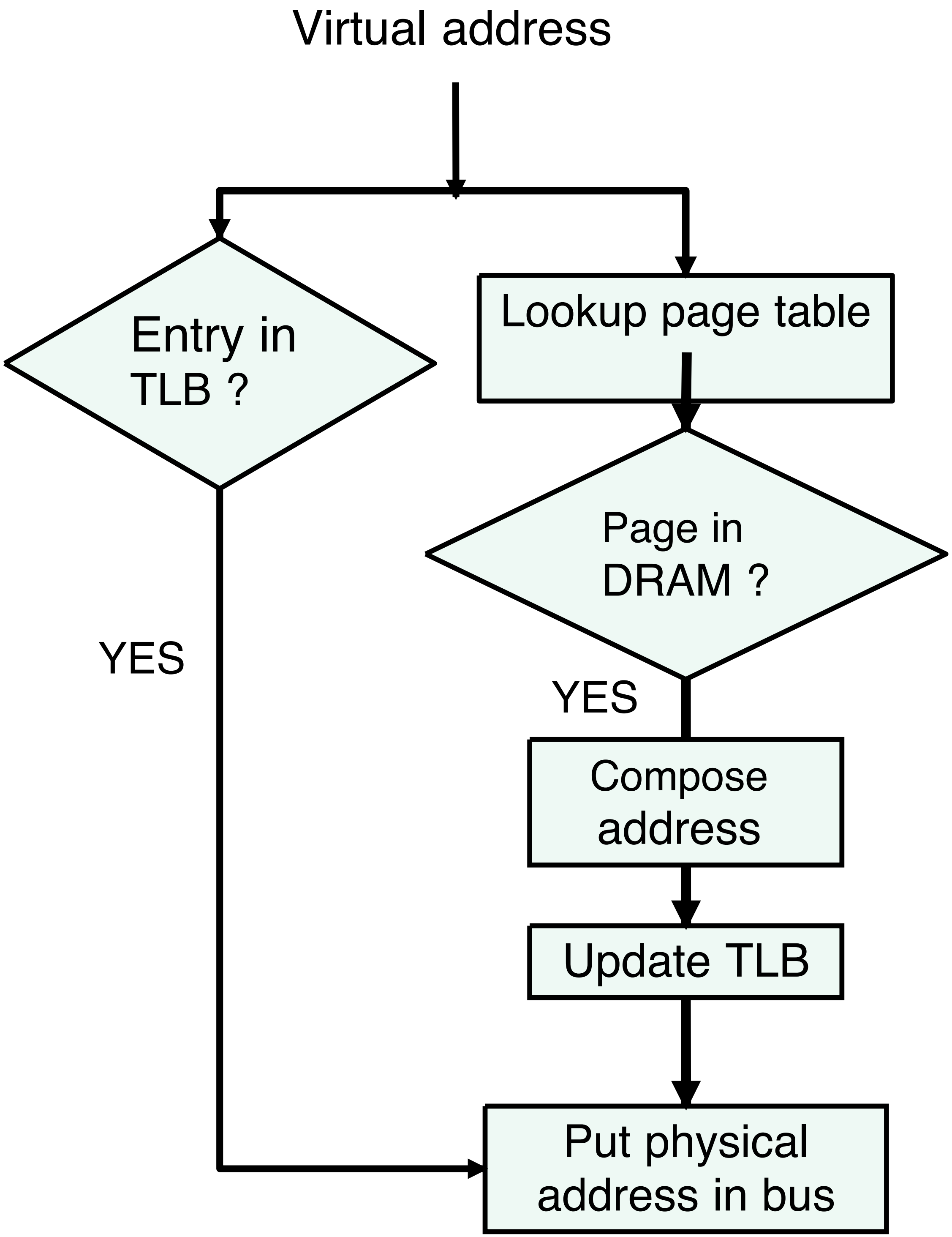
