CSE 120
Principles of Operating Systems

Fall 2021

Lecture 10: Paging

Yiying Zhang

Lecture Overview

We'll cover more paging mechanisms:

- Optimizations
  - Managing page tables (space)
  - Efficient translations (TLBs) (time)
  - Demand paged virtual memory (space), next lecture

- Midterm grades to be released soon
- Homework 3 out
- Work on your project 2!

All problems in computer science can be solved by another level of indirection

Butler Lampson

All problems in computer science can be solved by another level of indirection

Butler Lampson

David Wheeler

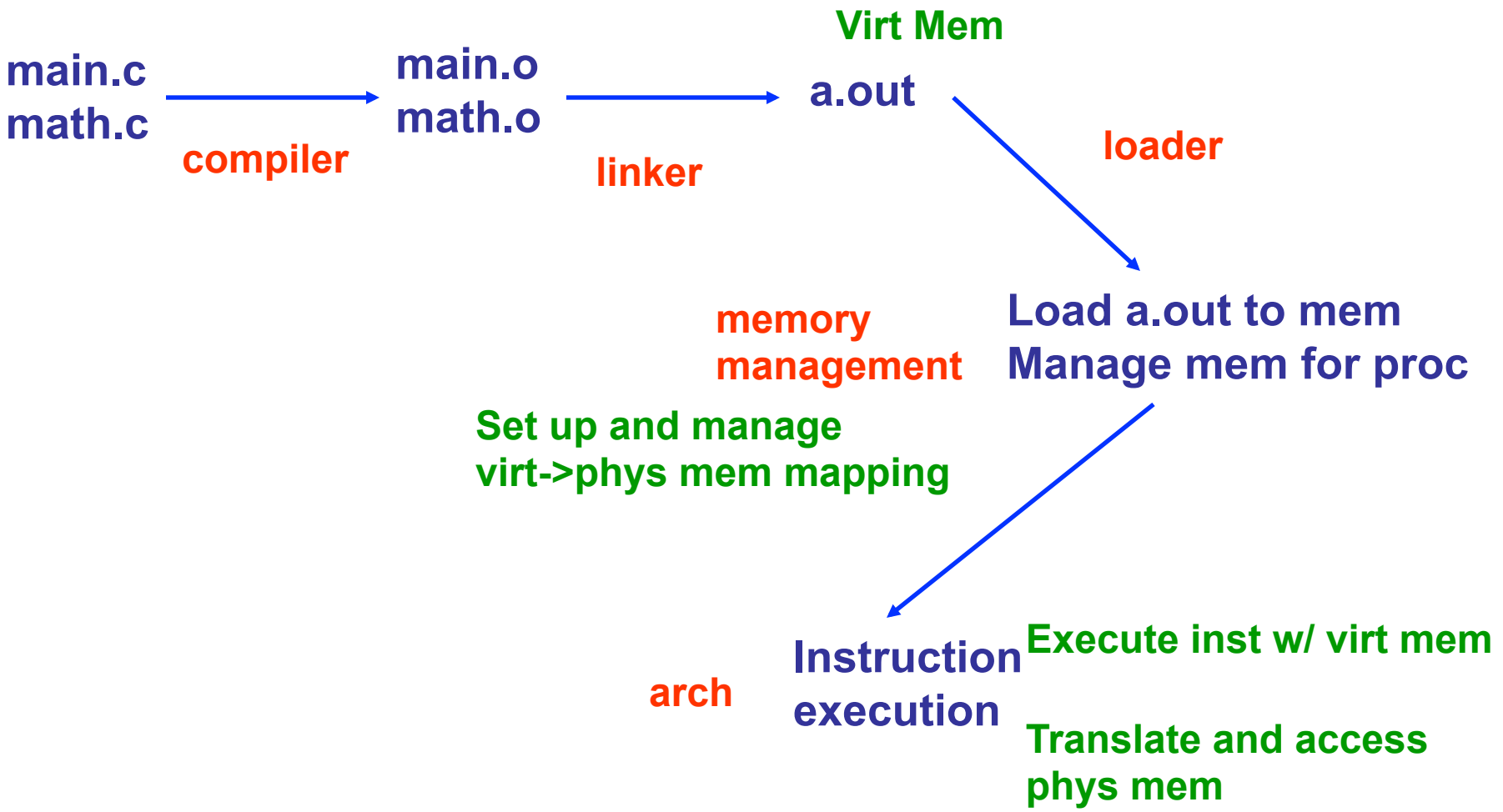
but that usually will create another problem

– David Wheeler

Summary: Evolution of Memory Management (before paging)

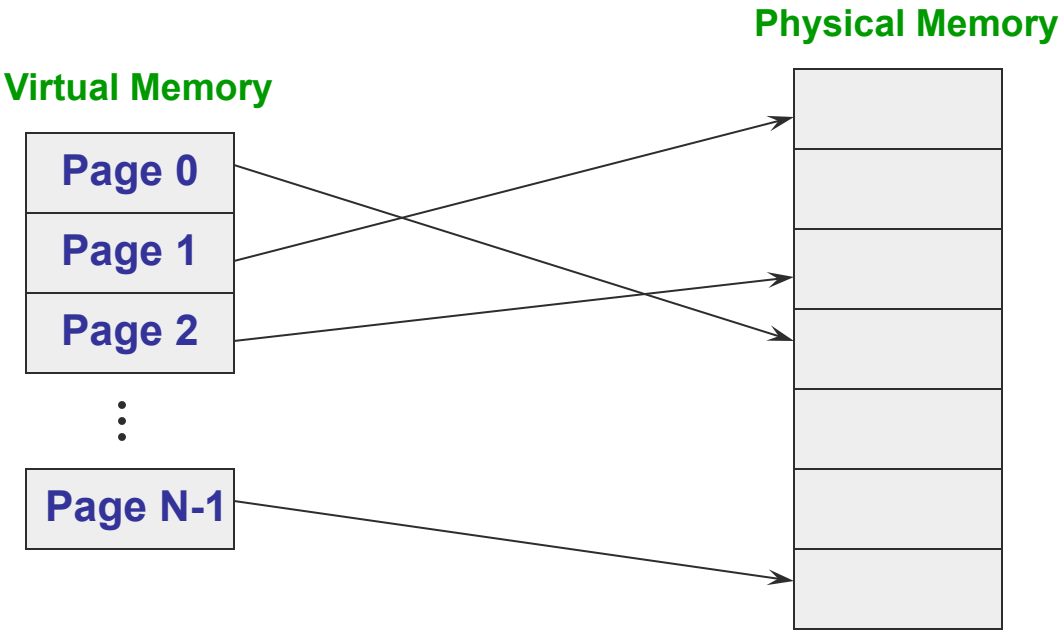
Scheme	How	Pros	Cons
Simple uniprogramming	1 segment loaded to starting address 0	Simple	1 process 1 segment No protection
Simple multiprogramming	1 segment relocated at loading time	Simple, Multiple processes	1 segment/process No protection External frag.
Base & Bound	Dynamic mem relocation at runtime	Simple hardware, Multiple processes Protection	1 segment/process, External frag.
Multiple segments	Dynamic mem relocation at runtime	Sharing, Protection, multi segs/process	More hardware, External frag.

