Santa Clara University

COEN 210,
Fall 2022

Chapter 5, part 3 – Virtual Memory

Yan Cui

ycui4@scu.edu

Announcements

- Homework 5
  - Due: March 19
- Presentation
  - Wednesday, Lecture time
  - 10 minutes Each group (8 mins for presentation + 2mins for questions)
  - Each of you should present at that day
- Report
  - Requirement on the documents
  - Due: March 24

Virtual Memory

Motivations

- **Physical memory may not be as large as "possible address space" spanned by a processor**, e.g.
  - A processor can address 4G bytes with 32-bit address
  - But installed main memory may only be 1GB

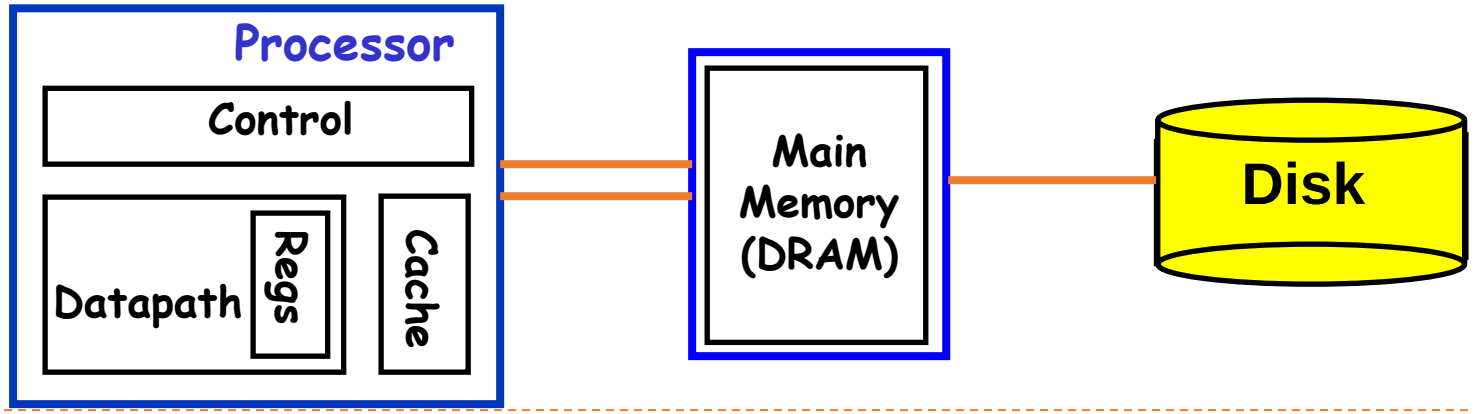
How if we want to simultaneously run many programs which require a total memory consumption greater than the installed main memory capacity?

- Terminology:
  - A running program is called a process or a thread
  - Operating System (OS) controls the processes

Motivations

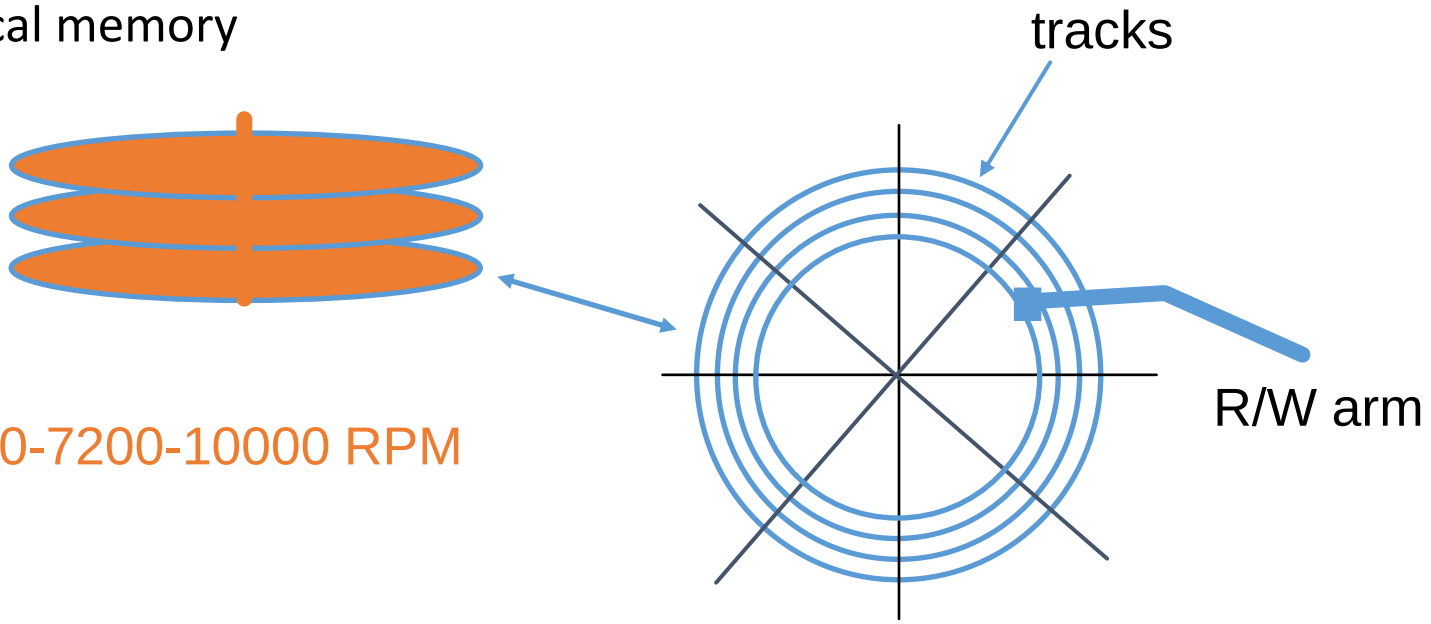
1. Allow program to be written without memory constraints
   - program can exceed the size of the main memory
2. Many Programs sharing DRAM Memory so that context switches can occur
3. Relocation: Parts of the program can be placed at different locations in the memory instead of a big chunk

- Virtual Memory:
  I. Main Memory holds many programs running at same time (processes)
  II. use Main Memory as a kind of “cache” for disk



