Chapter 5: Memory Systems

Part 2

(contain OHPs by H&P, Morgan Kaufmann)

Virtual Memory Management
(OS Topic)
(Assume your knowledge of OS)

Memory Management (반복)

- Cache memory management (Architecture topic)
  - Cache part of main memory
  - Implemented by hardware: fast, simple
  - † May think it as part of processor, i.e., on-chip cache
    - Hardware accelerator: OS not know about it
- Virtual memory management (OS topic)
  - Use main memory as cache for disk
  - Implemented by software
    - Disk access is already slow (10ms)

† Principles (caching, locality, management) same for both

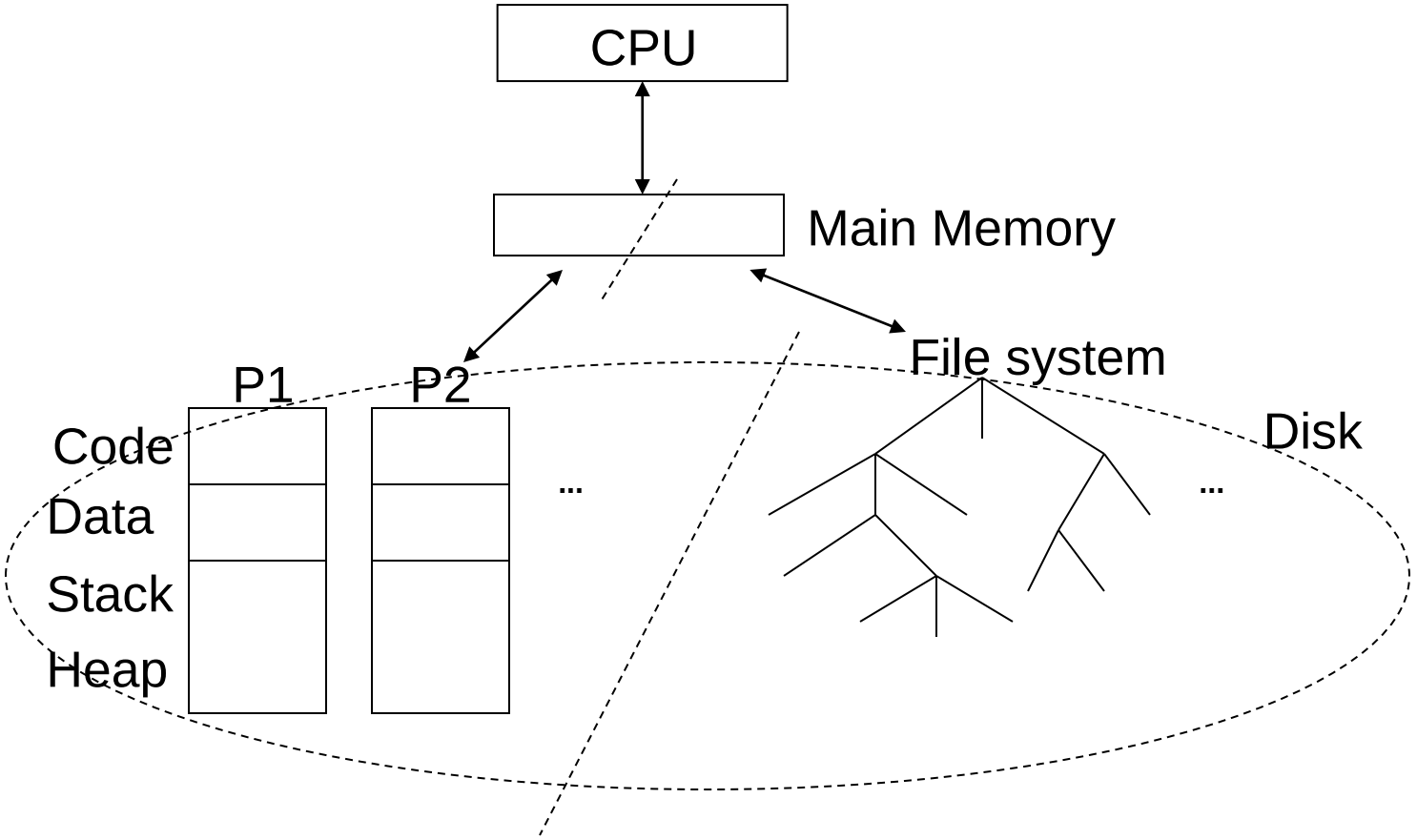
- Usage: independent of each other

Memory Management

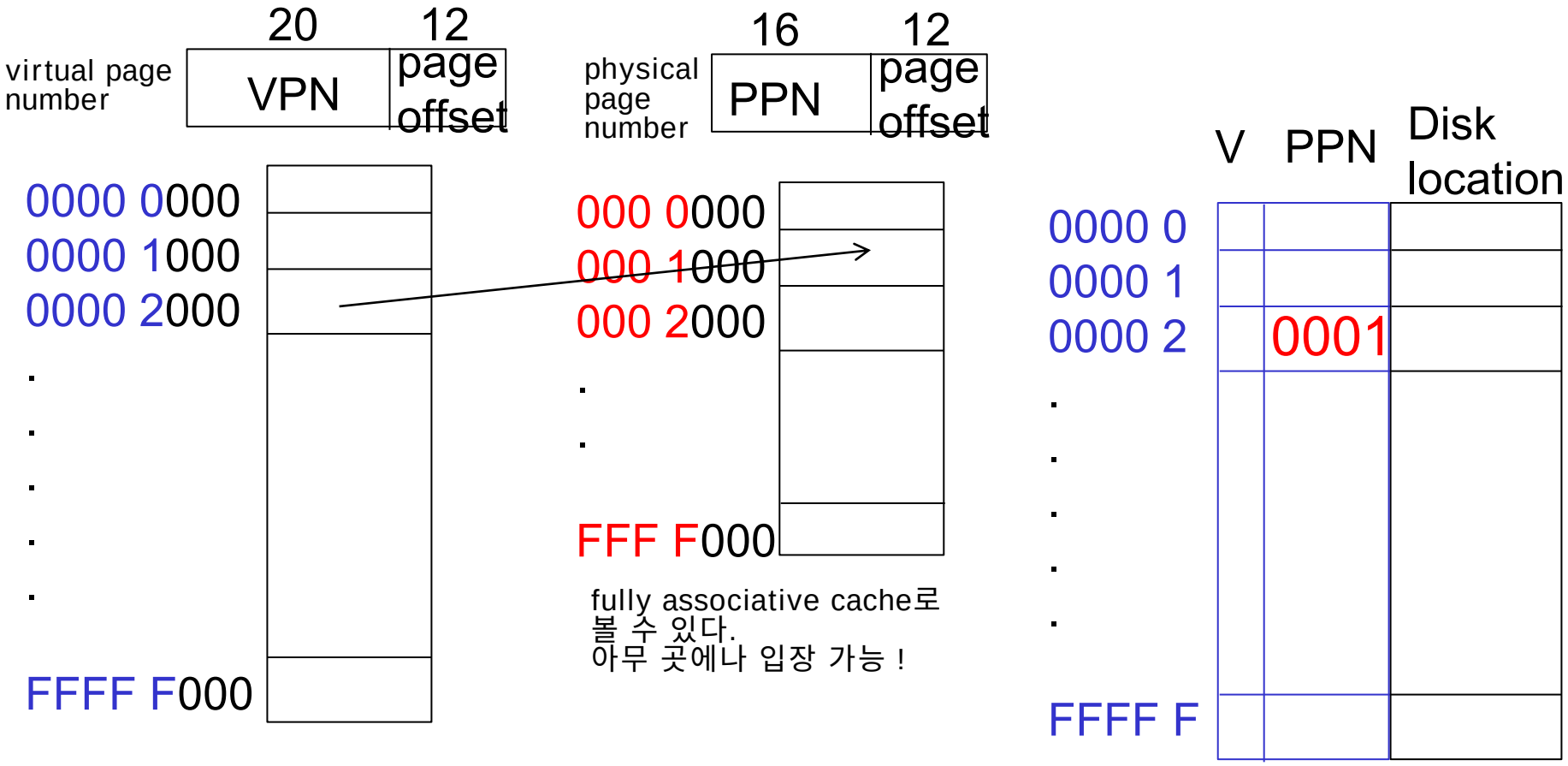
- Forget about cache memory for now
  - It is hardware accelerator, part of CPU
  - With cache, CPU feel that main memory is faster
- Virtual memory management
  - Use main memory as cache for disk
  - Focus only on main memory and disk

