MEMORY: SWAPPING

Shivaram Venkataraman

CS 537, Fall 2024

ADMINISTRIVIA

Project 3 due very soon?

Midterm 1: Multiple choice questions
Oct 15th from 5.45pm to 7.15pm

Old exams on Canvas
Review session
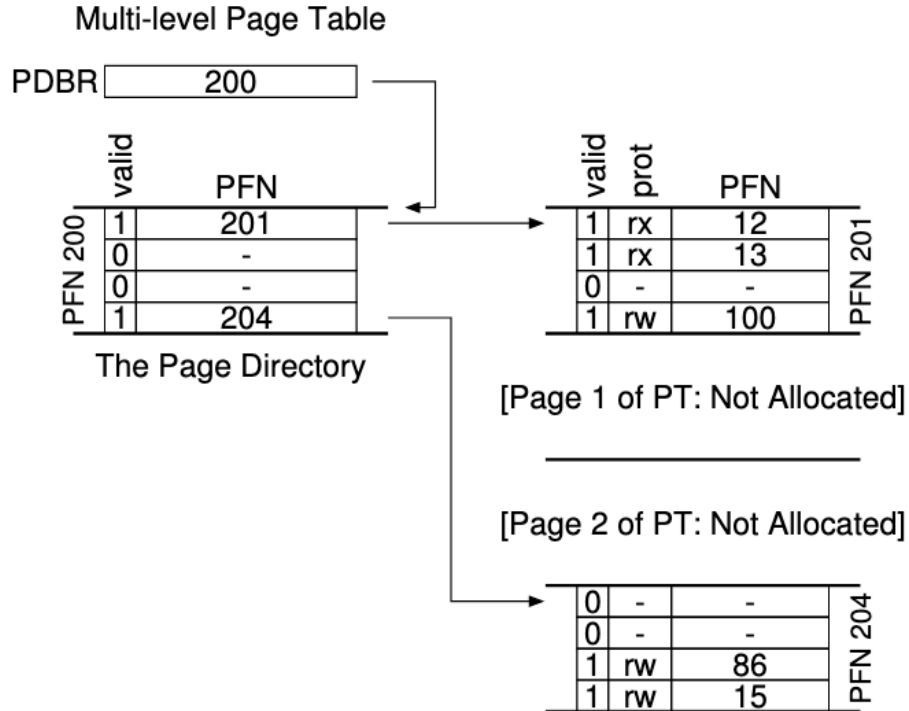   Lecture next week

AGENDA / LEARNING OUTCOMES

Memory virtualization

How we support virtual mem larger than physical mem?

What are mechanisms and policies for this?

RECAP

MULTILEVEL PAGE TABLES

Multi-level Page Table

PDBR | 200 |

The Page Directory

valid		PFN
1		201
0		-
0		-
1		204

(PFN 200)

valid	prot	PFN
1	rx	12
1	rx	13
0	-	-
1	rw	100

(PFN 201)

[Page 1 of PT: Not Allocated]

———————————

[Page 2 of PT: Not Allocated]

0	-	-
0	-	-
1	rw	86
1	rw	15

(PFN 204)

ADDRESS FORMAT FOR MULTILEVEL PAGING

30-bit address:

outer page	inner page	page offset (12 bits)

How should logical address be structured? How many bits for each paging level? Goal?

- – Each inner page table fits within a page
- – PTE size * number PTE = page size

    Assume PTE size = 4 bytes

    Page size = 2^{12} bytes = 4KB

- → # bits for selecting inner page =

Remaining bits for outer page:

- – 30 – ____ – ____ = ____ bits

MULTILEVEL TRANSLATION EXAMPLE

page directory

PPN	valid
0x3	1
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
0x92	1

page of PT (@PPN:0x3)

PPN	valid
0x10	1
0x23	1
-	0
-	0
0x80	1
0x59	1
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0

page of PT (@PPN:0x92)

PPN	valid
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
-	0
0x55	1
0x45	1

translate 0xFEED0

20-bit address:

outer page(4 bits)	inner page(4 bits)	page offset (12 bits)
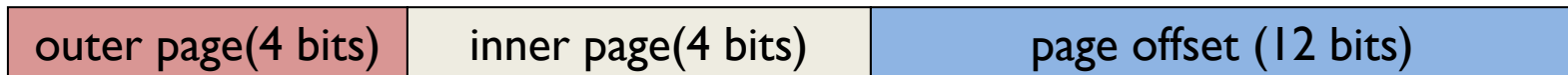
INVERTED PAGE TABLE

