Private/Shared Classification in Complexity-Effective Coherence Protocols

Alberto Ros[1] Stefanos Kaxiras[2]
Mahdad Davari[2] Konstantinos Koukos[2] Erik Hagersten[2]

[1]Universidad de Murcia
aros@ditec.um.es

[2]Uppsala University

Oct 22, 2015

Outline

Outline

Motivation

- Cache coherence protocols ease programming
- Coherence overhead is an important issue
- But, coherence is sporadically needed
  - → Why pay always?

Motivation

- Cache coherence protocols ease programming
- Coherence overhead is an important issue
- But, coherence is sporadically needed
  - → Why pay always?
- Our goal → Simplify coherence
  - And enforce it only when needed

Motivation

- Cache coherence protocols ease programming
- Coherence overhead is an important issue
- But, coherence is sporadically needed
  - → Why pay always?
- Our goal → Simplify coherence
  - And enforce it only when needed
- How? VIPS[1] family of cache coherence protocols

[1] *Ros & Kaxiras, "Complexity-Effective Multicore Coherence", PACT'12*

VIPS-M: Motivation

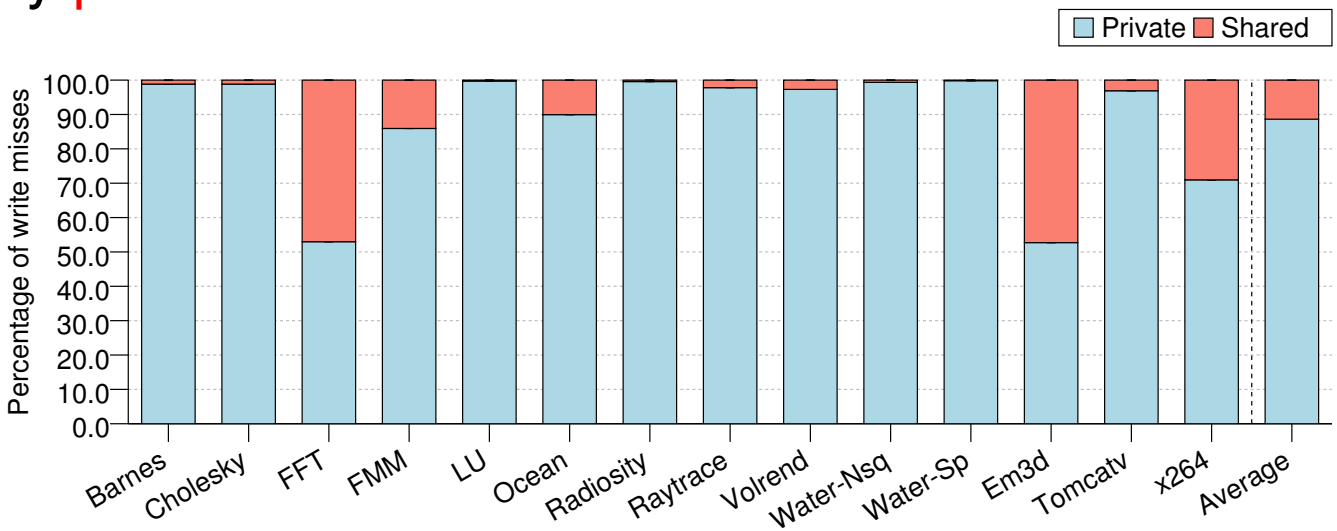
- Write-through protocols are simple
  - Only Valid and Invalid states

VIPS-M: Motivation

- Write-through protocols are simple
  - Only Valid and Invalid states
  - But they are not efficient because of write misses
- Which write misses?

VIPS-M: Motivation

- Write-through protocols are simple
  - Only Valid and Invalid states
  - But they are not efficient because of write misses
- Which write misses?
- Mostly private data misses $\approx$ 90%



VIPS: Write policy

Dynamic write policy in the L1s (private caches, in general)

- Write-back for **P**rivate blocks
  - Simple (no coherence required) as in uniprocessors
  - Efficient $\rightarrow$ no extra misses
- Write-through for **S**hared blocks
  - Simple (only two states, **VI**)
  - Efficient $\rightarrow$ coherence misses

VIPS: **V**alid/**I**nvalid **P**rivate/**S**hared

VIPS-M: Self-invalidation

- We provide sequential consistency for DRF programs
- Self-Invalidation of shared data from L1s
  - Selective Flush (SF) upon synchronization points
  - We eliminate invalidations
  - The directory is gone!
- Multiple writers allowed for shared data
  - No need to request write permission
  - Write-through of *diffs*

VIPS: THE GOOD PART

- Simplifies the protocol to just two states (VI)
- Write-throughs eliminate the need of tracking writers
  - → No indirection for read misses
  - → Directory area reduction
- Self-invalidation eliminates the need to track readers
  - → No need to send invalidations
  - → Indirection completely removed
  - → The directory is gone!

viPS: The problematic part

- Classify data (cache blocks) into private and shared
  - Page-level classification using the OS and the TLBs
  - Both page table and TLB entries have a P/S bit
  - The first TLB miss by a core sets the page to P
  - Subsequent TLB misses from other cores set the page to S
    - Interrupts the single core having the page as P
    - Forces the WT of every dirty block in the page

viPS: The problematic part

- Classify data (cache blocks) into private and shared
  - Page-level classification using the OS and the TLBs
  - Both page table and TLB entries have a P/S bit
  - The first TLB miss by a core sets the page to P
  - Subsequent TLB misses from other cores set the page to S
    - Interrupts the single core having the page as P
    - Forces the WT of every dirty block in the page

This talk focuses on the PS classification!

viPS: Ideal classification

- Characteristics of an ideal PS classification:
  - Fine granularity
  - OS agnostic
  - No remote interruptions
  - No indirection
    - → Unexpected messages from the physical to virtual word
    - → Allows for simple virtual-cache coherence[2]

[2] *Kaxiras & Ros, "A New Perspective for Efficient Virtual-Cache Coherence", ISCA'13*

viPS: Ideal classification

- Characteristics of an ideal PS classification:
  - Fine granularity ☹
  - OS agnostic ☹
  - No remote interruptions ☹
  - No indirection ☺
    - → Unexpected messages from the physical to virtual word
    - → Allows for simple virtual-cache coherence[2]

[2] *Kaxiras & Ros, "A New Perspective for Efficient Virtual-Cache Coherence", ISCA'13*

Outline

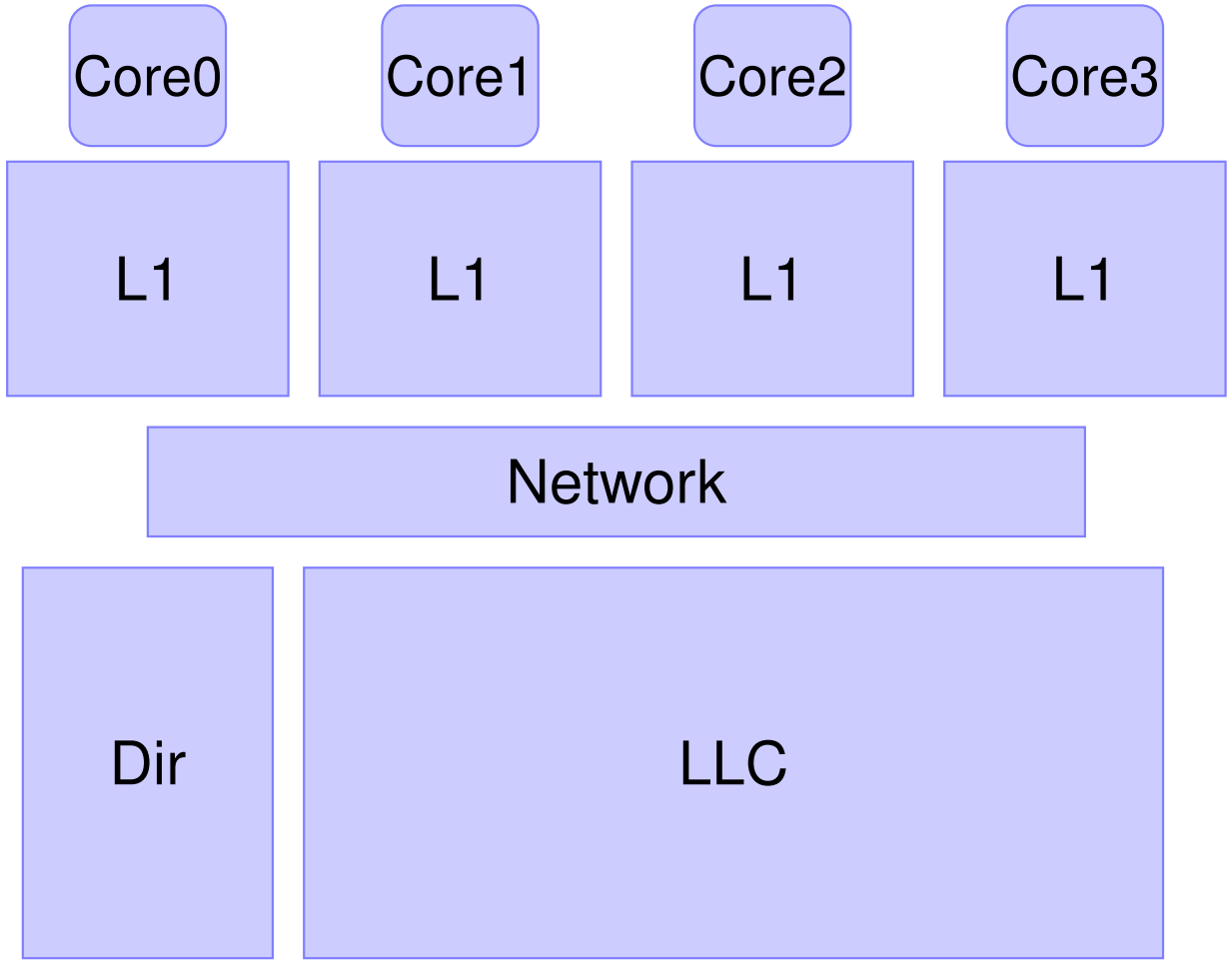
Motivation

- Idea: Fine-grain classification in hardware with a single-pointer directory: DIR_1 directory
- Advantages of DIR_1 directories (e.g. DIR_1-B, DIR_1-NB)
  - Only need to store the owner of the block
  - $O(\log n)$ storage
- Drawbacks of DIR_1 directories
  - Wide sharing
    - $\rightarrow$ Not allowed DIR_1-NB or broadcast DIR_1-NB
  - Lost of information
    - $\rightarrow$ Invalidation or backup storage

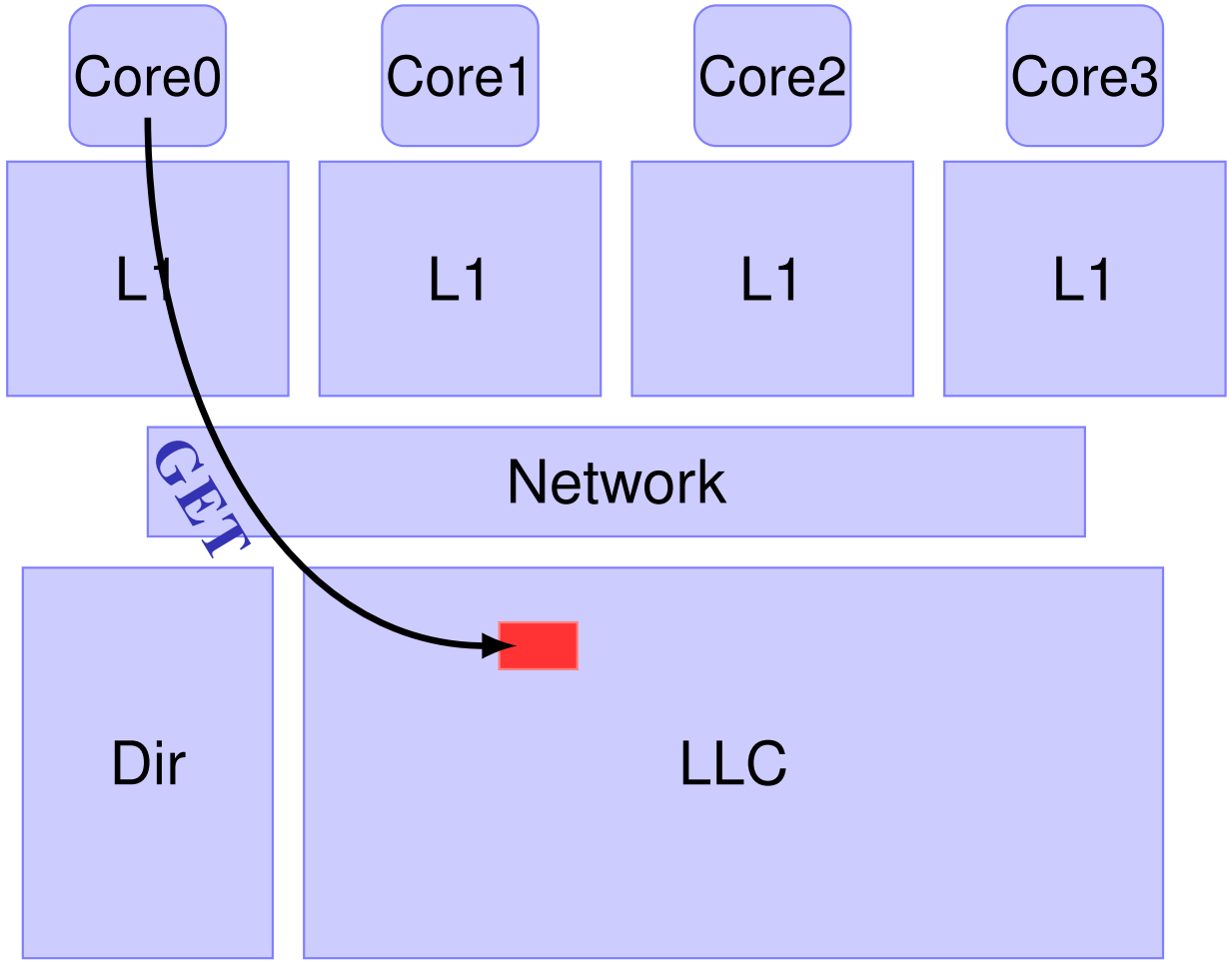
PROPOSAL

- Our solution
  - DIR_1-SISD: Self-contained, self-corrected directory
  - Allows wide sharing
  - Does not employ broadcast/multicast
  - No invalidations required upon lost of information
    - Directory and cache contents are not inclusive
  - No backup required
    - Self-contained
- How?
  - Private blocks: tracked by DIR_1
  - Shared blocks: coherence responsibility delegated to cores
    - Self-invalidation, self-downgrade SISD

