复习：

1. CPU的结构：ALU、CU、寄存器、中断系统、通路
2. 内部异常、外部中断、中断处理、中断服务程序
3. 指令周期内各个工作阶段的数据流
4. 系统并行性及指令流水的原理
5. 影响指令流水效率的因素
   - 结构相关
   - 数据相关
   - 控制相关（转移相关）
6. 流水线性能：吞吐率、加速比、效率
7. 多发射流水线：超流水、超长指令字(VLIW)、超标量、乱序(O-o-O)多发射(multi-issue)
8. 指令流水线、运算流水线
9. 指令流水：提高指令级并行性ILP（*Instruction-Level Parallelism*）

Parallelism and GPU

Rao Fu

Xinze Li

May 6, 2019

Content

- Why parallelism?
  - The death of single-core scaling
- Parallel architecture
  - Superscalar
  - Multi-core
  - SIMD
  - Multi-threading
- Why GPU is good at parallel programing?
  - A toy example
  - Tiled matrix multiplication

Why Parallelism?

ILP scaling tapped out

ILP: Instruction-level parallelism

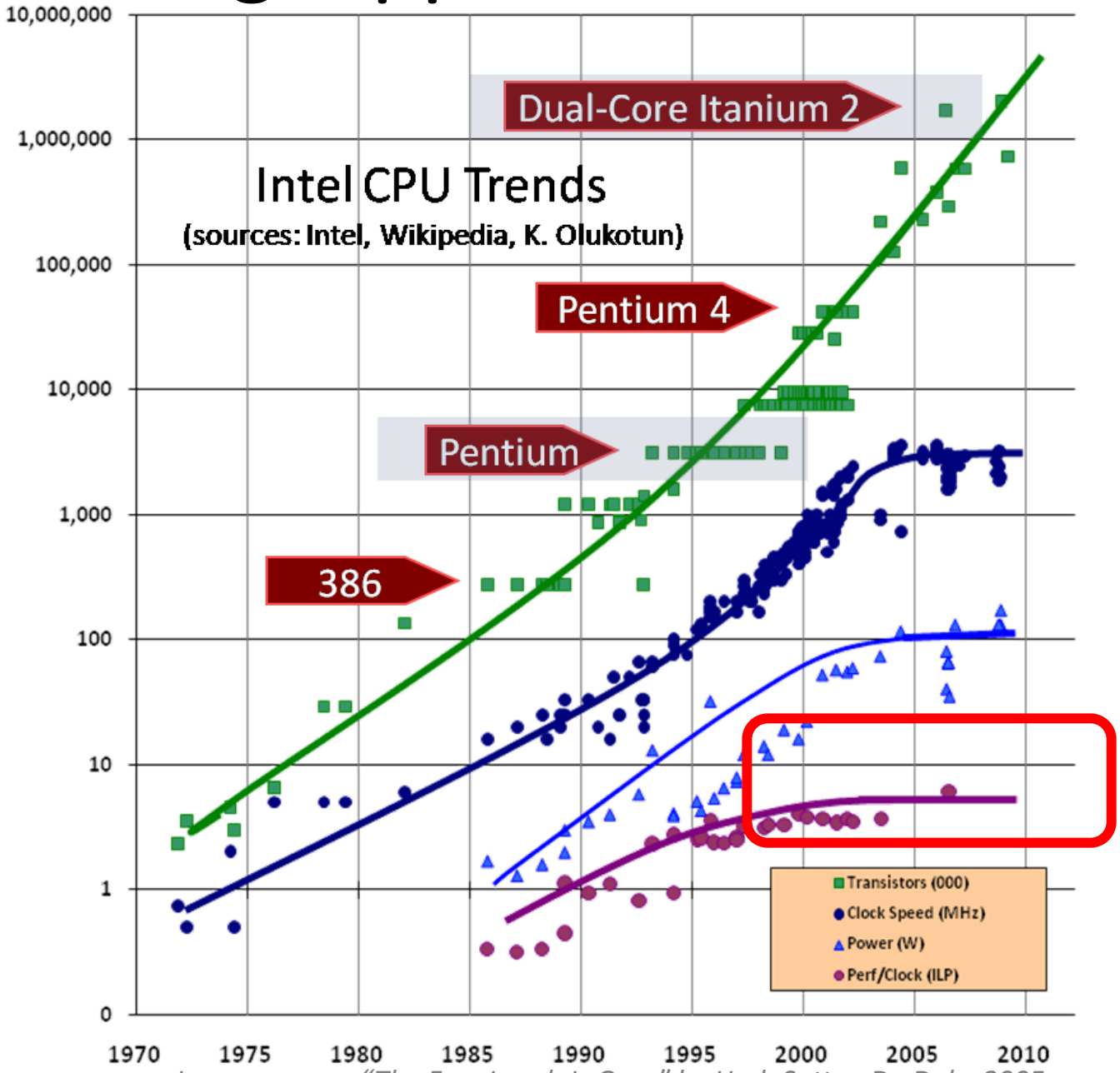


Image source: "The Free Lunch Is Over" by Herb Sutter, Dr. Dobs 2005

Diminishing returns of ILP execution

Most available ILP exploit by processor capable of issuing four instructions per clock

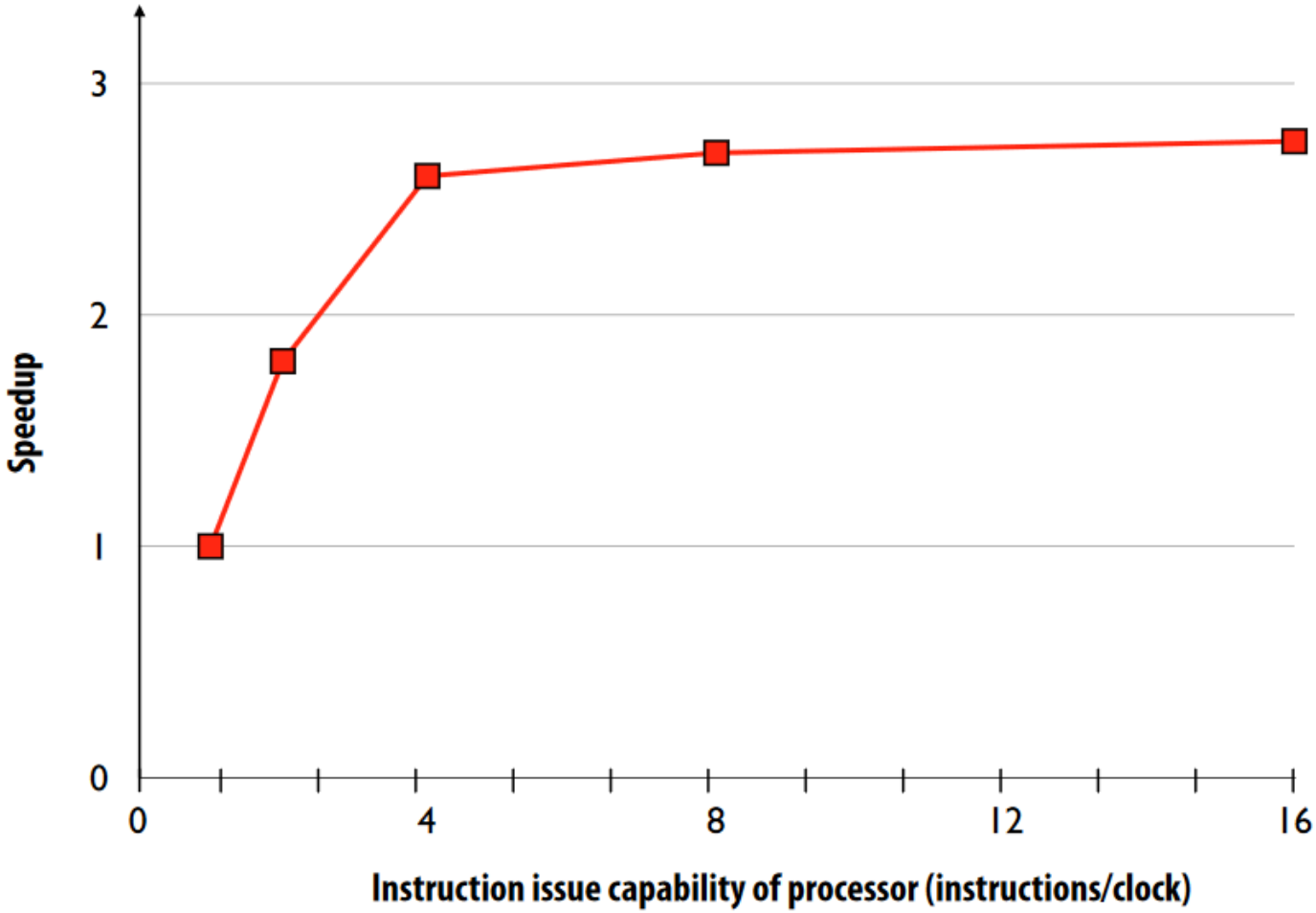
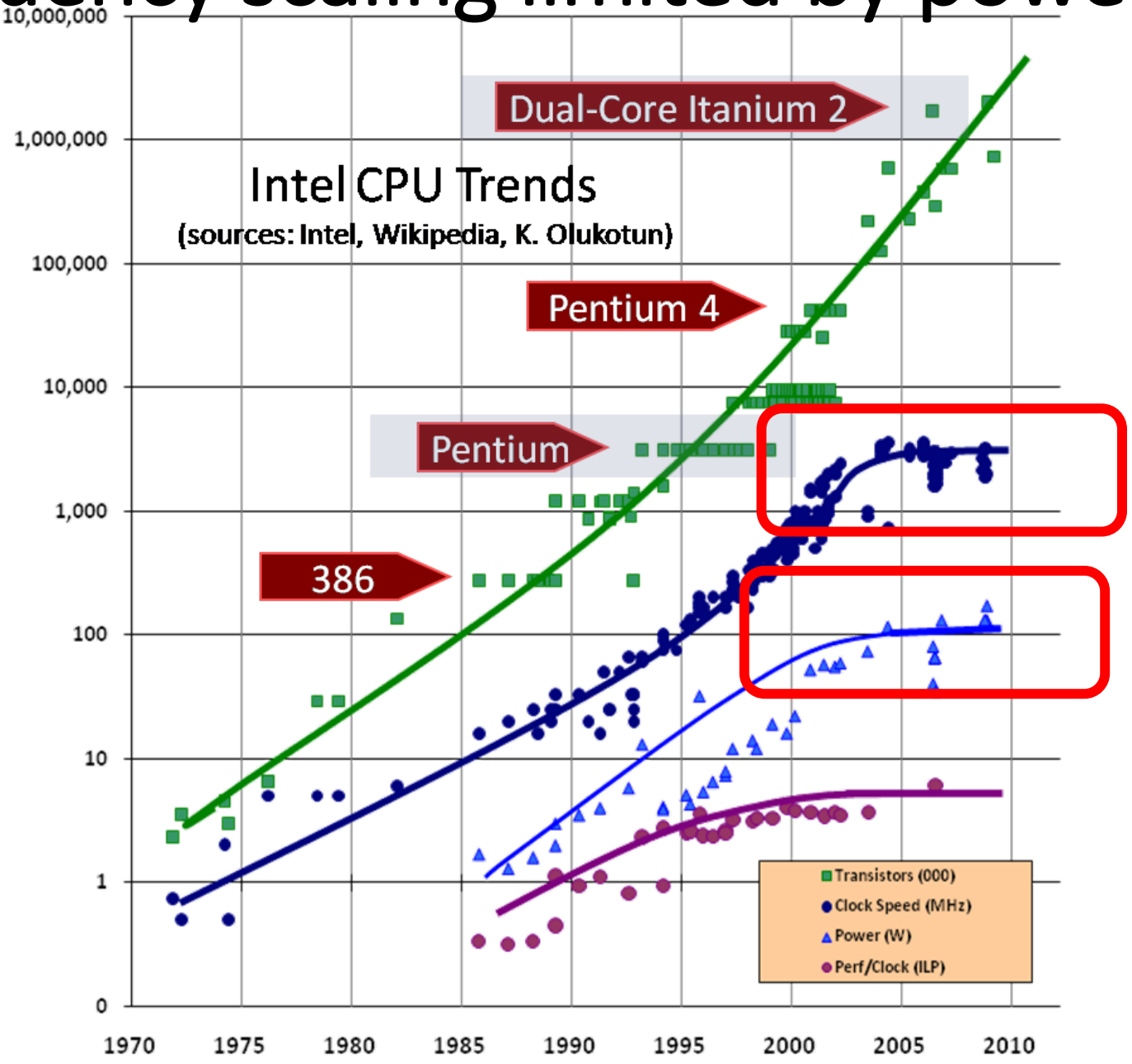
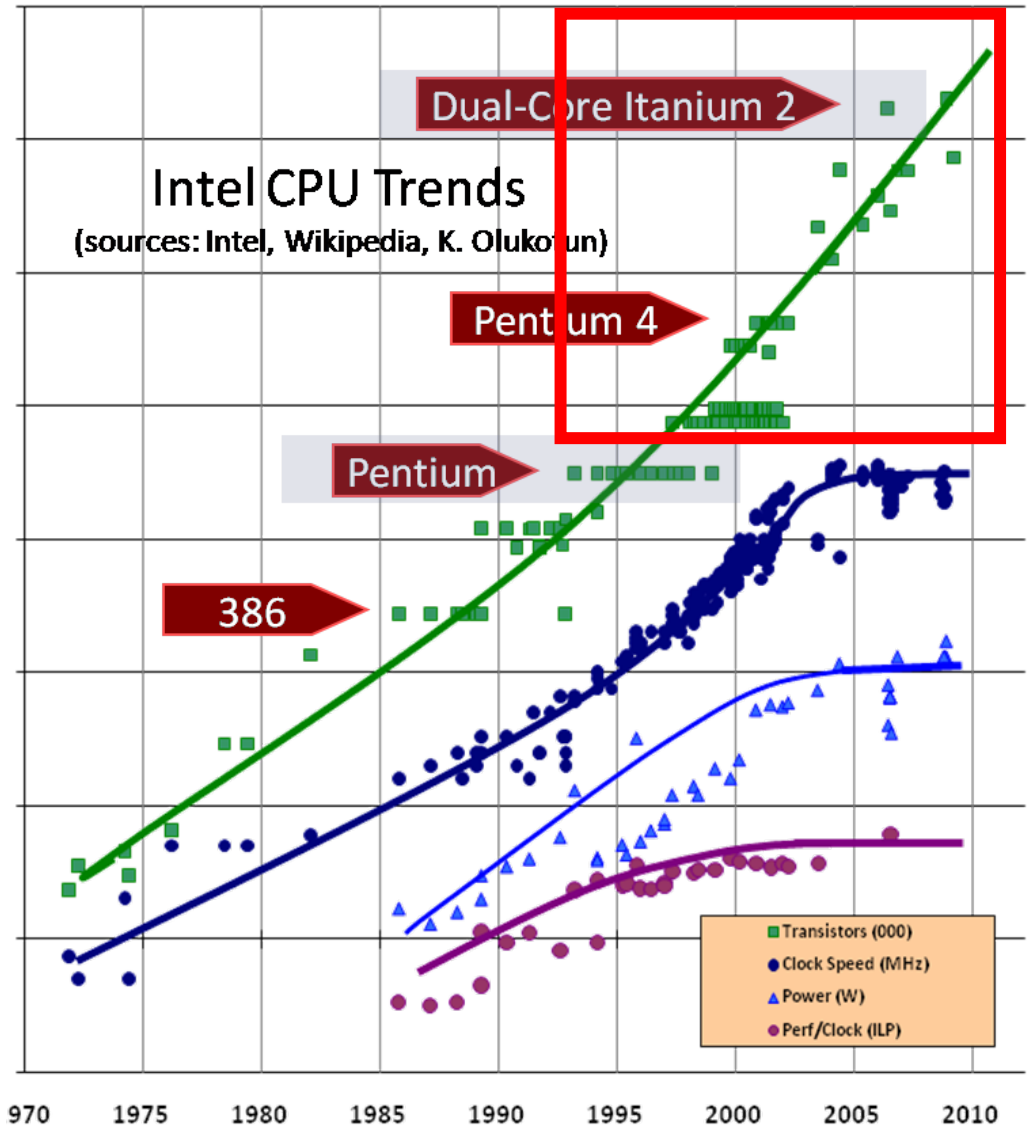


Image source: "Parallel Computer Architecture: A Hardware/Software Approach" by Culler & Singh

Frequency scaling limited by power



The death of single-core scaling



Architects are now building faster processors by adding processing cores.