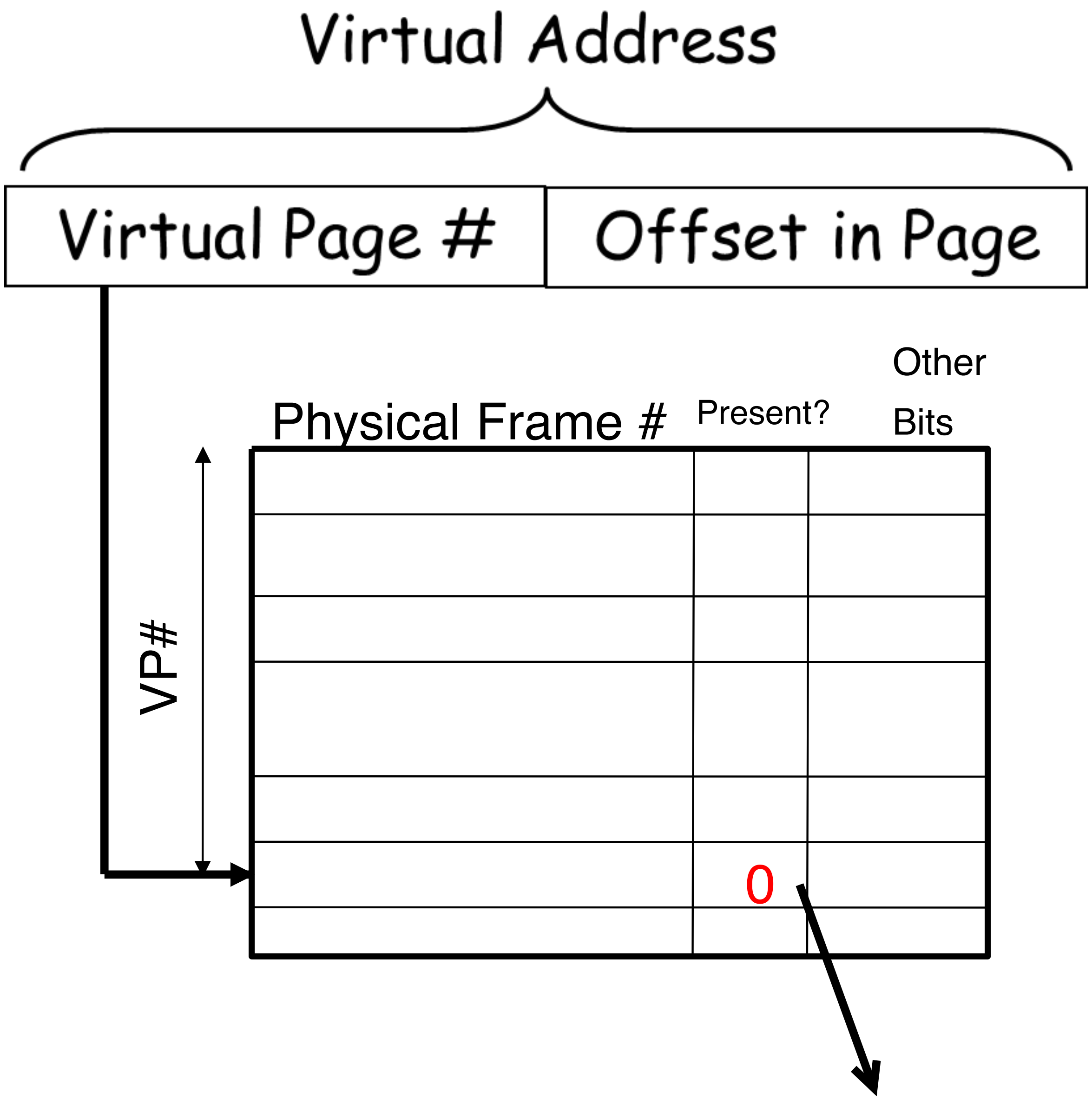
Virtual Memory – 2