[lec9] The Big Picture



[lec9] Paging

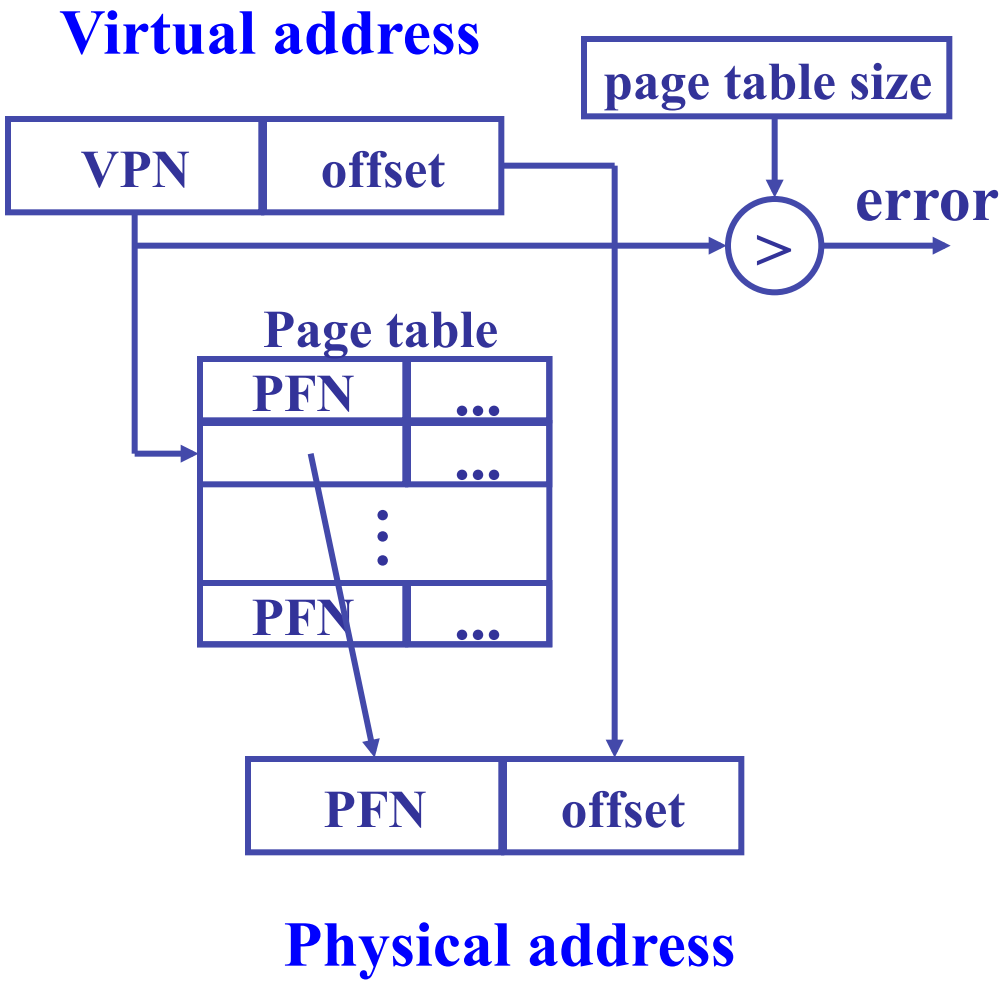
- Paging solves the external fragmentation problem by using fixed sized units in both physical and virtual memory



[lec9] Paging

- Translating addresses
  - Virtual address has two parts: virtual page number and offset
  - Virtual page number (VPN) is an index into a page table
  - Page table determines page frame number (PFN)
  - Physical address is PFN::offset (“::” means concatenate)
- Page tables
  - Map virtual page number (VPN) to page frame number (PFN)
    - » VPN is the index into the table that determines PFN
  - One page table entry (PTE) per page in virtual address space
    - » Or, one PTE per VPN

[lec9] Paging



- Context switch
  - similar to the segmentation scheme
- Pros:
  - easy to allocate memory
  - easy to swap
  - easy to share

