Parallel Computing

Shared-Memory Systems

- Parallel computing
- Parallel architecture
- Abstraction models
- Centralized shared-memory architecture
    - Snoopy coherence protocols
    - Performance of symmetric shared-memory multiprocessors
- Distributed shared-memory systems
    - Directory-based protocols
- Synchronization
- Memory consistency

ENGINEERING@SYRACUSE

Parallel Computing

- Application demands: our insatiable need for computing cycles
  - Scientific computing: biology, chemistry, physics, …
  - General-purpose computing: video, graphics, CAD, databases, …
- Technology trends
  - Number of transistors on chip growing rapidly.
  - Clock rates expected to go up only slowly.
- Architecture trends
  - Instruction-level and thread-level parallelism valuable but limited.
- Economics
- Current trends
  - Today’s microprocessors are multiprocessors.
  - Servers and workstations becoming MP.

Parallel Applications

- Predictive modeling and simulations:
  - Numerical weather forecasting
  - Oceanography and astrophysics
  - Socioeconomics
- Engineering design and automation:
  - Finite-element structural analysis
  - Computational aerodynamics
  - AI and expert systems
  - CAD/CAM
- Energy resources exploration:
  - Seismic exploration
  - Oil field modeling
  - Plasma fusion, nuclear-energy research
- Medical, military, and basic research:
  - Tomography
  - Genetic engineering
  - Weapons research
  - Basic research (VLSI IC analysis, polymer chemistry, quantum mechanics)

Commercial Computing

- Also relies on parallelism for high end
  - Scale not so large, but use much more widespread.
  - Computational power determines scale of business that can be handled.
- Databases, online-transaction processing, decision support, data mining, data warehousing
- TPC benchmarks
  - Explicit scaling criteria provided.
  - Size of enterprise scales with size of system.
  - Problem size no longer fixed as p increases, so throughput is used as a performance measure (transactions per minute: tpm).

Algorithms for Vectorizations

- Vector/matrix arithmetic
- Matrix multiplication, decomposition, conversion of matrices, eigenvalue computations, sparse matrix computations, least square problems
- Signal/image processing:
  - Convolution and correlation, digital filtering, fast Fourier transformation, pattern recognition, scene analysis, and vision
- Optimization processes:
  - Linear programming, sorting and searching, integer programming, branch and bound algorithms, combinatorial analysis, constrained optimization
- Statistical analysis:
  - Probability distribution functions, variance analysis, nonparametric statistics, multivariate statistics, sampling, and histogramming
- Partial differential equations:
  - Ordinary differential equations, partial differential equations, finite-element analysis, domain decompositions, numerical integrations
- Special functions and graph algorithms:
  - Power series and functions, interpolation and approximation, searching techniques, graph matching, logic set operations, transitive closures

Other Apps at Smaller Scale

- Limited to scientific computing?
  - Multimedia processing (compression, graphics, audio synth, image processing)
  - Standard benchmark kernels (matrix multiply, FFT, convolution, sort)
  - Lossy Compression (JPEG, MPEG video and audio)
  - Lossless compression (zero removal, RLE, differencing, LZW)
  - Cryptography (RSA, DES/IDEA, SHA/MD5)
  - Speech and handwriting recognition
  - Operating systems/networking (parity, checksum)
  - Databases (hash/join, data mining, image/video serving)
  - Language runtime support (stdlib)

Application Trends

- Demand for cycles fuels advances in hardware and vice versa.
  - Cycle drives exponential increase in microprocessor performance.
  - Drives parallel architecture harder: most demanding applications.
- Goal of applications in using parallel machines: speedup
  - Speedup (p processors) = Perf (p)/Perf (1)
- For a fixed problem size (input dataset), performance = 1/time
  - Speedup fixed problem (p processors) = Time (1)/Time (p)

General Technology Trends

- Microprocessor performance increases 50 to 100 percent per year.
- Transistor count doubles every three years.
- DRAM size quadruples every three years.
- Huge investment per generation is carried by huge commodity market.
- Not that single-processor performance levels off, but parallelism is a natural way to improve it.

ENGINEERING@SYRACUSE

Parallel Architectures

Architecture Trends

- Greatest trend in VLSI generation is increase in parallelism.
- Up to 1985: bit-level parallelism: 4-bit → 8-bit → 16-bit
  - Slows after 32 bit .
  - 64-bit has now been adopted, 128-bit far.
  - Great inflection point when 32-bit micro and cache fit on a chip.
- Mid-’80s to mid-’90s: instruction-level parallelism
  - Pipelining and simple instruction sets + compiler advances (RISC)
  - On-chip caches and functional units => superscalar execution
  - Greater sophistication: out-of-order execution, speculation, prediction
    - To deal with control transfer and latency problems
- After 2000: thread-level parallelism
- Now: multicores

Economics

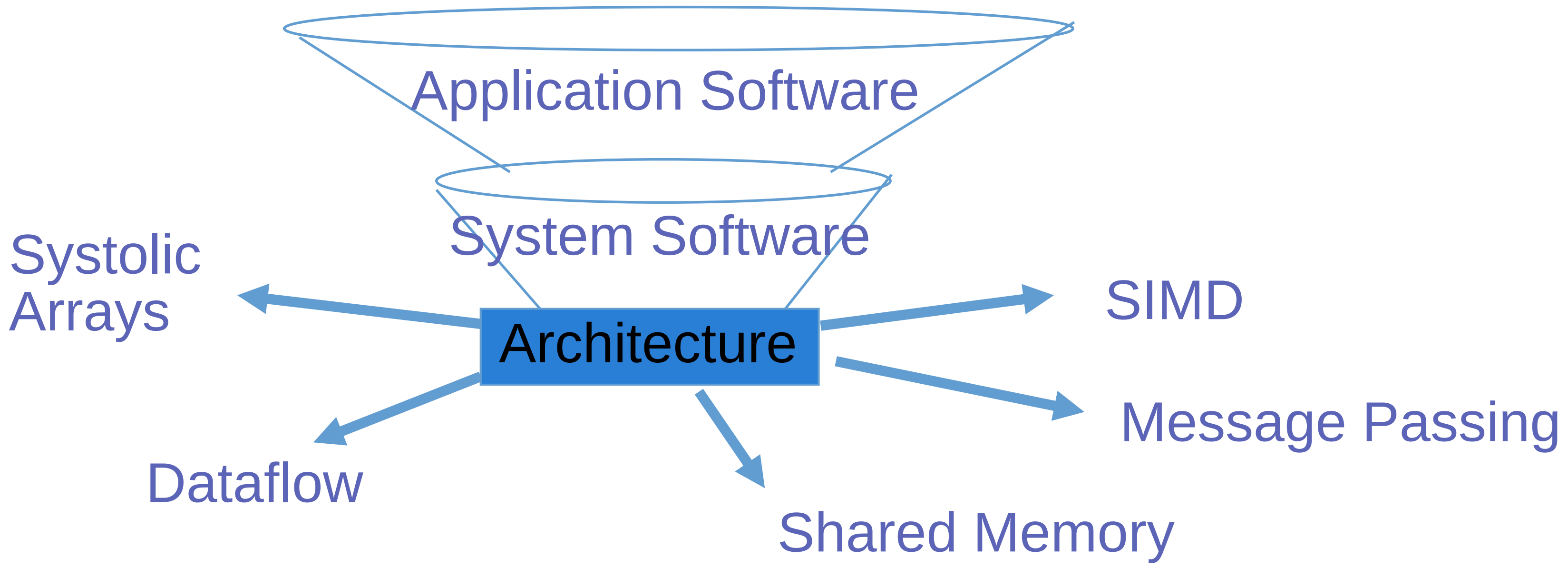
- Commodity microprocessors not only fast but CHEAP.
  - Development cost is tens of millions of dollars (5 to 100 typical).
  - Many more are sold compared to supercomputers.
  - Crucial to take advantage of the investment and use the commodity building block.
  - Exotic parallel architectures no more than special purpose.
- Multiprocessors being pushed by software vendors (e.g., database) as well as hardware vendors.
- Standardization by Intel makes small, bus-based SMPs commodity.
- Desktop: few smaller processors vs. one larger one?
  - Multiprocessor on a chip

What Is a Parallel Architecture?

- A parallel computer is a collection of processing elements that cooperate to solve large problems fast.
- Some broad issues:
  - Resource allocation:
    - How large a collection?
    - How powerful are the elements?
    - How much memory?
  - Data access, communication, and synchronization
    - How do the elements cooperate and communicate?
    - How are data transmitted between processors?
    - What are the abstractions and primitives for cooperation?
  - Performance and scalability
    - How does it all translate into performance?
    - How does it scale?

Convergence of a Parallel Architecture

- Parallel architectures tied to programming models.
- Divergent architectures, with no predictable pattern of growth.



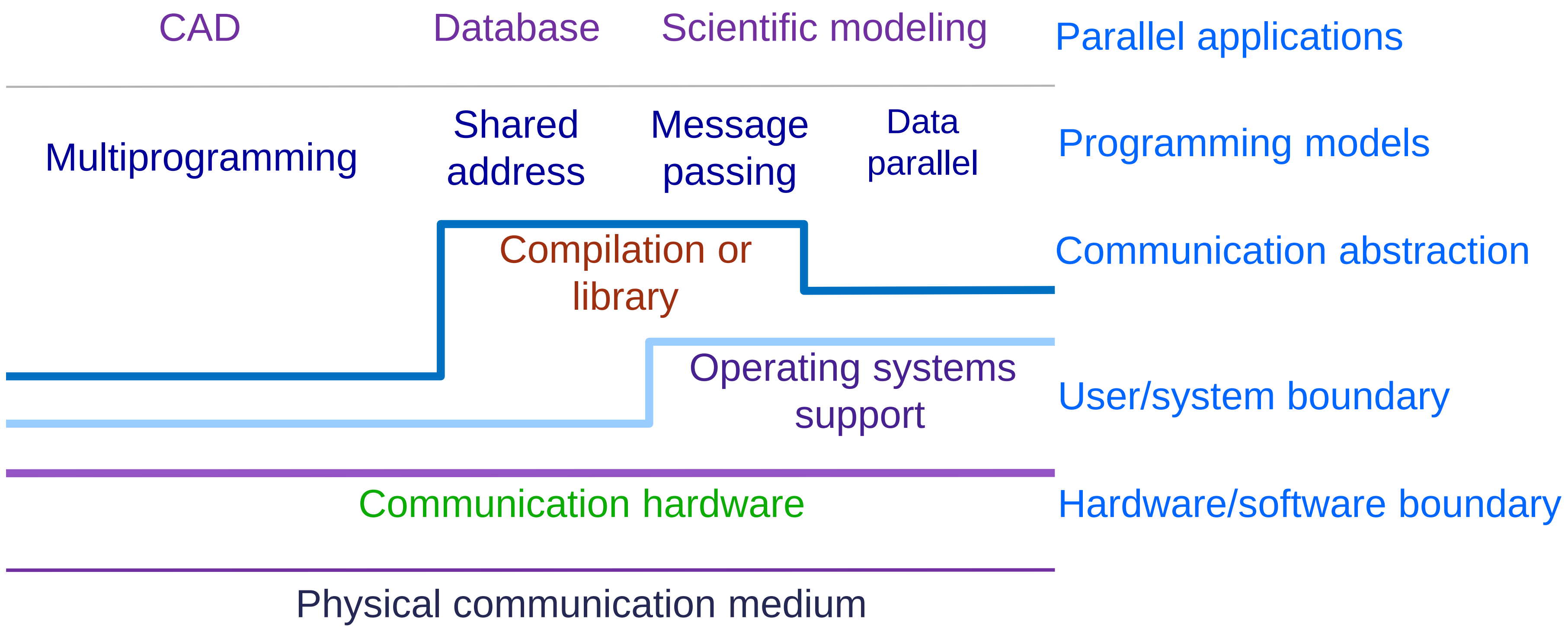
Today

- Extension of “computer architecture” to support communication and cooperation.
  - OLD: instruction set architecture
  - NEW: communication architecture
- Defines:
  - Critical abstractions, boundaries, and primitives (interfaces)
  - Organizational structures that implement interfaces (hardware or software)
- Compilers, libraries, and OS are important bridges today.

