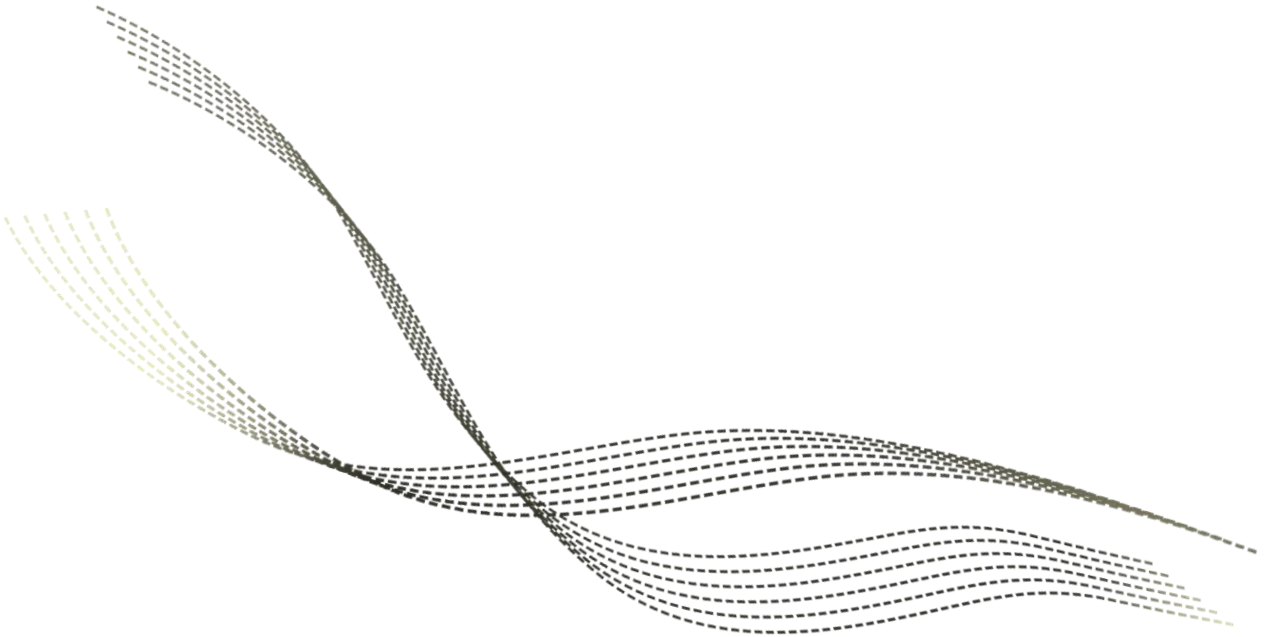
高级计算机体系结构

Advanced Computer Architecture



缓存和主存系统

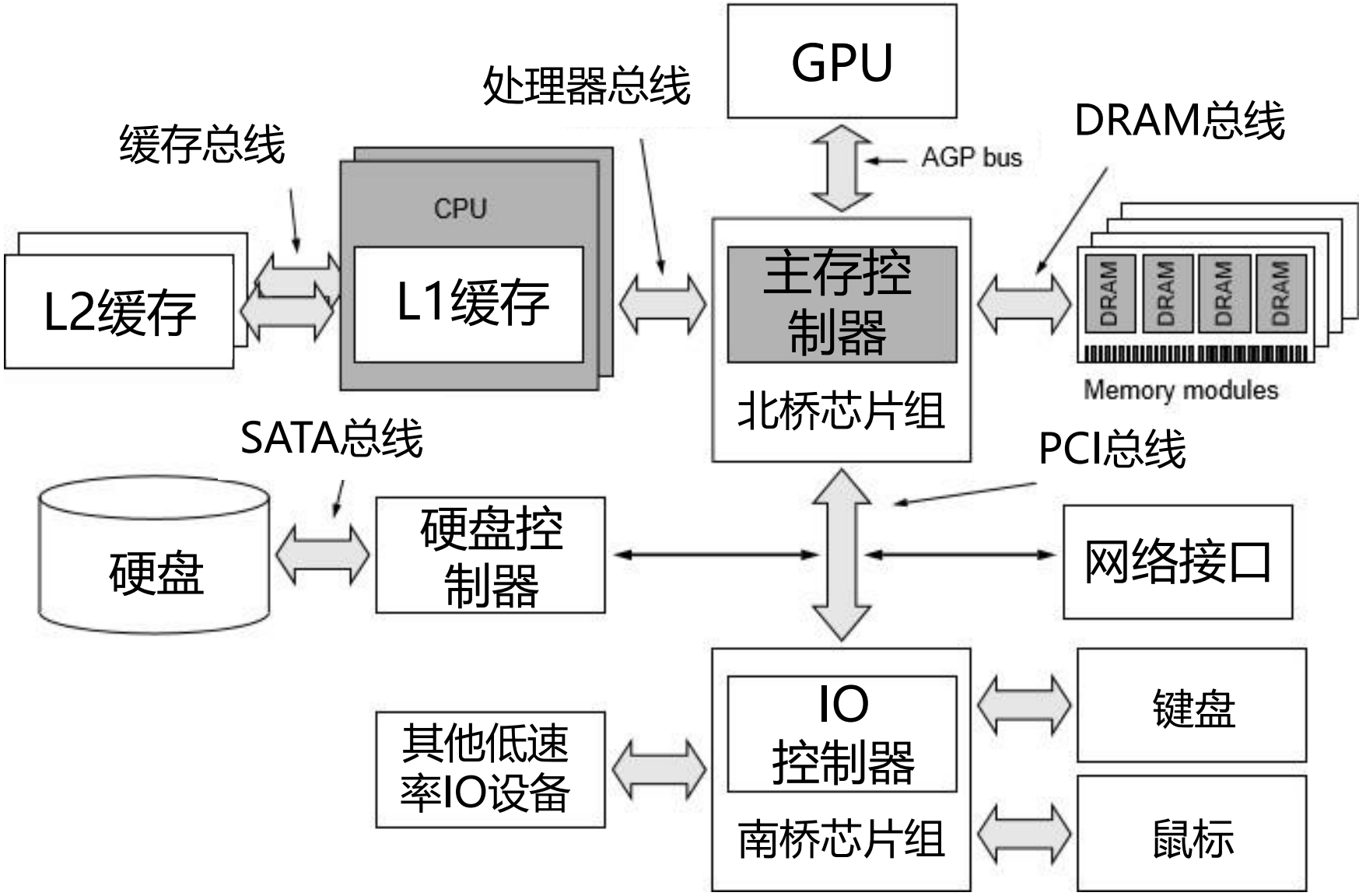
沈明华

目录

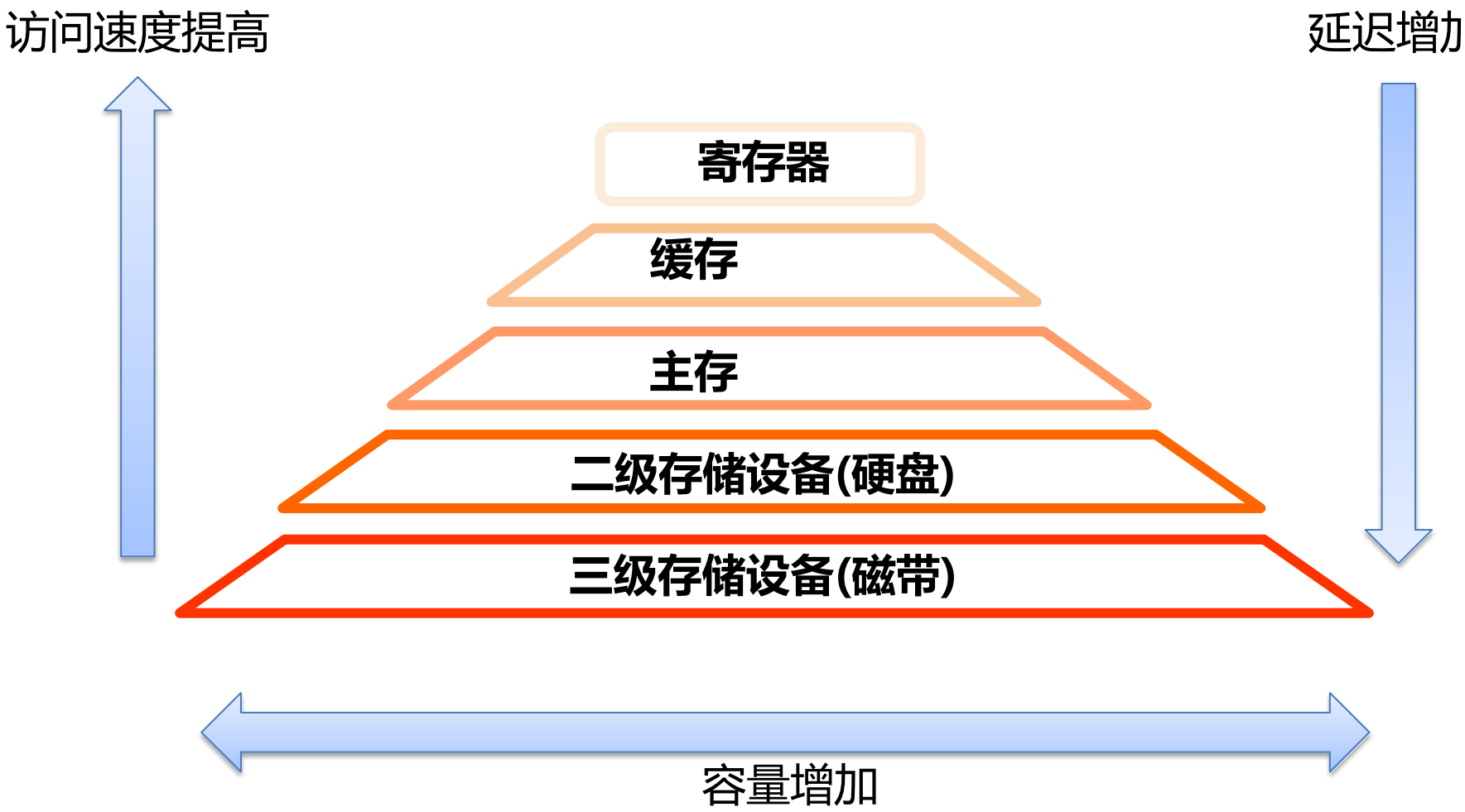
CONTENTS

PART 01 内存层次结构

■ 经典PC构成


GPU
处理器总线
缓存总线
AGP bus
DRAM总线
CPU
L2缓存
L1缓存
主存控制器
北桥芯片组
DRAM
DRAM
DRAM
DRAM
Memory modules
SATA总线
PCI总线
硬盘
硬盘控制器
网络接口
IO控制器
南桥芯片组
其他低速率IO设备
键盘
鼠标

■ 内存层次结构



■ 参数对比

Level	1	2	3	4
Called	Registers	Cache	Main memory	Disk storage
Typical size	< 1 KB	< 16 MB	< 16 GB	> 100 GB
Implementation technology	Custom memory with multiple ports, CMOS	On-chip or off-chip CMOS SRAM	CMOS DRAM	Magnetic disk
Access time (in ns)	0.25 -0.5	0.5 to 25	80-250	5,000,000
Bandwidth (in MB/sec)	20,000-100,000	5,000-10,000	1000-5000	20-150
Managed by	Compiler	Hardware	Operating system	Operating system/operator
Backed by	Cache	Main memory	Disk	CD or Tape