Disk Technology in Brief

- Disk is mechanical memory

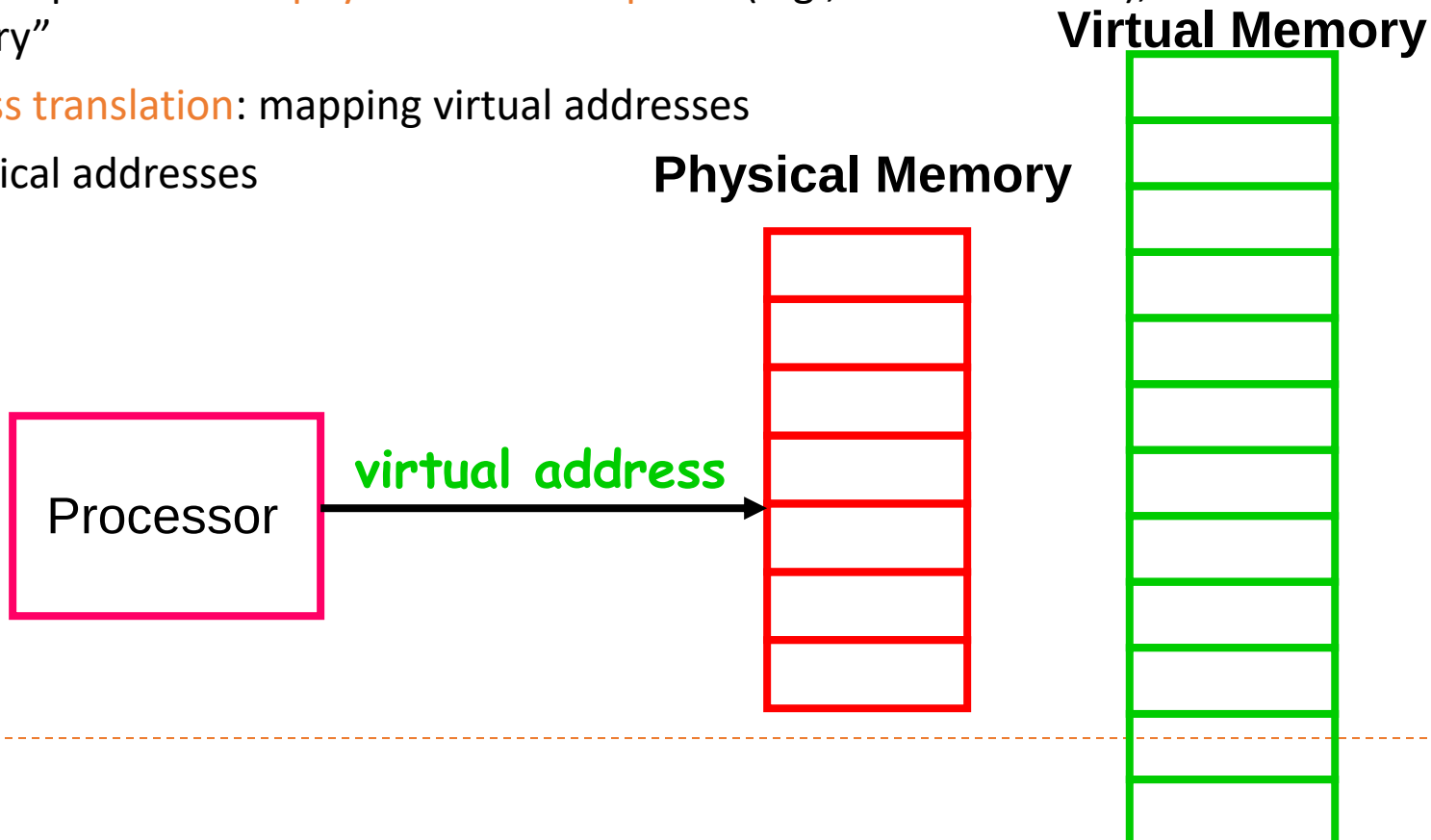


3600–4200-5400-7200-10000 RPM rotation speed

- Disk Access Time =
  seek time + rotational delay + transfer time
  - usually measured in milliseconds
- "Miss" to disk is extremely expensive
  - typical access time = millions of clock cycles
- Addressing a sector

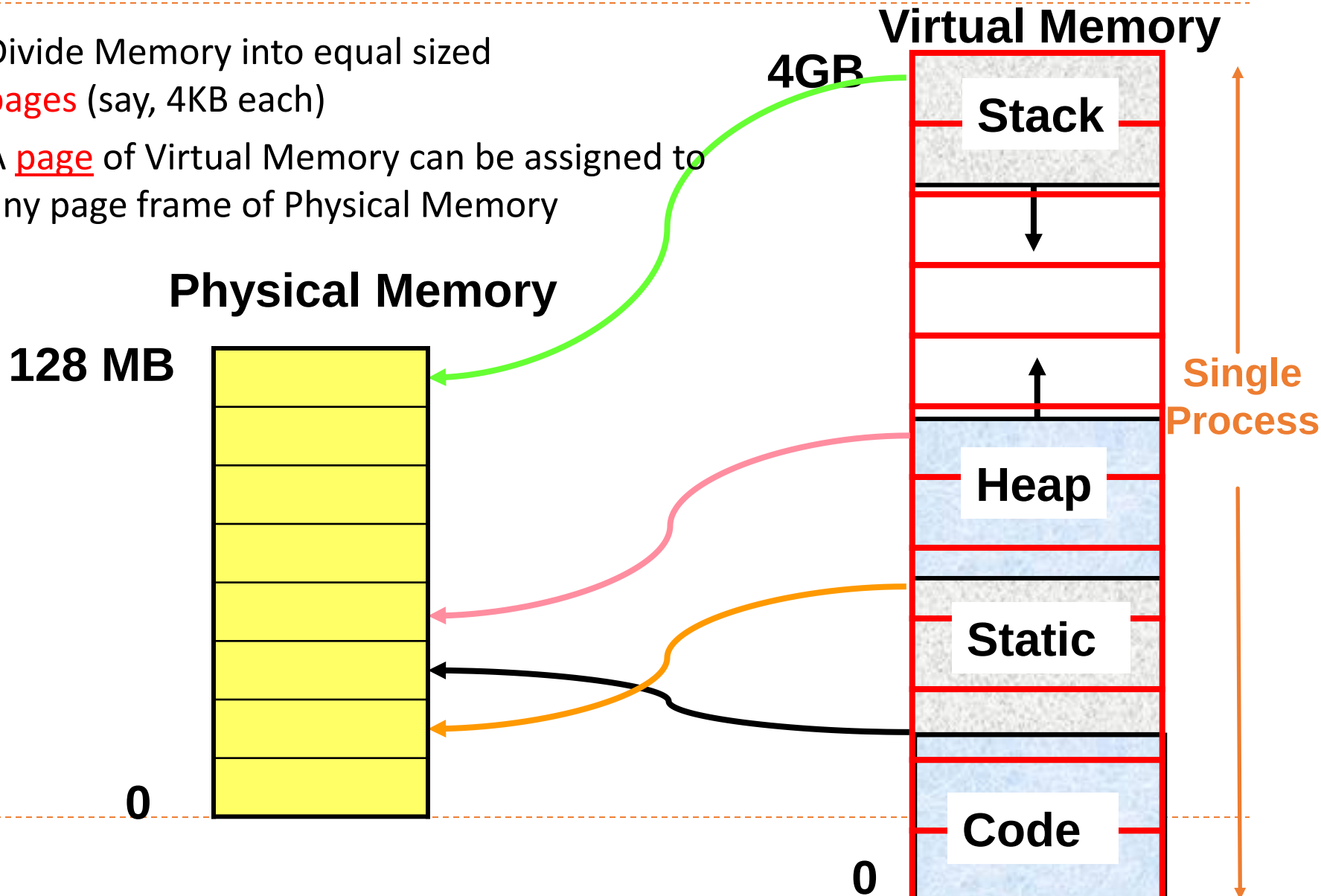
Virtual to Physical Memory mapping

- The processor generates virtual addresses while the memory is accessed using physical address.
- Each process has its own private "virtual address space" (e.g., 2^{32} Bytes); CPU actually generates "virtual addresses"
- Each computer has a "physical address space" (e.g., 128 MB DRAM); also called "real memory"
- Address translation: mapping virtual addresses to physical addresses



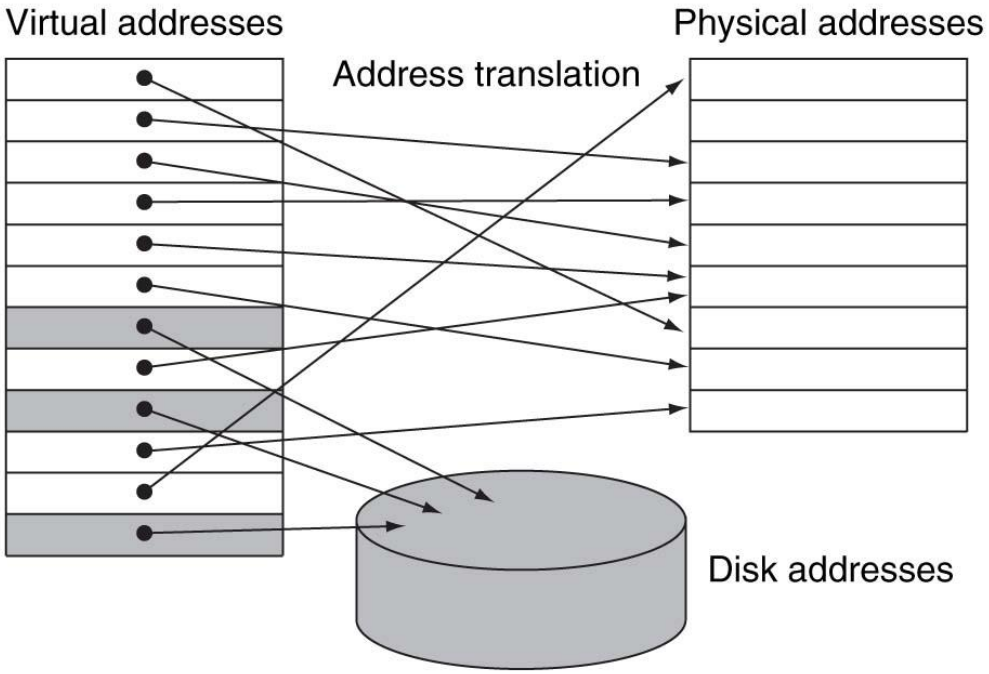
Mapping Virtual Memory to Physical Memory