ENGINEERING@SYRACUSE

Abstraction Models

Layers


CAD
Database
Scientific modeling
Parallel applications
Multiprogramming
Shared address
Message passing
Data parallel
Programming models
Compilation or library
Communication abstraction
Operating systems support
User/system boundary
Communication hardware
Hardware/software boundary
Physical communication medium

Programming Model

- Specifies communication and synchronization.
  - Multiprogramming: no communication or synchronization at program level
  - Shared address space: like bulletin board
  - Message passing: like letters or phone calls, explicit point to point
  - Data parallel: more regimented, global actions on data
    - Implemented with shared address space or message passing

Communication Model

- User-level communication primitives:
  - Realizes the programming model.
  - Mapping exists between language primitives of programming model and these primitives.
- Supported directly by HW, or OS, or via user SW.
- Today:
  - Compilers and software play important roles as bridges today.
  - Technology trends exert strong influence.
- Result is convergence in organizational structure.
  - Relatively simple, general-purpose communication primitives

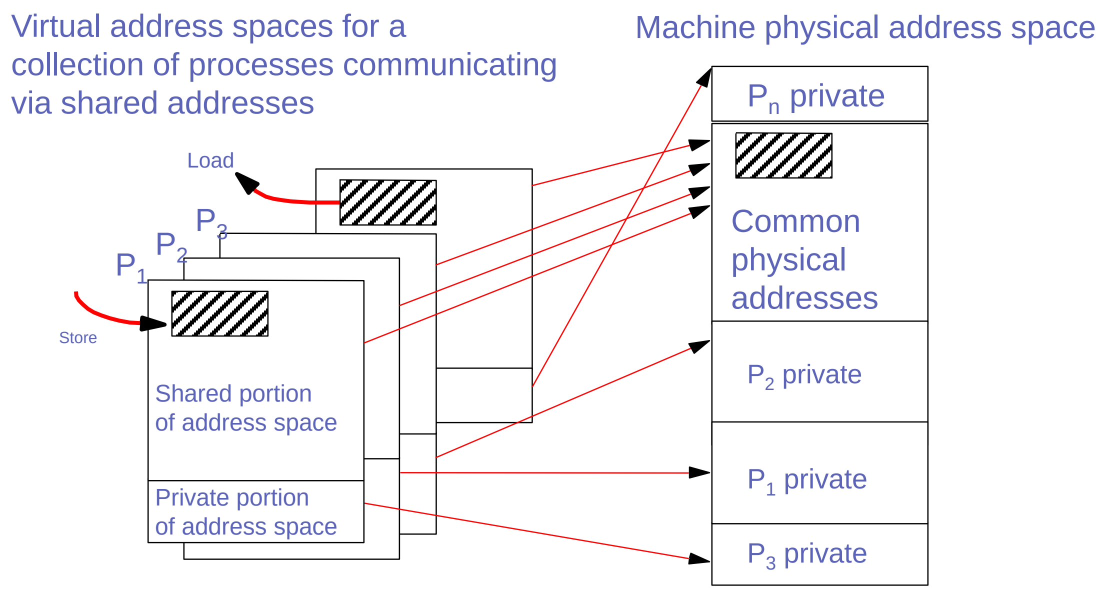
Communication Architecture

= user/system interface + implementation

- Communication primitives exposed to user-level by HW and system-level SW.
- Organizational structures that implement the primitives: HW or OS.
- Structure of network.
- Goals:
  - Performance
  - Broad applicability
  - Programmability
  - Scalability
  - Low cost

Shared Address Space Model

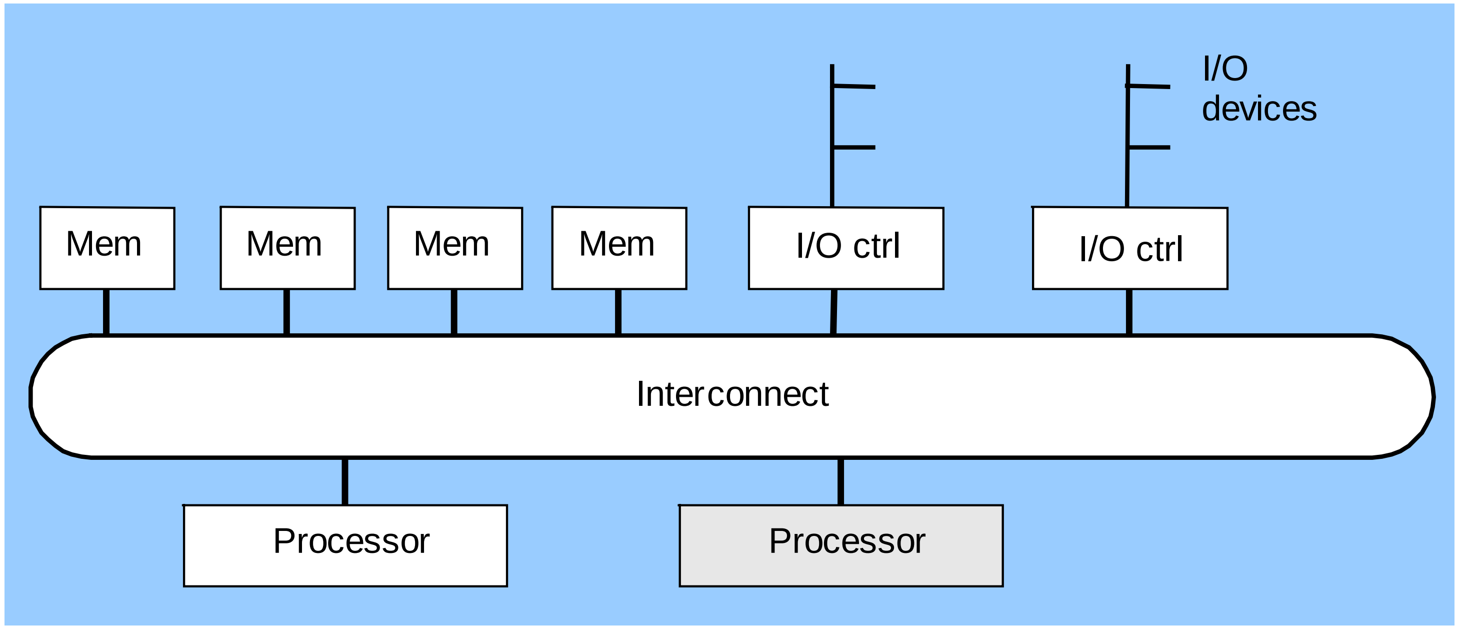
- Process: virtual address space plus one or more threads of control.
- Portions of address spaces of processes are shared.


Virtual address spaces for a collection of processes communicating via shared addresses
Machine physical address space
Load
Store
P_1
P_2
P_3
Shared portion of address space
Private portion of address space
P_n private
Common physical addresses
P_2 private
P_1 private
P_3 private

- Writes to shared address visible to other threads.
- Natural extension of uniprocessors model: conventional memory operations for communication; special atomic operations for synchronization.
- OS uses shared memory to coordinate processes.

ENGINEERING@SYRACUSE

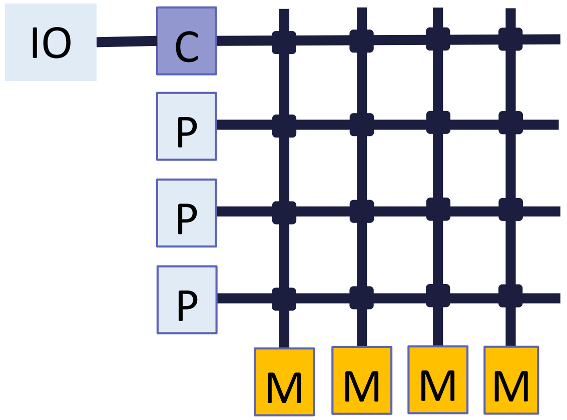
Communication at SAS

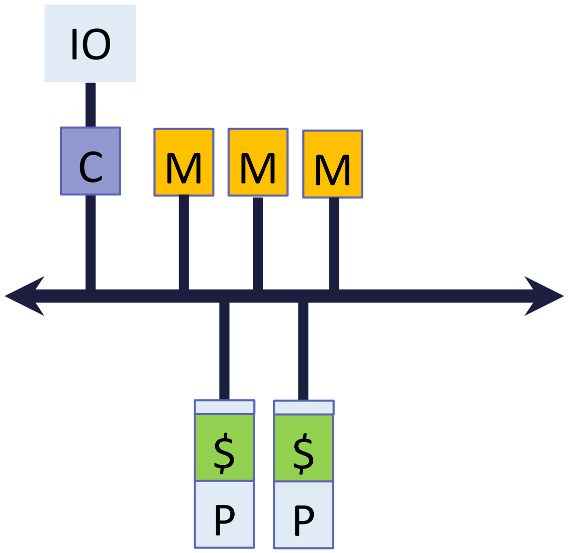


- Memory capacity increased by adding modules, I/O by controllers.
- Add processors for processing!
- For higher-throughput multiprogramming or parallel programs.

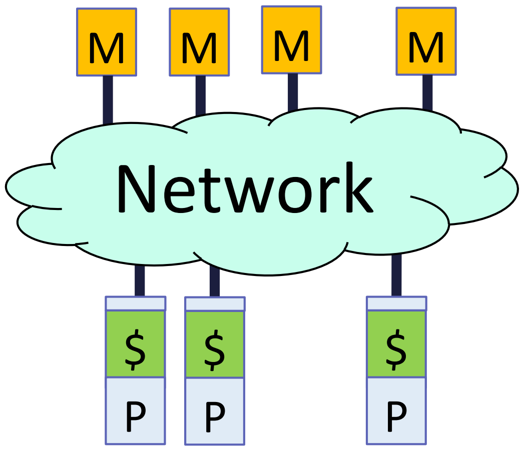
History

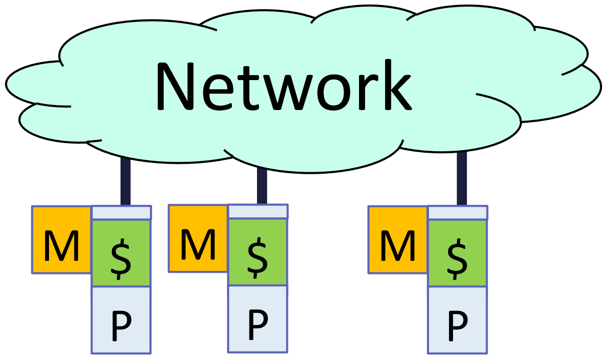
- "Mainframe" approach
  - Motivated by multiprogramming.
  - Extends crossbar used for mem BW and I/O.
  - Originally processor cost limited to small.
  - Later, cost of crossbar.
  - Bandwidth scales with p.
  - High incremental cost; use multistage instead.
- "Minicomputer" approach
  - Almost all microprocessor systems have bus.
  - Motivated by multiprogramming, TP.
  - Used heavily for parallel computing.
  - Called symmetric multiprocessor (SMP).
  - Latency larger than for uniprocessor.
  - Bus is bandwidth bottleneck.
    - Caching is key: coherence problem.
  - Low incremental cost.





Scaling Up



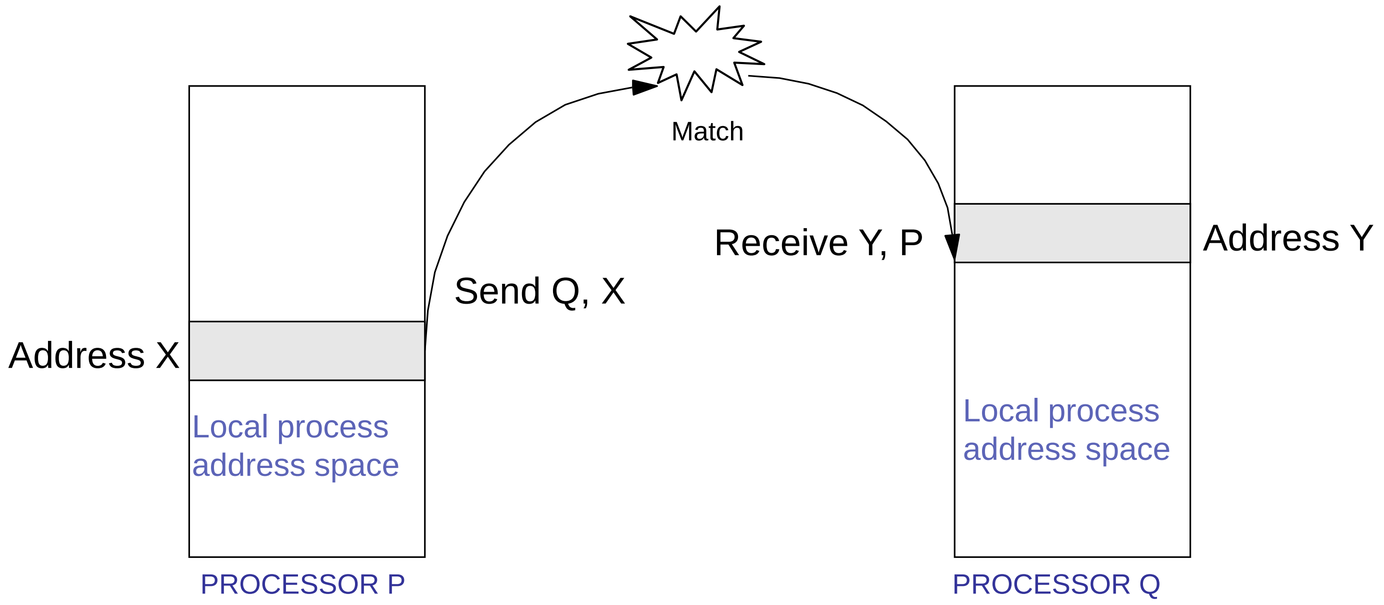


- Interconnect: cost (crossbar) or bandwidth (bus)
- Dance hall: bandwidth still scalable, lower cost than crossbar
  - Latencies to memory uniform, but large
- Distributed memory or nonuniform memory access (NUMA)
  - Construct shared address space out of simple message transactions across a general-purpose network (REQUEST/RESPONSE).
- Caching shared (particularly nonlocal) data?

