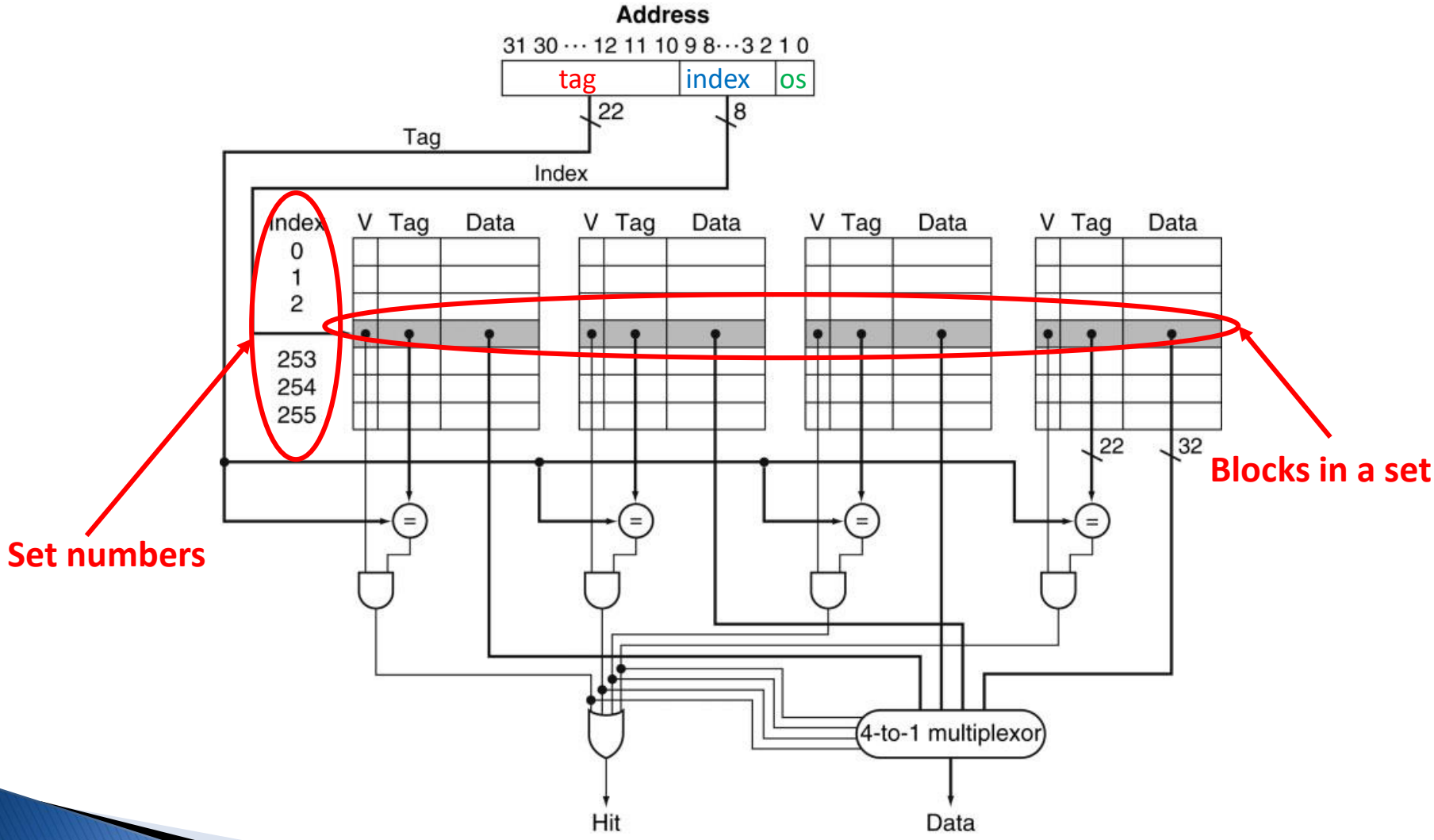
CSE 140
Computer Architecture

Lecture 8 – Virtual Memory (1)

Announcement

- HW #3a
  - Due in 2 weeks (same as HW #3 – out next week)
- Project #1
  - Due 10/11 (Friday) at 11:59pm
- Reading assignment #4
  - Chapter 5.7 – 5.8
    - Do all Participation Activities in each section
    - Access through CatCourses
    - Due Thursday (9/26) at 11:59pm
  - Basic idea of using cache (at CatCourses, under Files)
    - ***How L1 and L2 CPU Caches Work.pdf***

4-Way Set Associative Cache Circuit



Block Replacement Policy

- Direct-Mapped Cache
  - index completely specifies which position a block can go in on a miss
- N-Way Set Assoc.
  - index specifies a set, but block can occupy any position within the set on a miss
- Fully Associative
  - block can be written into any position
- Question: if we have the choice, where should we write an incoming block to?
  1. If there are any locations with valid bit off (empty), then usually write the new block into the first one.
  2. If all possible locations already have a valid block, we must pick a replacement policy: rule by which we determine which block gets "cached out" on a miss.

