ADVANCED PARALLEL COMPUTING LECTURE 08 - ADVANCED COHERENCE TECHNIQUES

Holger Fröning
holger.froening@ziti.uni-heidelberg.de
Institute of Computer Engineering
Ruprecht-Karls University of Heidelberg

Some material by Falsafi, Hardavellas, Nowatzyk of EPFL, Northwestern, CMU

SCALABLE CACHE COHERENCE EXAMPLES

- Replace bus-based, snooping-based protocols
  - Topology bandwidth
  - Snooping bandwidth
- Examples for more complex coherence protocols
  - AMD's Probe Filter
  - COMA
  - Token Coherence

AMD’S PROBE FILTER “HT ASSIST”

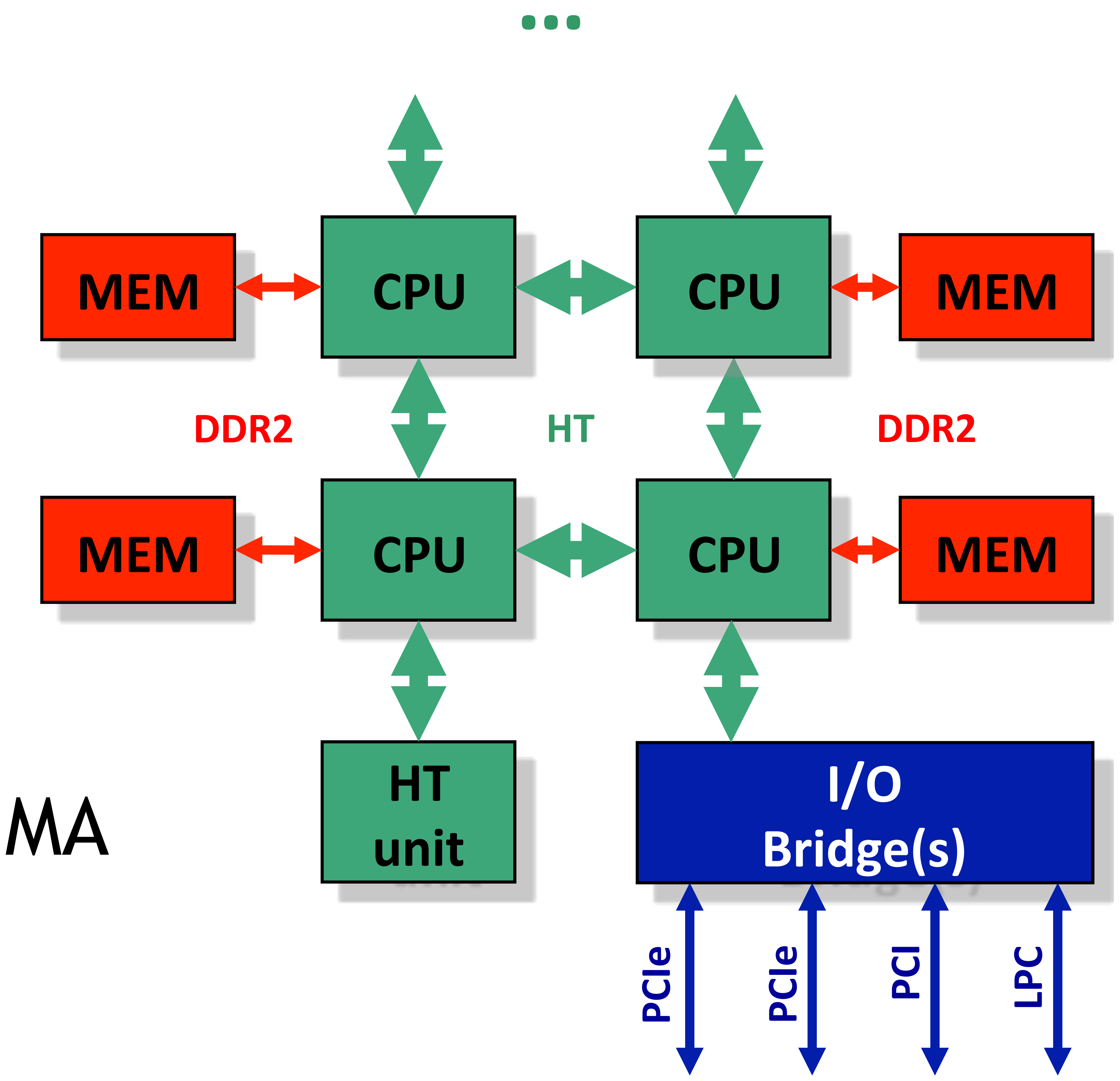
GENERAL ARCHITECTURE

K10/Bulldozer architecture

Variable topology

- cHT links
- Node degree = 3

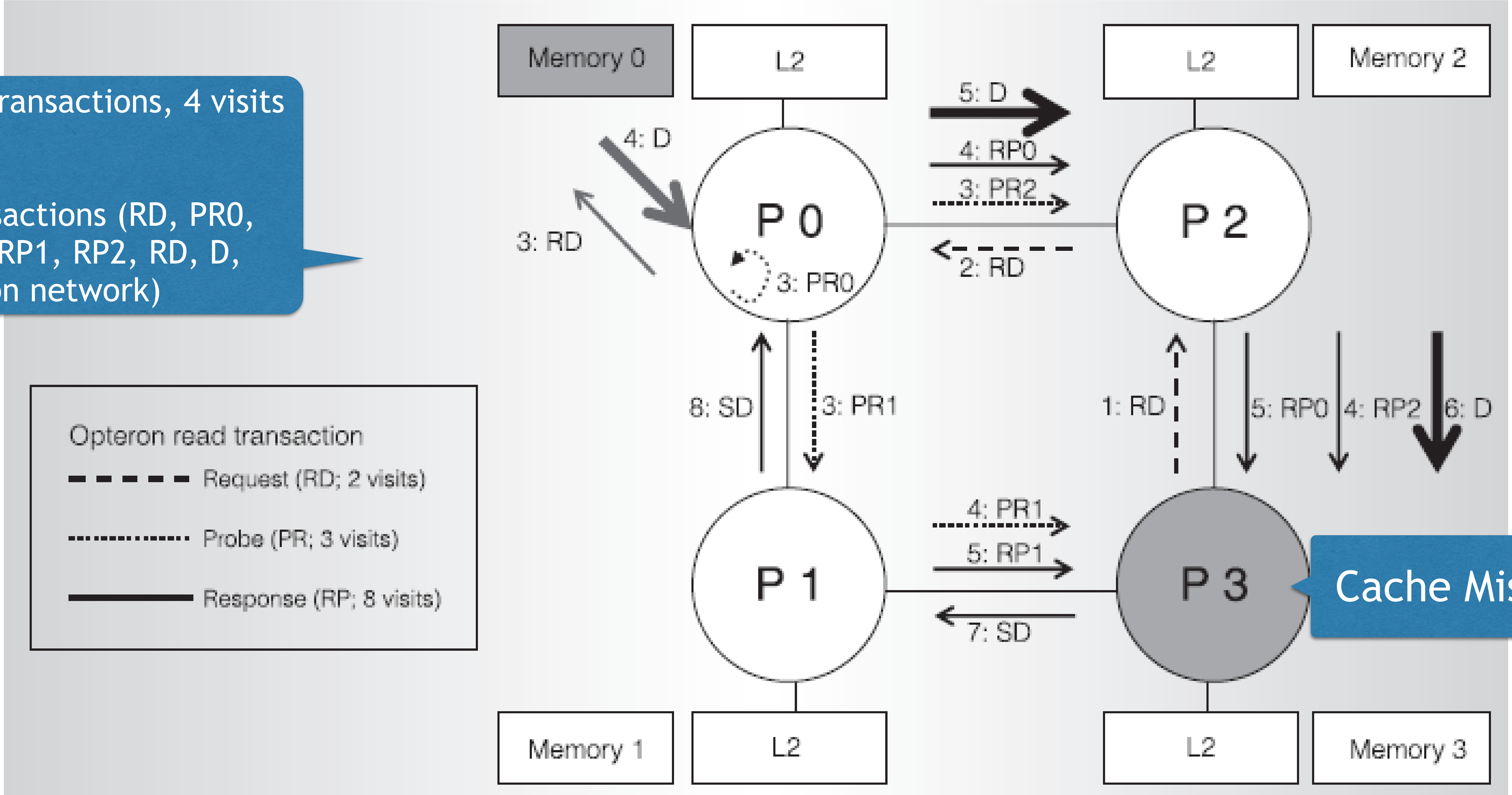
Classification according to this lecture: NUMA



COHERENCE PROTOCOL

Mandatory: 2 transactions, 4 visits total

Result: 10 transactions (RD, PR0, PR1, PR2, RP0, RP1, RP2, RD, D, SD), 13 visits (on network)



Cache Miss

Pat Conway and Bill Hughes. 2007. The AMD Opteron Northbridge Architecture. IEEE Micro 27, 2 (March 2007), 10-21.

PROBE FILTERING

Minimize probe count

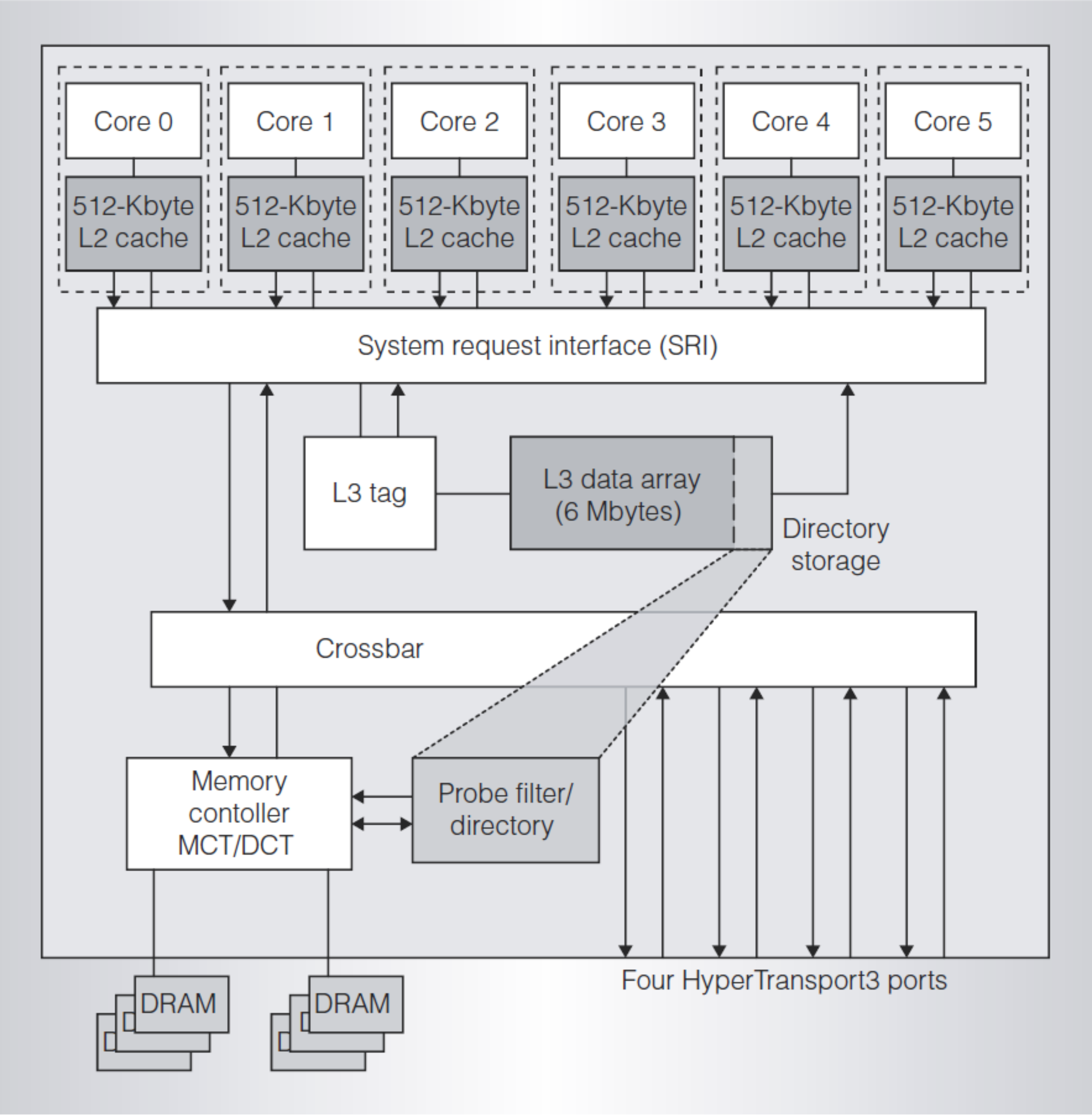
- Options for filter placement?

