Caching and Demand-Paged Virtual Memory

Chapter 9 OSPP

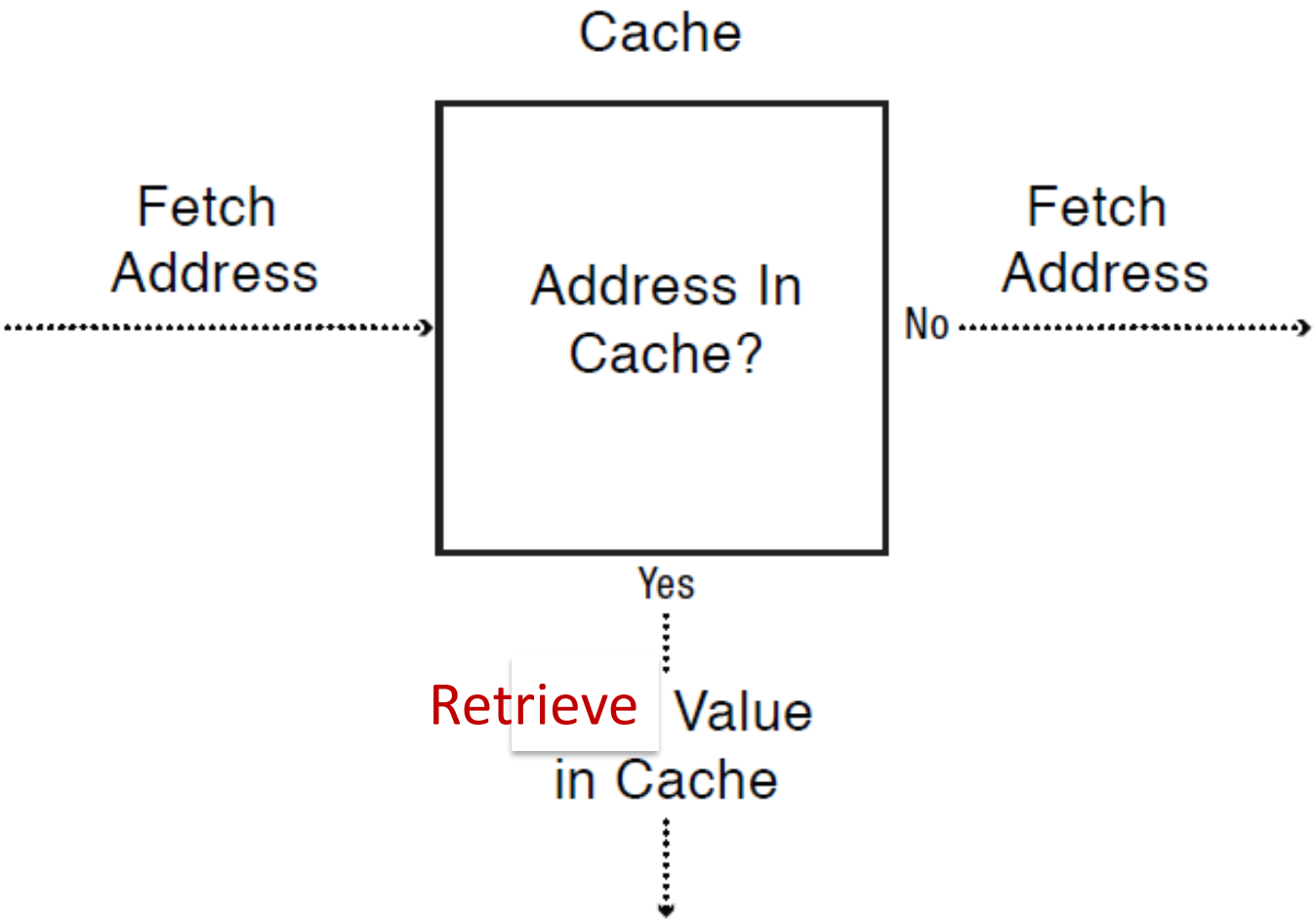
Caching: Address Translation, and Virtual Memory

- Caching
  - Speed up address translation (TLB)
  - Implement virtual memory (memory as a cache for backing store): demand-paging
  - Memory-mapped files

