ECE 669

Parallel Computer Architecture

Lecture 18

Scalable Parallel Caches

Overview

- Most cache protocols are more complicated than two state
- Snooping not effective for network-based systems
  - Consider three alternate cache coherence approaches
  - Full-map, limited directory, chain
- Caching will affect network performance
- Limitless protocol
  - Gives appearance of full-map
- Practical issues of processor – memory system interaction

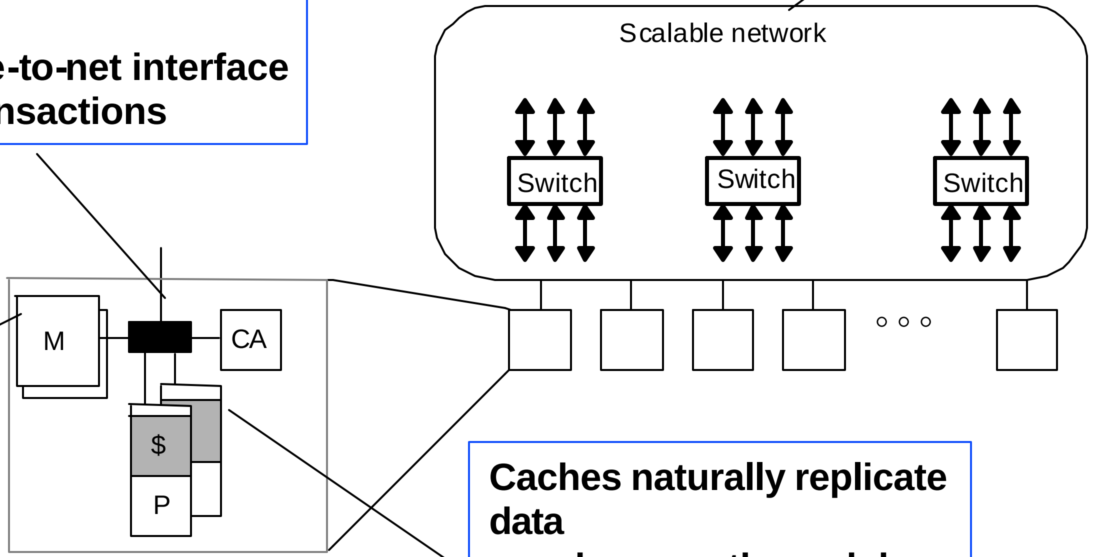
Context for Scalable Cache Coherence

Realizing Pgm Models through net transaction protocols
- efficient node-to-net interface
- interprets transactions

Scalable Networks
- many simultaneous transactions

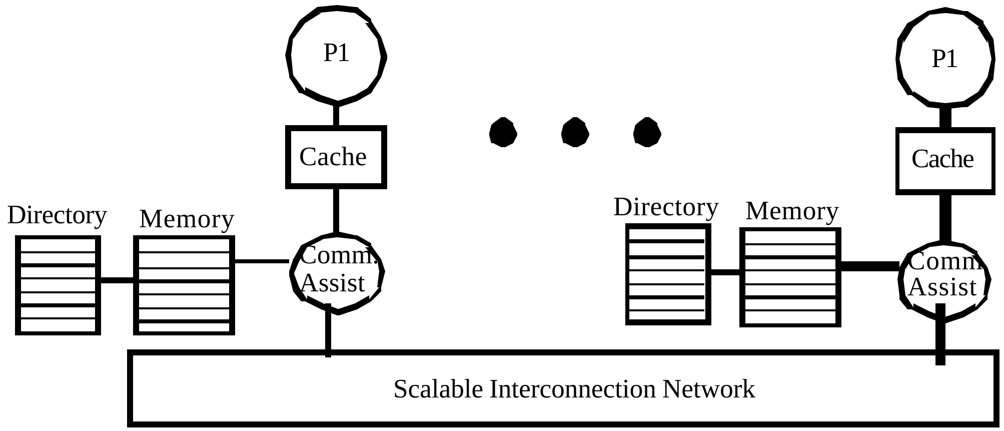
Scalable distributed memory

Caches naturally replicate data
- coherence through bus snooping protocols
- consistency



Need cache coherence protocols that scale!
- no broadcast or single point of order

Generic Solution: Directories

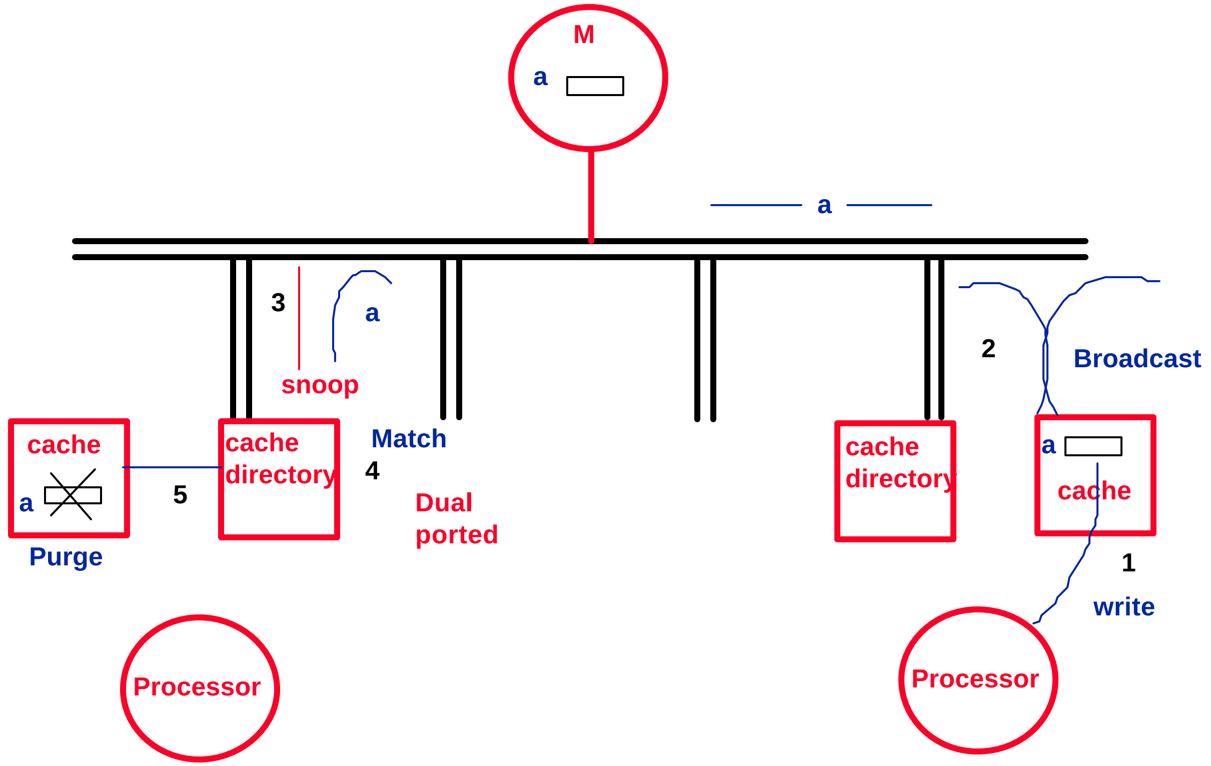


- **Maintain state vector explicitly**
  - **associate with memory block**
  - **records state of block in each cache**
- **On miss, communicate with directory**
  - **determine location of cached copies**
  - **determine action to take**
  - **conduct protocol to maintain coherence**

A Cache Coherent System Must:

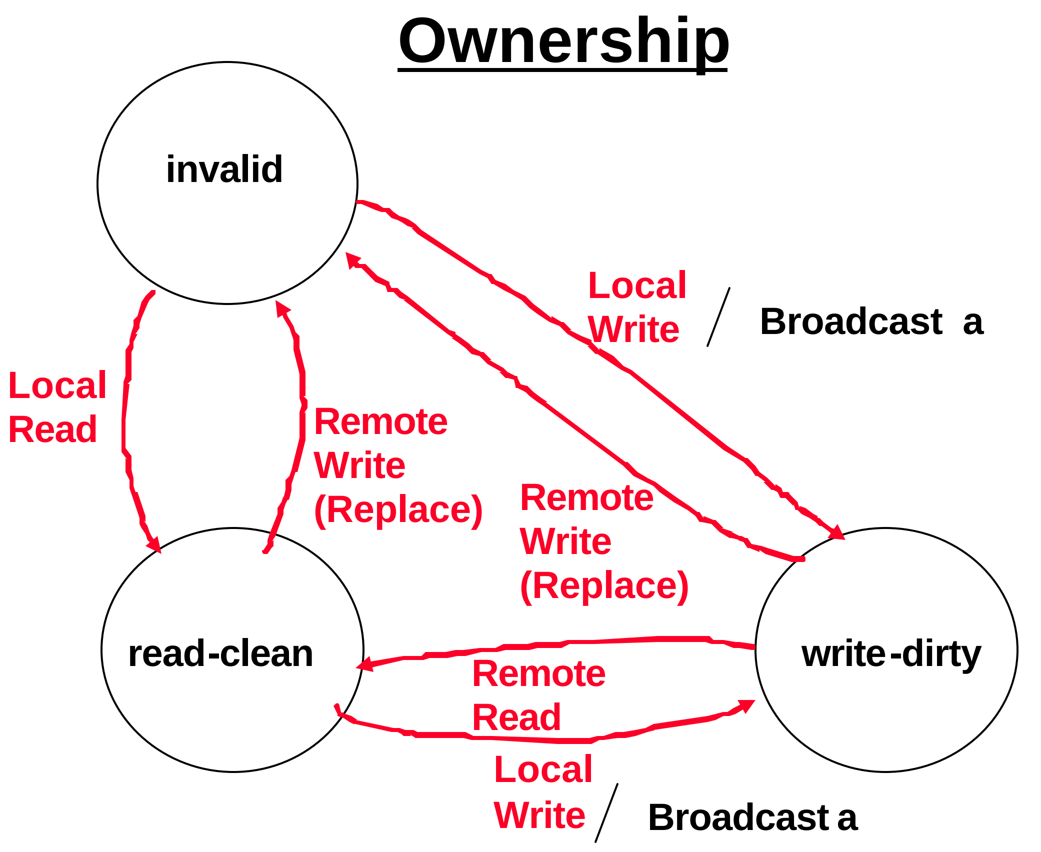
- **Provide set of states, state transition diagram, and actions**
- **Manage coherence protocol**
  - **(0) Determine when to invoke coherence protocol**
  - **(a) Find info about state of block in other caches to determine action**
    - **whether need to communicate with other cached copies**
  - **(b) Locate the other copies**
  - **(c) Communicate with those copies (inval/update)**
- **(0) is done the same way on all systems**
  - **state of the line is maintained in the cache**
  - **protocol is invoked if an “access fault” occurs on the line**
- **Different approaches distinguished by (a) to (c)**

Coherence in small machines: Snooping Caches



- Broadcast address on shared write
- Everyone listens (snoops) on bus to see if any of their own addresses match
- How do you know when to broadcast, invalidate...
  - State associated with each cache line

State diagram for ownership protocols



- **In ownership protocol: writer owns exclusive copy**
- For each shared data cache block

Maintaining coherence in large machines

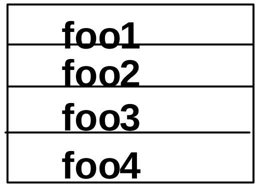
- Software
- Hardware - directories

° Software coherence

  Typically yields weak coherence

  i.e. Coherence at sync points (or fence pts)

° E.g.: When using critical sections for shared ops...



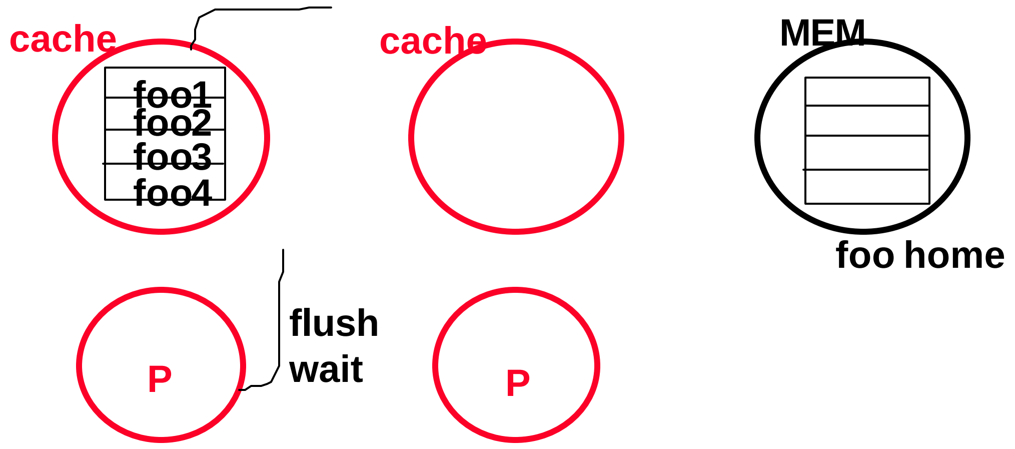
° Code

```
GET_FOO_LOCK
        /* MUNGE WITH FOOs */
                Foo1 =
                X = Foo2
                Foo3 =
                        .
                        .


RELEASE_FOO_LOCK
```

° How do you make this work?

Situation

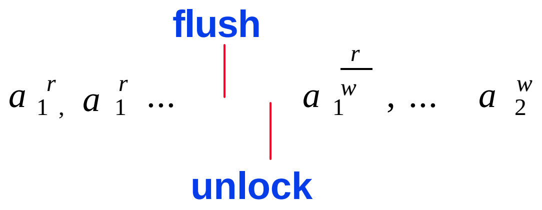


- Flush foo* from cache, wait till done
- Issues
  - Lock ?
  - Must be conservative

    - Lose some locality
  - But, can exploit appl. characteristics

    e.g. TSP, Chaotic

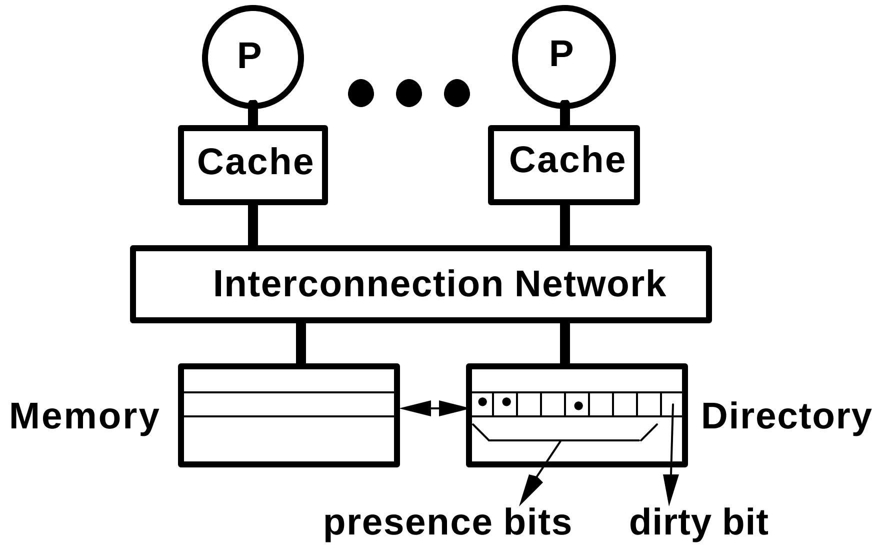
    Allow some inconsistency
  - Need special processor instructions



Scalable Approach: Directories

- Every memory block has associated directory information
  - keeps track of copies of cached blocks and their states
  - on a miss, find directory entry, look it up, and communicate only with the nodes that have copies if necessary
  - in scalable networks, communication with directory and copies is through network transactions
- Many alternatives for organizing directory information

Basic Operation of Directory



- k processors.
- With each cache-block in memory: k presence-bits, 1 dirty-bit
- With each cache-block in cache: 1 valid bit, and 1 dirty (owner) bit

- **Read from main memory by processor i:**
  - **If dirty-bit OFF then { read from main memory; turn p[i] ON; }**
  - **if dirty-bit ON then { recall line from dirty proc (cache state to shared); update memory; turn dirty-bit OFF; turn p[i] ON; supply recalled data to i;}**
- **Write to main memory by processor i:**
  - **If dirty-bit OFF then { supply data to i; send invalidations to all caches that have the block; turn dirty-bit ON; turn p[i] ON; ... }**

Scalable dynamic schemes

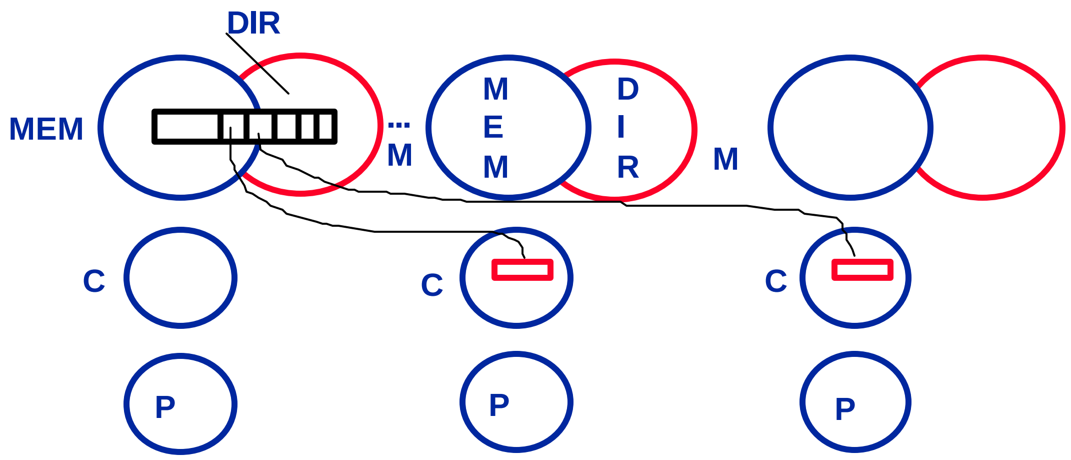
- Limited directories
- Chained directories
- Limitless schemes