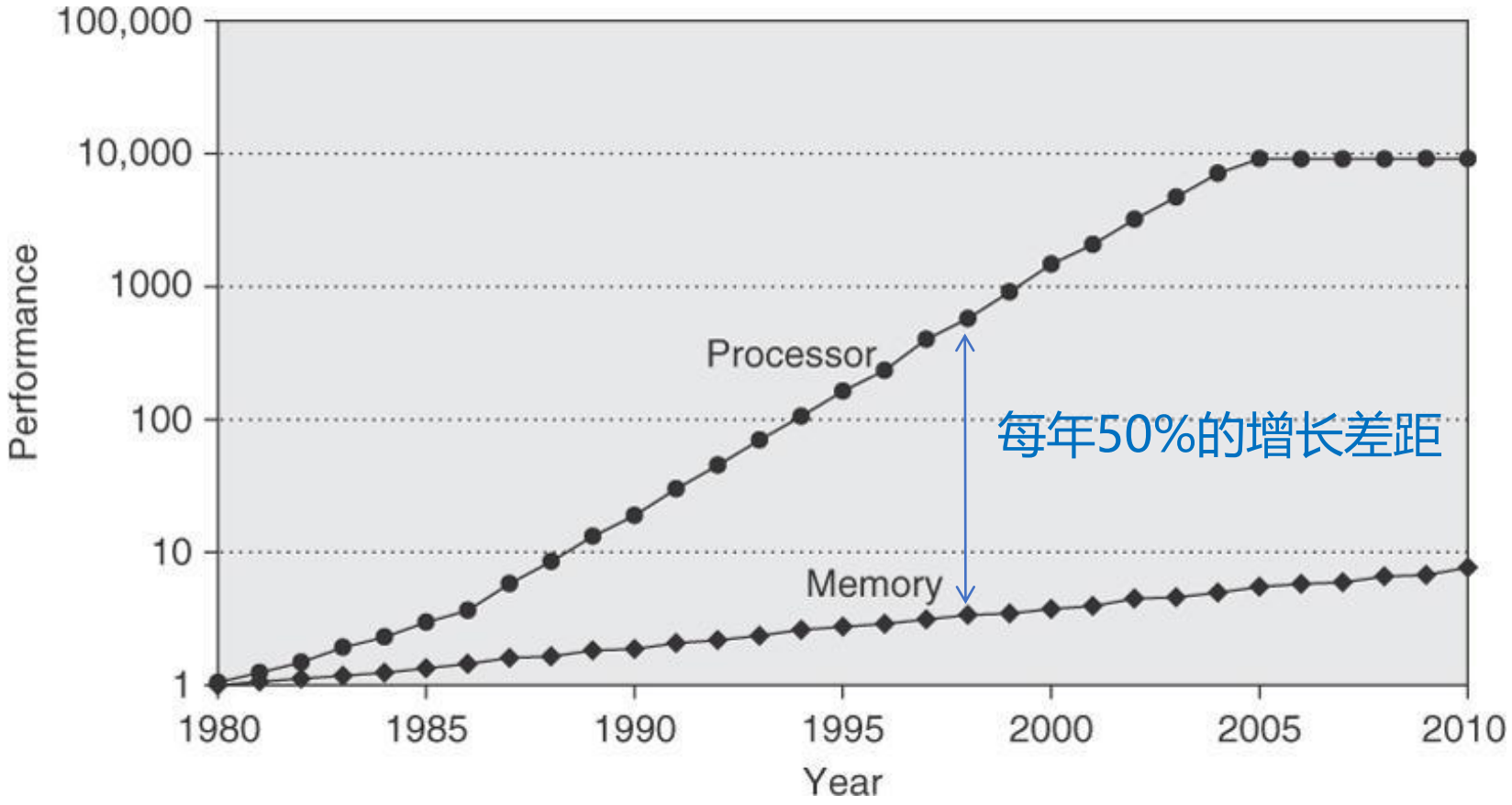
	Access type	Capacity	Latency	Bandwidth	Cost/MB
CPU registers	Random	64–1024 bytes	1–10 ns	System clock rate	High
Cache memory	Random	8–512 KB	15–20 ns	10–20 MB/s	$500
Main memory	Random	16–512 MB	30–50 ns	1–2 MB/s	$20–50
Disk memory	Direct	1–20 GB	10–30 ms	1–2 MB/s	$0.25
Tape memory	Sequential	1–20 TB	30–10,000 ms	1–2 MB/s	$0.025

PART 02

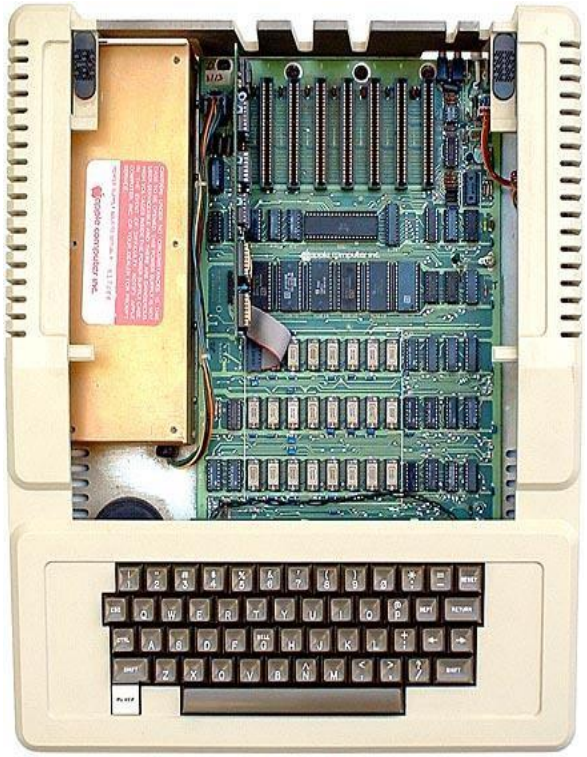
Cache基础

■ 背景

- 在1977年以前，DRAM的速度比CPU快
  - 但后来CPU的增长速率远远超过DRAM



Apple (1977)
CPU: 1000 ns
DRAM: 400 ns

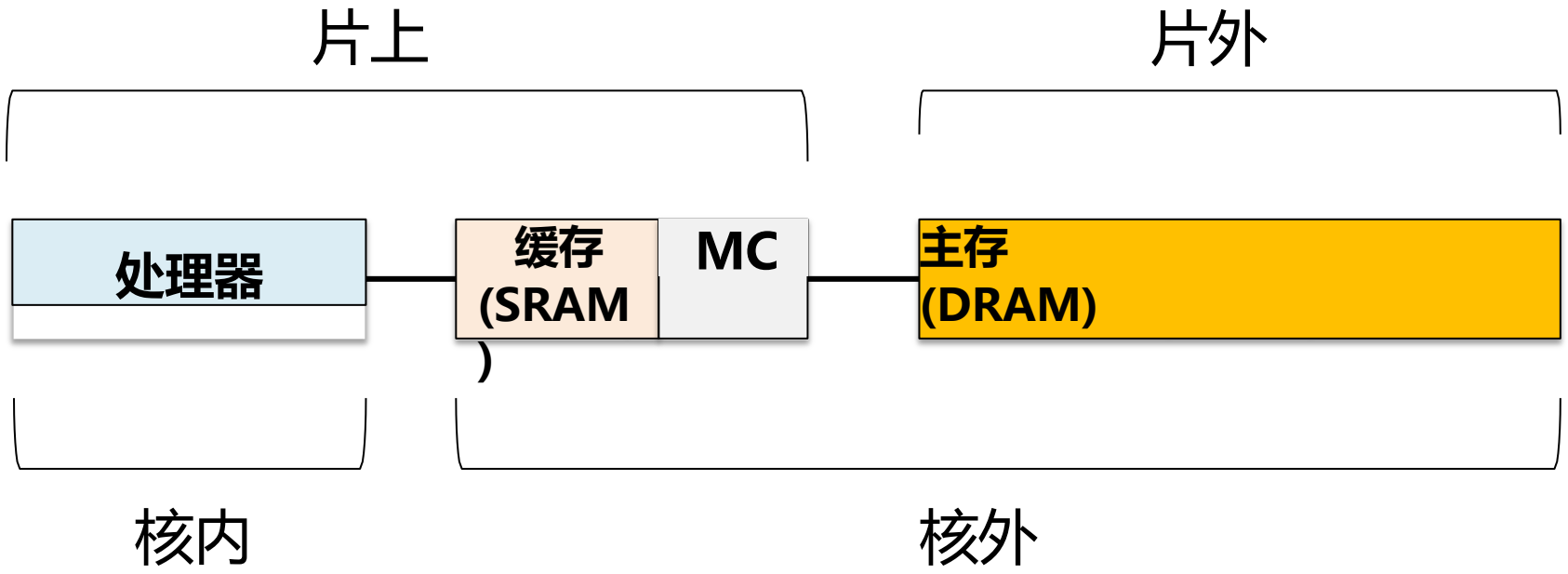


■ 背景

- DRAM的缓慢成为拖累CPU性能的瓶颈
  - 人们需要想办法设计一种新型的存储架构
  - 希望这个新架构具备以下特征
    - 速度要足够快，最好和寄存器一样快
    - 容量要足够大，最好和磁盘一样大
    - 价格要足够低
  - 最后人们设计出多级缓存架构，某种意义上完成这个不可能三角
    - 在DRAM和寄存器之间插入缓存(cache)
    - intel在1989年发布的80486是第一款带缓存的CPU

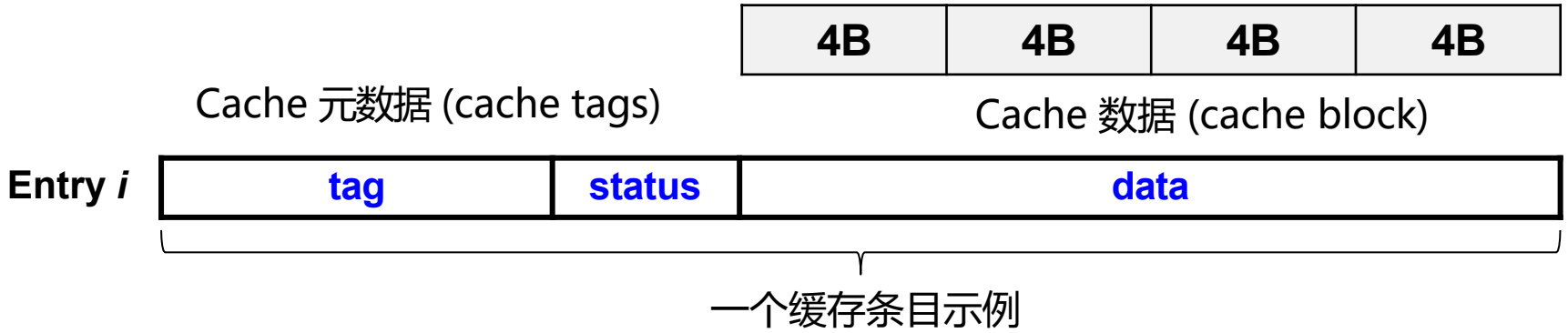
■ 基本概念

- 片上资源
  - Cache(SRAM)，相对于DRAM速度更快，容量更小
- 片外资源
  - Memory(DRAM)，容量远大于Cache，速度更慢



■ 缓存与数据访问粒度

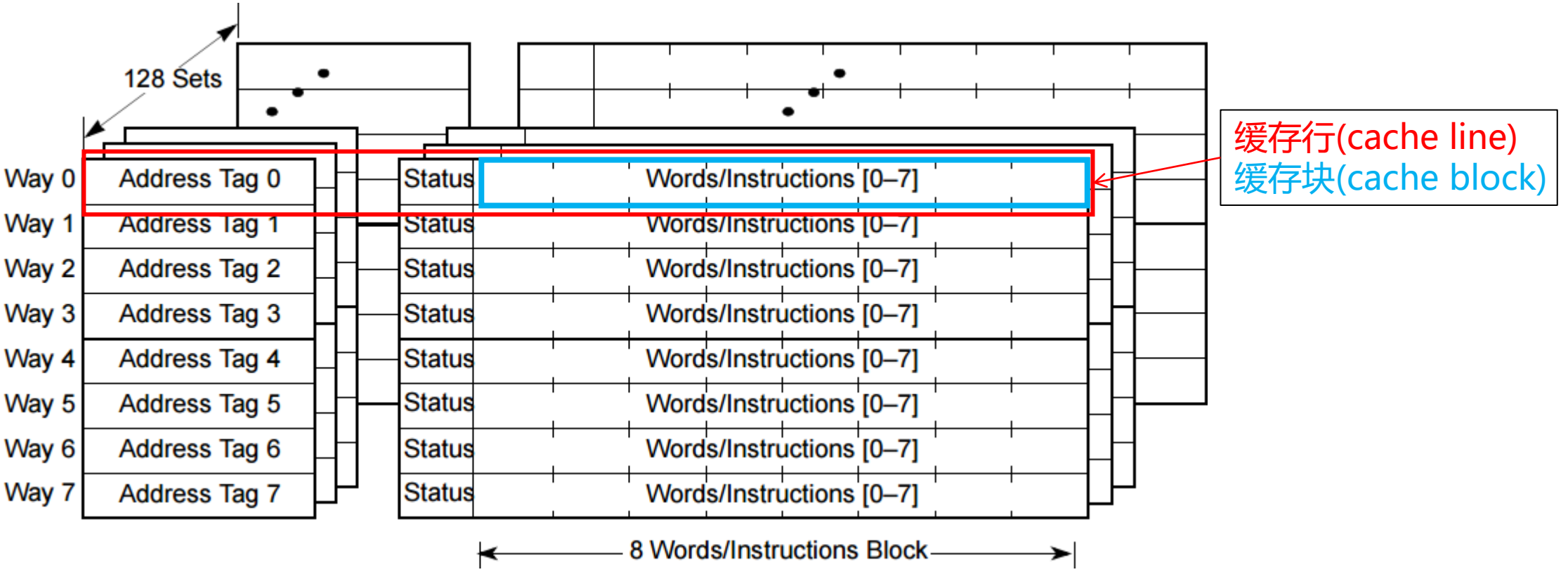
- Cache(高速缓存)储存处理器频繁访问的数据块
  - 缓存中存放数据块的结构称为**缓存行**(cache line)
  - 缓存行存放的数据块称为**缓存块**(cache block)
- 缓存条目(Cache Entry)中通常包含以下字段
  - 标签(tag)：标记数据的原地址
  - 状态(status)：通常包含一个有效位、一个脏位
  - 数据(data)：缓存中的数据，一般是16字节大小，因CPU而异