Use part of L3 for directory

- Not used for 1P systems
- No need to implement a dedicated directory

Inclusive: each CL must be listed

- Size is 1MB
- Track CLs that are in states {M,O,E,S}
- I: no probe filter entry -> uncached



Pat Conway, Nathan Kalyanasundharam, Gregg Donley, Kevin Lepak, and Bill Hughes. 2010. Cache Hierarchy and Memory Subsystem of the AMD Opteron Processor. IEEE Micro 30, 2 (March 2010), 16-29.

PROBE FILTERING

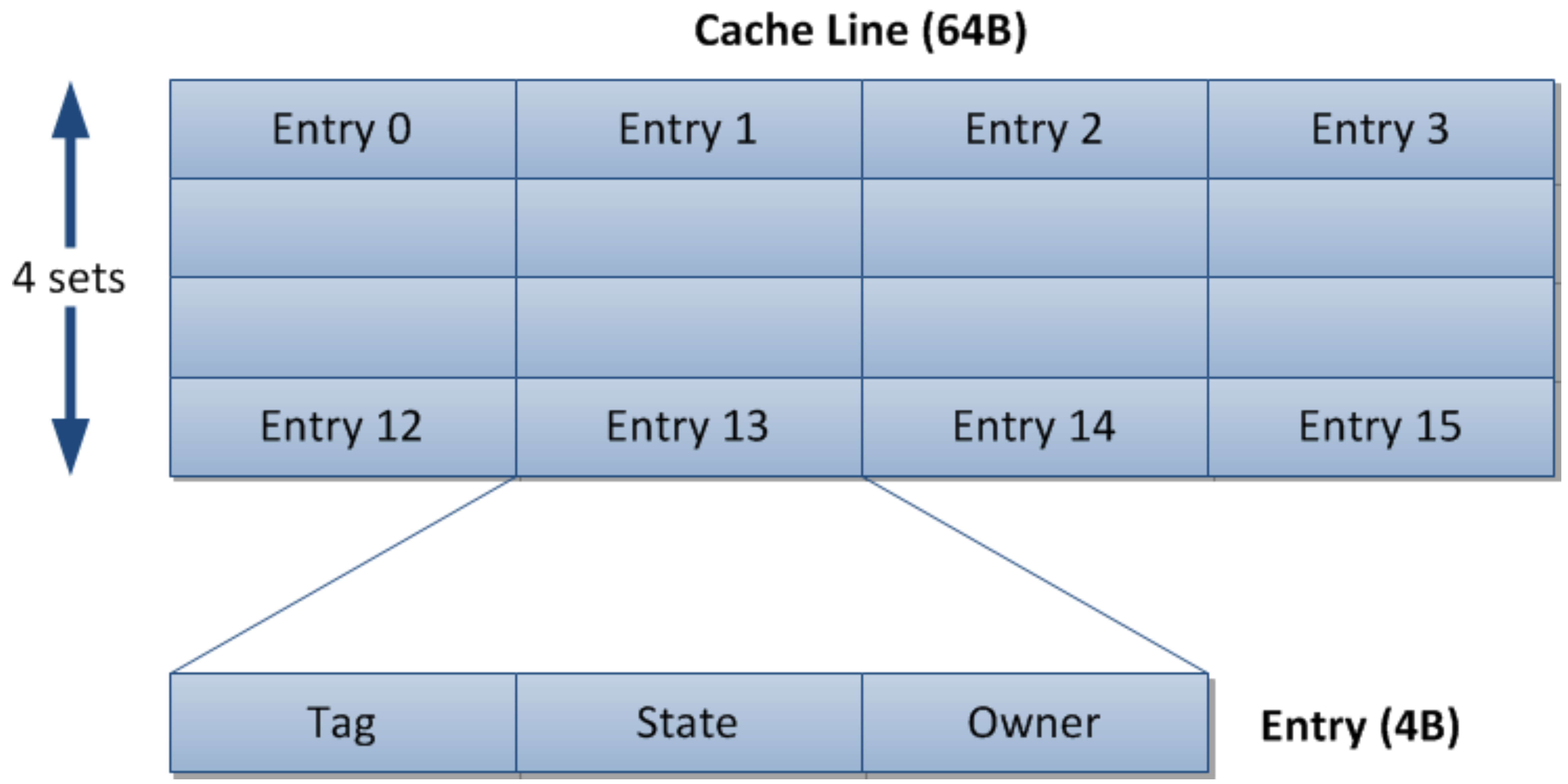
L3 cache line: 64B

Each probe filter entry: 4B, 4-way set-associative

=> 256k PF entries (1MB)

=> 16MB Cache max (no associativity conflicts)

=> 4MB Cache min (max. associativity conflicts)



Index: 256k entries/4 sets = 64k = 16b
Tag: 32b - 3b(state) - 3b(owner) = 26b

48b (PA) – 16b (index) – 6b (CL granularity) = 26b (tag)

PROBE FILTERING

Probe Filter States

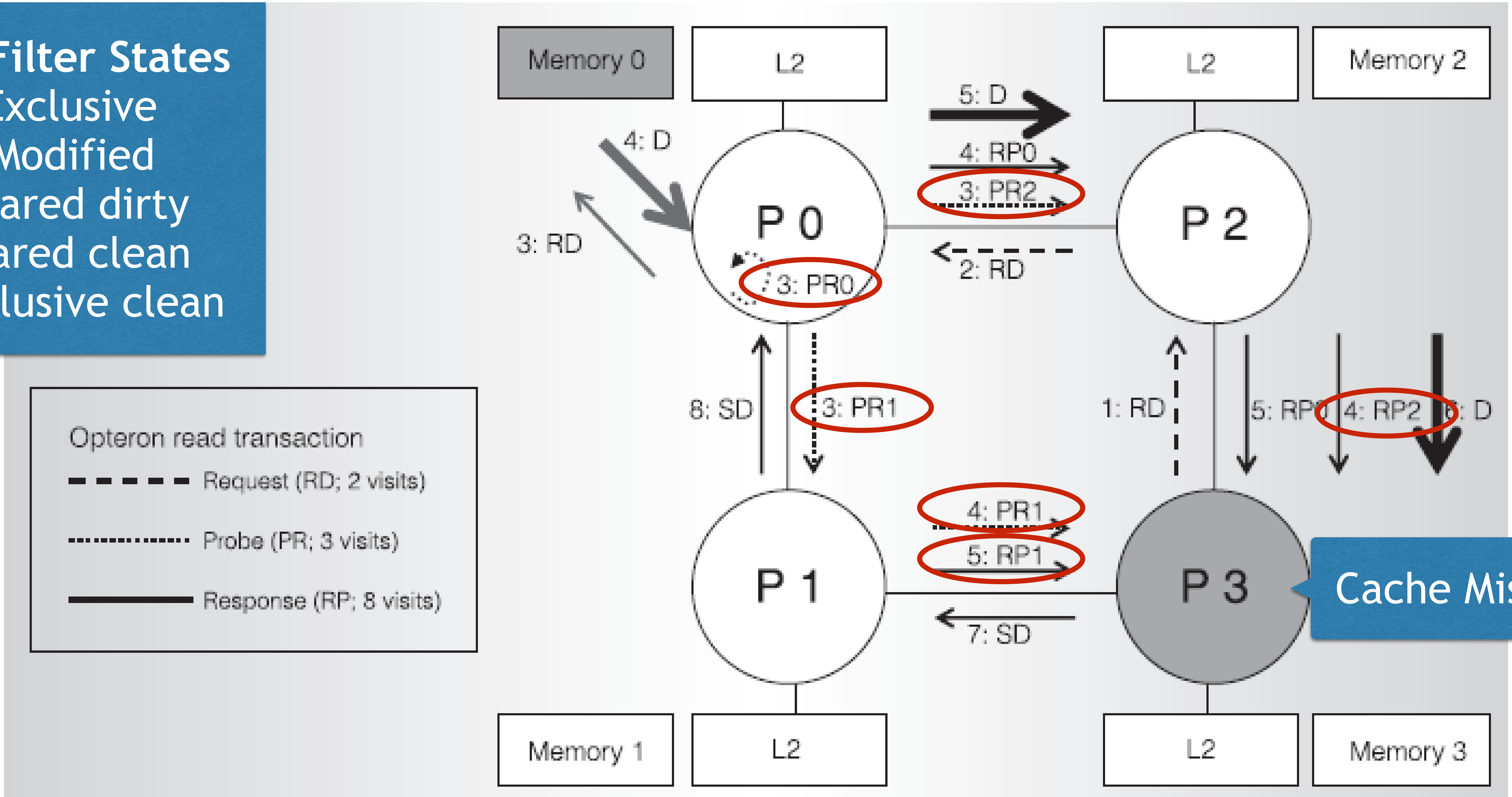
- E: exclusive (explicit eviction notification)
- M: modified
- O: shared dirty
- S: shared clean
- S1: exclusive clean

Three probing scenarios

- No probe required
- Directed probe
- Broadcast probe

PROBE FILTERING

Probe Filter States
E: Exclusive
M: Modified
O: Shared dirty
S: Shared clean
S1: Exclusive clean



Cache Miss

Pat Conway and Bill Hughes. 2007. The AMD Opteron Northbridge Architecture. IEEE Micro 27, 2 (March 2007), 10-21.