Only store entries for virtual pages w/ valid physical mappings

Naïve approach:

Search through data structure <ppn, vpn+asid> to find match

Too much time to search entire table

Better:

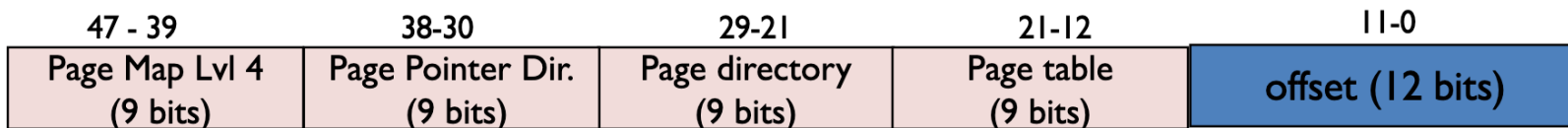Find possible matches entries by hashing vpn+asid

Smaller number of entries to search for exact match

Managing inverted page table requires software-controlled TLB

TRANSLATING LARGE PAGES
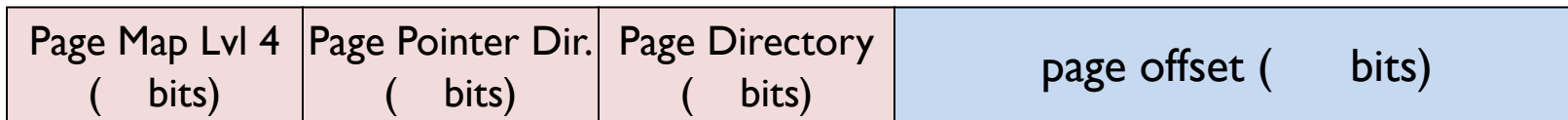
HugePages saves TLB entries. But how does it affect page translation?

4KB pages: 4 levels → 4 memory accesses

47 - 39	38-30	29-21	21-12	11-0
Page Map Lvl 4 (9 bits)	Page Pointer Dir. (9 bits)	Page directory (9 bits)	Page table (9 bits)	offset (12 bits)

2MB pages:

Page Map Lvl 4 (bits)	Page Pointer Dir. (bits)	Page Directory (bits)	page offset (bits)

SWAPPING

MOTIVATION

OS goal: Support processes when not enough physical memory
- Single process with very large address space
- Multiple processes with combined address spaces

User code should be independent of amount of physical memory
- Correctness, if not performance

Virtual memory: OS provides illusion of more physical memory

Why does this work?
- Relies on key properties of user processes (workload)
  and machine architecture (hardware)

WORKLOAD PROPERTIES

Leverage locality of reference within processes

- – Spatial: reference memory addresses **near** previously referenced addresses
- – Temporal: reference memory addresses that have referenced in the past
- – Processes spend majority of time in small portion of code
  - • Estimate: 90% of time in 10% of code

Implication:

- – Process only uses small amount of address space at any moment
- – Only small amount of address space must be resident in physical memory

