

CS2200
Systems and Networks
Spring 2022

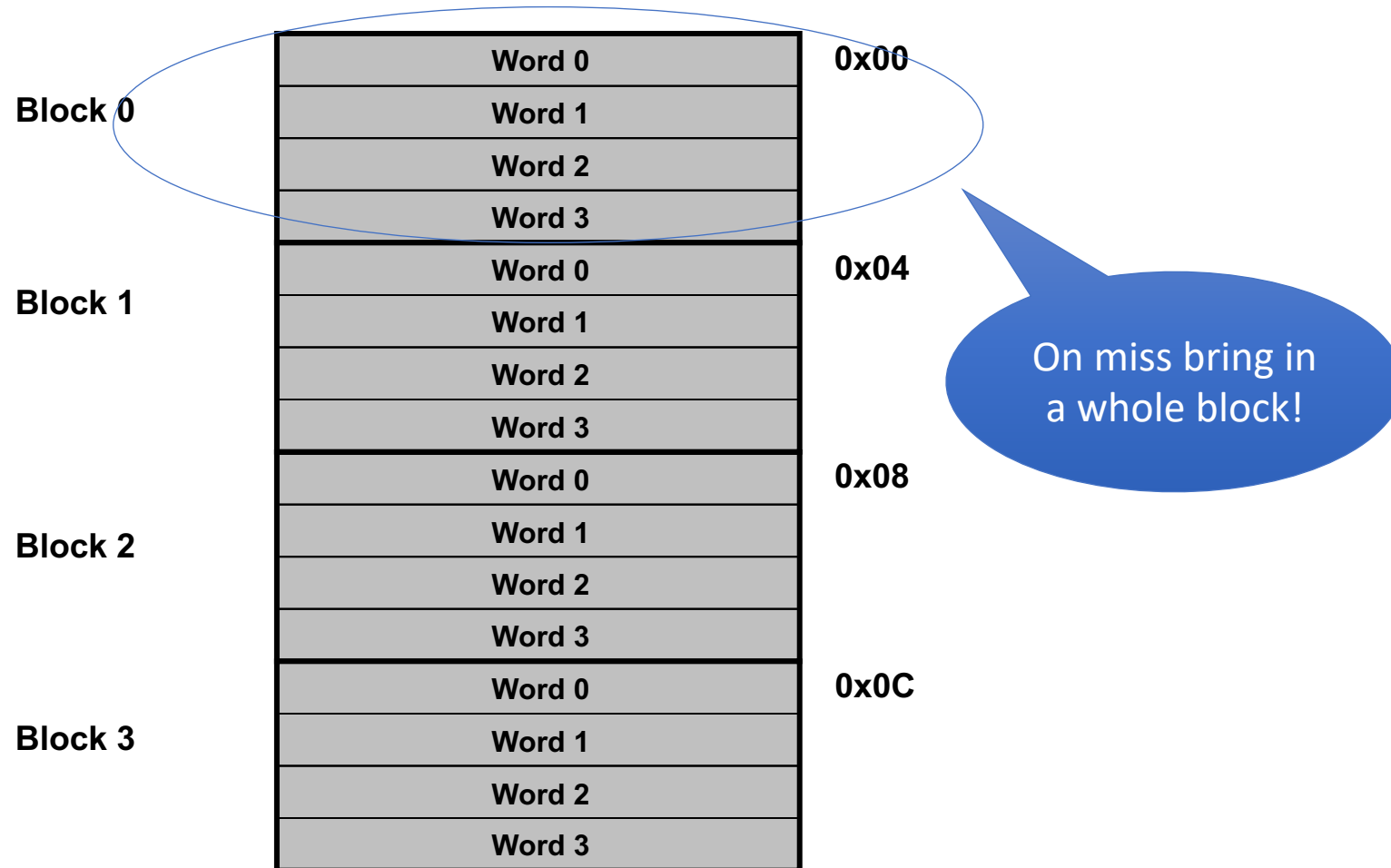
Lecture 20:
Memory Hierarchy pt 3

Alexandros (Alex) Daglis
School of Computer Science
Georgia Institute of Technology
adaglis@gatech.edu

How to improve cache efficiency

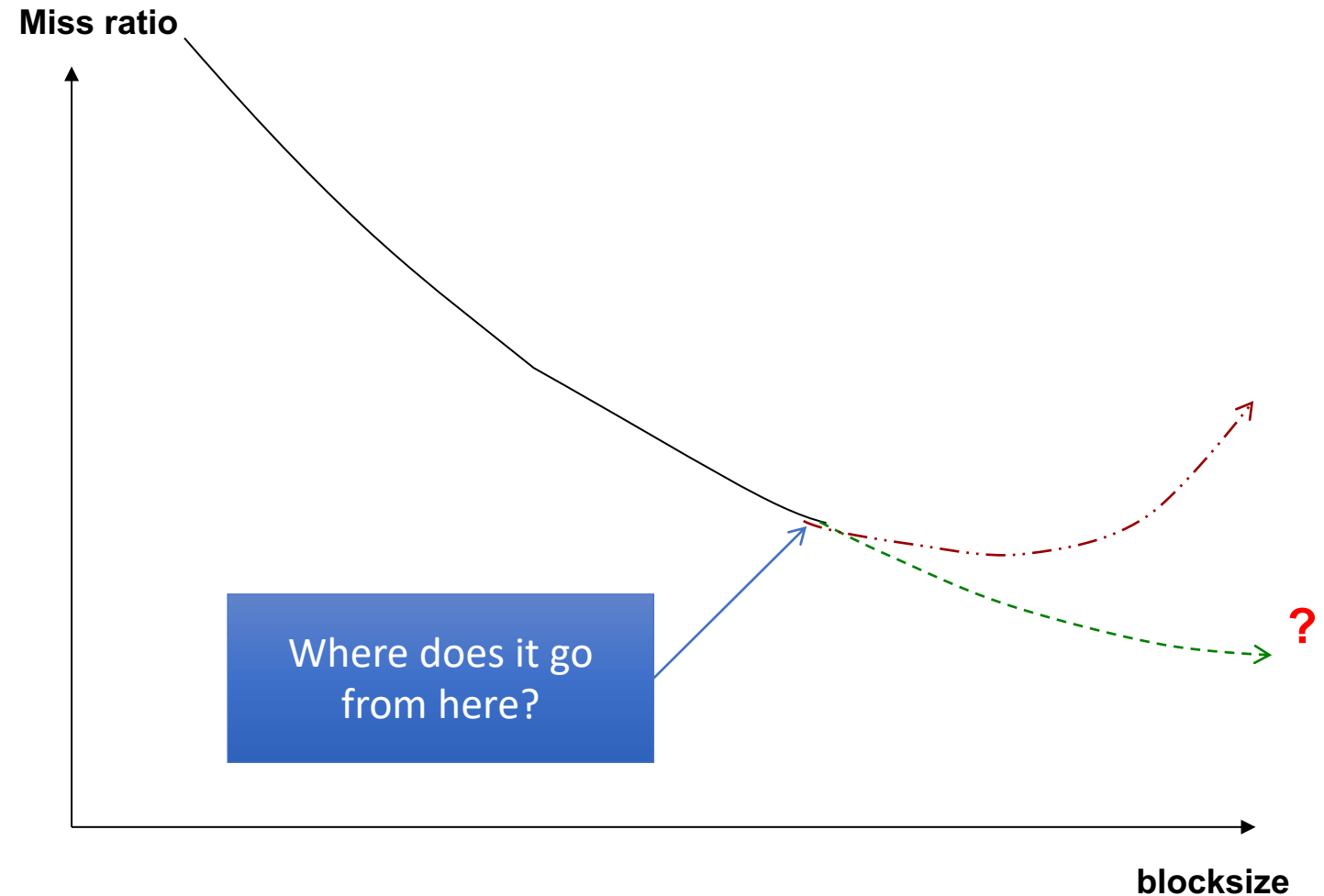
- Exploit spatial locality
 - Bring more from memory into cache at a time
- Better organization
 - Exploit working set concept

Spatial Locality



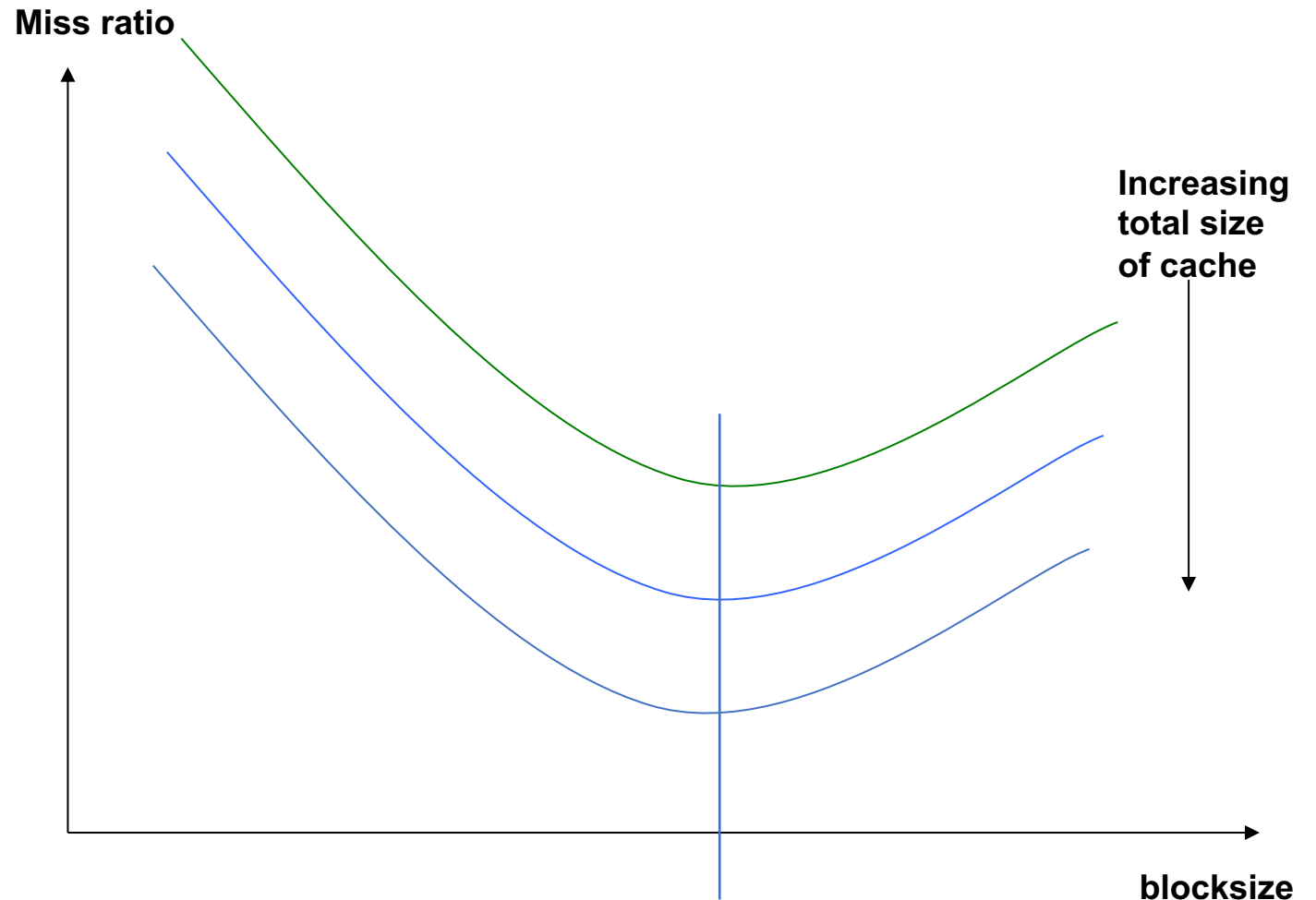
Increased block size?

- Exploits more spatial locality
- Reduces miss ratio



Increased block size?