[lec9] Paging Example

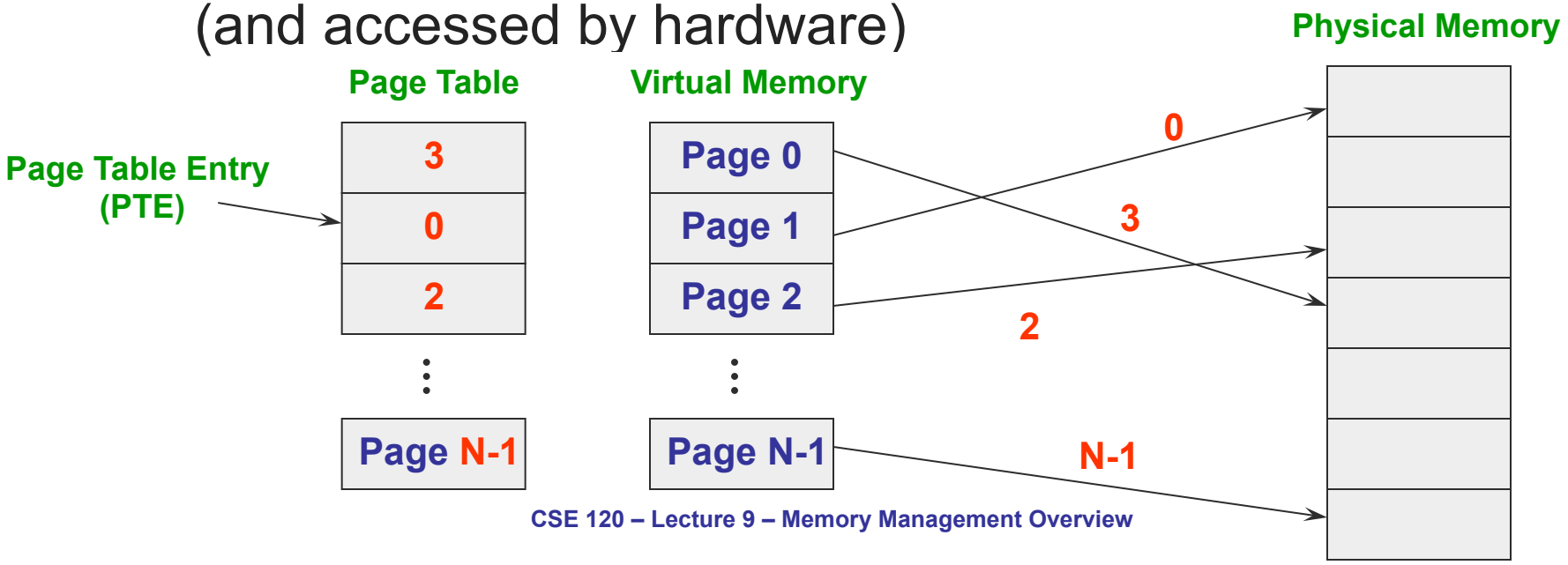
- Pages are 4K
  - 4K → offset is 12 bits → VPN is 20 bits (2^{20} VPNs), assuming 32bit system
- Virtual address is 0x7468
  - Virtual page is 0x7, offset is 0x468 (lowest 12 bits of address)
- Page table entry 0x7 contains 0x2
  - Page frame number is 0x2
  - Seventh virtual page is at address 0x2000 (physical page 2)
- Physical address = 0x2000 :: 0x468 = 0x2468

Summary so far

- Virtual memory
  - Processes use virtual addresses
  - Hardware translates virtual address into physical addresses with OS support
- Evolution of techniques
  - Single, fixed physical segment per process (no virt mem)
  - Single segment per process, static relocation (no virt mem)
  - Base-and-bound – dynamic relocating whole process
  - Segmentation – multiple (variable-size) segments with dynamic relocation
  - Paging – small, fixed size pages

Page Tables

- Page tables completely define the mapping between virtual pages and physical pages for an address space
- Each process has an address space, so each process has a page table
- Page tables are data structures maintained by the OS (and accessed by hardware)



Page Table Entries (PTEs)

- Page table entries control mapping
  - The Modify bit says whether or not the page has been written
    - It is set when a write to the page occurs
  - The Reference bit says whether the page has been accessed
    - It is set when a read or write to the page occurs
  - The Valid bit says whether or not the PTE can be used
    - It is checked each time the virtual address is used
  - The Protection bits say what operations are allowed on page
    - Read, write, execute
  - The page frame number (PFN) determines physical page

Paging implementation – how does it really work?

- Where to store page table?
- How to use MMU?
  - Even small page tables are too large to load into MMU
  - Page tables kept in mem and MMU only has their base addresses
- What happens at context switches?

Paging Advantages

- Easy to allocate (physical) memory
  - Memory comes from a free list of fix-sized chunks
  - Allocating a page is just removing it from the list
  - External fragmentation not a problem
- Easy to swap out chunks of a program
  - All chunks are the same size
  - Use valid bit to detect references to swapped pages
  - Pages are a convenient multiple of the disk block size
  - More on swapping next time

Paging Limitations

- Can still have internal fragmentation
  - Process may not use memory in multiples of a page
- Memory reference overhead
  - 2 references per address lookup (page table, then memory)
  - Solution – use a hardware cache of lookups (next lec)
- Memory required to hold page table can be significant

Deep thinking

- Why does the page table we talked about so far have to be contiguous in the physical memory?
  - Why did a segment have to be contiguous in memory?
- For a 4GB virtual address space, we just need 1M PTE (~4MB), what is the big deal?
- My PC has 2GB, why do we need PTEs for the entire 4GB address space?

How many PTEs do we need?
(assume page size is 4096 bytes)

- Worst case for 32-bit address machine?

- What about 64-bit address machine?

- Page size?
  - Small page -> big table
  - Large page ->small table but large internal fragmentation

