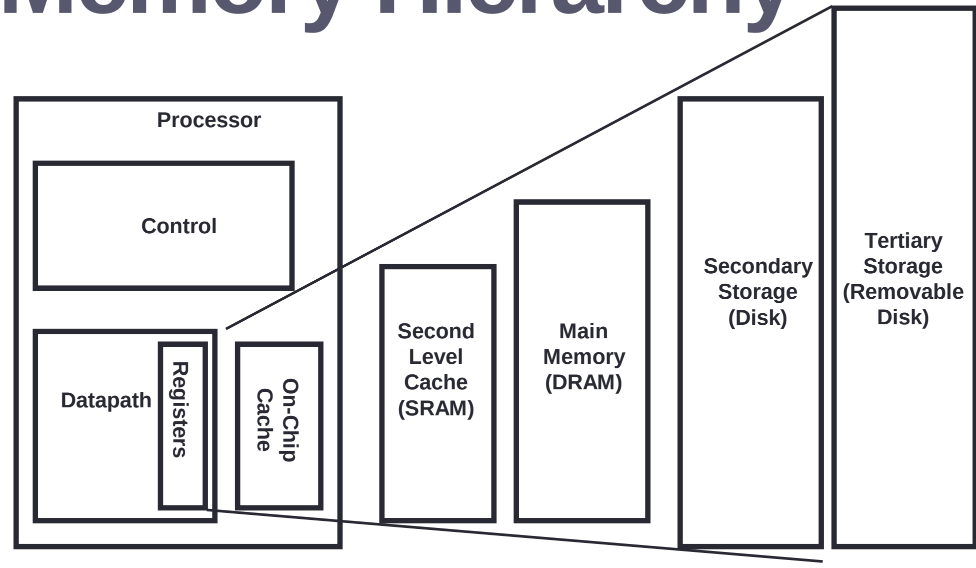
VIRTUAL MEMORY

Lecturer: Hui Annie Guo

h.guo@unsw.edu.au

K17-501F

Recall: Overview of Memory Hierarchy



- **registers ↔ memory**
  - **by compiler/programmer**
- **cache ↔ memory**
  - **by the hardware**
- **memory ↔ disks**
  - **by the hardware and operating system (virtual memory)**
  - **by the programmer**

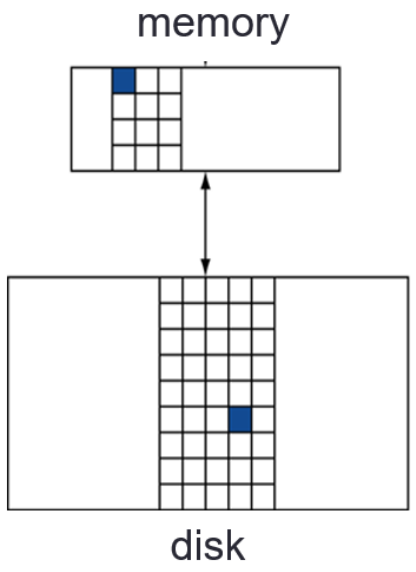
Lecture overview

- Topics
  - A glance of virtual memory
  - A hardware-centric view
    - Page table
    - TLB

- Suggested reading
  - H&P Chapter 5.7
  - https://en.wikipedia.org/wiki/Virtual_memory

Virtual memory

- A memory management technique
  - Use the main memory and disk to create an illusion of very large memory to the user.
- Use main memory as a “cache” for the secondary (disk) storage
  - Managed mainly by the operating system (OS)



Virtual memory (cont.)

- Programs have separate virtual memory space
- A process is a program that is being executed
- Processes share the main memory
  - Each gets a private address space, holding its code and data
  - They are protected from each other by OS

Virtual memory (cont.)

- Page
  - A virtual block
- Page hit
  - Data accessed are in the main memory
- Page fault
  - Data accessed are not in the main memory

Virtual memory (cont.)