† Separation of concern

Ignore Cache Memory



Virtual Memory



Page size: 2^{12} = 4KB

Virtual space: 2^{32} = 4GB

Main memory: 2^{28} = 256MB

Page Table (per-process) mapping 정보 저장

Quiz

- What kind of mapping (placement) do we use?
  - Identification
    - Table data structure in software
- How do multiple user processes share main memory?
  - OS manage per-process page table
    - Size of page table matters (see textbook for more)
  - Protection and sharing

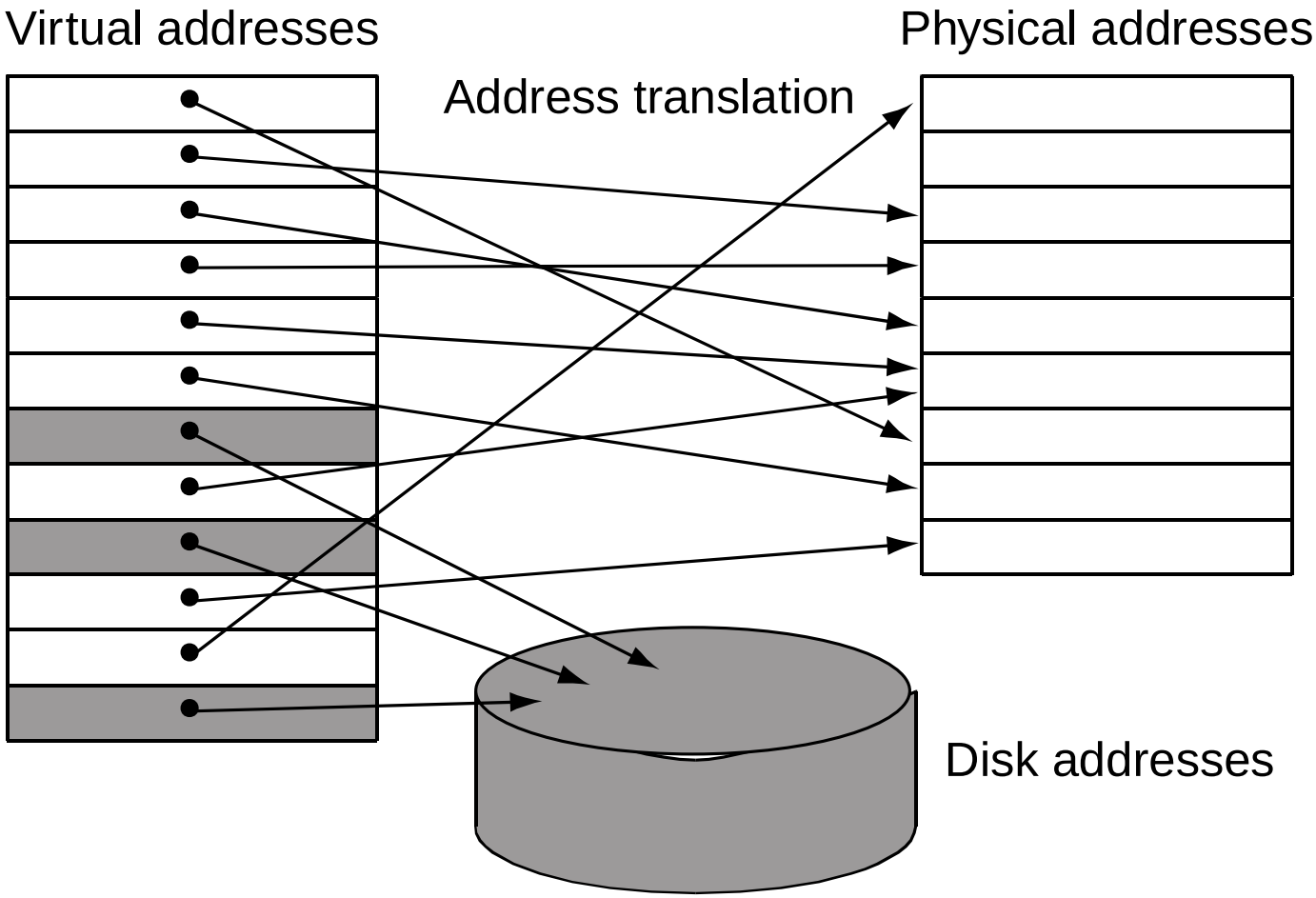
Motivations for VM (반복)

- Allow single program to exceed the size of main memory
  - Formerly, programmers divide program into pieces
    - Group them into overlays (modules)
  - Serious burden to programmers
    - † Today PM can be larger than VM, but there can be hundreds of processes
- Decouple main memory and process address space
  - Use main memory as cache
- Sharing of main memory among multiple programs
  - Efficient and safe (protection issue – more later)
- Simplify loading of the program for execution