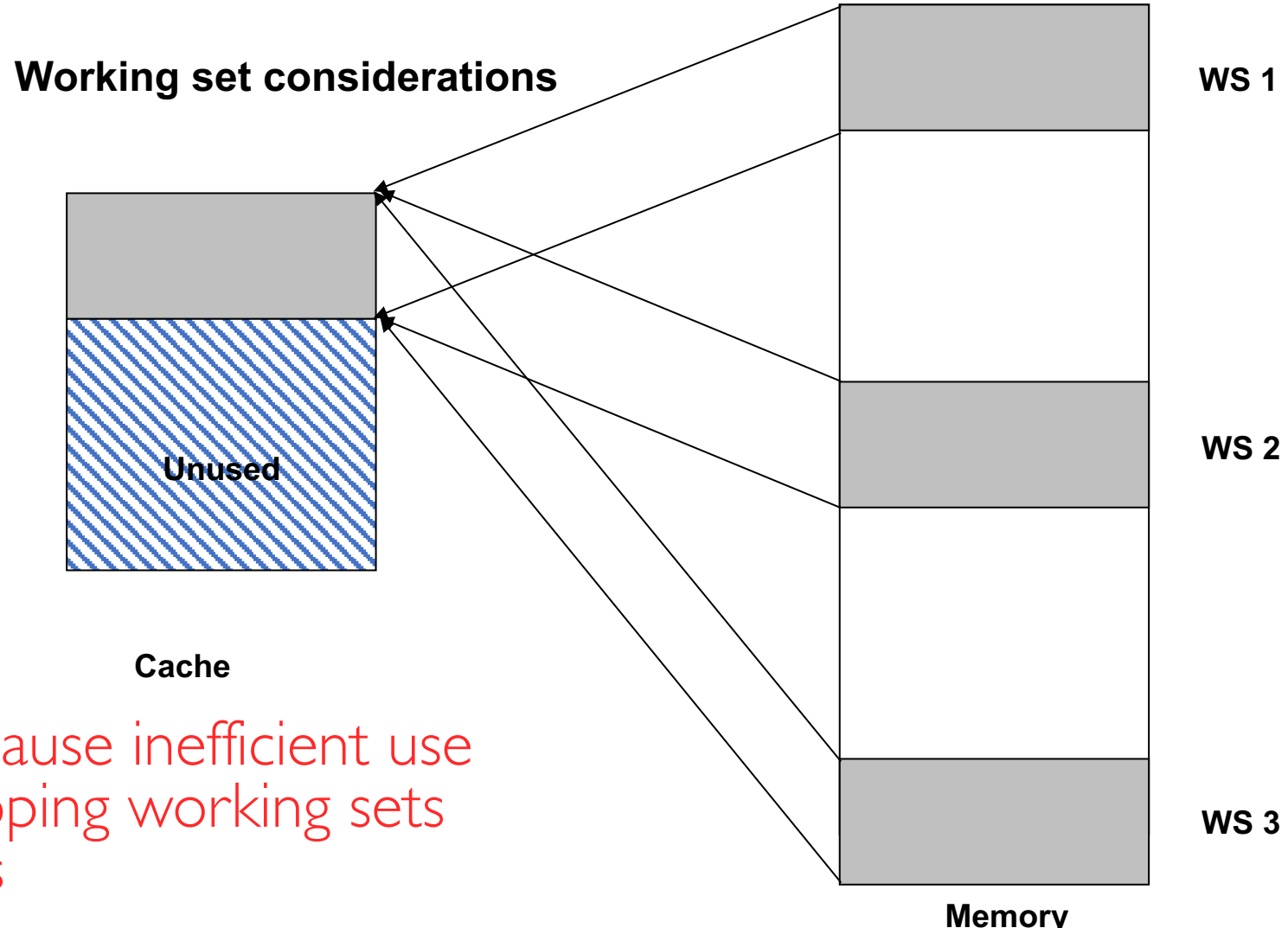
- There is a point where things get worse
- When the working set changes, larger blocks have to be fetched
- Memory can only transfer so fast and it can become the bottleneck



How to improve cache efficiency

- Exploit spatial locality
 - Bring more from memory into cache at a time
- Better organization
 - Exploit working set concept

Working set considerations

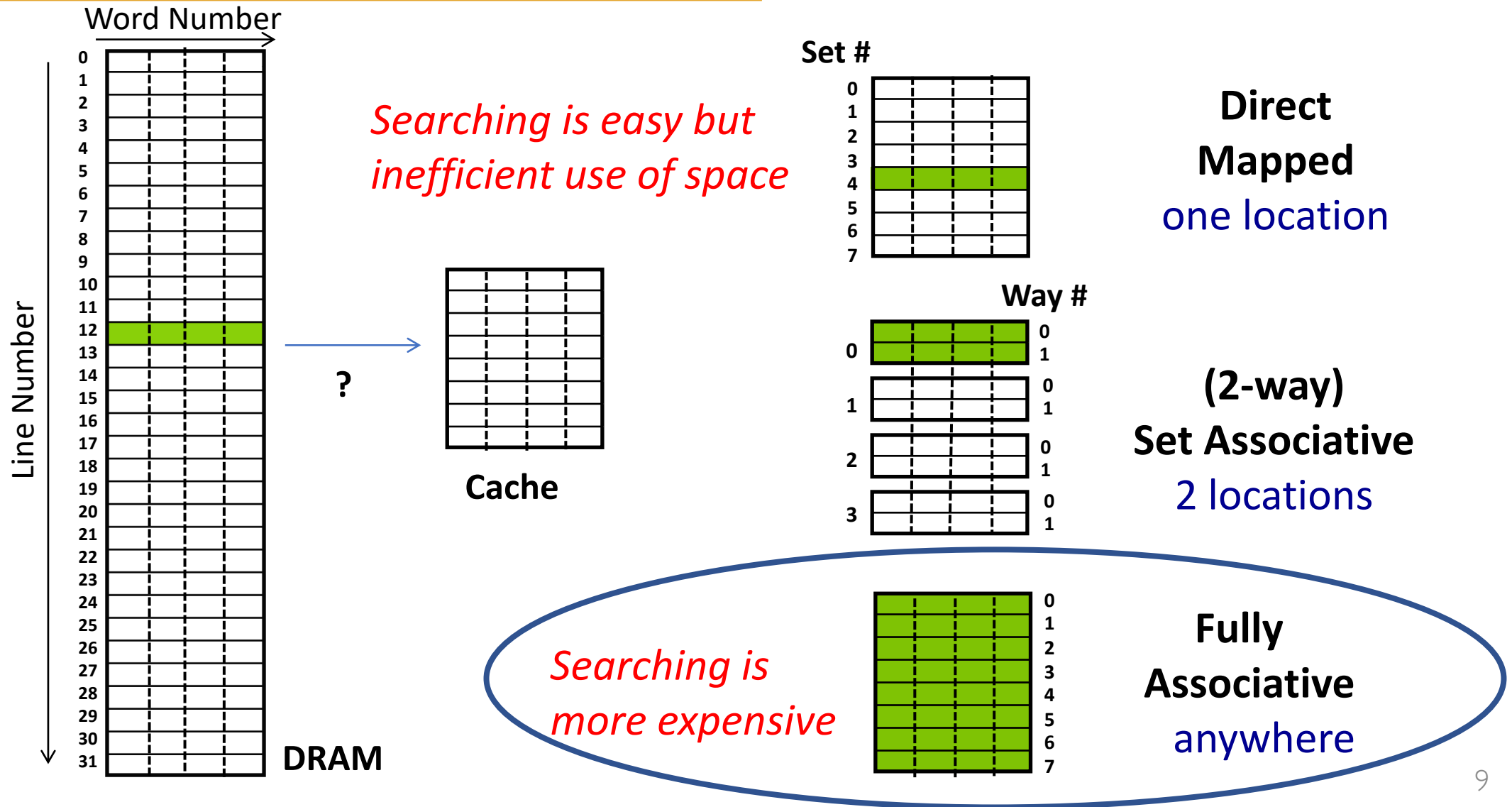


Direct mapping can cause inefficient use of cache with overlapping working sets due to conflict misses

What would be best?

- Allow any memory block to be brought into any cache block
 - This is similar to the ability of mapping any virtual page to any available physical page frame
- Fully associative mapping

Cache Placement



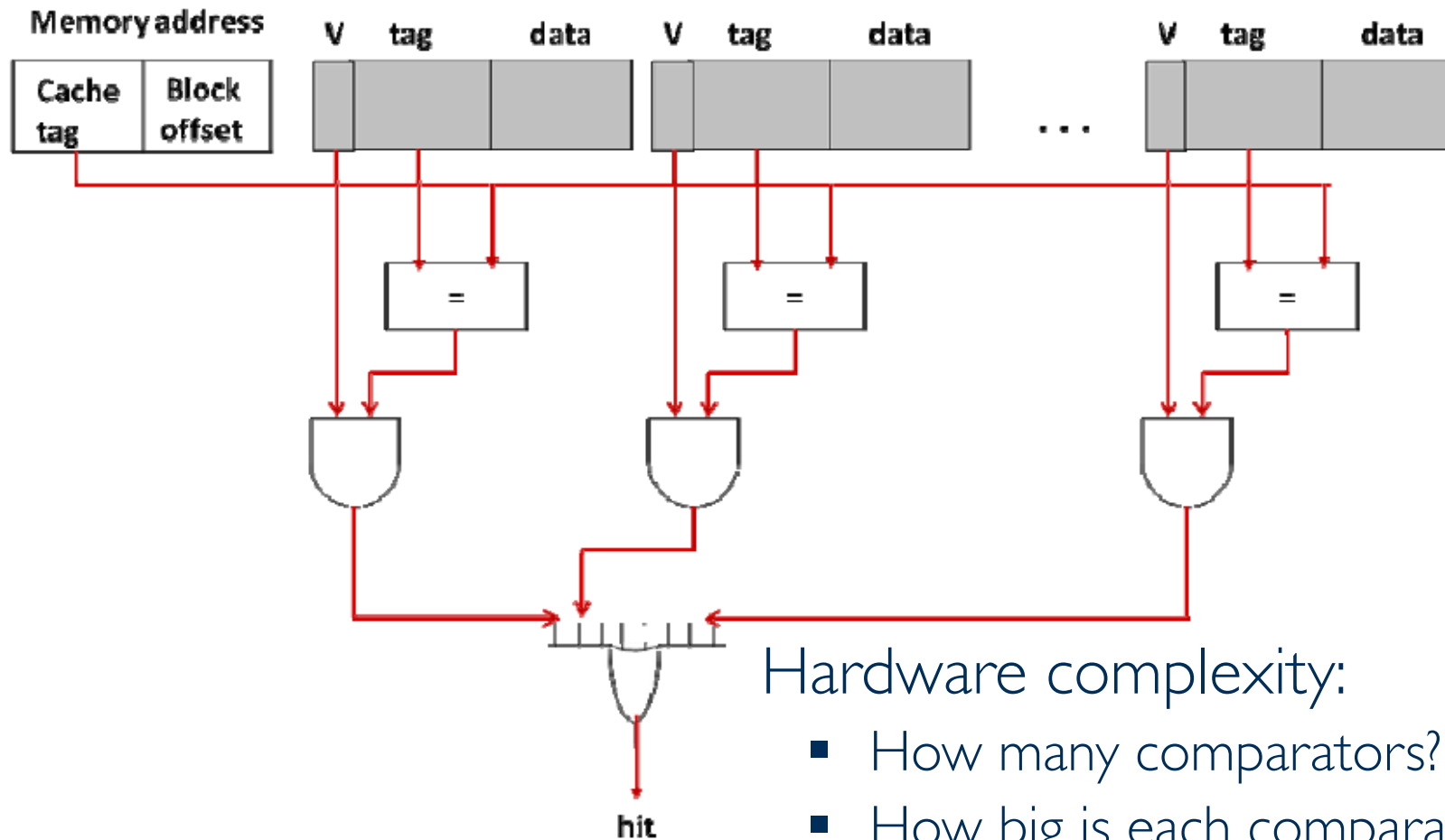
Address interpretation in FA cache

Cache Tag	Index
-----------	-------

- No splitting memory addresses into “index” and “tag”
- It all becomes tag!

