

Caches and TLBs

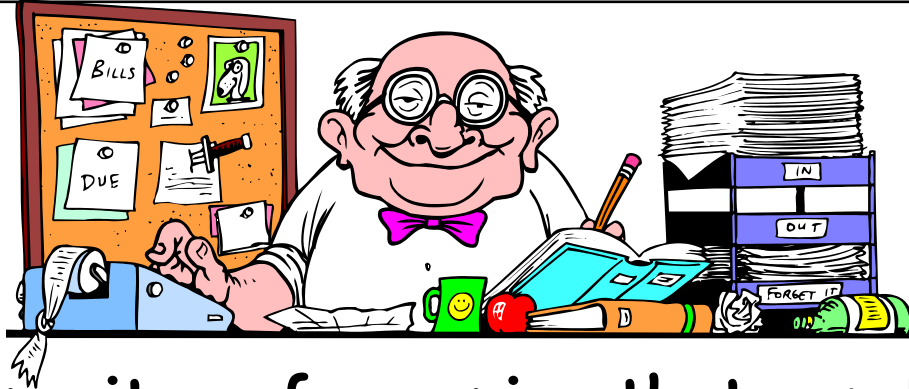
<http://inst.eecs.berkeley.edu/~cs162>

Goals for Today

- Caching and TLBs

Note: Some slides and/or pictures in the following are adapted from slides ©2005 Silberschatz, Galvin, and Gagne

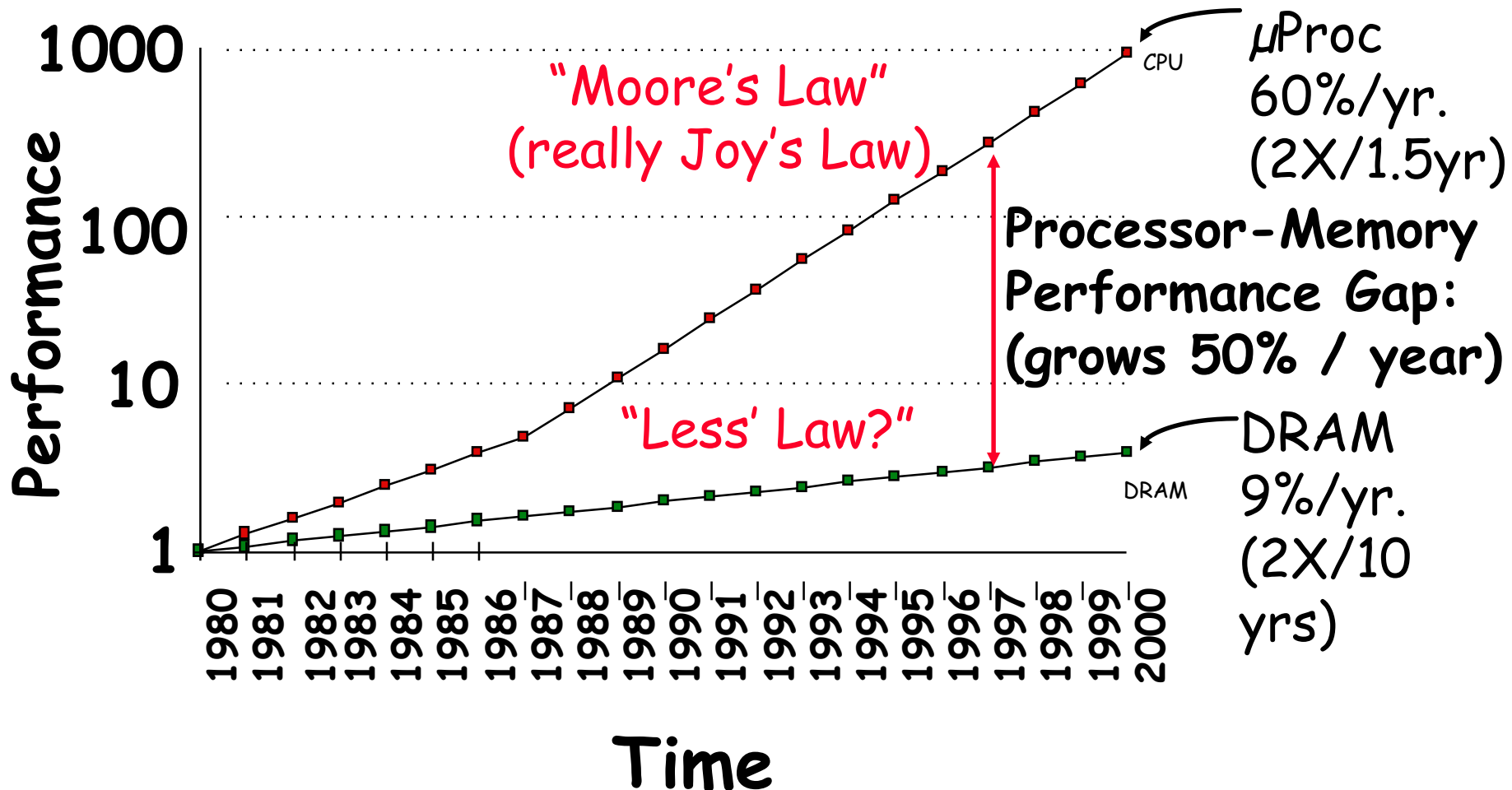
Caching Concept



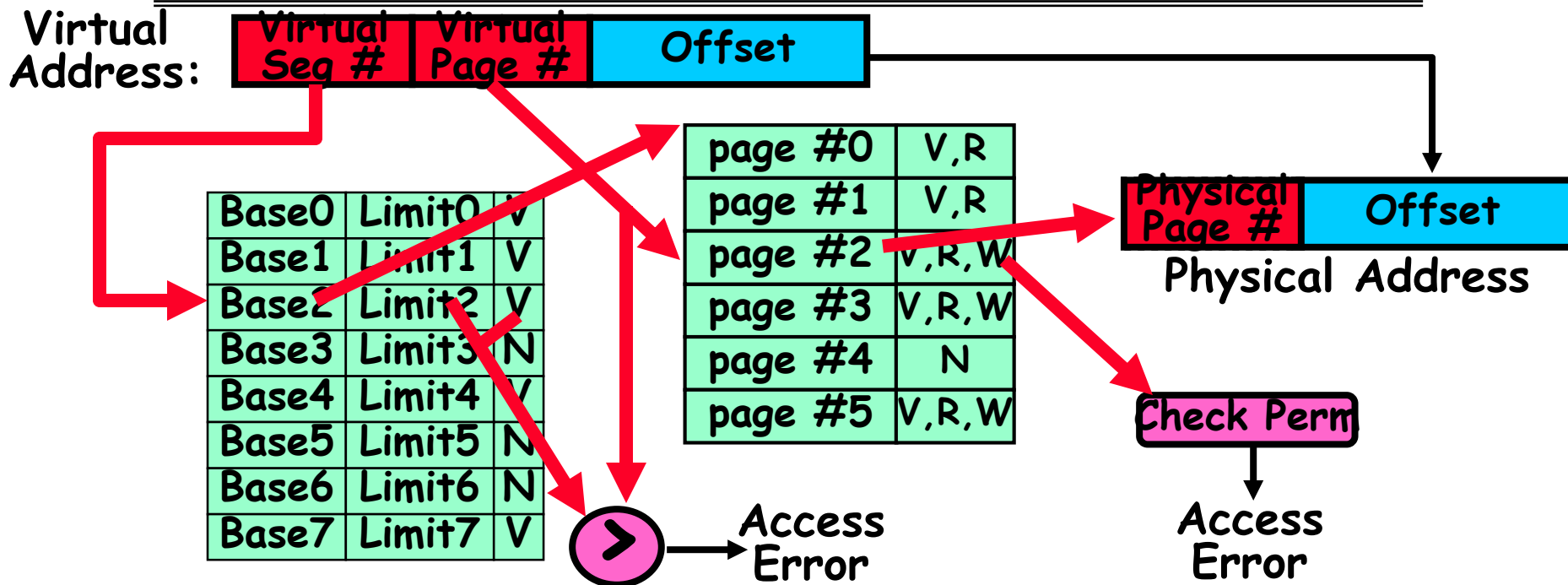
- **Cache**: a repository for copies that can be accessed more quickly than the original
 - Make frequent case fast and infrequent case less dominant
- Caching underlies many of the techniques that are used today to make computers fast
 - Can cache: memory locations, address translations, pages, file blocks, file names, network routes, etc...
- Only good if:
 - Frequent case frequent enough and
 - Infrequent case not too expensive
- Important measure: Average Access time =
 $(\text{Hit Rate} \times \text{Hit Time}) + (\text{Miss Rate} \times \text{Miss Time})$

