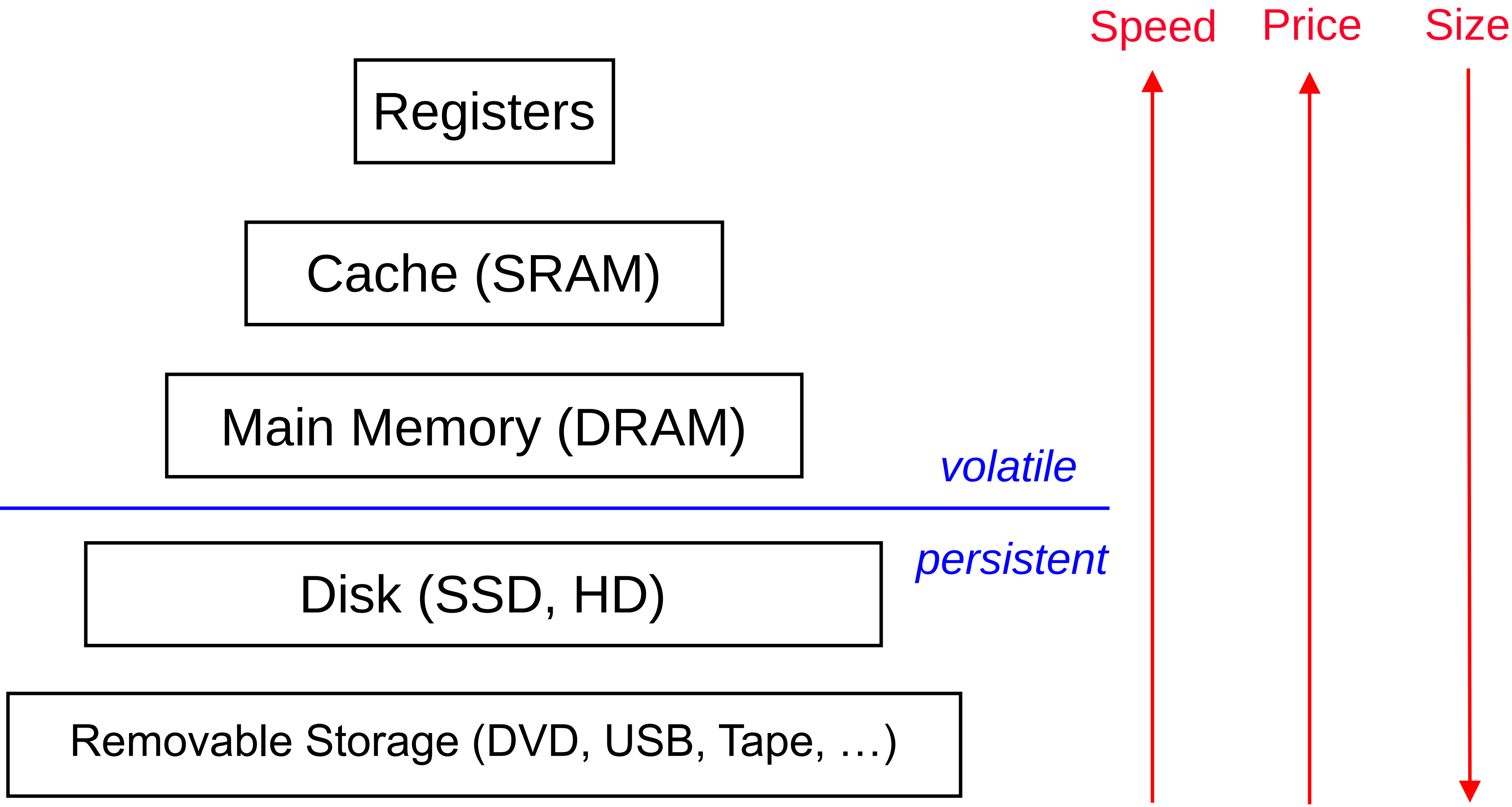
Memory Management

Motivation

- Ideally programmers want memory that is
  - Large
  - Fast
  - Non volatile
  - Transparent

- In reality, no such memory exists
  (or is prohibitively expensive)

Memory Management: Create pseudo-ideal memory



Outline

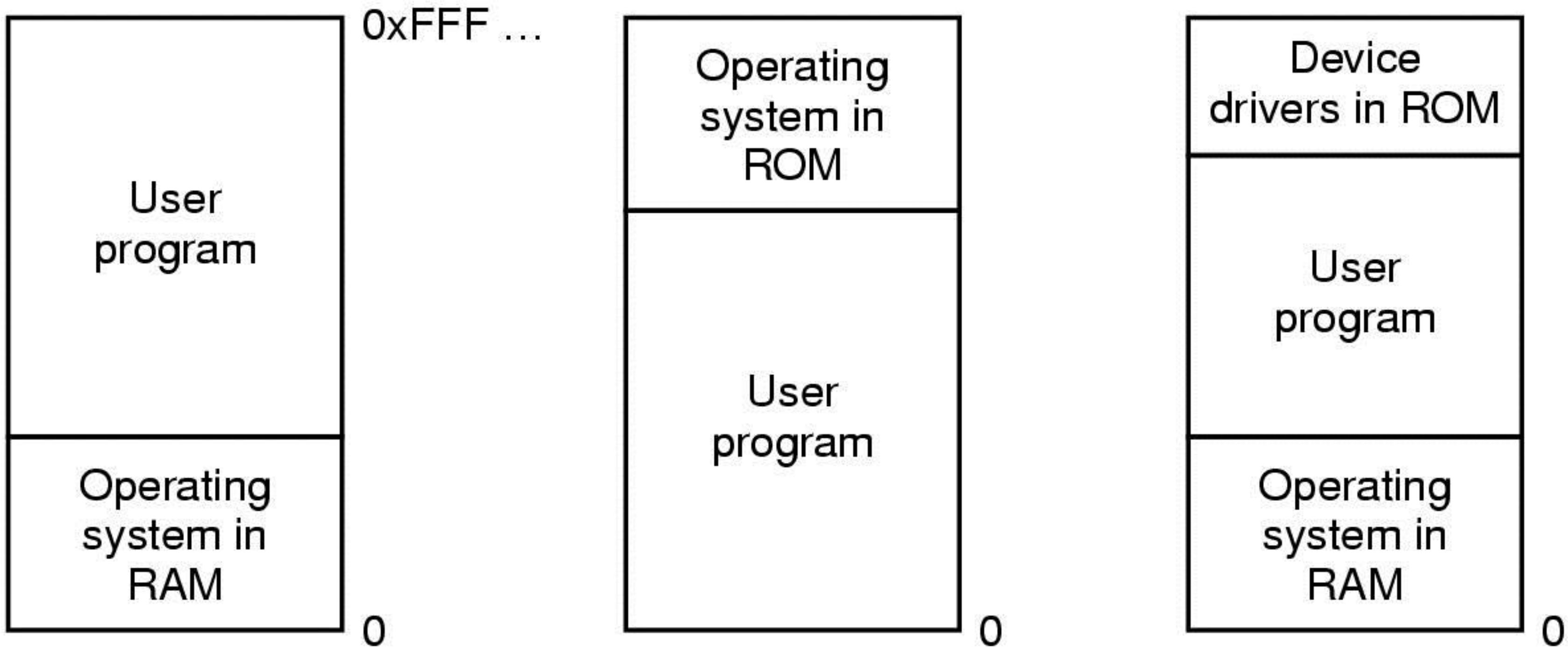
1. What to put in memory & resulting problems

2. Virtual memory and paging

3. Page replacement algorithms

Simplest Case: What *has to be* in memory?