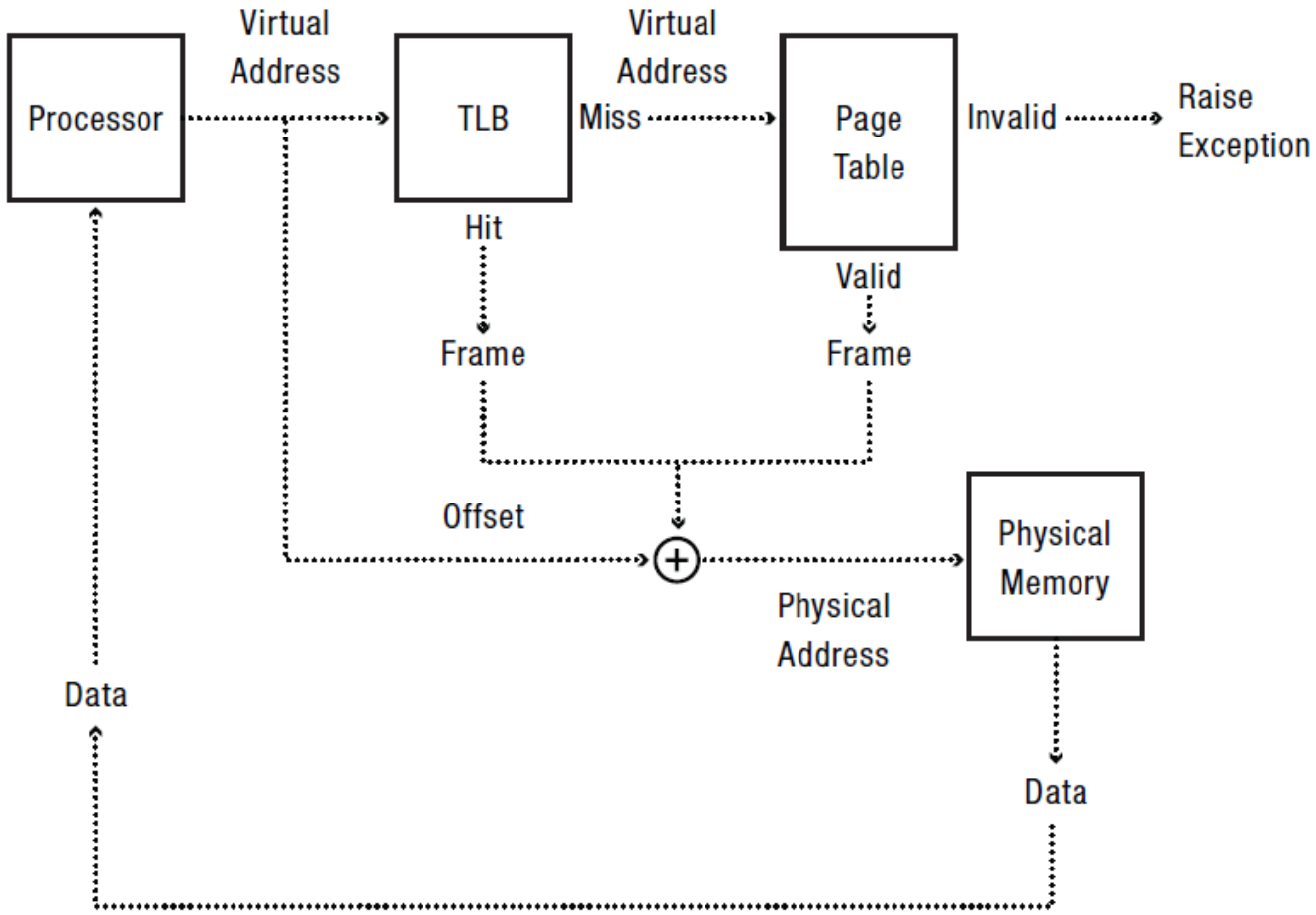
Definitions

- Cache
  - Copy of data that is faster to access than the original
  - Hit: if cache has copy
  - Miss: if cache does not have copy
- Cache block
  - Unit of cache storage (multiple memory locations)
- Temporal locality
  - Programs tend to reference the same memory locations multiple times
  - Example: instructions in a loop
- Spatial locality
  - Programs tend to reference nearby locations
  - Example: data in a loop

Cache Concept (Read)



TLB and Page Table Translation



Memory Hierarchy

Cache	Hit Cost	Size
1st level cache/first level TLB	1 ns	64 KB
2nd level cache/second level TLB	4 ns	256 KB
3rd level cache	12 ns	2 MB
Memory (DRAM)	100 ns	10 GB
Data center memory (DRAM)	100 μs	100 TB
Local non-volatile memory	100 μs	100 GB
Local disk	10 ms	1 TB
Data center disk	10 ms	100 PB
Remote data center disk	200 ms	1 XB