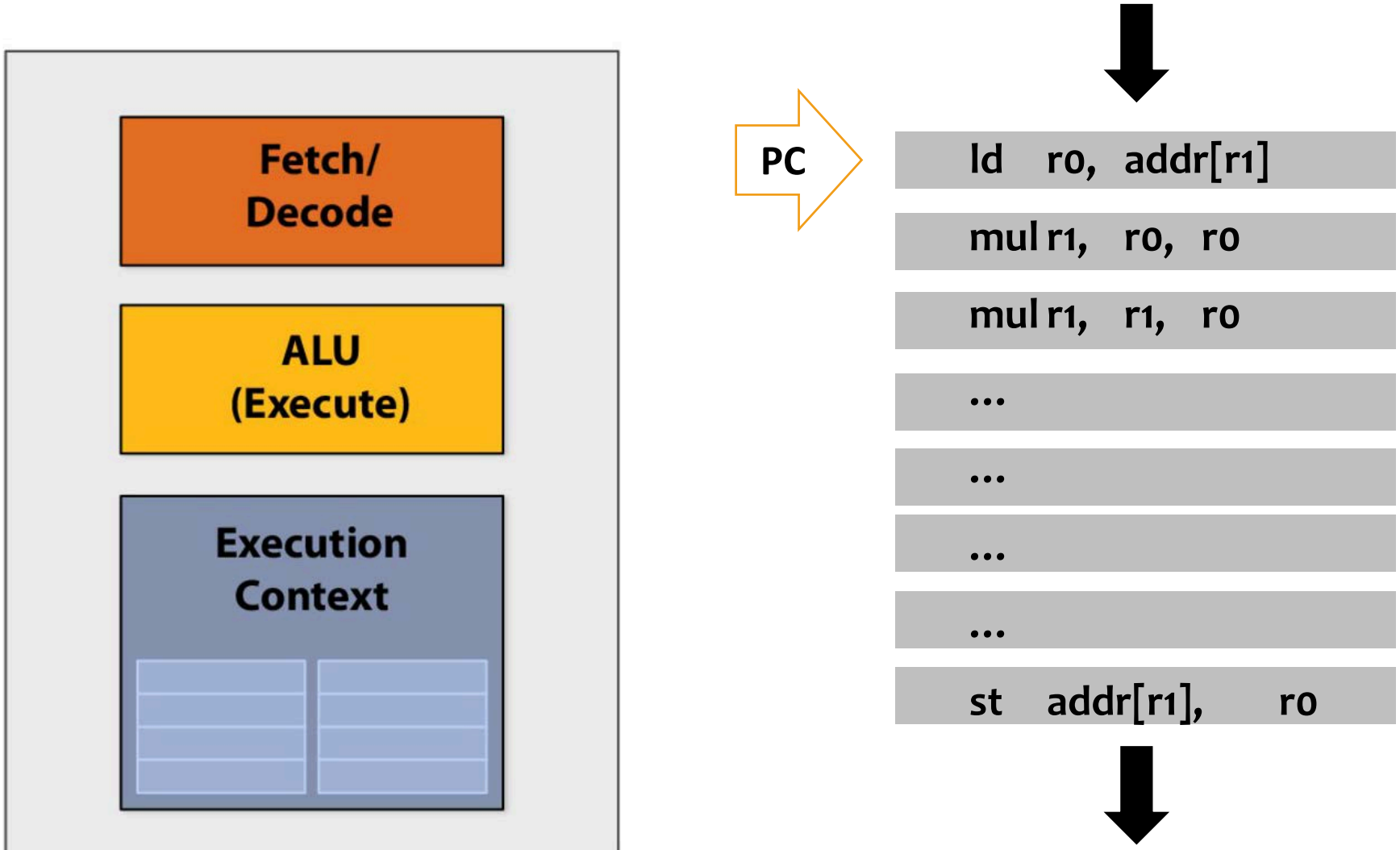
Software must be parallel to see performance gains.

No more free lunch for software developers !

Parallel Architecture

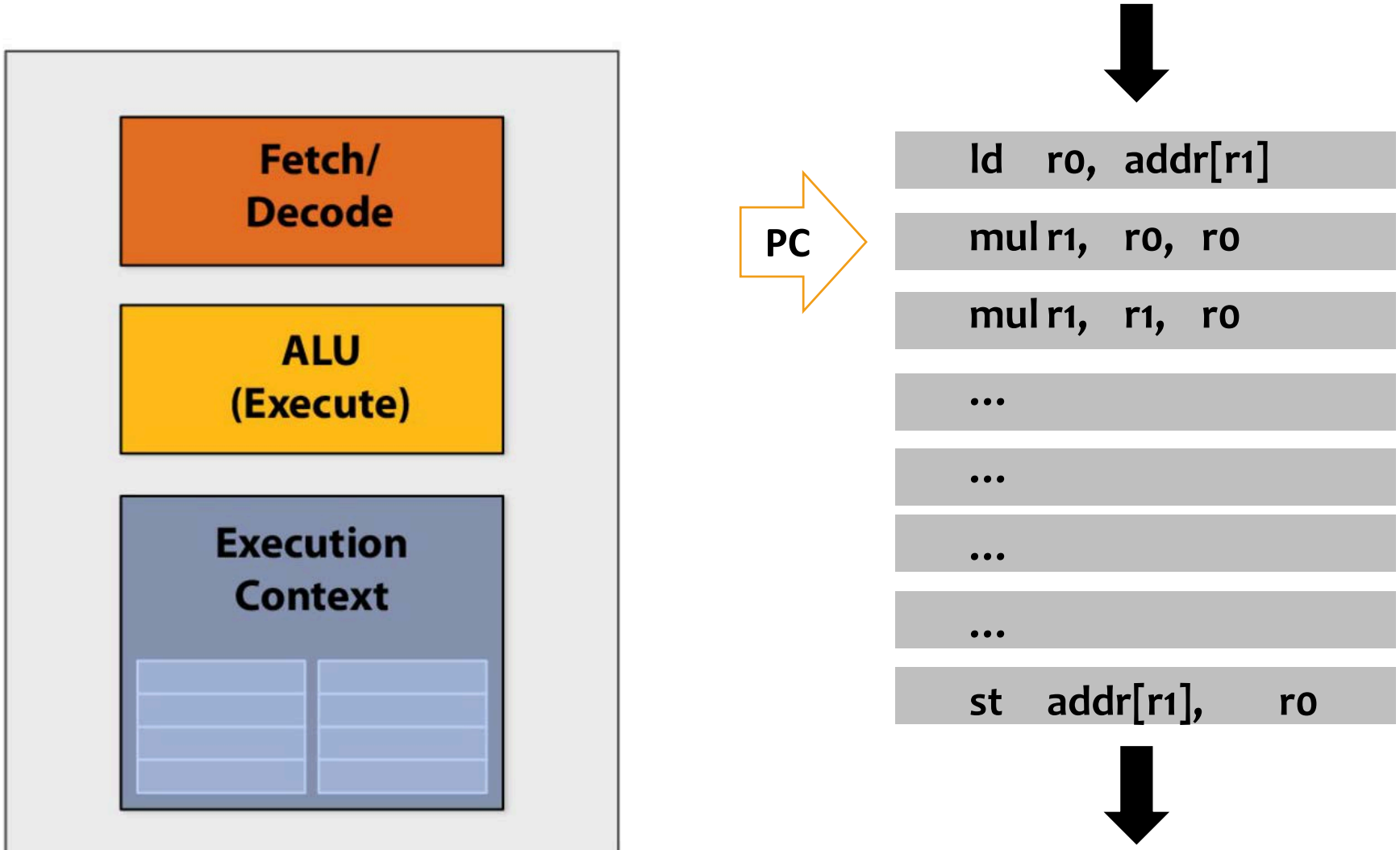
Execute program

A simple processor: execute one instruction per clock



Execute program

A simple processor: execute one instruction per clock

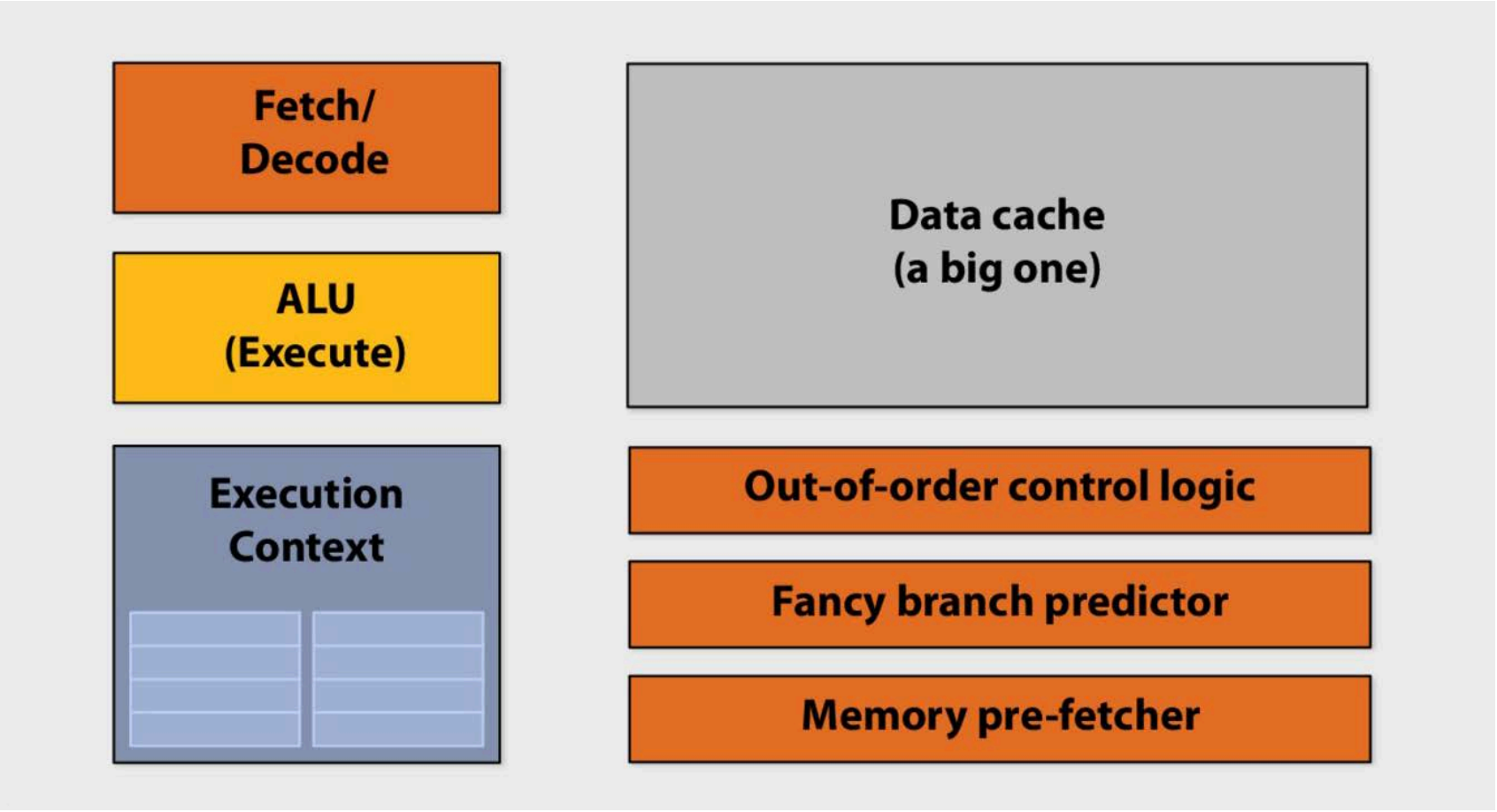


1. Superscalar

Execute program

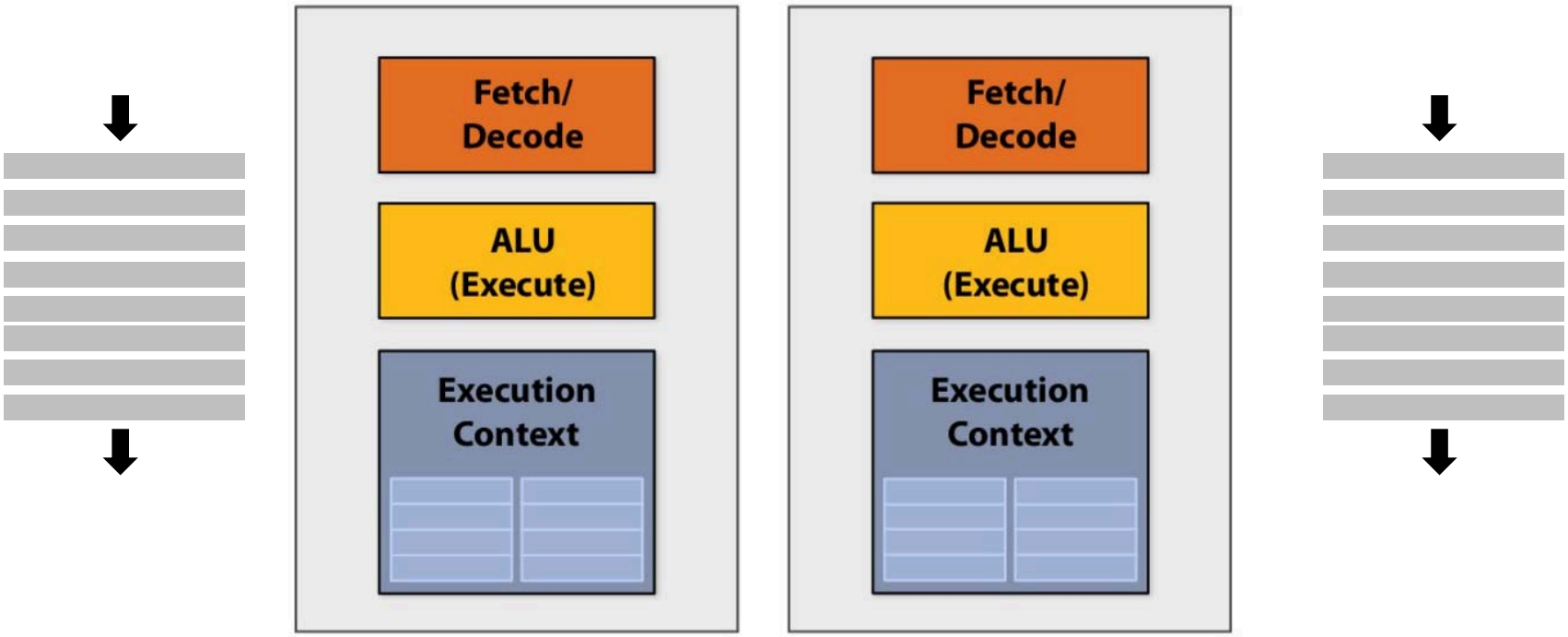
Execute more than one instruction per clock

More transistors make chip smarter!



2. Multi-core

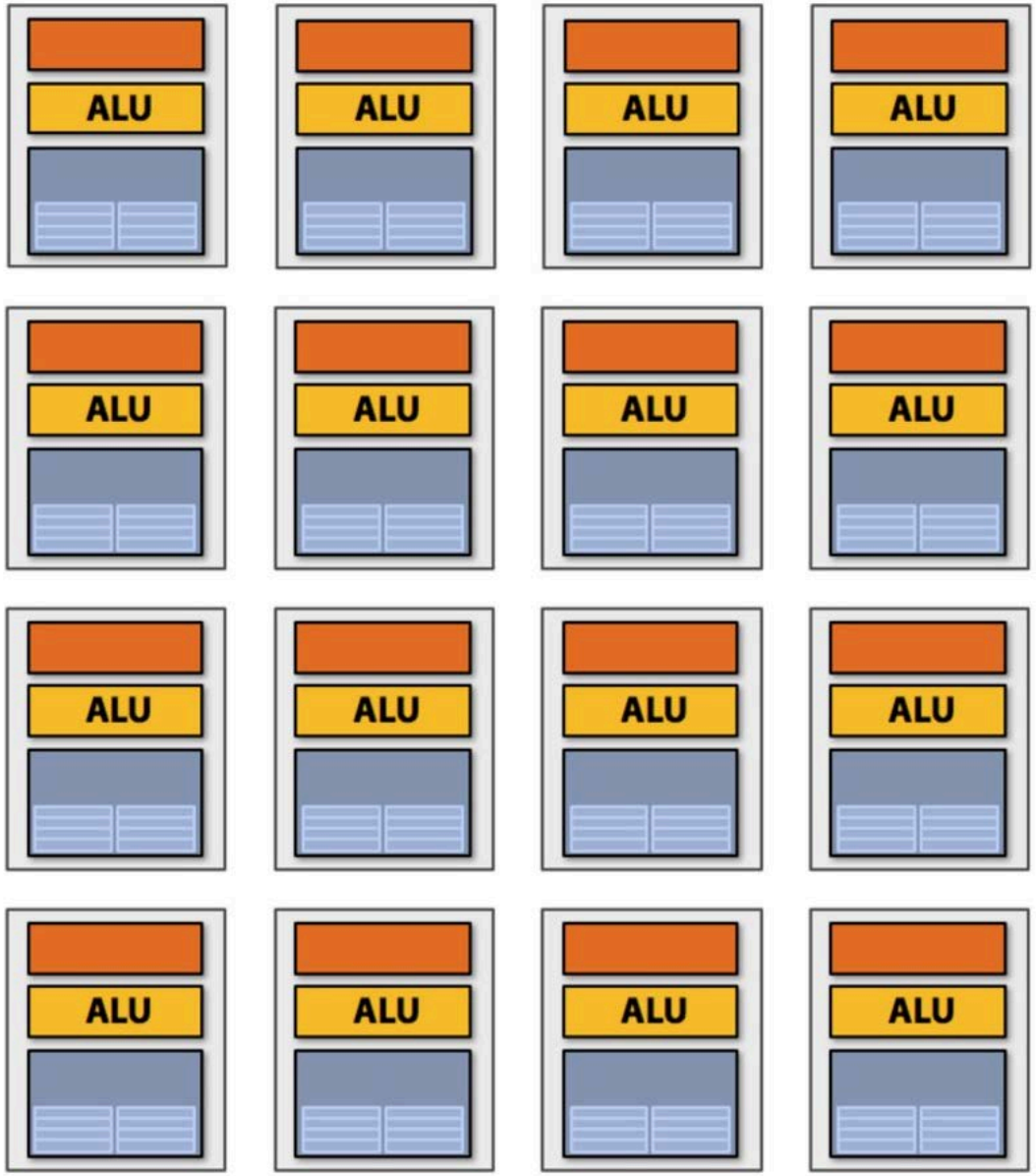
Two cores: compute two instructions in parallel



Each core is slower because we excise all the fancy stuff
But now we have two cores !