Virtual Memory

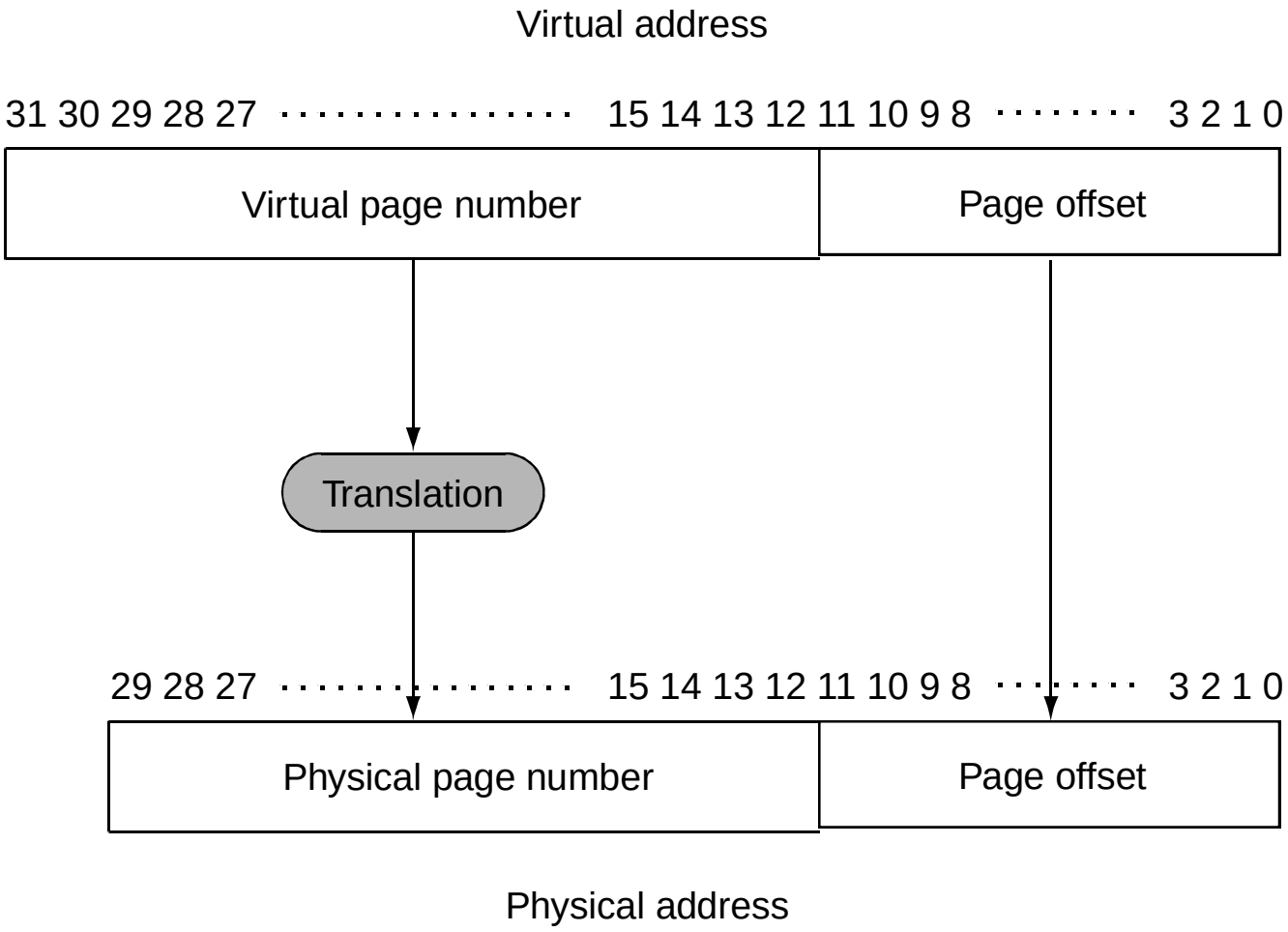
- When creating a process, OS:
  - Create space in disk or flash memory for all pages of process (swap space)
    - Create per-process page table
    - Record the location of each virtual page on disk
  - Also, track which processes and which virtual processes use each physical page
  - Page replacement (with approximated LRU)
    - Replaced pages written to disk
- Process (state of a program)
  - Page table (or PT register), program counter, registers

