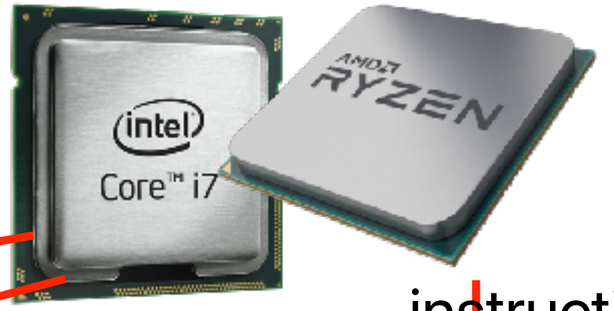
Virtual Memory (III) — Policies

Hung-Wei Tseng

Recap: Demand paging



instruction
0x0

data
0x80000000

Page fault!

Virtual Address Space for Chrome

Page fault!

instruction
0x0

Page fault!

Virtual Address Space for Apple Music

data
0x80008000

Page fault!

each of this is a
fix-sized page

Program

Instructions		Data	
0f00bb27		00c2e800	
509cbd23		00000008	
00005d24		00c2f000	
0000bd24		00000008	
2ca422a0		00c2f800	
130020e4		00000008	
00003d24		00c30000	
2ca4e2b3		00000008	

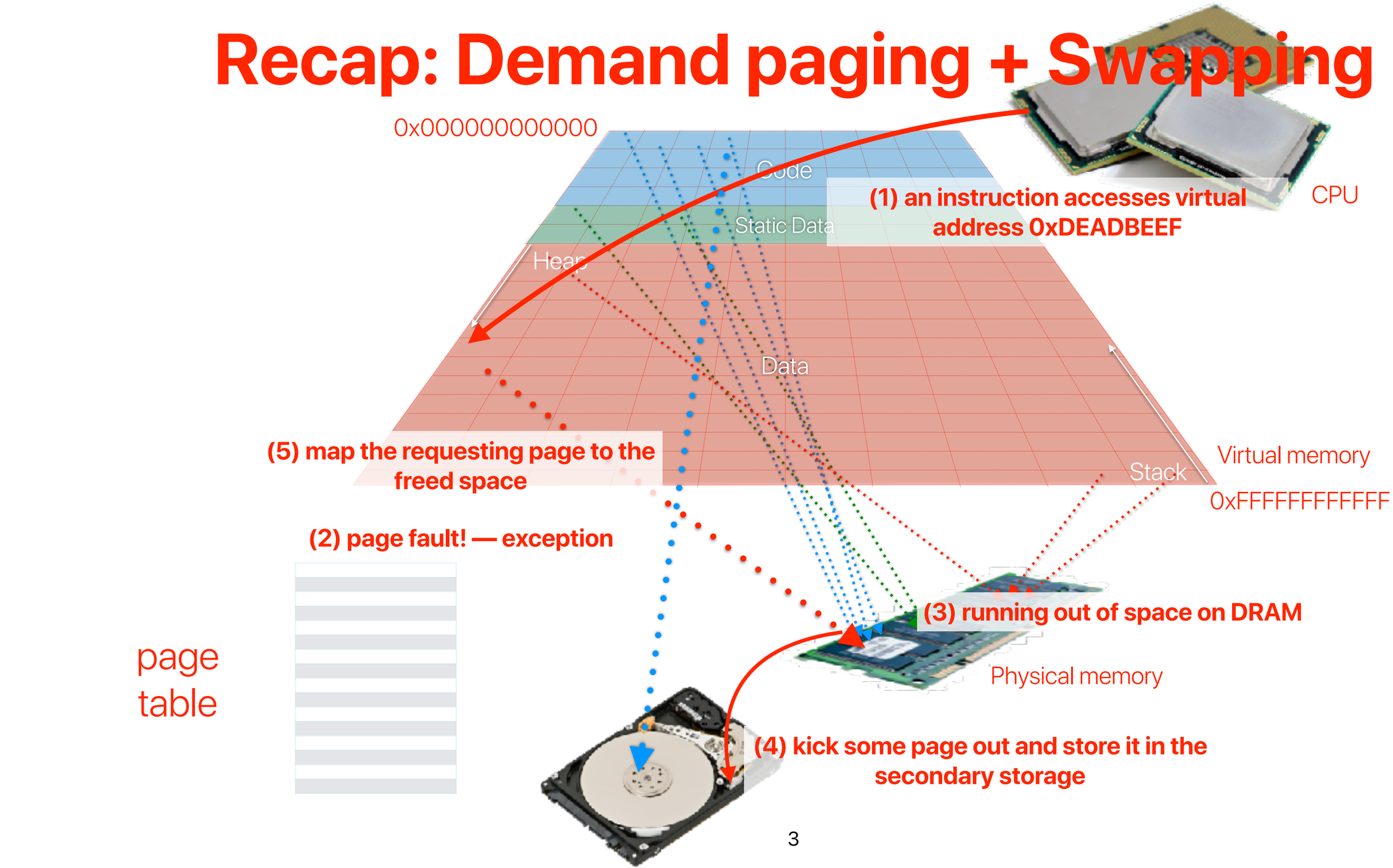
0f00bb27	00c2e800
509cbd23	00000008
00005d24	00c2f000
0000bd24	00000008
0f00bb27	00c2f800
509cbd23	00000008
00005d24	00c30000
0000bd24	00000008

Memory

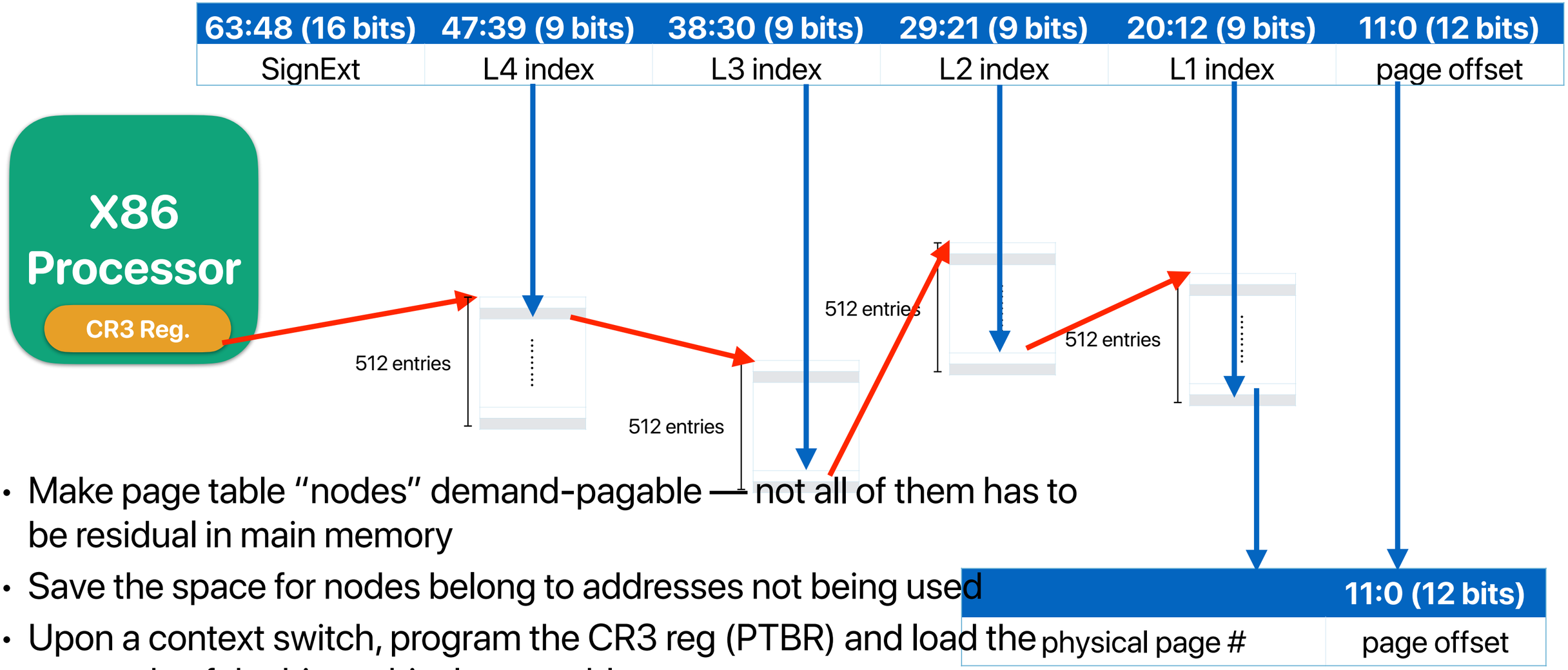
Program

Instructions		Data	
0f00bb27		00c2e800	
509cbd23		00000008	
00005d24		00c2f000	
0000bd24		00000008	
2ca422a0		00c2f800	
130020e4		00000008	
00003d24		00c30000	
2ca4e2b3		00000008	

Recap: Demand paging + Swapping



Recap: Hierarchical page table to make paging feasible



- Make page table "nodes" demand-pagable — not all of them has to be residual in main memory
- Save the space for nodes belong to addresses not being used
- Upon a context switch, program the CR3 reg (PTBR) and load the root node of the hierarchical page table

Recap: Page replacement policy

- Goal: Identify page to remove that will avoid future page faults (i.e. utilize locality as much as possible)
- Implementation Goal: Minimize the amount of software and hardware overhead
  - Example:
    - Memory (i.e. RAM) access time: 100ns
    - Disk access time: 10ms
    - P_f: probability of a page fault
    - Effective Access Time = $10^{-7} + P_f * 10^{-3}$
  - When $P_f = 0.001$:
    Effective Access Time = 10,100ns
  - **Takeaway: Disk access tolerable only when it is extremely rare**

What VAX/VMS proposed to achieve these goals?

- Considering the optimization goals and the proposed VAX/VMS mechanisms, which of the following combinations is incorrect?