Why Bother with Caching?

Processor-DRAM Memory Gap (latency)

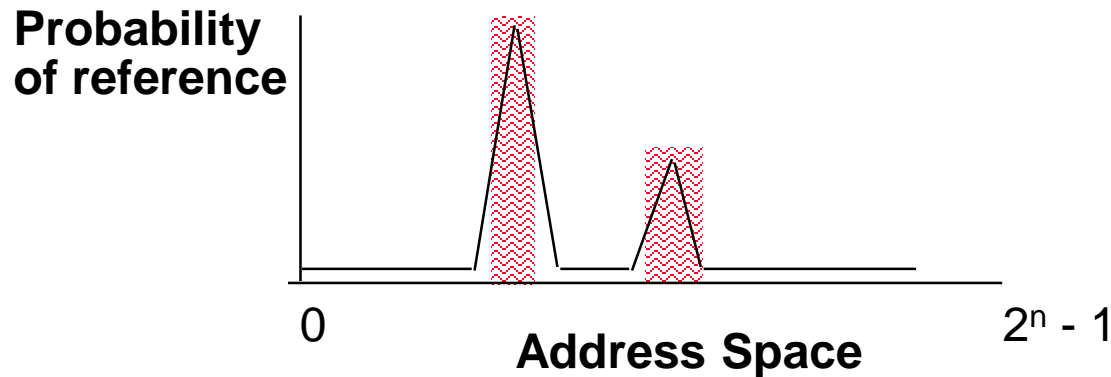



Another Major Reason to Deal with Caching

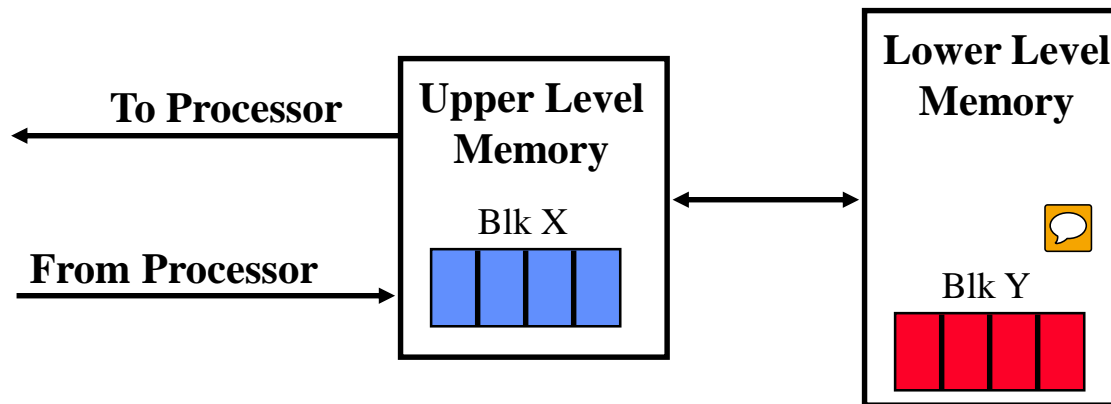


- Cannot afford to translate on every access
 - At least three DRAM accesses per actual DRAM access
 - Or: perhaps I/O if page table partially on disk!
- Even worse: What if we are using caching to make memory access faster than DRAM access???
- Solution? Cache translations!
 - Translation Cache: TLB ("Translation Lookaside Buffer")

Why Does Caching Help? Locality!