(Paging)


Virtual address
Entry in TLB ?
Lookup page table
Page in DRAM ?
YES
YES
Compose address
Update TLB
Put physical address in bus

Page Fault



Interrupt Service Handler

PageFaultHandler() {

....

}

Page Fault! - Hardware Interrupt

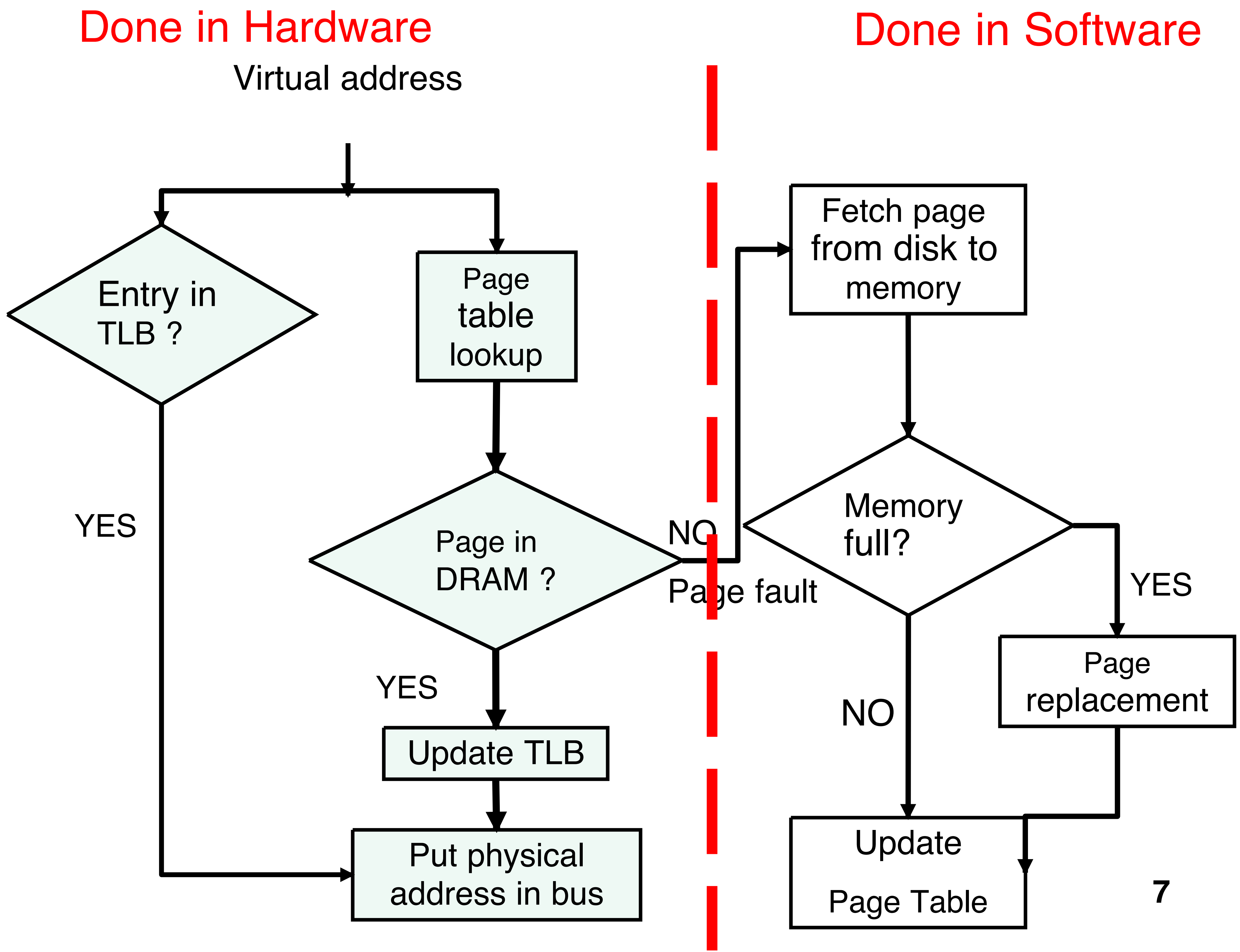
(H/w passes on control to S/w)

Page-Faults

- **If a Page-table mapping indicates an absence of the page in physical memory, hardware raises a "Page Fault".** — OS deal with page fault
- **OS traps this fault and the interrupt handler services the fault by initiating a disk-read request.**
- **Once page is brought in from disk to main memory, page-table entry is updated and the process which faulted is restarted.**
  - **May involve replacing another page and invalidating the corresponding page-table entry.**

Physical Memory is limited!

- When bringing in a page, something has to be evicted.
- What should we evict? – page replacement algorithm.



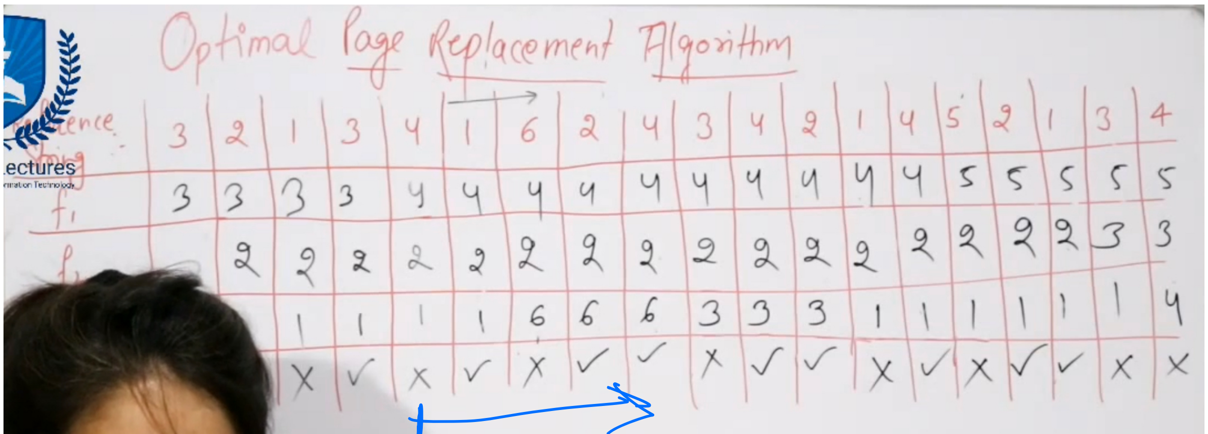
Done in Hardware
Done in Software
Virtual address
Entry in TLB ?
Page table lookup
YES
Page in DRAM ?
NO
Page fault
YES
Update TLB
Put physical address in bus
Fetch page from disk to memory
Memory full?
YES
Page replacement
NO
Update Page Table