Goal		Optimization	
A ✓	Process startup cost	W	Demand-zero & copy-on-refernce
B ✓	Process performance interference	X	Process-local replacement
C	Page table lookup overhead	Y	Page clustering
D ✓	Paging load on disks	Z	Page caching

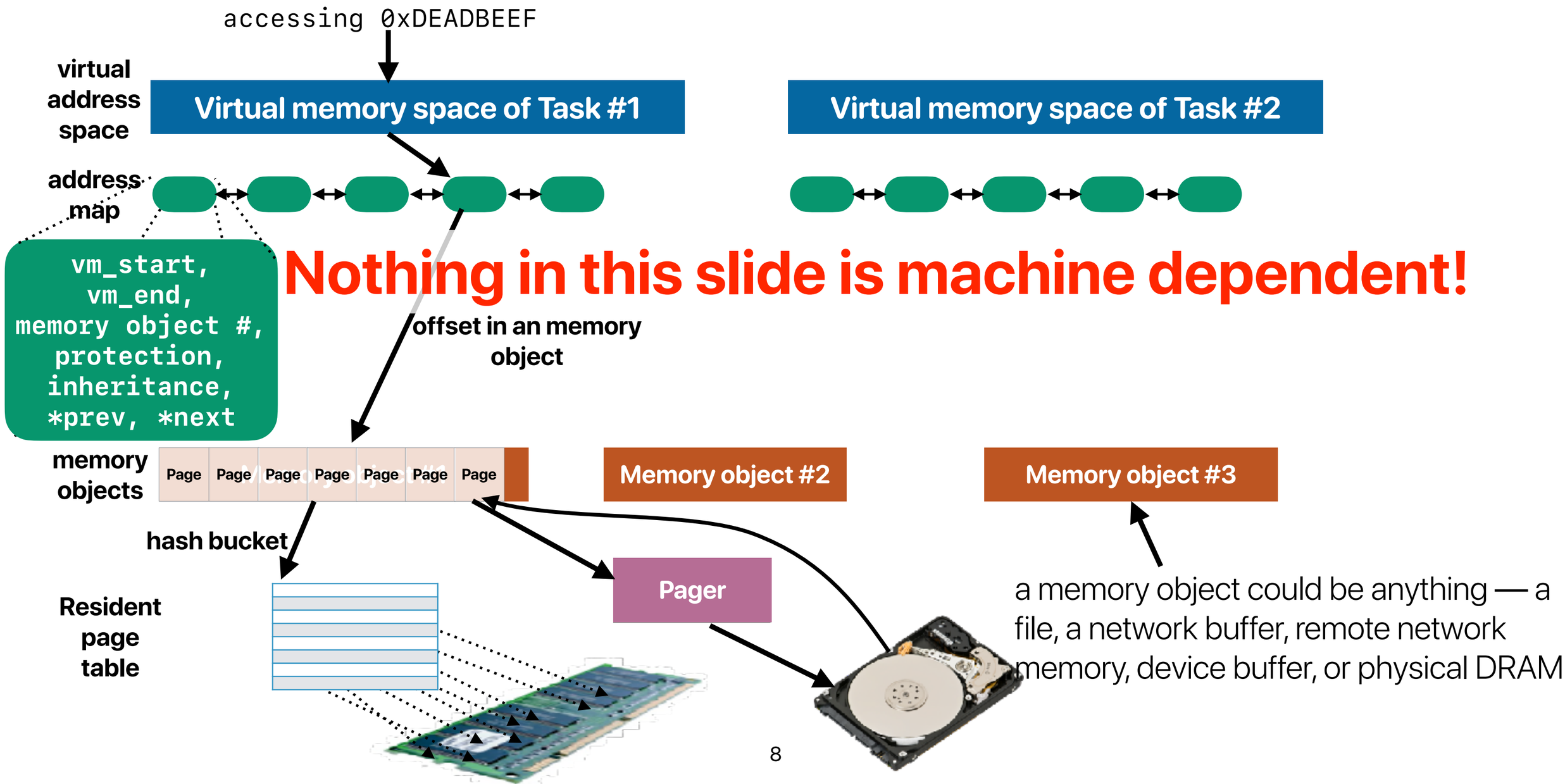
also helps reduce disk loads

- We're still using their proposed techniques almost everyday!
- It's basically the baseline UNIX VM design

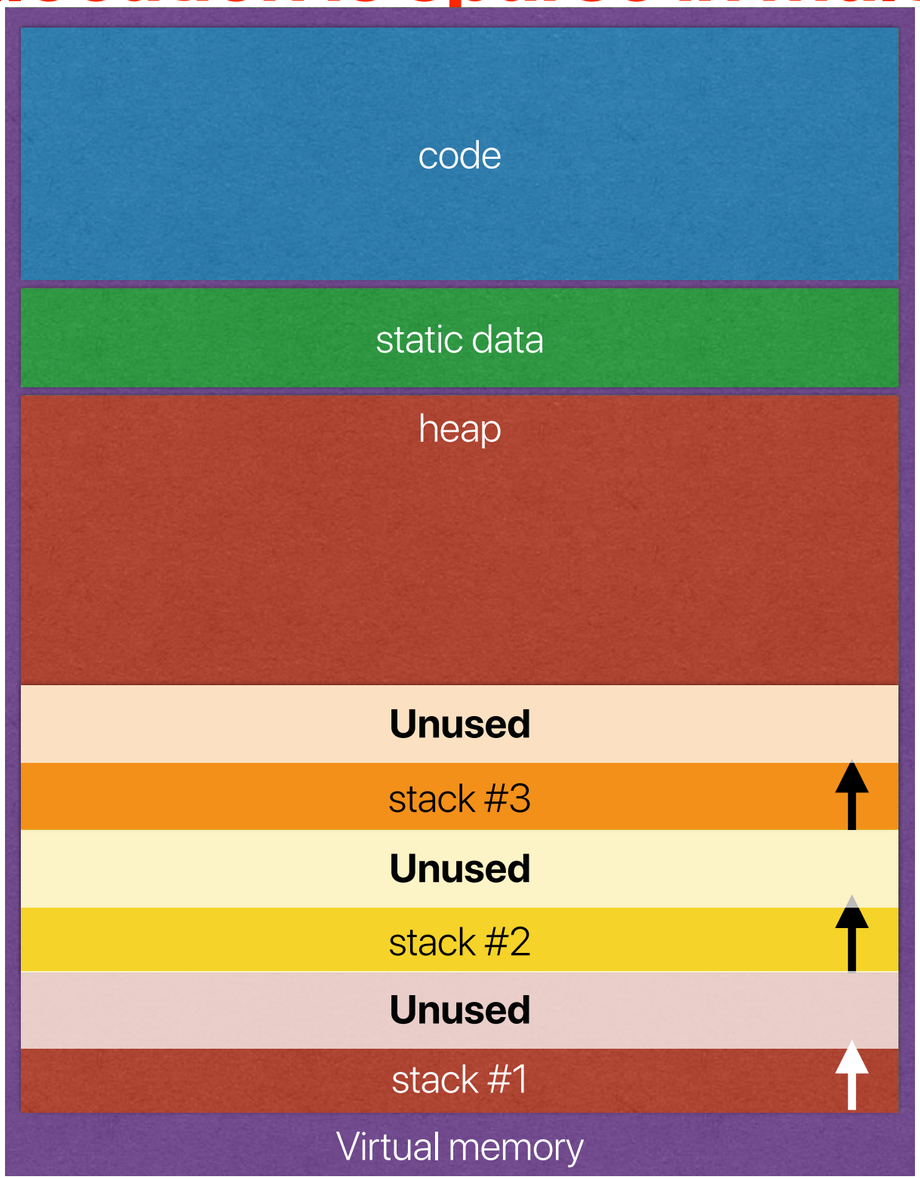
Machine-Independent Virtual Memory Management for Paged Uniprocessor and Multiprocessor Architectures

Richard Rashid, Avadis Tevanian, Michael Young, David Golub, Robert Baron, David Black,
William Bolosky, and Jonathan Chew
Carnegie-Mellon University, NeXT, University of Rochester

Recap: Overview of Mach's VM



Recap: Address allocation is sparse in multithreading model!



address map

Outline

- Page replacement policies
- Page replacement policy once used in UNIX: Converting a Swap-Based System to do Paging in an Architecture Lacking Page-Reference Bits
- Another popular page replacement policy: WSClcok - A Simple and Effective Algorithm for Virtual Memory Management

Page replacement policies from textbooks

Page replacement policy

- We need to determine:
  - Which page(s) to remove
  - When to remove the page(s)
- Goals
  - Identify page to remove that will avoid future page faults (i.e. utilize locality as much as possible)
  - Minimize the amount of software and hardware overhead

Page replacement algorithms

- FIFO: Replace the oldest page
- LRU: Replace page that was the least recently used (longest since last use)