$\downarrow$

Use software

Other approach: Full Map (not scalable)

Scalable hardware schemes



General directories:

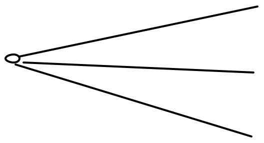
- **On write, check directory**

  if shared, send inv msg

- **Distribute directories with MEMs**

  Directory bandwidth scales in proportion to memory bandwidth

- **Most directory accesses during memory access -- so not too many extra network requests (except, write to read VAR)**

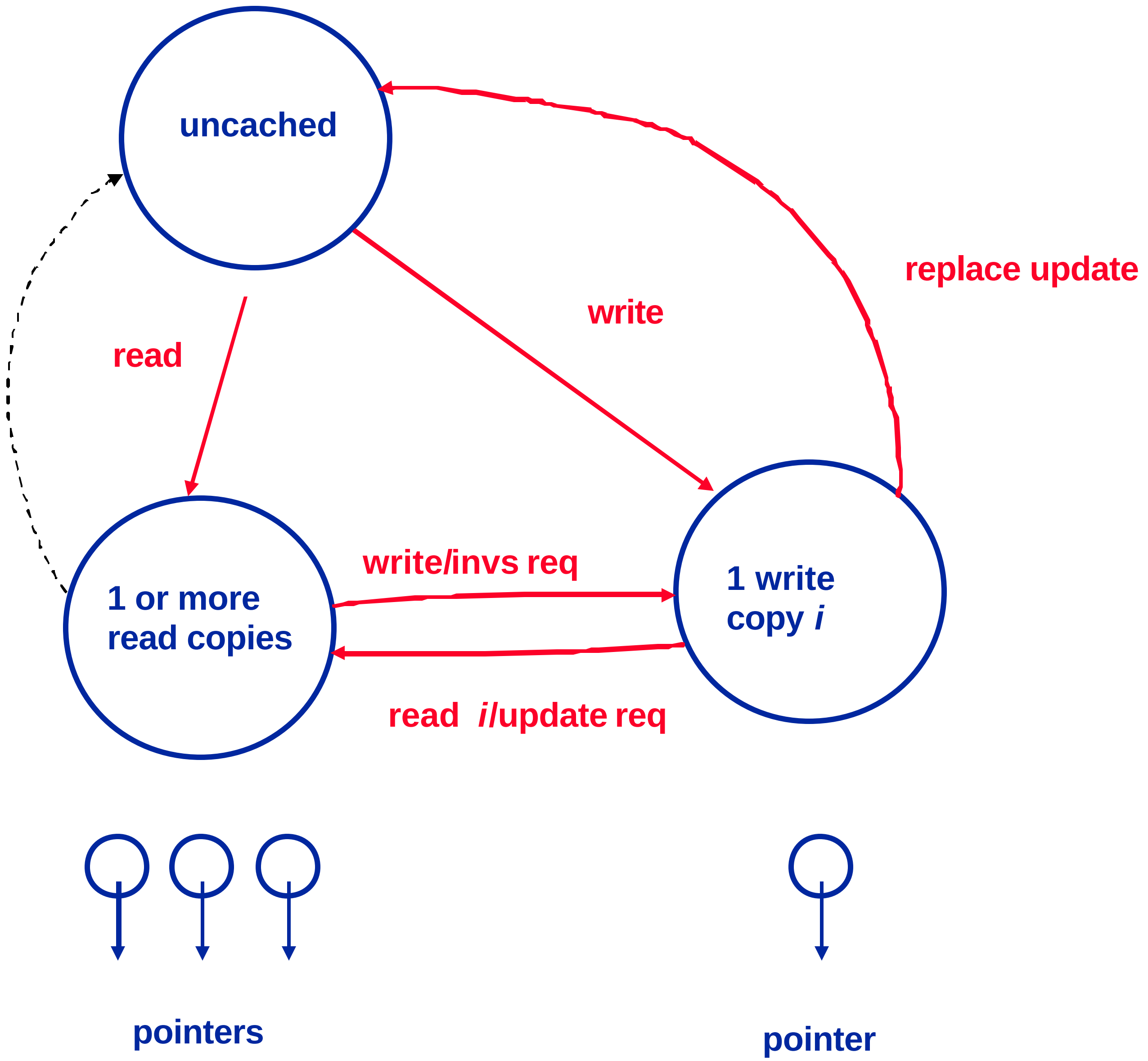


Limited directories

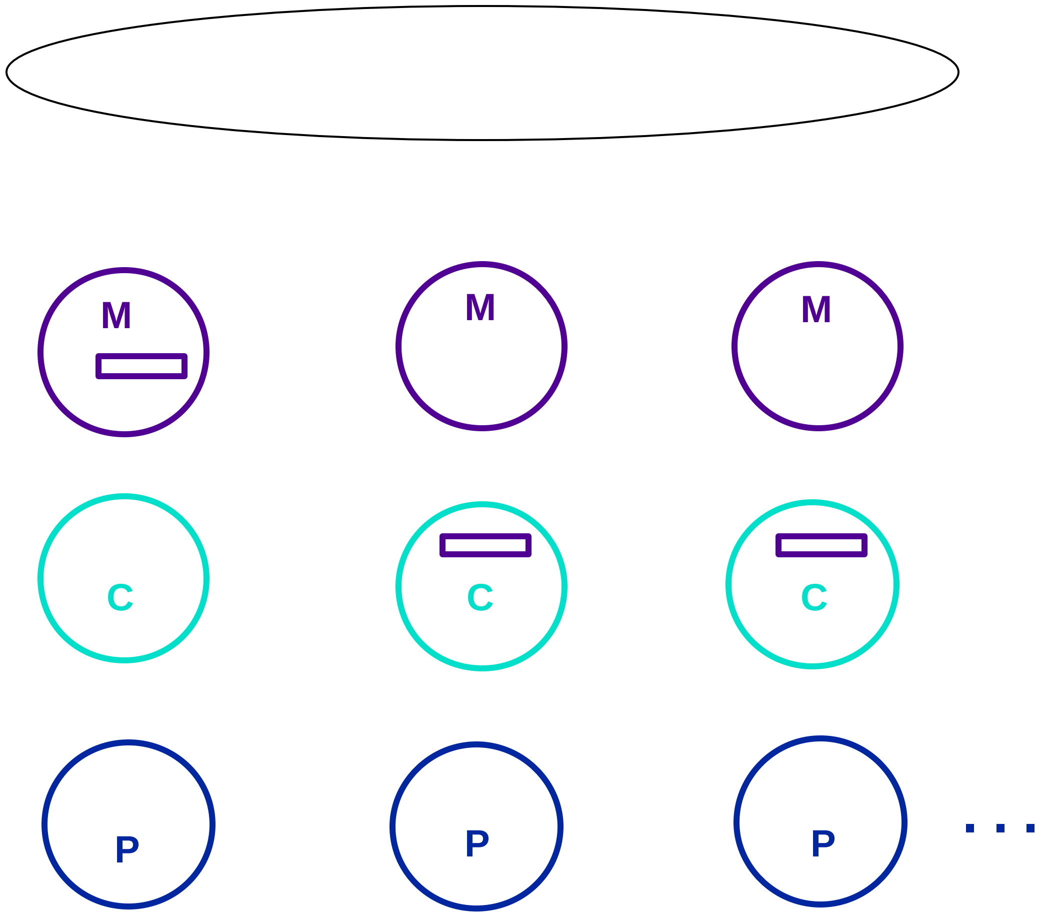
Chained directories

LimitLESS directories

Memory controller - (directory) state diagram for memory block