- Divide Memory into equal sized pages (say, 4KB each)
- A page of Virtual Memory can be assigned to any page frame of Physical Memory



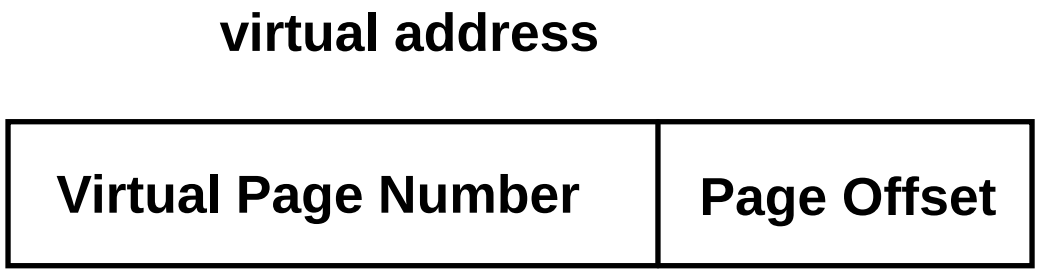
Virtual to Physical Memory mapping

- **Pages are mapped from virtual addresses to physical addresses.**
- **A virtual page is mapped to a physical page**
  - Allows some chunks of virtual memory to be present on disk, not in main memory
  - Allows multiple programs to use (different chunks of physical) memory at same time

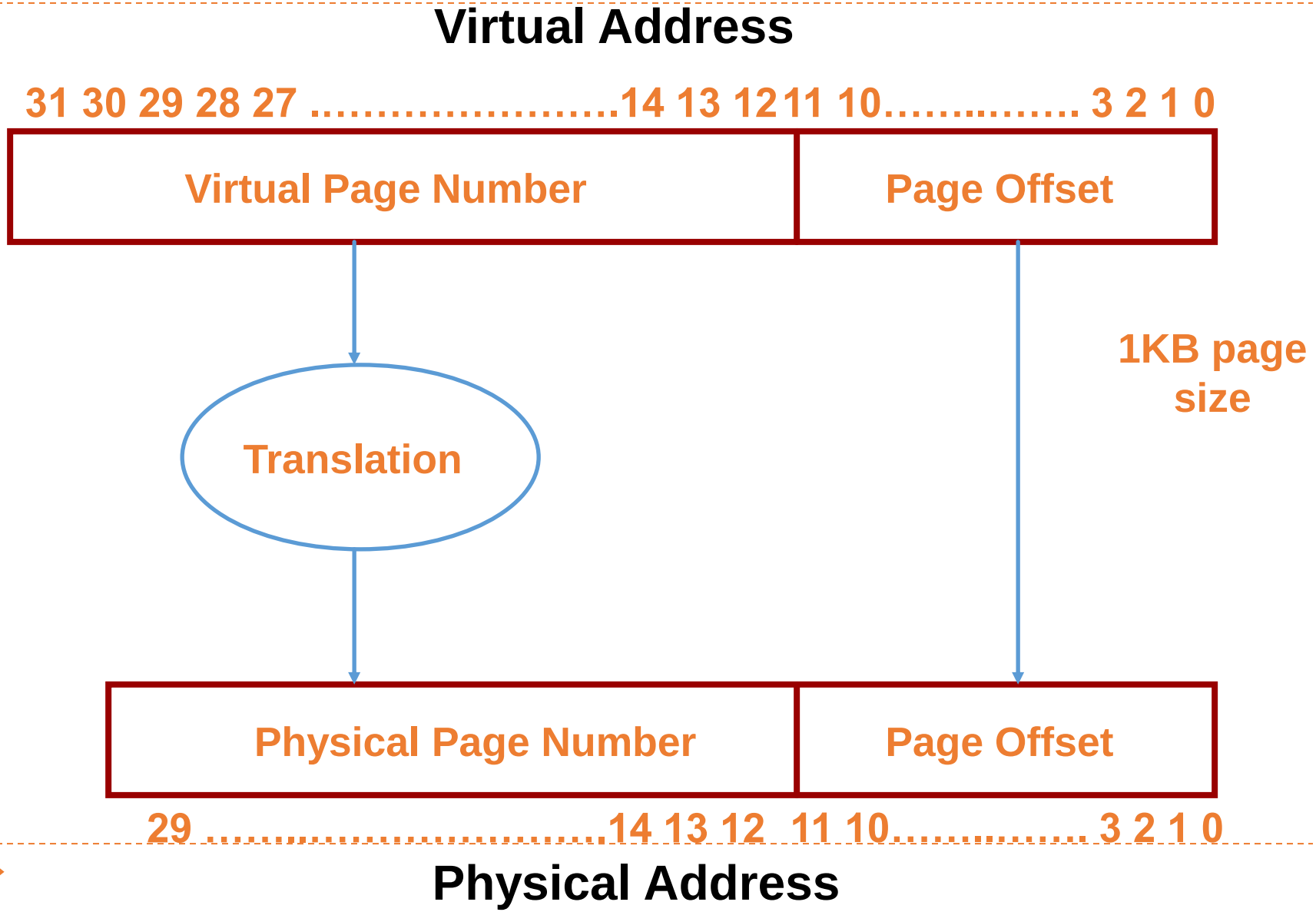


How to Perform Address Translation?

- VM divides memory into equal sized pages
- Address translation maps entire pages
  - offsets within the pages do not change
  - if page size is a power of two, the virtual address is separated into two fields:

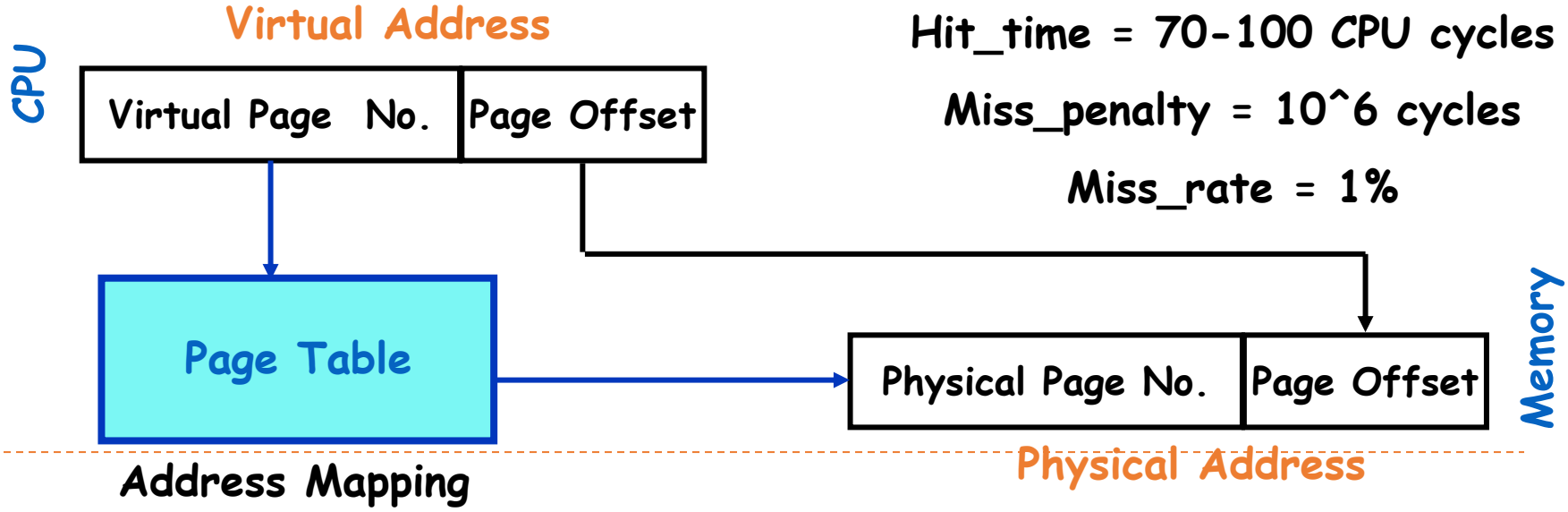


Mapping Virtual to Physical Address

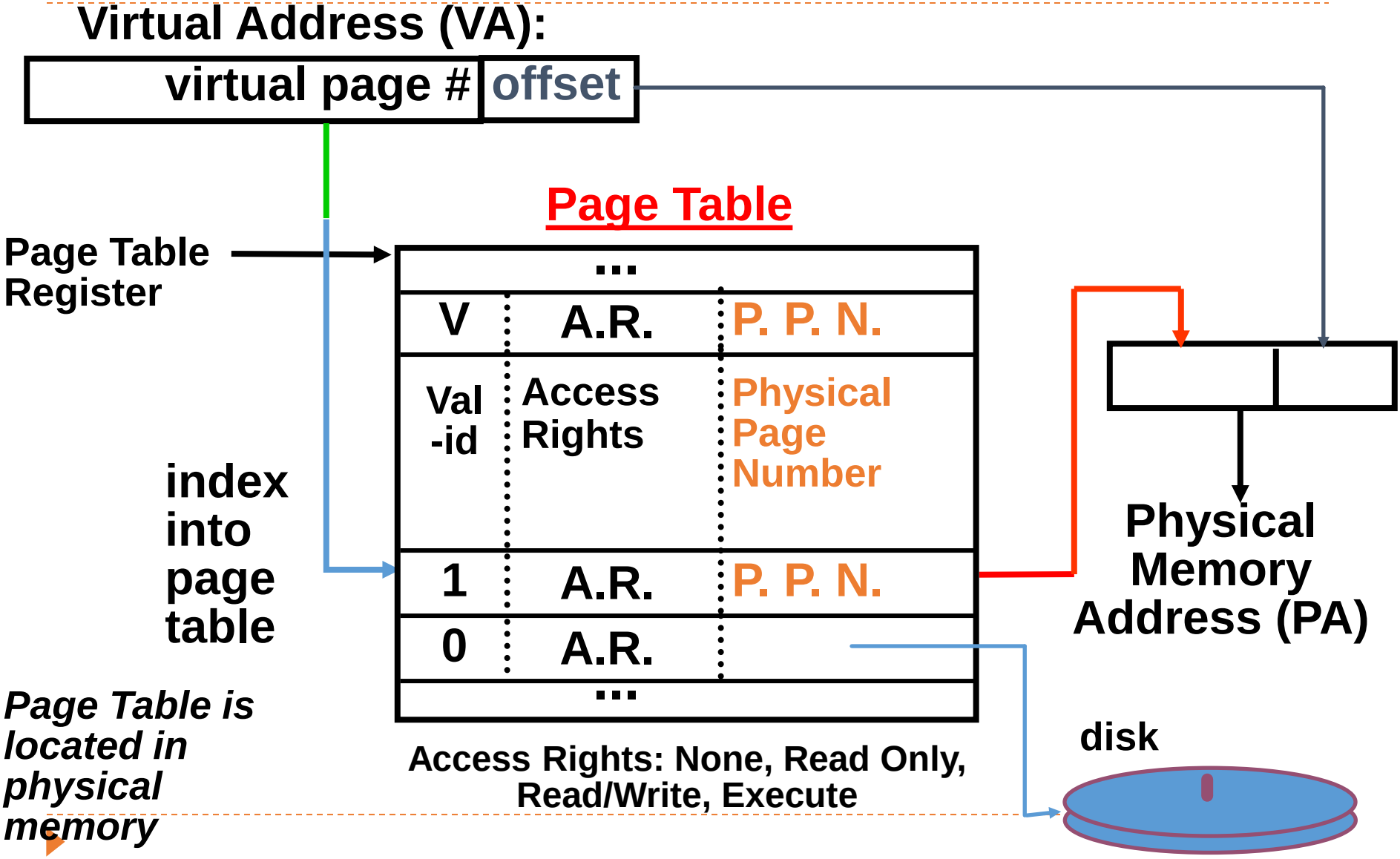


Address Translation

- Want fully associative page placement
- How to locate the physical page?
- A full Search impractical (too many pages)
- A page table is a data structure which contains the mapping of virtual pages to physical pages
  - There are several different ways, all up to the operating system, to keep and update this data
- Each process running in the system has its own page table



