Lessons Learned During the Development of the CapoOne Deterministic Multiprocessor Replay System

Pablo Montesinos, Matthew Hicks, Wonsun Ahn, Samuel T. King and Josep Torrellas

Department of Computer Science
University of Illinois at Urbana-Champaign

Motivation: Time Travel

Motivation: Time Travel

- Allows us to visit and recreate past states and events in computer

Motivation: Time Travel

- Allows us to visit and recreate past states and events in computer
- Wide range of uses:

Motivation: Time Travel

- Allows us to visit and recreate past states and events in computer
- Wide range of uses:
  - Debugging

Motivation: Time Travel

- Allows us to visit and recreate past states and events in computer
- Wide range of uses:
  - Debugging
  - Security

Motivation: Time Travel

- Allows us to visit and recreate past states and events in computer
- Wide range of uses:
  - Debugging
  - Security
  - High-availability

Motivation: Time Travel

- Allows us to visit and recreate past states and events in computer
- Wide range of uses:
  - Debugging
  - Security
  - High-availability
- Enabled by using Deterministic Replay of Execution

How Deterministic Replay Works

How Deterministic Replay Works

- Phase I: **Initial Execution** (a.k.a **Recording**)

How Deterministic Replay Works

- Phase I: **Initial Execution** (a.k.a **Recording**)

- Phase II: **Replay**

How Deterministic Replay Works

- Phase I: **Initial Execution** (a.k.a **Recording**)
  - Execute and record certain non-deterministic events into log

- Phase II: **Replay**

How Deterministic Replay Works

- Phase I: **Initial Execution** (a.k.a **Recording**)
  - Execute and record certain non-deterministic events into log
  - Sources of non-determinism: interrupts, memory access interleaving ...
- Phase II: **Replay**

How Deterministic Replay Works

- Phase I: **Initial Execution** (a.k.a **Recording**)
  - Execute and record certain non-deterministic events into log
  - Sources of non-determinism: interrupts, memory access interleaving ...
- Phase II: **Replay**
  - Restore to a previous checkpoint

How Deterministic Replay Works

- Phase I: **Initial Execution** (a.k.a **Recording**)
  - Execute and record certain non-deterministic events into log
  - Sources of non-determinism: interrupts, memory access interleaving ...
- Phase II: **Replay**
  - Restore to a previous checkpoint
  - Re-execute and use log to force software down the same execution path

SW-Based vs HW-Based Deterministic Replay

SW-Based vs HW-Based Deterministic Replay

SW Based Schemes

SW-Based vs HW-Based Deterministic Replay

SW Based Schemes

+ Flexible, integrate well with OS, apps

SW-Based vs HW-Based Deterministic Replay

SW Based Schemes

+ Flexible, integrate well with OS, apps

− Very slow on multiprocessors

SW-Based vs HW-Based Deterministic Replay

SW Based Schemes

vs

HW Based Schemes

+ Flexible, integrate well with OS, apps

− Very slow on multiprocessors

SW-Based vs HW-Based Deterministic Replay

SW Based Schemes

vs

HW Based Schemes

+ Flexible, integrate well with OS, apps

− Very slow on multiprocessors

+ Fast multiprocessor execution

SW-Based vs HW-Based Deterministic Replay

SW Based Schemes

vs

HW Based Schemes

- ➕ Flexible, integrate well with OS, apps
- ➖ Very slow on multiprocessors

- ➖ Poor integration with SW
- ➕ Fast multiprocessor execution

HW-Assisted Deterministic Replay

HW Assisted Schemes

+ Flexible, integrate well with OS, apps
+ Fast multiprocessor execution

HW-Assisted Deterministic Replay

CapoOne

- ✚ Flexible, integrate well with OS, apps
- ✚ Fast multiprocessor execution

Today's Agenda

Today's Agenda

- Overview: Capo and CapoOne

Today's Agenda

- Overview: Capo and CapoOne
- From full-system replay to sphere-based replay

Today's Agenda

- Overview: Capo and CapoOne
- From full-system replay to sphere-based replay
- Exiting the replay sphere

Today's Agenda

- Overview: Capo and CapoOne
- From full-system replay to sphere-based replay
- Exiting the replay sphere
- System Issues

Capo (ASPLOS 09)

Capo (ASPLOS 09)

- **SW-HW interface** for HW-Assisted deterministic replay

Capo (ASPLOS 09)

- **SW-HW interface** for HW-Assisted deterministic replay
  - Integrates HW-Based replay systems with O.S. and applications

Capo (ASPLOS 09)

- **SW-HW interface** for HW-Assisted deterministic replay
  - Integrates HW-Based replay systems with O.S. and applications
  - Narrow: compatible with any HW-Based replay system

Capo (ASPLOS 09)

- **SW-HW interface** for HW-Assisted deterministic replay
  - Integrates HW-Based replay systems with O.S. and applications
  - Narrow: compatible with any HW-Based replay system

- **Replay Sphere:** new abstraction

Capo (ASPLOS 09)

- **SW-HW interface** for HW-Assisted deterministic replay
  - Integrates HW-Based replay systems with O.S. and applications
  - Narrow: compatible with any HW-Based replay system

- **Replay Sphere:** new abstraction
  - Isolates SW that is being recorded (replayed) from the rest

Capo (ASPLOS 09)

- **SW-HW interface** for HW-Assisted deterministic replay
  - Integrates HW-Based replay systems with O.S. and applications
  - Narrow: compatible with any HW-Based replay system

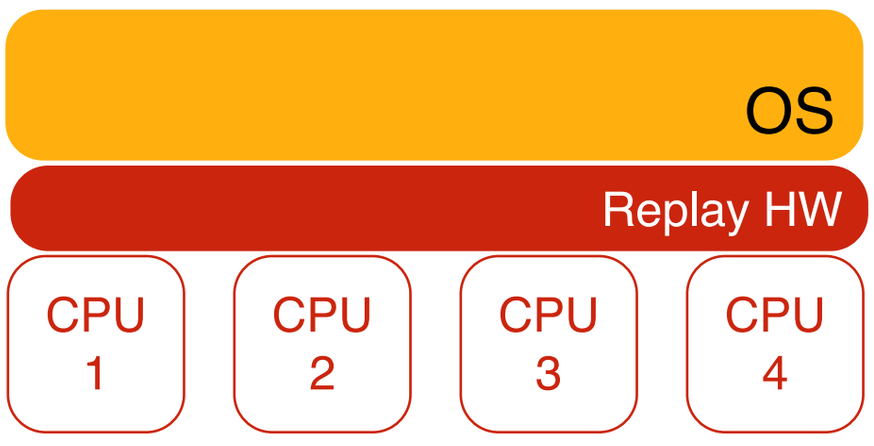
- **Replay Sphere:** new abstraction
  - Isolates SW that is being recorded (replayed) from the rest
  - Separates the responsibilities of the HW and the SW components

Capo (ASPLOS 09)

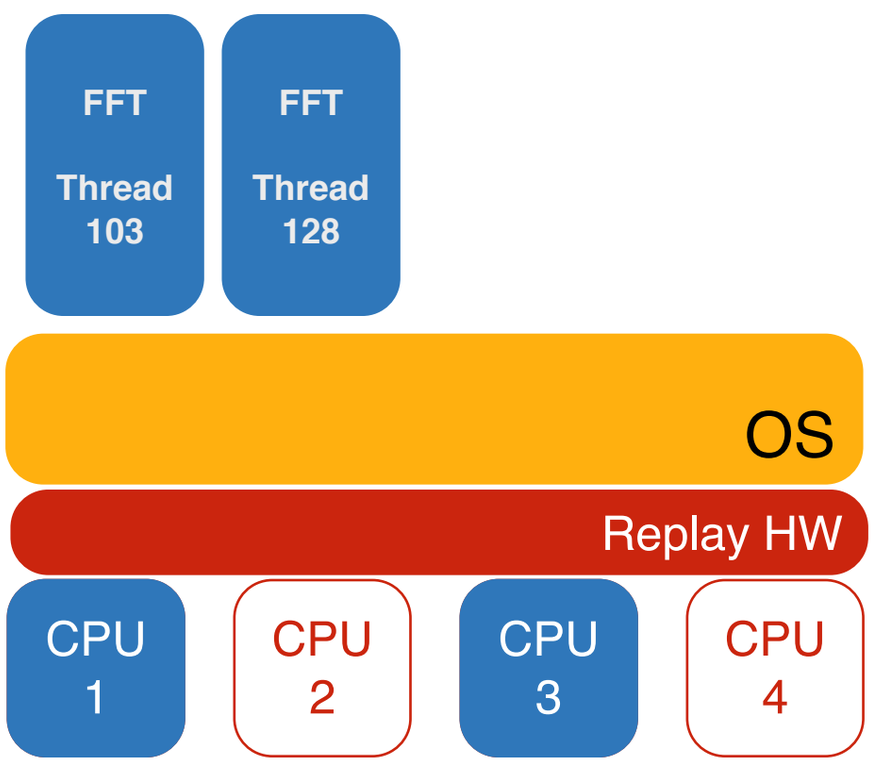
- **SW-HW interface** for HW-Assisted deterministic replay
  - Integrates HW-Based replay systems with O.S. and applications
  - Narrow: compatible with any HW-Based replay system
- **Replay Sphere:** new abstraction
  - Isolates SW that is being recorded (replayed) from the rest
  - Separates the responsibilities of the HW and the SW components
- **CapoOne:** first implementation of Capo

Replay Sphere

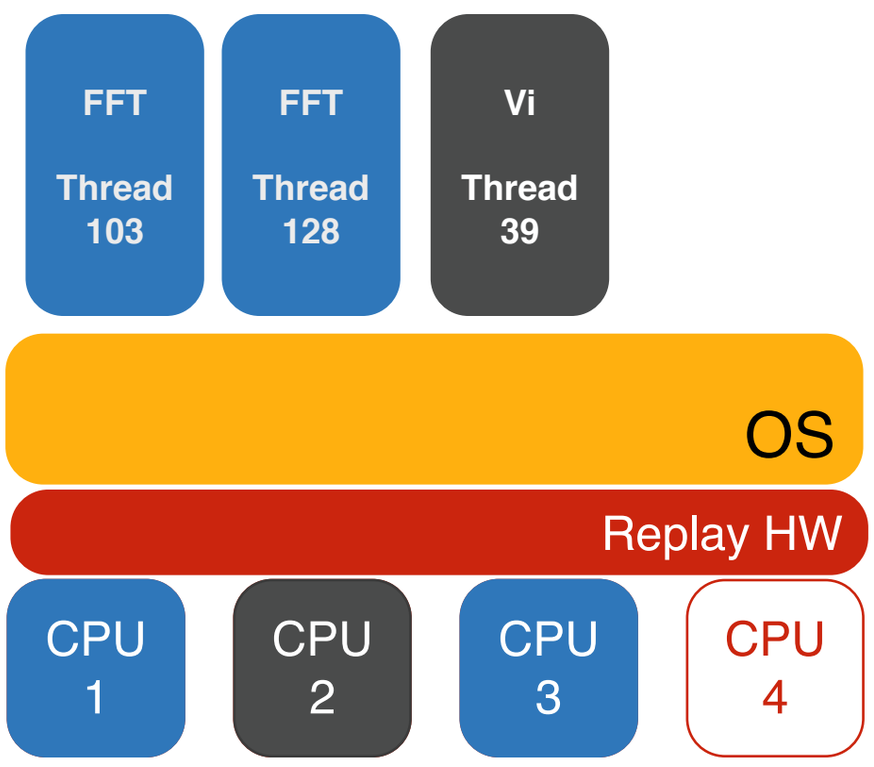
Replay Sphere



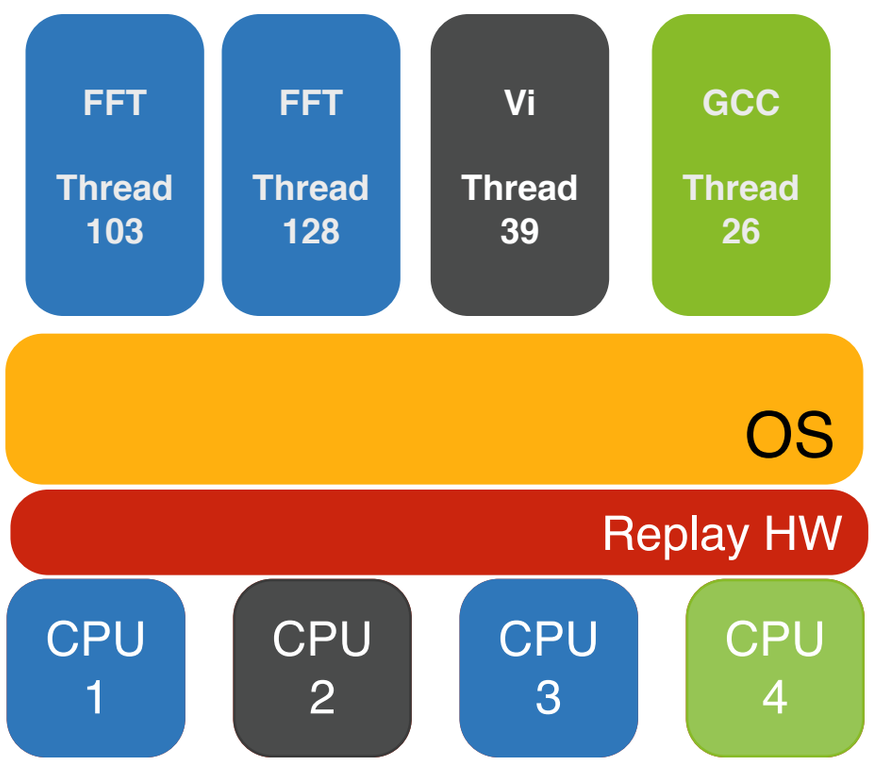
Replay Sphere



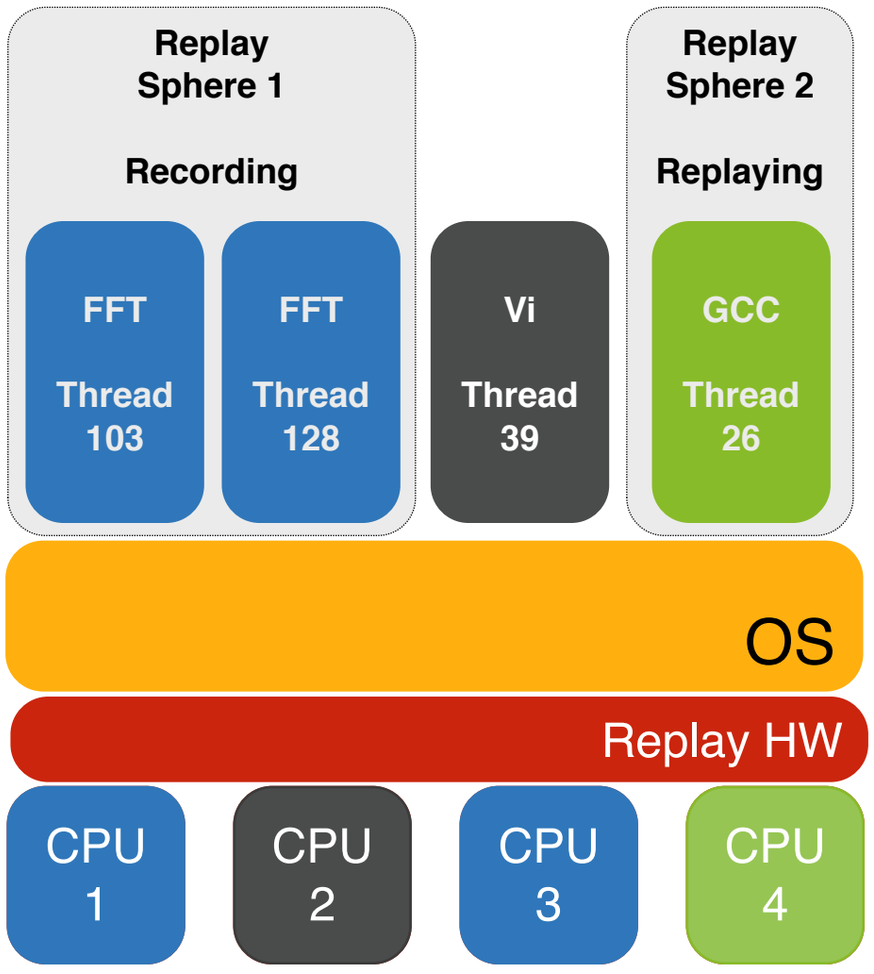
Replay Sphere



Replay Sphere

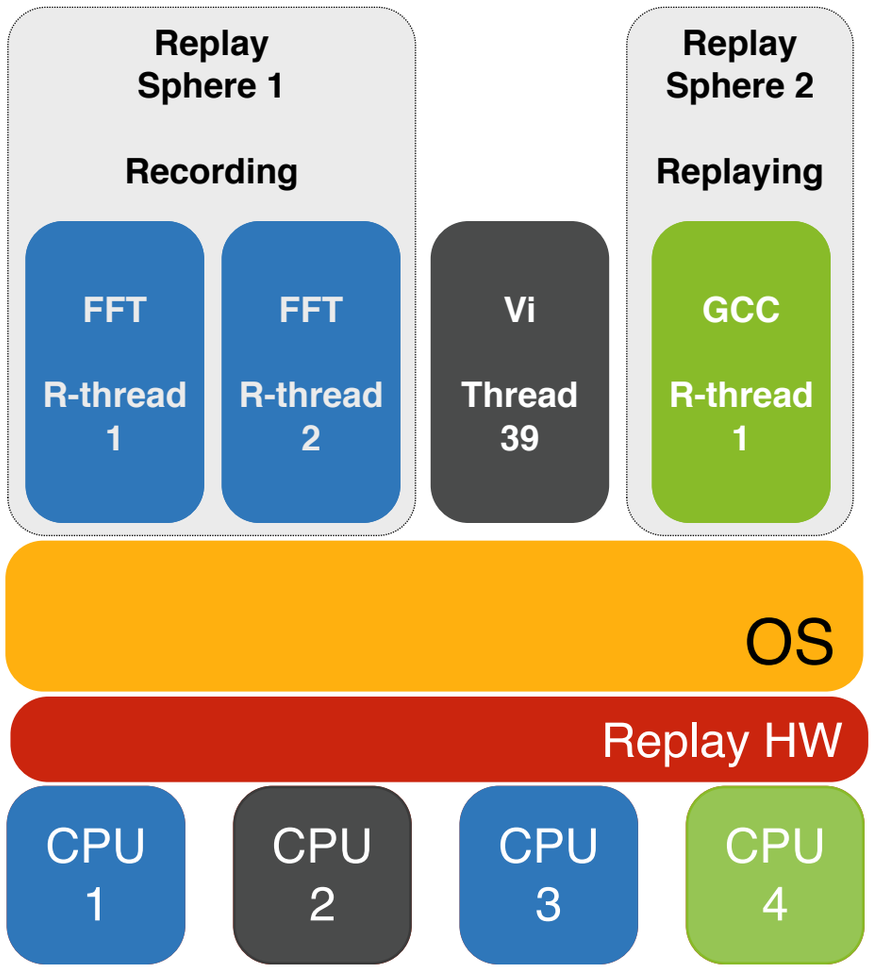


Replay Sphere



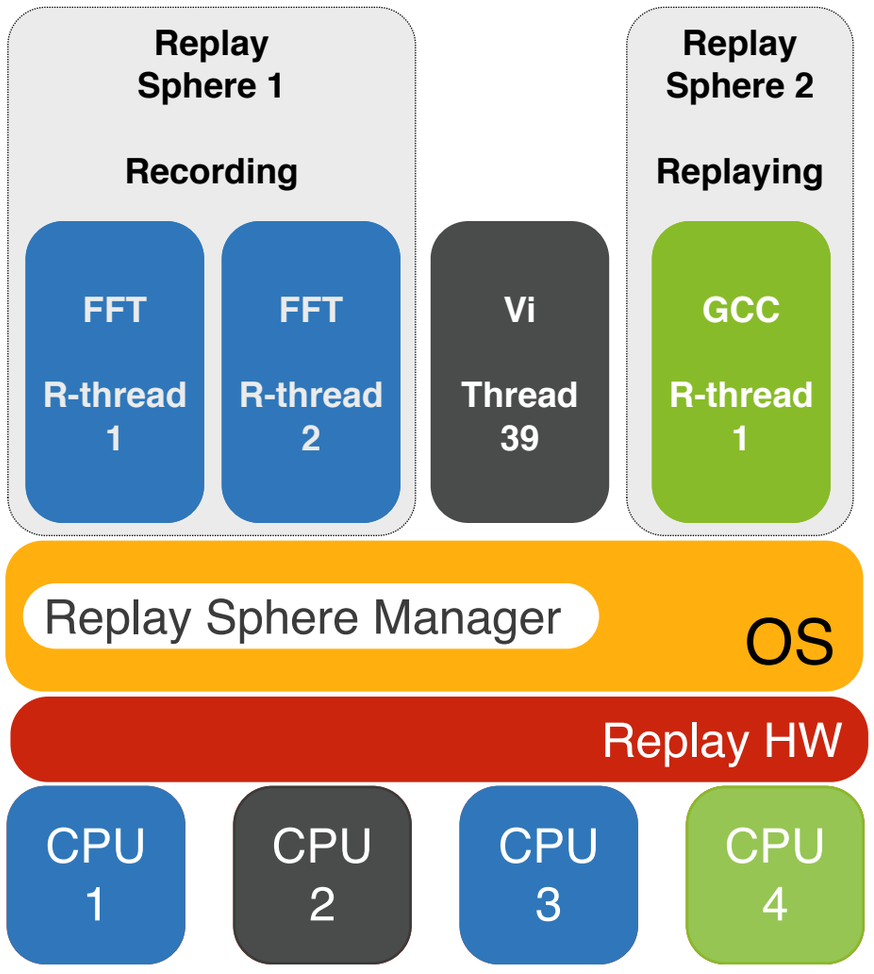
- Set of threads recorded and replayed as a unit and their address space

Replay Sphere



- Set of threads recorded and replayed as a unit and their address space
  - Only user-mode threads run inside spheres: R-threads

Replay Sphere



- Set of threads recorded and replayed as a unit and their address space
  - Only user-mode threads run inside spheres: R-threads

Replay Sphere: Separating Responsibilities

Replay Sphere: Separating Responsibilities

- Replay HW:
  - Records memory access interleaving of R-threads running within same sphere
  - Produces per-sphere **Memory Interleaving Log**
  - Enforces same memory interleaving during replay

Replay Sphere: Separating Responsibilities

- Replay HW:
  - Records memory access interleaving of R-threads running within same sphere
  - Produces per-sphere **Memory Interleaving Log**
  - Enforces same memory interleaving during replay

- Replay Sphere Manager:
  - Logs the other sources of non-determinism that affect the sphere
  - Produces per-sphere **Input Log**
    - Includes system call return values, signals, data copied into the sphere...
  - Injects data from log into sphere during replay

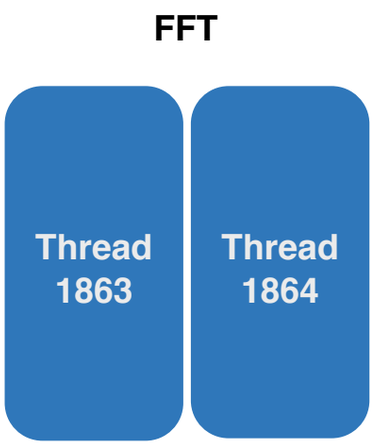
CapoOne: First Capo Implementation

CapoOne: First Capo Implementation

- Records and replays multithreaded Linux apps running on multiprocessors

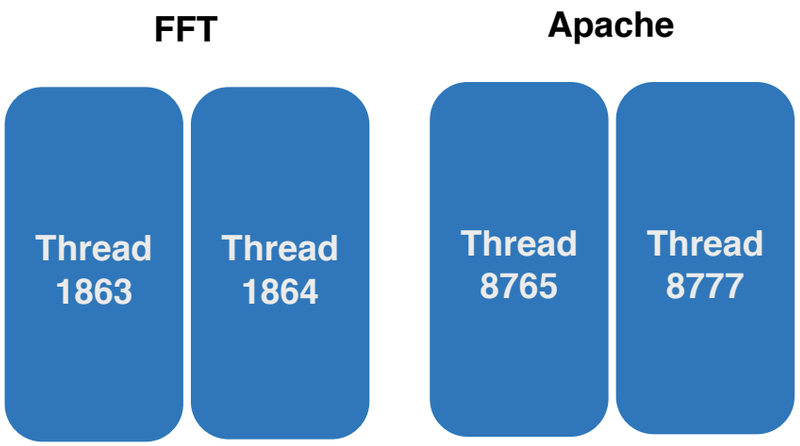
CapoOne: First Capo Implementation

- Records and replays multithreaded Linux apps running on multiprocessors



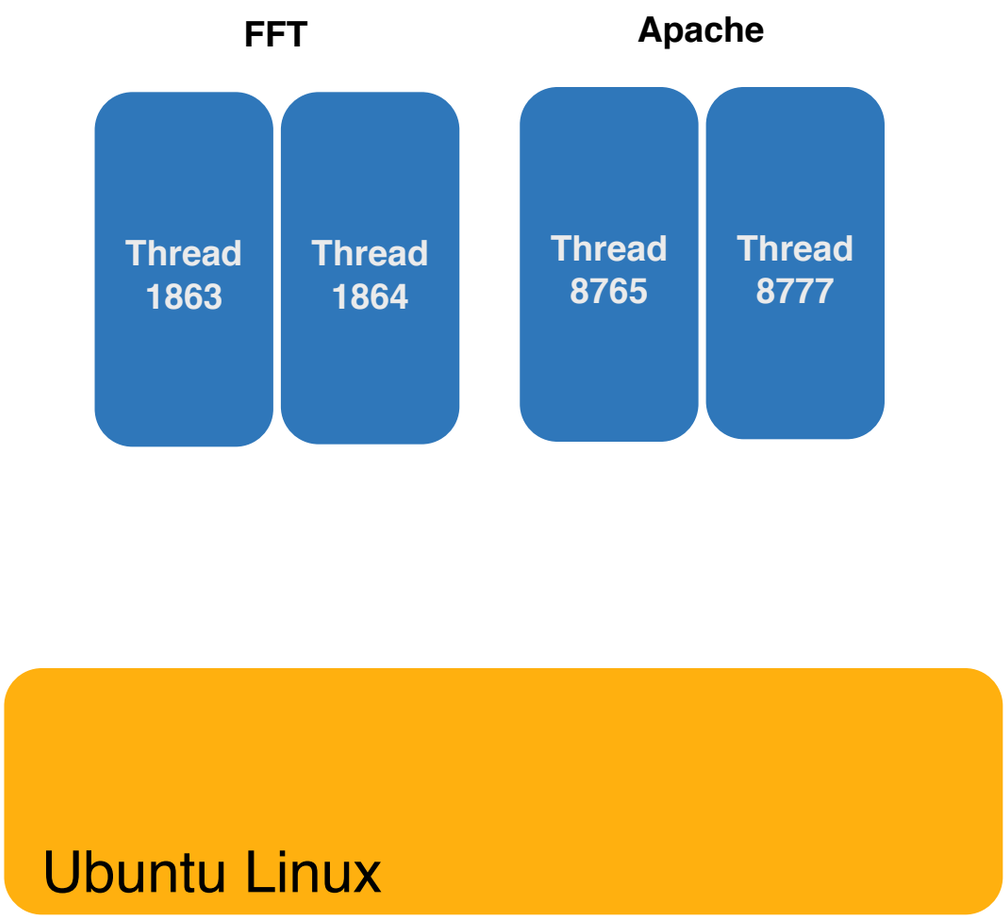
CapoOne: First Capo Implementation

- Records and replays multithreaded Linux apps running on multiprocessors



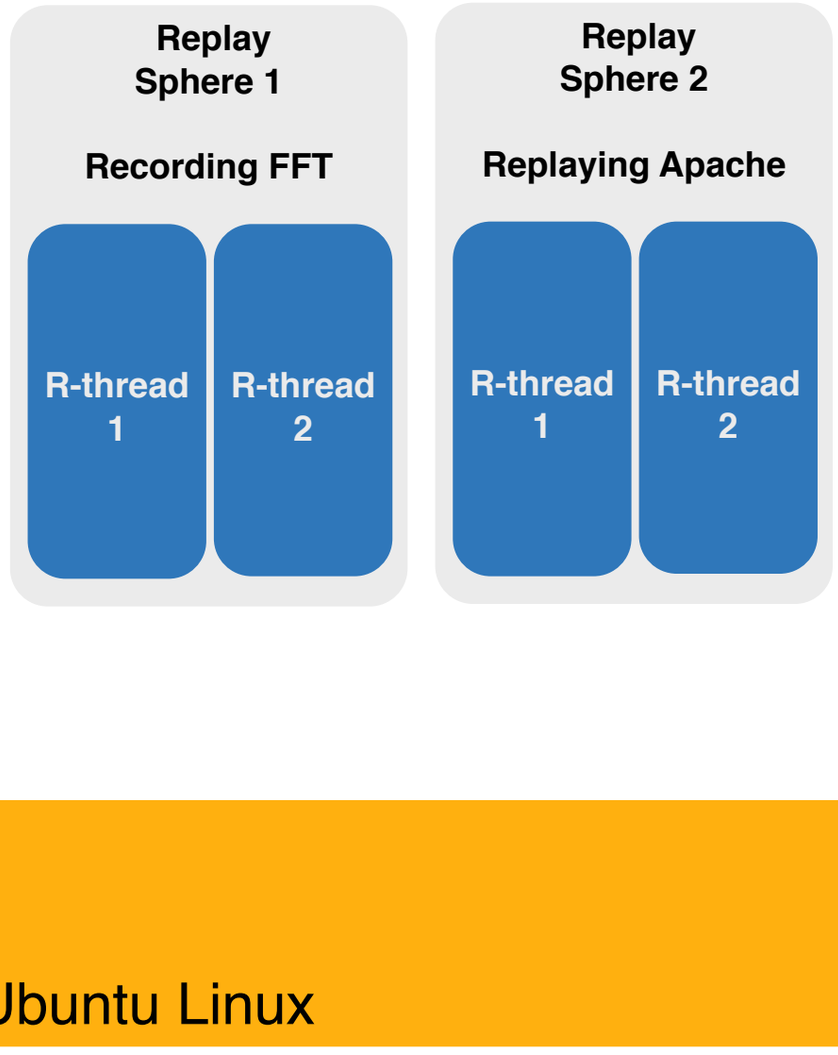
CapoOne: First Capo Implementation

- Records and replays multithreaded Linux apps running on multiprocessors
- Ubuntu Linux with modified kernel:



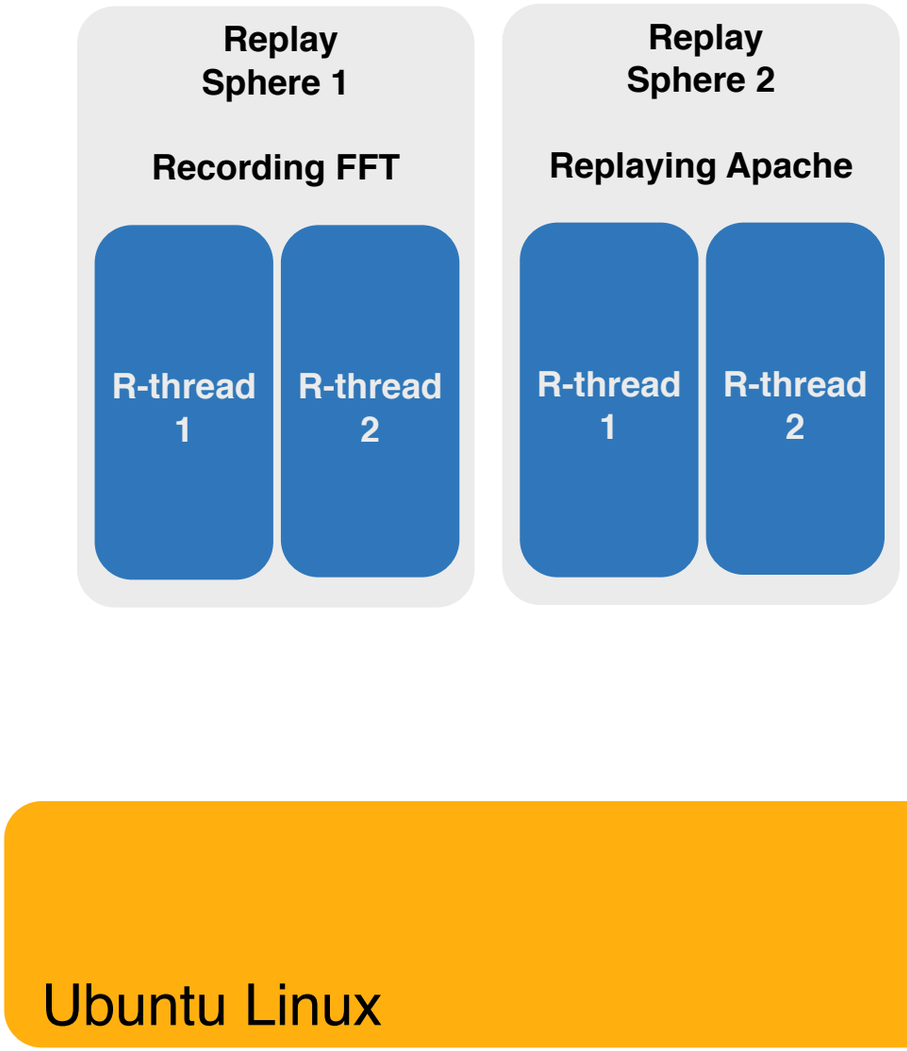
CapoOne: First Capo Implementation

- Records and replays multithreaded Linux apps running on multiprocessors
- Ubuntu Linux with modified kernel:
  - Added support for spheres, R-threads



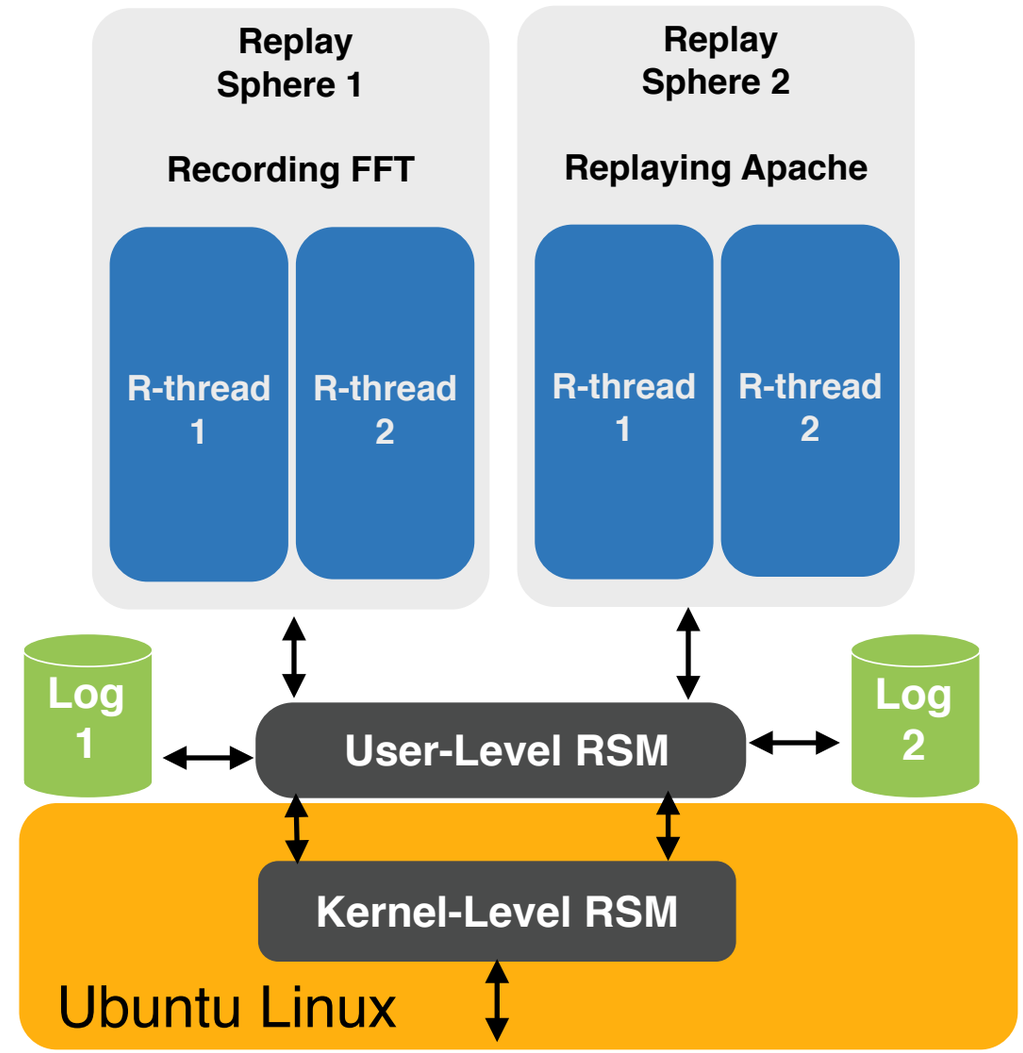
CapoOne: First Capo Implementation

- Records and replays multithreaded Linux apps running on multiprocessors
- Ubuntu Linux with modified kernel:
  - Added support for spheres, R-threads
  - Made some functions more deterministic



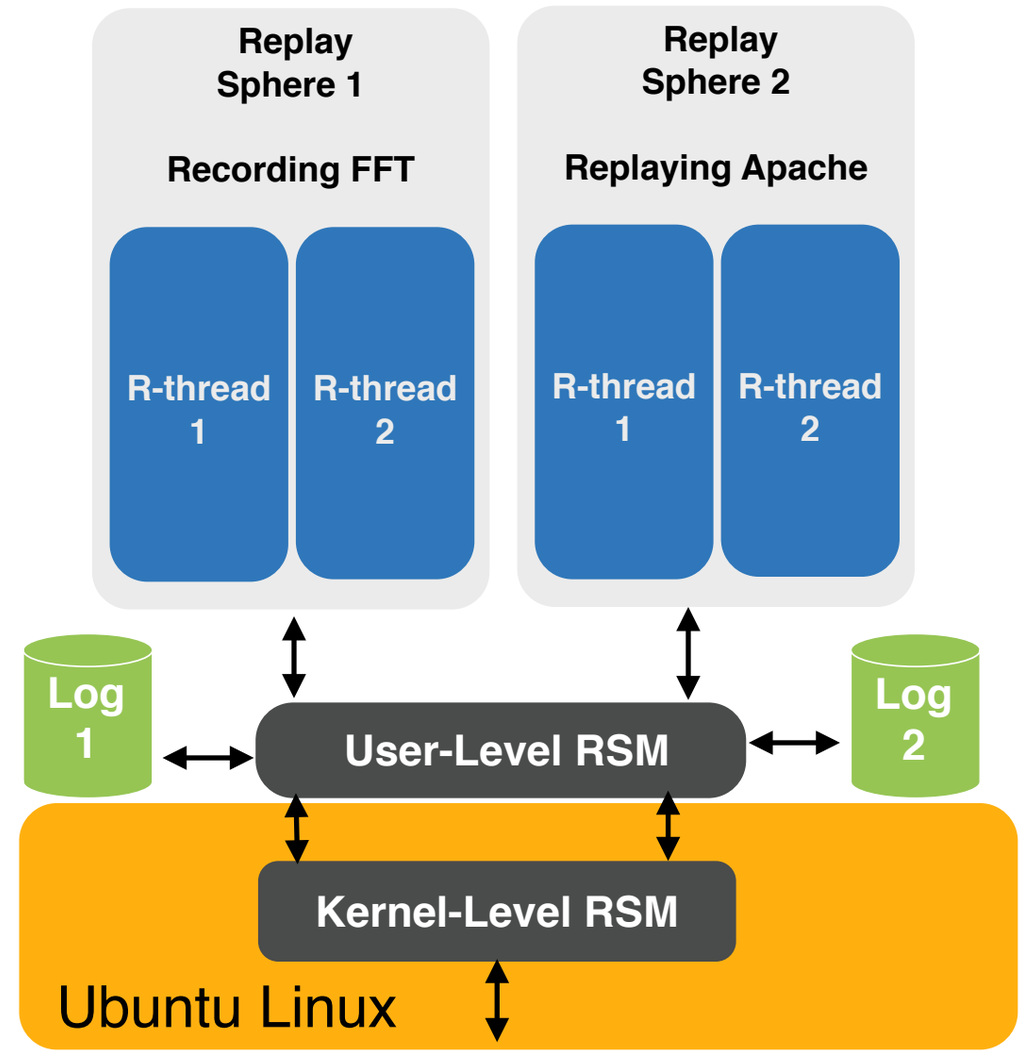
CapoOne: First Capo Implementation

- Records and replays multithreaded Linux apps running on multiprocessors
- Ubuntu Linux with modified kernel:
  - Added support for spheres, R-threads
  - Made some functions more deterministic
- Split Replay Sphere Manager



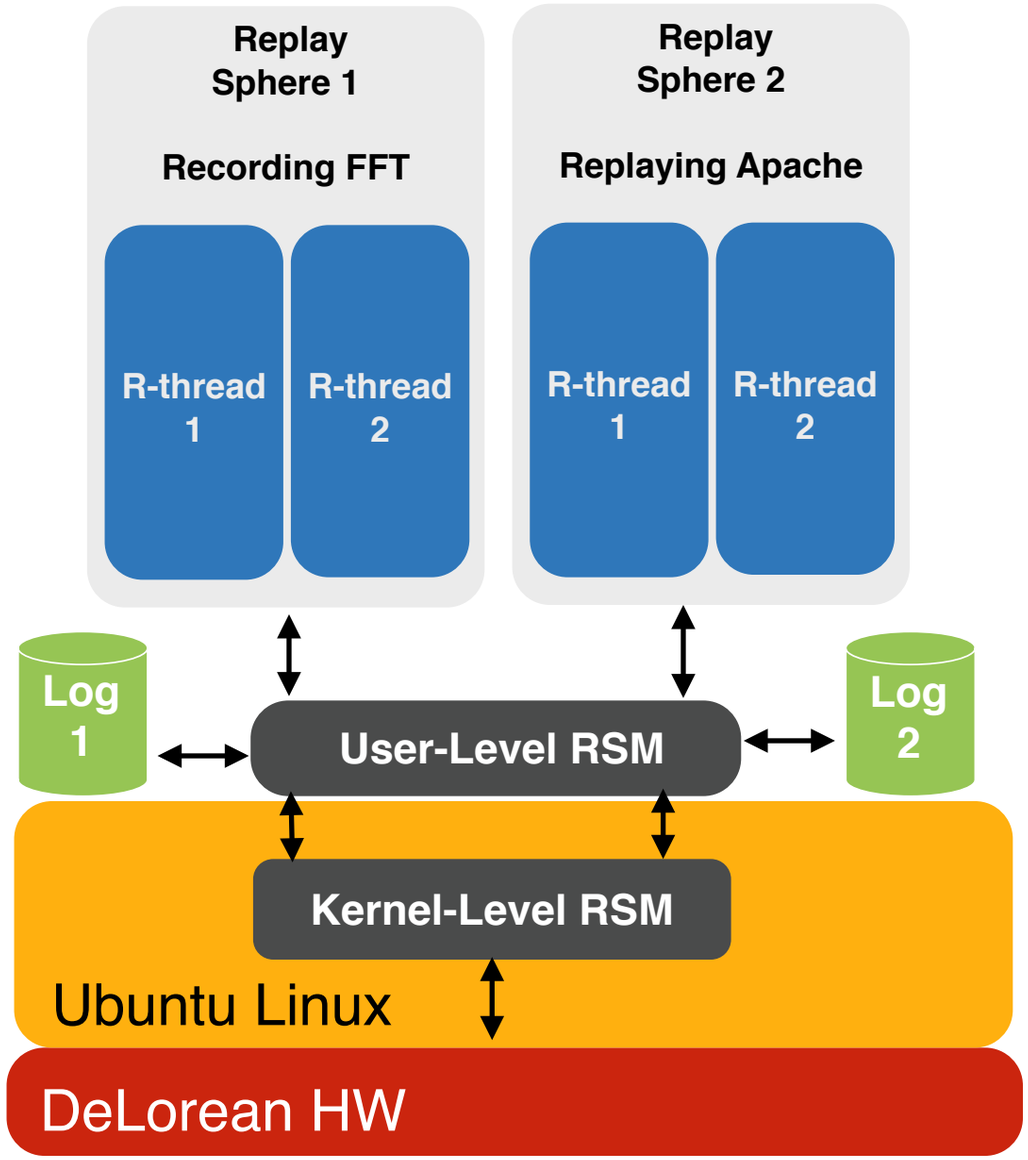
CapoOne: First Capo Implementation

- Records and replays multithreaded Linux apps running on multiprocessors
- Ubuntu Linux with modified kernel:
  - Added support for spheres, R-threads
  - Made some functions more deterministic
- Split Replay Sphere Manager
- Simulated HW-Based replay system:



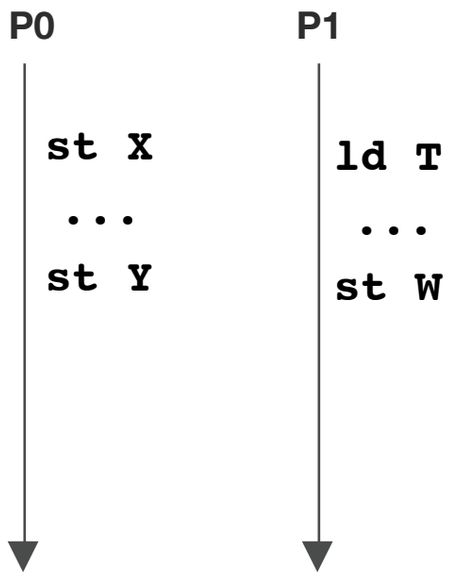
CapoOne: First Capo Implementation

- Records and replays multithreaded Linux apps running on multiprocessors
- Ubuntu Linux with modified kernel:
  - Added support for spheres, R-threads
  - Made some functions more deterministic
- Split Replay Sphere Manager
- Simulated HW-Based replay system:
  - DeLorean [Montesinos *et al.* ISCA'08]