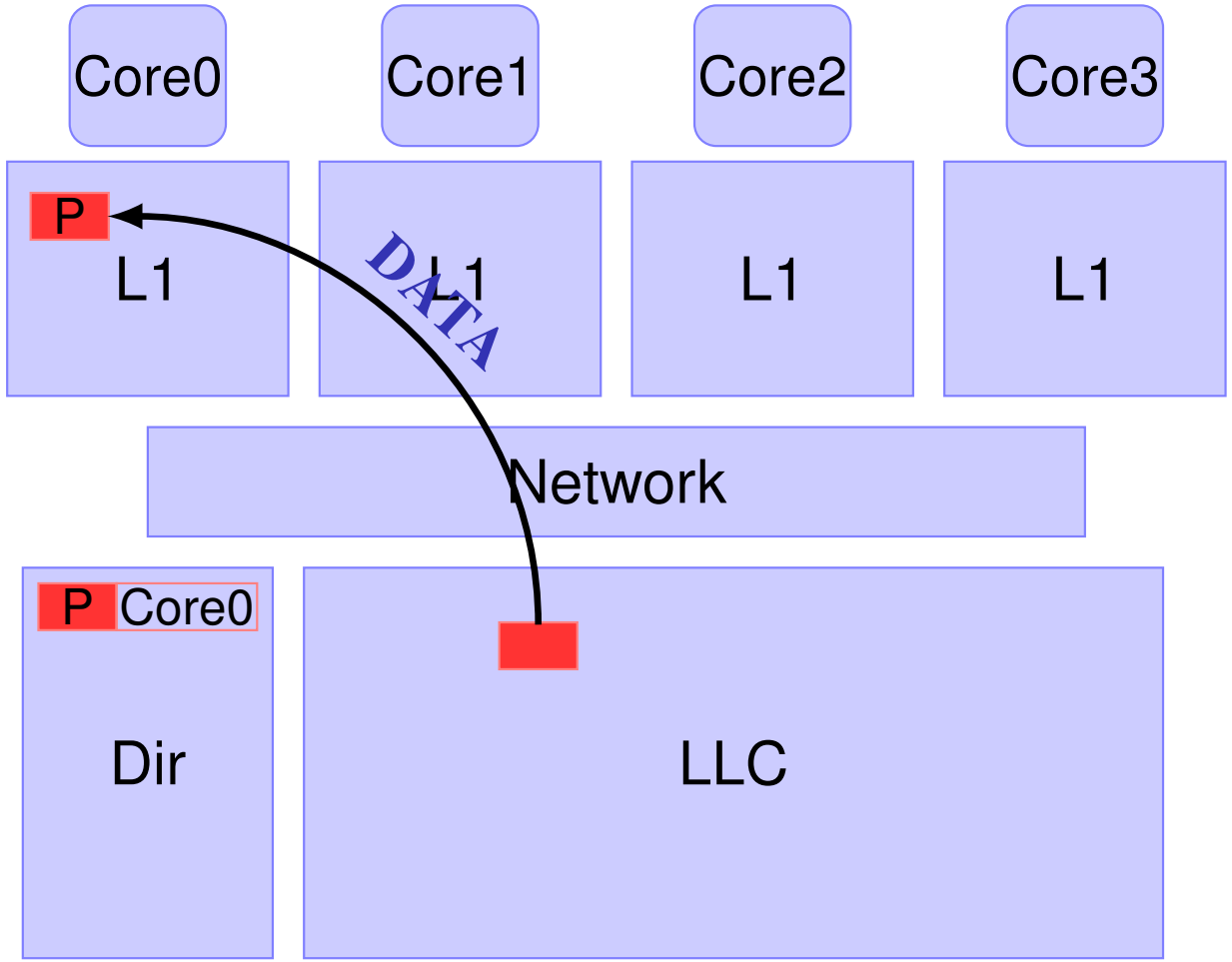
PRIVATE CACHE ENTRIES



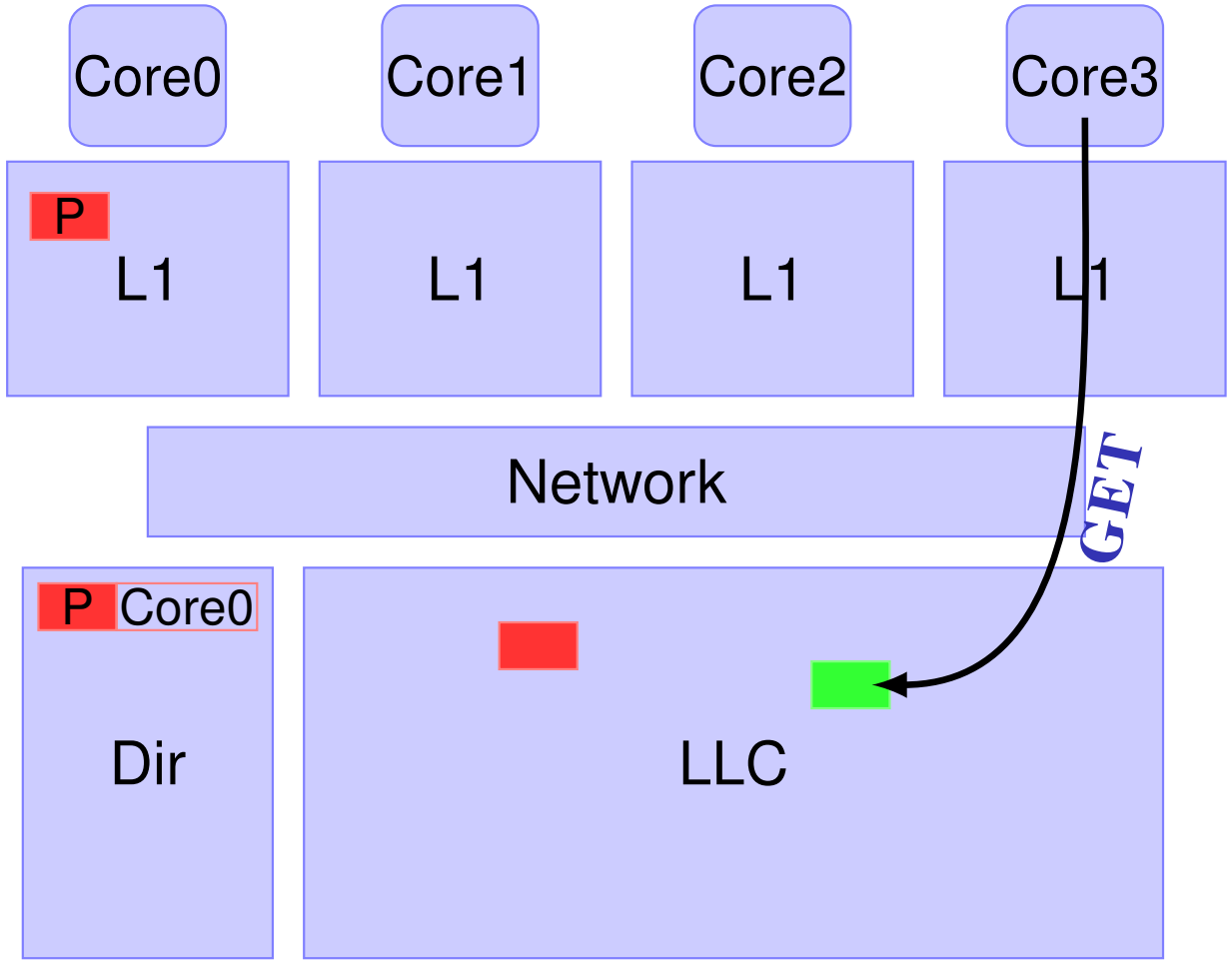
PRIVATE CACHE ENTRIES



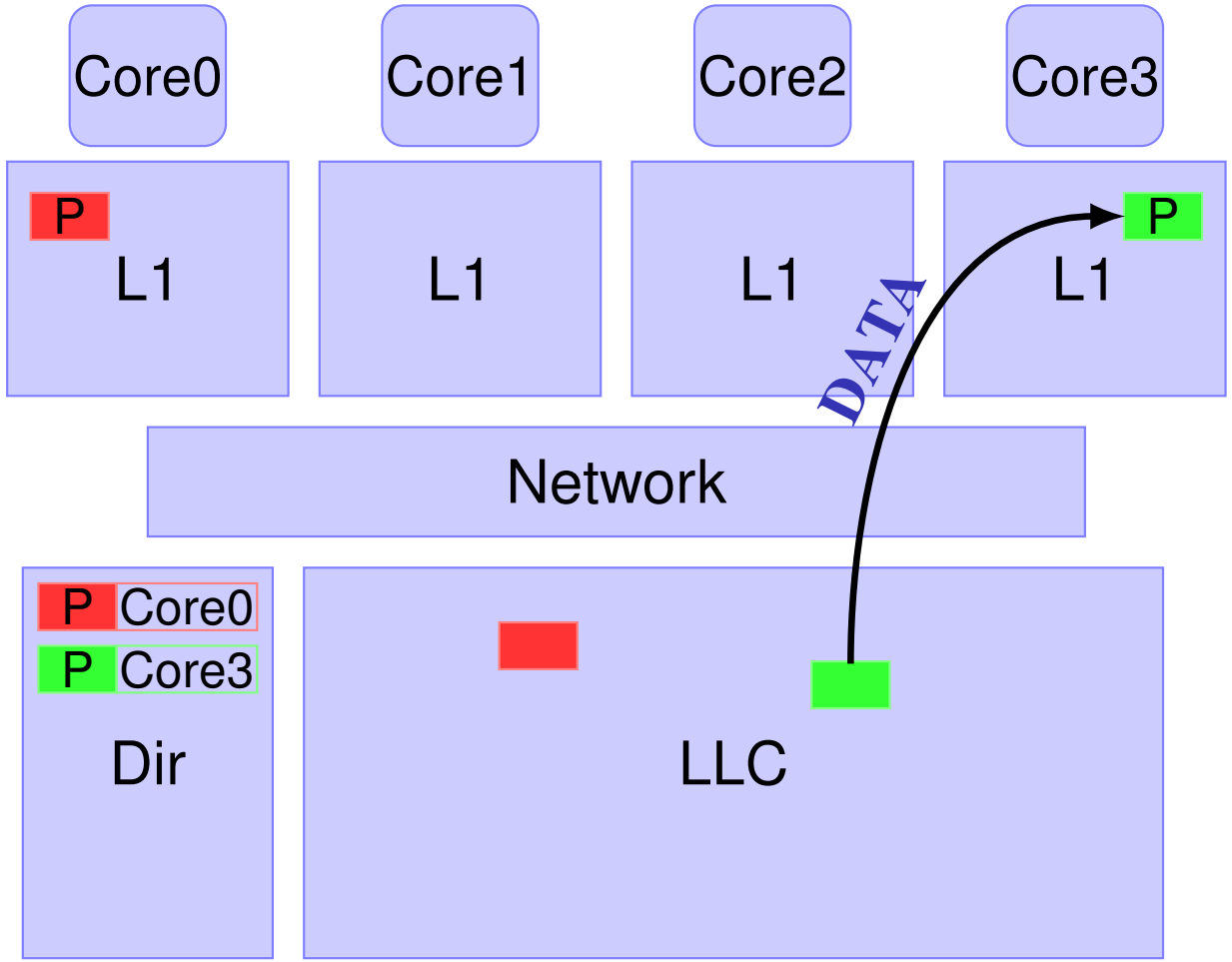
PRIVATE CACHE ENTRIES



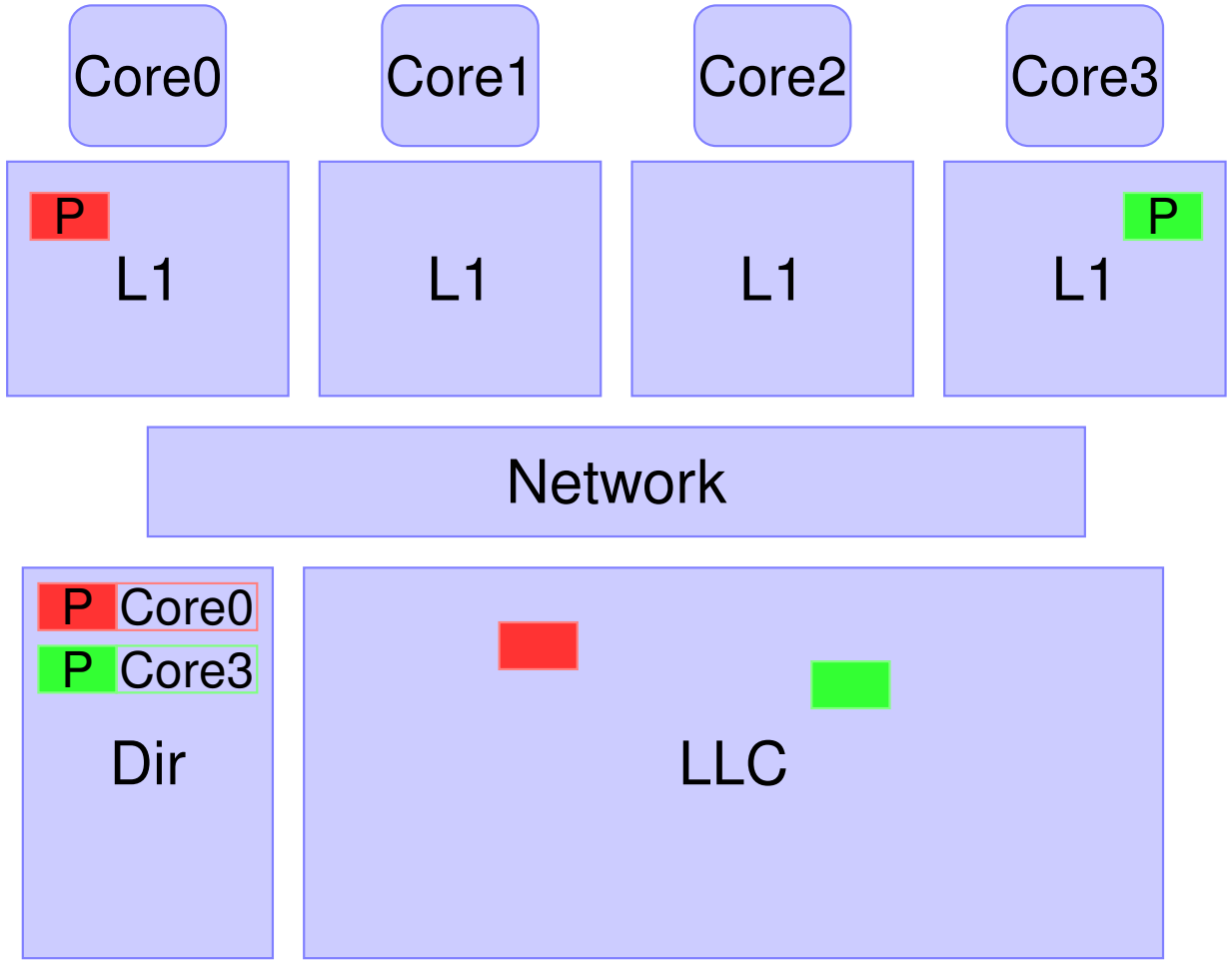
PRIVATE CACHE ENTRIES



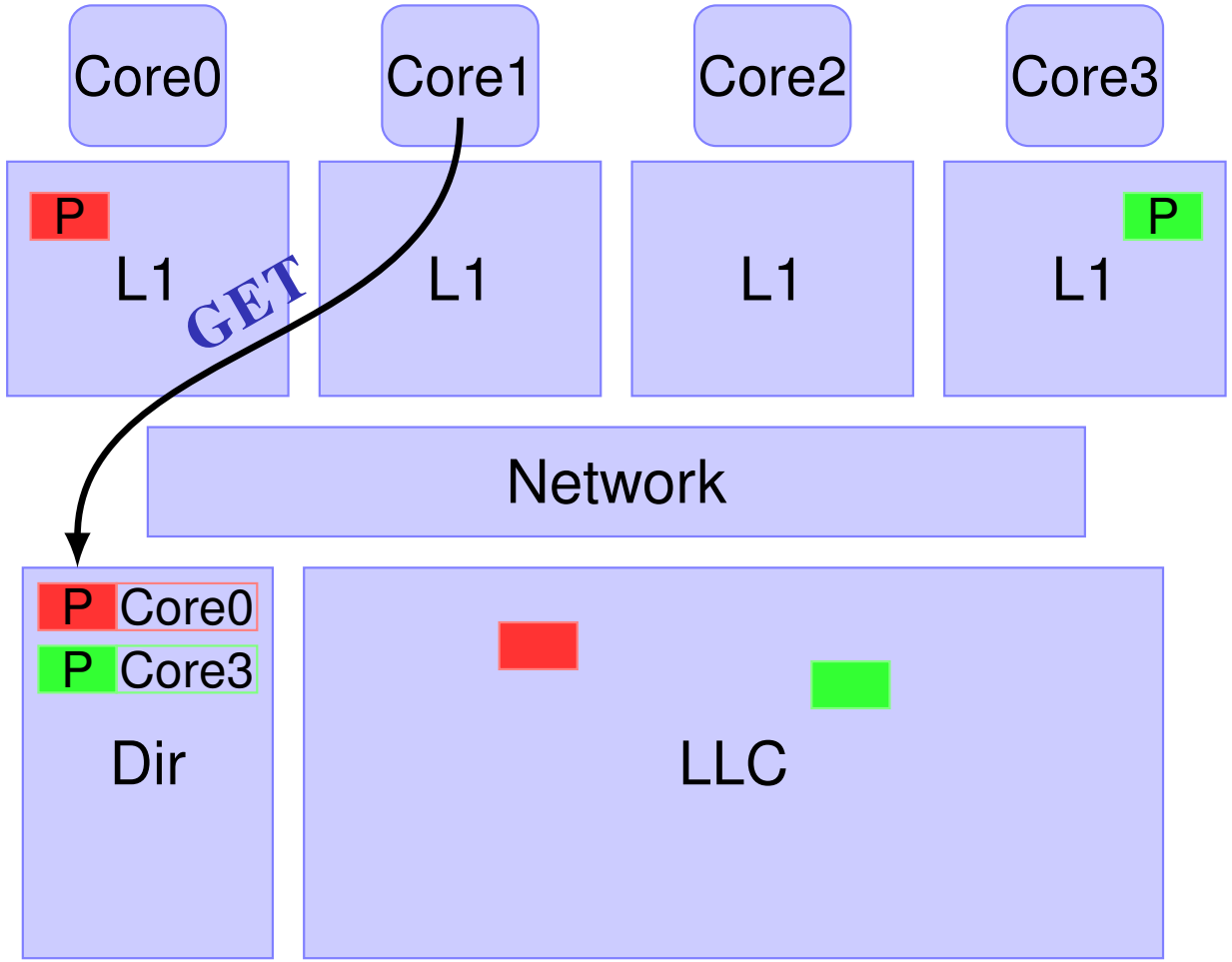
PRIVATE CACHE ENTRIES



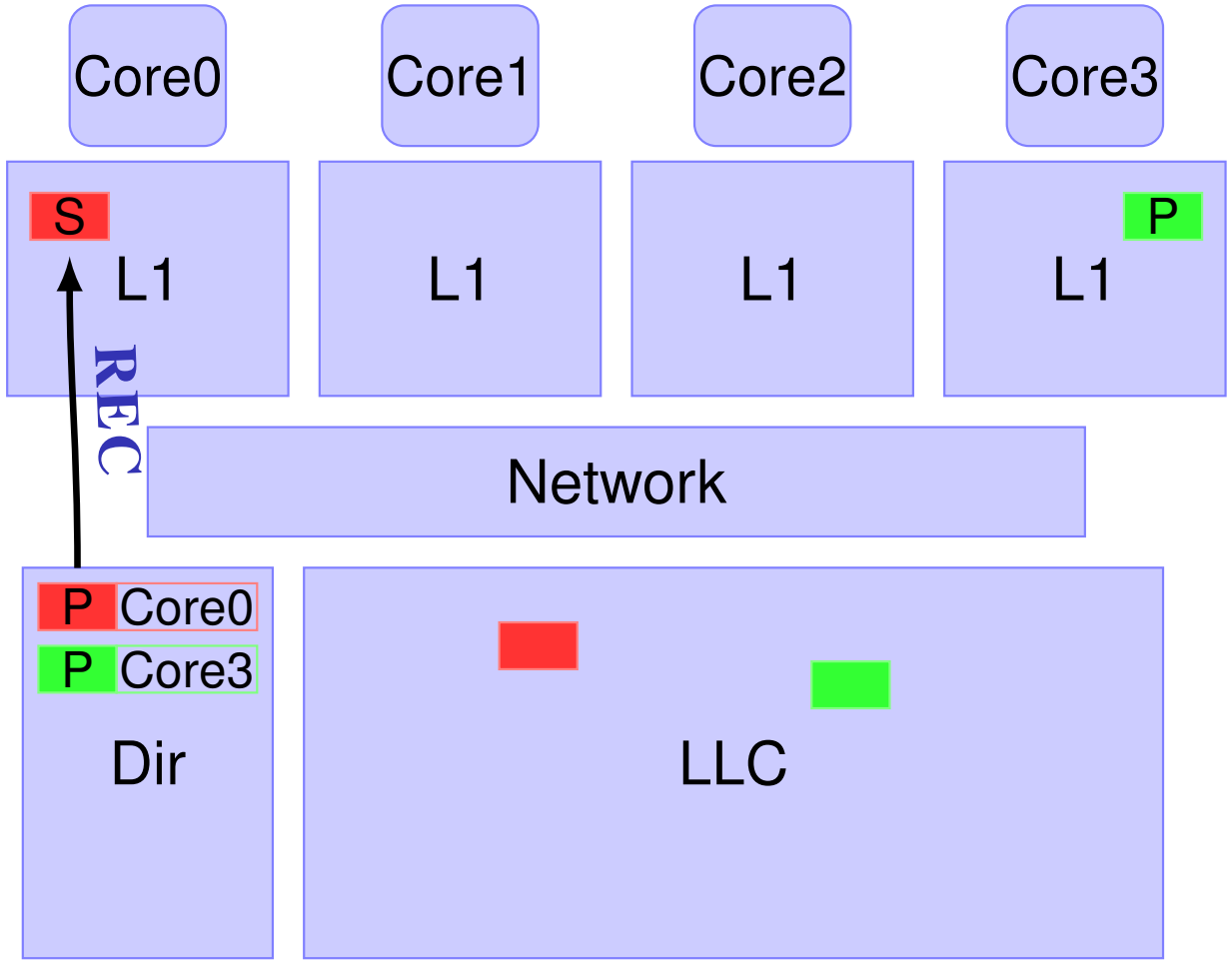
PRIVATE TO SHARED TRANSITION



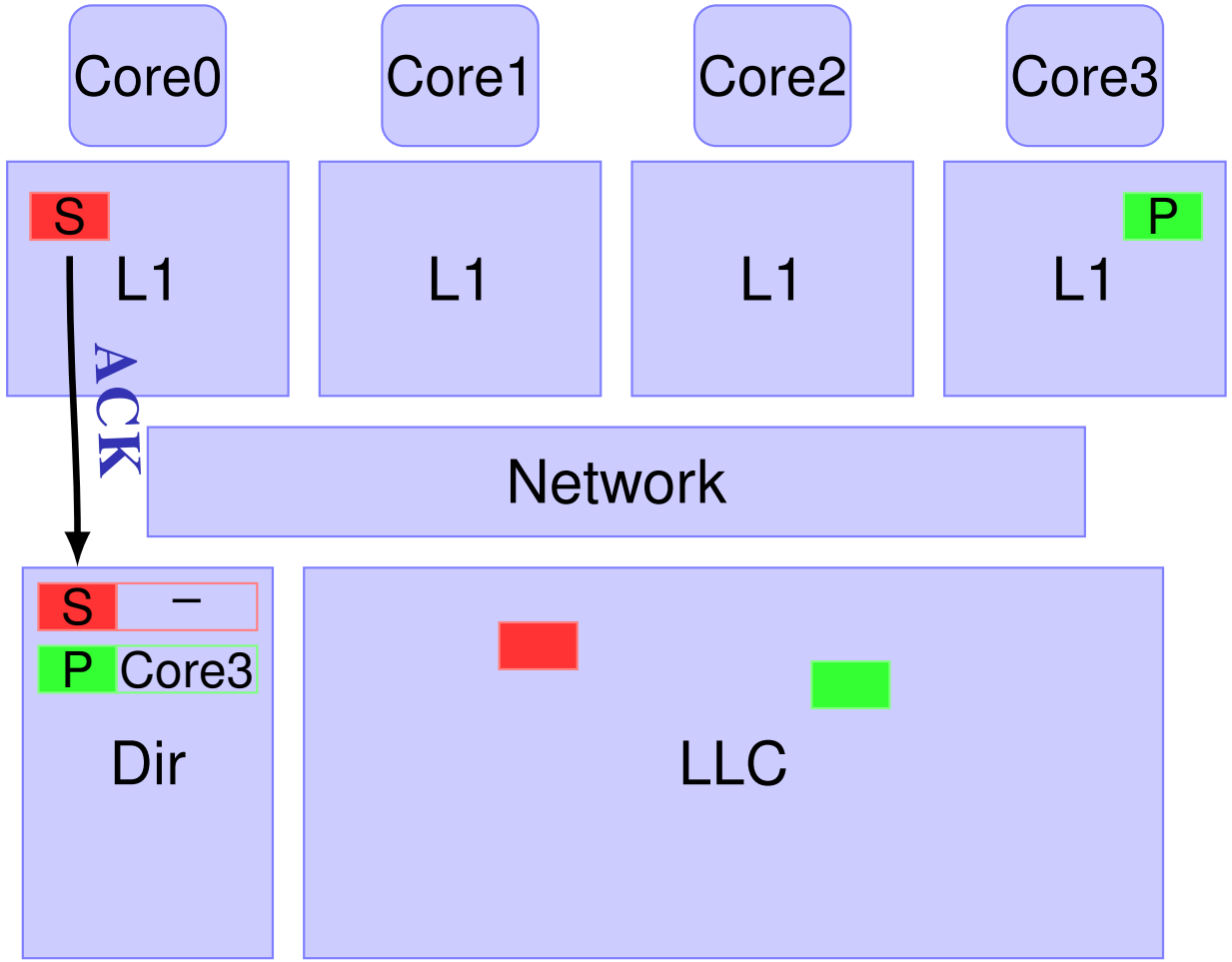
PRIVATE TO SHARED TRANSITION



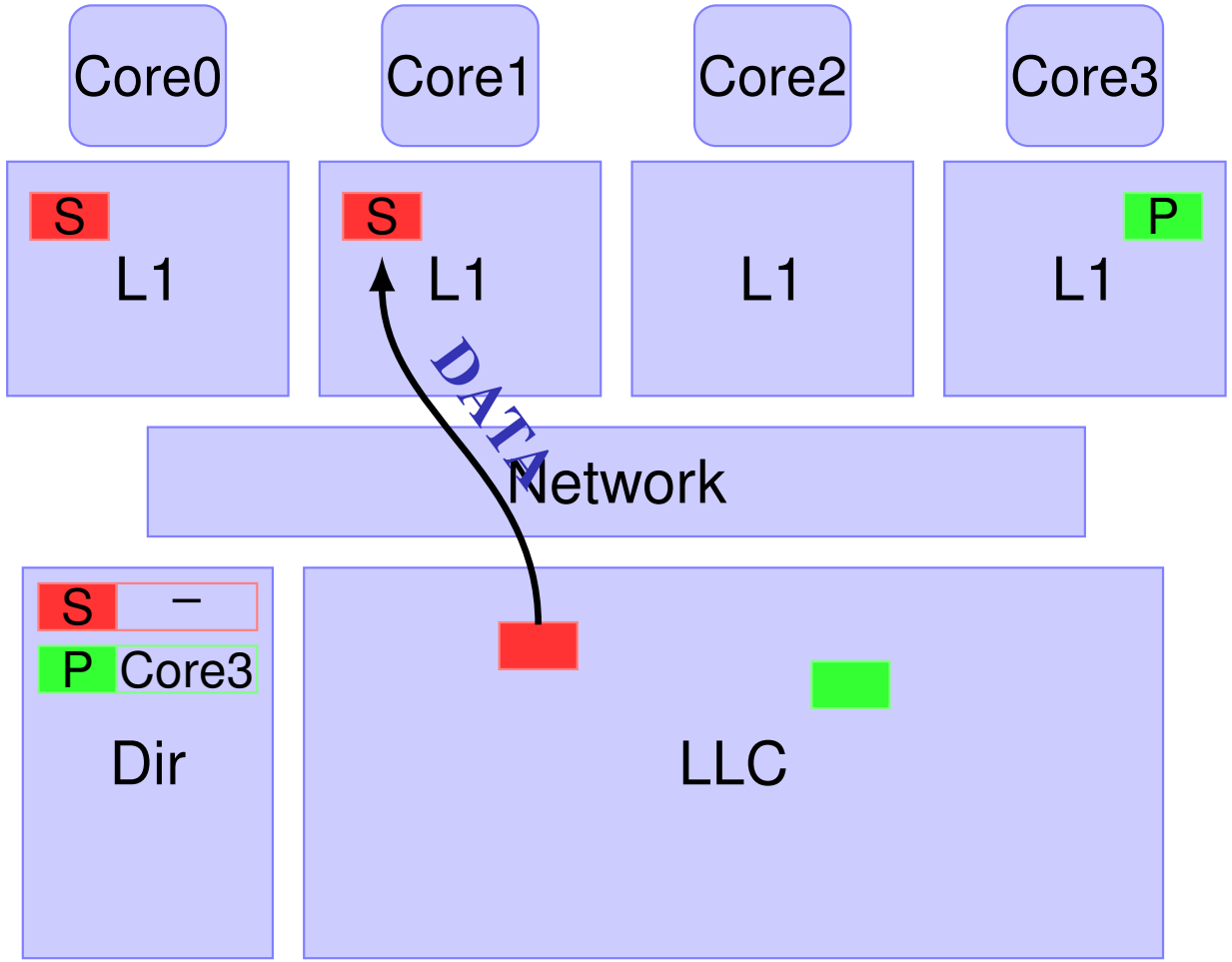
PRIVATE TO SHARED TRANSITION



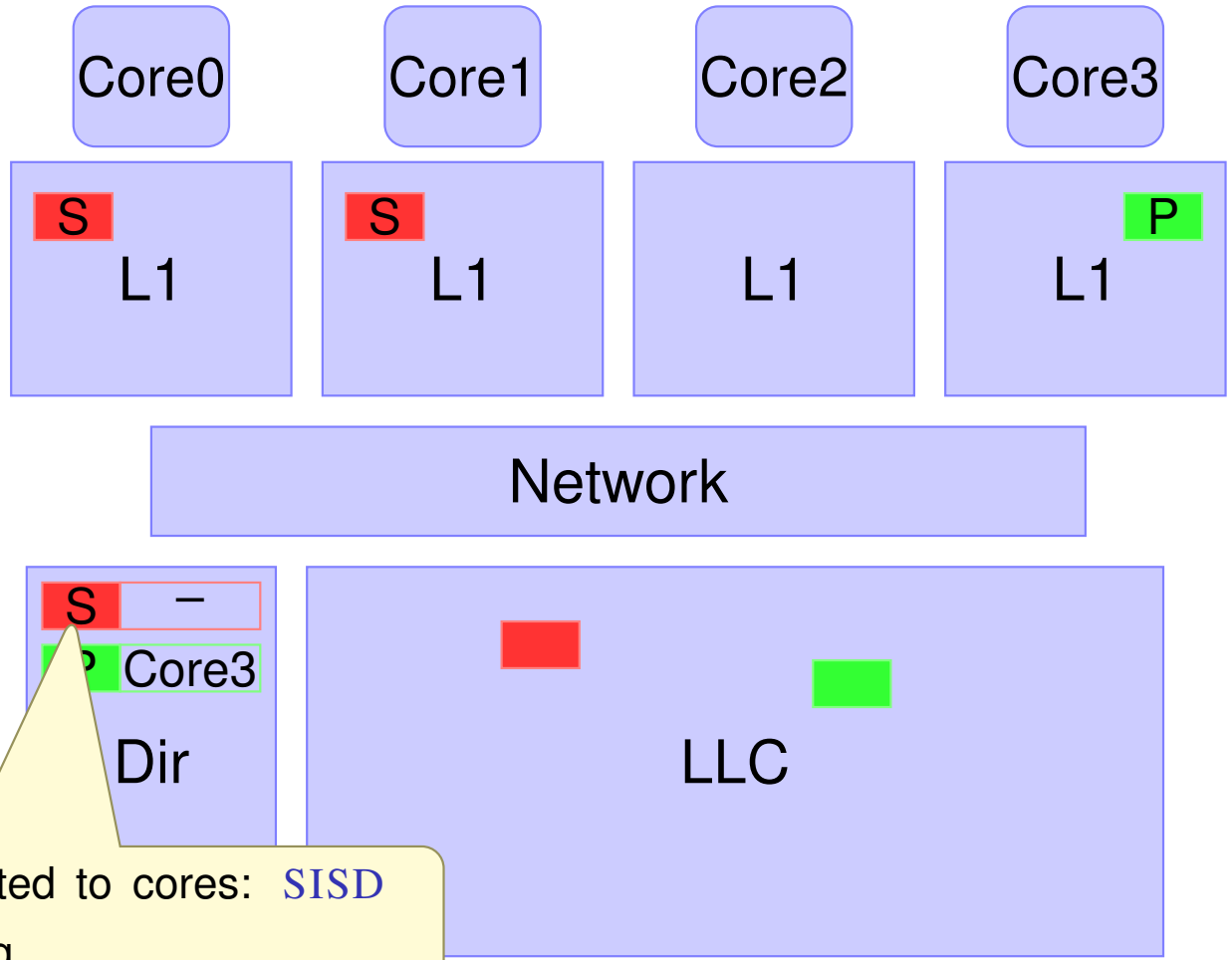
PRIVATE TO SHARED TRANSITION



