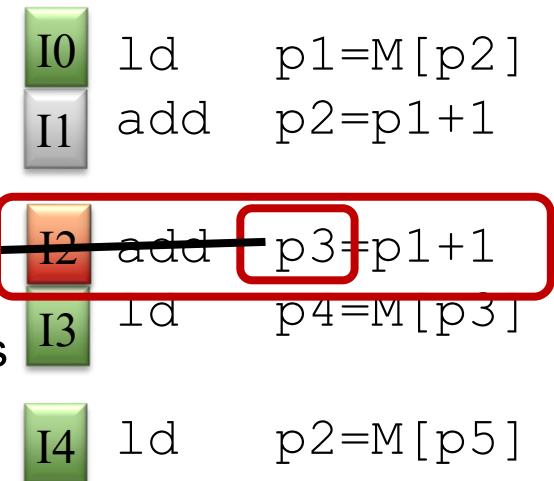
Freeway: Identifying Dependent Slices

Rename

Register Dependence Table

p1		1
p2		0
p3		1
p4		0
p5		0

p3 consumers would be
dependent slice instructions



Freeway: Identifying Dependent Slices

Rename

Register Dependence Table

p1		1
p2		0
p3		1
p4		0
p5		0

1: Dependent Slice

I0	ld	p1=M[p2]
I1	add	p2=p1+1
I2	add	p3=p1+1
I3	ld	p4=M[p3]
I4	ld	p2=M[p5]

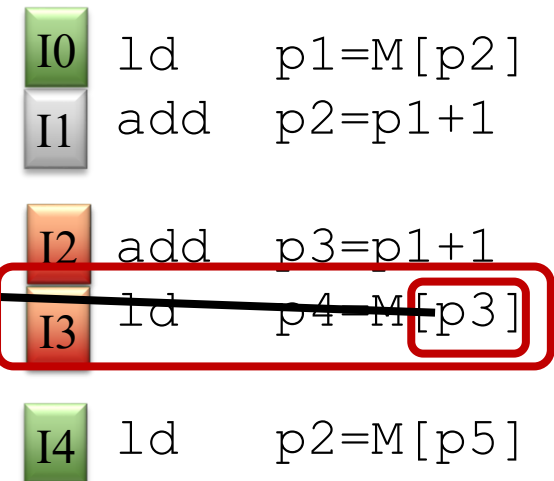
Freeway: Identifying Dependent Slices

Rename

Register Dependence Table

p1		1
p2		0
p3		1
p4		0
p5		0

1: Dependent Slice



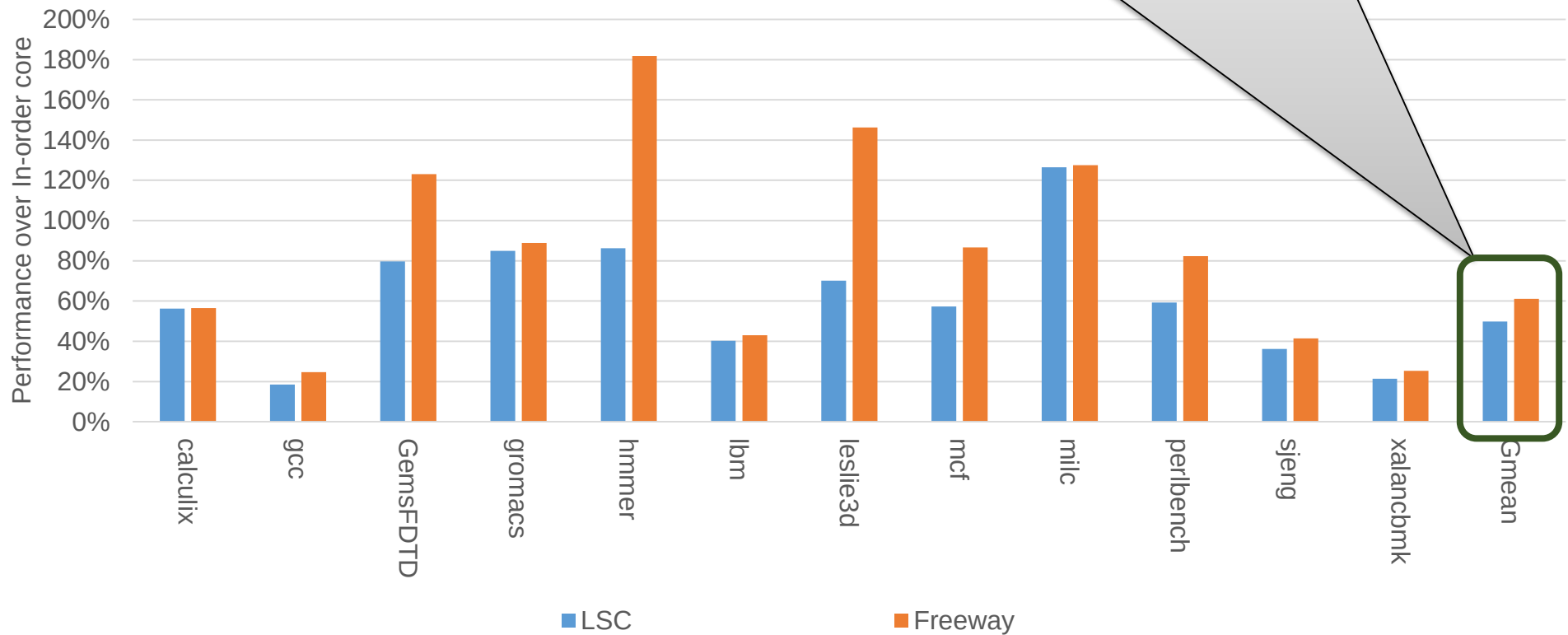
Freeway identifies dependent slices with only one additional bit per RDT entry