■ L1 Cache

□ L1 Cache

- 128个缓存组，每组包括8个缓存行，每个缓存块为8个字
- 32位系统上一个字的大小为4字节



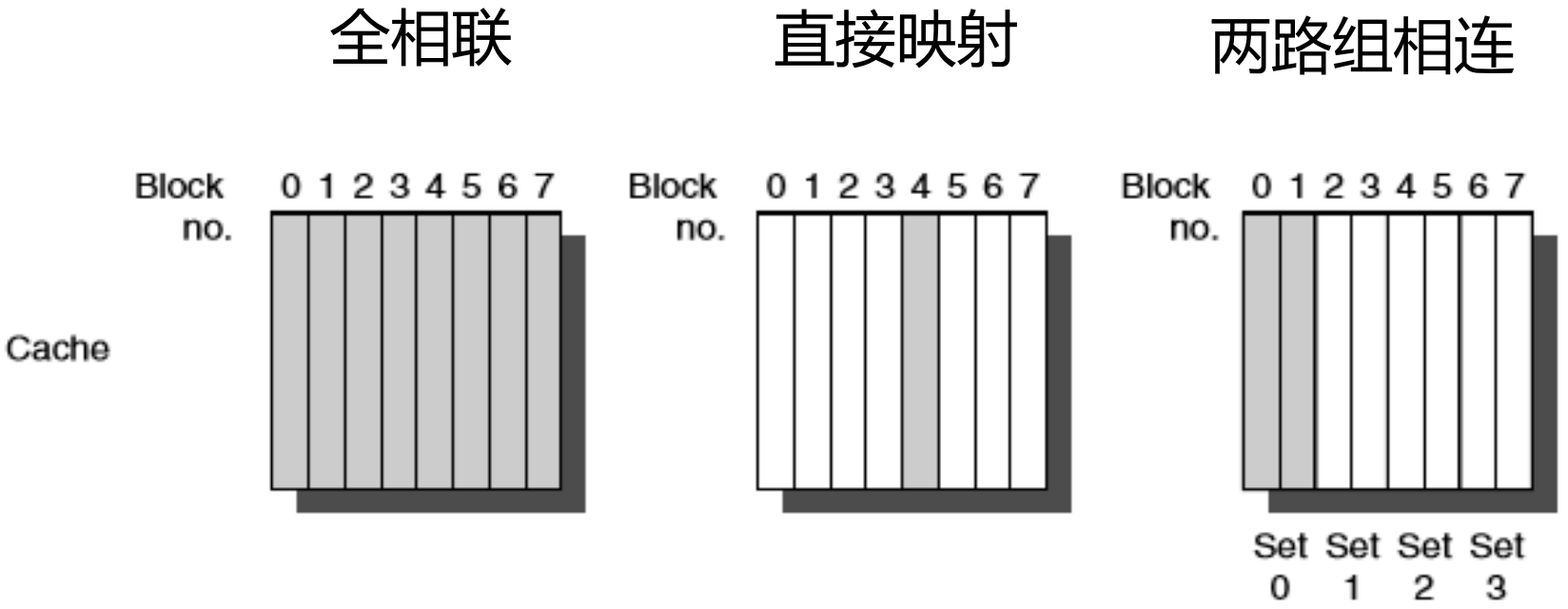
■ 缓存的逻辑结构

- 直接映射(directed mapping)
  - 数据块只能放到缓存中的特定位置
  - 一个缓存行就是一个组
- 全相连(fully associative)
  - 数据块可以放置在缓存的任意位置
  - 所有缓存行属于同一个组
- 组相连(set associative)
  - 数据块只能被放置在缓存中一组受限制的位置
  - 四路组相连：每组包含四个缓存行

■ 缓存的逻辑结构

◘ 映射示意

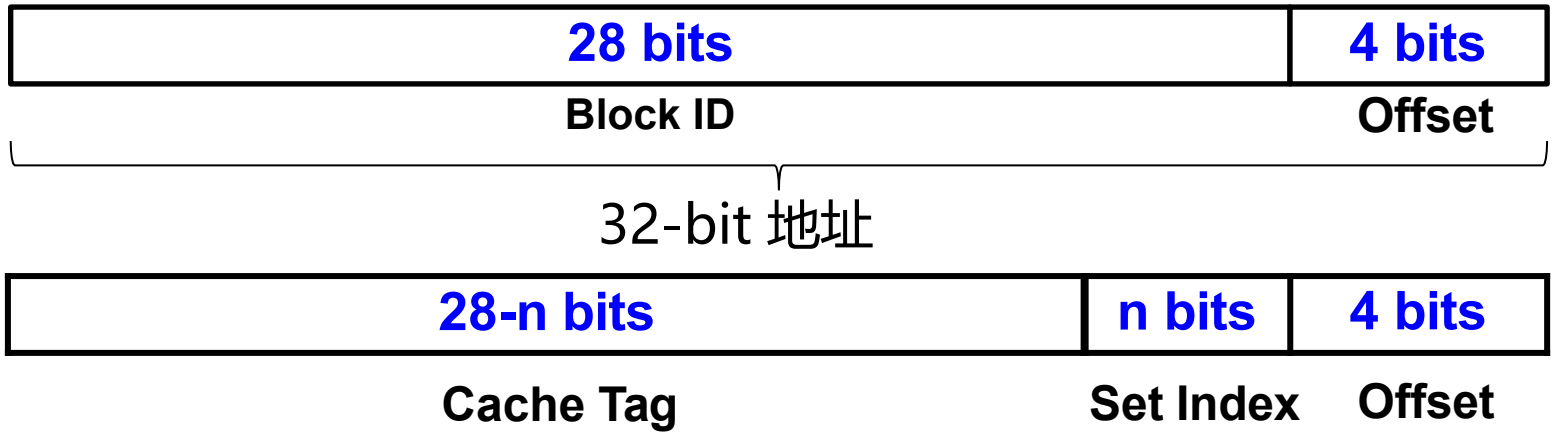
- 主存中地址为12的块，可以映射到全相联缓存的任意一个缓存行
- 而在直接映射中只能放在编号为4的行，因为12 mod 8 = 4
- 两路组相连可以放在第0组，因为12 mod 4 = 0



■ 定位缓存块

□ 划分一个32位地址

- 偏移量(Offset)
  - ➢ 定位块内字节，4-bit 的 Offset 表示缓存块的大小为16字节
  - ➢ 另外28bit的地址作为Block ID
- 组索引(Set Index)，若缓存为组相连结构
  - ➢ 组索引用于确定数据所在的组，n-bit的组索引表示有 2^n 个缓存组
  - ➢ 剩余28-n bit作为缓存的tag，用于匹配地址防止访问错误的数据
  - ➢ 直接映射可以看作是组大小为1的组相连，此时n=0，tag有28bit



■ 缓存的逻辑结构

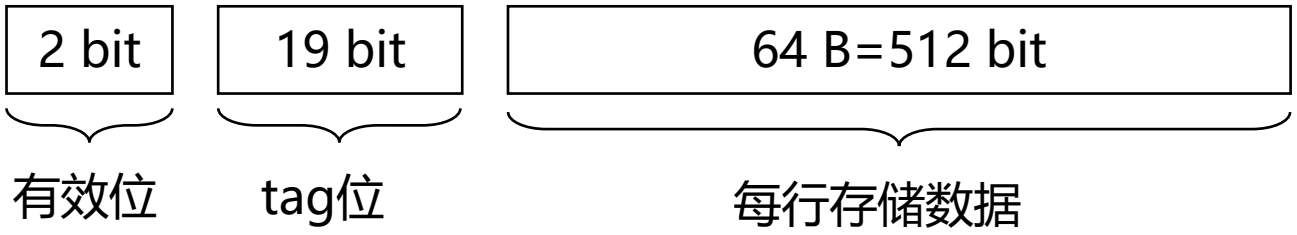
□ 例题

- 若某个计算机主存地址空间大小为256MB，按字节编址。其缓存一共8个缓存行，行长64B
  - 问：若不考虑缓存的替换算法控制位(只考虑有效位、脏位)，采用直接映射方式，则该缓存的总容量是多少?
  - 问：采用直接映射方式，主存地址3200(十进制)对应的Cache行号是多少？如果是二路组相连呢?

■ 缓存的逻辑结构

□ 例题解答1

- 问1：若不考虑Cache的替换算法控制位(只考虑有效位、脏位)，采用直接映射方式，则该缓存的总容量是多少?
- 答：总数据容量为 4264 个 bit
  - 缓存总容量包括：存储容量、tag位容量、控制位容量(脏位和有效位)
  - 标记位：主存地址有28位(256MB=2^{28}B)，其中6位是块内地址(64B=2^6B)，3位是行号(采用直接映射，8行等于2^3)。剩余28-6-3=19位是cache tag标记位
  - 总容量：$8 \times (2 + 19 + 512) = 4264$ bit



思考：若采用二路组相连，标记(tag)位应该是几位?

■ 缓存的逻辑结构

□ 例题解答2

- 问2：采用直接映射方式，主存地址3200(十进制)对应的Cache行号是多少？如果是二路组相连呢？
- 答：行号是2；二路组相连时行号为4或5
  - 主存地址3200对应的块号(block ID)为$3200/64 = 50$。而缓存只有8行，50 mod 8 = 2，因此对应的行号是2
  - 若采用二路组相连的结构，50 mod 4 = 2，对应的组号为2。第二组对应的行号是4和5

局部性原则

- 时间局部性
  - 程序会重复访问同一数据
  - 例如循环中的变量
- 空间局部性
  - 程序会倾向于访问相邻内存中的数据
  - 例如矩阵求和时，顺序访问每一个元素

■ 降低缓存失效率

□ 缓存有三种失效情形

- 强制失效
  - 首次访问某内存块时，缓存中一定不存在该内存块
  - 如程序的冷启动
- 容量失效
  - 缓存容量不足，活跃数据块数量超出缓存行数量
  - 需要频繁替换缓存
- 冲突失效
  - 多个块竞争同一组缓存，导致内存块被频繁替换
  - 注意和容量失效区分，缓存行数量充足时也有可能发生冲突失效
- 缓存失效意味着CPU需要从更慢的主存中取数据
- 有必要采取措施降低缓存失效率，从而提高存储系统的总体性能

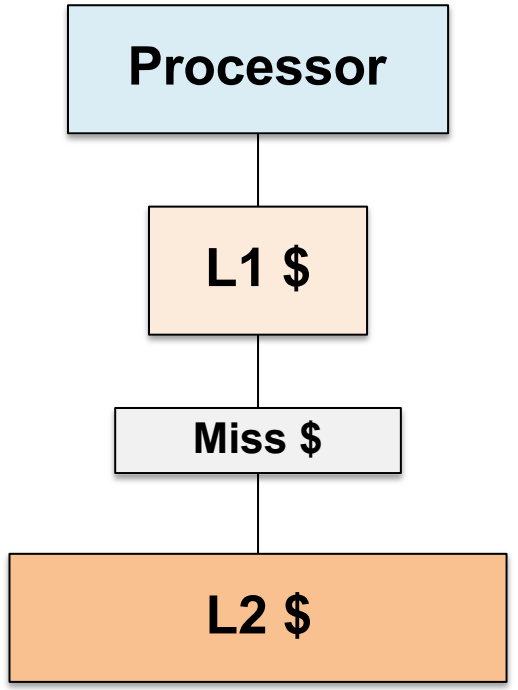
■ 缺失缓存

□ 缺失缓存(Miss Cache)