i7 has 8MB as shared 3rd level cache; 2nd level cache is per-core

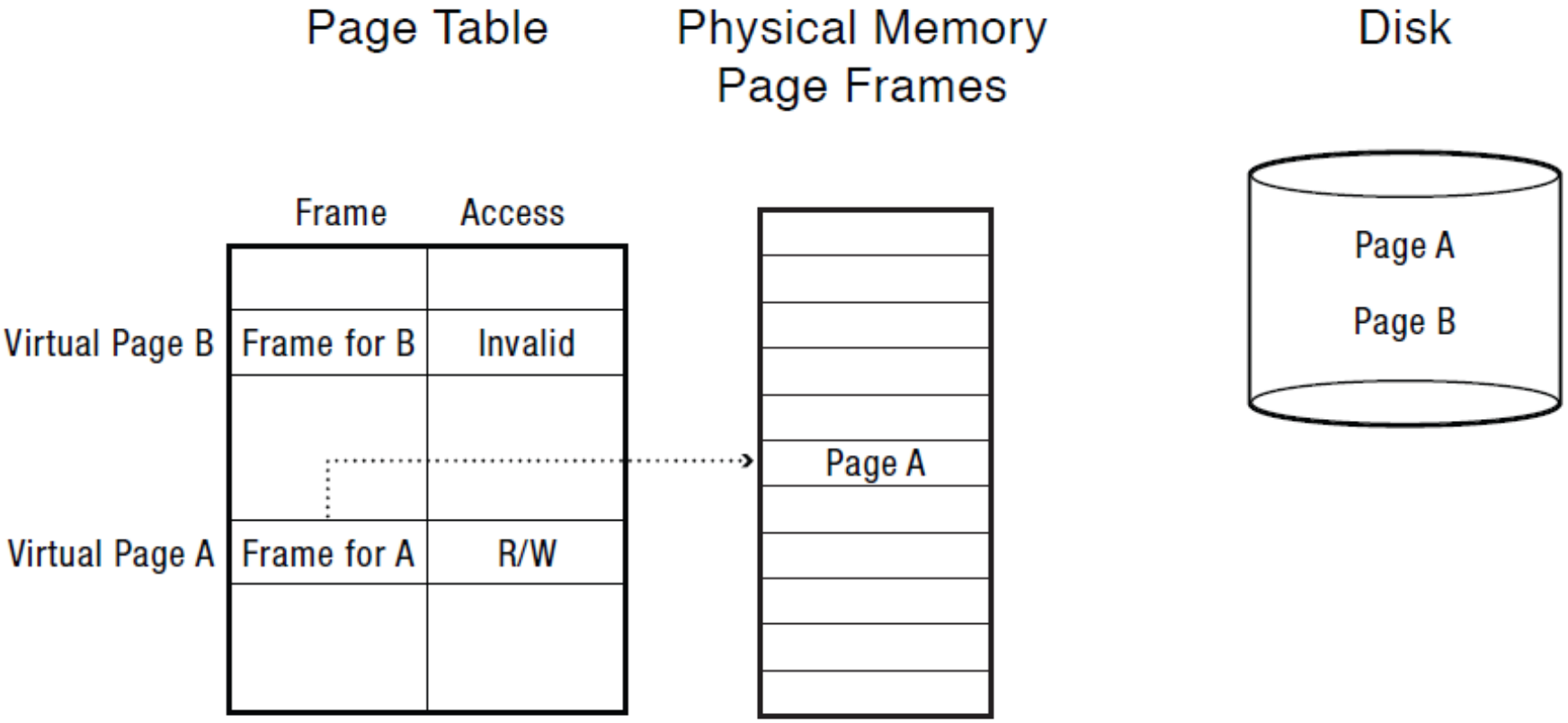
Main Points

- Can we provide the illusion of near infinite memory in limited physical memory?
  - Demand-paged virtual memory
  - Memory-mapped files
- How do we choose which page to replace?
  - FIFO, MIN, LRU, LFU, Clock
- What types of workloads does caching work for, and how well?
  - Spatial/temporal locality vs. Zipf workloads

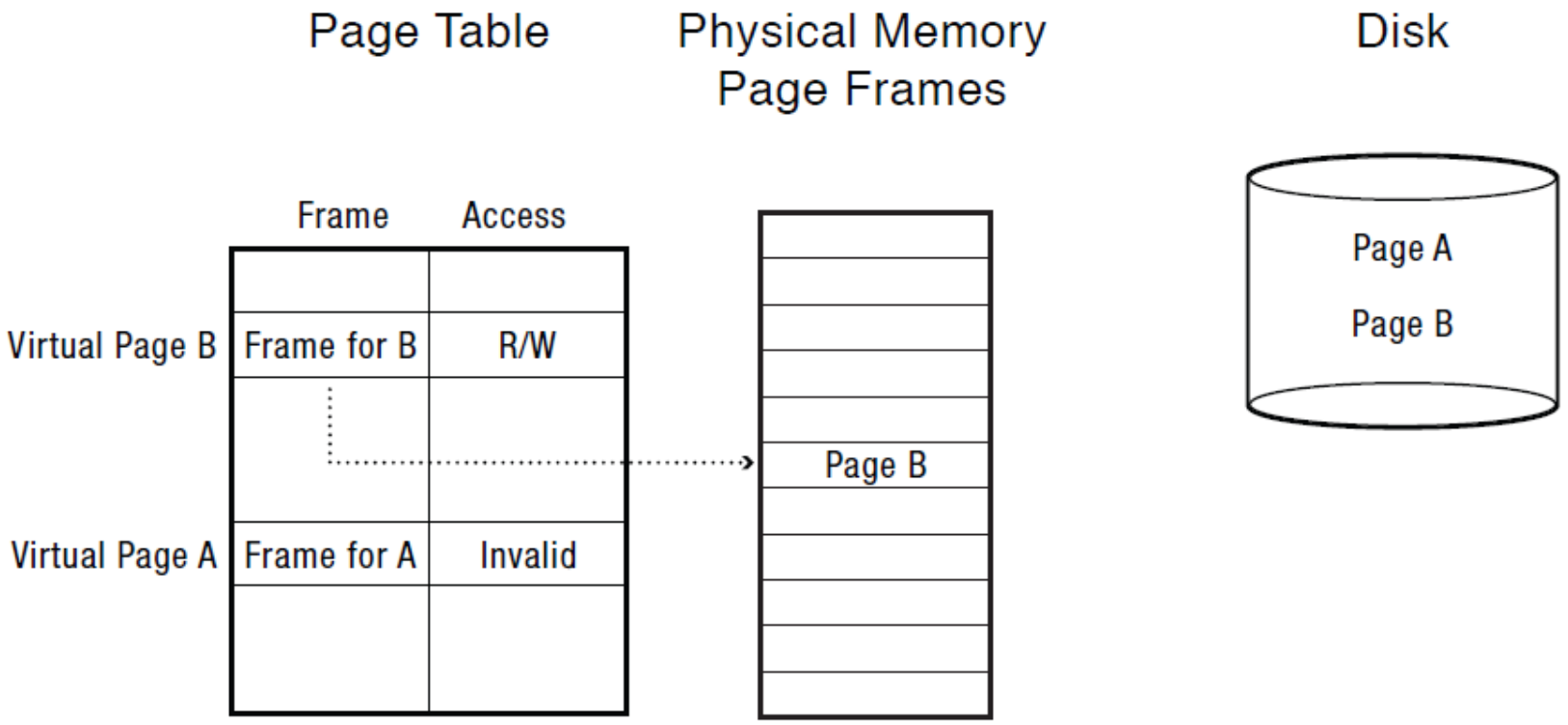
Hardware address translation is a power tool