With the very same idea…

Sixteen cores: compute 16 instructions in parallel



3. SIMD

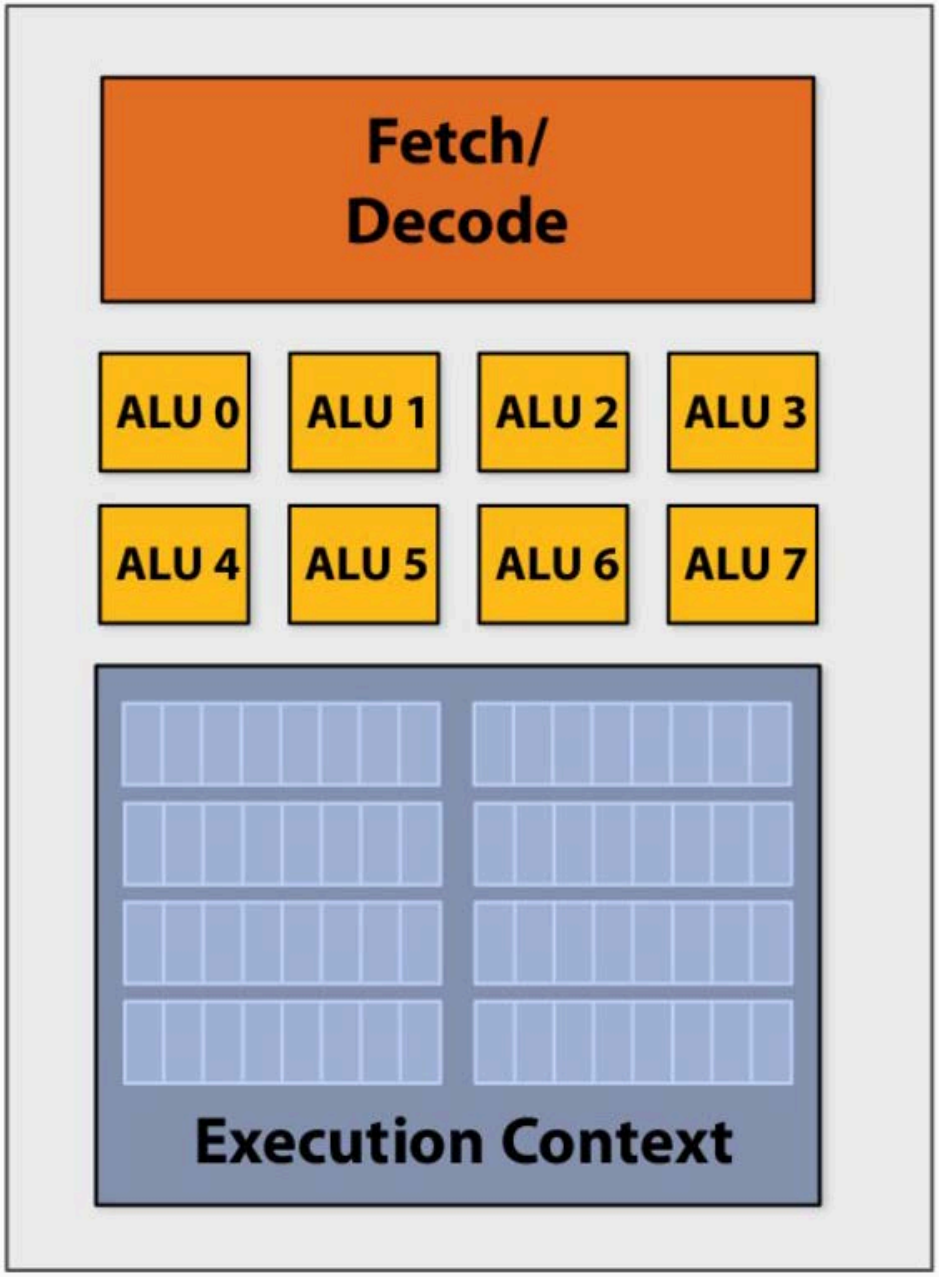
SIMD processing

Single Instruction, Multiple Data

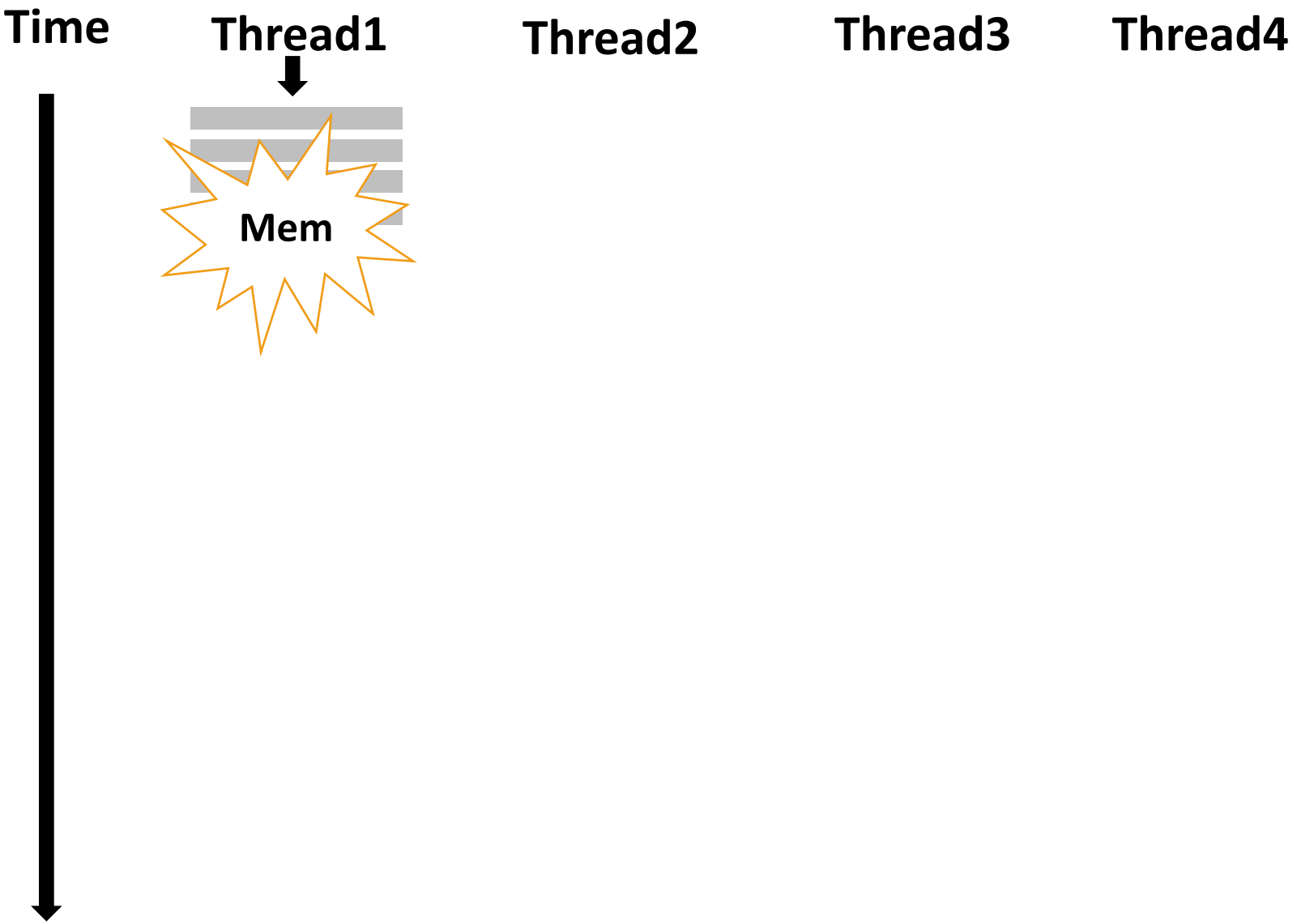
Same instruction broadcast to all ALUs

Efficient design for data-parallel workloads

Executed in parallel on all ALUs



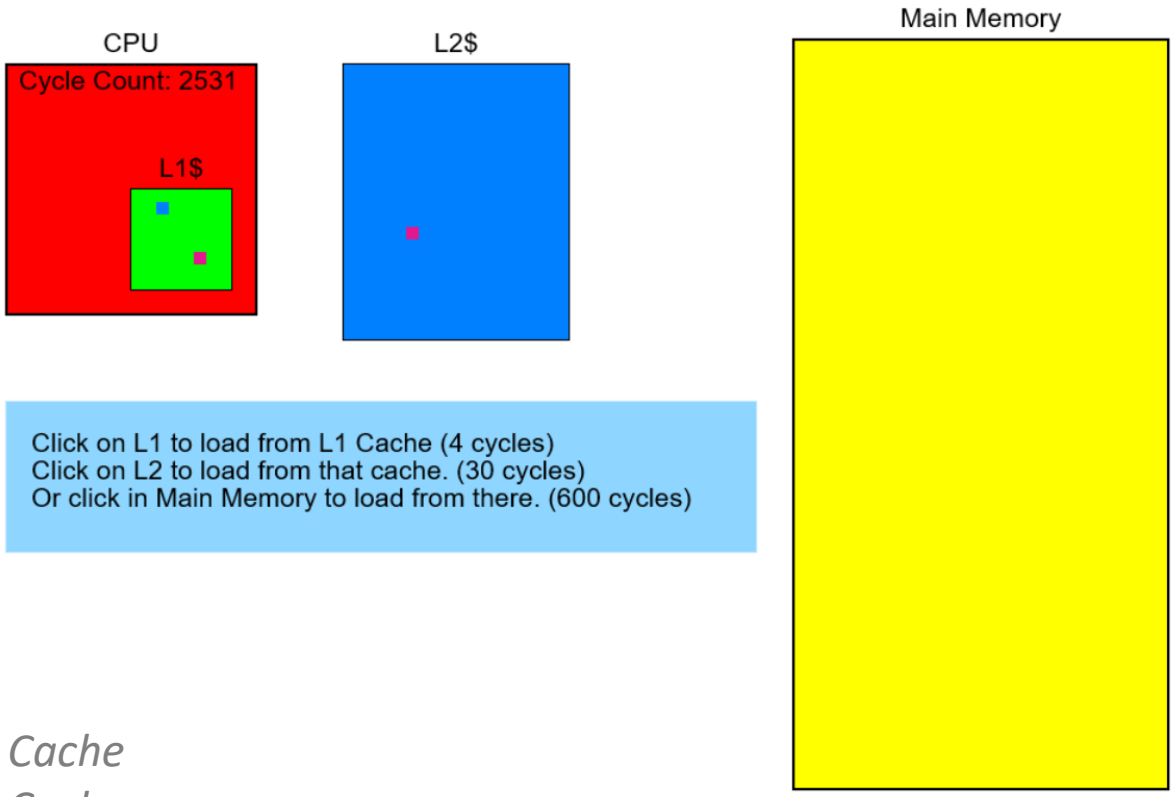
4. Multi-threading



One core, one thread.

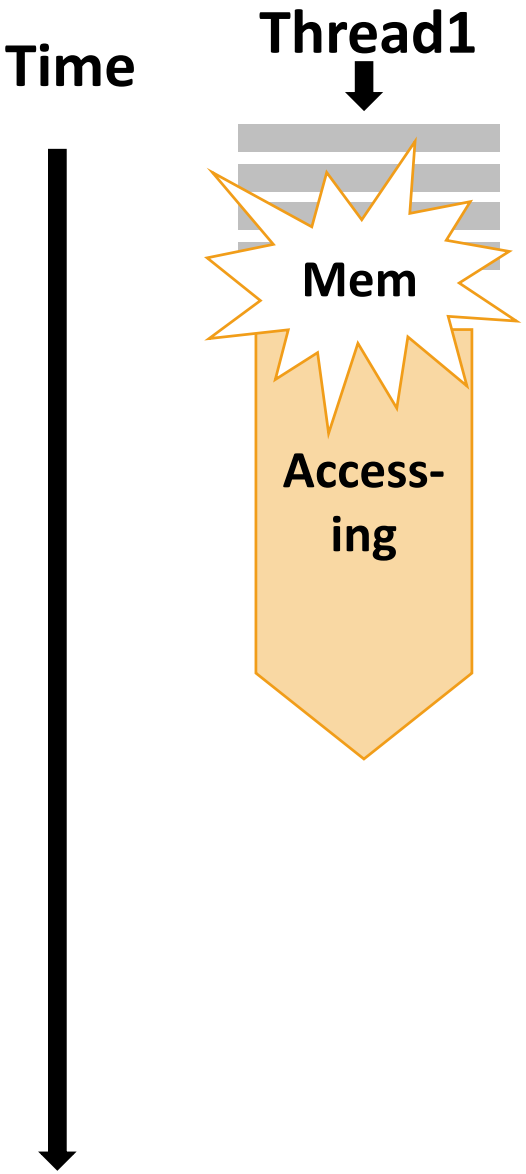
Memory accessing takes time

CacheFun

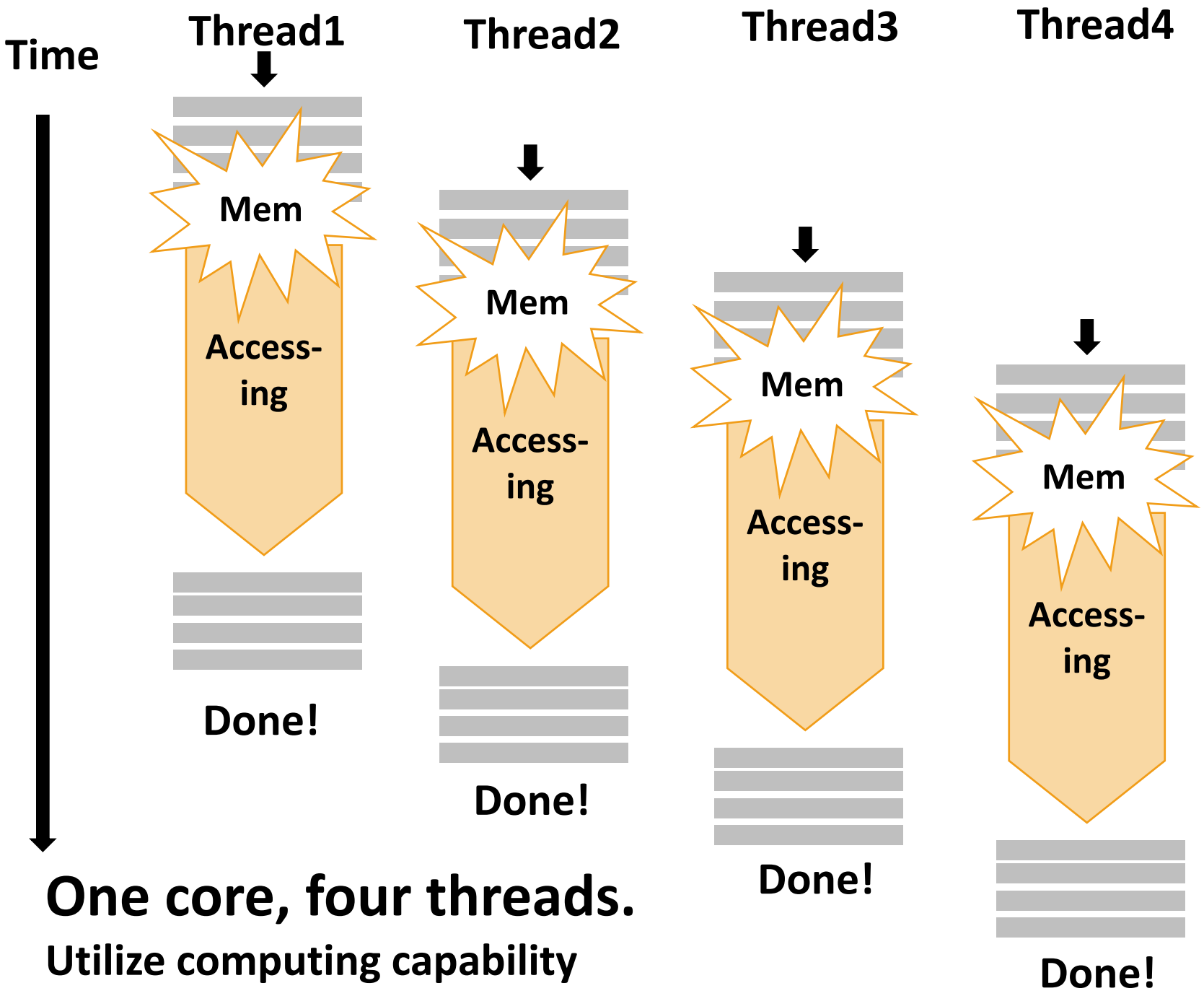


L1$: Level 1 Cache
L2$: Level 2 Cache

http://www.overbyte.com.au/misc/Lesson3/CacheFun.html



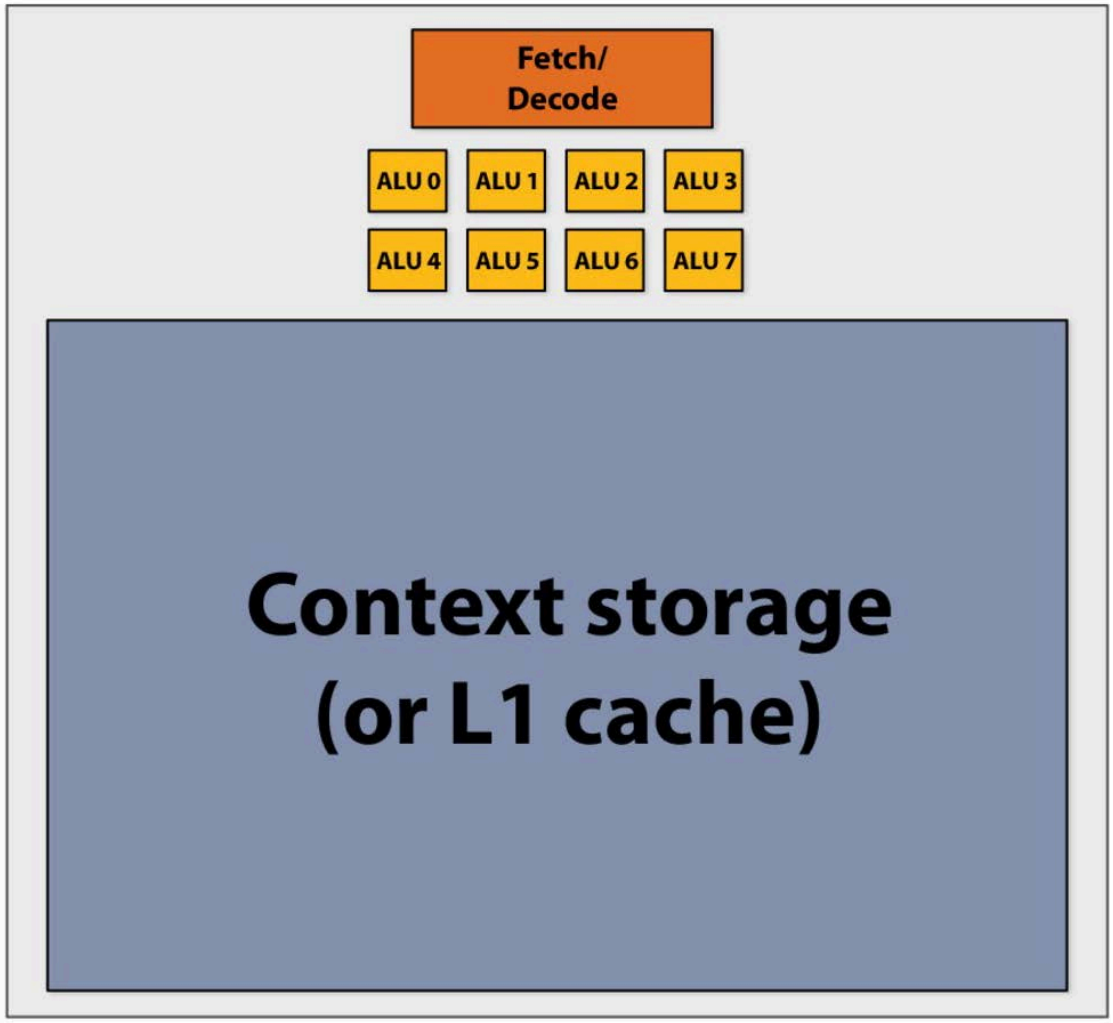
One core, one thread.