Cache Tag

Fully associative cache circuitry



Cache organizations

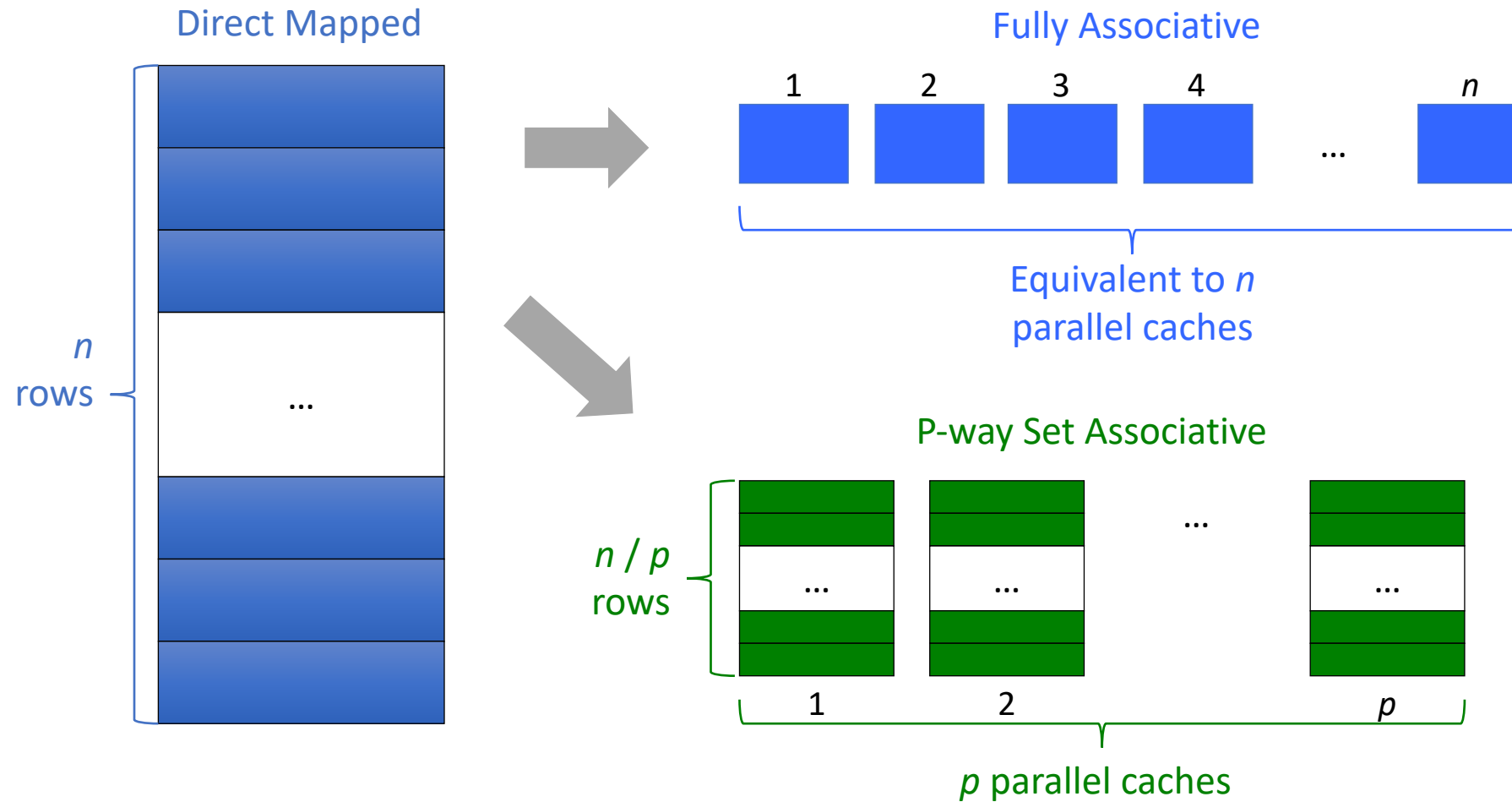
- Fully associative cache →
 - Too much hardware complexity
 - Most flexible



- Direct mapped cache →
 - Least hardware complexity
 - Least flexible

- Can we do better? Is there a compromise?
- Yes! It's called a set-associative cache
- Direct-mapped and fully-associative caches are cases of a set-associative cache on opposite ends of the spectrum!

Generalization?



Iso-capacity cache comparison

	V	Tag	data
0			
1			
2			
3			
4			
5			
6			
7			
8			
9			
10			
11			
12			
13			
14			
15			

(a) Direct-mapped cache

	V	Tag	data
0			
1			
2			
3			
4			
5			
6			
7			

	V	Tag	data
0			
1			
2			
3			
4			
5			
6			
7			

(b) 2-way set associative

Memory block in one of two places

Memory block in exactly one place

	V	Tag	data
0			
1			
2			
3			

	V	Tag	data
0			
1			
2			
3			

	V	Tag	data
0			
1			
2			
3			

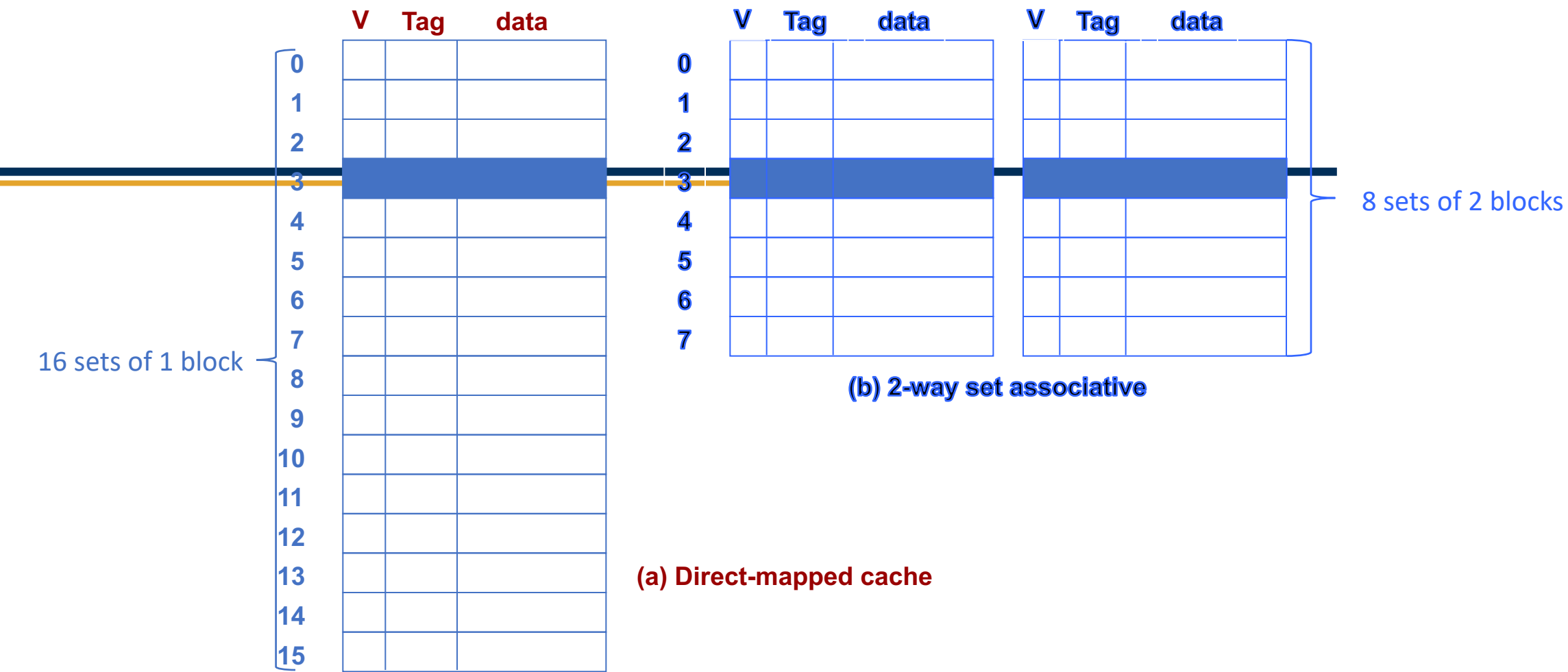
	V	Tag	data
0			
1			
2			
3			

(c) 4-way set associative

Memory block in one of four places

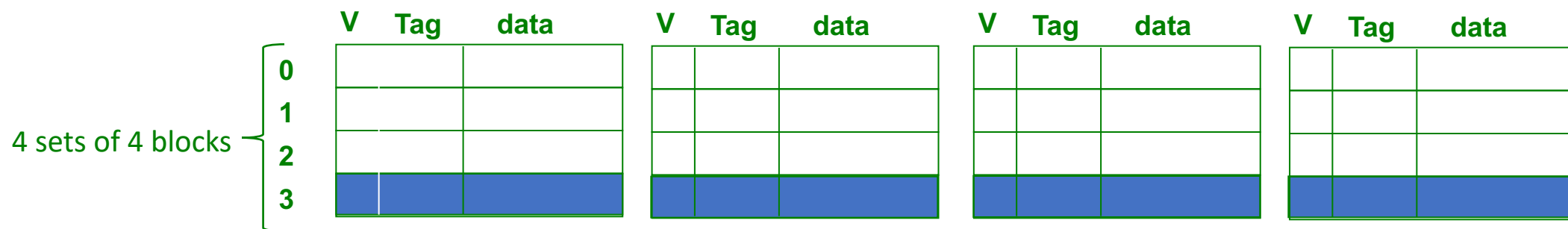
Terminology clarification

- Unfortunate but...
 - Four different ways of saying the same thing
 - Cache **line**
 - Cache **block**
 - Cache **entry**
 - Cache **element**
 - All mean the same thing...the basic unit of data transferred into/out of the cache at a time
 - The textbook has a couple of typos in which it erroneously implies cache set is also synonymous to cache block
- A cache **set** is a “row” in the cache. The number of blocks per set is determined by the type of the cache
 - Direct mapped: **n** sets, **1** block
 - P-way set associative: **n/p** sets, **p** blocks
 - Fully associative: **1** set, **n** blocks