- Kernel trap on read/write to selected addresses
  - Copy on write
  - Fill on reference
  - Zero on use
  - Demand paged virtual memory
  - Modified bit emulation
  - Memory mapped files
  - Use bit emulation

Demand Paging (Before)



Demand Paging (After)



Caching and Demand-Paged Virtual Memory

Chapter 9 OSPP

Today

- Virtual memory
- Lab #2

Demand Paging – quick walk

One page table per process!

1. TLB miss
2. Page table walk
3. Page fault (page invalid in page table)
4. Trap to kernel
5. Convert virtual address to file + offset
6. Allocate page frame
   - Evict page if needed
7. Initiate disk block read into page frame
8. Disk interrupt when DMA complete
9. Mark page as valid
10. Resume process at faulting instruction
11. TLB miss
12. Page table walk to fetch translation
13. Execute instruction

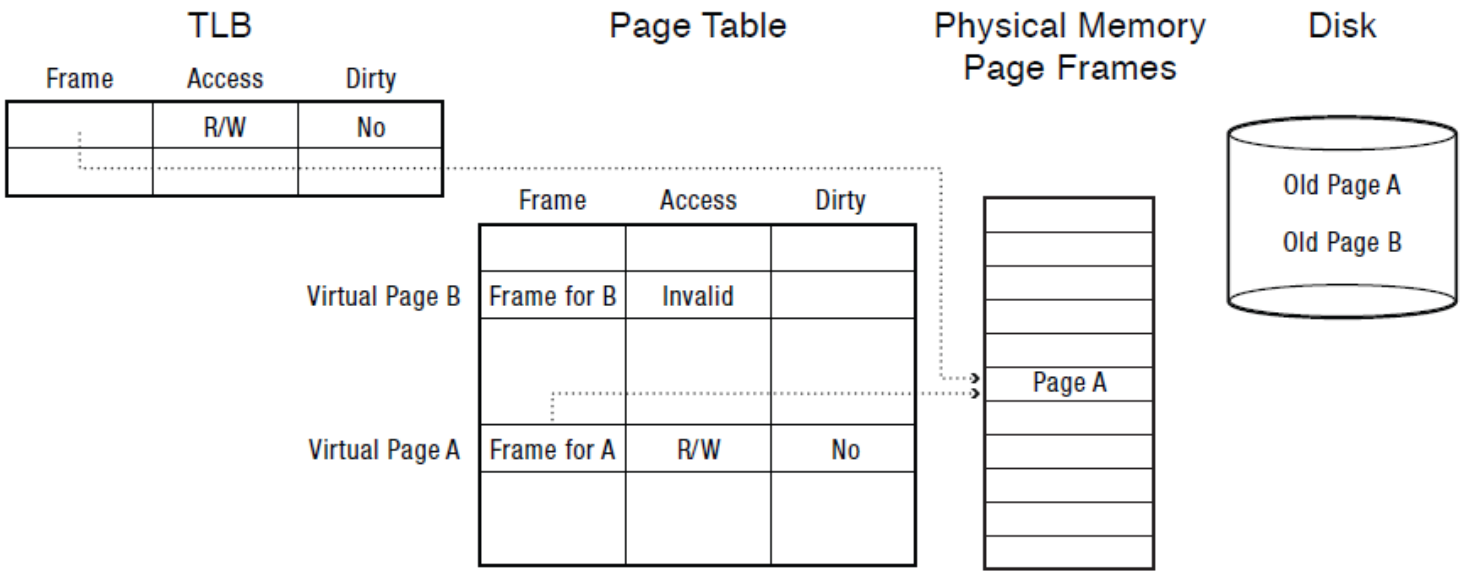
Allocating a Page Frame

- Select old page to evict – which one?
- Find all page table entries that refer to old page
  - If page frame is shared
- Set each page table entry to invalid
- Remove any TLB entries
  - Copies of now invalid page table entry
- Write changes on page back to disk, if necessary

