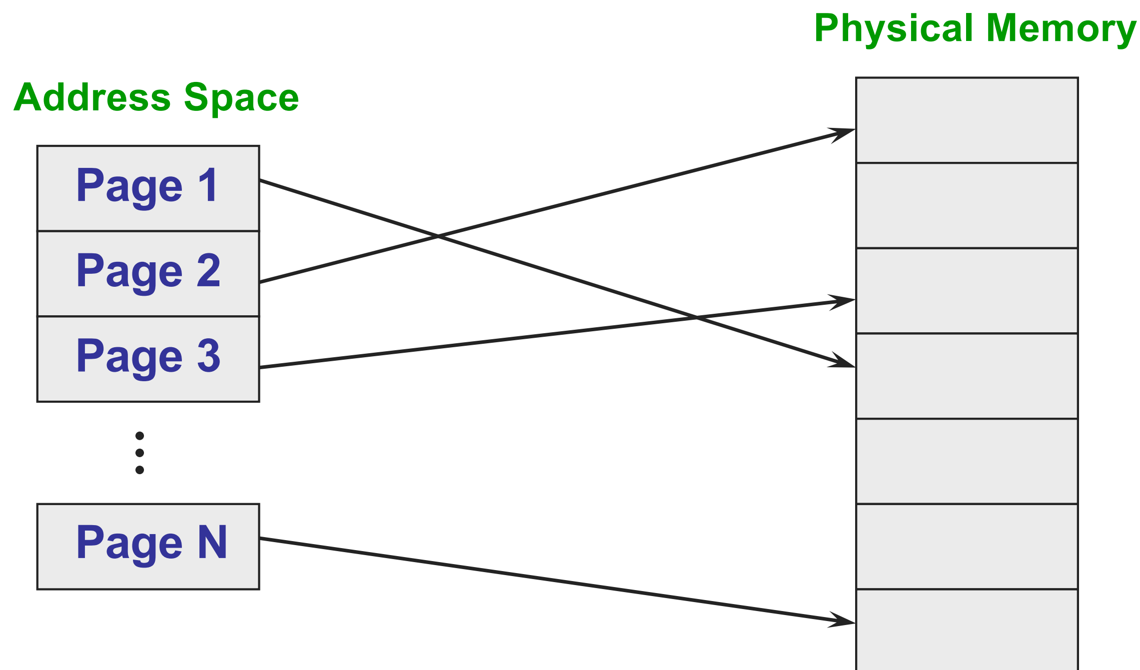
EECS 482
Introduction to Operating Systems

