CS405 Computer System Architecture

Dr CK Raju

Dept of CSE, MITS

November 1, 2018

Presentation Outline

1. Module I: Parallel Computer Models
   - Evolution of Computer Architecture
   - System Attributes to Performance
   - Multiprocessors and Multicomputers
   - Multivector and SIMD
2. Module II: Processors and Memory Hierarchy
3. Module III: Multiprocessors and Multicomputers
4. Module VI: Multithreaded and Data Flow Architecture

Contents of Module I

Contents of Module I

- Architecture of a Simple Processor

Contents of Module I

- Architecture of a Simple Processor
- Parallel Computer Models

Contents of Module I

- Architecture of a Simple Processor
- Parallel Computer Models
- Evolution of Computer Architecture

Contents of Module I

- Architecture of a Simple Processor
- Parallel Computer Models
- Evolution of Computer Architecture
- System Attributes to Performance

Contents of Module I

- Architecture of a Simple Processor
- Parallel Computer Models
- Evolution of Computer Architecture
- System Attributes to Performance
- Multiprocessors and Multicomputers

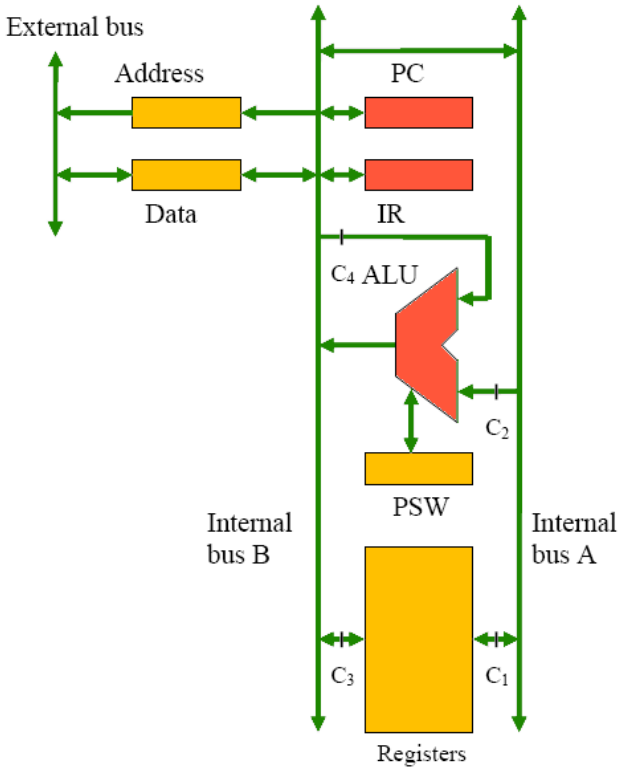
Contents of Module I

- Architecture of a Simple Processor
- Parallel Computer Models
- Evolution of Computer Architecture
- System Attributes to Performance
- Multiprocessors and Multicomputers
- Multivector and SIMD Computers

Contents of Module I

- Architecture of a Simple Processor
- Parallel Computer Models
- Evolution of Computer Architecture
- System Attributes to Performance
- Multiprocessors and Multicomputers
- Multivector and SIMD Computers
- Conditions of Parallelism

Hardware Architecture



ALU (arithmetic, logical and shift),

FPU (Floating Point Unit),

IR (Instruction Register),

PC (Program Counter),

PSW (Program Status Word)

Algorithm of a Simple Processor

repeat forever
begin

- fetch the next instruction from main memory - almost same for all instructions
- execute - i.e. carry out - the instruction fetched - might vary with every instruction

Algorithm of a Simple Processor

repeat forever
begin

- fetch the next instruction from main memory - almost same for all instructions
- execute - i.e. carry out - the instruction fetched - might vary with every instruction

Note: There are two views for the processor
(a) In one view, machine instruction set architecture (ISA) defines microprocessor
(b) In other view, hardware design and circuitry defines the microprocessor end

Evolution of Computer Architecture

Generation	Technology	Systems
First (1945-54)	Vacuum tubes, relays	Eniac, IBM 701
Second (1955-64)	Transistors	Univac, IBM 7090,
Third (1965-74)	Integrated Circuits	IBM 360, PDP-8
Fourth (1975-90)	Microprocessors	VAX 9000, Cray X-MP
Fifth (1991 onwards)	Artificial Intelligence	IBM ES 9000, Xeon PHI

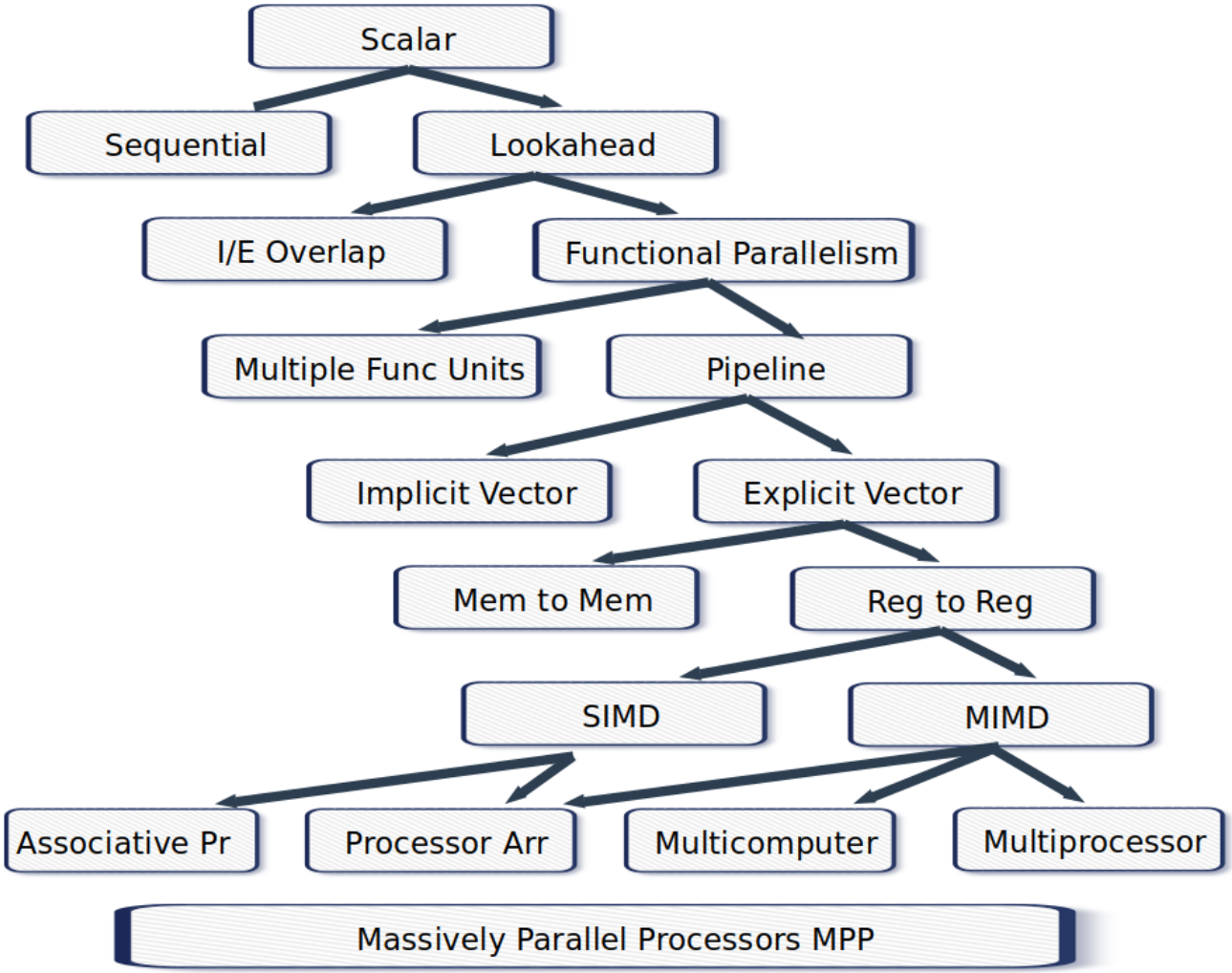
Evolution of Computer Architecture

Generation	Architecture	Systems
First	PC, ALU, FP Arithmetic	Eniac, IBM 701
Second	Multiplexed Memory Access	Univac, IBM 7090,
Third	SSI, MSI, Cache	IBM 360, PDP-8
Fourth	LSI, VLSI, multiprocessors	VAX 9000, Cray X-MP
Fifth	Scalable Multicomputers	IBM ES 9000, Xeon PHI

Evolution of Computer Architecture

Generation	Software	Systems
First	Assembly, Machine	Eniac, IBM 701
Second	HLL with compilers	Univac, IBM 7090,
Third	Multiprogramming OS	IBM 360, PDP-8
Fourth	Multiprocessor OS, Parallel	VAX 9000, Cray X-MP
Fifth	Superscalar, massively parallel	IBM ES 9000, Xeon PHI

Evolution of Computer Architecture



Lookahead, Parallelism and Pipelining

Lookahead, Parallelism and Pipelining

- Lookahead techniques were introduced to prefetch instructions in order to overlap instruction fetch decode execute and to enable functional parallelism

Lookahead, Parallelism and Pipelining

- Lookahead techniques were introduced to prefetch instructions in order to overlap instruction fetch decode execute and to enable functional parallelism
- Functional parallelism - multiple functional units executed simultaneously and practise pipelining

Lookahead, Parallelism and Pipelining

- Lookahead techniques were introduced to prefetch instructions in order to overlap instruction fetch decode execute and to enable functional parallelism
- Functional parallelism - multiple functional units executed simultaneously and practise pipelining
- Pipelining - performing identical operations repeatedly over vector data strings - either implicitly or explicitly

Lookahead, Parallelism and Pipelining

- Lookahead techniques were introduced to prefetch instructions in order to overlap instruction fetch decode execute and to enable functional parallelism
- Functional parallelism - multiple functional units executed simultaneously and practise pipelining
- Pipelining - performing identical operations repeatedly over vector data strings - either implicitly or explicitly

Your smartphone could be an octa-core processor-based ...

Lookahead, Parallelism and Pipelining

- Lookahead techniques were introduced to prefetch instructions in order to overlap instruction fetch decode execute and to enable functional parallelism
- Functional parallelism - multiple functional units executed simultaneously and practise pipelining
- Pipelining - performing identical operations repeatedly over vector data strings - either implicitly or explicitly

Your smartphone could be an octa-core processor-based ... What should be its parallel architecture ...

Lookahead, Parallelism and Pipelining

- Lookahead techniques were introduced to prefetch instructions in order to overlap instruction fetch decode execute and to enable functional parallelism
- Functional parallelism - multiple functional units executed simultaneously and practise pipelining
- Pipelining - performing identical operations repeatedly over vector data strings - either implicitly or explicitly

Your smartphone could be an octa-core processor-based ... What should be its parallel architecture ... think, refer and give a feedback in the next session

Flynn's Classification

Flynn's Classification

- Michael Flynn - introduced classification based on instruction streams and data streams

Flynn's Classification

- Michael Flynn - introduced classification based on instruction streams and data streams
- Conventional sequentional machines were SISD - single instruction stream over single data stream

Flynn's Classification

- Michael Flynn - introduced classification based on instruction streams and data streams
- Conventional sequentional machines were SISD - single instruction stream over single data stream
- Vector computers equipped with scalar or vector hardware appear as SIMD

Flynn's Classification

- Michael Flynn - introduced classification based on instruction streams and data streams
- Conventional sequentional machines were SISD - single instruction stream over single data stream
- Vector computers equipped with scalar or vector hardware appear as SIMD
- Parallel computers are reserved for MIMD machines

Flynn's Classification

- Michael Flynn - introduced classification based on instruction streams and data streams
- Conventional sequentional machines were SISD - single instruction stream over single data stream
- Vector computers equipped with scalar or vector hardware appear as SIMD
- Parallel computers are reserved for MIMD machines
- A MISD architecture called systolic arrays for pipelined execution of specific algorithms is also present

Factors influencing machine performance

Machine performance is usually dependent on a variety of factors like

Factors influencing machine performance

Machine performance is usually dependent on a variety of factors like

- hardware technology

Factors influencing machine performance

Machine performance is usually dependent on a variety of factors like

- hardware technology
- innovative architectural features

Factors influencing machine performance

Machine performance is usually dependent on a variety of factors like

- hardware technology
- innovative architectural features
- efficient resource management

Factors influencing machine performance

Machine performance is usually dependent on a variety of factors like

- hardware technology
- innovative architectural features
- efficient resource management
- algorithm design

Factors influencing machine performance

Machine performance is usually dependent on a variety of factors like

- hardware technology
- innovative architectural features
- efficient resource management
- algorithm design
- data structures, language efficiency, programmer skills, compiler technology etc

Note: Generally its *turnaround time* that is considered that includes disk and memory access, input and output activities, compilation time, OS overhead and CPU time. Since CPU time includes system CPU time and user CPU time, we focus only on *user CPU time* for the purpose of benchmarking performance.

Clock Rate and CPI - cycles per instruction

Clock Rate and CPI - cycles per instruction

- CPU driven by clock with constant cycle time T

Clock Rate and CPI - cycles per instruction

- CPU driven by clock with constant cycle time T
- Clock Rate is f = 1/T

Clock Rate and CPI - cycles per instruction

- CPU driven by clock with constant cycle time T
- Clock Rate is $f = 1/T$
- Size of program is determined by Instruction Count I_c - no of instructions to be executed in the program

Clock Rate and CPI - cycles per instruction

- CPU driven by clock with constant cycle time T
- Clock Rate is $f = 1/T$
- Size of program is determined by Instruction Count I_c - no of instructions to be executed in the program
- Different instructions use different clock cycles

Clock Rate and CPI - cycles per instruction

- CPU driven by clock with constant cycle time T
- Clock Rate is $f = 1/T$
- Size of program is determined by Instruction Count I_c - no of instructions to be executed in the program
- Different instructions use different clock cycles
- Cycles per instruction CPI is therefore used as a parameter to measure time

Clock Rate and CPI - cycles per instruction

- CPU driven by clock with constant cycle time T
- Clock Rate is $f = 1/T$
- Size of program is determined by Instruction Count I_c - no of instructions to be executed in the program
- Different instructions use different clock cycles
- Cycles per instruction CPI is therefore used as a parameter to measure time
- Average CPI is found out for the program (taking into consideration all its instructions and the clock cycles needed for executing these)

Performance Factors

Performance Factors

- Let Instruction Count of a program be I_c

Performance Factors

- Let Instruction Count of a program be I_c
- CPU time = I_c x CPI x T (const cycle time = 1/f)

Performance Factors

- Let Instruction Count of a program be I_c
- CPU time = I_c x CPI x T (const cycle time = 1/f)
- Usually an instruction might contain - fetch (instruction), decode (instruction), fetch operands, execute, and store (results)

Performance Factors

- Let Instruction Count of a program be I_c
- CPU time = I_c x CPI x T (const cycle time = 1/f)
- Usually an instruction might contain - fetch (instruction), decode (instruction), fetch operands, execute, and store (results)
- Here instruction decode and execute are carried out by CPU. Other stages need access to memory.

Performance Factors

- Let Instruction Count of a program be I_c
- CPU time = I_c x CPI x T (const cycle time = 1/f)
- Usually an instruction might contain - fetch (instruction), decode (instruction), fetch operands, execute, and store (results)
- Here instruction decode and execute are carried out by CPU. Other stages need access to memory.
- A memory cycle is time needed to complete one memory reference - usually memory cycle is k times processor cycle (ratio)

Performance Factors

- Let Instruction Count of a program be I_c
- CPU time = I_c x CPI x T (const cycle time = 1/f)
- Usually an instruction might contain - fetch (instruction), decode (instruction), fetch operands, execute, and store (results)
- Here instruction decode and execute are carried out by CPU. Other stages need access to memory.
- A memory cycle is time needed to complete one memory reference - usually memory cycle is k times processor cycle (ratio)
- Therefore total time = I_c x (p + m x k) x T where p - processor cycles needed for decode and execute, m is no of memory references needed and k is ratio between memory cycle and processor cycle, I_c is the instruction count and T is the processor cycle time

System Attributes

System Attributes

- The five performance factors - I_c , p, m, k, and T are influenced by four system attributes

System Attributes

- The five performance factors - I_c , p, m, k, and T are influenced by four system attributes
- ISA instruction set architecture, Compiler technology, CPU implementation and control, Cache and memory hierarchy

System Attributes

- The five performance factors - I_c , p, m, k, and T are influenced by four system attributes
- ISA instruction set architecture, Compiler technology, CPU implementation and control, Cache and memory hierarchy
- ISA affects program length (I_c) and processor cycles needed (p)

System Attributes

- The five performance factors - I_c , p, m, k, and T are influenced by four system attributes
- ISA instruction set architecture, Compiler technology, CPU implementation and control, Cache and memory hierarchy
- ISA affects program length (I_c) and processor cycles needed (p)
- Compiler technology affects program length (I_c), processor cycles needed (p) and memory reference count (m)

System Attributes

- The five performance factors - I_c , p, m, k, and T are influenced by four system attributes
- ISA instruction set architecture, Compiler technology, CPU implementation and control, Cache and memory hierarchy
- ISA affects program length (I_c) and processor cycles needed (p)
- Compiler technology affects program length (I_c), processor cycles needed (p) and memory reference count (m)
- CPU implementation and control determine total processor time p x T needed

System Attributes

- The five performance factors - I_c , p, m, k, and T are influenced by four system attributes
- ISA instruction set architecture, Compiler technology, CPU implementation and control, Cache and memory hierarchy
- ISA affects program length (I_c) and processor cycles needed (p)
- Compiler technology affects program length (I_c), processor cycles needed (p) and memory reference count (m)
- CPU implementation and control determine total processor time p x T needed
- Memory technology and hierarchy design influence memory access latency (k x T)

Performance Factors versus System Attributes

THE STATE OF COMPUTING
System Attributes to Performance

	Performance Factors				
System Attributes	Instruction Count (I_c)	Average Cycles per Instruction (CPI)			Processor Cycle Time (τ)
		Processor Cycles per Instruction (CPI and p)	Memory References per Instruction (m)	Memory Access Latency (k)	
Instruction-set Architecture	☑	☑			
Compiler Technology	☑	☑	☑		
Processor Implementation and Control		☑			☑
Cache and Memory Hierarchy				☑	☑

Multiprocessors and Multicomputers

There are two main categories of parallel computer systems

- systems with common memory that are shared
- systems with distributed memory that are not shared

Shared memory Multiprocessors

Shared memory multiprocessor models are of three types:

- Uniform-memory-access (UMA)
- Nonuniform-memory-access (NUMA)
- Cache-only memory architecture (COMA)

These systems differ in how the memory and peripheral resources are shared or distributed.