One core, four threads.

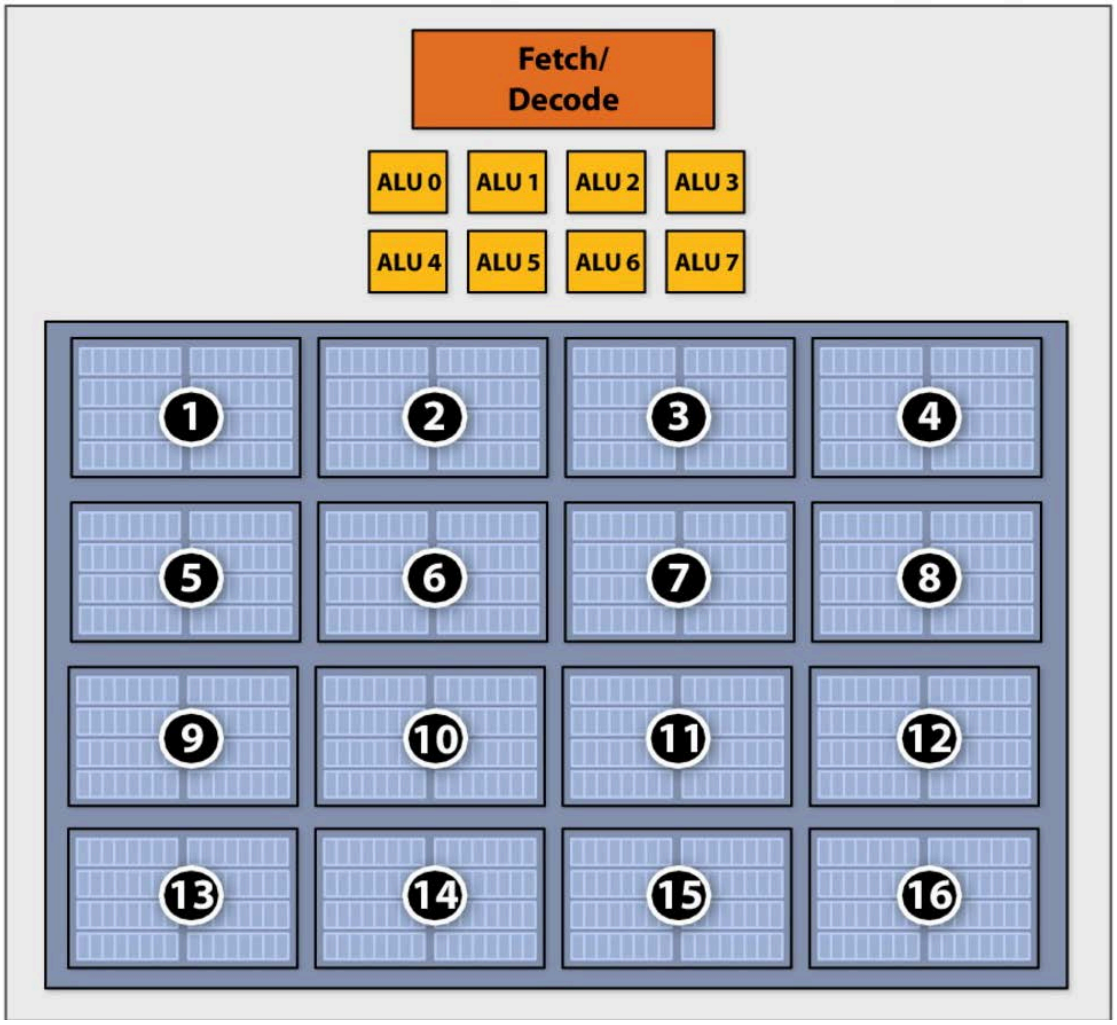
Utilize computing capability

1 core, 1 hardware thread



1 core, 16 hardware threads

Storage for small working set per thread



Parallel architectures that modern processors employ:

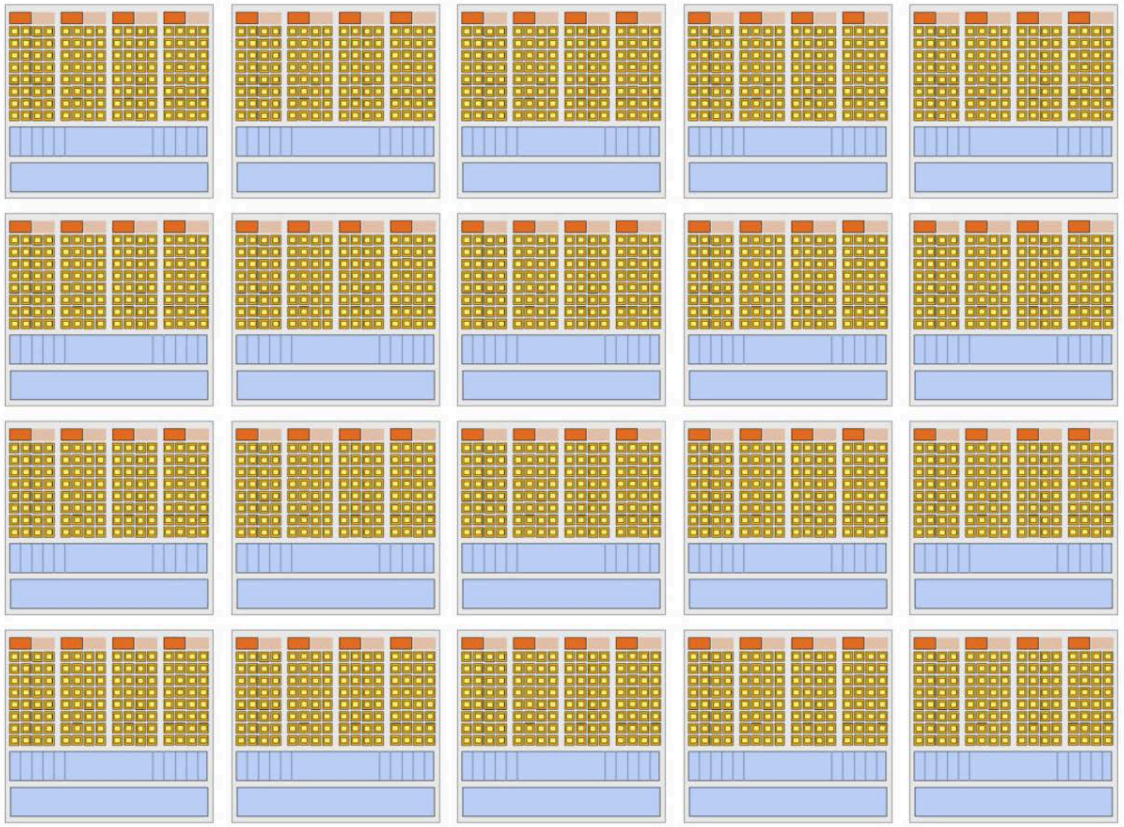
- **Multi-core**
- **SIMD**
- **Multi-threading**

GPU architecture pushes these ideas into extreme scales!

GPU: Graphic Processing Unit

With the very same idea…

GPU: NVIDIA GTX1080



128 SIMD/core

64 warps

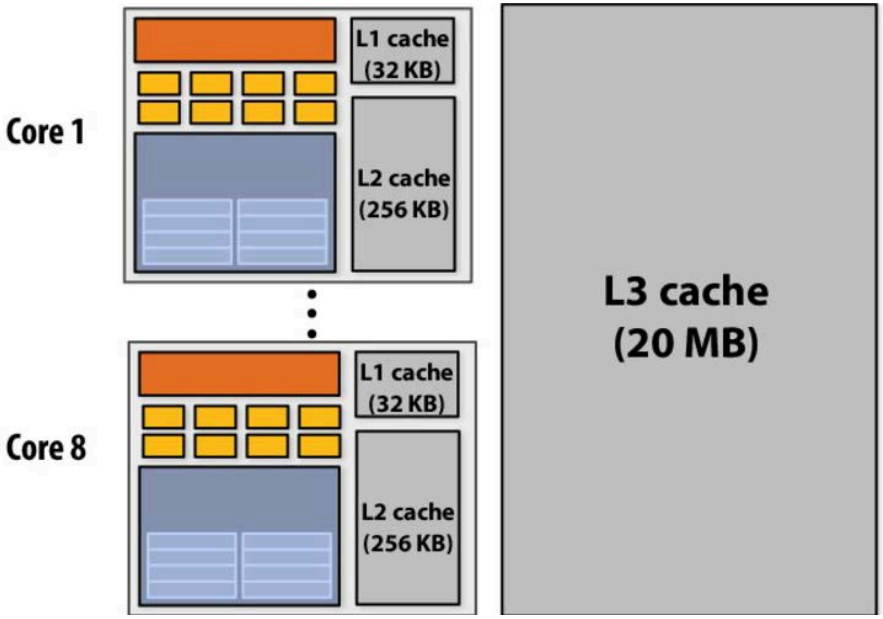
** Warp是GPU中软件层面的概念，是GPU线程调度的最小单位。在GPU中每次执行一个warp*

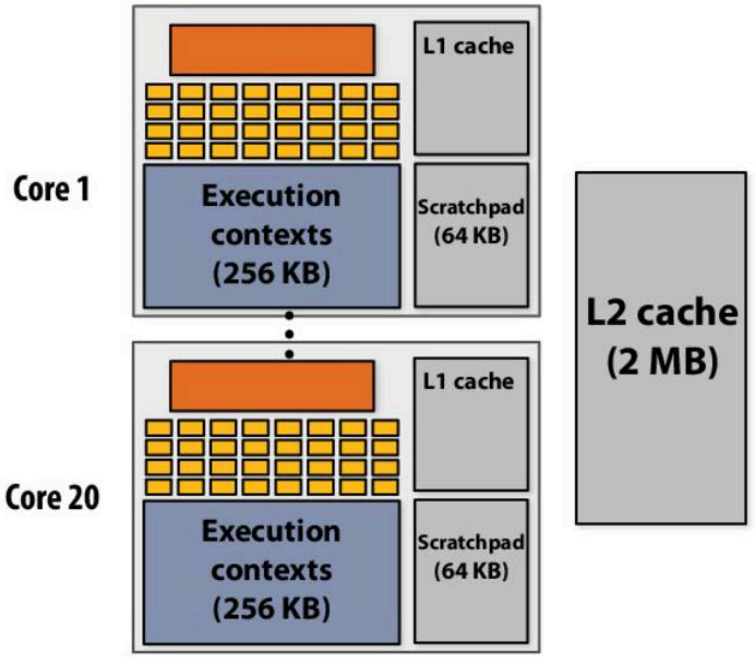
Over 2048 elements can be processed concurrently by a core

And it has 20 cores!

GPU vs. CPU

From parallelism perspective





CPU
Big caches
Modest memory bandwidth
Few threads

GPU
Small caches
Huge memory bandwidth
many threads

A Toy Example

Task: element-wise multiplication of two vectors A and B

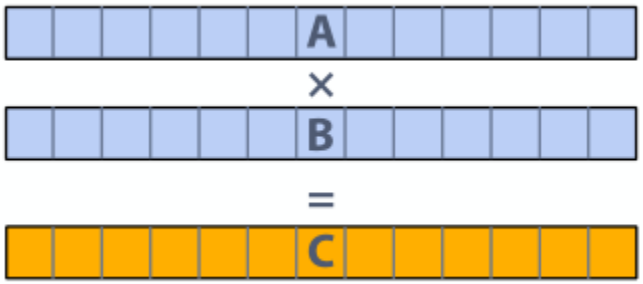
Assume vectors contain millions of elements

-Load input A[i]

-Load input B[i]

-Compute A[i]×B[i]

-Store result into C[i]

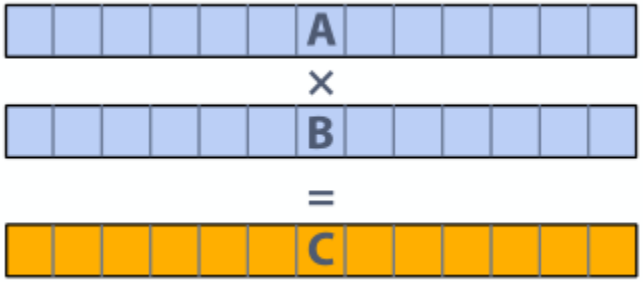


A Toy Example

Task: element-wise multiplication of two vectors A and B

Assume vectors contain millions of elements

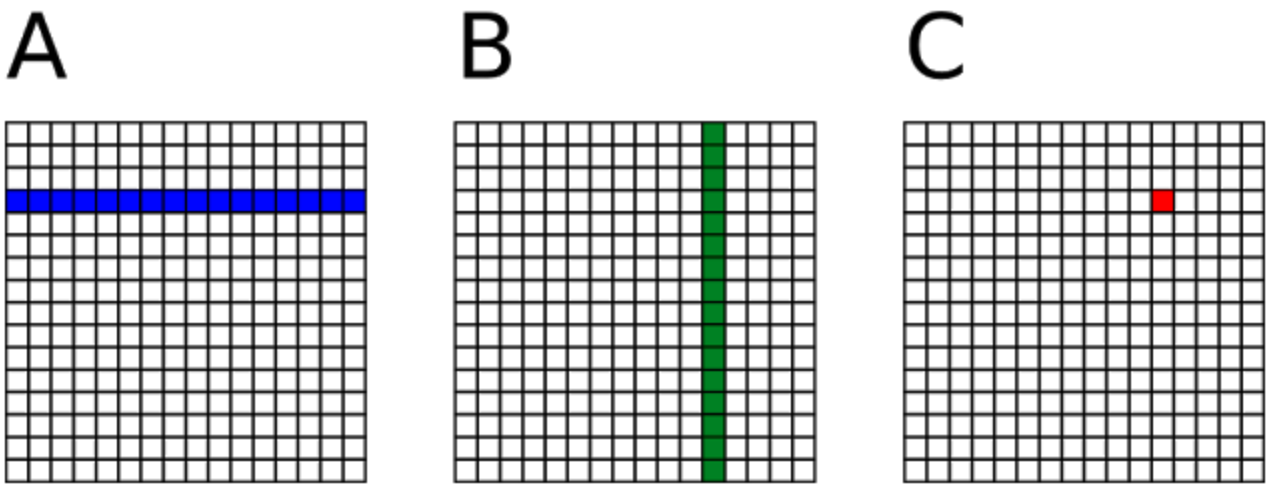
- **Load input A[i]**
- **Load input B[i]**
- **Compute A[i]×B[i]**
- **Store result into C[i]**



Three memory operations for every MUL
NVIDIA GTX 1080 GPU can do 2560 MULs per clock (@ 1.6 GHz)
Need 45 TB/sec of bandwidth to keep functional units busy (only have 320 GB/sec)

<1% GPU efficiency… but 4.2x faster than eight-core CPU in lab!

Matrix Multiplication (Naïve Method)



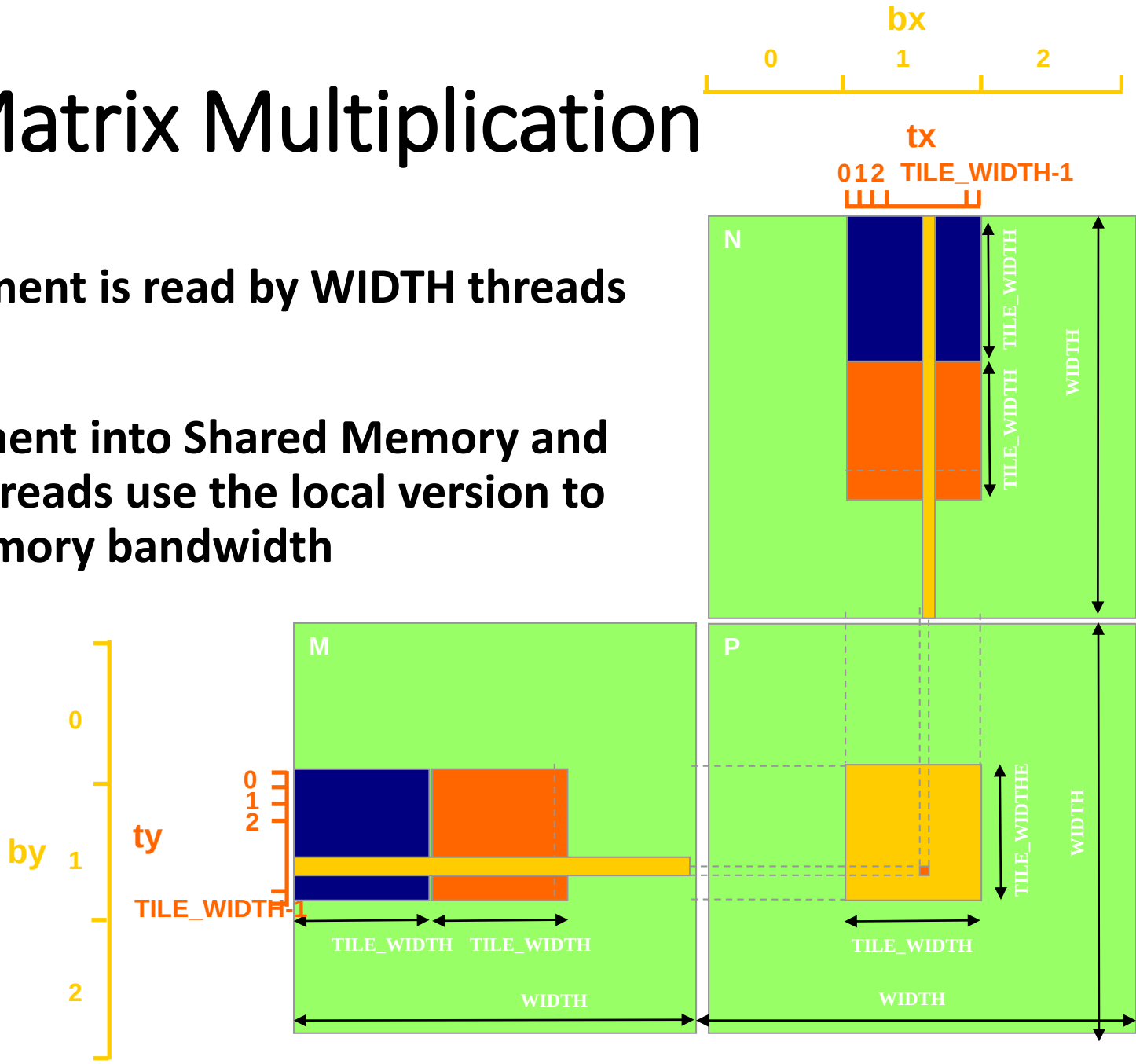
Load a row in A and a column in B, in order to obtain a single element in C

Why is it slow?

