Precise Runahead Execution

Ajeya Naithani, Josue Feliu,
Almutaz Adileh, Lieven Eeckhout

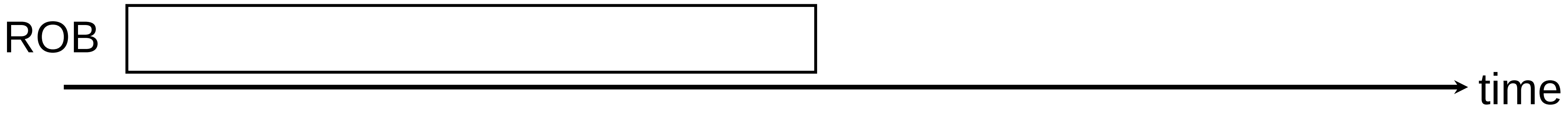
GHENT
UNIVERSITY

UNIVERSITAT
POLITÈCNICA
DE VALÈNCIA

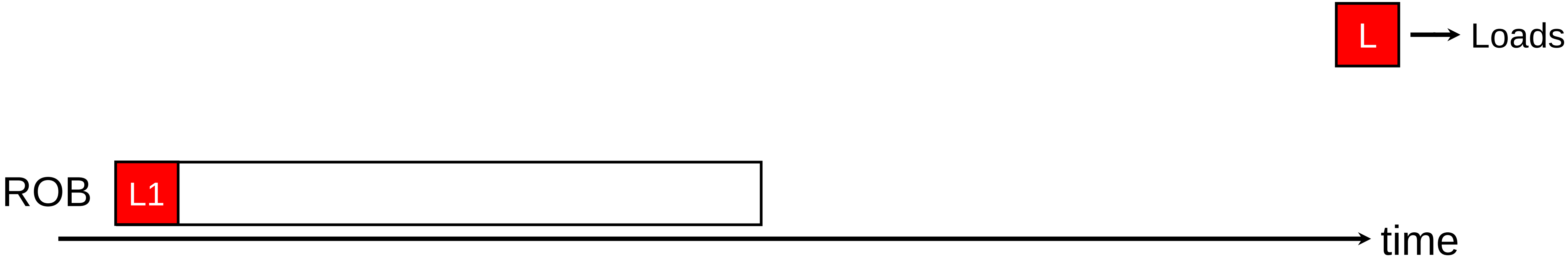
Full-Window Stalls Degrade Performance

Full-Window Stalls Degrade Performance

Full-Window Stalls Degrade Performance



Full-Window Stalls Degrade Performance



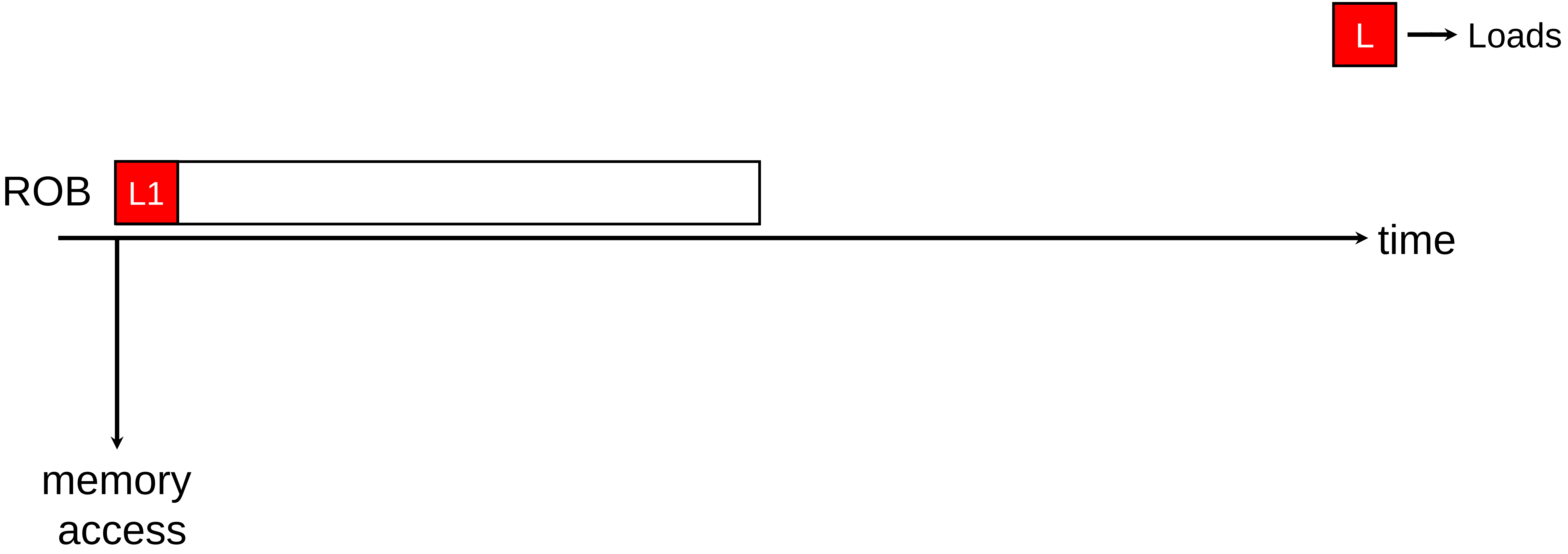
L → Loads

ROB

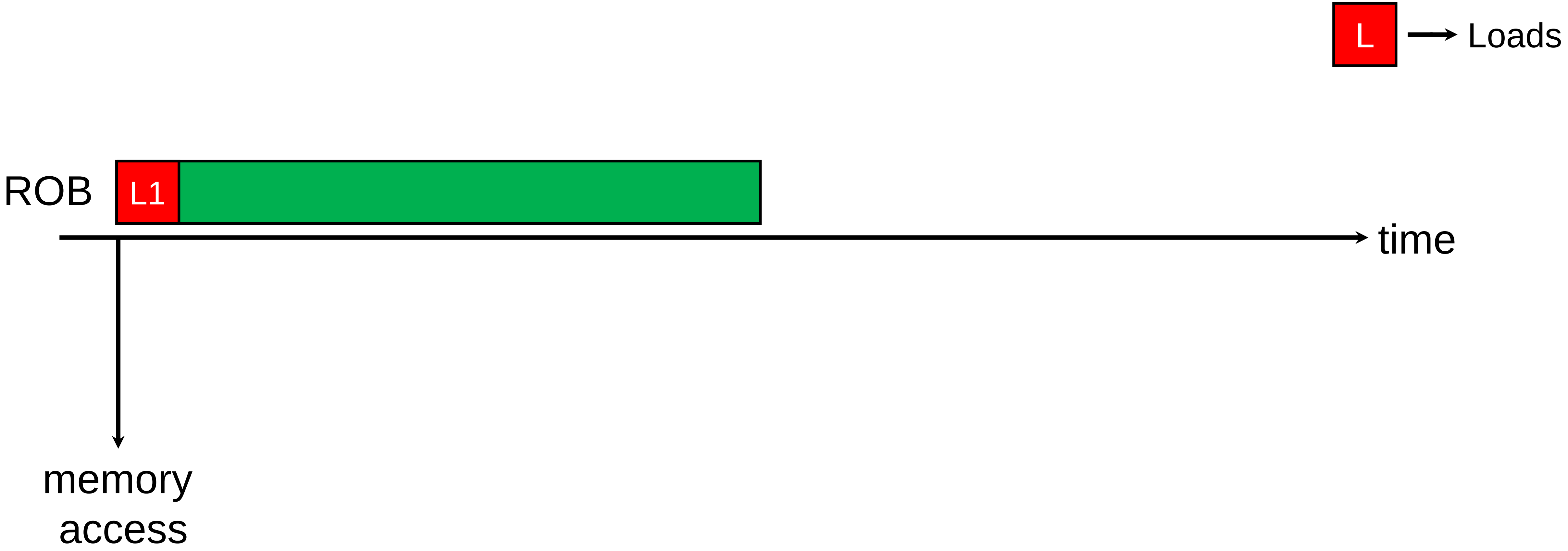
L1

time

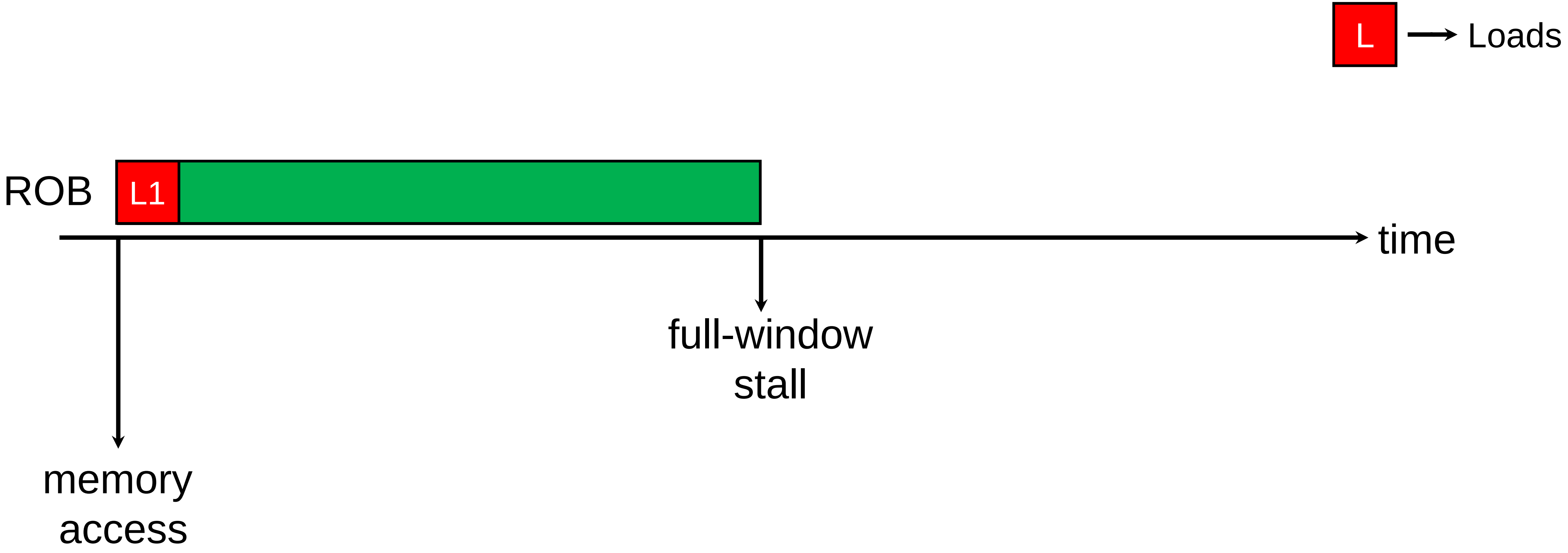
Full-Window Stalls Degrade Performance



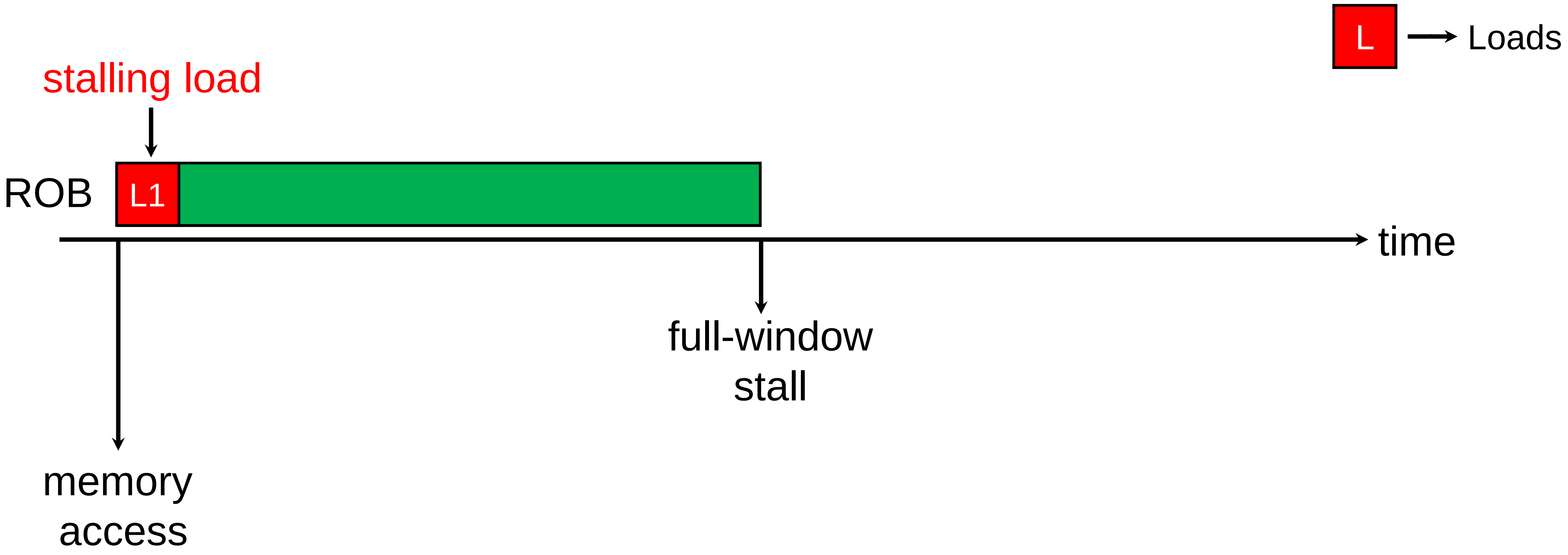
Full-Window Stalls Degrade Performance



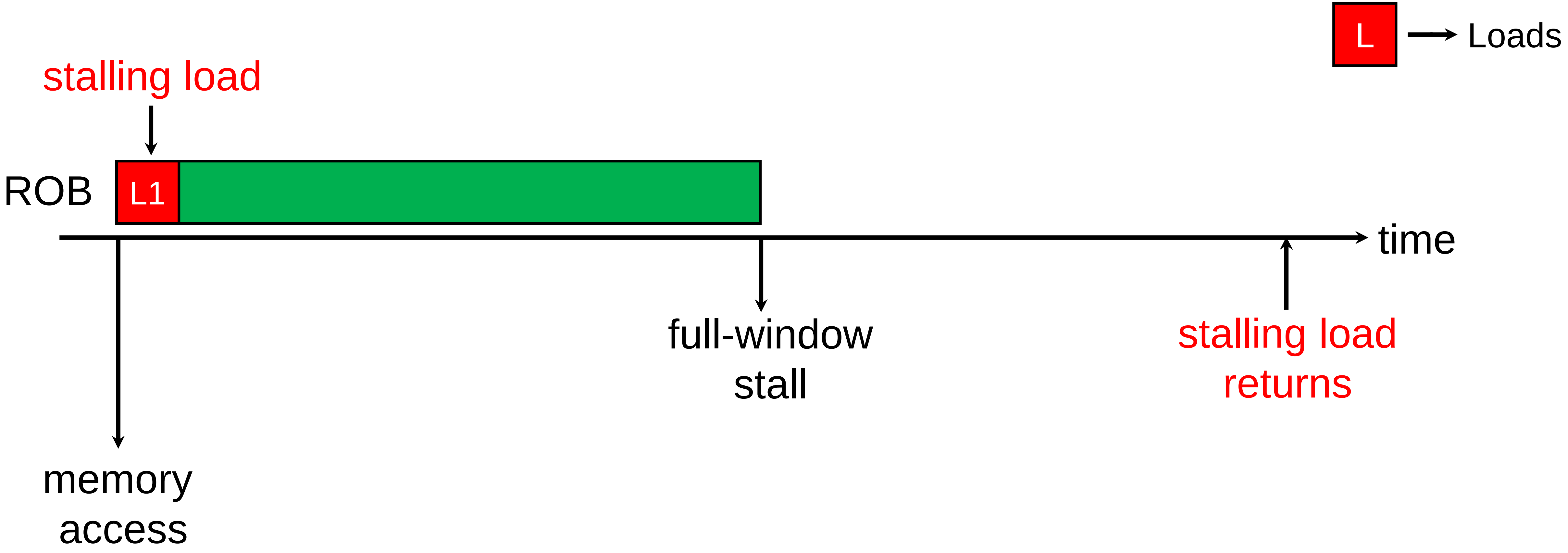
Full-Window Stalls Degrade Performance



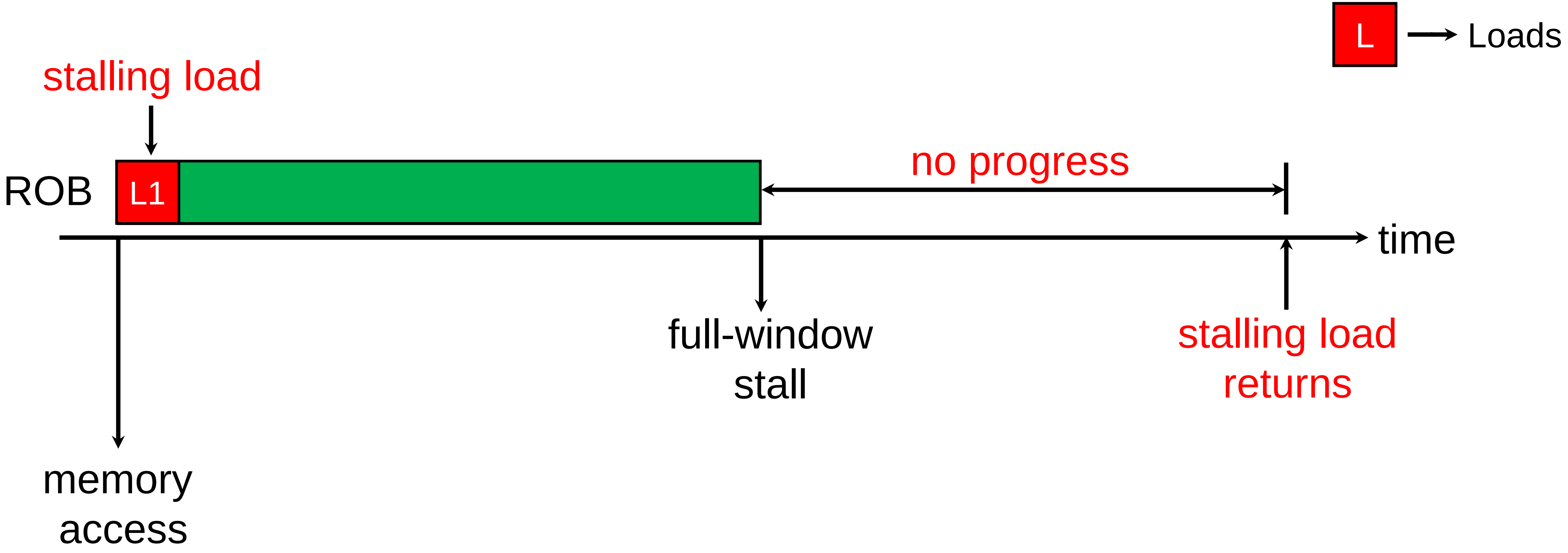
Full-Window Stalls Degrade Performance



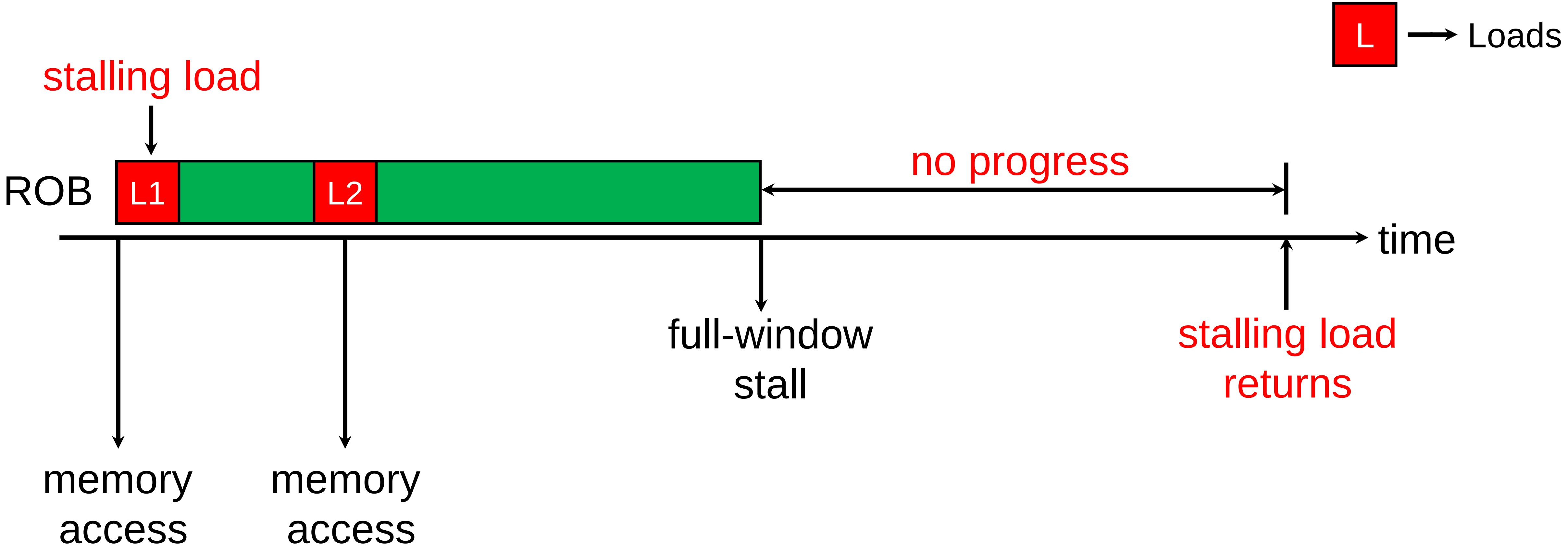
Full-Window Stalls Degrade Performance



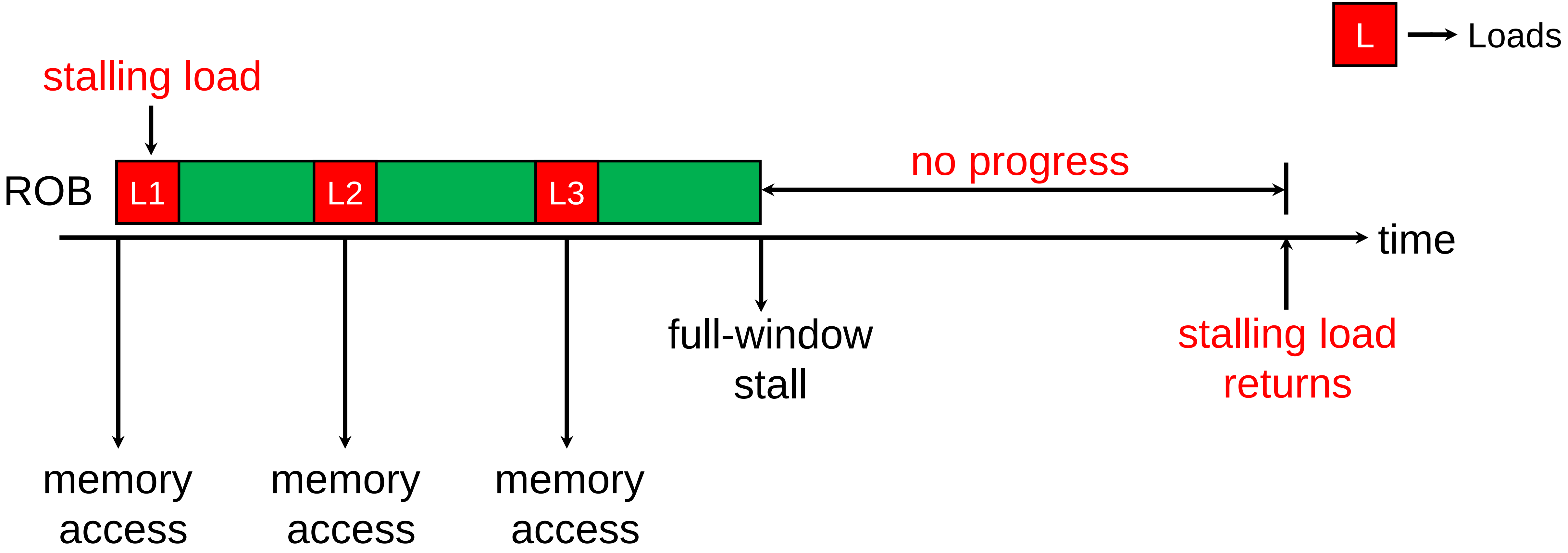
Full-Window Stalls Degrade Performance



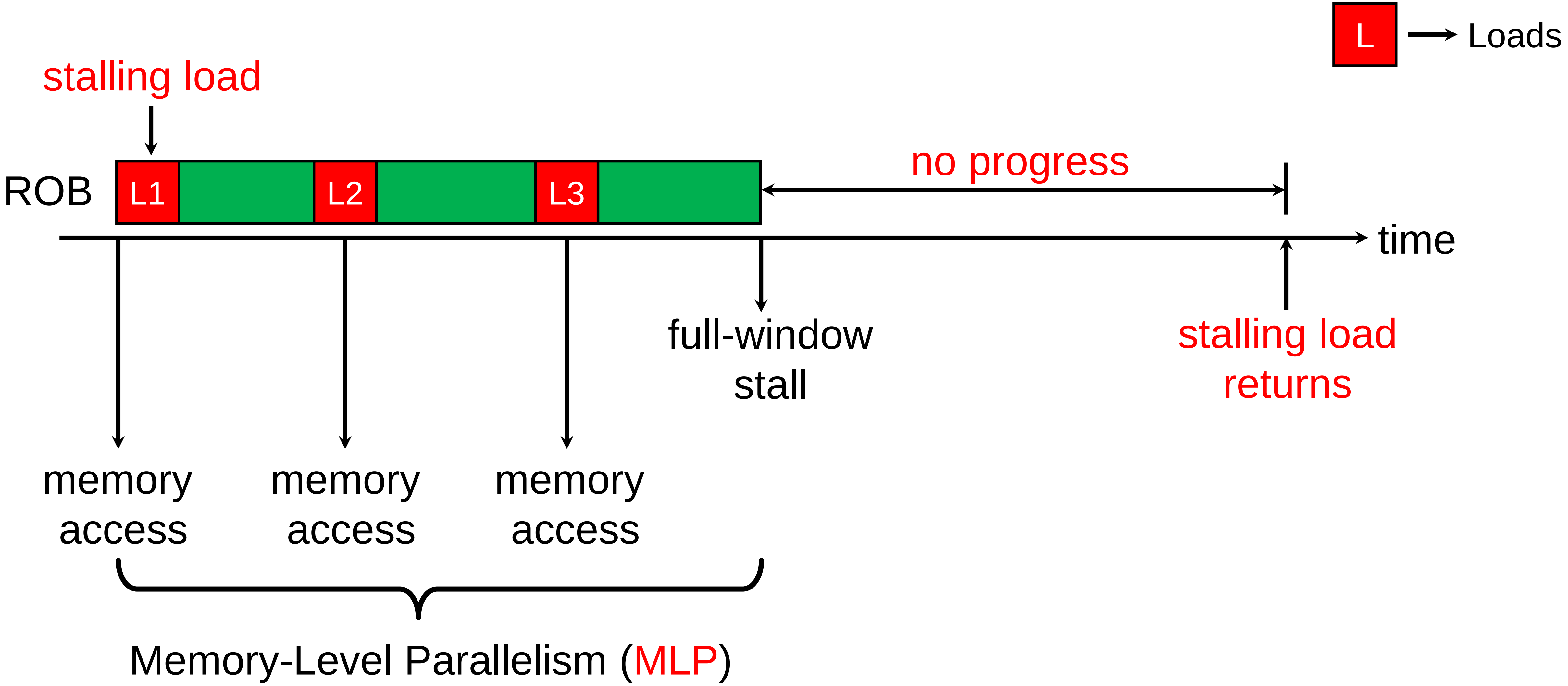
Full-Window Stalls Degrade Performance



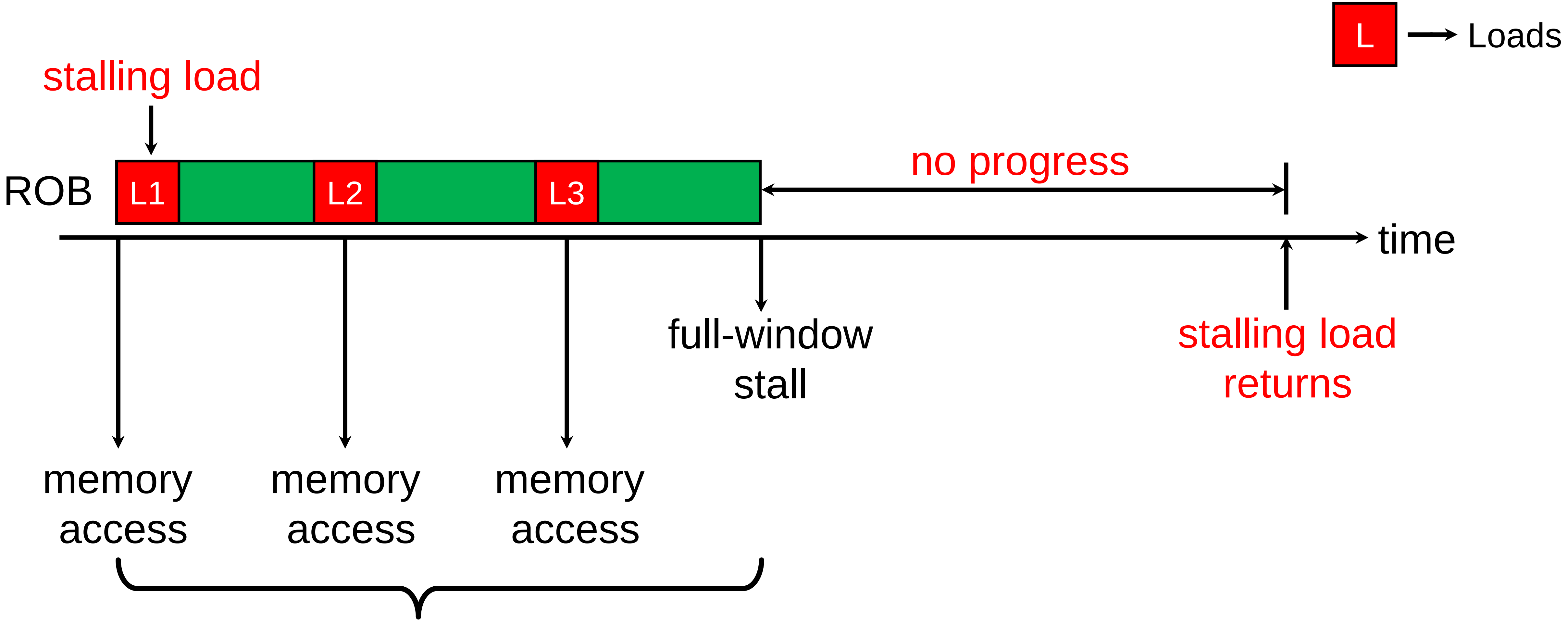
Full-Window Stalls Degrade Performance



Full-Window Stalls Degrade Performance

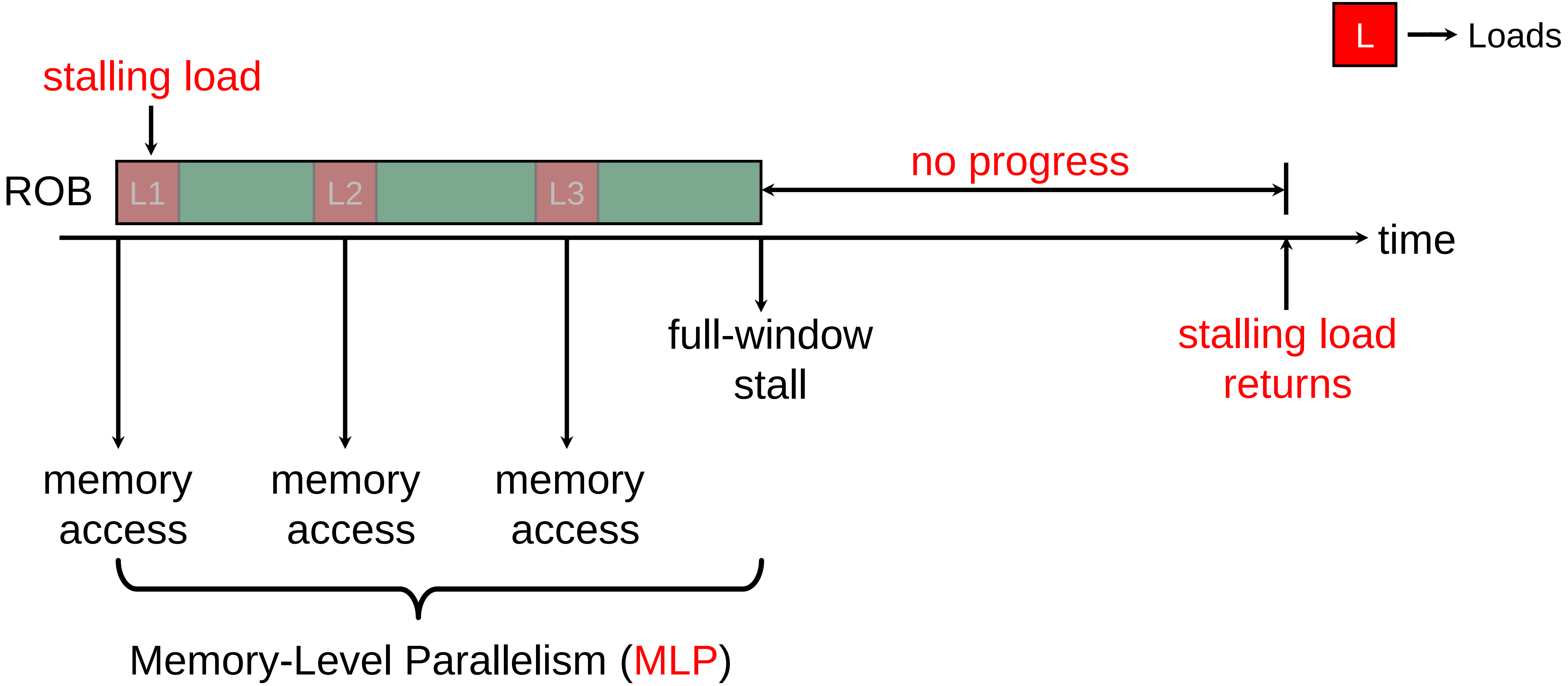


Runahead Execution Prefetches under a Full-Window Stall

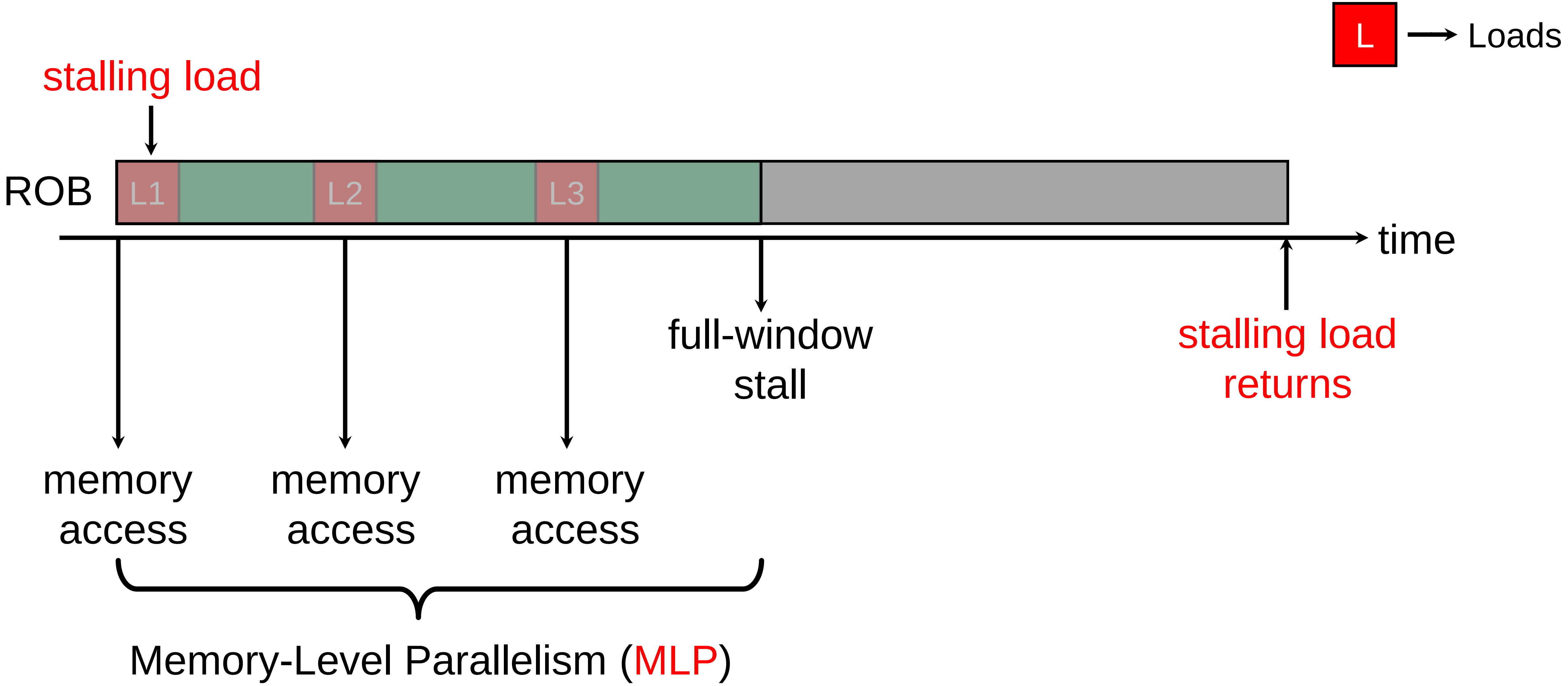


Memory-Level Parallelism (MLP)

Runahead Execution Prefetches under a Full-Window Stall



Runahead Execution Prefetches under a Full-Window Stall



L → Loads

stalling load

ROB

L1

L2

L3

time

memory access

memory access

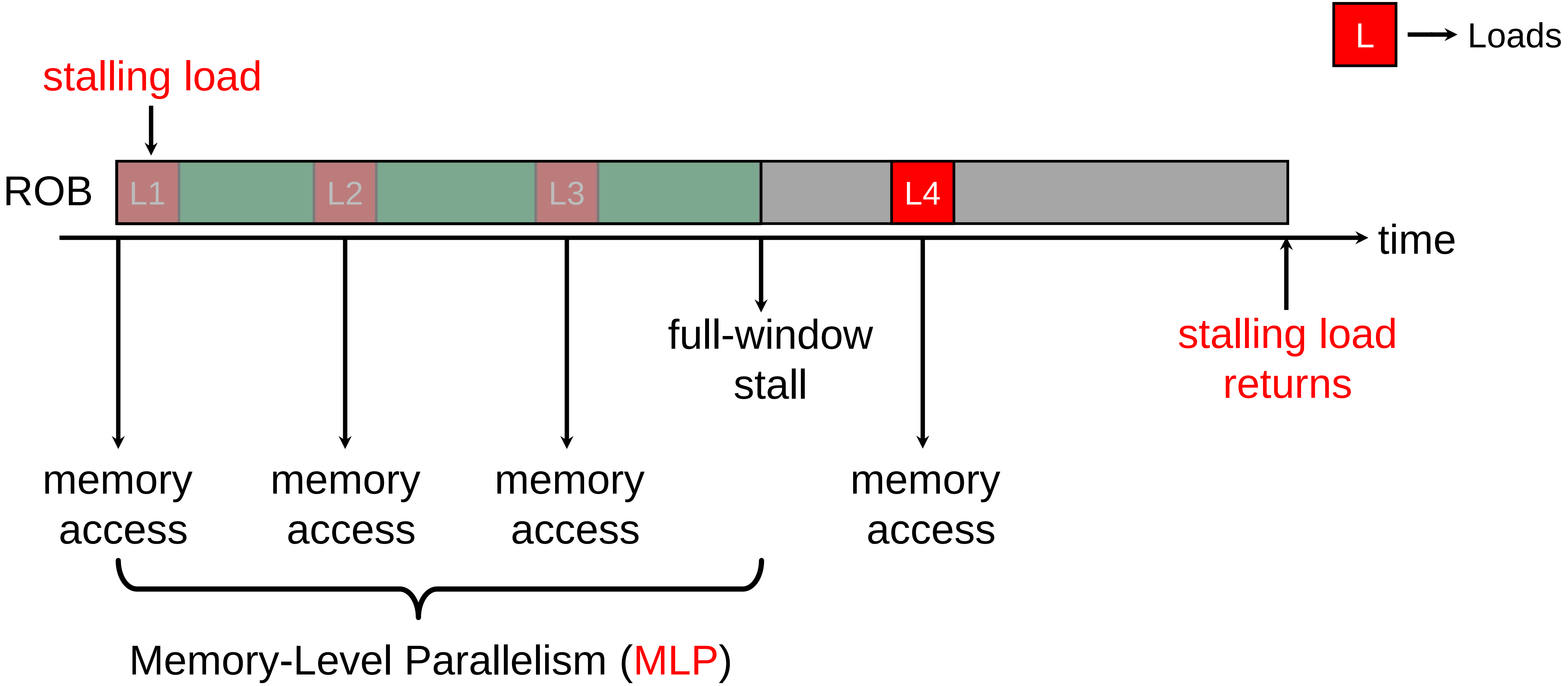
memory access

full-window stall

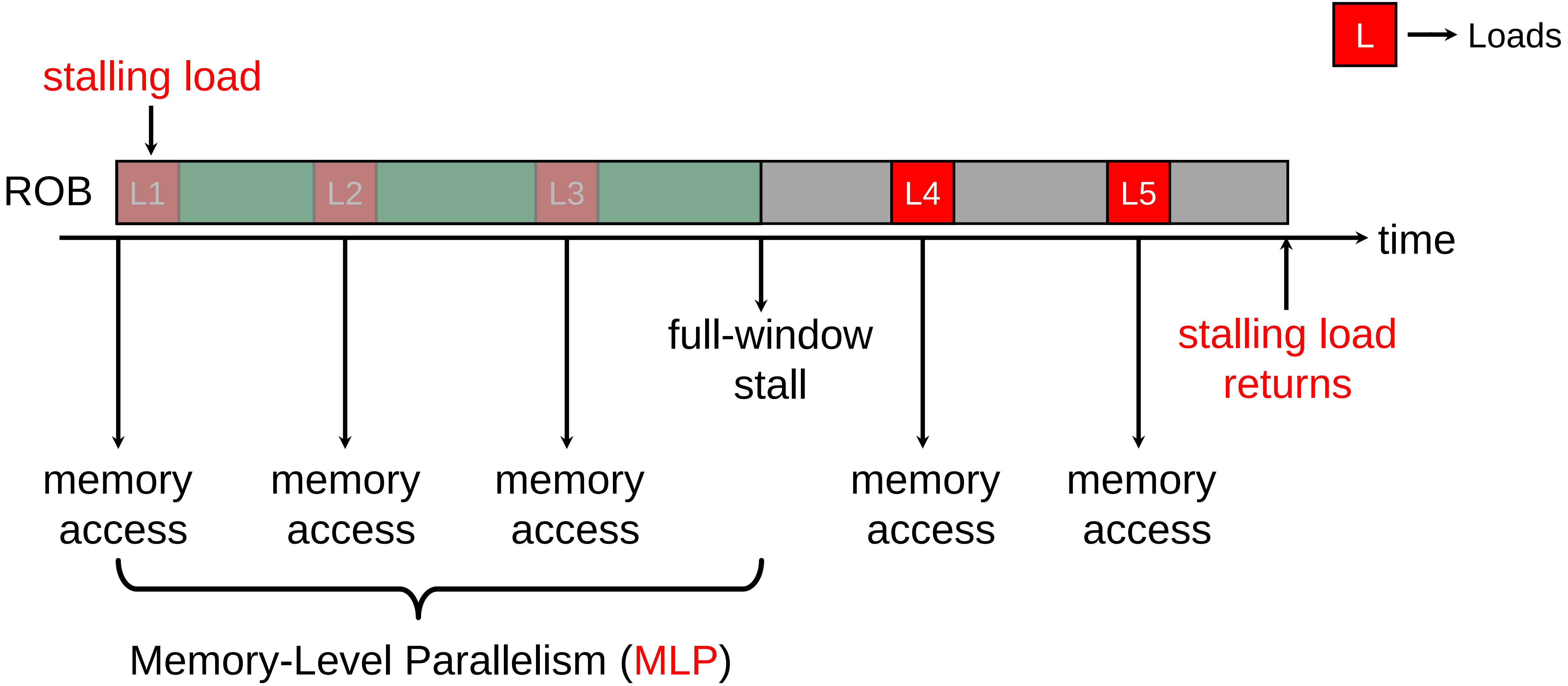
stalling load returns

Memory-Level Parallelism (MLP)

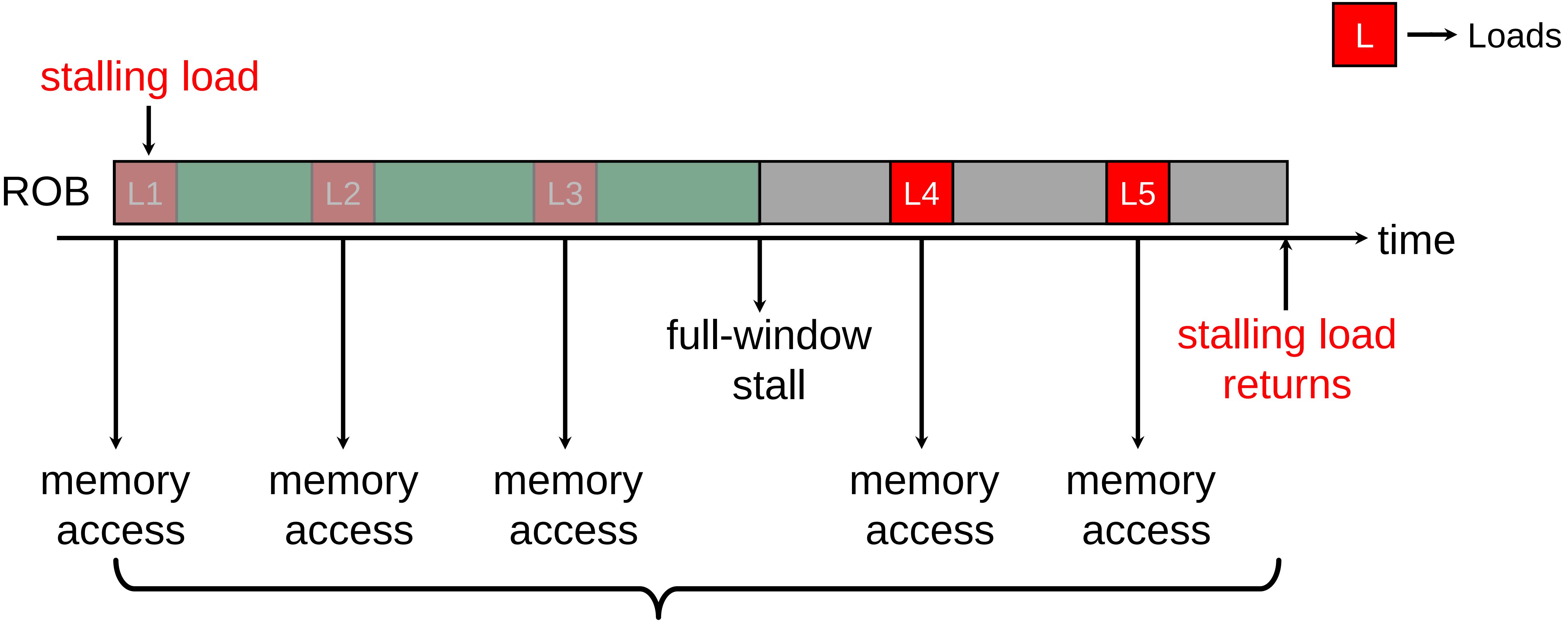
Runahead Execution Prefetches under a Full-Window Stall



Runahead Execution Prefetches under a Full-Window Stall

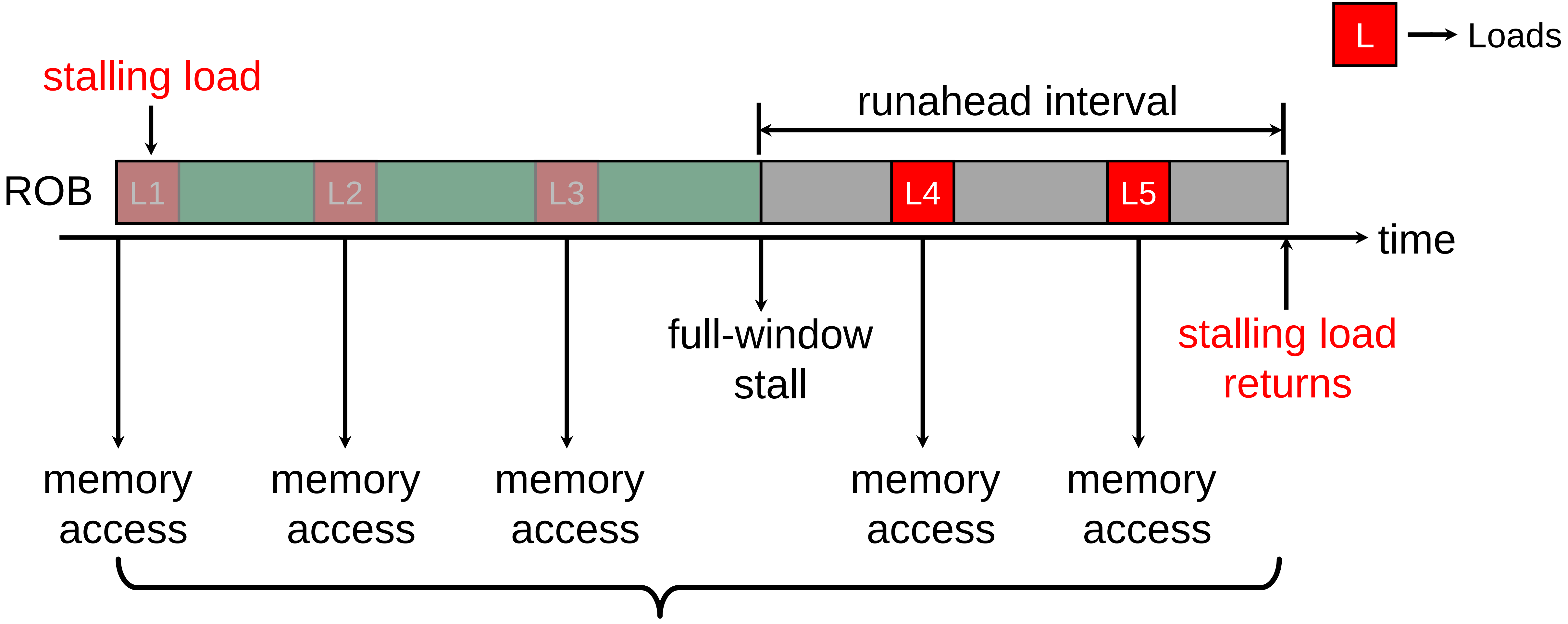


Runahead Execution Prefetches under a Full-Window Stall



Increased Memory-Level Parallelism (MLP)

Runahead Execution Prefetches under a Full-Window Stall

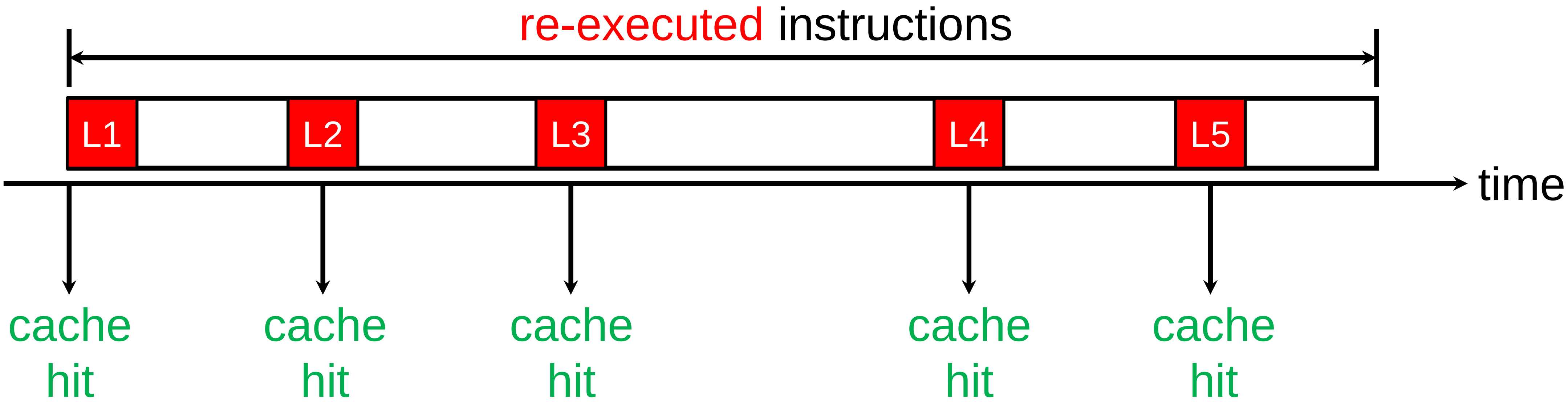


Increased Memory-Level Parallelism (MLP)

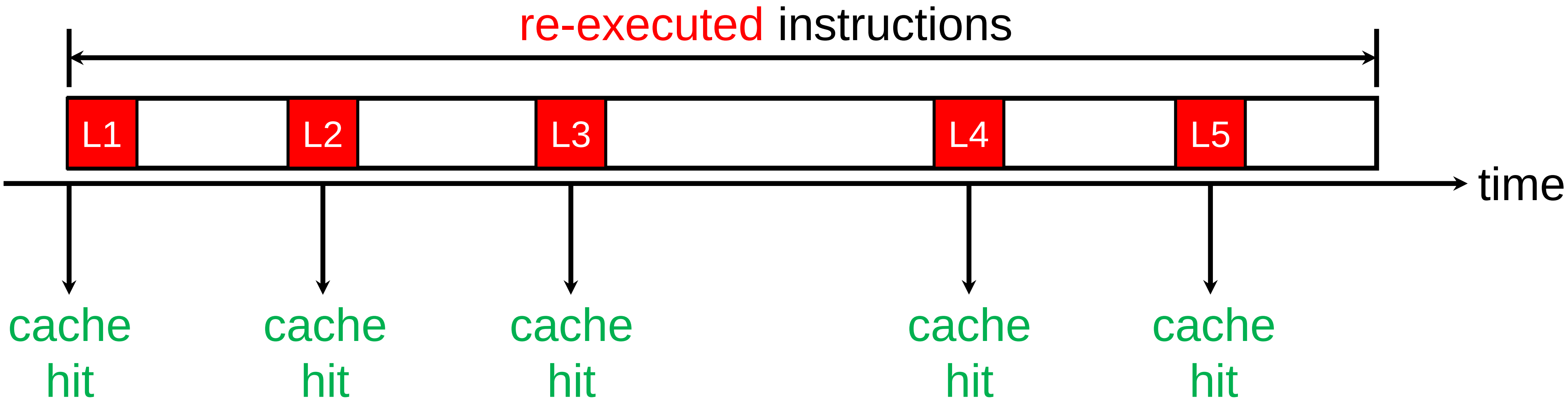
Runahead Execution Re-Executes All Instructions

