

Cluster-Based Computing

Jay R. Moorman

ECE 497 SSL

September 2, 1999

Electrical and Computer Engineering
University of Illinois at Urbana-Champaign

Overview of Presentation

- History / Evolution of Computing
- Cache Coherence Schemes
 - DASH
- Fault Containment
 - OS support (HIVE)
 - HW support
- Discussion

PART I

9/2/99

History of Computing

- Perspectives on Supercomputing: Three Decades of Change
 - Paul R. Woodward
- ECE 291 Lecture Notes
 - John Lockwood

The Computer Evolution . . .

1642: Blaise Pascal invents Mechanical Calculator



... The Computer Evolution ...

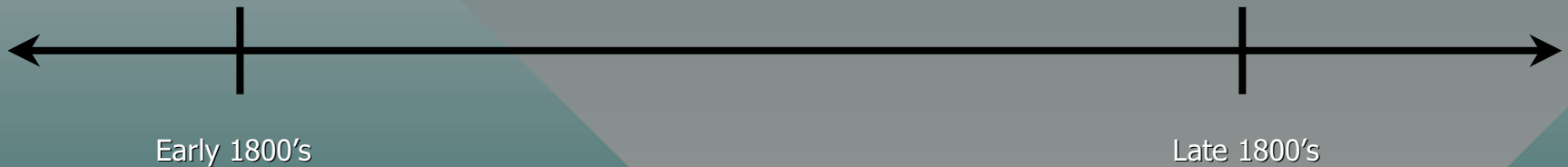


1700's

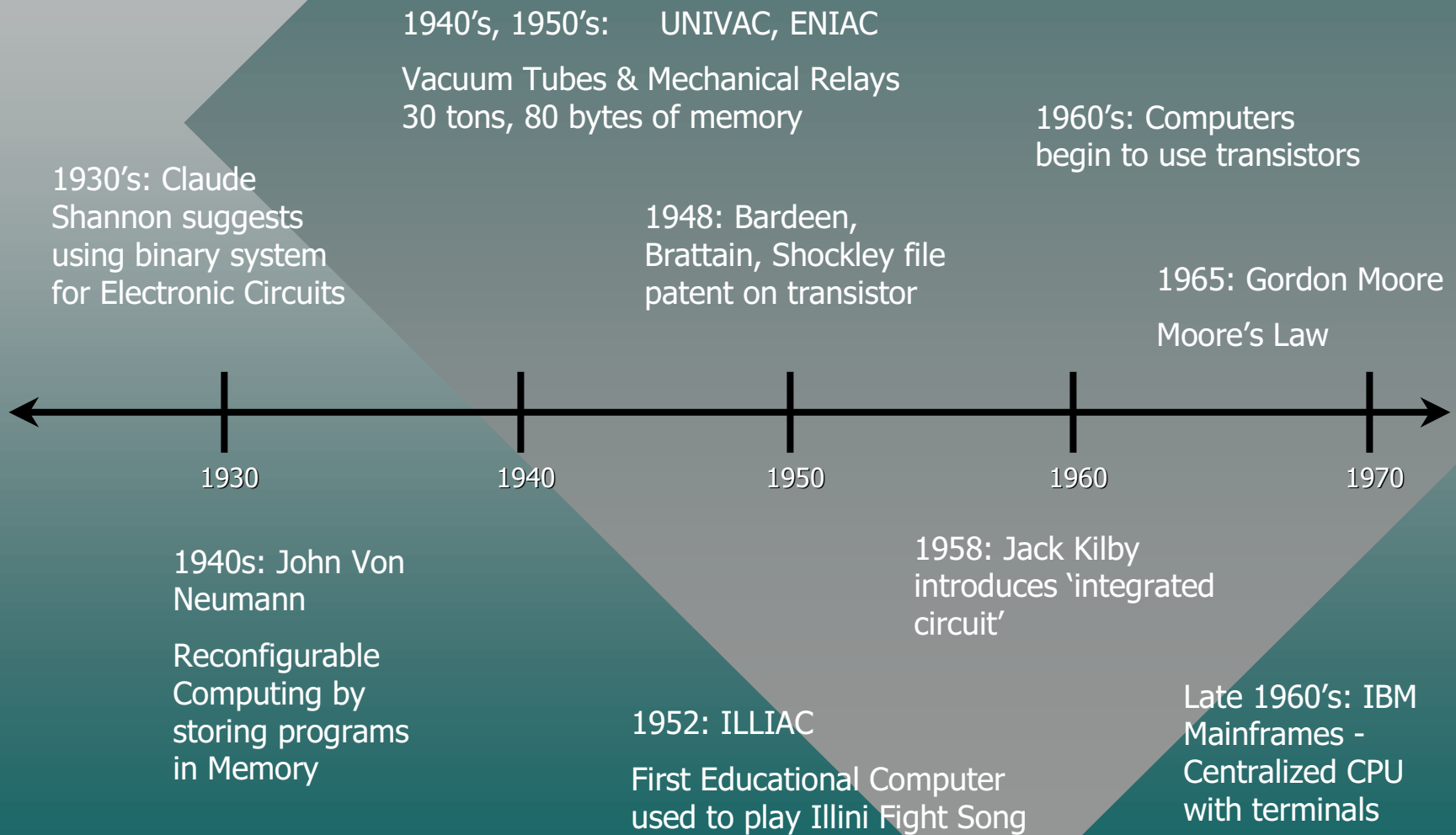
... The Computer Evolution ...

1830: Charles Babbage
invents Steam Powered
Analytical Engine

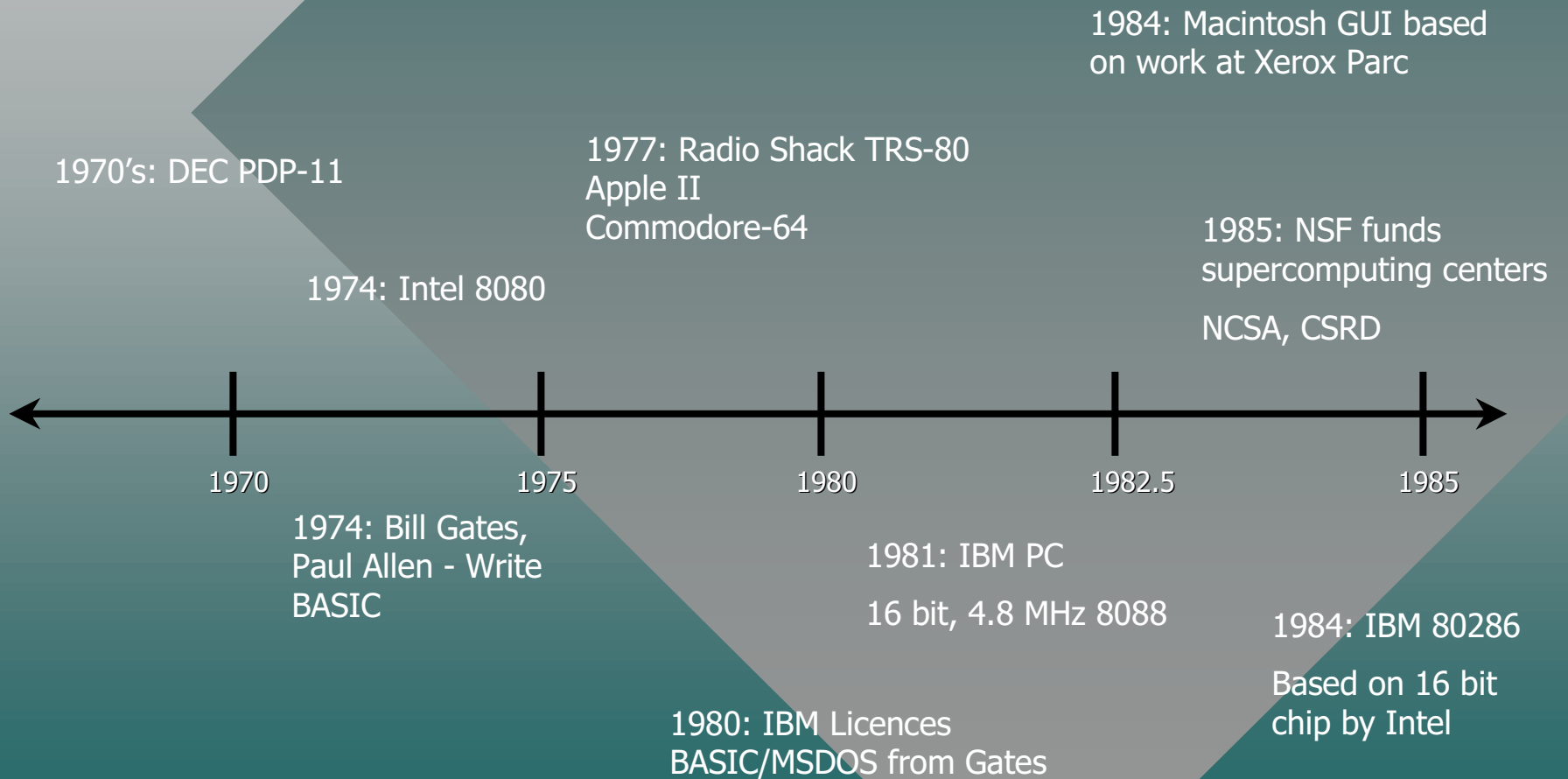
1880's: John H. Patterson: First
Application device - Mechanical
Cash Register