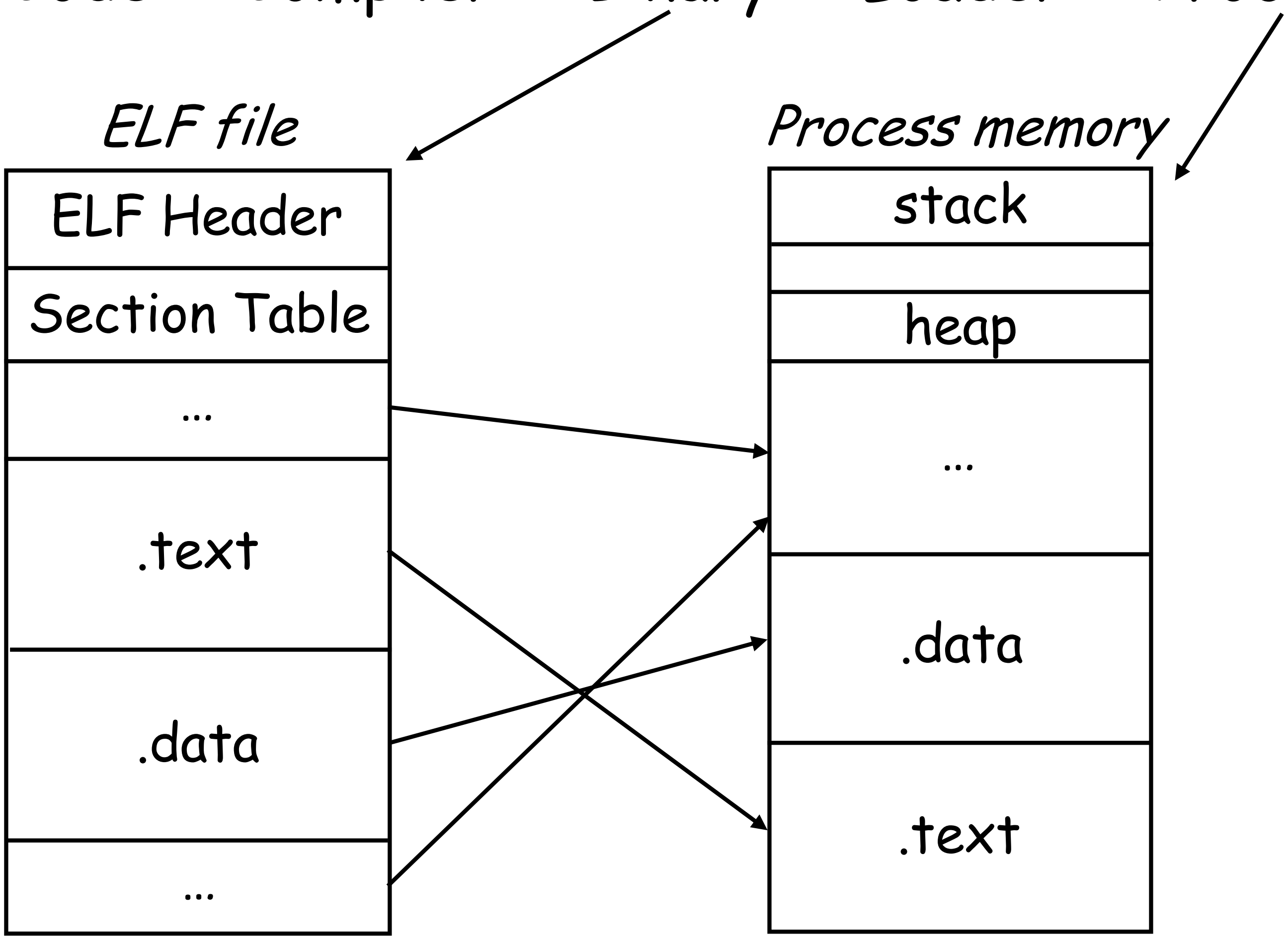
1. A process
2. The OS?



Three different types of memory organiztion

How Does a Process's memory look like?