- L1缓存和L2缓存之间插入一个高速全相连缓存
- 包含2~5个缓存行的数据
- 目的是减少冲突失效

□ 工作流程

- 当L1缓存失效时，检查缺失缓存。若块存在于缺失缓存，可以直接访问
- 否则从L2缓存中获取，取得数据仅存入缺失缓存，避免频繁的替换L1缓存数据



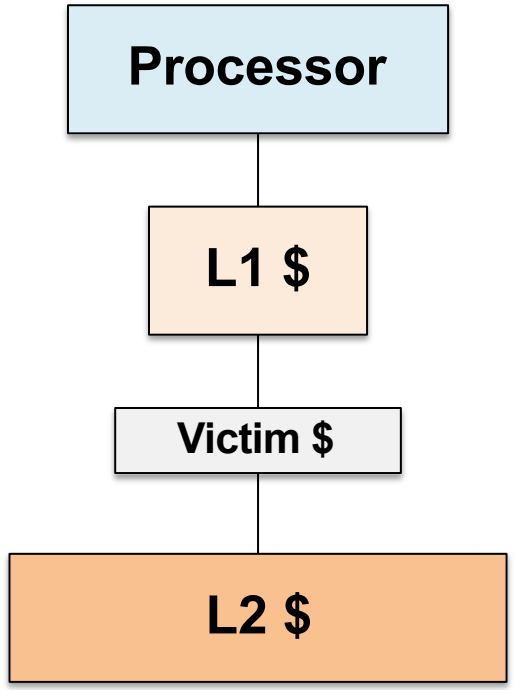
■ 受害者缓存

□ 受害者缓存(Victim Cache)

- 改进替换策略
- 比缺失缓存更小、但性能更好
- 即使只有1个缓存行也能生效

□ 工作流程

- 当L1缓存miss时，检查受害者缓存。若块存在于受害者缓存，将其调入L1缓存，并将替换出的块存入受害者缓存
- 否则从L2缓存中获取数据，存入L1缓存
  - ➢ L1缓存中被剔除的数据进入受害者缓存



■ 缓存预取

□ 预取(prefetching)

- 缺失缓存和受害者缓存适合时间局部性

- 而缓存预取是一种主动提前加载数据/指令到缓存的技术，适合空间局部性
- 适用场景：
  - 分支预测，提前预取预测分支的指令
  - 连续访问场景，提前加载连续内存块
- 存储优化
  - 若直接将预取数据放入L1缓存，因为预取数据不一定被使用，所以会提高缓存缺失率
  - 因此将预取块暂存至专用缓冲区，当处理器真正访问时再移入L1缓存

■ 缓存管理

□ 缓存管理分为三个组成部分

- 分区策略
  - ➢ 决定数据是否被缓存，和数据应该被放置在哪个缓存行中
- 预取策略
  - ➢ 决定是否以及何时将数据读入缓存
- 局部性优化
  - ➢ 通过重新组织代码等手段，最大限度地利用缓存的行为特性，降低缓存缺失率

■ 数据预取的影响

□ 预取器的准确率和覆盖率

- 有用预取：预取的内容最终被处理器访问
- 准确率：有用预取的次数 / 总预取次数
- 覆盖率：有预取时减少的缓存缺失次数 / 无预取时的总缓存缺失次数
  - ➢ 只有准确率评价指标，预取器会倾向于少取，从而达到100%的准确率

■ 缓存的写策略

◘ 缓存命中时的写操作

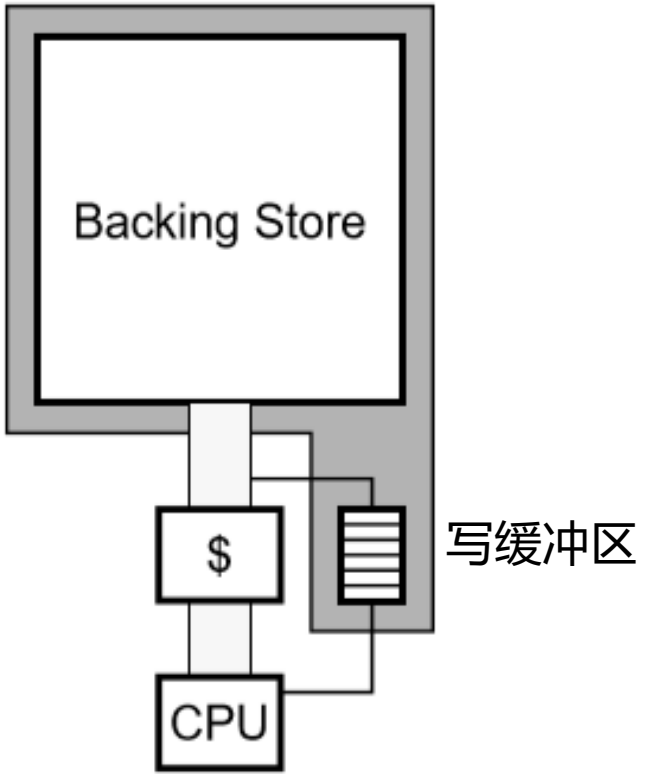
- 写回(write-back)：只将值写入缓存，被修改的块在替换时才写入主存
  - 节约带宽以及能耗
- 写直通(write-through)：每次对缓存的写操作都会同步写入主存
  - 易于实现，缓存数据与主存保持一致

◘ 缓存缺失时的写操作

- 写分配(write allocate)，将主存中的块先加载到缓存，然后写缓存
- 非写分配(no-write allocate)，在更低层级的主存中直接修改该块

缓存的写策略

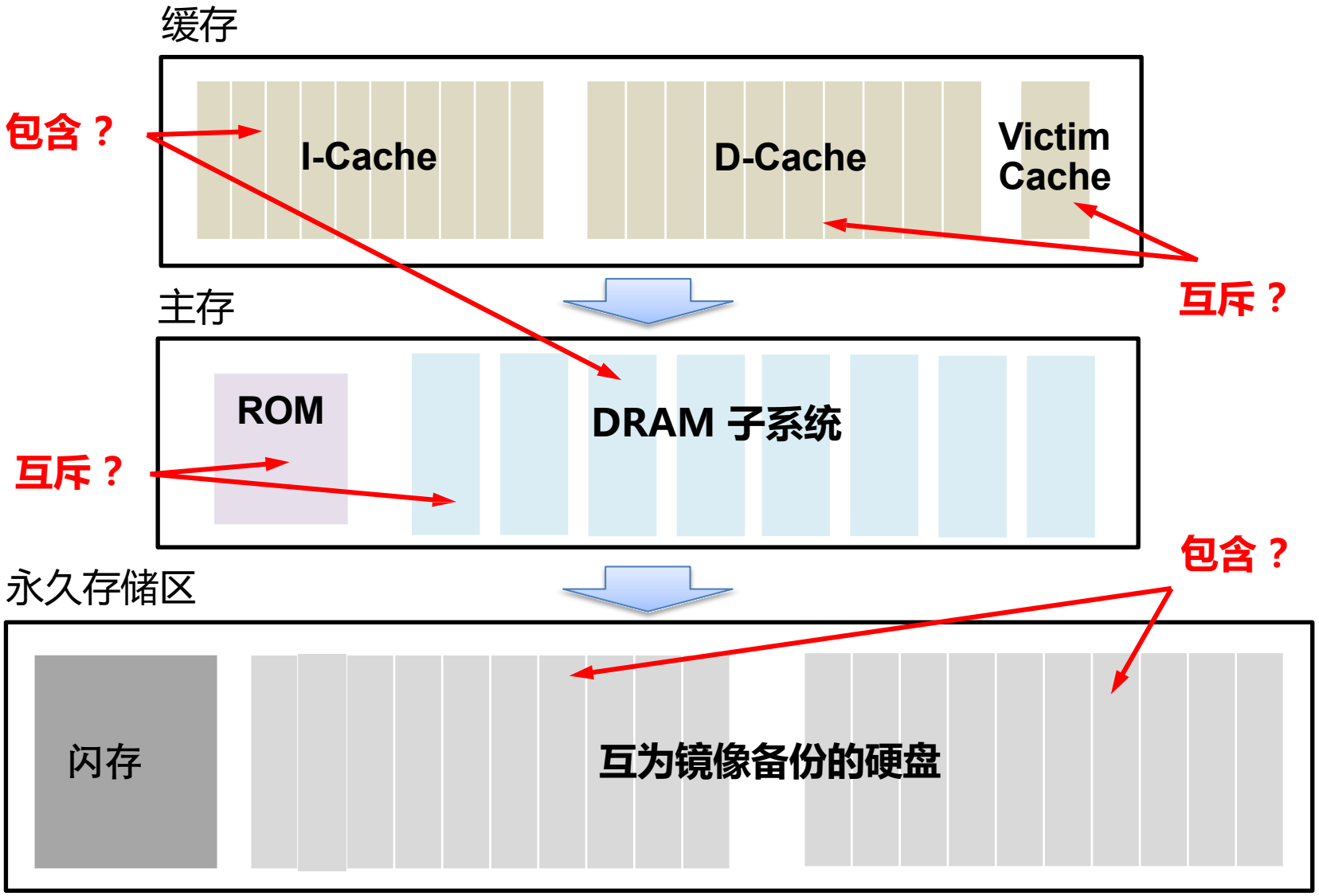
- 将每次写操作都发送到主存的代价很高
  - 解决方案：在系统中插入新存储结构，适配写直通方案
- 写缓冲区(writer buffer)
  - 无标签的FIFO队列，暂存待写主存的数据，缓冲区满或需同步数据时必须将缓冲区的所有内容转储到主存
- 写缓存(write cache)
  - 带标签的缓存结构，暂存待写主存的数据。当需要访问主存时，必须先查询写缓存，避免读取脏数据



■ 一致性管理

- 内存层次结构被垂直划分为不同层级，而每个层级可以进一步被“水平划分”
- 包含(inclusion)和互斥(exclusion)的原则定义了任意两个分区之间存在的特定关系类型
  - 包含关系：分区中的每个缓存项在另一分区中都有副本
  - 互斥关系：两个分区不会有相同的缓存项

■ 存储系统的分层结构


缓存
包含？
I-Cache
D-Cache
Victim Cache
互斥？
主存
ROM
DRAM 子系统
互斥？
包含？
永久存储区
闪存
互为镜像备份的硬盘

■ 存储系统的分层结构

□ 包含与互斥对存储容量的影响

- 问1：在某存储系统中，主存容量为12MB、Cache容量为400KB，则该存储系统容量为?
  - 12MB+400KB?
  - 12MB?
  - 12MB-400KB?
  - 400KB?
- 问2：又是一个存储系统，Cache容量400KB，Victim Cache容量100KB，则该存储系统容量?
  - 400KB+100KB?
  - 400KB?