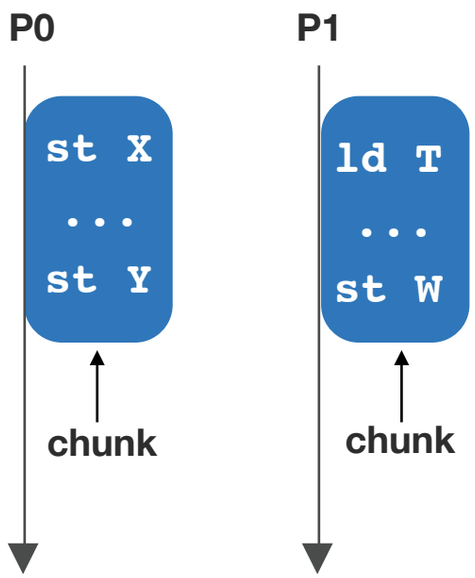


DeLorean: Chunk-Based Record/Replay

DeLorean: Chunk-Based Record/Replay

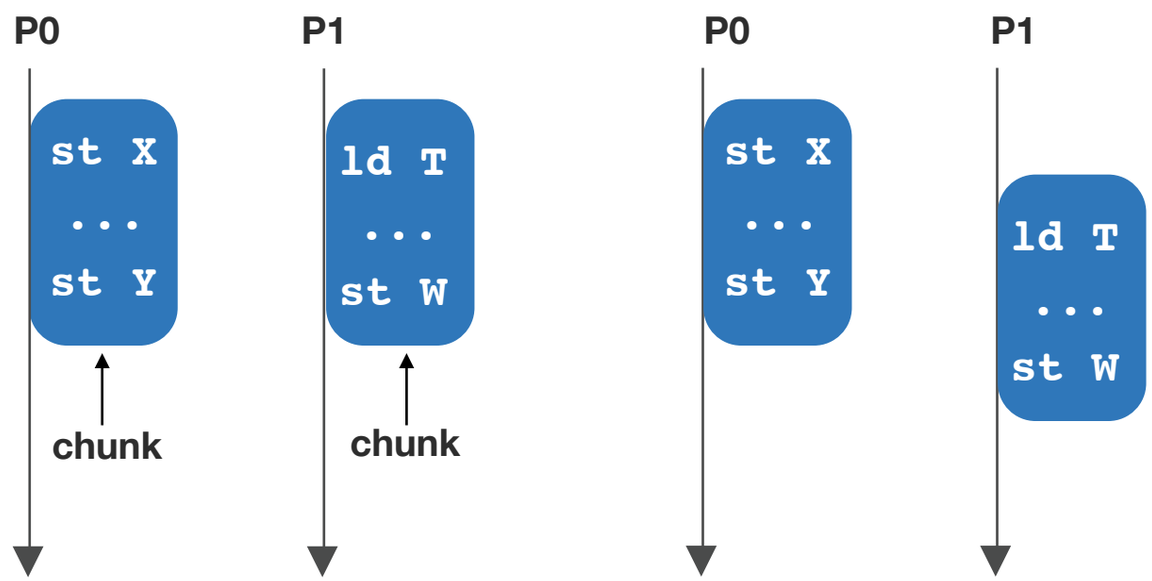


DeLorean: Chunk-Based Record/Replay



HW groups consecutive dynamic instructions

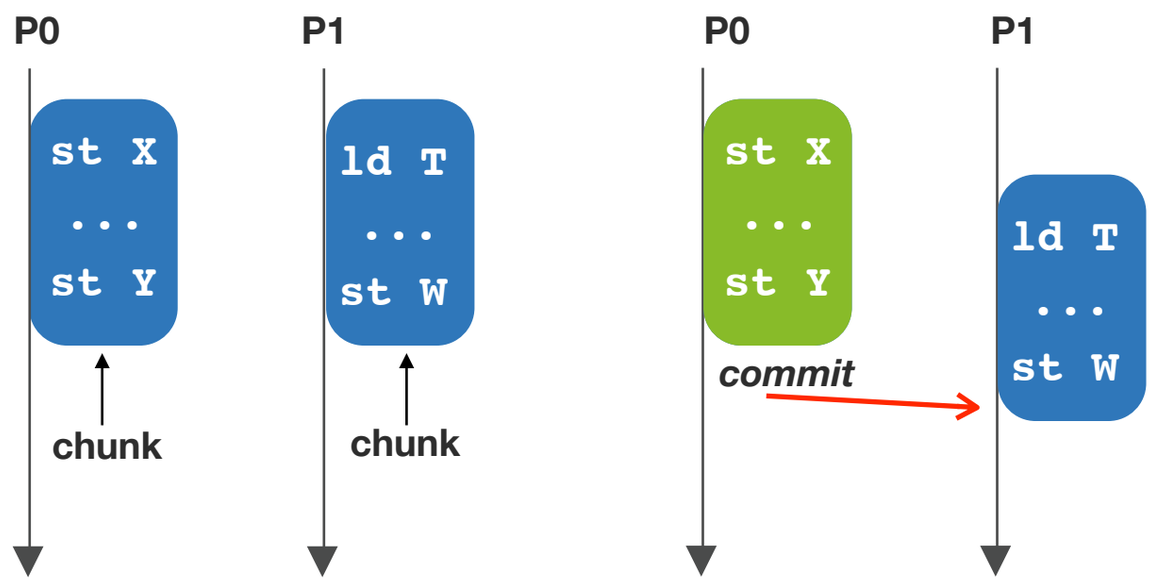
DeLorean: Chunk-Based Record/Replay



HW groups consecutive dynamic instructions

Execute chunks atomically

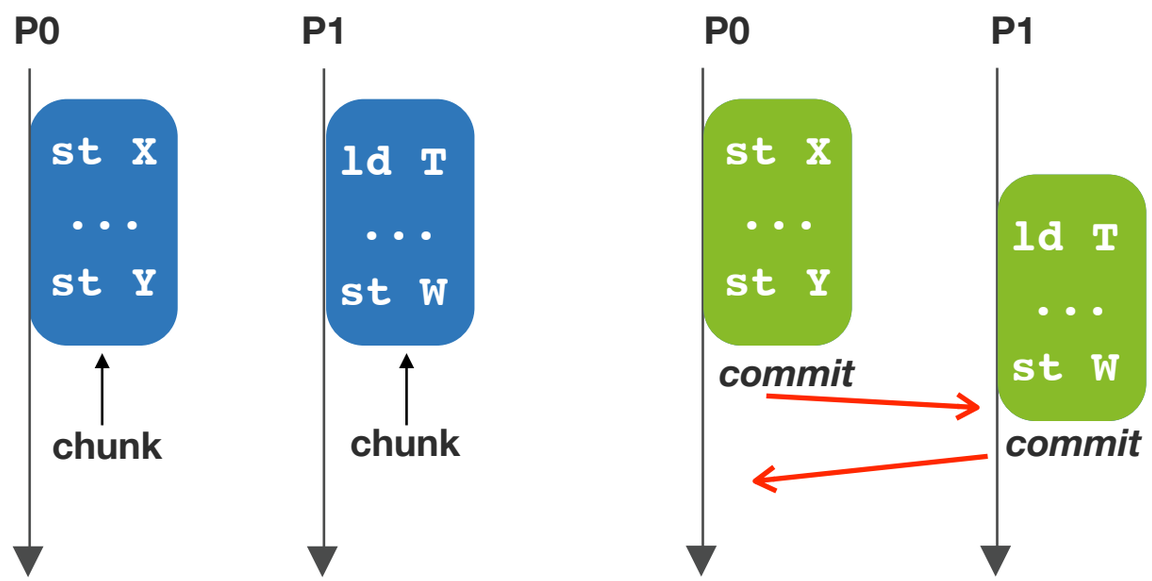
DeLorean: Chunk-Based Record/Replay



HW groups consecutive dynamic instructions

Execute chunks atomically

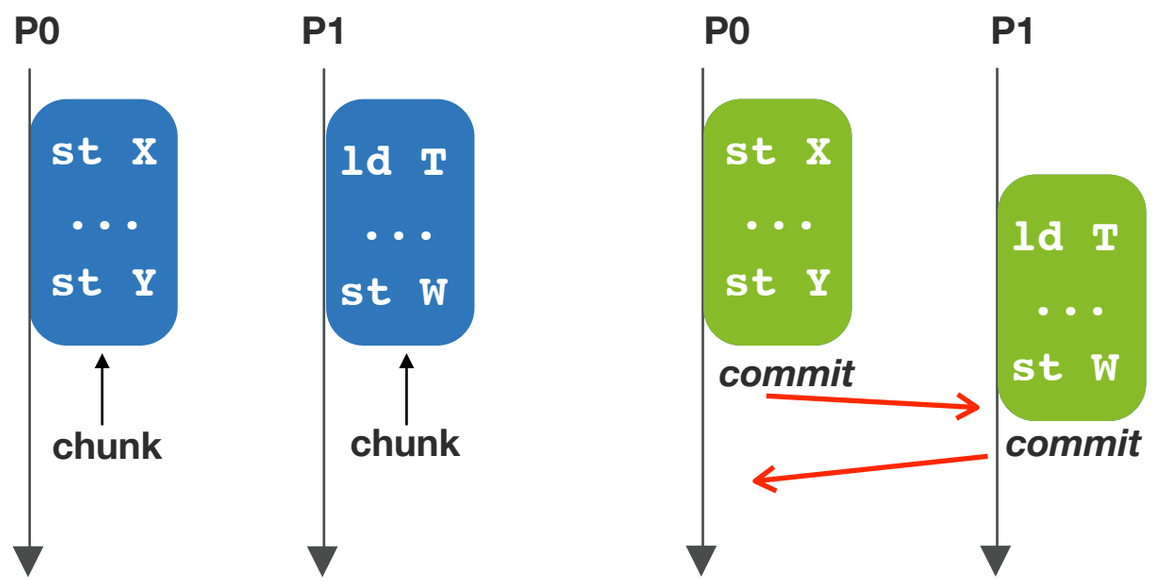
DeLorean: Chunk-Based Record/Replay



HW groups consecutive dynamic instructions

Execute chunks atomically

DeLorean: Chunk-Based Record/Replay

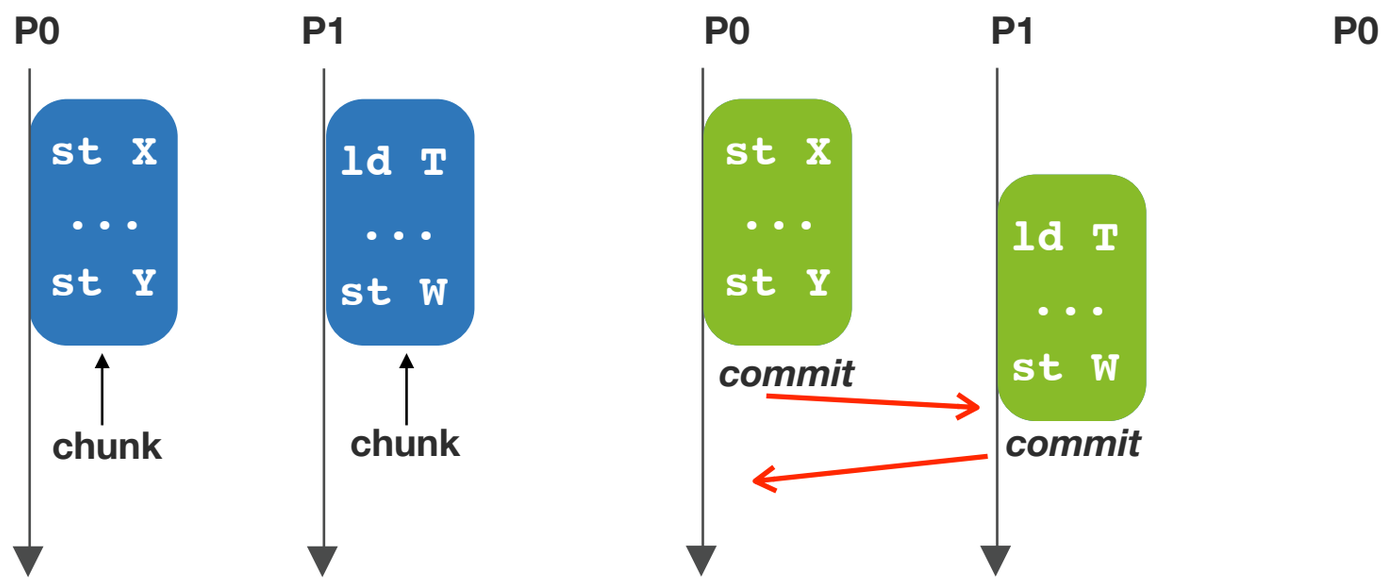


HW groups consecutive dynamic instructions

Execute chunks atomically

Execute chunks in isolation

DeLorean: Chunk-Based Record/Replay

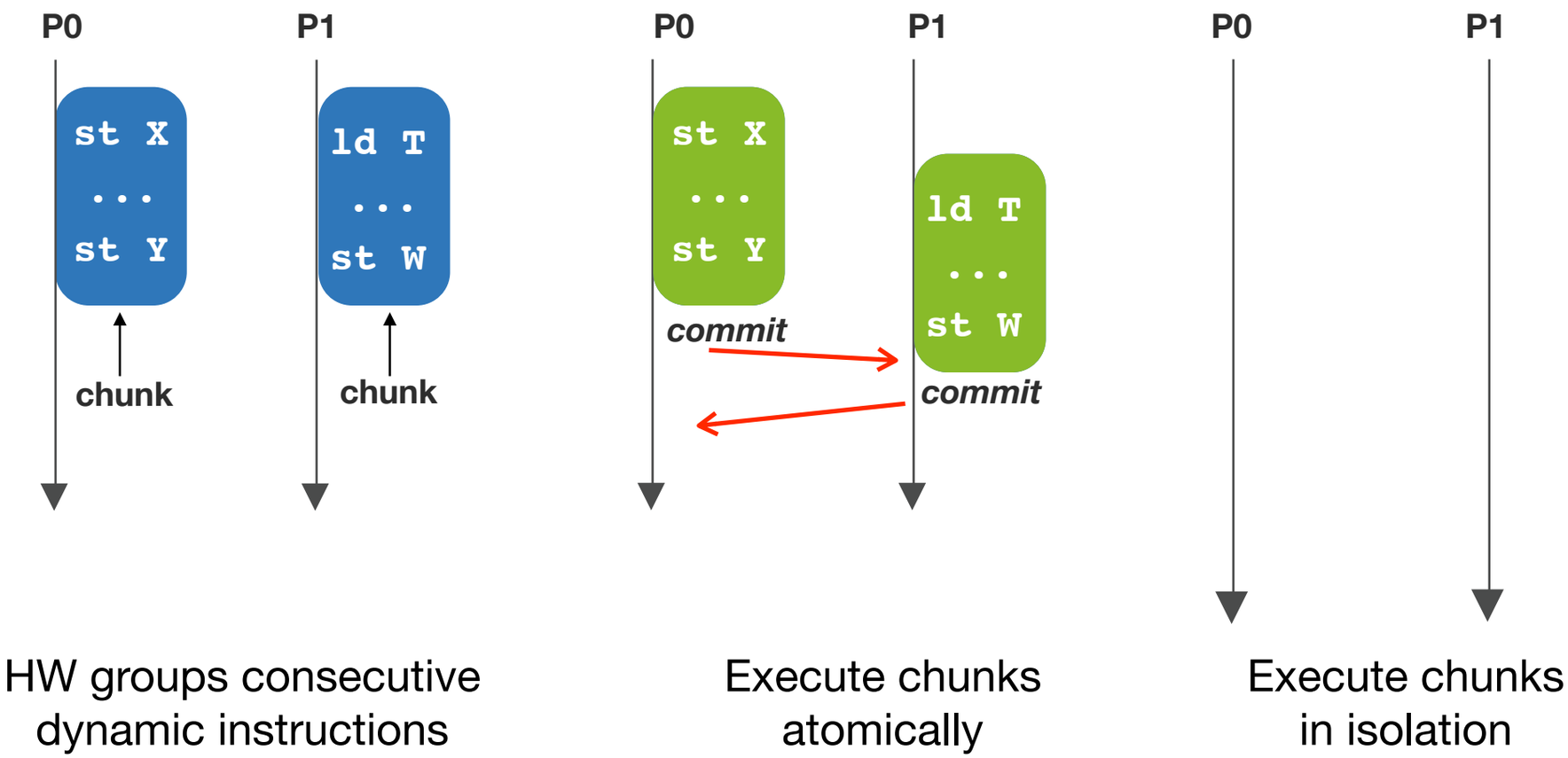


HW groups consecutive dynamic instructions

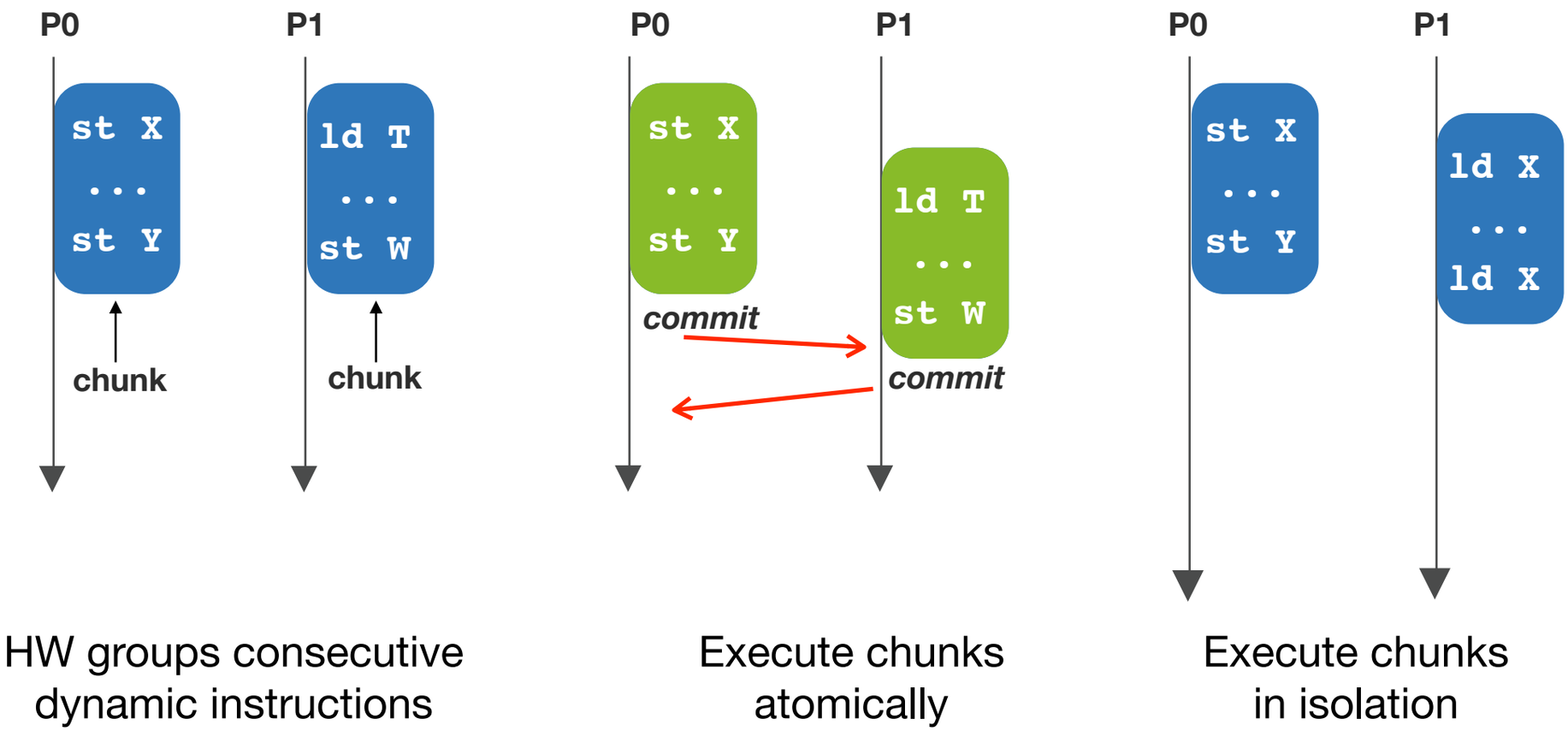
Execute chunks atomically

Execute chunks in isolation

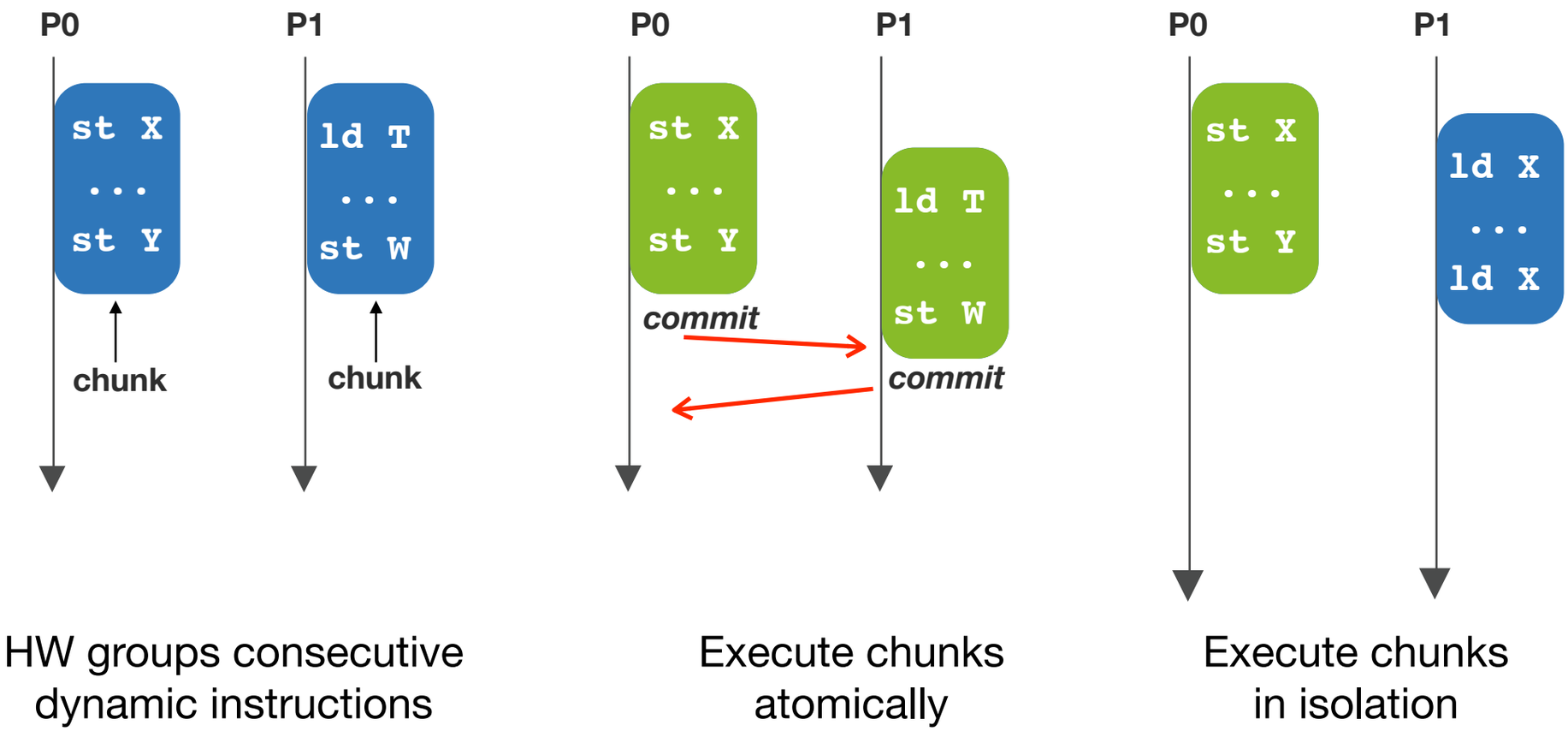
DeLorean: Chunk-Based Record/Replay



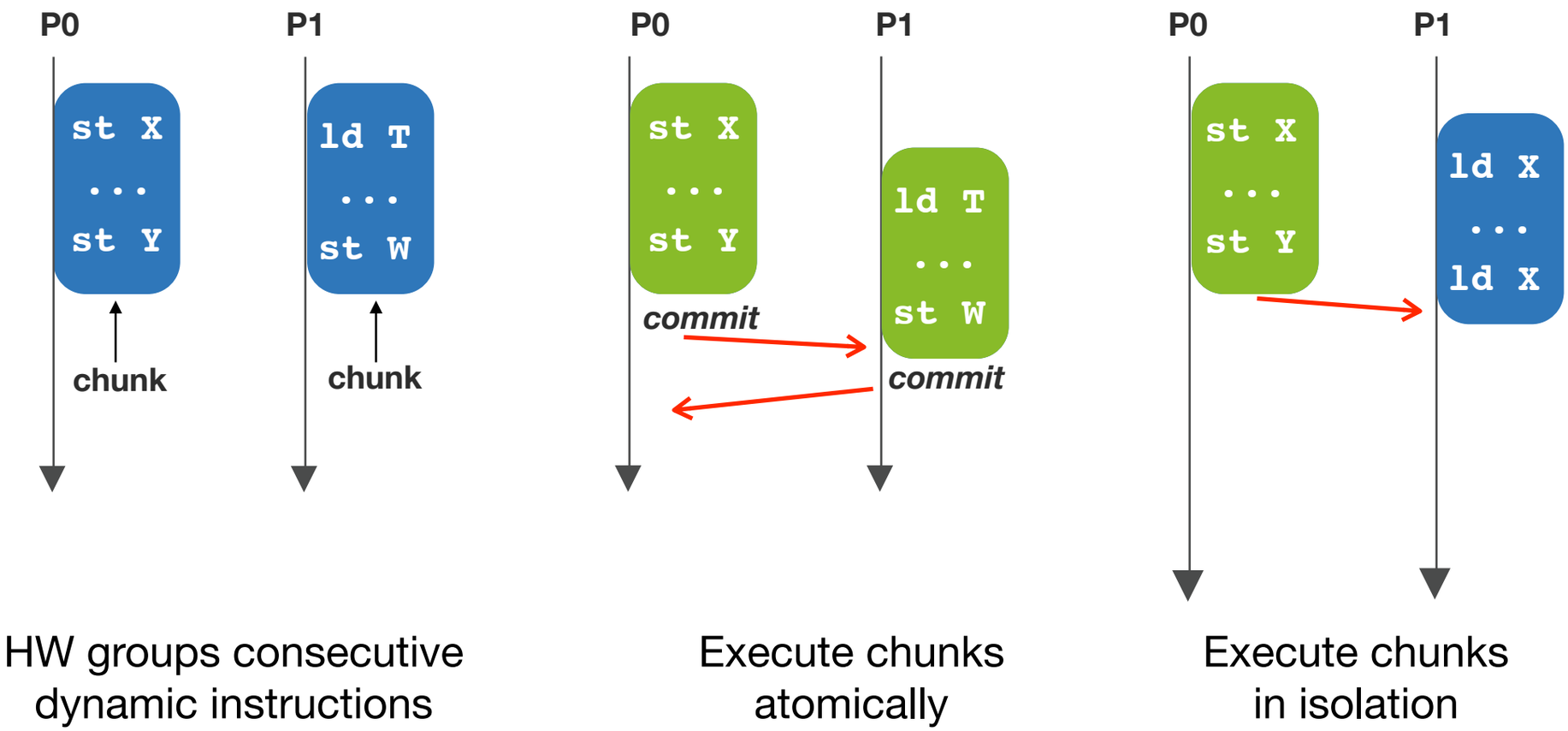
DeLorean: Chunk-Based Record/Replay



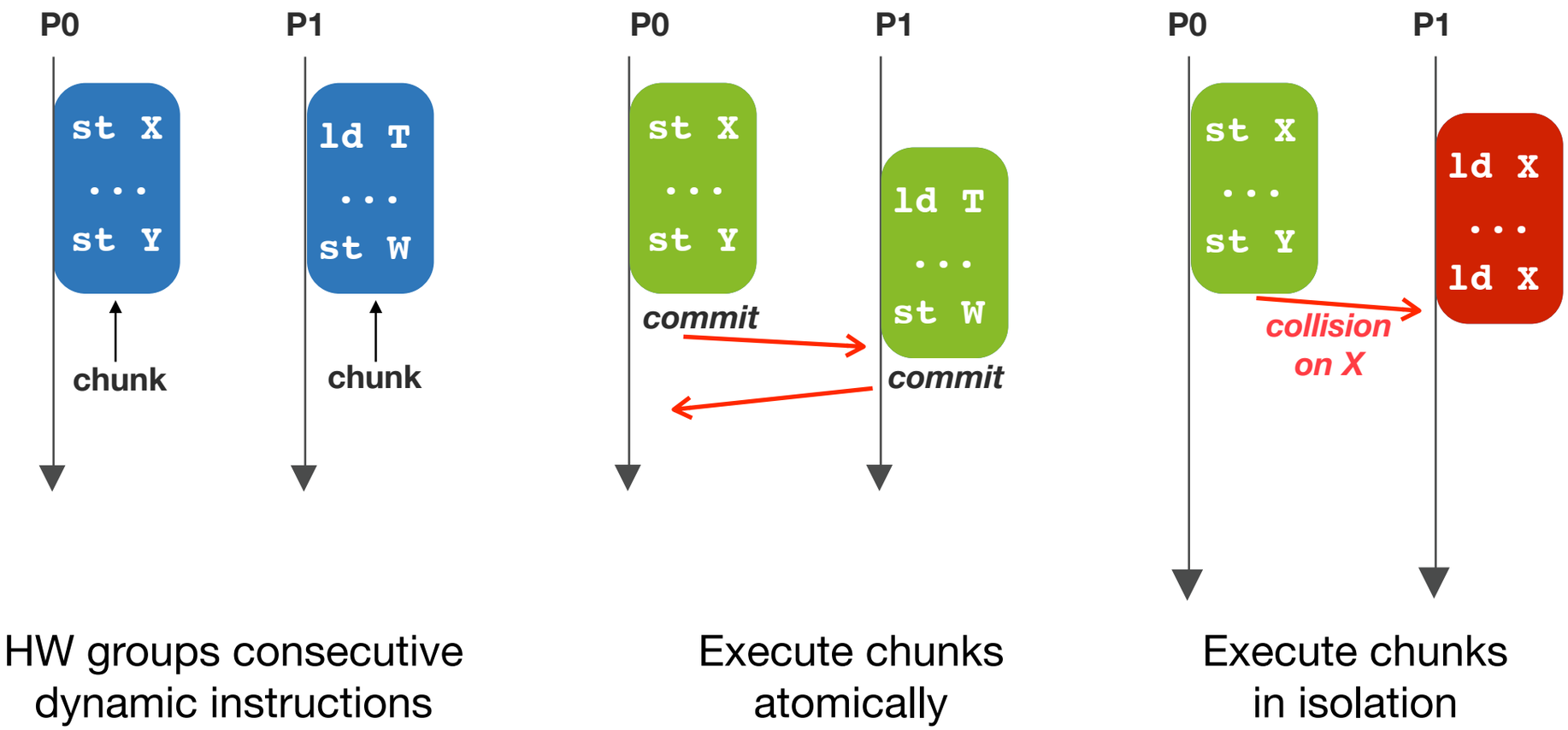
DeLorean: Chunk-Based Record/Replay



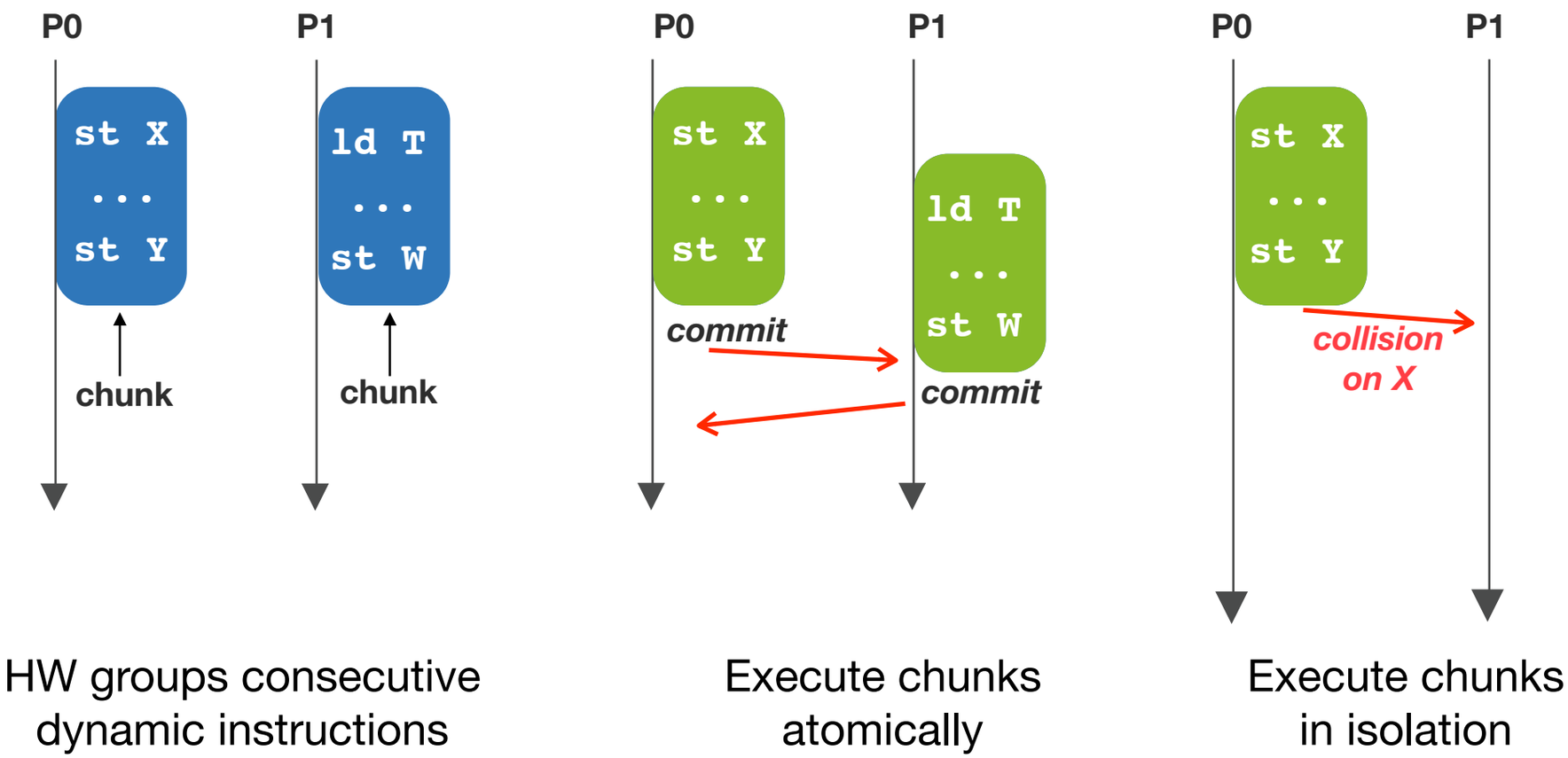
DeLorean: Chunk-Based Record/Replay



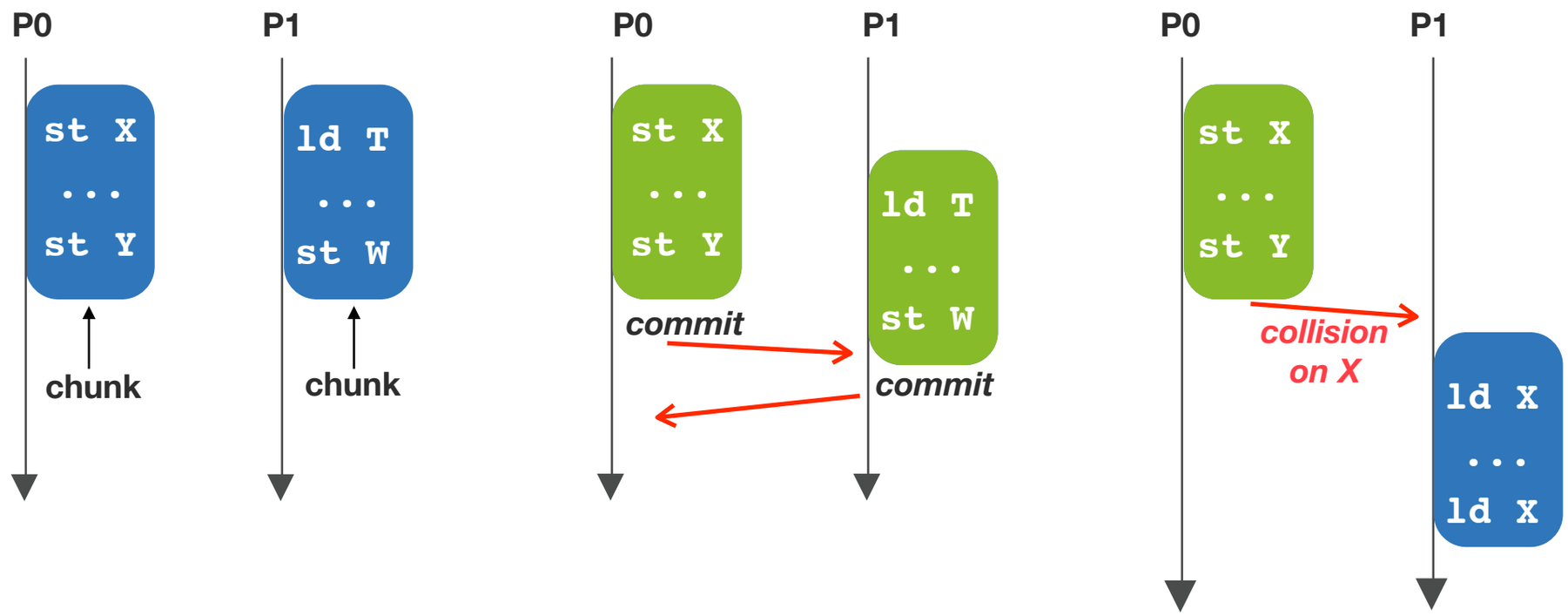
DeLorean: Chunk-Based Record/Replay



DeLorean: Chunk-Based Record/Replay



DeLorean: Chunk-Based Record/Replay

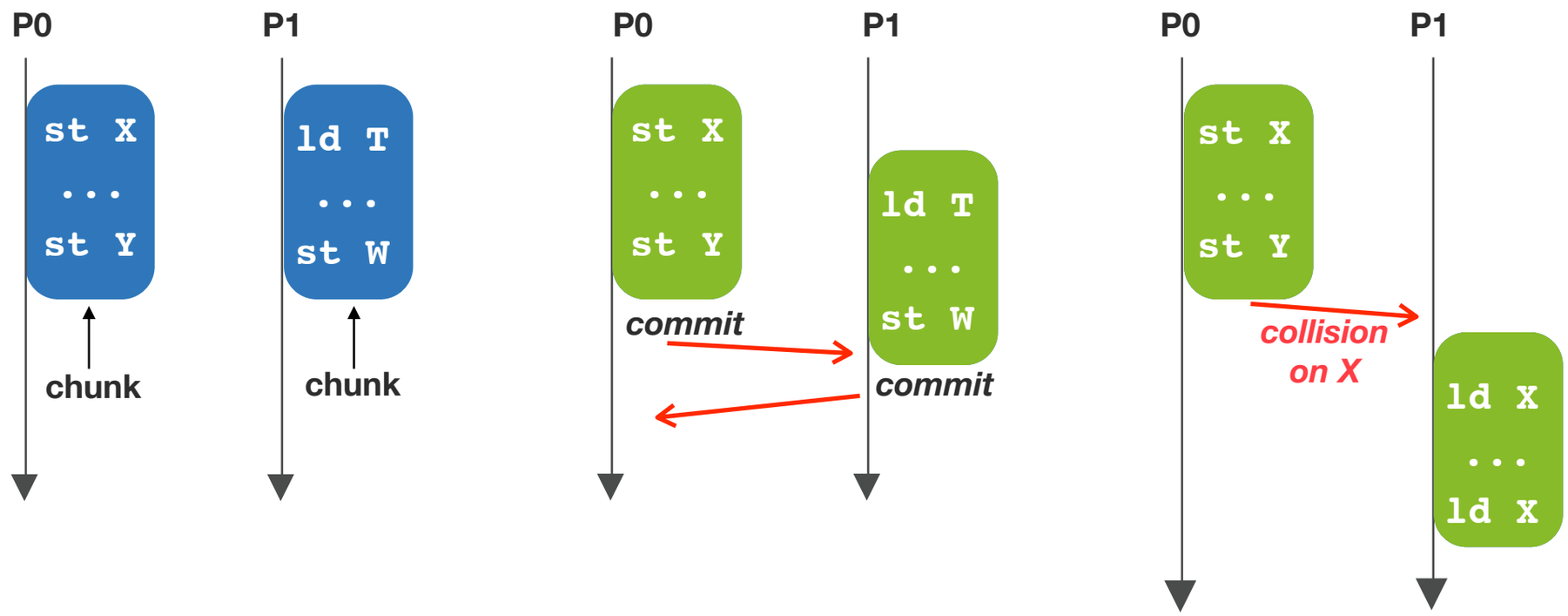


HW groups consecutive dynamic instructions

Execute chunks atomically

Execute chunks in isolation

DeLorean: Chunk-Based Record/Replay

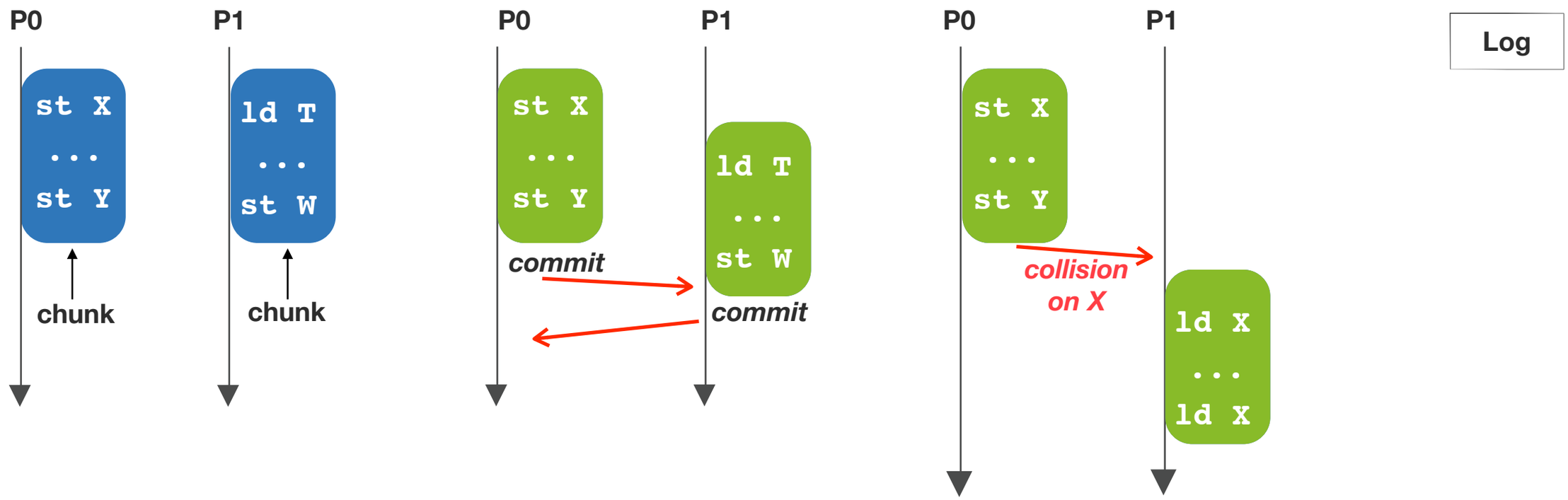


HW groups consecutive dynamic instructions

Execute chunks atomically

Execute chunks in isolation

DeLorean: Chunk-Based Record/Replay



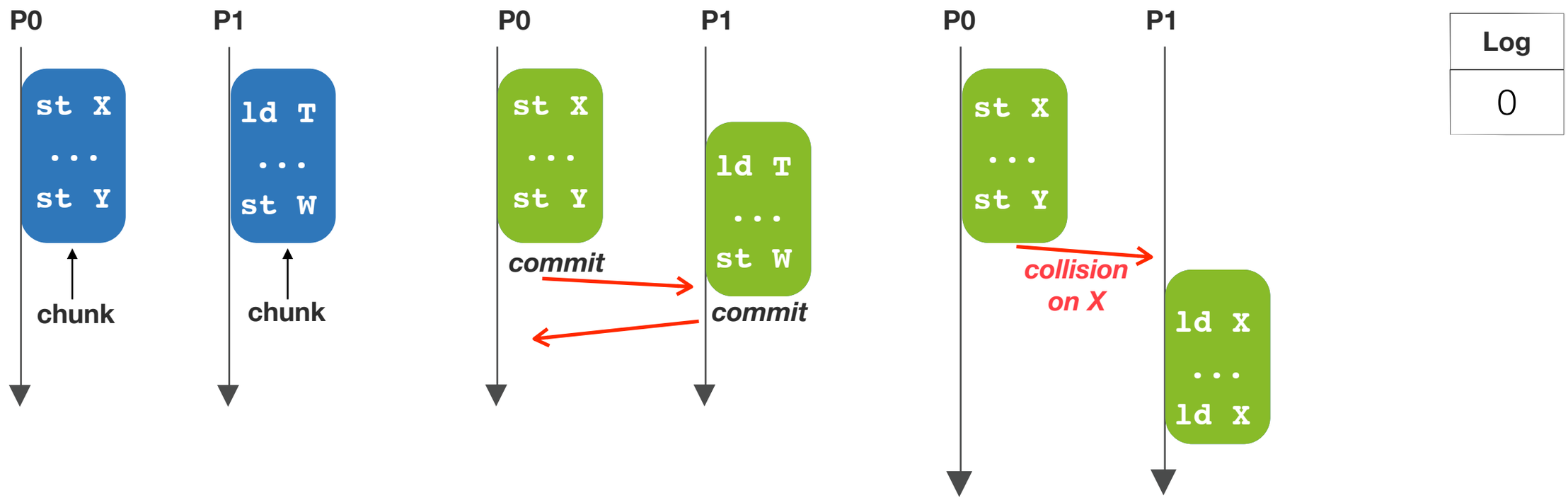
HW groups consecutive dynamic instructions

Execute chunks atomically

Execute chunks in isolation

Record chunk commit order

DeLorean: Chunk-Based Record/Replay



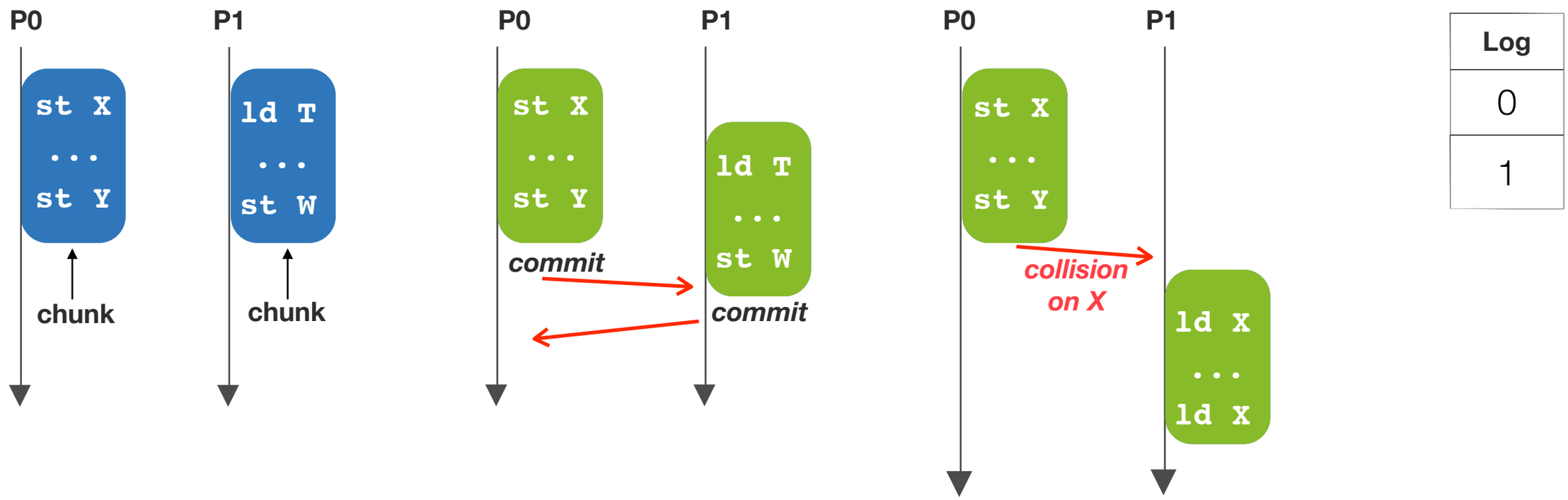
HW groups consecutive dynamic instructions

Execute chunks atomically

Execute chunks in isolation

Record chunk commit order

DeLorean: Chunk-Based Record/Replay



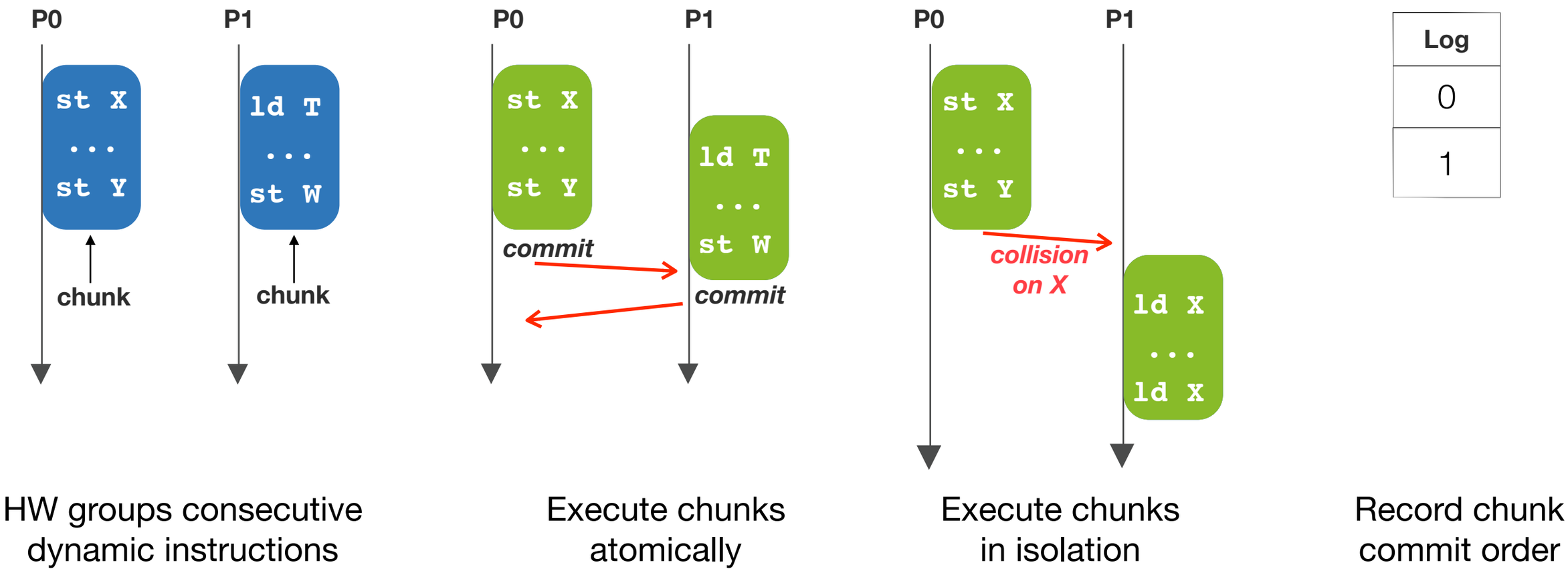
HW groups consecutive dynamic instructions

Execute chunks atomically

Execute chunks in isolation

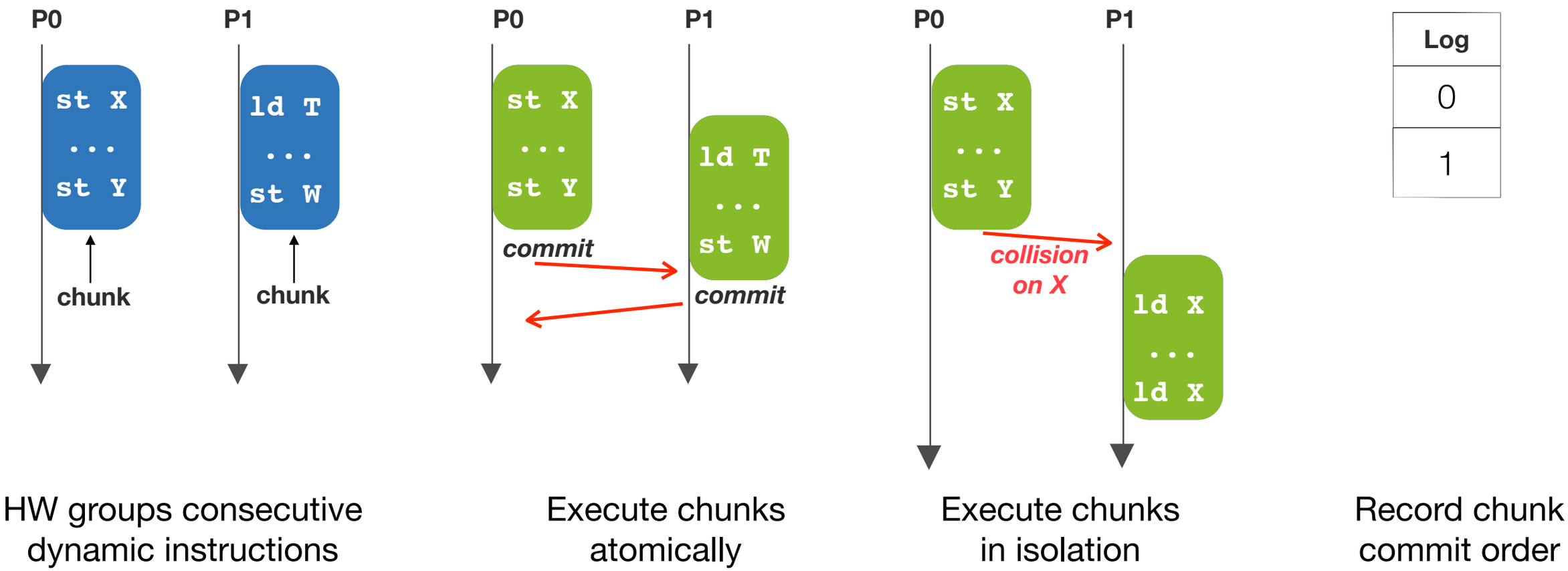
Record chunk commit order

DeLorean: Chunk-Based Record/Replay



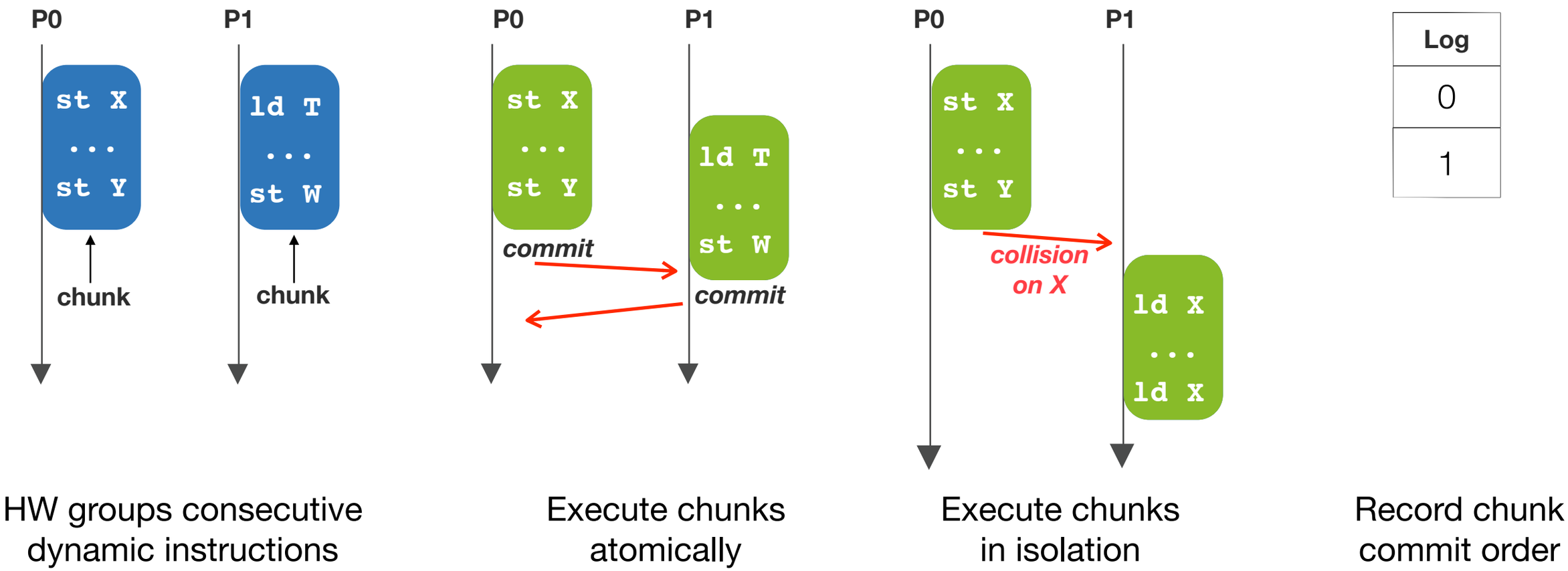
- HW records chunk commit order in Processor Interleaving Log

DeLorean: Chunk-Based Record/Replay



- HW records chunk commit order in Processor Interleaving Log
  - Also records size of some irregular chunks in Chunk Size Log

DeLorean: Chunk-Based Record/Replay



- HW records chunk commit order in Processor Interleaving Log
  - Also records size of some irregular chunks in Chunk Size Log
- During replay: generate same chunks and commit them in same order

From Full-System Replay to Sphere-Based Replay

From Full-System Replay to Sphere-Based Replay

- Only log non-deterministic events that affect a sphere

From Full-System Replay to Sphere-Based Replay

- Only log non-deterministic events that affect a sphere
  - Log entries record R-thread IDs, not processor IDs

From Full-System Replay to Sphere-Based Replay

- Only log non-deterministic events that affect a sphere
  - Log entries record R-thread IDs, not processor IDs
  - Information about other non-deterministic events is discarded

From Full-System Replay to Sphere-Based Replay

- Only log non-deterministic events that affect a sphere
  - Log entries record R-thread IDs, not processor IDs
  - Information about other non-deterministic events is discarded

- Memory Interleaving Log must only include interleaving of instructions from R-threads within same sphere

From Full-System Replay to Sphere-Based Replay

- Only log non-deterministic events that affect a sphere
  - Log entries record R-thread IDs, not processor IDs
  - Information about other non-deterministic events is discarded
- Memory Interleaving Log must only include interleaving of instructions from R-threads within same sphere
  - Changed DeLorean's chunk truncation rules to enforce isolation

From Full-System Replay to Sphere-Based Replay

- Only log non-deterministic events that affect a sphere
  - Log entries record R-thread IDs, not processor IDs
  - Information about other non-deterministic events is discarded
- Memory Interleaving Log must only include interleaving of instructions from R-threads within same sphere
  - Changed DeLorean's chunk truncation rules to enforce isolation
  - Example: System calls cause chunk truncation in CapoOne, not in DeLorean

New Chunking Rules Provide R-Thread Isolation

New Chunking Rules Provide R-Thread Isolation

DeLorean

Processor P1

Processor
Interleaving
Log

New Chunking Rules Provide R-Thread Isolation

DeLorean

Processor P1

500
501

Processor
Interleaving
Log

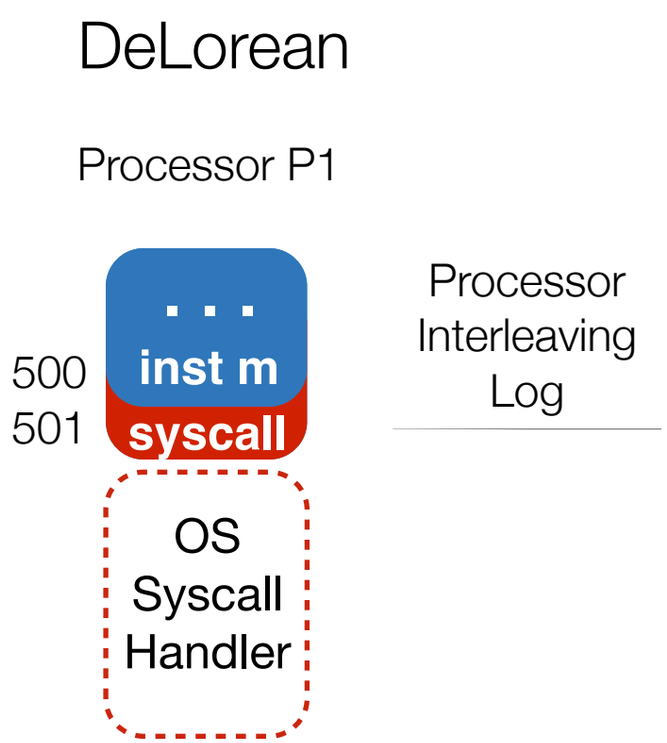
New Chunking Rules Provide R-Thread Isolation

DeLorean

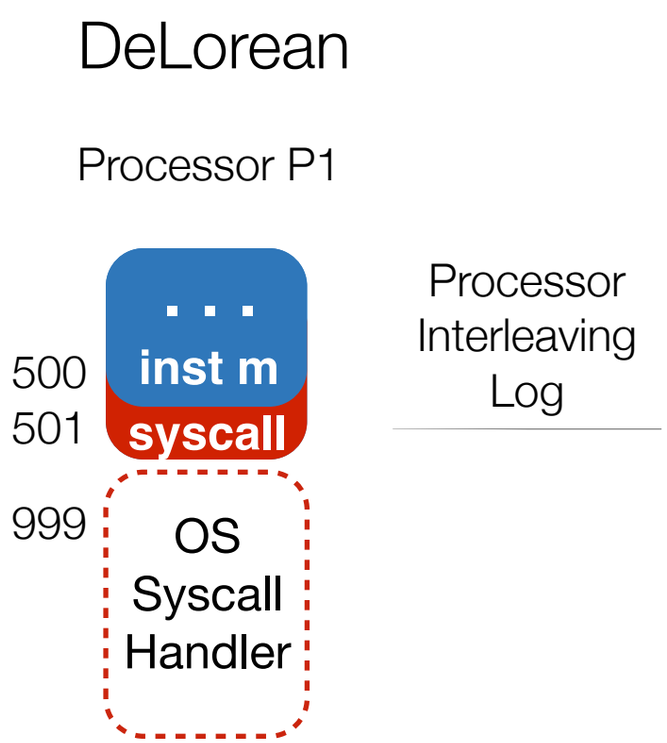
Processor P1

Processor Interleaving Log

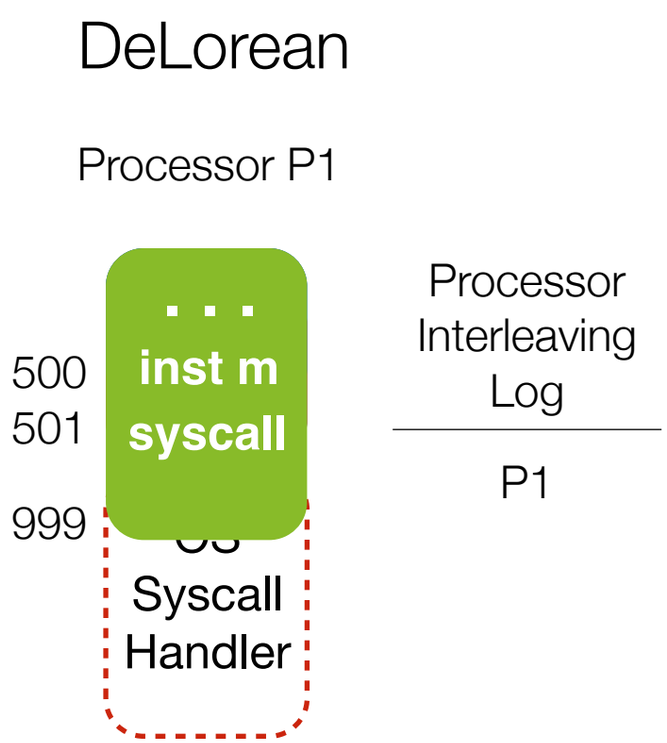
New Chunking Rules Provide R-Thread Isolation



New Chunking Rules Provide R-Thread Isolation



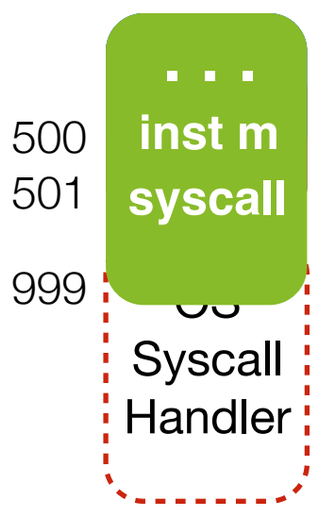
New Chunking Rules Provide R-Thread Isolation



New Chunking Rules Provide R-Thread Isolation

DeLorean

Processor P1



Processor Interleaving Log

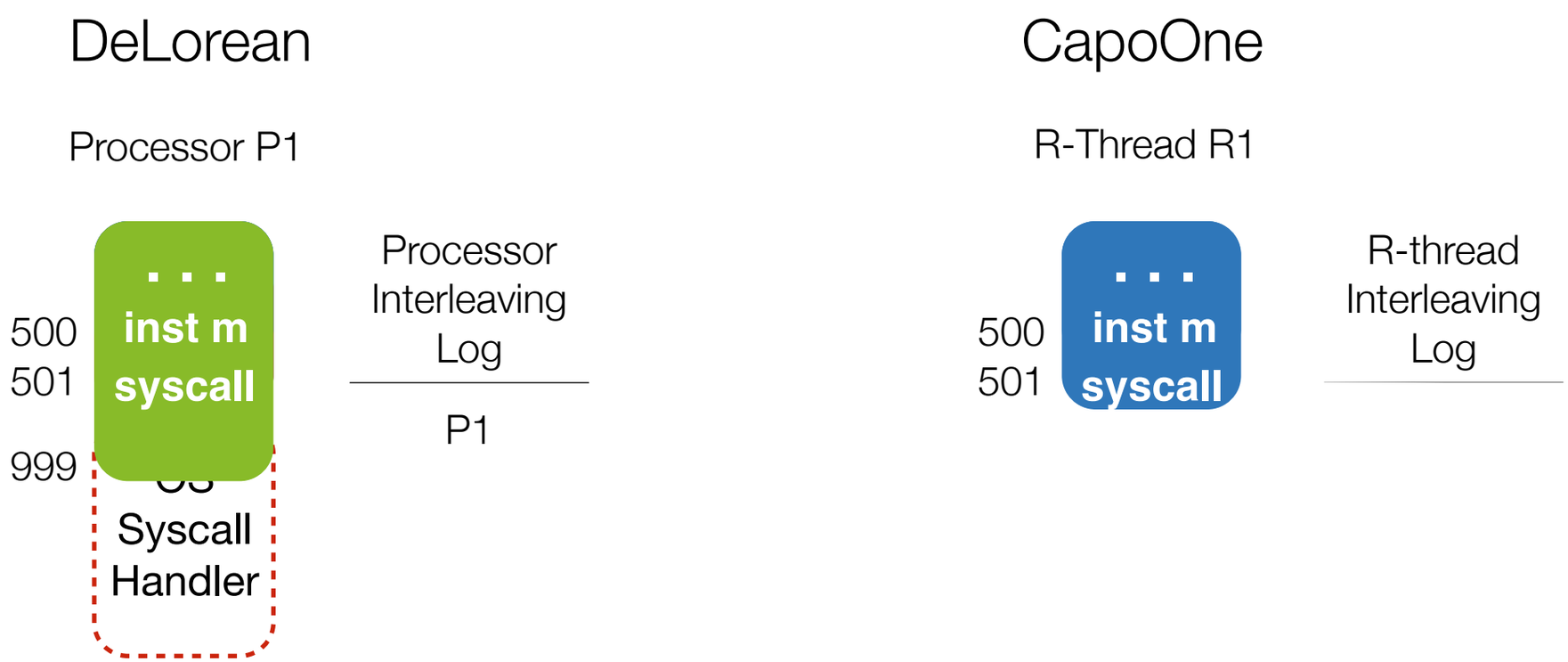
P1

CapoOne

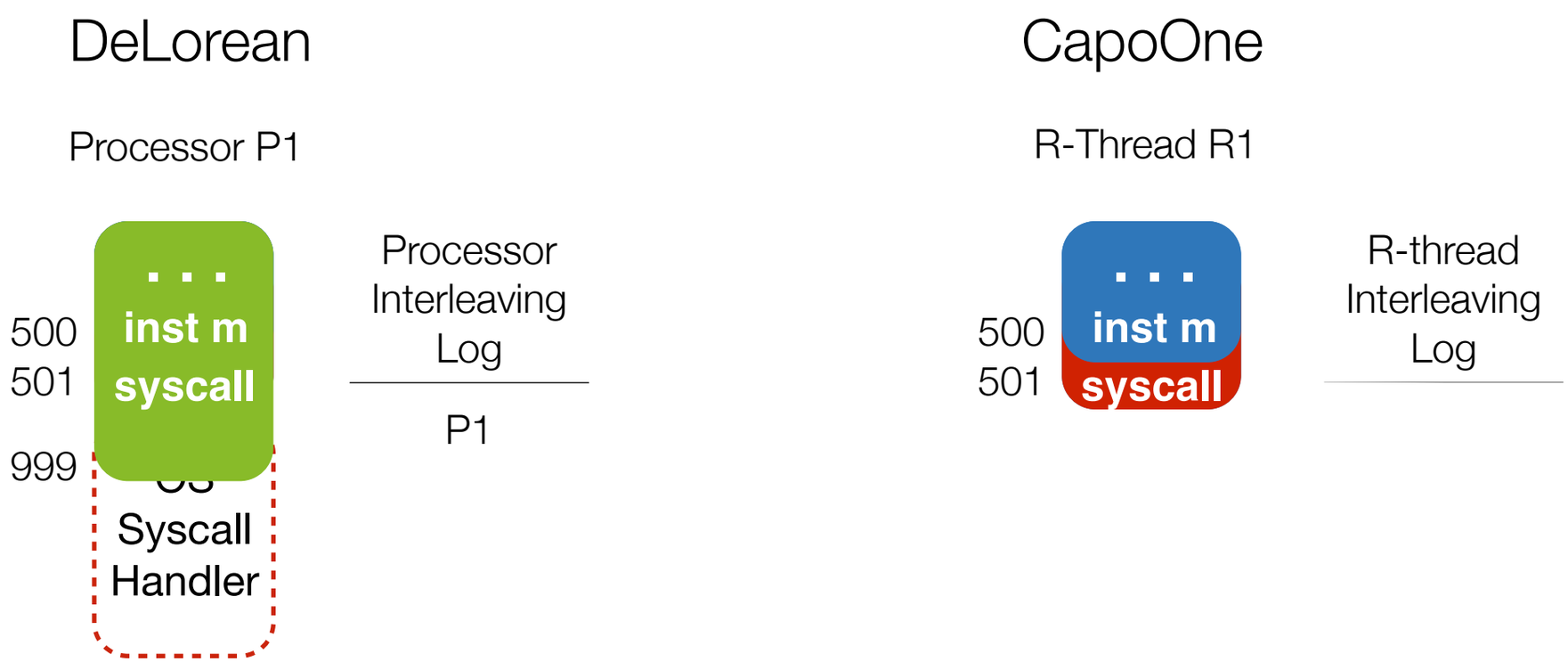
R-Thread R1

R-thread Interleaving Log

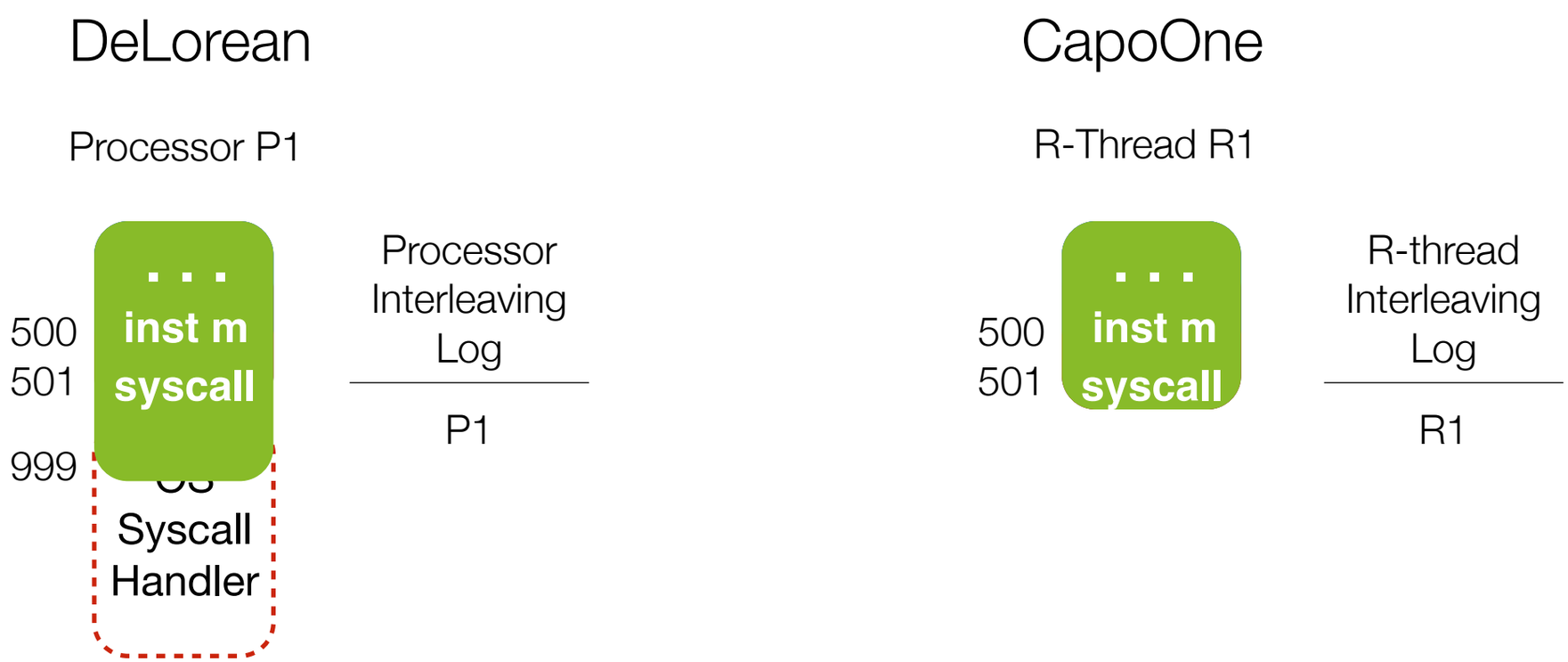
New Chunking Rules Provide R-Thread Isolation



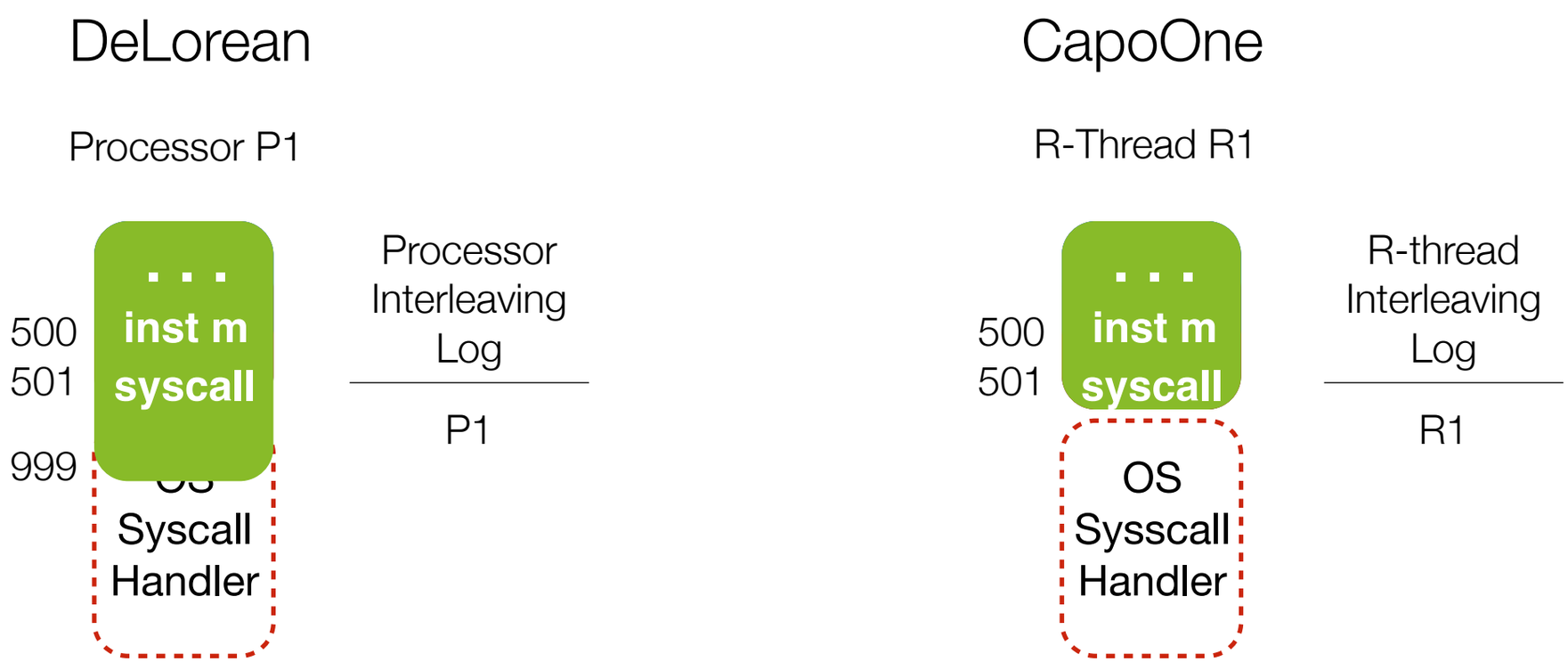
New Chunking Rules Provide R-Thread Isolation



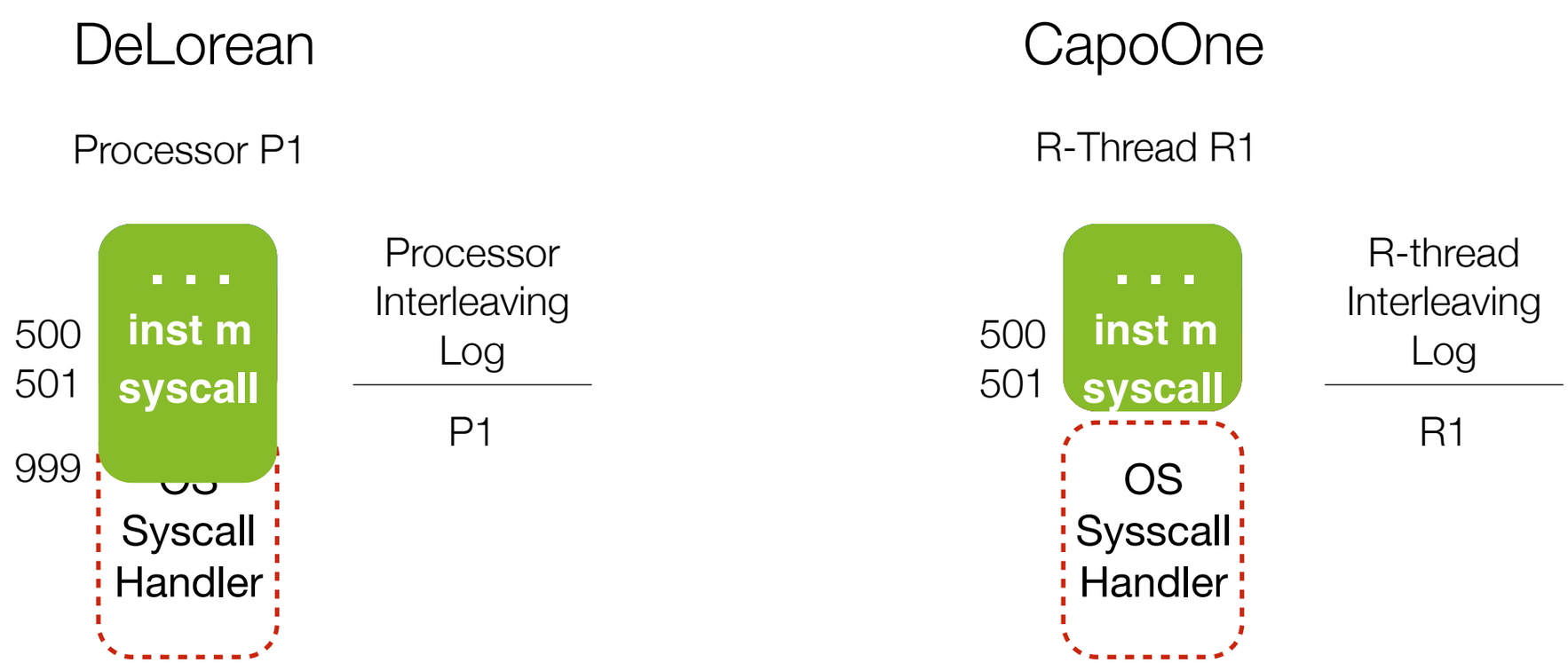
New Chunking Rules Provide R-Thread Isolation



New Chunking Rules Provide R-Thread Isolation

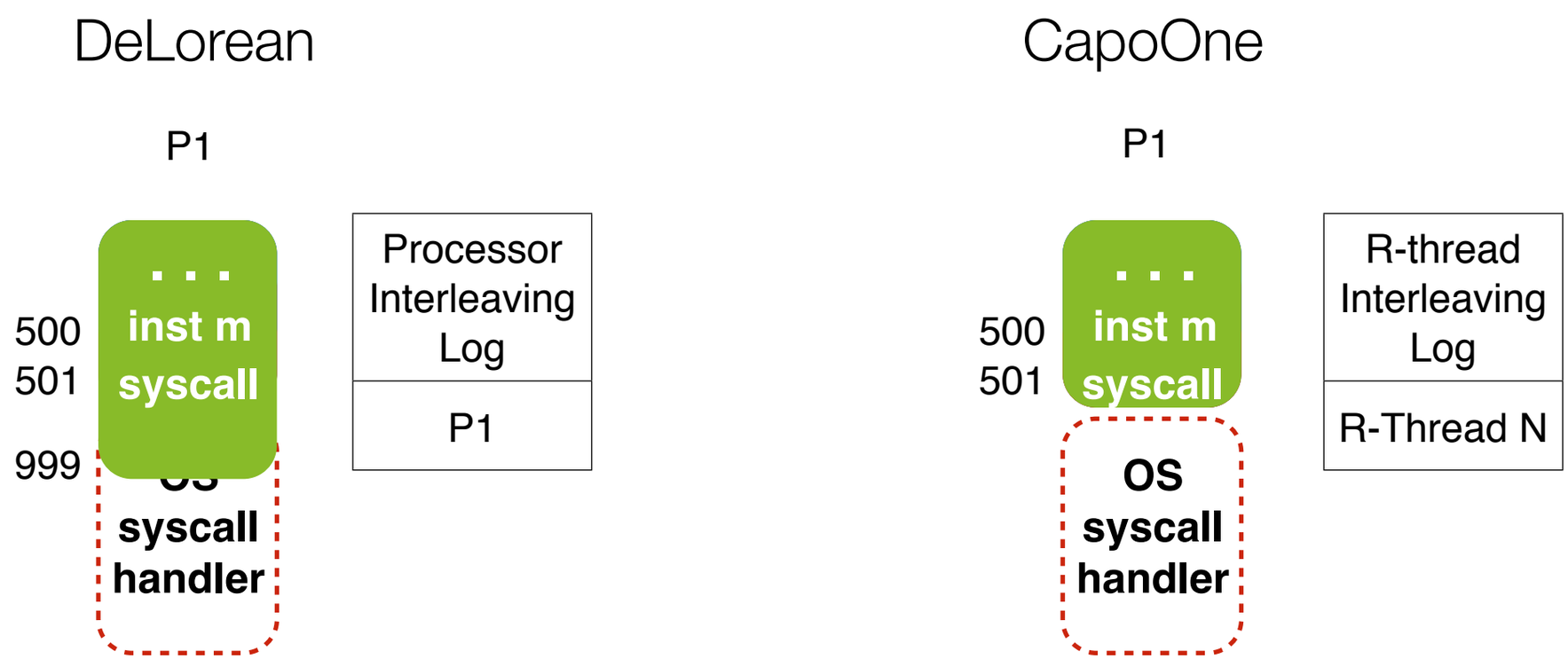


New Chunking Rules Provide R-Thread Isolation



- Inform HW to stop logging interleaving of instructions from current proc
  - Always do it when execution leaves replay sphere
  - Otherwise, OS instructions might become part of interleaving log

New Chunking Rules Provide R-Thread Isolation



- Inform HW to stop logging interleaving of instructions from current proc
  - Always do it when execution leaves replay sphere
  - Otherwise, OS instructions might become part of interleaving log

New Chunking Rules Provide R-Thread Isolation

DeLorean

CapoOne

RSM and HW must work together to avoid any OS instruction to pollute the Memory Interleaving Log

- Inform HW to stop logging interleaving of instructions from current proc
  - Always do it when execution leaves replay sphere
  - Otherwise, OS instructions might become part of interleaving log

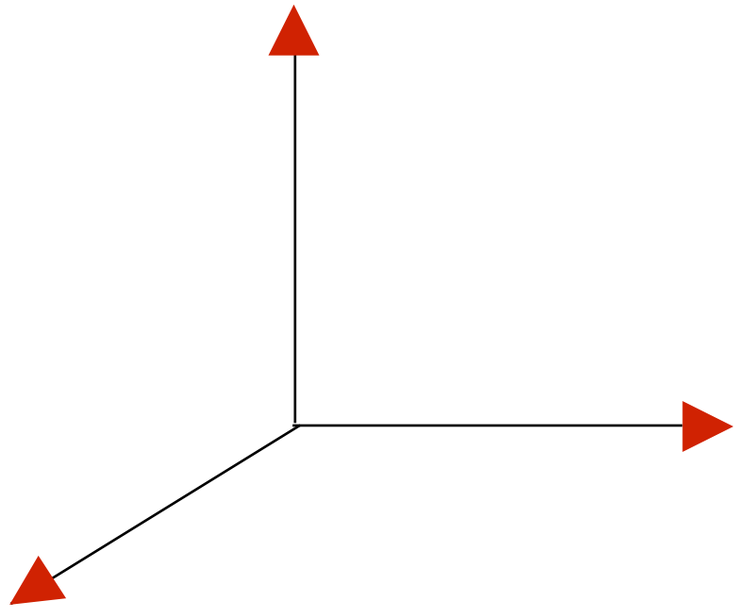
Handling Interrupts

Handling Interrupts

- Ensure interrupt handler code is not part of Memory Interleaving Log

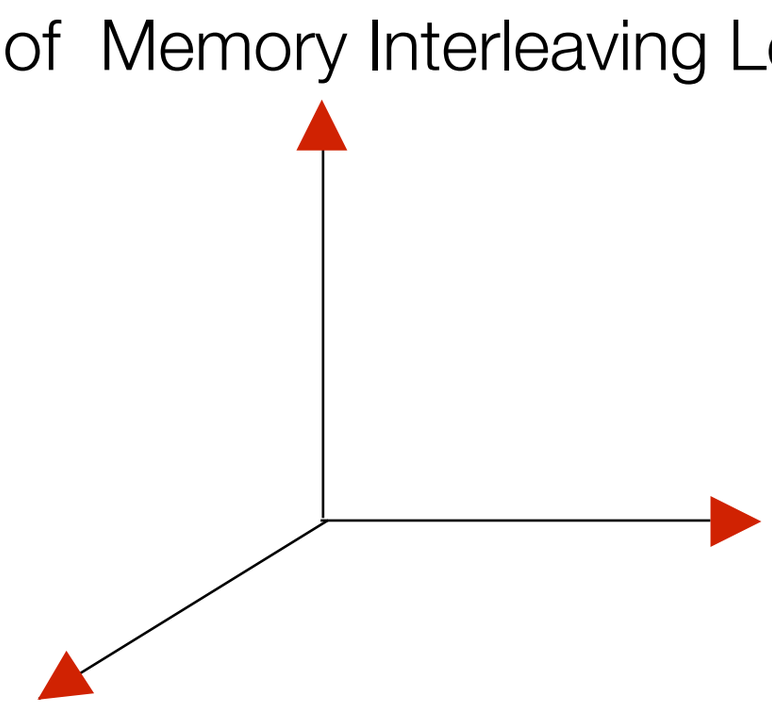
Handling Interrupts

- Ensure interrupt handler code is not part of Memory Interleaving Log
- Balance conflicting demands because of chunk-based execution:



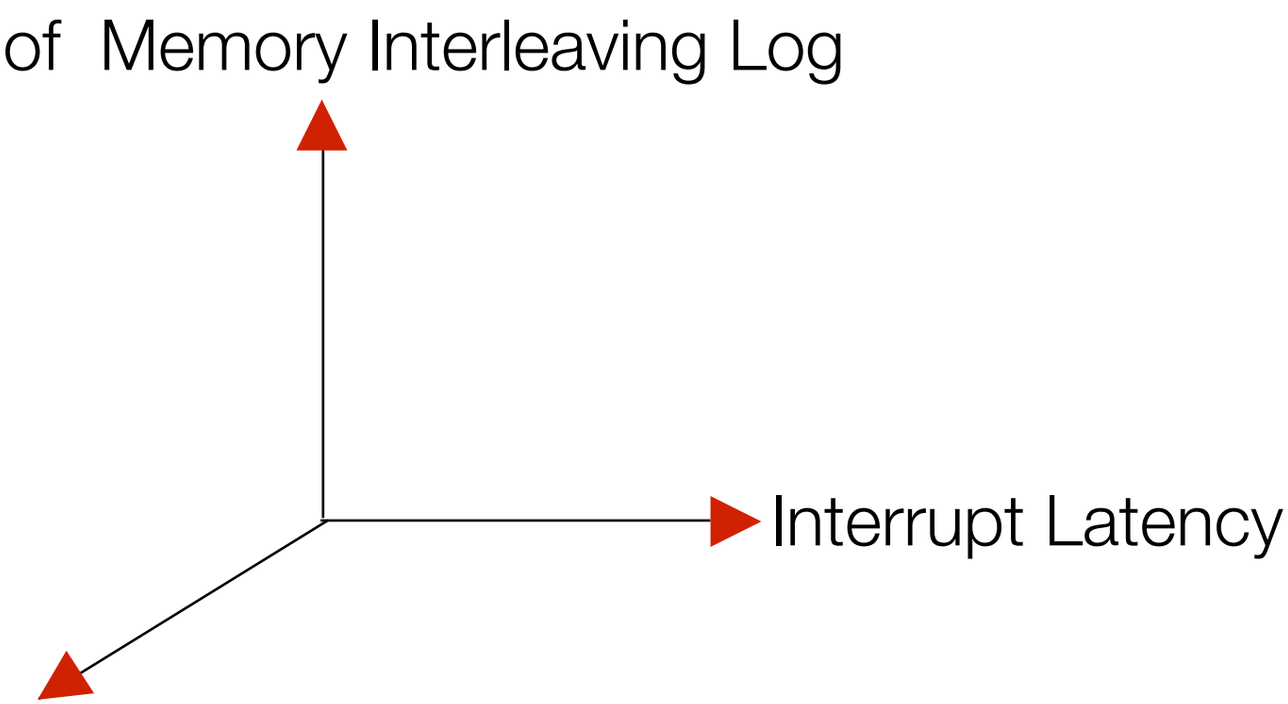
Handling Interrupts

- Ensure interrupt handler code is not part of Memory Interleaving Log
- Balance conflicting demands because of chunk-based execution:



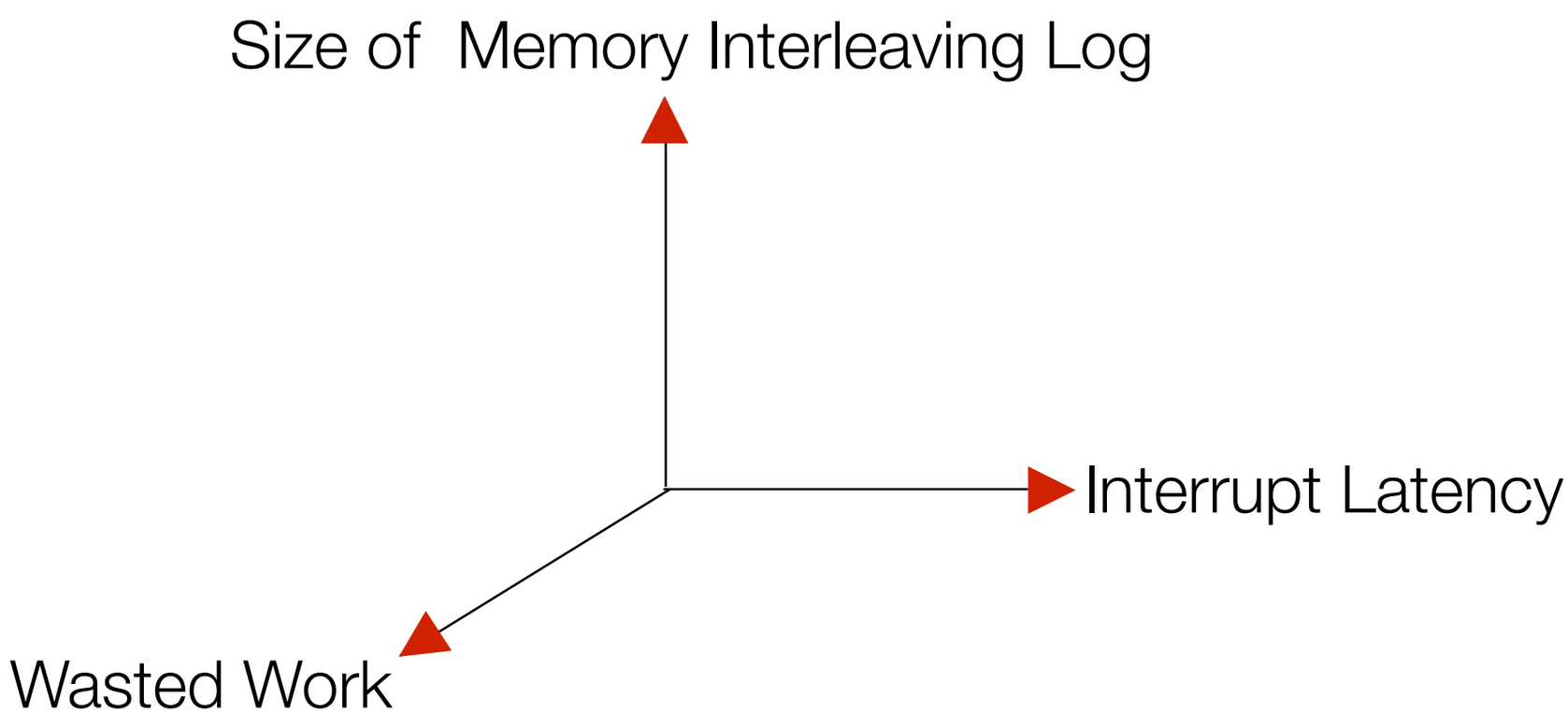
Handling Interrupts

- Ensure interrupt handler code is not part of Memory Interleaving Log
- Balance conflicting demands because of chunk-based execution:



Handling Interrupts

- Ensure interrupt handler code is not part of Memory Interleaving Log
- Balance conflicting demands because of chunk-based execution:



Handling Interrupts: Three Approaches

Handling Interrupts: Three Approaches

Finish First

Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First			

Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First			

Original Execution

R-thread R1

Chunk Size Log

Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First			

Original Execution

R-thread R1

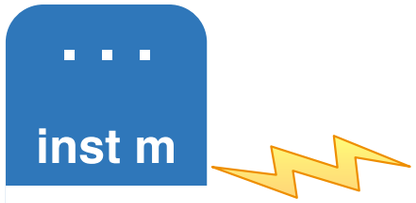
Chunk Size Log

Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First			

Original Execution

R-thread R1



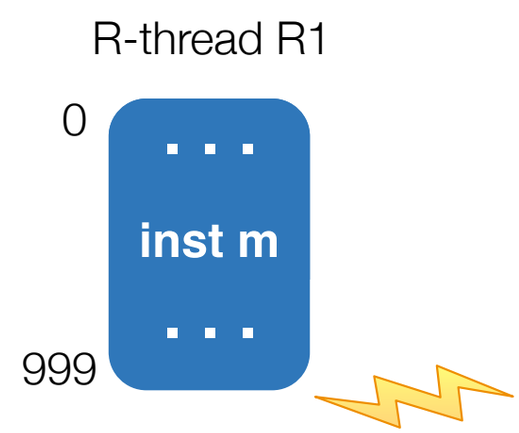
Chunk Size Log

Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First			

Original Execution

Chunk Size Log

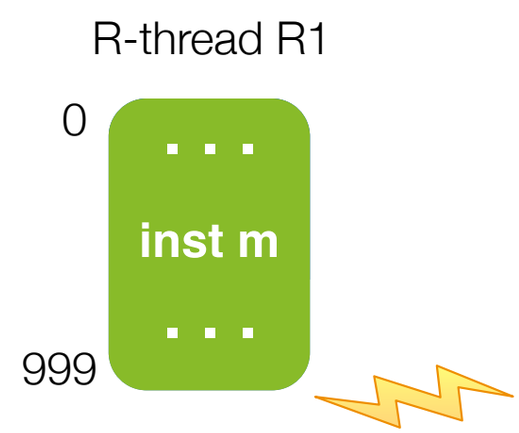


Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First			

Original Execution

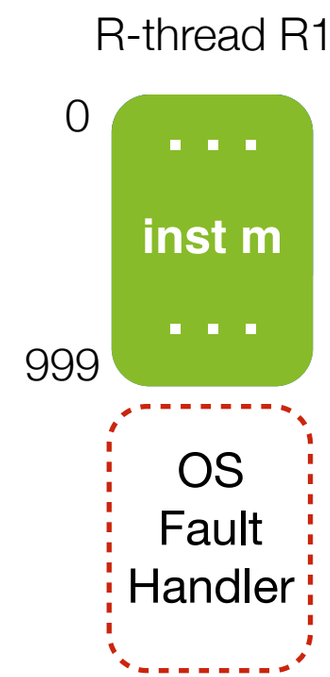
Chunk Size Log



Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First			

Original Execution



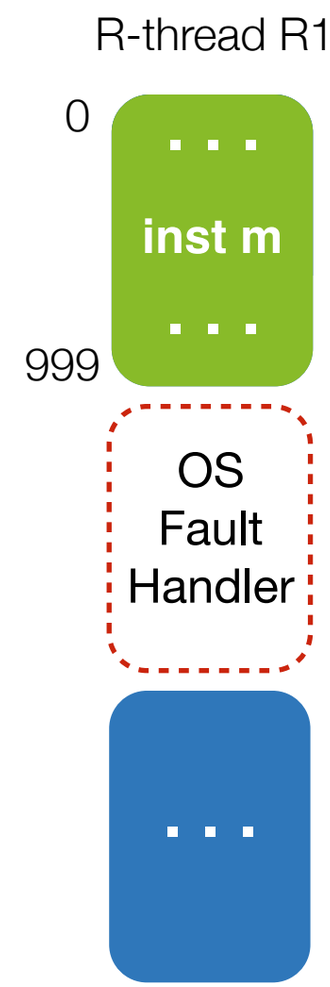
Chunk Size Log

Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First			

Original Execution

Chunk Size Log

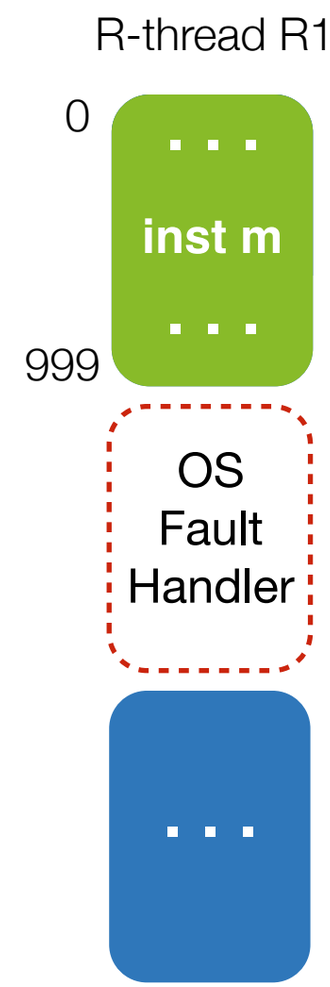


Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First	✔		

Original Execution

Chunk Size Log

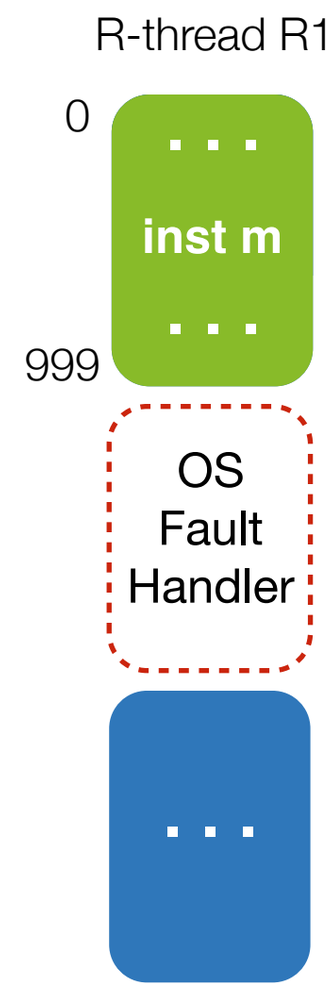


Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First	✔	✔	

Original Execution

Chunk Size Log

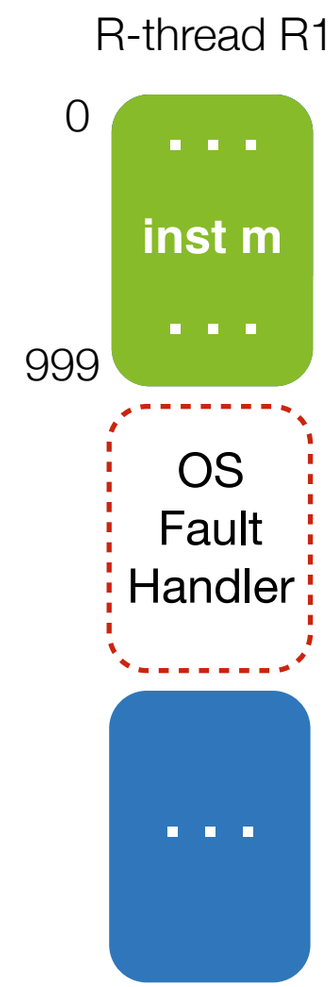


Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First	✔	✔	—

Original Execution

Chunk Size Log



Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First	✓	✓	—
Commit Now			

Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First	✔	✔	—
Commit Now			

Original Execution

R-thread R1

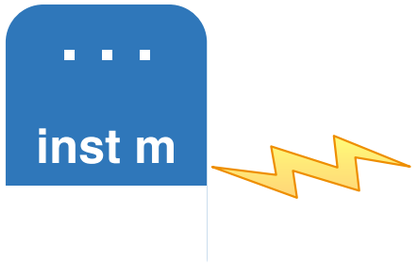
Chunk Size Log

Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First	✔	✔	—
Commit Now			

Original Execution

R-thread R1

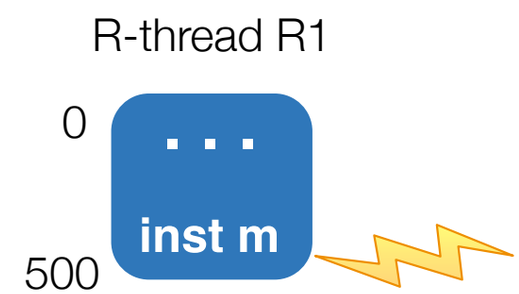


Chunk Size Log

Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First	✔	✔	—
Commit Now			

Original Execution

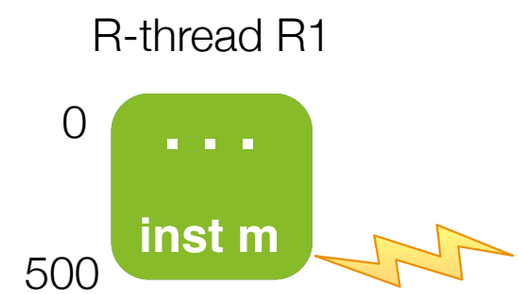


Chunk Size Log

Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First	✔	✔	—
Commit Now			

Original Execution

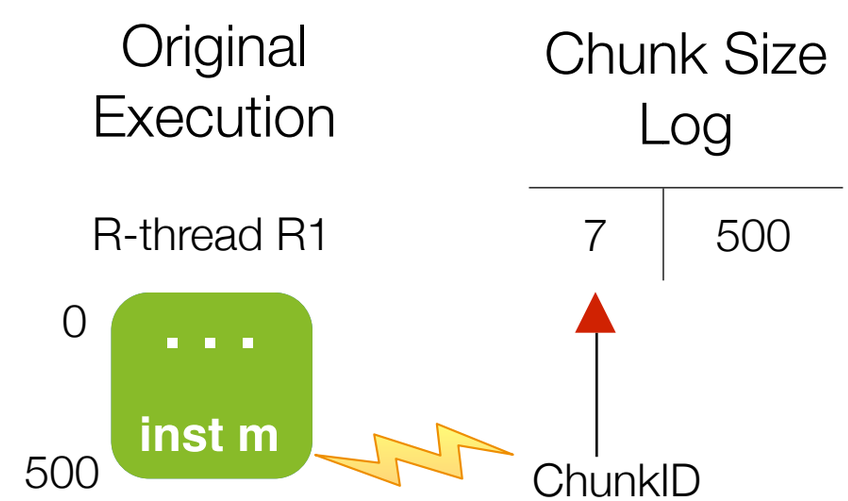


Chunk Size Log

7	500

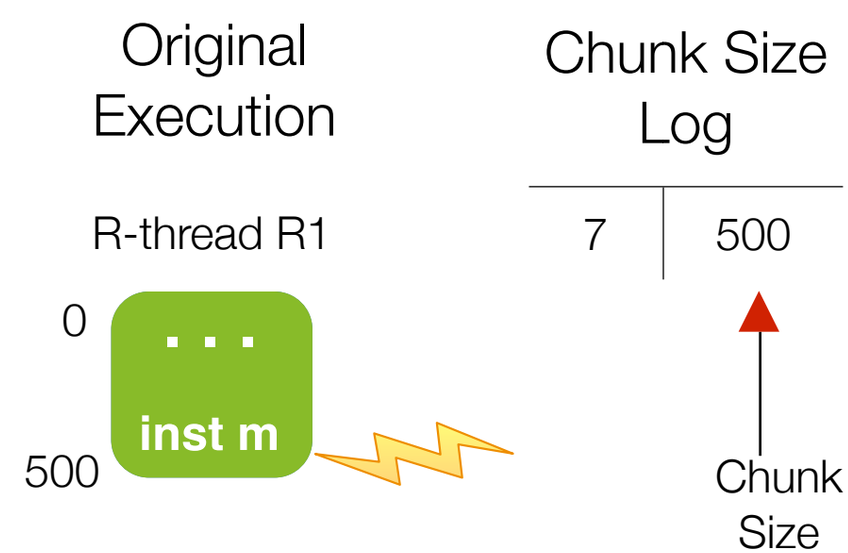
Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First	✓	✓	—
Commit Now			



Handling Interrupts: Three Approaches

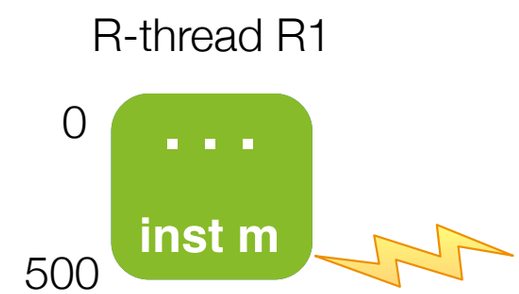
	Log Size	Wasted Work	Interrupt Latency
Finish First	✓	✓	—
Commit Now			



Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First	✔	✔	—
Commit Now			

Original Execution



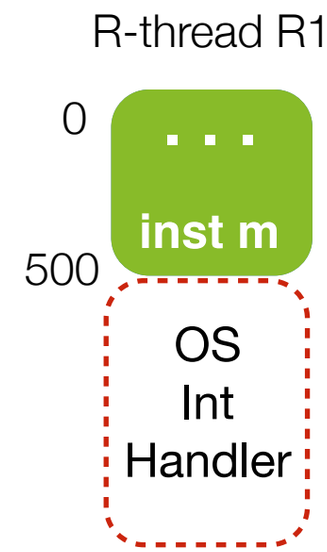
Chunk Size Log

7	500

Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First	✔	✔	—
Commit Now			

Original Execution



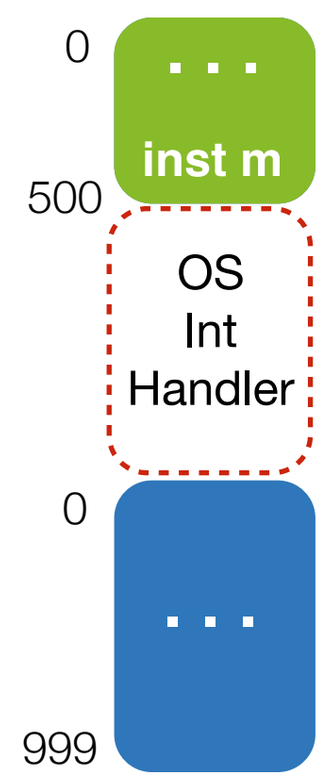
Chunk Size Log

7	500

Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First	✔	✔	—
Commit Now			

Original Execution



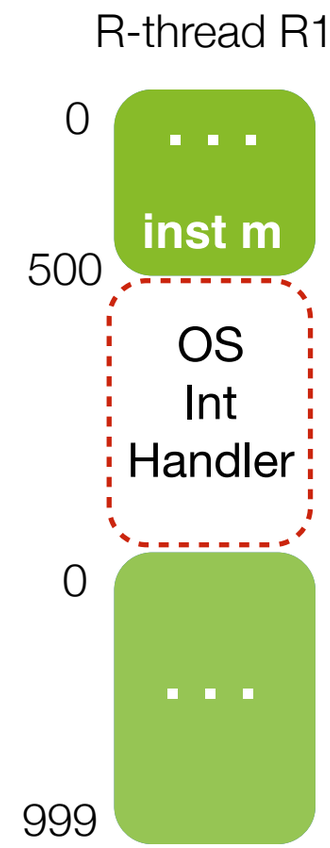
Chunk Size Log

7	500

Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First	✔	✔	—
Commit Now			

Original Execution

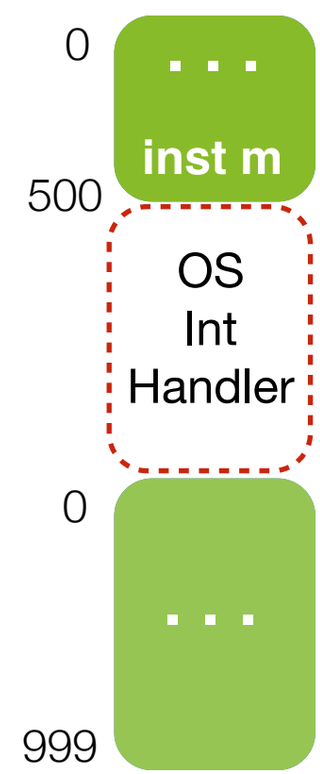


Chunk Size Log	
7	500

Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First	✔	✔	—
Commit Now	—		

Original Execution

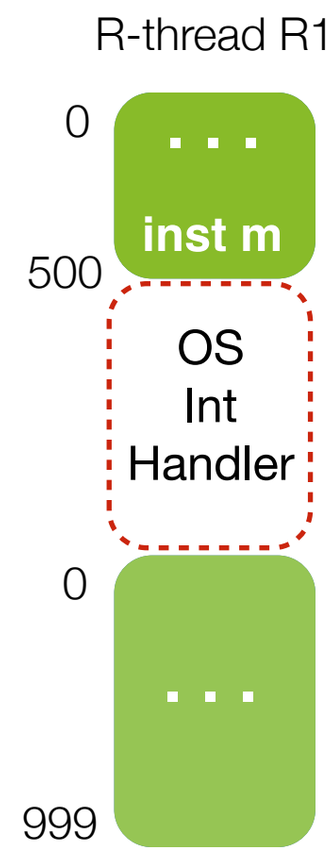


Chunk Size Log	
7	500

Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First	✓	✓	—
Commit Now	—	✓	

Original Execution

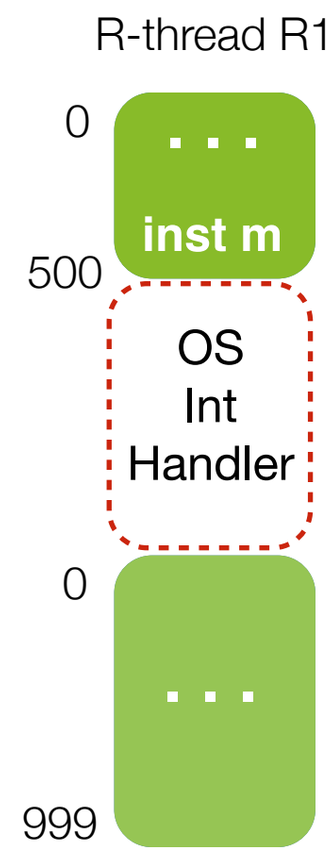


Chunk Size Log	
7	500

Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First	✓	✓	—
Commit Now	—	✓	✓

Original Execution



Chunk Size Log

7	500

Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First	✓	✓	—
Commit Now	—	✓	✓
Squash Now			

Handling Interrupts: Three Approaches

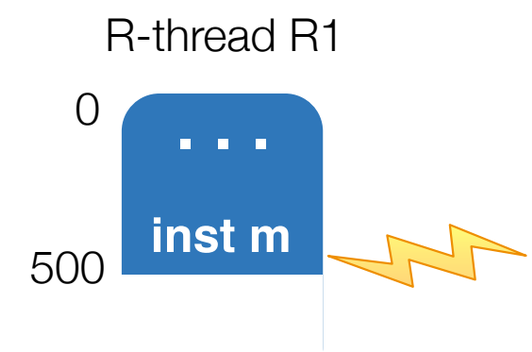
	Log Size	Wasted Work	Interrupt Latency
Finish First	✓	✓	—
Commit Now	—	✓	✓
Squash Now			

Original Execution	Chunk Size Log
R-thread R1	

Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First	✓	✓	—
Commit Now	—	✓	✓
Squash Now			

Original Execution

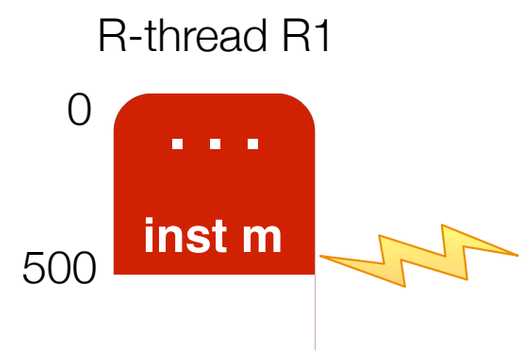


Chunk Size Log

Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First	✓	✓	–
Commit Now	–	✓	✓
Squash Now			

Original Execution



Chunk Size Log

Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First	✓	✓	—
Commit Now	—	✓	✓
Squash Now			

Original Execution

R-thread R1

Chunk Size Log

Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First	✓	✓	—
Commit Now	—	✓	✓
Squash Now			

Original Execution

R-thread R1

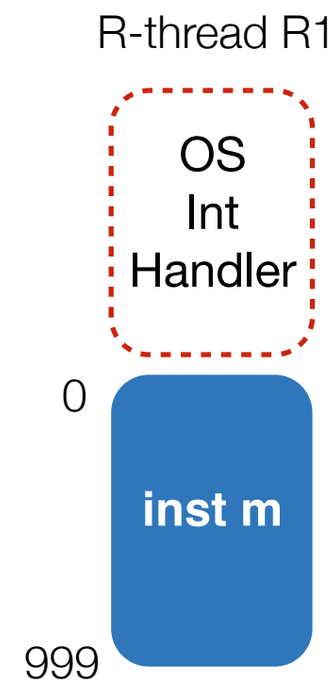
OS Int Handler

Chunk Size Log

Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First	✓	✓	—
Commit Now	—	✓	✓
Squash Now			

Original Execution

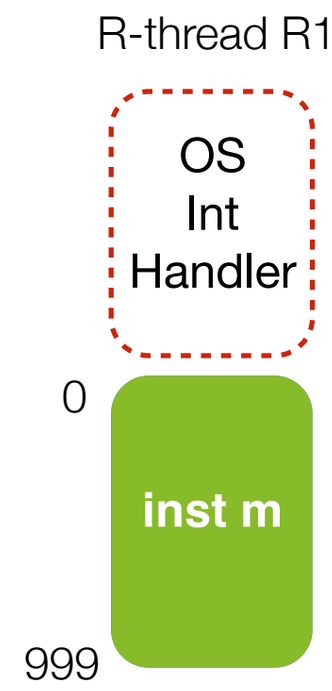


Chunk Size Log

Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First	✓	✓	–
Commit Now	–	✓	✓
Squash Now			

Original Execution

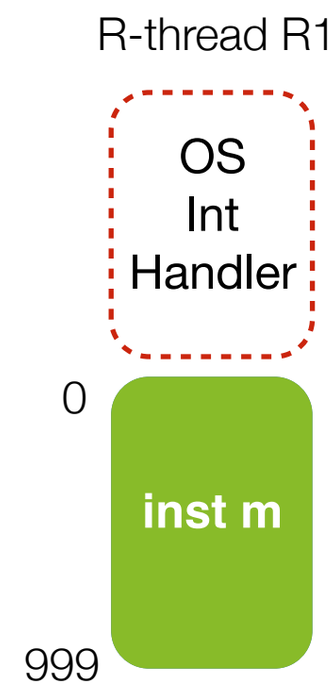


Chunk Size Log

Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First	✓	✓	—
Commit Now	—	✓	✓
Squash Now	✓		

Original Execution

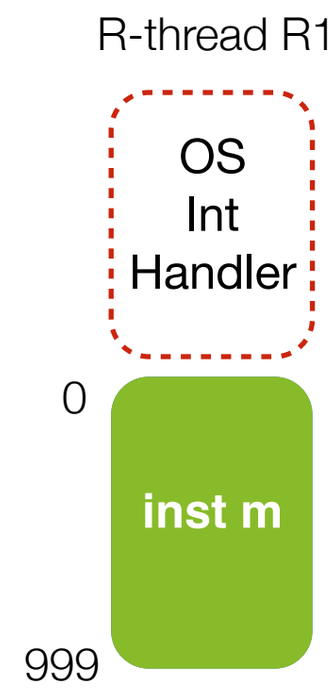


Chunk Size Log

Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First	✓	✓	–
Commit Now	–	✓	✓
Squash Now	✓	–	

Original Execution

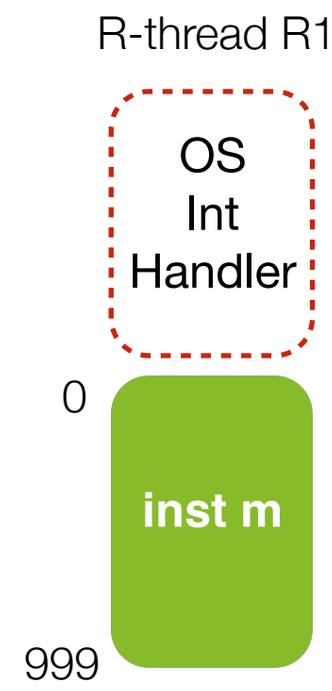


Chunk Size Log

Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First	✓	✓	—
Commit Now	—	✓	✓
Squash Now	✓	—	✓

Original Execution

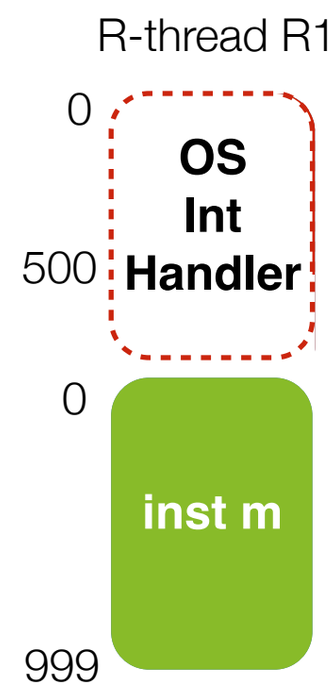


Chunk Size Log

Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency
Finish First	✓	✓	—
Commit Now	—	✓	✓
Squash Now	✓	—	✓

Original Execution



Chunk Size Log

Handling Interrupts: Three Approaches

	Log Size	Wasted Work	Interrupt Latency	Original Execution	Chunk Size Log
Finis					
Com					
Squa					

Highly non-deterministic events, such as interrupts, can be treated as deterministic events

CapoOne uses SquashNow: easy to implement and little overhead

Handling Faults

Handling Faults

- Commit all instructions up to faulting one and log chunk size

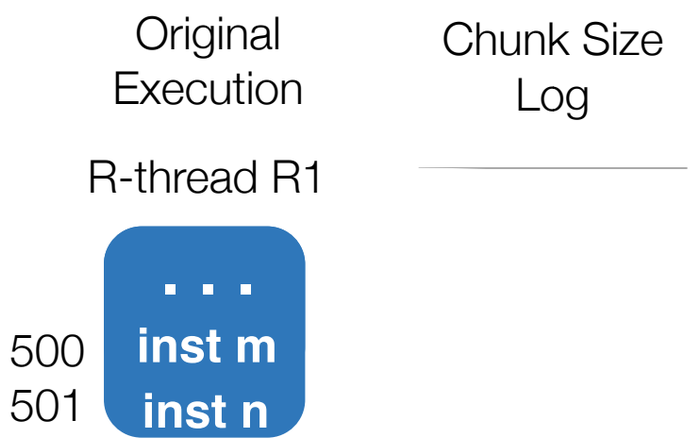
Handling Faults

- Commit all instructions up to faulting one and log chunk size

Original Execution	Chunk Size Log
R-thread R1	

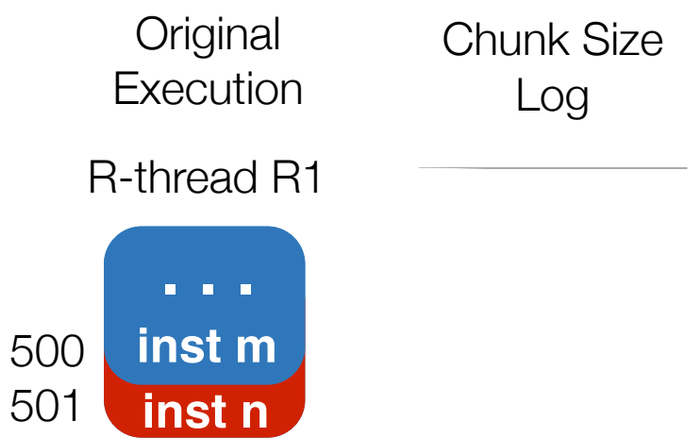
Handling Faults

- Commit all instructions up to faulting one and log chunk size



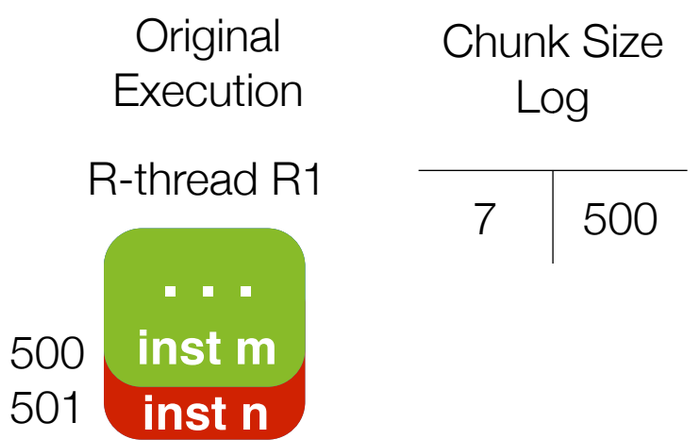
Handling Faults

- Commit all instructions up to faulting one and log chunk size



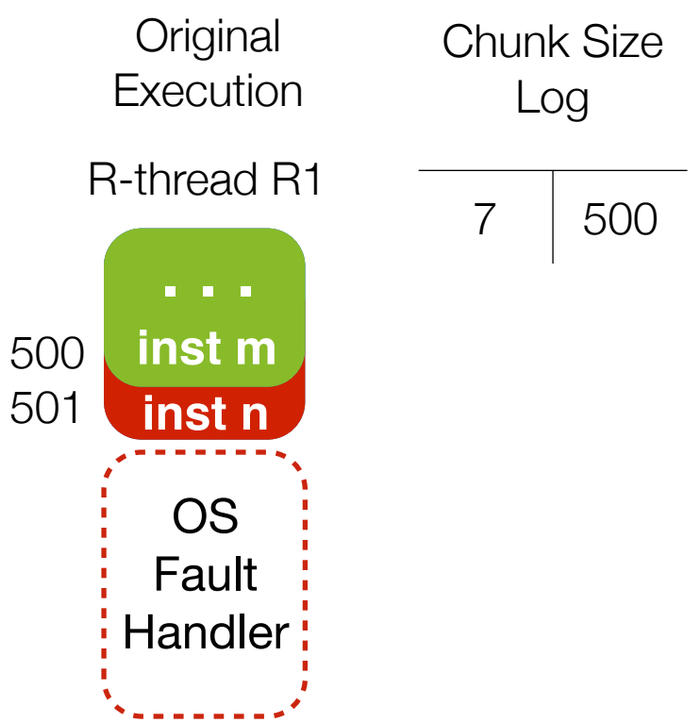
Handling Faults

- Commit all instructions up to faulting one and log chunk size



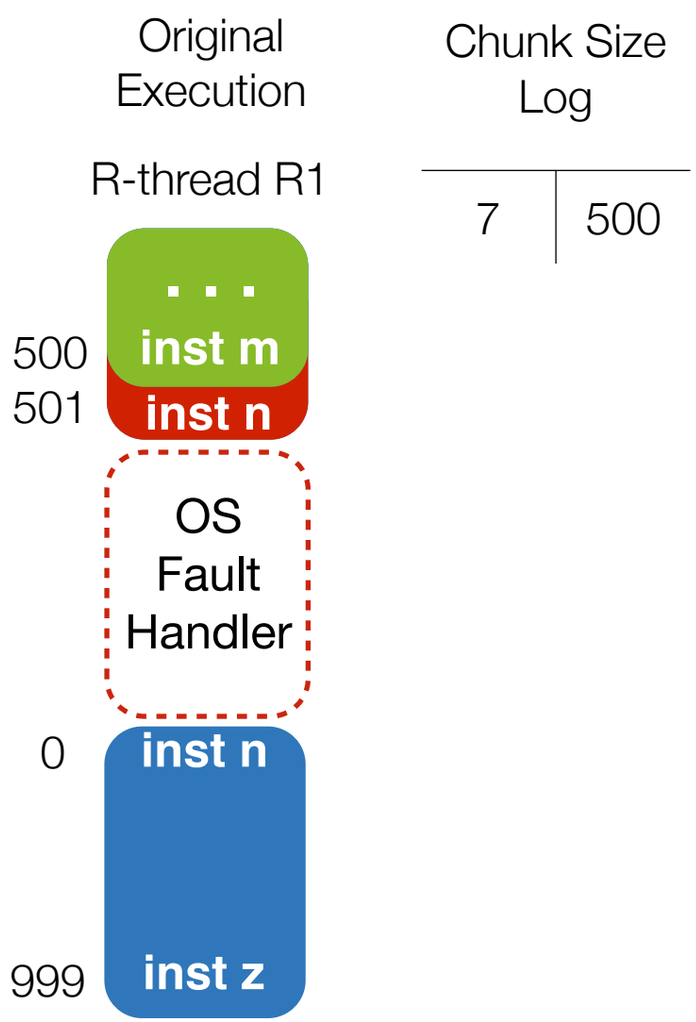
Handling Faults

- Commit all instructions up to faulting one and log chunk size



Handling Faults

- Commit all instructions up to faulting one and log chunk size



Chunk Size Log

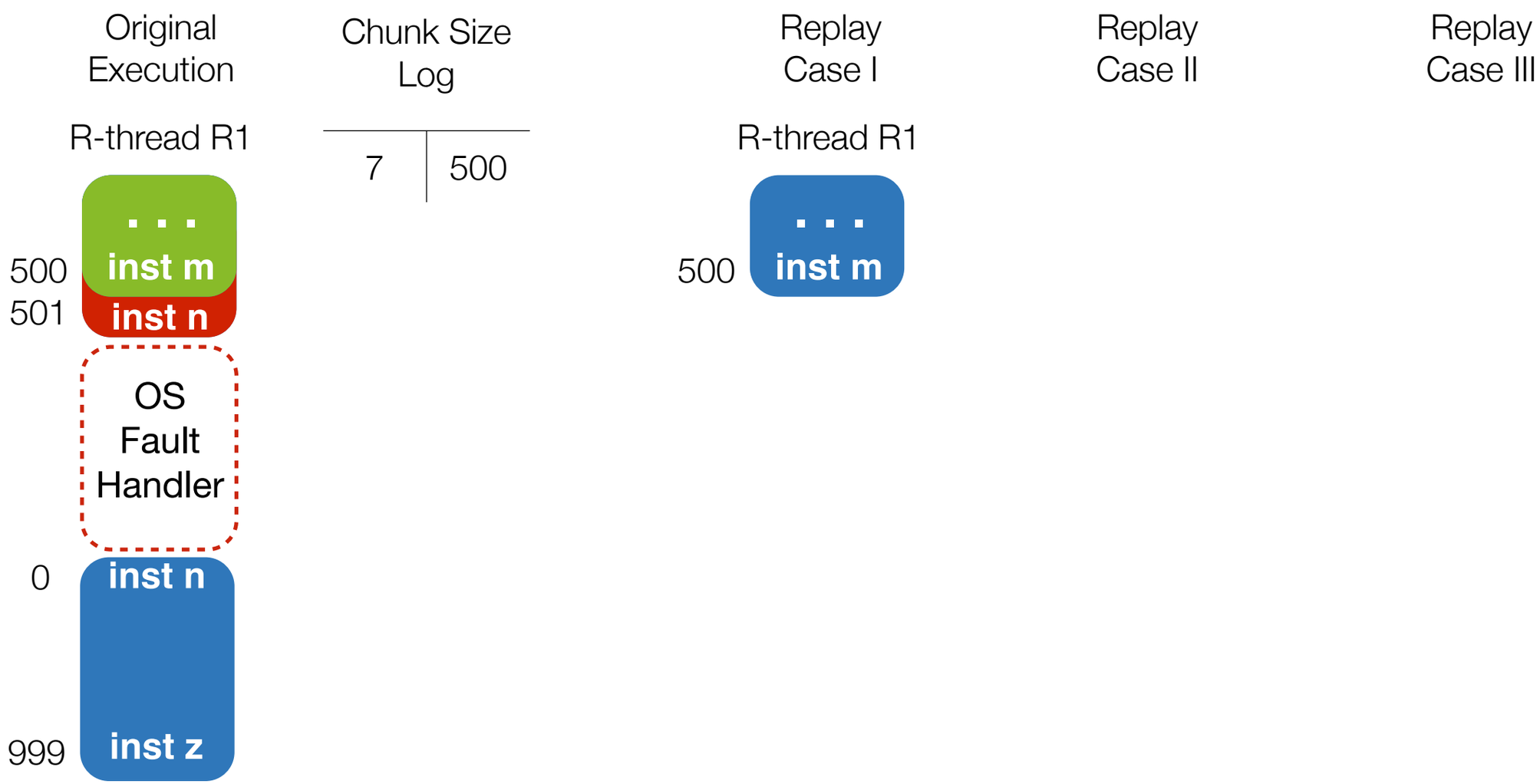
7	500

Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky

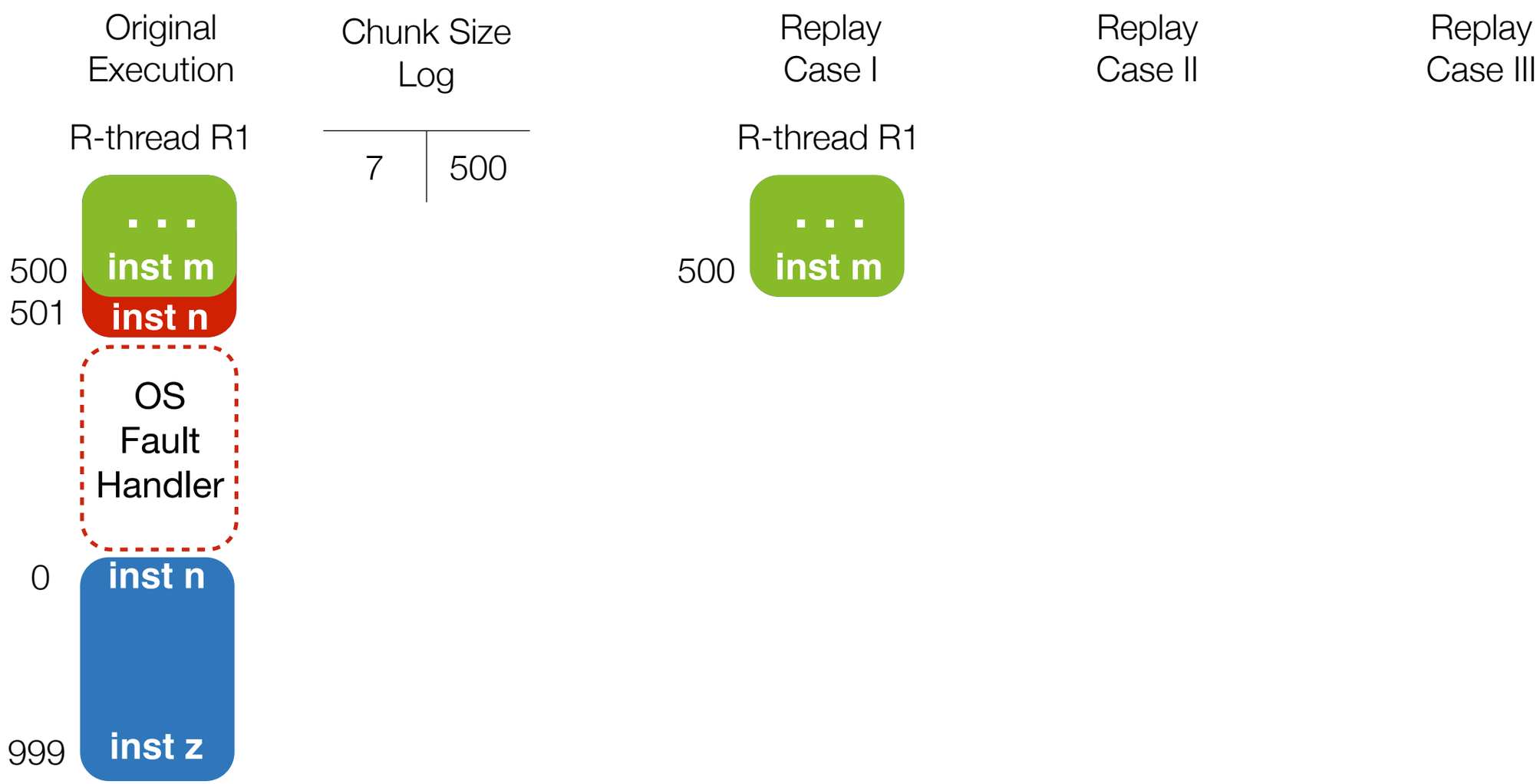
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky



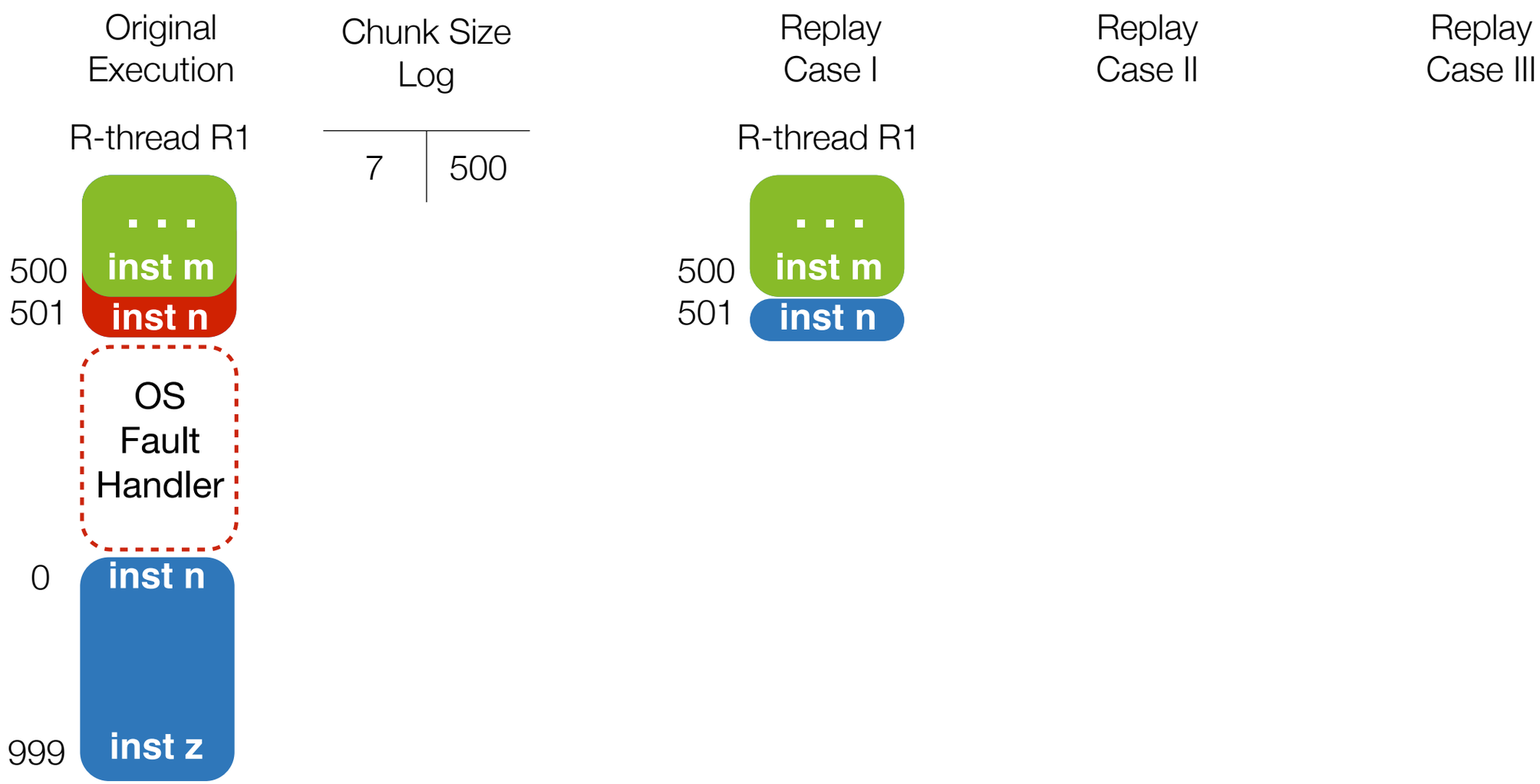
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky



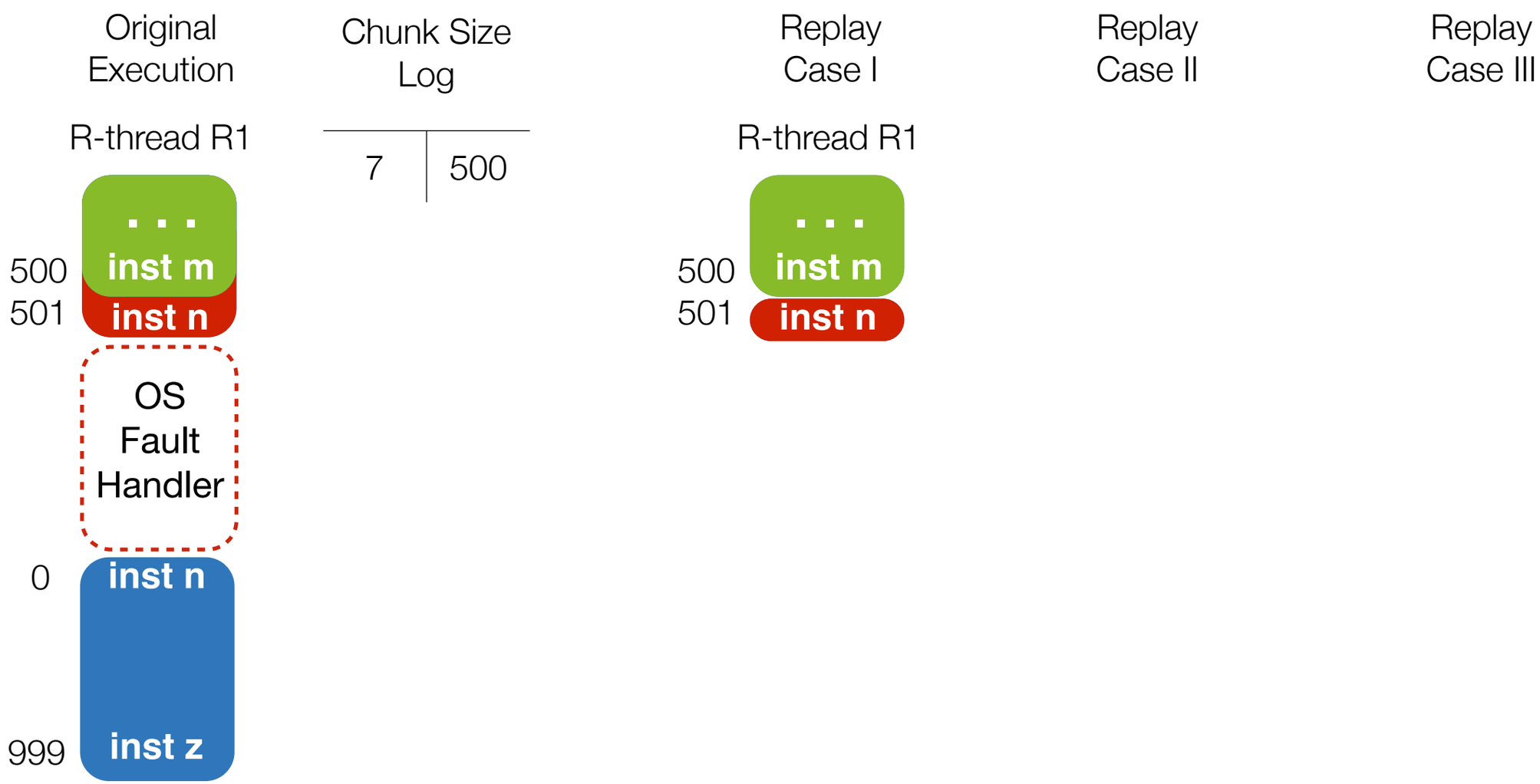
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky



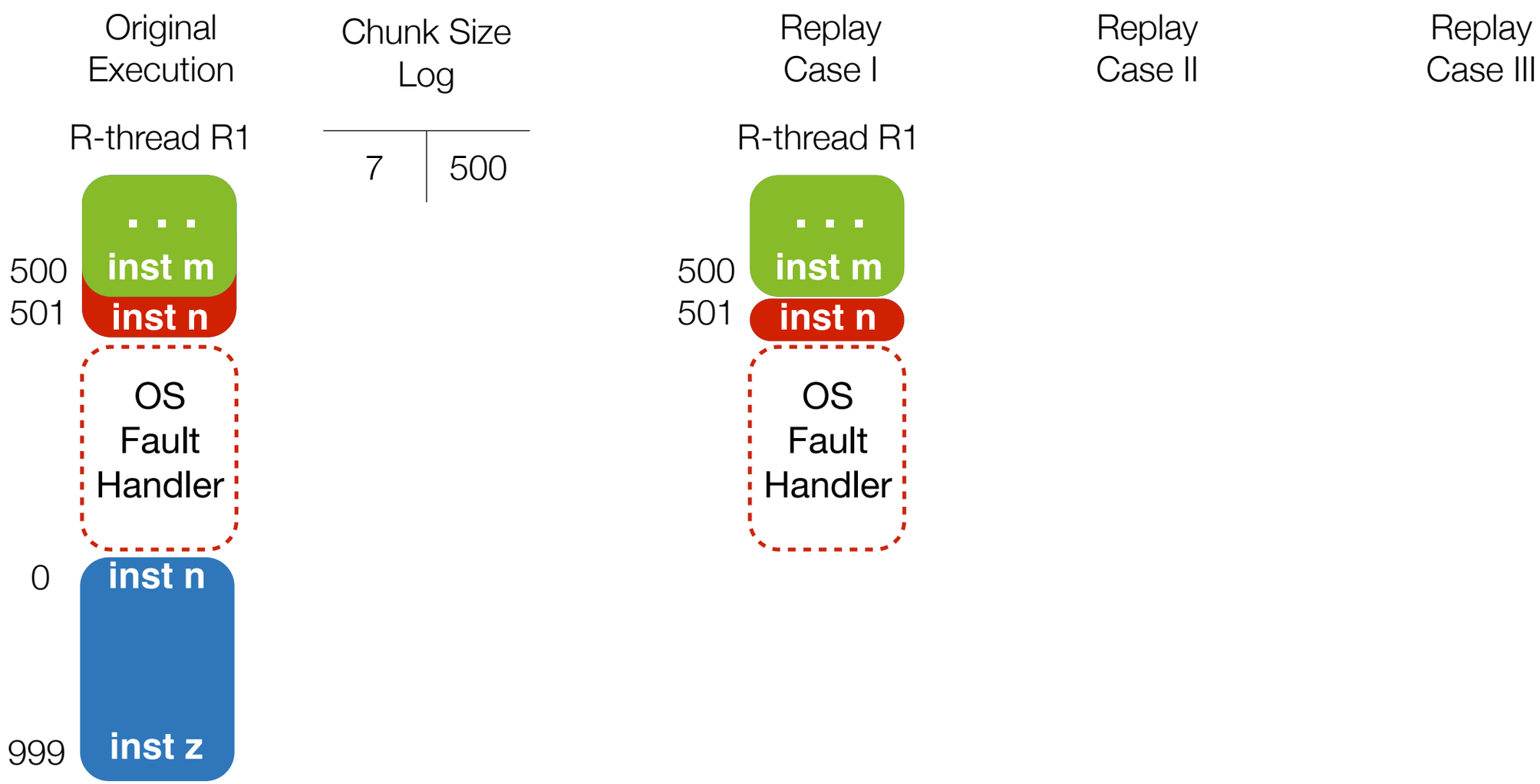
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky



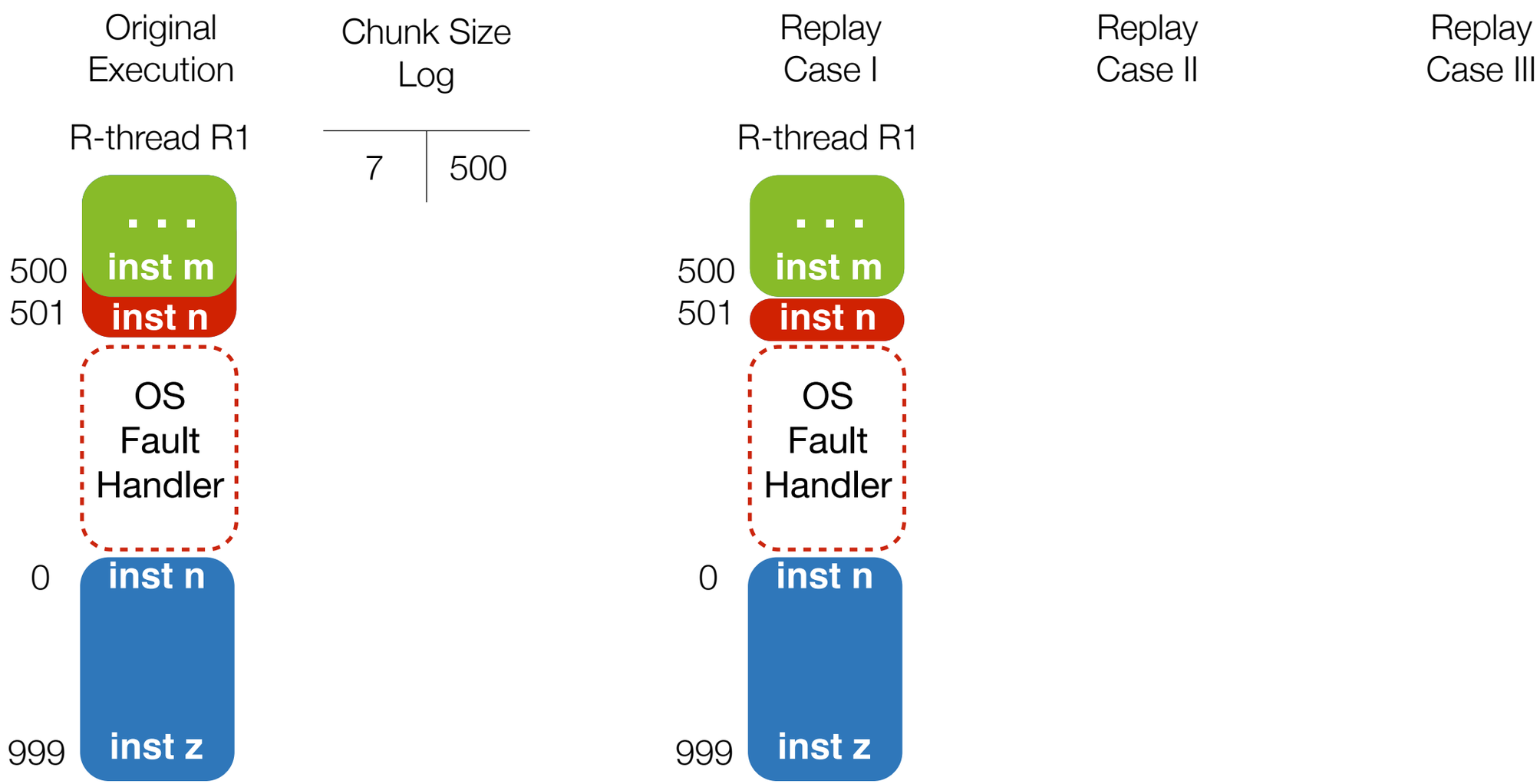
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky



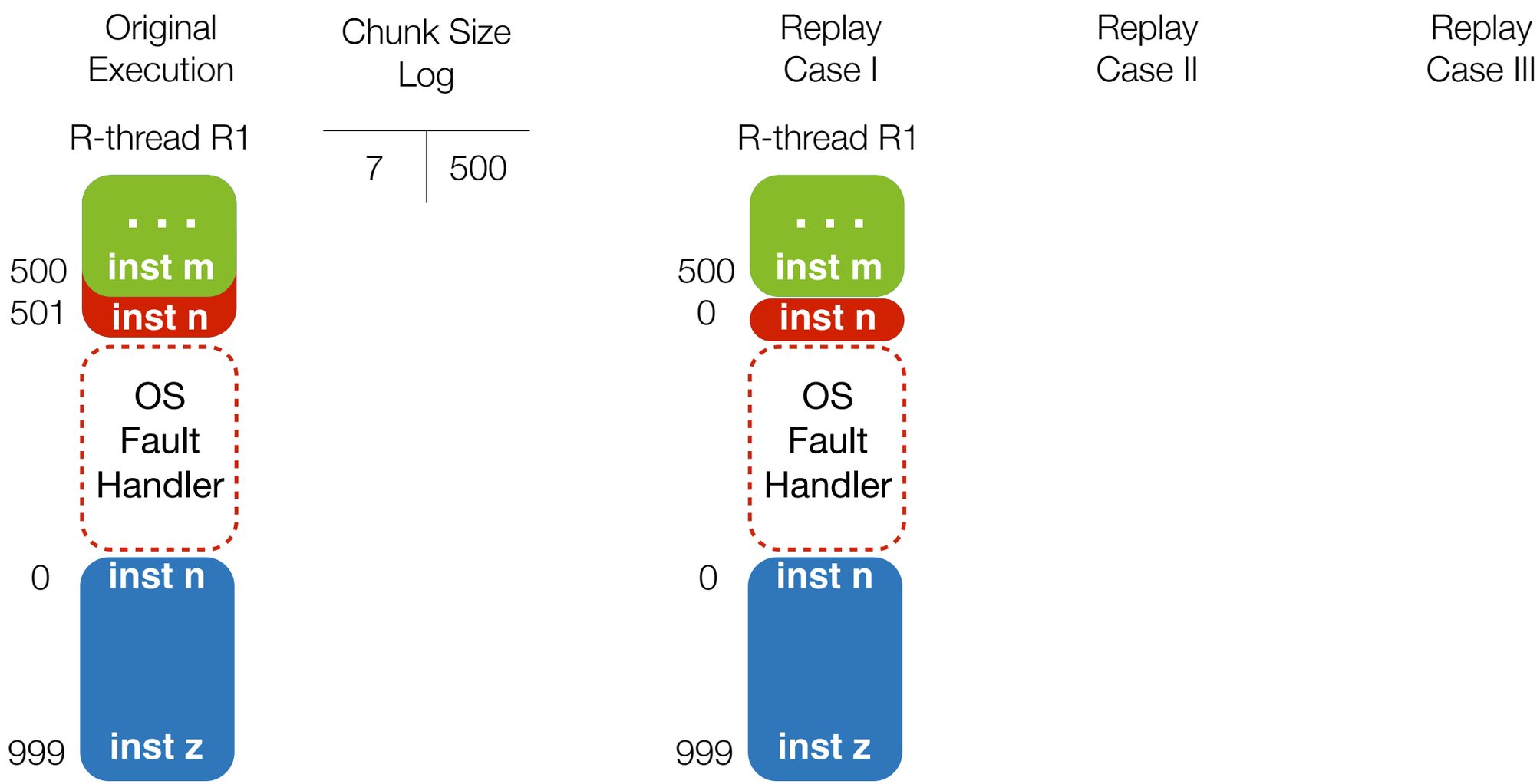
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky



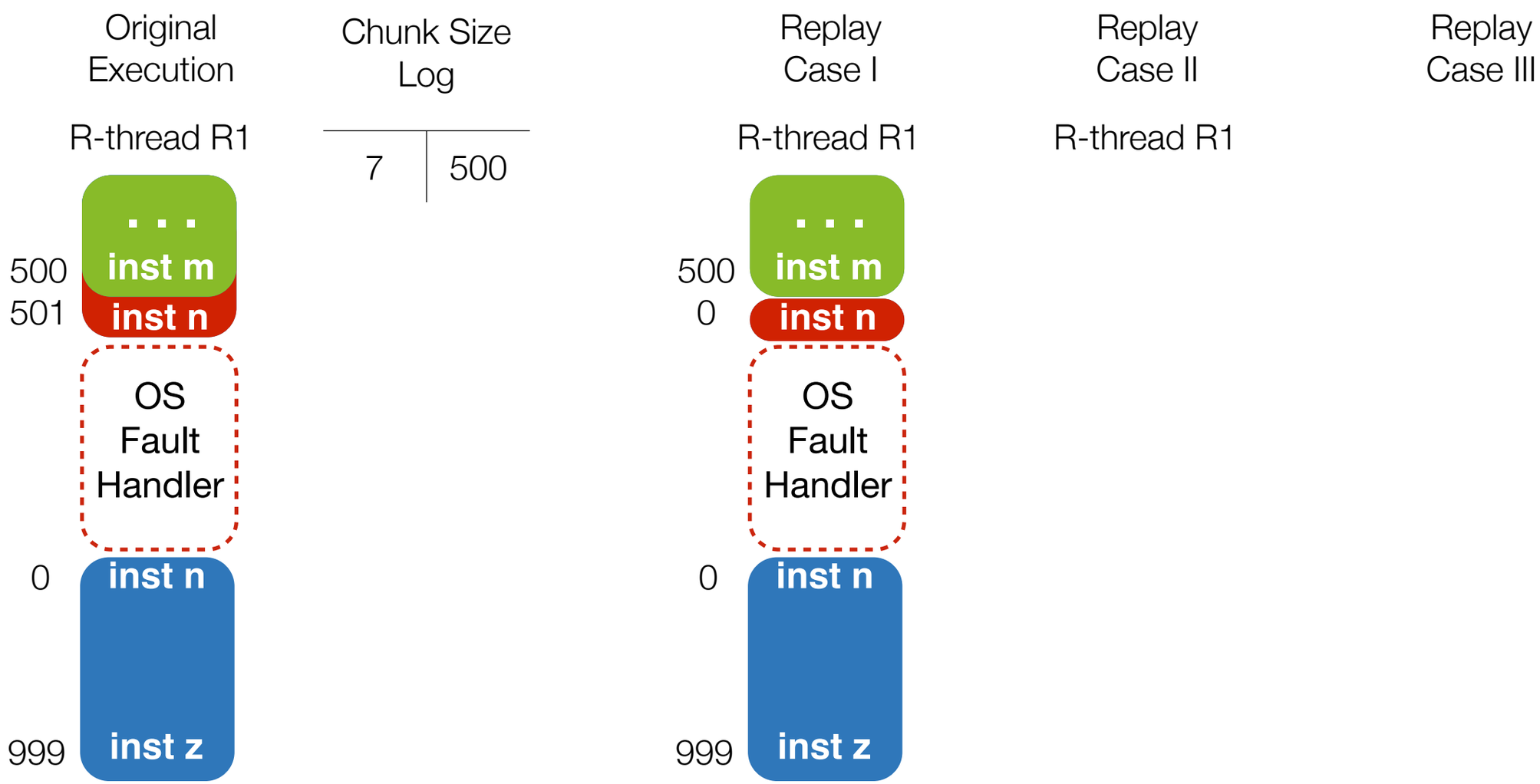
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky



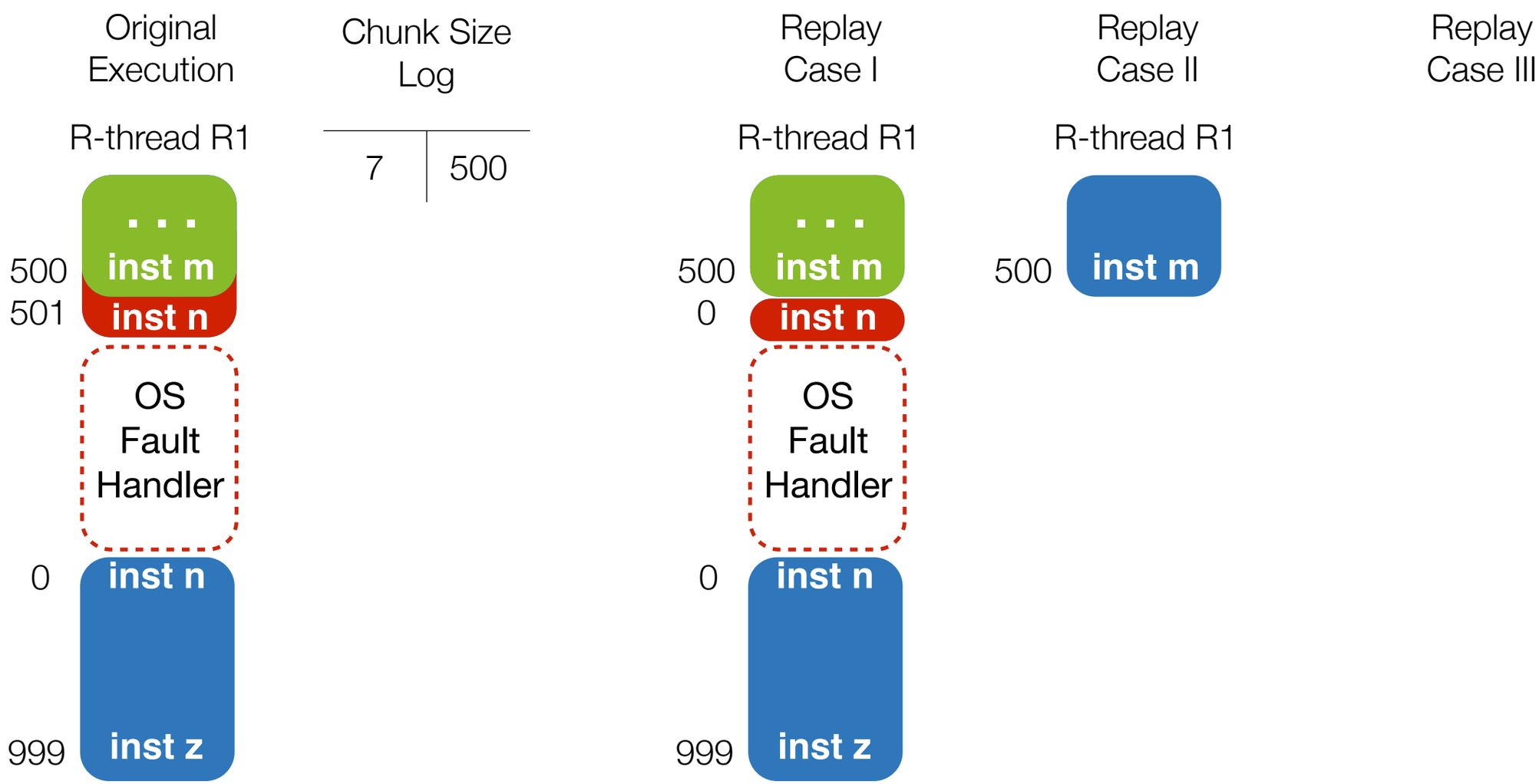
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky



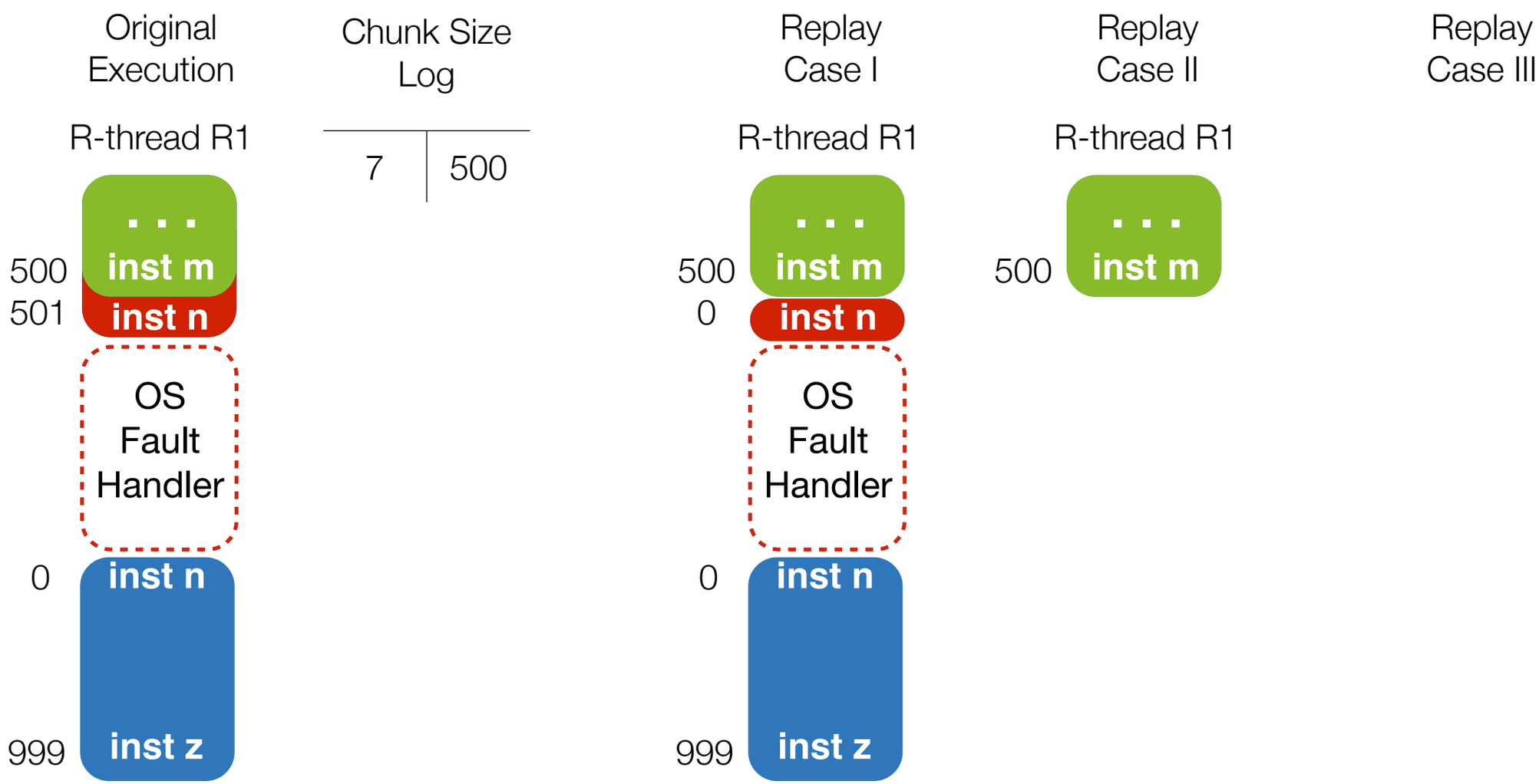
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky



