The Memory Hierarchy

1. Overview of the memory hierarchy
2. Basics of Cache Memories
3. Virtual Memory
4. Satisfying a memory request
5. Performance and some cache memory optimizations

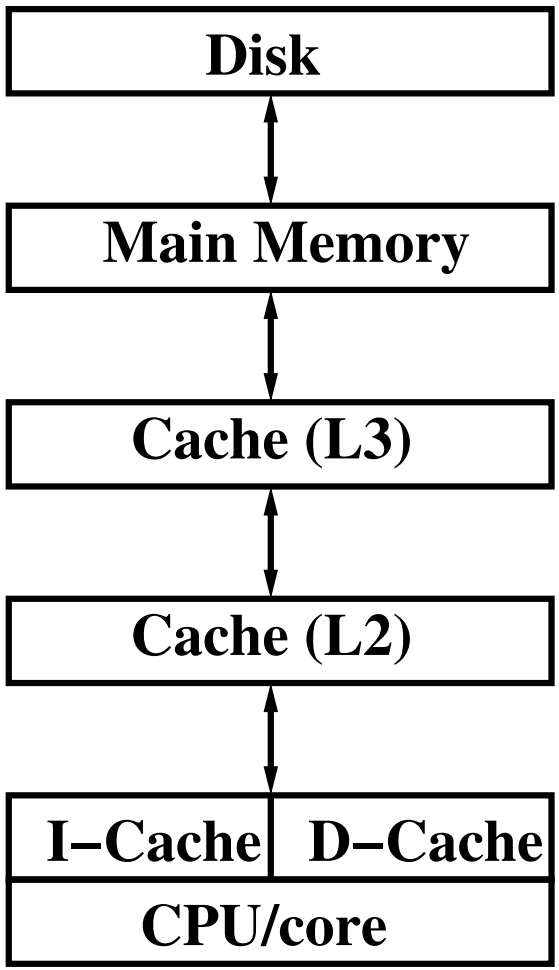
The Memory Hierarchy

UNIVERSITY OF Cincinnati

Basic Problem: How to design a reasonable cost memory system that can deliver data at speeds close to the CPU's consumption rate.

Current Answer: Construct a memory hierarchy with slow (inexpensive, large size) components at the higher levels and with fast (most expensive, smallest) components at the lowest level.

Migration: As information is referenced, migrate the data into and out-of the lowest level memories.



How does this help?

- Programs are well behaved and tend to follow the observed "principles of locality" for programs.

  Principle of temporal locality: states that a referenced data object will likely be referenced again in the near future.

  Principle of spatial locality: states that if data at location x is referenced at time t then it is likely that a nearby data location $x + \Delta x$ will be referenced at a nearby time $t + \Delta t$.

- Also consider the *90/10 rule*: A program executes about 90% of it's instructions from about 10% of its code space.

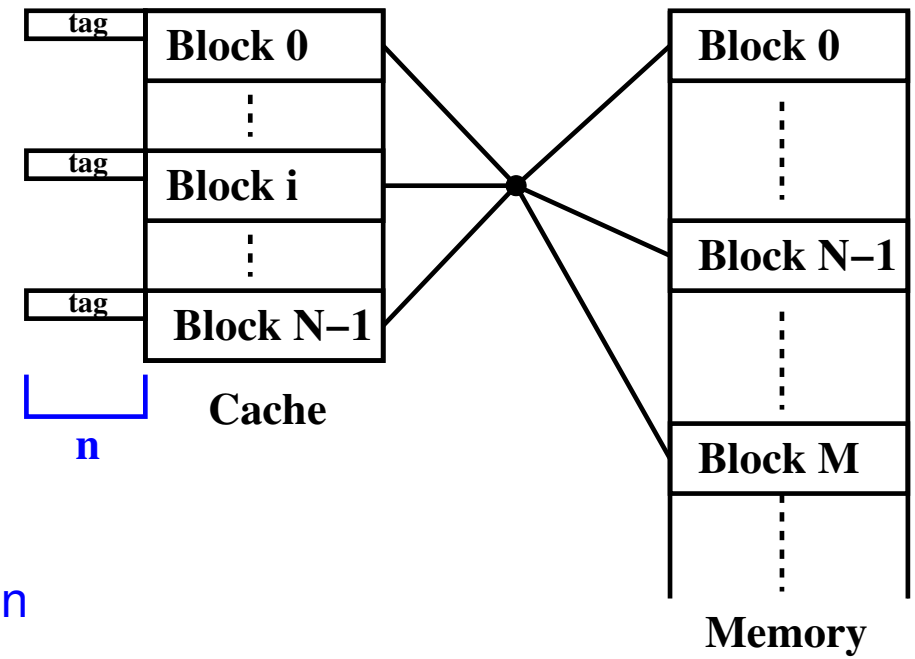
Cache Memories

UNIVERSITY OF Cincinnati

- Fixed-sized blocks mapped to the cache (from the [physical or virtual] memory space)
- Due to speed considerations, all operations implemented in hardware.
- 4 types of mapping policies:
  - fully associative
  - direct mapped
  - set associative
  - sector mapped (not discussed further)

Fully Associative Cache

Any block in the (physical or virtual) memory can be mapped into any block in cache (just like paged virtual memory). Memory addresses are formed as: | n | d | where n is the memory tag and d is the address in the block.