Source Code -> Compiler -> Binary -> Loader -> Process



Example: ELF & Linux

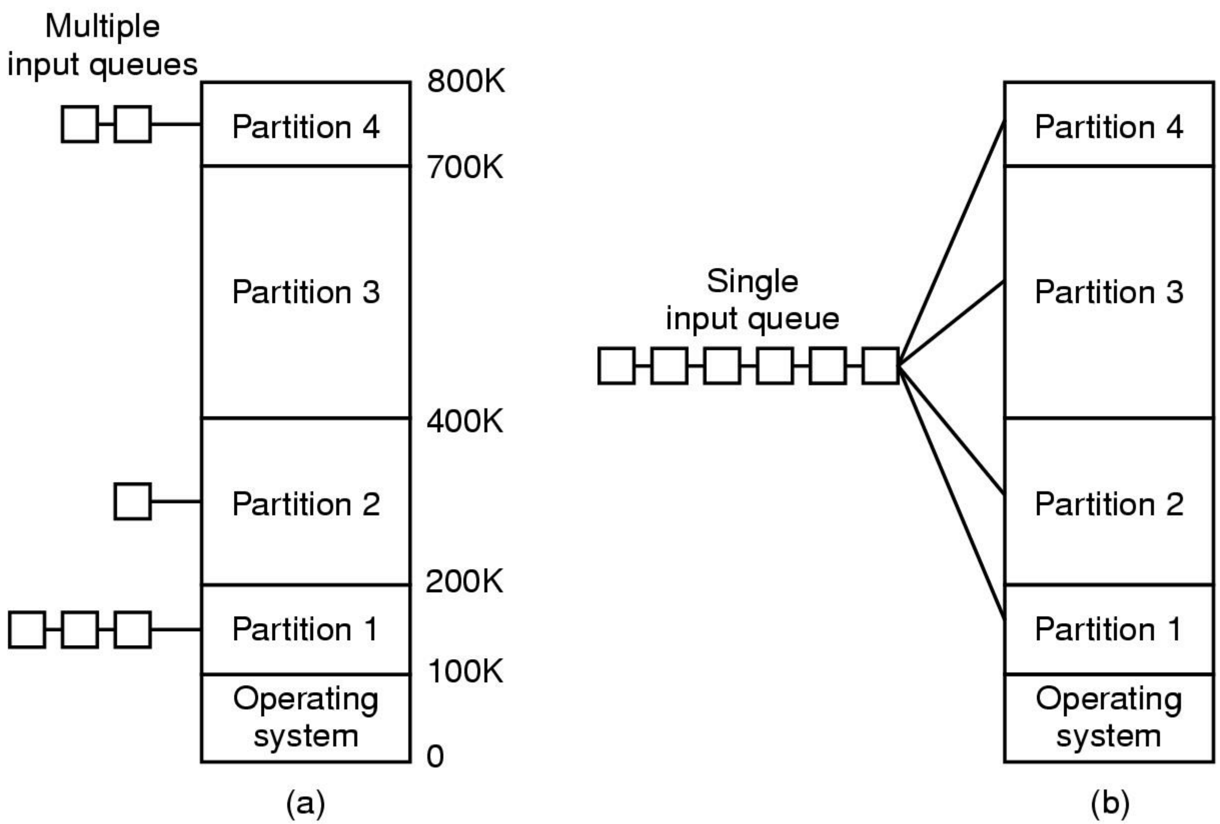
More Complex: Multiprogramming

- We usually want more than one process
  - Flexibility (i.e., *switching speed* in this case)
  - Utilization (CPU vs. I/O)
  - Multi-user environments

- Where in the memory should we place our programs?
  - Limited amount of memory
  - More than one program
  - Programs have different sizes
  - Program size might grow (or shrink)

- First try: partition the memory

Multiprogramming with Fixed Partitions

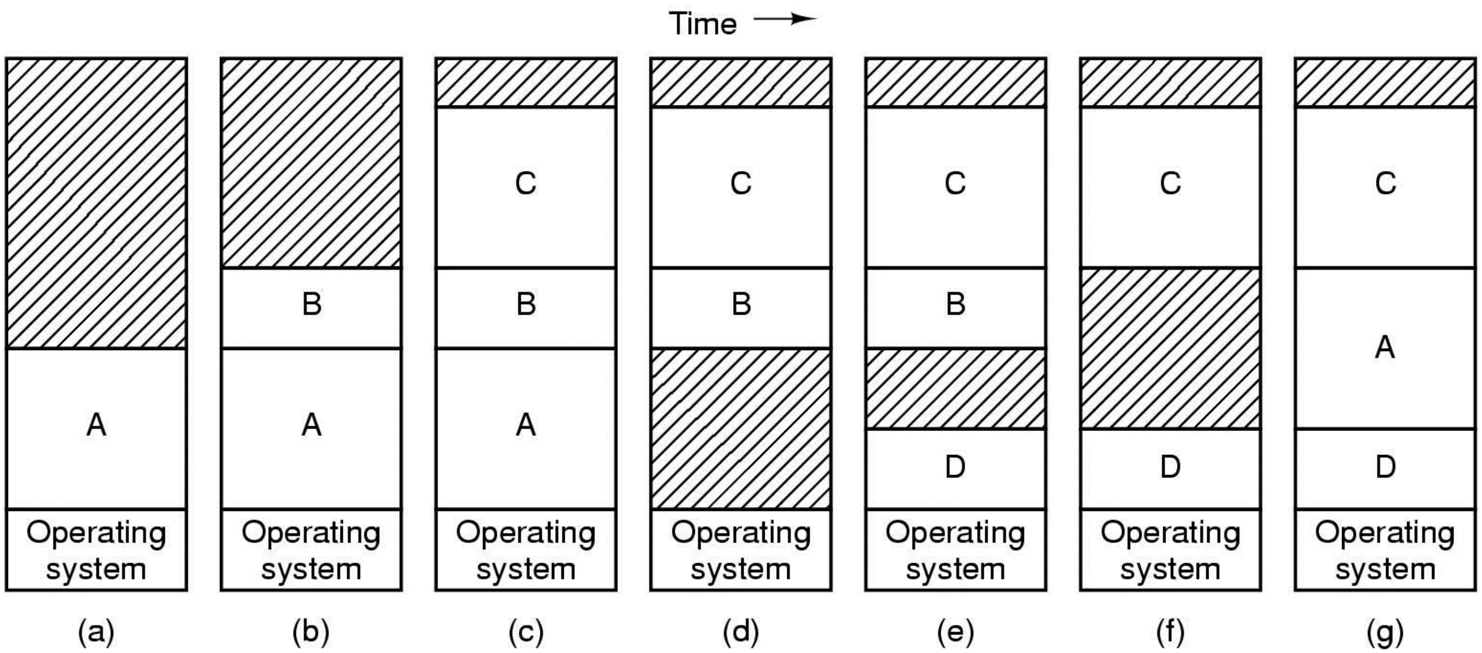


Separate input queues for each partition vs single input queue

⇨ What is the trade-off here?