Winter 2018

Harsha V. Madhyastha

Recap: Paging

- Both address spaces and physical memory broken up into fixed size pages

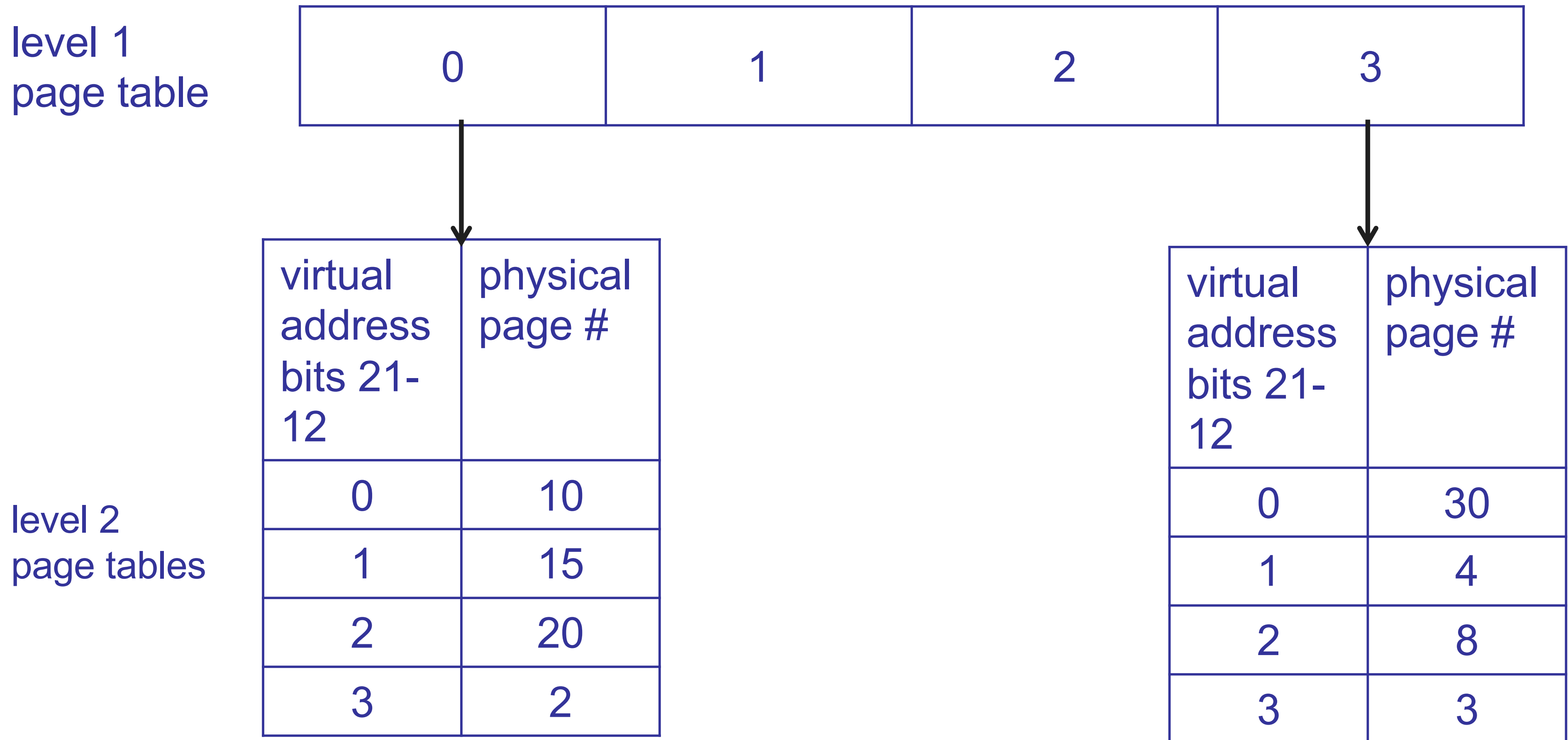


Recap: Paging

- Virtual address to physical address translation using page table

Virtual page #	Physical page #	Resident	Protection
0	105	0	RX
1	15	1	R
2	283	1	RW
3	invalid		
...	invalid		
1048575	invalid		

Recap: Multi-Level Paging



Reduce translation overhead using TLB

Page Replacement

- Not all valid pages may fit in physical memory
  - Some pages must be paged out (written) to disk
  - Disk is the “backing store”, phys mem acts as cache
- To read in a page from disk, some resident page may need to be paged out first
- Which page to evict when you need a free frame?
  - Goal: minimize page faults

Replacement policies

- Random
- FIFO
  - Replace page brought into memory longest time ago
  - May replace pages that continue to be frequently used

A	B	C	D	C	E	A	C	D	B	C
A	A	A	A	A	E	E	E	E	E	E
	B	B	B	B	B	A	A	A	A	A
		C	C	C	C	C	C	C	B	B
			D	D	D	D	D	D	D	C

Optimal replacement

- Can you think of optimal strategy?
  - Replace page that won't be used for the longest time in the future
    - » Minimizes cache misses
    - » Requires knowledge of the future

A	B	C	D	C	E	A	C	D	B	C
A	A	A	A	A	A	A	A	A	A	A
	B	B	B	B	E	E	E	E	B	B
		C	C	C	C	C	C	C	C	C
			D	D	D	D	D	D	D	D

Least recently used (LRU)

- Approximate OPT by using past references
  - If page hasn't been used for a while, it probably won't be used for a long time in the future
- Why would this work well?
  - Temporal locality: Recently accessed pages are often accessed again

A	B	C	D	C	E	A	C	D	B	C
A	A	A	A	A	E	A	A	A	A	A
	B	B	B	B	B	B	B	B	B	B
		C	C	C	C	C	C	C	C	C
			D	D	D	D	D	D	D	D

LRU

- LRU: annoyingly good approximation of OPT
  - Surprisingly hard to beat with more sophistication
  - But, can also be very wrong

- Can you identify worst-case pattern for LRU?

