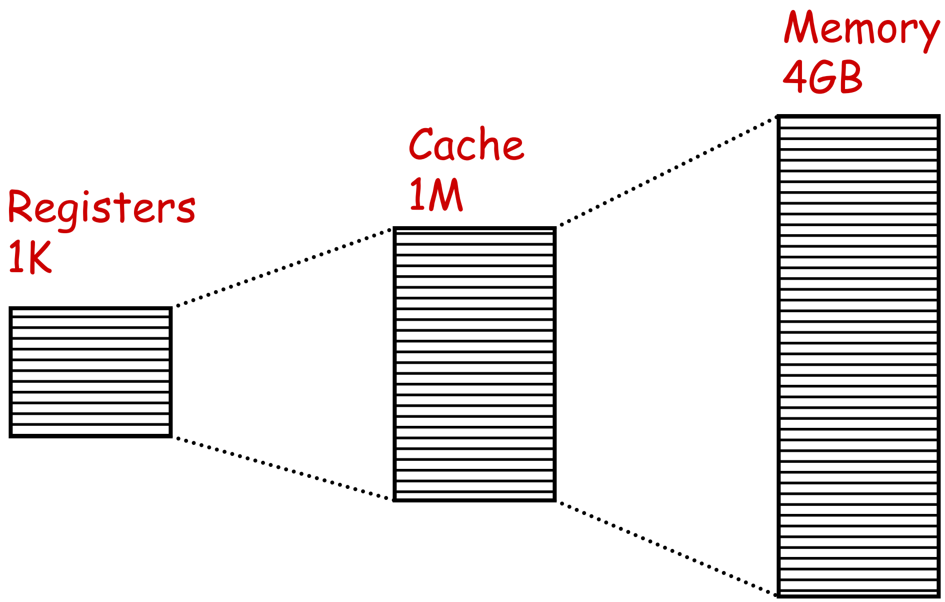
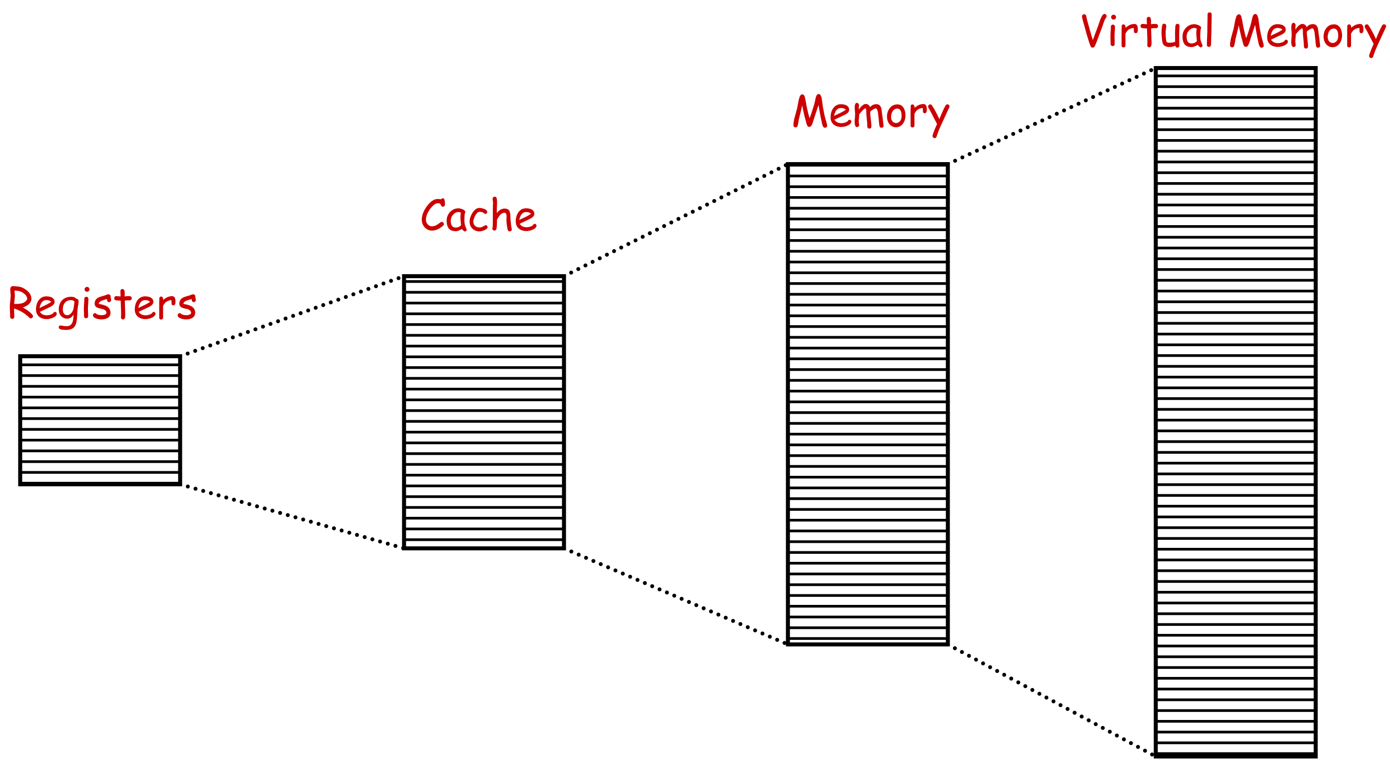
Virtual Memory

Memory Hierarchy



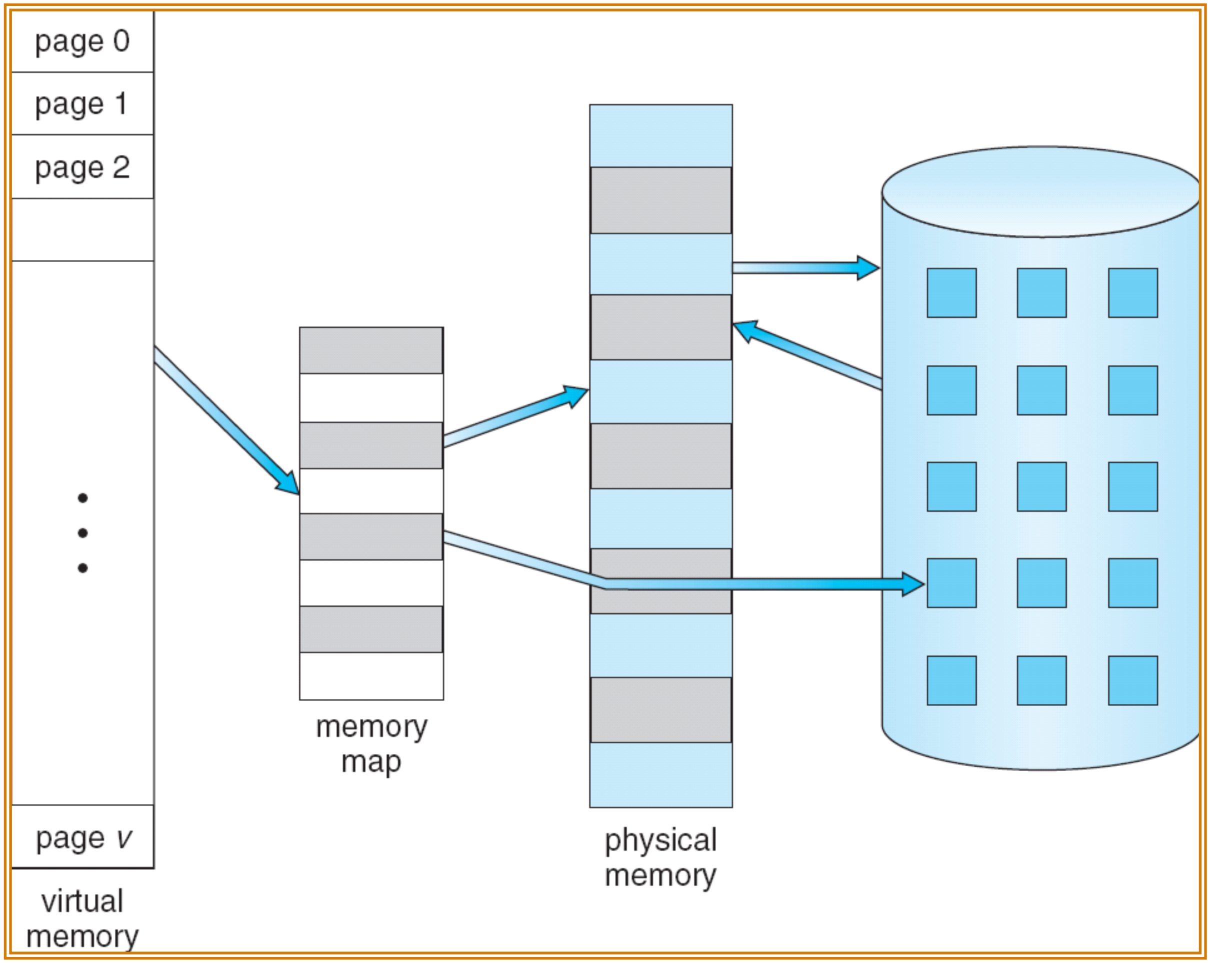
Question: What if we want to run a process that requires 10GB memory?