Tiled Matrix Multiplication

Each input element is read by WIDTH threads

Load each element into Shared Memory and have several threads use the local version to reduce the memory bandwidth



Bandwidth is a critical resource

Ideal parallel programs will:

- **Organize computation to fetch data from memory less often**
  - **Reuse data previously loaded by the same thread**
  - **Share data across threads (inter-thread cooperation)**
- **Request data less often (instead, do more arithmetic)**
  - **Useful term: "arithmetic intensity" — ratio of math operations to data access operations in an instruction stream**
  - **Main point: programs must have high arithmetic intensity to utilize modern processors efficiently**

Question -

How to further improve the GPU efficiency of matrix multiplication ?

References

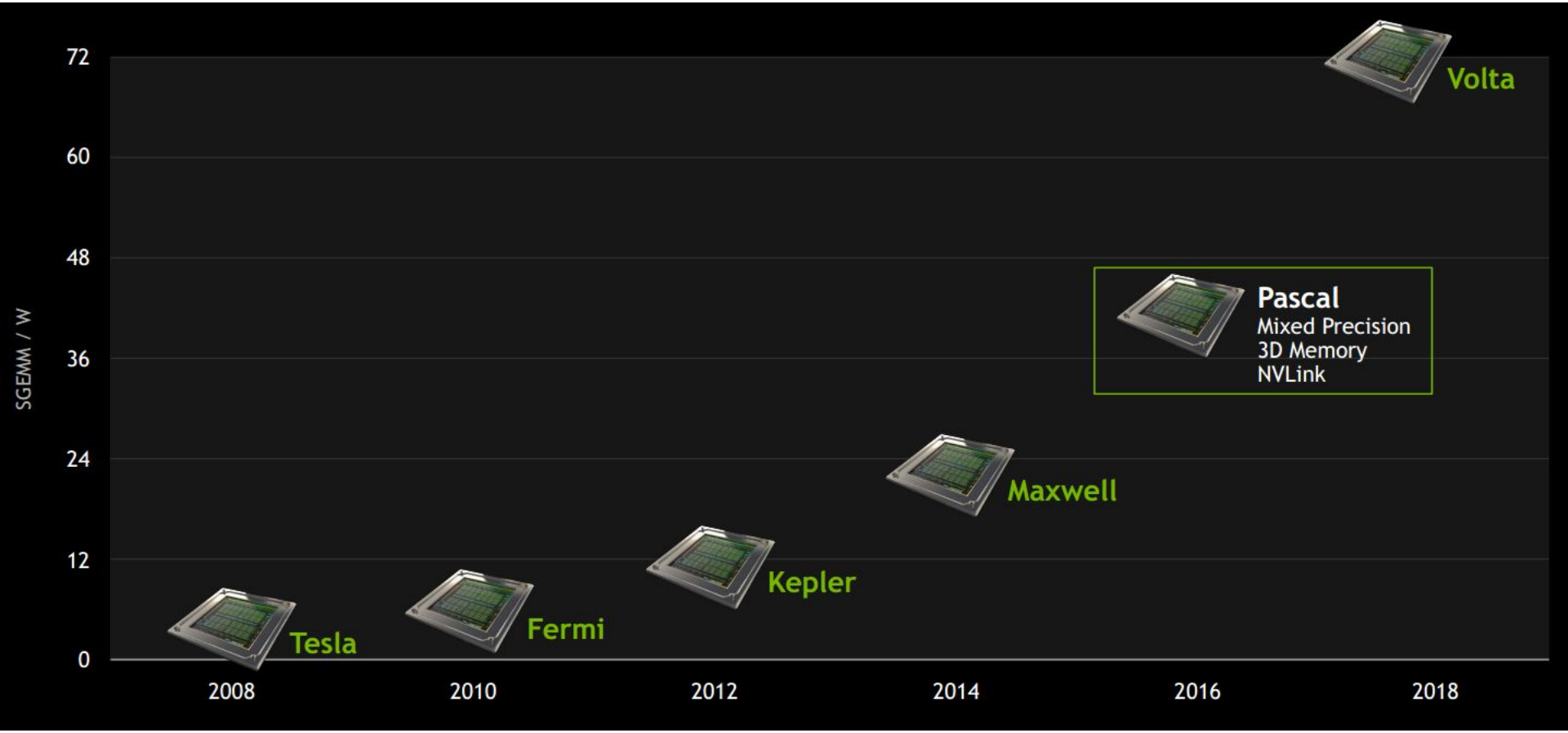
Course:

- **Parallel Computer Architecture and Programming : CMU 15-418/618, Spring 2017**

Git:

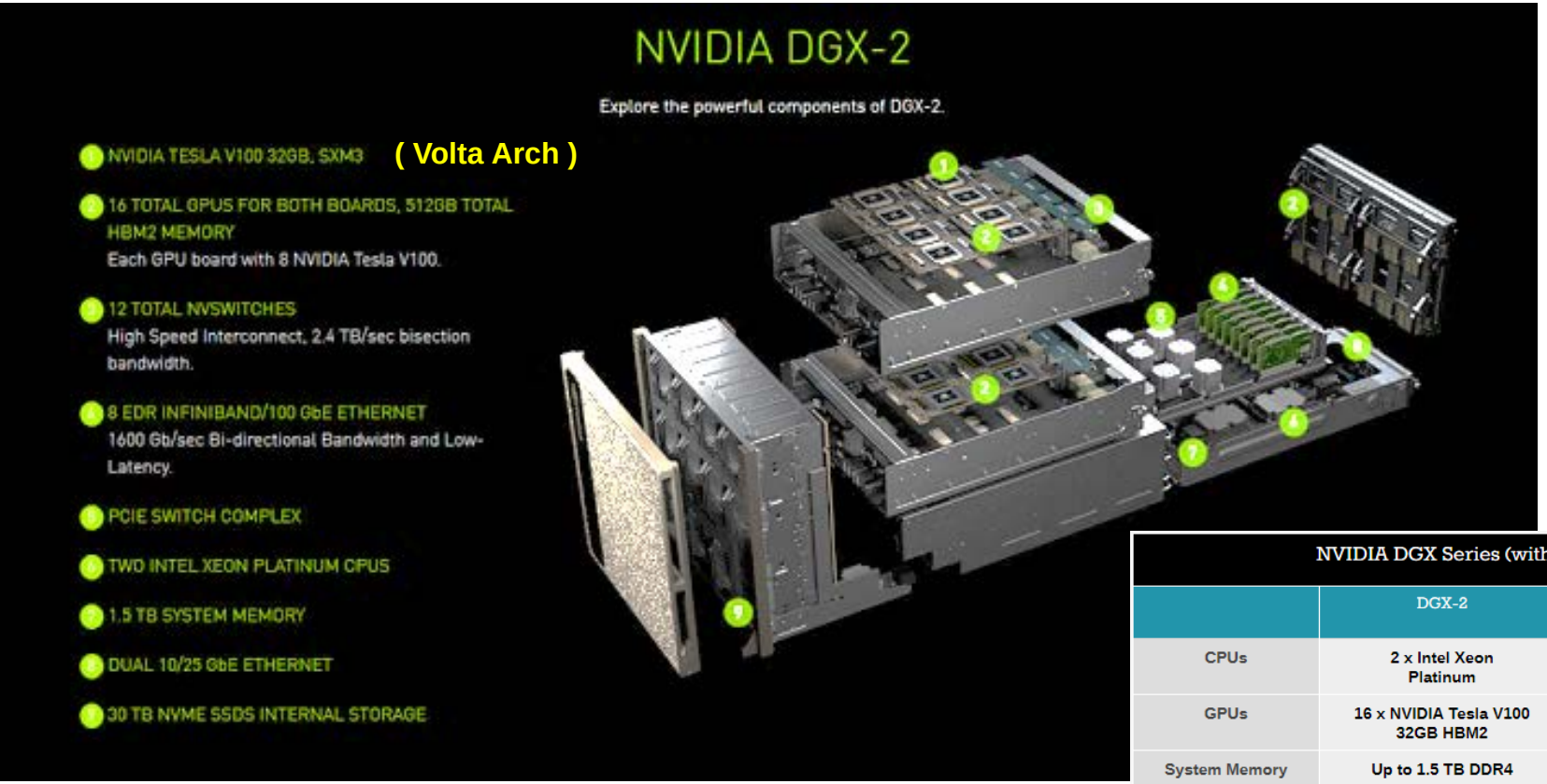
https://github.com/metorm/metorm.github.io/blob/master/_posts/2016-09-01-CUDA-matrix-multiply-optimization.markdown

NVIDIA GPU Architecture Roadmap



Single precision floating General Matrix Multiply (SGEMM)

NVIDIA DGX-2 System (released on Mar. 27, 2018)



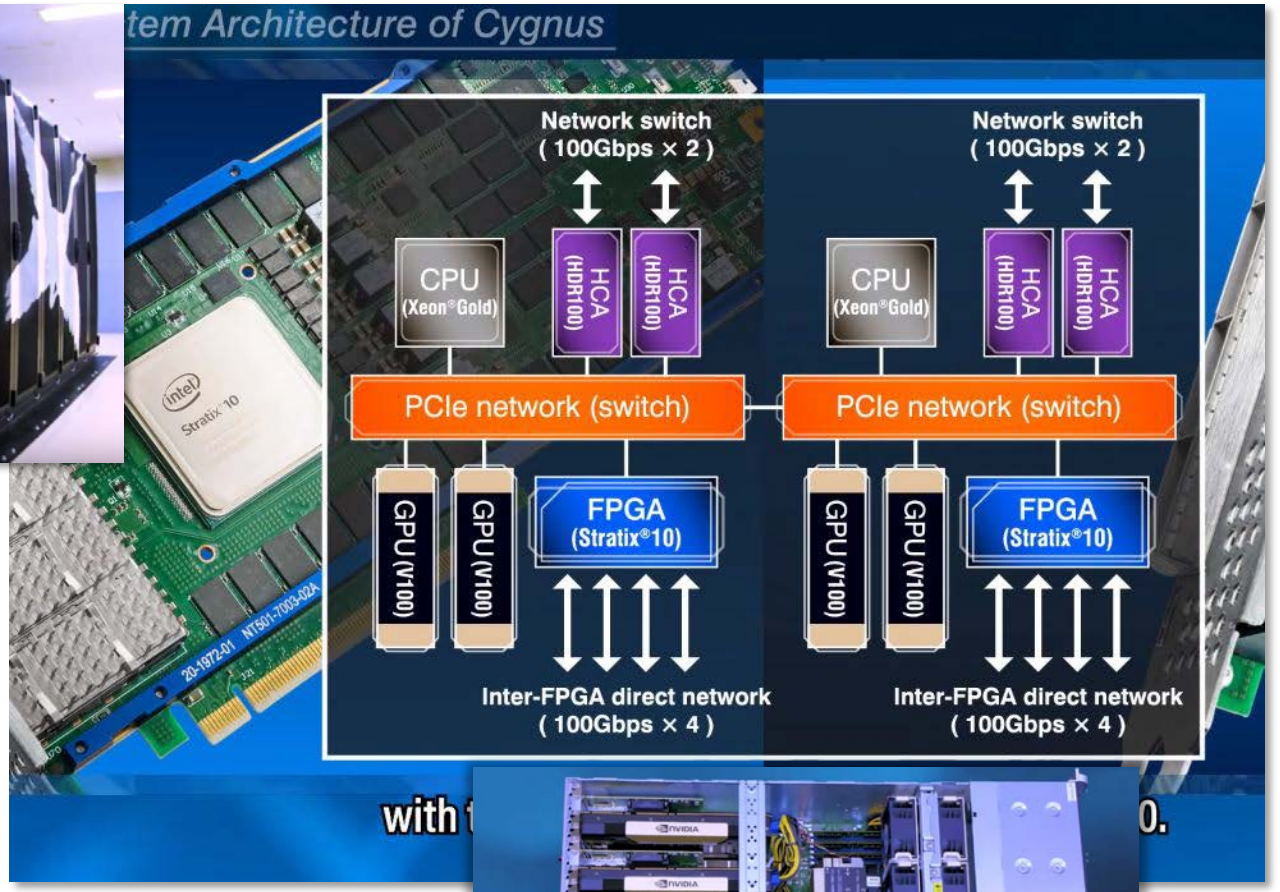
NVIDIA DGX Series (with Volta)		
	DGX-2	DGX-1
CPUs	**2 x Intel Xeon Platinum**	2 x Intel Xeon E5-2600 v4
GPUs	**16 x NVIDIA Tesla V100 32GB HBM2**	8 x NVIDIA Tesla V100 16 GB HBM2
System Memory	**Up to 1.5 TB DDR4**	Up to 0.5 TB DDR4
GPU Memory	**512 GB HBM2 (16 x 32 GB)**	256 GB HBM (8 x 32 GB)
Storage	**30 TB NVMe** Up to 60 TB	4 x 1.92 TB NVMe
Networking	**8 x 100 GbE**	4 x IB + 2 x 10 GbE
Power	**10 kW**	3.5 kW
Size	**350 lbs**	134 lbs
GPU Throughput	**Tensor: 1920 TFLOPs** FP16: 480 TFLOPs FP32: 240 TFLOPs FP64: 120 TFLOPs	Tensor: 960 TFLOPs FP16: 240 TFLOPs FP32: 120 TFLOPs FP64: 60 TFLOPs
Cost	$399,000	$149,000

learn

“The more you ~~buy~~, the more you save”

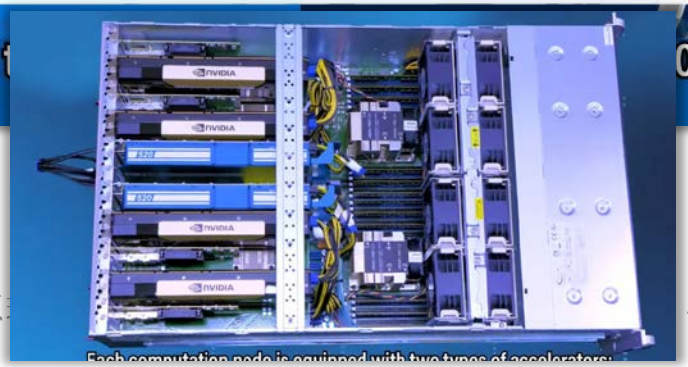
——黄仁勋, Co-founder / President of NVIDIA

异构超算：日本的新超级计算机Cygnus，GPU和FPGA双硬件加持 (2019年4月)



Cygnus内部共80个节点

- 每个节点配备
  - 2x Xeon Skylake Gold CPU
  - 4x Nvidia Tesla V100 GPU
- 其中32个节点还配备
  - 2x Intel Stratix 10 FPGA



- **GPU：粗粒度并行计算，SIMD**
- **FPGA：细粒度并行计算，非SIMD**

B0911006Y-01　2018-2019学年春季学期

计算机组成原理

第 18 讲　控制单元的功能

CPU控制单元会发出什么样的操作命令/控制信号? 如何产生这些信号? 在“多级时序系统”中又如何控制产生这些微操作命令?