Runahead Execution Re-Executes All Instructions

Runahead Execution Re-Executes All Instructions

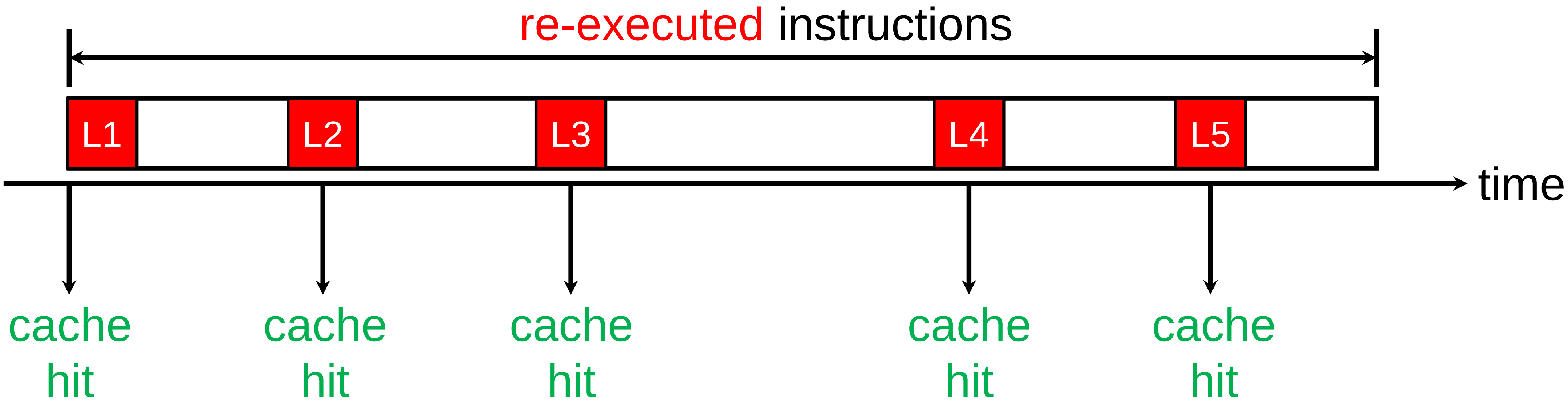


Runahead Execution Re-Executes All Instructions



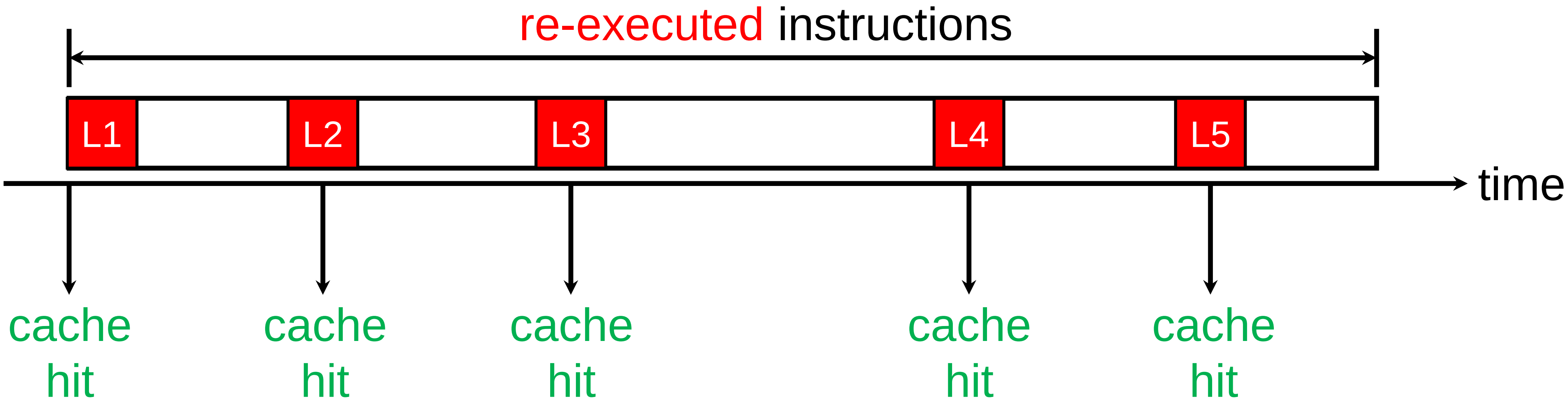
fetch

Runahead Execution Re-Executes All Instructions



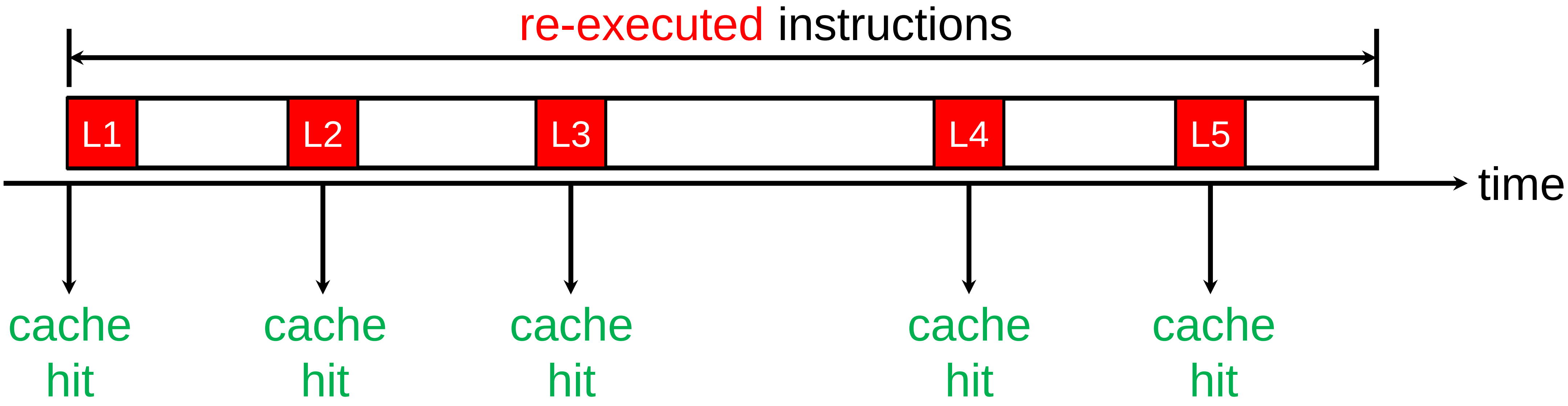
fetch → decode

Runahead Execution Re-Executes All Instructions



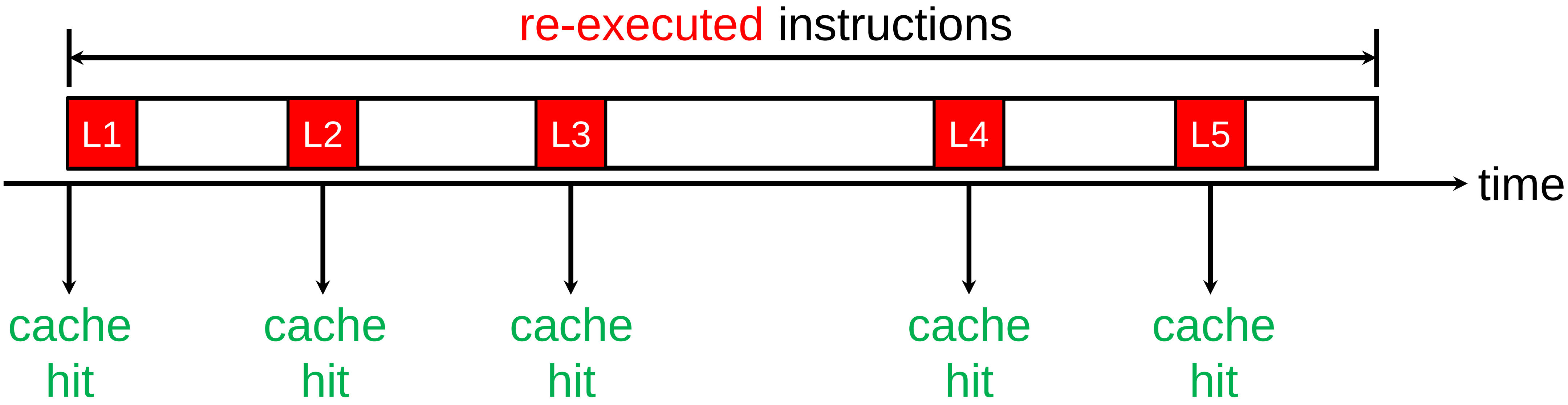
fetch → decode → rename

Runahead Execution Re-Executes All Instructions



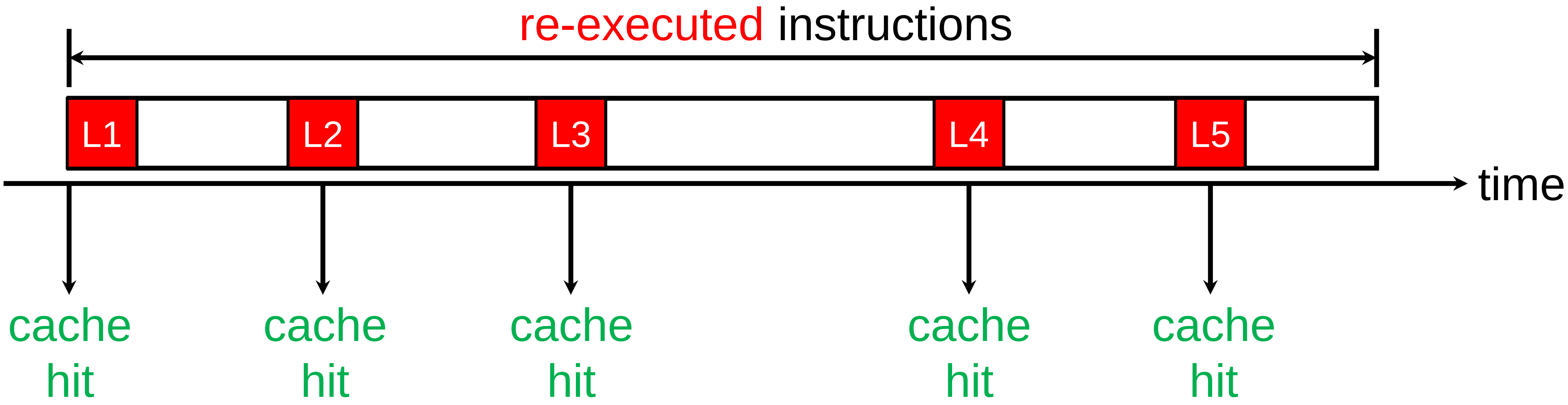
fetch → decode → rename → dispatch

Runahead Execution Re-Executes All Instructions



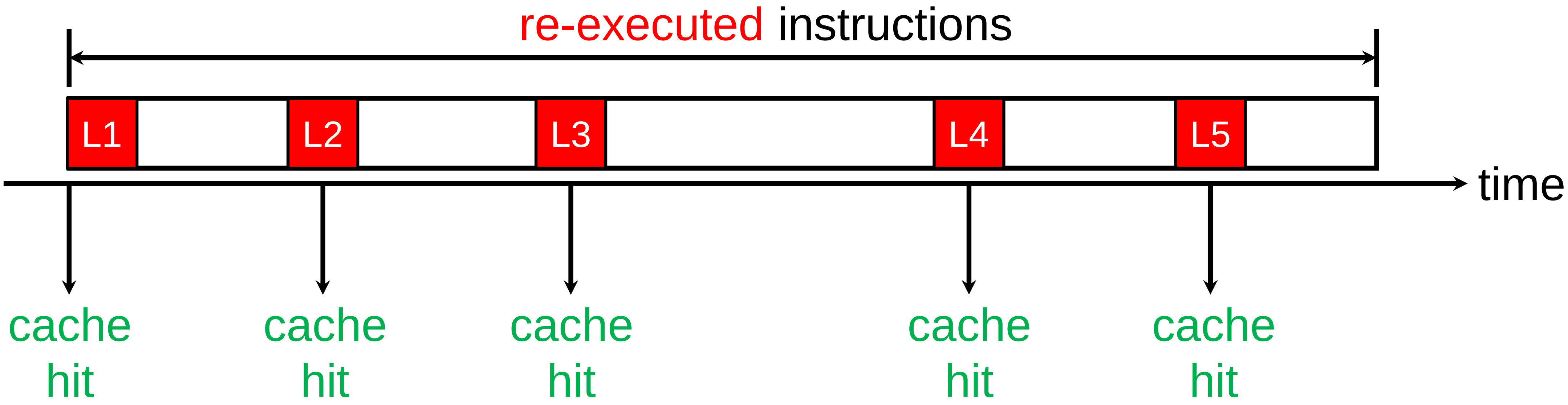
fetch → decode → rename → dispatch → issue

Runahead Execution Re-Executes All Instructions



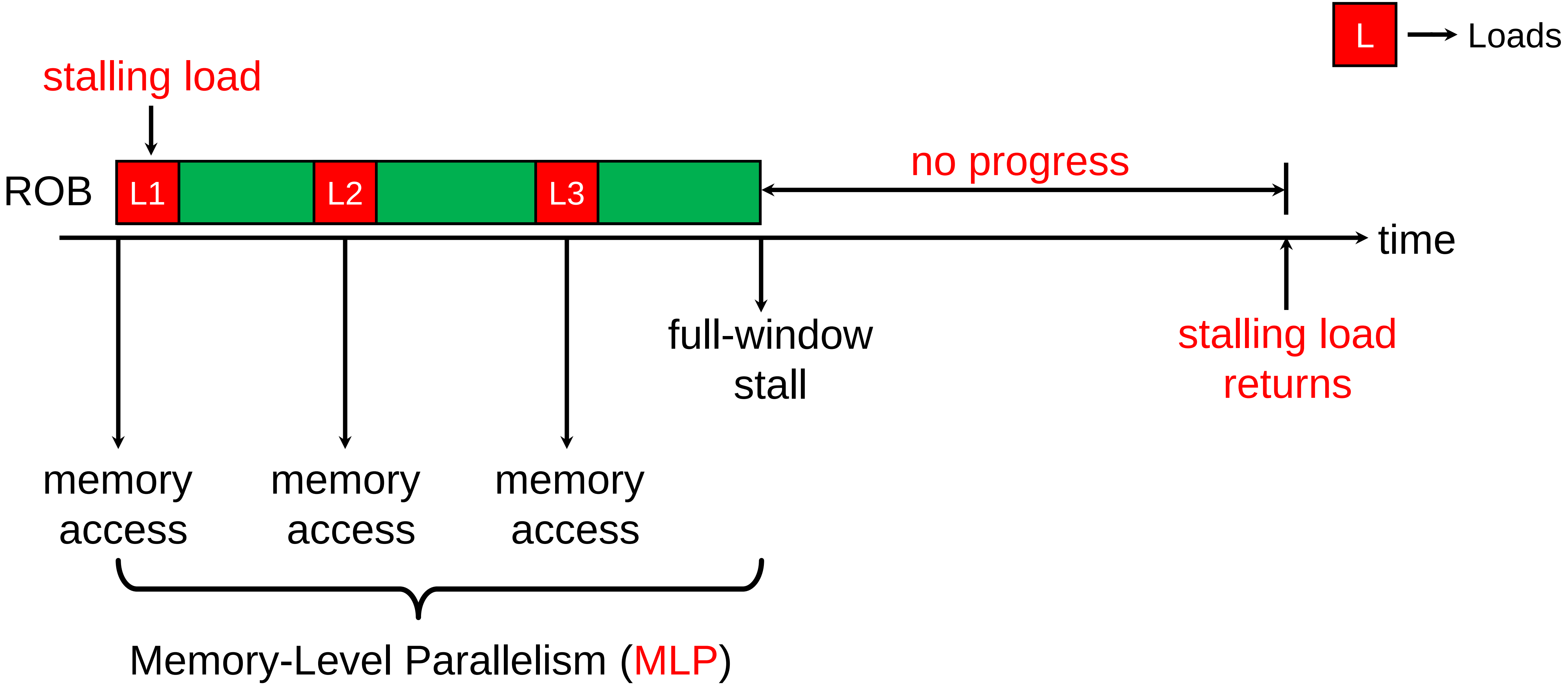
fetch → decode → rename → dispatch → issue → execute

Runahead Execution Re-Executes All Instructions

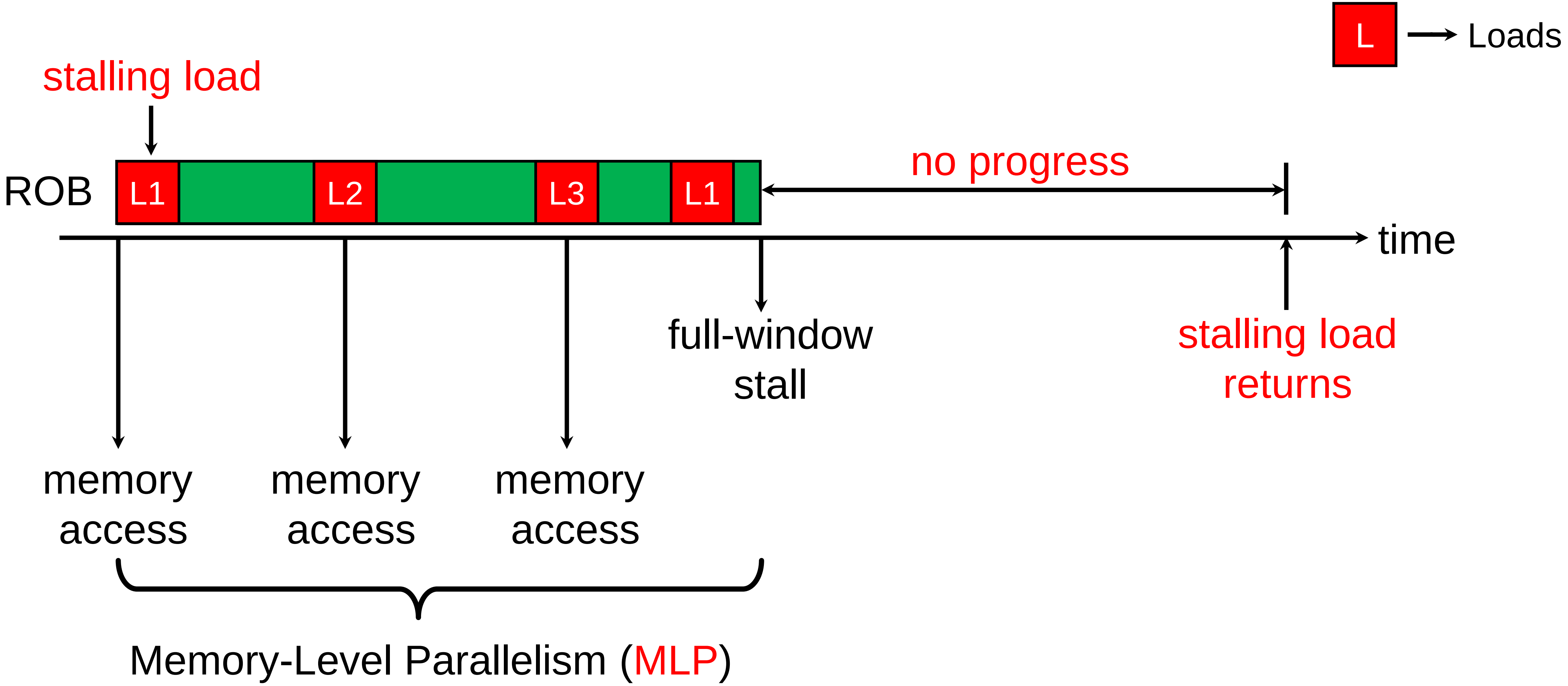


fetch → decode → rename → dispatch → issue → execute → commit

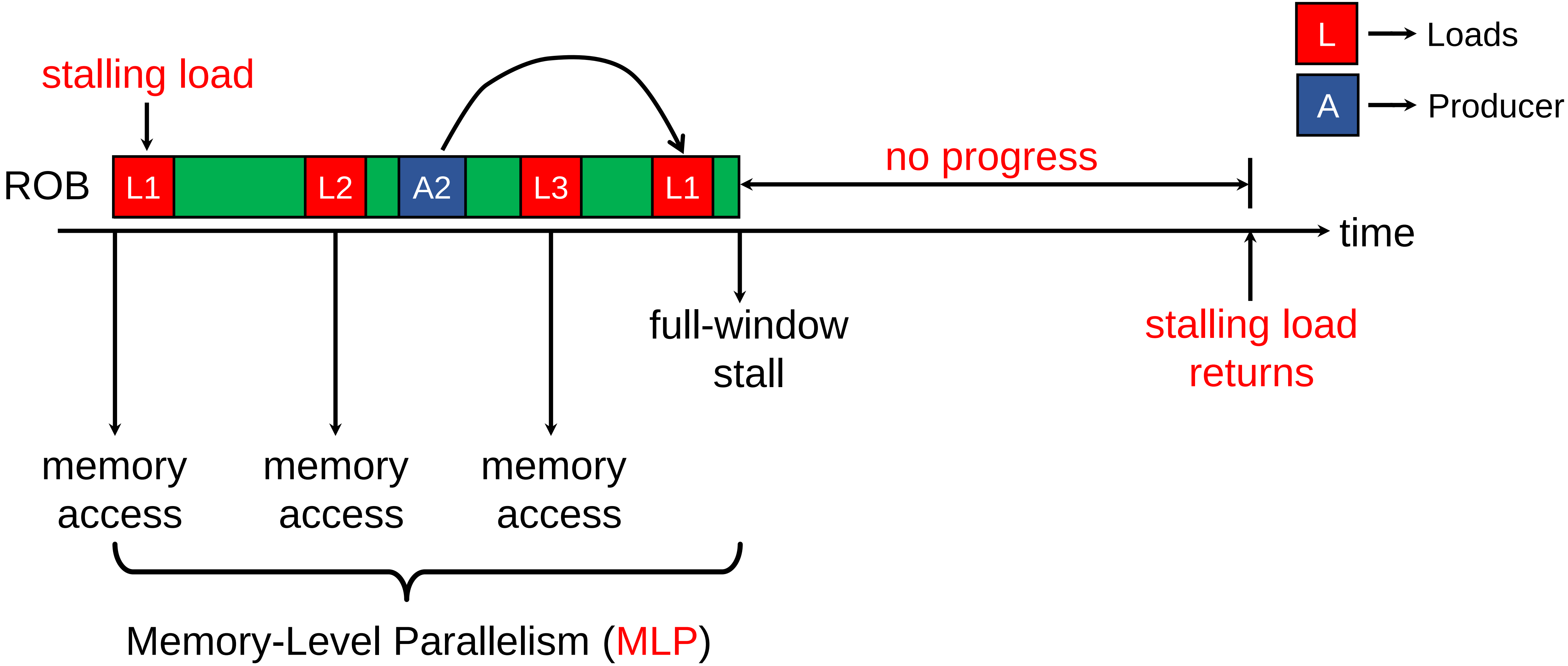
Runahead Buffer Finds Blocking Chain in the ROB



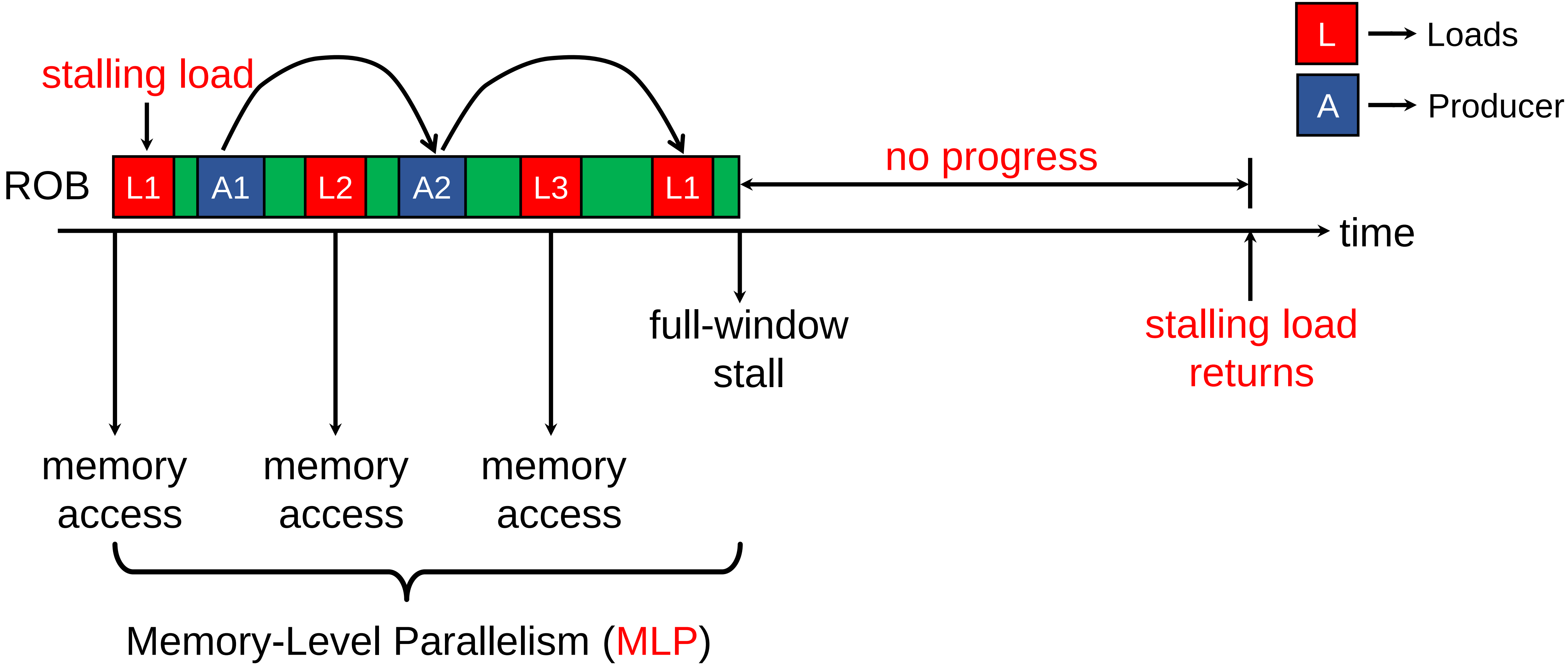
Runahead Buffer Finds Blocking Chain in the ROB



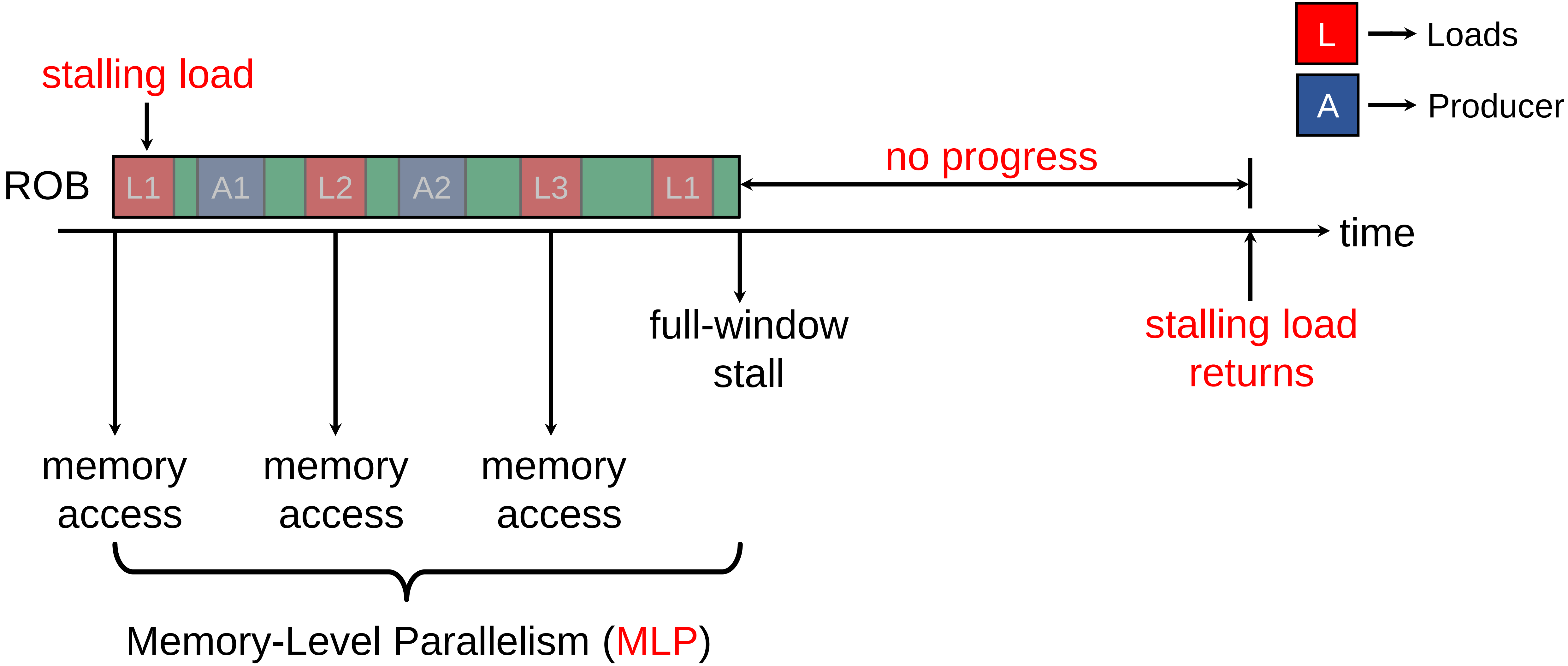
Runahead Buffer Finds Blocking Chain in the ROB



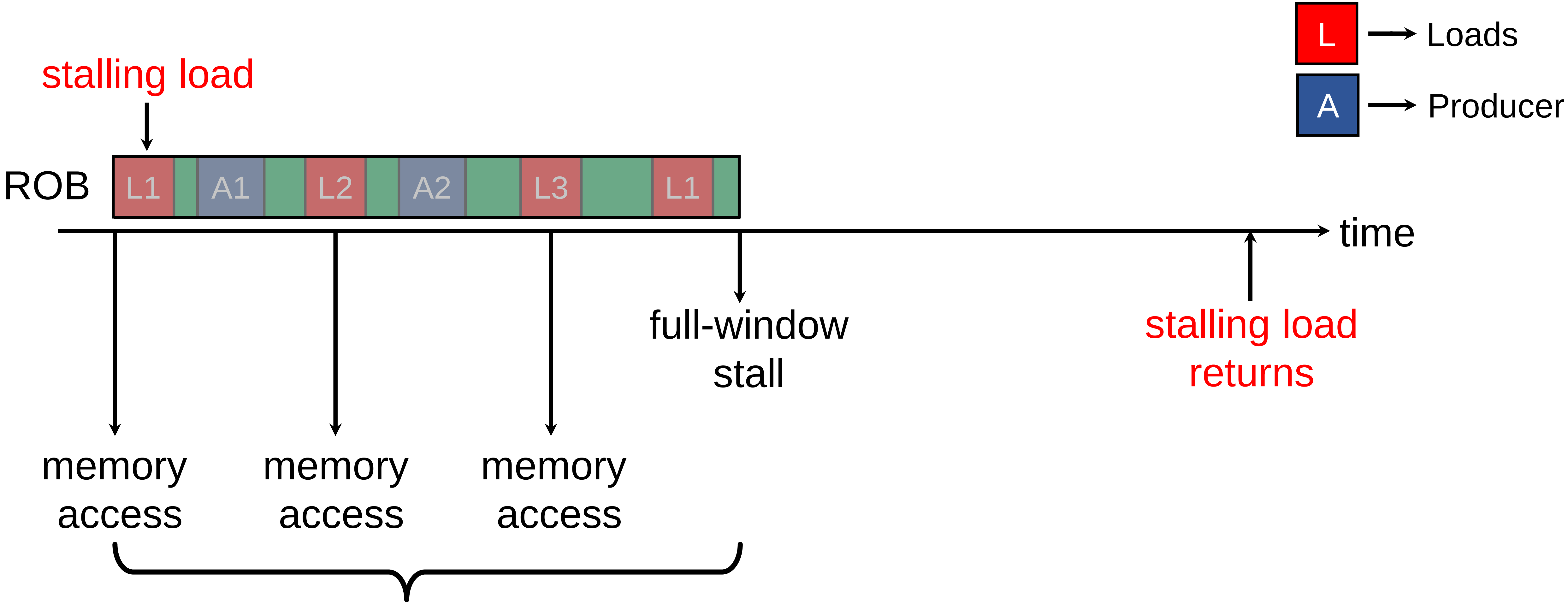
Runahead Buffer Finds Blocking Chain in the ROB



Runahead Buffer Executes Blocking Chain Speculatively

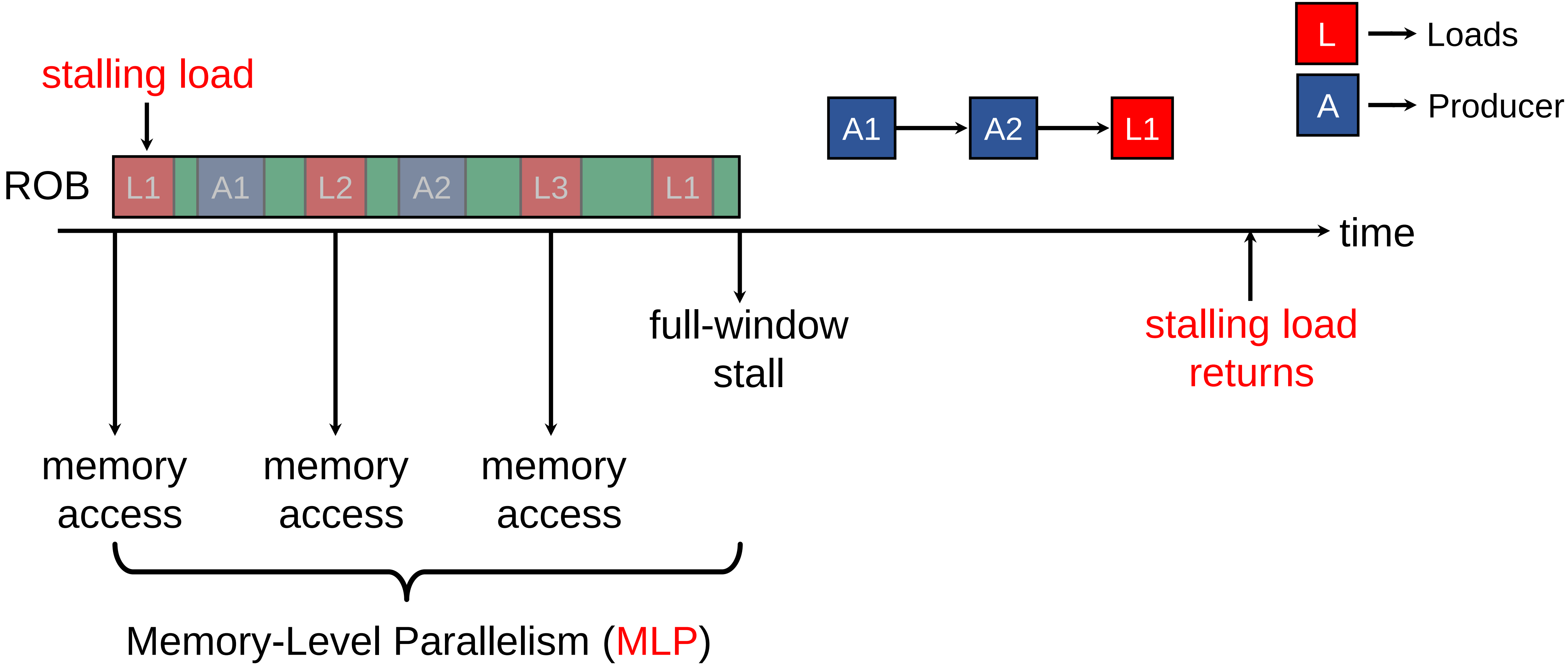


Runahead Buffer Executes Blocking Chain Speculatively

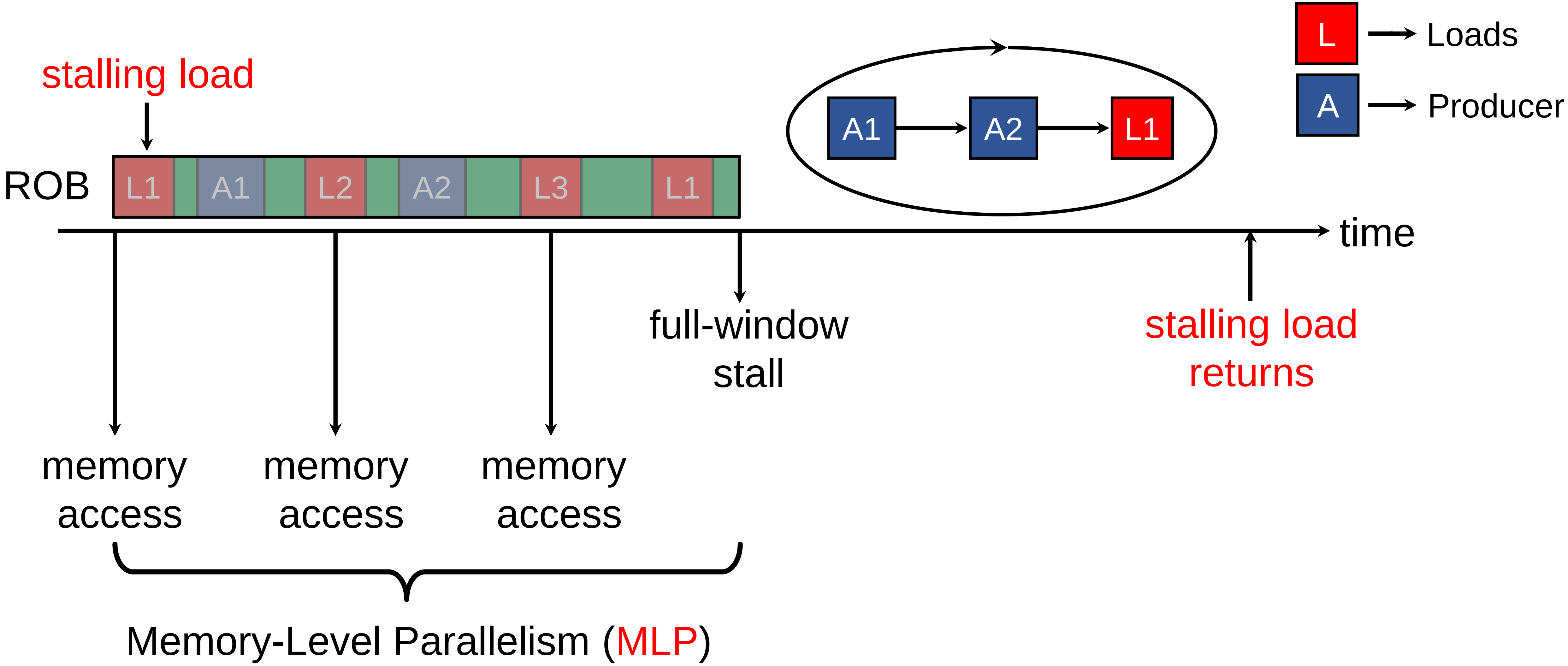


Memory-Level Parallelism (MLP)

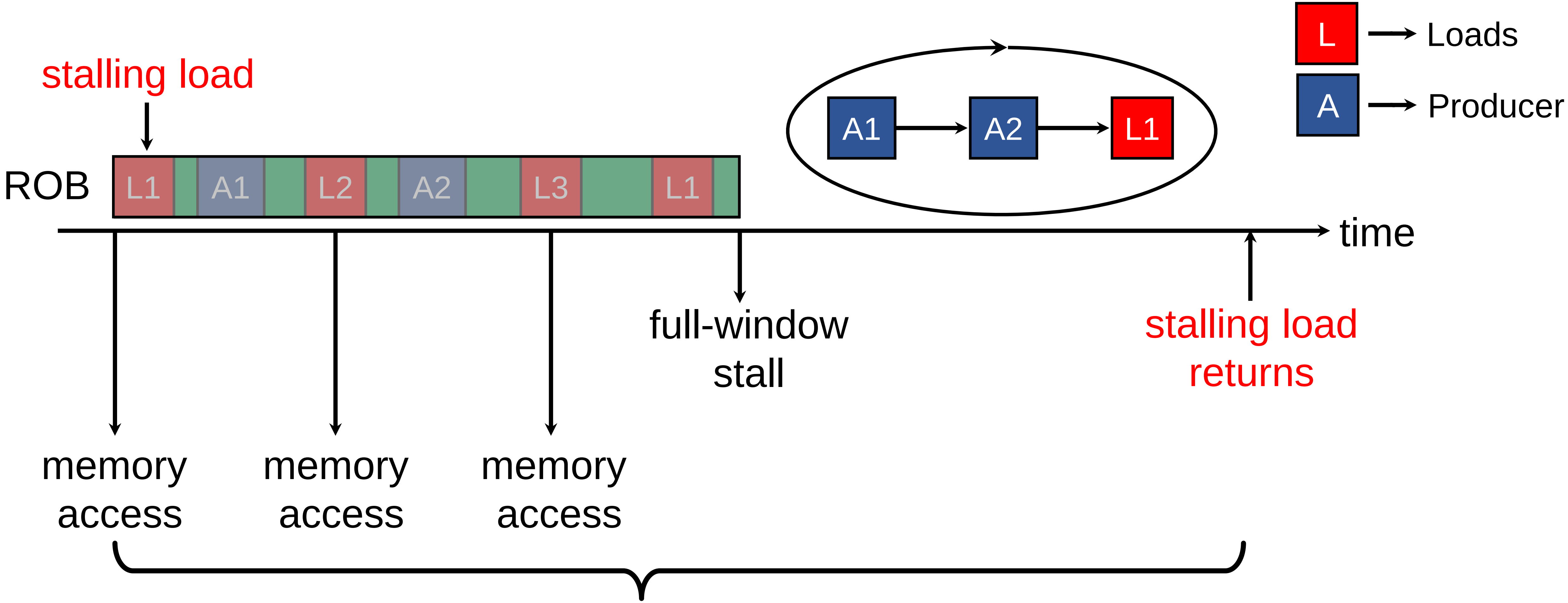
Runahead Buffer Executes Blocking Chain Speculatively



Runahead Buffer Executes Blocking Chain Speculatively



Runahead Buffer Executes Blocking Chain Speculatively

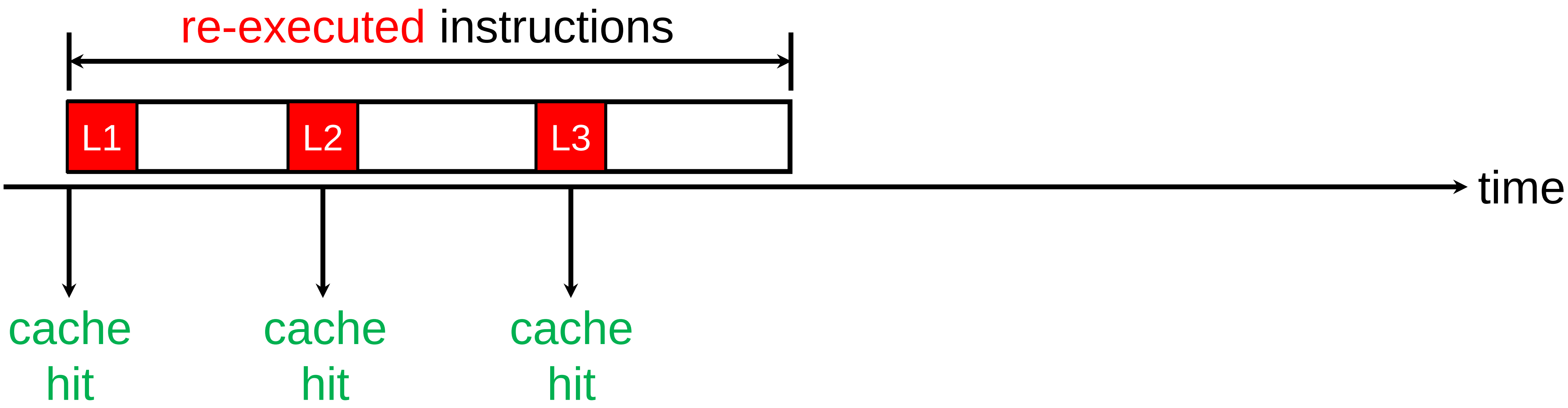


Increased Memory-Level Parallelism (MLP)

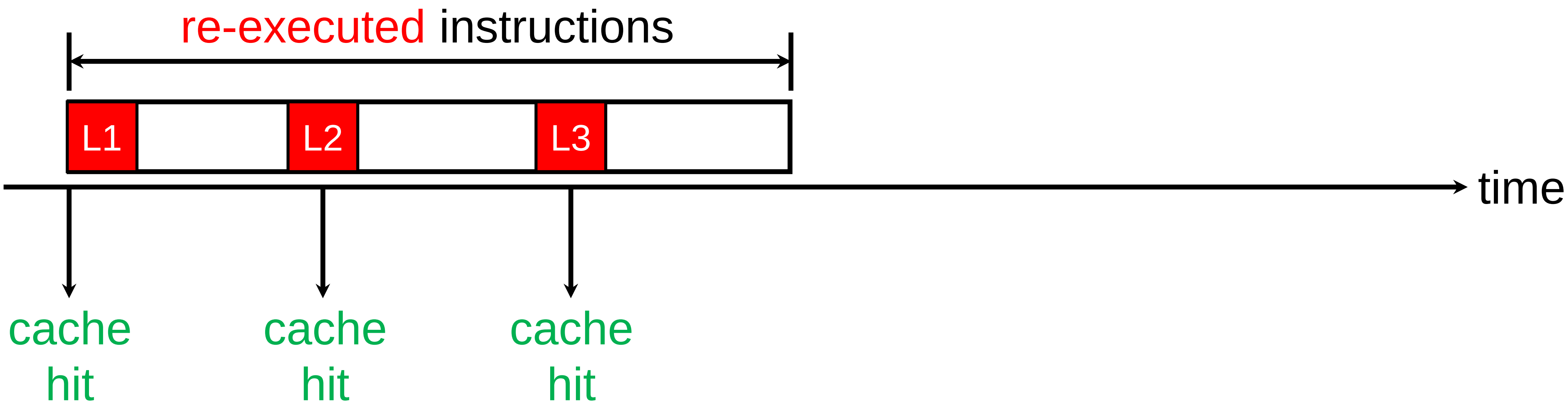
Runahead Buffer Re-Executes the Window

Runahead Buffer Re-Executes the Window

Runahead Buffer Re-Executes the Window



Runahead Buffer Re-Executes the Window



fetch → decode → rename → dispatch → issue → execute → commit

Runahead Techniques Relative to OoO Core

Runahead Techniques Relative to OoO Core

Runahead execution*

Runahead buffer**

*[Mutlu et al. ISCA’05]

**[Hashemi et al. MICRO’15]

Runahead Techniques Relative to OoO Core

	Runahead execution*	Runahead buffer**
Flush ROB		

*[Mutlu et al. ISCA’05]

**[Hashemi et al. MICRO’15]

Runahead Techniques Relative to OoO Core

	Runahead execution*	Runahead buffer**
Flush ROB	✓	

*[Mutlu et al. ISCA’05]

**[Hashemi et al. MICRO’15]

Runahead Techniques Relative to OoO Core

	Runahead execution*	Runahead buffer**
Flush ROB	✓	✓

*[Mutlu et al. ISCA’05] **[Hashemi et al. MICRO’15]

Flushing and Re-Filling Incur High Overhead

Flushing and Re-Filling Incur High Overhead

- Front-end refill = 8 cycles

Flushing and Re-Filling Incur High Overhead

- Front-end refill = 8 cycles

- ROB = 192, width = 4
  ROB fill time = 48 cycles

Flushing and Re-Filling Incur High Overhead

- Front-end refill = 8 cycles

- ROB = 192, width = 4
  ROB fill time = 48 cycles

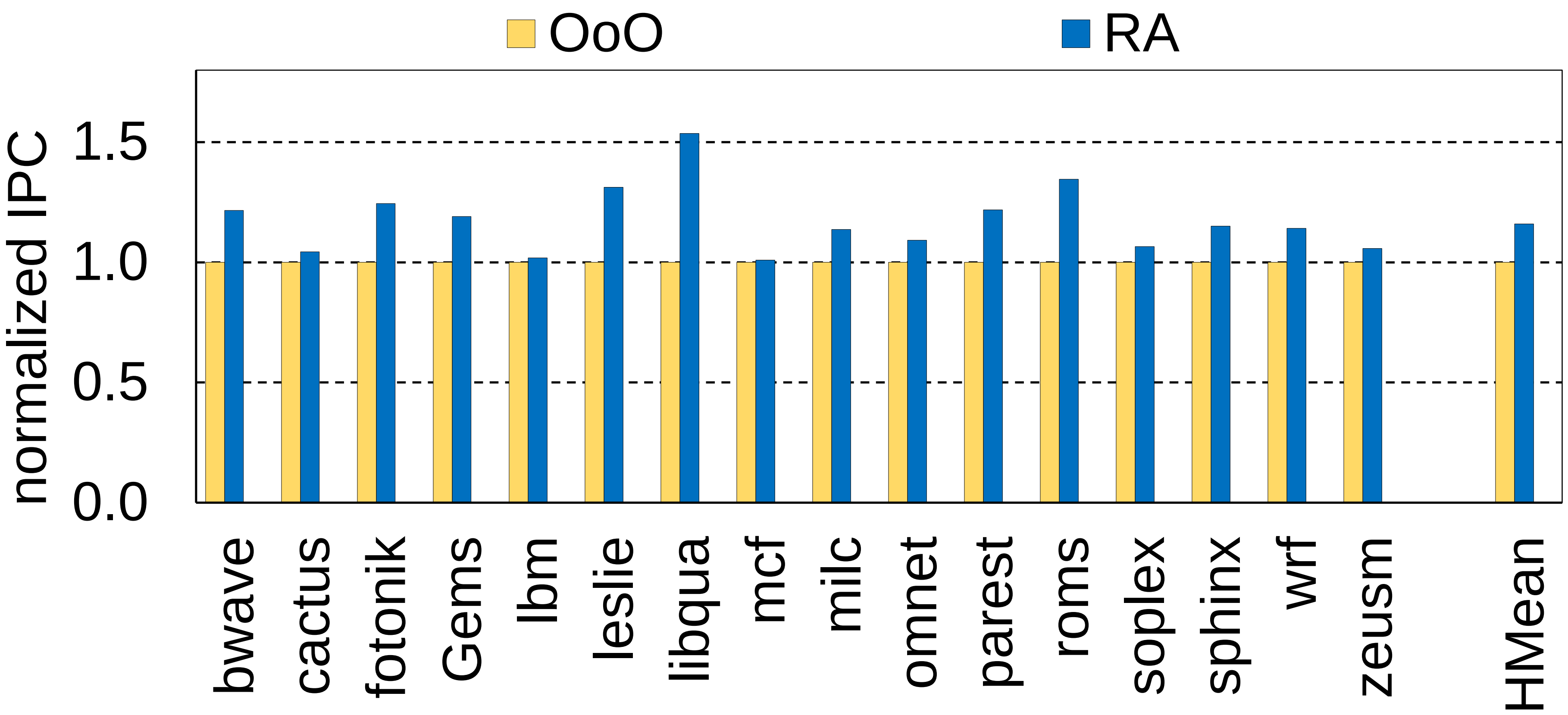
- Total overhead = 56 cycles

Flushing and Re-Filling Incur High Overhead

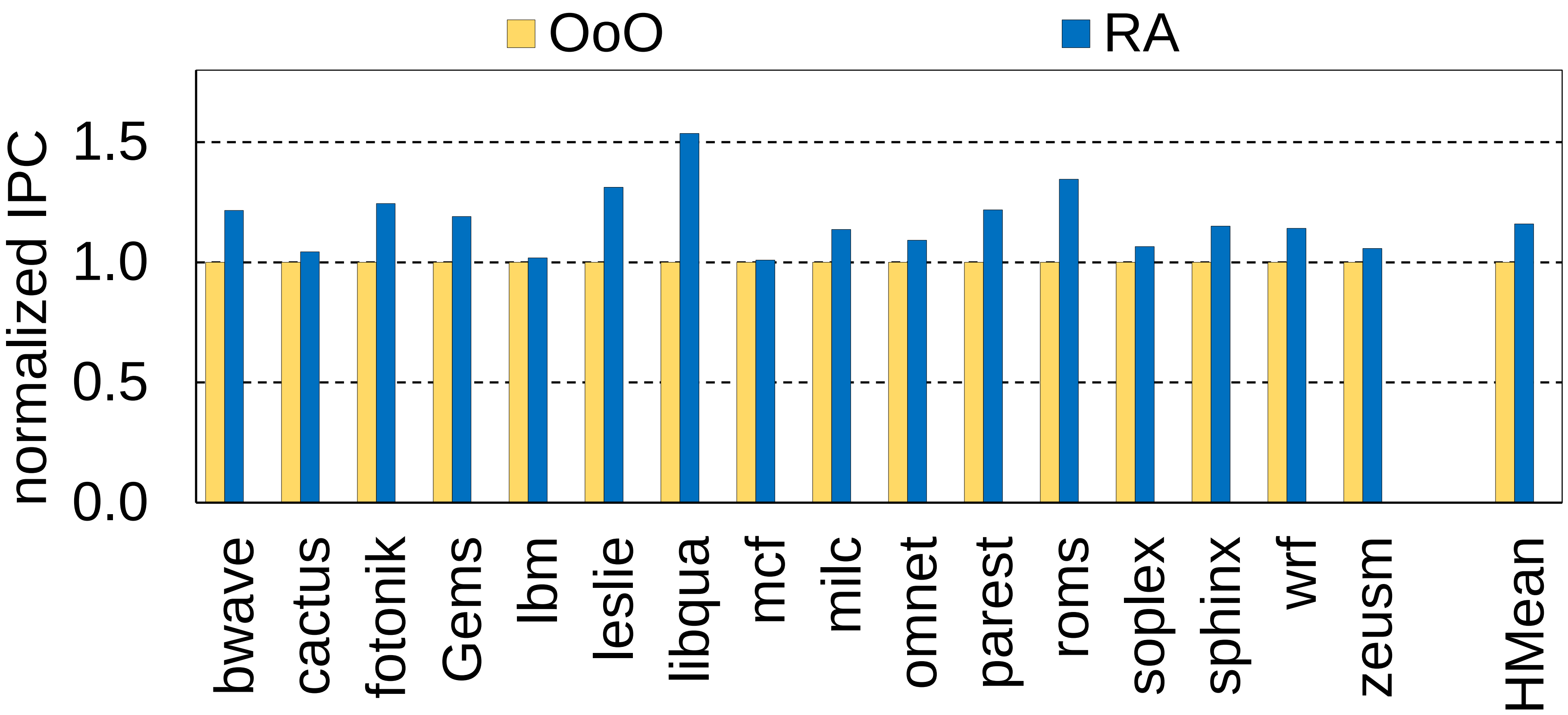
- Front-end refill = 8 cycles
- ROB = 192, width = 4
  ROB fill time = 48 cycles
- Total overhead = 56 cycles