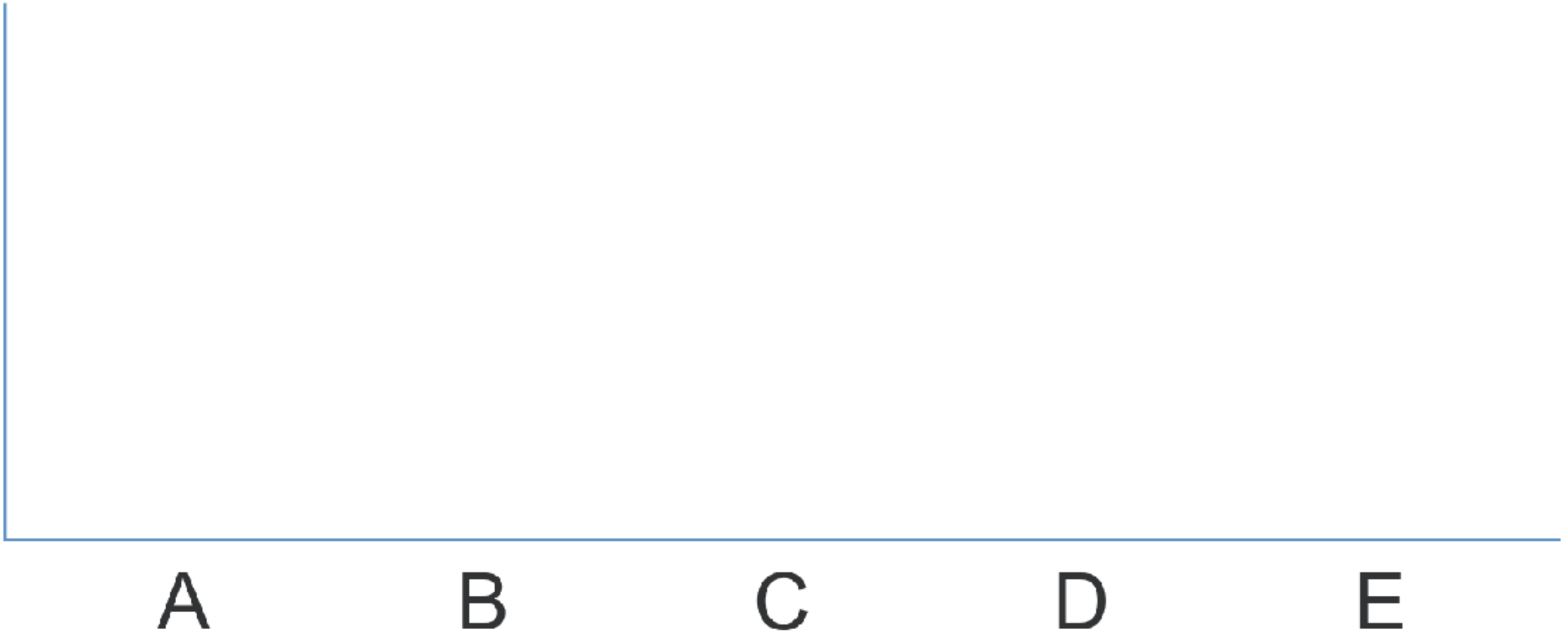
FIFO v.s. LRU

- Assume your OS uses FIFO policy when handle page faults. Also assume that we have 3 physical memory pages available. Compared with the same machine using an OS with LRU page replacement police, how many more page faults will you see for the FIFO based OS in the following page reference sequence?

	0	1	2	3	4	5	6	7	8	9	10	11
Page #	2	3	2	1	5	2	4	5	3	2	5	2

A. 0
B. 1
C. 2
D. 3
E. 4

FIFO



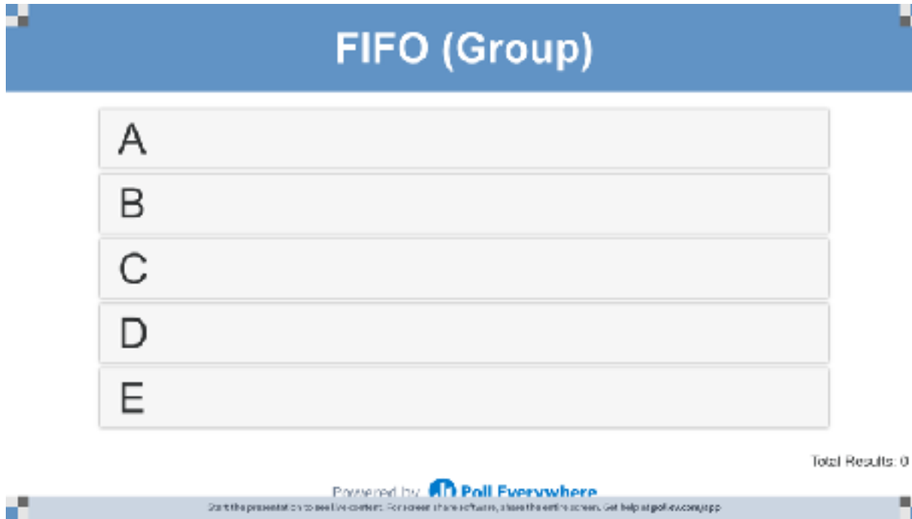
Total Results: 0

FIFO v.s. LRU

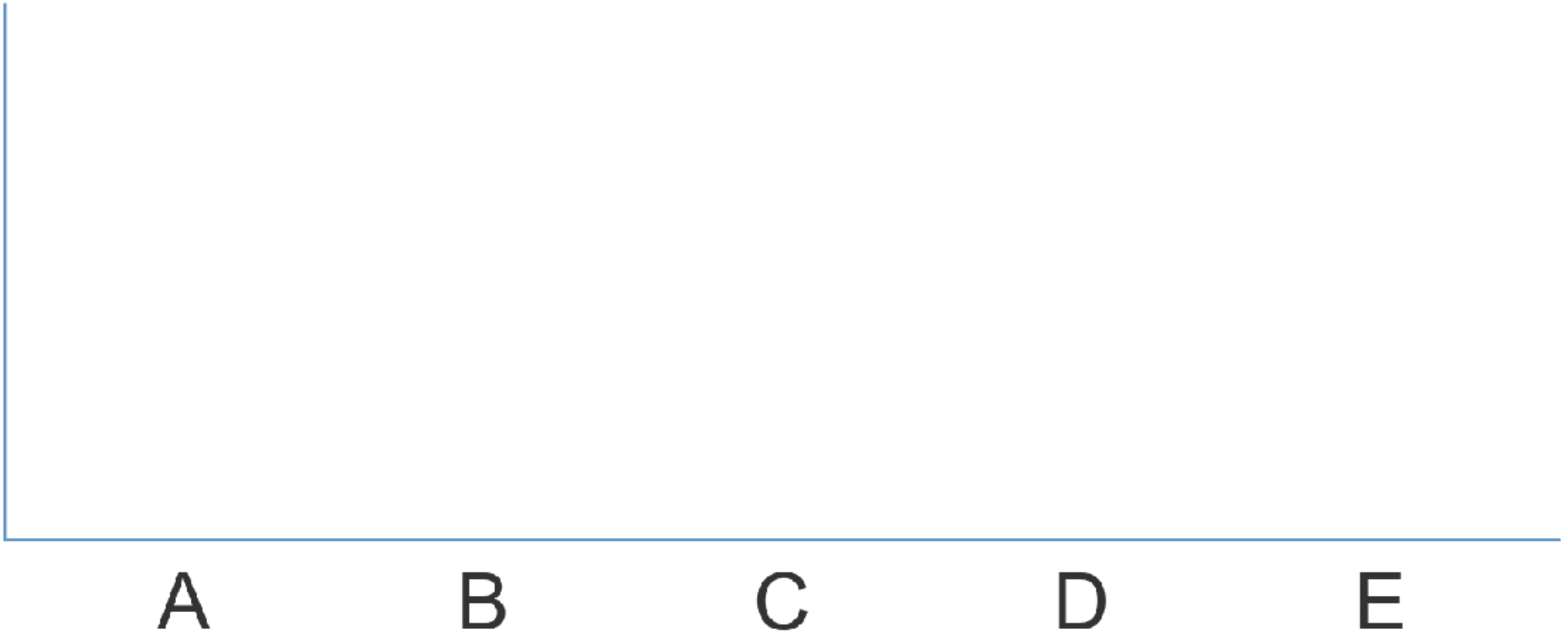
- Assume your OS uses FIFO policy when handle page faults. Also assume that we have 3 physical memory pages available. Compared with the same machine using an OS with LRU page replacement police, how many more page faults will you see for the FIFO based OS in the following page reference sequence?

	0	1	2	3	4	5	6	7	8	9	10	11
Page #	2	3	2	1	5	2	4	5	3	2	5	2

A. 0
B. 1
C. 2
D. 3
E. 4



FIFO (Group)



Total Results: 0

FIFO v.s. LRU

- Assume your OS uses FIFO policy when handle page faults. Also assume that we have 3 physical memory pages available. Compared with the same machine using an OS with LRU page replacement police, how many more page faults will you see for the FIFO based OS in the following page reference sequence?

A. 0
B. 1
C. 2
D. 3
E. 4

	0	1	2	3	4	5	6	7	8	9	10	11
Page #	2	3	2	1	5	2	4	5	3	2	5	2

FIFO

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

LRU

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

FIFO v.s. LRU

	FIFO	LRU
Implementation	Easy — circular queue	May require hardware support or linked list or additional timestamps in page tables
Execution overhead	Low	High — you need to manipulate the list or update every counter
Performance	Usually not as good as LRU	Usually better than FIFO

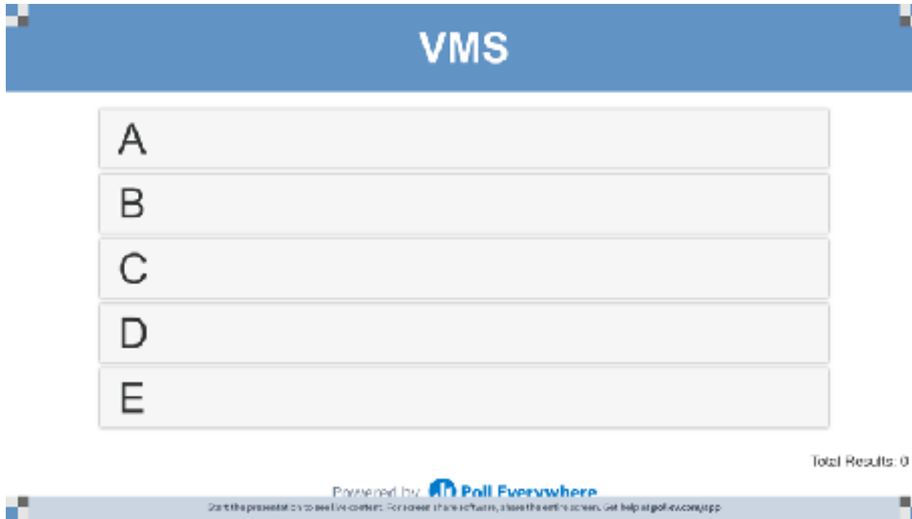
Converting a Swap-Based System to do Paging in an Architecture Lacking Page-Reference Bits

Özalp Babaoglu and William Joy*
Cornell University and University of California, Berkeley