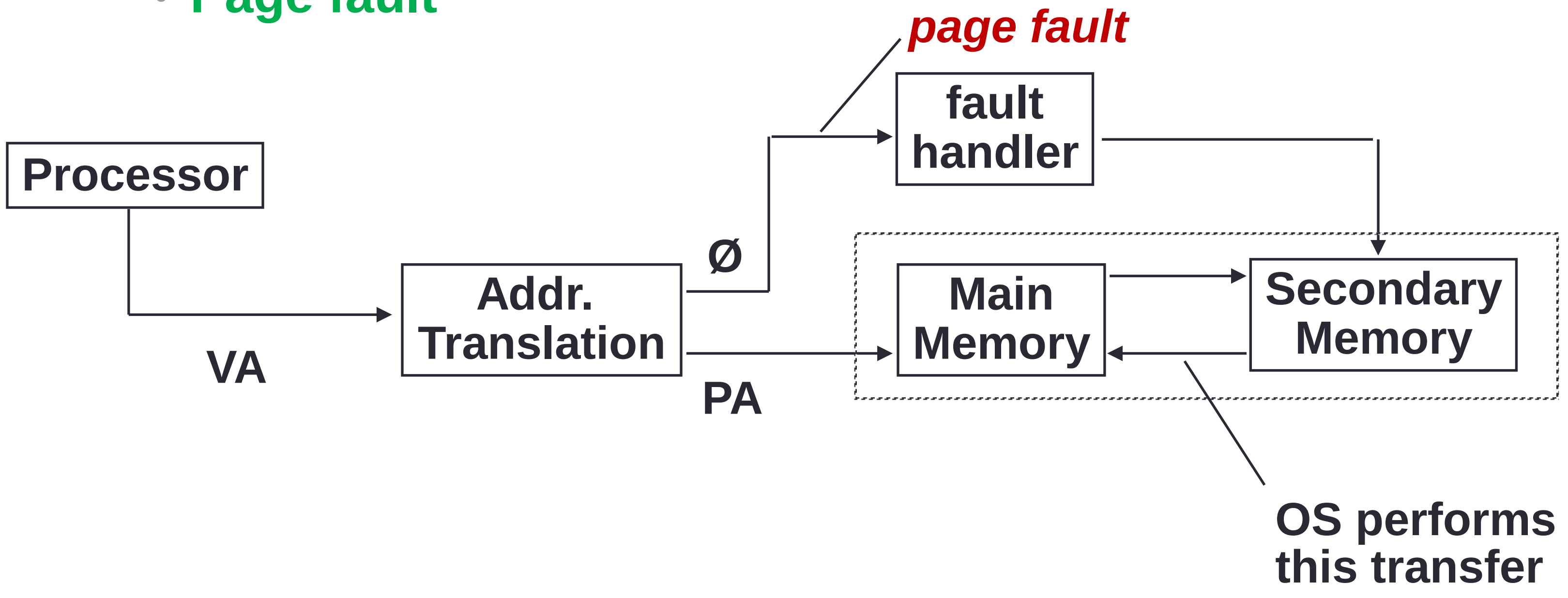
- The basic tasks of using virtual memory
  - Translation between virtual address (VA) and physical address (PA)
  - Access control of physical memory space
- The control of using virtual memory is often carried out in a component, called memory management unit (MMU)
  - VM translation “miss” is effectively a page fault

Four design issues

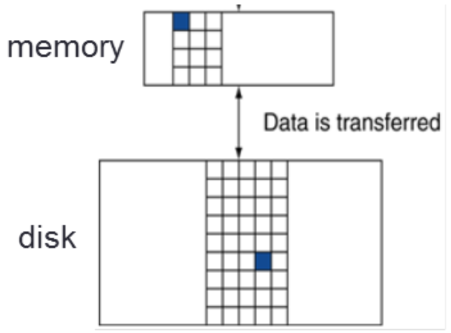
- **Where to place a page?**
  - **Fully associative or highly associative**
- **How to find a page?**
  - **Address translation**
- **Which page should be replaced on a page fault?**
  - **Sophisticated (LRU + "working set")**
- **What happens when you want to write a page back?**
  - **Always write-back and write allocate**
    - **Disk writes take millions of clock cycles**

