



太原理工大学  
TAIYUAN UNIVERSITY OF TECHNOLOGY

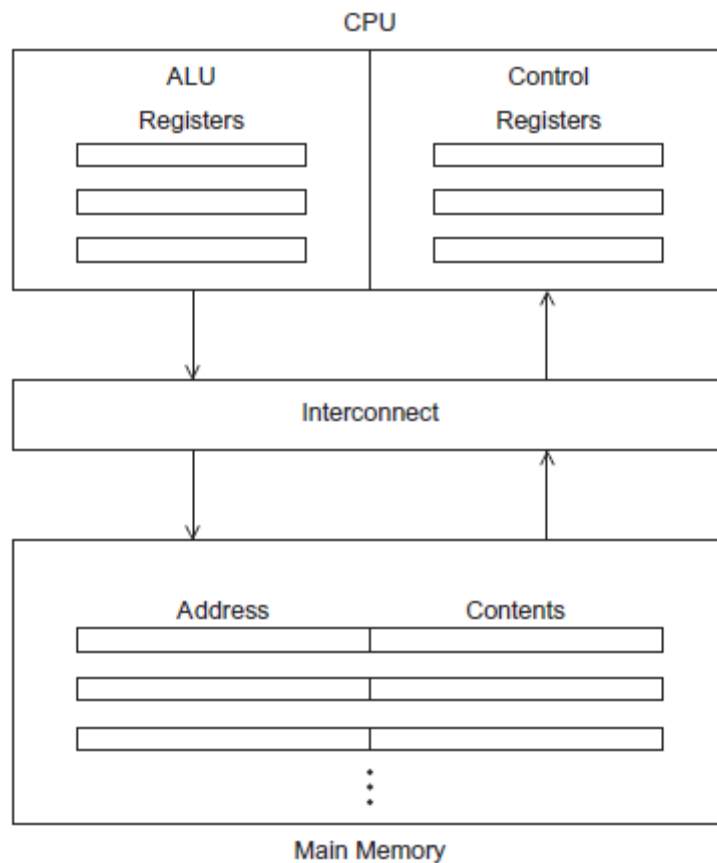
## 第二章 并行硬件和并行软件

岳俊宏

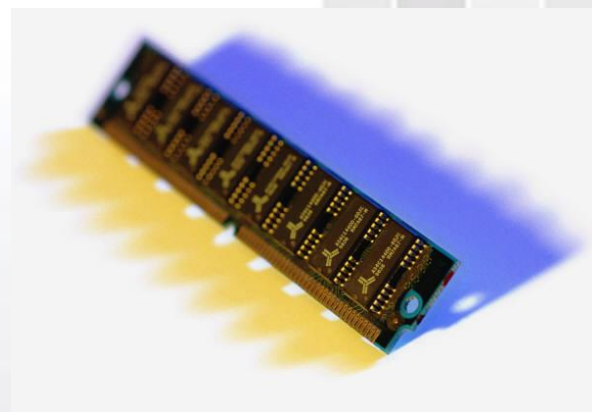
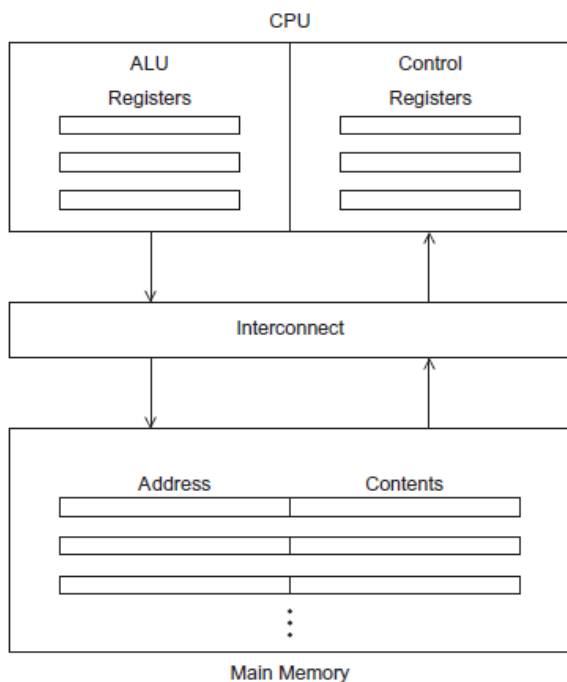
E-mail:yuejunhong@tyut.edu.cn; Tel:18234095983

The background features a faint world map. A large blue diamond shape is centered on the page, with a solid blue triangle in its top-left corner. The text "基本概念" is written in blue inside the diamond.

# 基本概念



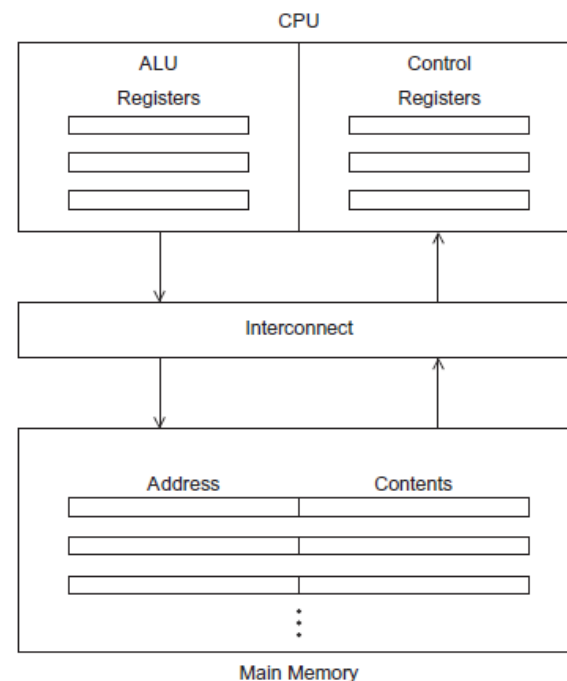
- 主存中有许多区域，每个区域都可以存储指令和数据。
- 每个区域都有一个地址，可以通过这个地址来访问相应的区域及区域中存储的数据和指令。



分成两部分：

□ 控制单元- 负责决定应该执行程序中的哪些指令。(the boss)

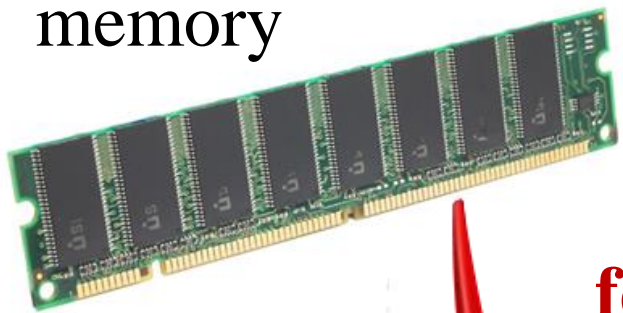
□ 算术逻辑单元(ALU) -负责执行指令。  
(the worker)



- CPU中的数据和程序执行时的状态信息存储在特殊的快速存储介质中，这种介质就是**寄存器**。
- 程序计数器—用来存放下一条指令的地址（控制单元的一个特殊的寄存器）。
- 总线—包括一组并行的线以及控制这些线的硬件。它是CPU和主存之间的互连结构，能够进行传输指令和数据。



