1. Memory hierarchy general
2. Basics of cache
3. Improving cache performance
4. **Virtual memory**
   - Virtual memory general
   - Page table
   - TLB

Introduction

- **In this section we will discuss one more level of memory hierarchy called virtual memory.**
- **Virtual memory uses two memory levels – main memory and secondary memory.**
- **We can achieve something more than performance with virtual memory.**

Recall : Memory Hierarchy

Capacity Access Time Cost	Level	Staging Xfer Unit
CPU Registers **100s Bytes** **<10s ns**	**Registers**	
	↕ Instr. Operands	**prog./compiler** **1-8 bytes**
Cache **K Bytes** **10-100 ns** **$.01-.001/bit**	**Cache**	
	↕ Blocks	**cache cntl** **8-128 bytes**
Main Memory **M Bytes** **100ns-1us** **$.01-.001**	**Memory**	
	↕ Pages	**OS** **512-4K bytes**
Disk **G Bytes** **ms** **10^{-3} - 10^{-4} cents**	**Disk**	
	↕ Files	**user/operator** **Mbytes**
Tape **infinite** **sec-min** **10^{-6}**	**Tape**	

Upper Level

faster ↑

↓ Larger

Lower Level

Memory Hierarchy Requirements

- **If Principle of Locality allows caches to offer (close to) speed of cache memory with size of DRAM memory,**
  then recursively why not use at next level to give speed of DRAM memory, size of Disk memory?
- **Share memory between multiple processes but still provide protection – don't let one program read/write memory from another**
- **Address space – give each program the illusion that it has its own private memory**
  - **compiler, linker, and loader are simplified because they see only the virtual address space abstracted from physical memory allocation.**

Virtual Memory

- Called "Virtual Memory"
- Uses 2 storage levels
  - primary (DRAM) and secondary (Hard Disk)
- Let OS to share memory, protect programs from each other
- Today, more important for protection vs. just another level of memory hierarchy
- Each process thinks it has its own memory space to itself.
- Historically, it predates caches
- Distinguish between virtual and physical addresses
  - virtual address is used by the programmer to address memory within a process's address space
  - physical address is used by the hardware to access a physical memory location