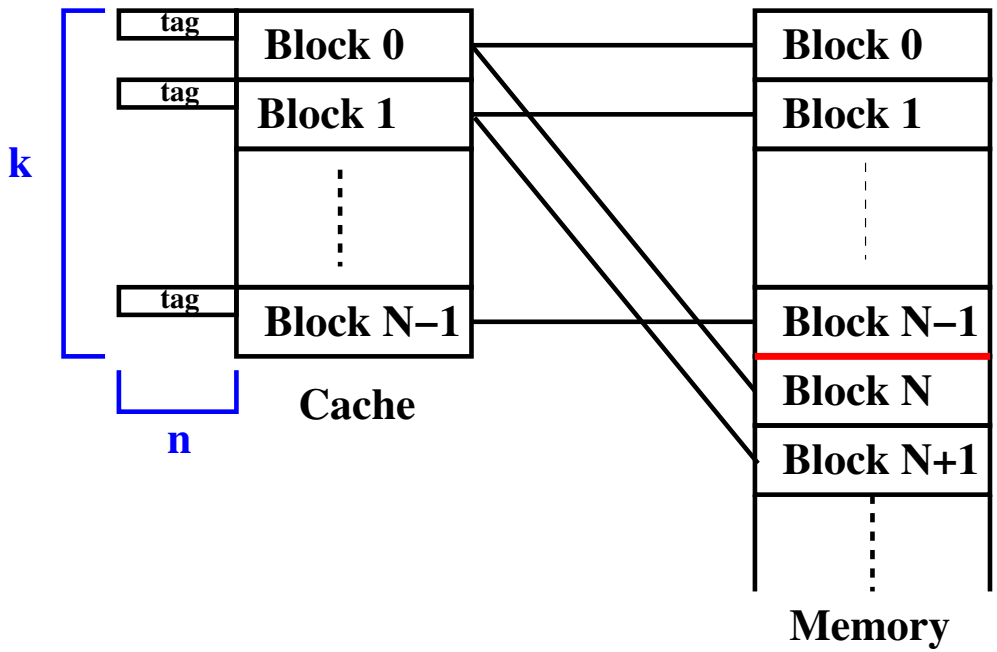
Lookup Algorithm

if $\exists\alpha : 0 \leq \alpha < N \wedge$ cache[α].tag = n
then return cache[α].word[d]
else cache-miss

Direct Mapped Cache

UNIVERSITY OF Cincinnati

If cache is partitioned into N fixed size blocks then each cache block k will contain only (physical or virtual) memory blocks $k + nN$, where $n = 0, 1, ...$ The memory addresses are formed as: | n | k | d | where n is the memory tag, k is the cache block index, and d is the address in the block.



Lookup Algorithm

if cache[k].tag = n
  then return cache[k].word[d]
  else cache-miss

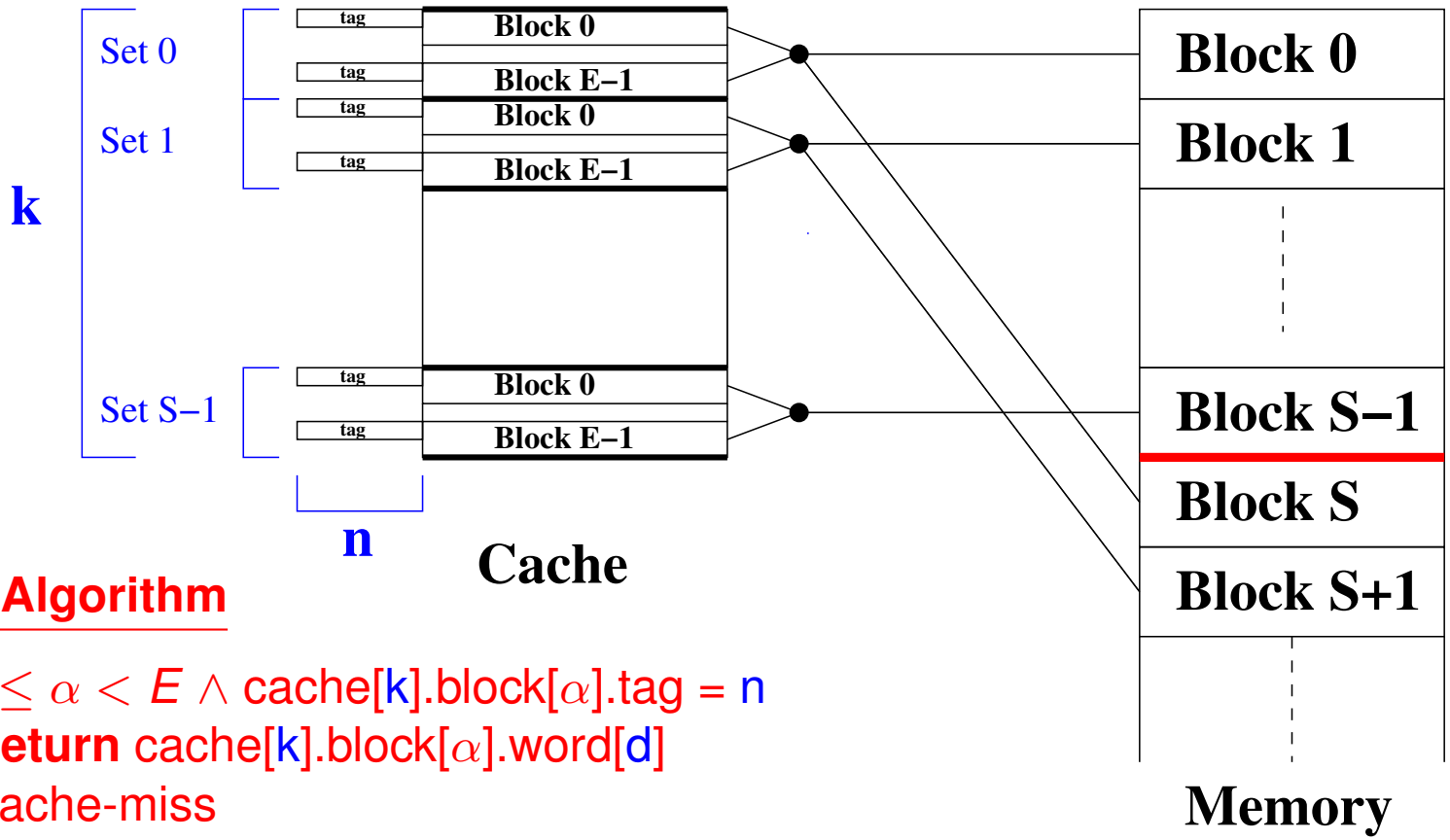
Set Associative Cache

UNIVERSITY OF Cincinnati

Midpoint between directed mapped and fully associative (at the extremes it degenerates to each). Basic idea: divide cache into S sets with E = N/S block frames per set (N total blocks in the cache). Memory address are formed as:

n	k	d

where n is the memory tag, k is the cache set index, and d is the address in the block.