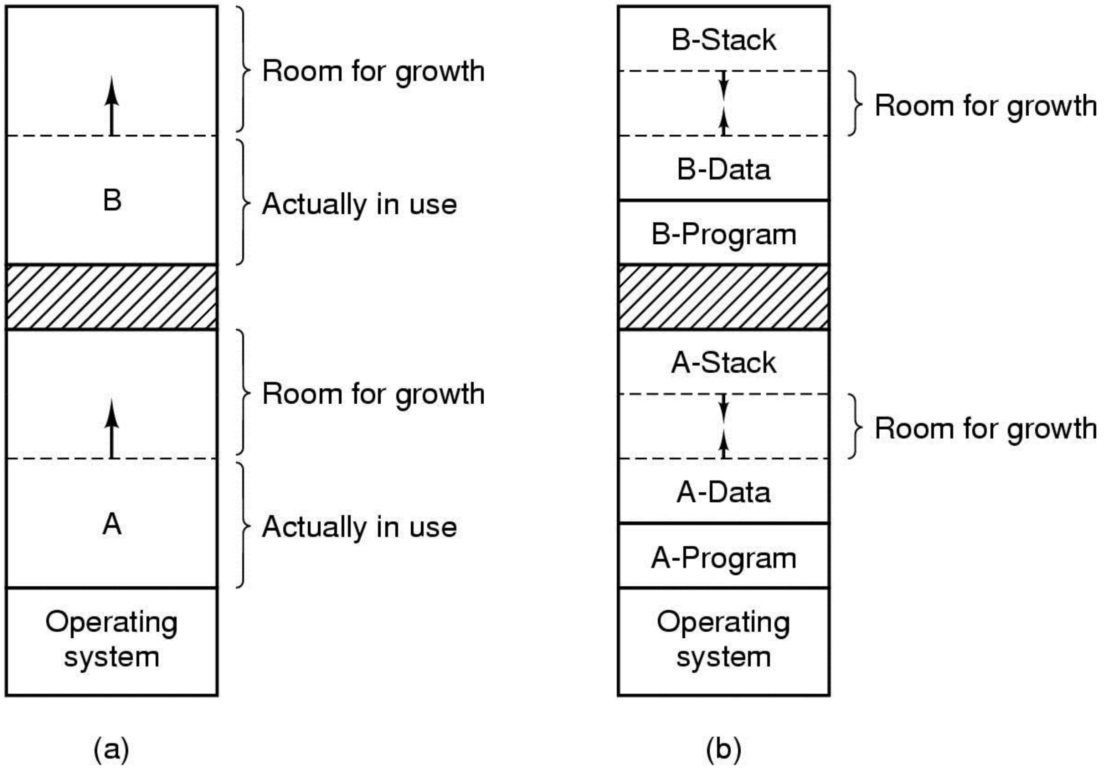
⇨ Can't we use dynamic partition sizes?

Swapping (1)



- Memory allocation changes as
  - Processes come into memory
  - Leave memory
- Solution for relocation needed
  - Addressing
  - Process “growth”?

Swapping (2)



(a) Allocating space for growing data segment
(b) Allocating space for growing stack & data segment

Issues with Swapping

- Summary of the discussed problems
  - Finding matching partition sizes / size constraints
  - Relocation
  - Dynamically growing/shrinking segments
  - Swapping large processes is too slow

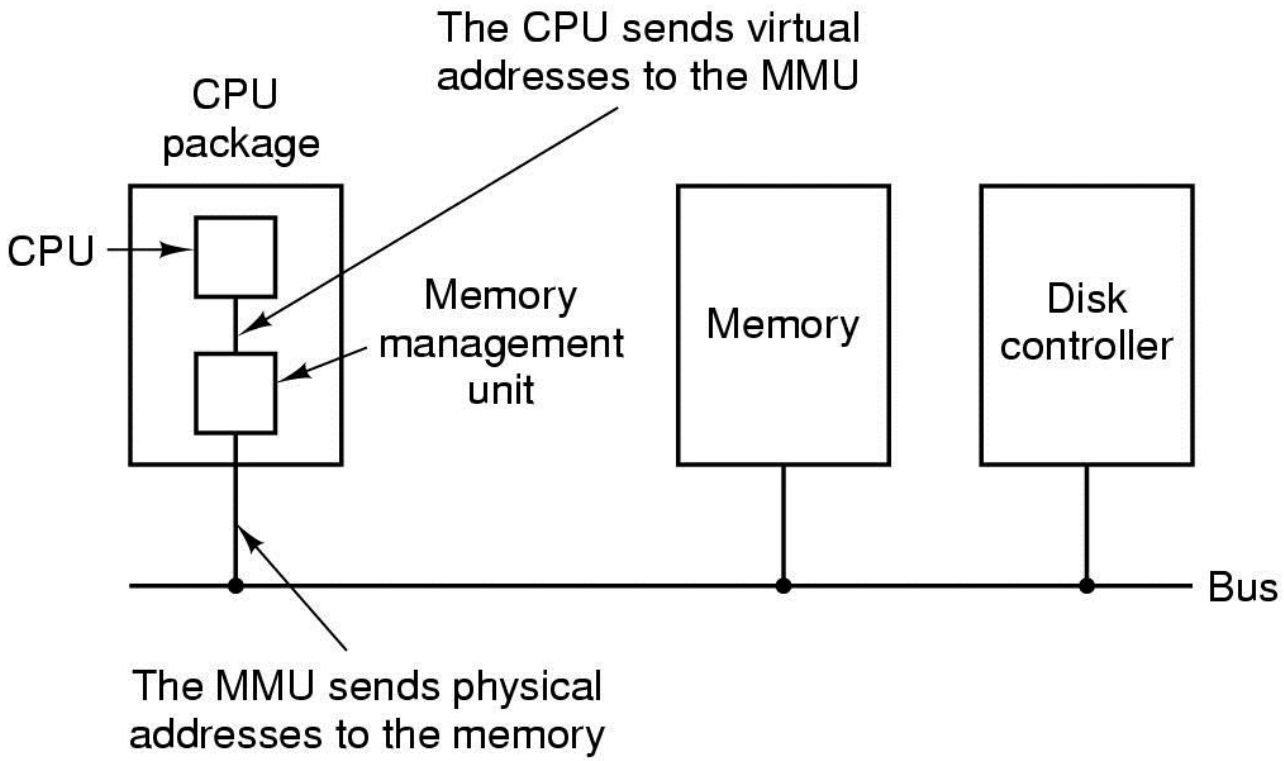
- (Partial) Solution: Virtual Memory

Virtual Memory

- Separates:
  - Virtual (logical) addresses
  - Physical addresses

- Requires a translation at run time
  - Virtual → Physical
  - Handled in HW (MMU)