主讲教师：　张　科

2019年5月6日

中国科学院大学 University of Chinese Academy of Sciences
中科院计算所 INSTITUTE OF COMPUTING TECHNOLOGY,CAS

第 9 章　控制单元的功能

9.1　操作命令的分析

9.2　控制单元的功能

9.1 操作命令的分析

完成一条指令分 4 个工作周期

取指周期

间址周期

执行周期

中断周期

9.1 操作命令的分析

一、取指周期

PC ⟶ MAR ⟶ 地址线

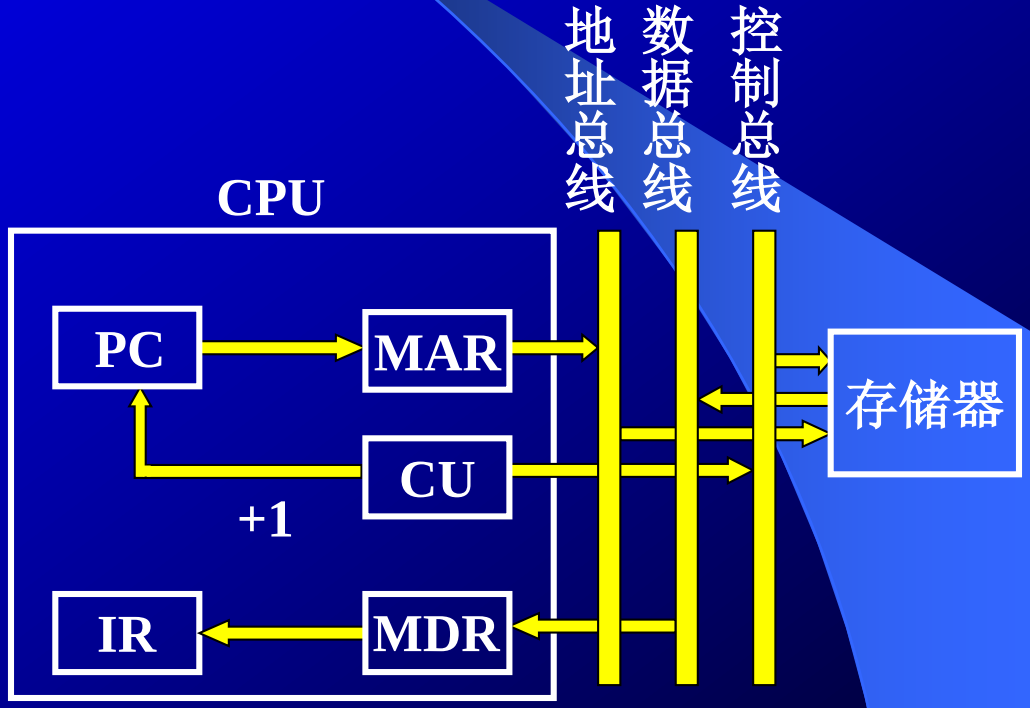
1 ⟶ R (即MemRead)

M (MAR) ⟶ MDR

MDR ⟶ IR

OP（IR）⟶ CU

(PC) + 1 ⟶ PC



二、间址周期

9.1

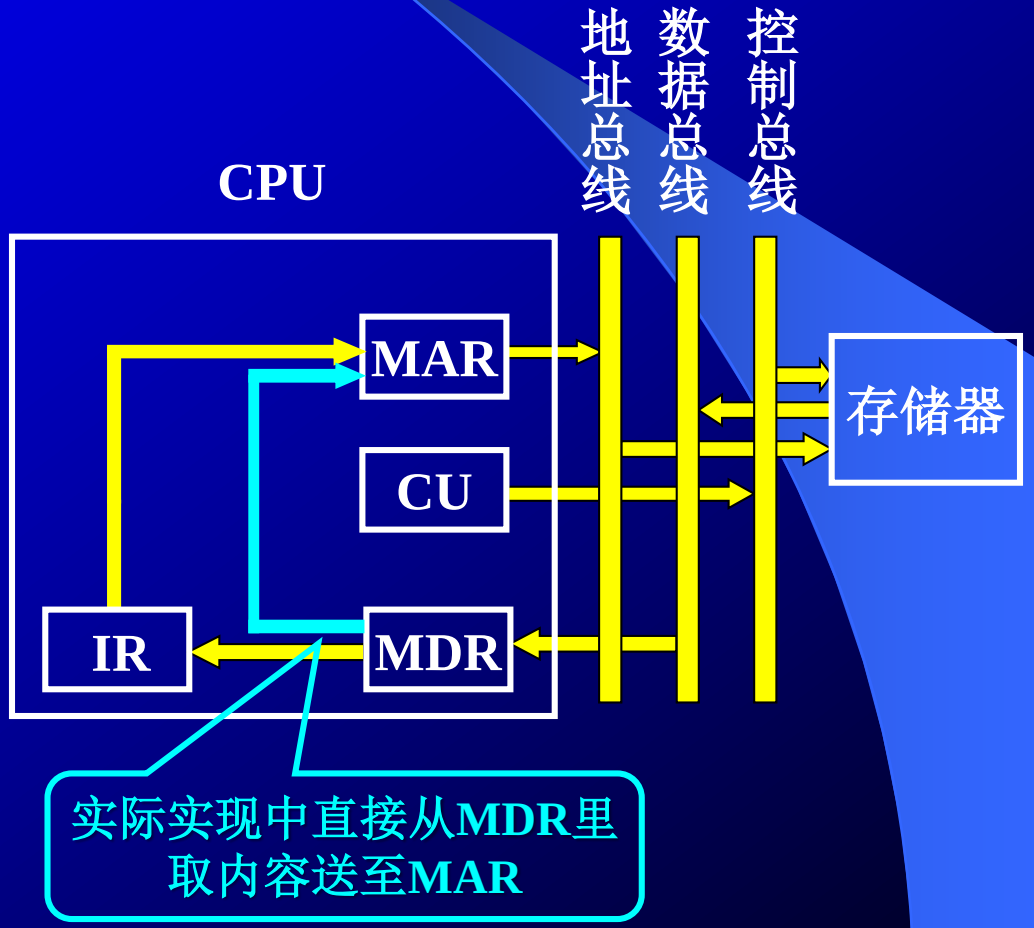
指令形式地址 ⟶ MAR

记为Ad (IR) ⟶ MAR

1⟶R

M (MAR) ⟶MDR

MDR⟶ Ad (IR)



三、执行周期

9.1

1. 非访存指令

(1) **CLA** 清累加器 0 → ACC

(2) **COM** 累加器取反 $\overline{ACC}$ → ACC

符号位不变

(3) **SHR** 算术右移 L(ACC) → R(ACC), ACC_0 → ACC_0

(4) **CSL** 循环左移 R(ACC) → L(ACC), ACC_0 → ACC_n

(5) **STP** 停机指令 0 → G

运行标志触发器

2. 访存指令

(1) 加法指令　**ADD　X**

Ad(IR) ⟶ MAR

1 ⟶ R

M(MAR) ⟶ MDR

(ACC) + (MDR) ⟶ ACC

(2) 存数指令　**STA　X**

Ad(IR) ⟶ MAR

1 ⟶ W

ACC ⟶ MDR

MDR ⟶ M(MAR)

实际中要考虑写使能和写数据的信号关系

(3) 取数指令　LDA　X

Ad (IR) ⟶ MAR

1 ⟶ R

M (MAR) ⟶ MDR

MDR ⟶ ACC

3. 转移指令

(1) 无条件转　JMP　X

Ad (IR) ⟶ PC

(2) 条件转移　BAN　X　（负则转）

$A_0 \cdot Ad(IR) + \bar{A}_0 (PC) \longrightarrow PC$

累加器最高位

4. 三类指令的指令周期

非访存 指令周期 | 取指周期 | 执行周期 |

直接访存 指令周期 | 取指周期 | 执行周期 |

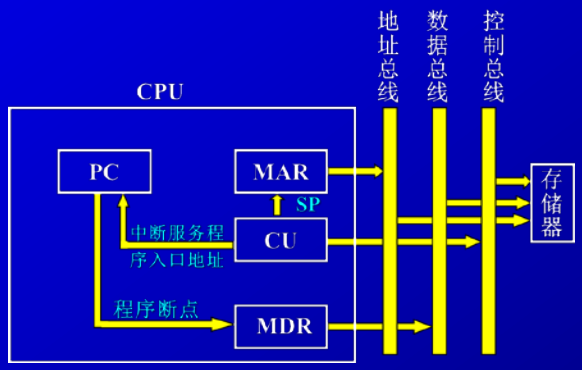
间接访存 指令周期 | 取指周期 | 间址周期 | 执行周期 |

转移 指令周期 | 取指周期 | 执行周期 |

包含无条件转移和条件转移两种

间接转移 指令周期 | 取指周期 | 间址周期 | 执行周期 |

四、中断周期



9.1

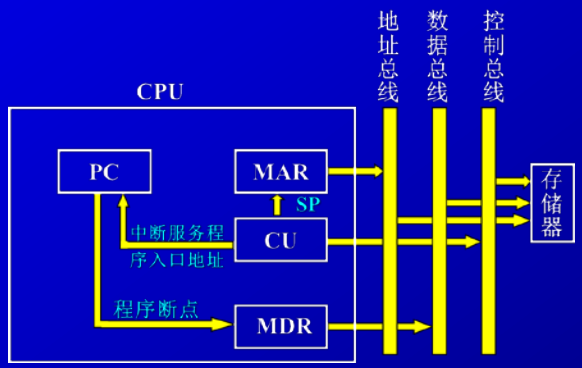
程序断点存入“0”地址	程序断点 进栈
0 ⟶ MAR	(SP) – 1 ⟶ MAR
1 ⟶ W	1 ⟶ W
PC ⟶ MDR	PC ⟶ MDR
MDR ⟶ M (MAR)	MDR ⟶ M (MAR)
向量地址 ⟶ PC	向量地址 ⟶ PC
0 ⟶ EINT（置“0”）	0 ⟶ EINT（置“0”）

PC已被+1，为程序断点

中断服务程序入口地址

关中断，P364

四、中断周期



9.1

程序断点存入“0”地址	程序断点 进栈
0 → MAR	(SP) – 1 → MAR
1 → W	1 → W
PC → MDR	PC → MDR
MDR → M(MAR)	MDR → M(MAR)

PC已被+1，为程序断点

中断服务程序入口地址

中断识别程序入口地址 M → PC

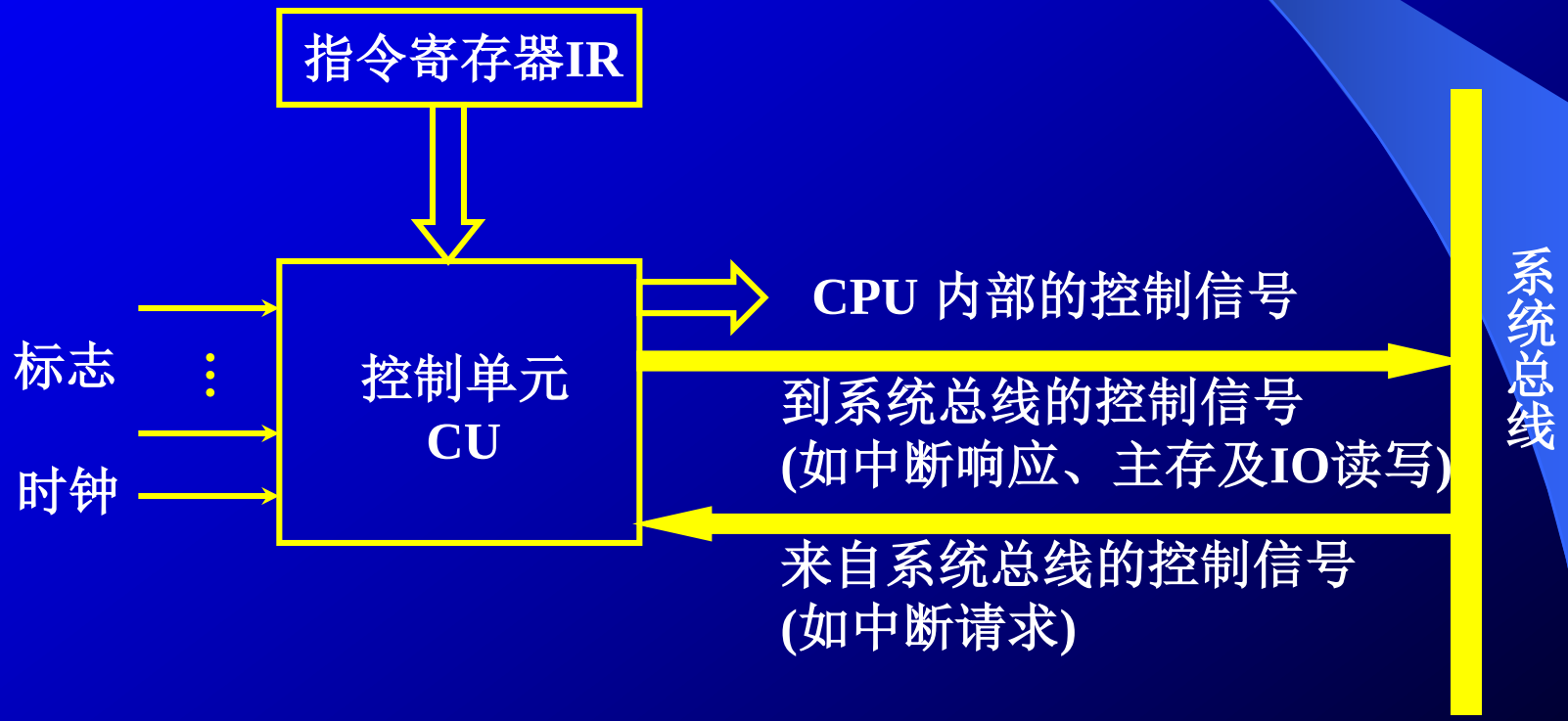
用软件查询中断服务程序入口地址

0 → EINT（置“0”）	0 → EINT（置“0”）

关中断，P364

9.2 控制单元的功能

一、控制单元的外特性



1. 输入信号

(1) 时钟

CU 受时钟控制

一个时钟脉冲

发一个操作命令或一组需同时执行的操作命令

(2) 指令寄存器 OP (IR) → CU

控制信号 与操作码有关

(3) 标志

CU 受标志控制

(4) 外来信号

如 INTR 中断请求

HRQ 总线请求

2. 输出信号

(1) CPU 内的各种控制信号

$R_i \longrightarrow R_j$

(PC) + 1 ⟶ PC

ALU　＋、－、与、或 ……

(2) 送至控制总线的信号

低电平有效

$\overline{\text{MREQ}}$	访存控制信号
$\overline{\text{IO/M}}$	访 IO/ 存储器的控制信号
$\overline{\text{RD}}$	读命令
$\overline{\text{WR}}$	写命令
INTA	中断响应信号
HLDA	总线响应信号

