High Performance Computing BE 2019 Pattern

Prof V B More

Course Objectives & Outcomes

To understand different parallel programming models

CO1:

Understand various Parallel Paradigm

U1: Introduction to Parallel Computing

Syllabus:

Introduction to Parallel Computing: Motivating Parallelism

Modern Processor: Stored-program computer architecture, General-purpose Cache-based Microprocessor architecture **Parallel Programming Platforms**: Implicit Parallelism, Dichotomy of Parallel Computing Platforms, Physical Organization of Parallel Platforms, Communication Costs in Parallel Machines. Levels of parallelism,

Models: SIMD, MIMD, SIMT, SPMD, Data Flow Models, Demand-driven Computation,

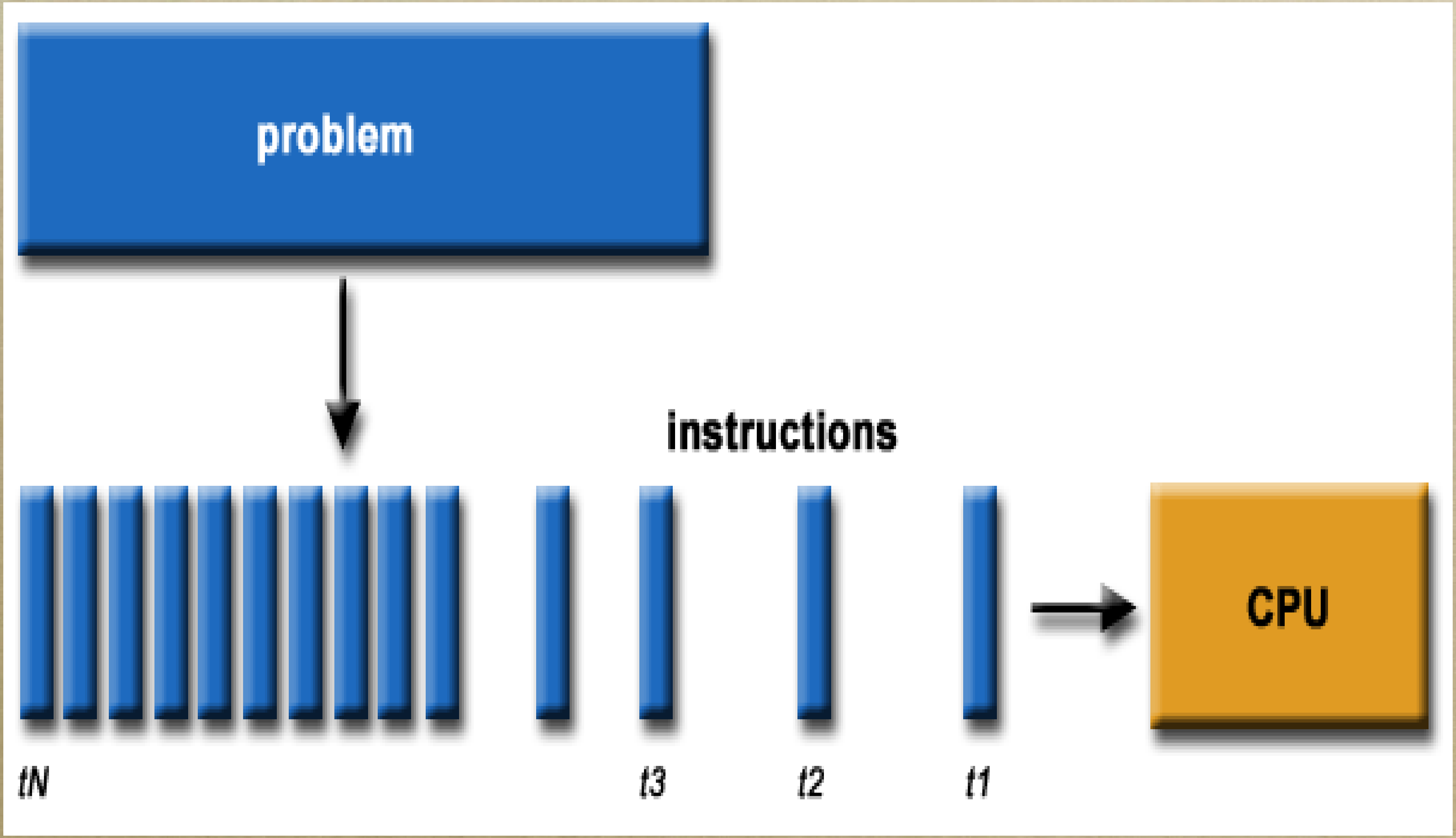
Architectures: N-wide superscalar architectures, multi-core, multi-threaded.

Introduction to Parallel Computing

Traditionally, software has been written for ***serial*** computation:

- To be run on a single computer having a single Central Processing Unit (CPU);
- A problem is broken into a discrete series of instructions.
- Instructions are executed one after another.
- Only one instruction may execute at any moment in time.

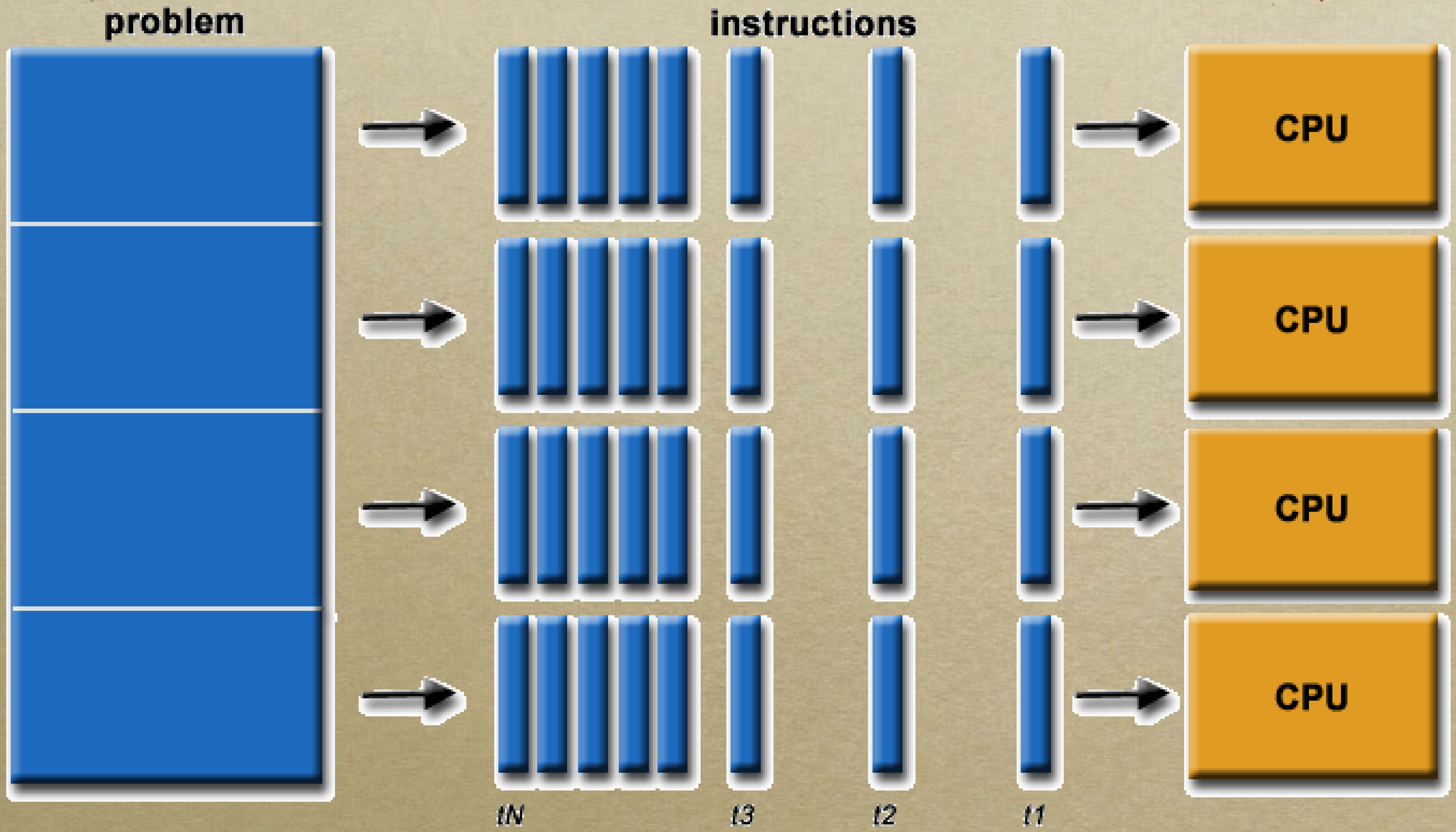
Serial computation:



Parallel Computing

- In the simplest sense, parallel computing is the simultaneous use of multiple compute resources to solve a computational problem.
- To be run using multiple CPUs
- A problem is broken into discrete parts that can be solved concurrently
- Each part is further broken down to a series of instructions
- Instructions from each part execute simultaneously on different CPUs

CONT..



Motivating Parallelism

- Development of parallel software has traditionally been thought of as time and effort intensive.
- This can be largely attributed to the inherent complexity of specifying and coordinating concurrent tasks, a lack of portable algorithms, standardized environments, and software development toolkits.

1. **The Computational Power Argument – from Transistors to FLOPS**
2. **The Memory/Disk Speed Argument**
3. **The Data Communication Argument**

The Computational Power Argument – from Transistors to FLOPS ...

- **In 1965, Gordon Moore made the following simple observation:**

"The complexity for minimum component costs has increased at a rate of roughly a factor of two per year. Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least 10 years. That means by 1975, the number of components per integrated circuit for minimum cost will be 65,000."

The Memory/Disk Speed Argument

The overall speed of computation is determined not just by the speed of the processor, but also by the ability of the memory system to feed data to it. While clock rates of high-end processors have increased at roughly 40% per year over the past decade, DRAM access times have only improved at the rate of roughly 10% per year over this interval.

- The overall performance of the memory system is determined by the fraction of the total memory requests that can be satisfied from the cache

The Data Communication Argument

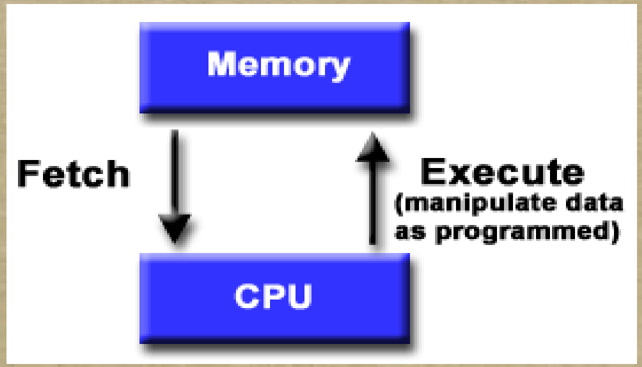
- In many applications there are constraints on the location of data and/or resources across the Internet. An example of such an application is mining of large commercial datasets distributed over a relatively low bandwidth network. In such applications, even if the computing power is available to accomplish the required task without resorting to parallel computing, it is infeasible to collect the data at a central location. In these cases, the motivation for parallelism comes not just from the need for computing resources but also from the infeasibility or undesirability of alternate (centralized) approaches.

