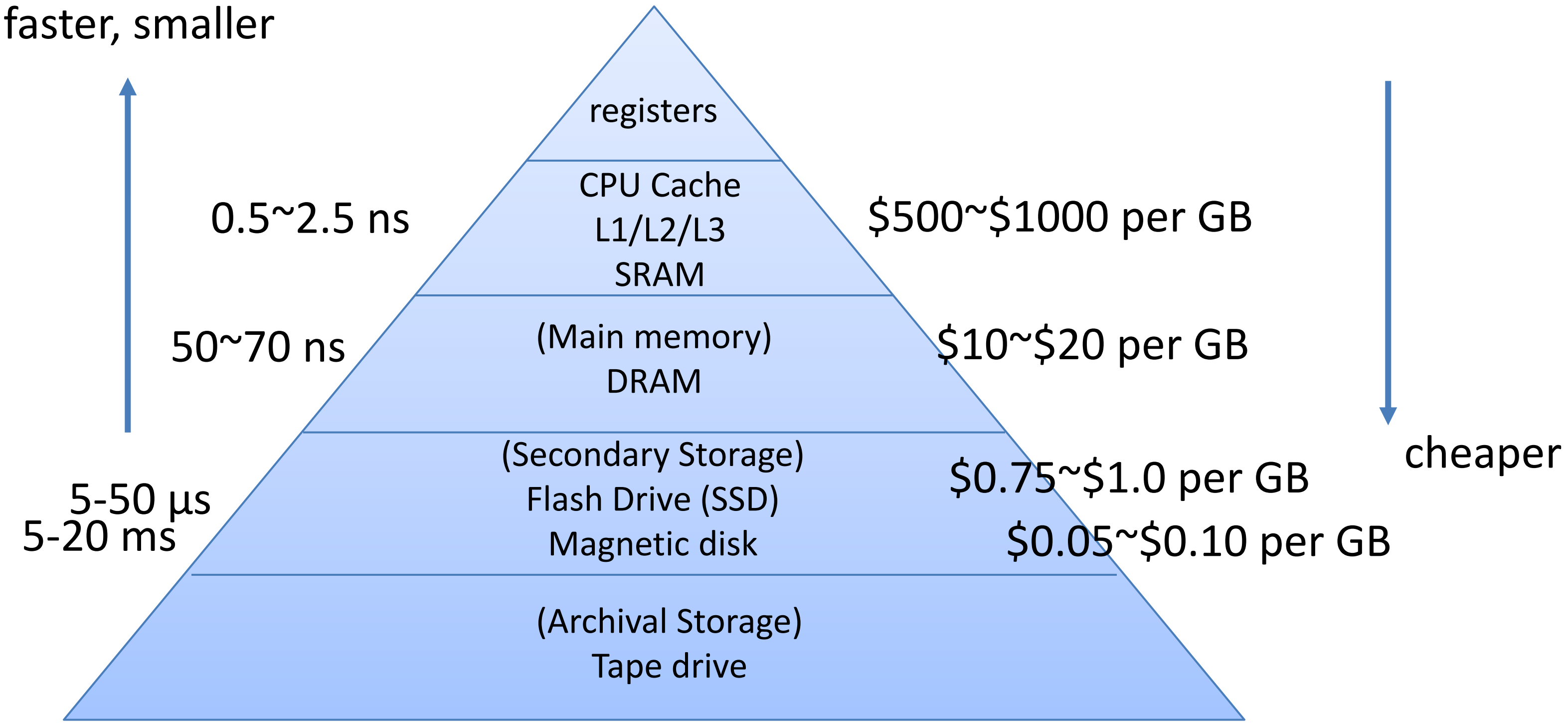
Memory Hierarchy

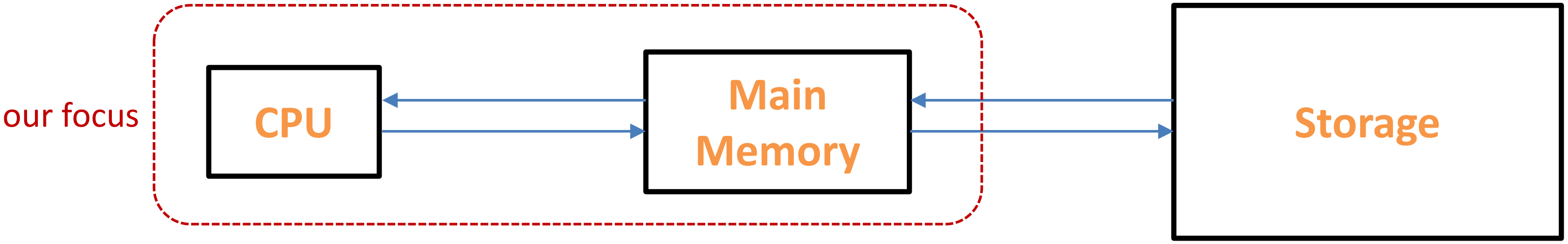
Memory Hierarchy

- Multiple levels of memory with different speeds and sizes
  - memory: any medium that can store data (e.g., DRAM, disk …)

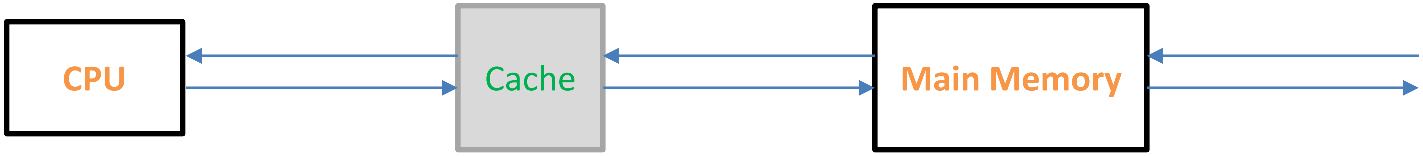


Cache Memory Concept

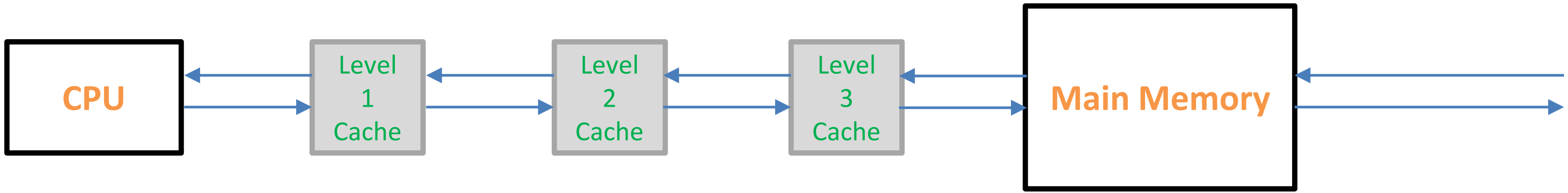
- Cache memory
  - Faster but smaller memory that holds data temporarily to improve the access time of a slower memory
  - Transparent



No cache



One level cache

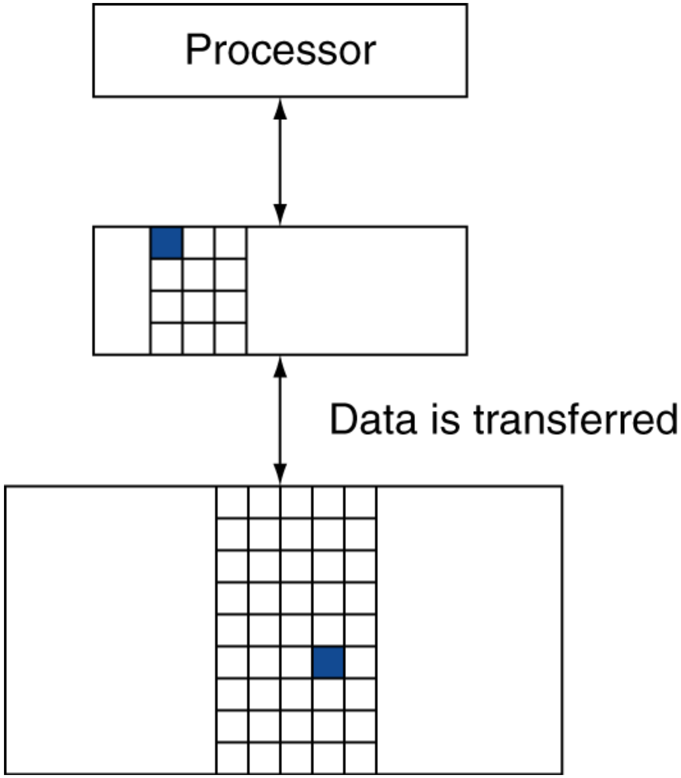


Multi-level cache

Cache Memory

- Underlying principle of Cache
  - Principle of Locality
    - Programs tend to access the same data in a near future
    - Temporal locality
      - If data is accessed, it is likely that it will be accessed again soon
    - Spatial locality
      - If data is accessed, it is likely that near-by data will also be accessed soon
- Cache memory is smaller than the lower level memory
  - Select only a small set of data to store in the cache

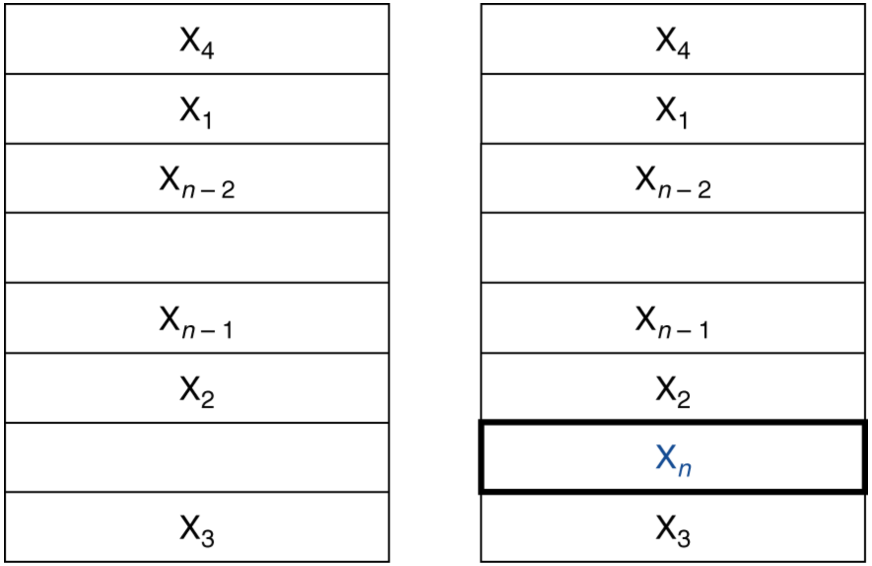
Cache Terminology



- Block (or line)
  - Minimum unit of information
- Hit
  - When data is present in the upper level
- Hit rate (hit ratio)
  - Fraction of memory access that is 'hit'
- Miss, miss rate
- Hit time
  - Time to access the upper level (including the time to determine hit or miss)
- Miss penalty
  - Time to copy a block from the lower level
    - Replace and transfer data

Cache Access

- Assume that:
  - CPU requests one word at a time
  - Block size is one word
- Memory access: $X_1, \ldots, X_{n-1}, X_n$