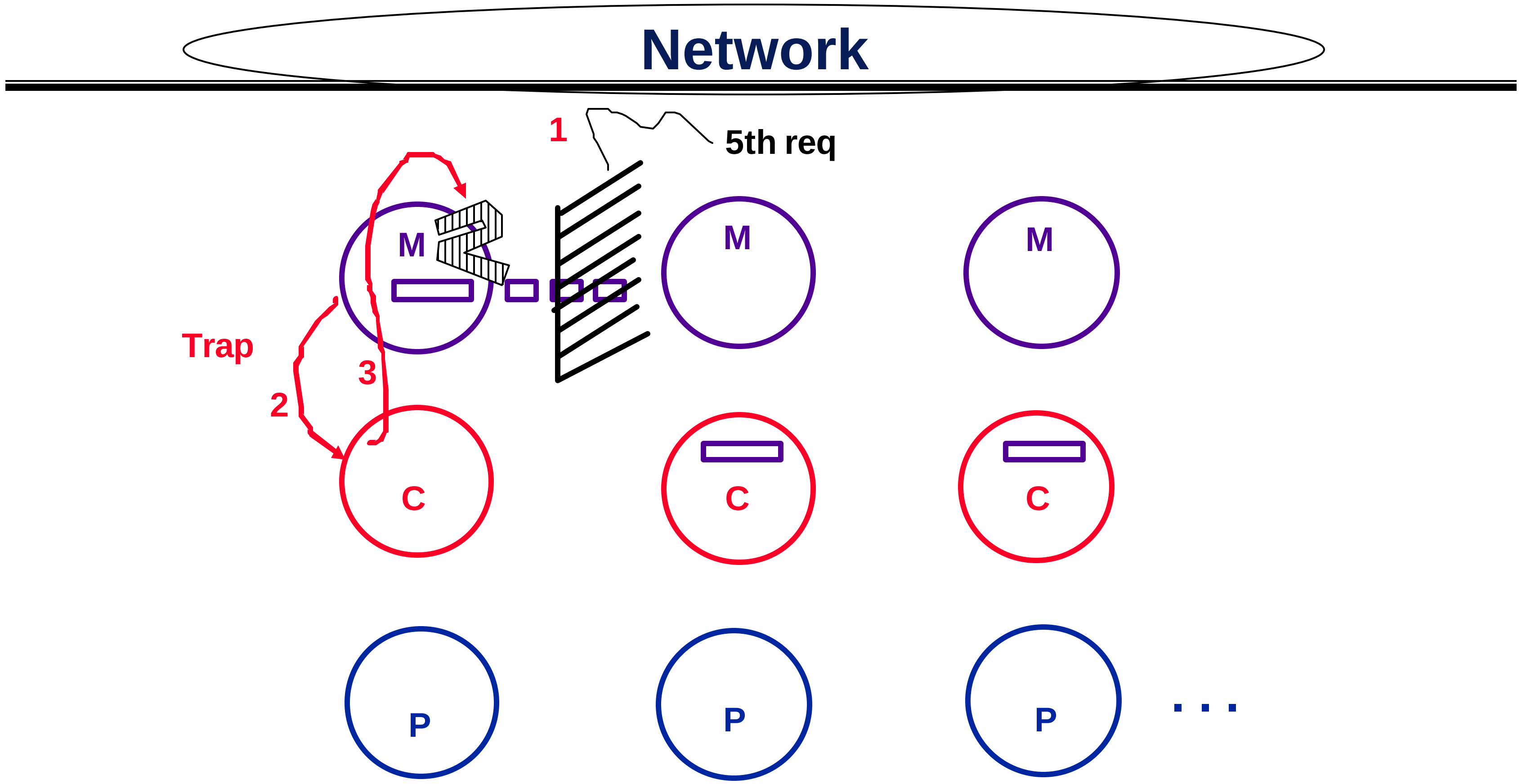
Network



Network

Limited directories: Exploit worker set behavior

- Invalidate 1 if 5th processor comes along (sometimes can set a broadcast invalidate bit)
- Rarely more than 2 processors share
- Insight: The set of 4 pointers can be managed like a fully-associative 4 entry cache on the virtual space of all pointers
- But what do you do about widely shared data?

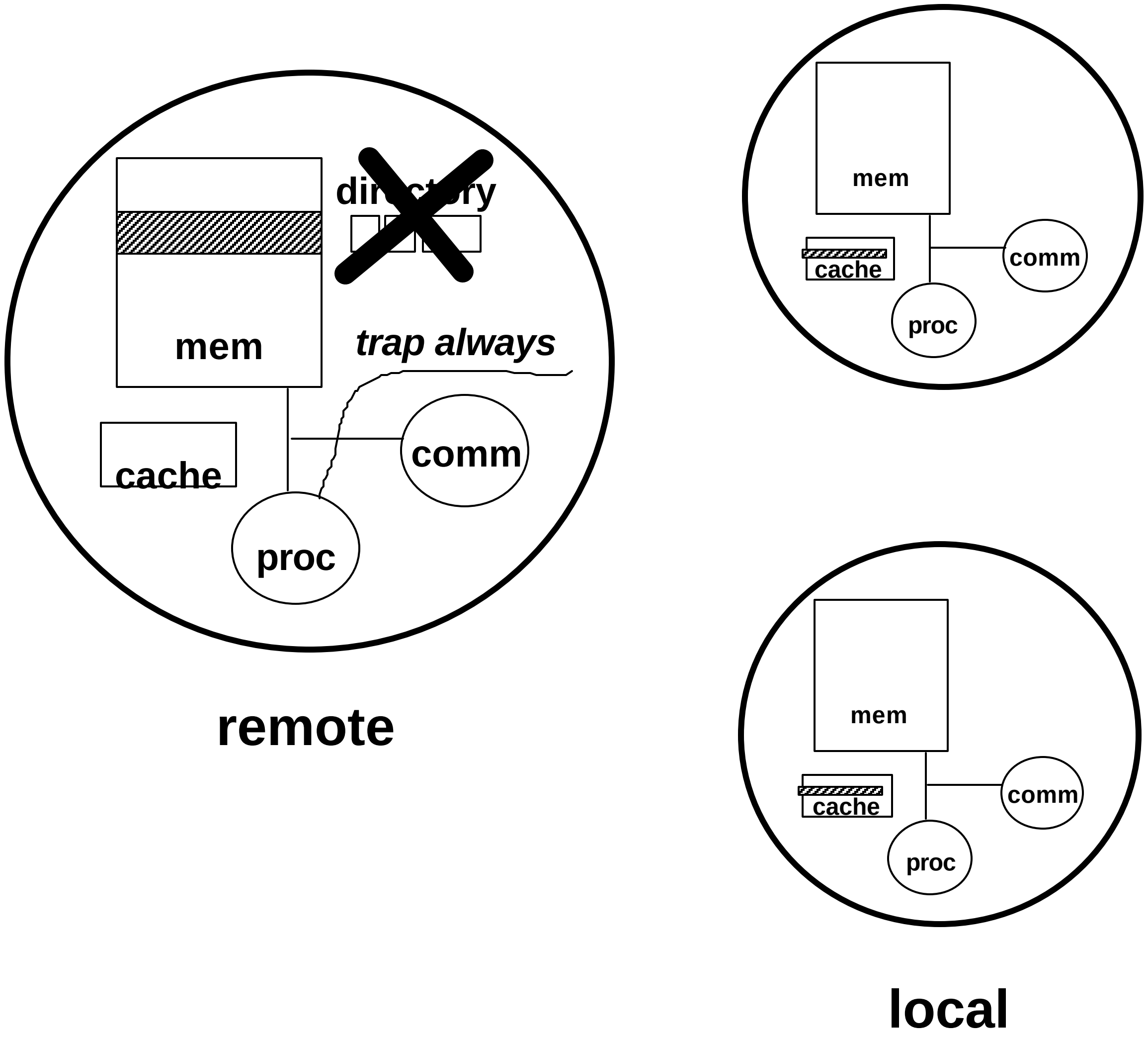


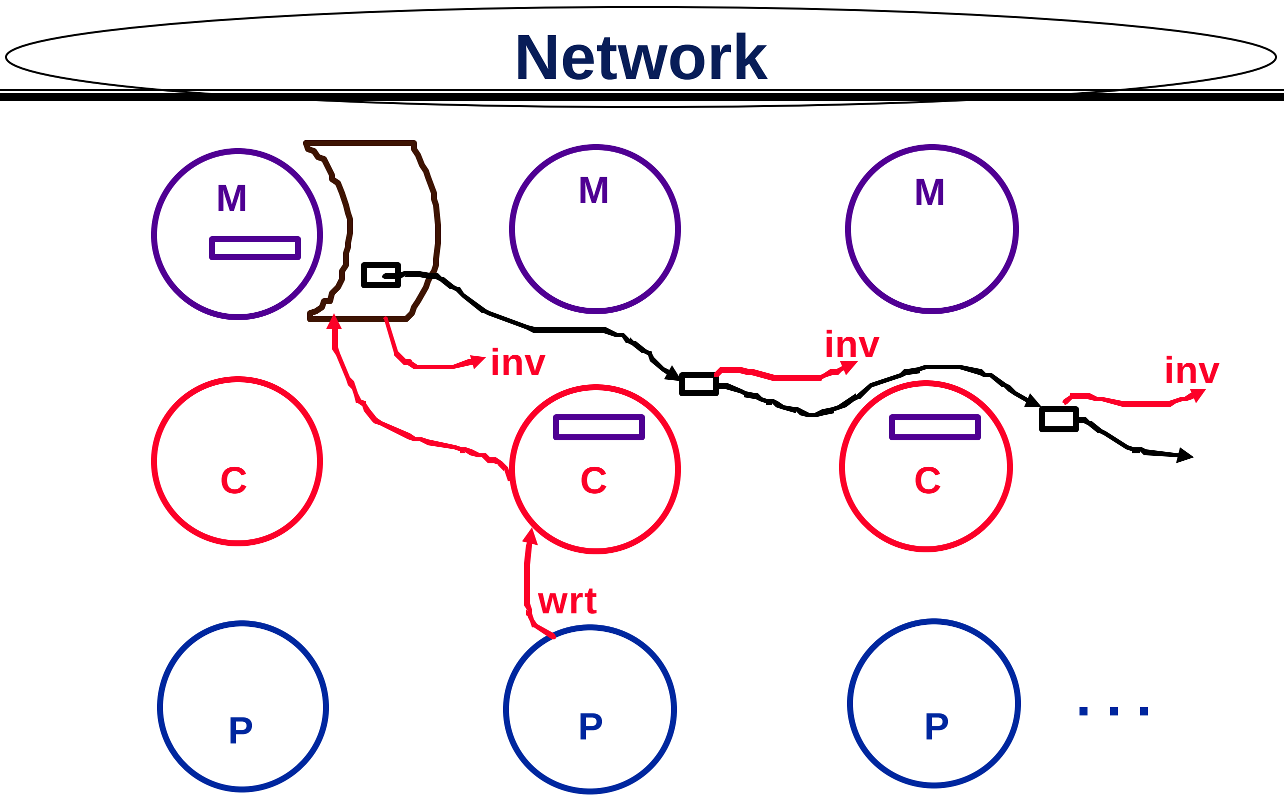
- **LimitLESS directories:**

  <u>Limited</u> directories <u>L</u>ocally <u>E</u>xtended through <u>S</u>oftware <u>S</u>upport