Memory access with virtual address

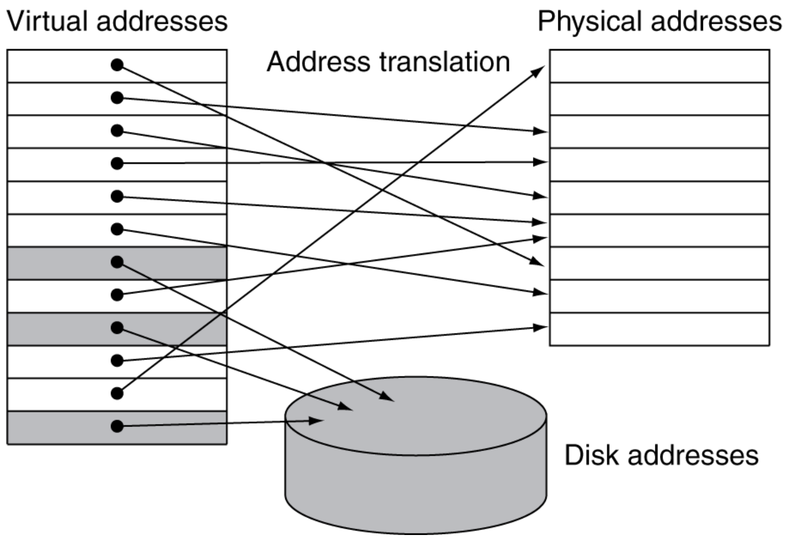
- Given a virtual address, the address translation generates two possible results
  - PA
    - Page hit
  - no PA
    - Page fault

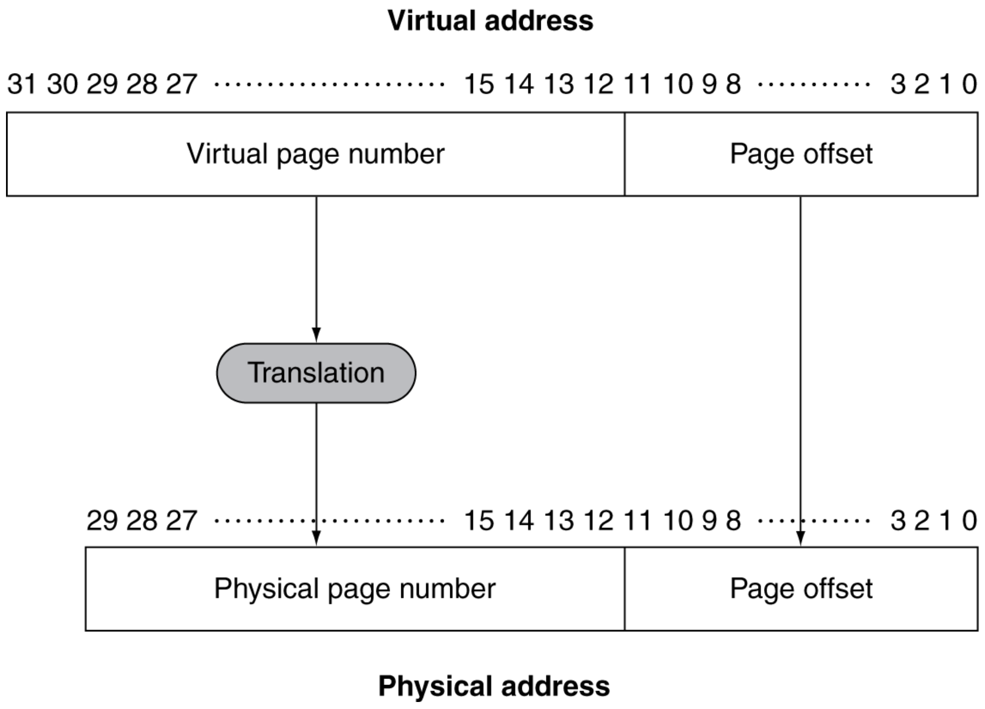


Address translation



- Virtual address
  - Virtual page number + Page offset
- Physical address
  - Physical page number + Page offset
- Translation is based on page table