PROBE FILTERING

Performance benefits highly application-dependent

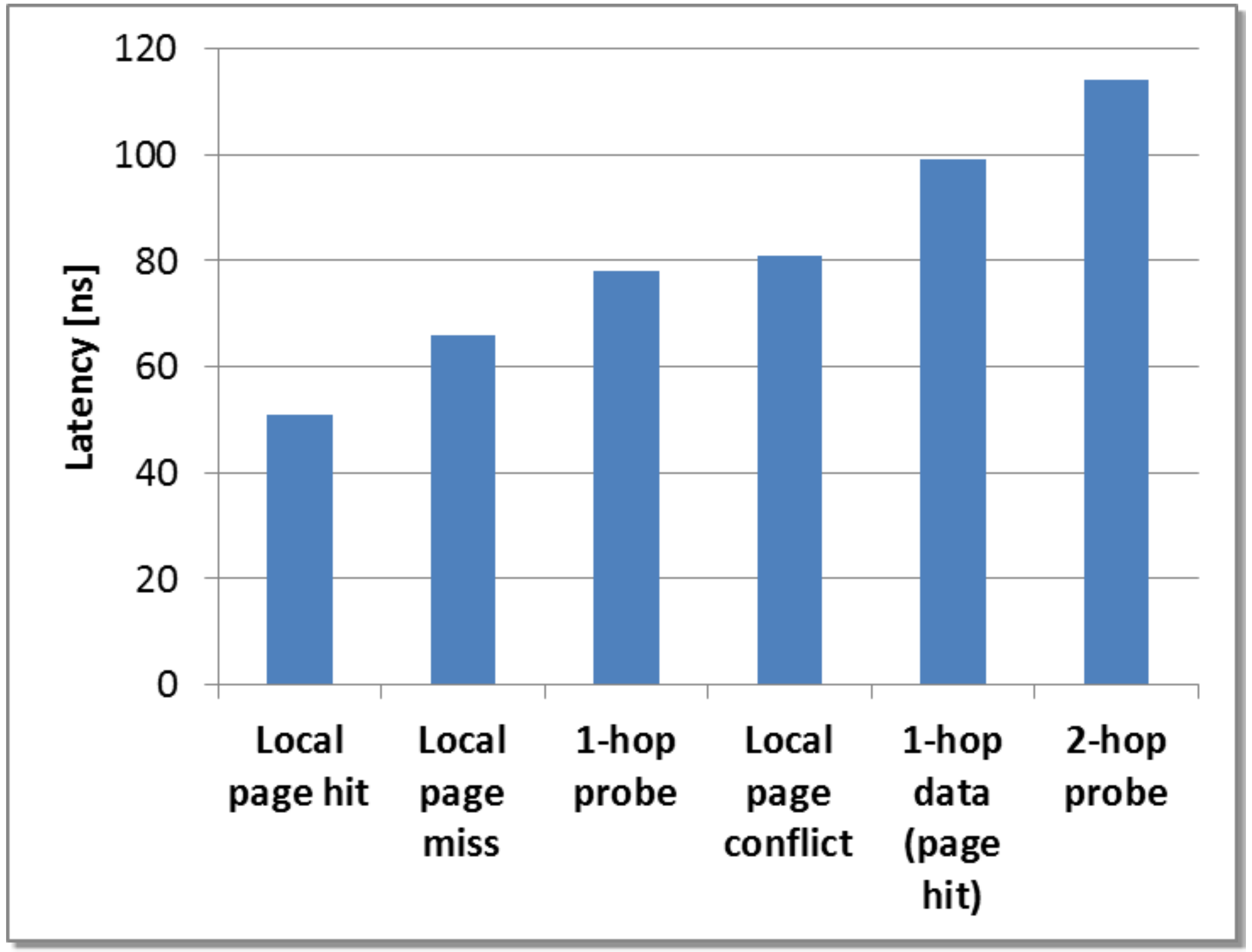
AMD Istanbul: 2.4GHz core, 2.2GHz NB, DDR2-800, HT2400

Local DRAM request

- 51ns (page hit)
- 66ns (page miss)
- 81ns (page conflict)

Remote

- 1-hop probe: 78ns
- 1-hop data: 99ns (page hit)
- 2-hop probe: 114ns



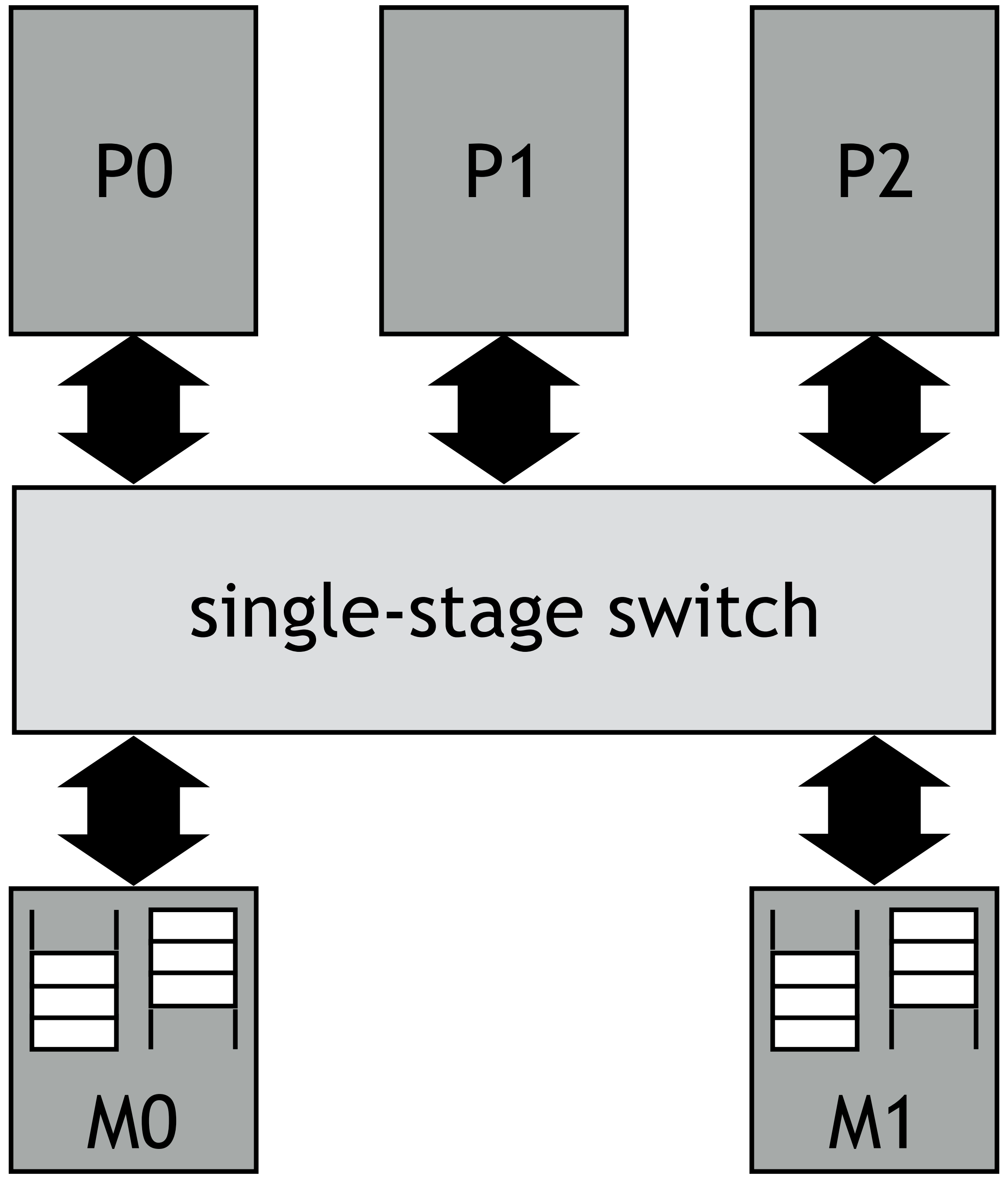
COMA: CACHE-ONLY MEMORY ARCHITECTURE

MOTIVATION

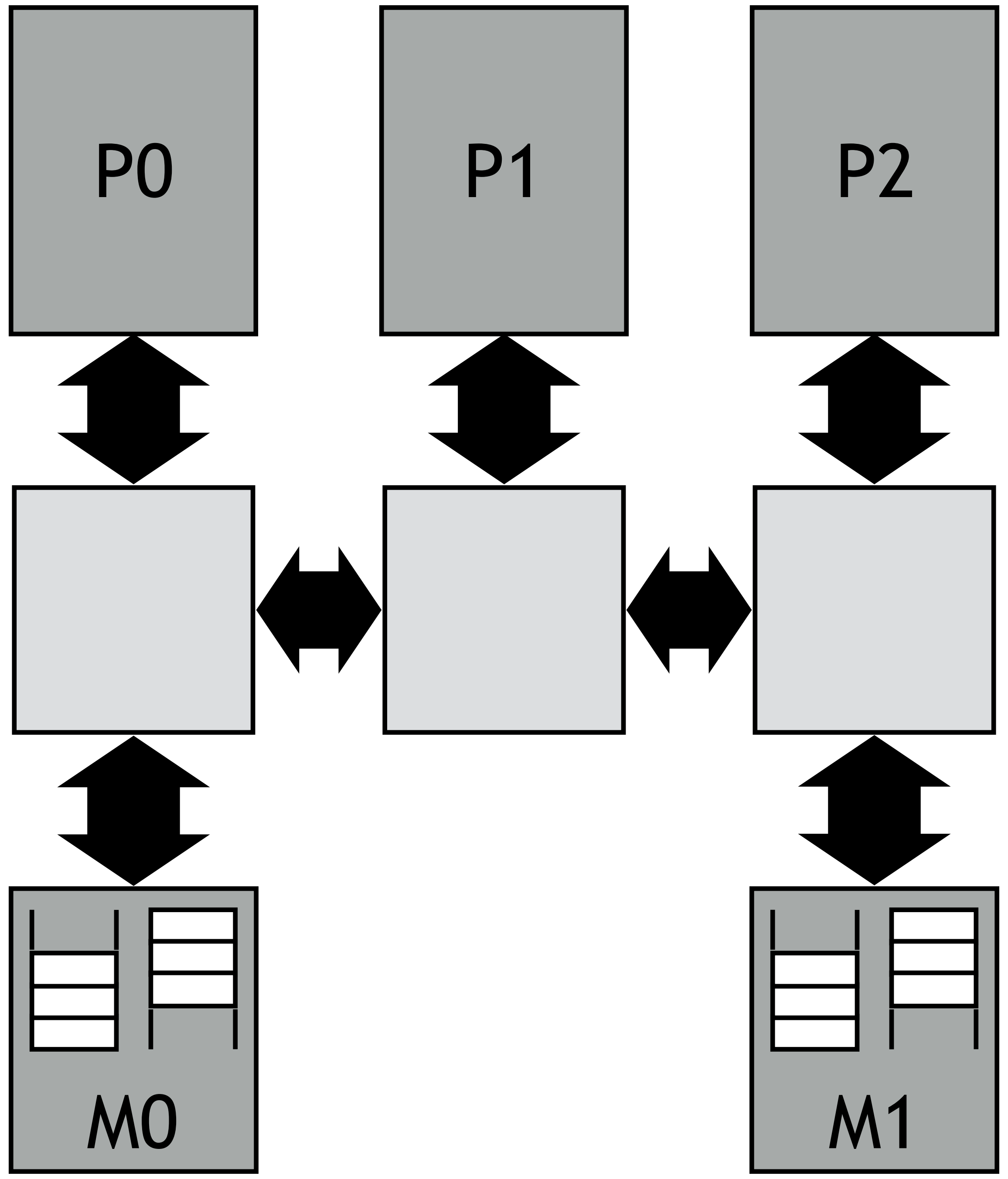
Reduce the impact of long-latency memory accesses