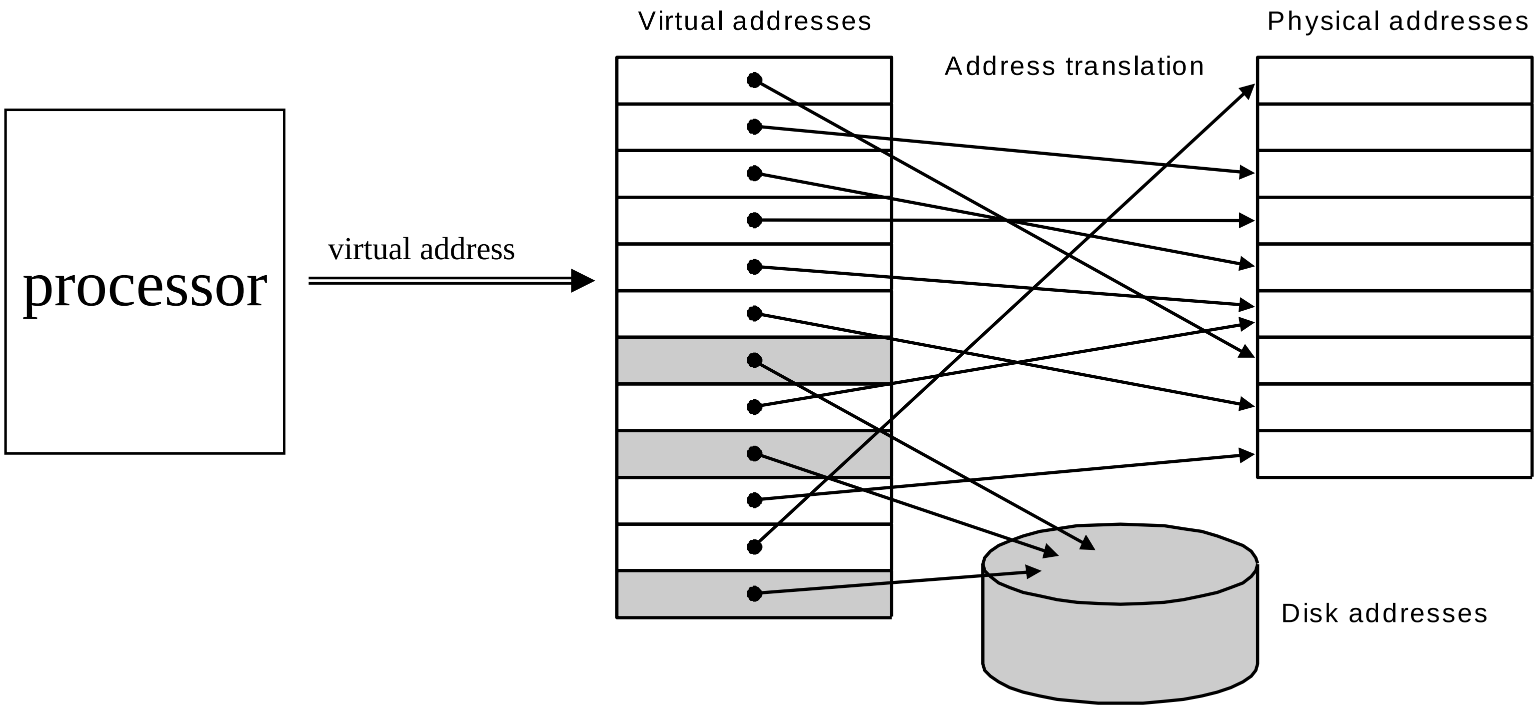
Virtual Memory

- illusion of having more physical memory

  Addressable Memory Space vs. Physical Memory
  - example:
    - 32bit memory address can specify 4GB memory
    - physical main memory = 128MB ~ 1GB
- "Virtual Memory" provides appearance of very large memory
  - total memory of all jobs >> physical memory
  - address space of each job > physical memory
- Simplifies memory management for multi-processing system
  - each program operates in its own virtual address space as if it is the only program running in the system

Virtual Memory

- **Main memory can act as a cache for the secondary storage (disk)**
- **Only a portion of memory is actively being used → main memory need to contain only the active portion of the many programs just as a cache does.**

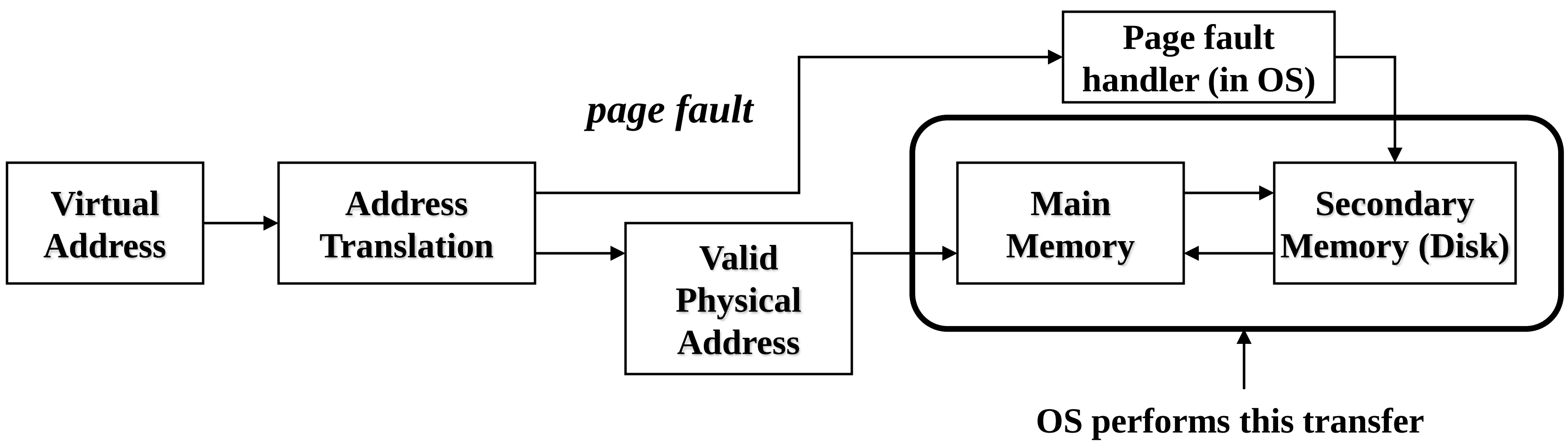


Address Translation

- **Program uses virtual addresses**
  - **Relocation: a program can be loaded anywhere in physical memory without recompiling or re-linking**
- **Hardware (HW) provides virtual $\Rightarrow$ physical mapping**
  - **need a translation table for each process**