Memory Hierarchy



Answer: Pretend we had something bigger
=> Virtual Memory

Virtual Memory That is Larger Than Physical Memory



Demand Paging

- Bring a page into memory only when it is needed
  - Less I/O needed
  - Less memory needed
  - Faster response
  - More users

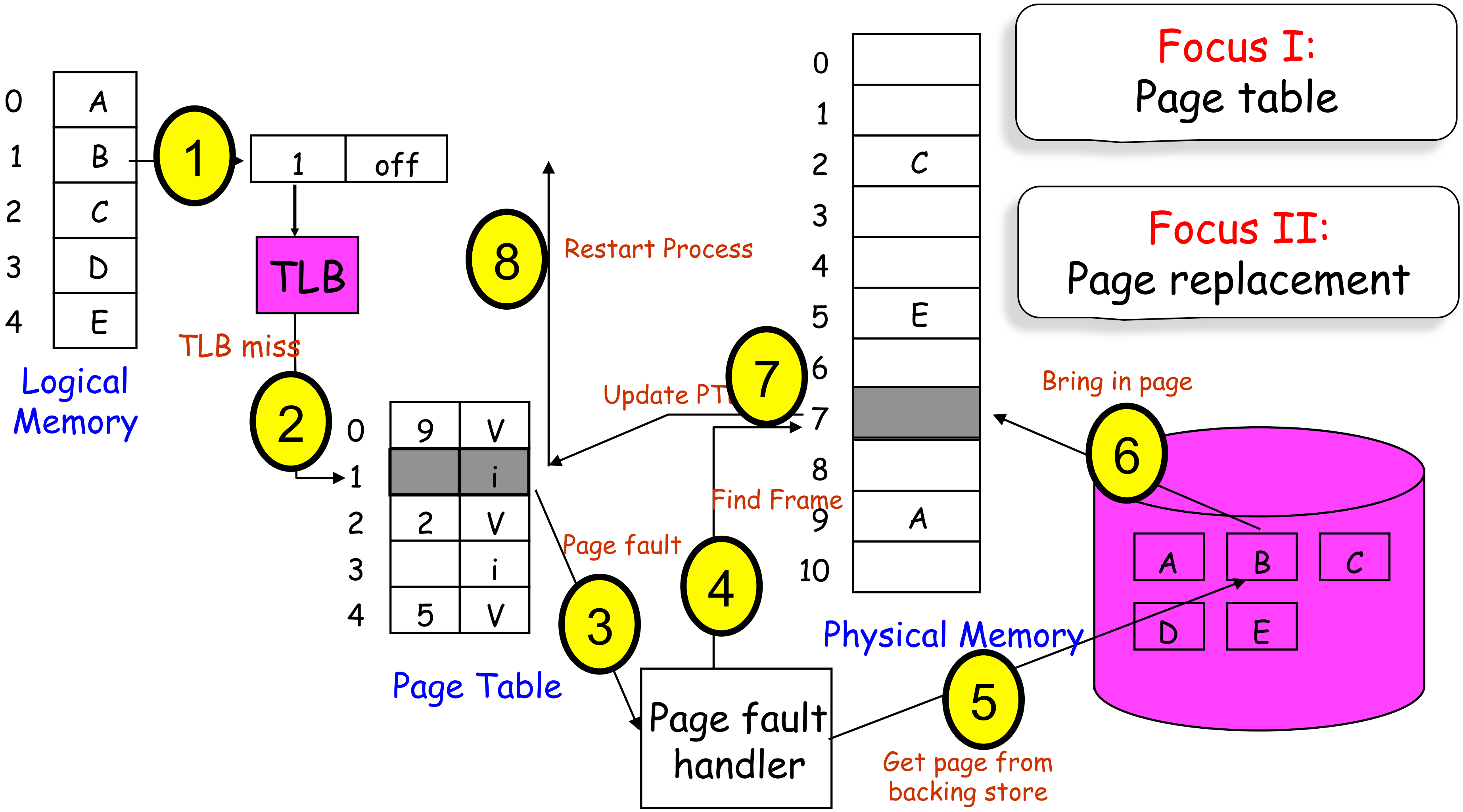
- Page is needed $\Rightarrow$ reference to it
  - invalid reference $\Rightarrow$ abort
  - not-in-memory $\Rightarrow$ bring to memory

Page Fault

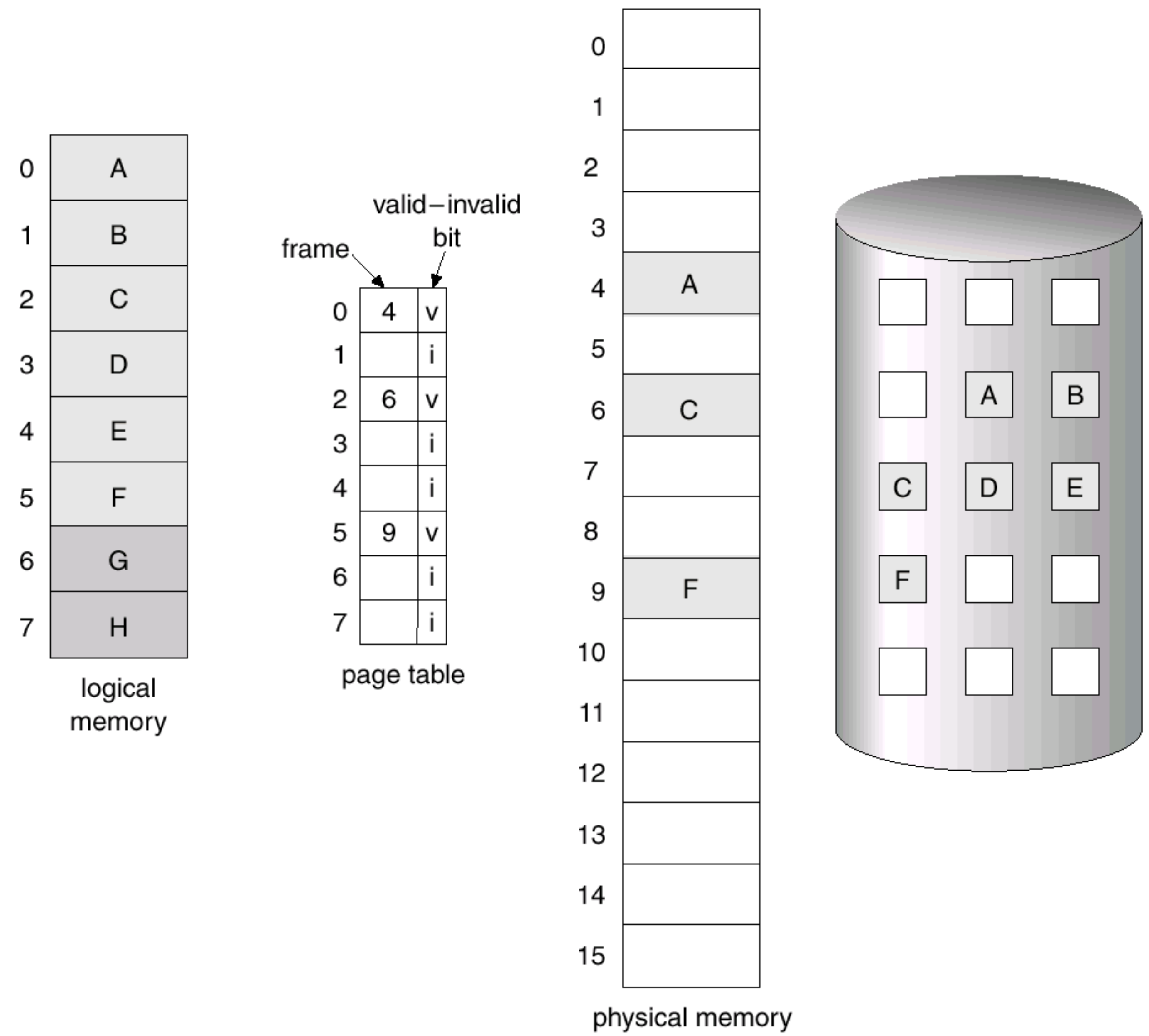
- If there is ever a reference to a page, first reference will trap to OS $\Rightarrow$ page fault

- Page fault handler looks at the cause and decide:
  - Invalid reference $\Rightarrow$ abort
  - Just not in memory
    - Get empty frame
    - Swap page into frame
    - Reset tables, valid bit = 1
    - Restart instruction