Virtual Memory

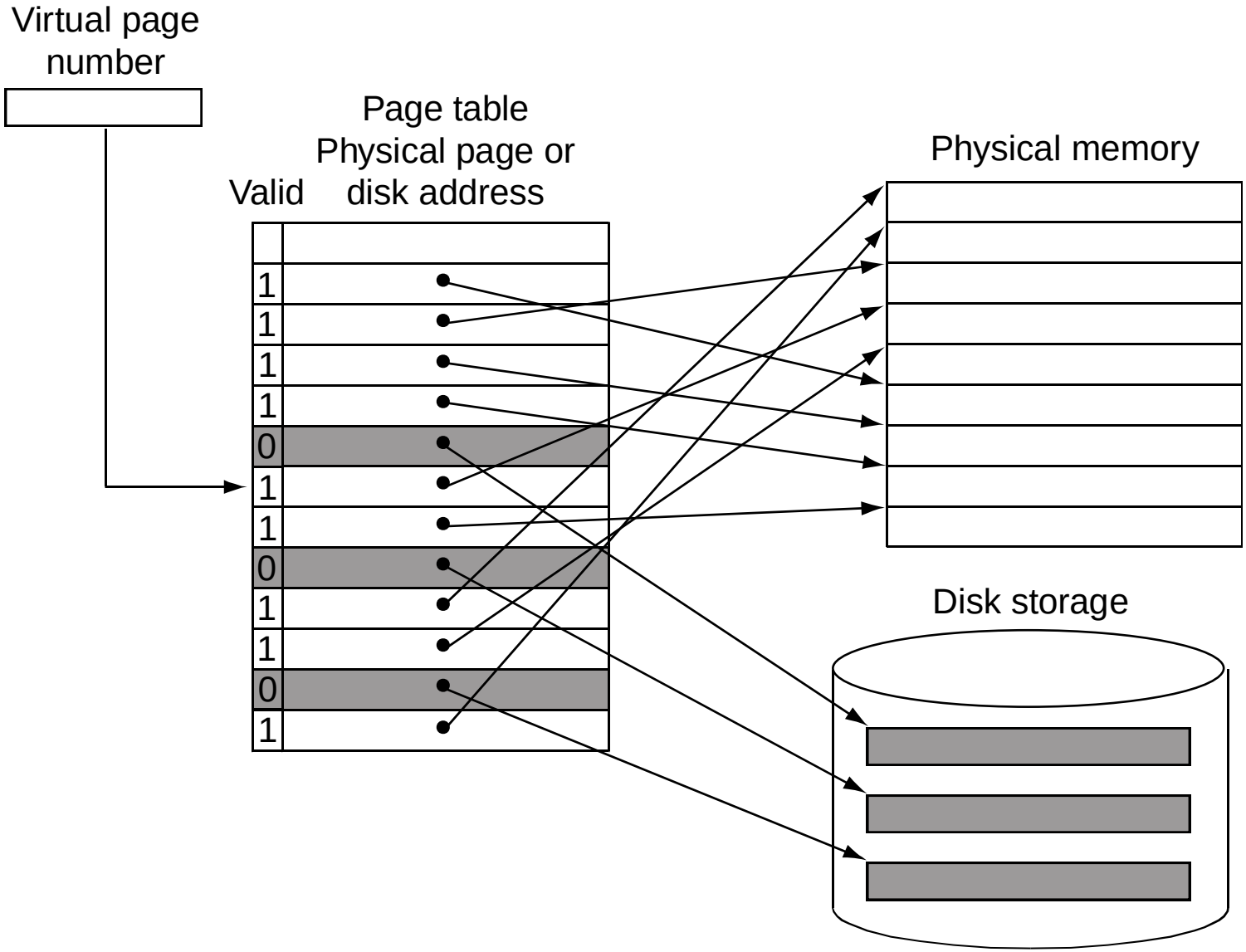


Address Translation (V-to-P)

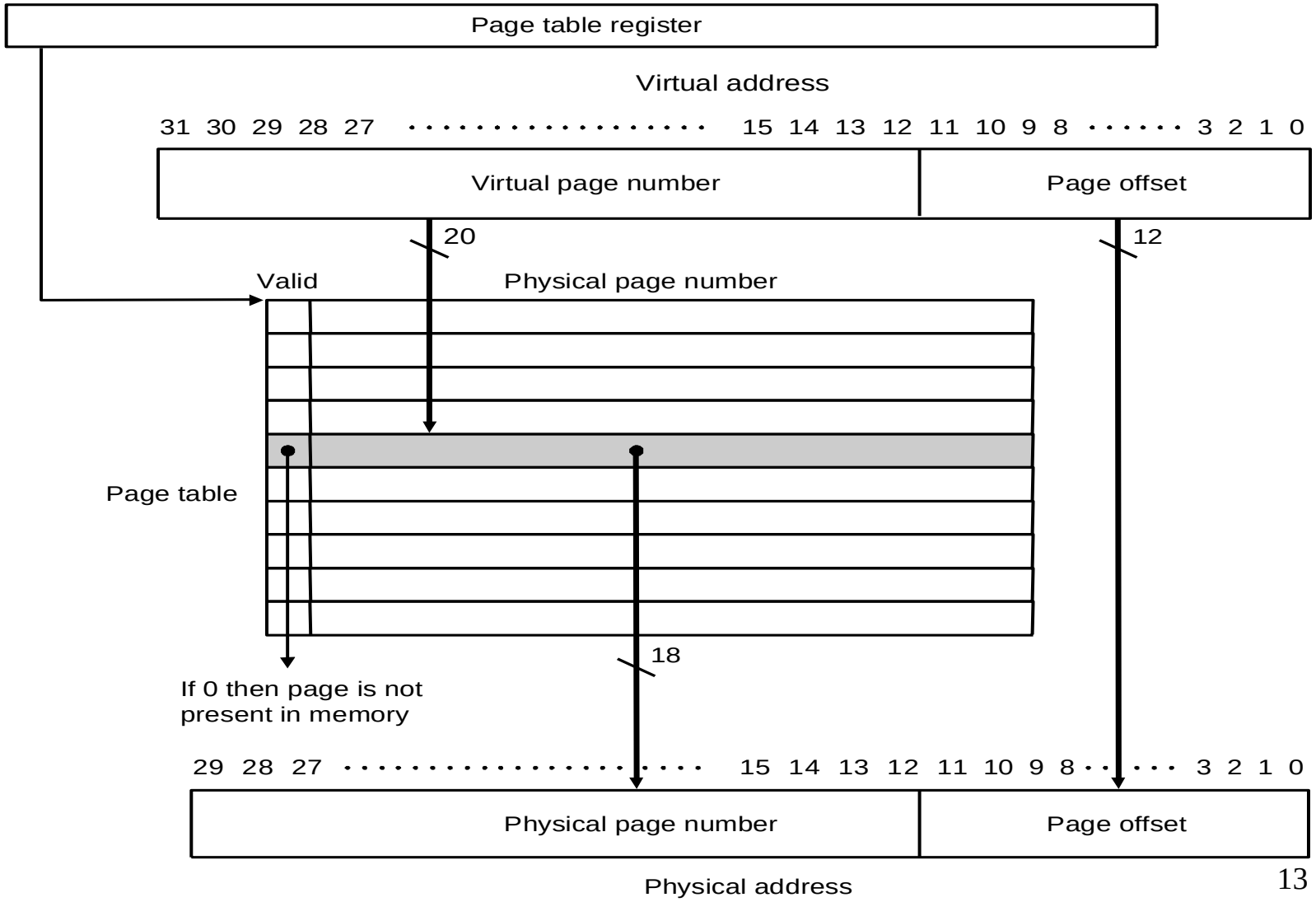
- Virtual-to-physical address translation



Per-Process Page Tables



Per-Process Page Tables



Page Tables

- Stores placement information
  - Array of page table entries, indexed by virtual page number
  - Page table register in CPU points to page table in physical memory
- If page is present in memory
  - PTE stores the physical page number
  - Plus other status bits (referenced, dirty, …)
- If page is not present (page fault)
  - PTE can refer to location in swap space on disk

Page Fault Penalty

- On page fault, the page must be fetched from disk
  - Takes millions of clock cycles
  - Handled by OS code
    - May switch process
- Try to minimize page fault rate
  - Pages should be large enough
    - From 4KB up to 64KB
      - † In embedded, go opposite to 1KB
  - Fully associative placement
  - Smart replacement algorithms

Four Issues in Virtual Memory

- Q1: placement
- Q2: identification
- Q3: write strategy
  - Disk write take millions of cycles
  - Block at once, not individual locations
  - Use write-back disk write 시간이 너무 오래 걸린다.
    - Dirty bit in PTE when page is written
- Q4: replacement policy
  - Some form of LRU
    - Always approximated

Replacement (skip)

- To reduce page fault rate, prefer LRU replacement
  - Reference bit (aka use bit) in PTE set to 1 on access to page
  - Periodically cleared to 0 by OS
  - A page with reference bit = 0 has not been used recently