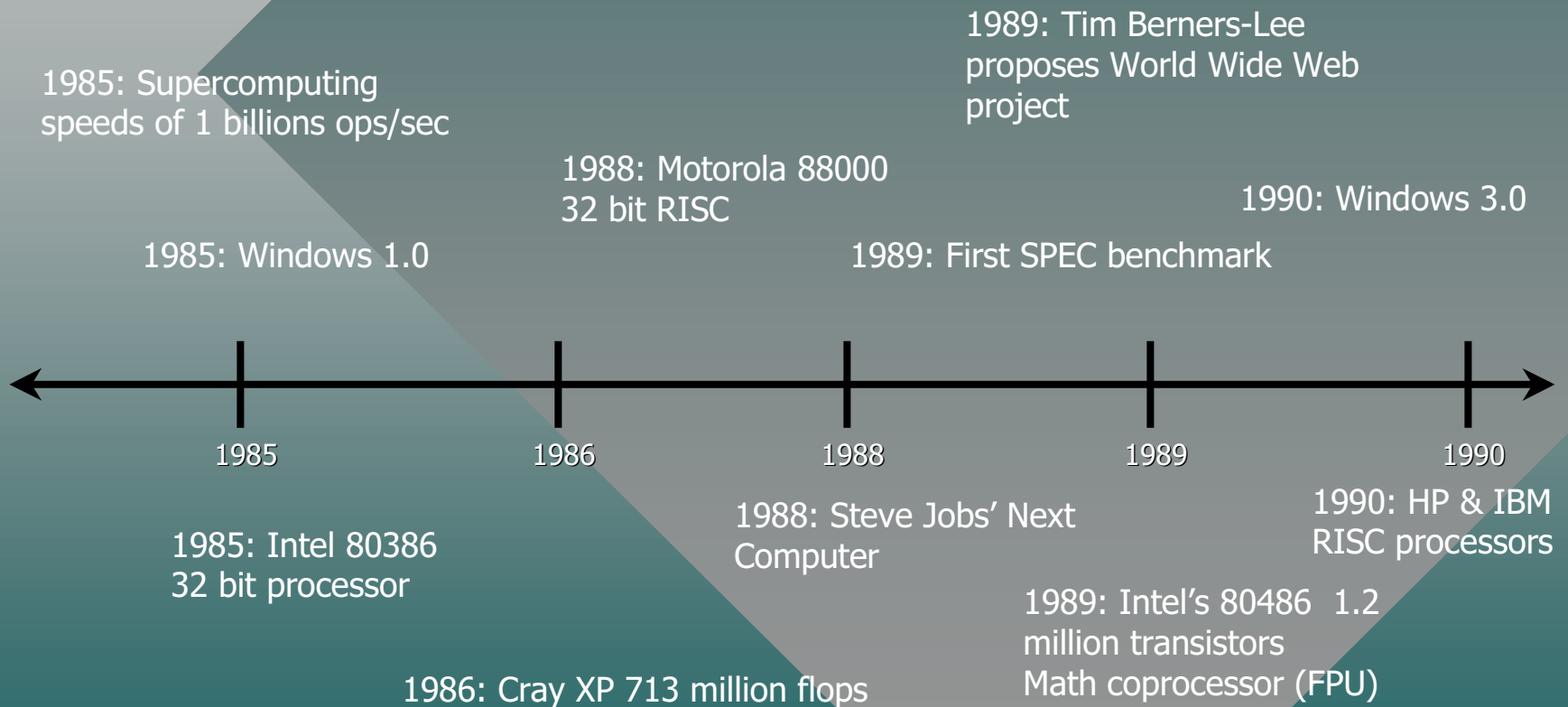
... The Computer Evolution ...



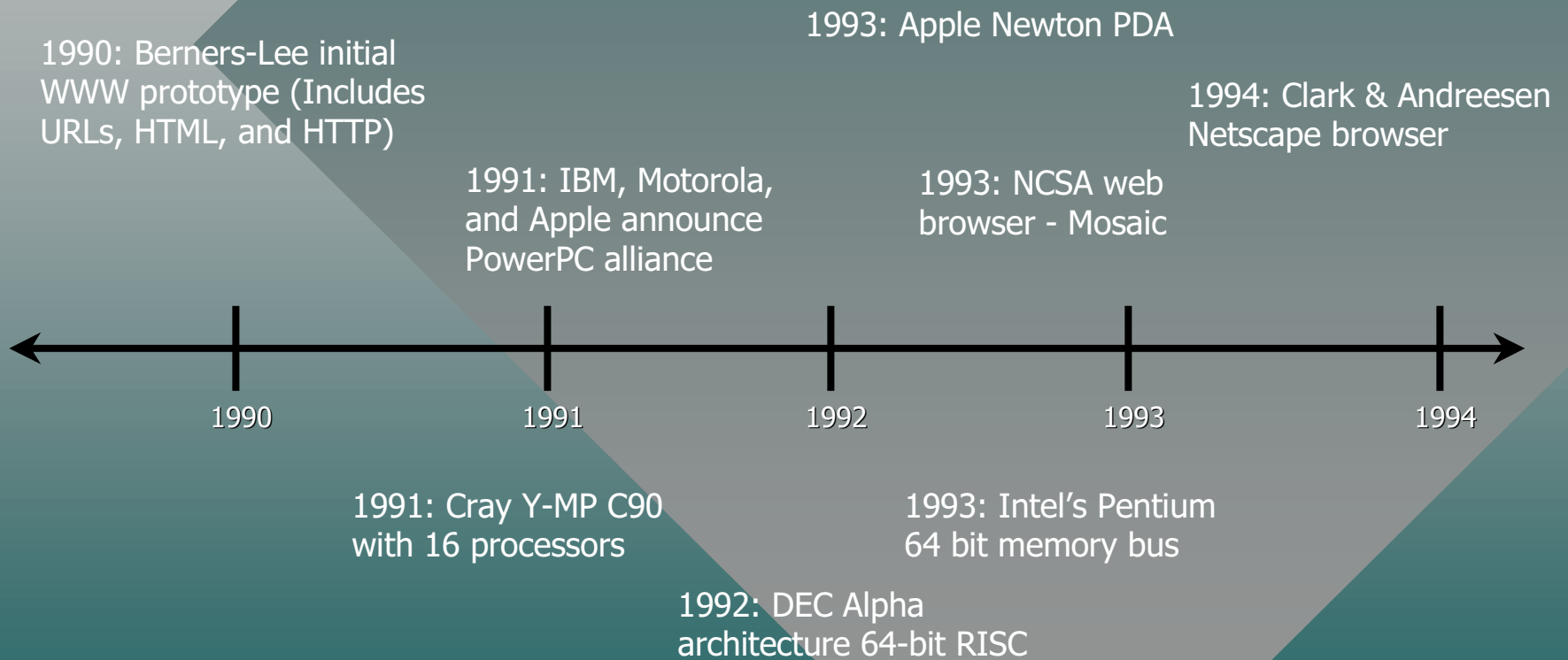
... The Computer Evolution ...