Runahead causes a pipeline bubble of 56 cycles **per invocation**

Flushing and Re-Filling Incur High Overhead

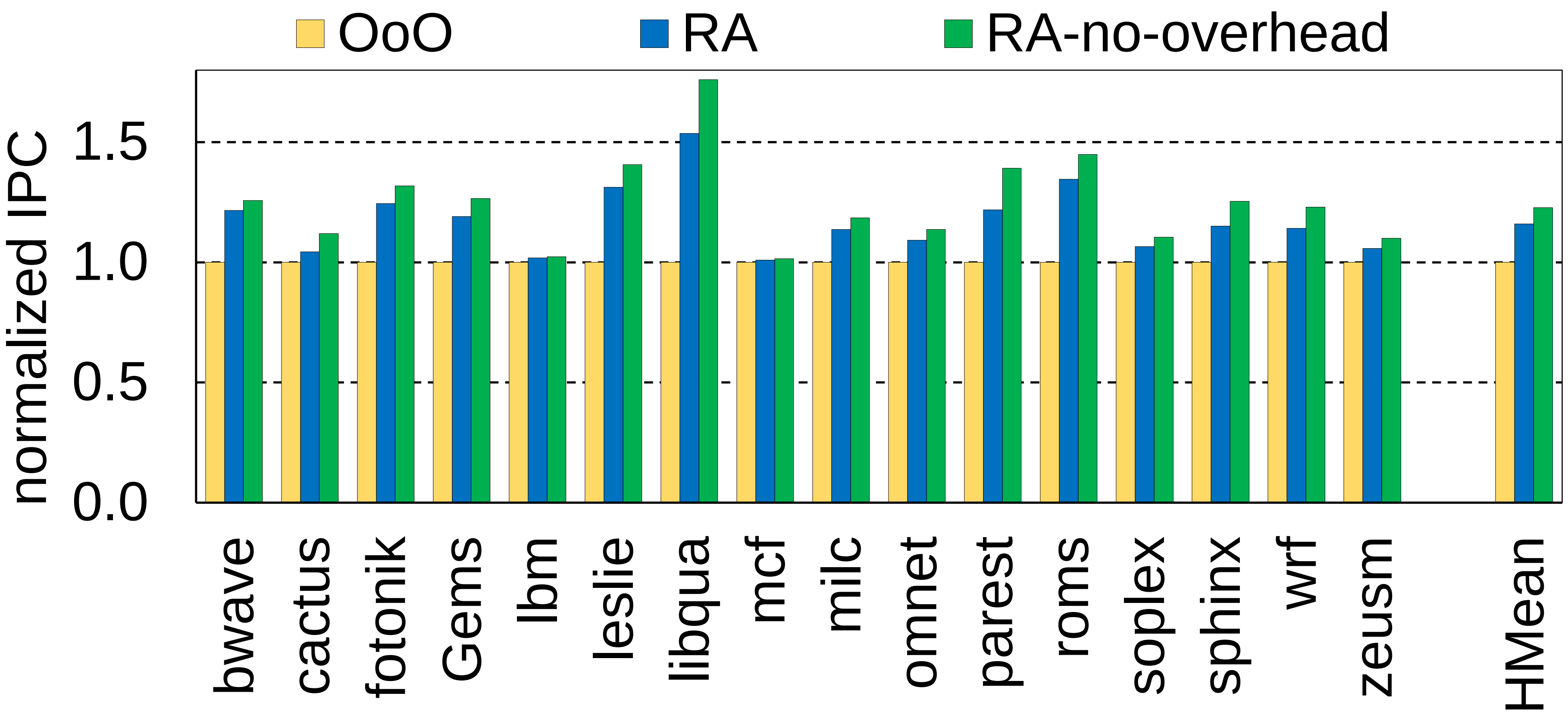


Flushing and Re-Filling Incur High Overhead



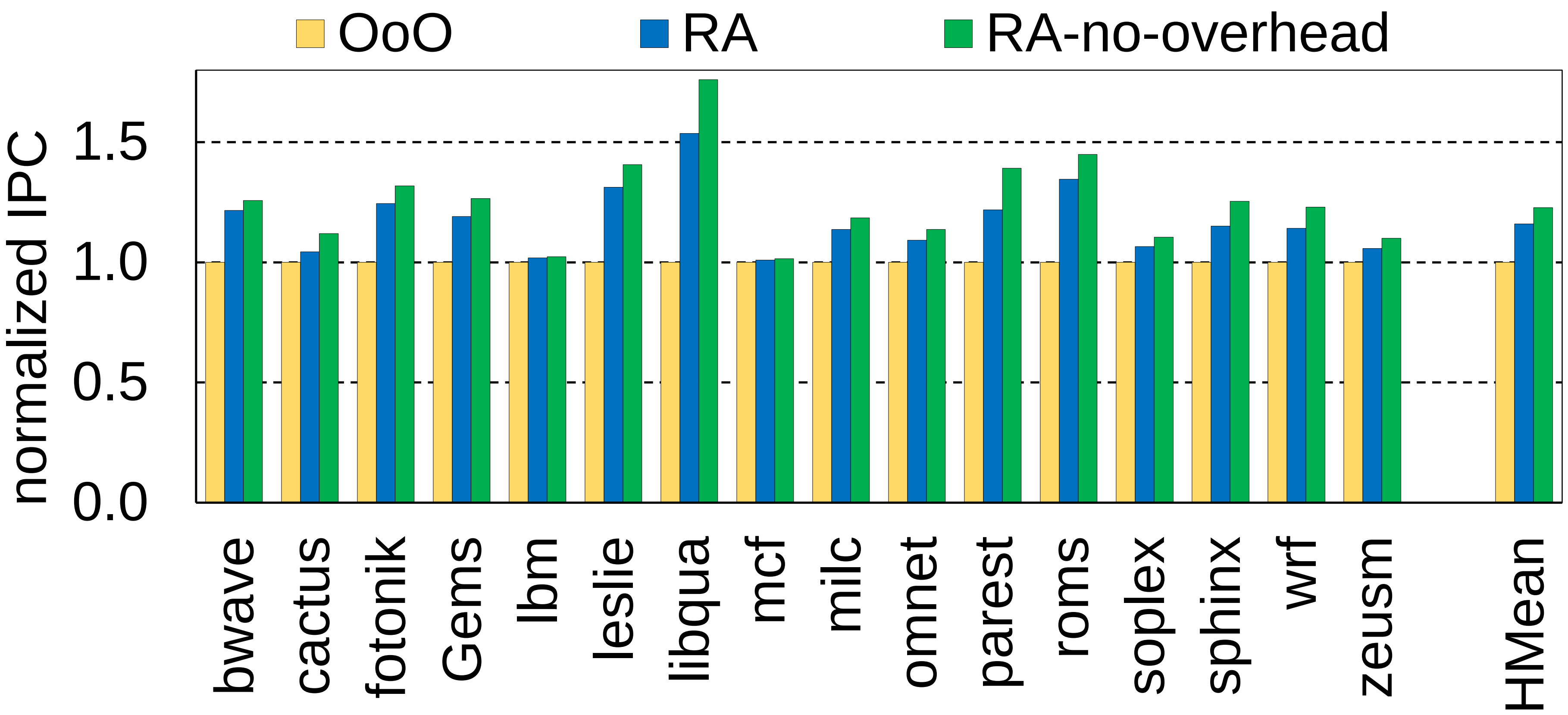
runahead: 15.9%

Flushing and Re-Filling Incur High Overhead



runahead: 15.9%

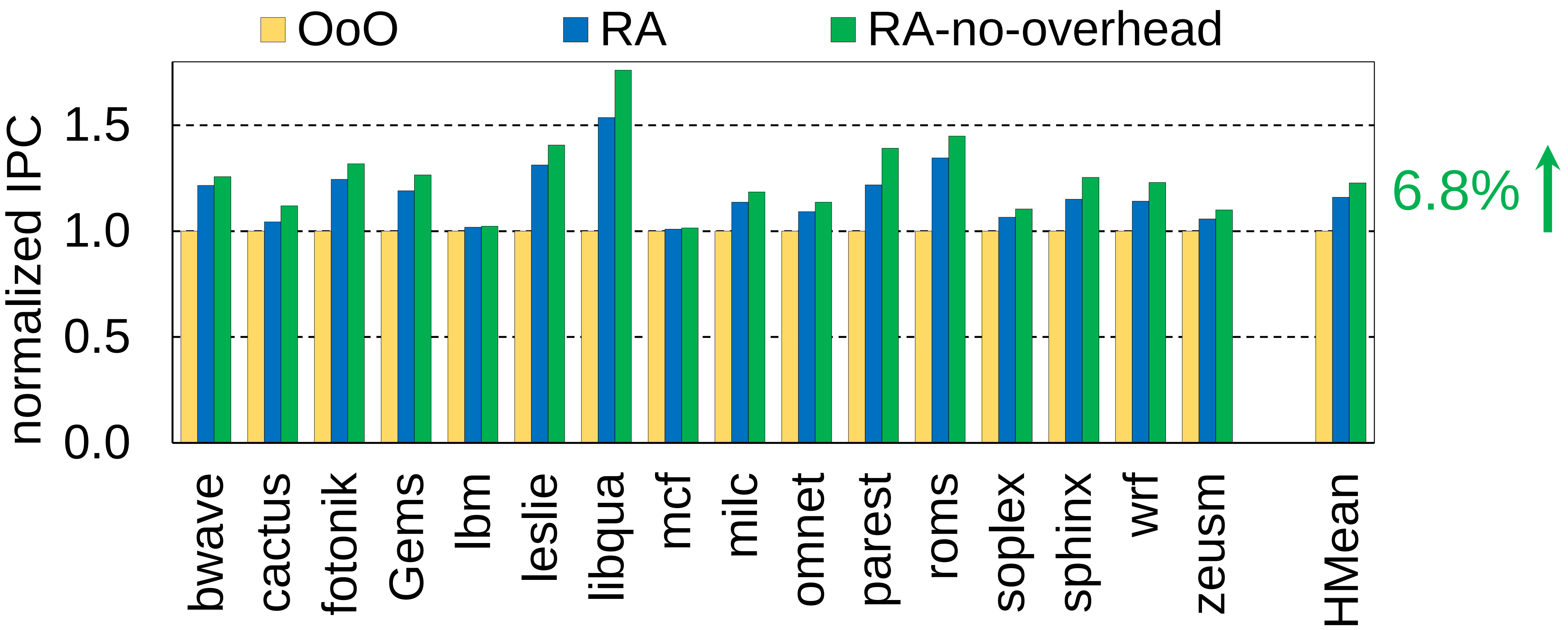
Flushing and Re-Filling Incur High Overhead



runahead: 15.9%

runahead without flushing: 22.7%

Flushing and Re-Filling Incur High Overhead



runahead: 15.9%

runahead without flushing: 22.7%

Runahead Techniques Relative to OoO Core

	Runahead execution*	Runahead buffer**
Flush ROB	✓	✓

*[Mutlu et al. ISCA’05] **[Hashemi et al. MICRO’15]

Runahead Techniques Relative to OoO Core

	Runahead execution*	Runahead buffer**
Flush ROB	✓	✓
Short intervals		

*[Mutlu et al. ISCA’05] **[Hashemi et al. MICRO’15]

Runahead Techniques Relative to OoO Core

	Runahead execution*	Runahead buffer**
Flush ROB	✓	✓
Short intervals	✗	

*[Mutlu et al. ISCA’05] **[Hashemi et al. MICRO’15]

Runahead Techniques Relative to OoO Core

	Runahead execution*	Runahead buffer**
Flush ROB	✓	✓
Short intervals	✗	✗

*[Mutlu et al. ISCA’05]

**[Hashemi et al. MICRO’15]

Runahead Techniques Relative to OoO Core

	Runahead execution*	Runahead buffer**
Flush ROB	✓	✓
Short intervals	✗	✗
Instructions executed		

*[Mutlu et al. ISCA'05]

**[Hashemi et al. MICRO'15]

Runahead Techniques Relative to OoO Core

	Runahead execution*	Runahead buffer**
Flush ROB	✓	✓
Short intervals	✗	✗
Instructions executed	**All**	

*[Mutlu et al. ISCA’05]

**[Hashemi et al. MICRO’15]

Runahead Techniques Relative to OoO Core

	Runahead execution*	Runahead buffer**
Flush ROB	✓	✓
Short intervals	✗	✗
Instructions executed	**All**	Only one slice

*[Mutlu et al. ISCA'05] **[Hashemi et al. MICRO'15]

Runahead Techniques Provide Limited Prefetch Coverage

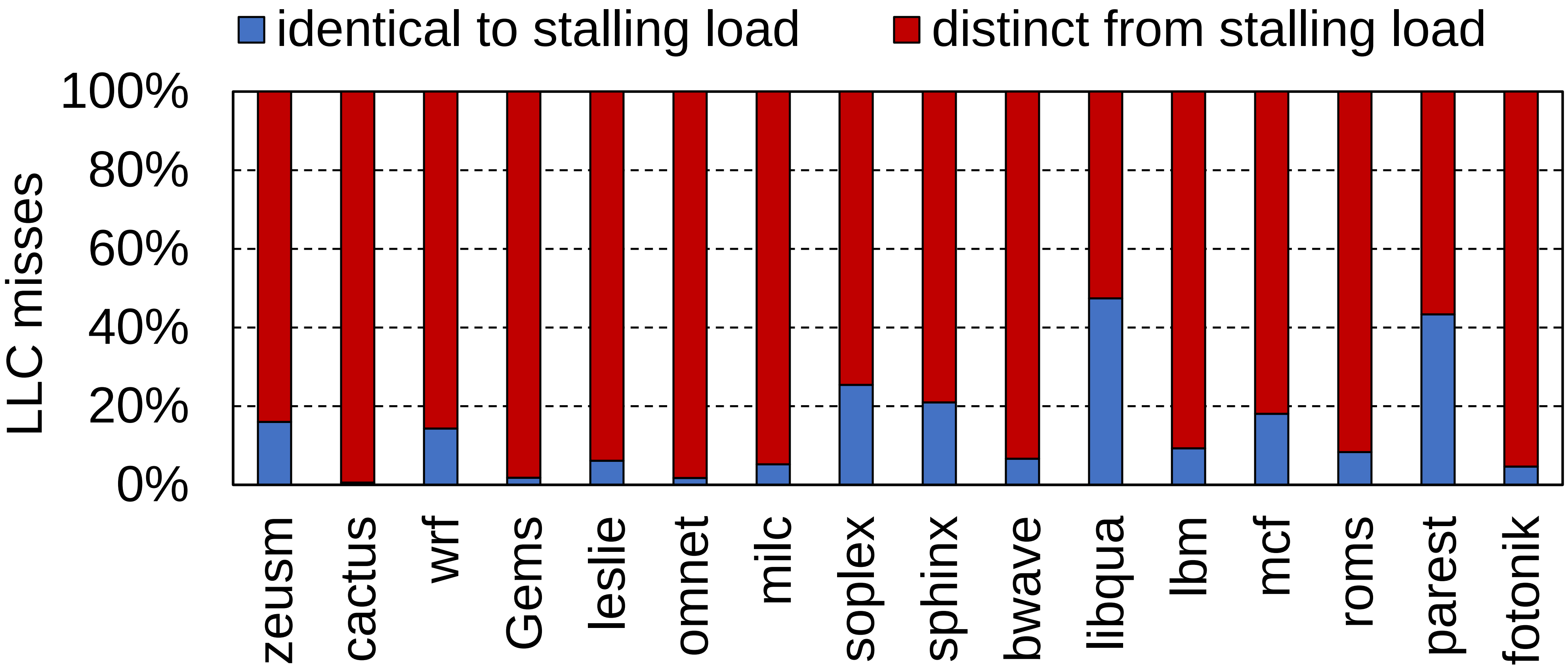
- Runahead execution: Executes **useless** instructions

Runahead Techniques Provide Limited Prefetch Coverage

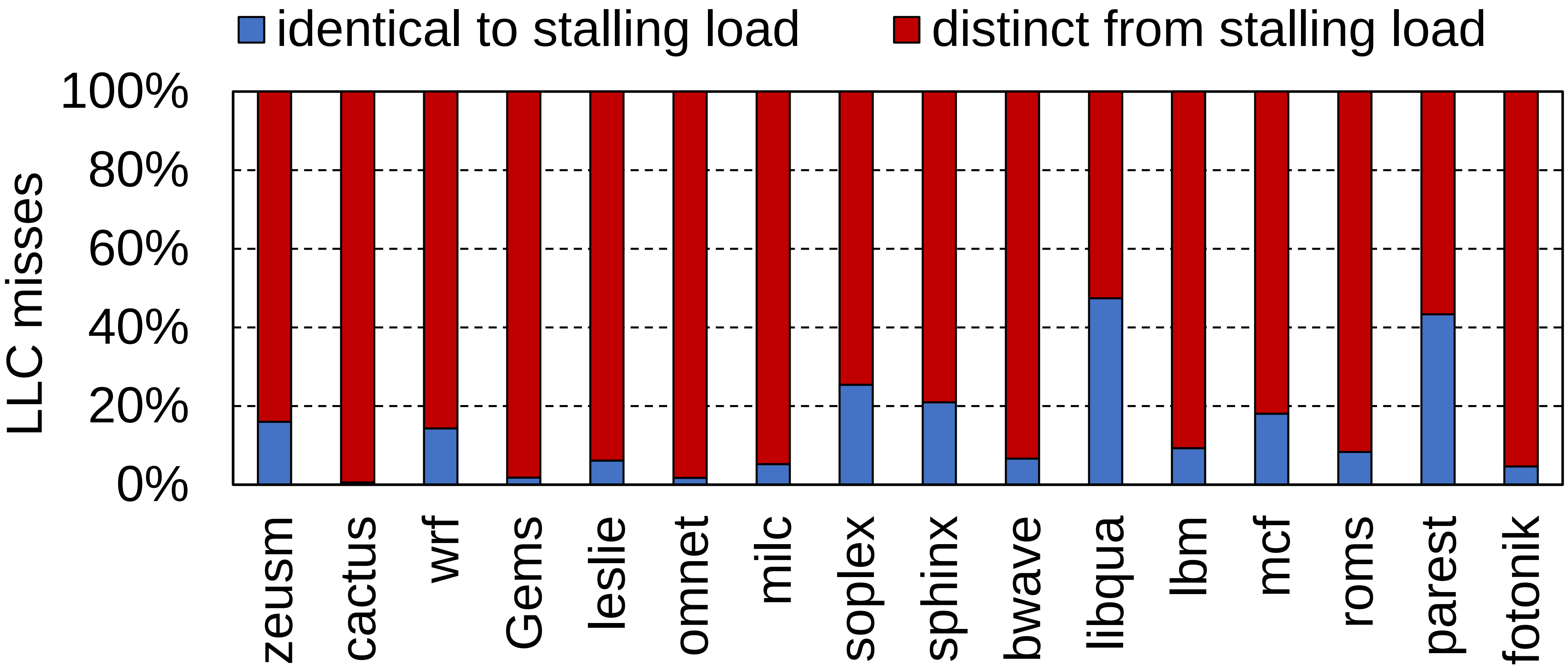
- Runahead execution: Executes **useless** instructions

- Runahead buffer: High coverage for **only one slice**

Only One Load does not Lead to Majority of Memory Accesses

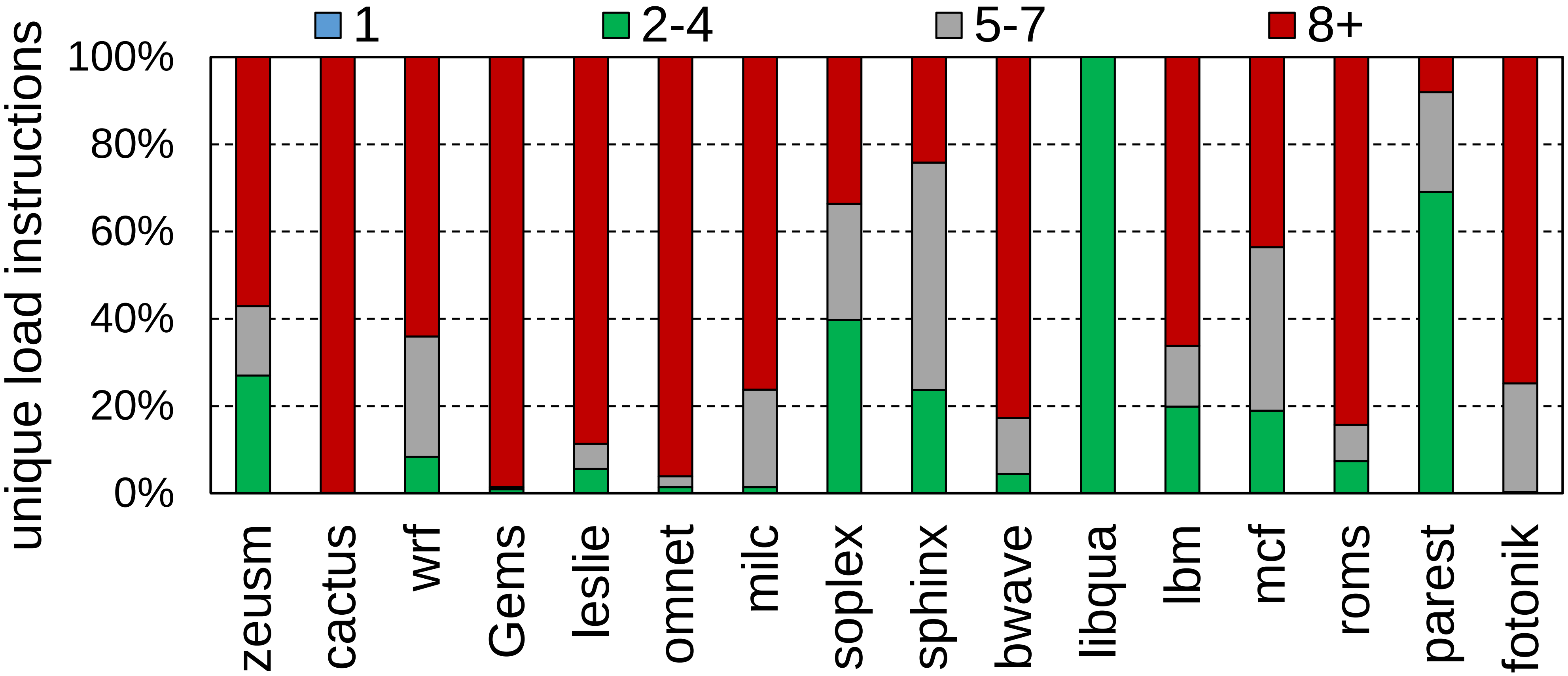


Only One Load does not Lead to Majority of Memory Accesses

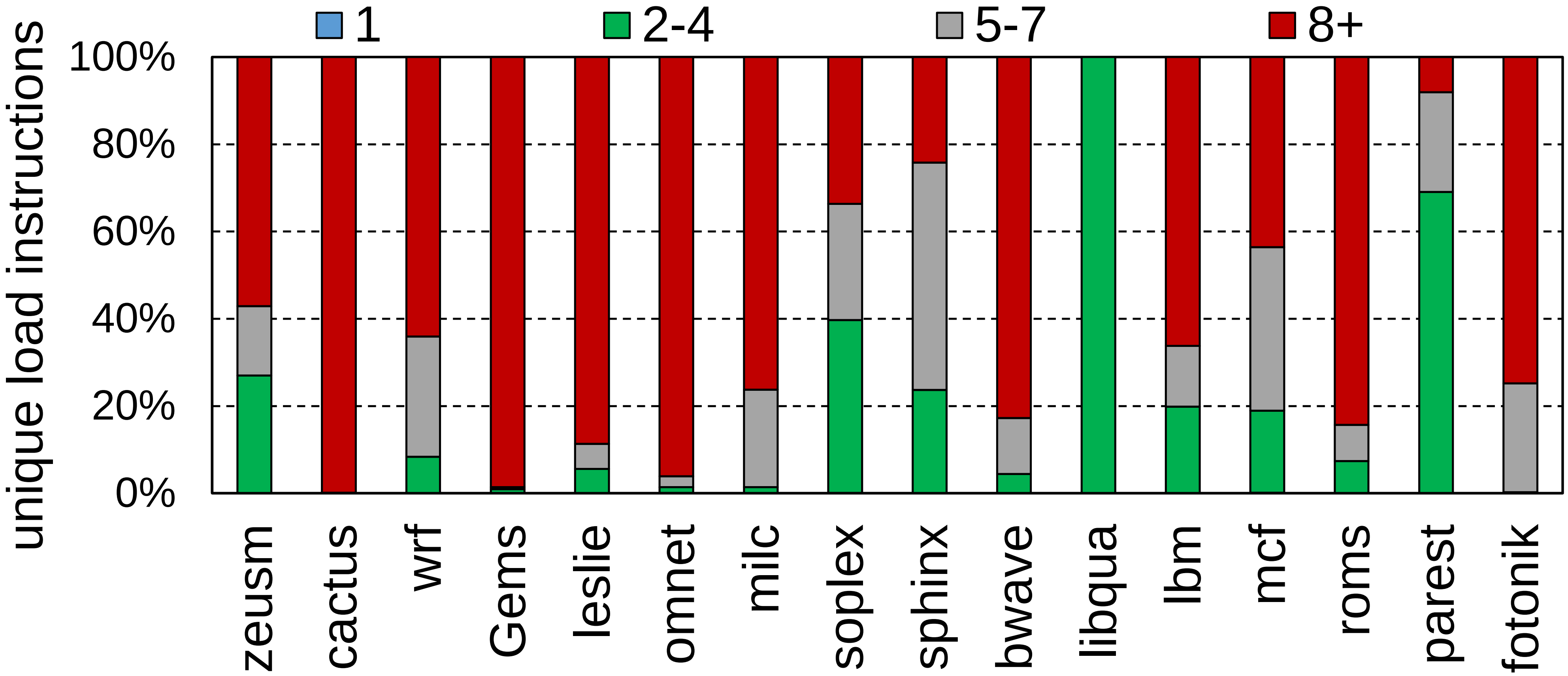


Most of the long-latency loads during runahead differ from the stalling load

Applications Access Memory through Multiple Slices



Applications Access Memory through Multiple Slices



There are more than eight unique load instructions accessing memory during each runahead interval

Runahead Techniques Relative to OoO Core

	Runahead execution*	Runahead buffer**
Flush ROB	✓	✓
Short intervals	✗	✗
Instructions executed	All	Only one slice

*[Mutlu et al. ISCA’ 05] **[Hashemi et al. MICRO’ 15]

Runahead Techniques Relative to OoO Core

	Runahead execution*	Runahead buffer**
Flush ROB	✓	✓
Short intervals	✗	✗
Instructions executed	All	Only one slice
Performance		

*[Mutlu et al. ISCA’ 05] **[Hashemi et al. MICRO’ 15]

Runahead Techniques Relative to OoO Core

	Runahead execution*	Runahead buffer**
Flush ROB	✓	✓
Short intervals	✗	✗
Instructions executed	All	Only one slice
Performance	High ↑	

*[Mutlu et al. ISCA’ 05] **[Hashemi et al. MICRO’ 15]

Runahead Techniques Relative to OoO Core

	Runahead execution*	Runahead buffer**
Flush ROB	✓	✓
Short intervals	✗	✗
Instructions executed	All	Only one slice
Performance	High ↑	High ↑

*[Mutlu et al. ISCA’ 05] **[Hashemi et al. MICRO’ 15]

Runahead Techniques Relative to OoO Core

	Runahead execution*	Runahead buffer**
Flush ROB	✓	✓
Short intervals	✗	✗
Instructions executed	All	Only one slice
Performance	High ↑	High ↑
Energy-Efficiency		

*[Mutlu et al. ISCA' 05]

**[Hashemi et al. MICRO' 15]

Runahead Techniques Relative to OoO Core

	Runahead execution*	Runahead buffer**
Flush ROB	✓	✓
Short intervals	✗	✗
Instructions executed	All	Only one slice
Performance	High ↑	High ↑
Energy-Efficiency	Low ↓	

*[Mutlu et al. ISCA’ 05] **[Hashemi et al. MICRO’ 15]

Runahead Techniques Relative to OoO Core

	Runahead execution*	Runahead buffer**
Flush ROB	✓	✓
Short intervals	✗	✗
Instructions executed	All	Only one slice
Performance	High ↑	High ↑
Energy-Efficiency	Low ↓	Same

*[Mutlu et al. ISCA’ 05]

**[Hashemi et al. MICRO’ 15]

Runahead Techniques Relative to OoO Core

	Runahead execution*	Runahead buffer**	
Flush ROB	✓	✓	
Short intervals	✗	✗	
Instructions executed	All	Only one slice	
Performance	High ↑	High ↑	
Energy-Efficiency	Low ↓	Same	

*[Mutlu et al. ISCA’ 05]

**[Hashemi et al. MICRO’ 15]

Runahead Techniques Relative to OoO Core

	Runahead execution*	Runahead buffer**	
Flush ROB	✓	✓	✗
Short intervals	✗	✗	
Instructions executed	All	Only one slice	
Performance	High ↑	High ↑	
Energy-Efficiency	Low ↓	Same	

*[Mutlu et al. ISCA’ 05]

**[Hashemi et al. MICRO’ 15]

Runahead Techniques Relative to OoO Core

	Runahead execution*	Runahead buffer**	
Flush ROB	✓	✓	✗
Short intervals	✗	✗	✓
Instructions executed	All	Only one slice	
Performance	High ↑	High ↑	
Energy-Efficiency	Low ↓	Same	

*[Mutlu et al. ISCA’ 05]

**[Hashemi et al. MICRO’ 15]

Runahead Techniques Relative to OoO Core

	Runahead execution*	Runahead buffer**	
Flush ROB	✓	✓	✗
Short intervals	✗	✗	✓
Instructions executed	All	Only one slice	All slices
Performance	High ↑	High ↑	
Energy-Efficiency	Low ↓	Same	

*[Mutlu et al. ISCA’ 05] **[Hashemi et al. MICRO’ 15]

Runahead Techniques Relative to OoO Core

	Runahead execution*	Runahead buffer**	
Flush ROB	✓	✓	✗
Short intervals	✗	✗	✓
Instructions executed	All	Only one slice	All slices
Performance	High ↑	High ↑	Very high ↑↑
Energy-Efficiency	Low ↓	Same	

*[Mutlu et al. ISCA’ 05]

**[Hashemi et al. MICRO’ 15]

Runahead Techniques Relative to OoO Core

	Runahead execution*	Runahead buffer**	
Flush ROB	✓	✓	✗
Short intervals	✗	✗	✓
Instructions executed	All	Only one slice	All slices
Performance	High ↑	High ↑	Very high ↑↑
Energy-Efficiency	Low ↓	Same	High ↑

*[Mutlu et al. ISCA’ 05] **[Hashemi et al. MICRO’ 15]

Runahead Techniques Relative to OoO Core

	Runahead execution*	Runahead buffer**	Precise runahead***
Flush ROB	✓	✓	✗
Short intervals	✗	✗	✓
Instructions executed	All	Only one slice	All slices
Performance	High ↑	High ↑	Very high ↑↑
Energy-Efficiency	Low ↓	Same	High ↑

*[Mutlu et al. ISCA' 05] **[Hashemi et al. MICRO' 15] ***[Naithani et al. HPCA' 20]

Precise Runahead Execution (PRE)

Precise Runahead Execution (PRE)

Key insight: There are sufficient resources to (start) run ahead without flushing the ROB

Precise Runahead Execution (PRE)

Key insight: There are sufficient resources to (start) run ahead without flushing the ROB

When running ahead:

Precise Runahead Execution (PRE)

Key insight: There are sufficient resources to (start) run ahead without flushing the ROB

When running ahead:

1. Executes only useful instructions in runahead mode

Precise Runahead Execution (PRE)

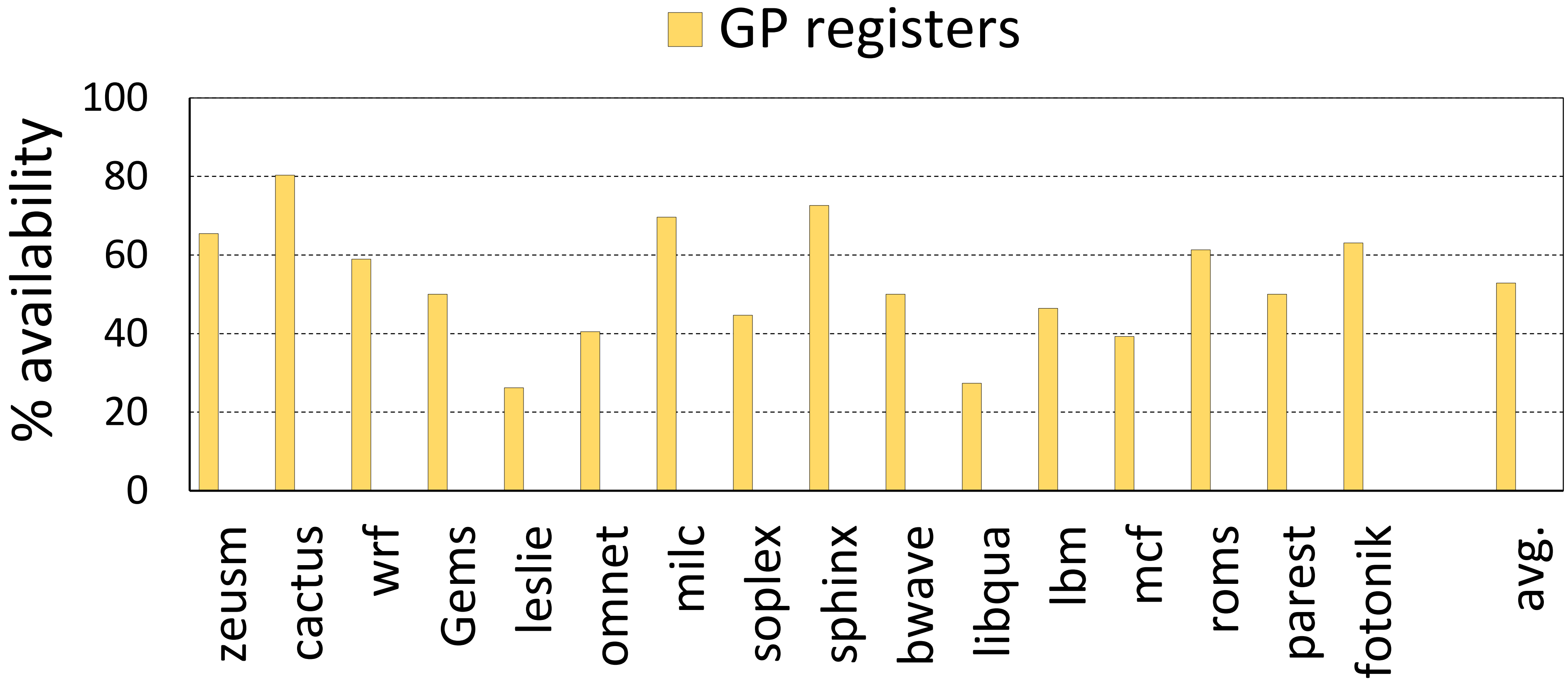
Key insight: There are sufficient resources to (start) run ahead without flushing the ROB

When running ahead:

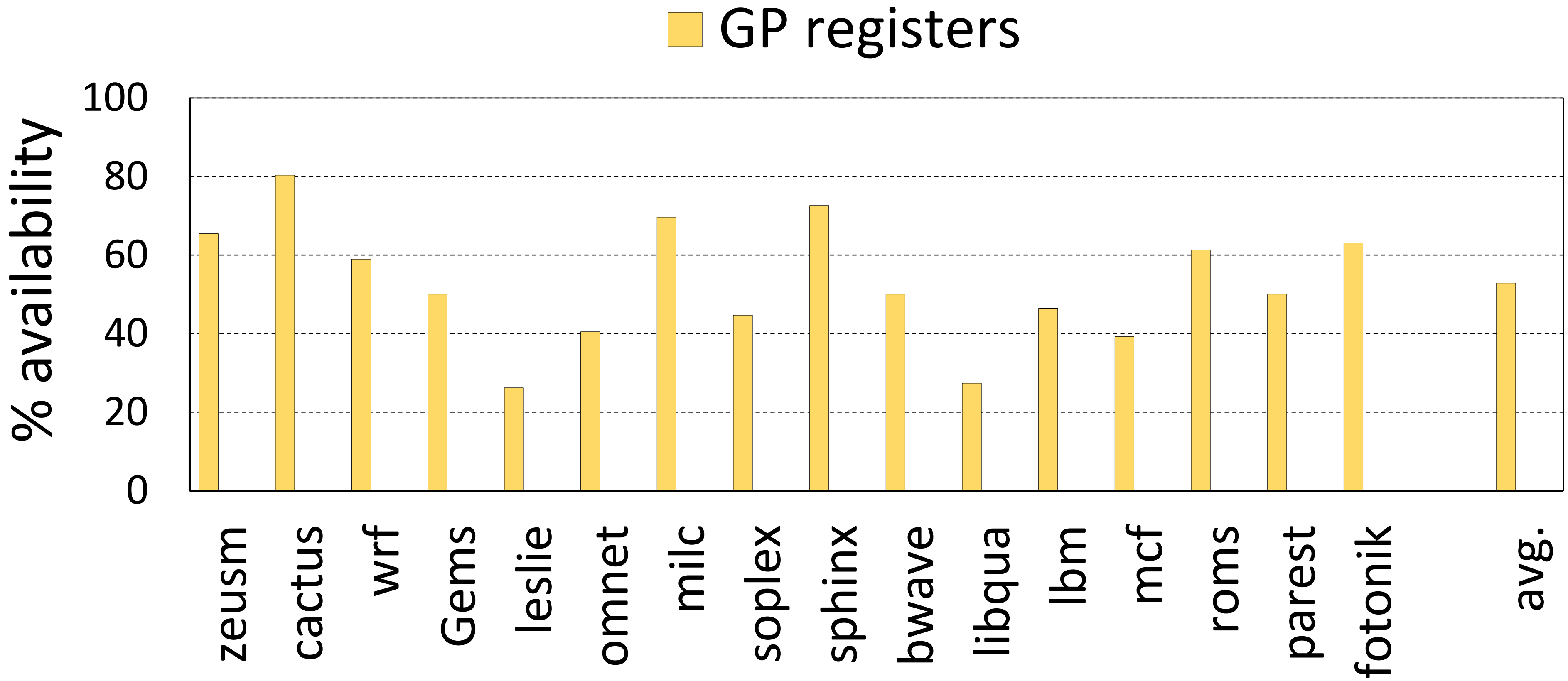
1. Executes only useful instructions in runahead mode

2. Efficiently manages microarchitectural resources

Processor Resources at Full-Window Stall

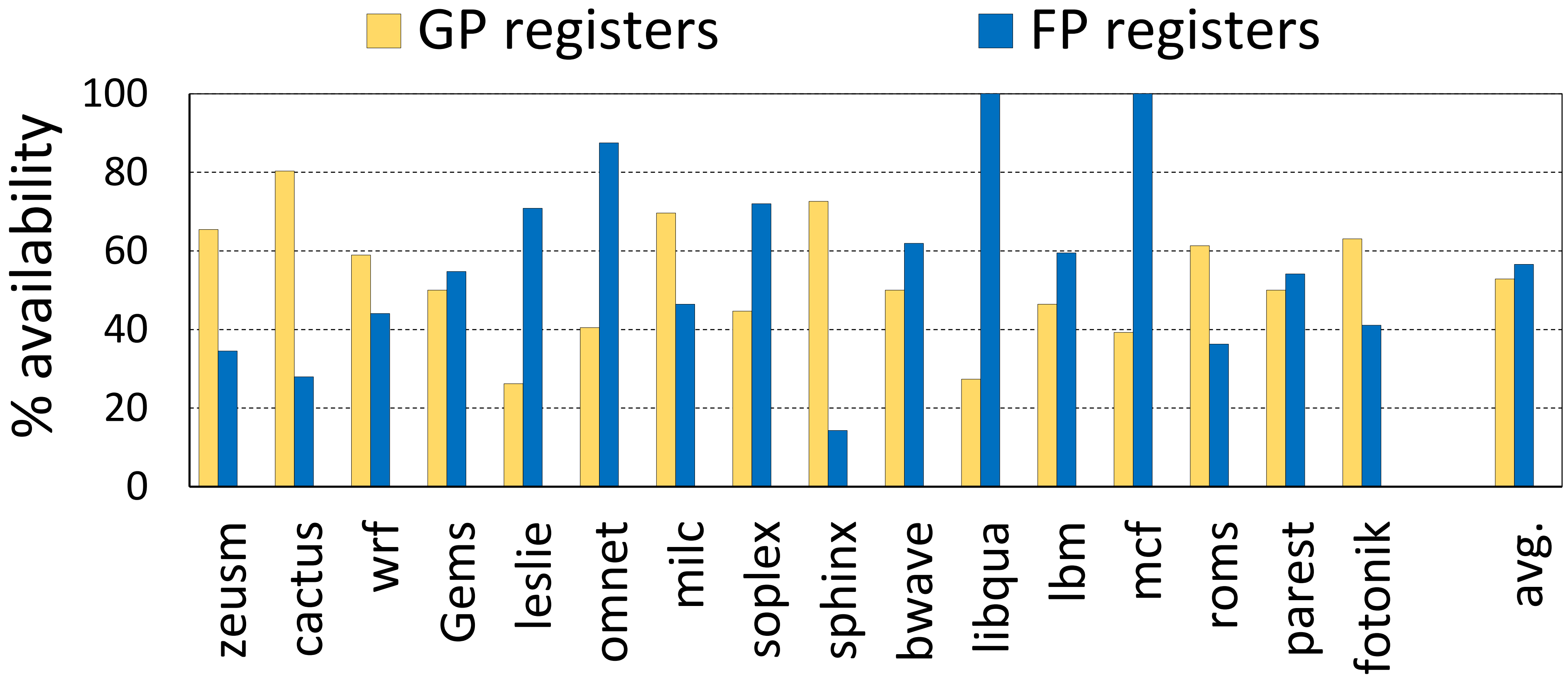


Processor Resources at Full-Window Stall



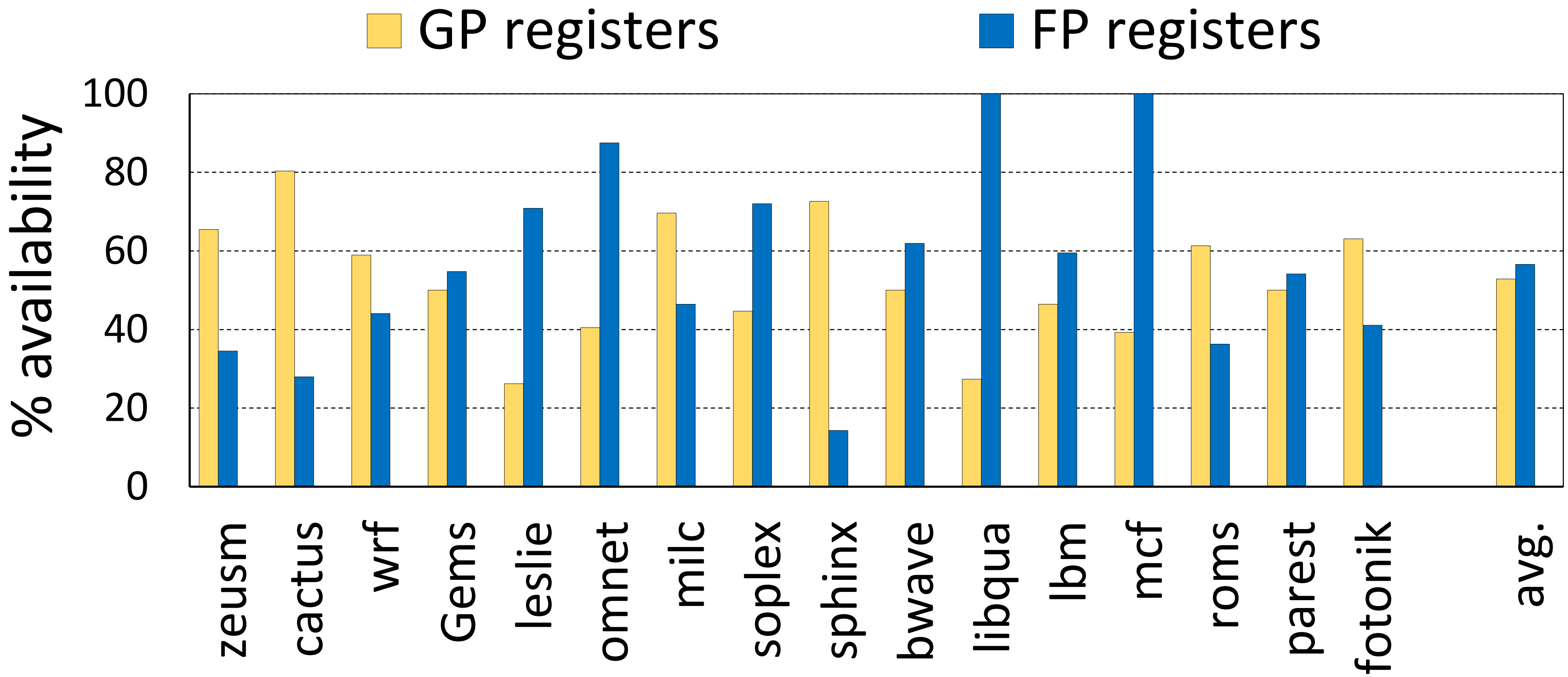
GP registers: **52%**

Processor Resources at Full-Window Stall



GP registers: **52%**

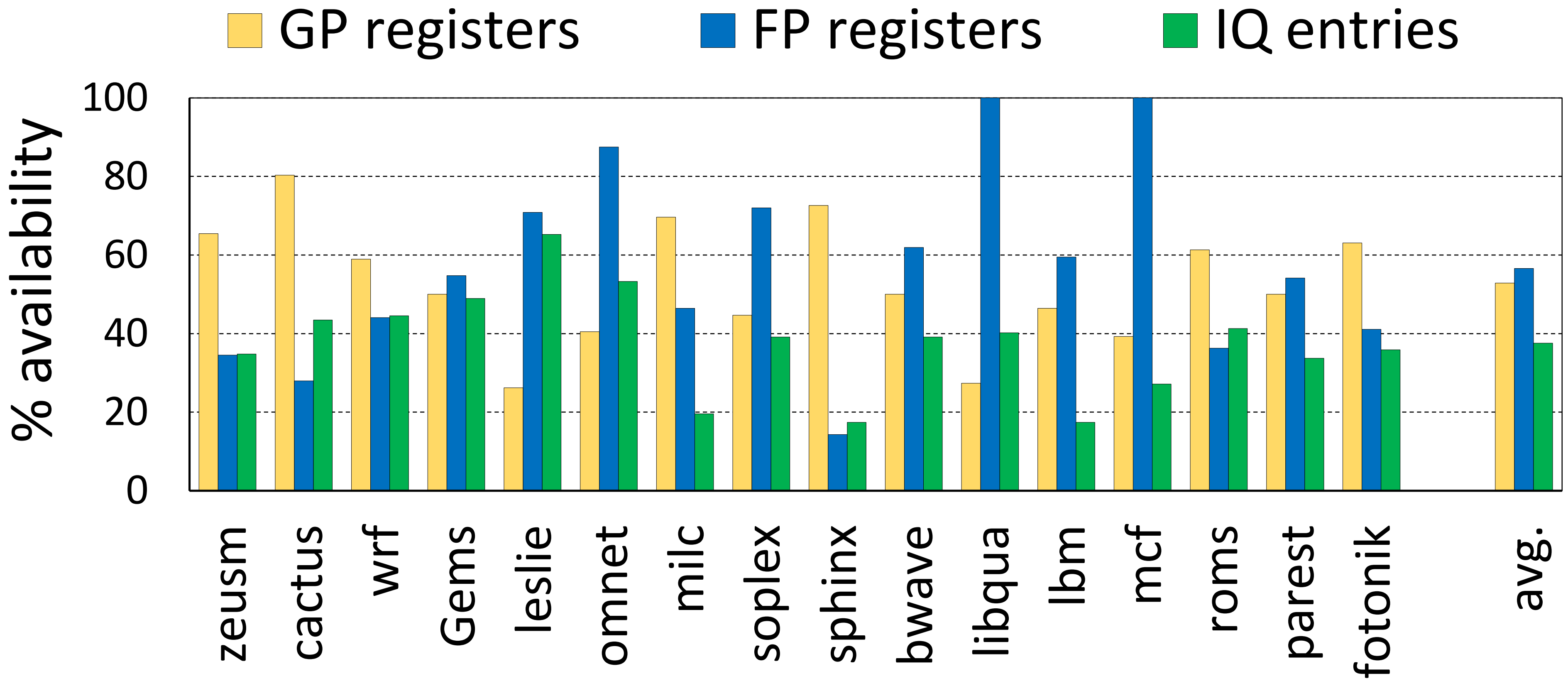
Processor Resources at Full-Window Stall



GP registers: **52%**

FP registers: **56%**

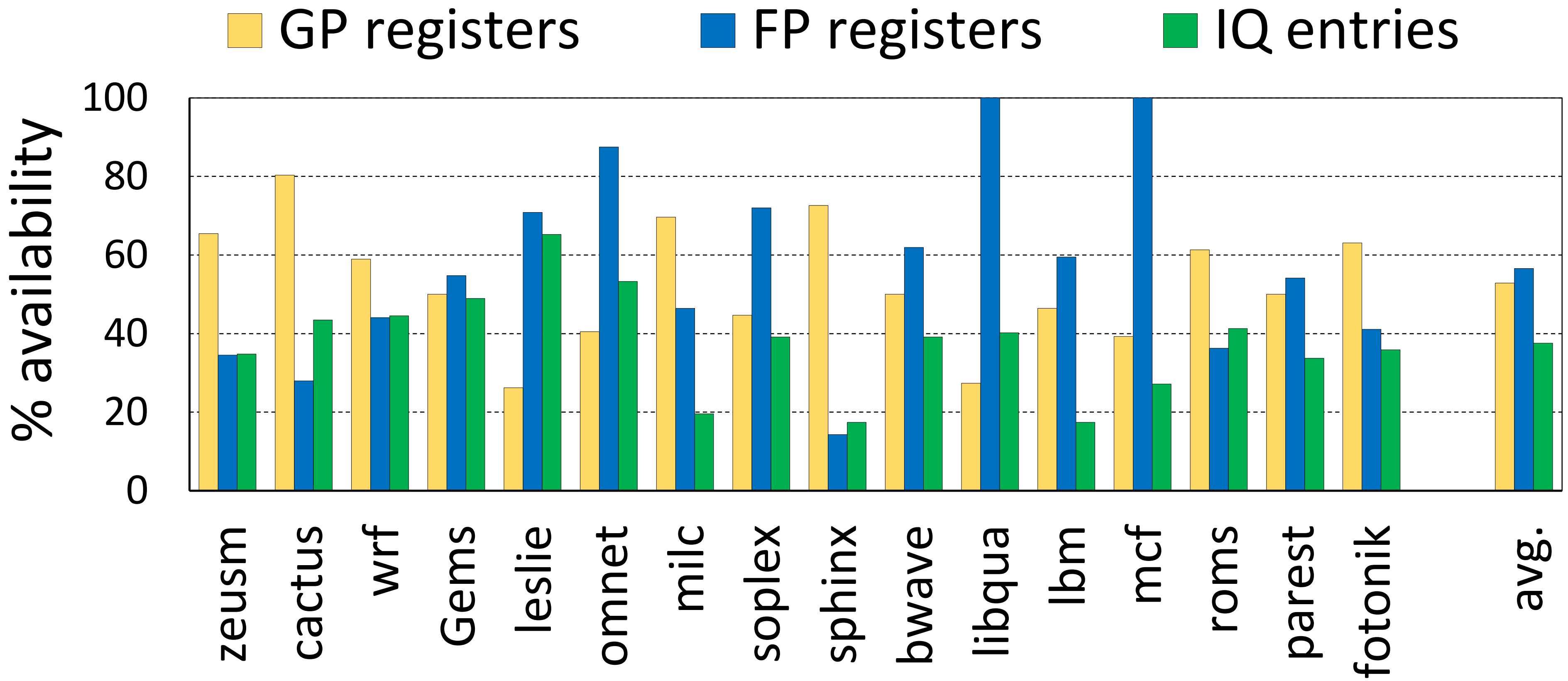
Processor Resources at Full-Window Stall