Managing Page Tables

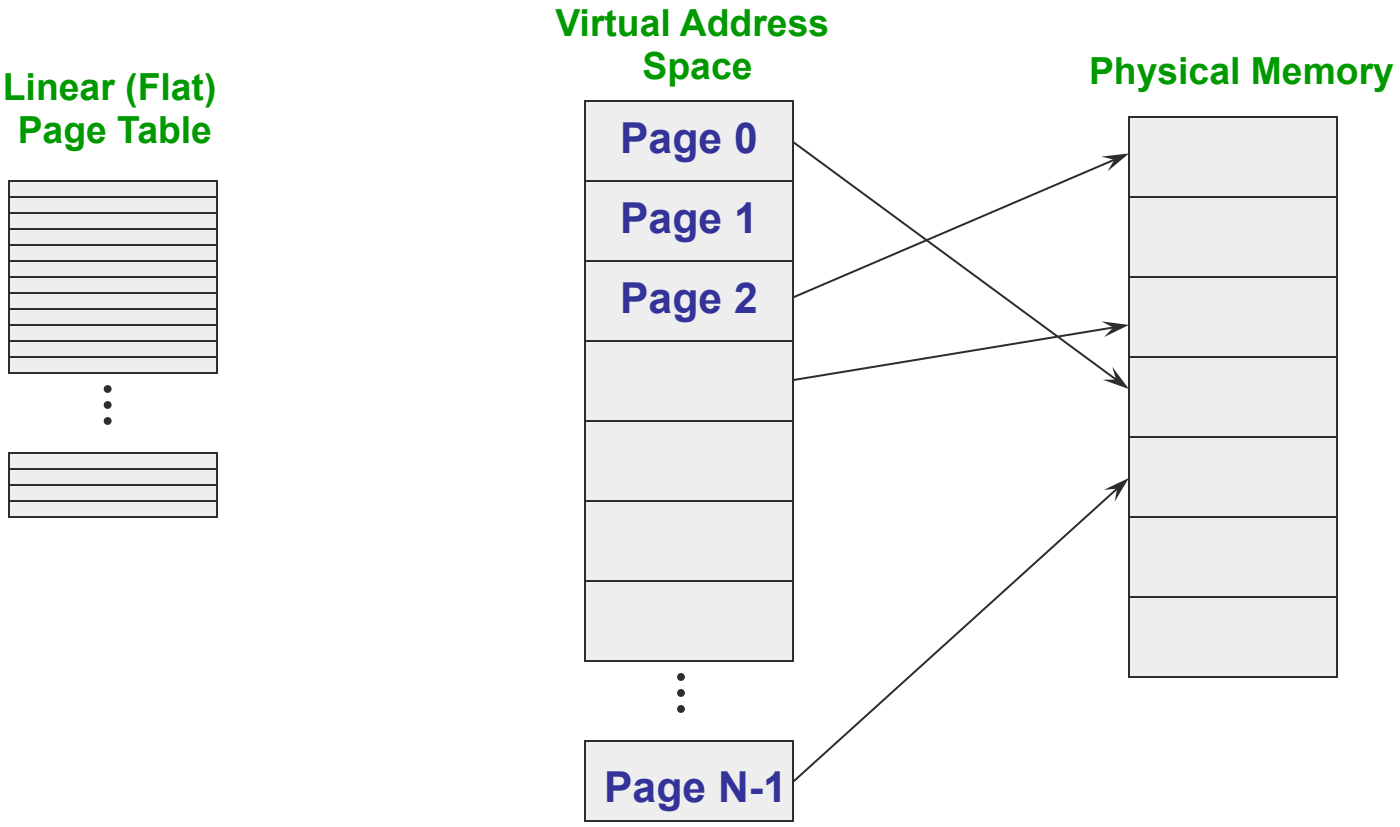
- How can we reduce page table space overhead?
  - Observation: Only need to map the portion of the address space actually being used (tiny fraction of entire addr space)

- How can we be flexible?

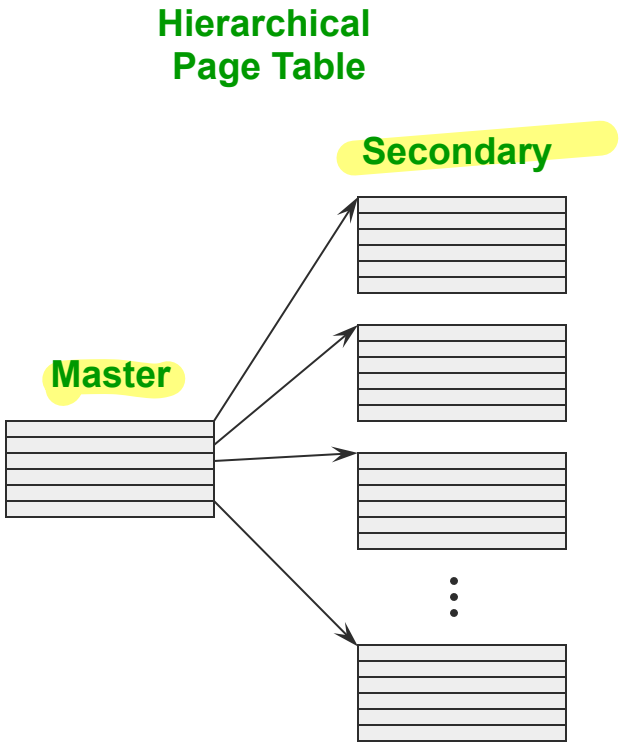
  "All computer science problems can be solved with an extra level of indirection."