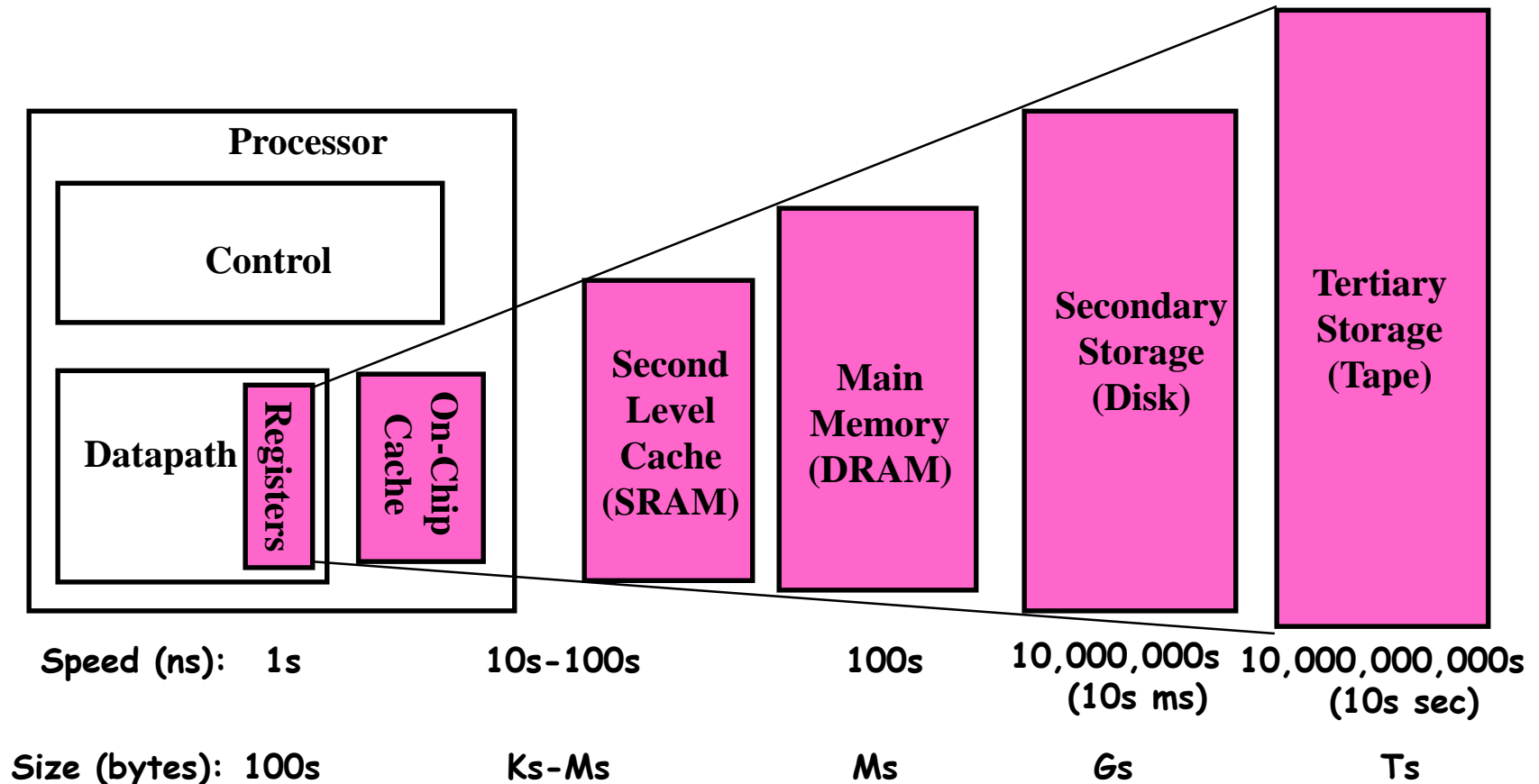


- **Temporal Locality** (Locality in Time): 
 - Keep recently accessed data items closer to processor
- **Spatial Locality** (Locality in Space):
 - Move contiguous blocks to the upper levels



Memory Hierarchy of a Modern Computer System

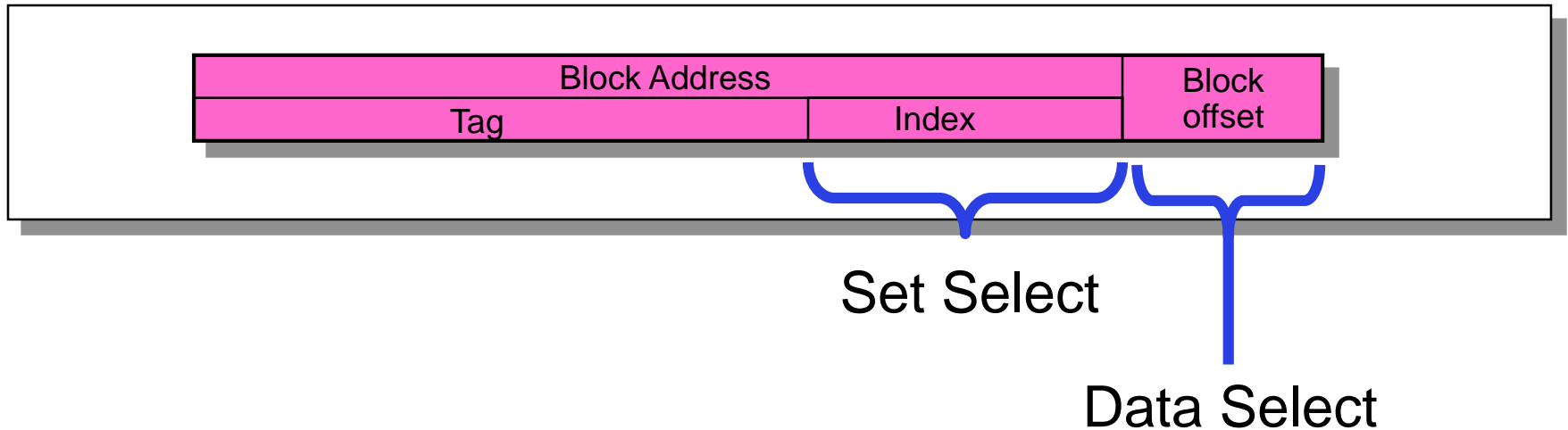
- Take advantage of the principle of locality to:
 - Present as much memory as in the cheapest technology
 - Provide access at speed offered by the fastest technology



A Summary on Sources of Cache Misses

- **Compulsory** (cold start or process migration, first reference): first access to a block
 - “Cold” fact of life: not a whole lot you can do about it
 - Note: If you are going to run “billions” of instruction, Compulsory Misses are insignificant
- **Capacity**:
 - Cache cannot contain all blocks access by the program
 - Solution: increase cache size
- **Conflict** (collision):
 - Multiple memory locations mapped to the same cache location
 - Solution 1: increase cache size
 - Solution 2: increase associativity
- **Coherence** (Invalidation): other process (e.g., I/O) updates memory

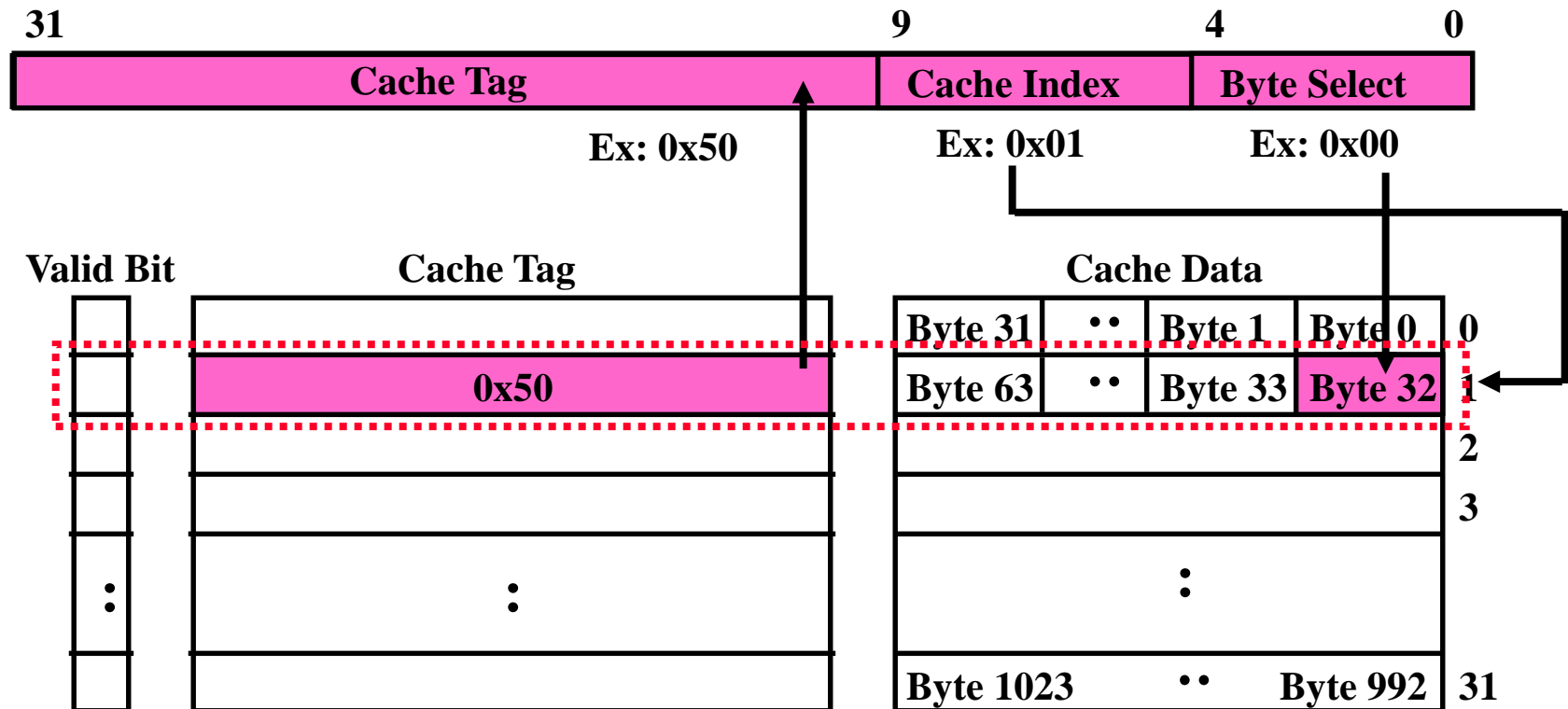
How is a Block found in a Cache?