Evaluation Methodology

- Core based on Intel Silvermont/ARM Cortex-A9
 - 2-wide
 - 64-entry instruction window
- Workloads
 - SPECcpu2006 with multiple inputs
- Evaluated cores:
 - In-order (Baseline)
 - Load Slice Core [ISCA'15]
 - Freeway
 - Ideal-sOoO (MLP limit)
 - OoO (ILP+MLP limit)

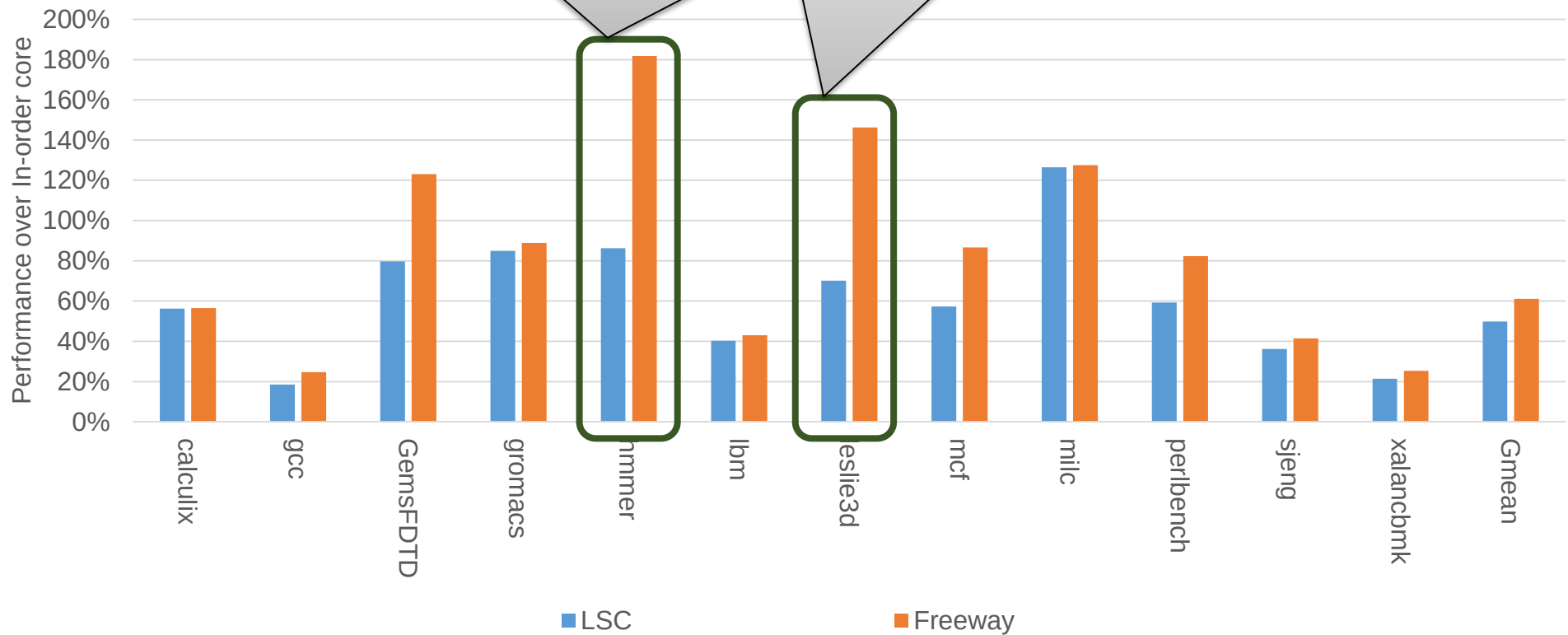
Performance Comparison

Freeway delivers 12% more performance than LSC



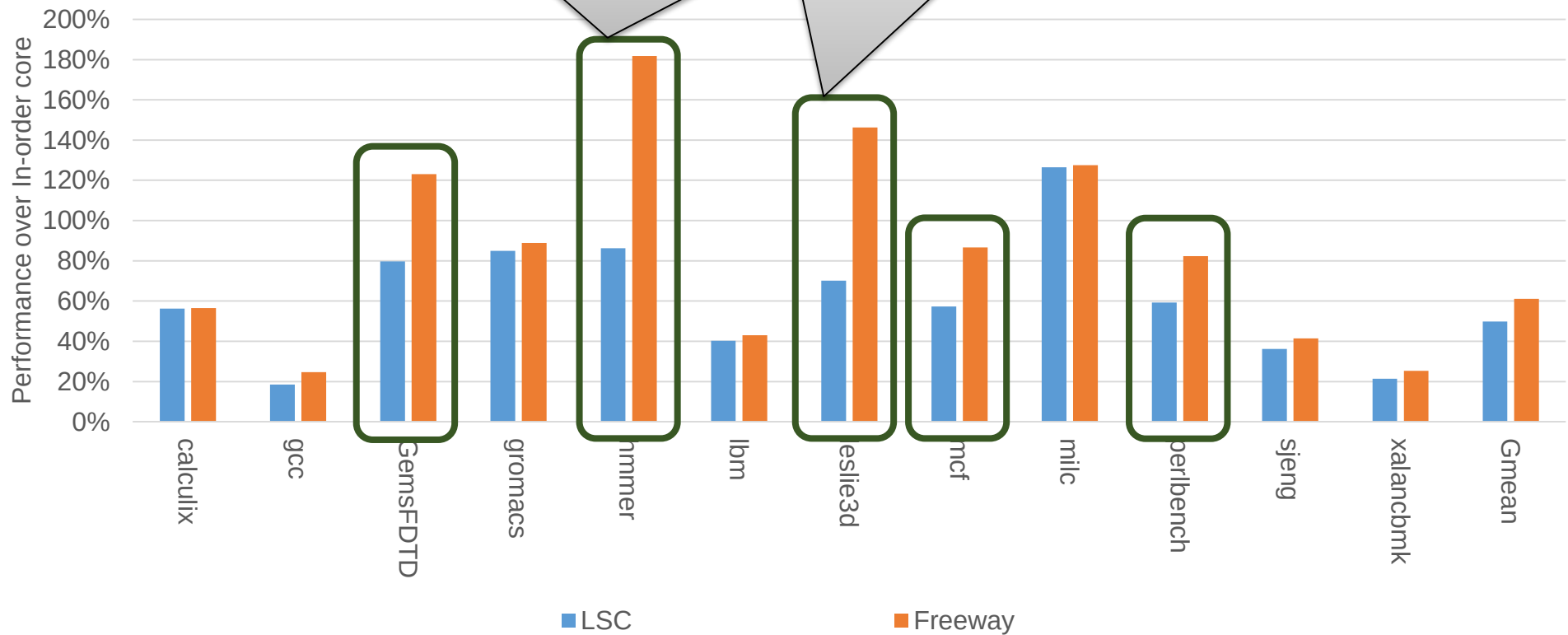
Performance Comparison

Freeway delivers twice as much performance as LSC



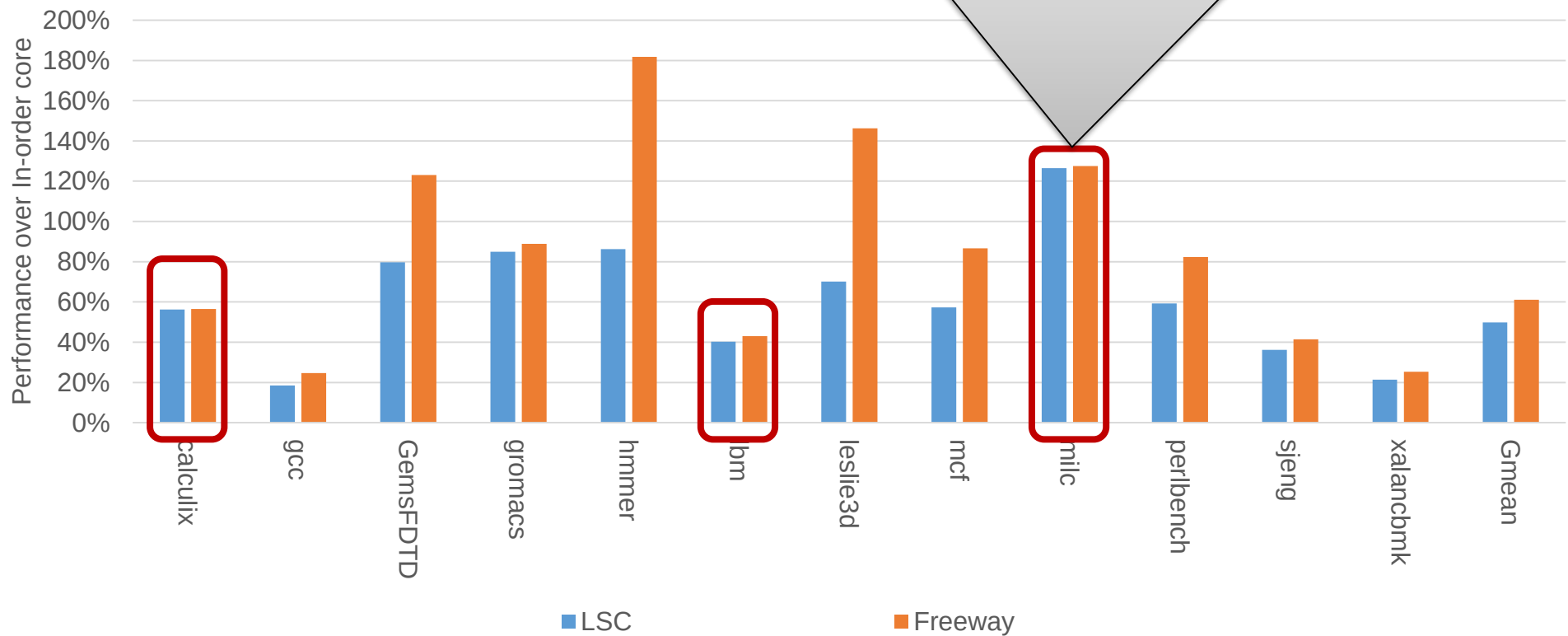
Performance Comparison

Freeway delivers twice as much performance as LSC