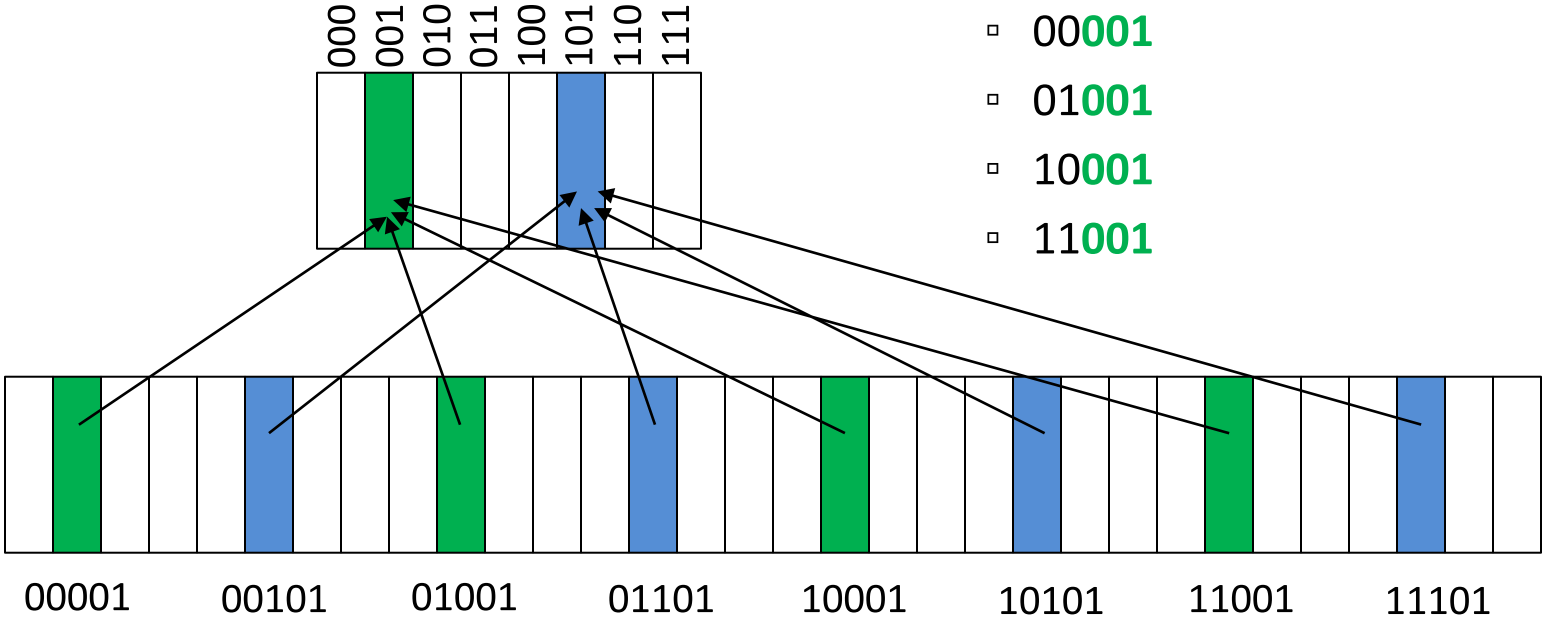
a. Before the reference to X_n b. After the reference to X_n

- Cache placement issue

Direct-mapped Cache

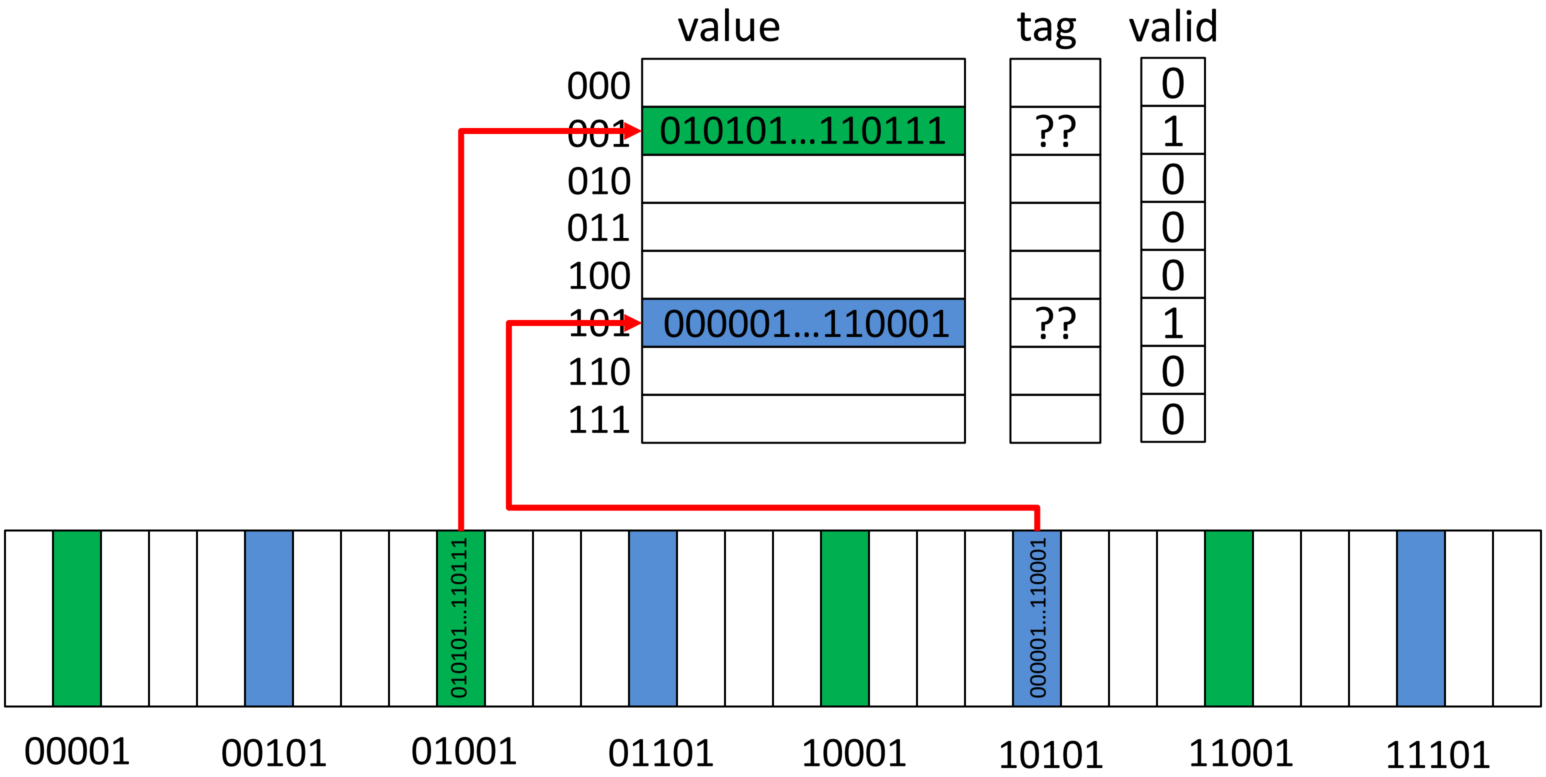
- In direct-mapped cache:
  - Cache location is determined as a function of the memory address
  - A block (word) can go into only one place in the cache

    ⟷ fully associative cache
  - Cache slot = (word address) mod (# of cache blocks)
    - # of cache blocks are in power of 2
      - Blocks that go into location **001**
        - 00**001**
        - 01**001**
        - 10**001**
        - 11**001**



Direct-mapped Cache

- Multiple block maps to one location in the cache
  - How do we know which address the block came from?
    - Block in 001 comes from 00001, 01001, 10001, 11001
- Tags
  - High-order bits of the address – ***00***001, ***01***001, ***10***001, ***11***001



	value	tag	valid
000			0
001	010101…110111	??	1
010			0
011			0
100			0
101	000001…110001	??	1
110			0
111			0

Accessing a Cache

Decimal address of reference	Binary address of reference	Hit or miss in cache	Assigned cache block (where found or placed)
22	10110_{two}	miss (5.9b)	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	miss (5.9c)	$(11010_{two} \bmod 8) = 010_{two}$
22	10110_{two}	hit	$(10110_{two} \bmod 8) = 110_{two}$
26	11010_{two}	hit	$(11010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	miss (5.9d)	$(10000_{two} \bmod 8) = 000_{two}$
3	00011_{two}	miss (5.9e)	$(00011_{two} \bmod 8) = 011_{two}$
16	10000_{two}	hit	$(10000_{two} \bmod 8) = 000_{two}$
18	10010_{two}	miss (5.9f)	$(10010_{two} \bmod 8) = 010_{two}$
16	10000_{two}	hit	$(10000_{two} \bmod 8) = 000_{two}$

Index	V	Tag	Data
000	N		
001	N		
010	N		
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

Index	V	Tag	Data
000	N		
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	N		
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	11_{two}	Memory (11010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

Index	V	Tag	Data
000	Y	10_{two}	Memory (10000_{two})
001	N		
010	Y	10_{two}	Memory (10010_{two})
011	Y	00_{two}	Memory (00011_{two})
100	N		
101	N		
110	Y	10_{two}	Memory (10110_{two})
111	N		