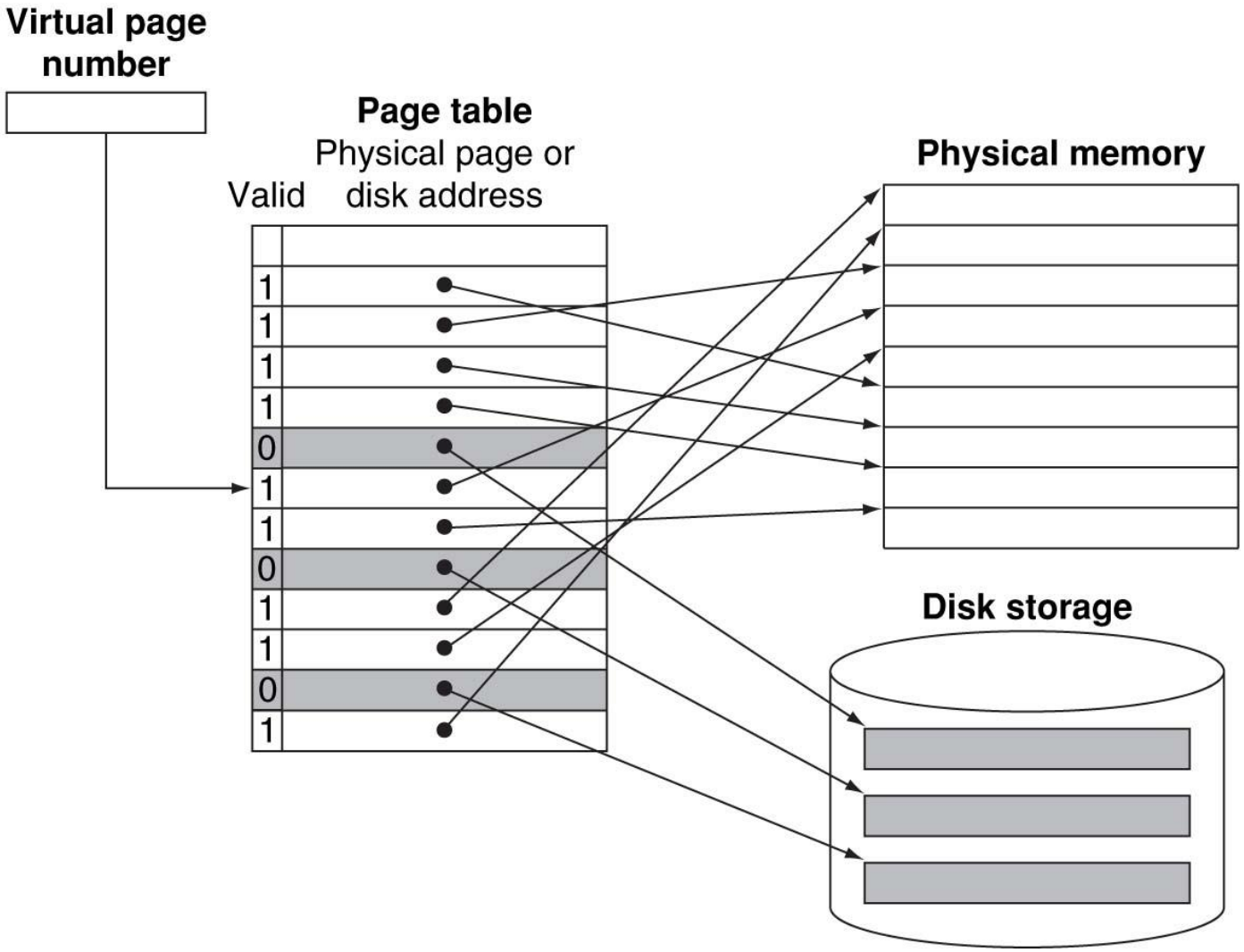
Address Translation: Page Table



Handling Page Faults

- A page fault is like a cache miss, if the page is not present in the memory.
  - Must find page in lower level of hierarchy
- If valid bit is zero, the Physical Page Number points to a page on disk
- When OS starts a new process, it creates space on disk for all the pages of the process, sets all valid bits in page table to zero, and all Physical Page Numbers to point to disk.
  - called Demand Paging - pages of the process are loaded from disk only as needed
  - If all the pages in main memory are in use, the OS choose a page to replace – LRU replacement scheme.

Page Table


Virtual page number
Page table
Physical page or disk address
Valid
1
1
1
1
0
1
1
0
1
1
0
1
Physical memory
Disk storage

Comparing the 2 hierarchies

Cache	Virtual Memory
Block or Line	Page
Miss	Page Fault
Block Size: 32-64B	Page Size: 4K-16KB
Placement: Direct Mapped, N-way Set Associative	Fully Associative
Replacement: LRU or Random	Least Recently Used (LRU) approximation
Write Thru or Back	Write Back
How Managed: Hardware	Hardware + Software (Operating System)

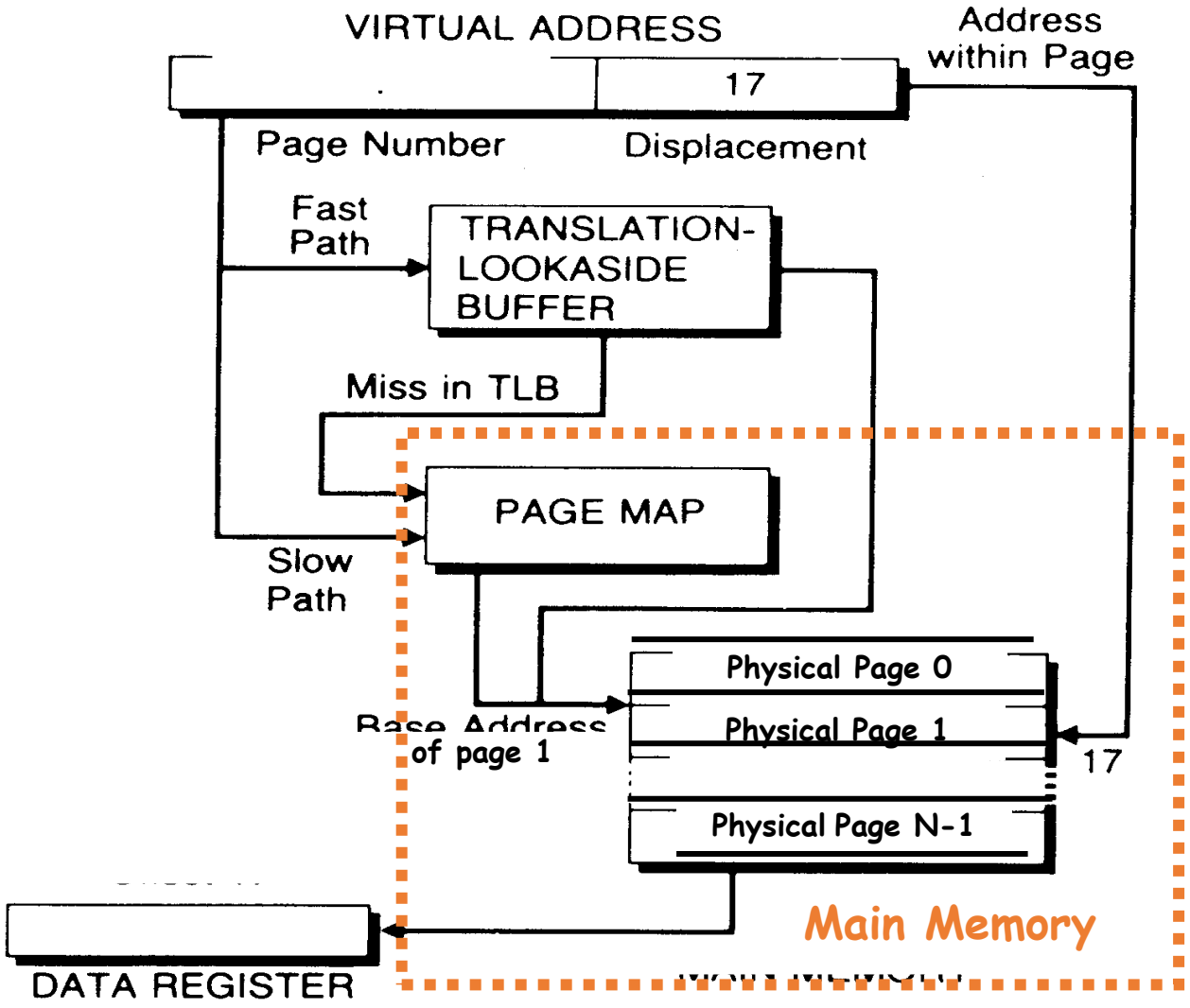
Motivation for Optimizing for Space

- Page Table too big!
- Stored in main memory
  - 4GB Virtual Address Space ÷ 4 KB page
    - 1 million Page Table Entries
    - 4 MB just for Page Table of single process!
- Variety of solutions to tradeoff Page Table size for slower performance
- Multilevel page table, Paging page tables, etc.
- (Take O/S Class to learn more)