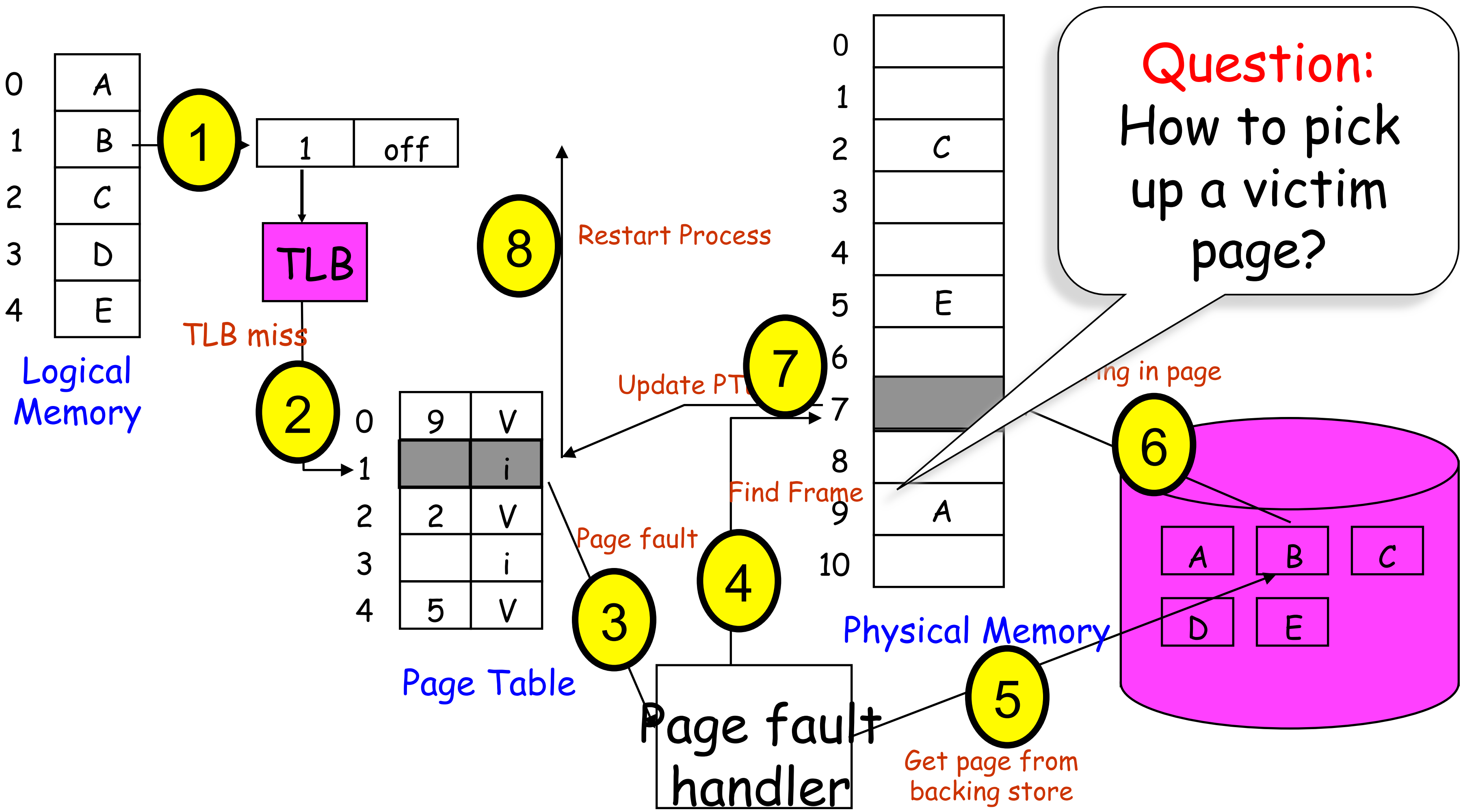
Steps in Handling a Page Fault



Page Table When Some Pages Are Not in Main Memory



Page Replacement



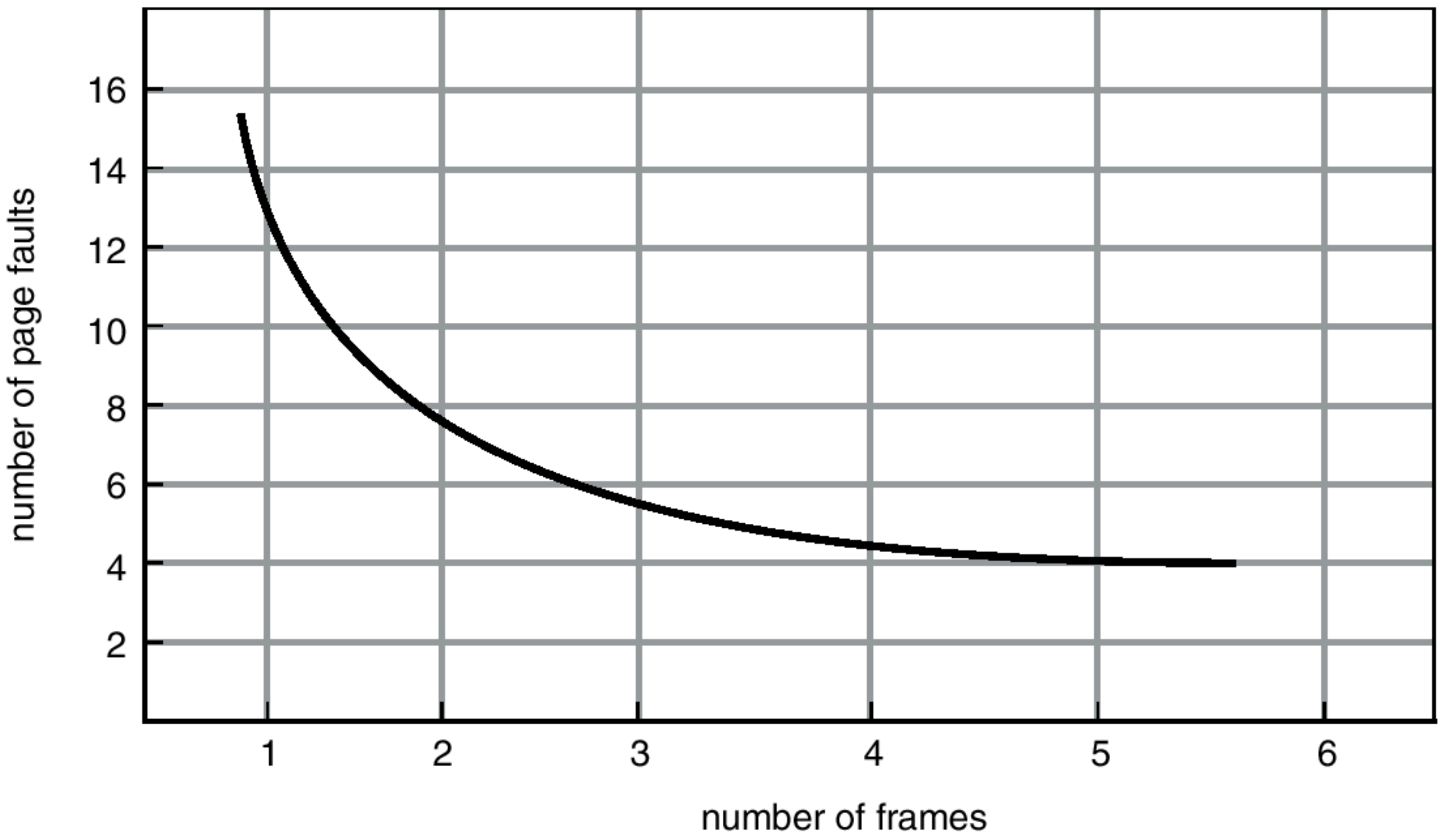
Page Replacement Algorithms

- So, when we have a page fault we have to find an eviction candidate.
- Optimally, we would like to **evict the page that will not be referenced again for the longest amount of time**.
  - In reality the OS has no way of knowing when each of the pages will be referenced next

Page Replacement Algorithms

- NRU
- FIFO
- FIFO w/ Second Chance
- Clock
- LRU
- NFU
- Aging
- Working set
- WSClock

Ideal Graph of Page Faults Versus The Number of Frames



Optimal Algorithm

- Replace page that will not be used for longest period of time in the future!
- 4 frames example

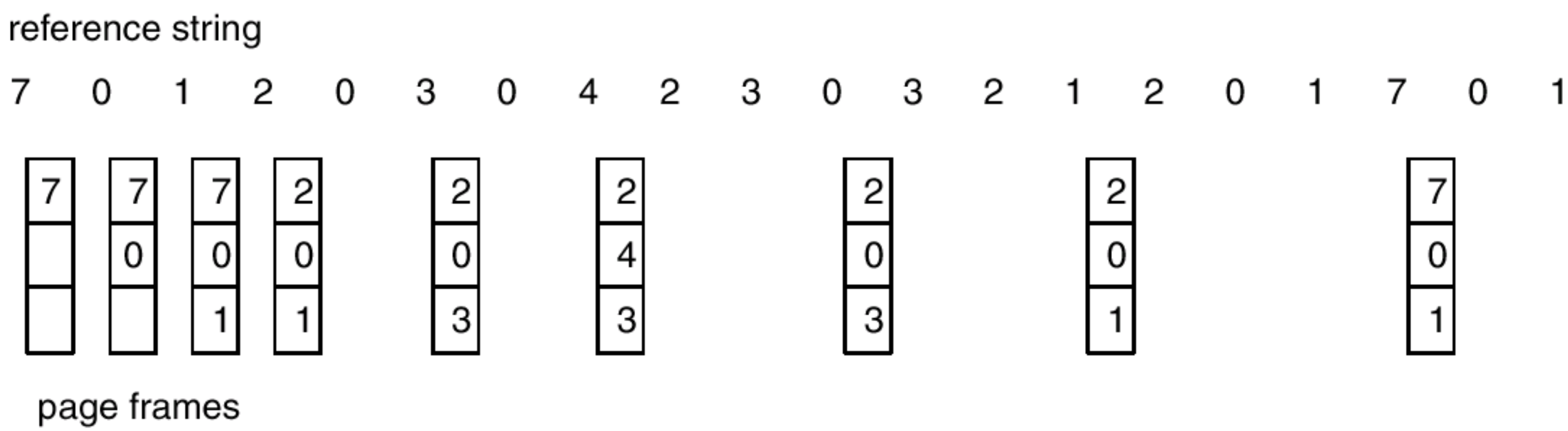
1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5

1	4
2	
3	
4	5

6 page faults

- How do you know this?
- Used for measuring how well your algorithm performs

Optimal Page Replacement



- 20 references
- 9 page faults

FIFO

- Simple design of having a queue maintained for pages in memory.
  - The head of the queue contains oldest page in memory.
  - The tail of the queue contains the newest page in memory.