Message-Passing Model

- Complete computer as building block, including I/O.
  - Communication via explicit I/O operations
- Programming model: directly access only private address space (local memory), communication via explicit messages (send/receive).
- High-level block diagram similar to distributed-memory SAS.
  - But communication integrated at IO level, needn’t be into memory system
  - Like networks of workstations (clusters), but tighter integration
  - Easier to build than scalable SAS
- Programming model removed from basic hardware operations.
  - Library or OS intervention

Message-Passing Abstraction



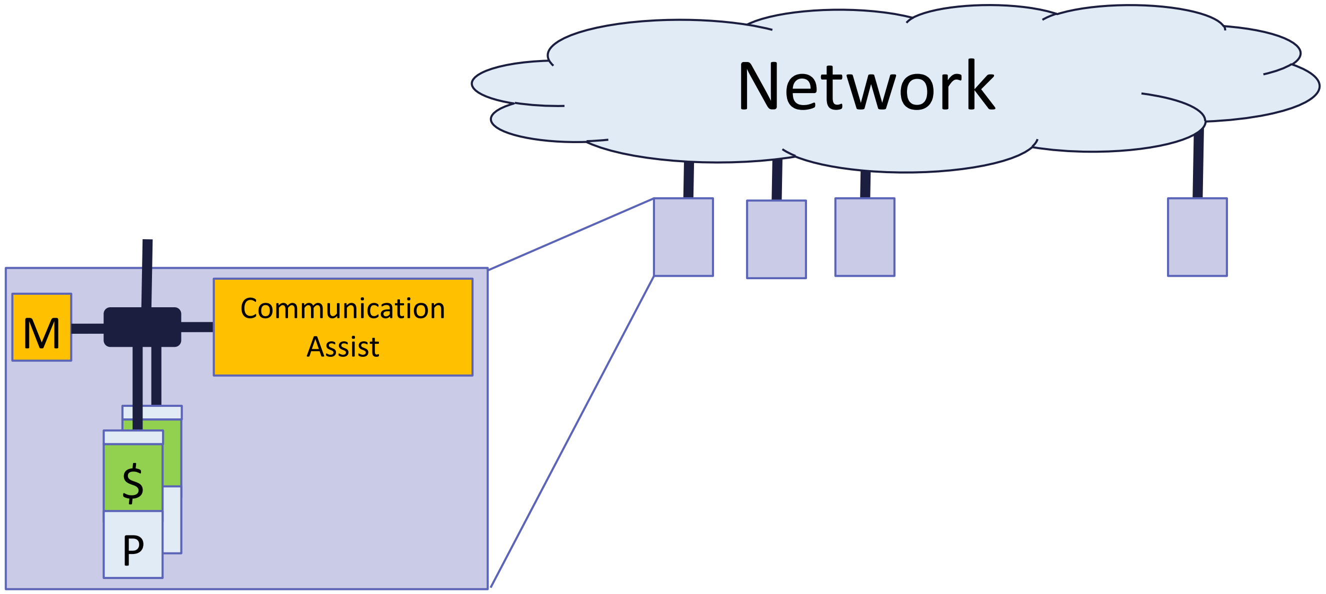
- Send specifies buffer to be transmitted and receiving process.
- Receive specifies sending process and application storage to receive into.
- Memory-to-memory copy, but need to name processes.
- User process names local data and entities in process/tag space too.
- In simplest form, the send/receive match achieves pairwise synchronous event.
- Many overheads: copying, buffer management, protection.

Toward Architectural Convergence

- Evolution and role of software have blurred boundary.
  - Send/receive supported on SAS machines via buffers.
  - Can construct global address space on MP using hashing.
  - Page-based (or finer-grained) shared virtual memory.
- Hardware organization converging, too.
  - Tighter NI integration even for MP (low-latency, high-bandwidth).
  - At lower level, even hardware SAS passes hardware messages.
- Even clusters of workstations/SMPs are parallel systems.
  - Emergence of fast system area networks (SAN)
- Programming models distinct, but organizations converging.
  - Nodes connected by general network and communication assists.
  - Implementations also converging, at least in high-end machines.

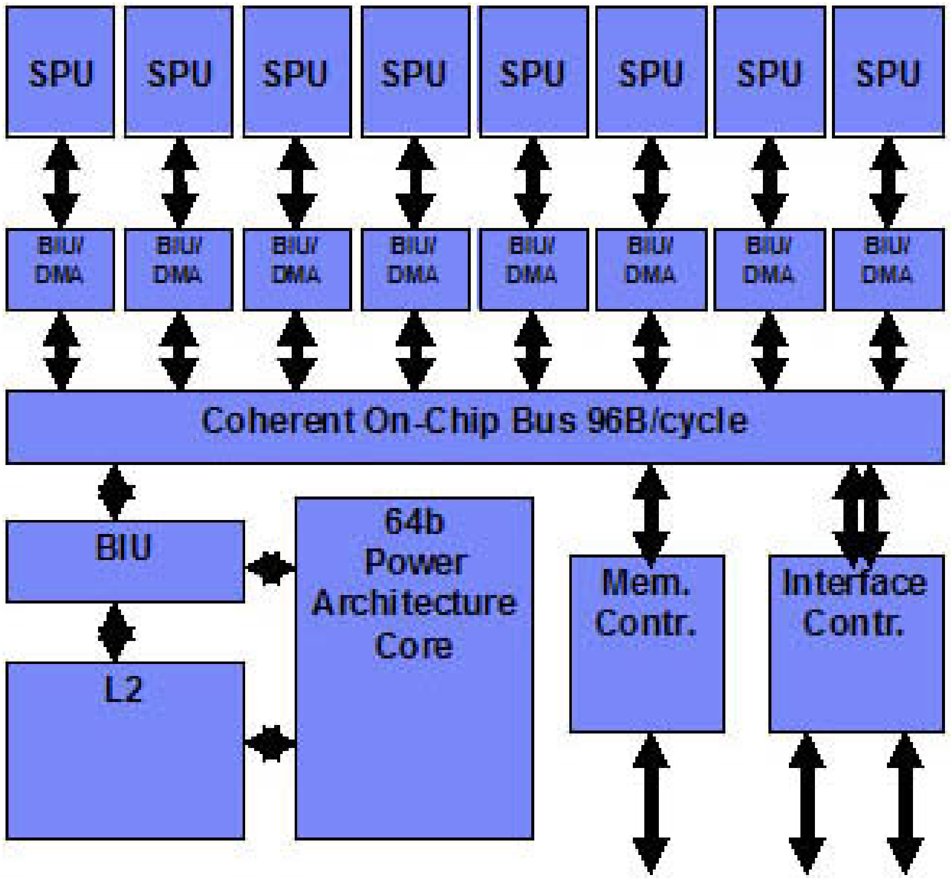
Generic Multiprocessor

- Node: processor(s), memory system, plus communication assist.
- Network interface and communication controller.
- Scalable network.
- Convergence allows lots of innovation, now within framework.
- Integration of assist with node, what operations, how efficiently...



STI Cell: Multiprocessor on Chip

- Low power and cost, high performance
- Cryptography, graphics, physics, FFT, matrix operations, and scientific workloads
- Heterogeneous chip multiprocessor 64-bit
- Eight specialized SIMD co-processors: synergistic processor unit (SPU)
- Clock speed: > 4 GHz
- Peak performance (single precision): > 256 GFlops
- Local storage size per SPU: 256 KB
- Area: 221 mm²
- Technology 90 nm SOI
- Total number of transistors: 234 M
- Three to 12 times faster than any desktop processor, including Itanium 2



Multithreads to Multiprocessors

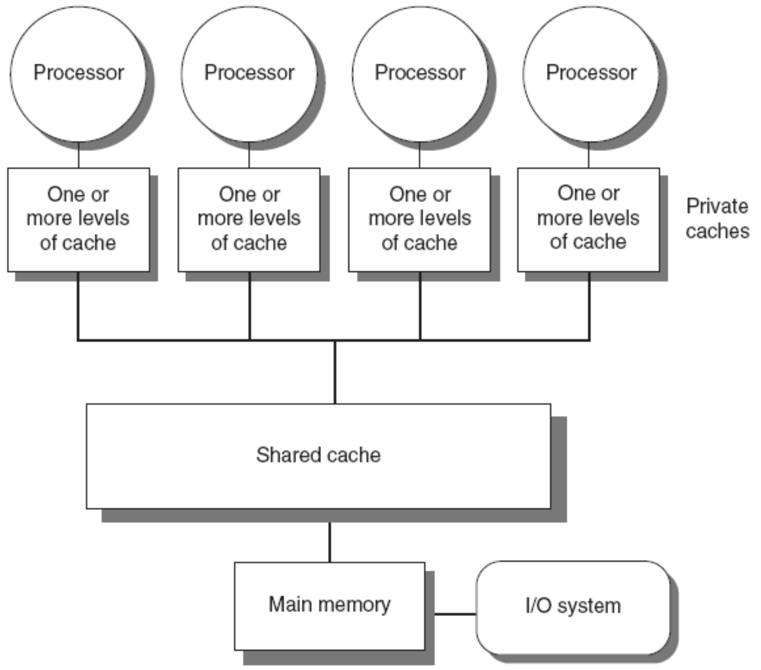
- Thread-level parallelism:
  - Have multiple program counters
  - Uses MIMD model
  - Targeted for tightly coupled shared-memory multiprocessors
- For *n* processors, need *n* threads.
- Amount of computation assigned to each thread = grain size.
  - Threads can be used for data-level parallelism, but the overheads may outweigh the benefit.

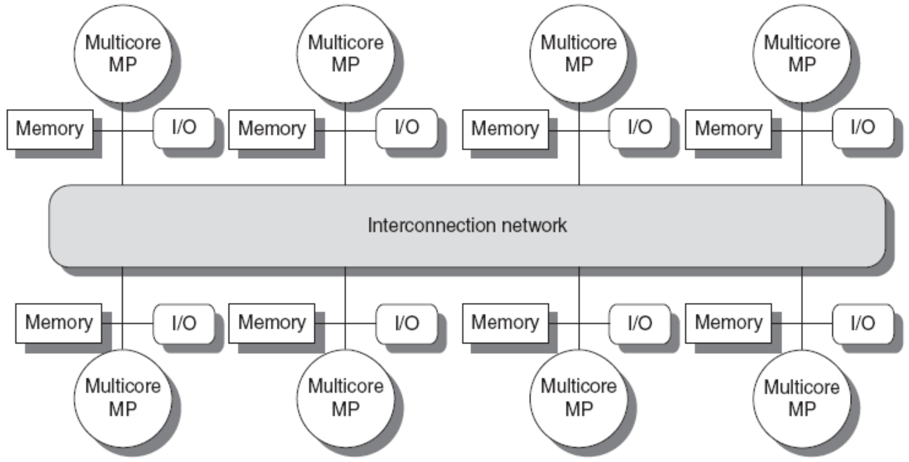
ENGINEERING@SYRACUSE

Centralized Shared-Memory Architectures

Types

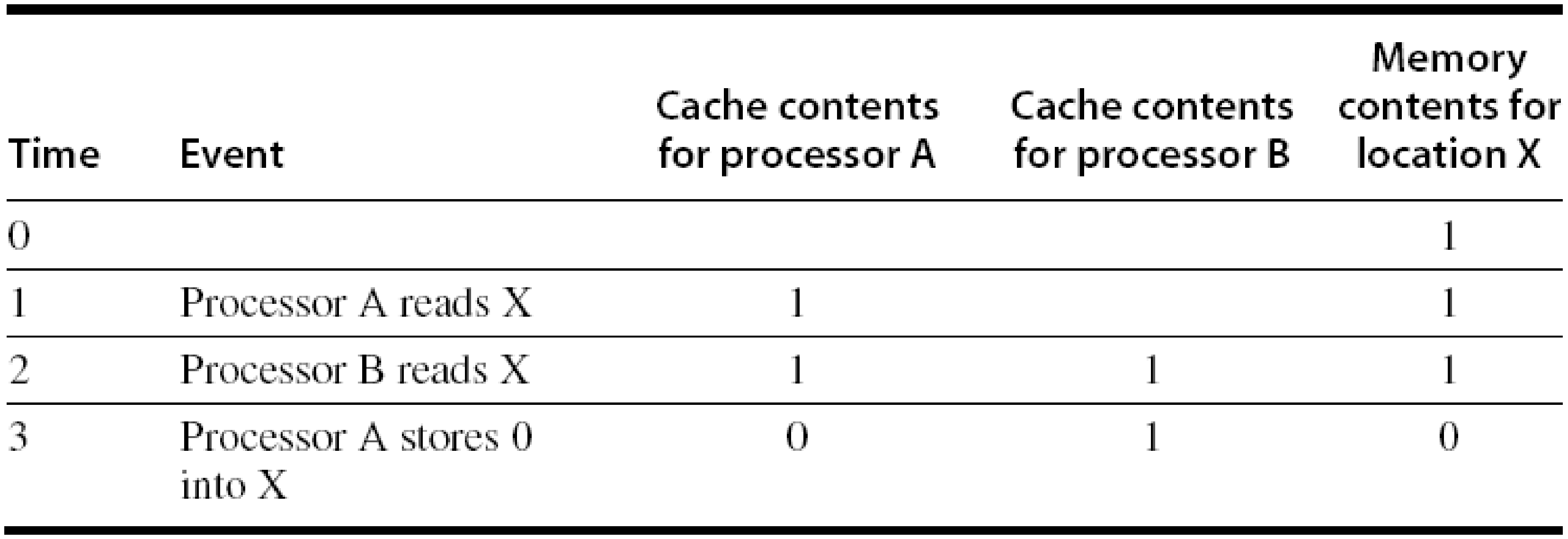
- Symmetric multiprocessors (SMP)
  - Small number of cores
  - Share single memory with uniform memory latency
- Distributed shared memory (DSM)
  - Memory distributed among processors
  - Nonuniform memory access/latency (NUMA)
  - Processors connected via direct (switched) and nondirect (multihop) interconnection networks


Processor
Processor
Processor
Processor
One or more levels of cache
One or more levels of cache
One or more levels of cache
One or more levels of cache
Private caches
Shared cache
Main memory
I/O system



Cache Coherence

- Processors may see different values through their caches.

