Course Wrap-Up

CS 258, Spring 99
David E. Culler
Computer Science Division
U.C. Berkeley

Today's Plan

- Whirlwind tour of where we've been
- Some thought on where things are headed
- HKN evaluation

CS 258
Parallel Computer Architecture

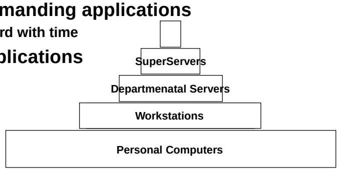
CS 258, Spring 99
David E. Culler
Computer Science Division
U.C. Berkeley

What will you get out of CS258?

- In-depth understanding of the design and engineering of modern parallel computers
  - technology forces
  - fundamental architectural issues
    - » naming, replication, communication, synchronization
  - basic design techniques
    - » cache coherence, protocols, networks, pipelining, …
  - methods of evaluation
  - underlying engineering trade-offs
- from moderate to very large scale
- across the hardware/software boundary

Will it be worthwhile?

- Absolutely!
  - even through few of you will become PP designers
- The fundamental issues and solutions translate across a wide spectrum of systems.
  - Crisp solutions in the context of parallel machines.
- Pioneered at the thin-end of the platform pyramid on the most-demanding applications
  - migrate downward with time
- Understand implications for software



SuperServers

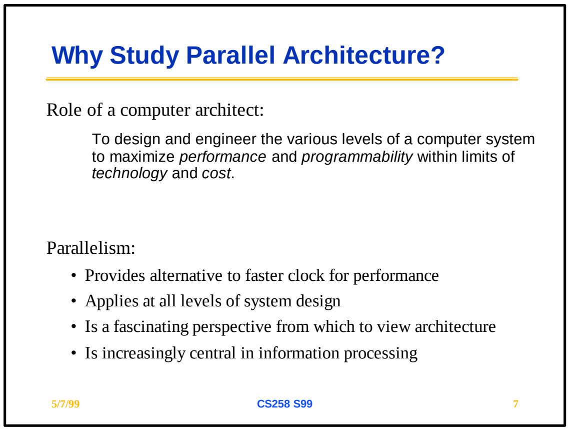
Departmenatal Servers

Workstations

Personal Computers

What is 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?



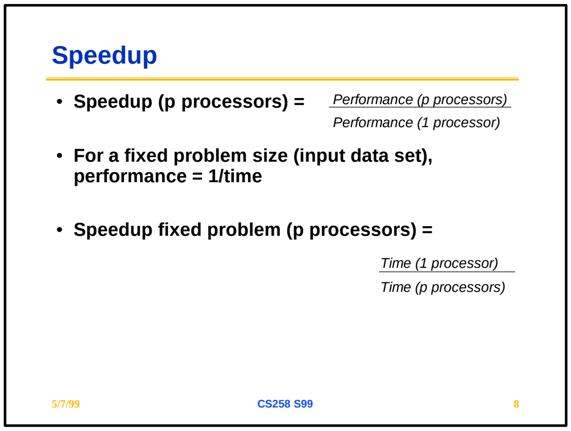
Why Study Parallel Architecture?

Role of a computer architect:

To design and engineer the various levels of a computer system to maximize *performance* and *programmability* within limits of *technology* and *cost*.

Parallelism:

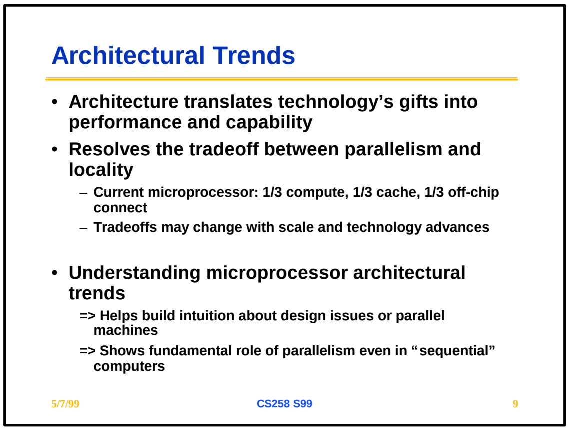
- Provides alternative to faster clock for performance
- Applies at all levels of system design
- Is a fascinating perspective from which to view architecture
- Is increasingly central in information processing



Speedup

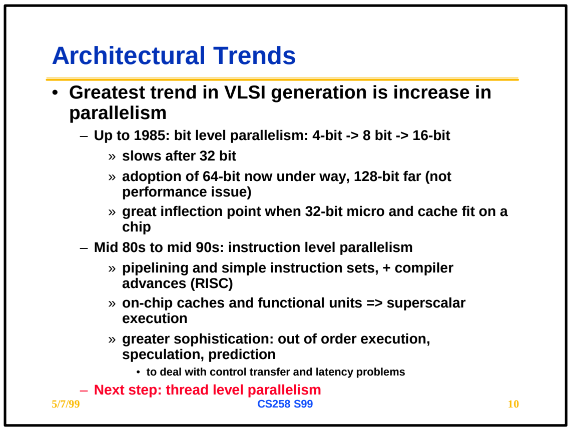
- **Speedup (p processors) =** $\dfrac{\textit{Performance (p processors)}}{\textit{Performance (1 processor)}}$
- **For a fixed problem size (input data set), performance = 1/time**
- **Speedup fixed problem (p processors) =**

$$\frac{\textit{Time (1 processor)}}{\textit{Time (p processors)}}$$



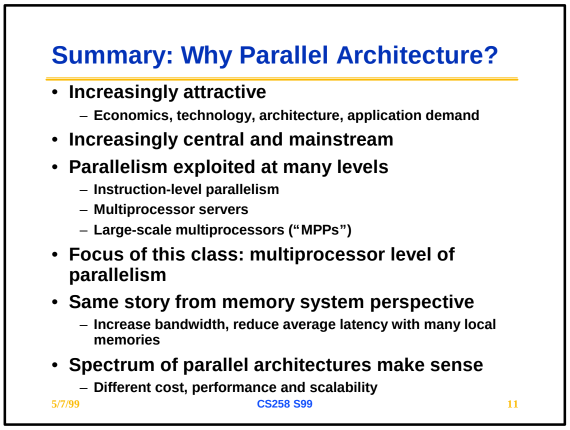
Architectural Trends

- **Architecture translates technology's gifts into performance and capability**
- **Resolves the tradeoff between parallelism and locality**
  - **Current microprocessor: 1/3 compute, 1/3 cache, 1/3 off-chip connect**
  - **Tradeoffs may change with scale and technology advances**
- **Understanding microprocessor architectural trends**
  - **=> Helps build intuition about design issues or parallel machines**
  - **=> Shows fundamental role of parallelism even in "sequential" computers**



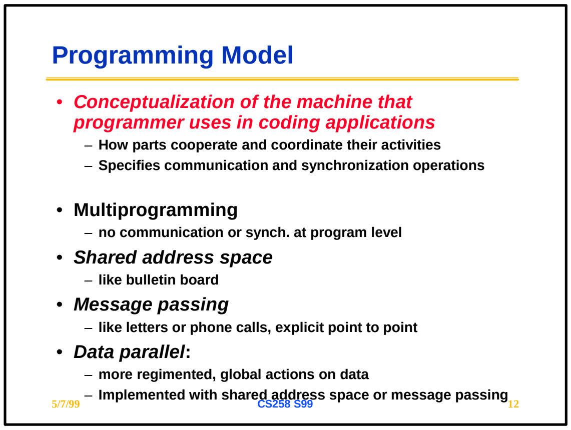
Architectural Trends

- **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**
    - **» adoption of 64-bit now under way, 128-bit far (not performance issue)**
    - **» 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**
  - **Next step: thread level parallelism**



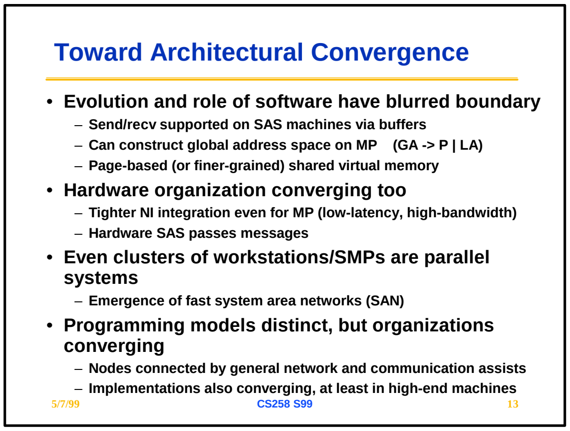
Summary: Why Parallel Architecture?

