ELE 455/555
Computer System Engineering

Section 3 – Memory

Class 4 – Virtual Memory

Virtual Memory

Virtual Machines

- Virtual Machine
  - Host computer emulates guest operating system and machine resources
    - Improved isolation of multiple guests
    - Avoids security and reliability problems
    - Aids sharing of resources
  - Virtualization has a performance impact
    - Feasible with modern high-performance computers
  - Examples
    - IBM VM/370 (1970s technology!)
    - VMWare
    - Microsoft Virtual PC
    - Cloud computing

Virtual Memory

Virtual Machines

- System Virtual Machine
  - Virtualizes the entire computing experience
    - Users believe they are running stand-alone
  - Can support multiple OSes
  - Share hardware resources
  - Supported by the virtual machine monitor (VMM)
    - aka hypervisor
  - Hardware is called the host
  - VMs are called the guests

Virtual Memory

Virtual Machines

- Amazon Web Services (AWS)
  - Uses virtual machines in its web services to:
    - Protect users from each other
    - One guest image can be distributed to as many hosts as needed
    - Can reliably “kill” a VM when done (security)
    - Hides hardware – giving AWS flexibility in age, location, speed, …
    - Resource control and monitoring – can limit the resources available to each guest

Virtual Memory

Overview

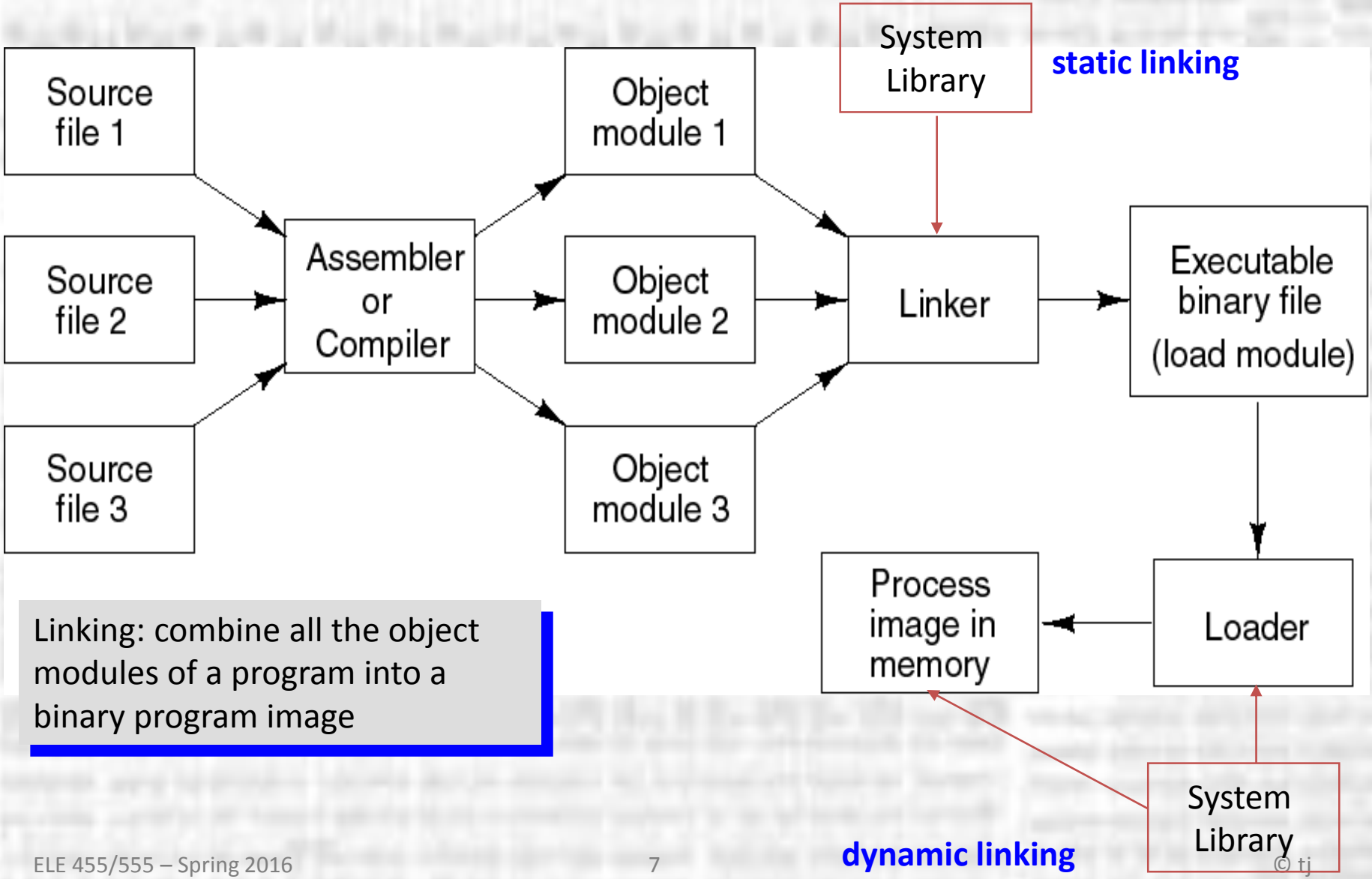
- Virtual Memory
  - Use main memory as a cache for secondary memory
    - Typically HDD for PC systems, Flash for mobile systems
  - Why?
    - Allow multiple programs (VMs) to share a common memory
    - Manage the limitations of main memory size
  - How?
    - Cache portions of the secondary memory in main memory
    - Allows different programs to be resident in main memory with-in different cache blocks (pages)

Virtual Memory

Overview

- Virtual Memory
  - If we cache different programs into main memory we create an addressing issue – processor uses the absolute address in main memory
    - How does the processor know where specific parts of code get cached to?
    - The location can change each time the program is started
    - The location can change if it gets swapped out and then back in again
  - Each program gets its own address space
    - This is a fictitious (virtual) address space
    - Only the program can access its address space
    - The address space is defined at compile time and does not change
  - The processor must translate(through HW and SW) from this "virtual address" to a physical address.