GP registers: **52%**

FP registers: **56%**

Processor Resources at Full-Window Stall

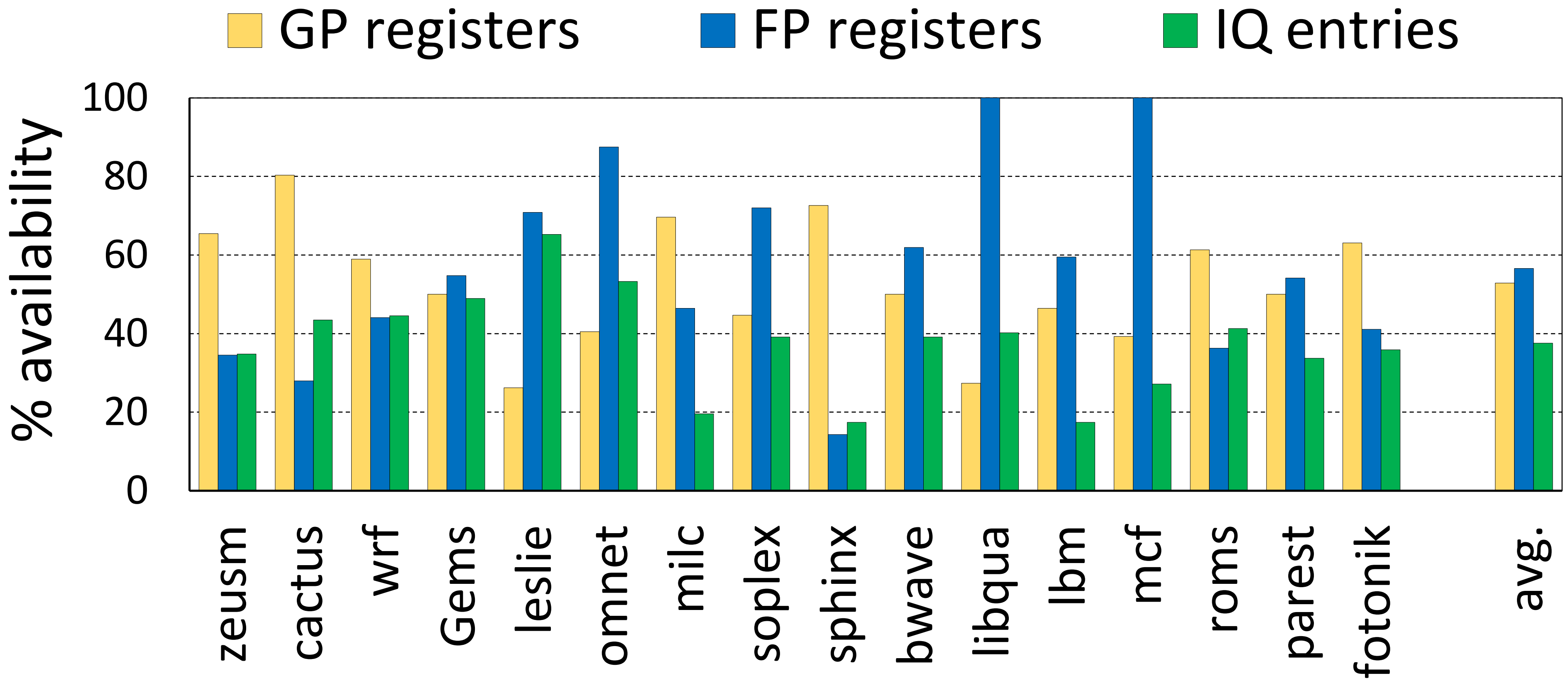


GP registers: **52%**

FP registers: **56%**

IQ entries: **37%**

Processor Resources at Full-Window Stall



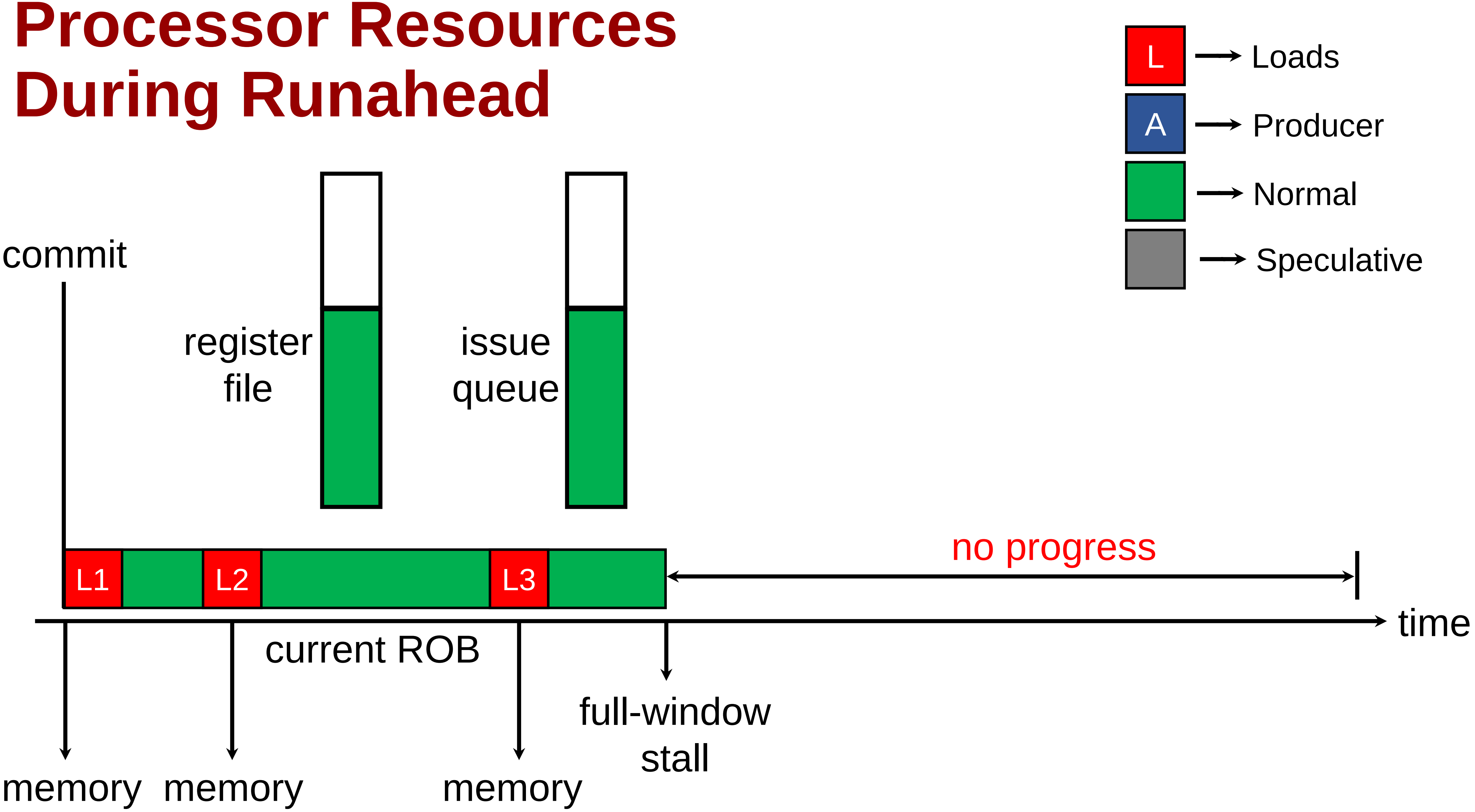
GP registers: **52%**

FP registers: **56%**

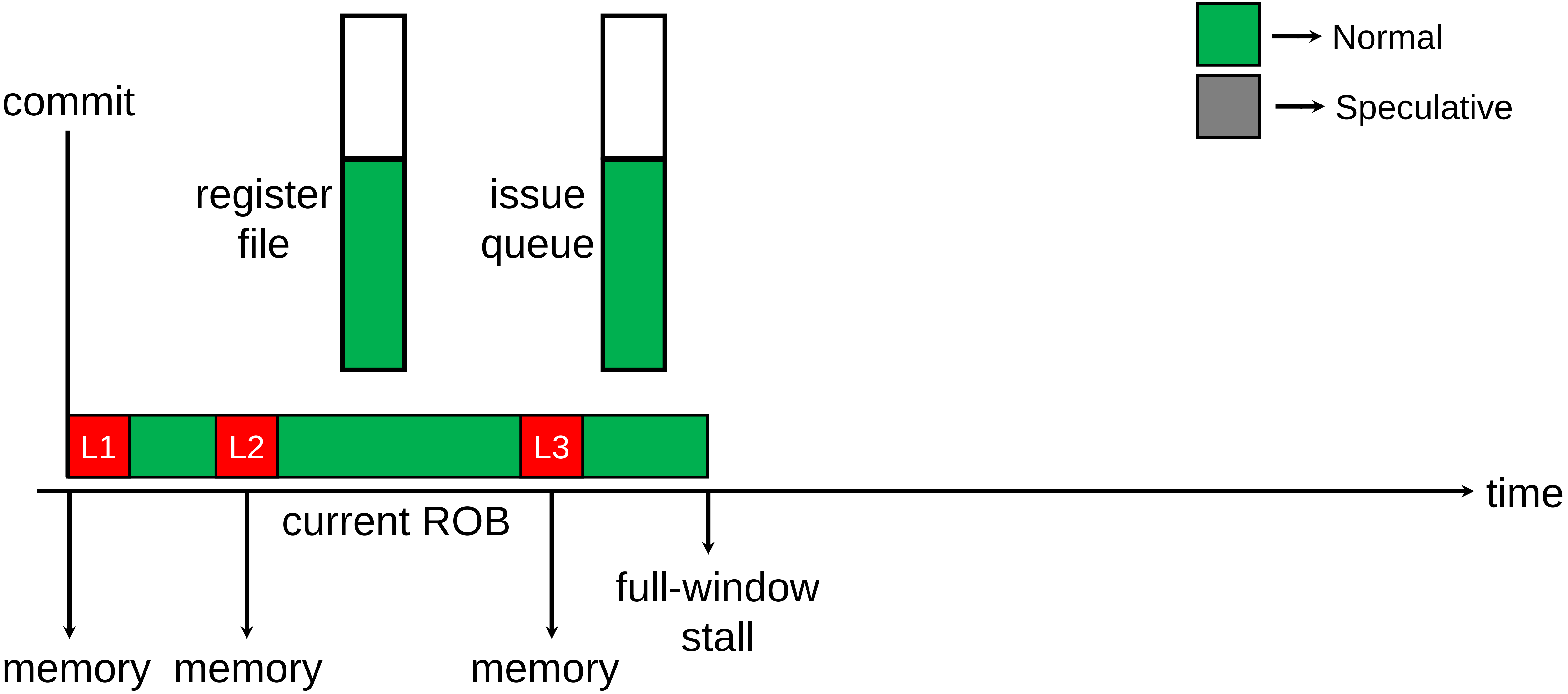
IQ entries: **37%**

There are sufficient resources to start runahead without flushing the ROB

Processor Resources During Runahead



Processor Resources During Runahead



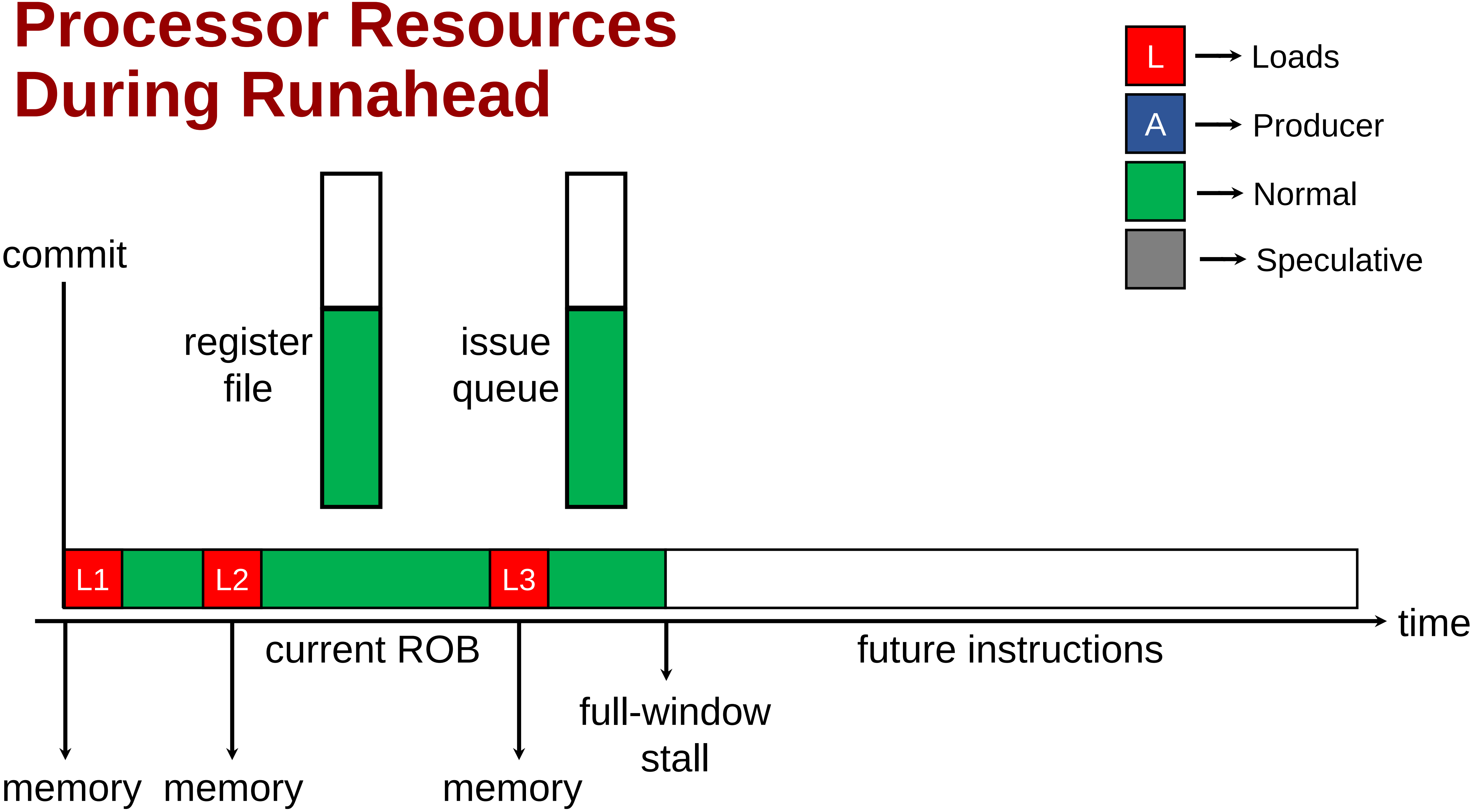
L → Loads

A → Producer

🟩 → Normal

⬜ → Speculative

Processor Resources During Runahead



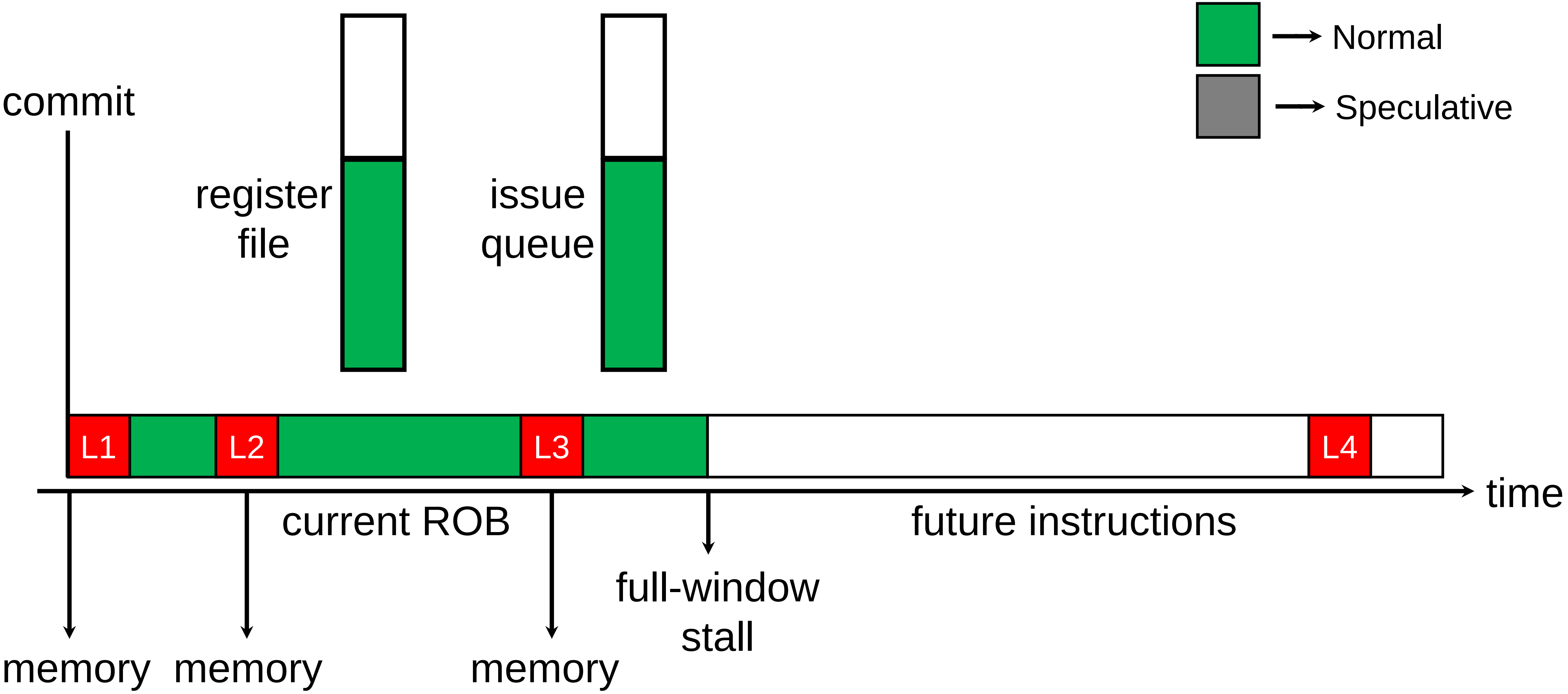
Processor Resources During Runahead

🟥 L → Loads

🟦 A → Producer

🟩 → Normal

⬛ → Speculative



commit

register file

issue queue

L1

L2

L3

L4

time

current ROB

future instructions

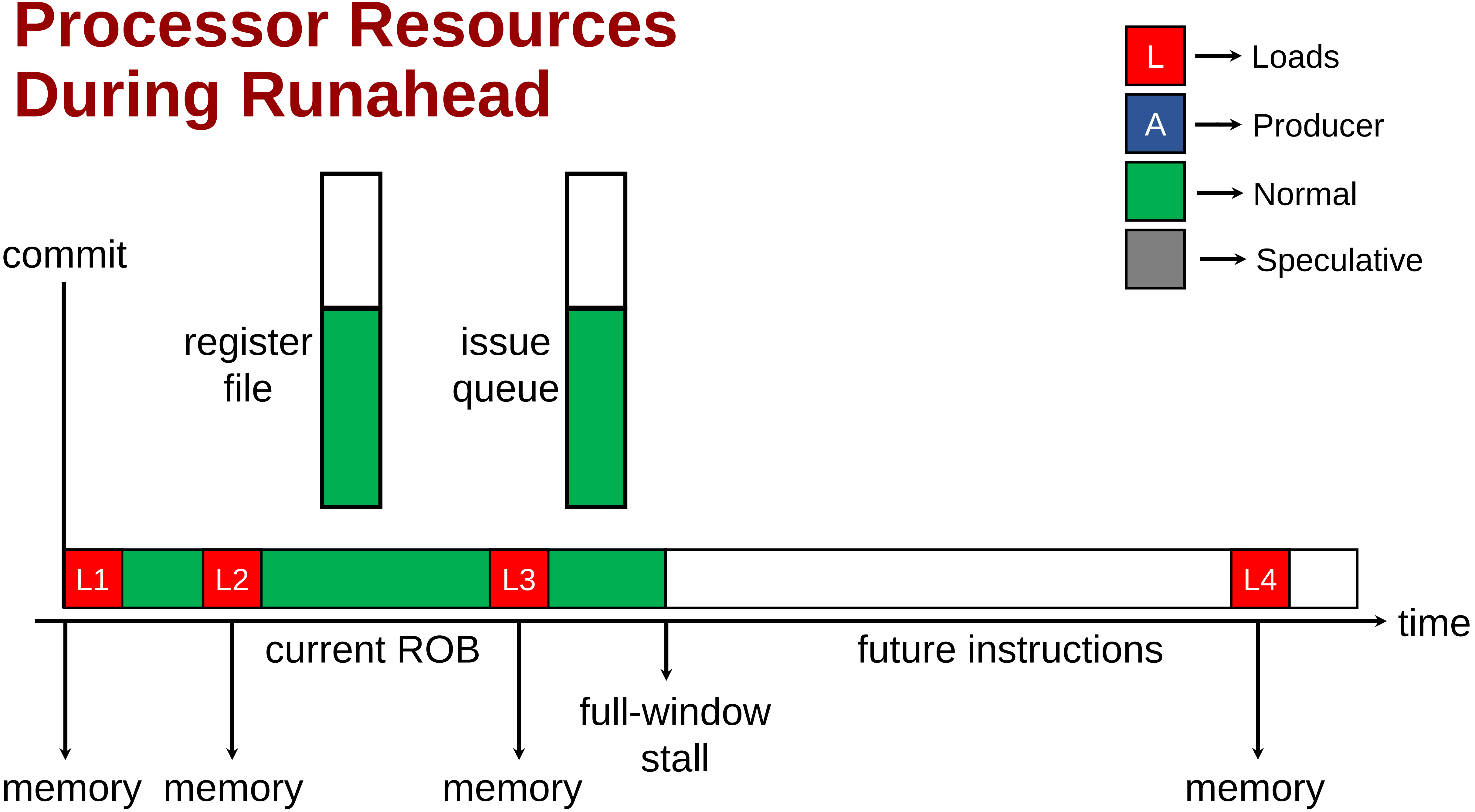
full-window stall

memory

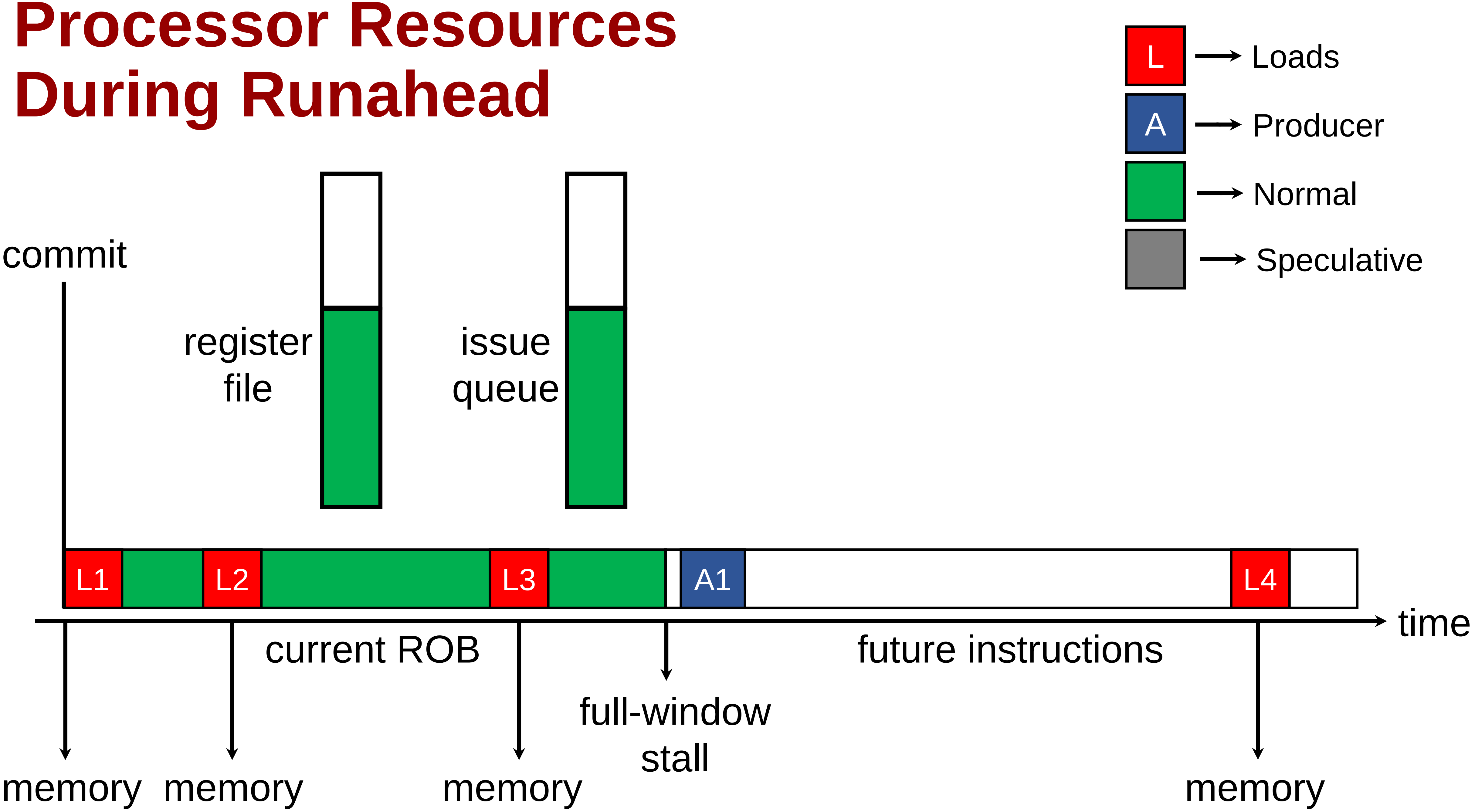
memory

memory

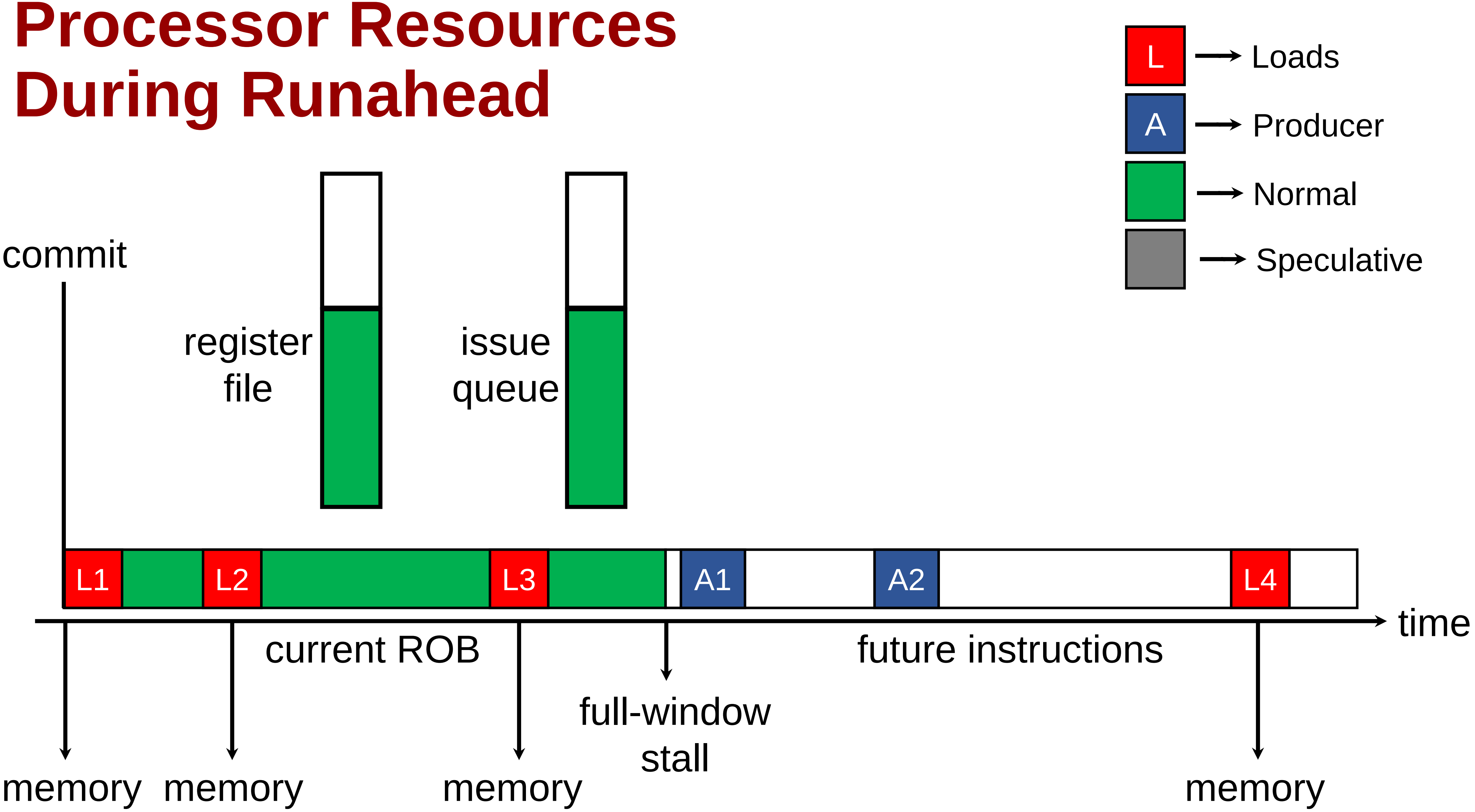
Processor Resources During Runahead



Processor Resources During Runahead



Processor Resources During Runahead



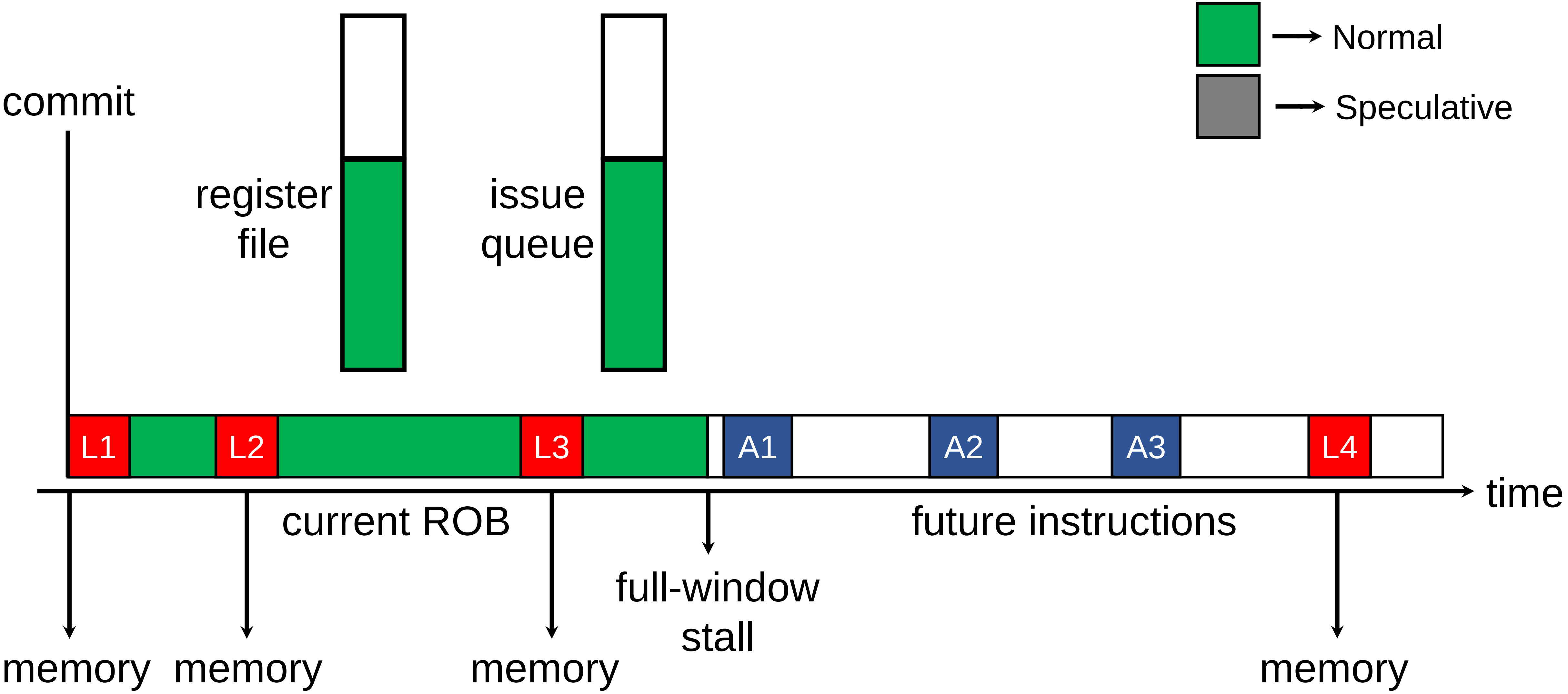
Processor Resources During Runahead

🟥 L → Loads

🟦 A → Producer

🟩 → Normal

⬜ → Speculative



commit

register file

issue queue

L1 L2 L3 A1 A2 A3 L4

time

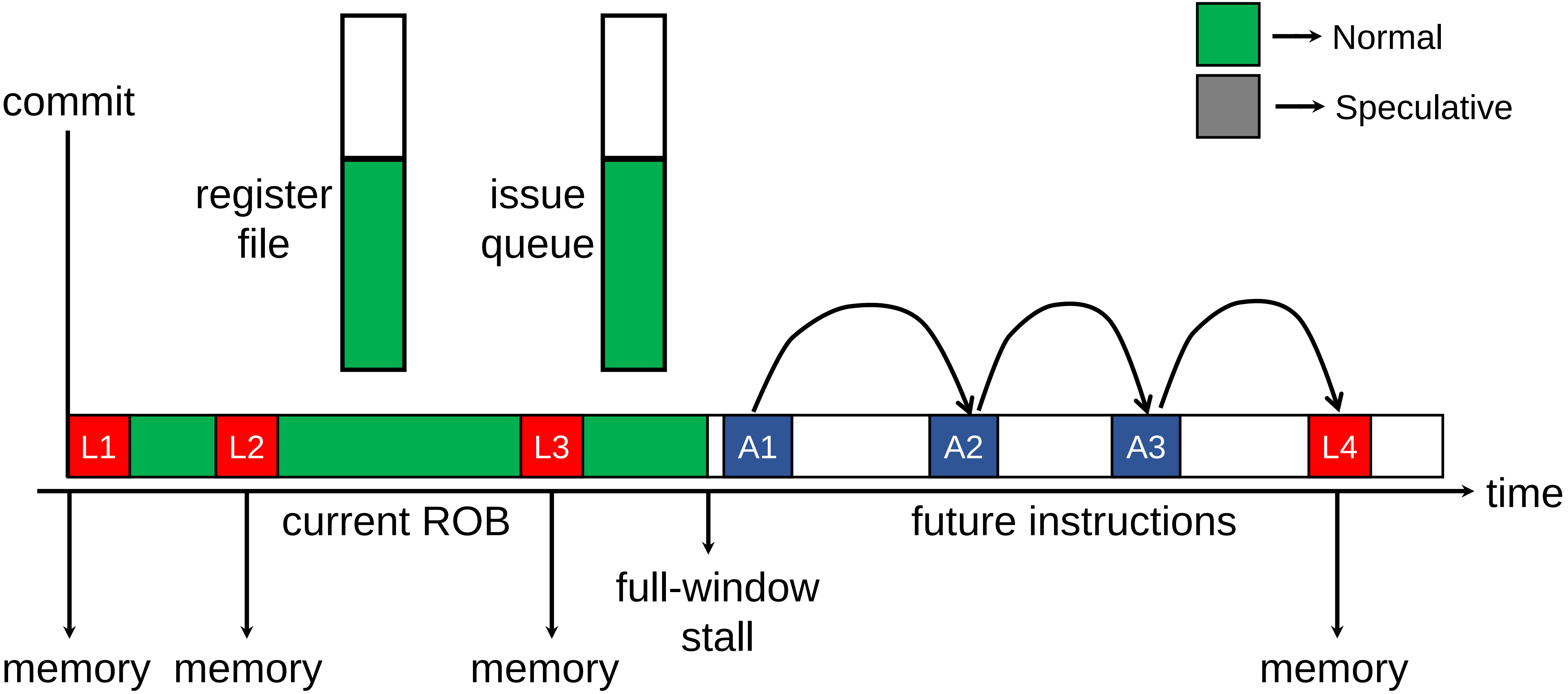
current ROB

future instructions

full-window stall

memory memory memory memory

Processor Resources During Runahead



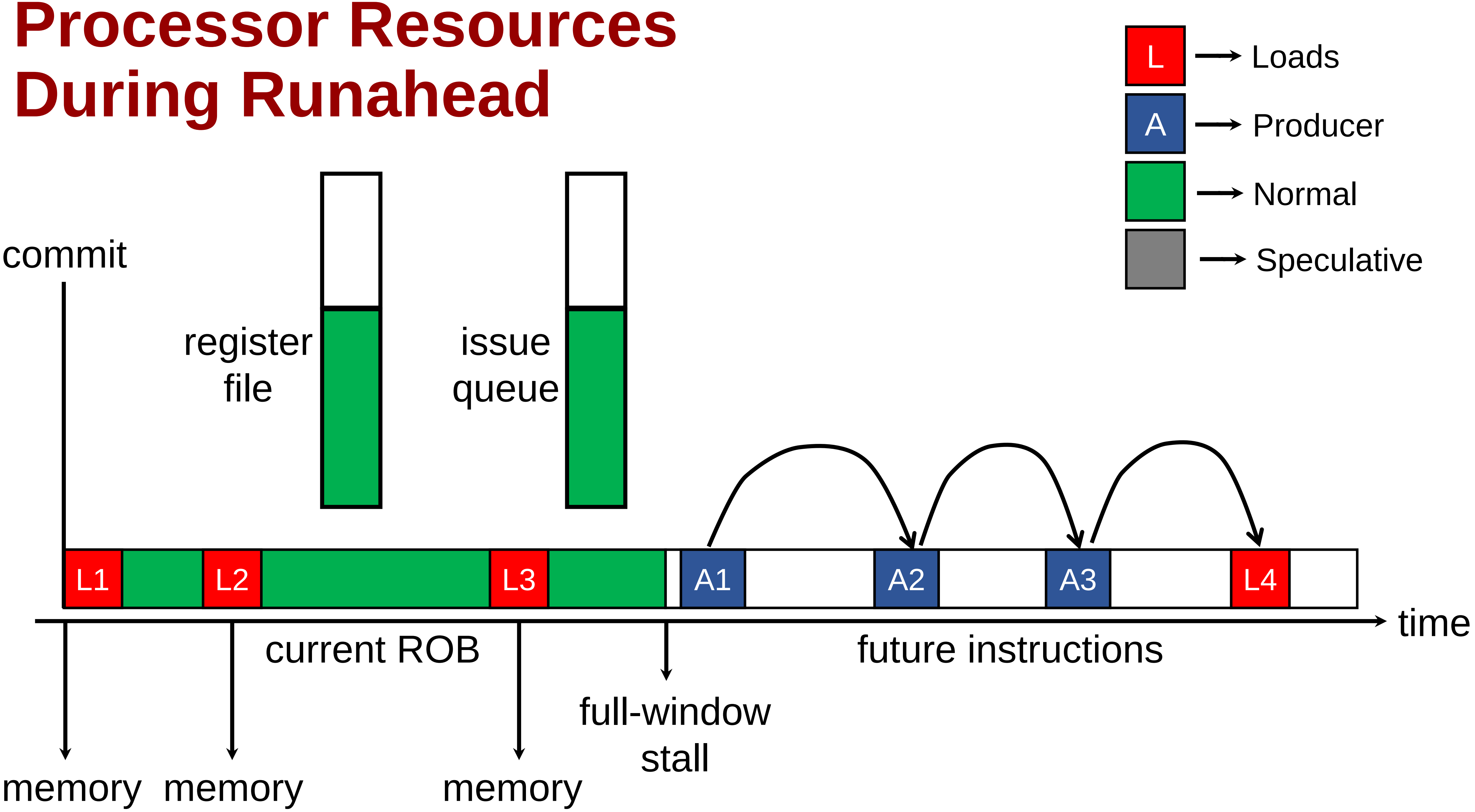
🟥 L → Loads

🟦 A → Producer

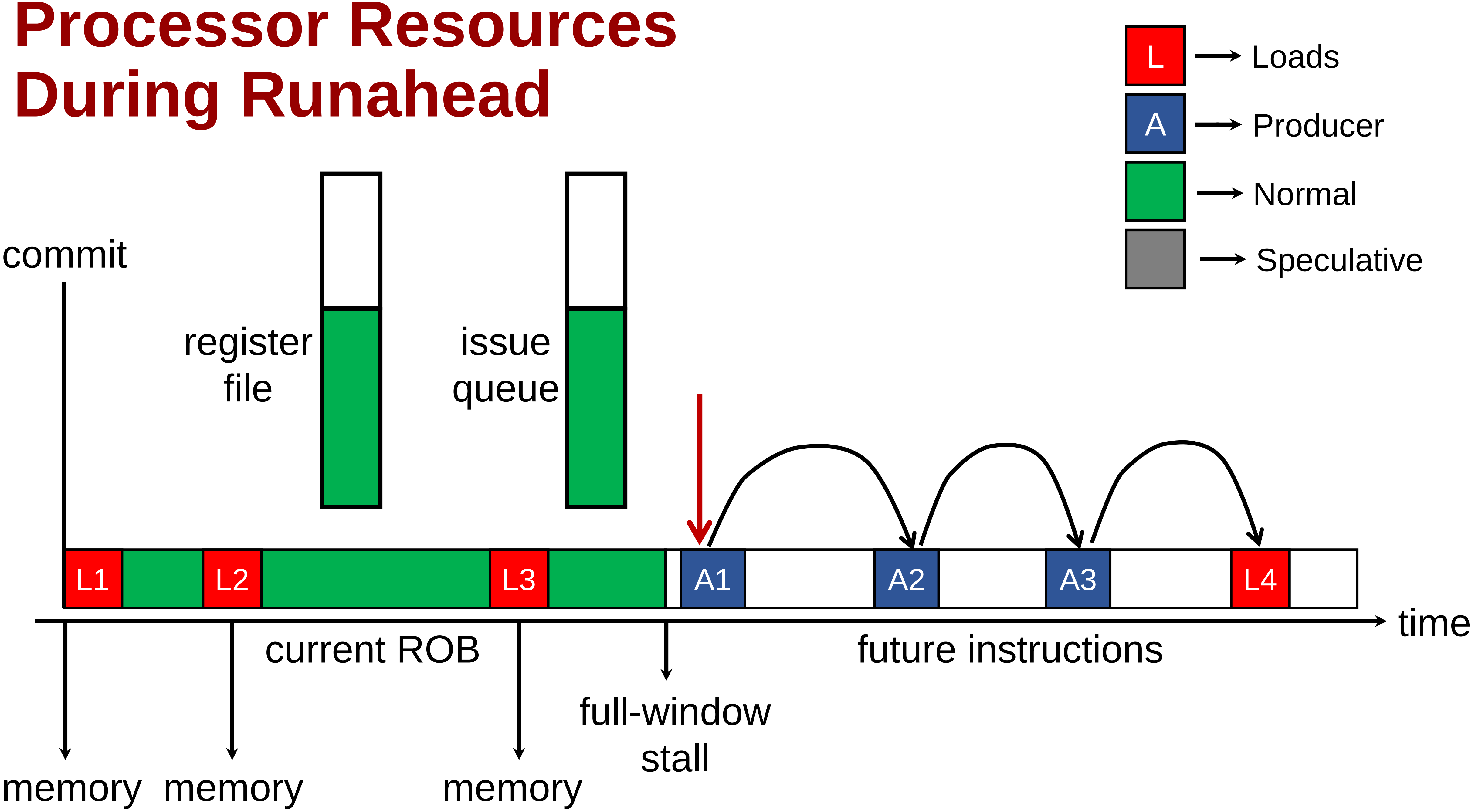
🟩 → Normal

⬛ → Speculative

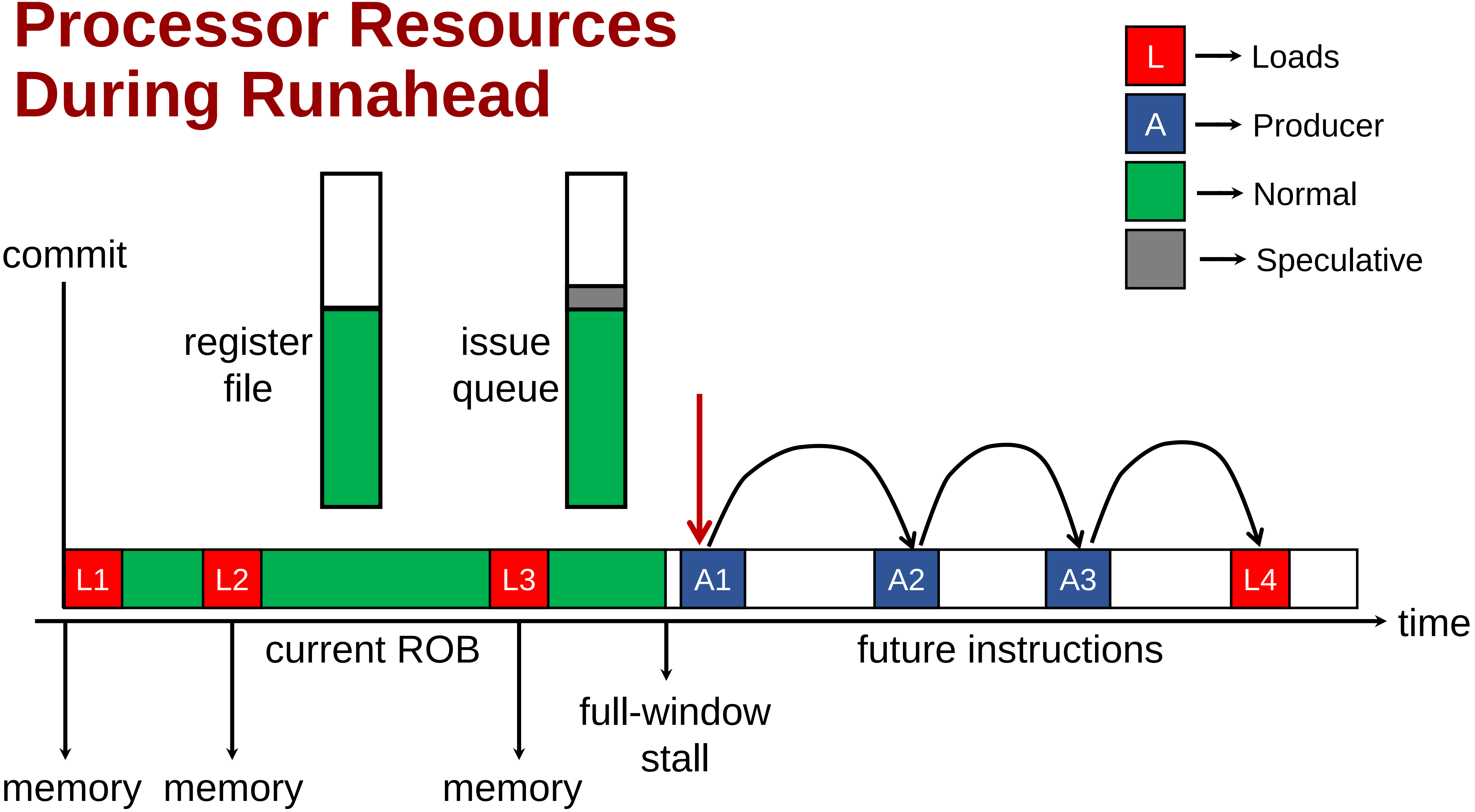
Processor Resources During Runahead



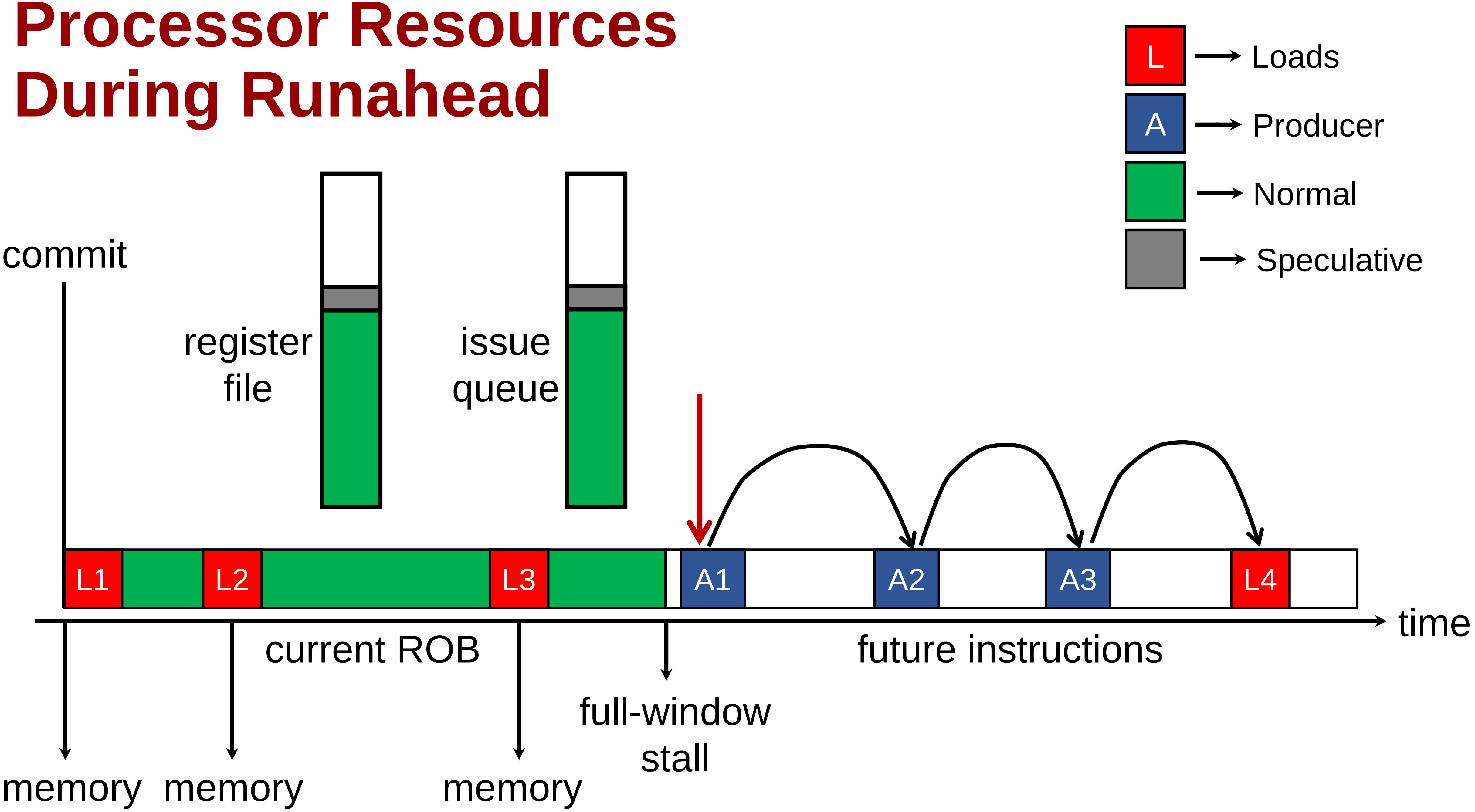
Processor Resources During Runahead



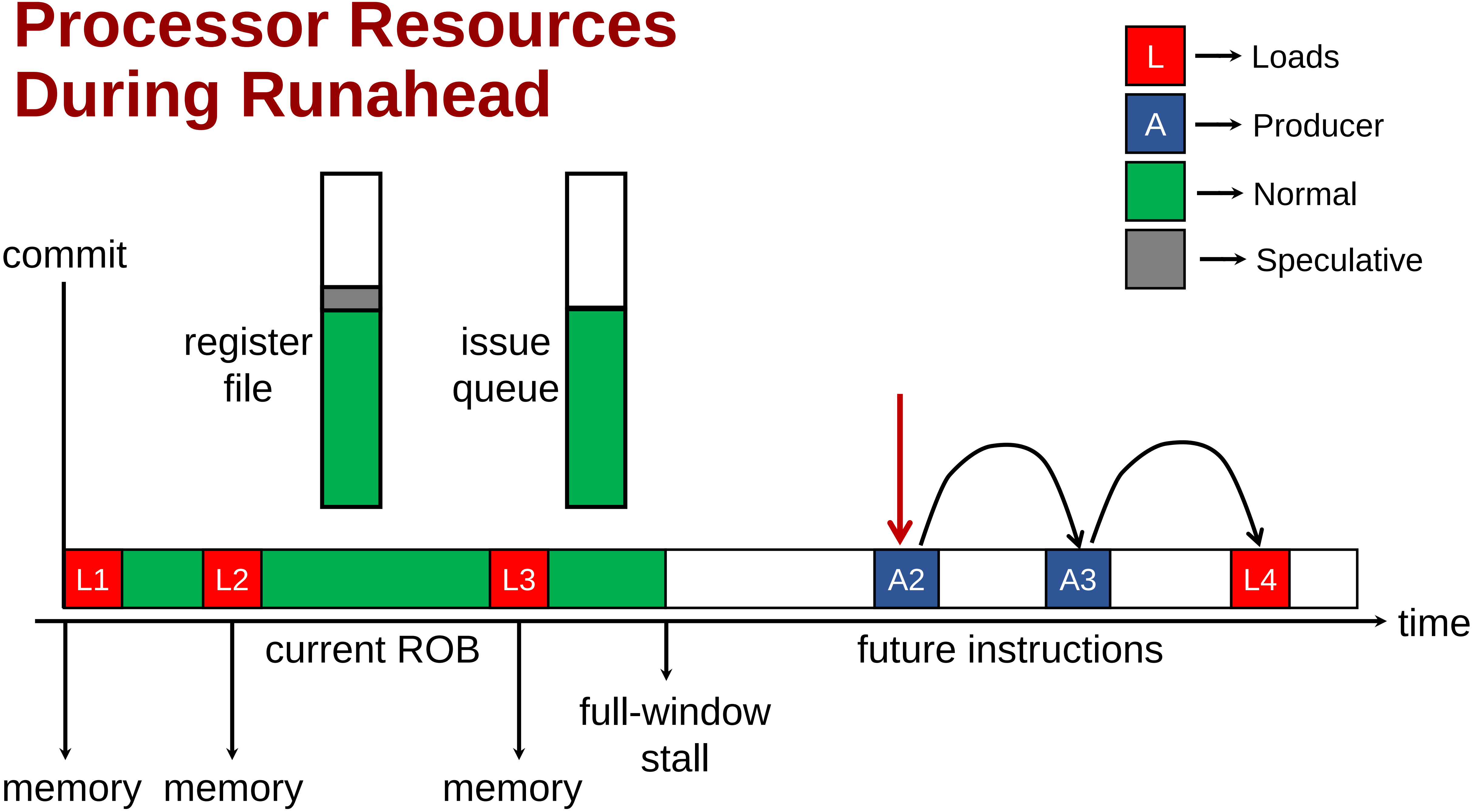
Processor Resources During Runahead



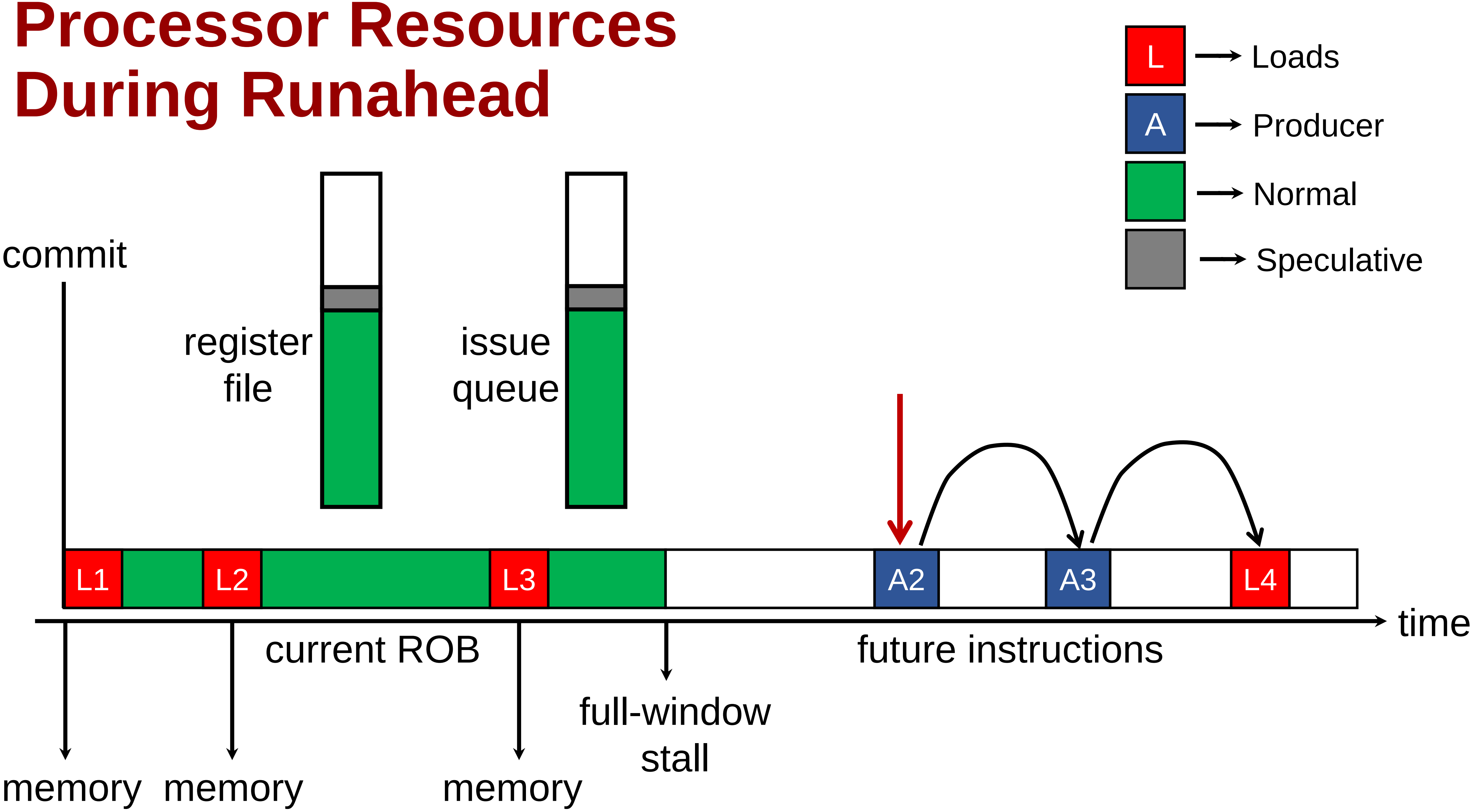
Processor Resources During Runahead



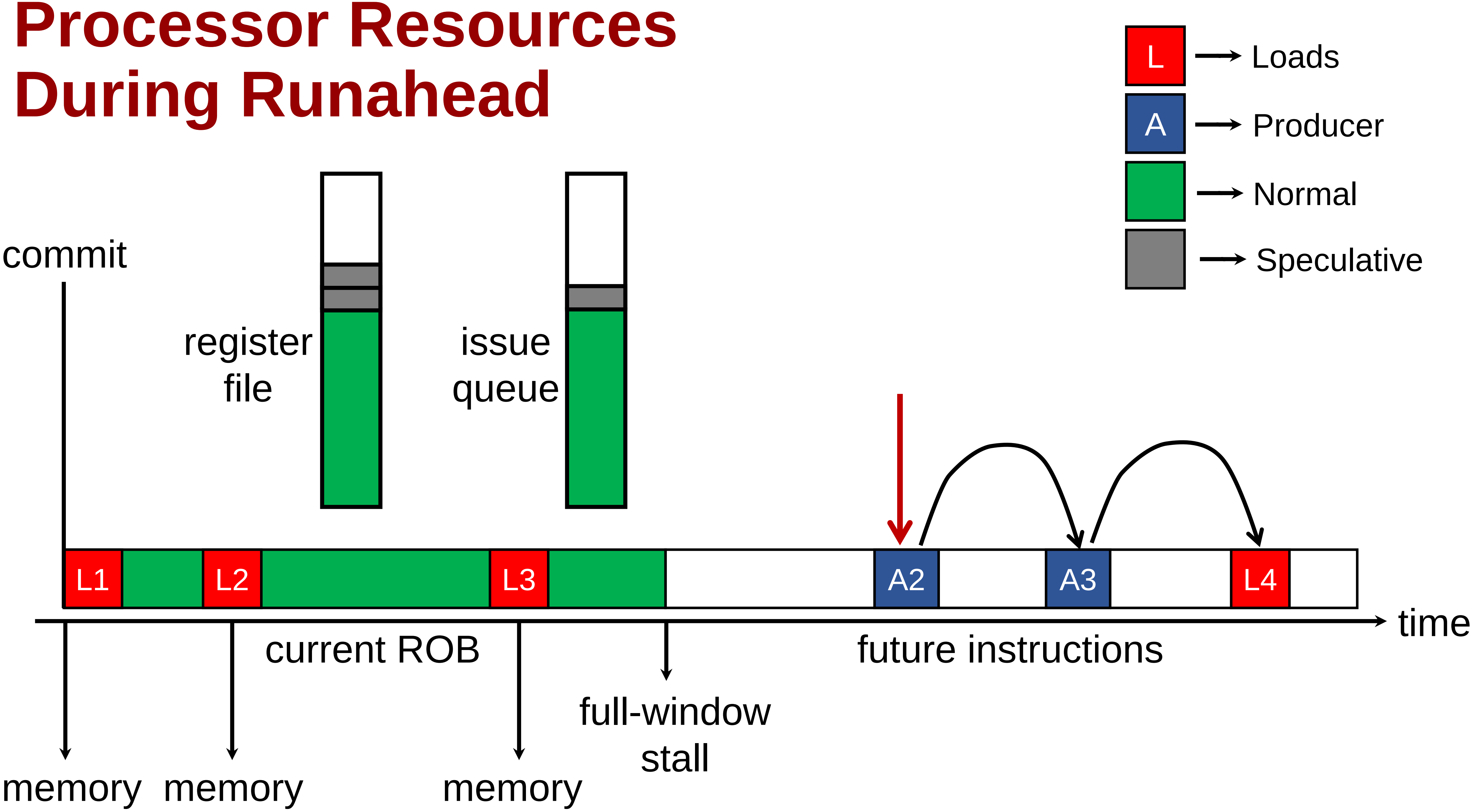
Processor Resources During Runahead



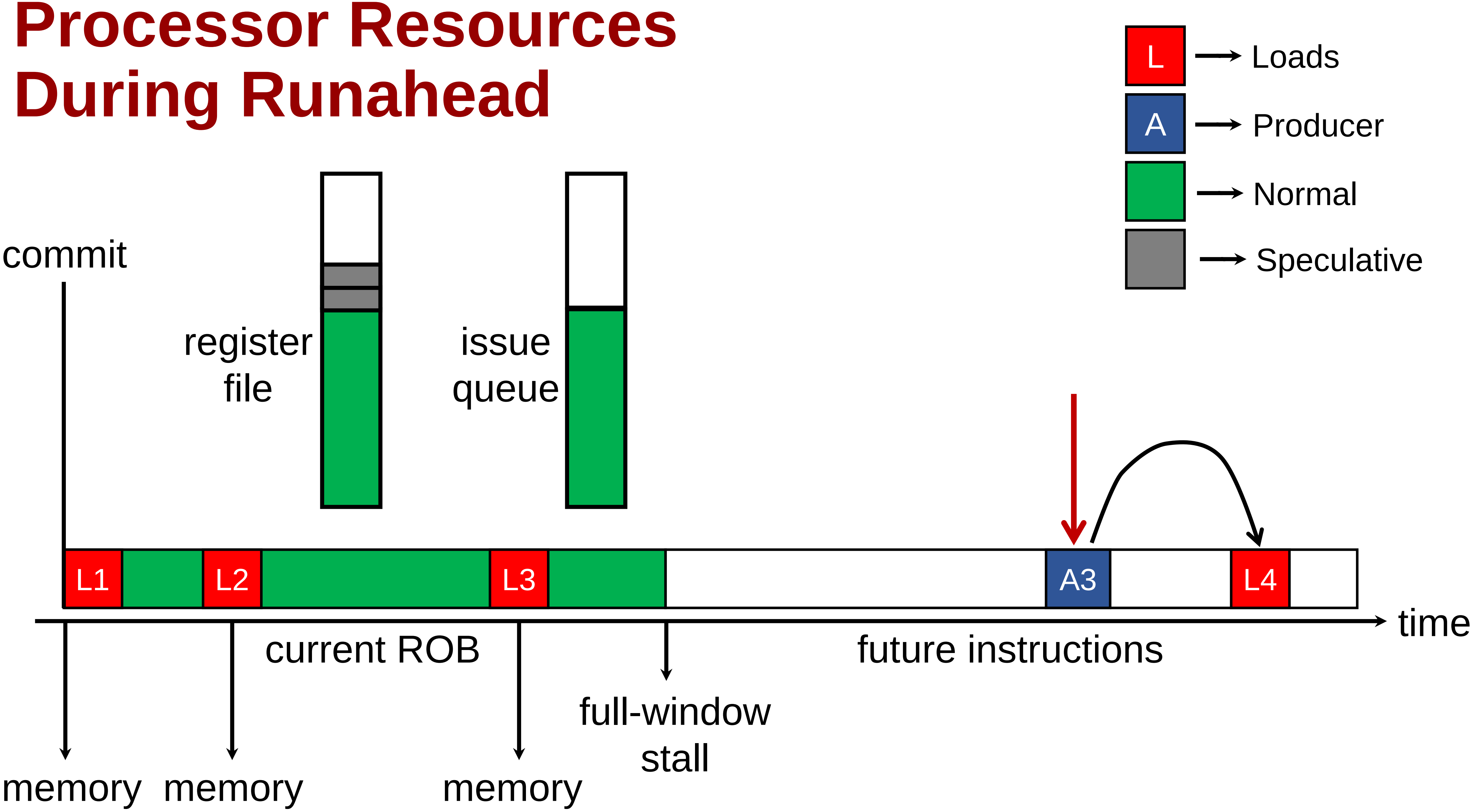
