Memory Hierarchy Design

(Computer Architecture: Appendix B and Chapter 2)

Yanyan Shen

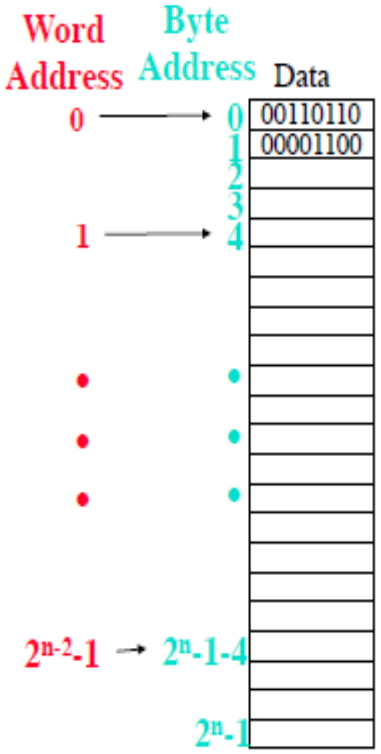
Department of Computer Science and Engineering

Roadmap

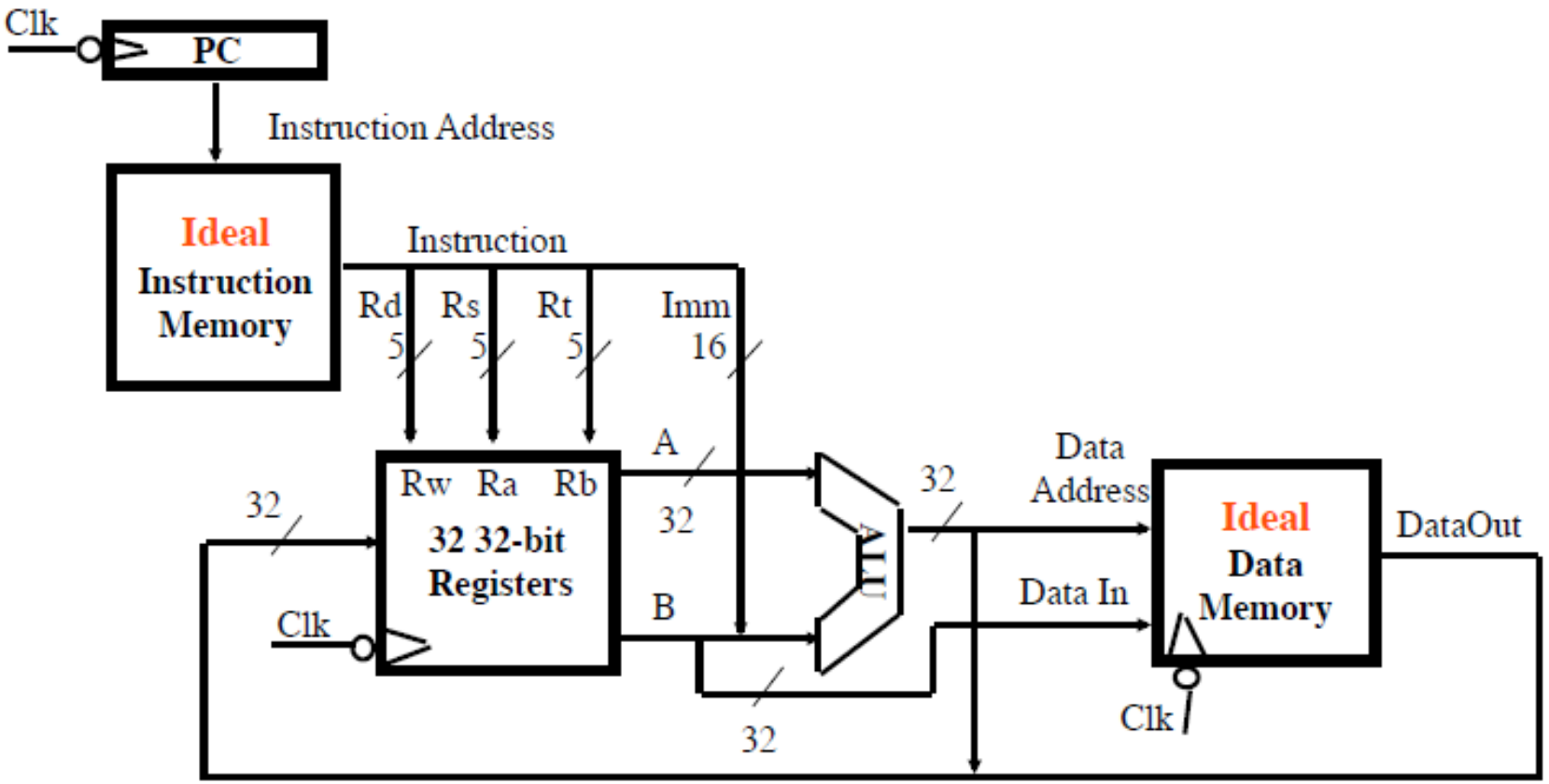
- **2.1 Cache Organization**
- Virtual Memory
- Six Basic Cache Optimizations
- Ten Advanced Optimizations of Cache Performance
- Memory Technology and Optimizations
- Virtual Memory and Protection
- Protection: Virtual Memory and Virtual Machines

Computer Memory

- Memory is a large linear array of bytes.
  - Each byte has a **unique address** (location).
  - Byte of data at address 0x100, and 0x101
- Most computers support byte (8-bit) addressing.
- Data may have to be **aligned** on word (4 bytes) or double word (8 bytes) boundary.
  - int is 4 bytes
  - double precision floating point is 8 bytes
- 32-bit vs. 64-bit addresses!
  - we will assume 32-bit for rest of course, unless otherwise stated!

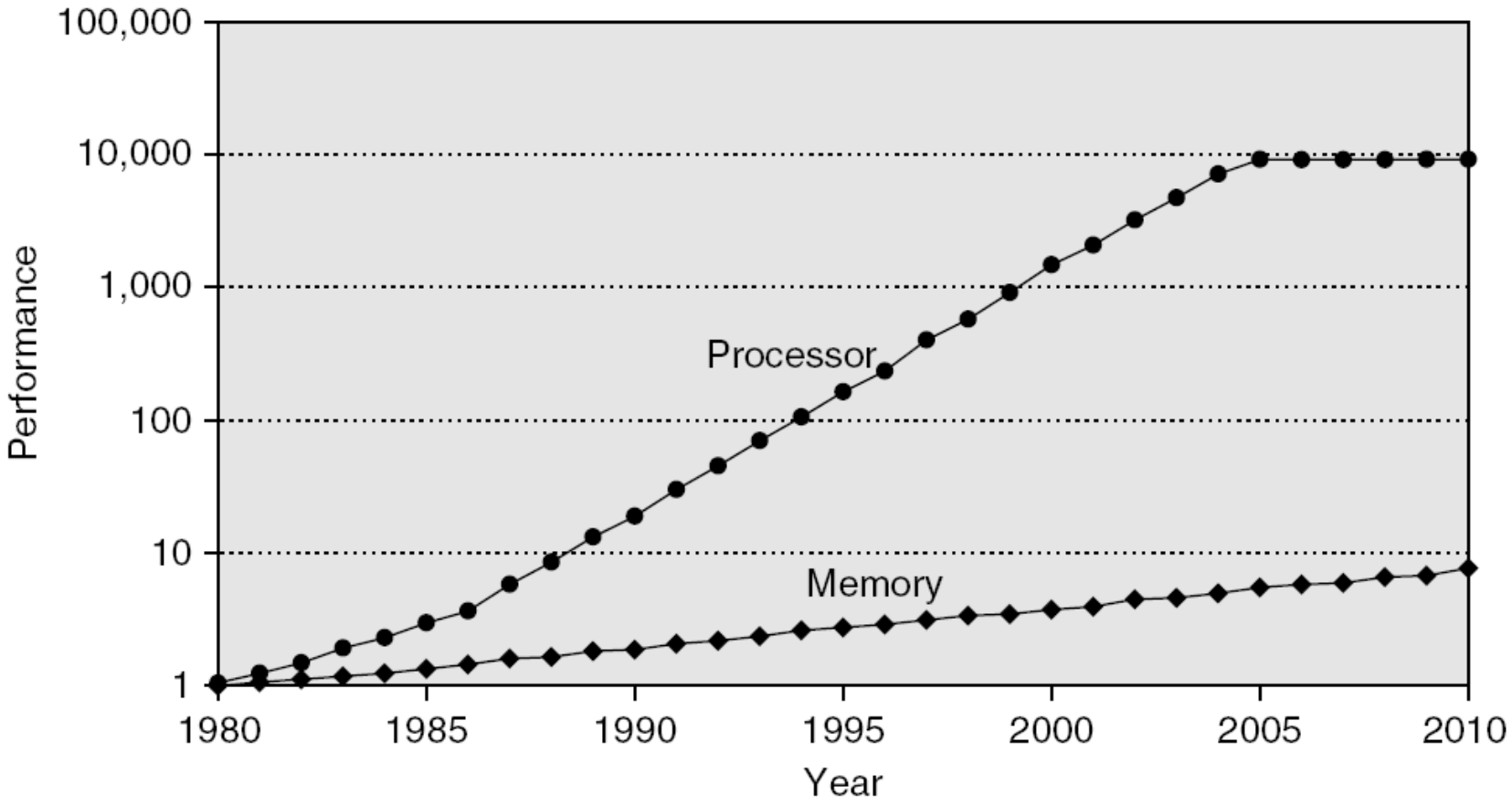


Our Naïve View of Memory



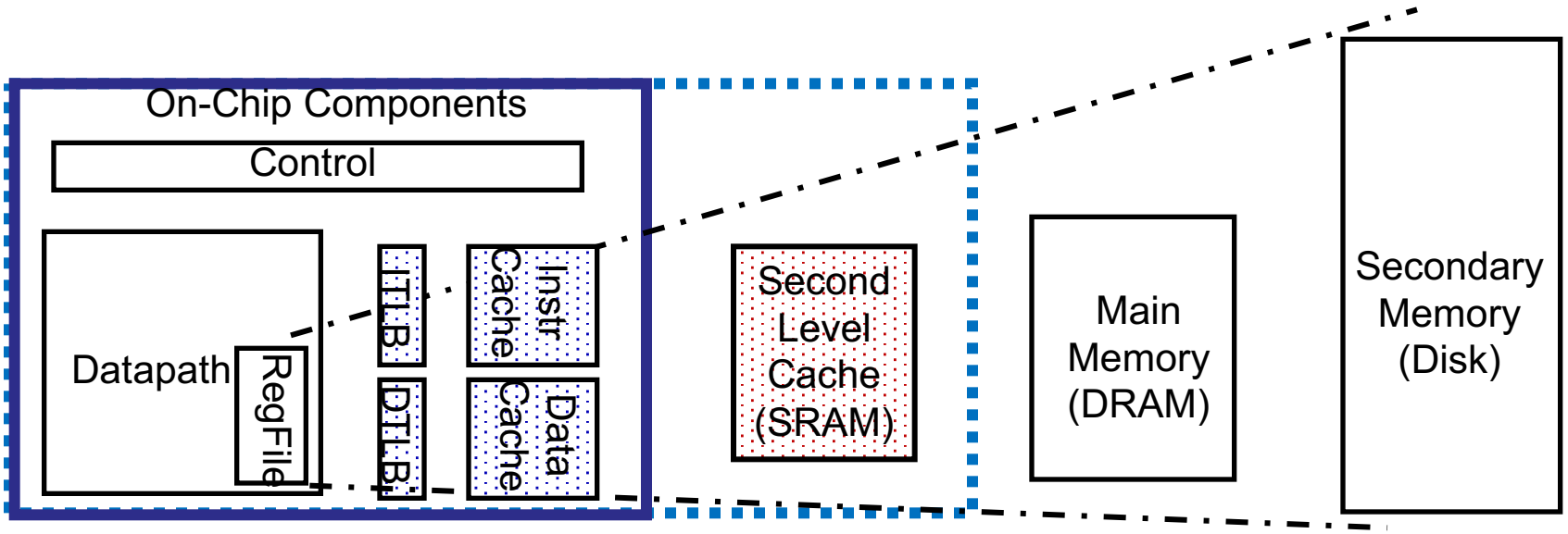
What issues do we need to worry about in implementing the memory system?

Performance Gap between Memory and Processor



A Typical Memory Hierarchy

Take advantage of the principle of locality to present the user with (1) as much memory as is available in the ***cheapest*** technology, (2) but at the speed offered by the ***fastest*** technology

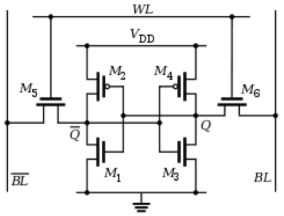


Access time (ns):	0.1	1	10	100	10,000
Size (bytes):	100	10K	M	G	T
Cost:	highest				lowest

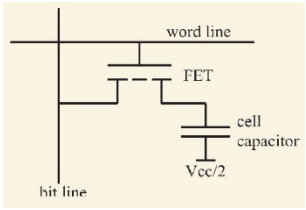
SRAM and DRAM

- Caches use *SRAM* for speed and technology compatibility
  - **Fast** (typical access times of 0.5 to 2.5 nsec)
  - **Low density** (6 transistor cells), higher power, expensive ($2000 to $5000 per GB in 2008)
  - **Static**: content will last "forever" (as long as power is left on)

- Main memory uses *DRAM* for size (density)
  - **Slower** (typical access times of 50 to 70 nsec)
  - **High density** (1 transistor cells), lower power, cheaper ($20 to $75 per GB in 2008)
  - **Dynamic**: needs to be "refreshed" regularly (~ every 8 ms)
    - consumes1% to 2% of the active cycles of the DRAM
  - Addresses divided into 2 halves (row and column)
    - *RAS* or *Row Access Strobe* triggering the row decoder
    - *CAS* or *Column Access Strobe* triggering the column selector



SRAM: 6 transistors



DRAM: 1 transistor

Locality

Principle of Locality:

Programs tend to reuse data and instructions near those they have used recently, or those were recently referenced themselves.

Temporal locality

- Recently referenced items are likely to be referenced in the near future.

Spatial locality

- Items with nearby addresses tend to be referenced close together in time.

The Memory Hierarchy: How Does it Work?

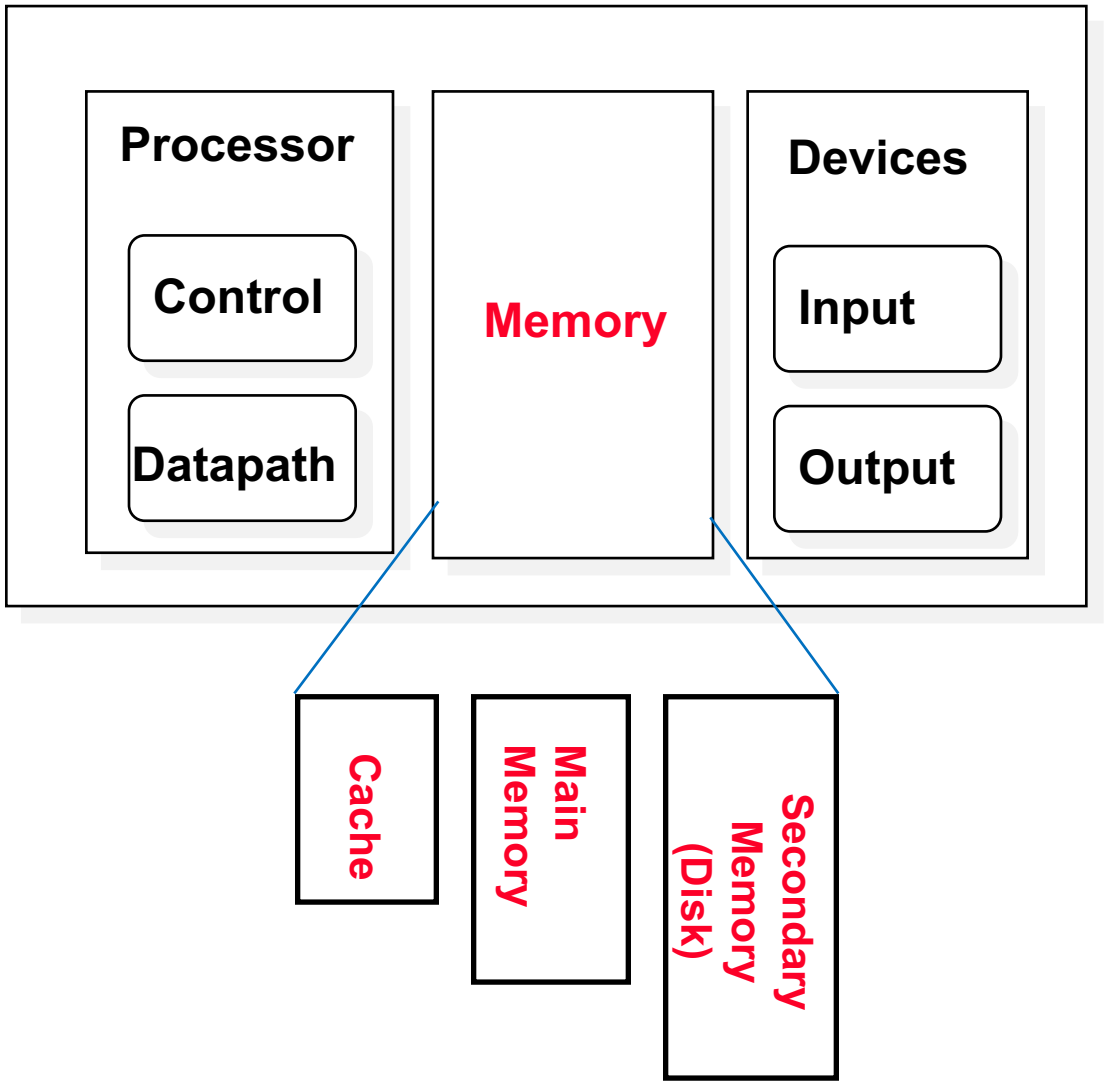
- Temporal Locality (locality in time)
  - If a memory location is referenced then it will tend to be referenced again soon

  ⇒ Keep most recently accessed data items closer to the processor

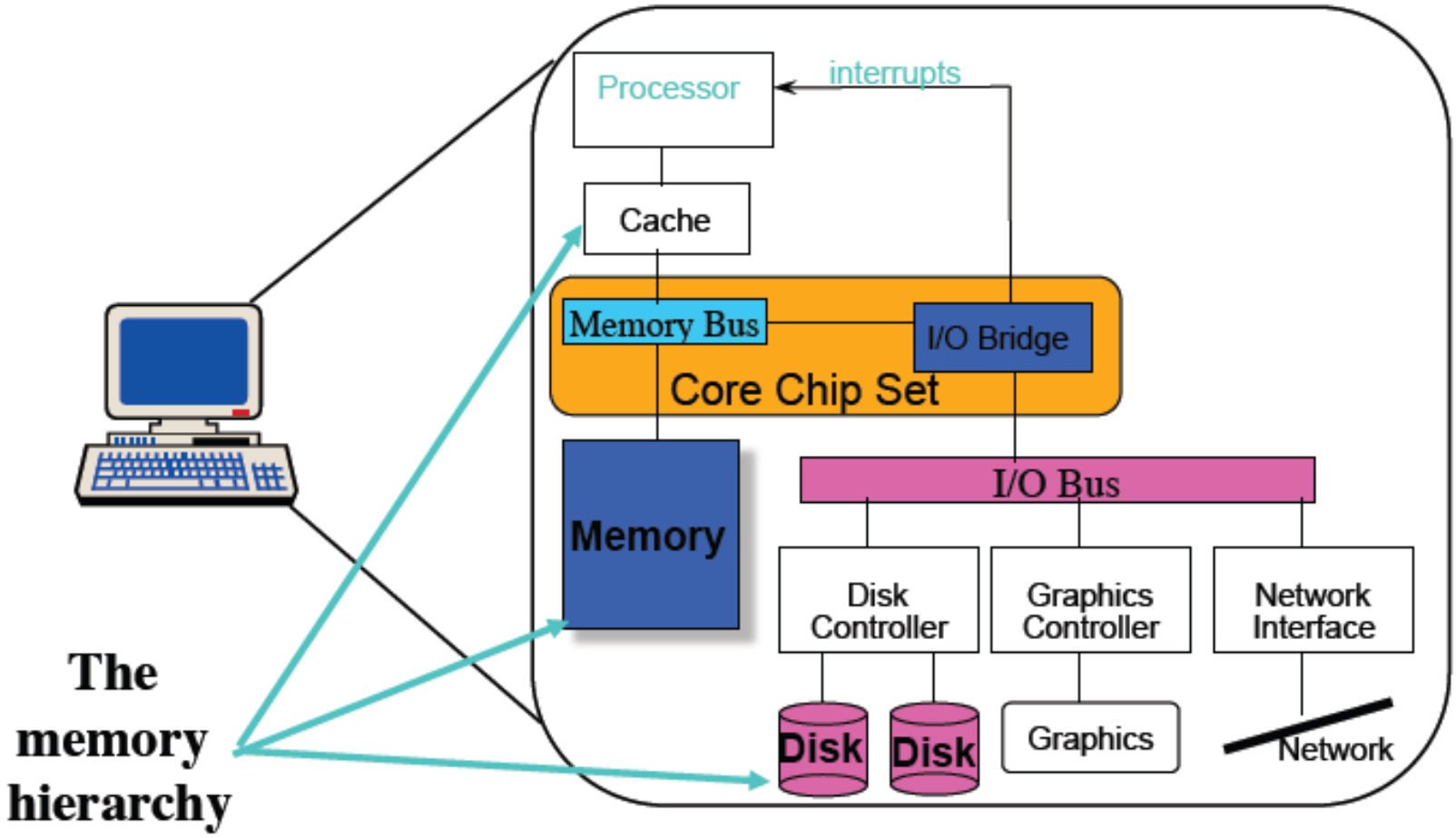
- Spatial Locality (locality in space)
  - If a memory location is referenced, the locations with nearby addresses will tend to be referenced soon