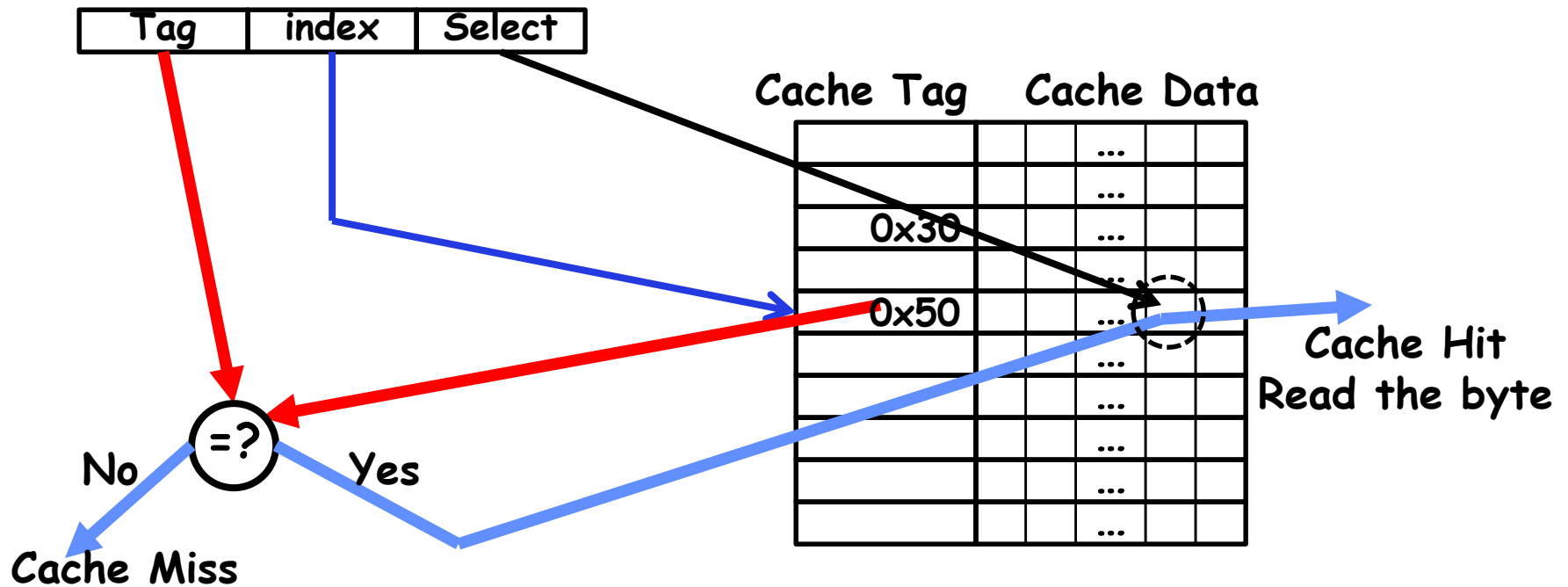
- **Index Used to Lookup Candidates in Cache**
 - Index identifies the set
- **Tag used to identify actual copy**
 - If no candidates match, then declare cache miss
- **Block is minimum quantum of caching**
 - Data select field used to select data within block
 - Many caching applications don't have data select field

Review: Direct Mapped Cache

- **Direct Mapped 2^N byte cache:**
 - The uppermost (32 - N) bits are always the Cache Tag
 - The lowest M bits are the Byte Select (Block Size = 2^M)
- **Example: 1 KB Direct Mapped Cache with 32 B Blocks**
 - Index chooses potential block
 - Tag checked to verify block
 - Byte select chooses byte within block