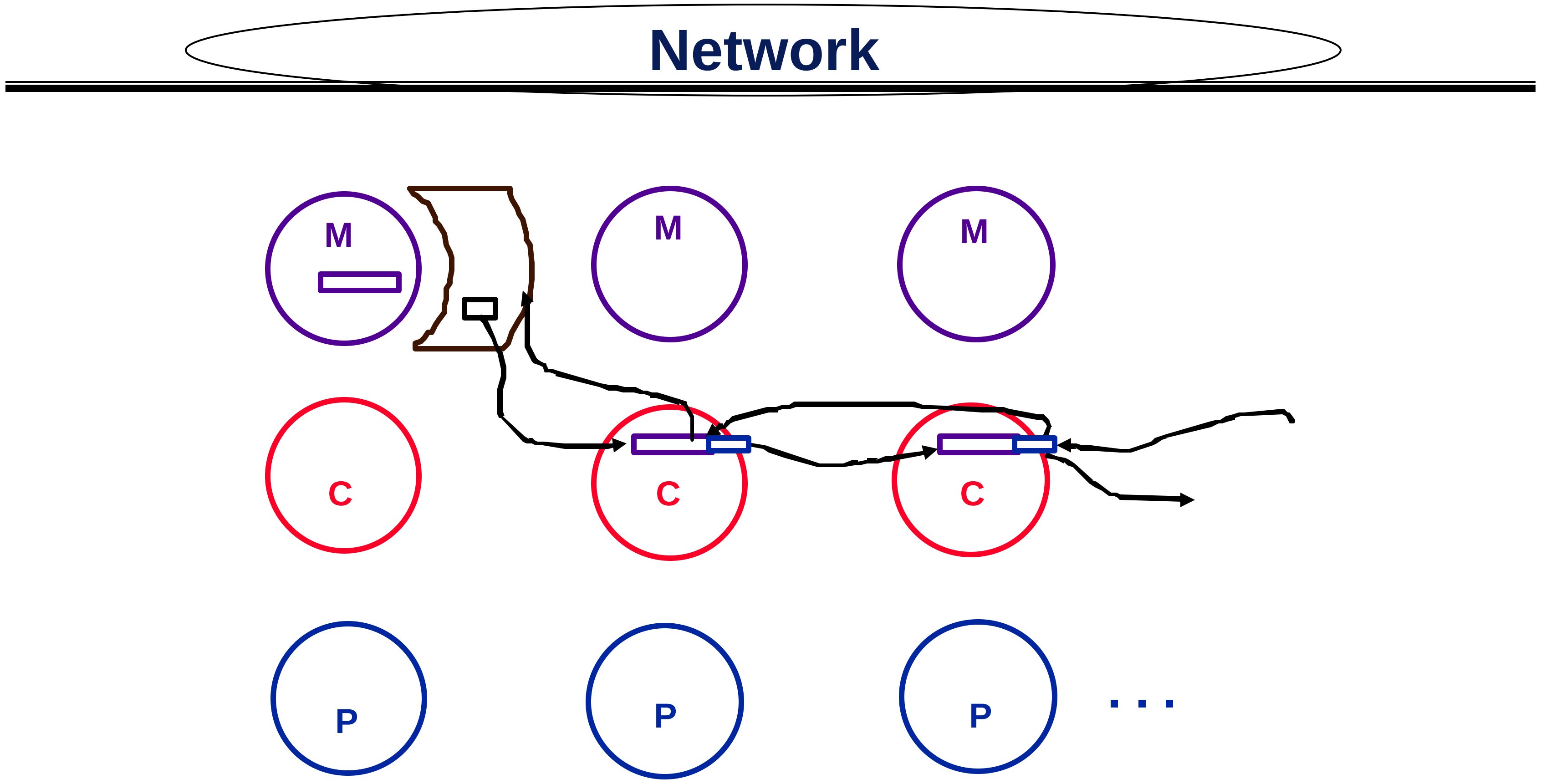
  - **Trap processor when 5th request comes**
  - **Processor extends directory into local memory**