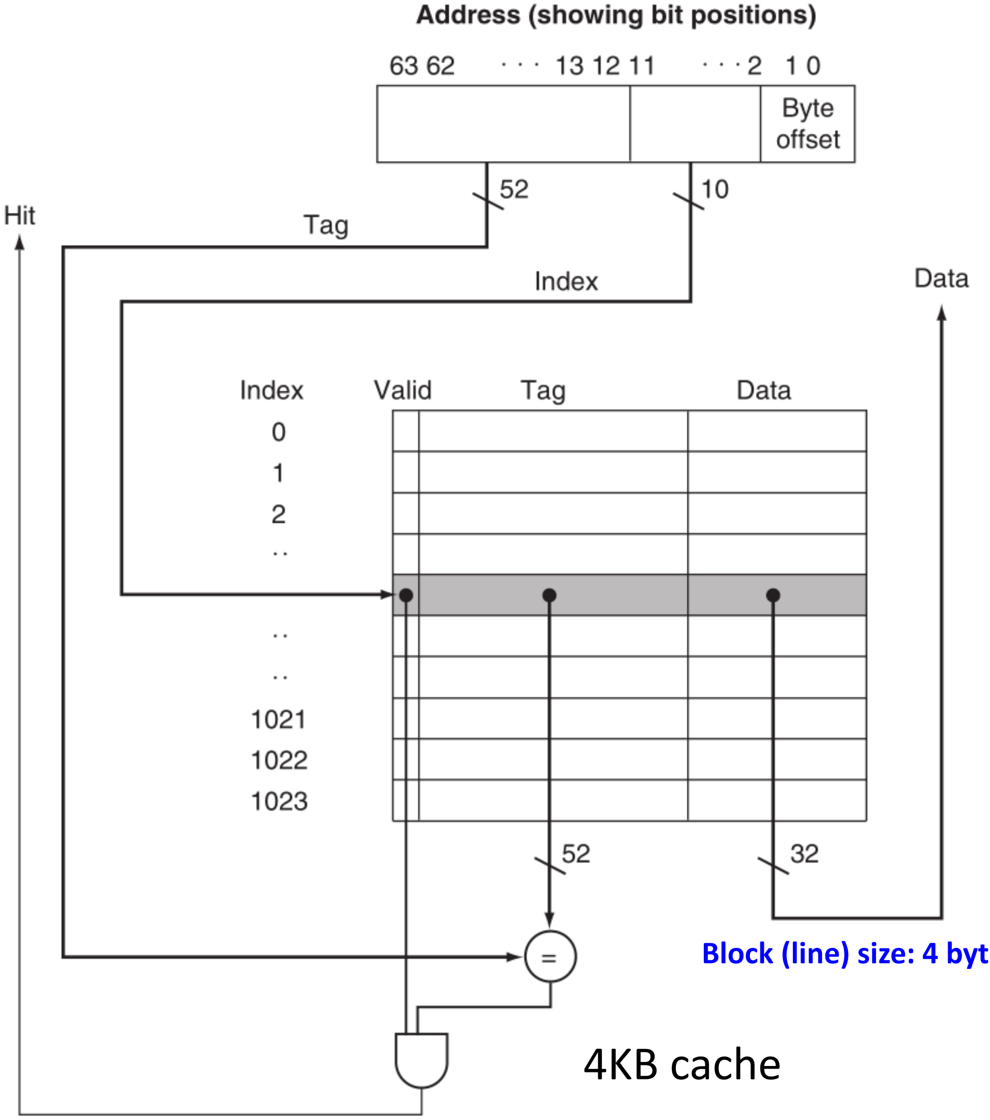
Address Subdivision

- Address is divided into tag part and index part
  - tag part: 52 bits
  - index part: 10 bits
  - byte offset: 2 bits

- Total number of bits in the cache
  n: number of index bits
  m: 2^m is the # of words
  (m=0 if 1 word in a line)

$$2^n \times \sum \begin{cases} \text{valid bit: 1} \\ \text{tag bits: 64-(n + m + 2)} \\ \text{data bits: } 2^m\text{x32} \end{cases}$$

*Cache size usually includes only the data size

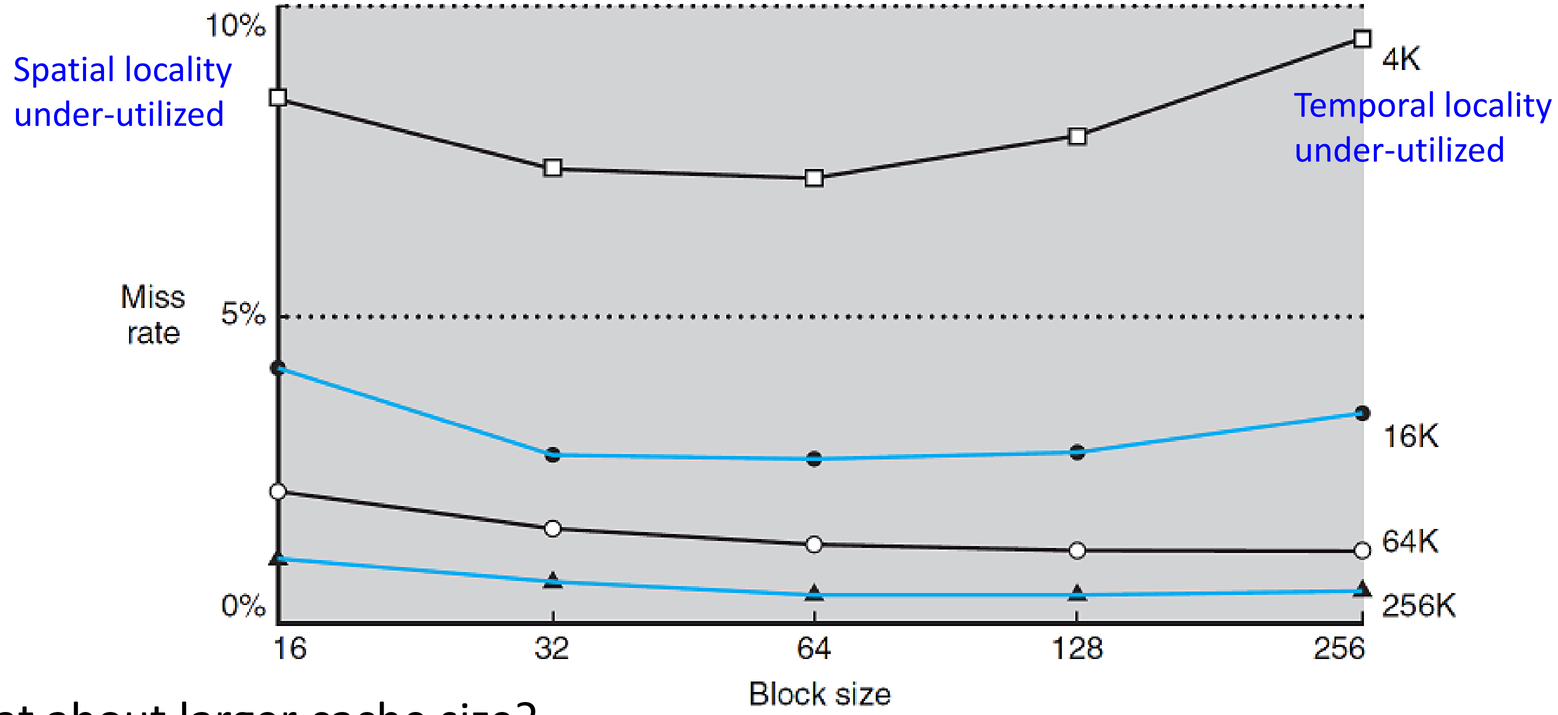


Block (line) size: 4 bytes

4KB cache

Block Size and Miss Rate

- Larger block size

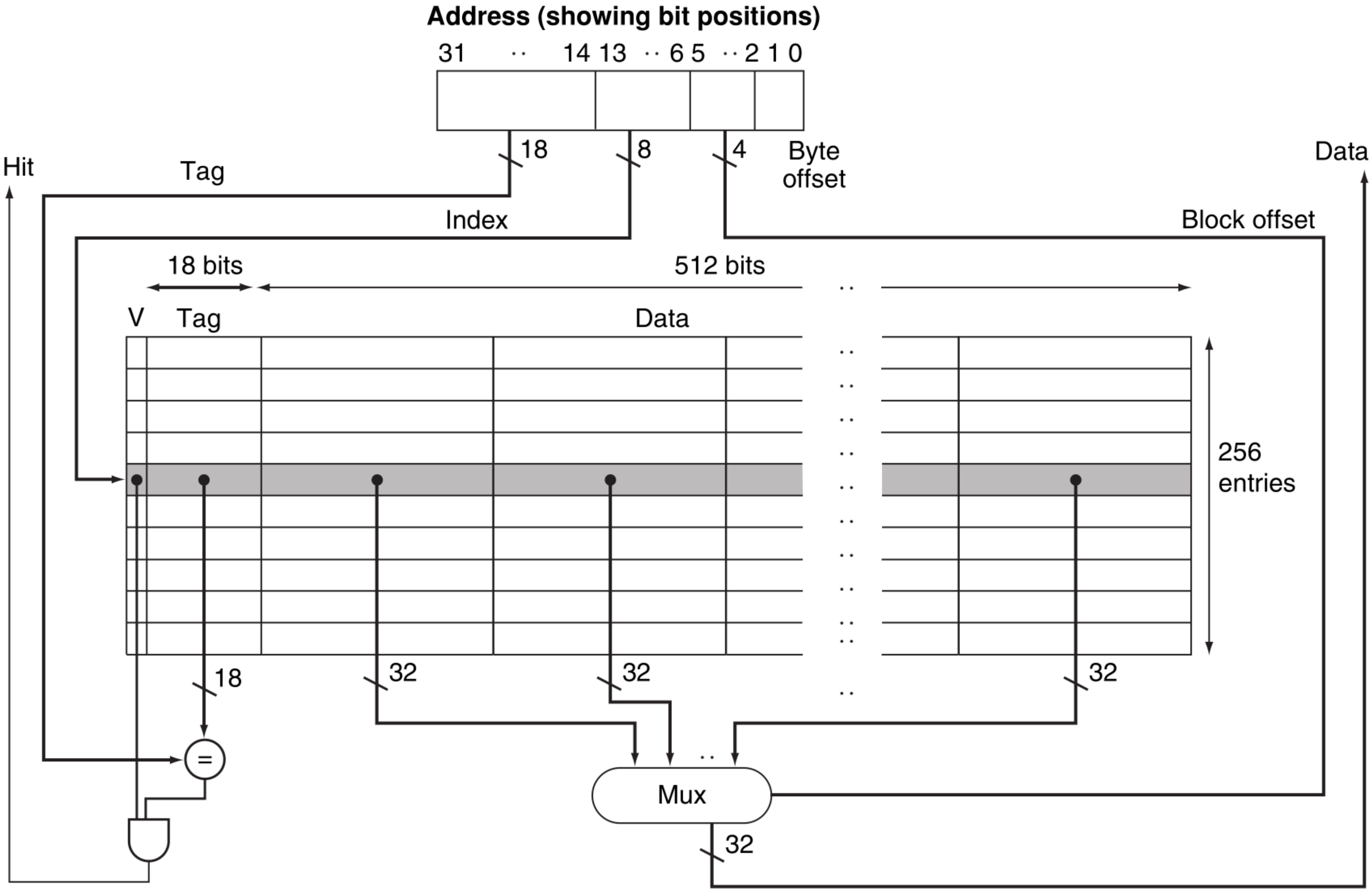


* What about larger cache size?