First-In-First-Out (FIFO) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pages can be in memory at a time per process)

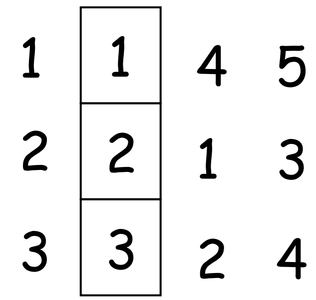
1	1	4	5
2	2	1	3
3	3	2	4

9 page faults

- 4 frames,
  - how many page faults?

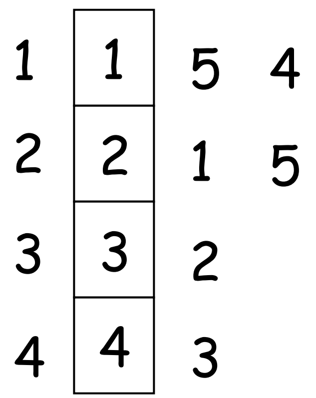
First-In-First-Out (FIFO) Algorithm

- Reference string: 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5
- 3 frames (3 pages can be in memory at a time per process)



9 page faults

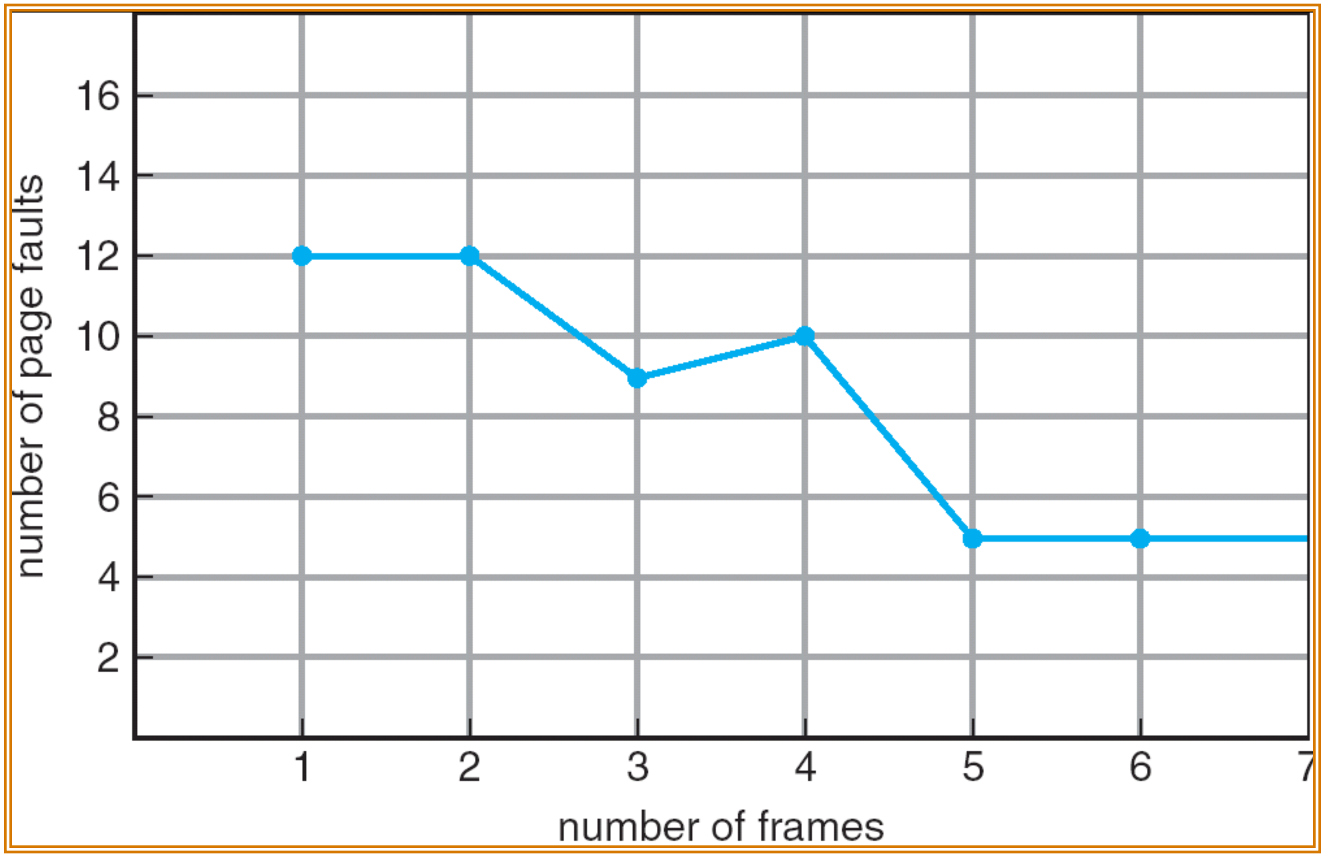
- 4 frames



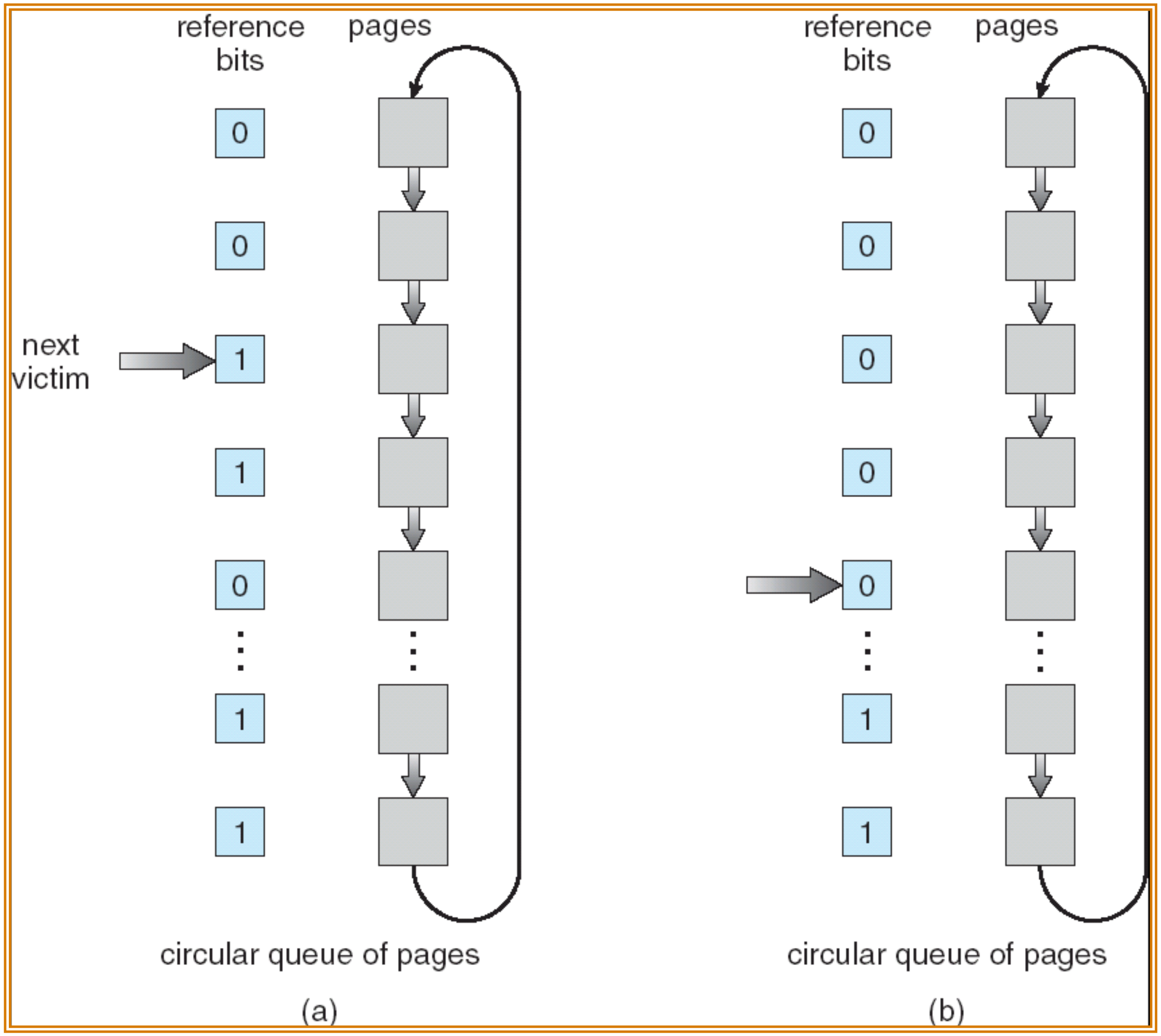
10 page faults

- Belady's Anomaly: more frames $\Rightarrow$ more page faults

FIFO Illustrating Belady's Anomaly



Clock Page-Replacement Algorithm



Least Recently Used (LRU)

- Idea: pages used recently will likely be used again soon
    - throw out page that has been unused for longest time
- Implementions
    - keep a linked list of pages
        - most recently used at front, least at rear
        - update this list <u>every memory reference</u> !!
    - Keep counter in each page table entry
        - write access time into counter
        - choose page with lowest value counter

Not Recently Used (NRU)

- Examine Modified (**M**) and Reference (**R**) bits associated with each page
- At each clock interrupt, the reference bits of all the pages are cleared
- Page fault occurs, OS places all pages in 1 or 4 classifications
  - Class 0: R=0, M=0
  - Class 1: R=0, M=1
  - Class 2: R=1, M=0
  - Class 3: R=1, M=1
- Remove a page at random from the lowest nonempty class.