  ⇒ Move blocks consisting of contiguous words closer to the processor

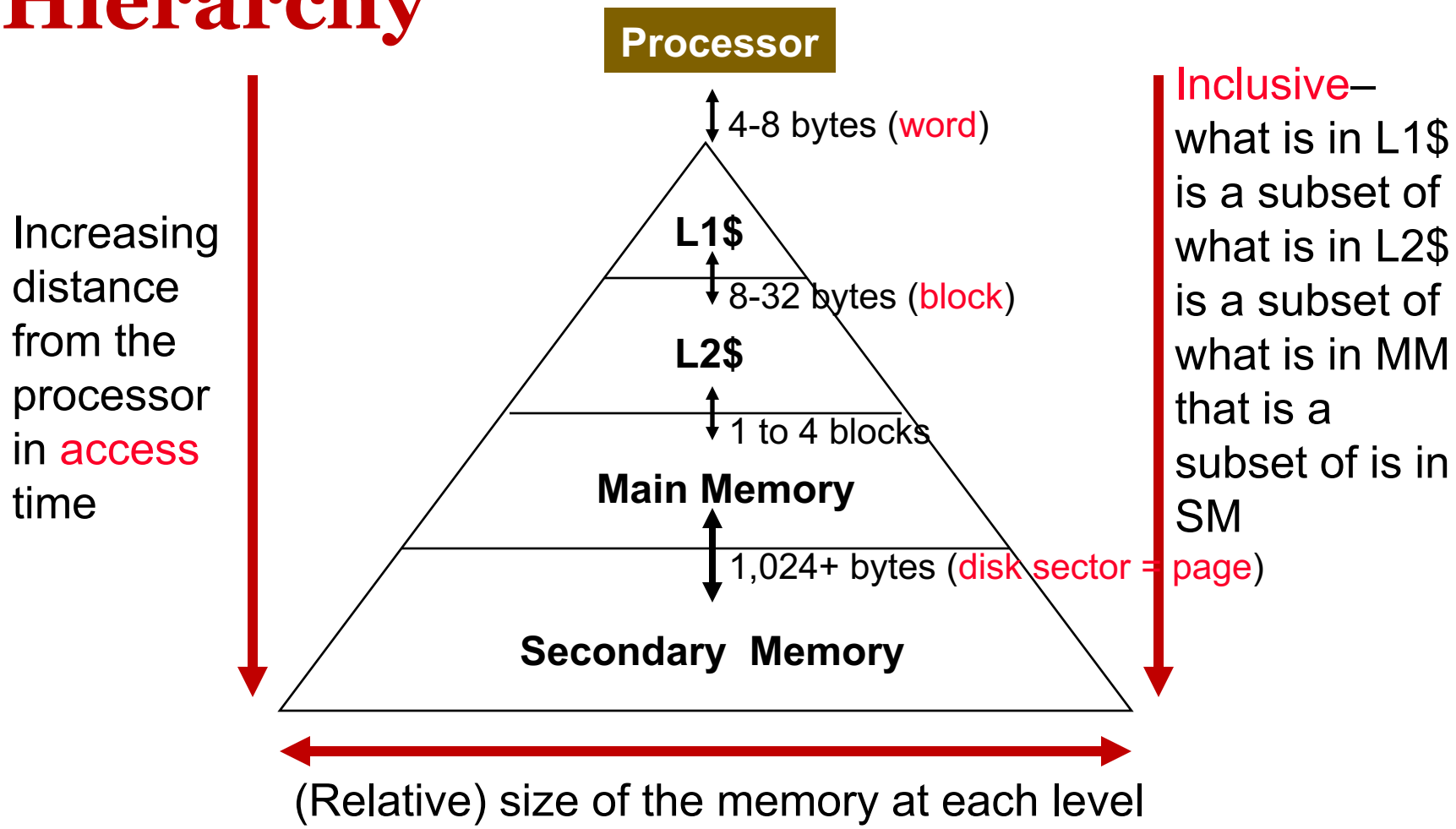
Overview


Processor
Control
Datapath
Memory
Devices
Input
Output
Cache
Main Memory
Secondary Memory (Disk)

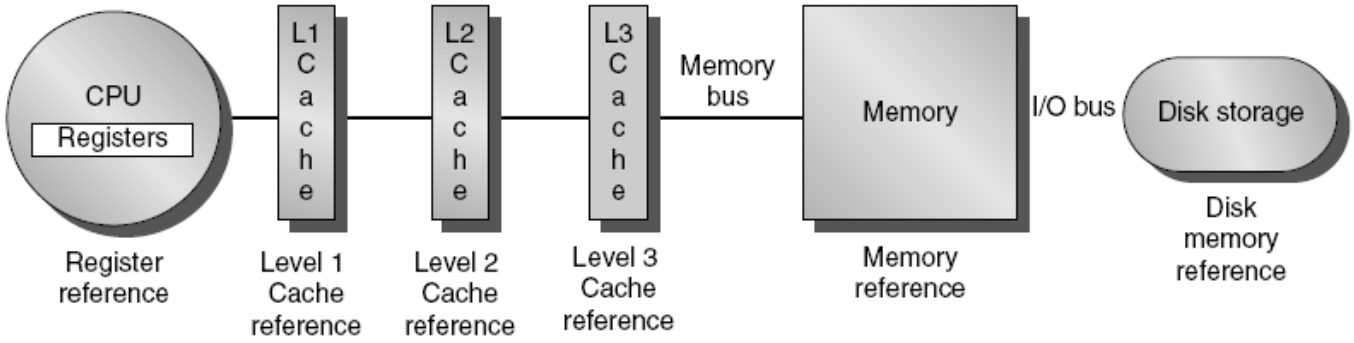
Real Organization of Hierarchy



Characteristics of the Memory Hierarchy

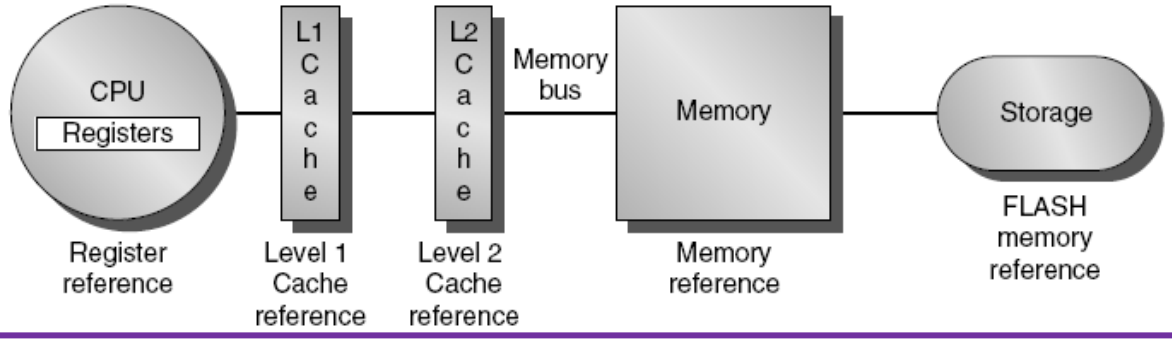


Example



Size:	1000 bytes	64 KB	256 KB	2–4 MB	4–16 GB	4–16 TB
Speed:	300 ps	1 ns	3–10 ns	10–20 ns	50–100 ns	5–10 ms

(a) Memory hierarchy for server



Size:	500 bytes	64 KB	256 KB	256–512 MB	4–8 GB
Speed:	500 ps	2 ns	10–20 ns	50–100 ns	25–50 us

(b) Memory hierarchy for a personal mobile device

Terminology

- **Block** (or line): the minimum unit of information that is present (or not) in a cache
- **Hit Rate**: the fraction of memory accesses found in a level of the memory hierarchy
  - **Hit Time**: Time to access that level which consists of

    (1) Time to access the block + (2) Time to determine hit/miss
- **Miss Rate**: the fraction of memory accesses *not* found in a level of the memory hierarchy ⇒ 1 - (Hit Rate)
  - **Miss Penalty**: Time to replace a block in that level with the corresponding block from a lower level which consists of

    (1) Time to access the block in the lower level + (2) Time to transmit that block to the level that experienced the miss + (3) Time to insert the block in that level + (4) Time to pass the block to the requestor

Hit Time << Miss Penalty


鲁大师 www.ludashi.com
问题反馈 论坛
硬件检测
温度监测
性能测试
节能降温
驱动管理
电脑优化
新机推荐
电脑维修
鲁大师
360旗下安全产品
最后一次检测电脑硬件配置情况是在：2014年9月29日
重新检测
生成报告
保存截屏
电脑概览
硬件健康
处理器信息
主板信息
内存信息
硬盘信息
显卡信息
显示器信息
光驱信息
网卡信息
声卡信息
其他硬件
功耗估算
扫描完毕，没有发现需要安装的驱动！
功能： √ 独立显卡 √ 无线上网 X 摄像头 √ 光盘刻录 √ 蓝牙
intel
处理器品牌
处理器信息
当前温度： 42 ℃
复制信息
处理器 英特尔 Core i7-4770 @ 3.40GHz 四核
速度 3.40 GHz (100 MHz x 34.0)
处理器数量 核心数: 4 / 线程数: 8
插槽/插座 CPU 1
一级数据缓存 4 x 32 KB, 8-Way, 64 byte lines
一级代码缓存 4 x 32 KB, 8-Way, 64 byte lines
二级缓存 4 x 256 KB, 8-Way, 64 byte lines
三级缓存 8 MB, 16-Way, 64 byte lines
特征 MMX, SSE, SSE2, SSE3, SSSE3, SSE4.1, SSE4.2, HTT, EM64T, EI...
主程序版本: 3.70.14.1050 主设备库版本: 1.0.14.1050 已经是最新版本

How is the Hierarchy Managed?

- registers ↔ memory
  - by compiler
- cache ↔ main memory
  - by the cache controller hardware
- main memory ↔ disks
  - by the operating system (virtual memory)
  - physical address mapping assisted by the hardware (TLB)

Four Memory Hierarchy Questions

Q1 (block placement): where can a block be placed in cache?

Q2 (block identification): how is a block found if it is in cache?

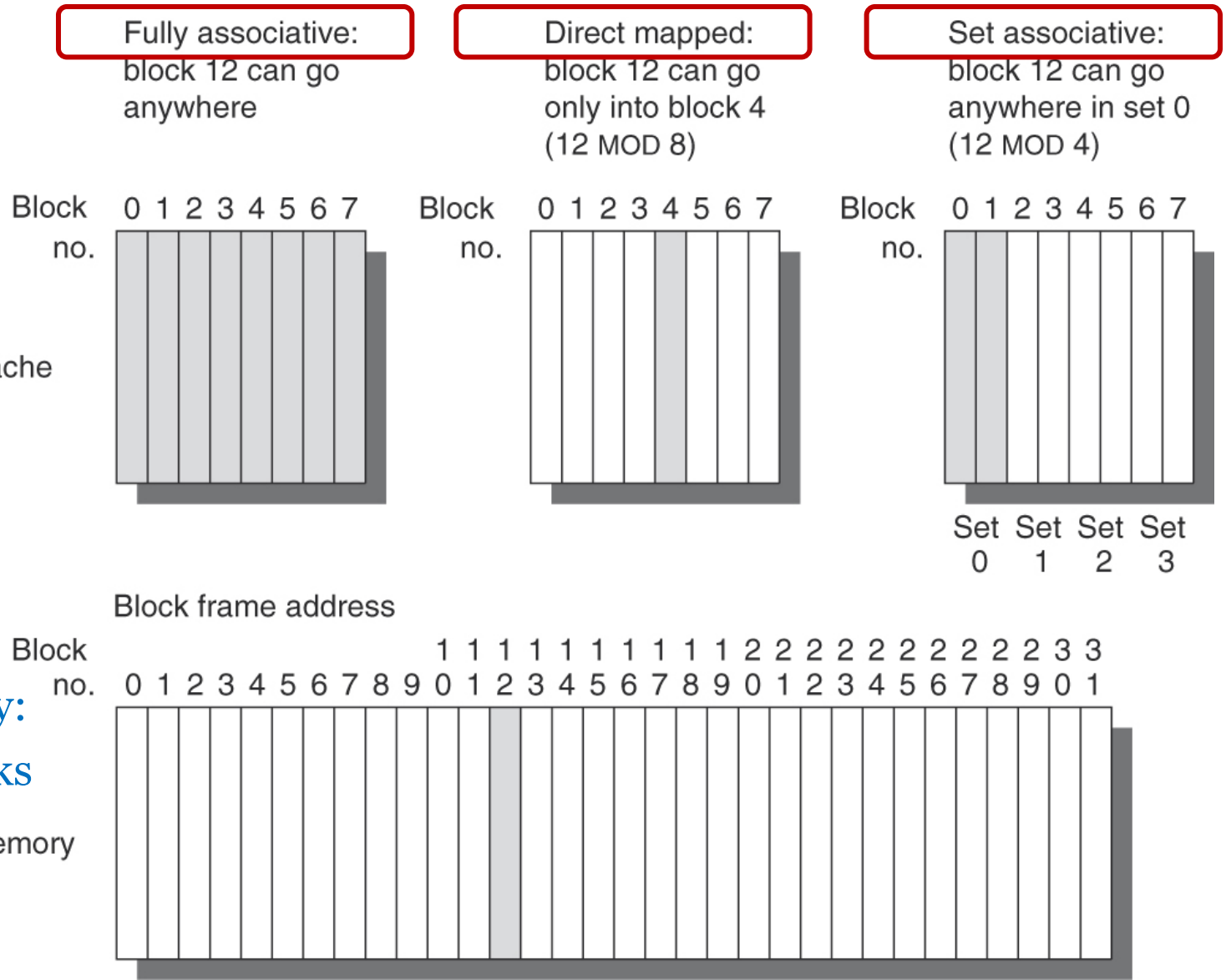
Q3 (block replacement): which block should be replaced on a miss?

Q4 (write strategy): what happens on a write?

Q1: Block Placement

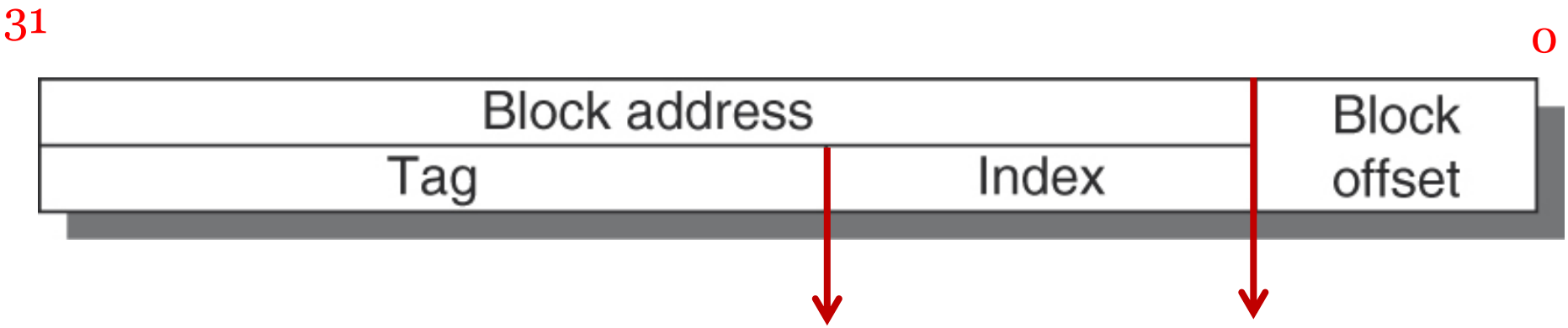
Cache:
8 blocks

Memory:
32 blocks



Q2: Block Identification

- Caches have ***an address tag*** on each block frame that gives the block address
  - It is checked to see if it matches the block address from the processor
- Each cache block has ***a valid bit***
  - To indicate if the entry has a valid address



Two questions to answer (in hardware):

- ***Question 2***: How do we know if a data item is in the cache?

- ***Question 1***: If it is, how do we find it?

Direct mapped

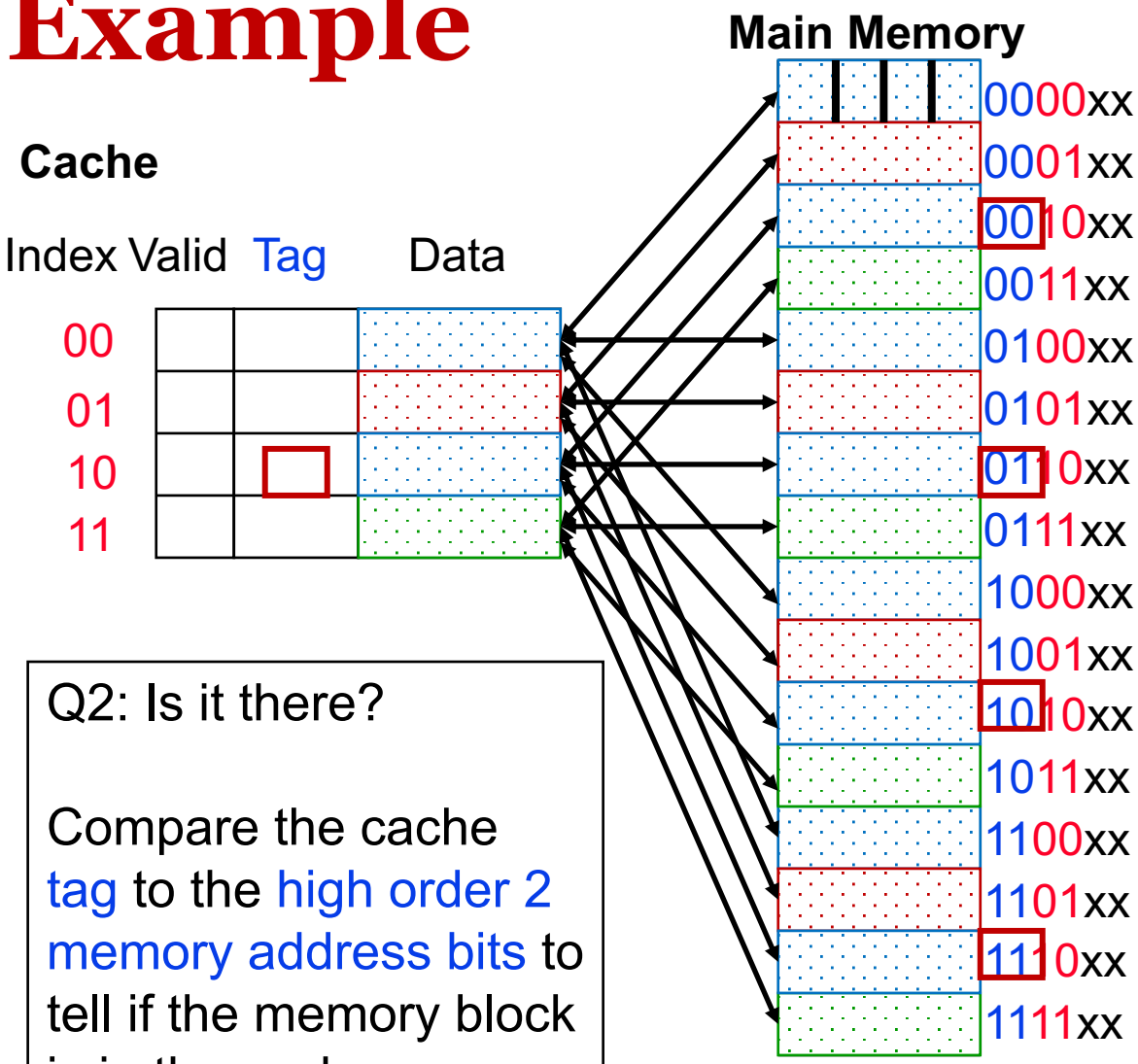
- Each memory block is mapped to exactly one block in the cache
  - lots of blocks must share blocks in the cache

- Address mapping (***to answer Q1***):

(block address) modulo (# of blocks in the cache)

- Have a tag associated with each cache block that contains the address information (the upper portion of the address) required to identify the block (***to answer Q2***)

Caching: A Simple First Example



Byte addressable!

- One-word blocks
- Two low order bits define the byte in the word (32b words)

Q1: How do we find it?