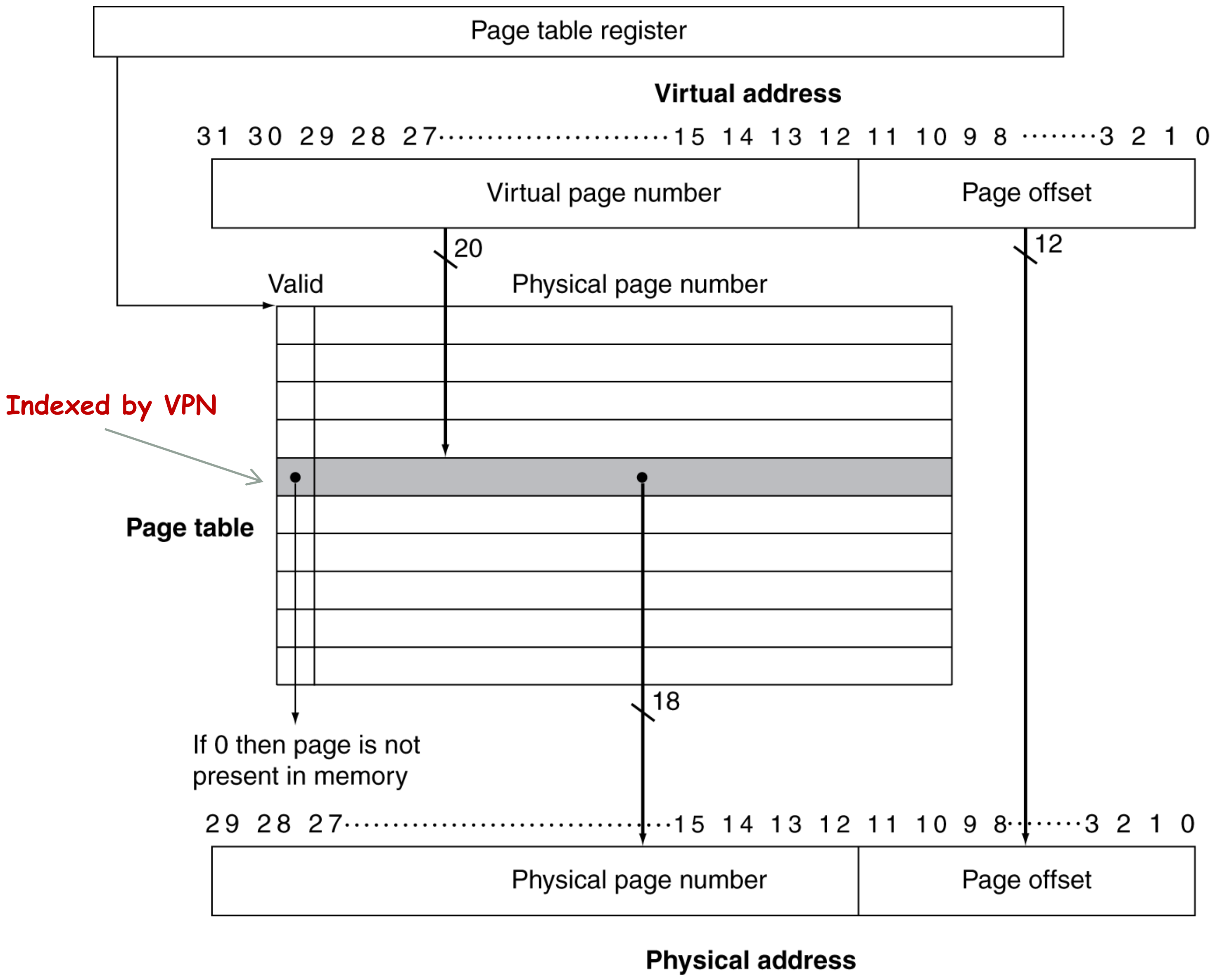




Page table (PT)

- Usually is implemented in the physical memory.
- Stores page placement information
  - An array of page table entries (PTE), indexed by virtual page number (VPN)
  - Page table register (PTR) points to the page table location in the memory
- If a page is present in memory
  - PTE stores the physical page number
  - Plus other status bits (referenced, dirty, …)
- If a page is not present
  - PTE refers to the location in the swap space on disk

Translation using page table



In-class exercise (1)

- **Assume a program is executed in a virtual memory space with the page size of 8KB without page fault. The PTR for the execution is set to 0x00110. The figure to the right shows the contents of a set of main memory locations.**
  - **(a) What is the Page offset field size in the virtual address?**

main memory

ADDR	CONTENTS
0x00000	0x0000
0x00100	0x0010
0x00110	0x0022
0x00120	0x0045
0x00130	0x0078
0x00145	0x0010
0x10000	0x2333
0x10020	0x4444
0x22000	0x1111
0x22020	0x2222
0x45000	0x5555
0x45020	0x6676