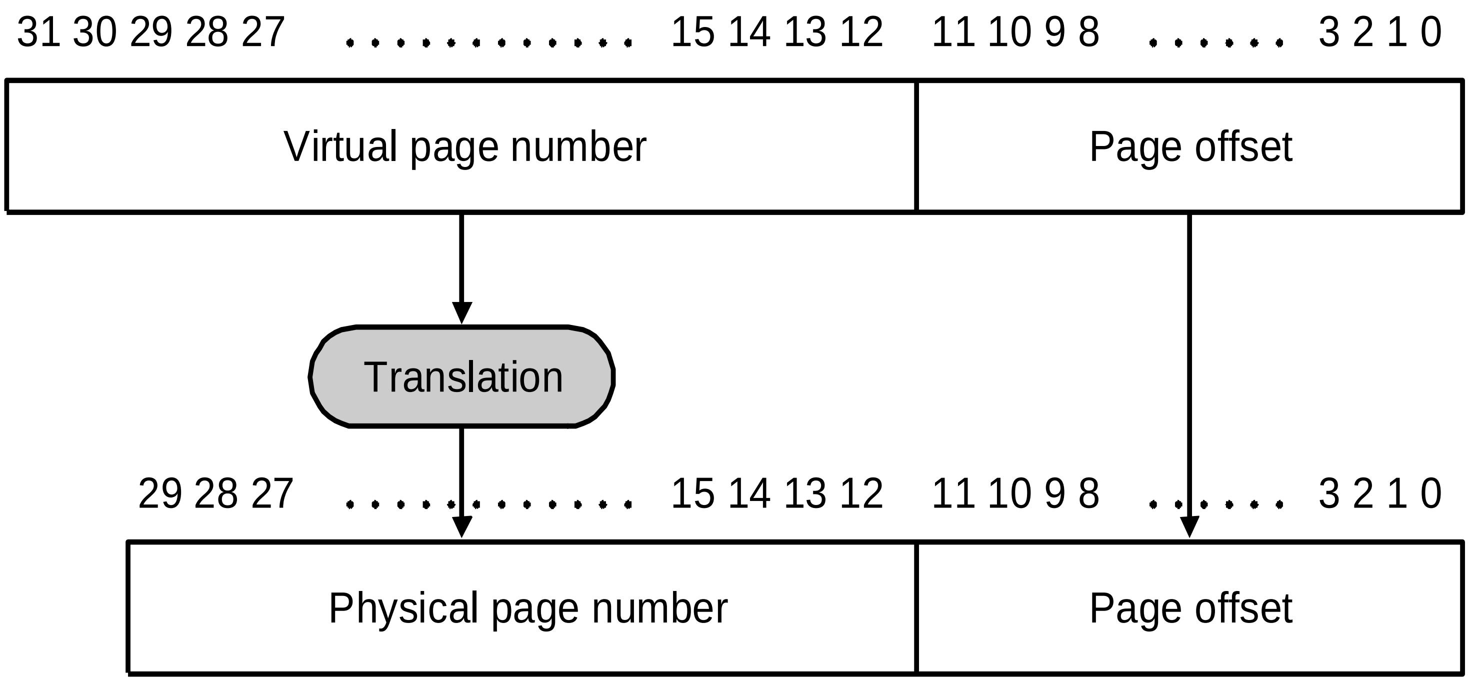
Address Translation

- **When a virtual address is missing from main memory, the OS handles the miss**
  - **read the missing data, create the translation, return to re-execute the instruction that caused the miss**



Pages: Virtual Memory Blocks (Fixed Block size Scheme)

- **Page :** **A virtual memory Block**
- **Mapping from a virtual to physical address**
  - **virtual address = virtual page number + page offset**
  - **physical address = physical page number + page offset**