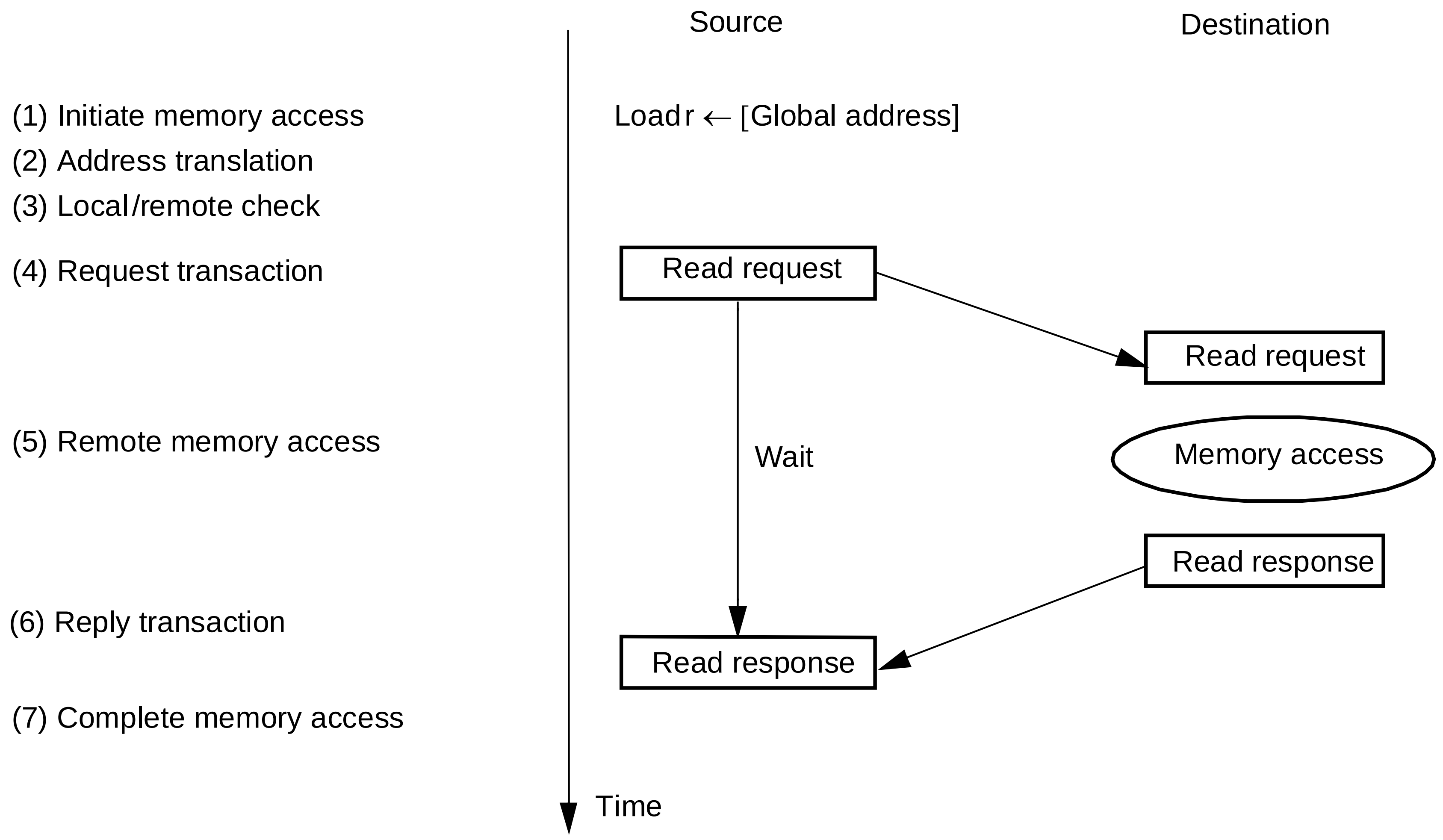


Time	Event	Cache contents for processor A	Cache contents for processor B	Memory contents for location X
0				1
1	Processor A reads X	1		1
2	Processor B reads X	1	1	1
3	Processor A stores 0 into X	0	1	0

Cache Coherence (cont.)

- Coherence
  - All reads by any processor must return the most recently written value.
  - Writes to the same location by any two processors are seen in the same order by all processors.
- Consistency
  - When a written value will be returned by a read.
  - If a processor writes location A followed by location B, any processor that sees the new value of B must also see the new value of A.

Shared Address Space



Enforcing Coherence

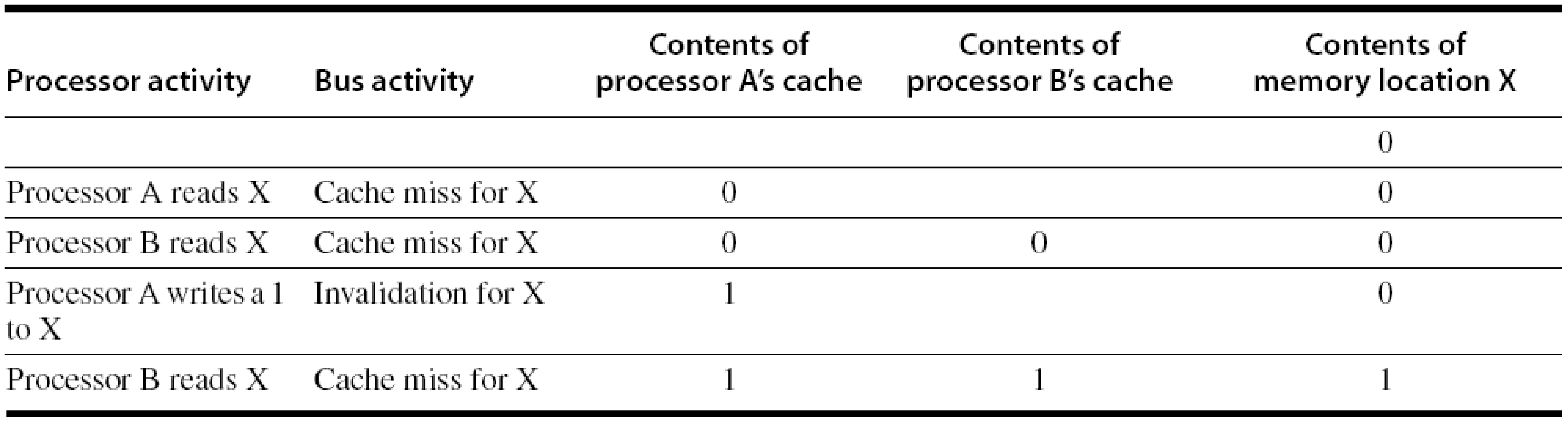
- Coherent caches provide:
  - *Migration*: movement of data
  - *Replication*: multiple copies of data
- Cache coherence protocols
  - Directory-based
    - Sharing status of each block kept in one location.
  - Snooping
    - Each core tracks sharing status of each block.

ENGINEERING@SYRACUSE

Snoopy Coherence Protocols

Snoopy Coherence Protocols

- Write invalidate
  - On write, invalidate all other copies.
  - Use bus itself to serialize.
    - Write cannot complete until bus access is obtained.



Processor activity	Bus activity	Contents of processor A's cache	Contents of processor B's cache	Contents of memory location X
				0
Processor A reads X	Cache miss for X	0		0
Processor B reads X	Cache miss for X	0	0	0
Processor A writes a 1 to X	Invalidation for X	1		0
Processor B reads X	Cache miss for X	1	1	1

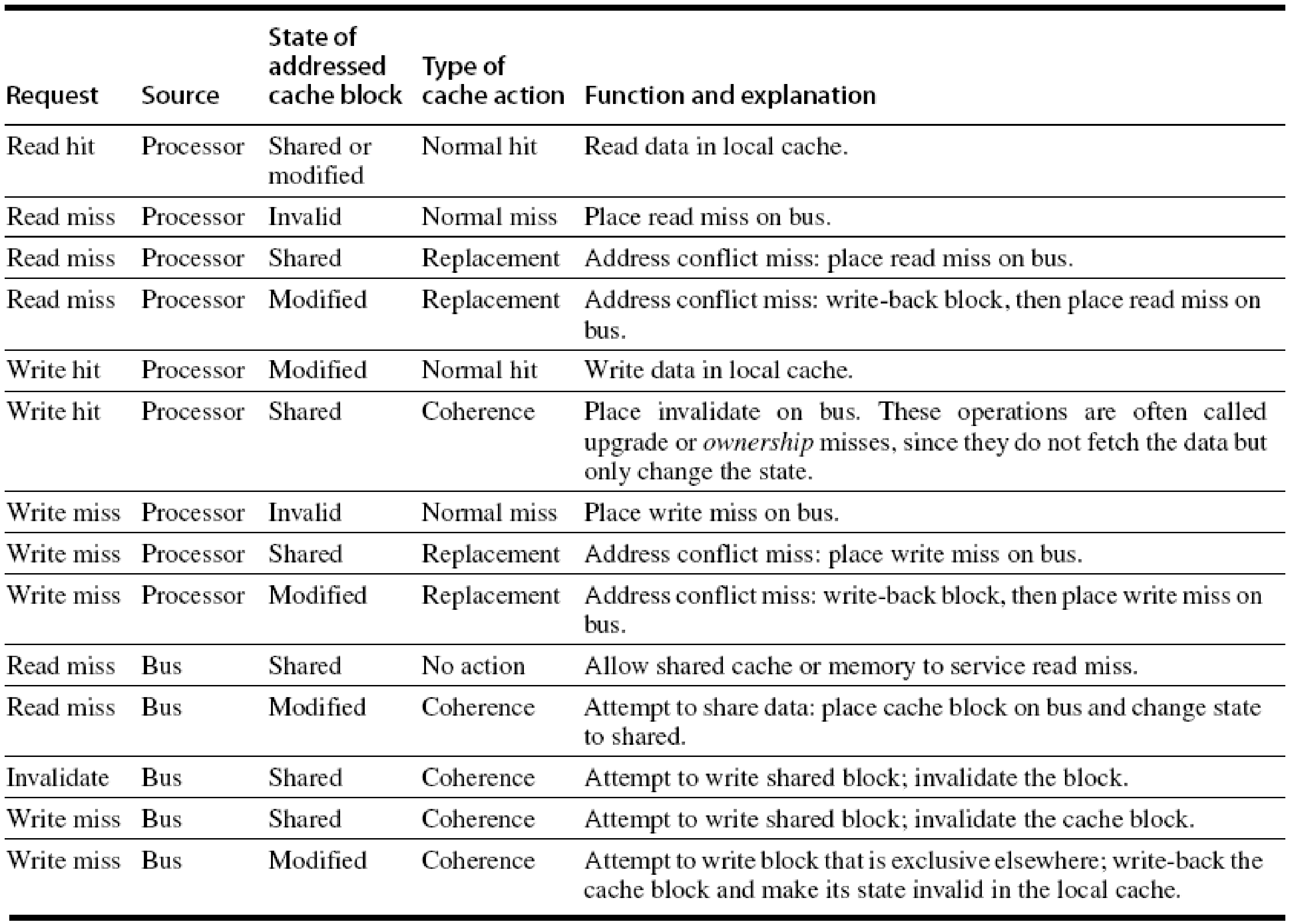
- Write update
  - On write, update all copies.

Snoopy Coherence Protocols (cont.)

- Locating an item when a read miss occurs
  - In write-back cache, the updated value must be sent to the requesting processor.

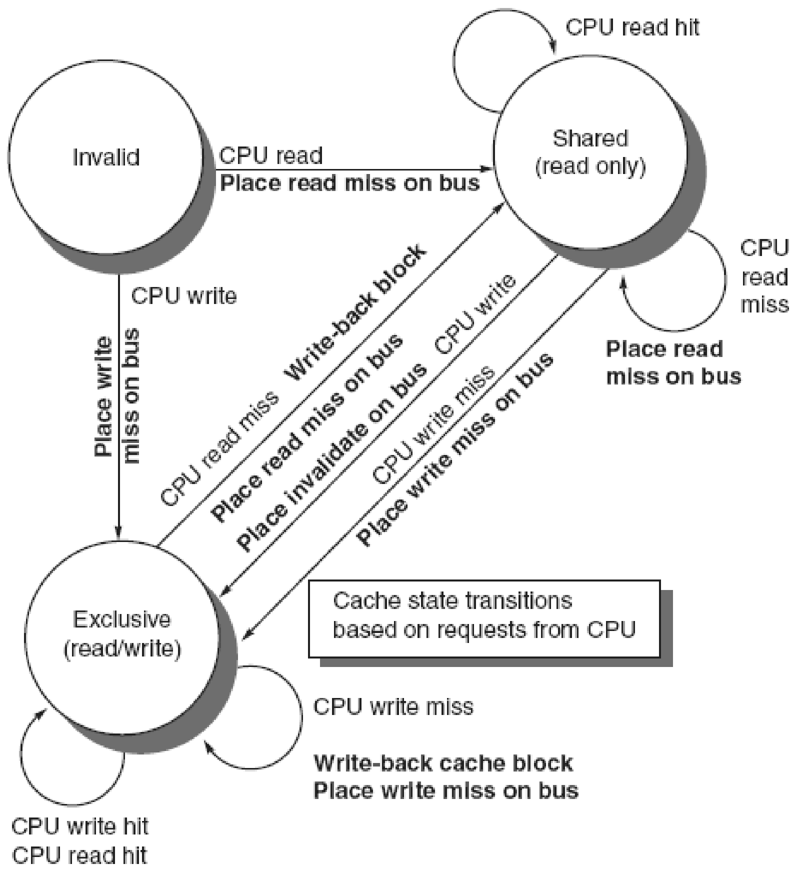
- Cache lines marked as shared or exclusive/modified
  - Only writes to shared lines that need an invalidate broadcast.
    - After this, the line is marked as exclusive.

Snoopy Coherence Protocols (cont.)