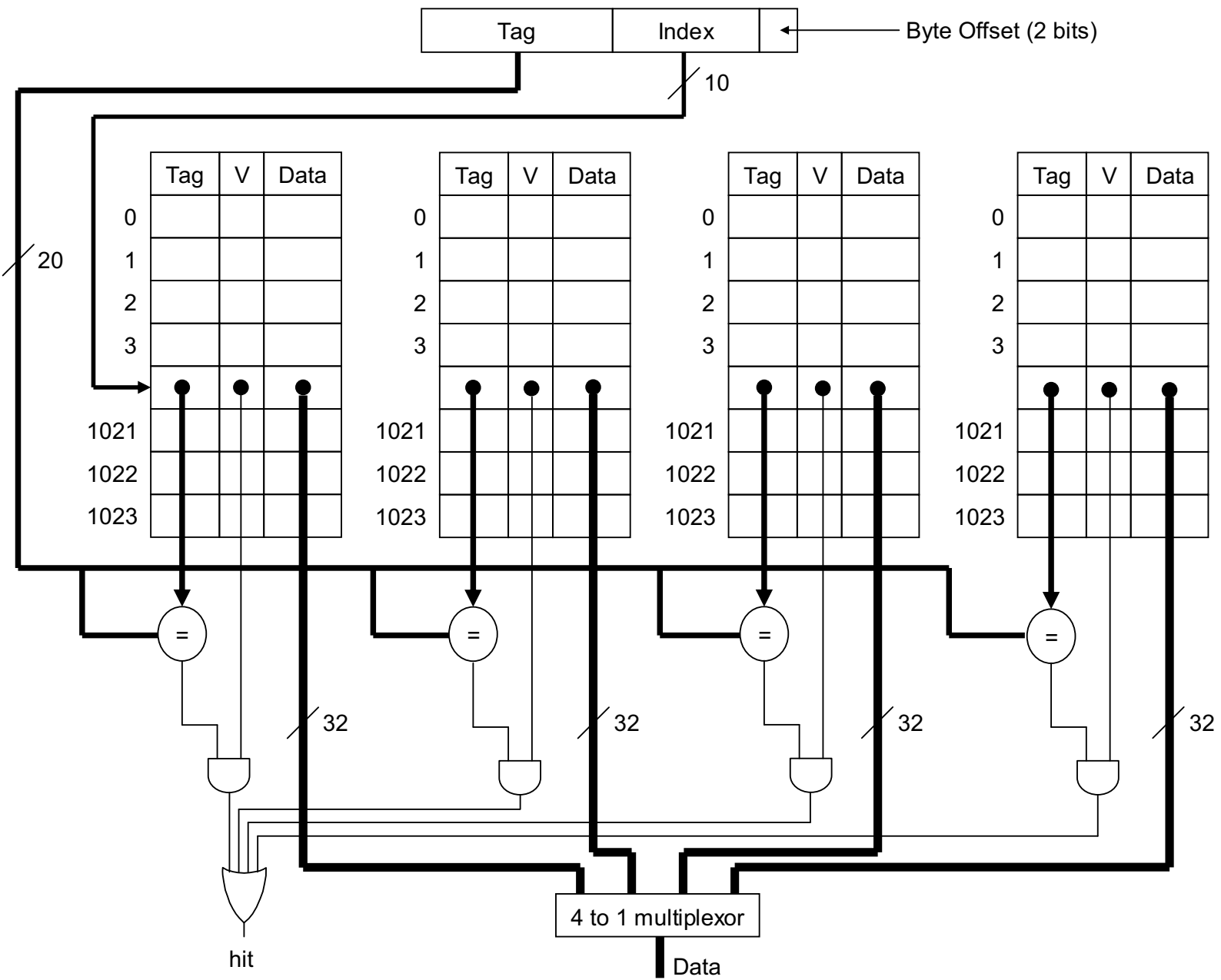
(a) Direct-mapped cache

(b) 2-way set associative



(c) 4-way set associative

4-way SA cache organization

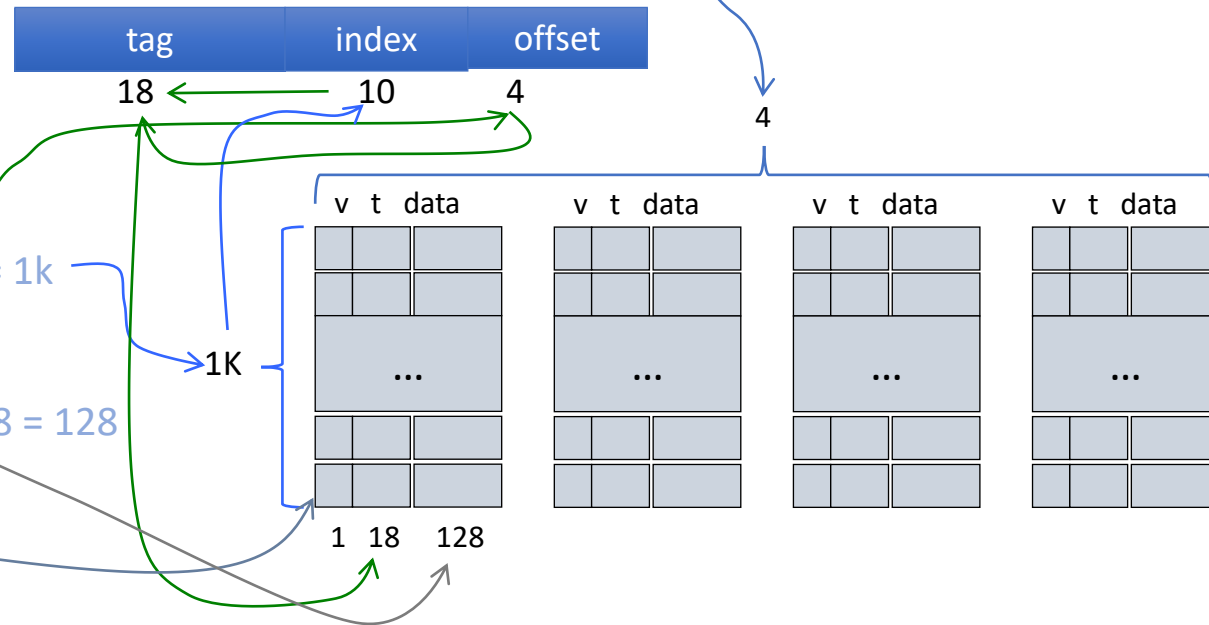


Example: 4-way set associative cache

- 4-way set associative cache
- 32-bit memory, byte-addressable
- Cache size of 64 Kbytes.
- Cache block size is 16 bytes.
- Write-through policy
- One valid bit per block.

Compute the total amount of storage for implementing the cache (i.e. actual data plus the metadata)

$$(1+18+128) * 1K * 4 / 8 = 75,264 \text{ bytes}$$



Hardware complexity?

4 18-bit comparators



In a fully associative cache ...

...with 64K bytes of data, 64 bytes / block and a t -bit tag

- 31% A. I just want the participation credit
- 24% B. There are four t -bit tag comparators
- 17% C. There are 64 t -bit tag comparators
- 28% D. There are 1k t -bit tag comparators
- 0% E. There is one t -bit tag comparator for the entire cache

FA caches have one comparator for each block/line/entry in the cache.
That means $64K/64 = 1K$ is the number of comparators.



In a 4-way set associative cache, ...

...with 64K bytes of data, 64 bytes / block, with a t -bit tag

- 43% A. I just want the participation credit
- 50% B. There are four t -bit tag comparators
- 0% C. There are 64 t -bit tag comparators
- 7% D. There are 1k t -bit tag comparators
- 0% E. There is one t -bit tag comparator for the entire cache

What about cache replacement policy?

	C0			C1			LRU
	V	Tag	data	V	Tag	data	
0							
1							
2							
3							
4							
5							
6							
7							

- What kind of cache is this? 2-way set associative
- How many LRU bits do we need? Just one.
- What happens on every memory access? Set LRU to 0/1 if we read from/store in C0/C1
- So what do we do with a 4-way set associative cache?

LRU replacement in a 4-way cache

	C0			C1			C2			C3			LRU
	V	Tag	data	V	Tag	data	V	Tag	data	V	Tag	data	
0													c1 -> c3 -> c0 -> c2
1													c0 -> c2 -> c1 -> c3
2													c2 -> c3 -> c0 -> c1
3													c3 -> c2 -> c1 -> c0

- Do we need a state machine for each cache line?
- Using as many state machines as the number of rows in the cache is a lot of hardware
- Each state machine → 4! States → 5 bit state register

What happens on a context switch?

- TLB? Flush user portion
- Cache? Nothing! We're using physical addresses



One
caveat

Note: If the OS brings in a page from disk directly to a physical page frame and bypasses the cache, it must flush any cache locations for the previous contents. This isn't a context switch issue, it's an I/O issue: if I/O bypasses the cache, then any cache entries referencing the swapped out page are definitely invalid.



On a process context switch

- 30% A. I just want the participation credit
- 13% B. The cache entries for the process being suspended are flushed.
- 10% C. The cache and TLB entries for the process being suspended are flushed
- 37% D. The TLB entries for the process being suspended are flushed
- 10% E. None of the above

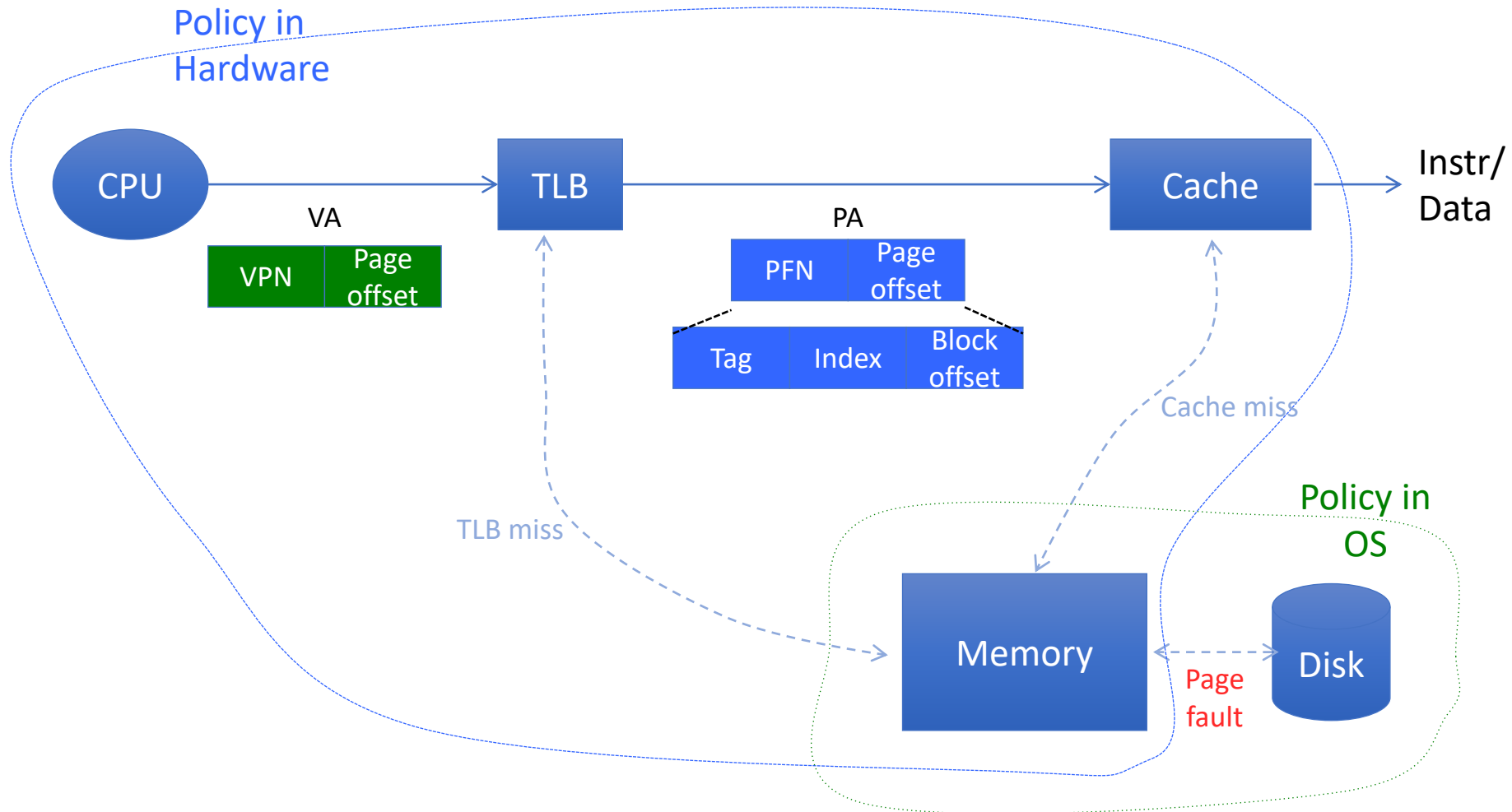
The caches we've studied so far hold physical addresses (address translation has already occurred before the cache gets involved), then no flush is needed on context switch.
If the cache holds virtual addresses, then flushing cache entries from the page is required.



When a page is evicted from a page frame in memory

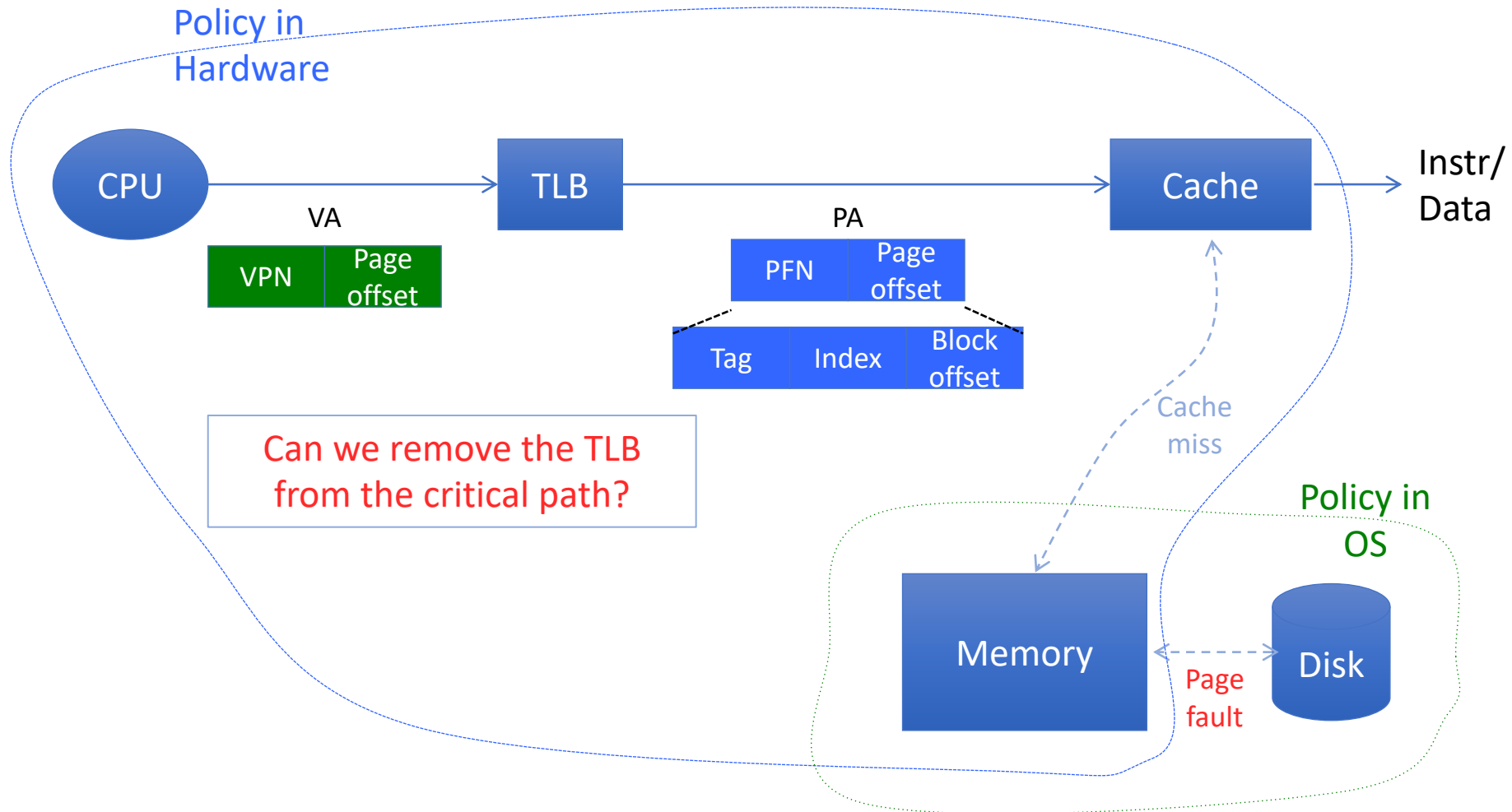
- A. I just want the participation credit
- B. The cache entries for the evicted page are flushed.
- C. The cache and TLB entries for the evicted page are flushed
- D. The TLB entries for the evicted page are flushed
- E. None of the above

