

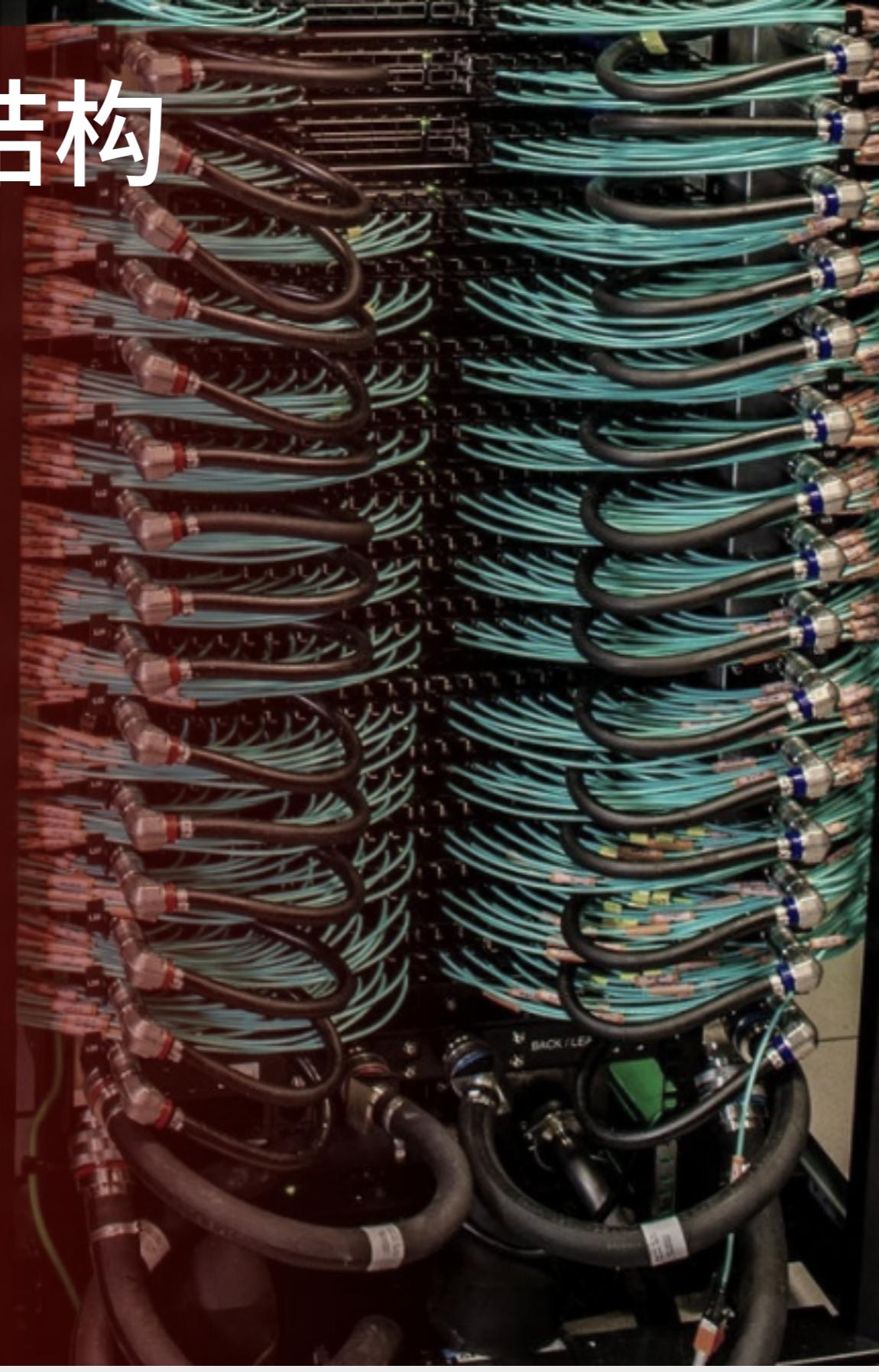
# 计算机组成和体系结构

## 第十三讲

四川大学网络空间安全学院

2020年5月25日

封面来自mantra.ai



# 版权声明

---

课件中所使用的图片、视频等资源版权归原作者所有。

课件原创内容采用 [创作共用署名—非商业使用—相同方式共享4.0国际版许可证\(Creative Commons BY-NC-SA 4.0 International License\)](#) 授权使用。

Copyright@四川大学网络空间安全学院计算机组成与体系结构课程组，2020



# 上期内容回顾

---

- 操作系统
  - 操作系统的发展、分类和标志技术
- 保护环境
  - 虚拟机、子系统和逻辑分区
  - XEN、KVM、AIX和Docker, Intel VT-x和VT-d
- 编程工具
  - 汇编器、加载和链接器, 编程语言, 编译过程, JVM
- 数据库工具
  - 数据库视图, 数据库事务

# 本期学习目标

---

- RISC指令集与CISC指令集
- 费林分类(Flynn's Taxonomy)
- 并行计算及多处理器体系结构
  - 超标量体系结构，超长指令字体系结构，矢量体系结构
  - 处理器/内存互连网络
  - 统一内存访问（UMA）与非统一内存访问（NUMA）
- 其它并行处理体系结构：
  - 数据流计算、神经网络和脉动阵列

# 中英文缩写对照表

英文缩写	英文全称	中文全称
CISC	Complex Instruction Set Computer	复杂指令计算机
MPP	Massive Parallel Processor	大规模并行处理
NUMA	Non-uniform Memory Access	非统一内存访问
RISC	Reduce Instruction Set Computer	精简指令集计算机
RPC	Remote Procedure Call	远程过程调用
SMP	Symmetric Multiprocessor	对称多处理器
UMA	Uniform Memory Access	统一内存访问
VLIW	Very Long Instruction Word	超长指令字体系结构



# 可选的体系结构

# RISC与CISC指令集

---

- RISC全称为Reduced Instruction Set Computer，精简指令集计算机
- CISC全称为Complex Instruction Set Computer，复杂指令集计算机
- 最重要的区别是RISC采用**等长指令**，且**各指令的执行周期数相同**，CISC采用**变长指令**，且**不同指令的执行周期数不同**

# RISC与CISC：设计出发点

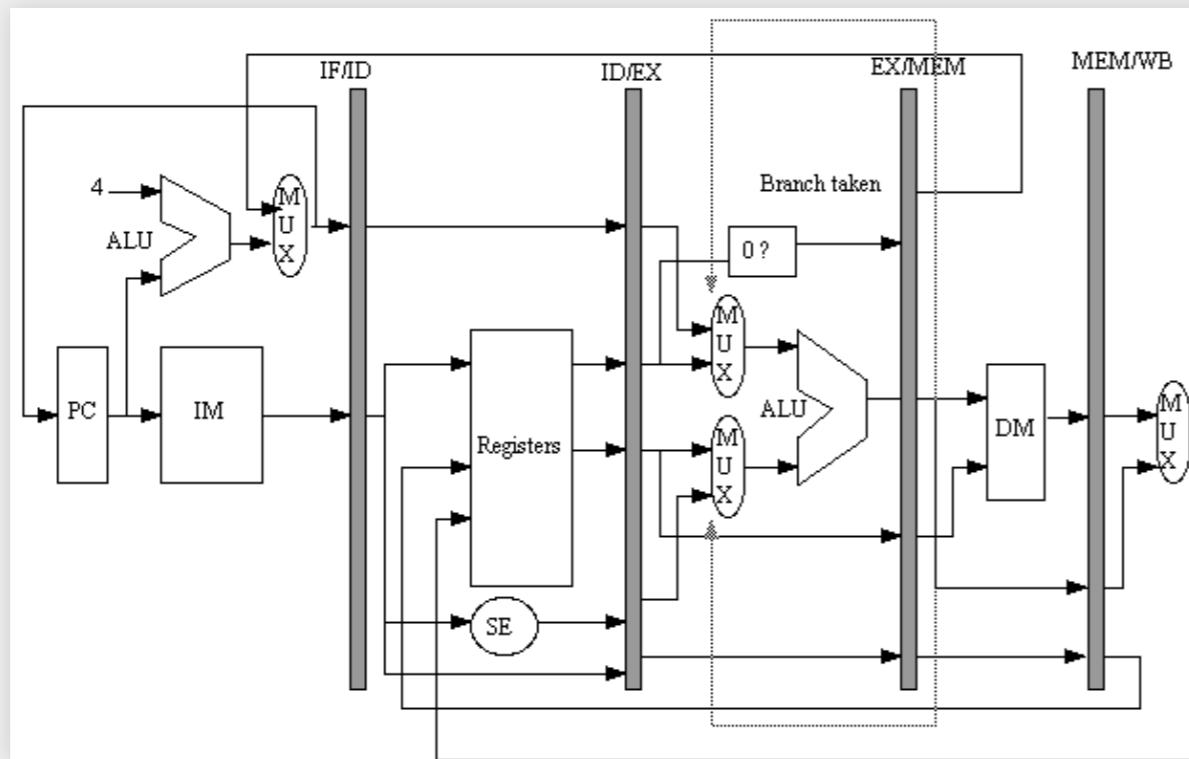
---

$$\text{CPU Time} = \frac{\text{seconds}}{\text{program}} = \frac{\text{instructions}}{\text{program}} \times \frac{\text{avg. cycles}}{\text{instruction}} \times \frac{\text{seconds}}{\text{cycle}}$$