Optimal Page Replacement Algorithm

Find future

- **Why optimal?**
  - No other algorithm can have # of page faults lower than this, for a given page reference stream.
- **Algorithm:**
  - At any point, amongst the given pages in memory, evict the one whose first reference from now is the furthest.

go to future



page replace, look at future to find the number that is furthest from now. in this case, 3 is the furthest. so replace 3.

An example of OPT

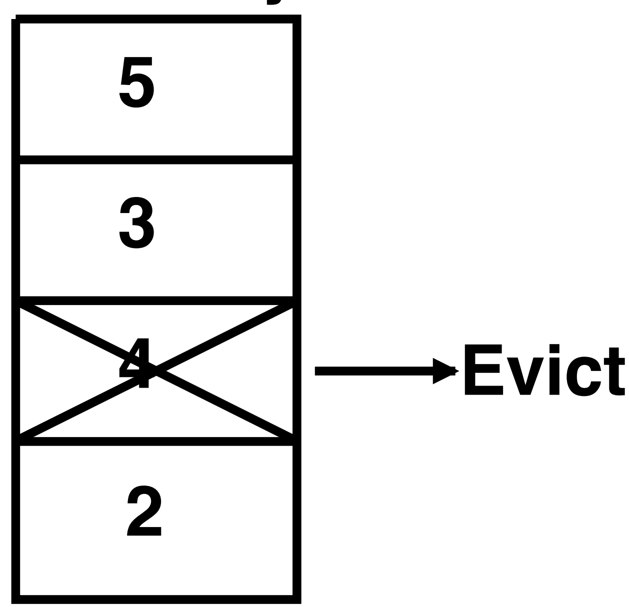
Reference String

…… 5, 3, 3, 5, 2, 4, 4, 3, 2, …..

At this point,

what do we replace?

Current Physical Memory

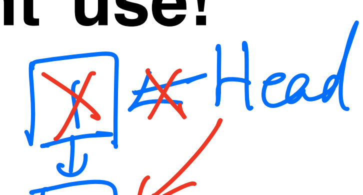


Problem with OPT

- Not implementable!
- Requires us to know the future.
- But it has the best page fault behavior
- How do we approach OPT?

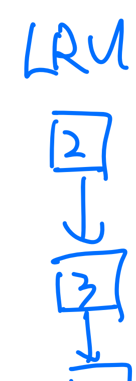
1. First-in First-out

- Maintain a linked list of pages in the order they were brought into physical memory.
- On a page fault, evict the one at the head.
- Put the newly brought in page (from disk) at tail of this list.
- Problems:
  - Reference String: 1,2,3,4,1,1,5,1,1,…
  - Page fault at (5) would replace (1) !
  - Need to know what is in recent use!



2. Least Recently Used

- Order the list of physical memory pages in decreasing order of recency of usage.
- Replace the page at the tail.
- Problem:
  - This list will need to be updated on each memory reference.
  - Asking the h/w to do this is ridiculous!
- Solution: Approximate LRU



3. Not Recently Used

- Referenced bit set on each Read/write by h/w
- Modified set on each write by h/w
- On startup set both R and M bits to 0.
- Periodically (using clock interrupts) the R bit is cleared. (but do not clear out modified bits).

- On a page fault, examine the state of a page
  - Class 0: R = M = 0
  - Class 1: R = 0, M = 1
  - Class 2: R = 1 M = 0
  - Class 3: R = 1 M = 1

- NRU replaces a page chosen at random from the lowest numbered nonempty class.

P_F #1 0 0 0 0 0 0
: 1 0 0 0 0 0

ref=1: 1 0 0 0 1 0
PF5: 1 0 0 0 1 0

Q1: Are all those steps in one pf.
Q2: what if multiple

PF-1: …
PF-3: 0 0 1 0 0 0
PF4: 0 0 0 1 0 0

ref to one page frame before page fault?