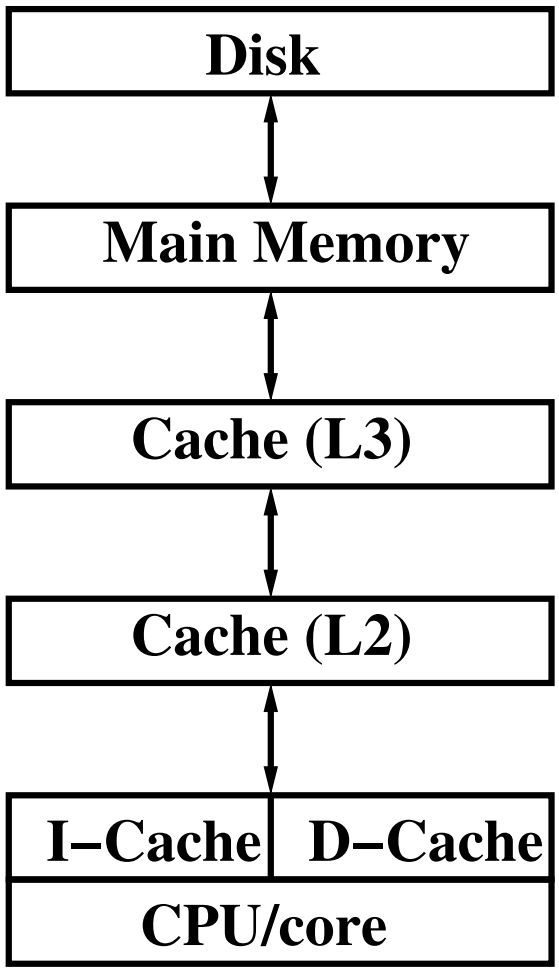


Lookup Algorithm

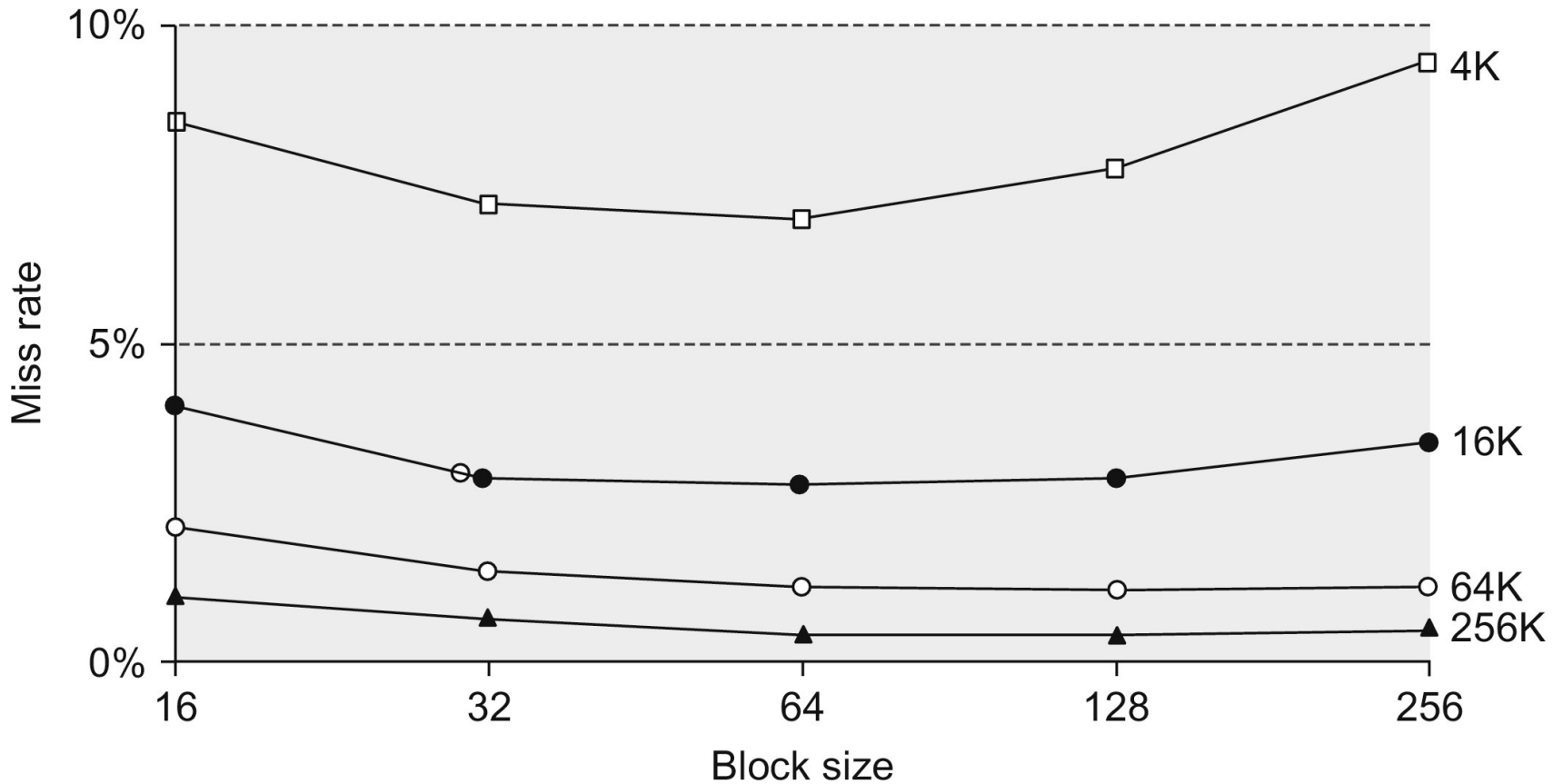
if $\exists \alpha : 0 \leq \alpha < E \wedge$ cache[k].block[α].tag = n
then return cache[k].block[α].word[d]
else cache-miss