- **Increasingly attractive**
  - **Economics, technology, architecture, application demand**
- **Increasingly central and mainstream**
- **Parallelism exploited at many levels**
  - **Instruction-level parallelism**
  - **Multiprocessor servers**
  - **Large-scale multiprocessors ("MPPs")**
- **Focus of this class: multiprocessor level of parallelism**
- **Same story from memory system perspective**
  - **Increase bandwidth, reduce average latency with many local memories**
- **Spectrum of parallel architectures make sense**
  - **Different cost, performance and scalability**



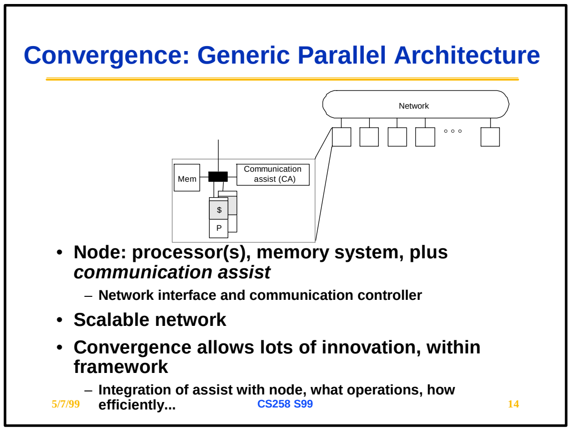
Programming Model

- ***Conceptualization of the machine that programmer uses in coding applications***
  - **How parts cooperate and coordinate their activities**
  - **Specifies communication and synchronization operations**
- **Multiprogramming**
  - **no communication or synch. 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**



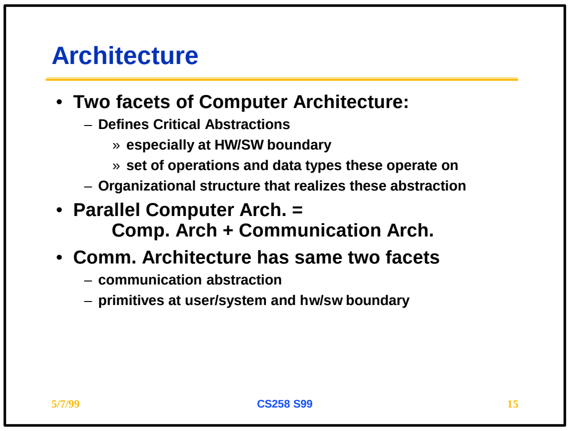
Toward Architectural Convergence

- Evolution and role of software have blurred boundary
  - Send/recv supported on SAS machines via buffers
  - Can construct global address space on MP (GA -> P | LA)
  - Page-based (or finer-grained) shared virtual memory
- Hardware organization converging too
  - Tighter NI integration even for MP (low-latency, high-bandwidth)
  - Hardware SAS passes 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



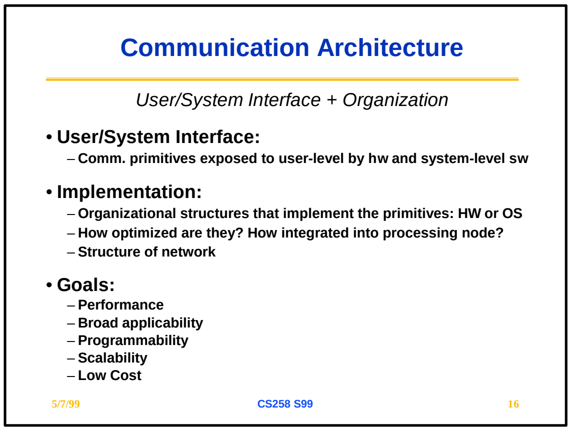
Convergence: Generic Parallel Architecture

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



Architecture

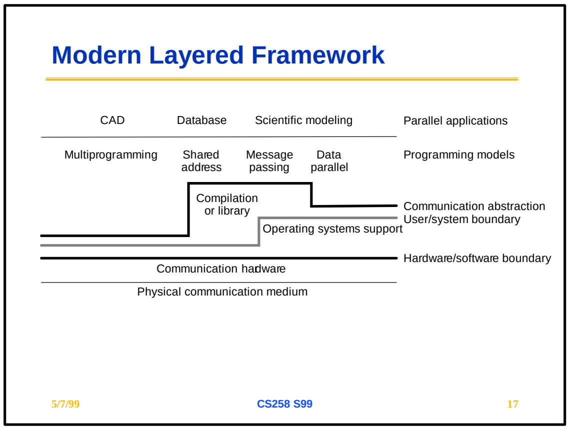
- Two facets of Computer Architecture:
  - Defines Critical Abstractions
    - » especially at HW/SW boundary
    - » set of operations and data types these operate on
  - Organizational structure that realizes these abstraction
- Parallel Computer Arch. =
  Comp. Arch + Communication Arch.
- Comm. Architecture has same two facets
  - communication abstraction
  - primitives at user/system and hw/sw boundary



Communication Architecture

User/System Interface + Organization

- User/System Interface:
  - Comm. primitives exposed to user-level by hw and system-level sw
- Implementation:
  - Organizational structures that implement the primitives: HW or OS
  - How optimized are they? How integrated into processing node?
  - Structure of network
- Goals:
  - Performance
  - Broad applicability
  - Programmability
  - Scalability
  - Low Cost



Modern Layered Framework

CAD Database Scientific modeling Parallel applications

Multiprogramming Shared address Message passing Data parallel Programming models

Compilation or library

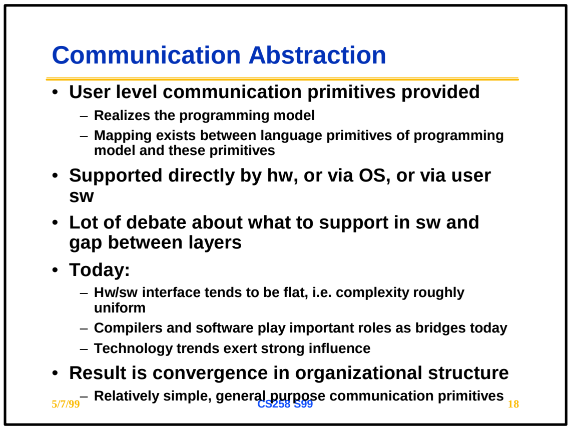
Communication abstraction
User/system boundary

Operating systems support

Hardware/software boundary

Communication hadware

Physical communication medium



Communication Abstraction

- User level communication primitives provided
  - Realizes the programming model
  - Mapping exists between language primitives of programming model and these primitives
- Supported directly by hw, or via OS, or via user sw
- Lot of debate about what to support in sw and gap between layers
- Today:
  - Hw/sw interface tends to be flat, i.e. complexity roughly uniform
  - 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