- RISC和CISC选择用不同的方式提高计算性能
- RISC的主要策略是减少每条指令的执行时间
- CISC的主要策略是减少程序执行需要的指令

# RISC和CISC：电路实现

- RISC指令集相对比较简单，因此电路可以采用硬连线实现，并且便于流水线和预测型程序执行的实现



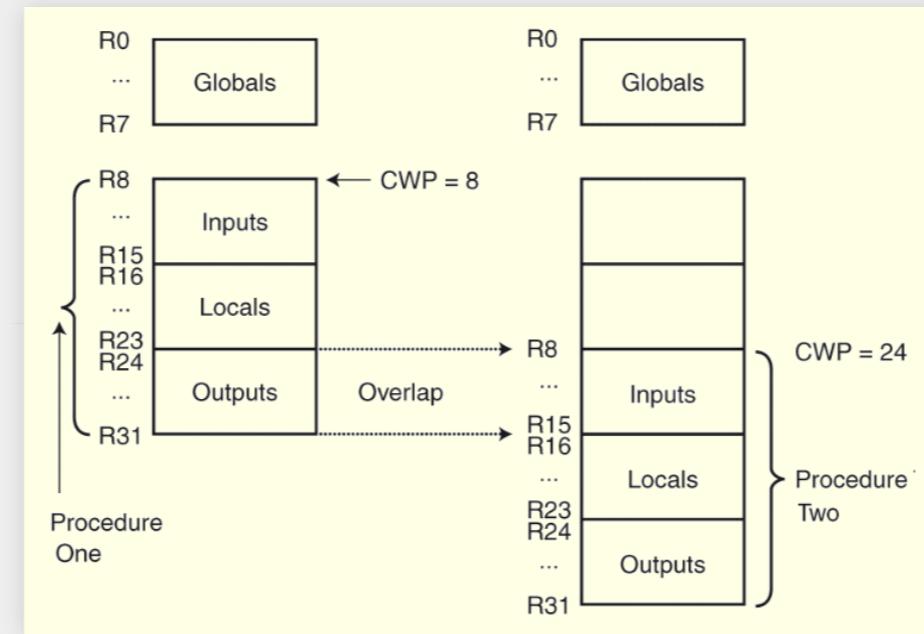
- CISC的电路相对复杂，一般需要采用微码进行控制

# RISC：寄存器优化

---

- 由于RISC指令集中**只有LOAD和STORE指令**可以访问内存，因此为了提高计算速度，通常支持较多的寄存器
- 寄存器除了保存中间计算结果，提供快速访问之外，还可以优化参数传递
- 子程序调用时，参数不需要存到内存的栈中，而是存储到寄存器中

# RISC：寄存器优化参数传递



- 通过滑动窗口的方式为调用的子程序分配寄存器
- 当前窗口指针(Current Window Pointer, CWP)指向当前的寄存器窗口
- 原程序中保存子程序参数的寄存器与子程序的寄存器重合

# RISC和CISC：对比

---

## RISC

许多寄存器集

一条指令包含最多三个操作数

通过寄存器窗口传递参数

单周期指令，高度流水线化

硬连线电路实现

指令数量少，长度等长

只有LOAD和STORE指令可以访问内存，寻址方法较少

## CISC

一个寄存器集

一条指令通常只有1个或2个操作数

通过内存栈传递参数

多周期指令，流水线化程度低

微代码控制

指令数量多，长度不同

许多指令都可以访问内存，支持多种寻址方法

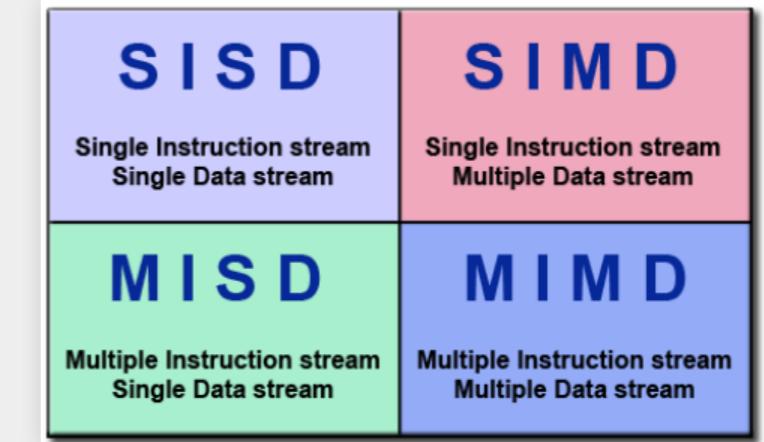
# 体系结构分类

---

- 尽管存在一些局限，费林分类法(Flynn's Taxonomy)是目前流传最广的计算机体系结构分类方法
- 费林分类法考虑了处理器的数量（代表了处理指令的能力）以及处理器中数据通路的数量（代表了同时处理多个数据的能力）
- 费林分类法将计算机按照可以处理指令流的数量和数据流的数量进行分类

# 费林分类

- SISD (Single instruction stream, single data stream)
  - 单指令流、单数据流体系结构
  - 冯诺依曼模型等
- SIMD (Single instruction stream, multiple data stream)
  - 单指令流、多数据流体系结构
  - 矢量机等
- MISD (Multiple instruction stream, single data stream)
  - 多指令流、单数据流体系结构
- MIMD (Multiple instruction stream, multiple data stream)
  - 多指令流、多数据流体系结构
  - 许多并行处理架构属于这种分类



# 费林分类的局限性

---

- MISD分类实际中并没有对应的体系结构
- 并行计算的分类过于笼统，对于MIMD类型的体系结构无法进行明确的分类
- 两类扩展：基于内存访问方式分类，以及基于互连网络拓扑分类

# 基于内存访问方式分类

---

- 对称多处理器(Symmetric Multiprocessor, SMP)和大规模并行处理器(Massive Parallel Processor, MPP)架构的主要区别是处理器访问内存的方式
  - SMP中处理器共享内存，MPP中处理器不共享内存
- 

	SMP	MPP
处理器数量	较少	较多
访问内存方式	共享内存	分布式内存
处理器间通信	通过内存交换数据	通过互连网络交换数据

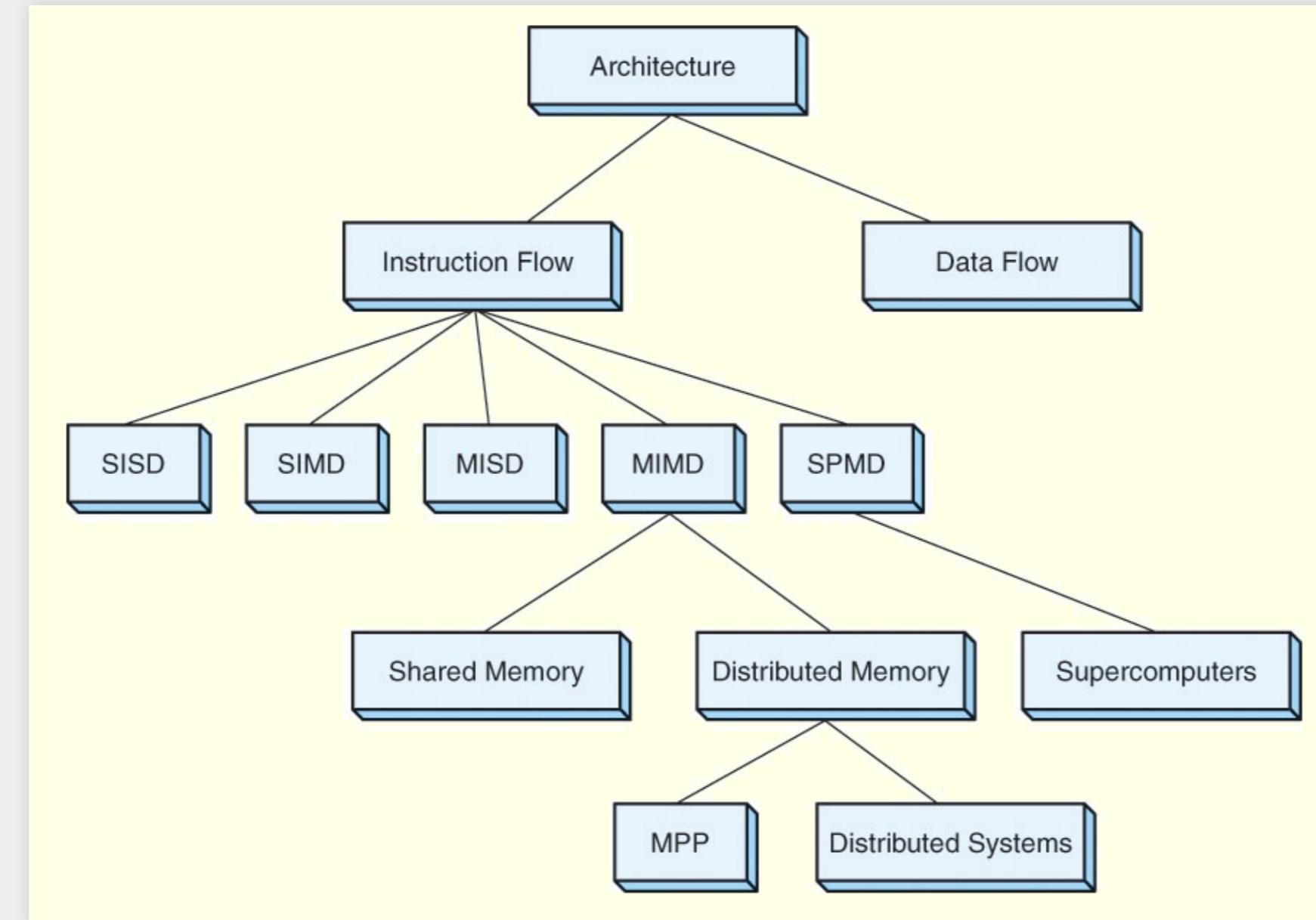
---

# 费林分类：分布式系统扩展

---

- 扩展的费林分类还包括数据流处理器和单程序多数据(Single program, multiple data stream, SPMD)
- SPMD类型的处理器拥有不同的处理器和数据流，尽管采用同一份程序，但是根据处理逻辑是否包括结点相关的信息，可能执行不同的指令