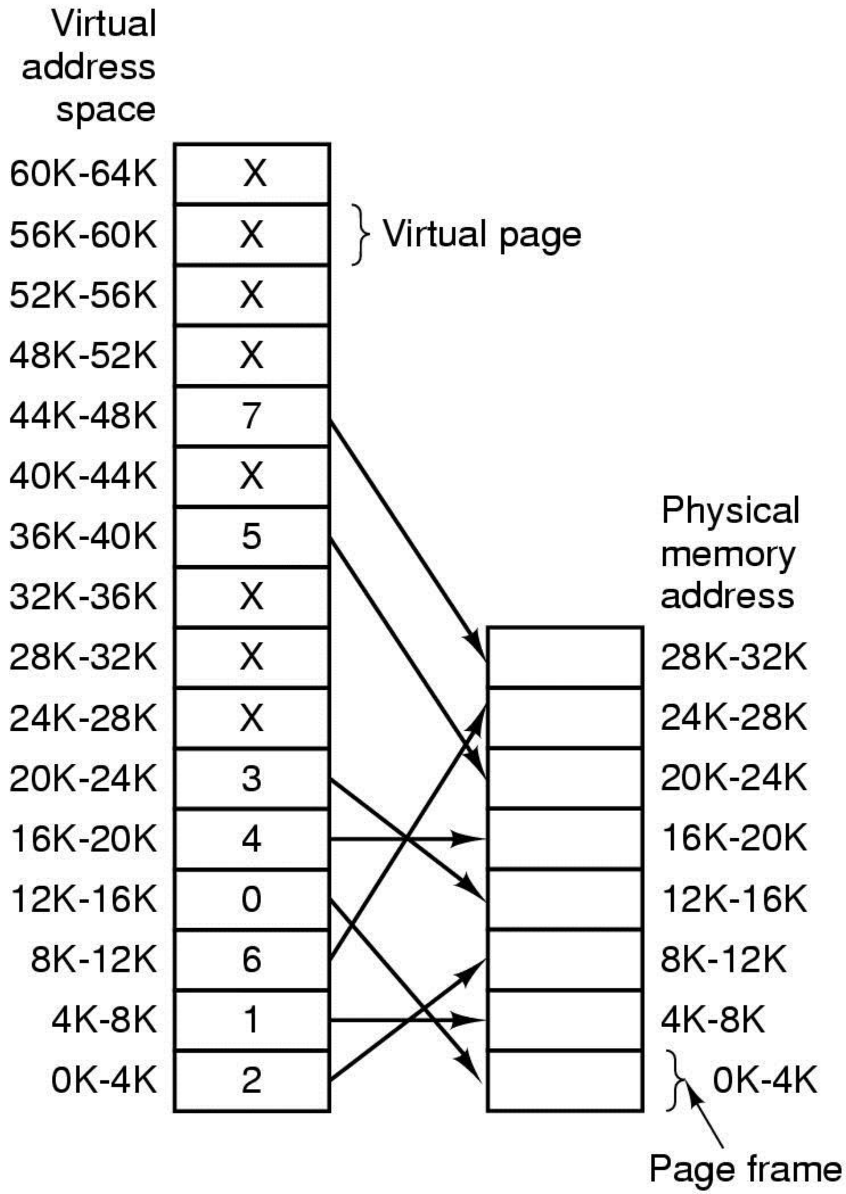
Virtual Memory: Paging



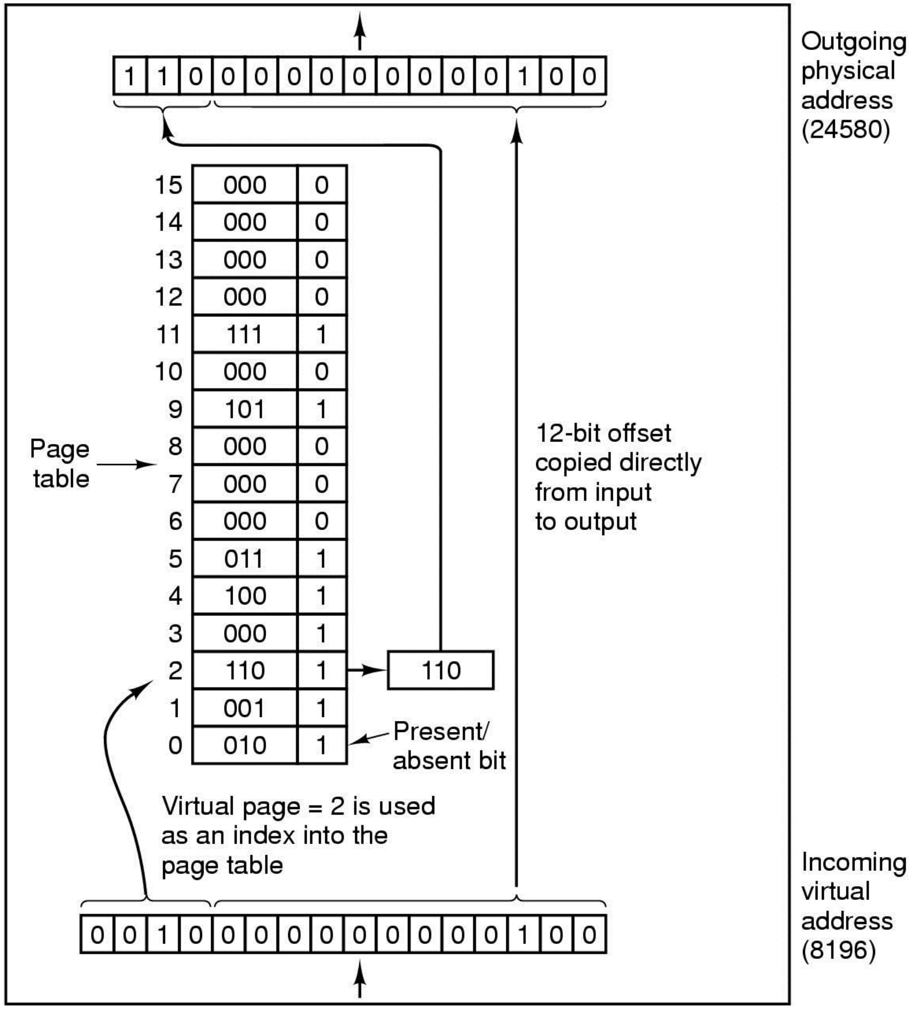
The position and function of the MMU

Paging

- The relation between virtual addresses and physical memory addresses given by page table
- One page table per process is needed (per thread?)
- Page table needs to be reloaded at context switch