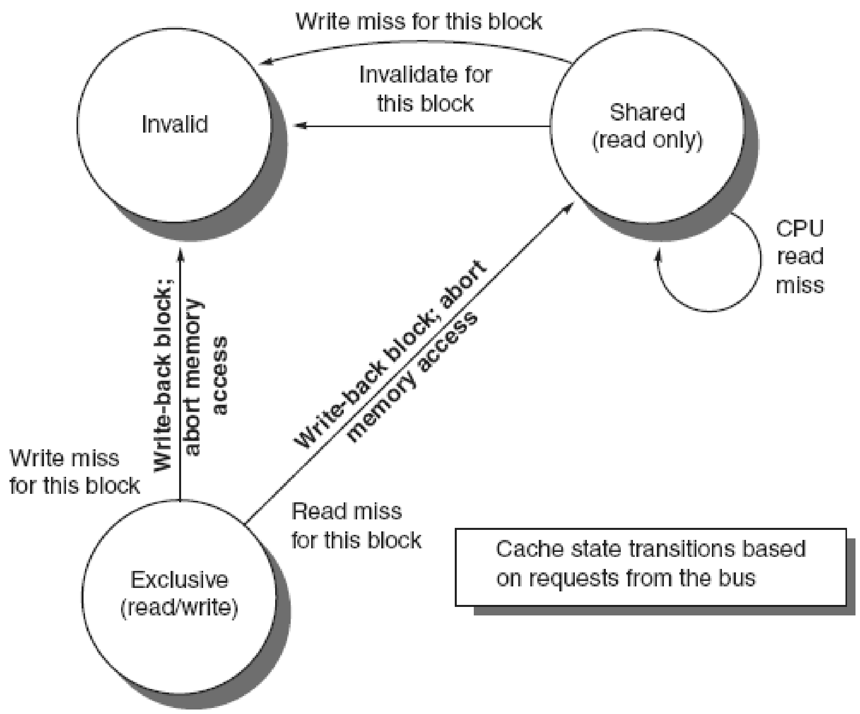


Request	Source	State of addressed cache block	Type of cache action	Function and explanation
Read hit	Processor	Shared or modified	Normal hit	Read data in local cache.
Read miss	Processor	Invalid	Normal miss	Place read miss on bus.
Read miss	Processor	Shared	Replacement	Address conflict miss: place read miss on bus.
Read miss	Processor	Modified	Replacement	Address conflict miss: write-back block, then place read miss on bus.
Write hit	Processor	Modified	Normal hit	Write data in local cache.
Write hit	Processor	Shared	Coherence	Place invalidate on bus. These operations are often called upgrade or *ownership* misses, since they do not fetch the data but only change the state.
Write miss	Processor	Invalid	Normal miss	Place write miss on bus.
Write miss	Processor	Shared	Replacement	Address conflict miss: place write miss on bus.
Write miss	Processor	Modified	Replacement	Address conflict miss: write-back block, then place write miss on bus.
Read miss	Bus	Shared	No action	Allow shared cache or memory to service read miss.
Read miss	Bus	Modified	Coherence	Attempt to share data: place cache block on bus and change state to shared.
Invalidate	Bus	Shared	Coherence	Attempt to write shared block; invalidate the block.
Write miss	Bus	Shared	Coherence	Attempt to write shared block; invalidate the cache block.
Write miss	Bus	Modified	Coherence	Attempt to write block that is exclusive elsewhere; write-back the cache block and make its state invalid in the local cache.

Snoopy Coherence Protocols (cont.)



Cache state transitions based on requests from CPU


Write miss for this block
Invalidate for this block
Invalid
Shared (read only)
CPU read miss
Write miss for this block
Write-back block; abort memory access
Write-back block; abort memory access
Read miss for this block
Exclusive (read/write)

Cache state transitions based on requests from the bus

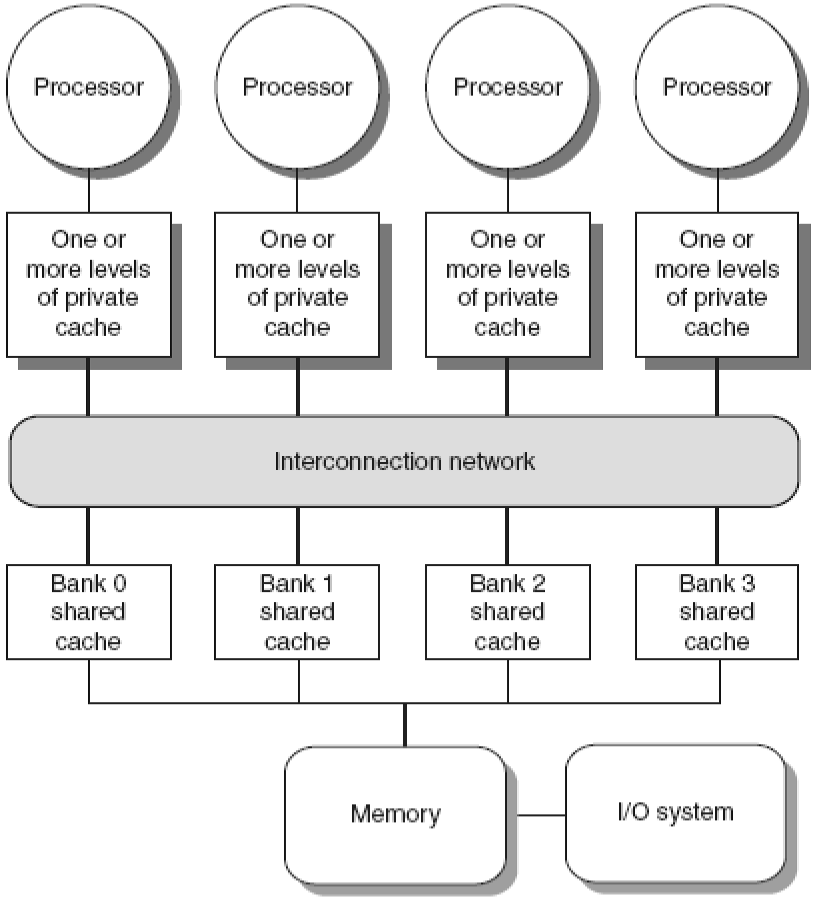
Snoopy Coherence Protocols (cont.)

- Complications for the basic MSI protocol
  - Operations are not atomic.
    - E.g., detect miss, acquire bus, receive a response.
    - Creates possibility of deadlock and races.
    - One solution: Processor that sends invalidate can hold bus until other processors receive the invalidate.

- Extensions
  - Add exclusive state to indicate clean block in only one cache (MESI protocol).
    - Prevents needing to write invalidate on a write
  - Owned state.

Coherence Protocols: Extensions

- Shared memory bus and snooping bandwidth is bottleneck for scaling symmetric multiprocessors.
  - Duplicating tags.
  - Place directory in outermost cache.
  - Use crossbars or point-to-point networks with banked memory.



Coherence Protocols (cont.)

- AMD Opteron
  - Memory directly connected to each multicore chip in NUMA-like organization.
  - Implement coherence protocol using point-to-point links.
  - Use explicit acknowledgments to order operations.

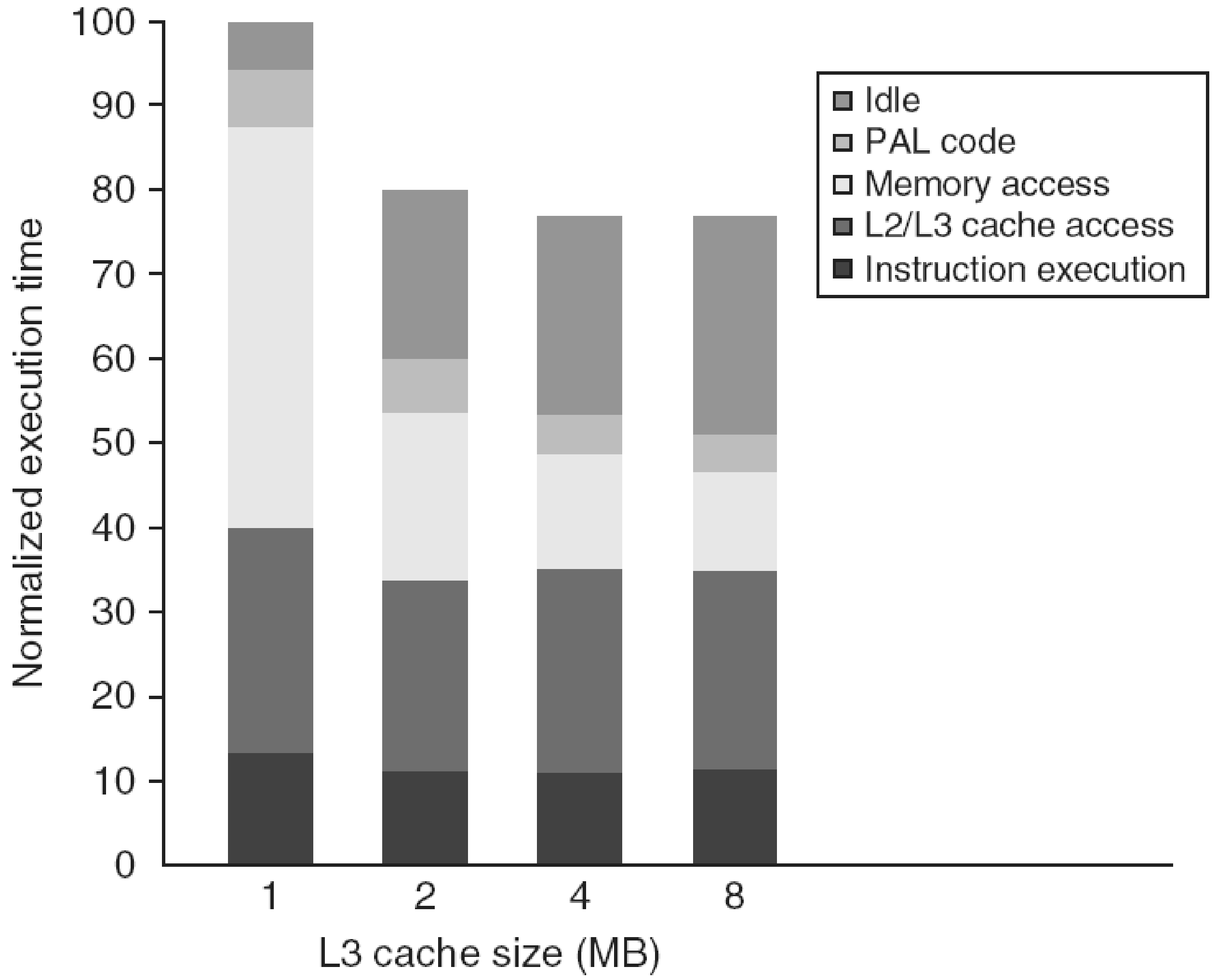
ENGINEERING@SYRACUSE

Performance of Symmetric Shared-Memory Multiprocessors

Performance

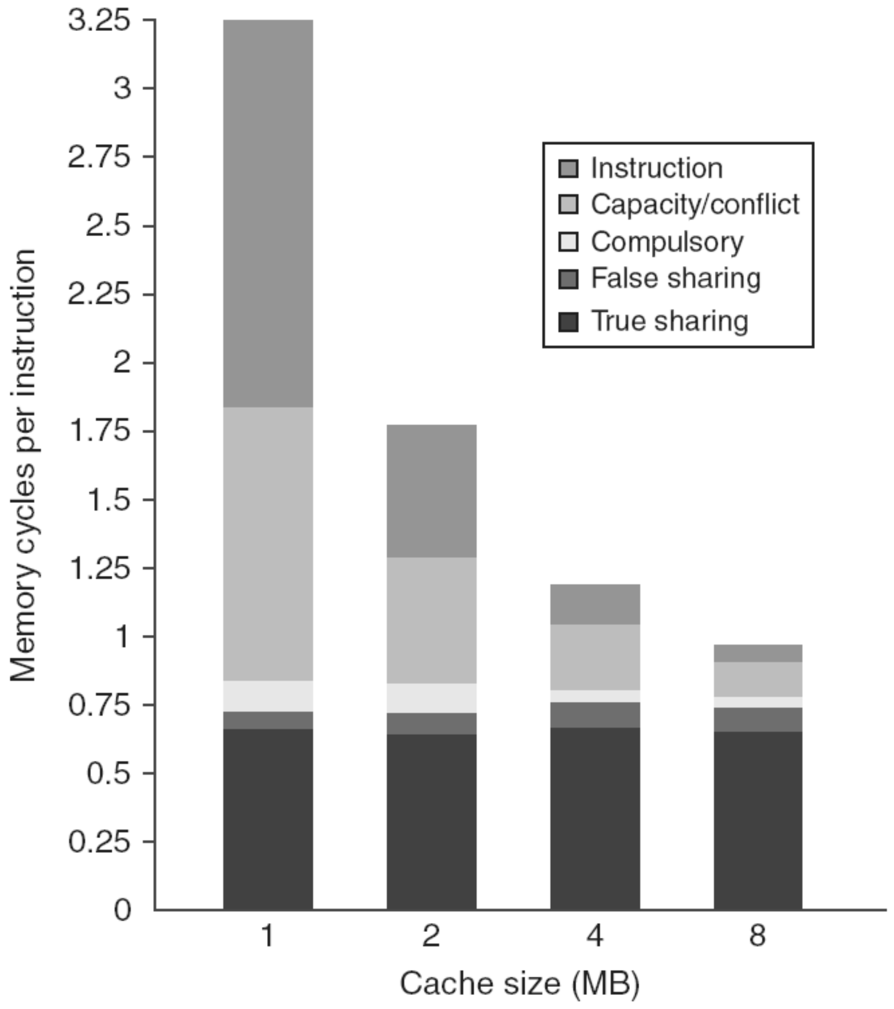
- Coherence influences cache miss rate.
  - Coherence misses
    - True sharing misses
      - Write to shared block (transmission of invalidation).
      - Read an invalidated block.
    - False sharing misses
      - Read an unmodified word in an invalidated block.