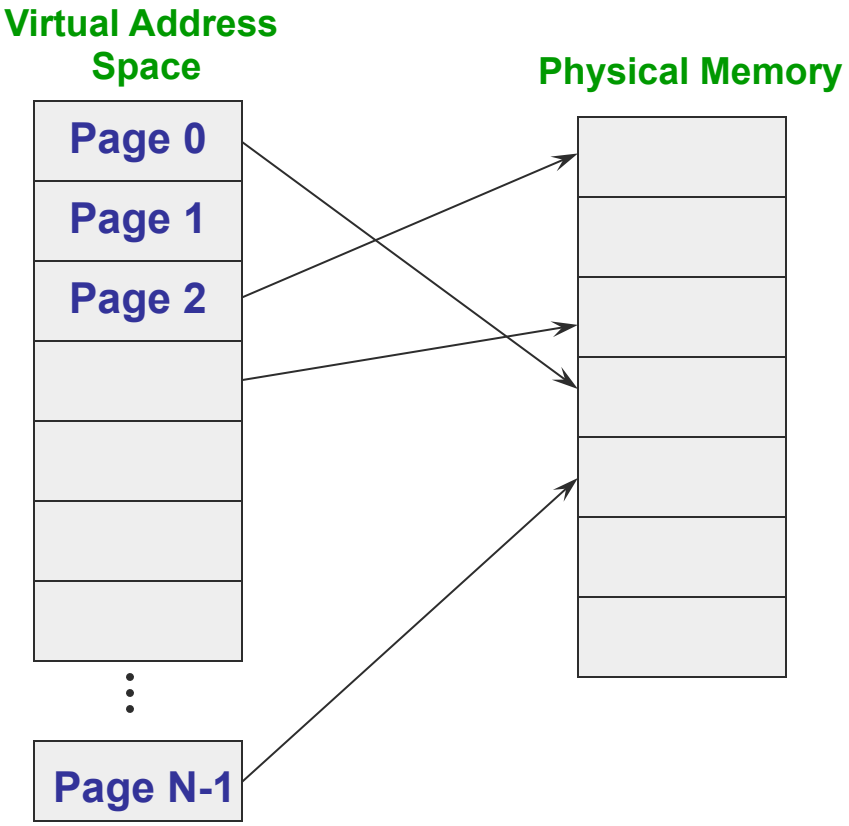
  two-level page tables

Page Table Evolution

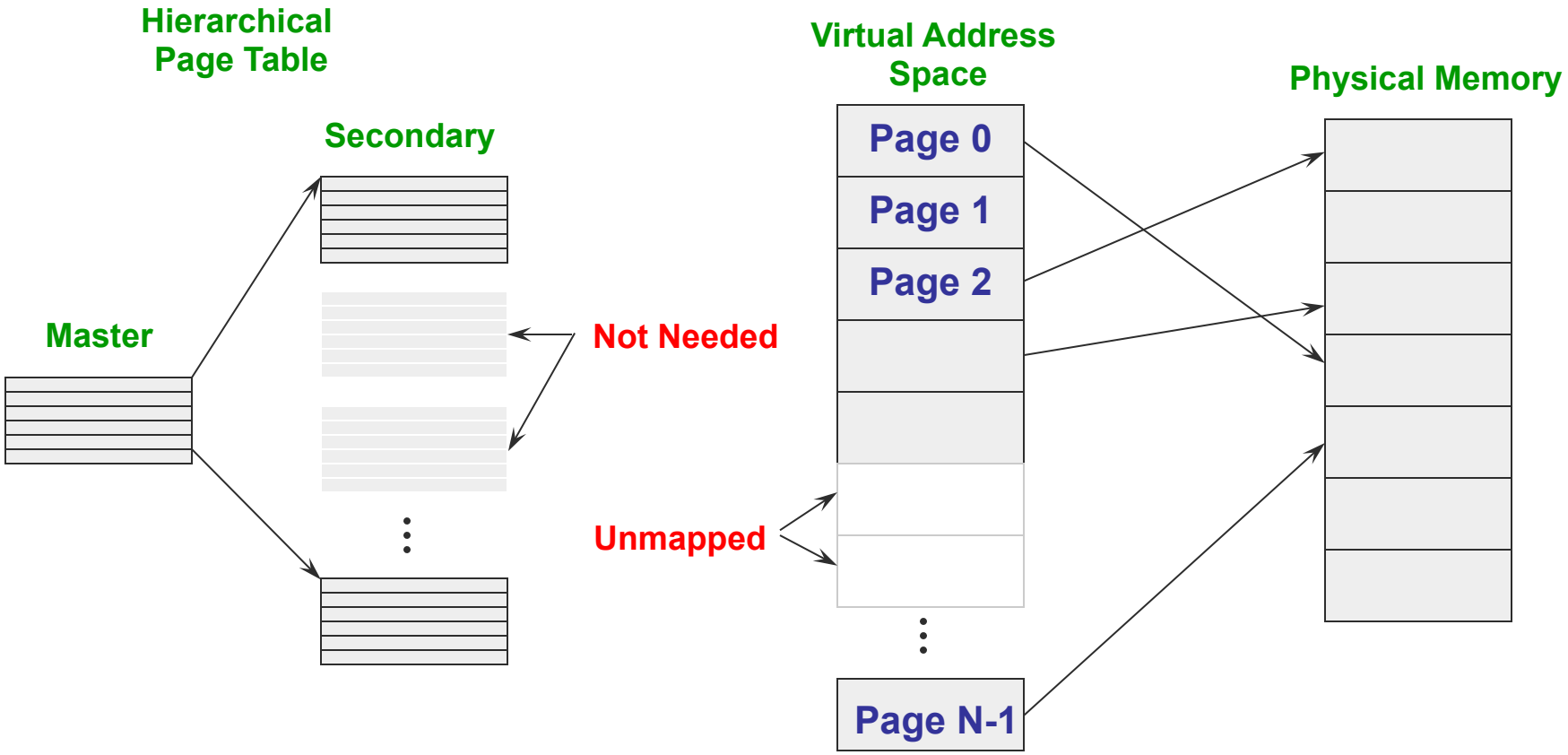


Page Table Evolution





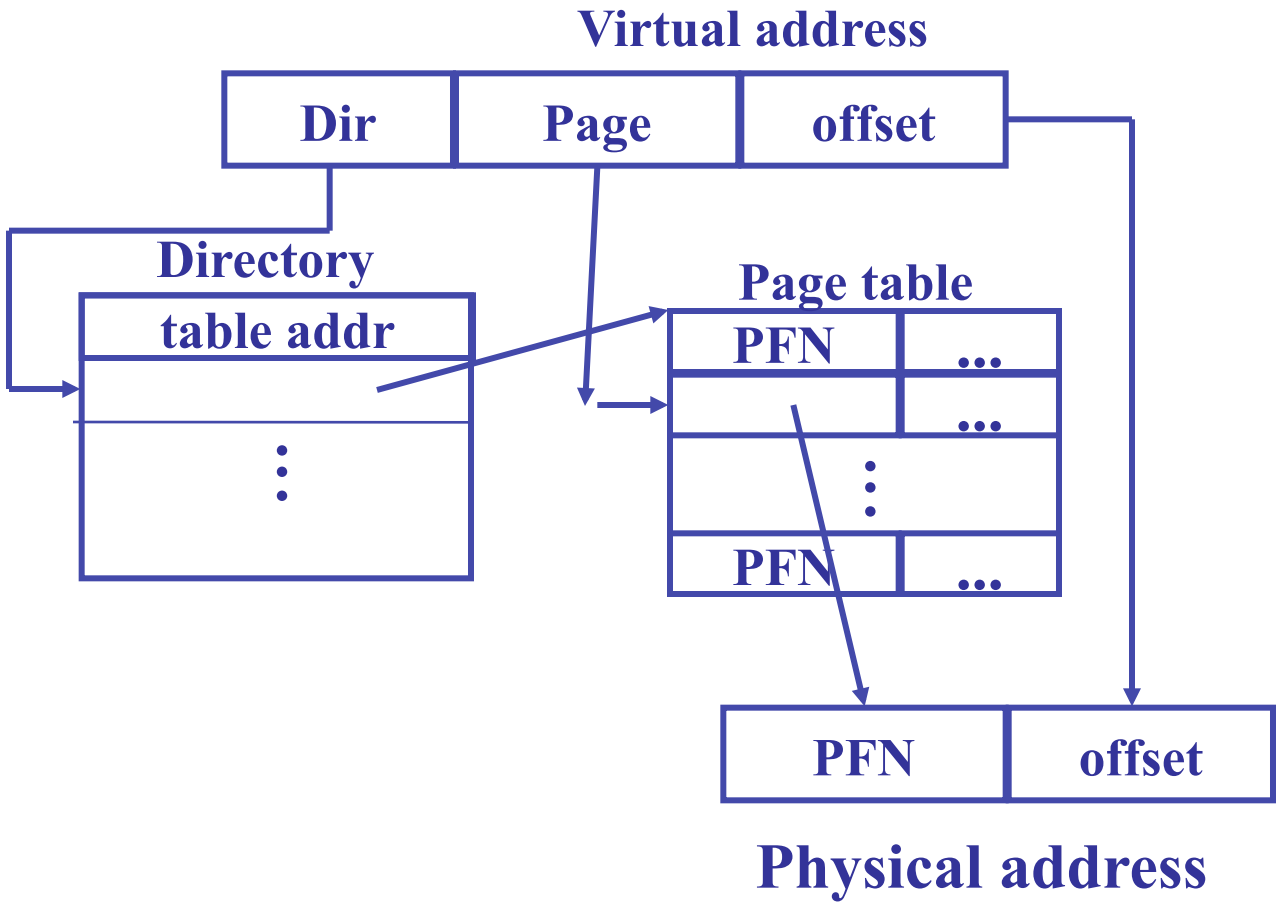
Page Table Evolution



Two-Level Page Tables

- Two-level page tables
  - Virtual addresses (VAs) have three parts:
    - Directory (master page number), secondary page number, and offset
  - Directory page table maps VAs to secondary page table
  - Secondary page table maps page number to physical page
  - Offset indicates where in physical page address is located