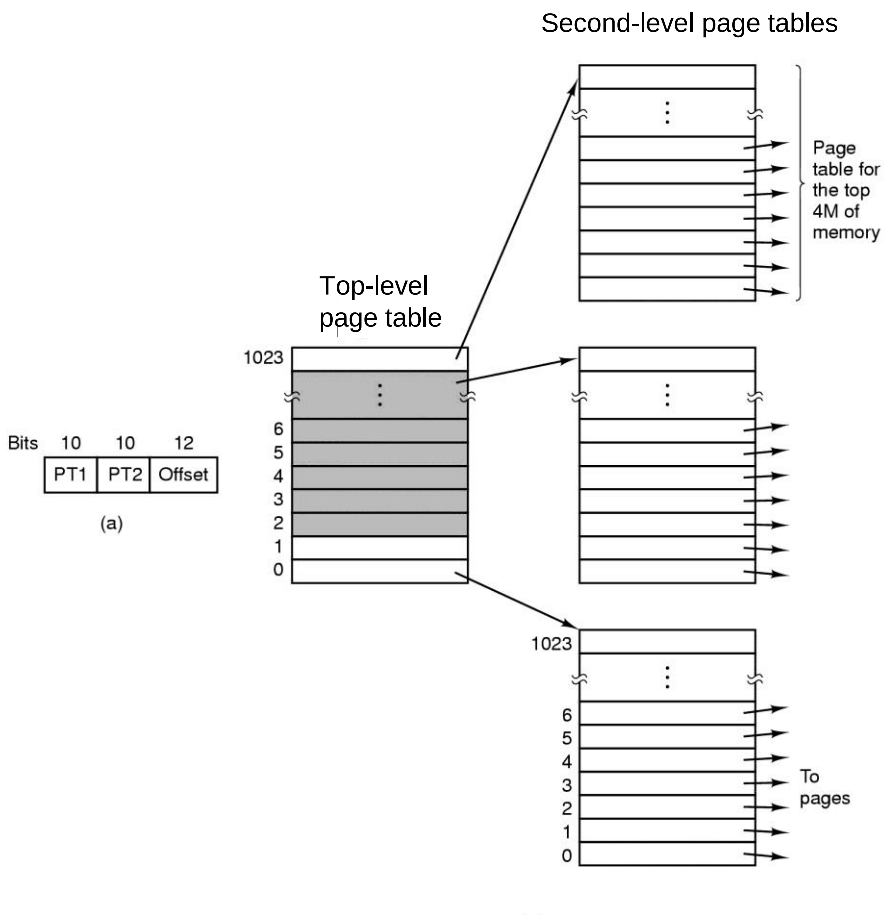


Page Tables



Internal operation of MMU with 16 4 KiB pages

Hierarchical Page Tables



- ❑ 32 bit address with 2 page table fields
- ❑ Two-level page tables

Paging

Every memory lookup:

1. Find the page in the page table
2. Find the (physical) memory location

Now we have two memory accesses (per reference) ☹

Solution: Translation Lookaside Buffer (TLB)
(yet another cache…)

TLBs – Translation Lookaside Buffers

Valid	Virtual page	Modified	Protection	Page frame
1	140	1	RW	31
1	20	0	R X	38
1	130	1	RW	29
1	129	1	RW	62
1	19	0	R X	50
1	21	0	R X	45
1	860	1	RW	14
1	861	1	RW	75

A TLB to speed up paging

TLB Handling

Memory lookup:

- Look for page in TLB (fast)
  - If hit, fine go ahead!
  - If miss, find it and put it in the TLB:
    - Find the page in the page table (hit)
    - Reload the page from disk (miss)

Paging

- So we know how to find a page table entry

- If it is not in the page table
  - Get it from disk

- What if the physical memory is full?
  - Throw some page out
  - Remember cache replacement policies?

Page Replacement Algorithms

- Page fault forces choice
  - Which page must be removed
  - Make room for incoming page(s)
- Modified page must first be saved
  - Unmodified just overwritten
- Better not to choose an often used page
  - Will probably need to be brought back in soon

Optimal Page Replacement Algorithm

- Replace page needed at the farthest point in future
  - Optimal but unrealizable

- Estimate by …
  - Logging page use on previous runs of process
  - Although this is impractical

Page Replacement Algorithms

1. Not Recently Used (NRU)
2. FIFO
3. Second Chance FIFO / Clock
4. Least Recently Used (LRU)
5. Not Frequently Used (NFU) / Aging
6. Working Set WSClock algorithm

1. Not Recently Used (NRU)

- Each page has Reference bit, Modified bit
  - Bits are set by HW when page is referenced, modified
  - Reference bit is periodically unset (at clock ticks)

- Pages are classified
  - Class 0: not referenced, not modified
  - Class 1: not referenced, modified
  - Class 2: referenced, not modified
  - Class 3: referenced, modified