Understanding Parallel Architecture

- Traditional taxonomies not very useful
- Programming models not enough, nor hardware structures
  - Same one can be supported by radically different architectures

=> Architectural distinctions that affect software

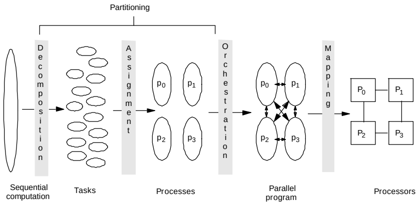
  - Compilers, libraries, programs

- Design of user/system and hardware/software interface
  - Constrained from above by progr. models and below by technology
- Guiding principles provided by layers
  - What primitives are provided at communication abstraction
  - How programming models map to these
  - How they are mapped to hardware

Fundamental Design Issues

- At any layer, interface (contract) aspect and performance aspects
  - Naming: How are logically shared data and/or processes referenced?
  - Operations: What operations are provided on these data
  - Ordering: How are accesses to data ordered and coordinated?
  - Replication: How are data replicated to reduce communication?
  - Communication Cost: Latency, bandwidth, overhead, occupancy

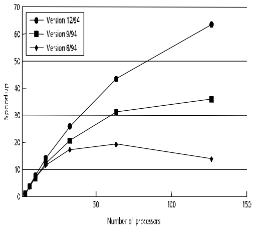
4 Steps in Creating a Parallel Program



- **Decomposition** of computation in tasks
- **Assignment** of tasks to processes
- **Orchestration** of data access, comm, synch.
- **Mapping** processes to processors

Performance Goal => Speedup

- Architect Goal
  - observe how program uses machine and improve the design to enhance performance
- Programmer Goal
  - observe how the program uses the machine and improve the implementation to enhance performance
- What do you observe?
- Who fixes what?



Recap: Performance Trade-offs

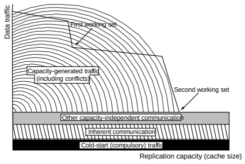
- Programmer's View of Performance

$$\text{Speedup} \leq \frac{\text{Sequential Work}}{\mathit{Max}\ (\text{Work} + \text{Synch Wait Time} + \text{Comm Cost} + \text{Extra Work})}$$

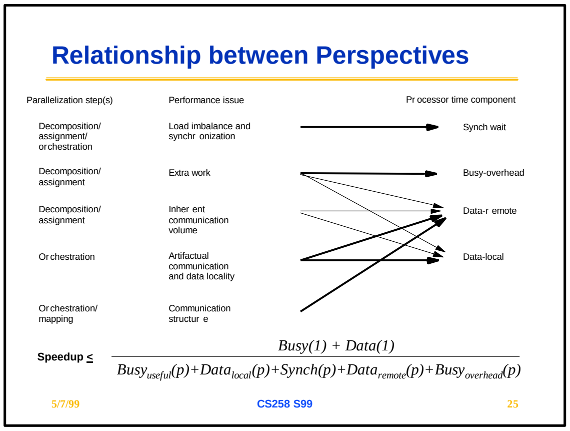
- Different goals often have conflicting demands
  - Load Balance
    - » fine-grain tasks, random or dynamic assignment
  - Communication
    - » coarse grain tasks, decompose to obtain locality
  - Extra Work
    - » coarse grain tasks, simple assignment
  - Communication Cost:
    - » big transfers: amortize overhead and latency
    - » small transfers: reduce contention

Working Set Perspective

•At a given level of the hierarchy (to the next further one)



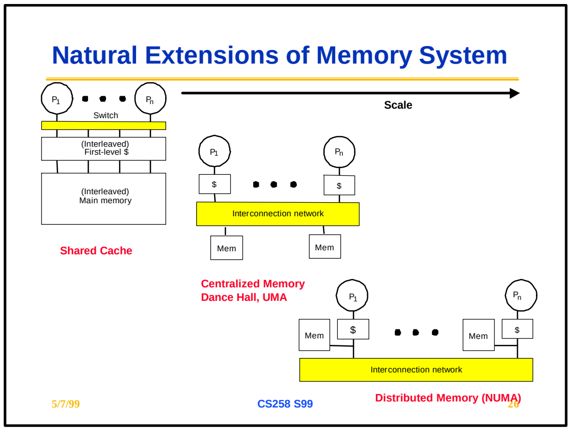
- Hierarchy of working sets
- At first level cache (fully assoc, one-word block), inherent to algorithm
  - » ***working set curve*** for program
- Traffic from any type of miss can be local or nonlocal (communication)



Relationship between Perspectives

Parallelization step(s)
Performance issue
Pr ocessor time component
Decomposition/ assignment/ or chestration
Load imbalance and synchr onization
Synch wait
Decomposition/ assignment
Extra work
Busy-overhead
Decomposition/ assignment
Inher ent communication volume
Data-r emote
Or chestration
Artifactual communication and data locality
Data-local
Or chestration/ mapping
Communication structur e

$$\textbf{Speedup} \leq \frac{Busy(1) + Data(1)}{Busy_{useful}(p)+Data_{local}(p)+Synch(p)+Data_{remote}(p)+Busy_{overhead}(p)}$$

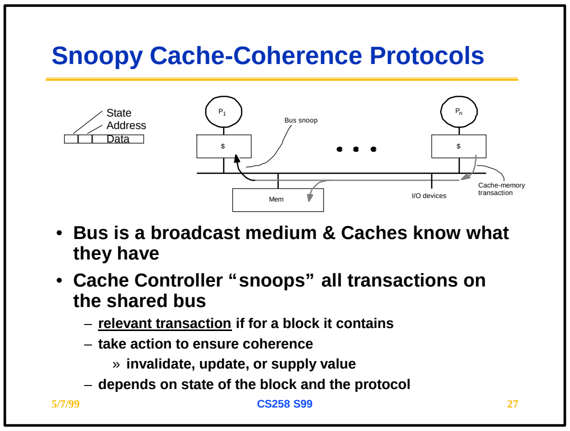


Natural Extensions of Memory System

Shared Cache

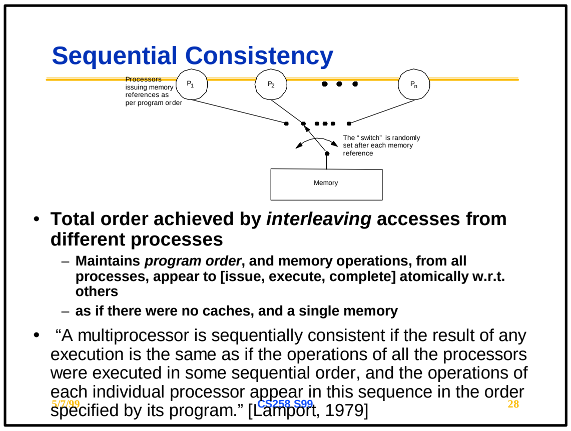
Centralized Memory
Dance Hall, UMA

Distributed Memory (NUMA)



Snoopy Cache-Coherence Protocols

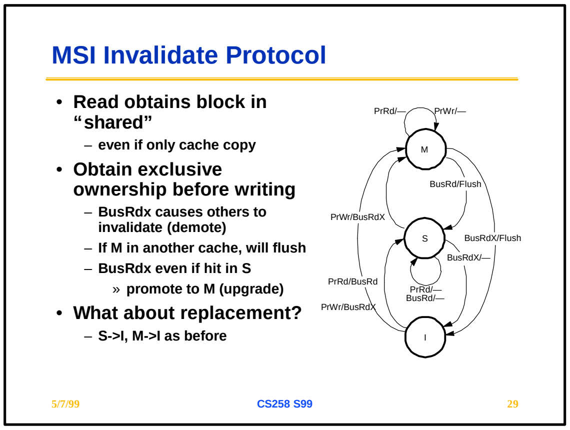
- Bus is a broadcast medium & Caches know what they have
- Cache Controller "snoops" all transactions on the shared bus
  - relevant transaction if for a block it contains
  - take action to ensure coherence
    - » invalidate, update, or supply value
  - depends on state of the block and the protocol