Block Replacement Policy: LRU

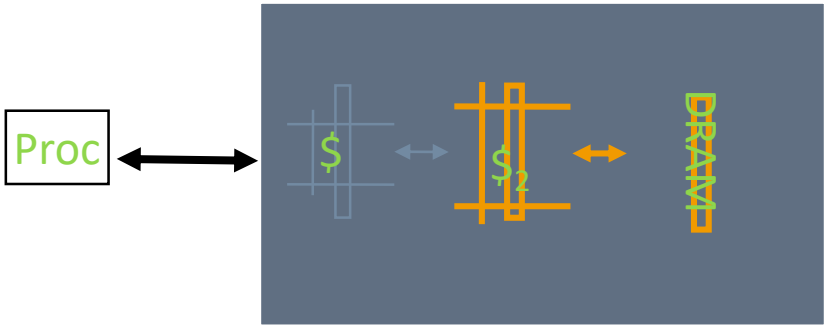
- LRU (Least Recently Used)
  - Idea: cache out block which has been accessed (read or write) least recently – not necessarily the oldest one!
  - Pro: temporal locality → recent past use implies likely future use: in fact, this is a very effective policy
  - Con: with 2-way set assoc, easy to keep track (one LRU bit); with 4-way or greater, requires complicated hardware and much time to keep track of this

Big Idea

- How to choose between associativity, block size, replacement & write policy?
- Design against a performance model
  - Minimize: Average Memory Access Time
    - = Hit Time + Miss Penalty x Miss Rate
  - Influenced by technology & program behavior
- Create the illusion of a memory that is large, cheap, and fast - on average
- How can we improve miss penalty?

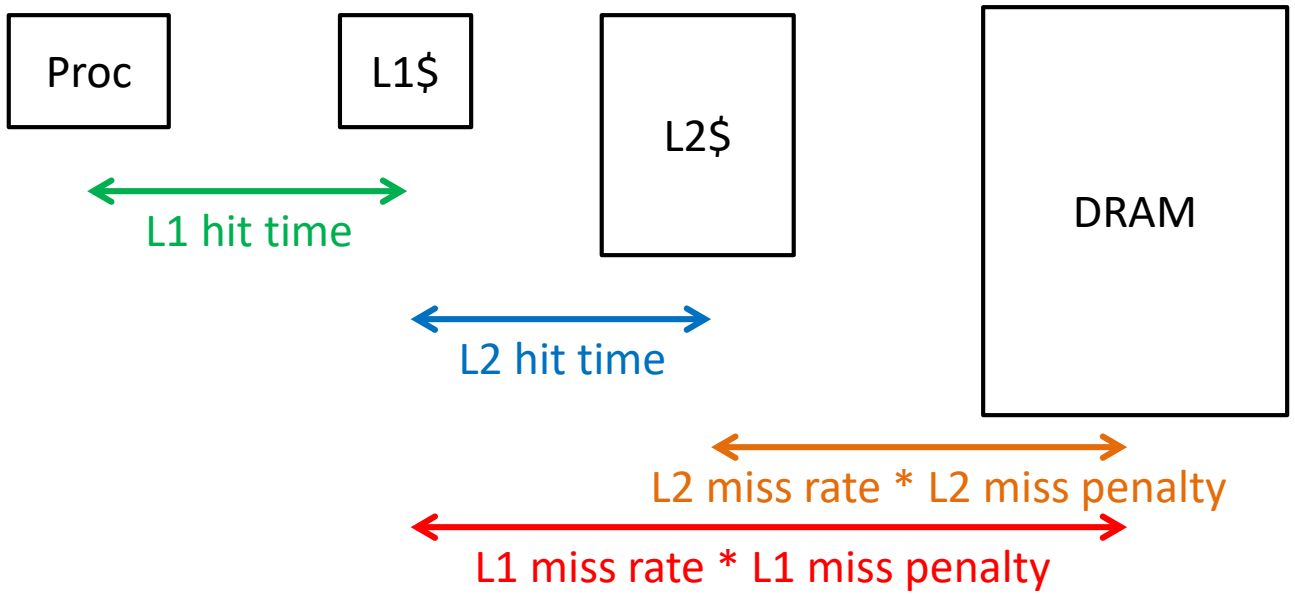
Improving Miss Penalty

- When caches first became popular, Miss Penalty
  ~ 10 processor clock cycles
- Today 2.4 GHz Processor (0.4 ns per clock cycle) and 80 ns to go to DRAM
  ~ 200 processor clock cycles!