PRIVATE TO SHARED TRANSITION



PRIVATE TO SHARED TRANSITION

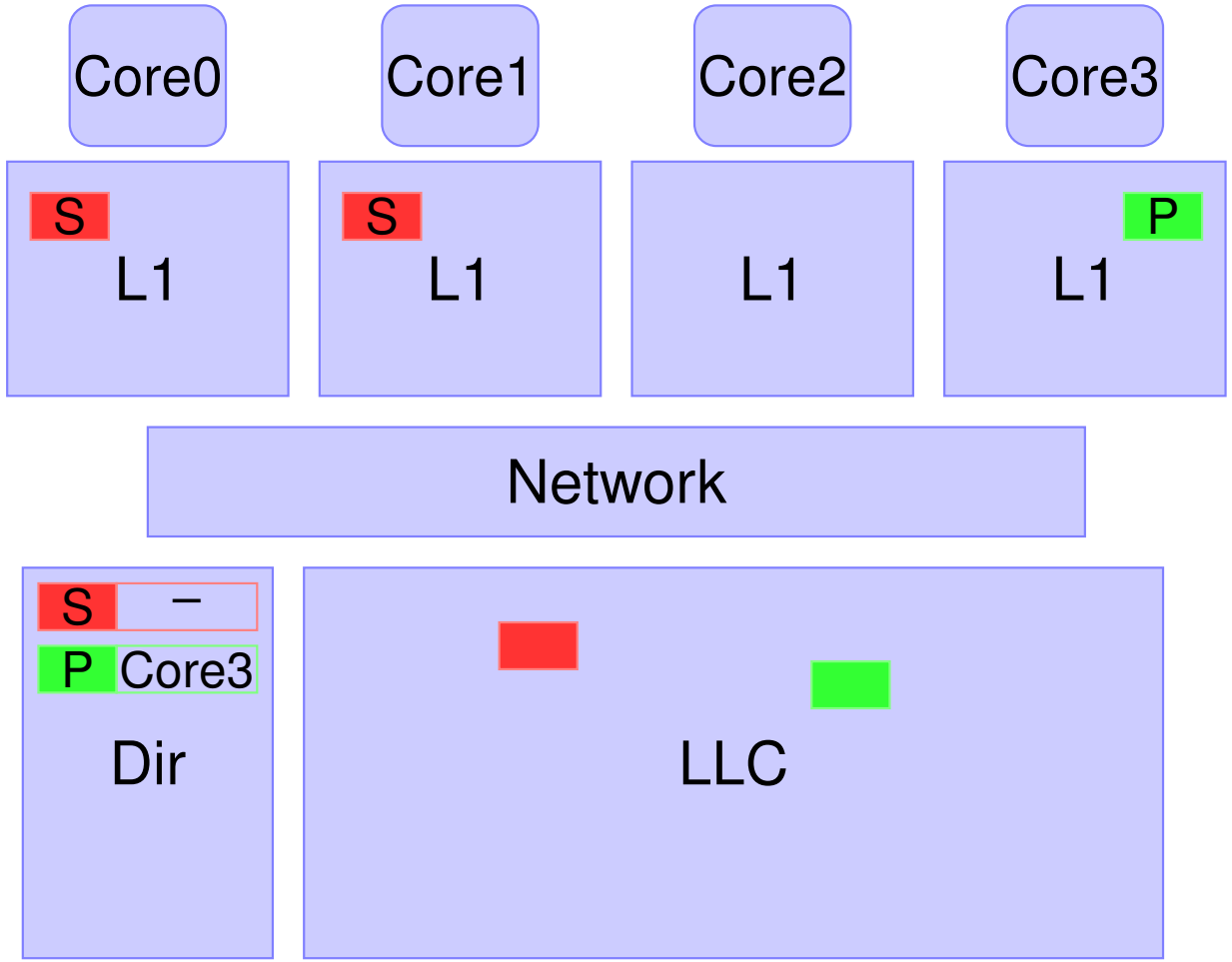


Coherence delegated to cores: SISD

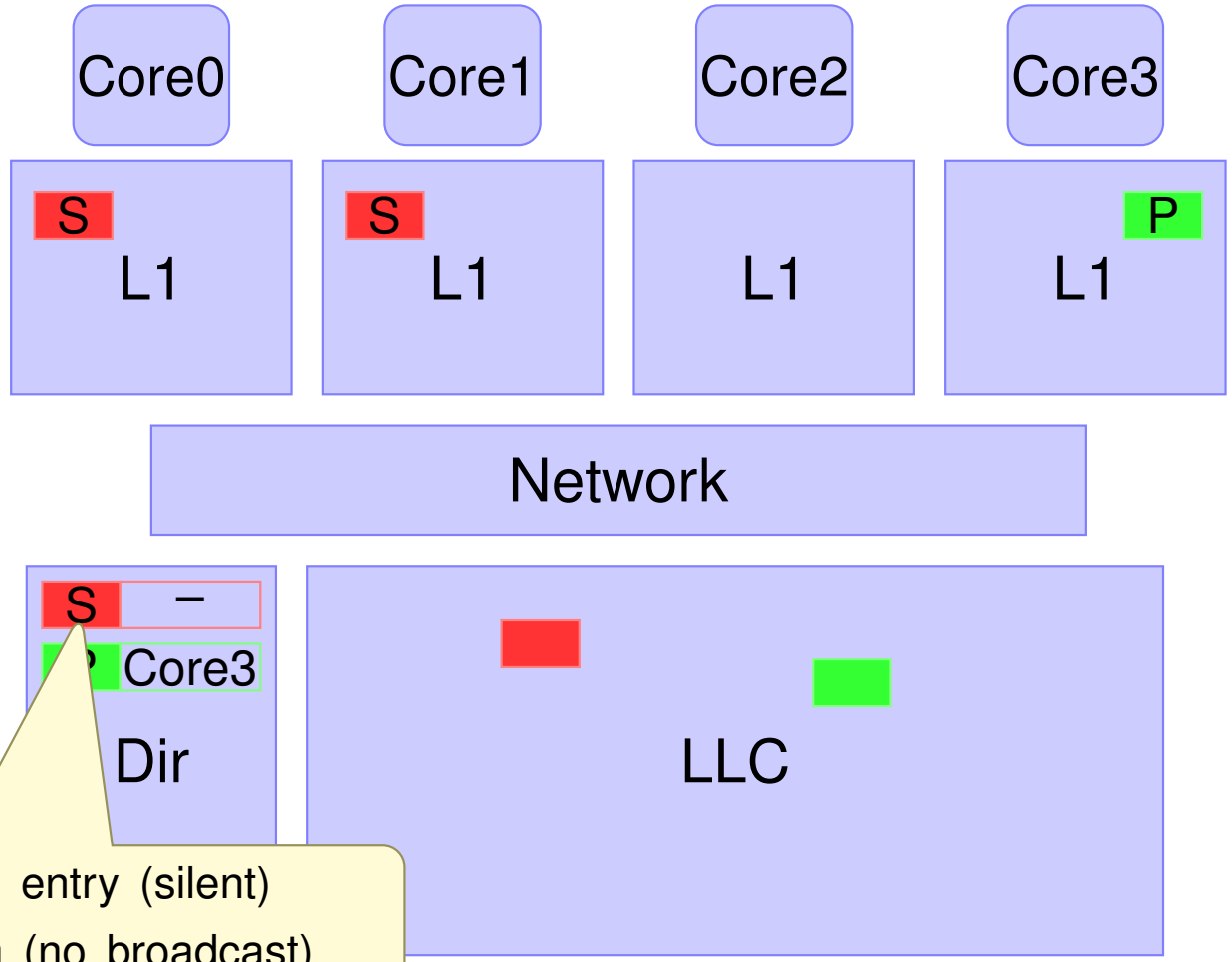
⇒ Wide sharing

⇒ No sharing vector

EVICTION OF SHARED DIRECTORY ENTRIES



EVICTION OF SHARED DIRECTORY ENTRIES

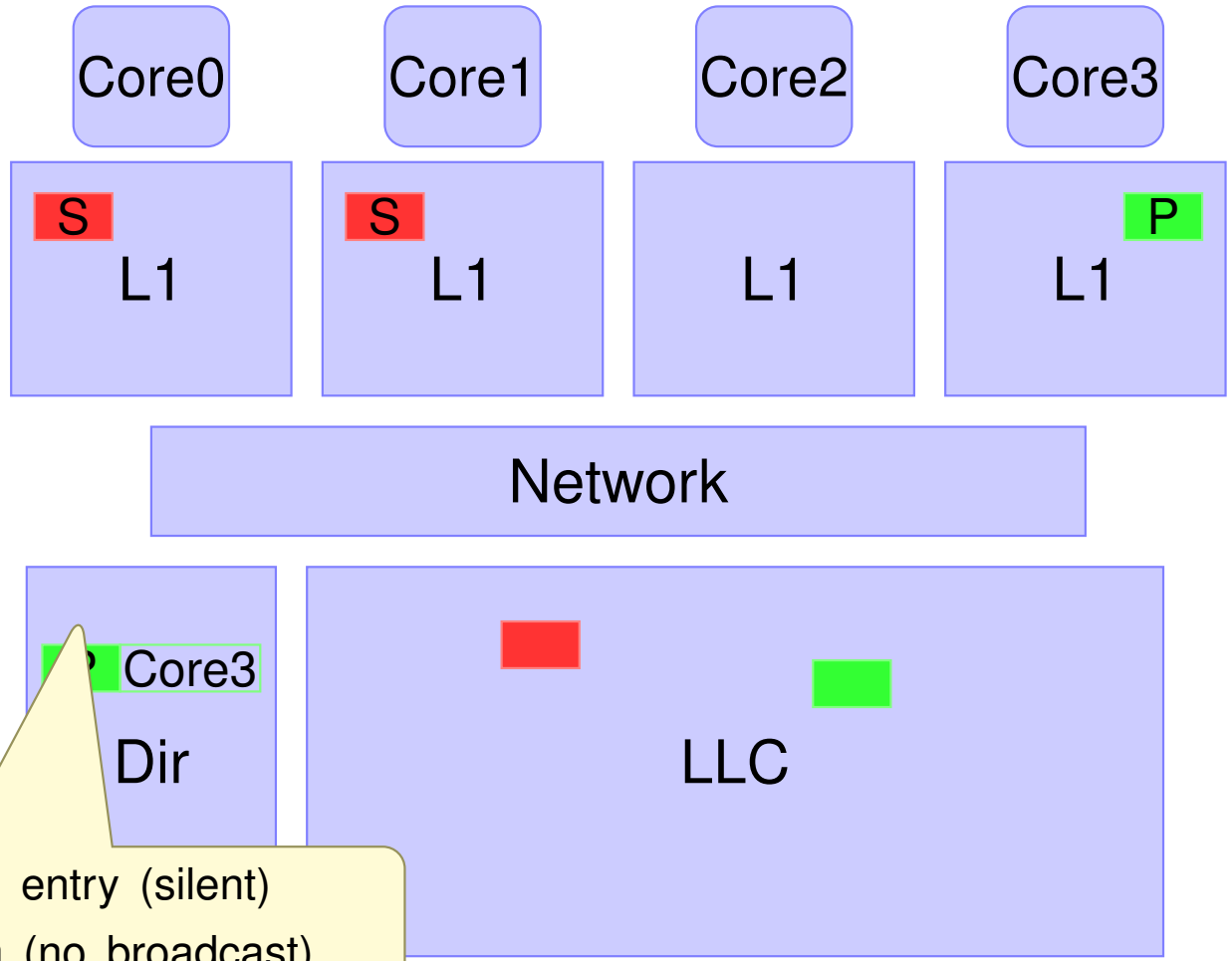


Eviction of shared entry (silent)

⇒ No inclusion (no broadcast)

⇒ No backup (self-contained)

EVICTION OF SHARED DIRECTORY ENTRIES

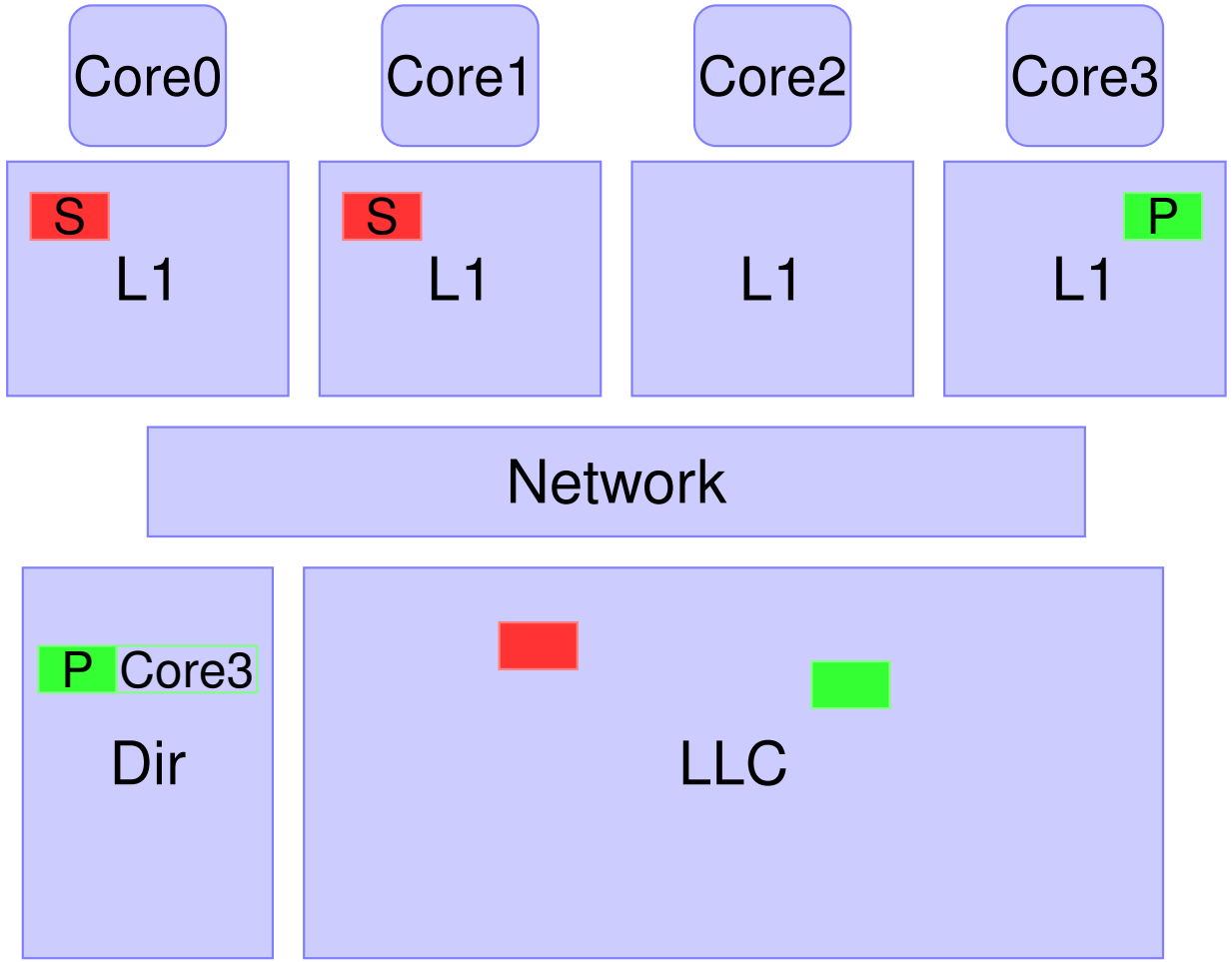


Eviction of shared entry (silent)

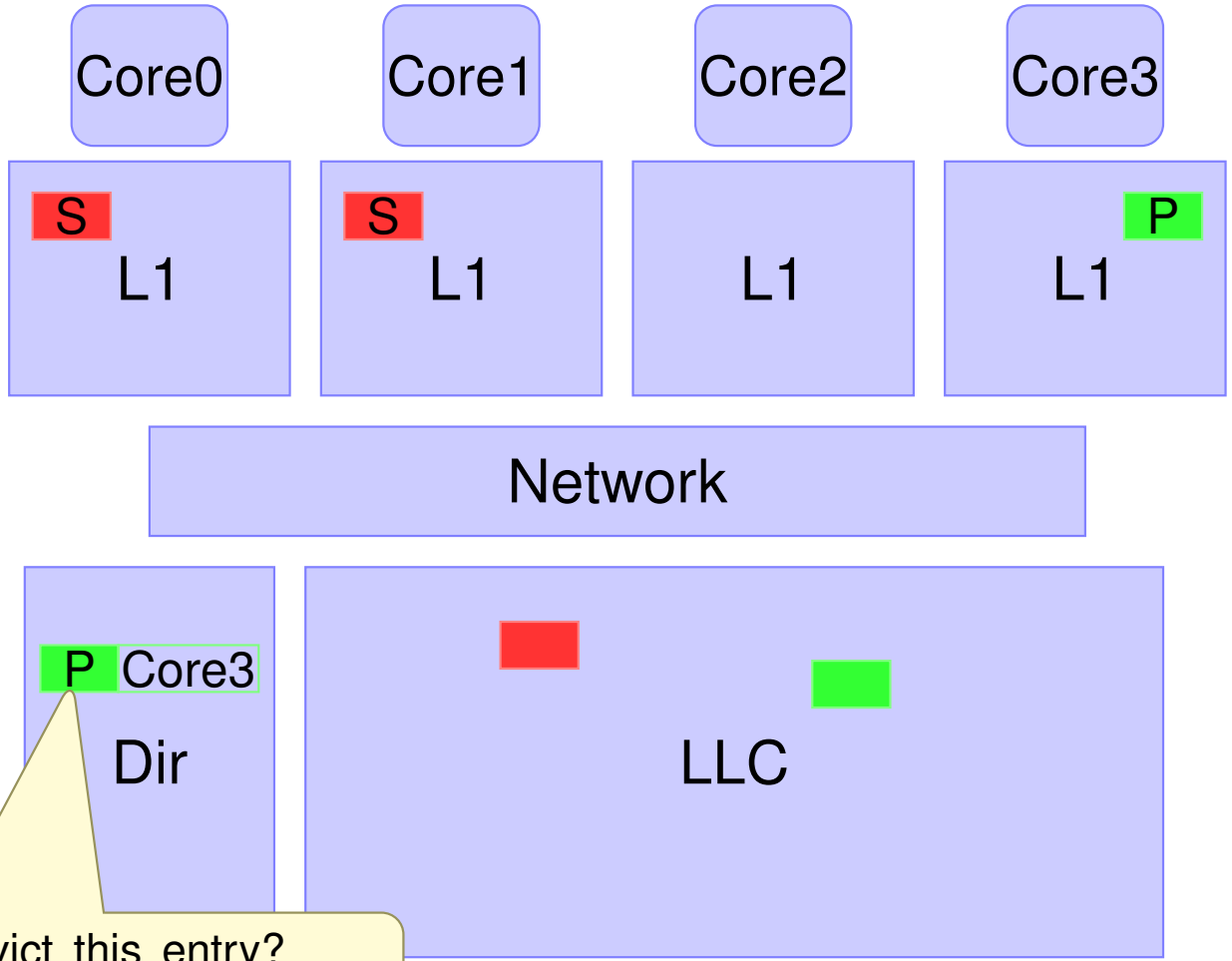
⇒ No inclusion (no broadcast)

⇒ No backup (self-contained)