- Miss penalty
  - Time to load a block from MM(Main memory) to Cache
    - Set-up time + transfer time
  - Reducing the miss penalty (by hiding the transfer time)
    - "early-restart" → more effective for instruction than data
    - "critical word first"

Example Cache

- Intrinsity FastMATH embedded microprocessor

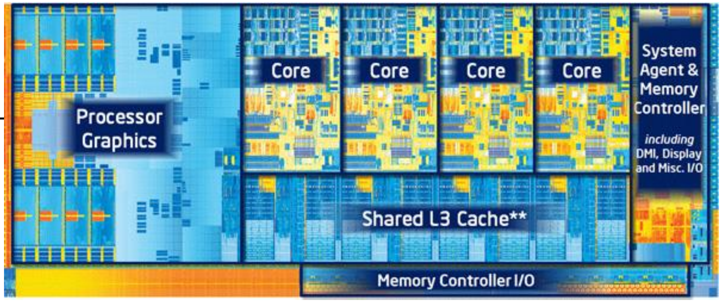


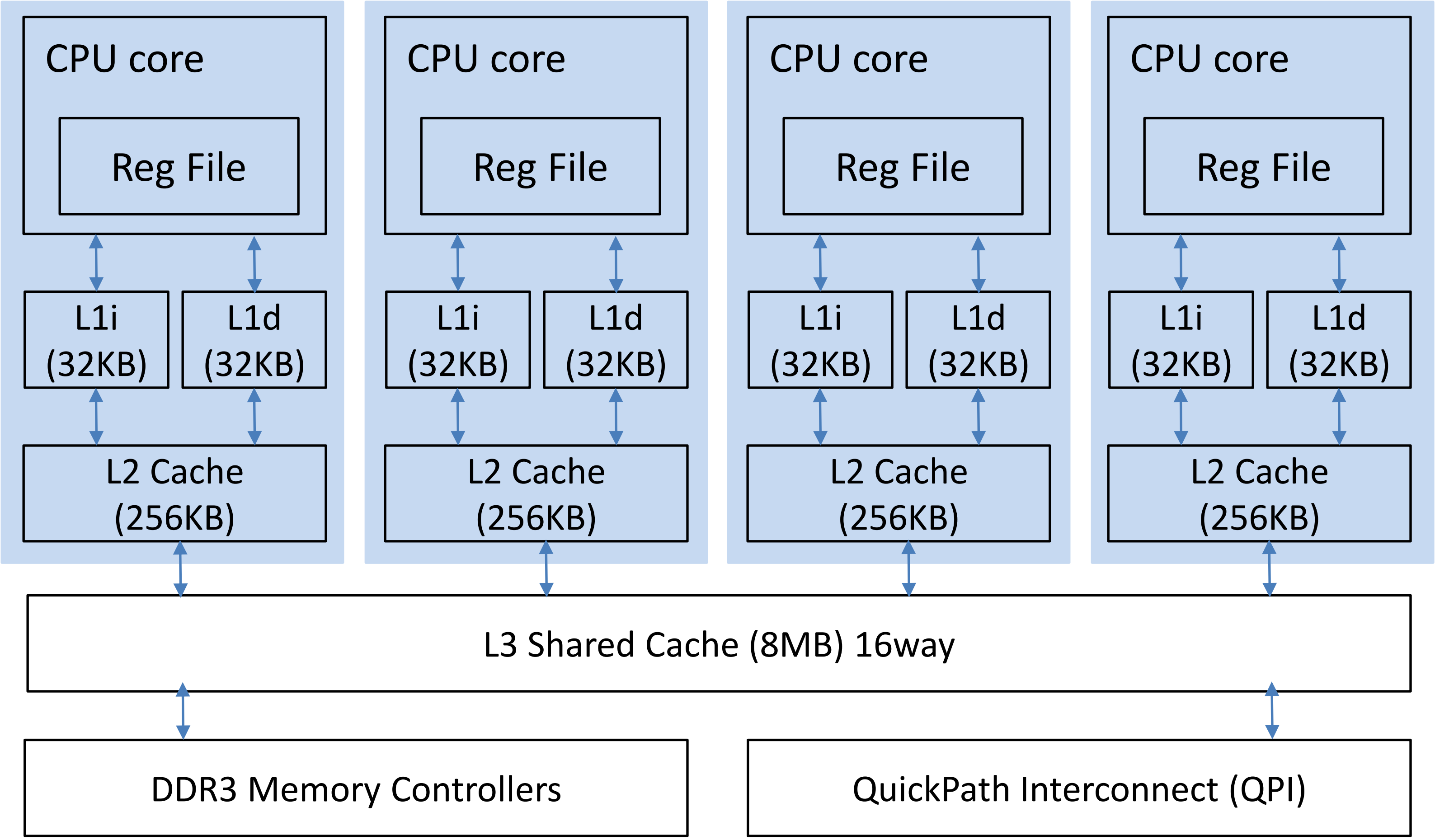
Intel core i7 Block Diagram

Four x86 SMT(Simultaneous Multithreading) Processors

Dedicated L1, L2 Cache

Shared L3 Cache





CPU core

Reg File

L1i (32KB)

L1d (32KB)

L2 Cache (256KB)

CPU core

Reg File

L1i (32KB)

L1d (32KB)

L2 Cache (256KB)

CPU core

Reg File

L1i (32KB)

L1d (32KB)

L2 Cache (256KB)

CPU core

Reg File

L1i (32KB)

L1d (32KB)

L2 Cache (256KB)

L3 Shared Cache (8MB) 16way

DDR3 Memory Controllers

QuickPath Interconnect (QPI)

Three 8-byte channels, 32GB/s

25.6GB/s Bandwidth

Cache Miss Handling in Processor

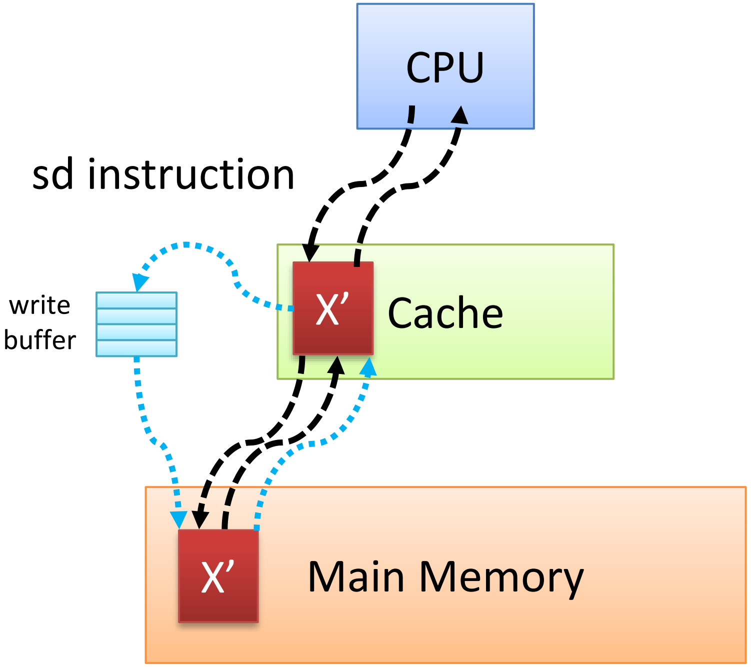
- Separate cache for instruction and data stream
  - Instruction and data cache at L1

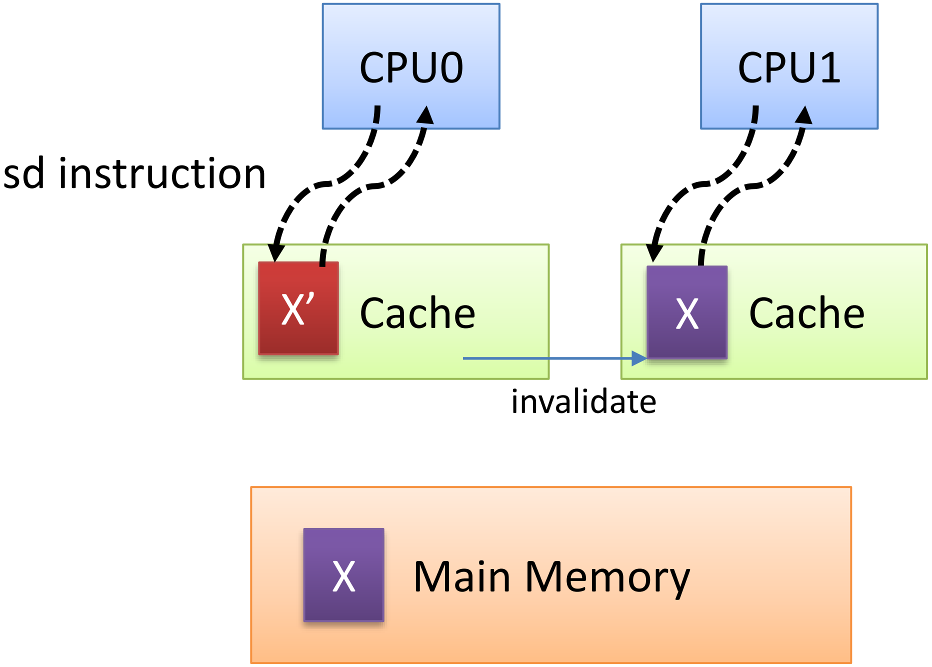
Processors	MIPS32 74K Atheros AR9344	Intel i7-4770	Apple A10	ARM Cortex-A57
Instruction cache	32KB, 32B line, 4-way	32KB, 64B line, 8-way	64KB, 64B line, 4-way	48KB, 64B line, 3-way
Data cache	32KB, 32B line, 4-way	32KB, 64B line, 8-way	64KB, 64B line, 4-way	48KB, 64B line, 2-way

- Cache miss on instruction (identical for data miss)
  - Send PC-4 to memory unit
  - Instruct MM (main memory) to perform read
  - Wait until data is ready
  - Write data to cache entry, fill up the tag and valid fields
  - Re-send instruction address to the memory – hit!
- ☆(advanced) Improving the cache performance
  - In In-order execution, pipeline needs to stall on cache miss!
  - Out-of-order execution: non-blocking cache
  - Victim cache – small (fully associative) cache holding the replace blocks

Write Handling in Cache

- Write-through cache
  - Write operation completes after updating the main memory and the cache
  - Data consistency maintained
- Write-back cache
  - Write operation completes after only updating the cache
  - Main memory is synced only when a block is replaced from cache





Cache Performance

- CPU time
  - Normal execution: includes cache hit time
  - Memory stall cycle: cache miss penalty

$$\begin{aligned} CPUtime &= total\ clock\ cycles\ \times clock\ cycle\ time \\ &= (CPU\ cycles + \boldsymbol{Memory\ stall\ cycles}) \times clock\ cycle\ time \end{aligned}$$

- Assume the read and write miss penalties are the same
  - Write-through: complication from *write buffer*
  - Write-back: potential stall from sync when a block is replaced

$$\begin{aligned} \boldsymbol{Memory\ stall\ cycles} &= Read\ stall\ cycles + Write\ stall\ cycles \\ &= inst\ count \times \frac{memory\ access}{inst\ count} \times miss\ rate \times miss\ penalty \end{aligned}$$