- NRU removes page at random
  - From lowest numbered non empty class

- NRU is simple and gives decent performance

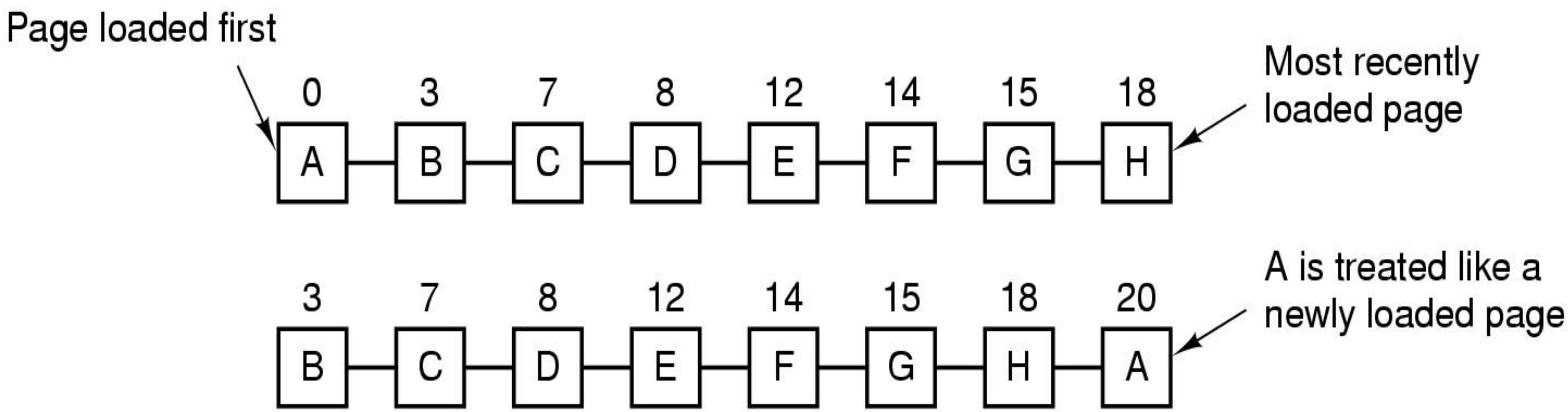
2. FIFO Page Replacement

- Maintain a linked list of all pages
  - In the order they came into memory

- Page at beginning of list replaced

- Disadvantage
  - Page in memory the longest may be used often

3. Second Chance FIFO



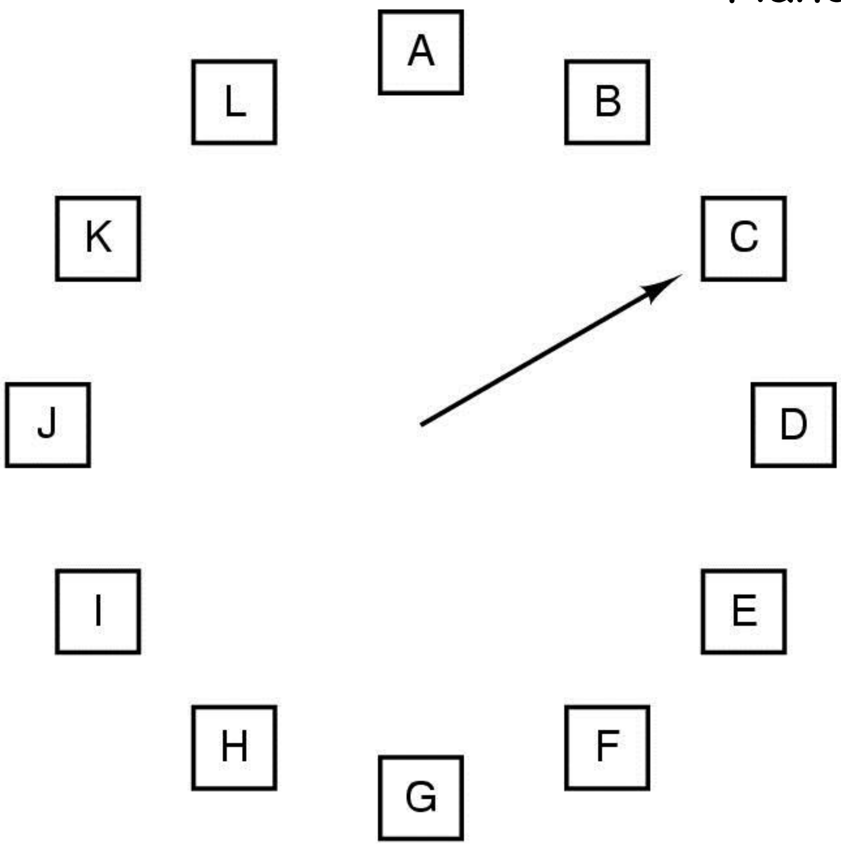
- Operation of a second chance
  - Pages sorted in FIFO order
  - Inspect the R bit, give the page a second chance if R=1
  - Put the page at the end of the list (like a new page)
  - Eventually the page might be selected anyway (how?)

  Example: Page fault @ 20, *A* has *R* bit set

- 2nd Chance moves pages around in the list …

The Clock Page Replacement Algorithm

Hand points to oldest page



When a page fault occurs, the page the hand is pointing to is inspected. The action taken depends on the R bit:

- R = 0: Evict the page
- R = 1: Clear R and advance hand

How does this compare with second chance?

4. Least Recently Used (LRU): HW solution 1

- Locality: pages used recently will be used soon
  - Throw out the page that has been unused longest

- Must keep a linked list of pages
  - Most recently used at front, least at rear
  - Update this list every memory reference !!

- Alternative: counter (64 bit) in the page table entry
  - Increment counter on every instruction
  - Store counter in page table on memory reference
  - Choose page with lowest value counter

LRU: HW solution 2

Page

	0	1	2	3
0	0	1	1	1
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0

Page

0	1	2	3
0	0	1	1
1	0	1	1
0	0	0	0
0	0	0	0

Page

0	1	2	3
0	0	0	1
1	0	0	1
1	1	0	1
0	0	0	0

Page

0	1	2	3
0	0	0	0
1	0	0	0
1	1	0	0
1	1	1	0

Page

0	1	2	3
0	0	0	0
1	0	0	0
1	1	0	1
1	1	0	0

0	0	0	0
1	0	1	1
1	0	0	1
1	0	0	0

0	1	1	1
0	0	1	1
0	0	0	1
0	0	0	0

0	1	1	0
0	0	1	0
0	0	0	0
1	1	1	0

0	1	0	0
0	0	0	0
1	1	0	1
1	1	0	0

0	1	0	0
0	0	0	0
1	1	0	0
1	1	1	0

LRU using a matrix – pages referenced in order 0,1,2,3,2,1,0,3,2,3

1. Set row k to 1
2. Set column k to 0

5. Simulating LRU in Software: NFU

- Both solutions (counter, matrix) require extra hardware
- And they don't scale very well with large page tables …
- Approximations of LRU can be done in SW

NFU (Not Frequently Used)

- A counter is associated with each page
- At each clock interrupt add R to the counter
- Problems?

NFU variant: Aging

- NFU tends to have a long memory!
- Pages frequently used in some phase may still be hot in later phases, even though they are not used!
- Solution: right shift counter and set leftmost MSB to R (R=1, c=010 → c'=101)
- This algorithm is called *aging*

Aging

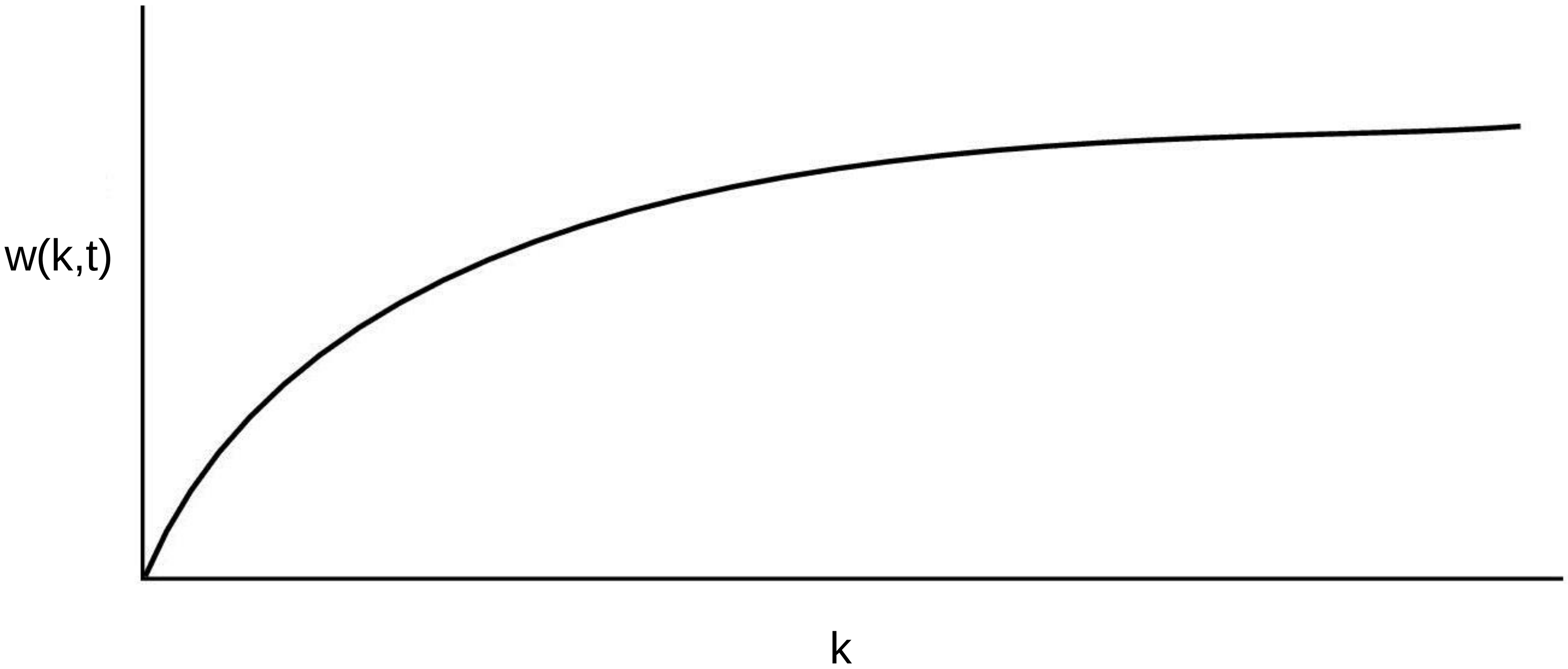
Page	R bits for pages 0-5, clock tick 0	R bits for pages 0-5, clock tick 1	R bits for pages 0-5, clock tick 2	R bits for pages 0-5, clock tick 3	R bits for pages 0-5, clock tick 4
	1 0 1 0 1 1	1 1 0 0 1 0	1 1 0 1 0 1	1 0 0 0 1 0	0 1 1 0 0 0
0	10000000	11000000	11100000	11110000	01111000
1	00000000	10000000	11000000	01100000	10110000
2	10000000	01000000	00100000	00100000	10001000
3	00000000	00000000	10000000	01000000	00100000
4	10000000	11000000	01100000	10110000	01011000
5	10000000	01000000	10100000	01010000	00101000
	(a)	(b)	(c)	(d)	(e)