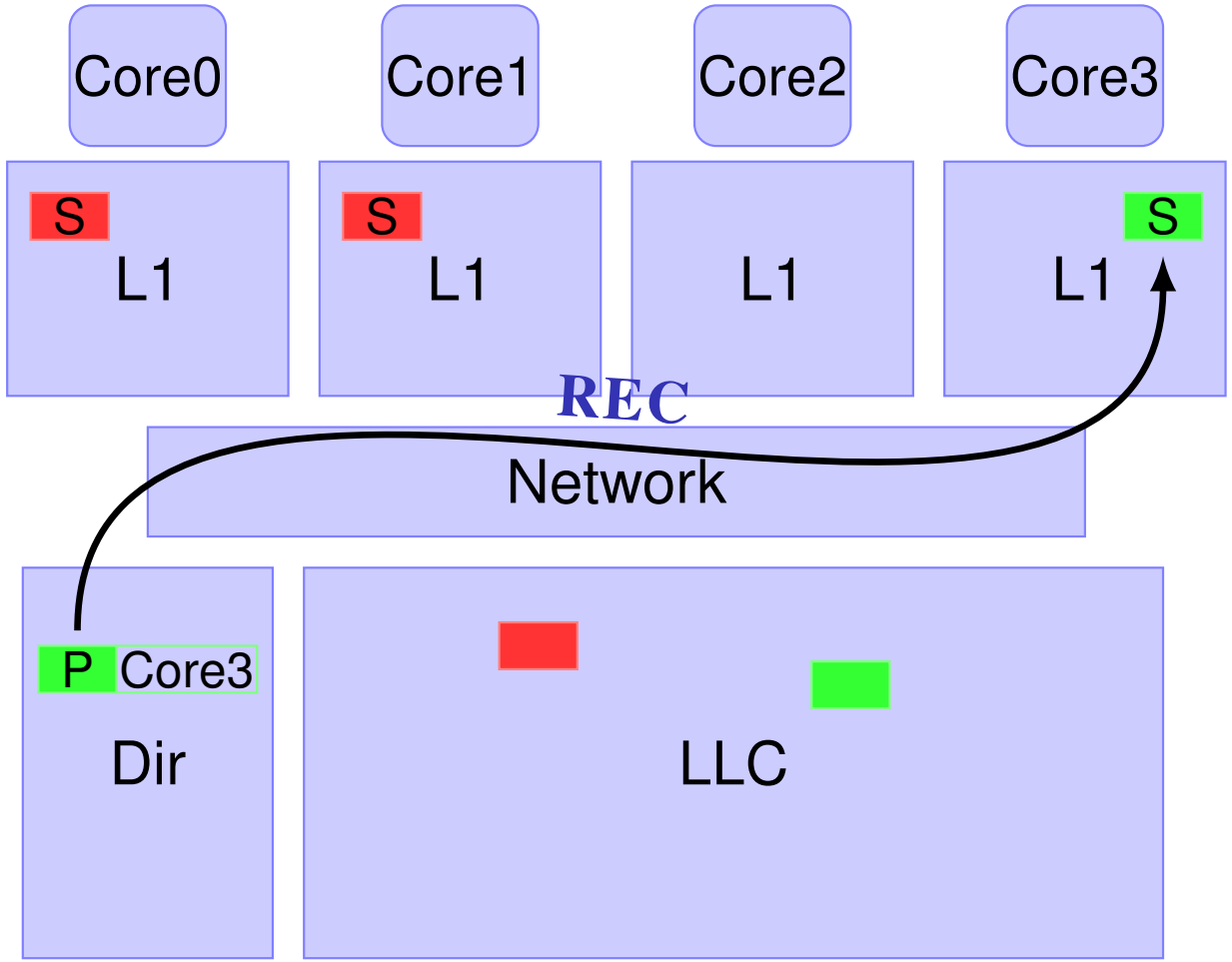
EVICTION OF PRIVATE DIRECTORY ENTRIES



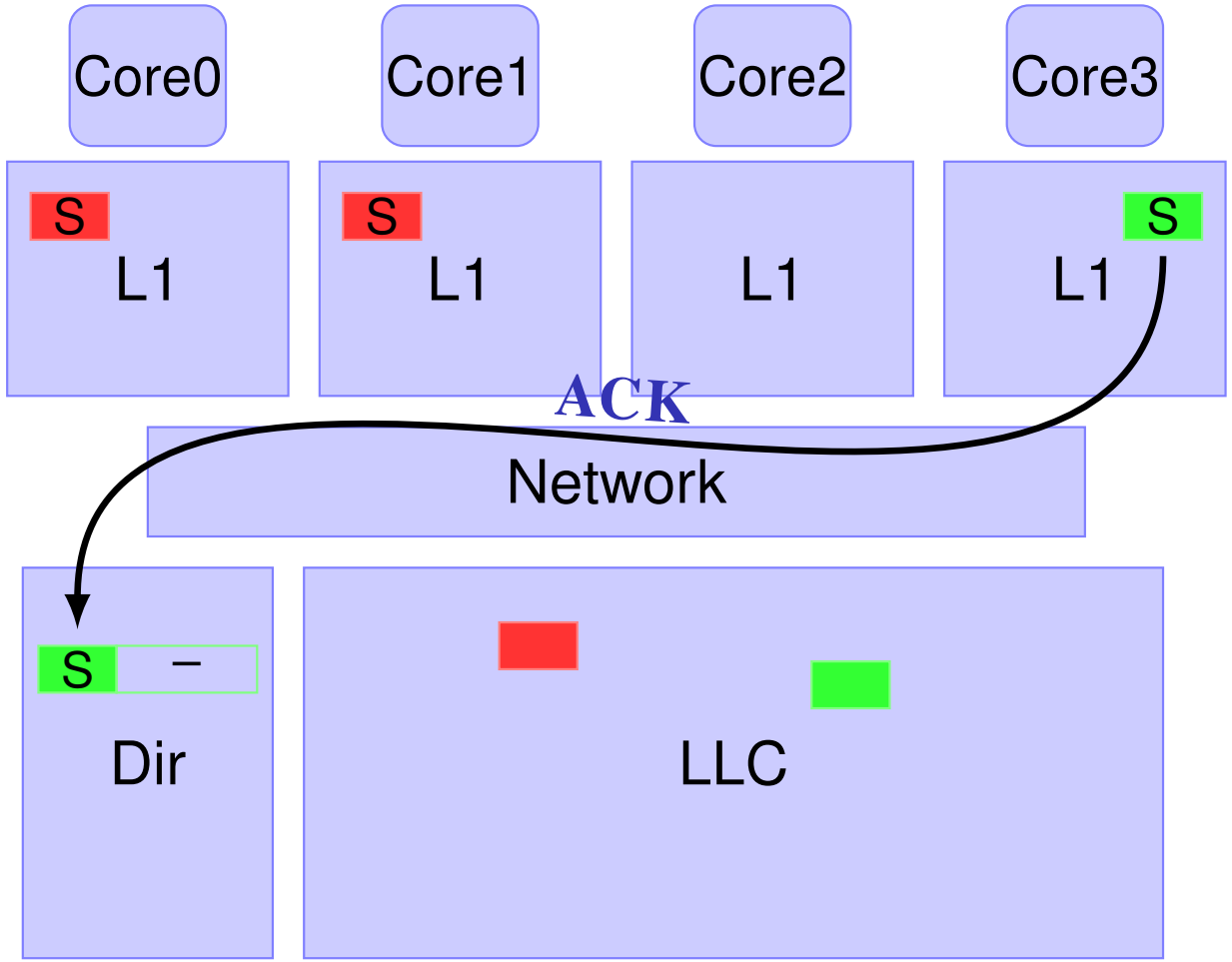
EVICTION OF PRIVATE DIRECTORY ENTRIES



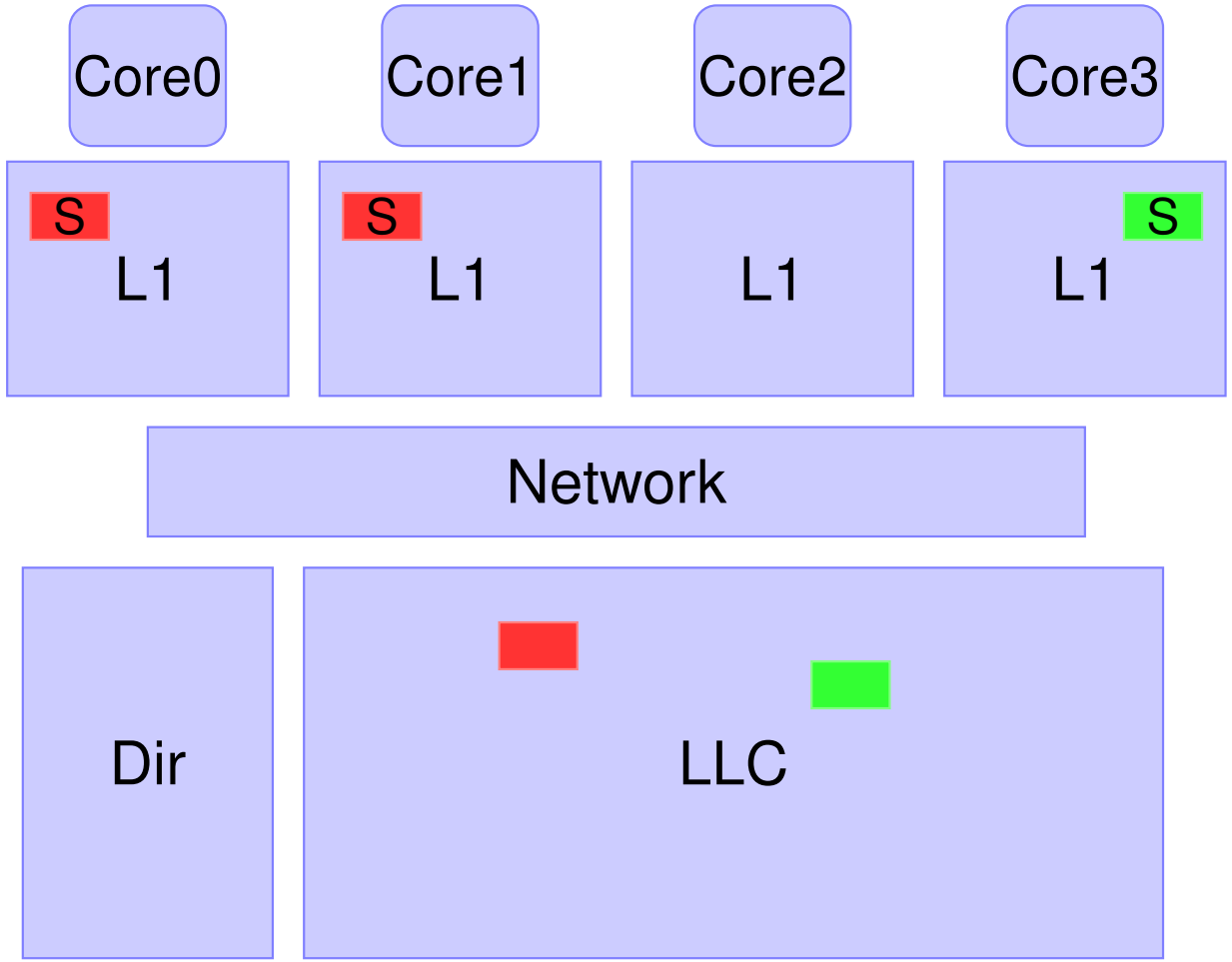
EVICTION OF PRIVATE DIRECTORY ENTRIES



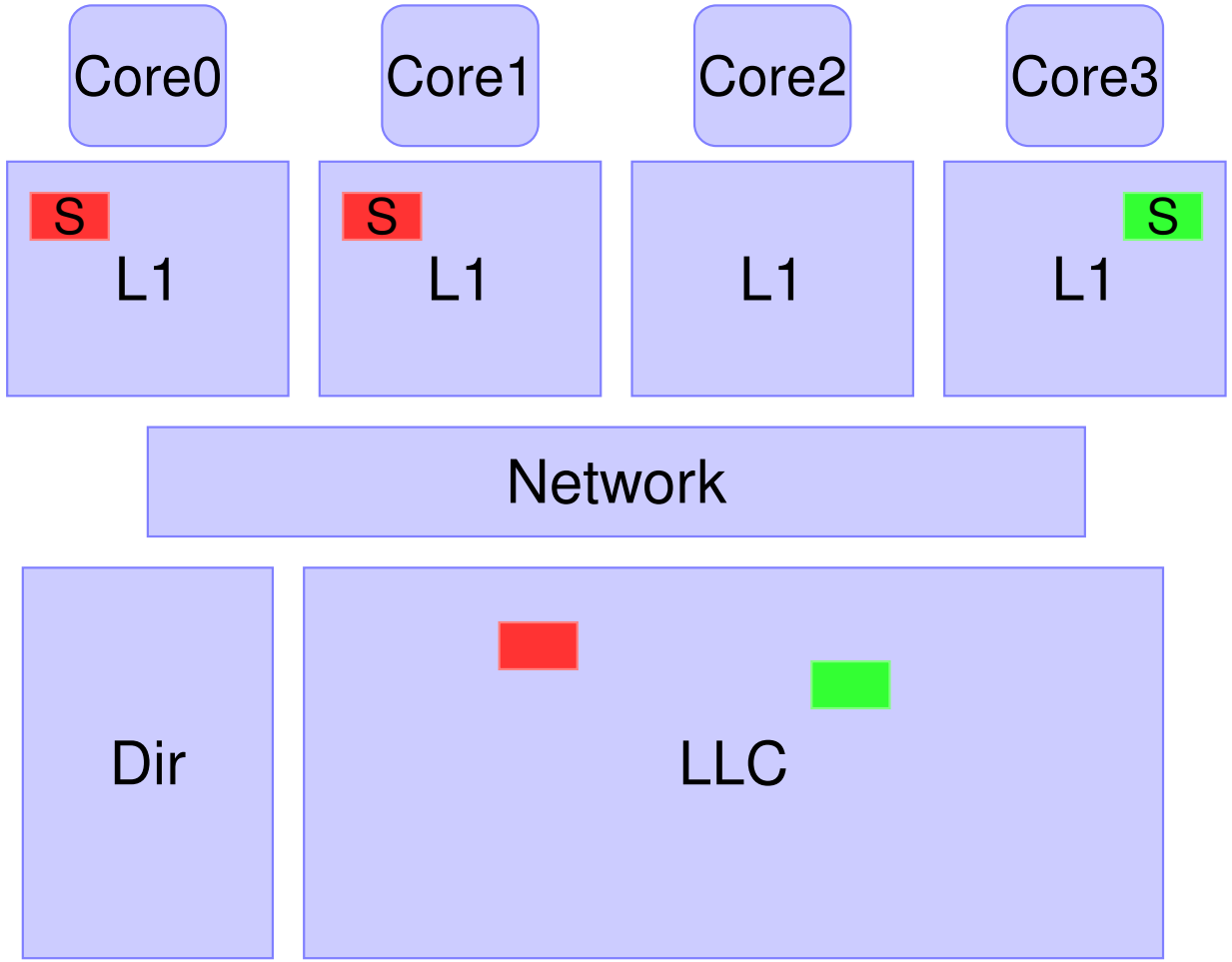
EVICTION OF PRIVATE DIRECTORY ENTRIES



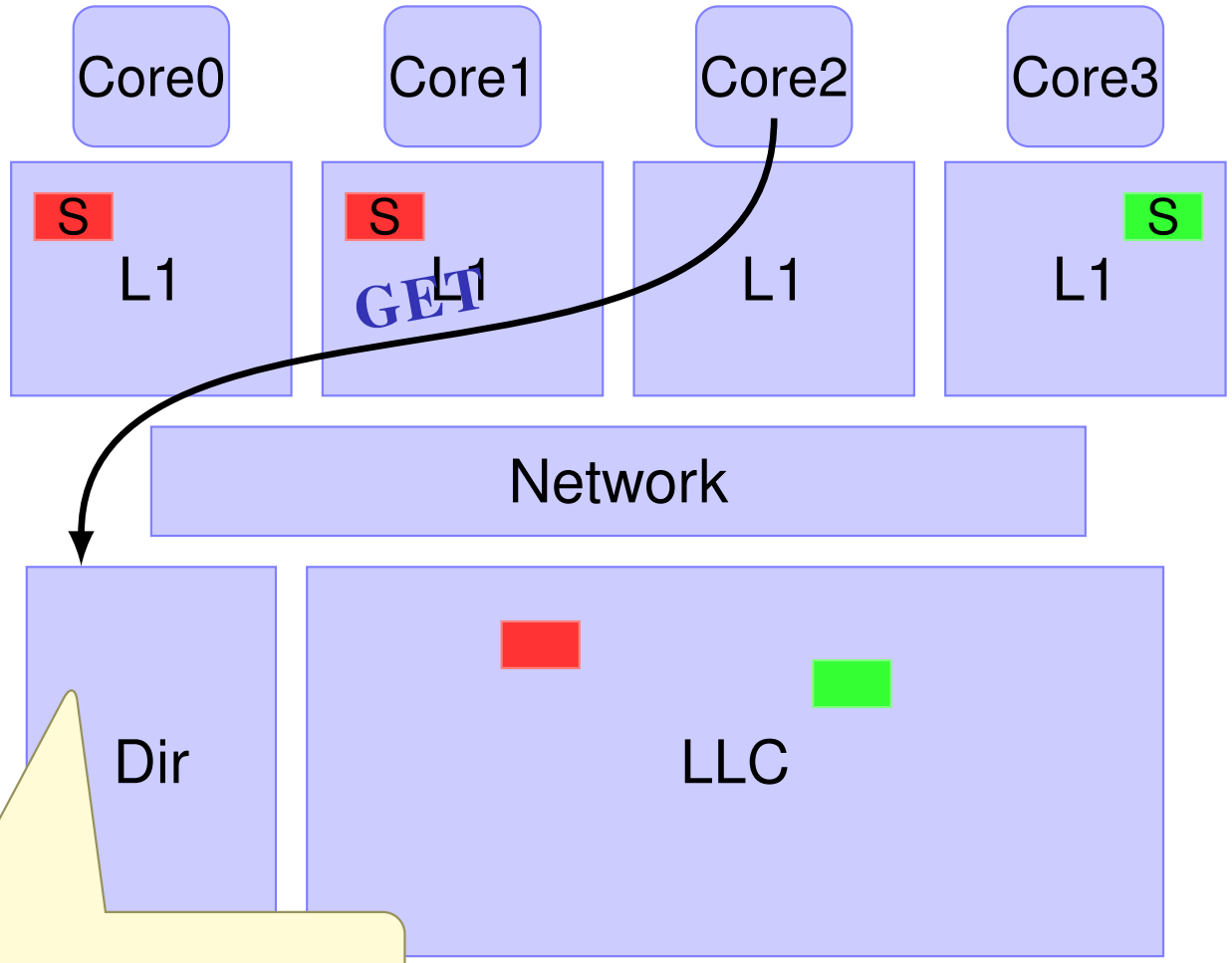
EVICTION OF PRIVATE DIRECTORY ENTRIES



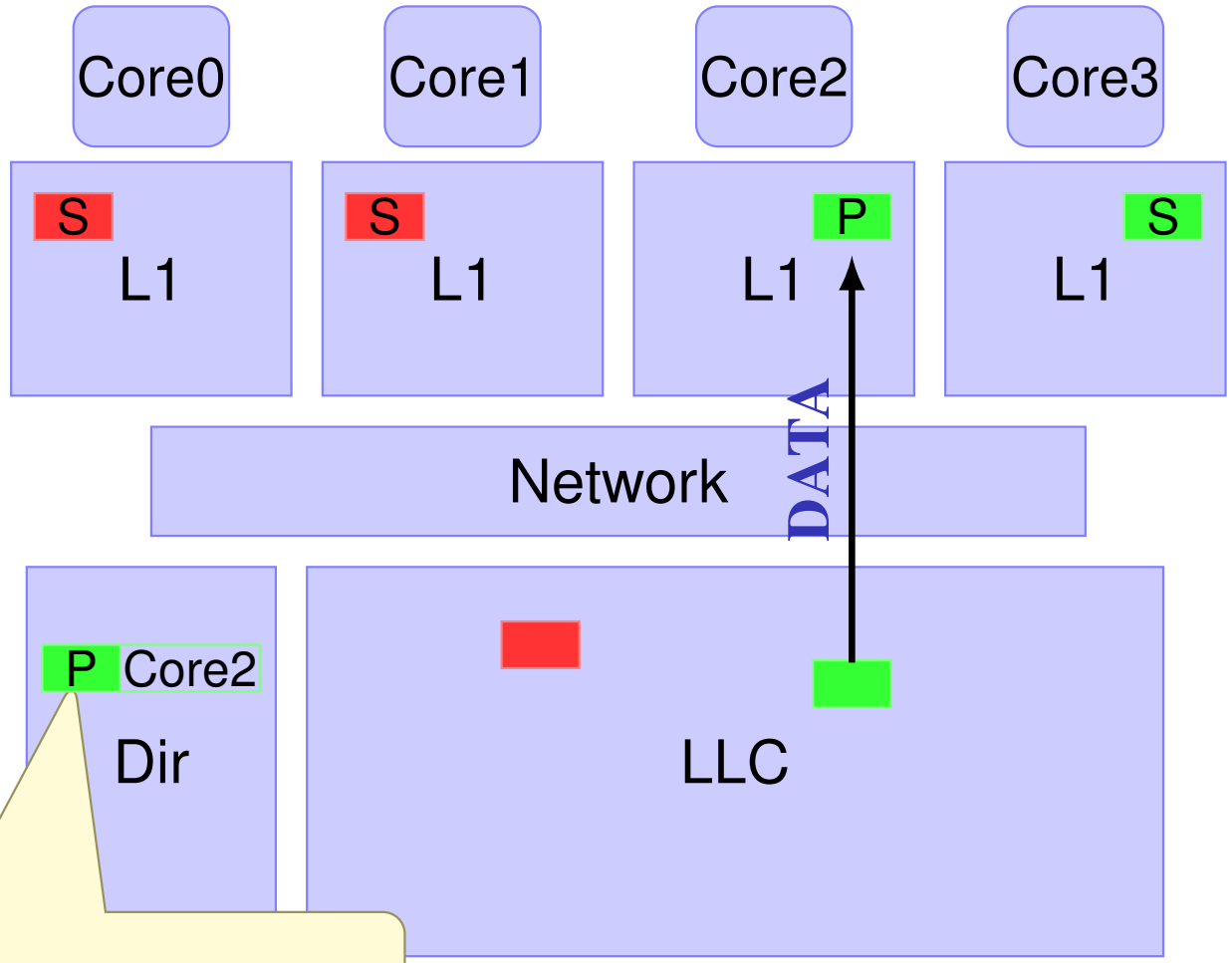
HIDDEN SHARERS



HIDDEN SHARERS



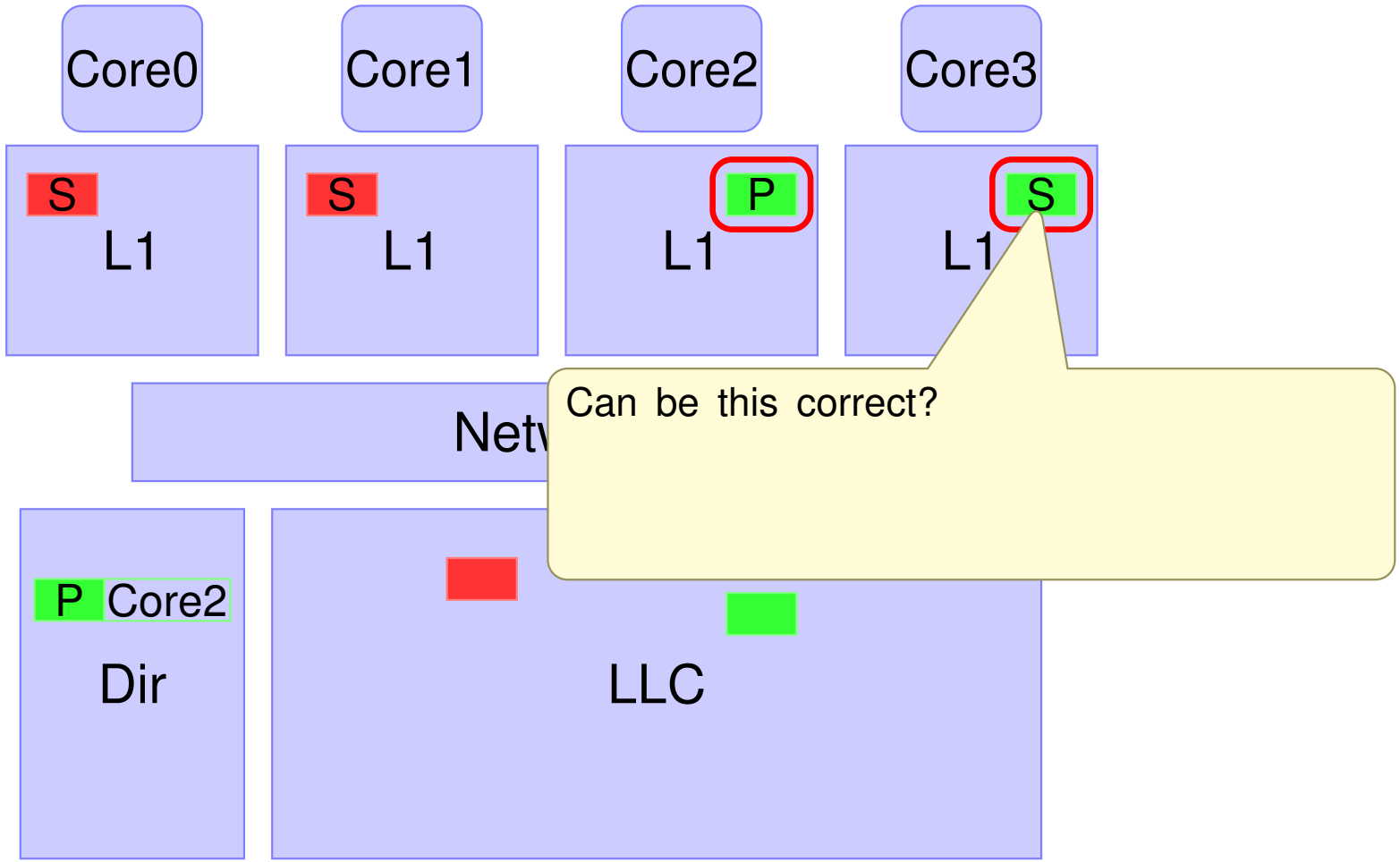
HIDDEN SHARERS



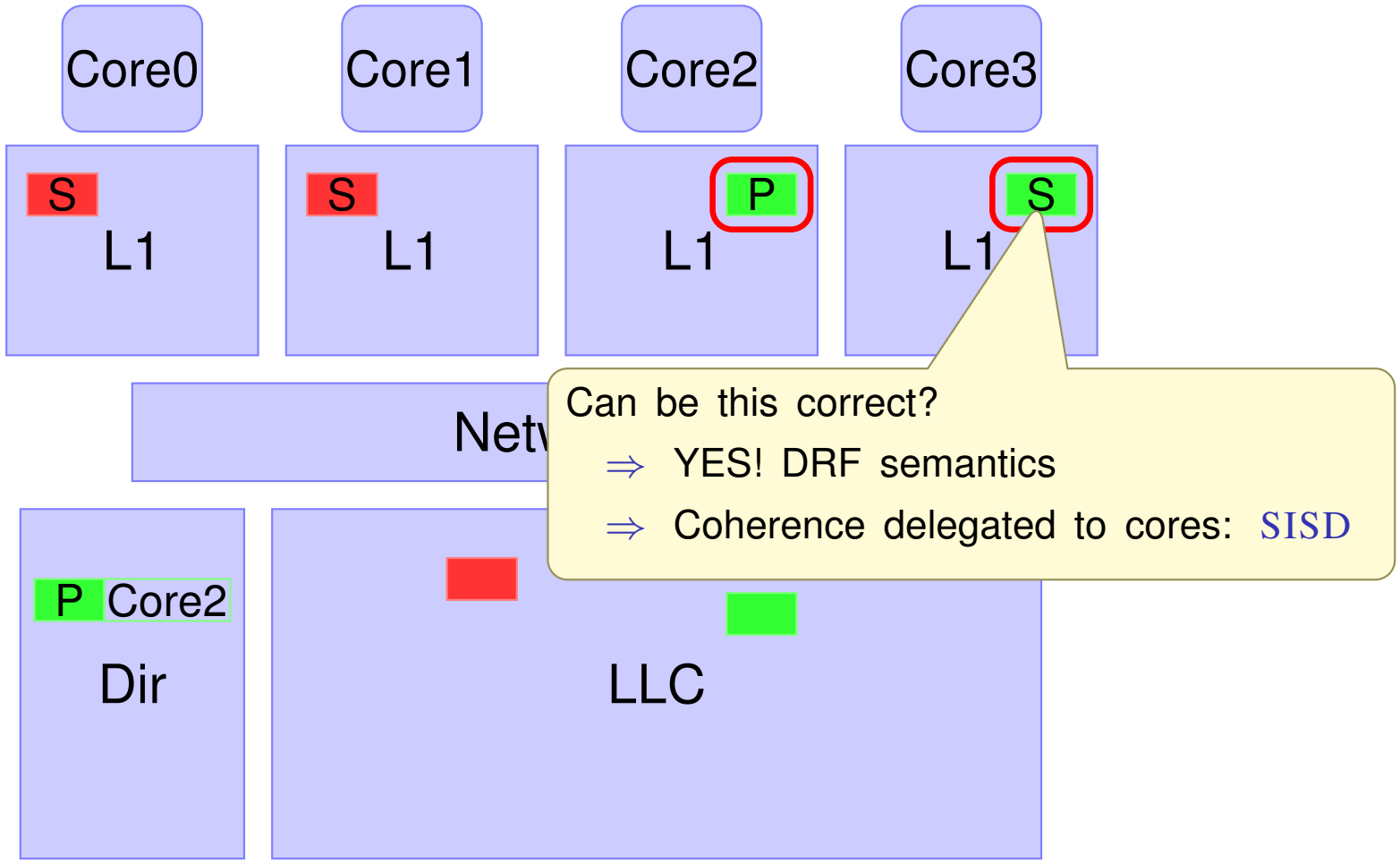
What to do?