In-class exercise (1)

- **Assume a program is executed in a virtual memory space with the page size of 8KB without page fault. The PTR for the execution is set to 0x00110. Figure to the right shows the contents of a set of main memory locations.**
  - **(b) Does a PA exist for the following virtual address?**

    0x010020

main memory

ADDR	CONTENTS
0x00000	0x0000
0x00100	0x0010
0x00110	0x0022
0x00120	0x0045
0x00130	0x0078
0x00145	0x0010
0x10000	0x2333
0x10020	0x4444
0x22000	0x1111
0x22020	0x2222
0x45000	0x5555
0x45020	0x6676

In-class exercise (1)

- **Assume a program is executed in a virtual memory space with the page size of 8KB without page fault. The PTR for the execution is set to 0x00110. Figure to the right shows the contents of a set of main memory locations.**
  - **(c) What value will be returned to processor if it requests to read the following address?**

    0x010020

main memory

ADDR	CONTENTS
0x00000	0x0000
0x00100	0x0010
0x00110	0x0022
0x00120	0x0045
0x00130	0x0078
0x00145	0x0010
0x10000	0x2333
0x10020	0x4444
0x22000	0x1111
0x22020	0x2222
0x45000	0x5555
0x45020	0x6676

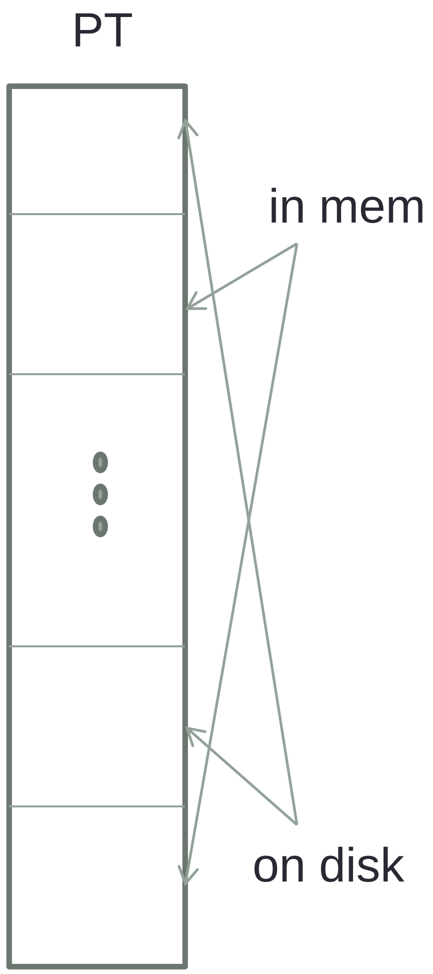
In-class exercise (2)

- **For the virtual memory space, physical memory size and the page size given in exercise (1), what is the size of a page table?**