Sequential Consistency

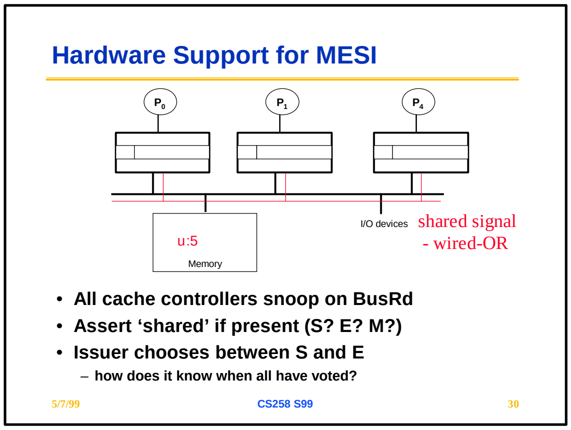
Processors issuing memory references as per program order
P1
P2
Pn
The " switch" is randomly set after each memory reference
Memory

- Total order achieved by *interleaving* accesses from different processes
  - Maintains *program order*, and memory operations, from all processes, appear to [issue, execute, complete] atomically w.r.t. others
  - as if there were no caches, and a single memory
- "A multiprocessor is sequentially consistent if the result of any execution is the same as if the operations of all the processors were executed in some sequential order, and the operations of each individual processor appear in this sequence in the order specified by its program." [Lamport, 1979]



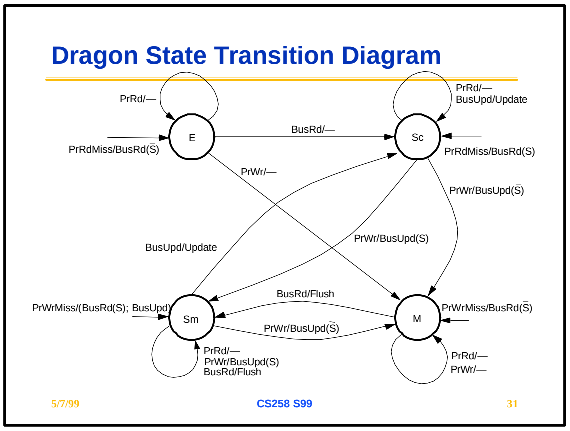
MSI Invalidate Protocol

- Read obtains block in "shared"
  - even if only cache copy
- Obtain exclusive ownership before writing
  - BusRdx causes others to invalidate (demote)
  - If M in another cache, will flush
  - BusRdx even if hit in S
    - » promote to M (upgrade)
- What about replacement?
  - S->I, M->I as before

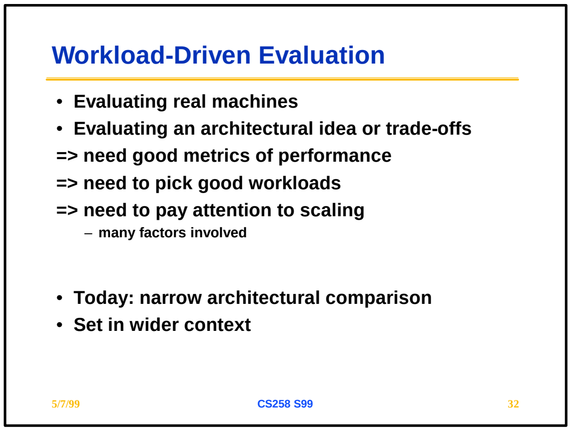


Hardware Support for MESI

- All cache controllers snoop on BusRd
- Assert 'shared' if present (S? E? M?)
- Issuer chooses between S and E
  - how does it know when all have voted?



Dragon State Transition Diagram



Workload-Driven Evaluation

- Evaluating real machines
- Evaluating an architectural idea or trade-offs

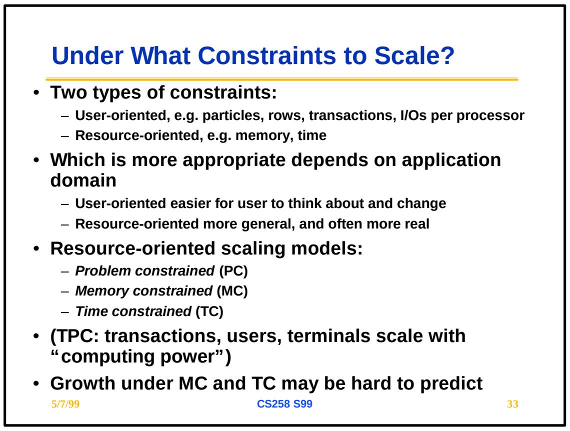
=> need good metrics of performance

=> need to pick good workloads

=> need to pay attention to scaling

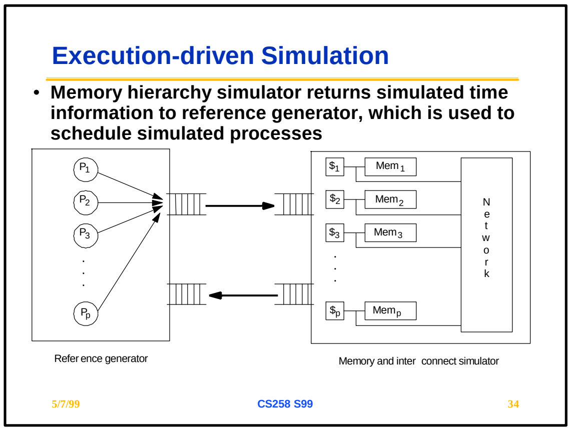
  – many factors involved

- Today: narrow architectural comparison
- Set in wider context



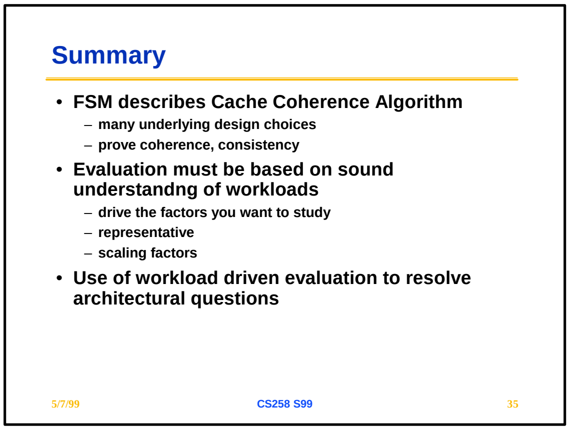
Under What Constraints to Scale?