⇒ Make it private!

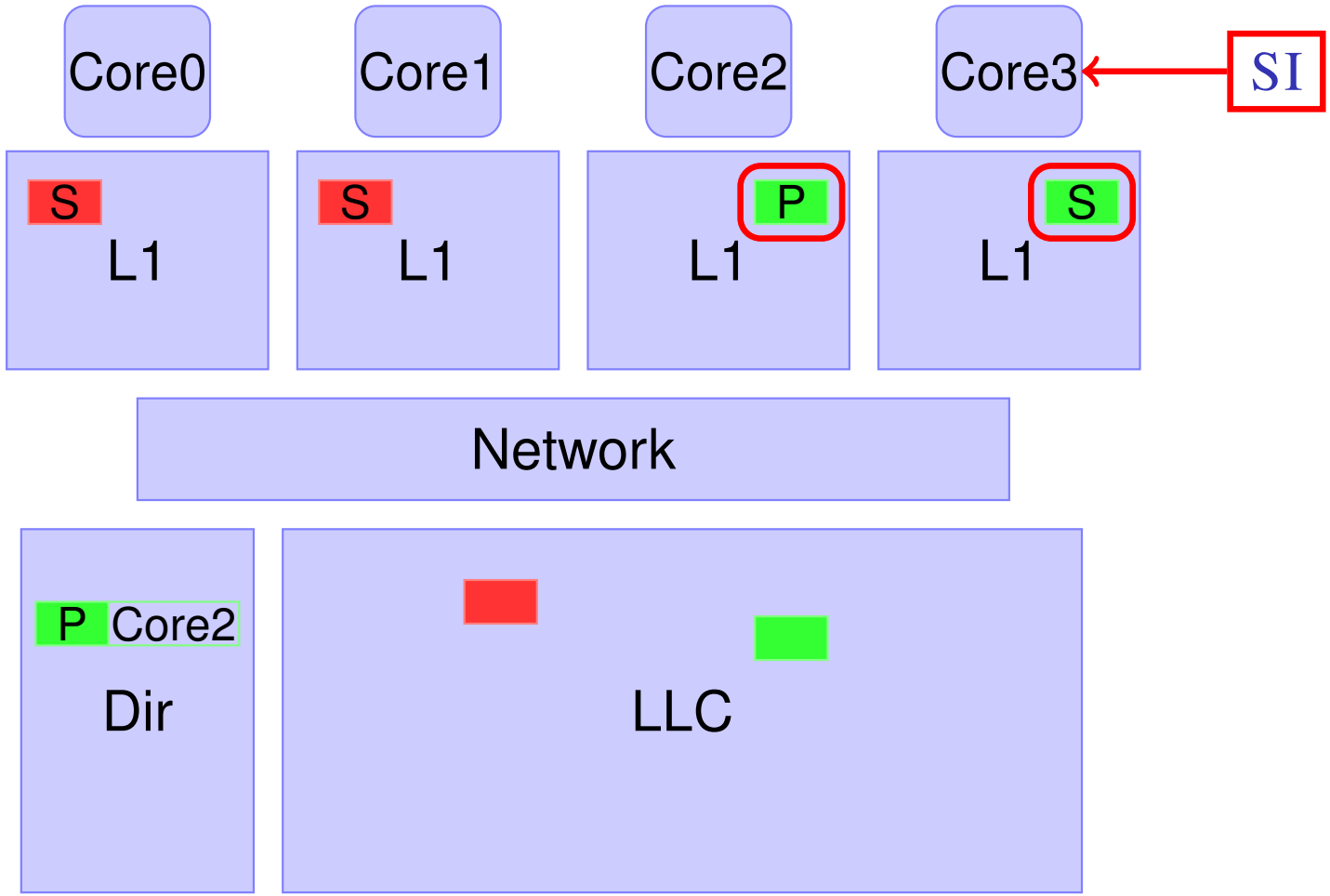
HIDDEN SHARERS



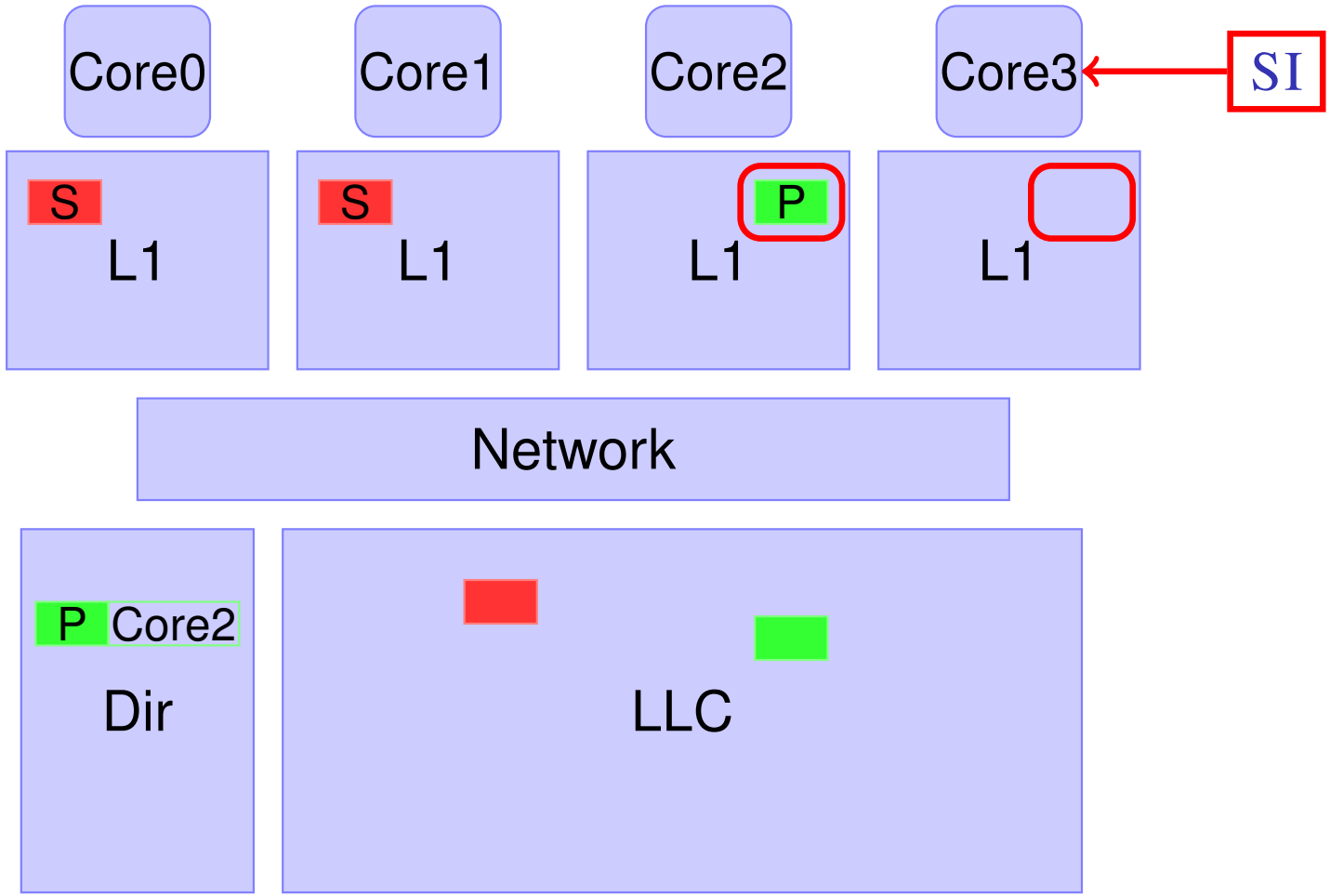
HIDDEN SHARERS



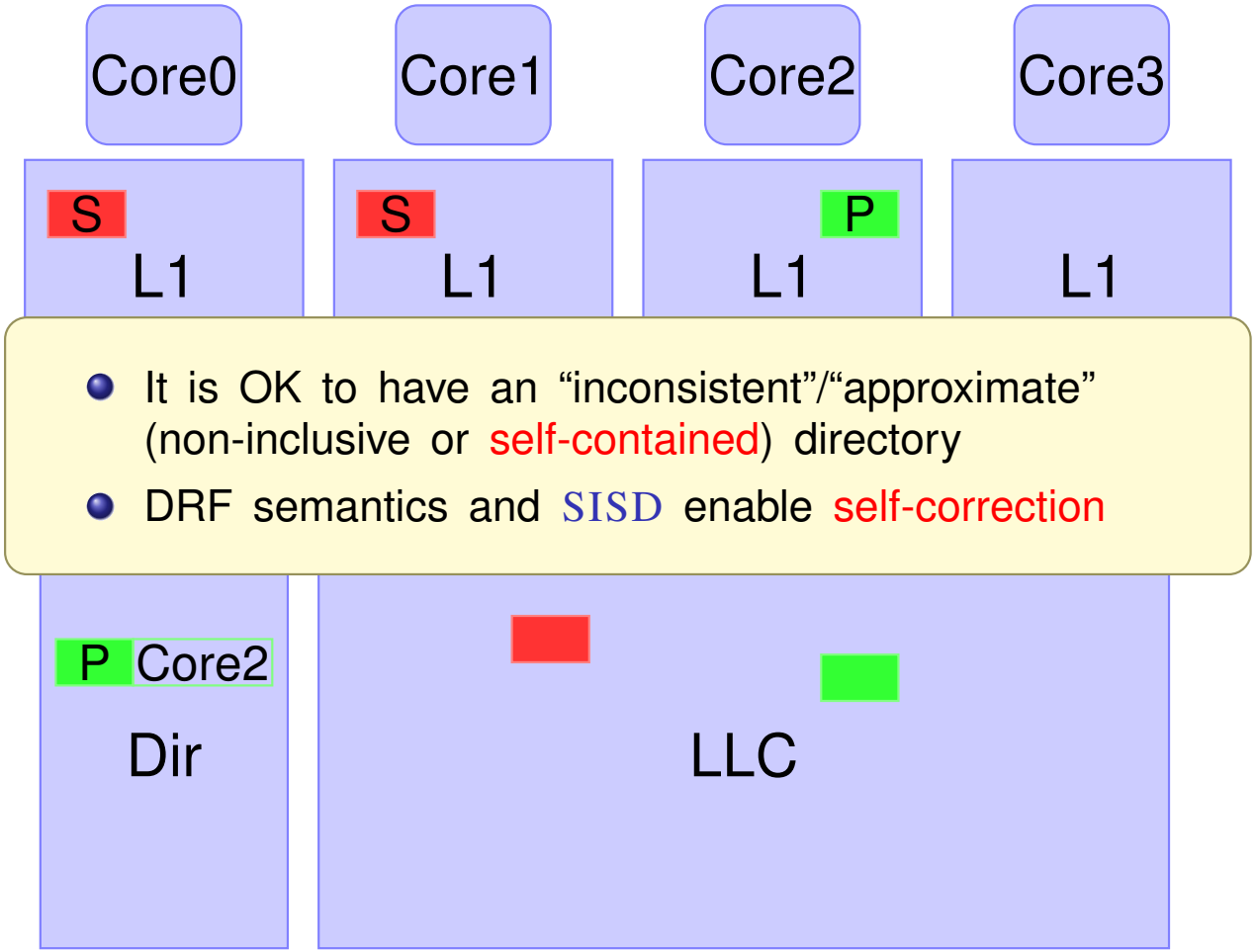
HIDDEN SHARERS



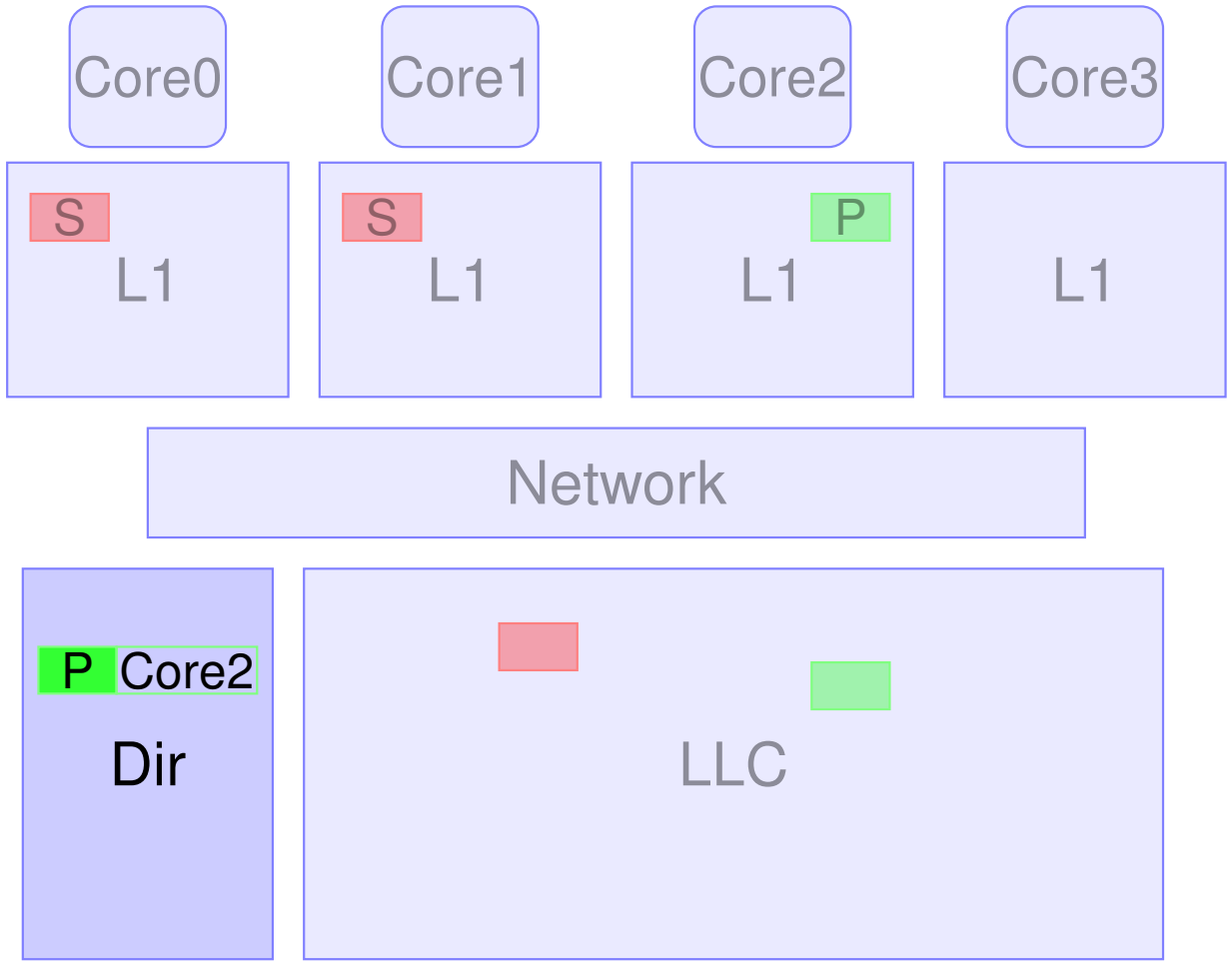
HIDDEN SHARERS



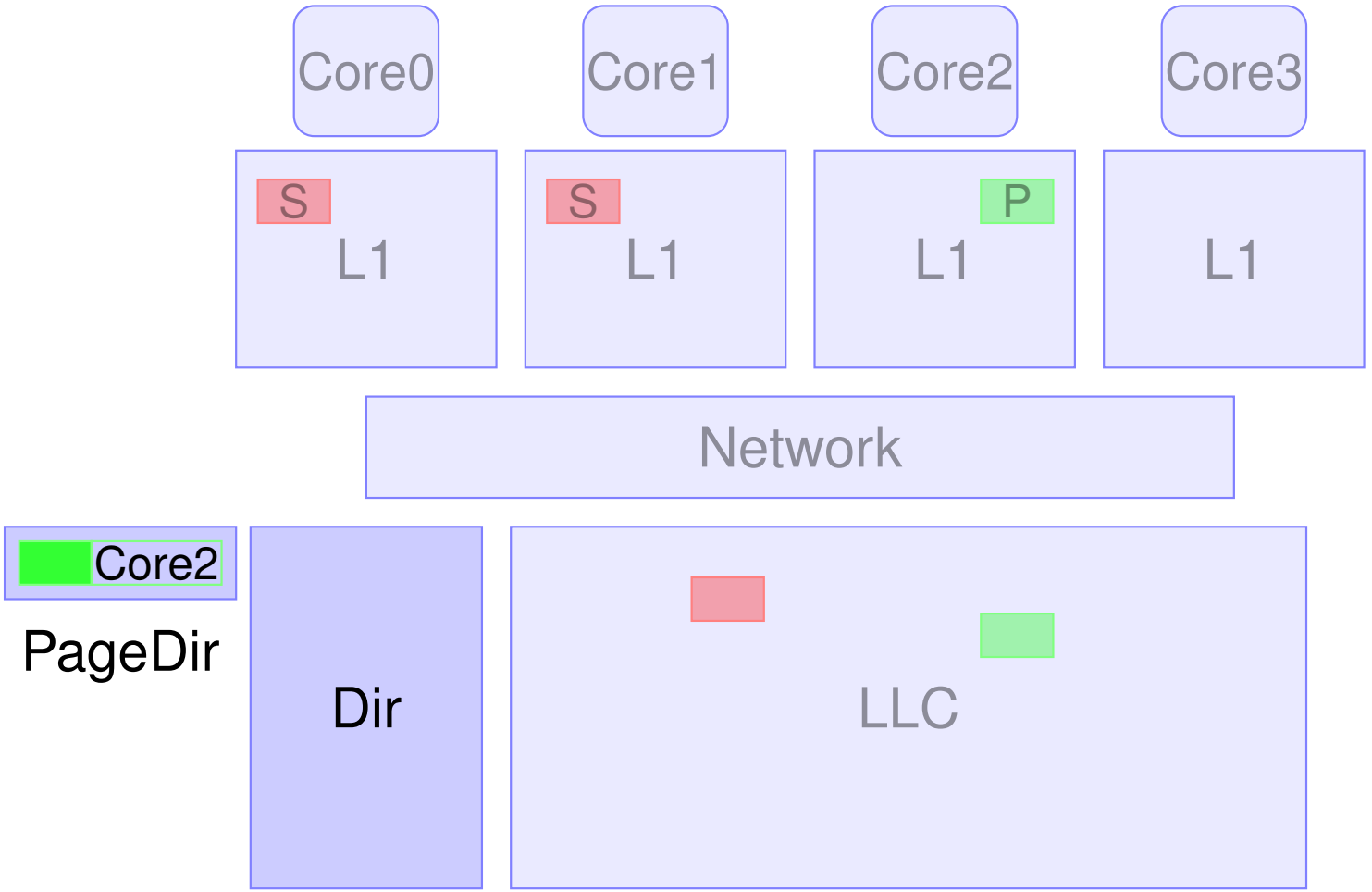
HIDDEN SHARERS



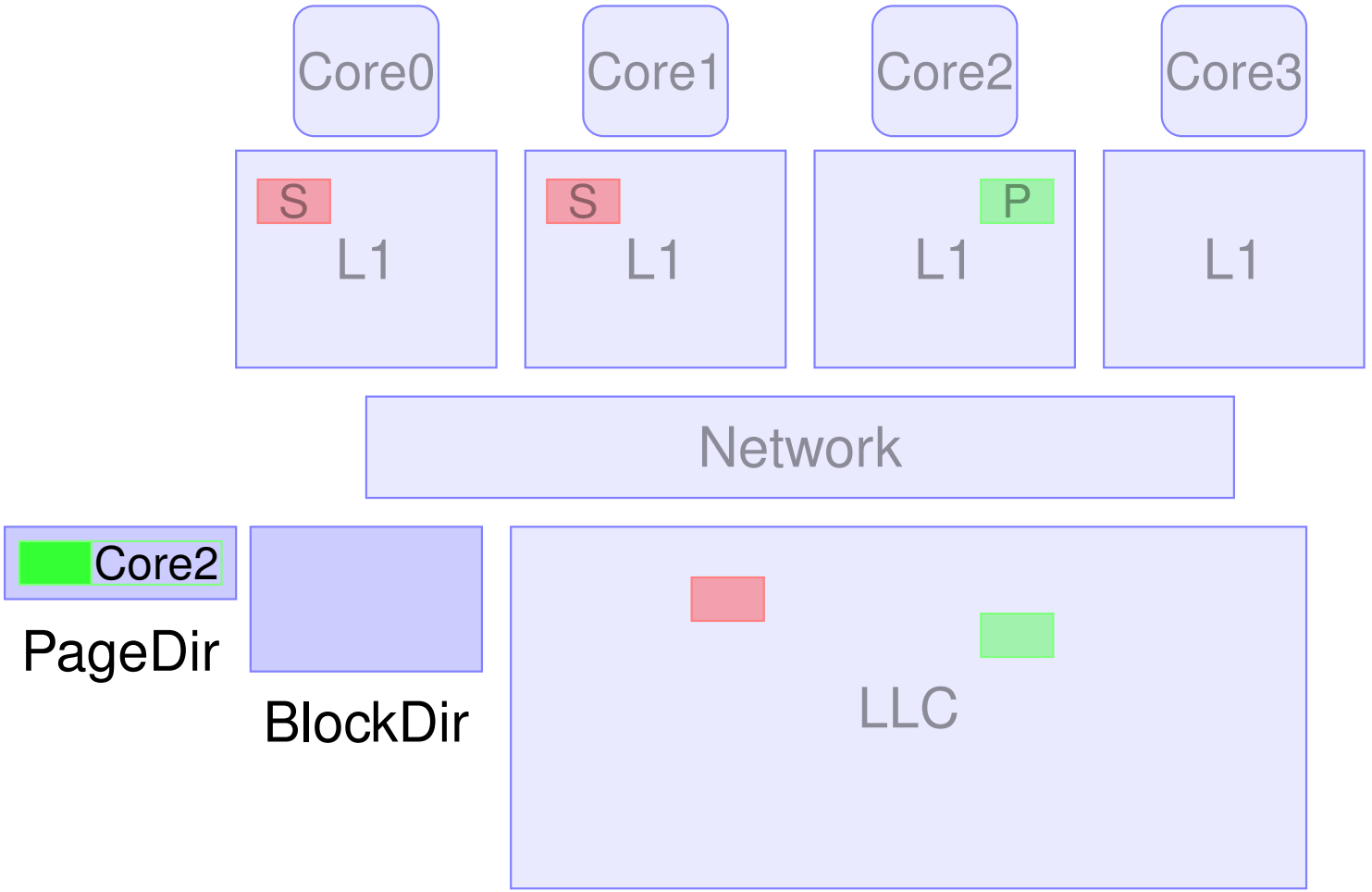
DIRECTORY COMPRESSION



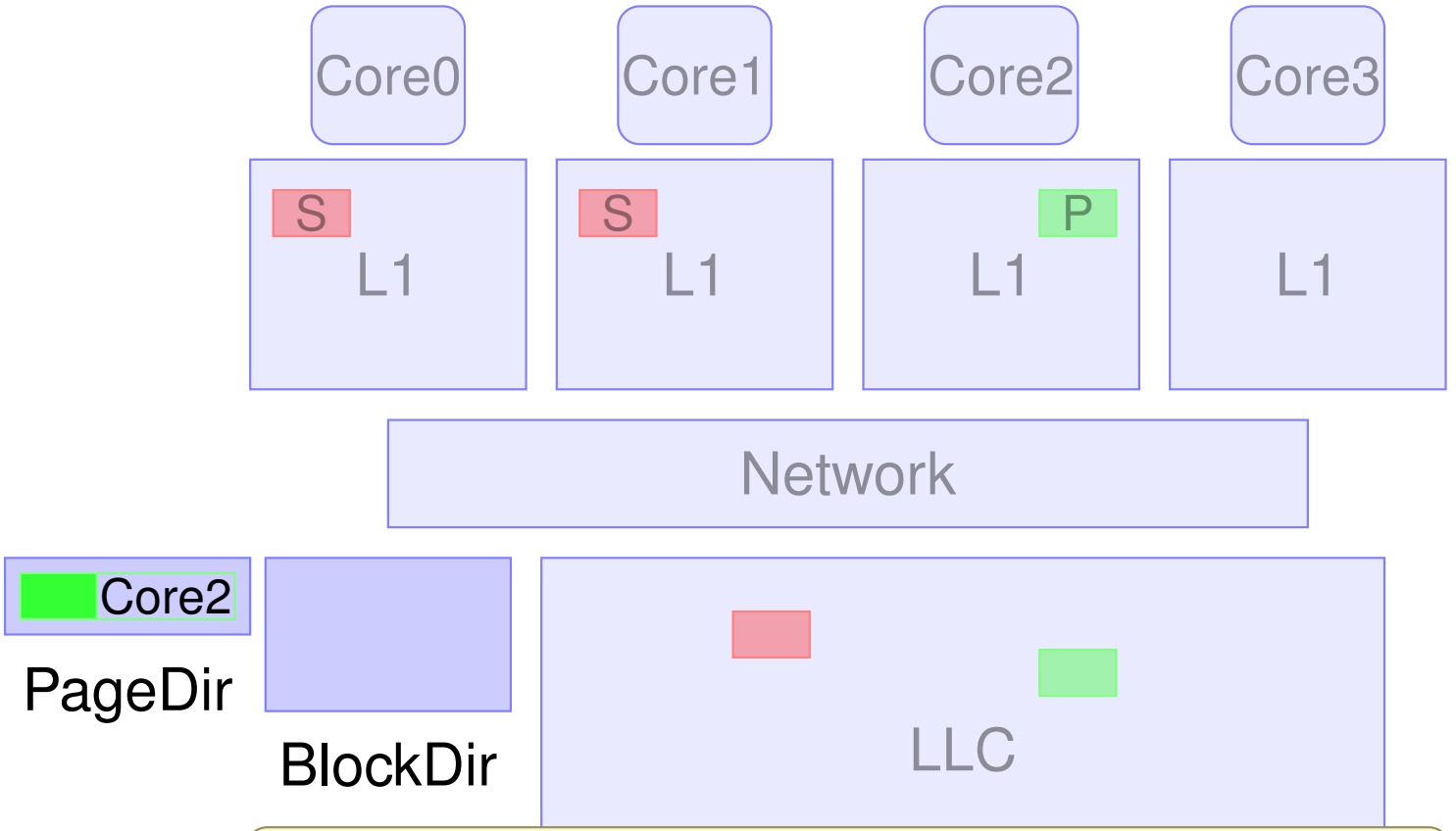
DIRECTORY COMPRESSION



DIRECTORY COMPRESSION



DIRECTORY COMPRESSION



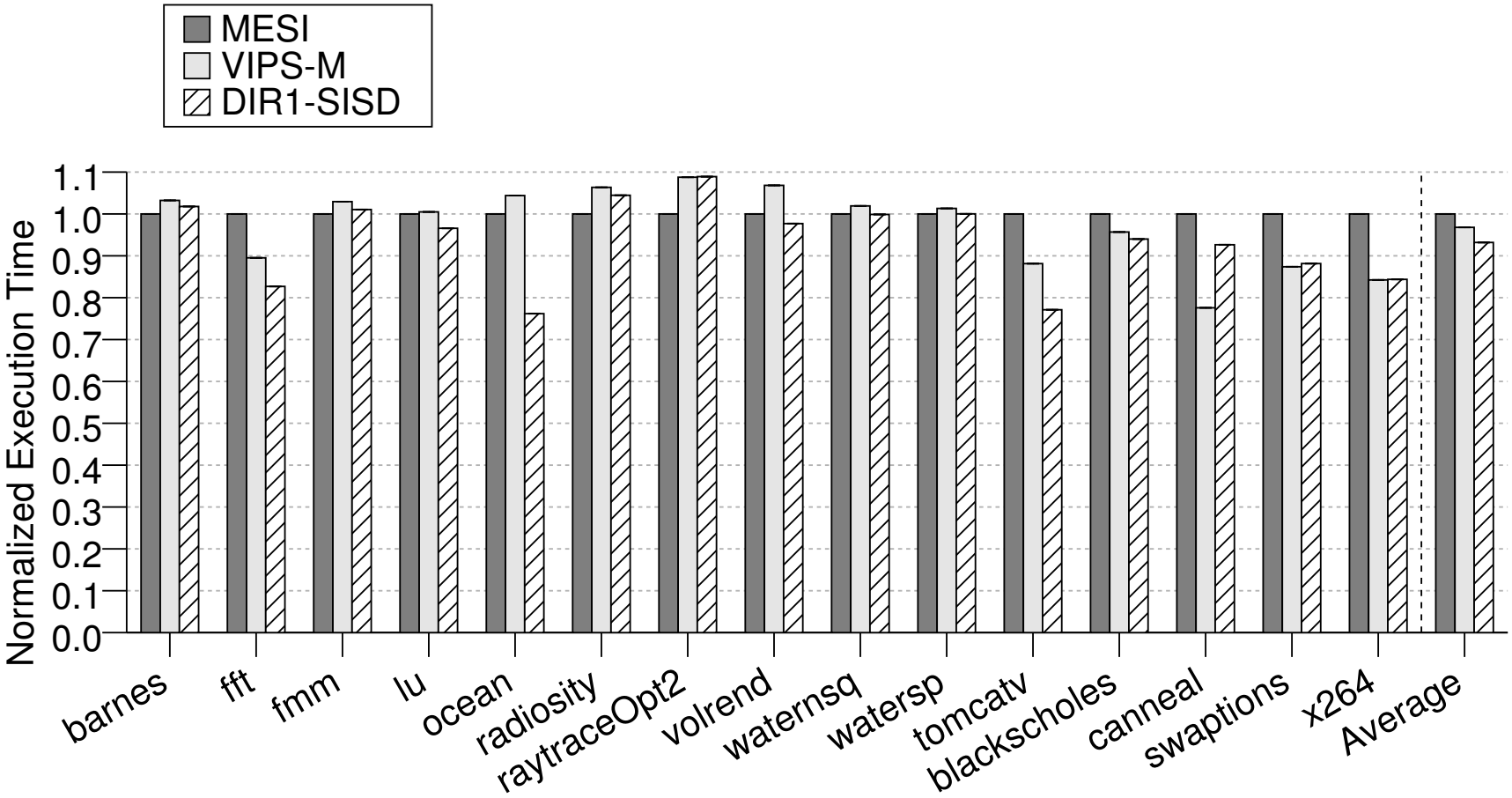
- Potential compression factor of 64 for 4KB pages
- Very simple compression mechanism

Simulation environment

- Simulated a 16-core system with GEMS
  - L1 (private): 32KB 4-way
  - L2 (shared): 512KB 16-way (per bank)
- Benchmarks: SPLASH-2 and PARSEC
- Protocols:
  - MESI directory-based protocol
  - VIPS-M protocol:
    - OS-based classification
    - Backed up in the page table

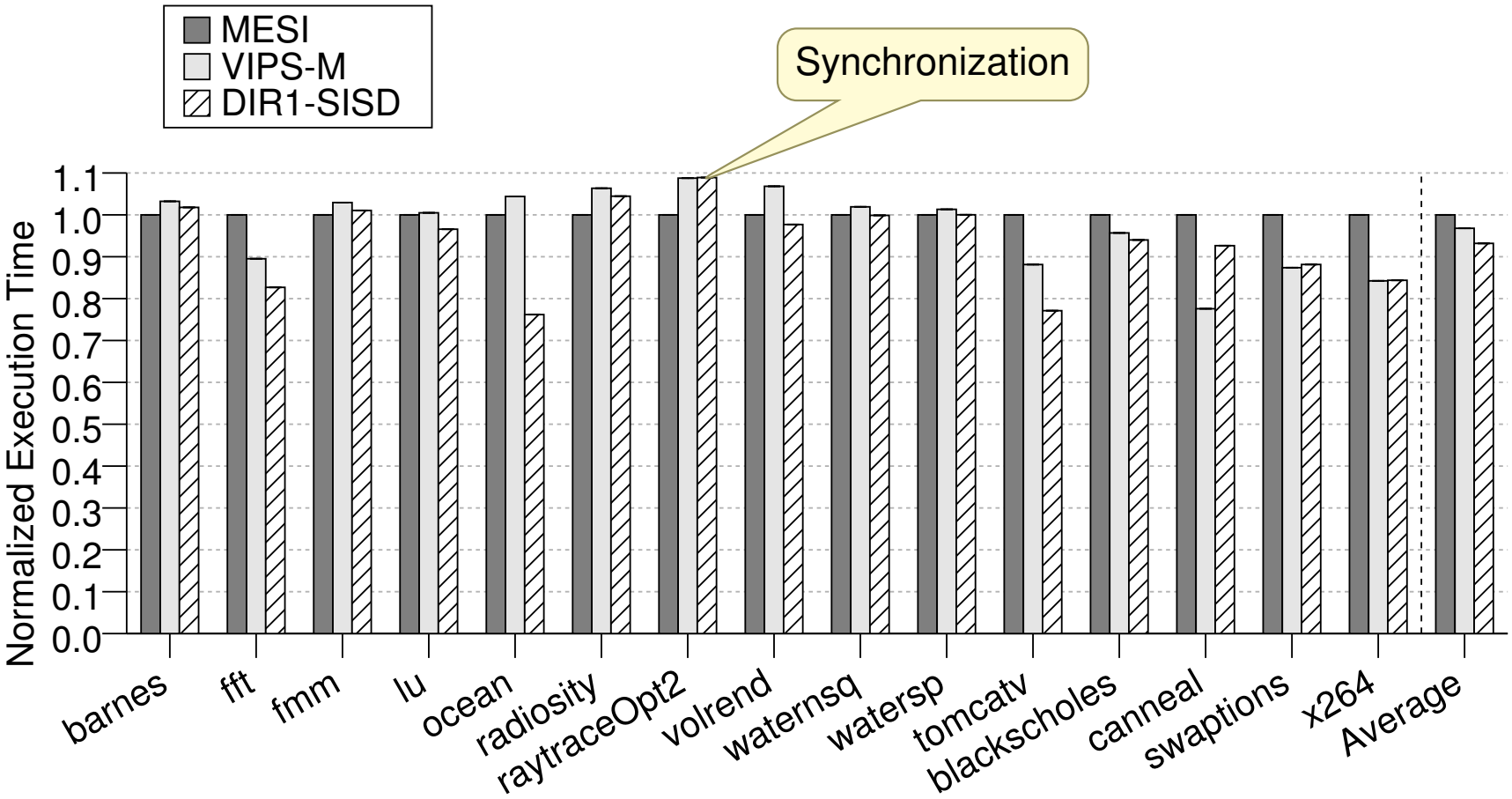
EXECUTION TIME

- Execution time improved by 7% compared to MESI



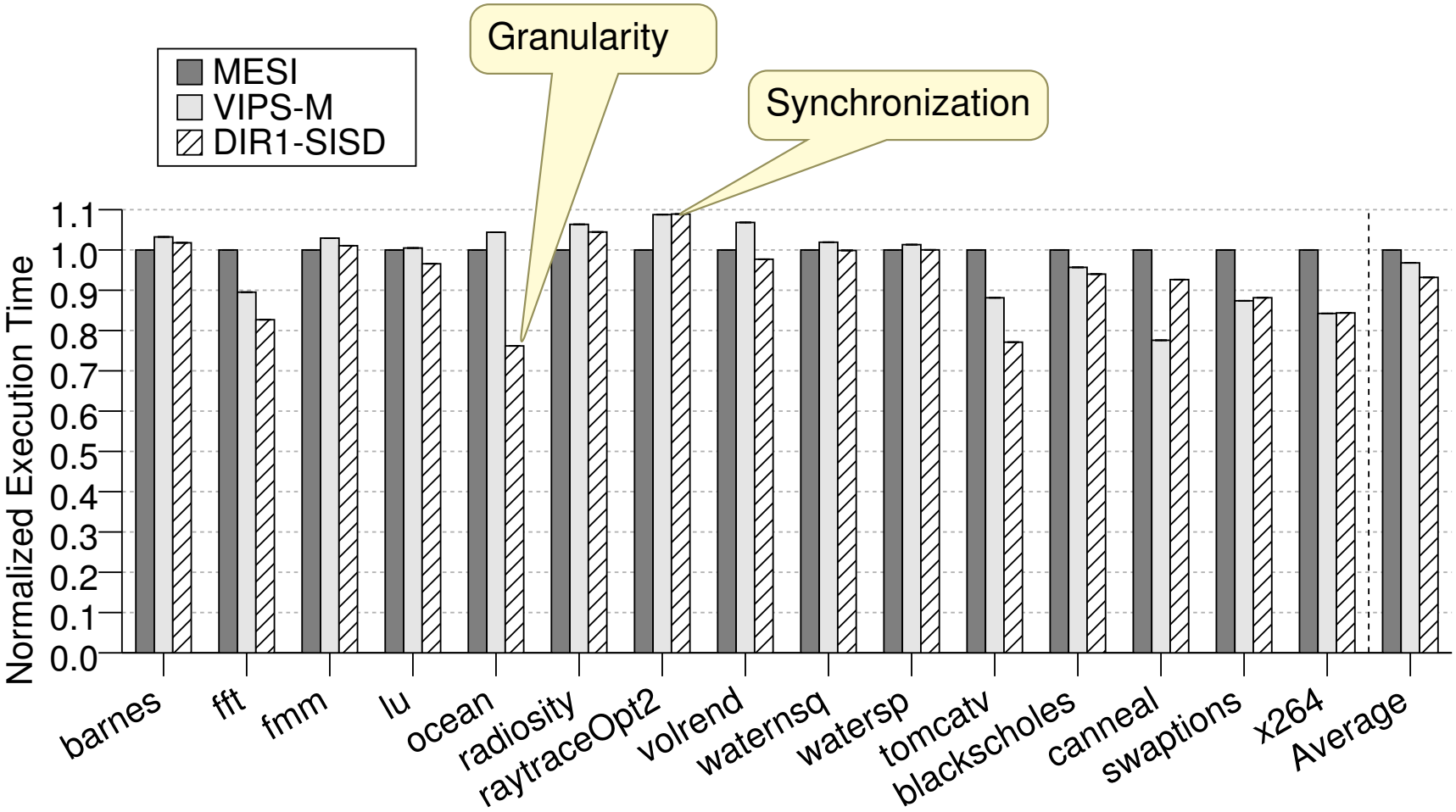
EXECUTION TIME

- Execution time improved by 7% compared to MESI



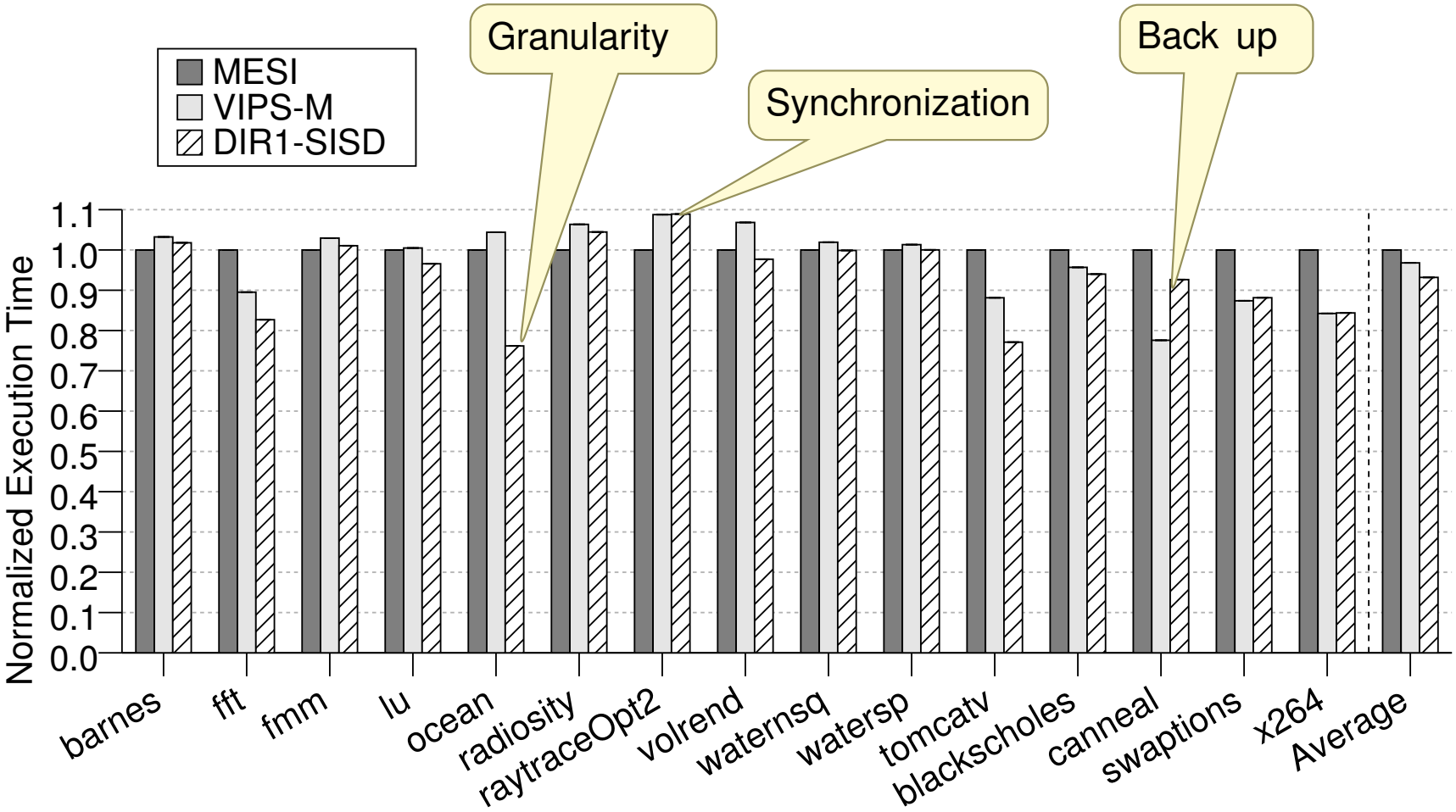
EXECUTION TIME