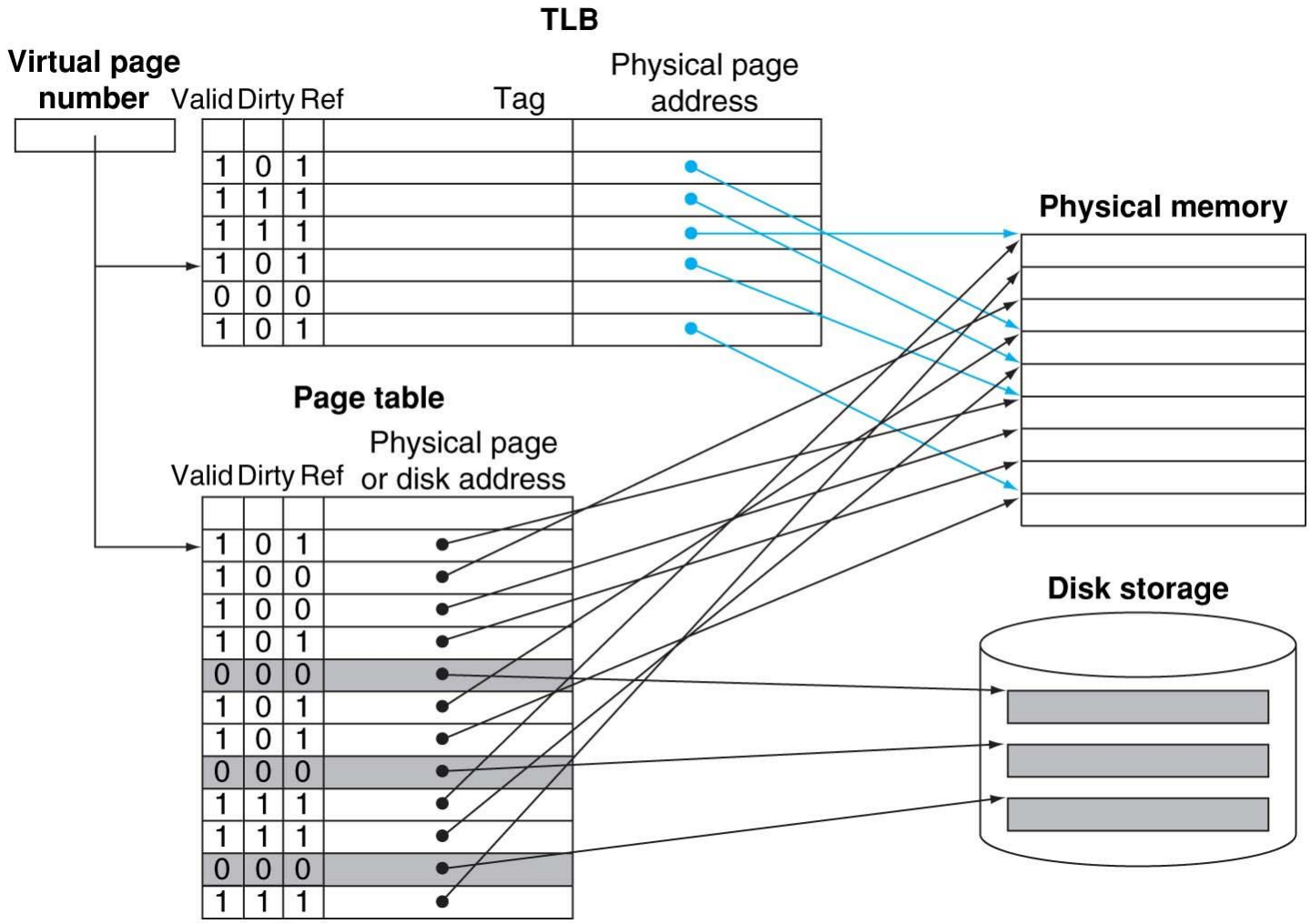
How to Translate Fast?

- Problem: Virtual Memory requires two memory accesses!
  - one to translate Virtual Address into Physical Address (page table lookup) - Page Table is in physical memory
  - one to transfer the actual data (hopefully cache hit)
- Observation: since there is locality in pages of data, must be locality in virtual addresses of those pages!
  - E.g. when a translation for a virtual page number is used, it will probably be needed again soon because the reference on that page have temporal and spatial locality.
- Why not create a cache of virtual to physical address translations to make translation fast? (smaller is faster)
- Modern processors includes a special cache that keeps track of recently used translations
- For historical reasons, such a “page table cache” is called a Translation Lookaside Buffer, or TLB

Translation-Lookaside Buffer (TLB)



TLB and Page Table



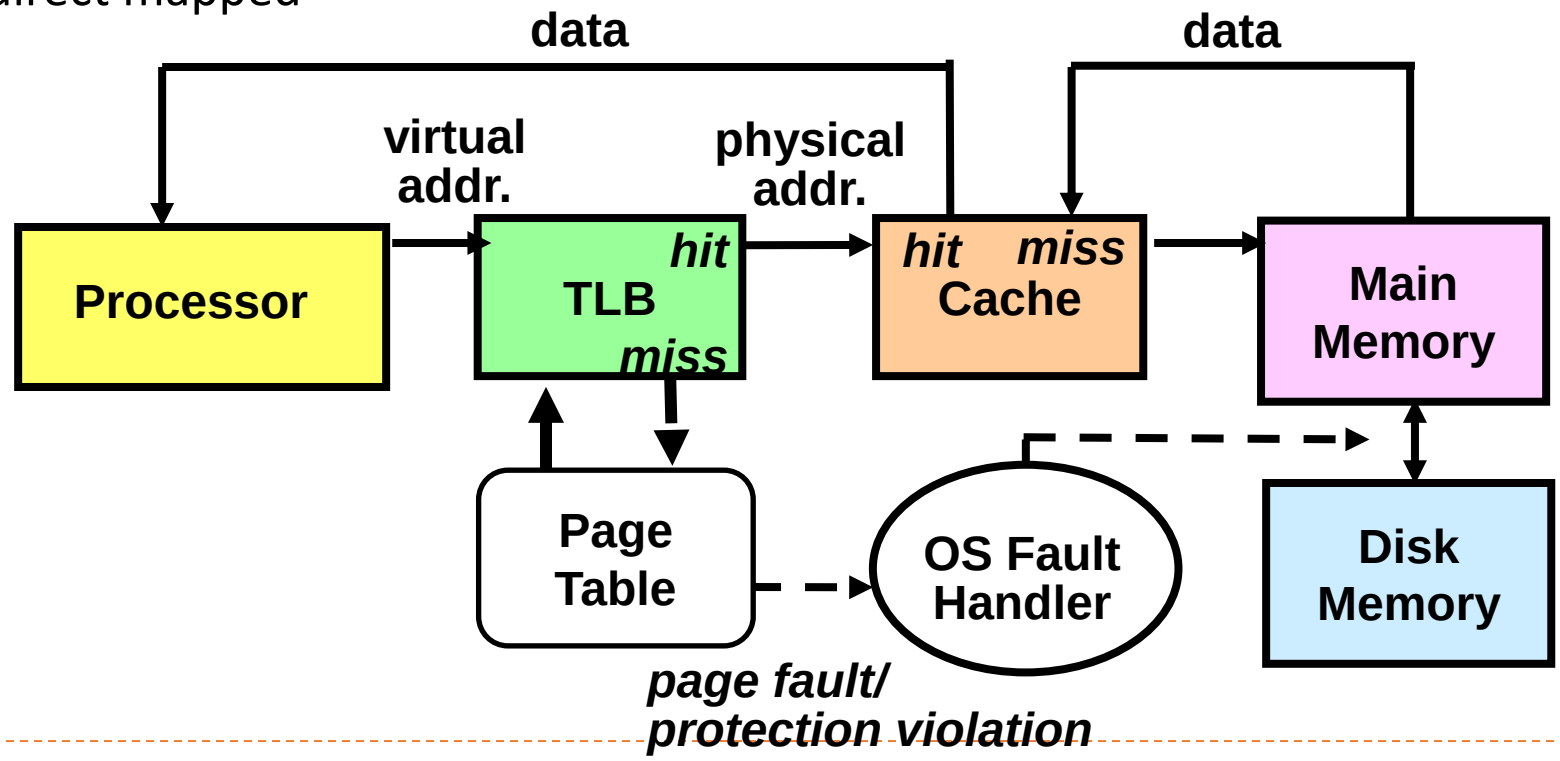
Typical TLB Format

Virtual Page Number	Physical Page Number	Valid	Ref	Dirty	Access Rights
“tag”	“data”				

- TLB just a cache of the page table mappings
- Dirty: since use write back, need to know whether or not to write page to disk when replaced
- Ref: Used to calculate LRU on replacement
  - Reference bit - set when page accessed; OS periodically sorts and moves the referenced pages to the top & resets all Ref bits
  - Must provide timer interrupt to update LRU bits