Solution: another cache between memory and the processor cache: *Second Level (L2) Cache*

Analyzing Multi-level cache hierarchy



Avg Mem Access Time (AMAT)

= L1 hit time + L1 miss rate * L1 miss penalty

L1 miss penalty = L2 hit time + L2 miss rate * L2 miss penalty

AMAT = L1 hit time + L1 miss rate * (L2 hit time + L2 miss rate * L2 miss penalty)

Ways to reduce miss rate

- Larger cache
  - limited by cost and technology
  - hit time of first level cache < cycle time (bigger caches are slower)
- More places in the cache to put each block of memory – associativity
  - fully-associative
    - any block any line
  - N-way set associated
    - N places for each block
    - direct map: N=1

Typical Scale

- L1
  - size: tens of KB (Core i7: 32+32 per core)
  - hit time: complete in one clock cycle
  - miss rates: 1-5%
- L2
  - size: hundreds of KB to 1 MB (Core i7: 256k per core)
  - hit time: few clock cycles
  - miss rates: 10-20%
- L2 miss rate is fraction of L1 misses that also miss in L2
  - why so high?

Example: with L2 cache

- Assume
  - L1 Hit Time = 1 cycle
  - L1 Miss rate = 5%
  - L2 Hit Time = 5 cycles
  - L2 Miss rate = 15% (% L1 misses that miss in L2)
  - L2 Miss Penalty = 200 cycles
- L1 miss penalty = 5 + 0.15 * 200 = 35
- Avg mem access time = 1 + 0.05 x 35

  = **2.75 cycles**

Example: without L2 cache

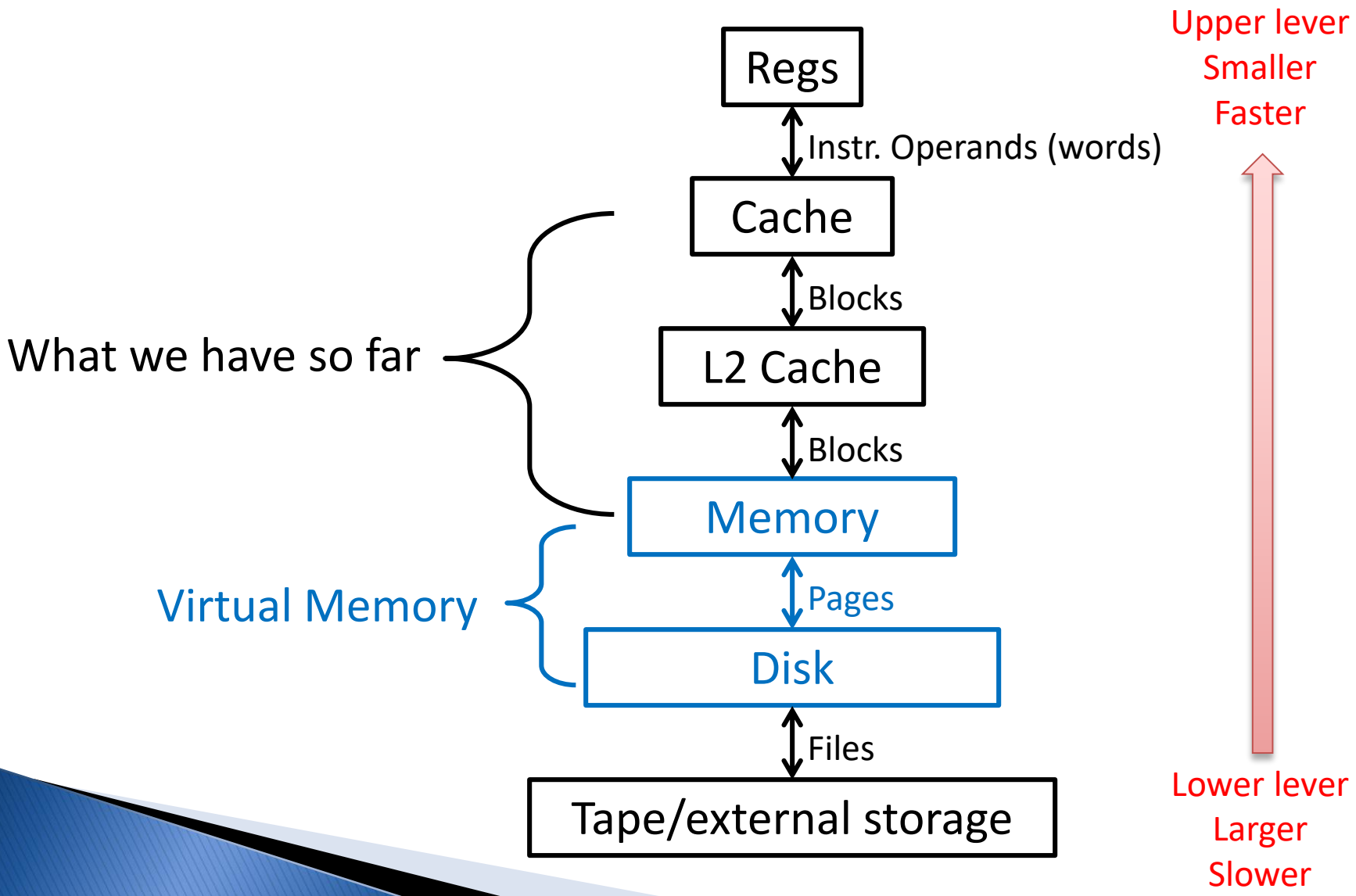
- Assume
  - L1 Hit Time = 1 cycle
  - L1 Miss rate = 5%
  - L1 Miss Penalty = 200 cycles
- Avg mem access time = 1 + 0.05 x 200