Reference Book: Ananth

Modern Processor:

1. Stored-program computer architecture : Its defining property, which set it apart from earlier designs, is that its instructions are numbers that are stored as data in memory. Instructions are read and executed by a control unit; a separate arithmetic/logic unit is responsible for the actual computations and manipulates data stored in memory along with the instructions

A von Neumann computer uses the stored-program concept. The CPU executes a stored program that specifies a sequence of read and write operations on the memory.



Cont..

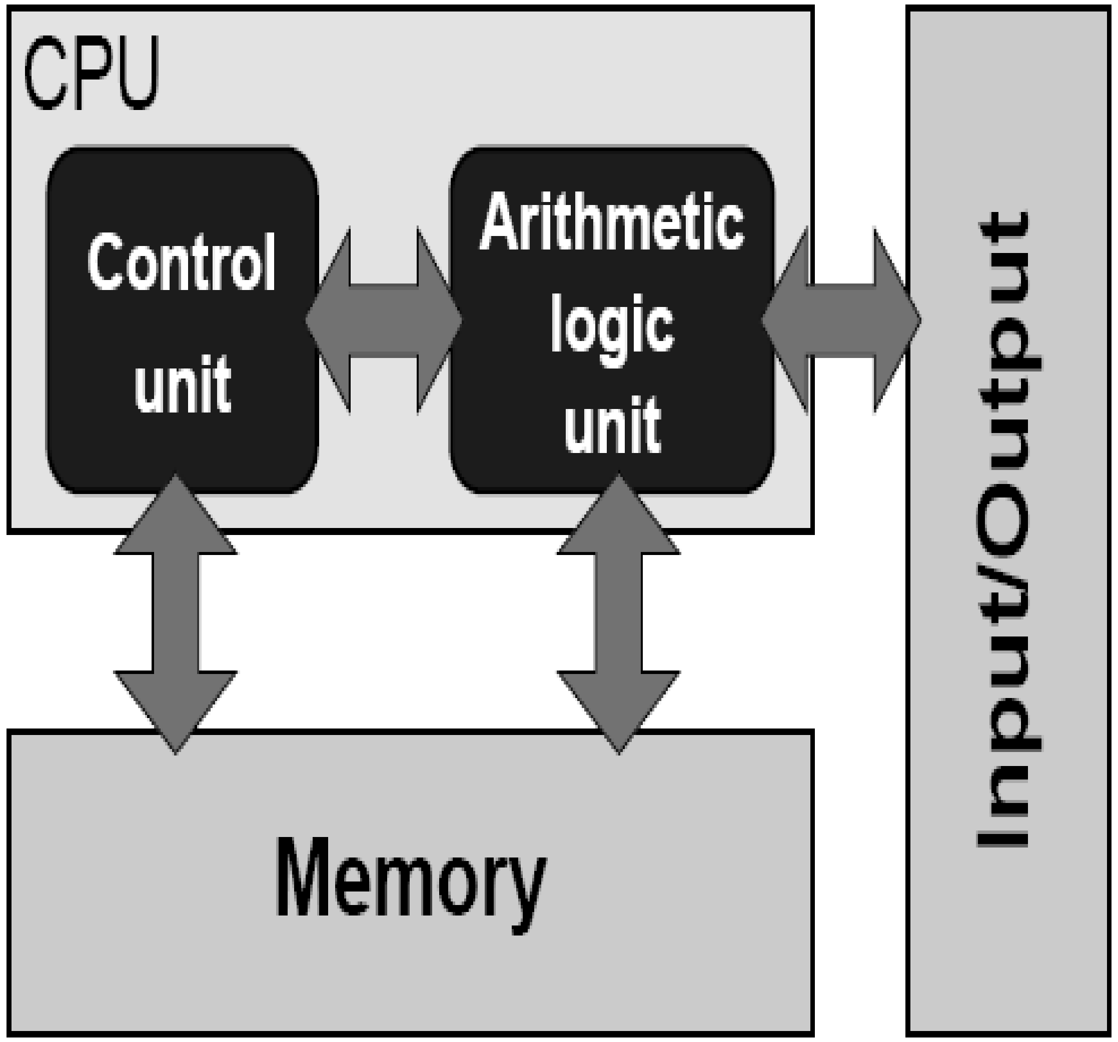


Figure 1.1: Stored-program computer architectural concept. The "program," which feeds the control unit, is stored in memory together with any data the arithmetic unit requires.

Cont..

Instructions and data must be continuously fed to the control and arithmetic units, so that the speed of the memory interface poses a limitation on compute performance.

The architecture is inherently sequential, processing a single instruction with (possibly) a single operand or a group of operands from memory.(SISD)

Cont..

2. General-purpose Cache-based Microprocessor architecture :

- Microprocessors implement stored pgm....
- Modern processors have lot of componets but only a small

part does the actual work -AU for fp and int operations.

Rest are CPU regs, nowdays processors req all operands to

reside in regs.

- LD(load) and ST(store) units handle instruction transfer.
- Queues for instructions
- Finally Cache

Cont...

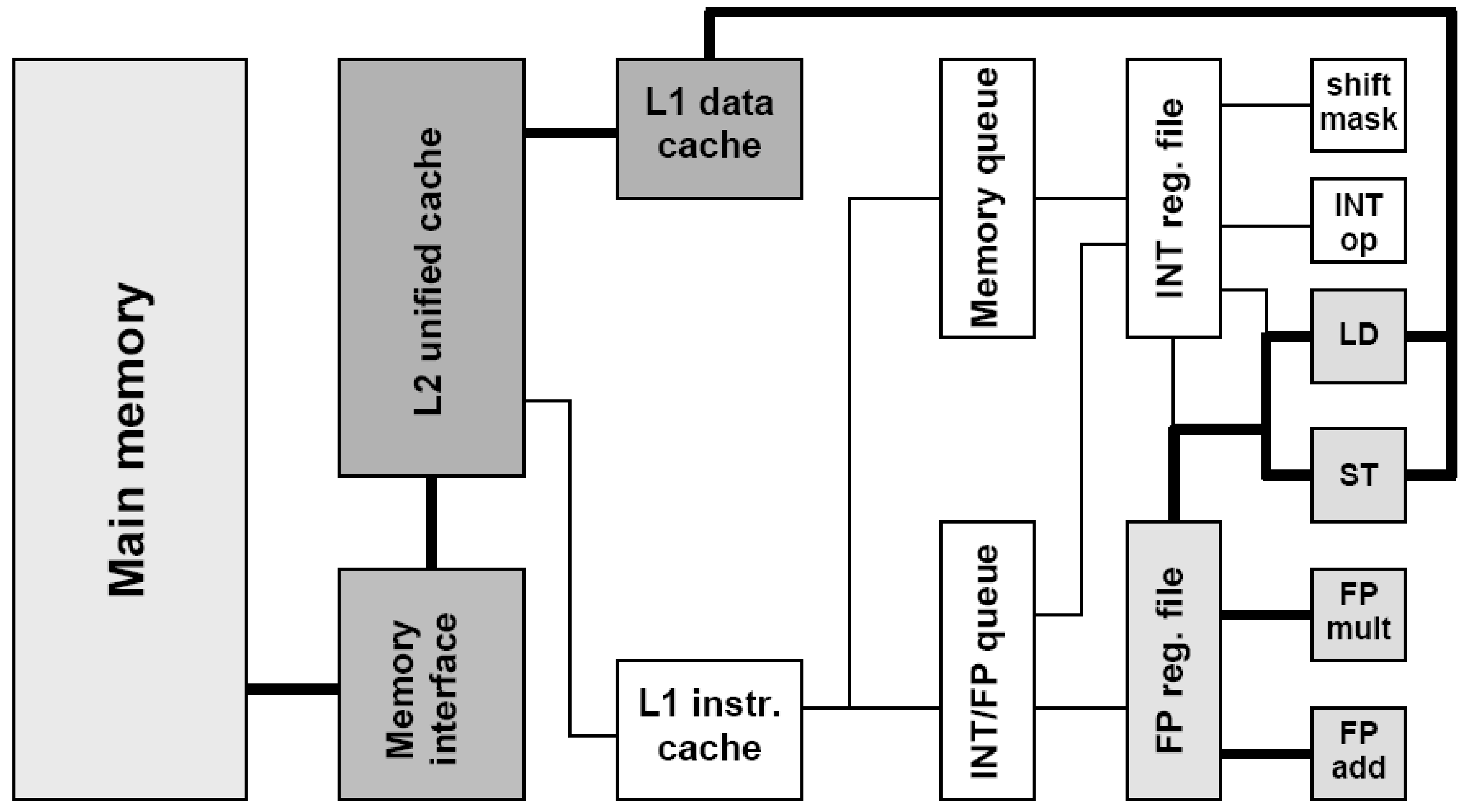


Figure 1.2: Simplified block diagram of a typical cache-based microprocessor (one core). Other cores on the same chip or package (socket) can share resources like caches or the memory interface. The functional blocks and data paths most relevant to performance issues in scientific computing are highlighted.

Parallel Programming Platforms

A] Implicit Parallelism

Trends in Microprocessor Architecture:

1.Pipelining and Superscalar Execution

Pipelining: several instructions are simultaneously at different

stages of their execution

- Superscalar: several instructions are simultaneously at the same stages of their execution

2. Very Long Instruction Word Processors [VLIW]

Pipelining

Pipelining overlaps various stages of instruction execution to achieve performance. Pipelining, however, has several limitations.

- The speed of a pipeline is eventually limited by the slowest stage. For this reason, conventional processors rely on very deep pipelines (20 stage pipelines in state-of-the-art Pentium processors).
- However, in typical program traces, every 5-6th instruction is a conditional jump! This requires very accurate branch prediction.
- •The penalty of a mis-prediction grows with the depth of the pipeline, since a larger number of instructions will have to be flushed.

Superscalar

One simple way of alleviating these bottlenecks is to use multiple pipelines.

The question then becomes one of selecting these instructions

In Below example, there is some wastage of resources due to data dependencies.