Zero pointer LimitLESS: All software coherence



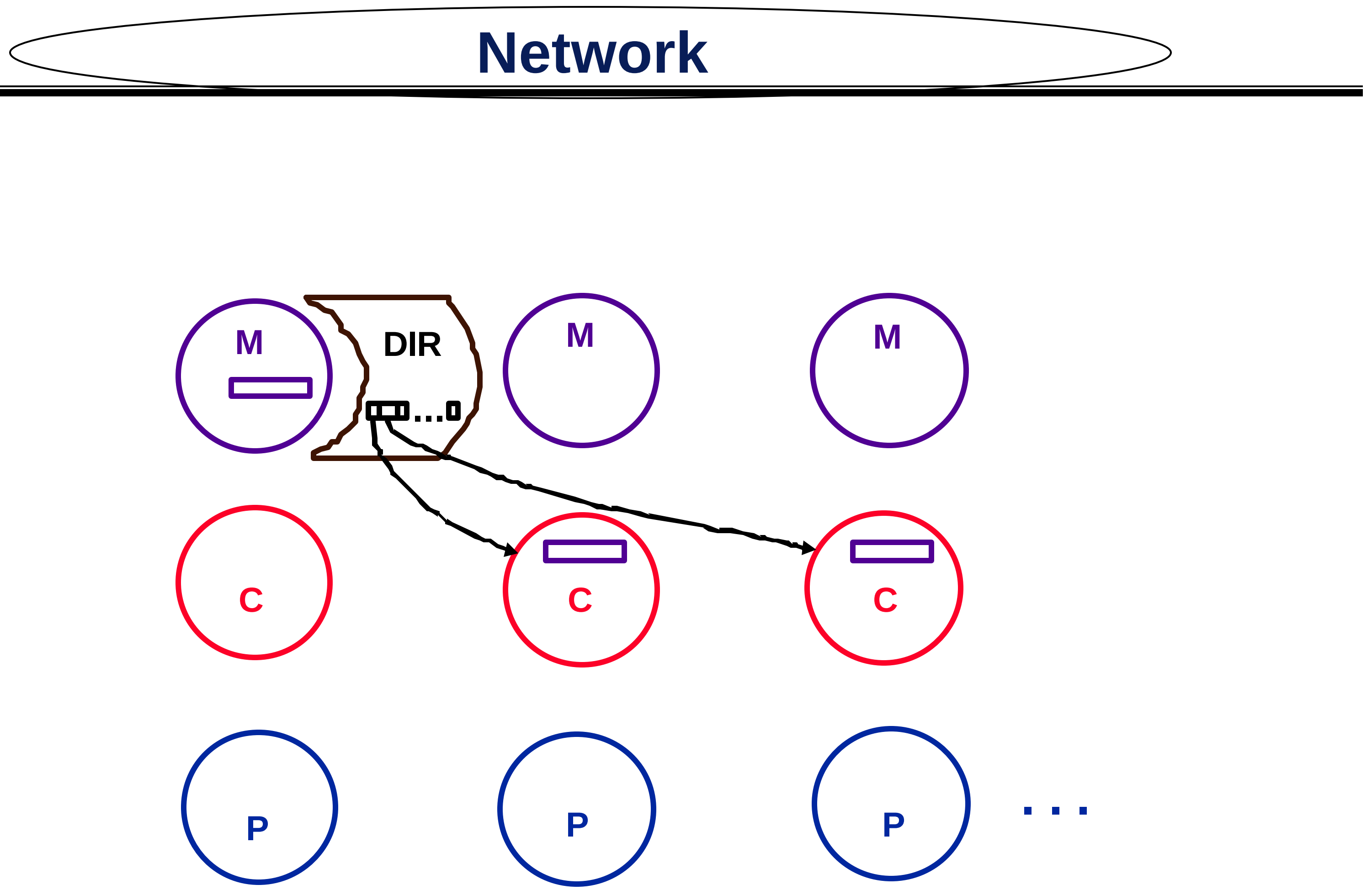


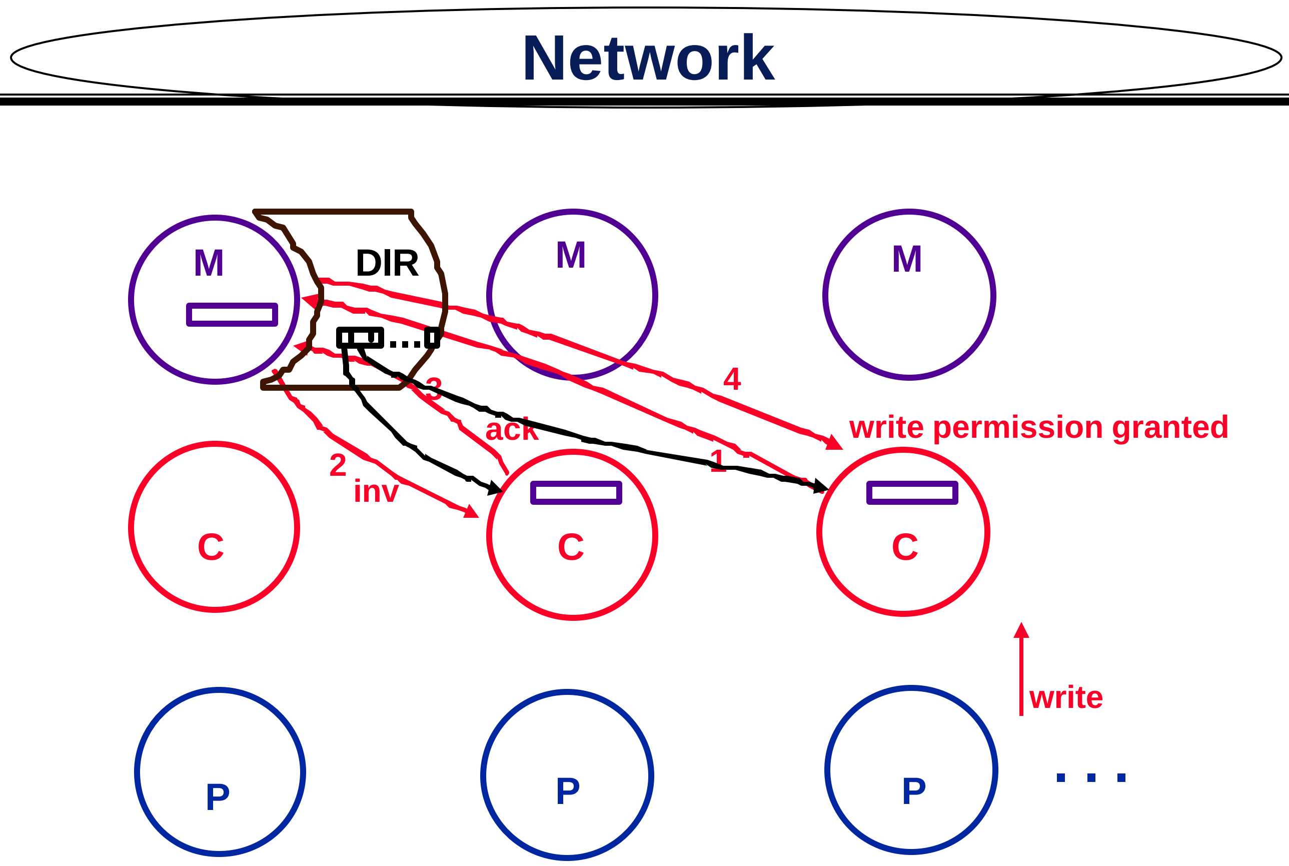
- **Chained directories: Simply different data structure for directory**
  - **Link all cache entries**
  - **But longer latencies**
  - **Also more complex hardware**
  - **Must handle replacements of elements in chain due to misses!**



Doubly linked chains

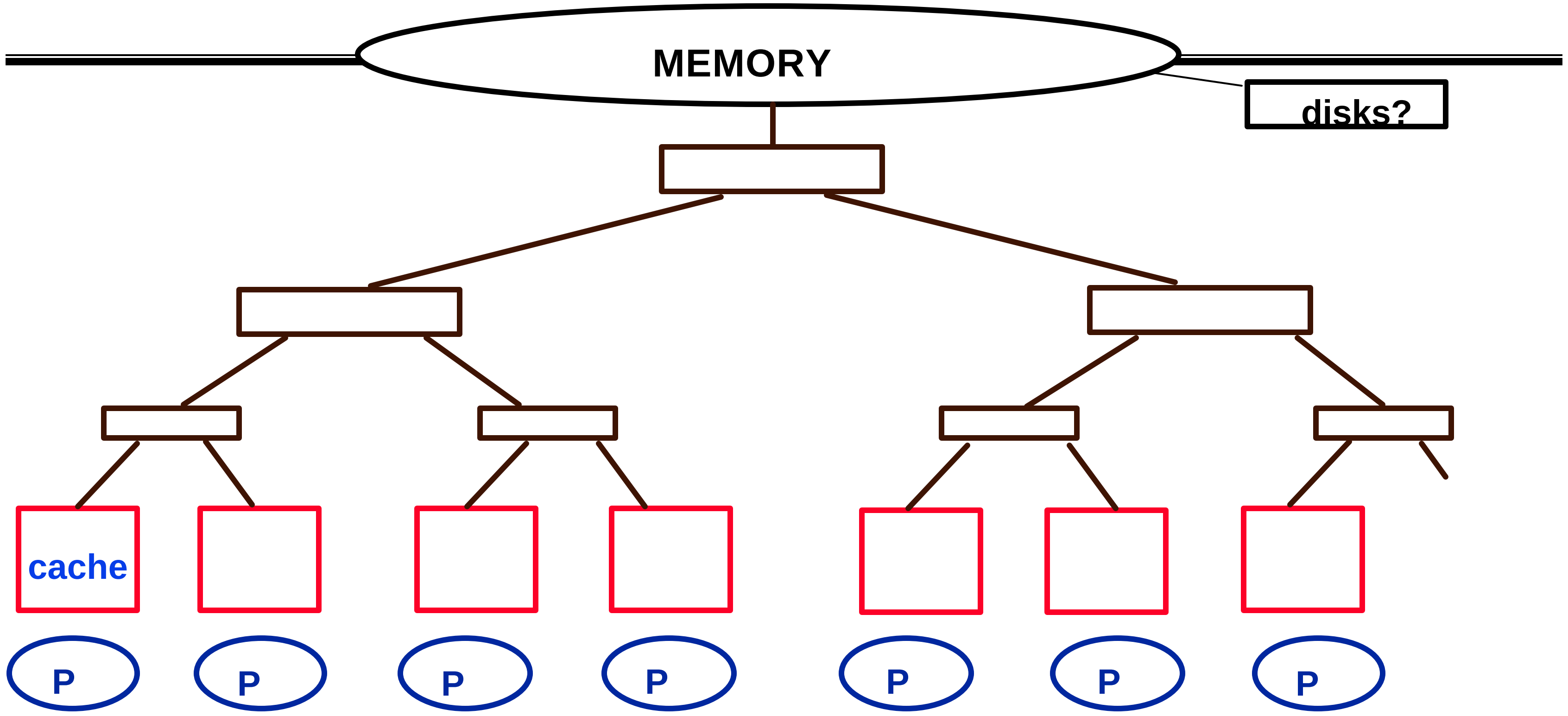
Of course, can do these data structures though software + msgs as in LimitLESS


Network
M
DIR
M
M
C
C
C
P
P
P
. . .