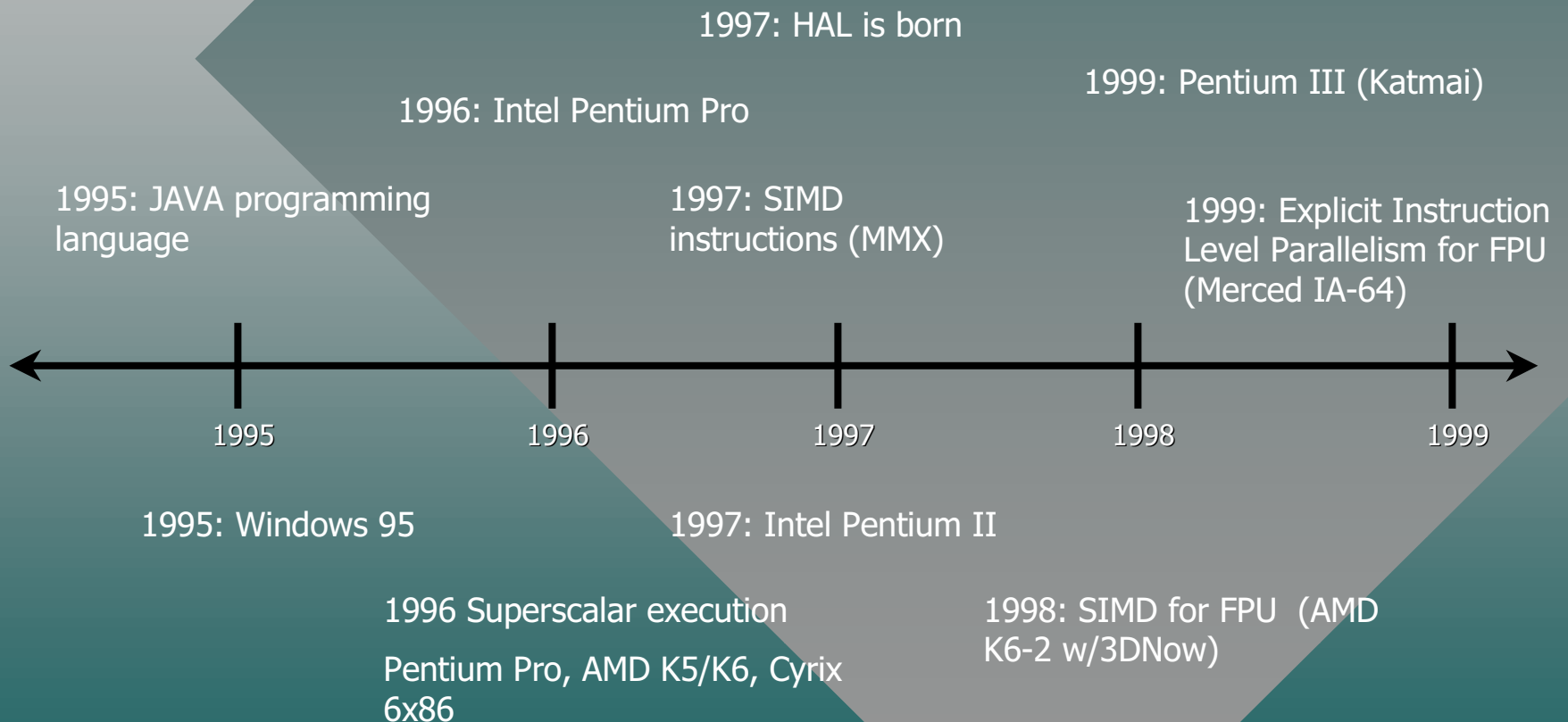
... The Computer Evolution ...



... The Computer Evolution ...



... The Computer Evolution ...



PART II

Cache Coherence Schemes

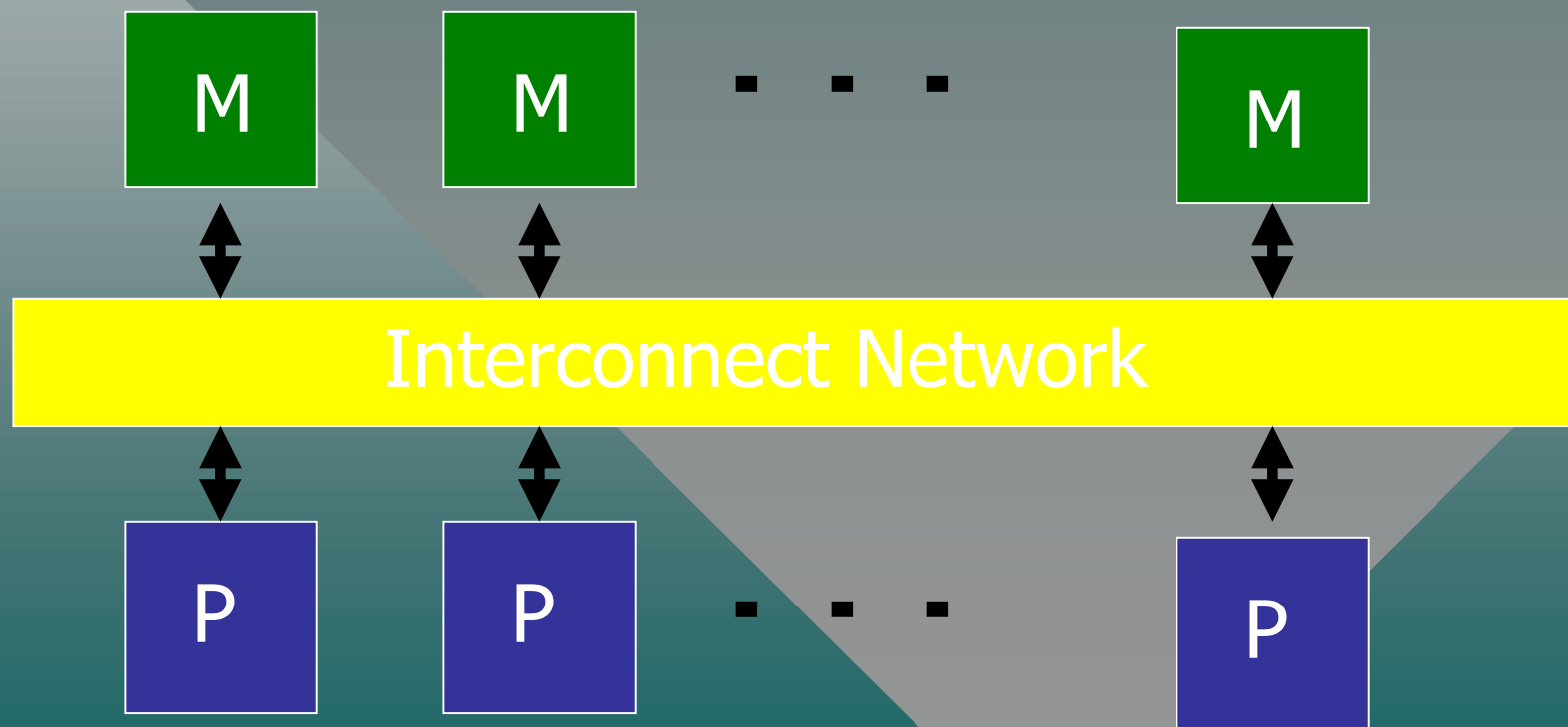
- A Survey of Cache Coherence Schemes for Multiprocessors
 - Per Stenstrom
- The Directory-Based Cache Coherence Protocol for the DASH Multiprocessor
 - Lenoski, Laudon, Gharachorloo, Gupta, Hennessy

