课程编号180086085404P2008H 2024-2025学年秋季学期

# 计算机体系结构

# 第7讲 虚拟内存、主存系统、缓存一致性

主讲教师： 刘珂

2024年10月22日

## Some Challenges in Virtual Memory

# Page Table Challenges

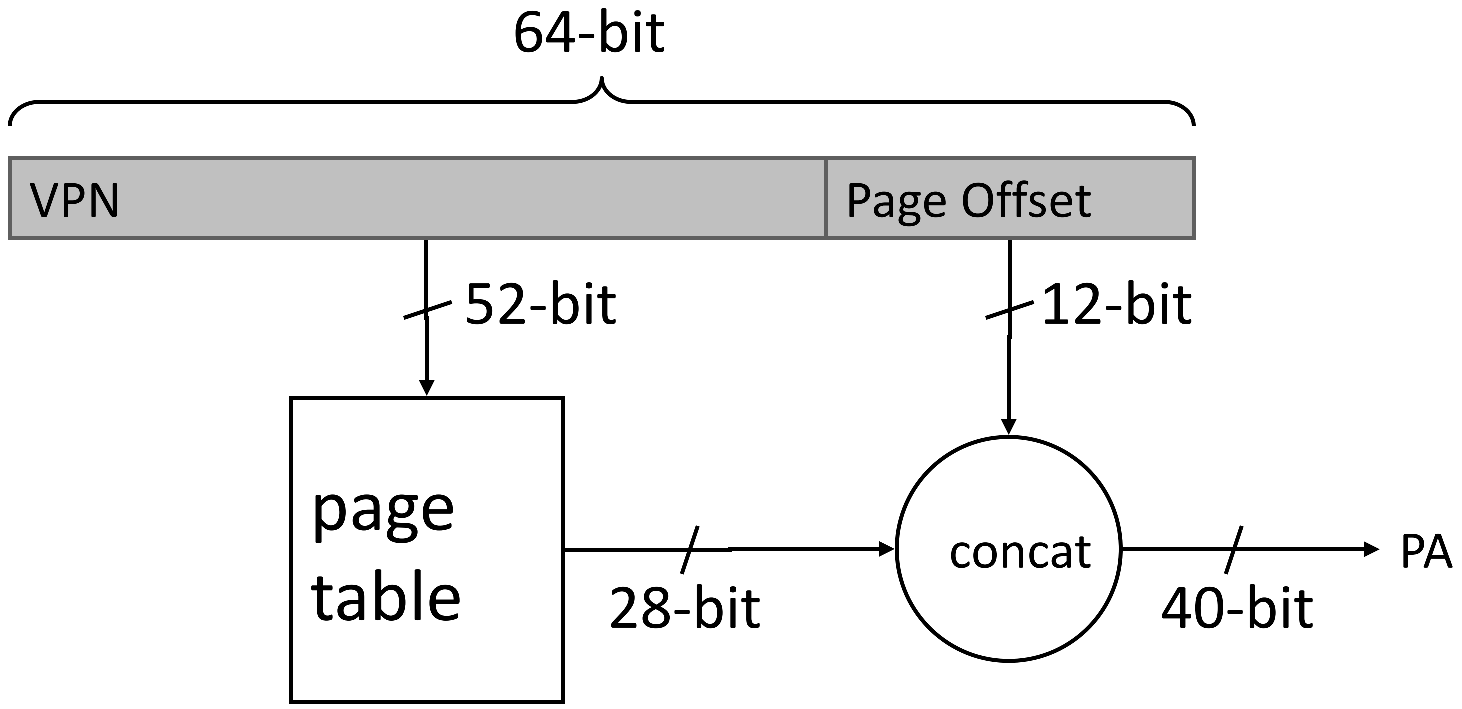
- Challenge 1: Page table is large 页表很大
  - at least part of it needs to be located in physical memory
  - solution: multi-level (hierarchical) page tables 解决方案：多级页表

- Challenge 2: Each instruction fetch or load/store requires at least two memory accesses 每个指令获取或数据访存至少两次内存访问：

  1. one for address translation (page table read)
  2. one to access data with the physical address (after translation)

  - Two memory accesses to service an instruction fetch or load/store greatly degrades execution time 两次内存访问指令获取或数据访存会大大降低执行时间
    - speed up the translation…
- Challenge 3: When do we do the translation in relation to cache access? 我们什么时候进行与缓存访问相关的翻译？

# Challenge I: Page Table Size



- Suppose 64-bit VA and 40-bit PA, how large is the page table?
  - **$2^{52}$ entries x ~4 bytes ≈ $2^{54}$ bytes**

    and that is for just one process!

    and the process may not be using the entire VM space!

# Virtual Memory Challenge II

- Idea: Use a hardware structure that caches PTEs → Translation lookaside buffer 使用缓存 PTE 的 TLB

- What should be done on a TLB miss? TLB 未命中该如何
  - What TLB entry to replace? 要替换哪个 TLB 条目
  - Who handles the TLB miss? HW vs. SW? 谁来处理 TLB 未命中

- What should be done on a page fault? 页面错误该如何
  - What virtual page to replace from physical memory?
  - Who handles the page fault? HW vs. SW?

## Translation Lookaside Buffer (TLB)

- Idea: Cache the page table entries (PTEs) in a hardware structure in the processor to speed up address translation 将页表条目（PTE）缓存在处理器的硬件结构中，以加快地址转换

- Translation lookaside buffer (TLB)
  - Small cache of most recently used translations (PTEs) 缓存最近使用的PTE
  - Reduces number of memory accesses required for *most* instruction fetches and loads/stores to only one 将大多数指令获取和数据访存所需的内存访问次数减少到只有一个

# Translation Lookaside Buffer (TLB)

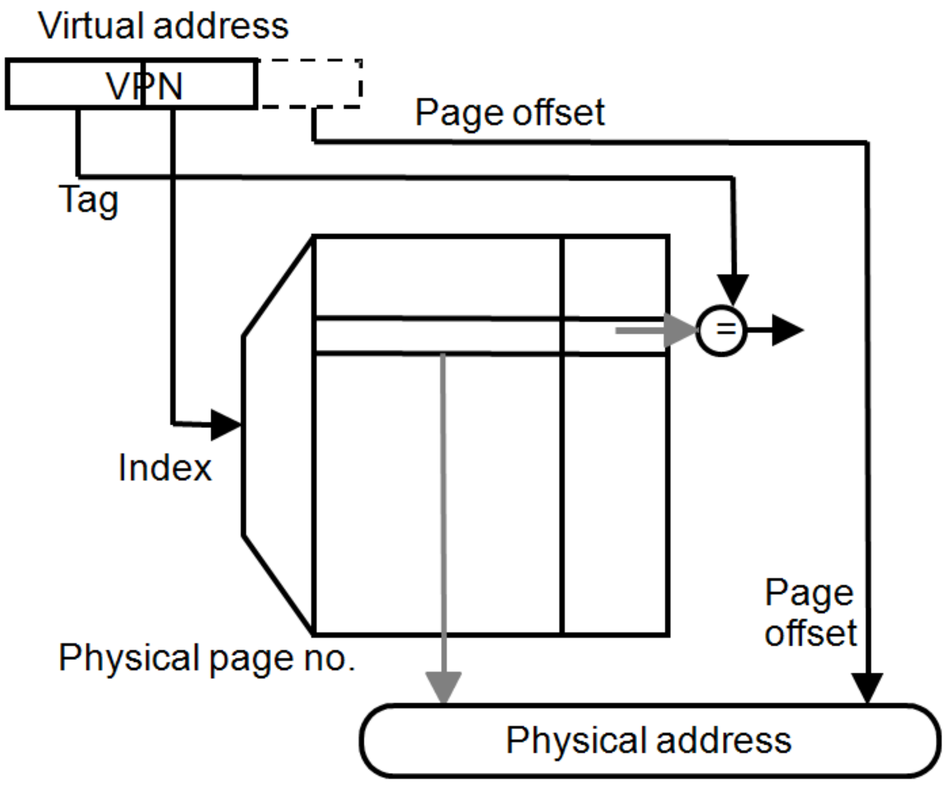
- Page table accesses have a lot of temporal locality 页表访问具有很大的时间局部性
  - Data accesses have temporal and spatial locality
    - 数据访问具有时间和空间局部性
  - Large page size (say 4KB, 8KB, or even 1-2GB)
  - Consecutive instructions and loads/stores are likely to access same page 连续的指令和访存大概率会访问同一页面
- TLB
  - Small: accessed in ~ 1 cycle
  - Typically 16 - 512 entries
  - High associativity
  - > 95-99 % hit rates typical (depends on workload)
  - Reduces number of memory accesses for most instruction fetches and loads/stores to only one

## Speeding up Translation with a TLB

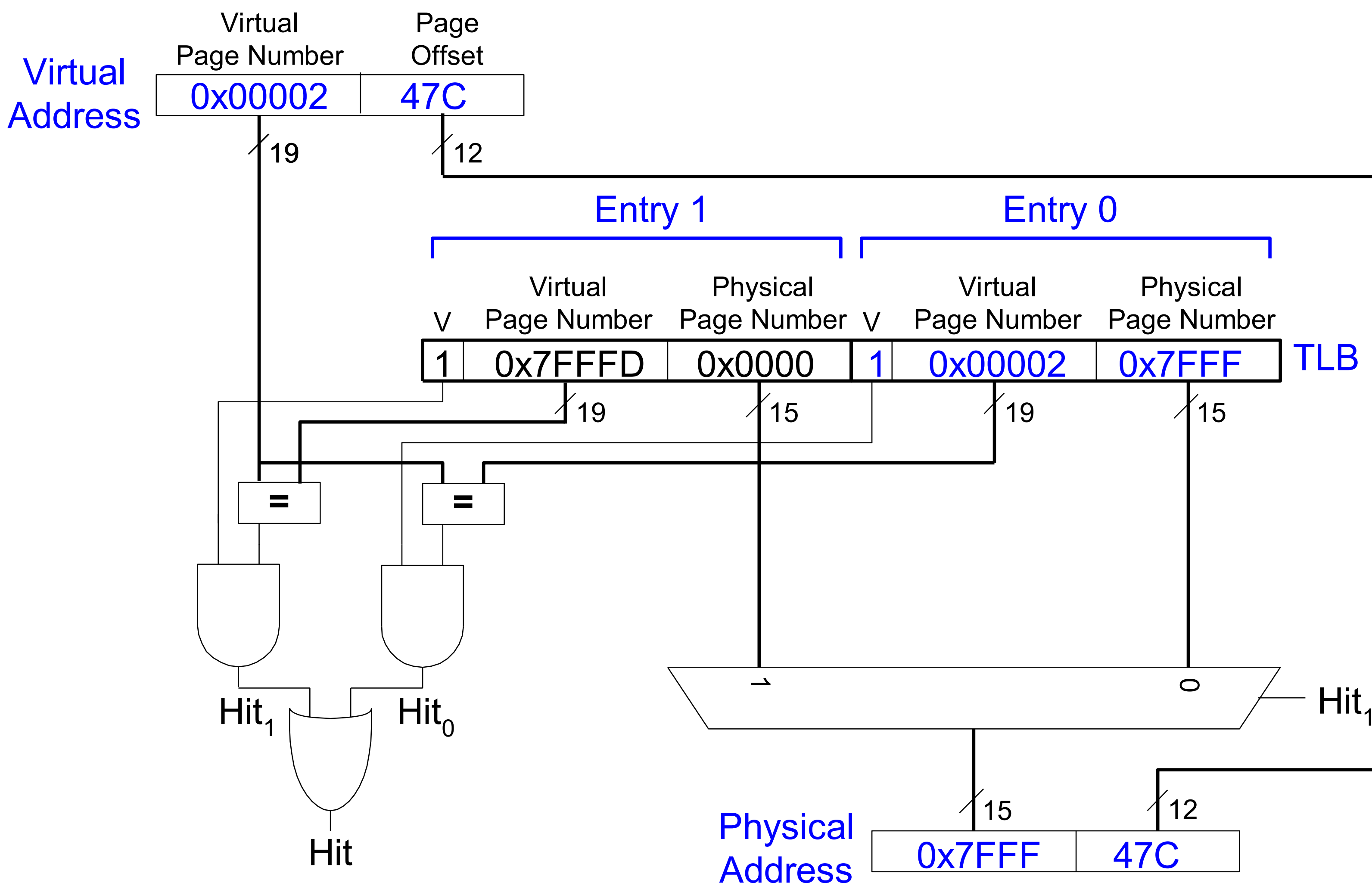
- Essentially a cache of recent address translations
  - Avoids going to the page table on every reference

- **Index** = lower bits of VPN (virtual page #)
- **Tag** = unused bits of VPN + process ID
- **Data** = a page-table entry
- **Status** = valid, dirty

The usual cache design choices (placement, replacement policy, multi-level, etc.) apply here too.



# Example Two-Entry TLB



# Handling TLB Misses

- The TLB is small; it cannot hold <u>all</u> PTEs
  - Some translations will inevitably miss in the TLB
  - Must access memory to find the appropriate PTE
    - Called **walking** the page directory/table
    - Large performance penalty

- Who handles TLB misses? Hardware or software?

# Handling TLB Misses

- Approach #1. **Hardware-Managed** (e.g., x86)
  - The hardware does the **page walk**
  - The hardware fetches the PTE and inserts it into the TLB
    - If the TLB is full, the entry **replaces** another entry
  - Done transparently to system software

- Approach #2. **Software-Managed** (e.g., MIPS)
  - The hardware raises an exception
  - The operating system does the **page walk**
  - The operating system fetches the PTE
  - The operating system inserts/evicts entries in the TLB

# Virtual Memory Challenge III

- The relation between TLB and L1 cache
  - TLB 和 L1 缓存之间的关系
  - Address Translation and Caching

- When do we do the address translation?
  - Before or after accessing the L1 cache?

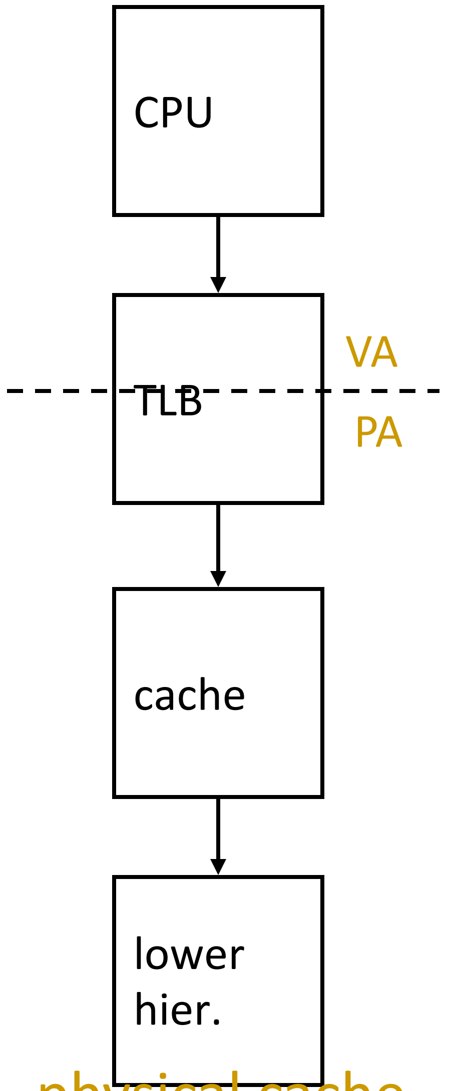
## Address Translation and Caching

- When do we do the address translation? 什么时候进行地址翻译？
  - Before or after accessing the L1 cache?

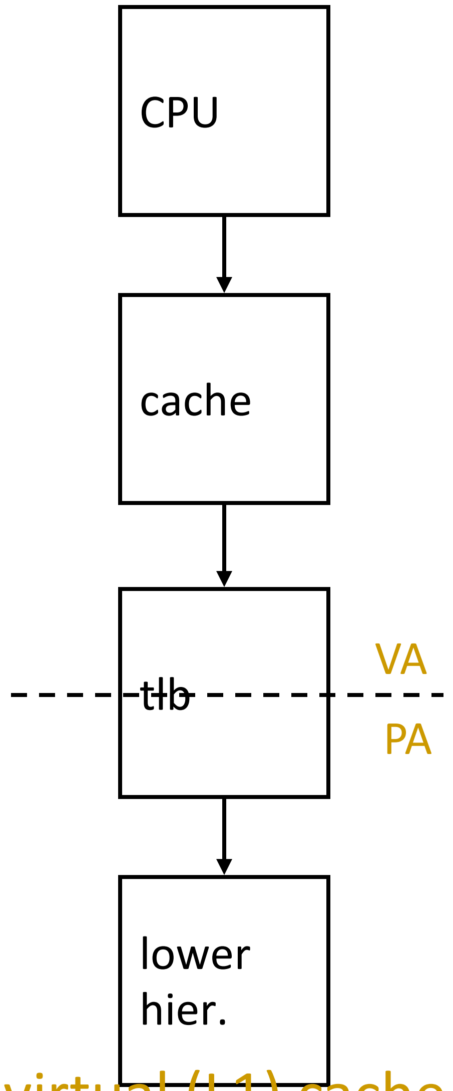
- In other words, is the cache virtually addressed or physically addressed? 缓存是虚拟寻址还是物理寻址？
  - Virtual versus physical cache

- What are the issues with a virtually addressed cache?
- Aliasing problem:
  - Two different virtual addresses can map to the same physical address → same physical address can be present in multiple locations in the cache → can lead to inconsistency in data