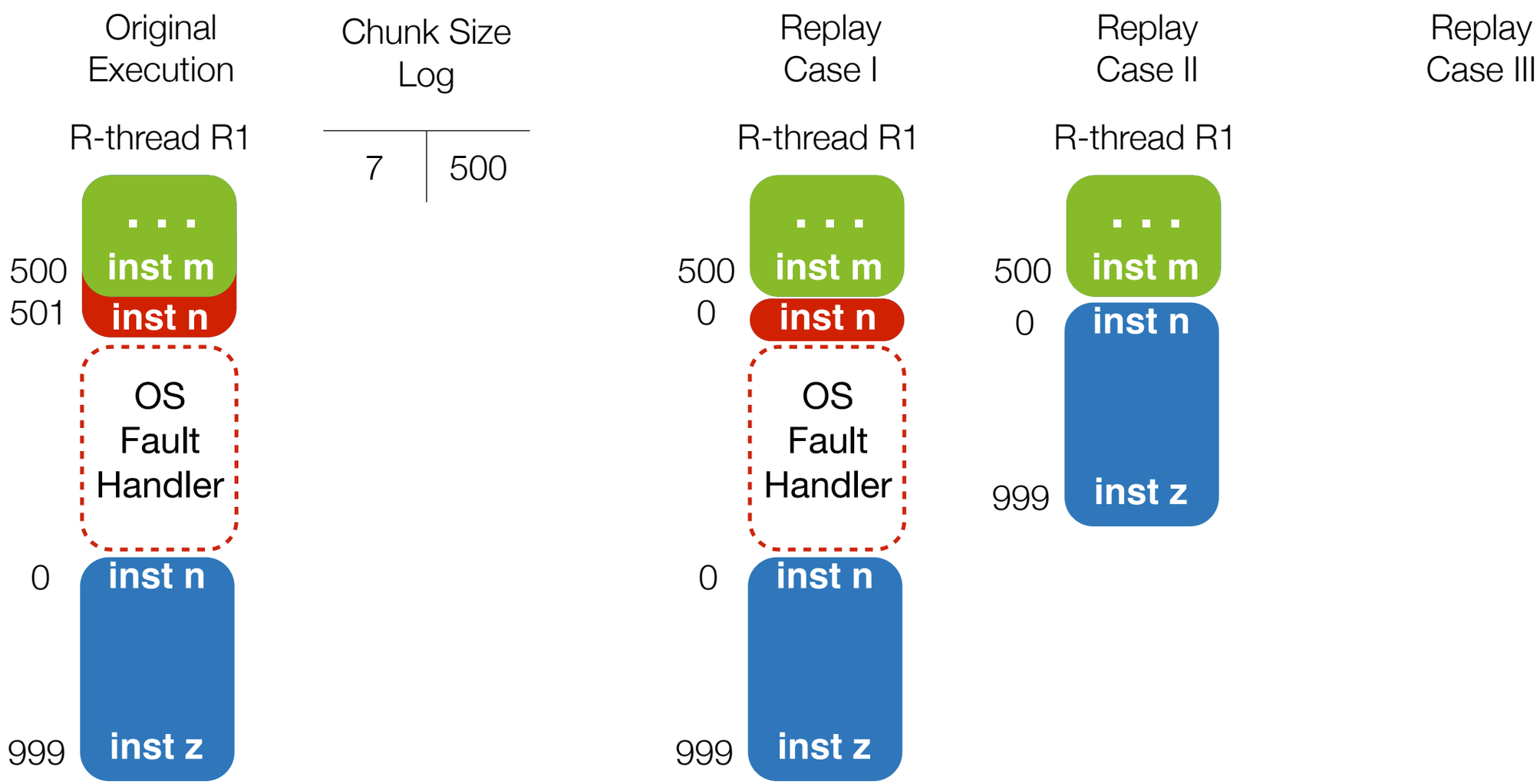
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky



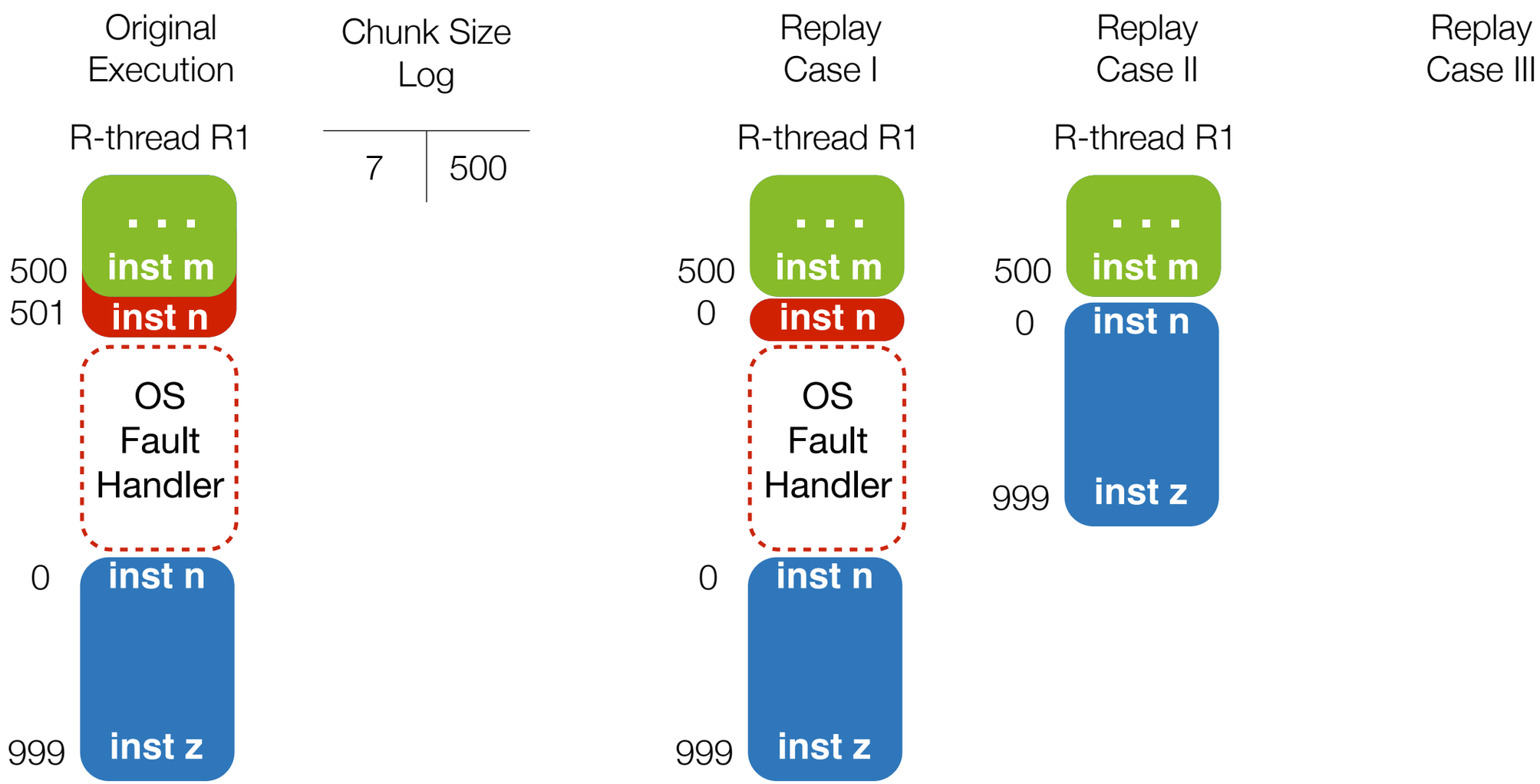
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky



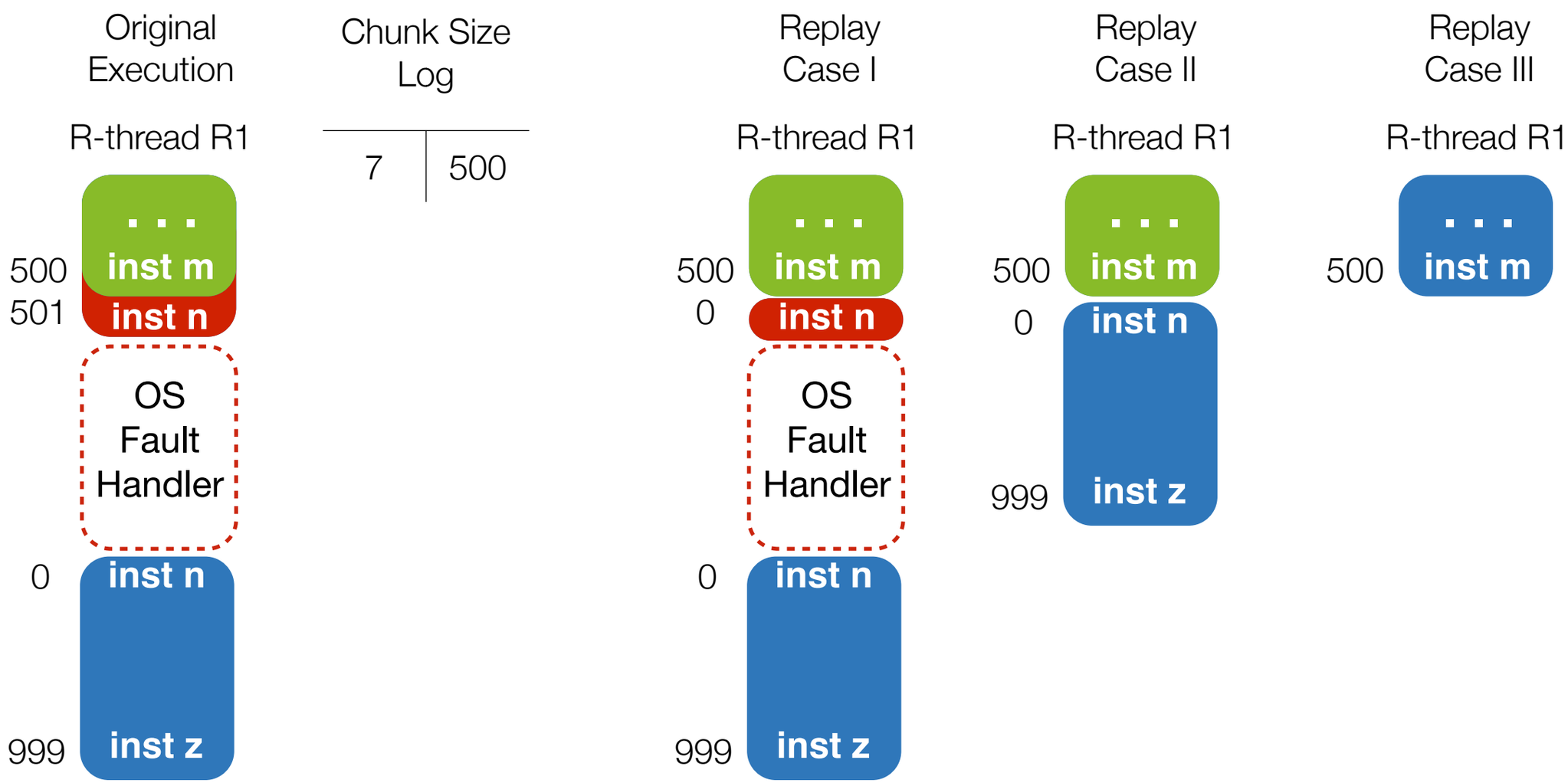
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky



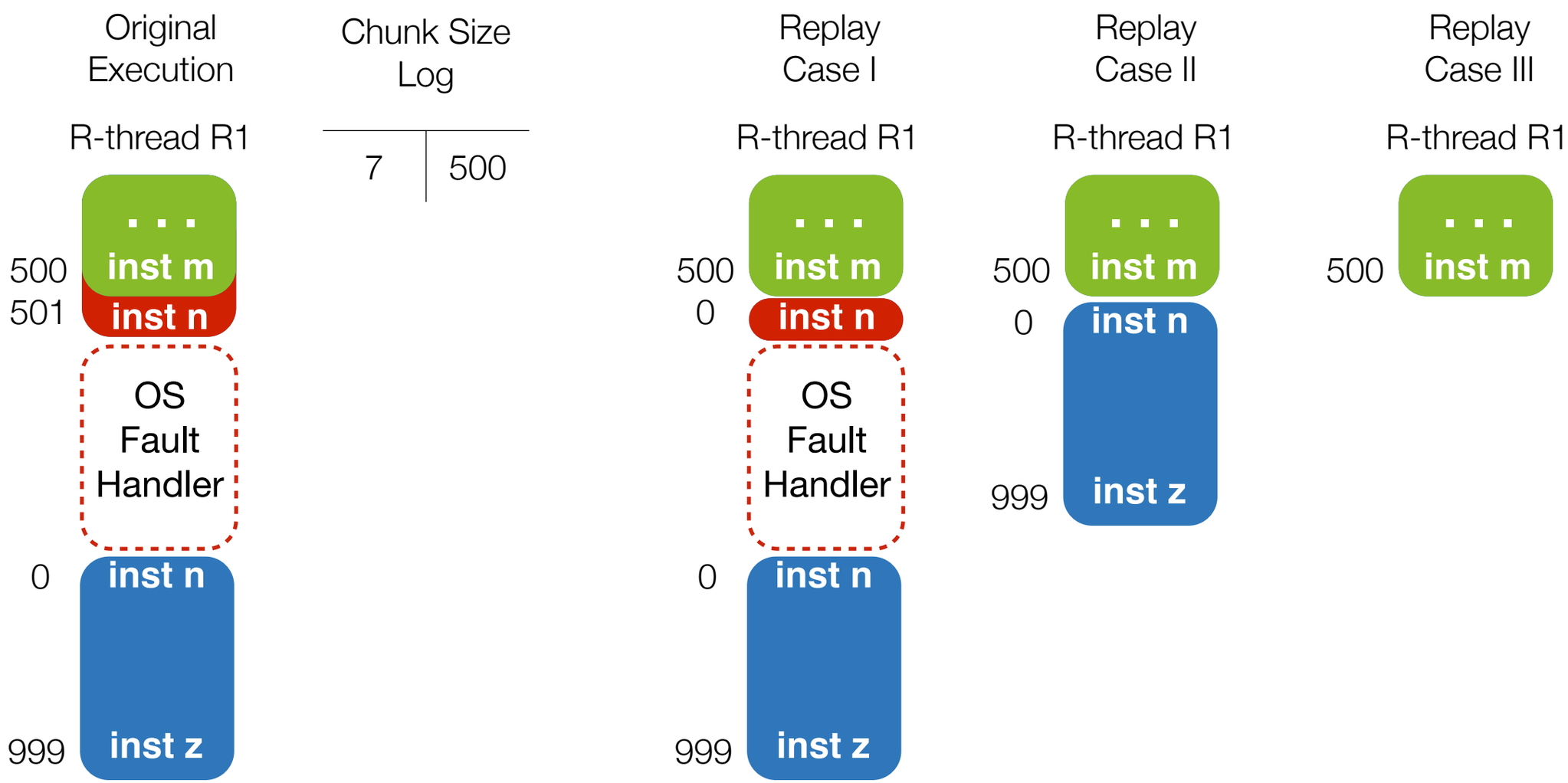
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky



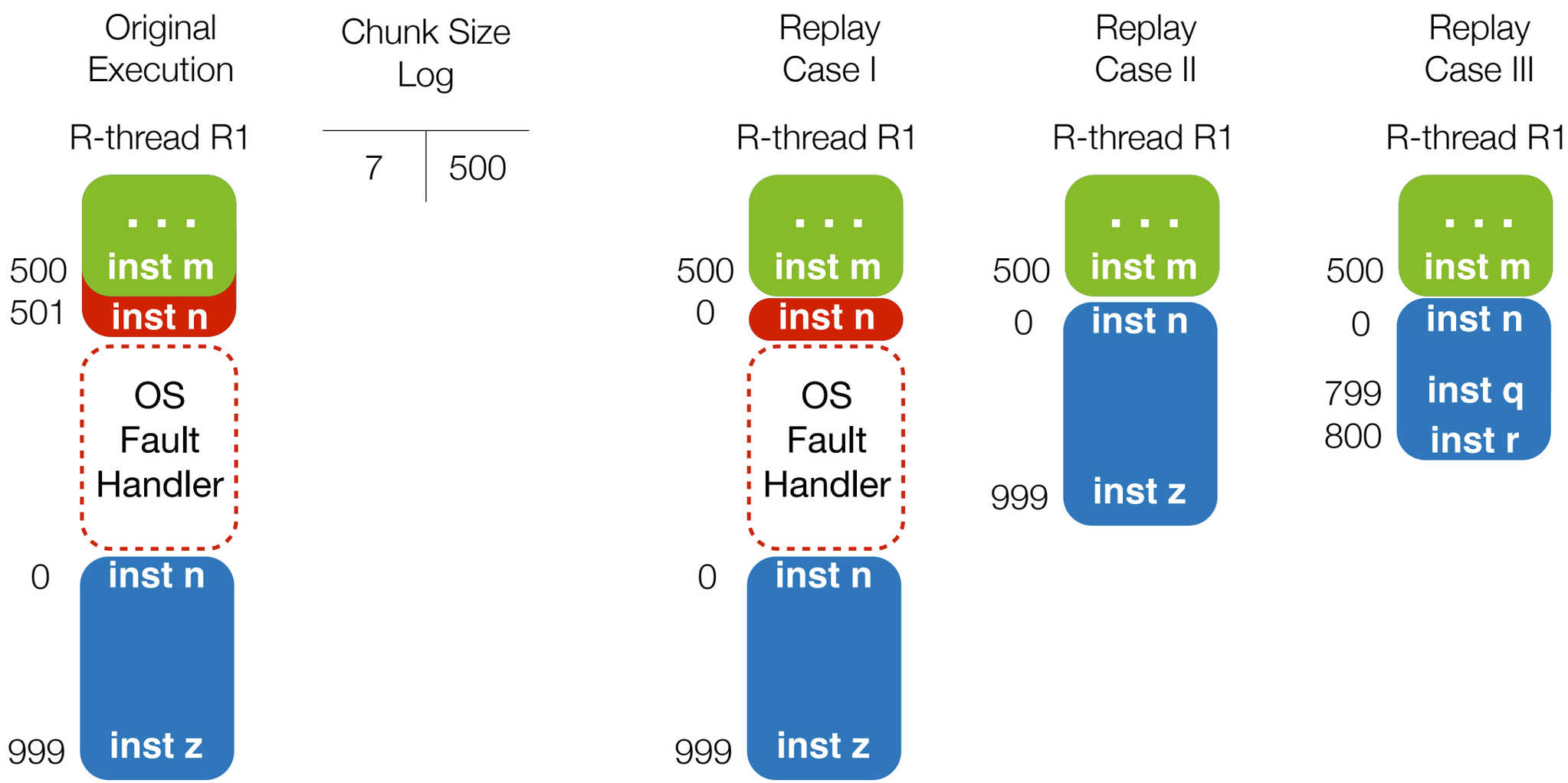
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky



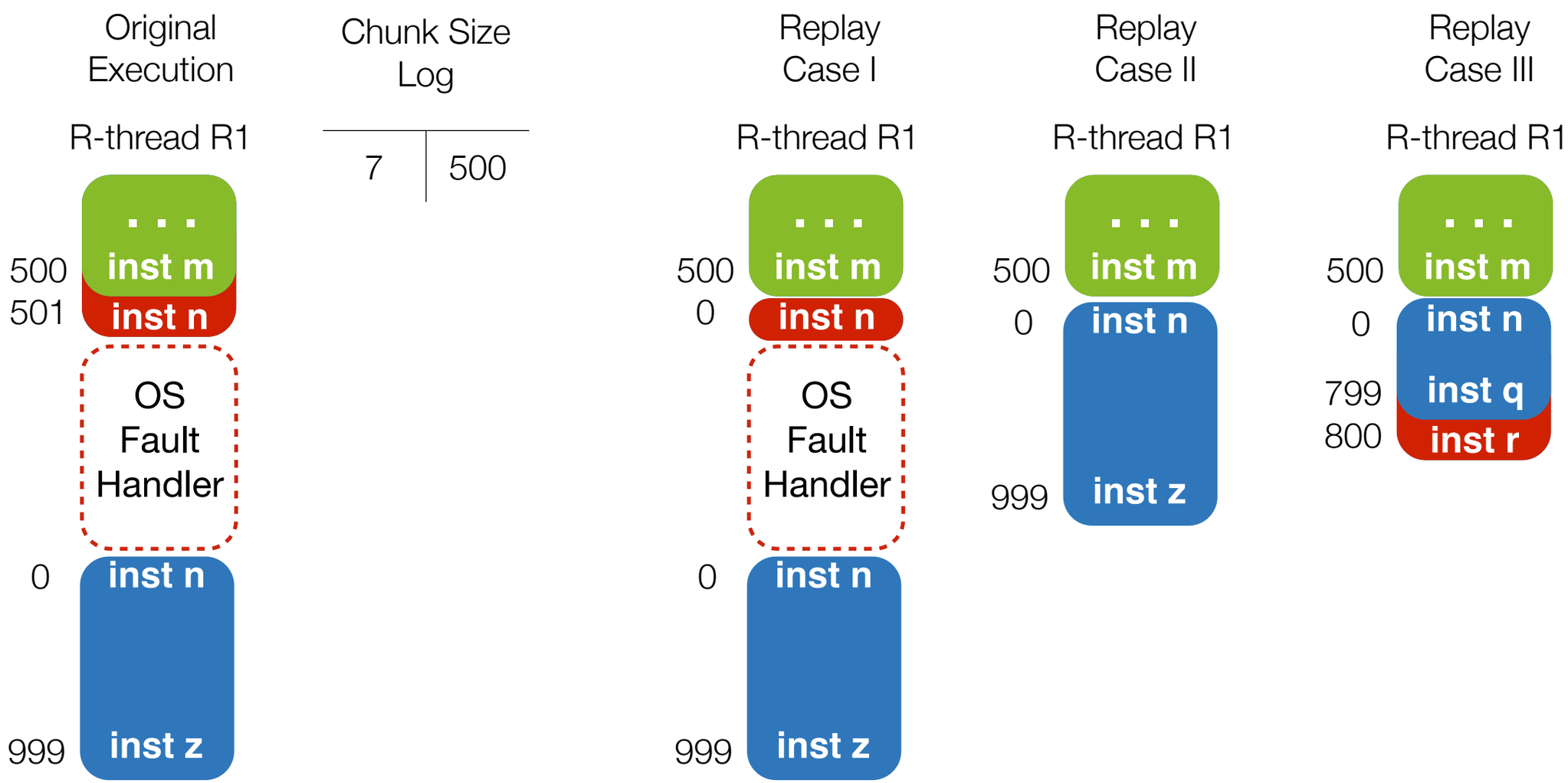
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky



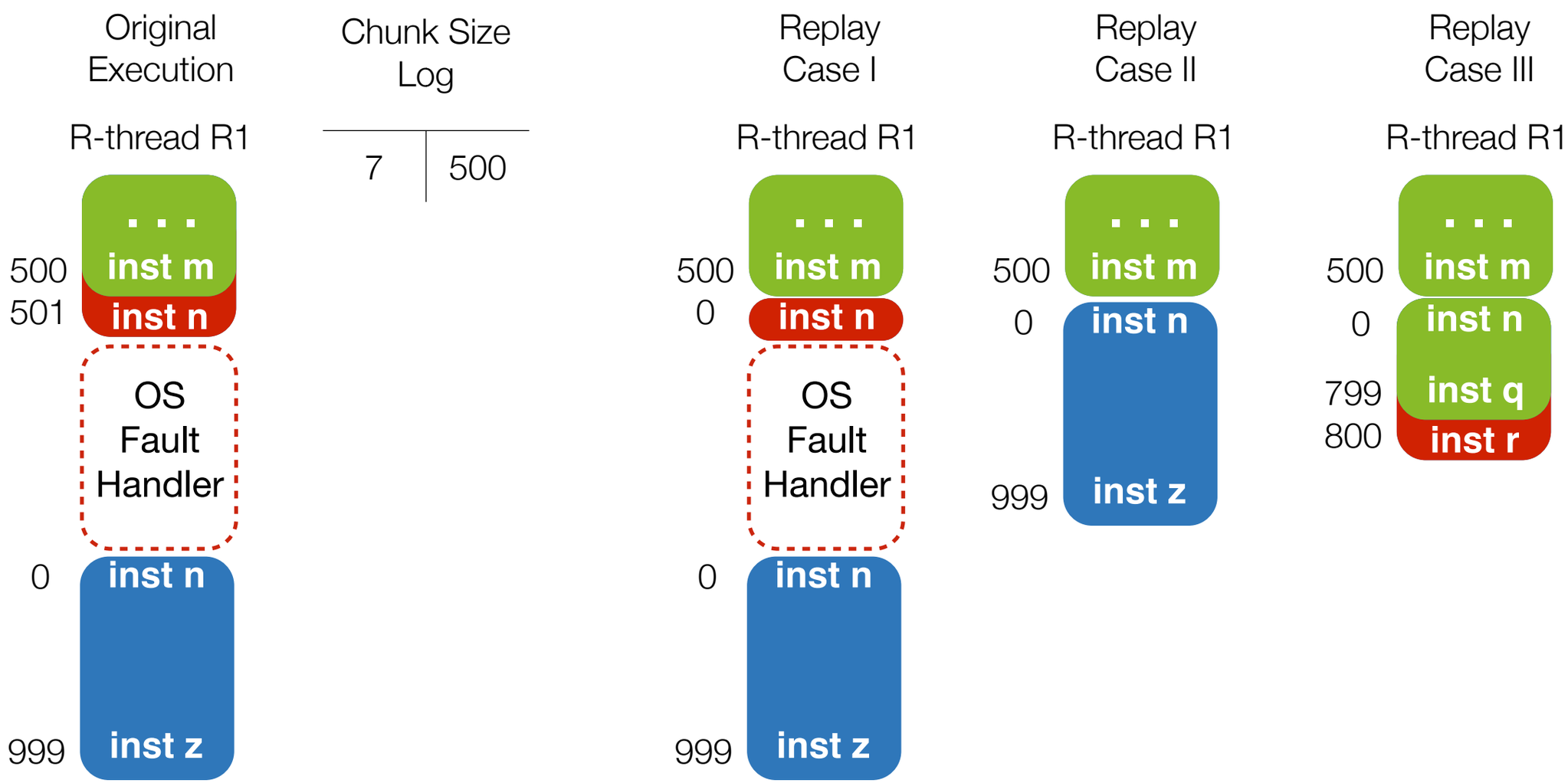
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky



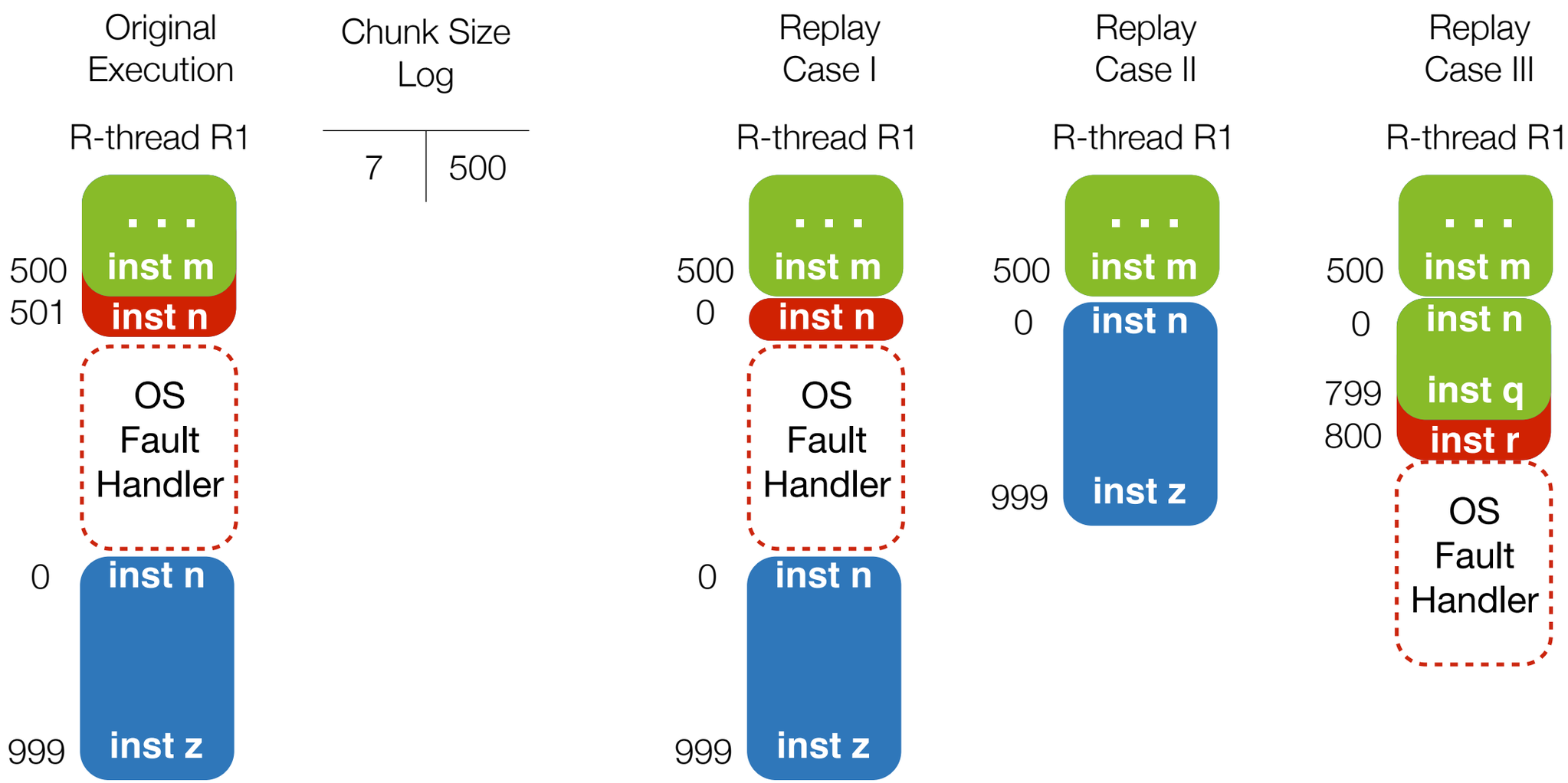
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky



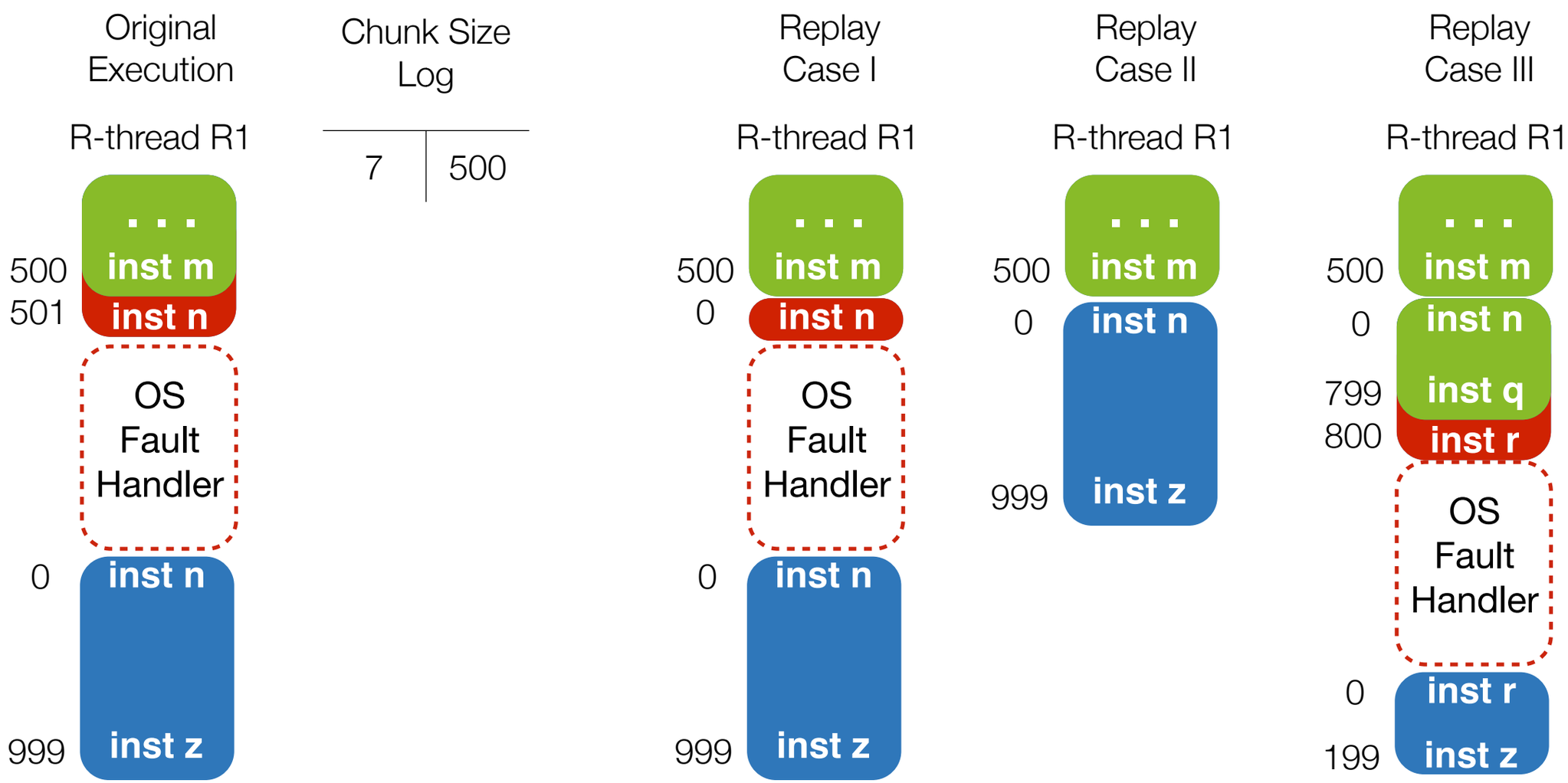
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky



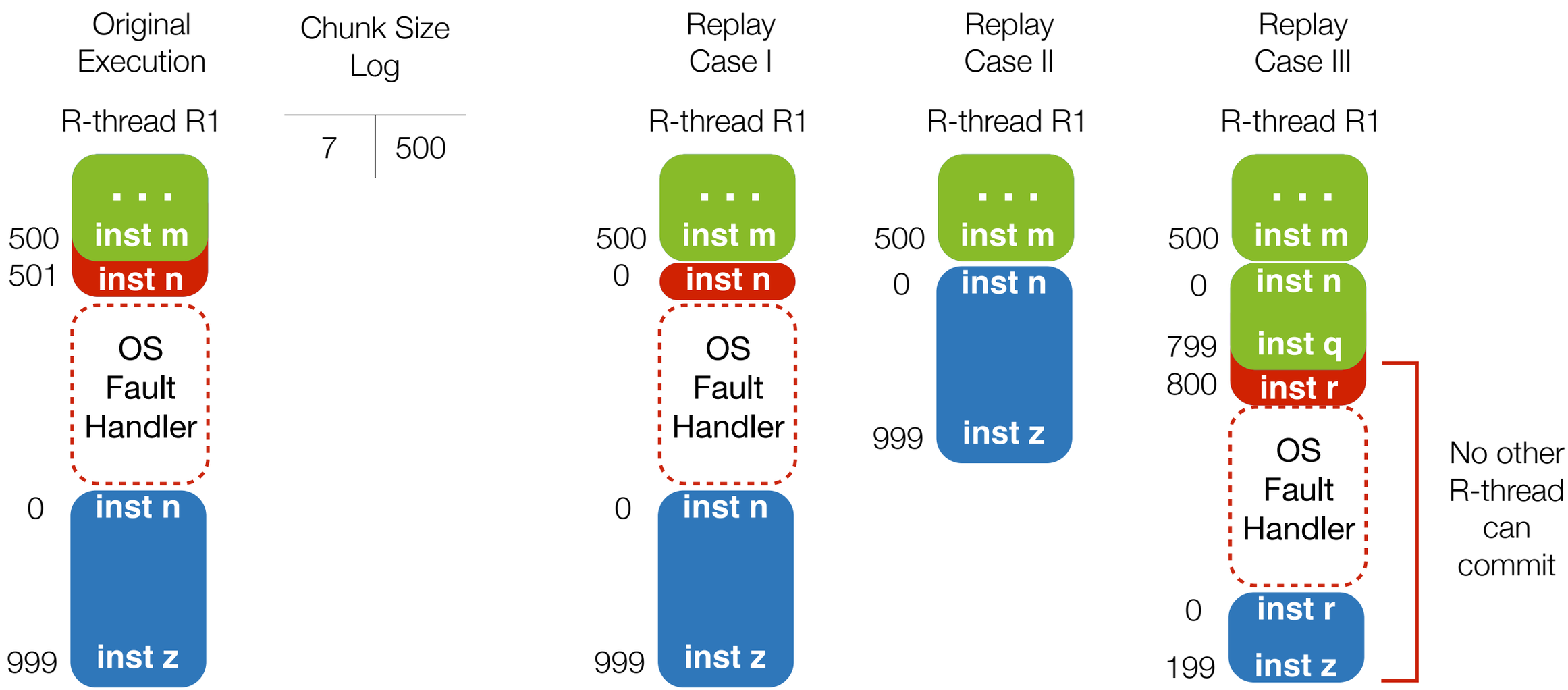
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky



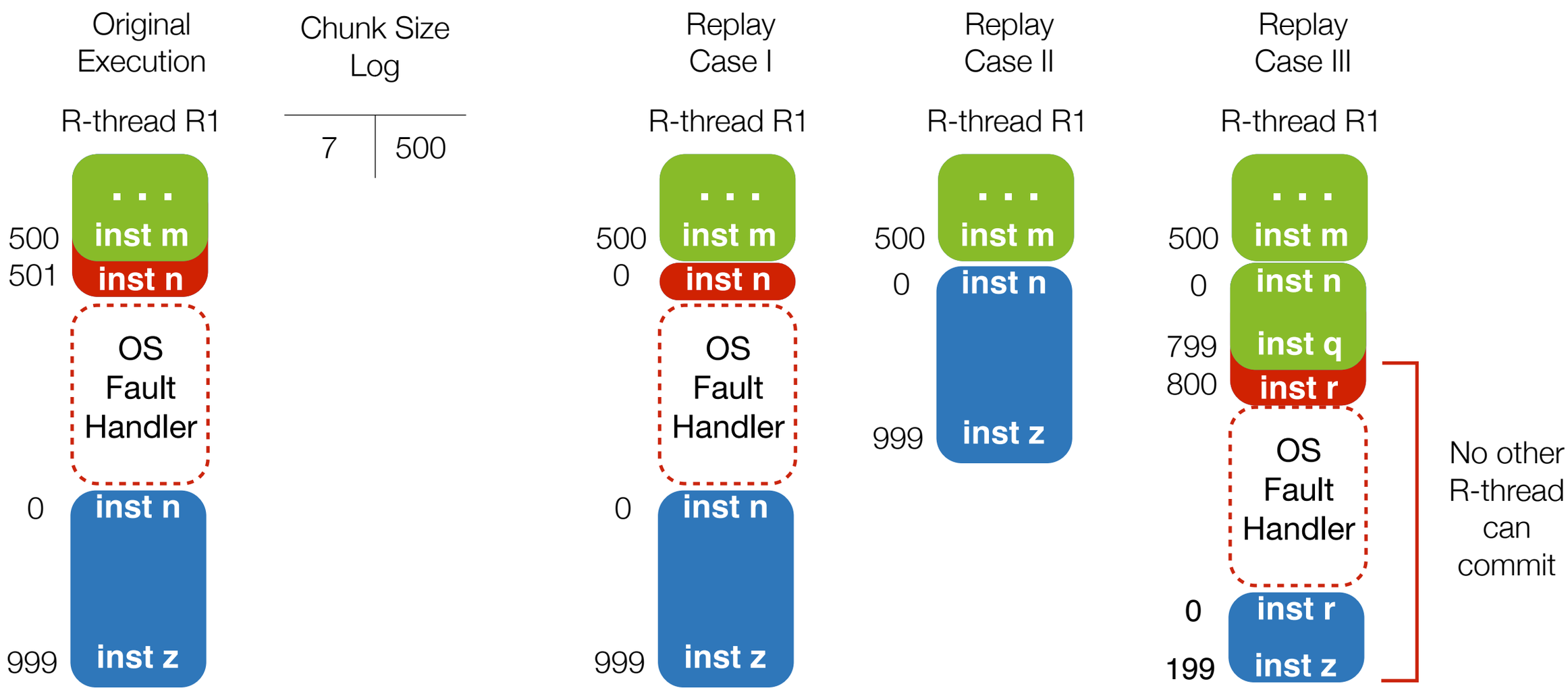
Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky



Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky

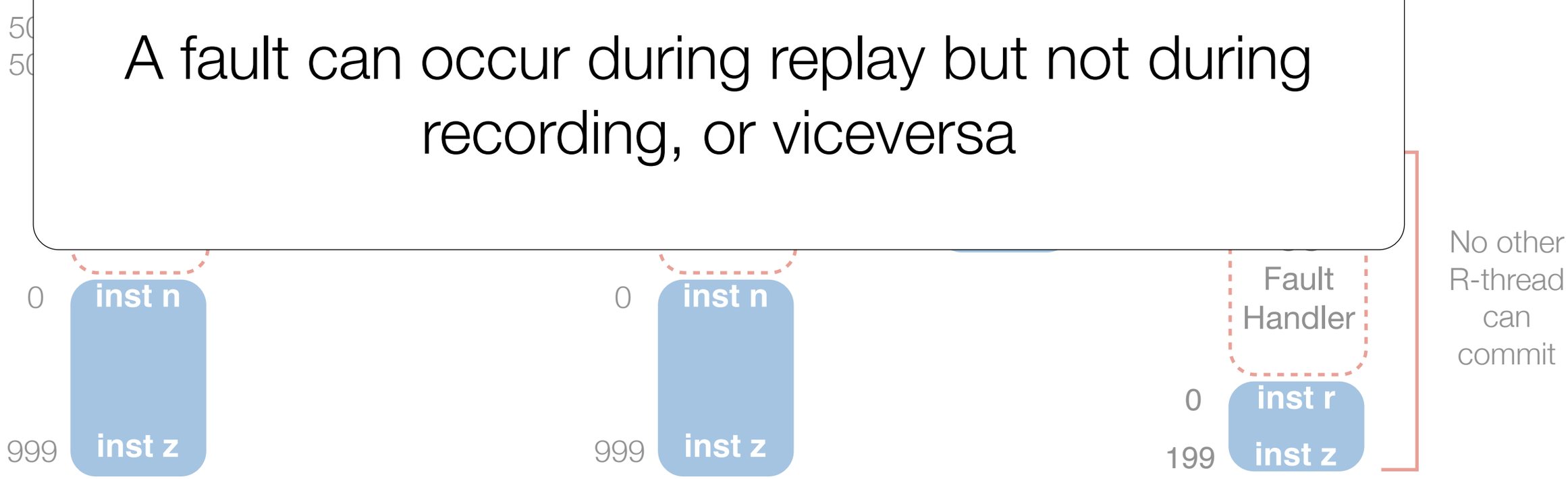


Handling Faults

- Commit all instructions up to faulting one and log chunk size
- Handling faults during replay can be tricky

Faults are synchronous events, but they are not necessarily deterministic

A fault can occur during replay but not during recording, or viceversa



Handling Traps and Programmed Exceptions

Handling Traps and Programmed Exceptions

- Commit all instructions including the one raising the trap

Handling Traps and Programmed Exceptions

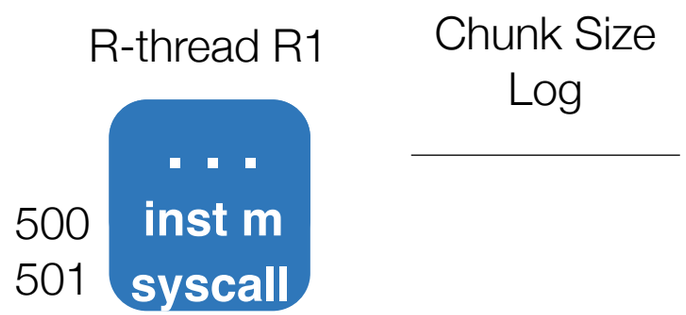
- Commit all instructions including the one raising the trap
- No need to record the size of irregular chunk

R-thread R1

Chunk Size
Log

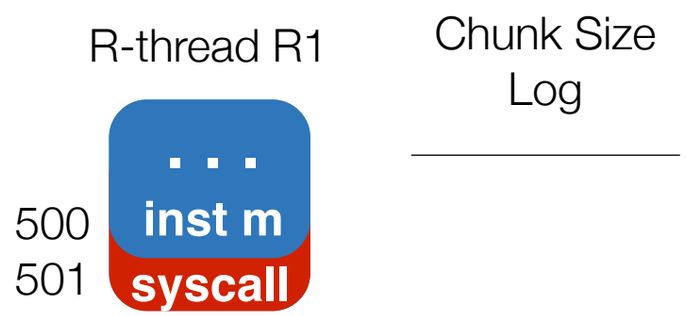
Handling Traps and Programmed Exceptions

- Commit all instructions including the one raising the trap
- No need to record the size of irregular chunk



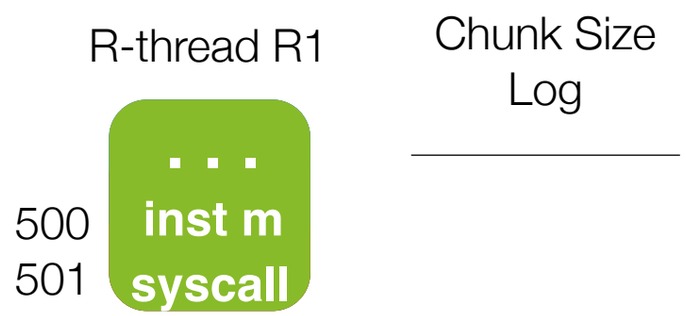
Handling Traps and Programmed Exceptions

- Commit all instructions including the one raising the trap
- No need to record the size of irregular chunk



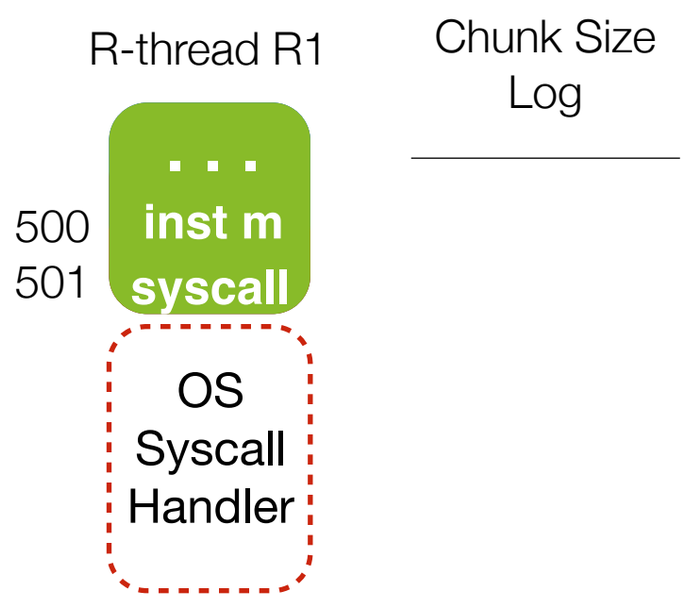
Handling Traps and Programmed Exceptions

- Commit all instructions including the one raising the trap
- No need to record the size of irregular chunk



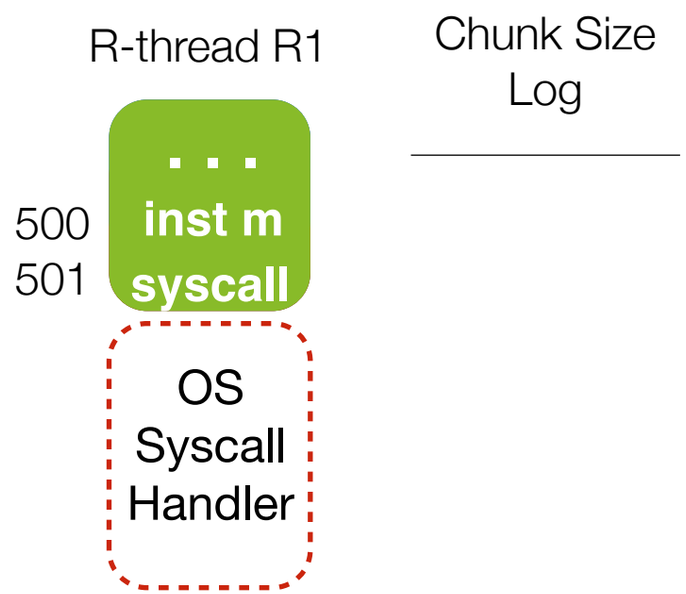
Handling Traps and Programmed Exceptions

- Commit all instructions including the one raising the trap
- No need to record the size of irregular chunk



Handling Traps and Programmed Exceptions

- Commit all instructions including the one raising the trap
- No need to record the size of irregular chunk

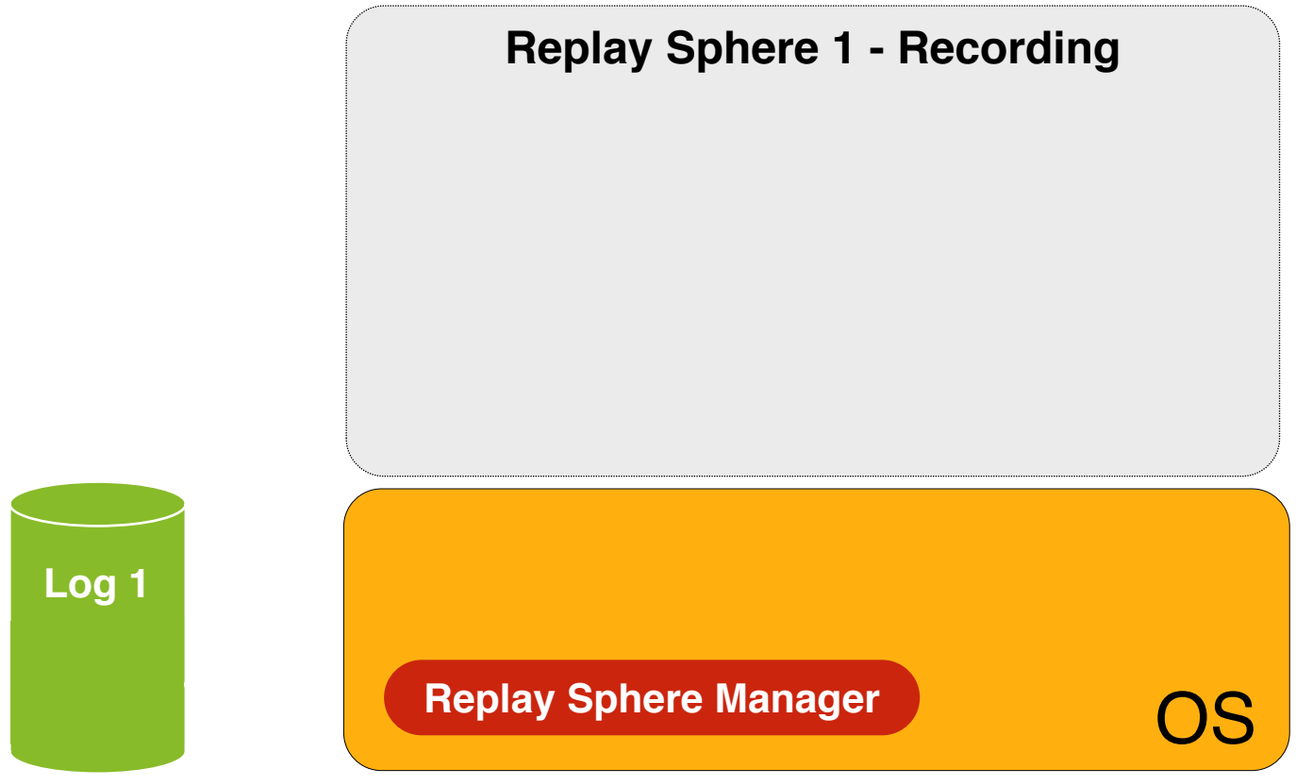


Handling Traps and Programmed Exceptions

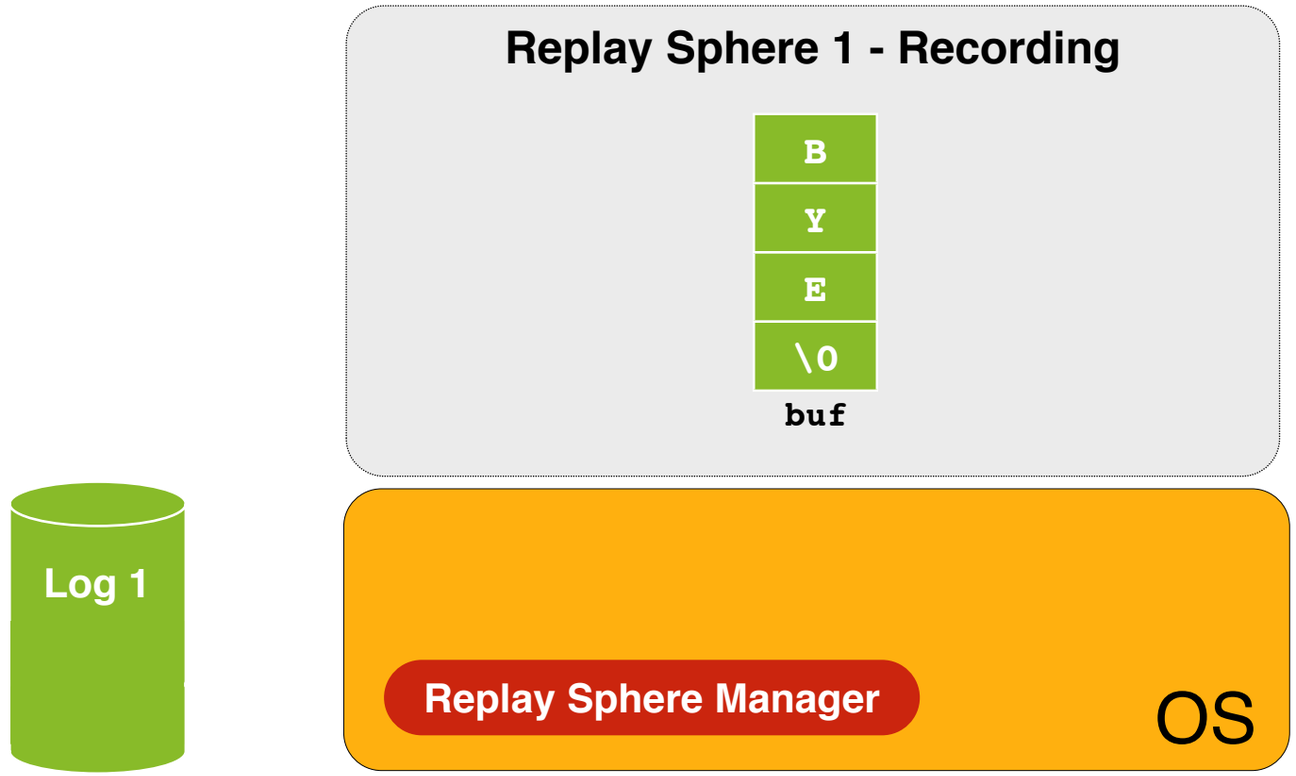
- Commit all instructions including the one raising the trap
- Never attempt to recover the origin of irregular-sized chunks

Traps and programmed exceptions are deterministic events: no need to record irregular-sized chunks