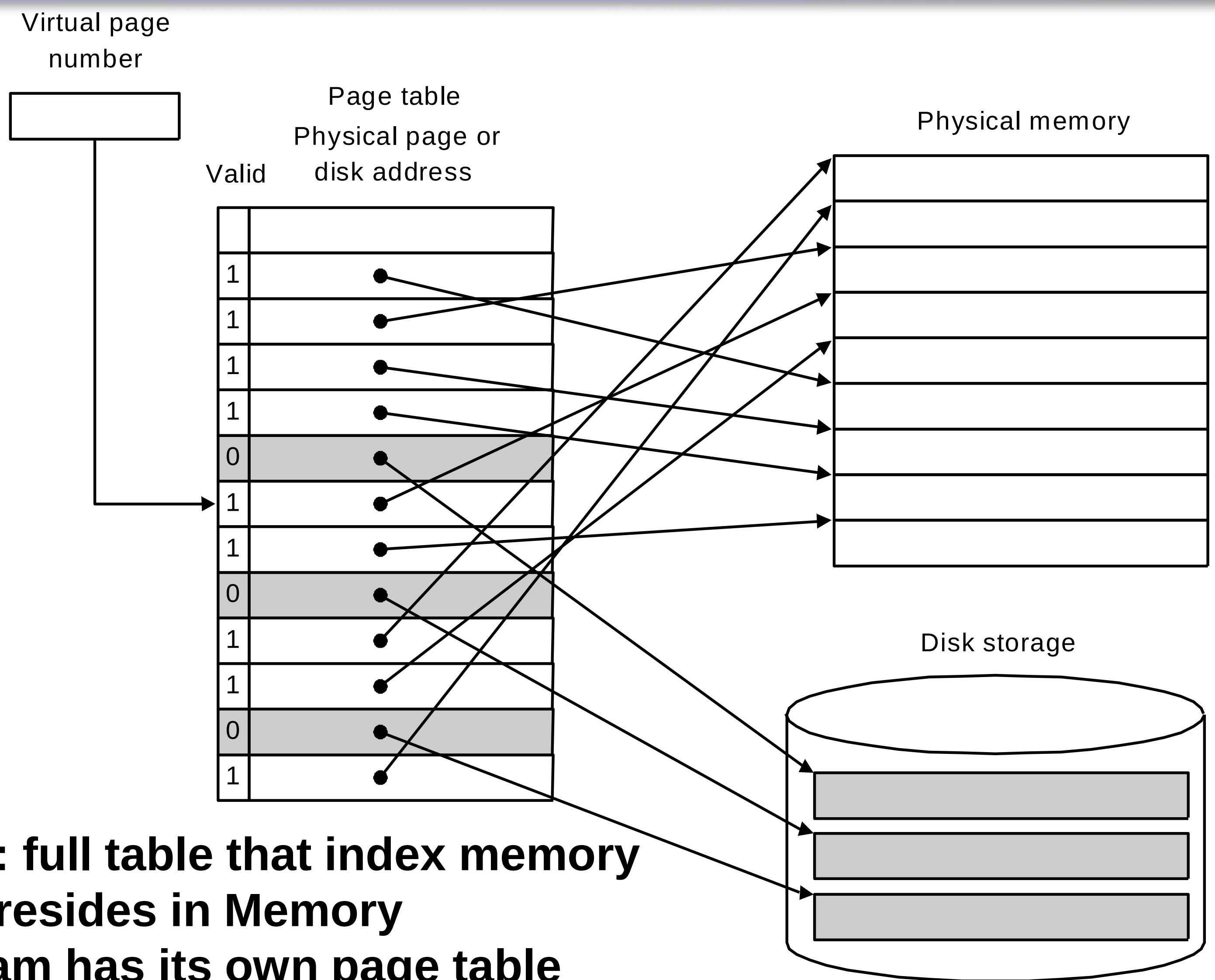


Physical address

Page Faults

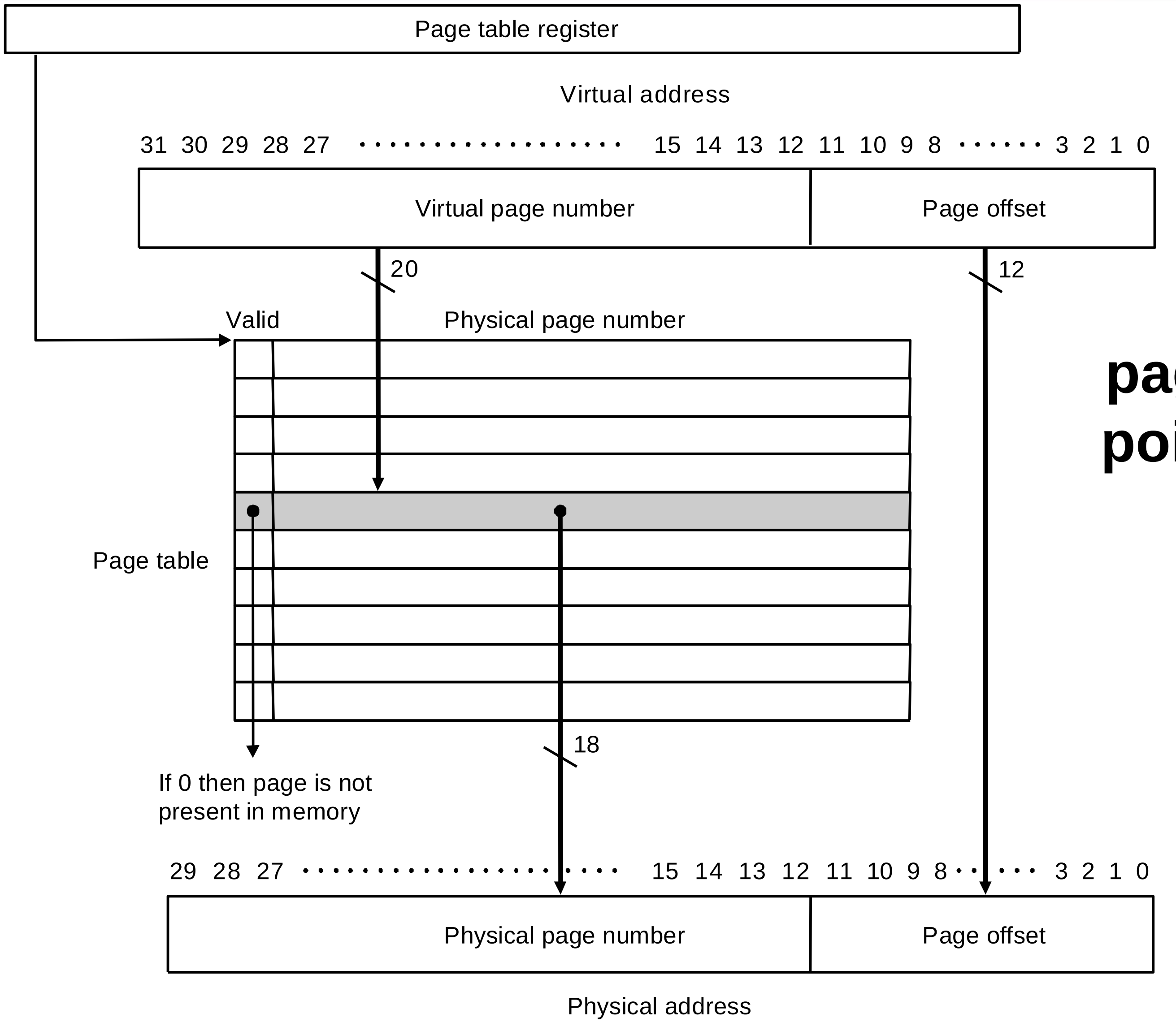
- **Page fault: a Virtual memory miss**
  - **The data is not in memory, retrieve it from disk**
  - **Reducing page faults is important (LRU is worth the price)**
  - **Can handle the faults in software instead of hardware**
  - **Using write-through scheme is too expensive, so we use writeback**

Page Tables



- **Page Table : full table that index memory**
- **Page Table resides in Memory**
- **Each program has its own page table**