Cache Policies

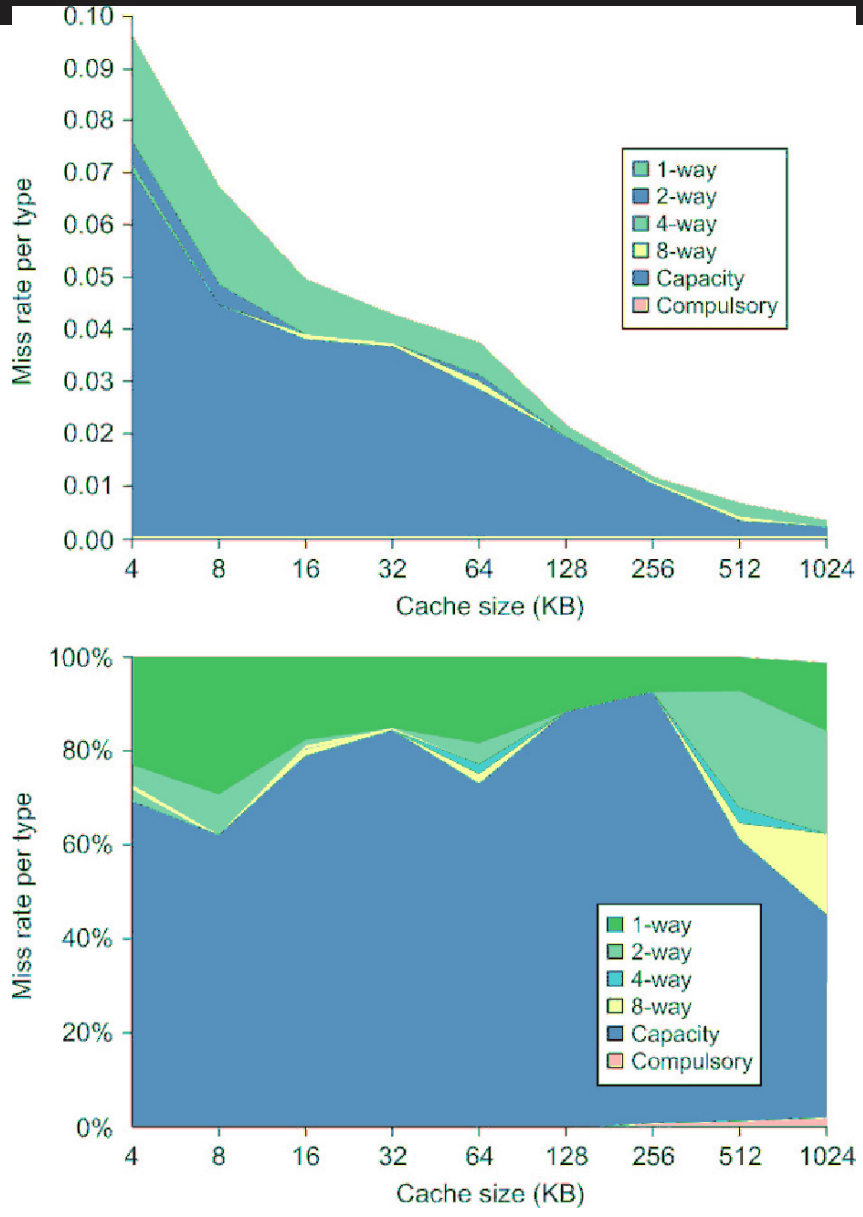
- Read/Write Policies
  - Read through
  - Early restart
  - Critical word first
  - Write through
  - Write back
  - Way prediction
  - Write allocate/no-allocate
- Structure
  - Unified or Split Caches
- Contents across Cache Levels
  - Multilevel inclusion
  - Multilevel exclusion



Cache Miss Rates by Size



Cache Miss Rates by Type



Lecture break; continue next class

Virtual Memory

Addresses used by programs *do not* correspond to actual addresses of the program/data locations in main memory. Instead, there exists a translation mechanism between the CPU and memory that actually associates CPU addresses (virtual) to their actual (physical) in memory

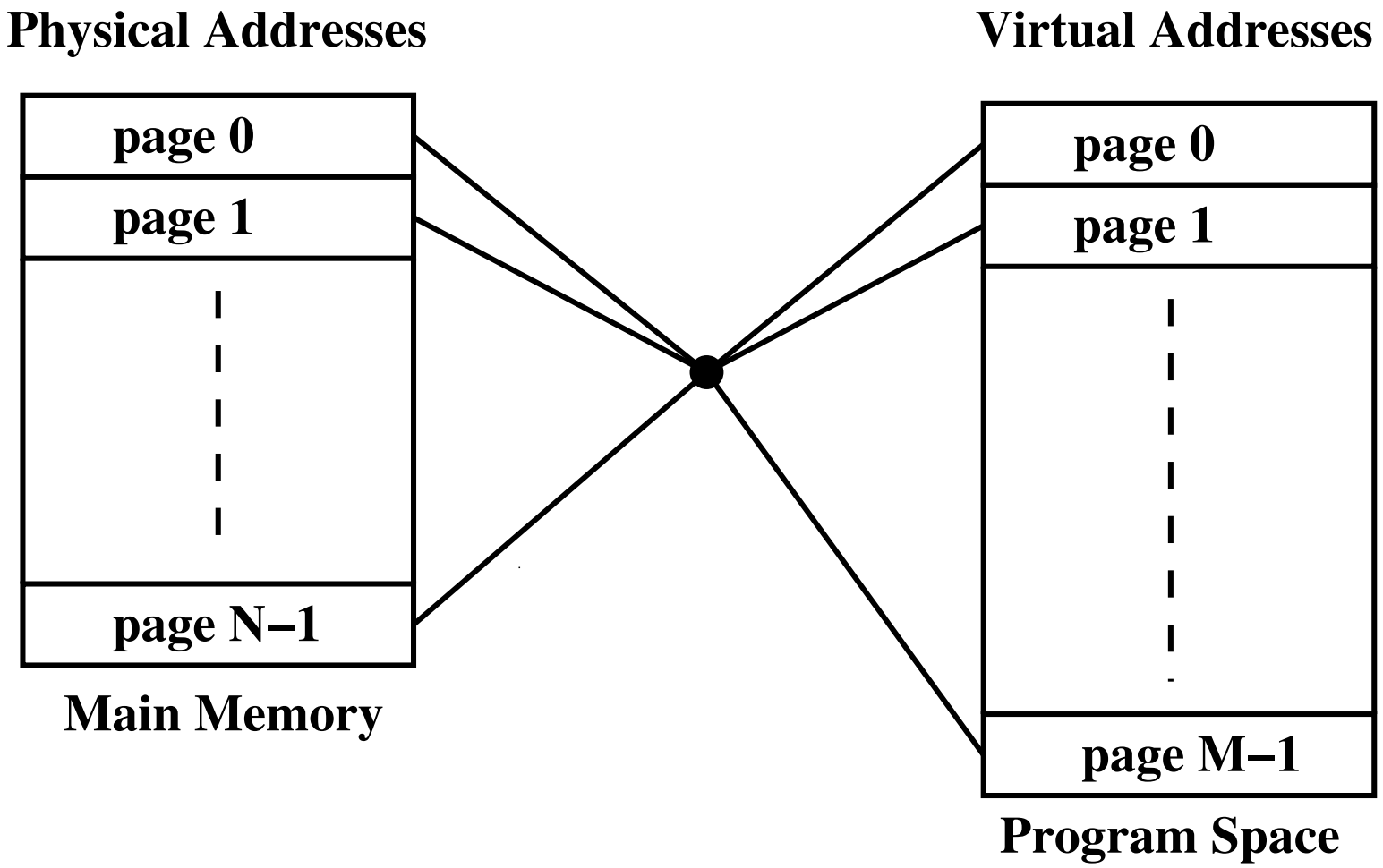
The two most important types:

- Paged virtual memory
- Segmented virtual memory

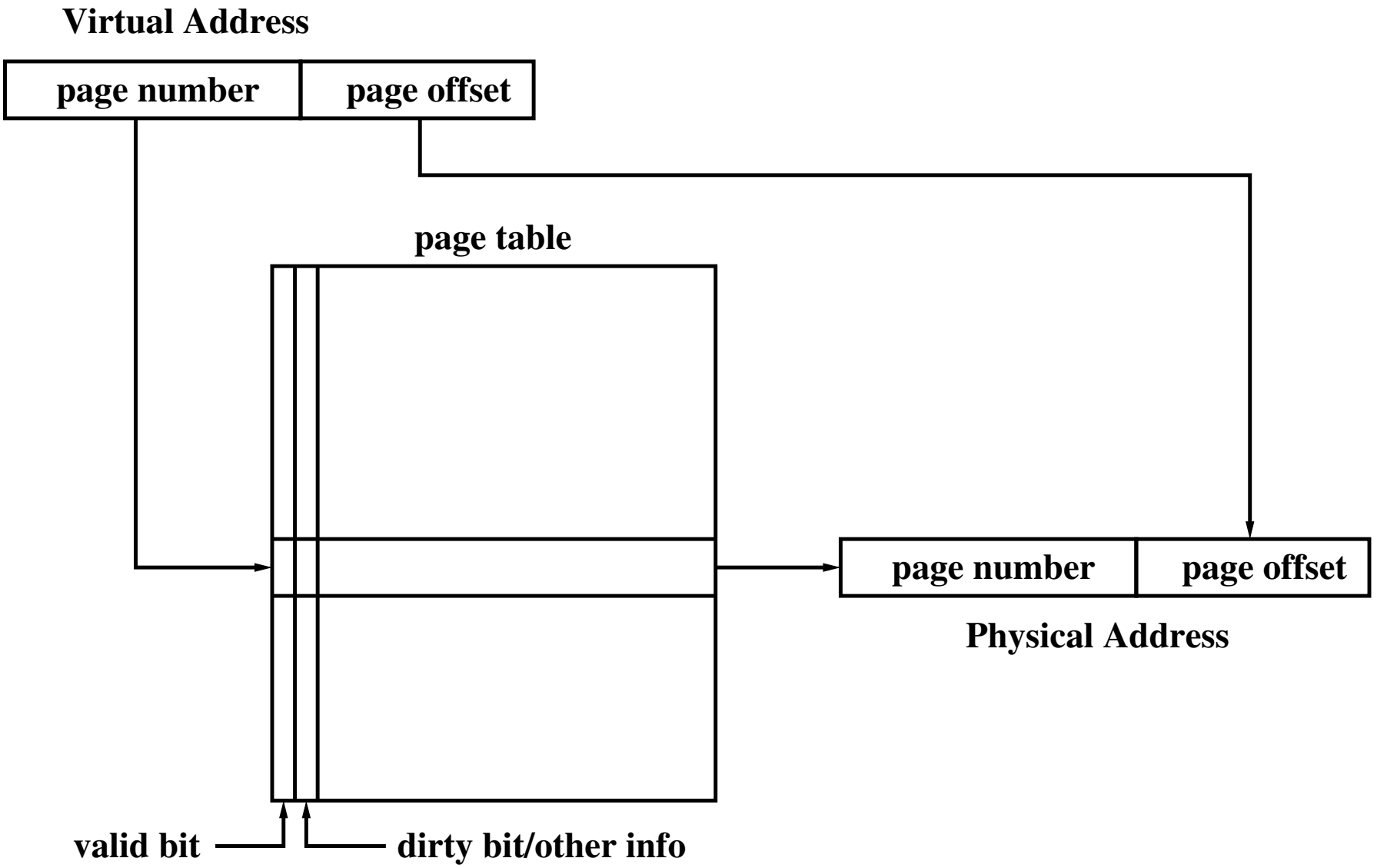
Demand Paged Virtual Memory

- Logically subdivide virtual *and* physical spaces into fixed size units called ***pages***.
- Keep virtual space on disk (swap for working/dirty pages).
- As referenced (on demand), bring pages into main memory (and into the memory hierarchy) and update the page table.
- Virtual pages can be mapped into any available physical page.
- Requires some page replacement algorithm: random, FIFO, LRU, LFU.