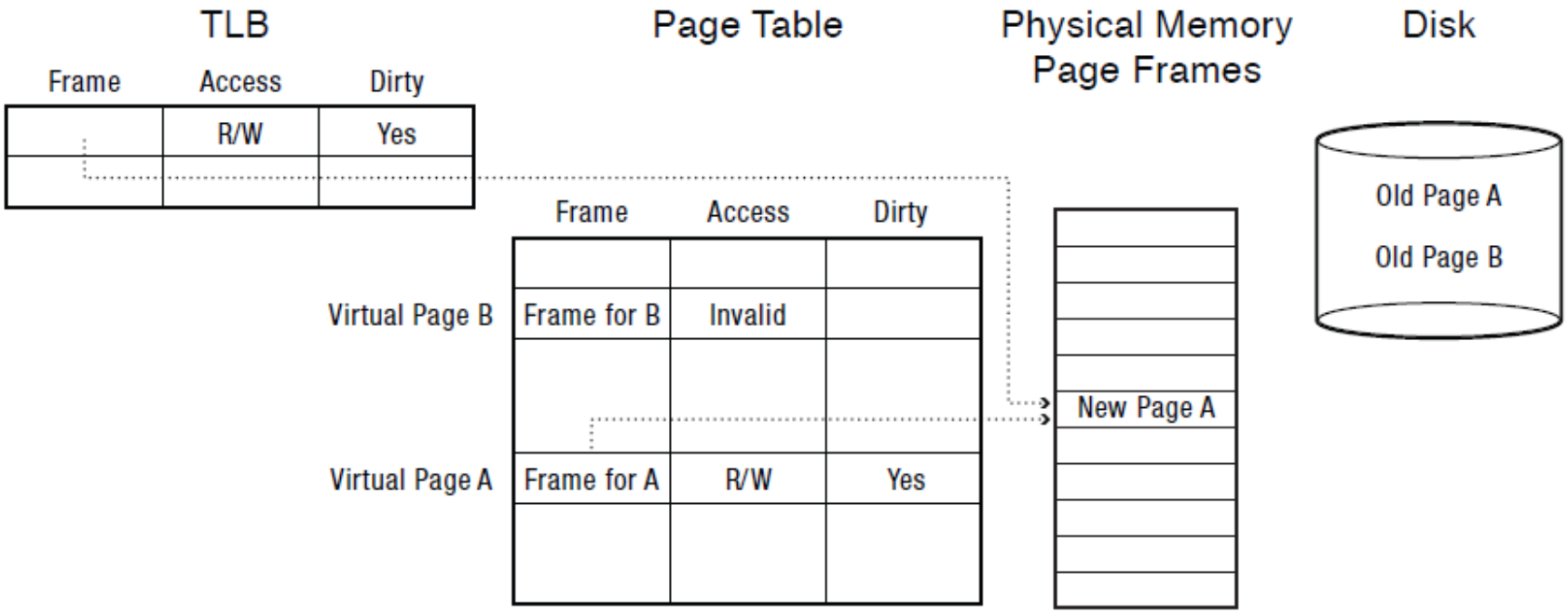
How do we know if page has been modified?

- Every page table entry has some bookkeeping
  - Has page been modified? Dirty bit.
    - Set by hardware on store instruction
    - In both TLB and page table entry
  - Has page been recently used? In use bit.
    - Set by hardware in page table entry
- Bookkeeping bits can be reset by the OS kernel
  - When changes to page are flushed to disk
  - To track whether page is recently used

Keeping Track of Page Modifications (Before)



Keeping Track of Page Modifications (After)



Virtual or Physical Dirty/Use Bits

- Most machines keep dirty/use bits in the page table entry
- Physical page is
  - Modified if *any* page table entry that points to it is modified
  - Recently used if *any* page table entry that points to it is recently used

Tidbit: Emulating a Modified Bit

- Some processor archs. do not keep a modified bit per page
  - Extra bookkeeping and complexity
- Kernel can emulate a modified bit:
  - Set all clean pages as read-only
  - On first write to page, trap into kernel
  - Kernel sets modified bit, marks page as read-write
  - Resume execution
- Kernel needs to keep track of both
  - Current page table permission (e.g., read-only)
  - True page table permission (e.g., writeable)
- Can also emulate a recently used bit

Memory-Mapped Files

- Explicit read/write system calls for files
  - Data copied to user process using system call
  - Application operates on data
  - Data copied back to kernel using system call