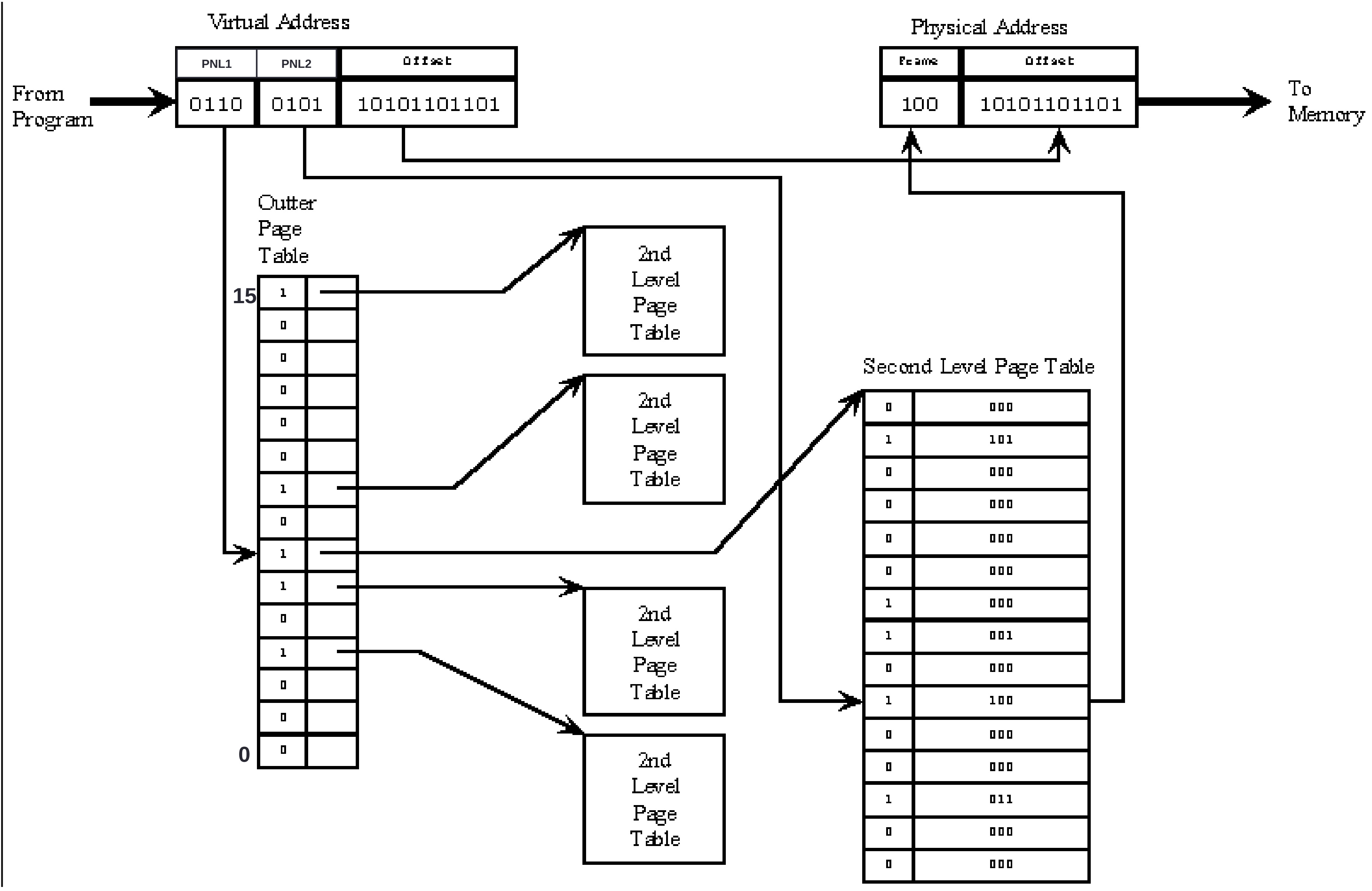
- **What if the virtual address is increased to 32 bits?**

Multiple level page table*

- **Only store a fraction of the table in the memory**
  - **Dividing the page table into a set of sub-tables**
  - **Some sub-tables stored in the memory, and**
  - **Other sub-tables paged out to the disk**



Translation with multi-level page table*



Fast translation using TLB

- **Address translation incurs extra memory references**
  - **access PT**
  - **access the actual memory**
- **But access to page table has good locality**
  - **Leading to theTLB design**