- Two types of constraints:
  - User-oriented, e.g. particles, rows, transactions, I/Os per processor
  - Resource-oriented, e.g. memory, time
- Which is more appropriate depends on application domain
  - User-oriented easier for user to think about and change
  - Resource-oriented more general, and often more real
- Resource-oriented scaling models:
  - *Problem constrained* (PC)
  - *Memory constrained* (MC)
  - *Time constrained* (TC)
- (TPC: transactions, users, terminals scale with “computing power”)
- Growth under MC and TC may be hard to predict



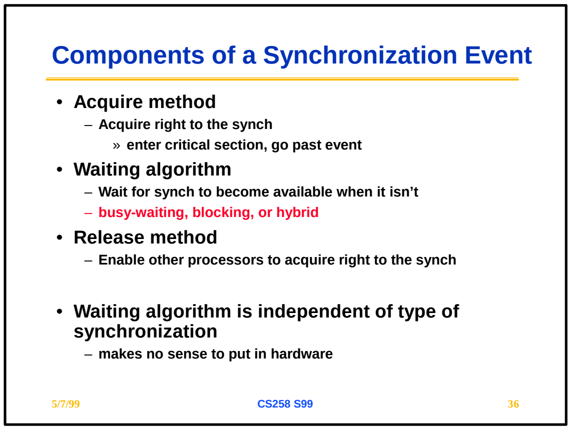
Execution-driven Simulation

- Memory hierarchy simulator returns simulated time information to reference generator, which is used to schedule simulated processes



Summary

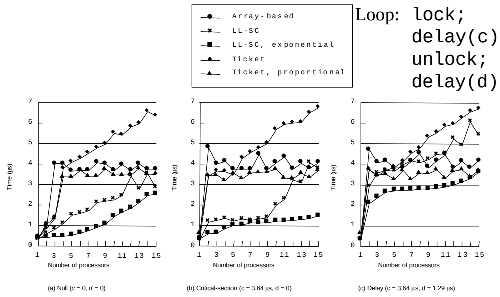
- FSM describes Cache Coherence Algorithm
  - many underlying design choices
  - prove coherence, consistency
- Evaluation must be based on sound understandng of workloads
  - drive the factors you want to study
  - representative
  - scaling factors
- Use of workload driven evaluation to resolve architectural questions



Components of a Synchronization Event

- Acquire method
  - Acquire right to the synch
    - » enter critical section, go past event
- Waiting algorithm
  - Wait for synch to become available when it isn’t
  - busy-waiting, blocking, or hybrid
- Release method
  - Enable other processors to acquire right to the synch
- Waiting algorithm is independent of type of synchronization
  - makes no sense to put in hardware

Lock Performance on SGI Challenge



(a) Null (*c* = 0, *d* = 0)

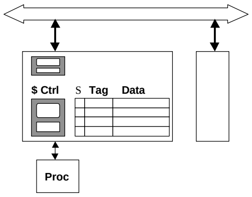
(b) Critical-section (c = 3.64 µs, d = 0)

(c) Delay (c = 3.64 µs, d = 1.29 µs)

```
Loop: lock;
      delay(c);
      unlock;
      delay(d);
```

Reality

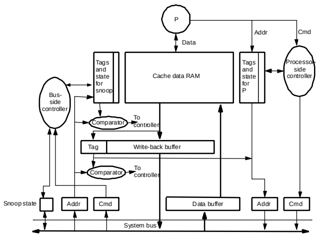
- **Protocol defines logical FSM for each block**
- **Cache controller FSM**
  - **multiple states per miss**
- **Bus controller FSM**
- **Other $Ctrls Get bus**
- **Multiple Bus trnxs**
  - **write-back**
- **Multi-Level Caches**
- **Split-Transaction Busses**



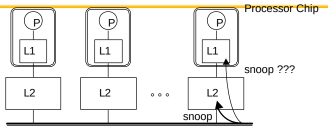
Preliminary Design Issues

- **Design of cache controller and tags**
  - **Both processor and bus need to look up**
- **How and when to present snoop results on bus**
- **Dealing with write-backs**
- **Overall set of actions for memory operation not atomic**
  - **Can introduce race conditions**
- **atomic operations**
- **New issues deadlock, livelock, starvation, serialization, etc.**

Basic design



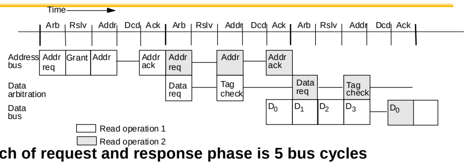
Multilevel Cache Hierarchies



- **Independent snoop hardware for each level?**
  - **processor pins for shared bus**
  - **contention for processor cache access ?**
- **Snoop only at L2 and propagate relevant transactions**
- **Inclusion property**

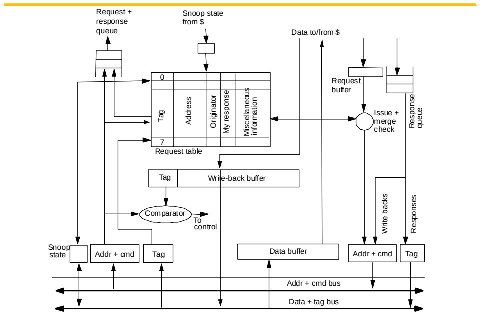
(1) contents L1 is a subset of L
(2) any block in modified state in L1 is in modified state in L2
1 => all transactions relavant to L1 are relavant to L2
2 => on BusRd L2 can wave off memory access and inform L1

Bus Design (continued)

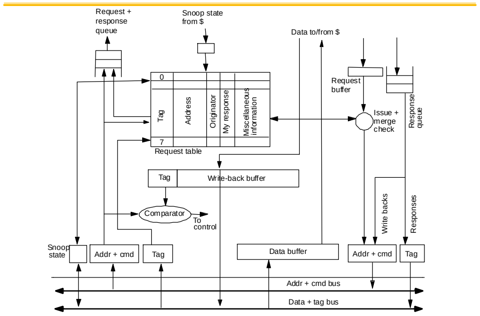


- **Each of request and response phase is 5 bus cycles**
  - **Response: 4 cycles for data (128 bytes, 256-bit bus), 1 turnaround**
  - **Request phase: arbitration, resolution, address, decode, ack**
  - **Request-response transaction takes 3 or more of these**
- **Cache tags looked up in decode; extend ack cycle if not possible**
  - **Determine who will respond, if any**
  - **Actual response comes later, with re-arbitration**
- **Write-backs only request phase : arbitrate both data+addr buses**
- **Upgrades have only request part; ack'ed by bus on grant (commit)**

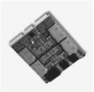
Bus Interface with Request Table

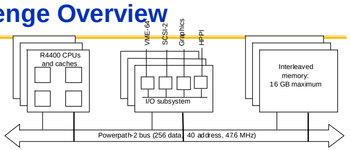


Bus Interface with Request Table



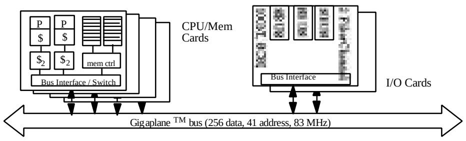
SGI Challenge Overview


(a) A four-processor board


R4400 CPUs and caches
VME-64
SCSI-2
Graphics
HPPI
I/O subsystem
Interleaved memory: 16 GB maximum
Powerpath-2 bus (256 data, 40 address, 47.6 MHz)
(b) Machine organization

- 36 MIPS R4400 (peak 2.7 GFLOPS, 4 per board) or 18 MIPS R8000 (peak 5.4 GFLOPS, 2 per board)
- 8-way interleaved memory (up to 16 GB)
- 4 I/O busses of 320 MB/s each
- 1.2 GB/s Powerpath-2 bus @ 47.6 MHz, 16 slots, 329 signals
- 128 Bytes lines (1 + 4 cycles)
- Split-transaction with up to 8 outstanding reads

SUN Enterprise Overview

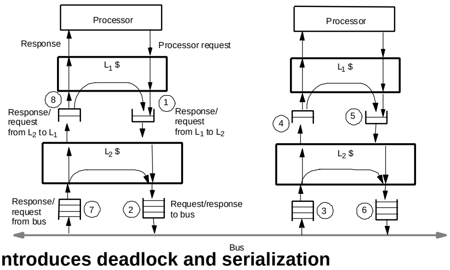


- Up to 30 UltraSPARC processors (peak 9 GFLOPs)
- Gigaplane^{TM} bus has peak bw 2.67 GB/s; upto 30GB memory
- 16 bus slots, for processing or I/O boards
  - 2 CPUs and 1GB memory per board
    - » memory distributed, unlike Challenge, but protocol treats as centralized
  - Each I/O board has 2 64-bit 25Mhz SBUSes