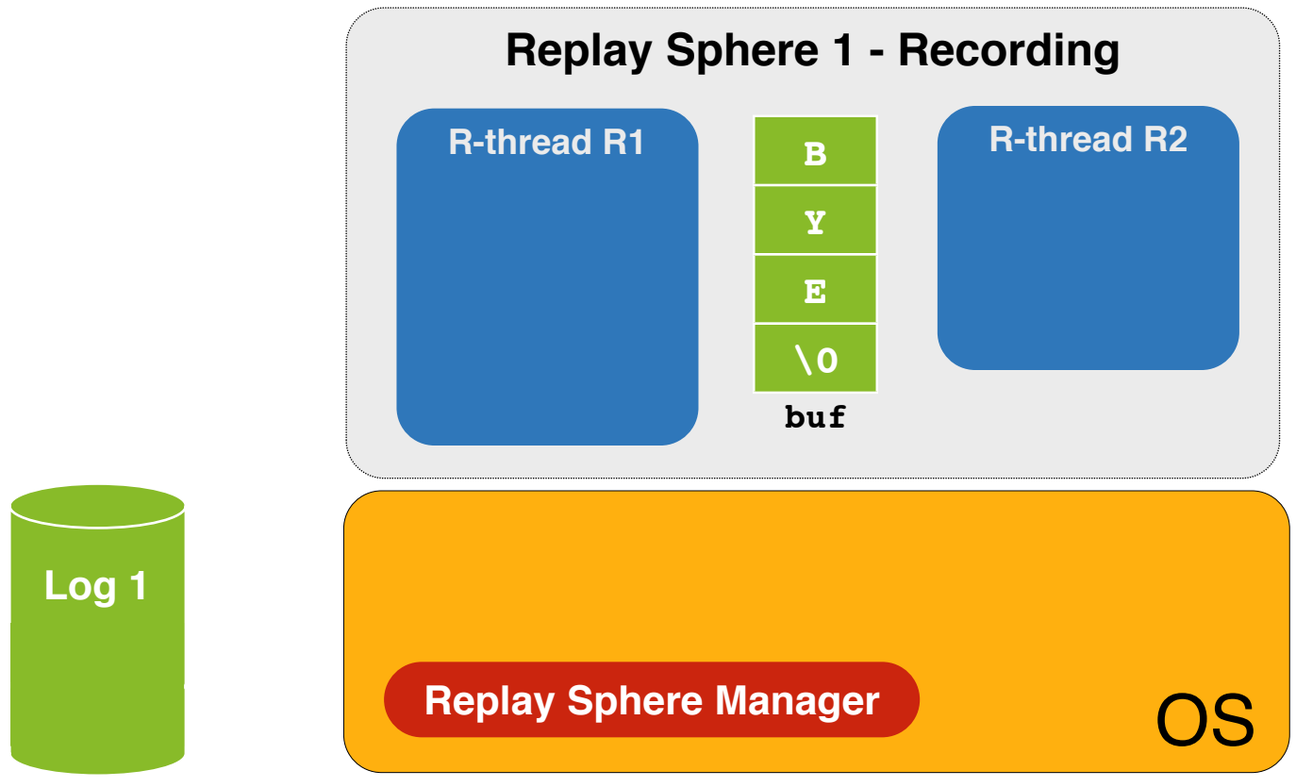
Injecting Data into Spheres



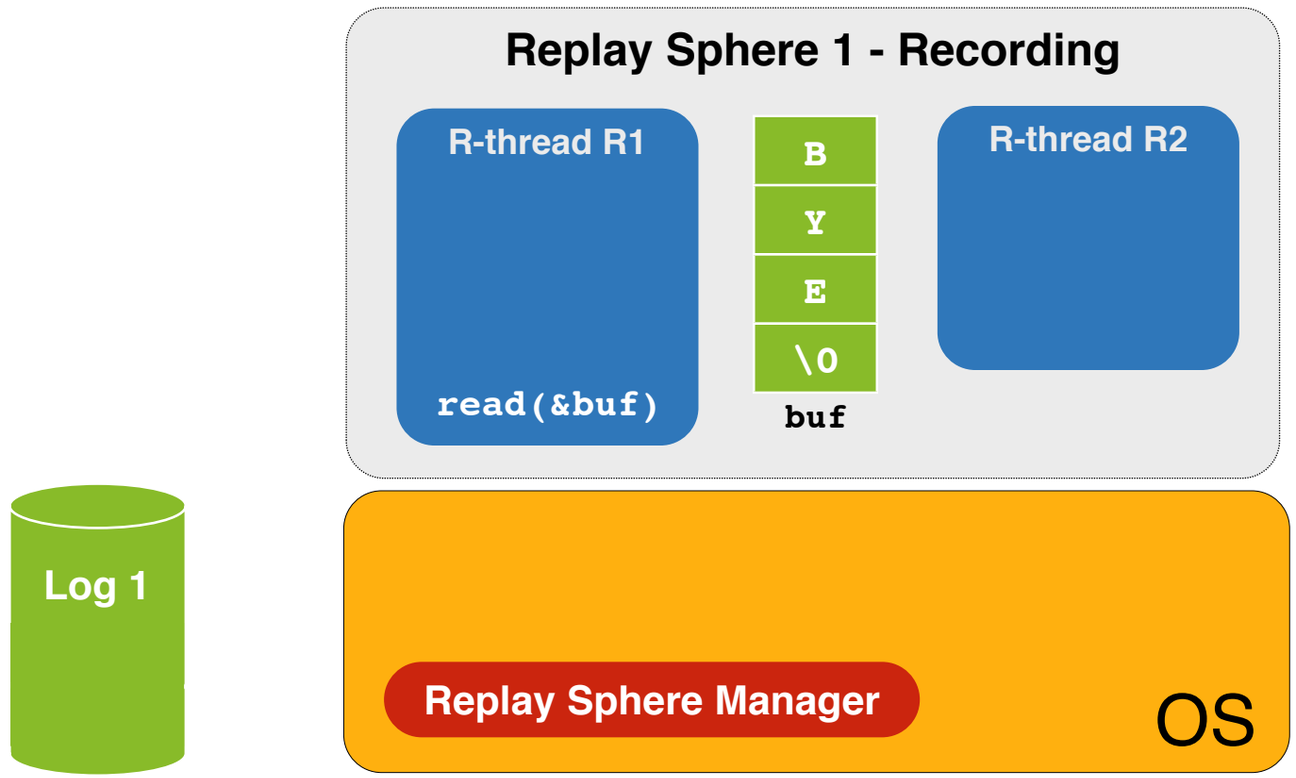
Injecting Data into Spheres



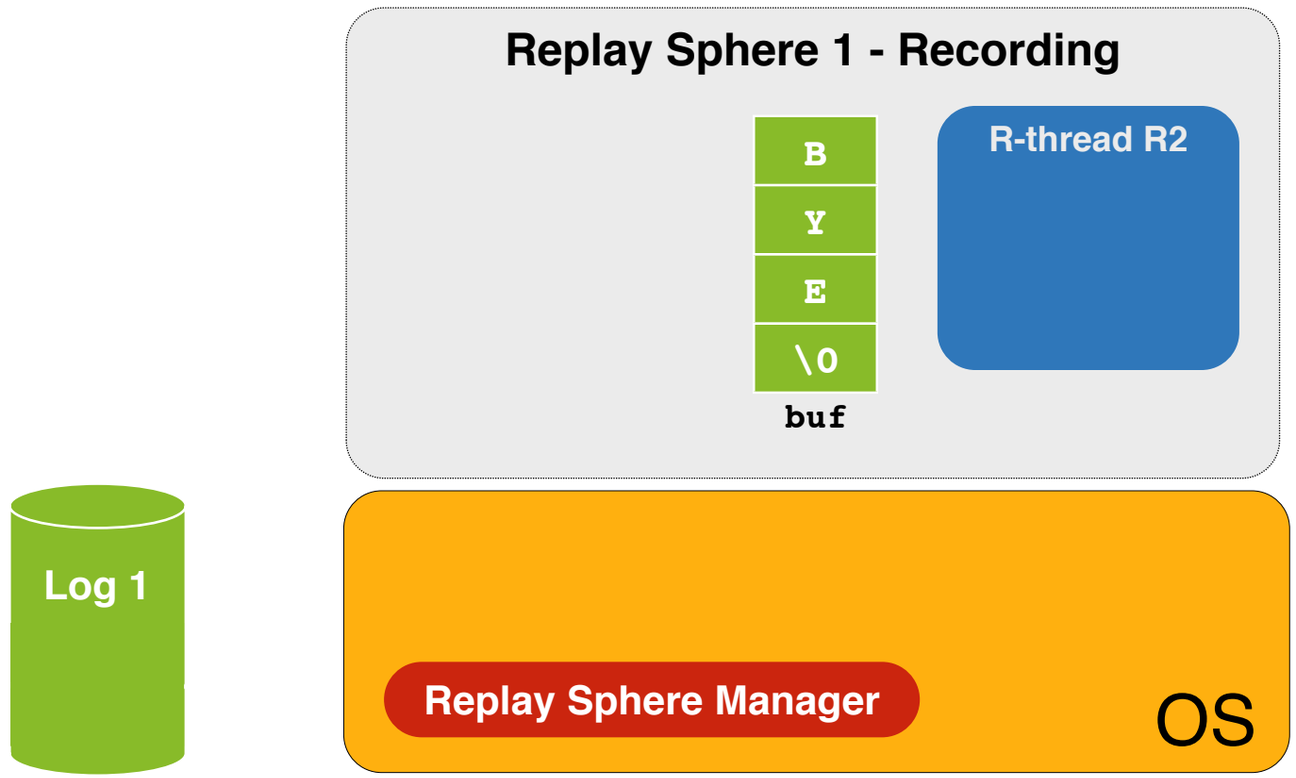
Injecting Data into Spheres



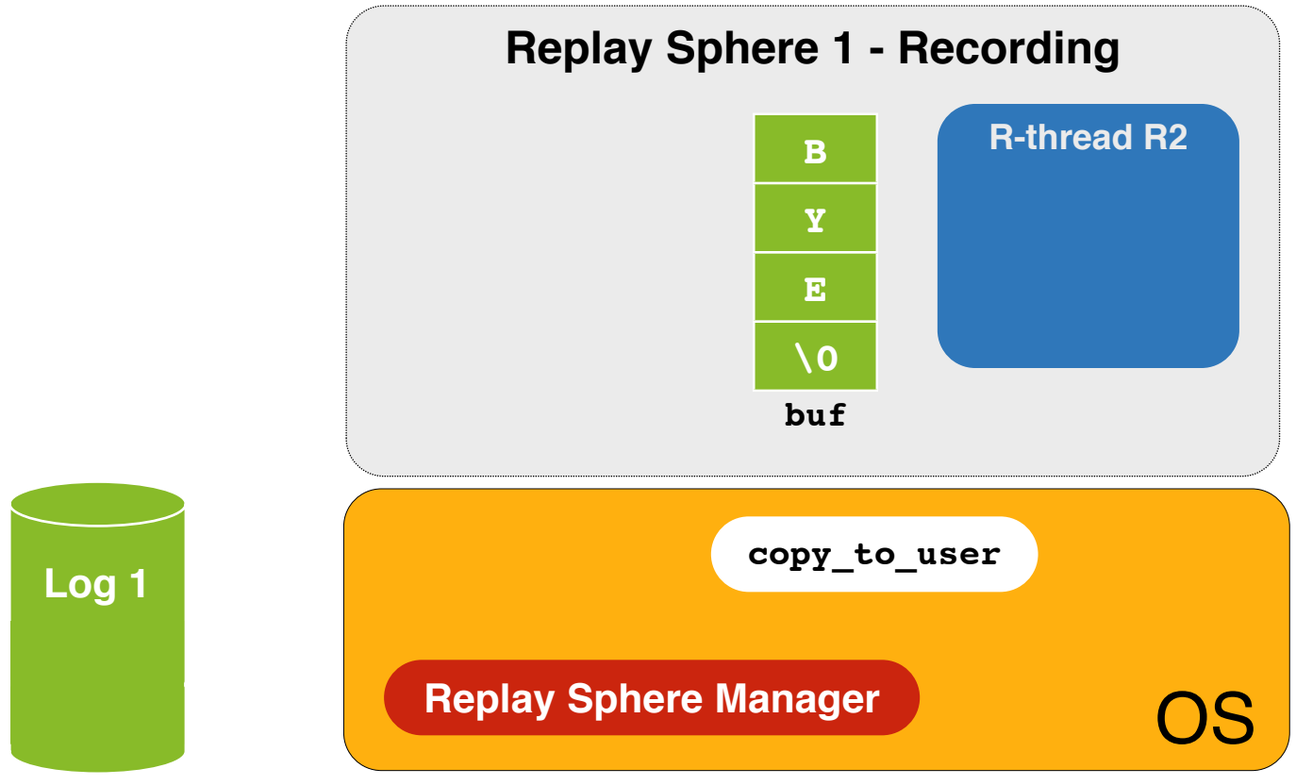
Injecting Data into Spheres



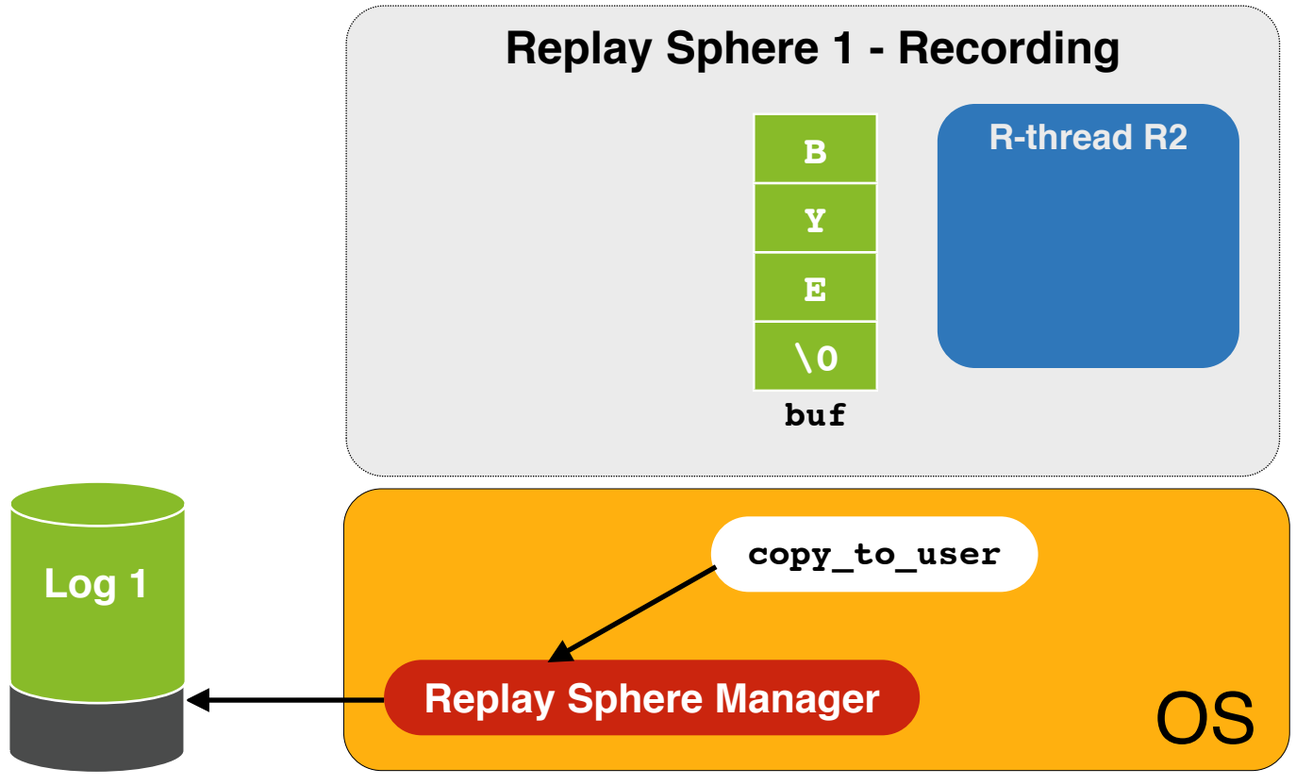
Injecting Data into Spheres



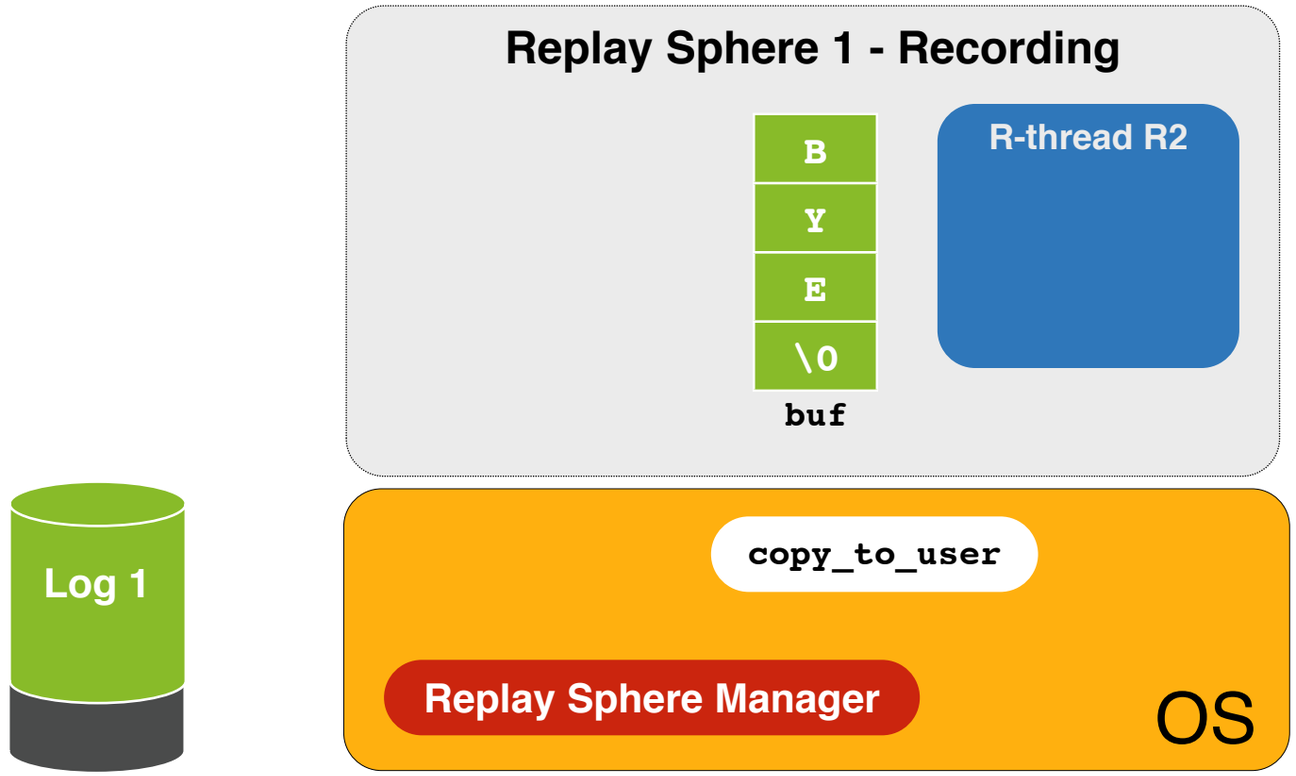
Injecting Data into Spheres



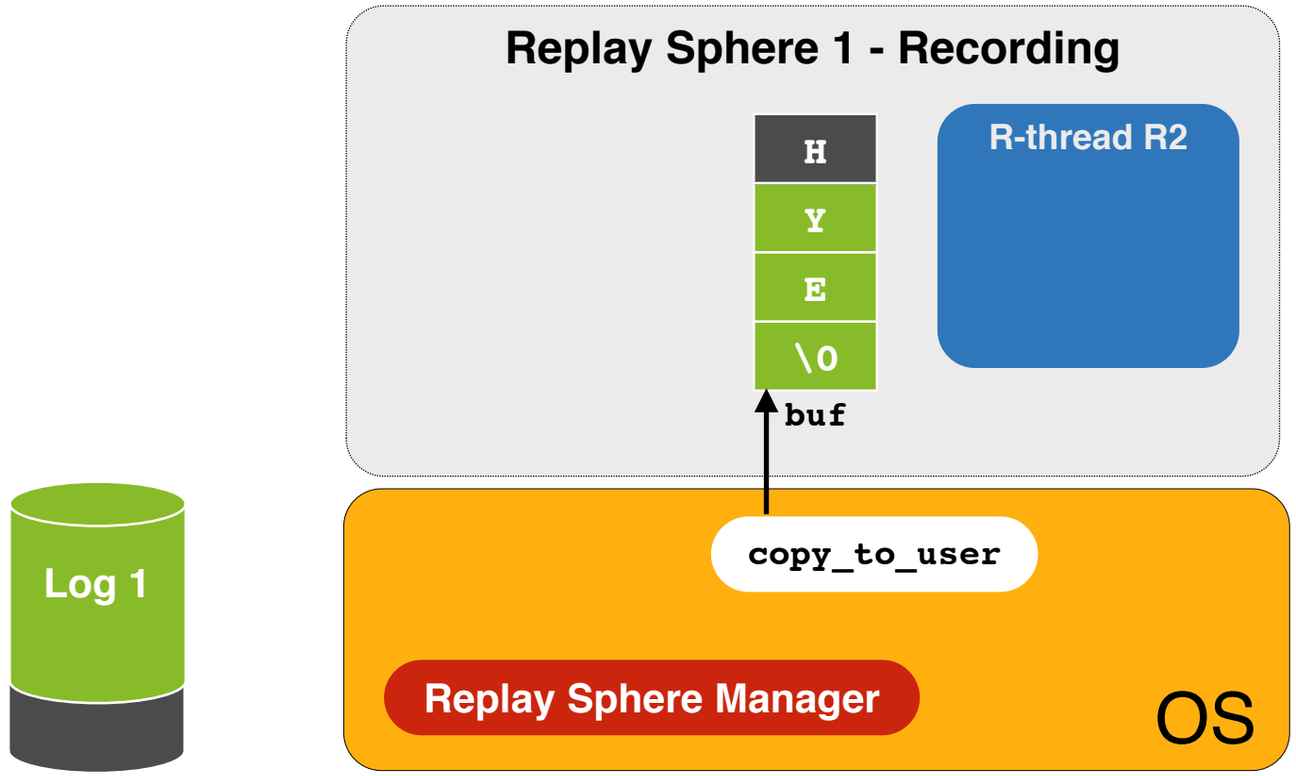
Injecting Data into Spheres



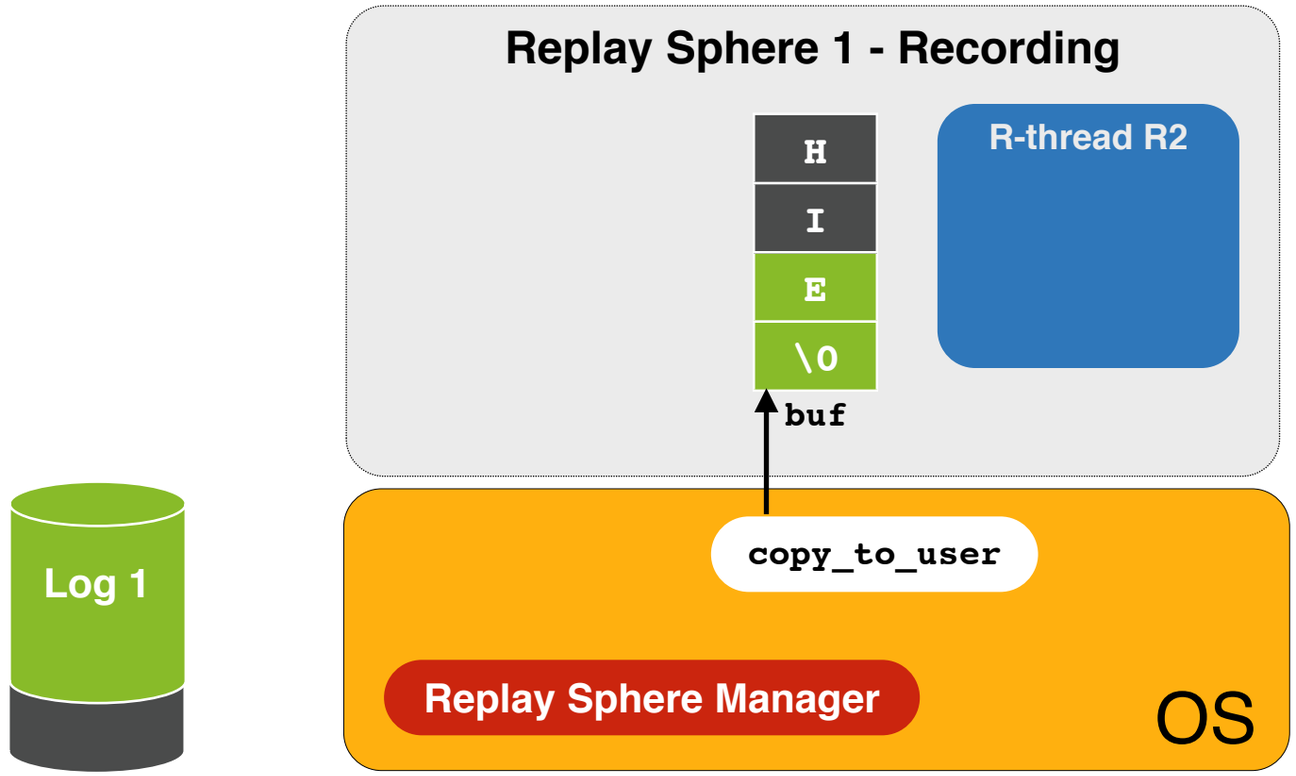
Injecting Data into Spheres



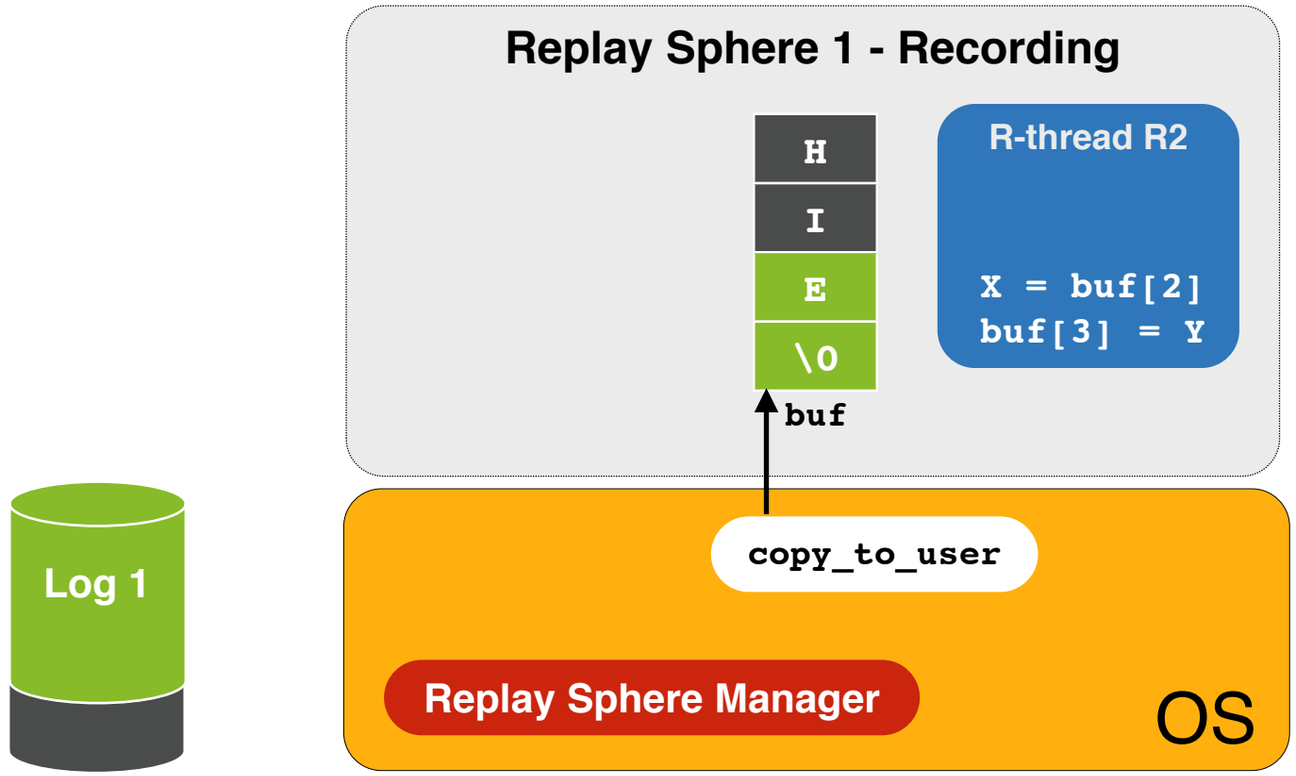
Injecting Data into Spheres



Injecting Data into Spheres

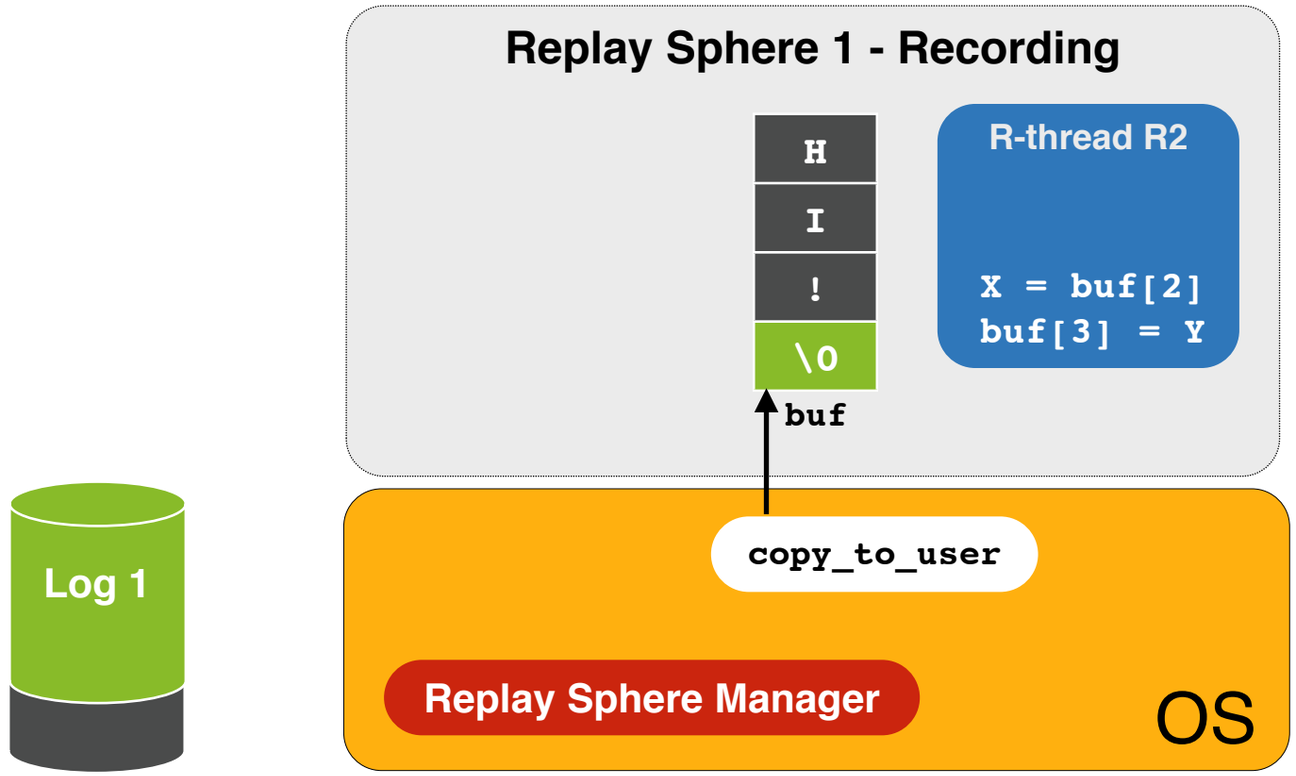


Injecting Data into Spheres



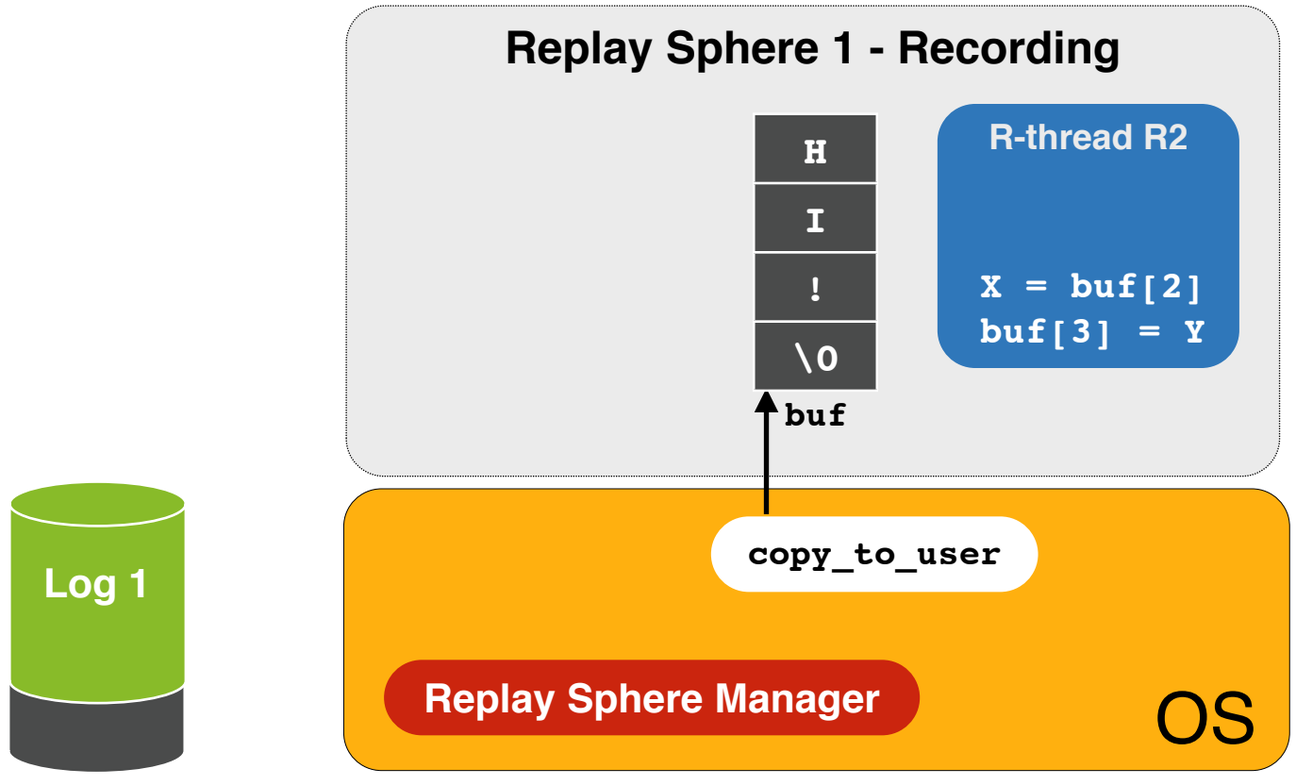
- Problem: interleaving between OS copies and R-threads not recorded

Injecting Data into Spheres



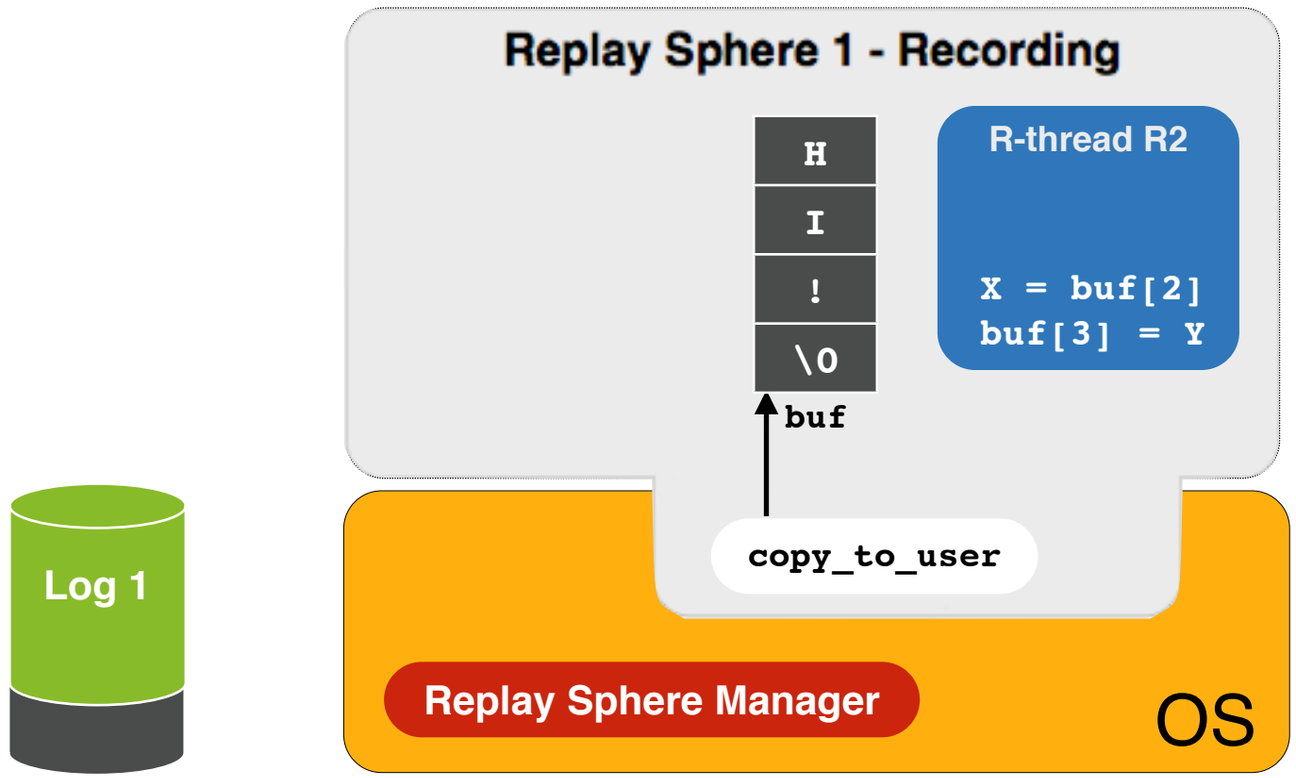
- Problem: interleaving between OS copies and R-threads not recorded

Injecting Data into Spheres



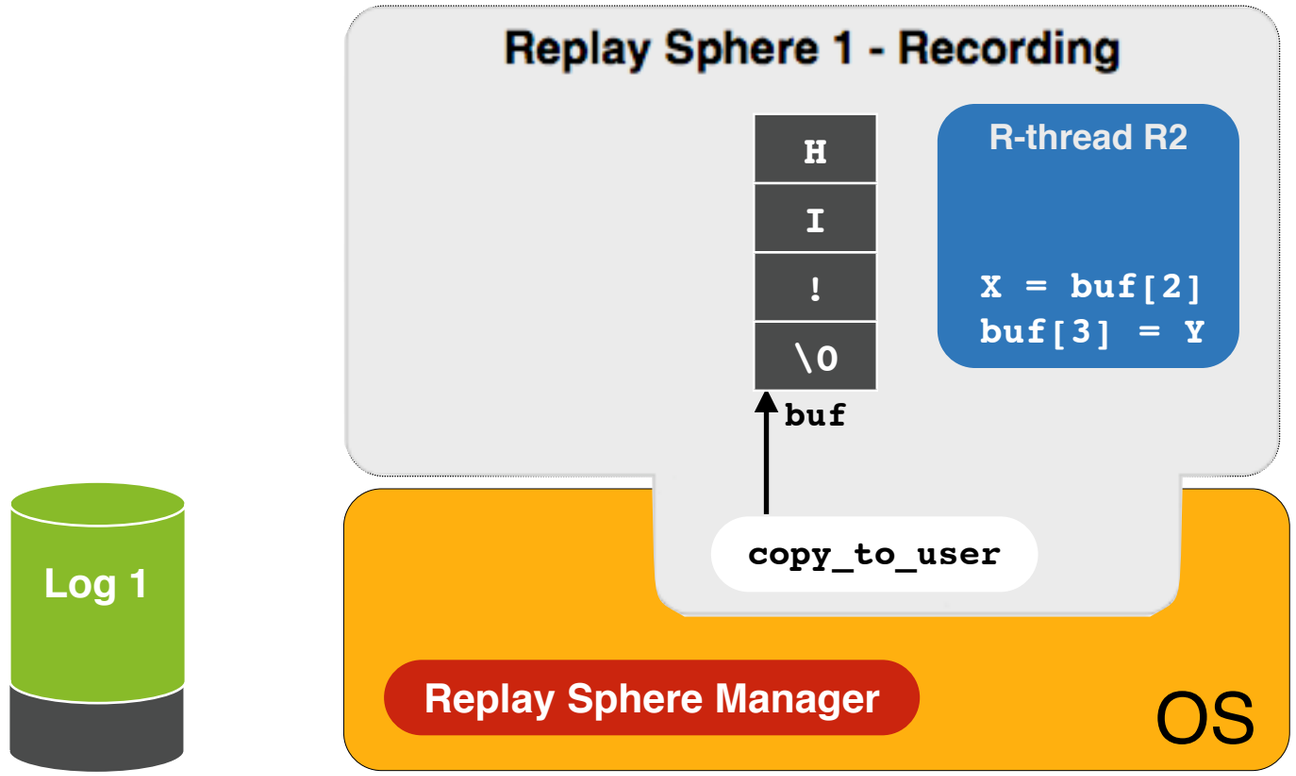
- Problem: interleaving between OS copies and R-threads not recorded

Injecting Data into Spheres



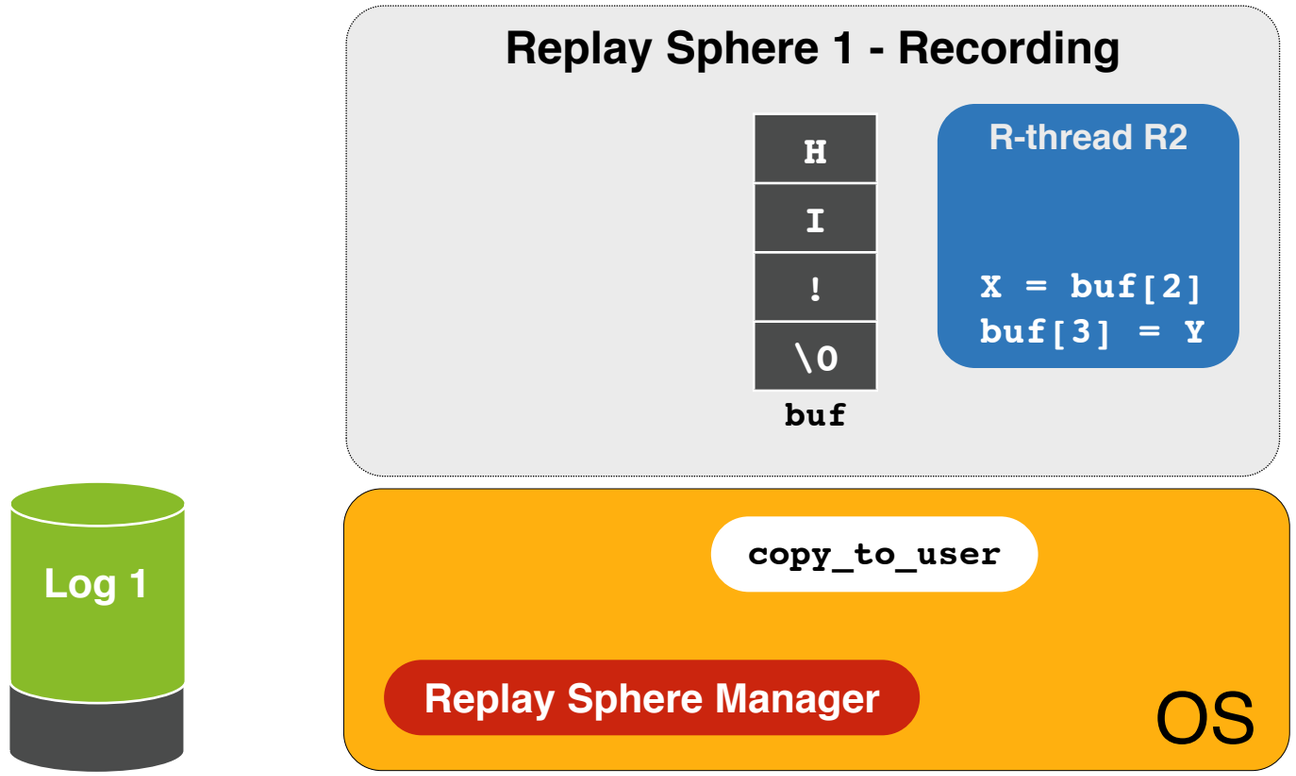
- Problem: interleaving between OS copies and R-threads not recorded
- Solution: insert `copy_to_user` into sphere:

Injecting Data into Spheres



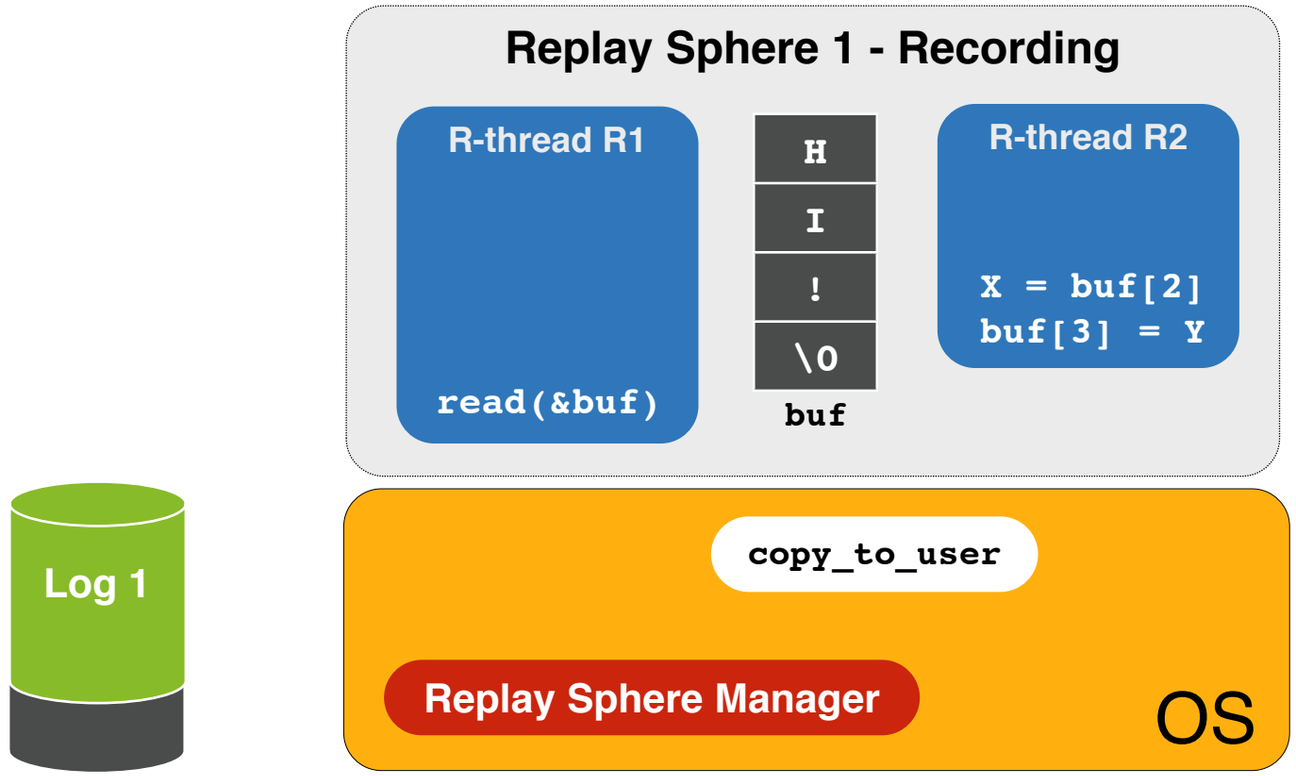
- Problem: interleaving between OS copies and R-threads not recorded
- Solution: insert `copy_to_user` into sphere:
  - HW can log memory access interleaving

Injecting Data into Spheres



- Problem: interleaving between OS copies and R-threads not recorded
- Solution: insert `copy_to_user` into sphere:
  - HW can log memory access interleaving
  - `copy_to_user` exits sphere once copy is over

Injecting Data into Spheres



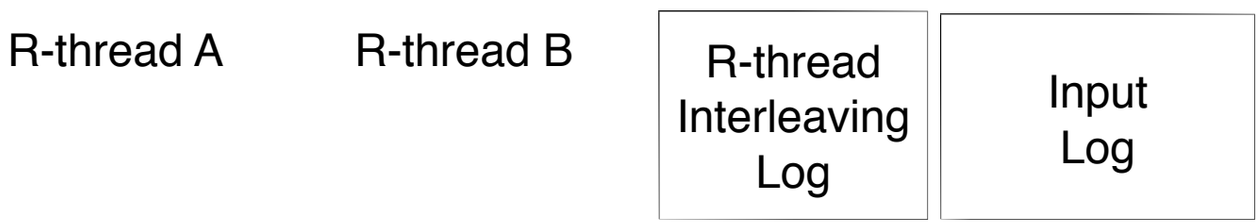
- Problem: interleaving between OS copies and R-threads not recorded
- Solution: insert `copy_to_user` into sphere:
  - HW can log memory access interleaving
  - `copy_to_user` exits sphere once copy is over

Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log

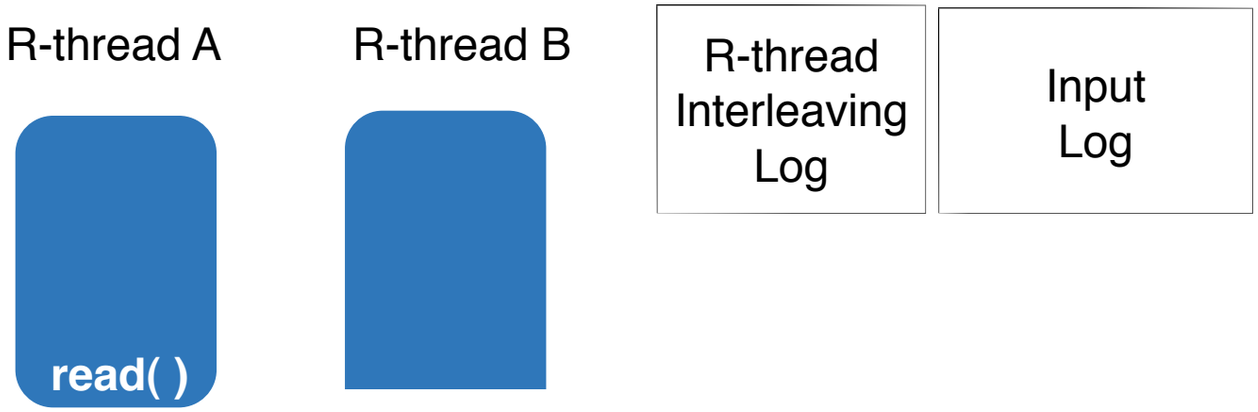
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log



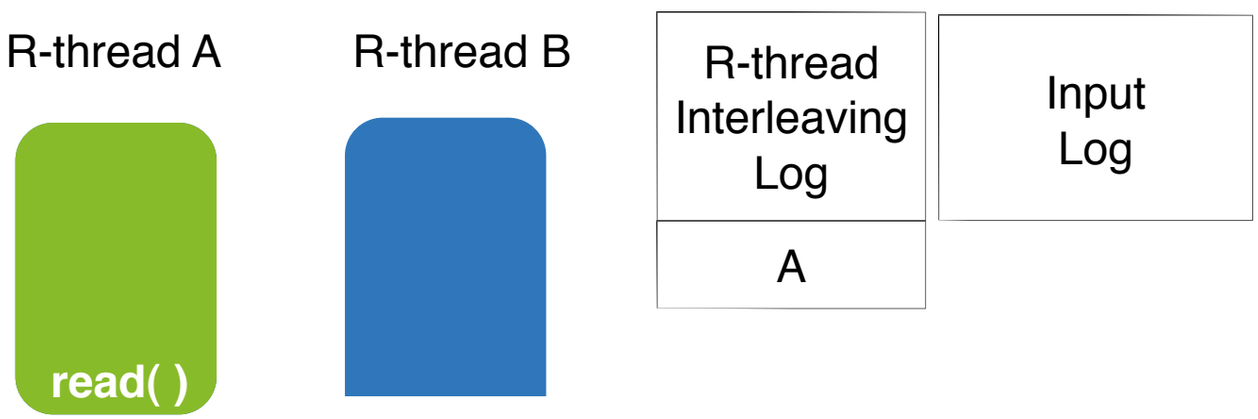
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log



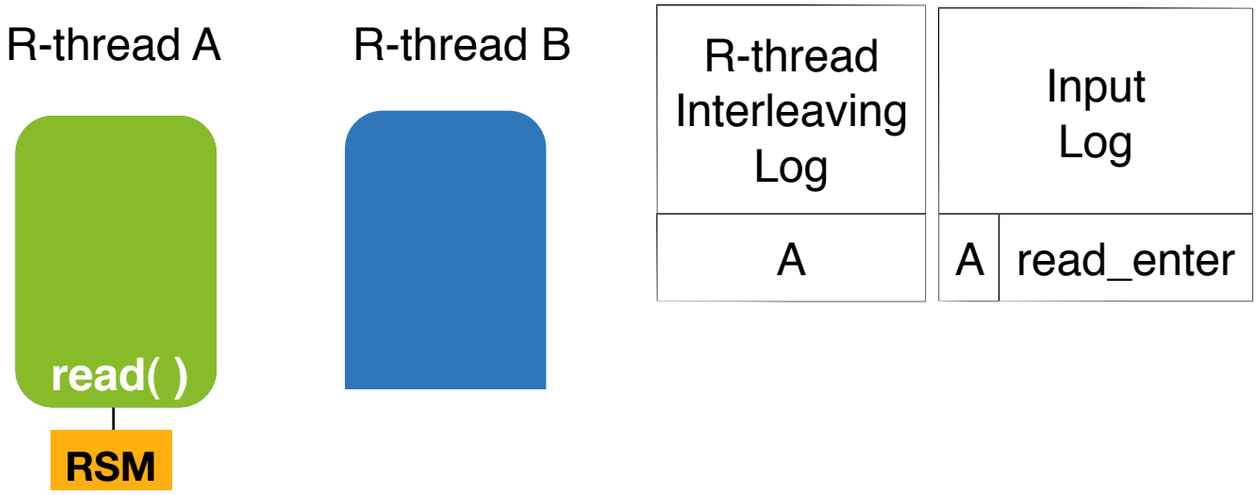
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log



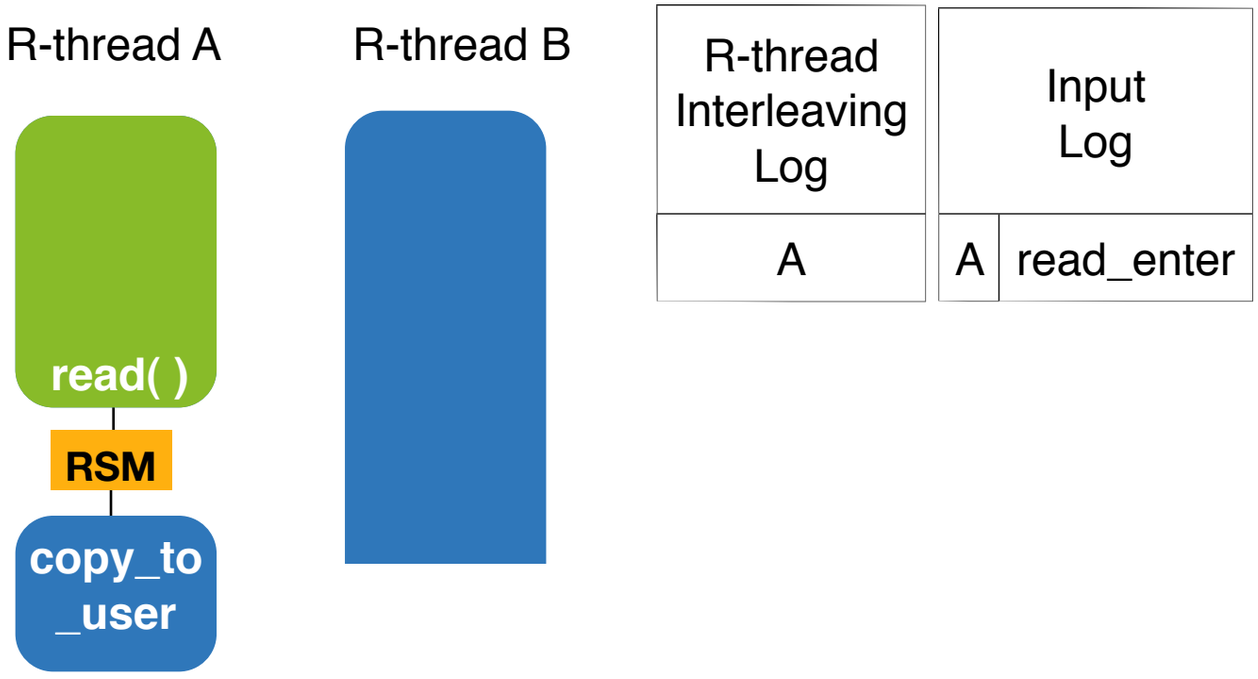
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log



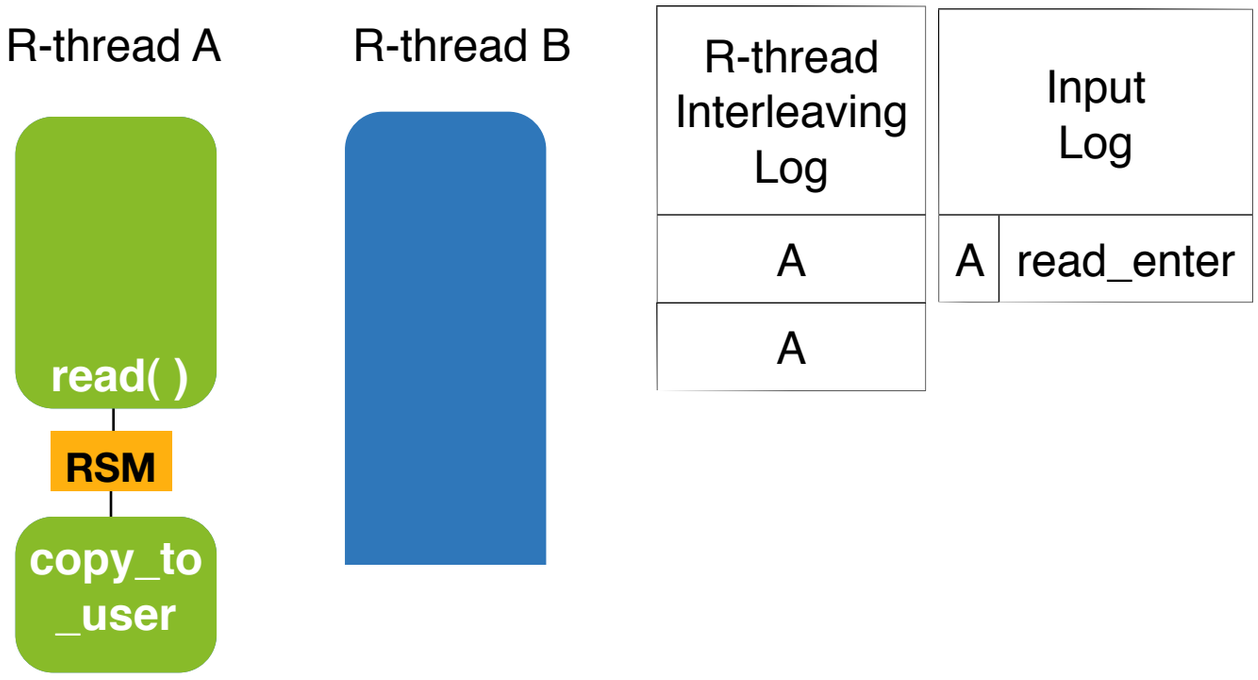
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log



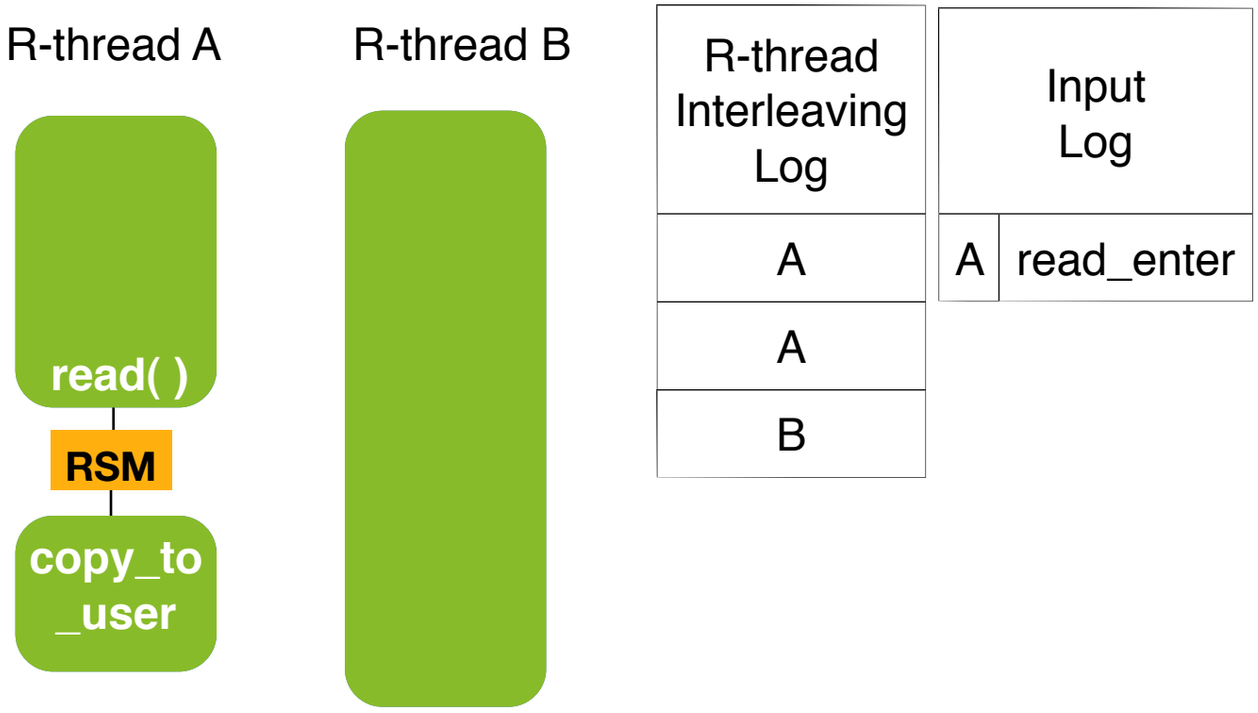
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log



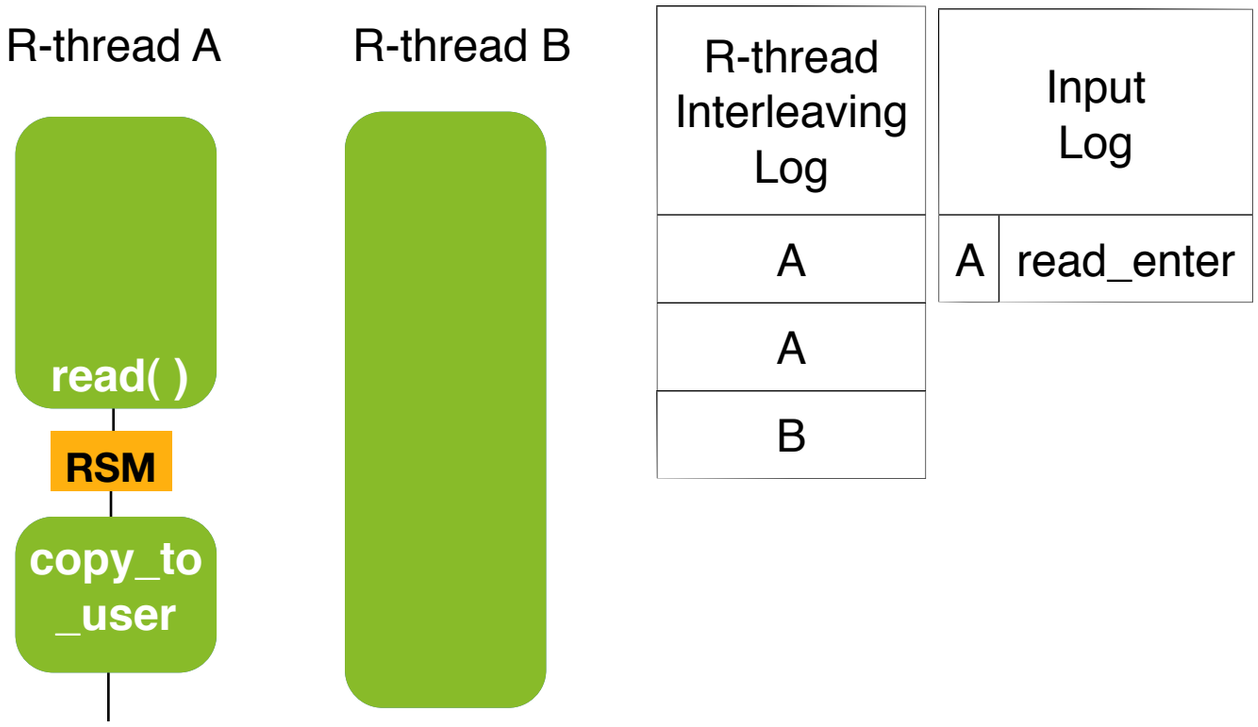
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log



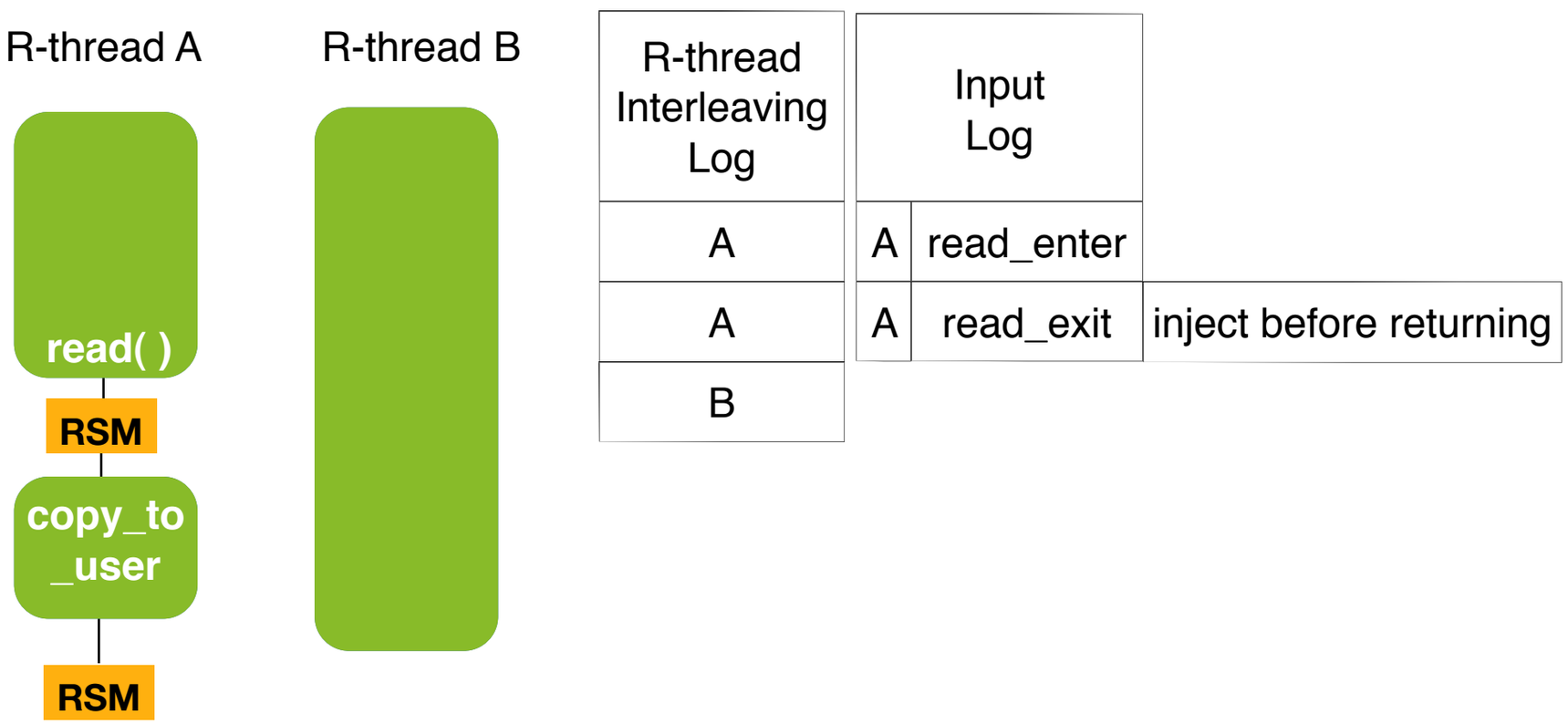
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log



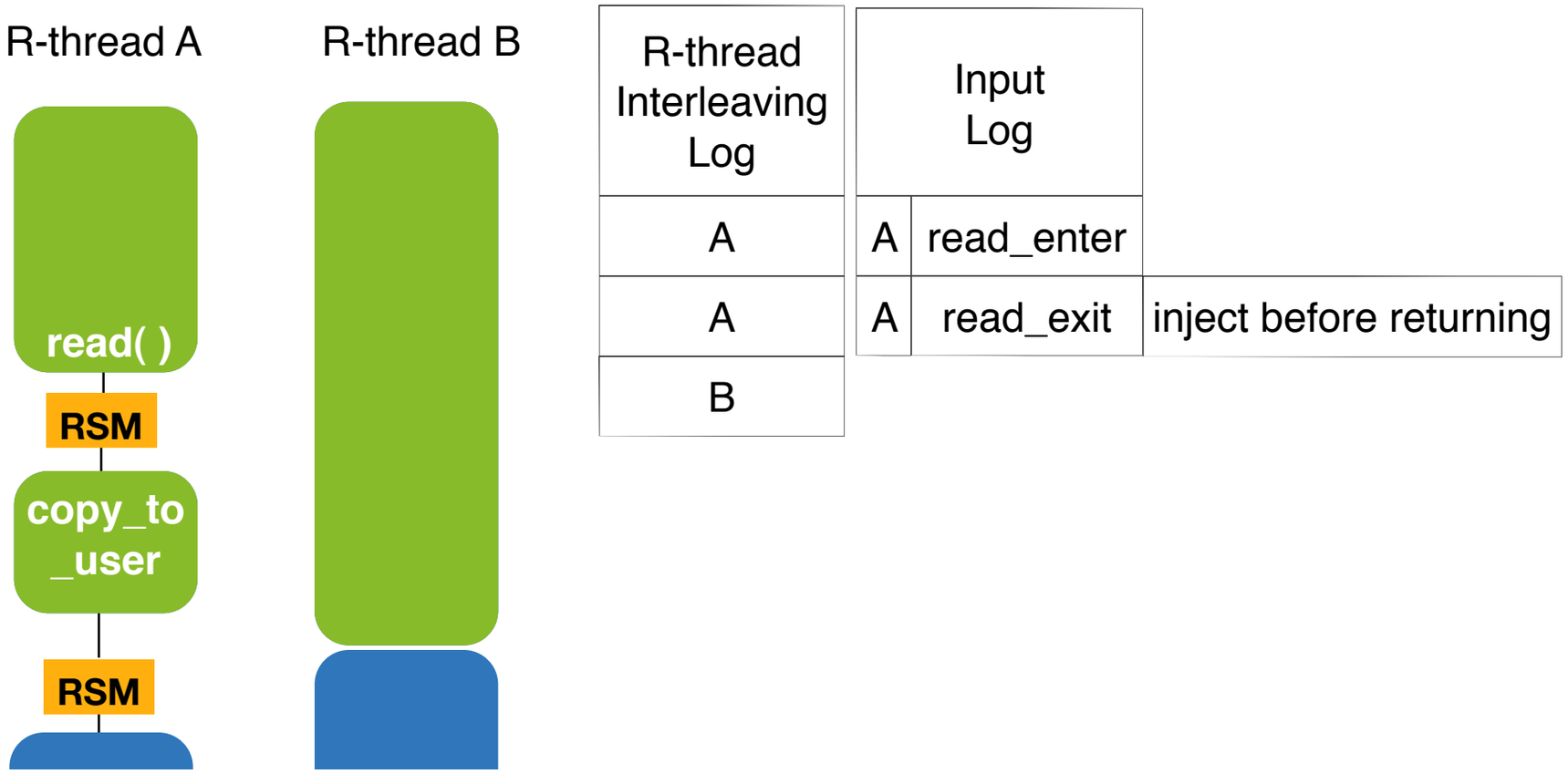
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log



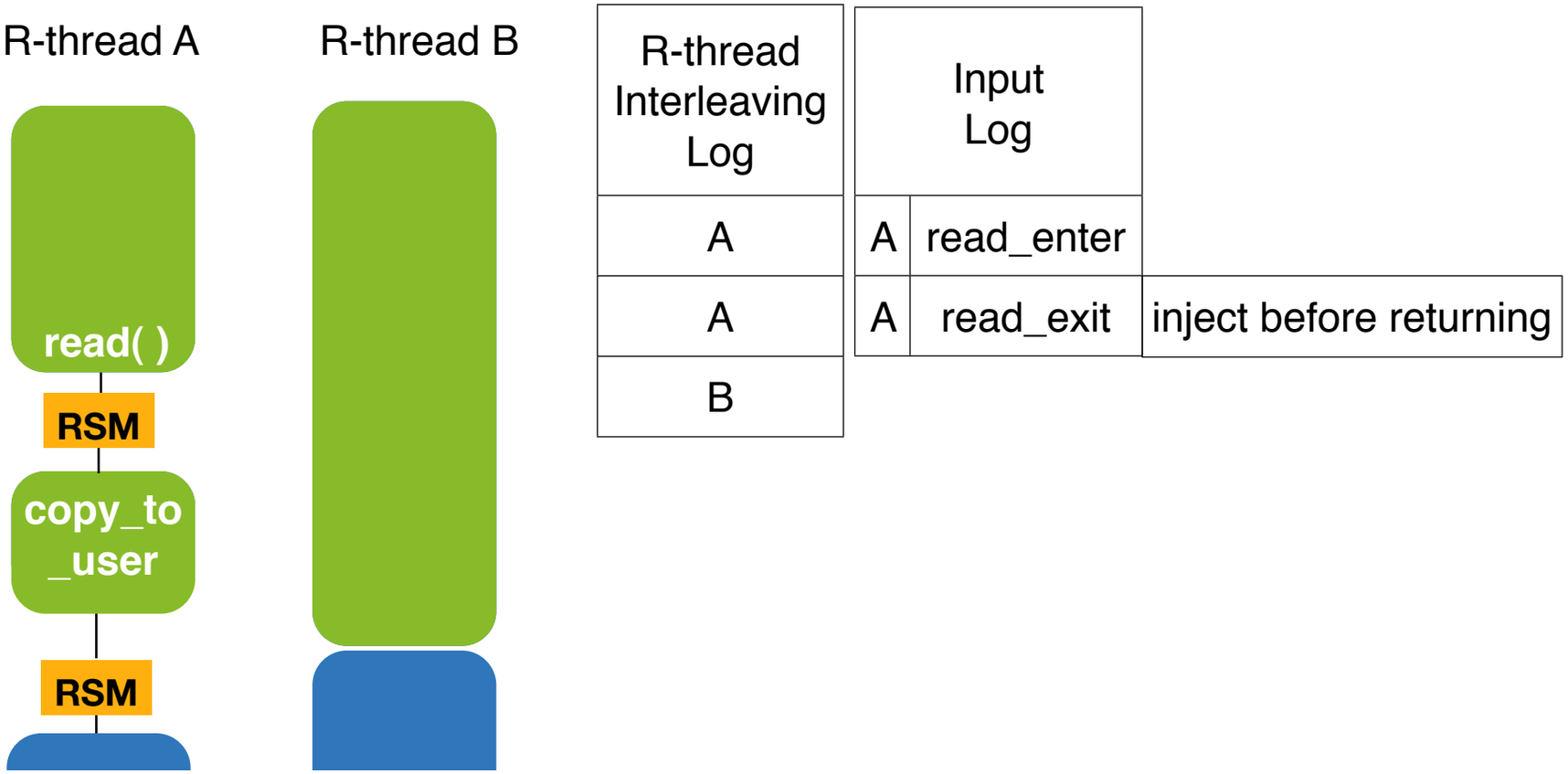
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log



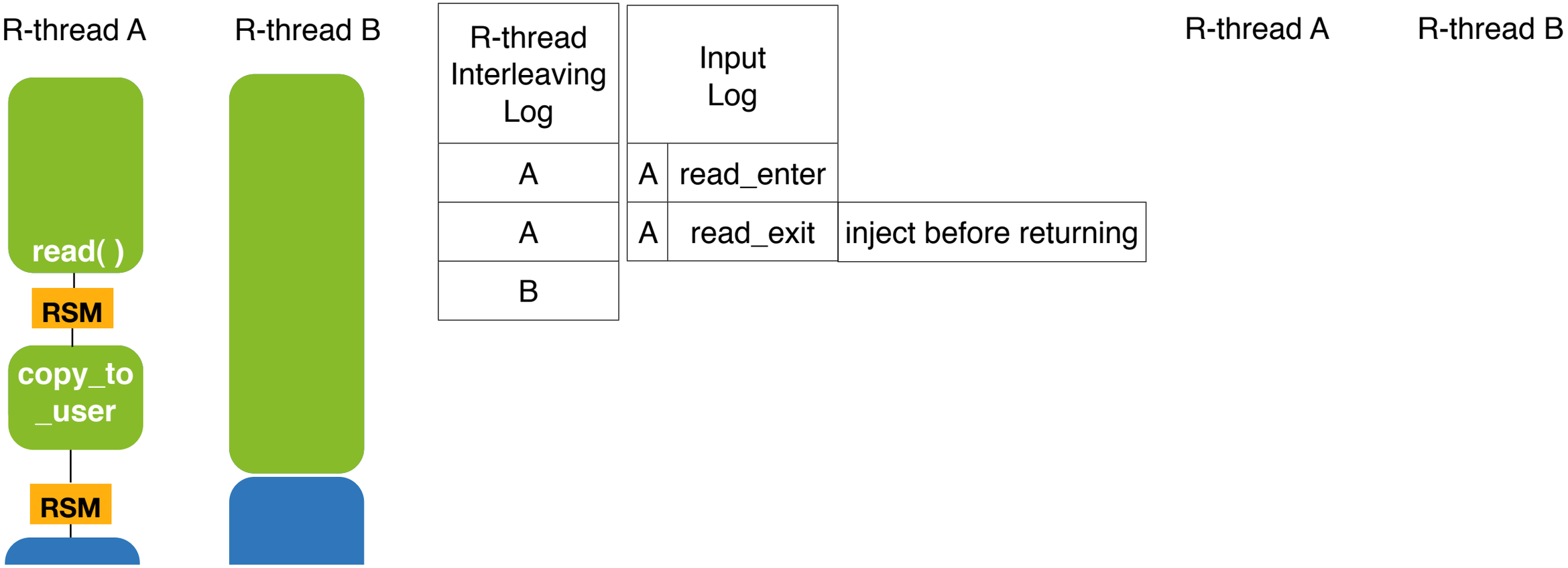
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log



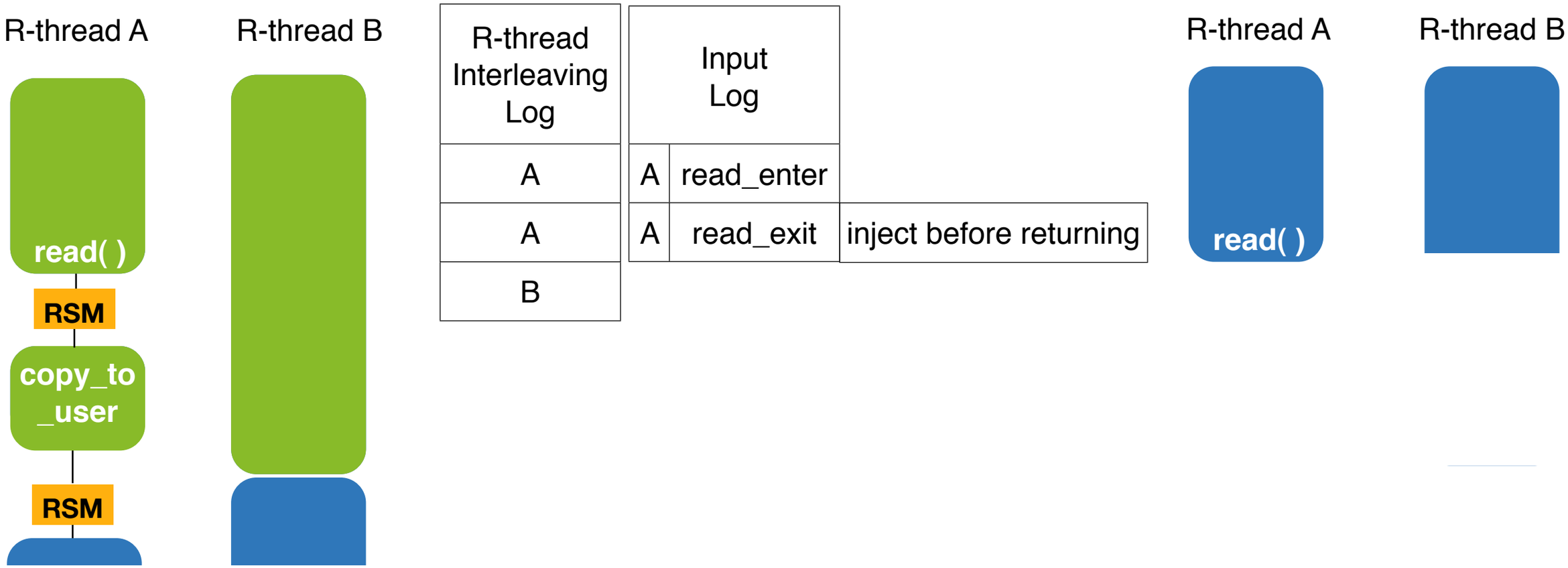
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log