HARDWARE: MEMORY HIERARCHY

Leverage memory hierarchy of machine architecture
Each layer acts as "backing store" for layer above
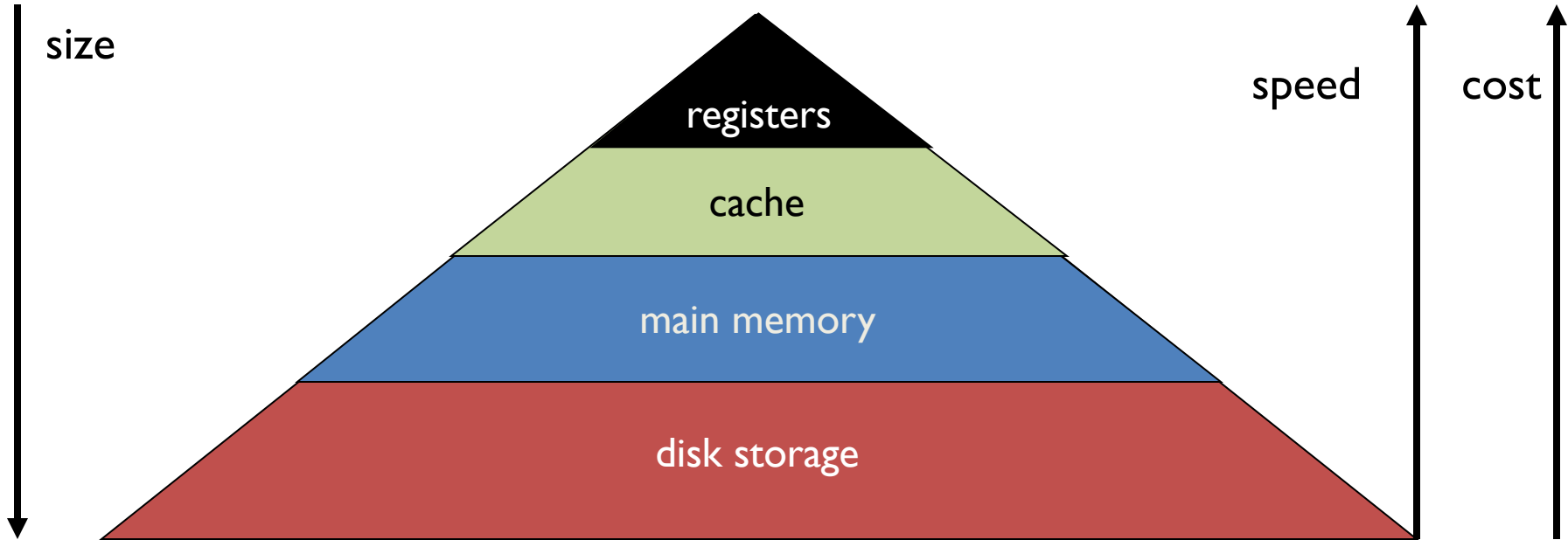
SWAPPING INTUITION

Idea: OS keeps unreferenced pages on disk
- Slower, cheaper backing store than memory

Process can run when not all pages are loaded into main memory
OS and hardware cooperate to make large disk seem like memory
- Same behavior as if all of address space in main memory

Requirements:
- OS must have **mechanism** to identify location of each page in address space →
  in memory or on disk
- OS must have **policy** to determine which pages live in memory and which on disk

VIRTUAL ADDRESS SPACE MECHANISMS

Each page in virtual address space maps to one of three locations:
- Physical main memory: Small, fast, expensive
- Disk (backing store): Large, slow, cheap
- Nothing (error): Free

Extend page tables with an extra bit: present
- permissions (r/w), valid, present
- Page in memory: present bit set in PTE
- Page on disk: present bit cleared
  - PTE points to block on disk
  - Causes trap into OS when page is referenced
  - Trap: page fault

Disk

Phys Memory

PFN	valid	prot	present
10	1	r-x	1
-	0	-	-
23	1	rw-	0
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
-	0	-	-
28	1	rw-	0
4	1	rw-	1

What if access vpn 0xb?

VIRTUAL MEMORY MECHANISMS

First, hardware checks TLB for virtual address

  – if TLB hit, address translation is done; page in physical memory

Else   ...

  – Hardware or OS walk page tables

  – If PTE designates page is present, then page in physical memory
       (i.e., present bit is cleared)

Else

  – Trap into OS (not handled by hardware)

  – OS selects victim page in memory to replace

    • Write victim page out to disk if modified (use dirty bit in PTE)

  – OS reads referenced page from disk into memory

  – Page table is updated, present bit is set

  – Process continues execution

QUIZ 8

Virtual address space of 16KB with 64-byte pages.
How many bits in a virtual address?

Total number of entries in the Linear Page Table?