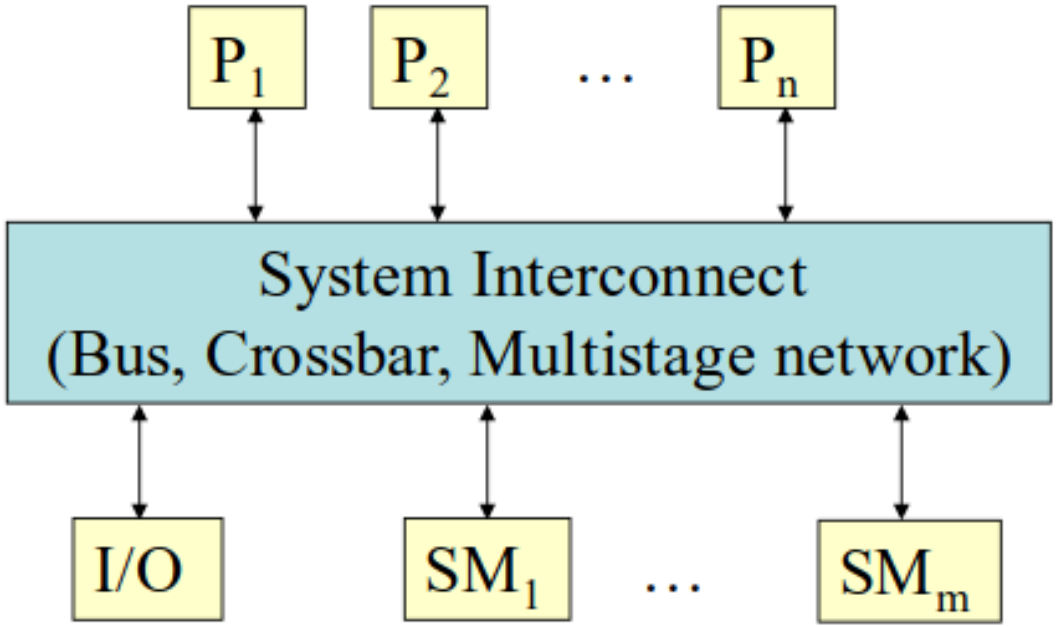
UMA Model I

- Physical memory uniformly shared by all processors, with equal access time to all words
- Processors may have local cache memories
- Peripherals also shared in some fashion
- Tightly coupled systems use a common bus, crossbar, or multistage network to connect processors, peripherals and memories
- Many manufacturers have multiprocessor (MP) extensions of uniprocessor (UP) product lines

UMA Model II

- Synchronization and communication among processors achieved through shared variables in common memory
- Symmetric MP systems - all processors have access to all peripherals, and any processor can run the OS and I/O device drivers
- Asymmetric MP systems - not all peripherals accessible by all processors; kernel runs only on selected processors (master node); others are called attached processors (AP)

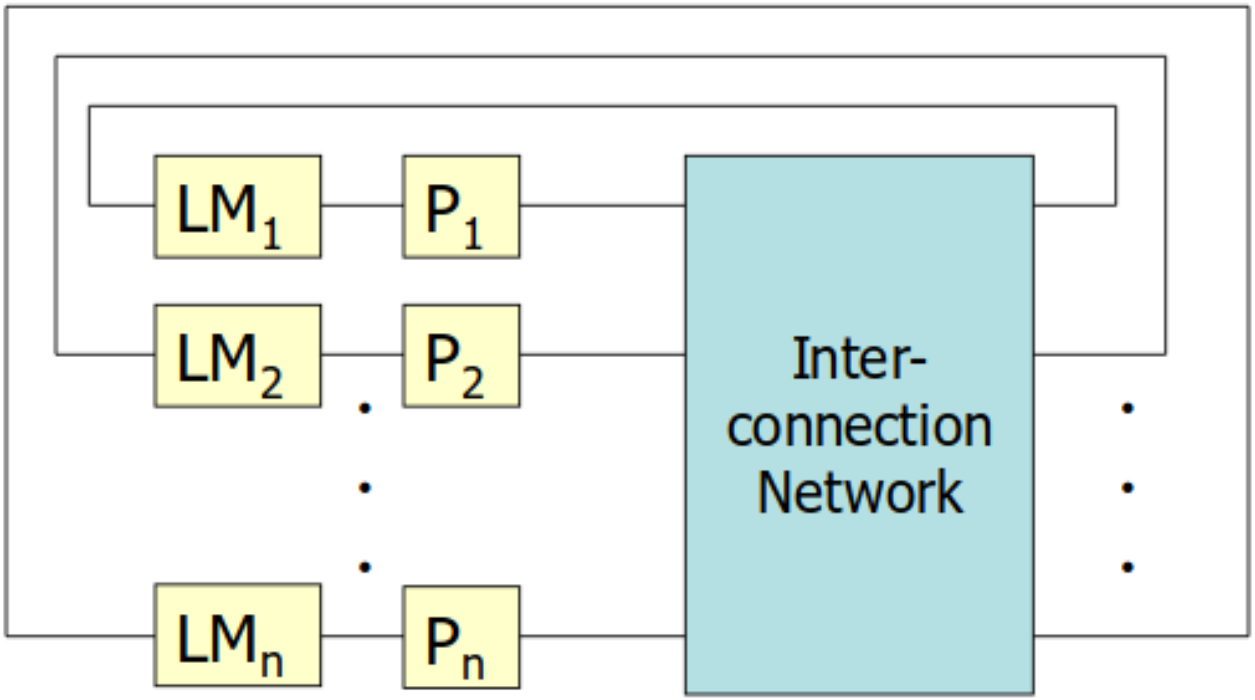
The UMA Multiprocessor Model



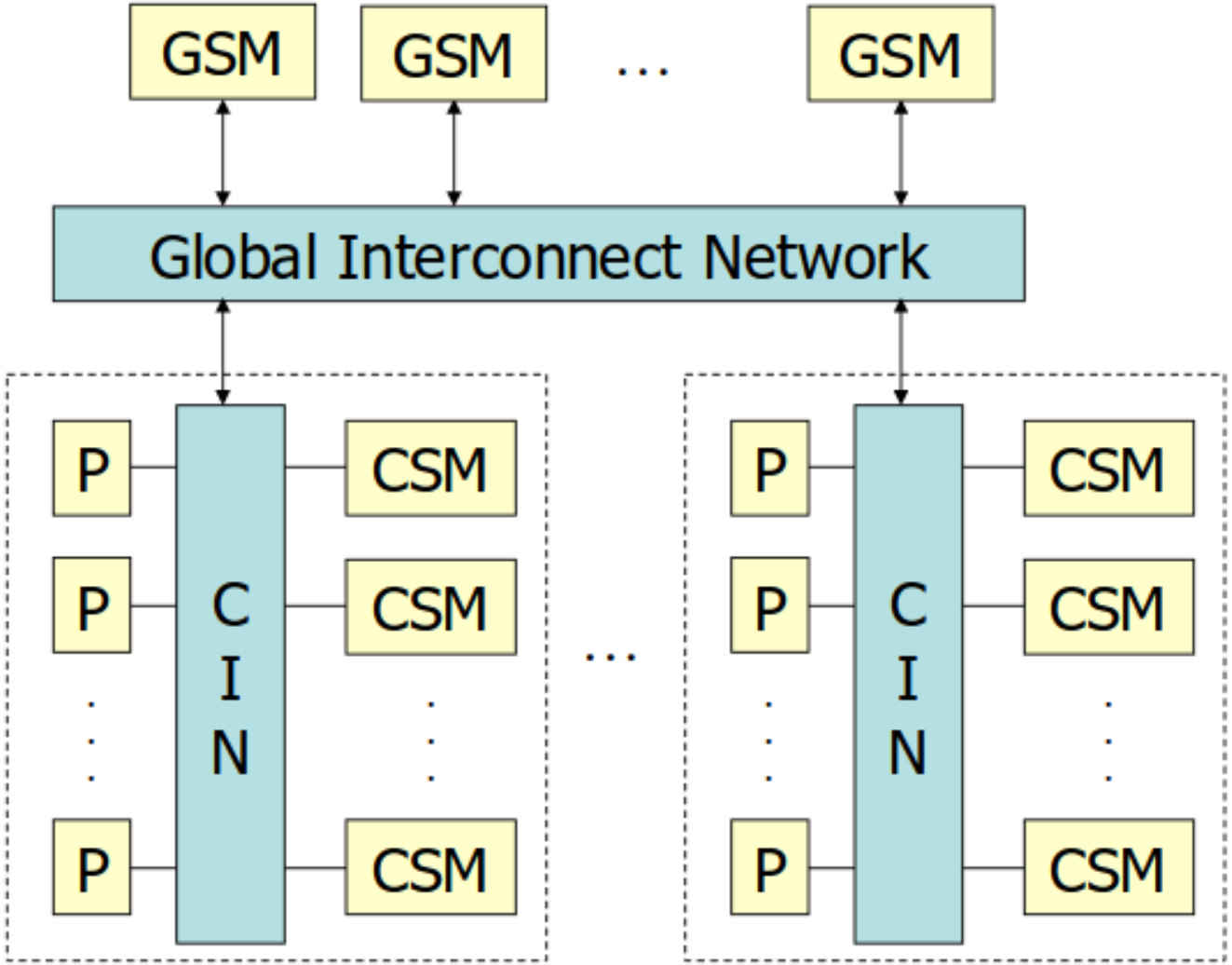
NUMA Model I

- Shared memories, but access time depends on the location of the data item
- Shared memory is distributed among the processors as local memory, but each of these is still accessible by all processors (with varying access times)
- Memory access is fastest from the locally connected processor, with the interconnection network adding delays for other processor accesses.
- Additionally there may be global memory in a multiprocessor system, with two separate interconnection networks, one for clusters of processors and their cluster memories, and the other for the global shared memories

Shared Local Memories



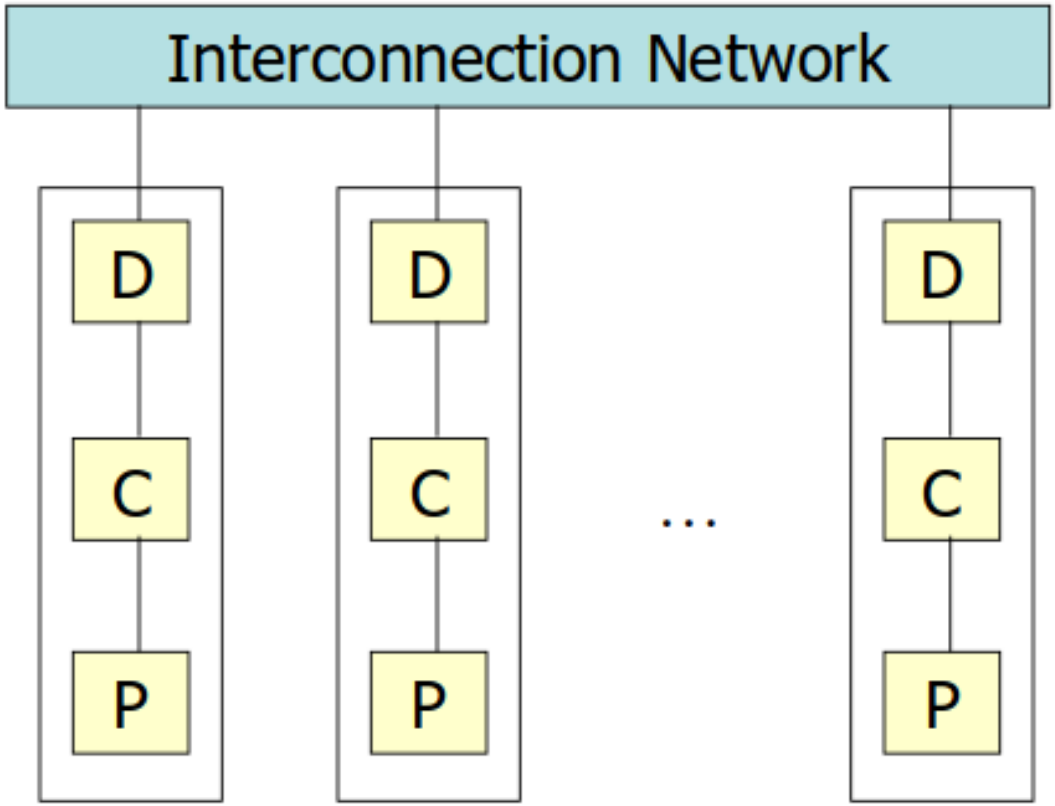
Hierarchical Cluster Model



COMA Model

- In the COMA model, processors only have cache memories; the caches, taken together, form a global address space
- Each cache has an associated directory that aids remote machines in the lookups; hierarchical directories may exist in machines based on this model
- Initial data placement is not critical, as cache blocks will eventually migrate to where they are needed

Cache-Only Memory Architecture



Other Models

There are also other models used in multiprocessor systems, based on a combination of models just described.

- Cache-coherent non-uniform memory access (each processor has a cache directory, and the system has a distributed shared memory)
- cache-coherent cache-only model (processors have caches, no shared memory, caches must be kept coherent)

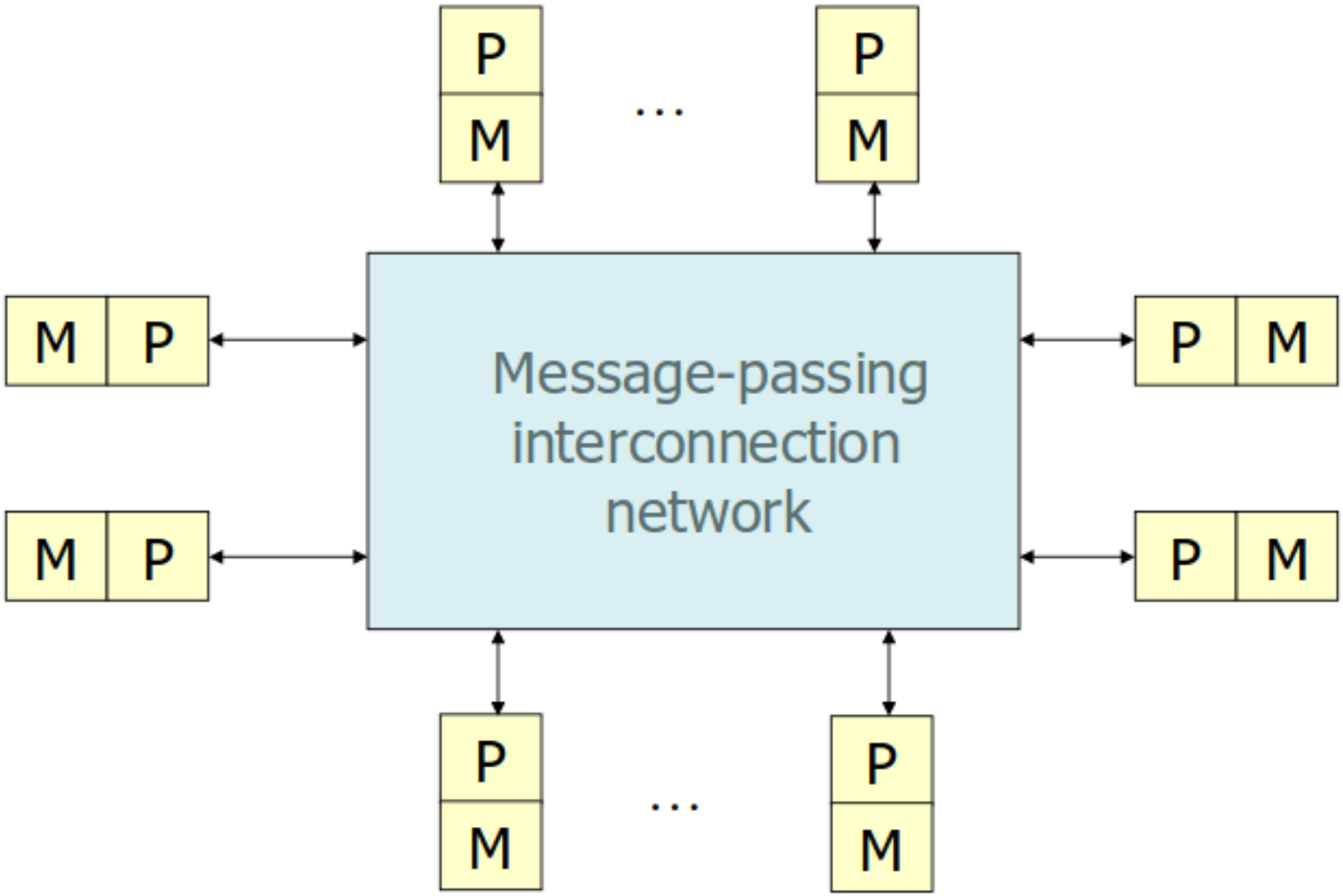
Some Early Commercial Multiprocessor Systems

Company and Model	Hardware and Architecture	Software and Applications	Remarks
Sequent Symmetry S-81	Bus-connected with 30 i386 processors, IPC via SLIC bus; Weitek floating-point accelerator.	DYNIX/OS, KAP/Sequent preprocessor, transaction multiprocessing.	Latter models designed with faster processors of the family.
IBM ES/9000 Model 900/VF	6 ES/9000 processors with vector facilities, crossbar connected to I/O channels and shared memory.	OS support: MVS, VM KMS, AIX/370, parallel Fortran, VSF V2.5 compiler.	Fiber optic channels, integrated cryptographic architecture.
BBN TC-2000	512 M88100 processors with local memory connected by a Butterfly switch, a NUMA machine.	Ported Mach/OS with multiclustering, parallel Fortran, time-critical applications.	Latter models designed with faster processors of the family.

Multicomputer Models

- Multicomputers consist of multiple computers, or nodes interconnected by a message passing network
- Each node is autonomous, with its own processor and local memory and sometimes local peripherals
- The message-passing network provides point-to-point static connections among the nodes
- Local memories are not shared, so traditional multicomputers are sometimes called no-remote-memory-access (or NORMA) machines
- Inter-node communication is achieved by passing messages through the static connection network

Generic Message Passing Multicomputer



Multicomputer Generations

- Each multicomputer uses routers and channels in its interconnection network, and heterogeneous systems may involve mixed mode types and uniform data representation and communication protocols
- First generation: hypercube architecture, software-controlled message switching, processor boards
- Second generation: mesh-connected architecture, hardware message switching, software for medium-grain distributed computing
- Third generation: Fine-grained distributed computing, with each VLSI chip containing the processor and communication resources.

Some Early Commercial Multicomputer Systems

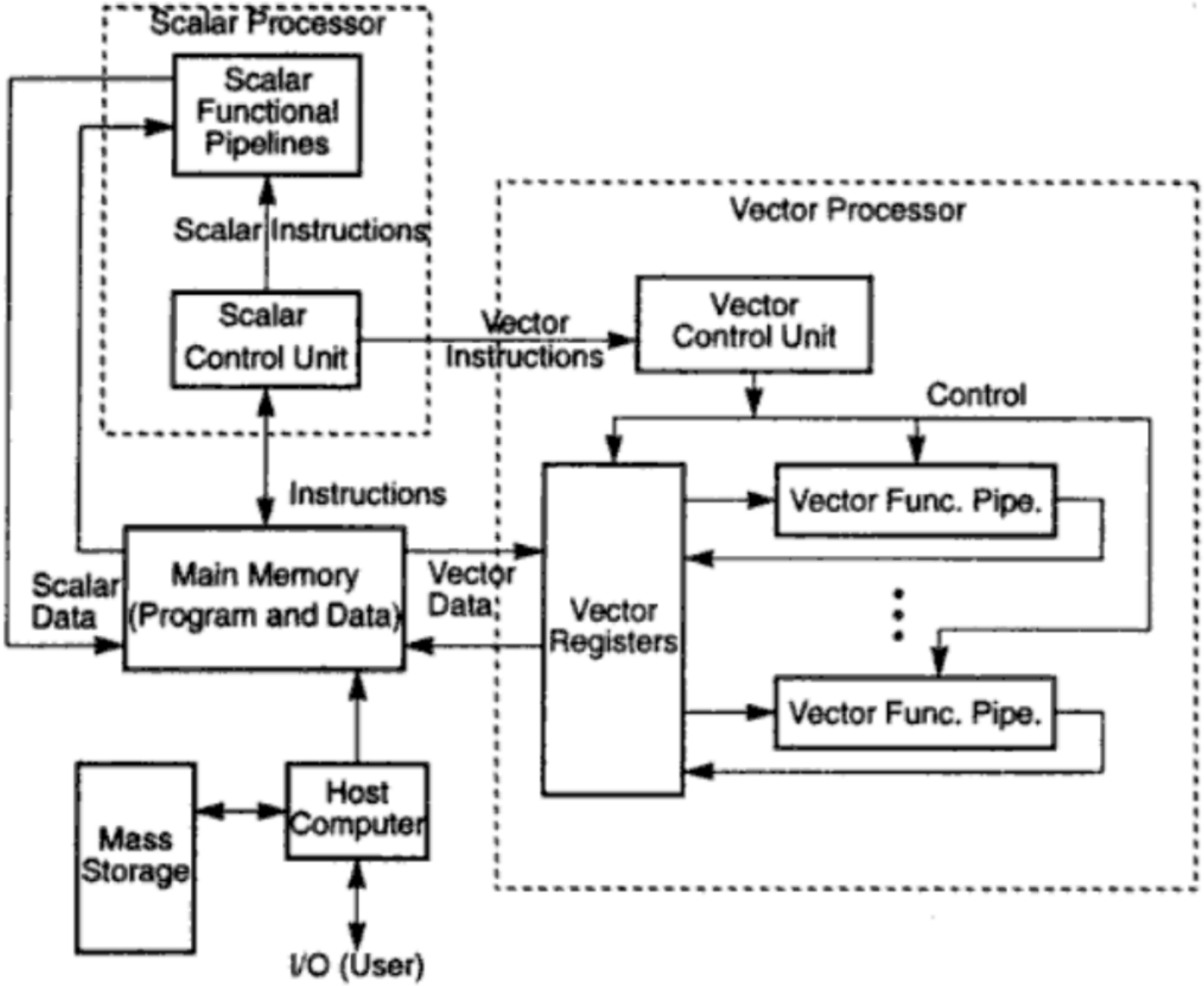
System Features	Intel Paragon XP/S	nCUBE/2 6480	Parsys SuperNode1000
Node Types and Memory	50 MHz i860 XP computing nodes with 16–128 Mbytes per node, special I/O service nodes.	Each node contains a CISC 64-bit CPU, with FPU, 14 DMA ports, with 1–64 Mbytes /node.	EC-funded Esprit supernode built with multiple T-800 Transputers per node.
Network and I/O	2-D mesh with SCSI, HIPPI, VME, Ethernet, and custom I/O.	13-dimensional hypercube of 8192 nodes, 512-Gbyte memory, 64 I/O boards.	Reconfigurable interconnect, expandable to have 1024 processors.
OS and Software Task Parallelism Support	OSF conformance with 4.3 BSD, visualization and programming support.	Vertex/OS or UNIX supporting message passing using wormhole routing.	IDRIS/OS UNIX-compatible.
Application Drivers	General sparse matrix methods, parallel data manipulation, strategic computing.	Scientific number crunching with scalar nodes, database processing.	Scientific and academic applications.
Performance Remarks	5–300 Gflops peak 64-bit results, 2.8–160 GIPS peak integer performance.	27 Gflops peak, 36 Gbytes/s I/O	200 MIPS to 13 GIPS peak.