A	B	C	D	E	A	B	C	D	E	A
A	A	A	A	E	E	E	E	D	D	D
	B	B	B	B	A	A	A	A	E	E
		C	C	C	C	B	B	B	B	A
			D	D	D	D	C	C	C	C

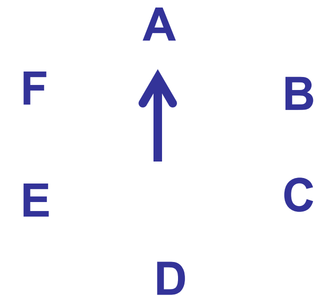
Approximating LRU

- Most MMUs maintain a “referenced” bit for each resident page
  - Set by MMU when page is read or written
  - Can be cleared by OS

- How to use reference bit to identify old pages?
  - Clear reference bit for all pages
  - After some time, examine reference bits
  - Reference bit = 0 for pages not accessed recently

Clock replacement algorithm

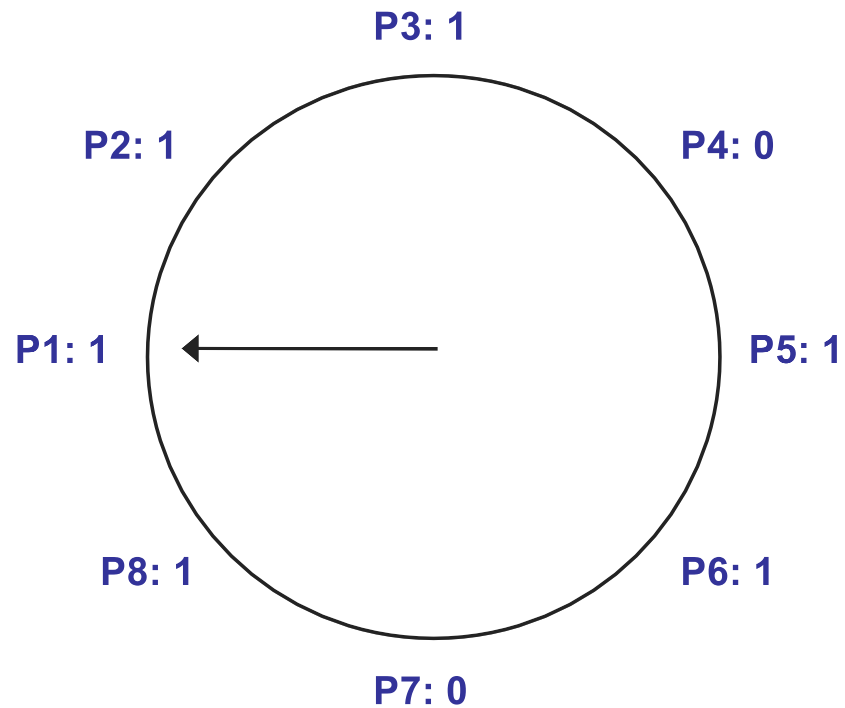
- Arrange resident pages around a clock

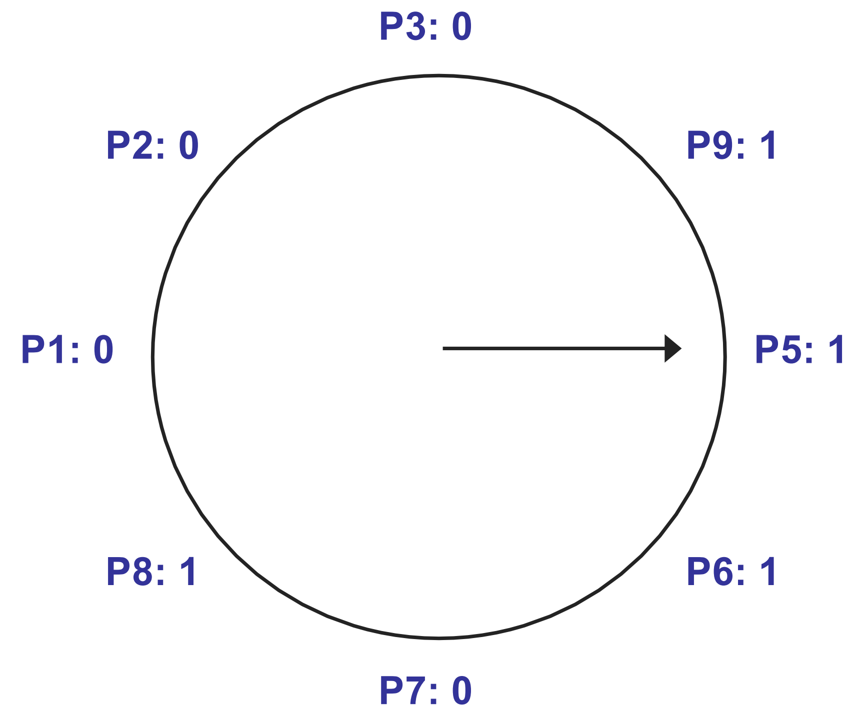


- To select a page for eviction:
  - Consider page pointed to by clock hand
  - If not referenced, evict page
  - If referenced, clear reference bit

- What if all pages referenced since last sweep?
- What about new pages?

Clock example





Page eviction

- Where to evict page to?
- When do you NOT need to write page to disk?
  - Rely on hardware/MMU to maintain dirty bit in PTE

- Why not write to disk on every store?
- When do you save by not writing on every store?
- How can you optimize eviction?

Project 2

- Fill out peer evaluations
- Keep track of interesting bugs for interviews

Page table contents

Written by OS, Read by MMU

Written by OS/MMU
Read by OS

Physical page #	Resident	Protection	Dirty	Referenced

```
if (virtual page is non-resident or protected) {
        trap to OS fault handler
        retry access
} else {
        physical page # = pageTable[virtual page #].physPageNum
        pageTable[virtual page #].referenced = true
        if (access is write) {
                pageTable[virtual page #].dirty = true
        }
        access physical memory at {physical page #}{offset}
}
```

Page table contents

Physical page #	Resident	Protection	Dirty	Referenced

- Why no valid bit in PTE?
  - All invalid virtual pages are non-resident
- Valid non-resident pages: where's disk block?
  - OS must maintain this, MMU simply traps to OS
- How can we make do without resident bit?