Cache Coherence

- Maintain Data Consistency
- Shared-Memory Multiprocessors
- Use Hardware, Software, or Both

Shared Memory Multiprocessor

- Collection of Processors and Memory

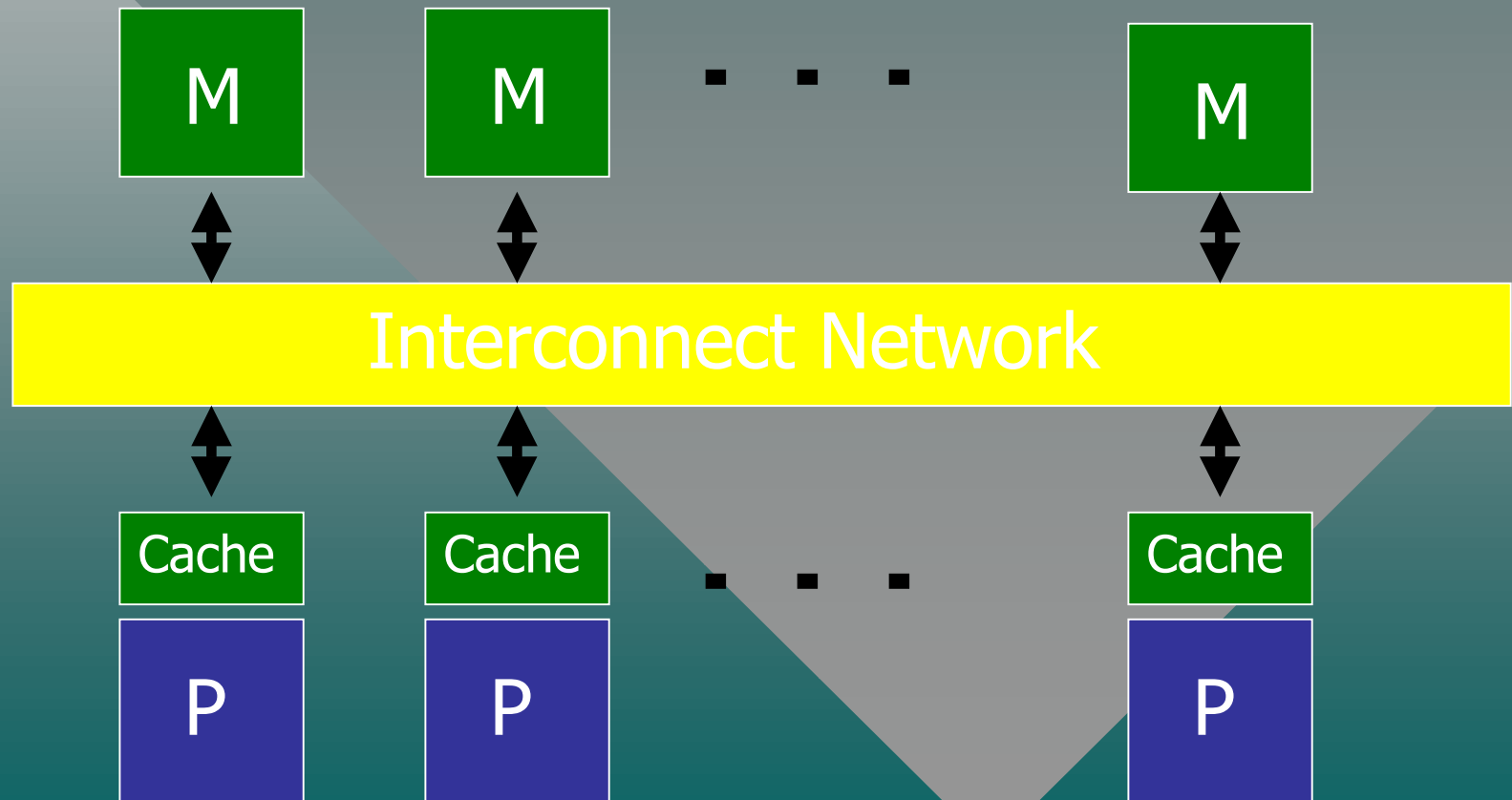


Shared Memory Problems

- Memory Contention
 - Serialize memory requests
- Communication Contention
 - Interconnect Network
- Latency
 - Long times for memory accesses

Cache Memory

- Locality of Memory References
 - Temporal Locality (Time)
 - Spatial Locality (Space)



The Problem: Cache Coherence

- Process Communication via Shared Memory
- Multiple Copies in Multiple Caches
- Copies Must Be Kept Coherent
 - Shared read-write data structures

The Solution: Cache Coherence

- Hardware Based Protocols
 - Snoopy cache protocols
 - Directory schemes
 - Cache-coherent network architectures
- Software Based Protocols
 - Limit caching to safe times
 - Cacheability marking by compiler

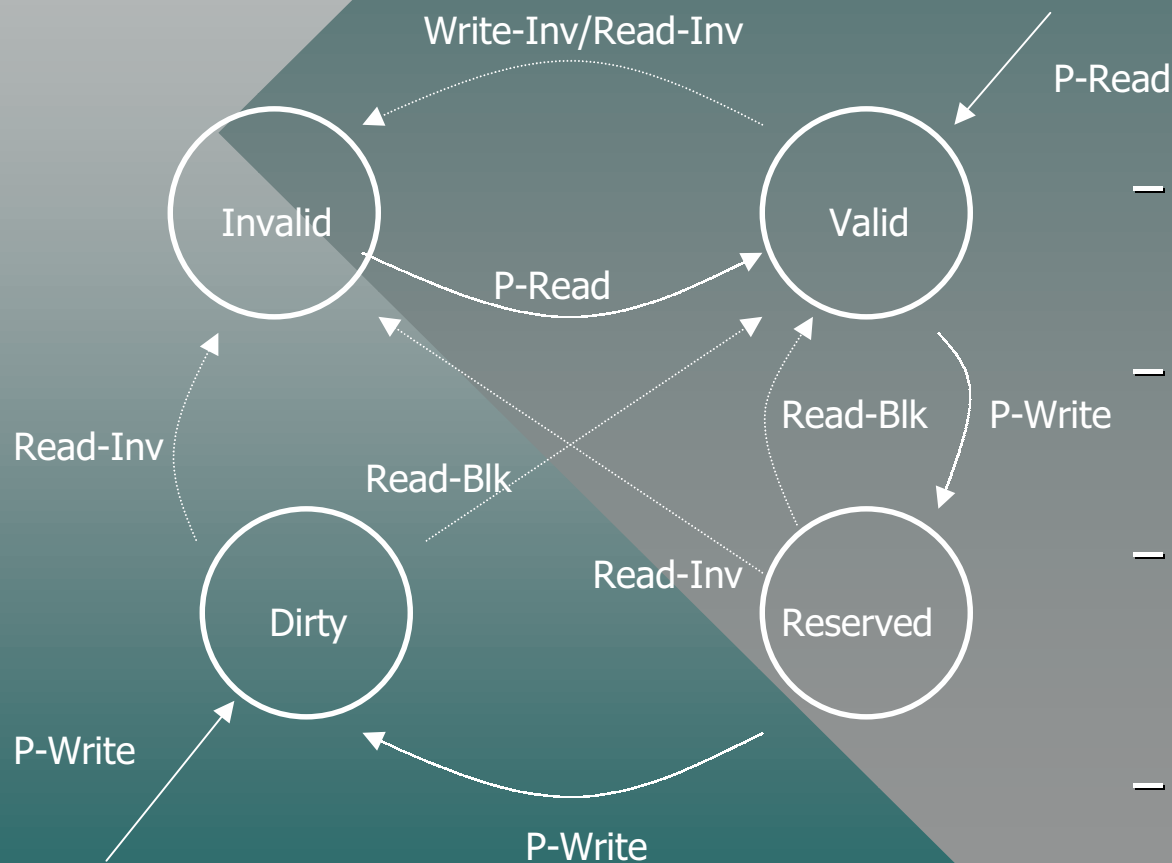
General Cache Coherence

- Hardware
 - Detects inconsistency
 - Applies algorithm to make consistent
- Policies (Consistency Commands)
 - Write-invalidate
 - Altered cache invalidates all other cache copies
 - Write-update
 - Altered cache sends updates to other cache copies