4. Approximate LRU using counters

when page fault, replace the smallest value of counter bits.

- Keep a counter for each Phys page.
- Initially set to 0.
- At the end of each time interval (interval to be determined), shift the bits right by one position.
- Copy the reference bit to the MSB of counter and reset reference.

- For a page replacement, pick the one with the lowest counter value.

- It is an approximation of LRU because:
  - we do not differentiate between references that occured in the same tick.
  - the history is limited by the size of the counter.

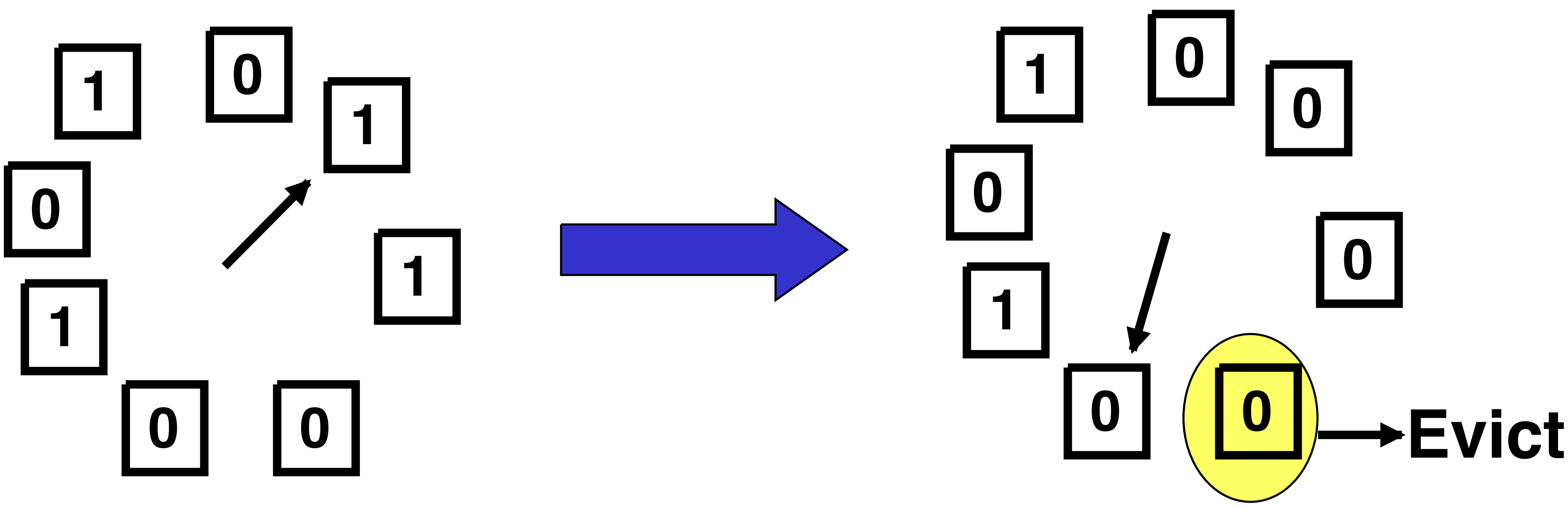
8 bit counter

	0	0	0	0	0	0	0	0
start	1	0	0	0	0	0	0	0
curt 1	0	1	0	0	0	0	0	0
curt 2	0	0	1	0	0	0	0	0

curt 0 0 0 1 0 0 0 0
ref 1 0 0 0 1 0 0 0
curt 0 1 0 0 0 1 0 0

5. Second Chance Replacement or Clock Algorithm

- Same as FIFO, except you skip over the pages whose reference bit is set, resetting this bit, and moving those pages to end of list.
- Implementation:



Summary of page replacement algorithms

- OPT, FIFO, NRU, second-chance/clock, LRU, approximate LRU

- In practice, OSes use second chance/clock or some variations of it.

Belady's Anamoly

- Normally you expect number of page faults to decrease as you increase physical memory size.