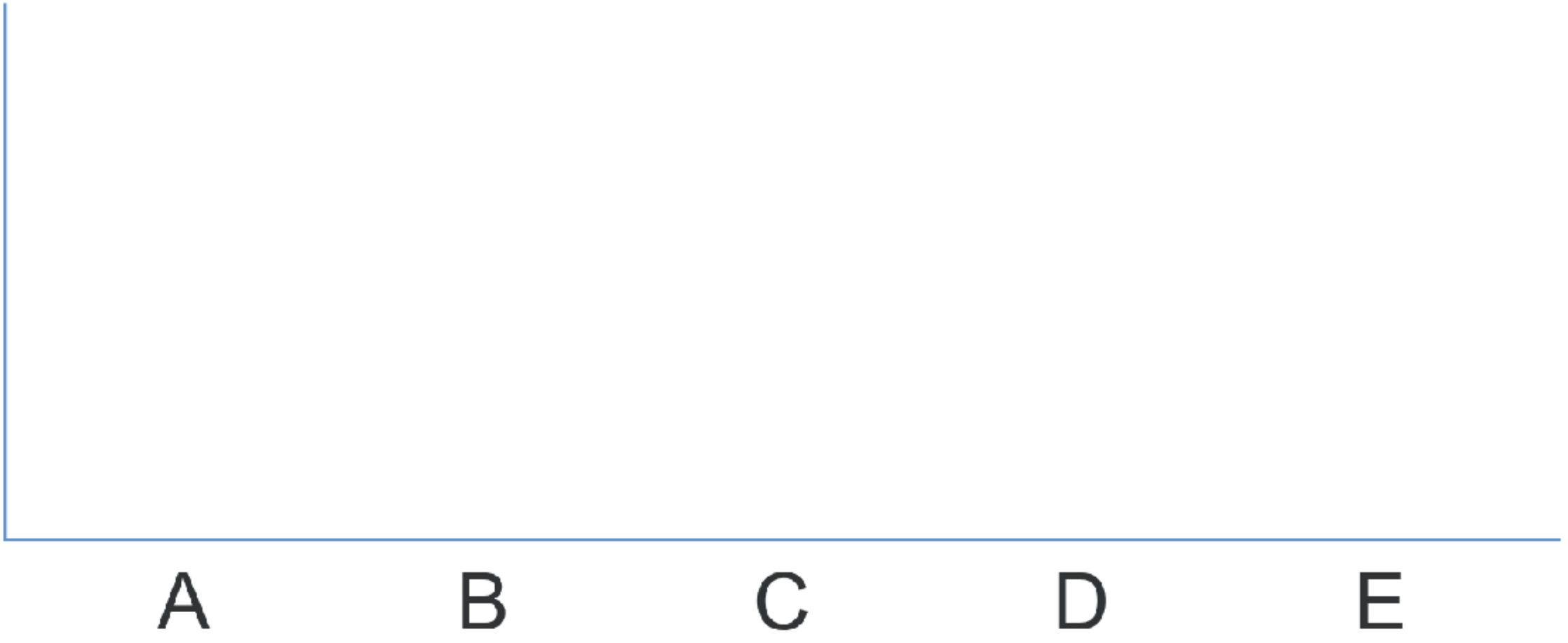
The VMS/Old UNIX VM

- Regarding the original UNIX VM (basically the VMS), please identify how many of the following statements are correct.
  - ① VAX machine provides no hardware support for page replacement policies
  - ② VMS implements FIFO policy for page replacement
  - ③ A process's resident set cannot be adjusted even though that process is the only process in the system
  - ④ VMS swaps out all memory page belong to a process when that process is switched out

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4



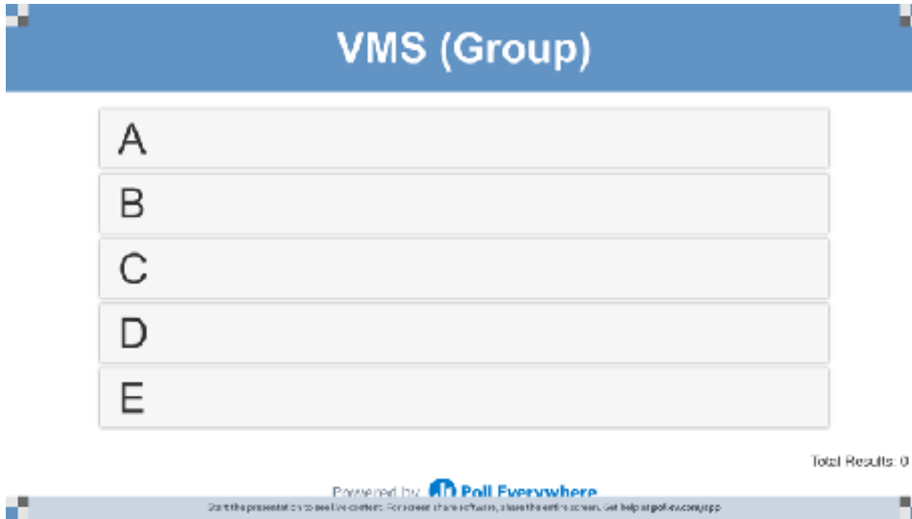
VMS



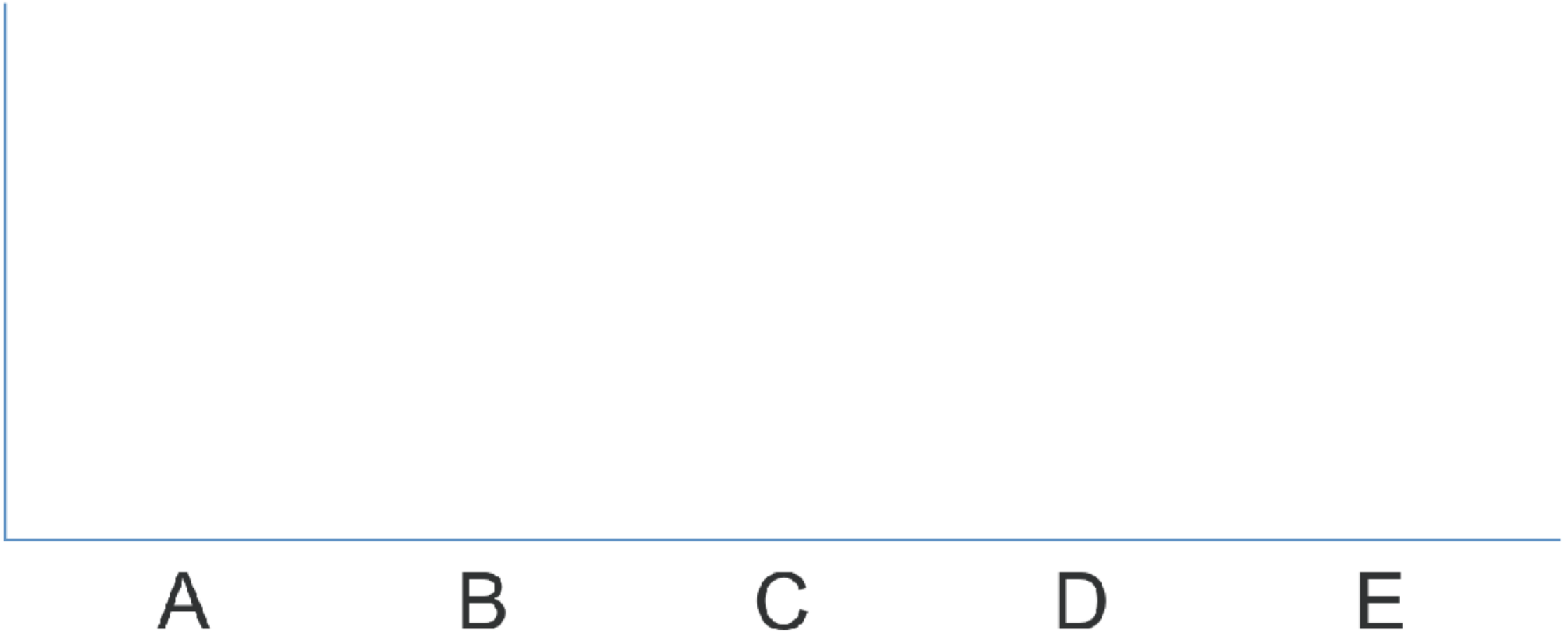
Total Results: 0

The VMS/Old UNIX VM

- Regarding the original UNIX VM (basically the VMS), please identify how many of the following statements are correct.
  - ① VAX machine provides no hardware support for page replacement policies
  - ② VMS implements FIFO policy for page replacement
  - ③ A process's resident set cannot be adjusted even though that process is the only process in the system
  - ④ VMS swaps out all memory page belong to a process when that process is switched out
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4



VMS (Group)



Total Results: 0

The VMS/Old UNIX VM

- Regarding the original UNIX VM (basically the VMS), please identify how many of the following statements are correct.
  - ① VAX machine provides no hardware support for page replacement policies
  - ② VMS implements FIFO policy for page replacement
  - ③ A process's resident set cannot be adjusted even though that process is the only process in the system
  - ④ VMS swaps out all memory page belong to a process when that process is switched out

— Really inefficient if you have frequent context switches or if you have many applications in-fly

A. 0
B. 1
C. 2
D. 3
E. 4

Whenever a process is removed from memory, its entire resident set is written to the swap file, along with some

tions) base their decisions on. Without even this minimal page reference information, the only reasonable algorithms for replacing pages are the First-In-First-Out (FIFO) and the Random (RAND)