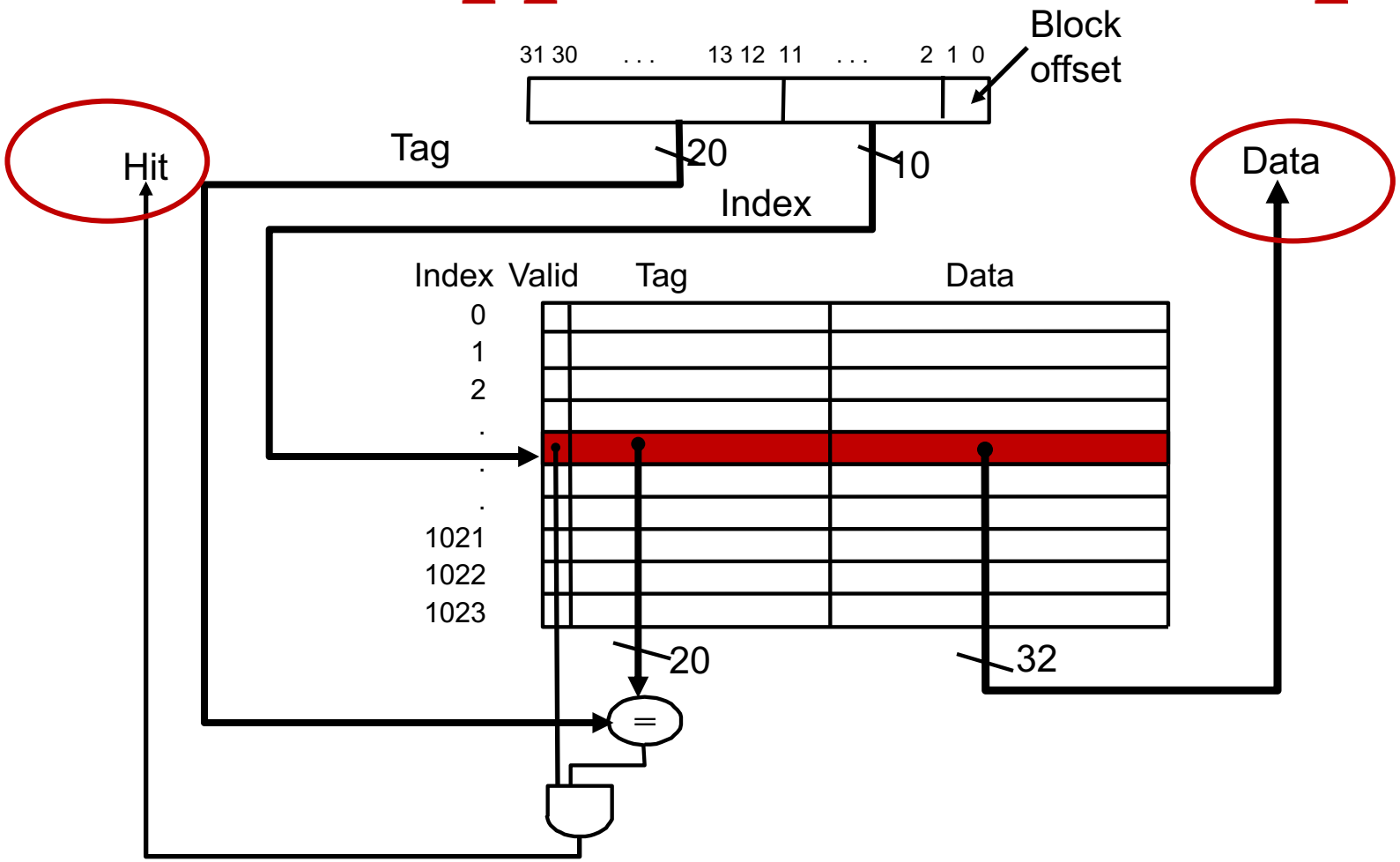
Use next 2 low order memory address bits – the index – to determine which cache block (i.e., modulo the number of blocks in the cache)

Q2: Is it there?

Compare the cache tag to the high order 2 memory address bits to tell if the memory block is in the cache

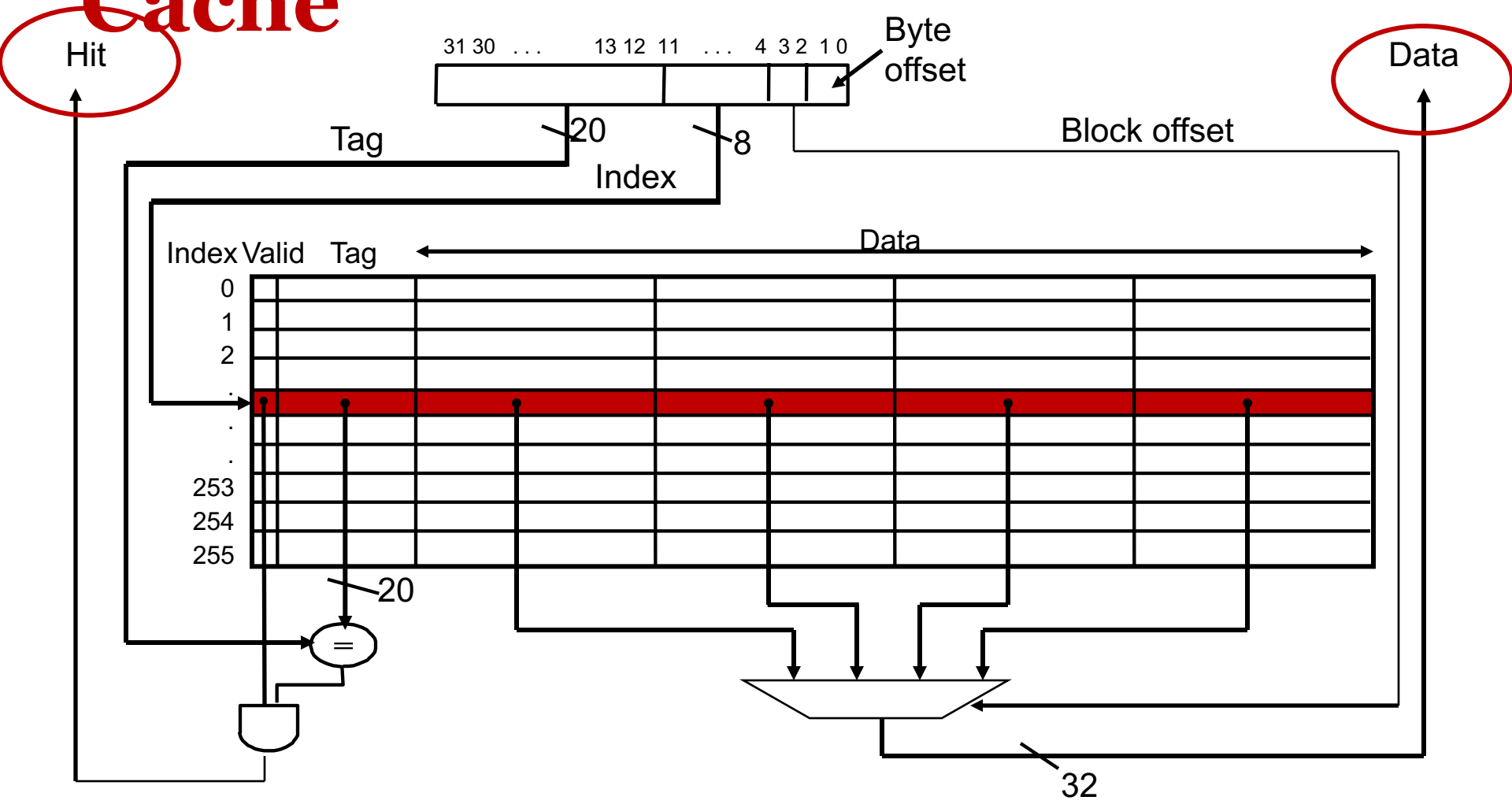
(block address) modulo (# of blocks in the cache)

Direct Mapped Cache Example



One word blocks, cache size = 1K words (or 4KB)

Multiword Block Direct Mapped Cache



Four words/block, cache size = 1K words

Set Associative Mapped

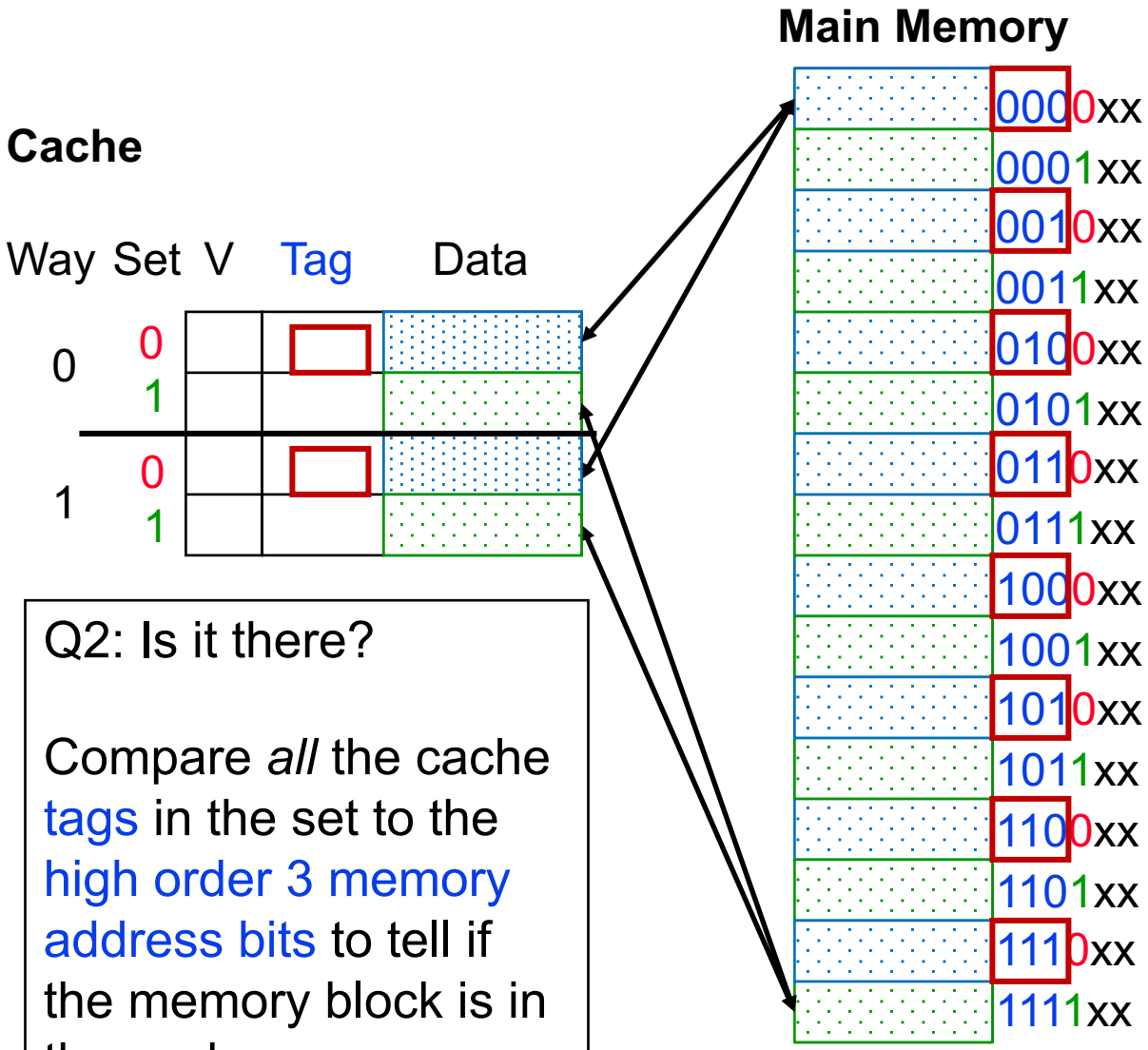
Allow more flexible block placement

- In a direct mapped cache a memory block maps to exactly one cache block
- At the other extreme, could allow a memory block to be mapped to *any* cache block – fully associative cache

- A compromise is to divide the cache into sets each of which consists of n "ways" (n-way set associative). A memory block maps to a unique set (specified by the index field) and can be placed in any way of that set (so there are *n* choices)

  (block address) modulo (# sets in the cache)

Set Associative Cache Example



- One-word blocks
- Two low order bits define the byte in the word (32b words)

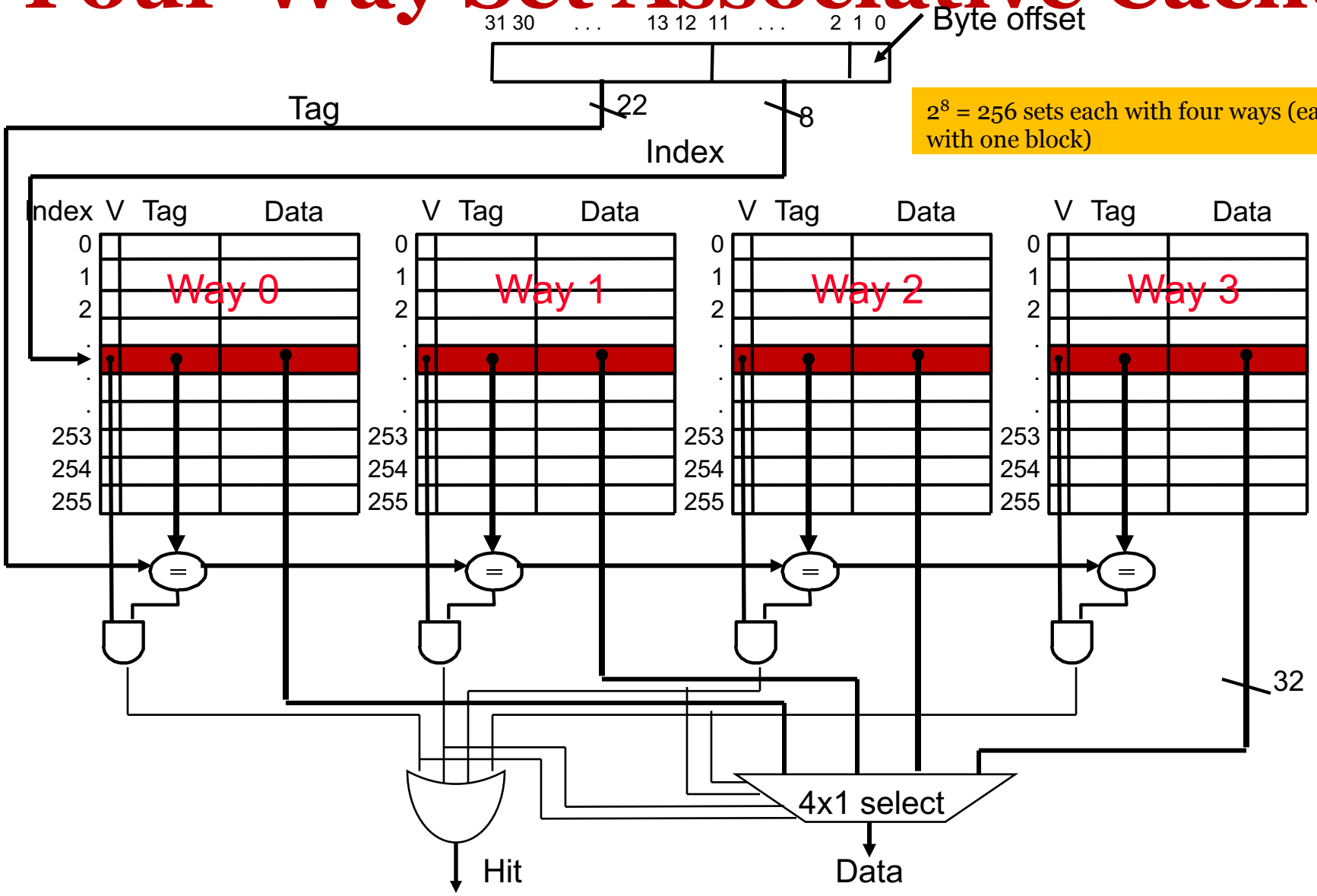
Q1: How do we find it?

Use next 1 low order memory address bit to determine which cache set (i.e., modulo the number of sets in the cache)

Q2: Is it there?

Compare *all* the cache tags in the set to the high order 3 memory address bits to tell if the memory block is in the cache

Four-Way Set Associative Cache



2^8 = 256 sets each with four ways (each with one block)

Q3: Block Replacement

When a miss occurs, the cache controller must select a block to be replaced with the desired data

	Associativity								
	Two-way			Four-way			Eight-way		
Size	LRU	Random	FIFO	LRU	Random	FIFO	LRU	Random	FIFO
16 KB	114.1	117.3	115.5	111.7	115.1	113.3	109.0	111.8	110.4
64 KB	103.4	104.3	103.9	102.4	102.3	103.1	99.7	100.5	100.3
256 KB	92.2	92.1	92.5	92.1	92.1	92.5	92.1	92.1	92.5

Data cache misses per 1000 instructions comparing **LRU**, **Random** and **FIFO**

Q4: Write Strategy

Write data to cache. Two write policies.

Write-through

The information is written to both the block in the cache and the block in the lower-level memory

Write-back

The information is written only to the block in the cache. The modified cache block is written to main memory **only when it is replaced**

Handling Cache Hits

- Read hits (I$ and D$)
  - this is what we want!

- Write hits (D$ only)
  - require the cache and memory to be consistent
    - always write the data into both the cache block and the next level in the memory hierarchy (write-through)
    - writes run at the speed of the next level in the memory hierarchy – so slow! – or can use a write buffer and stall only if the write buffer is full
  - allow cache and memory to be inconsistent
    - write the data only into the cache block (write-back the cache block to the next level in the memory hierarchy when that cache block is "evicted")
    - need a dirty bit for each data cache block to tell if it needs to be written back to memory when it is evicted – can use a write buffer to help "buffer" write-backs of dirty blocks

Handling Cache Misses (Single Word Blocks)

- Read misses (I$ and D$)
  - stall the pipeline, fetch the block from the next level in the memory hierarchy, install it in the cache and send the requested word to the processor, then let the pipeline resume
- Write misses (D$ only)
  1. Write allocate – stall the pipeline, fetch the block from next level in the memory hierarchy, install it in the cache (which may involve having to evict a dirty block if using a write-back cache), write the word from the processor to the cache, then let the pipeline resume, or;
  2. No-write allocate – skip the cache write (but must invalidate that cache block since it will now hold stale data) and just write the word to the write buffer (and eventually to the next memory level), no need to stall if the write buffer isn't full

Handling Cache Misses (Multiword Blocks)

- Read misses (I$ and D$)
  - Processed the same as for single word blocks – a miss returns the entire block from memory
  - Miss penalty grows as block size grows
    - Early restart – processor resumes execution as soon as the requested word of the block is returned
    - Requested word first – requested word is transferred from the memory to the cache (and processor) first
  - Nonblocking cache – allows the processor to continue to access the cache while the cache is handling an earlier miss
- Write misses (D$)
  - If using write allocate must *first* fetch the block from memory and then write the word to the block

Memory Hierarchy Design

Memory hierarchy design becomes more **crucial** with recent multi-core processors

Aggregate peak bandwidth grows with # cores:

- Intel Core i7 can generate two references per core per clock
- Four cores and 3.2 GHz clock
  - 25.6 billion 64-bit data references/second +
  - 12.8 billion 128-bit instruction references
  - = 409.6 GB/s!
- DRAM bandwidth is only 6% of this (25 GB/s)

Measuring Memory Hierarchy Performance

Average Memory Access Time

*= Hit Time + Miss Rate * Miss Penalty*

Hit Ratio?

Hit Time: The time to hit in the cache

Miss Penalty: The time to get data from the main memory

Miss Rate: Fraction of memory accesses not found in the cache

Then, Processor Performance?

Previously

CPU Time

*= Instruction Counts * CPI * Cycle Time*

*= CPU Execution Clock Cycles * Cycle Time*

Now

CPU Time

*= (CPU Execution Clock Cycles + Memory Stall Clock Cycles) * Cycle Time*

Thus, cache behavior have a great impact on processor performance!!