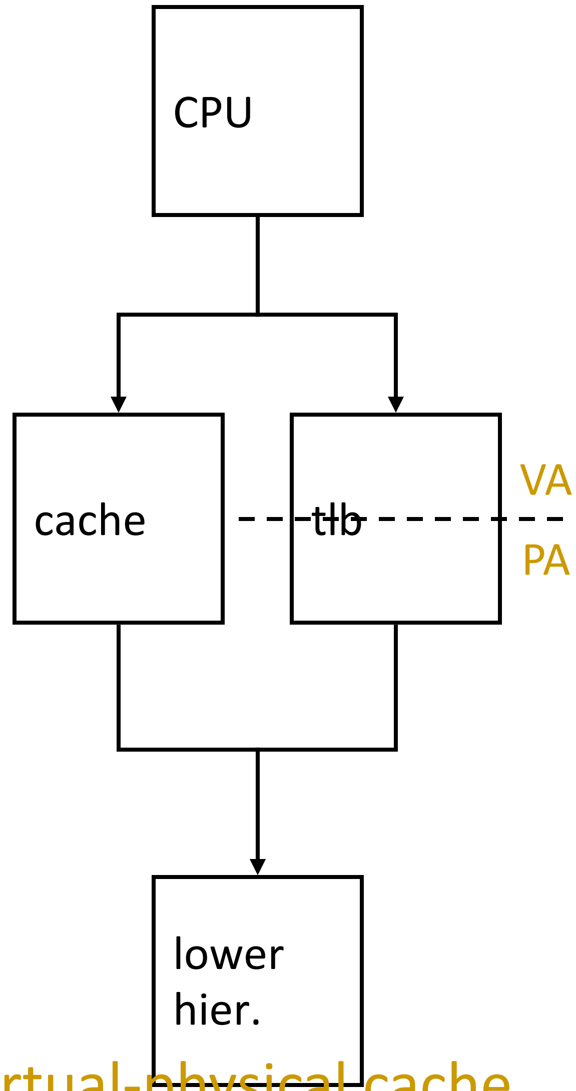
# Cache-VM Interaction



physical cache



virtual (L1) cache

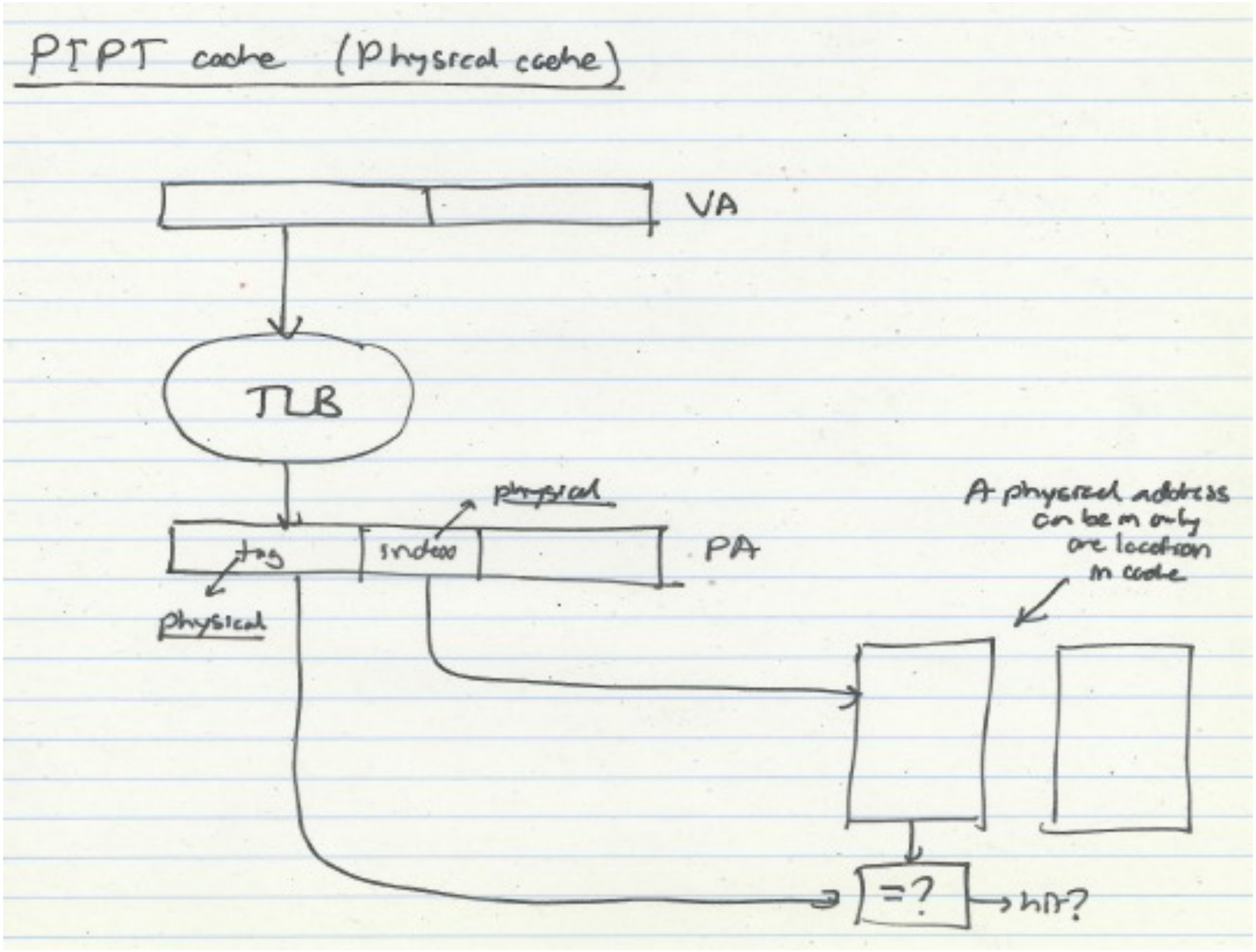


virtual-physical cache

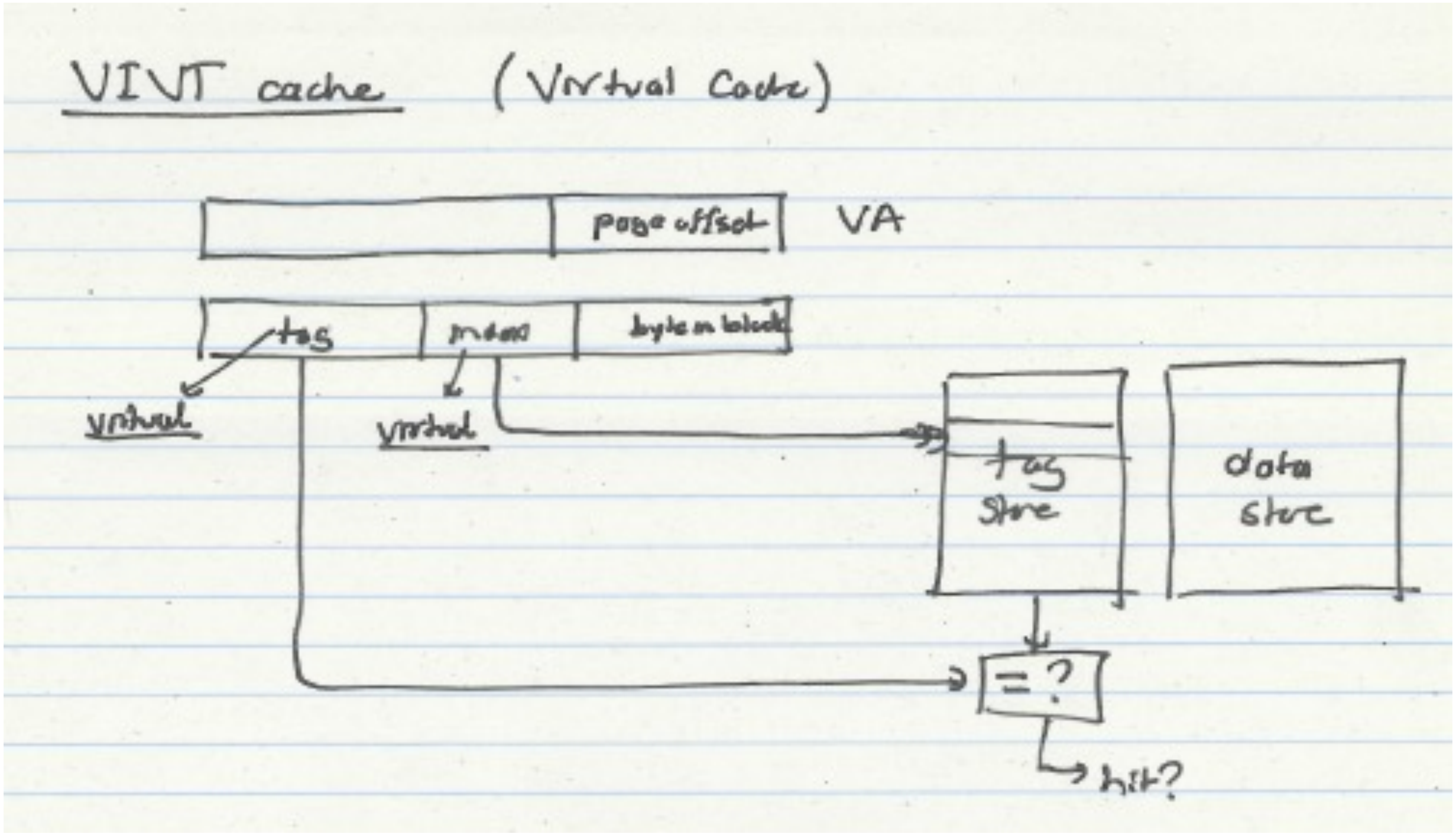
# Memory Aliasing 别名

- Same VA can map to two different PA
  - 同一个 VA 可以映射到两个不同的 PA
  - Why?
    - VA is in different processes
- Aliasing: Different VAs can map to the same PA
  - 不同的 VA 可以映射到同一个 PA
  - Why?
    - Different pages can share the same physical frame within or across processes
    - Reasons: shared libraries, shared data, copy-on-write pages within the same process, …
- Do aliasing create problems when we have a cache? E.g., L1 Cache
  - Is the cache virtually or physically addressed?

# Physical Cache


PIPT cache (Physical cache)
VA
TLB
physical
tag
index
PA
physical
A physical address can be in only one location in cache
=?
hit?

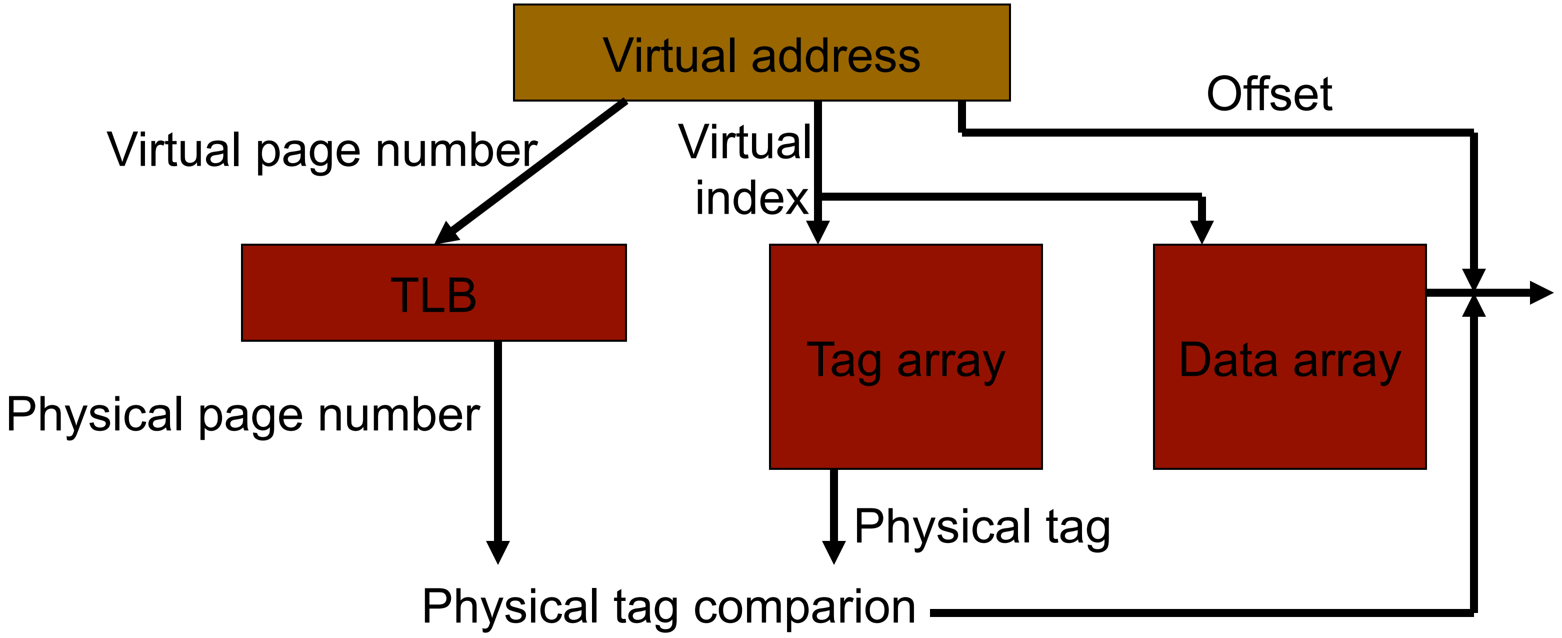
# Virtual Cache



## TLB and Cache

- Is the cache indexed with virtual or physical address?
  - To index with a physical address, we will have to first look up the TLB, then the cache -> longer access time
  - Multiple virtual addresses can map to the same physical address – can we ensure that these different virtual addresses will map to the same location in cache? Else, there will be two different copies of the same physical memory word

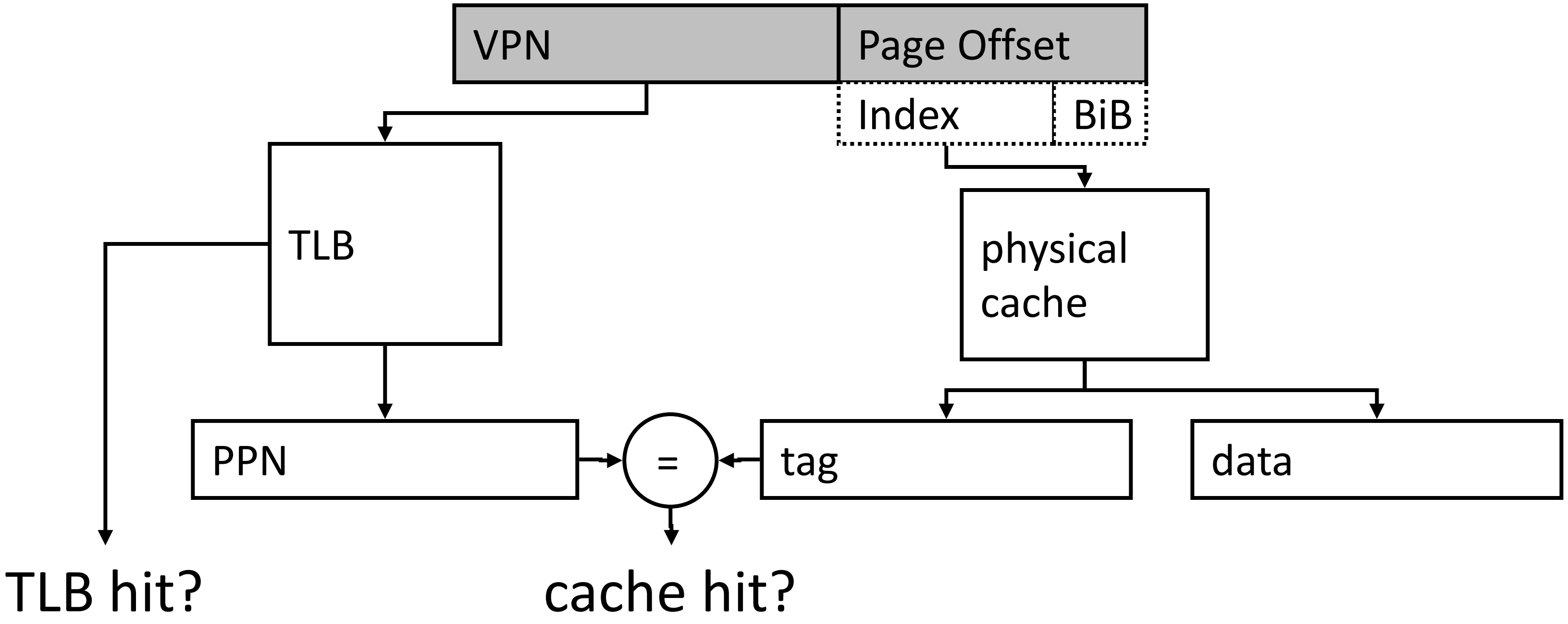
# Virtual-Physical Cache



Virtually Indexed; Physically Tagged Cache

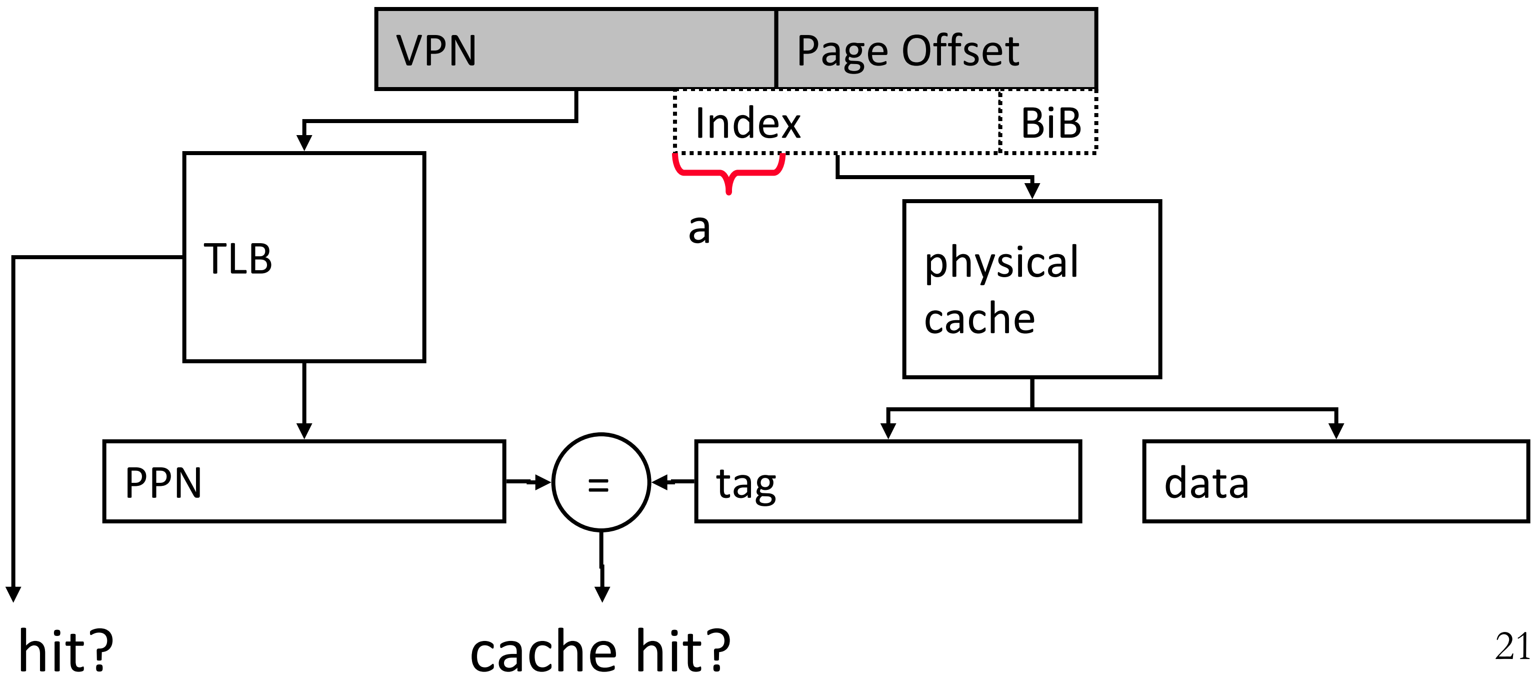
# Virtually-Indexed Physically-Tagged

- If C≤(page_size × associativity), the cache index bits come only from page offset (same in VA and PA)
- If both cache and TLB are on chip
  - index both arrays concurrently using VA bits
  - check cache tag (physical) against TLB output at the end



# Virtually-Indexed Physically-Tagged

- If C>(page_size × associativity), the cache index bits include VPN ⇒
  - Aliasing can cause problems
  - The same physical address can exist in two locations
- Solutions?



# Some Solutions to the Aliasing Problem

- Limit cache size to (page size times associativity) 将缓存大小限制为(页面大小*关联度)
  - get index from page offset 从页面偏移量获取索引
- On a write to a block, search all possible indices that can contain the same physical block, and update/invalidate 在写入块时，搜索可以包含相同物理块的所有可能索引，并使更新/失效
  - Used in Alpha 21264, MIPS R10K
- Restrict page placement in OS
  - make sure index(VA) = index(PA)
  - Called page coloring
  - Used in many SPARC processors

## Virtual Memory Support and Examples

## Supporting Virtual Memory

- Virtual memory requires both HW+SW support
  - Page Table is in memory
  - Can be cached in special hardware structures called Translation Lookaside Buffers (TLBs) 可以缓存在TLB – 一种特殊硬件结构
- The hardware component is called the MMU (memory management unit 内存管理单元)
  - Includes Page Table Base Register(s), TLBs, page walkers
- It is the job of the software to leverage the MMU to
  - Populate page tables, decide what to replace in physical memory 填充页表，在物理内存中决定替换页
  - Change the Page Table Register on context switch (to use the running thread's page table) 在进程切换时更改页表注册（以使用正在运行的线程的页表）
  - Handle page faults and ensure correct mapping

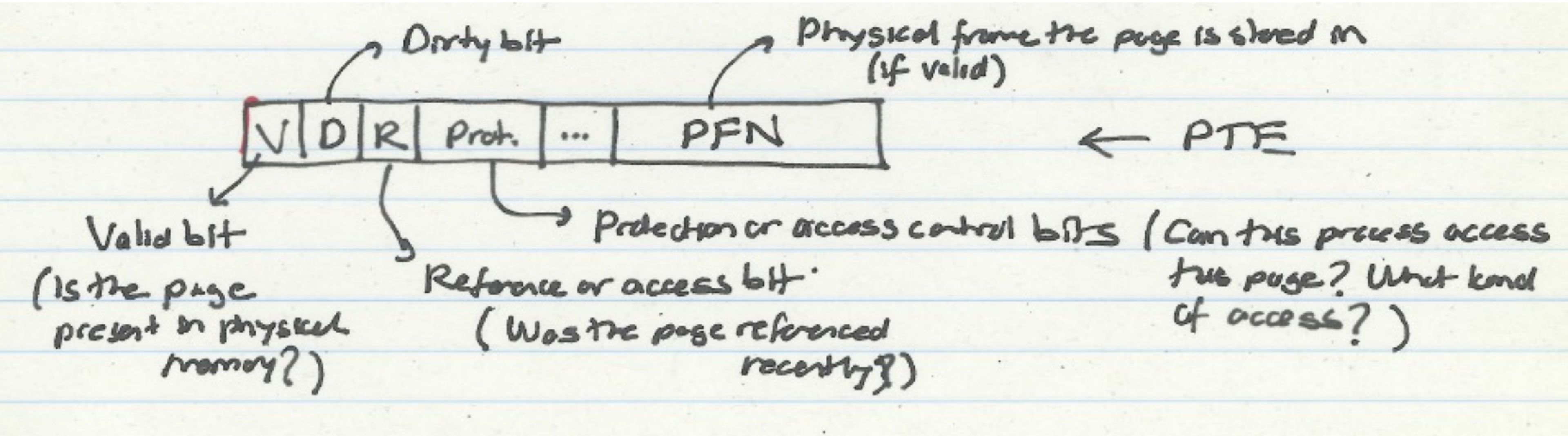
## Address Translation

- Page size specified by the ISA
  - Today: 4KB, 8KB, 2GB, … (small and large pages mixed together)
  - Trade-offs? (remember cache block)

- A Page Table Entry (PTE) for each virtual page
  - What is in a PTE?

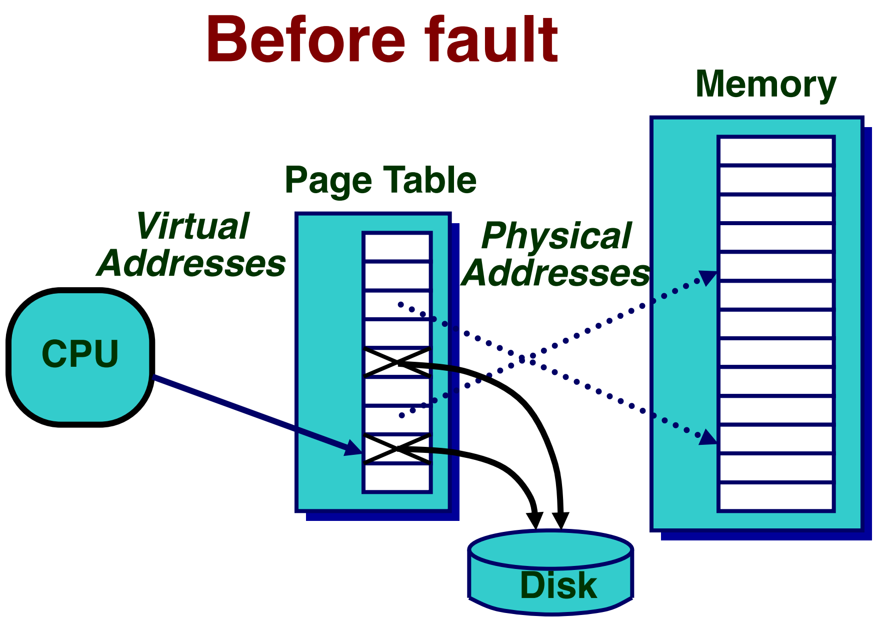
# What Is in a Page Table Entry (PTE)?

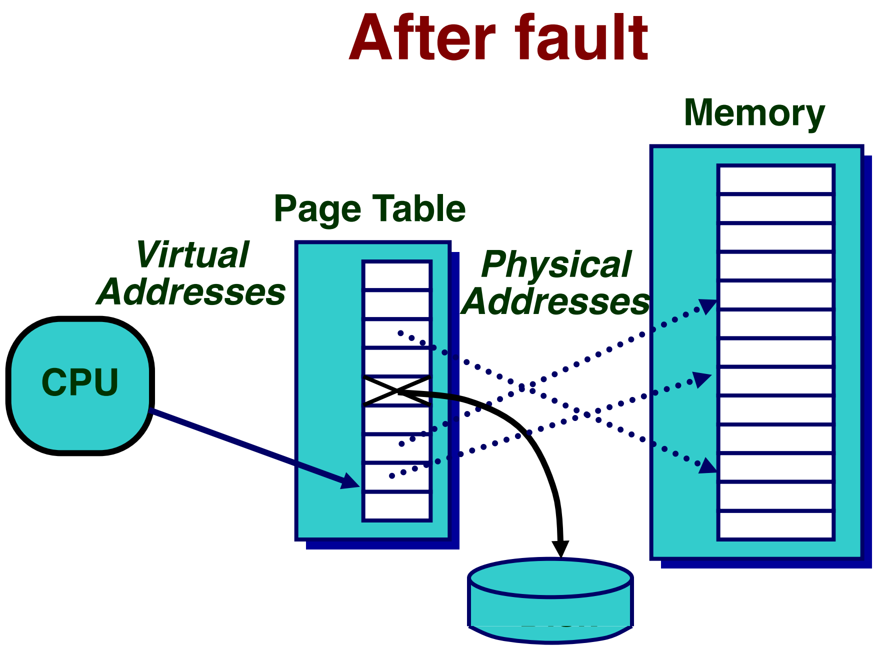
- Page table is the "tag store" for the physical memory data store
  - A mapping table between virtual memory and physical memory
- PTE is the "tag store entry" for a virtual page in memory
  - Need a valid bit → to indicate validity/presence in physical memory
  - Need tag bits (PFN) → to support translation
  - Need bits to support replacement
  - Need a dirty bit to support "write back caching"
  - Need protection bits to enable access control and protection


Dirty bit
Physical frame the page is stored in (if valid)
V
D
R
Prot.
...
PFN
← PTE
Valid bit (Is the page present in physical memory?)
Reference or access bit (Was the page referenced recently?)
Protection or access control bits (Can this process access this page? What kind of access?)