Snoopy Cache Protocols

- Consistency Commands are Broadcast
- Each Cache Listens to Commands (Snoops)
- Broadcast Interconnect is Necessary

Write-Invalidate Snoopy Cache



– Invalid -> Inconsistent Copy

– Valid -> Consistent Copy w/Memory

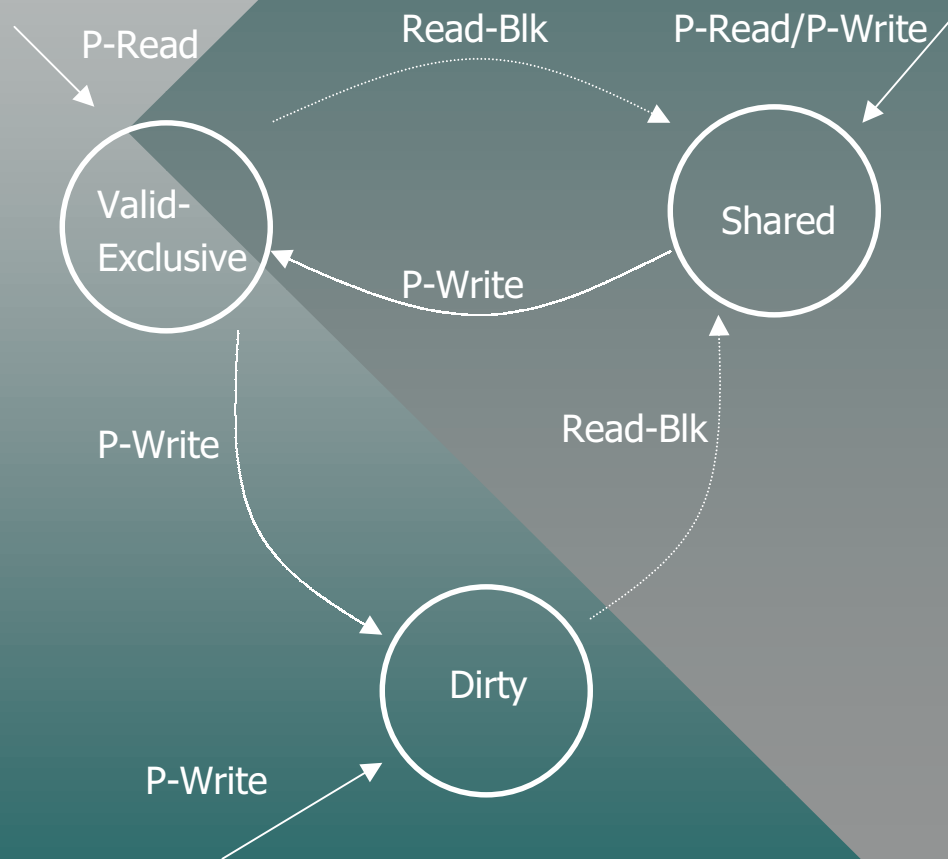
– Reserved -> Only Consistent Copy w/Memory

– Dirty -> Only Copy of Modified Data

Snoopy Cache - Write Once

- Read Miss
 - Memory provides consistent copy
 - Dirty cache provides copy
- Write Hit
 - Local copy becomes dirty
 - Broadcast Write-Inv command
- Write Miss
 - Copy supplied by memory/dirty cache
 - Broadcast Read-Inv command
 - Local copy is dirty

Write-Update Snoopy Cache



– Shared -> Shared
Consistent Copy

– Valid Exclusive -> Only
Consistent Copy
w/Memory

– Dirty -> Only Copy of
Modified Data

Snoopy Cache - Firefly

- Read Miss
 - Shared caches provide synchronized copy
 - Dirty cache provides copy
 - Memory provides exclusive copy
- Write Hit
 - Local copy becomes dirty
 - Broadcast update if shared copy
- Write Miss
 - Copy supplied by memory as dirty
 - Supplied by shared cache
 - Broadcast update

General Snoopy Cache Issues

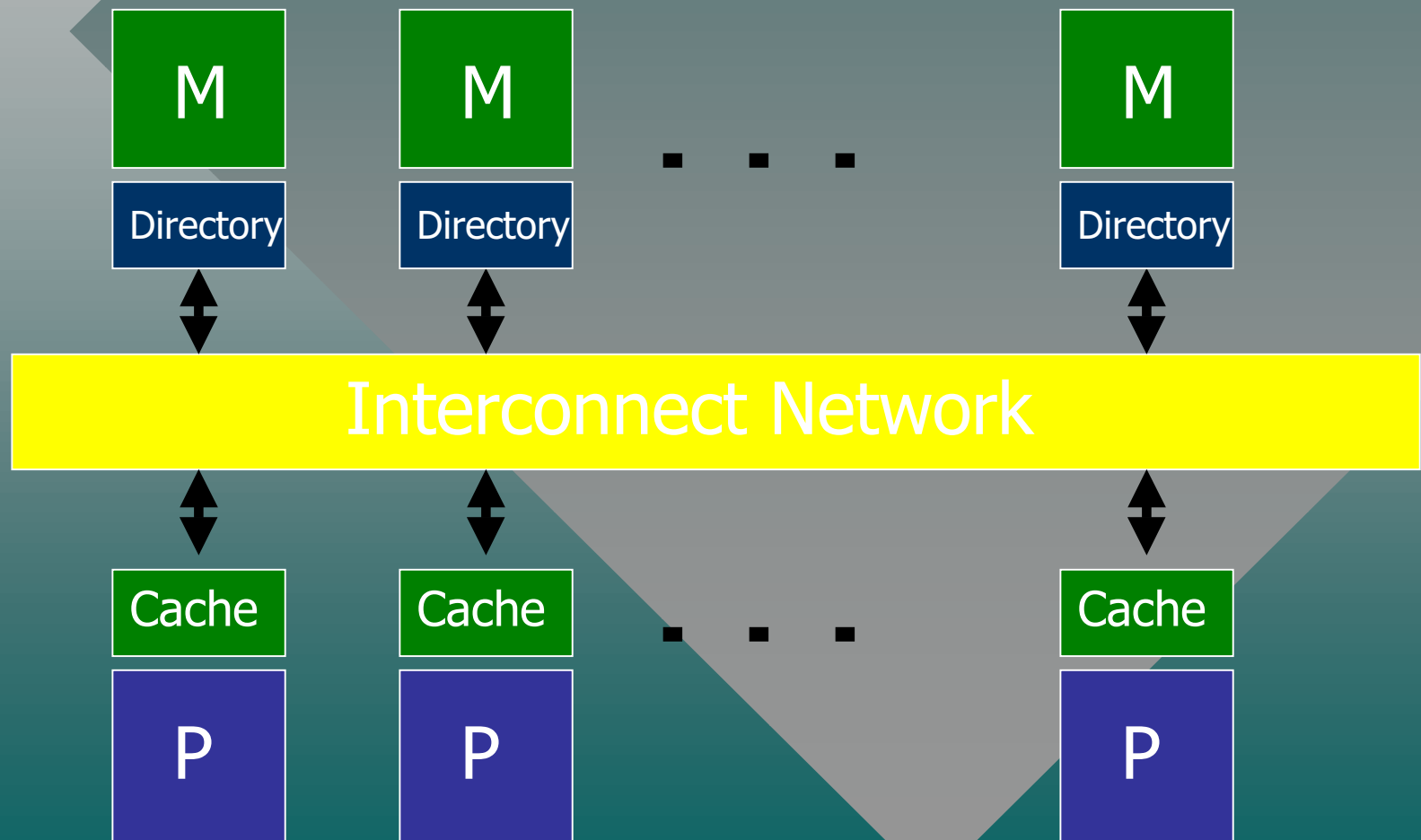
- Easy to implement
 - Used by commercial bus-based multiprocessors
 - Snoopy Cache vs. Uniprocessor Cache
 - Cache controller (FSM with coherence protocol)
 - Cache directory (Store state of each block)
 - Bus controller (Bus snooping to monitor commands)
- Extensions
 - Write-Invalidate -> Read Broadcast (Reduce Bus Traffic)
 - Write-Update -> Do not update unused copies
- Number of Processors Limited by Bus