- However, this may not be the case in certain replacement algorithms

Example of Belady's Anamoly

- FIFO replacement Algorithm
- Refernce string:

  0 1 2 3 0 1 4 0 1 2 3 4
- 3 physical frames

  F F F F F F F - - F F -

  # of faults = 9
- 4 physical frames

  F F F F - - F F F F F F

  # of faults = 10

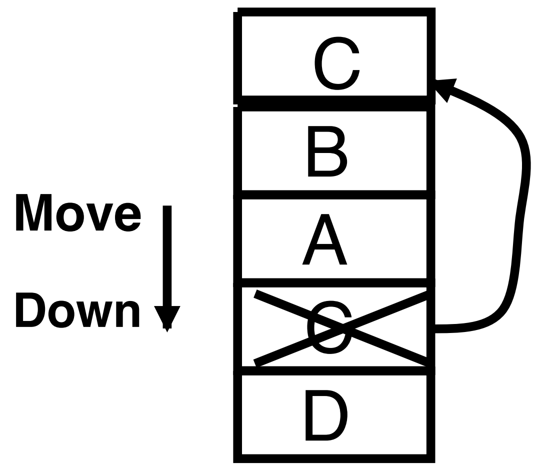
- **Algorithms which do NOT suffer from Belady's anamoly are called stack algorithms**
- **E.g. OPT, LRU.**

Modeling Paging

- Paging behavior characterized by
  - Reference string
  - Physical memory size
  - Replacement algorithm

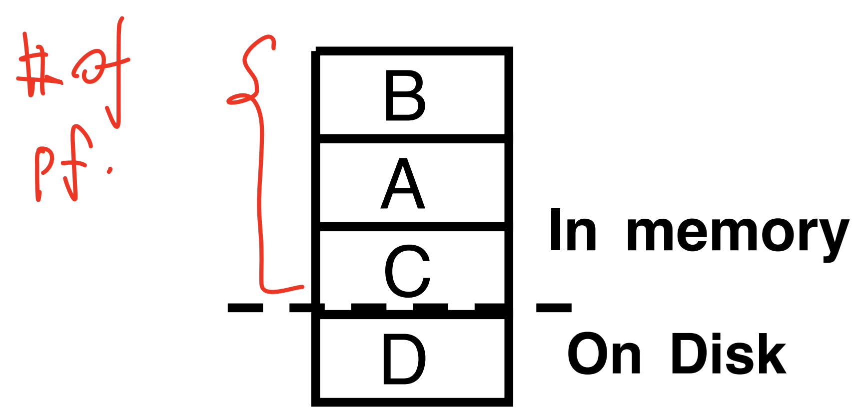
- **Visualize it as a stack (say *M*), where a page that is "referenced" is brought to the top of the stack from wherever it is.**

e.g. A, B, C and D are virtual pages



When C is referenced …

- **Whatever is in recent use is on the top of M, and the ones that are not in recent use are at the bottom.**
- **In fact, the top P entries of M represent the pages in physical memory, where P is the # of physical frames.**



- **Distance String:**
  - **For each element of reference string, this represents the distance of that element from the top of stack in M.**

An example of how M changes with 5 virtual pages

Reference String

A	B	C	D	A	B	E	A	B	C	D	E
A	B	C	D	A	B	E	A	B	C	D	E
	A	B	C	D	A	B	E	A	B	C	D
		A	B	C	D	A	B	E	A	B	C
			A	B	C	D	D	D	E	A	B
						C	C	C	D	E	A
∞	∞	∞	∞	3	3	∞	2	2	4	4	4

Distance String

Define vector C

- C[i] represents the number of times "i" appears in the distance string.

Reference String

A	B	C	D	A	B	E	A	B	C	D	E
A	B	C	D	A	B	E	A	B	C	D	E
	A	B	C	D	A	B	E	A	B	C	D
		A	B	C	D	A	B	E	A	B	C
			A	B	C	D	D	D	E	A	B
						C	C	C	D	E	A
∞	∞	∞	∞	3	3	∞	2	2	4	4	4

Distance String

\# of cases

C vector: C[0]=0, C[1]=0, C[2]=2,

C[3]=2, C[4]=3, C[5] …=0

C[∞]=5

Define Vector F

- F[j] is the number of page faults that will occur for the given reference string with "j" physical frames.

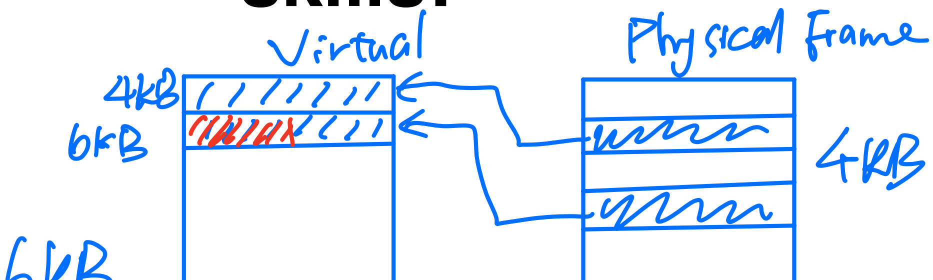
top "j" in M

- F[j] = C[j] + C[j+1] + C[j+2] + C[j+3] … + C[∞]