  = 11 cycles
- 4x faster with L2 cache! (2.75 vs. 11)

Summary

- We've discussed memory caching in detail. Caching in general shows up over and over in computer systems
  - Filesystem cache, Web page cache, Game databases/tablebases, Software memorization, Others?
- Big idea: if something is expensive but we want to do it repeatedly, do it once and cache the result.
- Cache design choices:
  - Size of cache: speed v. capacity
  - Block size (i.e., cache aspect ratio)
  - Write Policy (Write through v. write back)
  - Associativity choice of N (direct-mapped v. set v. fully associative)
  - Block replacement policy
  - 2nd level cache?
  - 3rd level cache?
- Use performance model to pick between choices, depending on programs, technology, budget, ...

Another View of the Memory Hierarchy



Memory Hierarchy Requirements

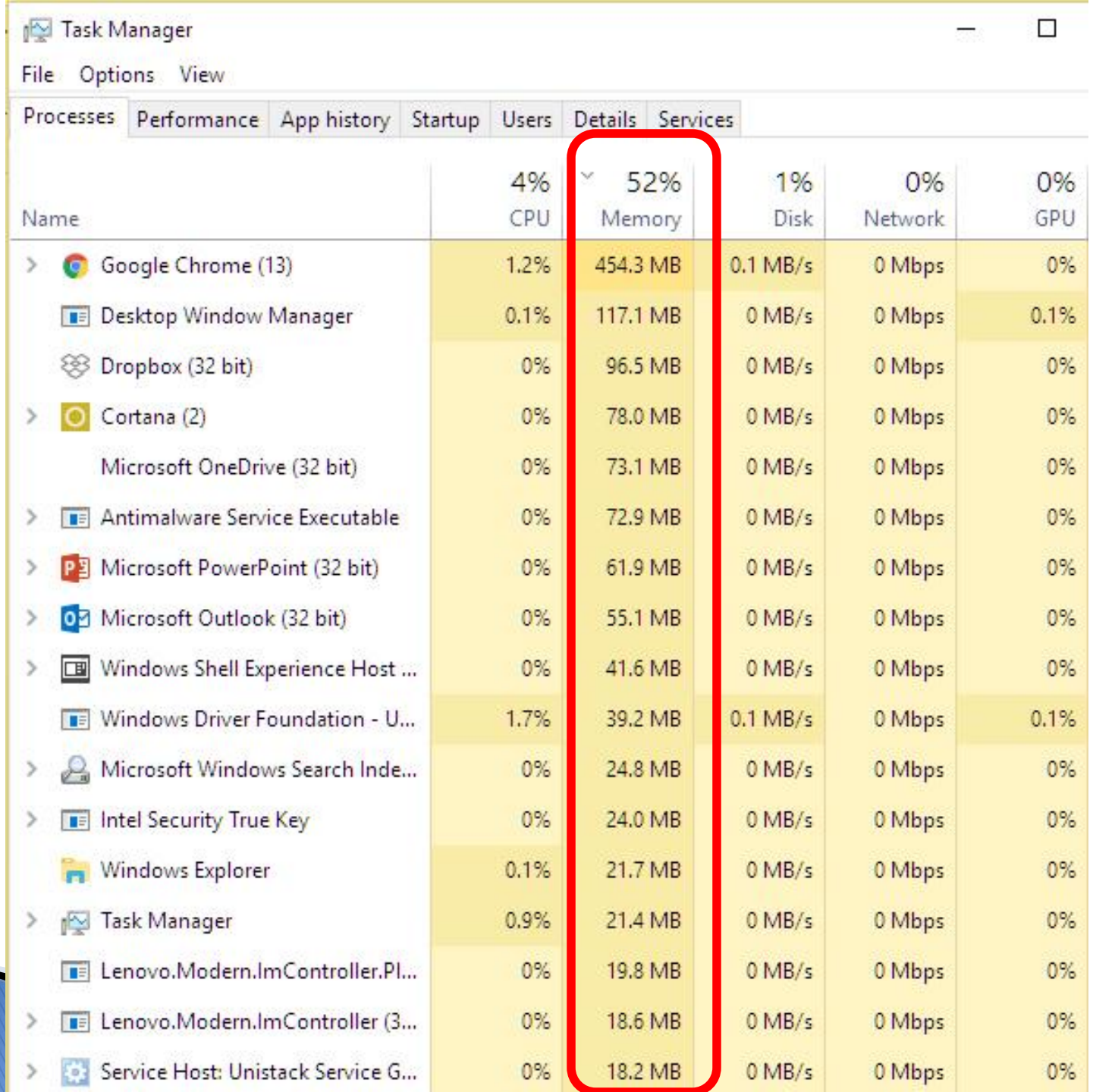
- If Principle of Locality allows caches to offer (close to) speed of cache memory with size of DRAM memory, then recursively why not use at next level to give speed of DRAM memory, size of Disk memory?
- While we're at it, what other things do we need from our memory system?
  - **Do your computers run only ONE program at a time?**