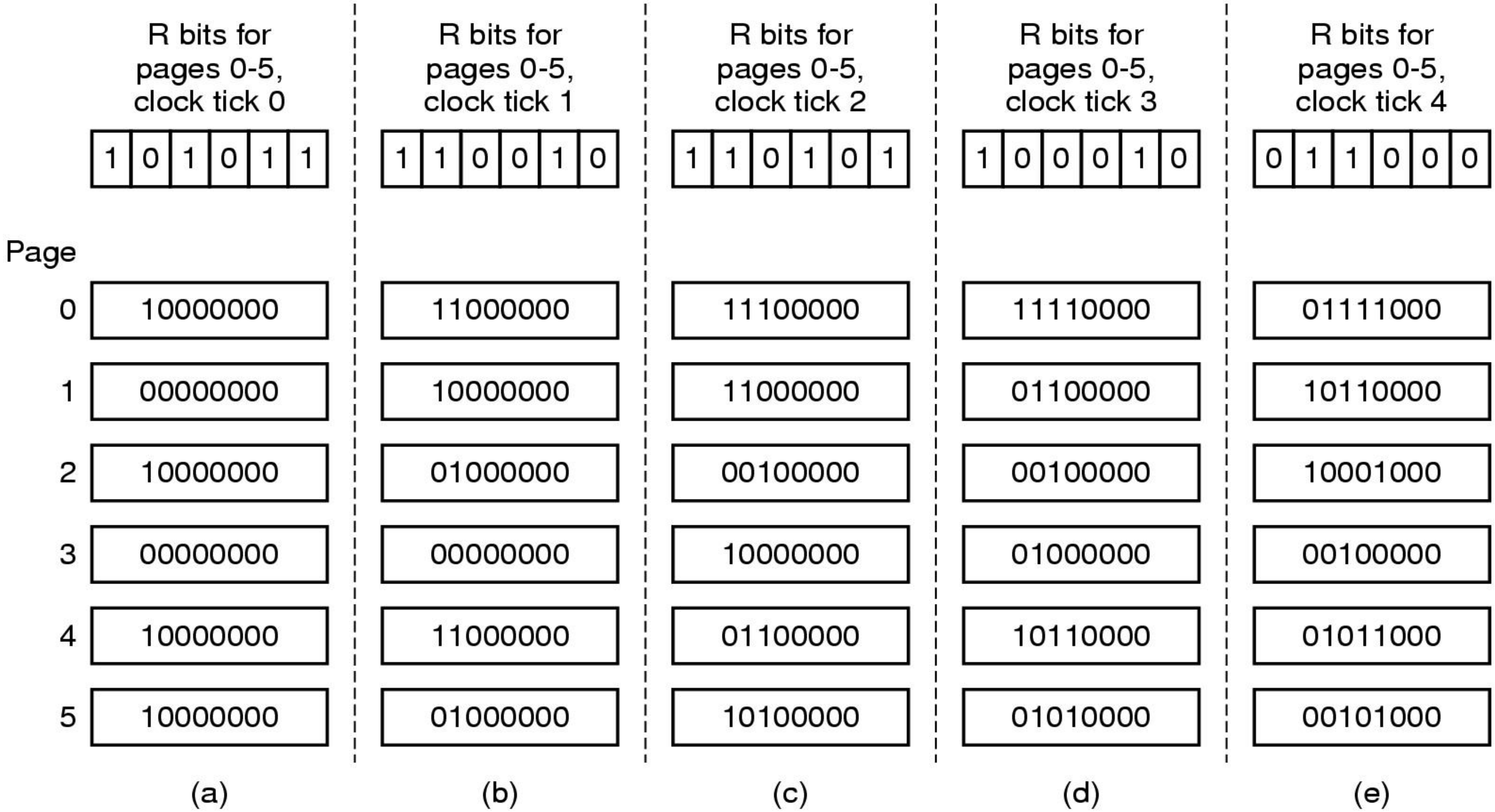
NFU (Not Frequently Used)

- Most machines do not have the hardware to perform true LRU, but it may be simulated.
- We can use a counter to keep track of the number of references for each page
- Page with the lowest frequency is evicted.

Aging

- Counters are each shifted right 1 bit before the R bit is added.
- R bit is added to the leftmost, rather than the rightmost bit.
- This modified algorithm is known as aging.

Aging


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

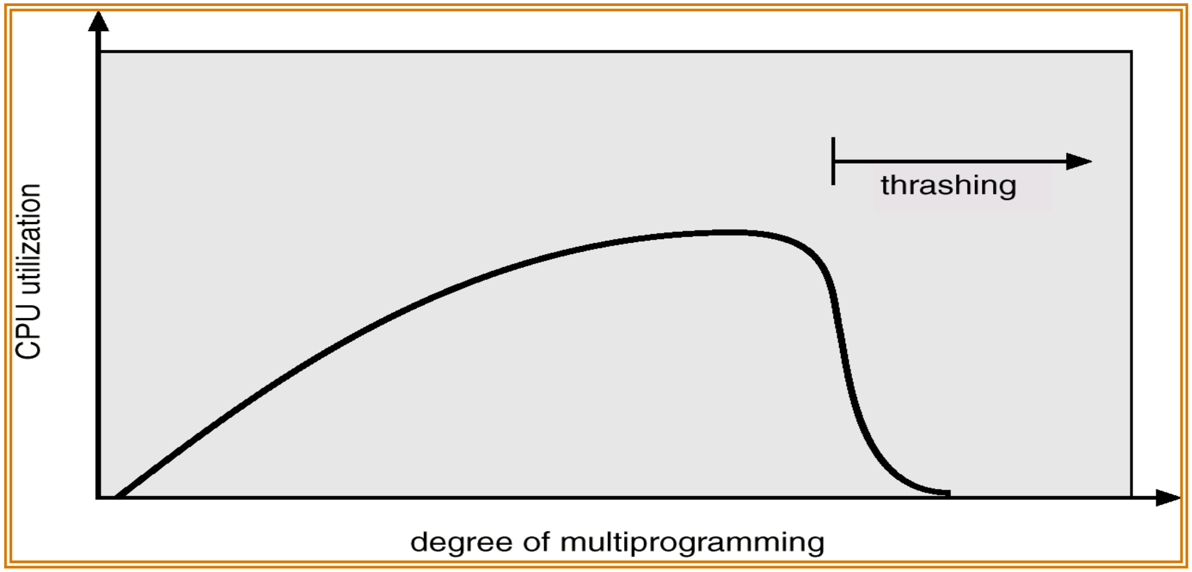
Working Set Page Replacement

- Locality of Reference – a process references only a small fraction of its pages during any particular phase of its execution.
- The set of pages that a process is currently using is called the working set.