CPU Time as function of CPI

$$CPU\ Time$$

$$= (CPU\ Execution\ Clock\ Cycles + Memory\ Stall\ Clock\ Cycles) * Cycle\ Time$$

$$= (IC * CPI + IC * \frac{Memory\ Accesses}{Instruction} * Miss\ Rate * Miss\ Penalty) * Cycle\ Time$$

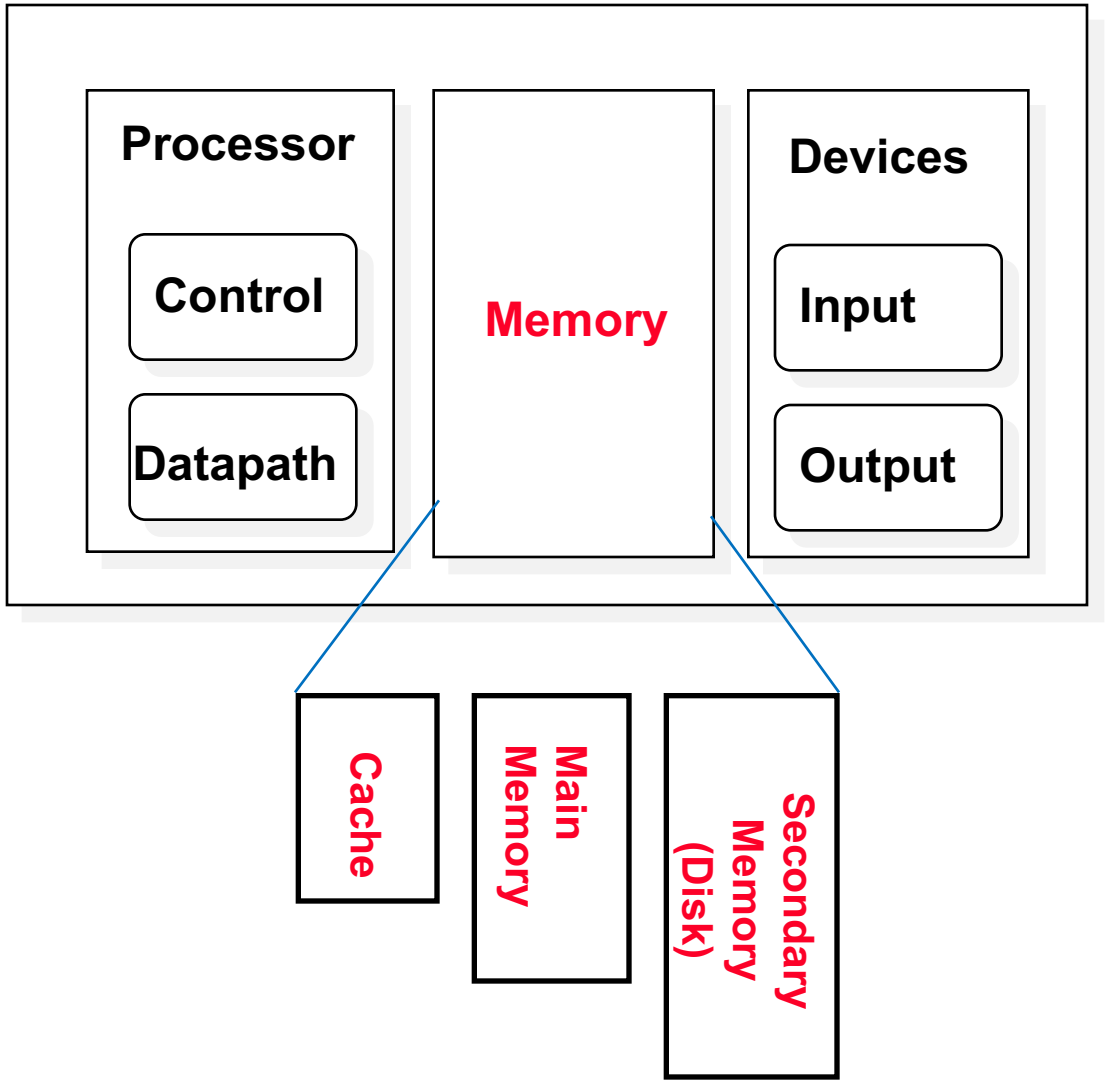
$$= IC * (CPI + \frac{Memory\ Accesses}{Instruction} * Miss\ Rate * Miss\ Penalty) * Cycle\ Time$$

The lower CPI, the greater impact of the misses

Roadmap

- Cache Organization
- **Virtual Memory**
- Six Basic Cache Optimizations
- Ten Advanced Optimizations of Cache Performance
- Memory Technology and Optimizations
- Virtual Memory and Protection
- Protection: Virtual Memory and Virtual Machines

Review: Major Components of a Computer

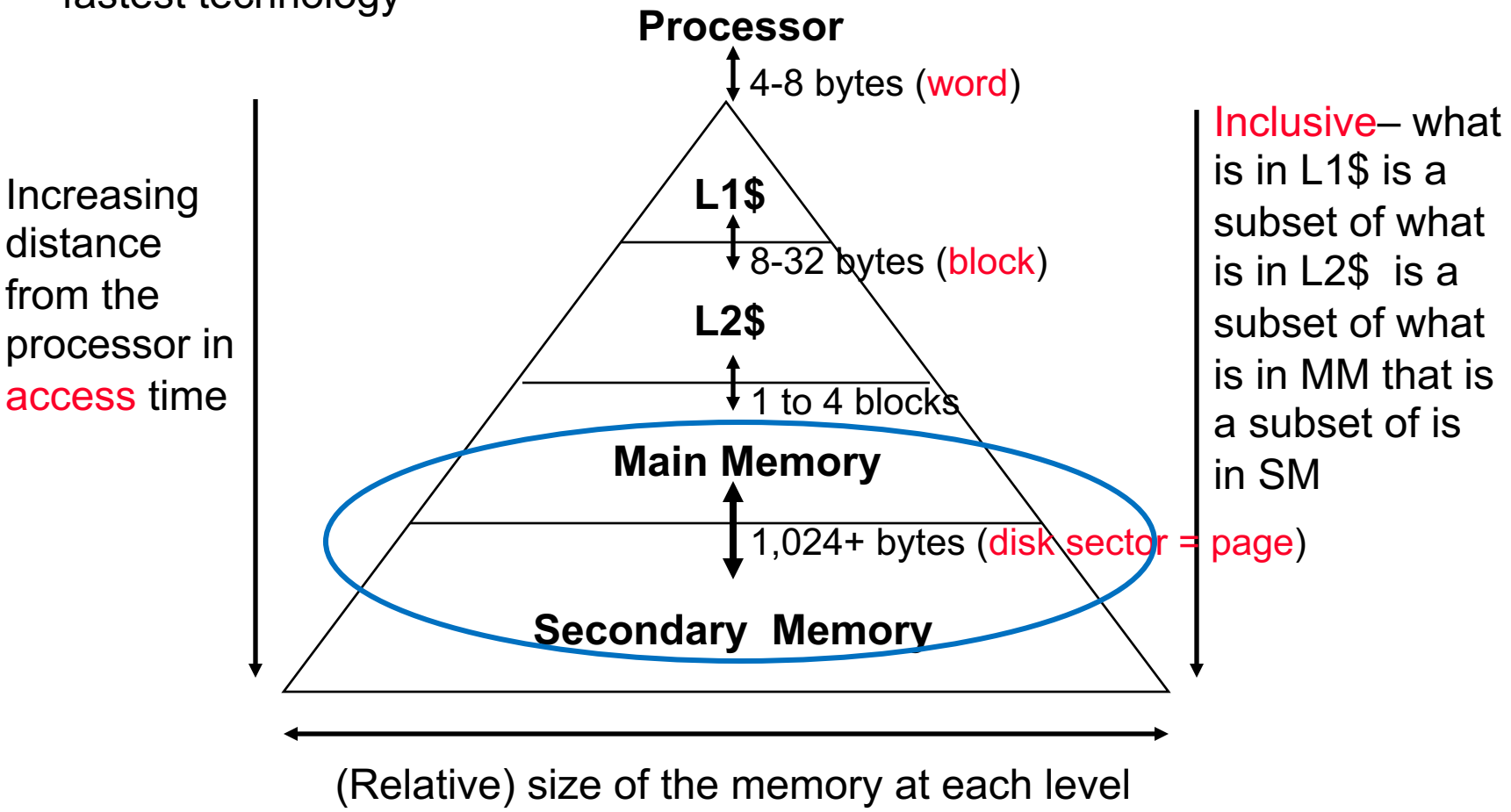


How is the Hierarchy Managed?

- registers ↔ memory
  - by compiler (programmer?)
- cache ↔ main memory
  - by the cache controller hardware
- main memory ↔ disks
  - by the operating system (virtual memory)
  - physical address mapping assisted by the hardware (TLB)

Review: The Memory Hierarchy

Take advantage of the principle of locality to present the user with as much memory as is available in the cheapest technology at the speed offered by the fastest technology



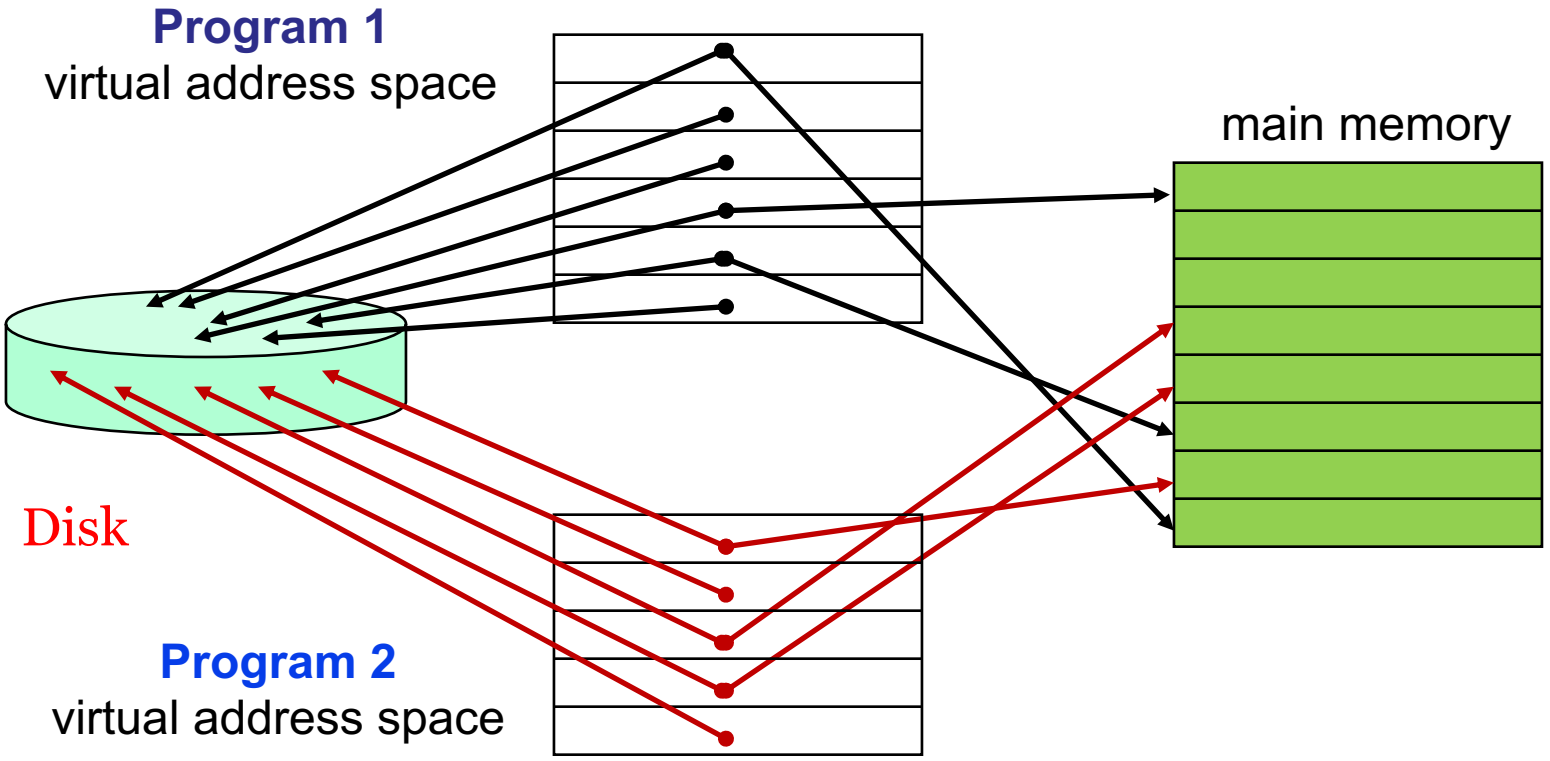
Virtual Memory - Why and How?

- Why virtual memory?
  - (1) Provides the ability to easily run programs larger than the size of physical memory
  - (2) Simplifies loading a program for execution by providing for code relocation (i.e., the code can be loaded anywhere in main memory)
  - (3) Allows efficient and safe sharing of memory among multiple programs
- What makes it work? – again the ***Principle of Locality***
  - A program is likely to access a relatively small portion of its address space during any period of time

- Each program is compiled into its own address space – a "virtual" address space
  - During run-time each virtual address must be translated to a physical address (an address in main memory)

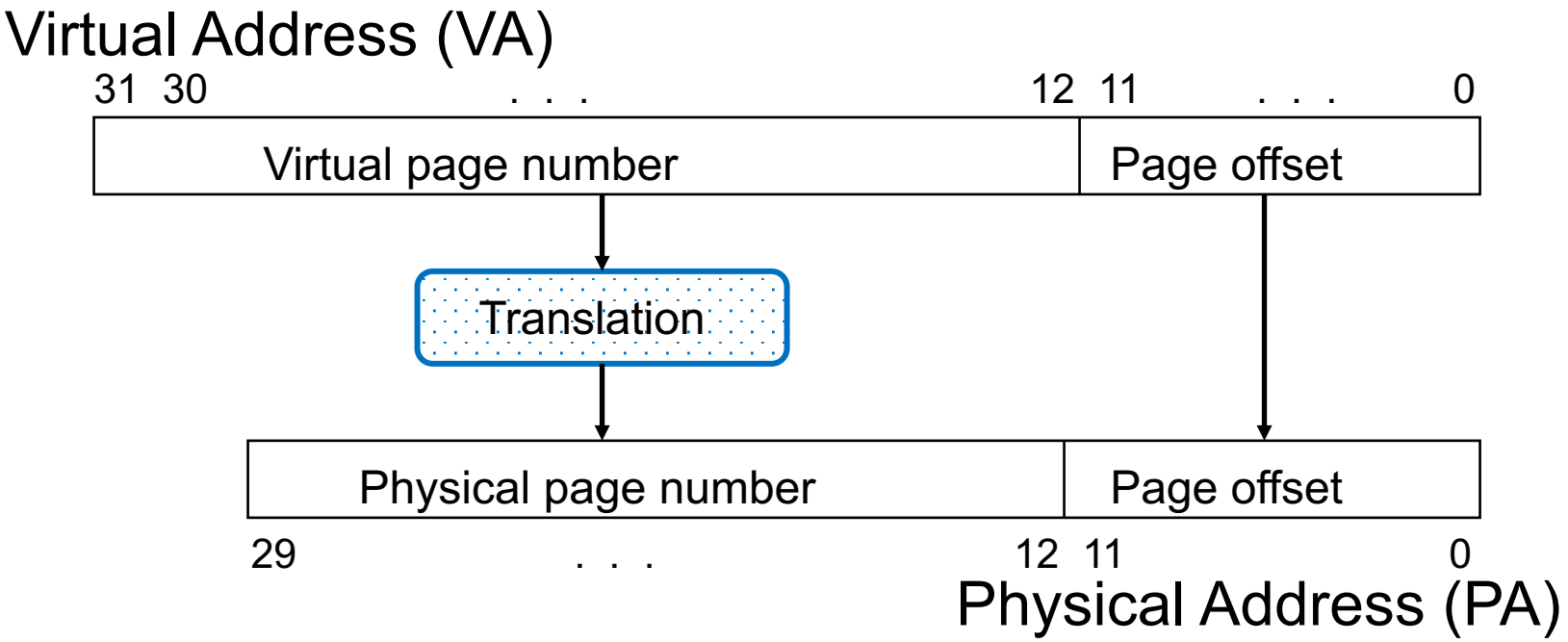
Two Programs Sharing Physical Memory

- A program's address space is divided into pages (all one fixed size) or segments (variable sizes)
- Each process has a page table containing mapping from logical address to physical address



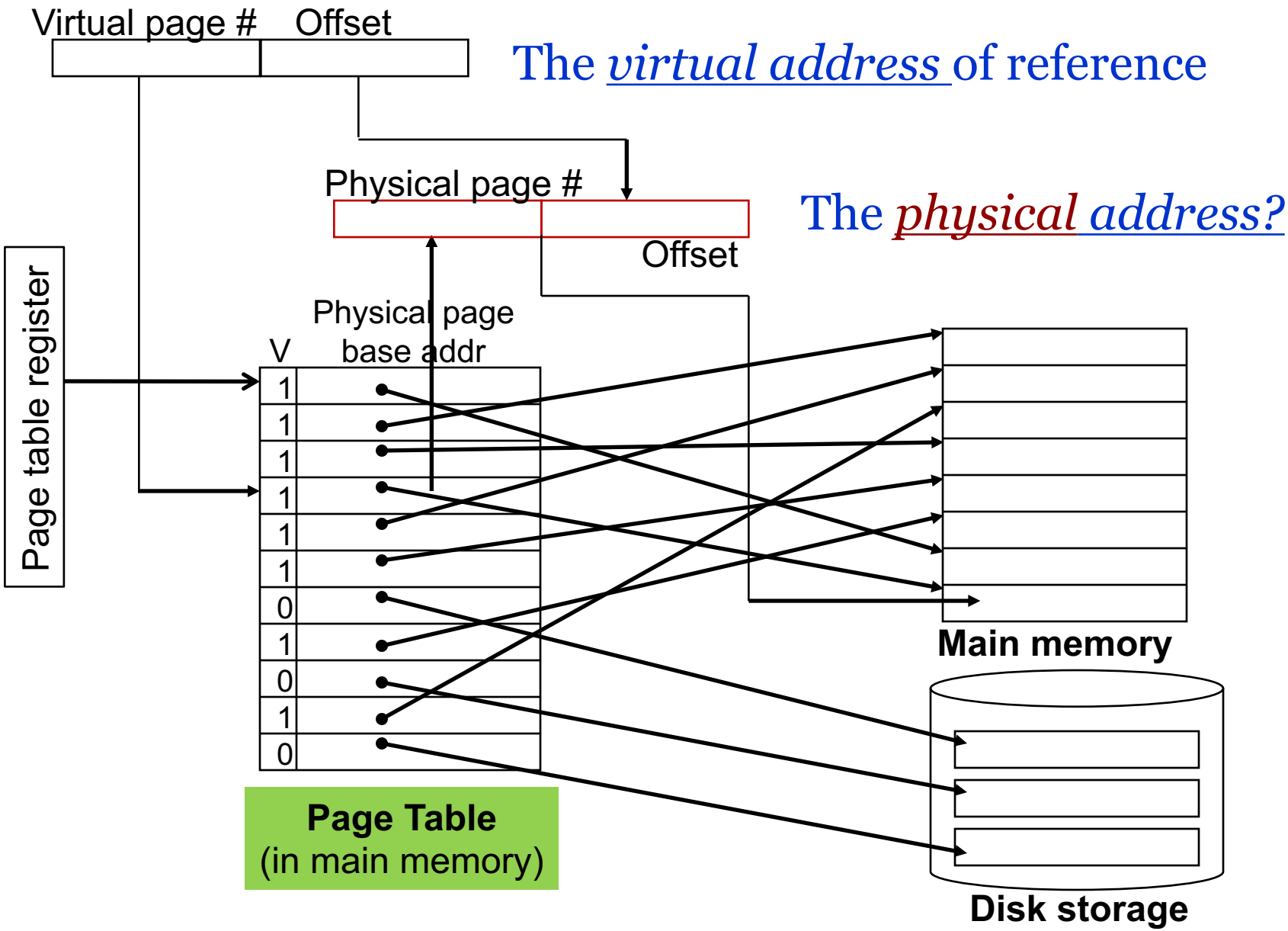
Address Translation

- A virtual address is translated to a physical address by a *combination of hardware and software*



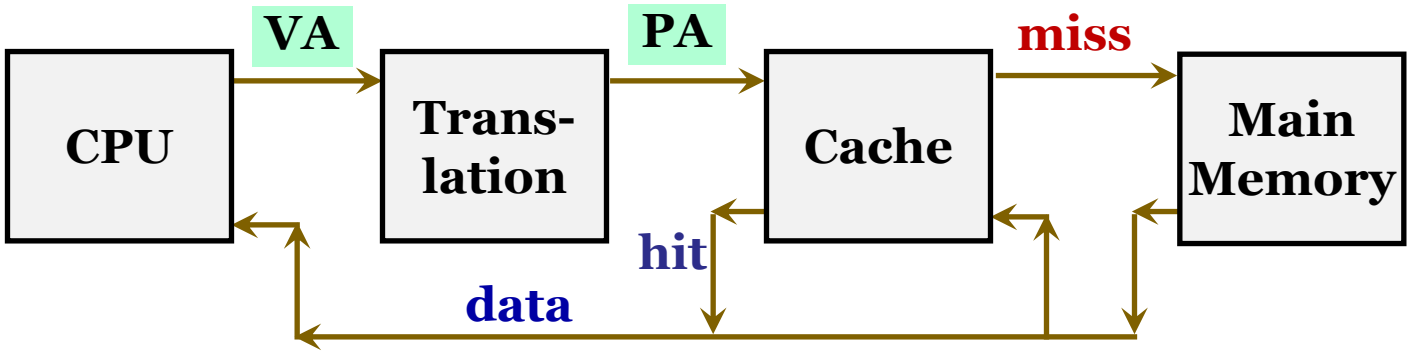
- So each memory request *first* requires an address translation from the virtual space to the physical space
  - A virtual memory miss (i.e., when the page is not in physical memory) is called a page fault