Two-level page table with a page directory.
Bits to select the inner page?
(assume PTE size = 4 bytes)

```
page   0:00000000000000000000000000000000000000000000000000000000000000000
page   1:7f7f7f7f7f7f7fe57f7f7f7f7f7f7f7f7f7f7f7f7f7f7fde7f7f7f7f7f7f7fb77f
page   2:151d0d0a111d080905130e070c01091e12081d0b07010406071b0807121c0917
page   3:7f7f7f7f7f7f7f7fed7f7f7f7f7f7f7f8e7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f
page   4:7f7f7f7f7f7f7f7f7f7f7f7f7f877f7f7f7f7f7f7f7f7f7f7f7f7f7fe17f7f7f7f
page   5:1c010a0f061b03021e00060c1b0a111813190010001a00020d130013030a0116
page   6:00000000000000000000000000000000000000000000000000000000000000000
page   7:0d0104011e0e08040803181c1902121a0c180010170d031e190816051316120d
page   8:7f7f7f7f7f9f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f
page   9:00000000000000000000000000000000000000000000000000000000000000000
page  10:00000000000000000000000000000000000000000000000000000000000000000
page  11:0e111413081114091a041e1d1e000c0216121616001a1d13081d101b131e1007
page  12:0d040a0e080a0e1606050e090704191803140d02021e0310151715020b031618
page  13:8384fe9588a57f9bc1cfebccd0e87fa79ef3977ffda3f8d5ecc3a97f7f909981
page  14:07091c0408110e0d0004091a1318041e190d1d0e0a160415051c131a1b141206
page  15:00021b1307090f161c04061e08020f0c100907171d0f05141a1d0f1714001002
page  16:7ffcfa7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7fbd7f7f7f7f7f
page  17:130a18141d06021b13080903130c0810140e0b1b131716011a0710141e171206
page  18:0614140a1c1411010c080e1c1a01151c10021a0d1e1b191c021809040b12000d
page  19:00000000000000000000000000000000000000000000000000000000000000000
page  20:071500160519121b1e19131a0d0b0f190a100d0014041602170003040150f0618
page  21:7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7fac7f7f7f7f7f7f7f7f
page  22:00000000000000000000000000000000000000000000000000000000000000000
page  23:7f7f7fc07f7f7f7f7f827f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7fd17f7f7f7f7f7f
page  24:00000000000000000000000000000000000000000000000000000000000000000
page  25:7f7fb07f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7fb5
page  26:151d0602080a1a0101100e06150c1e061003031d1b170f14070506080c0f080a
page  27:7f7f7f7f7f7f7fef7f7f7f7fca7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f
page  28:00000000000000000000000000000000000000000000000000000000000000000
page  29:030e0e0b02141e0b1b0a080e1e1813010d00010b07030f181c1c0d051d0d0a19
page  30:7f7fb97f7f7f7f7f7f7f7f7f9a7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f
```

Page size are 32 bytes
VA space is 1024 pages (32 KB)
Physical Mem 128 pages

Multi-level page table.
 Upper five bits index into PD
 Each page holds 32 PTEs.

The format of a PTE and PDE is
VALID | PFN6 ... PFN0

PDBR has 13 (decimal)

0x0214

Page 3

:7f7f7f7f7f7f7f7f7fed7f7f7f7f7f7f7f7f8e7f7f7f7f7f7f7f7f7f7f7f7f7f7f7f

Page 7

0d0104011e0e08040803181c1902121a0c180010170d031e190816051316120d

SWAPPING POLICIES

SWAPPING POLICIES

Goal: Minimize number of page faults

- Page faults require milliseconds to handle (reading from disk)
- Implication: Plenty of time for OS to make good decision

OS has two decisions
- Page selection
  When should a page (or pages) on disk be **brought into** memory?

- Page replacement
  Which resident page (or pages) in memory should be **thrown out** to disk?

PAGE SELECTION

Demand paging: Load page only when page fault occurs
- Intuition: Wait until page must absolutely be in memory
- When process starts: No pages are loaded in memory
- Problems: Pay cost of page fault for every newly accessed page

Prepaging (anticipatory, prefetching): Load page before referenced
- OS predicts future accesses (oracle) and brings pages into memory early
- Works well for some access patterns (e.g., sequential)

Hints: Combine above with user-supplied hints about page references
- User specifies: may need page in future, don't need this page anymore, or sequential access pattern, ...
- Example: madvise() in Unix

PAGE REPLACEMENT

Which page in main memory should selected as victim?

- – Write out victim page to disk if modified (dirty bit set)
- – If victim page is not modified (clean), just discard

OPT: Replace page not used for longest time in future

- – Advantages: Guaranteed to minimize number of page faults
- – Disadvantages: Requires that OS predict the future; Not practical, but good for comparison

PAGE REPLACEMENT

FIFO: Replace page that has been in memory the longest
- Intuition: First referenced long time ago, done with it now
- Advantages: Fair: All pages receive equal residency; Easy to implement
- Disadvantage: Some pages may always be needed

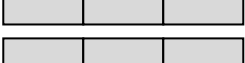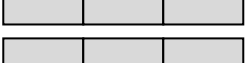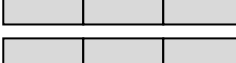LRU: Least-recently-used: Replace page not used for longest time in past
- Intuition: Use past to predict the future
- Advantages: With locality, LRU approximates OPT
- Disadvantages:
  - Harder to implement, must track which pages have been accessed
  - Does not handle all workloads well
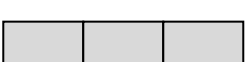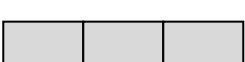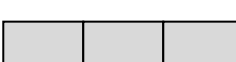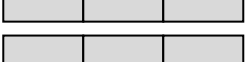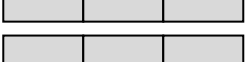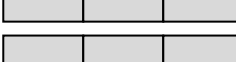
PAGE REPLACEMENT

OPT   FIFO   LRU

Page reference string:
DDBBACBDBD

Metric:
Miss count

D

D

B

B

A

C

B

D

B

D

PAGE REPLACEMENT COMPARISON

Add more physical memory, what happens to performance?