Multivector and SIMD Computers

Vector Processors

- Vector Processor Variants
  - Vector Supecomputers
  - Attached Processors
- Vector Processor Models / Architecture
  - Register-to-register architecture (vectors are retrieved from registers)
  - Memory-to-memory architecture (vectors are retrieved from memory)
- Representative Systems
  - Cray-I
  - Cray Y-MP (2, 4 or 8 processors with 16Gflops peak performance)
  - Convex C1, C2, C3 series (C3800 family with 8 processors, 4 GB main memory, 2 Gflops peak performance)
  - DEC VAX 9000 (pipeline chaining support)

Architecture of Vector Supercomputer - register-to-register



Representative Vector Supercomputers

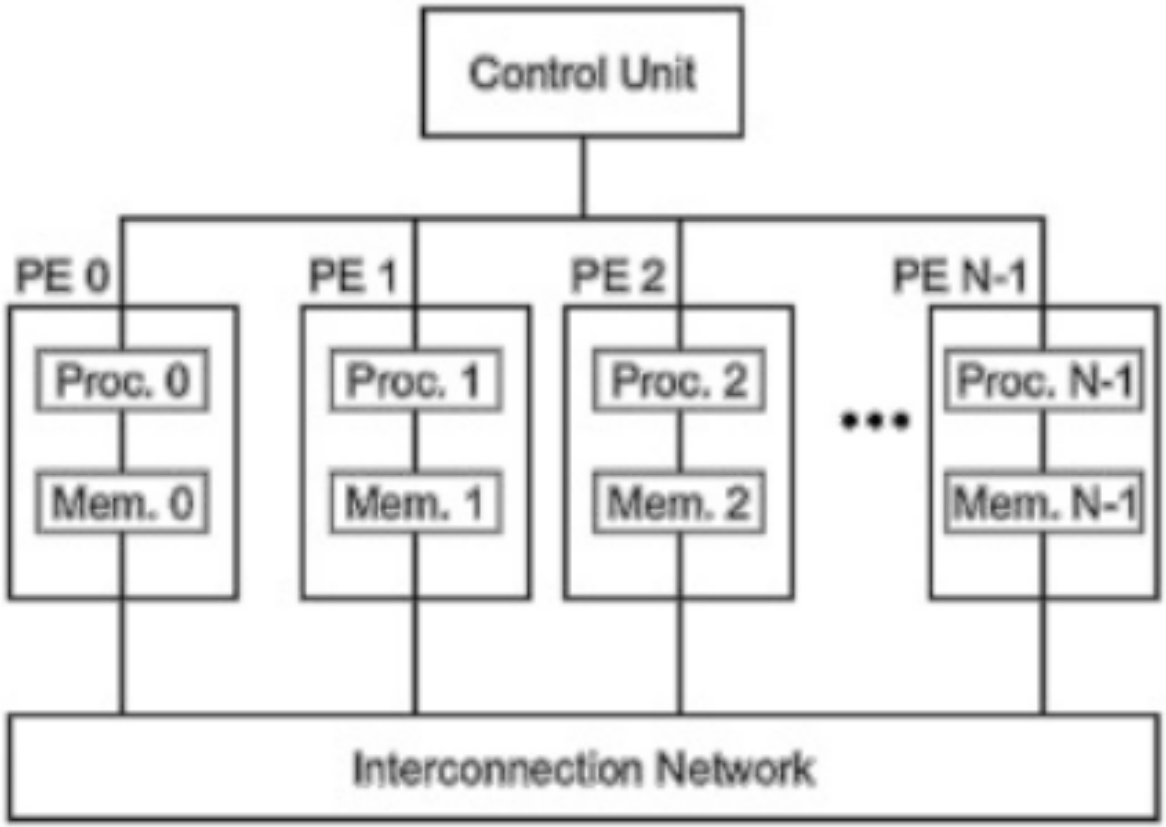
System Model	Vector Hardware Architecture and Capabilities	Compiler and Software Support
Convex C3800 family	GaAs-based multiprocessor with 8 processors and 500-Mbyte/s access port. 4 Gbytes main memory. 2 Gflops peak performance with concurrent scalar/vector operations.	Advanced C, Fortran, and Ada vectorizing and parallelizing compilers. Also support inter-procedural optimization, POSIX 1003.1/OS plus I/O interfaces and visualization system
Digital VAX 9000 System	Integrated vector processing in the VAX environment, 125–500 Mflops peak performance. 63 vector instructions. 16 x 64 x 64 vector registers. Pipeline chaining possible.	MS or ULTRIX/OS, VAX Fortran and VAX Vector Instruction Emulator (VVIEF) for vectorized program debugging.
Cray Research Y-MP and C-90	Y-MP runs with 2, 4, or 8 processors, 2.67 Gflop peak with Y-MP8256. C-90 has 2 vector pipes/CPU built with 10K gate ECL with 16 Gflops peak performance.	CF77 compiler for automatic vectorization, scalar optimization, and parallel processing. UNICOS improved from UNIX/V and Berkeley BSD/OS.

Multivector and SIMD Computers

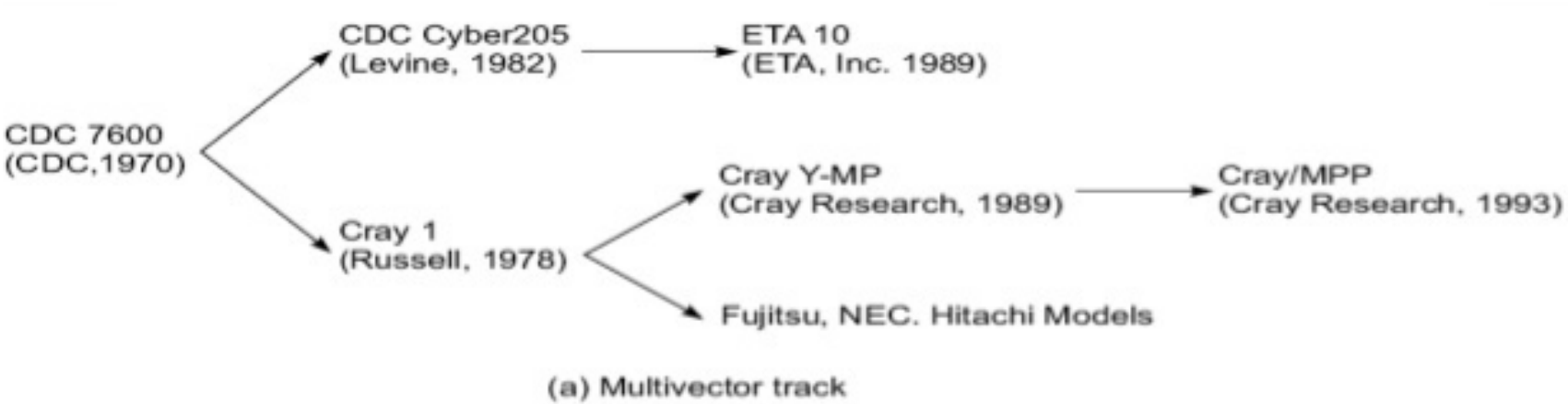
SIMD Supercomputers

- SIMD Machine Model
  - S = <N, C, I, M, R>
  - N: No of PEs in the machine
  - C: Set of instructions (scalar/program flow) directly executed by control unit
  - I: Set of instructions broadcast by CU to all PEs for parallel execution
  - M: Set of masking schemes (to know which processors are active)
  - R: Set of data routing functions
- Representative Systems
  - MasPar MP-1 (1024 to 16384 PEs)
  - CM-2 (65536 PEs)
  - DAP600 Family (upto 4096 PEs)
  - Illiac-IV (64 PEs)

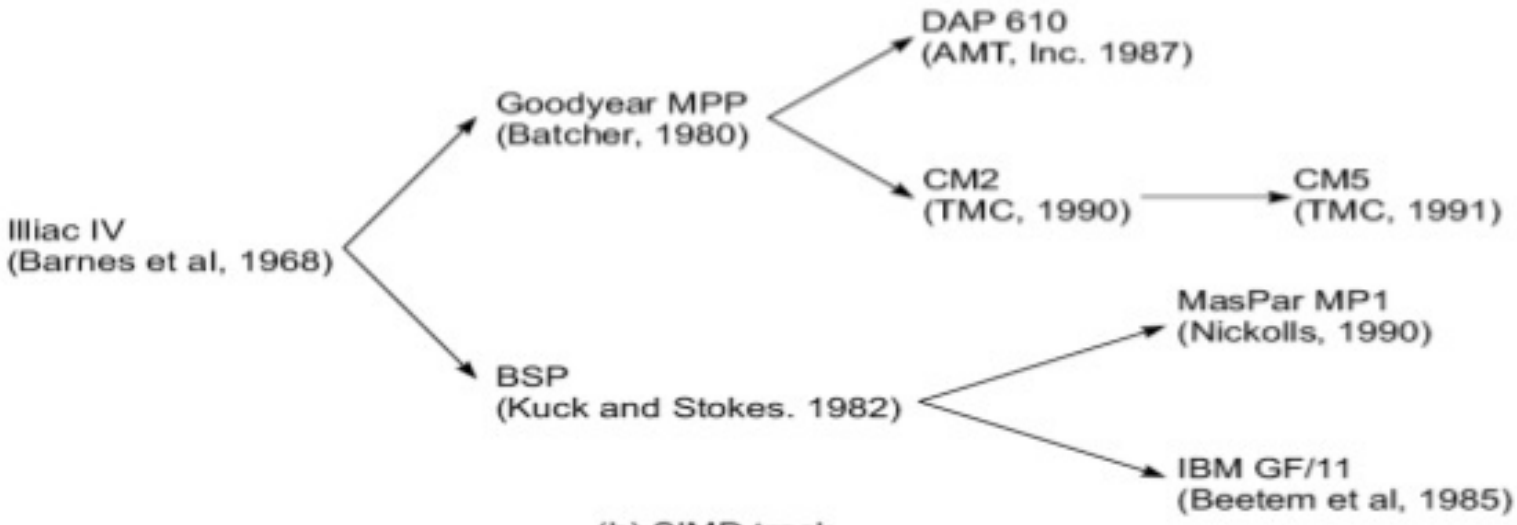
Operational Model of SIMD Supercomputers



History of Multivector and SIMD tracks



(a) Multivector track



(b) SIMD track

Contents of Module I

- Architecture of a Simple Processor
- Parallel Computer Models
- Evolution of Computer Architecture
- System Attributes to Performance
- Multiprocessors and Multicomputers
- Multivector and SIMD Computers
- **Conditions of Parallelism**

Conditions of Parallelism

- Data, Control and Resource Dependences
- Hardware and Software Parallelism
- Role of Compilers

Conditions of Parallelism

- **Data, Control and Resource Dependences**
- Hardware and Software Parallelism
- Role of Compilers

Data Dependences

- Flow Dependence
- Antidependence
- Output Dependence
- I/O Dependence
- Unknown Dependence

Data Dependences

- Flow Dependence

  S1: Load R1, A
  S2: Add R2, R1
  S2 is flow dependent on S1

- Antidependence
- Output Dependence
- I/O Dependence
- Unknown Dependence

Data Dependences

- Flow Dependence
- Antidependence

  S2: Add R2, R1

  S3: Mov R1, R3

  S3 is anti-dependent on S2
- Output Dependence
- I/O Dependence
- Unknown Dependence

Data Dependences

- Flow Dependence
- Antidependence
- Output Dependence

Statements that work on same output

- I/O Dependence
- Unknown Dependence

Data Dependences

- Flow Dependence
- Antidependence
- Output Dependence
- I/O Dependence

  If same I/O device (eg:- file) is involved in two statements, the statements are said to be I/O dependent.

- Unknown Dependence

Data Dependences

- Flow Dependence
- Antidependence
- Output Dependence
- I/O Dependence
- Unknown Dependence
  - The subscript of a variable is itself subscribed
  - Subscript does not contain the loop index variable
  - Variable appears more than once with subscripts having different coefficients of loop variable
  - Subscript is nonlinear in the loop index variable

Control Dependence

- Loop constructs
  - Control Independent

```
DO 20 K = 1,N
      A(K) = C(K)
      IF (A(K).LT.0) A(K) = 1
20 CONTINUE
```

  - Control Dependent

```
DO 30 K = 1, N
      IF (A(K-1).EQ.0) A(K) = 0
30 CONTINUE
```

Resource Dependence

Concerned with conflicts in using shared resources

Shared resources - Integer units, Floating-point units, Registers, Memory areas, ALU etc

Eg:- In *ALU dependence*, conflicting resource is ALU

Resource Dependence: Bernstein's Condition 1966

If P_1 and P_2 are to be parallelizable processes, whose inputs are I_1 and I_2 respectively and outputs are O_1 and O_2,

The following Berstein conditions should be met

- $I_1 \cap O_2 = \emptyset$
- $I_2 \cap O_1 = \emptyset$
- $O_1 \cap O_2 = \emptyset$

Resource Dependence: Hardware Parallelism and Software Parallelism

Hardware Parallelism: defined largely by machine architecture and hardware multiplicity

Involves cost and performance tradeoffs

Software Parallelism: function of algorithm, programming style and program design

Resource Dependence : Compiler techniques

Compiler techniques used to exploit hardware features to improve performance.

Compiling for multiprocessors much more challenging than for uniprocessors.

Granularity and Communication latency significant in code optimization and scheduling process.

Presentation Outline

1. Module I: Parallel Computer Models
   - Evolution of Computer Architecture
   - System Attributes to Performance
   - Multiprocessors and Multicomputers
   - Multivector and SIMD
2. Module II: Processors and Memory Hierarchy
3. Module III: Multiprocessors and Multicomputers
4. Module VI: Multithreaded and Data Flow Architecture

Contents of Module II

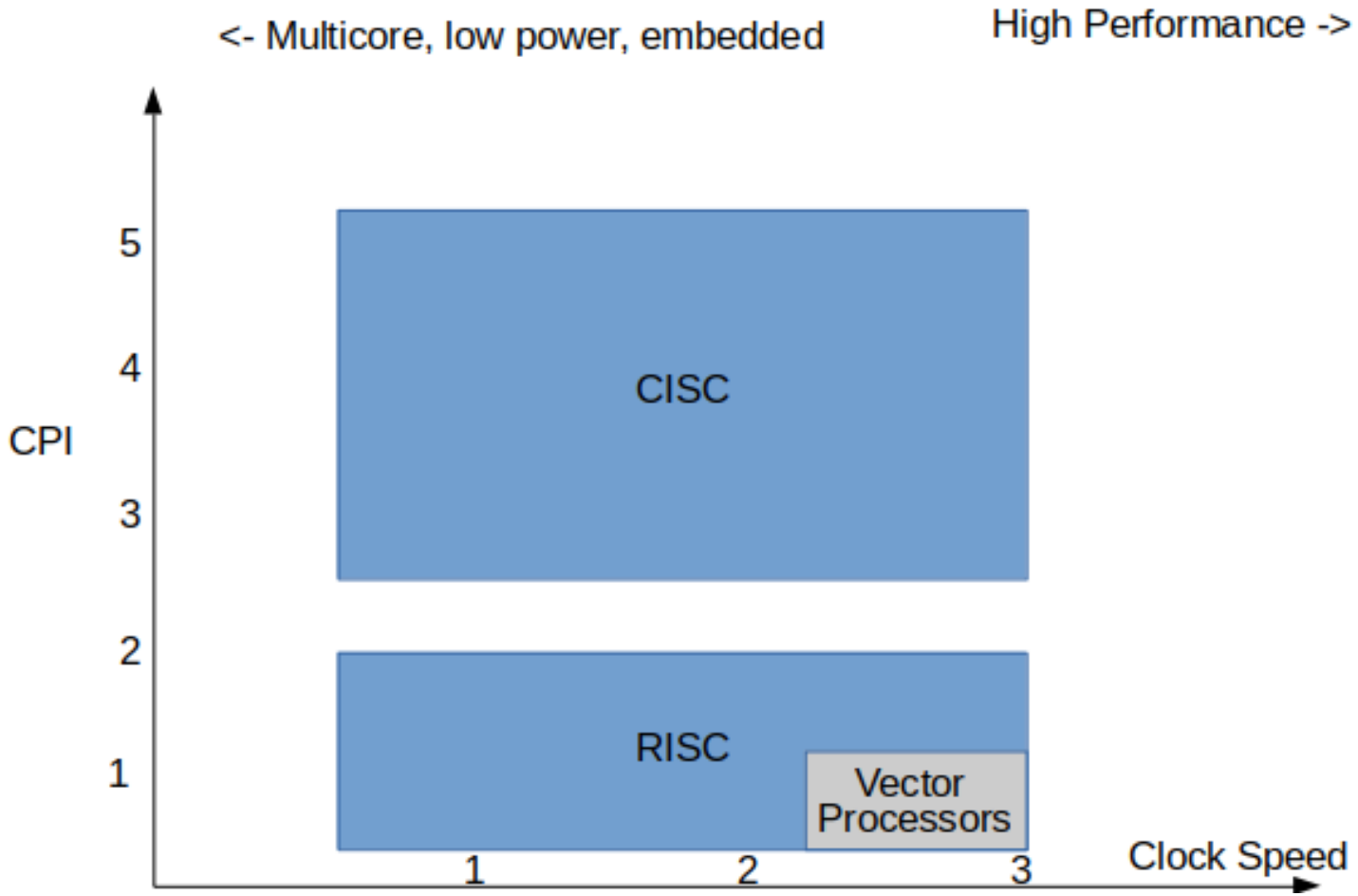
- Design Space of Processors
- Instruction Set Architecture
- CISC Scalar Processors
- RISC Scalar Processors
- Superscalar and Vector Processors
- Super Scalar Processors
- Vector and Symbolic Processors
- Memory Hierarchy Technology
- Amdahl's Law

Advanced Processor Technology

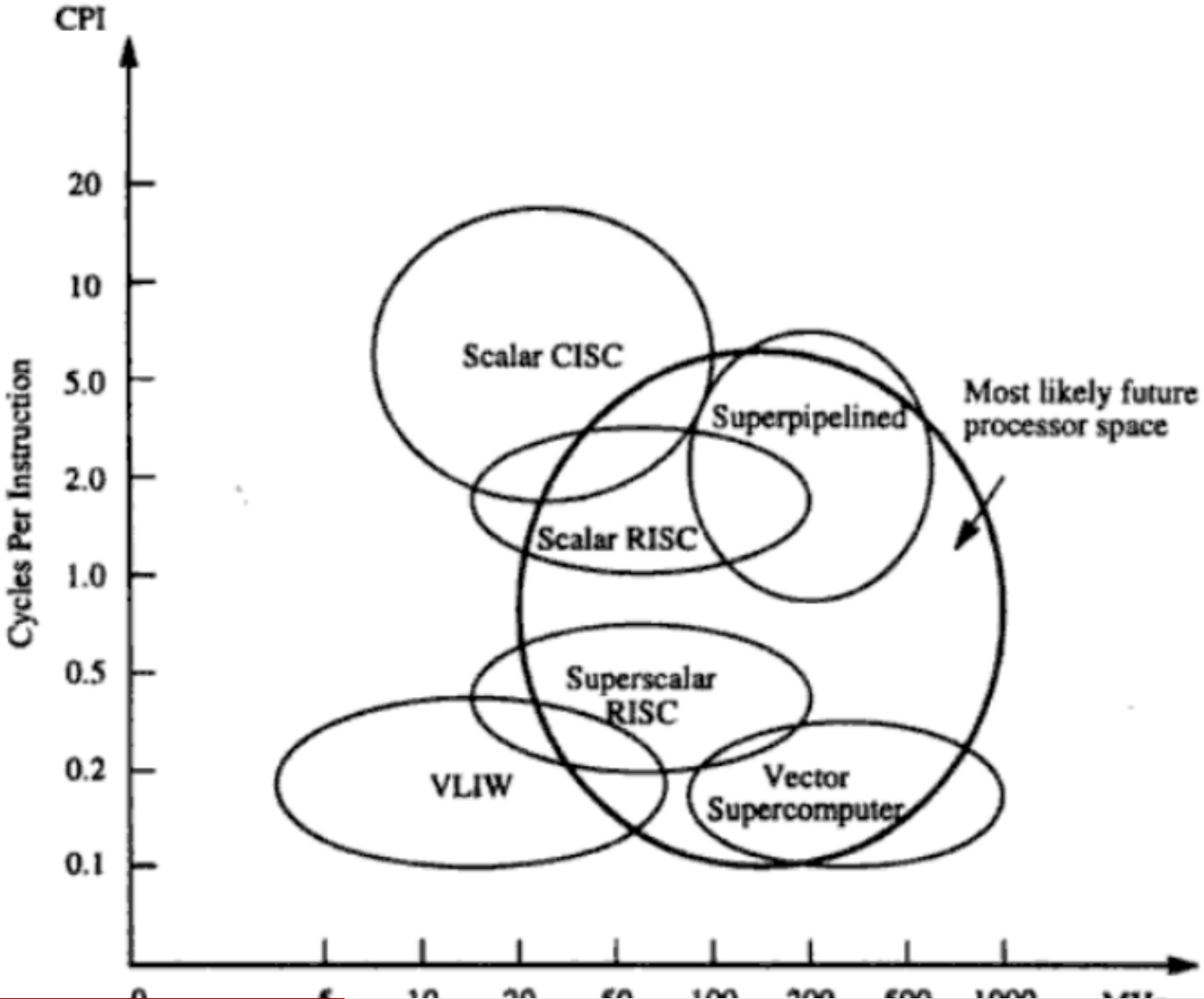
Major Processor Families to be explored here

- CISC
- RISC
- Superscalar
- VLIW
- Superpipelined
- Vector
- Symbolic (for AI)

Design Space of Processors



Design Space of Processors



Design Space of Processors

Complex Instruction Set Computing Architecture - Intel Pentium, M68040, VAX 8600, IBM 390 etc

CPI will be anywhere from 2 to 20, Clock speed may reach upto 3 GHz

Reduced Instruction Set Computing Architecture - Sparc, Power Series, MIPS, Alpha, ARM etc

CPI will be anywhere between 1 and 2

Superscalar architecture is a subclass of RISC processors - allows multiple instructions to be issued simultaneously during each cycle. Thus effective CPI gets reduced.