Paged Virtual Memory: Structure



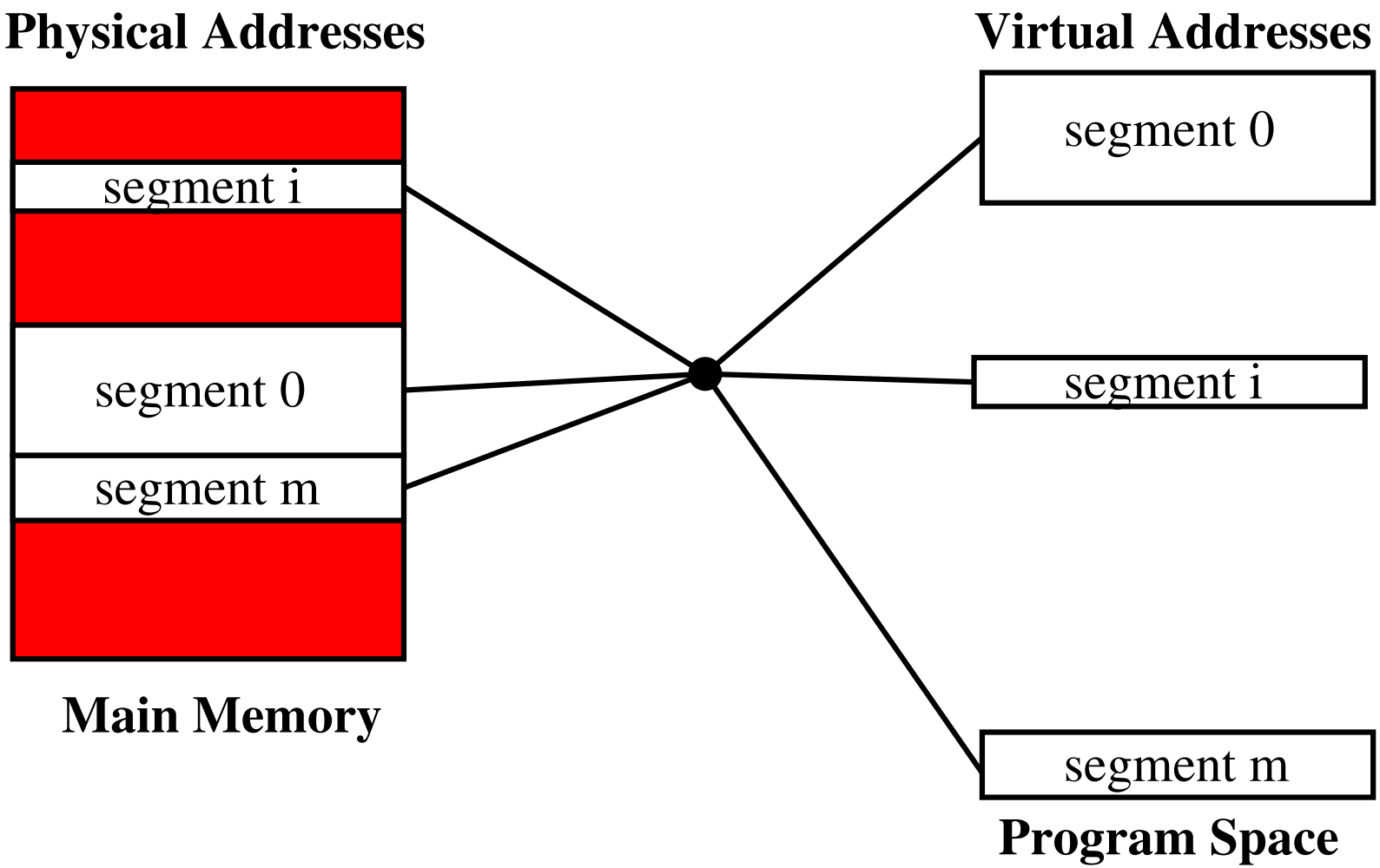
Paged Virtual Memory: Address Translation



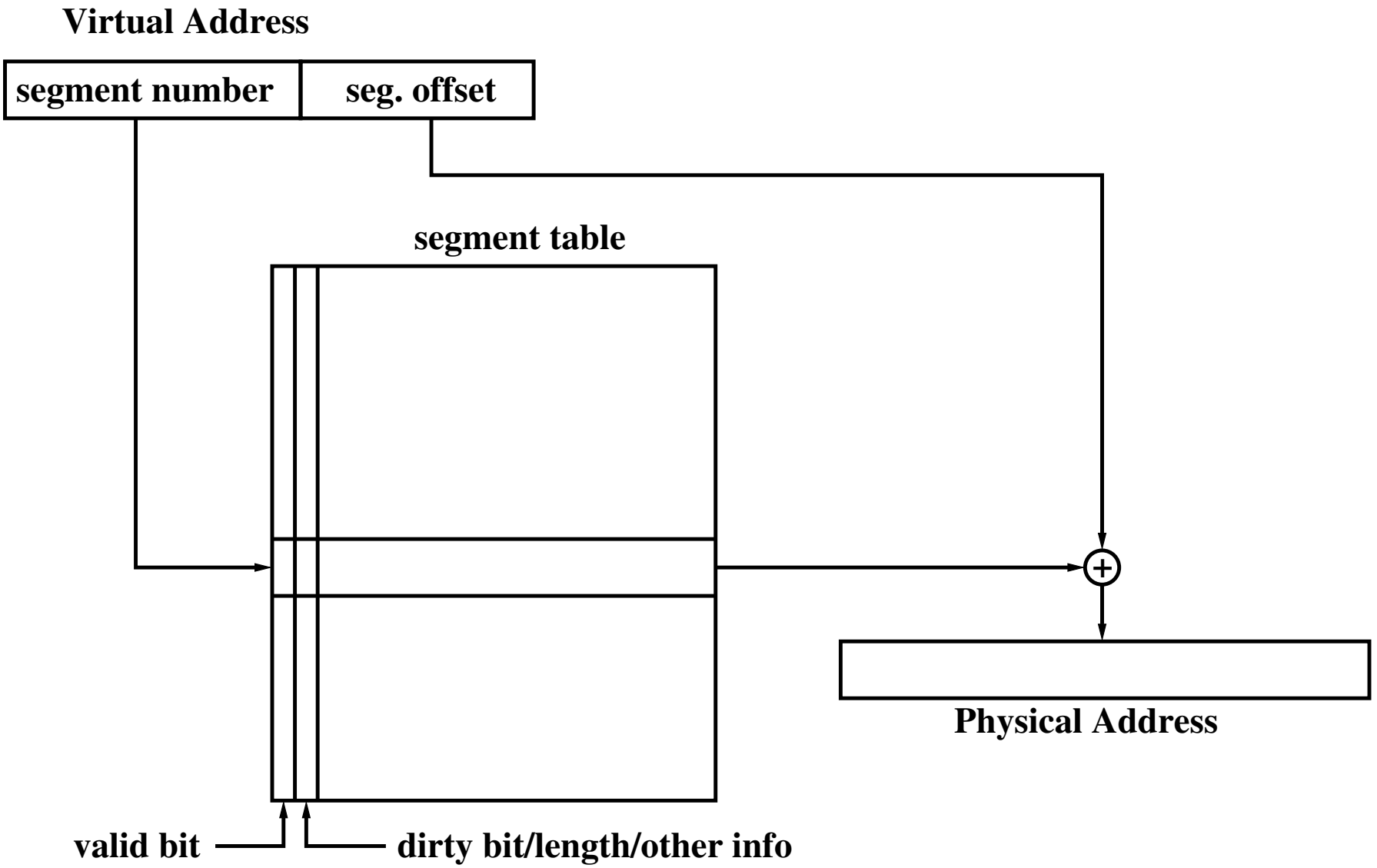
Segmented Virtual Memory

- Identify ***segments*** in virtual space by the logical structure of the program. Using logical structure, we can identify access control to the segment.
- Dynamically map segments into physical space as segments are referenced (on demand). Update segment table accordingly
- Keep virtual space on disk (swap for working/dirty pages).
- Need segment placement algorithm: best fit, worst fit, first fit.
- Needs some segment replacement algorithm.