# 扩展后的费林分类



# 并行计算体系结构：设计目的与局限性

---

- 并行计算的目的是通过并行执行指令提高系统的吞吐能力
- 实际中，许多任务无法完全并行计算，依然存在需要串行处理的部分
- 串行处理过程中，其它的处理器处于闲置状态
- 因此，通常情况下，**处理能力**增加n倍不代表**吞吐**也能够增加n倍
- 后面我们将介绍一些并行计算体系结构

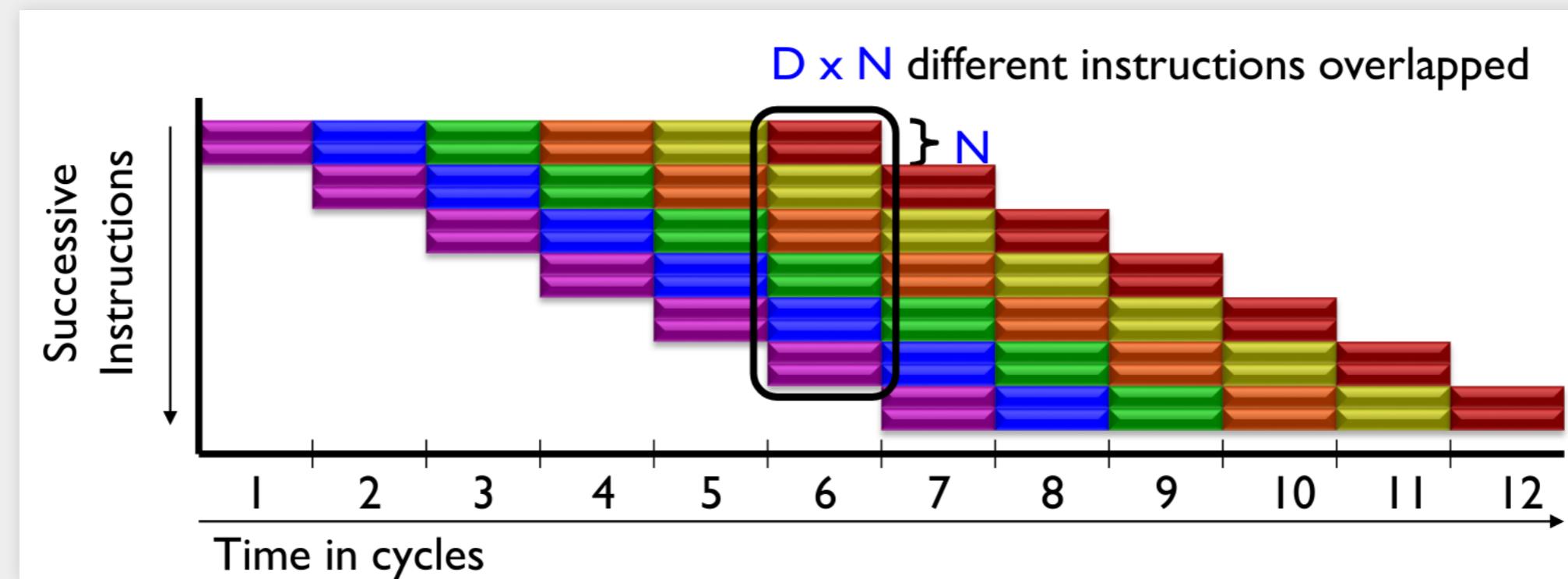
# 并行计算体系结构：超流水线

---

- 之前课上介绍的流水线称为标量流水线
- 每个时钟周期从内存中取一条指令，流水线的每个阶段需要一个周期
- 时钟周期取决于最慢的流水线阶段：  
$$T = \max(T_1, \dots, T_s)$$
- 超流水线 (Super pipelining) : 在一个周期内，通过提高时钟频率，完成多个流水线阶段
- 超流水线可以降低延迟，但是不能提高吞吐
- 如何提高吞吐？

# 并行计算体系结构：超标量计算机

- 超标量流水线：每个时钟周期，每个流水线阶段可以同时执行多条指令



图片来源：<https://compas.cs.stonybrook.edu/~nhonarmand/courses/sp16/cse502/slides/06-pipelining.pdf>

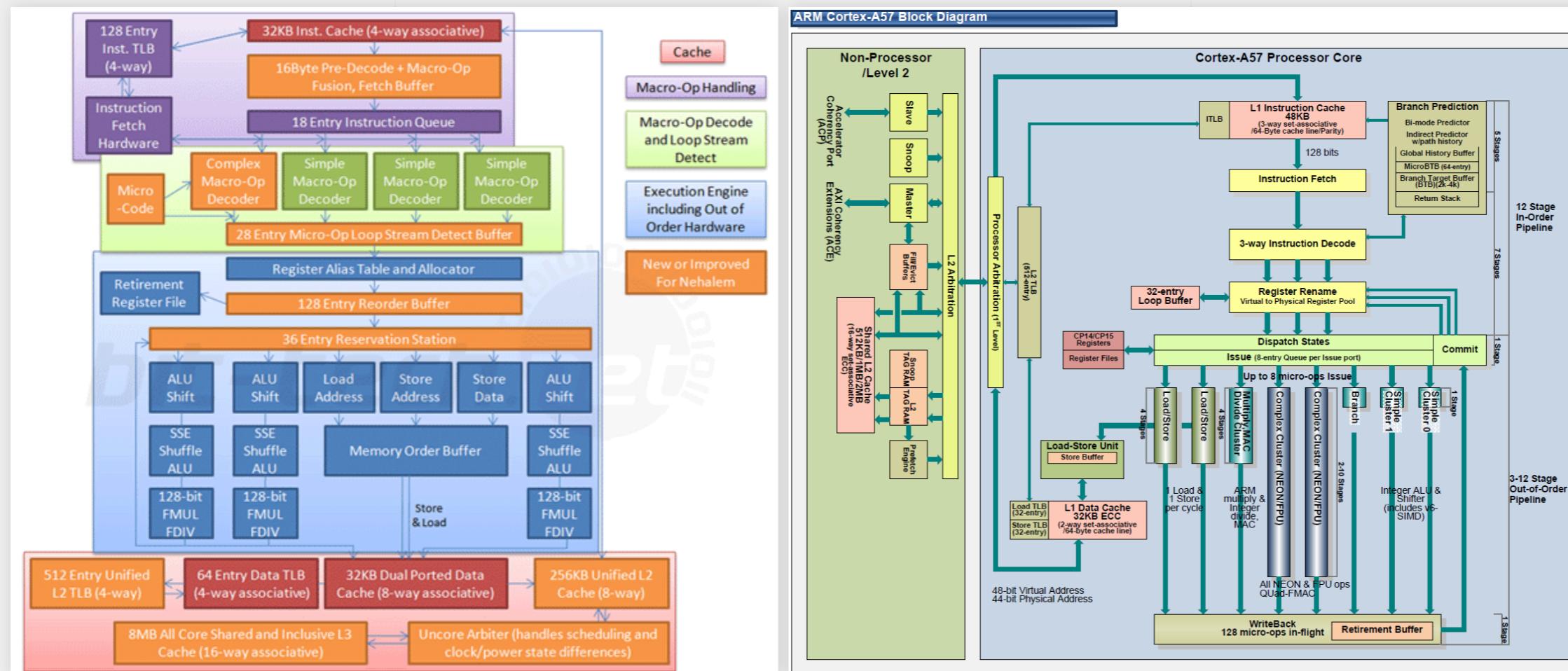
# 并行计算体系结构：超标量计算机的实现

---

- 取指令单元可以同时读取 $N$ 条指令
- 译码器对 $N$ 条指令同时译码，生成对应的微码
- $N$ 发射 $S$ 级流水线代表一个周期可以执行 $N$ 条指令、共有 $S$ 个阶段