VLIW Very long instruction word - can use even more functional units than superscalar processor - still lower CPI

Vector processors - use multiple functional units for concurrent scalar and vector operations

CISC-RISC Characteristics

Architectural Characteristic	Complex Instruction Set Architecture	Reduced Instruction Set Architecture
Instructional Set Size and instruction formats	Large set of instructions with variable formats	Small set of instructions (32 bit format)
Addressing Modes	12-24	3-5
General Purpose Registers and cache design	8-24 GPRs originally with unified cache for instructions and data.	32-192 with mostly split data cache and instruction cache
CPI	CPI between 2-15	Less than 1.5
CPU Control	Earlier microcoded using ROM, modern one - hardwired	Mostly hardwired without control memory

Fig: RISC Architecture with hardwired control, split instruction cache and data cache

CISC Scalar Processors

Feature	Intel i486	Motorola MC68040	NS 32532
Instruction-set size and word length	157 instructions, 32 bits.	113 instructions, 32 bits.	63 instructions, 32 bits.
Addressing modes	12	18	9
Integer unit and GPRs	32-bit ALU with 8 registers.	32-bit ALU with 16 registers.	32-bit ALU with 8 registers.
On-chip cache(s) and MMUs	8-KB unified cache for both code and data.	4-KB code cache 4-KB data cache with separate MMUs.	512-B code cache 1-KB data cache.
Floating-point unit, registers, and function units	On-chip with 8 FP registers adder, multiplier, shifter.	On-chip with 3 pipeline stages, 8 80-bit FP registers.	Off-chip FPU NS 32381, or WTL 3164.
Pipeline stages	5	6	4
Protection level	4	2	2
Memory organization and TLB/ATC entries	Segmented paging with 4 KB/page and 32 entries in TLB.	Paging with 4 or 8 KB/page, 64 entries in each ATC.	Paging with 4 KB/page, 64 entries.
Technology, clock rate, packaging, and year introduced	CHMOS IV, 25 MHz, 33 MHz, 1.2M transistors, 168 pins, 1989.	0.8−μm HCMOS, 1.2M transistors, 20 MHz, 40 MHz, 179 pins, 1990.	1.25−μm CMOS 370K transistors, 30 MHz, 175 pins, 1987.
Claimed performance	24 MIPS at 25 MHz,	20 MIPS at 25 MHz, 30 MIPS at 60 MHz.	15 MIPS at 30 MHz.
Successors	i586,	MC68050,	unknown.

CISC

Features of a typical CISC instruction set

- ISA of CISC contains approx 120 to 350 instructions in varying instruction or data formats
- Uses small set of 8 to 24 general-purpose registers (GPRs)
- Has over a dozen memory addressing modes
- Many HLL statements are directly implemented in hardware/firmware (PUSH/POP)

Why RISC preferred over CISC

Features of a typical CISC instruction set

- Analysis of programs revealed that only 25 percent of instructions in CISC are frequently used 95 percent of time
- 75 percent of hardware supported instructions often unused in CISC
- Question : Why waste valuable chip area for rarely used instructions ?
- RISC set contains less than 100 instructions with fixed instruction formats (32/64 bits)
- In RISC, only three to five simple addressing modes are used
- Most instructions are register-based
- Memory access is done by load/store instructions only
- Most instructions execute in one cycle with hardwired control

Instruction Set Architecture - CISC

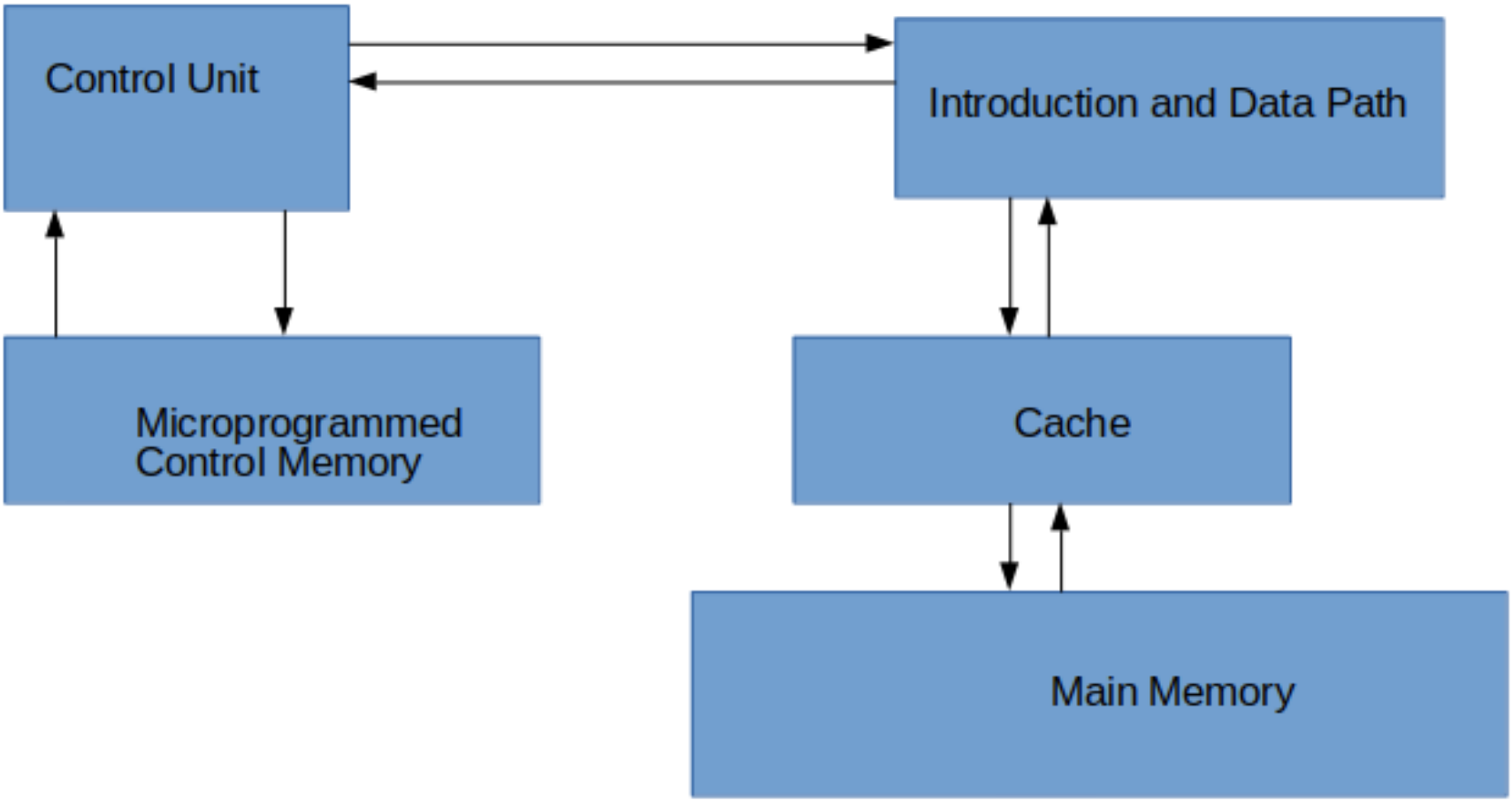


Fig: CISC Architecture aiwth microprogrammed control and unified cache

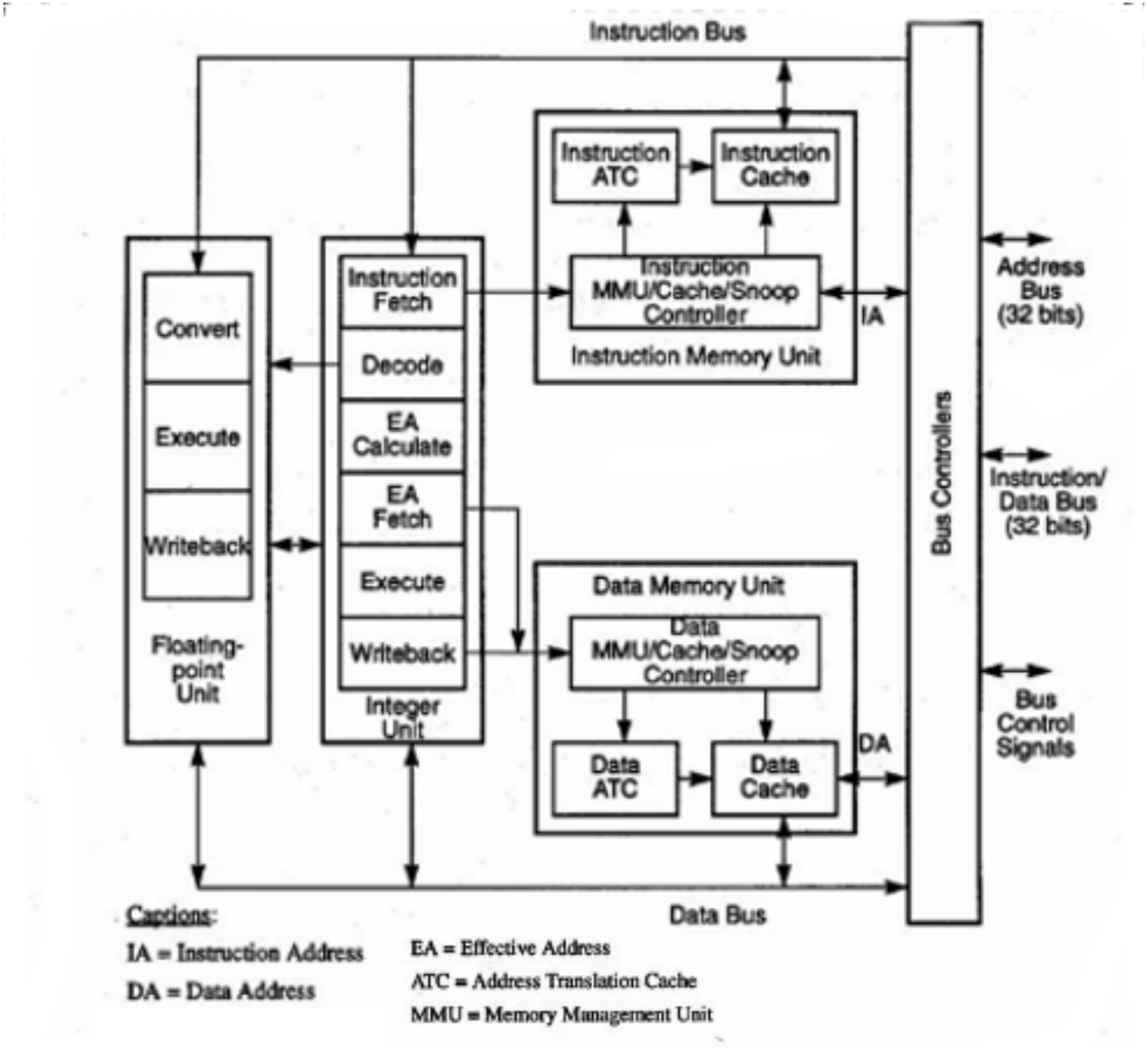
CISC Scalar Processors - 1990

Feature	**Intel i486**	**Motorola MC68040**
ISA - size and word length	157 instructions, 32 bits	113 instructions, 32 bits
On-chip cache and MMUs	8KB unified cache with separate MMUs	4 KB code cache, 4 KB data cache
Floating-point unit, registers and function units	On-chip with 8 FP registers, adder, multiplier, shifter	On-chip with 3 pipeline stages, eight 80-bit FP registers
Pipeline Stages	5	6

CISC Scalar Processors (continued) - 1990

Feature	**Intel i486**	**Motorola MC68040**
Protection levels	4	2
Memory organisation and TLB entries	Segmented paging with 4 KB per page and 32 entries in TLB	Paging with 4 or 8 KB per page, 64-entries in each ATC
Claimed performance	24 MIPS at 25 MHz	20 MIPS at 25 MHz, 30 MIPS at 60 MHz

CISC Architecture of MC68040 processor



Instruction Set Architecture - RISC

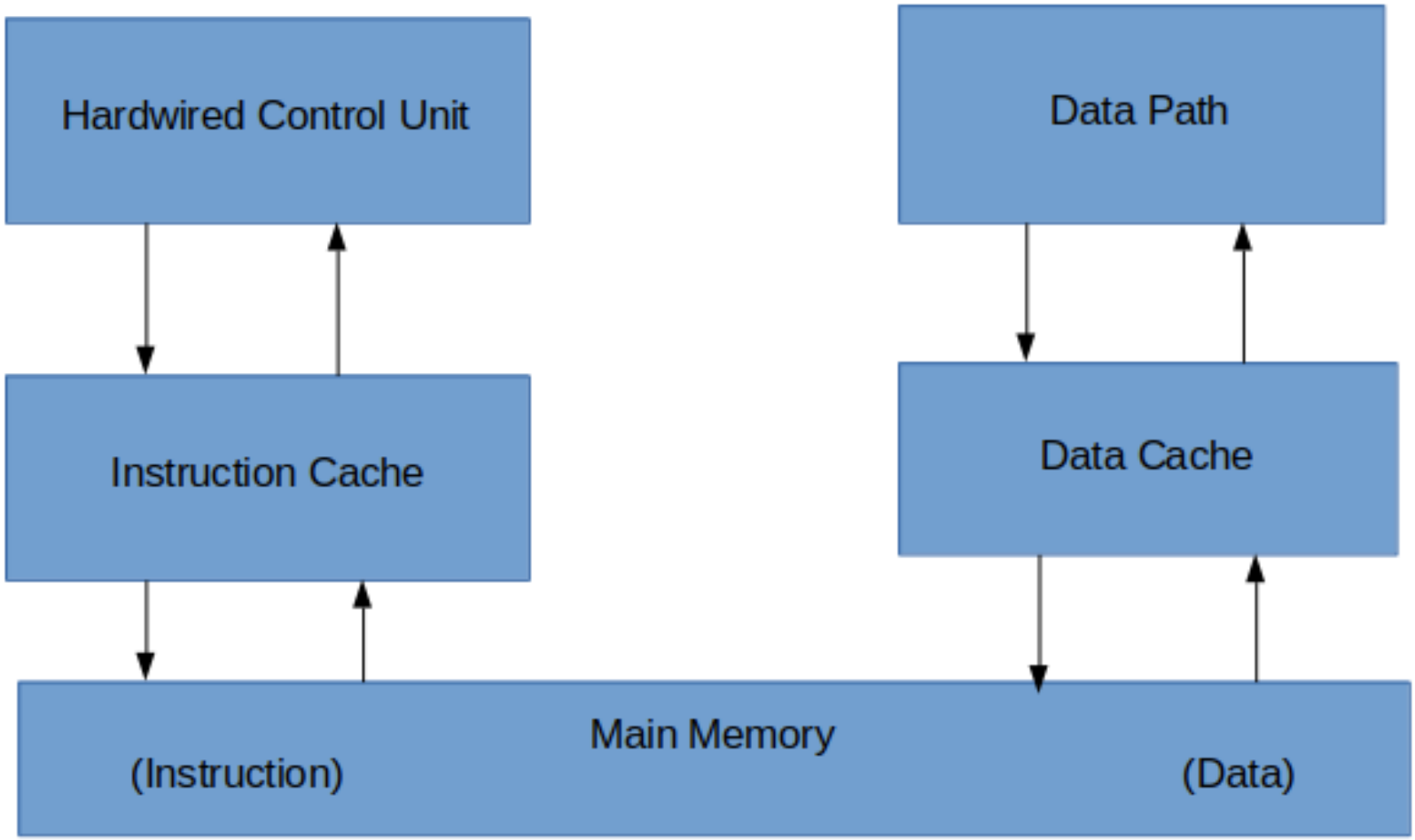


Fig: RISC Architecture with hardwired control, split instruction cache and data cache

RISC Scalar Processors - 1990

Feature	**Sun SPARC CY7C601**	**Intel i860**
ISA - formats and addressing modes	69 instructions, 32-bit format, 7 data types, 4-stage instr pipeline	82 instructions, 32-bit format, 4 addressing modes
Integer unit, GPRs Cache, MMU, Memory Organisation	32-bit RISC/IU, 136 registers in 8 windows Off-chip cache/MMU on CY7C604 with 64-entry TLB	32-bit RISC core, 32 registers 4KB code, 8KB data, On-chip MMU, paging with 4 KB/page
Floating-point unit, registers and function units	Off-chip FPU on CY7C602, 32-bit registers, 64-bit pipeline	On-chip 64-bit FP multiplier and FP adder with 32 FP registers, 3D graphics unit

RISC Scalar Processors (continued) - 1990

Feature	**Sun SPARC CY7C601**	**Intel i860**
Operation Modes	Concurrent IU and FPU operations	Allow dual instructions and dual FP operations
Technology, Clock rate, Packaging and Year	0.8 micrometer CMOS IV, 33 MHz, 207 pins,1989	1 micrometer CMHMOS IV, over 1 M transistors, 40 MHz, 168 pins, 1989
Claimed performance	24 MIPS for 33 MHz, 50 MIPS for 80 MHz	40 MIPS and 60 Mflops for 40 MHz

RISC Architecture of Intel i860 processor

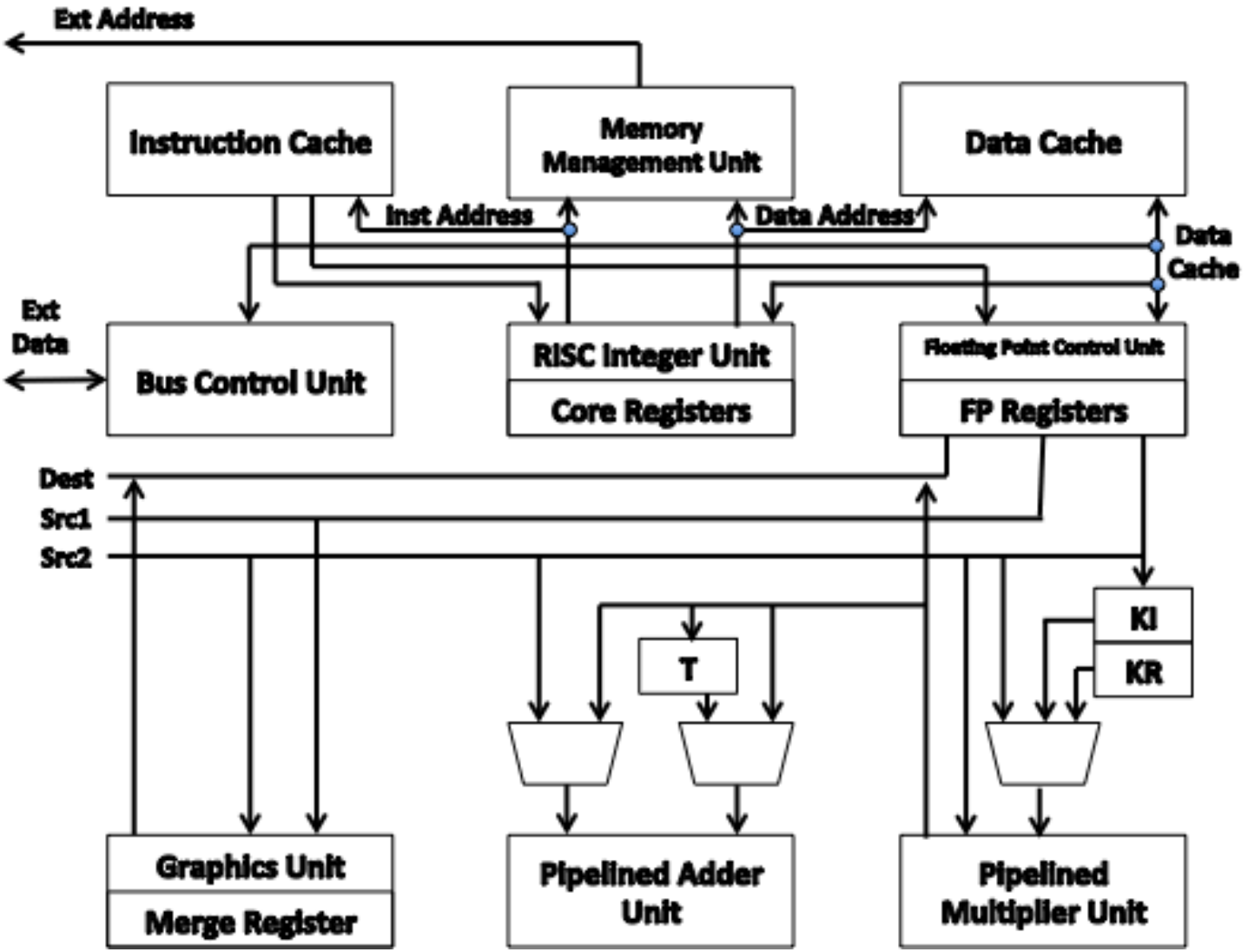


Fig: RISC Architecture with hardwired control, split instruction cache and data cache

Impact of RISC Technology

- Debate for supremacy of RISC over CISC lasted for a decade
- RISC will outperform CISC, if the program length doesn't increase drastically
- Note 1 : Converting from CISC to RISC will increase program size by 40 percent approx
- Increase in code length insignificant when compared with reduction in CPI
- Processor improvement includes full 64-bit architecture and supporting chips
- Note 2 : RISC and CISC are implemented with same hardware technology, blurring the boundary of distinction
- Chip manufacturers of modern day use a mix of both technologies in their current designs

CISC vs RISC

- CISC Advantages
  - Smaller program size (fewer instructions needed)
  - Simpler control unit design
  - Simpler compiler design
- RISC Advantages
  - Has potential to be faster
  - Many more registers
- RISC Problems
  - More complicated register decoding system
  - Hardwired control is less flexible than microcode

Superscalar, Vector Processors

- Scalar Processor

  Executes one instruction per cycle
  Usually only one instruction pipeline

- Superscalar Processor

  Multiple Instruction Pipelines present
  Multiple instructions issued per cycle
  Multiple results generated per cycle

- Vector Processor

  Issues one instruction that operate on multiple data items (array)
  Conducive to pipelining with one result produced per cycle

Superscalar Processors

- A subclass of RISC processors allowing multiple instructions to be issued simultaneously during each cycle
- Effective CPI is less than that of a generic scalar RISC processor
- Clock rates of scalar RISC and superscalar RISC machines almost similar

Superscalar Constraints

- Two instructions should not be issued at same time if they are not independent
- This restriction ties the instruction-level parallelism directly to the code being executed
- The instruction-issue degree in a superscalar processor is usually limited to 2 - 5 in practise.