Segmented Virtual Memory: Structure



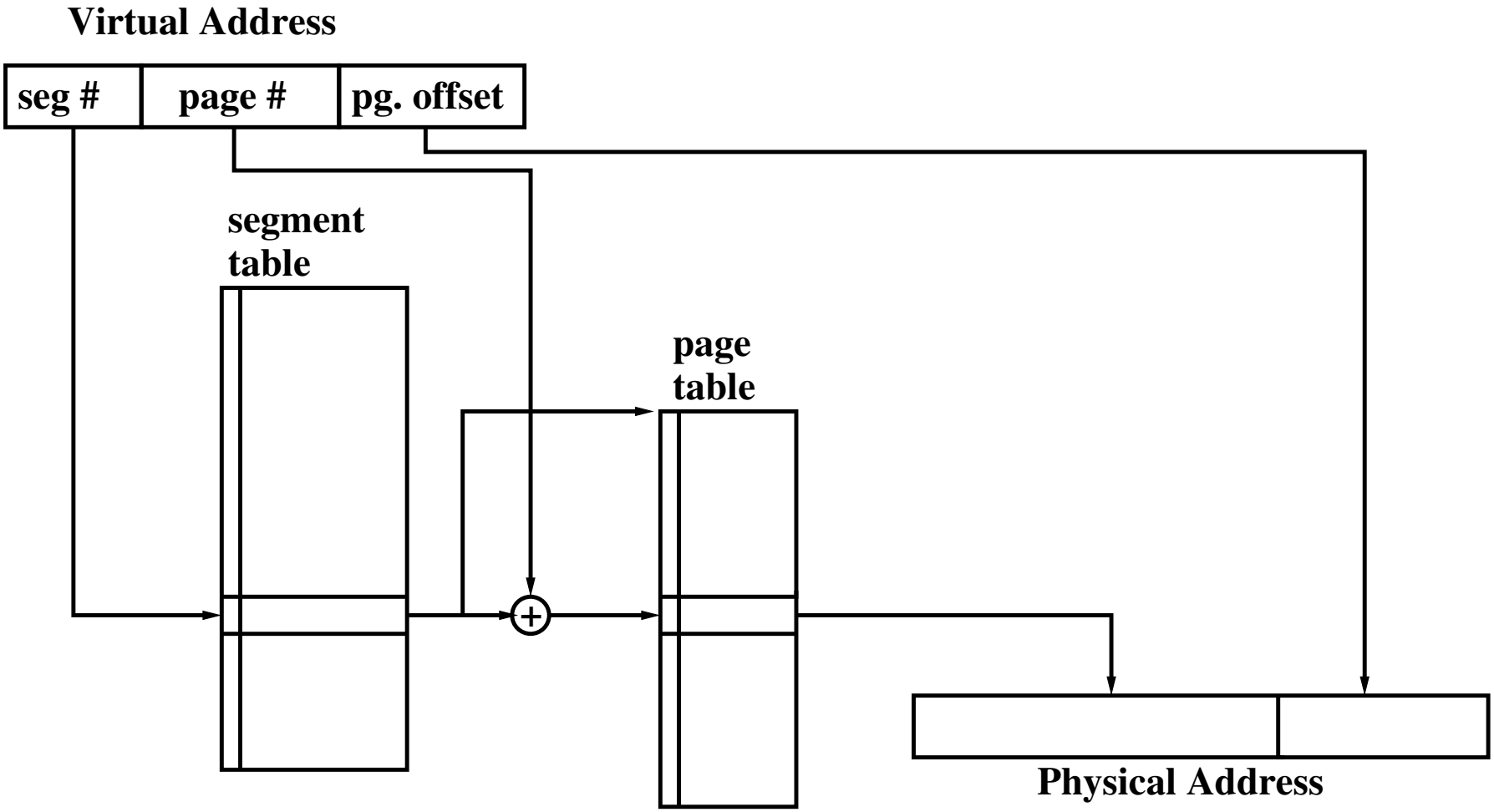
Seg. Virt. Memory: Address Translation



Segmented-Paged Virtual Memory

- Page the segments and map pages into physical memory. Considerably improving and simplifying physical memory use/management.
- Adds an additional translation step.
- Drastically shrinks page table sizes.
- Retains protection capabilities of segments.
- Mechanism used in most contemporary systems (ok, virtualization adds yet another level of indirection....we will discuss this later).

Segmented-Paged: Address Translation



Hardware Implications of Virtual Memory

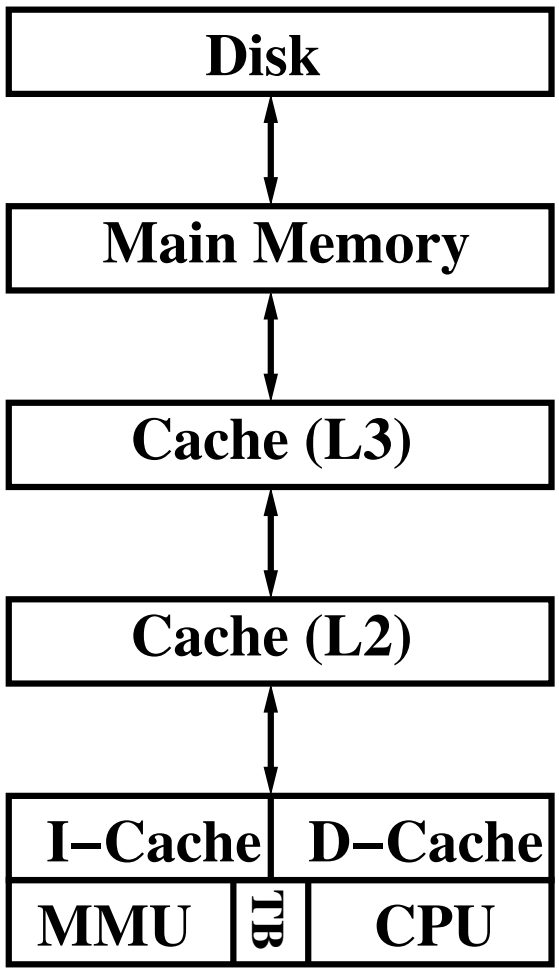
- Virtual memory requires a careful coordination between the hardware and O/S.
- Virtual memory translation will be performed (at least mostly) in hardware.
- When hardware cannot translate a virtual address, it will trap to the O/S.
- Hardware structures for translation will limit the type of virtual memory that an O/S can deploy.