Superscalar Example

```
1. load R1, @1000
2. load R2, @1008
3. add R1, @1004
4. add R2, @100C
5. add R1, R2
6. store R1, @2000
```

(i)

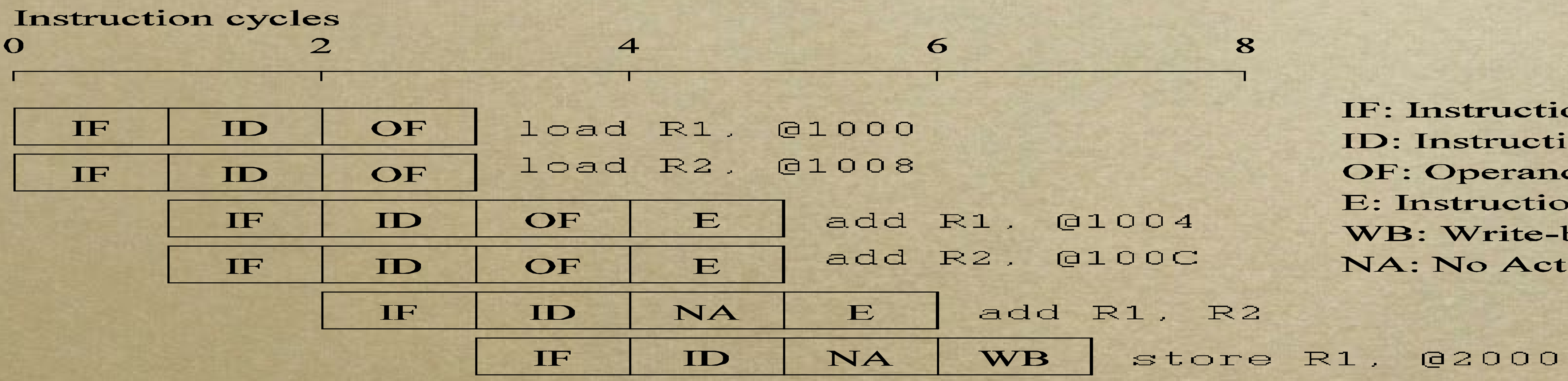
```
1. load R1, @1000
2. add R1, @1004
3. add R1, @1008
4. add R1, @100C
5. store R1, @2000
```

(ii)

```
1. load R1, @1000
2. add R1, @1004
3. load R2, @1008
4. add R2, @100C
5. add R1, R2
6. store R1, @2000
```

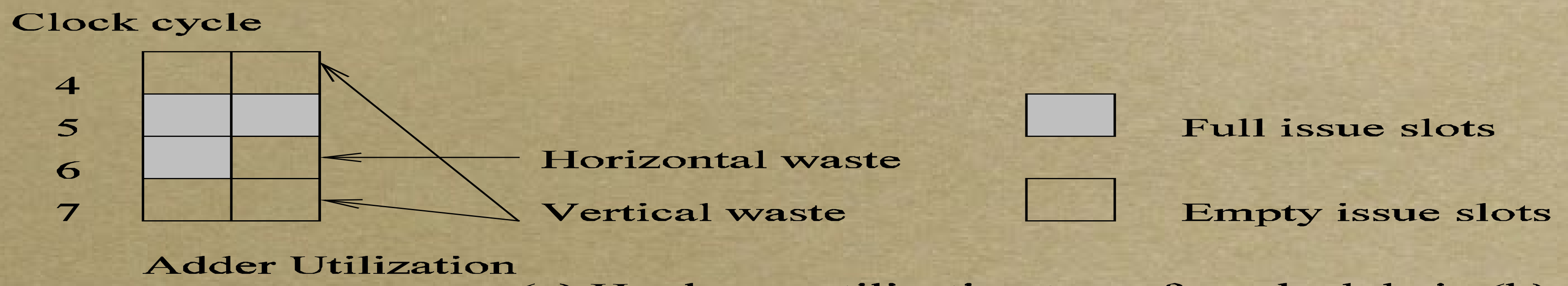
(iii)

(a) Three different code fragments for adding a list of four numbers.



IF: Instruction Fetch
ID: Instruction Decode
OF: Operand Fetch
E: Instruction Execute
WB: Write-back
NA: No Action

(b) Execution schedule for code fragment (i) above.



(c) Hardware utilization trace for schedule in (b).

Very Long Instruction Word

The hardware cost and complexity of the superscalar scheduler is a major consideration in processor design.

- To address this issues, VLIW processors rely on compile time analysis to identify and bundle together instructions that can be executed concurrently.
- These instructions are packed and dispatched together, and thus the name very long instruction word.

VLIW & Superscalar

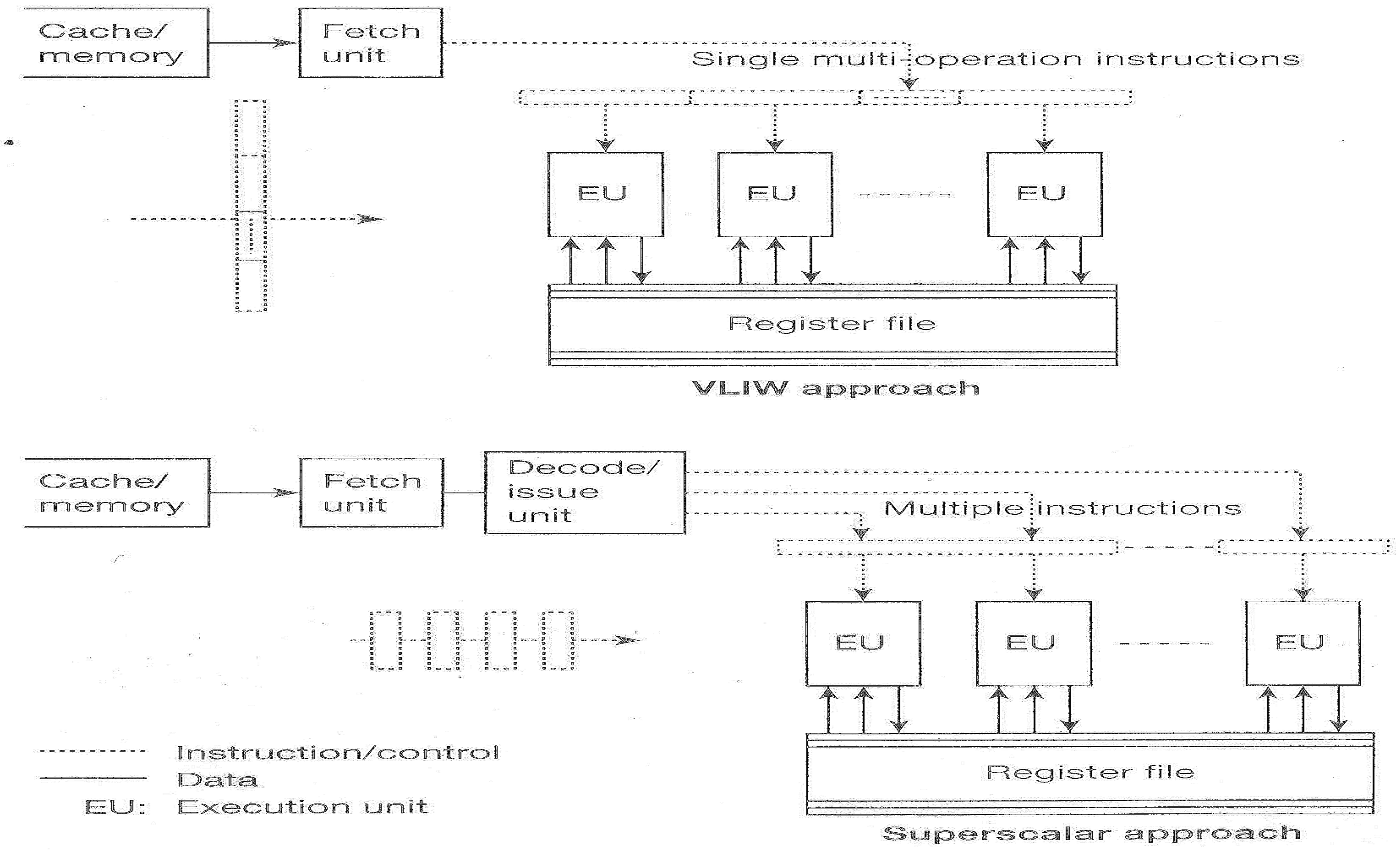


Figure 6.2 Main differences between superscalar processors and VLIW architectures.

Cont..

B]Dichotomy of Parallel Computing Platforms:

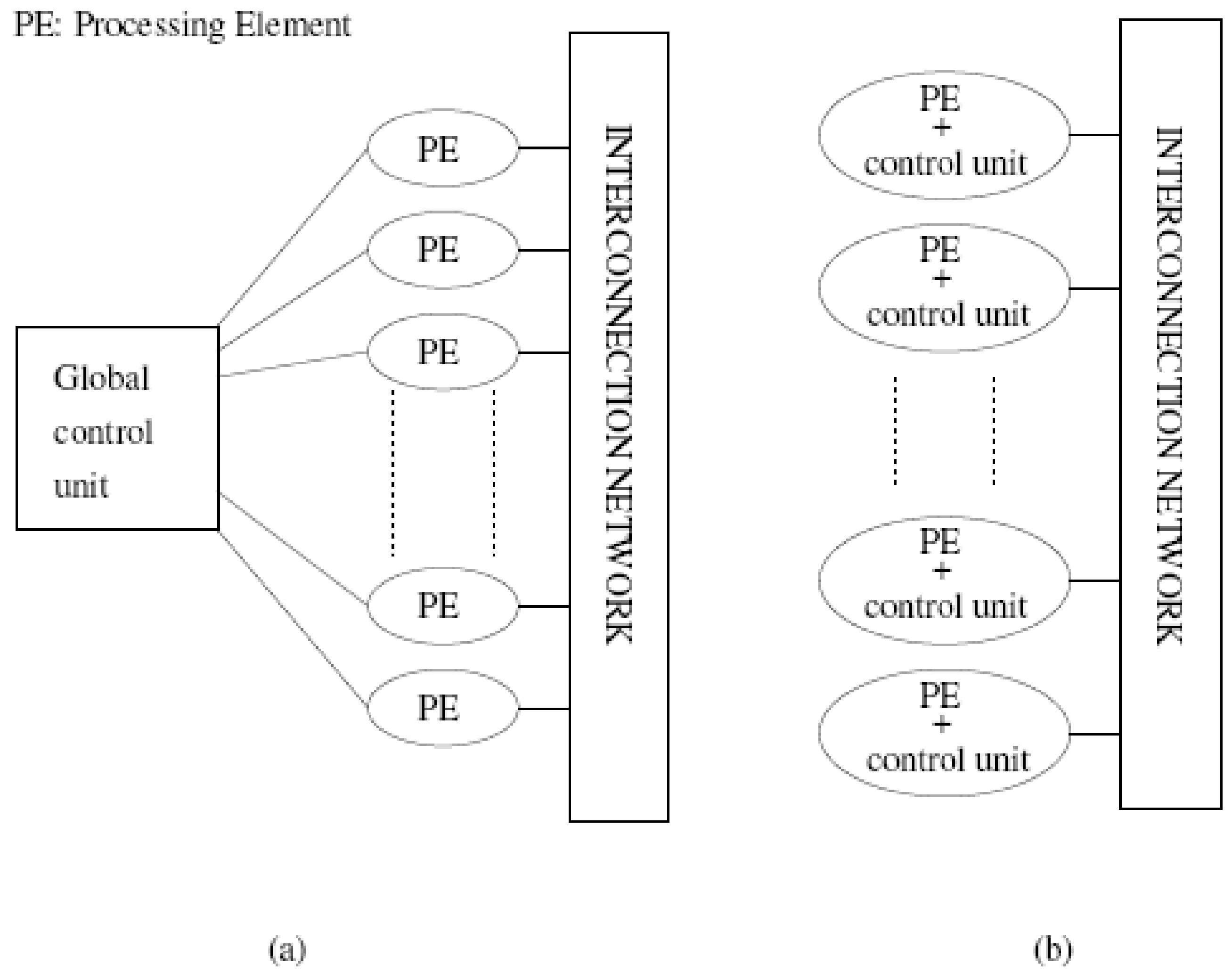
1. Control Structure of Parallel Platforms (Ex: Parallelism from single instruction on multiple processors)

- Parallelism can be expressed at various levels of granularity – from instruction level to processes.
- Between these extremes exist a range of models, along with corresponding architectural support.