Processor Resources During Runahead



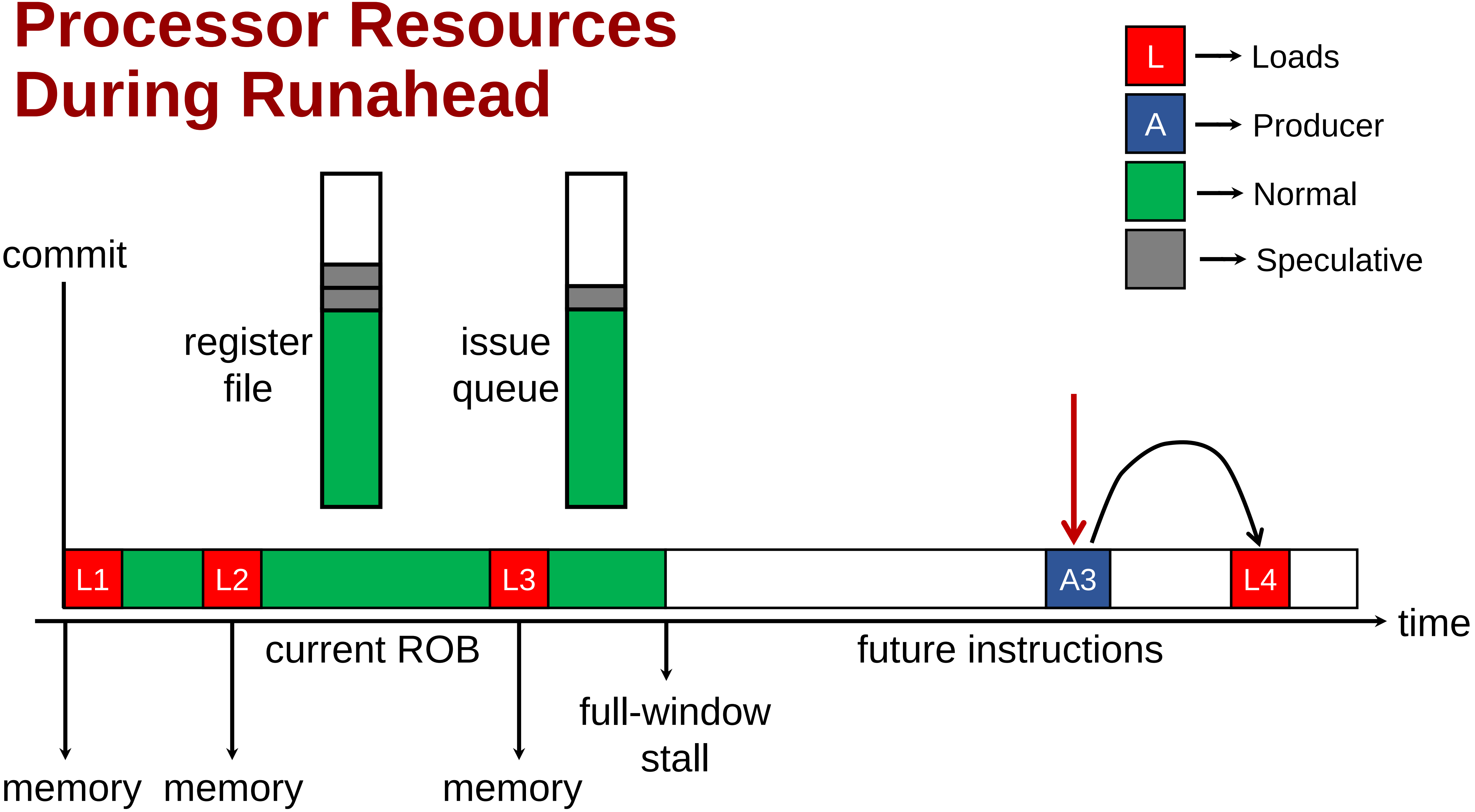
Processor Resources During Runahead



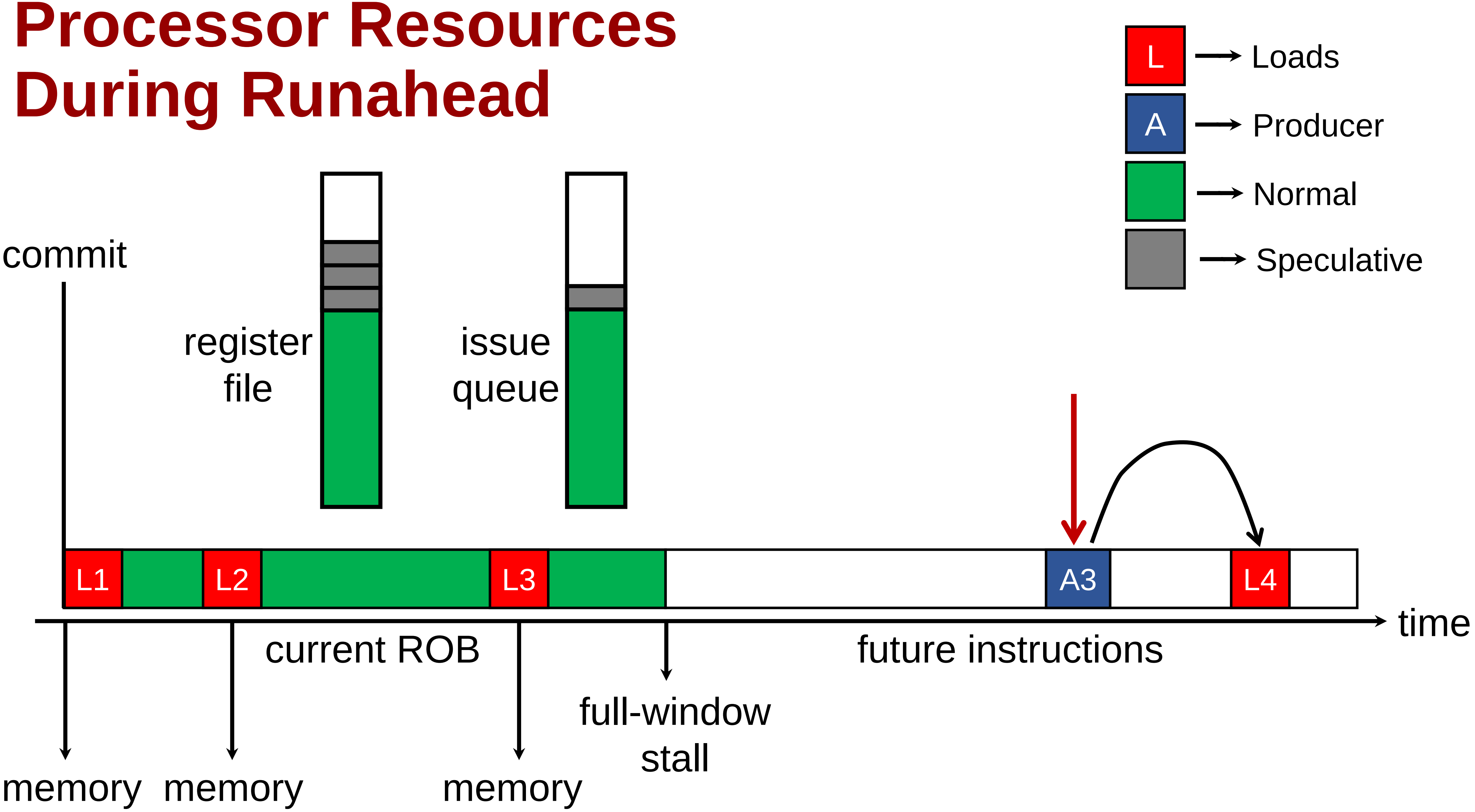
Processor Resources During Runahead



Processor Resources During Runahead



Processor Resources During Runahead



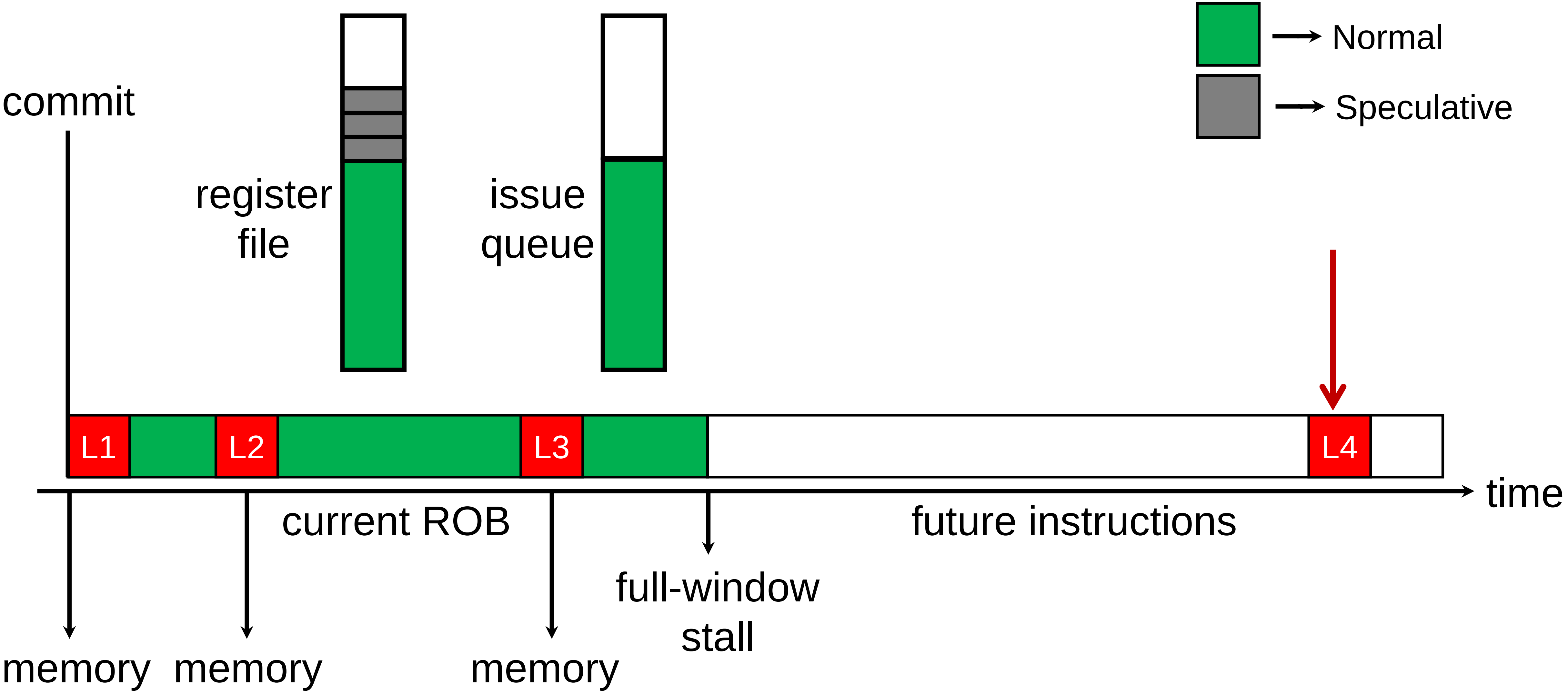
Processor Resources During Runahead

L → Loads

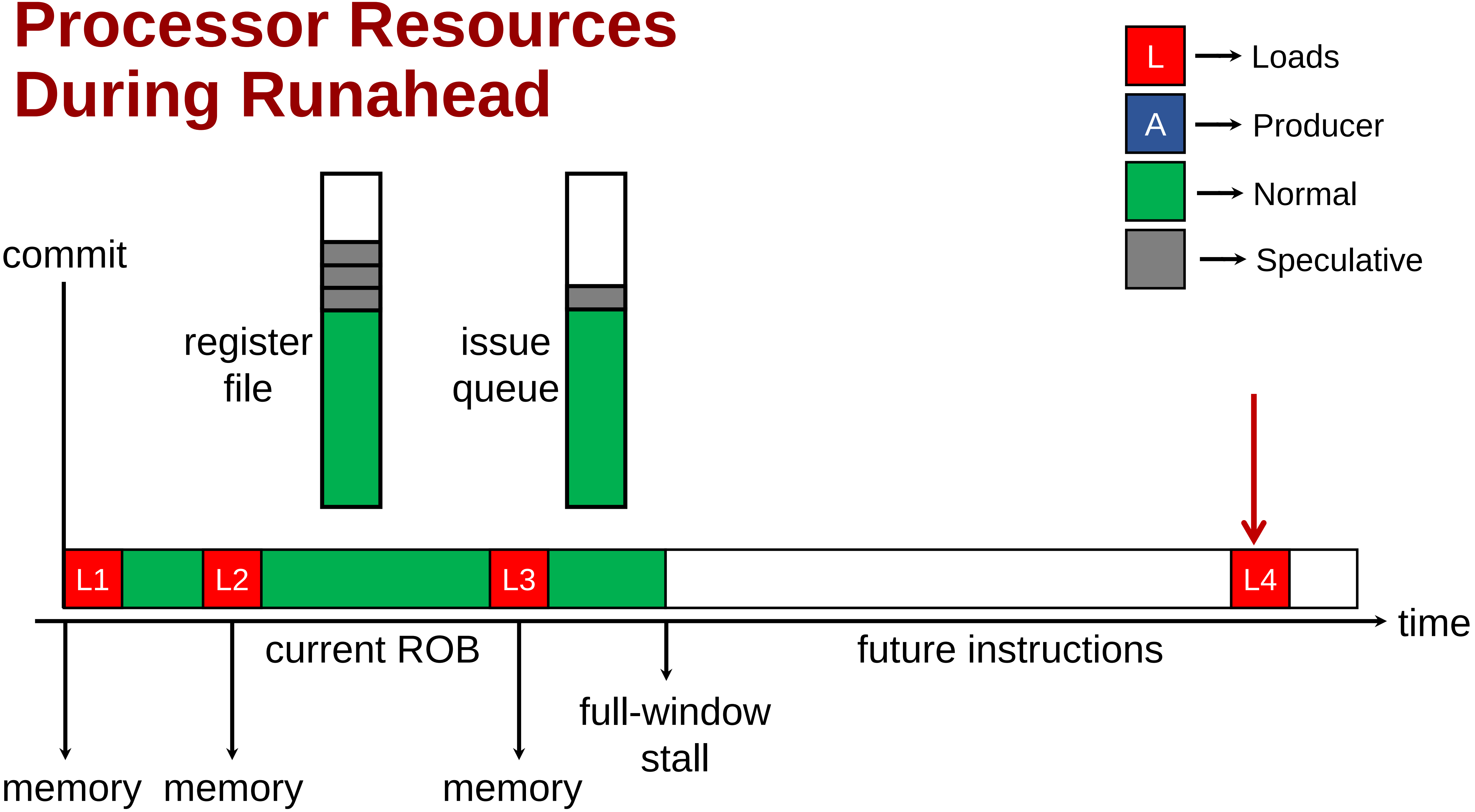
A → Producer

🟩 → Normal

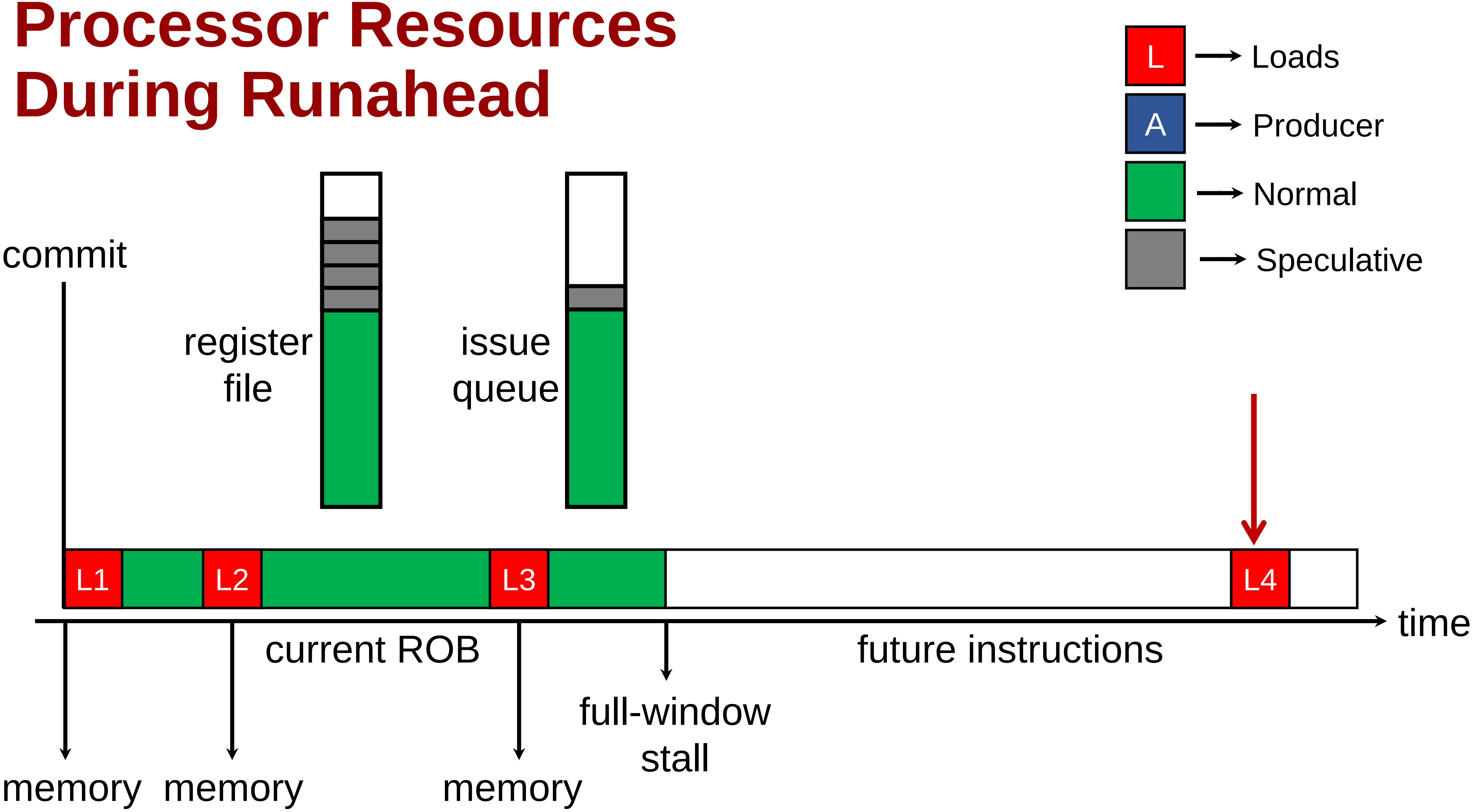
▨ → Speculative



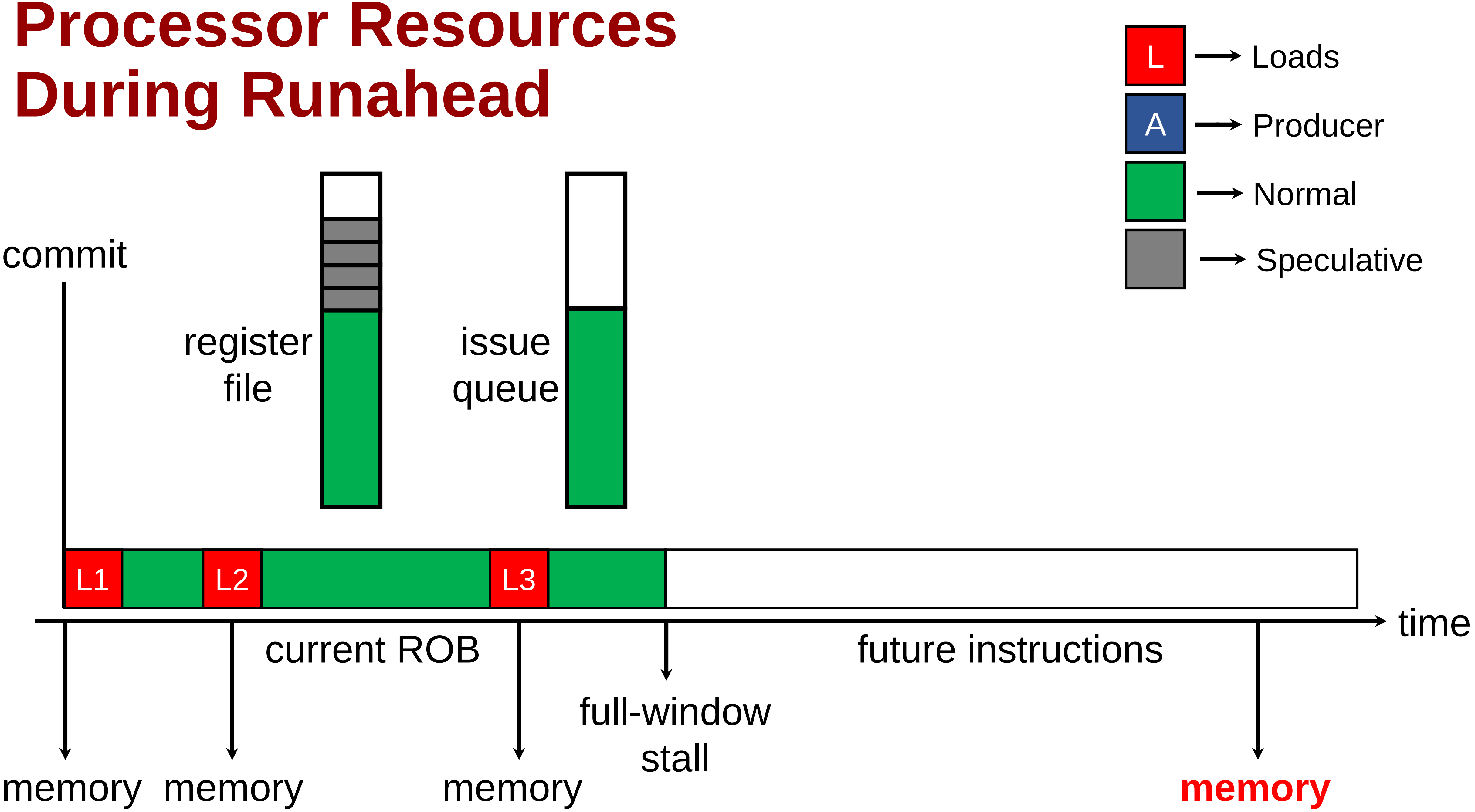
Processor Resources During Runahead



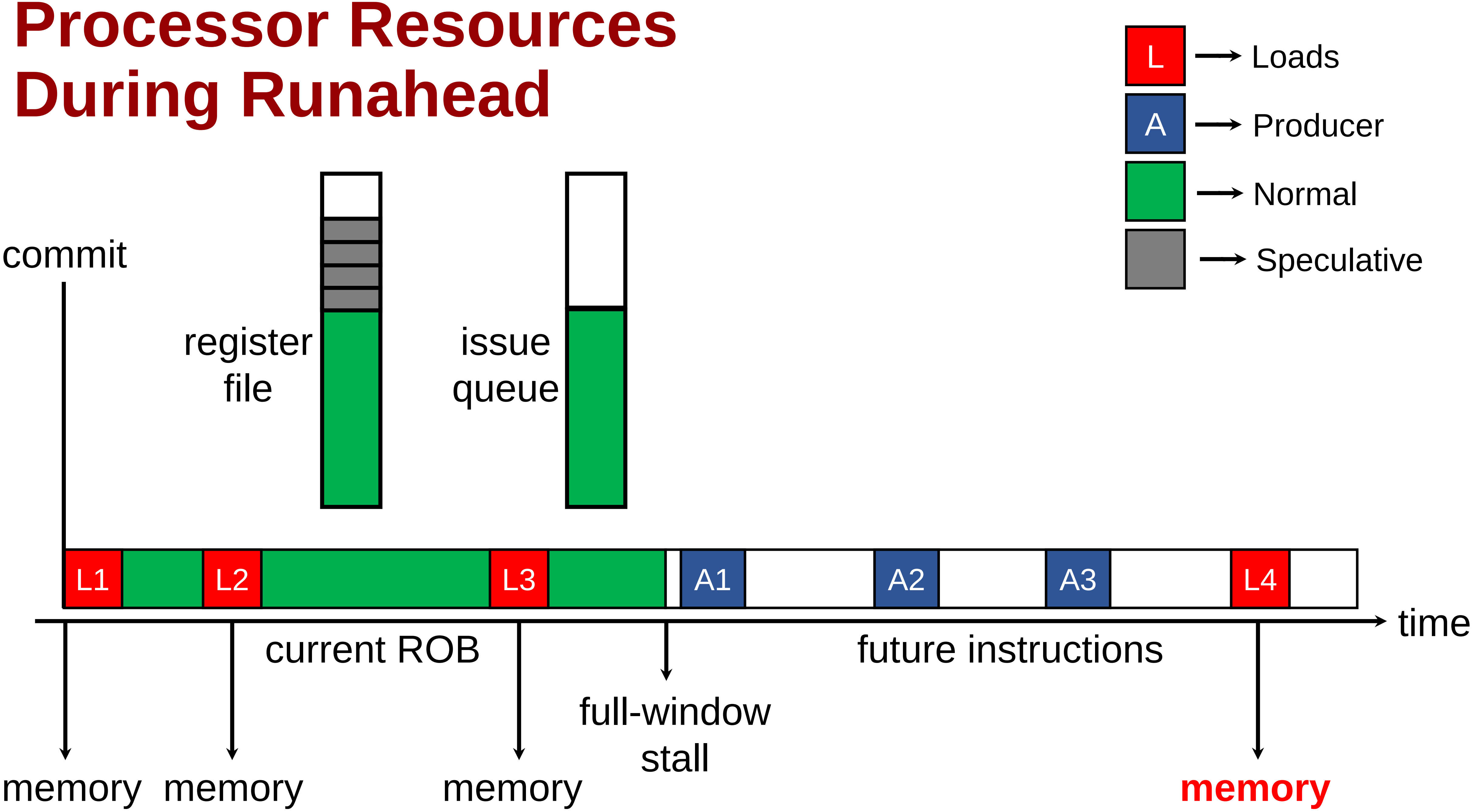
Processor Resources During Runahead



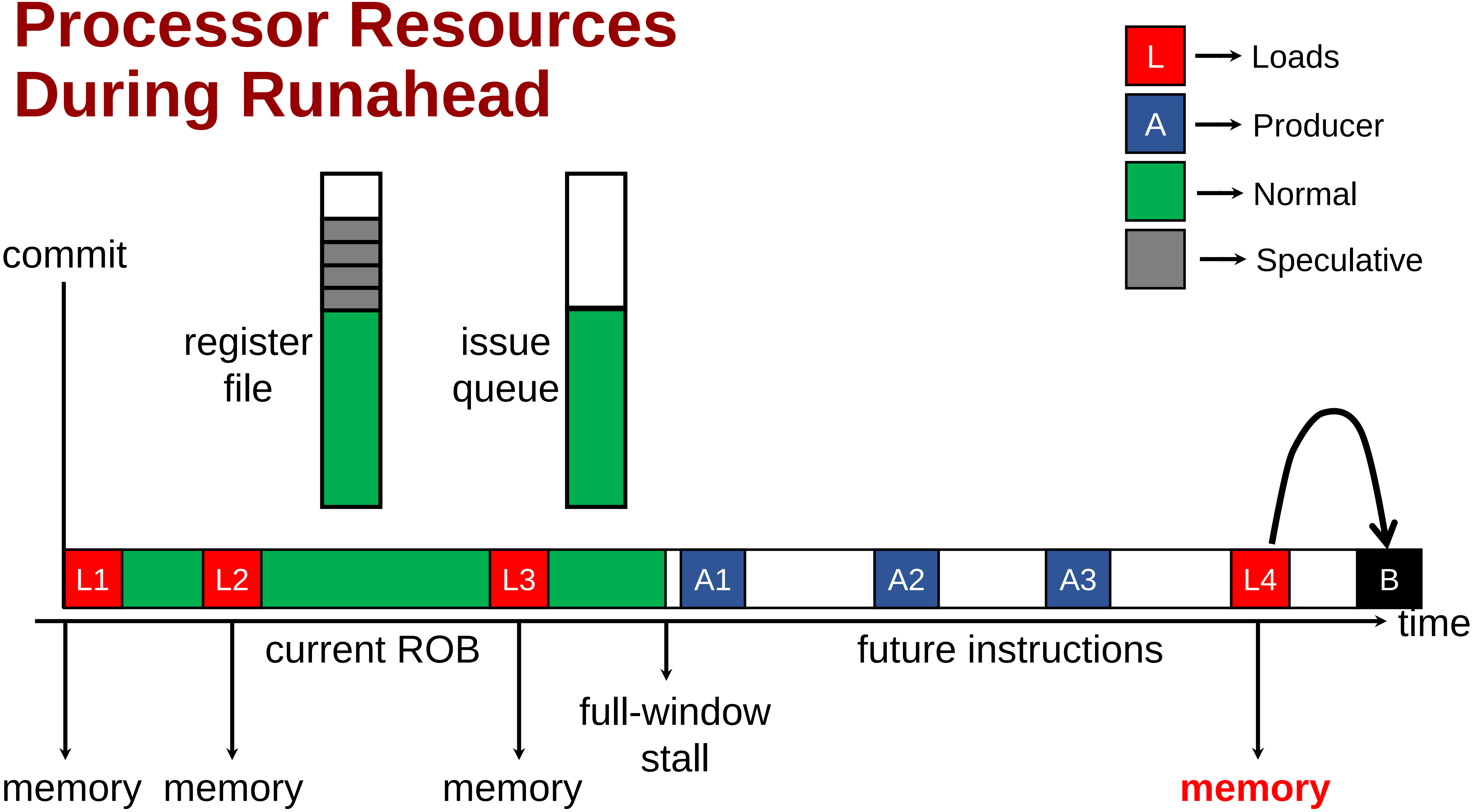
Processor Resources During Runahead



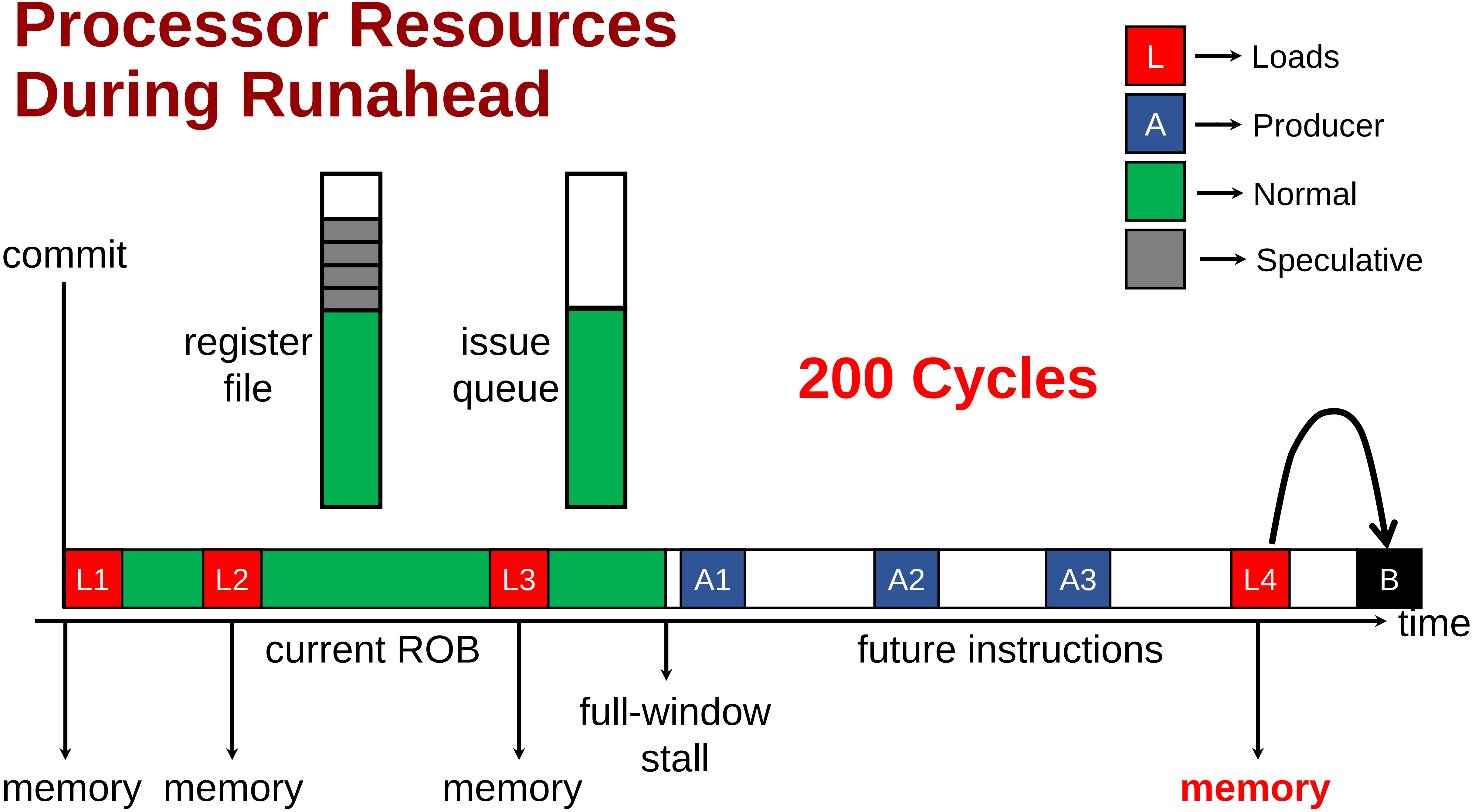
Processor Resources During Runahead



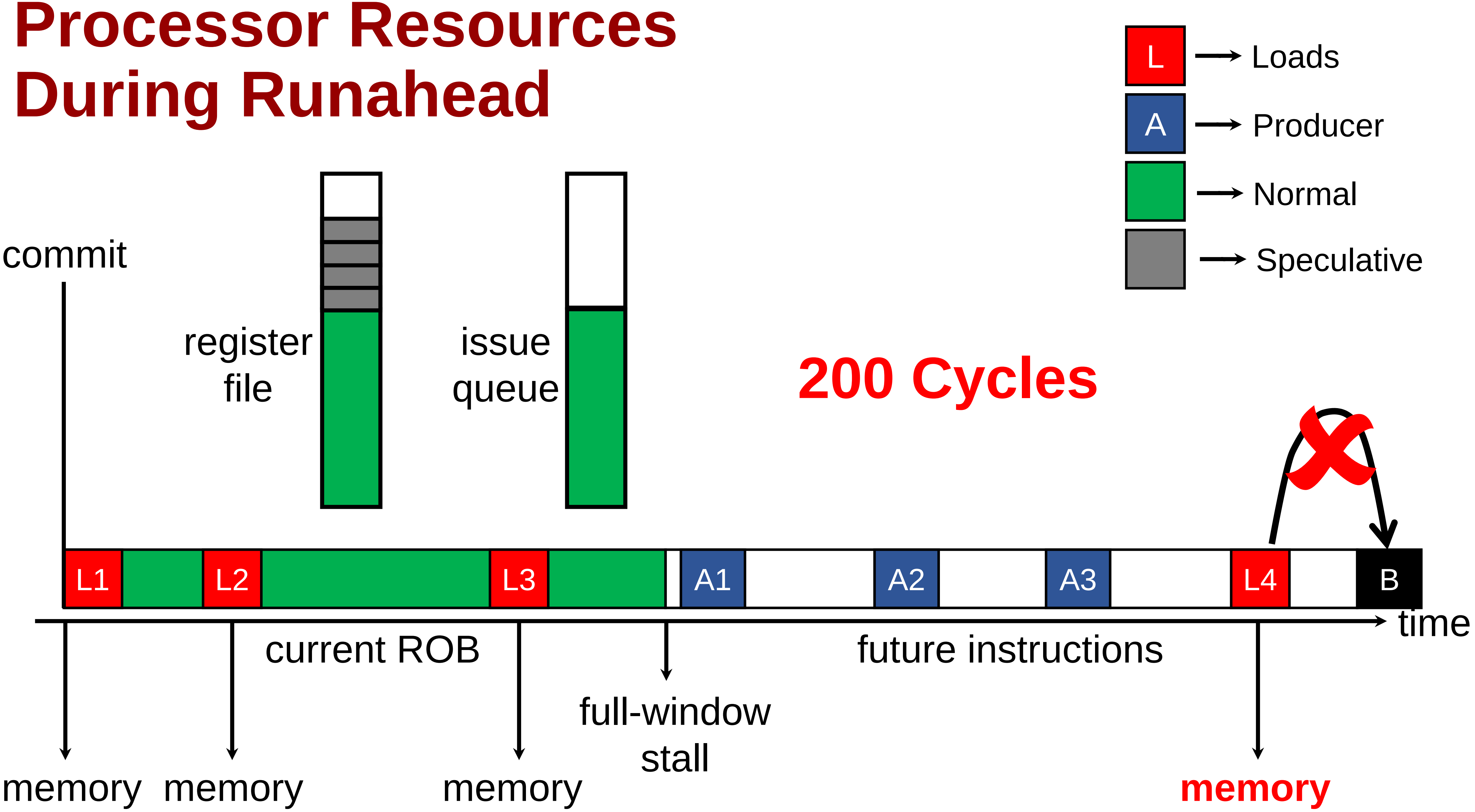
Processor Resources During Runahead



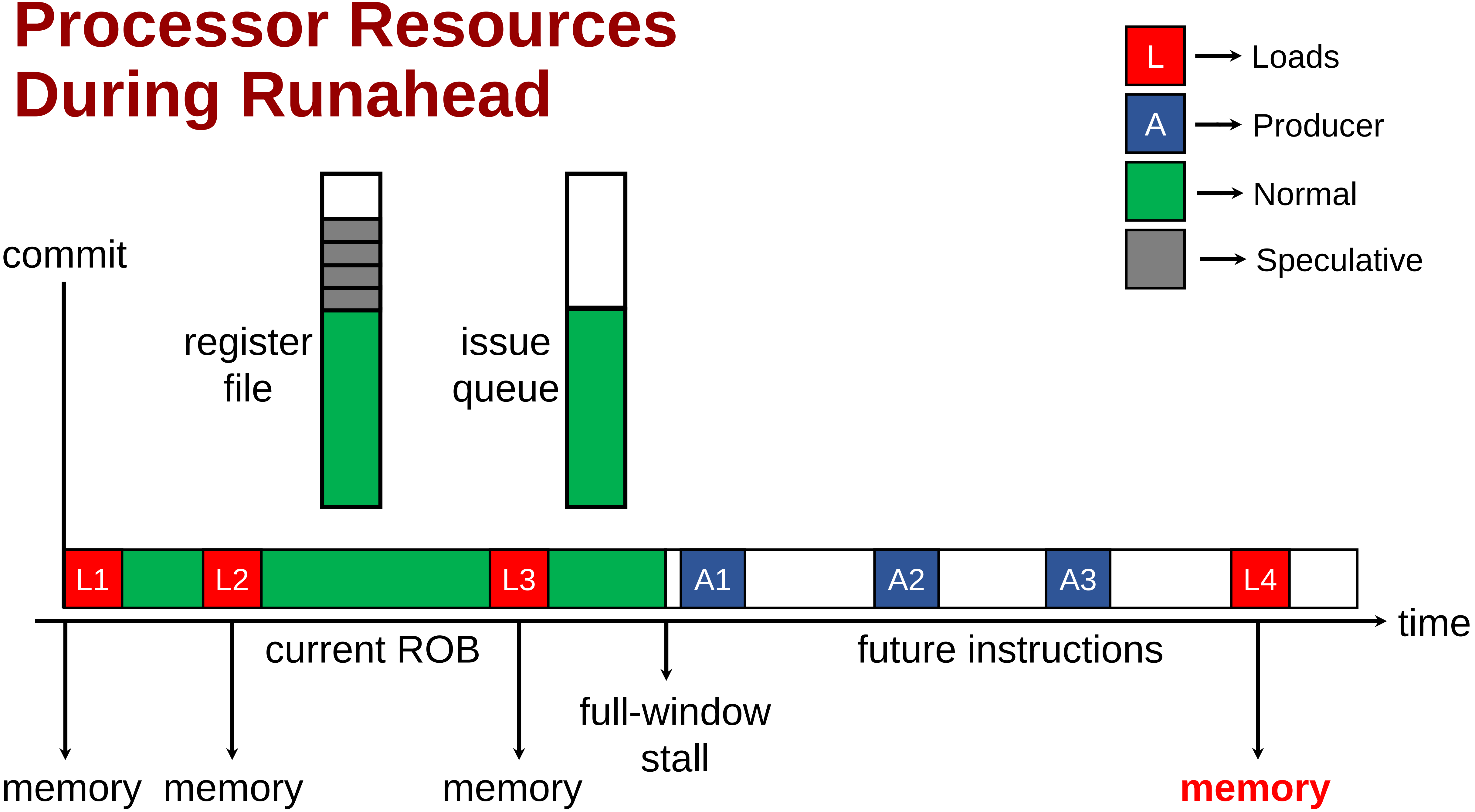
Processor Resources During Runahead



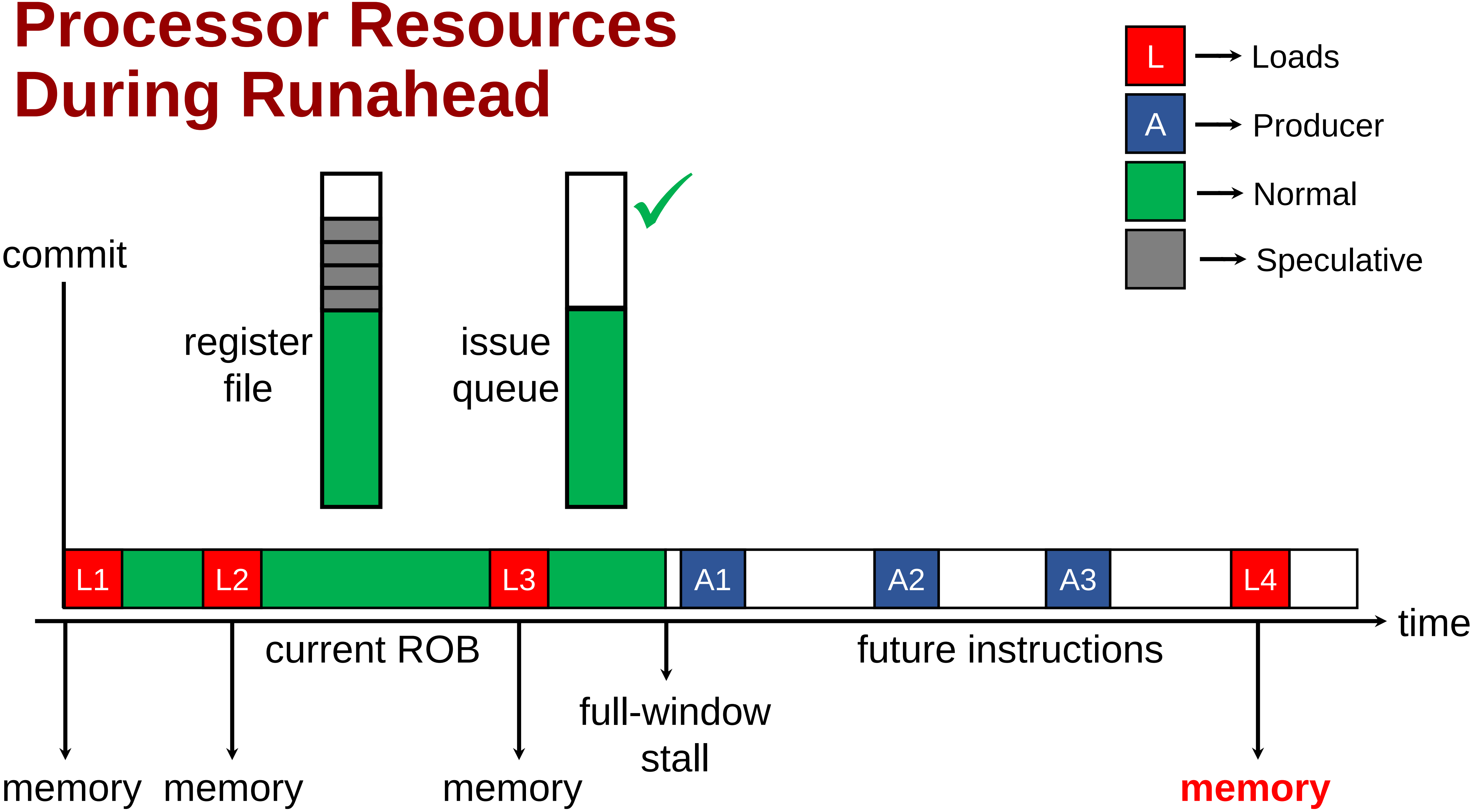
Processor Resources During Runahead



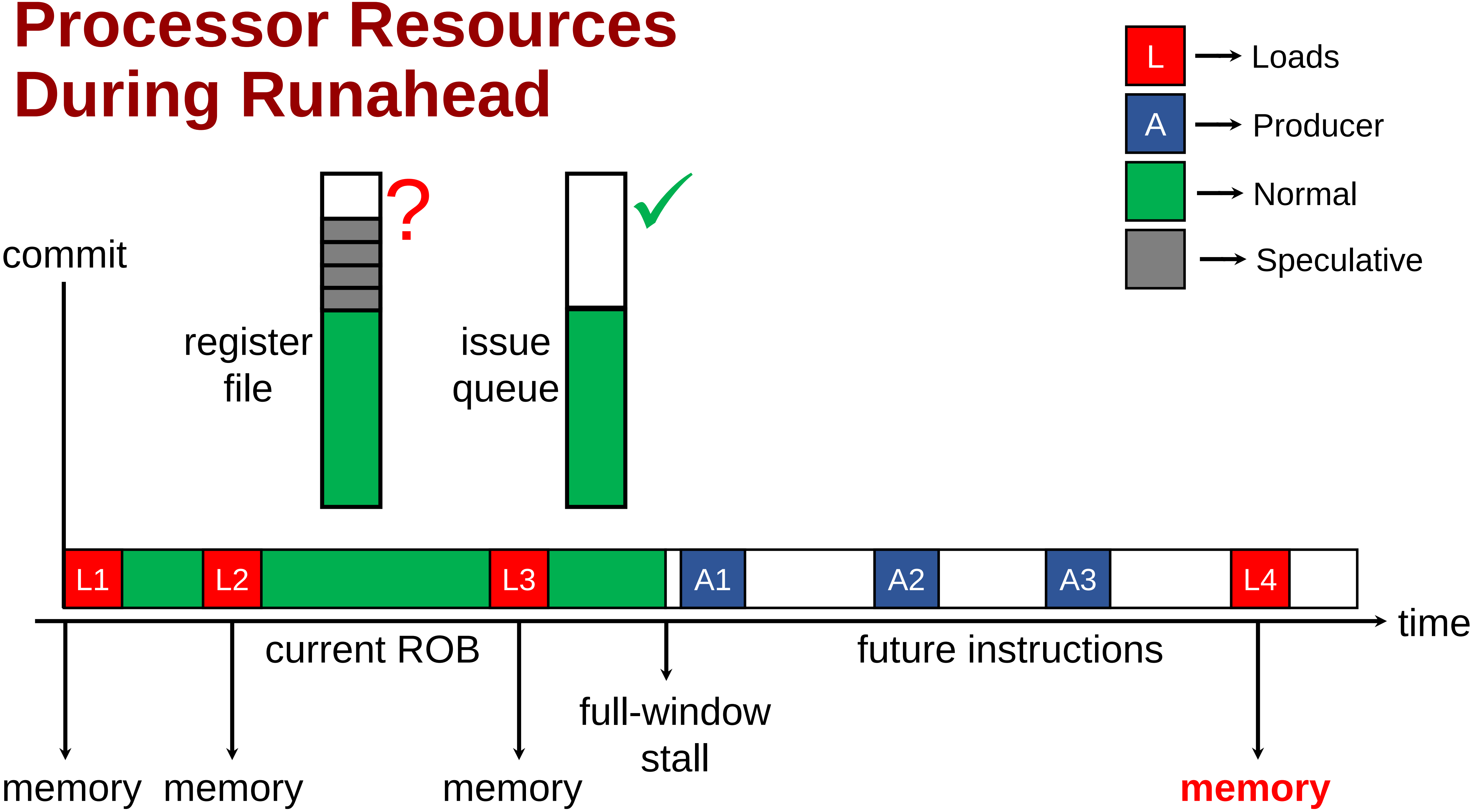
Processor Resources During Runahead



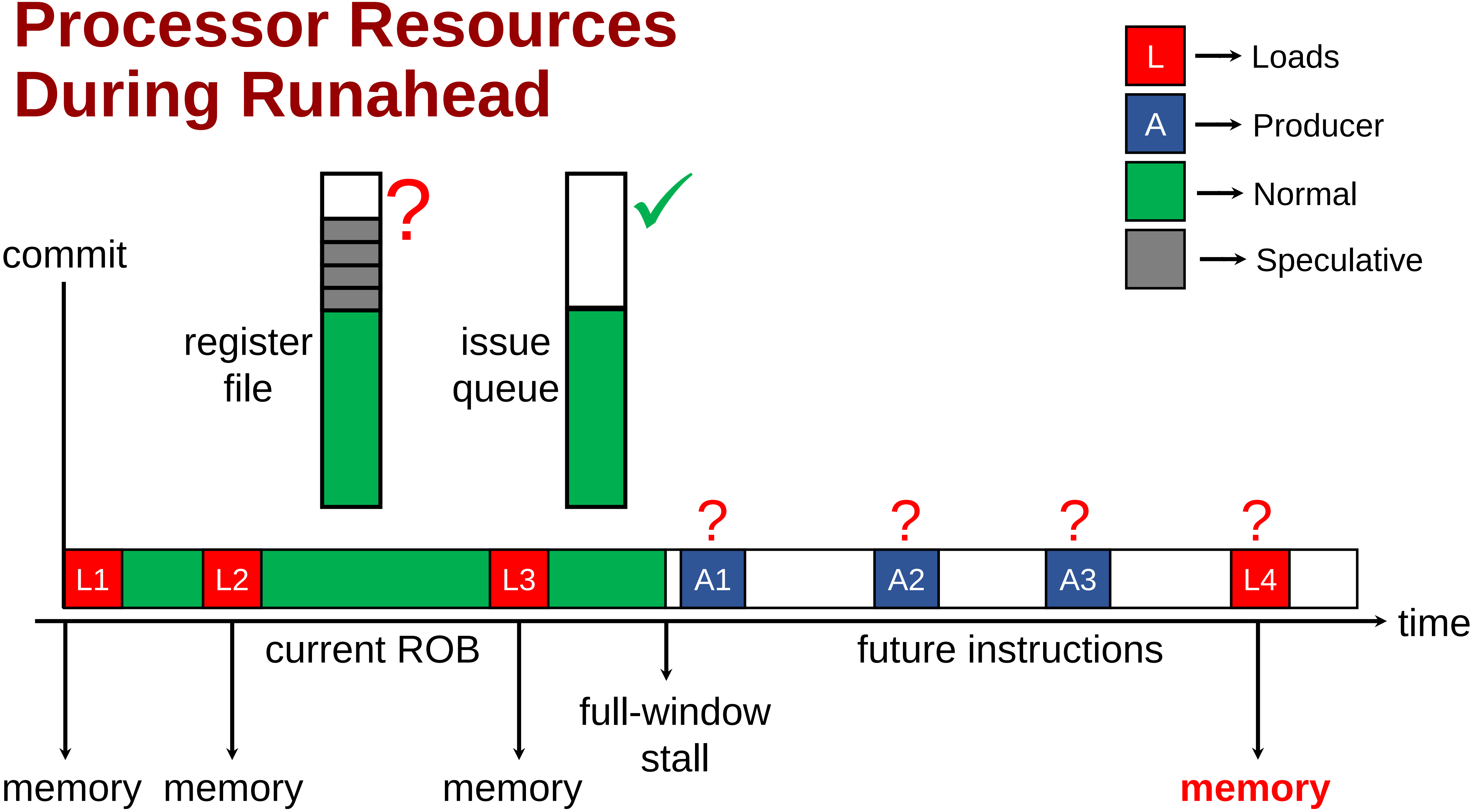
Processor Resources During Runahead



Processor Resources During Runahead



Processor Resources During Runahead



Two Key Questions

Two Key Questions

1. How to identify only useful instructions?

Two Key Questions

1. How to identify only useful instructions?

2. How to recycle (physical) registers?

Two Key Questions

1. How to identify only useful instructions? ⇒ Iterative Backward Dependency Analysis (IBDA)

2. How to recycle (physical) registers?

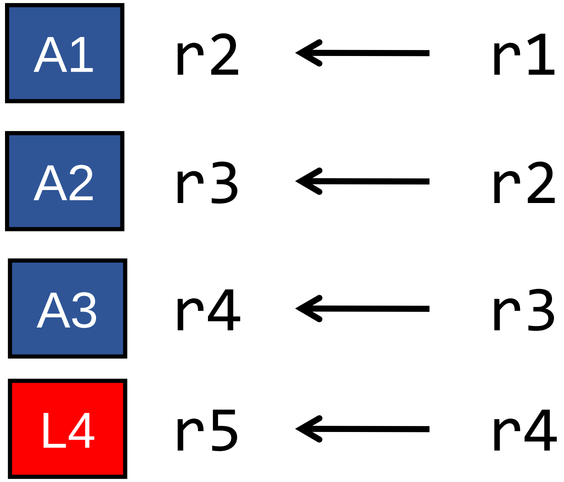
Two Key Questions

1. How to identify only useful instructions? ⇒ Iterative Backward Dependency Analysis (IBDA)

2. How to recycle (physical) registers? ⇒ Runahead Register Reclamation

Iteratively Identifying the Stalling Slices

Iteratively Identifying the Stalling Slices



A1 r2 ← r1

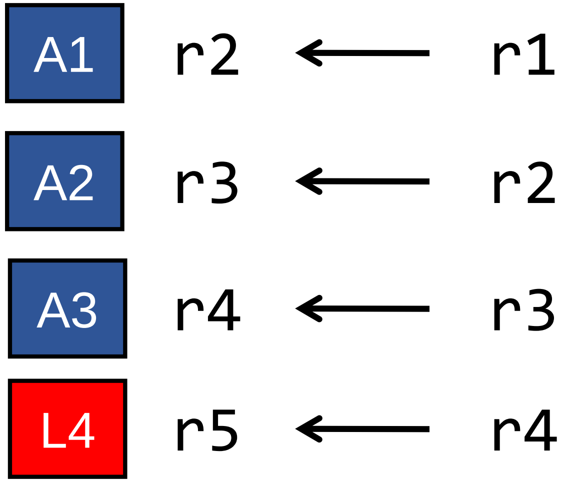
A2 r3 ← r2

A3 r4 ← r3

L4 r5 ← r4

Iteratively Identifying the Stalling Slices

Register Allocation Table (RAT)