Cache Performance

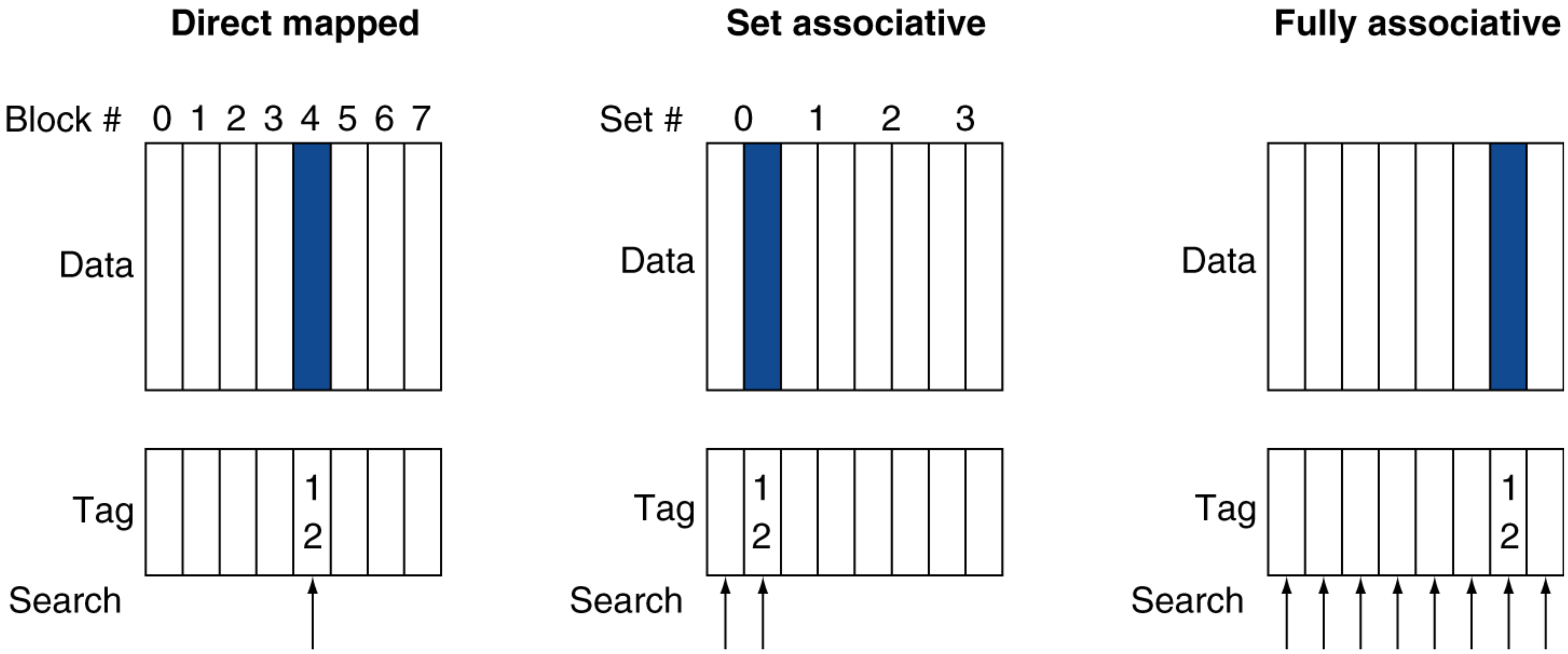
- Instruction cache miss rate: 2%
- Data cache miss rate: 4%
- Ideal case CPI: 2
- Miss penalty: 100 cycles
- Load and store: 36% of instructions
- Question: How much faster is the processor with perfect cache?
  - Number of instructions: N
  - Instruction miss and data miss are separate
    - total penalty cycles = inst miss cycles + data miss cycles
  - inst miss cycles = N x 2% x 100 = **2N**
  - data miss cycles = N x 36% x 4% x 100 = **1.44N**
  - Total cycles = 2N + **2N** + **1.44N** = 5.44N
  - Total cycles for perfect cache = 2N
  - Performance is better by 5.44N/2N = **2.72**

계산 전반적으로 재검토

Data miss시 cache hit 사이클도 추가해야함.

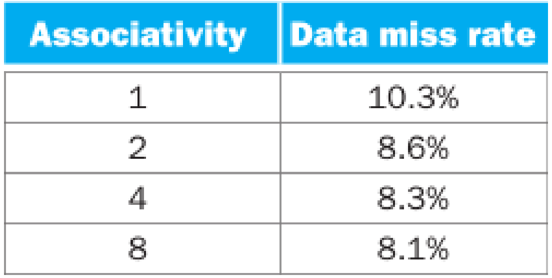
Associative Cache

- Direct-mapped cache: a block can be placed in one location
- Fully associative cache: a block can be placed anywhere in the cache
- Set associative cache: more than one location, but not anywhere (e.g., n-way set associative cache)



Associative Cache

- Direct-mapped cache and fully associative cache are *special cases* of set associative cache
  - direct-mapped: one-way set associative
  - fully associative: n-way set associative
- Advantage of associative cache
  - Decreased miss rate

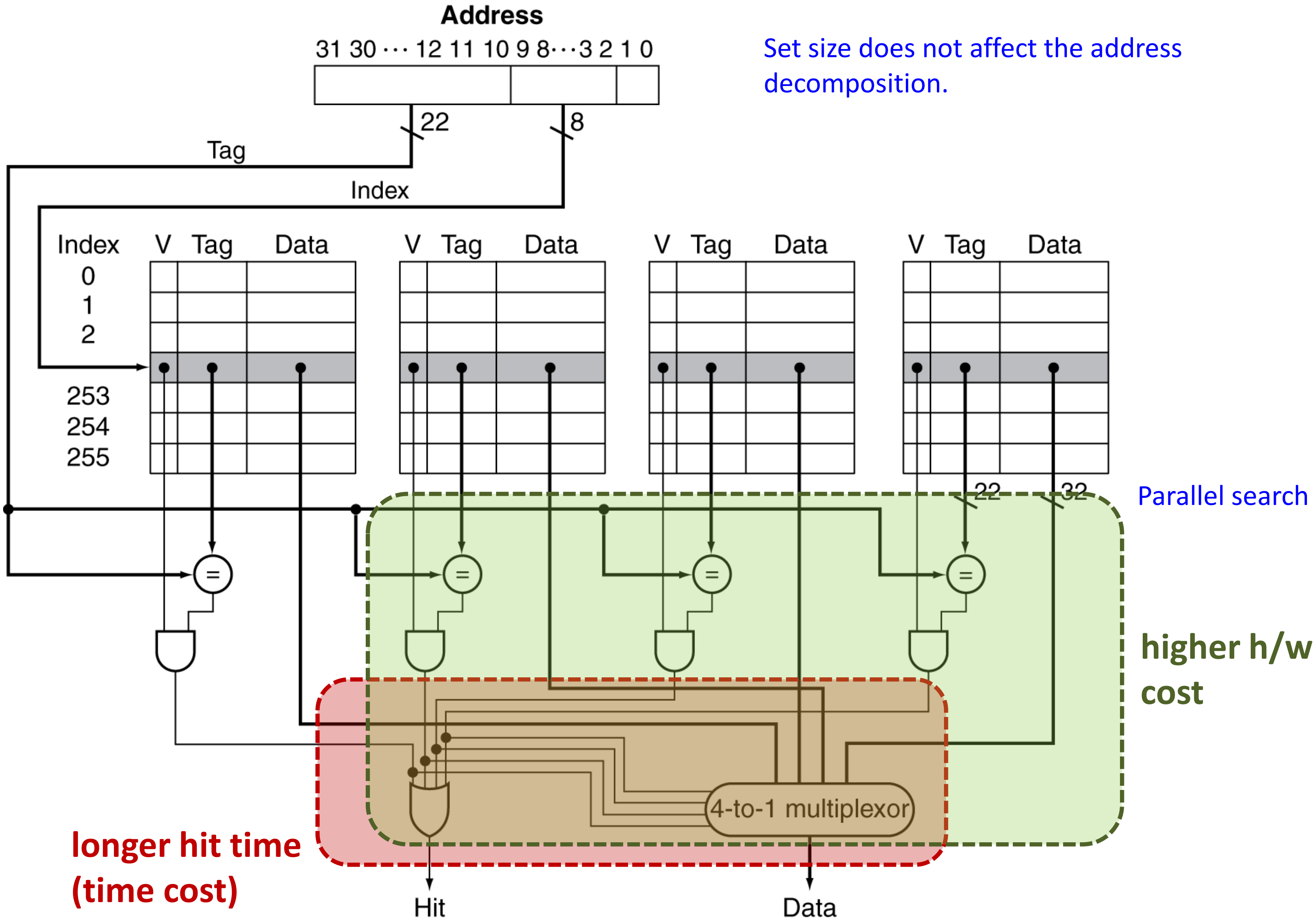


Associativity	Data miss rate
1	10.3%
2	8.6%
4	8.3%
8	8.1%

64KB data cache
16-word line
Intrinsity FastMATH processor for SPEC CPU2000 benchmarks

- Disadvantage
  - Longer hit time
    - All tags within the set must be compared and selected
  - Higher H/W cost

4-way Set Associative Cache Schematic



Set size does not affect the address decomposition.

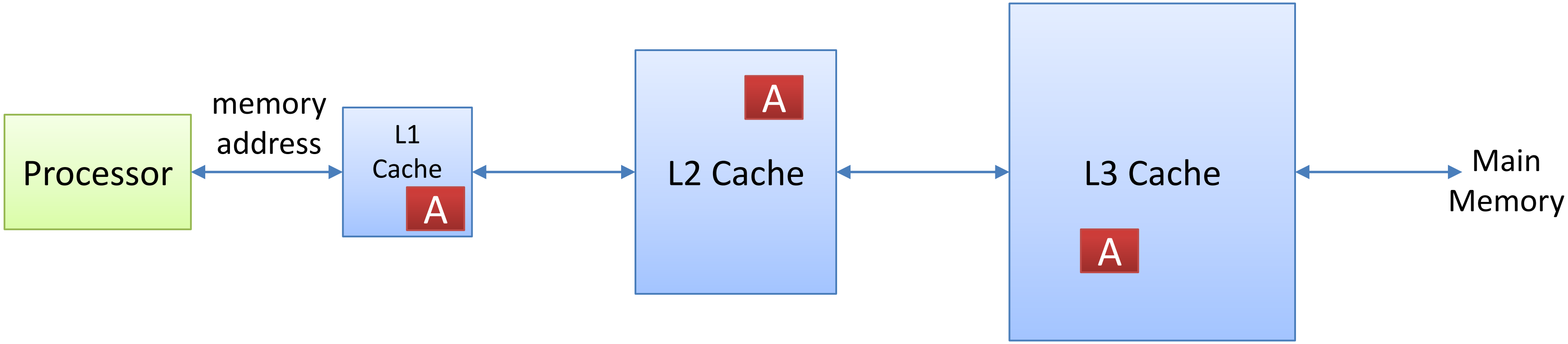
Parallel search

higher h/w cost

longer hit time (time cost)

Multi-level Cache

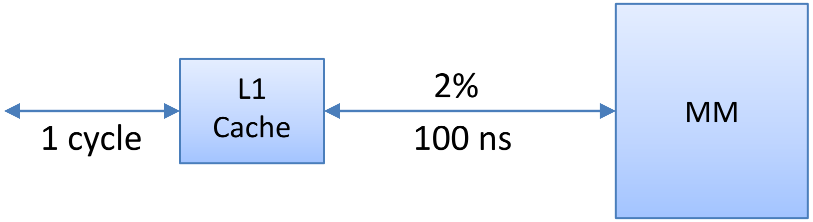
- Modern processors have more than 1 level of cache
  - L1, L2, shared L3 cache on chip
- L1: small, fast
- L2: larger, slower
  - higher associativity, larger block size
  - Access time is less critical than L1
- L1 cache miss → served by L2 cache
  - L2 cache miss → served by L3 cache



Multilevel Cache Performance

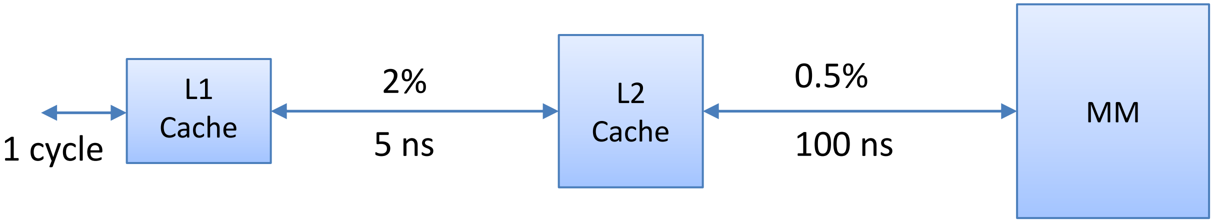
Base CPI	1.0
Clock rate	4 Ghz
L1 Miss rate	2%
Main memory access latency	100 ns