Memory Hierarchy Requirements

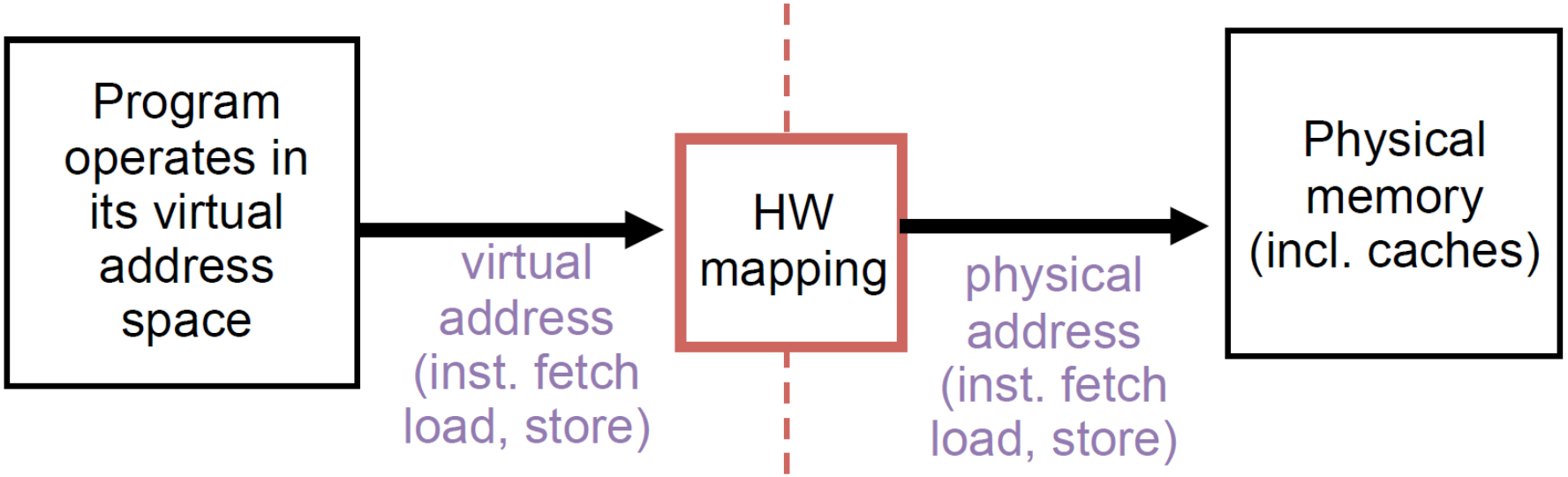
- Allow multiple processes to simultaneously occupy memory and provide protection – don't let one program read/write memory from another
- Address space – give each program the illusion that it has its own private memory
  - Suppose code starts at address 0x40000000. But different processes have different code, both residing at the same address. So each program has a different view of memory.