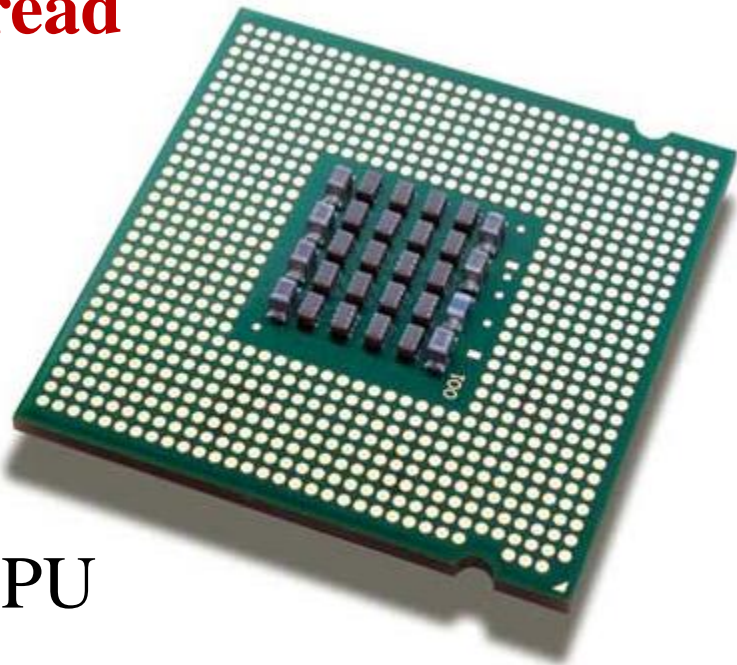
memory



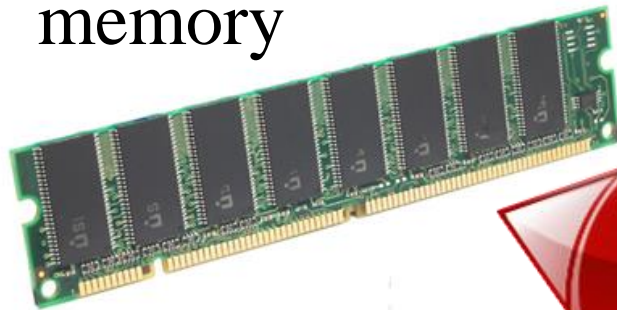
**fetch/read**



CPU



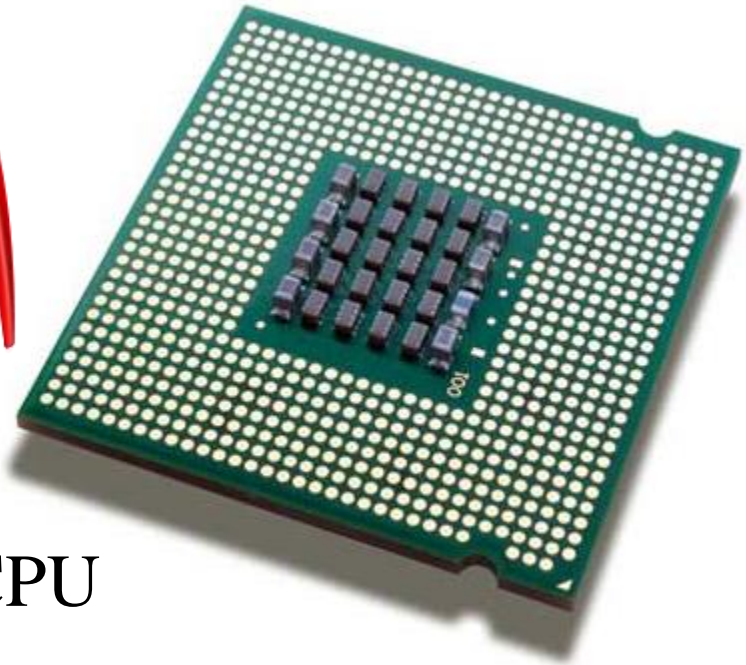
memory



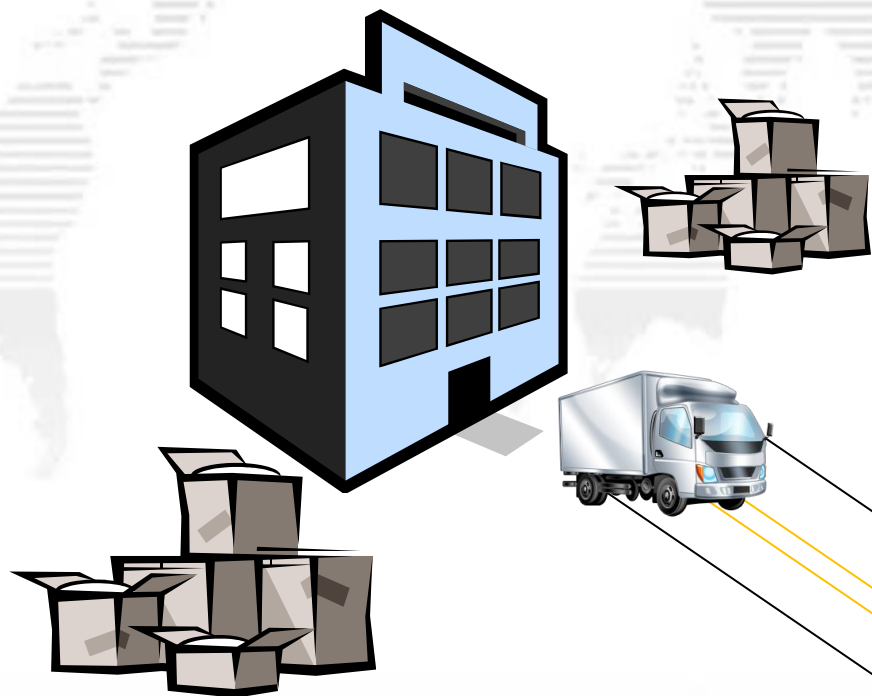
**write/store**



CPU







主存和CPU之间的分离称为**冯诺依曼瓶颈**。

互连结构限定了指令和数据访问的速率。



### □ 冯诺依曼模型的改进

- ✓ 不再将所有数据和指令存储在主存中，可以将部分数据块或者代码块存储在一个靠近CPU寄存器的特殊存储器里。

### □ 高速缓冲存储器简称**缓存**

- ✓ 访问它的时间比访问其它存储区域时间短。
- ✓ 相比于主存，CPU能更快速地访问的存储区域。（一般与CPU位于同一块芯片）

- **局部性原理**：程序访问完一个存储区域往往会访问接下来的区域。
- **时间局部性**— 程序在不久的将来进行访问。
- **空间局部性**— 访问临近的区域。

□ 高速缓冲块：一次内存访问存取的一整块代码和数据。

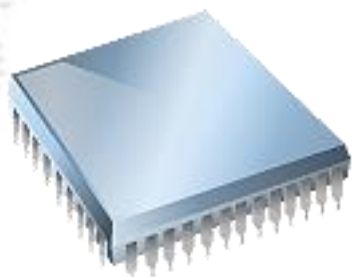
若高速缓冲块一次可以存放16个浮点数，则一次可将16个数从主存读到缓存中。

```
float z[1000];  
...  
sum = 0.0;  
for (i = 0; i < 1000; i++)  
    sum += z[i];
```

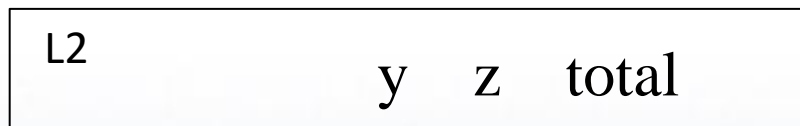
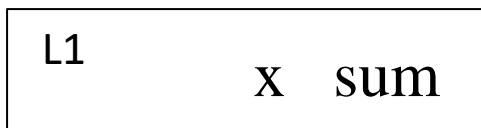
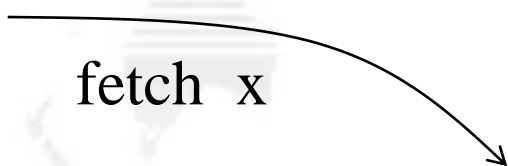
smallest & fastest

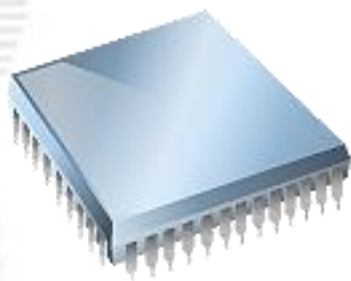


largest & slowest

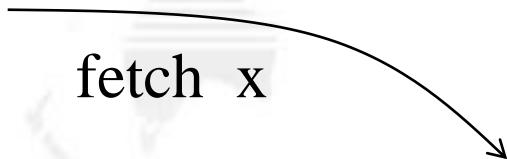


fetch x





fetch x



L1

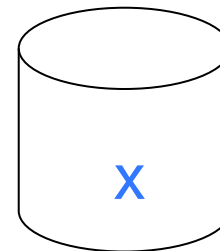
y sum

L2

r1 z total

L3

A[ ] radius center



main  
memory

- 当CPU向缓存中写数据时，Cache中的值与主存中的值就会不同或者不一致。
- **写直达缓存**：当CPU向缓存写数据时，高速缓存行会立即写入主存中。
- **写回缓存**：数据不是立即更新到主存中，而是将发生数据更新的高速缓存行标记为脏。当发生高速缓存行替换时，标记为脏的高速缓存行被写入主存中。

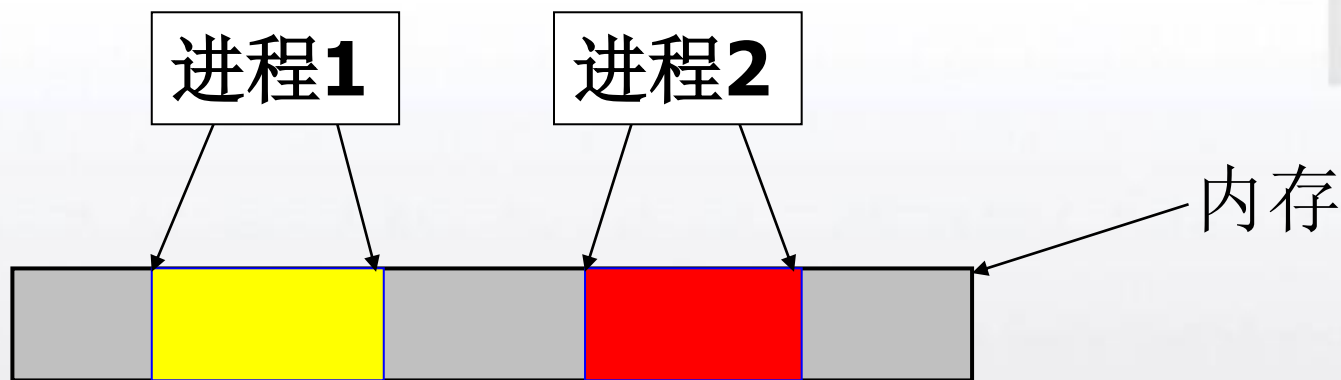


- 操作系统是用来管理计算机的软件和硬件资源的主要**软件**。
- 它决定什么程序能运行以及什么时候运行。
- 它控制运行中程序的内存分配以及硬盘、网卡等外设的访问。



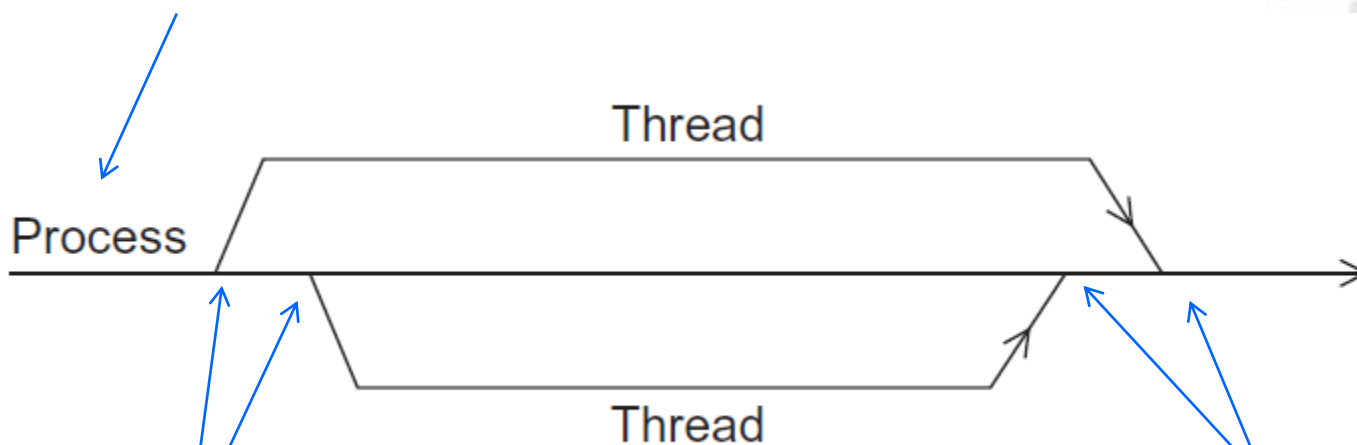
- 进程是正在运行的程序的实例
- 进程包括如下结构：
  - 可执行的机器语言程序
  - 一块内存空间
  - 操作系统分配给进程的资源描述符
  - 安全信息
  - 进程状态信息