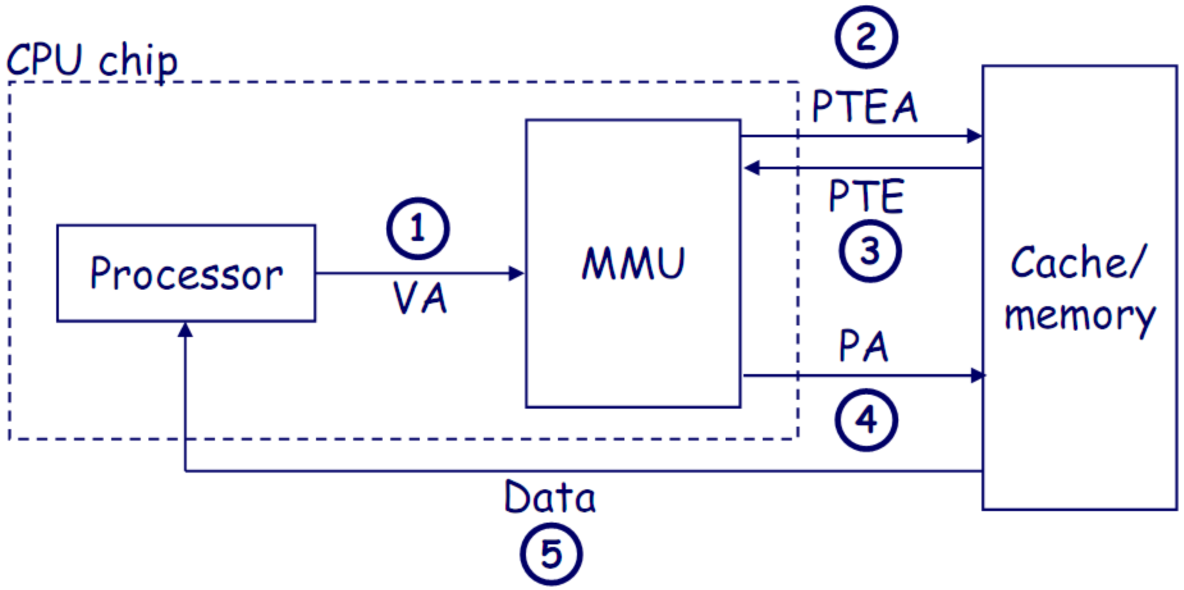
# Page Fault ("A Miss in Physical Memory")

- If a page is not in physical memory but disk
  - Page table entry indicates virtual page not in memory
  - Access to such a page triggers a page fault exception
  - OS trap handler invoked to move data from disk into memory
    - Other processes can continue executing
    - OS has full control over placement





# Address Translation: Page Hit



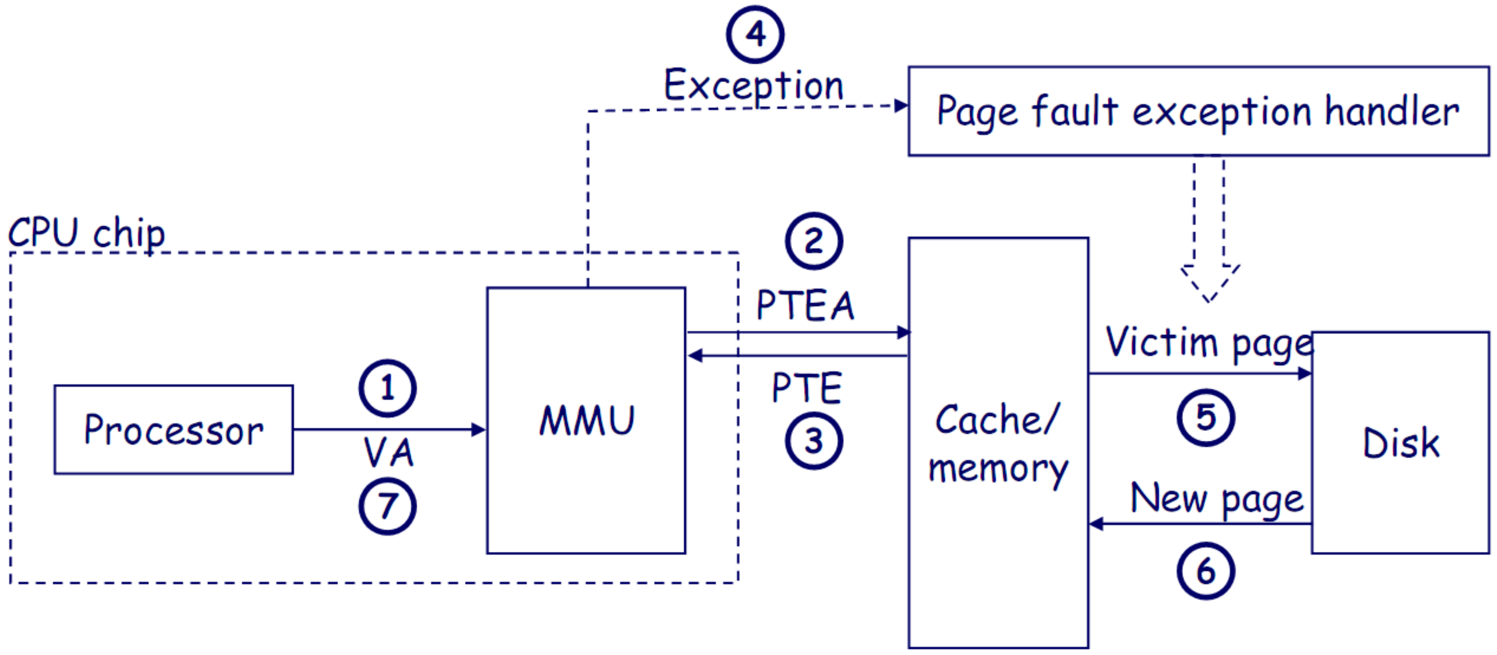
1) Processor sends virtual address to MMU

2-3) MMU fetches PTE from page table in memory

4) MMU sends physical address to L1 cache

5) L1 cache sends data word to processor

# Address Translation: Page Fault



1) Processor sends virtual address to MMU

2-3) MMU fetches PTE from page table in memory

4) Valid bit is zero, so MMU triggers page fault exception

5) Handler identifies victim, and if dirty pages it out to disk

6) Handler pages in new page and updates PTE in memory

7) Handler returns to original process, restarting faulting instruction.

## Cache versus Page Replacement

- Physical memory (DRAM) is a cache for disk
  - Usually managed by system software via the virtual memory subsystem

- Page replacement is similar to cache replacement
- Page table is the "tag store" for physical memory data store

- What is the difference?
  - Required speed of access to cache vs. physical memory
  - Number of blocks in a cache vs. physical memory
  - "Tolerable" amount of time to find a replacement candidate (disk versus memory access latency)
  - Role of hardware versus software

## Page Replacement Algorithms

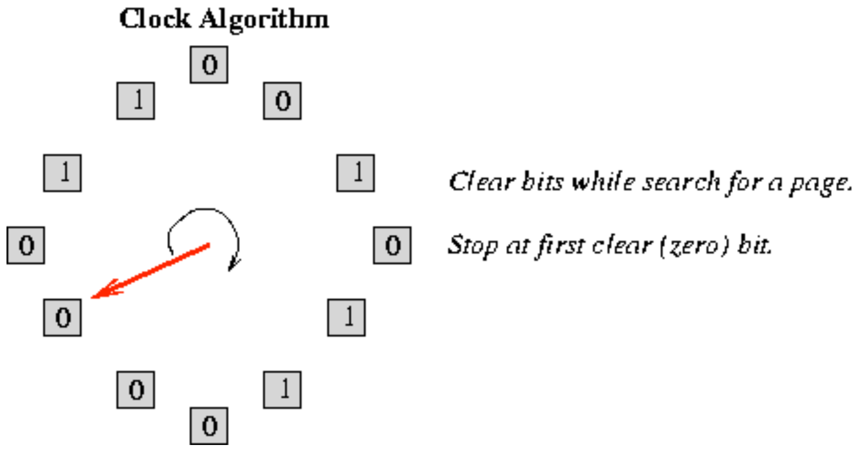
- If physical memory is full (i.e., list of free physical pages is empty), which physical frame to replace on a page fault? 如果物理内存已满(即可用物理页列表为空)，则在错页时要替换哪个物理页？

- Is True LRU feasible?
  - 4GB memory, 4KB pages, how many possibilities of ordering?
  - 1,000,000!

- Modern systems use approximations of LRU
  - E.g., the CLOCK algorithm
- And, more sophisticated algorithms to take into account "frequency" of use
  - E.g., the ARC algorithm
  - Megiddo and Modha, "ARC: A Self-Tuning, Low Overhead Replacement Cache," FAST 2003.

# CLOCK Page Replacement Algorithm

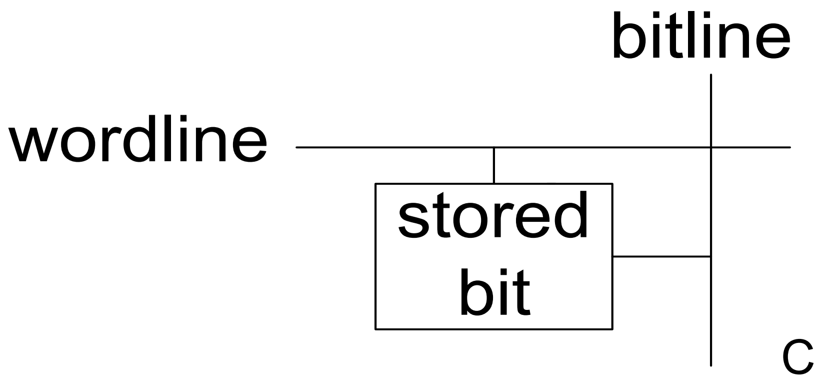
- Keep a circular list of physical frames in memory (OS does)
- Keep a pointer (hand) to the last-examined frame in the list
- When a page is accessed, set the R bit in the PTE
- When a frame needs to be replaced, replace the first frame that has the reference (R) bit not set, traversing the circular list starting from the pointer (hand) clockwise
  - During traversal, clear the R bits of examined frames
  - Set the hand pointer to the next frame in the list

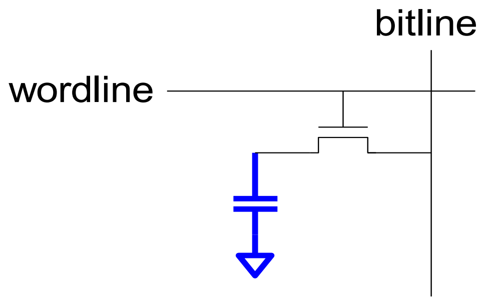

Clock Algorithm
0
1
0
1
1
0
0
0
1
0
1
0
Clear bits while search for a page.
Stop at first clear (zero) bit.

## Memory Architecture and Subsystems

# How is Access Controlled?

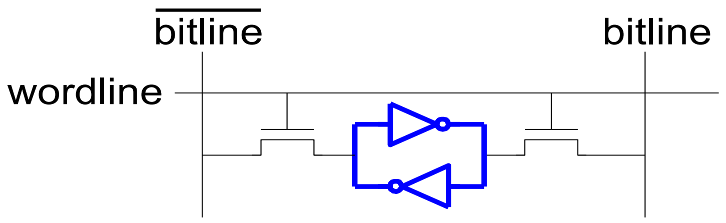
- Access transistors configured as switches connect the bit cell/storage (存储单元) to the bitline.
- Access controlled by the wordline





**DRAM**

Cross-coupled invertors
交叉耦合反向器



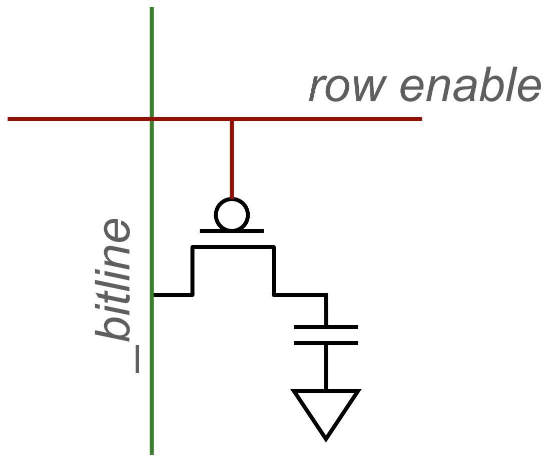
**SRAM**

Need different sensing circuitries

# Memory Technology: DRAM

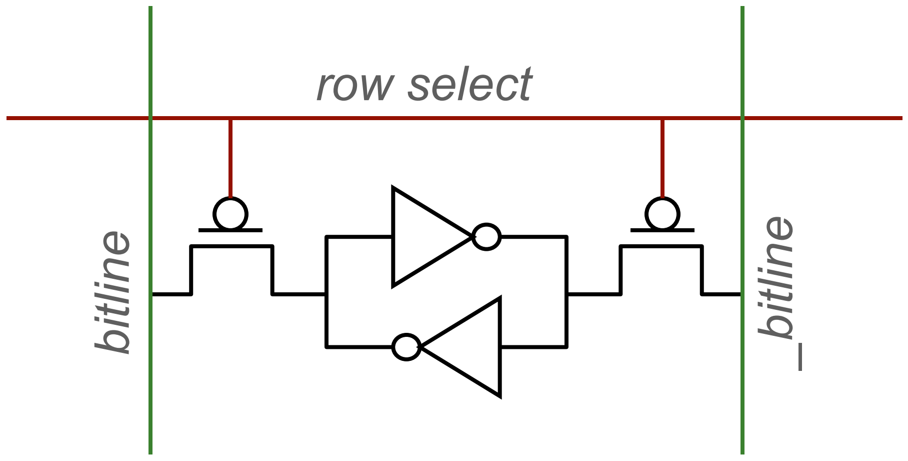
- Dynamic random access memory
- Capacitor charge state indicates stored value
  - Whether the capacitor is charged or discharged indicates storage of 1 or 0
  - 1 capacitor
  - 1 access transistor

- Capacitor leaks through the RC path
  - DRAM cell loses charge over time
  - DRAM cell needs to be refreshed

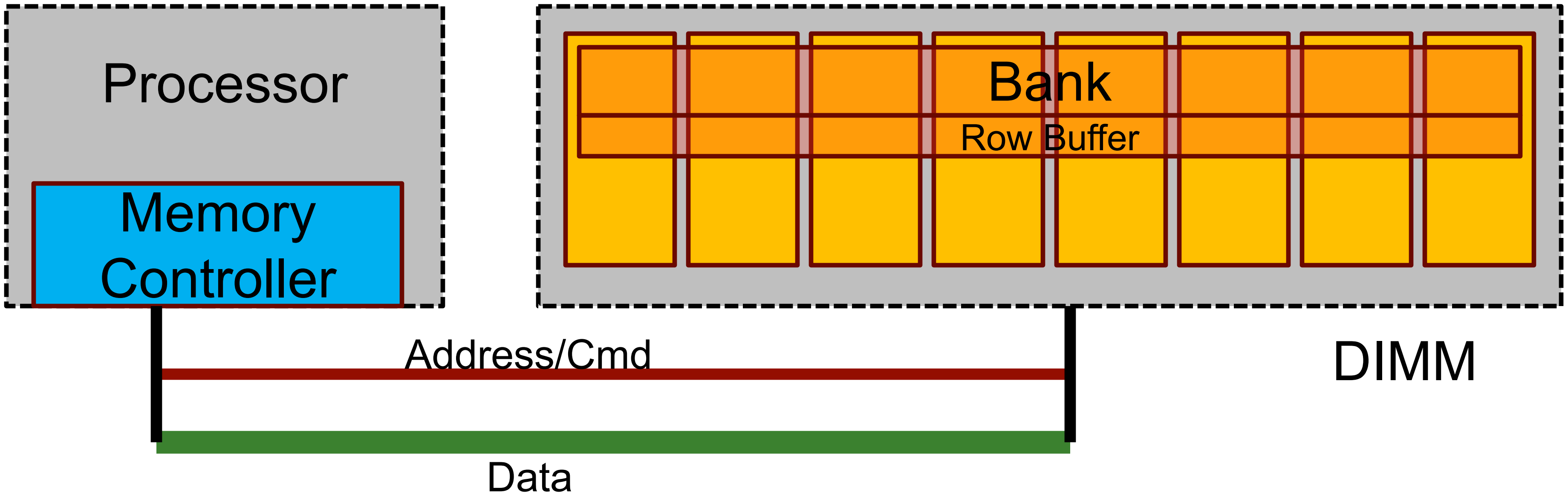


# Memory Technology: SRAM

- Static random access memory
- Two cross coupled inverters store a single bit
  - Feedback path enables the stored value to persist in the "cell"
  - 4 transistors for storage
  - 2 transistors for access

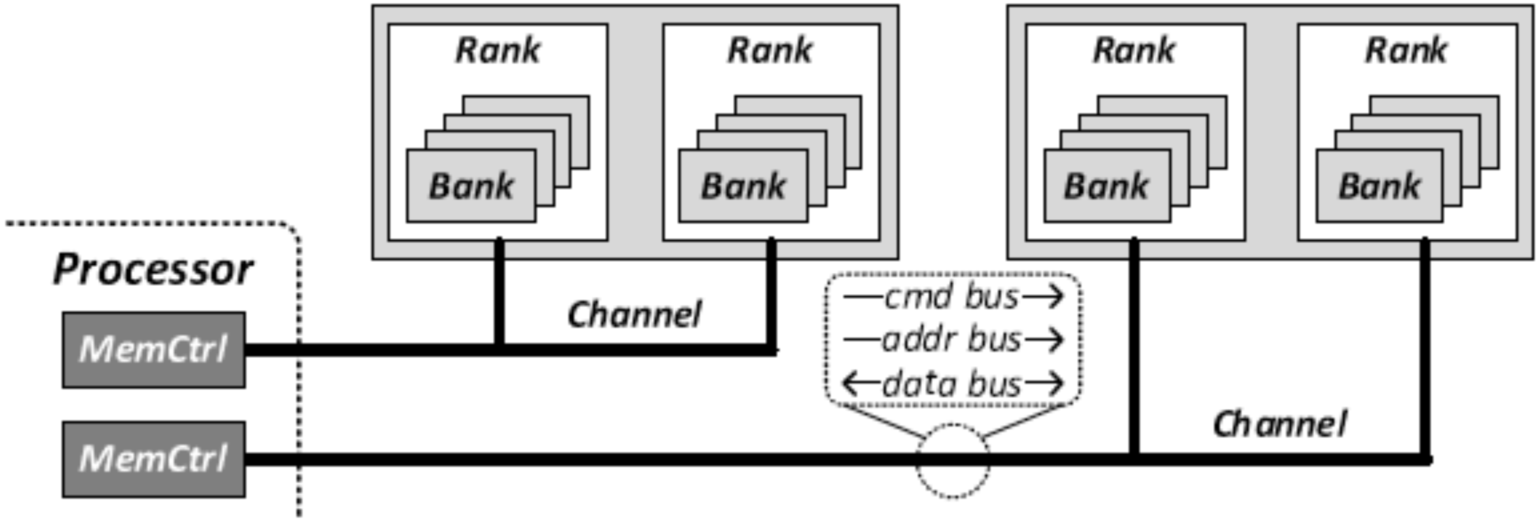


# Memory Architecture



- DIMM: a PCB with DRAM chips on the back and front
- Rank: a collection of DRAM chips that work together to respond to a request and keep the data bus full
- A 64-bit data bus will need 8 x8 DRAM chips or 4 x16 DRAM chips or..
- Bank: a subset of a rank that is busy during one request
- Row buffer: the last row (say, 8 KB) read from a bank, acts like a cache

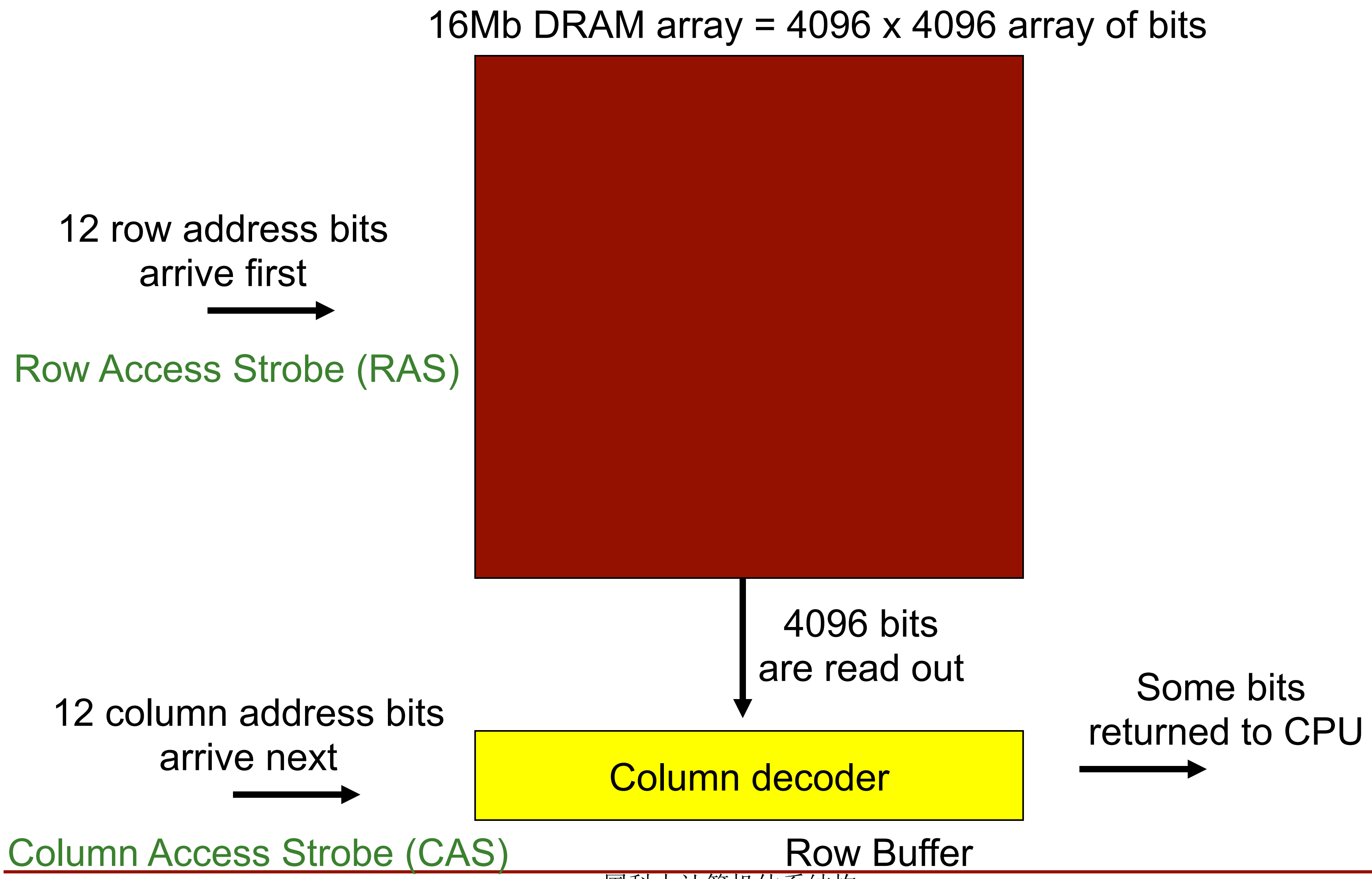
## Generalized Memory Structure



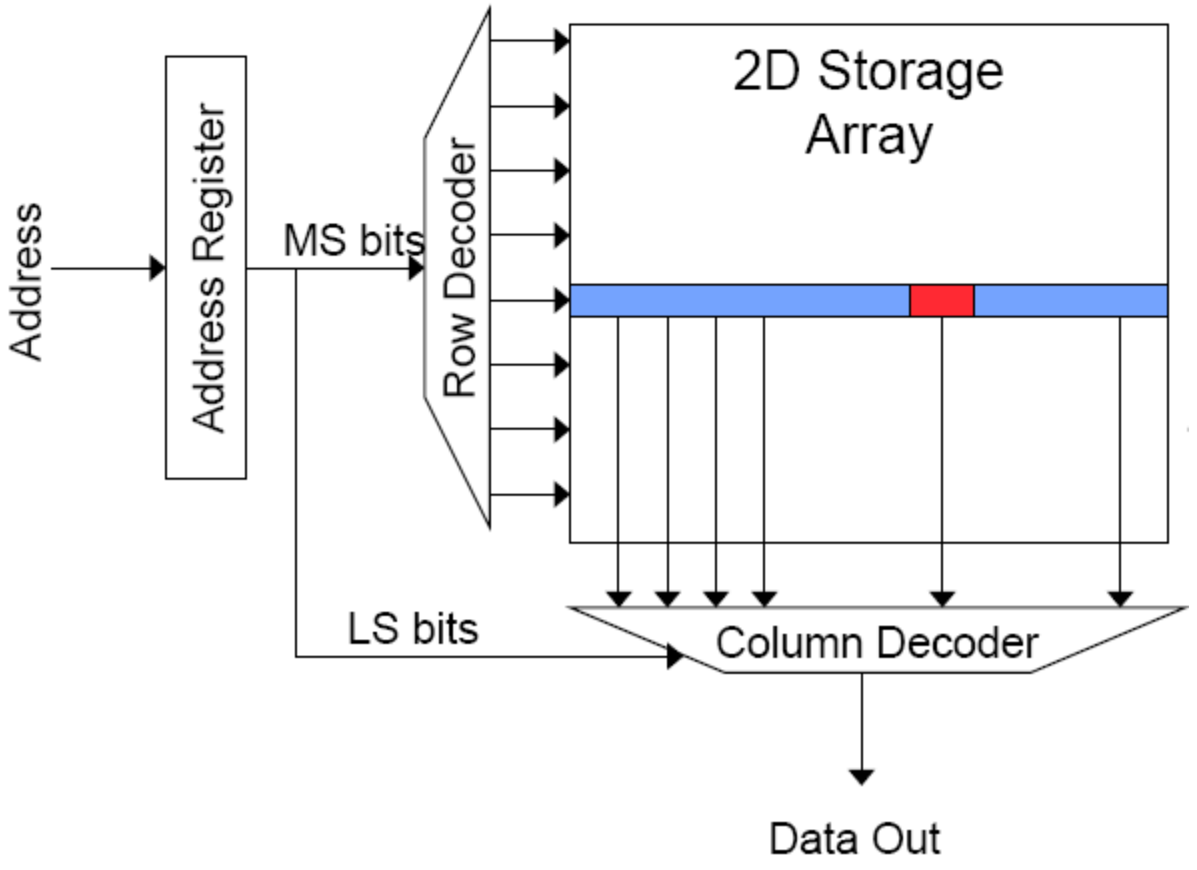
Kim+, "A Case for Exploiting Subarray-Level Parallelism in DRAM," ISCA 2012.