- Execution time improved by 7% compared to MESI



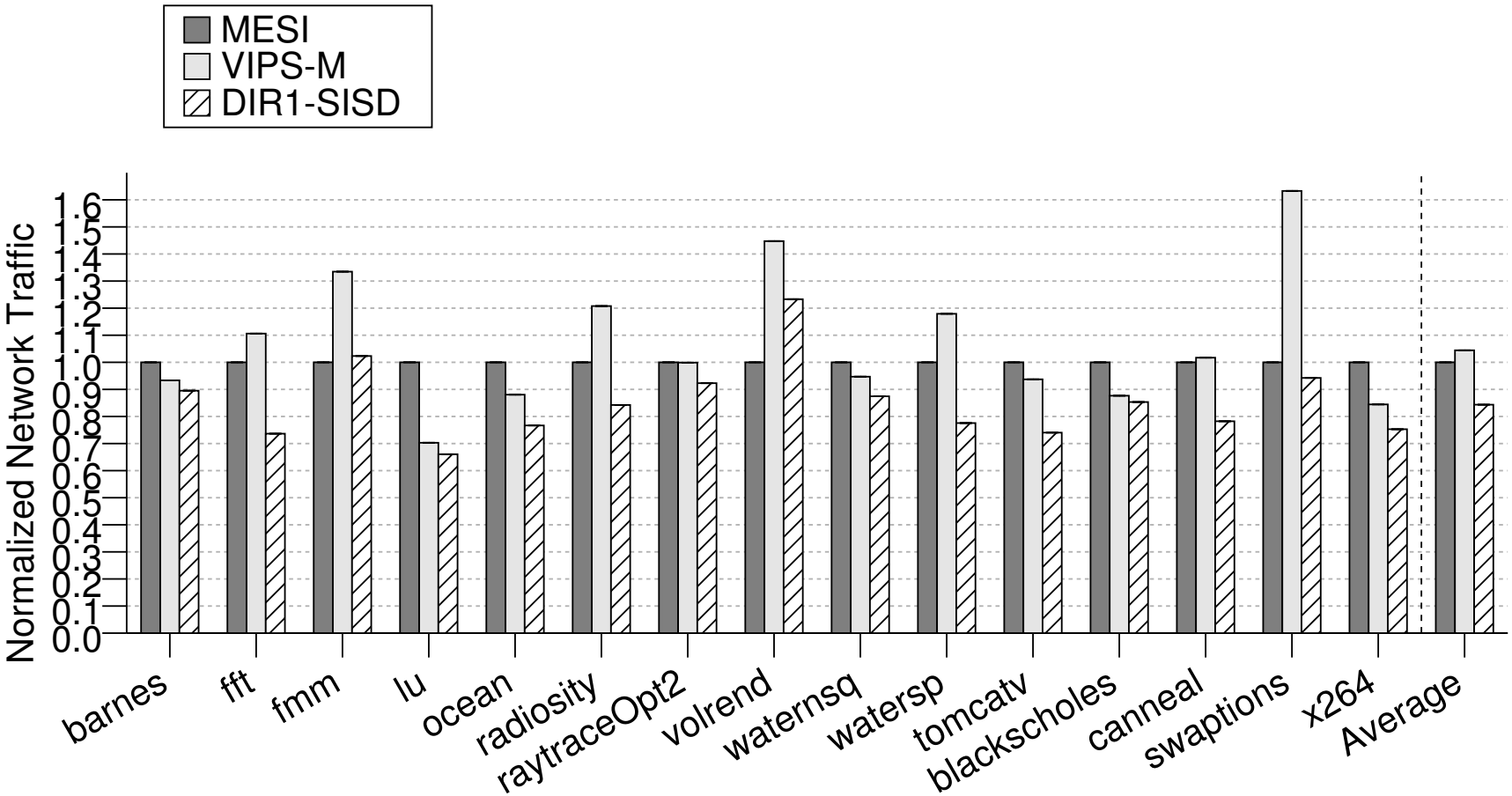
EXECUTION TIME

- Execution time improved by 7% compared to MESI



Execution time

- Network traffic improved by 15% compared to MESI
- 20% compared to VIPS-M (improvements in all benchmarks)
  - More selective self-invalidation; better classification



Conclusions

- A DIR_1 directory for efficient classification
- Self-contained directory
  - No inclusion enforced
  - No back up required
- Single pointer stored
  - But allows wide sharing
  - Without employing broadcasts

Outline

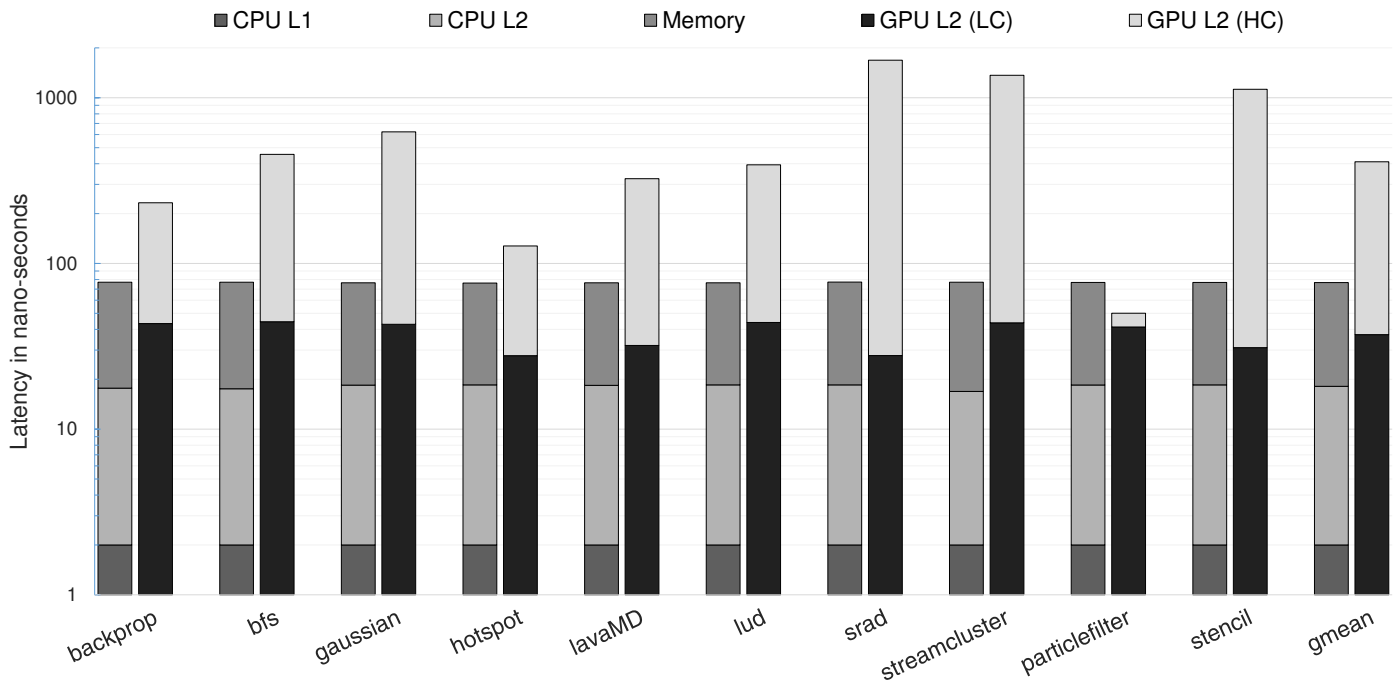
Motivation

- Cache coherence in fused CPU-GPU devices
- Existing solutions use directories and invalidation
  - Heterogeneous MOESI/VI[3]

[3] *Power et al., "A Heterogeneous CPU-GPU Simulator", CAL'15*

MOTIVATION

- Cache coherence in fused CPU-GPU devices
- Existing solutions use directories and invalidation
  - Heterogeneous MOESI/VI[3]
  - Not efficent for the CPU ⇒ Indirection to the GPU



[3] *Power et al., "A Heterogeneous CPU-GPU Simulator", CAL'15*

Motivation

- VIPS-M does not have indirection
- Can VIPS-M provide complexity-effective coherence in fused CPU-GPU devices?