Superscalar Pipelines

- One or more of pipelines in a superscalar processor may stall if insufficient functional units exist to perform an instruction phase (fetch, decode, execute, write-back).
- Ideally no more than one stall cycle should occur/
- A superscalar processor should be able to achieve the same effective parallelism as a vector machine with equivalent functional units.

Generic Superscalar Architecture

A typical superscalar architecture will have

- Multiple instruction pipelines
- An instruction cache that can provide multiple instruction per fetch
- Multiple buses among the function units

In theory, all functional units can be active simultaneously.

Very Long Instruction Word (VLIW) Architecture

- Has instruction words hundreds of bits in length
- All functional units share use of common large register file
- VLIW instruction typically is 256 or 1024 bits in length
- Different fields of instruction word will carry opcodes for different function units
- Programs written in conventional short instruction need to be compacted by compiler
- Effective CPI usually less than 1 (0.33 in example below) lower than superscalar computers
- Decoding of VLIW instructions usually easier than superscalar instructions

Pipelining in VLIW Processors

Decoding of instructions is easier as each region of instruction word is usually limited to the type of instruction it can contain

Code density in VLIW is less than in superscalars, because if a region of a VLIW word isn't needed in a particular instruction, it must still exist (to be filled with a NOP)

Note: Superscalars can be compatible with scalar processors; such compatibility is difficult with VLIW parallel and non-parallel architectures

VLIW Opportunities

- Random parallelism among scalar operations is exploited in VLIW, instead of regular parallelism in a vector or SIMD machine.
- Efficiency of machine is entirely dictated by success or goodness of compiler in planning the operations placed in same instruction words.
- Different implementations of same VLIW architecture may not be binary compatible with each other, resulting in different latencies.

VLIW Processors

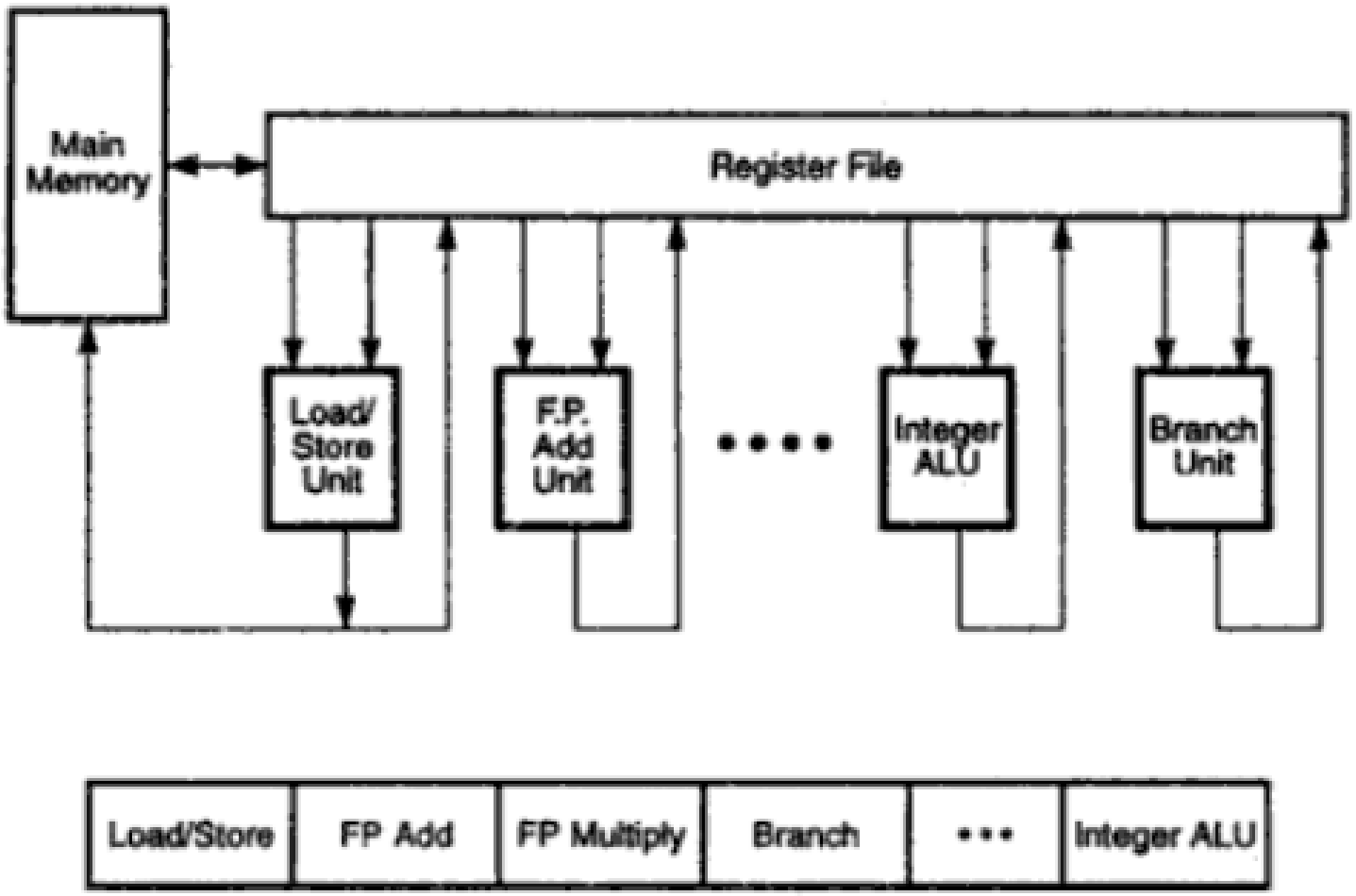


Fig: Instruction format in a typical VLIW processor

VLIW Execution

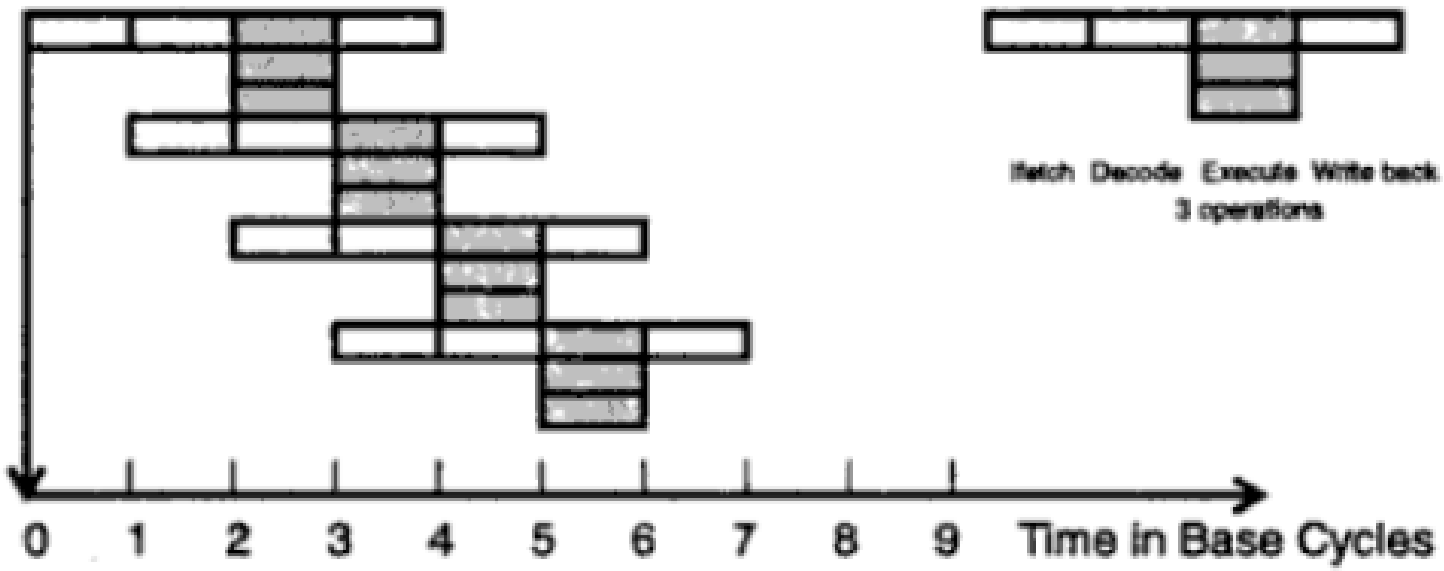
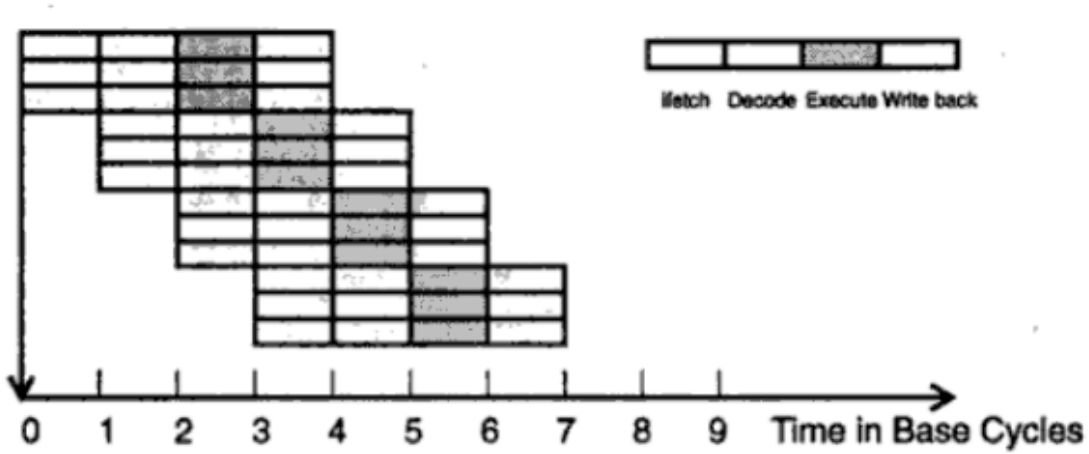


Fig: VLIW Execution with degree m = 3, effective CPI = 0.33

VLIW Summary

- VLIW reduces effort required to detect parallelism using hardware or software techniques.
- Main advantage of VLIW architecture is its simplicity in hardware structure and instruction set.
- Unfortunately, VLIW does require careful analysis of code in order to "compact" the most appropriate "short" instructions into a VLIW word.

Superscalar Processors



Pipelining Concept in Superscalar Processors - here degree m = 3
m instructions per cycle can be executed

Presentation Outline

Contents of Module III

Contents of Module III

- Multiprocessors System Interconnect

Contents of Module III

- Multiprocessors System Interconnect
- Cache Coherence and Synchronization Mechanisms

Contents of Module III

- Multiprocessors System Interconnect
- Cache Coherence and Synchronization Mechanisms
- Cache Coherence Problem

Contents of Module III

- Multiprocessors System Interconnect
- Cache Coherence and Synchronization Mechanisms
- Cache Coherence Problem
- Snoopy Bus Protocol

Contents of Module III

- Multiprocessors System Interconnect
- Cache Coherence and Synchronization Mechanisms
- Cache Coherence Problem
- Snoopy Bus Protocol
- Directory Based Protocol

Contents of Module III

- Multiprocessors System Interconnect
- Cache Coherence and Synchronization Mechanisms
- Cache Coherence Problem
- Snoopy Bus Protocol
- Directory Based Protocol
- Hardware Synchronization Problem

Multiprocessors and Multicomputers

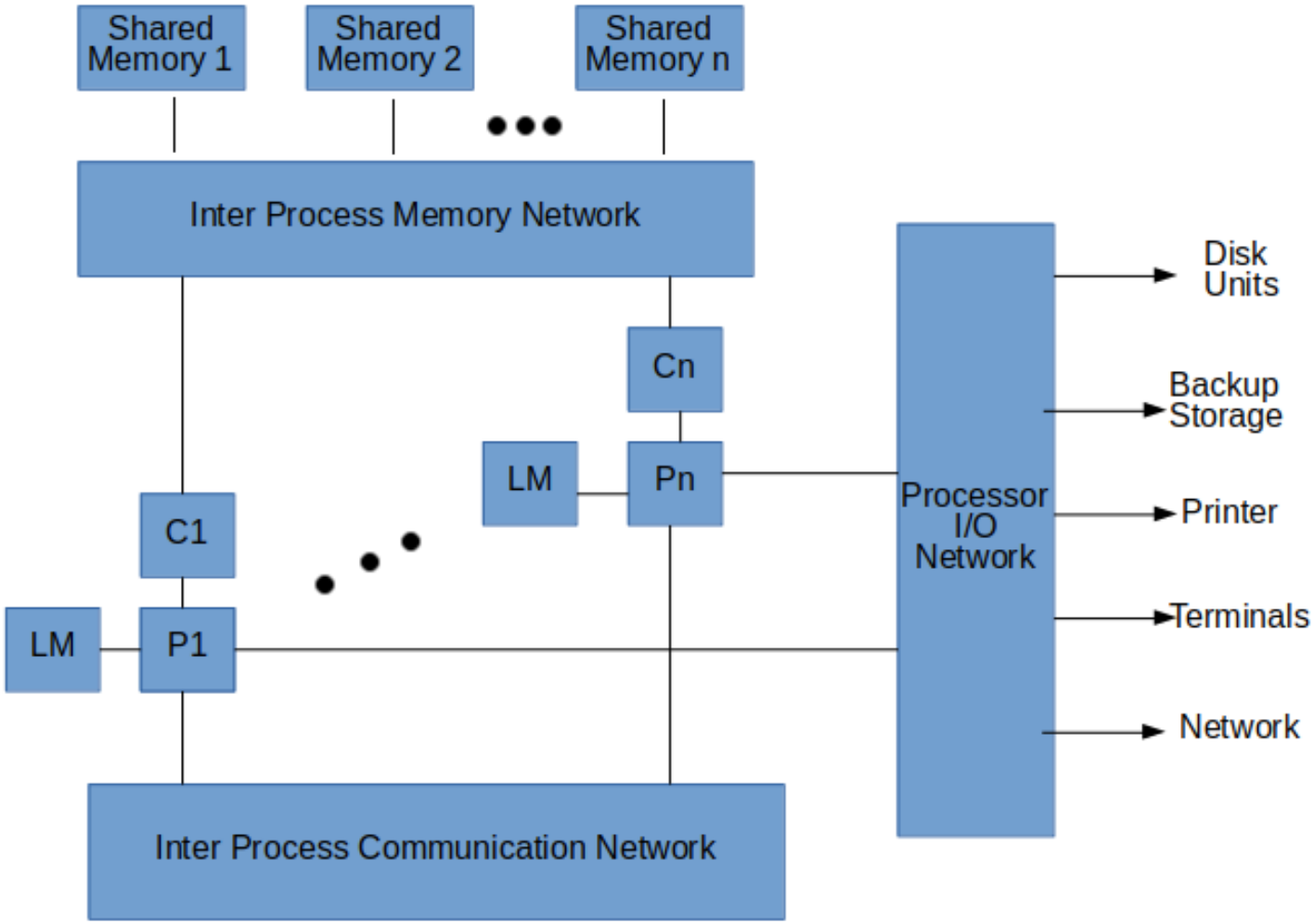


Fig: Interconnection Structures in a Generalised Multiprocessor System with Local Memory

Network Characteristics

- Design choices for multiprocessor networks
  - Timing Protocol

    Synchronous - controlled by a global clock
    Asynchronous - uses handshaking or interlocking mechanisms
  - Switching Method

    Circuit Switching - device gets path for whole duration
    Packet Switching - information packets compete for single path
  - Network Control Strategy

    Centralised - global controller manages requests
    Distributed - requests are handled independently

Hierarchical Bus Systems

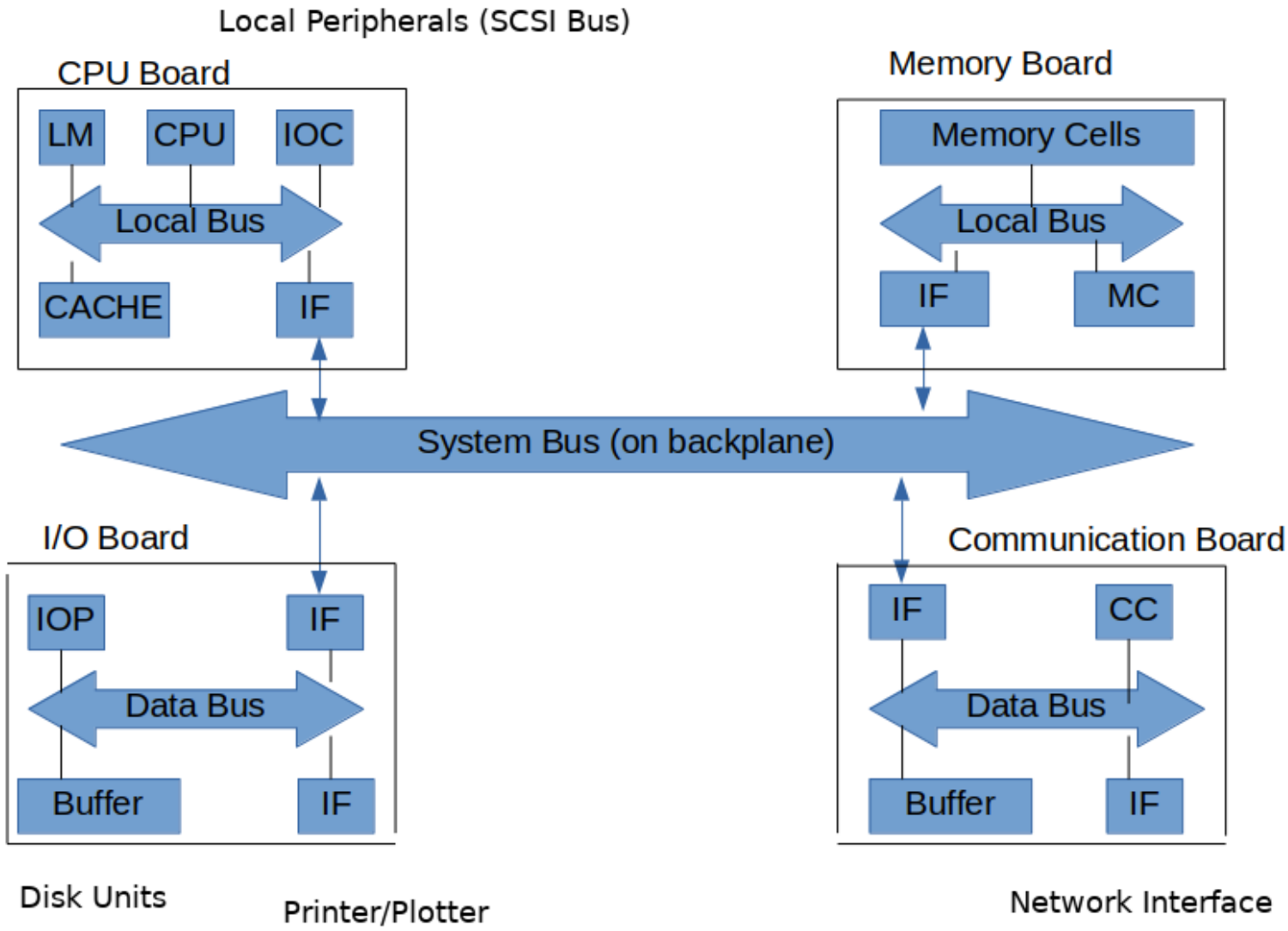


Fig: Bus Systems at Board Level, Backplane Level and I/O Level

Hierarchical Bus Systems

- Concepts
  - Local Bus

    Buses implemented within processor chips on on PCBs
    Eg: Memory Bus - to connect memory with interface logic
  - Backplane Bus