Performance Study: Commercial Workload

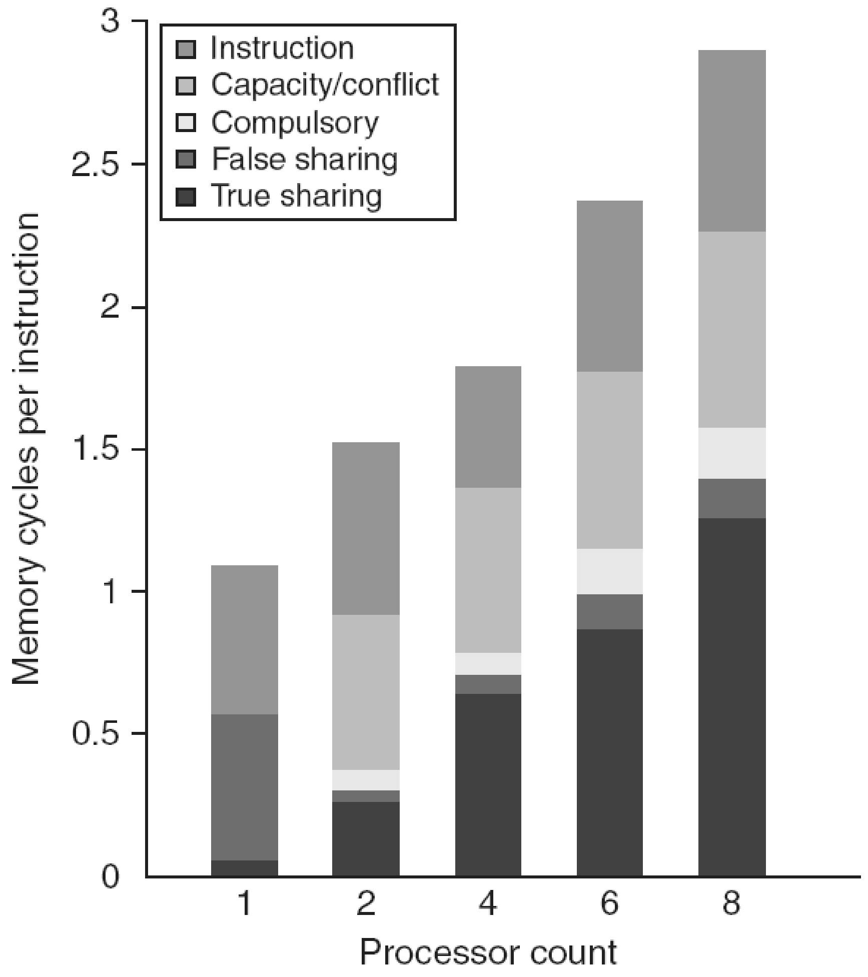


ENGINEERING@SYRACUSE

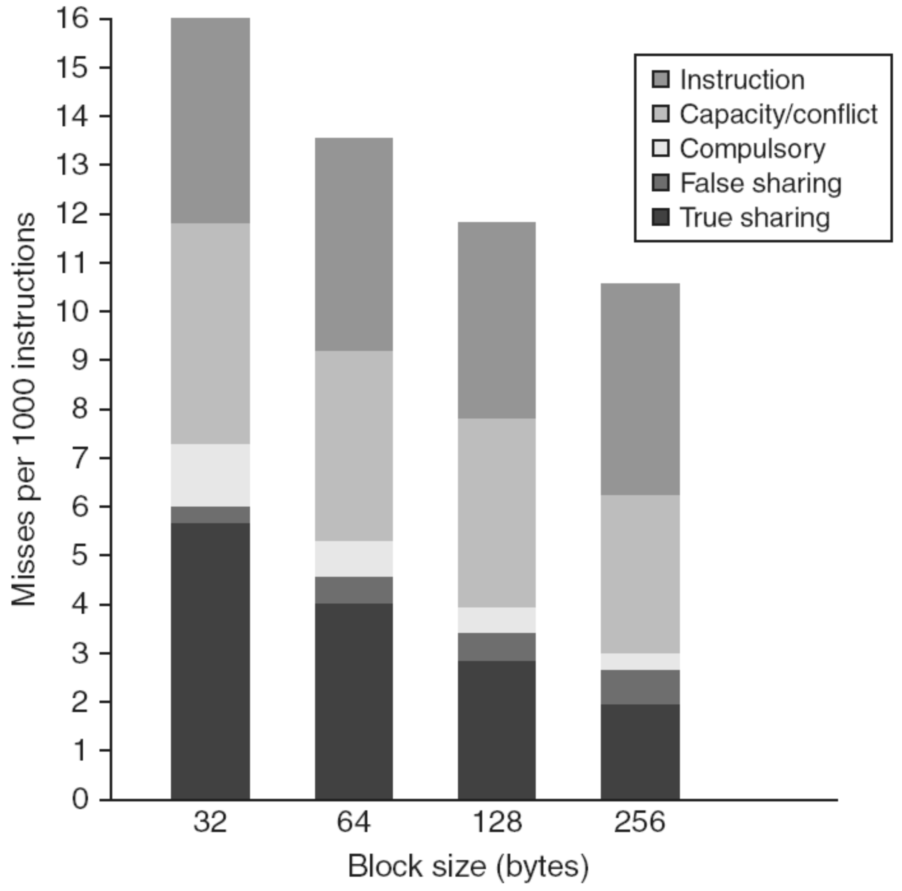
Performance Study: Commercial Workload (cont.)



Performance Study: Commercial Workload (cont.)



Performance Study: Commercial Workload (cont.)

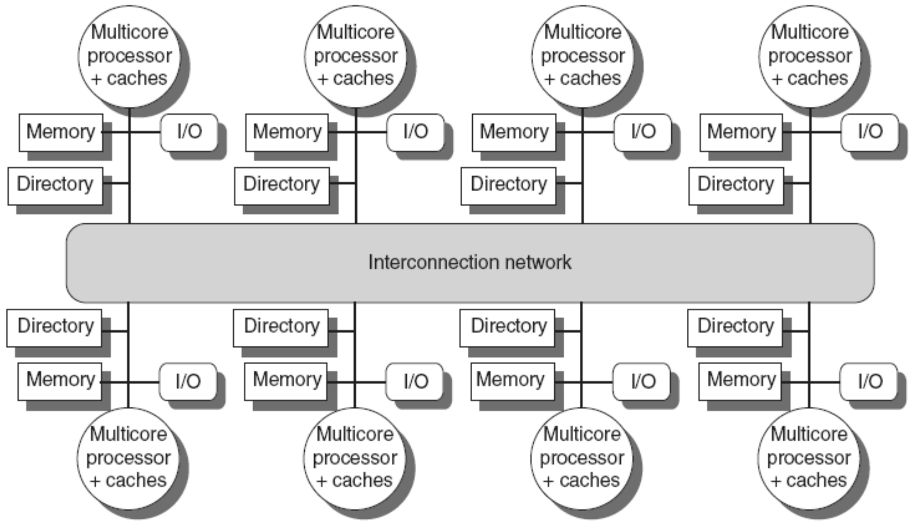


ENGINEERING@SYRACUSE

Distributed Shared-Memory Systems

Distributed Shared-Memory Systems

- Directory keeps track of every block.
  - Which caches have each block
  - Dirty status of each block
- Implement in shared L3 cache.
  - Keep bit vector of size = number cores for each block in L3.
  - Not scalable beyond shared L3.
- Implement in a distributed fashion:



Directory Protocols

- For each block, maintain state.
  - Shared
    - One or more nodes have the block cached; value in memory is up to date.
    - Set of node IDs.
  - Uncached
  - Modified
    - Exactly one node has a copy of the cache block; value in memory is out of date.
    - Owner node ID.
- Directory maintains block states and sends invalidation messages.

Synchronous Message Passing

(1) Initiate send

(2) Address translation on P_{src}

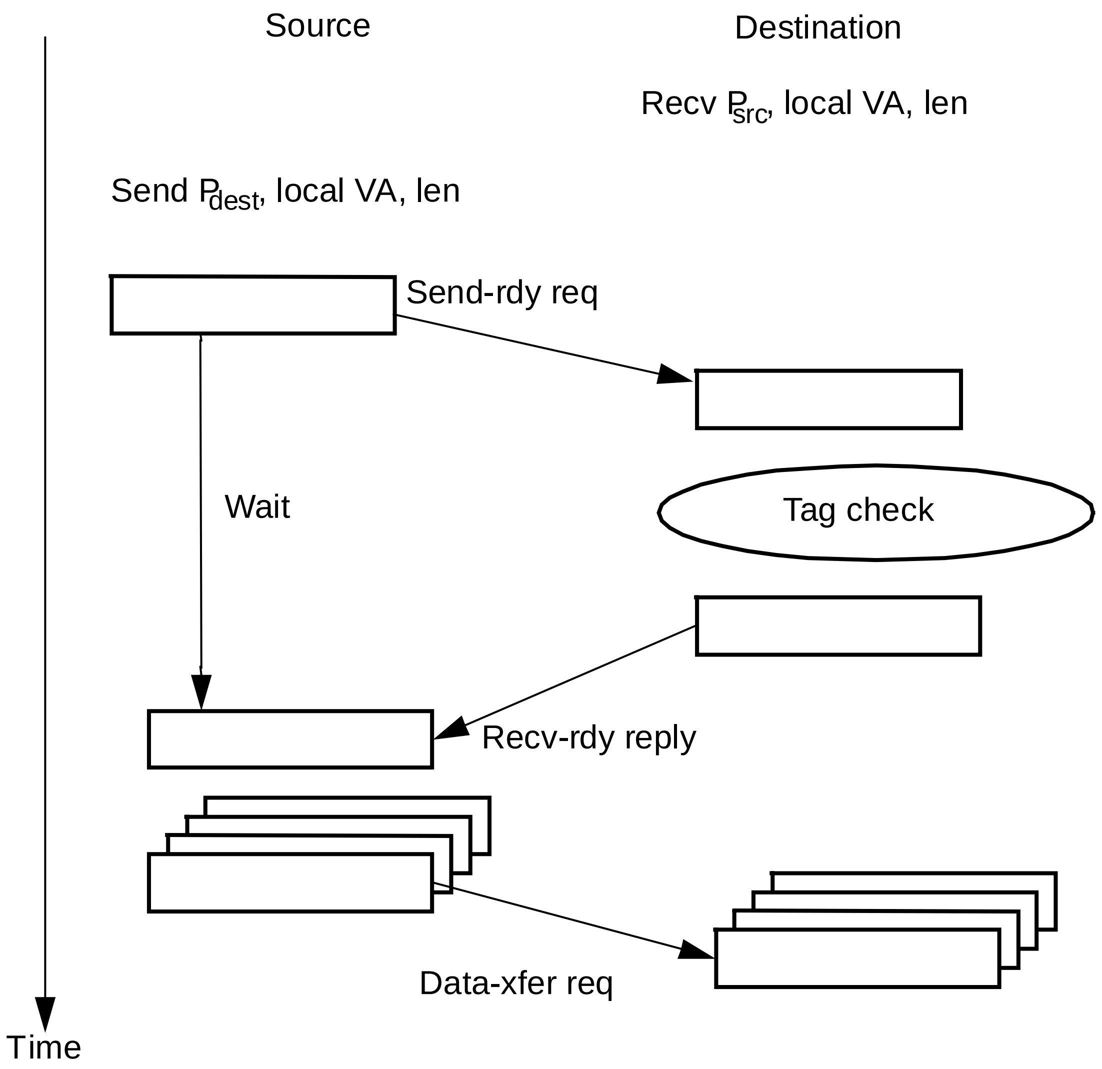
(3) Local/remote check

(4) Send-ready request

(5) Remote check for
posted receive
(assume success)