  - Clear protection bits when non-resident

Page table contents

Physical page #	Protection	Dirty	Referenced

- Can we make do without dirty bit?
  - Have OS set the dirty bit itself
  - Naive solution: trap on every store & mark dirty

- How to reduce # of page faults?
  - Only care about transition from clean to dirty
  - Clean pages have 0 write protection bit
  - Dirty pages have usual value

Page table contents

Physical page #	Protection	Referenced

- Can we make do without referenced bit?
  - Can use similar trick as that used for dirty bit
  - Insight: only care about unreferenced->referenced

- MMU simpler but page fault handler more complex

Mid-Term

- 6:30-8:30pm next Tuesday (not Michigan time!)
  - No lecture on Monday, office hours instead
- Assigned seating
  - Will post room assignments on Piazza
  - Sorted by student ID in every room

Mid-Term

- Closed book exam
- All topics until end of "Threads and Concurrency" + Projects 1 and 2

- Questions will be of two types:
  - Analysis
  - Synthesis (think before coding!)

How to Study?

- Practice sample exams
  - Emulate exam environment
- Understand parts of project code you did not write
- Pay attention to lecture material not on projs
- Get together in groups and create questions

Exam-taking tips

- Skim problems – answer easiest first
- Read coding questions carefully
  - Think and design before writing code
- Don't get bogged down on any 1 question
  - Can get partial credit even on tough questions

How to Answer: Synthesis

- Write C++ style syntax
  - Can use STL data structures
- Examples:
  - queue<thread*> readyq
  - queue.push(t)
  - mutex.lock()
  - sem.down()
- Remember to initialize all variables
- Be aware of layer at which you are writing code

How to Answer: Analysis

- Show all possible interleavings in trace
- Example:
  - thread 1: call produce()
  - thread 1: cokeLock.lock()
  - thread 1: while(numCokes==0)
  - thread 1: noCoke.wait(cokeLock)
  - thread 2: call consume()
  - thread 2: cokeLock.lock()
  - thread 3: noCoke.signal()