Lee+, "Decoupled Direct Memory Access," PACT 2015.

# DRAM Array Access


16Mb DRAM array = 4096 x 4096 array of bits
12 row address bits arrive first
Row Access Strobe (RAS)
4096 bits are read out
12 column address bits arrive next
Column decoder
Some bits returned to CPU
Column Access Strobe (CAS)
Row Buffer

# Memory Bank Organization and Operation



- Read access sequence:

  1. Decode row address & drive word-lines

  2. Selected bits drive bit-lines
     - Entire row read

  3. Amplify row data

  4. Decode column address & select subset of row
     - Send to output

  5. Precharge bit-lines
     - For next access

## DRAM Main Memory

- Main memory is stored in DRAM cells that have much higher storage density

- DRAM cells lose their state over time – must be refreshed periodically, hence the name *Dynamic*

- DRAM access suffers from long access time and high energy overhead

# DRAM vs. SRAM

- DRAM
  - Slower access (capacitor)
  - Higher density (1T 1C cell)
  - Lower cost
  - Requires refresh (power, performance, circuitry)
  - Manufacturing requires putting capacitor and logic together

- SRAM
  - Faster access (no capacitor)
  - Lower density (6T cell)
  - Higher cost
  - No need for refresh
  - Manufacturing compatible with logic process (no capacitor)

## Organizing a Rank

- DIMM, rank, bank, array → form a hierarchy in the storage organization
- Because of electrical constraints, only a few DIMMs can be attached to a bus
- One DIMM can have 1-4 ranks
- For energy efficiency, use wide-output DRAM chips – better to activate only 4 x16 chips per request than 16 x4 chips
- For high capacity, use narrow-output DRAM chips – since the ranks on a channel are limited, capacity per rank is boosted by having 16 x4 2Gb chips than 4 x16 2Gb chips

## Organizing Banks and Arrays

- A rank is split into many banks (4-16) to boost parallelism within a rank
- Ranks and banks offer memory-level parallelism
- A bank is made up of multiple arrays (subarrays, tiles, mats)
- To maximize density, arrays within a bank are made large → rows are wide → row buffers are wide (8KB read for a 64B request, called overfetch)
- Each array provides a single bit to the output pin in a cycle (for high density)

## Row Buffers

- Each bank has a single row buffer

- Row buffers act as a cache within DRAM
  - Row buffer hit: ~20 ns access time (must only move data from row buffer to pins)
  - Empty row buffer access: ~40 ns (must first read arrays, then move data from row buffer to pins)
  - Row buffer conflict: ~60 ns (must first precharge the bitlines, then read new row, then move data to pins)

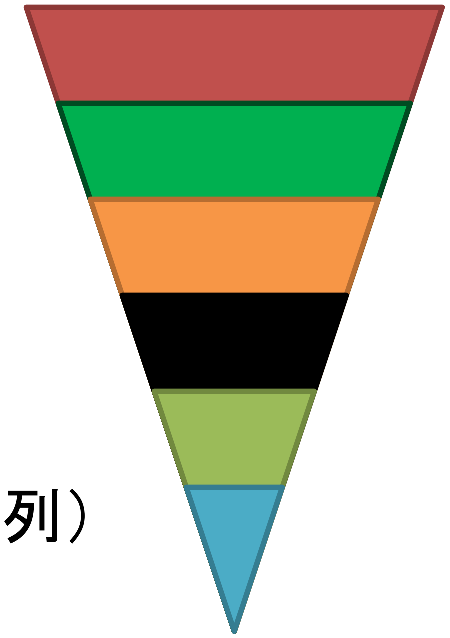
- In addition, must wait in the queue (tens of nano-seconds) and incur address/cmd/data transfer delays (~10 ns)

# Building Larger Memories

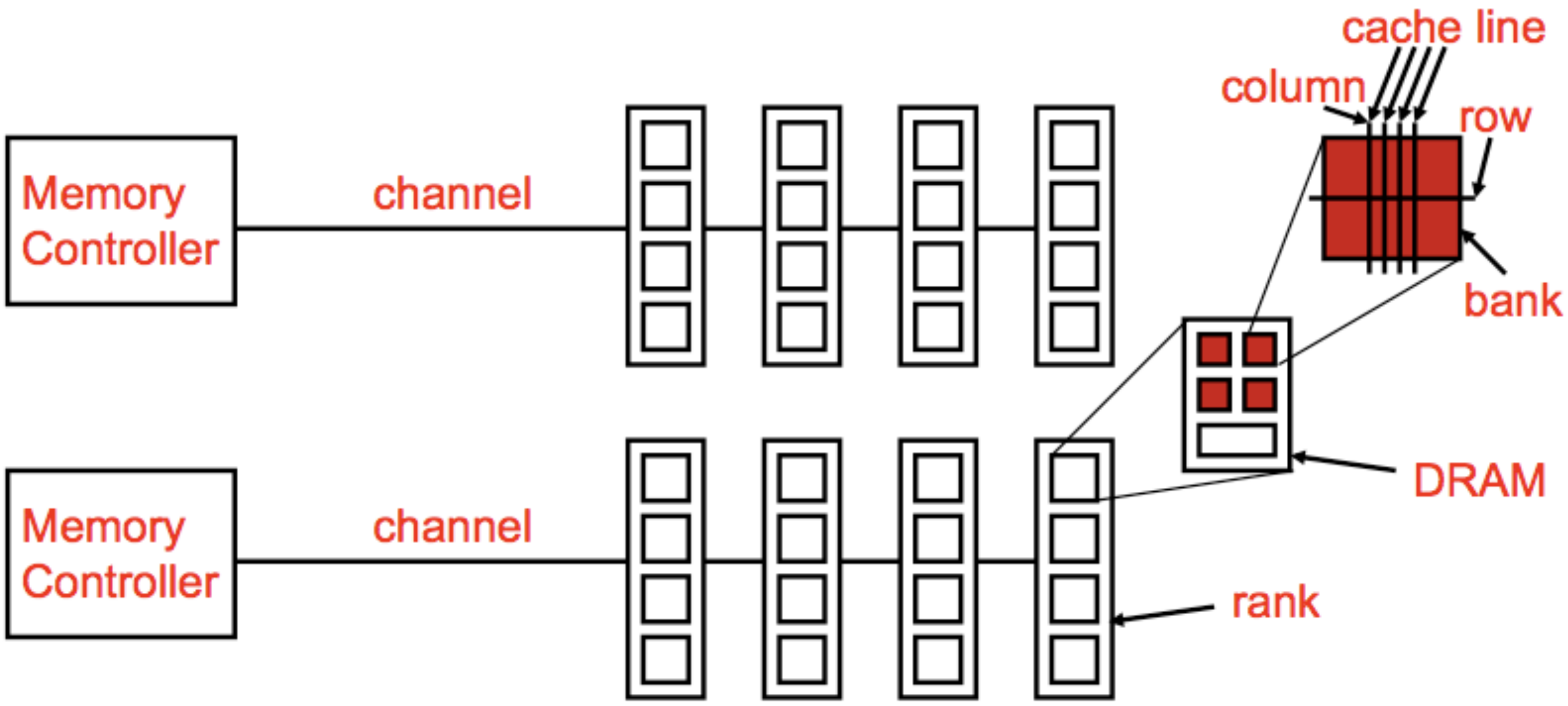
- Requires larger memory arrays
- Large → slow
- How do we make the memory large without making it very slow?
- Idea: Divide the memory into smaller arrays and interconnect the arrays to input/output buses
  - Large memories are hierarchical array structures
  - DRAM: Channel → Rank → Bank → Subarrays → Mats

## DRAM Subsystem Organization

- Channel（通道）
- DIMM（模组）
- Rank（内存组）
- Chip（内存芯片）
- Bank（内存库）
- Row/Column（矩阵行列）



# Generalized Memory Structure

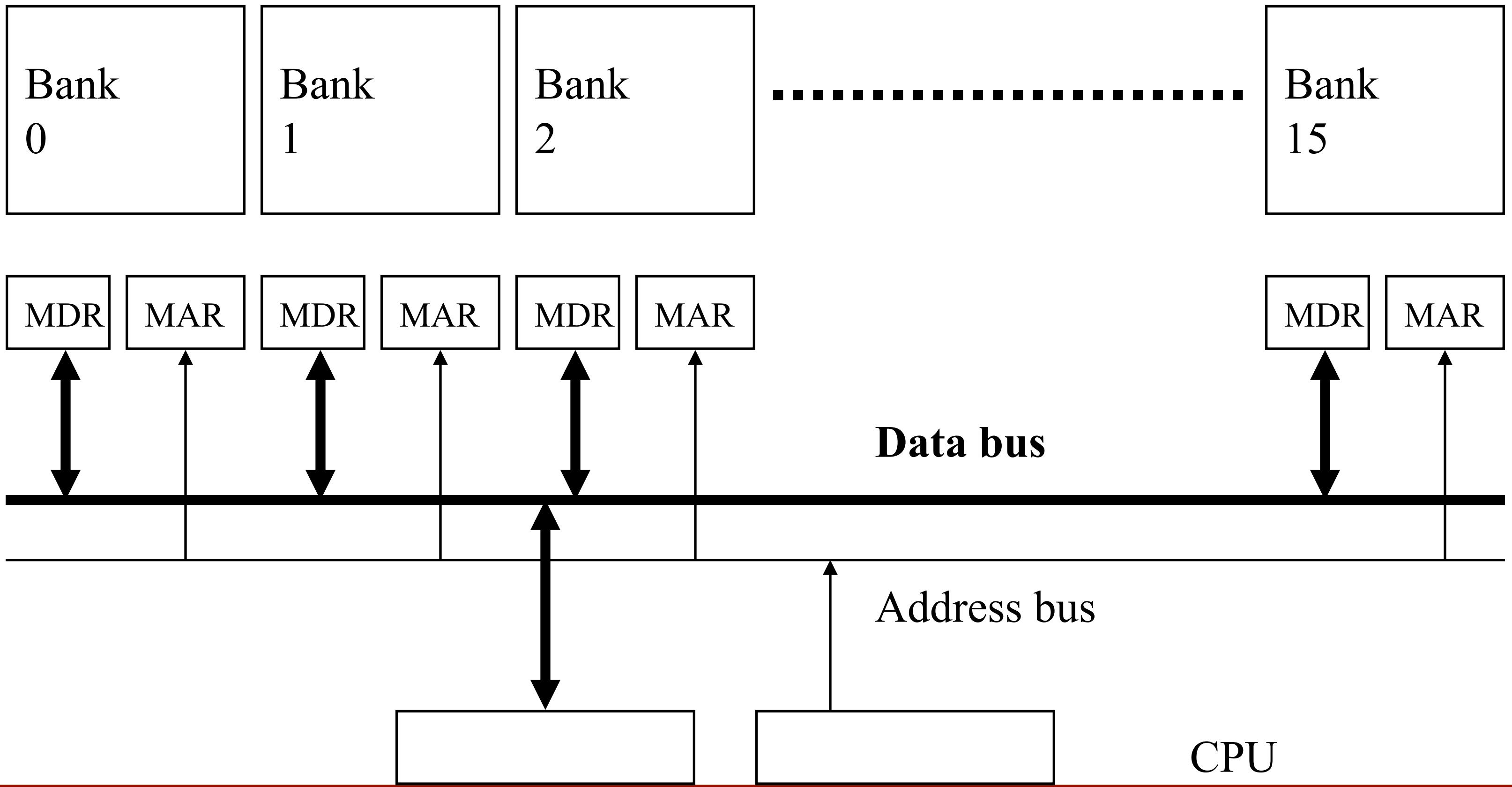


# General Principle: Interleaving (Banking)

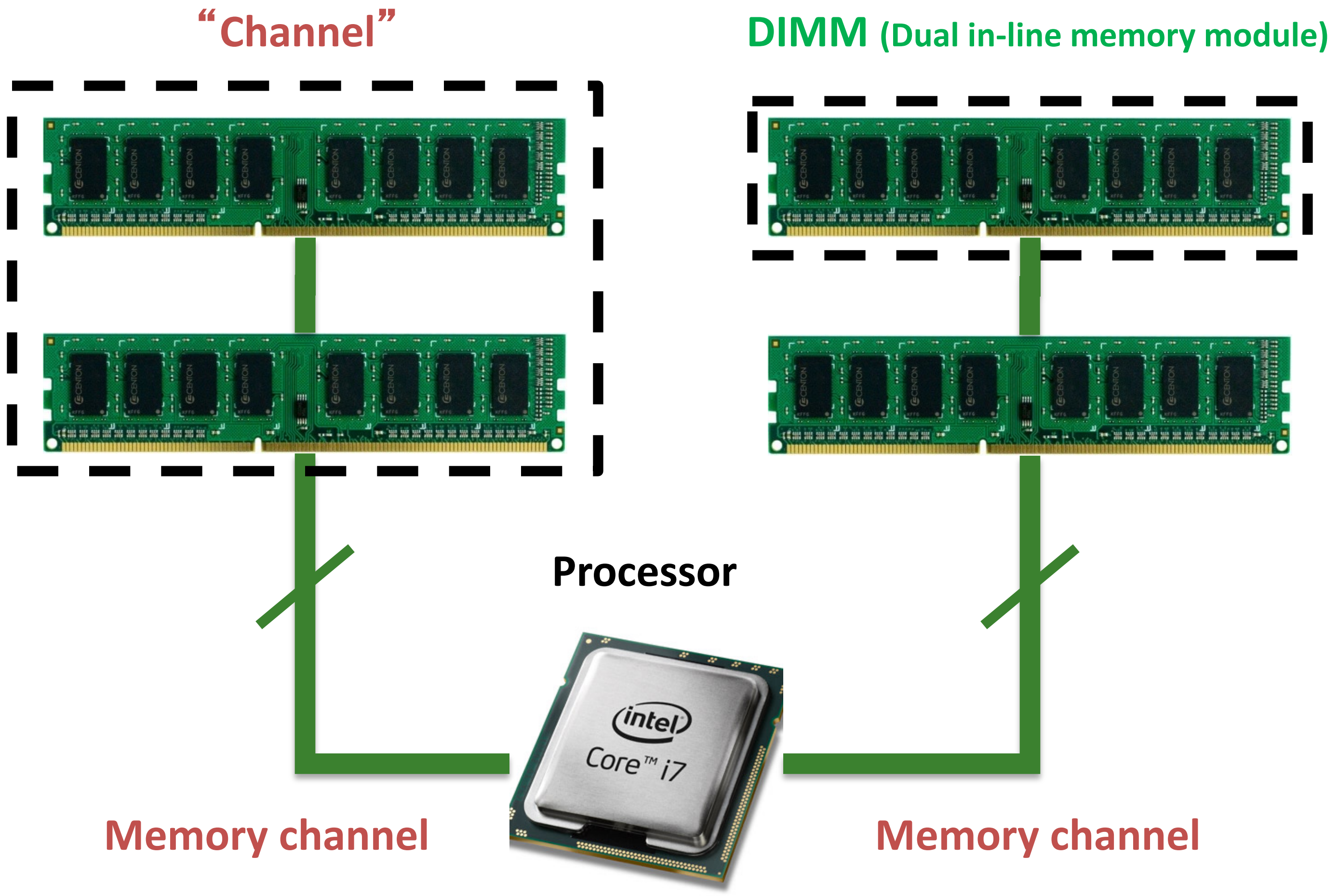
- Interleaving （交替访问）(banking 存储库)
  - Problem: a single monolithic large memory array takes long to access and does not enable multiple accesses in parallel
  - Goal: Reduce the latency of memory array access and enable multiple accesses in parallel
  - Idea: Divide a large array into multiple banks that can be accessed independently (in the same cycle or in consecutive cycles)
    - Each bank is smaller than the entire memory storage
    - Accesses to different banks can be overlapped （重叠访问）
    - The access latency can be manageable (延迟可控)

# Memory Banking Example

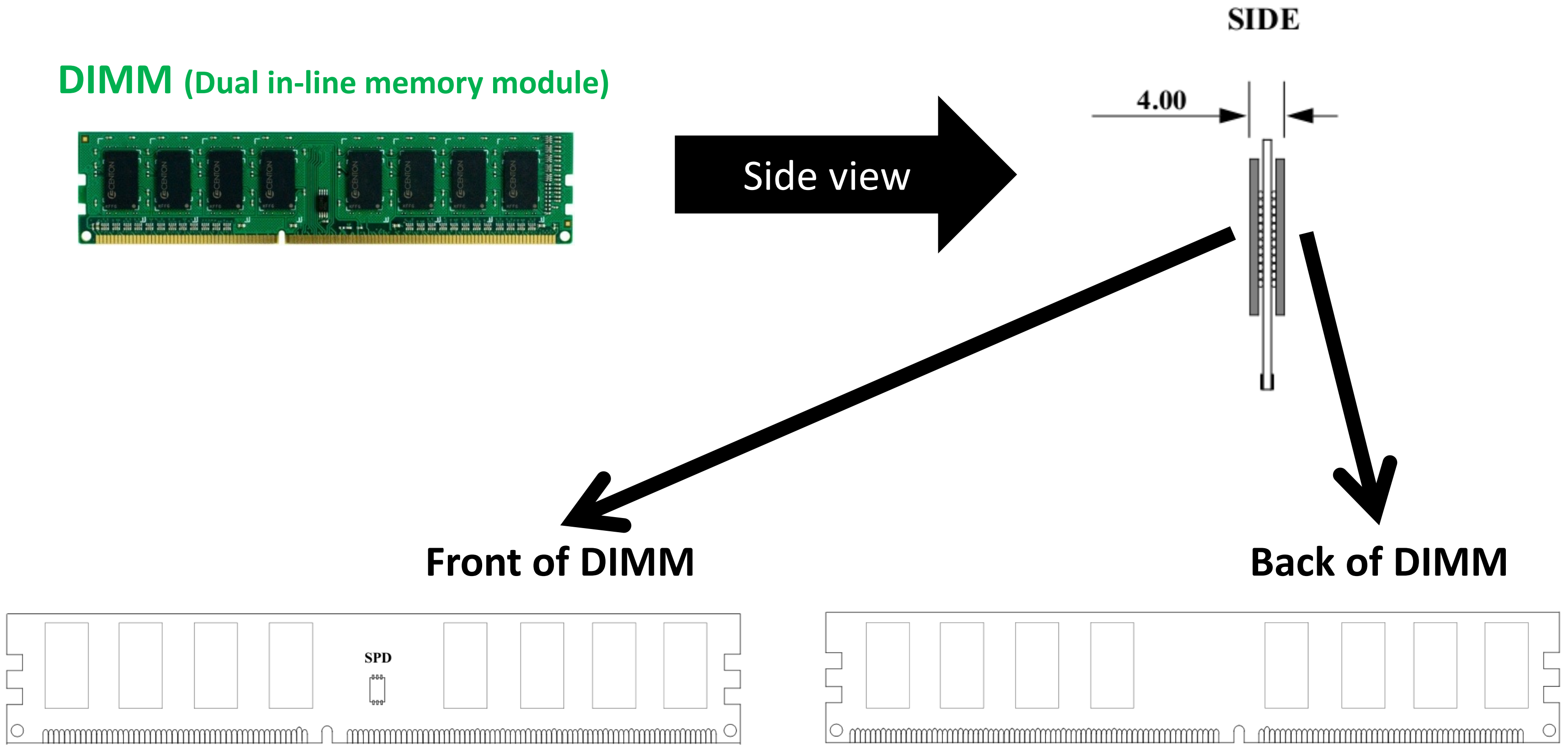
- Memory is divided into banks that can be accessed independently; banks share address and data buses (to reduce memory chip pins)
- Can start and complete one bank access per cycle
- Can sustain N concurrent accesses if all N go to different banks