- 给出一个单处理器系统同时运行多个程序的错觉。
- 每个进程轮流运行（时间片：几毫秒）
- 在一个进行的时间片结束后，它将等待直到再次轮到它运行。



- 线程被包含在进程中
- 允许程序员将进程划分为多个大致独立的任务.
- 当某个任务阻塞时能够执行其他任务.

the “master” thread



starting a thread

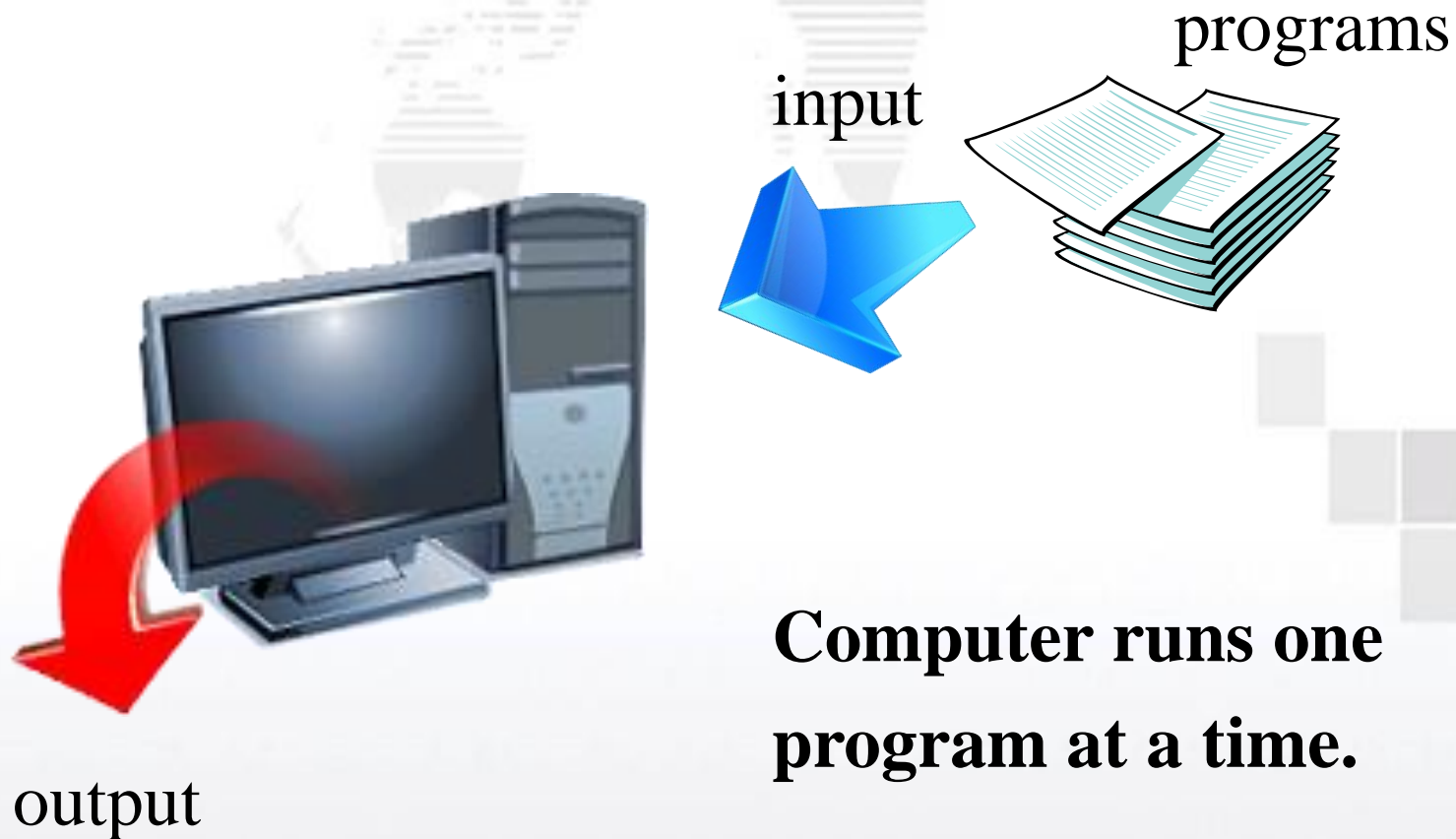
Is called forking

terminating a thread

Is called joining

The background features a faint world map. A large blue diamond shape is centered on the slide, with a solid blue triangle in its top-left corner. The text "并行硬件" is written in blue inside the diamond.

# 并行硬件



- ◆ Flynn经典分类
- ◆ 内存结构分类
- ◆ 互连网络
- ◆ Cache一致性

**Flynn分类法**经常用来对计算机体系结构进行分类。

- ✓ 1966年被提出（50多年）；
- ✓ 从处理器的角度进行分类：按照处理器能够同时管理的**指令**流数目和**数据**流数目来对系统分类。

<u><b>SISD</b></u> Single Instruction Single Data	<u><b>SIMD</b></u> Single Instruction Multiple Data
<u><b>MISD</b></u> Multiple Instruction Single Data	<u><b>MIMD</b></u> Multiple Instruction Multiple Data



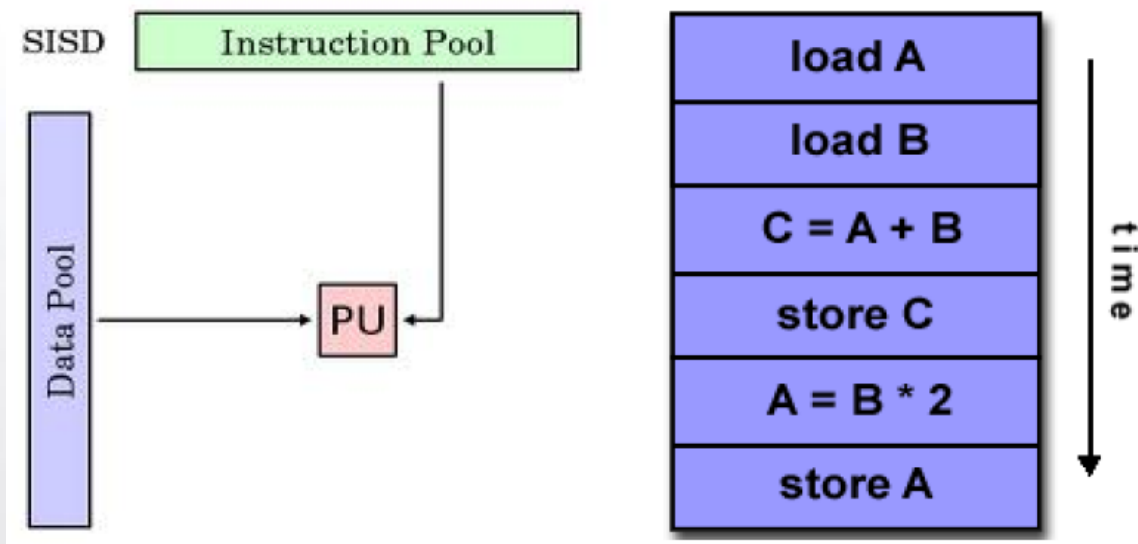
单指令单数据流(SISD):是一个**串行**的计算机系统

**单指令:** 在一个时钟周期内只有一个指令流被CPU进行处理。

**单数据:** 在一个时钟周期内只有一个数据流作为输入而使用。

Example:

- ✓ 冯诺依曼系统
- ✓ 单核处理器

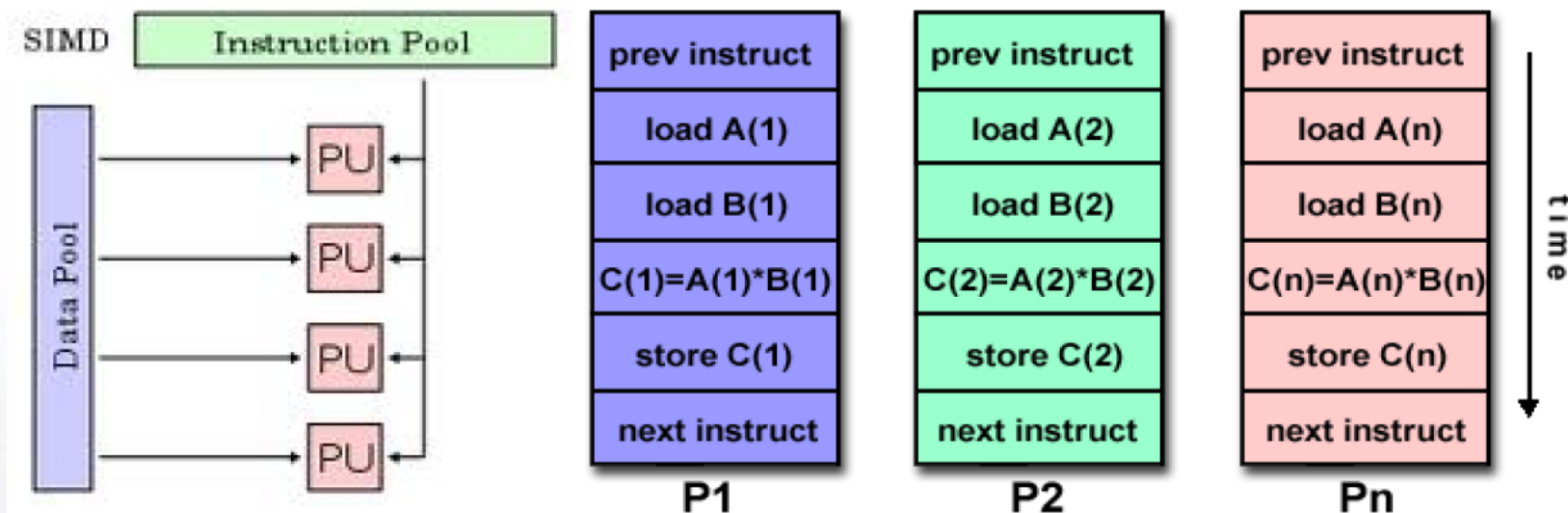


单指令多数据流(SIMD):是**并行系统**。

- ✓ **单指令**: 在一个时钟周期内, 所有的处理器单元执行相同的指令
- ✓ **多数据流**: 每个处理器单元能够作用在不同的数据单元上。
- ✓ SIMD系统通过对多个数据执行相同的指令从而实现在多个数据流上的操作。