Cont..

- Processing units in parallel computers either operate under the centralized control of a single control unit or work independently.
- If there is a single control unit that dispatches the same instruction to various processors (that work on different data), the model is referred to as single instruction stream, multiple data stream (SIMD).
- If each processor has its own control control unit, each processor can execute different instructions on different data items. This model is called multiple instruction stream, multiple data stream (MIMD).

Cont...



A typical SIMD architecture (a) and a typical MIMD architecture (b).

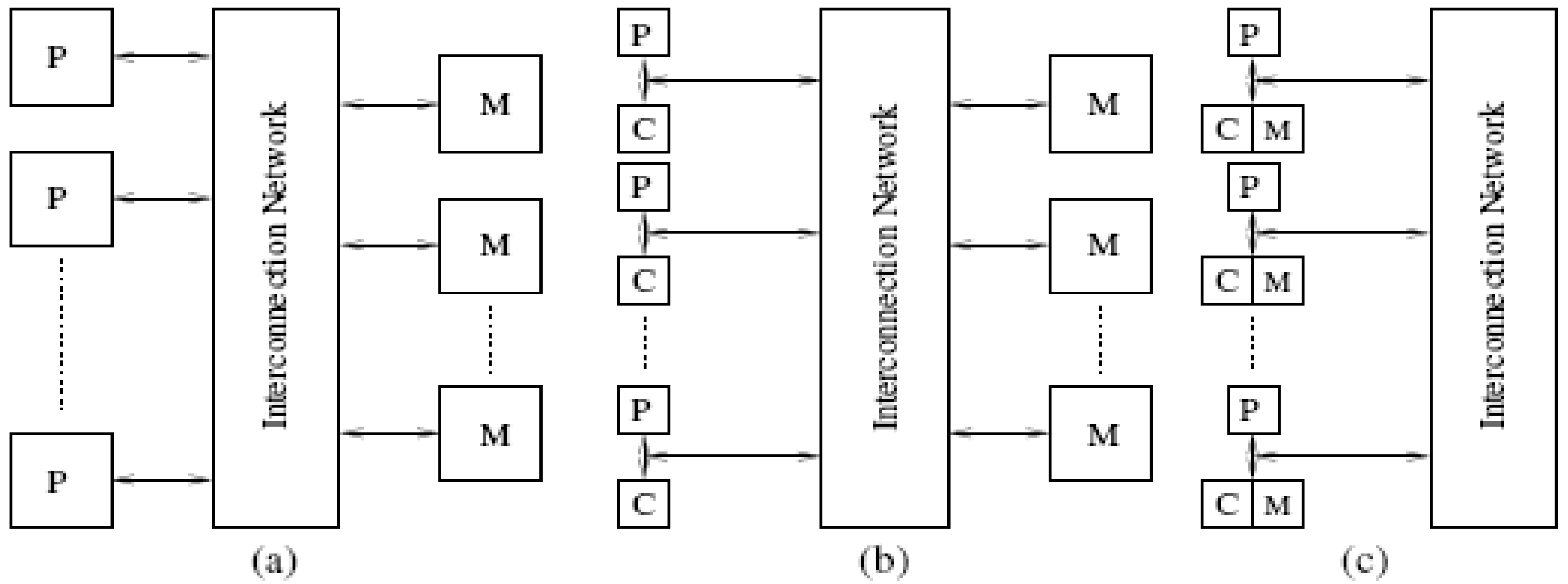
Cont..

2. Communication Model of Parallel Platforms (Shared address space platforms: UMA & NUMA , message passing Platform)

- Part (or all) of the memory is accessible to all processors.
- Processors interact by modifying data objects stored in this shared-address-space.
- If the time taken by a processor to access any memory word in the system global or local is identical, the platform is classied as a uniform memory access (UMA), else, a non_uniform memory access (NUMA) machine.

UMA & NUMA

NUMA and UMA Shared-Address-Space Platforms



Typical shared-address-space architectures: (a) Uniform-memory-access shared-address-space computer; (b) Uniform-memory-access shared-address-space computer with caches and memories; (c) Non-uniform-memory-access shared-address-space computer with local memory only.

Cont..

C]Physical Organization of Parallel Platforms :

1. Architecture of an Ideal Parallel Computer

Exclusive-read, exclusive-write (EREW) PRAM

Concurrent-read, exclusive-write (CREW) PRAM.

Exclusive-read, concurrent-write (ERCW) PRAM

Concurrent-read, concurrent-write (CRCW) PRAM

Cont..

2. **Interconnection Networks for Parallel Computers**

Classification of interconnection networks: (a) a static

- **network; and (b) a dynamic network.**

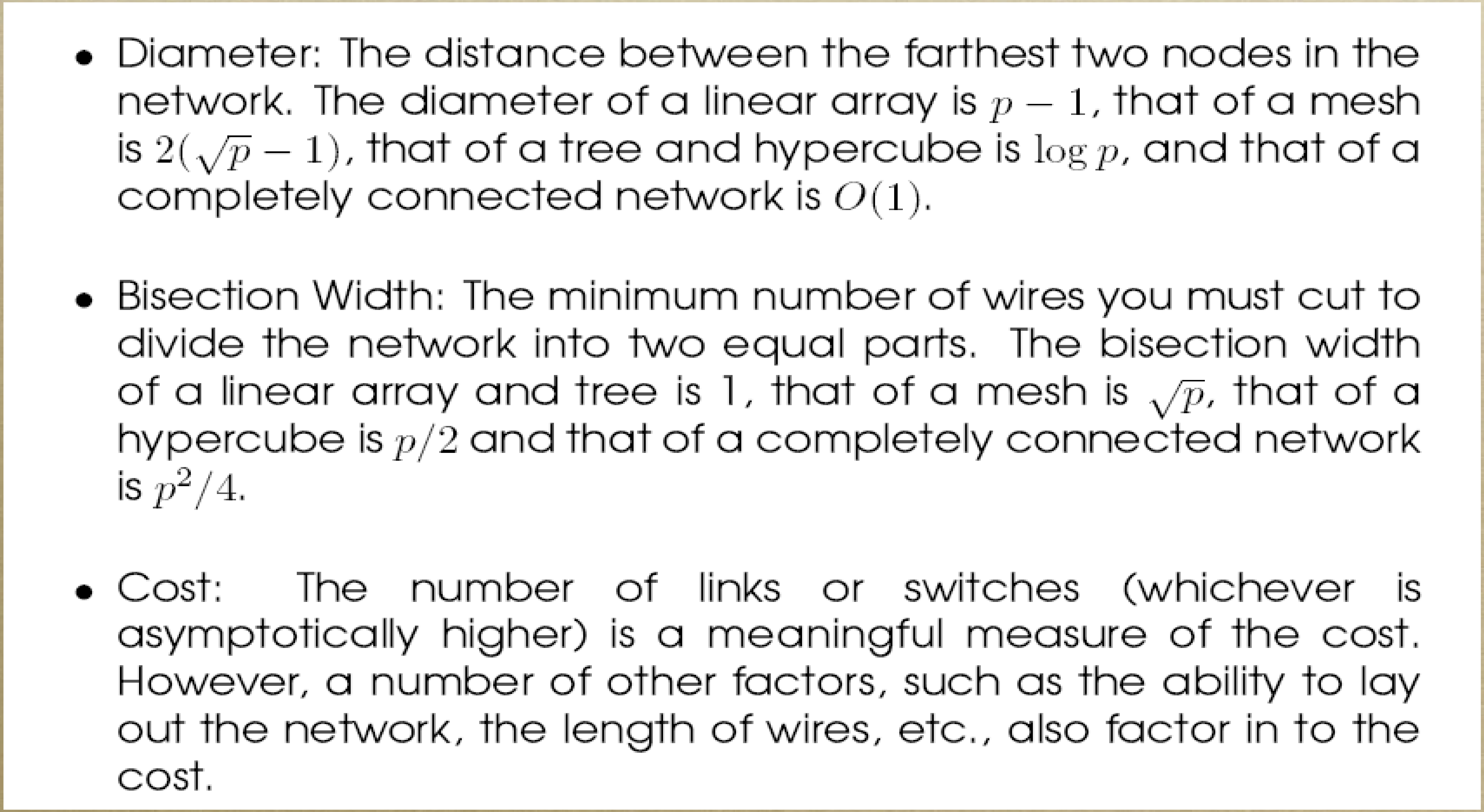
3. Network Topologies: Bus-Based Networks, Crossbar Networks, Multistage Networks, Star-Connected Network, Linear Arrays, Meshes, and *k-d Meshes etc*

4. Evaluating Static Interconnection Networks : Diameter, connection and Bisection Width

5. Evaluating Dynamic Interconnection Networks

6. Cache Coherence in Multiprocessor Systems: Snoopy cache based and Directory based

Evaluating Static Interconnection Networks : Diameter, connection and Bisection Width