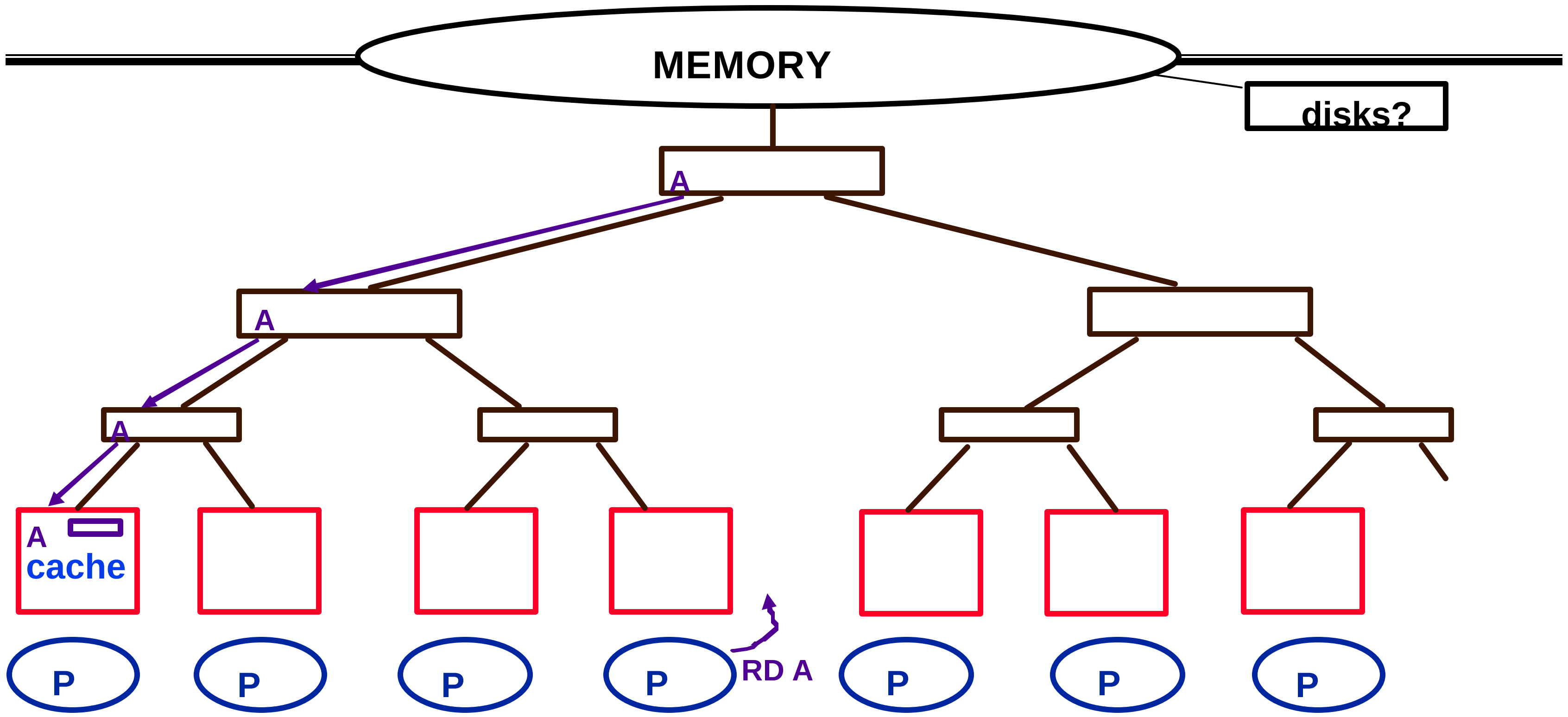


Full map: Problem

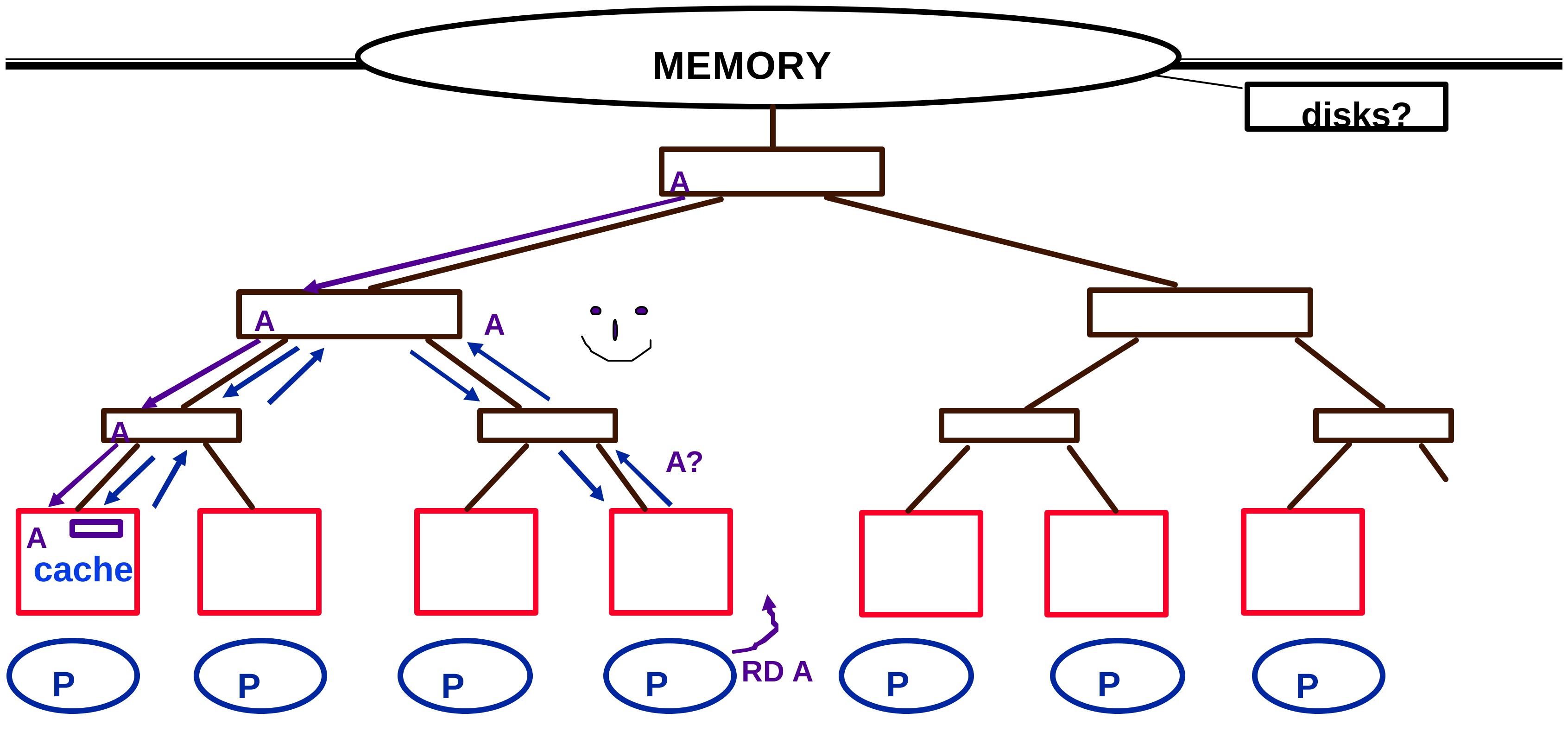
- Does not scale -- need N pointers
- Directories distributed, so not a bottleneck



- Hierarchical - E.g. KSR (actually has rings...)



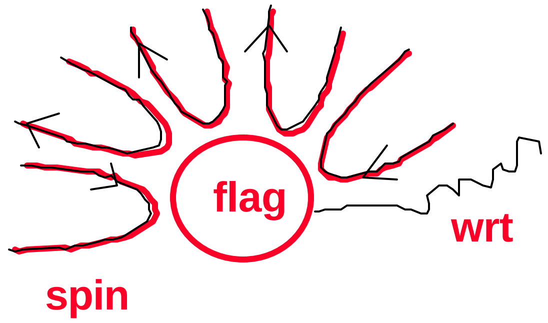
Hierarchical - E.g. KSR (actually has rings...)

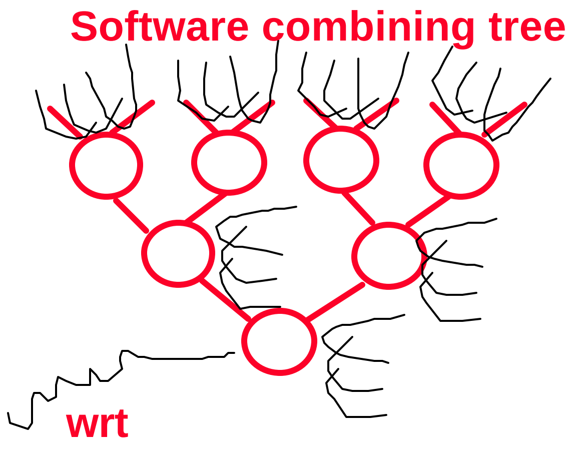


Hierarchical - E.g. KSR (actually has rings...)