# 并行计算体系结构：超标量计算机实例

- Intel Nehalem: 4发射20-24级流水线
- ARM Cortex A57: 3发射15-24级流水线



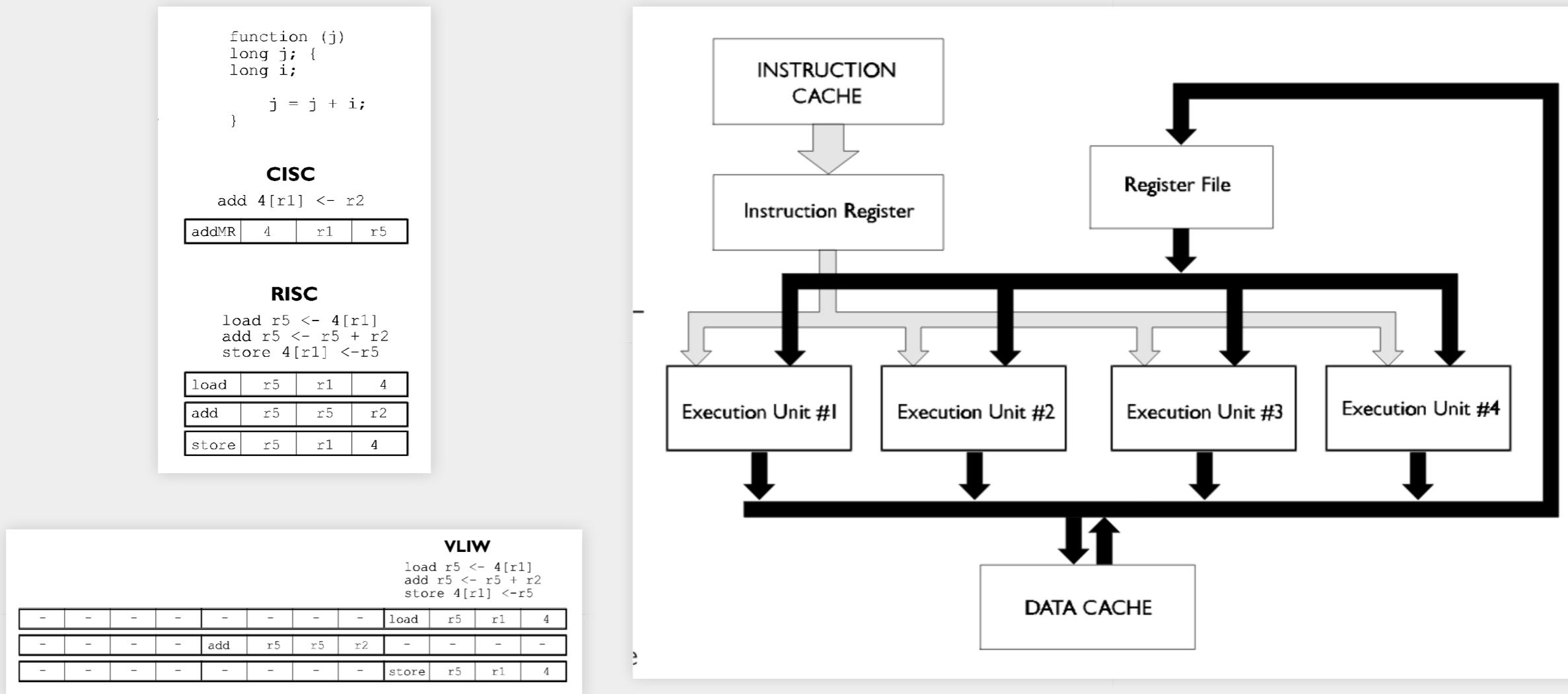
图片来源：[http://www.qdpma.com/CPU/CPU\\_Nehalem.html](http://www.qdpma.com/CPU/CPU_Nehalem.html), <https://www.extremetech.com/computing/199187-first-samsung-cortex-a57-a53-chips-arrive-with-significant-performance-boots>

# 并行计算体系结构：超长指令字计算机

---

- 超标量计算机实际中并不能保证每次都能并行执行 $N$ 条指令：指令之间可能存在竞争导致无法并行
- 超长指令字（Very long instruction word, VLIW）计算机允许一条指令中指定多个操作
- VLIW和超标量计算机都可以实现多发射，二者的区别在于VLIW不依赖硬件进行指令并行组合，因此电路实现更加简单
- VLIW的并行指令组合依靠编译器完成，因此编译器的实现较为复杂

# 并行计算体系结构：VLIW示例



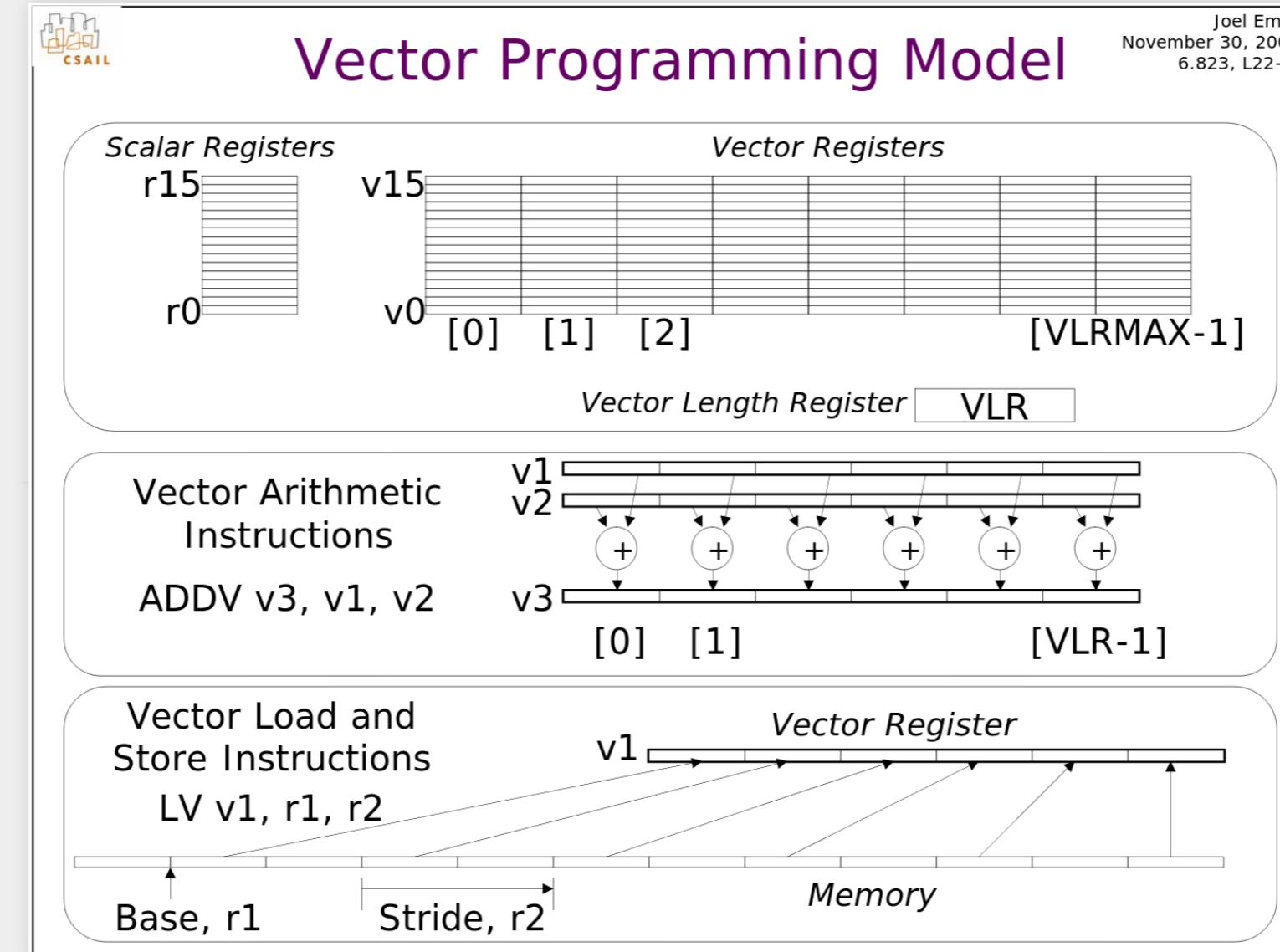
图片来源：[http://twins.ee.nctu.edu.tw/courses/ca\\_08/literature/11\\_vliw.pdf](http://twins.ee.nctu.edu.tw/courses/ca_08/literature/11_vliw.pdf)

# 并行计算体系结构：矢量计算机

---

- 矢量计算机支持对矢量的数据进行操作
- 早期的超级计算机都是矢量计算机
- 矢量计算机按照数据访问方式分为
  - 寄存器到寄存器：所有的操作数都是寄存器
  - 内存到内存：操作数可以直接从内存读取到算术运算单元

# 并行计算体系结构：矢量计算机示例



图片来源：[https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-823-computer-system-architecture-fall-2005/lecture-notes/l22\\_vector.pdf](https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-823-computer-system-architecture-fall-2005/lecture-notes/l22_vector.pdf)