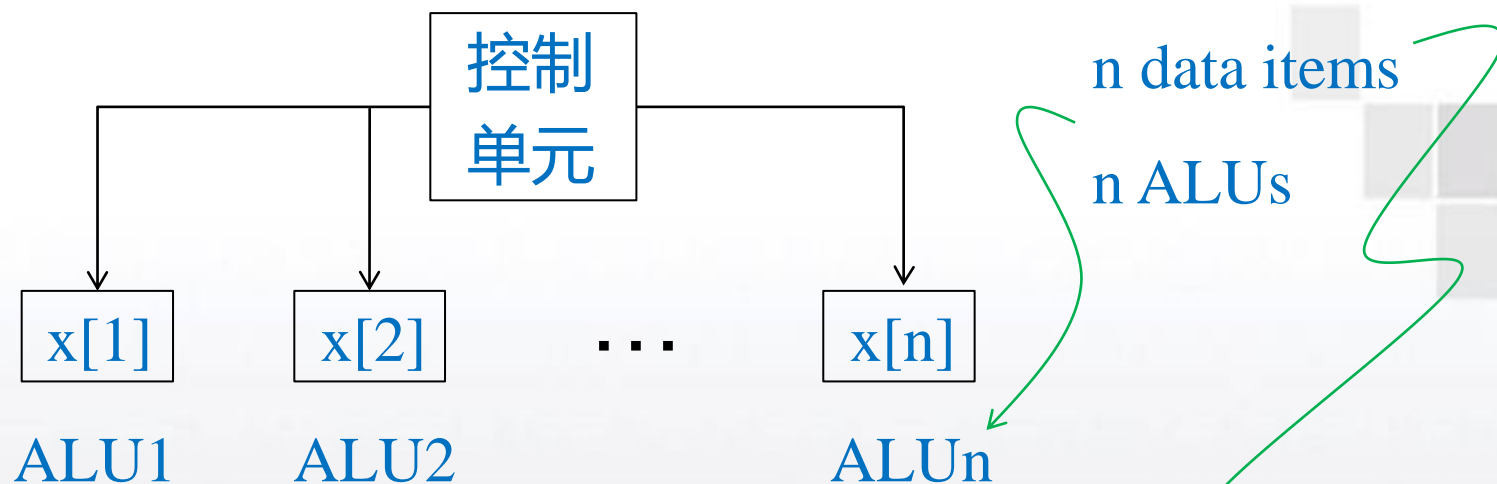
通过在处理器之间划分数据实现并行性(**数据并行**)

Eg: 数组(array data)



◆ 一个抽象的SIMD = 一个控制单元 + 多个ALU。

Eg: 假设执行一个向量加法，有两个数组x和y，每个都有n个元素，现将y中的元素加到x中。



```
for (i = 0; i < n; i++)  
    x[i] += y[i];
```

- ◆ 如果没有足够多的ALU分配给各个数据项 应该怎么办?
- ◆ 划分工作, 并进行迭代循环处理。
- ◆ 如:  $m = 4$  ALUs , 且  $n = 15$  数据项.

Round3	ALU <sub>1</sub>	ALU <sub>2</sub>	ALU <sub>3</sub>	ALU <sub>4</sub>
1	X[0]	X[1]	X[2]	X[3]
2	X[4]	X[5]	X[6]	X[7]
3	X[8]	X[9]	X[10]	X[11]
4	X[12]	X[13]	X[14]	

### SIMD缺陷

- ✓ 在经典的SIMD设计中，ALU必须同步操作.
- ✓ 所有的ALU需要执行相同的指令或处于空闲状态.
- ✓ SIMD适合处理大型数据并行问题，但是在处理其它类型或更复杂问题上并不适合，性能较差。

## SIMD系统的发展

- ✓ 20世纪90年代早期: Thinking machine公司是并行超级计算机最大的制造者 (制造SIMD系统) .
- ✓ 20世纪90年代末: 向量处理器.
- ✓ 如今: 图形处理单元 (Graphics processing unit, GPU) .

**向量处理器**对数组或数据向量进行操作，而传统的CPU是对单独数据元素或标量进行操作。

## ◆ 向量寄存器

- ✓ 能够操作由多个操作数组成的向量，并能同时对其内容进行操作的寄存器。

## ◆ 向量化和流水化的功能单元

- ✓ 对向量中的每个单元做相同的操作。

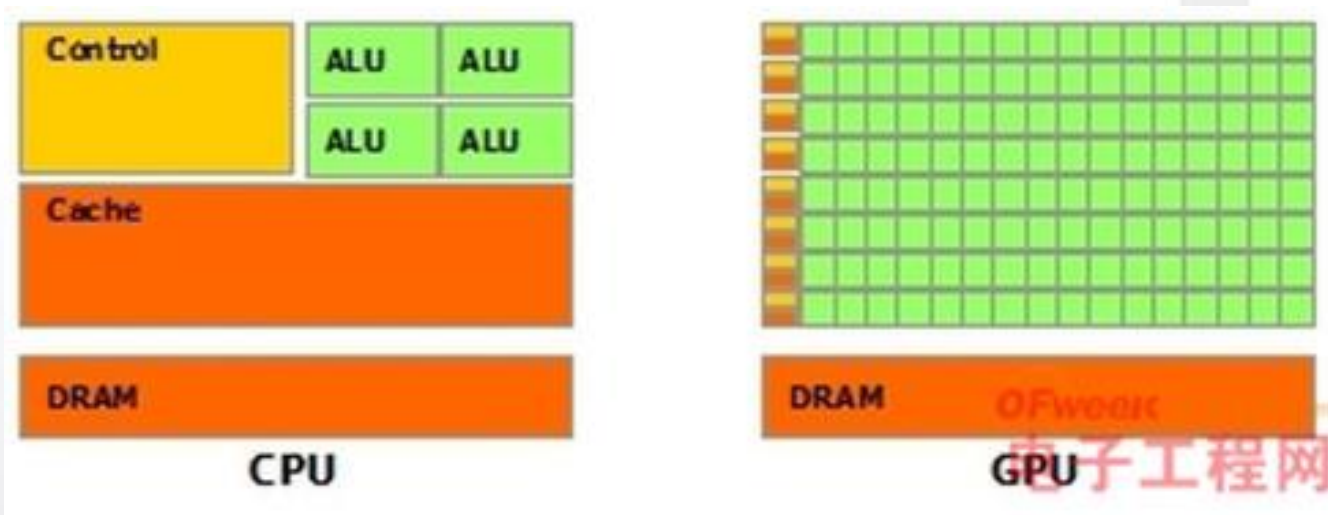
## ◆ 向量指令

- ✓ 在向量上操作而不是在标量上操作的指令



- ◆ 向量处理器的优点：快、容易使用，有高的带宽等；
- ◆ 向量处理器的缺点
  - ✓ 不能处理不规则的数据结构；
  - ✓ 在处理更大规模问题方面，能力会受到限制，扩展性差。

- ◆ 图形处理器（Graphics Processing Unit, GPU）是一种专门在个人电脑、工作站、游戏机和一些移动设备上做图像和图形相关运算工作的微处理器。

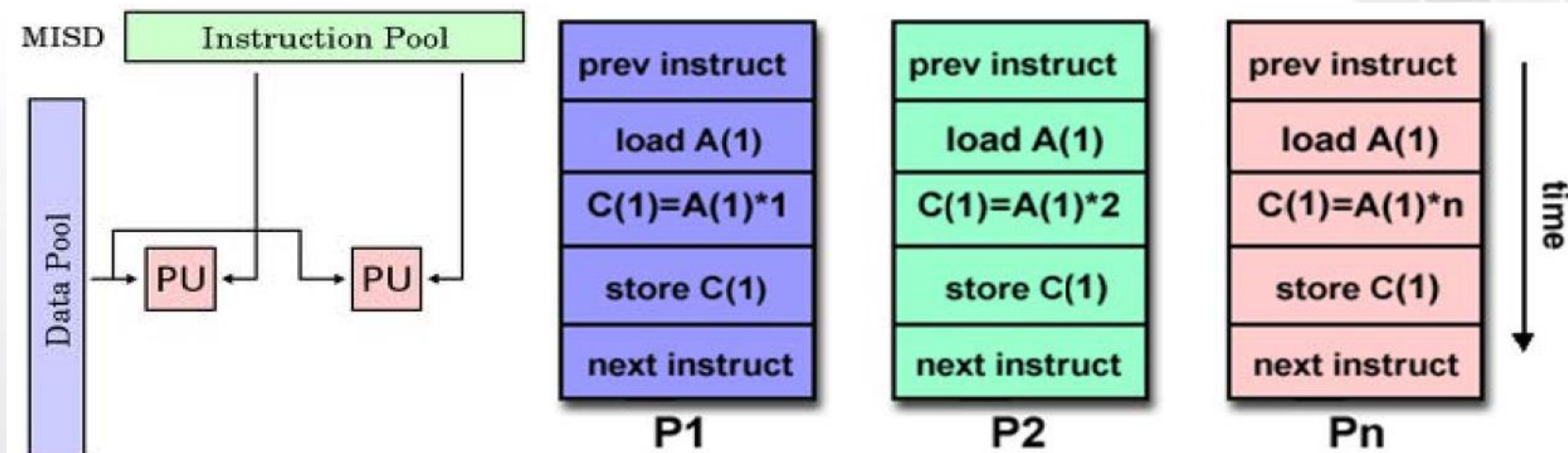


<b>GPU</b>	简单运算 进行大量并行计算的体力劳动者	GPU相当于一个接受CPU调度的“拥有大量计算能力”的员工	NVIDIA 超级计算的 Tesla 系列 (NV Tesla V100)  NVIDIA TITAN V 是为 PC 打造的更强大的显卡	大型矩阵运算;  图形处理; 深度学习
<b>CPU</b>	复杂运算 复杂脑力劳动的教授	CPU是一个有多种功能的优秀领导者。它的优点在于调度、管理、协调能力强, 计算能力则位于其次	个人、商务电脑	