Exploit locality - frequent memory accesses

Automatically replicate and migrate data across memory modules to exploit locality effects



equidistant



inequidistant

CACHE-ONLY MEMORY ARCHITECTURE

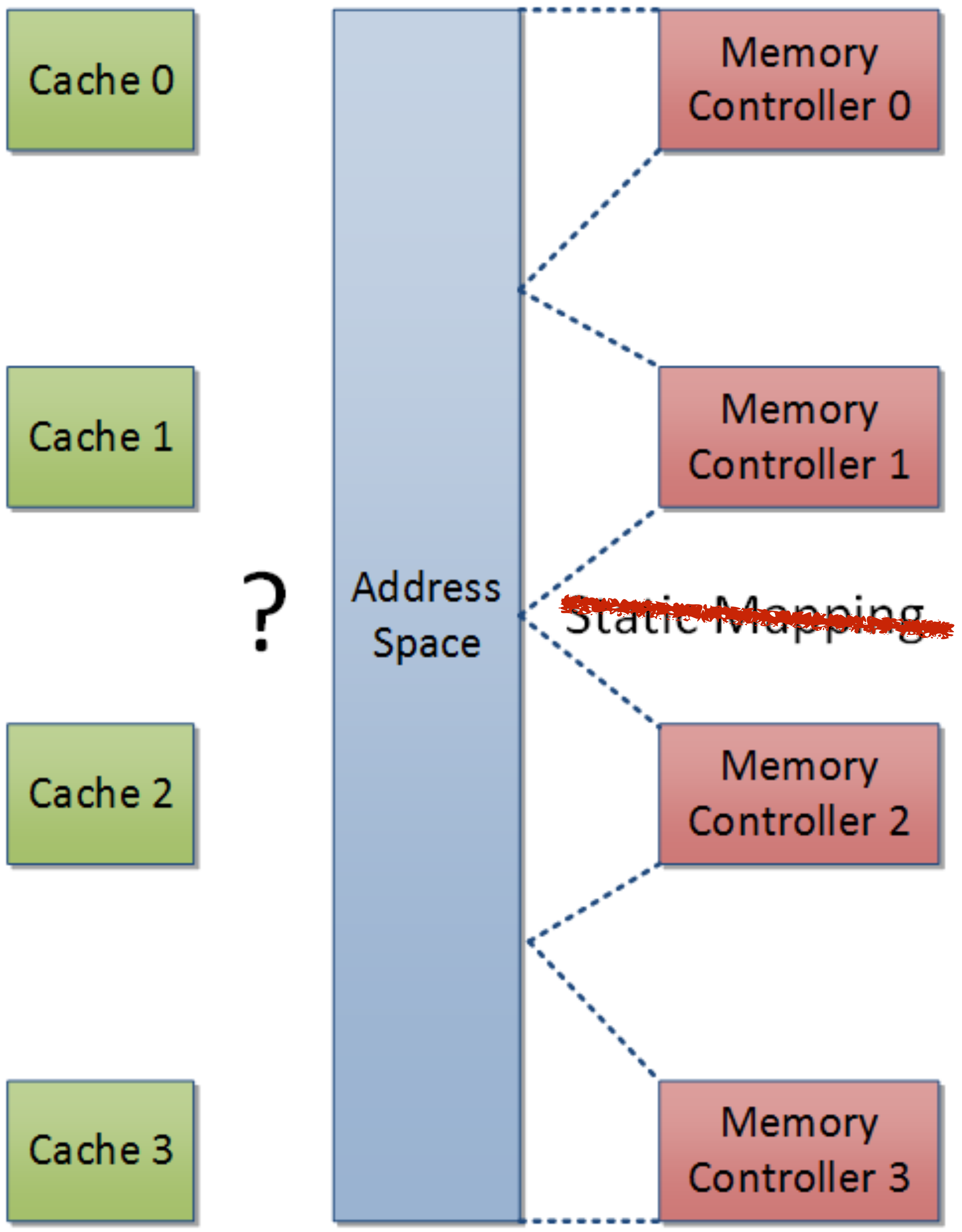
Make all memory available for migration/replication

- Overcome capacity restrictions
- Data distribution is now obsolete, looks like centralized main memory
- More complex coherence protocols :(

All memory is DRAM cache, called attraction memory

Key questions

- How to find data? Hierarchical or flat structures (dedicated directories)
- How to deal with replacements?



HIERARCHICAL COMA

Attraction memory as one giant hardware cache

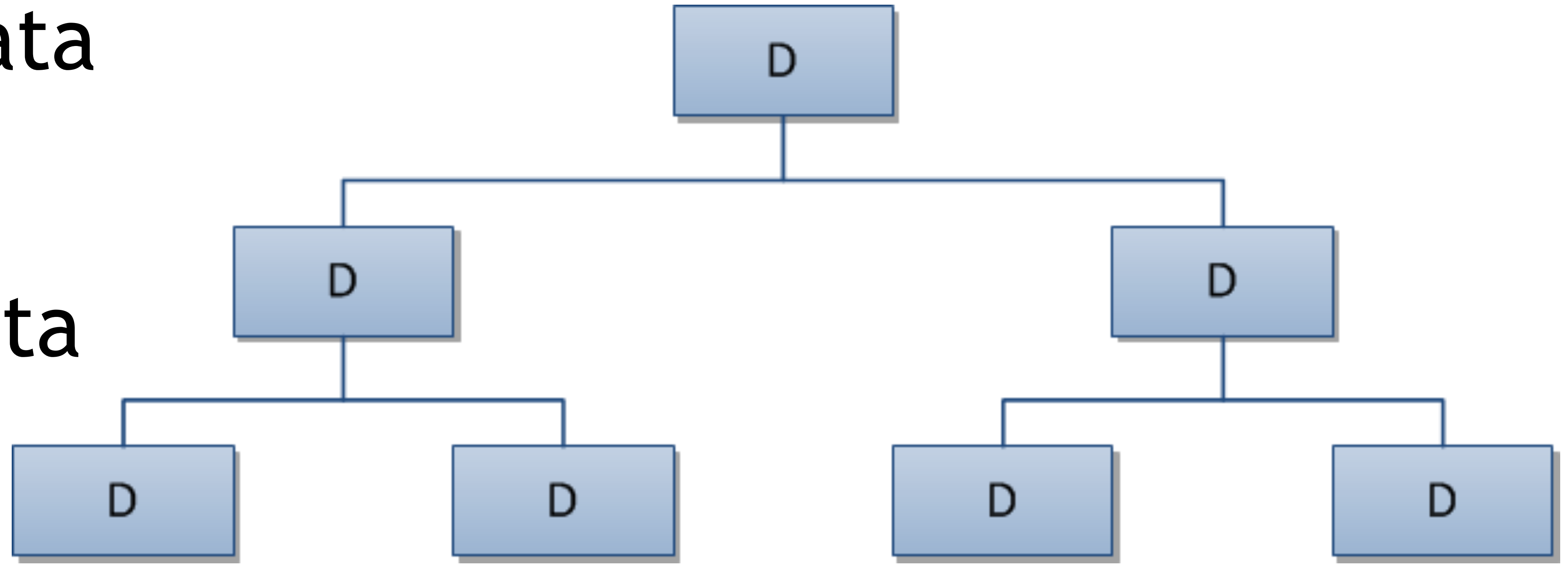
Maintains both address tags and state

Data addressed, allocated, kept coherent in blocks ("items")

Directory info on a per-block basis

Not home-based

- Data is migratory -> read requests "attract" data
- Must find a "home" during replacement
- Must find the directory entry before finding data



FLAT COMA

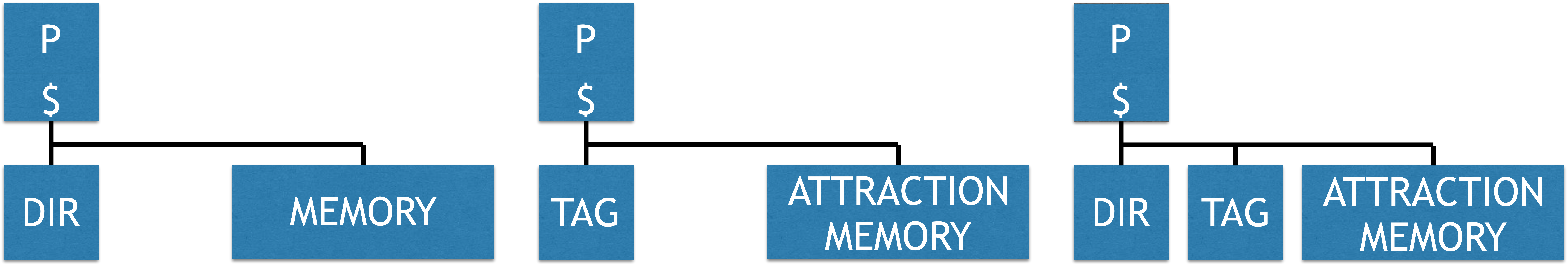
NUMA (standard cache-coherent systems, x86, POWER, etc)

Hierarchical COMA

- Data Diffusion Machine (1980 – mid 1990s)
- KSR1 (mid 1990s, “Allcache” -> “Allcrash”)

Flat COMA

- Fixed home node for directory, but not data (mid 1990s)



NUMA: only cache blocks migrate, memory/ directory does not

Hierarchical COMA: memory and directory can migrate

Flat COMA: memory blocks can freely migrate, but directory entries do not

CACHE-ONLY MEMORY ARCHITECTURE

(+) Certainly beneficial for locality

(+) No data distribution necessary

(+) No/less redundant cache block copies

(-) Latency is significantly increased due to tree traversals and multiple main memory accesses

(-) Main memory has to be highly associative

(-) Replacement of exclusive copies

TOKEN COHERENCE
DECOUPLING PERFORMANCE AND CORRECTNESS

GENERAL GOAL

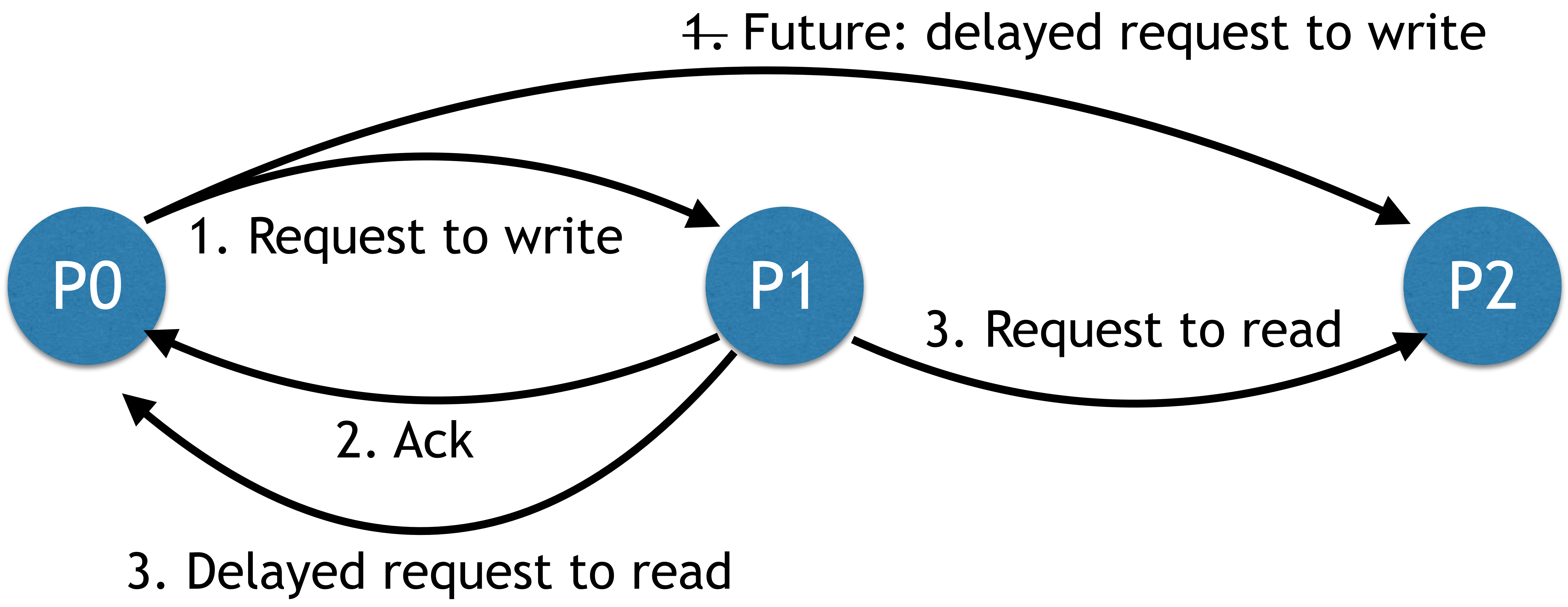
- Best of both
  - Broadcast with direct responses (like snooping)
  - Use unordered interconnect (like directory)
- Works fine with no races
  - But what happens in the case of races?
- Successive refinement
  - Common case (fast)
  - Uncommon case (safety)
  - Pathological case (starvation freedom)