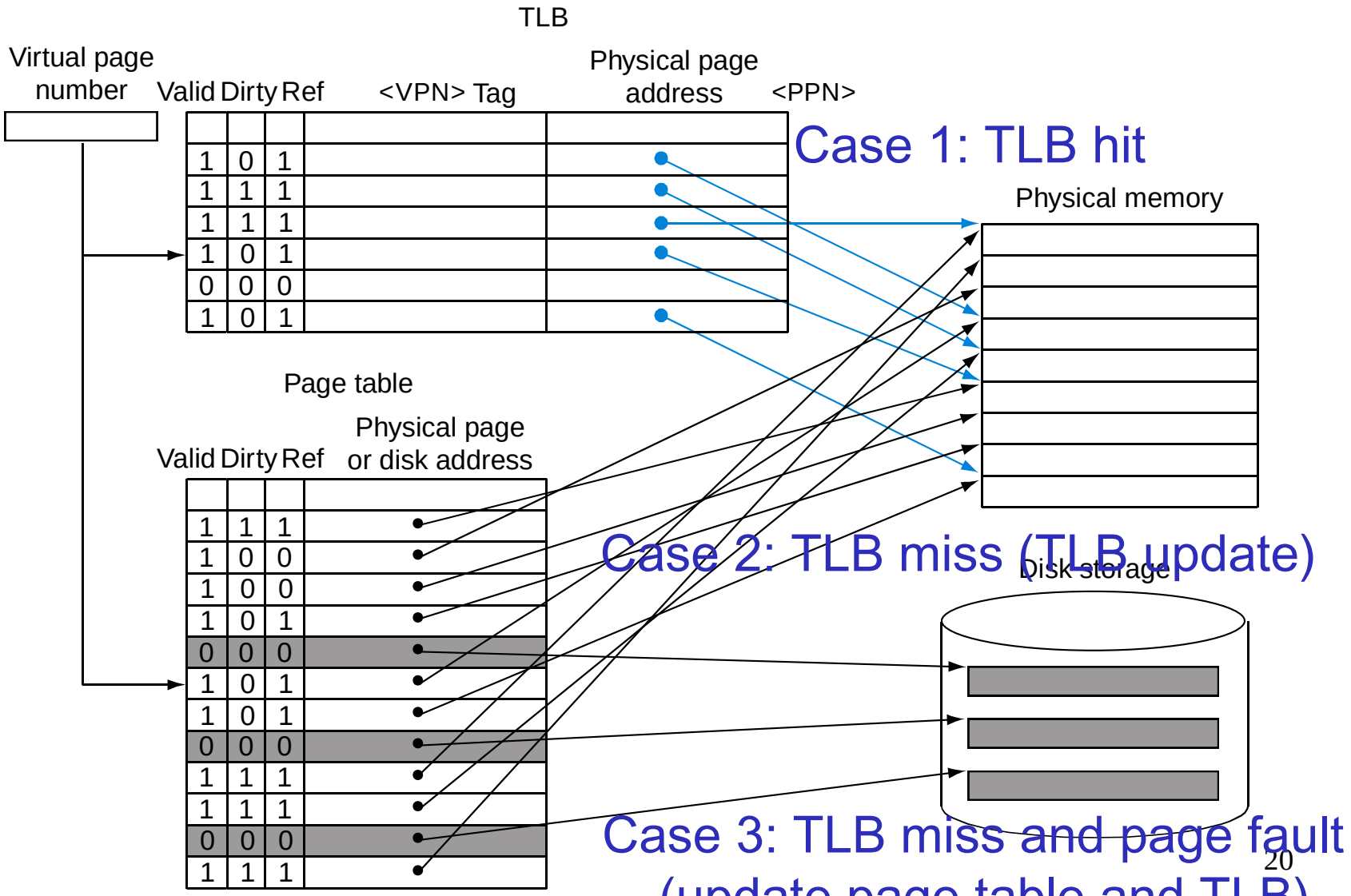
Cache and Virtual Memory

- Interactions

Virtual-to-Physical Address Translation

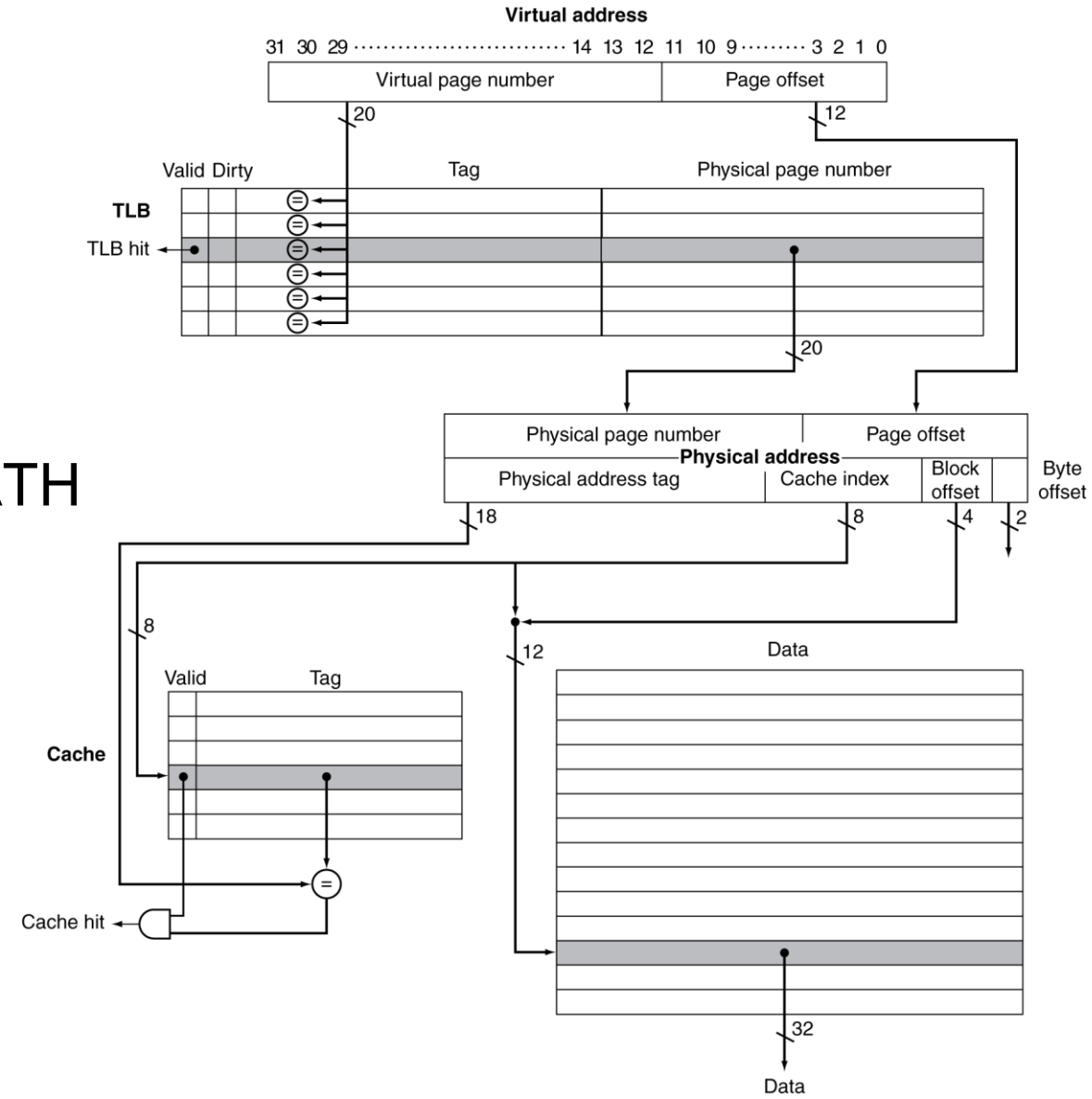
- Problem
  - Main memory access mean two accesses page table + physical memory
    - Address translation (page table in main memory)
    - Actual data or instruction access
- Solution
  - Use the idea of caching again page 크기가 크기 때문에 locality 크다 !
  - Translation look-aside buffer (TLB)
    - Small and fast cache within the CPU
    - Cache recently used VPN-PPN pairs (huge locality)