### 多指令单数据流(MISD):

**多指令:**每个处理单元通过单独的指令流独立地对数据进行操作.

**单数据流:**单个数据流被输入多个处理单元.



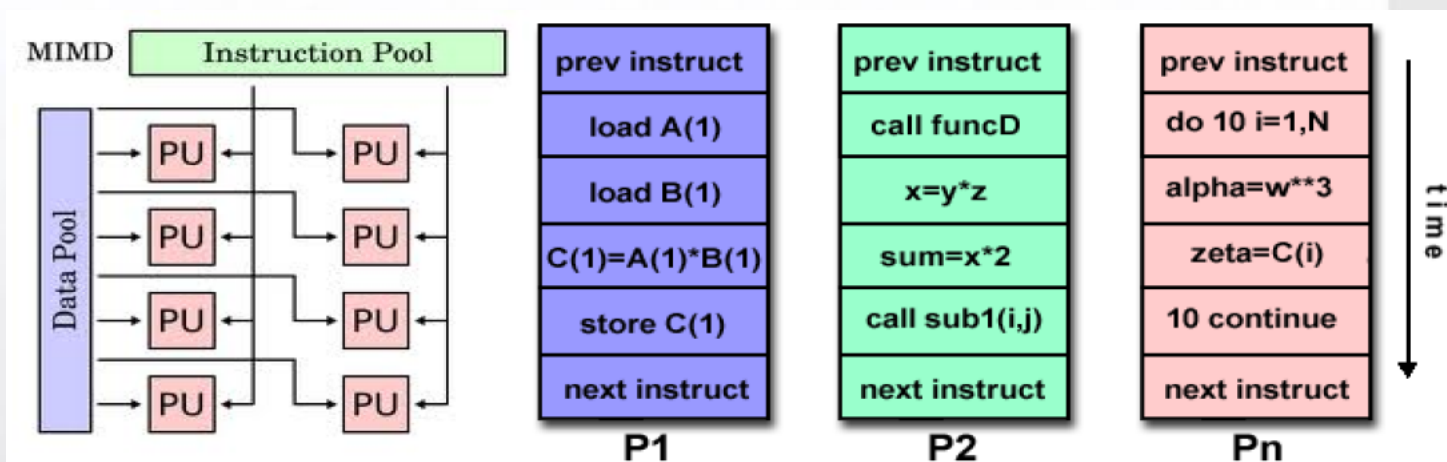
Example:

Only experiment by CMU in 1971;

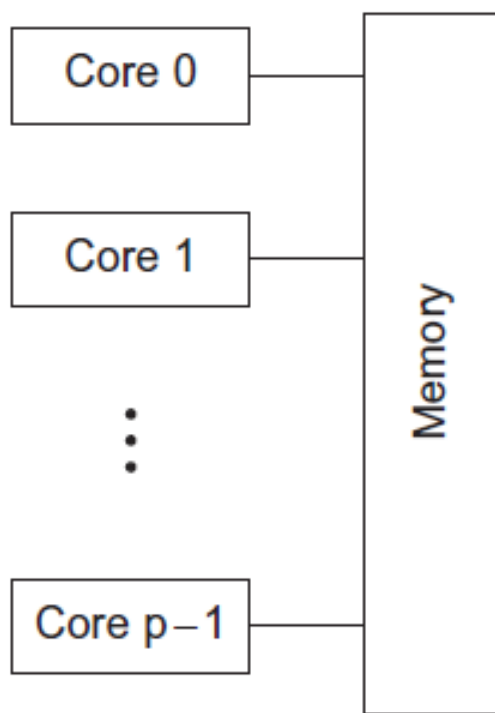
Could be used for fault tolerance (容错)

### 多指令多数据流(MIMD):

- ◆ **多指令**:每个处理器可能执行不同的指令流
- ◆ **多数据**:每个处理器可能使用不同的数据流
- ◆ 支持多个指令流同时运行在多个数据流上。
- ◆ 通常由完全独立的处理单元或核心组成, 每个处理单元都有自己的控制单元和ALU.

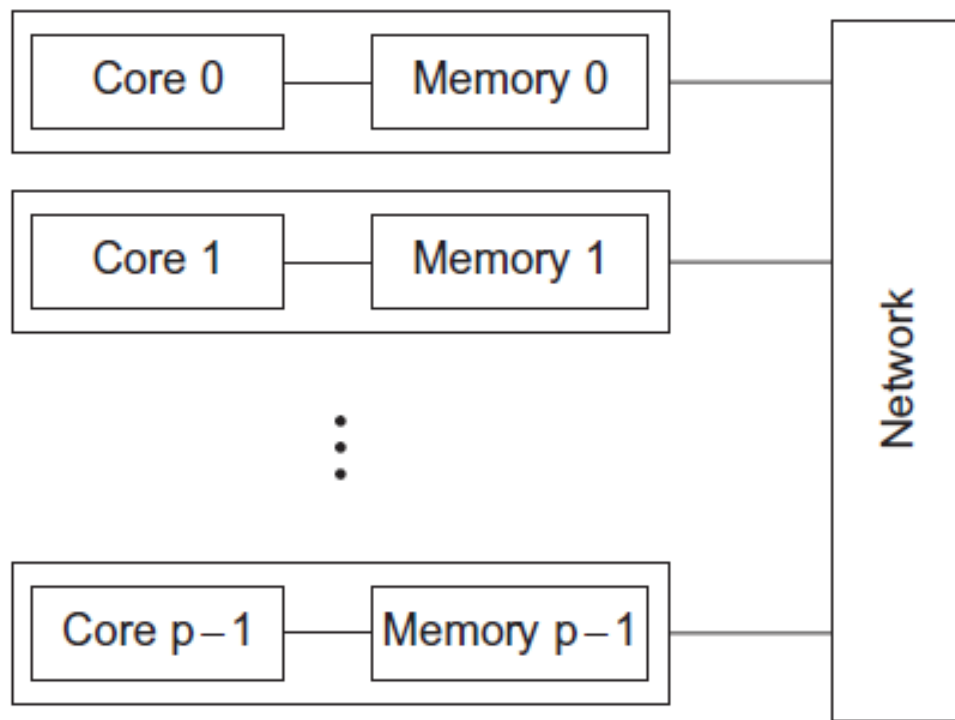


### ◆ Example: 大多数现代计算机, 如多核CPU



(a)

共享内存系统



(b)

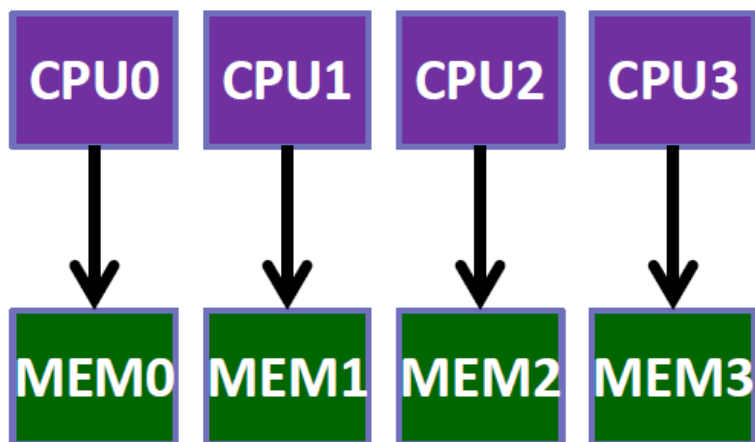
分布式内存系统



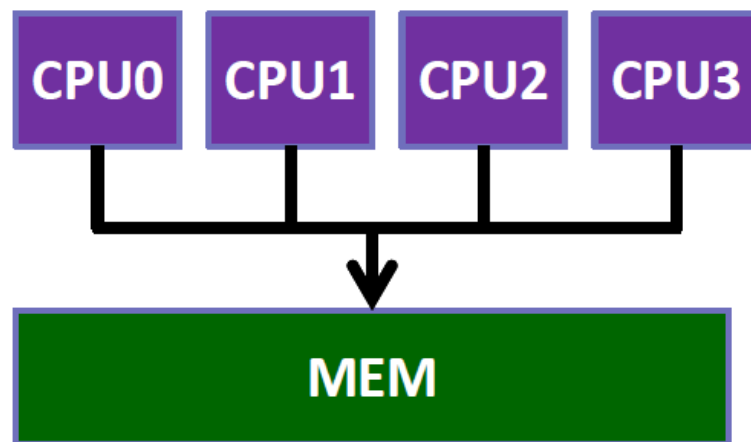
# 内存结构分类



- 共享内存计算机体系结构
- 分布式内存计算机体系结构



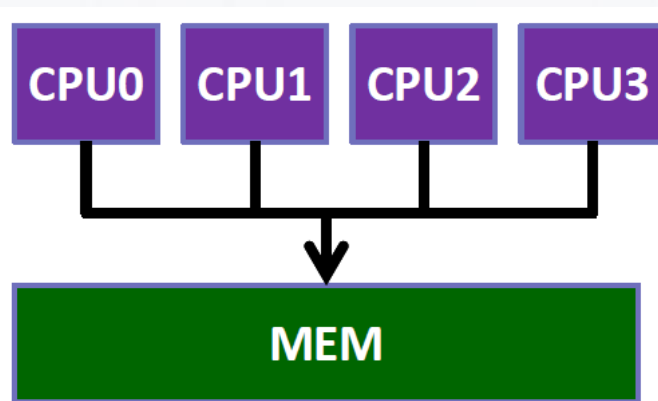
**Distributed memory**



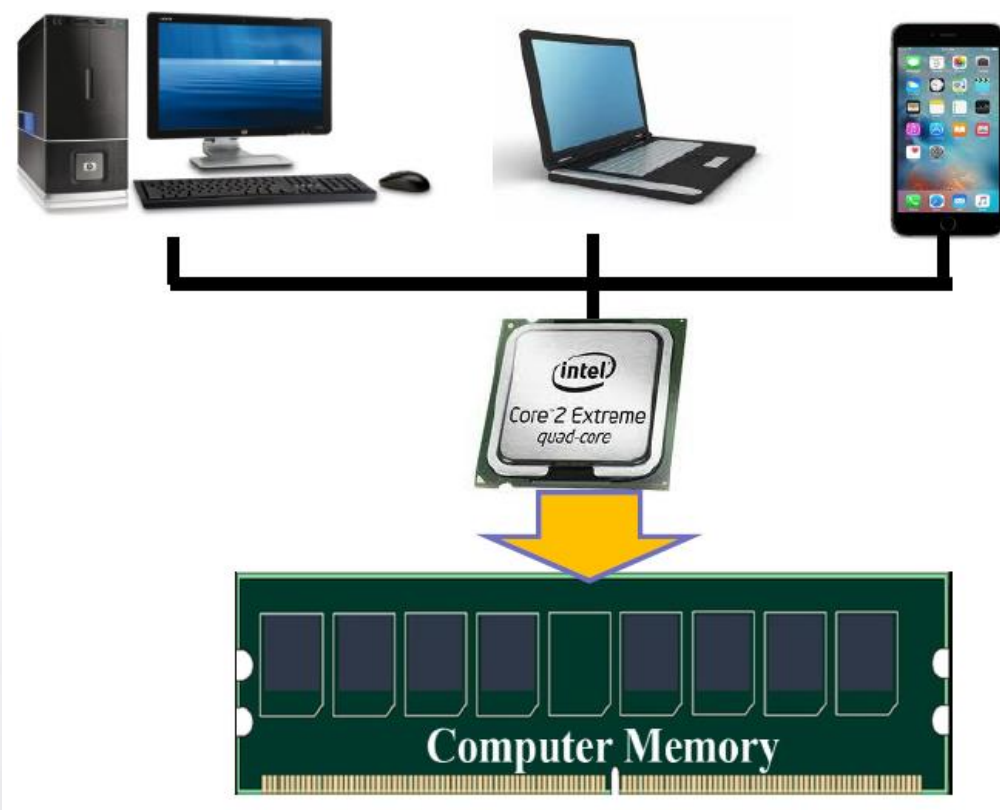
**Shared memory**

### □ 共享内存系统

- ✓ 一组自治的处理器通过互连网络与内存系统连接.
- ✓ 各个核可以共享访问计算机的内存系统，每个处理器能够访问每个内存区域.
- ✓ 可以通过检测和更新内存系统来协调各个核，处理器通过访问共享的数据结构来隐式的通信。