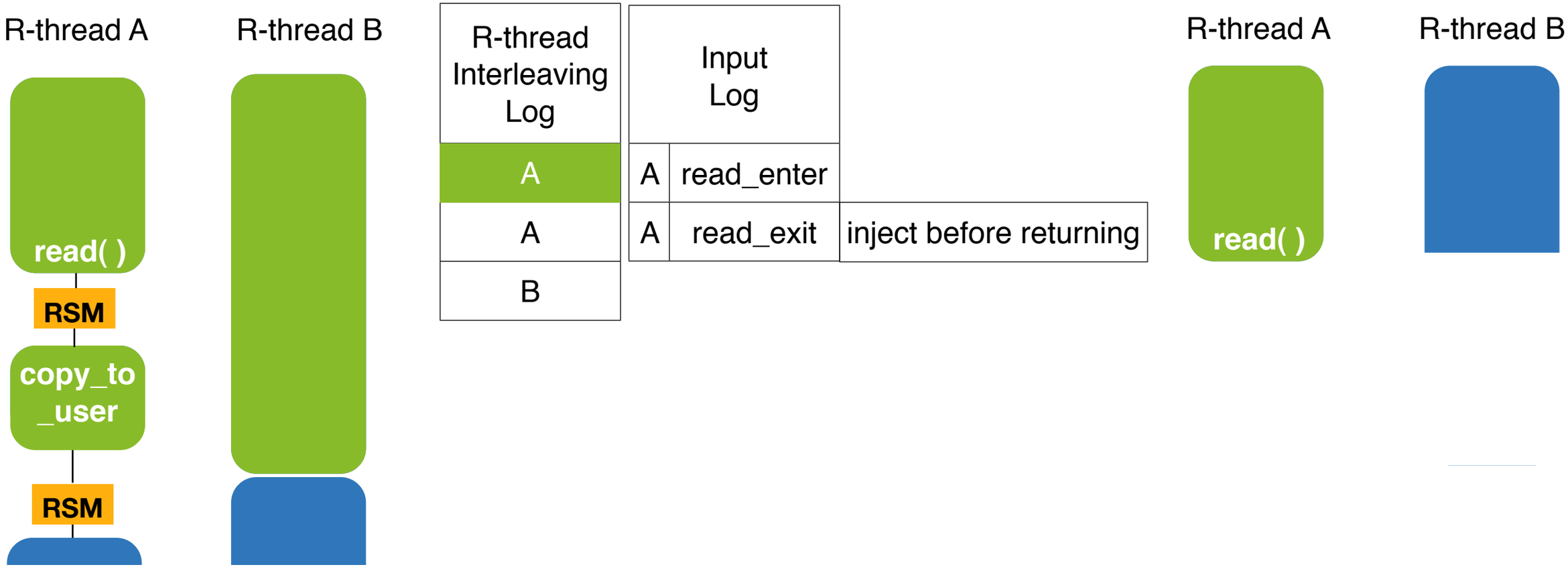
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log



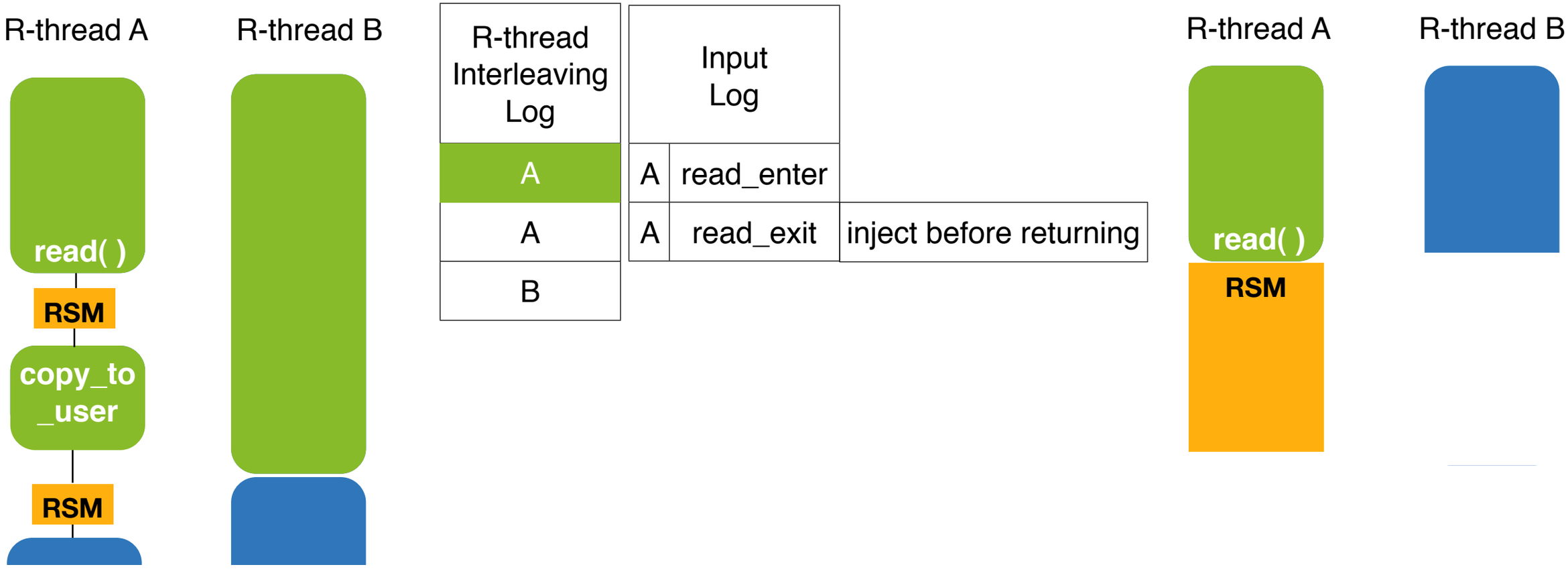
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log



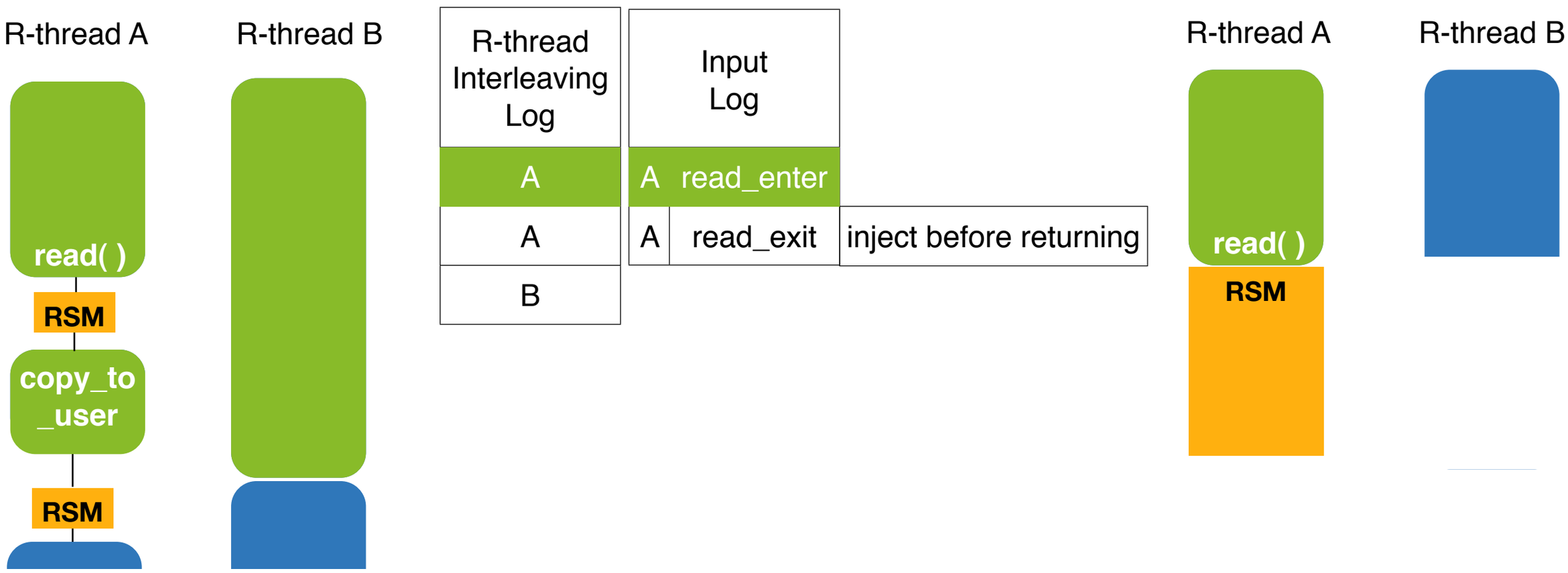
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log



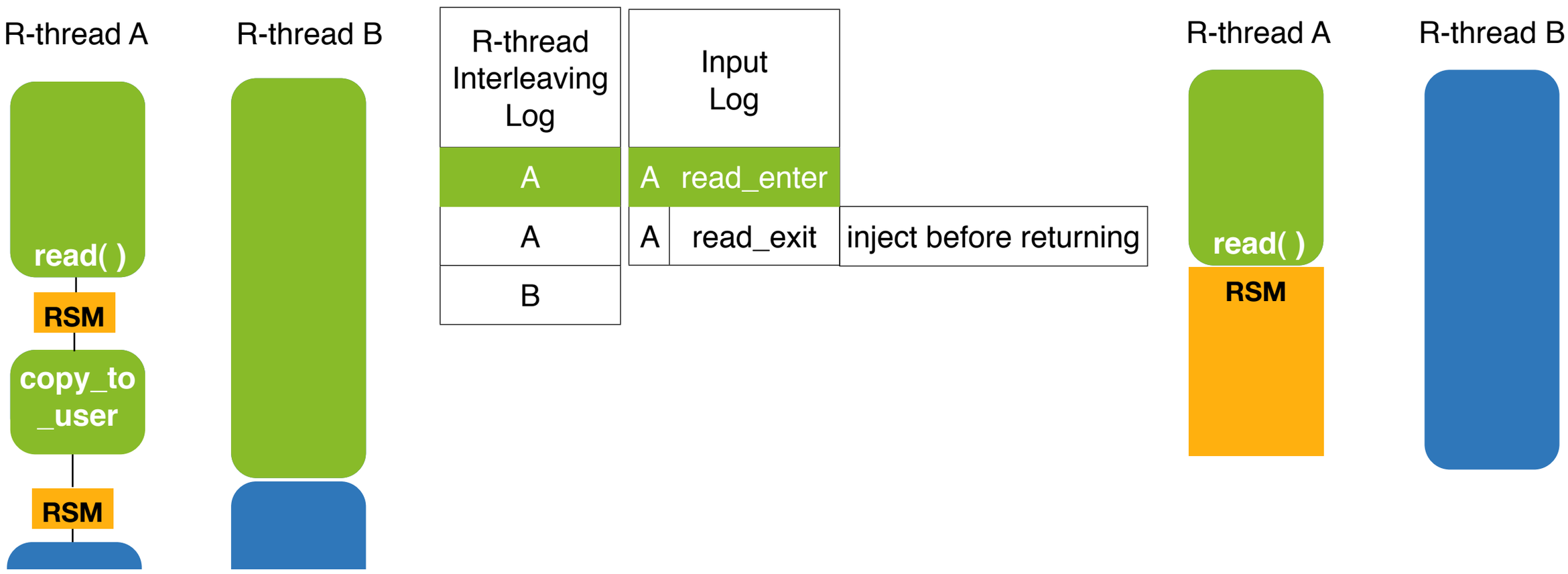
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log



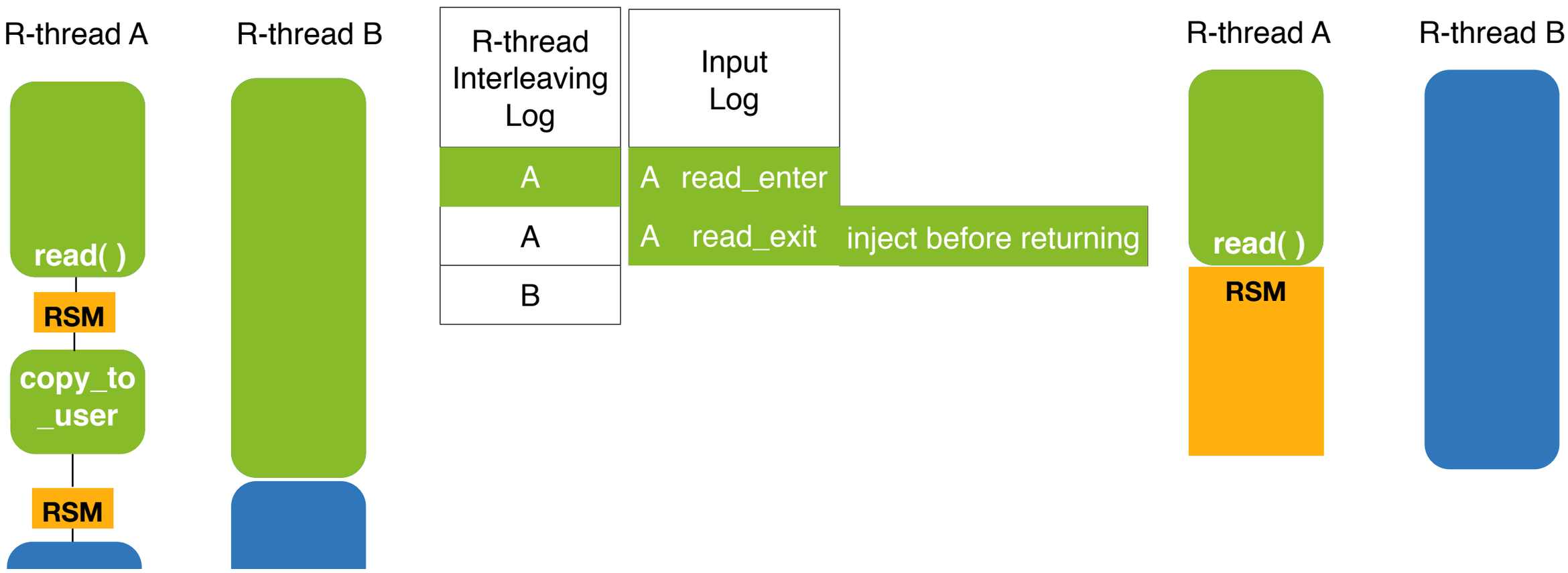
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log



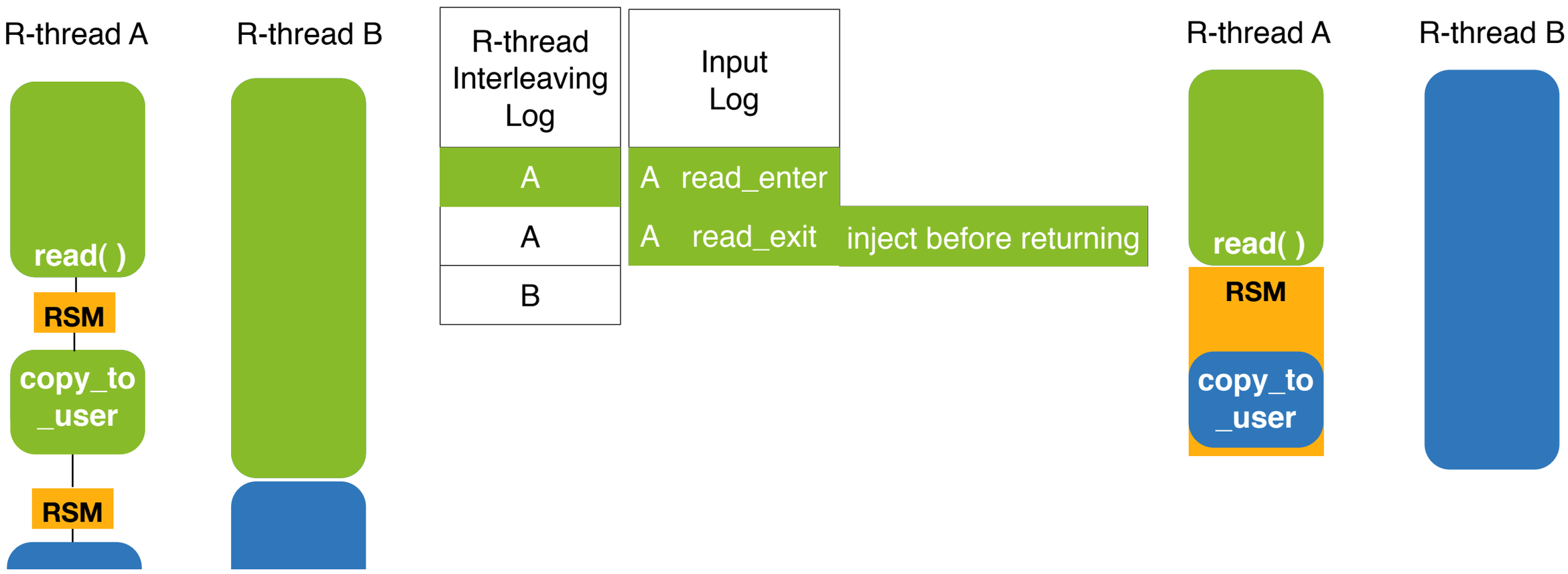
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log



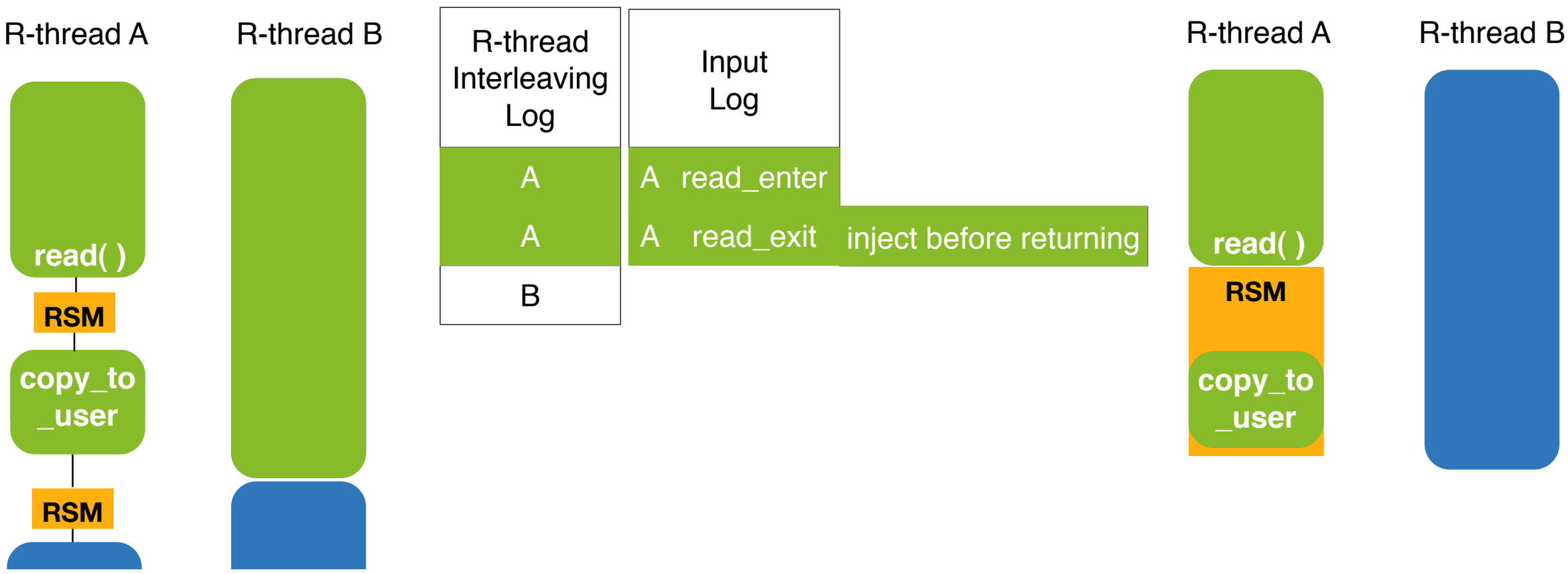
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log



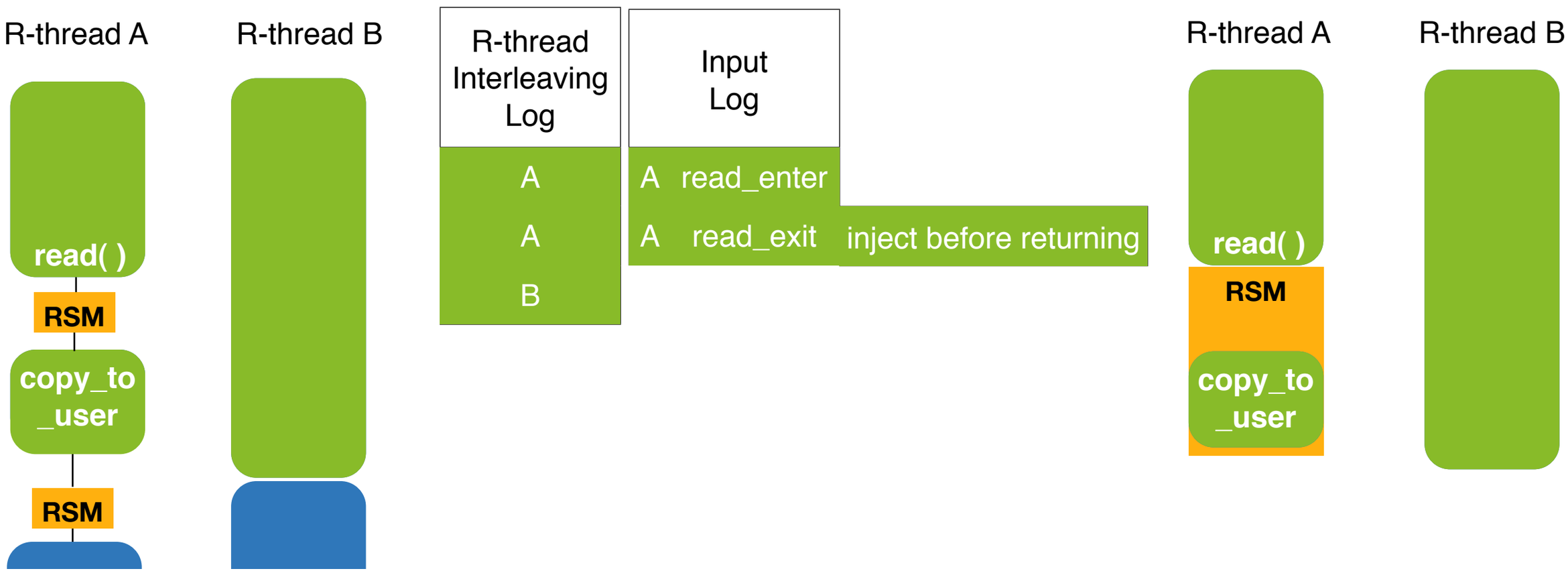
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log



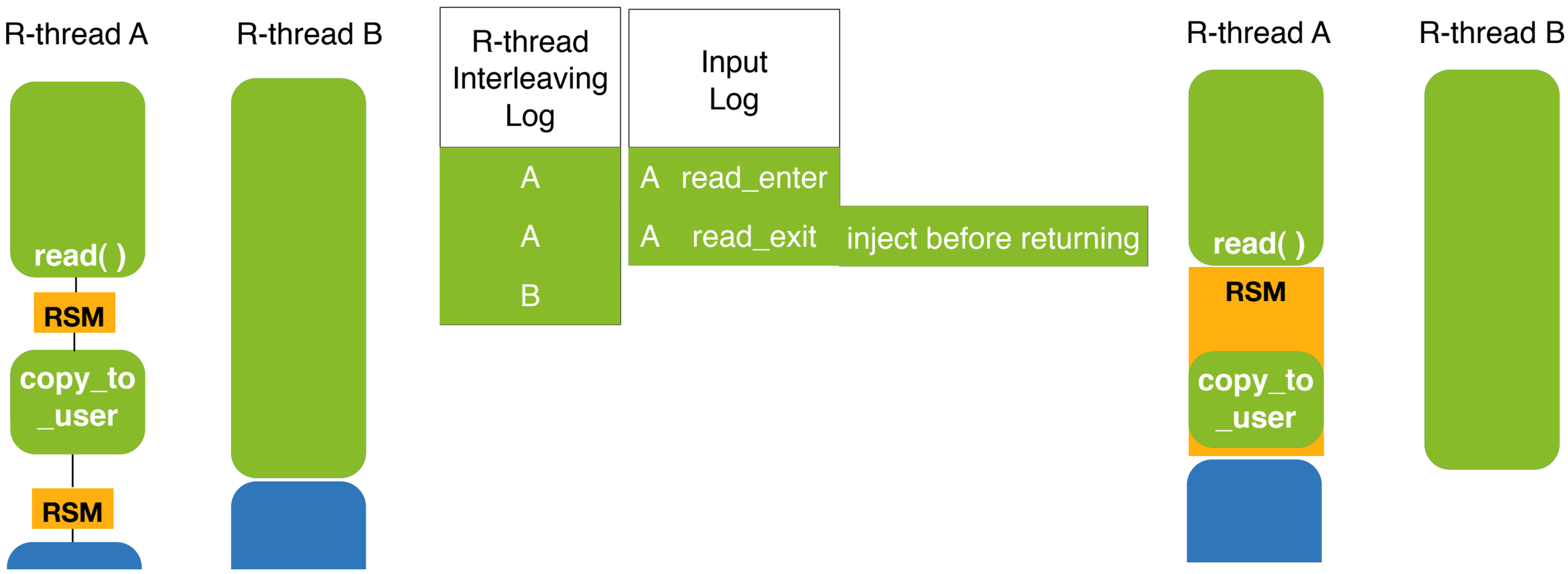
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log



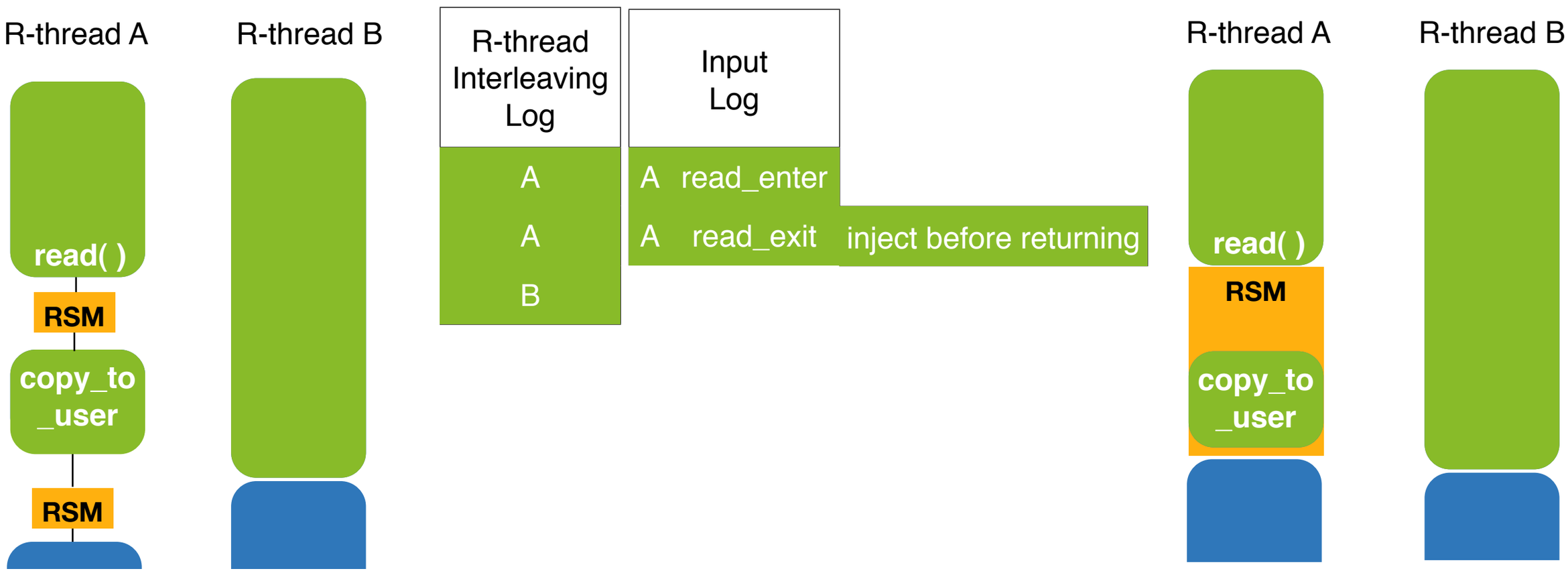
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log



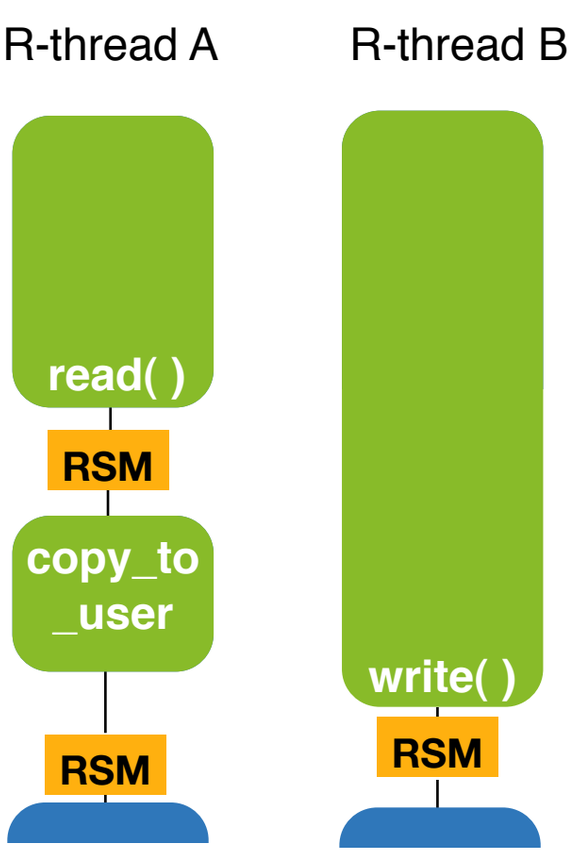
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log



Injecting Data into Spheres (II)

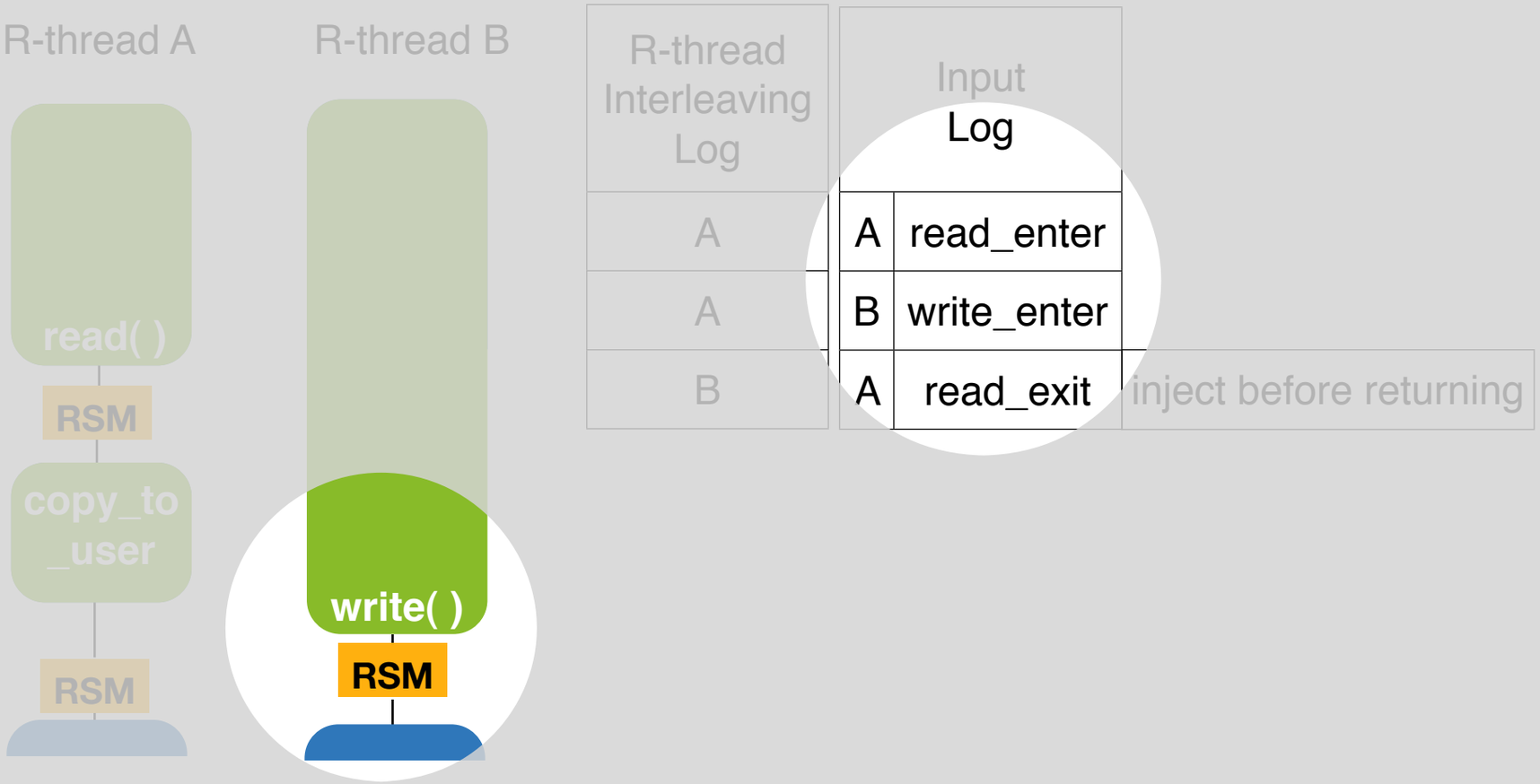
- Look for interactions between R-thread Interleaving Log and Input Log



R-thread Interleaving Log	Input Log		
A	A	read_enter	
A	B	write_enter	
B	A	read_exit	inject before returning

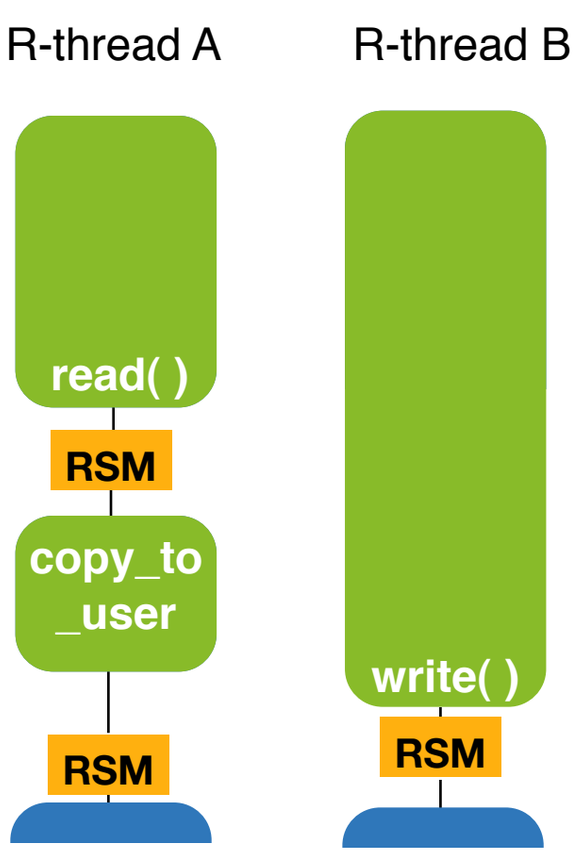
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log



Injecting Data into Spheres (II)

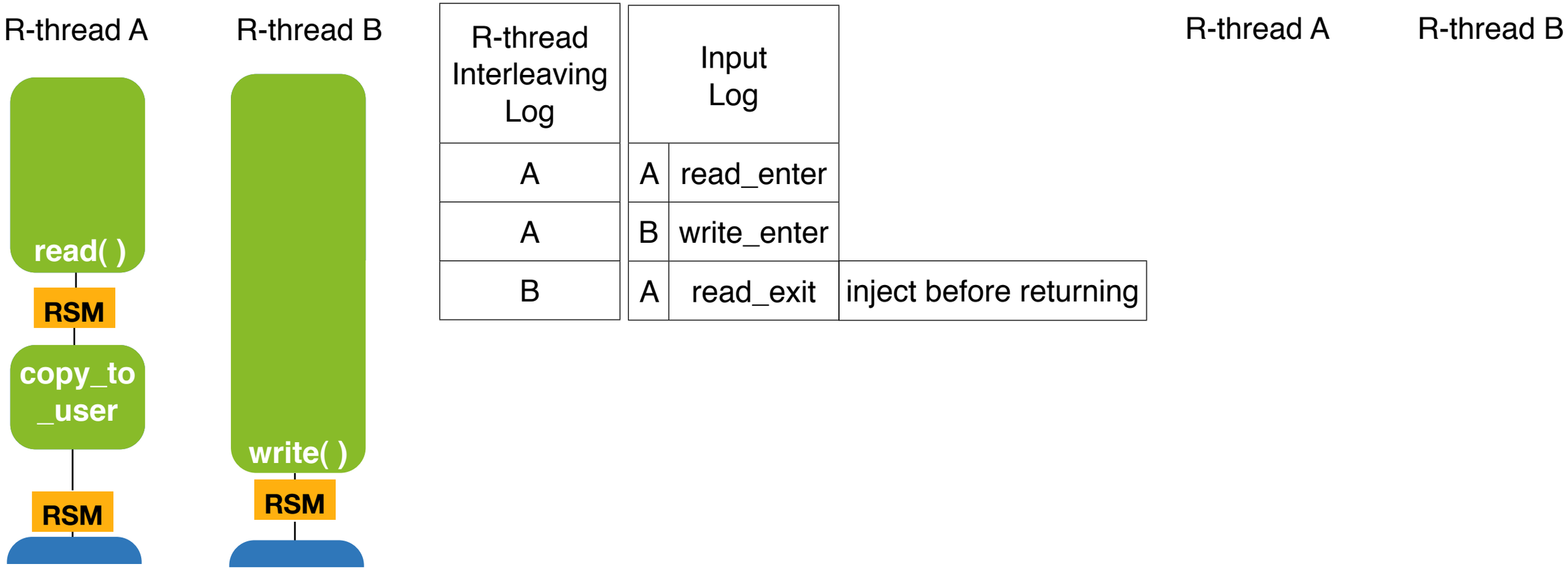
- Look for interactions between R-thread Interleaving Log and Input Log



R-thread Interleaving Log	Input Log		
A	A	read_enter	
A	B	write_enter	
B	A	read_exit	inject before returning

Injecting Data into Spheres (II)

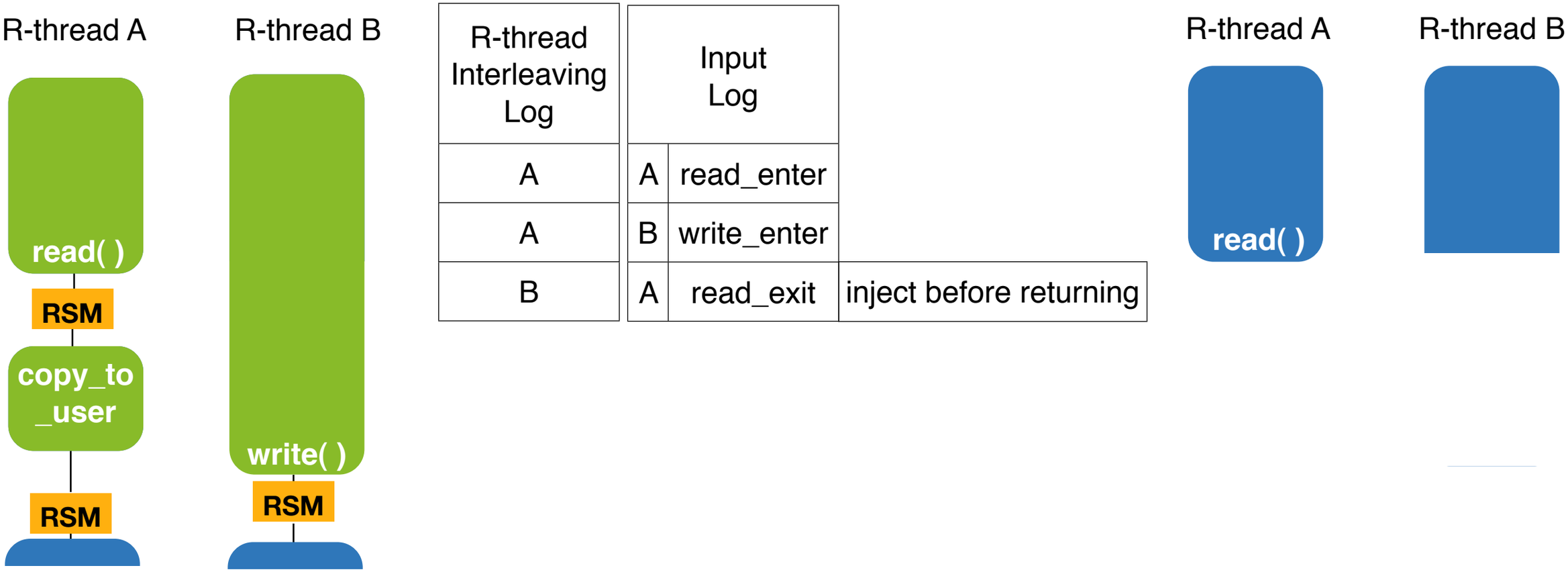
- Look for interactions between R-thread Interleaving Log and Input Log



R-thread Interleaving Log	Input Log		
A	A	read_enter	
A	B	write_enter	
B	A	read_exit	inject before returning

Injecting Data into Spheres (II)

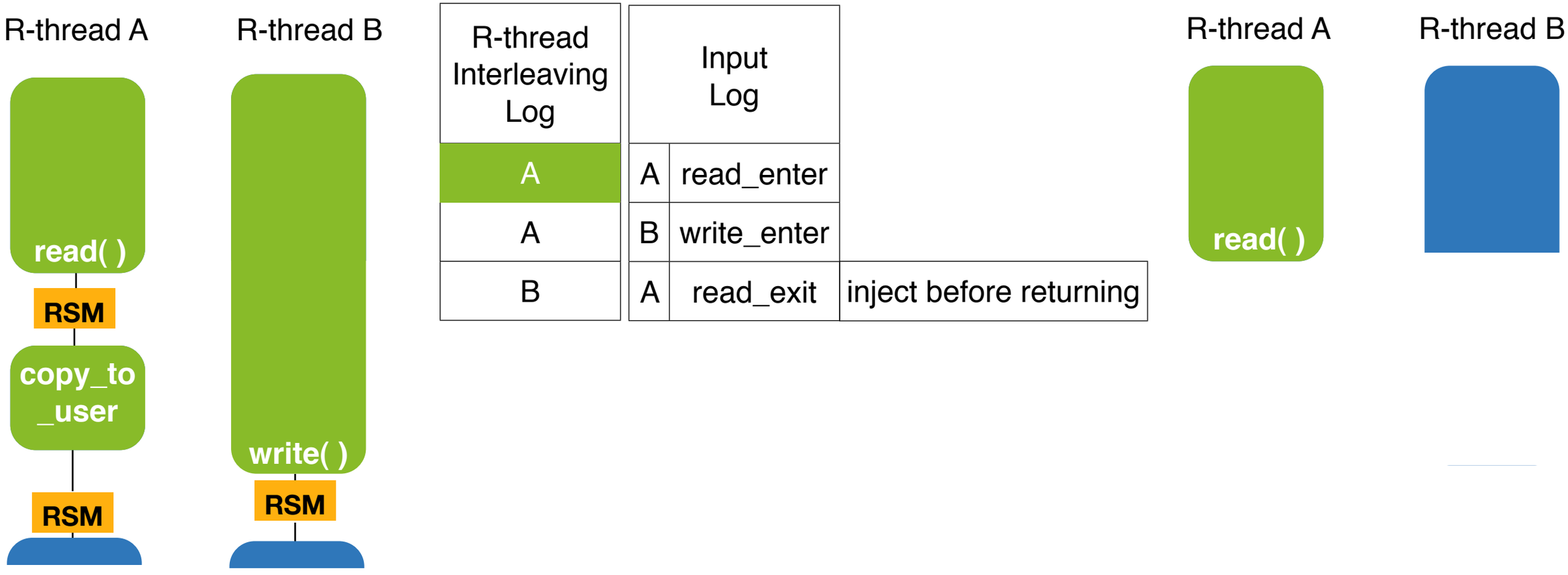
- Look for interactions between R-thread Interleaving Log and Input Log



R-thread Interleaving Log	Input Log		
A	A	read_enter	
A	B	write_enter	
B	A	read_exit	inject before returning

Injecting Data into Spheres (II)

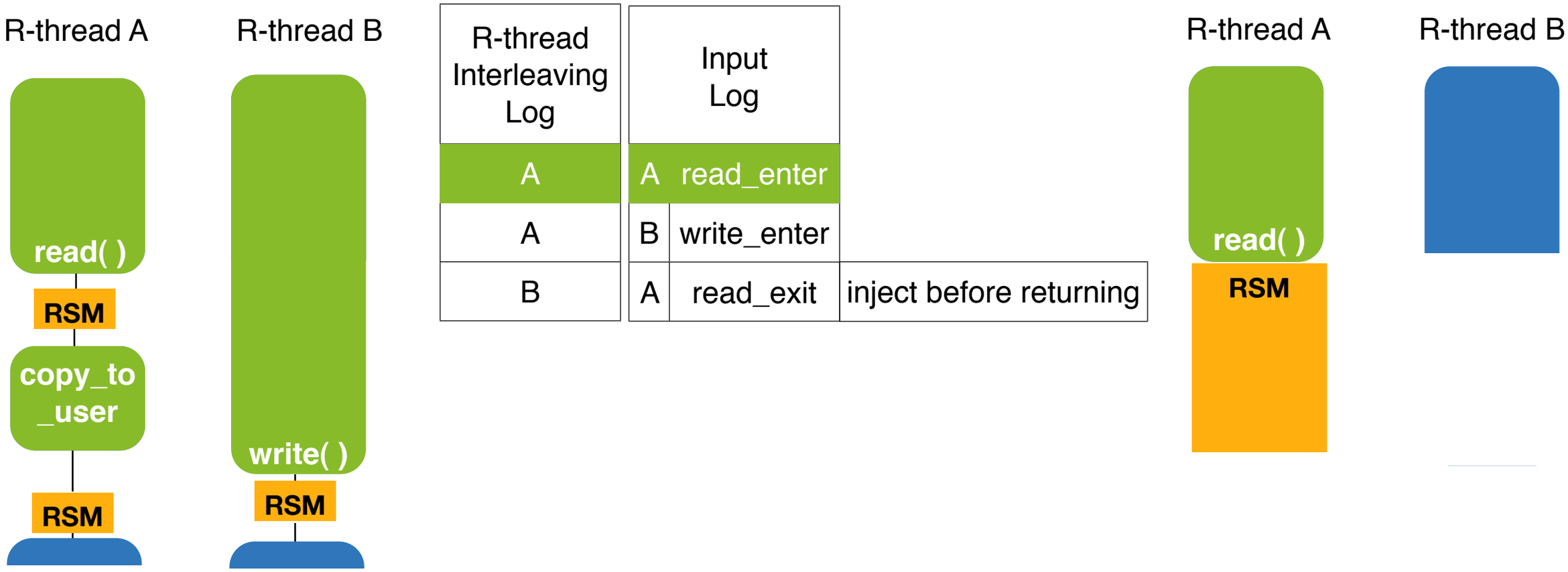
- Look for interactions between R-thread Interleaving Log and Input Log



R-thread Interleaving Log	Input Log		
A	A	read_enter	
A	B	write_enter	
B	A	read_exit	inject before returning

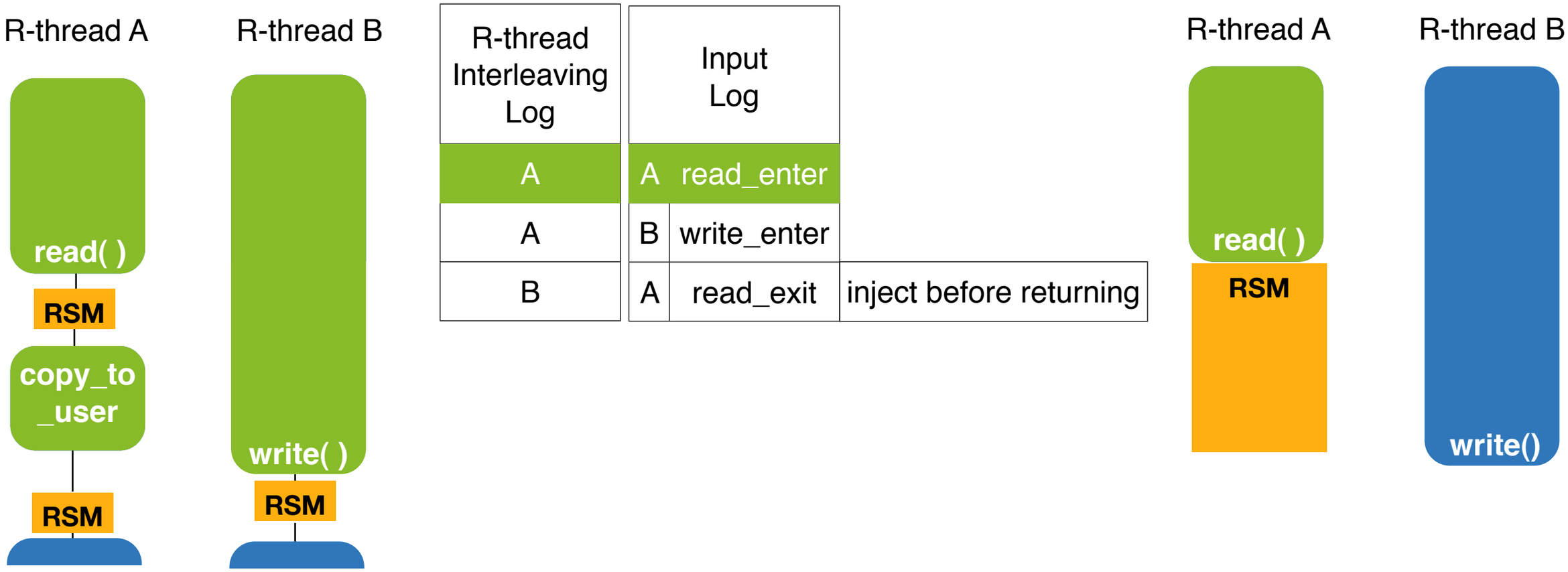
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log



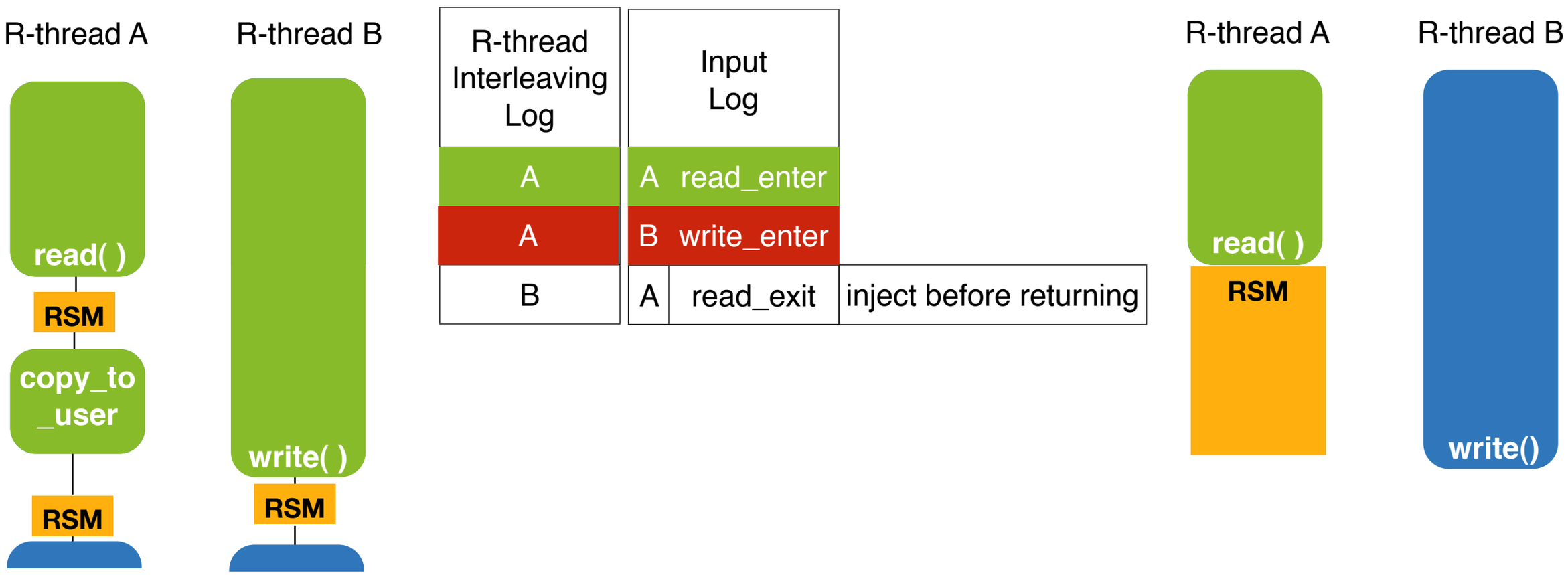
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log



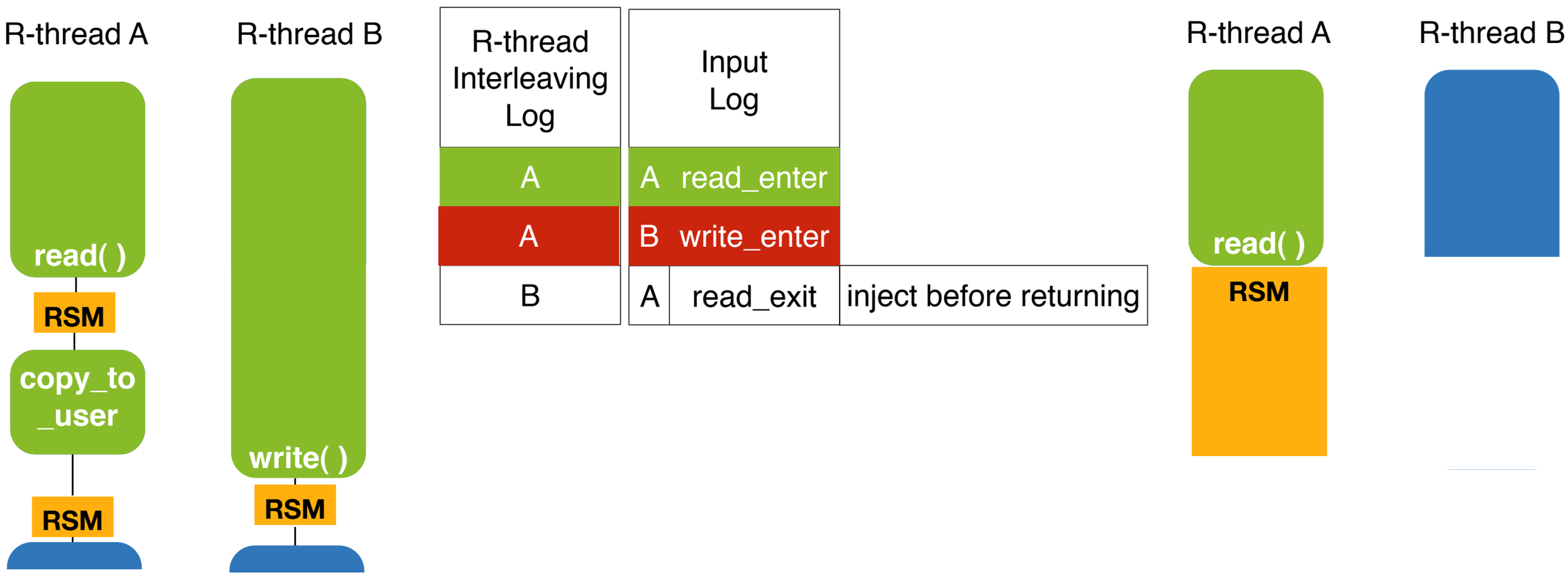
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log



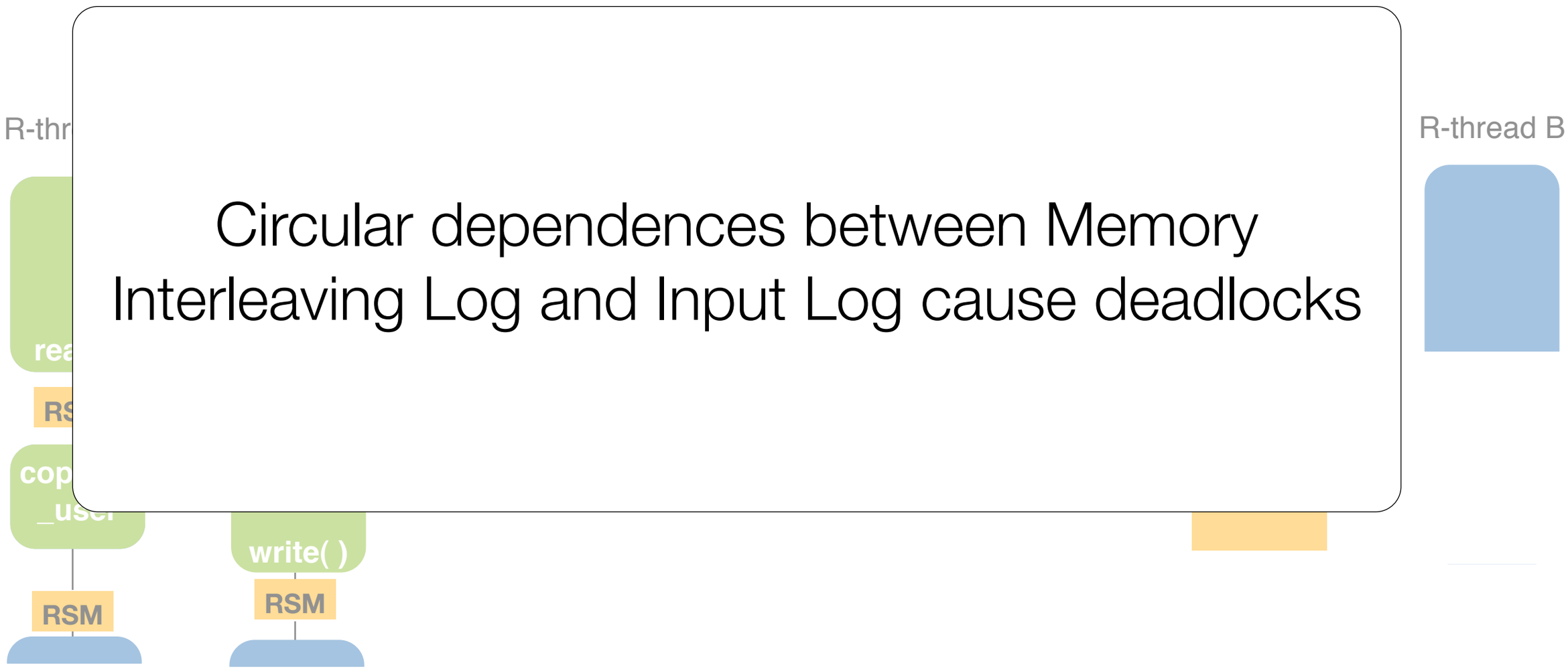
Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log



Injecting Data into Spheres (II)

- Look for interactions between R-thread Interleaving Log and Input Log



Circular dependences between Memory Interleaving Log and Input Log cause deadlocks

Cache Overflows

- CapoOne keeps speculative data in cache until commit
  - A chunk may access more lines mapping to a cache set than ways the set has
  - BulkSC-based system do not allow storing speculative state in memory
- When a cache would overflow, CapoOne must commit current chunk
  - Non-deterministic event: must log chunk size

Cache Overflows

Cache Overflows

Original Execution	Chunk Size Log
R-thread R1	

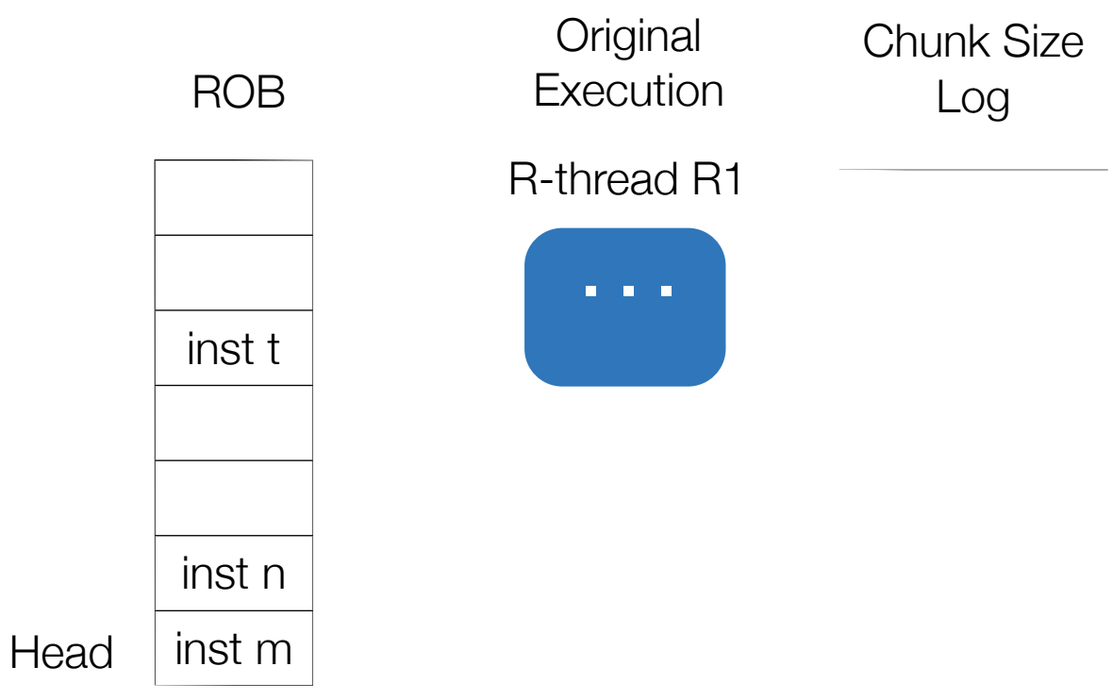
Cache Overflows

Original Execution

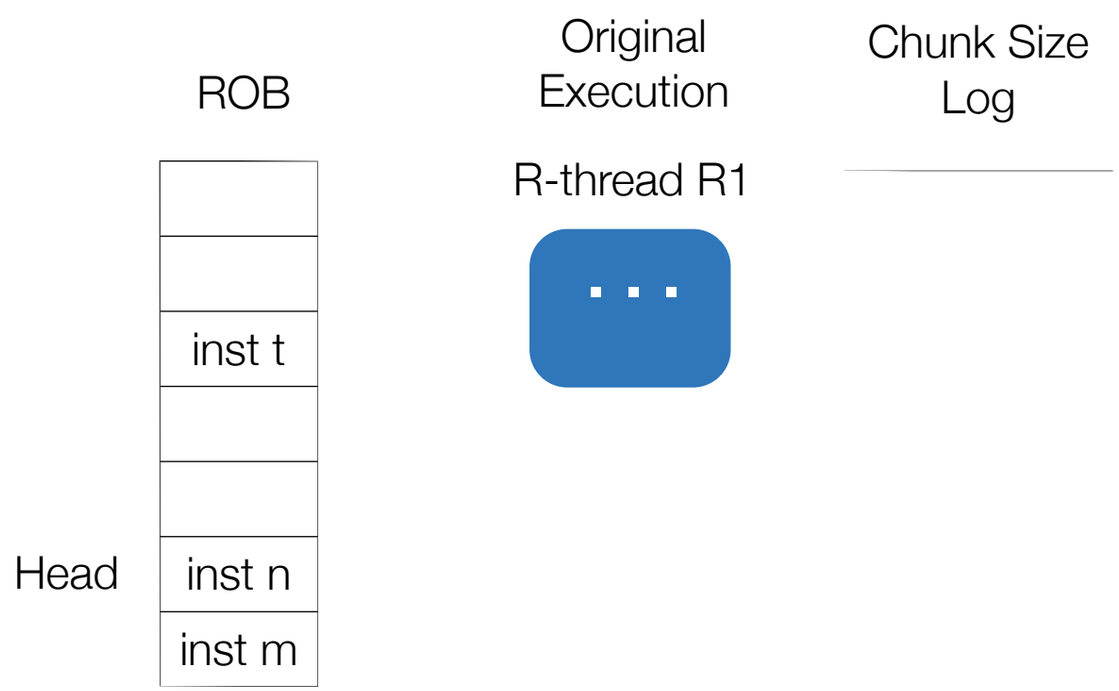
R-thread R1

Chunk Size Log

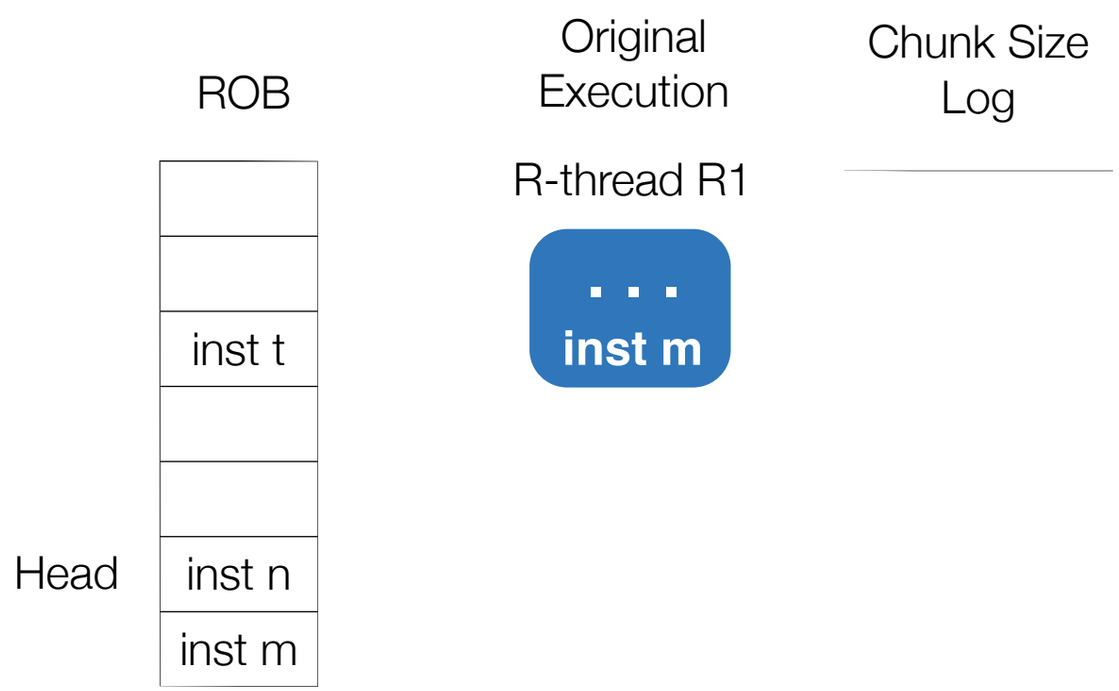
Cache Overflows



Cache Overflows

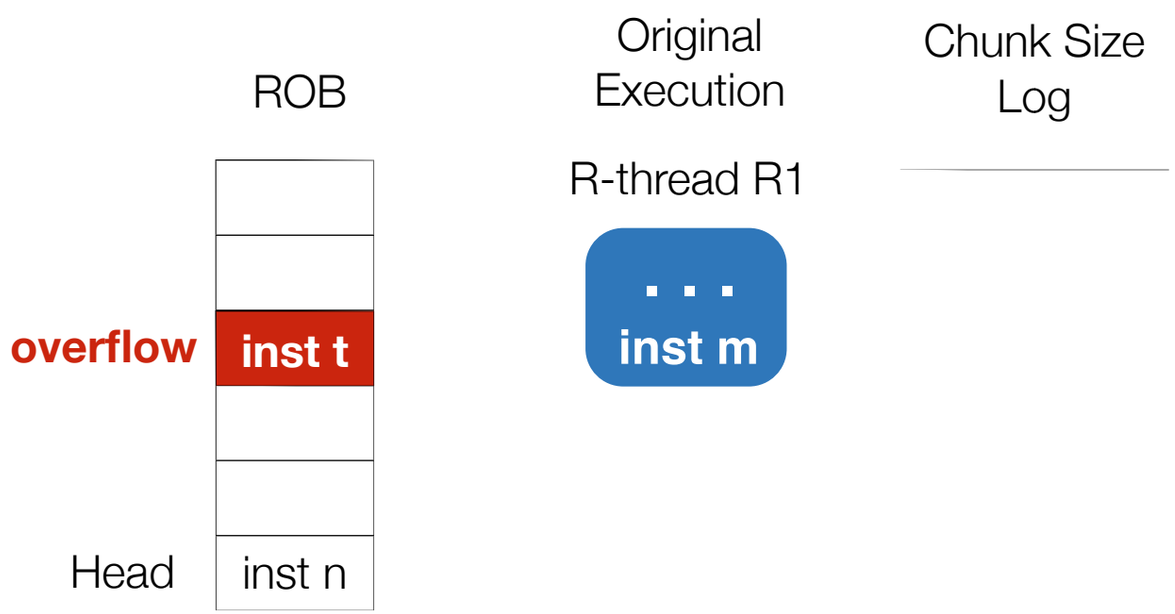


Cache Overflows



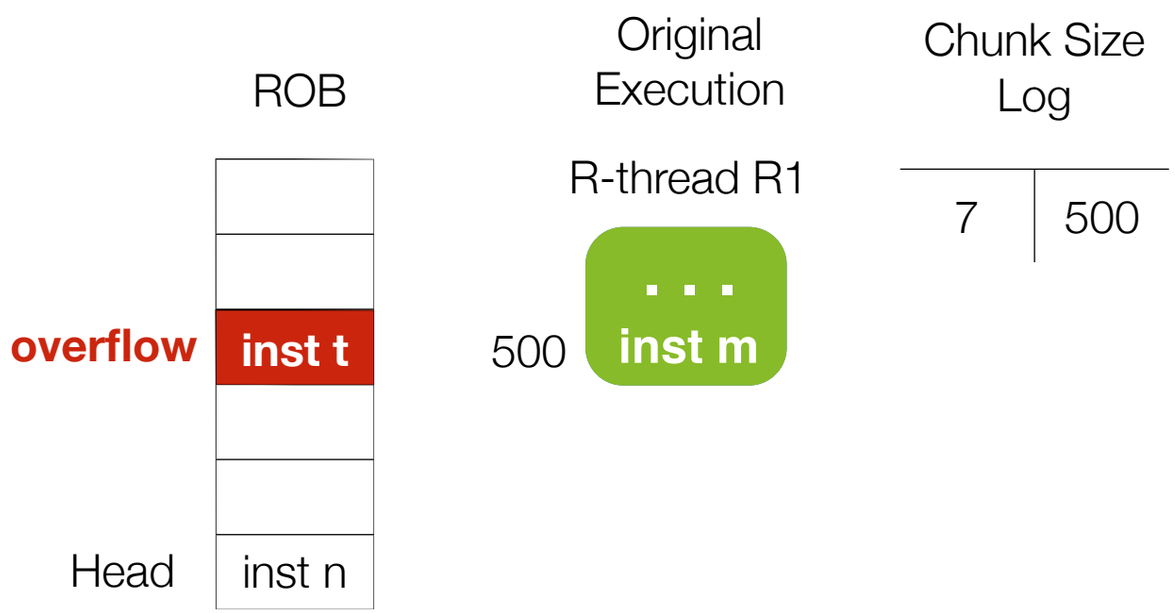
Cache Overflows

Cache Overflows



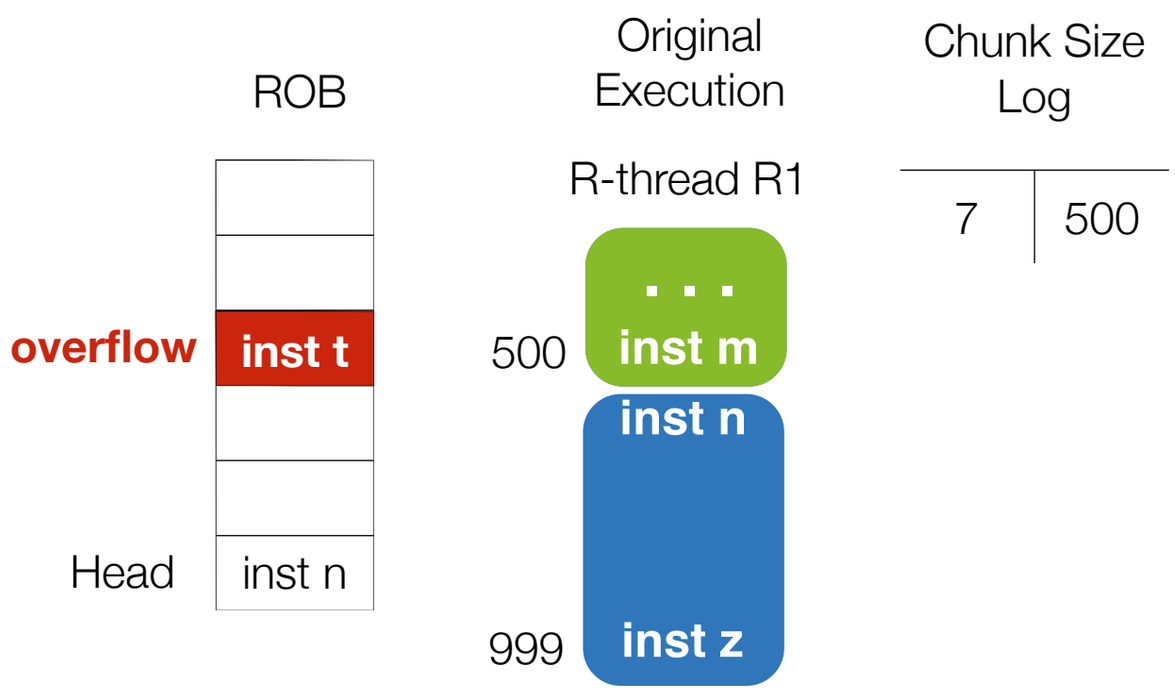
- The instruction causing the overflow might not be at the head of ROB