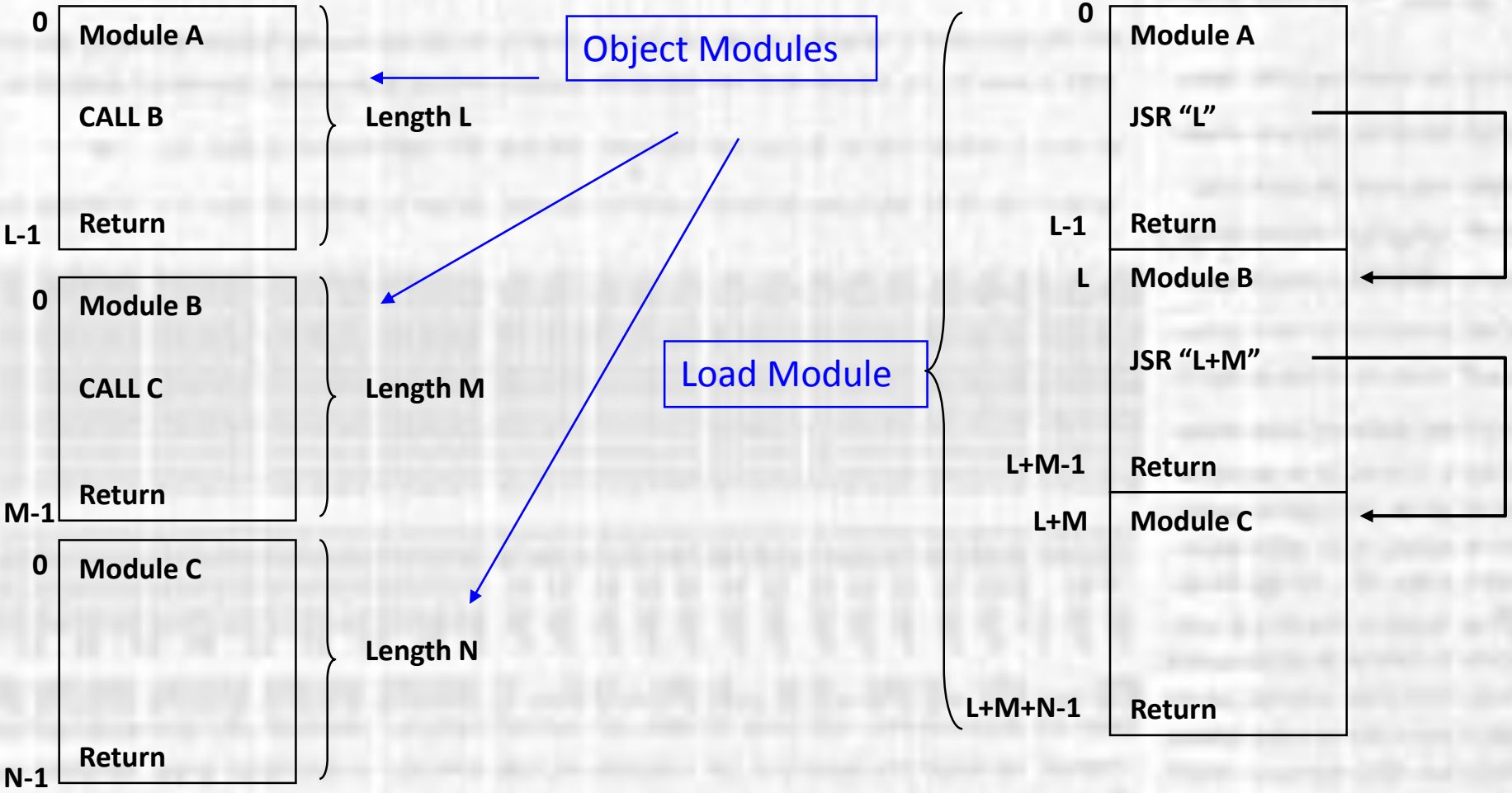
Steps for Loading a Process in Memory



-
-

Linking: combine all the object modules of a program into a binary program image

The Linking Function



Virtual Memory

Overview

- Terminology
  - Virtual Memory operates like a cache but some of the terms used are different for historical reasons
    - Address generated by the processor – virtual address
    - Blocks used in the VM system are called Pages
    - Misses in the VM system are called Page Faults
    - Physical Memory – typically refers to main memory (DRAM)
    - Address Translation – converting the virtual memory address created by the processor to a physical memory address associated with the DRAM, flash or disk
    - Swap Space – a copy of all the virtual memory space (required by a process) on the HDD – makes finding unloaded pages easier (16GB)