# The DRAM Subsystem

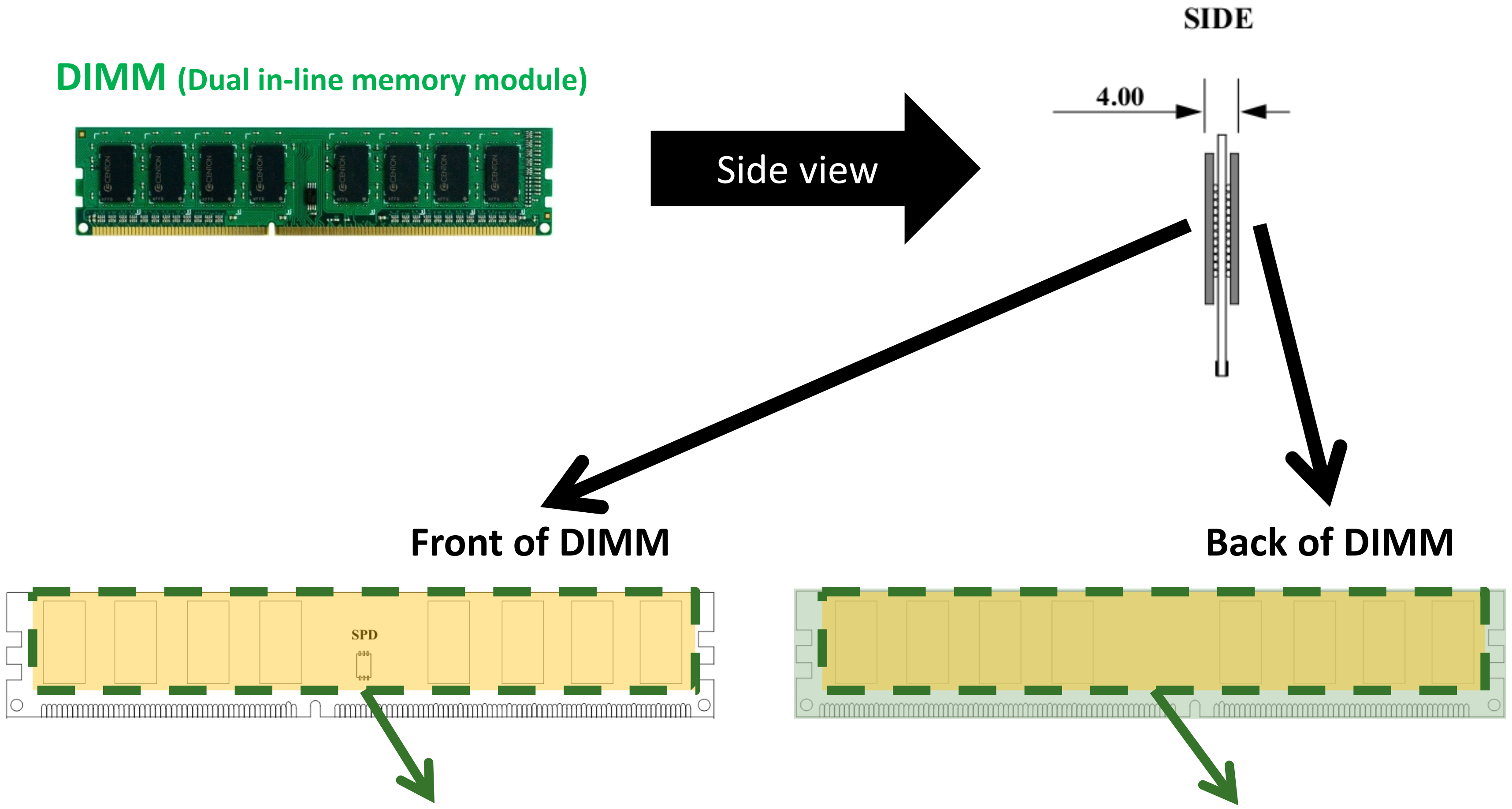


# Breaking down a DIMM (module)



## Breaking down a DIMM (module)

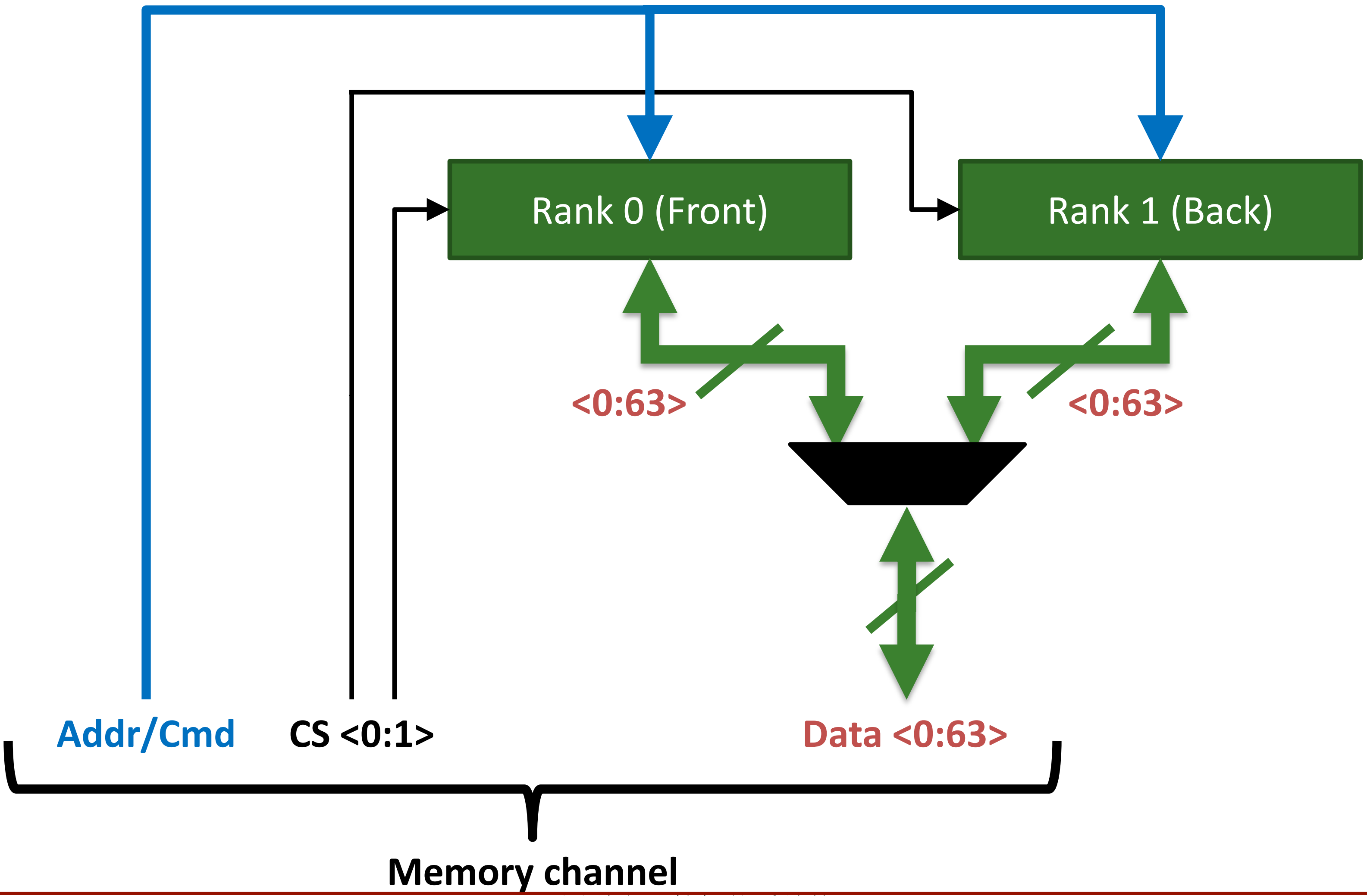
**DIMM** **(Dual in-line memory module)**



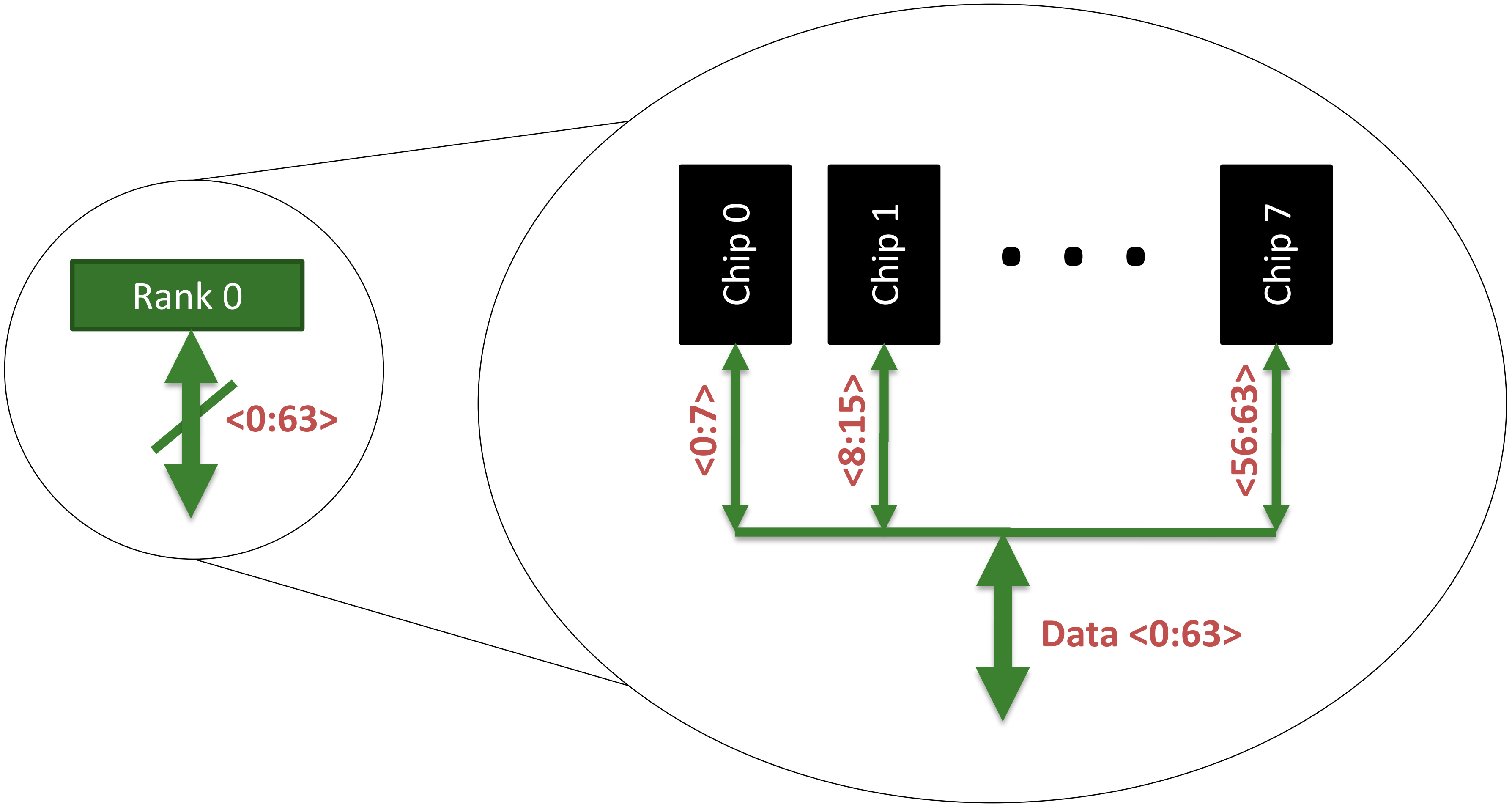
**Rank 0:** collection of 8 chips

**Rank 1**

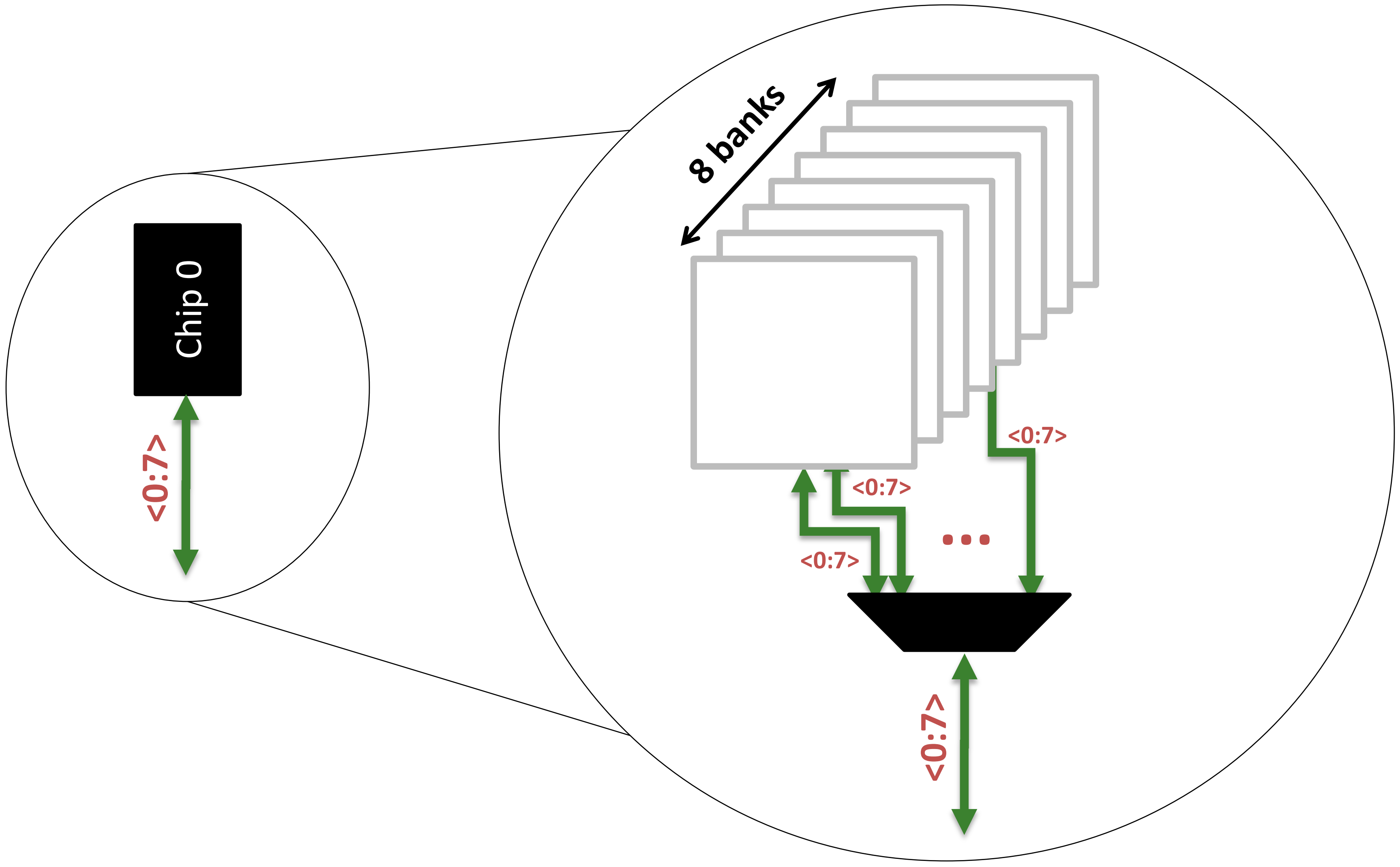
# Rank


Rank 0 (Front)
Rank 1 (Back)
<0:63>
<0:63>
Addr/Cmd
CS <0:1>
Data <0:63>
Memory channel

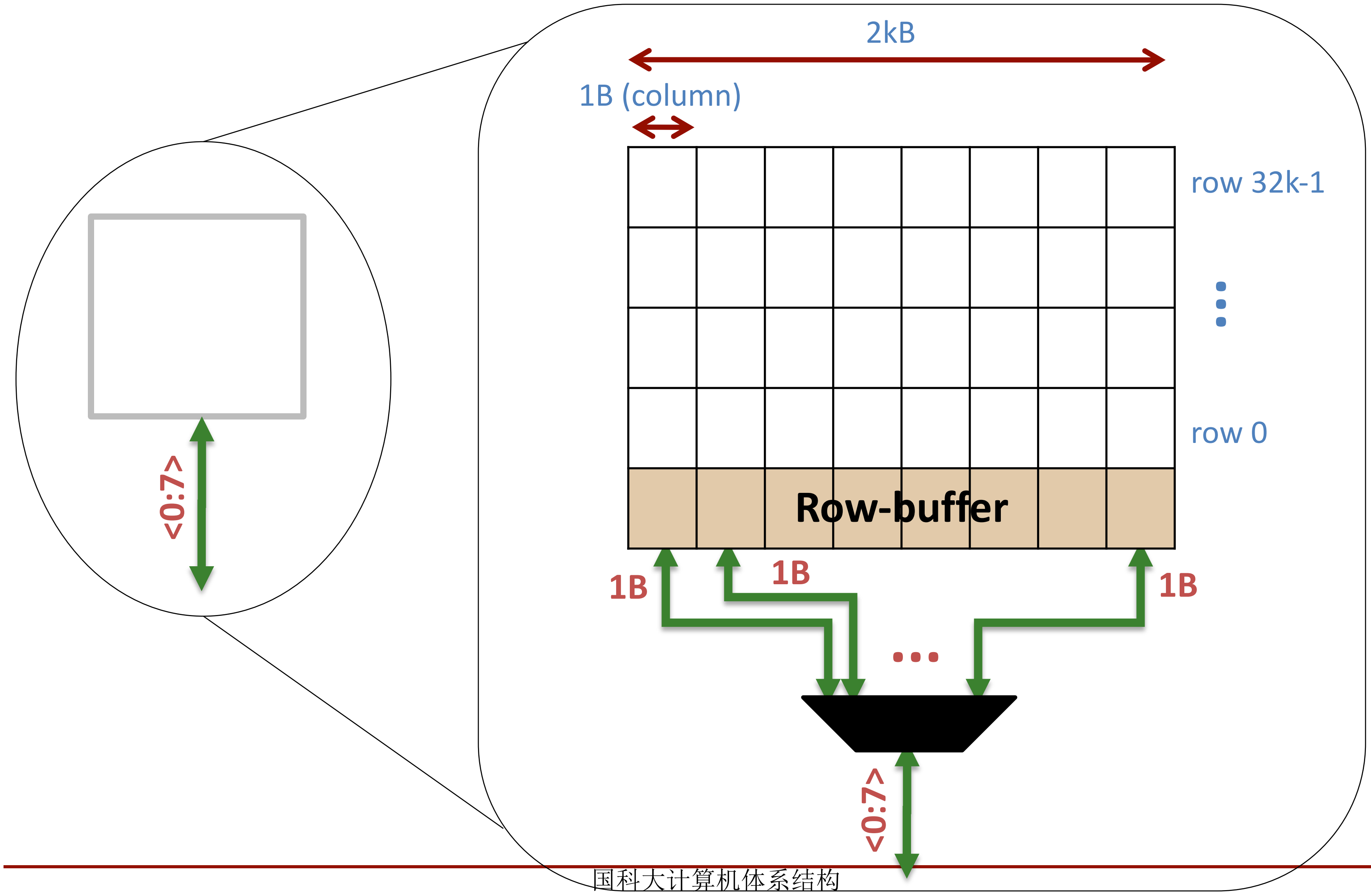
## Breaking down a Rank



## Breaking down a Chip

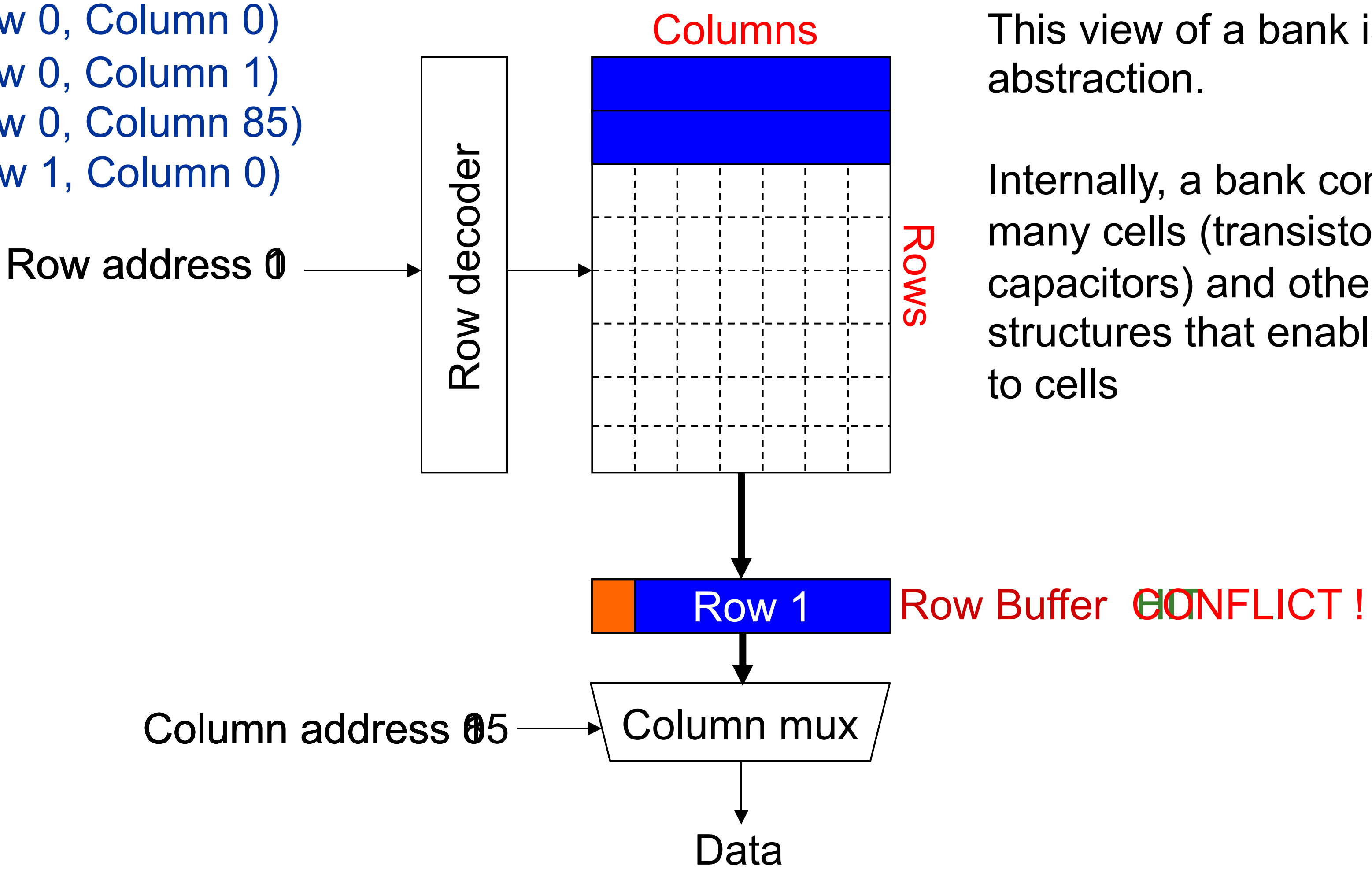


# Breaking down a Bank



# Digging Deeper: DRAM Bank Operation

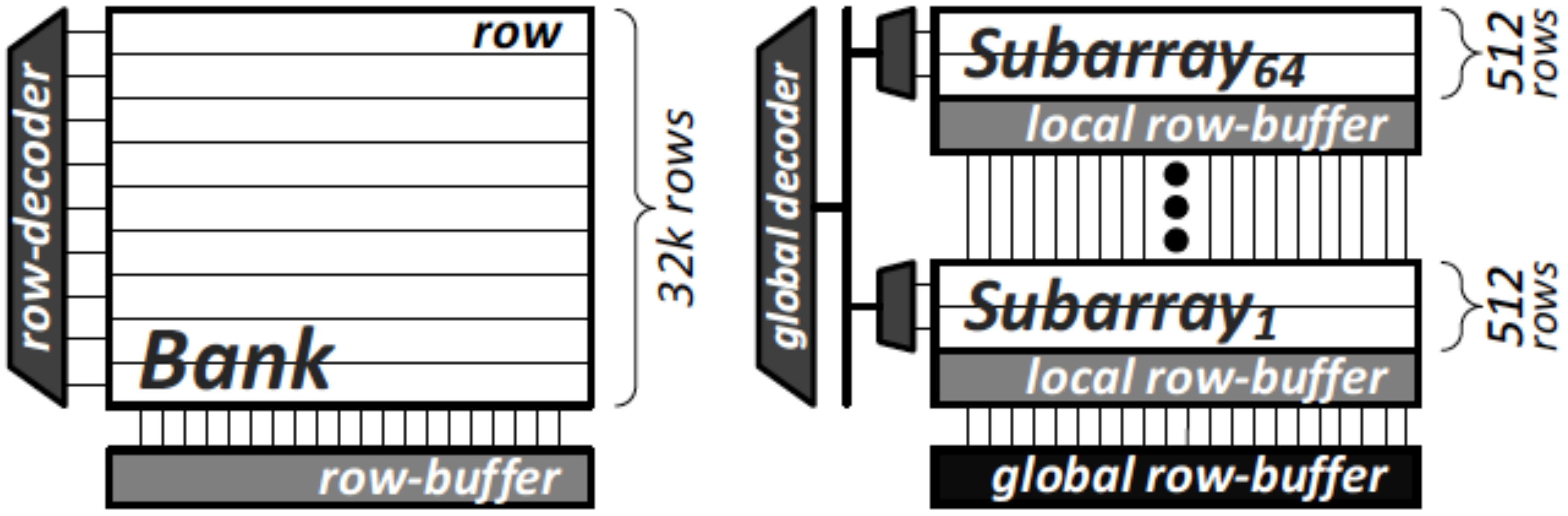
Access Address:
(Row 0, Column 0)
(Row 0, Column 1)
(Row 0, Column 85)
(Row 1, Column 0)



This view of a bank is an abstraction.