Thrashing

- If a process does not have “enough” frames, the page-fault rate is very high.
- This leads to low CPU utilization
- Thrashing ≡ a process is busy swapping pages in and out

Thrashing



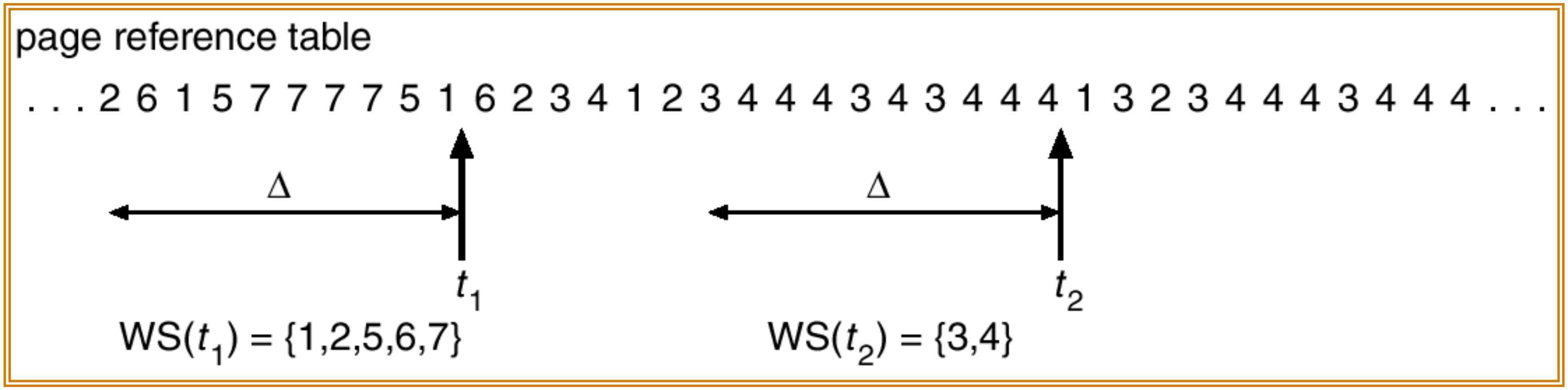
- Why does paging work?
  - Locality model
    - Process migrates from one locality to another
    - Localities may overlap
- Why does thrashing occur?
  Σ size of locality > total memory size

Working-Set Model

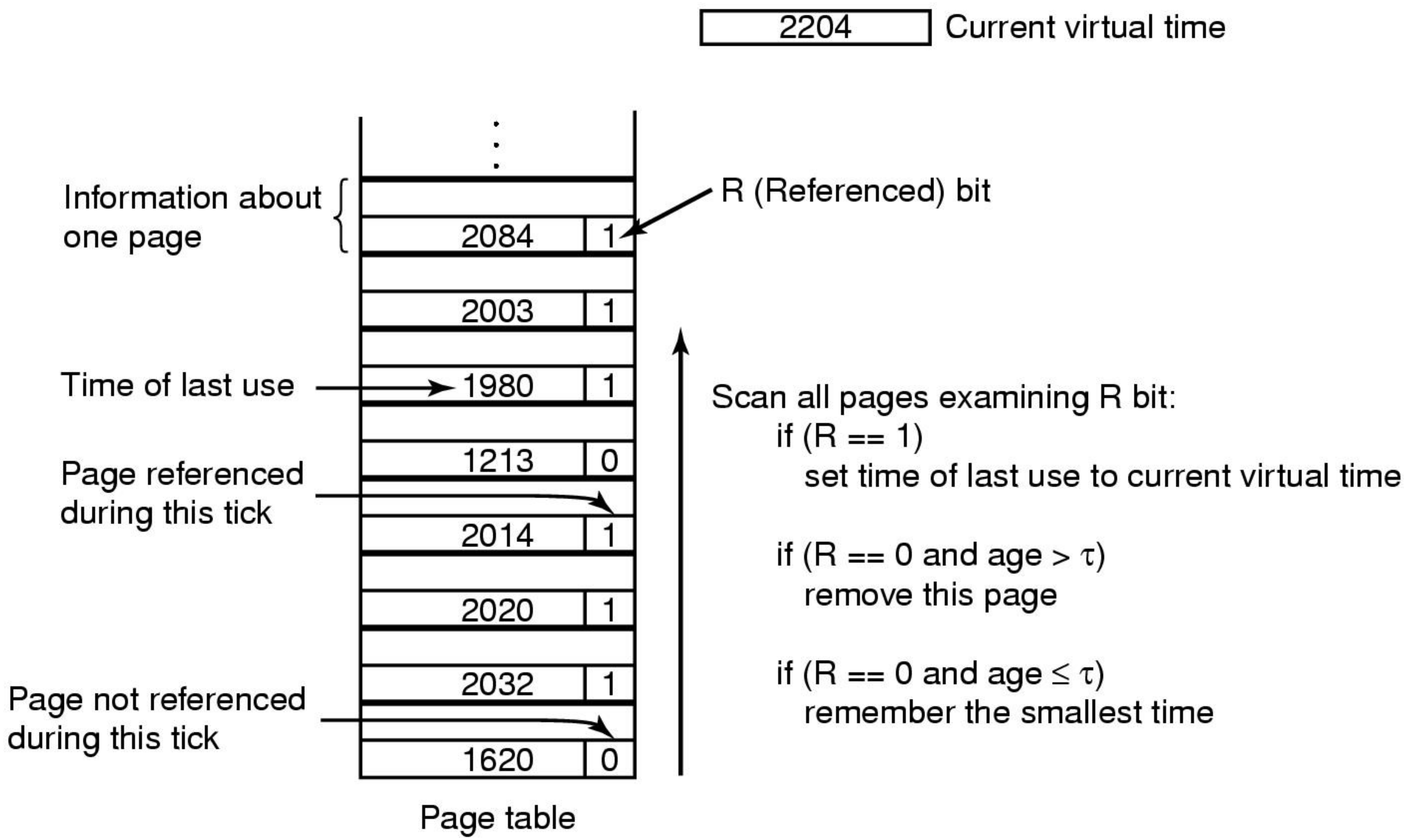
- $\Delta \equiv$ working-set window $\equiv$ a fixed number of page references
  Example: 10,000 instructions
- WS_i (working set of Process P_i) =
  total number of pages referenced in the most recent Δ
  - if Δ too small will not encompass entire locality
  - if Δ too large will encompass several localities
  - if $\Delta = \infty \Rightarrow$ will encompass entire program
- $D = \Sigma\ WS_i \equiv$ total demand frames
  - if $D > m \Rightarrow$ Thrashing
  - Policy if $D >$ m, then suspend one of the processes

Working Set Page Replacement (cont.)

- The working set algorithm is based on determining a working set and evicting any page that is not in the current working set upon a page fault.



Working Set Page Replacement (cont.)



Working Set Page Replacement (cont.)

- So, what happens in a multiprogramming environment as processes are switched in and out of memory?
  - Do we have to take a lot of page faults when the process is first started?
  - It would be nice to have a particular processes working set loaded into memory before it even begins execution. This is called **prepaging**.

Working Set Page Replacement (cont.)

- What happens when there is more than one page with R=0?
- What happens when all pages have R=1?
- This algorithm requires the entire page table be scanned at each page fault until a suitable candidate is located.
  - All entries must have their Time of last use updated even after a suitable entry is found.

WSClock Page Replacement (Cont.)

- What happens when R=0?
  - Is age > τ, and page is clean then it is evicted
    - If it is dirty then we can proceed to find a page that may be clean. We will still need to write to disk and this write is scheduled.
    - No clean page old enough? Evict a dirty one.
- No old enough pages?
  - Evict the oldest page, clean or dirty
- Page replacement separated from dirty page writing

Page Replacement Algorithm Summary

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

PA3 : Demand Paging

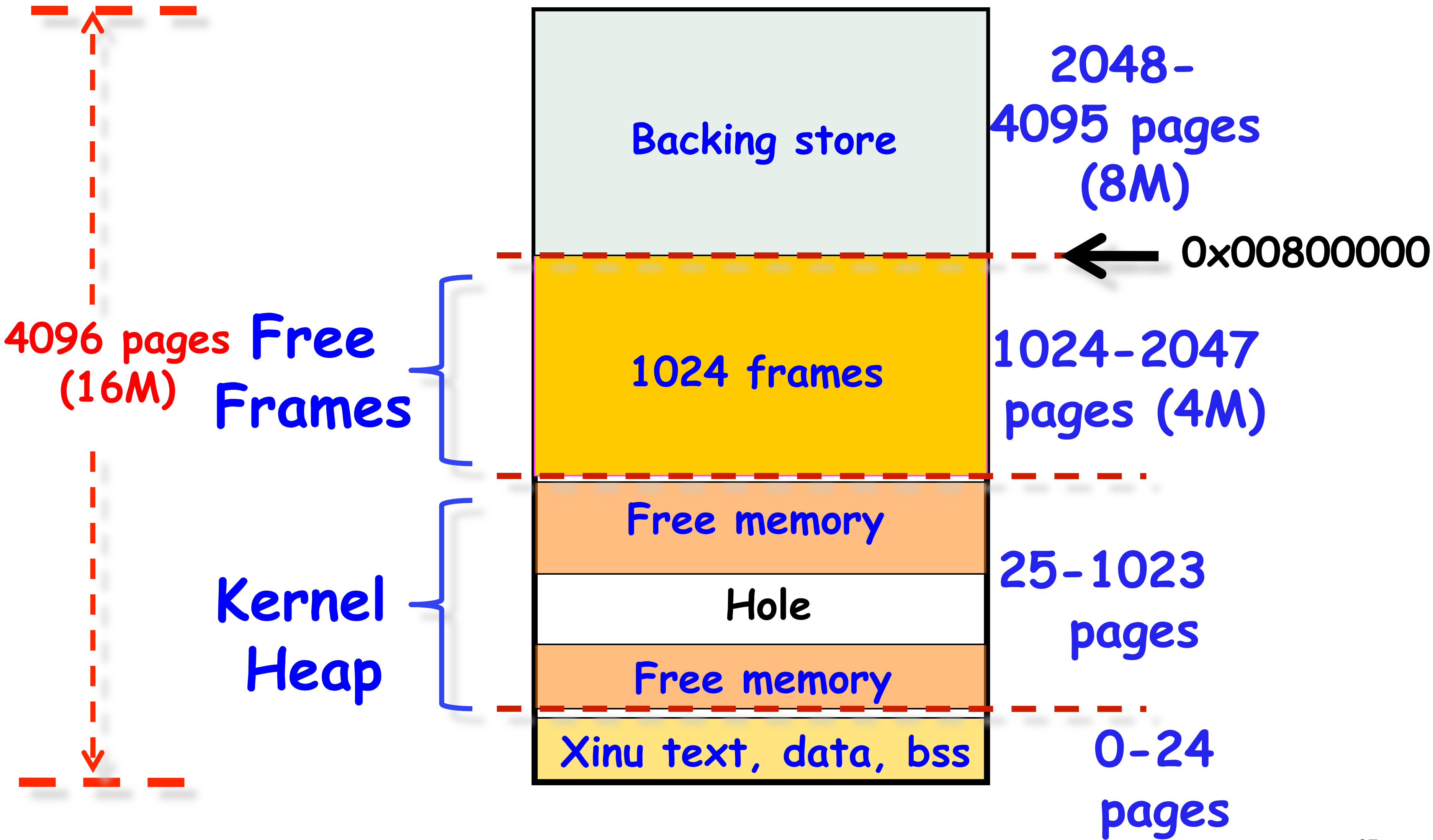
Here, the disk refers to the backing store!