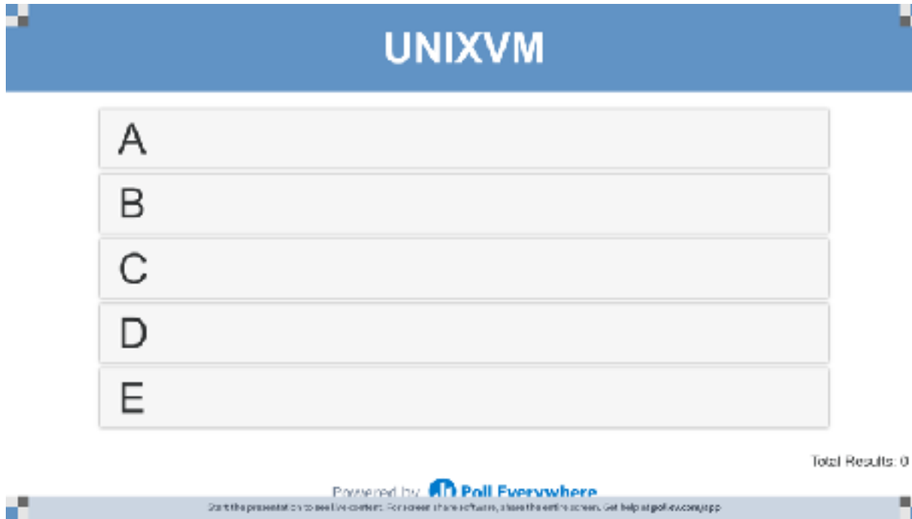
In VMS, the vendor-supplied operating system for the VAX, the solution to the replacement decision is simple. Each process is assigned a fixed-size memory partition, called a resident set, that is managed according to the FIFO policy. Pages

The Why of Babaoglu new UNIX VM

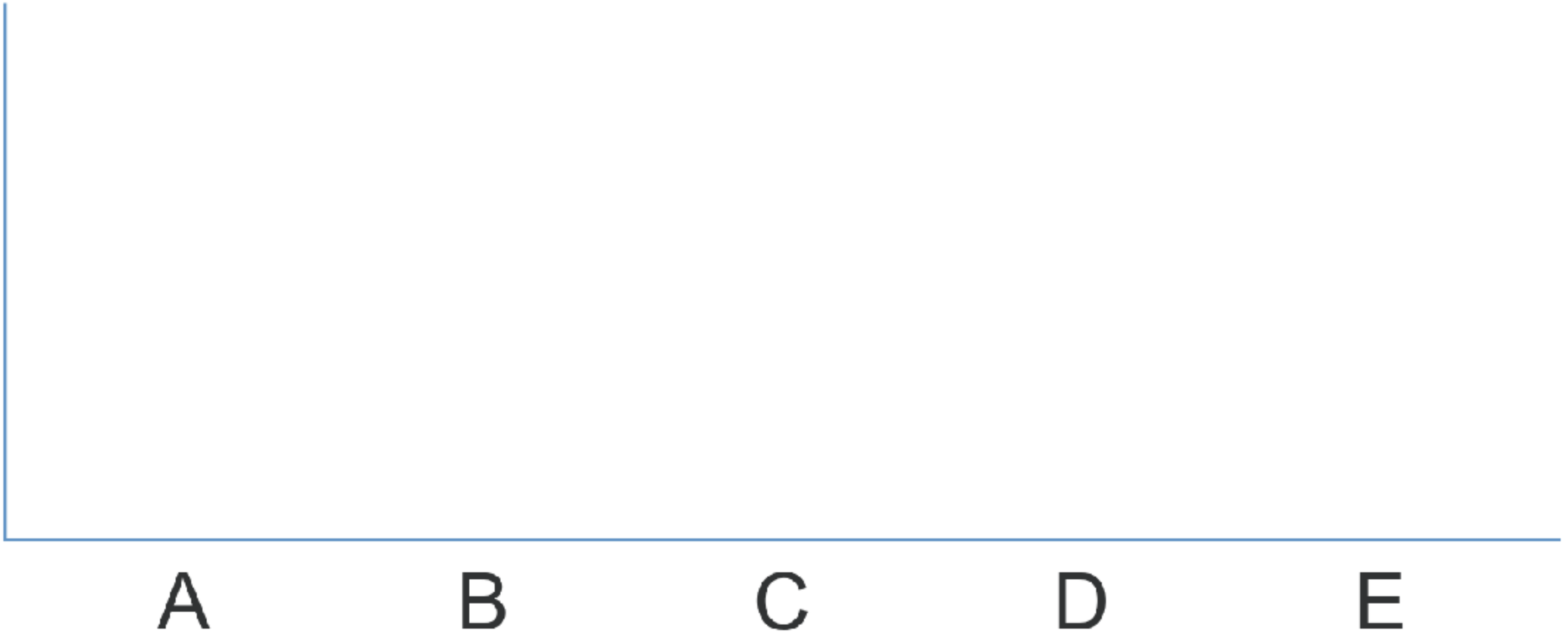
- The original UNIX is a **swap-based** system
  - Whenever you have a context switch, swap the whole process out from the memory
  - Really inefficient if you have frequent context switches or if you have many applications in-fly
  - Imply that the **modern** UNIX or Linux **does not** do this
- Efficient page replacement policies and other virtual optimization techniques cannot be implemented easily without appropriate hardware support

The page replacement policy proposed

- How many of following statements fit the page replacement policy that the paper implements?
  - ① It uses LRU (least-recently-used) as the page replacement policy
  - ② Page replacement policy are only triggered whenever a page fault occurs
  - ③ It attaches a timestamp to each page table entry instead of using the reference bit from hardware
  - ④ Processes are allocated a fixed set of pages and swap in/out to/from those pages
  - ⑤ The page replacement policy helps to guarantee the response time of short programs
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4



UNIXVM

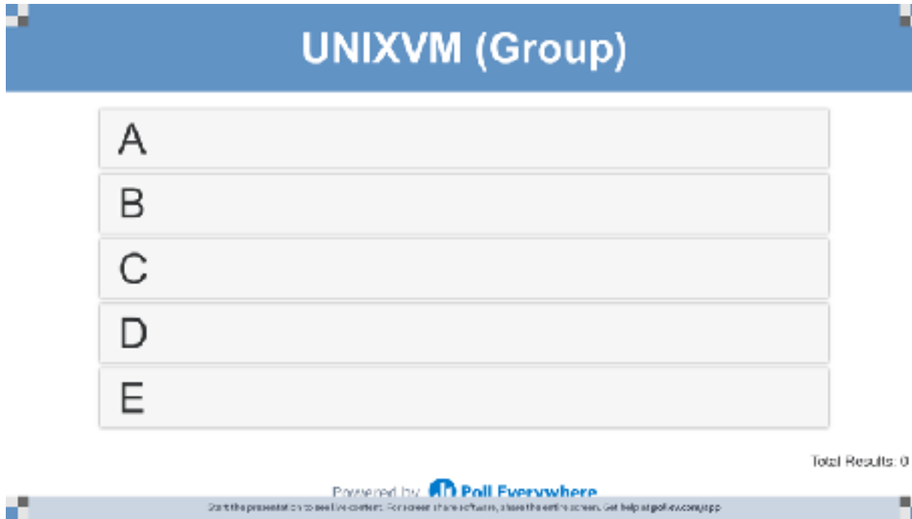


Total Results: 0

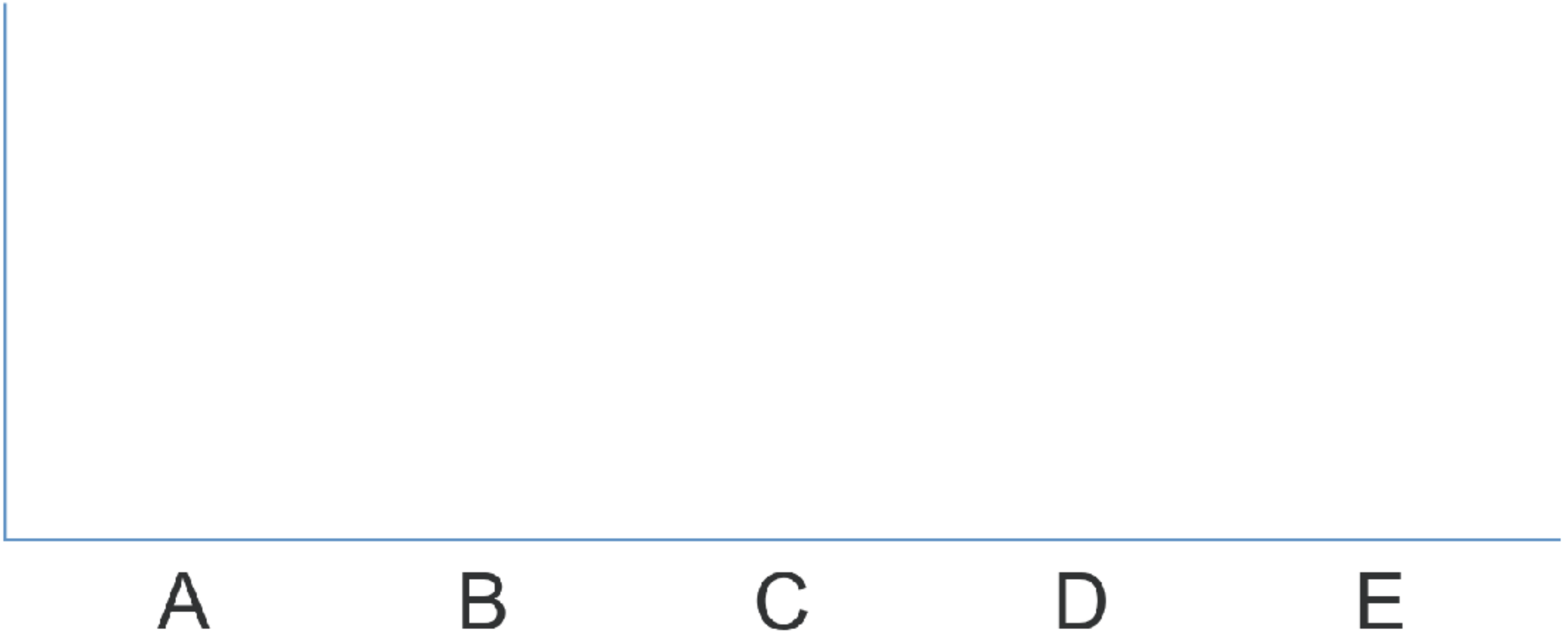
https://www.pollev.com/hungweitseng close in 1:00

The page replacement policy proposed

- How many of following statements fit the page replacement policy that the paper implements?
  - ① It uses LRU (least-recently-used) as the page replacement policy
  - ② Page replacement policy are only triggered whenever a page fault occurs
  - ③ It attaches a timestamp to each page table entry instead of using the reference bit from hardware
  - ④ Processes are allocated a fixed set of pages and swap in/out to/from those pages
  - ⑤ The page replacement policy helps to guarantee the response time of short programs
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4