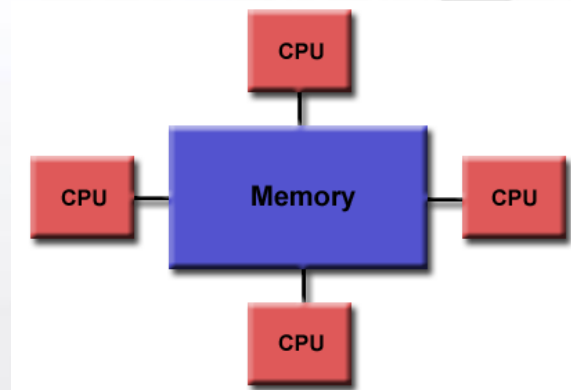
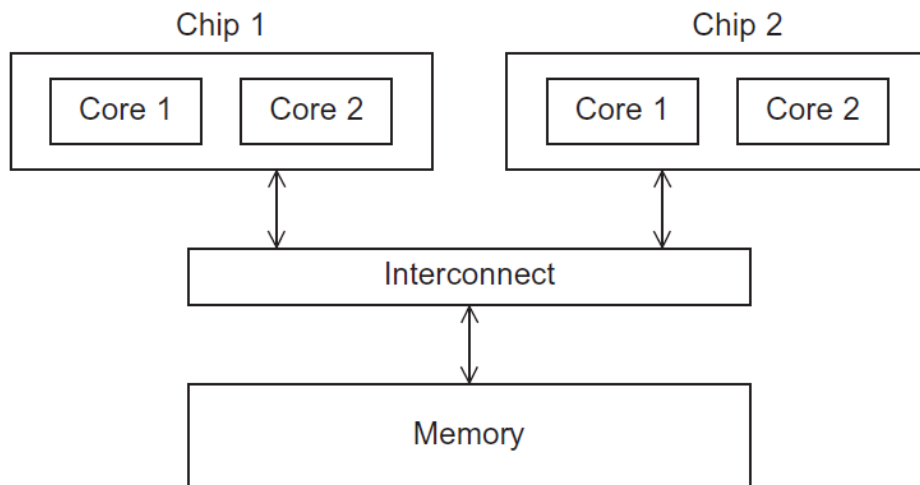


- 大多数共享内存系统使用一个多核处理器或者多个多核处理器
- 在一块芯片上  
有多个CPU或者核

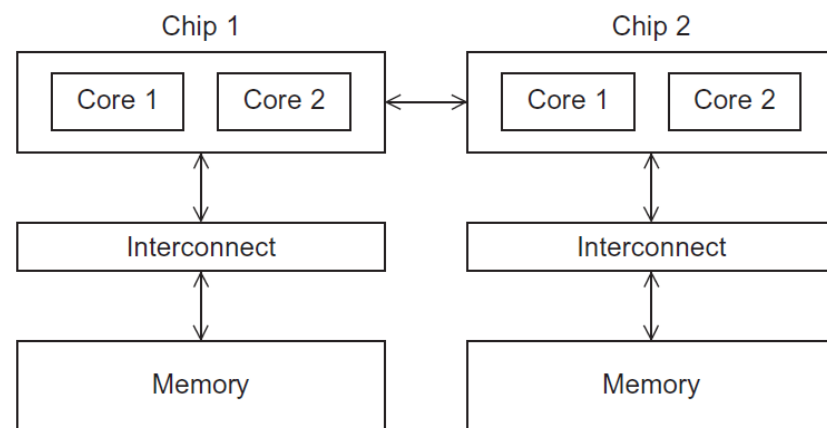
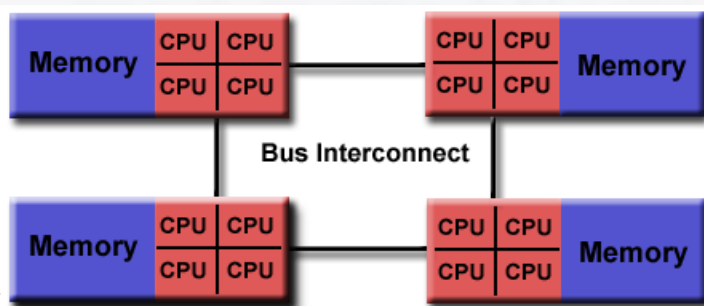


□ 一致内存访问(UMA):互连网络将所有的处理器直接连接到主存。

- ✓ 每个核访问内存任意位置的时间相等
- ✓ 如：对称多处理器（SMP）



- 非一致内存访问(NUMA): 将每个处理器直接连到单独的一块内存上, 然后通过处理器中内置的特殊硬件使得各个处理器可以访问内存中的其它块。
  - ✓ 通常是通过物理连接两个或多个SMP来实现的
  - ✓ 一个SMP可以直接访问另一个SMP的内存
  - ✓ 访问核直接连接到的内存位置比必须通过另一个芯片进行访问的内存位置更快。
  - ✓ 如: HPC服务器



### □ UMA与NUMA系统的特点：

- ✓ 一致内存访问(UMA)系统容易编程。
- ✓ 非一致内存访问（NUMA）系统拥有更大容量的内存。

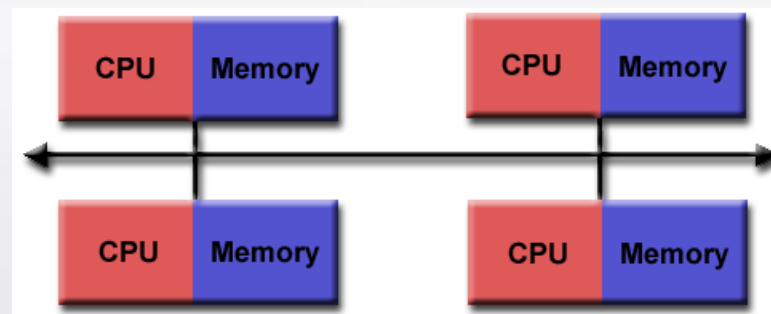
## □ 分布式内存系统

- ✓ 每个核都有自己独立的私有的内存.
- ✓ 核之间的通信需要通过网络发送消息, 进行显式的通信.
- ✓ 将多台计算机连接起来形成一个计算平台, 而没有共享内存

□ 网络提供一种基础架构, 使地理上分布的计算机大型网络转换成一个分布式内存系统。

## □ 网络设备:

- ✓ Ethernet
- ✓ InfiniBand



## □ 集群(最广泛)

- ✓ 一组商品化系统组成(如PC).
- ✓ 通过商品化网络连接(如以太网).

## □ 集群的节点指的是通过通信网络相互连接的独立计算单元

## □ 混合系统：节点通常都是有一个或多个多核处理器的共享内存系统.





Cluster: tens of servers



Supercomputer: hundreds of servers



Datacenter: thousands of servers



# 互连网络

- 影响分布式和共享内存系统的性能。

- 两类：

- ✓ 共享内存互连
- ✓ 分布式内存互连

## □ 总线

- ✓ 由一组并行通信线和控制对总线访问的硬件组成.
- ✓ 连接到总线上的设备共享通信线.
- ✓ 随着连接到总线设备的增多, 争夺总线的概率增大, 总线的预期性能会下降.

### □ 交换互连网络

- ✓ 使用交换器来控制相互连接设备之间的数据传递.

### □ 交叉开关矩阵

- ✓ 允许在不同设备之间同时进行通信.
- ✓ 比总线速度快
- ✓ 交换器和链路带来的开销相对较高.

□ 两类：

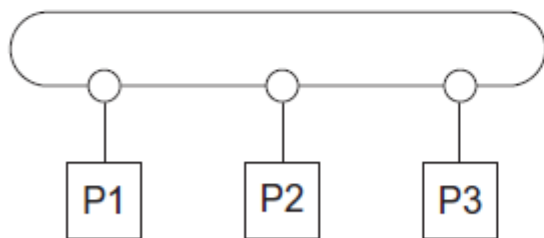
□ 直接互连

✓ 每个交换器与一个处理器-内存对直接相连，交换器之间也相互连接.