# 并行计算体系结构：矢量计算机示例



Joel Emer  
November 30, 2005  
6.823, L22-9

## Vector Code Example

# C code	# Scalar Code	# Vector Code
<pre># C code for (i=0; i&lt;64; i++)     C[i] = A[i] + B[i];</pre>	<pre>LI R4, 64 loop:     L.D F0, 0(R1)     L.D F2, 0(R2)     ADD.D F4, F2, F0     S.D F4, 0(R3)     DADDIU R1, 8     DADDIU R2, 8     DADDIU R3, 8     DSUBIU R4, 1     BNEZ R4, loop</pre>	<pre>LI VLR, 64 LV V1, R1 LV V2, R2 ADDV.D V3, V1, V2 SV V3, R3</pre>

图片来源：[https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-823-computer-system-architecture-fall-2005/lecture-notes/l22\\_vector.pdf](https://ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-823-computer-system-architecture-fall-2005/lecture-notes/l22_vector.pdf)

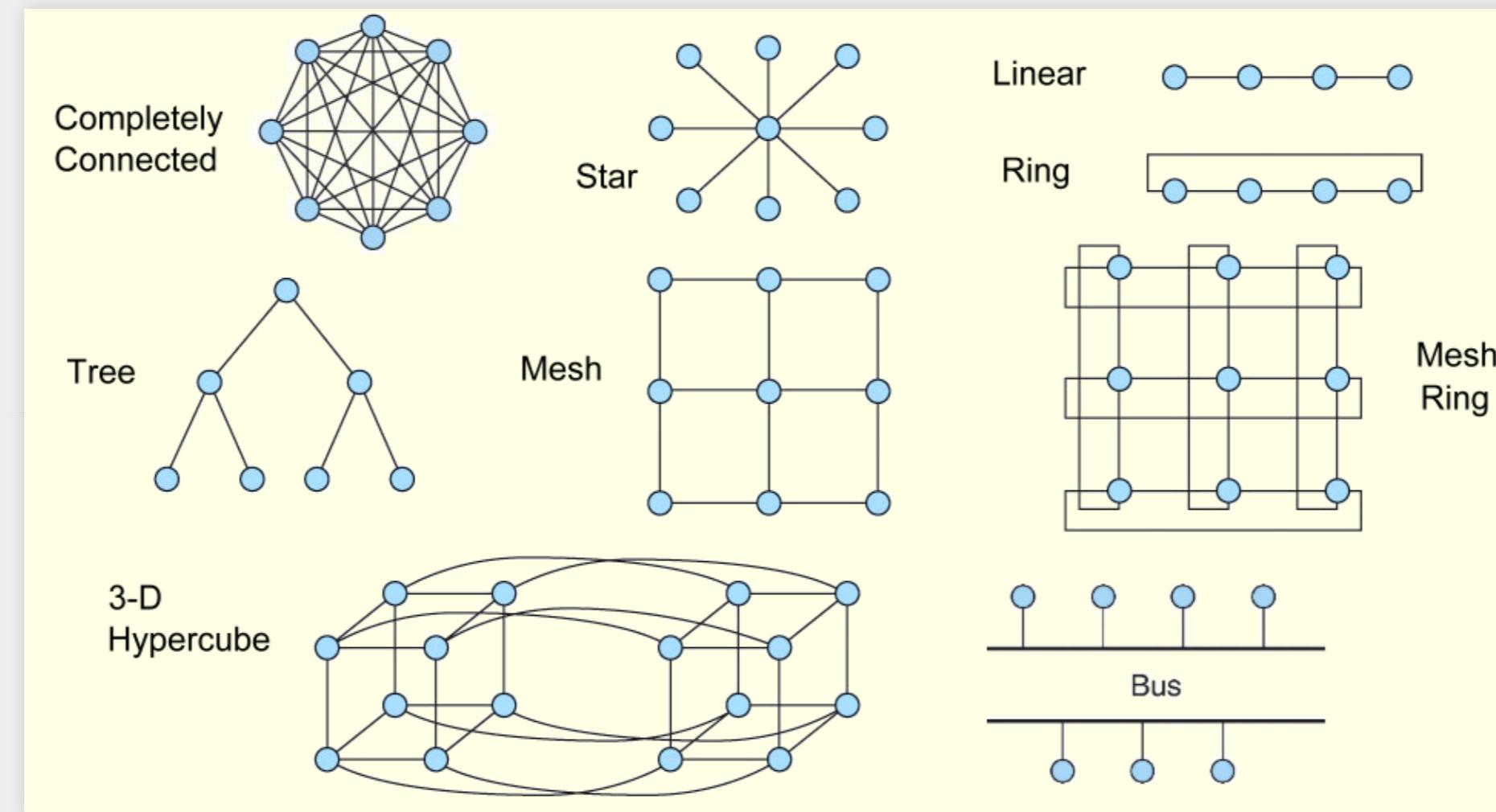
# 互连网络

---

- 多处理器系统的处理器和处理器之间，以及处理器和内存之间需要进行数据交互
- 对于多处理器系统，通常采用互连网络进行数据传输和通信
- 互连网络可以按照不同的拓扑、路由策略和交换策略进行分类

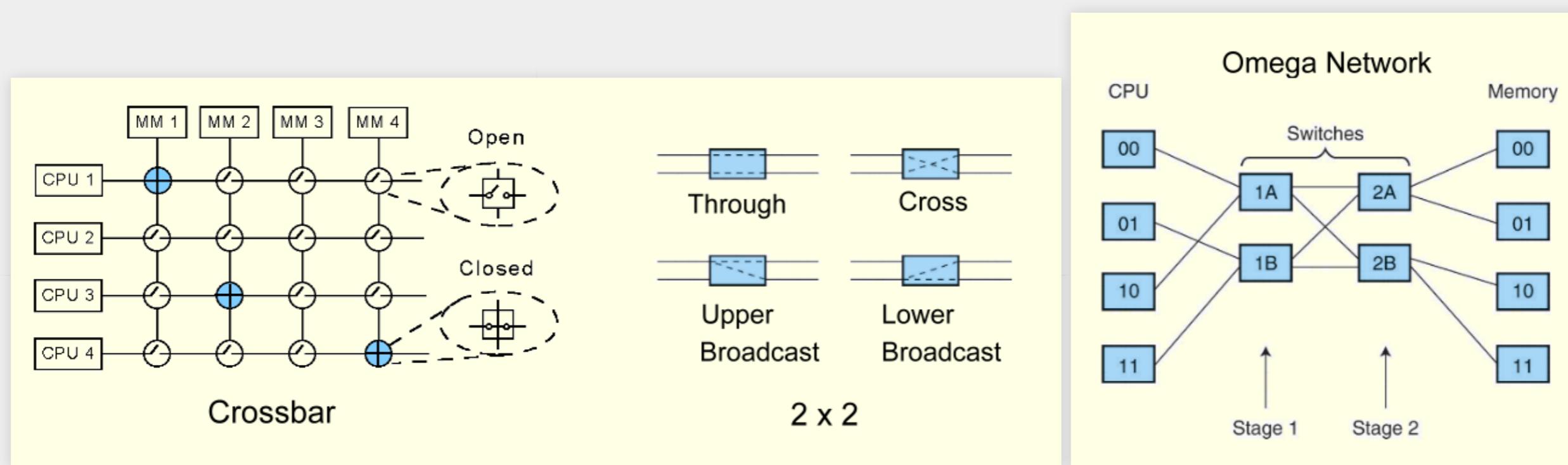
# 常见互连网络拓扑

- 处理器和处理器之间的互连网络通常采用静态互连网络
- 常见的拓扑如下所示



# 互连网络：交换网络

- 处理器和内存之间通常采用动态互连网络
- 例如下面所示的交叉开关(Crossbar)交换网络(switching network)以及多阶段互连网络（也叫做Omega网络）



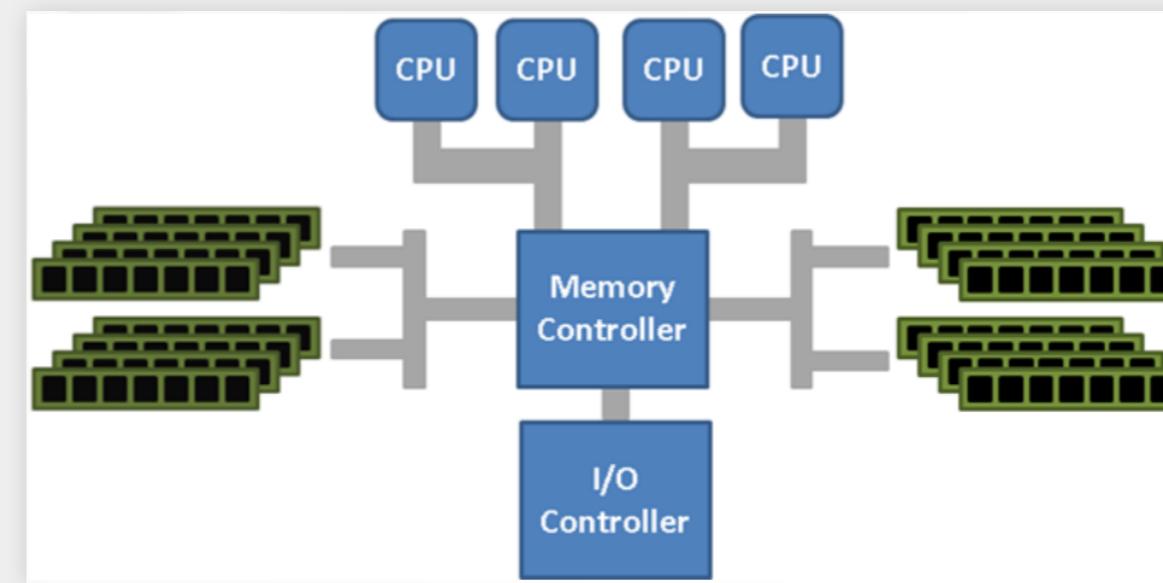
# 互连网络：动态互连网络电路实现对比

---

- 基于总线：比较经济，但是可能成为多处理器传输瓶颈；并行总线可以提高传输性能，但是成本较高
- 交叉开关支持无阻塞访问（不同处理器可以同时进行数据传输），但是连接 $n$ 个对象需要 $n^2$ 个开关
- Omega网络是阻塞的，但是相比总线传输竞争较少，实现比交叉开关更经济，连接 $n$ 个对象需要  $\log_2 n$  个阶段，每个阶段只需要  $\frac{n}{2}$  个开关