Multi-Level Caches with ST Bus

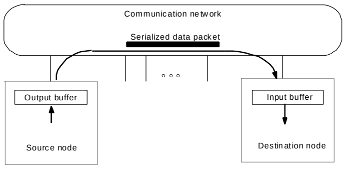
Key new problem: many cycles to propagate through hierarchy

- Must let others propagate too for bandwidth, so queues between levels



- Introduces deadlock and serialization problems

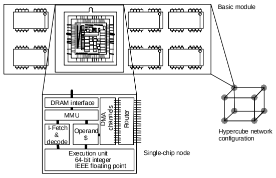
Network Transaction Primitive



- one-way transfer of information from a source output buffer to a dest. input buffer
  - causes some action at the destination
  - occurrence is not directly visible at source
- deposit data, state change, reply

NOW Handout Page 8

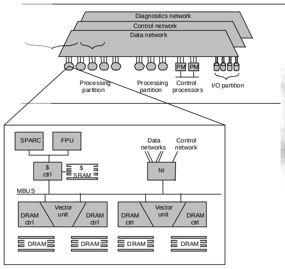
nCUBE/2 Machine Organization

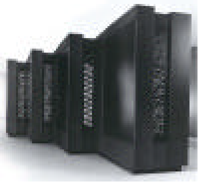


1024 Nodes

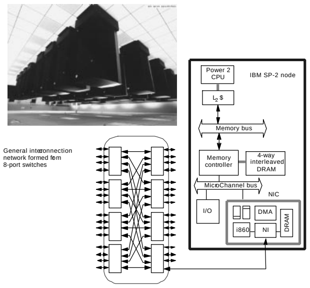
- Entire machine synchronous at 40 MHz

CM-5 Machine Organization

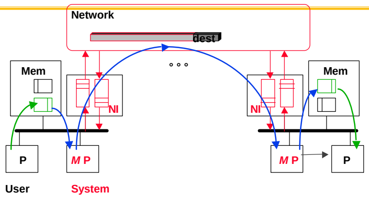




System Level Integration

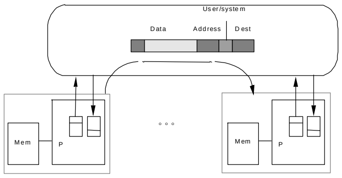


Levels of Network Transaction



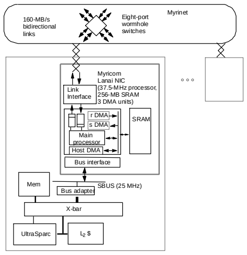
- User Processor stores cmd / msg / data into shared output queue
  - must still check for output queue full (or make elastic)
- Communication assists make transaction happen
  - checking, translation, scheduling, transport, interpretation
- Effect observed on destination address space and/or events
- Protocol divided between two layers

User Level Handlers



- Hardware support to vector to address specified in message
  - message ports in registers

Case Study: NOW


160-MB/s bidirectional links
Eight-port wormhole switches
Myrinet
Myricom Lanai NIC (37.5-MHz processor, 256-MB SRAM 3 DMA units)
Link Interface
r DMA
s DMA
Main processor
Host DMA
SRAM
Bus interface
Mem
Bus adapter
SBUS (25 MHz)
X-bar
UltraSparc
L_2 $

- General purpose processor embedded in NIC

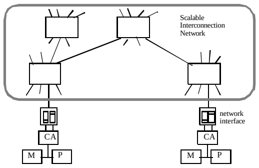
Scalable Synchronization Operations

- Messages: point-to-point synchronization
- Build all-to-all as trees
- Recall: sophisticated locks reduced contention by spinning on separate locations
  - caching brought them local
  - test&test&set, ticket-lock, array lock
    - » O(p) space
- Problem: with array lock location determined by arrival order => not likely to be local
- Solution: queue-lock
  - build distributed linked-list, each spins on local node

Scalable, High Perf. Interconnection Network

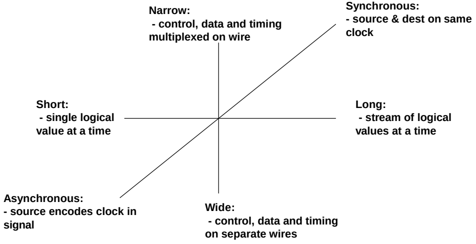
- At Core of Parallel Computer Arch.
- Requirements and trade-offs at many levels
  - Elegant mathematical structure
  - Deep relationships to algorithm structure
  - Managing many traffic flows
  - Electrical / Optical link properties
- Little consensus
  - interactions across levels
  - Performance metrics?
  - Cost metrics?
  - Workload?

=> need holistic understanding



Link Design/Engineering Space

- Cable of one or more wires/fibers with connectors at the ends attached to switches or interfaces


Narrow:
- control, data and timing multiplexed on wire

Synchronous:
- source & dest on same clock

Short:
- single logical value at a time

Long:
- stream of logical values at a time

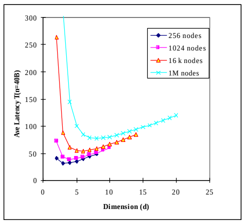
Asynchronous:
- source encodes clock in signal

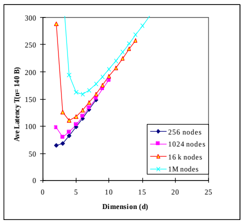
Wide:
- control, data and timing on separate wires

Summary

Topology	Degree	Diameter	Ave Dist	Bisection	D (D ave) @ P=1024
1D Array	2	N-1	N / 3	1	huge
1D Ring	2	N/2	N/4	2	
2D Mesh	4	2 ($N^{1/2}$ - 1)	2/3 $N^{1/2}$	$N^{1/2}$	63 (21)
2D Torus	4	$N^{1/2}$	1/2 $N^{1/2}$	$2N^{1/2}$	32 (16)
k-ary n-cube	2n	nk/2	nk/4	nk/4	15 (7.5) @n=3
Hypercube	n =log N		n	n/2	N/2 10 (5)

Latency with Equal Pin Count



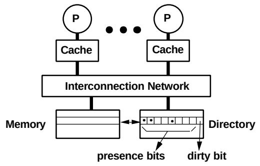


- Baseline d=2, has w = 32 (128 wires per node)
- fix 2dw pins => w(d) = 64/d
- distance up with d, but channel time down

Summary (Routing)

- Routing Algorithms restrict the set of routes within the topology
  - simple mechanism selects turn at each hop
  - arithmetic, selection, lookup
- Deadlock-free if channel dependence graph is acyclic
  - limit turns to eliminate dependences
  - add separate channel resources to break dependences
  - combination of topology, algorithm, and switch design
- Deterministic vs adaptive routing
- Switch design issues
  - input/output/pooled buffering, routing logic, selection logic
- Flow control
- Real networks are a ‘package’ of design choices

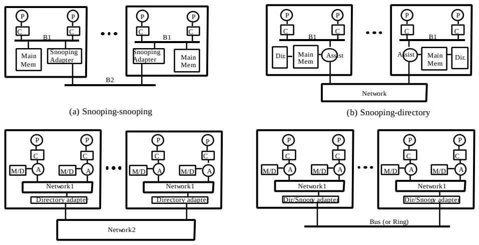
Basic Operation of Directory



- 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; ... }
  - ...