Putting cache and VM together

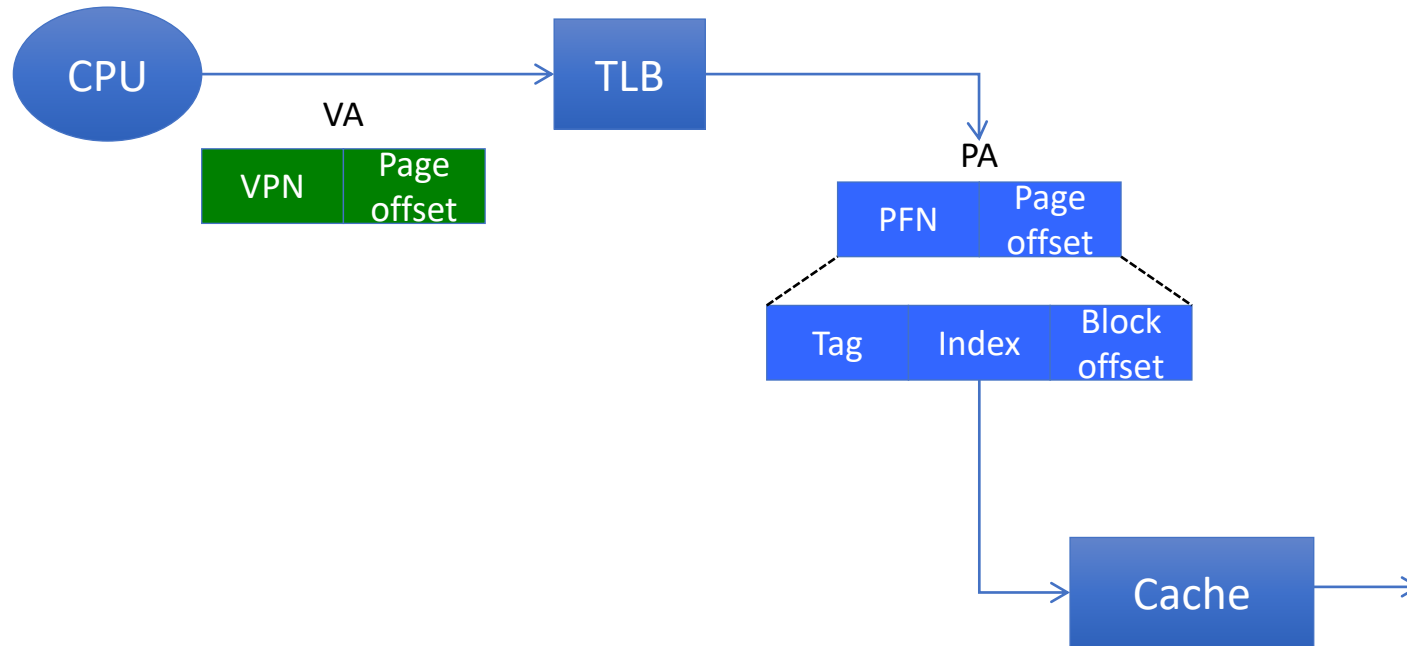


Adding some speed

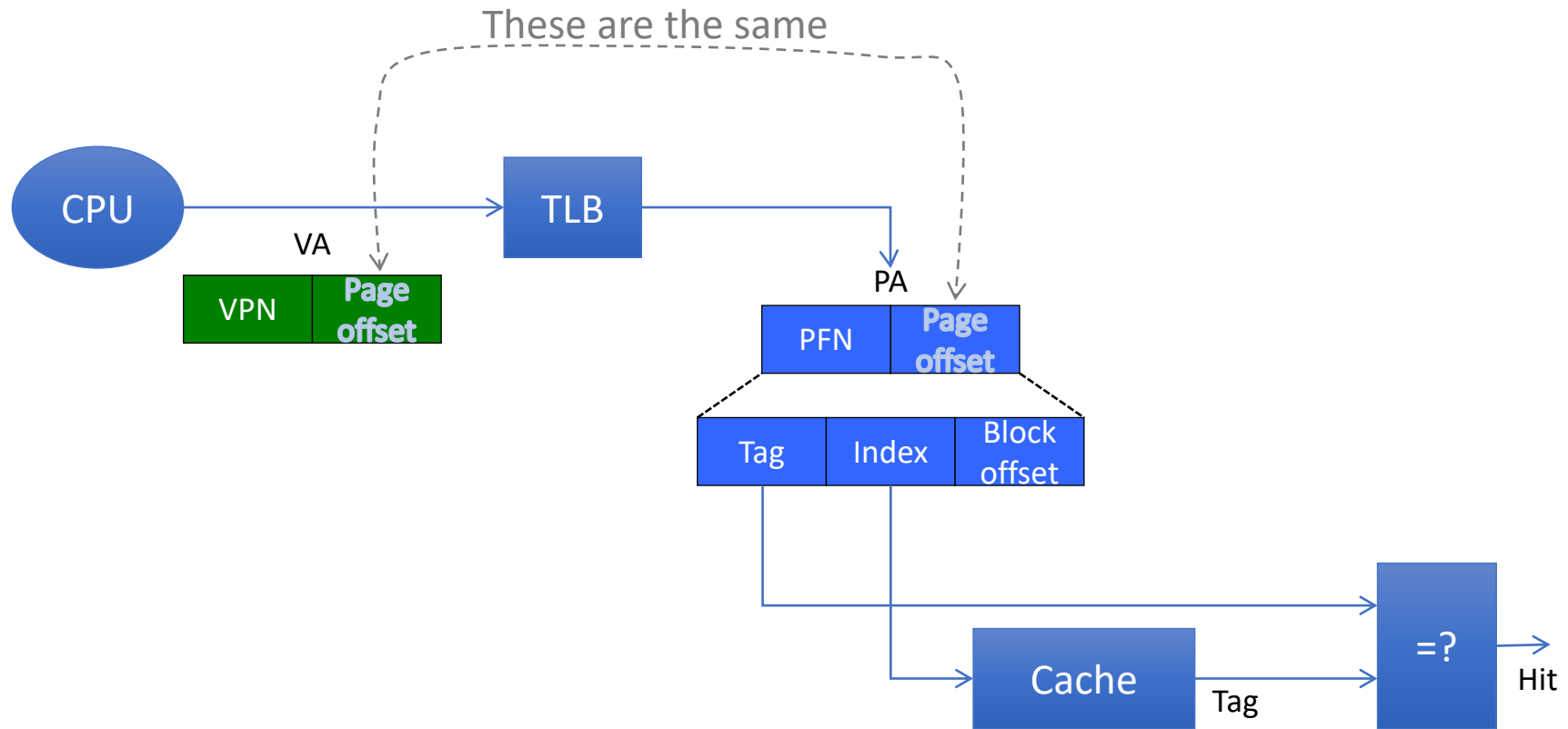
- TLB access is on the critical path of cache access → increased clock cycle time



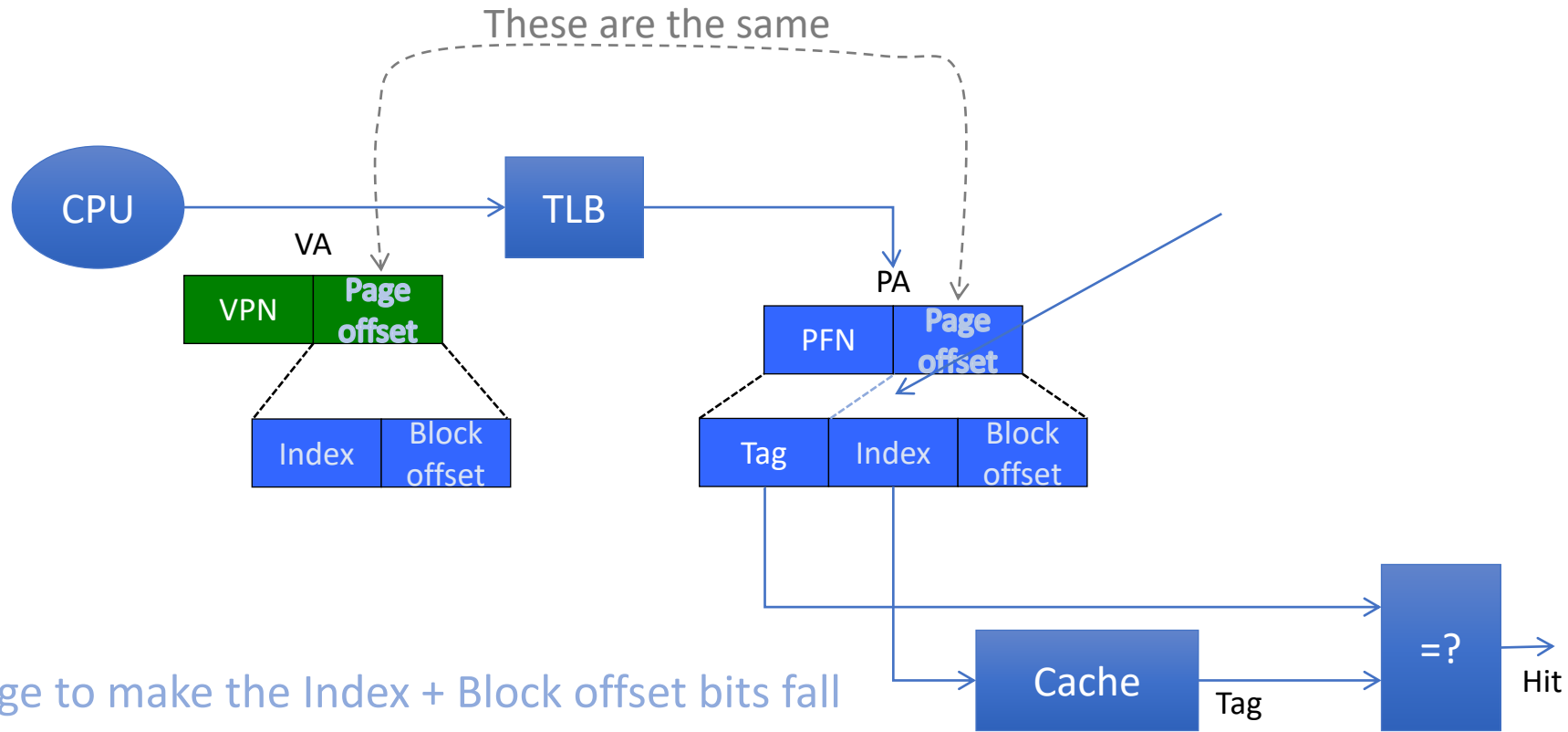
Recall how TLB and Cache work together



Recall



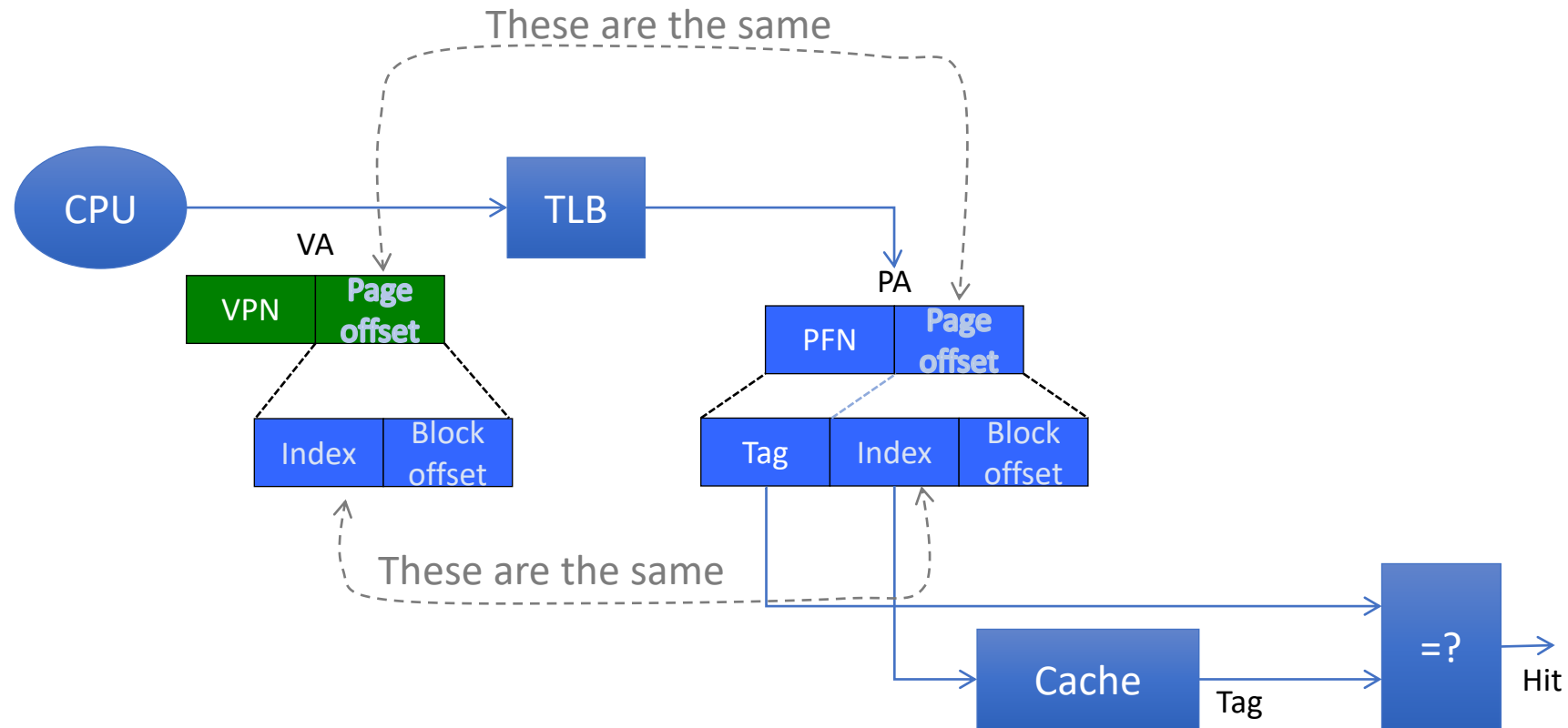
Make the cache index \leq page offset?