A1 r2 ← r1

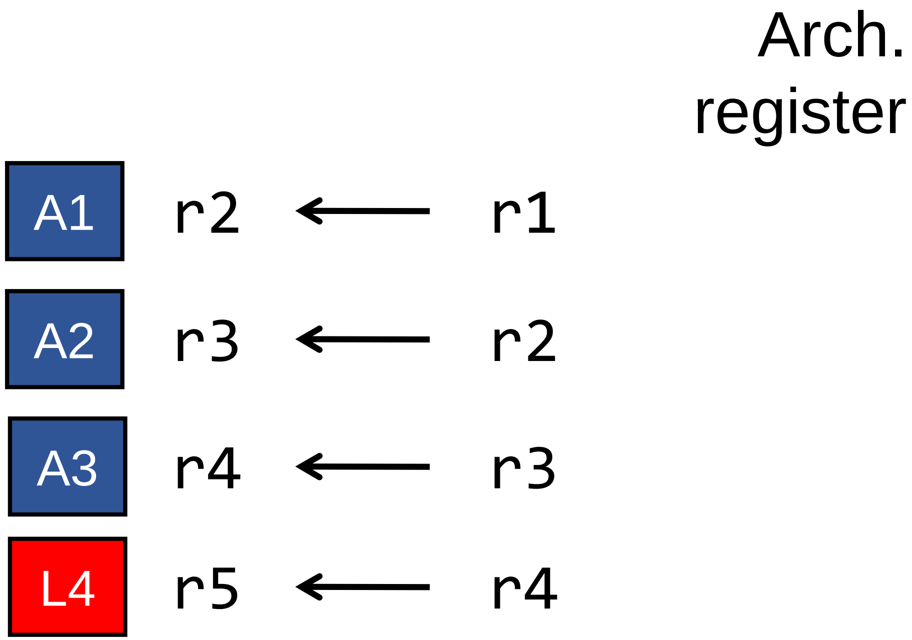
A2 r3 ← r2

A3 r4 ← r3

L4 r5 ← r4

Iteratively Identifying the Stalling Slices

Register Allocation Table (RAT)



Arch. register

A1 r2 ← r1

A2 r3 ← r2

A3 r4 ← r3

L4 r5 ← r4

Iteratively Identifying the Stalling Slices

Register Allocation Table (RAT)

Arch. register	Phy. register

A1 r2 ← r1

A2 r3 ← r2

A3 r4 ← r3

L4 r5 ← r4

Iteratively Identifying the Stalling Slices

A1 r2 ← r1

A2 r3 ← r2

A3 r4 ← r3

L4 r5 ← r4

Register Allocation Table (RAT)

Arch. register	Phy. register
r1	P1
r2	P2
r3	P3
r4	P4
r5	P5

Iteratively Identifying the Stalling Slices

A1 r2 ← r1

A2 r3 ← r2

A3 r4 ← r3

L4 r5 ← r4

Register Allocation Table (RAT)

Arch. register	Phy. register	Last-writer instruction
r1	P1	
r2	P2	
r3	P3	
r4	P4	
r5	P5	

Iteratively Identifying the Stalling Slices

A1 r2 ← r1

A2 r3 ← r2

A3 r4 ← r3

L4 r5 ← r4

Register Allocation Table (RAT)

Arch. register	Phy. register	Last-writer instruction
r1	P1	A0
r2	P2	
r3	P3	
r4	P4	
r5	P5	

Iteratively Identifying the Stalling Slices

A1 r2 ← r1

A2 r3 ← r2

A3 r4 ← r3

L4 r5 ← r4

Register Allocation Table (RAT)

Arch. register	Phy. register	Last-writer instruction
r1	P1	A0
r2	P2	A1
r3	P3	
r4	P4	
r5	P5	

Iteratively Identifying the Stalling Slices

A1 r2 ← r1

A2 r3 ← r2

A3 r4 ← r3

L4 r5 ← r4

Register Allocation Table (RAT)

Arch. register	Phy. register	Last-writer instruction
r1	P1	A0
r2	P2	A1
r3	P3	A2
r4	P4	
r5	P5	

Iteratively Identifying the Stalling Slices

A1 r2 ← r1

A2 r3 ← r2

A3 r4 ← r3

L4 r5 ← r4

Register Allocation Table (RAT)

Arch. register	Phy. register	Last-writer instruction
r1	P1	A0
r2	P2	A1
r3	P3	A2
r4	P4	A3
r5	P5	

Iteratively Identifying the Stalling Slices

A1 r2 ← r1

A2 r3 ← r2

A3 r4 ← r3

L4 r5 ← r4

Register Allocation Table (RAT)

Arch. register	Phy. register	Last-writer instruction
r1	P1	A0
r2	P2	A1
r3	P3	A2
r4	P4	A3
r5	P5	L4

Iteratively Identifying the Stalling Slices

A1 r2 ← r1

A2 r3 ← r2

A3 r4 ← r3

L4 r5 ← r4

Register Allocation Table (RAT)

Arch. register	Phy. register	Last-writer instruction
r1	P1	A0
r2	P2	A1
r3	P3	A2
r4	P4	A3
r5	P5	L4

Stalling Slice Table (SST)

Iteratively Identifying the Stalling Slices

A1 r2 ← r1

A2 r3 ← r2

A3 r4 ← r3

L4 r5 ← r4

Register Allocation Table (RAT)

Arch. register	Phy. register	Last-writer instruction
r1	P1	A0
r2	P2	A1
r3	P3	A2
r4	P4	A3
r5	P5	L4

Stalling Slice Table (SST)

Iteration-1:
L4 stalls the window

Iteratively Identifying the Stalling Slices

A1 r2 ← r1

A2 r3 ← r2

A3 r4 ← r3

L4 r5 ← r4

Register Allocation Table (RAT)

Arch. register	Phy. register	Last-writer instruction
r1	P1	A0
r2	P2	A1
r3	P3	A2
r4	P4	A3
r5	P5	L4

Stalling Slice Table (SST)

L4

Iteration-1:
L4 stalls the window

Iteratively Identifying the Stalling Slices

A1 r2 ← r1

A2 r3 ← r2

A3 r4 ← r3

L4 r5 ← r4

Register Allocation Table (RAT)

Arch. register	Phy. register	Last-writer instruction
r1	P1	A0
r2	P2	A1
r3	P3	A2
r4	P4	A3
r5	P5	L4

Stalling Slice Table (SST)

L4

Iteratively Identifying the Stalling Slices

A1 r2 ← r1

A2 r3 ← r2

A3 r4 ← r3

L4 r5 ← r4

Register Allocation Table (RAT)

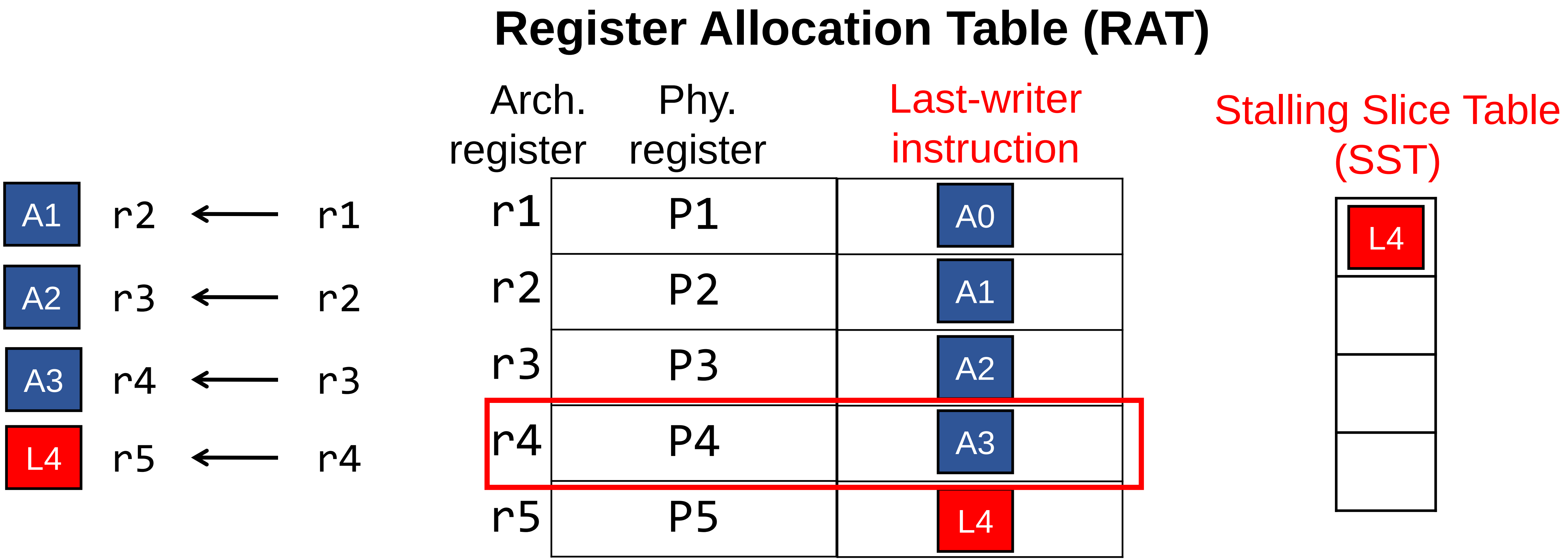
Arch. register	Phy. register	Last-writer instruction
r1	P1	A0
r2	P2	A1
r3	P3	A2
r4	P4	A3
r5	P5	L4

Stalling Slice Table (SST)

L4

Iteration-2:
L4 hits in the SST

Iteratively Identifying the Stalling Slices



A1 r2 ← r1

A2 r3 ← r2

A3 r4 ← r3

L4 r5 ← r4

Register Allocation Table (RAT)

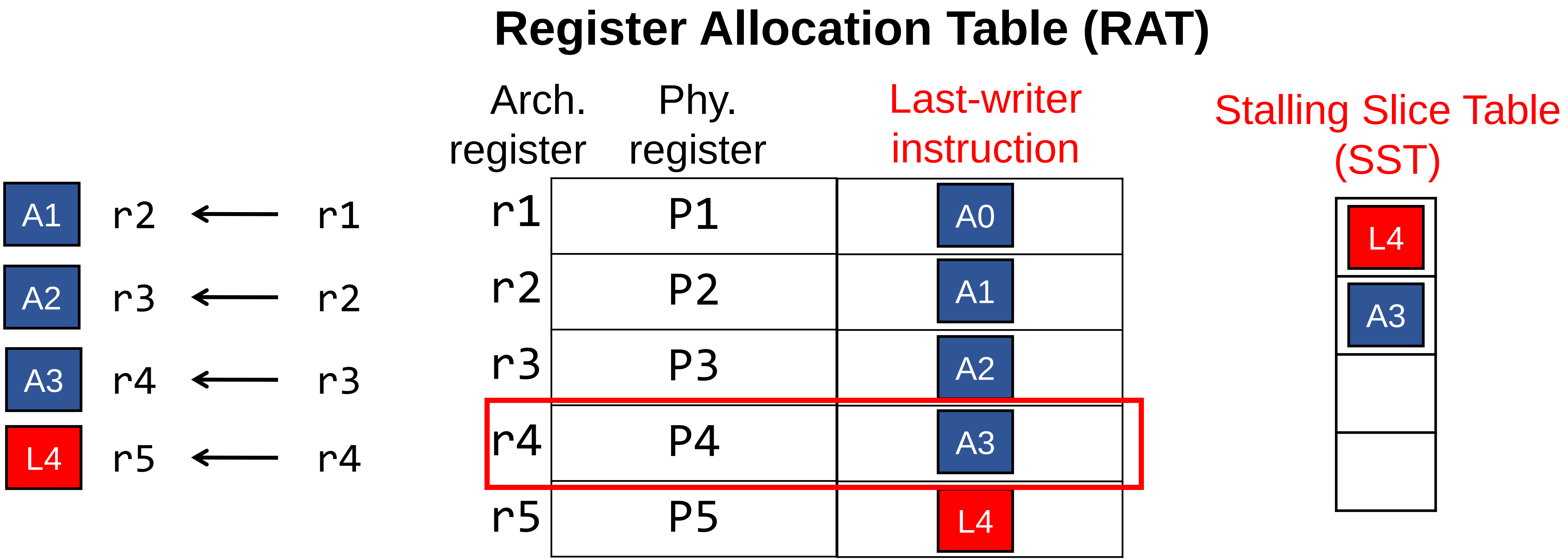
Arch. register	Phy. register	Last-writer instruction
r1	P1	A0
r2	P2	A1
r3	P3	A2
r4	P4	A3
r5	P5	L4

Stalling Slice Table (SST)

L4

Iteration-2:
L4 hits in the SST ⇒ While renaming source r4, read A3

Iteratively Identifying the Stalling Slices



A1	r2	←	r1
A2	r3	←	r2
A3	r4	←	r3
L4	r5	←	r4

Register Allocation Table (RAT)

Arch. register	Phy. register	Last-writer instruction
r1	P1	A0
r2	P2	A1
r3	P3	A2
r4	P4	A3
r5	P5	L4

Stalling Slice Table (SST)

L4
A3

Iteration-2:
L4 hits in the SST

While renaming source r4, read A3

Iteratively Identifying the Stalling Slices

A1 r2 ← r1

A2 r3 ← r2

A3 r4 ← r3

L4 r5 ← r4

Register Allocation Table (RAT)

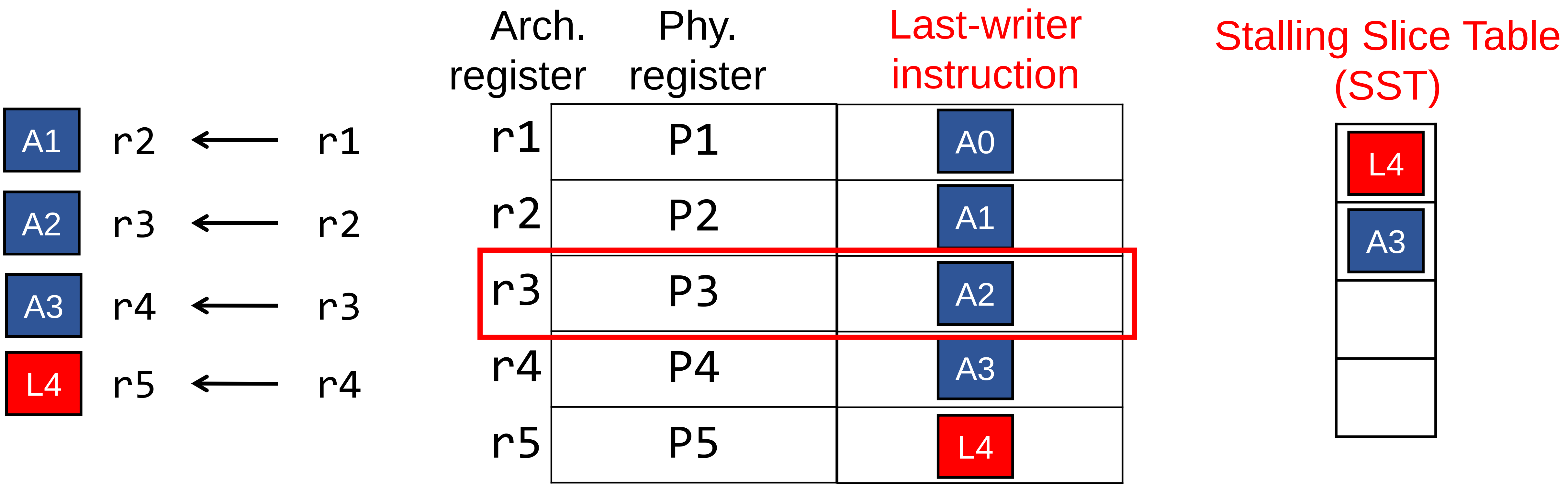
Arch. register	Phy. register	Last-writer instruction
r1	P1	A0
r2	P2	A1
r3	P3	A2
r4	P4	A3
r5	P5	L4

Stalling Slice Table (SST)

L4
A3

Iteration-3:
A3 hits in the SST

Iteratively Identifying the Stalling Slices



A1 r2 ← r1

A2 r3 ← r2

A3 r4 ← r3

L4 r5 ← r4

Register Allocation Table (RAT)

Arch. register	Phy. register	Last-writer instruction
r1	P1	A0
r2	P2	A1
r3	P3	A2
r4	P4	A3
r5	P5	L4

Stalling Slice Table (SST)

L4
A3

Iteration-3:
A3 hits in the SST ⇒ While renaming source r3, read A2

Iteratively Identifying the Stalling Slices

A1	r2	←	r1
A2	r3	←	r2
A3	r4	←	r3
L4	r5	←	r4

Register Allocation Table (RAT)

Arch. register	Phy. register	Last-writer instruction
r1	P1	A0
r2	P2	A1
r3	P3	A2
r4	P4	A3
r5	P5	L4

Stalling Slice Table (SST)
L4
A3
A2

Iteration-3:
A3 hits in the SST ⇒ While renaming source r3, read A2

Iteratively Identifying the Stalling Slices

A1 r2 ← r1

A2 r3 ← r2

A3 r4 ← r3

L4 r5 ← r4

Register Allocation Table (RAT)

Arch. register	Phy. register	Last-writer instruction
r1	P1	A0
r2	P2	A1
r3	P3	A2
r4	P4	A3
r5	P5	L4

Stalling Slice Table (SST)

L4
A3
A2

Iteration-4:
A2 hits in the SST

Iteratively Identifying the Stalling Slices

A1	r2	←	r1
A2	r3	←	r2
A3	r4	←	r3
L4	r5	←	r4

Register Allocation Table (RAT)

Arch. register	Phy. register	Last-writer instruction
r1	P1	A0
r2	P2	A1
r3	P3	A2
r4	P4	A3
r5	P5	L4

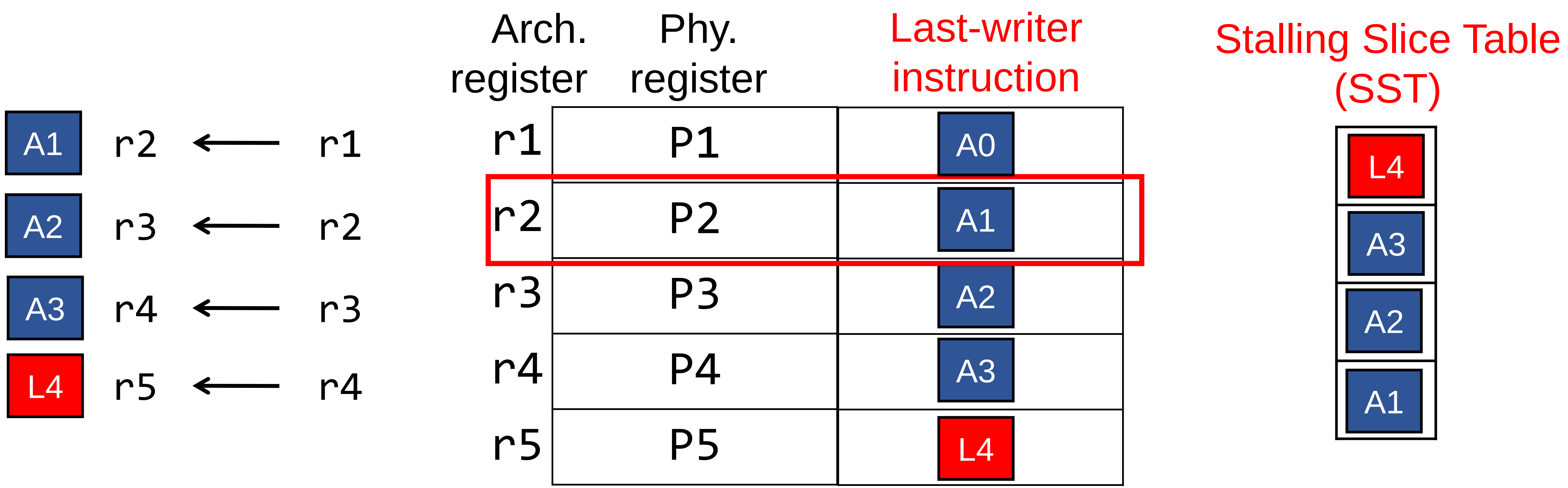
Stalling Slice Table (SST)

- L4
- A3
- A2

Iteration-4:
A2 hits in the SST → While renaming source r2, read A1

Iteratively Identifying the Stalling Slices

Register Allocation Table (RAT)



A1 r2 ← r1

A2 r3 ← r2

A3 r4 ← r3

L4 r5 ← r4

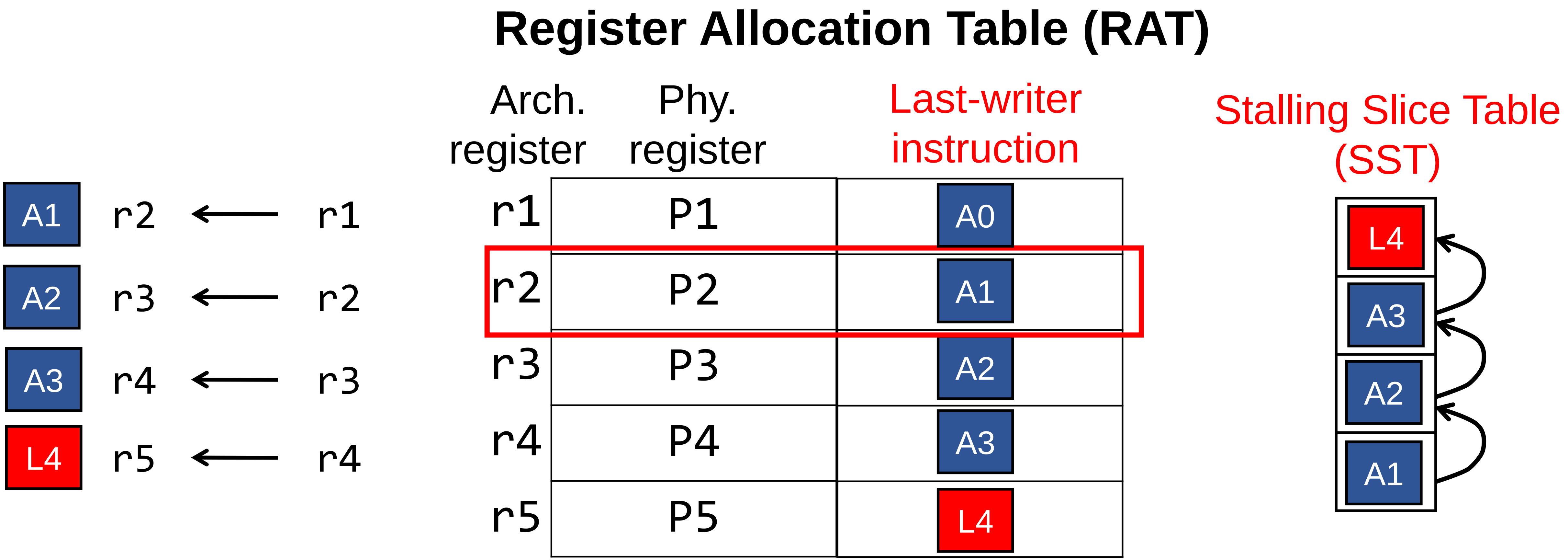
Arch. register	Phy. register	Last-writer instruction
r1	P1	A0
r2	P2	A1
r3	P3	A2
r4	P4	A3
r5	P5	L4

Stalling Slice Table (SST)

L4
A3
A2
A1

Iteration-4:
A2 hits in the SST ⇒ While renaming source r2, read A1

Iteratively Identifying the Stalling Slices



Register Allocation Table (RAT)

A1 r2 ← r1

A2 r3 ← r2

A3 r4 ← r3

L4 r5 ← r4

Arch. register	Phy. register	Last-writer instruction
r1	P1	A0
r2	P2	A1
r3	P3	A2
r4	P4	A3
r5	P5	L4

Stalling Slice Table (SST)

Iteration-4:
A2 hits in the SST

While renaming source r2, read A1

Runahead Register Reclamation

Runahead Register Reclamation

normal mode

Runahead Register Reclamation

normal mode

	instruction	dest	src1	src2	OldPhy register

Runahead Register Reclamation

normal mode

	instruction	dest	src1	src2	OldPhy register
I1	add r1 ← r2, r3	P1	P2	P3	P0

Runahead Register Reclamation

normal mode

	instruction	dest	src1	src2	OldPhy register
I1	add r1 ← r2, r3	P1	P2	P3	P0
I2	mul r2 ← r1, r4	P5	P1	P4	P2

Runahead Register Reclamation

normal mode

	instruction	dest	src1	src2	OldPhy register
I1	add r1 ← r2, r3	P1	P2	P3	P0
I2	mul r2 ← r1, r4	P5	P1	P4	P2
I3	ld r1 ← mem[x]	P6			P1

Runahead Register Reclamation

normal mode

	instruction	dest	src1	src2	OldPhy register
I1	add r1 ← r2, r3	P1	P2	P3	P0
I2	mul r2 ← r1, r4	P5	P1	P4	P2
I3	ld r1 ← mem[x]	P6			P1
I4	add r2 ← r1, r3	P7	P6	P3	P5

Runahead Register Reclamation

normal mode

	instruction	dest	src1	src2	OldPhy register
I1	add r1 ← r2, r3	P1	P2	P3	P0
I2	mul r2 ← r1, r4	P5	P1	P4	P2
I3	ld r1 ← mem[x]	P6			P1
I4	add r2 ← r1, r3	P7	P6	P3	P5
I5	add r2 ← r4, r5	P9	P4	P8	P7

Runahead Register Reclamation

normal mode

	instruction	dest	src1	src2	OldPhy register
I1	add r1 ← r2, r3	P1	P2	P3	P0
I2	mul r2 ← r1, r4	P5	P1	P4	P2
I3	ld r1 ← mem[x]	P6			P1
I4	add r2 ← r1, r3	P7	P6	P3	P5
I5	add r2 ← r4, r5	P9	P4	P8	P7
I6	sub r1 ← r2, r6	P11	P9	P10	P6

Runahead Register Reclamation

normal mode

runahead mode

	instruction	dest	src1	src2	OldPhy register
I1	add r1 ← r2, r3	P1	P2	P3	P0
I2	mul r2 ← r1, r4	P5	P1	P4	P2
I3	ld r1 ← mem[x]	P6			P1
I4	add r2 ← r1, r3	P7	P6	P3	P5
I5	add r2 ← r4, r5	P9	P4	P8	P7
I6	sub r1 ← r2, r6	P11	P9	P10	P6

Runahead Register Reclamation

normal mode

	instruction	dest	src1	src2	OldPhy register
I1	add r1 ← r2, r3	P1	P2	P3	P0
I2	mul r2 ← r1, r4	P5	P1	P4	P2
I3	ld r1 ← mem[x]	P6			P1
I4	add r2 ← r1, r3	P7	P6	P3	P5
I5	add r2 ← r4, r5	P9	P4	P8	P7
I6	sub r1 ← r2, r6	P11	P9	P10	P6

runahead mode

	OldPhy register
I1	P0
I2	P2
I3	P1
I4	P5
I5	P7
I6	P6

Runahead Register Reclamation

runahead mode

	OldPhy register
I1	P0
I2	P2
I3	P1
I4	P5
I5	P7
I6	P6

Runahead Register Reclamation

runahead mode

	OldPhy register
I1	P0
I2	P2
I3	P1
I4	P5
I5	P7
I6	P6

Precise Register Deallocation Queue (PRDQ)

Runahead Register Reclamation

runahead mode

dispatch ⇨

	OldPhy register
I1	P0
I2	P2
I3	P1
I4	P5
I5	P7
I6	P6

Precise Register Deallocation Queue (PRDQ)

Runahead Register Reclamation

runahead mode

dispatch ⇨

	OldPhy register	Executed ?
I1	P0	
I2	P2	
I3	P1	
I4	P5	
I5	P7	
I6	P6	

Precise Register Deallocation Queue (PRDQ)

Runahead Register Reclamation

runahead mode

dispatch ⇨

	OldPhy register	Executed ?
I1	P0	0
I2	P2	0
I3	P1	0
I4	P5	0
I5	P7	0
I6	P6	0

Precise Register Deallocation Queue (PRDQ)

Runahead Register Reclamation

runahead mode

	OldPhy register	Executed ?
I1	P0	0
I2	P2	0
I3	P1	0
I4	P5	0
I5	P7	0
I6	P6	0

dispatch

execute

Precise Register Deallocation Queue (PRDQ)

Runahead Register Reclamation

runahead mode

dispatch

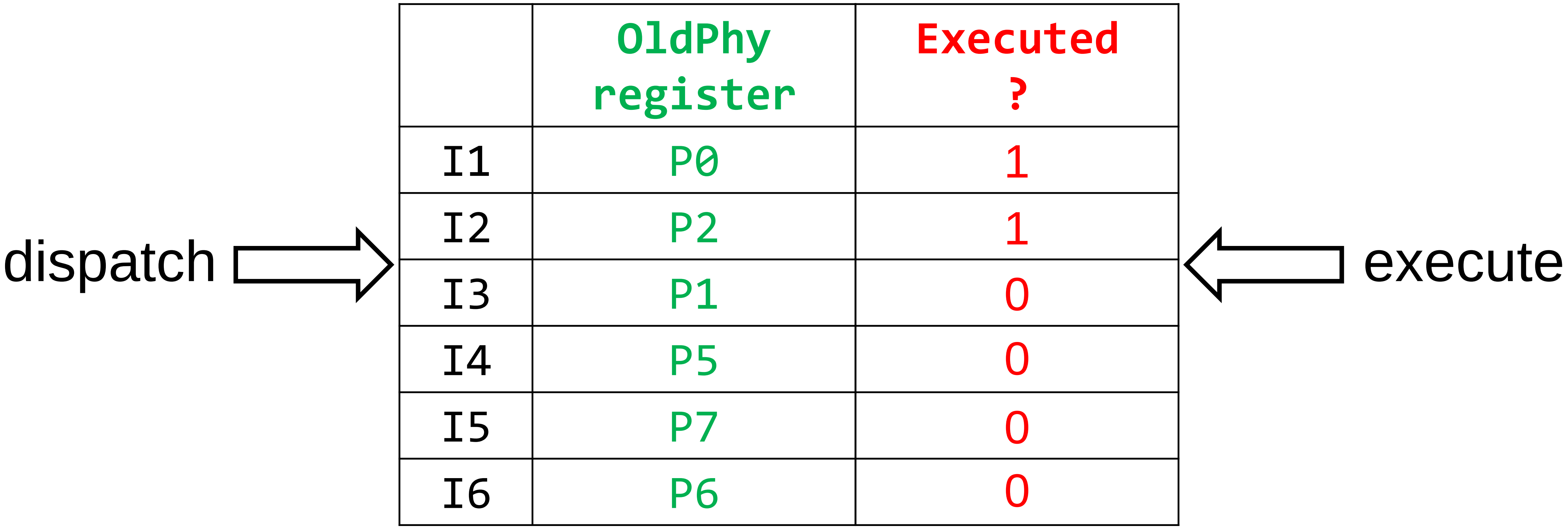
	OldPhy register	Executed ?
I1	P0	1
I2	P2	0
I3	P1	0
I4	P5	0
I5	P7	0
I6	P6	0

execute

Precise Register Deallocation Queue (PRDQ)

Runahead Register Reclamation

runahead mode



	OldPhy register	Executed ?
I1	P0	1
I2	P2	1
I3	P1	0
I4	P5	0
I5	P7	0
I6	P6	0

Precise Register Deallocation Queue (PRDQ)

Runahead Register Reclamation

runahead mode

dispatch

	OldPhy register	Executed ?
I1	P0	1
I2	P2	1
I3	P1	0
I4	P5	0
I5	P7	1
I6	P6	0

execute

Precise Register Deallocation Queue (PRDQ)

Runahead Register Reclamation

runahead mode

dispatch

	OldPhy register	Executed ?
I1	P0 ✓	1
I2	P2 ✓	1
I3	P1	0
I4	P5	0
I5	P7	1
I6	P6	0

execute

Precise Register Deallocation Queue (PRDQ)

Runahead Register Reclamation

runahead mode

dispatch

execute

	OldPhy register	Executed ?
I1	P0 ✓	1
I2	P2 ✓	1
I3	P1	0
I4	P5	0
I5	P7 ✗	1
I6	P6	0

Precise Register Deallocation Queue (PRDQ)

Runahead Register Reclamation

runahead mode

dispatch

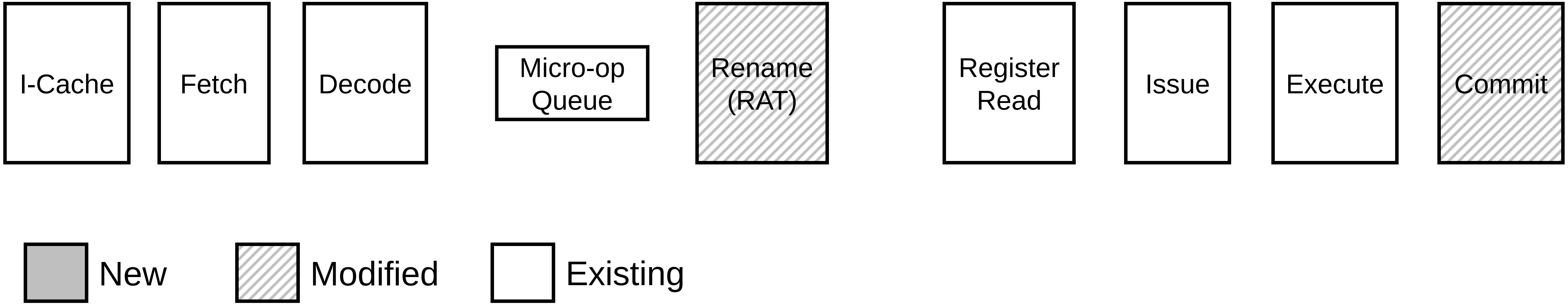
execute

	OldPhy register	Executed ?
I1	P0 ✓	1
I2	P2 ✓	1
I3	P1	0
I4	P5	0
I5	P7 ✗	1
I6	P6	0

Precise Register Deallocation Queue (PRDQ)

Putting it All Together

Putting it All Together



I-Cache

Fetch

Decode

Micro-op Queue

Rename (RAT)

Register Read

Issue

Execute

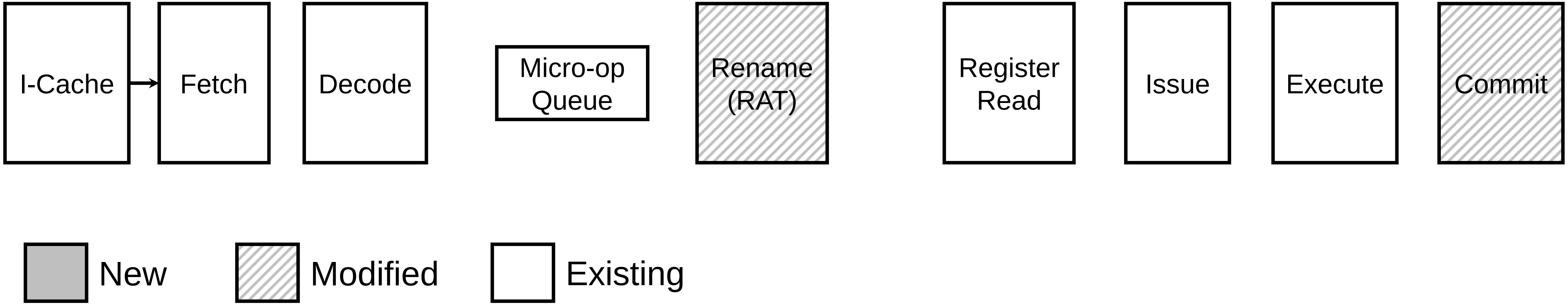
Commit

■ New

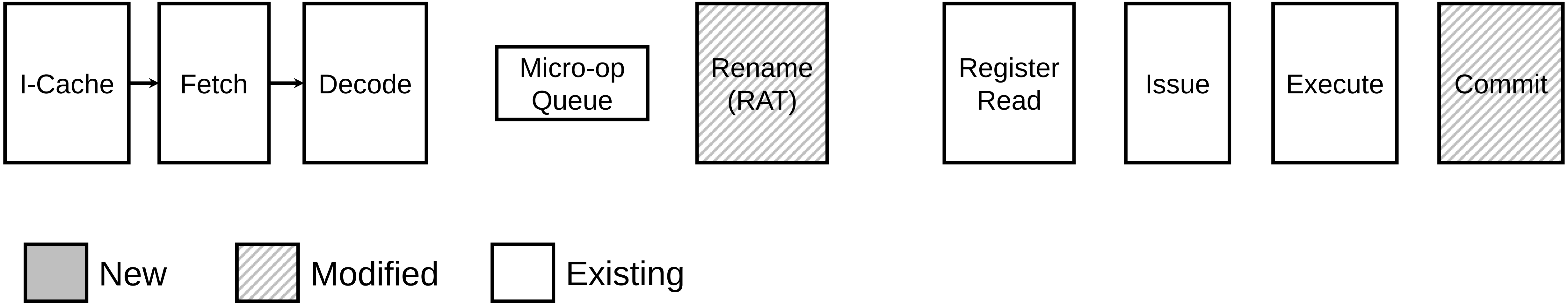
▨ Modified

□ Existing

Putting it All Together



Putting it All Together



I-Cache

Fetch

Decode

Micro-op Queue

Rename (RAT)

Register Read

Issue

Execute

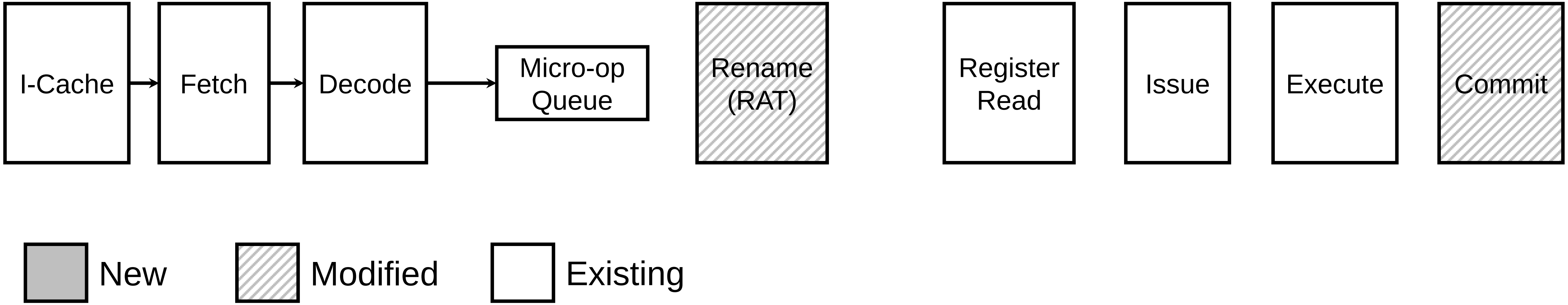
Commit

New

Modified

Existing

Putting it All Together



I-Cache

Fetch

Decode

Micro-op Queue

Rename (RAT)

Register Read

Issue

Execute

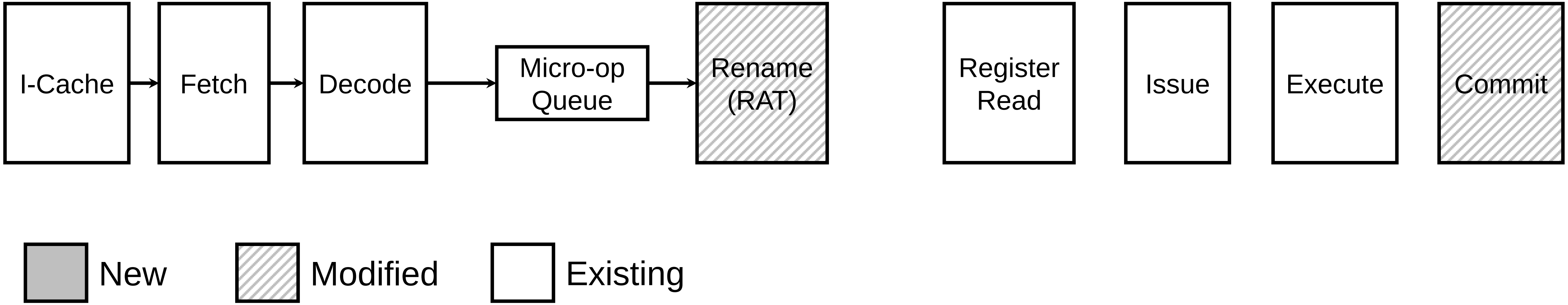
Commit

■ New

▨ Modified

□ Existing

Putting it All Together



I-Cache → Fetch → Decode → Micro-op Queue → Rename (RAT)

Register Read

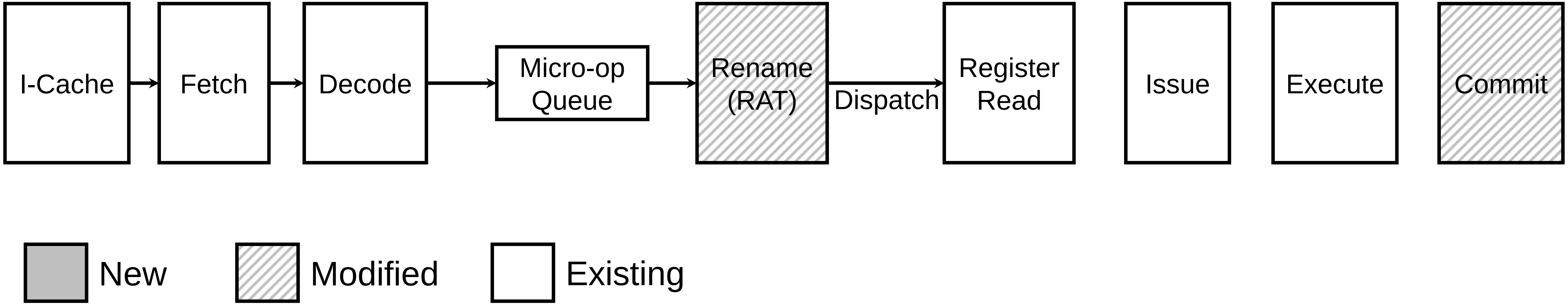
Issue

Execute

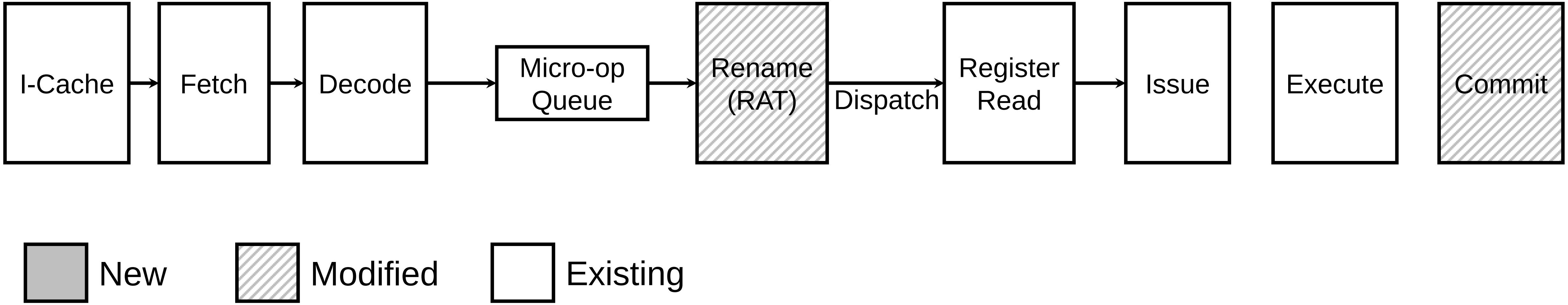
Commit

■ New ▨ Modified □ Existing

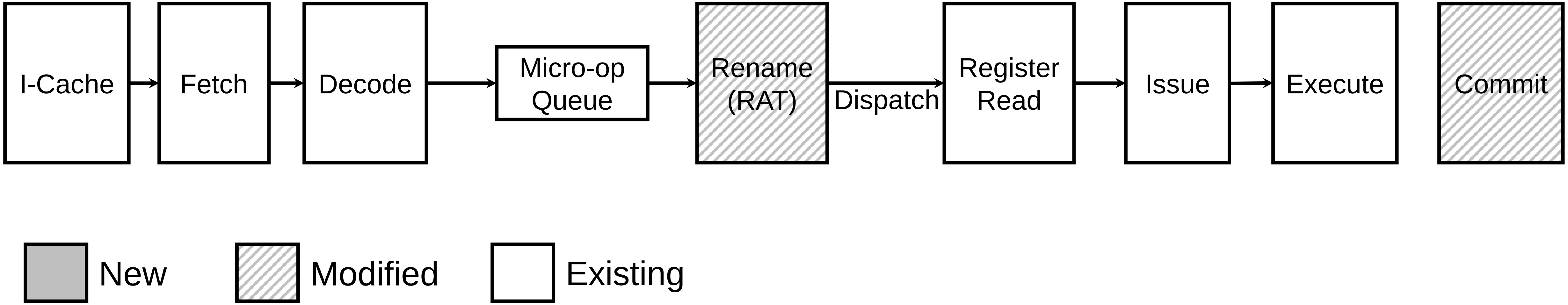
Putting it All Together



Putting it All Together



Putting it All Together


I-Cache

Fetch

Decode

Micro-op Queue

Rename (RAT)

Dispatch

Register Read

Issue

Execute

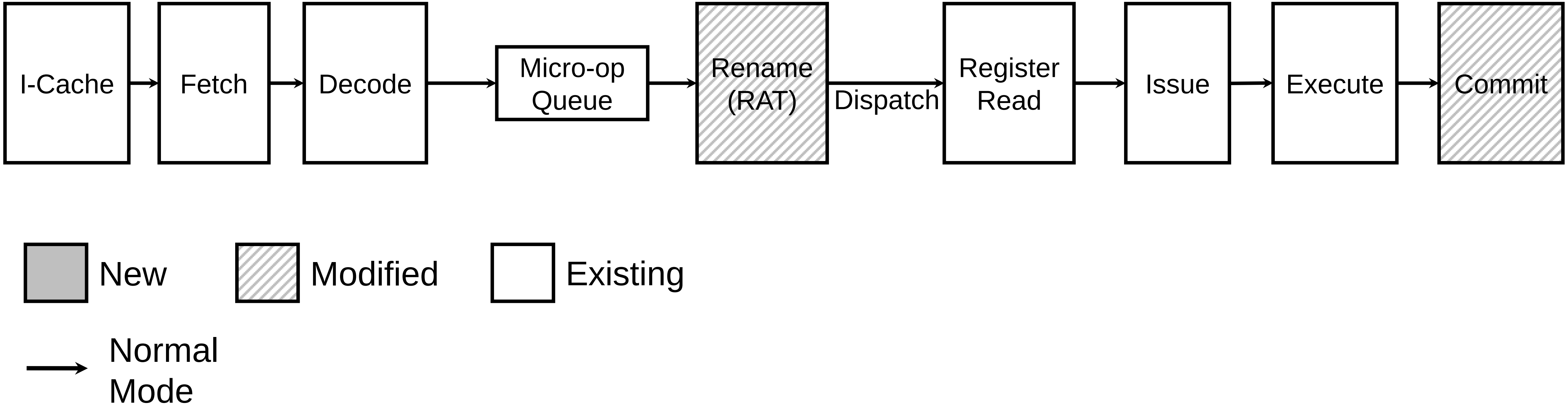
Commit

■ New

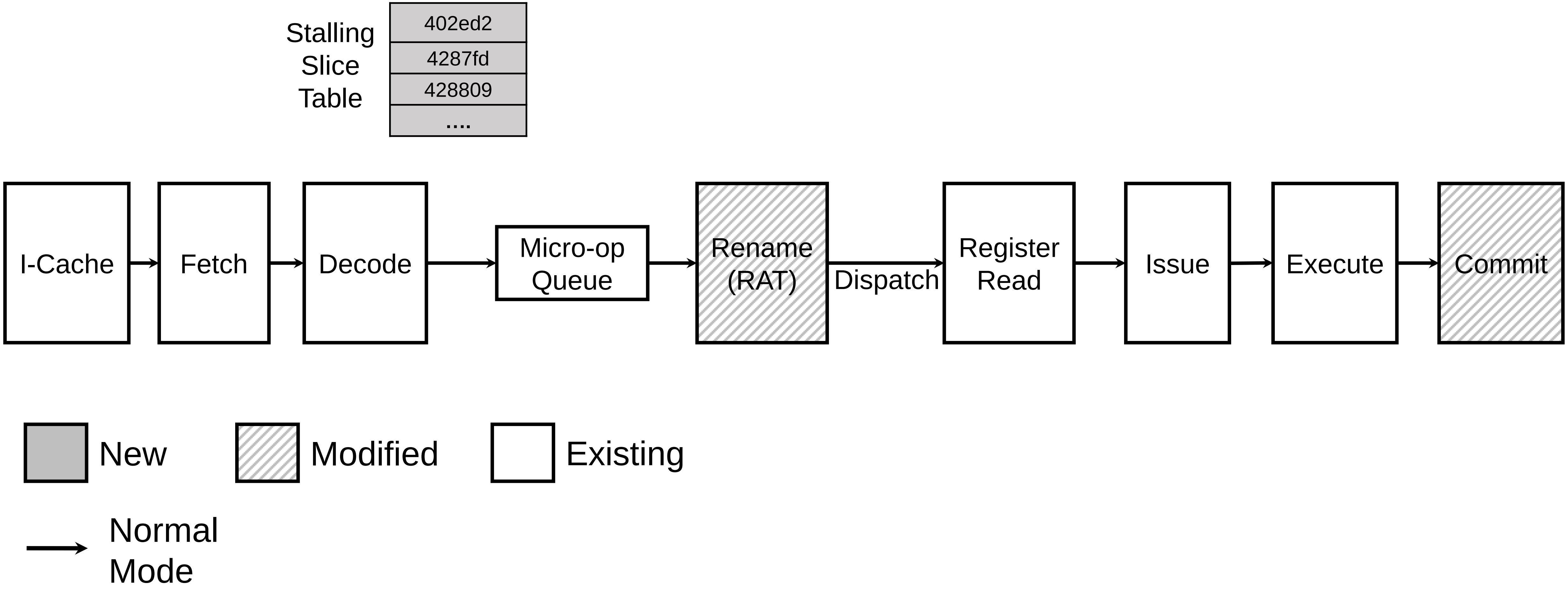
▨ Modified

□ Existing

Putting it All Together



Putting it All Together



Stalling Slice Table

402ed2
4287fd
428809
....