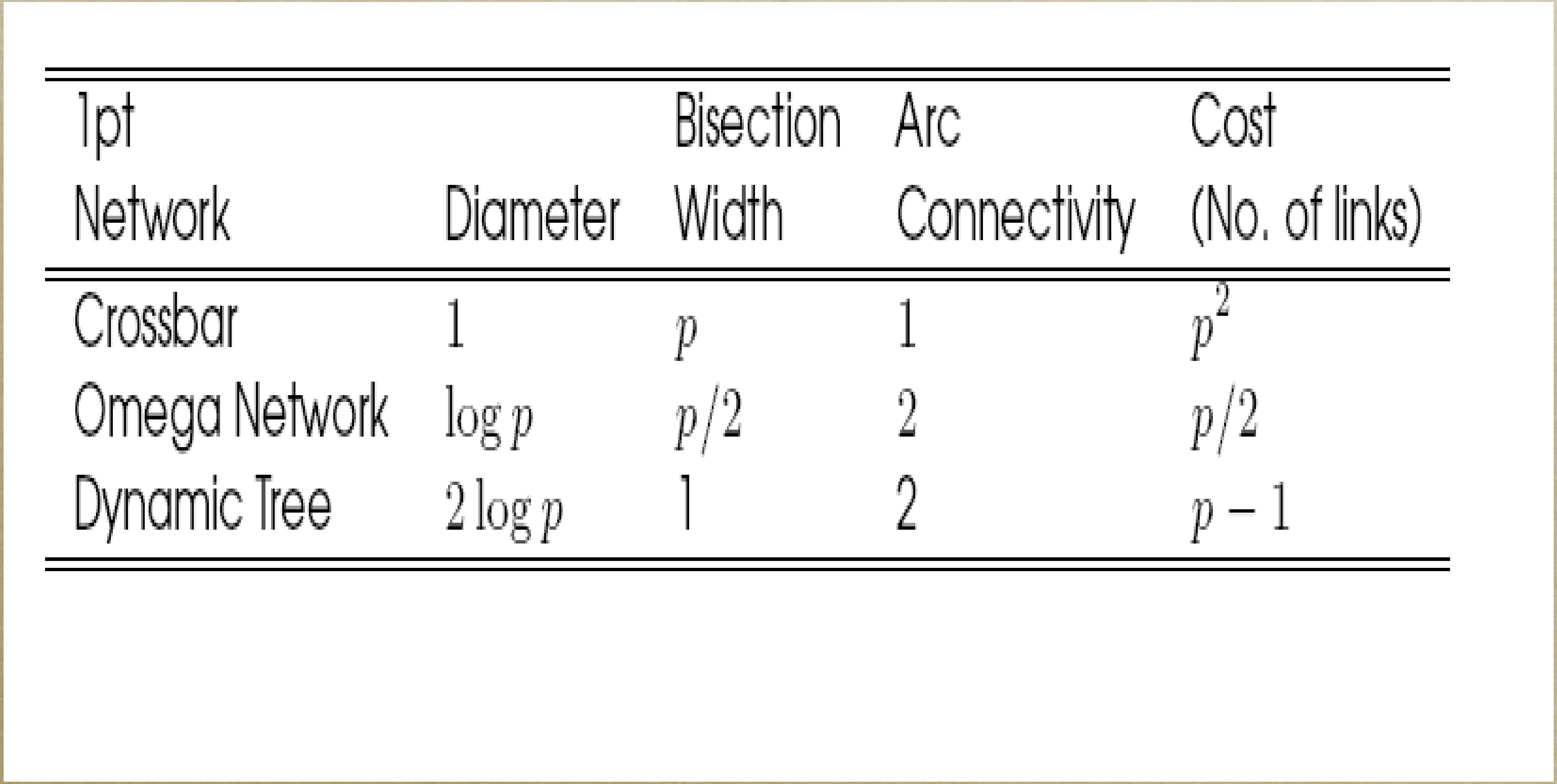


- Diameter: The distance between the farthest two nodes in the network. The diameter of a linear array is $p-1$, that of a mesh is $2(\sqrt{p}-1)$, that of a tree and hypercube is $\log p$, and that of a completely connected network is $O(1)$.

- Bisection Width: The minimum number of wires you must cut to divide the network into two equal parts. The bisection width of a linear array and tree is 1, that of a mesh is $\sqrt{p}$, that of a hypercube is $p/2$ and that of a completely connected network is $p^2/4$.

- Cost: The number of links or switches (whichever is asymptotically higher) is a meaningful measure of the cost. However, a number of other factors, such as the ability to lay out the network, the length of wires, etc., also factor in to the cost.

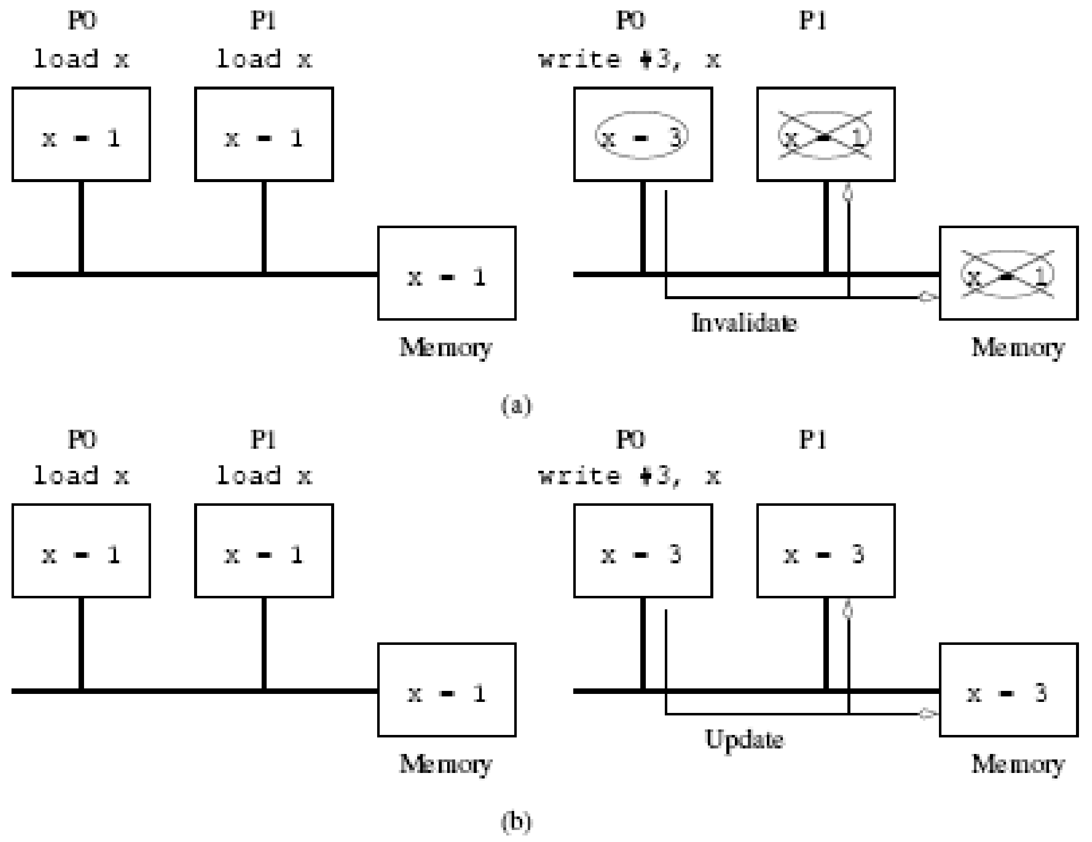
Evaluating Dynamic Interconnection Networks



1pt		Bisection	Arc	Cost
Network	Diameter	Width	Connectivity	(No. of links)
Crossbar	1	p	1	p^2
Omega Network	$\log p$	$p/2$	2	$p/2$
Dynamic Tree	$2 \log p$	1	2	$p - 1$

Cache Coherence in multiprocessor

When the value of a variable is changes, all its copies must either be invalidated or updated.



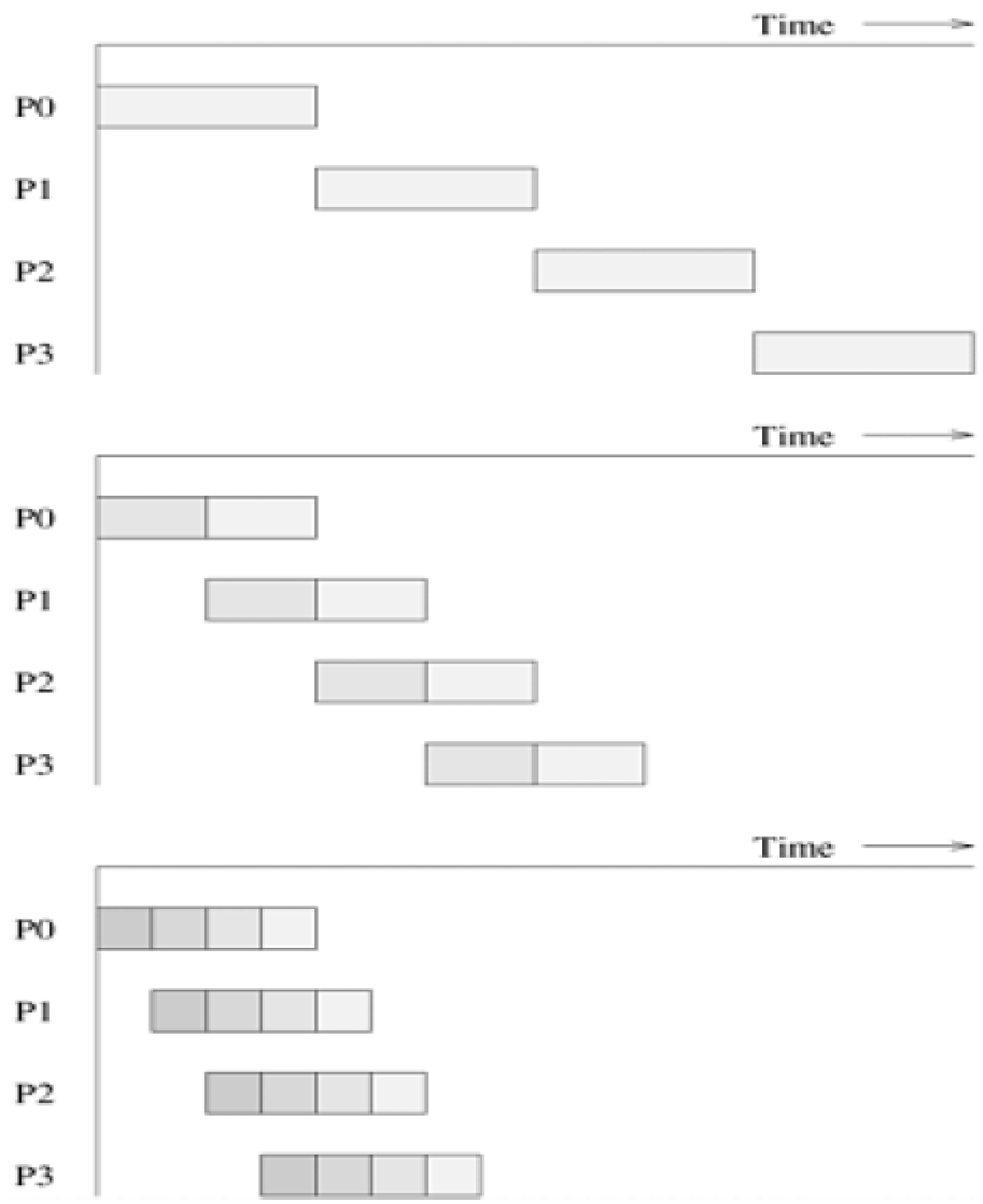
Cache coherence in multiprocessor systems: (a) Invalidate protocol; (b) Update protocol for shared variables.