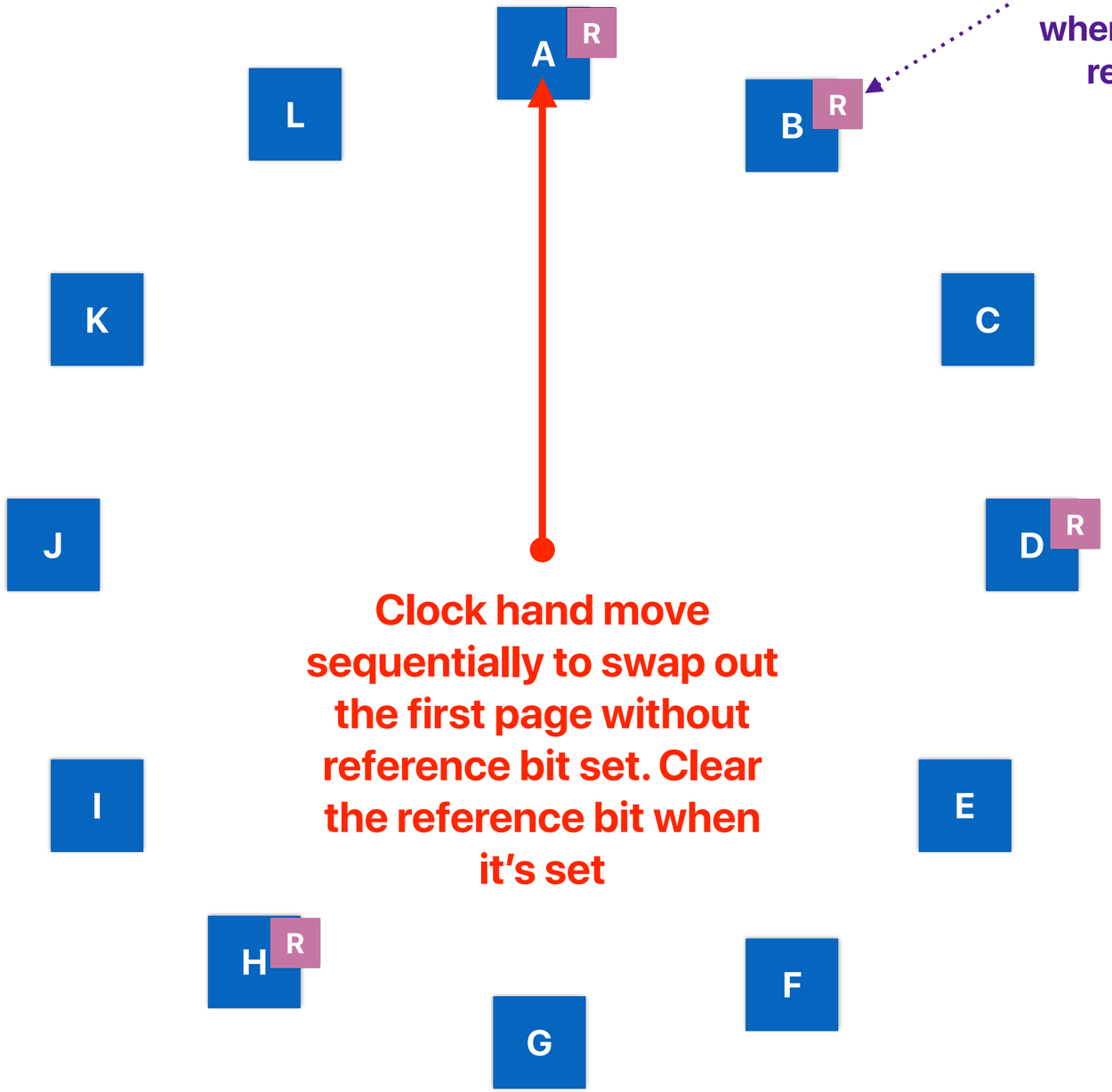
UNIXVM (Group)



Total Results: 0

Clock algorithm

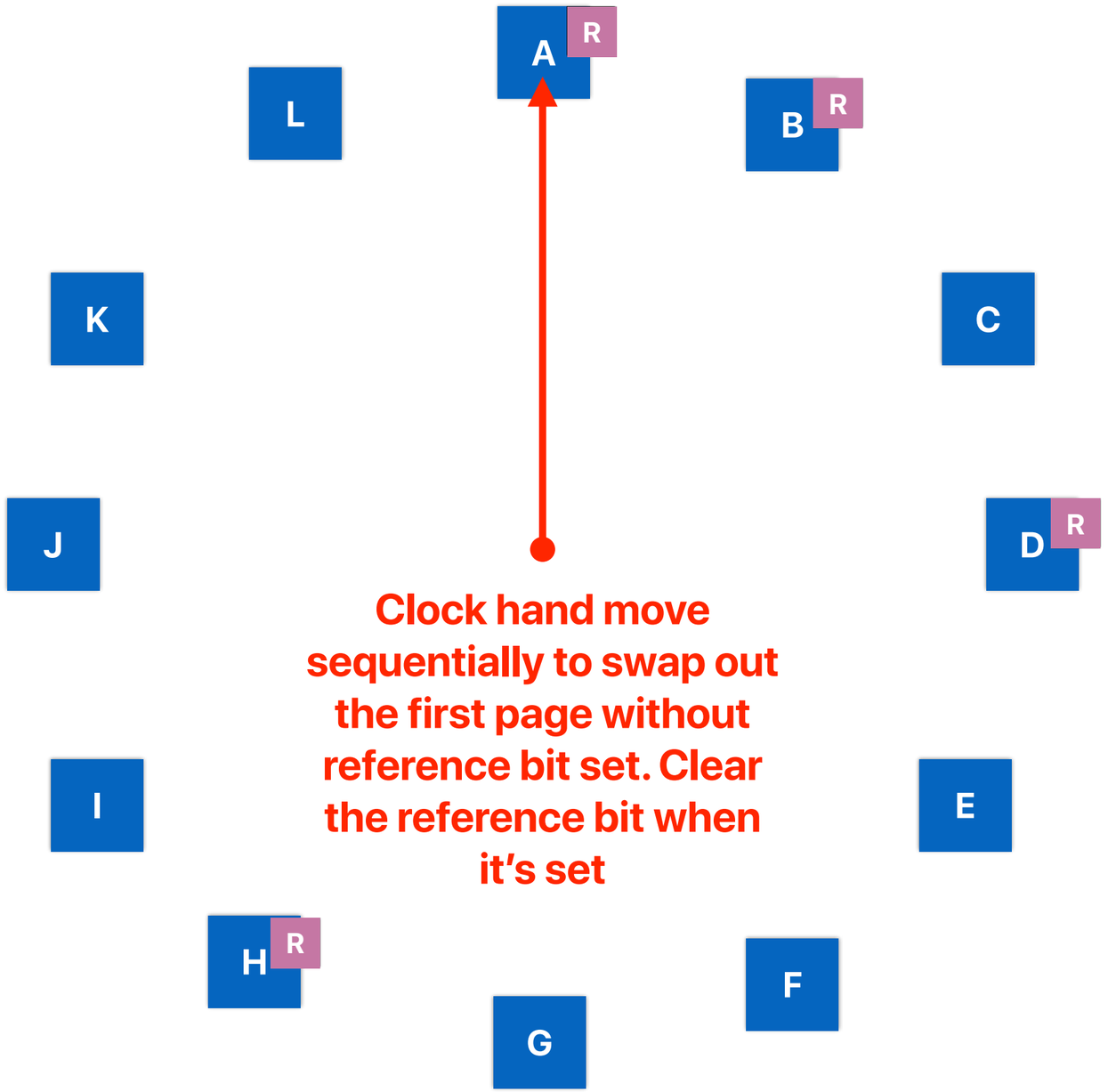
attach a “reference bit” to each PTE, set to true when the page is referenced



Clock hand move sequentially to swap out the first page without reference bit set. Clear the reference bit when it’s set

Clock algorithm in motion

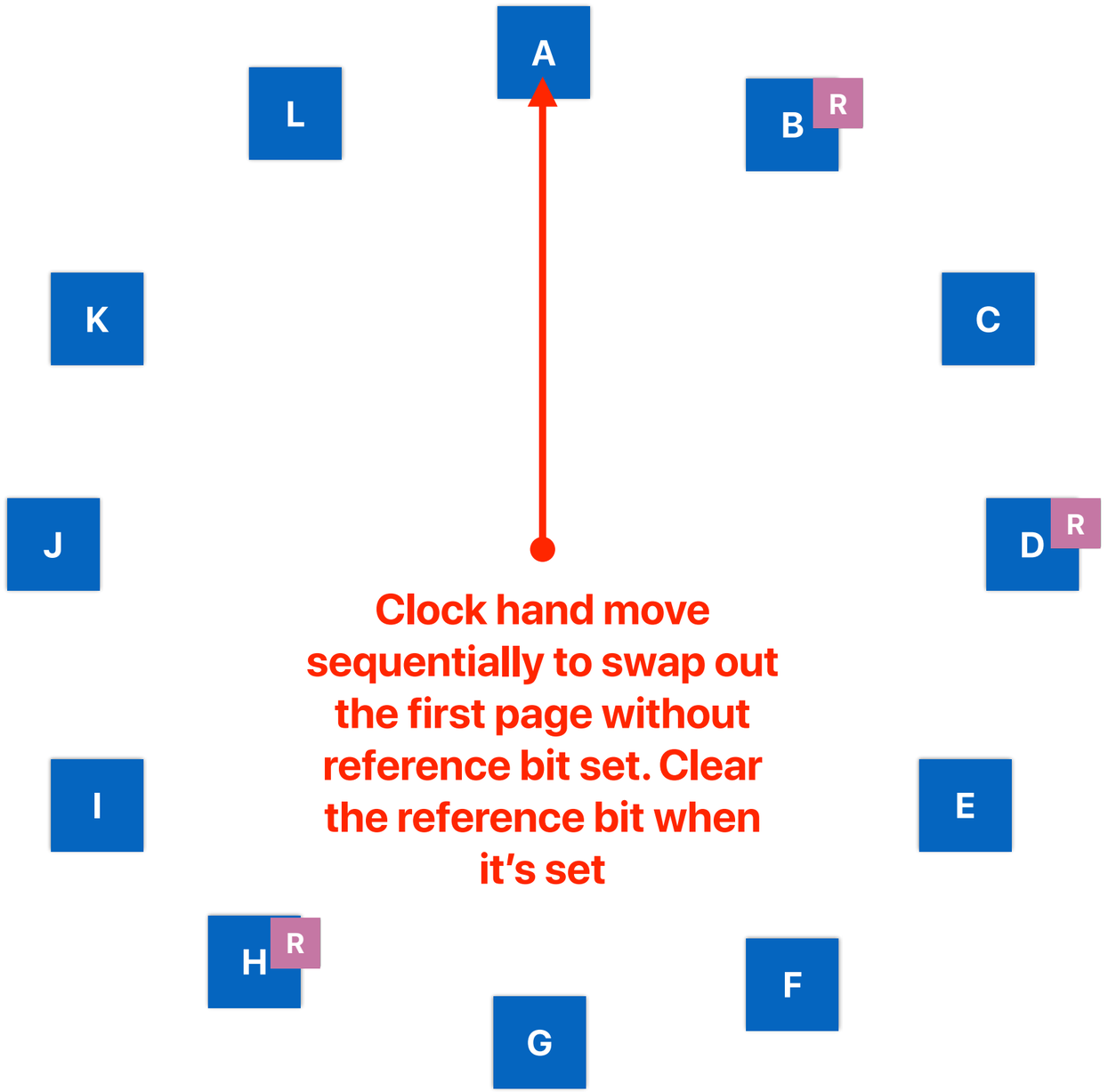
Where to put M ?