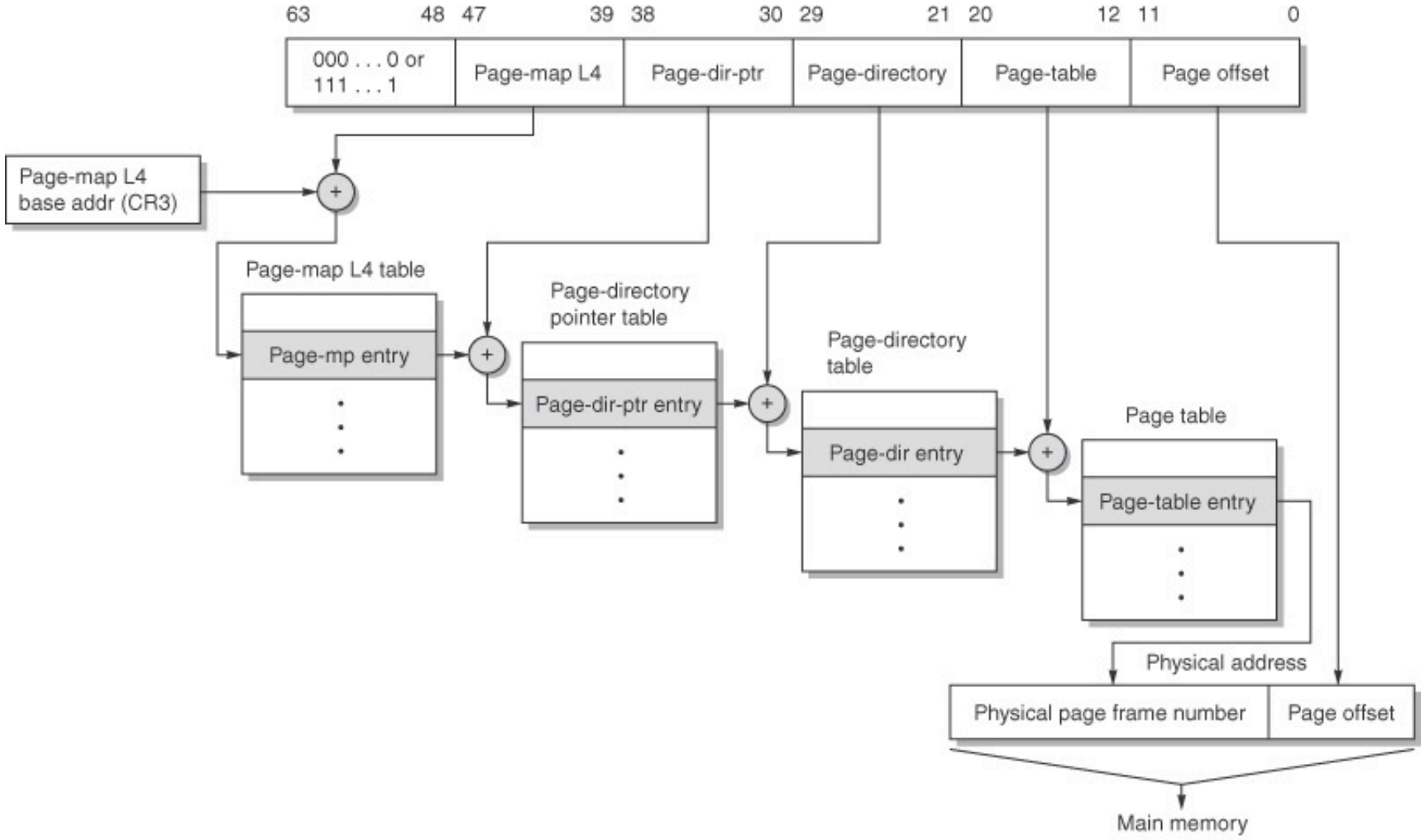
Two-Level Page Tables



Two-Level Page Tables

- Example
  - 4KB pages, 4 bytes/PTE, 32-bit address space
  - How many bits in offset?
    - 4KB = 12 bits
  - Want directory page table in one page, how many entries can we have in the directory page table?
    - 4KB/4 bytes = 1KB entries (each entry is a 32-bit address)
  - Hence, 1KB secondary page tables. How many bits?
  - Directory (1KB) = 10, offset = 12, 32 – 10 – 12 = 10 bits left
    - One secondary page table can host 4K/4bytes=1KB PTEs
    - 10 bits (inner) => exactly 1KB PTEs

Multiple-level page tables



Multi-level page tables

- 3 Advantages?
  - L1, L2, L3 tables do not have to be consecutive
  - They do not have to be allocated before use!
  - They can be swapped out to disk!

The power of an extra level of indirection!

- Problems?

Efficient Translations

- Our original page table scheme already increased the cost of doing memory lookups
  - Two lookups into the page table, another to fetch the data
  - One lookup and one data access for original flat page table
- Now 4-level page tables require five DRAM accesses for one memory operation!
  - Four lookups into the page tables, a fifth to fetch the data
- Solution: *reference locality!*
  - In a short period of time, a process is likely accessing only a few pages
  - Store part of the page table that is “hot” in a fast hardware unit

Next time

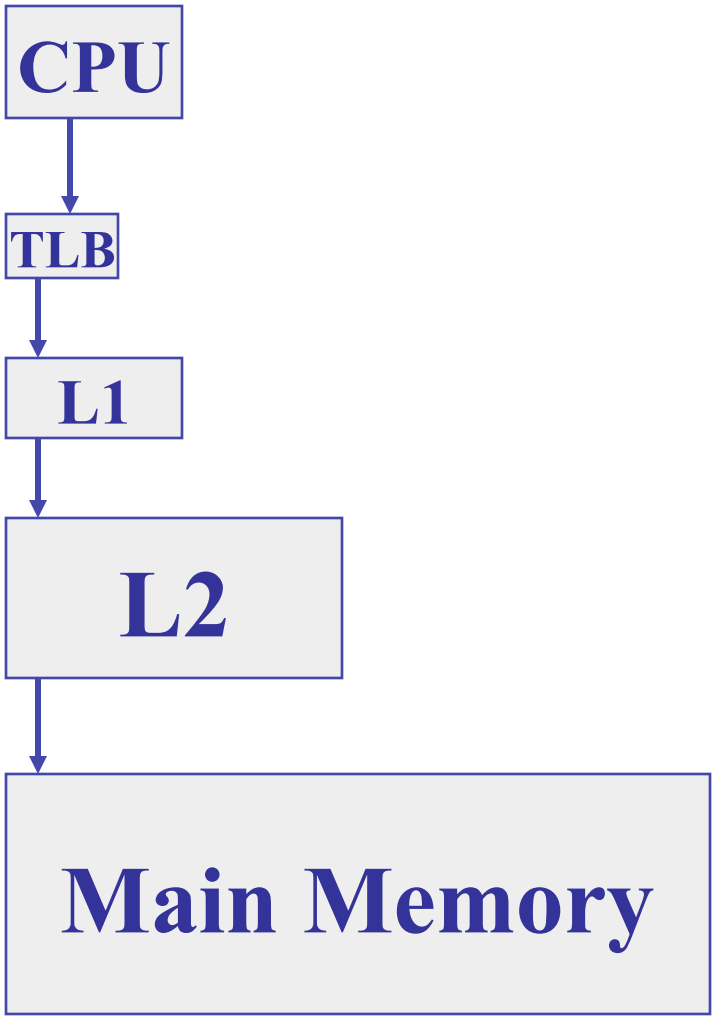
- Swapping, memory allocation, memory sharing

Backup Slides

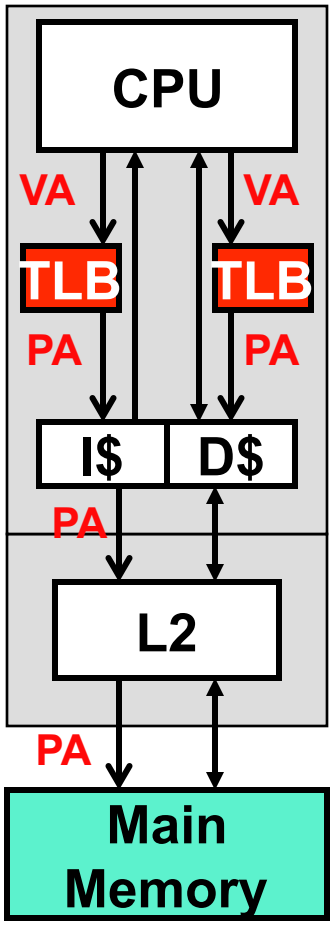
Memory Hierarchy Revisited

What does this imply about L1 addresses?

Where do we hope requests get satisfied?