Making Address Translation Faster

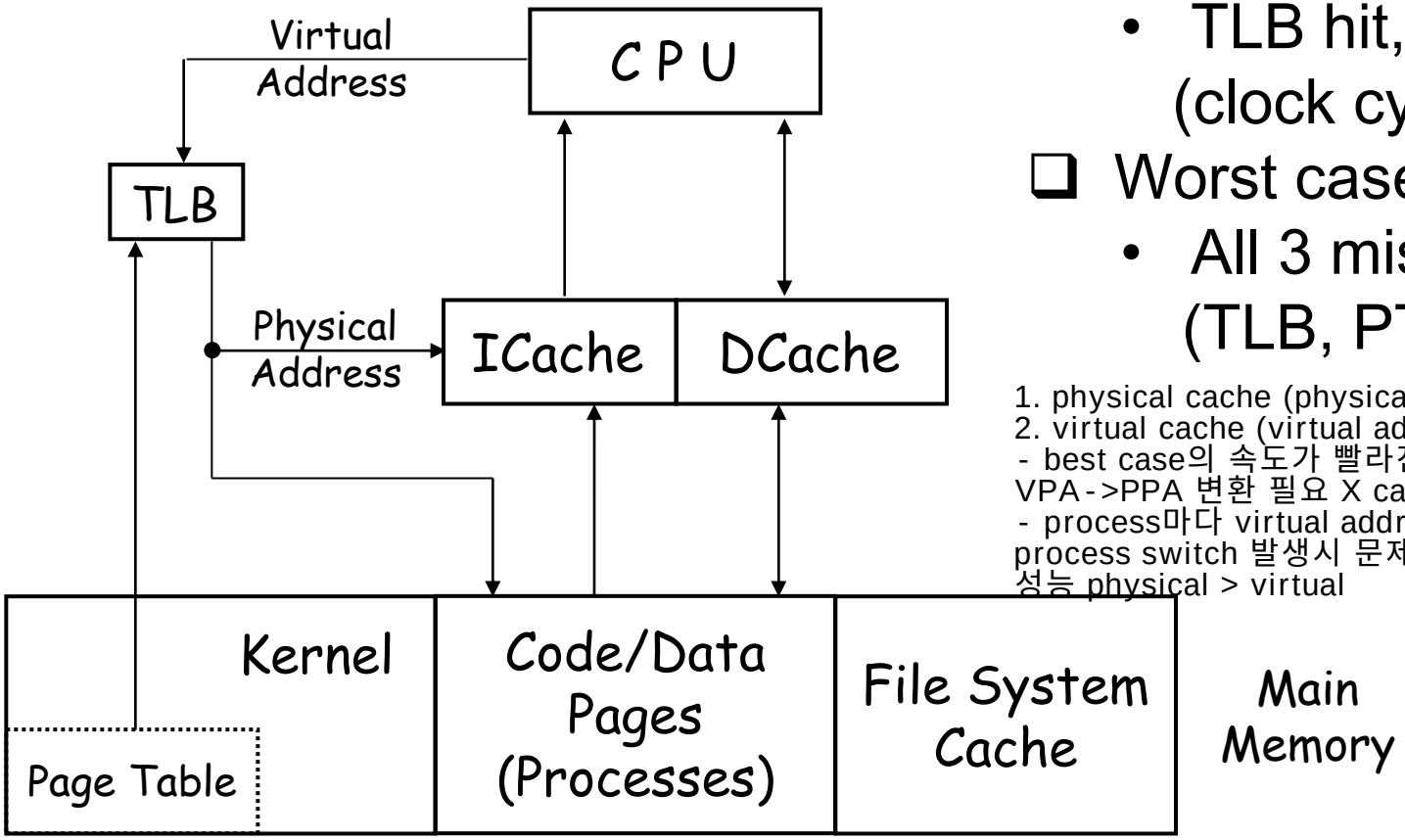


How does cache memory fit in?

Example:
Intrinsity FastMATH



Full Picture: Cache and Virtual Memory



- Best case
  - TLB hit, cache hit (clock cycle time)
- Worst case
  - All 3 misses (TLB, PT, cache)

1. physical cache (physical address 사용)
2. virtual cache (virtual address 사용)
- best case의 속도가 빨라진다.
VPA->PPA 변환 필요 X cache hit time만 걸림
- process마다 virtual address 가지기 때문에 process switch 발생시 문제 발생 !
성능 physical > virtual