Virtual Memory

- Next level in the memory hierarchy:
  - Provides program with illusion of a very large main memory
  - Working set of “pages” reside in main memory – others reside on disk.
- Also allows 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 all the memory to itself
- Historically, this concept predates caches


Task Manager
File Options View
Processes Performance App history Startup Users Details Services
Name
4% CPU
52% Memory
1% Disk
0% Network
0% GPU
Google Chrome (13) 1.2% 454.3 MB 0.1 MB/s 0 Mbps 0%
Desktop Window Manager 0.1% 117.1 MB 0 MB/s 0 Mbps 0.1%
Dropbox (32 bit) 0% 96.5 MB 0 MB/s 0 Mbps 0%
Cortana (2) 0% 78.0 MB 0 MB/s 0 Mbps 0%
Microsoft OneDrive (32 bit) 0% 73.1 MB 0 MB/s 0 Mbps 0%
Antimalware Service Executable 0% 72.9 MB 0 MB/s 0 Mbps 0%
Microsoft PowerPoint (32 bit) 0% 61.9 MB 0 MB/s 0 Mbps 0%
Microsoft Outlook (32 bit) 0% 55.1 MB 0 MB/s 0 Mbps 0%
Windows Shell Experience Host ... 0% 41.6 MB 0 MB/s 0 Mbps 0%
Windows Driver Foundation - U... 1.7% 39.2 MB 0.1 MB/s 0 Mbps 0.1%
Microsoft Windows Search Inde... 0% 24.8 MB 0 MB/s 0 Mbps 0%
Intel Security True Key 0% 24.0 MB 0 MB/s 0 Mbps 0%
Windows Explorer 0.1% 21.7 MB 0 MB/s 0 Mbps 0%
Task Manager 0.9% 21.4 MB 0 MB/s 0 Mbps 0%
Lenovo.Modern.ImController.Pl... 0% 19.8 MB 0 MB/s 0 Mbps 0%
Lenovo.Modern.ImController (3... 0% 18.6 MB 0 MB/s 0 Mbps 0%
Service Host: Unistack Service G... 0% 18.2 MB 0 MB/s 0 Mbps 0%