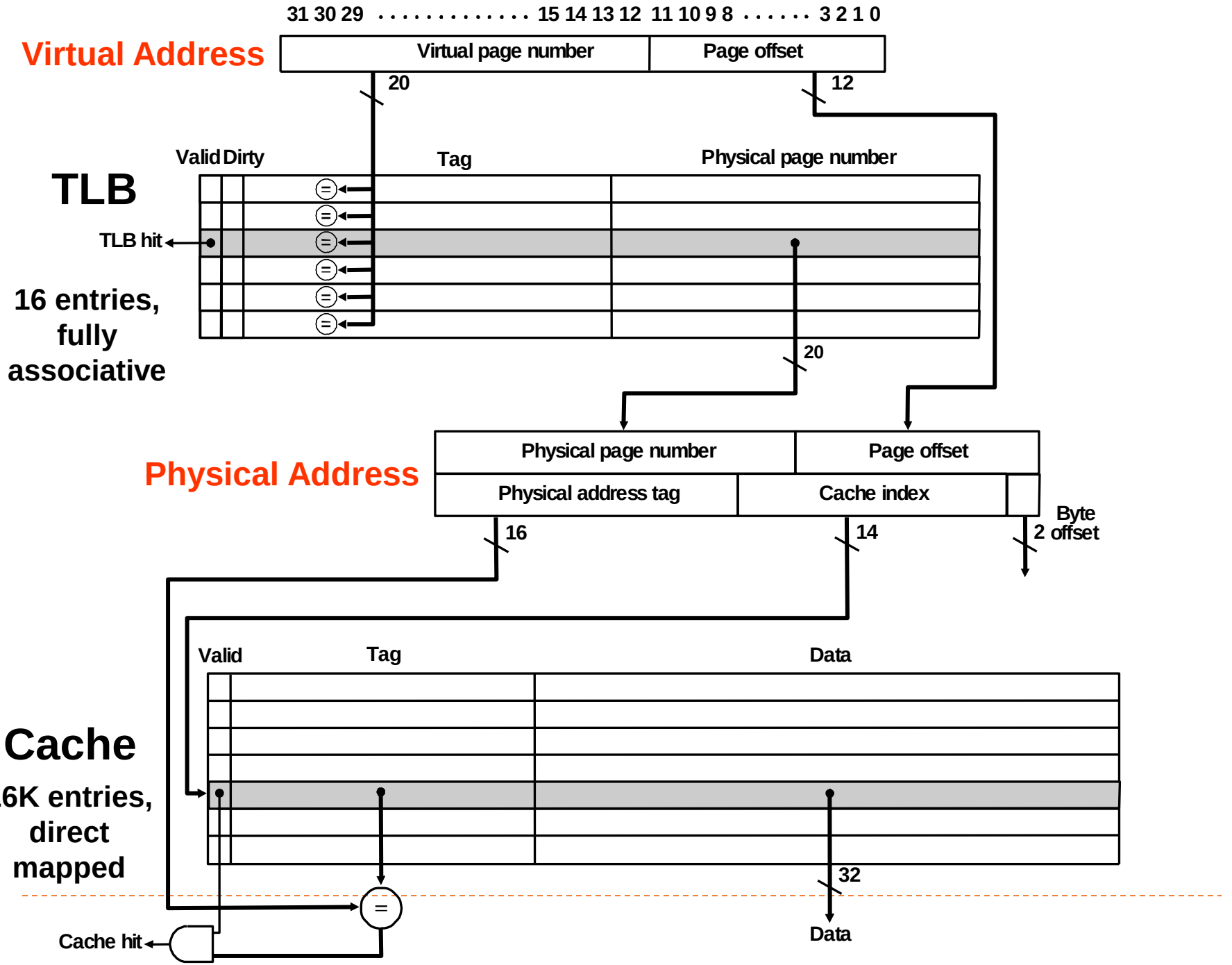
Translation Look-Aside Buffers

- TLB is usually small, typically 16-512 entries
- TLB access time comparable to cache, typically 0.5-1 clock cycle (much less than main memory access time)
- Like any other cache, the TLB can be fully associative, set associative, or direct mapped

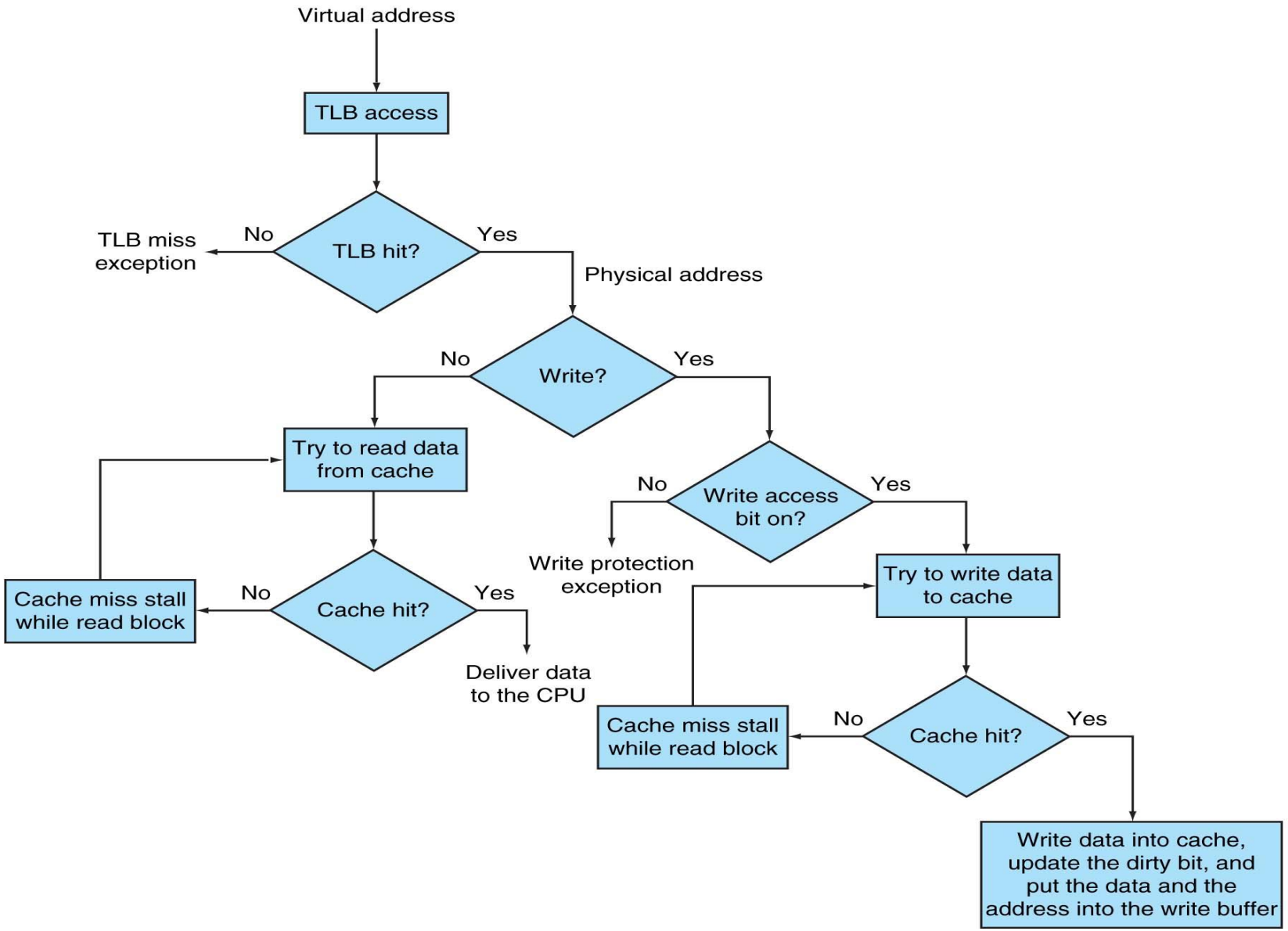


The Intrinsity FastMATH TLB

- Memory system: 4KB pages 32-bit address space
- The virtual page number is 20 bits
- The physical address = the virtual address
- Contain 16 entries
- Fully associative
- Each entry is 64 bits wide, 20-bit tag (virtual page number), 20-bit physical page number, a valid bit, a dirty bit and other bits.