- Performance with L1 Cache only



  - i) Avg CPI = 1 cycle x 1.0 + 400 cycles x 0.02 = 9
  - ii) Avg CPI = 1 cycle x 0.98 + (400+1) cycles x 0.02 = 0.98 + 8.02 = 9

- Performance with L2 cache

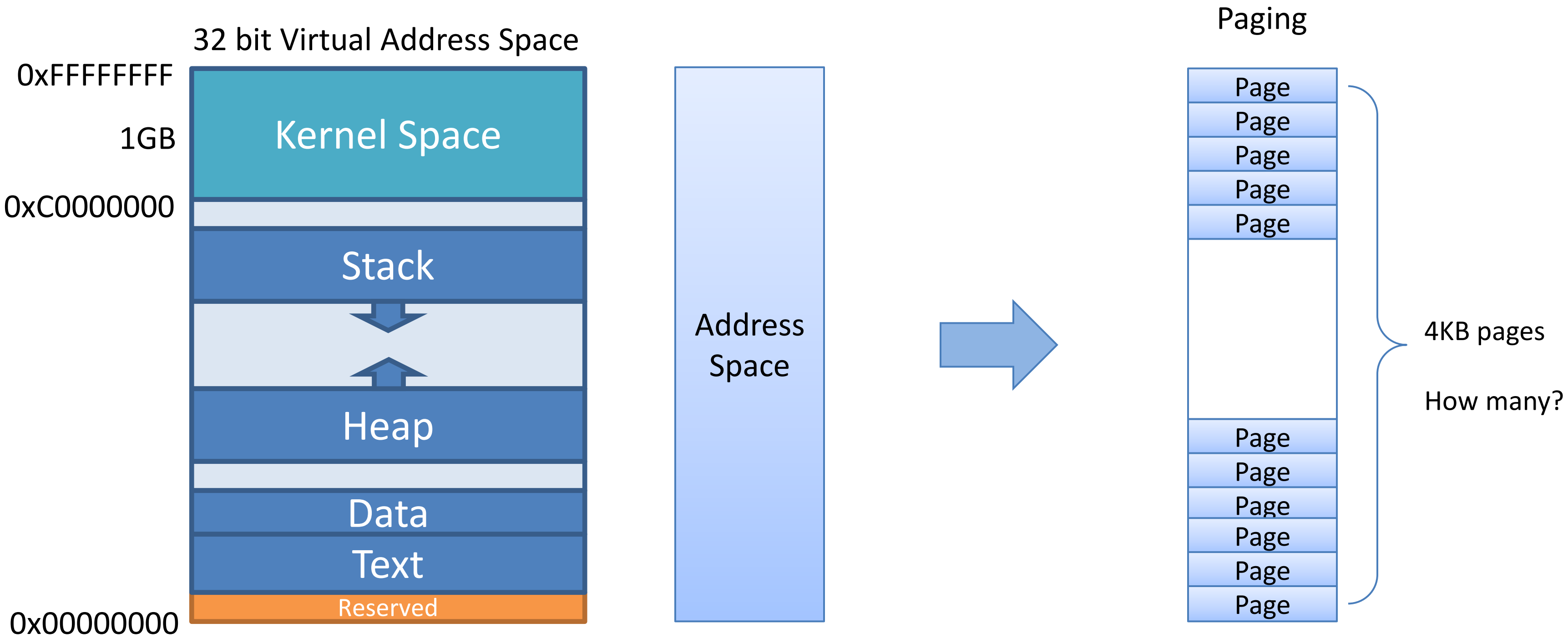


  - i) Avg CPI = 1 cycle x 1.0 + 20 cycles x 0.02 + 400 cycles x 0.005 = 3.4
  - ii) Avg CPI = 1 cycle x 0.98 + (20+1) cycles x 0.015 + (400+20+1) cycles x 0.005 = 3.4

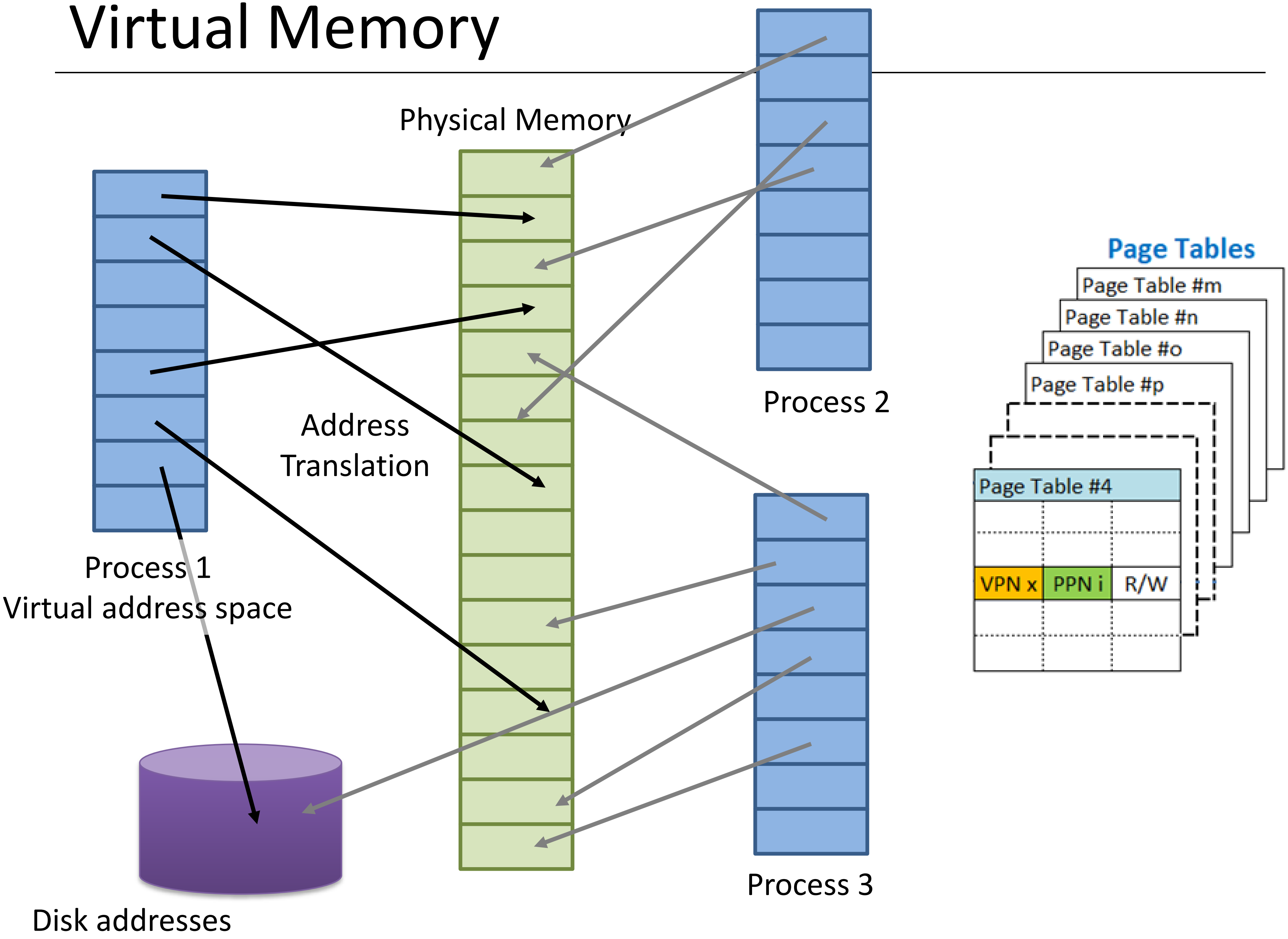
- Performance comparison
  - $\frac{9}{3.4} = 2.6$

Virtual Memory

- Virtual address space
  - Each process has 4GB address space (in 32 bit architecture)
  - Broken up into 4KB pages
  - Only small subset of these pages are in active use.