    A PCB on which other boards are plugged in using connectors
    Consists of shared signal paths and utility lines
  - I/O Bus

    Used to connect I/O devices like SCSI bus
    Specs include application profiles alongwith logical, electrical and mechanical properties

Hierarchical Cache/Bus Systems

Second level cache memory size should be equal to sum of all first level cache memory

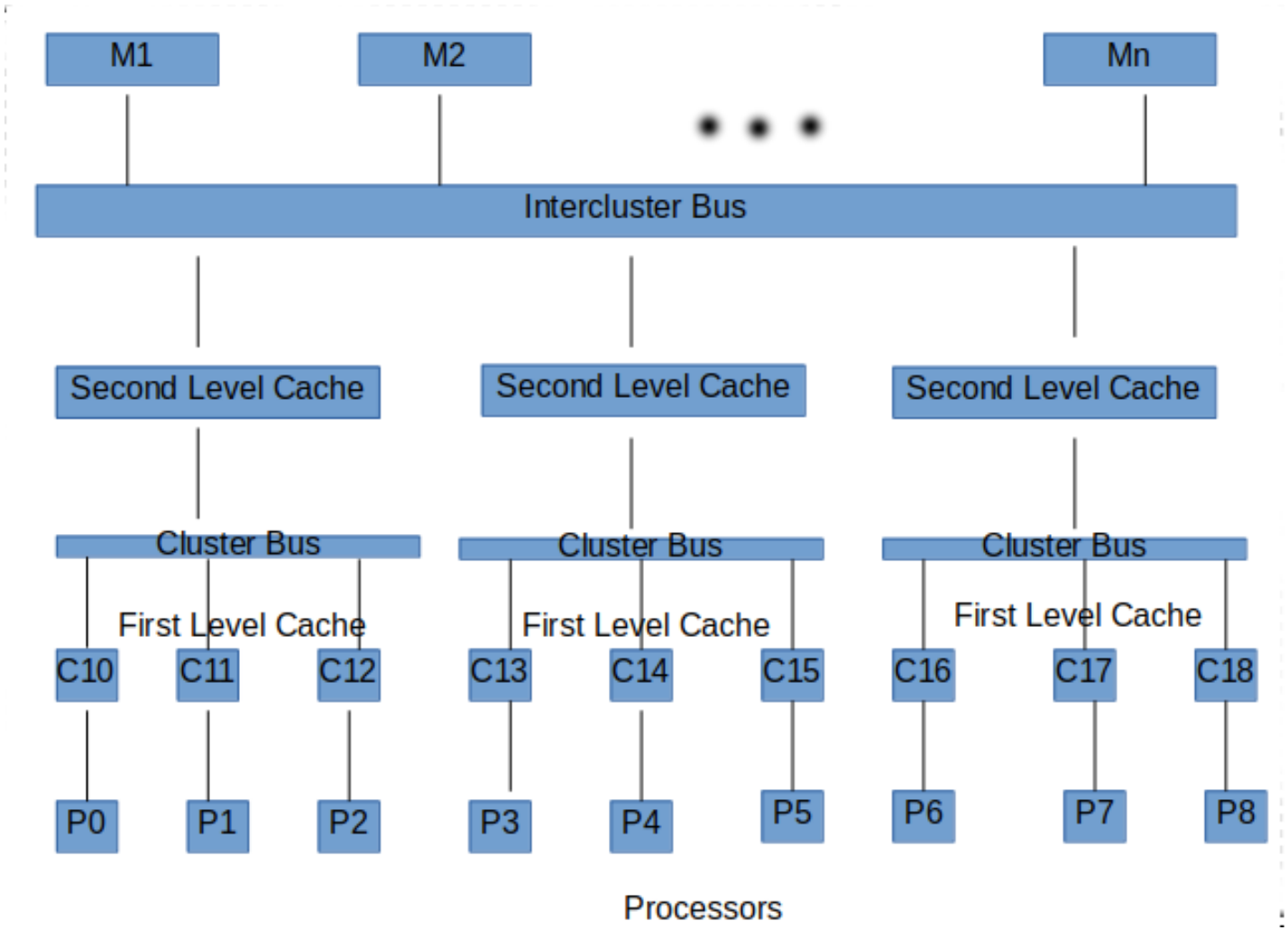


Fig: Hierarchical cache/bus architecture for designing a scalable multiprocessor

Hierarchical Cache/Bus Systems

Arbitration logic will extend memory access to any of the processor based on its own logic for handling requests/acknowledges from the processors - according to some fairness or priority rules.

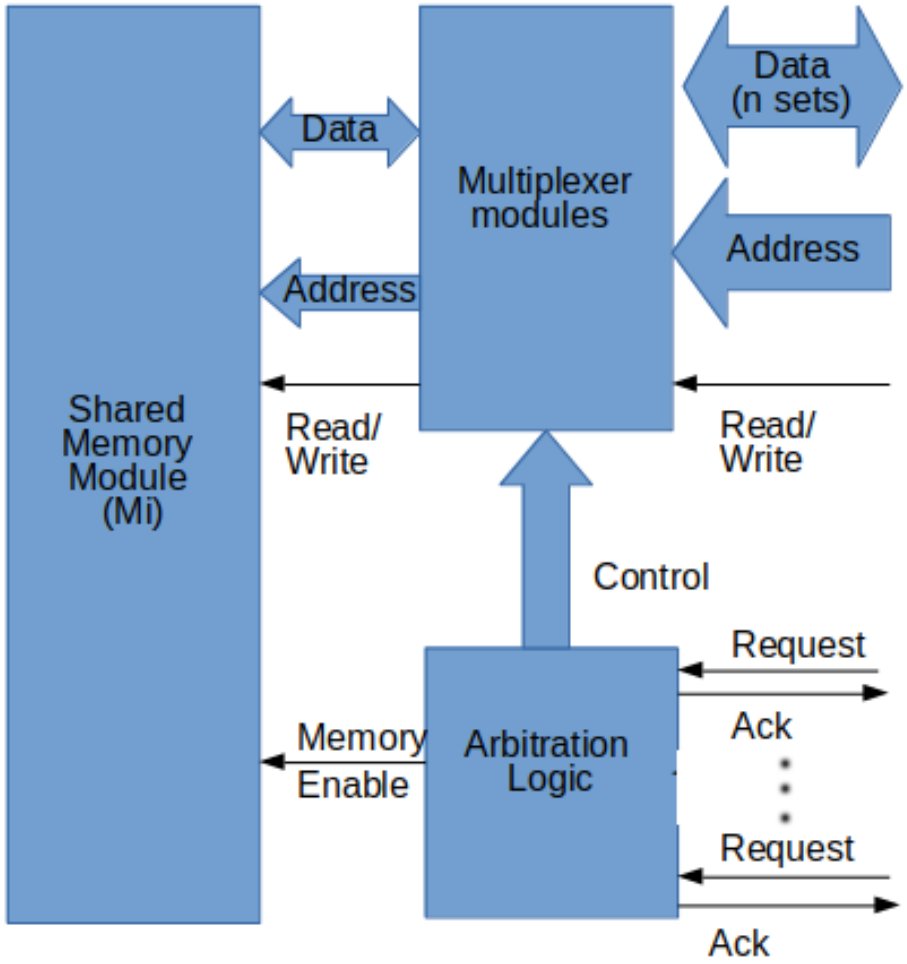
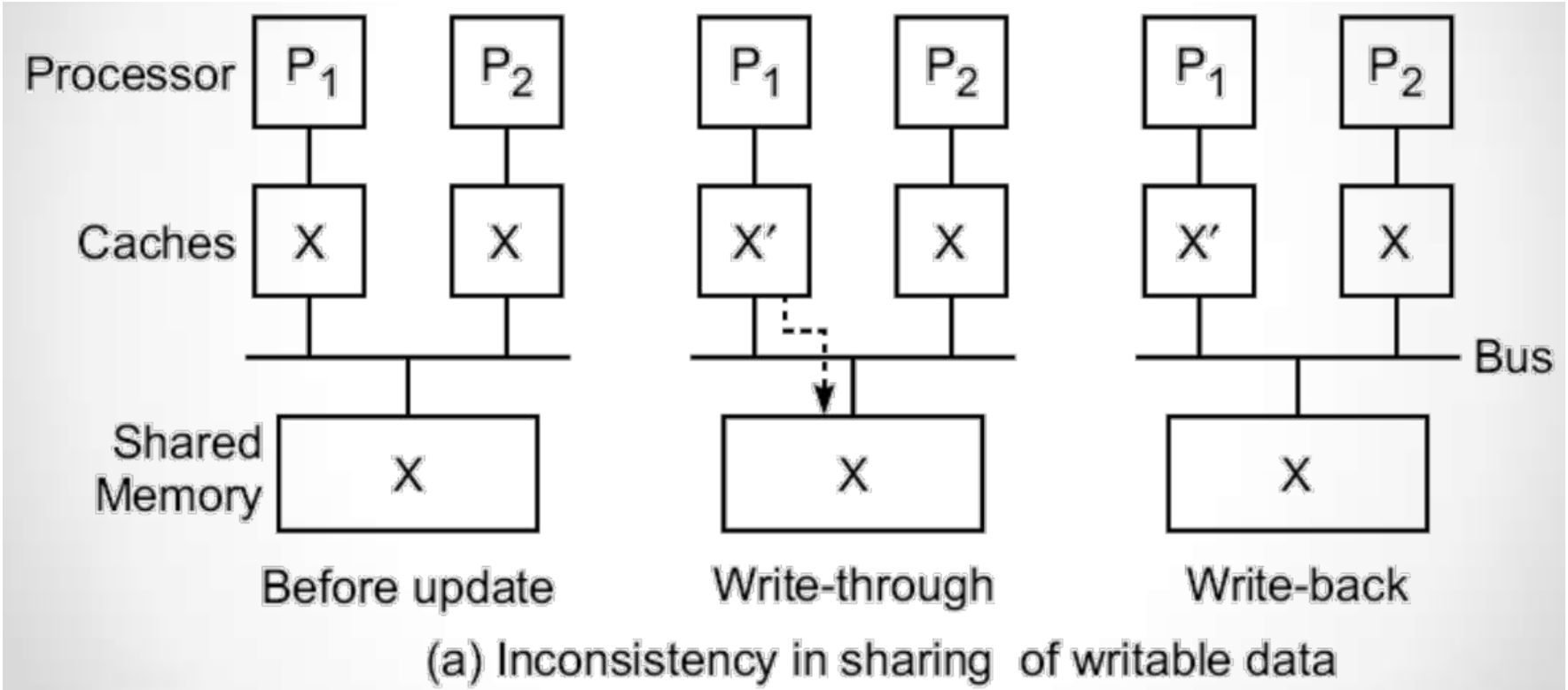


Fig: Crosspoint switches in a crossbar network

Cache Coherency - Problem and Approaches

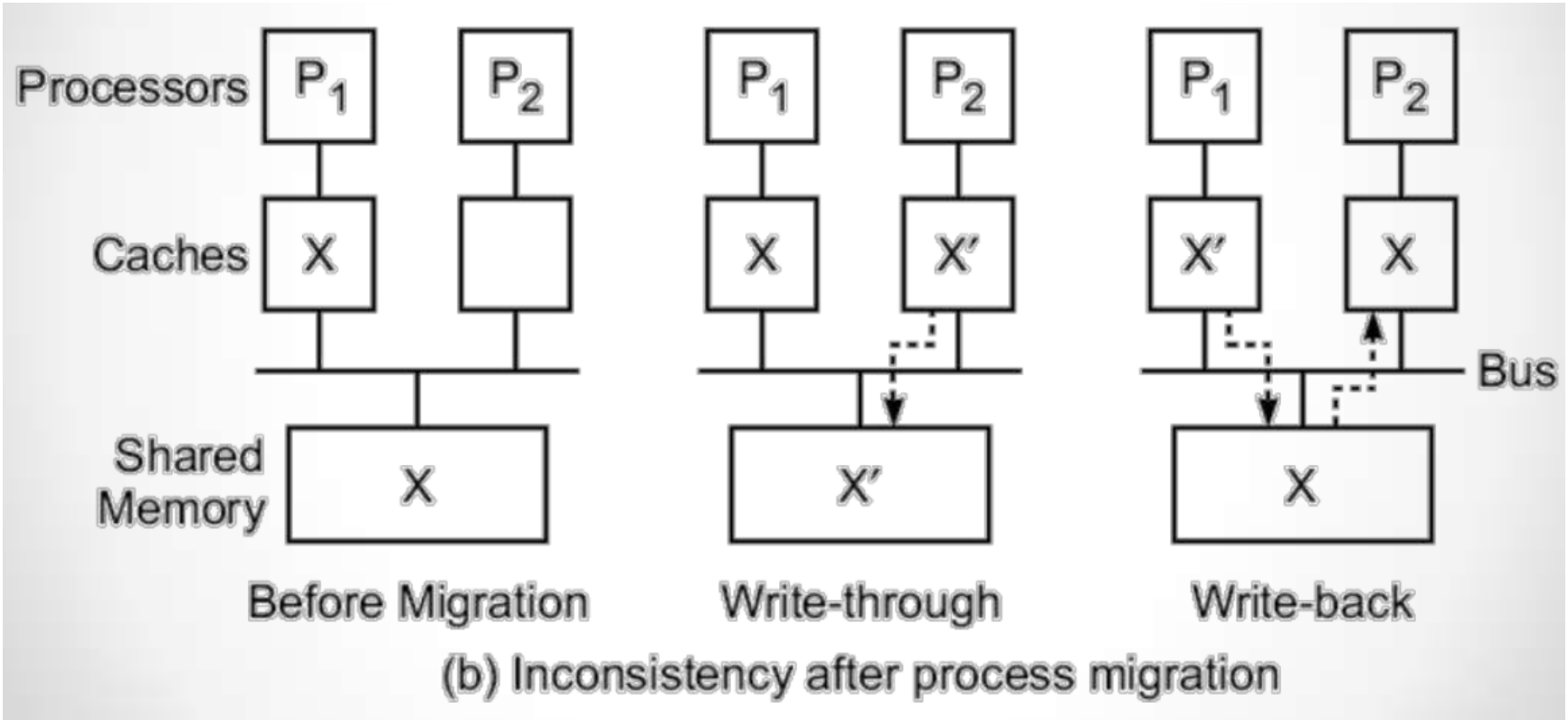
- Cache Coherence Problem
  - Inconsistent copies of same memory block in different caches
  - ***Sources of inconsistency***:
    - Sharing of writable data
    - Process migration
    - I/O activity
- Protocol Approaches
  - Snoopy Bus Protocol
  - Directory Based Protocol
- Write Policies
  - (Write-back, Write-through) **x** (Write-invalidate, Write-update)

Cache Coherency - inconsistency in sharing data

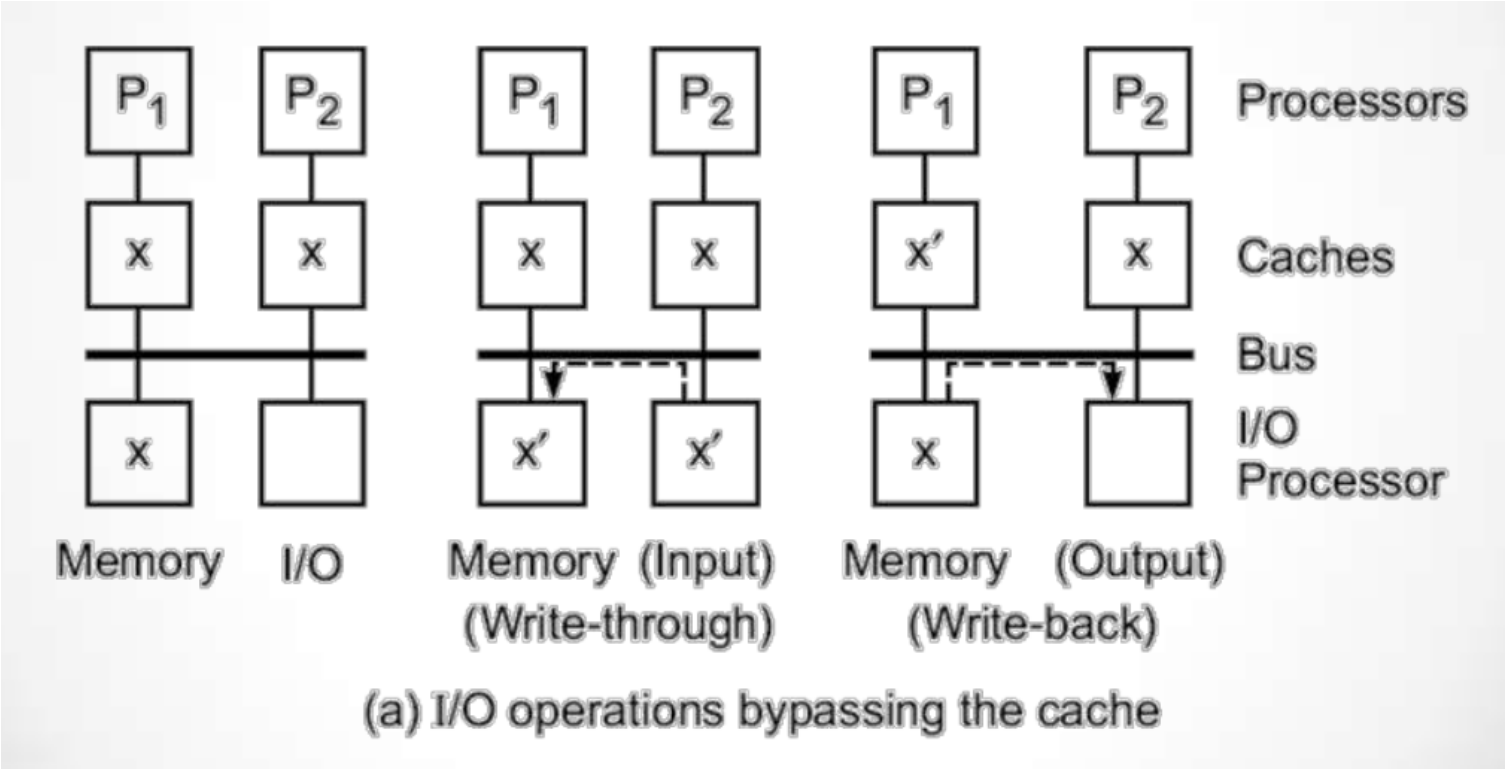


(a) Inconsistency in sharing of writable data

Cache Coherency - inconsistency after process migration

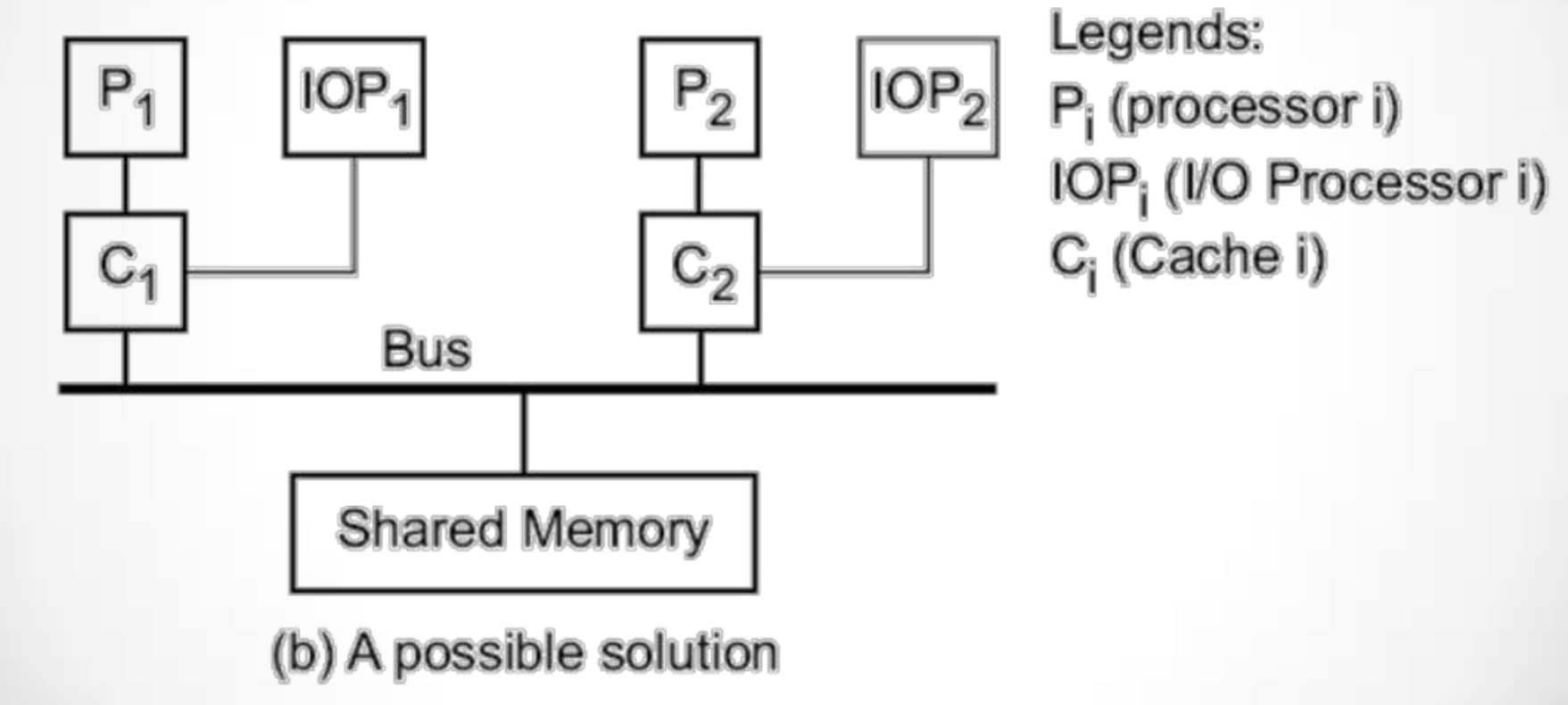


Cache Coherency - I/O inconsistencies



(a) I/O operations bypassing the cache

Cache Coherency - Introducing I/O processors



(b) A possible solution

Cache Coherency - Snoopy Bus Protocols

- Based on watching bus activities and carry out the appropriate coherency commands when necessary.
- Global memory is moved in blocks, and each block has a state associated with it, which determines what happens to the entire contents of the block.
- The state of a block might change as a result of the operations Read-Miss, Read-Hit, Write-Miss, and Write-Hit.