Direct Mapped Cache: Example



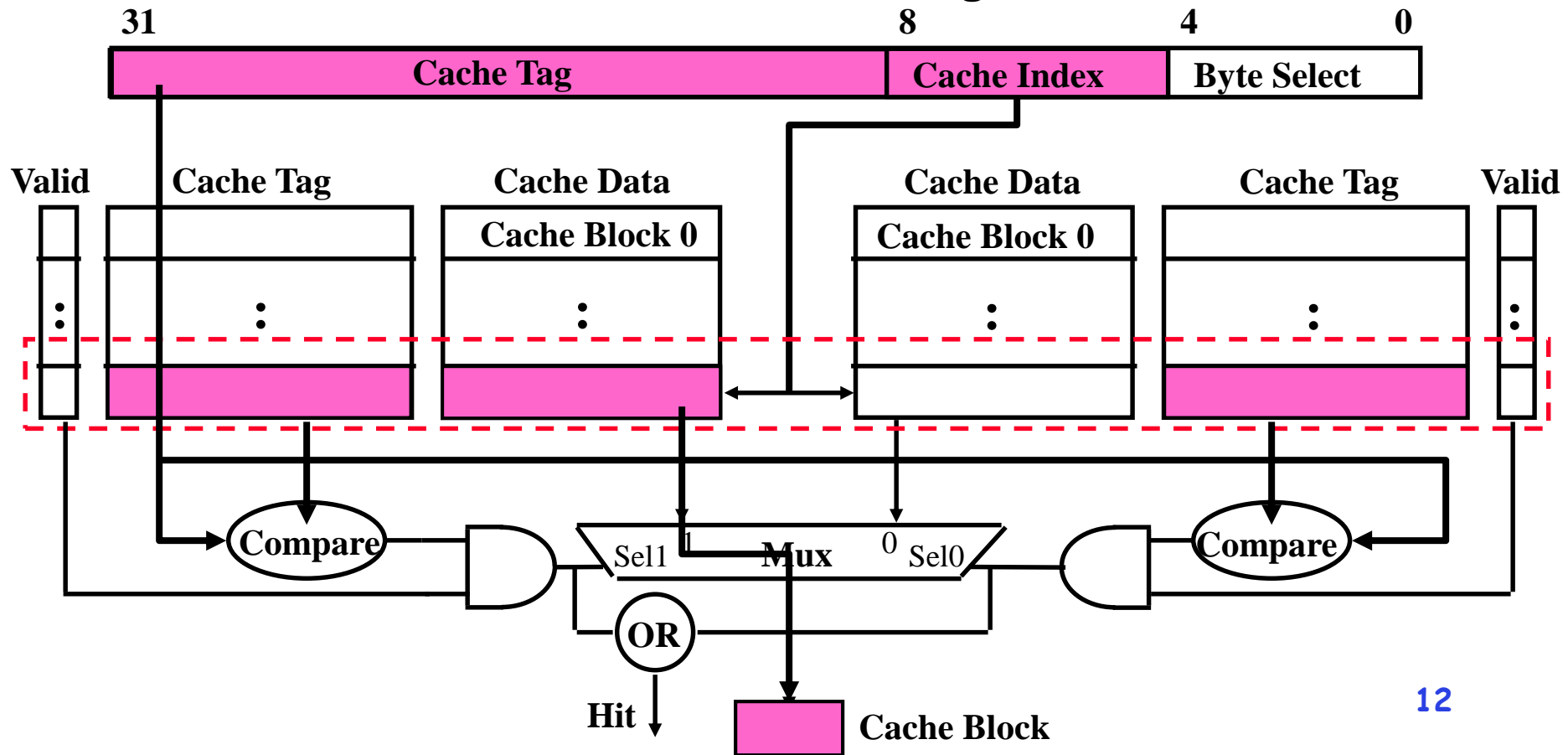
Reference stream

<0x50,0x04,0x00>
<0x30,0x02,0x00>
<0x50,0x04,0x03>
<0x30,0x02,0x02>
...
<0x60,0x04,0x00>
<0x30,0x02,0x02>
<0x60,0x04,0x01>
<0x50,0x04,0x05>



Review: Set Associative Cache

- **N-way set associative**: N entries per Cache Index
 - N direct mapped caches operates in parallel
- **Example: Two-way set associative cache**
 - Cache Index selects a "set" from the cache
 - Two tags in the set are compared to input in parallel
 - Data is selected based on the tag result



Set Associative Cache: Example(2-Way)

Tag	index	Select
-----	-------	--------



Cache Tag Cache Data

		...		
		...		
0x30		...		
		...		
0x50		...		

Cache Tag Cache Data

		...		
		...		
		...		
		...		
0x60		...		

=?

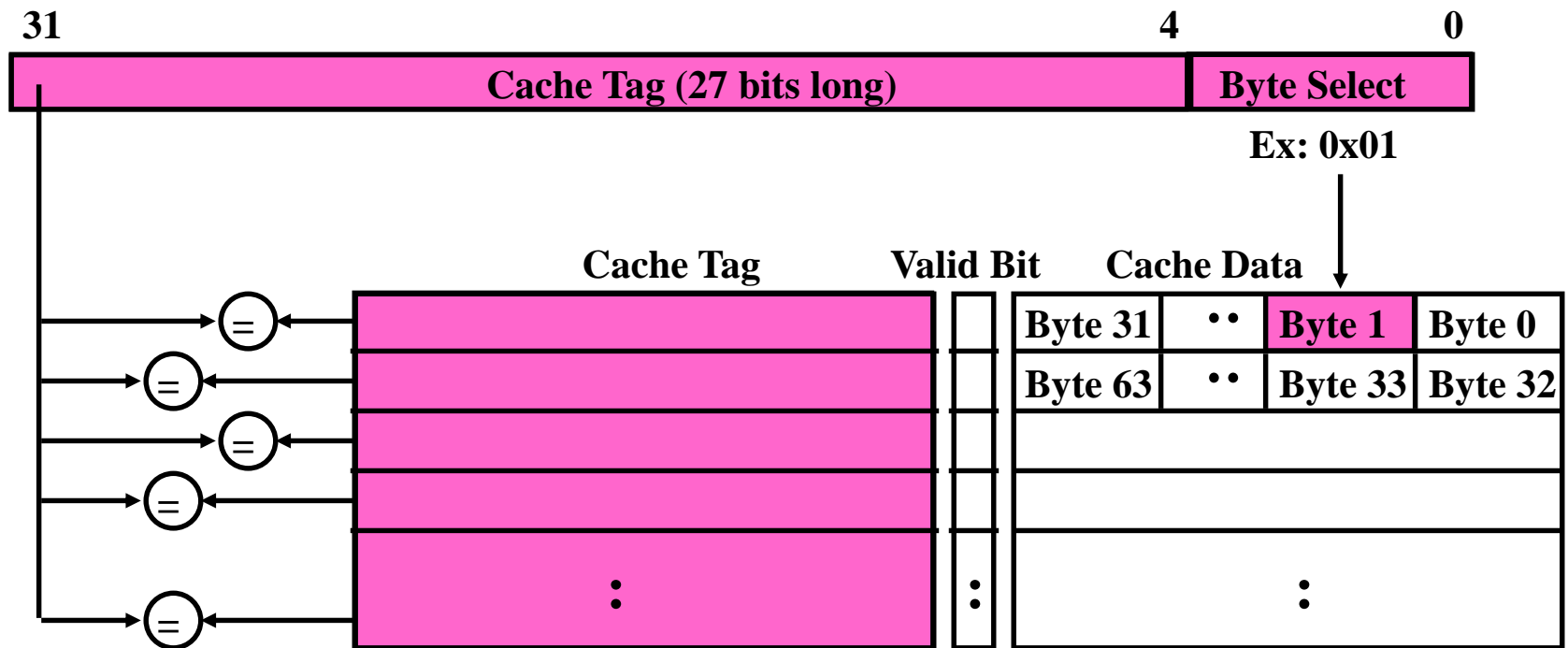
=?