Communication Costs in Parallel Machines

A]**Message Passing Costs in Parallel Computers:**

- ***1.Startup time (ts):*** The startup time is the time required to handle a message at the sending and receiving nodes
- ***2. Per-hop time (th):*** After a message leaves a node, it takes a finite amount of time to reach the next node in its path. The time taken by the header of a message to travel between two directly-connected nodes in the network is called the per-hop time. It is also known as ***node latency.***
- ***3. Per-word transfer time (tw):*** If the channel bandwidth is *r words per second, then each word takes time tw = 1/r to traverse the link*

Cont.. Store-and-Forward Routing, Packet Routing, Cut Through routing



(a) A single message sent over a store-and-forward network

(b) The same message broken into two parts and sent over the network.

(c) The same message broken into four parts and sent over the network.

B]Communication Costs in Shared-Address-Space Machines

- While the basic messaging cost applies to these machines as well, a number of other factors make accurate cost modeling more difficult.
- Memory layout is typically determined by the system.
- Finite cache sizes can result in cache thrashing.
- Overheads associated with invalidate and update operations are difficult to quantify.
- Spatial locality is difficult to model.
- Prefetching can play a role in reducing the overhead associated with data access.
- False sharing and contention are difficult to model.

Levels of parallelism

1. Data Parallelism: Many problems in scientific computing involve processing of large quantities of data stored on a computer. If this manipulation can be performed in parallel, i.e., by multiple processors working on different parts of the data, we speak of data parallelism. As a matter of fact, this is the dominant parallelization concept in scientific computing on MIMD-type computers. It also goes under the name of SPMD (Single Program Multiple Data), as usually the same code is executed on all processors, with independent instruction pointers.

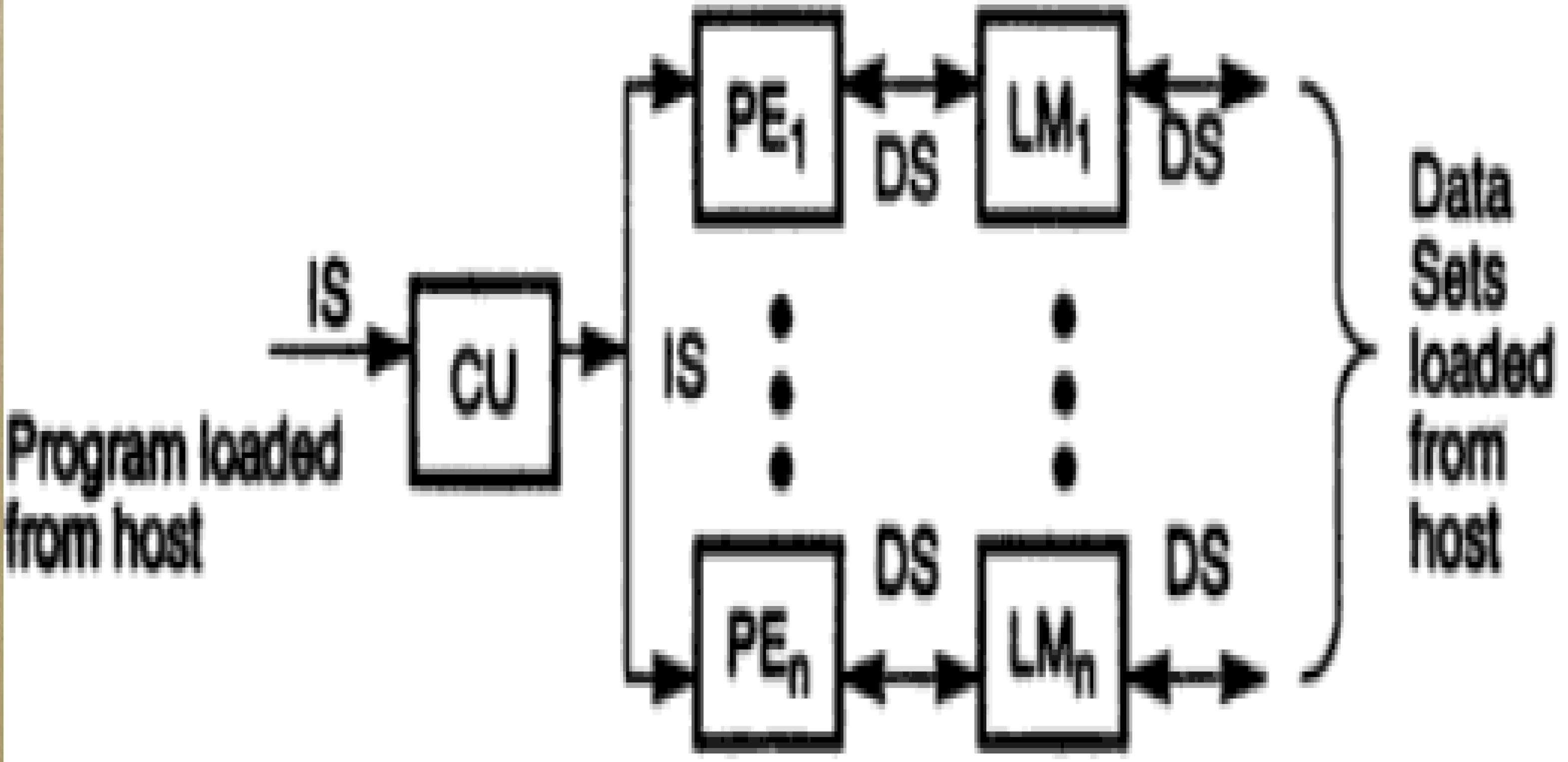
Ex: Medium-grained loop parallelism, Coarse-grained parallelism by domain decomposition

Levels of Parallelism

2. Functional Parallelism: Sometimes the solution of a “big” numerical problem can be split into more or less disparate subtasks, which work together by data exchange and synchronization. In this case, the subtasks execute completely different code on different data items, which is why functional parallelism is also called MPMD (Multiple Program Multiple Data).

Ex: Master Worker Scheme, Functional decomposition

Models : SIMD, MIMD, SIMT, SPMD



(b) SIMD architecture (with distributed memory)

<u>Captions:</u>

CU = Control Unit

PU = Processing Unit

MU = Memory Unit

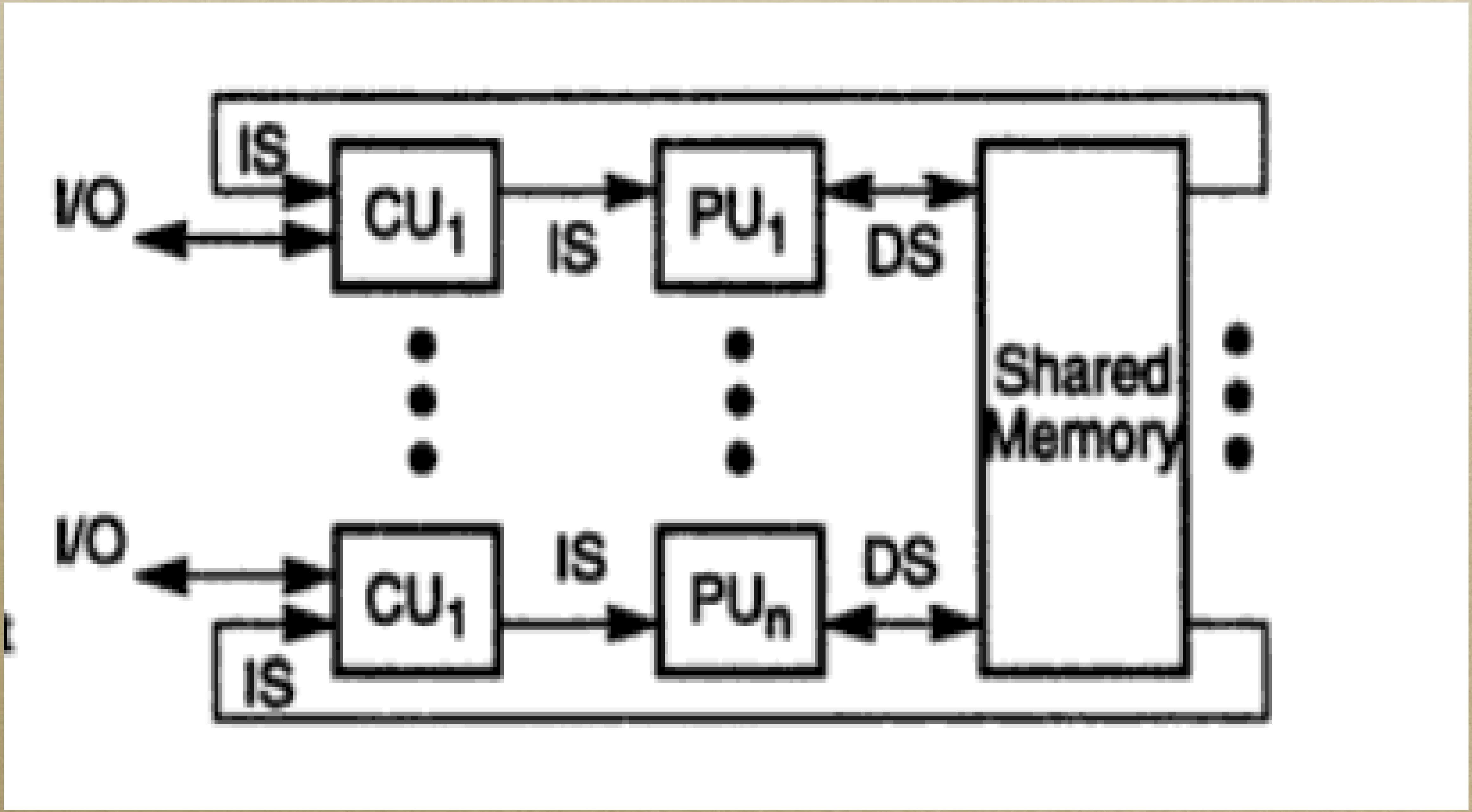
IS = Instruction Stream

DS = Data Stream

PE = Processing Element

LM = Local Memory

MIMD



SIMT

Single instruction, multiple threads (SIMT) is an execution model used in parallel computing where single instruction, multiple data (SIMD) is combined with multithreading. It is different from SPMD in that all instructions in all "threads" are executed in lock-step. The SIMT execution model has been implemented on several GPUs and is relevant for general-purpose computing on graphics processing units (GPGPU),

e.g. some supercomputers combine CPUs with GPUs.