Virtual Memory

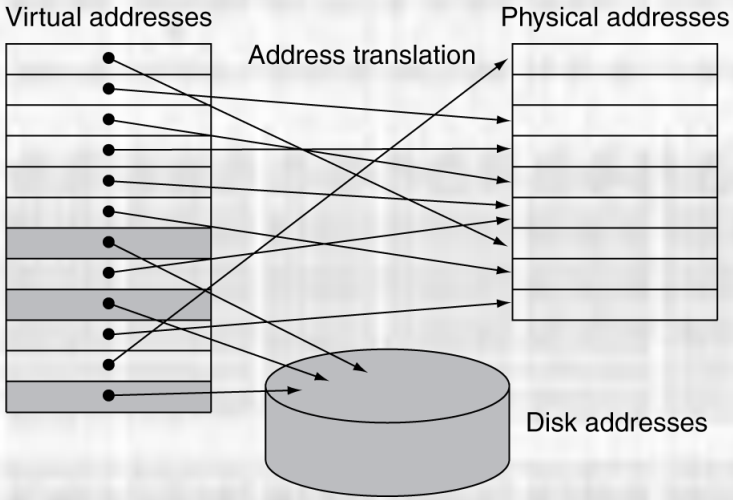
Overview

- Address Translation
  - Each program has its own virtual memory space
  - When loading the program the processor will map the virtual memory location loaded (used by the program) - into the corresponding physical address actually used to store the code/data
    - Not all of the program need be loaded – only the pages needed
  - Pages are fixed in size (remember these are cache blocks)
  - Pages can be loaded into any main memory location the processor chooses

Virtual Memory
Overview

- Address Translation

Virtual addresses – created by the compiler and used by the processor



Physical addresses – The actual location of blocks (pages) in memory or on disk

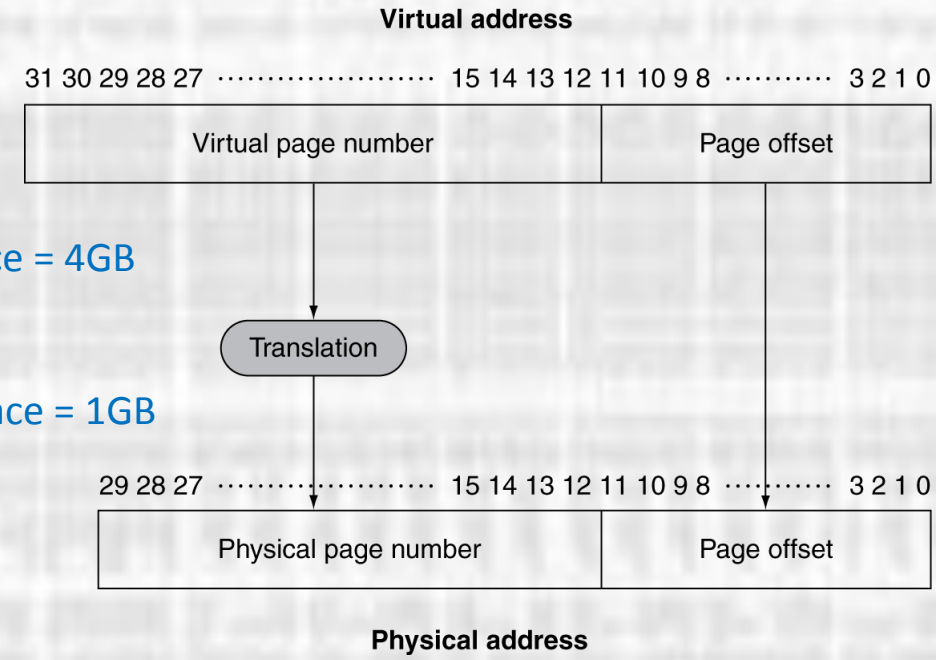
- Overly simplistic example

Virtual address		Physical address
1234	-->	dram 5463
1235	-->	dram 7638
1236	-->	hdd 1254
1456	-->	dram 5281

Virtual Memory

Overview

- Address Translation



Virtual address space = 4GB

Physical address space = 1GB

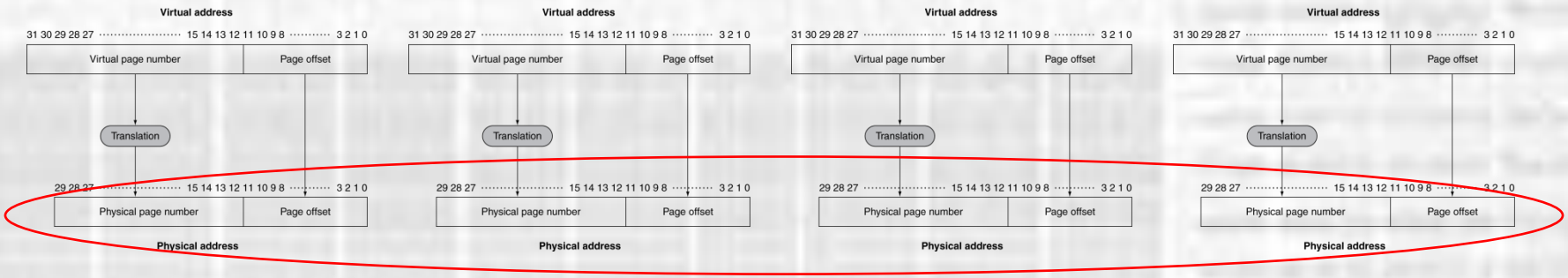
Page offset – the memory location within a page

Page size = 2^{12} = 4KB

Virtual Memory

Overview

- Address Translation
  - Each program (process) will have its own virtual memory space



Only 1 physical memory

  - But there is only 1 physical memory

  - The VMM is responsible for physical memory management
    - Protects programs from overwriting each other
    - Allows programs to share

Virtual Memory
Implementation

- Page Fault
  - The processor addresses a given virtual address
    - If the address maps to the main memory – HIT
    - If the address maps to the secondary memory – Miss = page fault
  - Page fault - requires a page to be read from secondary memory
    - At 10ms access times (HDD) = 10M clock cycles to get the first byte when running at 1GHz
    - Need to transfer 4KB
  - Must reduce page faults to the lowest possible value