Address Translation Mechanisms



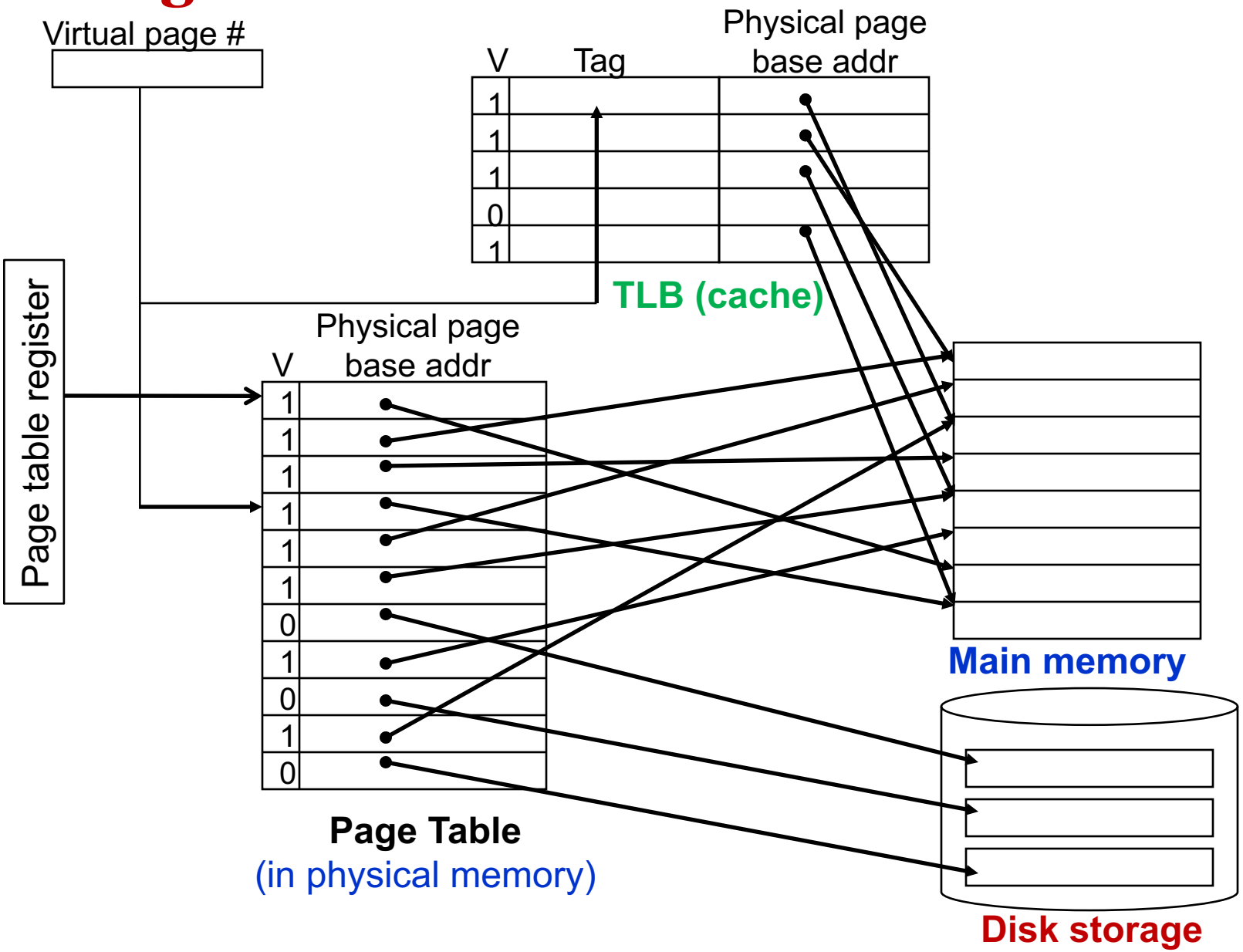
Virtual Addressing with a Cache

- Thus it takes an ***extra*** memory access to translate a VA to a PA



- This makes memory (cache) accesses very expensive (if every access was really *two* accesses)
- The hardware fix is to use a ***Translation Lookaside Buffer (TLB)*** – a small cache that keeps track of recently used address mappings to avoid having to do a page table lookup

Making Address Translation Fast



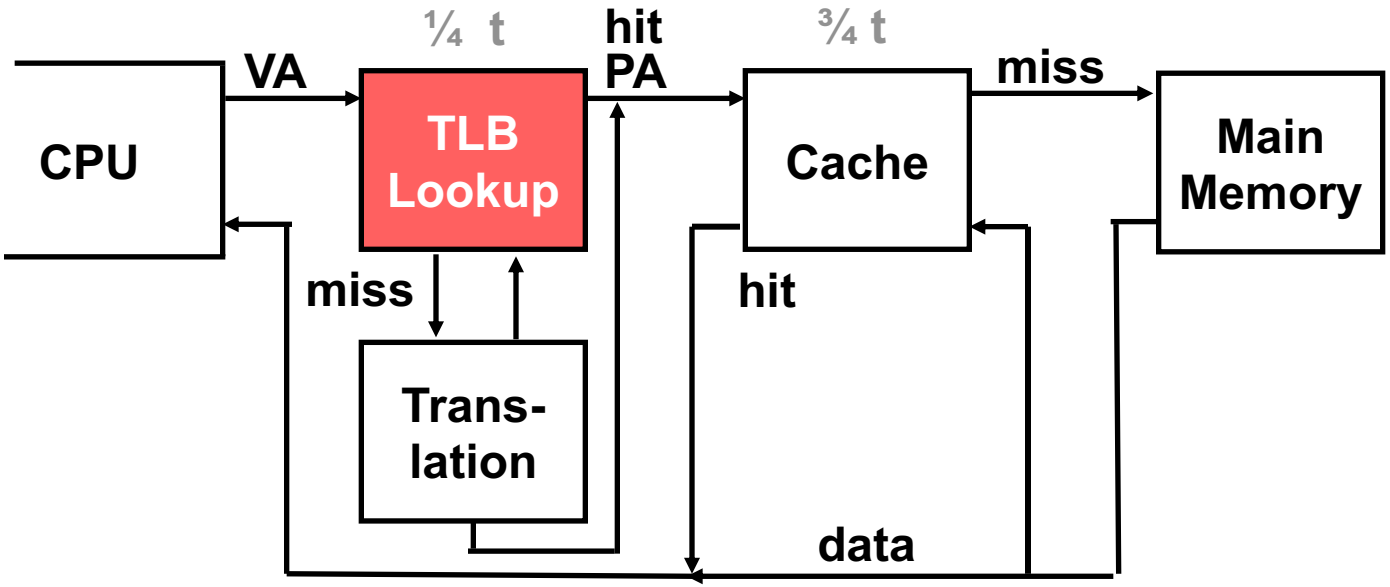
Translation Lookaside Buffers (TLBs)

- Just like any other cache, the TLB can be organized *as fully associative*, *set associative*, or *direct mapped*

V	Virtual Page #	Physical Page #	Dirty	Ref	Access

- **What** is used as the tag for ***identification***?
- TLB access time is typically smaller than cache access time (because TLBs are much smaller than caches)
  - TLBs are typically not more than 512 entries even on high end machines

A TLB in the Memory Hierarchy

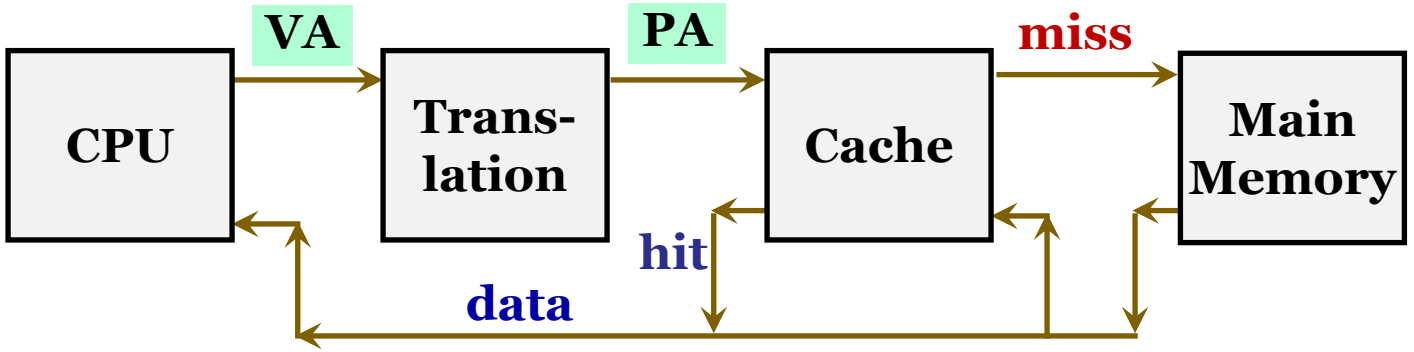


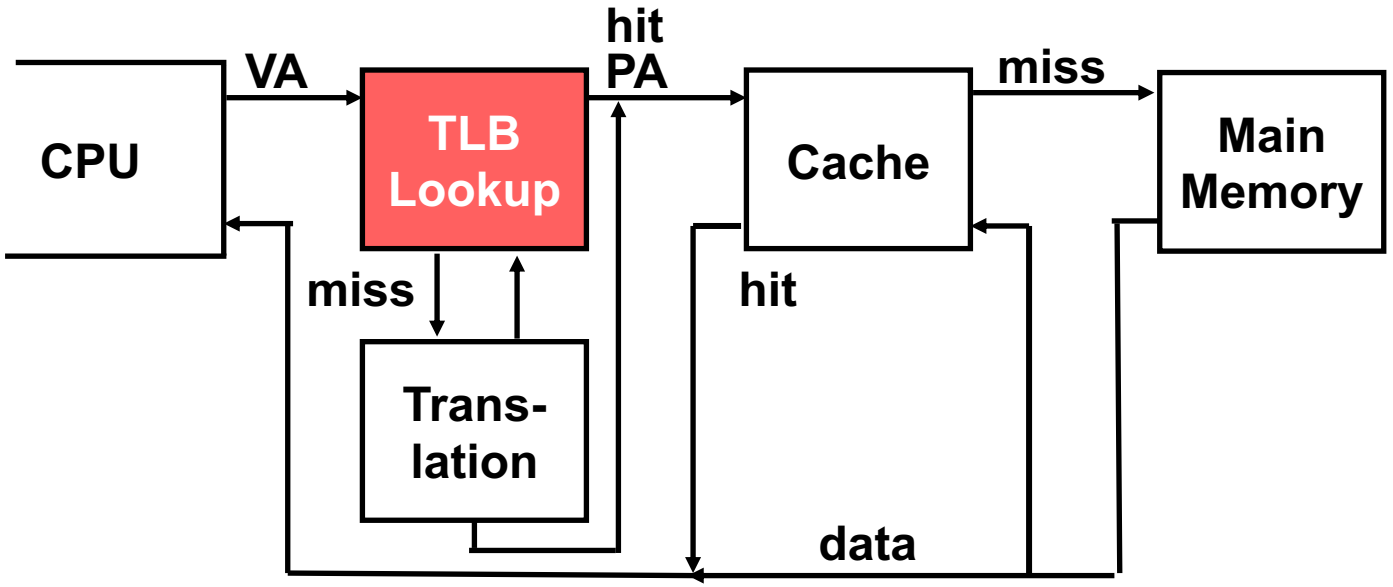
- A TLB **miss** – is it a page fault or merely a TLB miss?
  - If the page is loaded into main memory, then the TLB miss can be handled (in hardware or software) by loading the translation information from the page table into the TLB
    - Takes 10's of cycles to find and load the translation info into the TLB
  - If the page is not in main memory, then it's a true page fault
    - Takes 1,000,000's of cycles to service a page fault
- TLB misses are much more frequent than true ***page faults***

TLB Event Combinations

TLB	Page Table	Cache	Possible? Under what circumstances?
Hit	Hit	Hit	Yes – what we want!
Hit	Hit	Miss	Yes – although the page table is not checked if the TLB hits
Miss	Hit	Hit	Yes – TLB miss, PA in page table
Miss	Hit	Miss	Yes – TLB miss, PA in page table, but data not in cache
Miss	Miss	Miss	Yes – page fault
Hit	Miss	Miss/ Hit	Impossible – TLB translation not possible if page is not present in memory
Miss	Miss	Hit	Impossible – data not allowed in cache if page is not in memory

Review: with and w/o TLB





Roadmap

- Cache Organization
- Virtual Memory
- **2.3 Six Basic Cache Optimizations**
- Ten Advanced Optimizations of Cache Performance
- Memory Technology and Optimizations
- Virtual Memory and Protection
- Protection: Virtual Memory and Virtual Machines

Three Categories of Cache Optimizations

The Average Memory Access Time

$$Average\ Memory\ Access\ Time = Hit\ Time + Miss\ Rate * Miss\ Penalty$$

1) Reducing the miss rate

- Larger block size, larger cache size, higher associativity

2) Reducing the miss penalty

- Multilevel caches, giving reads priority over writes

3) Reducing the time to hit in the cache

- Avoiding address translation when indexing the cache

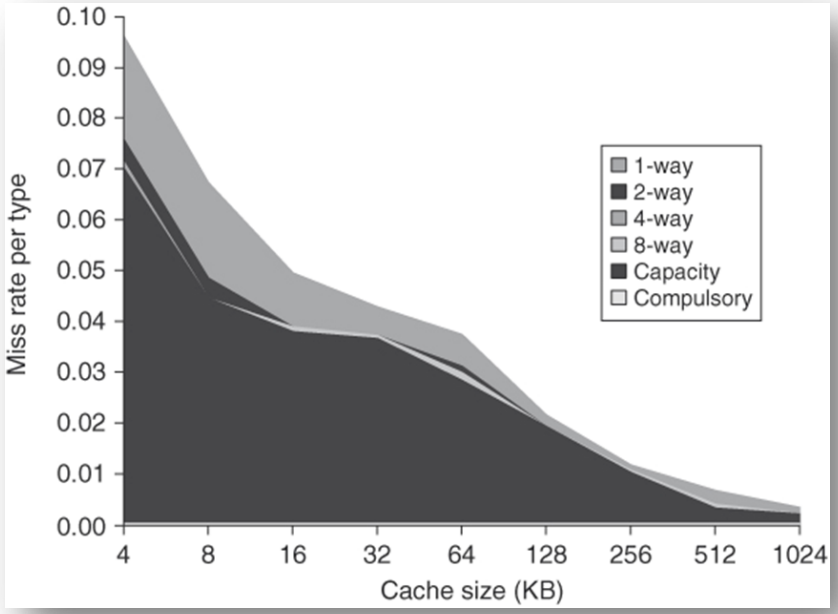
Three Categories of Misses

To gain better insights into the causes of cache misses

Compulsory	• The very first access to a block cannot be in the cache
Capacity	• If the cache cannot contain all the blocks needed during execution of a program, capacity misses occur
Conflict	• In case of set associative and direct mapped, conflict misses occur

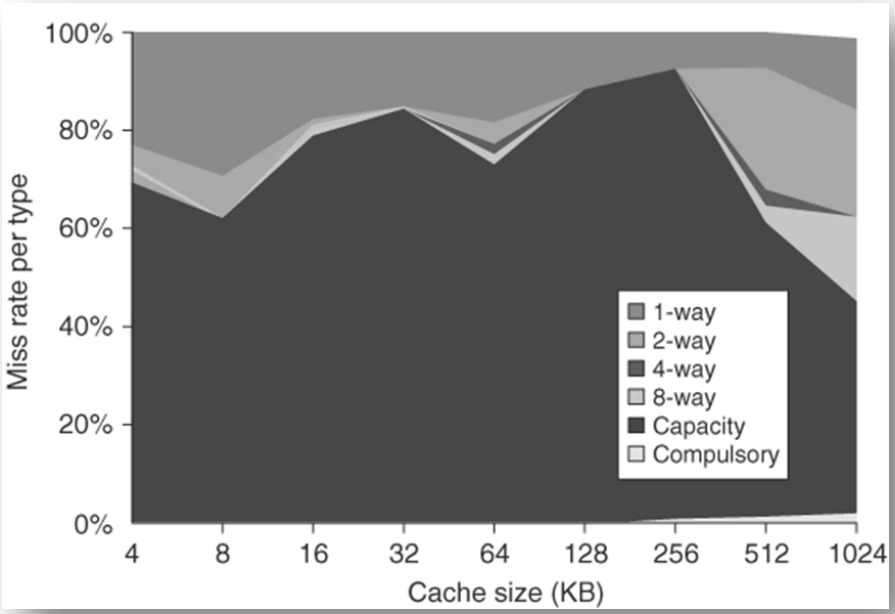
Comparison of Three Categories

Total miss rate



Actual data cache miss rates

Distribution of miss rate



The percentage in each category

Running SPEC2000

Six Basic Cache Optimizations

- Reduces compulsory misses
- Increases capacity and conflict misses, increases miss penalty

Larger block size

- Increases hit time, increases power consumption

Larger total cache capacity to reduce miss rate

- Reduces conflict misses
- Increases hit time, increases power consumption

Higher associativity

- Reduces overall memory access time

Higher number of cache levels

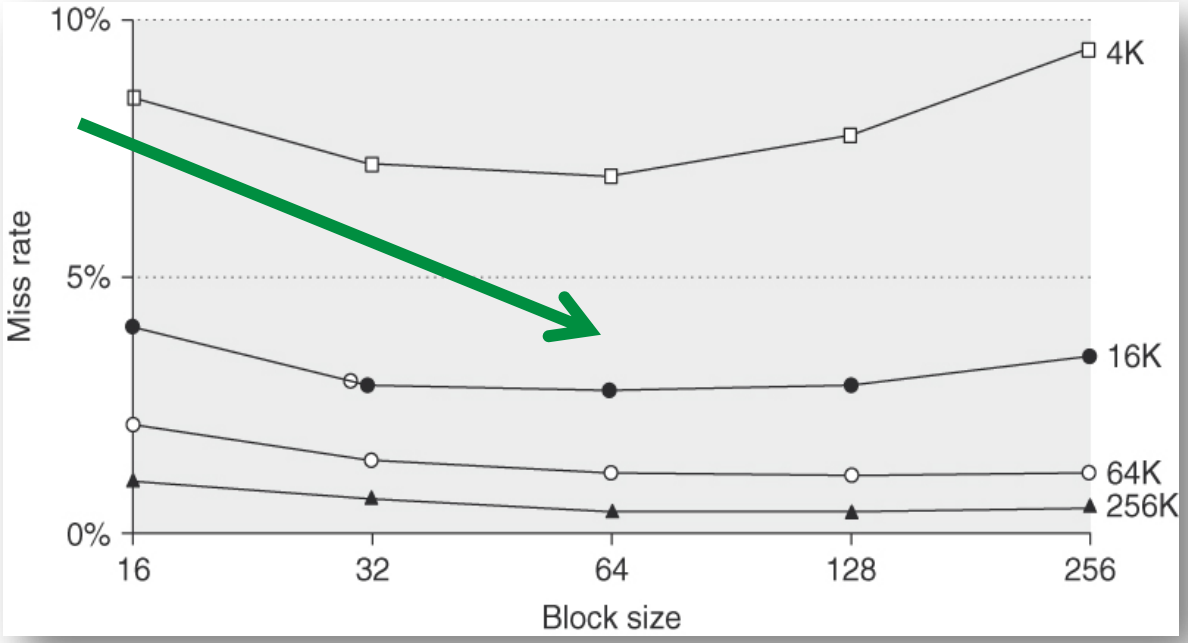
- Reduces miss penalty

Giving priority to read misses over writes

- Reduces hit time

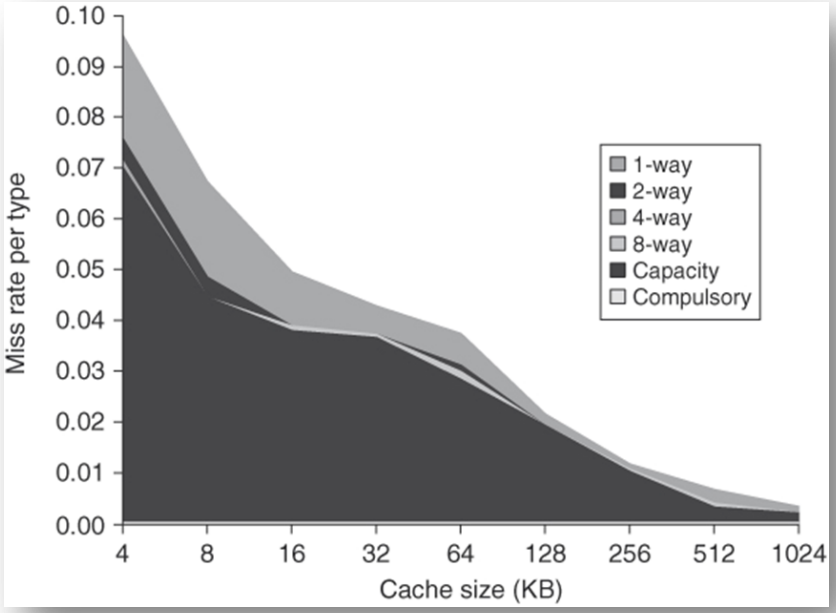
Avoiding address translation in cache indexing

1) Larger Block Size (Miss Rate↓)



- Reduce *compulsory misses* -> Principle of spatial locality
- At the same time, may increase *miss penalty*, *increase conflict misses*

2) Larger Caches (Miss Rate↓)

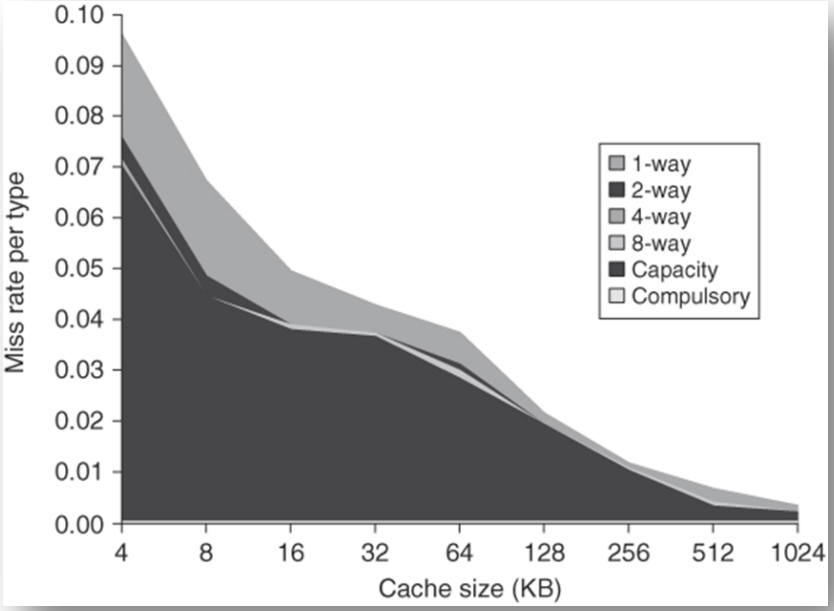


- Good for reducing capacity misses
- **Drawback**: potentially *longer hit time*, and *higher cost* and *power*
- Popular in off-chip caches

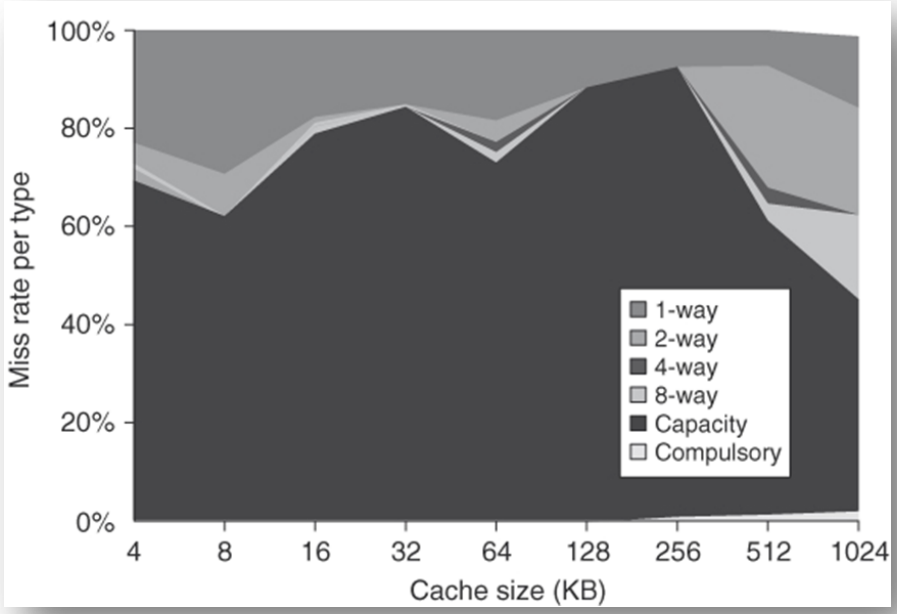
3) Higher Associativity (Miss Rate↓)

n-way cache, we increase n

With a higher associativity, the cache has a smaller number of sets.