(6) Reply transaction

(7) Bulk data transfer
Source VAÕ Dest VA or ID



ENGINEERING@SYRACUSE

Asynchronous Message Passing

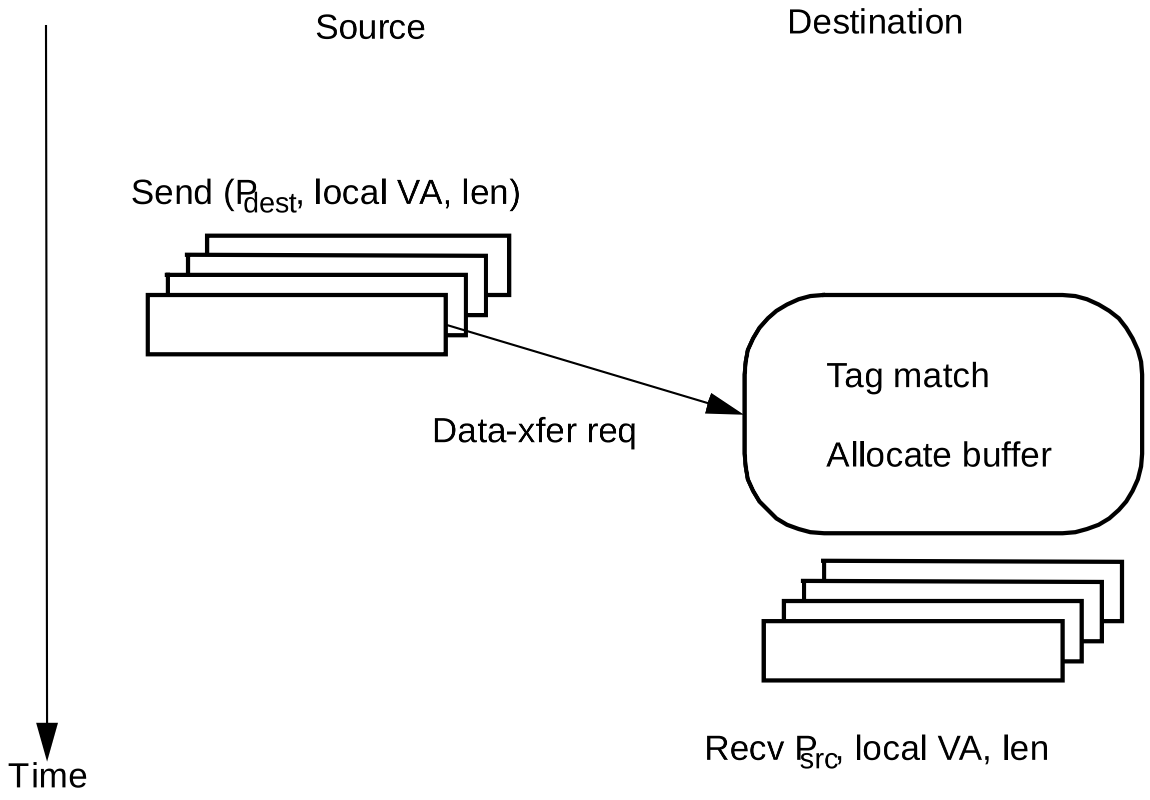
(1) Initiate send

(2) Address translation

(3) Local/remote check

(4) Send data

(5) Remote check for
posted receive; on fail,
allocate data buffer


Source
Destination
Send (P_{dest}, local VA, len)
Data-xfer req
Tag match
Allocate buffer
Recv P_{src}, local VA, len
Time

Asynchronous Message Passing: Conservative

(1) Initiate send

(2) Address translation on P_{dest}

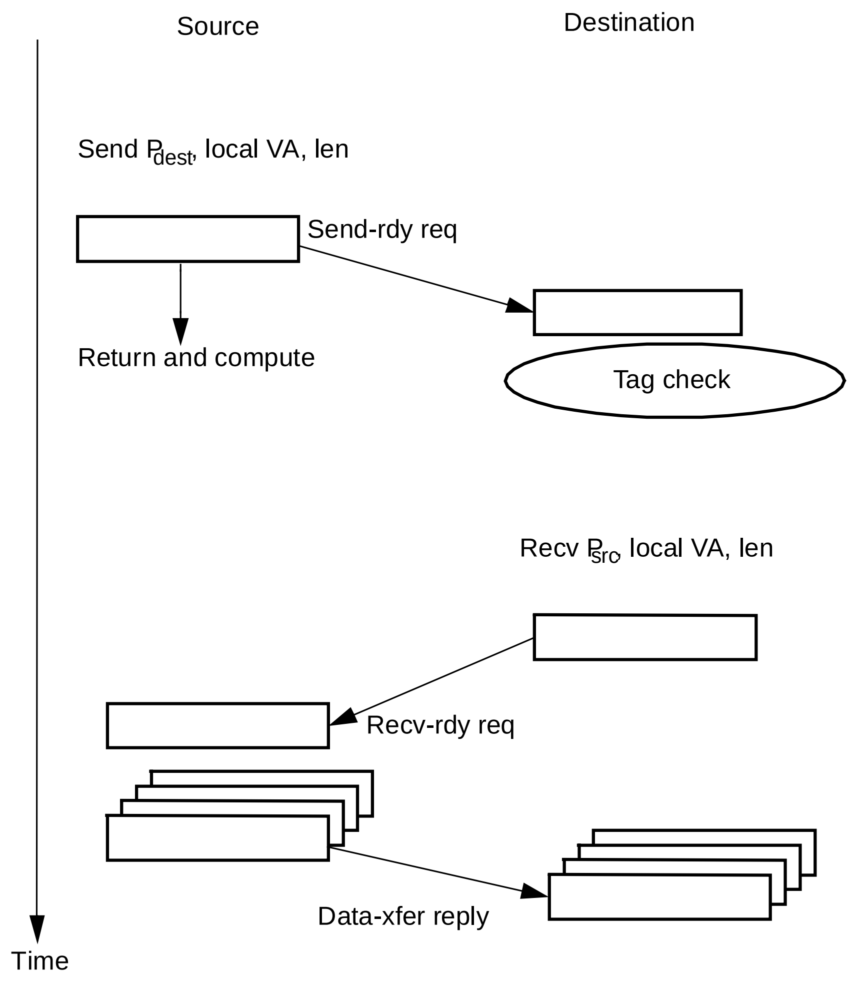
(3) Local/remote check

(4) Send-ready request

(5) Remote check for posted receive (assume fail); record send-ready

(6) Receive-ready request

(7) Bulk data reply
Source VAÕ Dest VA or ID

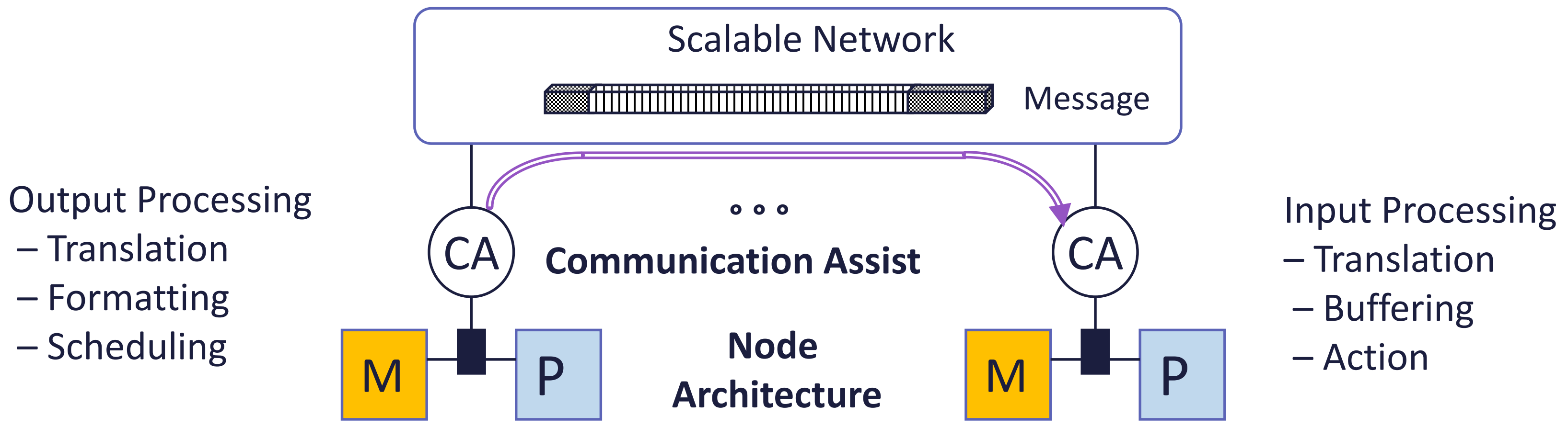


ENGINEERING@SYRACUSE

Key Features of MP Abstraction

- After handshake:
    - **Source** knows **send data address.**
    - **Destination** knows **receive data address.**
- Arbitrary storage outside the local address spaces
    - May post many sends before any receives
- Fundamentally a three-phase transaction
    - Includes a **request / response**
- Optimistic one-phase in limited “safe” cases

Network Transaction Processing



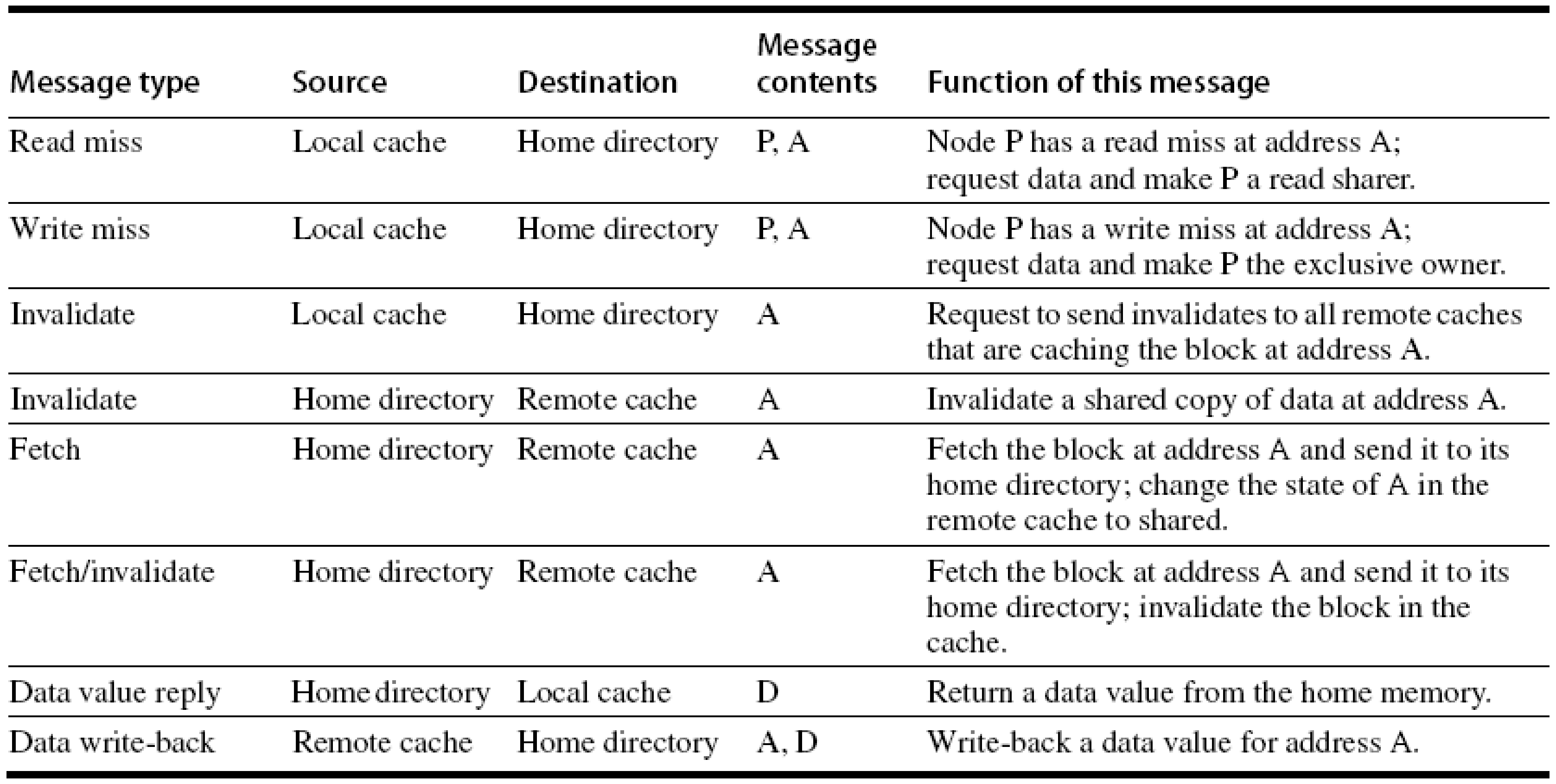
- How much dedicated processing in the communication assist?