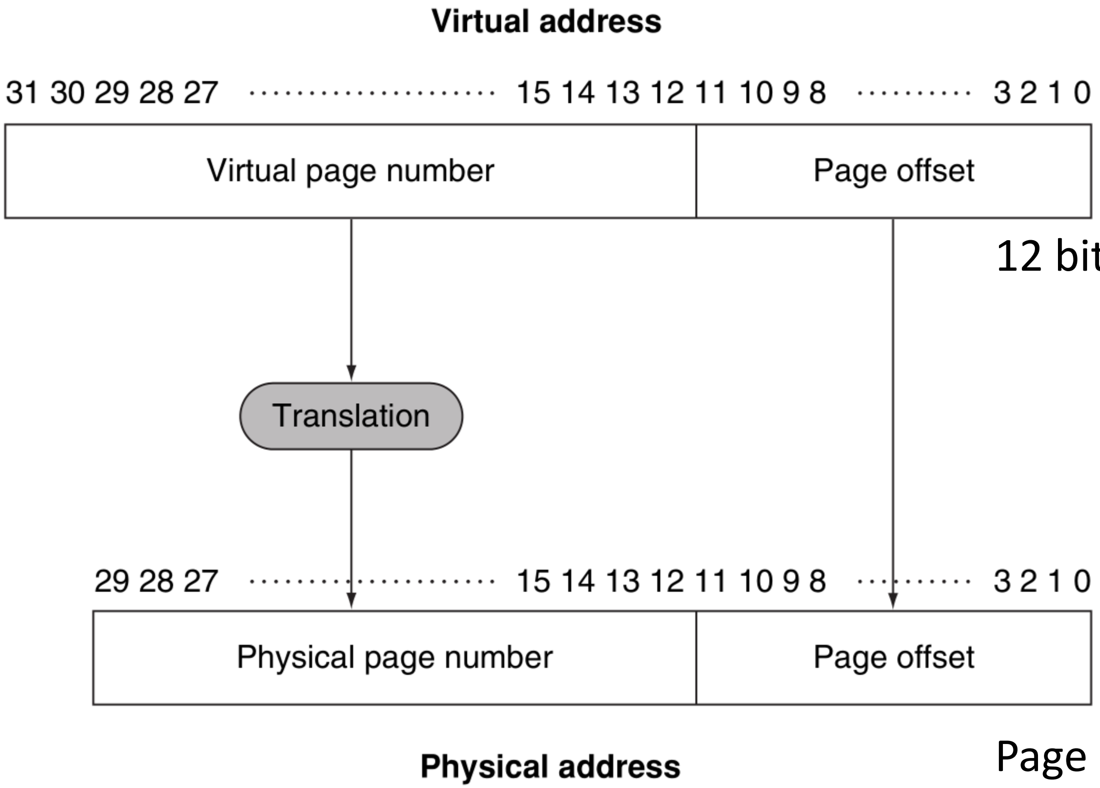


Virtual Memory



Virtual to Physical Address Mapping

- Virtual address consists of:
  - Virtual page number, Page offset



12 bits because it is 4KB

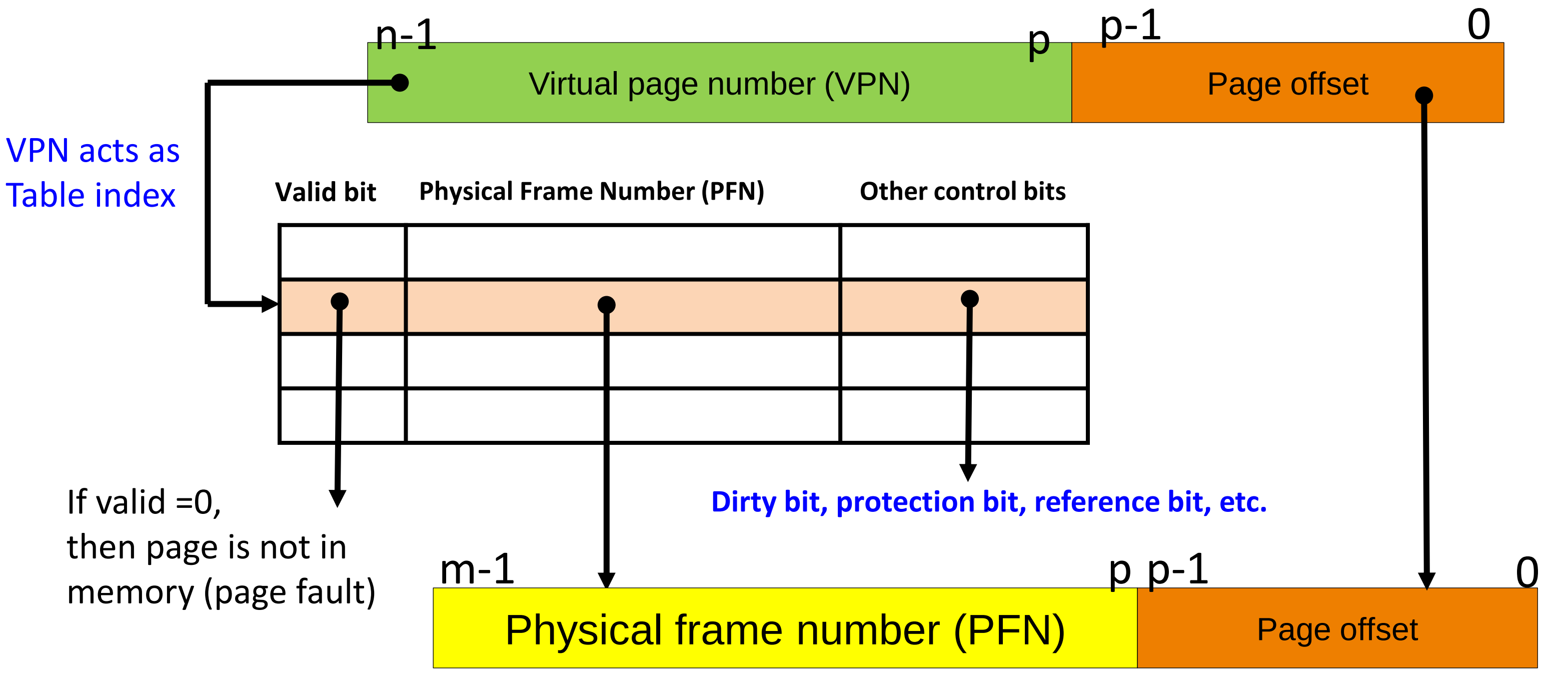
Page size=4KB

Number of physical pages (frames) = 2^{18} = 256K

Total Physical Mem: 256 x 4KB = 1GB

Address Translation

- How to obtain the physical frame number (PFN) for the virtual page number (VPN)?
  - **Page Table** contains the mapping information in each entry called **Page Table Entry (PTE)**
  - VPN is used as an index for the PTE in a page table



Address Translation in hardware

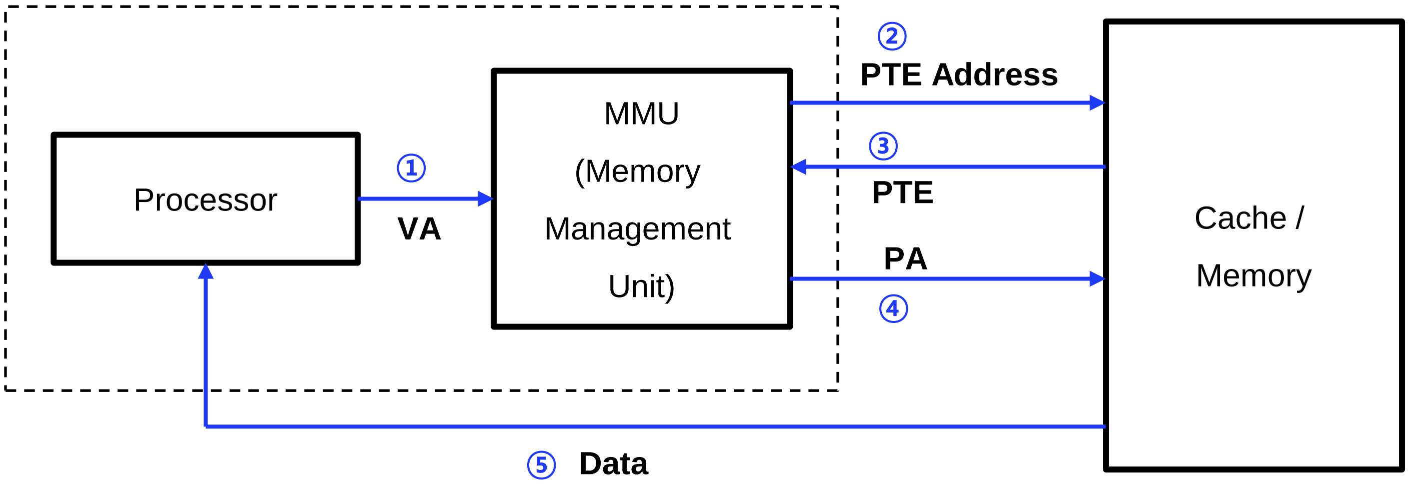
- When page is in the memory : **Page Hit**

1) Processor sends virtual address to MMU (Memory Management Unit)

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

4) MMU sends physical address to L1 cache

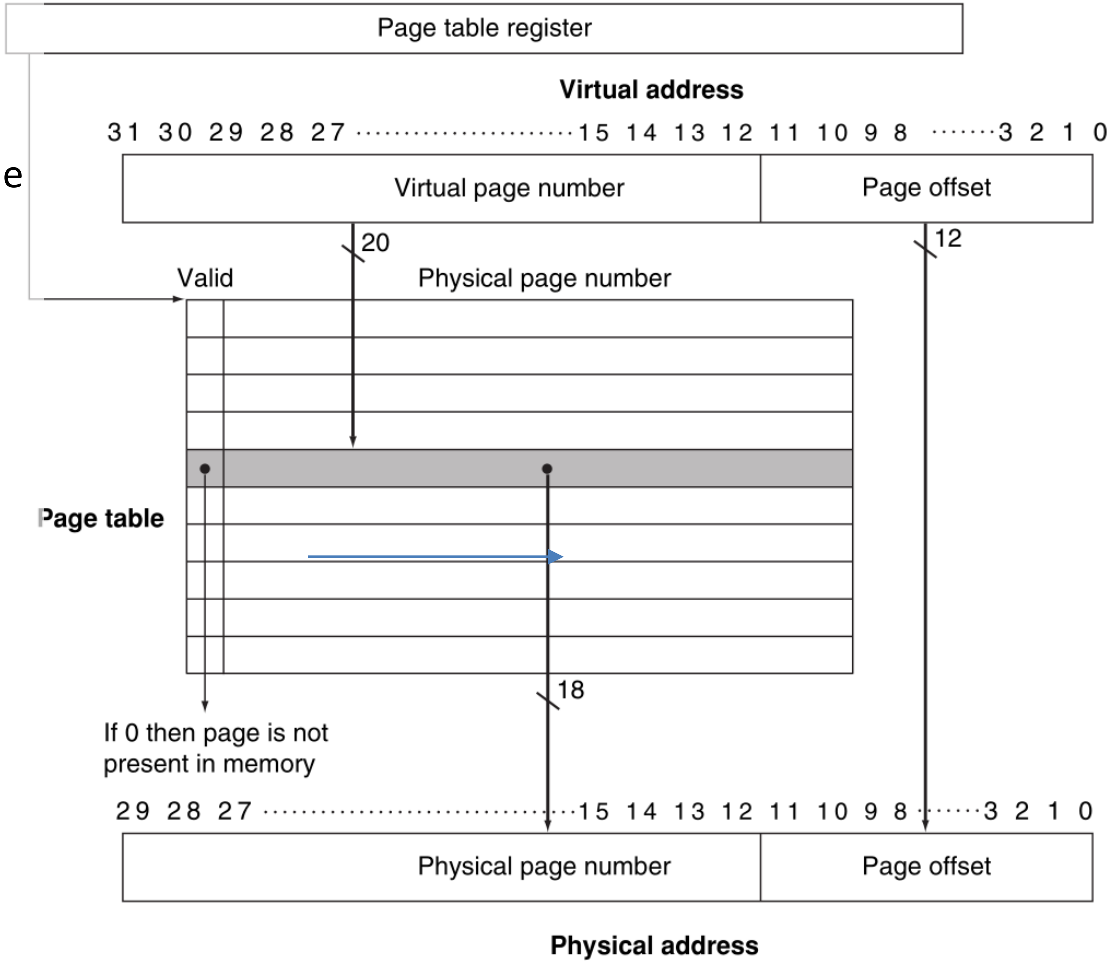
5) L1 cache sends a data word to the processor



Page Table Look-up

- Page table look-up is costly
  - Page table is stored in memory
  - Each memory access requires page table look-up.
    - One page table look-up may require more than one memory access
  - Leverage the reference locality!!
    - Translation of page will probably be needed again.