Total miss rate



Distribution of miss rate

3) Higher Associativity (cont')

Two general rules of thumb

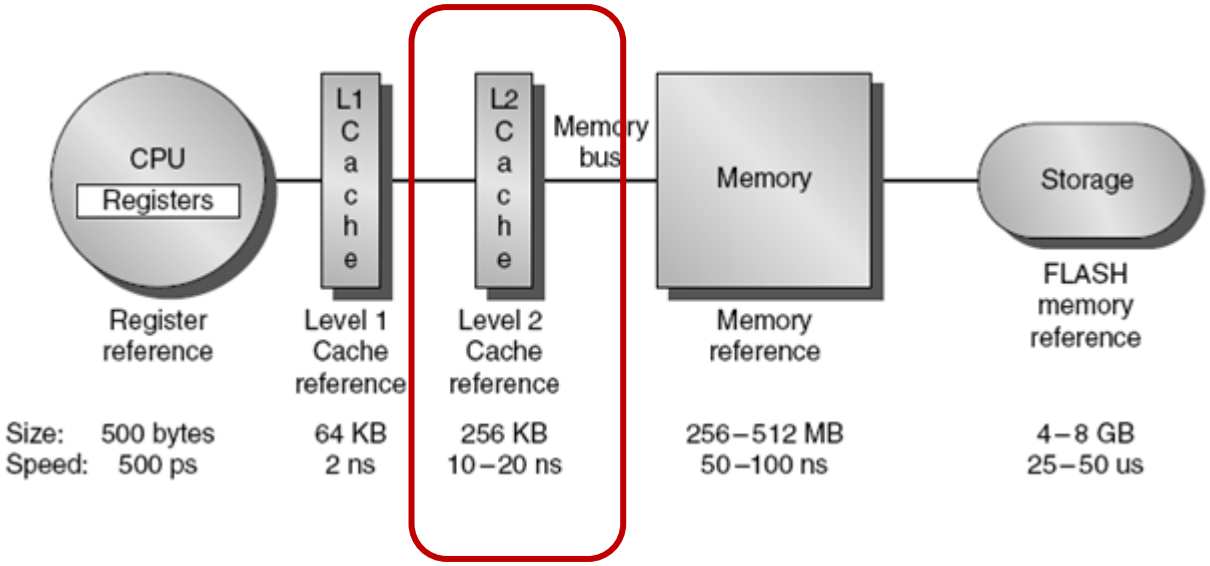
Eight-way set associative

- is for practical purposes as effective in reducing misses for these sized caches as fully associative

2:1 cache *rule of thumb*.

- A direct-mapped cache of size N has about the same miss rate as a two-way set associate cache of size N/2

4) Multilevel Caches (Penalty ↓)



- *Insert additional levels of cache* between level-1 cache and memory
  - Otherwise references should go to memory which introduces long latency!

4) Multilevel - Performance Analysis

$$\text{Average Memory Access Time} = \text{Hit Time}_{L1} + \text{Miss Rate}_{L1} * \text{Miss Penalty}_{L1}$$

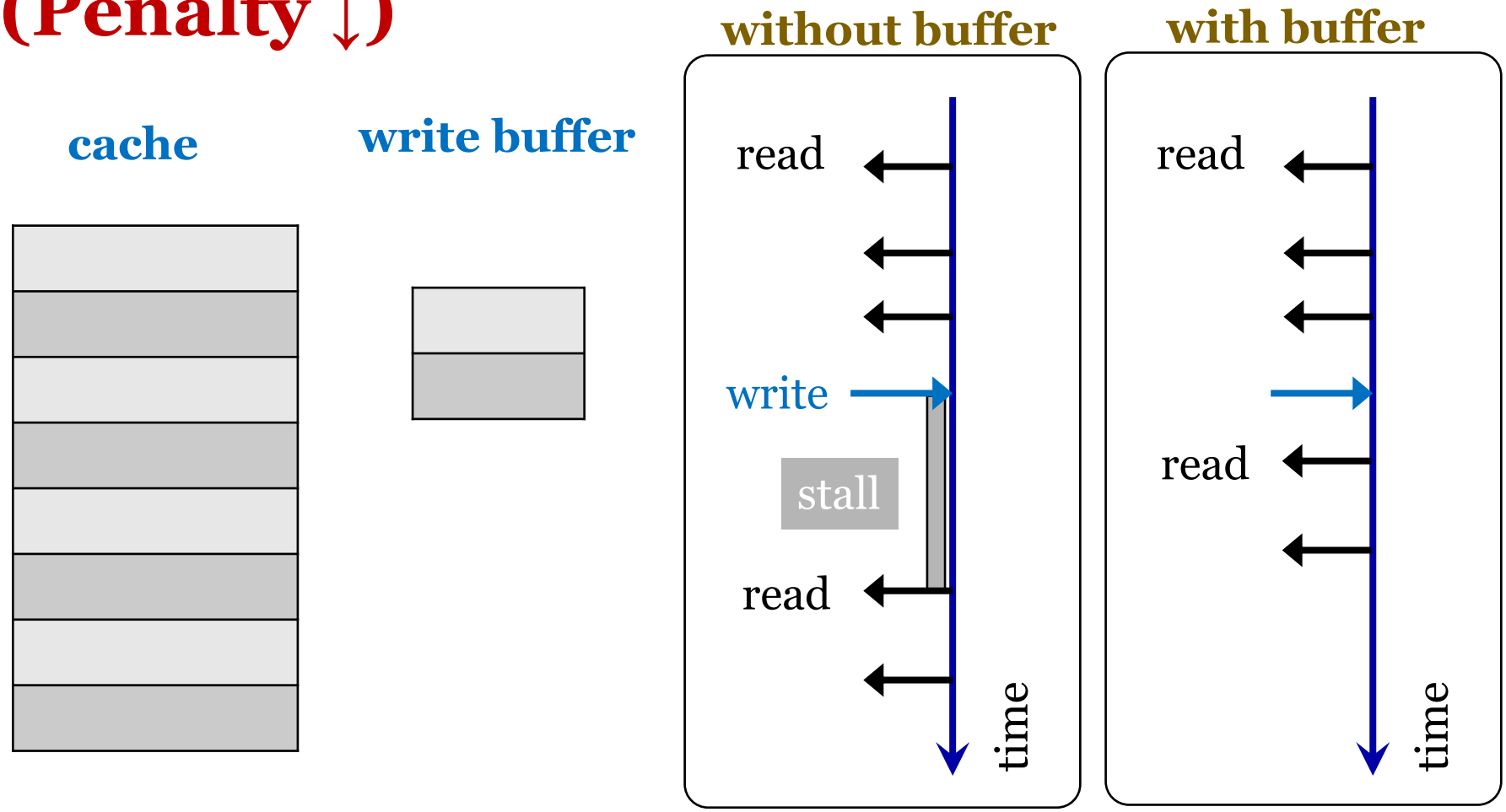
$$\text{Miss Penalty}_{L1} = \text{Hit Time}_{L2} + \text{Miss Rate}_{L2} * \text{Miss Penalty}_{L2}$$

Local miss rate: Miss Rate_{L1} Miss Rate_{L2}

Global miss rate: Miss Rate_{L1} $\text{Miss Rate}_{L1} * \text{Miss Rate}_{L2}$

(i.e., you can find neither in L1 nor L2)

5) Giving Priority to Read misses (Penalty ↓)



To serve reads before writes have been completed!

5) Potential Issue

```
SW R3, 512(R0)      ;M[512] ← R3      (cache index 0)
LW R1, 1024(R0)     ;R1 ← M[1024]     (cache index 0)
LW R2, 512(R0)      ;R2 ← M[512]      (cache index 0)
```

direct-mapped, write-through cache

a four-word write buffer

512 and 1024 to the same block

Read miss

the value in R2 always be equal to the value in R3?

5) Giving Priority to Read misses (Cont')

Two Solutions

The simplest solution

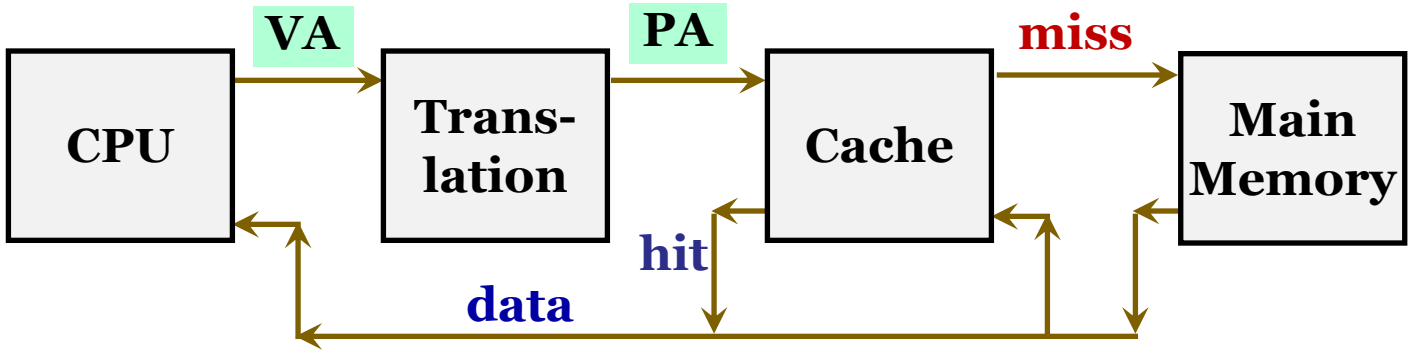
- For a read miss, to wait until the write buffer is empty

A better solution (priority to read misses)

- Upon a read miss, to check the contents of the write buffer.

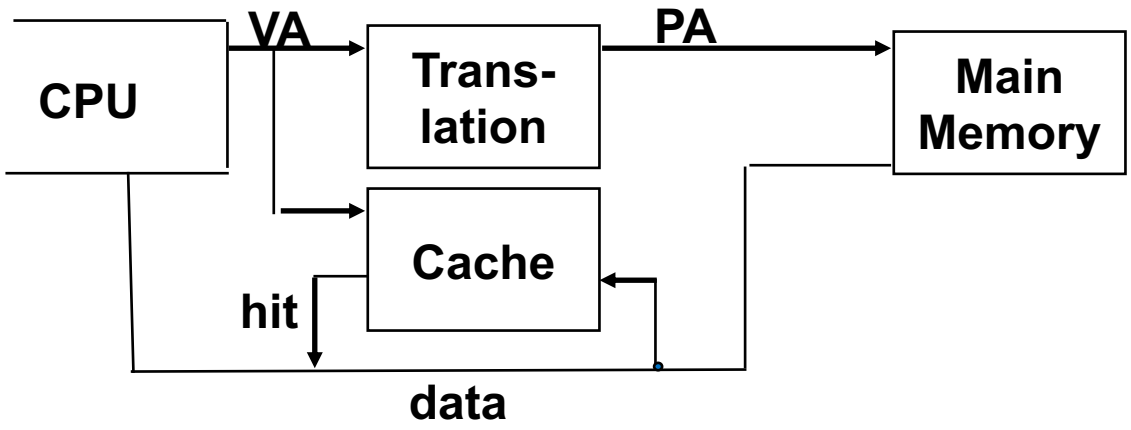
6) Avoiding Address Translations (Hit Time↓)

- For each memory access, we need to check whether the data is in cache
- To do the checking, we need to do **two tasks**,
  - Locating the possible data in cache (by using the index)
  - Comparing the tags



Cache with Virtual Address

If virtual addresses are used for cache, the time for address translations is saved (which is a frequent operation!!)



A virtually addressed cache would only require address translation on **cache misses**

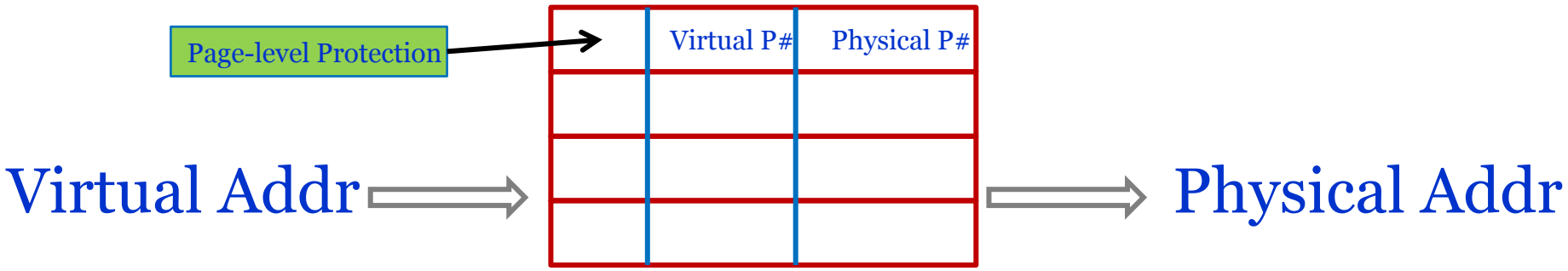
Cache with Virtual Address

Why not use virtual addresses to organize cache?

What are the main issues of using **virtual cache (cache organized with virtual addresses)**?

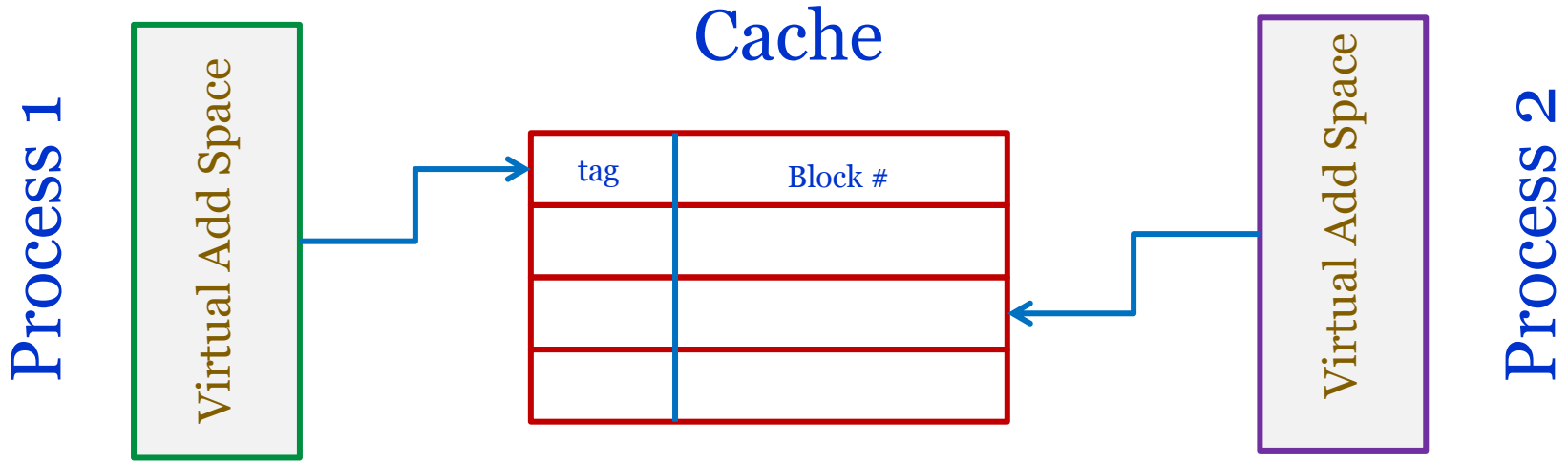
Issue 1: Protection

- Multiprogramming allows a computer to be shared by several programs running concurrently
- The OS should guarantee that processes do not interfere with each other's computations
- The computer designer helps the OS to provide protection so that one process cannot modify another
- Page-level protection is checked as part of the virtual to physical address translation

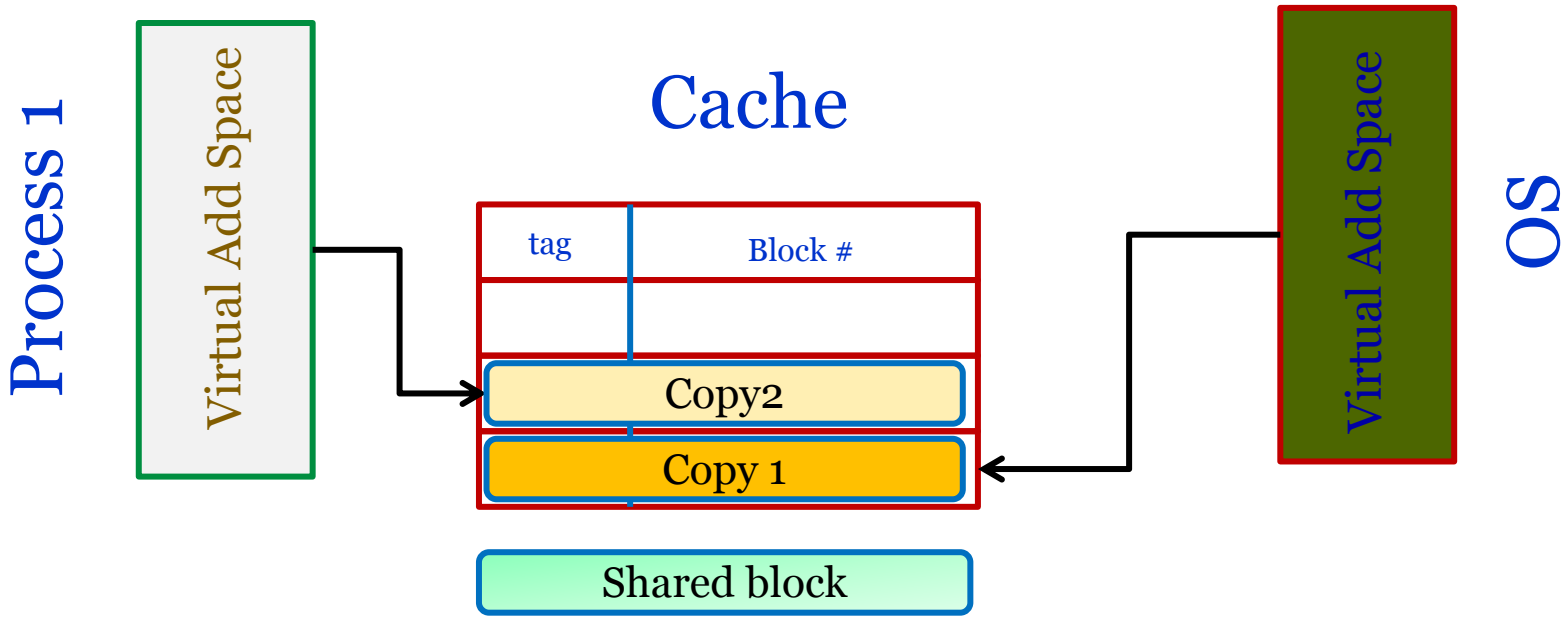


Issue 2: Process Switching

- The processor is shared by different processes
- It is common that a process is switched
- Note that each process is using its own virtual address space
- Each time a process is switched, the cache should be flushed!!