Virtual Memory
Implementation

- Page Fault
  - Approaches to reduce page fault impact
    - Page size – large enough to capture large portions of the program
      - 4KB, 16KB typical – but growing
      - 1KB for mobile – smaller aps
    - Fully associative placement
    - Use SW to manage page faults
      - Small overhead compared to the delay of a miss
      - Can be very sophisticated in placement and replacement
    - Use write-back
      - Only when necessary (dirty)

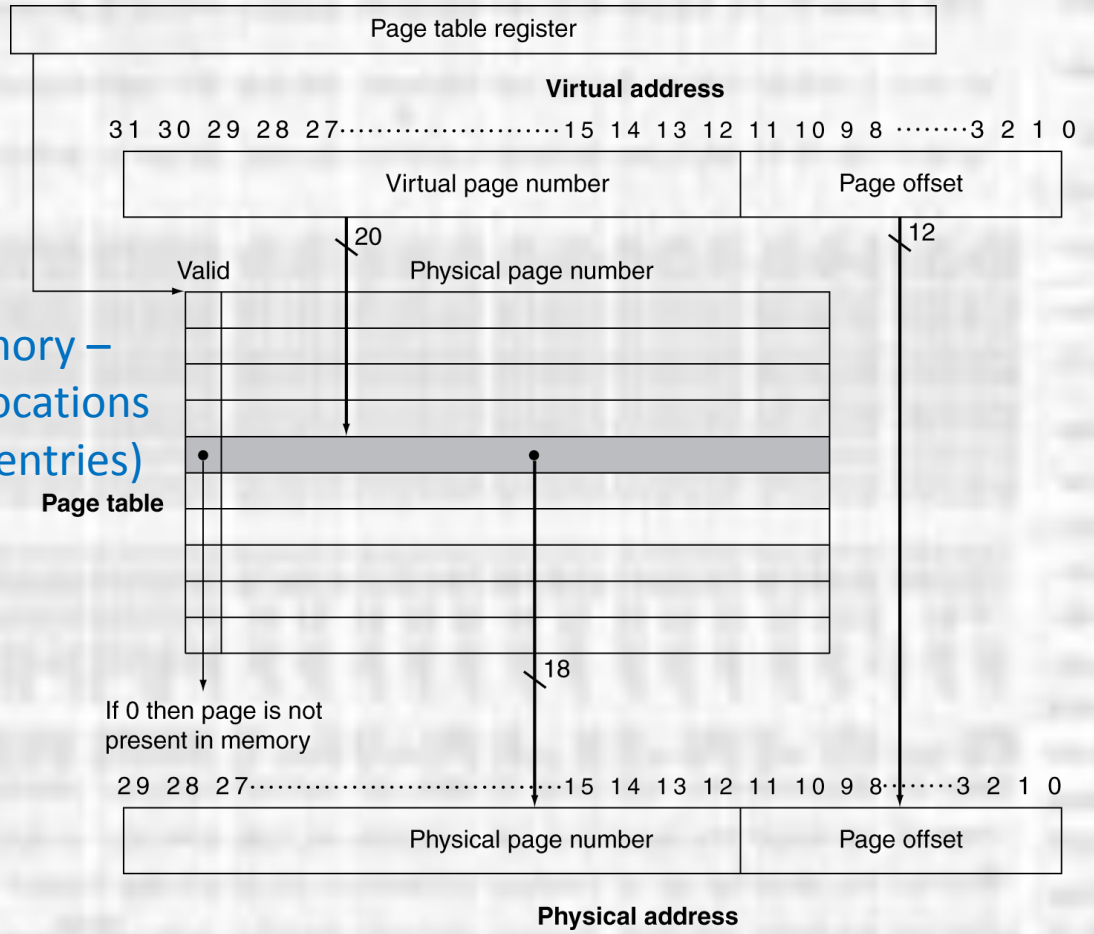
Virtual Memory
Implementation

- Page Tables
  - Remember – fully associative placement is costly due to the time and circuitry needed to find the block (page) you are looking for
  - VM uses the existing memory addressing/decoding capability built into main memory
    - Each program (process) is allocated a space in main memory to place the address translation information for that process – Page Table
    - The processor includes a register to point to the location of the page table for the currently running program – Page Table Register
  - The page table maps each possible virtual memory location to a physical memory location

Virtual Memory
Implementation

- Page Tables

Starting address of the page table in main memory – process dependent



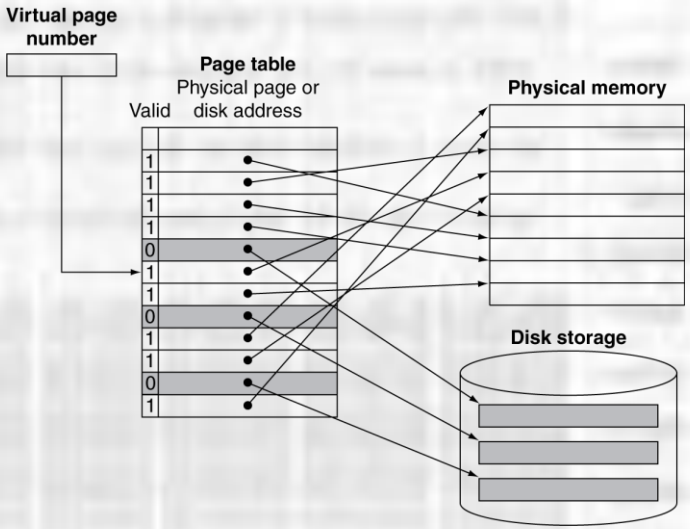
Main memory – memory locations (2^{20} = 2M entries)