ENGINEERING@SYRACUSE

ENGINEERING@SYRACUSE

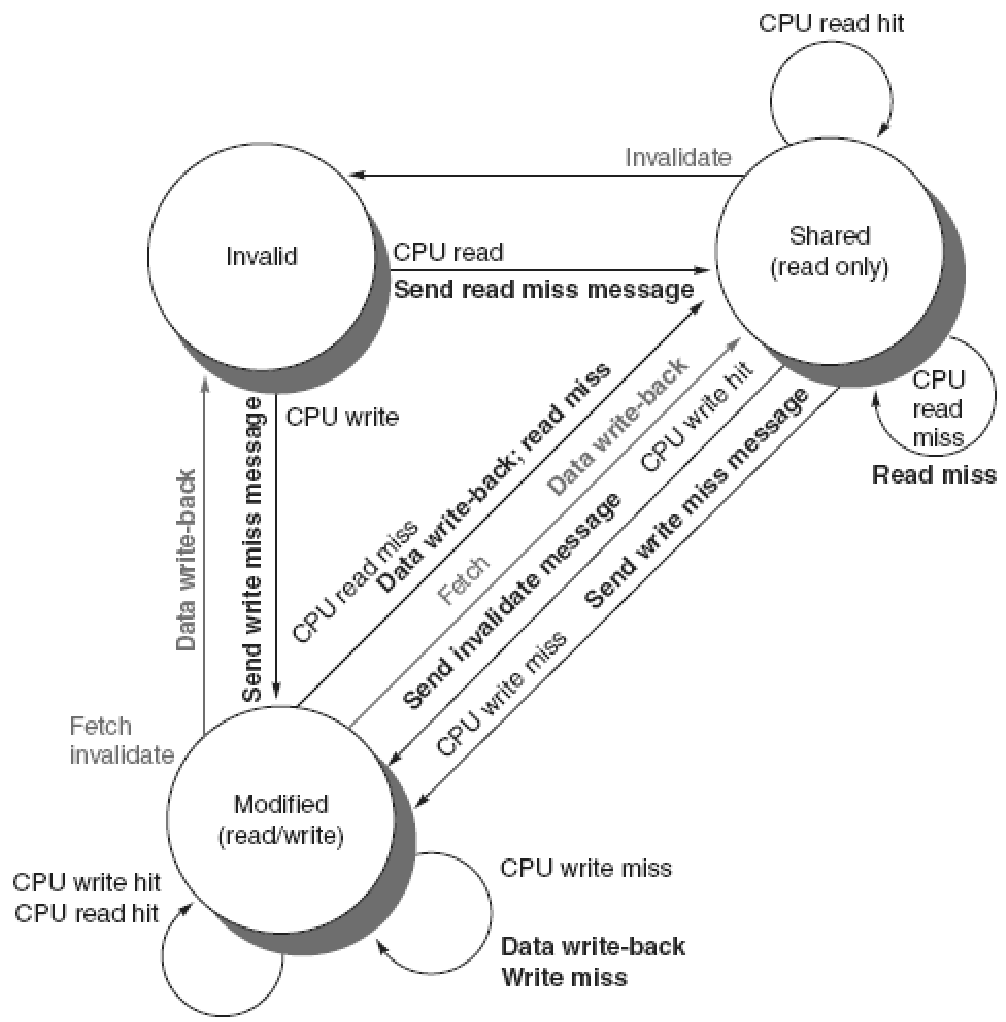
Directory-Based Protocols

Messages in MP Systems



Message type	Source	Destination	Message contents	Function of this message
Read miss	Local cache	Home directory	P, A	Node P has a read miss at address A; request data and make P a read sharer.
Write miss	Local cache	Home directory	P, A	Node P has a write miss at address A; request data and make P the exclusive owner.
Invalidate	Local cache	Home directory	A	Request to send invalidates to all remote caches that are caching the block at address A.
Invalidate	Home directory	Remote cache	A	Invalidate a shared copy of data at address A.
Fetch	Home directory	Remote cache	A	Fetch the block at address A and send it to its home directory; change the state of A in the remote cache to shared.
Fetch/invalidate	Home directory	Remote cache	A	Fetch the block at address A and send it to its home directory; invalidate the block in the cache.
Data value reply	Home directory	Local cache	D	Return a data value from the home memory.
Data write-back	Remote cache	Home directory	A, D	Write-back a data value for address A.

Directory Protocols



ENGINEERING@SYRACUSE

Directory Protocols (cont.)

- For uncached block:
  - Read miss
    - Requesting node is sent the requested data and is made the only sharing node; block is now shared.
  - Write miss
    - The requesting node is sent the requested data and becomes the sharing node; block is now exclusive.
- For shared block:
  - Read miss
    - The requesting node is sent the requested data from memory; node is added to sharing set.
  - Write miss
    - The requesting node is sent the value; all nodes in the sharing set are sent invalidate messages; sharing set contains only requesting node; block is now exclusive.

Directory Protocols (cont.)

- For exclusive block:
  - Read miss
    - The owner is sent a data fetch message; block becomes shared; owner sends data to the directory; data written back to memory; sharers set contains old owner and requestor.
  - Data write back
    - Block becomes uncached; sharer set is empty.
  - Write miss
    - Message is sent to old owner to invalidate and send the value to the directory; requestor becomes new owner; block remains exclusive.

ENGINEERING@SYRACUSE

Synchronization and Memory Consistency

Synchronization

- Basic building blocks:
  - Atomic exchange
    - Swaps register with memory location
  - Test-and-set
    - Sets under condition
  - Fetch-and-increment
    - Reads original value from memory and increments it in memory
  - Requires memory read and write in uninterruptable instruction
  - Load linked/store conditional
    - If the contents of the memory location specified by the load linked are changed before the store conditional to the same address, the store conditional fails.

Implementing Locks

- Spin lock
  - If no coherence:

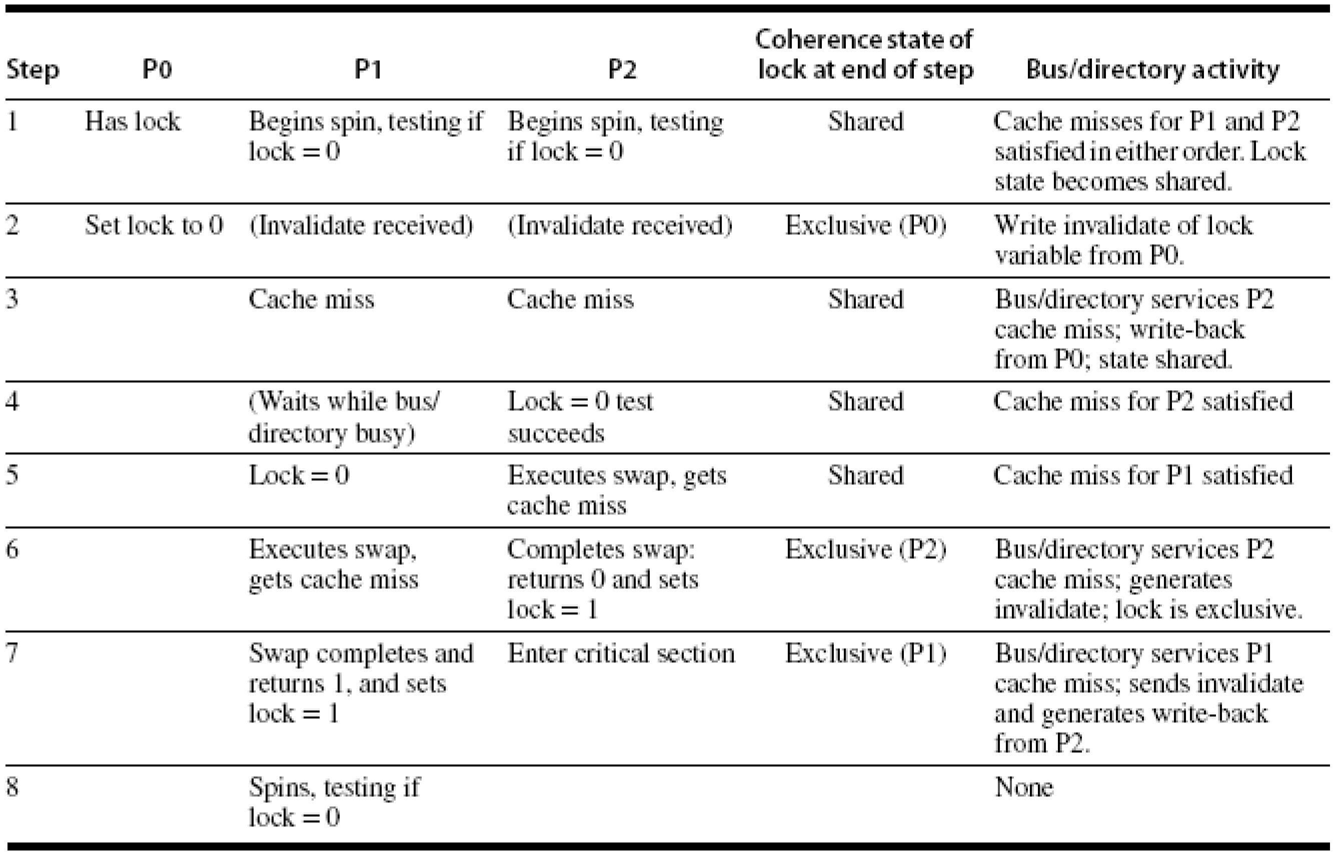
```
              DADDUI R2,R0,#1
lockit:       EXCH   R2,0(R1)      ;atomic exchange
              BNEZ   R2,lockit     ;already locked?
```

  - If coherence:

```
lockit: LD      R2,0(R1)           ;load of lock
        BNEZ    R2,lockit          ;not available-spin
        DADDUI  R2,R0,#1           ;load locked value
        EXCH    R2,0(R1)           ;swap
        BNEZ    R2,lockit          ;branch if lock wasn’t 0
```

Implementing Locks (cont.)

- Advantage of this scheme: reduces memory traffic



Step	P0	P1	P2	Coherence state of lock at end of step	Bus/directory activity
1	Has lock	Begins spin, testing if lock = 0	Begins spin, testing if lock = 0	Shared	Cache misses for P1 and P2 satisfied in either order. Lock state becomes shared.
2	Set lock to 0	(Invalidate received)	(Invalidate received)	Exclusive (P0)	Write invalidate of lock variable from P0.
3		Cache miss	Cache miss	Shared	Bus/directory services P2 cache miss; write-back from P0; state shared.
4		(Waits while bus/ directory busy)	Lock = 0 test succeeds	Shared	Cache miss for P2 satisfied
5		Lock = 0	Executes swap, gets cache miss	Shared	Cache miss for P1 satisfied
6		Executes swap, gets cache miss	Completes swap: returns 0 and sets lock = 1	Exclusive (P2)	Bus/directory services P2 cache miss; generates invalidate; lock is exclusive.
7		Swap completes and returns 1, and sets lock = 1	Enter critical section	Exclusive (P1)	Bus/directory services P1 cache miss; sends invalidate and generates write-back from P2.
8		Spins, testing if lock = 0			None

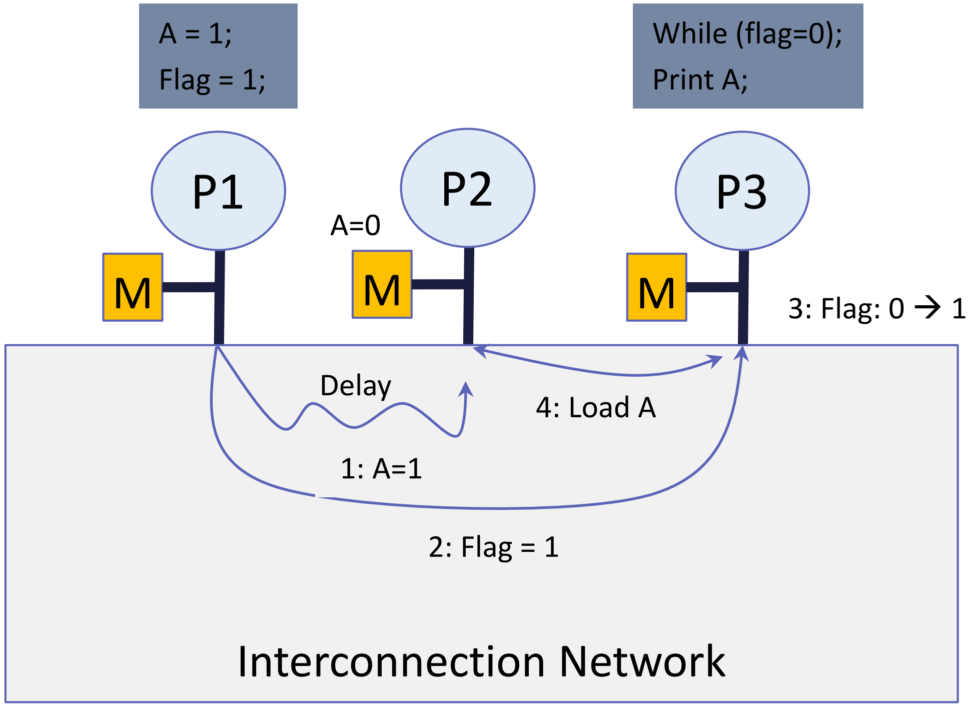
Models of Memory Consistency

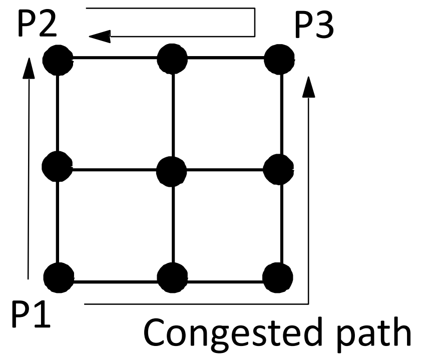
```
Processor 1:
A=0
…
A=1
if (B==0) …
```

```
Processor 2:
B=0
…
B=1
if (A==0) …
```

- Should be impossible for both if statements to be evaluated as true.
  - Delayed write invalidate?
- Sequential consistency:
  - Result of execution should be the same as long as:
    - Accesses on each processor were kept in order
    - Accesses on different processors were arbitrarily interleaved

Consistency Challenge





ENGINEERING@SYRACUSE

Implementing Locks

- To implement, delay completion of all memory accesses until all invalidations caused by the access are completed.
  - Reduces performance!

- Alternatives:
  - Program-enforced synchronization to force write on processor to occur before read on the other processor
    - Requires synchronization object for A and another for B
      - “Unlock” after write
      - “Lock” after read

Relaxed Consistency Models

- Rules:
  - X → Y
    - Operation X must complete before operation Y is done.
    - Sequential consistency requires:
      - R → W, R → R, W → R, W → W
  - Relax W → R
    - “Total store ordering”
  - Relax W → W
    - “Partial store order”
  - Relax R → W and R → R
    - “Weak ordering” and “release consistency”

Relaxed Consistency Models (cont.)

- Consistency model is multiprocessor-specific.
- Programmers will often implement explicit synchronization.
- Speculation gives much of the performance advantage of relaxed models with sequential consistency.
  - Basic idea: If an invalidation arrives for a result that has not been committed, use speculation recovery.

ENGINEERING@SYRACUSE

Conclusions

In Conclusion

- Parallel computing: science, engineering, commercial
- Parallel architecture: convergence, scalability
- Abstraction models: programming, communication
- Centralized shared-memory architecture: small scale
  - Snoopy coherence protocols
  - Performance of symmetric shared-memory multiprocessors
- Distributed shared-memory systems: large scale
  - Directory-based protocols
- Synchronization: atomic, spins, load locked, store conditional
- Memory consistency: relaxed models

ENGINEERING@SYRACUSE