Internally, a bank consists of many cells (transistors & capacitors) and other structures that enable access to cells

# A DRAM Bank Internally Has Sub-Banks

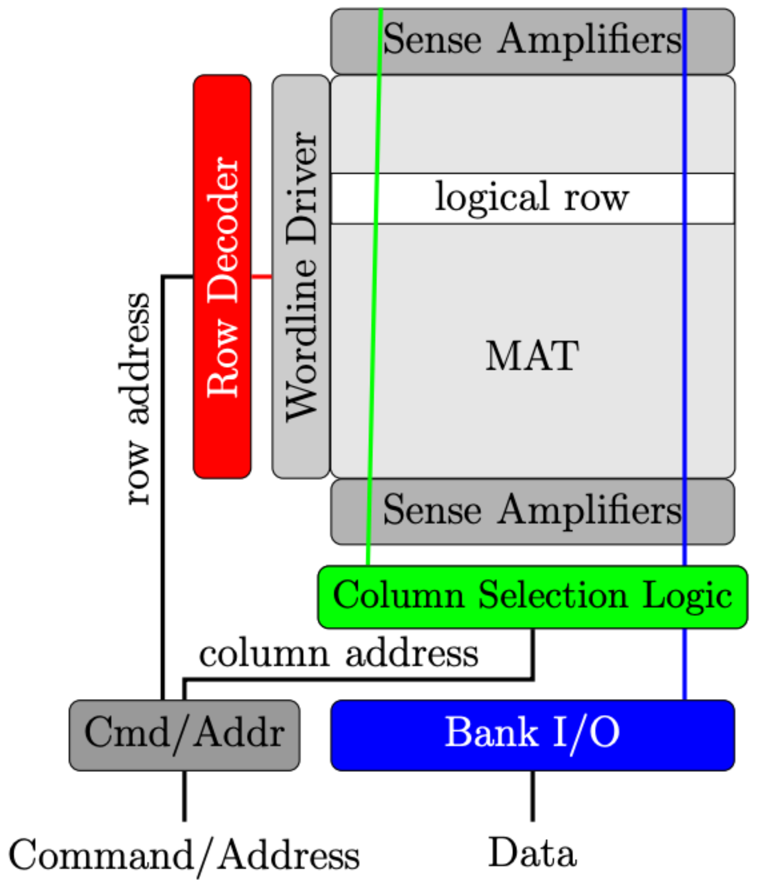


(a) Logical abstraction (b) Physical implementation

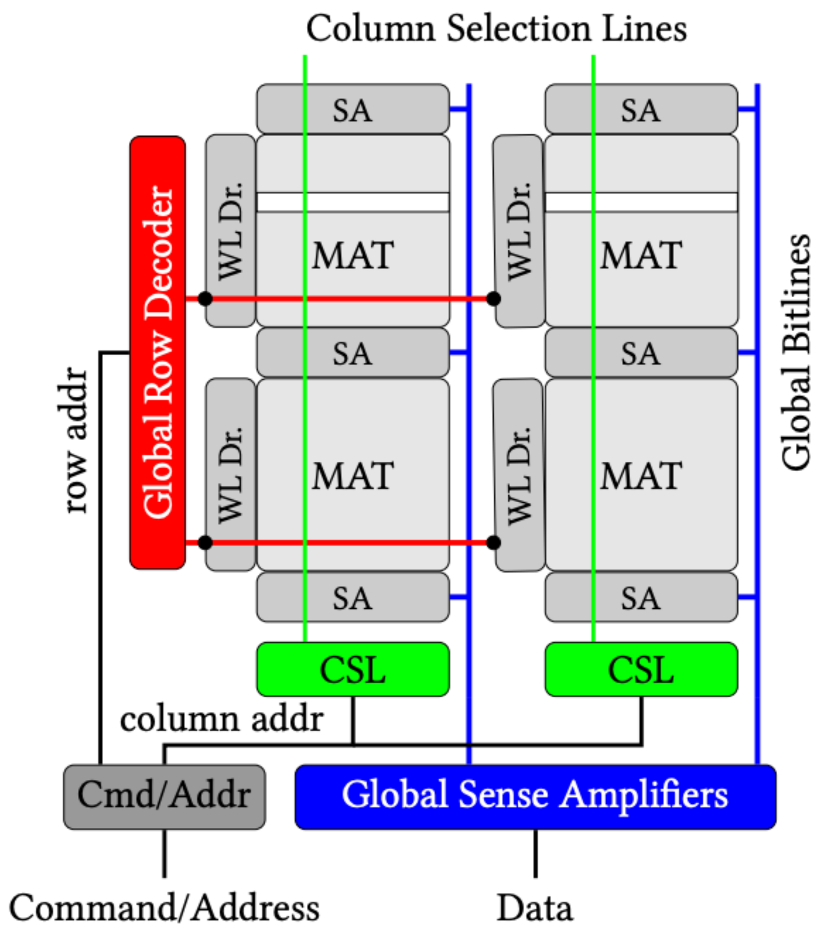
**Figure 1.** DRAM bank organization

Kim et al., "A Case for Exploiting Subarray-Level Parallelism in DRAM," ISCA 2012.

# Another View of a DRAM Bank



**Logical Abstraction**

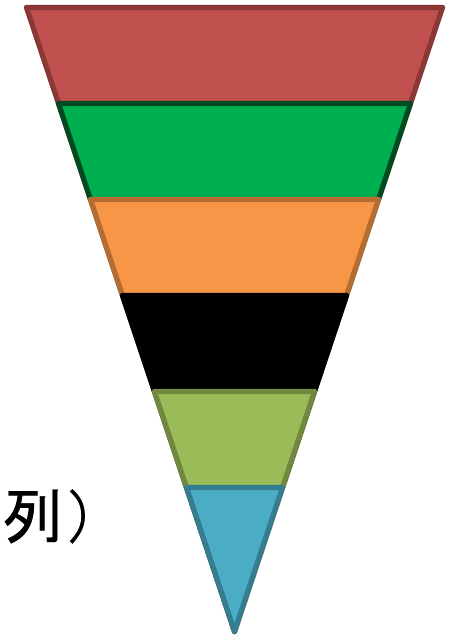


**Physical View**

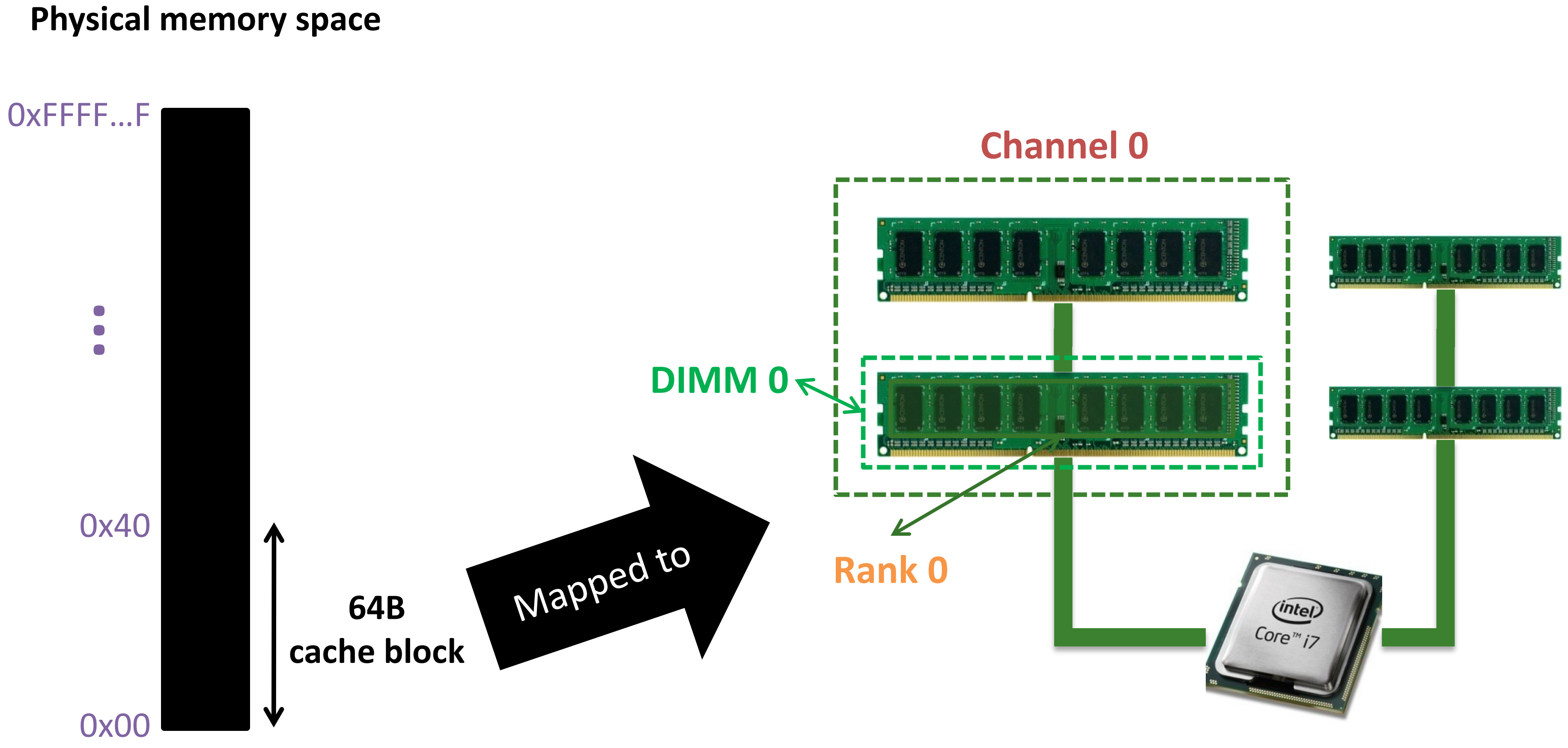
Seshadri+, "In-DRAM Bulk Bitwise Execution Engine," ADCOM 2020.

# DRAM Subsystem Organization

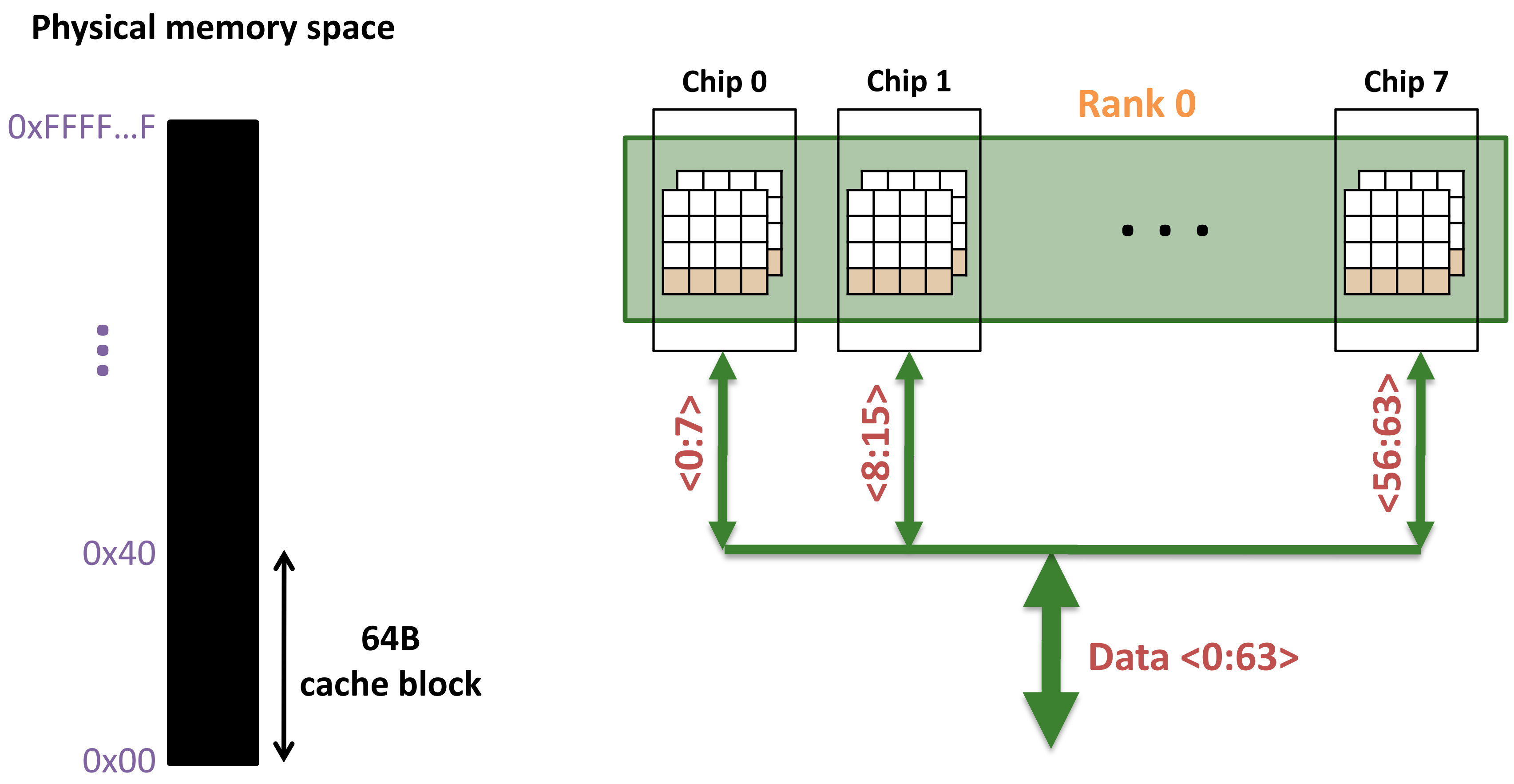
- Channel（通道）
- DIMM（模块）
- Rank（内存组）
- Chip（内存芯片）
- Bank（内存库）
- Row/Column（矩阵行列）