- **It is now straightforward to prove LRU does not suffer from Belady's anamoly.**
  - **The M vector tracks what is in physical memory in the top P slots for LRU.**
  - **Note that vector C[i] is independent of physical memory size.**
  - **When you go from physical memory with j frames to (j+x) frames, note that the number of C vector terms in the RHS of equation for F decreases => Page faults can only decrease if at all!**

Paging Issues

- Keep the essentials of what you currently need (working set) in physical memory.
- When something you need is not in memory, bring it in from disk:
  - On demand (demand-paging)
  - Ahead of need (pre-paging)
- Programs need to exhibit good locality to avoid "thrashing" of pages in memory.
- This usually requires good programming skills!



Fragmentation in paging

- Virtual Memory allows "non-contiguous" allocation of physical memory without requiring any programming changes.
- Note that there is only internal fragmentation of, and that too only in the last allocated page.
- Smaller the page, smaller the internal fragmentation.
- However, this reduces spatial locality.

(sbrk)

(1) process → OS → Promise

(2) process → OS → frame

Putting it all together!

Global - Data segment

int A[~~2~~8K];

int B[~~2~~8k];

Local - stack

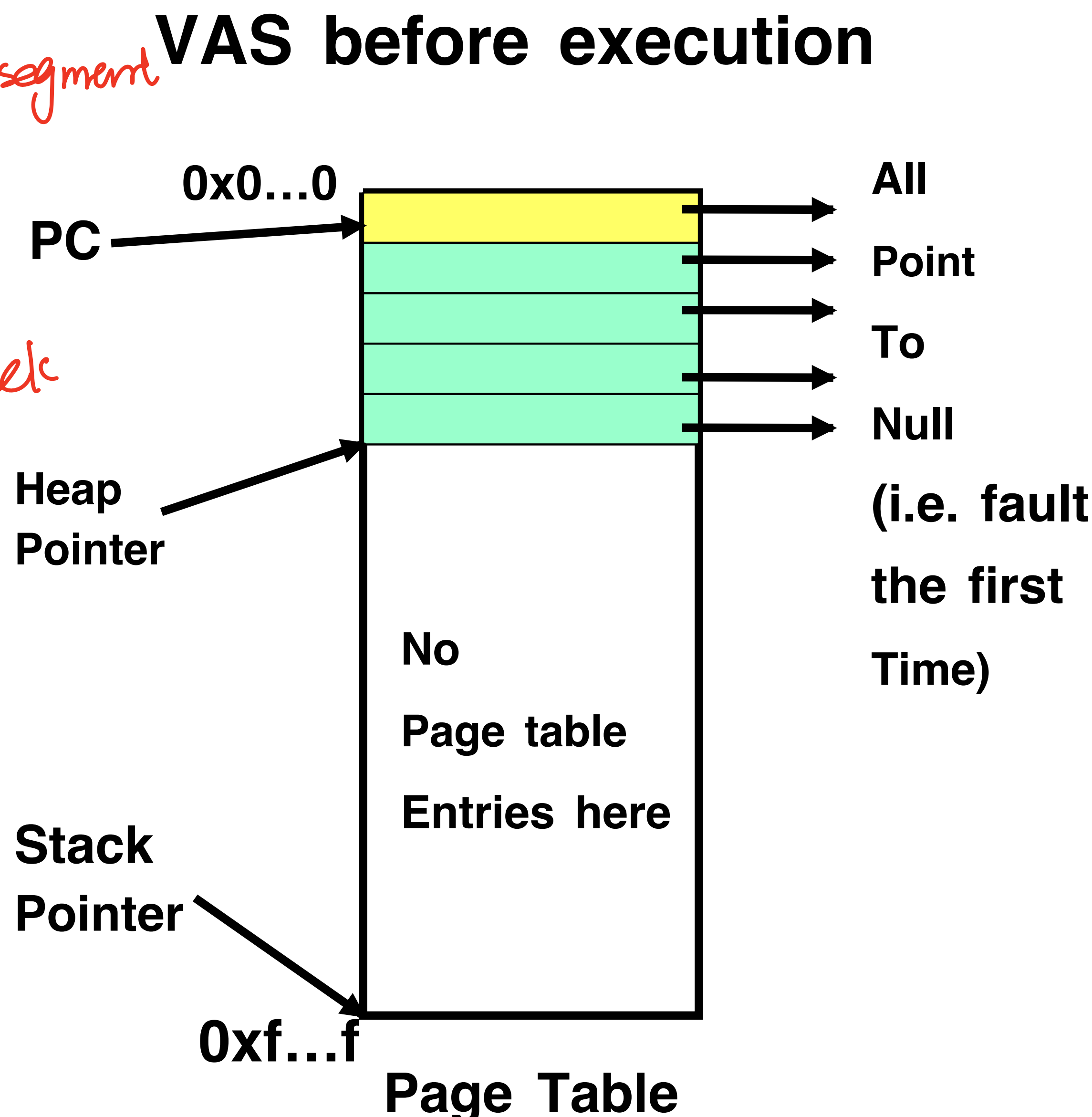
main() {

int i, j, p;