Motivation

- VIPS-M does not have indirection
- Can VIPS-M provide complexity-effective coherence in fused CPU-GPU devices?
  - Yes!
- Key aspects:

Motivation

- VIPS-M does not have indirection
- Can VIPS-M provide complexity-effective coherence in fused CPU-GPU devices?
  - Yes!
- Key aspects:
  1. Heterogeneous race free (HRF) semantics[4]

[4] *Hower et al., "Heterogeneous-race-free Memory Models", ASPLOS'14*

Motivation

- VIPS-M does not have indirection
- Can VIPS-M provide complexity-effective coherence in fused CPU-GPU devices?
  - Yes!
- Key aspects:
  1. Heterogeneous race free (HRF) semantics[4]
  2. Heterogeneous adaptive classification

[4] *Hower et al., "Heterogeneous-race-free Memory Models", ASPLOS'14*

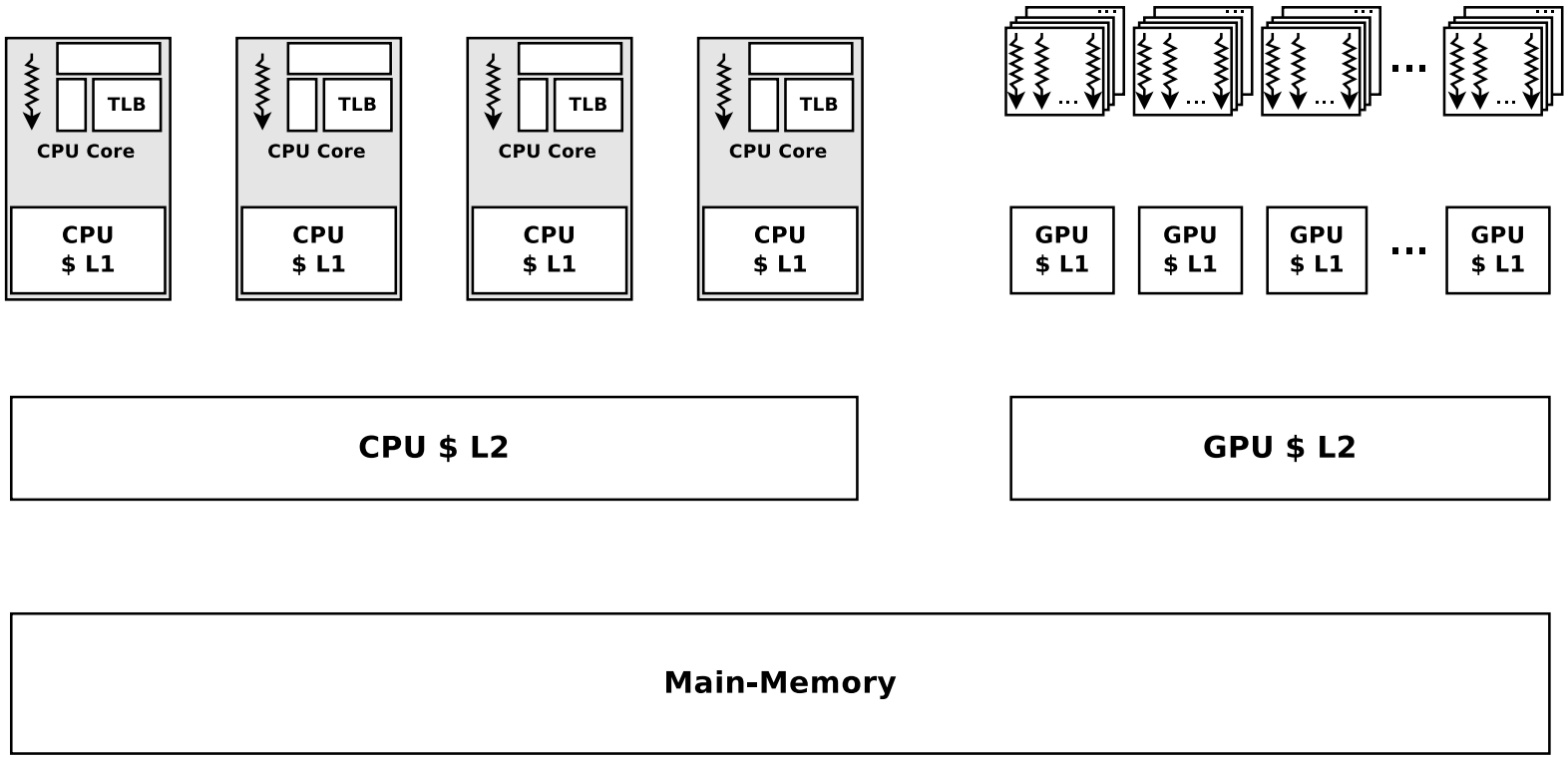
Motivation

- VIPS-M does not have indirection
- Can VIPS-M provide complexity-effective coherence in fused CPU-GPU devices?
  - Yes!
- Key aspects:
  1. Heterogeneous race free (HRF) semantics[4]
  2. Heterogeneous adaptive classification
  3. Virtual cache coherence[5]

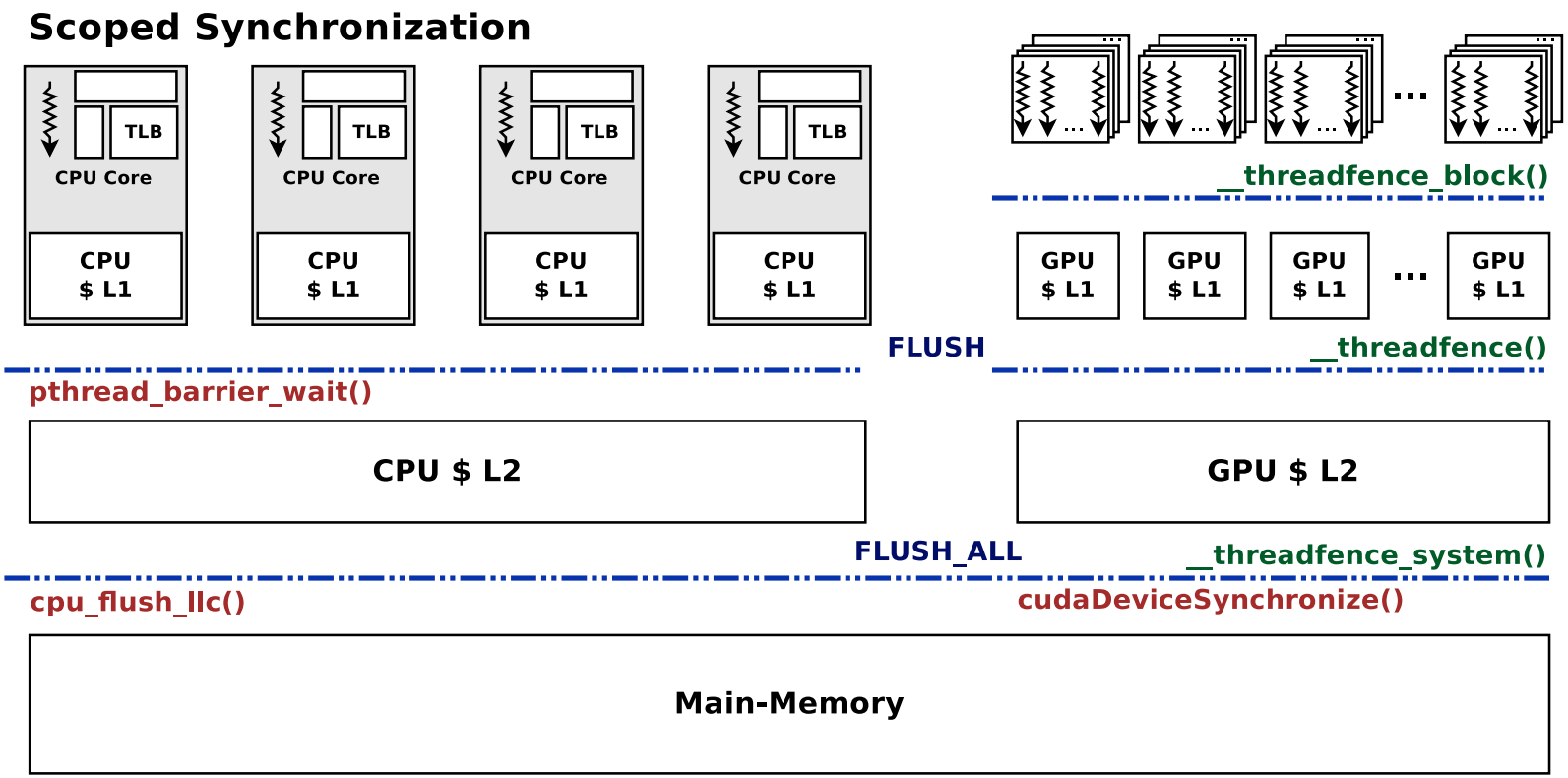
[4] *Hower et al., "Heterogeneous-race-free Memory Models", ASPLOS'14*

[5] *Kaxiras & Ros, "A New Perspective for Efficient Virtual-Cache Coherence", ISCA'13*

HRF AND SCOPED SYNCHRONIZATION



HRF AND SCOPED SYNCHRONIZATION

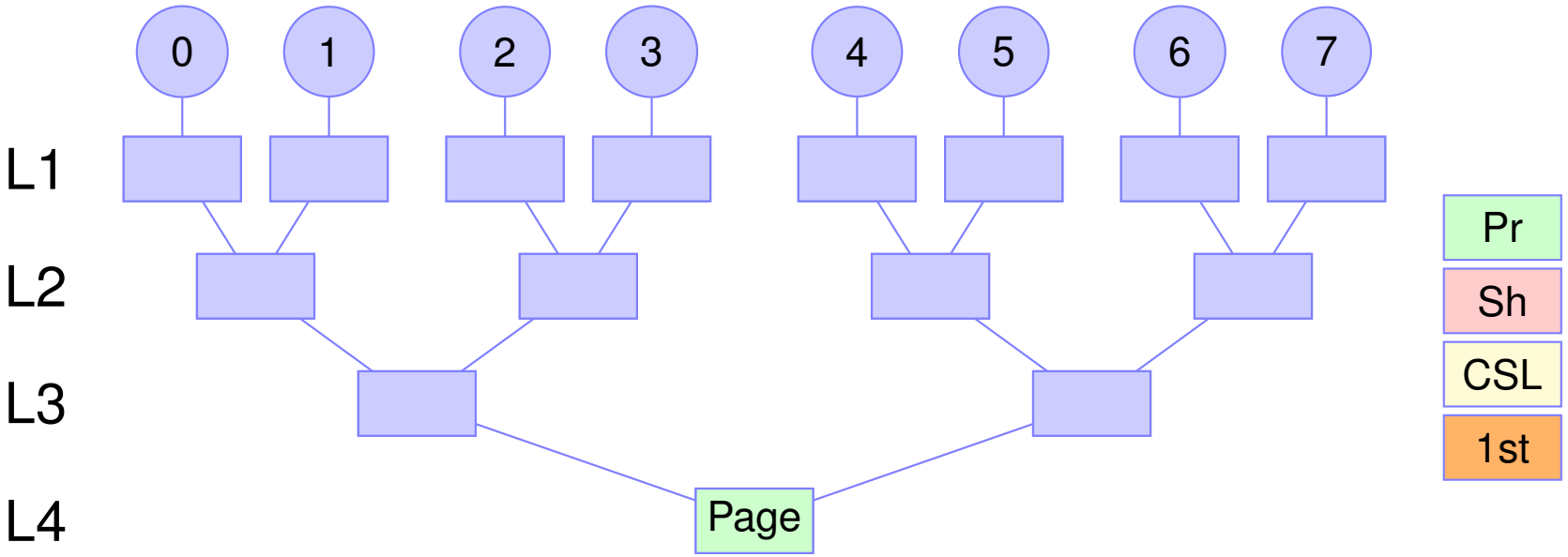


CURRENT SOLUTION FOR CLASSIFICATION: VIPS-H

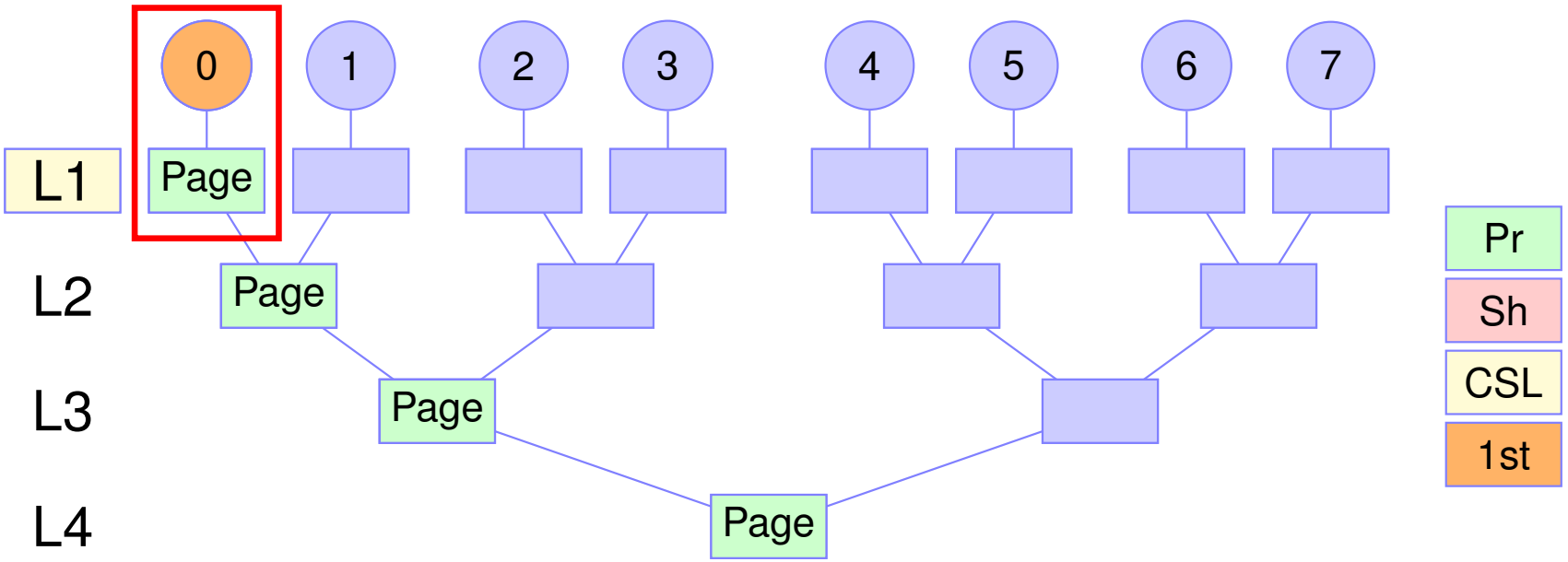
- VIPS-H [6]: Hierarchical P/S classification
  - A block can be shared inside a cluster but be private outside
  - The level where this transition happens is the common sharing level (CSL)
  - Restrict SI/SD to shared blocks within a cluster
- Result
  - The protocol remains simple ⇒ NO hierarchical complexity
  - Hierarchical complexity transferred to classification ⇒ OS
    - So all complexity is transferred to software

[6] *Ros, Davari & Kaxiras, "Hierarchical Private/Shared Classification: the Key to Simple and Efficient Coherence for Clustered Cache Hierarchies", HPCA'15*