■ 缓存命中率

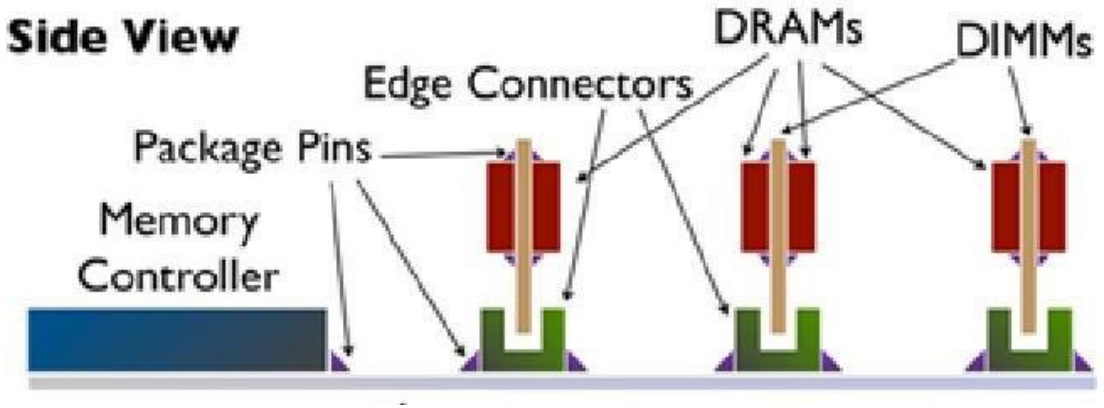
□ 缓存命中率

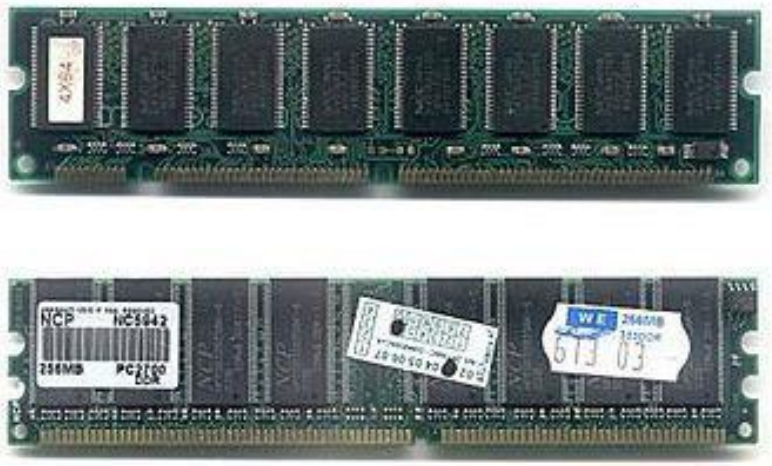
- CPU访问的数据恰好在缓存中的比率称为缓存命中率
  - $H = N_c/(N_c + N_m)$，其中N_c是缓存命中次数，N_m则是未命中次数
- 已知缓存命中率为H，尝试计算缓存+主存系统的平均访问时间
  - 假设t_c为命中缓存的访问时间，t_m是未命中时的访问时间
  - 那么平均访问时间：$T_a = Ht_c + (1 - H)t_m$
- 例题
  - 假设某单缓存主存系统的缓存命中率$H = 95\%$，$t_c = 1$且$t_m = 5$
  - 问性能是没有缓存的同一主存系统的多少倍
  - 答：$\frac{1}{S} = \frac{T_a}{T_m} = \frac{0.95\times t_c + (1-0.95)\times t_m}{1\times t_m}$，得到$S \approx 4.17$倍

PART 03 DRAM基础

■ 主存的物理架构

- 双列直插式内存模块(DIMM, dual in-line memory module)
  - 主存系统通常由多个独立的DIMM组成
  - DIMM包含一个或多个独立的rank
  - rank包含一组共同操作的DRAM芯片
  - DRAM芯片包含一个或多个独立的bank
  - Bank由一个或多个储存阵列(arrays)组成