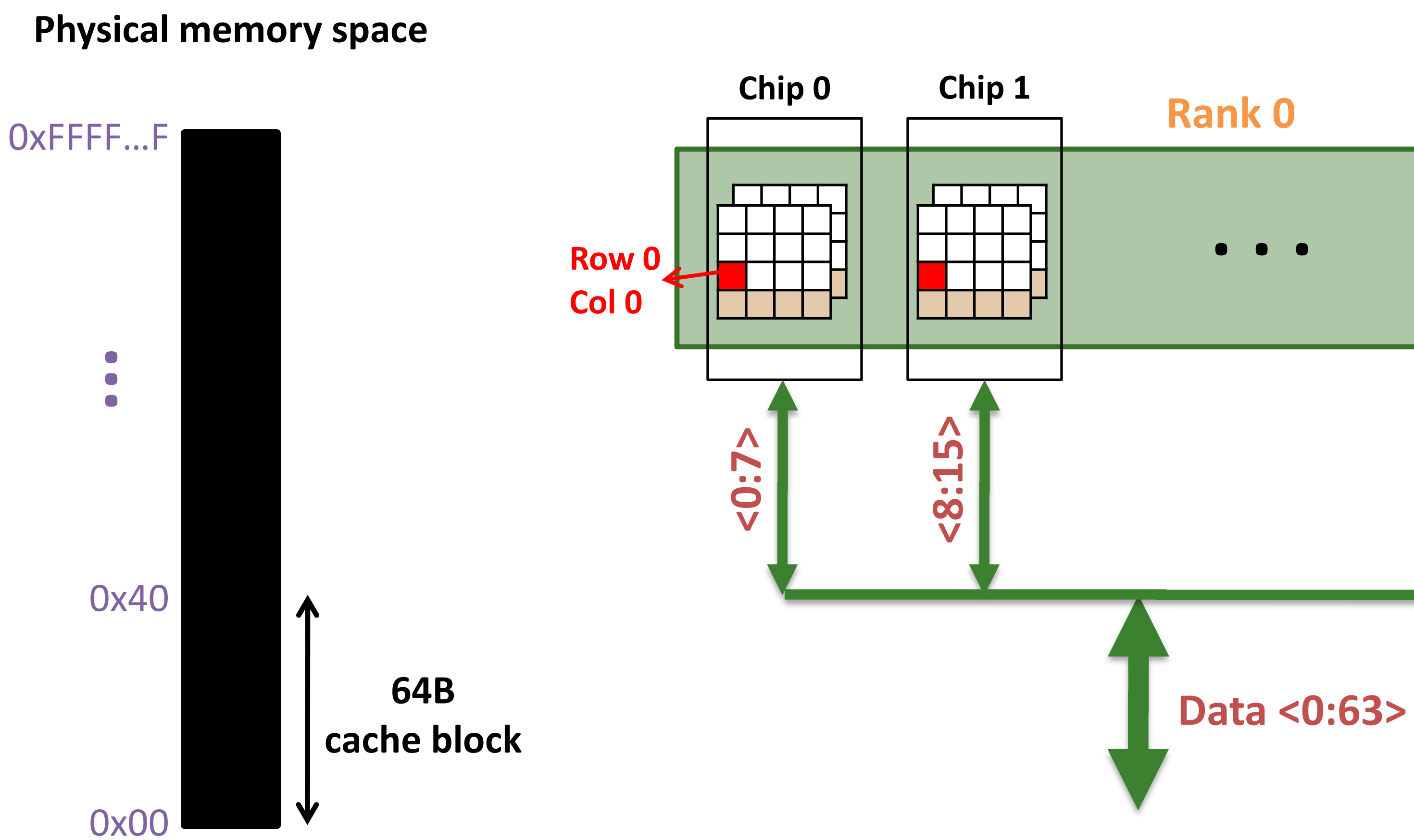
# Example: Transferring a cache block



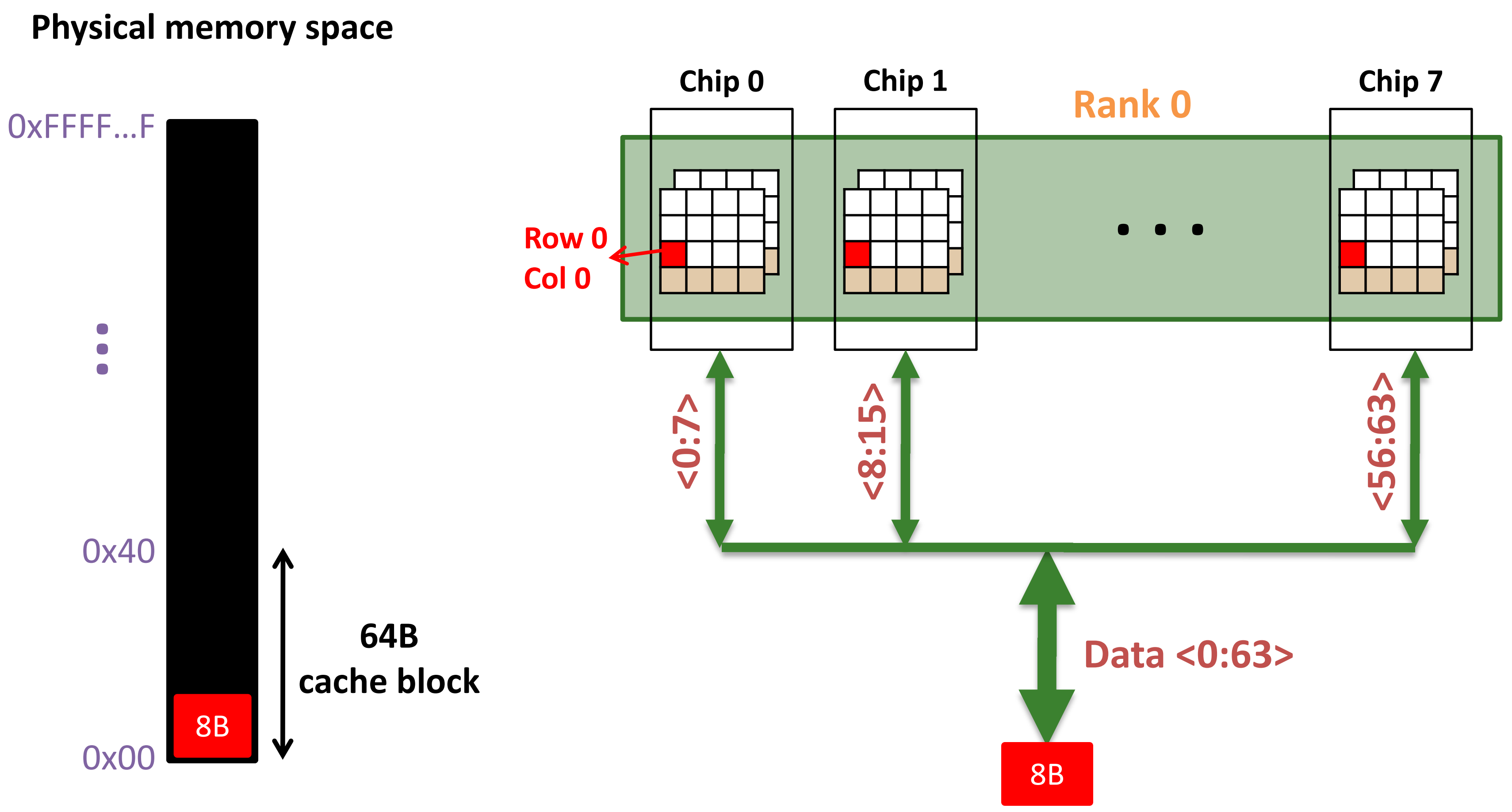
# Example: Transferring a cache block



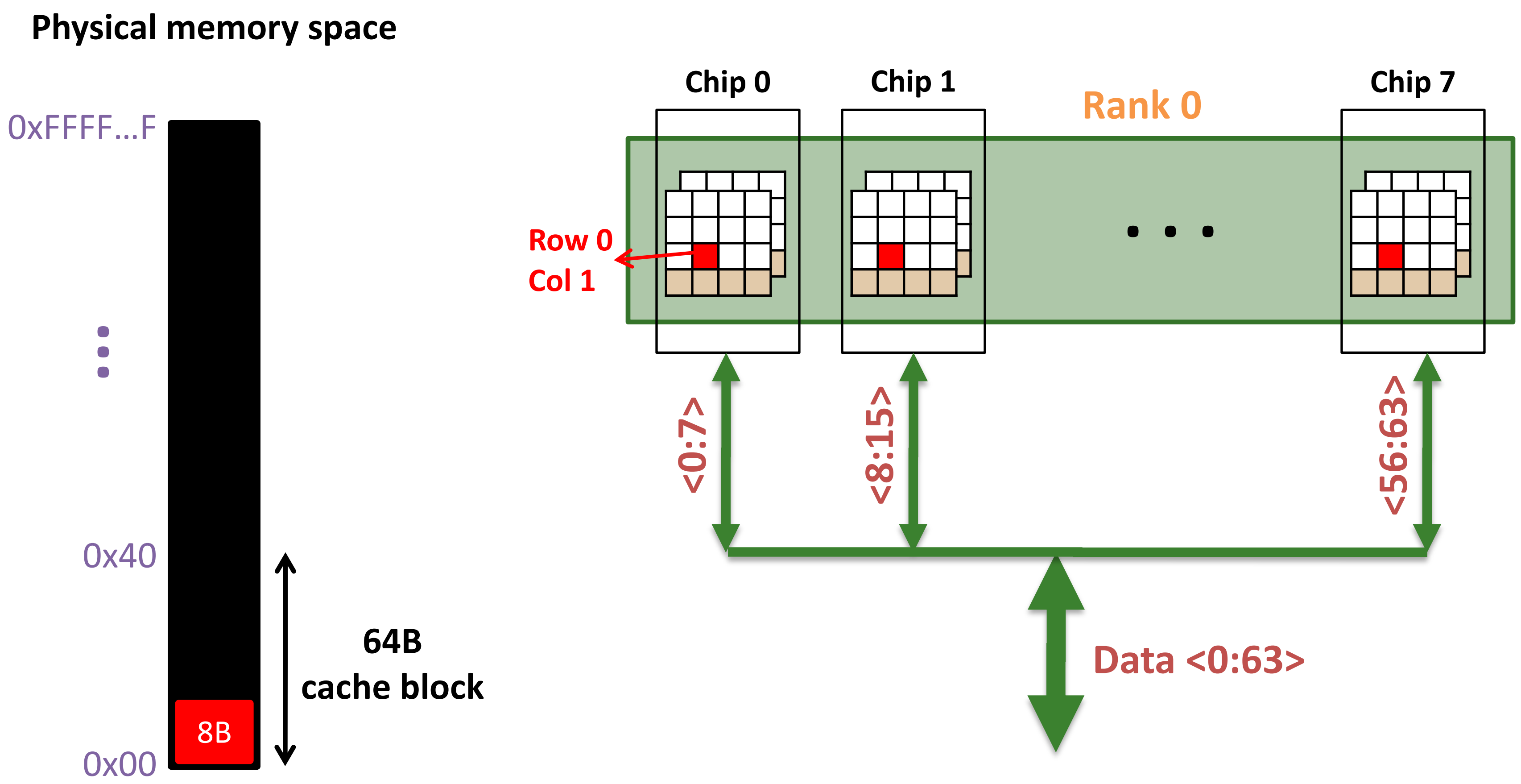
# Example: Transferring a cache block



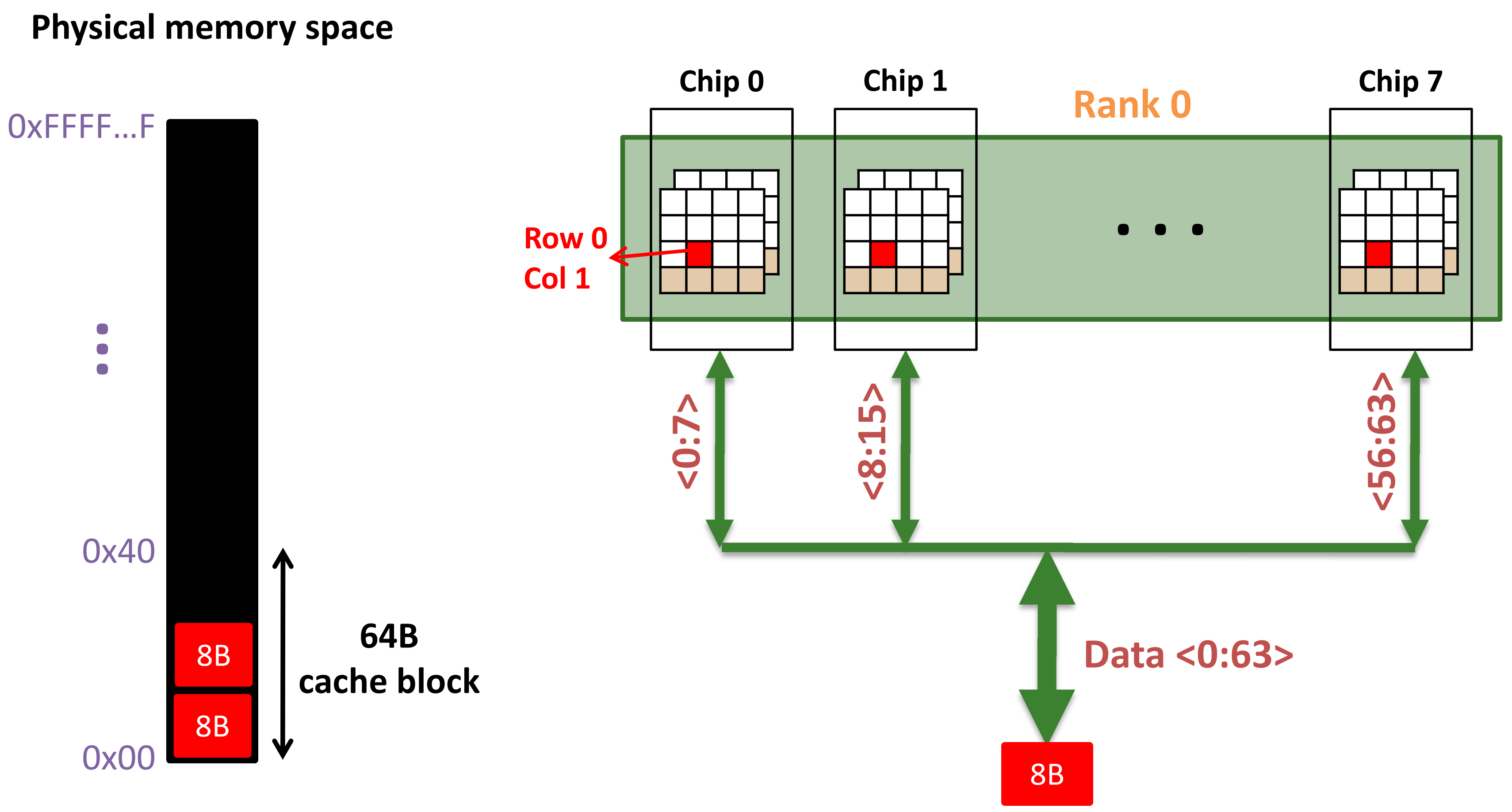
# Example: Transferring a cache block



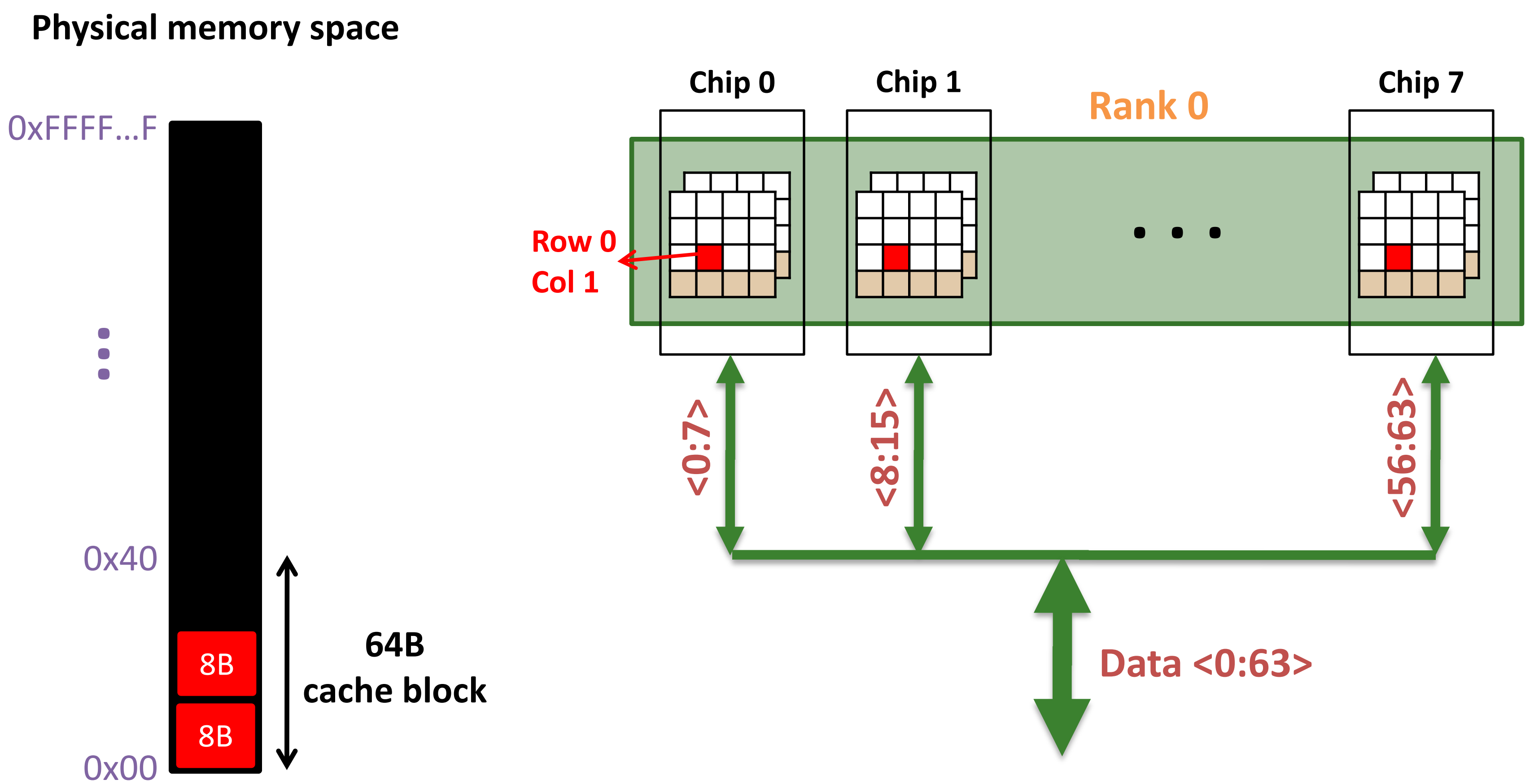
# Example: Transferring a cache block



# Example: Transferring a cache block



# Example: Transferring a cache block



A 64B cache block takes 8 I/O cycles to transfer.

During the process, 8 columns are read sequentially.

# Memory Controller

# Open/Closed Page Policies

- If an access stream has locality, a row buffer is kept open
  - Row buffer hits are cheap (open-page policy)
  - Row buffer miss is a bank conflict and expensive because precharge is on the critical path

- If an access stream has little locality, bitlines are precharged immediately after access (close-page policy)
  - Nearly every access is a row buffer miss
  - The precharge is usually not on the critical path

- Modern memory controller policies lie somewhere between these two extremes (usually proprietary)

## Reads and Writes

- A single bus is used for reads and writes
- The bus direction must be reversed when switching between reads and writes; this takes time and leads to bus idling
- Hence, writes are performed in bursts; a write buffer stores pending writes until a high water mark is reached
- Writes are drained until a low water mark is reached

## Address Mapping Policies

- Consecutive cache lines can be placed in the same row to boost row buffer hit rates
- Consecutive cache lines can be placed in different ranks to boost parallelism
- Example address mapping policies:
  - row:rank:bank:channel:column:blkoffset
  - row:column:rank:bank:channel:blkoffset

## Scheduling Policies

- FCFS: Issue the first read or write in the queue that is ready for issue
- First Ready - FCFS: First issue row buffer hits if you can
- Stall Time Fair: First issue row buffer hits, unless other threads are being neglected

## Refresh

- Every DRAM cell must be refreshed within a 64 ms window
- A row read/write automatically refreshes the row
- Every refresh command performs refresh on a number of rows, the memory system is unavailable during that time
- A refresh command is issued by the memory controller once every 7.8us on average

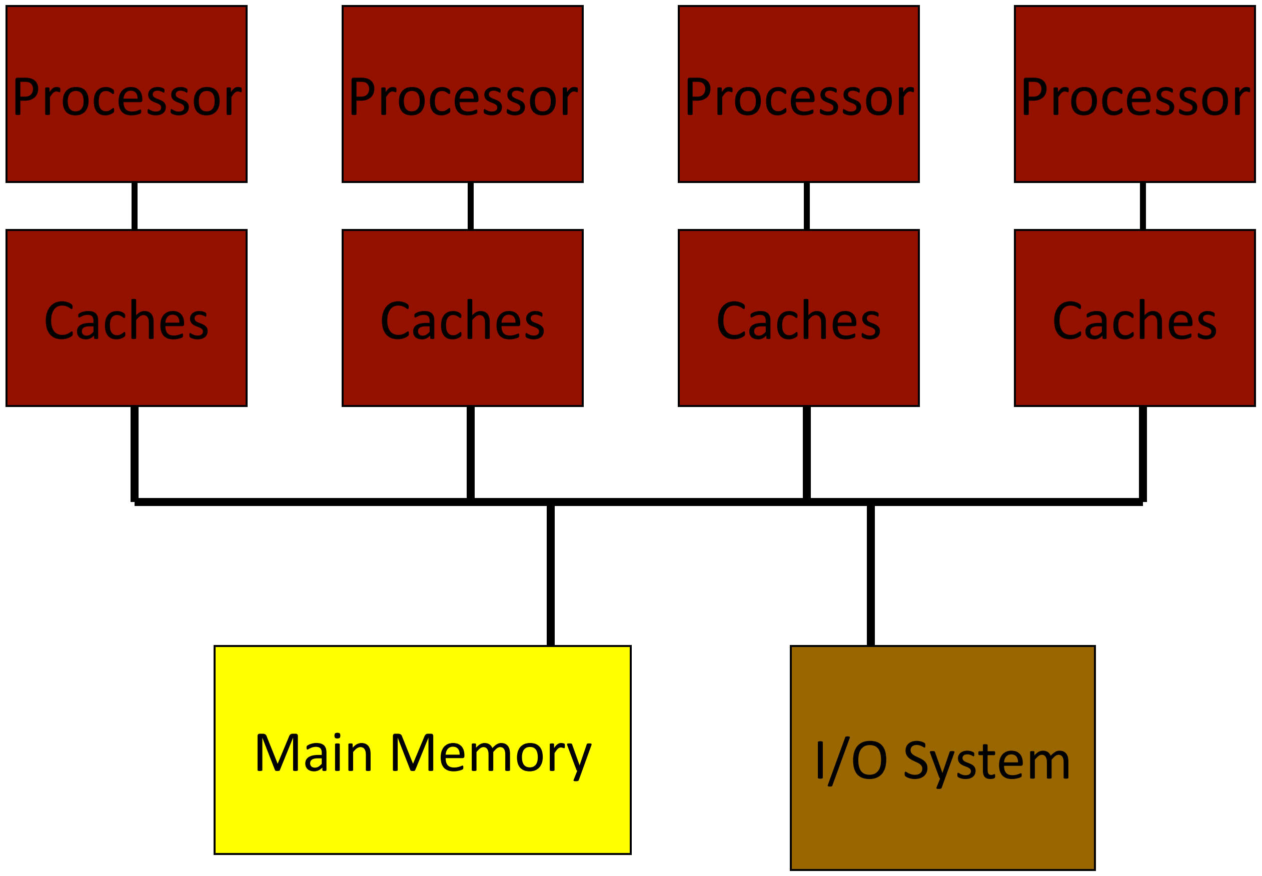
## The following slides will be covered in the next lecture

## Coherent Protocols

## Multiprocs -- Memory Organization - I

- Centralized shared-memory multiprocessor  or Symmetric shared-memory multiprocessor (SMP)
- Multiple processors connected to a single centralized memory – since all processors see the same memory organization -> uniform memory access (UMA)
- Shared-memory because all processors can access the entire memory address space
- Can centralized memory emerge as a bandwidth bottleneck? – not if you have large caches and employ fewer than a dozen processors

# SMPs or Centralized Shared-Memory



# SMPs

- Centralized main memory and many caches -> many copies of the same data

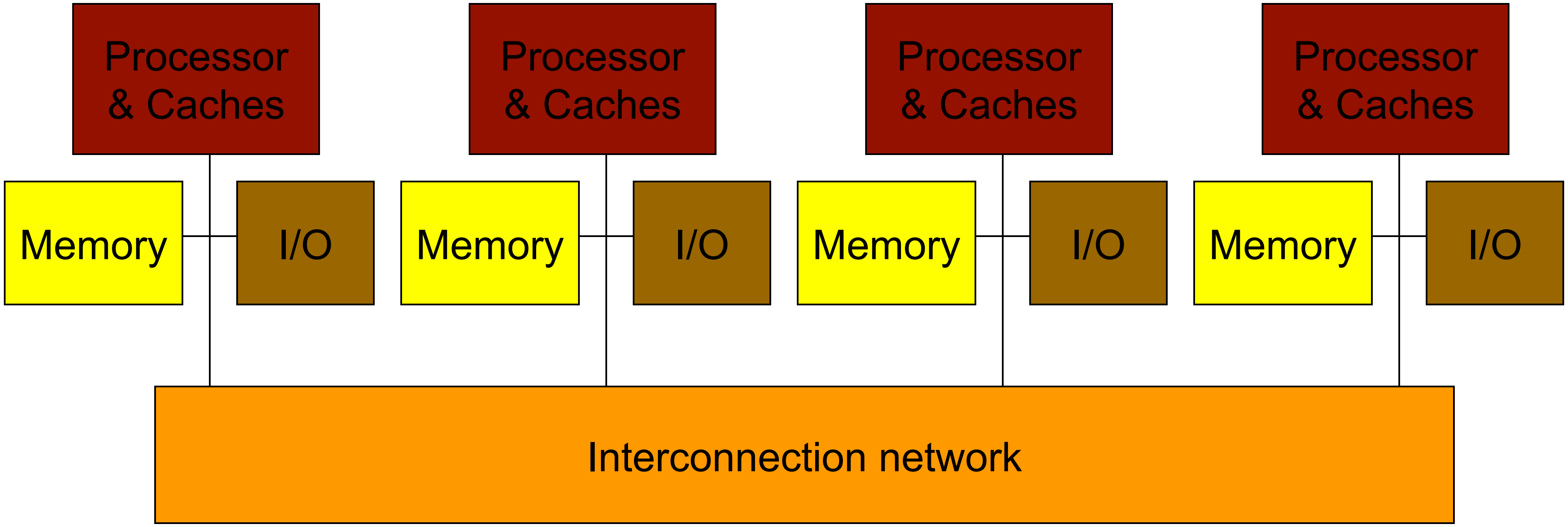
- A system is cache coherent if a read returns the most recently written value for that word

| Time | Event | Value of X in Cache-A | Cache-B | Memory |
|---|---|---|---|---|
| 0 | | - | - | 1 |
| 1 | CPU-A reads X | 1 | - | 1 |
| 2 | CPU-B reads X | 1 | 1 | 1 |
| 3 | CPU-A stores 0 in X | 0 | 1 | 0 |

## Multiprocs -- Memory Organization - II

- For higher scalability, memory is distributed among processors -> distributed memory multiprocessors
- If one processor can directly address the memory local to another processor, the address space is shared -> distributed shared-memory (DSM) multiprocessor
- If memories are strictly local, we need messages to communicate data -> cluster of computers or multicomputers
- Non-uniform memory architecture (NUMA) since local memory has lower latency than remote memory

# Distributed Memory Multiprocessors



## Cache Coherence

- A memory system is coherent if:

- Write propagation: P1 writes to X, sufficient time elapses, P2 reads X and gets the value written by P1

- Write serialization: Two writes to the same location by two processors are seen in the same order by all processors

- The memory consistency model defines "time elapsed" before the effect of a processor is seen by others and the ordering with R/W to other locations (loosely speaking – more later)

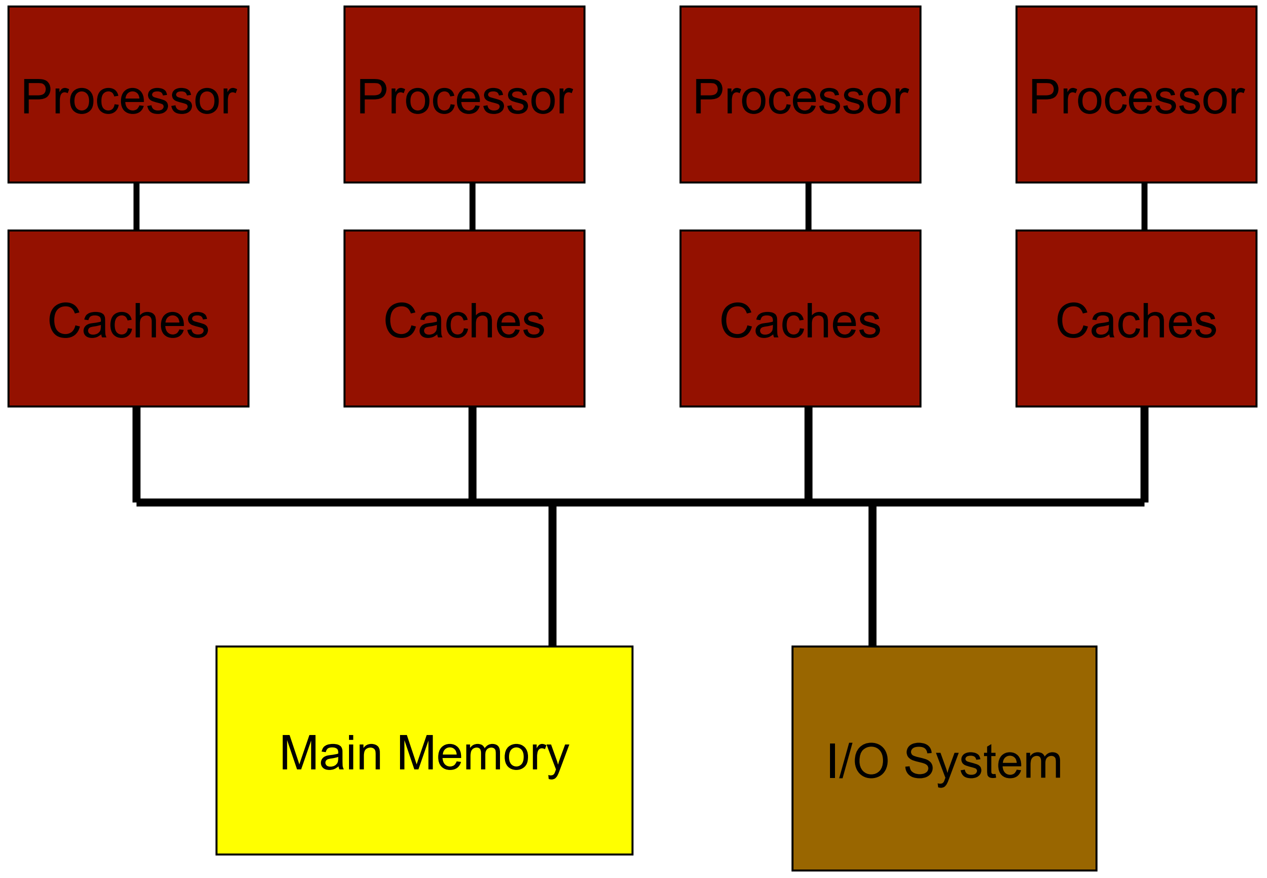
# Cache Coherence Protocols

- Directory-based: A single location (directory) keeps track of the sharing status of a block of memory