# 统一内存访问

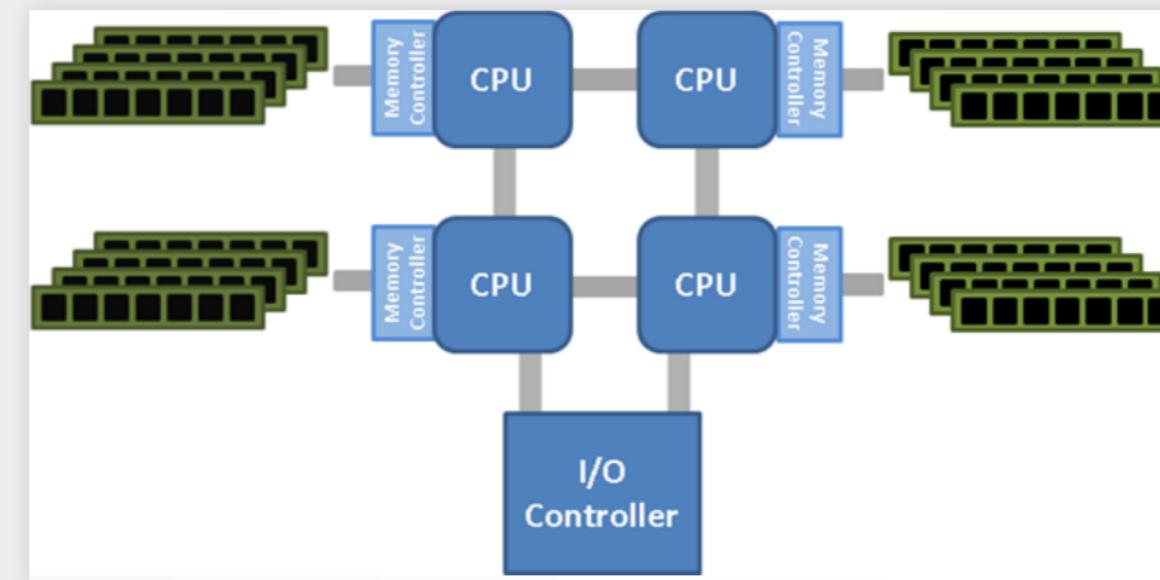
- 对于共享内存式的MIMD体系结构，按照访问内存的方式可以分为两类
- 统一内存访问（Uniform memory access, UMA）：所有的处理器访问所有的内存时间都是相同的



图片来源：<https://www.sqlskills.com/blogs/jonathan/understanding-non-uniform-memory-accessarchitectures-numa/>

# 非统一内存访问

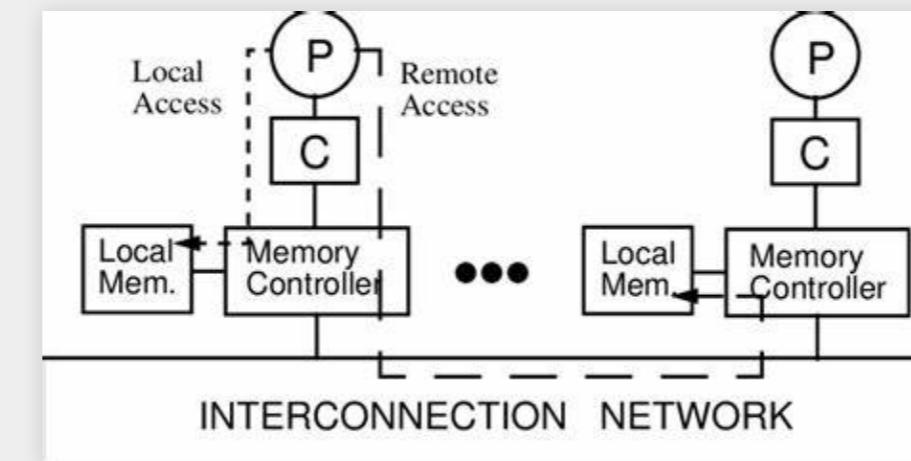
- 非统一内存访问 (Non-uniform memory access, NUMA) : 每个处理器有自己的内存，访问自己的内存比访问其它处理器的内存速度更快
- 除了内存之外，每个处理器还可能有属于自己的缓存，因此必须解决缓存一致性的问题



图片来源：<https://www.sqlskills.com/blogs/jonathan/understanding-non-uniform-memory-accessarchitectures-numa/>

# 非统一内存访问：缓存一致性

- 为了保证非统一内存访问中各处理器的缓存一致性，通常会引入缓存控制器
- 保证了缓存一致性的NUMA系统称为CC-NUMA（Cache Coherent NUMA）



# 非统一内存访问：缓存写协议

---

- 缓存直写策略 (Write-through) : 对一个处理器的缓存修改同步到所有处理器的内存中
- 缓存直写更新 (Write-through with update) : 更新缓存时将更新的内容发送给每个处理器
- 缓存直写失效 (Write-through with invalidate) : 更新缓存时通知其它处理器地址内容失效
- 直写更新策略会产生更大的数据传输量，因为更新的地址和内容都需要发送给每个处理器，但是可以保证各处理器的缓存内容总是最新的
- 直写失效策略需要更少的带宽，因为只需要发送更新的地址，但是访问新的数据需要重新读取内存

# 非统一内存访问：缓存写协议

---

- 缓存写回策略 (write-back) : 只有缓存换出的时候才写入内存
- 写回的时候写缓存的处理器需要获得独占权，其它所有处理器中的缓存块会被标记为失效

# 分布式计算和云计算

---

- 多处理器系统都可以认为是分布式的
- 但是术语分布式计算通常指通过网络连接来自不同计算机的处理器和内存
- 和多处理器系统相比，分布式计算的处理器和内存的连接方式更加松散
- 分布式系统通过远程过程调用（Remote Procedure Call, RPC）实现远程数据的访问和指令执行

# 非冯诺依曼计算模型

---

- 上述并行处理依然基于冯诺依曼模型：指令的取-译码-执行、数据保存在内存中
- 在一些特殊问题中，采用非冯诺依曼模型可以提高计算的性能
- 主要有三类常见的非冯诺依曼模型：数据流模型、神经网络和脉动阵列