What if we arrange to make the Index + Block offset bits fall within Page offset?

Then we can get the cache index from the VA to start the cache read without waiting for the TLB!

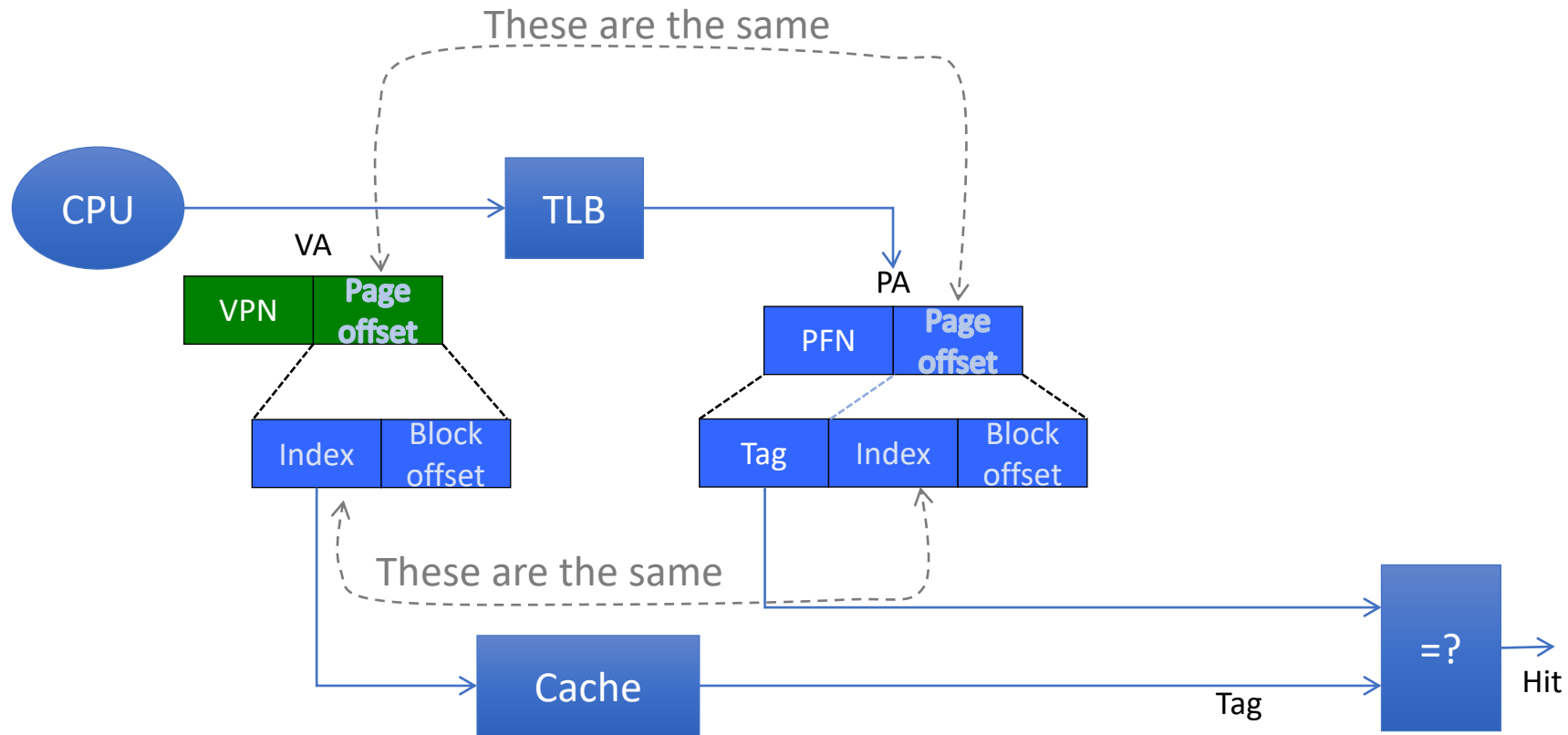
Now Index is the same for VA and PA!



What if we arrange to make the Index + Block offset bits fall within Page offset?

Then we can get the cache index from the VA to start the cache read without waiting for the TLB!

Cache and TLB access can start in parallel!



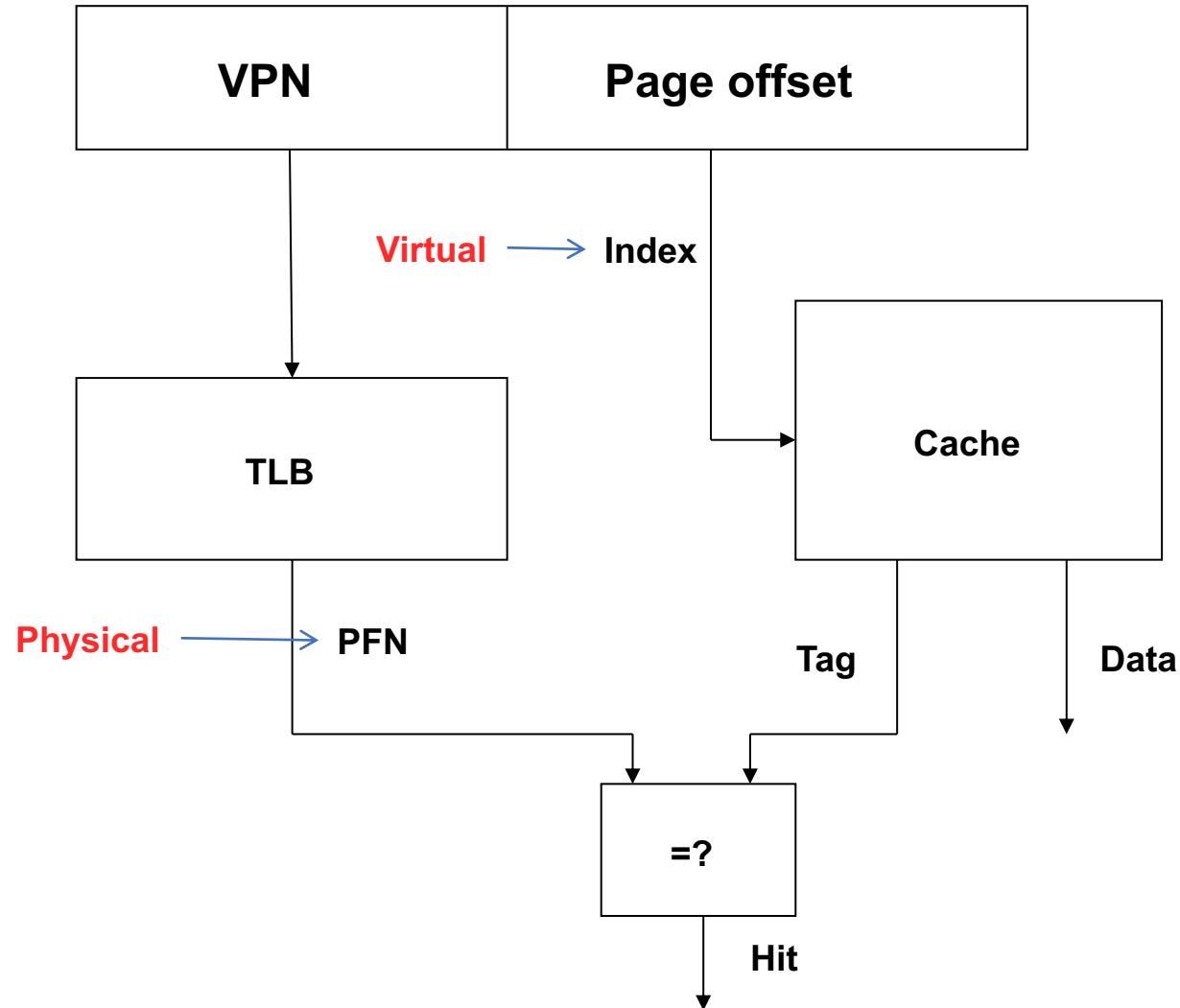
What if we arrange to make the Index + Block offset bits fall within Page offset?

Then we can get the cache index from the VA to start the cache read without waiting for the TLB!

Virtually indexed physically tagged (VIPT) cache

How to get TLB out of the critical path?

TLB & cache access in parallel





In a virtually indexed physically tagged cache

32% A. I just want the participation credit

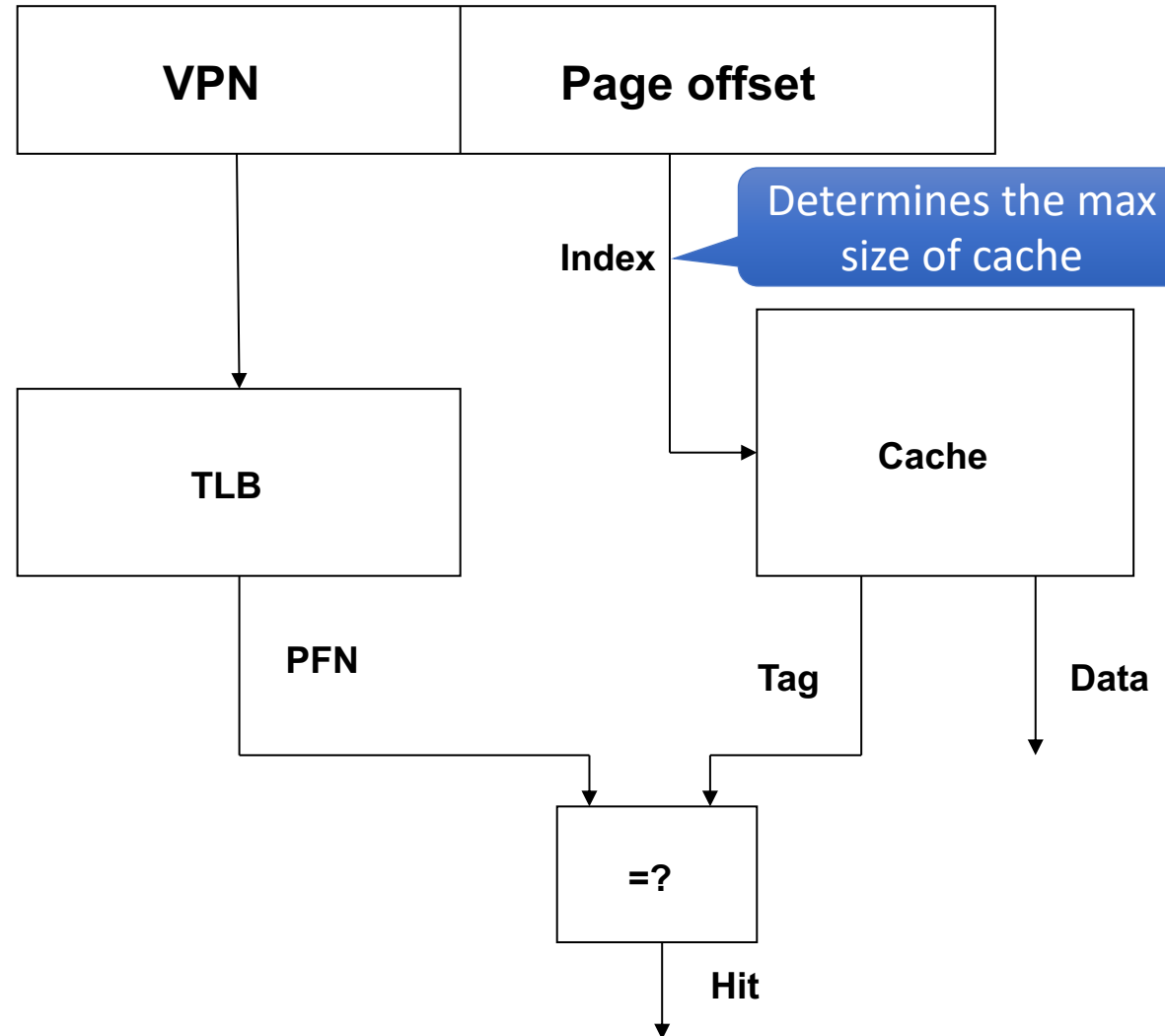
61% B. The TLB and cache are accessed in parallel