Issue 3: Synonyms or Aliases



- OS and processes are using different virtual addresses for the same physical address
- The duplicate addresses, ***called synonyms or aliases***, could result in two copies of the same data in the virtual cache
- Lead to **coherence issues:** Must update all cache entries with the same physical address or the memory becomes inconsistent

Idea: Overlapping Address Translation and Cache Reading

Goal: to get the best of both virtual and physical caches!

Virutally indexed, physically tagged!

- But, how?
- Note that *page offset* is identical in both virtual and physical addresses

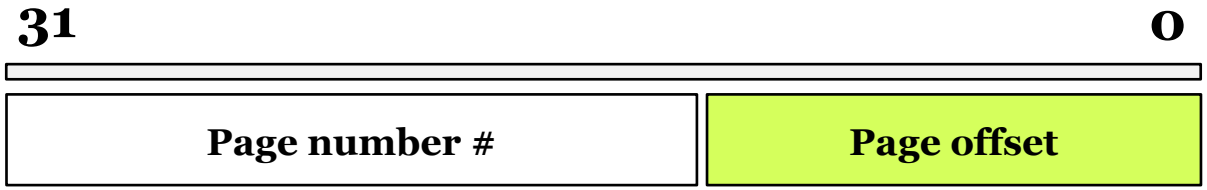
Approach: overlapping the cache access with the TLB access

Works (1) when the high order bits of the VA are used to access the TLB while (2) the low order bits are used as index into cache

Reducing Translation Time

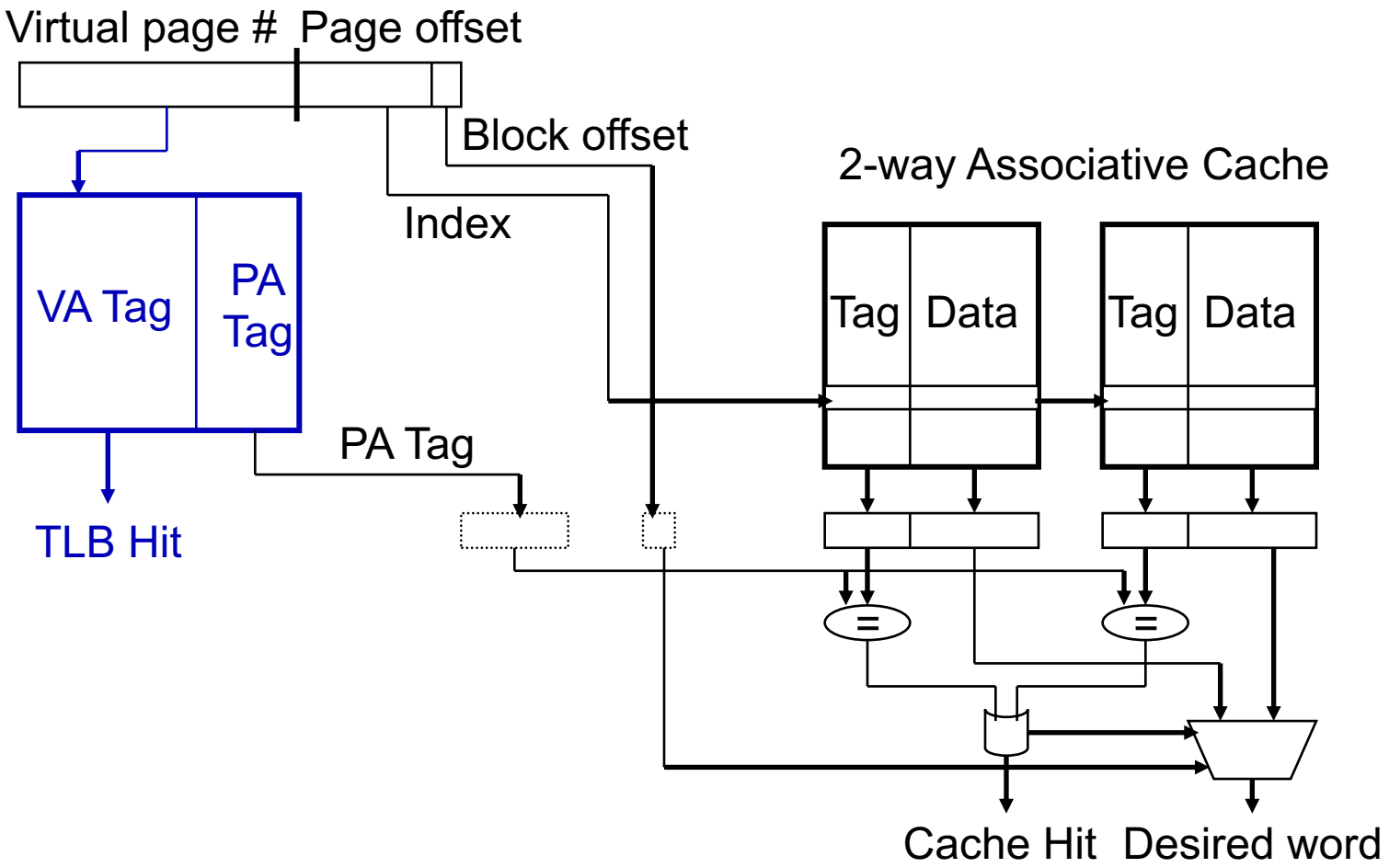
- ***Idea***: *use page offset to index the cache (page offset is not used for address translation)*
- At the same time
  - The cache is being read using the index
  - The address is translated
- The tag match still uses physical addresses

Virtual memory organization



Cache organization

Reducing Translation Time



Impact of Cache Performance and Complexity

Technique	Hit time	Miss penalty	Miss rate	Hardware complexity
Larger block size				
Larger cache size				
Higher associativity				
Multilevel caches				
Read priority over writes				
Avoiding address translation during cache indexing				

Impact of Cache Performance and Complexity

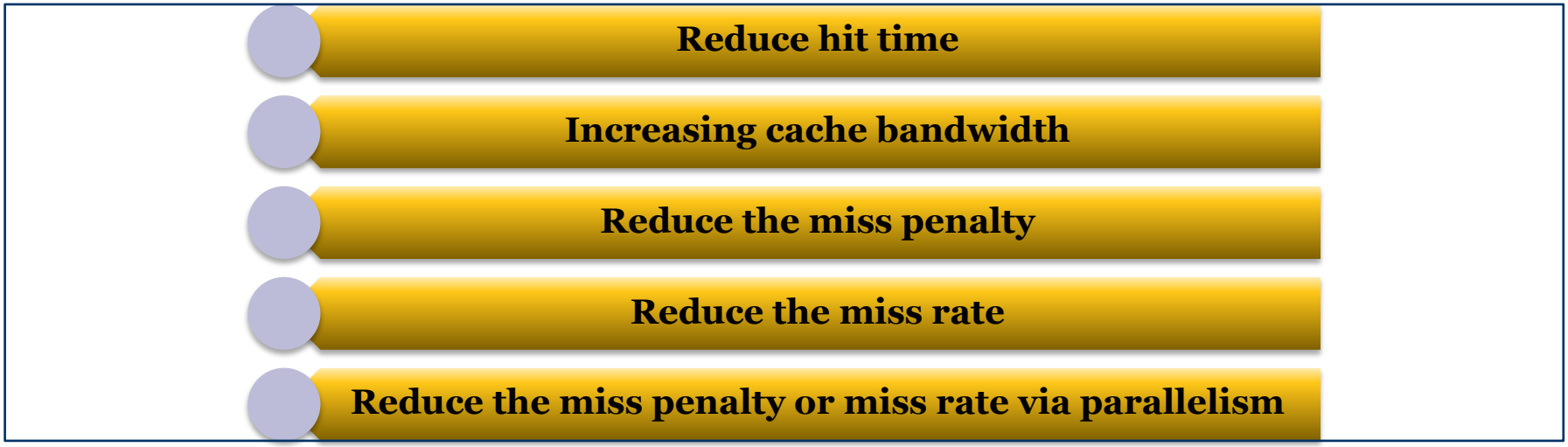
Technique	Hit time	Miss penalty	Miss rate	Hardware complexity
Larger block size		–	+	0
Larger cache size	–		+	1
Higher associativity	–		+	1
Multilevel caches		+		2
Read priority over writes		+		1
Avoiding address translation during cache indexing	+			1

Roadmap

- Cache Organization
- Virtual Memory
- Six Basic Cache Optimizations
- **2.4 Ten Advanced Optimizations of Cache Performance**
- Memory Technology and Optimizations
- Virtual Memory and Protection
- Protection: Virtual Memory and Virtual Machines

Ten Advanced Optimizations

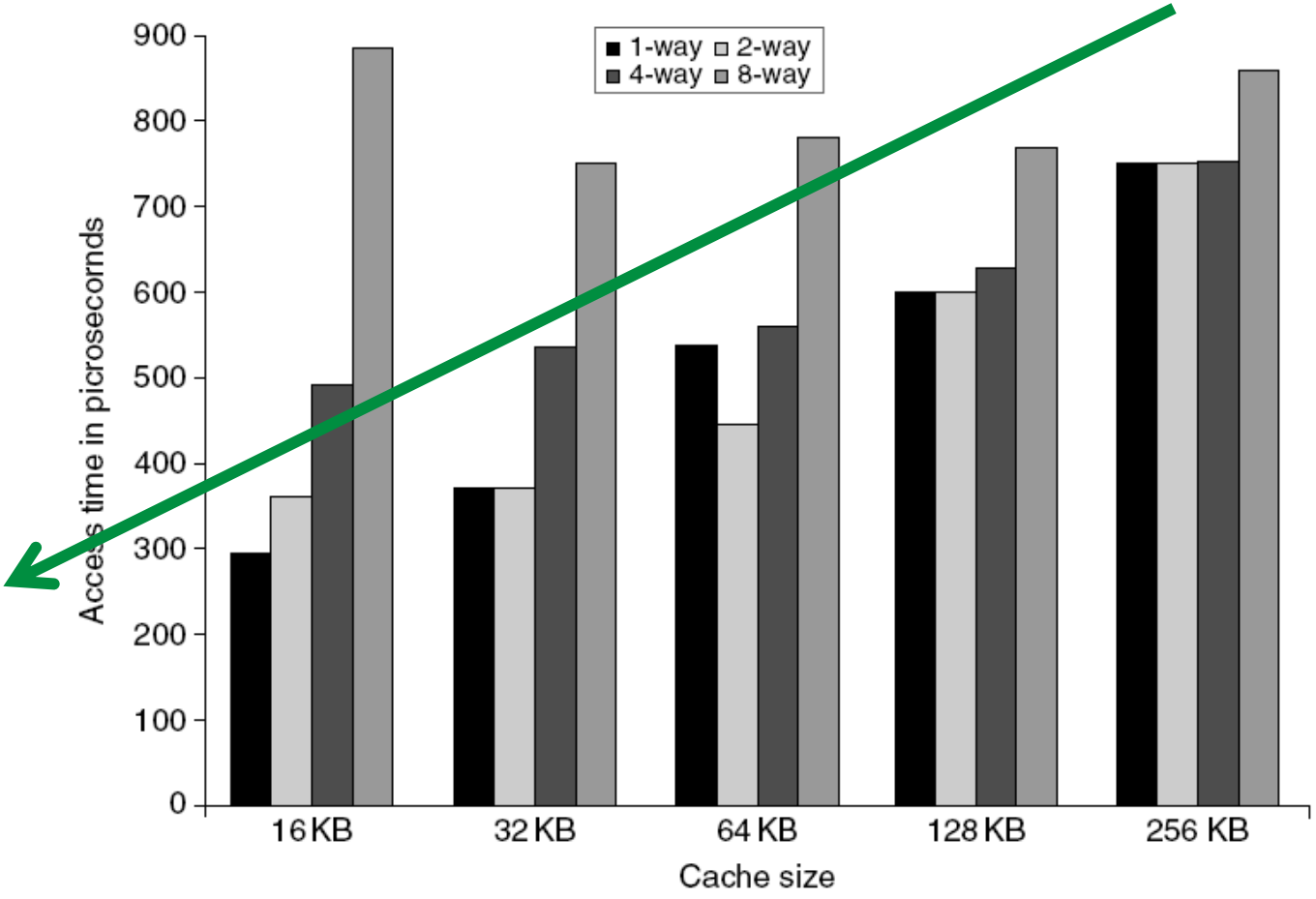
- The previous six basic optimizations try to reduce *miss rate*, *miss penalty* and *hit time*
- In the ten advanced optimizations, two more are added
  - *Cache bandwidth*
  - *Power consumption*



O1: Small and simple first level caches

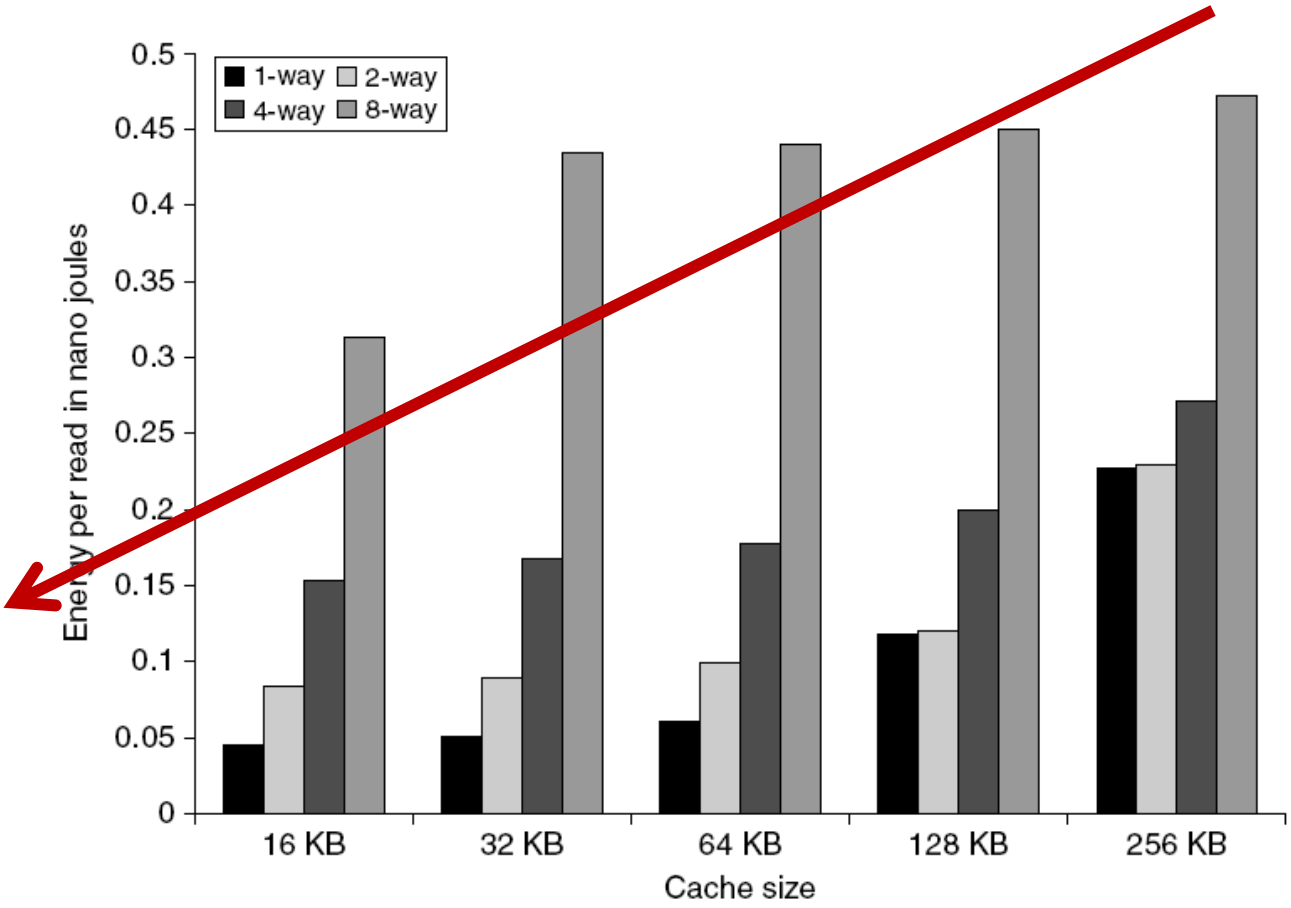
- Critical timing path in a cache hit:
  - addressing ***tag memory (in cache)*** using the index portion of the address
  - comparing the read tag value to the address
  - Choosing the correct data item (set associative)
- A smaller L1 cache reduce ***the hit time***!

Access Time versus L1 Size and Associativity



Access time vs. size and associativity

Energy versus L1 Size and Associativity



Energy per read vs. size and associativity

Higher Associativity in L1 Cache

In recent designs, there are three factors that have led to the use of higher associativity

Many processors take at least two clock cycles to access the cache

- Thus, the impact of a slightly longer hit time may not be critical

To keep TLB out of the critical path, many L1 caches should be virtually indexed (but physically tagged).

- This limits the size of the cache to the page size times the associativity

Higher associativity reduces conflict misses

Limiting the cache size

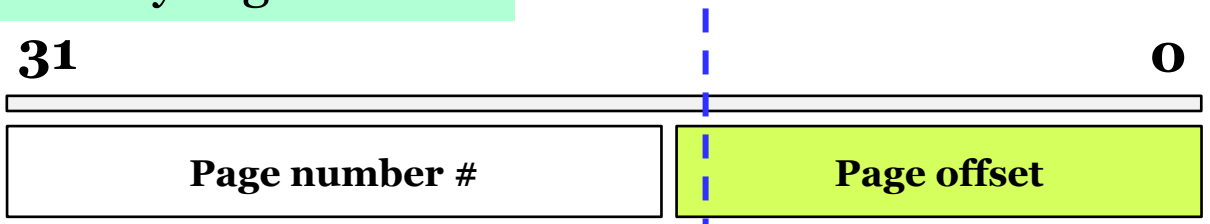
To keep TLB out of the critical path, virtually indexed physically tagged cache.

This limits the size of the cache

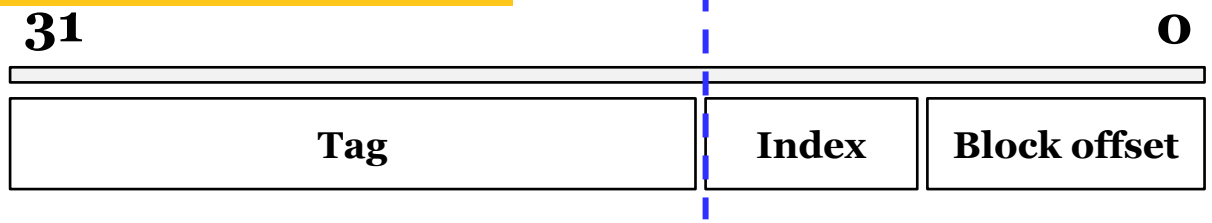
= the page size × the associativity

Limiting the cache size (Cont')

Virtual memory organization



Cache organization



Cache Size = Block Size × # of Sets × # of Ways

= 2^(block_offset + index) × # of Ways

≤ 2^(page_offset) × # of Ways

Page Size!

More: Limitation of virtually indexed, physically tagged approach

Example

The index is 9 bits, the block offset is 6 bits

Directed mapped cache

What is the size of the cache?

What is the minimum size of the page?

The index is 9 bits, the block offset 6 bits

8-way associative cache

What is the size of the cache?

What is the minimum size of the page?