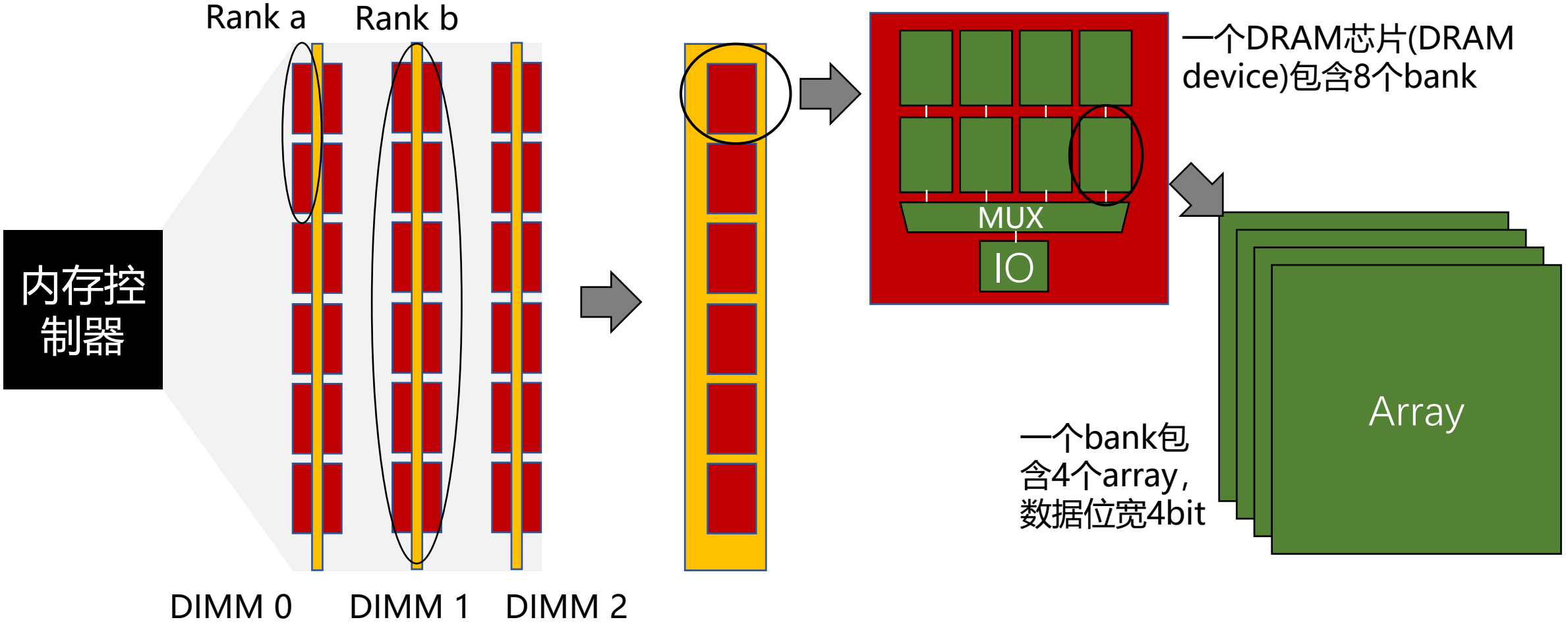




■ 主存的物理架构

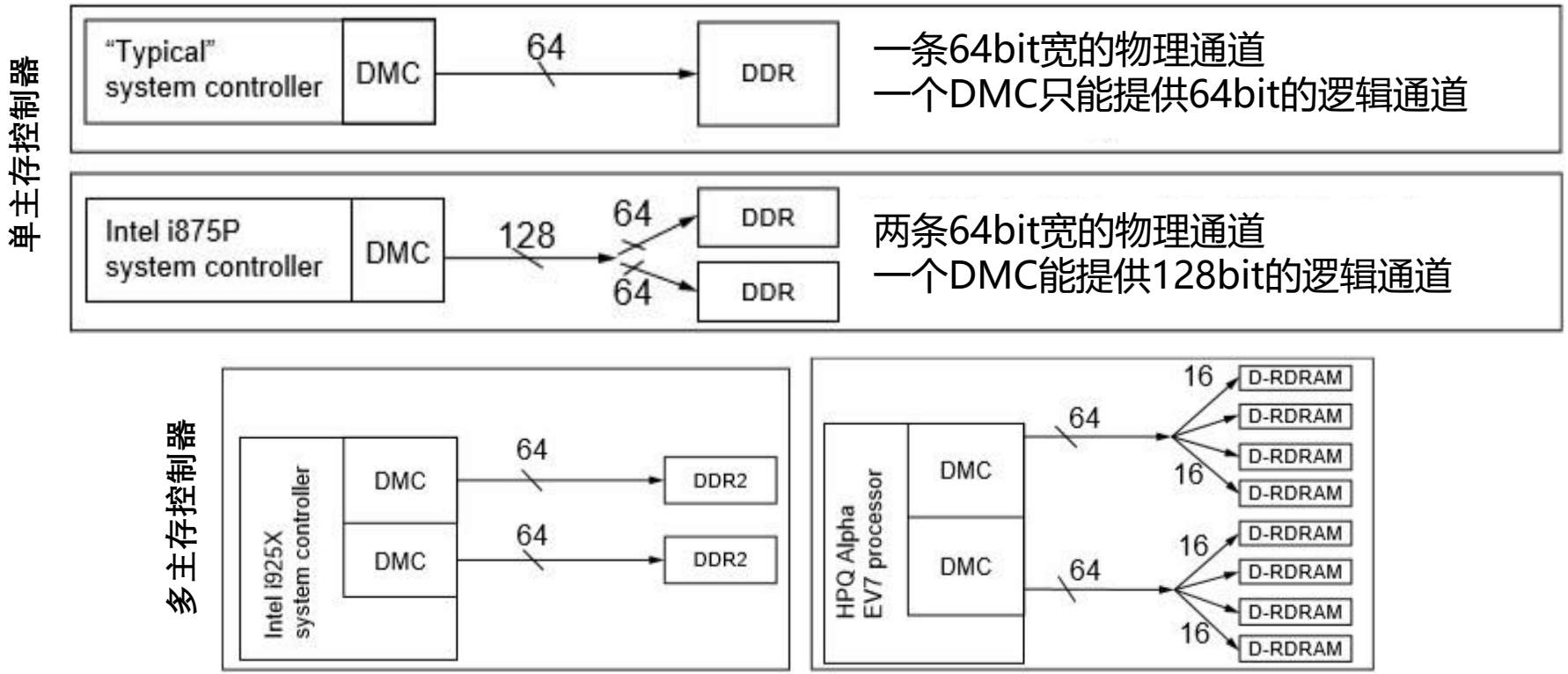
□ 物理架构示意

– Rank形式多种多样，以下Rank a/b都有可能



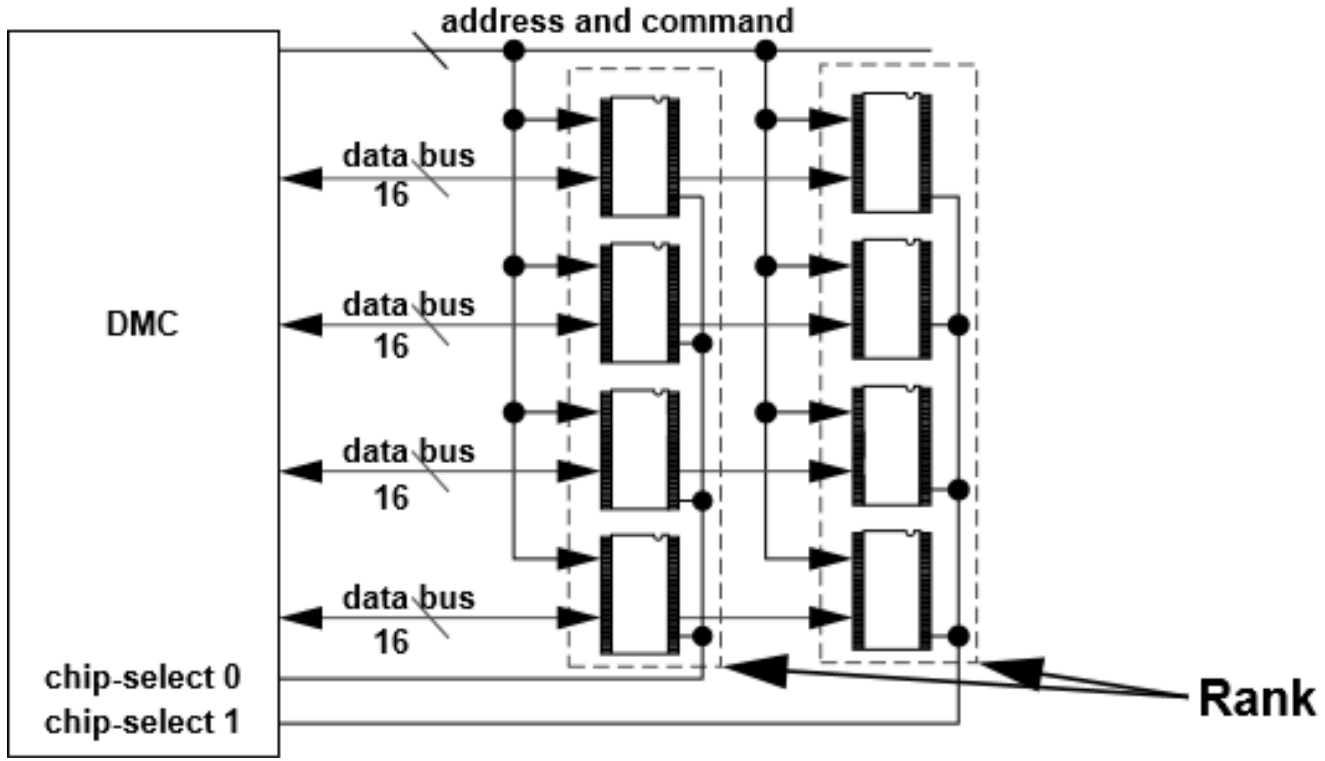
■ 主存通道

- 主存通道(channel)：内存控制器(MC)与DRAM模块之间的数据通路



■ 存储系统

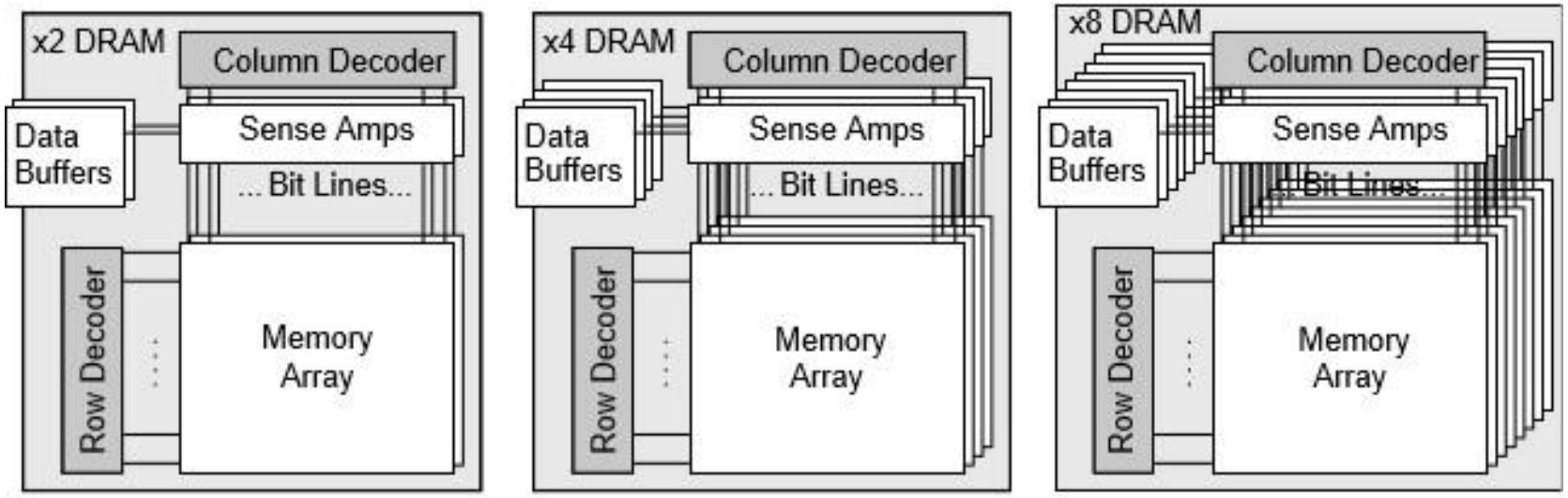
- Rank
  - 同步响应一个指令的一组DRAM芯片集合
- Bank
  - DRAM芯片内部的一组独立内存阵列



■ 内存系统的定义

- DRAM芯片位宽用“xN”描述，N表示输出引脚的数量

- x4 DRAM表示在单个Bank中有4个独立的内存阵列array
  - 对列进行读写操作时，可以一次性传输4bit数据

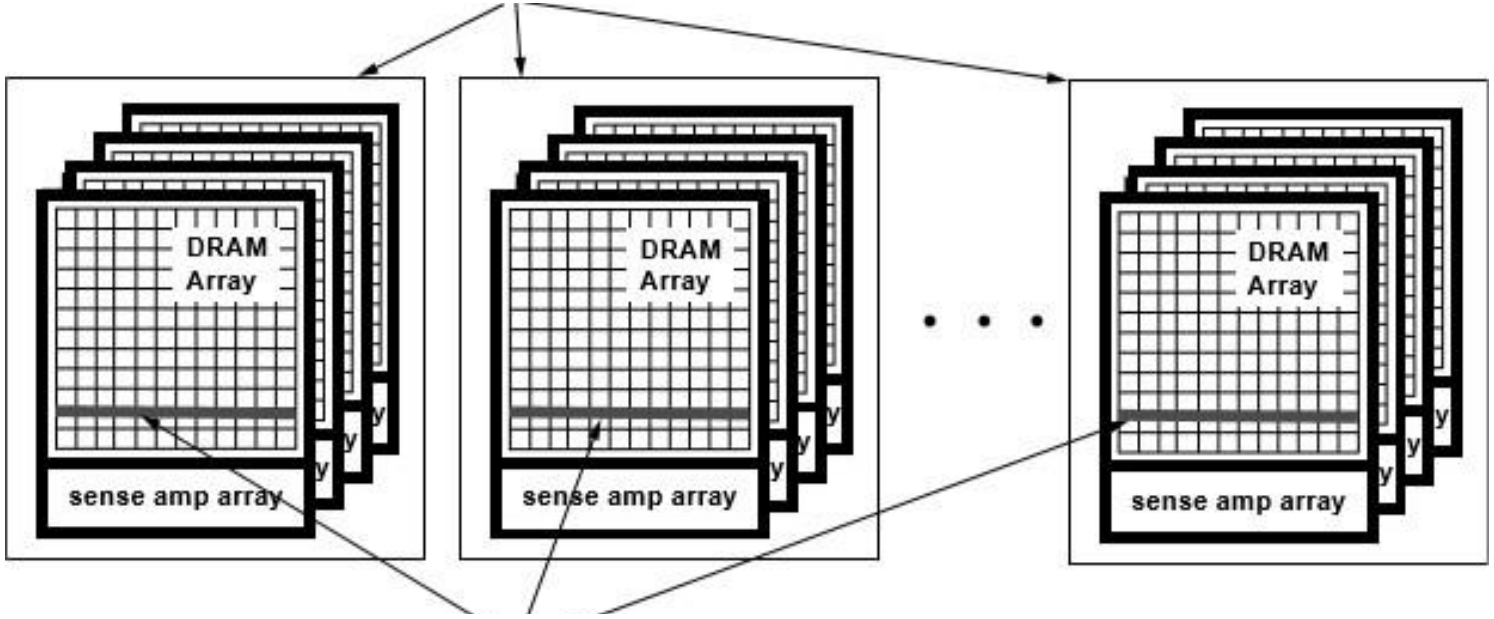


x2 DRAM　　x4 DRAM　　x8 DRAM

■ 内存系统的定义

- DRAM行
  - DRAM中的行(row)也被称为一个DRAM页(page)
- 行激活指令
  - 可以激活指定rank中所有DRAM芯片里行地址相同的阵列，以便高效地进行内存访问操作

在一个rank中，多个DRAM芯片是可以并行访问的



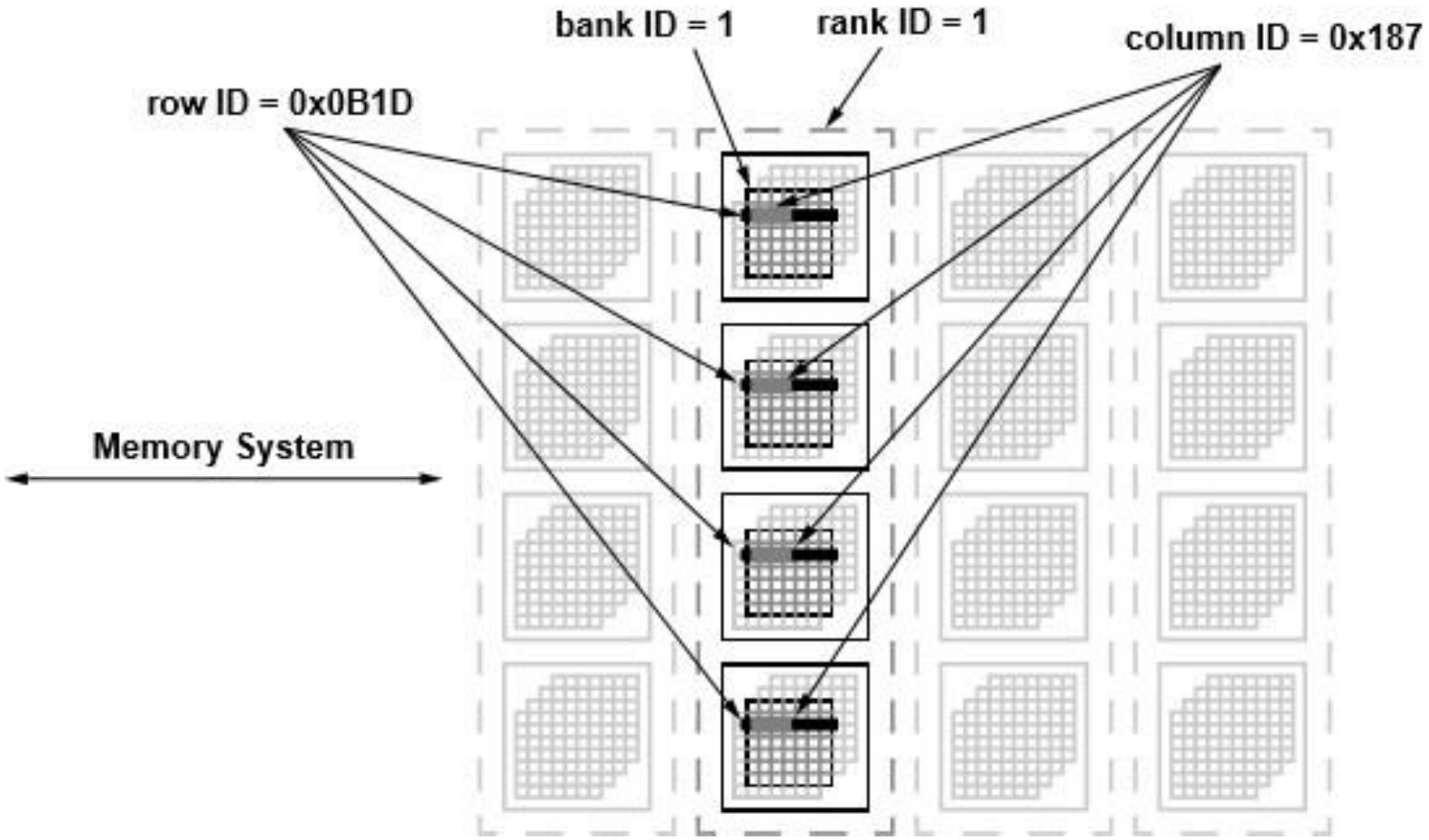
一个DRAM行可以横跨多个DRAM芯片

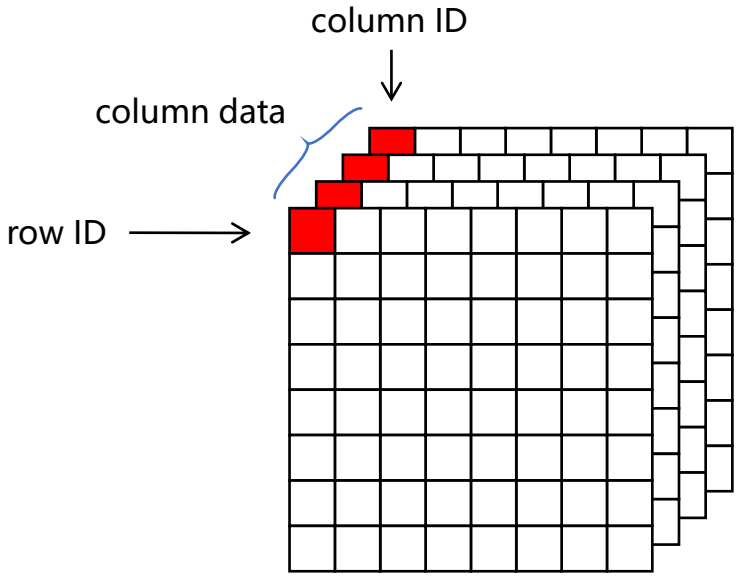
■ DRAM中的并行

- Rank
  - 是一组可独立寻址的DRAM芯片集合，可支持DIMM级别的并行操作
- Bank
  - DRAM芯片内部，每一组可独立运行的内存阵列被称为Bank
  - DRAM中的多个Bank可以同时处理不同的请求(交错执行)
- 在Rank和Bank层面实现并发，可以对不同的请求进行流水线处理