Page Tables

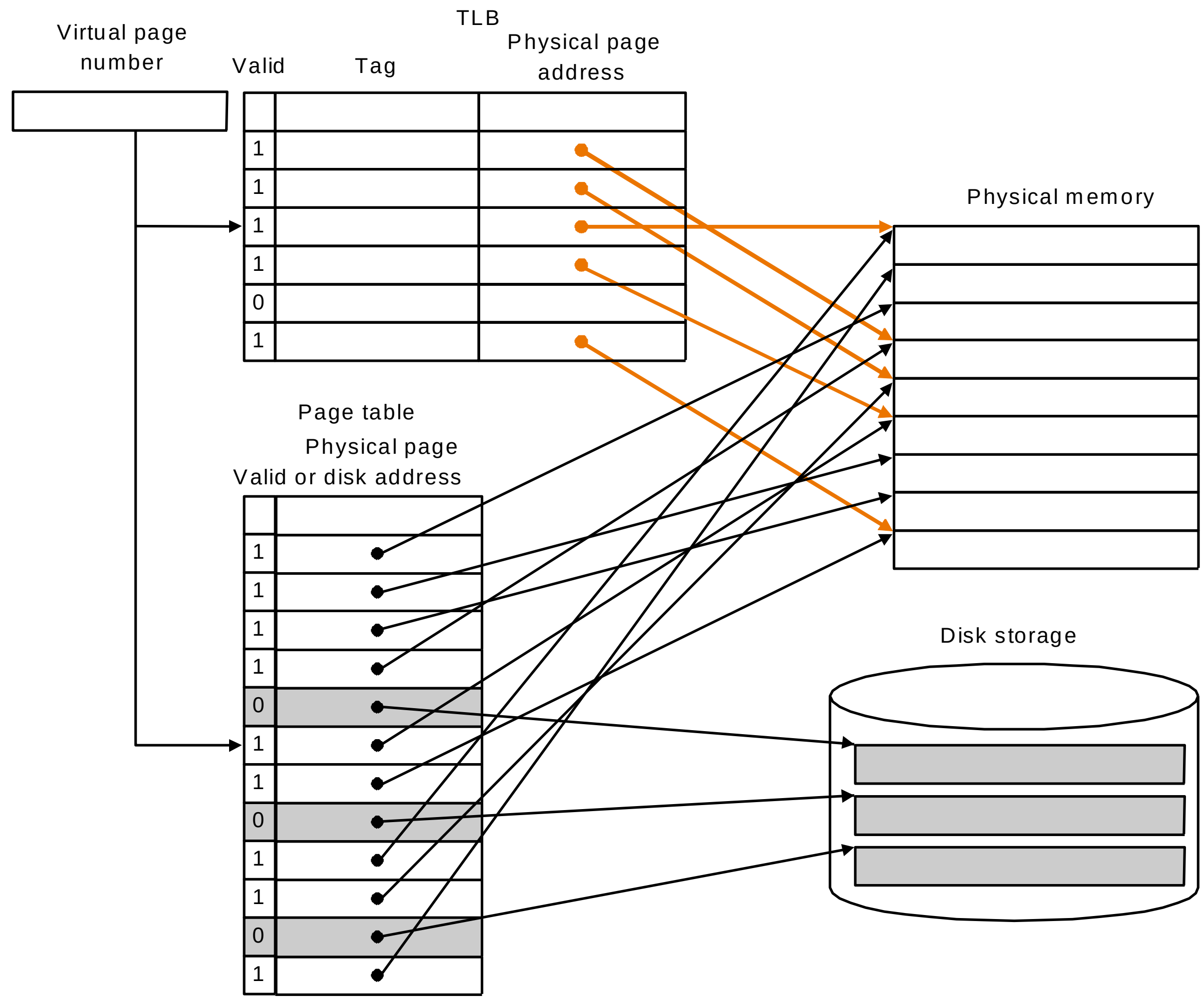


page table register : points to the start of page table

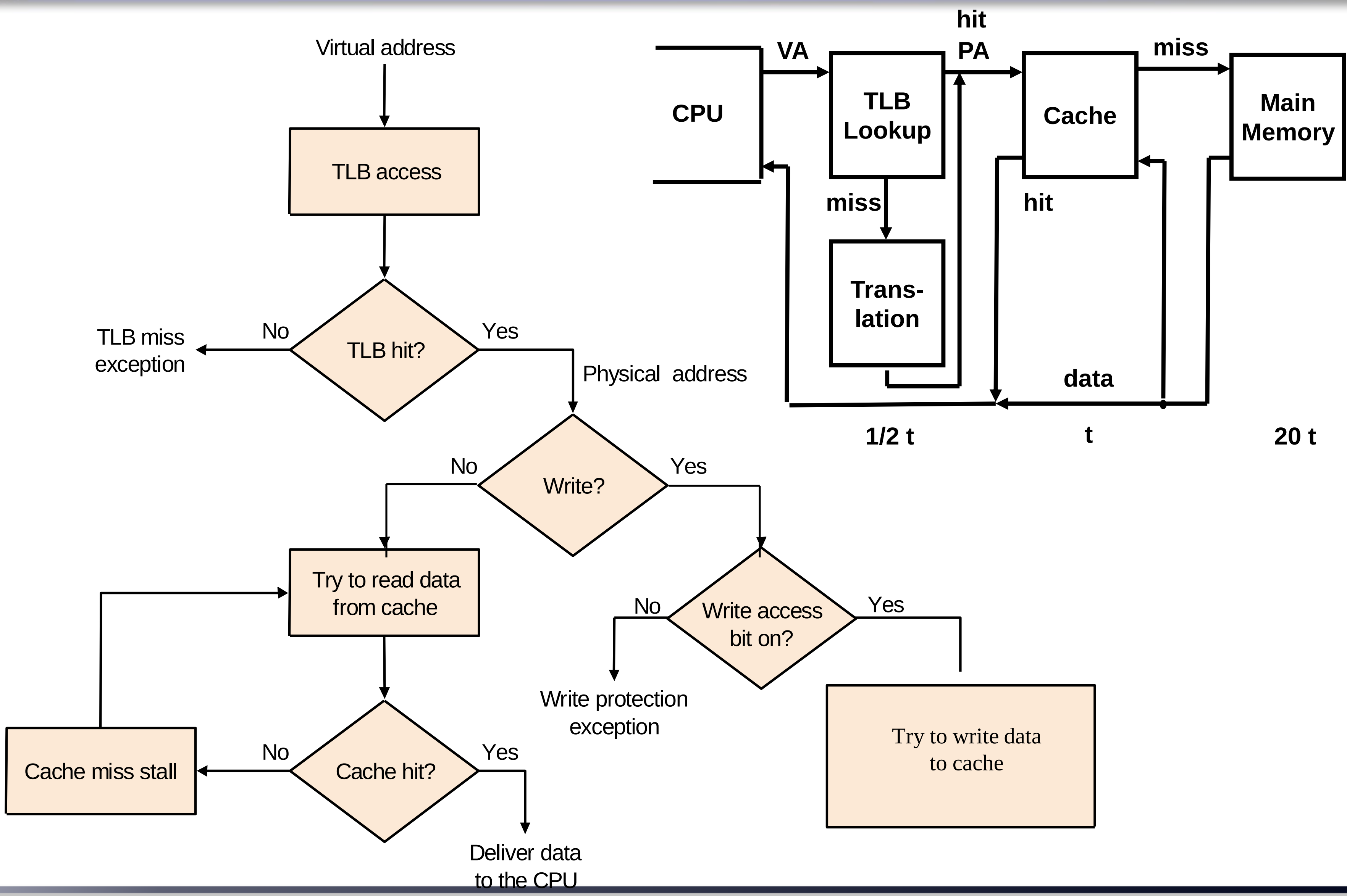
Making Address Translation Fast

- **TLB(Translation Lookaside Buffer) :**
  a Special Cache for Address Translations

  - **Key to improve the performance :** locality of reference to page table



TLBs and caches



Some Issues

- **Processor speeds continue to increase very fast much faster than either DRAM or disk access times**
- **Design challenge: dealing with this growing disparity**
- **Trends:**
  - **Synchronous DRAMs (provide a burst of data)**
  - **redesign DRAM chips to provide higher bandwidth or processing**
  - **restructure code to increase locality**
  - **use prefetching : A block of data is brought into the cache before it is actually referenced. Compiler tries to identify data blocks needed in the future.**

Summary

- Virtual Memory is a caching between main memory and disk, allowing a program to expand its address space beyond the limit of main memory.
- Page tables map virtual address to physical address
- TLBs are important for fast translation
- Techniques to reduce miss rate (← The high cost of page fault)
  - Pages are large to take advantage of spatial locality
  - TLB is a fully associative
  - Operating system uses techniques such as LRU and reference bit to choose pages to be replaced
- Virtual Memory use Write back and track if a page is changed (with dirty bit)
- Today VM allows many processes to share single memory; VM protection is more important than memory hierarchy