31 30 29 . . . . . 15 14 13 12 11 10 9 8 . . . . . 3 2 1 0
Virtual Address
Virtual page number
Page offset
20
12
Valid Dirty
Tag
Physical page number
TLB
TLB hit
16 entries, fully associative
20
Physical Address
Physical page number
Page offset
Physical address tag
Cache index
Byte 2 offset
16
14
Valid
Tag
Data
Cache
16K entries, direct mapped
32
=
Cache hit
Data

Steps in Memory Access - Example



TLB Event Combinations

- TLB / Cache miss: page / block not in “cache”
- Page Table miss: page NOT in memory

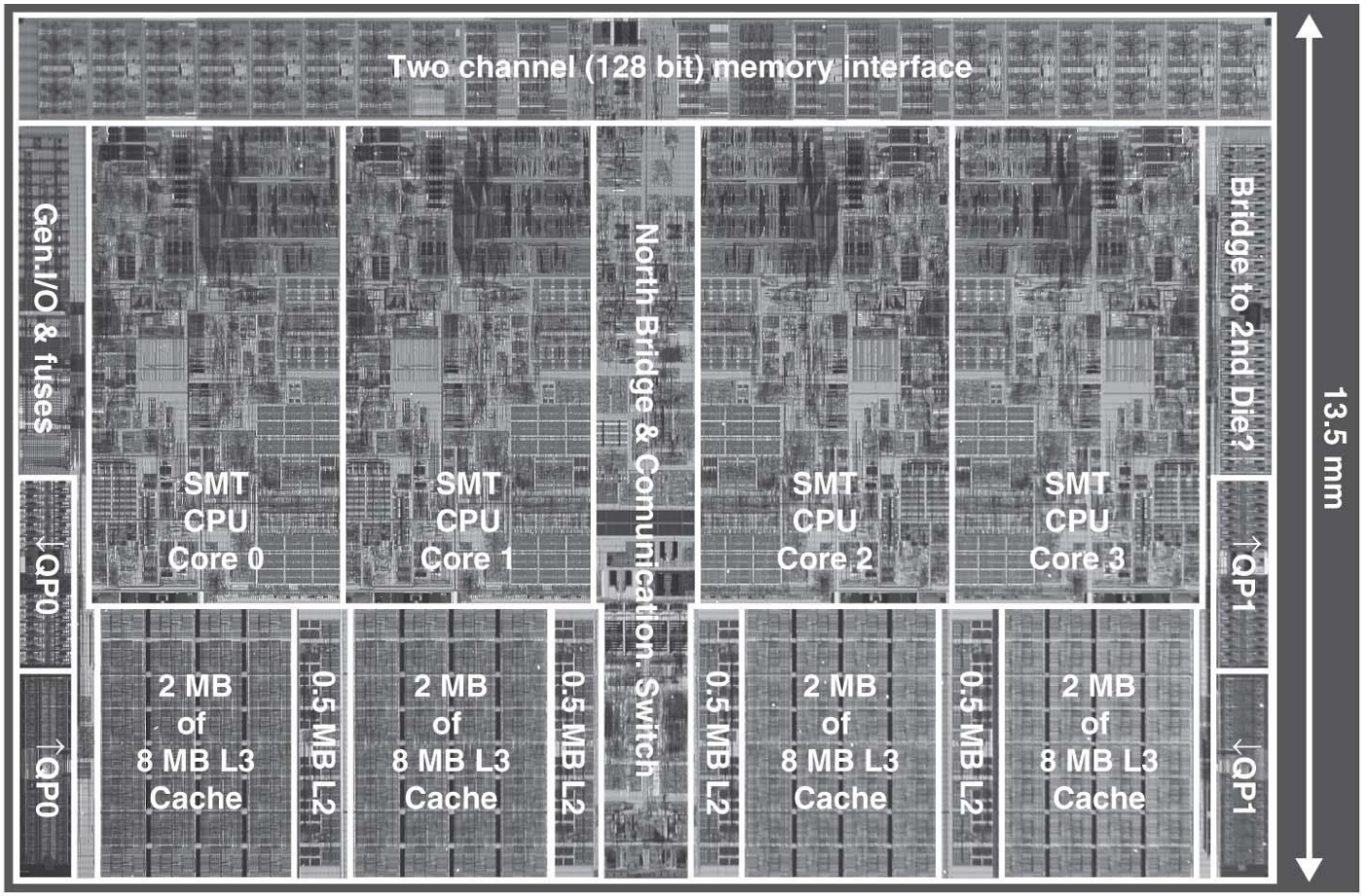
TLB	Page Table	Cache	Possible? Under what circumstances?
Hit	Hit	Hit	Yes – what we want!
Hit	Hit	Miss	Yes – although page table is not checked if TLB hits
Miss	Hit	Hit	Yes – TLB miss, PA in page table
Miss	Hit	Miss	Yes – TLB miss, PA in page table but data not in cache
Miss	Miss	Miss	Yes – page fault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

Real Stuff: Pentium Pro Memory Hierarchy

- Address Size: 32 bits (VA, PA)
- VM Page Size: 4 KB, 4 MB
- TLB organization: separate i,d TLBs
  (i-TLB: 32 entries,
  d-TLB: 64 entries)
  4-way set associative
  LRU approximated
  hardware handles miss
- L1 Cache: 8 KB, separate i,d
  4-way set associative
  LRU approximated
  32 byte block
  write back
- L2 Cache: 256 or 512 KB

Intel “Nehalim” quad-core processor

13.5×19.6 mm die;
731 million
transistors;
Two 128-bit
memory channels



Each processor has: private 32-KB instruction and 32-KB data caches and a 512-KB L2 cache. The four cores share an 8-MB L3 cache. Each core also has a two-level TLB.

Comparing Intel's Nehalim to AMD's Opteron

	Intel Nehalem	AMD Opteron X4
Virtual addr	48 bits	48 bits
Physical addr	44 bits	48 bits
Page size	4KB, 2/4MB	4KB, 2/4MB
L1 TLB (per core)	L1 I-TLB: 128 entries L1 D-TLB: 64 entries Both 4-way, LRU replacement	L1 I-TLB: 48 entries L1 D-TLB: 48 entries Both fully associative, LRU replacement
L2 TLB (per core)	Single L2 TLB: 512 entries 4-way, LRU replacement	L2 I-TLB: 512 entries L2 D-TLB: 512 entries Both 4-way, round-robin LRU
TLB misses	Handled in hardware	Handled in hardware

Further Comparison

	Intel Nehalem	AMD Opteron X4
L1 caches (per core)	L1 I-cache: 32KB, 64-byte blocks, 4-way, approx LRU, hit time n/a L1 D-cache: 32KB, 64-byte blocks, 8-way, approx LRU, write-back/allocate, hit time n/a	L1 I-cache: 32KB, 64-byte blocks, 2-way, LRU, hit time 3 cycles L1 D-cache: 32KB, 64-byte blocks, 2-way, LRU, write-back/allocate, hit time 9 cycles
L2 unified cache (per core)	256KB, 64-byte blocks, 8-way, approx LRU, write-back/allocate, hit time n/a	512KB, 64-byte blocks, 16-way, approx LRU, write-back/allocate, hit time n/a
L3 unified cache (shared)	8MB, 64-byte blocks, 16-way, write-back/allocate, hit time n/a	2MB, 64-byte blocks, 32-way, write-back/allocate, hit time 32 cycles

Reference Reading

- 5.7 5.13