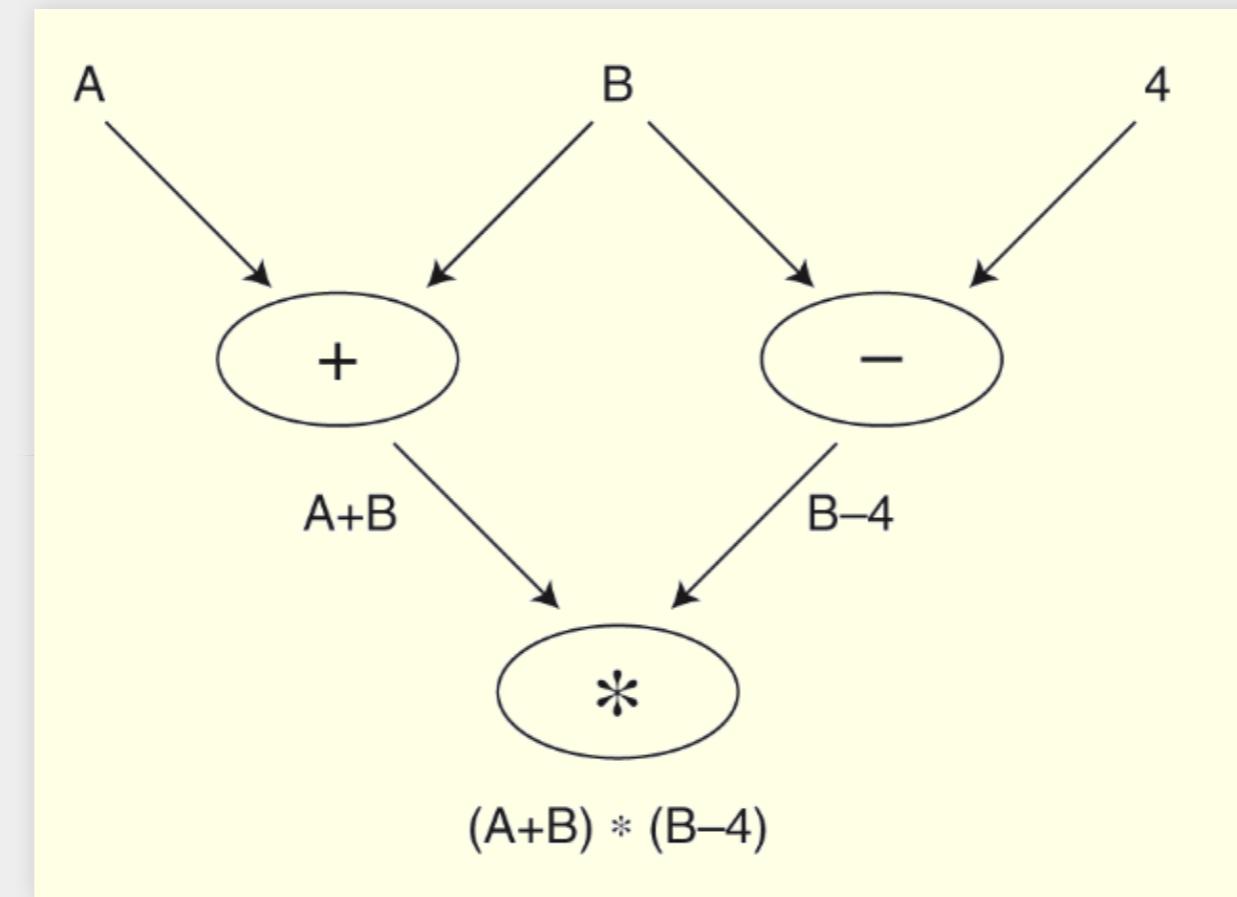
# 数据流计算

---

- 数据流计算通常具有多个可以互相通信的计算单元 (Process Element)
- 每个计算单元通常具有两个单元：激活单元 (Enabling Unit) 和功能单元 (Function Unit)
- 按照激活单元的工作机制，一般分为静态数据流和动态数据流