□ 间接互连

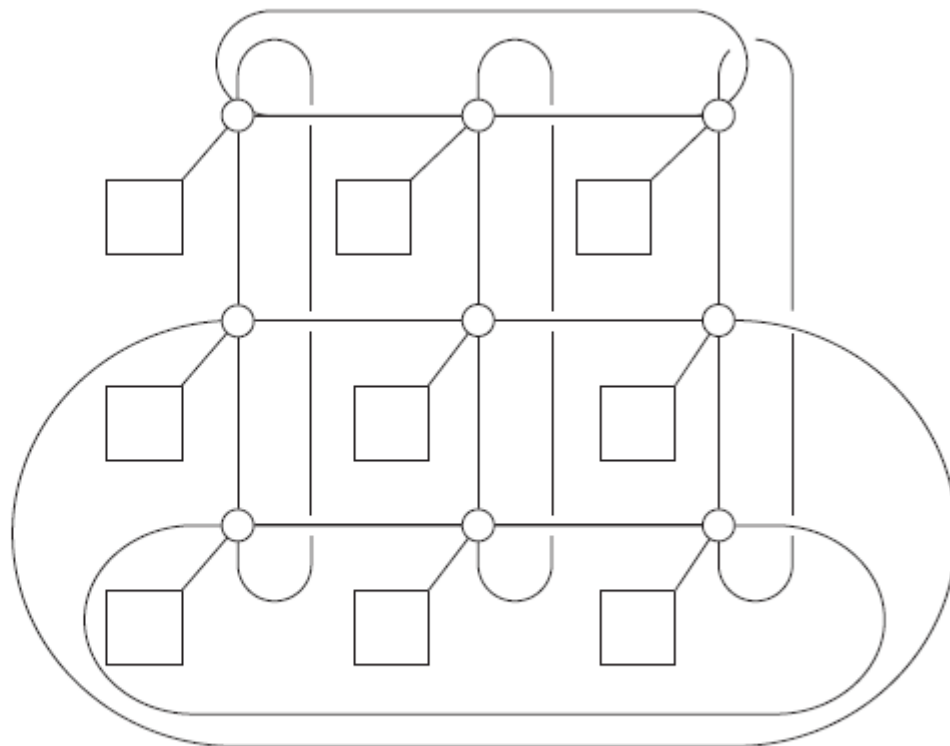
✓ 交换器不一定与处理器直接连接.

### □ 直接互连



(a)

环

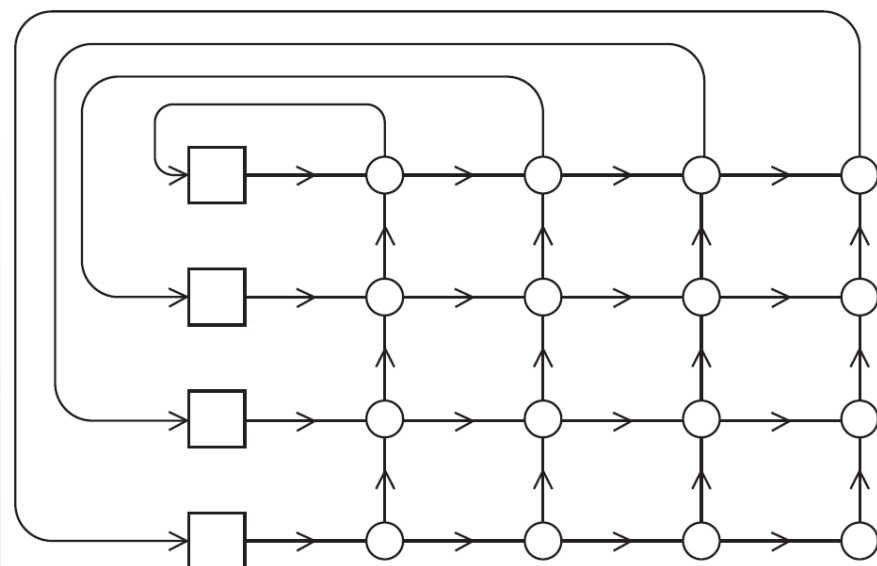
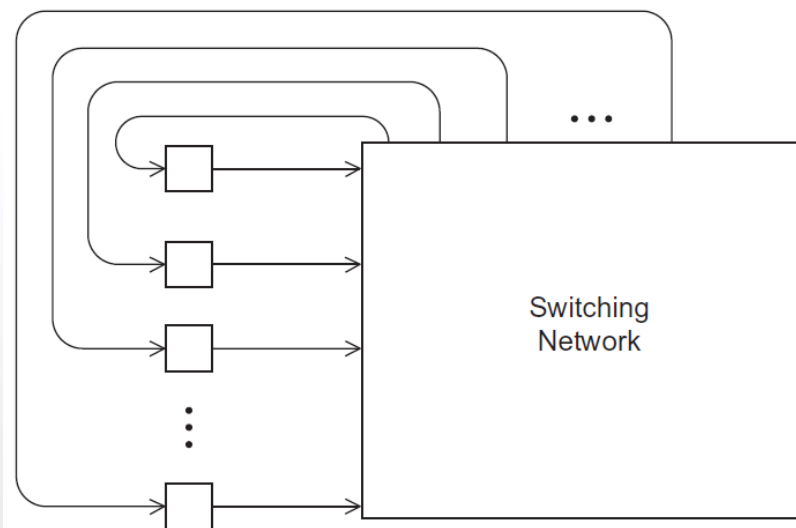


(b)

二维环面网络

## □ 间接互连

- ✓ 通常由一些单向连接和一组处理器组成，每个处理器有一个输入链路和一个输出链路，以及一个交换网络。
- ✓ 交叉开关和Omega 网络是间接网络中相对简单的例子

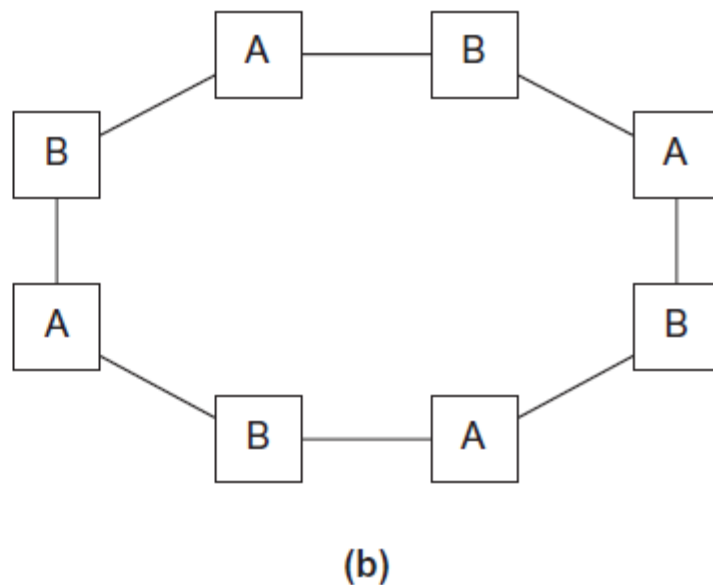
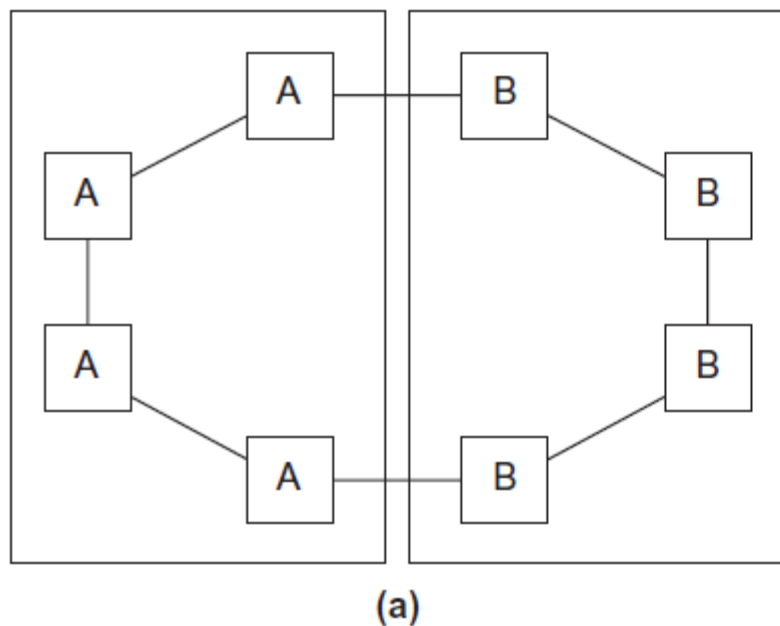






## □ 等分宽度

- ✓ “同时通信链路数目” 或 “连接性” 的量度。
- ✓ 并行系统被等分成两部分，在这两部分之间能同时发生多少通信



## □ 带宽

- ✓ 链路传输数据的速度.
- ✓ 通常用兆位每秒或者兆字节每秒表示.

## □ 等分带宽

- ✓ 网络质量的度量.
- ✓ 不是计算连接两个等分之间的链路数，而是计算链路的带宽。

- 任何时间数据被传输，我们感兴趣的是数据需要多长时间才能到达目的地。
- 延迟
  - ✓ 从发送源开始传输数据到目标地开始接收数据之间的时间。
- 带宽
  - ✓ 目的地在开始接收数据后接收数据的速度。

- 如果一个互连网络的延迟是  $l$  秒，带宽是  $b$  字节每秒，则传输一个  $n$  字节的消息需要花费的时间： $l + n / b$

latency (seconds)

length of message (bytes)

bandwidth (bytes per second)

## □ 网络设备: InfiniBand

- ✓ 用于高性能计算的计算机网络通信链路，具有非常高的吞吐量
- ✓ 它是超级计算机中最常用的互连网络
- ✓ 生产厂商：纳斯达克（Mellanox）

## InfiniBand vs. Ethernet

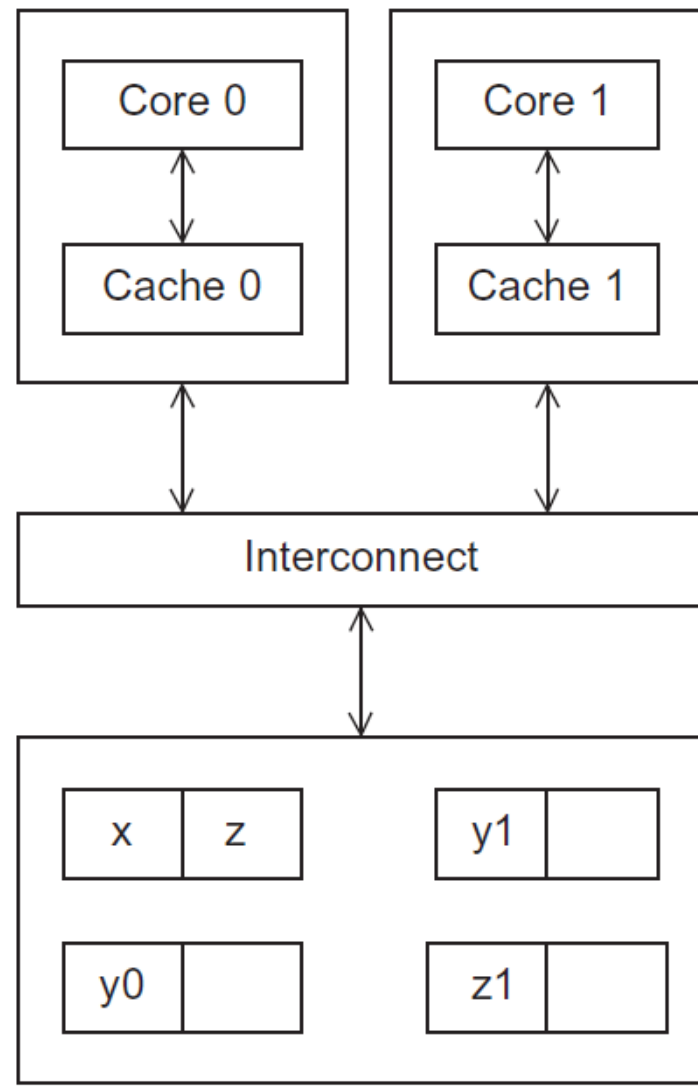
	InfiniBand	Ethernet
Protocol	Guaranteed credit based flow control	Best effort delivery
	End-to-End congestion management	TCP/IP protocol. Designed for L3/L4 switching
	Hardware based retransmission	Software based retransmission
RDMA	YES	NO (only now starting)
Latency	Low	High
Throughput	High	Low
Max cable length	4km	upto 70km
Price	36port switch: 25k USD QDR adapter: 500USD	36port switch: 1.5k USD Network card: 50 USD