LRU, OPT:

- Guaranteed to have fewer (or same number of) page faults
- Smaller memory sizes are guaranteed to contain a subset of larger memory sizes
- Stack property: smaller cache always subset of bigger

FIFO:

- Usually have fewer page faults
- Belady's anomaly: May actually have more page faults!

FIFO PERFORMANCE MAY DECREASE!

Consider access stream: ABCDABEABCDE

Physical memory size: 3 pages vs. 4 pages

How many misses with FIFO?

IMPLEMENTING LRU

Software Perfect LRU
- OS maintains ordered list of physical pages by reference time
- When page is referenced: Move page to front of list
- When need victim: Pick page at back of list
- Trade-off: Slow on memory reference, fast on replacement

Hardware Perfect LRU
- Associate timestamp register with each page
- When page is referenced: Store system clock in register
- When need victim: Scan through registers to find oldest clock
- Trade-off: Fast on memory reference, slow on replacement (especially as size of memory grows)

In practice
LRU is an approximation anyway, so approximate more?

CLOCK ALGORITHM

Hardware
- Keep use (or reference) bit for each page frame
- When page is referenced: set use bit

Operating System
- Page replacement: Look for page with use bit cleared
  (has not been referenced for awhile)
- Implementation:
  - Keep pointer to last examined page frame
  - Traverse pages in circular buffer
  - Clear use bits as search
  - Stop when find page with already cleared use bit, replace this page

CLOCK: LOOK FOR A PAGE

Physical Mem:

Use=   Use=   Use=   Use=

| 0 | 1 | 2 | 3 |

↑
clock hand
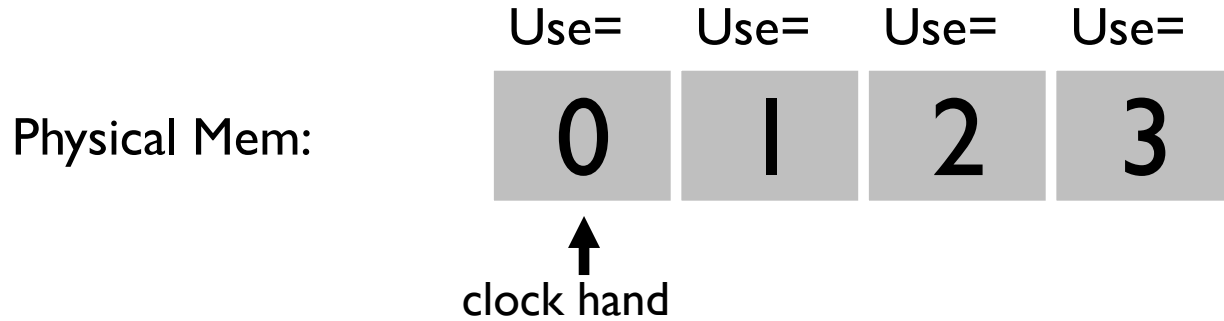
Use = 1,1,0,1 to begin

CLOCK EXTENSIONS

Replace multiple pages at once
- – Intuition: Expensive to run replacement algorithm and to write single block to disk
- – Find multiple victims each time and track free list

Use dirty bit to give preference to dirty pages
- – Intuition: More expensive to replace dirty pages
  Dirty pages must be written to disk, clean pages do not
- – Replace pages that have use bit and dirty bit cleared

SUMMARY: VIRTUAL MEMORY

Abstraction: Virtual address space with code, heap, stack

Address translation

- Contiguous memory: base, bounds, segmentation

- Using fixed sizes pages with page tables

Challenges with paging

- Extra memory references: avoid with TLB

- Page table size: avoid with multi-level paging, inverted page tables etc.

Larger address spaces: Swapping mechanisms, policies (LRU, Clock)

NEXT STEPS

Next class: Midterm 1 review!