Example Two-level Hierarchies



(a) Snooping-snooping

(b) Snooping-directory

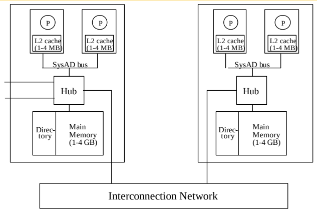
(c) Directory-directory

(d) Directory-snooping

Sharing Patterns Summary

- Generally, few sharers at a write, scales slowly with P
  - Code and read-only objects (e.g, scene data in Raytrace)
    - » no problems as rarely written
  - Migratory objects (e.g., cost array cells in LocusRoute)
    - » even as # of PEs scale, only 1-2 invalidations
  - Mostly-read objects (e.g., root of tree in Barnes)
    - » invalidations are large but infrequent, so little impact on performance
  - Frequently read/written objects (e.g., task queues)
    - » invalidations usually remain small, though frequent
  - Synchronization objects
    - » low-contention locks result in small invalidations
    - » high-contention locks need special support (SW trees, queueing locks)
- Implies directories very useful in containing traffic
  - if organized properly, traffic and latency shouldn't scale too badly
- Suggests techniques to reduce storage overhead

Origin2000 System Overview

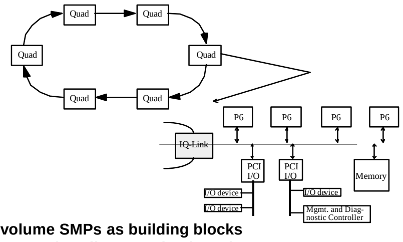


- Single 16”-by-11” PCB
- Directory state in same or separate DRAMs, accessed in parallel
- Upto 512 nodes (1024 processors)
- With 195MHz R10K processor, peak 390MFLOPS or 780 MIPS per proc
- Peak SysAD bus bw is 780MB/s, so also Hub-Mem
- Hub to router chip and to Xbow is 1.56 GB/s (both are off-board)

Summary

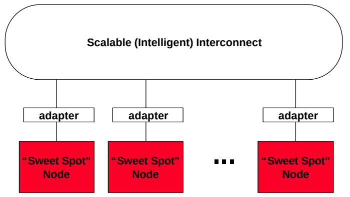
- In directory protocol there is substantial implementation complexity below the logical state diagram
  - directory vs cache states
  - transient states
  - race conditions
  - conditional actions
  - speculation
- Real systems reflect interplay of design issues at several levels
- Origin philosophy:
  - memory-less: node reacts to incoming events using only local state
  - an operation does not hold shared resources while requesting others

NUMA-Q System Overview



- Use of high-volume SMPs as building blocks
- Quad bus is 532MB/s split-transation in-order responses
  - limited facility for out-of-order responses for off-node accesses
- Cross-node interconnect is 1GB/s unidirectional ring
- Larger SCI systems built out of multiple rings connected by bridges

The Composibility Question

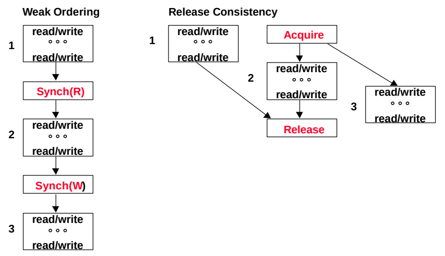


- Distributed address space => issue is NI
- CC Shared address space => composing protocols

Memory Consistency Model

- for a SAS specifies constraints on the order in which memory operations (to the same or different locations) can appear to execute with respect to one another,
- enabling programmers to reason about the behavior and correctness of their programs.
- fewer possible reorderings => more intuitive
- more possible reorderings => allows for more performance optimization
  - 'fast but wrong' ?

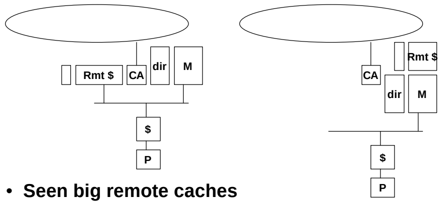
Preserved Orderings



Some Questions you might ask

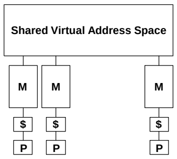
- Can all unnecessary communication be eliminated?
  - capacity-related communication?
  - false-sharing?
- How much hardware support can be eliminated?
- Can weak consistency models be exploited to reduce communication?
- Can we simplify hardware coherence mechanisms while avoiding capacity-related communication?

Overcoming Capacity Limitations



- Seen big remote caches
  - 32 MB on NUMA-Q
- What about using region of local mem as remote cache?
  - basic operation is to access mem. and check tag state
  - dispatch to specific protocol action

SAS w/o hardware support?

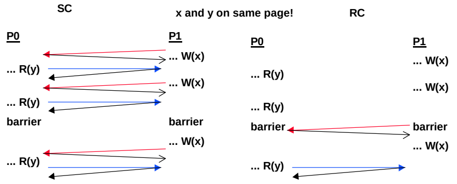


- Treat memory as fully-associative cache for global shared virtual address space
- Unit of coherence: page
- Basic components
  - Access control?
  - Tag and state check?
  - Protocol processing?
  - Communication?
- Problems?

Same virtual address represented at different physical addresses on each processor!
- what needs to be invalidated?
Inclusion??

Exploiting Weak Consistency

- So far in HW approaches
  - changes when invalidations must be processed
  - avoid stalling processor while invalidations processed
  - still propagate invalidations ASAP
- Can invalidations be avoided?


SC

x and y on same page!

RC

P0

P1

... W(x)

... R(y)

... W(x)

... R(y)

barrier

barrier

... W(x)

... R(y)

P0

P1

... W(x)

... R(y)

... W(x)

... R(y)

barrier

barrier

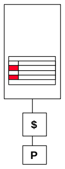
... W(x)

... R(y)

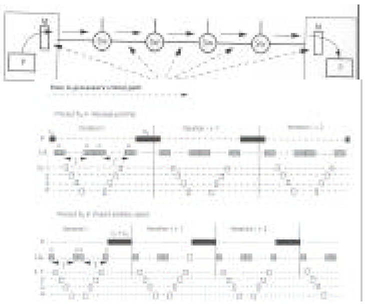
propagate inv. at synch points!

Middle Ground: Simple-COMA, Stache

- automatic migration at page level controlled in software
- fine grain access control in hardware
- page fault:
  - allocate page in local memory, but leave all blocks invalid
- page hit, cache miss:
  - access tag in parallel with memory access
    - » can be separate memory
  - physical address valid (not uniform)
  - on protocol transactions, reverse translate to shared virtual address
- No HW tag comparison. (just state)
- No local/remote check!



Communication pipeline



Approached to Latency Tolerance

- block data transfer
  - make individual transfers larger
- precommunication
  - generate comm before point where it is actually needed
- proceeding past an outstanding communication event
  - continue with independent work in same thread while event outstanding
- multithreading - finding independent work
  - switch processor to another thread

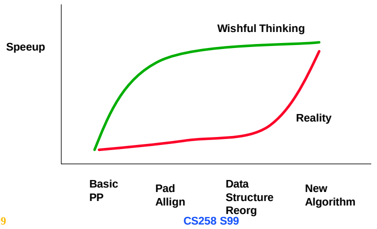
New Results and What's Ahead

Report from FCRC 99

- Theorists thinking about quantum computing
- Prog. linguists talking about cache optimizations
  - not cleaning up consistency
- Proc Architects are betting on speculation
  - branches, addresses, values, ...
- Performance Analysts squeezing bottlenecks
- Parallel Arch. dealing with scaling
  - speculation on coherence actions
  - optimistic multicast coherence
  - fundamentally limited by software/programmability
- John Hennessey - SAM and PostPC
- Jim Gray's Turing Award Lecture

Parallel Programming Effort (Singh)

- Optimizations for Portable Performance (SVM) help scalability on DSMs
- In both cases, it takes parallel algorithm development