Reference stream

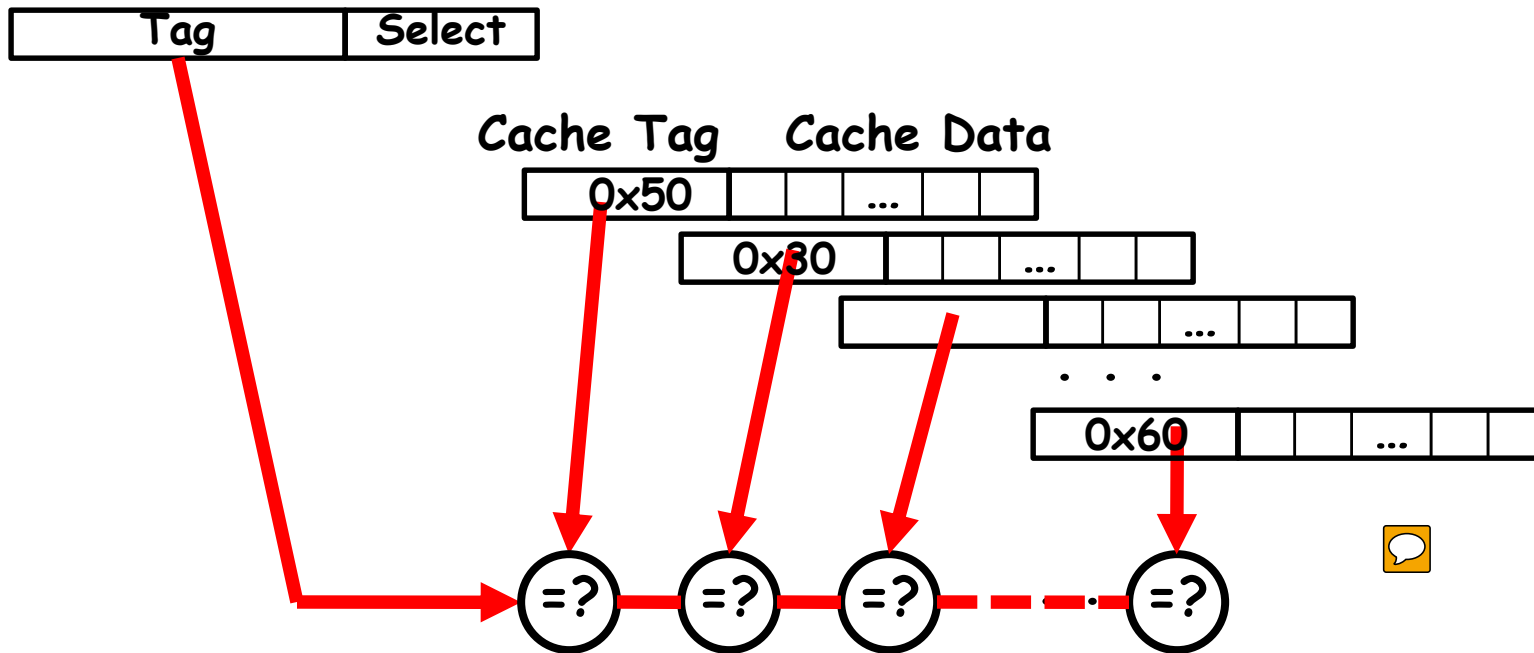
<0x50,0x04,0x00>
<0x30,0x02,0x00>
<0x50,0x04,0x03>
<0x30,0x02,0x02>
...
<0x60,0x04,0x00>
<0x30,0x02,0x02>
<0x60,0x04,0x01>
<0x50,0x04,0x05>

Review: Fully Associative Cache

- **Fully Associative:** Every block can hold any line
 - Address does not include a cache index
 - Compare Cache Tags of all Cache Entries in Parallel
- Example: Block Size=32B blocks
 - We need N 27-bit comparators
 - Still have byte select to choose from within block



Fully Associative Cache: Example



Reference stream

```
<0x50,0x04,0x00>  
<0x30,0x02,0x00>  
<0x50,0x04,0x03>  
<0x30,0x02,0x02>  
...  
<0x60,0x04,0x00>  
<0x30,0x02,0x02>  
<0x60,0x04,0x01>  
<0x50,0x04,0x05>
```


Review: Which block should be replaced on a miss?

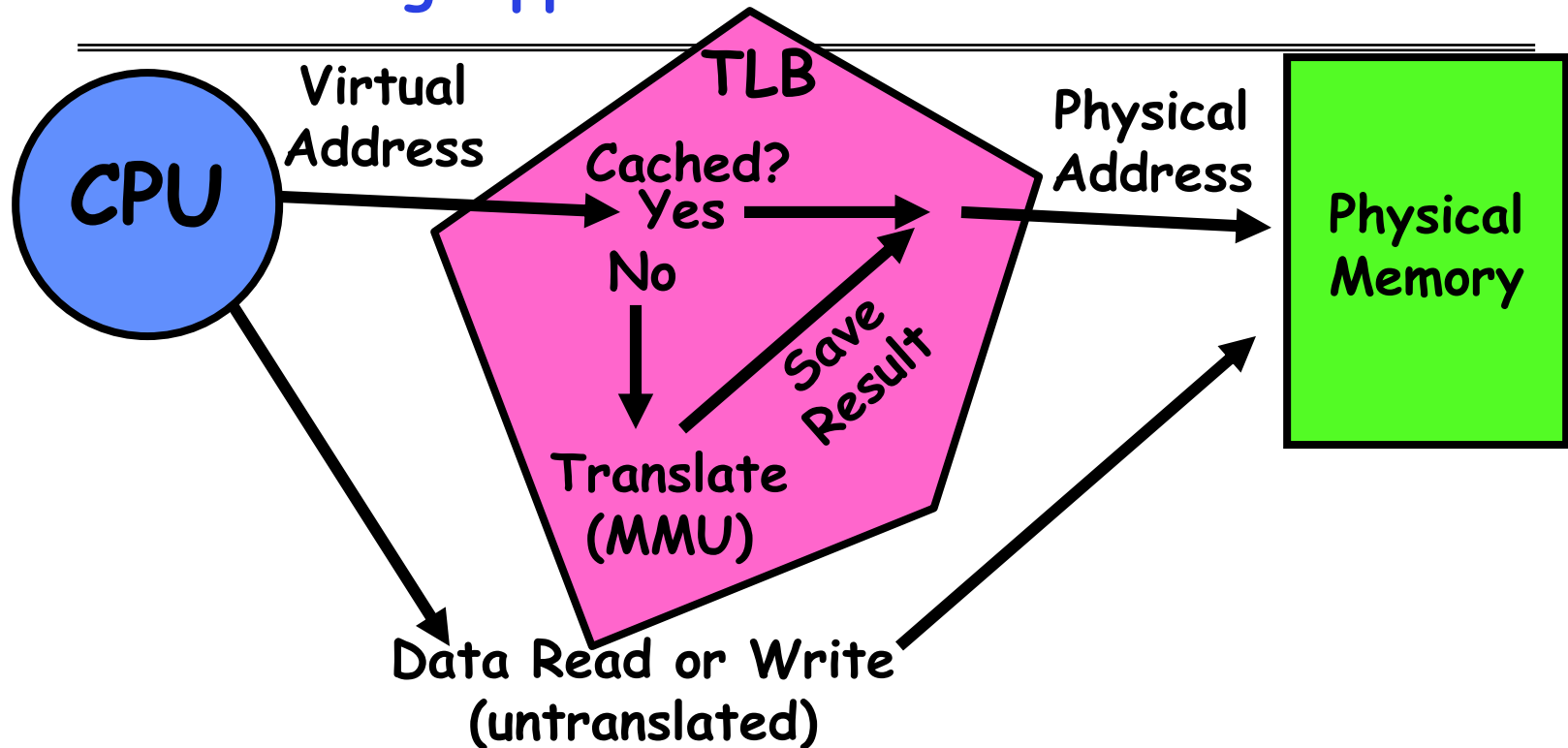
- Easy for Direct Mapped: Only one possibility
- Set Associative or Fully Associative:
 - Random
 - LRU (Least Recently Used)

Size	2-way		4-way		8-way	
	LRU	Random	LRU	Random	LRU	Random
16 KB	5.2%	5.7%	4.7%	5.3%	4.4%	5.0%
64 KB	1.9%	2.0%	1.5%	1.7%	1.4%	1.5%
256 KB	1.15%	1.17%	1.13%	1.13%	1.12%	1.12%

Review: What happens on a write?