Address known to the program and the processor

Physical address to read from / write to

Virtual Memory
Implementation

- Page Tables
  - Logical configuration



  - In most systems – 1 page table but separate data structures
    - 1 for main memory
    - 1 for secondary memory
      - Always need to know where the page is in secondary memory – never changes

Virtual Memory
Implementation

- Page Fault - read
  - Page is not in main memory and the valid bit is NOT set in the page table
    - Processor throws an exception → gives control to the OS
    - OS finds the page on disk (using the HDD portion of the page table)
      - Located in the swap space
    - If necessary – the OS determines which page to write back to the HDD if the memory was full
      - Must be dirty
      - LRU is typical – but with a sophisticated process
      - Often include a reference bit (indicates the page has been referenced recently)
    - OS returns control to the program and re-executes the read
      - This time it finds the page in main memory

Virtual Memory
Implementation

- Page Fault - write
  - Page is not in main memory and the valid bit is NOT set in the page table
    - Processor throws an exception → gives control to the OS
    - OS finds the page on disk (using the HDD portion of the page table)
    - If necessary – the OS determines which page to write back to the HDD if the memory was full
    - OS reads the page in
    - OS returns control to the program and re-executes the write
    - Uses a write-back approach – only writes when it is swapped back out and dirty

Virtual Memory

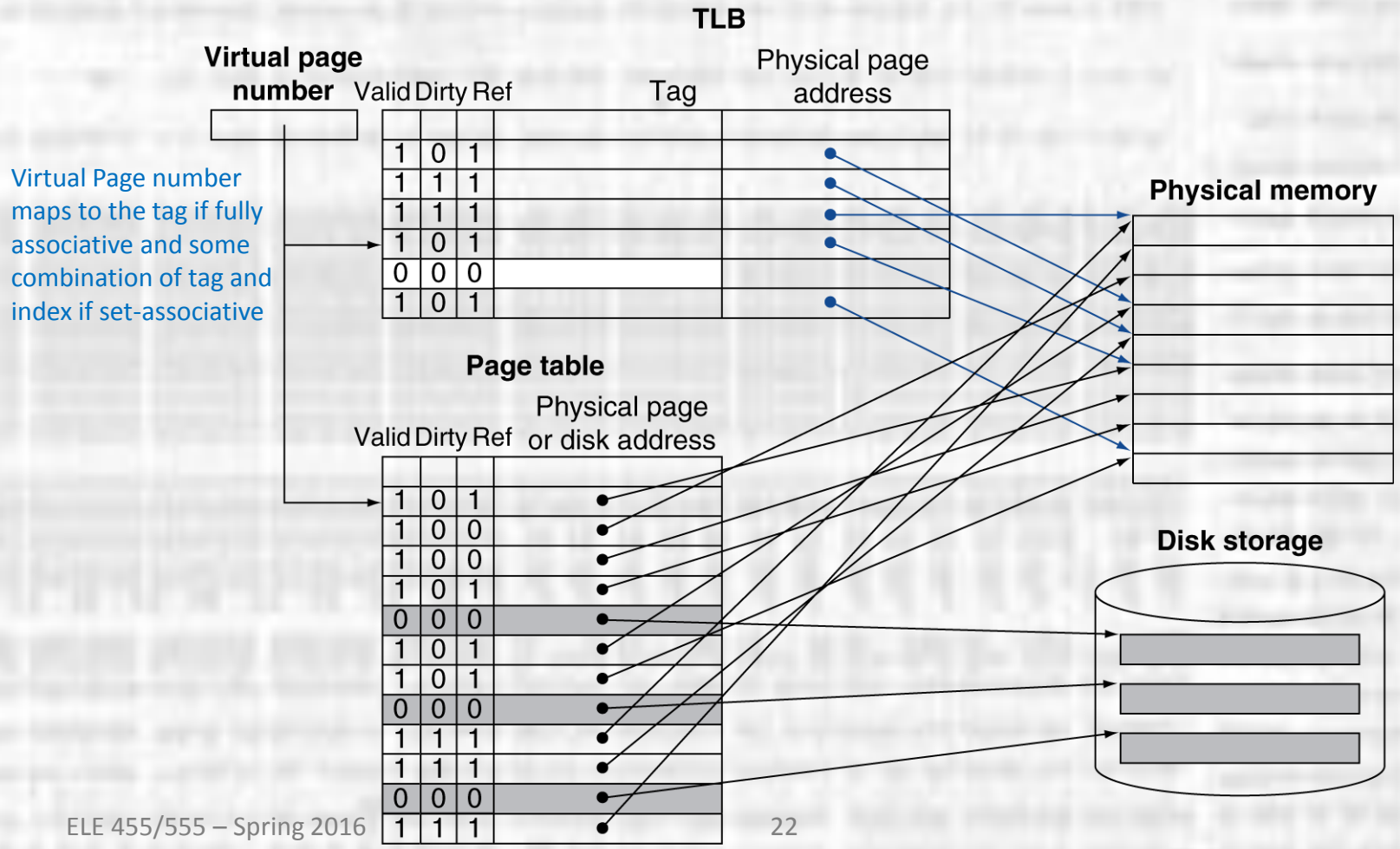
Translation-Lookaside Buffer

- Page Table Issue
  - Every memory access requires 2 accesses
    - 1 – for the page table lookup (translation) – in memory
    - 1 – for the actual memory read/write – in memory
  - Fortunately – Pages have high temporal and spatial locality
  - Create a cache of the translation entries
    - Translation-Lookaside Buffer (TLB)