Clock hand move sequentially to swap out the first page without reference bit set. Clear the reference bit when it's set

Clock algorithm in motion

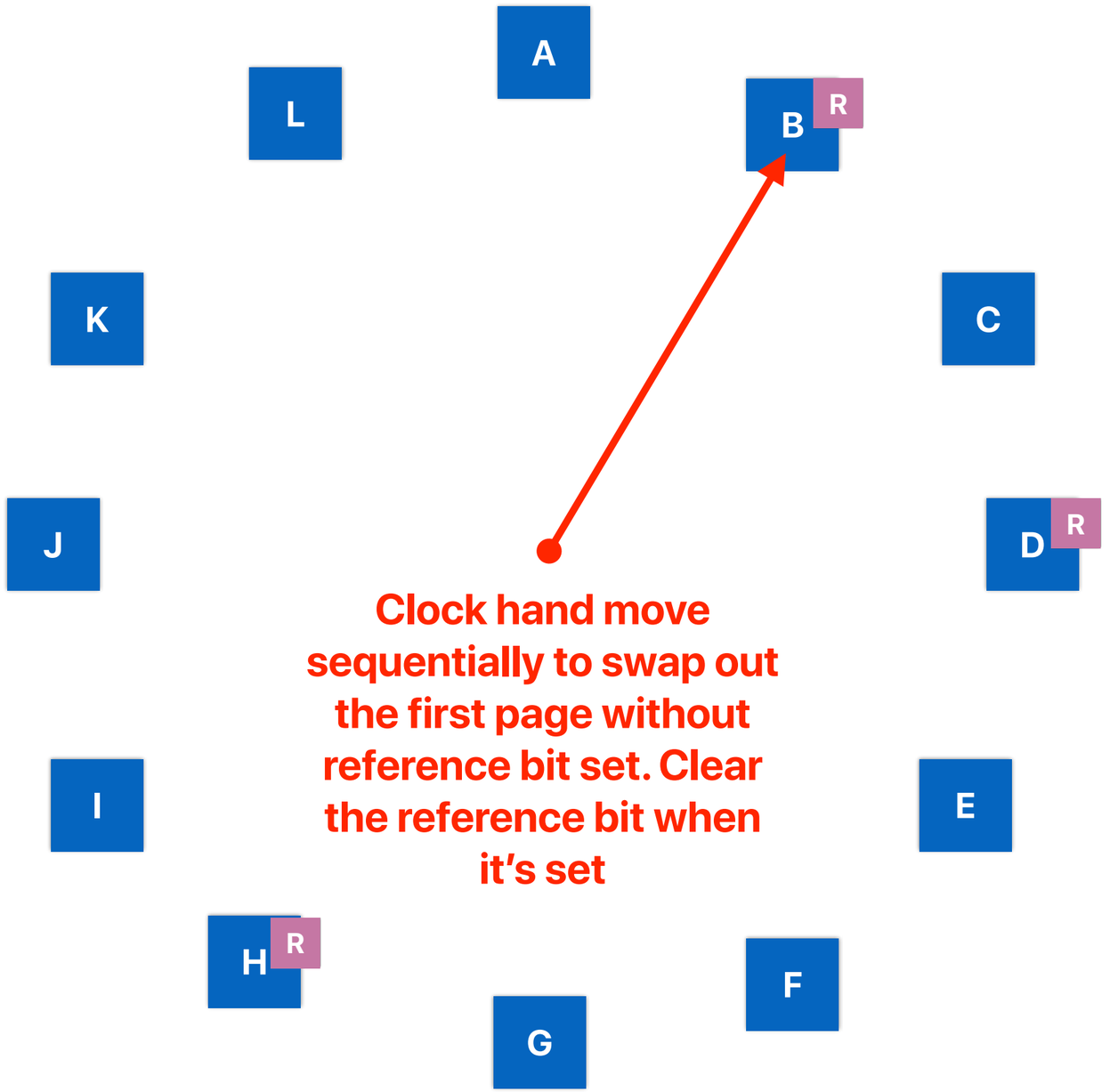
Where to put M ?



Clock hand move sequentially to swap out the first page without reference bit set. Clear the reference bit when it's set

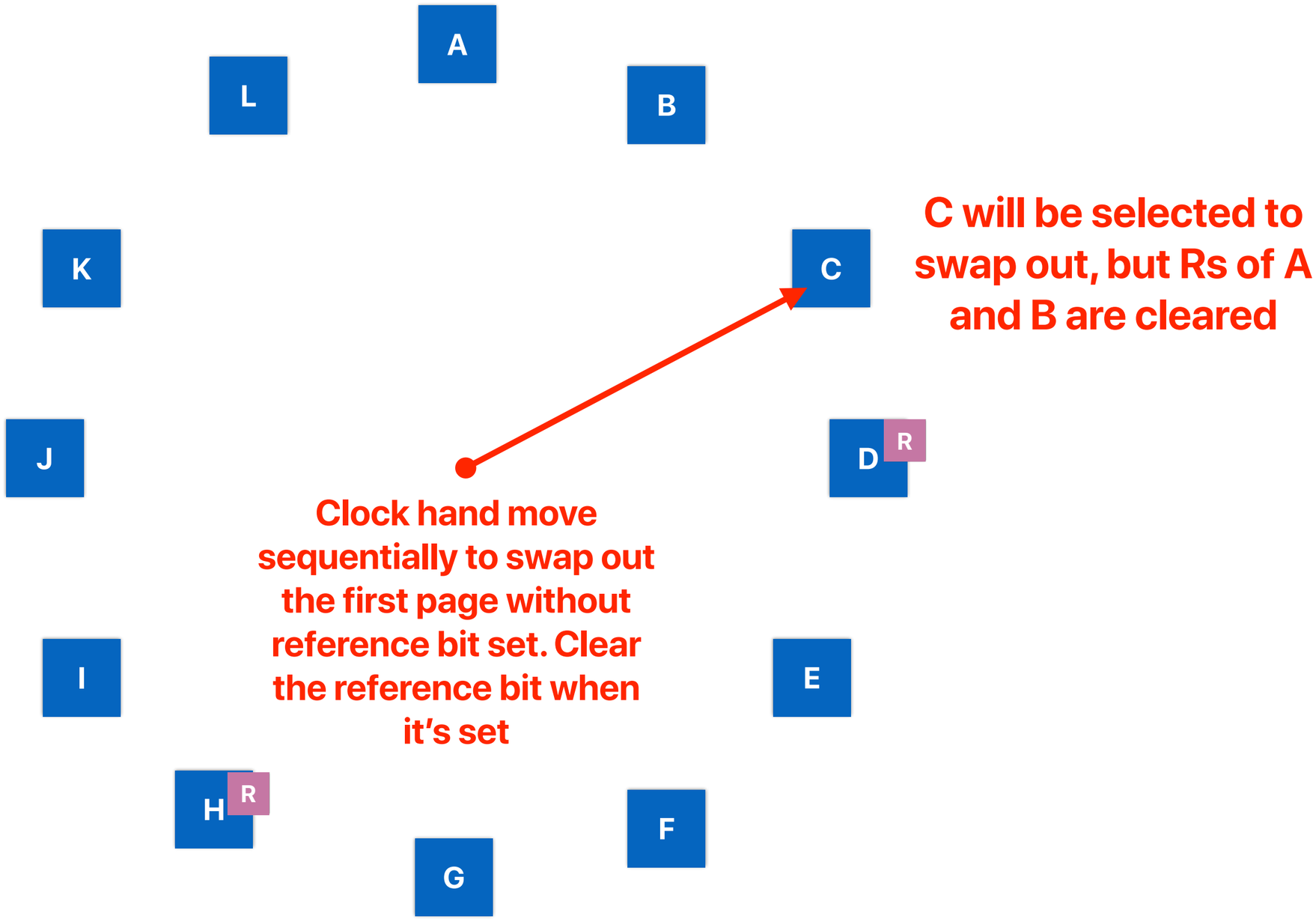
Clock algorithm in motion

Where to put M ?



Clock algorithm in motion

Where to put M ?