- The aging algorithm <u>simulates</u> LRU in software

Aging

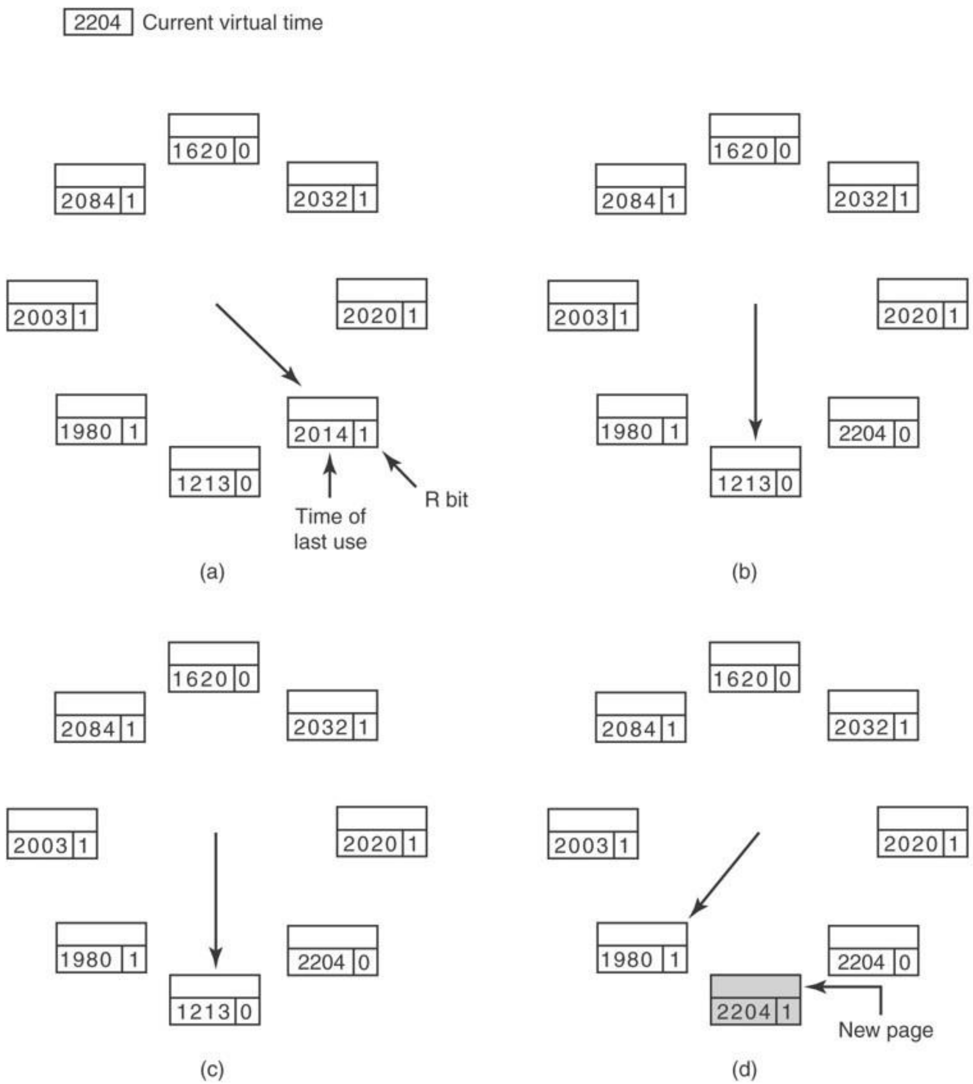
- This algorithm works fine, but …
  - Granularity
    - Page accesses between ticks?
  - Counter size
    - How long is the counter's "memory"?
- Tanenbaum: 8 bits @ 20ms considered ok

The Working Set Model



- The working set is the set of pages used by the *k* most recent memory references
- w(k,t) is the size of the working set at time *t*
- Note: after initial fast increase we see asymptotic behavior (many values of k give roughly the same working set size)

The WSClock Page Replacement Algorithm



- All pages form a ring (circular list)
- Each entry holds (time, R, M)
- R & M are set by the HW
- Start where the hand points
- if R==1
  - Set R = 0 (& update virt. time)
  - Move to next page
- if R==0
  - if age > τ && M==0: reclaim
  - if age > τ && M==1: schedule write
- When you get back to the start, either:
  - Some writes are scheduled
    - Search until one write finished
  - No writes are scheduled
    - Take first clean page
    - Take the page you're at

Operation of the WSClock algorithm

Review of Page Replacement Algorithms

Algorithm	Comment
Optimal	Not implementable, but useful as a benchmark
NRU (Not Recently Used)	Very crude
FIFO (First-In, First-Out)	Might throw out important pages
Second chance	Big improvement over FIFO
Clock	Realistic
LRU (Least Recently Used)	Excellent, but difficult to implement exactly
NFU (Not Frequently Used)	Fairly crude approximation to LRU
Aging	Efficient algorithm that approximates LRU well
Working set	Somewhat expensive to implement
WSClock	Good efficient algorithm