Cache-to-cache latency

Interconnect	High	Low
Ordered		Snooping
Un-ordered	Directories	Token Coherence

BASIC APPROACH (NOT YET CORRECT)

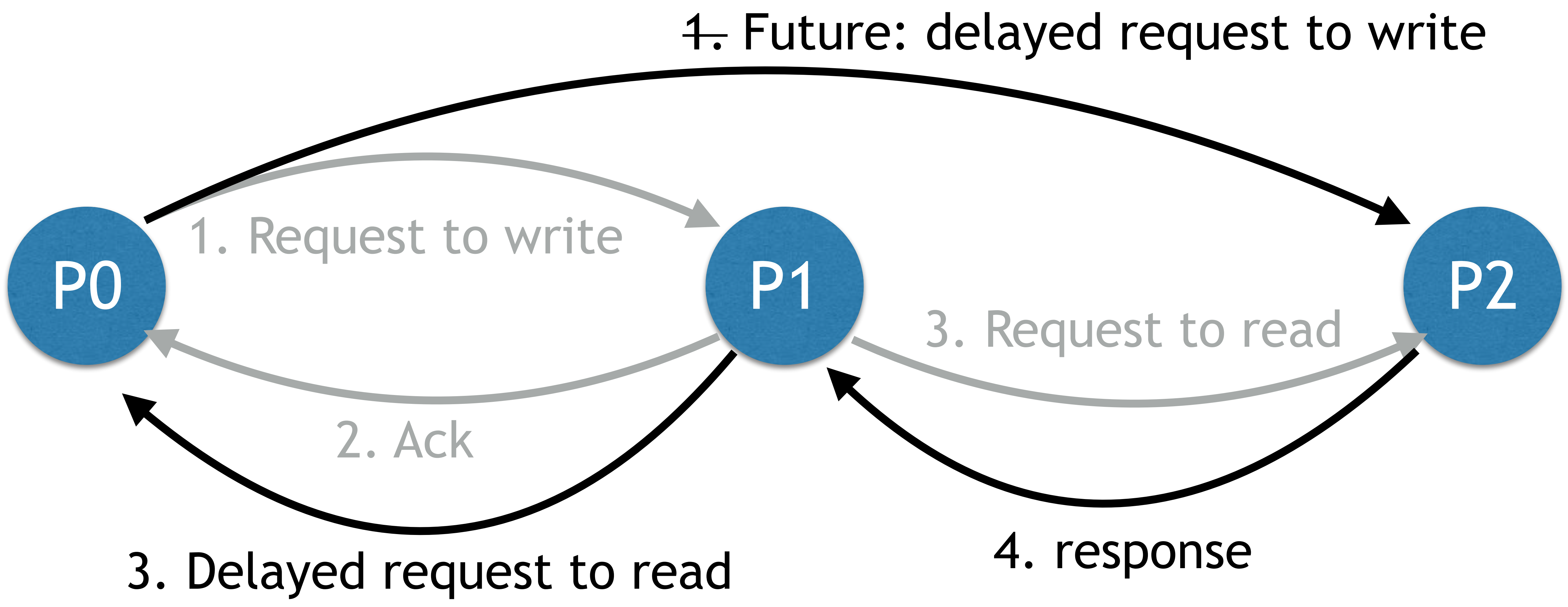
1. P0 issues a request to write (delayed to P2)
2. P1 acknowledges write request
3. P1 issues a request to read



P0 state	P1 state	P2 state
no copy	no copy	read/write

BASIC APPROACH (NOT YET CORRECT)

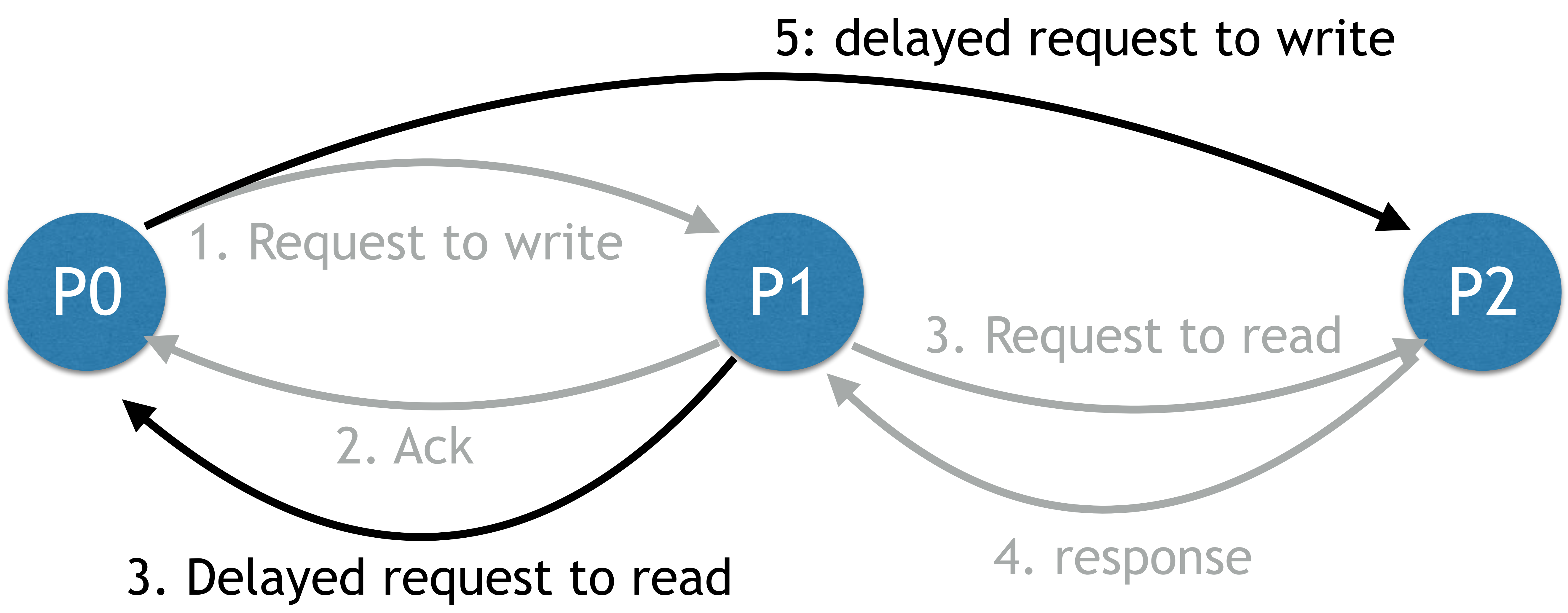
1. P0 issues a request to write (delayed to P2)
2. P1 acknowledges write request
3. P1 issues a request to read
4. P2 responds with data to P1



P0 state	P1 state	P2 state
no copy	no copy	read/write
	read-only	read-only

BASIC APPROACH (NOT YET CORRECT)

1. P0 issues a request to write (delayed to P2)
2. P1 acknowledges write request
3. P1 issues a request to read
4. P2 responds with data to P1
5. P0's delayed request arrives at P2
6. P2 responds to P0



P0 state	P1 state	P2 state
no copy	no copy	read/write
	read-only	read-only
read/write	read-only	no copy

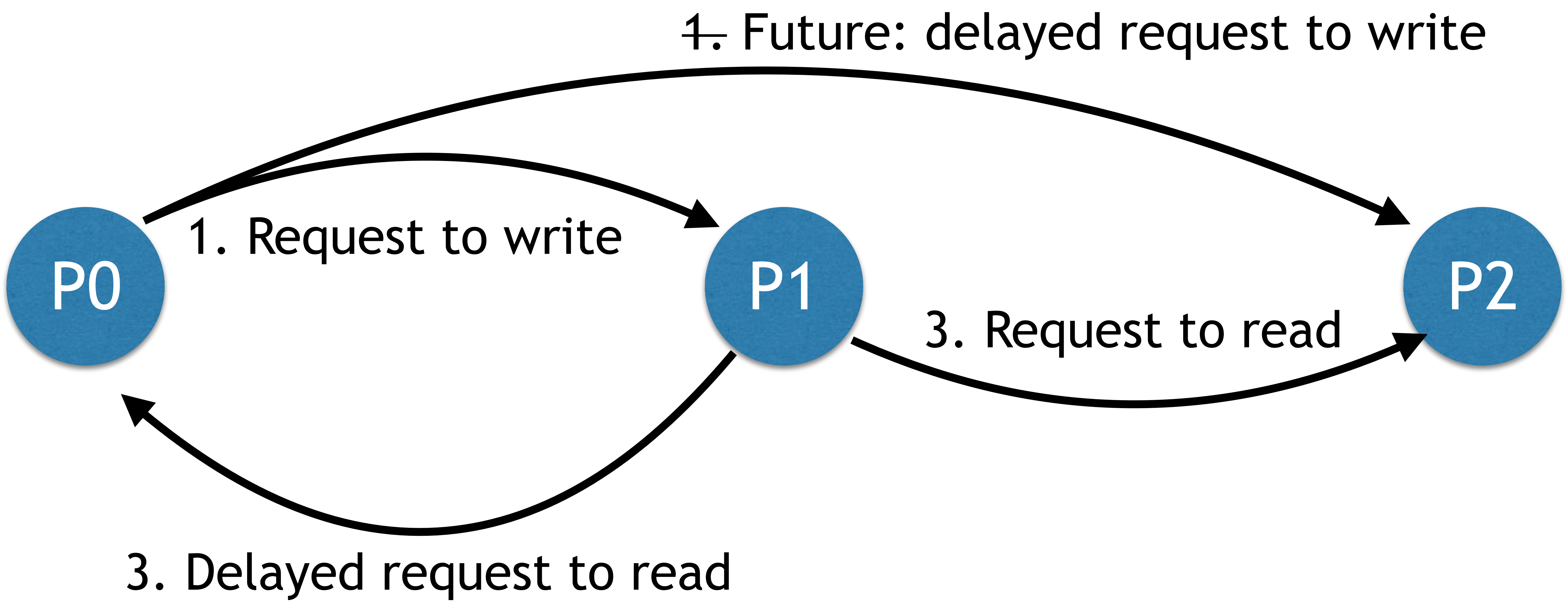
Problem: P0 and P1 are in inconsistent states (P0 can write, P1 can read)