Directory Protocols

- Consistency Commands sent to Cached Copies
 - Limits network traffic
- Directory
 - Used to track which caches have copies
- Used for General Interconnection Networks

Directory Based Cache Coherency

- Directory -> Bit Vector to specify which caches have copies



General Directory Issues

- Read Misses
 - Request to memory
 - Memory sends request to dirty cache
 - Cache writes back copy
 - Memory (or Dirty Cache) supplies copy to cache
- Expensive for Many Processors
 - Large directory bit vectors
 - Reduce directory size
 - Restrict number of cache pointers
 - Linked List of caches (Chained)
 - Trade-off (Network traffic vs. Directory size)

Cache-coherent Net Architectures

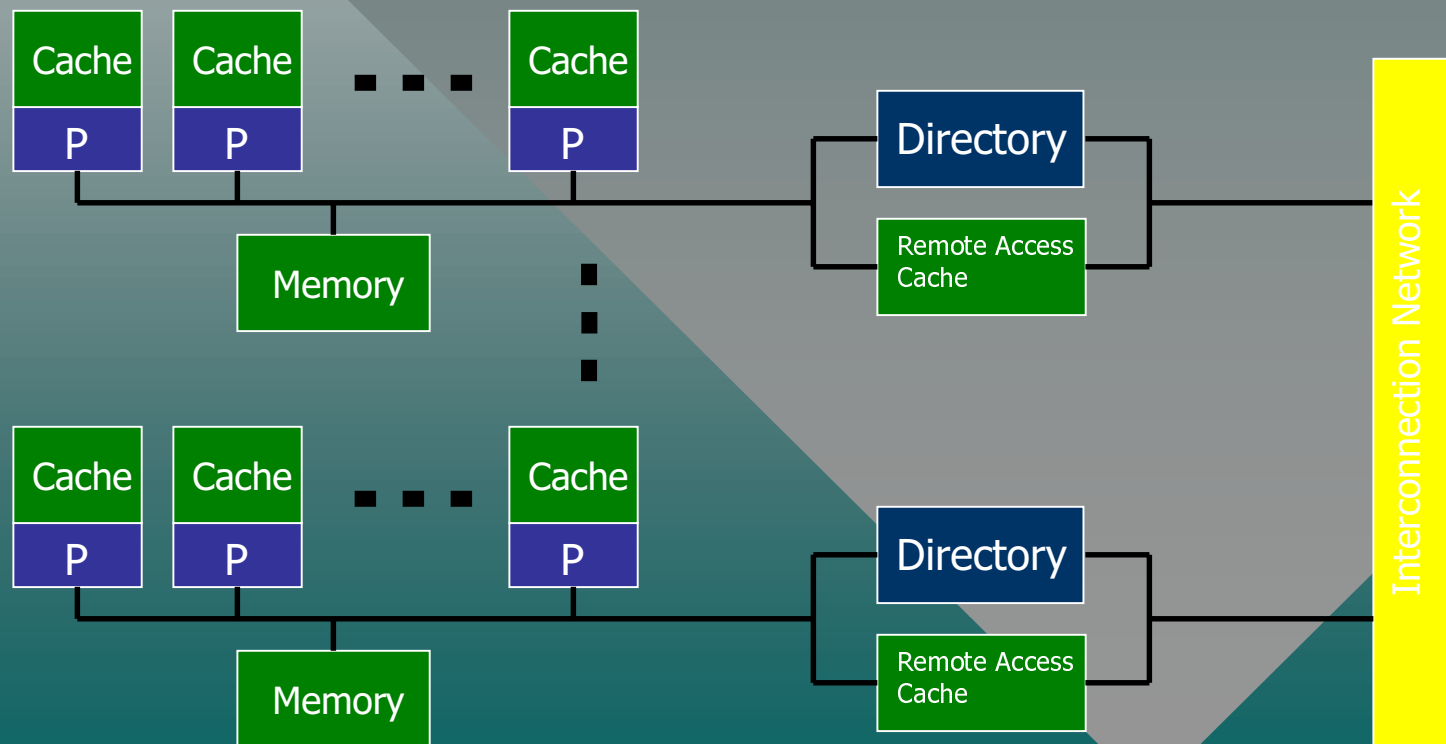
- Extensions to Bus-Based Protocols
 - Less costly than directory schemes
- Hierarchical Cache-Coherence Protocol
 - Hierarchy of caches/buses
 - Cache contains a copy of blocks cached beneath it
 - Vertical invalidate propagation
 - Grid of buses
 - Processor for each switch
 - Memory module for each grid column
 - Invalidations propagate to grid row then grid column
 - Data Diffusion
 - Hierarchy of buses with large processor caches
 - Higher order caches only contain state information

Software Based Cache Coherency

- Limit data caching
 - Analyze program at compile time
 - Mark variable cacheable/noncacheable
 - Find data dependencies
 - Break program into Computational Units
 - Invalidate variable across computational units
 - Within unit can have different policies
 - » Variables - Cache all or none
 - » Selective Invalidation
 - » Use timestamps

DASH

- Scalable Shared Memory Multiprocessor
- Distributed Directory Based Cache-Coherence Protocol
 - Point-to-point messages
 - No single serialization or control point



Directory Architecture for Shared Memory

- Interconnect
 - High bandwidth & low latency
- Memory
 - Distributed among nodes
- Nodes
 - Cluster of high performance processors/cache
 - Common cache for remote access
 - Directory controller (inter-node consistency)
 - Bus based snoopy scheme (intra-node consistency)

Directory Controller

- Directory Memory
 - For nodes portion of shared memory
 - Stores all remote nodes caching block
 - Uses point-to-point commands (No broadcasts)
- Not Dependent on Interconnection Technology

Design Issues

- Correctness
 - Memory consistency model
 - Strong consistency vs. Weak consistency
 - Deadlock
 - Error Handling (Fault tolerance)
- Performance
 - Latency
 - Bandwidth
 - Limited by serialization of messages (queuing delay)
 - Limited by traffic generated
- Distributed Control & Complexity

DASH Node Architecture

- Processors/Cache
 - For high performance processors (Silicon graphics)
 - 64k L1 i-cache and 64k L1 d-cache for each processor
 - 256k L2 d-cache (bus snooping and cluster consistency)
 - Pipelined synchronous bus
 - » Uses retries for long latency remote transactions
- Directory controller board
 - Directory memory
 - » Bit vector (1 bit state) for each cluster
 - Remote message controller
 - Network interface
- Remote Access Cache
 - Holds state of remote requests
 - Buffers network replies

DASH Cache Coherence

- Invalidation-Based Protocol
 - Memory Block State (in directory)
 - uncached-remote
 - shared-remote
 - dirty-remote
 - Home cluster kept coherent by snoopy protocol
 - Owning cluster makes directory changes
 - Home cluster
 - Dirty remote cache

DASH Coherence Primitives

- Read Requests
 - Perform local read (L1, L2, Memory)
 - Remote read
 - » Allocate RAC entry and query home cluster
 - » Owning cluster responds (NAK if just replied to another request)
- Read-Exclusive Requests
 - Acquire sole ownership
 - Local then Remote request (as above)
 - Each caching cluster is sent invalidation
 - » Acknowledge messages sent to requesting cache
 - » If reply from dirty cache data is sent to home cluster
- Writeback Requests
 - Local write to main memory
 - Remote message sent to update home cluster

DASH Correctness

- Release consistency (weak)
 - Processor can continue after issuing write
 - Order of memory accesses is guaranteed by:
 - Synchronization release
 - Fence operations (Block/Barrier)
- Deadlock
 - Hardware and protocol features
 - Message delivery without deadlocks
 - Divide request vs. reply messages
- Error Handling
 - ECC on main memory
 - Parity on directory memory
 - Length checking on network messages
 - Inconsistent bus checking