Cache Overflows



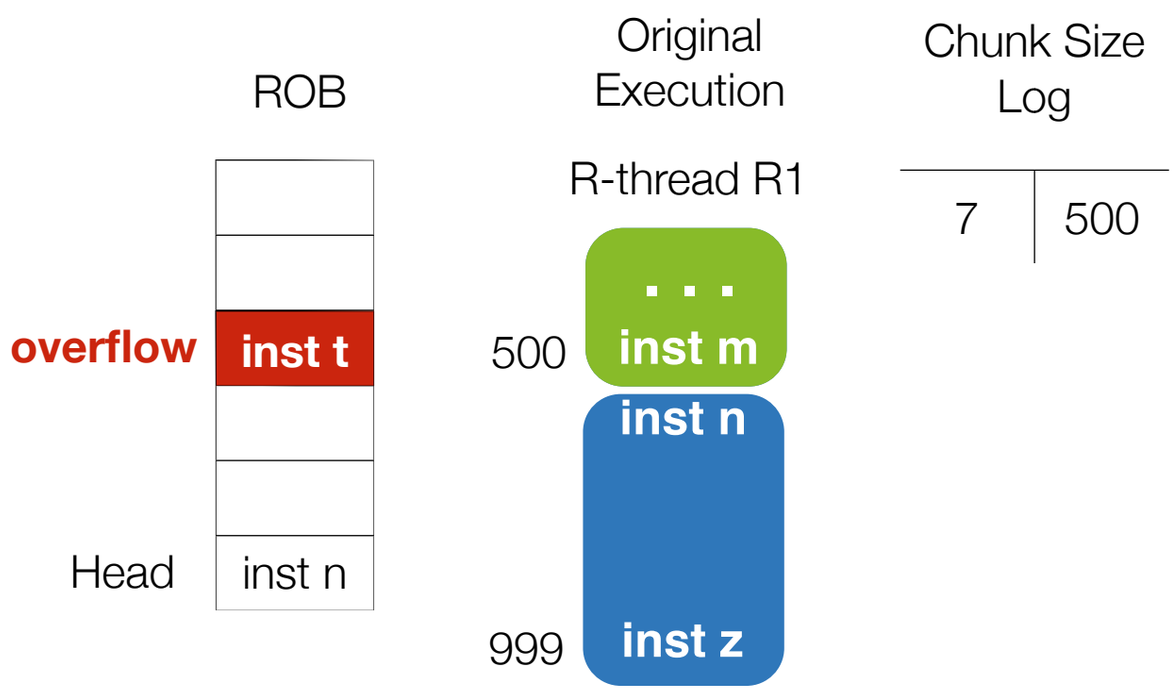
- The instruction causing the overflow might not be at the head of ROB

Cache Overflows



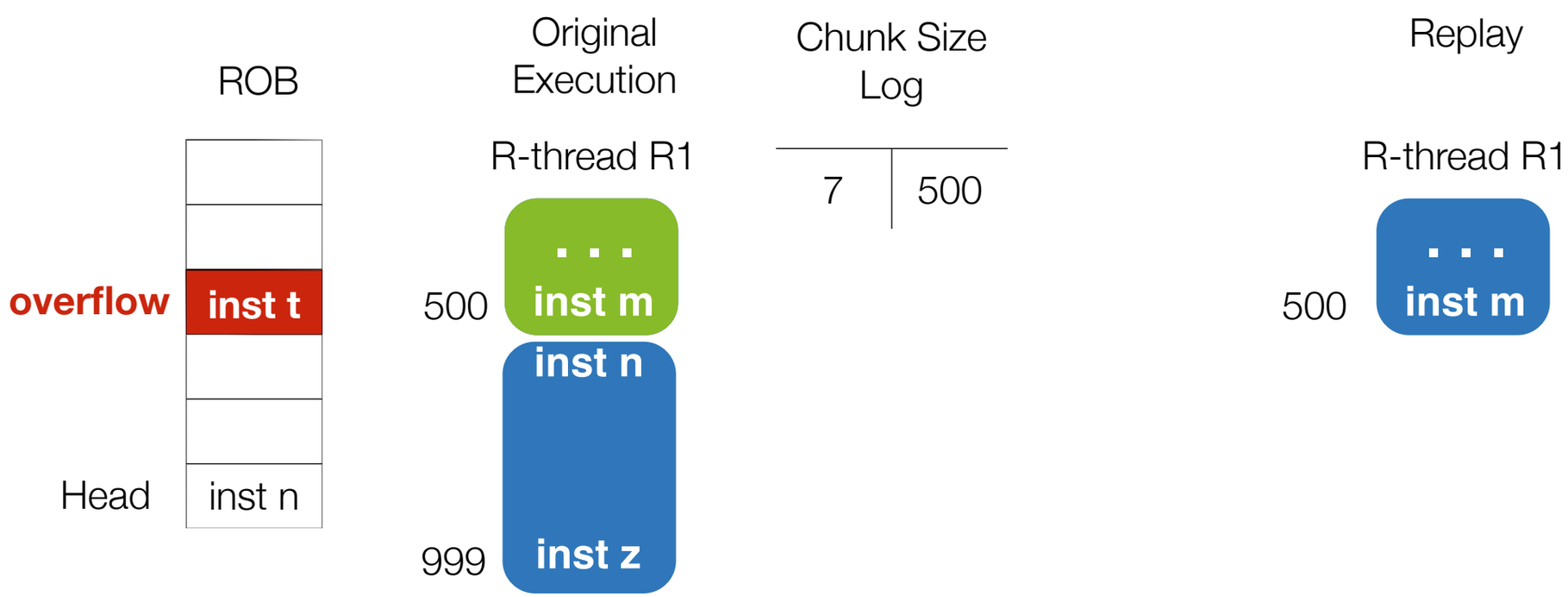
- The instruction causing the overflow might not be at the head of ROB

Cache Overflows



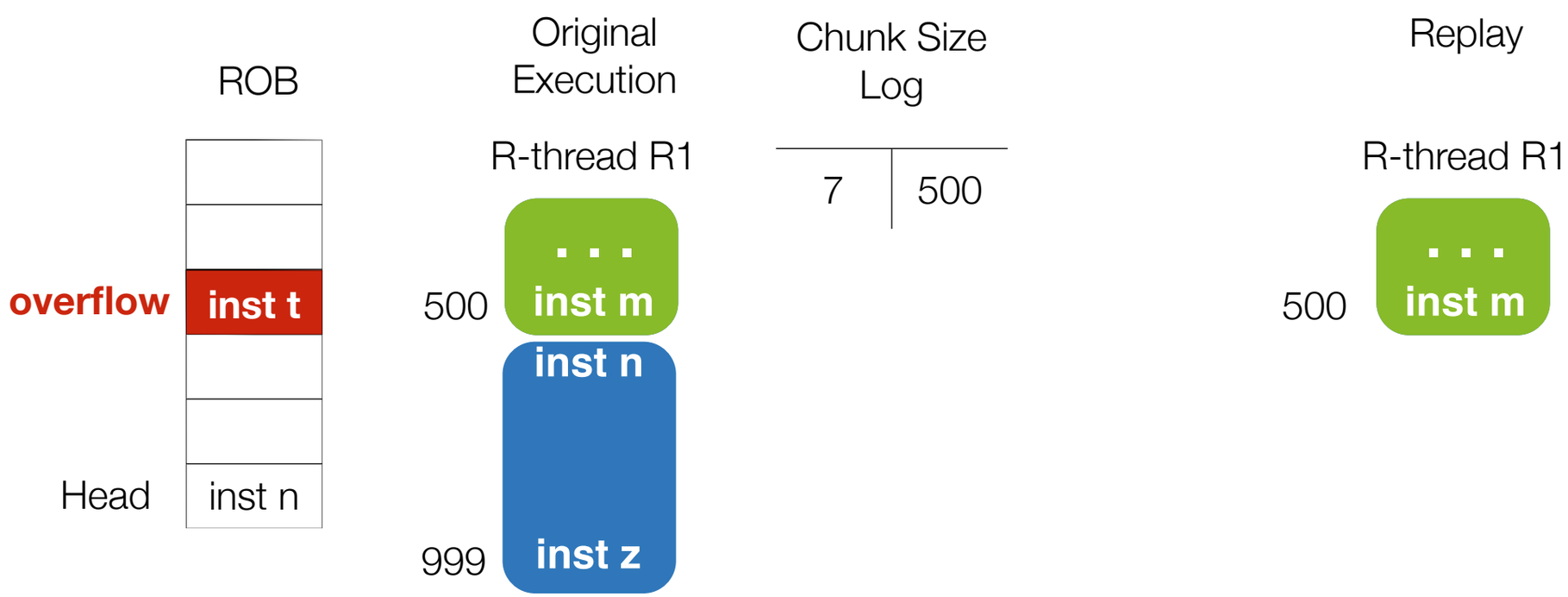
- The instruction causing the overflow might not be at the head of ROB
- Can happen during initial execution and/or during replay

Cache Overflows



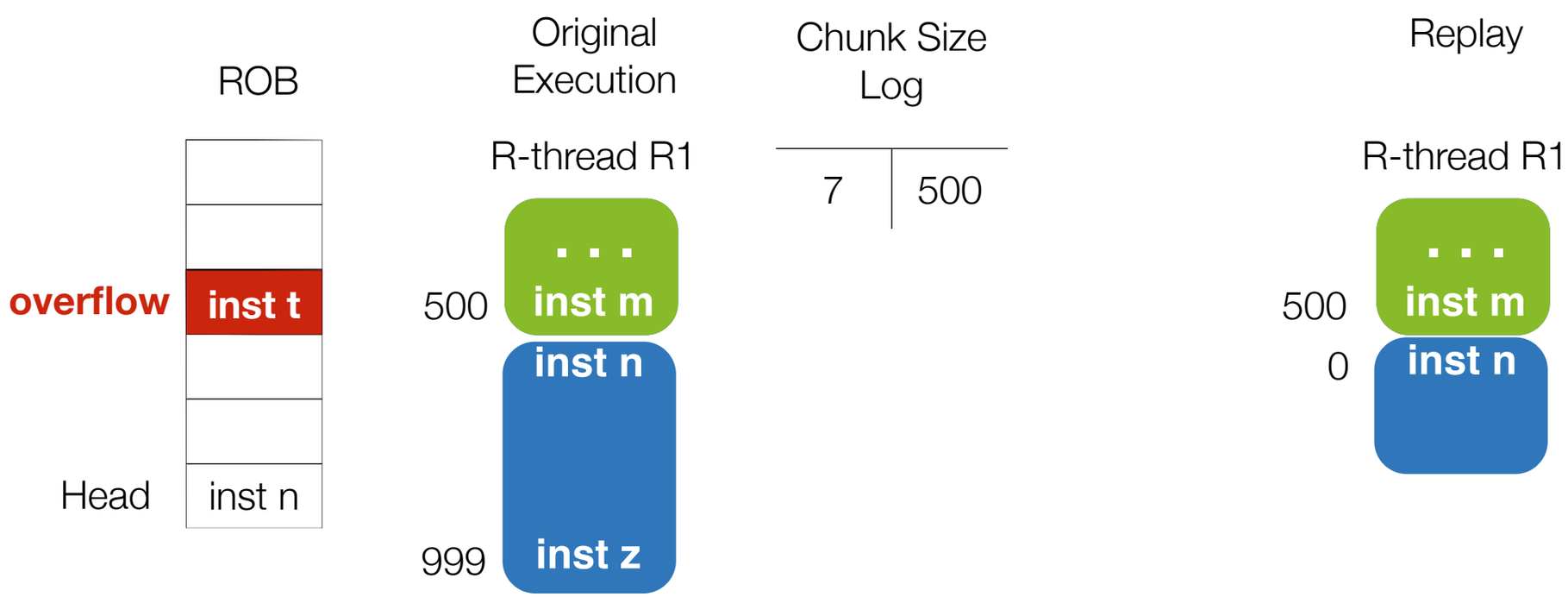
- The instruction causing the overflow might not be at the head of ROB
- Can happen during initial execution and/or during replay

Cache Overflows



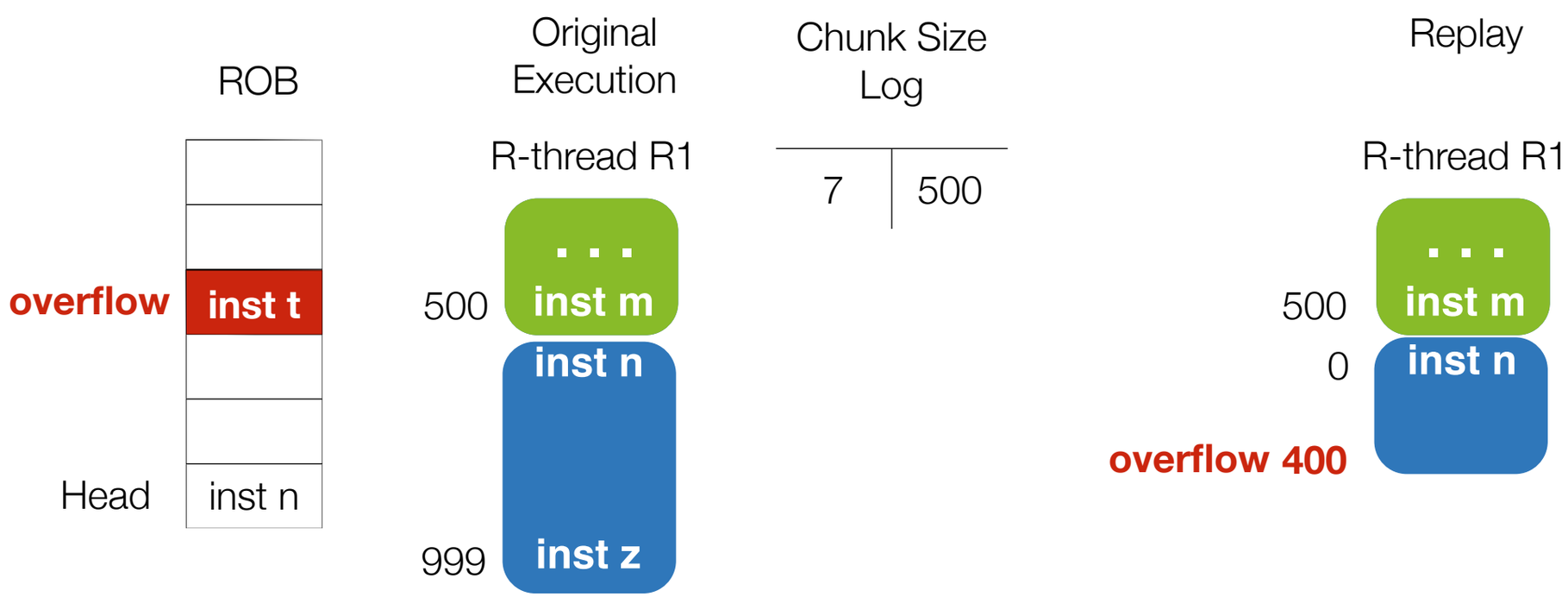
- The instruction causing the overflow might not be at the head of ROB
- Can happen during initial execution and/or during replay

Cache Overflows



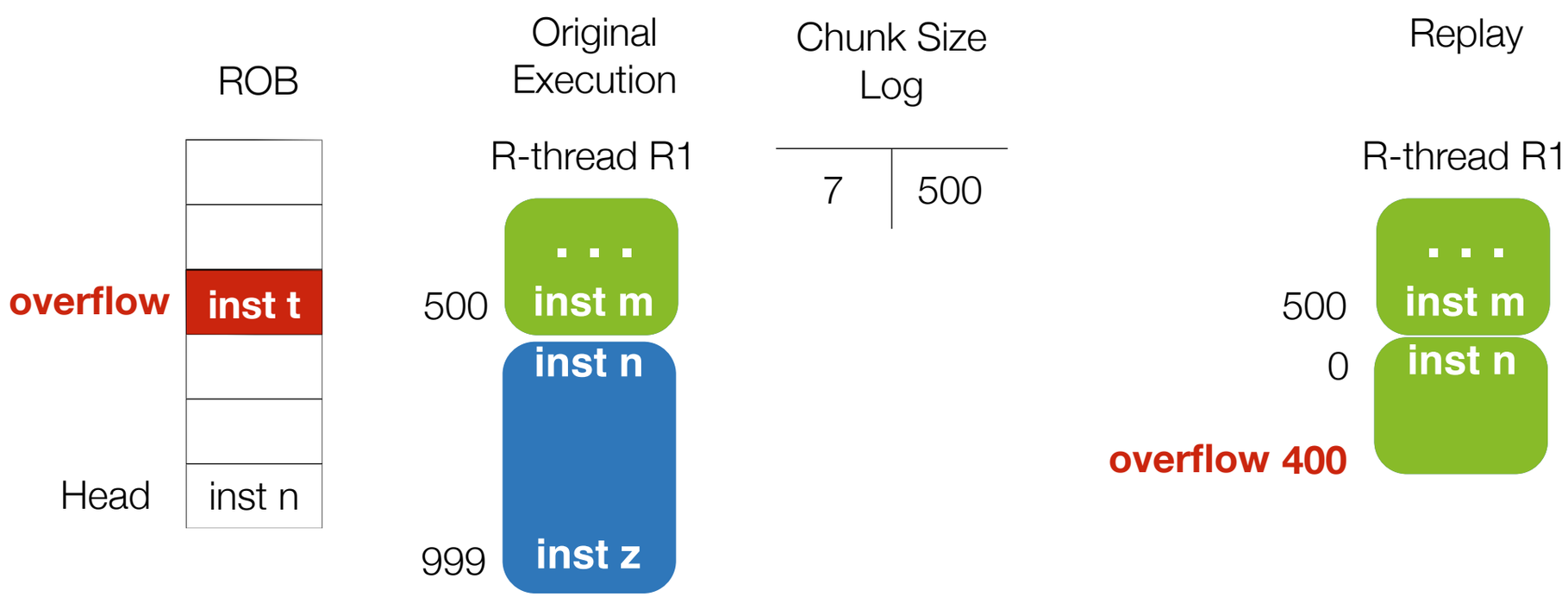
- The instruction causing the overflow might not be at the head of ROB
- Can happen during initial execution and/or during replay

Cache Overflows



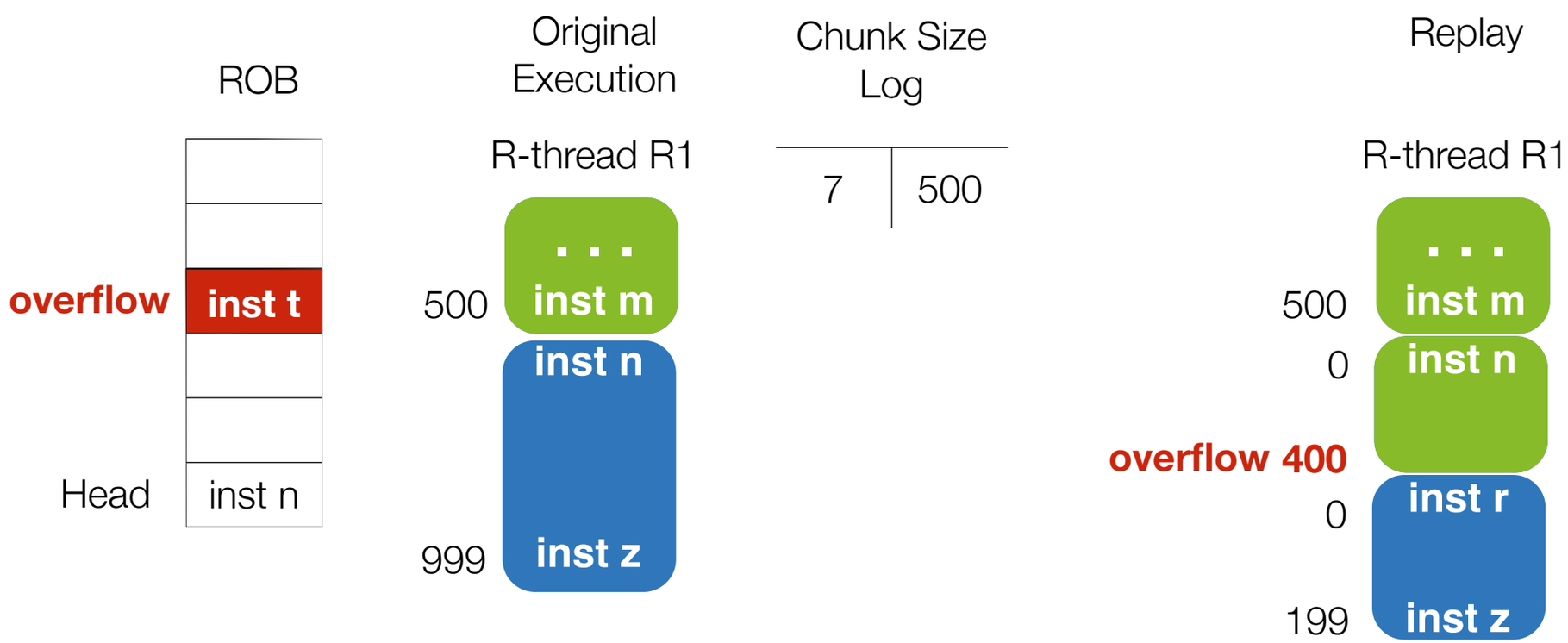
- The instruction causing the overflow might not be at the head of ROB
- Can happen during initial execution and/or during replay

Cache Overflows



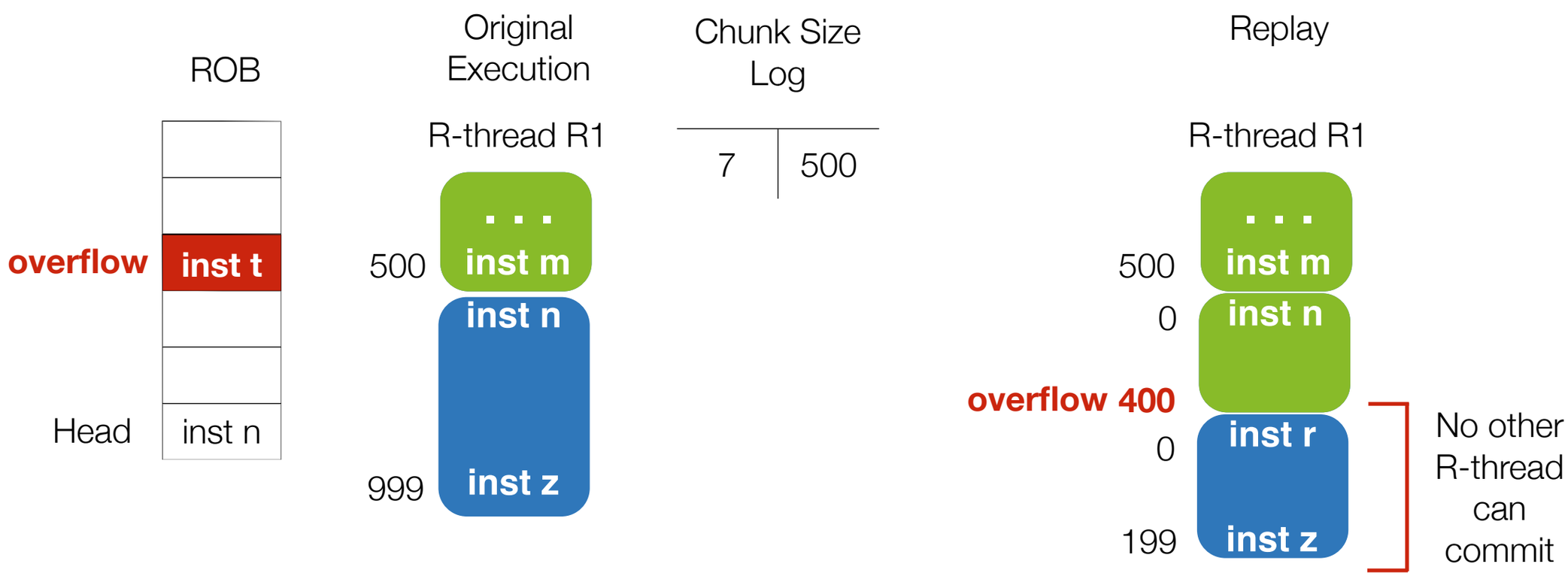
- The instruction causing the overflow might not be at the head of ROB
- Can happen during initial execution and/or during replay

Cache Overflows



- The instruction causing the overflow might not be at the head of ROB
- Can happen during initial execution and/or during replay

Cache Overflows



- The instruction causing the overflow might not be at the head of ROB
- Can happen during initial execution and/or during replay

Cache Overflows

ROB | Original Execution | Chunk Size Log | Replay

over

H

The instruction that caused cache overflow might not be the first instruction of next chunk

HW must ensure that chunk boundaries are consistent

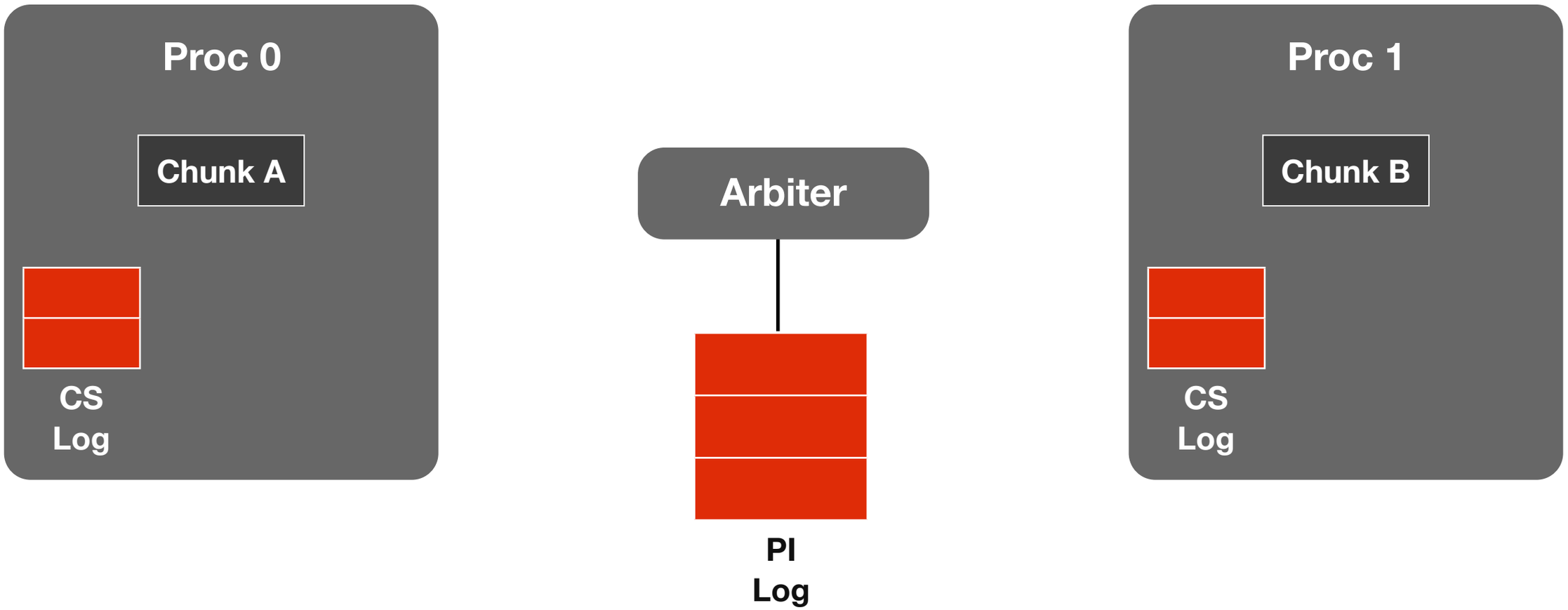
- The instruction causing the overflow might not be at the head of ROB
- Can happen during initial execution and/or during replay

Lessons Learned During the Development of the CapoOne Deterministic Multiprocessor Replay System

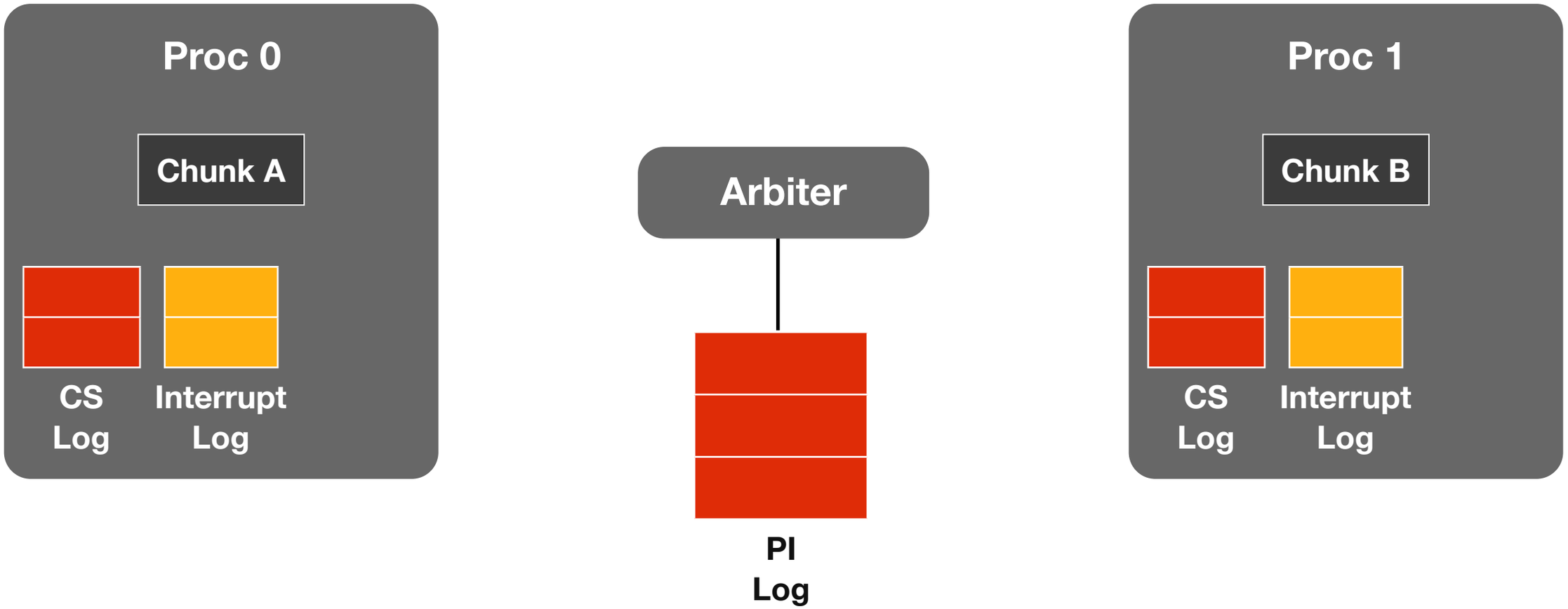
Pablo Montesinos, Matthew Hicks, Wonsun Ahn, Samuel T. King and Josep Torrellas

Department of Computer Science
University of Illinois at Urbana-Champaign

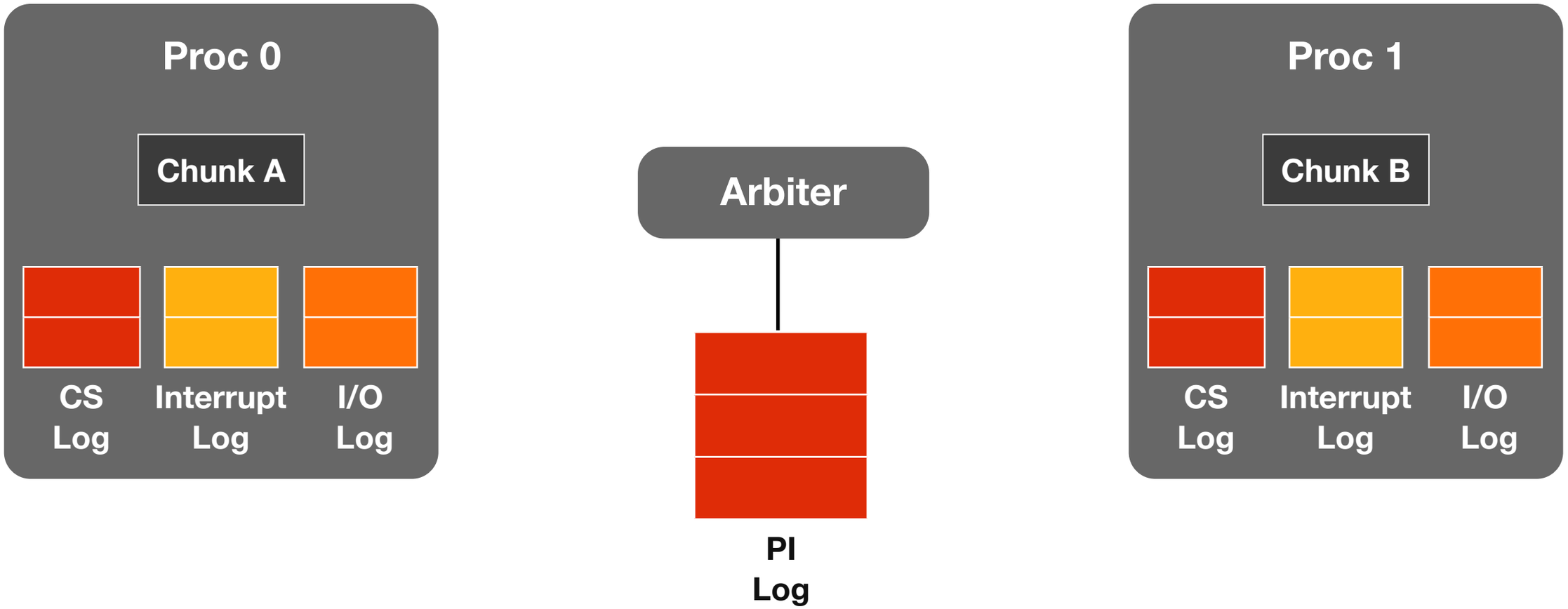
Overall DeLorean System



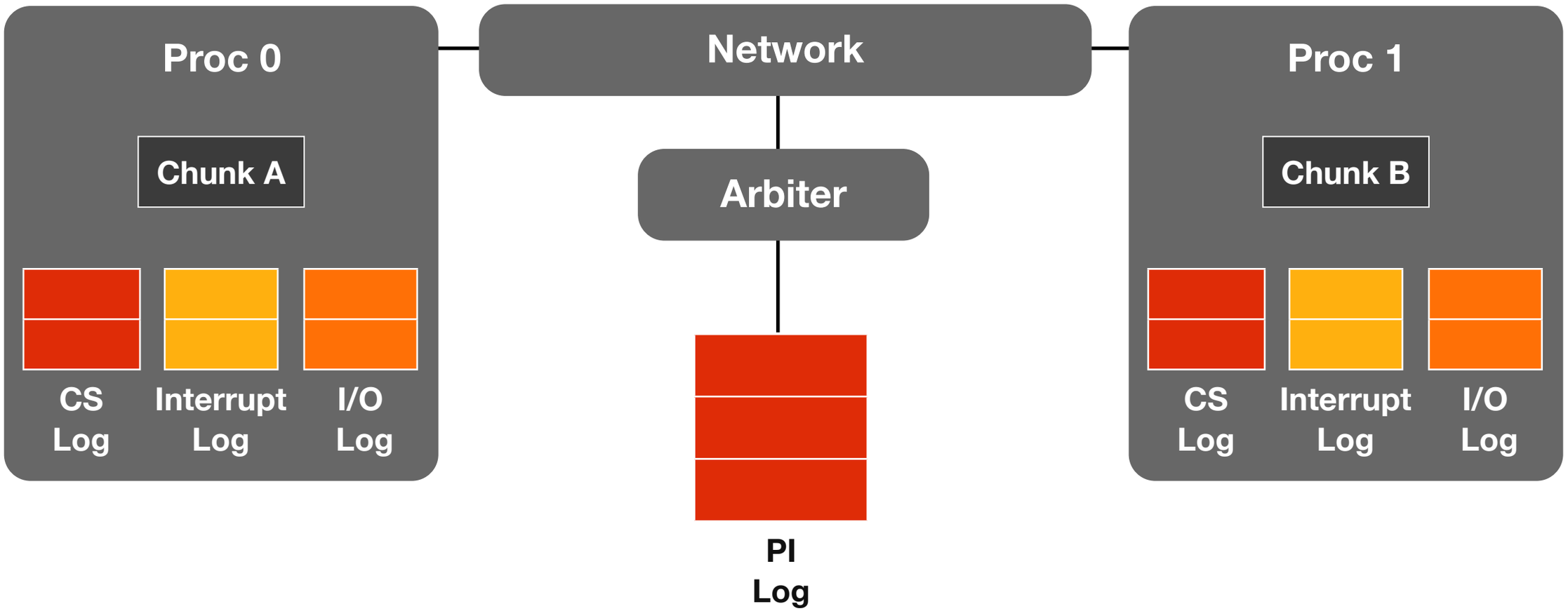
Overall DeLorean System



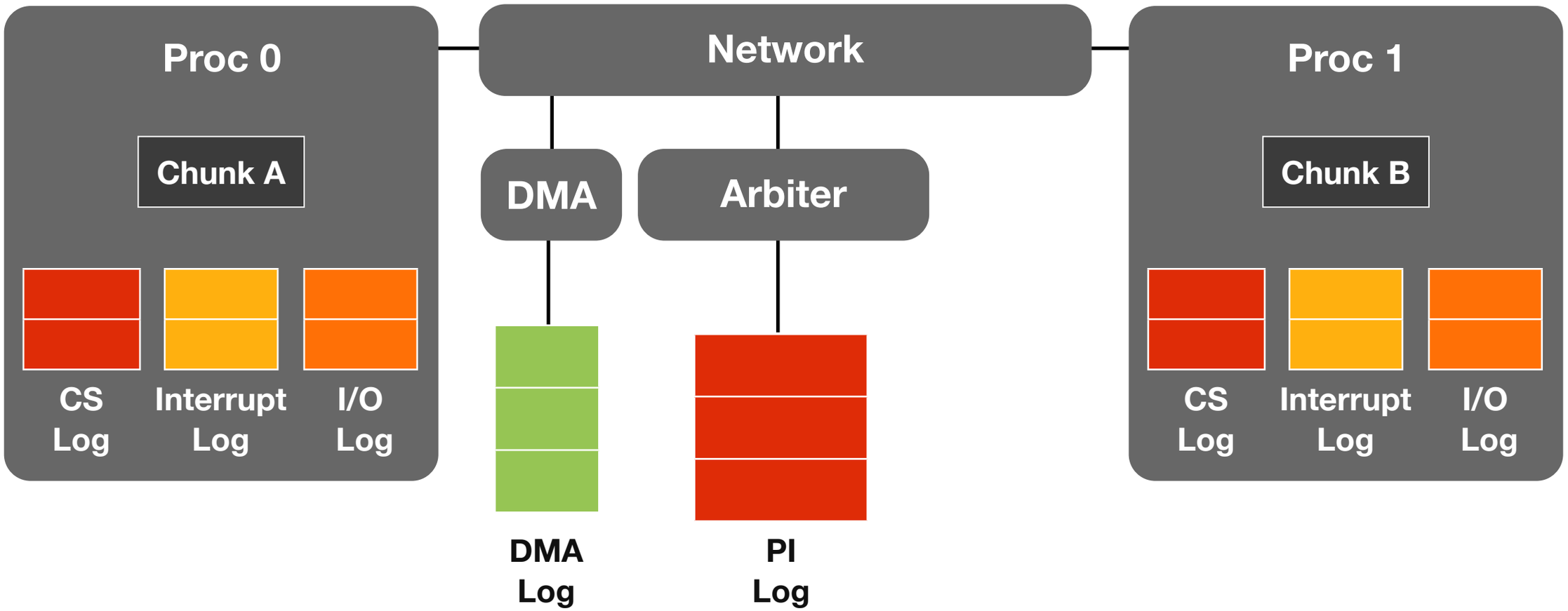
Overall DeLorean System



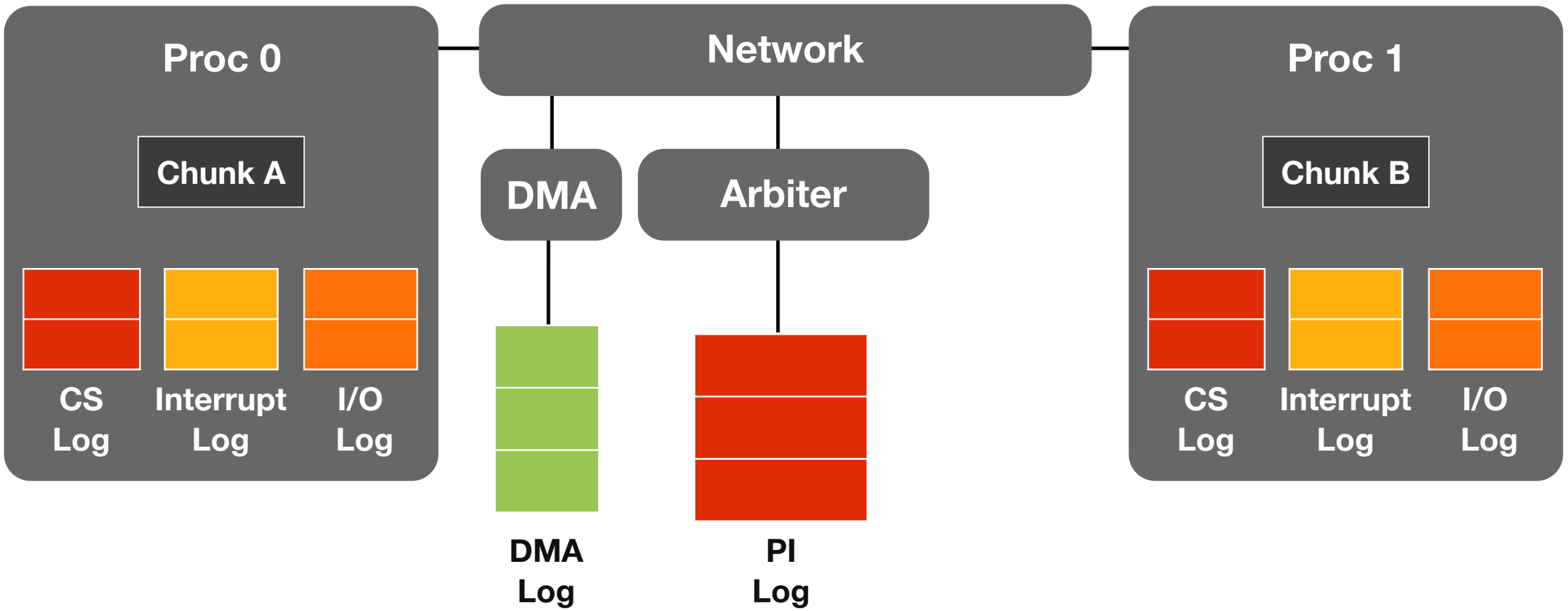
Overall DeLorean System



Overall DeLorean System



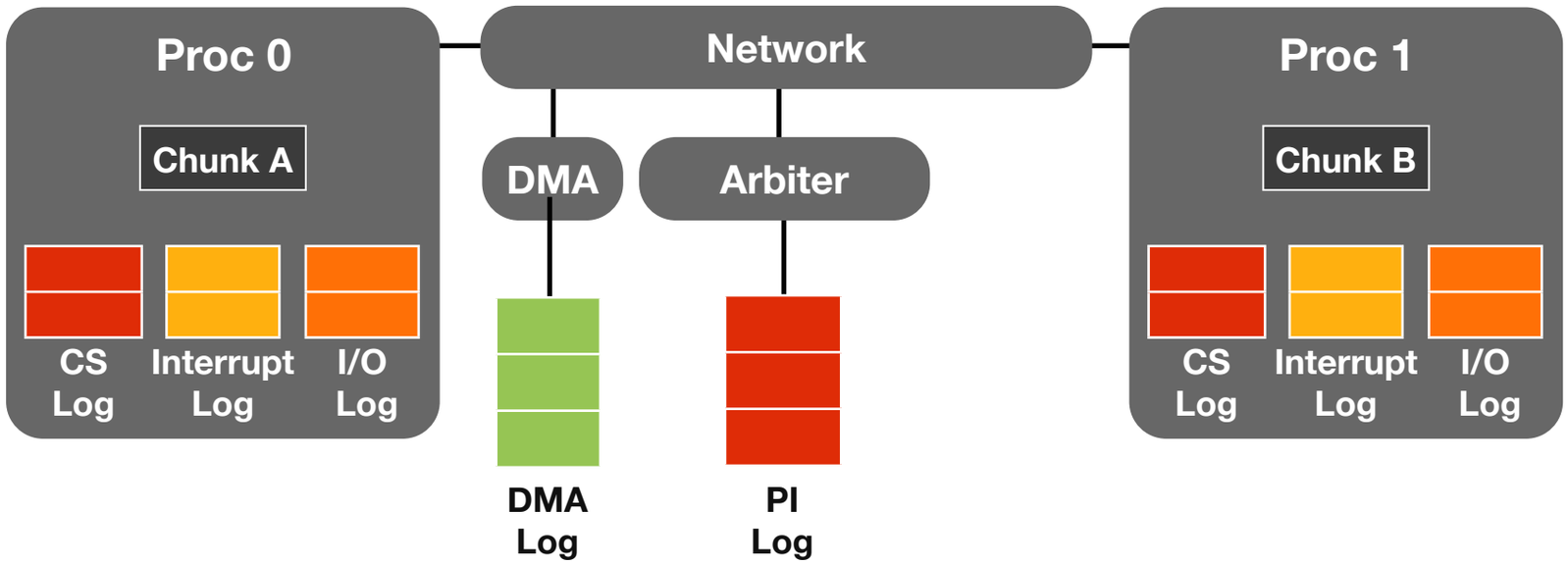
Overall DeLorean System



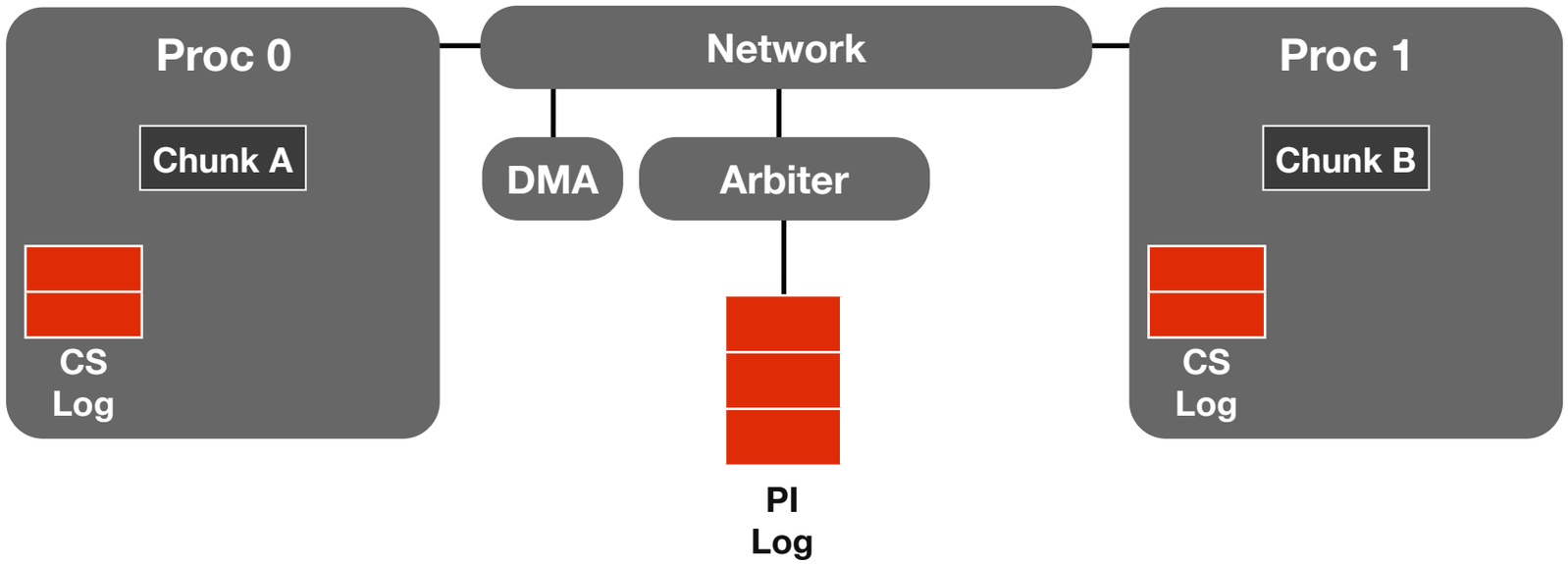
- Interrupt, I/O and DMA logs are common to other HW-based schemes

CapoOne: HW Implementation

CapoOne: HW Implementation

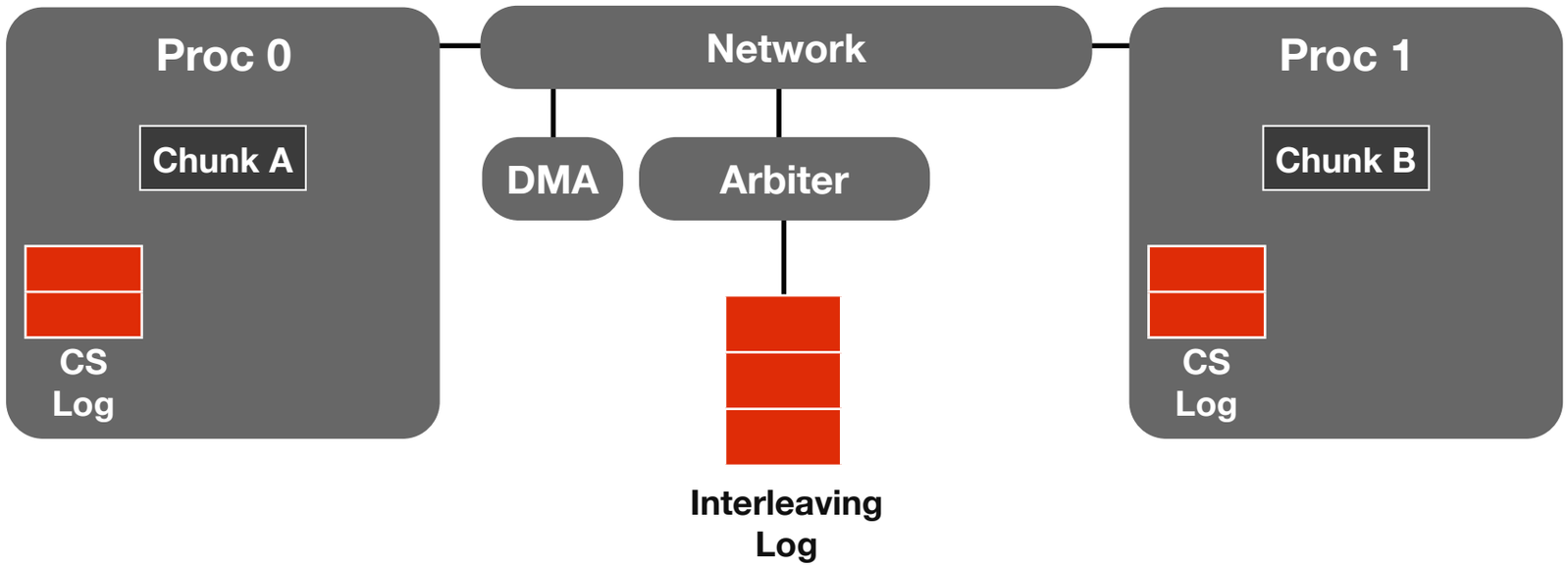


CapoOne: HW Implementation



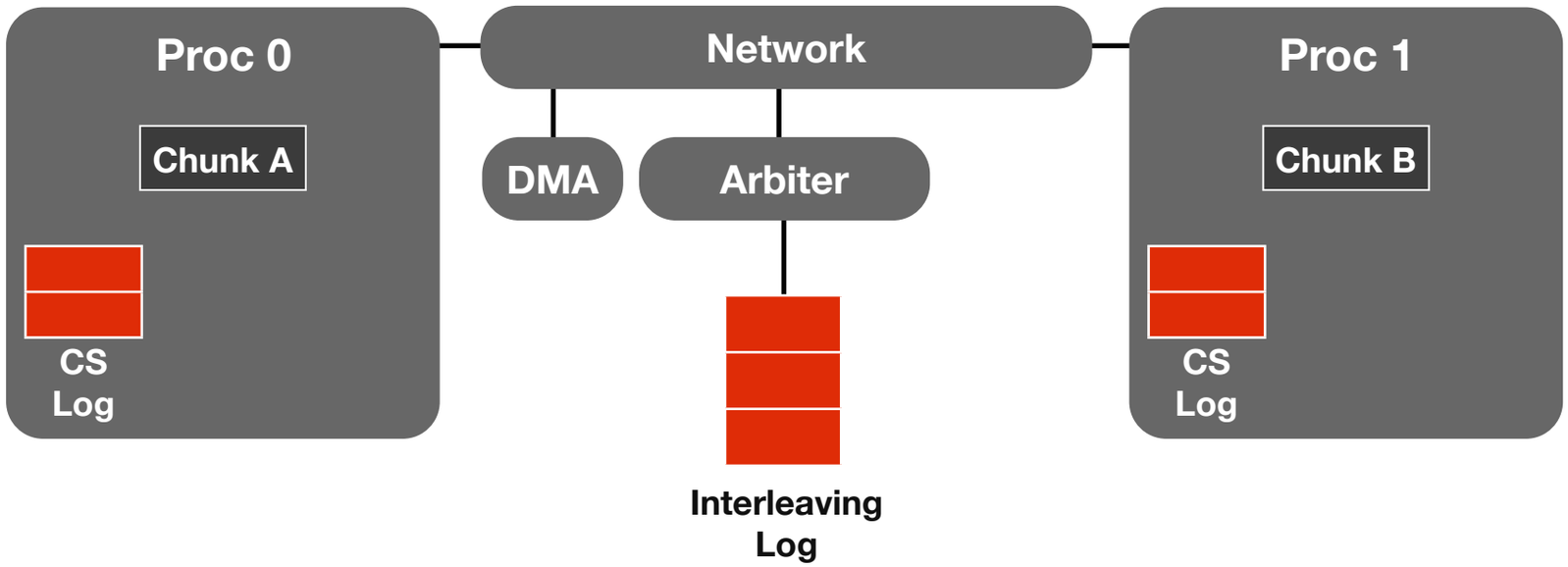
- No need for DeLorean's Interrupt Log, DMA Log nor I/O Log

CapoOne: HW Implementation



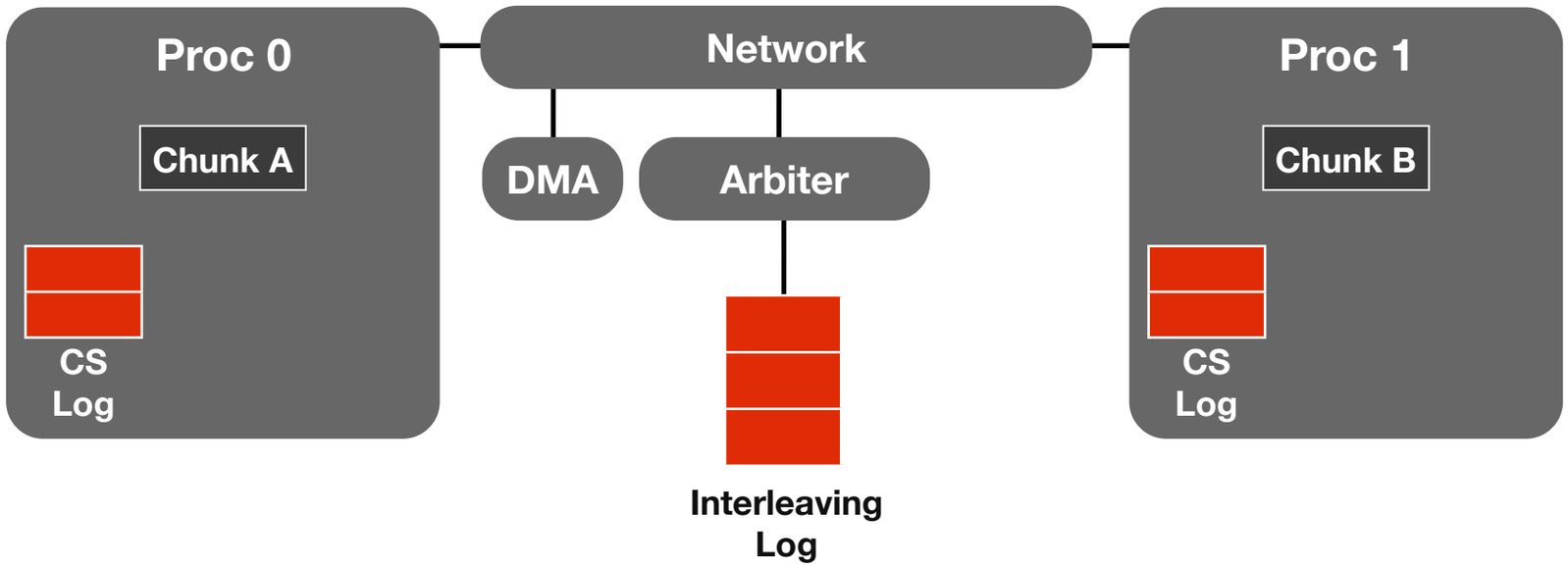
- No need for DeLorean's Interrupt Log, DMA Log nor I/O Log
  - PI Log becomes the per-sphere Interleaving Log

CapoOne: HW Implementation



- No need for DeLorean's Interrupt Log, DMA Log nor I/O Log
  - PI Log becomes the per-sphere Interleaving Log
  - CS Log becomes a per-R-thread Log

CapoOne: HW Implementation



- No need for DeLorean's Interrupt Log, DMA Log nor I/O Log
  - PI Log becomes the per-sphere Interleaving Log
  - CS Log becomes a per-R-thread Log

- Chunks only have instructions from one application (or the kernel)

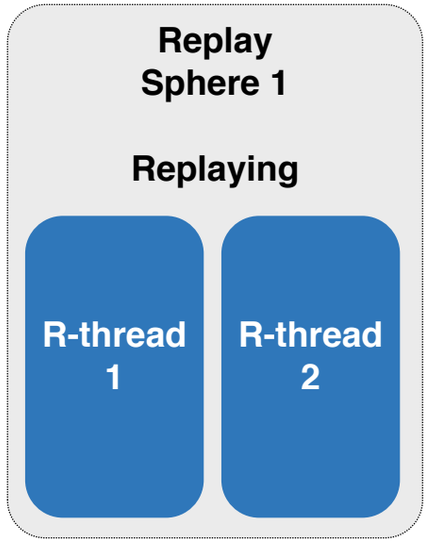
Emulating vs Re-Executing System Calls

Emulating vs Re-Executing System Calls

- During replay, the RSM emulates most system calls:

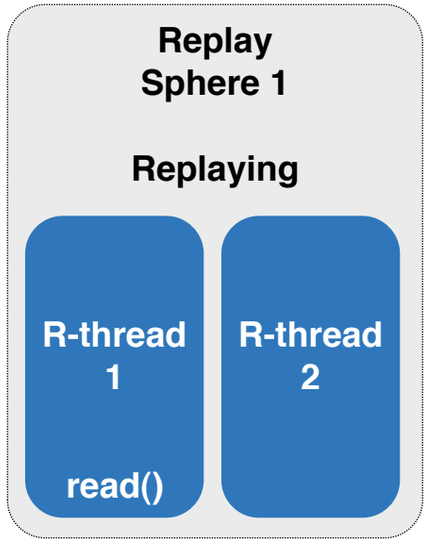
Emulating vs Re-Executing System Calls

- During replay, the RSM emulates most system calls:



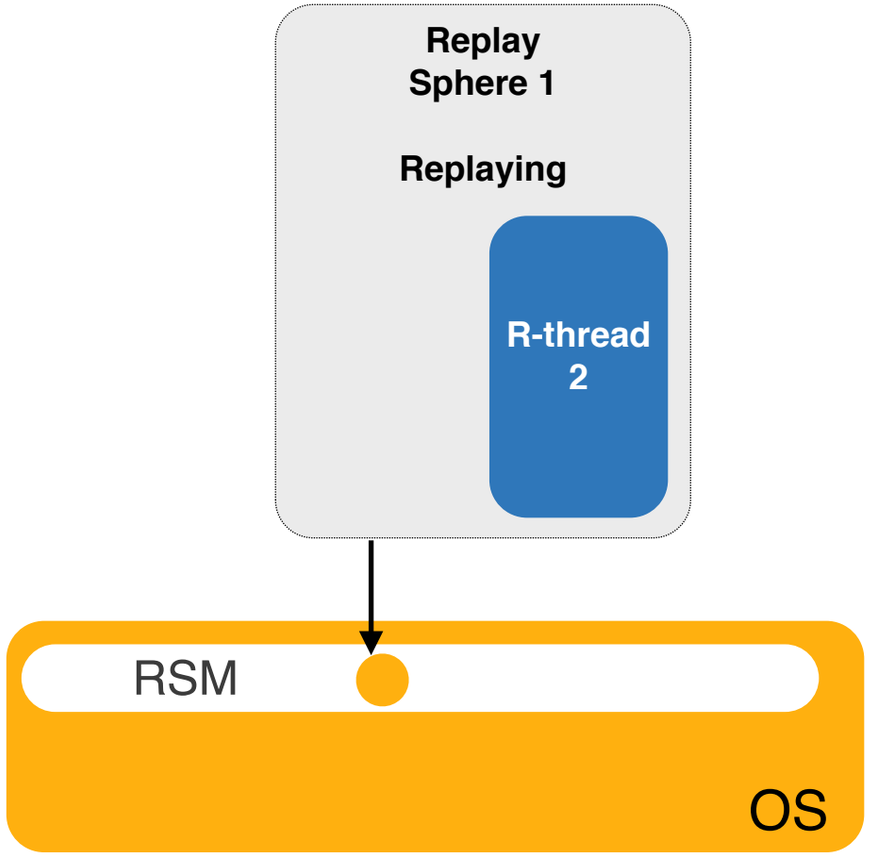
Emulating vs Re-Executing System Calls

- During replay, the RSM emulates most system calls:



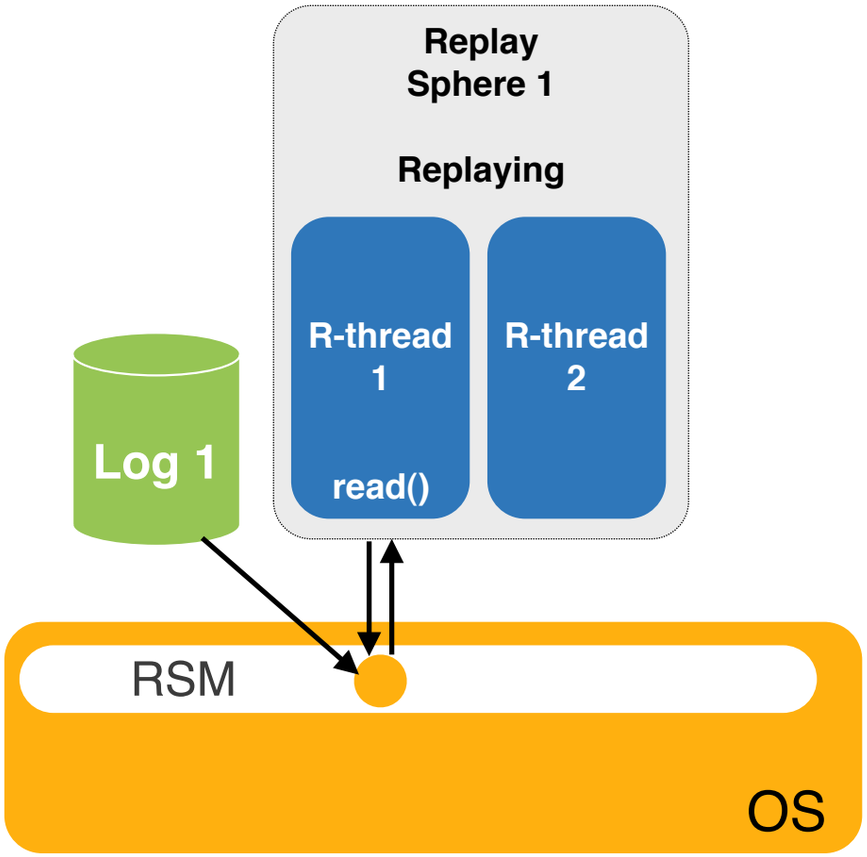
Emulating vs Re-Executing System Calls

- During replay, the RSM emulates most system calls:



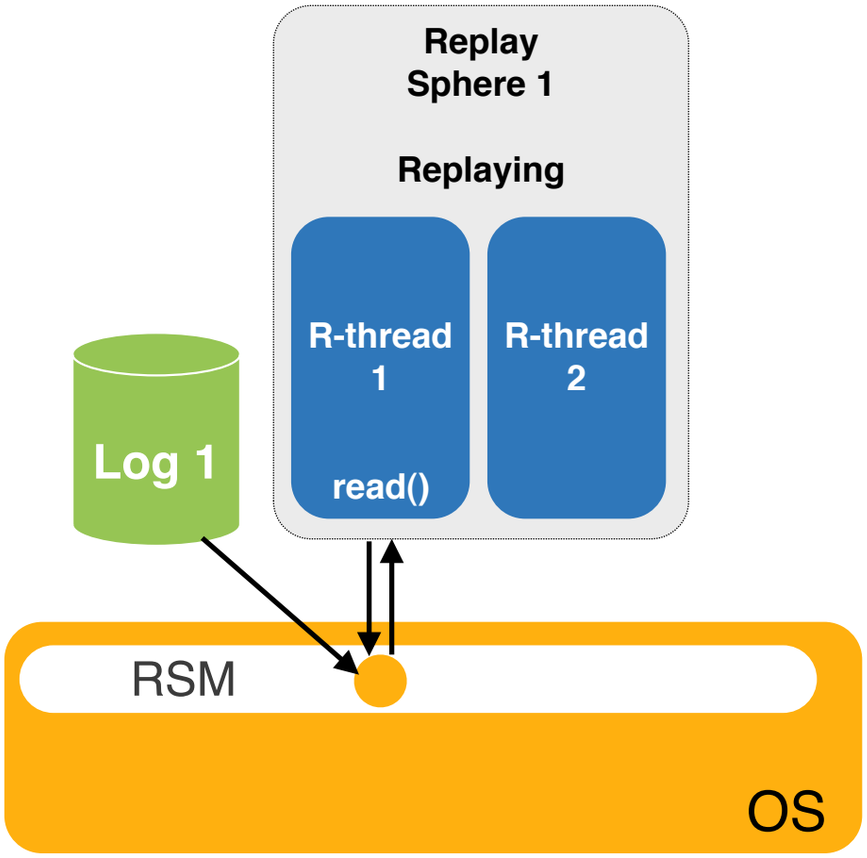
Emulating vs Re-Executing System Calls

- During replay, the RSM emulates most system calls:
  - RSM injects return values from Sphere Input Log, squashes outputs