KEY OBSERVATION: TOKEN COUNTING

- Explicitly encode permissions with tokens
  - All times, each blocks has T tokens (e.g., one token per processor)
  - Components exchange tokens & data
  - Tokens: in caches, memory or transit
- Controls reading & writing of data
  - One or more to read, all tokens to write
  - => Provides safety in all cases
- As before
  - Broadcast with direct responses (like snooping)
  - Use unordered interconnect (like directory)

BASIC APPROACH (REVISITED)

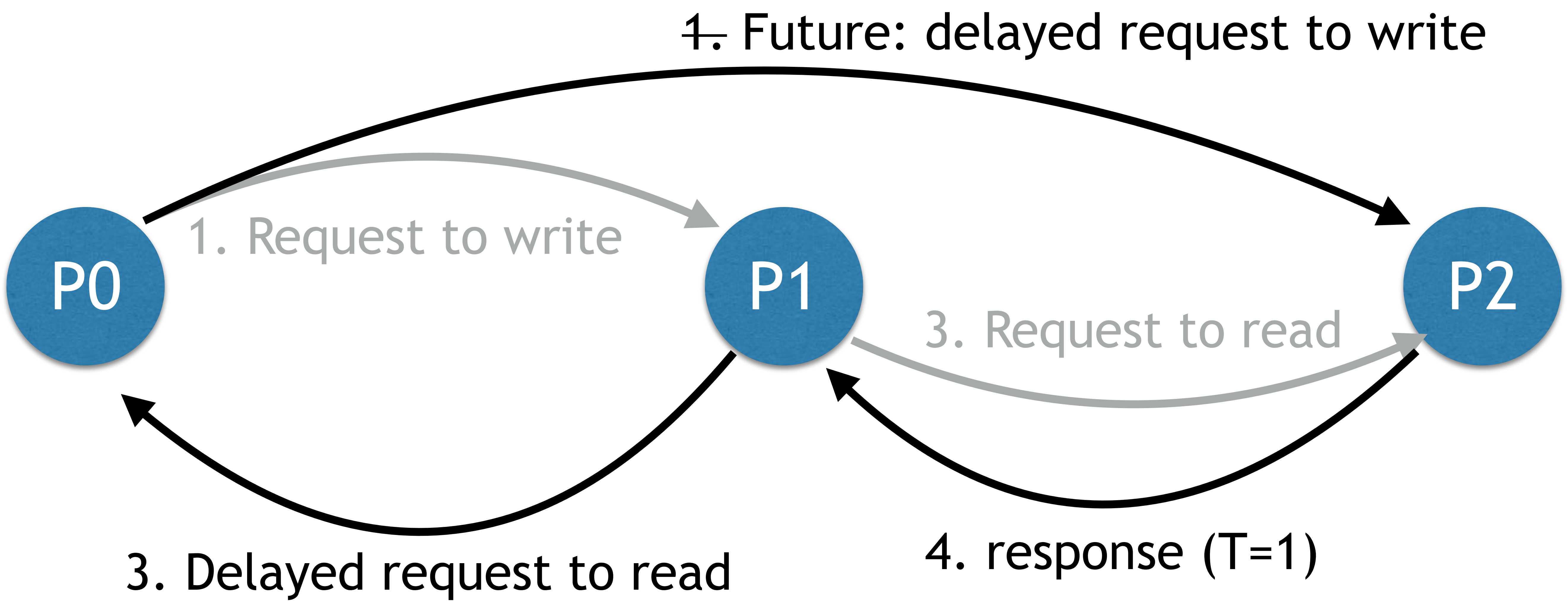
1. P0 issues a request to write (delayed to P2)
2. P1 acknowledges write request
3. P1 issues a request to read



P0 state	P1 state	P2 state
T=0	T=0	T=16 (R/W)

BASIC APPROACH (REVISITED)

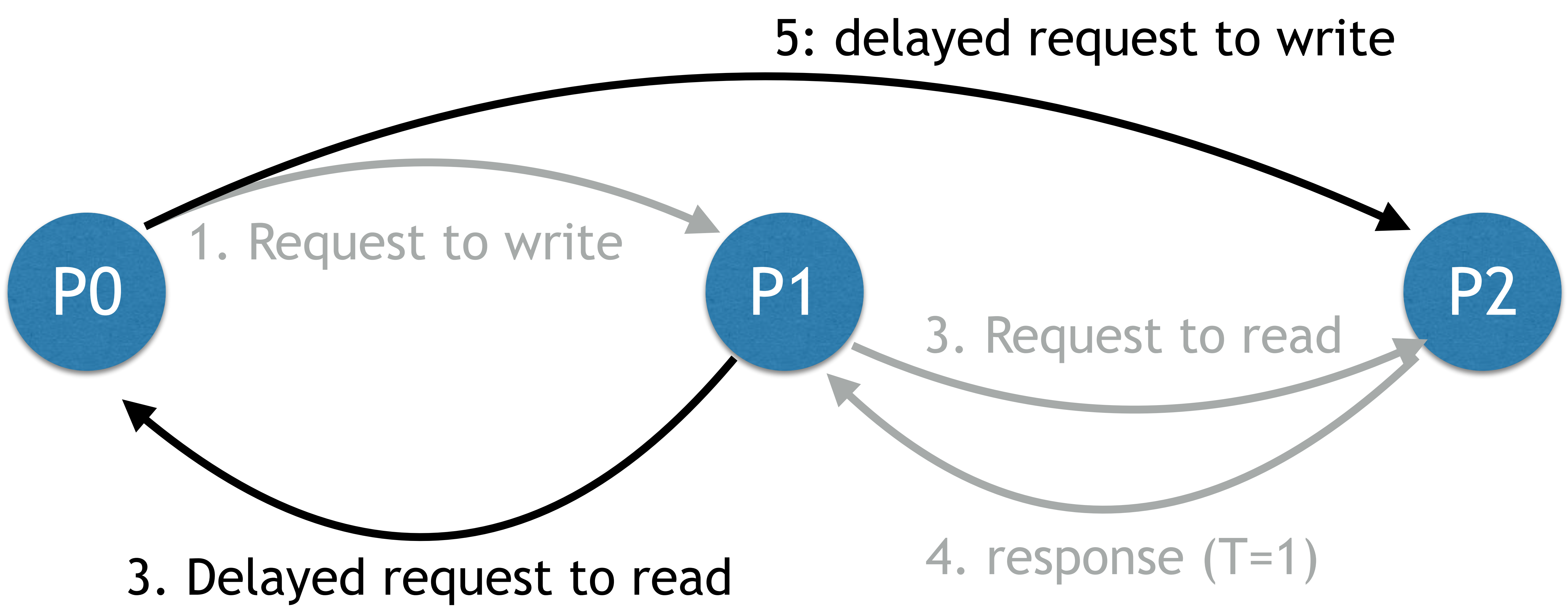
1. P0 issues a request to write (delayed to P2)
2. P1 acknowledges write request
3. P1 issues a request to read
4. P2 responds with data to P1



P0 state	P1 state	P2 state
T=0	T=0	T=16 (R/W)
	T=1 (R)	T=15 (R)

BASIC APPROACH (REVISITED)

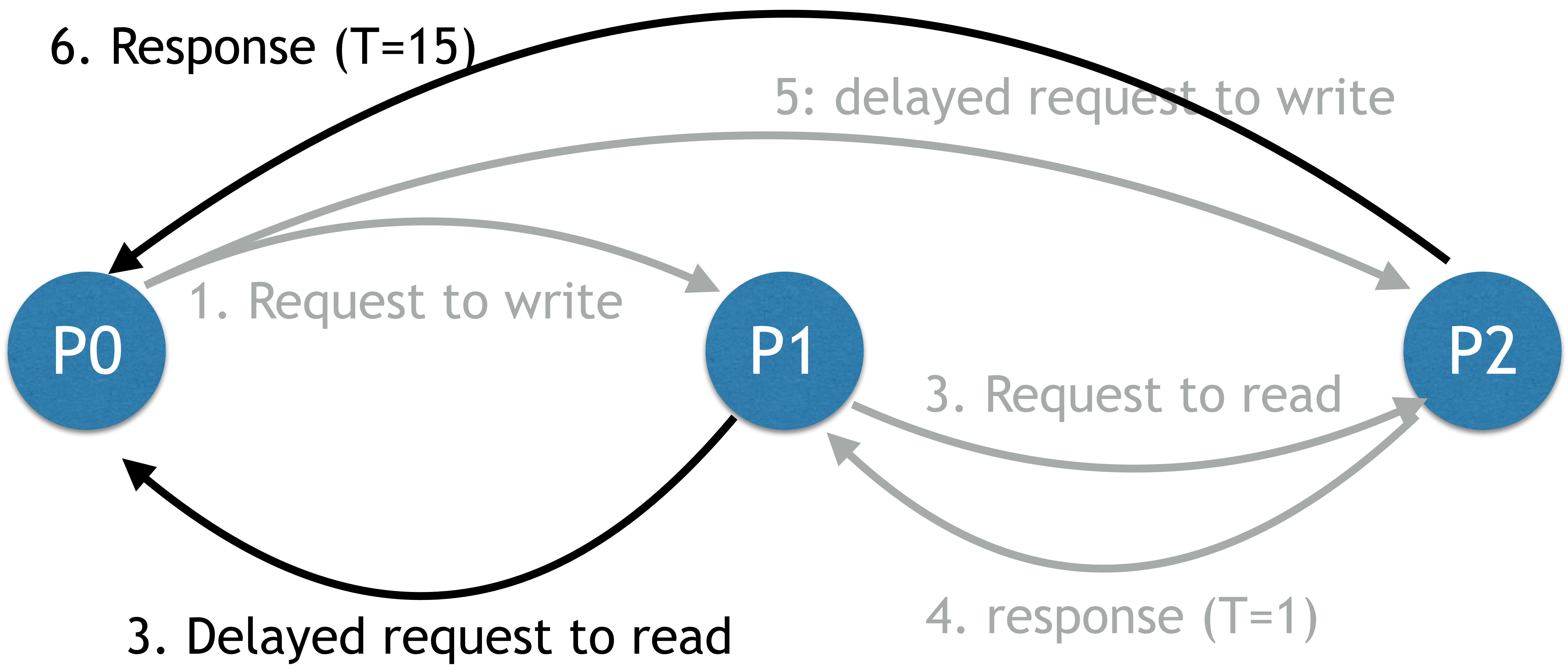
1. P0 issues a request to write (delayed to P2)
2. P1 acknowledges write request
3. P1 issues a request to read
4. P2 responds with data to P1
5. P0's delayed request arrives at P2



P0 state	P1 state	P2 state
T=0	T=0	T=16 (R/W)
	T=1 (R)	T=15 (R)

BASIC APPROACH (REVISITED)