Additional DASH Issues

- Additional Operations
 - Queue based lock primitives (Avoid hot spots)
 - Prefetch operations (Hide latency)
- Scalability
 - Directory scheme (bit vector) - doesn't scale well
- Validation
 - Software simulation of DASH
 - Existing parallel programs
 - Test scripts

PART III

Fault Containment

- Hive: Fault Containment for Shared-Memory Multiprocessors
 - Chapin, Rosenblum, Devine, Lahiri, Teodosiu, Gupta
- Hardware Fault Containment in Scalable Shared-Memory Multiprocessors
 - Teodosiu, Baxter, Govil, Chapin, Rosenblum, Horowitz

Hive - Operating System

- Shared Memory Systems
 - Reliability
 - Fault crashes entire system
 - Scalability
 - SMP Operating Systems don't scale well
- Internal Distributed System
 - Independent kernels (cells)
 - Memory share among cells
 - Contain faults within the faulty cell

- HIVE Advantages

- Reliability

- Contained faults only affect processes with resources in that cell

- Scalability

- Few kernel resources shared across cells
 - Increase number of kernel cells

- HIVE Challenges

- Fault containment

- Wild writes that corrupt memory

- Resource sharing

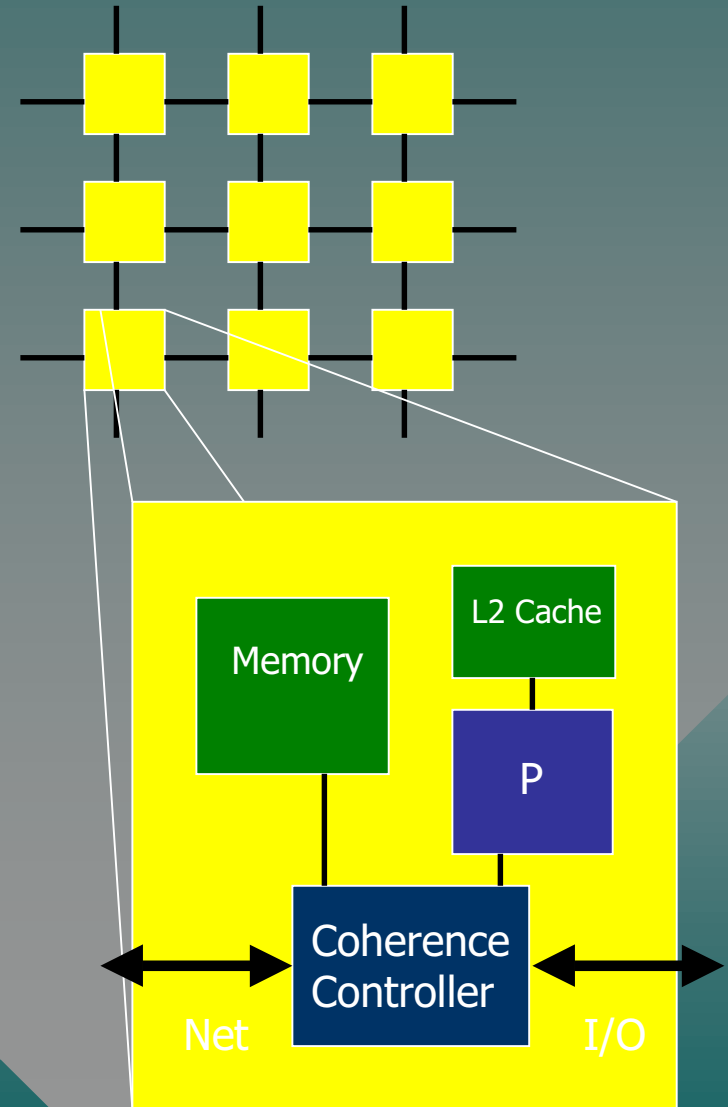
- Share across cell boundaries

- Single-system image

- Cell cooperation for standard OS for applications

FLASH Architecture

- CC-NUMA
- Multiple Nodes in Mesh Network
- Node
 - Processor/Cache
 - Memory
 - Coherence Controller



Faults

- Hardware
 - Node Failure
 - Halts processor
 - Node memory becomes inaccessible
 - Memory cached is lost
 - Memory Fault Model
 - Unaffected processes/memory continue
- Software
 - Wild writes

Hive Architecture

- Divided into Kernel Cells
 - Each cell owns a range of nodes
 - Independent Operating System
 - Manages processors, memory, I/O devices
- Fault Containment
- Resource Sharing

Hive Fault Containment

- Firewall Hardware
 - Defend memory pages against wild writes
- Discard Failed Cell Writable Pages
 - Prevent corrupt data from being read and used
- Aggressive Failure Detection
 - Use a distributed agreement protocol

OS Fault Containment

- Sending Bad Messages
 - Message checking and reply timeouts
- Providing Bad Data
 - Careful reference protocol
 - Checks for possible error conditions
- Causing Errored Remote Writes
 - Protect local memory pages
 - Wild writes
 - Corrupt pages cannot be read
 - Damaged pages are discarded
 - Failure detection
 - › Cells monitor each other (Hint alert)
 - › Consensus algorithm to reboot failed cell

Hive Resource Sharing

- User-level process (Wax)
 - Complete up-to-date view of system (Centralized)
 - Can run multithreads on different cells (Distributed)
 - If Cell running Wax fails, Wax restarts and builds a new view of the state of the system
- Share Processors/Memory across Cells
 - Logical-Level Sharing
 - Globally shared file buffer cache
 - Physical-Level Sharing
 - Transfer control of page frame to another cell

Hive Implementation

- Fault containment
 - Internal distributed system
 - Careful reference protocol
 - Wild write defense
 - Failure hints and recovery
- Memory Sharing (Without weakened fault containment)
 - Logical level (file data & anonymous data)
 - Physical level
- Wax - Not Implemented

Additional Hive Issues

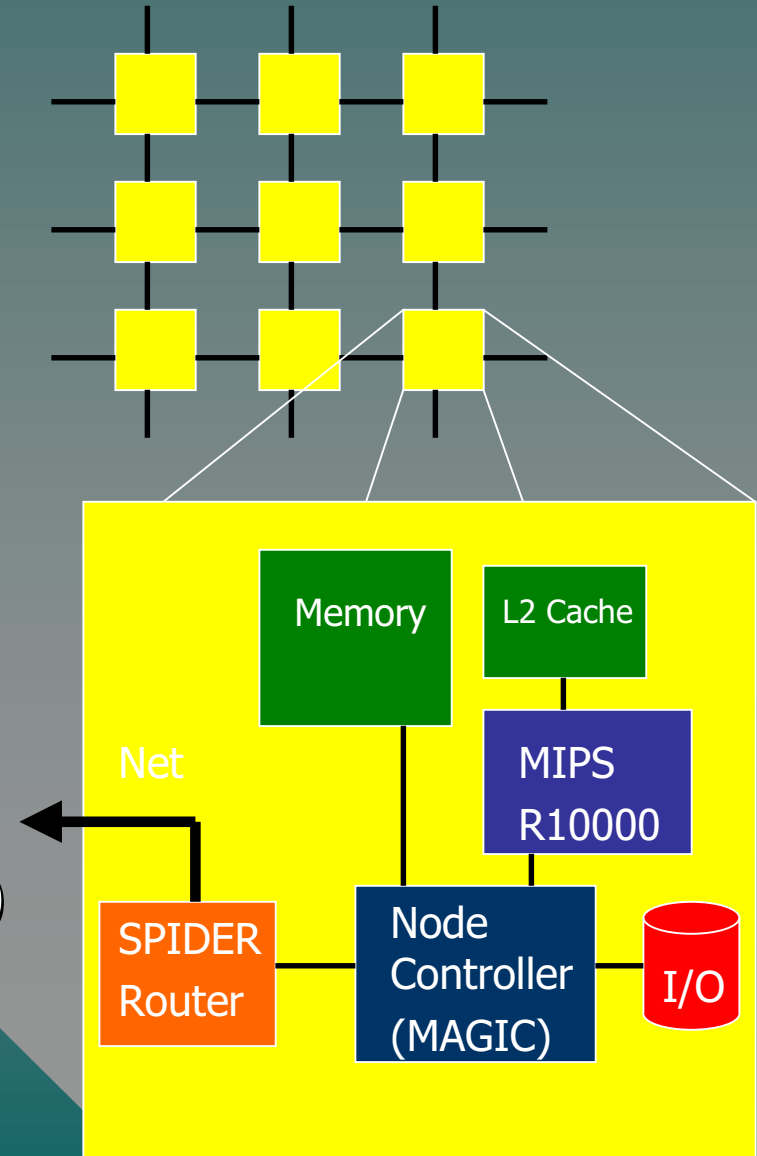
- RPC Optimization
 - Hardware message support to coherence controller - reduce intercell communication latency
- Simulations
 - SimOS (Model of FLASH hardware)
 - Overhead for Hive architecture is negligible
- Architecture Advantages
 - Heterogenous resource management
 - Support for CC-NOW (Network of Workstations)