I-Cache → Fetch → Decode → Micro-op Queue → Rename (RAT) → Dispatch → Register Read → Issue → Execute → Commit

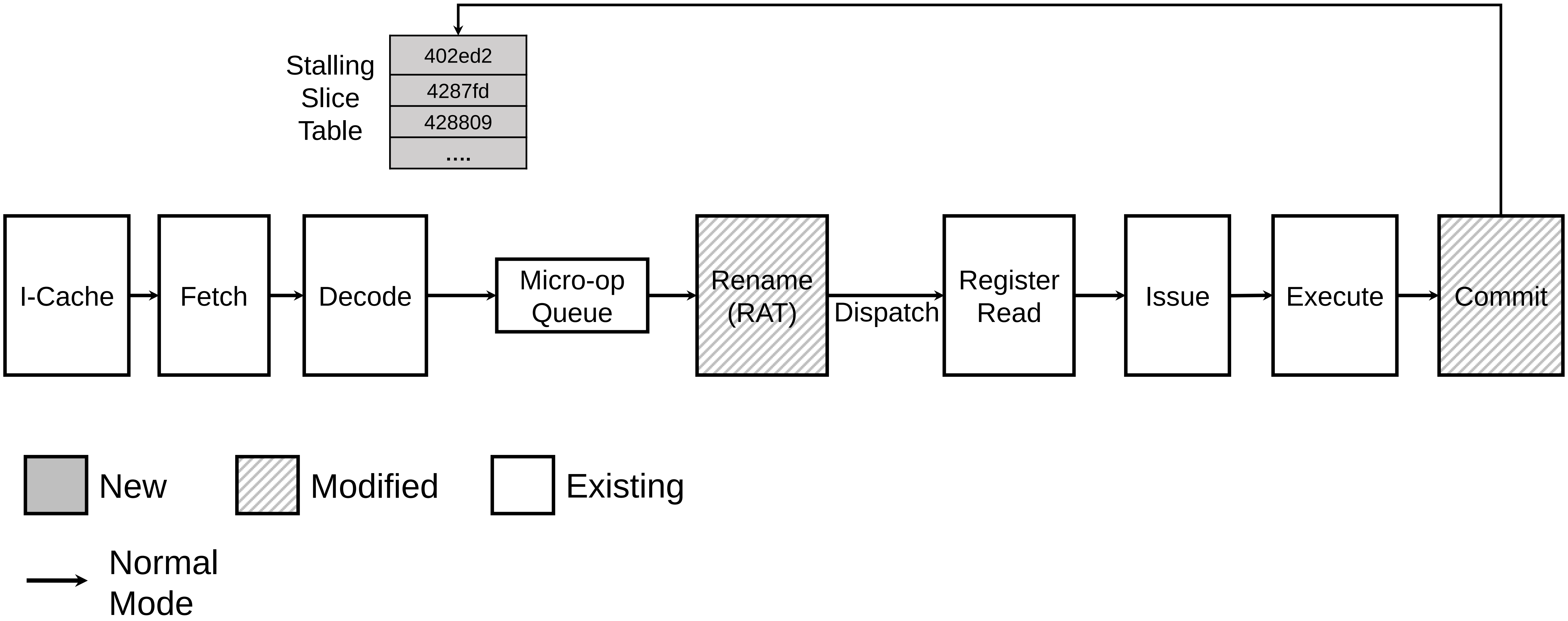
▩ New

▨ Modified

□ Existing

→ Normal Mode

Putting it All Together



Stalling Slice Table

402ed2

4287fd

428809

....

I-Cache

Fetch

Decode

Micro-op Queue

Rename (RAT)

Dispatch

Register Read

Issue

Execute

Commit

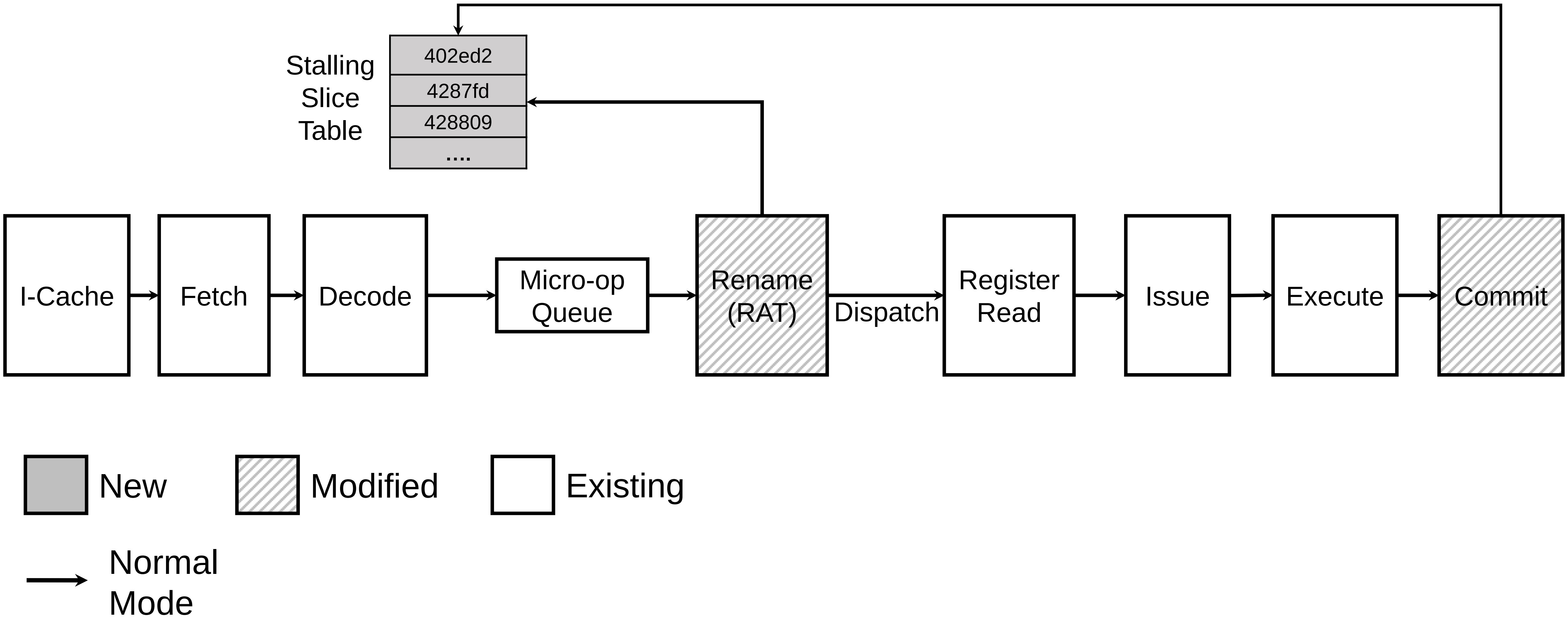
New

Modified

Existing

Normal Mode

Putting it All Together



Stalling Slice Table

402ed2

4287fd

428809

....

I-Cache

Fetch

Decode

Micro-op Queue

Rename (RAT)

Dispatch

Register Read

Issue

Execute

Commit

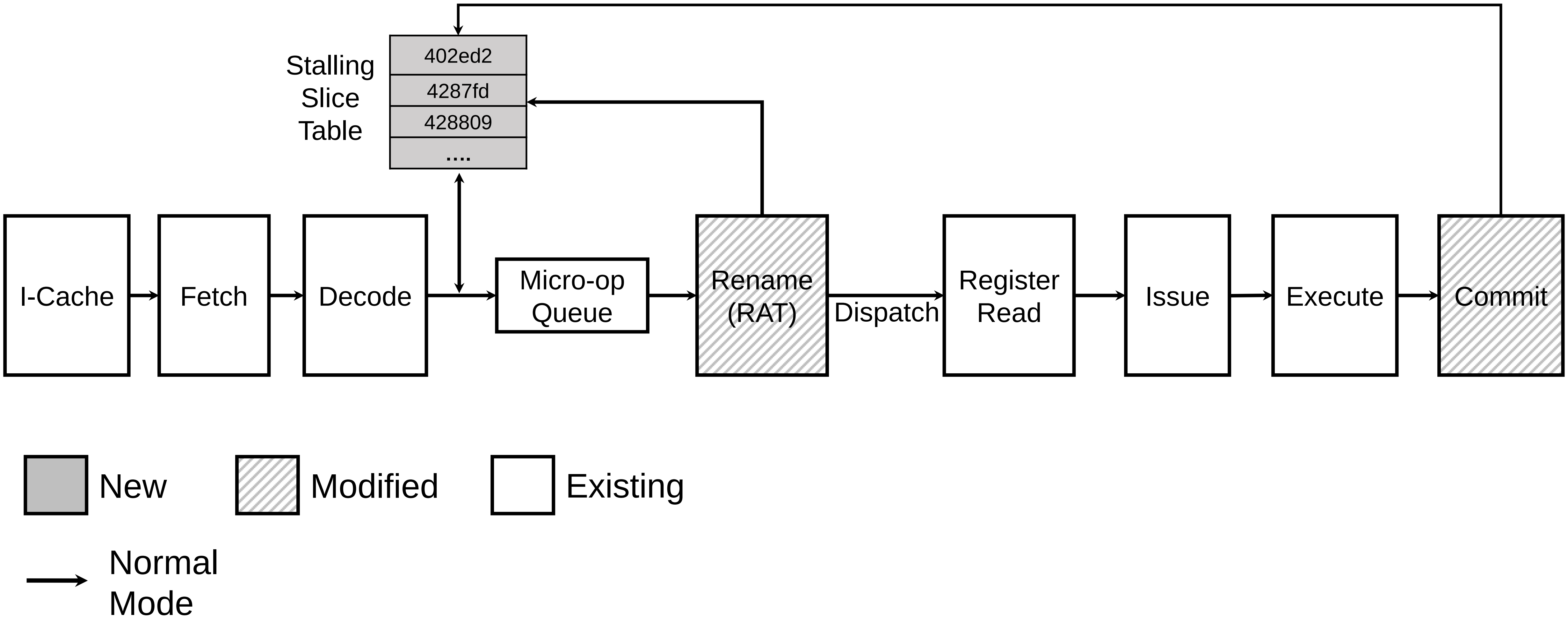
■ New

▨ Modified

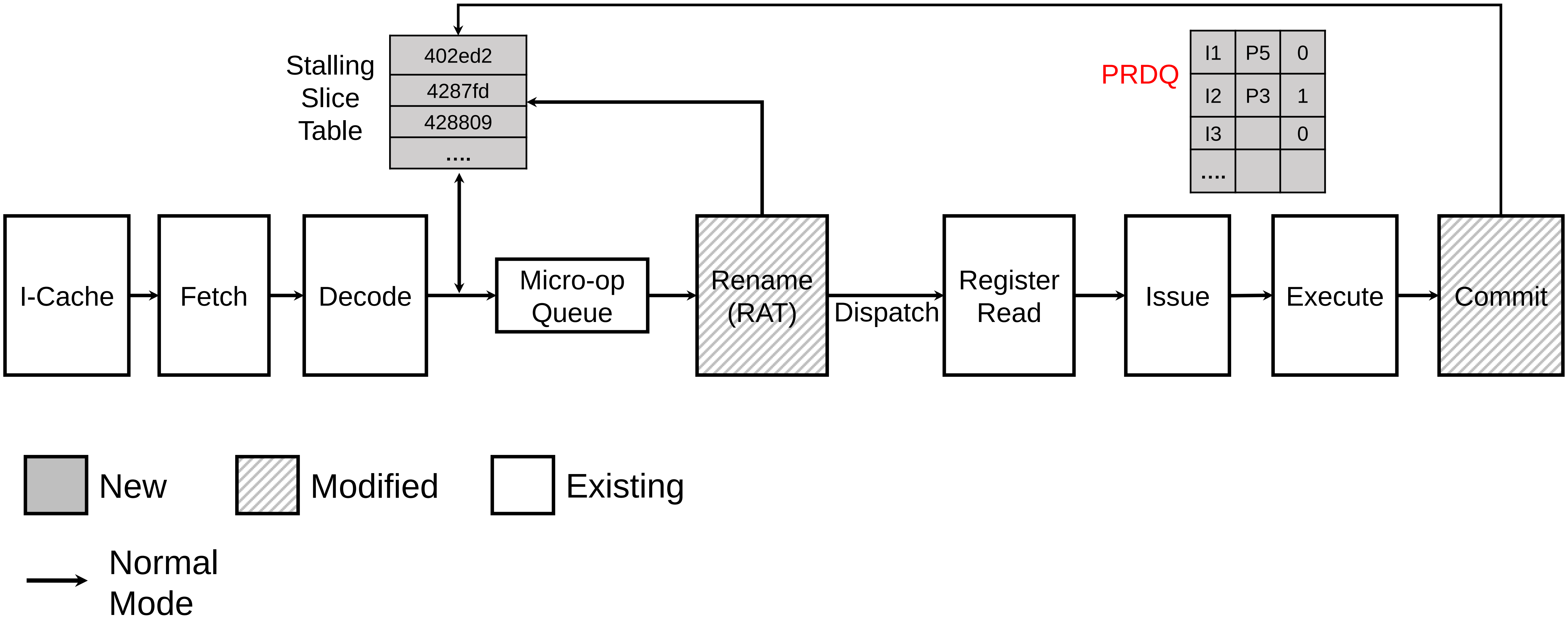
□ Existing

→ Normal Mode

Putting it All Together



Putting it All Together



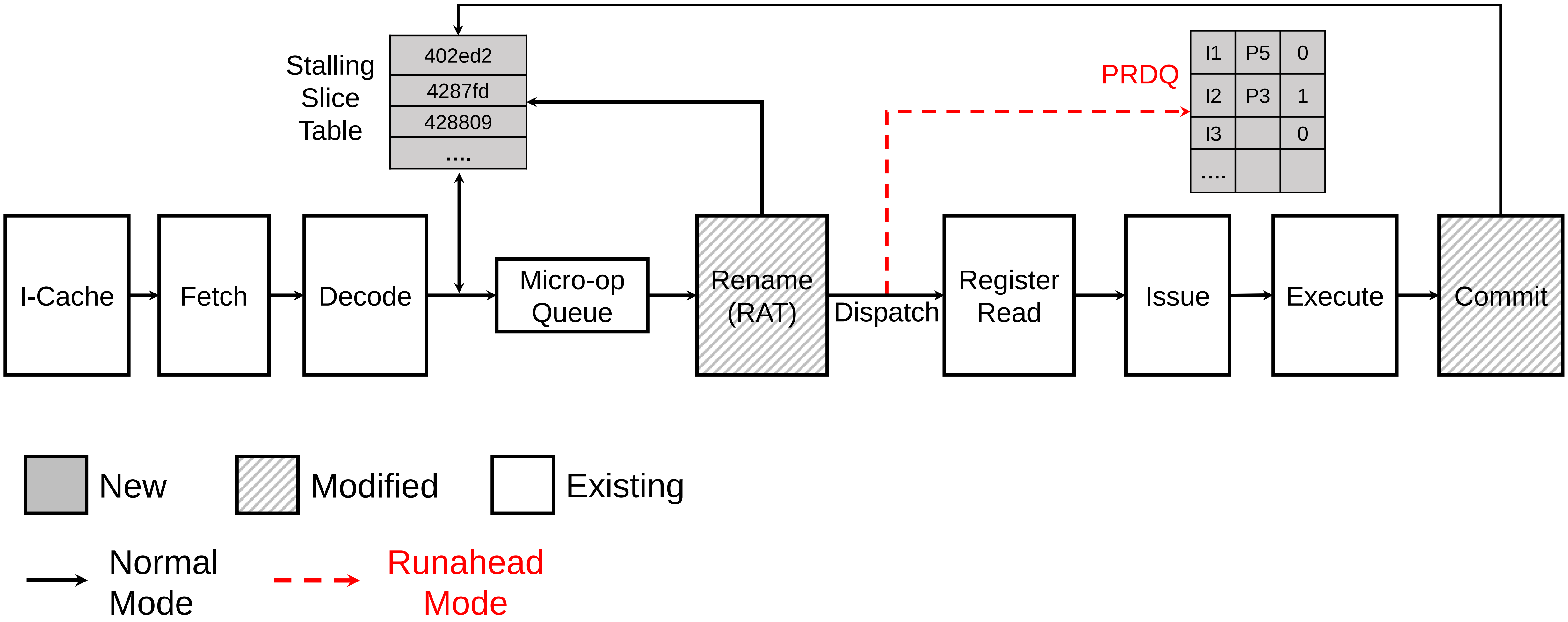
Stalling Slice Table

402ed2
4287fd
428809
....

PRDQ

I1	P5	0
I2	P3	1
I3		0
....		

Putting it All Together



Stalling Slice Table

402ed2
4287fd
428809
....

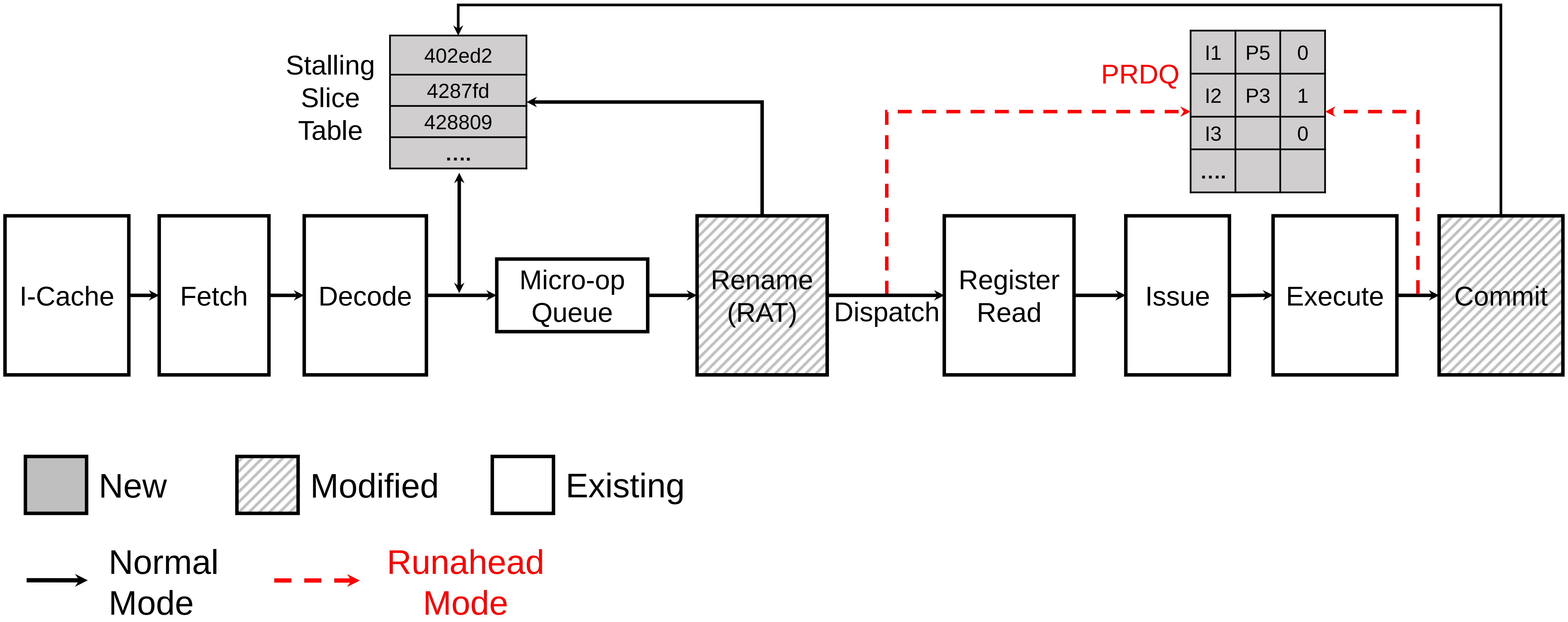
PRDQ

I1	P5	0
I2	P3	1
I3		0
....		

■ New ▨ Modified □ Existing

→ Normal Mode - - → Runahead Mode

Putting it All Together



Stalling Slice Table

402ed2
4287fd
428809
....

PRDQ

I1	P5	0
I2	P3	1
I3		0
....		

Evaluation

Evaluation

Simulator: Sniper 6.0, McPAT

Evaluation

Simulator: Sniper 6.0, McPAT

Workloads: SPEC CPU2006/CPU2017, 1B SimPoints

Evaluation

Simulator: Sniper 6.0, McPAT

Workloads: SPEC CPU2006/CPU2017, 1B SimPoints

Baseline: ROB=192, issue queue=92, load/store queue=64, register file=168/168

Evaluation

OoO: Baseline out-of-order core

Evaluation

OoO: Baseline out-of-order core

RA: Runahead execution*

-- No short runahead intervals

*[Mutlu et al. ISCA’05]

Evaluation

OoO: Baseline out-of-order core

RA: Runahead execution*

-- No short runahead intervals

-- No overlapping intervals

*[Mutlu et al. ISCA’05]

Evaluation

OoO: Baseline out-of-order core

RA: Runahead execution*

-- No short runahead intervals

-- No overlapping intervals

RA-buffer: Runahead buffer**

*[Mutlu et al. ISCA'05]

Evaluation

OoO: Baseline out-of-order core

RA: Runahead execution*

-- No short runahead intervals

-- No overlapping intervals

RA-buffer: Runahead buffer**

RA-hybrid: Better performing mechanism between RA-buffer and RA

*[Mutlu et al. ISCA’05] **[Hashemi et al. MICRO’15]

Evaluation

OoO: Baseline out-of-order core

RA: Runahead execution*

-- No short runahead intervals

-- No overlapping intervals

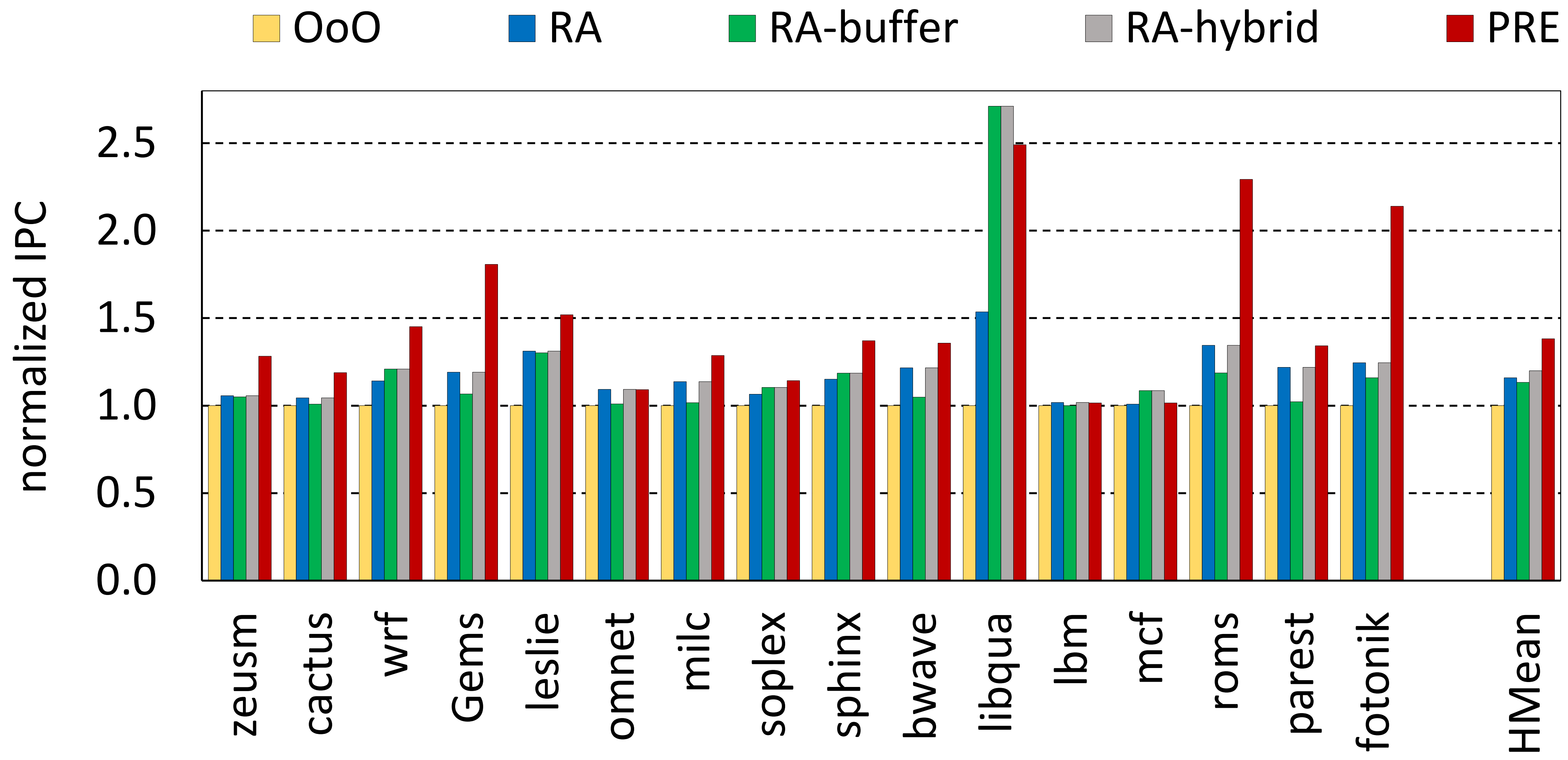
PRE:
Precise runahead execution***

RA-buffer: Runahead buffer**

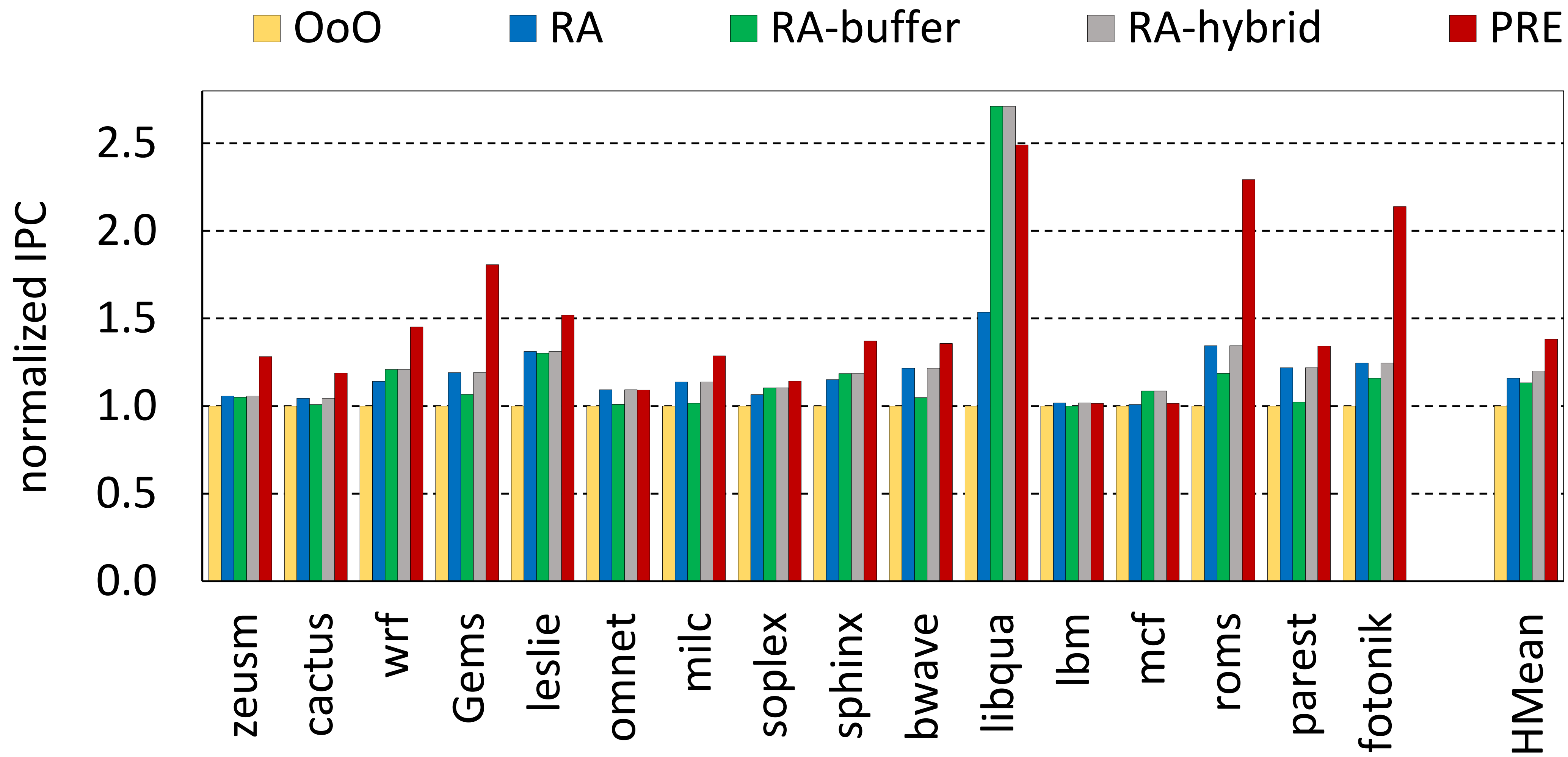
RA-hybrid: Better performing mechanism between RA-buffer and RA

*[Mutlu et al. ISCA'05] **[Hashemi et al. MICRO'15] ***[Naithani et al. HPCA'20]

Evaluation – Performance

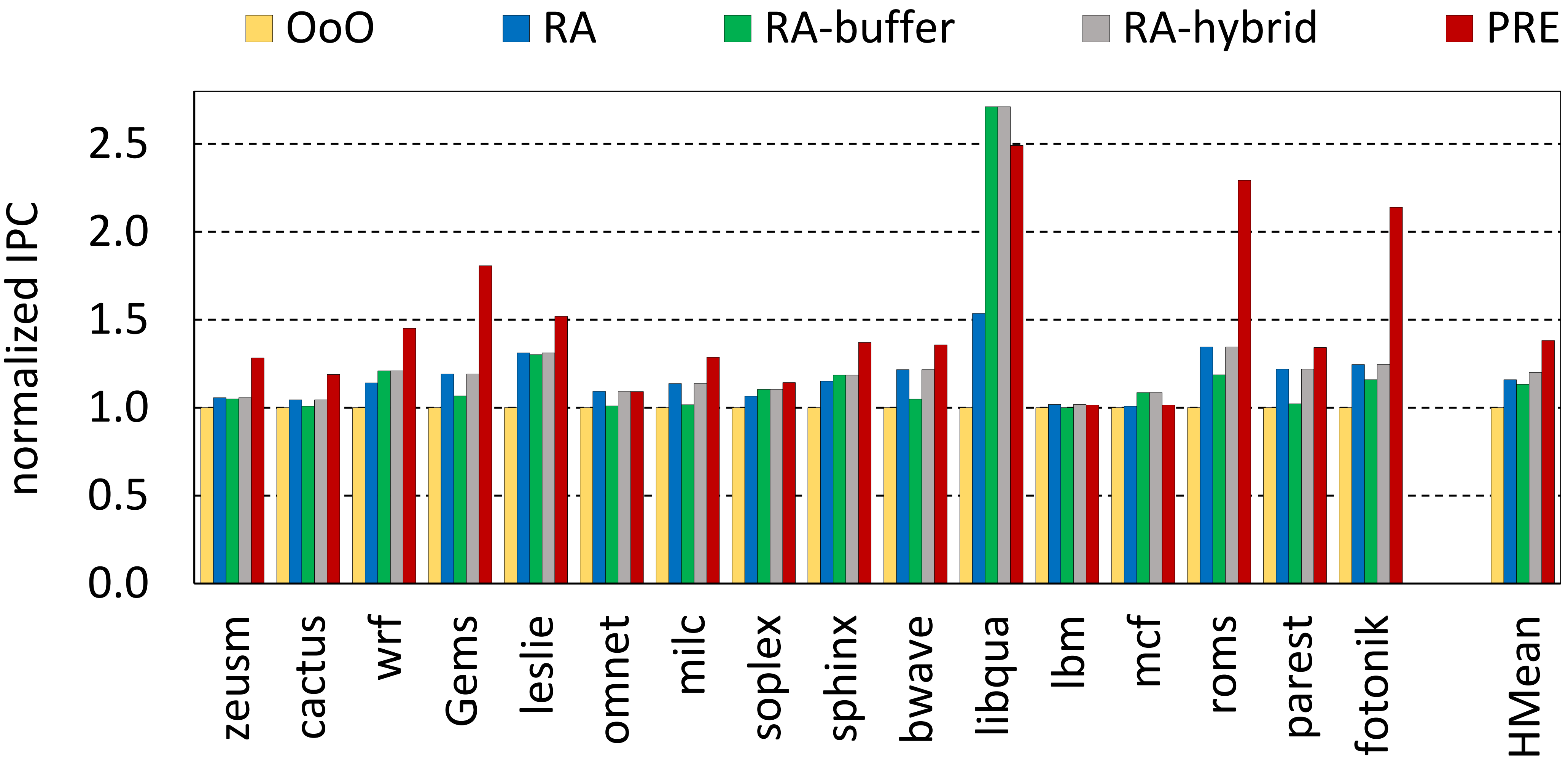


Evaluation – Performance



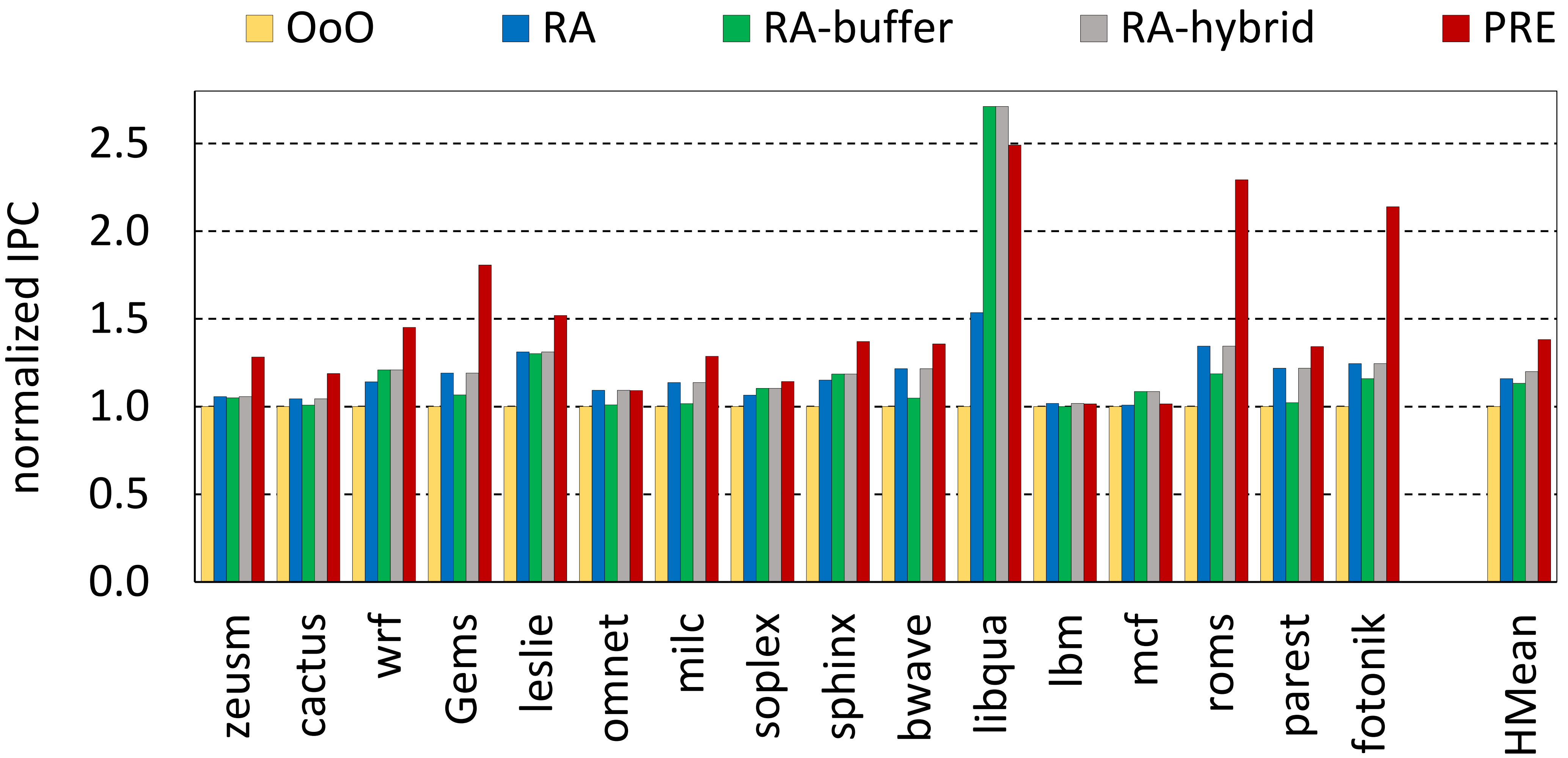
RA: 15.9%

Evaluation – Performance



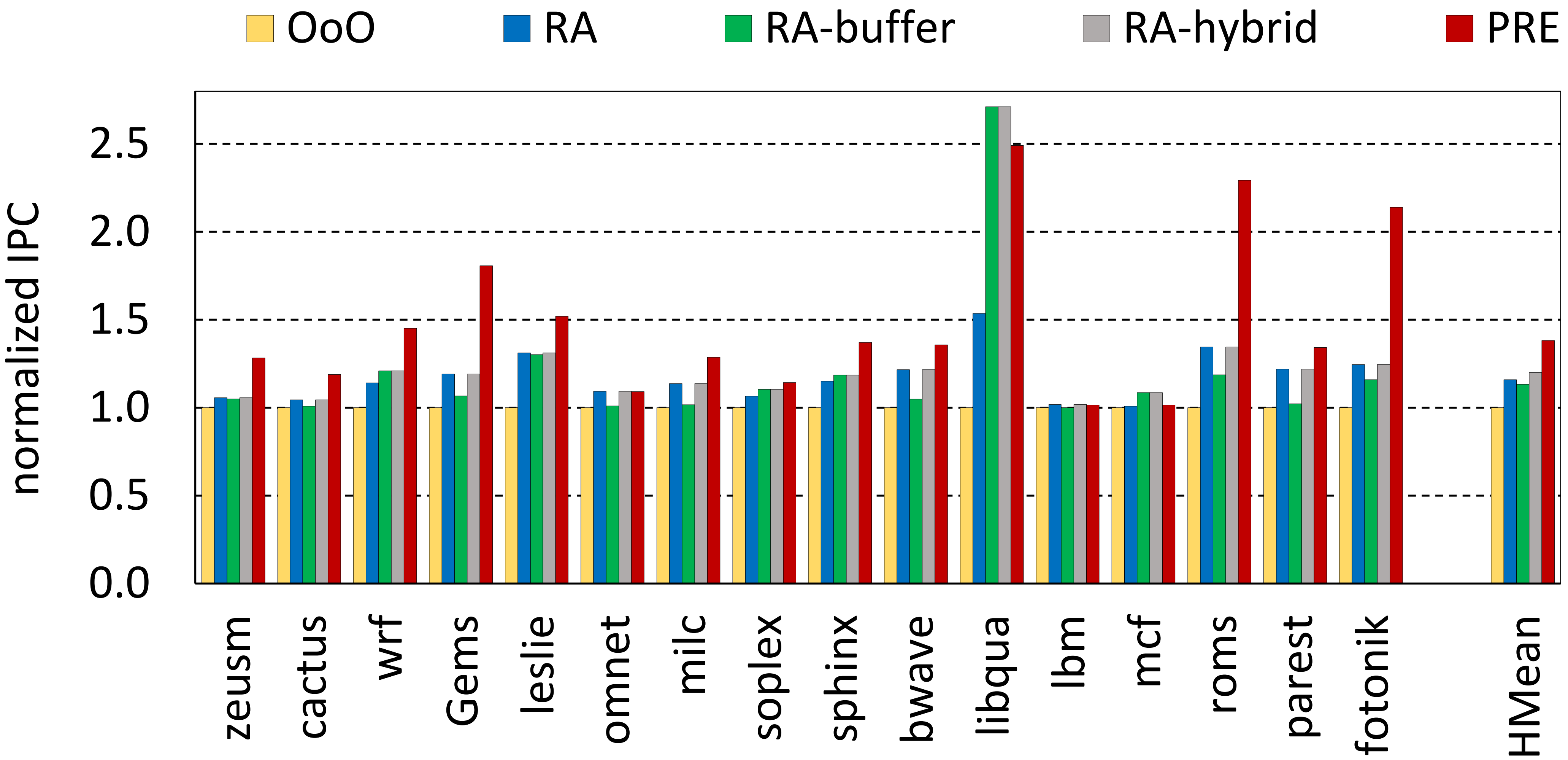
RA: 15.9% RA-buffer: 13.3%

Evaluation – Performance



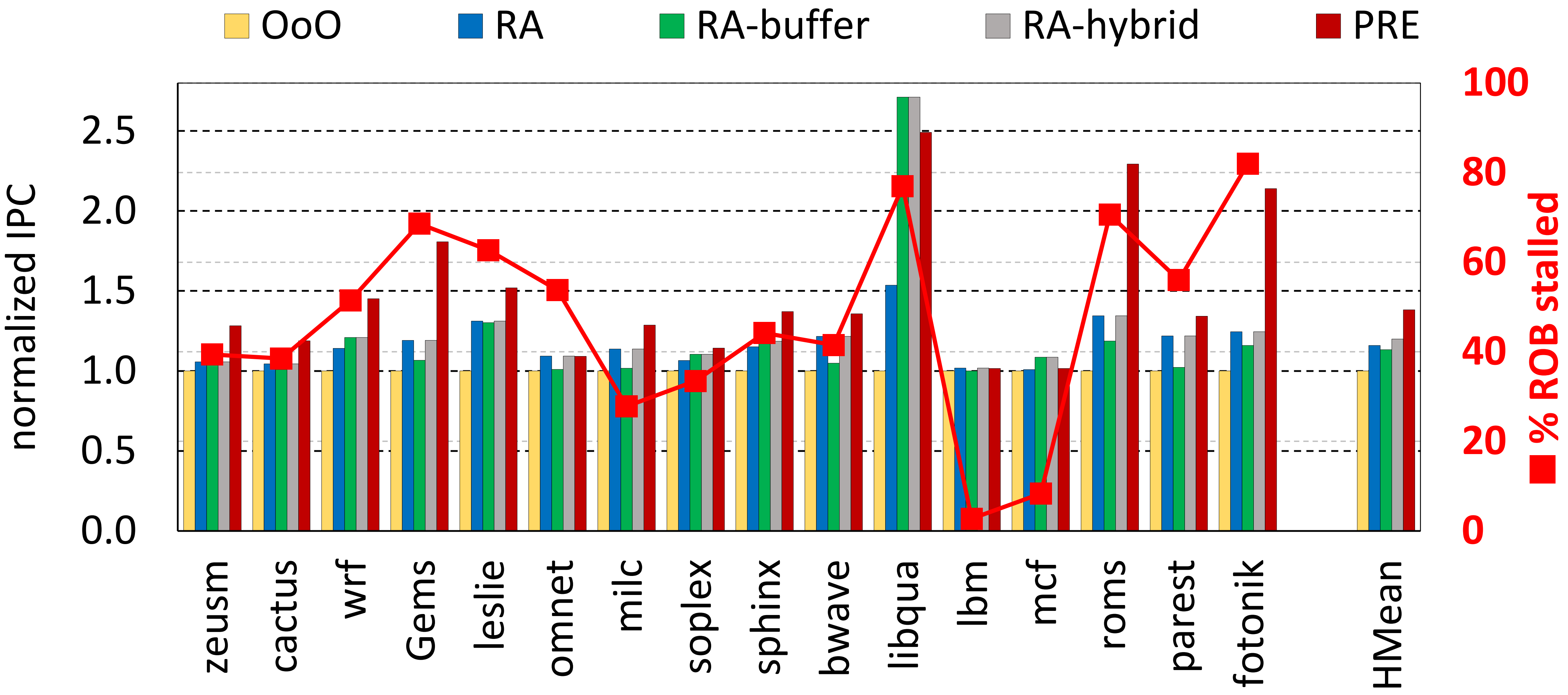
RA: 15.9%   RA-buffer: 13.3%   RA-hybrid: 20%

Evaluation – Performance



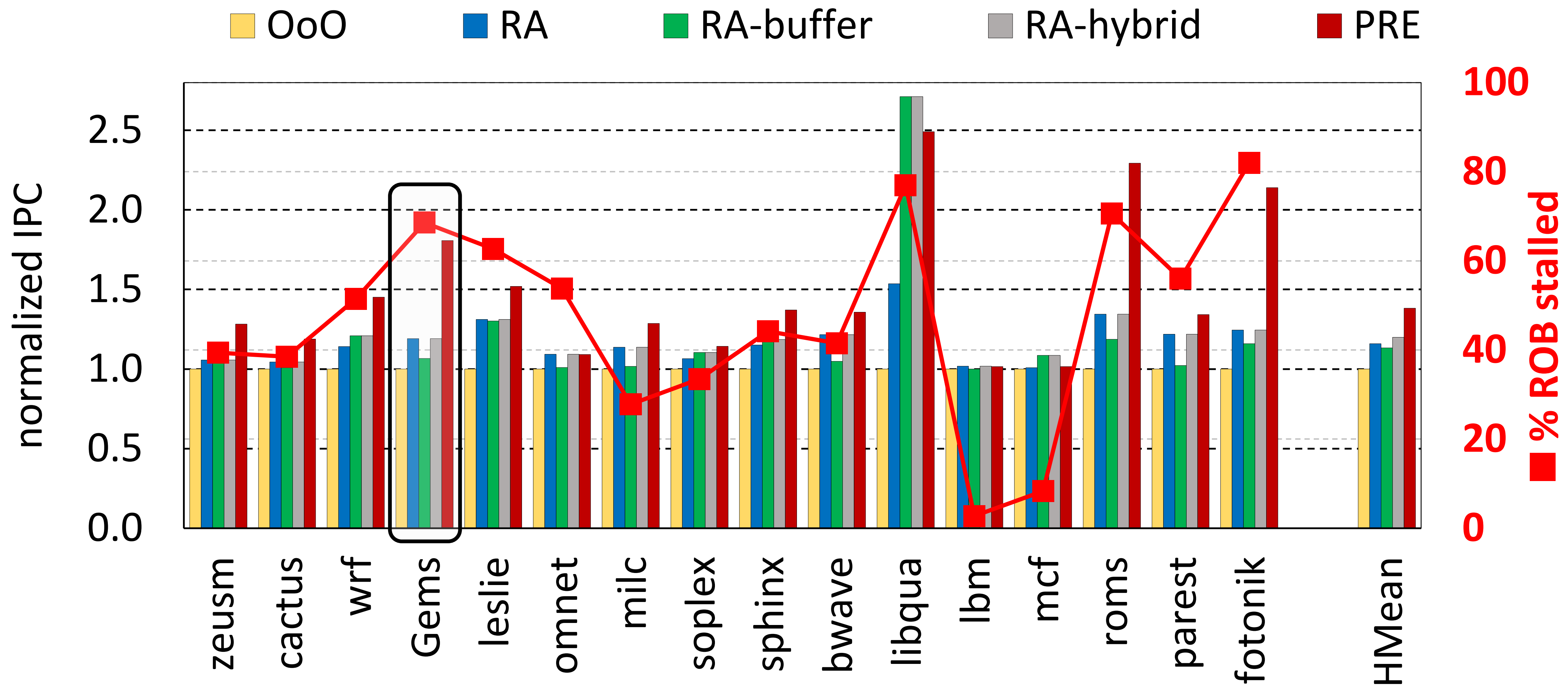
RA: 15.9% **RA-buffer: 13.3%** **RA-hybrid: 20%** **PRE: 38.2%**

Evaluation – Performance



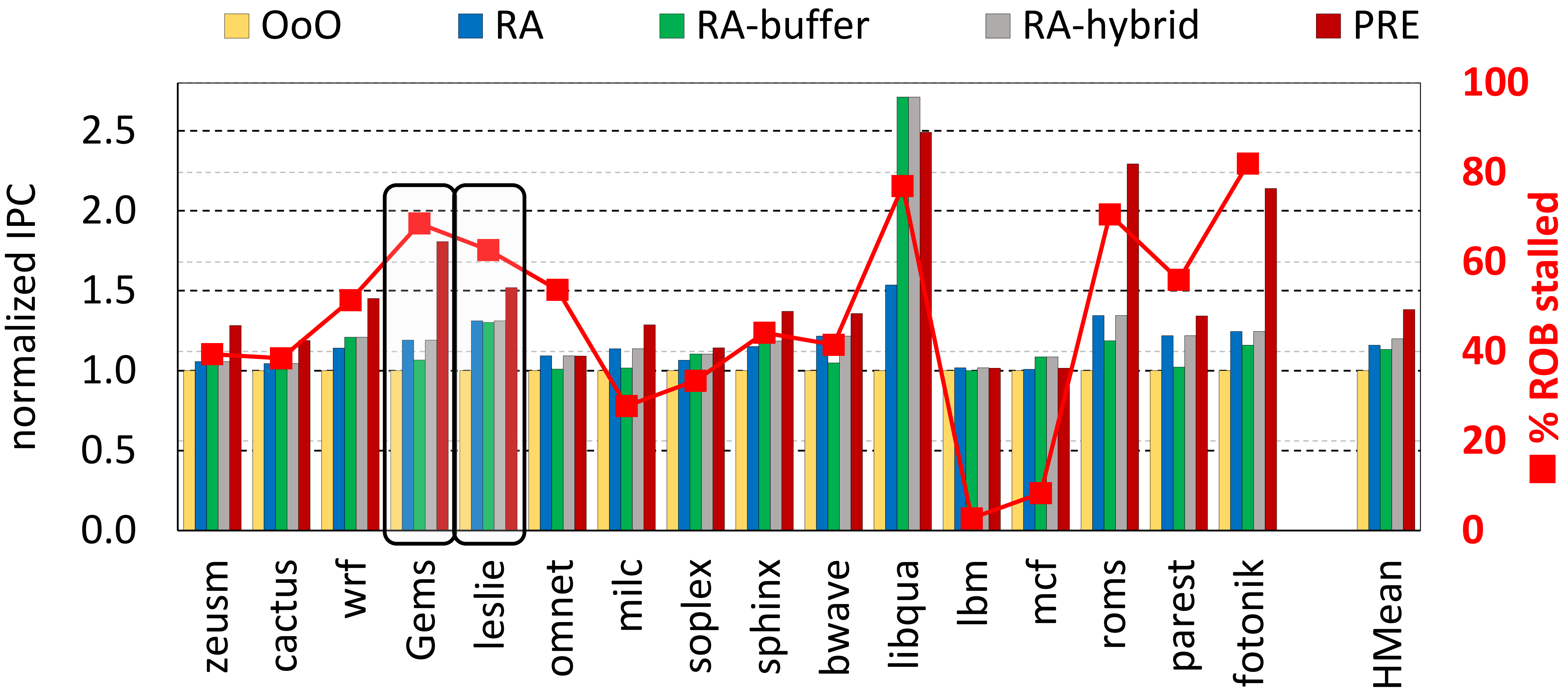
RA: 15.9% **RA-buffer: 13.3%** **RA-hybrid: 20%** **PRE: 38.2%**

Evaluation – Performance



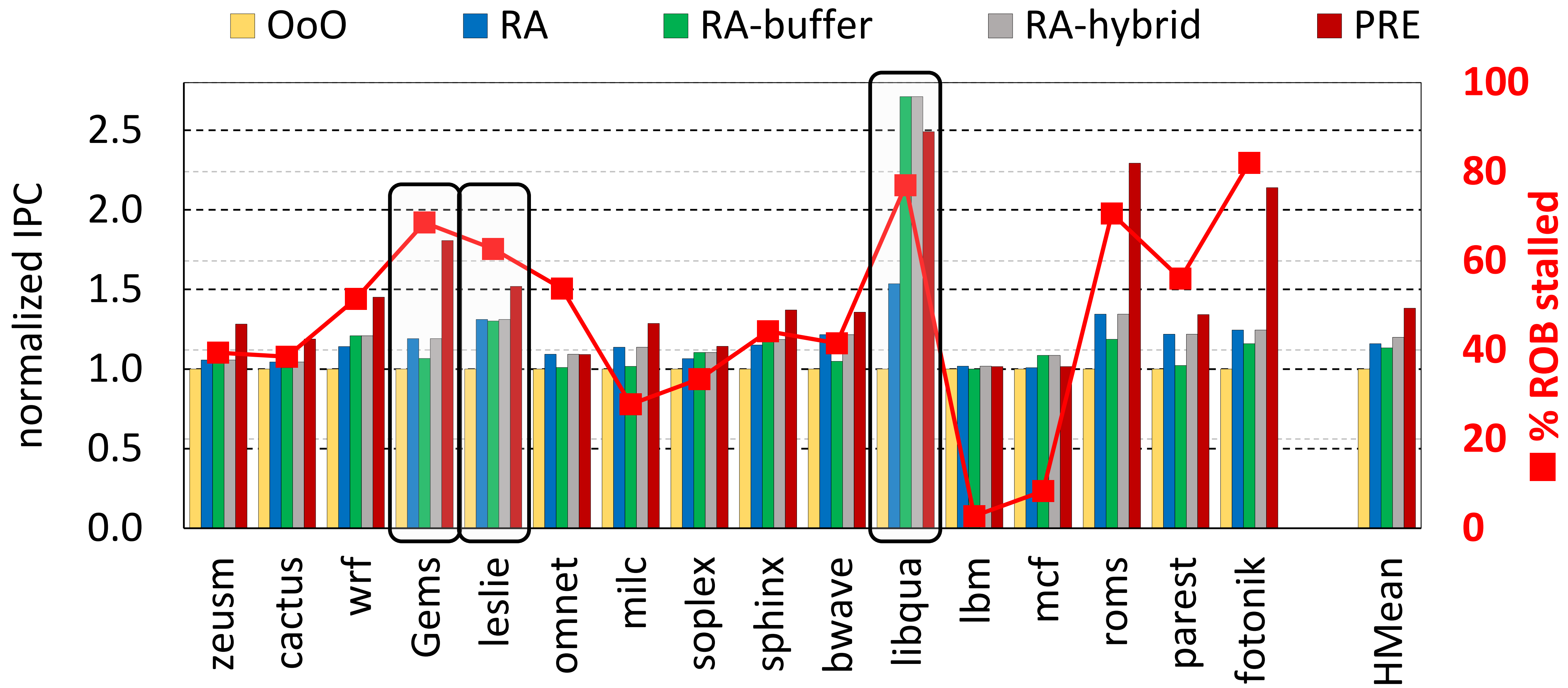
RA: 15.9% **RA-buffer: 13.3%** **RA-hybrid: 20%** **PRE: 38.2%**

Evaluation – Performance



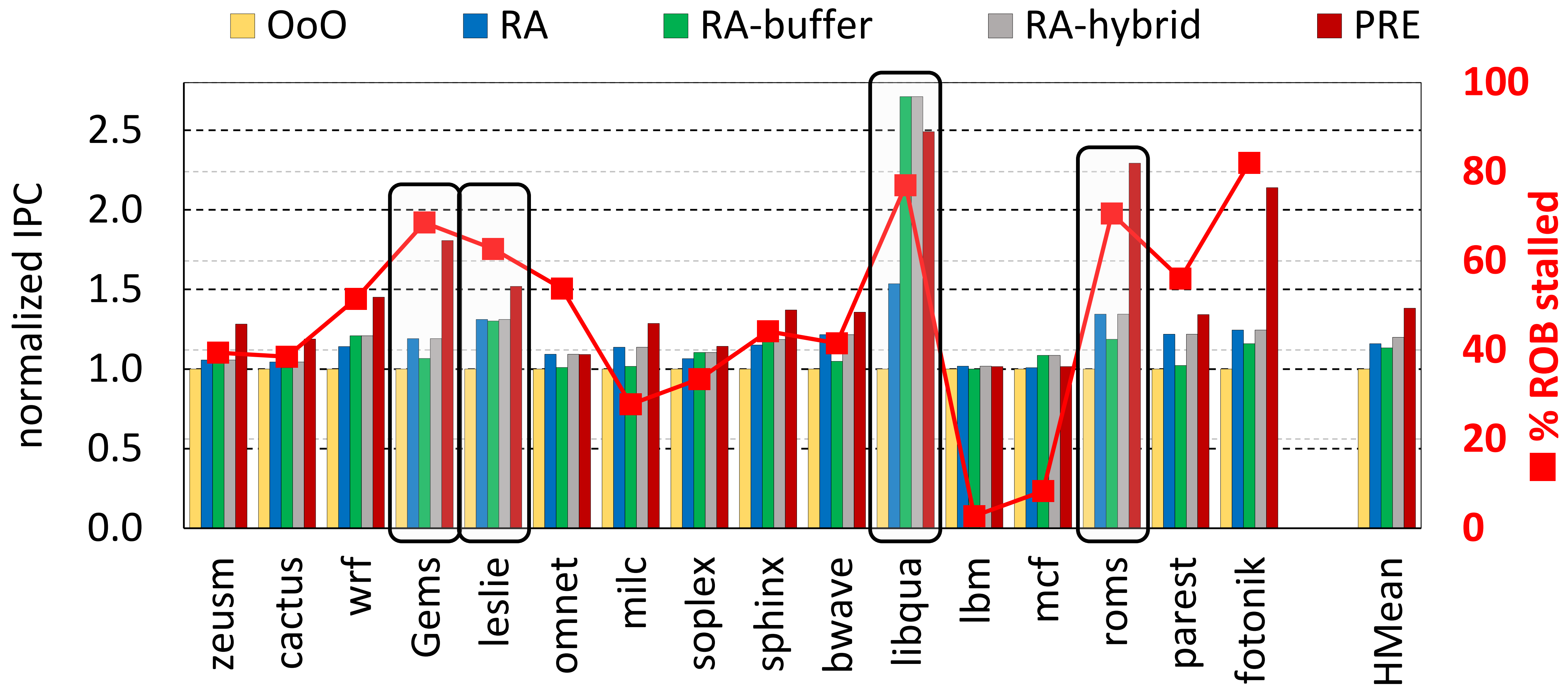
RA: 15.9% **RA-buffer: 13.3%** **RA-hybrid: 20%** **PRE: 38.2%**