1. P0 issues a request to write (delayed to P2)
2. P1 acknowledges write request
3. P1 issues a request to read
4. P2 responds with data to P1
5. P0's delayed request arrives at P2
6. P2 responds to P0



P0 state	P1 state	P2 state
T=0	T=0	T=16 (R/W)
T=0	T=1 (R)	T=15 (R)
T=15 (R)	T=1 (R)	T=0

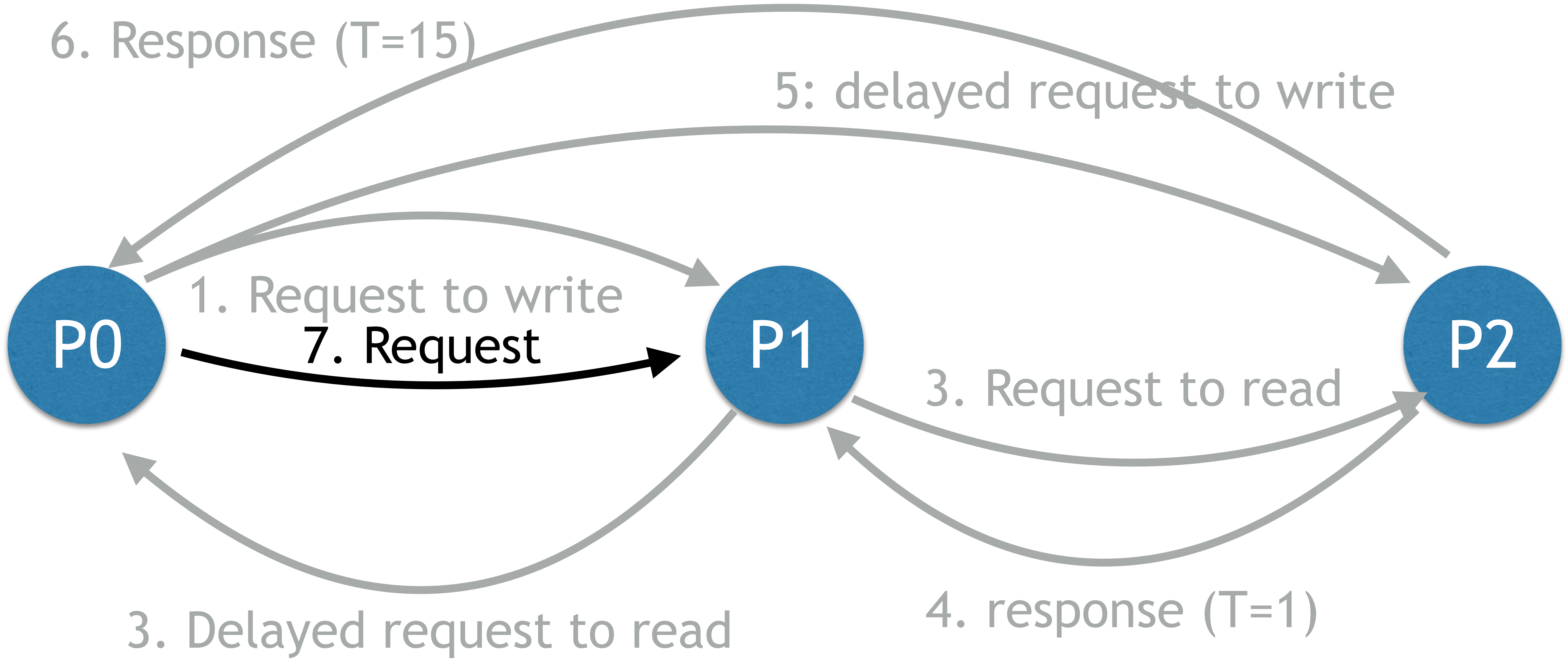
Valid! But now what? P0 wants all the tokens!

BASIC APPROACH (RE-REVISITED)

- Re-issue requests as needed
  - Needed due to racing requests (uncommon)
  - Timeout to detect failed completion
  - E.g., wait twice average miss latency
  - Small hardware overhead
- All races are handled in this uniform fashion

BASIC APPROACH (REVISITED)

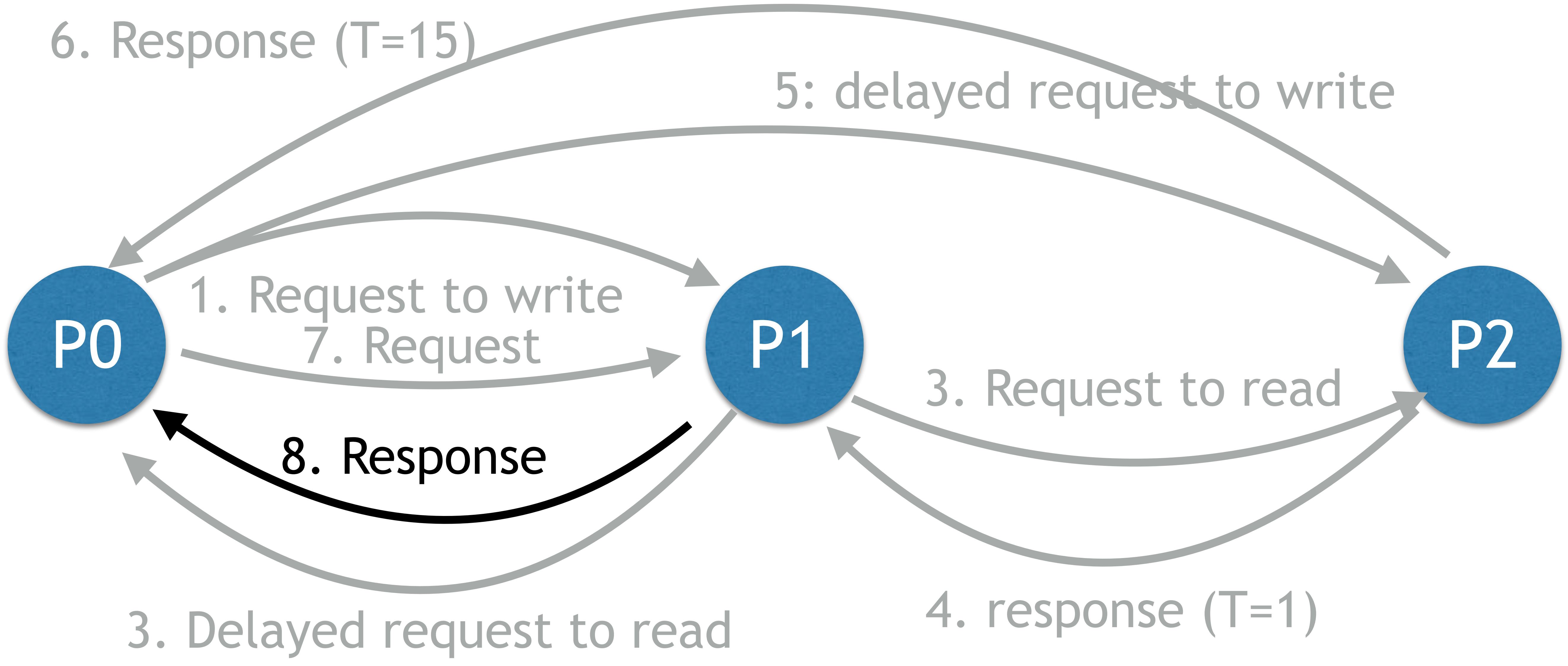
1. P0 issues a request to write (delayed to P2)
2. P1 acknowledges write request
3. P1 issues a request to read
4. P2 responds with data to P1
5. P0’s delayed request arrives at P2
6. P2 responds to P0
7. P0 re-issues request



P0 state	P1 state	P2 state
T=0	T=0	T=16 (R/W)
T=0	T=1 (R)	T=15 (R)
T=15 (R)	T=1 (R)	T=0

BASIC APPROACH (REVISITED)

1. P0 issues a request to write (delayed to P2)
2. P1 acknowledges write request
3. P1 issues a request to read
4. P2 responds with data to P1
5. P0's delayed request arrives at P2
6. P2 responds to P0
7. P0 re-issues request
8. P1 responds with a token



P0 state	P1 state	P2 state
T=0	T=0	T=16 (R/W)
T=0	T=1 (R)	T=15 (R)
T=15 (R)	T=1 (R)	T=0
T=16 (R/W)	T=0	T=0

P0 has all the tokens, thus RW access to the CL

GUARANTEEING STARVATION-FREEDOM

- Handle pathological cases
    - Infrequently invoked
    - Can be slow, inefficient and simple
- When normal requests fail to succeed (4x)
    - Longer timeout and issue a persistent request
    - Requests persist until satisfied
    - Table at each processor
    - “Deactivate” upon completion
- Implementation
    - Arbiter at memory orders persistent requests

Prime example of
optimizing the common case

MORE INFORMATIONS IN PAPERS

- Traffic optimizations
  - Transfer tokens without data
  - Add an “owner” token
  - Upgrade (read-only to read/write)
  - “Exclusive clean” state
- Note: no silent read-only replacements
  - Worst case: 10% interconnect traffic overhead
- Encoding tokens in memory
  - Using ECC bits
  - Reduce read-modify-writes with token cache

DO RE-ISSUED REQUESTS HURT?

Re-issues have to be uncommon for Token Coherence to perform well

- Re-issued requests are slower and consume more bandwidth than misses that succeed on the first attempt
- On average for the workloads, 97% of TokenB’s cache misses are issued only once

Races are rare in the workloads

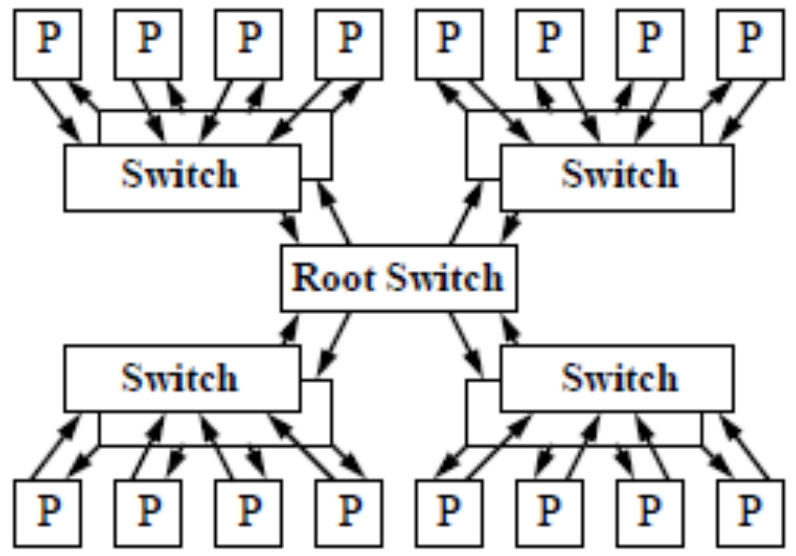
- Multiple processors rarely access the same data simultaneously due to the large amount of shared data