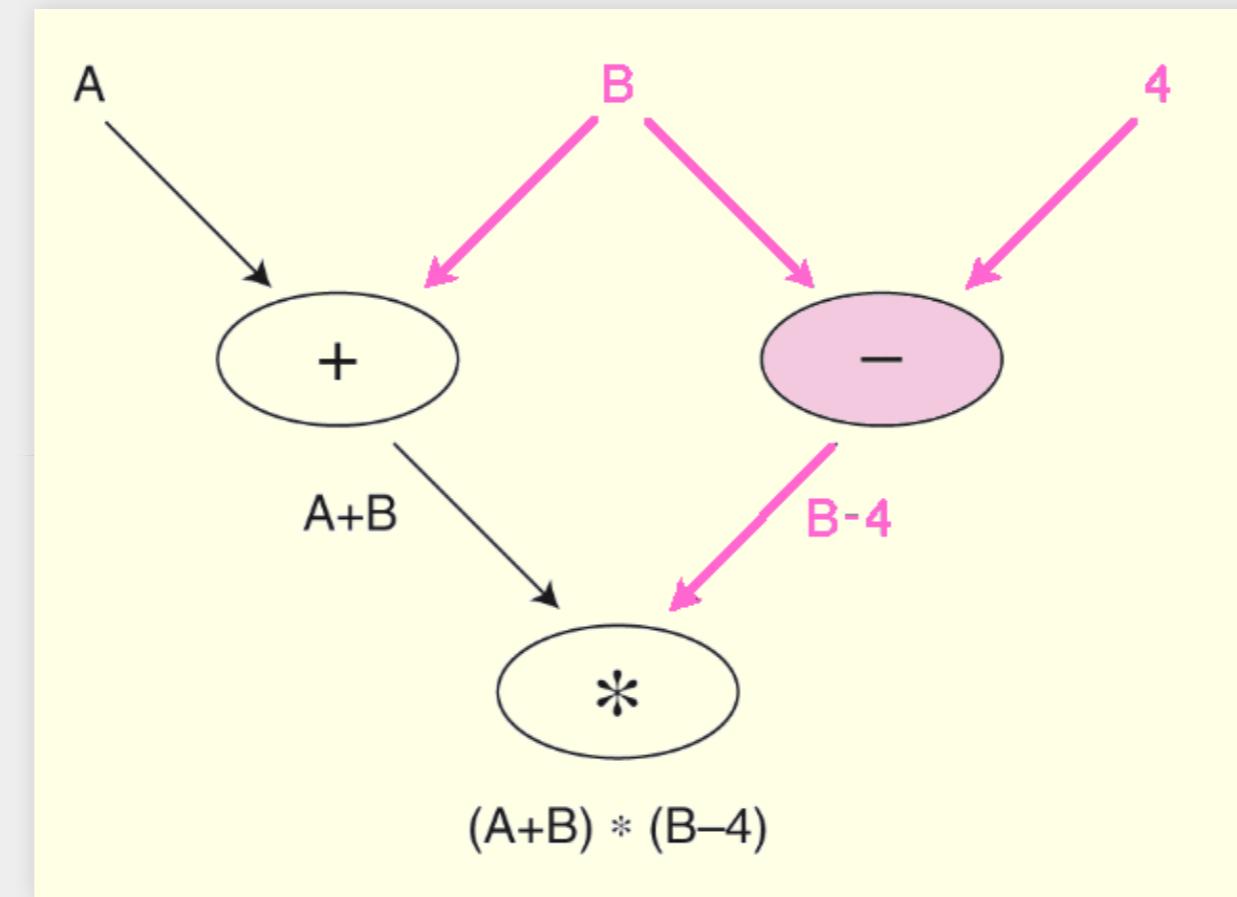
# 静态数据流

- 数据流构成一个流水线
- 每个时钟周期完成读取输入、计算、写输出的操作
- 时钟信号作为激活信号



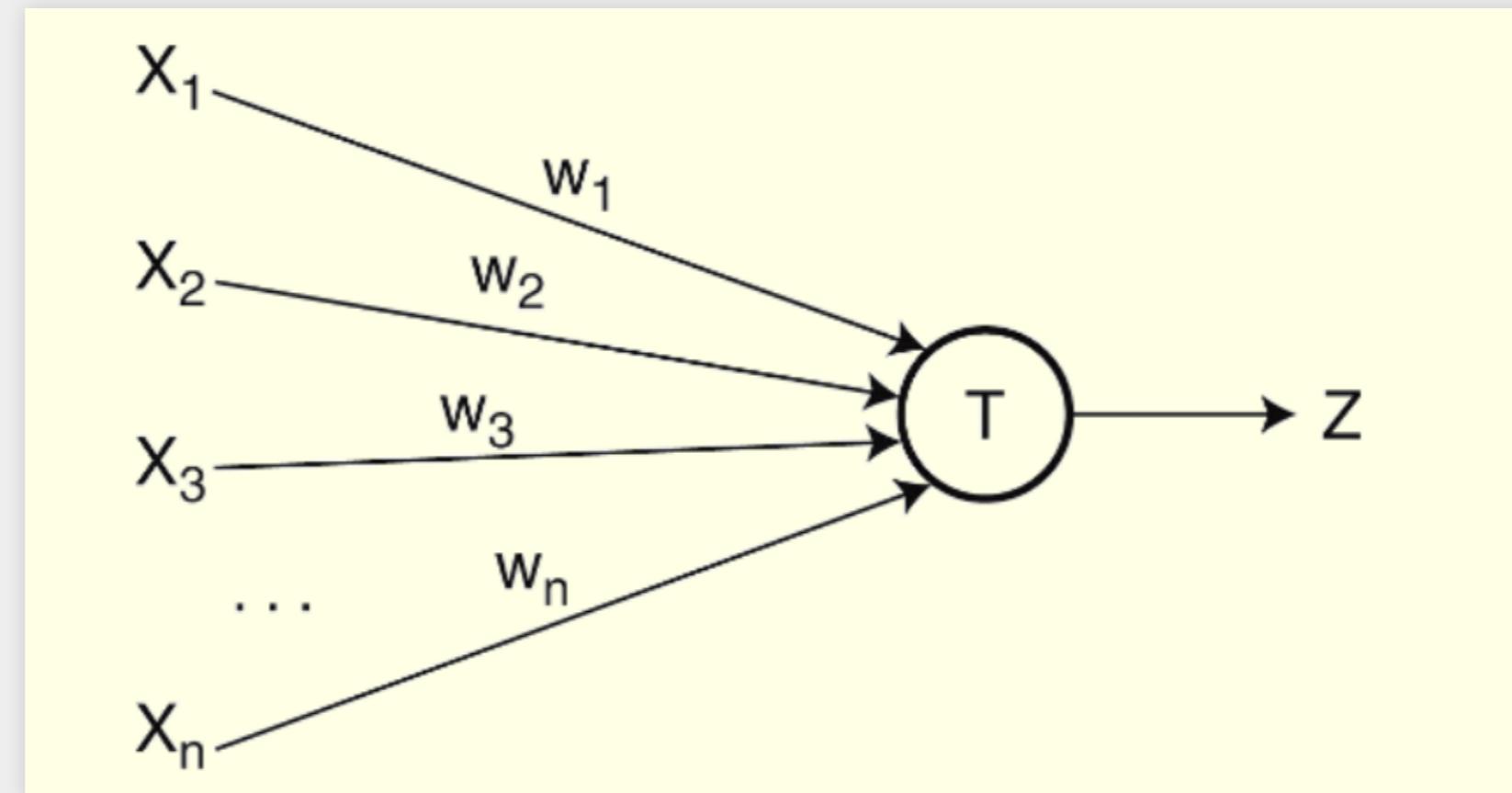
# 动态数据流

- 静态数据流周期取决于速度最慢的单元
- 动态数据流允许不同单元执行时间不同
- 通常采用数据令牌的方式

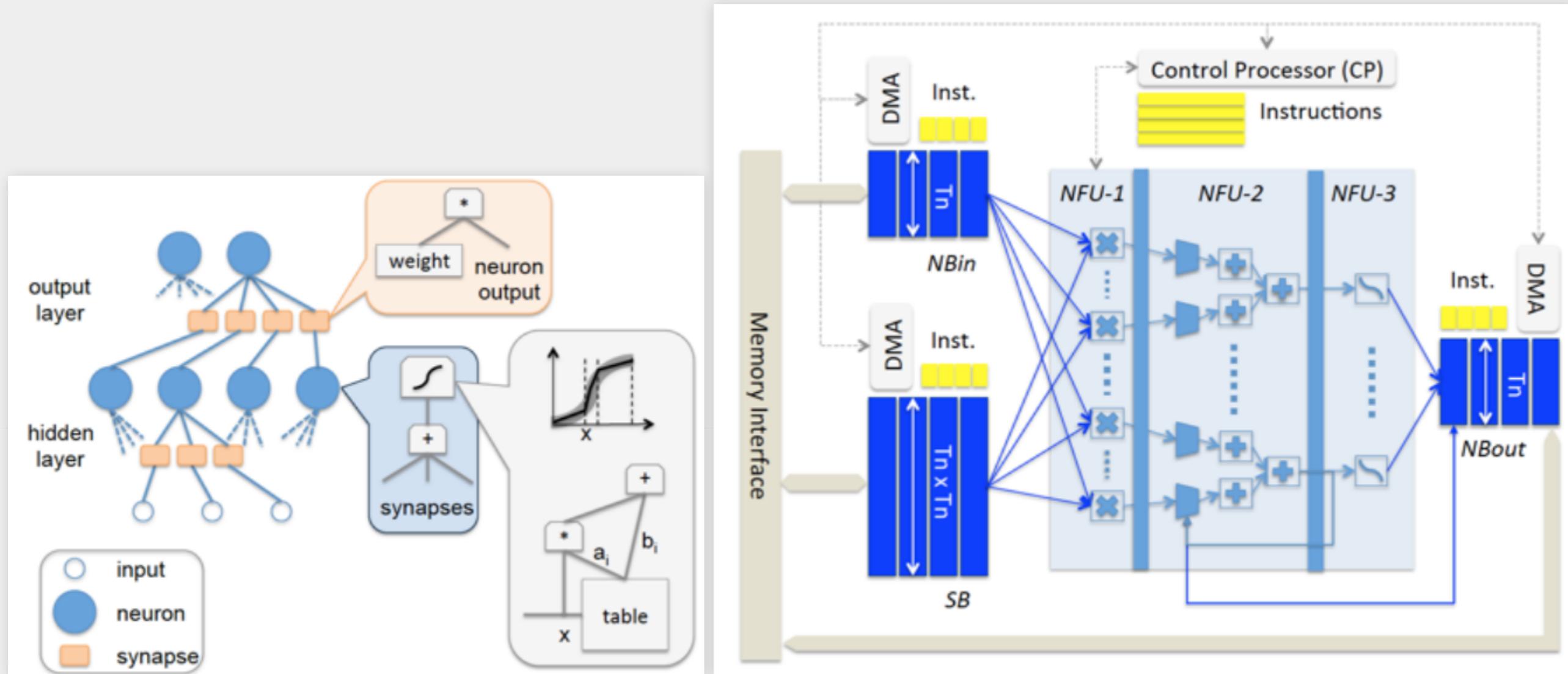


# 神经网络

- 神经网络首先是一种计算范式
- 模拟生物体内神经元的工作原理
- 例如感知器 (perceptron)



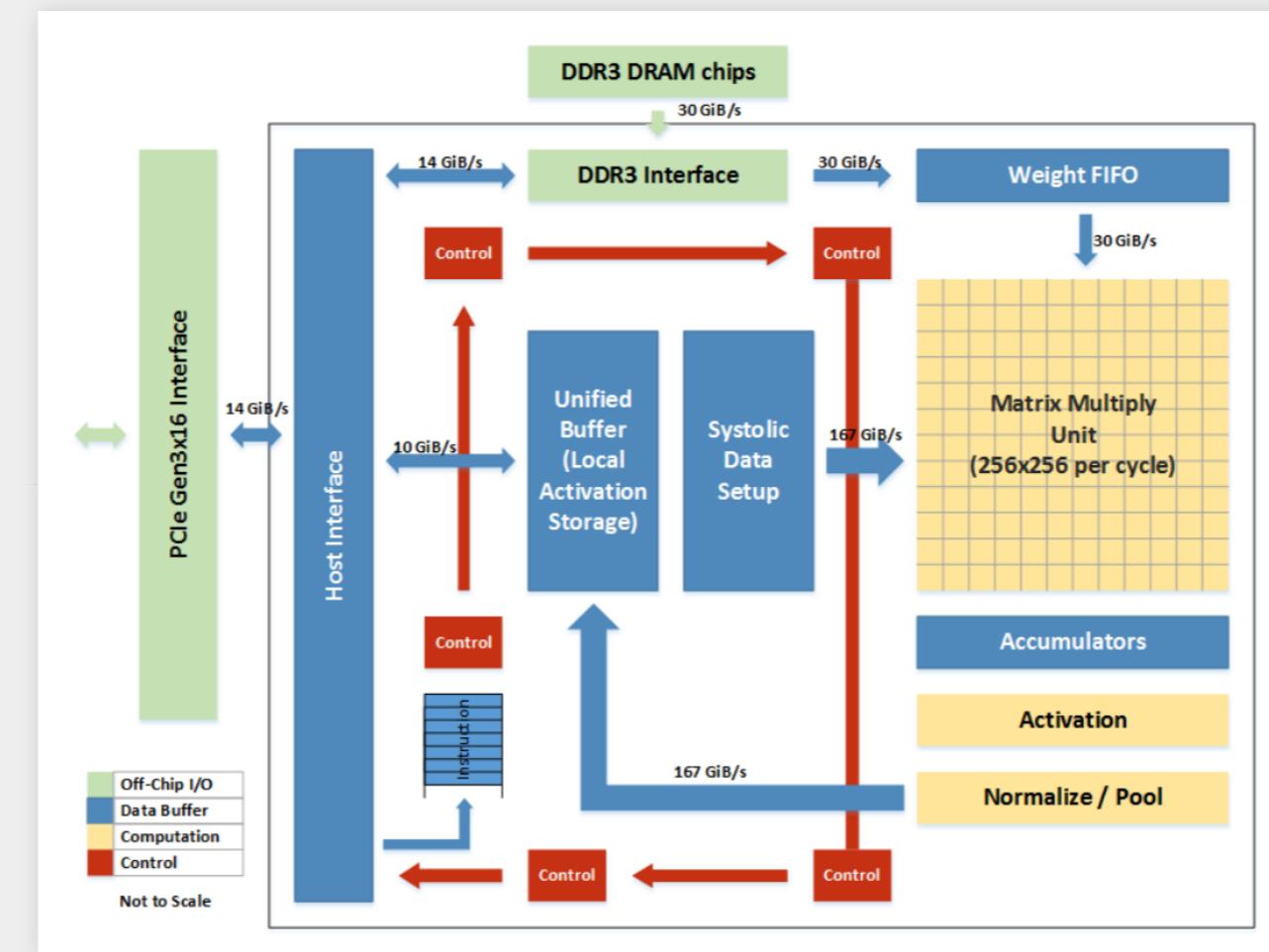
# 神经网络的实现



图片来源：Tianshi Chen. DianNao: A Small-Footprint High-Throughput Accelerator for Ubiquitous Machine-Learning. ASPLOS, 2014.

# 脉动阵列

- 脉动阵列也可以看作是数据流计算的一种
- 减少内存访问，将计算结果在计算单元之间传输



图片来源: <https://zhuanlan.zhihu.com/p/26522315>



第十三讲结束

# 本期内容总结

---

- RISC指令集与CISC指令集
- 费林分类(Flynn's Taxonomy)
- 并行计算及多处理器体系结构
  - 超标量体系结构，超长指令字体系结构，矢量体系结构
  - 处理器/内存互联网络
  - 统一内存访问（UMA）与非统一内存访问（NUMA）
- 其它并行处理体系结构：
  - 数据流计算、神经网络和脉动阵列

# 扩展阅读

---

- 非统一内存访问：<https://people.eecs.berkeley.edu/~culler/cs258-s99/slides/lec07/001.htm>



# Q & A