7% C. The TLB and cache are accessed sequentially

0% D. The TLB and cache are accessed at random

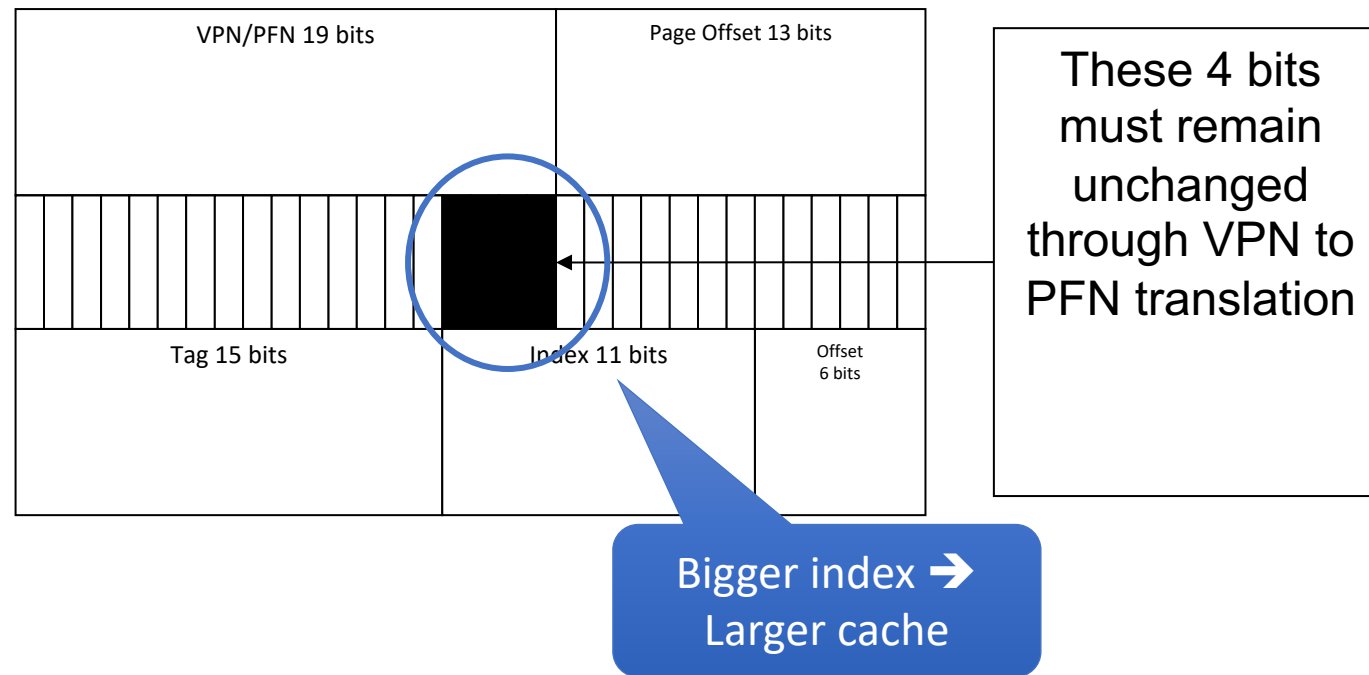
0% E. None of the above

Is there a limitation in VIPT cache size?



Page coloring example

OS guarantees some bits of VPN will remain unchanged

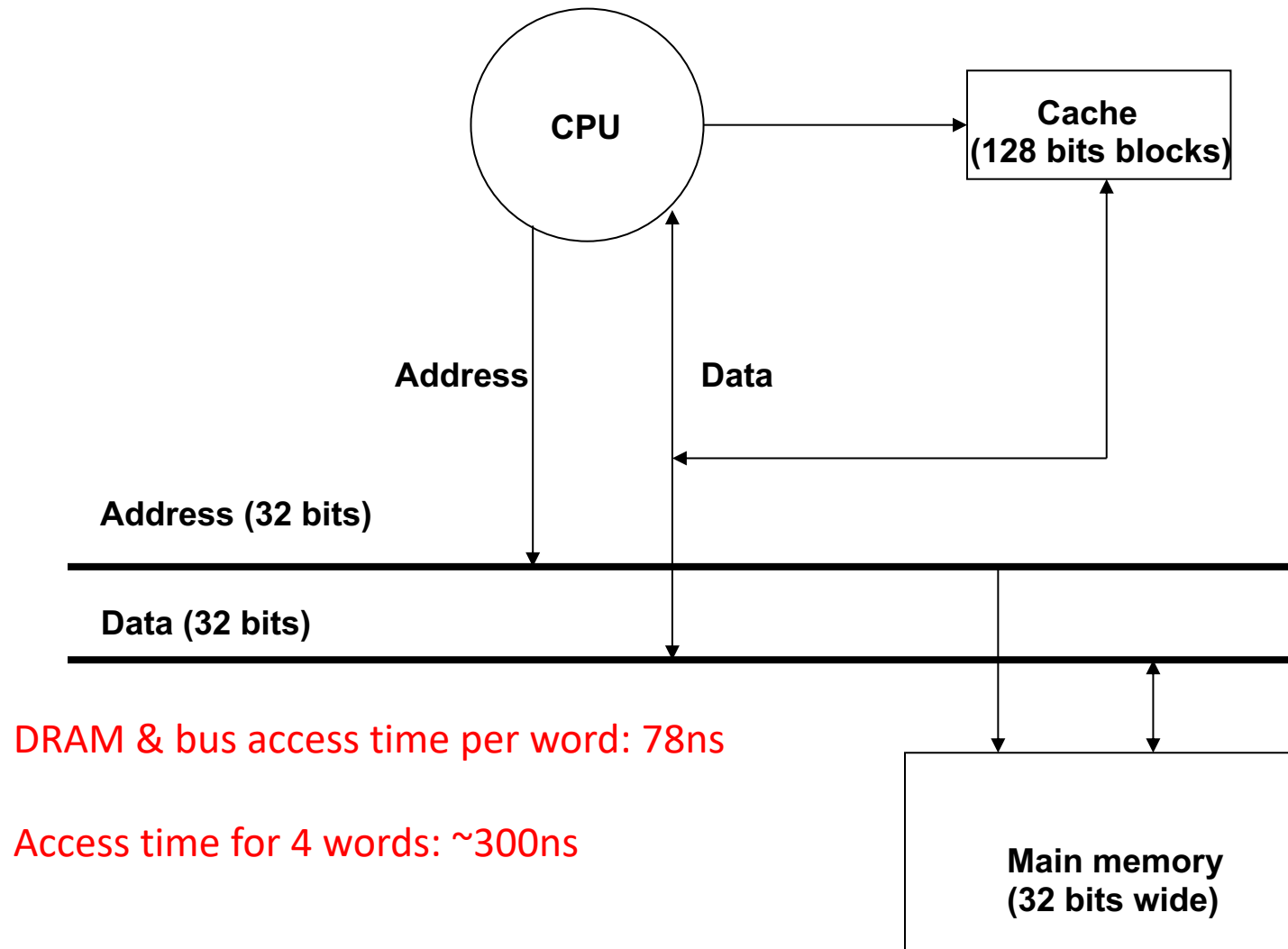


Work out Example 10 in the book on your own and check the solution

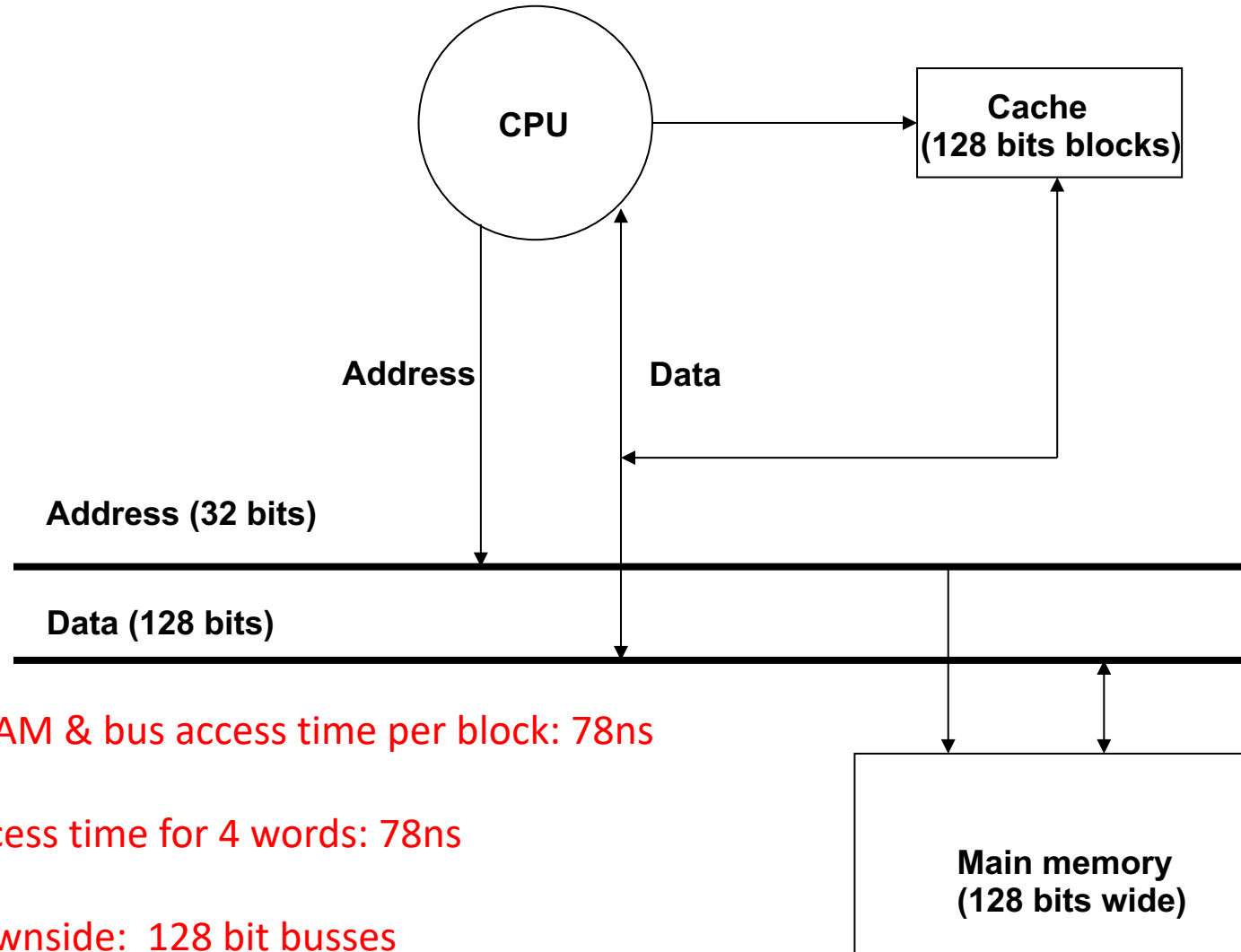
Page coloring impacts

- Require the low bits of the VPN and PFN to be identical (4 low order bits in this example)
 - This allows the use of these bits as part of either the virtual or physical address, just like we use the offset
- What does that impact?
 - A virtual page can only occupy a subset of page frames in which the low bits of the VPN match the low bits of the PFN (i.e., limit the fully associative nature of Virtual Memory Management).
 - We refer to this as **page coloring** which means the page replacement algorithm must keep track of the “color” of the VPNs and PFNs (namely those low bits) to ensure virtual pages are only loaded into like-colored page frames.
 - It could make it harder to fit a working set into physical memory, but it’s often workable because processes tend to use contiguous pages so the VPNs of the pages are spread evenly among the colors
 - In this example using the low 4 bits of the VPN/PFN, we get 16 different “colors” of page/page frame. So a page with a VPN ending in 0010_2 can only be placed in a page frame with a number that ends in 0010_2