- Memory-mapped files
  - Open file as a memory segment
  - Program uses load/store instructions on segment memory, implicitly operating on the file
  - Page fault if portion of file is not yet in memory
  - Kernel brings missing blocks into memory, restarts instruction
  - `mmap` in Linux

Advantages to Memory-mapped Files

- Programming simplicity, esp for large files
  - Operate directly on file, instead of copy in/copy out
- Zero-copy I/O
  - Data brought from disk directly into page frame
- Pipelining
  - Process can start working before all the pages are populated (automatically)
- Interprocess communication
  - Shared memory segment vs. temporary file

From Memory-Mapped Files to Demand-Paged Virtual Memory

- Every process segment backed by a file on disk
  - Code segment -> code portion of executable
  - Data, heap, stack segments -> temp files
  - Shared libraries -> code file and temp data file
  - Memory-mapped files -> memory-mapped files
  - When process ends, delete temp files
- Unified memory management across file buffer and process memory

Memory is a Cache for Disk: Cache Replacement Policy?

- On a cache miss, how do we choose which entry to replace?
  - Assuming the new entry is more likely to be used in the near future
  - In direct mapped caches, not an issue!

- Policy goal: reduce cache misses
  - Improve expected case performance
  - Also: reduce likelihood of very poor performance

A Simple Policy

- Random?
  - Replace a random entry
- FIFO?
  - Replace the entry that has been in the cache the longest time
  - What could go wrong?

FIFO in Action

Reference	A	B	C	D	E	A	B	C	D	E	A	B	C	D	E
1	A				E				D				C		
2		B				A				E				D	
3			C				B				A				E
4				D				C				B			

Worst case for FIFO is if program strides through memory that is larger than the cache

MIN

- MIN
  - Replace the cache entry that will not be used for the longest time into the future
  - Optimality proof based on exchange: if evict an entry used sooner, that will trigger an earlier cache miss
  - Can we know the future?
  - Maybe: compiler might be able to help.

LRU, LFU

- Least Recently Used (LRU)
  - Replace the cache entry that has not been used for the longest time in the past
  - Approximation of MIN
  - Past predicts the future: code?
- Least Frequently Used (LFU)
  - Replace the cache entry used the least often (in the recent past)
  - Important to focus on recent past: why?
  - Mountain hut URL just visited vs. Katy Perry a few minutes ago

LRU

Reference	A	B	A	C	B	D	A	D	E	D	A	E	B	A	C
1	A		+				+				+			+	
2		B			+								+		
3				C	←	←	←	←	E			+			
4						D		+		+					C

FIFO

1	A		+				+	←	←	←	E				
2		B			+						A			+	
3				C								+	B		
4						D		+		+					C

MIN

1	A		+				+				+			+	
2		B			+								+		C
3				C	←	←	←	←	E			+			
4						D		+		+					

Belady's Anomaly

FIFO (3 slots)												
Reference	A	B	C	D	A	B	E	A	B	C	D	E
1	A			D			E					+
2		B			A			+		C		
3			C			B			+		D	
FIFO (4 slots)												
1	A				+		E				D	
2		B				+		A				E
3			C						B			
4				D						C		

More memory does worse! LRU does not suffer from this.

Caching and Demand-Paged Virtual Memory

Chapter 9 OSPP

Today

- HW #3 out
- See on-line schedule for date changes
  - Exam #1 is 11/9; Lab #2 extended (small changes posted)
- VM continued
- Multi-level

- Thursday: won't cover all of memory hog paper

Lab #2

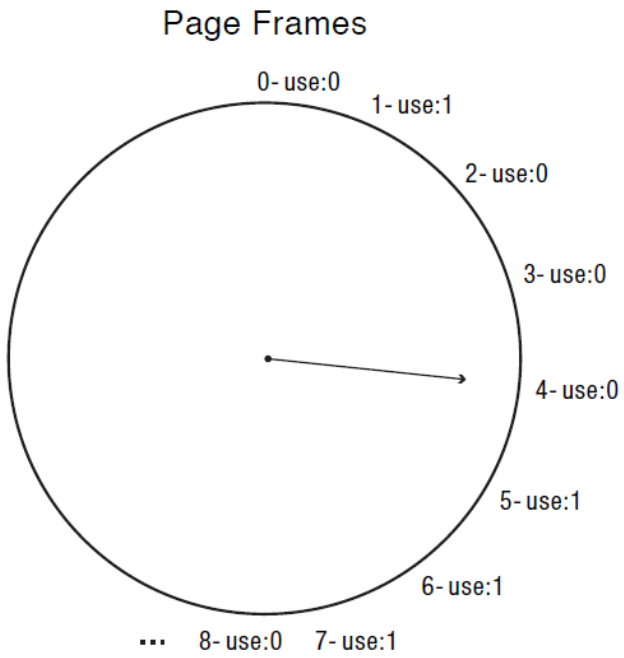
- Briefly

True LRU

- Hard to do in practice: why?
  - Hardware just has a single reference bit
  - Even if could store a clock value in the PT, would have to somehow order the entries and/or go through all of memory to determine LRU