- **Write through:** The information is written to both the block in the cache and to the block in the lower-level memory
- **Write back:** The information is written only to the block in the cache.
 - Modified cache block is written to main memory only when it is replaced
 - Question is block clean or dirty?
- Pros and Cons of each?
 - WT:
 - » PRO: read misses cannot result in writes
 - » CON: Processor held up on writes unless writes buffered
 - WB:
 - » PRO: repeated writes not sent to DRAM
processor not held up on writes
 - » CON: More complex
Read miss may require writeback of dirty data

Caching Applied to Address Translation



- Question is one of page locality: does it exist?
 - Instruction accesses spend a lot of time on the same page (since accesses sequential)
 - Stack accesses have definite locality of reference
 - Data accesses have less page locality, but still some...
- Can we have a TLB hierarchy?
 - Sure: multiple levels at different sizes/speeds

What Actually Happens on a TLB Miss?

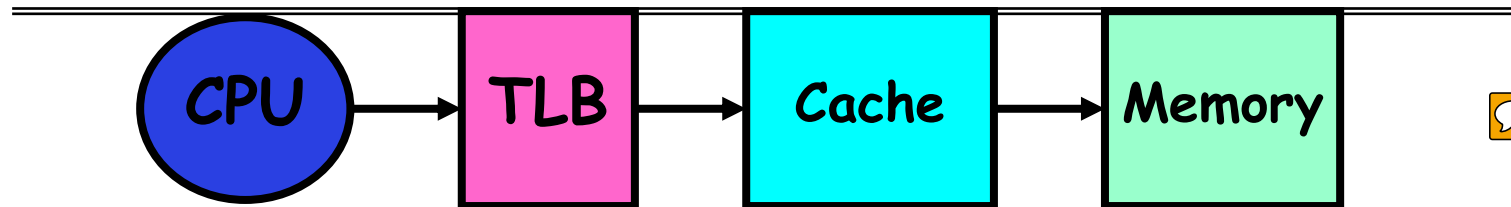


- **Hardware traversed page tables:**
 - On TLB miss, hardware in MMU looks at current page table to fill TLB (may walk multiple levels)
 - » If PTE valid, hardware fills TLB and processor never knows
 - » If PTE marked as invalid, causes Page Fault, after which kernel decides what to do afterwards
- **Software traversed Page tables (like MIPS)**
 - On TLB miss, processor receives TLB fault
 - Kernel traverses page table to find PTE
 - » If PTE valid, fills TLB and returns from fault
 - » If PTE marked as invalid, internally calls Page Fault handler
- **Most chip sets provide hardware traversal**
 - Modern operating systems tend to have more TLB faults since they use translation for many things
 - Examples:
 - » shared segments
 - » user-level portions of an operating system

What happens on a Context Switch?

- Need to do something, since TLBs map virtual addresses to physical addresses
 - Address Space just changed, so TLB entries no longer valid!
- Options?
 - Invalidate TLB: simple but might be expensive
 - » What if switching frequently between processes?
 - Include ProcessID in TLB
 - » This is an architectural solution: needs hardware
- What if translation tables change?
 - For example, to move page from memory to disk or vice versa...
 - Must invalidate TLB entry!
 - » Otherwise, might think that page is still in memory!

What TLB organization makes sense?



- Needs to be really fast
 - Critical path of memory access
 - » In simplest view: before the cache
 - » Thus, this adds to access time (reducing cache speed)
 - Seems to argue for Direct Mapped or Low Associativity
- However, needs to have very few conflicts!
 - With TLB, the Miss Time extremely high!
 - This argues that cost of Conflict (Miss Time) is much higher than slightly increased cost of access (Hit Time)
- **Thrashing:** continuous conflicts between accesses
 - What if use low order bits of page as index into TLB?
 - » First page of code, data, stack may map to same entry
 - » Need 3-way associativity at least?
 - What if use high order bits as index?
 - » TLB mostly unused for small programs

TLB organization: include protection

- How big does TLB actually have to be?
 - Usually small: 128-512 entries
 - Not very big, can support higher associativity
- **TLB usually organized as fully-associative cache**
 - Lookup is by Virtual Address
 - Returns Physical Address + other info
- What happens when fully-associative is too slow?
 - Put a small (4-16 entry) direct-mapped cache in front
 - Called a "TLB Slice"
- Example for MIPS R3000:

Virtual Address	Physical Address	Dirty	Ref	Valid	Access	ASID
0xFA00	0x0003	Y	N	Y	R/W	34
0x0040	0x0010	N	Y	Y	R	0
0x0041	0x0011	N	Y	Y	R	0

Example: R3000 pipeline includes TLB “stages”

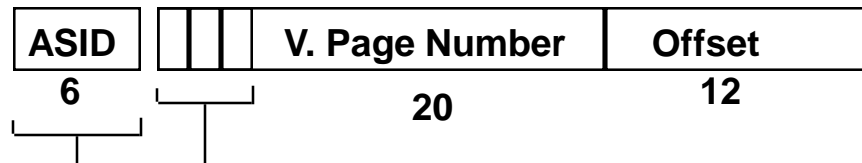
MIPS R3000 Pipeline

Inst Fetch		Dcd/ Reg		ALU / E.A		Memory	Write Reg	
TLB	I-Cache	RF	Operation				WB	
			E.A.	TLB		D-Cache		