Virtual Memory

Translation-Lookaside Buffer

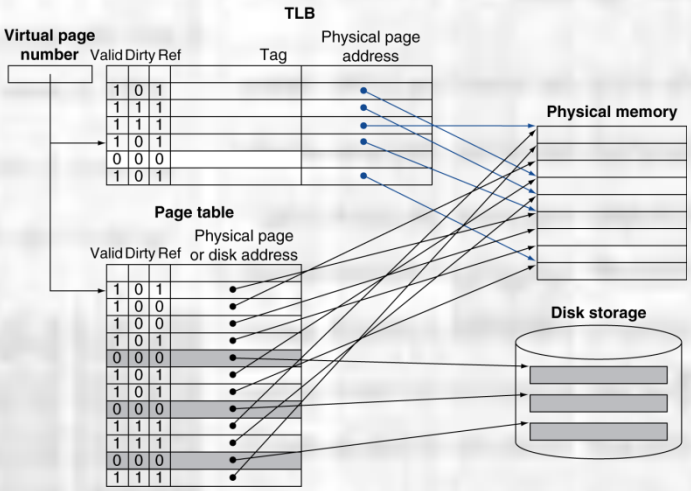
- Translation-Lookaside Buffer



Virtual Memory

Translation-Lookaside Buffer

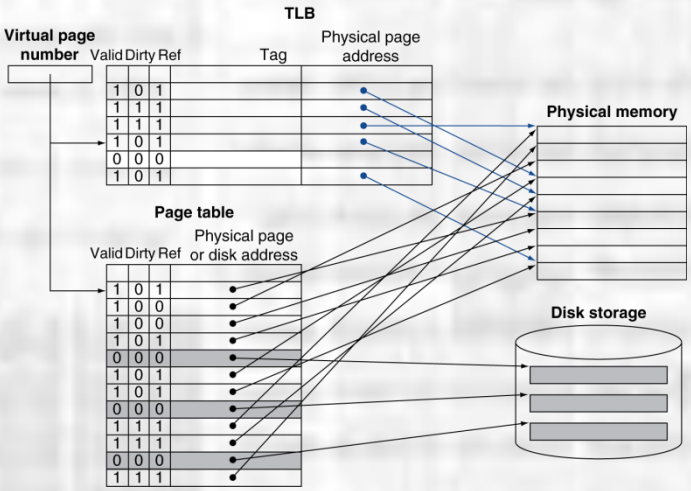
- TLB Hit
  - Page is in main memory
    - Tag matches
    - Valid bit is 1
  - Physical address is provided to the cache
    - Ref bit is set
    - If a write – dirty bit is set



Virtual Memory

Translation-Lookaside Buffer

- TLB Miss
  - Page is not in the TLB
  - Check the page table
    - Hit
      - load the page info into the TLB
        - Must have a replacement policy
      - Provide the physical address to the cache
    - Miss → Page Fault
      - Throw an exception …
      - Once the page is loaded in memory and the entry added to the TLB – re-issue the read/write



Virtual Memory

Translation-Lookaside Buffer

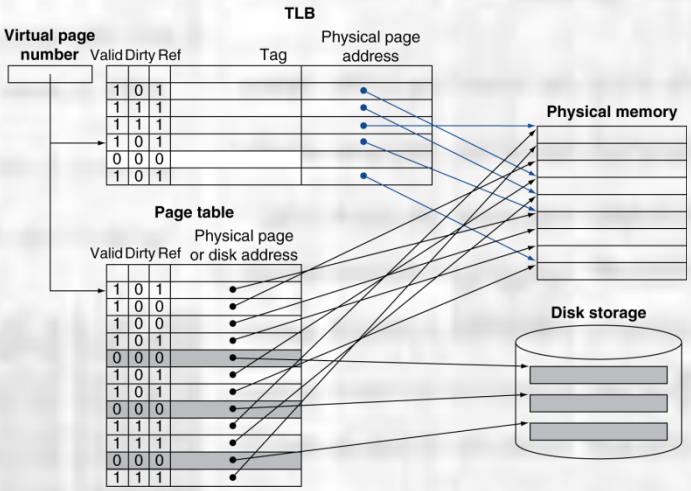
- Possible TLB situations

TLB	Page table	Cache	Possible? If so, under what circumstance?
Hit	Hit	Miss	Possible, although the page table is never really checked if TLB hits.
Miss	Hit	Hit	TLB misses, but entry found in page table; after retry, data is found in cache.
Miss	Hit	Miss	TLB misses, but entry found in page table; after retry, data misses in cache.
Miss	Miss	Miss	TLB misses and is followed by a page fault; after retry, data must miss in cache.
Hit	Miss	Miss	Impossible: cannot have a translation in TLB if page is not present in memory.
Hit	Miss	Hit	Impossible: cannot have a translation in TLB if page is not present in memory.
Miss	Miss	Hit	Impossible: data cannot be allowed in cache if the page is not in memory.