Virtual to Physical Address Translation

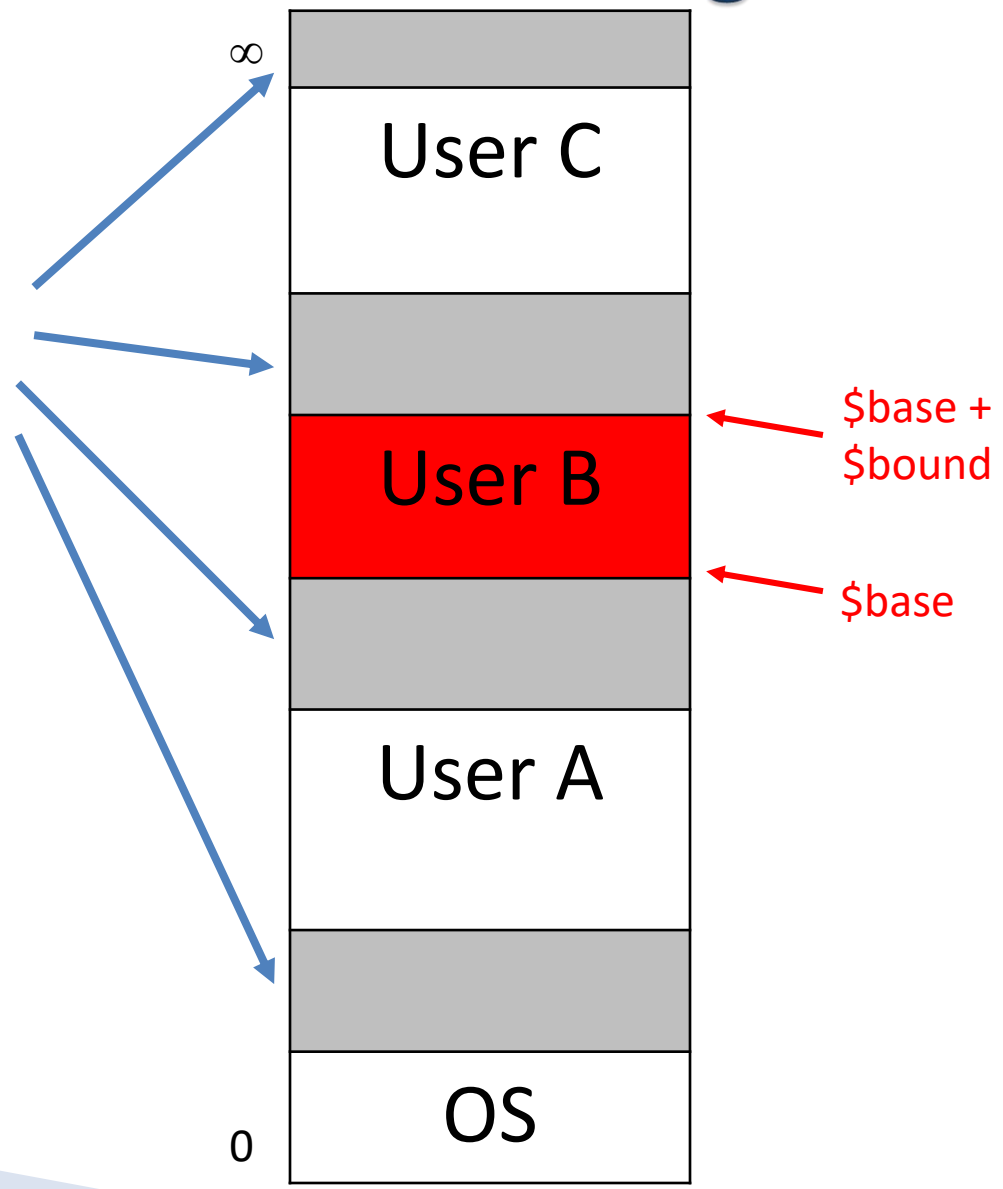


- Each program operates in its own virtual address space; thinking as the only program running
- Each is protected from the other
- OS can decide where each goes in memory
- Hardware gives virtual → physical mapping

HOW?

Simple Example: Base and Bound Reg

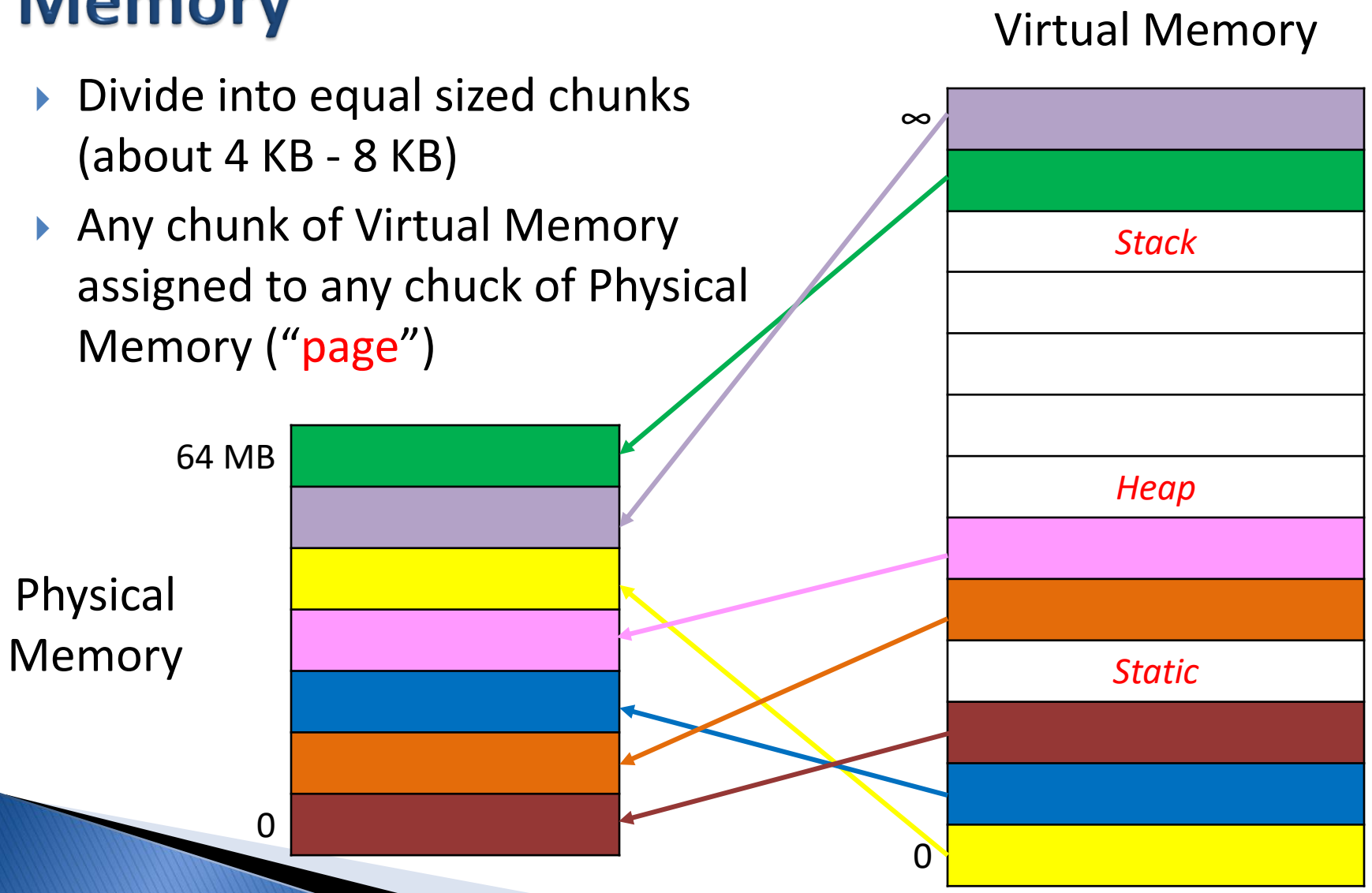
Enough space for User D, but not continuous (fragmentation!)