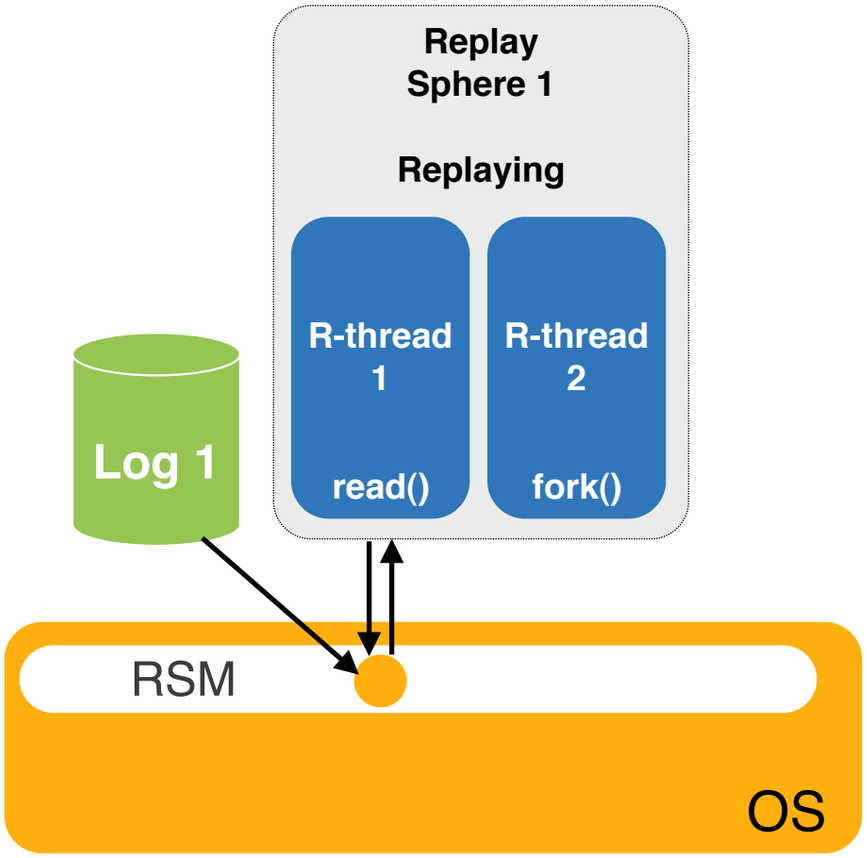
Emulating vs Re-Executing System Calls

- During replay, the RSM emulates most system calls:
  - RSM injects return values from Sphere Input Log, squashes outputs
- Some have to be re-executed



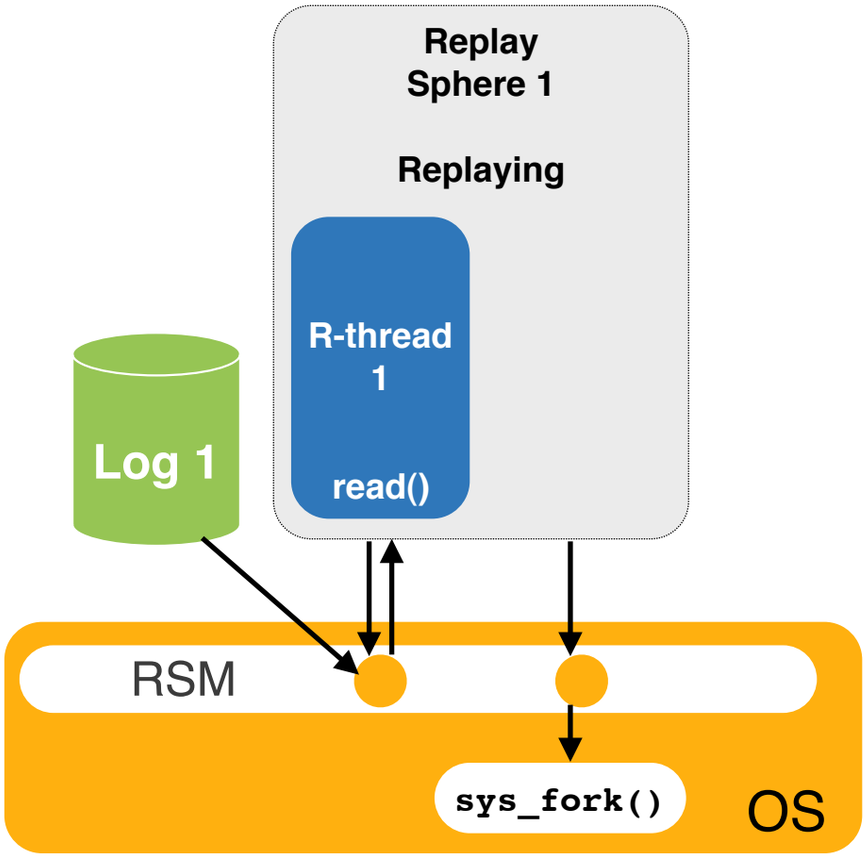
Emulating vs Re-Executing System Calls

- During replay, the RSM emulates most system calls:
  - RSM injects return values from Sphere Input Log, squashes outputs
- Some have to be re-executed



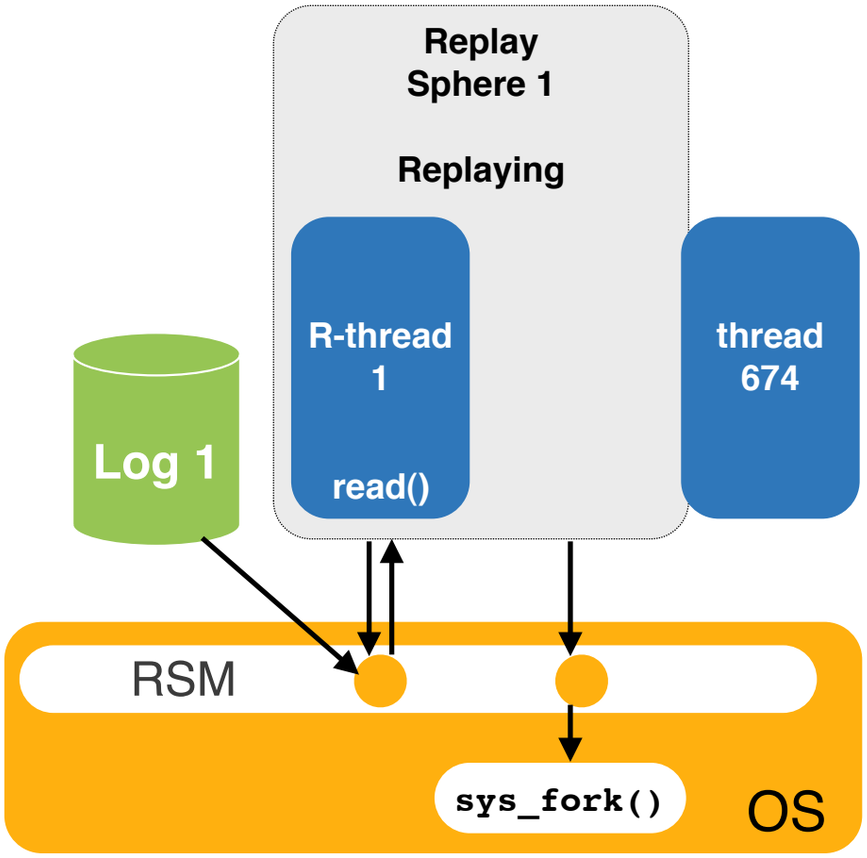
Emulating vs Re-Executing System Calls

- During replay, the RSM emulates most system calls:
  - RSM injects return values from Sphere Input Log, squashes outputs
- Some have to be re-executed



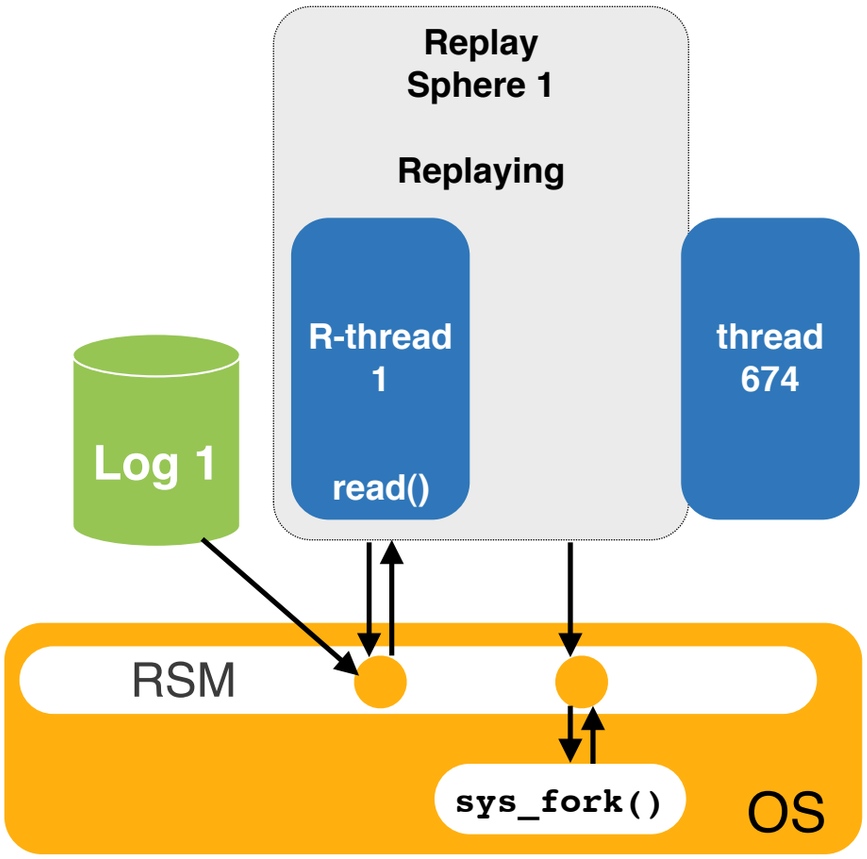
Emulating vs Re-Executing System Calls

- During replay, the RSM emulates most system calls:
  - RSM injects return values from Sphere Input Log, squashes outputs
- Some have to be re-executed



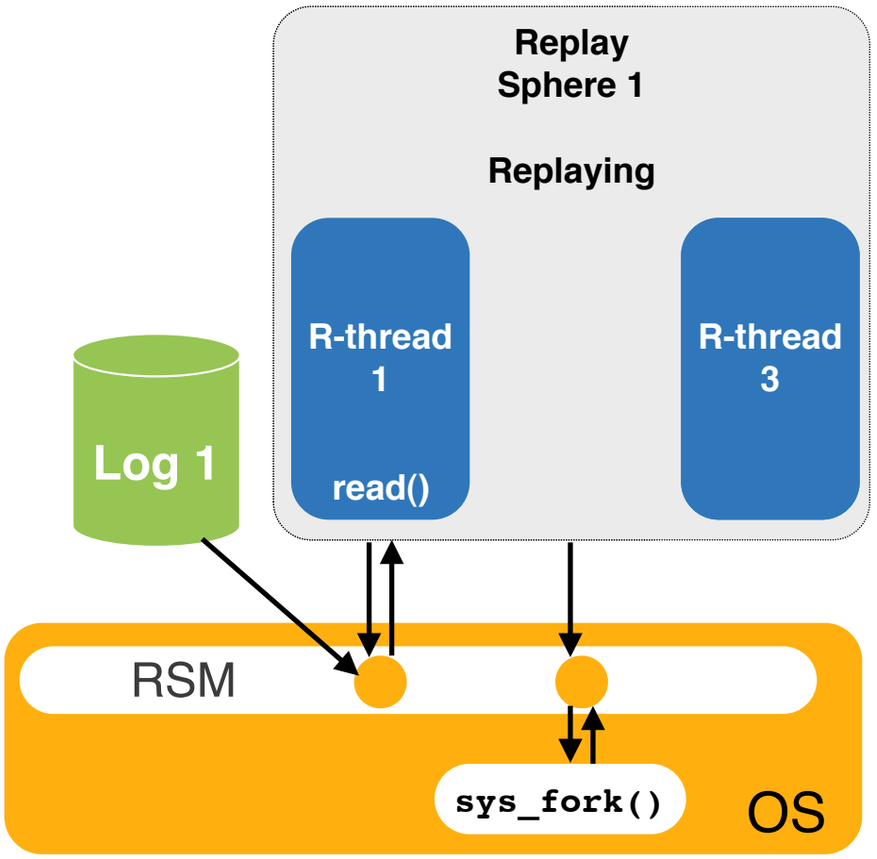
Emulating vs Re-Executing System Calls

- During replay, the RSM emulates most system calls:
  - RSM injects return values from Sphere Input Log, squashes outputs
- Some have to be re-executed



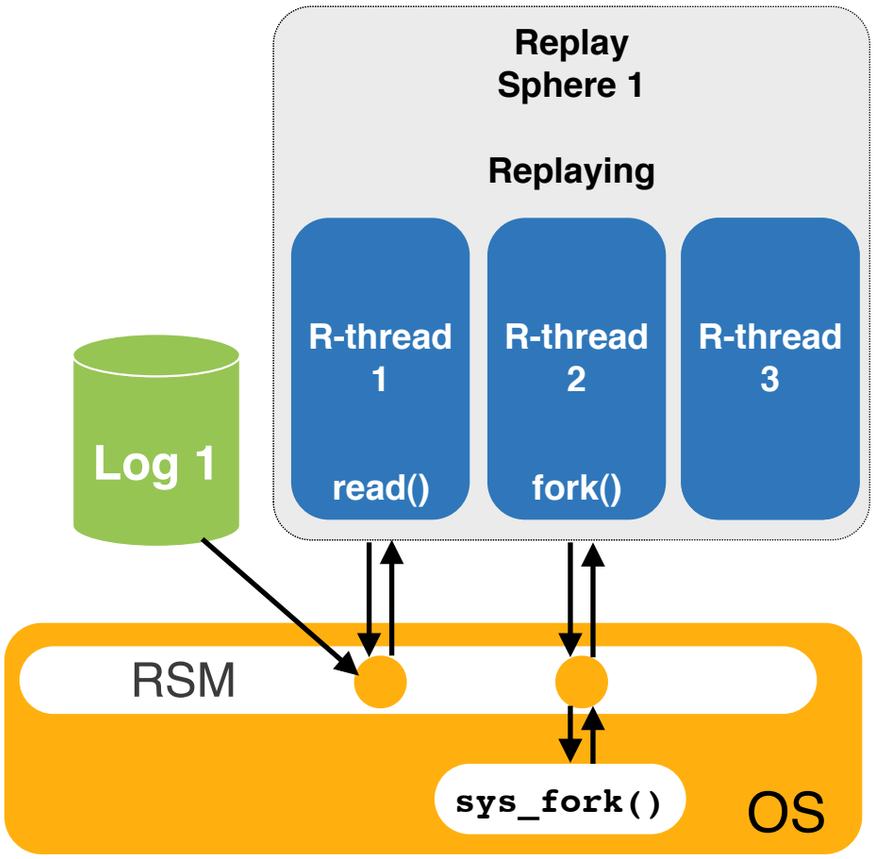
Emulating vs Re-Executing System Calls

- During replay, the RSM emulates most system calls:
  - RSM injects return values from Sphere Input Log, squashes outputs
- Some have to be re-executed



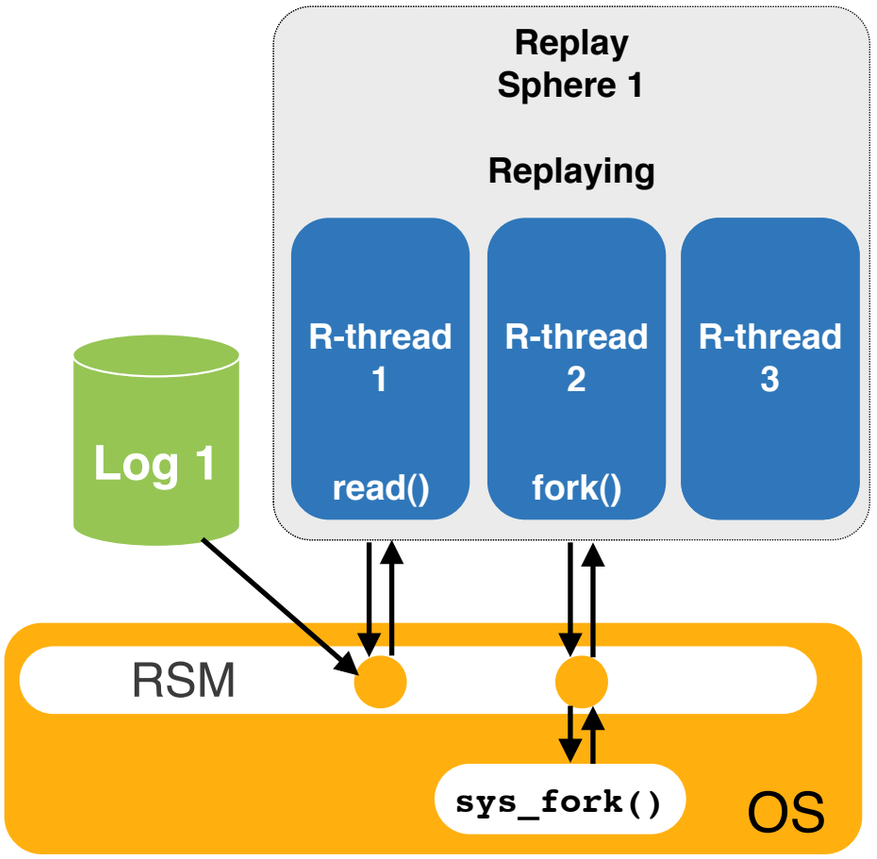
Emulating vs Re-Executing System Calls

- During replay, the RSM emulates most system calls:
  - RSM injects return values from Sphere Input Log, squashes outputs
- Some have to be re-executed



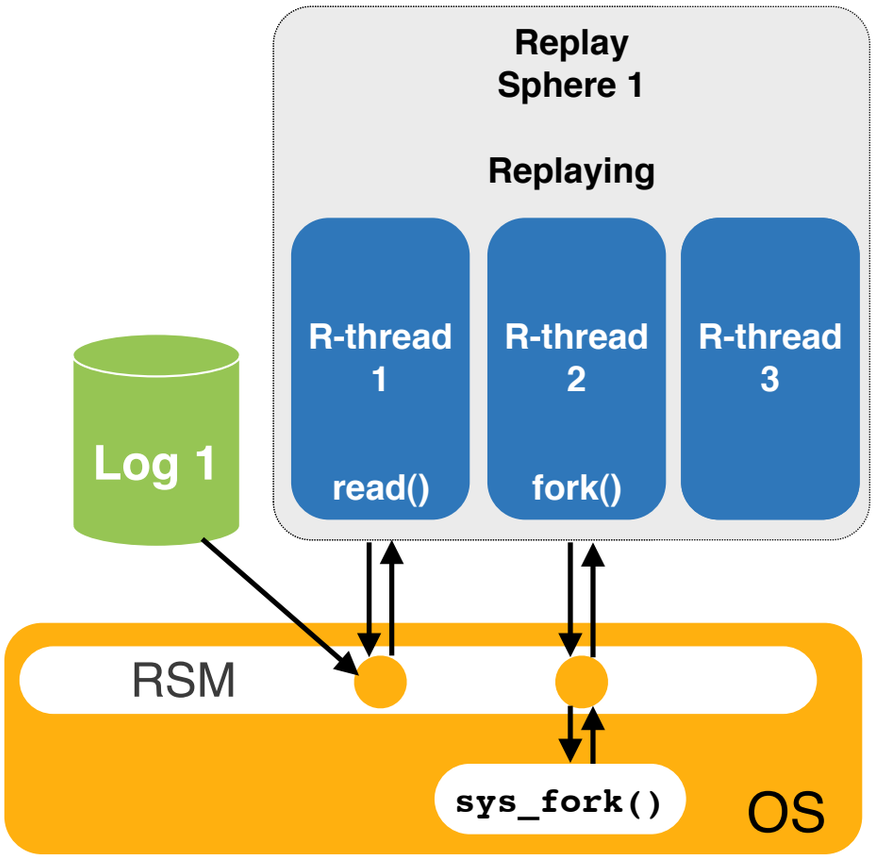
Emulating vs Re-Executing System Calls

- During replay, the RSM emulates most system calls:
  - RSM injects return values from Sphere Input Log, squashes outputs
- Some have to be re-executed
  - Thread management (`clone`)



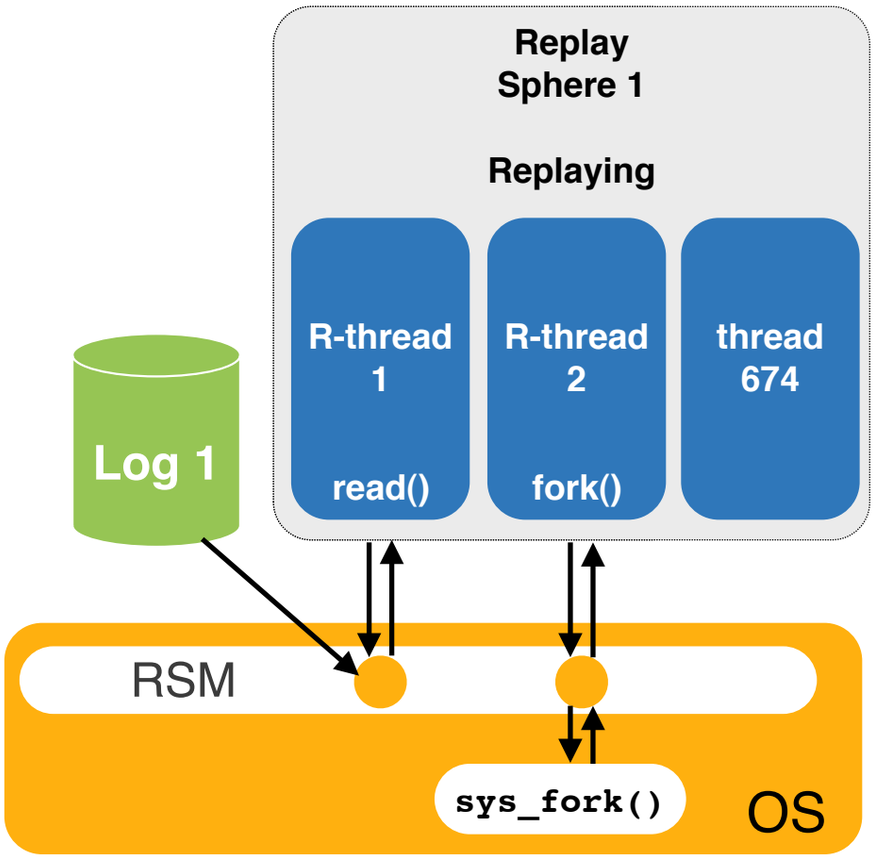
Emulating vs Re-Executing System Calls

- During replay, the RSM emulates most system calls:
  - RSM injects return values from Sphere Input Log, squashes outputs
- Some have to be re-executed
  - Thread management (`clone`)
  - Address space modification (`mprotect`)



Emulating vs Re-Executing System Calls

- During replay, the RSM emulates most system calls:
  - RSM injects return values from Sphere Input Log, squashes outputs
- Some have to be re-executed
  - Thread management (`clone`)
  - Address space modification (`mprotect`)

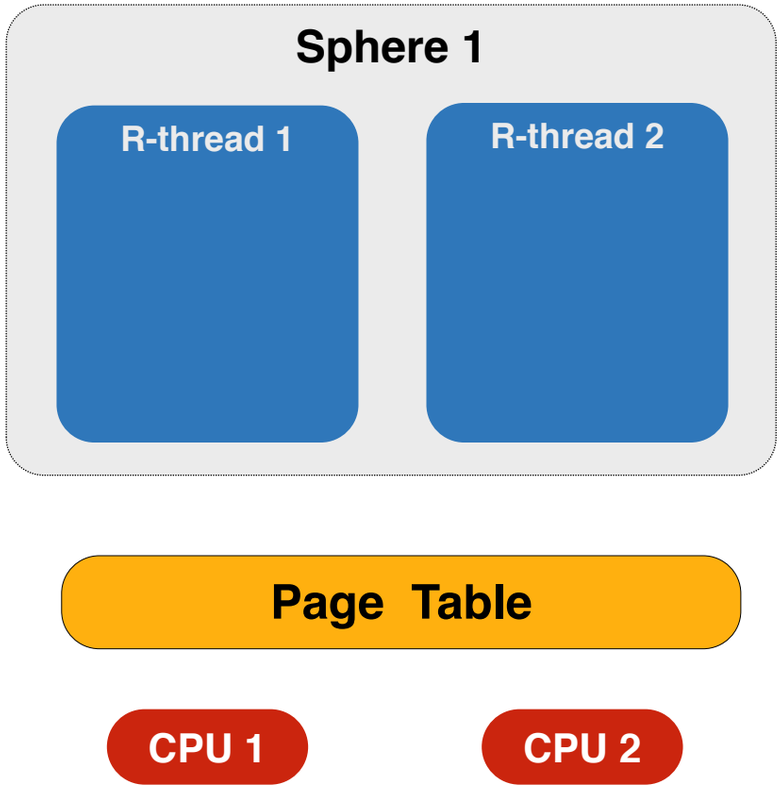


Implicit Dependencies

Implicit Dependencies

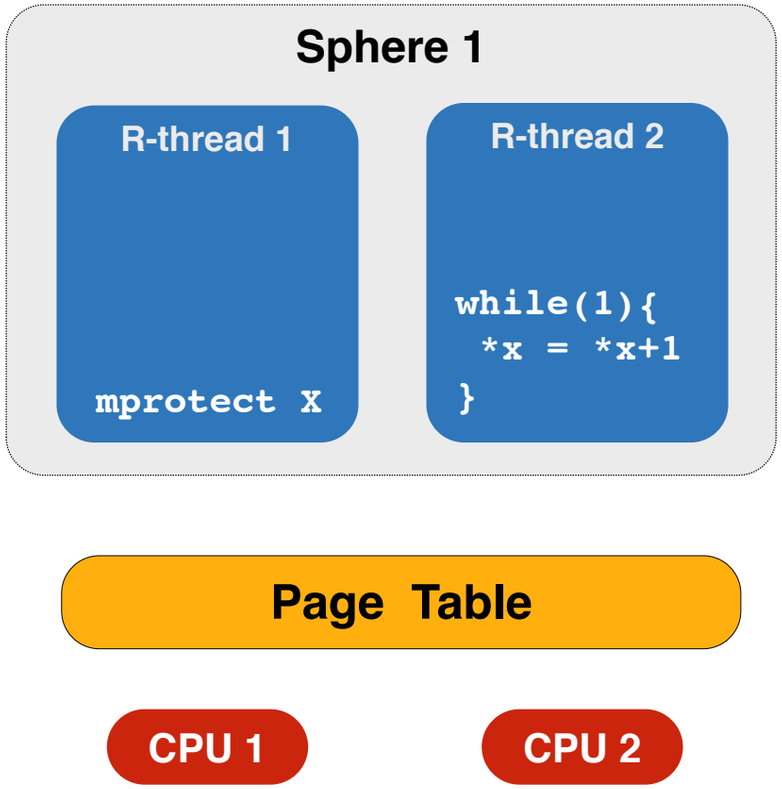
- R-thread changes mapping or protection of address space, and another R-thread uses this changed address space

Implicit Dependencies



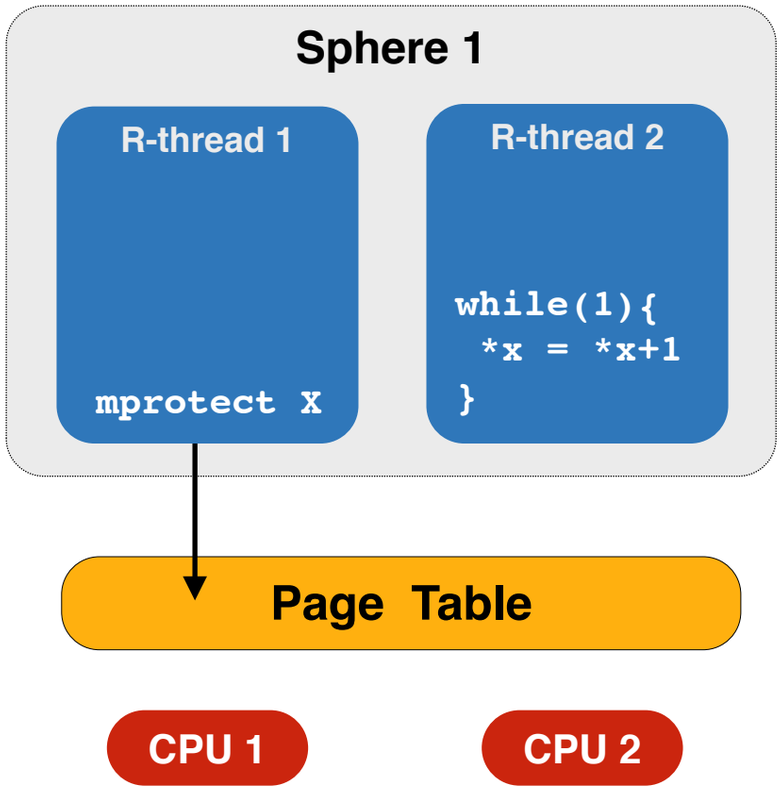
- R-thread changes mapping or protection of address space, and another R-thread uses this changed address space

Implicit Dependencies



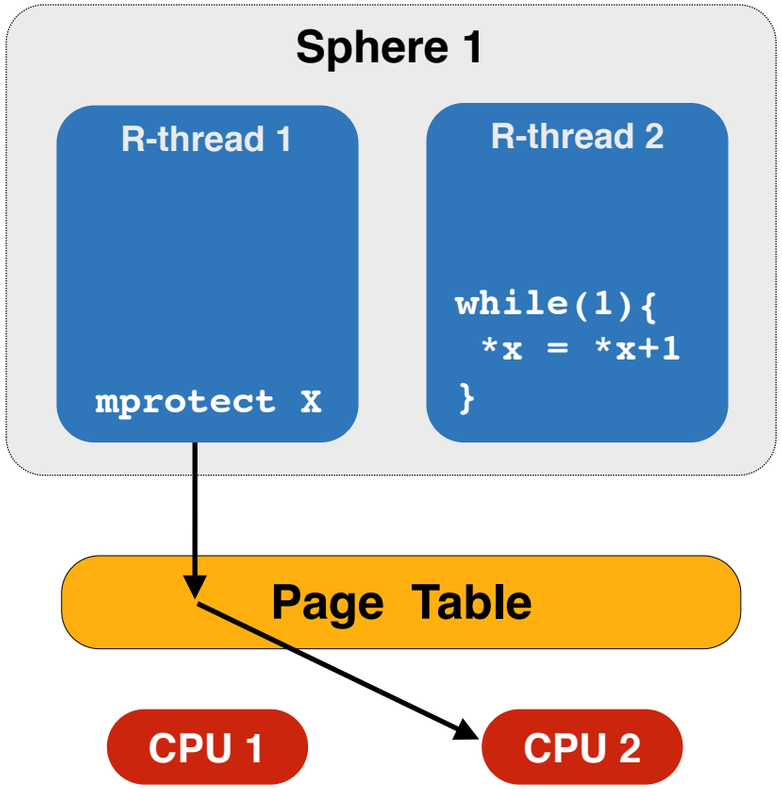
- R-thread changes mapping or protection of address space, and another R-thread uses this changed address space

Implicit Dependencies



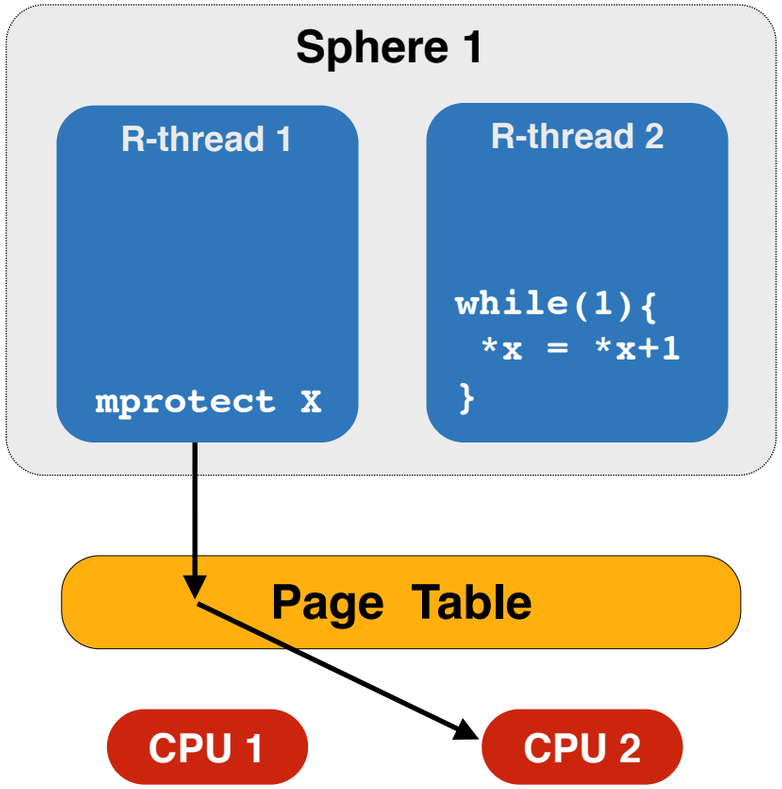
- R-thread changes mapping or protection of address space, and another R-thread uses this changed address space

Implicit Dependencies



- R-thread changes mapping or protection of address space, and another R-thread uses this changed address space

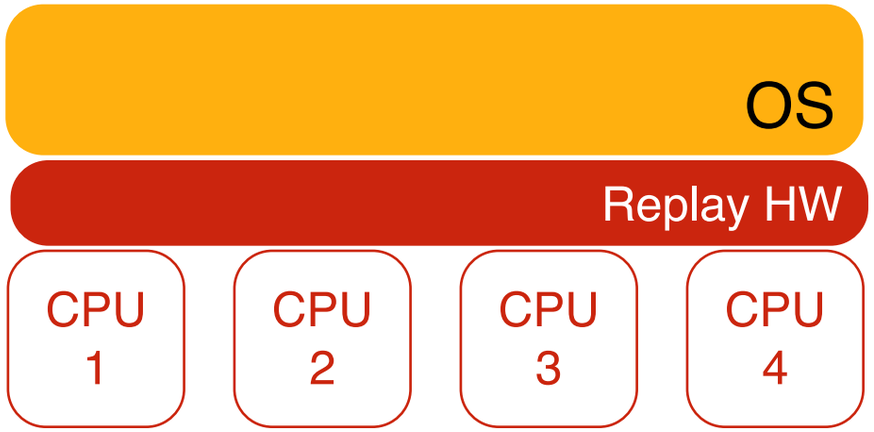
Implicit Dependencies



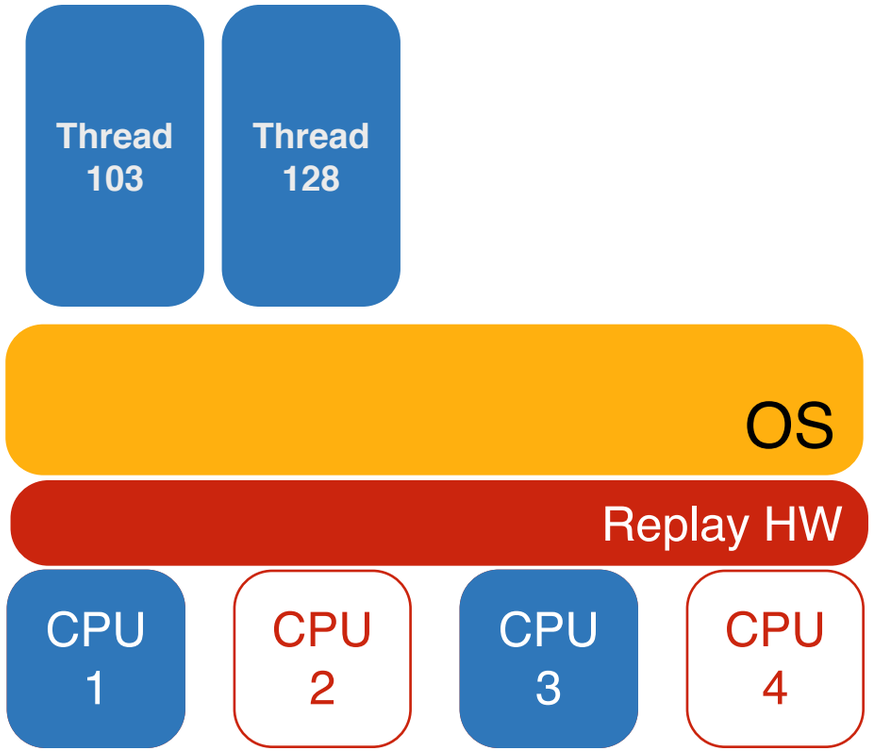
- R-thread changes mapping or protection of address space, and another R-thread uses this changed address space
- RSM can express these dependencies to hardware so these interactions can be recorded

Replay Sphere

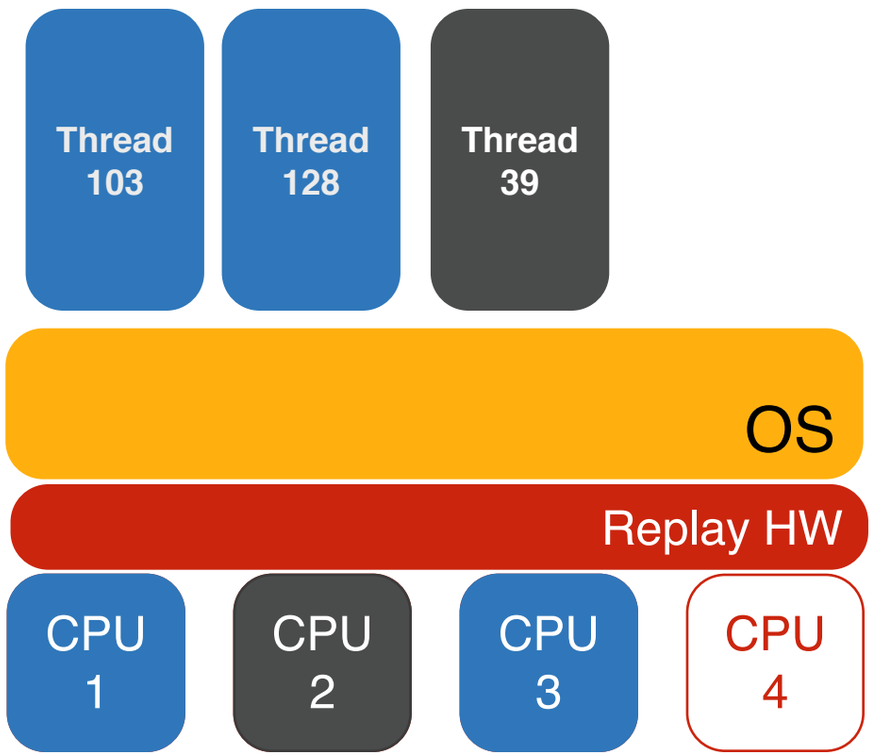
Replay Sphere



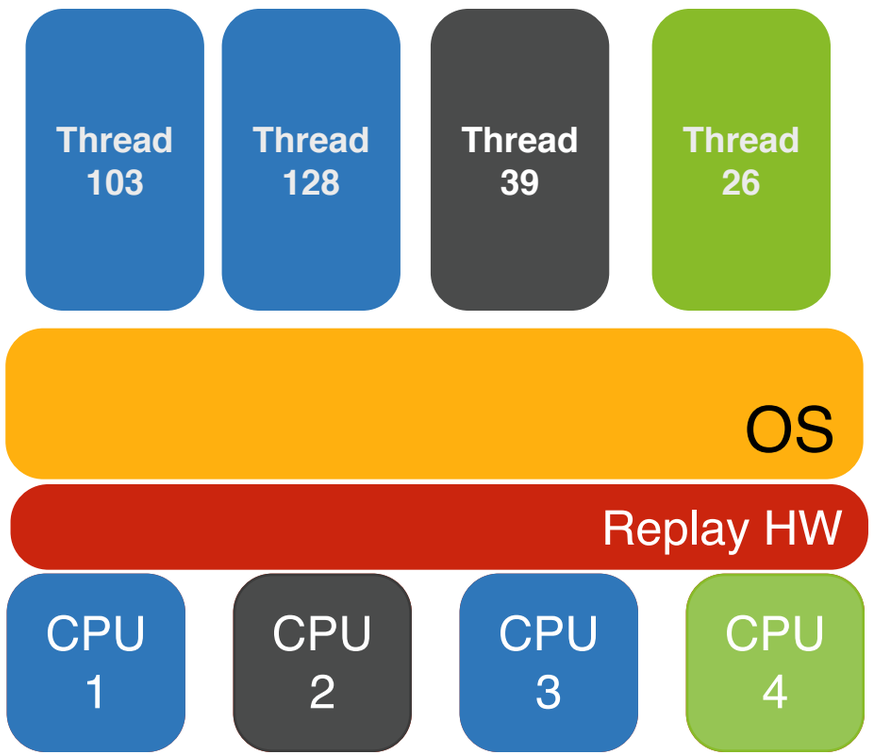
Replay Sphere



Replay Sphere

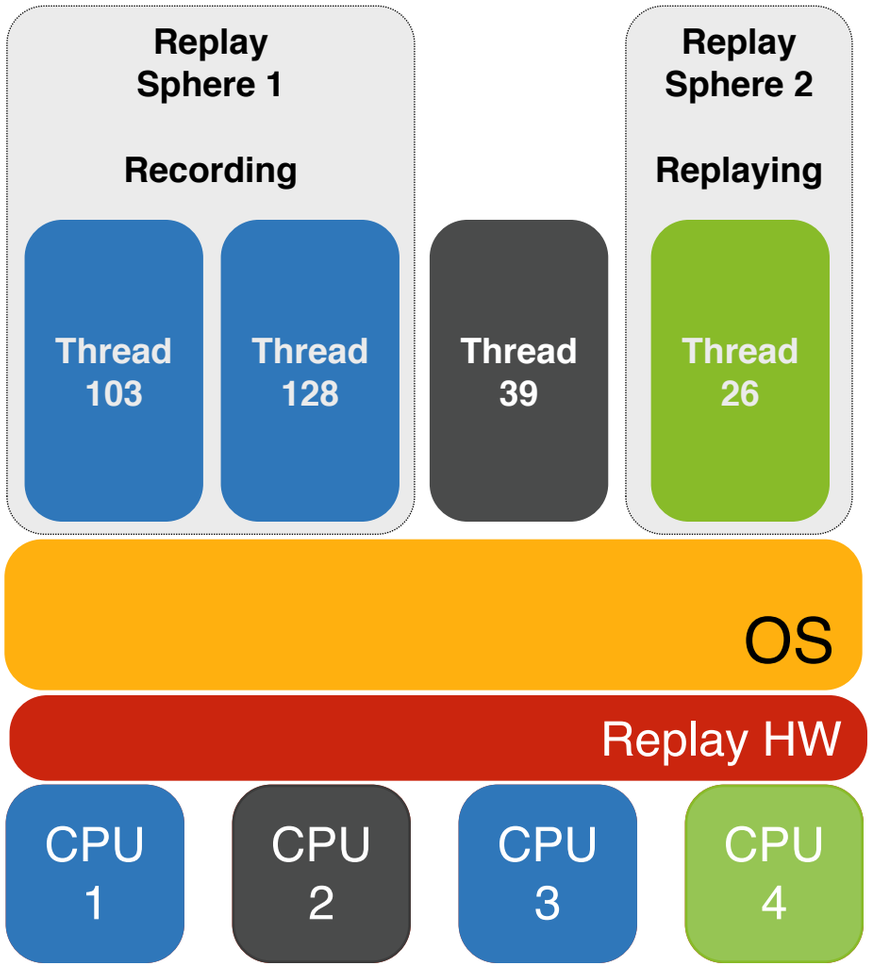


Replay Sphere



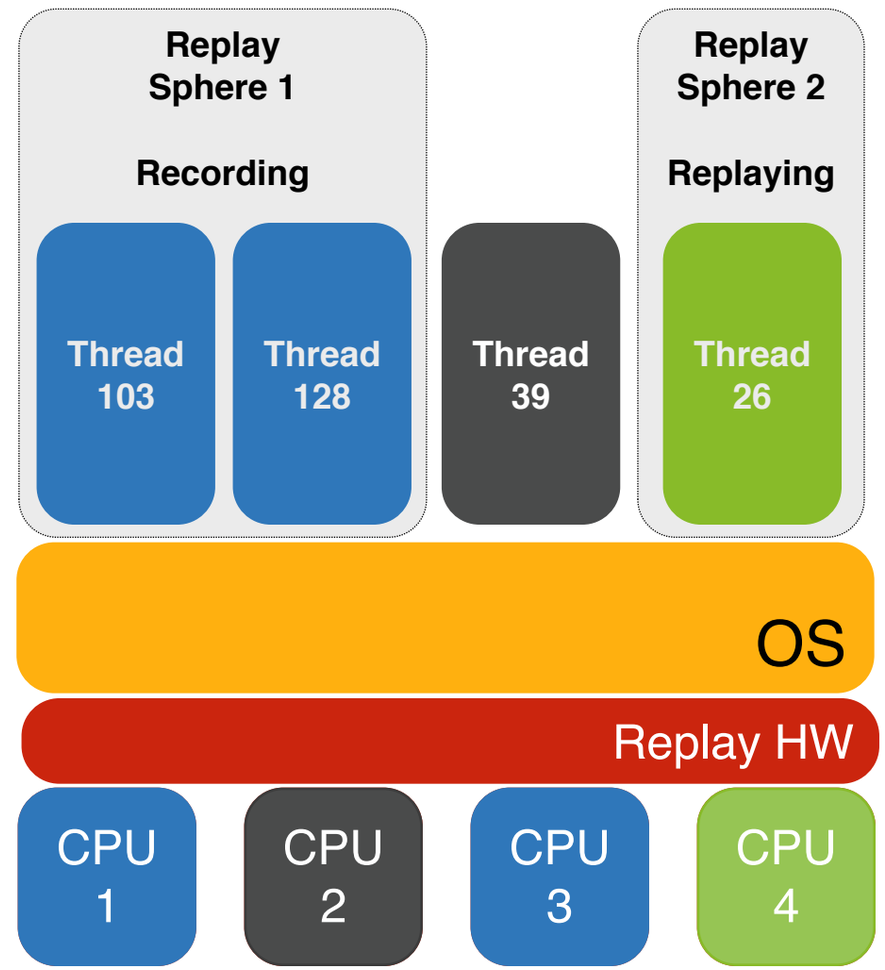
Replay Sphere

- Set of threads recorded and replayed as a unit and their address space



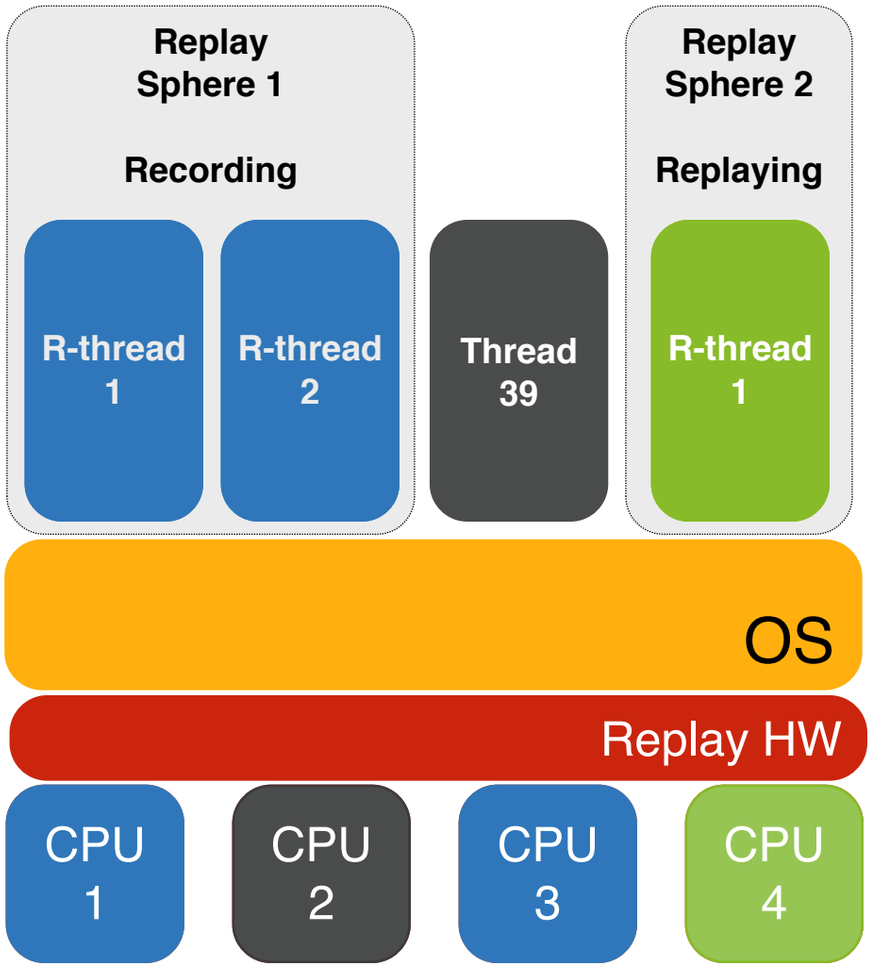
Replay Sphere

- Set of threads recorded and replayed as a unit and their address space
  - Only user-mode threads run inside spheres



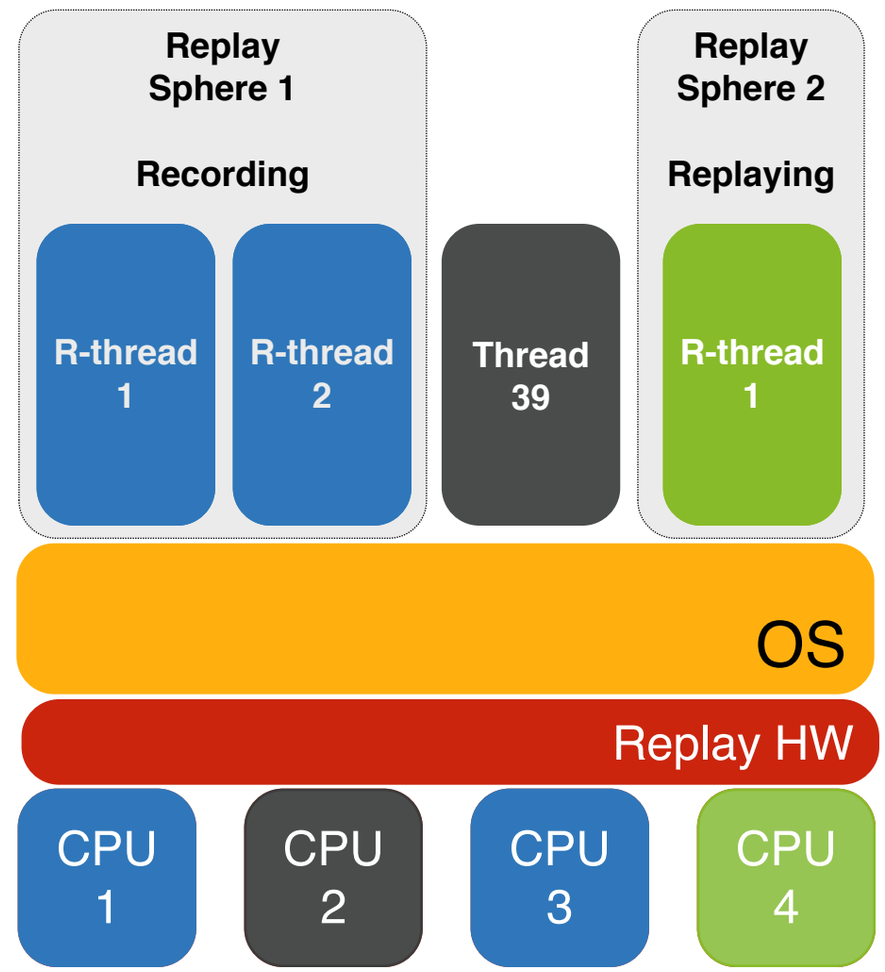
Replay Sphere

- Set of threads recorded and replayed as a unit and their address space
  - Only user-mode threads run inside spheres
  - Threads inside a sphere: R-threads



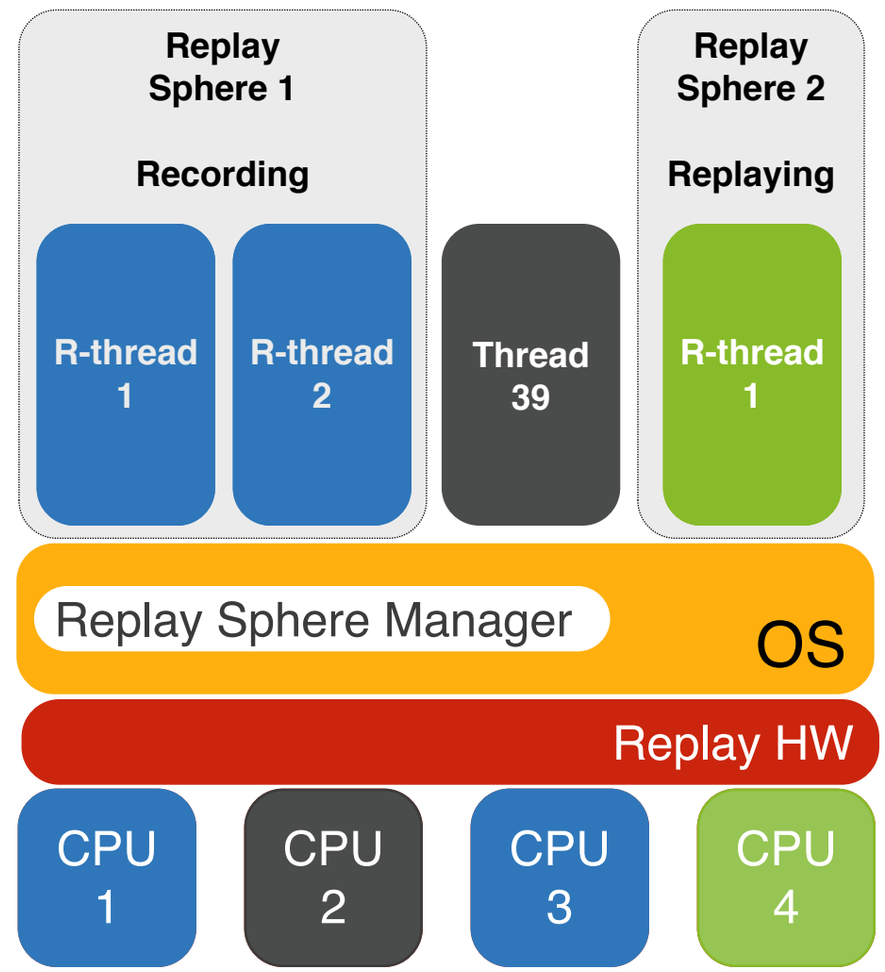
Replay Sphere

- Set of threads recorded and replayed as a unit and their address space
  - Only user-mode threads run inside spheres
  - Threads inside a sphere: R-threads
- R-threads that share memory run within same sphere



Replay Sphere

- Set of threads recorded and replayed as a unit and their address space
  - Only user-mode threads run inside spheres
  - Threads inside a sphere: R-threads
- R-threads that share memory run within same sphere



Self-Modifying Code

- Handling self-modifying code in chunk-based systems is laborious:
  - HW adds instruction fetches to read set
  - Instruction fetches are checked against local write set
- In CapoOne, when a processor detects it has modified code
- However,

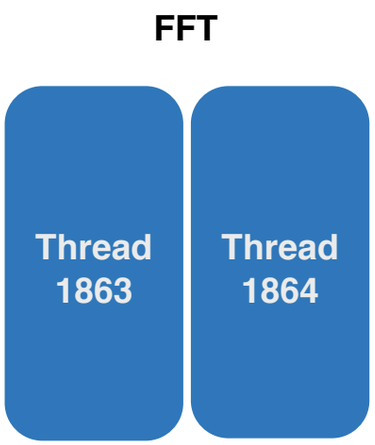
CapoOne: First Capo Implementation

CapoOne: First Capo Implementation

- Records and replays parallel Linux apps

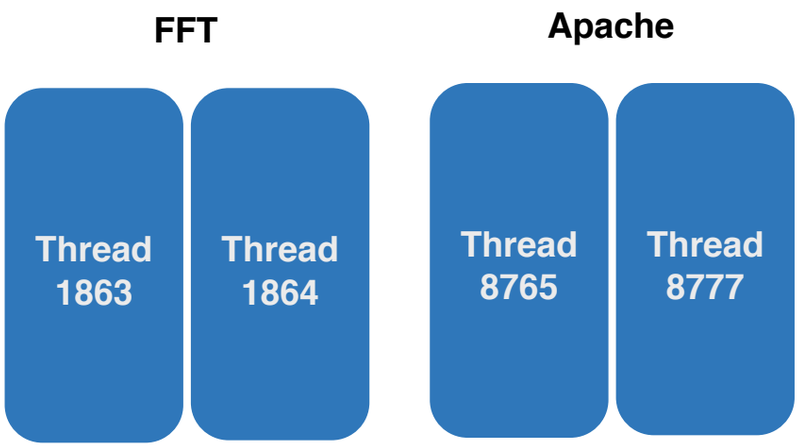
CapoOne: First Capo Implementation

- Records and replays parallel Linux apps



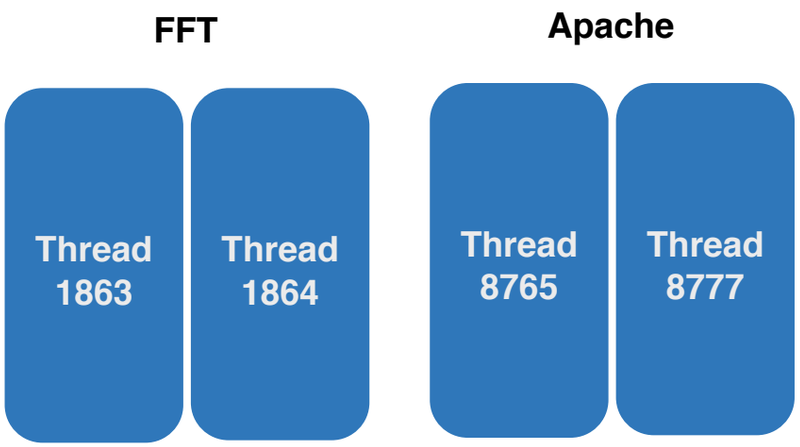
CapoOne: First Capo Implementation

- Records and replays parallel Linux apps



CapoOne: First Capo Implementation

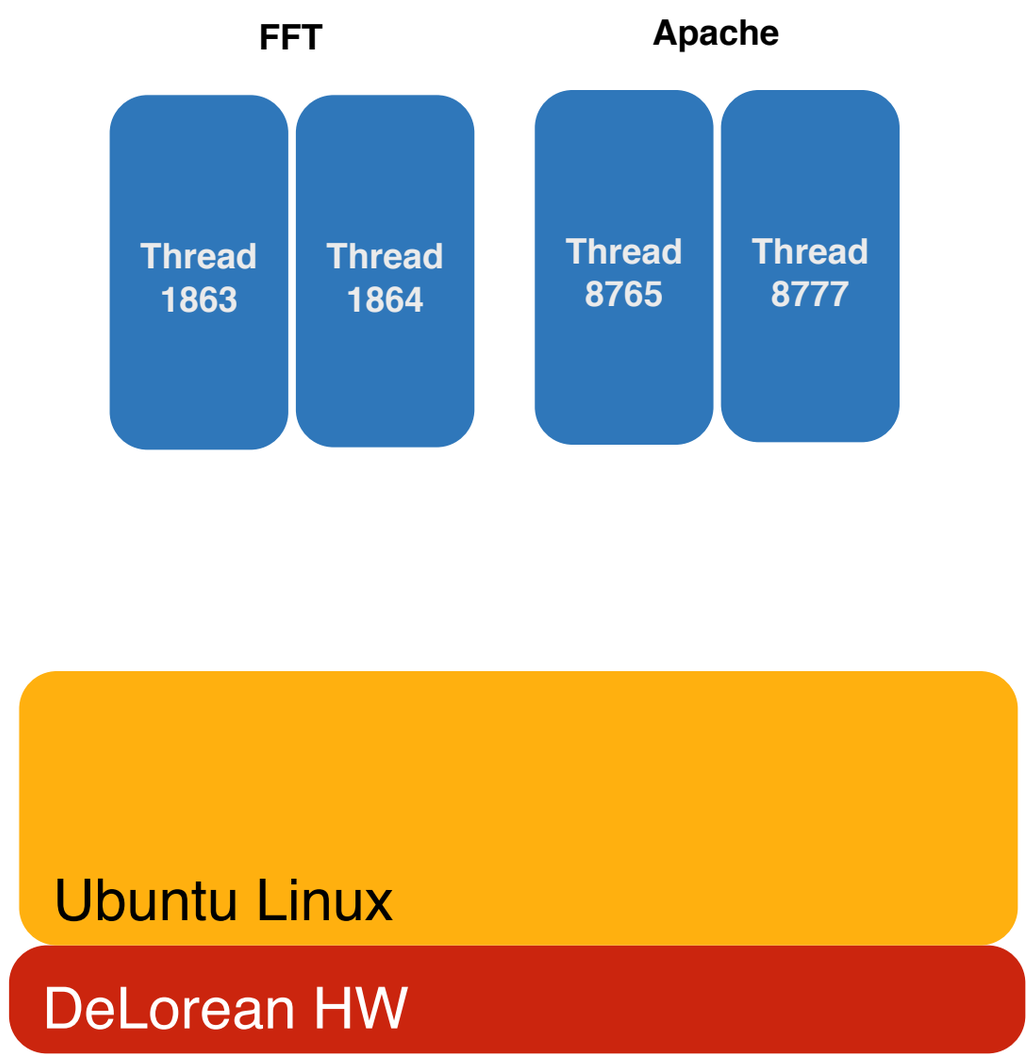
- Records and replays parallel Linux apps
- Simulated HW-Based replay system:
  - DeLorean [Montesinos *et al.* ISCA'08]



DeLorean HW

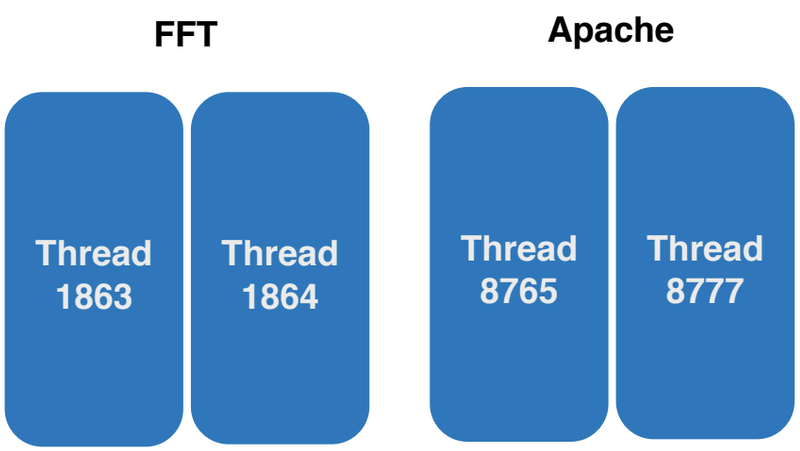
CapoOne: First Capo Implementation

- Records and replays parallel Linux apps
- Simulated HW-Based replay system:
  - DeLorean [Montesinos *et al.* ISCA'08]



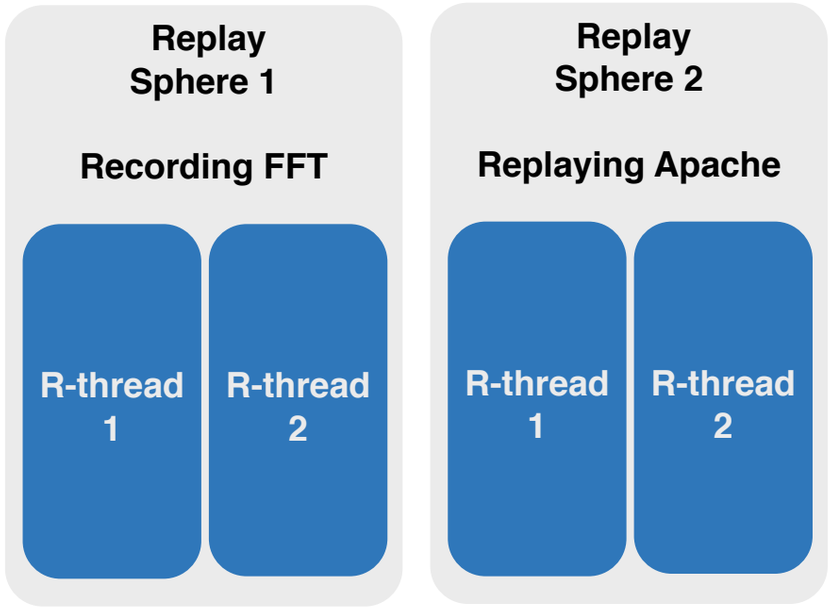
CapoOne: First Capo Implementation

- Records and replays parallel Linux apps
- Simulated HW-Based replay system:
  - DeLorean [Montesinos *et al.* ISCA'08]
- Ubuntu Linux with modified kernel:



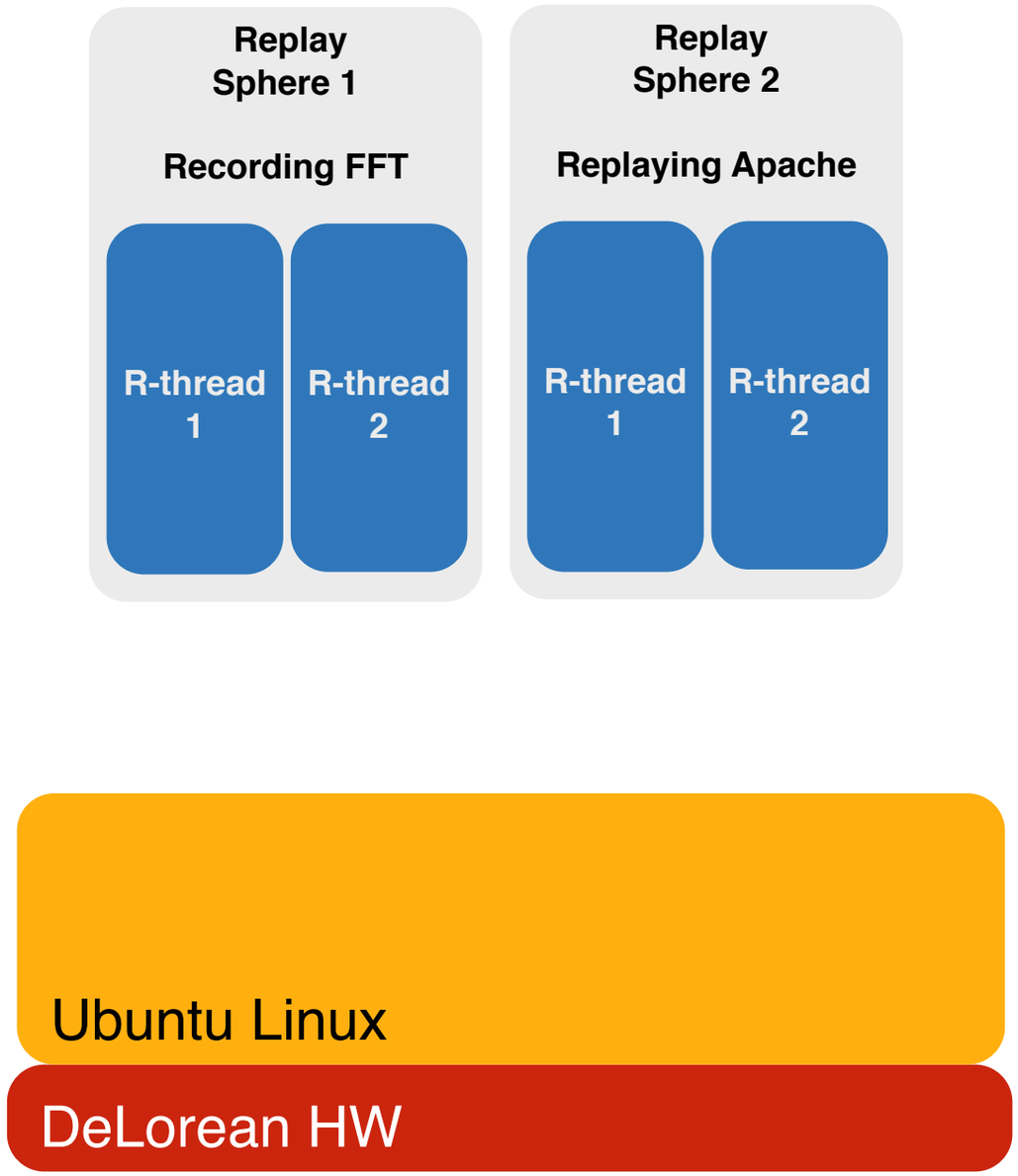
CapoOne: First Capo Implementation

- Records and replays parallel Linux apps
- Simulated HW-Based replay system:
  - DeLorean [Montesinos *et al.* ISCA'08]
- Ubuntu Linux with modified kernel:
  - Added support for spheres, R-threads



CapoOne: First Capo Implementation

- Records and replays parallel Linux apps
- Simulated HW-Based replay system:
  - DeLorean [Montesinos *et al.* ISCA'08]
- Ubuntu Linux with modified kernel:
  - Added support for spheres, R-threads
  - Made some functions more deterministic



CapoOne: First Capo Implementation

- Records and replays parallel Linux apps
- Simulated HW-Based replay system:
  - DeLorean [Montesinos *et al.* ISCA'08]
- Ubuntu Linux with modified kernel:
  - Added support for spheres, R-threads
  - Made some functions more deterministic
- Split Replay Sphere Manager