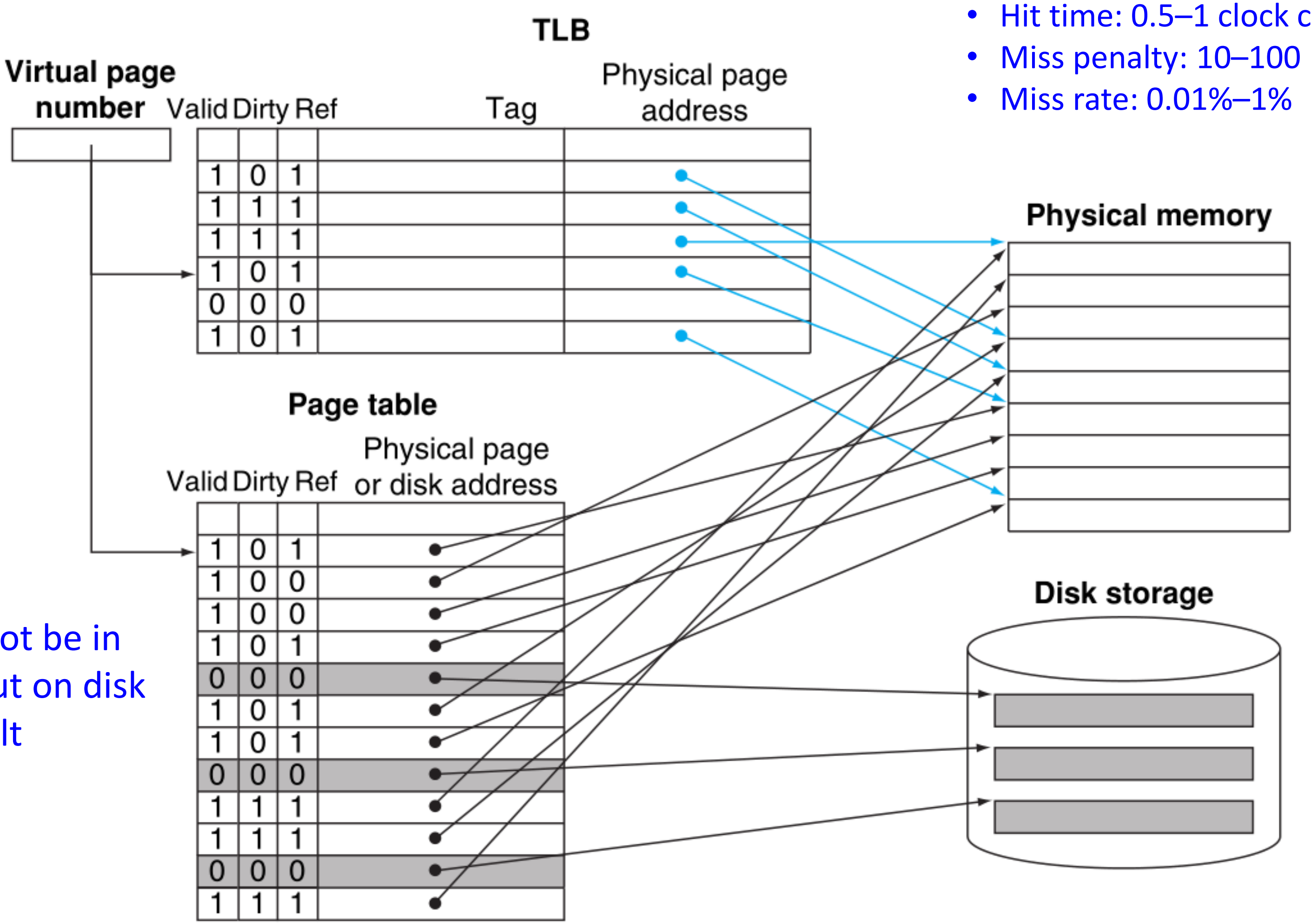
- Translation-lookaside buffer (TLB)
  - Special address translation cache
  - Contains subset of v-to-p page mappings



TLB (Table Lookaside Buffer)

- Since TLB is a cache, it has the tag field.
- If no entry is found in TLB, page table is searched.

- TLB size: 16–512 entries
- Block size: 1–2 page table entries (typically 4–8 bytes each)
- Hit time: 0.5–1 clock cycle
- Miss penalty: 10–100 clock cycles
- Miss rate: 0.01%–1%



TLB

Valid	Dirty	Ref	Tag	Physical page address
1	0	1		●
1	1	1		●
1	1	1		●
1	0	1		●
0	0	0		
1	0	1		●

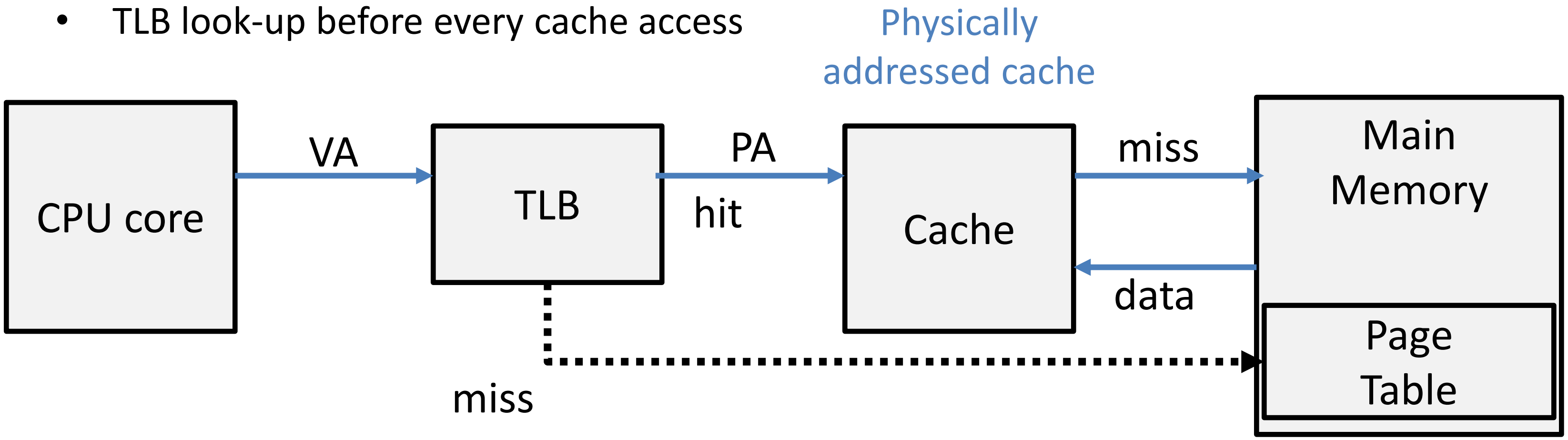
Page table

Valid	Dirty	Ref	Physical page or disk address
1	0	1	●
1	0	0	●
1	0	0	●
1	0	1	●
0	0	0	●
1	0	1	●
1	0	1	●
0	0	0	●
1	1	1	●
1	1	1	●
0	0	0	●
1	1	1	●

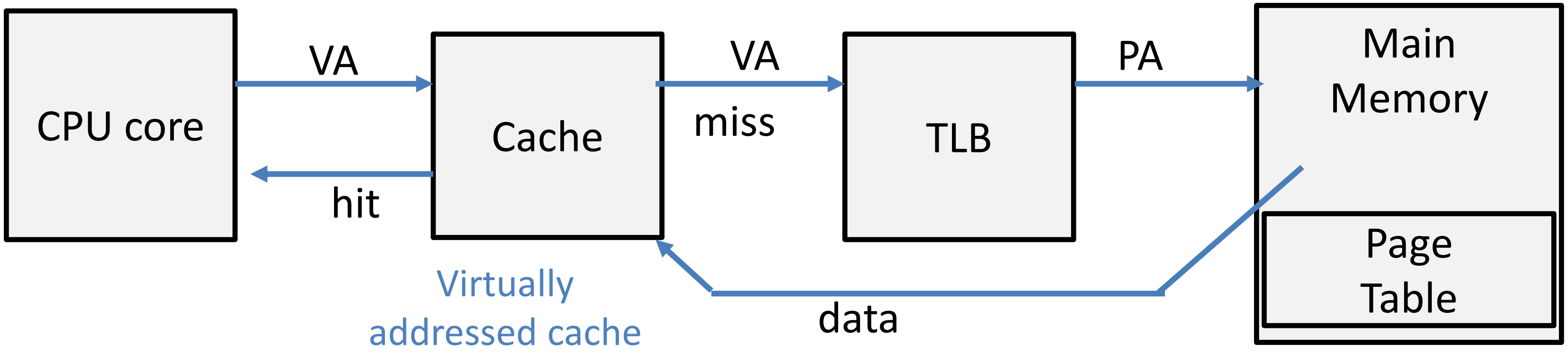
Page may not be in memory, but on disk
→ Page fault

TLB, CPU and Memory

- Physically addressed cache
  - TLB look-up before every cache access

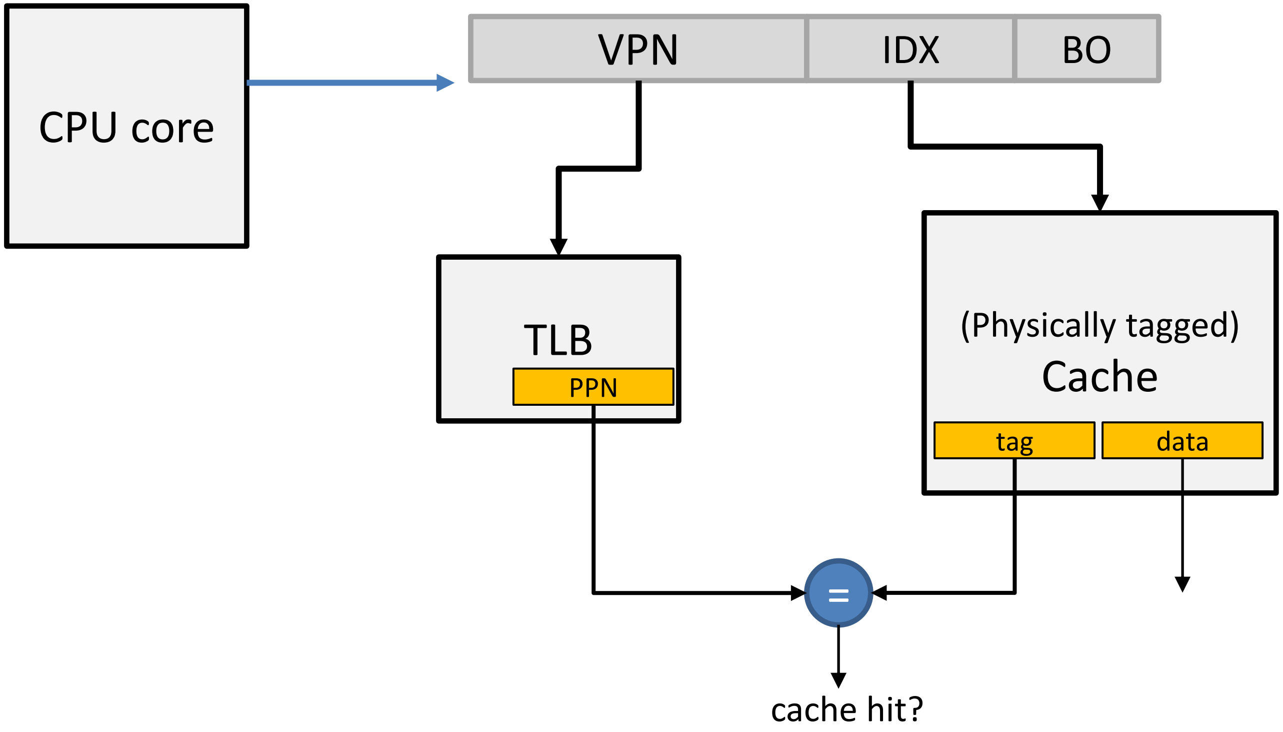


- Virtually addressed cache
  - Cache must be flushed frequently



TLB, CPU and Memory

- Virtually-indexed physically tagged cache (VIPT)
  - Cache index bit should not overlap with tag part
  - TLB look-up and cache index search runs in parallel
  - If cache becomes large, aliasing problem arises.
  - Most processors use VIPT for L1 cache


CPU core
VPN
IDX
BO
TLB
PPN
(Physically tagged)
Cache
tag
data
=
cache hit?