Clock Algorithm: Estimating LRU

- Periodically, sweep through all/some pages
- If page is unused, reclaim (no chance)
- If page is used, mark as unused
- remember clock hand for next time



Nth Chance: Not Recently Used

- Instead of one bit per page, keep an integer
  - notInUseSince: number of sweeps since last use

- Periodically sweep through all page frames

```
if (page is used) {
    notInUseSince = 0;
} else if (notInUseSince < N) {
    notInUseSince++;
} else {
     reclaim page;
}
```

Paging Daemon

- Periodically run some version of clock/Nth chance: background
- Goal to keep # of free frames > %
- Clean (write-back) and free frames as needed

Recap

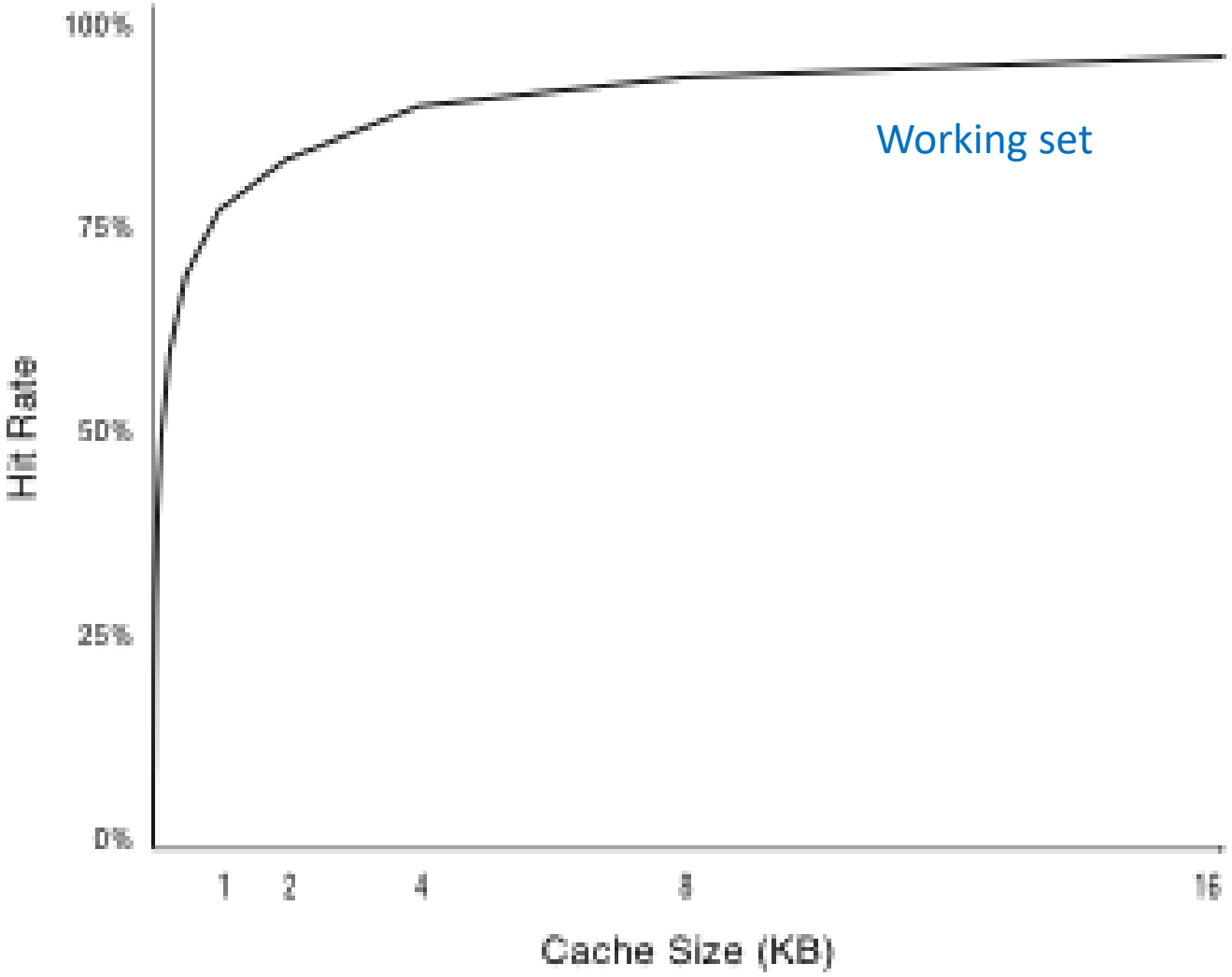
- MIN is optimal
  - replace the page or cache entry that will be used farthest into the future
- LRU is an approximation of MIN
  - For programs that exhibit spatial and temporal locality
- Clock/Nth Chance is an approximation of LRU
  - Bin pages into sets of “not recently used”

Working Set Model

- Working Set (WS): set of memory locations that need to be cached for reasonable cache hit rate
  - top: RES(ident) field (~ WS)
  - Driven by locality
  - Programs get whatever they need (to a point)
  - Pages accessed in last τ time or *k* accesses
  - Uses some version of clock (conceptually): min-max WS