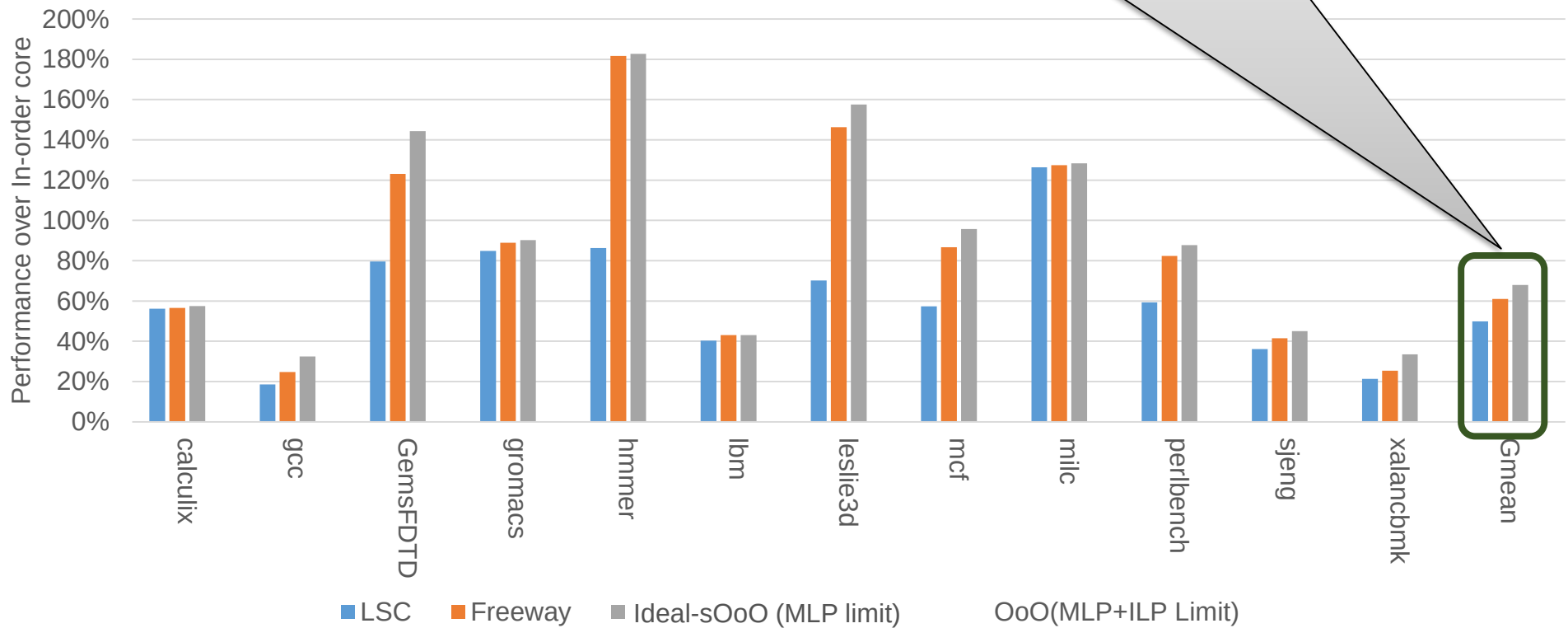
Performance Comparison

Same performance as LSC because of negligible dependent slices



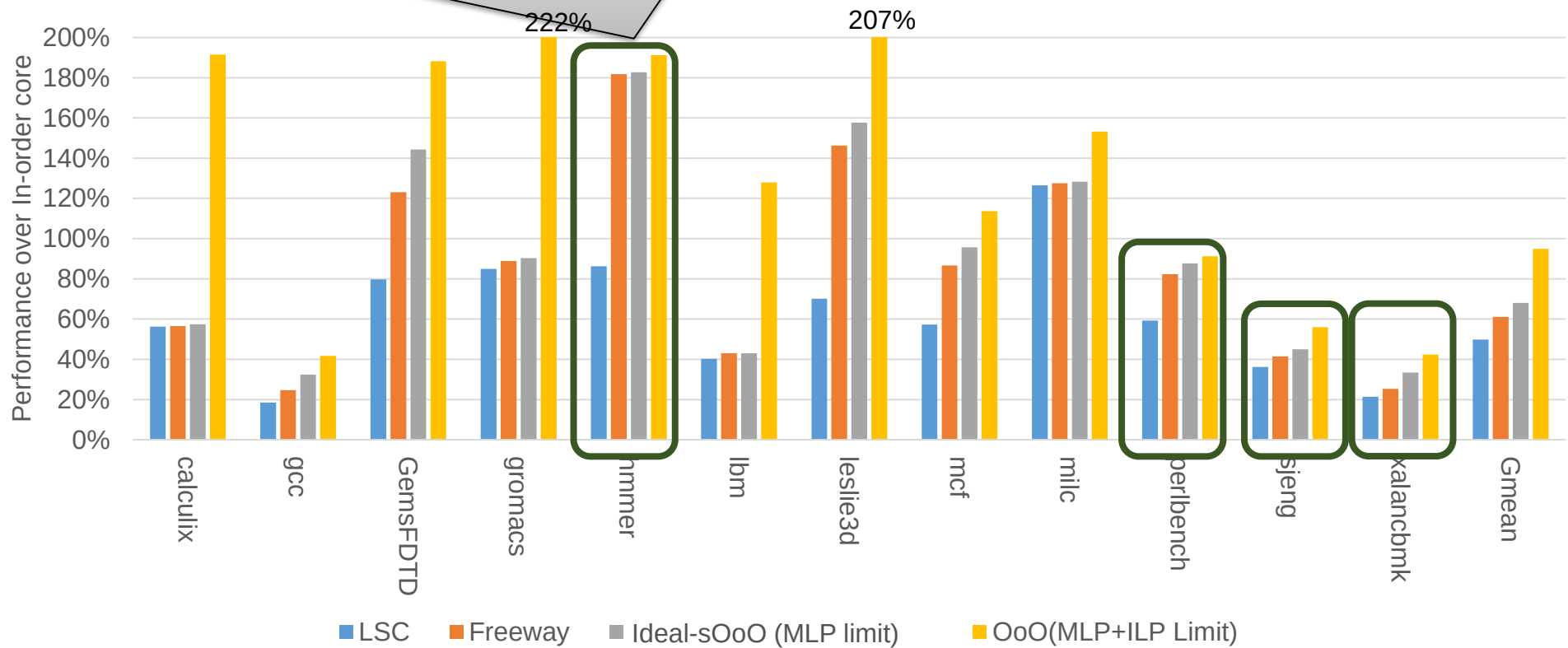
Performance Comparison

Freeway is within 7% of **Ideal** sOoO core



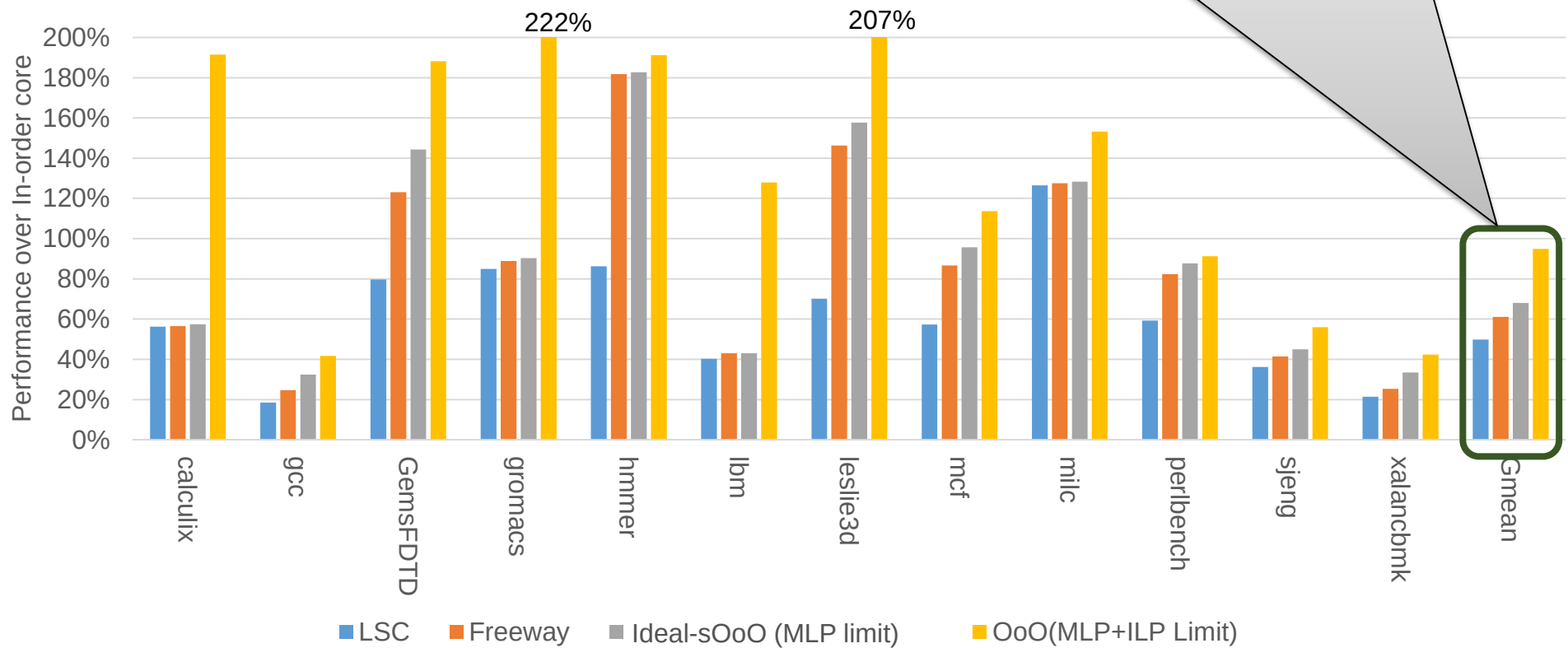
Performance Comparison

Freeway is within 15% of full OoO performance for a number of workloads



Performance Comparison

On avg, Freeway is within 33% of **Fully** OoO core



Freeway Summary

- Memory latency is still a critical bottleneck
 - Large datasets, growing number of cores and accelerators
- MLP is key to hiding memory latency
 - OoO cores pay high power cost to extract MLP
- Slice-OoO (LSC) aims to reduce the power cost
 - Leaves significant MLP due to slice dependencies
- Freeway
 - Introduces dependence aware slice execution
 - Approaches the MLP benefits of an OoO core with power cost of a slice-OoO core.