O2: Way Prediction

- To improve **hit time**, predict the way to pre-set **mux**
  - To reduce conflict misses but maintain the hit speed of direct-mapped cache
  - **Mis-prediction** gives longer hit time
- Prediction accuracy
  - > 90% for two-way
  - > 80% for four-way
  - **I-cache has better accuracy than D-cache**
  - First used on MIPS R10000 in mid-90s
  - Used on ARM Cortex-A8

O3: Pipelining Cache

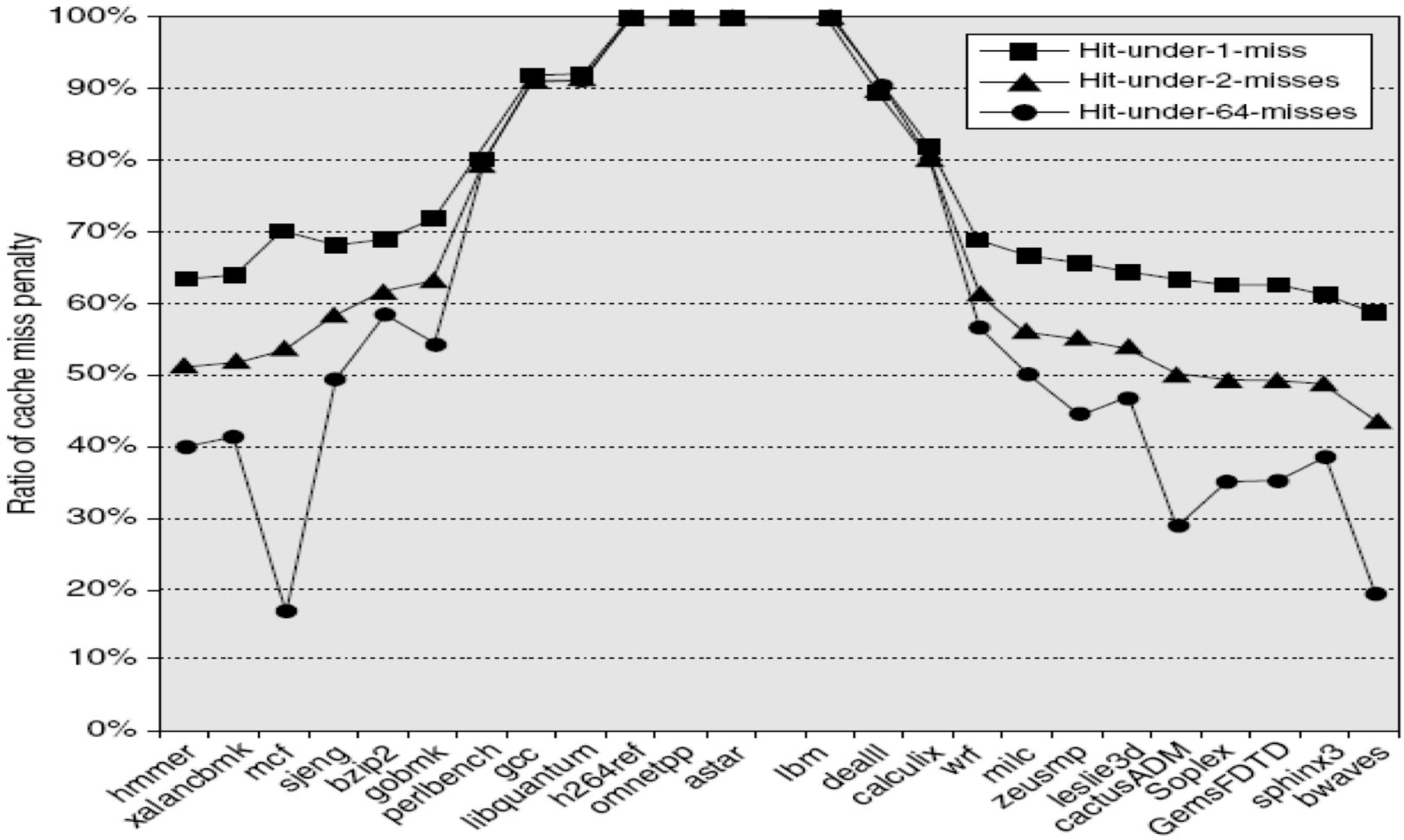
- **Pipeline cache** access to improve ***cache bandwidth***
  - The cycles for the pipeline for instruction cache access should be larger to make use of pipelining
  - Examples (*Intel evolvement of cache access cycles*):
    - Pentium: 1 cycle
    - Pentium Pro – Pentium III: 2 cycles
    - Pentium 4 – Core i7: 4 cycles
  - Allows more stages for pipelining results in the following consequences:
    - Meanwhile, it increases branch mis-prediction penalty
    - Makes it easier to increase ***associativity*** **(more time for performing parallel comparisons)**

O4: Nonblocking Caches

- Allow hits before previous misses complete
  - "Hit under miss"
  - "Hit under multiple miss"
- Thus, the miss penalty can be reduced!

- ***Rationale***: *Pipelined computers allow out-of-order execution*. Then, the processor need not stall on a data miss

Effectiveness of a Non-blocking Cache



O5: Multibanked Caches

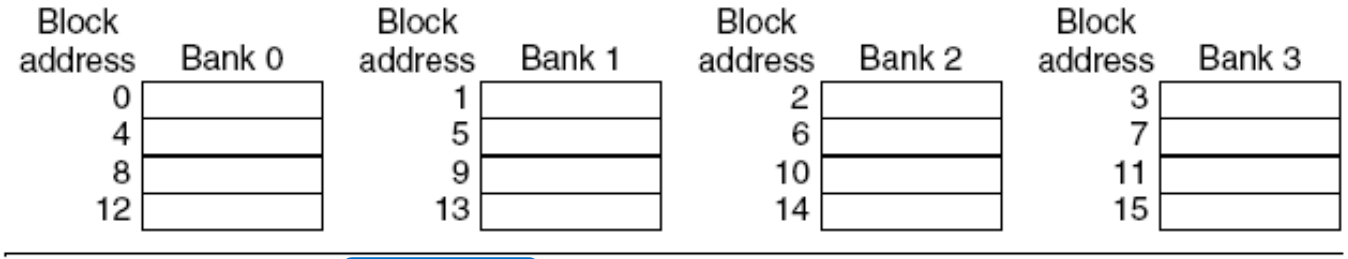


Figure 2.6 interleaved cache banks using block addressing. Assuming 64 bytes per blocks, each of these addresses would be multiplied by 64 to get byte addressing.

- Organize cache as ***independent banks*** to support ***simultaneous access***
  - ARM Cortex-A8 supports 1-4 banks for L2
  - Intel i7 supports 4 banks for L1 and 8 banks for L2
- Banking works best *when the accesses naturally spread themselves across the banks*

O6: Critical Word First, Early Restart

- **Observation: the processor normally needed just one word of the block at a time!**
- 1) Critical word first
  - Request missed word from memory first
  - Send it to the processor as soon as it arrives
- 2) Early restart
  - Request words in normal order
  - Send missed word to the processor as soon as it arrives
- *Miss penalty* can be reduced!
- **Effectiveness of these strategies depends on two factors! What are they?**
  - block size
  - likelihood of another access to the portion of the block that has not yet been fetched

O7: Merging Write Buffer

- When storing to a block that is already pending in the write buffer, update write buffer
- Reduces stalls due to full write buffer
- Thus, *miss penalty* can be reduced!

No write merging

Write address	V		V		V		V	
100	1	Mem[100]	0		0		0	
108	1	Mem[108]	0		0		0	
116	1	Mem[116]	0		0		0	
124	1	Mem[124]	0		0		0	

Write address	V		V		V		V	
100	1	Mem[100]	1	Mem[108]	1	Mem[116]	1	Mem[124]
	0		0		0		0	
	0		0		0		0	
	0		0		0		0	

With write merging

One entry has multiple words

O8: Compiler Optimizations

- ***Loop Interchange***: Swap nested loops to access memory in sequential order
- To reduce *Miss Rate*

```
/* Before */
for (j = 0; j < 100; j = j+1)
        for (i = 0; i < 5000; i = i+1)
                x[i][j] = 2 * x[i][j];
```

```
/* After */
for (i = 0; i < 5000; i = i+1)
        for (j = 0; j < 100; j = j+1)
                x[i][j] = 2 * x[i][j];
```

Suppose x is a two-dimensional array of size [5000,100]

X[i,j] and X[i,j+1] are adjacent

O9: Hardware Prefetching

To prefetch items before the processor requests them!

- Both instructions and data can be prefetched
  - **into caches or into an external buffer**
  - **Typically the processor fetches two blocks on a miss: the requested and the next block**
- ***To reduce penalty or miss rate***
- Prefetching relies on utilizing memory bandwidth that otherwise would be unused!

O10: Compiler Prefetching

- Insert ***prefetch instructions*** before data is needed
  - 1) Register prefetch
    - Loads data into register
  - 2) Cache prefetch
    - Loads data into cache

```
for (j = 0; j < 100; j = j+1) {
        prefetch(b[j+7][0]);
        /* b(j,0) for 7 iterations later */
        prefetch(a[0][j+7]);
        /* a(0,j) for 7 iterations later */
        a[0][j] = b[j][0] * b[j+1][0];};
for (i = 1; i < 3; i = i+1)
        for (j = 0; j < 100; j = j+1) {
                prefetch(a[i][j+7]);
                /* a(i,j) for +7 iterations */
                a[i][j] = b[j][0] * b[j+1][0];}
```

Summary of Impacts

Technique	Hit time	Band-width	Miss penalty	Miss rate	Power consumption	Hardware cost/ complexity
Small and simple caches						
Way-predicting caches						
Pipelined cache access						
Nonblocking caches						
Banked caches						
Critical word first and early restart						
Merging write buffer						
Compiler techniques to reduce cache misses						
Hardware prefetching of instructions and data						
Compiler-controlled prefetching						

Roadmap

- Review of Cache Organization
- Review of Virtual Memory
- Six Basic Cache Optimizations
- Ten Advanced Optimizations of Cache Performance
- **2.5 Memory Technology and Optimizations**
- Virtual Memory and Protection
- Protection: Virtual Memory and Virtual Machines

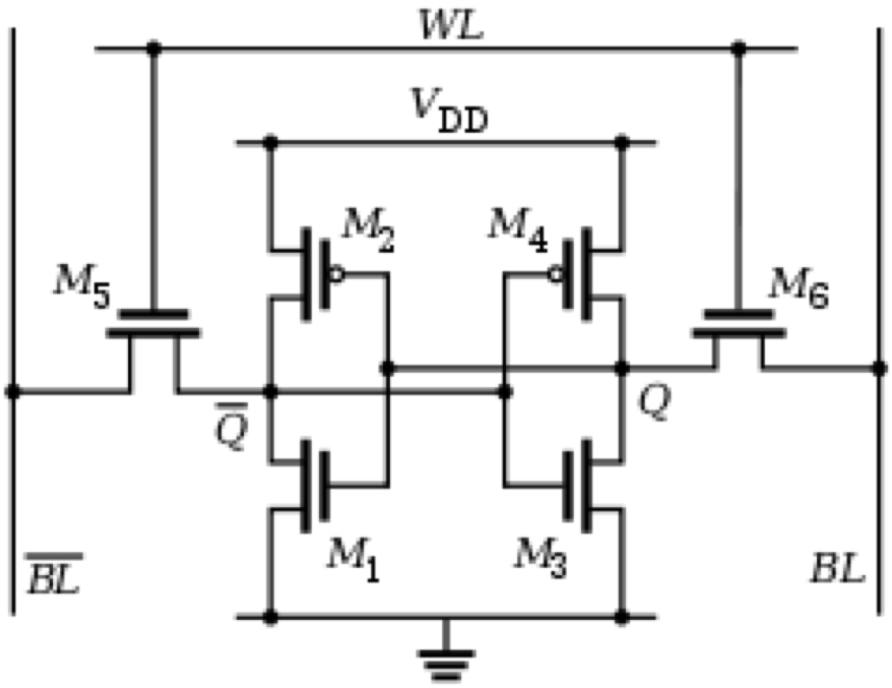
Memory Technology

- *Performance metrics*
  - Latency is concern of cache
    - Access time: Time between read request and when desired word arrives
    - Cycle time: Minimum time between unrelated requests to memory
  - Bandwidth is concern of multiprocessors and I/O

- DRAM used for main memory, SRAM used for cache

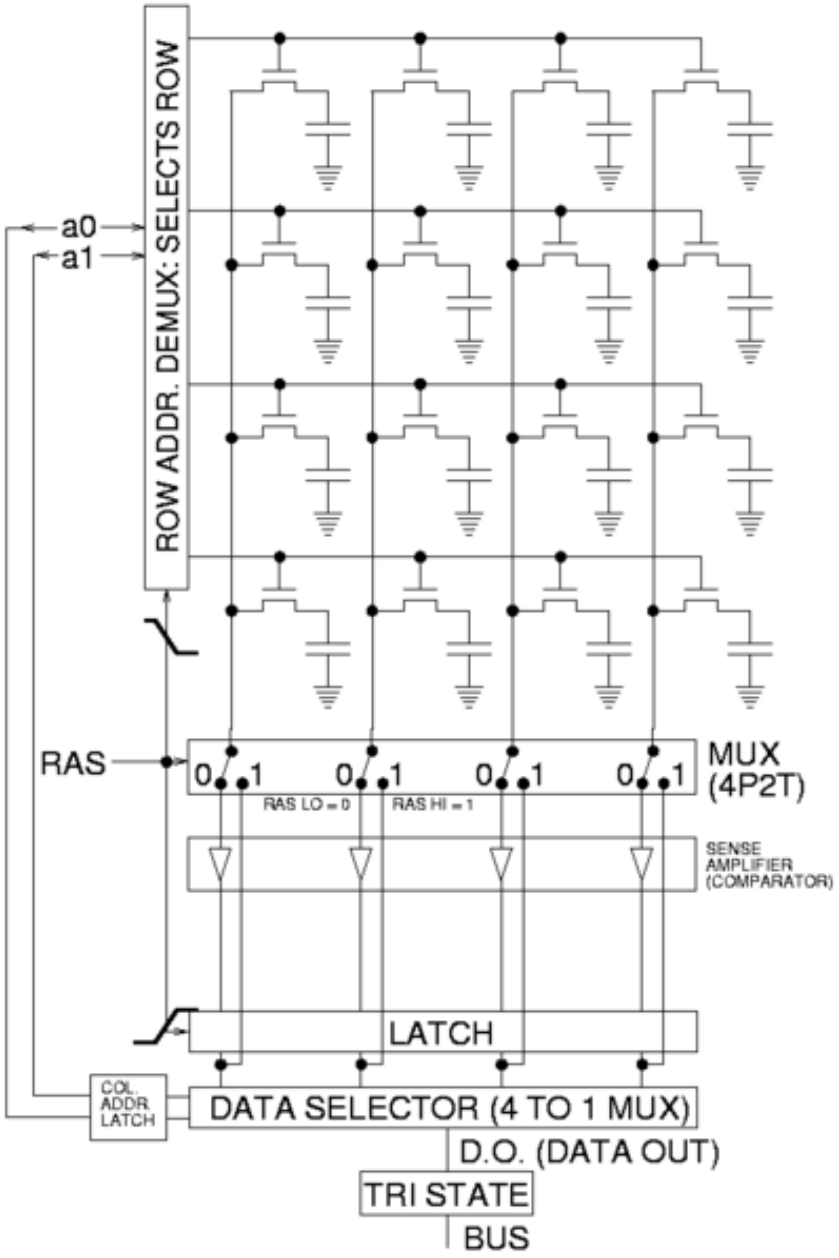
SRAM Review

- Requires *6 transistors/bit*
- Requires low power to retain bit
- *One cycle time* for access



DRAM Review

- *One transistor/bit*
- Address lines are multiplexed:
  - Upper half of address: row access strobe (RAS)
  - Lower half of address: column access strobe (CAS)
- Must be re-written after being read
- Must also be periodically refreshed
  - Every ~ 8 ms
  - Each row can be refreshed simultaneously



DRAM Technology

- Amdahl:
  - ***Memory capacity*** should grow linearly with *processor speed*
  - Unfortunately, memory capacity and speed has not kept pace with processors

- **Some optimizations:**
  - Multiple accesses to same row
  - Synchronous DRAM
    - Added clock signal to DRAM interface
    - Burst mode with critical word first
  - Wider interfaces
  - Double data rate (DDR)
  - Multiple banks on each DRAM device

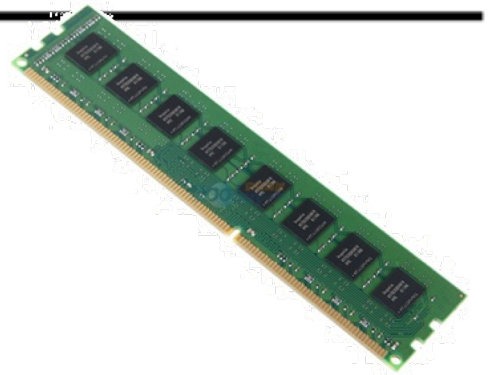
DRAM generations

Production year	Chip size	DRAM Type	Row access strobe (RAS)		Column access strobe (CAS)/ data transfer time (ns)	Cycle time (ns)
			Slowest DRAM (ns)	Fastest DRAM (ns)		
1980	64K bit	DRAM	180	150	75	250
1983	256K bit	DRAM	150	120	50	220
1986	1M bit	DRAM	120	100	25	190
1989	4M bit	DRAM	100	80	20	165
1992	16M bit	DRAM	80	60	15	120
1996	64M bit	SDRAM	70	50	12	110
1998	128M bit	SDRAM	70	50	10	100
2000	256M bit	DDR1	65	45	7	90
2002	512M bit	DDR1	60	40	5	80
2004	1G bit	DDR2	55	35	5	70
2006	2G bit	DDR2	50	30	2.5	60
2010	4G bit	DDR3	36	28	1	37
2012	8G bit	DDR3	30	24	0.5	31

Performance improvement of row access time is about only 5% per year

Relations of Different Numbers and Names

Standard	Clock rate (MHz)	M transfers per second	DRAM name	MB/sec /DIMM	DIMM name
DDR	133	266	DDR266	2128	PC2100
DDR	150	300	DDR300	2400	PC2400
DDR	200	400	DDR400	3200	PC3200
DDR2	266	533	DDR2-533	4264	PC4300
DDR2	333	667	DDR2-667	5336	PC5300
DDR2	400	800	DDR2-800	6400	PC6400
DDR3	533	1066	DDR3-1066	8528	PC8500
DDR3	666	1333	DDR3-1333	10,664	PC10700
DDR3	800	1600	DDR3-1600	12,800	PC12800
DDR4	1066–1600	2133–3200	DDR4-3200	17,056–25,600	PC25600



金士顿（Kingston）DDR3 1333 4G 台式机内存

DDR SDRAM

DDR2

- Lower power (2.5 V -> 1.8 V)
- Higher clock rates (266 MHz, 333 MHz, 400 MHz)

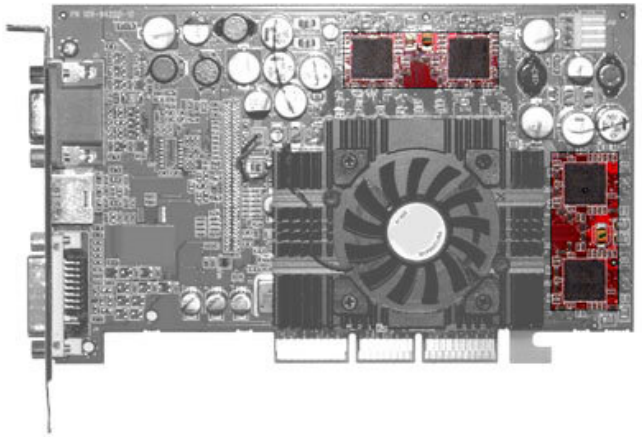
DDR3

- 1.5 V
- 800 MHz

DDR4

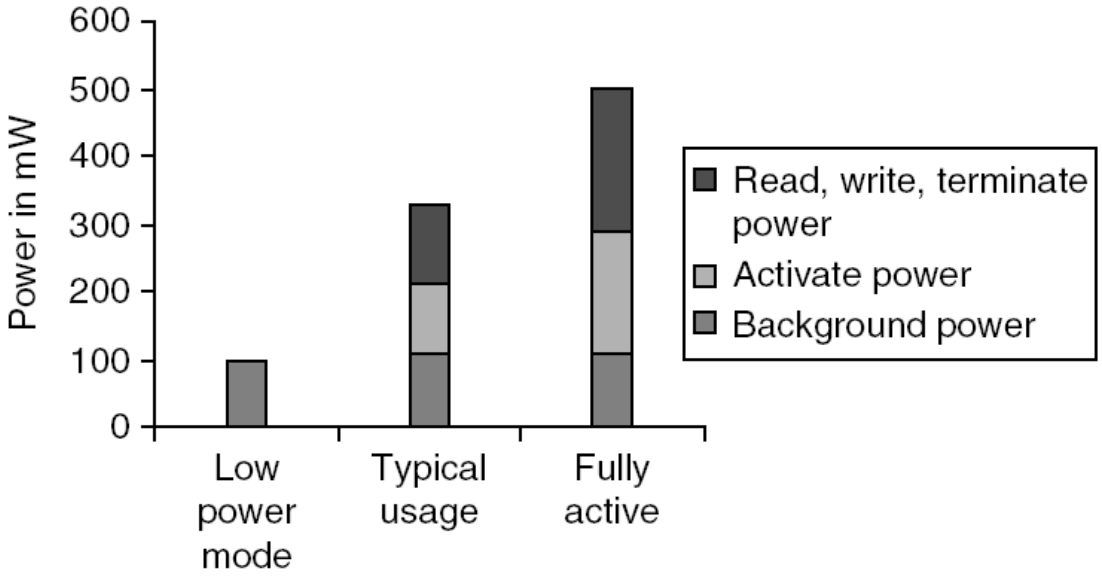
- 1-1.2 V
- 1600 MHz

Graphics memory



- Achieve 2-5X bandwidth per DRAM vs. DDR3
  - Wider interfaces (32 vs. 16 bit)
  - Higher clock rate
    - Possible because they are attached to GPU via soldering instead of socketted DIMM modules
- GDDR5 is graphics memory based on DDR3

Reducing power in SDRAMs



- Lower voltage, and use of banks
- Low power mode

Memory Dependability

- Memory is susceptible to *cosmic rays*
- ***Soft errors:*** dynamic errors
  - Detected and fixed by error correcting codes (ECC)
- ***Hard errors***: permanent errors
  - Use ***spare rows*** to replace defective rows

Flash Memory

- Type of EEPROM.
- ***Feature***: can hold data without any power. Non volatile
- Must be erased (*in blocks*) before being overwritten, not a single word
- Limited number of write cycles, typical of 100,000
- Cheaper than SDRAM, more expensive than disk
- Slower than SDRAM, faster than disk

New Trends: NVM

Non-volatile memory

You may already know:

Read-only memory

Flash memory

You probably do not know:

FeRAM

PCM

MRAM

Byte addressable
access time comparable to DRAM