Widely shared data

- 1. Synchronization variables





- 2. Read only objects

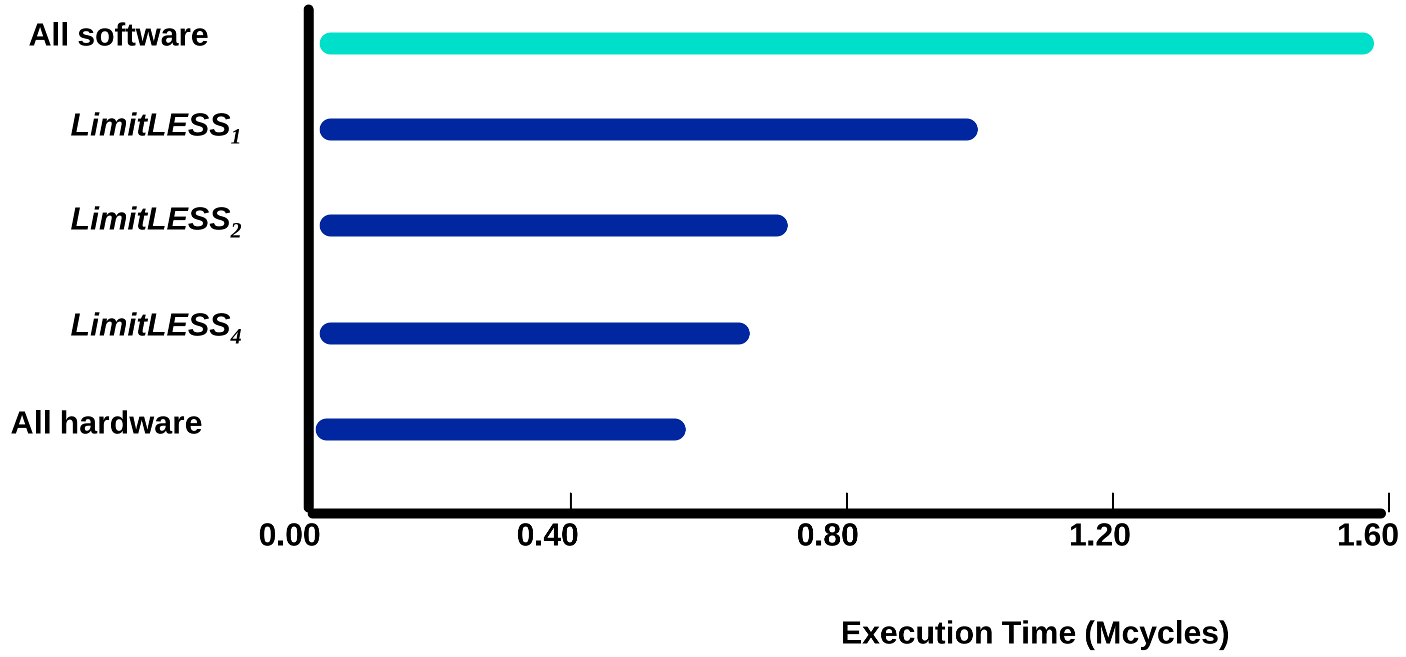
  Instructions
  Read-only data

  Can mark these and bypass coherence protocol

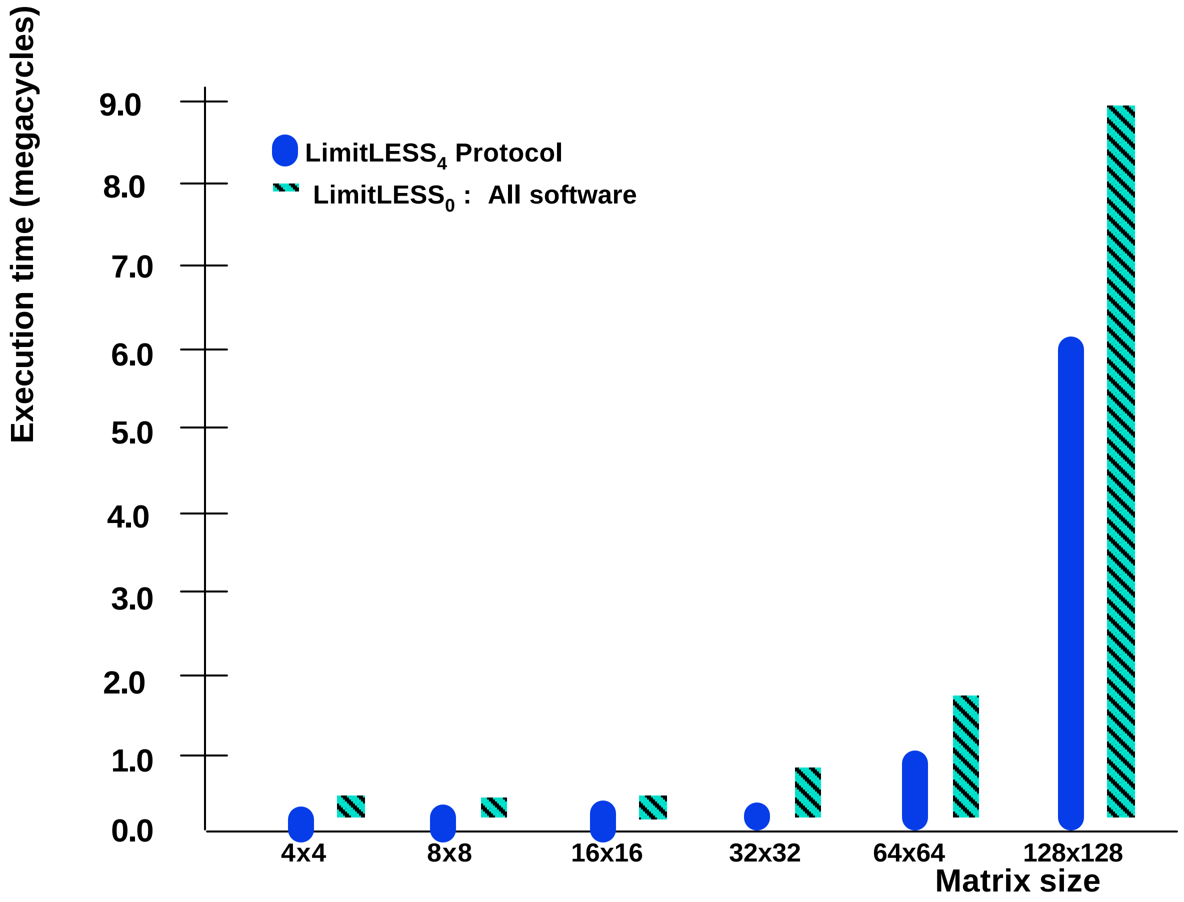
- 3. But, biggest problem:

  Frequently read, but rarely written data which does not fall into known patterns like synchronization variables

All software coherence

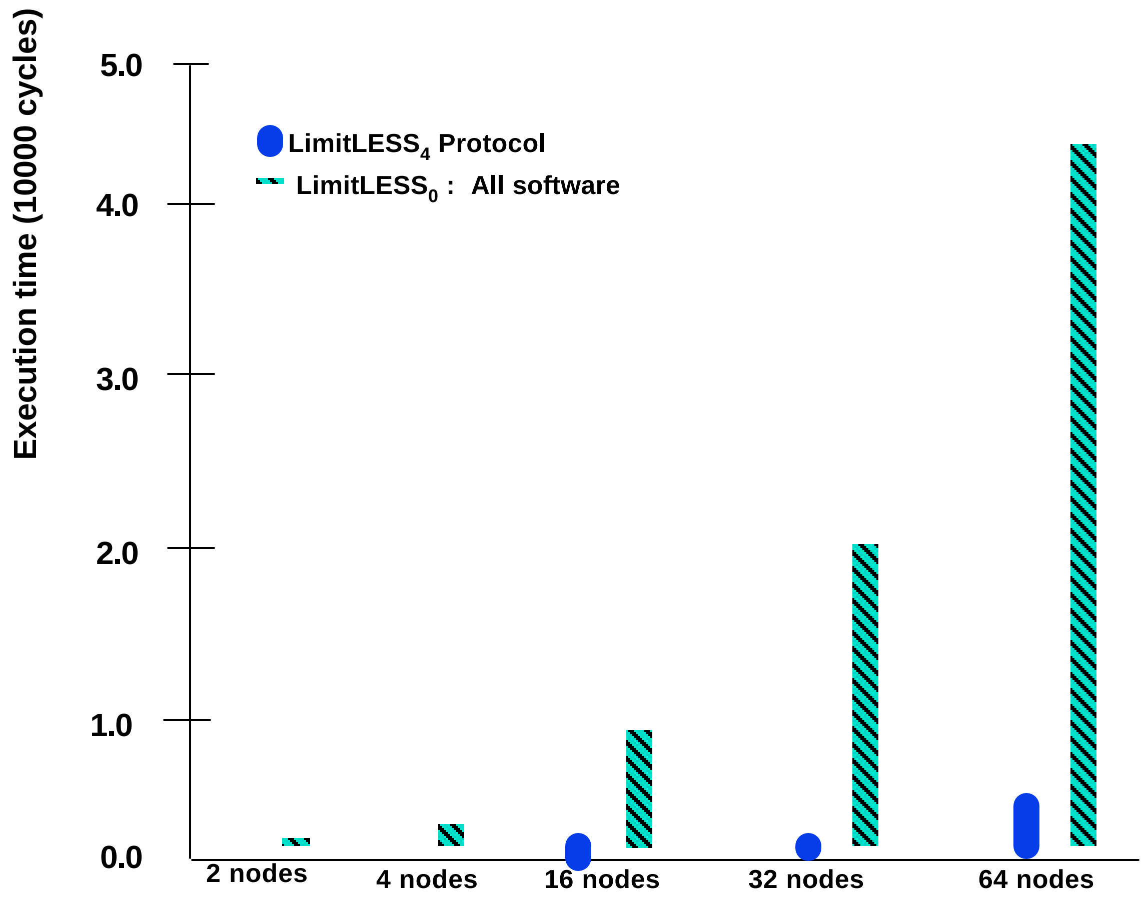


All software coherence



Cycle Performance for Block Matrix Multiplication, 16 processors

All software coherence



Performance for Single Global Barrier (first INVR to last RDATA)

Summary

- Tradeoffs in caching an important issue
- Limitless protocol provides software extension to hardware caching
- Goal: maintain coherence but minimize network traffic
- Full map not scalable and too costly
- Distributed memory makes caching more of a challenge