TLB: Translation look-aside buffer

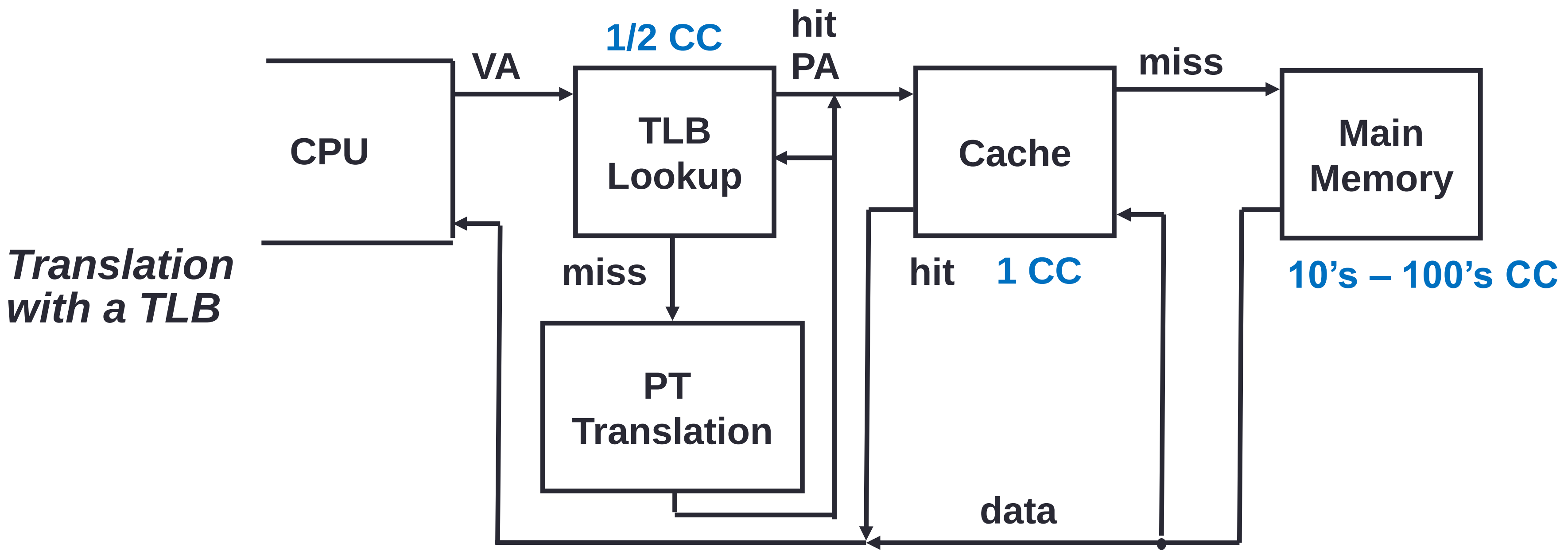
- **Is a special cache for page table**
- **Implemented on the processor chip**
- **Offers a fast VA to PA translation**
- **Just like any other cache, TLB can be organized as**
  - **fully associative,**
  - **set associative, or**
  - **direct mapped**

TLB: Translation look-aside buffer (cont.)

- **TLBs are usually small, typically not more than 128~256 entries**
  - **This permits fully associative lookup on high-end machines.**
  - **Most mid-range machines use small n-way set associative organizations.**

CPU datapath with TLB

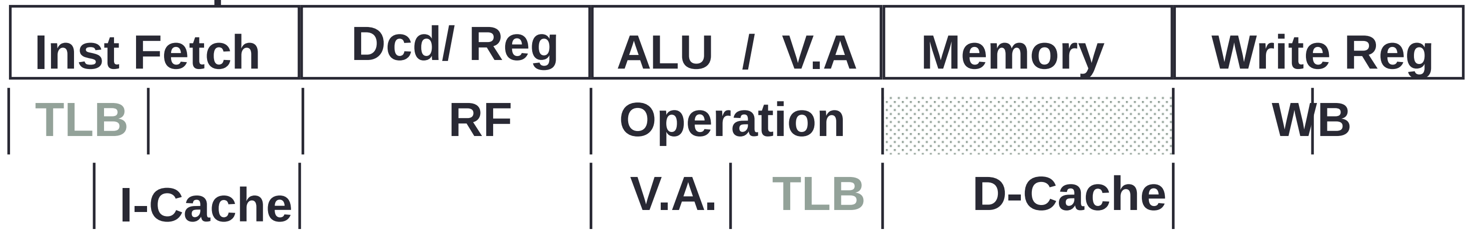
- **The delay of TLB lookup is less than one clock cycle**
  - **E.g. 1/2 clock cycles, as compared to many clock cycles of the page-table based translation**



Further improvement

- Reduce the impact of address translation on performance
  - Use the slack time in a pipeline stage execution
  - Overlap TLB access with another processor operation