补充：可参考AXI总线接口信号

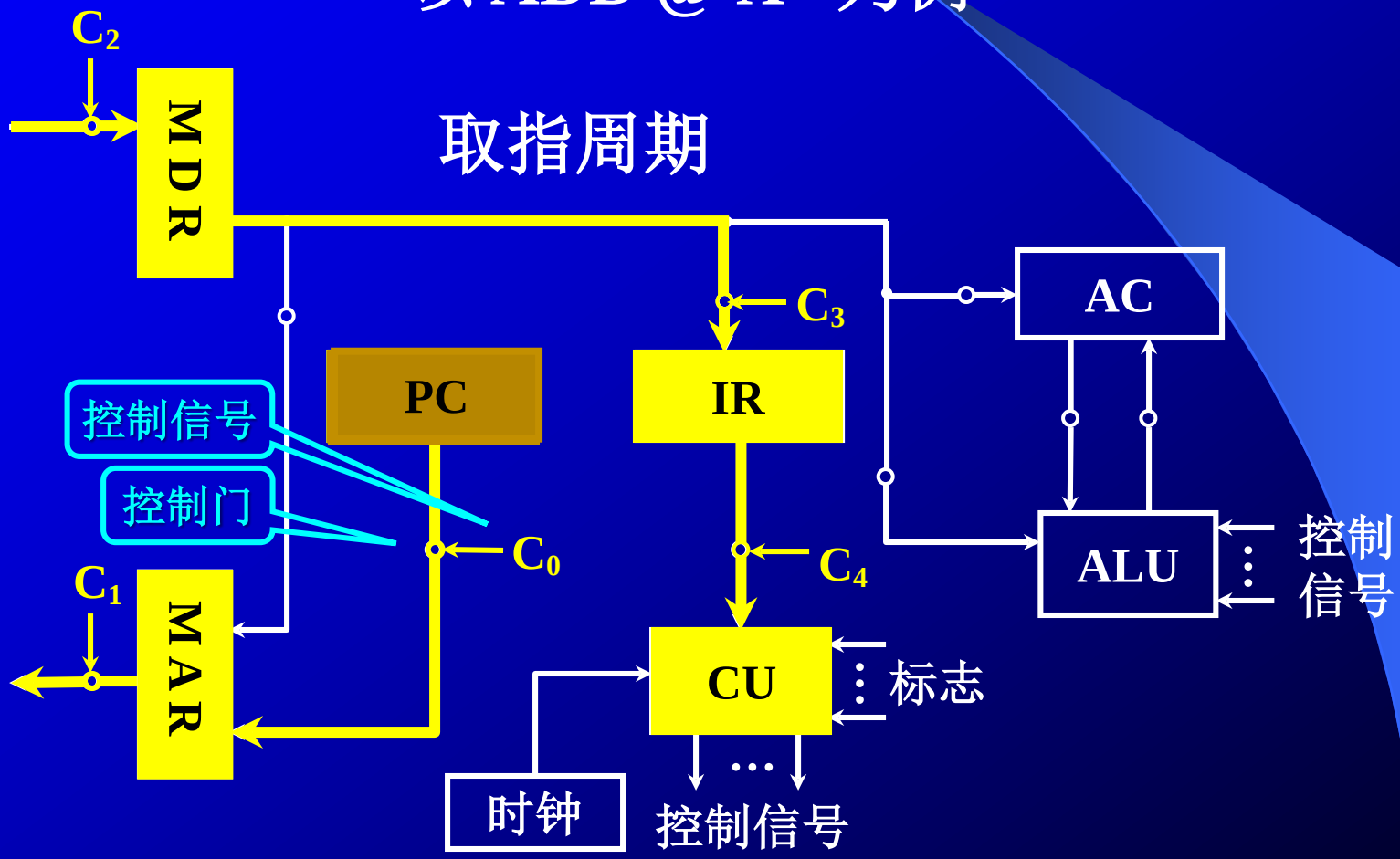
二、控制信号举例

9.2

1. 不采用 CPU 内部总线的方式

以 ADD @ X 为例

@：间址特征

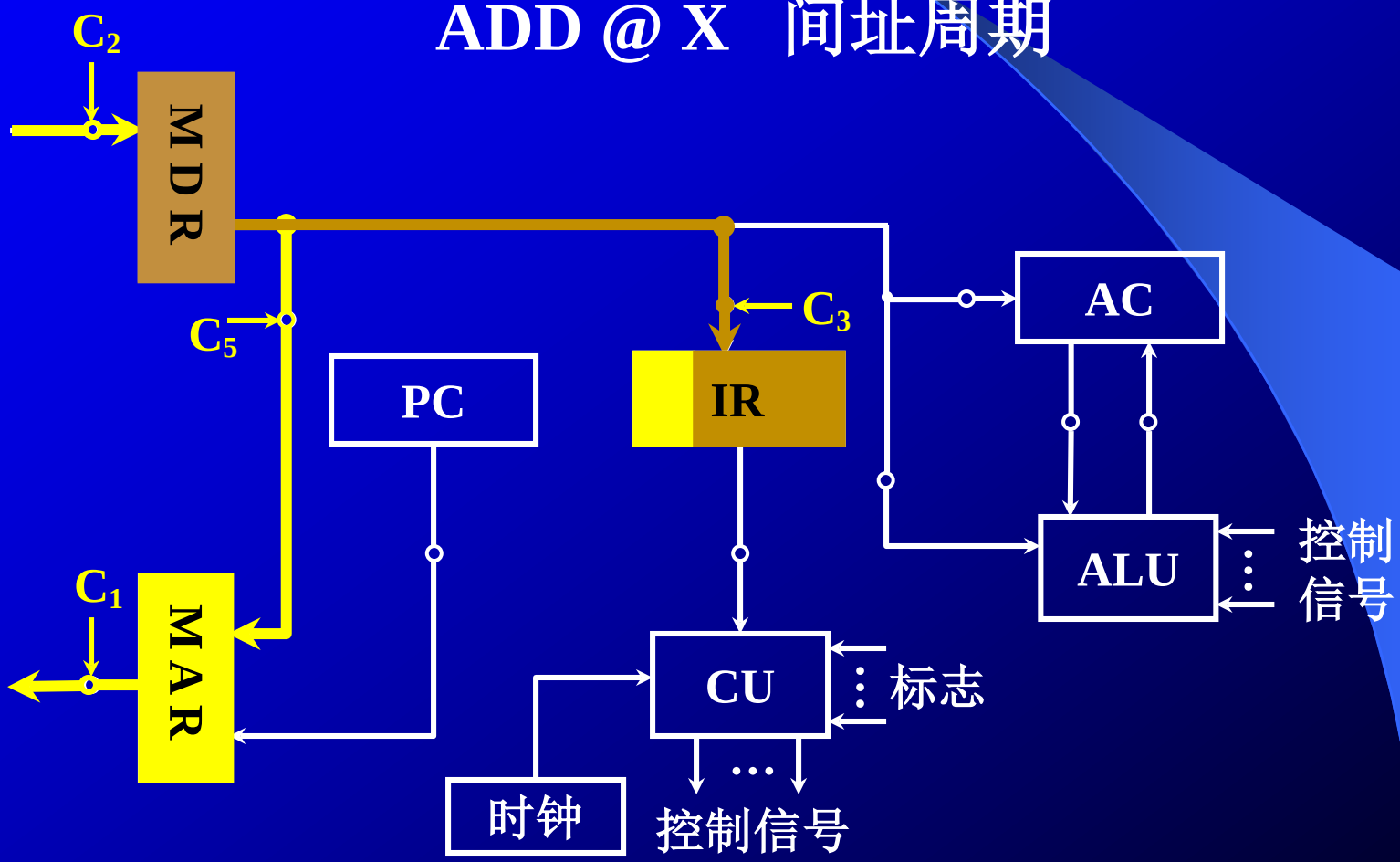


二、控制信号举例

9.2

1. 不采用 CPU 内部总线的方式

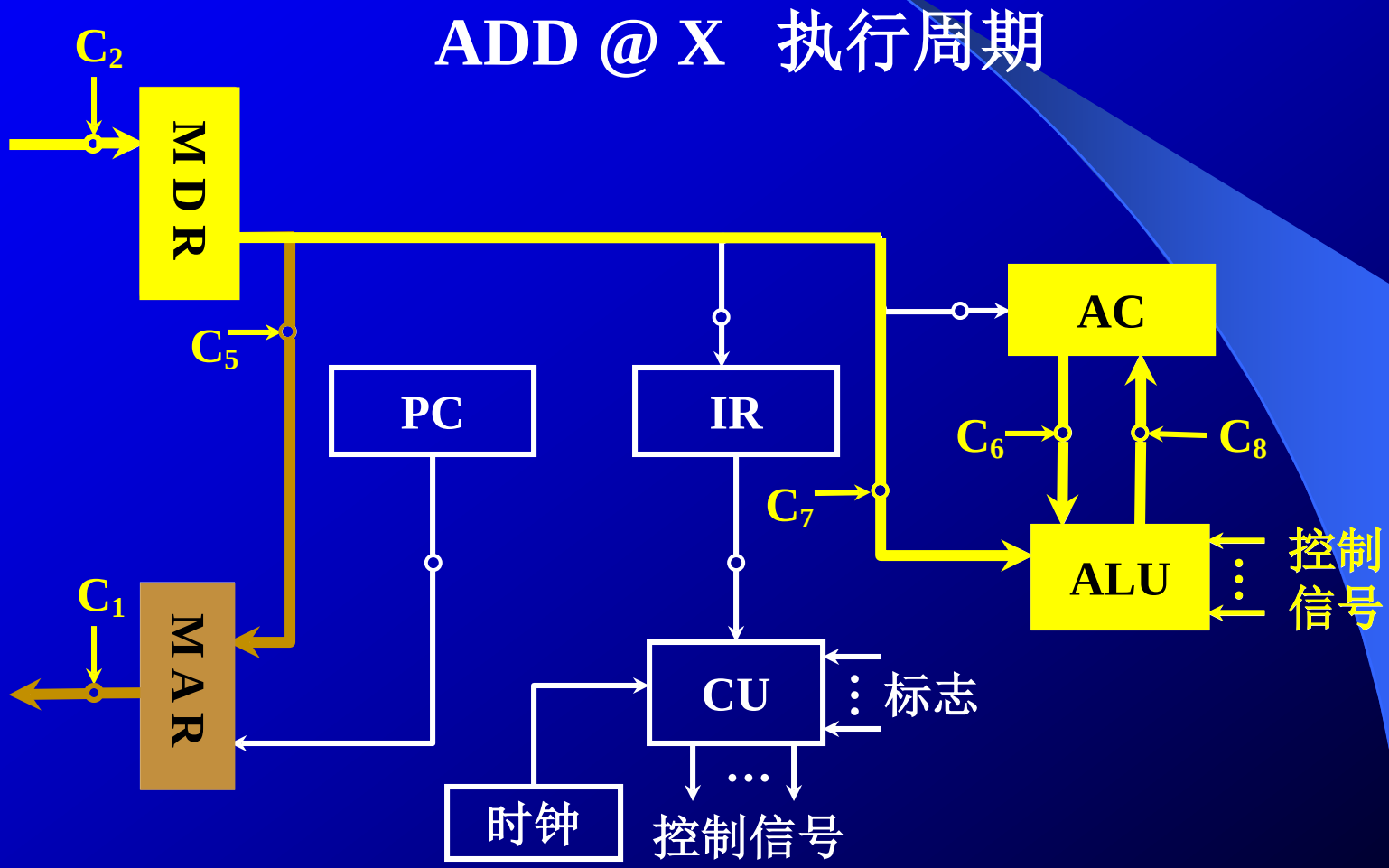
ADD @ X　间址周期



二、控制信号举例

9.2

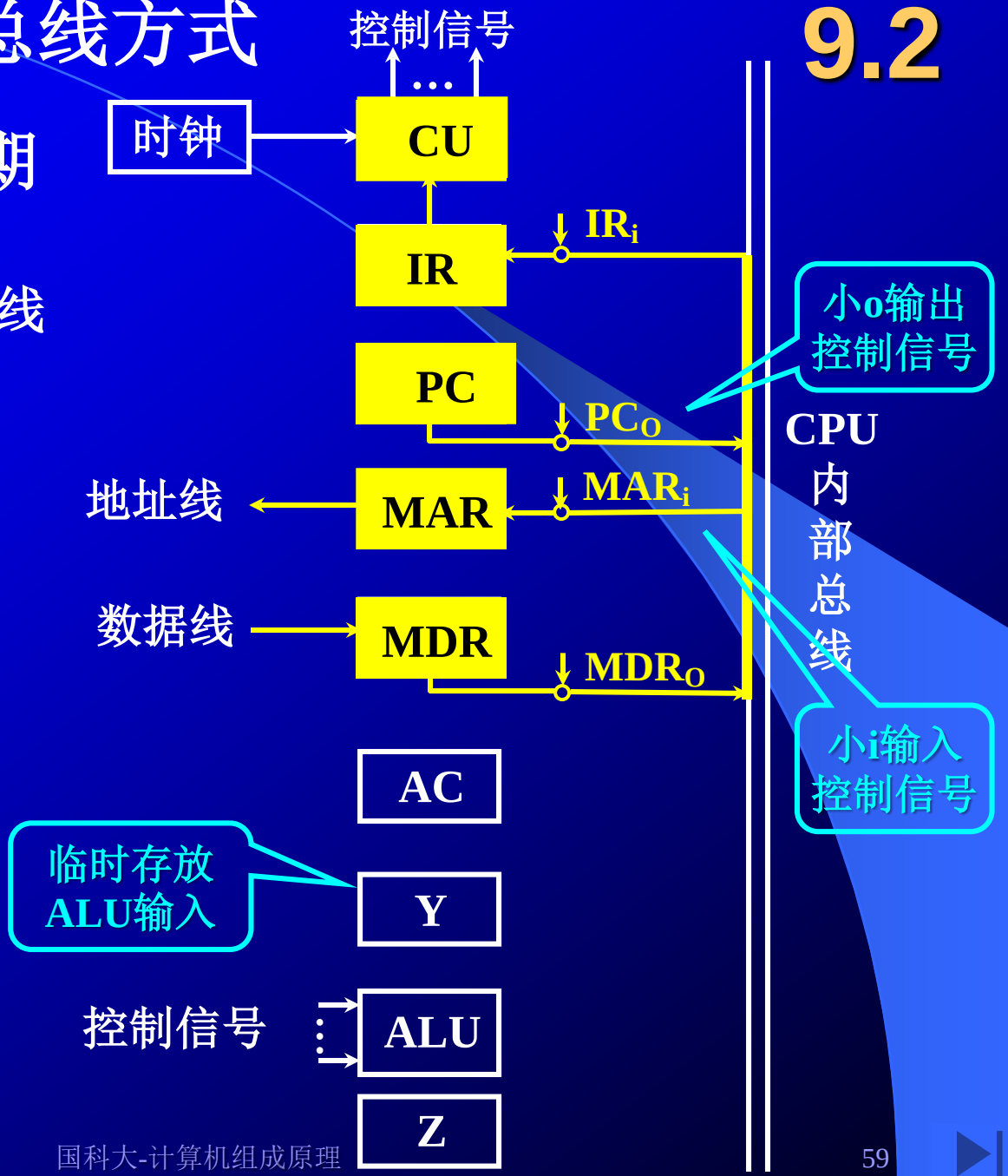
1. 不采用 CPU 内部总线的方式



2. 采用 CPU 内部总线方式

(1) ADD @ X 取指周期

- PC ⟶ MAR ⟶ 地址线
  PC_O MAR_i
- CU 发读命令 1 ⟶ R
- 数据线 ⟶ MDR
- MDR ⟶ IR
  MDR_O IR_i
- OP（IR）⟶ CU
- （PC）+ 1 ⟶ PC

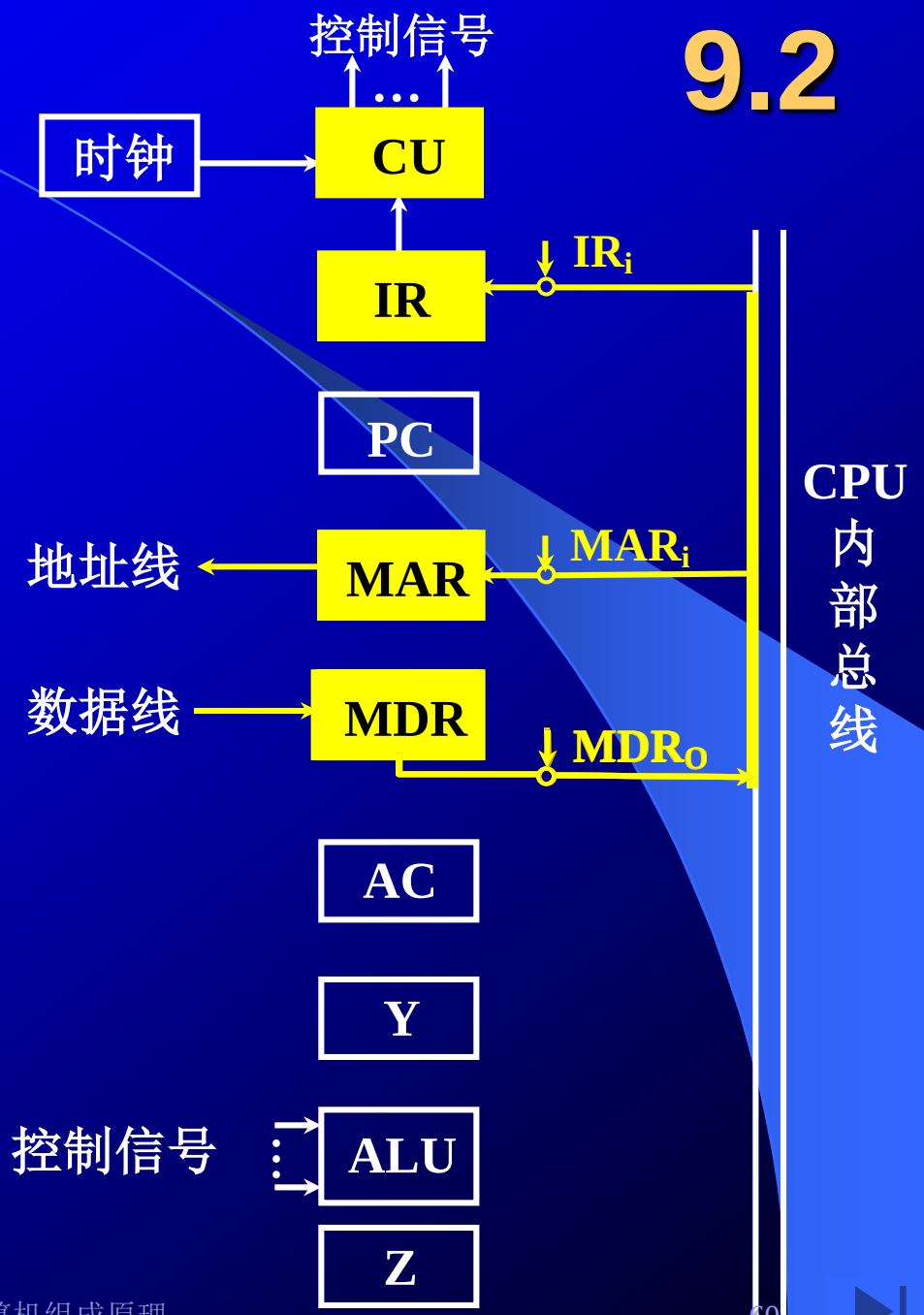


(2) ADD @ X 间址周期

形式地址 ⟶ MAR

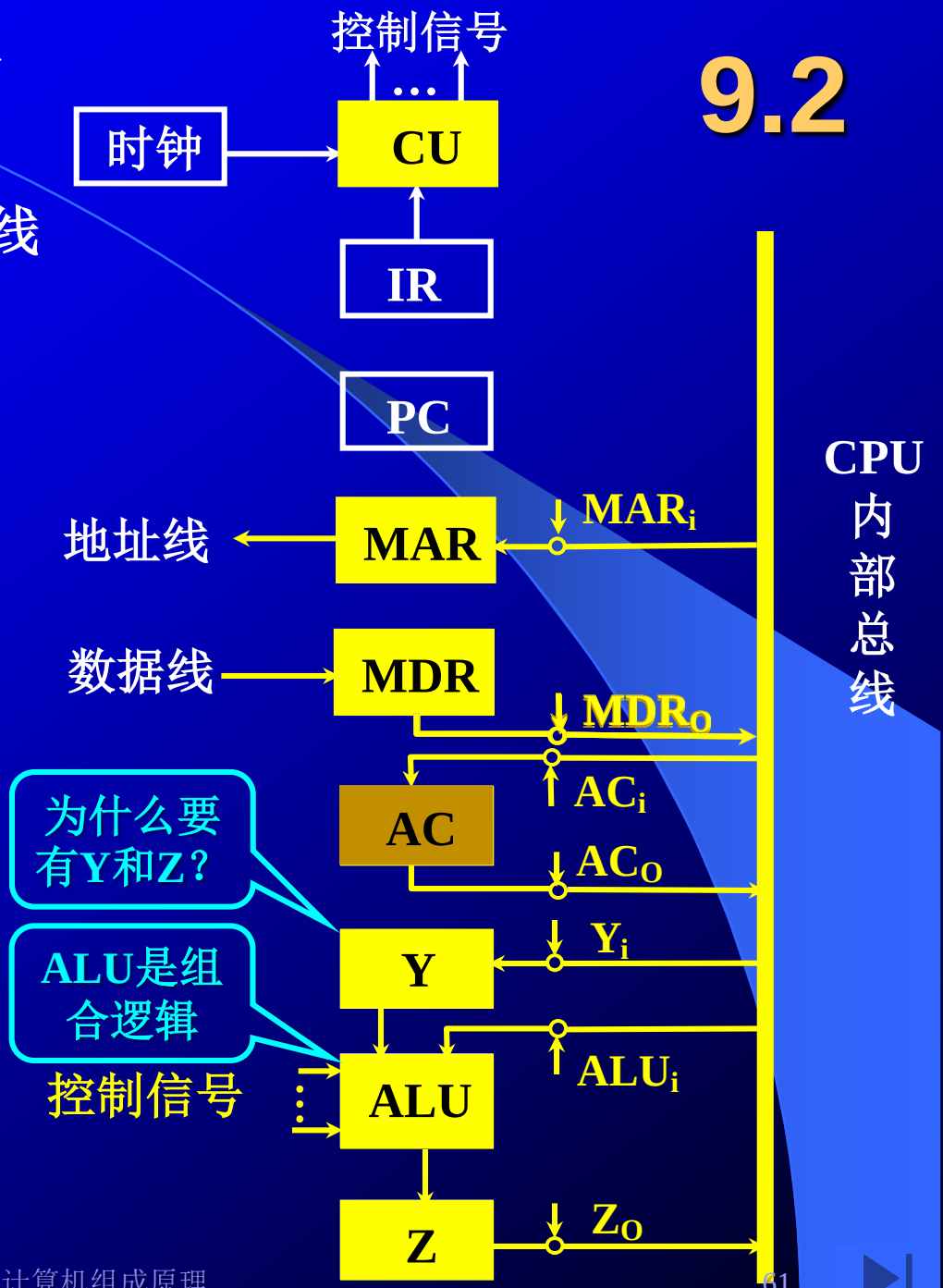
- MDR ⟶ MAR ⟶ 地址线
  MDR_O MAR_i
- 1 ⟶ R
- 数据线 ⟶ MDR
- MDR ⟶ IR
  MDR_O IR_i

有效地址 ⟶ Ad（IR）



(3) ADD @ X 执行周期

- MDR ⟶ MAR ⟶ 地址线
  MDR_O MAR_i
- 1 ⟶ R
- 数据线 ⟶ MDR
- MDR ⟶ Y ⟶ ALU
  MDR_O Y_i
- AC ⟶ ALU
  AC_O ALU_i
- （AC）+（Y）⟶ Z
- Z ⟶ AC
  Z_O AC_i