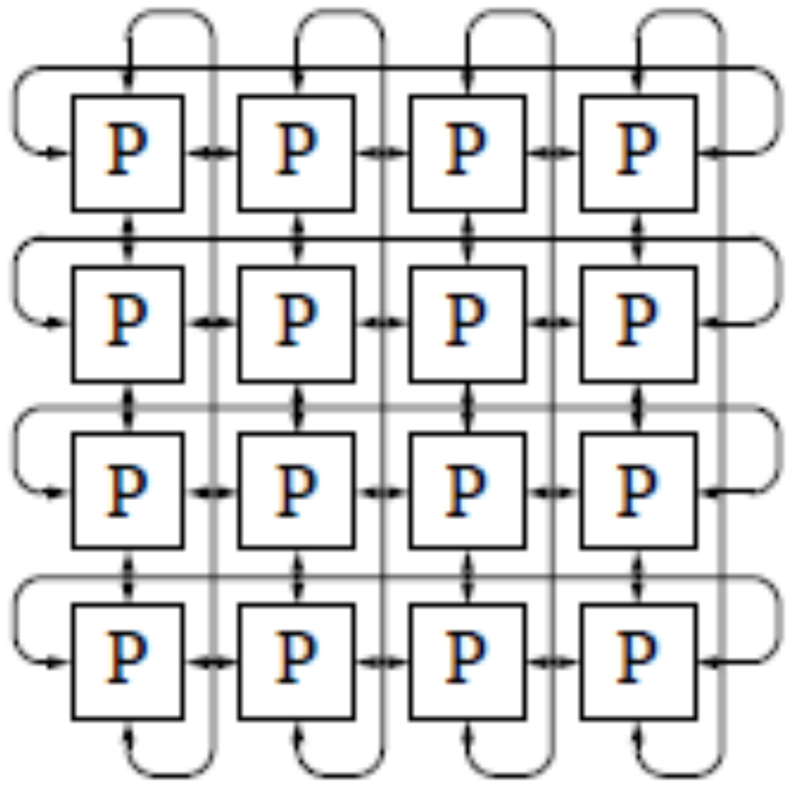
Persistent requests are the worst case, but they are really seldom

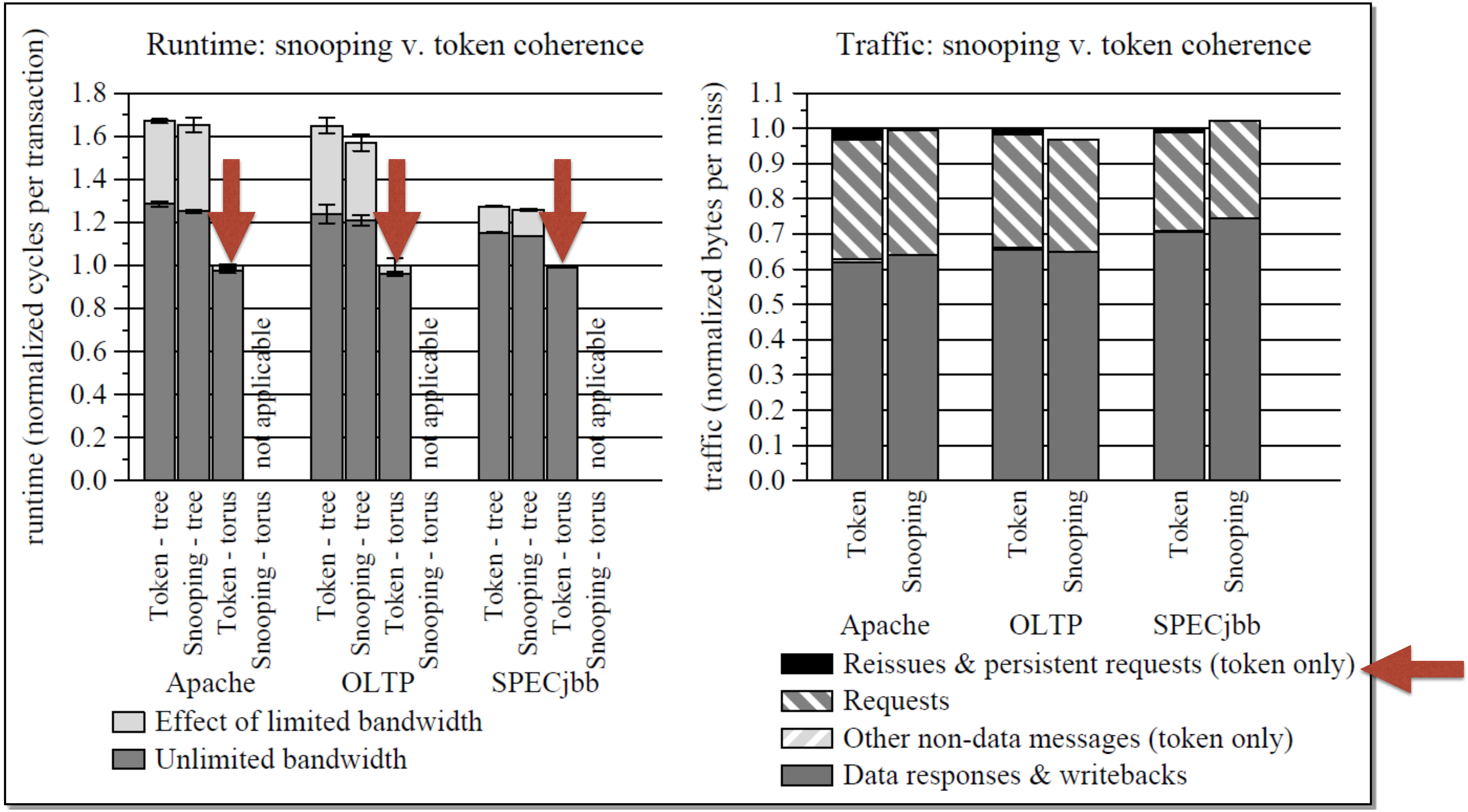
Workload	Percentage of Misses: Not Reissued	Reissued Once	Reissued > Once	Persistent Requests
Apache	95.75%	3.25%	0.71%	0.29%
OLTP	97.57%	1.79%	0.43%	0.21%
SPECjbb	97.60%	2.03%	0.30%	0.07%
Average	96.97%	2.36%	0.48%	0.19%

Milo M. K. Martin, Mark D. Hill, and David A. Wood. 2003. Token coherence: decoupling performance and correctness. ISCA '03. DOI: https://doi.org/10.1145/859618.859640

PERFORMANCE RESULTS - TOPOLOGY



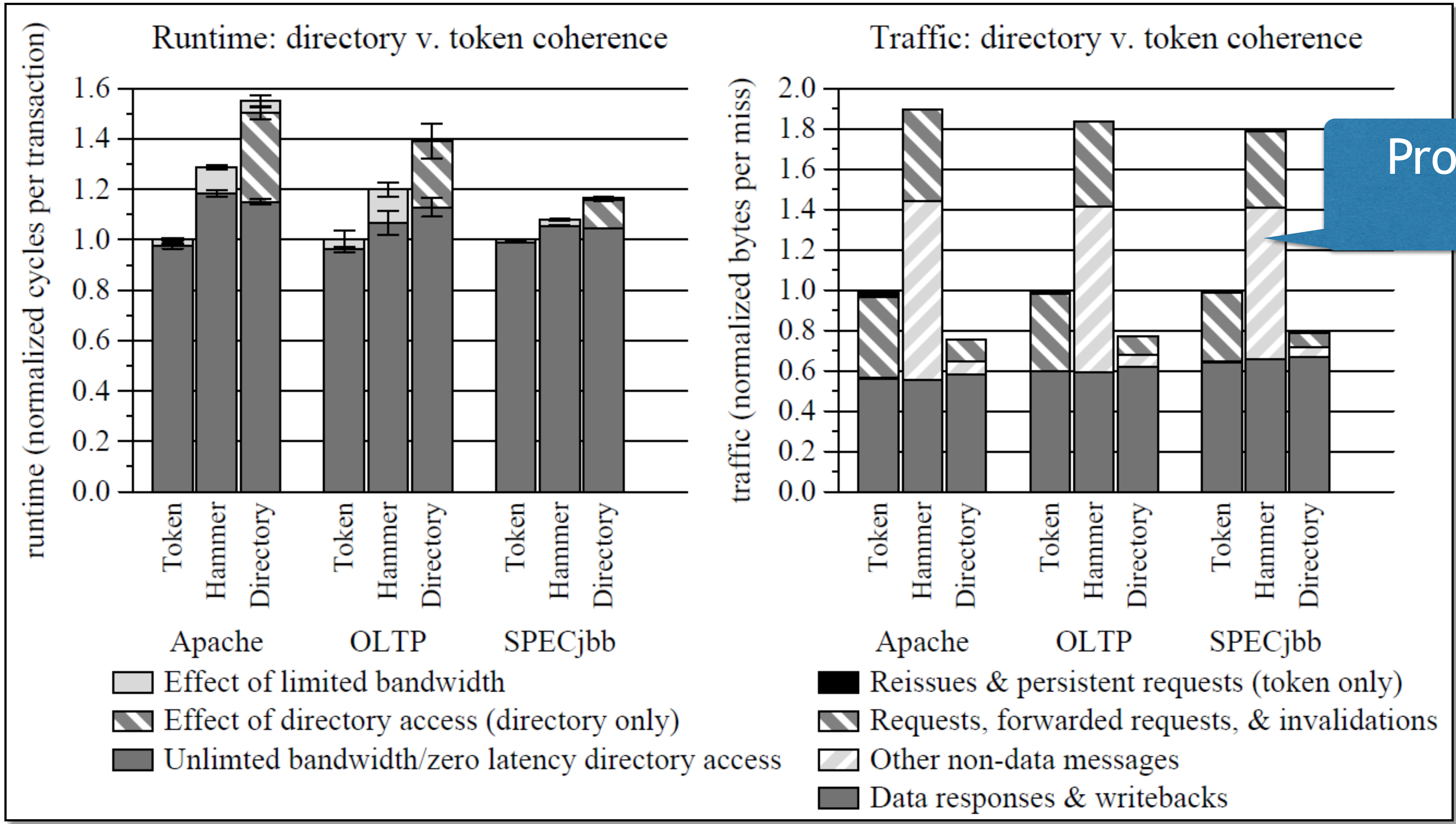




Milo M. K. Martin, Mark D. Hill, and David A. Wood. 2003. Token coherence: decoupling performance and correctness. ISCA '03. DOI: https://doi.org/10.1145/859618.859640

PERFORMANCE RESULTS - ALTERNATIVES

Milo M. K. Martin, Mark D. Hill, and David A. Wood. 2003. *Token coherence: decoupling performance and correctness*. ISCA '03. DOI: https://doi.org/10.1145/859618.859640



Probes and Probe Responses

Simulation of 16 processors
Hammer: broadcast-based, unordered interconnects (Opteron, POWER4, …)
Directory: full-map with directory in DRAM, no ordering, no NACKs or retries

TRADITIONAL VERSUS TOKEN COHERENCE

Traditional protocols

- Sensitive to request ordering
  - Interconnect or directory
- Monolithic
  - Complicated
  - Intertwine correctness and performance

Token coherence

- Track tokens (safety)
- Persistent requests (starvation avoidance)
- Requests are only “hints”

=> Separate correctness and performance
=> Decoupled Coherence

SUMMARY

Probe filtering: a highly viable approach to significantly improve the performance of snooping protocols

- However, limited in its use as it can require lots of directory space

COMA: not viable today as latency issues prevent a broad use, furthermore complicated hardware

Token Coherence: improving coherence performance

- Effectively, it reduces the associated overhead
- Solution to the problem? Maybe – it still relies on a (unreliable/unordered) broadcast
- High traffic (excellent for 16 processors, but questionable for 64)