Virtual Memory

Translation-Lookaside Buffer

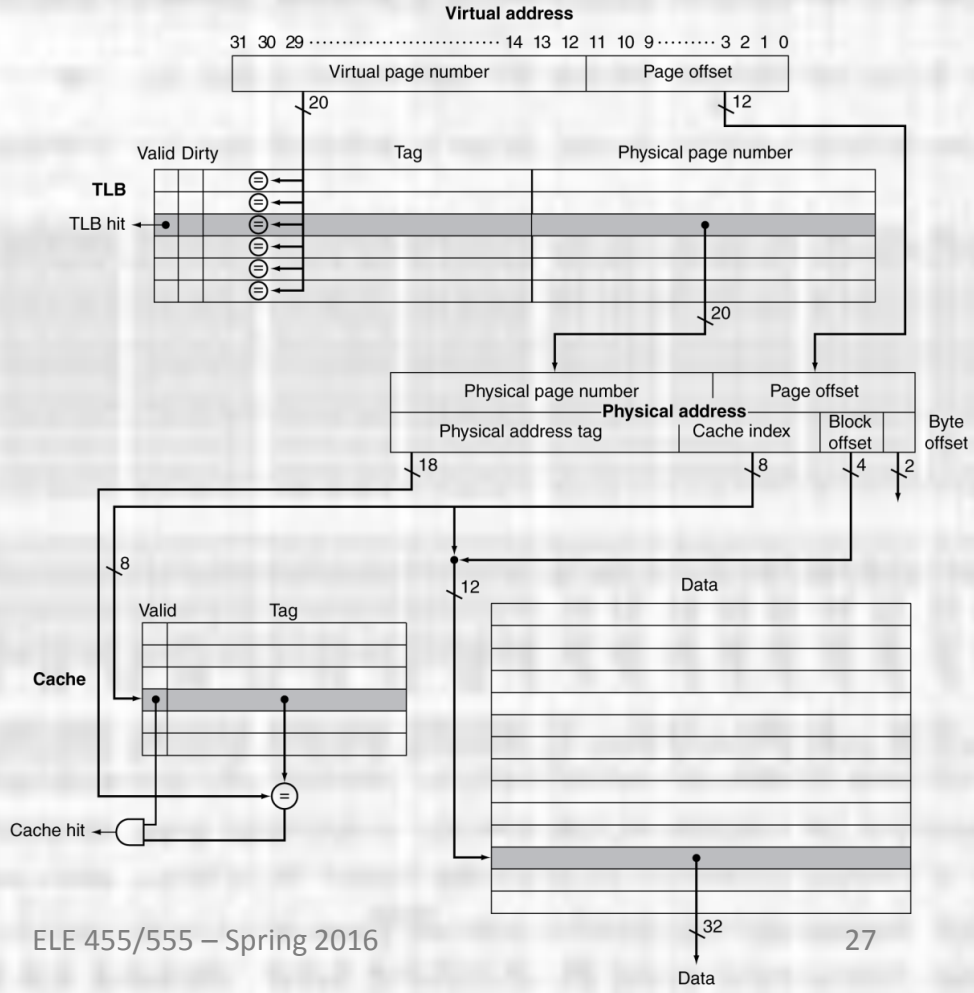
- Typical TLB
  - 16 - 512 blocks
  - 1 - 2 page table entries/block
  - Hit time: 0.5 – 1 clock cycle
  - Miss penalty: 10 - 100 clock cycles
    - TLB miss only – not a page fault
  - TLB Miss rate: 0.01% - 1%

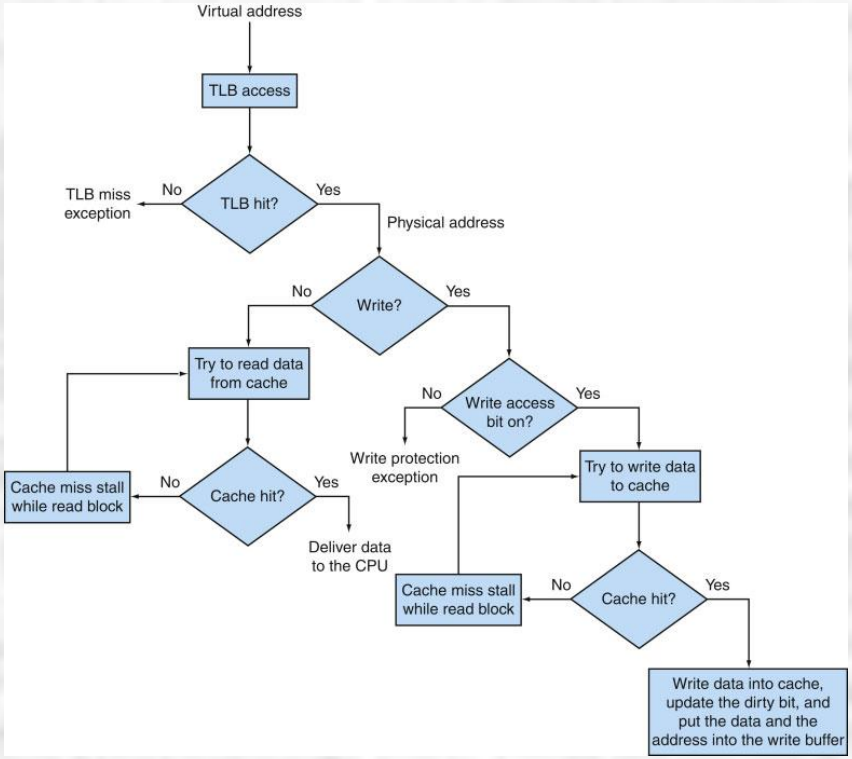


Virtual Memory

Translation-Lookaside Buffer

- Intrinsity





Virtual Memory

Virtually Addressable Cache

- Physical vs Virtual addressed caches
  - We have been working with physically addressed caches
  - There are versions of caches that can use the virtual address
    - virtually indexed, virtually tagged
      - TLB only used in misses
      - Can lead to aliasing when multiple processes access the same page
        - different virtual addresses → same page, changing 1 will not be reflected to the other
    - virtually indexed, physically tagged
      - Extend the mapping of the address to the indexes
      - Reduces some of the options in the VM space
      - Reduces the amount of mapping (page tables, …)