Write-Invalidate and Write-Through

- Multiple processors can read block copies from main memory safely until one processor updates its copy.
- At this time, all cache copies are invalidated and the memory is updated to remain consistent.

Snoopy Bus Protocols

State	Description
Valid [VALID]	The copy is consistent with global memory
Invalid [INV]	The copy is inconsistent

Snoopy Bus Protocols

- Write-Invalidate and Write-Through

Event	Actions
Read Hit	Use the local copy from the cache.
Read Miss	Fetch a copy from global memory. Set the state of this copy to Valid.
Write Hit	Perform the write locally. Broadcast an Invalid command to all caches. Update the global memory.
Write Miss	Get a copy from global memory. Broadcast an invalid command to all caches. Update the global memory. Update the local copy and set its state to Valid.
Replace	Since memory is always consistent, no write back is needed when a block is replaced.

Snoopy Bus Protocols

- Write-Invalidate and Write-Back (Ownership Protocol)
  - A valid block can be owned by memory and shared in multiple caches that can contain only the shared copies of the block.
  - Multiple processors can safely read these blocks from their caches until one processor updates its copy.
  - At this time, the writer becomes the only owner of the valid block and all other copies are invalidated.

Snoopy Bus Protocols

- Write-Invalidate and Write-Back (Ownership Protocol)

State	Description
Shared (Read-Only) [RO]	Data is valid and can be read safely. Multiple copies can be in this state
Exclusive (Read-Write) [RW]	Only one valid cache copy exists and can be read from and written to safely. Copies in other caches are invalid
Invalid [INV]	The copy is inconsistent

Snoopy Bus Protocols

- Write-Invalidate and Write-Back

Event	Action
Read Hit	Use the local copy from the cache.
Read Miss:	If no Exclusive (Read-Write) copy exists, then supply a copy from global memory. Set the state of this copy to Shared (Read-Only). If an Exclusive (Read-Write) copy exists, make a copy from the cache that set the state to Exclusive (Read-Write), update global memory and local cache with the copy. Set the state to Shared (Read-Only) in both caches.

Snoopy Bus Protocols

- Write-Invalidate and Write-Back

Write Hit	**If the copy is Exclusive (Read-Write), perform the write locally. If the state is Shared (Read-Only), then broadcast an Invalid to all caches. Set the state to Exclusive (Read-Write).**
Write Miss	**Get a copy from either a cache with an Exclusive (Read-Write) copy, or from global memory itself. Broadcast an Invalid command to all caches. Update the local copy and set its state to Exclusive (Read-Write).**
Block Replacement	**If a copy is in an Exclusive (Read-Write) state, it has to be written back to main memory if the block is being replaced. If the copy is in Invalid or Shared (Read-Only) states, no write back is needed when a block is replaced.**

Snoopy Bus Protocols

- Write-Once
  - This write-invalidate protocol, which was proposed by Goodman in 1983 uses a combination of write-through and write-back.
  - Write-through is used the very first time a block is written. Subsequent writes are performed using write back.

Snoopy Bus Protocols

- Write-Once

State	Description
Invalid [INV]	The copy is inconsistent.
Valid [VALID]	The copy is consistent with global memory.
Reserved [RES]	Data has been written exactly once and the copy is consistent with global memory. There is only one copy of the global memory block in one local cache.
Dirty [DIRTY]	Data has been updated more than once and there is only one copy in one local cache. When a copy is dirty, it must be written back to global memory

Snoopy Bus Protocols

- Write-Once

Event	Actions
Read Hit	Use the local copy from the cache.
Read Miss	If no Dirty copy exists, then supply a copy from global memory. Set the state of this copy to Valid. If a dirty copy exists, make a copy from the cache that set the state to Dirty, update global memory and local cache with the copy. Set the state to VALID in both caches.

Snoopy Bus Protocols

- Write-Once

Write Hit	**If the copy is Dirty or Reserved, perform the write locally, and set the state to Dirty. If the state is Valid, then broadcast an Invalid command to all caches. Update the global memory and set the state to Reserved.**
Write Miss	**Get a copy from either a cache with a Dirty copy or from global memory itself. Broadcast an Invalid command to all caches. Update the local copy and set its state to Dirty.**
Block Replacement	**If a copy is in a Dirty state, it has to be written back to main memory if the block is being replaced. If the copy is in Valid, Reserved, or Invalid states, no write back is needed when a block is replaced.**

Snoopy Bus Protocols

- Write-Update and Partial Write-Through
  - In this protocol an update to one cache is written to memory at the same time it is broadcast to other caches sharing the updated block.
  - These caches snoop on the bus and perform updates to their local copies.
  - There is also a special bus line, which is asserted to indicate that at least one other cache is sharing the block.

Snoopy Bus Protocols

- Write-Update and Partial Write-Through

State	Description
Valid Exclusive [VAL-X]	This is the only cache copy and is consistent with global memory
Shared [SHARE]	There are multiple caches copies shared. All copies are consistent with memory
Dirty [DIRTY]	This copy is not shared by other caches and has been updated. It is not consistent with global memory. (Copy ownership)

Snoopy Bus Protocols

- Write-Update and Partial Write-Through

Event	Action
Read Hit	Use the local copy from the cache. State does not change
Read Miss:	If no other cache copy exists, then supply a copy from global memory. Set the state of this copy to Valid Exclusive. If a cache copy exists, make a copy from the cache. Set the state to Shared in both caches. If the cache copy was in a Dirty state, the value must also be written to memory.

Snoopy Bus Protocols

- Write-Update and Partial Write-Through

Write Hit	**Perform the write locally and set the state to Dirty. If the state is Shared, then broadcast data to memory and to all caches and set the state to Shared. If other caches no longer share the block, the state changes from Shared to Valid Exclusion.**
Write Miss	**The block copy comes from either another cache or from global memory. If the block comes from another cache, perform the update and update all other caches that share the block and global memory. Set the state to Shared. If the copy comes from memory, perform the write and set the state to Dirty.**
Block Replacement	**If a copy is in a Dirty state, it has to be written back to main memory if the block is being replaced. If the copy is in Valid Exclusive or Shared states, no write back is needed when a block is replaced.**

Snoopy Bus Protocols

- Write-Update and Write-Back
  - This protocol is similar to the previous one except that instead of writing through to the memory whenever a shared block is updated, memory updates are done only when the block is being replaced.

Snoopy Bus Protocols

- Write-Update and Write-Back

State	Description
Valid Exclusive [VAL-X]	This is the only cache copy and is consistent with global memory
Shared Clean [SH-CLN]	There are multiple caches copies shared.
Shared Dirty [SH-DRT]	There are multiple shared caches copies. This is the last one being updated. (Ownership)
Dirty [DIRTY]	This copy is not shared by other caches and has been updated. It is not consistent with global memory. (Ownership)

Snoopy Bus Protocols

- Write-Update and Write-Back

Event	Action
Read Hit	Use the local copy from the cache. State does not change
Read Miss:	If no other cache copy exists, then supply a copy from global memory. Set the state of this copy to Valid Exclusive. If a cache copy exists, make a copy from the cache. Set the state to Shared Clean. If the supplying cache copy was in a Valid Exclusion or Shared Clean, its new state becomes Shared Clean. If the supplying cache copy was in a Dirty or Shared Dirty state, its new state becomes Shared Dirty.

Snoopy Bus Protocols

- Write-Update and Write-Back

Write Hit	**If the sate was Valid Exclusive or Dirty, Perform the write locally and set the state to Dirty. If the state is Shared Clean or Shared Dirty, perform update and change state to Shared Dirty. Broadcast the updated block to all other caches. These caches snoop the bus and update their copies and set their state to Shared Clean.**
Write Miss	**The block copy comes from either another cache or from global memory. If the block comes from another cache, perform the update, set the state to Shared Dirty, and broadcast the updated block to all other caches. Other caches snoop the bus, update their copies, and change their state to Shared Clean. If the copy comes from memory, perform the write and set the state to Dirty.**
Block Replacement	**If a copy is in a Dirty or Shared Dirty state, it has to be written back to main memory if the block is being replaced. If the copy is in Valid Exclusive, no write back is needed when a block is replaced.**

Directory Based Protocol

- Due to the nature of some interconnection networks and the size of the shared memory system, updating or invalidating caches using snoopy protocols might become unpractical.

- Cache coherence protocols that somehow store information on where copies of blocks reside are called directory schemes.

Directory Based Protocol

- A directory is a data structure that maintains information on the processors that share a memory block and on its state.
- The information maintained in the directory could be either centralized or distributed.
- A Central directory maintains information about all blocks in a central data structure.
- The same information can be handled in a distributed fashion by allowing each memory module to maintain a separate directory.

Directory Based Protocol

Protocol Categorization:

- Full Map Directories.
- Limited Directories.
- Chained Directories.

Directory Based Protocol

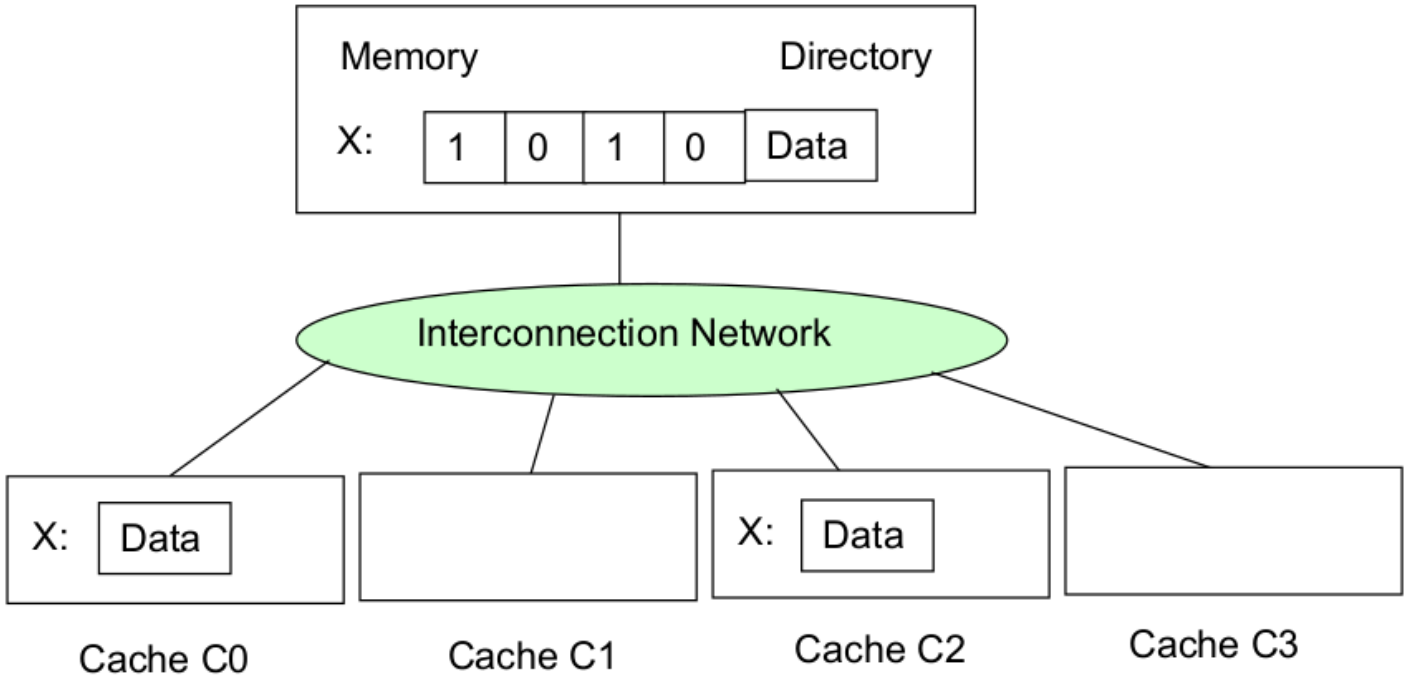
Full-map Directories:

-

- Each directory entry contains *N* pointers, where *N* is the number of processors.
- There could be *N* cached copies of a particular block shared by all processors.
- For every memory block, an *N* bit vector is maintained, where *N* equals the number of processors in the shared memory system. Each bit in the vector corresponds to one processor.

Directory Based Protocol

– Full-map Directories:



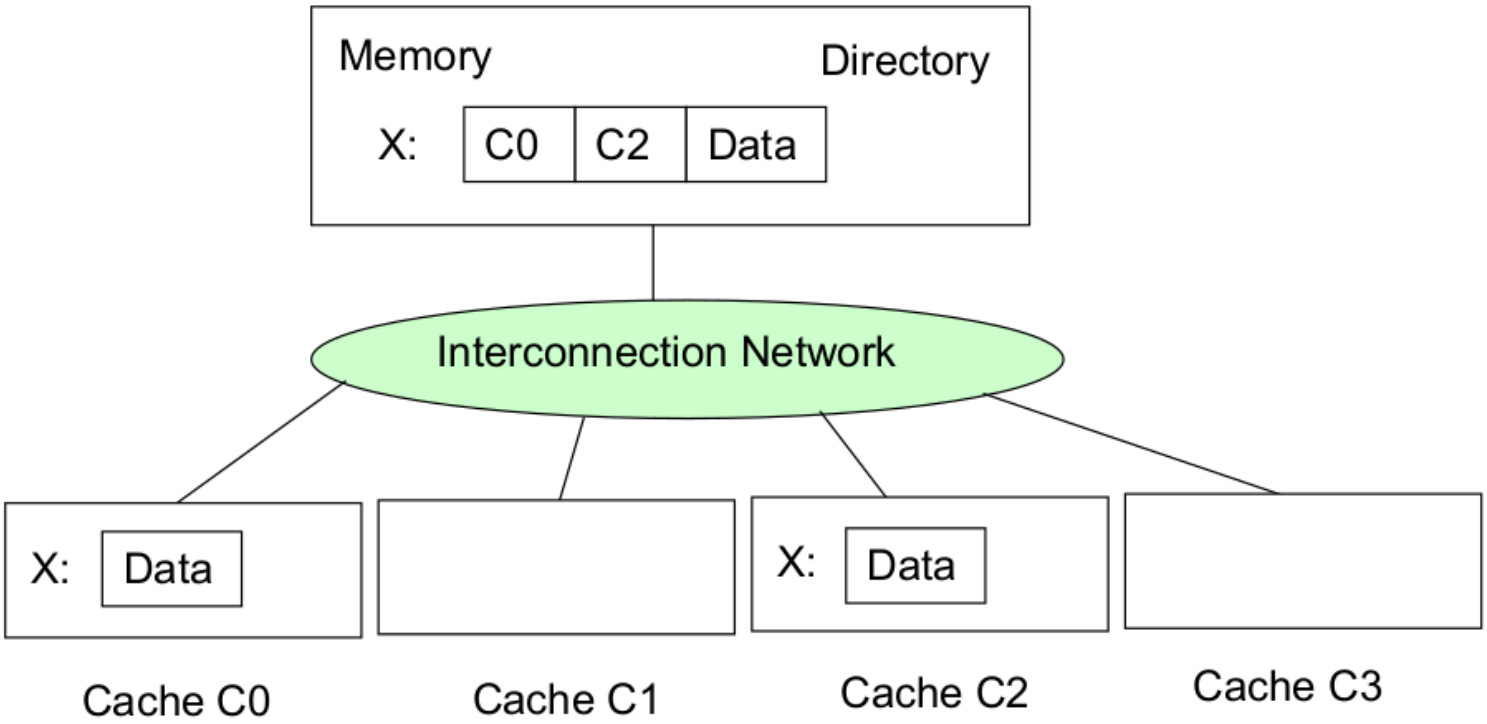
Directory Based Protocol

– Limited Directories:

- Fixed number of pointers per directory entry regardless of the number of processors.
- Restricting the number of simultaneously cached copies of any block should solve the directory size problem that might exist in full-map directories.

Directory Based Protocol

– Limited Directories:



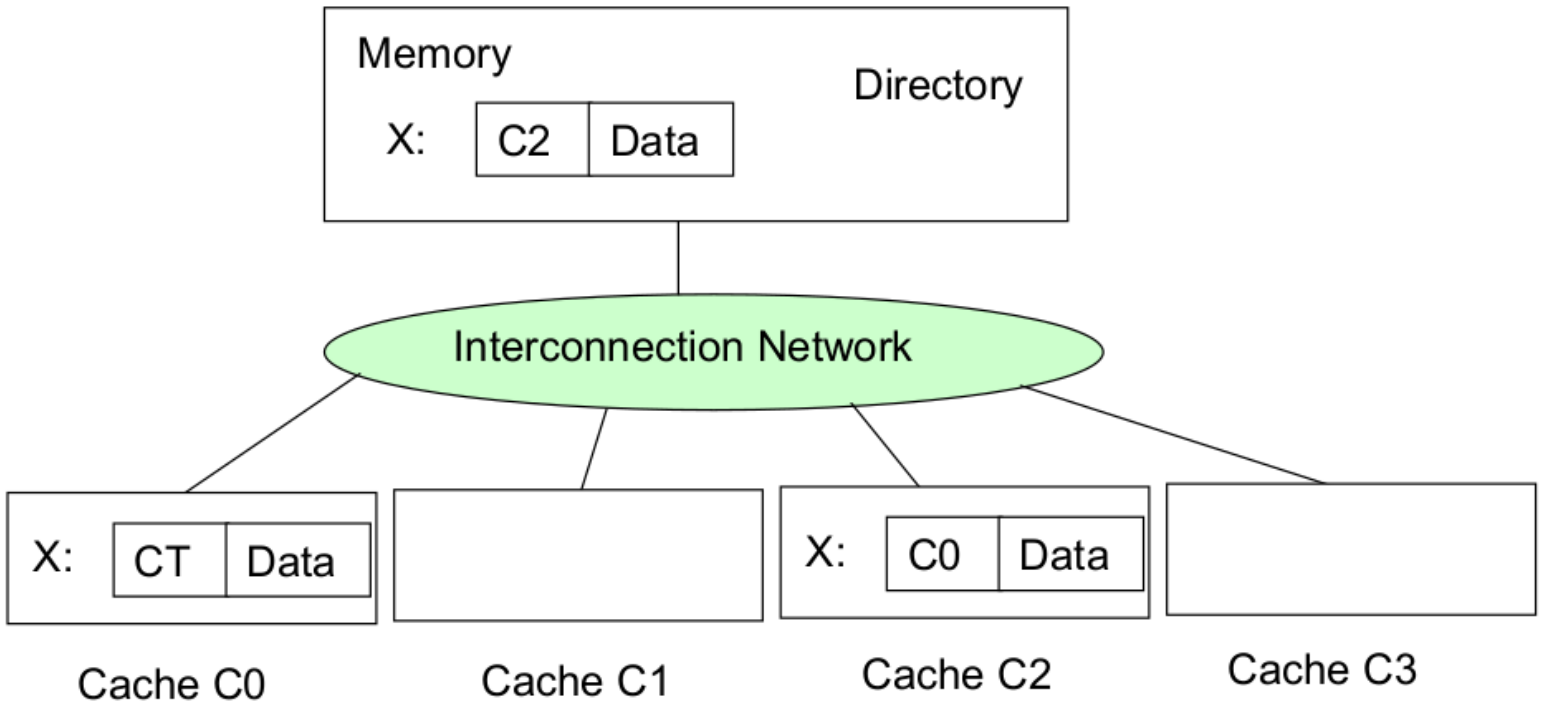
Directory Based Protocol

– Chained Directories:

- Chained directories emulate full-map by distributing the directory among the caches.
- Solving the directory size problem without restricting the number of shared block copies.
- Chained directories keep track of shared copies of a particular block by maintaining a chain of directory pointers.