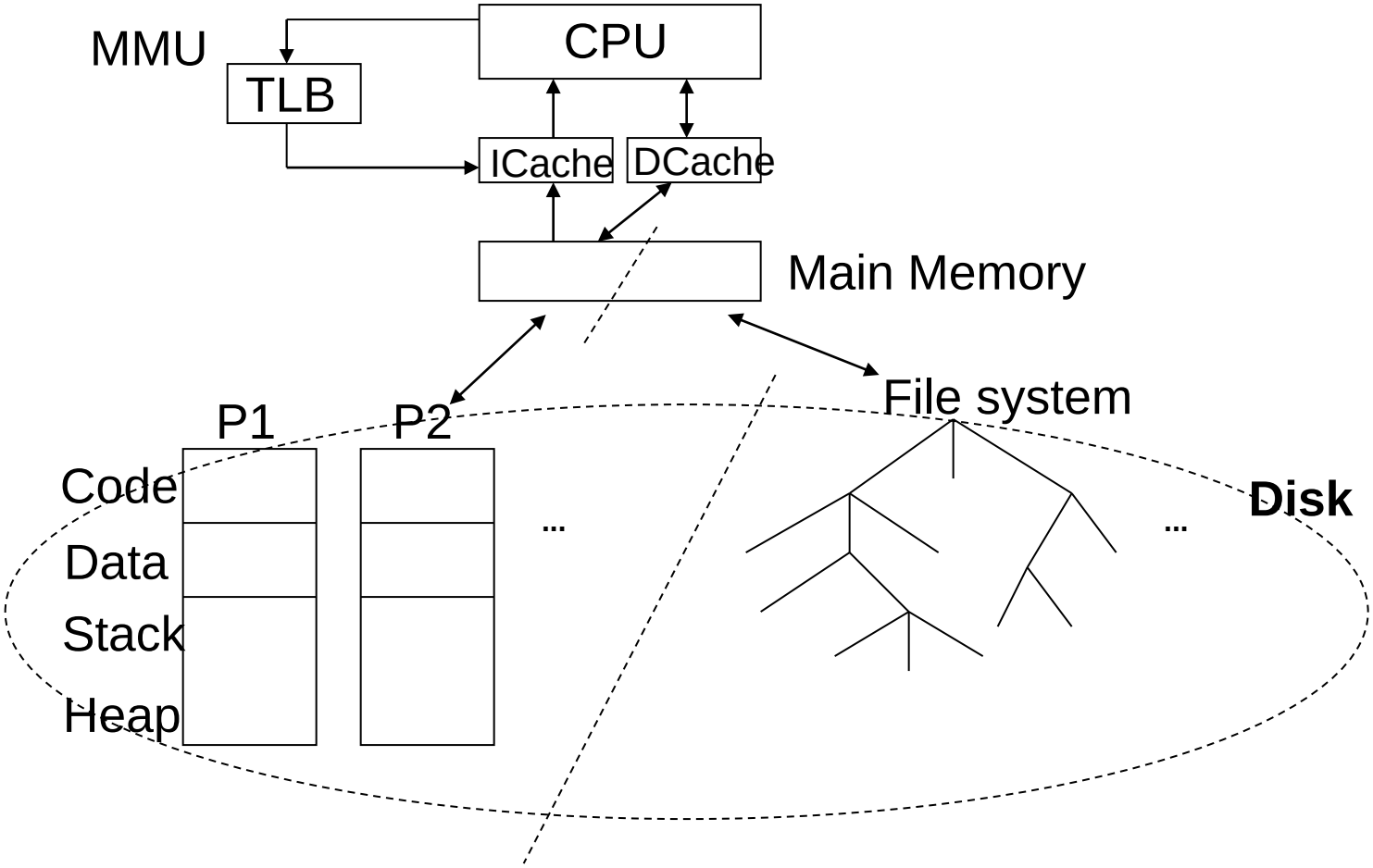
TLB, Cache, Main Memory

- TLB hit and cache hit: best case (clock cycle time)
- On TLB miss
  - Page table hit
    - Address translation using page table in main
    - Update TLB with new VPN-PPN pair
  - Page table miss (page fault)
    - OS handles fetching the page, updates the page table and TLB (process switch may occur)
    - Then restarts the faulting instruction
- On cache miss
  - Check with main memory

Fast Translation Using TLB

- Typical values for TLB
  - Size: 16 – 512 entries
  - Block size: 1-2 page table entries (4 – 8 bytes each)
  - Hit time: 0.5 – 1 cycle
  - Miss rate: 0.01% - 1%, miss penalty: 10 – 100 cycles
- Wide varieties of associativity in TLBs
  - Some use small, fully associative TLBs
    - Others use large TLBs, often with small associativity
  - With high associativity, hardware LRU is expensive
    - Many use random replacement policy (free counter)

General-Purpose Computer



Key Quantitative Design Parameters

(참고자료)

Feature	Typical values for L1 caches	Typical values for L2 caches	Typical values for paged memory	Typical values for a TLB
Total size in blocks	250–2000	2500–25,000	16,000–250,000	40–1024
Total size in kilobytes	16–64	125–2000	1,000,000–1,000,000,000	0.25–16
Block size in bytes	16–64	64–128	4000–64,000	4–32
Miss penalty in clocks	10–25	100–1000	10,000,000–100,000,000	10–1000
Miss rates (global for L2)	2%–5%	0.1%–2%	0.00001%–0.0001%	0.01%–2%

Cache Performance:

Three Cs Model

Sources of Misses

- Compulsory misses (aka cold start misses)
  - First access to a block
- Capacity misses
  - Due to finite cache size
  - A replaced block is later accessed again
- Conflict misses (aka collision misses)
  - In a non-fully associative cache
  - Due to competition for entries in a set
  - Would not occur in a fully associative cache of the same total size