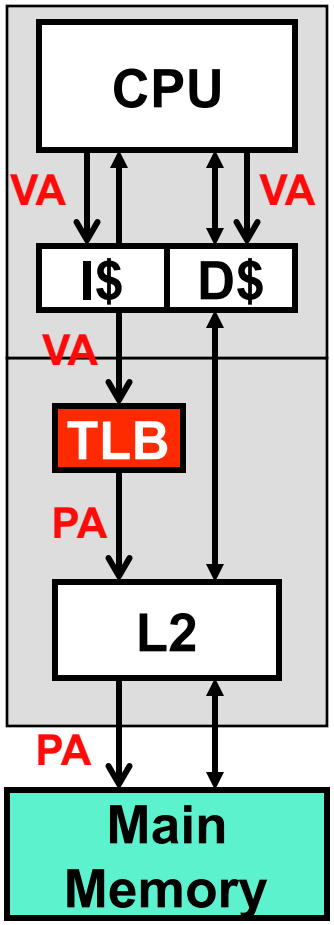


Physical (Address) Caches



- Memory hierarchy so far: **physical caches**
  - ♦ Indexed and tagged by PAs
    - » Physically Indexed (PI)
    - » Physically Tagged (PT)
  - ♦ Translate to PA to VA at the outset
  - \+ Cached inter-process communication works
    - » Single copy indexed by PA
  - – Slow: adds at least one cycle to t_{hit}

Virtual Caches (VI/VT)

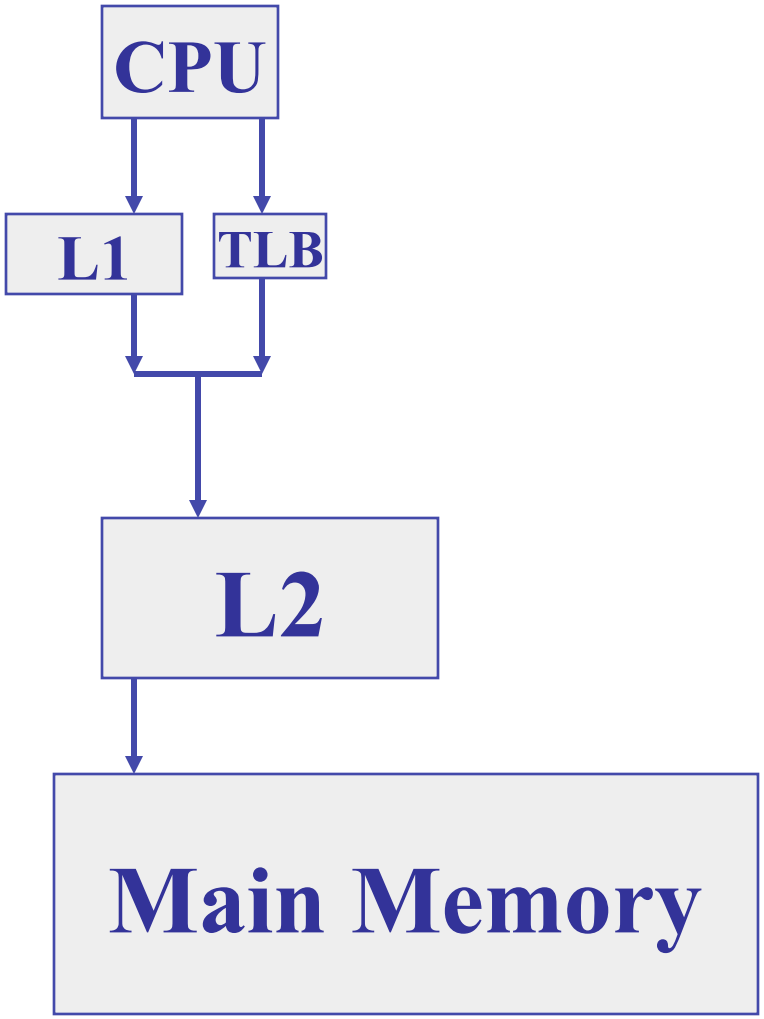


- Alternative: **virtual caches**
  - Indexed and tagged by VAs (VI and VT)
  - Translate to PAs only to access L2
  - \+ Fast: avoids translation latency in common case
  - – Problem: VAs from ***different processes*** are distinct physical locations (with different values) (call homonyms)
- What to do on process switches?
  - Flush caches? Slow
  - Add process IDs to cache tags
- Does inter-process communication work?
  - **Synonyms**: multiple VAs map to same PA
    - » Can't allow same PA in the cache twice
    - » Also a problem for DMA I/O
  - Can be handled, but very complicated

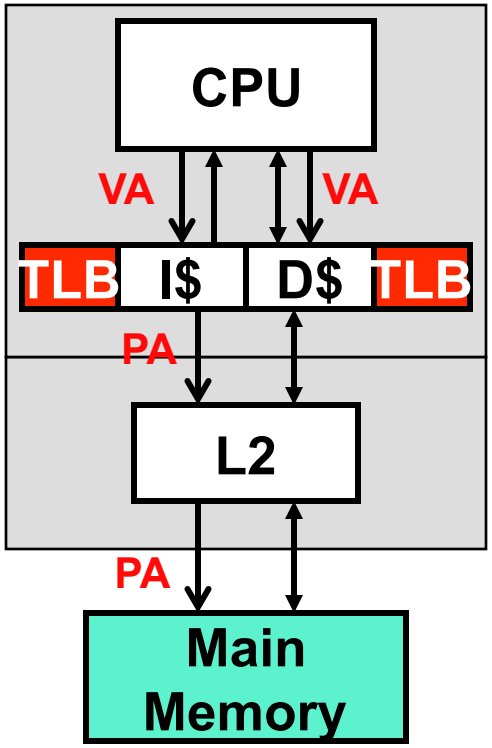
Memory Hierarchy Re-Revisited

What does this imply about L1 addresses?

Any speed benefits?
Any drawbacks?



Parallel TLB/Cache Access (VI/PT)



- Compromise: **access TLB in parallel**
  - *In small caches, index of VA and PA the same*
    - *VI == PI*
  - Use the VA to index the cache
  - Tagged by PAs
  - Cache access and address translation in parallel
  - + No context-switching/aliasing problems
  - + Fast: no additional t_{hit} cycles
  - Common organization in processors today