- Thrashing: when cache (i.e. memory) is too small
  - Σ of WS_i > Memory for all i running processes

Cache Working Set

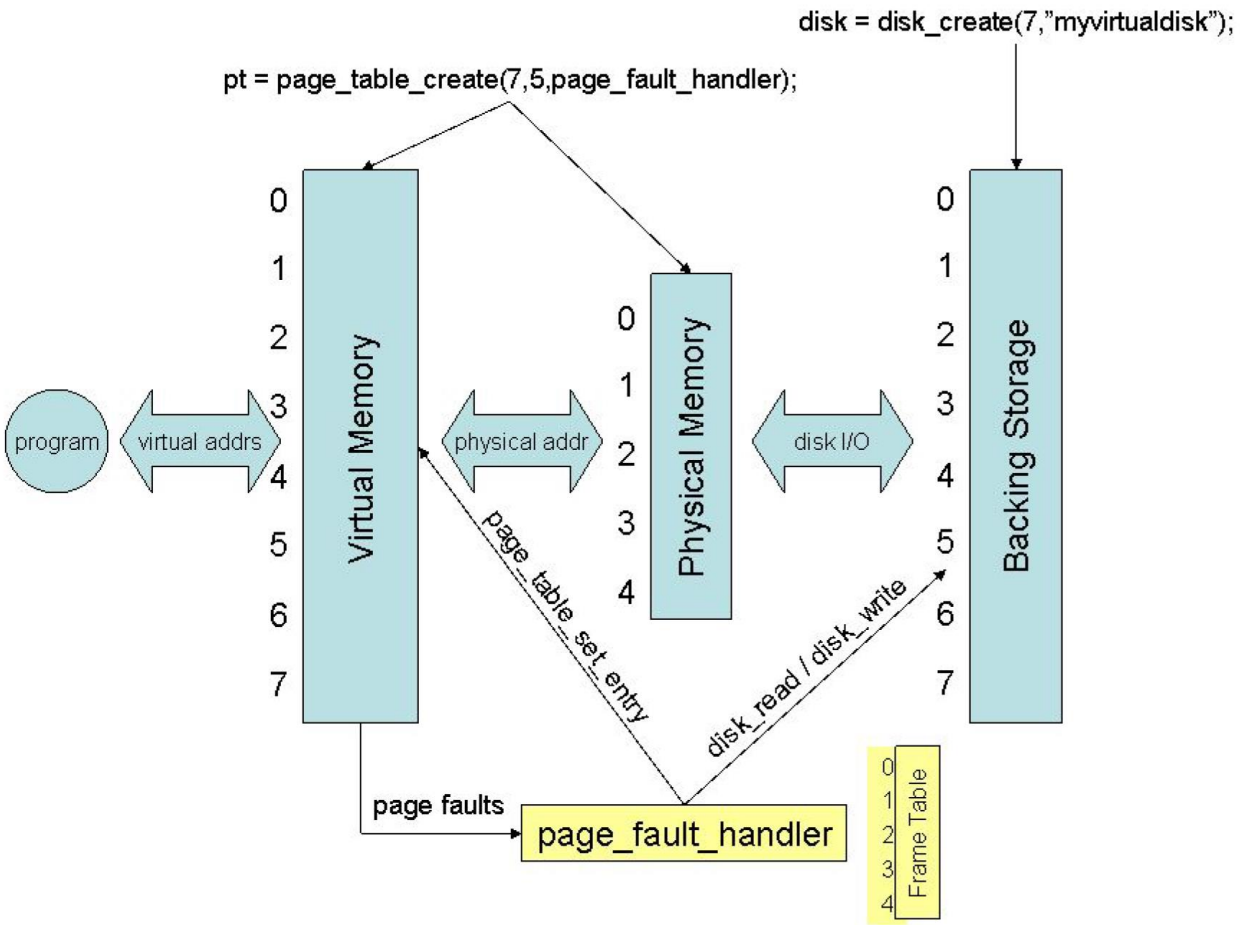


Memory Hogs

- How many pages to give each process?
- Ideally their working set
- But a hog or rogue can steal pages
  - For global page stealing, thrashing can cascade
- Solution: self-page
  - Problem?
  - Local solutions (e.g. multiple queues) are suboptimal

Lab #2: VM Page Replacement

Problem: you cannot modify the page table! Emulate.



Cannot change the hardware

Can change the OS (main.c)

Lab #2

- We provide page table and backing store; just use it.
- Study the code (i.e. the “mechanism”) figure it out!
  - See: two memory-mapped files for VAS and PMEM and a disk store file
- Memory-mapped I/O
  - VAS is mapped to a “file” on backing store
- Emulating “states” by changing protections on memory-mapped VAS (`mprotect`)
- Catch `SIGSEGV` in your user-level “OS” and take action

Hardware

- Emulate bits using memory protection fields
  - PROT_READ: read (accessed)
  - PROT_WRITE: written (dirty)
  - PROT_NONE: no rights (valid)

Next Week

- Wrap up memory next week
  - Caching, Address translation and a Paper to read

- Have a great weekend!