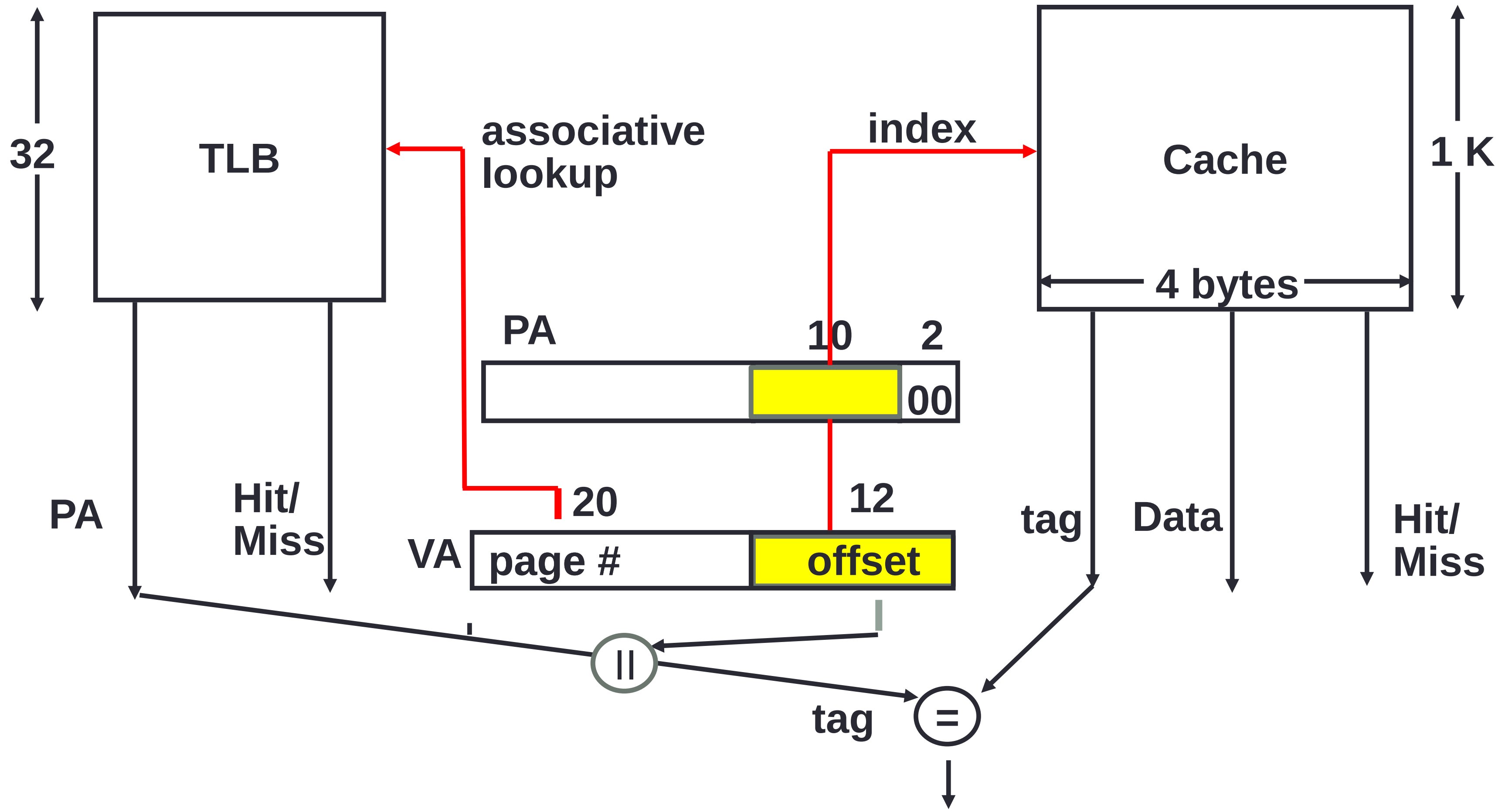
MIPS R3000 Pipeline



Overlap I-cache with TLB access

- **High order bits of VA are used to look up in TLB**
- **Low order bits of VA are used as index to search in cache**
- **Both can be performed in parallel**
- **See next slide for demonstration**

Overlap I-cache with TLB access (cont.)



IF cache hit AND (cache tag = PA) THEN deliver data to CPU
ELSE IF cache miss AND TLB hit THEN
    access memory with the PA from the TLB
ELSE do standard VA translation

Summary of memory hierarchy

- **Common principles apply at all levels of the memory hierarchy**
  - **Based on the notion of caching**
- **At each level in the hierarchy, four issues should be addressed**
  - **Block placement**
  - **Block identification**
  - **Replacement on a miss**
  - **Write policy**