Lecture break; continue next class

The Memory Hierarchy: Who does what?

UNIVERSITY OF Cincinnati

- MMU: Memory Management Unit
- TB: (address) Translation Buffer
- H/W: search to Main Memory
- O/S: handles page faults (invoking DMA operations)
  - if page being replaced is dirty, copy out first (DMA)
  - move new page to main memory (DMA)
- O/S: context swaps on page fault
- DMA: operates concurrently w/ other tasks



Notes

- Memory Management Unit
  - Manages memory subsystems to main memory
  - Translates addresses, searches caches, migrates data (to/from main memory out/in)
- Translation Buffer
  - Small cache to assist virtual $\rightarrow$ physical address translation process
  - generally small (*e.g.,* 64 entries), size does not need to correspond to cache sizes
- Simplifying Assumptions
  - L1, L2, & L3 use physical addresses
  - all caches are inclusive
  - paged virtual memory
  - ignoring details of TB misses

Satisfying A Memory Request

Satisfied in L1 cache:

1. MMU: translate address
2. MMU: search I or D cache as indicated by CPU, success (sometimes simultaneously with translation)
3. MMU: read/write information to/from CPU

Satisfying A Memory Request

Satisfied in L2 cache:

1. MMU: translate address
2. MMU: search I or D cache as indicated by CPU, failure
3. MMU: search L2 cache, success
4. MMU: place block from L2 $\rightarrow$ L1, critical word first?
5. MMU: read/write information to/from CPU

Satisfying A Memory Request

Satisfied in L3 cache:

1. MMU: translate address
2. MMU: search I or D cache as indicated by CPU, failure
3. MMU: search L2 cache, failure
4. MMU: search L3 cache, success
5. MMU: place block from L3 $\rightarrow$ L2, critical word first?
6. MMU: place block from L2 $\rightarrow$ L1, critical word first?
7. MMU: read/write information to/from CPU

Satisfying A Memory Request

Satisfied in main memory:

1. MMU: translate address
2. MMU: search I or D cache as indicated by CPU, failure
3. MMU: search L2 cache, failure
4. MMU: search L3 cache, failure
5. MMU: place block from memory $\rightarrow$ L3, critical word first?
6. MMU: place block from L3 $\rightarrow$ L2, critical word first?
7. MMU: place block from L2 $\rightarrow$ L1, critical word first?
8. MMU: read/write information to/from CPU

Satisfying A Memory Request

Not in main memory:

1. MMU: translate address, failure trap to O/S
2. O/S: page fault, block task for page fault
   - O/S: if page dirty
     - O/S: initiate DMA transfer to copy page to swap
     - O/S: block task for DMA interrupt
     - O/S: invoke task scheduler
     - O/S: on interrupt continue
   - O/S: initiate DMA transfer to copy page to main memory
   - O/S: block task for DMA interrupt
   - O/S: invoke task scheduler
   - O/S: on interrupt:
     - update page table
     - return task to ready to run list

Memory Performance Impact: Naive

CPU exec time

$$= (\textit{CPU clock cycles} + \textit{Memory stall cycles}) \times \textit{Clock cycle time}$$

Memory stall cycles (**doesn't consider hit times $\neq$ 1**)

$$= \textit{Number of Misses} \times \textit{Miss Penalty}$$

$$= IC \times \frac{\textit{Misses}}{\textit{Instruction}} \times \textit{Miss penalty}$$

$$= IC \times \frac{\textit{Memory accesses}}{\textit{Instruction}} \times \textit{Miss Rate} \times \textit{Miss penalty}$$

$$= IC \times \frac{\textit{Memory reads}}{\textit{Instruction}} \times \textit{Read miss Rate} \times \textit{Read miss penalty}$$

$$+ IC \times \frac{\textit{Memory Writes}}{\textit{Instruction}} \times \textit{Write miss Rate} \times \textit{Write miss penalty}$$

Average Memory Access Time

Average memory access time

$$= \textit{Hit time} + \textit{Miss Rate} \times \textit{Miss penalty}$$
$$= \textit{Hit time}_{L1} + \textit{Miss Rate}_{L1} \times (\textit{Hit time}_{L2} + \textit{Miss Rate}_{L2} \times \textit{Miss penalty}_{L2})$$

Accounting for Out-of-order Execution: (discussion probably premature at this time)

$$\frac{\textit{Memory stall Cycles}}{\textit{Instruction}} = \frac{\textit{Misses}}{\textit{Instruction}} \times (\textit{Total miss latency} - \textit{Overlapped miss latency})$$

Cache Misses

- Miss Rate

$$\textit{Local miss rate} = \frac{\#\ \textit{misses in a cache}}{\textit{total}\ \#\ \textit{of memory accesses to this cache}}$$

$$\textit{Global miss rate} = \frac{\#\ \textit{misses in a cache}}{\textit{total}\ \#\ \textit{of memory accesses generated by CPU}}$$

- Categories of Cache Misses
  - Compulsory: cold-start or first-reference misses.
  - Capacity: not all blocks needed fit into cache.
  - Conflict: some needed blocks map to same cache block.

Basic Cache Memory Optimizations

- Larger block sizes to reduce miss rate
- Larger caches to reduce miss rate
- Higher associativity to reduce miss rate
- Multi-level caches to reduce miss penalty
- Giving priority to read misses over writes to reduce miss penalty
- Avoid address translation to reduce hit time