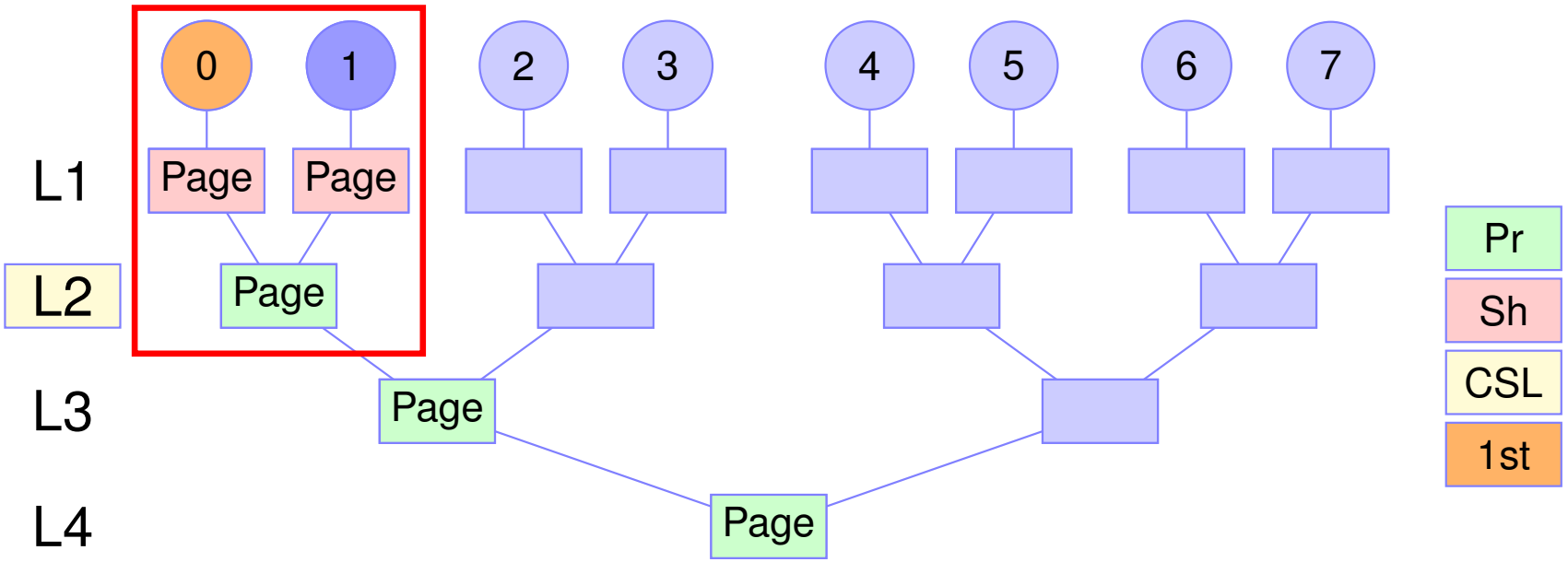
COMMON SHARING LEVEL



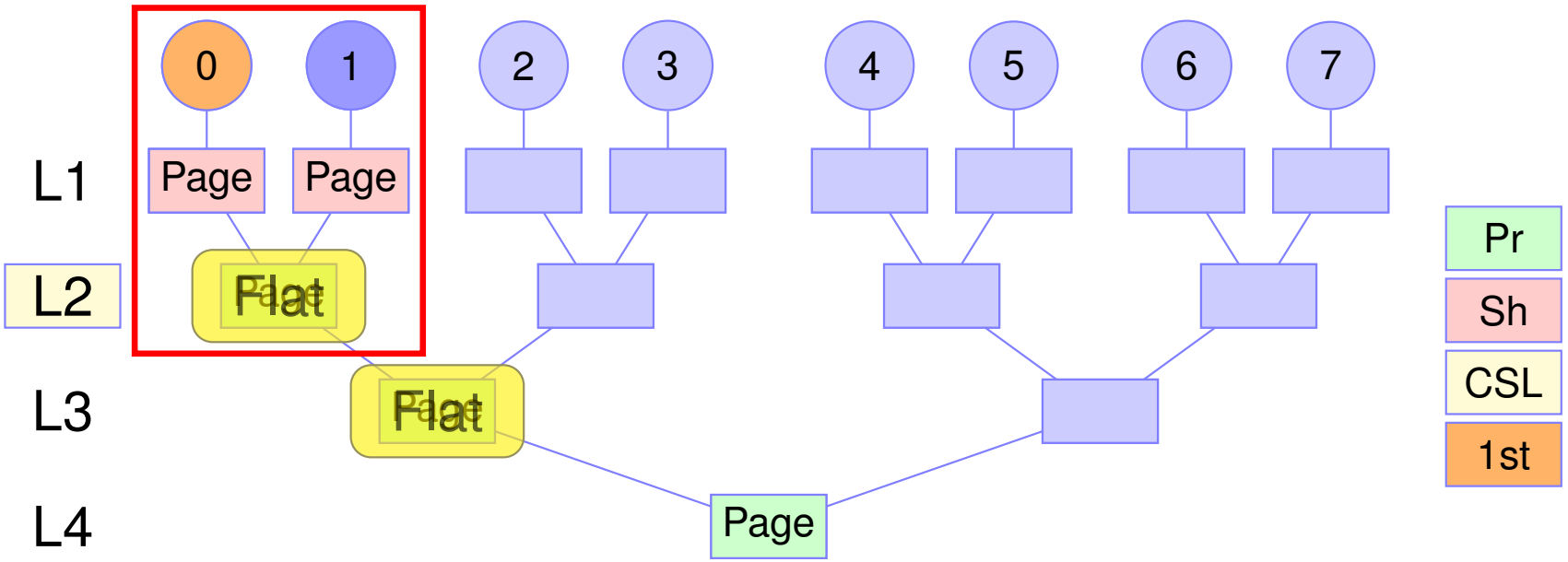
COMMON SHARING LEVEL



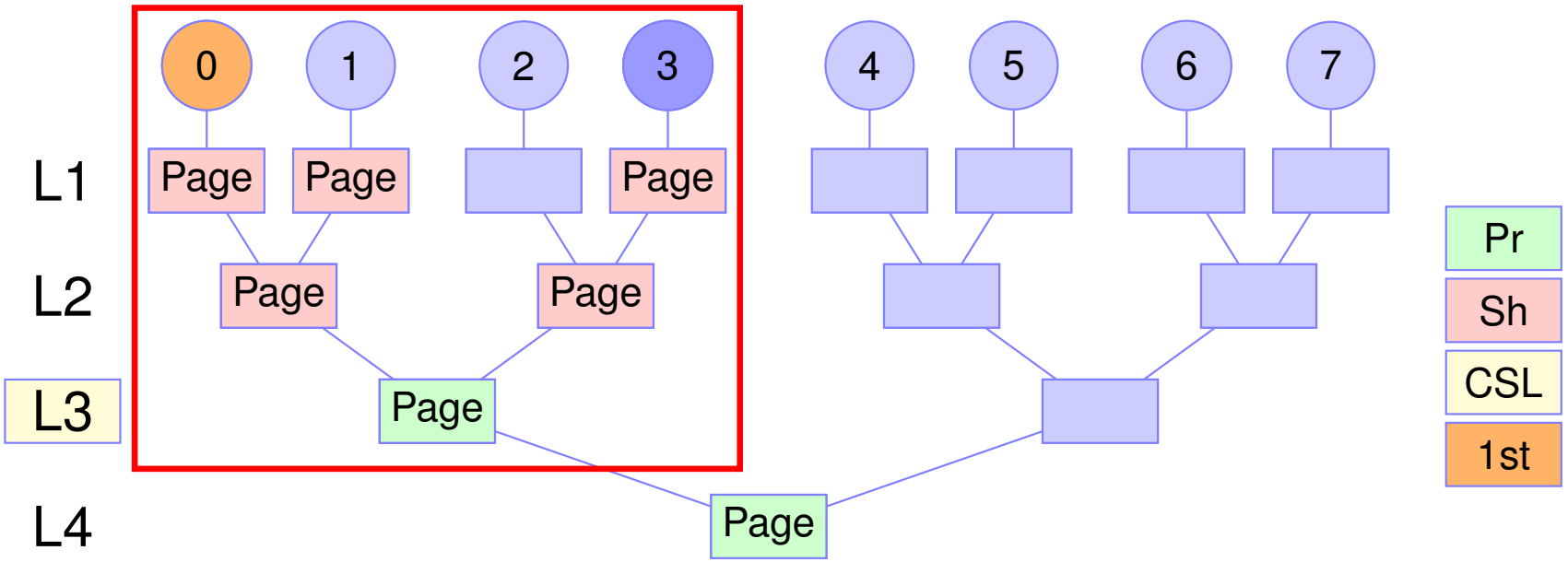
COMMON SHARING LEVEL



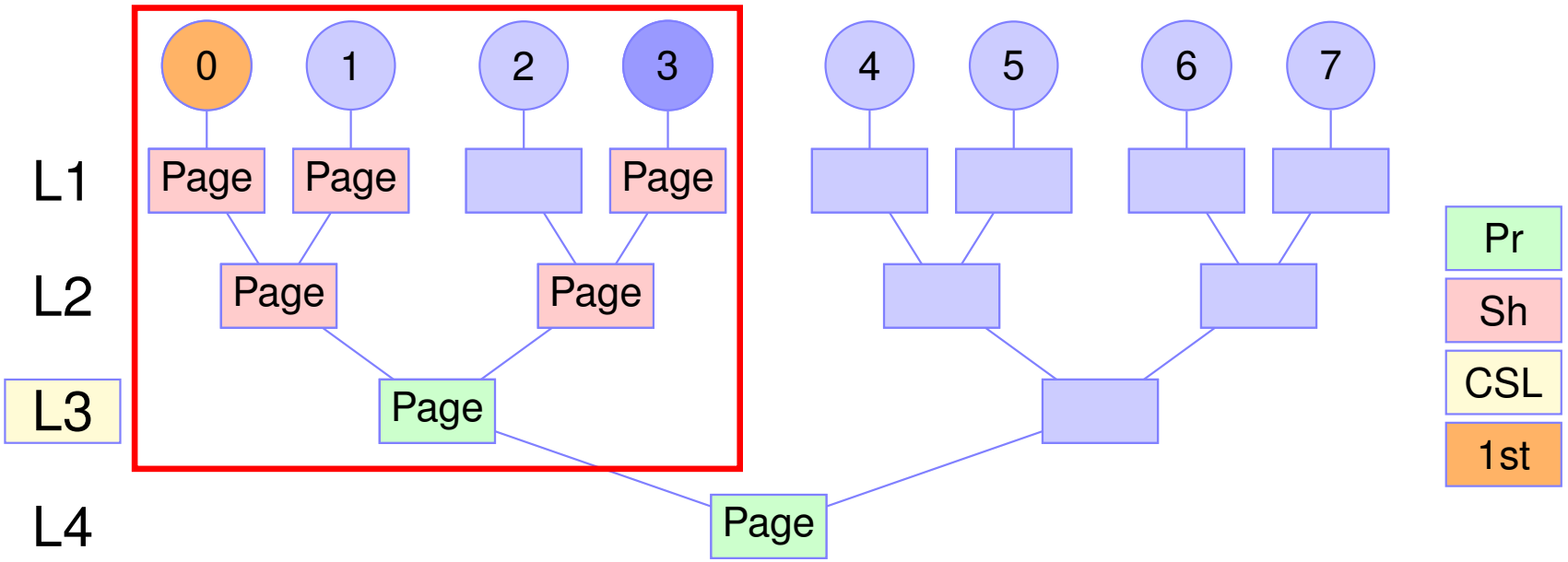
COMMON SHARING LEVEL



COMMON SHARING LEVEL



COMMON SHARING LEVEL



- SI/SD only for blocks in shared pages
- SD propagated until the CSL

Complexity, cost, and area

- H-MOESI: Hierarchical full-map
- VIPS-H: P/S bit

Number of states and extra bits required (16x4)

	H-MOESI			VIPS-H		
Controller	States Tot./Base	Bitmap bits	Total bits	States Tot./Base	P/S bit	Total bits
L1 cache	16 / 5	0	3	9 / 3	1	3
L2 cache	59 / 13	16	20	5 / 3	1	3
L3 cache	13 / 4	4	6	4 / 3	1	3
Total cost	844KB			204KB		

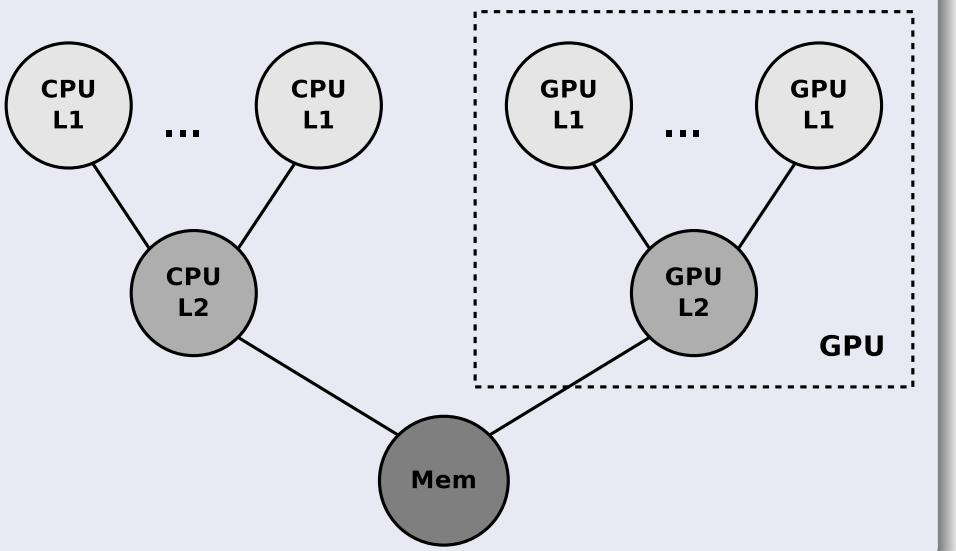
- 76% memory reduction compared to H-MOESI

GPU INTERRUPTS

- This approach is not appropriate for fused CPU-GPU devices.
  - Interrupting all GPU cores on a CSL change?
- Solution:
  - GPUs are throughput oriented
    - → Naïve classification for the GPU
  - CPUs are latency oriented
    - → Efficient, adaptive classification for the CPU

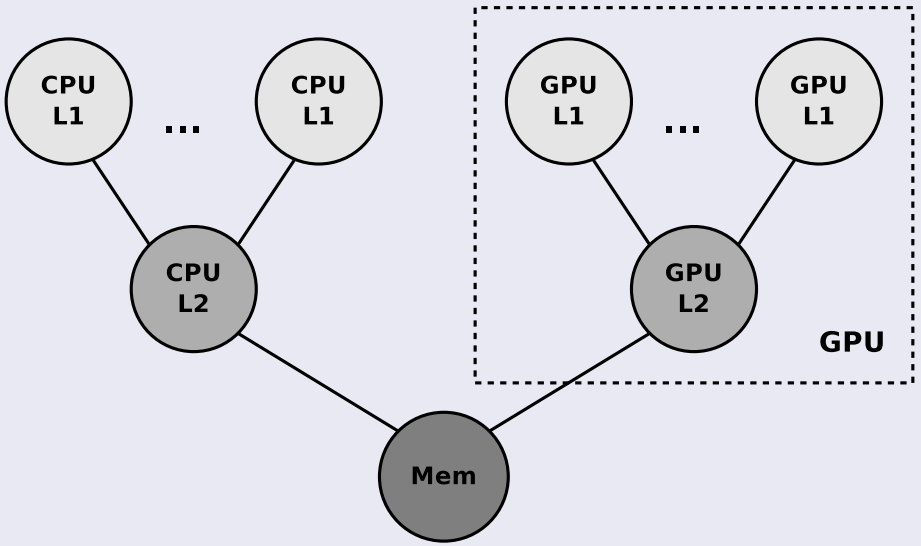
CLASSIFICATION FOR FUSED DEVICES

CLASSIFICATION IN HOMOGENEOUS SYSTEMS

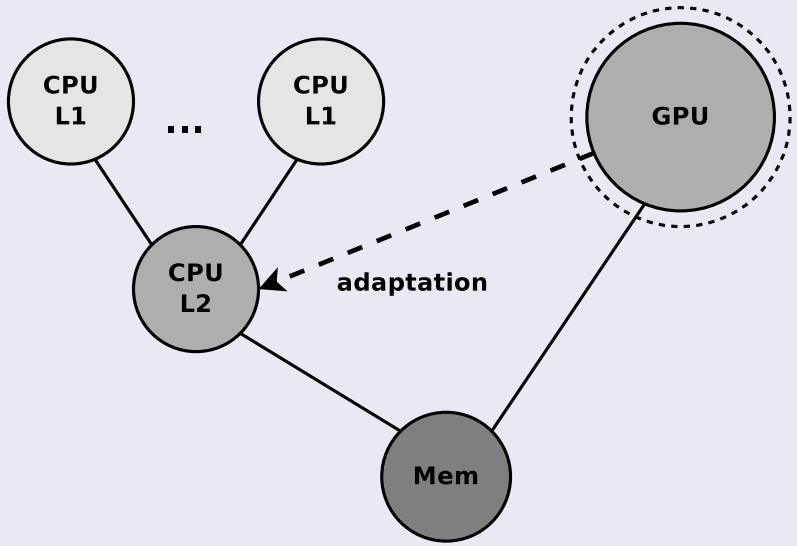


CLASSIFICATION FOR FUSED DEVICES

CLASSIFICATION IN HOMOGENEOUS SYSTEMS



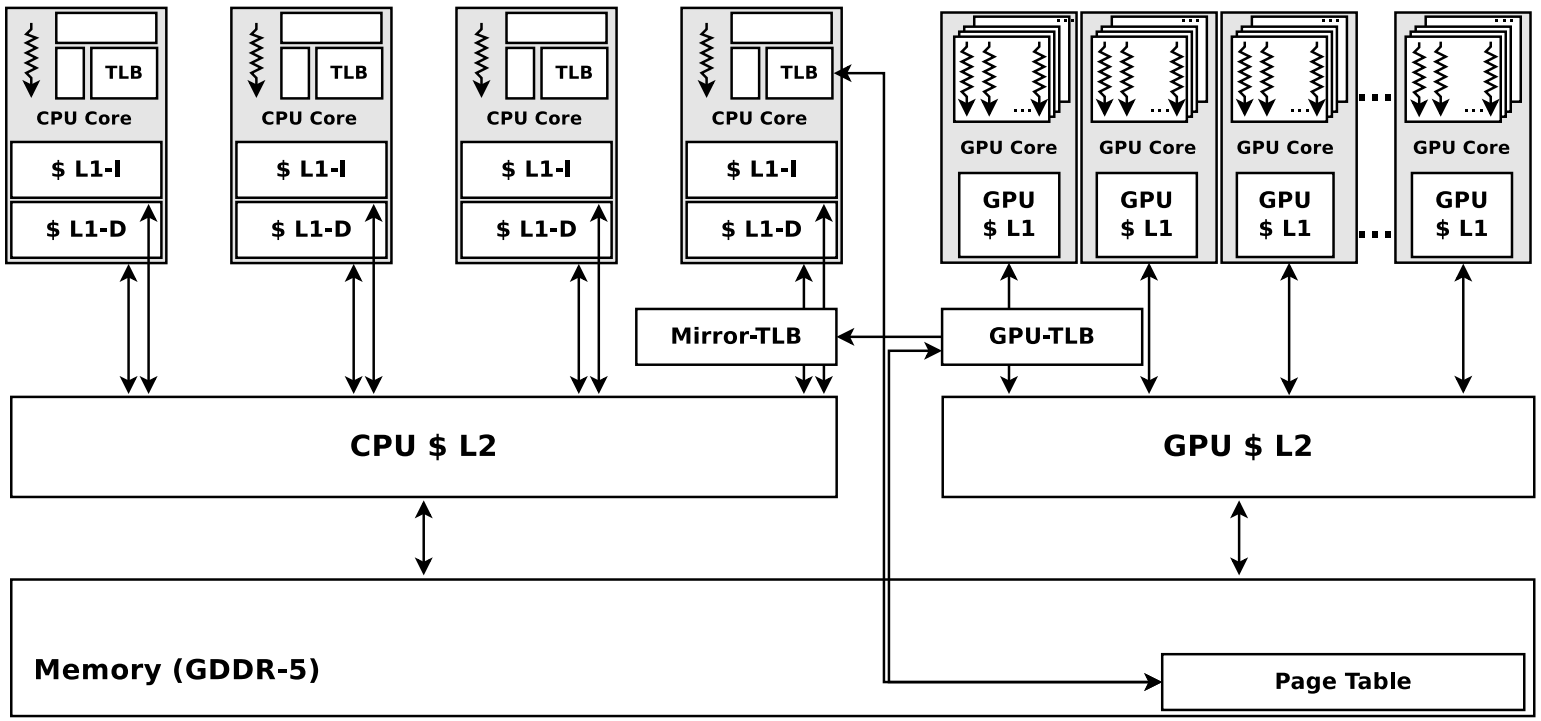
CLASSIFICATION IN HETEROGENEOUS SYSTEMS



- Inside the GPU blocks in L1 are considered always shared
  - No classification: Low synchronization / Low reuse
- L2 CPU with adaptive classification for efficiency
  - Critical for performance

Adaptation with a mirror TLB

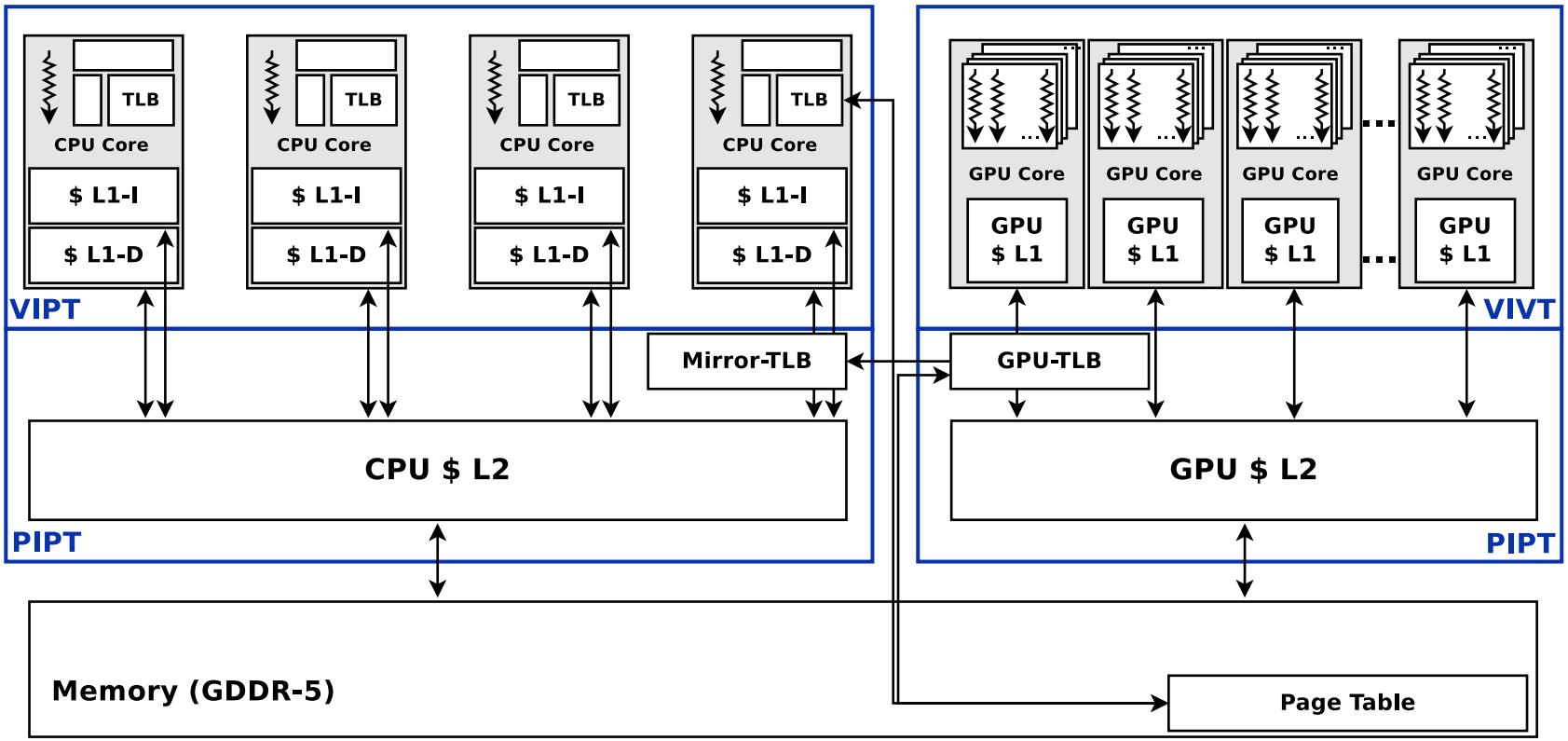
- A mirror TLB shows in the L2 CPU what is shared by the GPU
- The mirror TLB is updated along with the GPU TLB
  - Provides adaptation in the L2 CPU classification



Virtual Cache Coherence

- Requirements for simple virtual-cache coherence without reverse translation
  1. No indirection (unexpected messages) from the physical to the virtual domain
  2. DRF applications with synchronization exposed
- The proposed approach fulfill these requirements

Virtual Cache Coherence

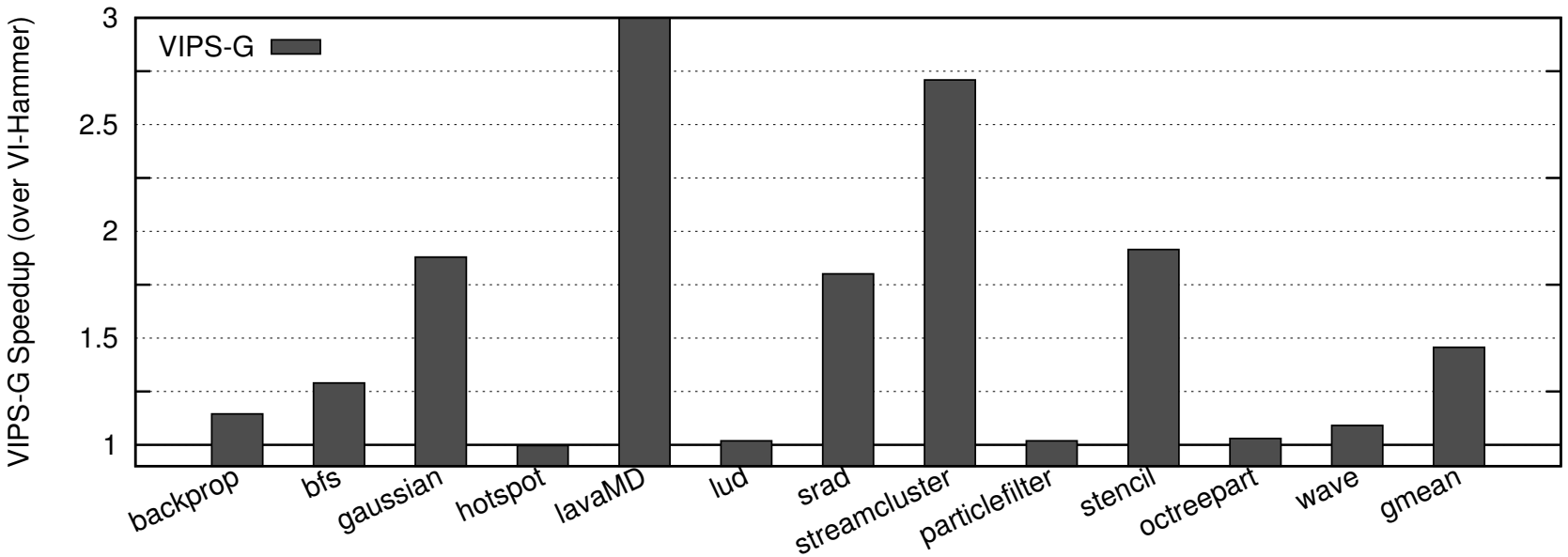


Simulation environment

- Simulated a 16-SM GPU with Gem5-GPU
- CPU
  - L1 32KB 8-way
  - L2 4MB 64-way (8 banks)
- GPU
  - L1 64KB 8-way
  - L2 2MB 32-way (16 banks)
- Benchmarks: Rodinia +3 more ported applications
- Protocols: MOESI/VI (VI_Hammer) and VIPS-G

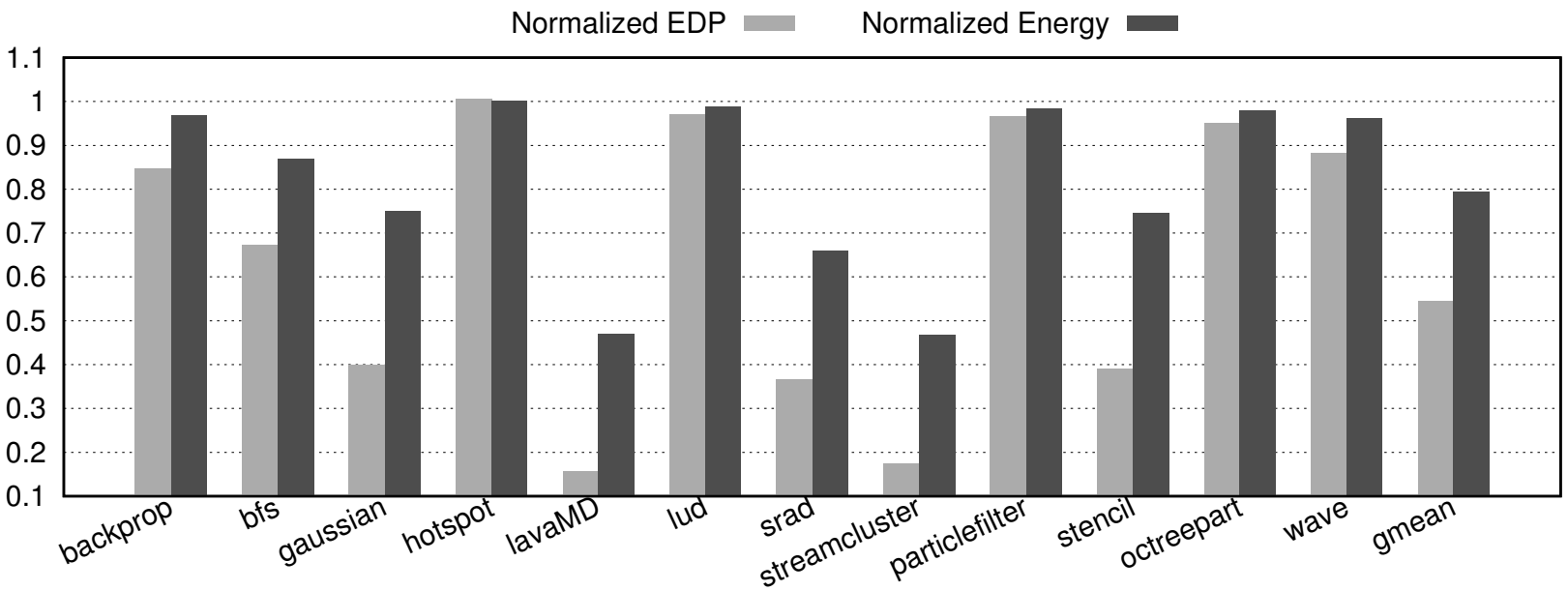
SPEEDUP

- 45% speedup over VI_HAMMER



EDP AND ENERGY

- EDP reduction of 45% compared to VI_HAMMER
- ≈20% on average lower energy compared to VI_HAMMER



Conclusions

- VIPS-G complexity-effective coherence for fused CPU-GPU devices
  - Heterogeneous race free model
  - Heterogeneous adaptive PS classification
  - Support for virtual cache coherence

Private/Shared Classification in Complexity-Effective Coherence Protocols

Alberto Ros[1] Stefanos Kaxiras[2]
Mahdad Davari[2] Konstantinos Koukos[2] Erik Hagersten[2]

[1]Universidad de Murcia
aros@ditec.um.es

[2]Uppsala University

Oct 22, 2015