- Snooping: Every cache block is accompanied by the sharing status of that block – all cache controllers monitor the shared bus so they can update the sharing status of the block, if necessary

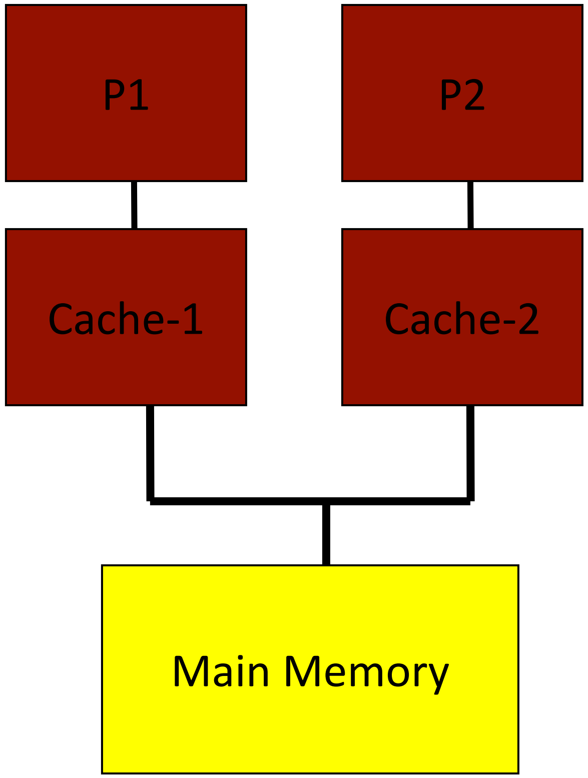
  - Write-invalidate: a processor gains exclusive access of a block before writing by invalidating all other copies
  - Write-update: when a processor writes, it updates other shared copies of that block

# SMPs or Centralized Shared-Memory



# SMP Example

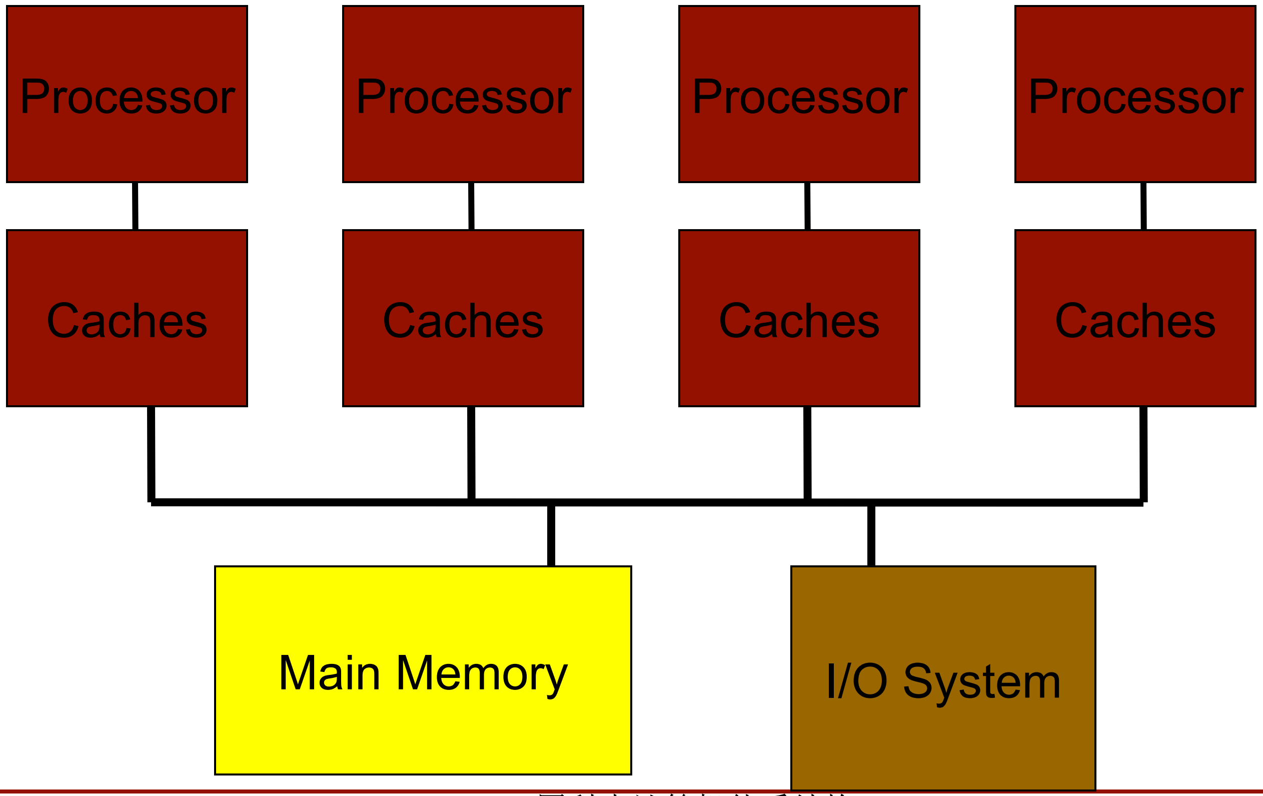
- P1 reads X: not found in cache-1, request sent on bus, memory responds, X is placed in cache-1 in shared state
- P2 reads X: not found in cache-2, request sent on bus, everyone snoops this request, cache-1 does nothing because this is just a read request, memory responds, X is placed in cache-2 in shared state



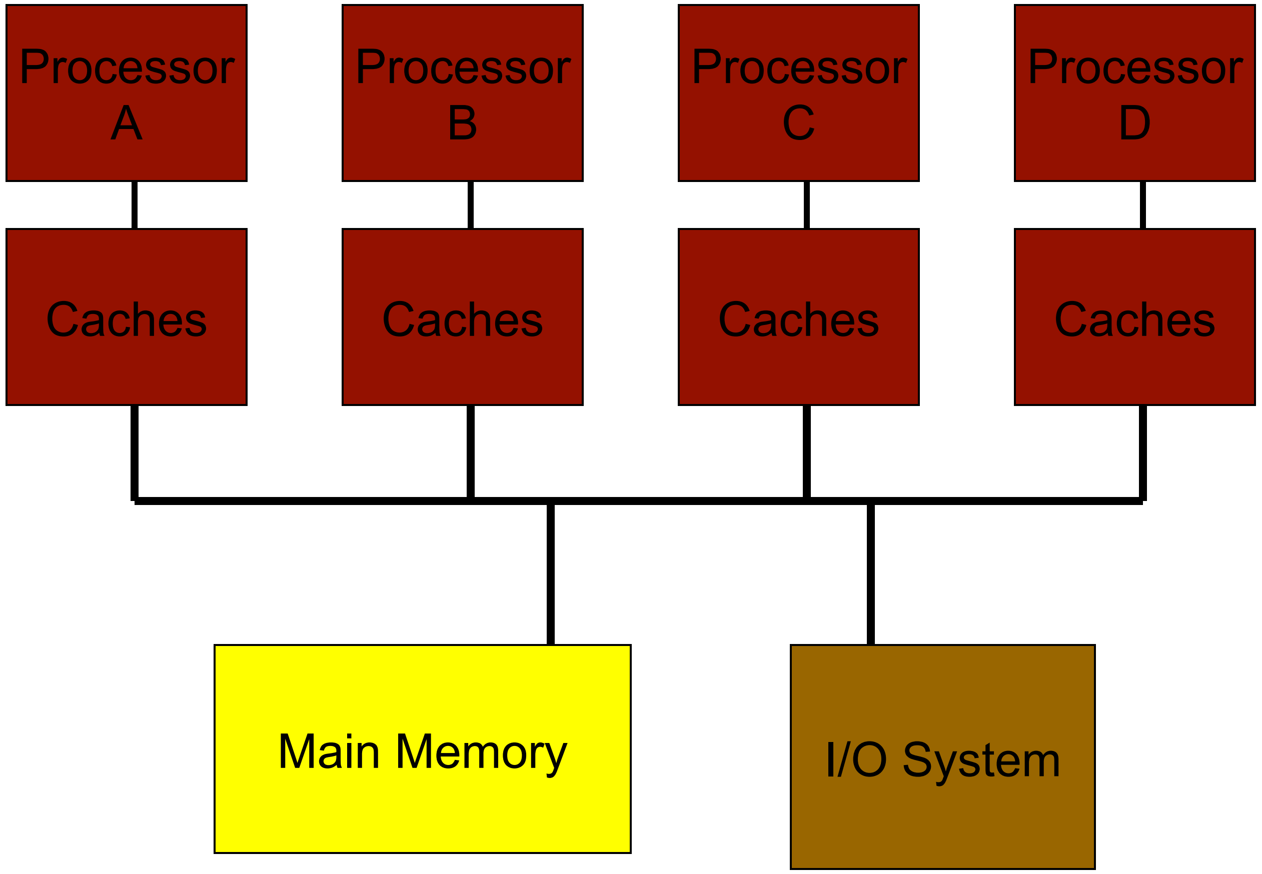
- P1 writes X: cache-1 has data in shared state (shared only provides read perms), request sent on bus, cache-2 snoops and then invalidates its copy of X, cache-1 moves its state to modified
- P2 reads X: cache-2 has data in invalid state, request sent on bus, cache-1 snoops and realizes it has the only valid copy, so it downgrades itself to shared state and responds with data, X is placed in cache-2 in shared state, memory is also updated

# Design Issues

- Invalidate
- Find data
- Writeback/writethrough
- Cache block states
- Contention for tags
- Enforcing write serialization



# SMP Example



A: Rd X
B: Rd X
C: Rd X
A: Wr X
A: Wr X
C: Wr X
B: Rd X
A: Rd X
A: Rd Y
B: Wr X
B: Rd Y
B: Wr X
B: Wr Y

# SMP Example

| | A | B | C | |
|---|---|---|---|---|
| A: Rd X | S | | | Rd-miss req; mem responds |
| B: Rd X | S | S | | Rd-miss req; mem responds |
| C: Rd X | S | S | S | Rd-miss req; mem responds |
| A: Wr X | M | I | I | Upgrade req; no resp; others inv |
| A: Wr X | M | I | I | Cache hit |
| C: Wr X | I | I | M | Wr-miss req; A resp & inv; no wrtbk |
| B: Rd X | I | S | S | Rd-miss req; C resp; wrtbk to mem |
| A: Rd X | S | S | S | Rd-miss req; mem responds |
| A: Rd Y | S (Y) | S (X) | S (X) | Rd-miss req; X evicted; mem resp |
| B: Wr X | S (Y) | M (X) | I | Upgrade req; no resp; others inv |
| B: Rd Y | S (Y) | S (Y) | I | Rd-miss req; mem resp; X wrtbk |
| B: Wr X | S (Y) | M (X) | I | Wr-miss req; mem resp; Y evicted |
| B: Wr Y | I | M (Y) | I | Wr-miss req; mem resp; others inv; X wrtbk |

# Example Protocol

| Request | Source | Block state | Action |
|---|---|---|---|
| Read hit | Proc | Shared/excl | Read data in cache |
| Read miss | Proc | Invalid | Place read miss on bus |
| Read miss | Proc | Shared | Conflict miss: place read miss on bus |
| Read miss | Proc | Exclusive | Conflict miss: write back block, place read miss on bus |
| Write hit | Proc | Exclusive | Write data in cache |
| Write hit | Proc | Shared | Place write miss on bus |
| Write miss | Proc | Invalid | Place write miss on bus |
| Write miss | Proc | Shared | Conflict miss: place write miss on bus |
| Write miss | Proc | Exclusive | Conflict miss: write back, place write miss on bus |
| Read miss | Bus | Shared | No action; allow memory to respond |
| Read miss | Bus | Exclusive | Place block on bus; change to shared |
| Write miss | Bus | Shared | Invalidate block |
| Write miss | Bus | Exclusive | Write back block; change to invalid |

## Cache Coherence Protocols

- Directory-based: A single location (directory) keeps track of the sharing status of a block of memory

- Snooping: Every cache block is accompanied by the sharing status of that block – all cache controllers monitor the shared bus so they can update the sharing status of the block, if necessary

  - Write-invalidate: a processor gains exclusive access of a block before writing by invalidating all other copies
  - Write-update: when a processor writes, it updates other shared copies of that block

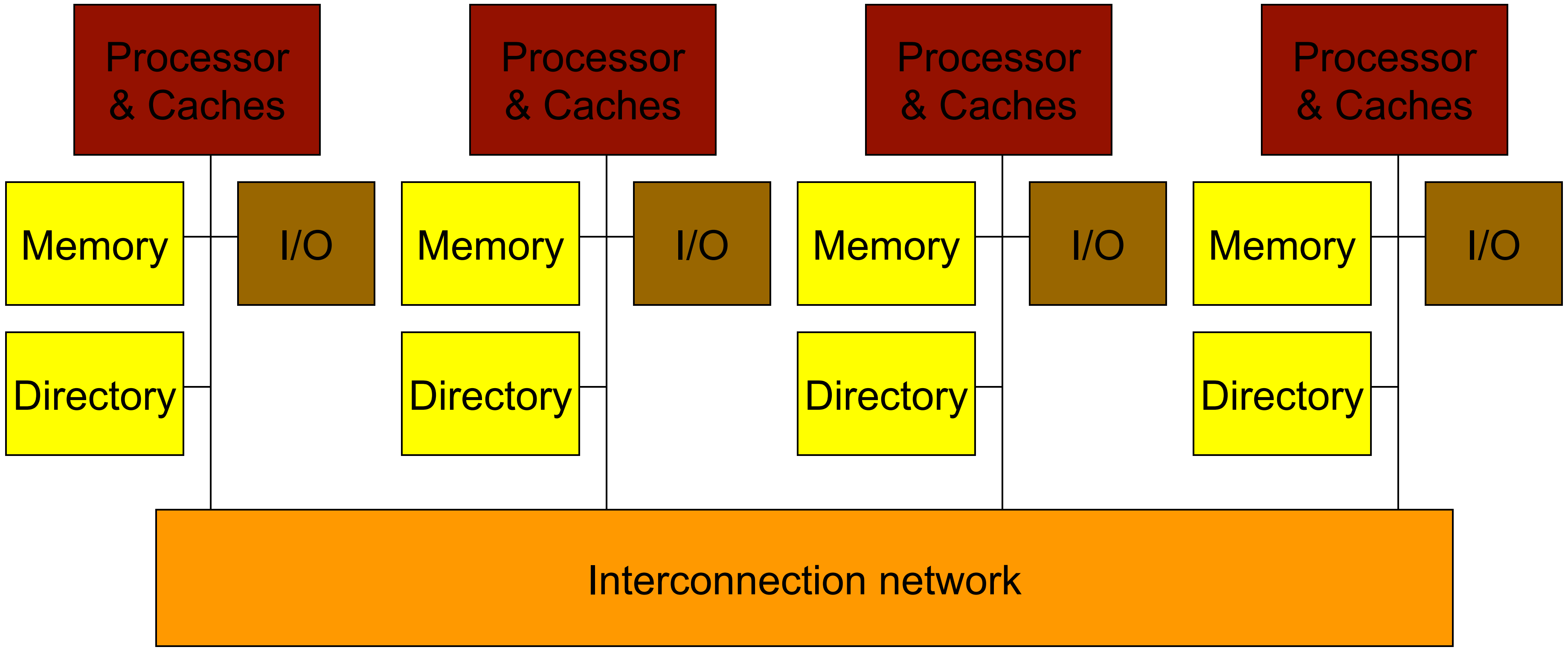
## Directory-Based Cache Coherence

- The physical memory is distributed among all processors
- The directory is also distributed along with the corresponding memory

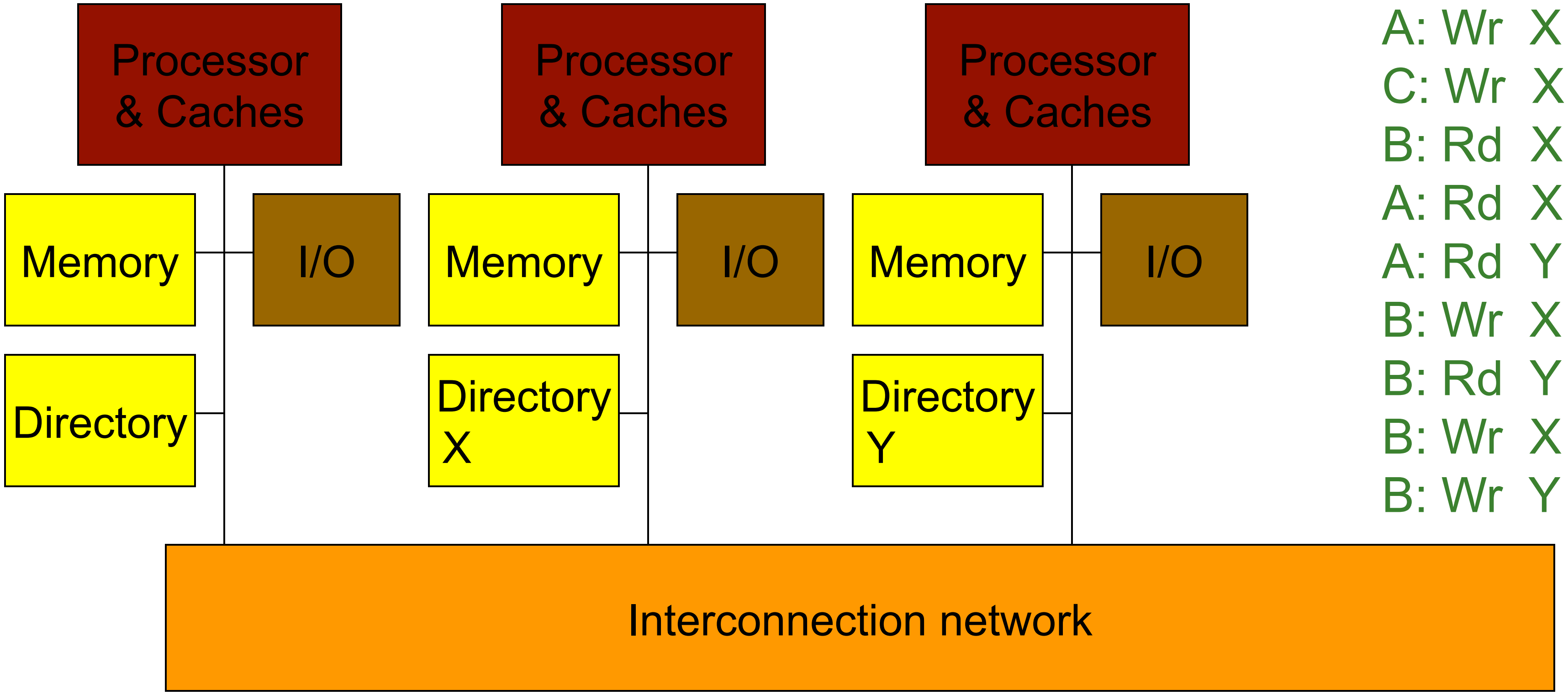
- The physical address is enough to determine the location of memory

- The (many) processing nodes are connected with a scalable interconnect (not a bus) – hence, messages are no longer broadcast, but routed from sender to receiver – since the processing nodes can no longer snoop, the directory keeps track of sharing state

# Distributed Memory Multiprocessors



# Directory-based Example



A: Rd X
B: Rd X
C: Rd X
A: Wr X
A: Wr X
C: Wr X
B: Rd X
A: Rd X
A: Rd Y
B: Wr X
B: Rd Y
B: Wr X
B: Wr Y

# Directory Example

| | A | B | C | Dir | Comments |
|---|---|---|---|---|---|
| A: Rd X | | | | | |
| B: Rd X | | | | | |
| C: Rd X | | | | | |
| A: Wr X | | | | | |
| A: Wr X | | | | | |
| C: Wr X | | | | | |
| B: Rd X | | | | | |
| A: Rd X | | | | | |
| A: Rd Y | | | | | |
| B: Wr X | | | | | |
| B: Rd Y | | | | | |
| B: Wr X | | | | | |
| B: Wr Y | | | | | |

# Directory Example

| | A | B | C | Dir | Comments |
|---|---|---|---|---|---|
| A: Rd X | S | | | S: A | Req to dir; data to A |
| B: Rd X | S | S | | S: A, B | Req to dir; data to B |
| C: Rd X | S | S | S | S: A,B,C | Req to dir; data to C |
| A: Wr X | M | I | I | M: A | Req to dir;inv to B,C;dir recv ACKs;perms to A |
| A: Wr X | M | I | I | M: A | Cache hit |
| C: Wr X | I | I | M | M: C | Req to dir;fwd to A; sends data to dir; dir to C |
| B: Rd X | I | S | S | S: B, C | Req to dir;fwd to C;data to dir;dir to B; wrtbk |
| A: Rd X | S | S | S | S:A,B,C | Req to dir; data to A |
| A: Rd Y | S(Y) | S | S | X:S: A,B,C (Y:S:A) | Req to dir; data to A |
| B: Wr X | S(Y) | M | I | X:M:B | Req to dir; inv to A,C;dir recv ACK;perms to B |
| B: Rd Y | S(Y) | S(Y) | I | X: - Y:S:A,B | Req to dir; data to B; wrtbk of X |
| B: Wr X | S(Y) | M(X) | I | X:M:B Y:S:A,B | Req to dir; data to B |
| B: Wr Y | I | M(Y) | I | X: - Y:M:B | Req to dir;inv to A;dir recv ACK; perms and data to B;wrtbk of X |

# Cache Block States

- What are the different states a block of memory can have within the directory?
- Note that we need information for each cache so that invalidate messages can be sent
- The block state is also stored in the cache for efficiency
- The directory now serves as the arbitrator: if multiple write attempts happen simultaneously, the directory determines the ordering

## Directory Actions

- If block is in uncached state:
  - Read miss: send data, make block shared
  - Write miss: send data, make block exclusive
- If block is in shared state:
  - Read miss: send data, add node to sharers list
  - Write miss: send data, invalidate sharers, make excl
- If block is in exclusive state:
  - Read miss: ask owner for data, write to memory, send data, make shared, add node to sharers list
  - Data write back: write to memory, make uncached
  - Write miss: ask owner for data, write to memory, send data, update identity of new owner, remain exclusive

## Acknowledgements

- EPFL, Onur Mutlu, Digital Design and Computer Architecture, 2020, 2023
- 计算机体系结构：量化研究方法（中文版第六版）
- UCSD CSE 240
- Washington University CSE3 78
- 国科大，胡伟武，计算机体系结构
- CMU, CS 447 Computer Architecture
- UC Berkeley, CS 152 and CS 61C
- 国科大南京学院，安学军等，计算机体系结构
- 浙江大学，计算机体系结构
- 南京大学，计算机体系结构