Simple memory system



Memory system matching cache block size



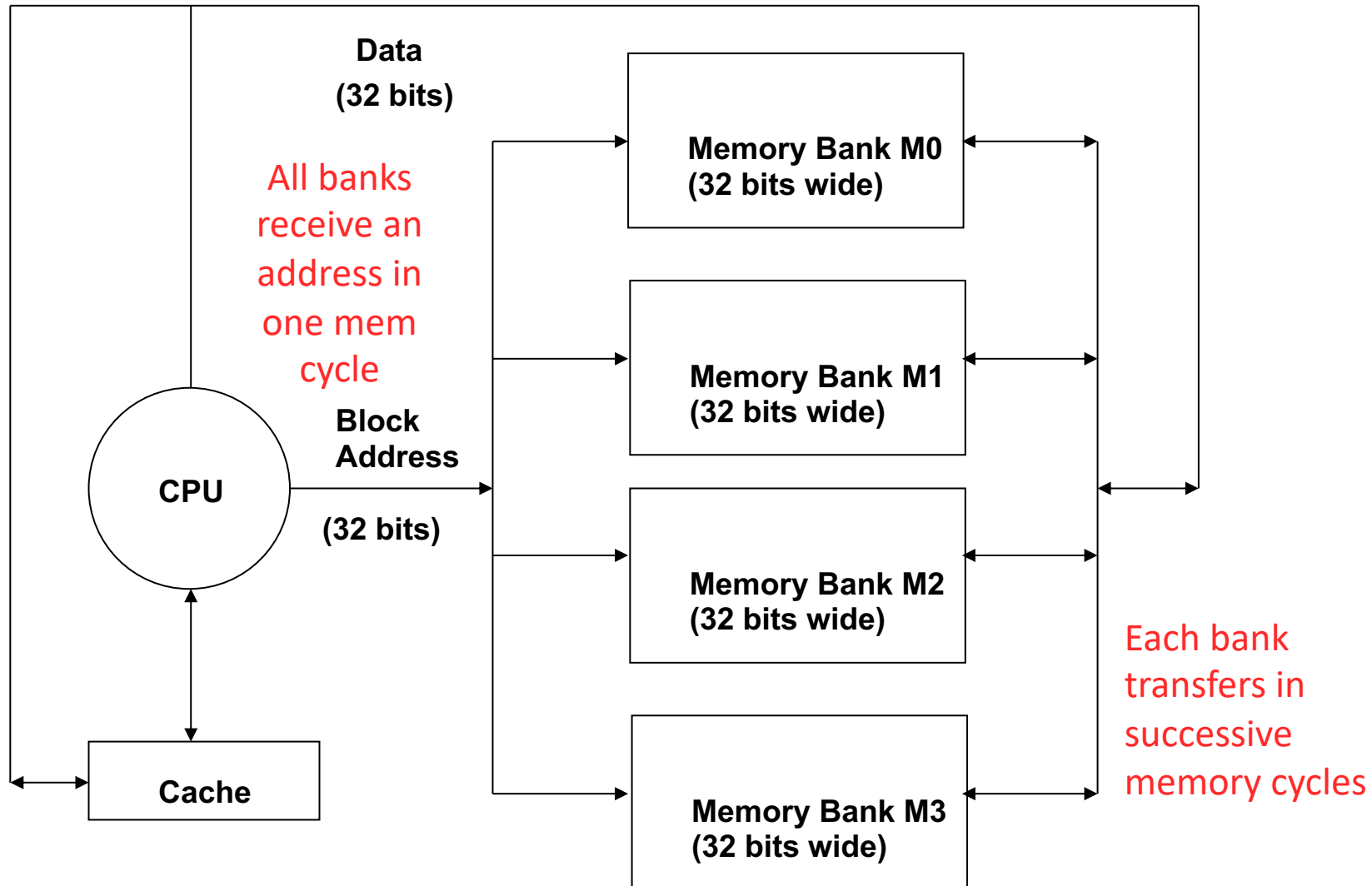
DRAM & bus access time per block: 78ns

Access time for 4 words: 78ns

Downside: 128 bit busses

Interleaved memory

Transfer time \ll memory access time

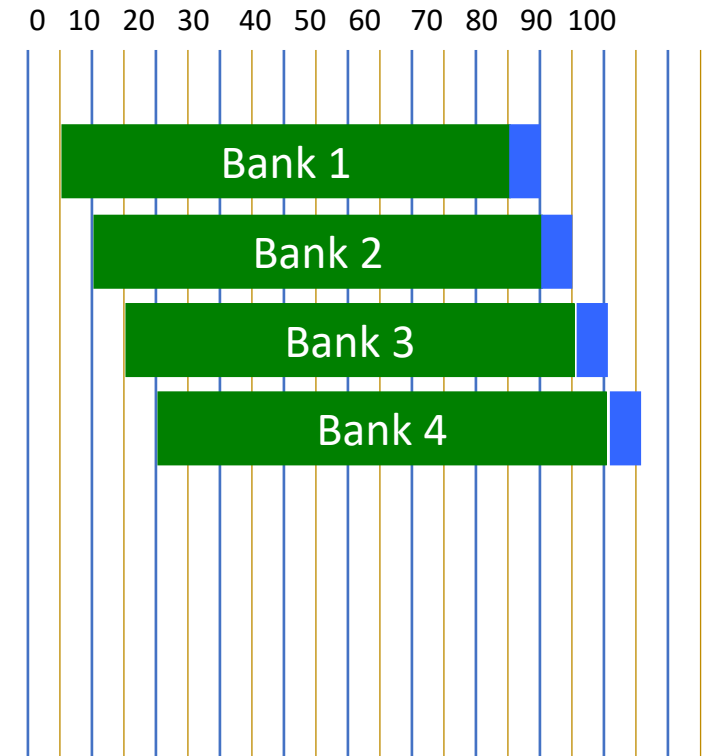


Example

- 4-way interleaved memory
- DRAM access time: 80 cycles.
- Memory bus cycle time: 5 cycles
- Compute the block transfer time for a block size of 4 words.
Assume all 4 words are first retrieved from the DRAM before the data transfer to the CPU.

Example solution sketch

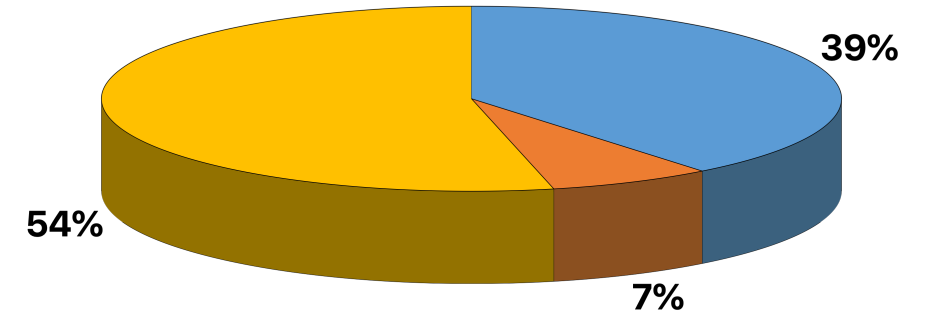
- Start memory fetches at 5, 10, 15, 20 cycles
- Bank 4 memory fetch finishes at 105 cycles
- Data transfers to cache complete at 90, 95, 100, 105 cycles thanks to bank-level parallelism










Interleaved memory is useful because

- A. I just want the participation credit
- B. It increases the cache block size
- C. It reduces the cache block size
- D. It allows concurrent fetches of data blocks from memory without requiring wider busses
- E. It decreases the size of the cache index



-  I just want the participation credit
-  It increases the cache block size
-  It reduces the cache block size
-  It allows concurrent fetches of data blocks from...
-  It decreases the size of the cache index

Cache hierarchy: Relative sizes & latencies, ca. 2017

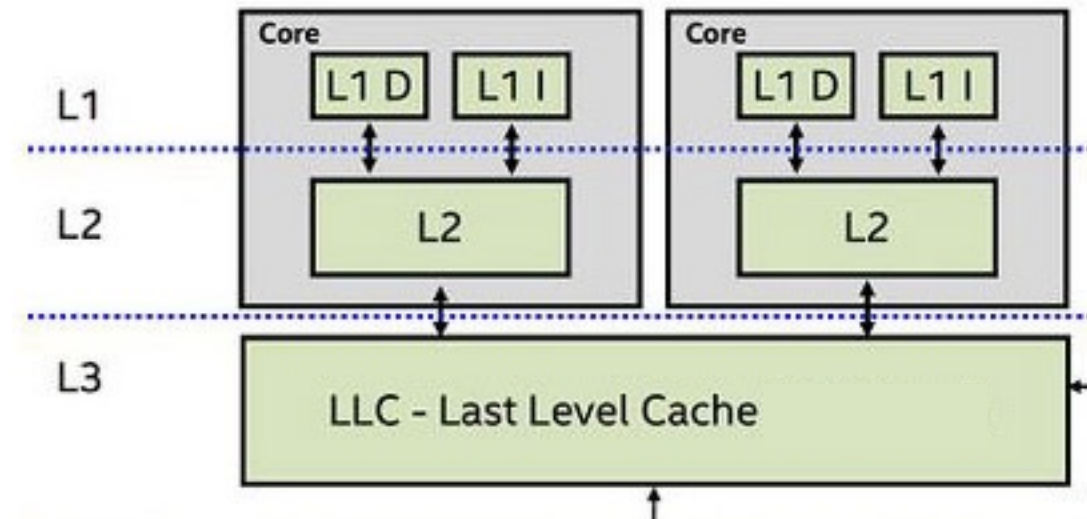
Skylake-SP server processors

Sizes:

- L1 Data cache = 32 KB private per core, 64 B/line, 8-WAY.
- L1 Instruction cache = 32 KB private per core, 64 B/line, 8-WAY.
- L2 cache = 1MB private per core, 64 B/line, 16-WAY
- L3 cache = shared 1.375MB per core, 64 B/line, 11-WAY

Latencies (@ 3.7GHz frequency):

- L1 Data Cache = 4 cycles
- L2 Cache = 12 cycles
- L3 Cache = 44 cycles
- DRAM (memory) Latency = 44 cycles + 51 ns



Summary of Chapter 9 terminology

Category	Vocabulary	Details
Principle of locality (Section 9.2)	Spatial	Access to contiguous memory locations
	Temporal	Reuse of memory locations already accessed
Cache organization	Direct-mapped	One-to-one mapping (Section 9.6)
	Fully associative	One-to-any mapping (Section 9.12.1)
	Set associative	One-to-many mapping (Section 9.12.2)
Cache reading/writing (Section 9.8)	Read hit/Write hit	Memory location being accessed by the CPU is present in the cache
	Read miss/Write miss	Memory location being accessed by the CPU is not present in the cache
Cache write policy (Section 9.8)	Write through	CPU writes to cache and memory
	Write back	CPU only writes to cache; memory updated on replacement
Cache parameters	Total cache size (S)	Total data size of cache in bytes
	Block Size (B)	Size of contiguous data in one data block
	Degree of associativity (p)	Number of homes a given memory block can reside in a cache
	Number of cache lines (L)	S/pB
	Cache access time	Time in CPU clock cycles to check hit/miss in cache
	Unit of CPU access	Size of data exchange between CPU and cache
	Unit of memory transfer	Size of data exchange between cache and memory
	Miss penalty	Time in CPU clock cycles to handle a cache miss
Memory address interpretation	Index (n)	$\log_2 L$ bits, used to look up a particular cache line
	Block offset (b)	$\log_2 B$ bits, used to select a specific byte within a block
	Tag (t)	$a - (n+b)$ bits, where a is number of bits in memory address; used for matching with tag stored in the cache

Summary of Chapter 9 terminology

Category	Vocabulary	Details
Cache entry/cache block/cache line	Valid bit	Signifies data block is valid
	Dirty bits	For write-back, signifies if the data block is more up to date than memory
	Tag	Used for tag matching with memory address for hit/miss
	Data	Actual data block
Performance metrics	Hit rate (h)	Percentage of CPU accesses served from the cache
	Miss rate (m)	$1 - h$
	Avg. Memory stall	$\text{Misses-per-instruction}_{\text{Avg}} * \text{miss-penalty}_{\text{Avg}}$
	Effective memory access time (AMAT _i) at level i	$\text{AMAT}_i = T_i + m_i * \text{AMAT}_{i+1}$
	Effective CPI	$\text{CPI}_{\text{Avg}} + \text{Memory-stalls}_{\text{Avg}}$
Types of misses	Compulsory miss	Memory location accessed for the first time by CPU
	Conflict miss	Miss incurred due to limited associativity even though the cache is not full
	Capacity miss	Miss incurred when the cache is full
Replacement policy	FIFO	First in first out
	LRU	Least recently used
Memory technologies	SRAM	Static RAM with each bit realized using a flip flop
	DRAM	Dynamic RAM with each bit realized using a capacitive charge
Main memory	DRAM access time	DRAM read access time
	DRAM cycle time	DRAM read and refresh time
	Bus cycle time	Data transfer time between CPU and memory
	Simulated interleaving using DRAM	Using page mode bits of DRAM

Pthreads API

- Over the break, when you feel bored on the beaches, it's a good idea to read and familiarize yourselves with pthreads
- Project 4 (released tomorrow) learning outcome is CPU scheduling algorithms
- Project implementation uses parallel programming with pthreads
→ good idea to get a head start over the break

Have a great Spring Break!