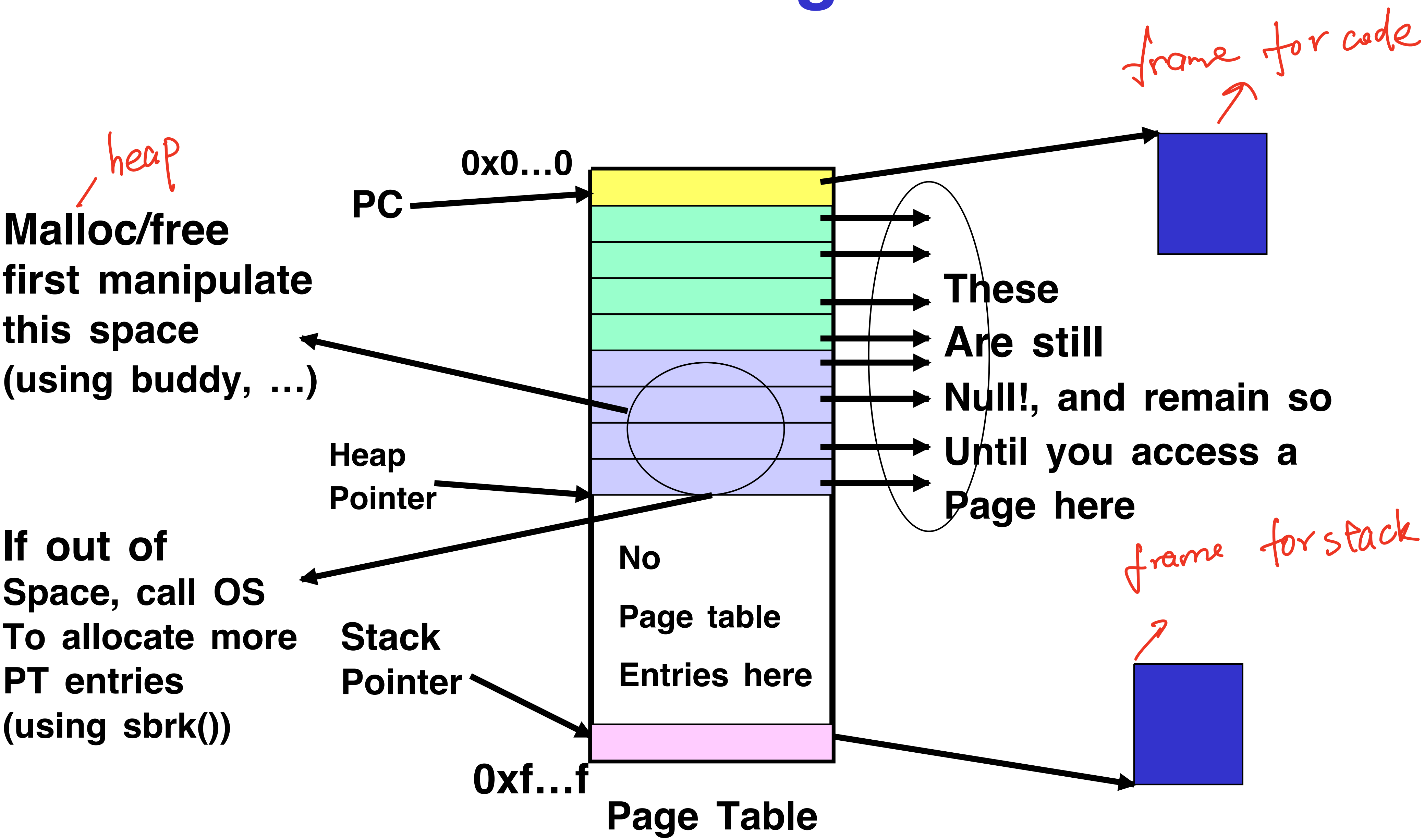
p = malloc(16K);

}

4K page size



After executing malloc



Note: you are not allocating physical memory using malloc()

Hardware causing OS to involve is call "interupt"
User side causing OS to involve is call "trap"