Evaluation – Performance



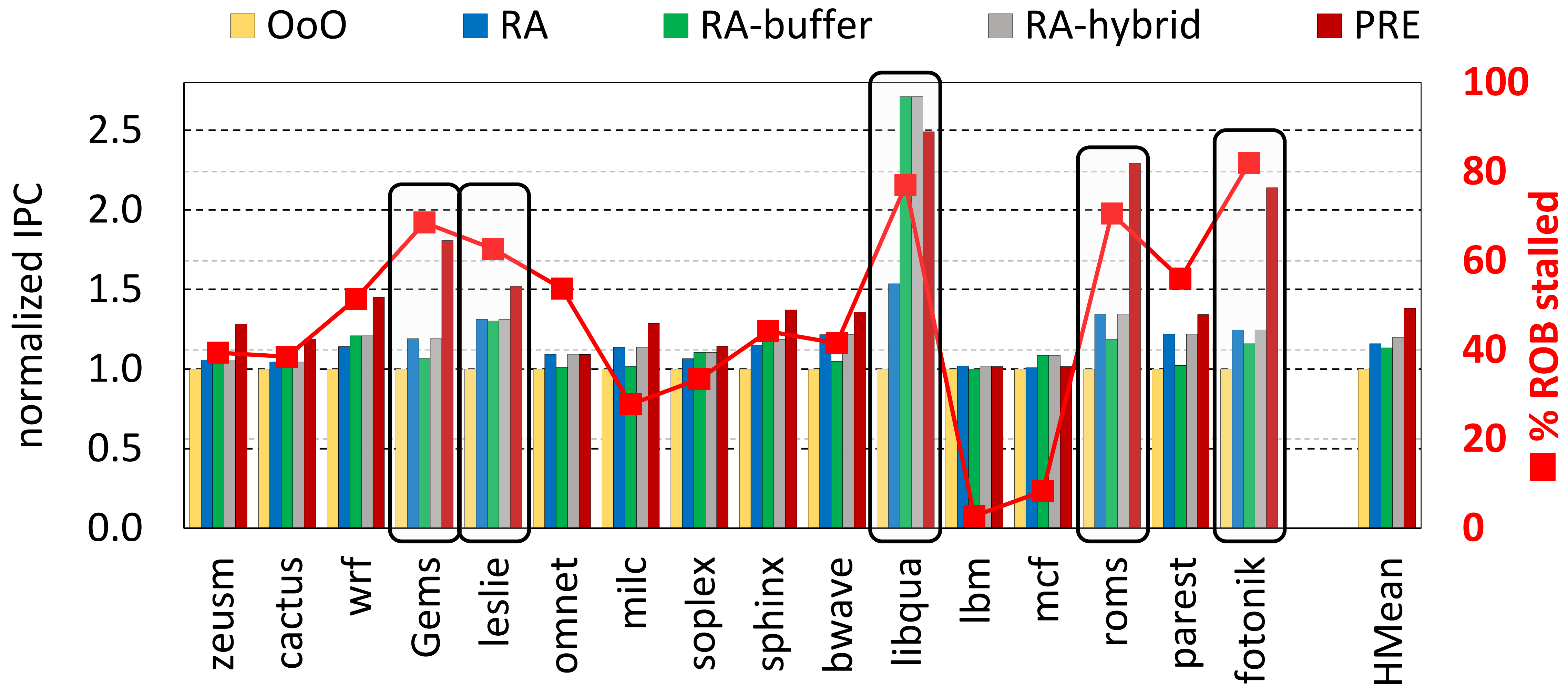
RA: 15.9% **RA-buffer: 13.3%** **RA-hybrid: 20%** **PRE: 38.2%**

Evaluation – Performance



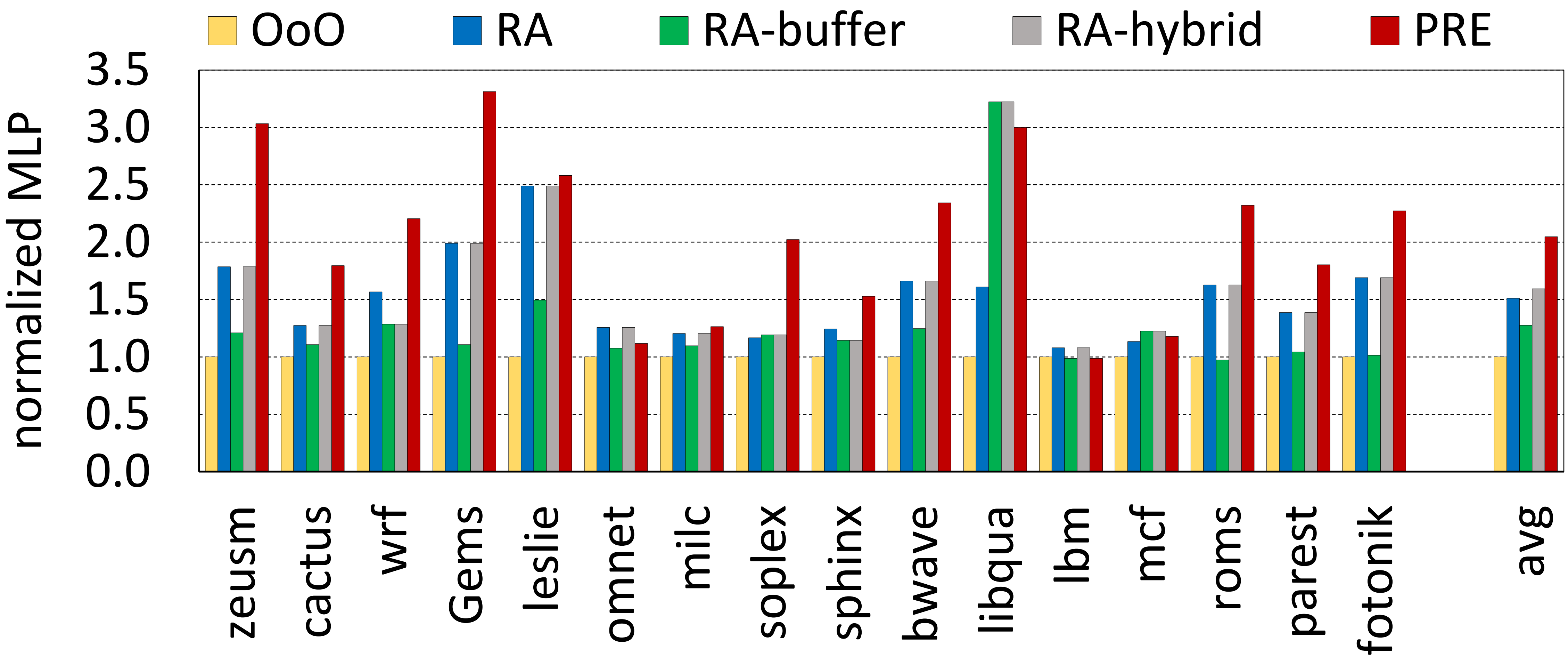
RA: 15.9% **RA-buffer: 13.3%** **RA-hybrid: 20%** **PRE: 38.2%**

Evaluation – Performance

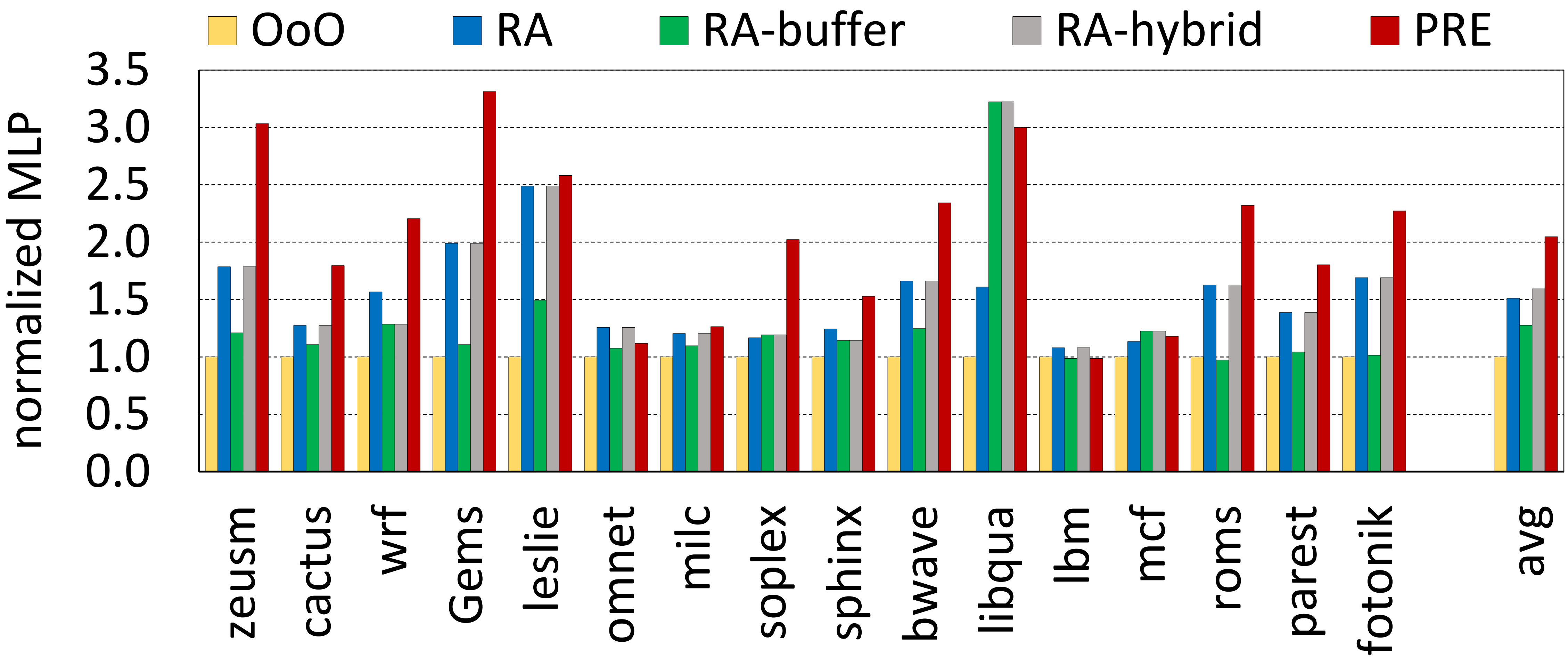


RA: 15.9% **RA-buffer: 13.3%** **RA-hybrid: 20%** **PRE: 38.2%**

Evaluation – Memory-Level Parallelism

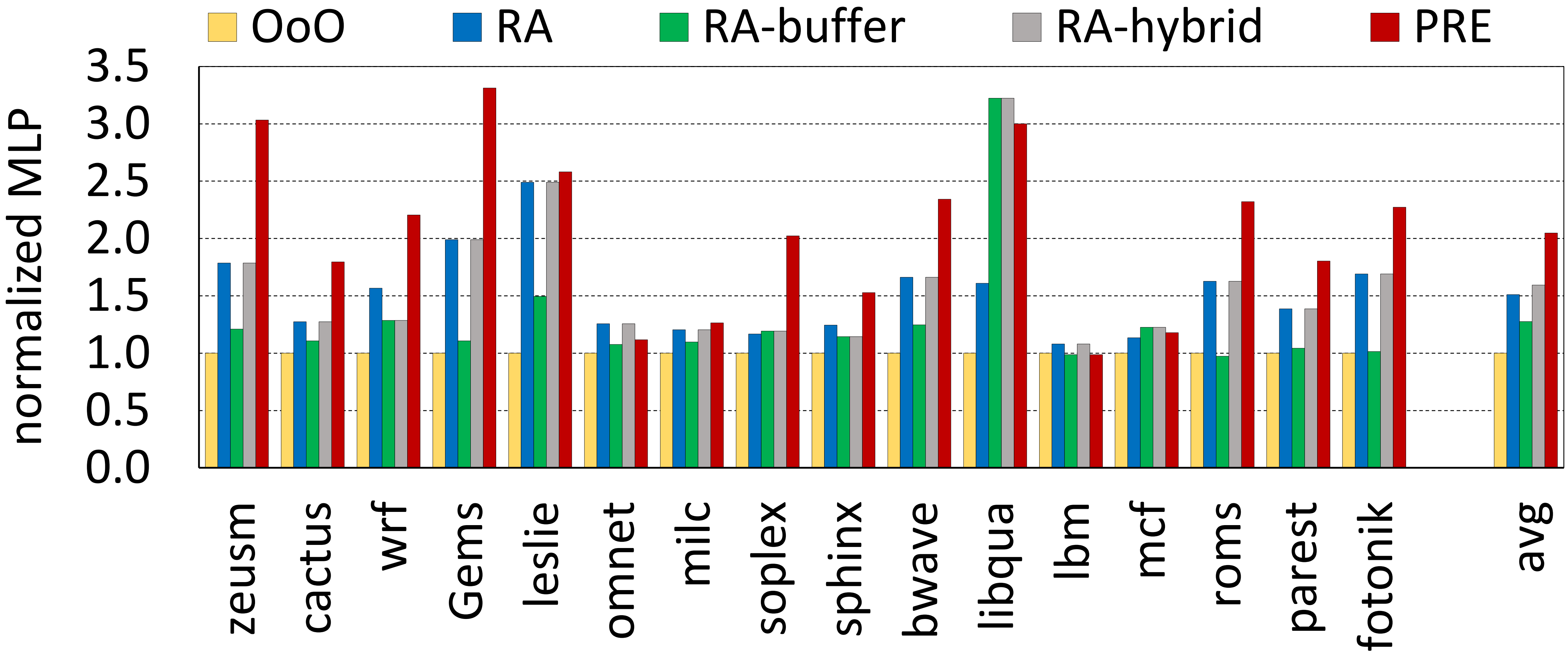


Evaluation – Memory-Level Parallelism



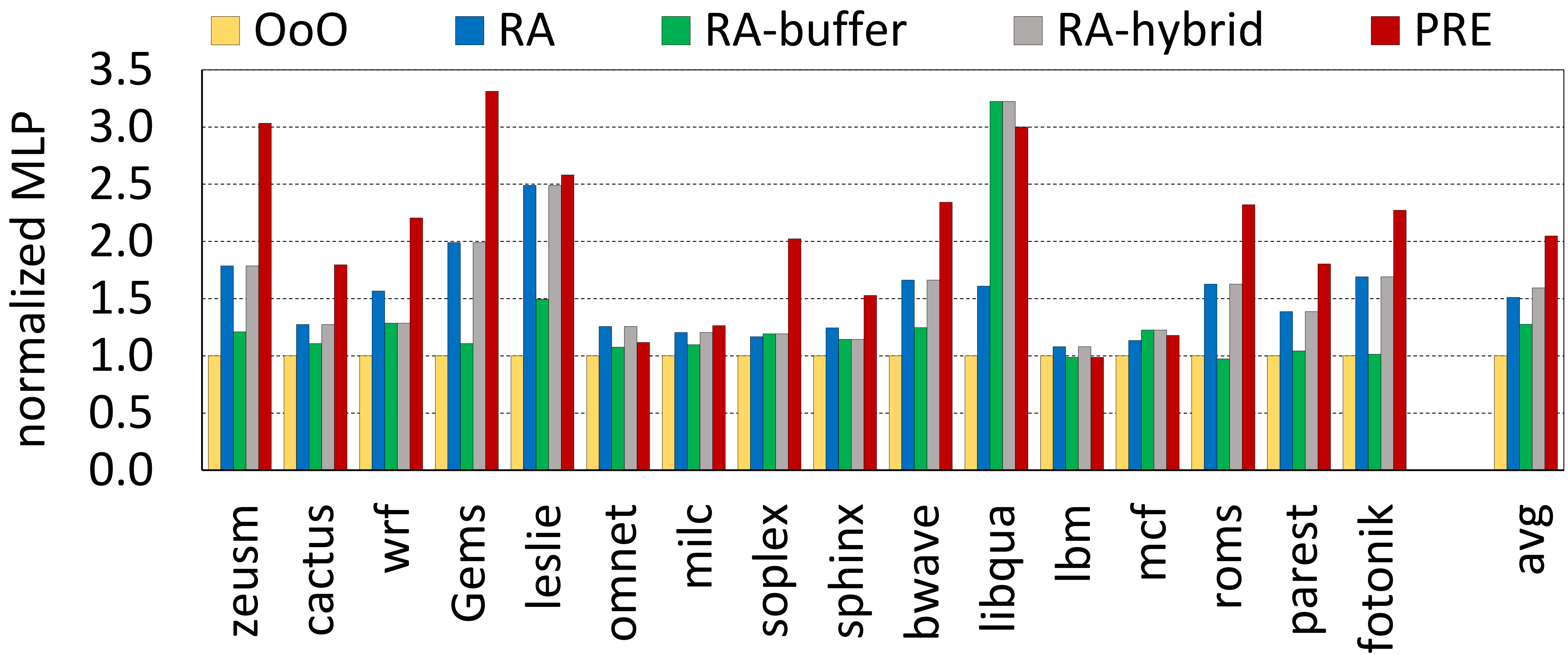
RA: 1.5X

Evaluation – Memory-Level Parallelism



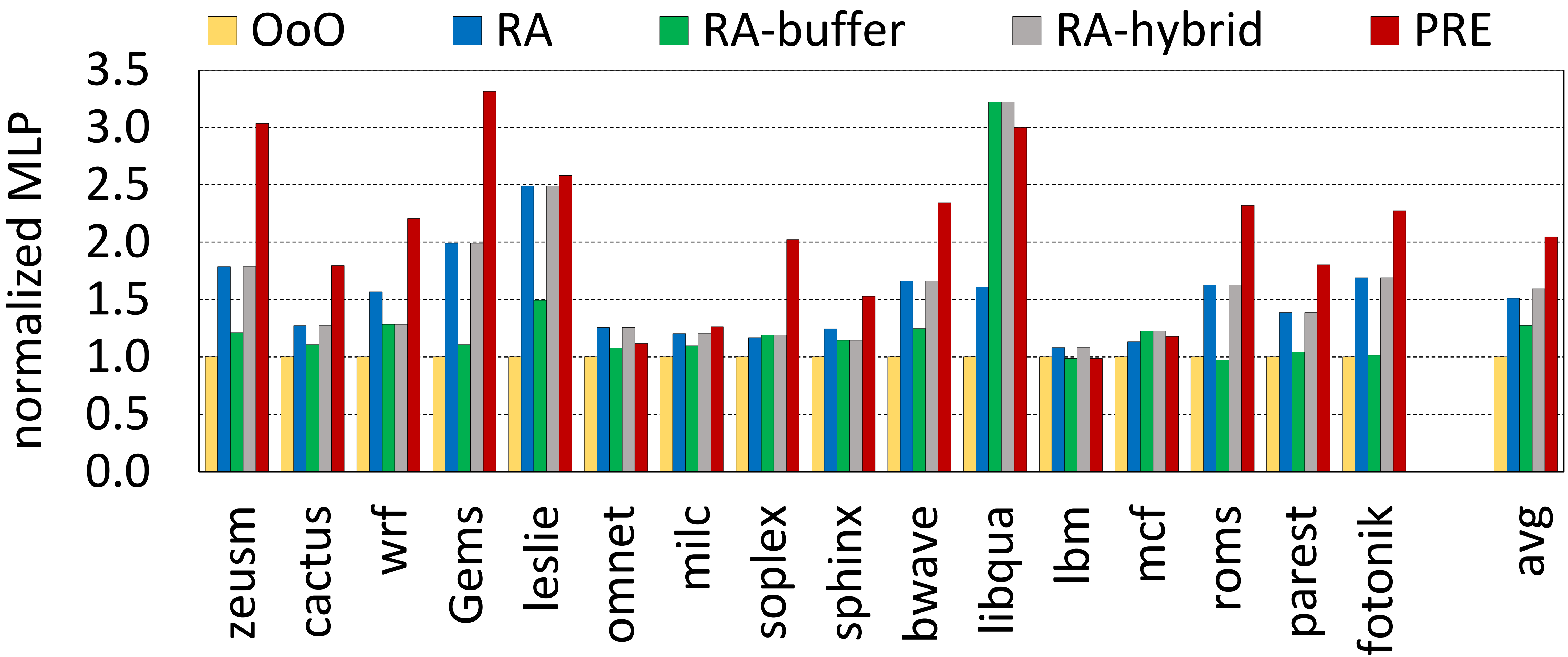
RA: 1.5X RA-buffer: 1.3X

Evaluation – Memory-Level Parallelism



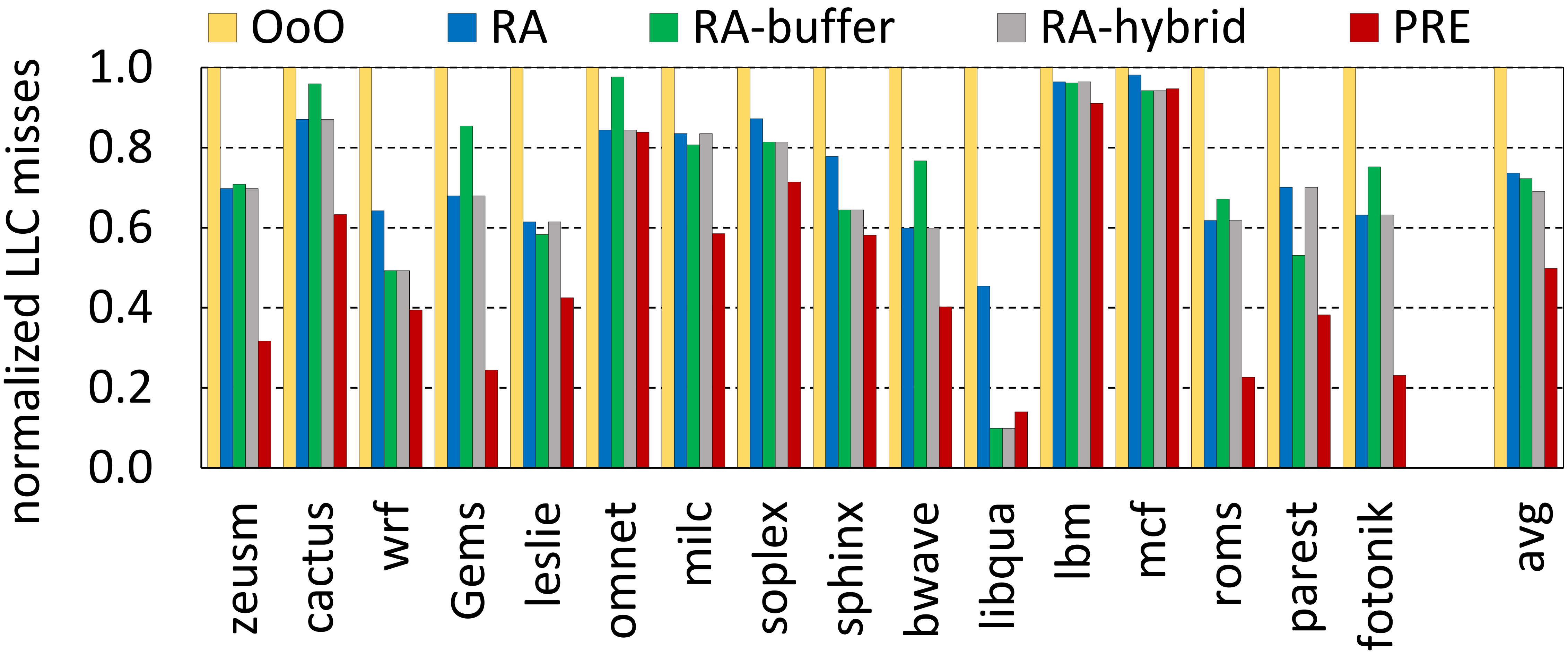
RA: 1.5X   RA-buffer: 1.3X   RA-hybrid: 1.6X

Evaluation – Memory-Level Parallelism

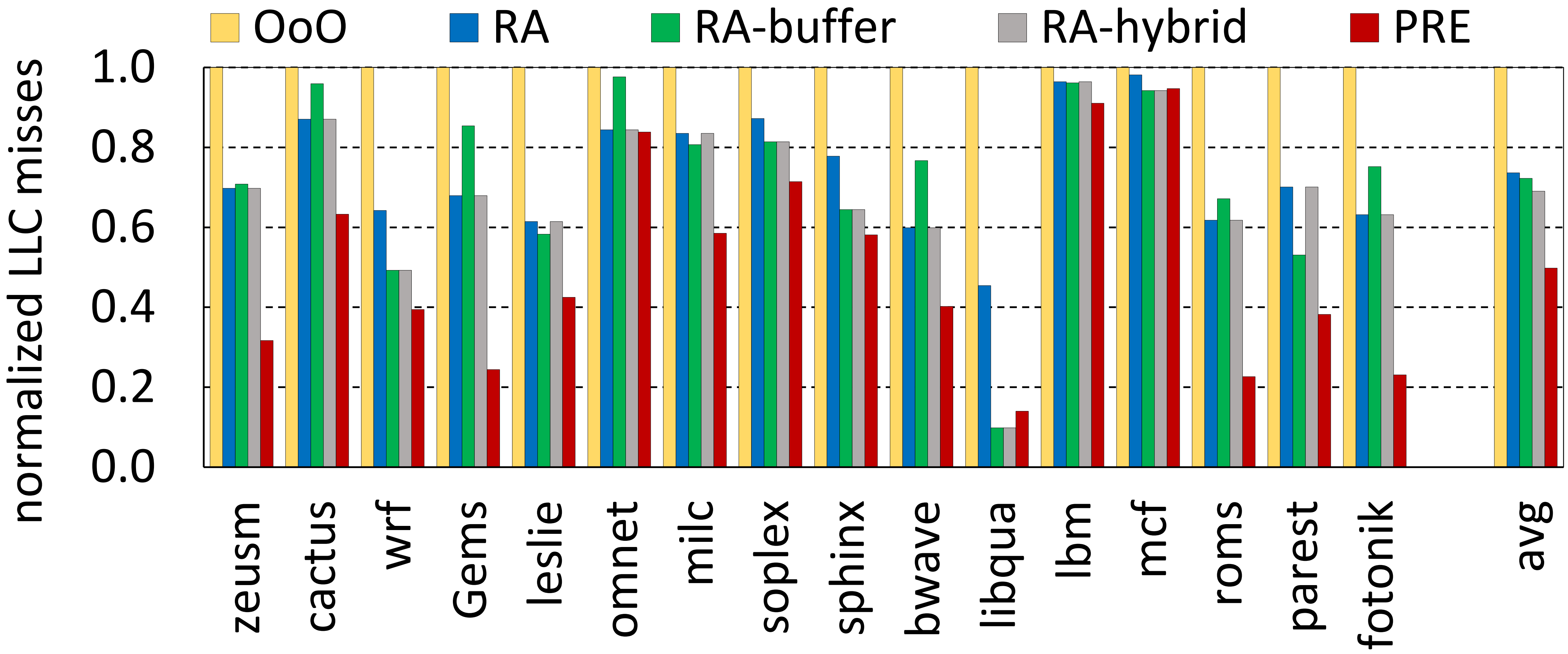


RA: 1.5X RA-buffer: 1.3X RA-hybrid: 1.6X PRE: 2.0X

Evaluation – LLC Miss Count Reduction

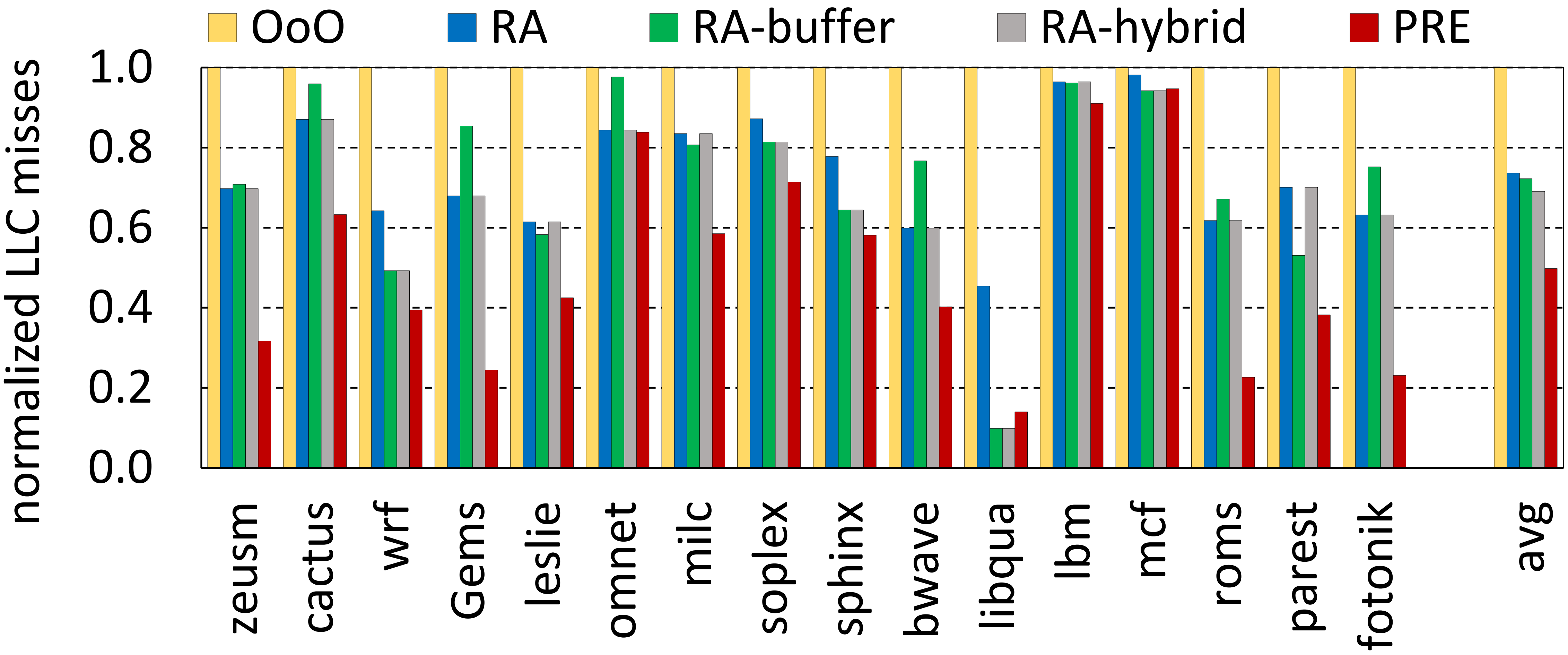


Evaluation – LLC Miss Count Reduction



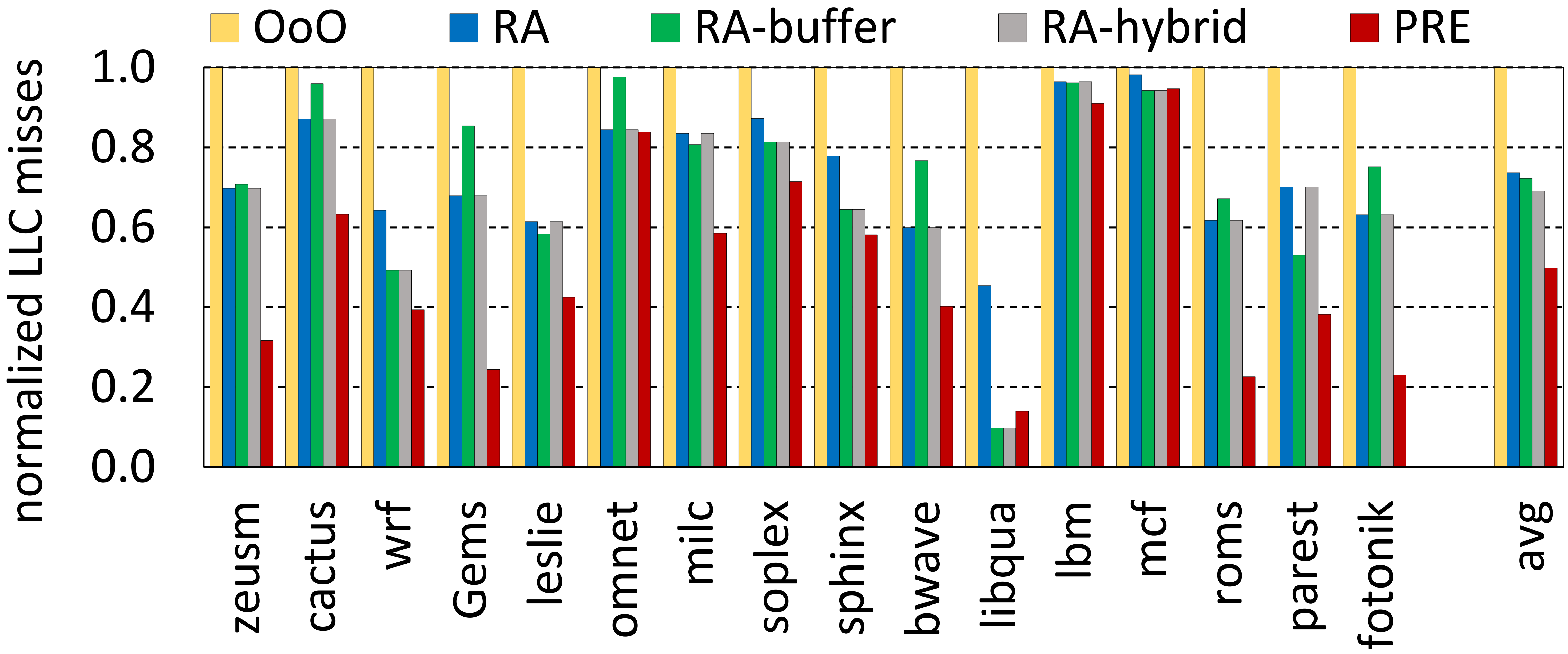
RA: 26.4%

Evaluation – LLC Miss Count Reduction



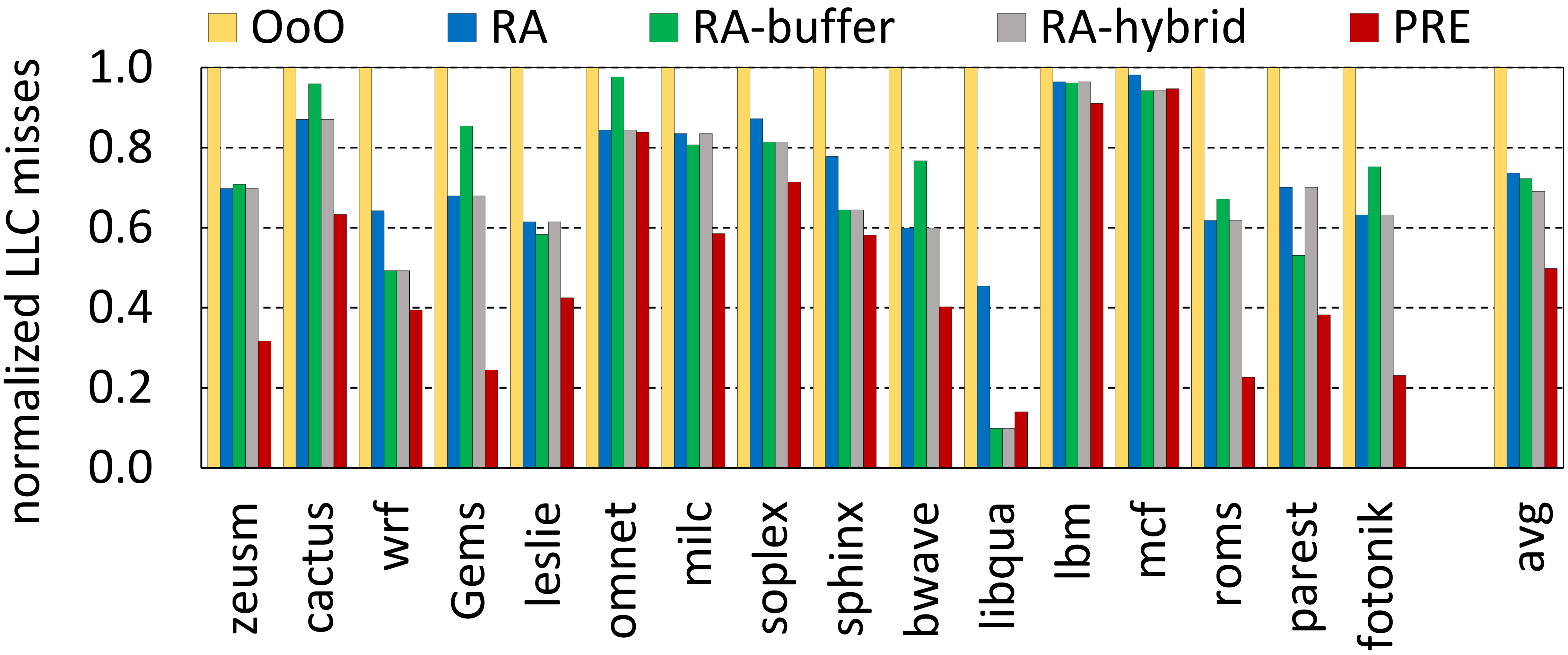
RA: 26.4%   RA-buffer: 27.7%

Evaluation – LLC Miss Count Reduction



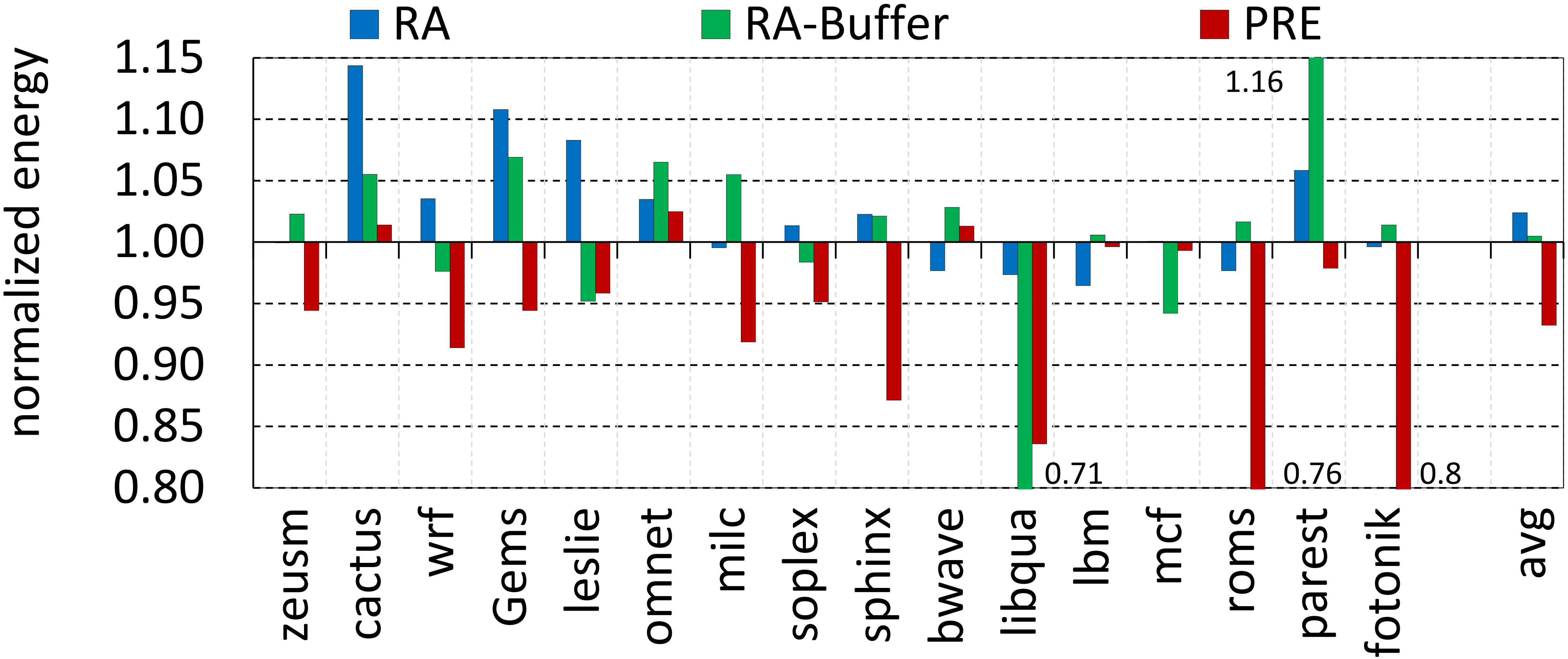
RA: 26.4%   RA-buffer: 27.7%   RA-hybrid: 31%

Evaluation – LLC Miss Count Reduction

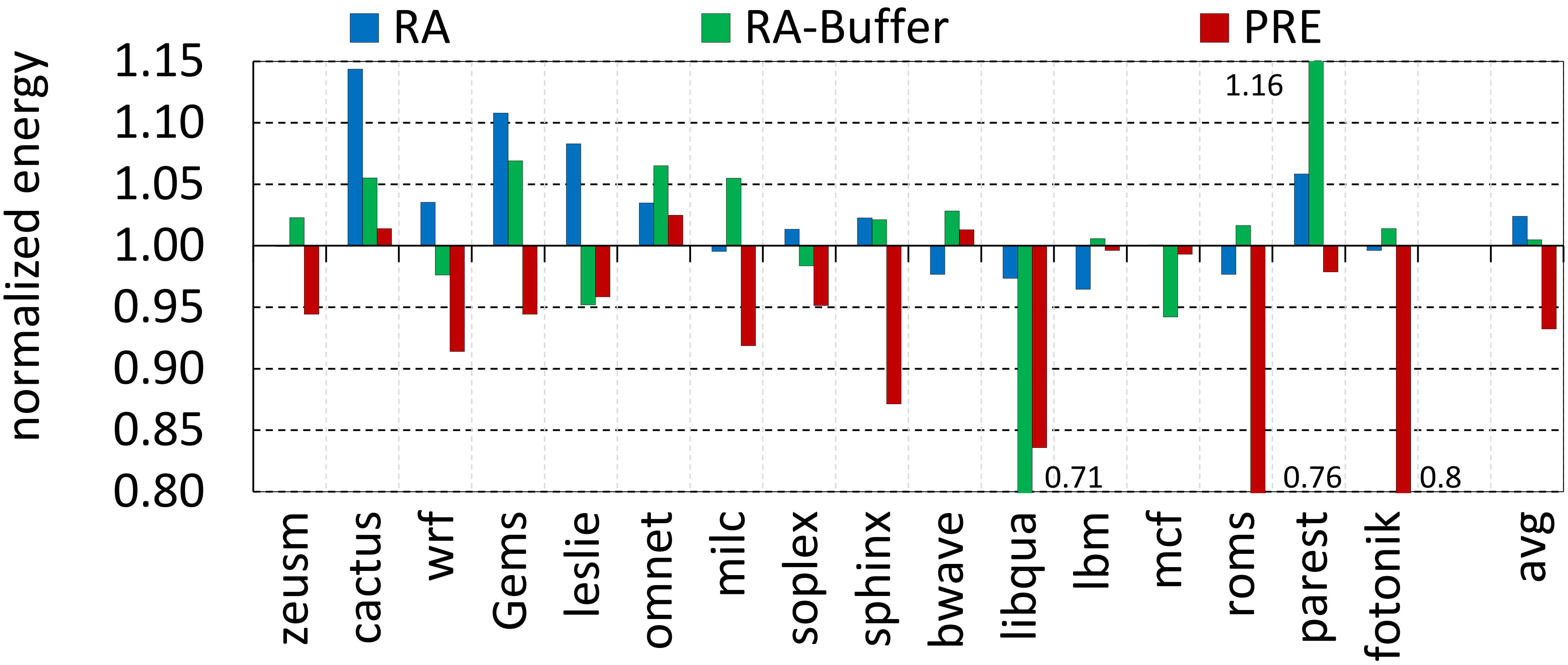


RA: 26.4%   RA-buffer: 27.7%   RA-hybrid: 31%   PRE: 50.2%

Evaluation – Energy

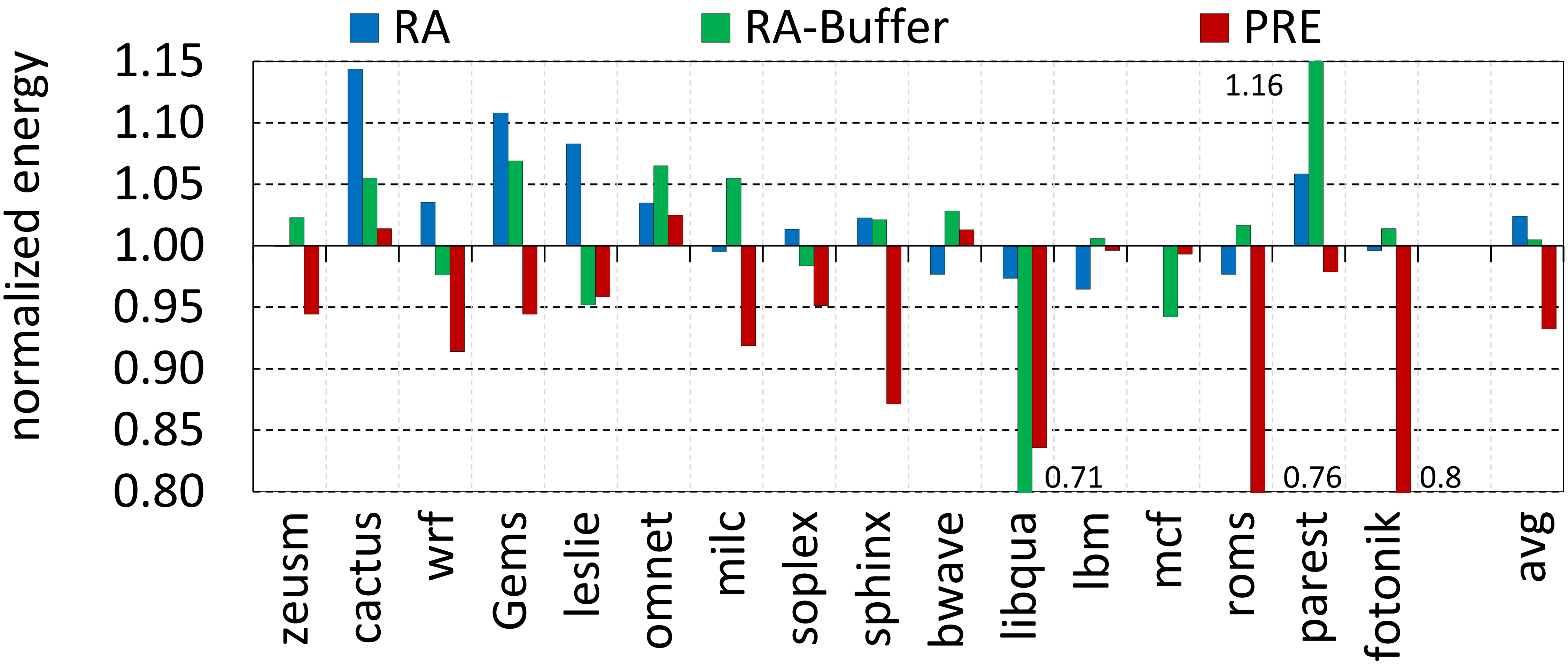


Evaluation – Energy



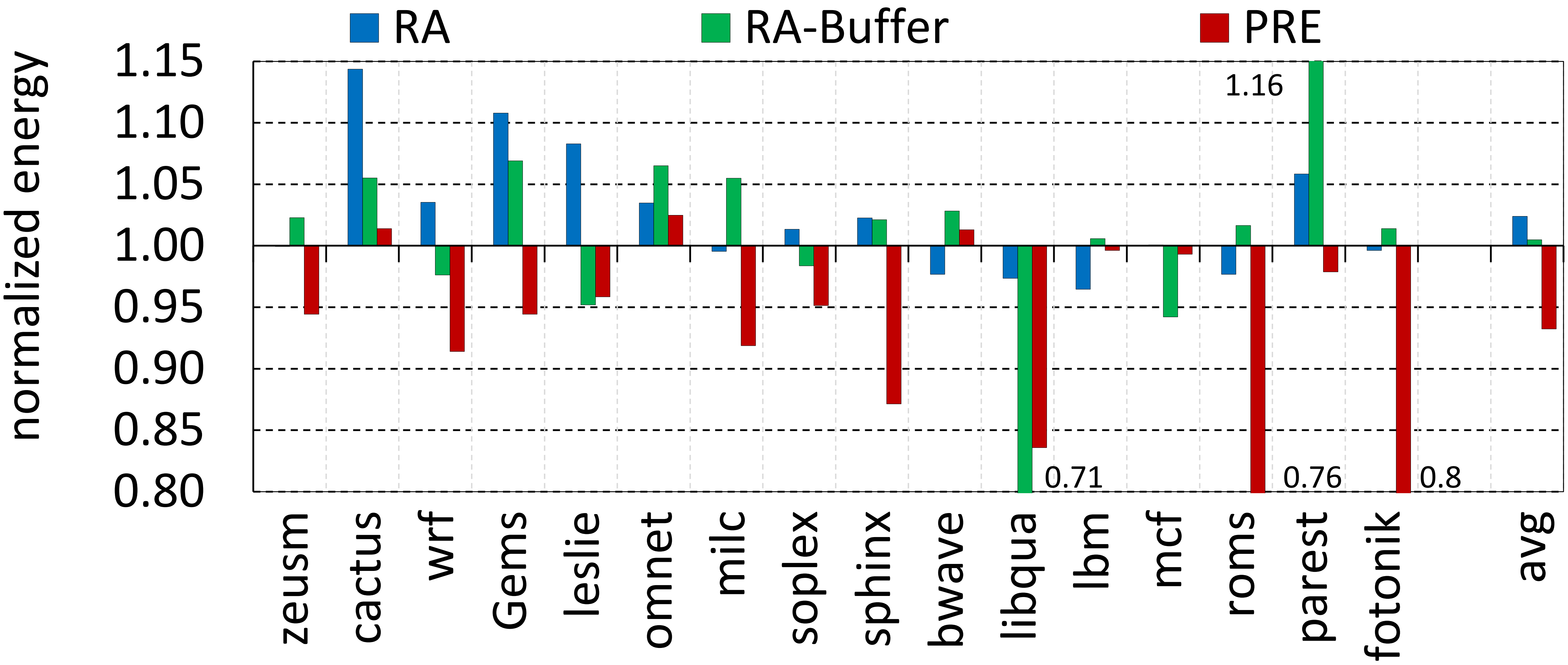
RA: +2.4%

Evaluation – Energy



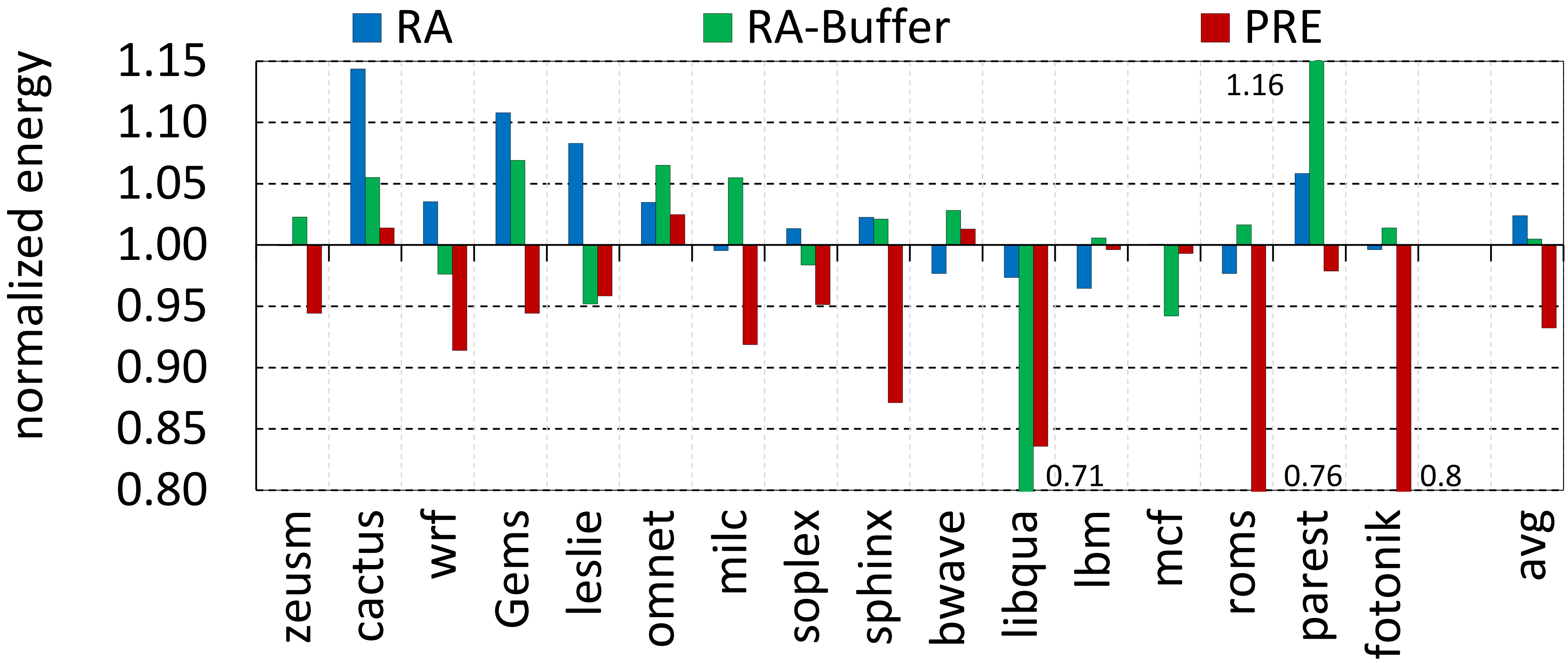
RA: +2.4%   RA-buffer: Same

Evaluation – Energy



RA: +2.4% RA-buffer: Same RA-hybrid: Same

Evaluation – Energy



RA: +2.4% **RA-buffer: Same** **RA-hybrid: Same** **PRE: -6.2%**

Conclusions

Conclusions

1. Never flushes the ROB

Conclusions

1. Never flushes the ROB

2. Executes only useful instructions in runahead mode

Conclusions

1. Never flushes the ROB

2. Executes only useful instructions in runahead mode

3. Efficiently manages microarchitectural resources

Conclusions

1. Never flushes the ROB

2. Executes only useful instructions in runahead mode

3. Efficiently manages microarchitectural resources

18.2% better performance

Conclusions

1. Never flushes the ROB

2. Executes only useful instructions in runahead mode

3. Efficiently manages microarchitectural resources

18.2% better performance **6.2% better energy**

Precise Runahead Execution

Ajeya Naithani
Josue Feliu
Almutaz Adileh
Lieven Eeckhout

GHENT UNIVERSITY

UNIVERSITAT POLITÈCNICA DE VALÈNCIA