Directory Based Protocol

– Chained Directories:



Directory Based Protocol

Invalidate Protocols:

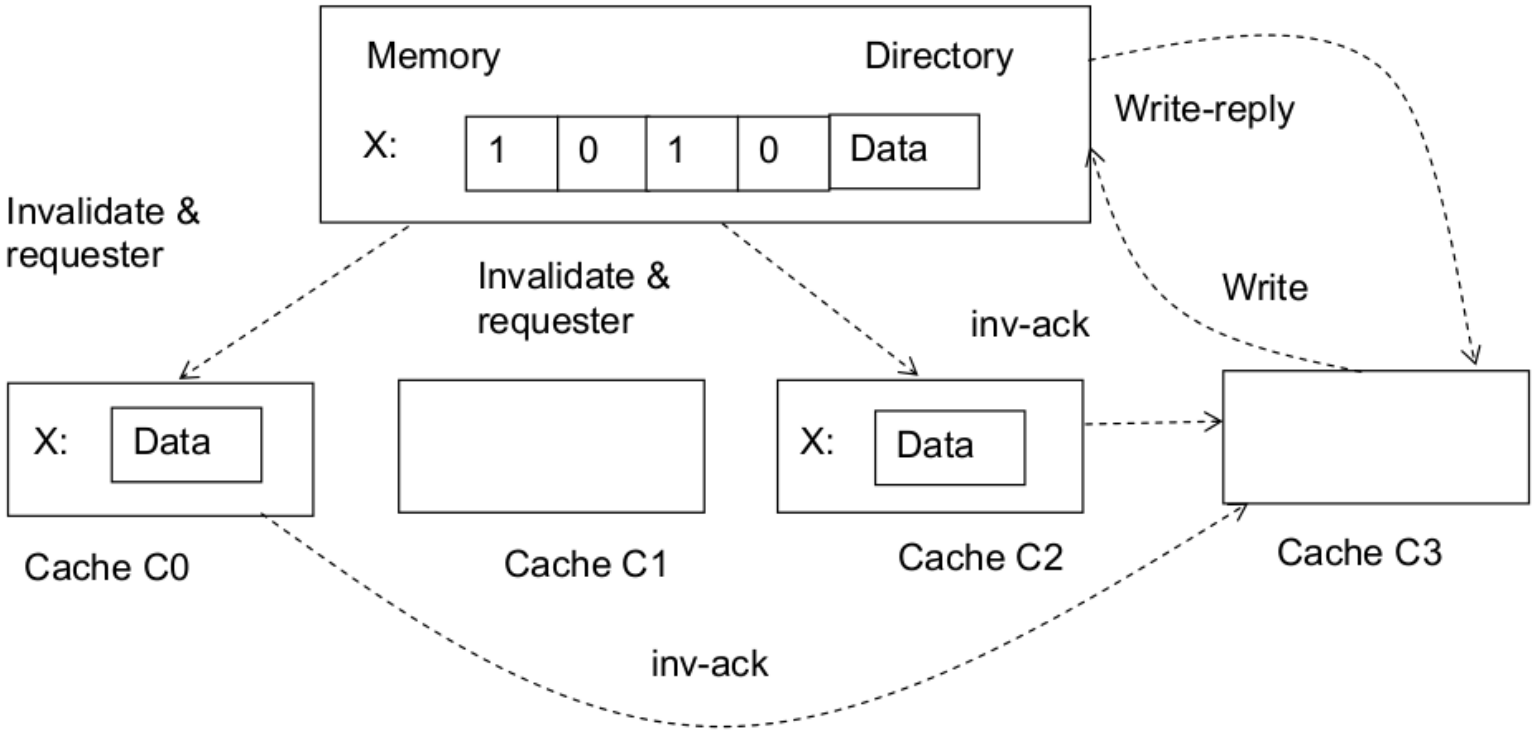
- Centralized directory invalidate.
- Scalable Coherent Interface (SCI).
- Stanford Distributed Directory (SDD).

Centralized Directory Invalidate

- Invalidating signals and a pointer to the requesting processor are forwarded to all processors that have a copy of the block.
- Each invalidated cache sends an acknowledgment to the requesting processor.
- After the invalidation is complete, only the writing processor will have a cache with a copy of the block.

Directory Based Protocol

Centralized Directory Invalidate – Write by P3



Directory Based Protocol

Scalable Coherent Interface (SCI)

- Doubly linked list of distributed directories.
- Each cached block is entered into a list of processors sharing that block.
- For every block address, the memory and cache entries have additional tag bits. Part of the memory tag identifies the first processor in the sharing list (the head). Part of each cache tag identifies the previous and following sharing list entries.

Scalable Coherent Interface (SCI)

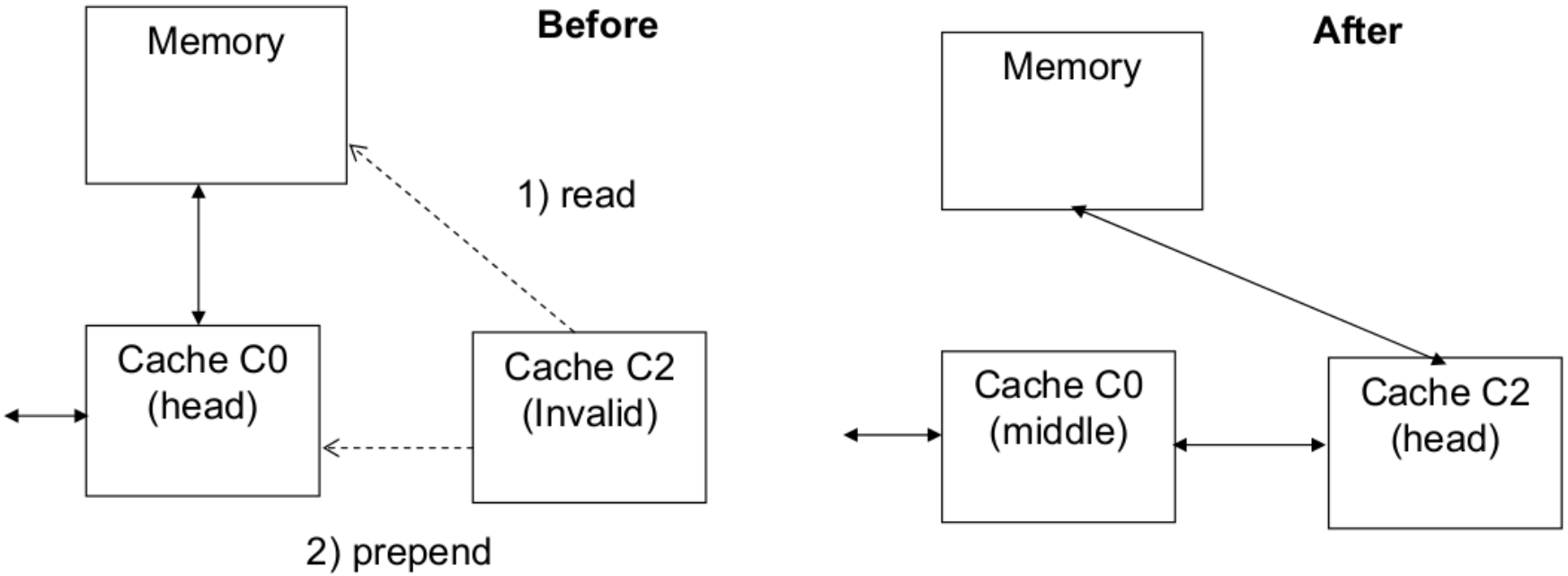
- Initially memory is in the uncached state and cached copies are invalid.
- A read request is directed from a processor to the memory controller. The requested data is returned to the requester's cache and its entry state is changed from invalid to the head state.
- This changes the memory state from uncached to cached.

Scalable Coherent Interface (SCI)

- When a new requester directs its read request to memory, the memory returns a pointer to the head.
- A cache-to-cache read request (called Prepend) is sent from the requester to the head cache.
- On receiving the request, the head cache sets its backward pointer to point to the requester's cache.
- The requested data is returned to the requester's cache and its entry state is changed to the head state.

Directory Based Protocol

- Scalable Coherent Interface (SCI)-Sharing list addition



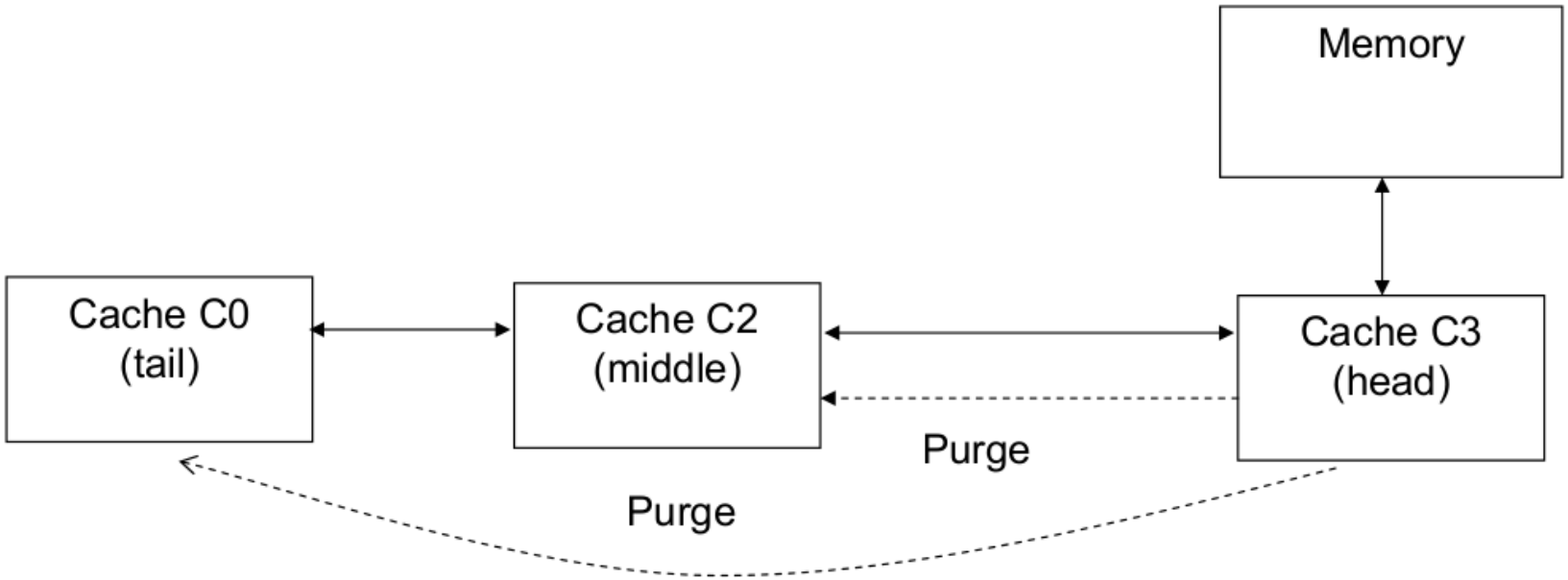
Directory Based Protocol

Scalable Coherent Interface (SCI)

– The head of the list has the authority to purge other entries in the list to obtain an exclusive (read-write) entry.

Directory Based Protocol

- Scalable Coherent Interface (SCI)-Head purging other entries.



Directory Based Protocol

Scalable Distributed Directory (SDD)

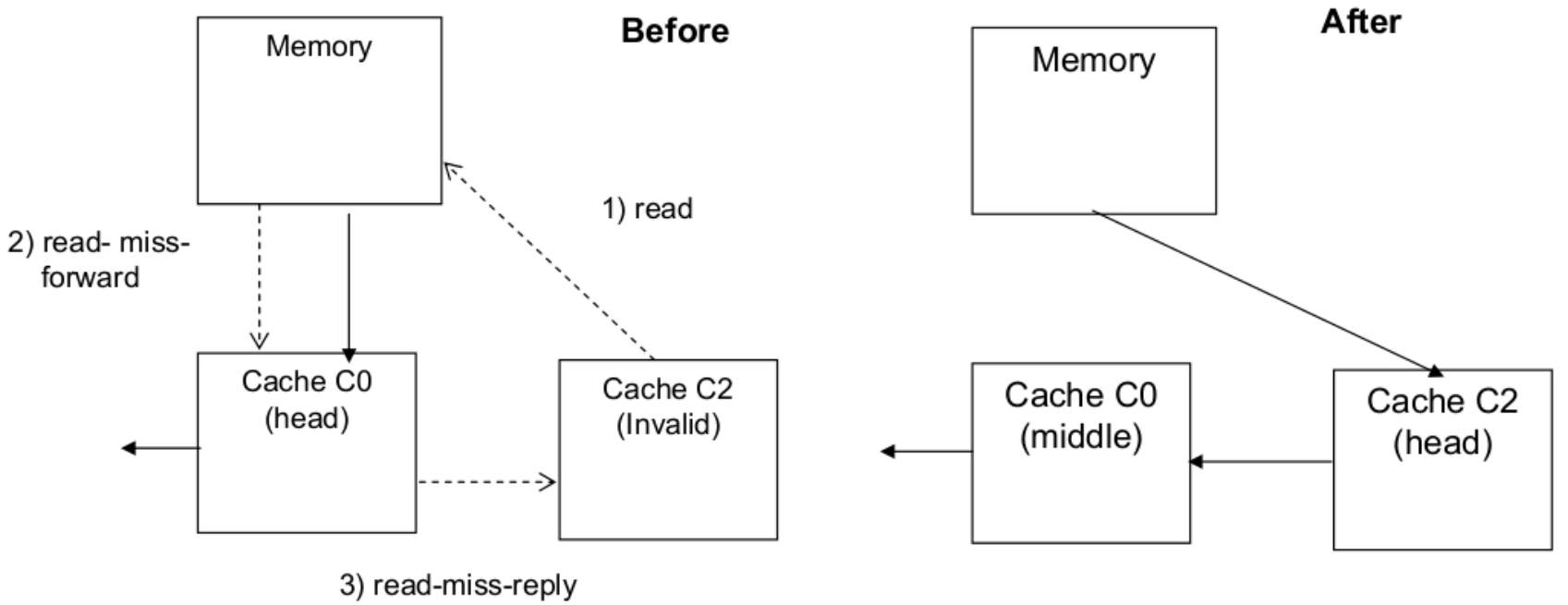
- A singly linked list of distributed directories.
- Similar to the SCI protocol, memory points to the head of the sharing list.
- Each processor points only to its predecessor.
- The sharing list additions and removals are handled different from the SCI protocol .

Scalable Distributed Directory (SDD)

- On a read miss, a new requester sends a read-miss message to memory.
- The memory updates its head pointers to point to the requester and send a read-miss-forward signal to the old head.
- On receiving the request, the old head returns the requested data along with its address as a read-miss-reply.
- When the reply is received, at the requester's cache, the data is copied and the pointer is made to point to the old head.

Directory Based Protocol

- Scalable Distributed Directory (SDD)-List addition



Directory Based Protocol

Scalable Distributed Directory (SDD)

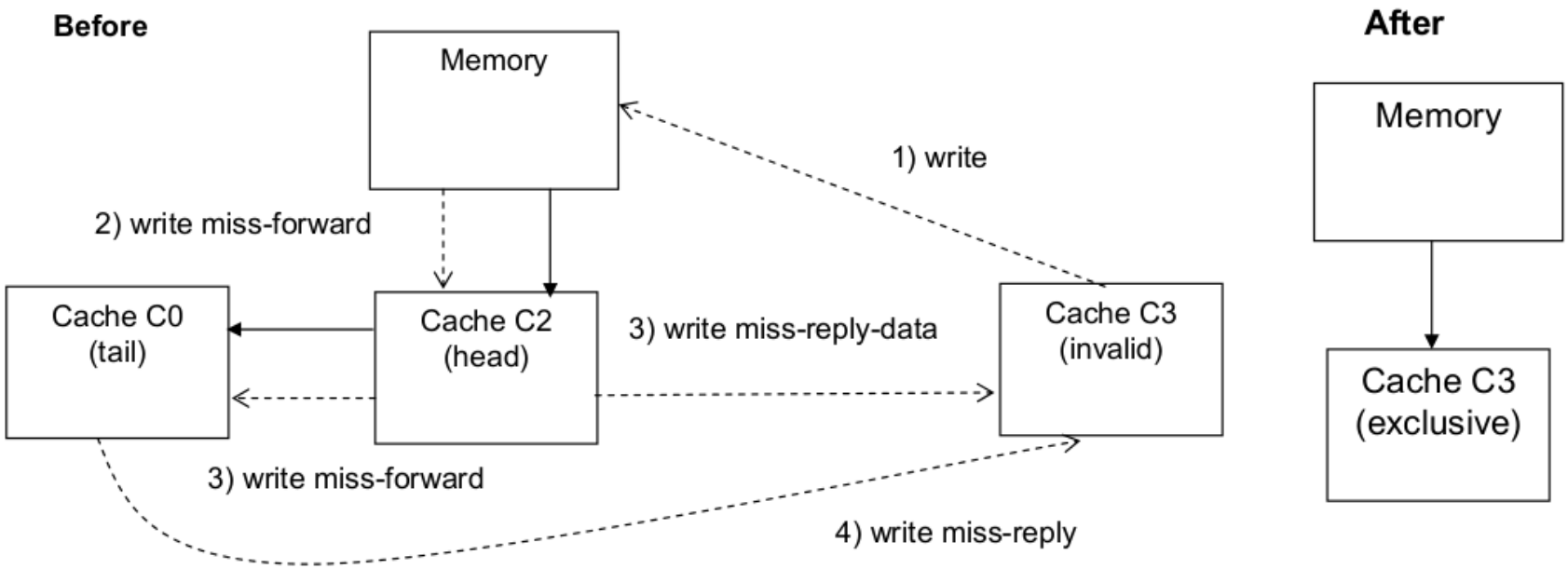
- On a write miss, a requester sends a write-miss message to memory.
- The memory updates its head pointers to point to the requester and sends a write-miss-forward signal to the old head.
- The old head invalidates itself, returns the requested data as a write-miss-reply-data signal, and send a write-miss-forward to the next cache in the list.

Scalable Distributed Directory (SDD)

- When the next cache receives the write-miss-forward signal, it invalidates itself and sends a write-miss-forward to the next cache in the list.
- When the write-miss-forward signal is received by the tail or by a cache that no longer has copy of the block, a write-miss-reply is sent to the requester.
- The write is complete when the requester receives both write-miss-reply-data and write-miss-reply.

Directory Based Protocol

- Scalable Distributed Directory (SDD)- Write Miss List Removal



Presentation Outline

1. Module I: Parallel Computer Models
   - Evolution of Computer Architecture
   - System Attributes to Performance
   - Multiprocessors and Multicomputers
   - Multivector and SIMD
2. Module II: Processors and Memory Hierarchy
3. Module III: Multiprocessors and Multicomputers
4. Module VI: Multithreaded and Data Flow Architecture

Contents

6.1 Latency Hiding Techniques

- For improving performance, latency - reducing, tolerating or hiding - techniques used
- Four Latency hiding techniques usually deployed
  - Using prefetching techniques - bringing instructions closer to microprocessor
  - Using Coherent caches - with h/w support to reduce cache misses
  - Using relaxed memory consistency models - buffering and pipelining of memory references
  - using multiple-contexts - allowing processor to switch from one context to another whenever long latency operation is encountered.

6.1 Latency Hiding Techniques

- a. Shared Virtual Memory
- b. Prefetching Techniques
- c. Distributed Coherent Caches
- d. Scalable Coherence Interface
- e. Relaxed Memory Consistency