- 由于CPU缓存是由系统硬件来管理的，程序员不能对它进行直接控制。

假设有两个核和两个Cache的共享系统

y0 privately owned by Core 0

y1 and z1 privately owned by Core 1



`x = 2; /* shared variable */`

Time	Core 0	Core 1
0	<code>y0 = x;</code>	<code>y1 = 3*x;</code>
1	<code>x = 7;</code>	Statement(s) not involving x
2	Statement(s) not involving x	<code>z1 = 4*x;</code>

y0 eventually ends up = 2

y1 eventually ends up = 6

z1 = ???



### □ 监听Cache一致性协议

- ✓ 多个核共享总线.
- ✓ 总线上传递的信号都能被连接到总线的所有核看到。
- ✓ 当核0更新它缓冲中x的副本时，它也将这个新信息在总线上广播。 .
- ✓ 如果核1正在监听总线，那么它会知道x已经更新，并将自己Cache中的x副本标记为非法的。

### □ 基于目录的Cache一致性协议

- ✓ 使用名为目录的数据结构，该目录存储每个缓存行的状态.
- ✓ 当一个变量被更新时，就会查询目录，并将所有包含该变量高速缓存行（可能位于不同的核上）置为非法。

The background features a faint world map. A large blue diamond shape is centered on the slide, with a solid blue triangle in its top-left corner. The text "并行软件" is written in blue inside the diamond.

# 并行软件

- 软件应用能够通过硬件和编译器的更新来保持所需的速度。
- 多核系统，分布式系统的出现。
- 从应用程序的类型可分为：
  - 共享内存程序：
    - ✓ 启动单个进程和派生线程.
    - ✓ 线程执行任务.
  - 分布式内存程序：
    - ✓ 启动多个进程.
    - ✓ 进程执行任务

- SPMD(single program multiple data),
- ✓ SPMD程序仅包含一段可执行代码，通过使用条件转移语句，可以让这一段代码在执行时表现得像在不同处理器上执行不同的程序。

```
if (I'm thread process i)
    do this;
else
    do that;
```



- 将任务在进程/线程间分配
  - ✓ 每个进程/线程获得大致相等的工作量
  - ✓ 通信量最小化.
- 安排进程/线程之间的同步.
- 安排进程/线程之间的通信.

```
double x[n], y[n];  
...  
for (i = 0; i < n; i++)  
    x[i] += y[i];
```

## □ 动态线程


- ✓ 主线程等待工作请求，派生新线程，当线程完成任务，就会终止执行，再合并到主线程。
- ✓ 充分利用了系统资源，但线程的创建和终止是非常耗时的。

## □ 静态线程

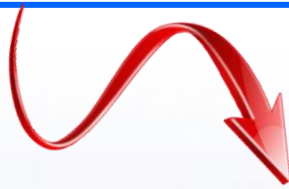
- ✓ 线程池被创建，并分配工作，但直到做一些清理时，才会终止所有线程。
- ✓ 如果所需资源是可用的，静态线程更高效。但也有可能会浪费系统资源。

在任何一个MIMD系统中，如果处理器异步执行，那么很可能会引发不确定性，比如输出。

```
...  
printf ( "Thread %d > my_val = %d\n" ,  
        my_rank , my_x ) ;  
...
```



Thread 1 > my\_val = 19  
Thread 0 > my\_val = 7



Thread 0 > my\_val = 7  
Thread 1 > my\_val = 19



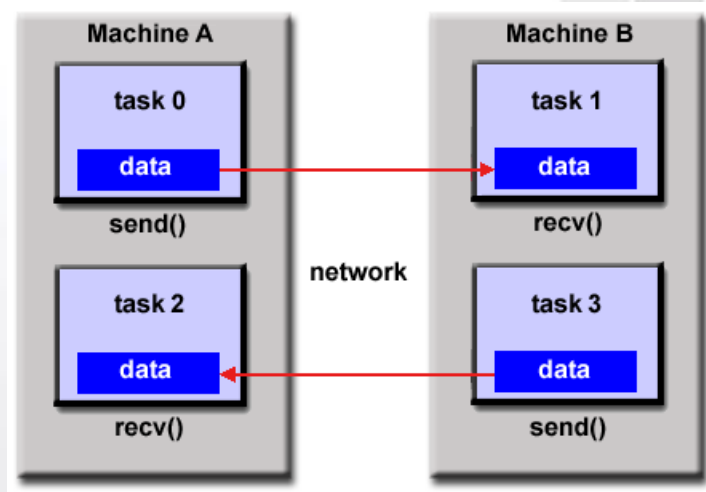
- 竞争条件
- 临界区
- 互斥的
- 互斥锁 (互斥量或者简单锁)

```
my_val = Compute_val ( my_rank ) ;  
Lock(&add_my_val_lock ) ;  
x += my_val ;  
Unlock(&add_my_val_lock ) ;
```

### □ 分布式内存系统上的API（并行编程模型）

- ✓ 消息传递
- ✓ 单向通信
- ✓ 划分全局地址空间的语言

- 在计算过程中使用自己的局部内存的一组任务
- 多个任务可以驻留在同一台物理机器上，或跨任意数量的机器
- 任务通过发送与接收消息的通信来交换数据 (Memory copy)
- MPI API:
  - ✓ Send, Recv, Bcast,
  - ✓ Gather, Scatter, etc.



消息传递的API要提供一个发送和接收的函数，进程之间通过标号相互识别。如

```
char message[100] ;
```

```
...
```

```
my_rank = Get_rank ( ) ;
```

```
i f ( my_rank == 1) {
```

```
    sprintf ( message , "Greetings from process 1" ) ;
```

```
    Send ( message , MSG_CHAR , 100 , 0 ) ;
```

```
} else if ( my_rank == 0) {
```

```
    Receive ( message , MSG_CHAR , 100 , 1 ) ;
```

```
    printf ( "Process 0 > Received: %s\n" , message ) ;
```

```
}10:41
```

- PGAS(Partitioned Global Address Space Languages)语言提供了一种共享内存程序的机制

```
shared i n t n = . . . ;  
shared double x [ n ] , y [ n ] ;  
private i n t i , my_first_element , my_last_element ;  
my_first_element = . . . ;  
my_last_element = . . . ;  
/ * Initialize x and y */  
.  
.  
f o r ( i = my_first_element ; i <= my_last_element ; i++)  
    x [ i ] += y [ i ] ;
```



# 输入输出

C语言中输入和输出函数设计的时候，并没有考虑到并行的情况，所以在并行编程的时候必须格外注意。

- 在分布式内存程序中，只有进程0能够访问stdin。在共享内存程序中，只有主线程或线程0能够访问stdin.
- 在分布式内存和共享内存系统中，所有进程/线程都能够访问stdout 和stderr.

- 但是，因为输出到stdout的的非确定性顺序，在大多数情况下，除了调试输出之外，只有一个进程/线程会将结果输出到stdout。
- 调试程序输出应该始终包括生成输出的进程/线程的序号或标识符。



## 并行计算机和程序设计模型的分类

- ◆ Flynn经典分类法——SISD/SIMD/MISD/MIMD
- ◆ 内存结构上分类——共享内存系统、分布式内存系统
- ◆ 程序设计模型分类——共享内存编程模式、分布式内存编程模式

- 并行编程模型作为一个抽象存在于硬件和内存架构之上。
- 在一般情况下，编程模型的设计与计算机体系结构相匹配。
  - ✓ 共享内存程序模型为共享内存机器
  - ✓ 消息传递程序模型为分布式内存机器
- 但是编程模型不受机器或内存体系结构的限制：
  - ✓ 消息传递模型可以在共享内存机器上支持：例如，MPI 可以在单个服务器上。
  - ✓ 共享内存模型也可以在分布式内存机器上：例如，划分全局地址空间的语言可以在分布式系统上。

下列计算机中哪个不属于分布式内存系统的计算机/平台：

- ☐ A 集群
- ☐ B 超级计算机
- ☒ C 冯诺依曼计算机
- ☐ D 数据中心

下列计算机中哪个不属于共享内存系统的计算机：

- ☐ A 冯诺依曼计算机
- ☐ B 笔记本电脑
- ☒ C 超级计算机
- ☐ D Lenove台式机

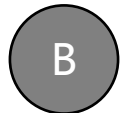
经典Flynn分类按照指令和数据将计算机体系结构分为SISD、SIMD、[填空1]和[填空2]。  
并行程序设计模型分为[填空3]和[填空4]。

经典Flynn分类按照指令和数据将计算机体系结构分为SISD、SIMD、[填空1]和[填空2]。  
并行程序设计模型分为[填空3]和[填空4]。

共享内存系统上能够运行分布式内存编程模式  
编写的并程序序



正确



错误

□分布式内存系统上只能运行分布式内存编程模式编写的并程序序。

- ☐ A 正确
- ☒ B 错误



□在分布式内存和共享内存系统中，只有0号进程/线程能够访问stdout和stderr。

- ☐ A 正确
- ☒ B 错误

□冯诺依曼计算机属于SISD架构的计算机。

A

正确

B

错误

□ 分布式内存系统属于MIMD架构的计算机系统。

A

正确

B

错误

请给出共享内存系统和分布式内存系统的定义，并举出对应的计算机有哪些？

**高性能计算不仅仅是人工智能  
高性能计算决定未来**



**结束!**

---