- From the OS perspective:
  - Pages are evicted to disk when memory is full
  - Pages loaded from disk when referenced again
  - References to evicted pages cause a page table miss
    - Page table entry (PTE) was invalid, causes fault
  - OS allocates a page frame, reads page from disk
  - When I/O completes, the OS fills in PTE, marks it valid, and restarts faulting process
- Dirty vs. clean pages
  - Actually, only dirty pages need to be written to disk
  - Clean pages do not – but you need to know where on disk to read them from again

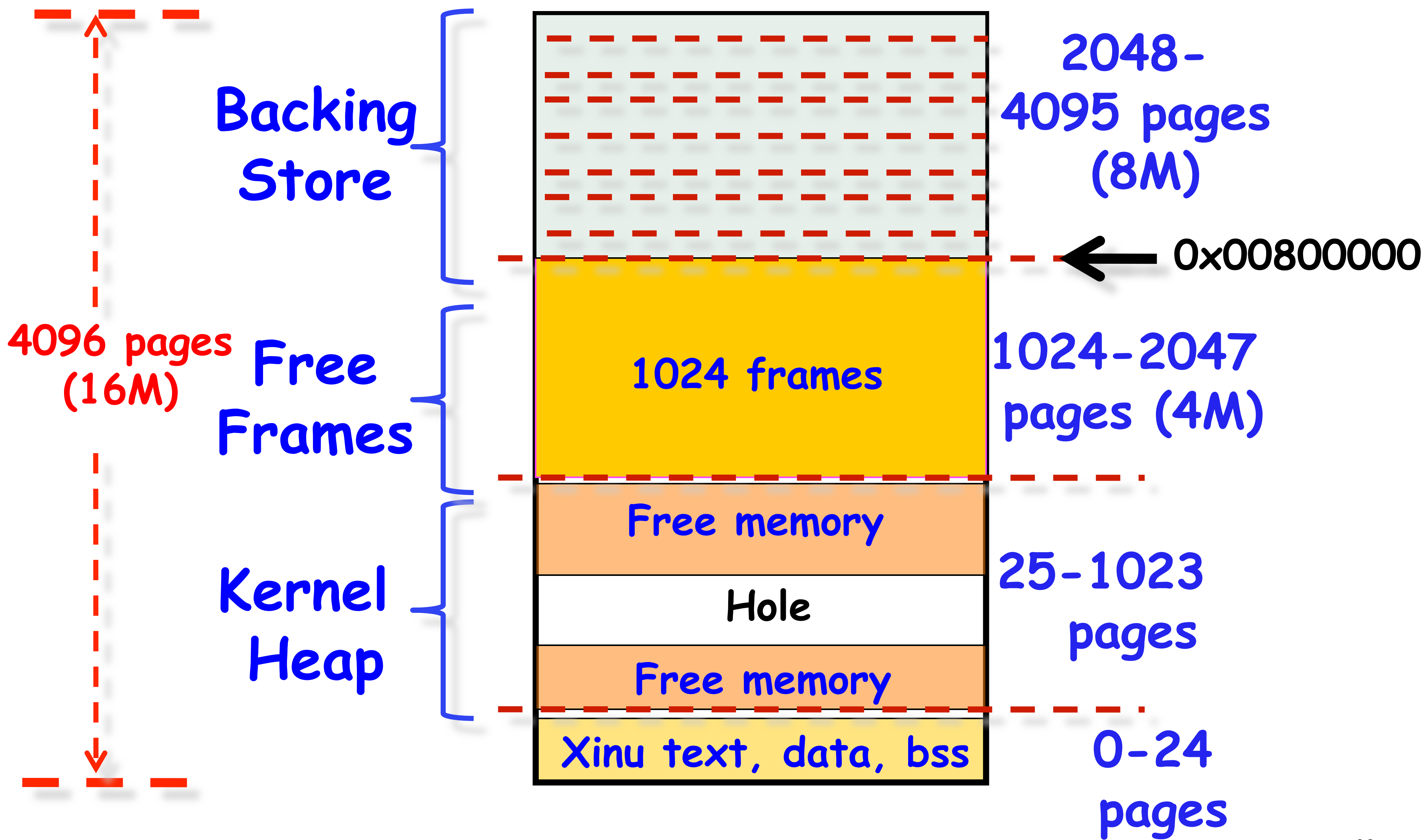
PA 3 : Demand Paging

- From the process perspective:
  - Demand paging is also used when it first starts up
  - When a process is created, it has
    - A brand new page table with all valid bits off
    - No pages in memory
  - When the process starts executing
    - Instructions fault on code and data pages
    - Faulting stops when all necessary code and data pages are in memory
    - Only code and data needed by a process needs to be loaded, which will change over time…
  - When the process terminates
    - All related pages reclaimed back to OS

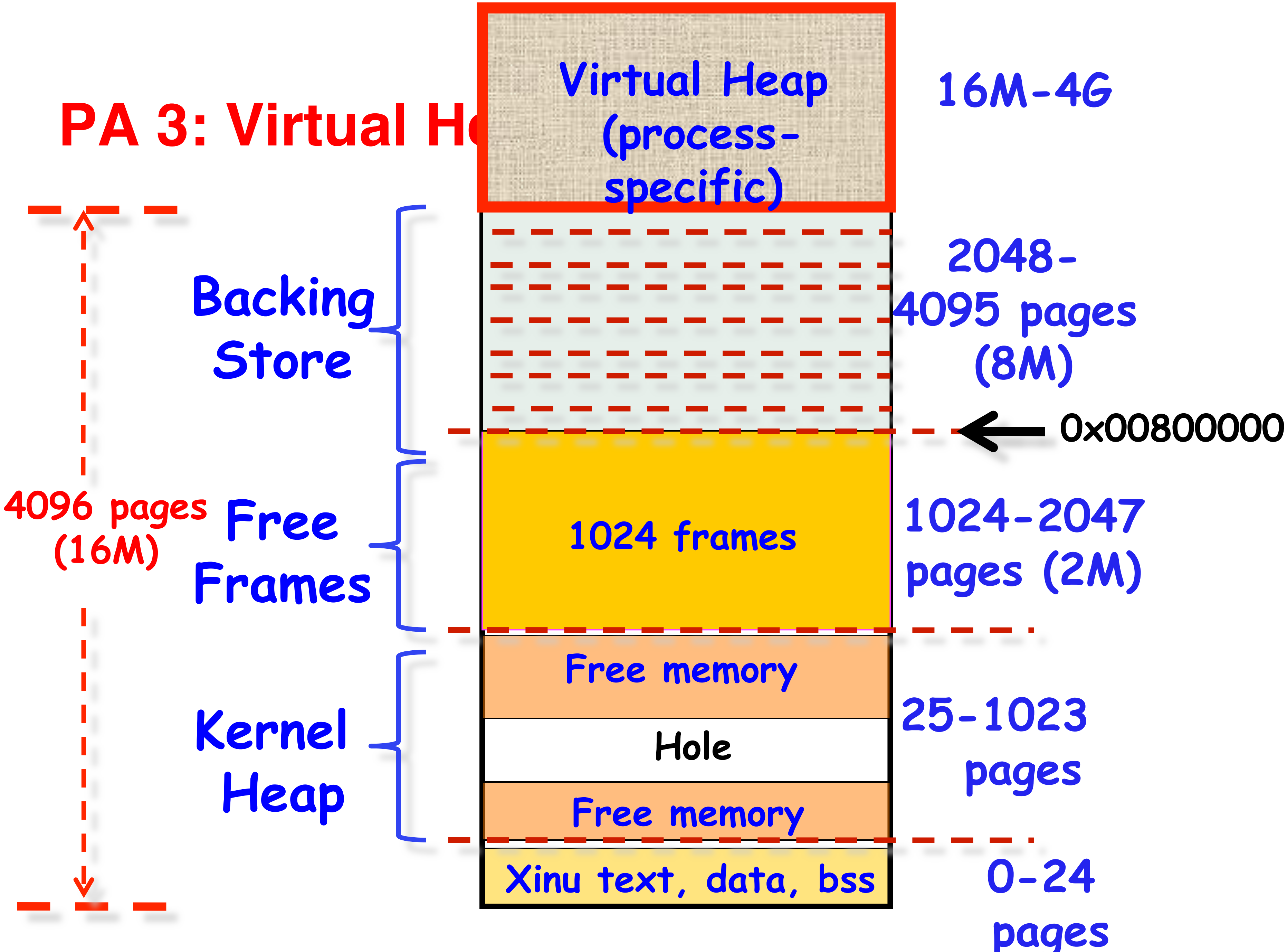
PA 3: Physical Memory Layout



PA 3: Backing Store



PA 3: Virtual He



PA 3: Backing Stores

- There are 16 backing stores in total:
  - APIs: get_bs/release_bs, read_bs/write_bs
  - Emulated by physical memory
  - Skeleton already given
    - You may want to add some sanity check!