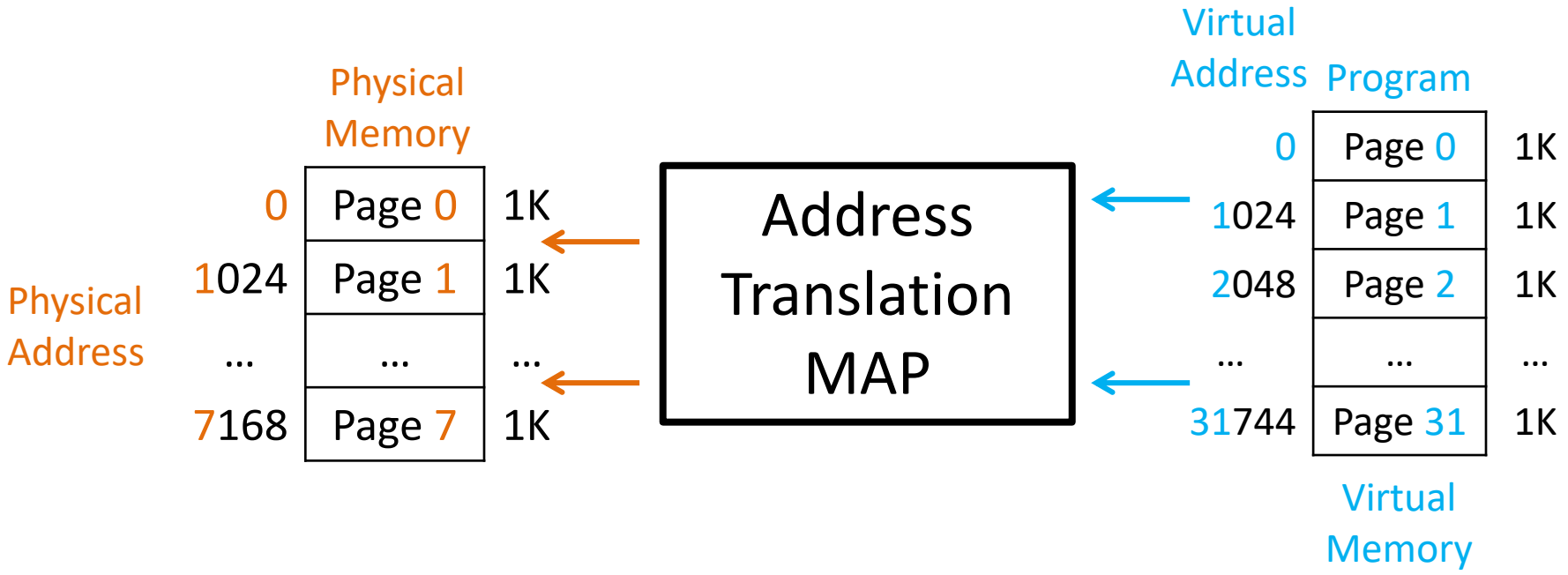
- Want:
  - Discontinuous mapping
  - Process size >> mem
- Addition not enough!
- Use Indirection!

Mapping Virtual Memory to Physical Memory

- Divide into equal sized chunks (about 4 KB - 8 KB)
- Any chunk of Virtual Memory assigned to any chuck of Physical Memory ("page")



Paging Organization (assume 1 KB pages)



- Page is the unit of mapping
  - Also the unit of transfer between disk and physical memory

Virtual Memory Mapping Function

- Cannot have simple function to predict arbitrary mapping
- Use table lookup ("Page Table") for mappings: Page number is index

- Virtual Memory Mapping Function
  - Physical Offset = Virtual Offset (same page size)
  - Physical Page Number
    = PageTable[Virtual Page Number]
  - Physical Page Number (P.P.N.) also called "Page Frame"

Page Table

- A page table is an operating system structure which contains the mapping of **virtual addresses** to **physical locations**
  - There are several different ways, all up to the operating system, to keep this data around (CSE 150)
- Each process running in the operating system has its **own page table**
  - "State" of a process is PC, all registers, plus page table
  - OS changes page tables by changing contents of Page Table Base Register

Address Mapping: Page Table

- Page Table is located in physical memory
- May contain extra entries about the page
  - Dirty bit, ref bit, access rights