Virtual Memory

Protection

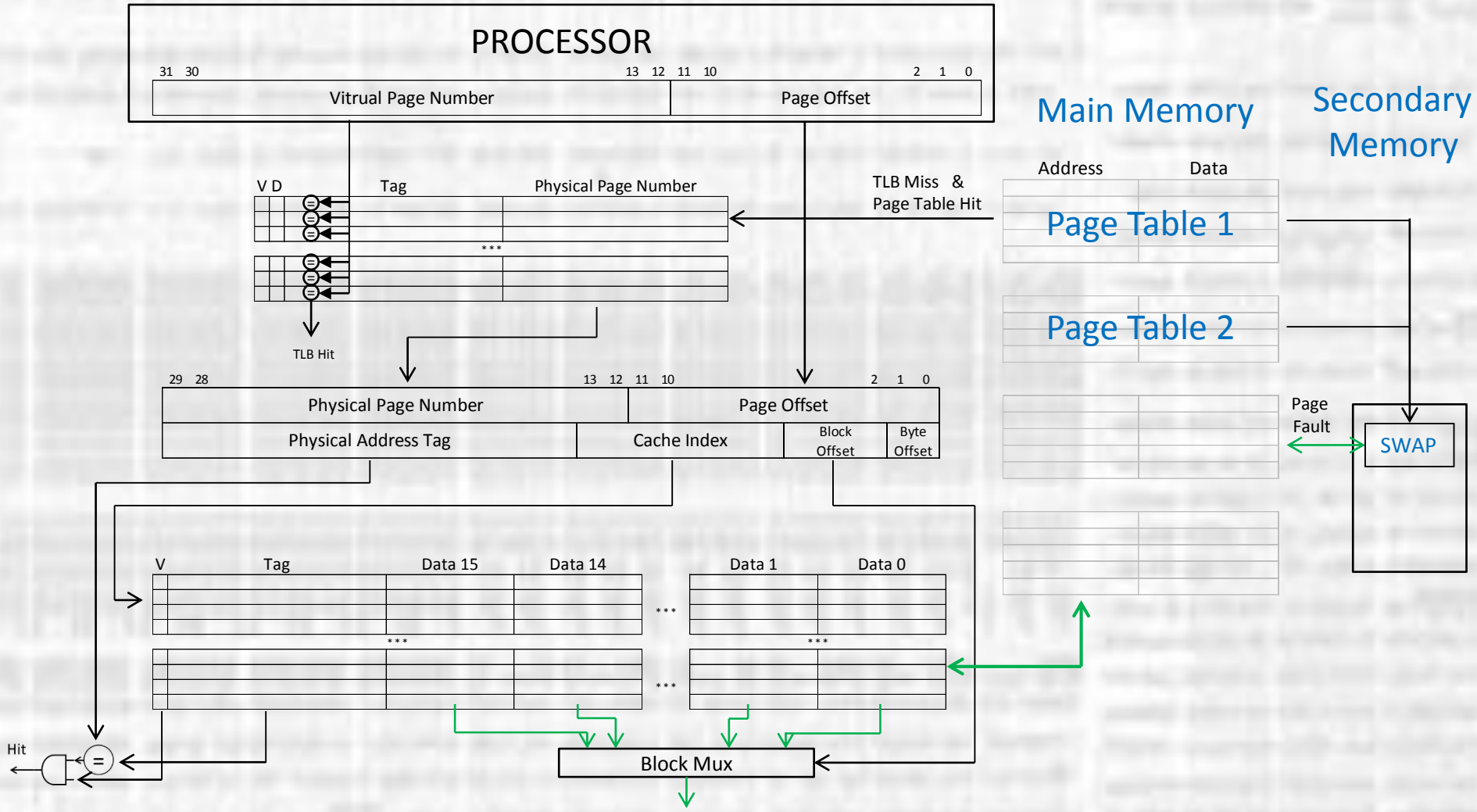
- Virtual Memory and Memory Protection
  - The OS controls the Page Tables
  - Protects processes from damaging each others memory by not placing any common memory translations in multiple page tables
    - If it's not in the page table – the process can't get to it
  - The OS can allow sharing
    - Program
      - Multiple mapping to the same location
      - Write access turned off
    - Data
      - Multiple mapping to the same location
      - Write access may be on or off

Virtual Memory
Protection

- Virtual Memory and Memory Protection
  - To provide protection the system needs 4 capabilities
    - Support for a user mode and a supervisor (kernal, executive) mode
      - Supervisor mode bit
    - Portions of the processor state can only be changed in supervisor mode
      - Special instructions
      - Page table and page table pointer
      - TLB
    - Ability to switch back and forth
      - To transition to supervisor mode – “syscall” executed on an exception
      - To return to user mode – “eret” to load the EPC into the PC and return
    - A portion of the address space for the OS and protected from all users

Virtual Memory

Cache Review



Virtual Memory

Cache Review

Start a new Process

OS: Copy process VM to swap
Create page table: map to HDD, not to MM
Start process

P: Request instruction using virtual address

→ TLB Miss
→ Page table miss
→ Page Fault
→ Exception

OS: Use page table to find page in swap
Transfer page to main memory
Update page table
Update TLB
Return

P: Repeat request

→ TLB Hit
→ Cache miss
→ Stall processor and load cache from main memory

P: Repeat request

→ TLB Hit
→ Cache hit
→ Request served