■ DRAM中的数据定位

- 一列数据(column data)是DRAM中可访问的最小地址单位

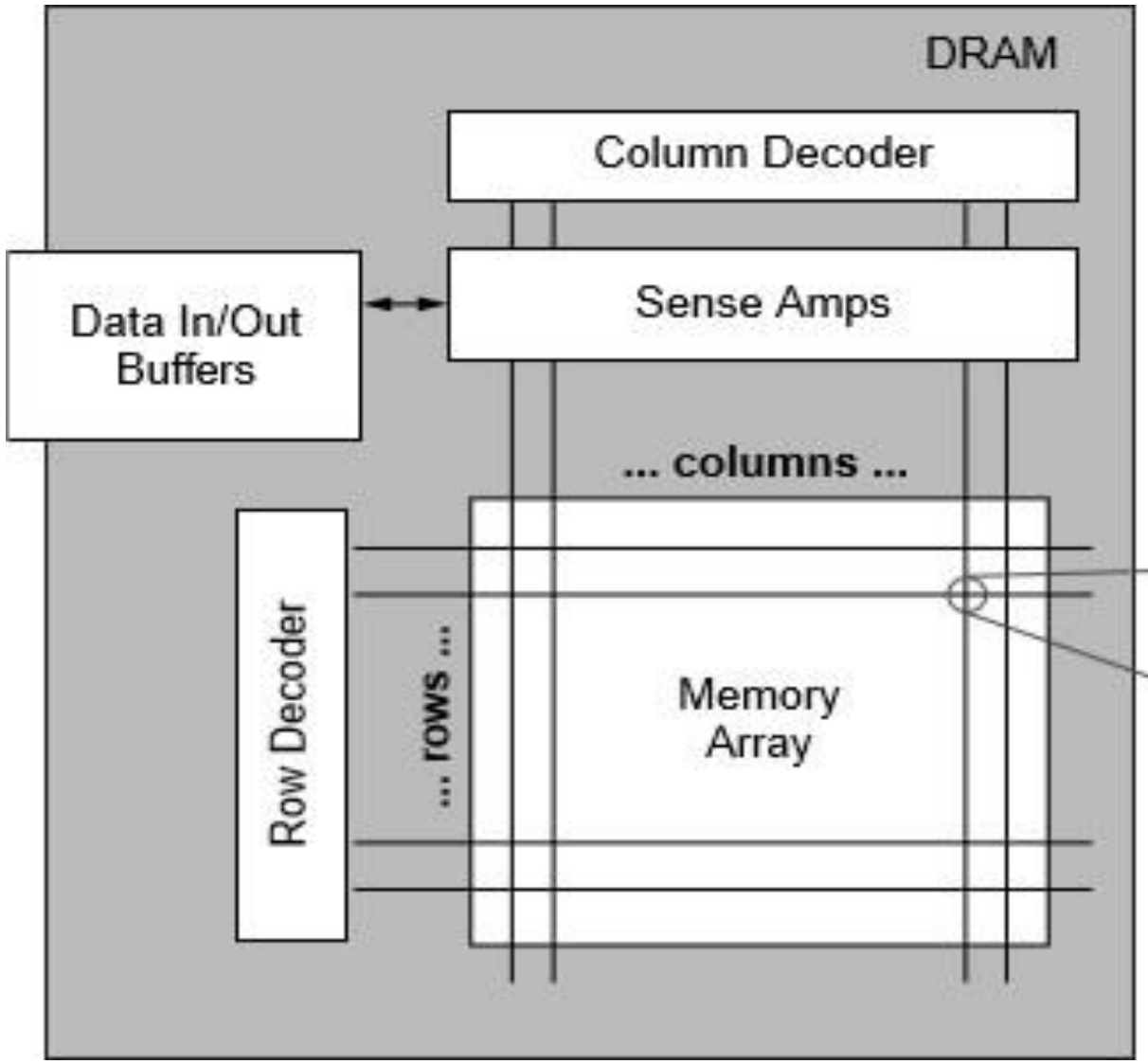




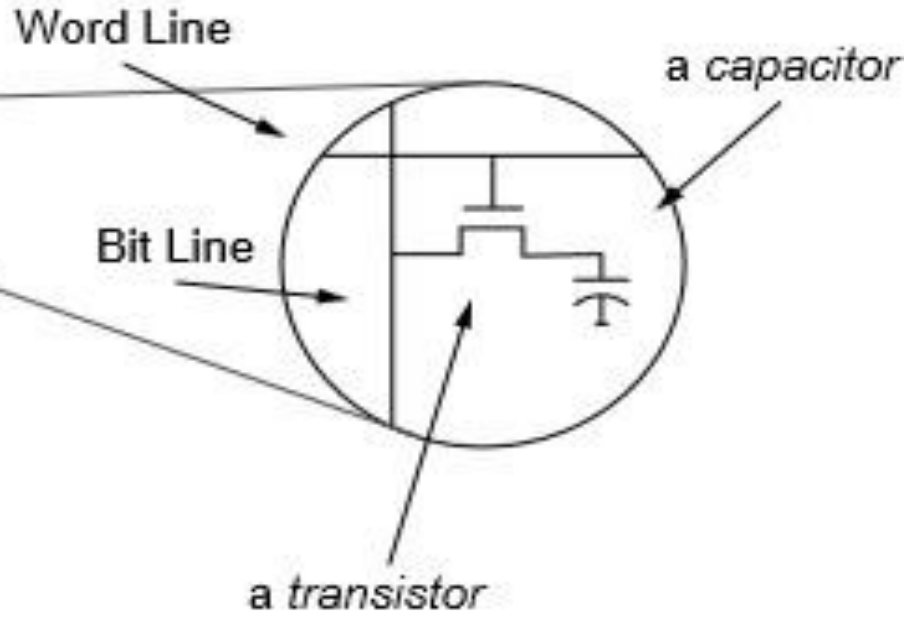
■ 地址映射

- 物理地址解析
  - 物理地址会被解析为多个索引
  - Channel ID，rank ID，bank ID，row ID，column ID
- 常见地址映射策略
  - row:rank:bank:channel:column:blkoffset
  - row:column:rank:bank:channel:blkoffset
- 性能优化
  - 连续的数据可以放置在连续的DRAM行，提高行缓冲区命中率
  - 连续的数据也可以被放置在不同的rank中，提高并行性

■ 1T1C DRAM单元

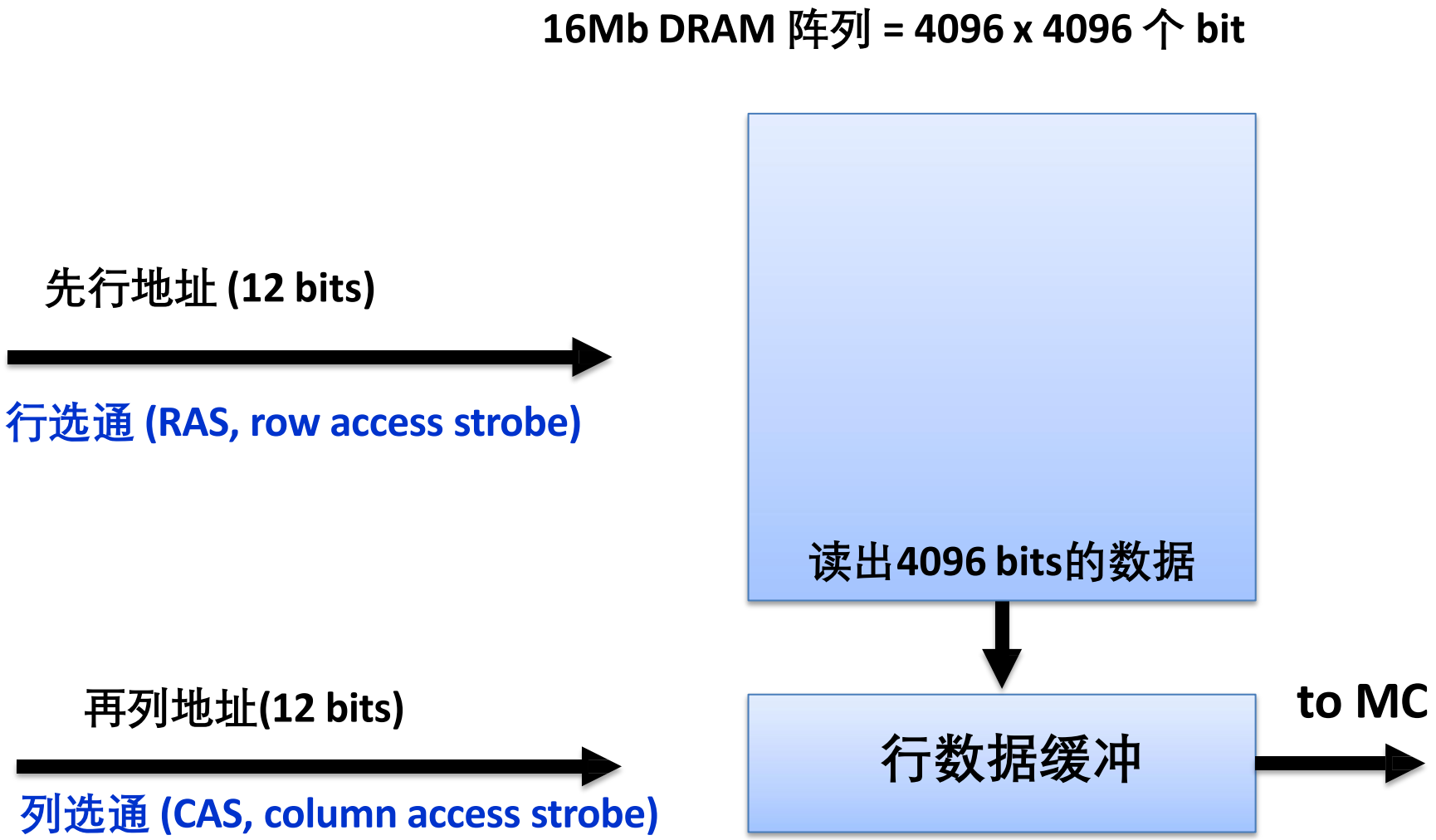


- 1个晶体管-电容器(1T1C)的组合可以存储一个bit
- 电容器需要定期刷新
- 控制总线包括行选通、列选通、时钟以及其他控制信号



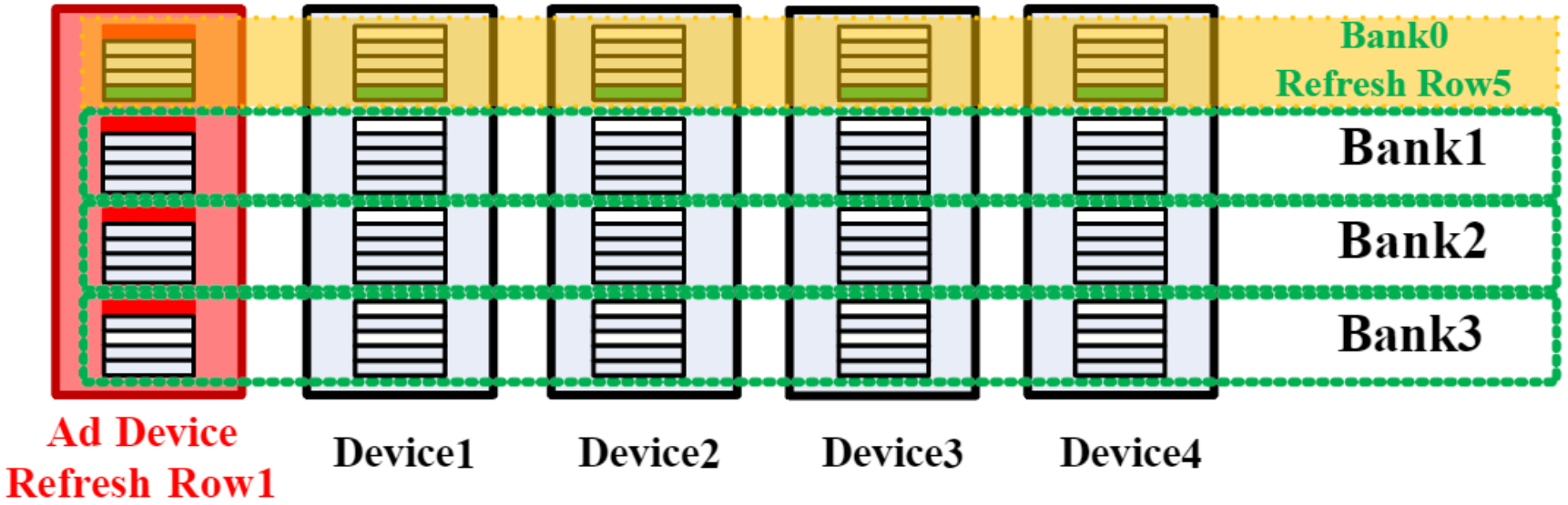
■ DRAM阵列访问

□ 经典DRAM系统：定位数据需要传递两次地址



■ 刷新机制

- 行是bank中最小的刷新单位
- 通常在温度低于85℃时，数据能稳定保留64ms
- 刷新操作可以选择bank级别(per-bank refresh)以及rank(per-rank refresh)级别
  - per-rank refresh时，刷新期间指定rank中的所有bank都会被锁定，暂时不能读写