SPMD

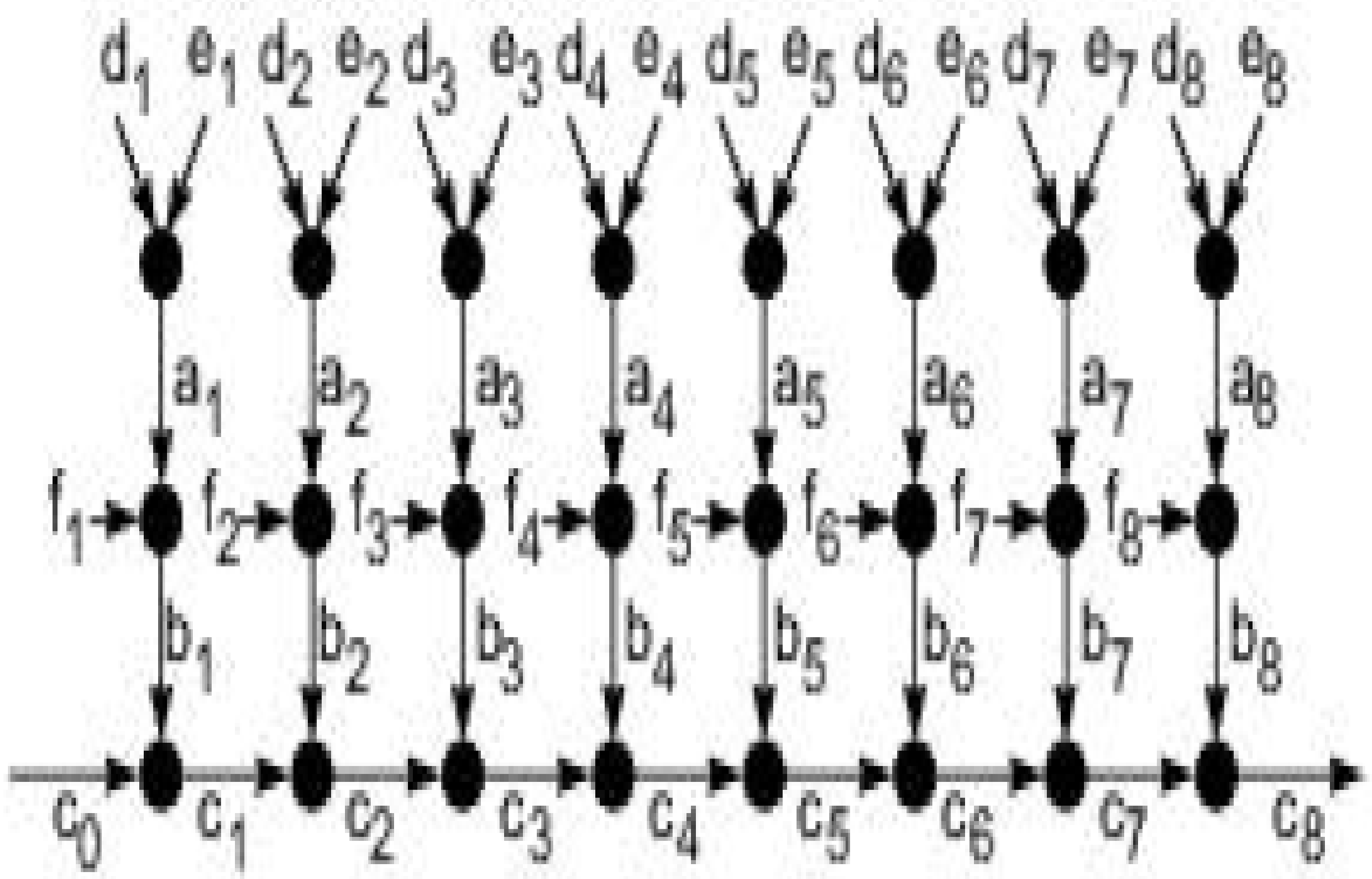
- In contrast to SIMD processors, MIMD processors can execute different programs on different processors.
- A variant of this, called single program multiple data streams (SPMD) executes the same program on different processors.
- It is easy to see that SPMD and MIMD are closely related in terms of programming flexibility and underlying architectural support.
- Examples of such platforms include current generation Sun Ultra Servers, SGI Origin Servers, multiprocessor PCs, workstation clusters, and the IBM SP.

Data Flow

- In a daraflow computer, the execution of an instruction is driven by data availability instead of being guided by a program counter, In theory, any instruction should be ready for execution whenever operands become available.
- The instructions in a data-driven program are not ordered in any way. instead of being stored separately in a main memory, data are directly held inside instructions.
- This data-driven scheme requires no program counter, and no eonlrol sequencer. However, it requires special mechanisms to detect data availability, to match data tokens with needy instructions, and to enable the chain reaction of asynchronous instruction executions. No memory sharing between instructions results in no side effects.

Cont..

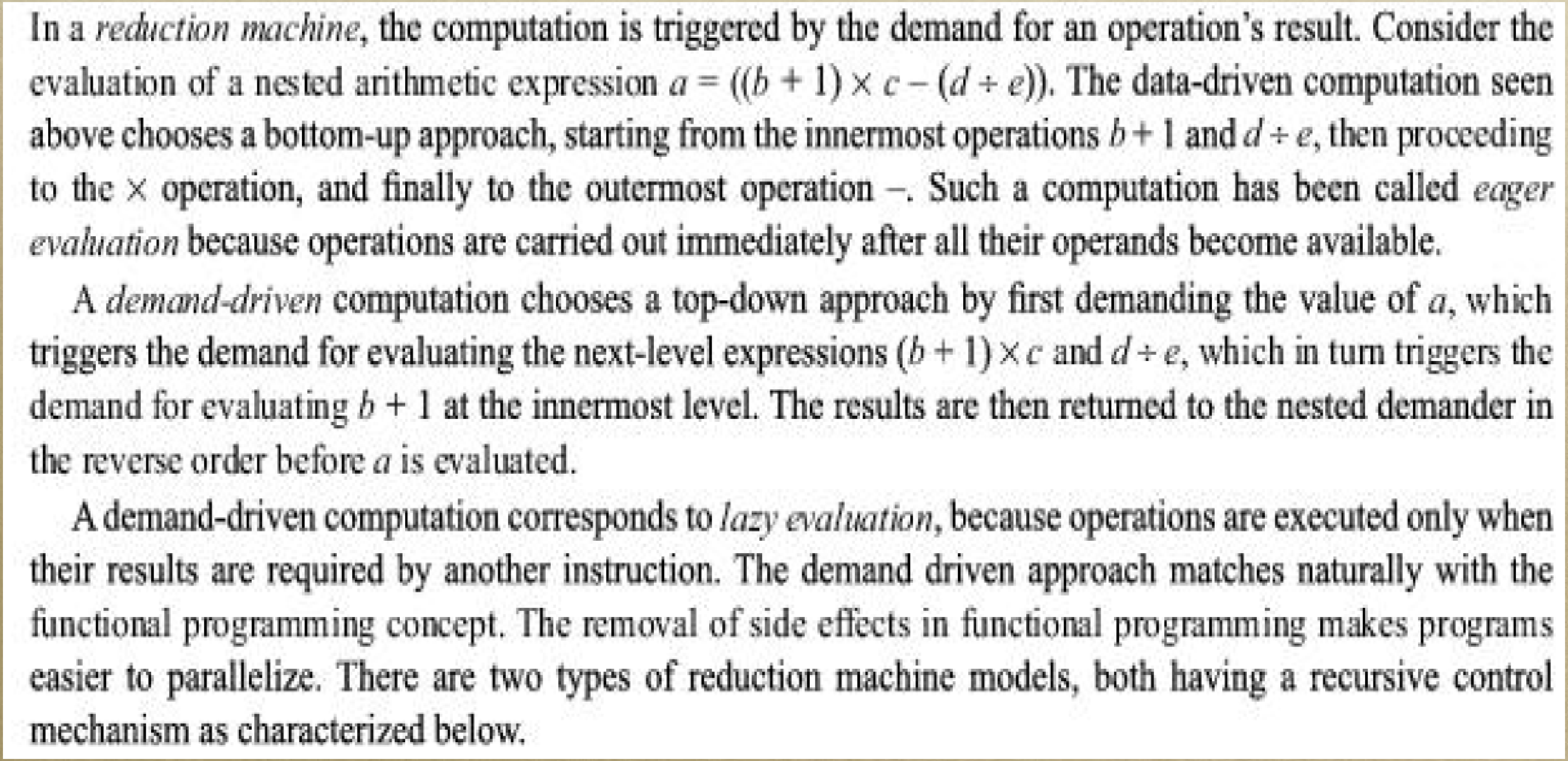
```
input d, e, f
  $c_0 = 0$
for i from 1 to 8 do
  begin
    $a_i := d_i + e_i$
    $b_i := a_i * f_i$
    $c_i := b_i + c_{i-1}$
  end
output a, b, c
```



d_1 e_1 d_2 e_2 d_3 e_3 d_4 e_4 d_5 e_5 d_6 e_6 d_7 e_7 d_8 e_8
a_1 a_2 a_3 a_4 a_5 a_6 a_7 a_8
f_1 f_2 f_3 f_4 f_5 f_6 f_7 f_8
b_1 b_2 b_3 b_4 b_5 b_6 b_7 b_8
c_0 c_1 c_2 c_3 c_4 c_5 c_6 c_7 c_8

(a) A sample program and its dataflow graph

Demand Driven



In a *reduction machine*, the computation is triggered by the demand for an operation's result. Consider the evaluation of a nested arithmetic expression $a = ((b + 1) \times c - (d \div e))$. The data-driven computation seen above chooses a bottom-up approach, starting from the innermost operations $b + 1$ and $d \div e$, then proceeding to the $\times$ operation, and finally to the outermost operation $-$. Such a computation has been called *eager evaluation* because operations are carried out immediately after all their operands become available.

A *demand-driven* computation chooses a top-down approach by first demanding the value of a, which triggers the demand for evaluating the next-level expressions $(b + 1) \times c$ and $d \div e$, which in turn triggers the demand for evaluating $b + 1$ at the innermost level. The results are then returned to the nested demander in the reverse order before a is evaluated.

A demand-driven computation corresponds to *lazy evaluation*, because operations are executed only when their results are required by another instruction. The demand driven approach matches naturally with the functional programming concept. The removal of side effects in functional programming makes programs easier to parallelize. There are two types of reduction machine models, both having a recursive control mechanism as characterized below.