TLB

64 entry, on-chip, fully associative, software TLB fault handler

Virtual Address Space

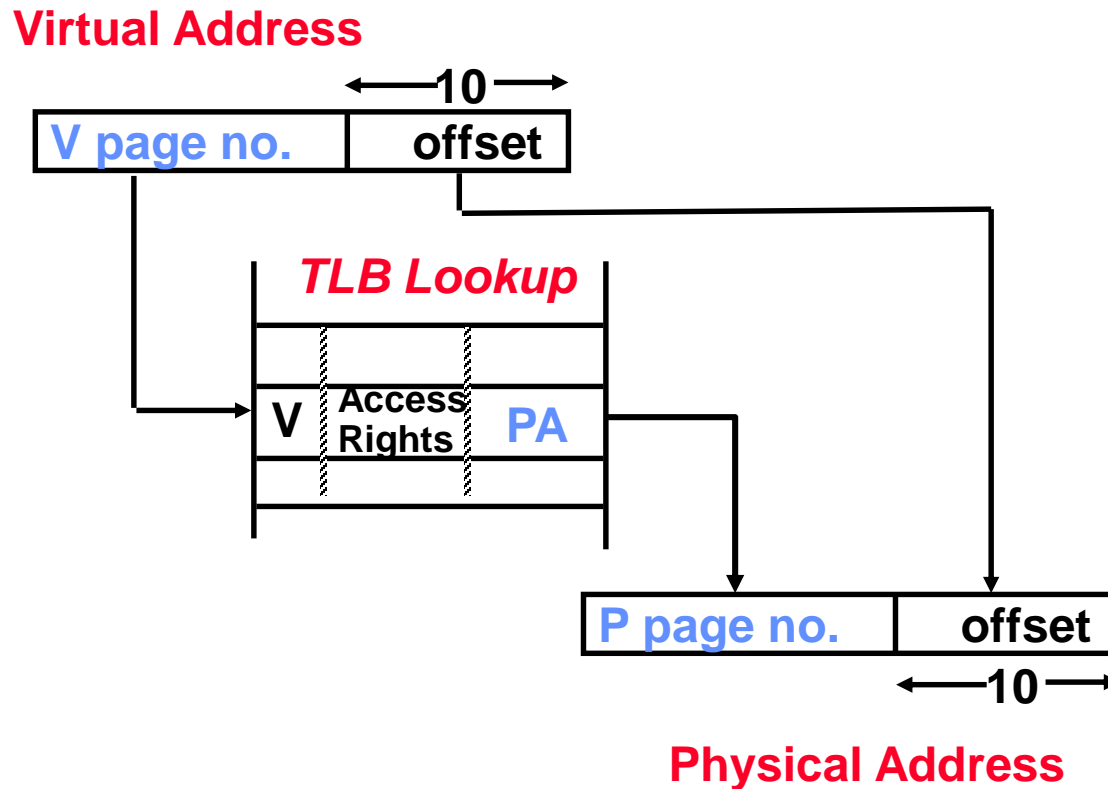


0xx User segment (caching based on PT/TLB entry)
100 Kernel physical space, cached
101 Kernel physical space, uncached
11x Kernel virtual space

Allows context switching among
64 user processes without TLB flush

Reducing translation time further

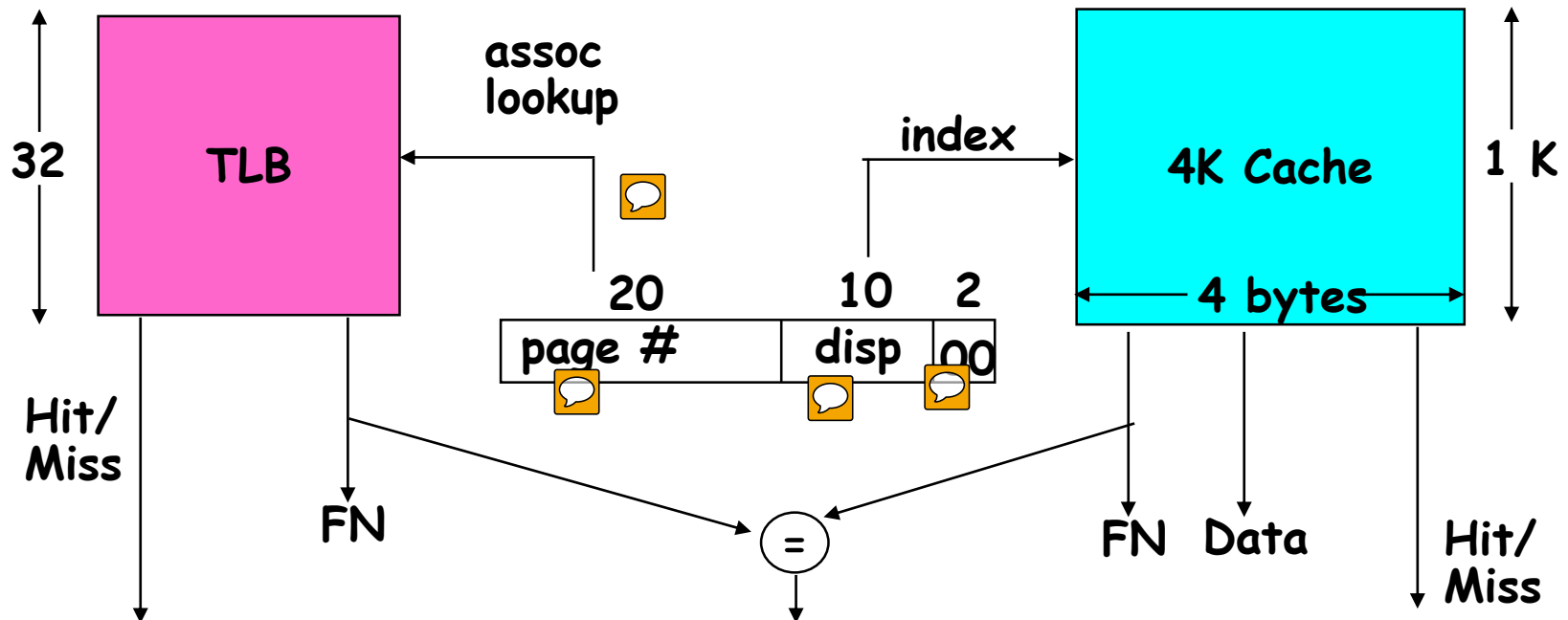
- As described, TLB lookup is in serial with cache lookup:



- Machines with TLBs go one step further: they overlap TLB lookup with cache access.
 - Works because offset available early

Overlapping TLB & Cache Access

- Here is how this might work with a 4K cache:



- What if cache size is increased to 8KB?**
 - Overlap not complete
 - Need to do something else.
- Another option: Virtual Caches**
 - Tags in cache are virtual addresses
 - Translation only happens on cache misses

Summary #1/2

- The Principle of Locality:
 - Program likely to access a relatively small portion of the address space at any instant of time.
 - » **Temporal Locality**: Locality in Time
 - » **Spatial Locality**: Locality in Space
- Three (+1) Major Categories of Cache Misses:
 - **Compulsory Misses**: sad facts of life. Example: cold start misses.
 - **Conflict Misses**: increase cache size and/or associativity
 - **Capacity Misses**: increase cache size
 - **Coherence Misses**: Caused by external processors or I/O devices
- Cache Organizations:
 - Direct Mapped: single block per set
 - Set associative: more than one block per set
 - Fully associative: all entries equivalent

Summary #2/2: Translation Caching (TLB)

- PTE: Page Table Entries
 - Includes physical page number
 - Control info (valid bit, writeable, dirty, user, etc)
- A cache of translations called a "Translation Lookaside Buffer" (TLB)
 - Relatively small number of entries (< 512)
 - Fully Associative (Since conflict misses expensive)
 - TLB entries contain PTE and optional process ID
- On TLB miss, page table must be traversed
 - If located PTE is invalid, cause Page Fault
- On context switch/change in page table
 - TLB entries must be invalidated somehow
- TLB is logically in front of cache
 - Thus, needs to be overlapped with cache access to be really fast