三、多级时序系统

9.2

1. 机器周期

(1) 机器周期的概念

所有指令执行过程中的一个基准时间

(2) 确定机器周期需考虑的因素

每条指令的执行 步骤

每一步骤 所需的 时间

(3) 基准时间的确定

- 以完成 最复杂 指令功能的时间 为准
- 以 访问一次存储器 的时间 为基准

若指令字长 = 存储字长　　取指周期 = 机器周期

2. 时钟周期（节拍、状态）

一个机器周期内可完成若干个微操作

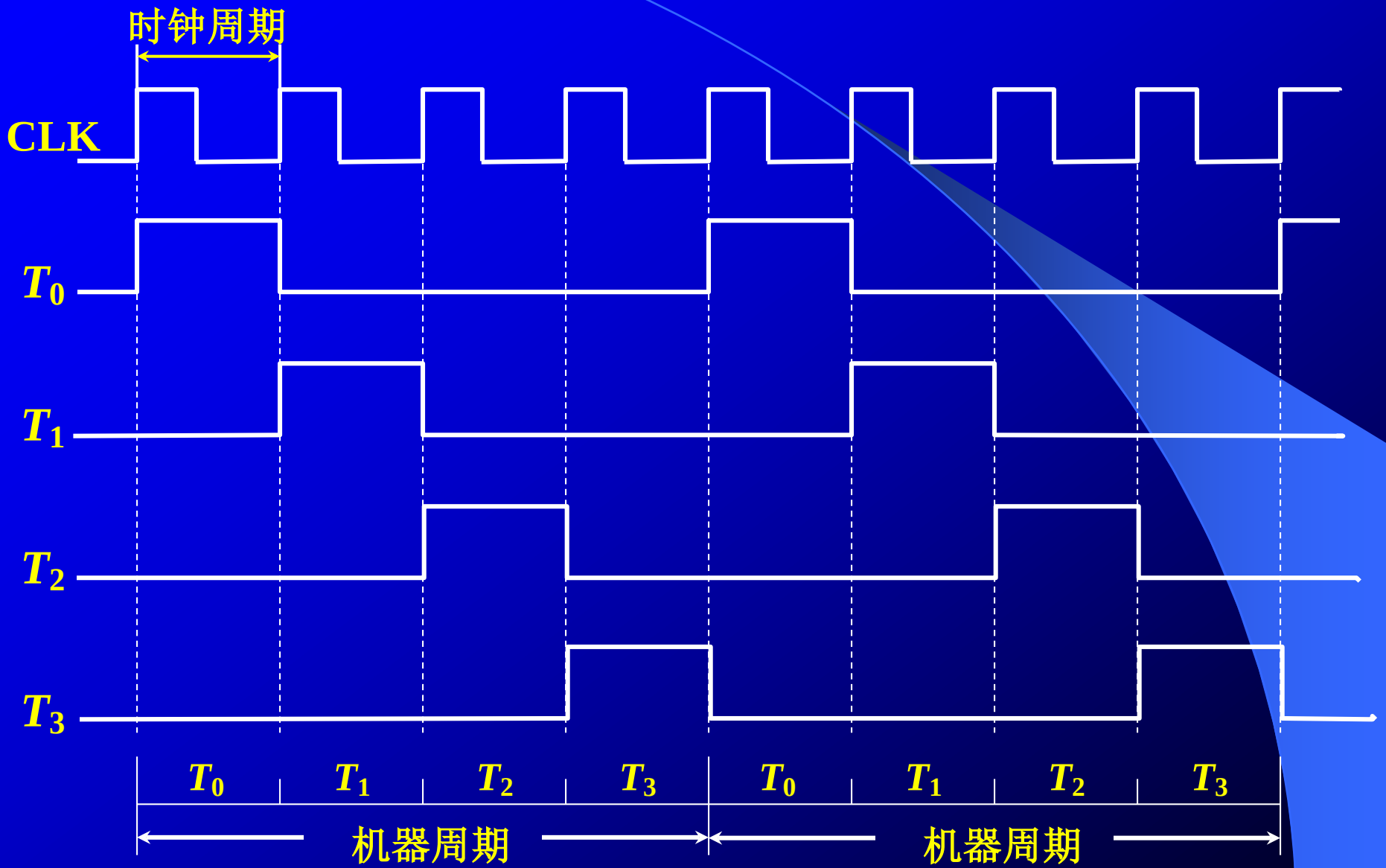
每个微操作需一定的时间

将一个机器周期分成若干个时间相等的时间段（节拍、状态、时钟周期）

时钟周期是控制计算机操作的最小单位时间

用时钟周期控制产生一个或几个同时的微操作命令

2. 时钟周期（节拍T_i、状态）



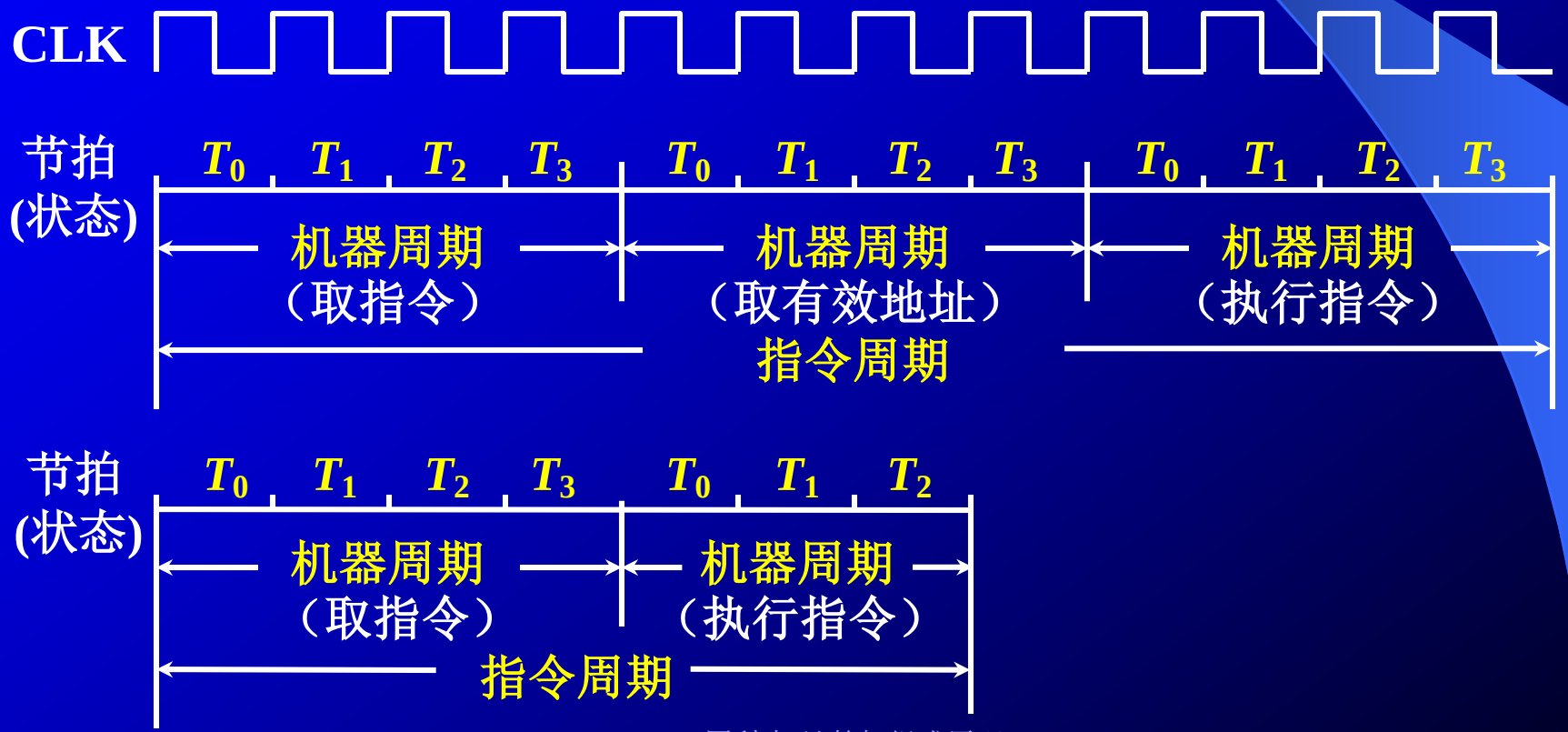
3. 多级时序系统

9.2

机器周期、节拍（状态）组成多级时序系统

一个指令周期包含若干个机器周期

一个机器周期包含若干个时钟周期



4. 机器速度与机器主频的关系 9.2

机器的 主频 f 越快 机器的 速度也越快

在机器周期所含时钟周期数 相同 的前提下，
两机 平均指令执行速度之比 等于 两机主频之比

$$\frac{\text{MIPS}_1}{\text{MIPS}_2} = \frac{f_1}{f_2}$$

机器速度 不仅与 主频有关，还与机器周期中所含 时钟周期（主频的倒数）数 以及指令周期中所含的 机器周期数有关

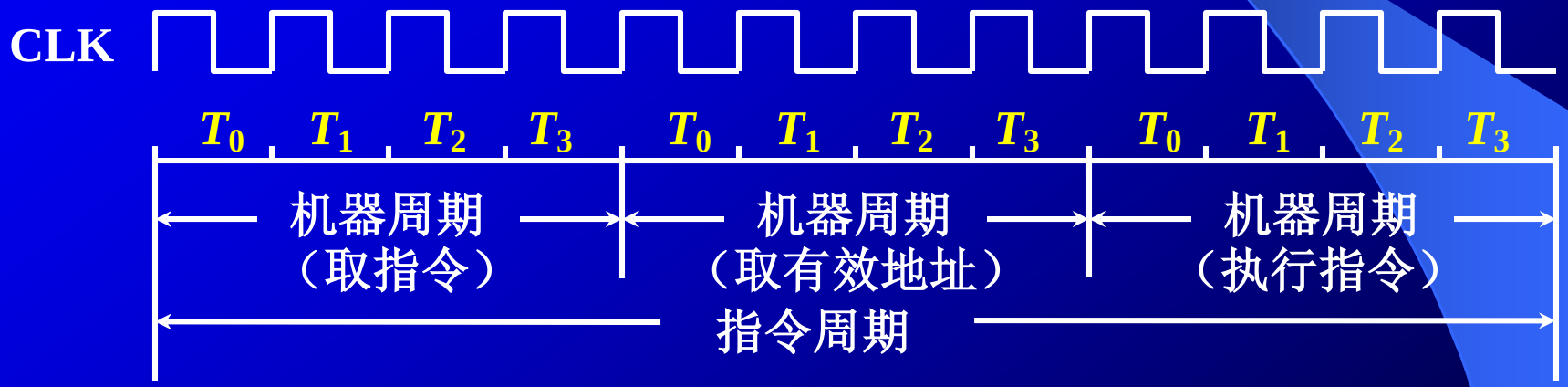
四、控制方式

9.2

产生不同微操作命令序列所用的时序控制方式

1. 同步控制方式

任一微操作均由 统一基准时标 的时序信号控制



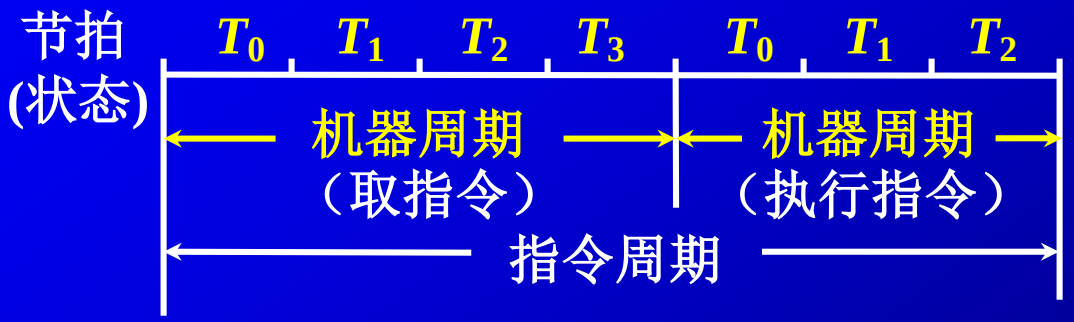
(1) 采用 定长 的机器周期

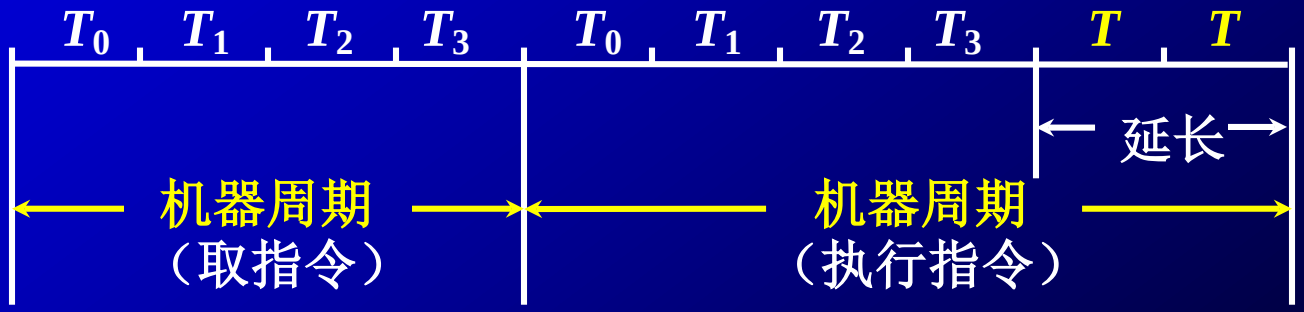
以 最长 的 微操作序列 和 最繁 的微操作作为 标准

机器周期内 节拍数相同

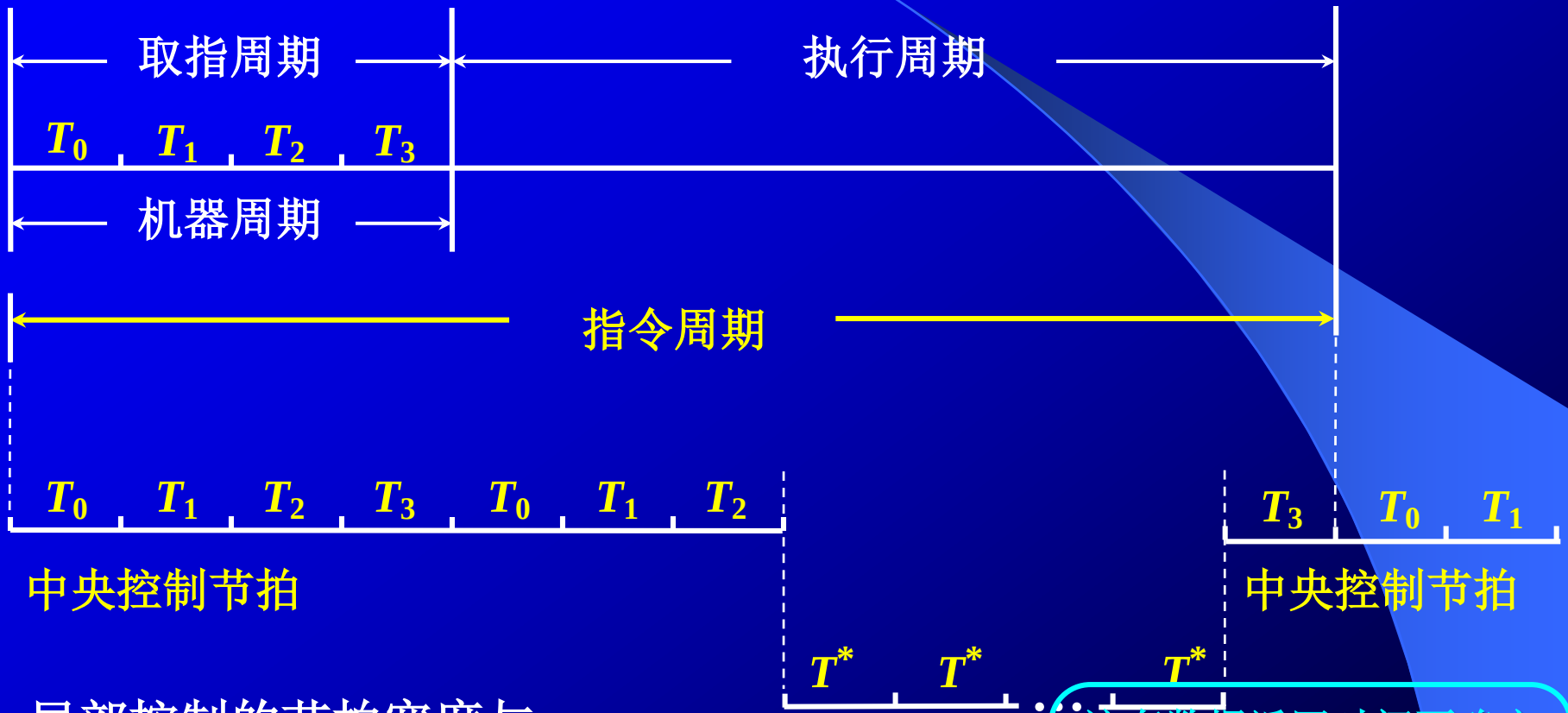
(2) 采用不定长的机器周期

机器周期内 节拍数不等





(3) 采用中央控制和局部控制相结合的方法 9.2



局部控制的节拍宽度与

中央控制的节拍宽度一致

访存数据返回时间不确定

乘法位数不确定

浮点移位不确定

→T^*个数不确定

2. 异步控制方式

无基准时标信号

无固定的周期节拍和严格的时钟同步

采用 应答方式

3. 联合控制方式

同步与异步相结合

I/O操作时间不确定，使用异步应答信号

4. 人工控制方式

(1) Reset

单步调试

(2) 连续 和 单条 指令执行转换开关

(3) 符合停机开关

自学第9.2.5小节的Intel 8085实例

作业

- 习题：9.1，9.3，9.6，9.12，9.14，~~9.8(第一章作业已布置)~~
- 本次作业提交截止时间：
  - 请于5月13日上课前提交
- 复习教材第9章，预习第10章

Q & A ?


中国科学院大学
University of Chinese Academy of Sciences
ICT
中科院计算所
INSTITUTE OF COMPUTING TECHNOLOGY,CAS