Hardware Fault Containment

- Current Shared-Memory Multiprocessors are Vulnerable to Faults
- Limit Fault Damage to Part of Machine
- Distributed Recovery Algorithm
 - Allows normal operations to resume in the functioning parts of the machine
- Work in Conjunction w/ Fault Containment OS (Hive)

FLASH Hardware

- CC-NUMA
- Actual Topology is Hierarchical Hypercube
- Node
 - MIPS R10000
 - Cache
 - Memory
 - Node Controller (MAGIC)
 - Router (SPIDER)



Fault Impact

- Increased Probability of HW Failure
 - Partial power supply
 - Cooling system losses
- Node Failures
 - Affect on system
 - Node resources inaccessible
 - Other process may stall
 - Interconnect becomes congested (buffers fill)
 - Limit affect via Node Maps
 - Availability of all nodes on the system
 - Kept up-to-date by recovery mechanism
 - Recovery algorithm must reprogram routers bordering a failed node to avoid interconnect congestion

... Fault Impact ...

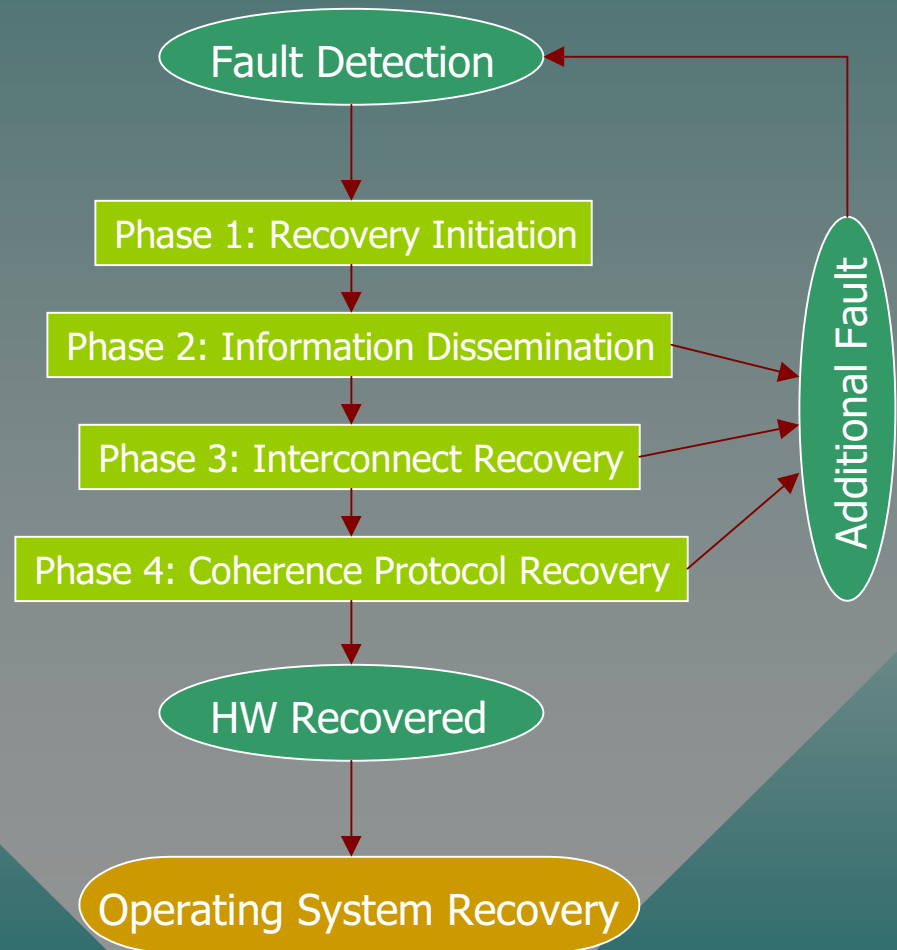
- Interconnect Failures
 - Affect on system
 - Loss of link/router reduces connectivity
 - » Static routes
 - Lost or truncated packets
 - Limit affect
 - Reprogram routing tables during recovery
 - Cache coherence and OS deals with lost/truncated packets
- Shared Memory
 - Affect from failed node
 - Cache line (home) becomes inaccessible
 - Incoherent cache line
 - » Failed cached copy or Packet with data lost
 - Deadlock

... Fault Impact

- Limit affect on shared memory
 - Recovery algorithm breaks deadlock
 - References to inaccessible cache line caught and halted
 - Processors cannot use incoherent lines
- Operating System
 - Affect on system
 - Cells crash
 - Lost traffic can damage I/O operations
 - Incorrect speculation can crash multiple cells
 - Limit affect via Hive
 - Hardware firewall
 - Wild write protection

Fault Recovery

- Recovery Algorithm
 - Failure types
 - Node failure
 - Link failure
 - Dedicated Interconnect Lines for Recovery Traffic
 - Implementation
 - Firmware
 - Software



Recovery Initiation

- Failure Detection
 - Memory operation timeout
 - NAK counters
 - MAGIC firmware
 - Truncated packet
- Initiate recovery code
- Determine set of working neighbors
 - Reachable nodes within one hop
- Trigger recover on good nodes
 - Speculative send to neighbors

Information Dissemination

- Learn Global State of System
 - Functioning nodes
 - Rounds used to exchange processor state knowledge
 - Breadth First Search of tree
 - Avoid computation serialization
 - Stable nodes send hints
 - Some nodes defer computation

Interconnect Recovery

- Isolate the Failed Regions
 - Program border routers to route around failed nodes
- Drain Stalled Interconnect Traffic
 - Two phase agreement by nodes
 - Timeout negotiation
 - Vote to proceed/restart
- Reprogram Routing Tables
 - New nonlocal coherence traffic can be sent

Coherence Protocol Recovery

- Reset Cache Coherence Protocol State
 - Dirty remote data flushed (sent to home node)
 - Each node resets cache state
- Determine Incoherent Cache Lines
 - Cached exclusive lines are lost
 - Incoherent lines marked

Operating System Recovery

- Hardware Interrupt
 - Informs node OS that hardware recovery is complete
- Hive
 - Cells adjust kernel data structures
 - Affected user applications terminated
 - Unaffected user applications resume

Additional HW Issues

- Simulation
 - SimOS & FlashLite
- Recovery
 - Dominated by Phase 2 for large systems
 - Flexible Reprogrammable Node Controller
- Support
 - No single point of failure
 - Fail-fast behavior - Failed nodes stop working without generating erroneous output
 - Monotonic failures - Failed nodes halt until repaired

PART IV

Discussion

- If Moore's Law continues what is the advantage/need for parallelism?
- Shared Memory vs. Message Passing
- Snoop Protocols vs. Directory Based
- Fault Tolerance vs. Fault Containment

The slide features a teal background with a large, light gray diamond shape in the center. The text "End of Presentation" is written in a bold, yellow, sans-serif font with a black drop shadow, centered within the diamond. On the left side of the slide, there is a vertical bar with a yellow top half and a dark red bottom half.

End of
Presentation

9/2/99