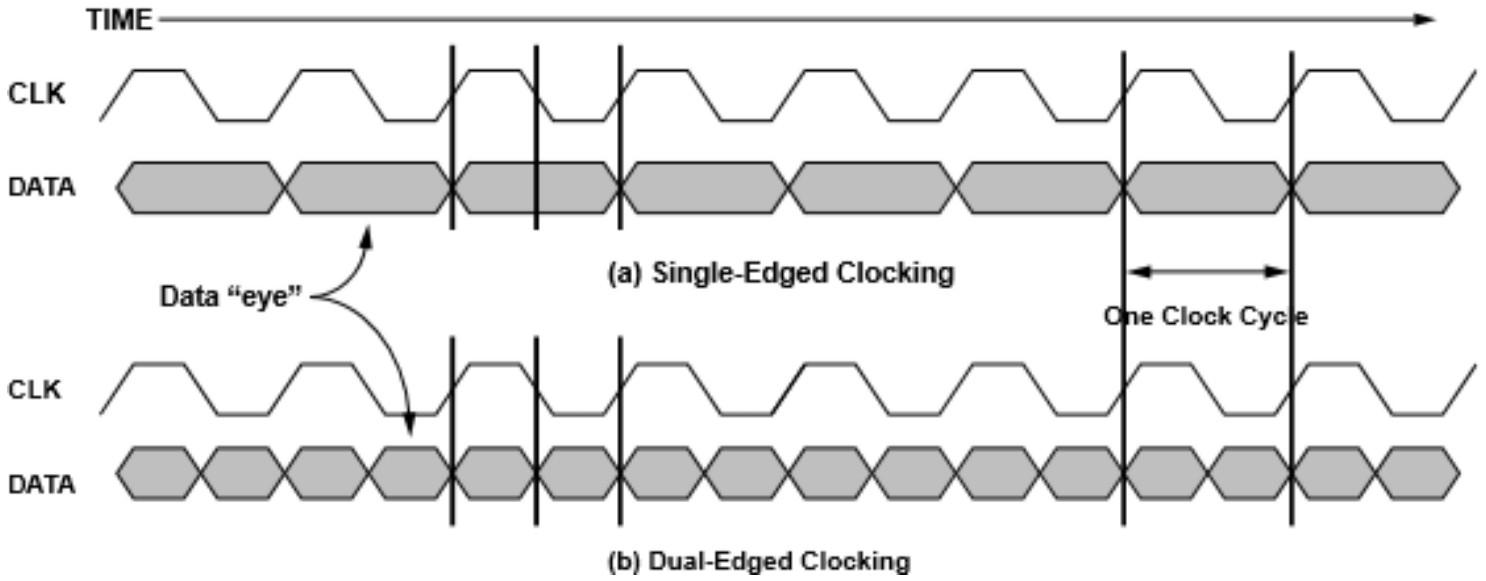


■ 访问DRAM的开销

- 行缓冲区(Row Buffer)是DRAM内部的Cache
  - Row Buffer命中~20ns
    - 缓冲区中的数据直接读取
  - Row Buffer未访问~40ns
    - 数据先读入缓冲区，再读取到外部
  - Row Buffer有冲突~60ns
    - 先把旧数据写回，再读入新目标行数据，最后读取数据到外部

■ 同步和异步DRAM

- 同步DRAM(SDRAM)
  - 用统一的时钟取代传统DRAM的RAS和CAS
  - 直接由系统时钟信号驱动工作
- double data rate(DDR) SDRAM
  - 数据在时钟的上升沿和下降沿各传输一次

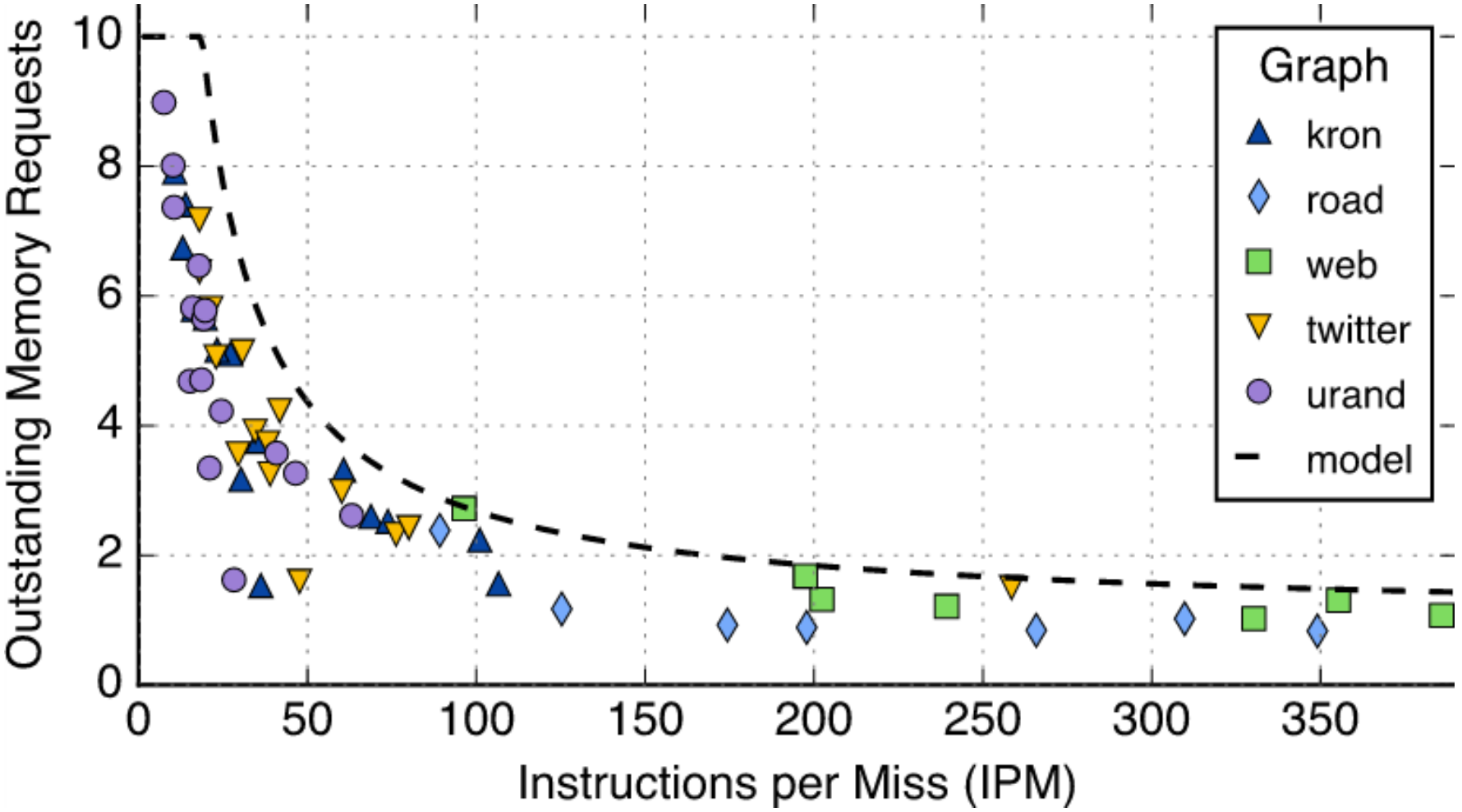


■ 内存墙和内存带宽墙

- 处理器引脚的数量很难大幅增加，内存带宽提升遇到瓶颈
- DRAM容量和访问时延发展不平衡，DRAM的容量每两年翻倍，但访问延迟几乎没有改善
- 功耗墙：DRAM系统的功耗占数据中心总功耗的25%-40%
- 优化方法：
  - 提高Row buffer命中率可以同时降低DRAM的延迟以及功耗
  - 需要在智能数据调度、访存请求重排等技术上发力

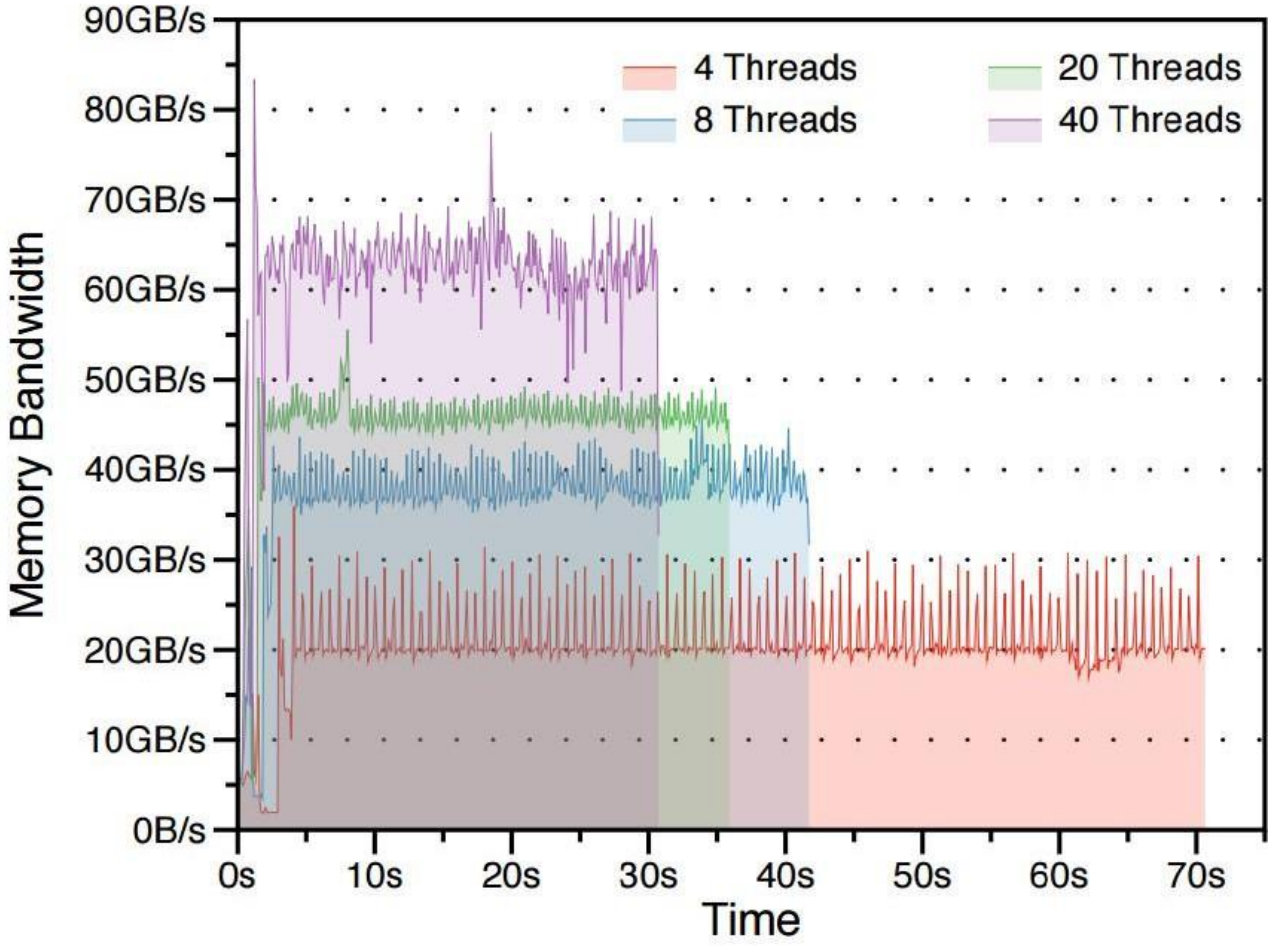
■ 图计算场景下的内存级并行

□ 内存带宽利用率受到高IPM的拖累



■ 图计算场景下的内存级并行

- 内存带宽的利用率随着线程数的上升而提升



感谢！