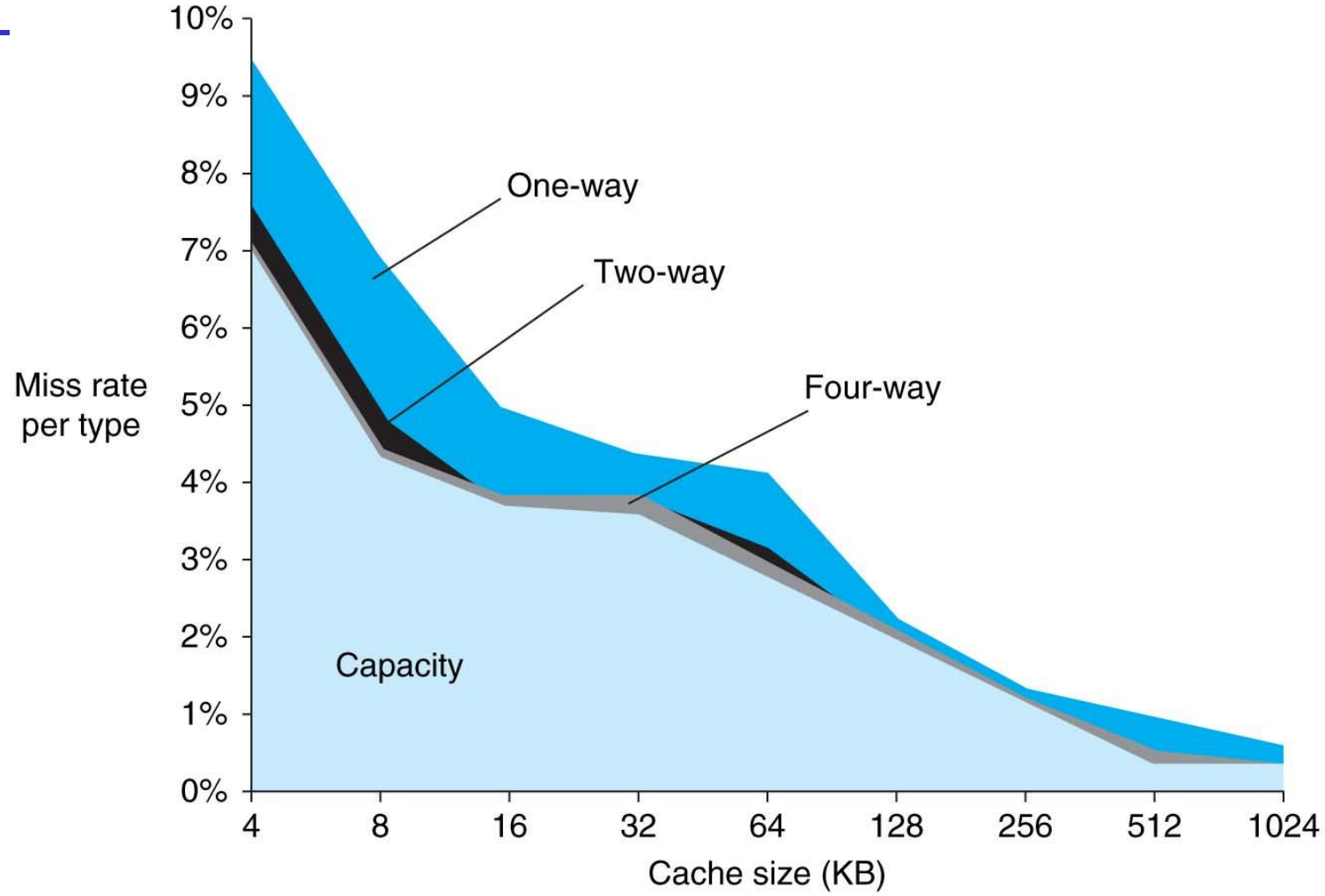
1. compulsory miss
- 계산 가능
2. capacity miss
- cache 크기 고정 시키고 fully associative로 계산하면 capacity miss 계산 가능
3. conflict miiss
- 전체 - (1+2)

Cache Design Trade-offs

Design change	Effect on miss rate	Negative performance effect
Increase cache size	Decrease capacity misses	May increase access time
Increase associativity	Decrease conflict misses	May increase access time
Increase block size	Decrease compulsory misses	Increases miss penalty. For very large block size, may increase miss rate due to pollution.

Cache size (KB)	Degree associative	Total miss rate	Miss rate components (relative percent) (sum = 100% of total miss rate)					
			Compulsory		Capacity		Conflict	
4	1-way	0.098	0.0001	0.1%	0.070	72%	0.027	28%
4	2-way	0.076	0.0001	0.1%	0.070	93%	0.005	7%
4	4-way	0.071	0.0001	0.1%	0.070	99%	0.001	1%
4	8-way	0.071	0.0001	0.1%	0.070	100%	0.000	0%
8	1-way	0.068	0.0001	0.1%	0.044	65%	0.024	35%
8	2-way	0.049	0.0001	0.1%	0.044	90%	0.005	10%
8	4-way	0.044	0.0001	0.1%	0.044	99%	0.000	1%
8	8-way	0.044	0.0001	0.1%	0.044	100%	0.000	0%
16	1-way	0.049	0.0001	0.1%	0.040	82%	0.009	17%
16	2-way	0.041	0.0001	0.2%	0.040	98%	0.001	2%
16	4-way	0.041	0.0001	0.2%	0.040	99%	0.000	0%
16	8-way	0.041	0.0001	0.2%	0.040	100%	0.000	0%
32	1-way	0.042	0.0001	0.2%	0.037	89%	0.005	11%
32	2-way	0.038	0.0001	0.2%	0.037	99%	0.000	0%
32	4-way	0.037	0.0001	0.2%	0.037	100%	0.000	0%
32	8-way	0.037	0.0001	0.2%	0.037	100%	0.000	0%
64	1-way	0.037	0.0001	0.2%	0.028	77%	0.008	23%
64	2-way	0.031	0.0001	0.2%	0.028	91%	0.003	9%
64	4-way	0.030	0.0001	0.2%	0.028	95%	0.001	4%
64	8-way	0.029	0.0001	0.2%	0.028	97%	0.001	2%
128	1-way	0.021	0.0001	0.3%	0.019	91%	0.002	8%
128	2-way	0.019	0.0001	0.3%	0.019	100%	0.000	0%
128	4-way	0.019	0.0001	0.3%	0.019	100%	0.000	0%
128	8-way	0.019	0.0001	0.3%	0.019	100%	0.000	0%
256	1-way	0.013	0.0001	0.5%	0.012	94%	0.001	6%
256	2-way	0.012	0.0001	0.5%	0.012	99%	0.000	0%
256	4-way	0.012	0.0001	0.5%	0.012	99%	0.000	0%
256	8-way	0.012	0.0001	0.5%	0.012	99%	0.000	0%
512	1-way	0.008	0.0001	0.8%	0.005	66%	0.003	33%
512	2-way	0.007	0.0001	0.9%	0.005	71%	0.002	28%
512	4-way	0.006	0.0001	1.1%	0.005	91%	0.000	8%
512	8-way	0.006	0.0001	1.1%	0.005	95%	0.000	4%

Figure 5.14 Total miss rate for each size cache and percentage of each according to the "three C's."


10%
9%
8%
7%
6%
5%
4%
3%
2%
1%
0%
Miss rate per type
One-way
Two-way
Four-way
Capacity
4
8
16
32
64
128
256
512
1024
Cache size (KB)

Quantitative Memory Systems Design

- Cache simulation
  - Compulsory/capacity/conflict misses
    - The three C's
  - How to quantify them
  - Address trace
- Techniques to reduce miss penalty
- Techniques to reduce miss rate
- Techniques to reduce hit time

Sections not included in this note

- Section 5.5 Dependable memory hierarchy
- Section 5.6 Virtual machines
- Section 5.9 Control of a simple cache
- Section 5.10 Parallelism and memory hierarchy: cache coherence
- Section 5.11 Redundant arrays of inexpensive disks
- Section 5.12 Implementing cache controllers
- Section 5.13 Real stuff: ARM Cortex-A8, Intel Core i7
- Section 5.14 Going faster: cache blocking and matrix multiply