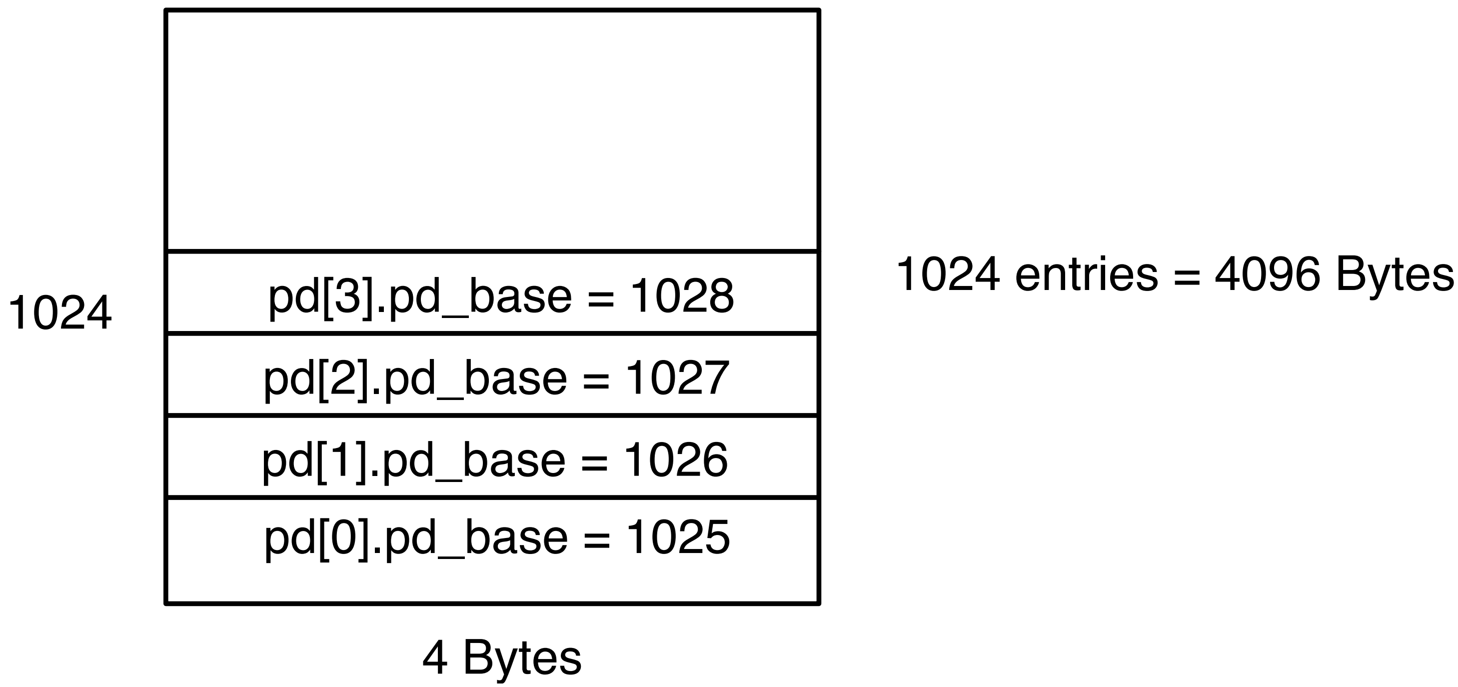
PA 3: Other Issues

- The NULL process
  - No private heap
- Global page table entries
  - The entire 16M physical memory
  - Identity mapping
- Page fault ISR
  - set_evec(int interrupt, (void (*isr)(void)))
- Support data structures
  - Inverted page table
  - Help functions
    - E.g., finding a backing store from a virtual address

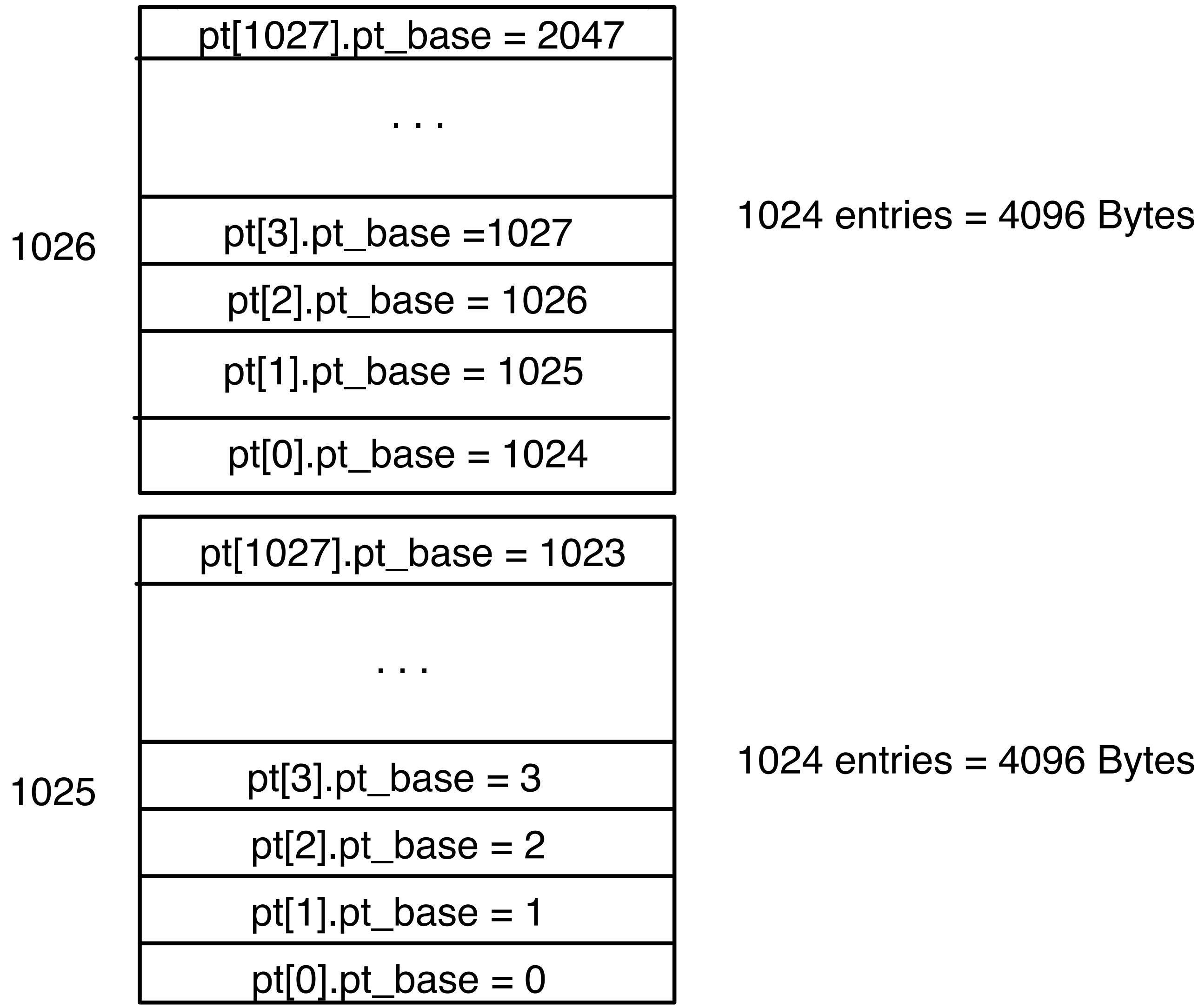
To be extended with your own page replacement algorithm

PA3: Page Directory for Null Process

Page Table Number	Page Number	Offset
31-22	21-12	11-0



PA3: Page Tables for Null Process (1)



PA3: Page Tables for Null Process (2)

1028

pt[1027].pt_base = 4095	1024 entries = 4096 Bytes
. . .	
pt[3].pt_base = 3075	
pt[2].pt_base = 3074	
pt[1].pt_base = 3073	
pt[0].pt_base = 3072	

1027

pt[1027].pt_base = 3071	1024 entries = 4096 Bytes
. . .	
pt[3].pt_base = 2051	
pt[2].pt_base = 2050	
pt[1].pt_base = 2049	
pt[0].pt_ase = 2048	

PA 3 : Demand Paging

- Goal: Be familiar with VM & Demand Paging
- You are asked to implement the following syscalls
  - xmmap, xmunmap, vcreate, vgetmem/vfreemem, srpolicy
  - It is a tough PA. Start NOW!