Looking Forward

- The only constant is “constant change”
- Where will the next “1000x” come from?
  - it is likely to be driven by the narrow top of the platform pyramid serving the most demanding applications
  - it will be constructed out of the technology and building blocks of the very large volume
  - it will be driven by billions of people utilizing ‘infrastructure services’

Prognosis

- Continuing on current trends, reach a petaop/s in 2010
  - clock rate is tiny fraction, density dominates
  - translating area into performance is PARALLEL ARCH
- Better communication interface
  - 10 GB/s links are on the drawing board
  - NGIO/FutureIO will standardize port into memory controller
- Gap to DRAM will grow, and grow, and grow...
  - processors will become more latency tolerant
  - many instructions per thread with OO exec
  - many threads’
- Bandwidth is key
- Proc diminishing fraction of chip
  - and unfathomably complex

Continuing Out

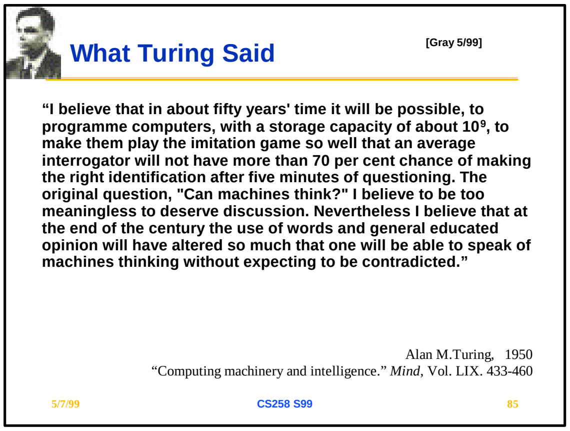
- Proc and Memory will integrate on chip
  - everything beyond embedded devices will be MP
  - PIN = Communication
- Systems will be a federation of components on a network
  - every component has a processor inside
    - » disk, display, microphone, ...
  - every system is a parallel machine
  - how do we make them so that they just work?

Fundamental Limits?

- Speed of light
  - Latency dominated by occupancy
  - occupancy dominated by overhead
    - » its all in the connectors
  - communication performance fundamentally limited by design methodology
    - » make the local case fast at the expense of the infrequent remote case
  - this may change when we a fundamentally managing information flows, rather than managing state
    - » we’re seeing this change at many levels

The Sages

- John Hennessey
  - only so much ILP, processors getting way too complex
  - soon every chip will be multiprocessor, we got to figure out how to make parallel software easier
  - focus on scalability, availability, management
  - post-PC era is emerging
    - » changes all the rules
    - » ubiquitous connectivity
    - » scalability, availability and management
      - it has to work
- Jim Gray

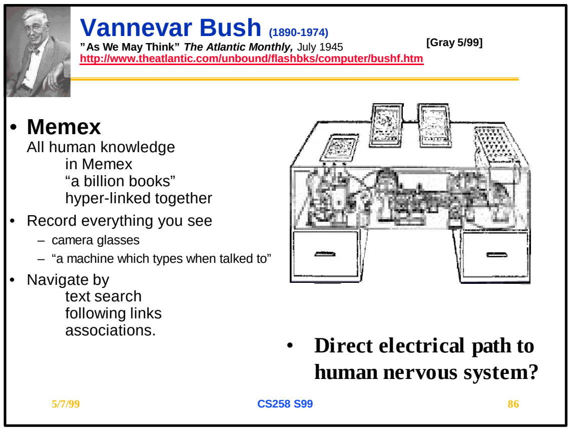


What Turing Said

[Gray 5/99]

“I believe that in about fifty years' time it will be possible, to programme computers, with a storage capacity of about 10^9, to make them play the imitation game so well that an average interrogator will not have more than 70 per cent chance of making the right identification after five minutes of questioning. The original question, "Can machines think?" I believe to be too meaningless to deserve discussion. Nevertheless I believe that at the end of the century the use of words and general educated opinion will have altered so much that one will be able to speak of machines thinking without expecting to be contradicted.”

Alan M.Turing, 1950
“Computing machinery and intelligence.” *Mind*, Vol. LIX. 433-460



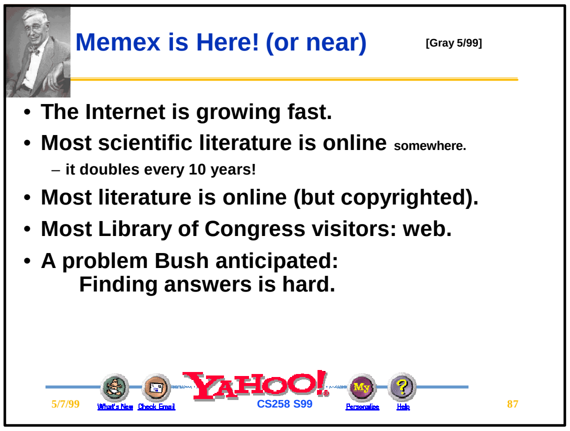
Vannevar Bush (1890-1974)

”As We May Think” *The Atlantic Monthly,* July 1945
http://www.theatlantic.com/unbound/flashbks/computer/bushf.htm

[Gray 5/99]

- **Memex**
  All human knowledge
  in Memex
  “a billion books”
  hyper-linked together
- Record everything you see
  - camera glasses
  - “a machine which types when talked to”
- Navigate by
  text search
  following links
  associations.

- **Direct electrical path to human nervous system?**

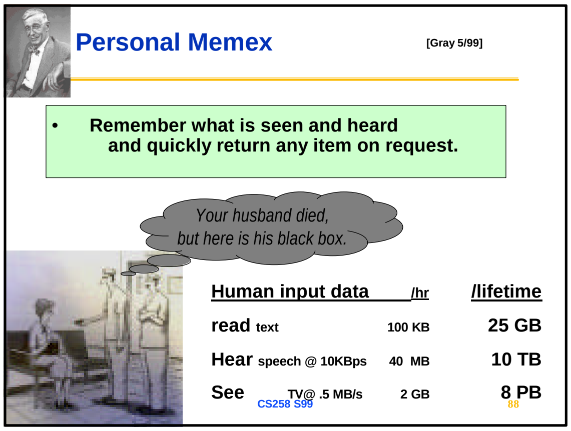


Memex is Here! (or near)

[Gray 5/99]

- **The Internet is growing fast.**
- **Most scientific literature is online** somewhere.
  - it doubles every 10 years!
- **Most literature is online (but copyrighted).**
- **Most Library of Congress visitors: web.**
- **A problem Bush anticipated:**
  Finding answers is hard.

YAHOO!
What's New
Check Email
Personalize
Help



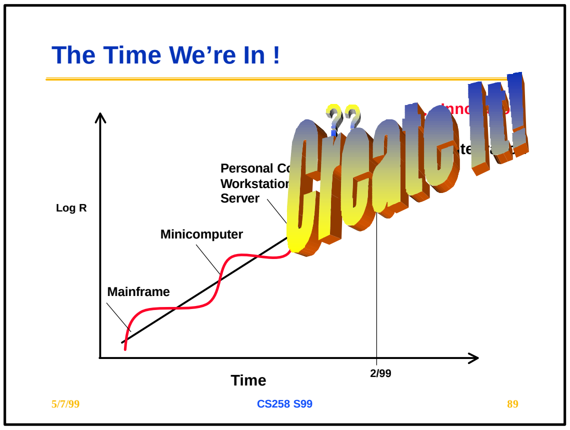
Personal Memex

[Gray 5/99]

- **Remember what is seen and heard and quickly return any item on request.**

*Your husband died,
but here is his black box.*

Human input data	**/hr**	**/lifetime**
read text	100 KB	25 GB
Hear speech @ 10KBps	40 MB	10 TB
See TV@ .5 MB/s	2 GB	8 PB



The Time We’re In !