N-Wide Superscalar

- Ideally: in an n-issue superscalar, n instructions are fetched,

decoded, executed, and committed per cycle

In practice:

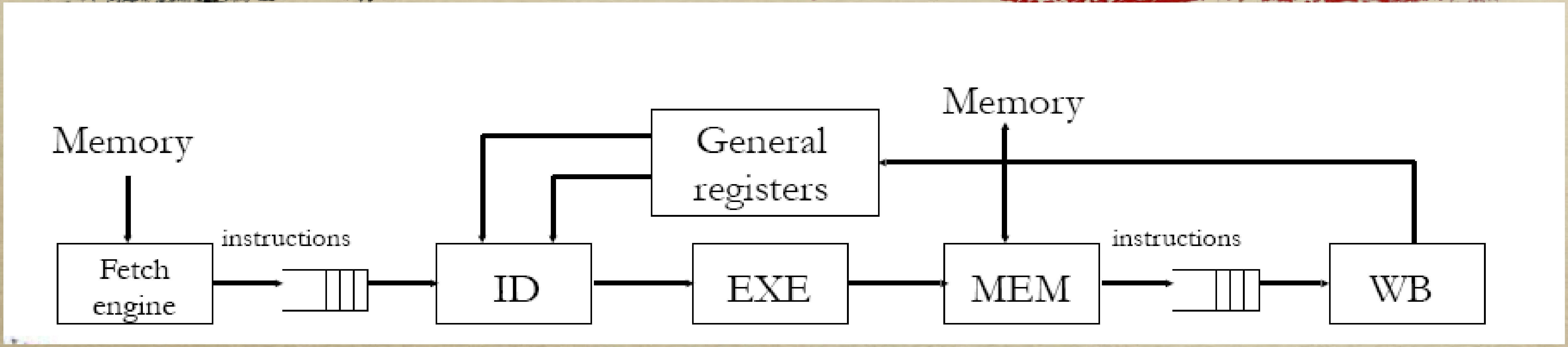
– Data, control, and structural hazards spoil issue flow

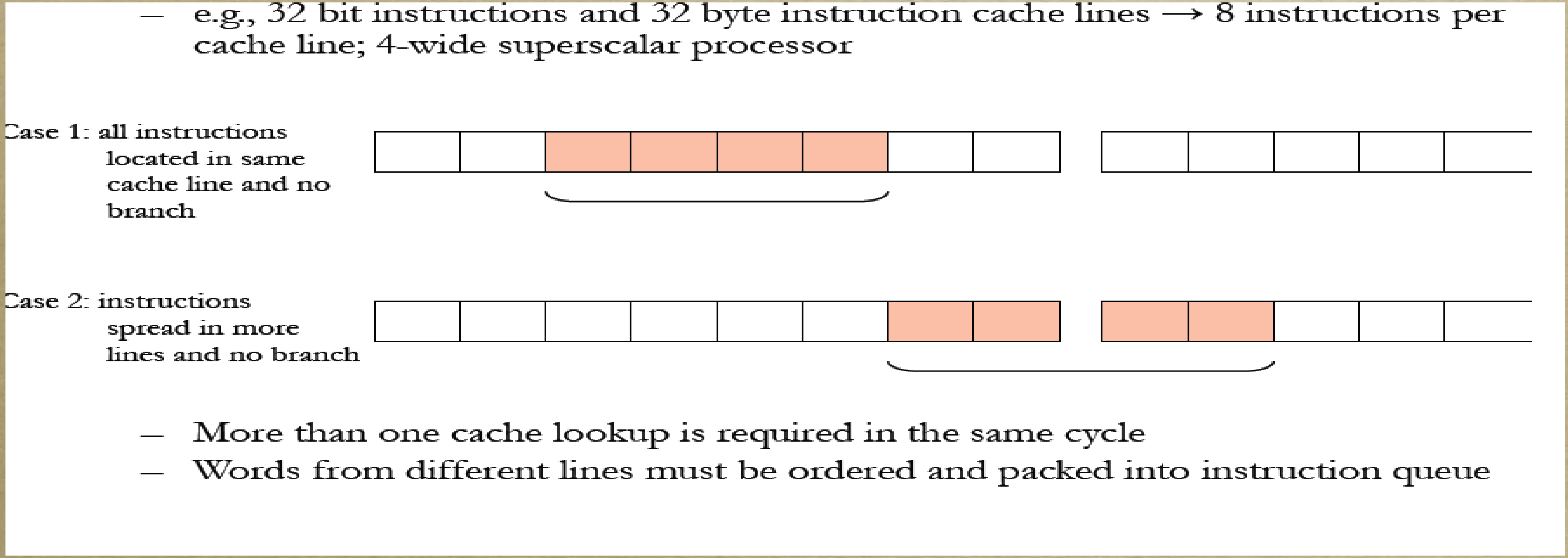
– Multi-cycle instructions spoil commit flow

- Buffers at issue (issue queue) and commit (reorder buffer)

decoupe these stages from the rest of the pipeline and regularize somewhat breaks in the flow

Cont..




- e.g., 32 bit instructions and 32 byte instruction cache lines $\longrightarrow$ 8 instructions per cache line; 4-wide superscalar processor

Case 1: all instructions located in same cache line and no branch
Case 2: instructions spread in more lines and no branch

- More than one cache lookup is required in the same cycle
- Words from different lines must be ordered and packed into instruction queue

Multicore Processor

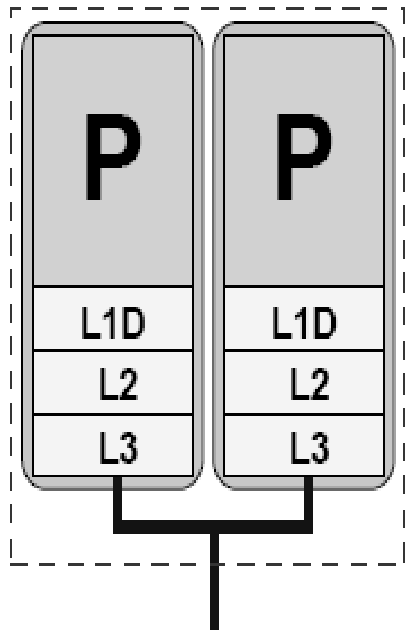


Figure 1.15: Dual-core processor chip with separate L1, L2, and L3 caches (Intel "Montecito"). Each core constitutes its own cache group on all levels.

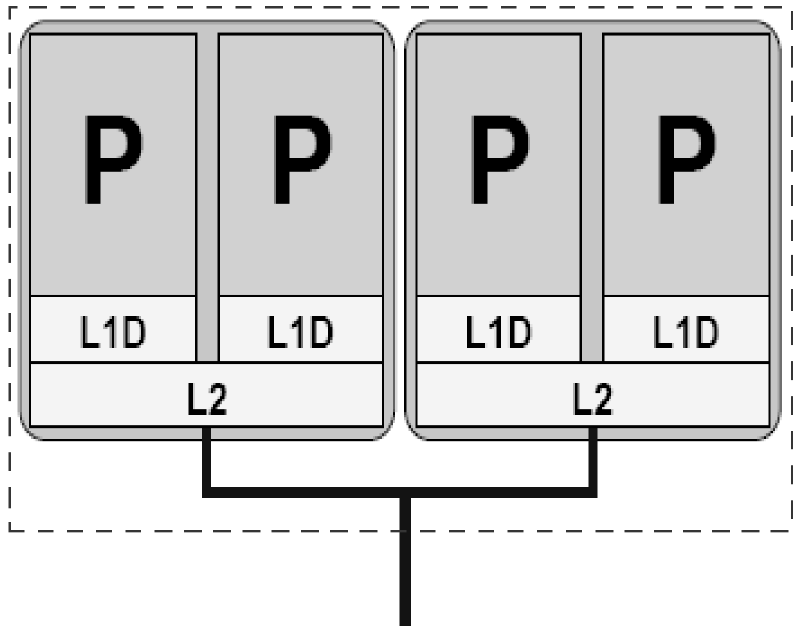


Figure 1.16: Quad-core processor chip, consisting of two dual-cores. Each dual-core has shared L2 and separate L1 caches (Intel "Harpertown"). There are two dual-core L2 groups.

Multithreaded Processor

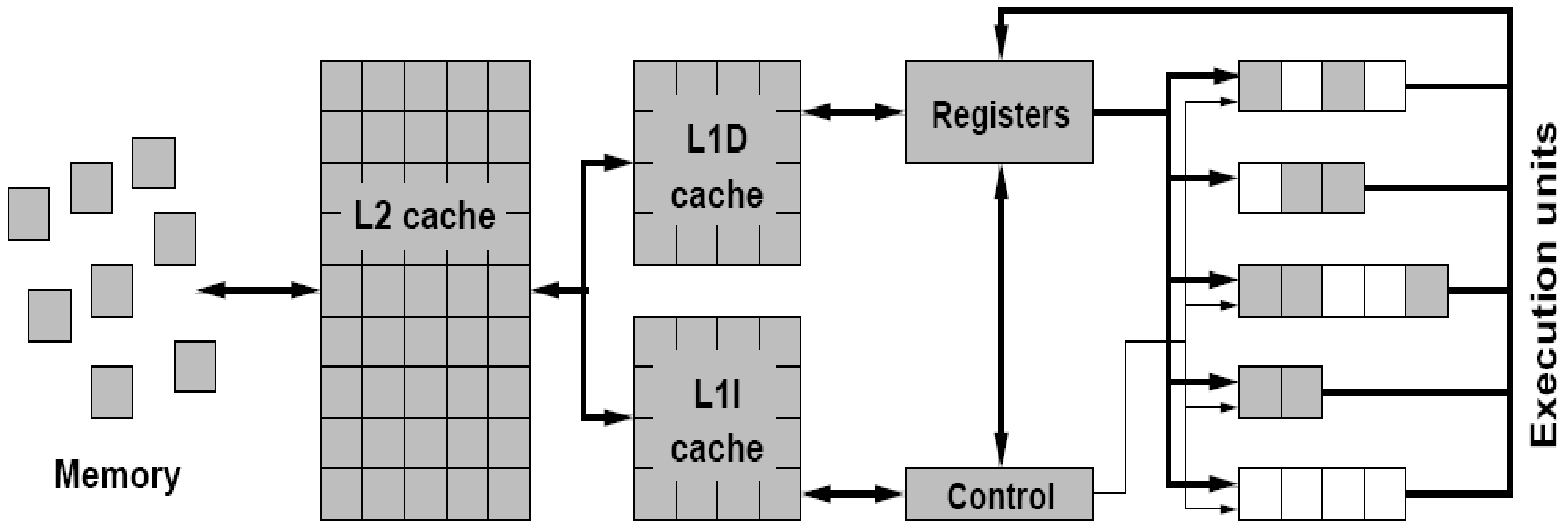


Figure 1.19: Simplified diagram of control/data flow in a (multi-)pipelined microprocessor without SMT. White boxes in the execution units denote pipeline bubbles (stall cycles). Graphics by courtesy of Intel.