C will be selected to swap out, but Rs of A and B are cleared

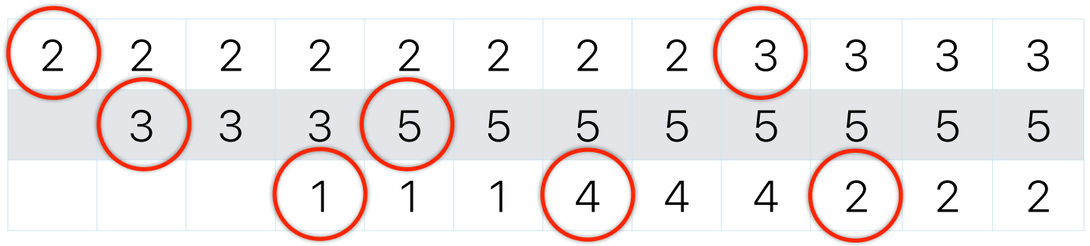
Clock hand move sequentially to swap out the first page without reference bit set. Clear the reference bit when it's set

Recap: LRU

- Assume your OS uses LRU policy when handle page faults. Also assume that we have 3 physical memory pages available. How many page faults will you see in the following page reference sequence?

	0	1	2	3	4	5	6	7	8	9	10	11
Page #	2	3	2	1	5	2	4	5	3	2	5	2

A. 5
B. 6
C. 7
D. 8
E. 9



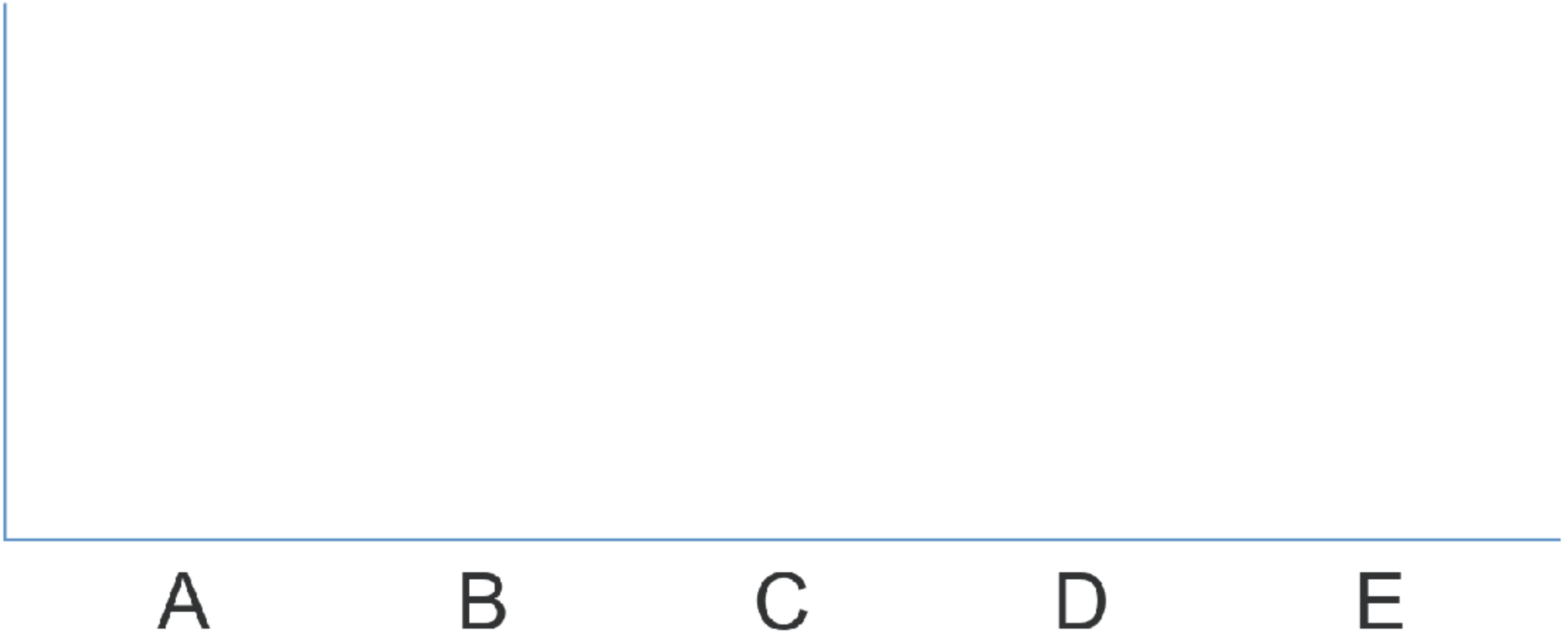
How good is clock?

- Assume your OS uses the clock policy when handle page faults. Also assume that we have 3 physical memory pages available. How many page faults will you see in the following page reference sequence?

	0	1	2	3	4	5	6	7	8	9	10	11
Page #	2	3	2	1	5	2	4	5	3	2	5	2

A. 5
B. 6
C. 7
D. 8
E. 9

Clock



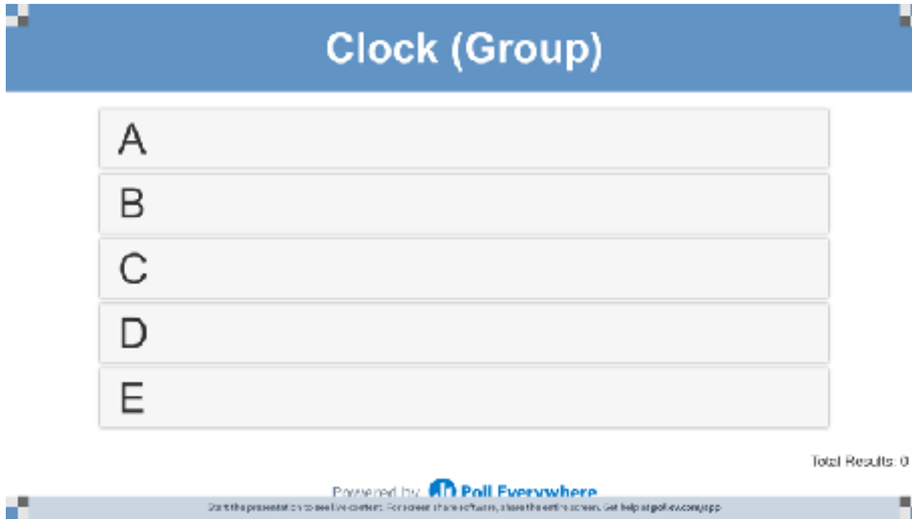
Total Results: 0

How good is clock?

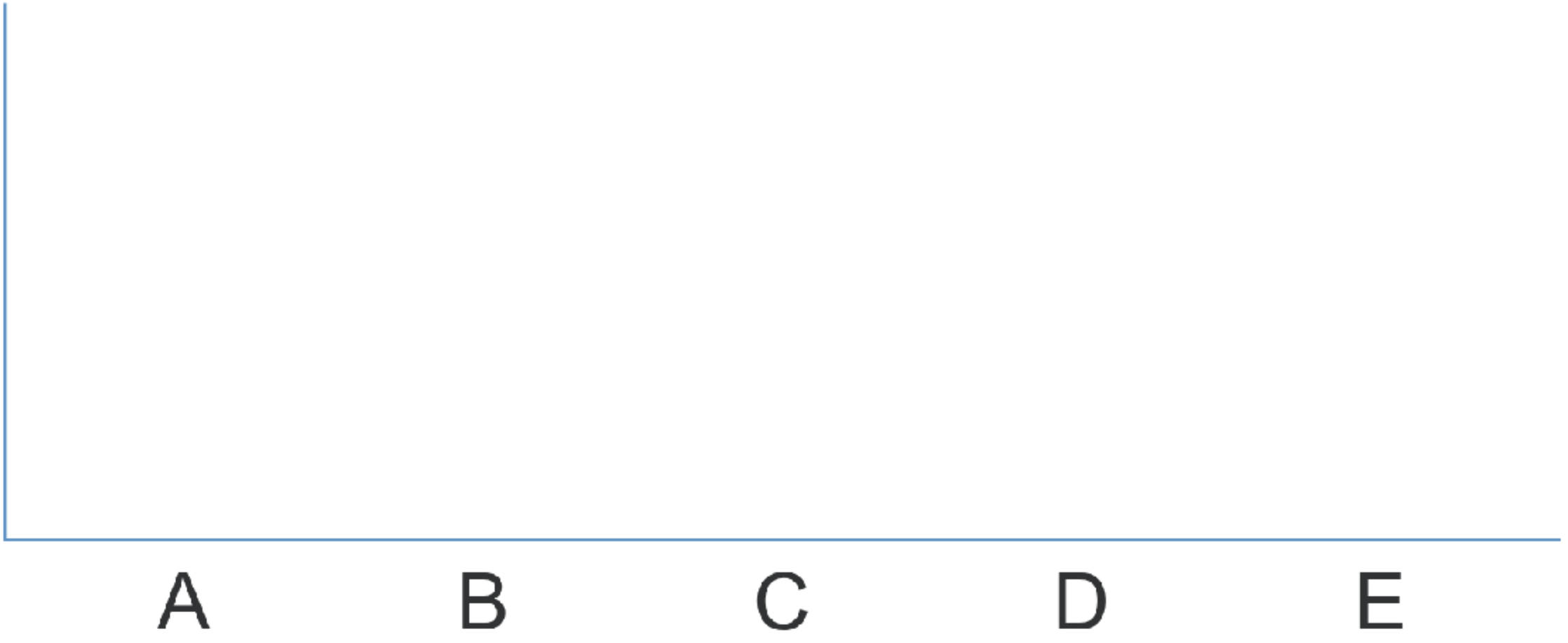
- Assume your OS uses the clock policy when handle page faults. Also assume that we have 3 physical memory pages available. How many page faults will you see in the following page reference sequence?

	0	1	2	3	4	5	6	7	8	9	10	11
Page #	2	3	2	1	5	2	4	5	3	2	5	2

A. 5
B. 6
C. 7
D. 8
E. 9



Clock (Group)



Total Results: 0

How good is clock?

- Assume your OS uses the clock policy when handle page faults. Also assume that we have 3 physical memory pages available. How many page faults will you see in the following page reference sequence?

	0	1	2	3	4	5	6	7	8	9	10	11
Page #	2	3	2	1	5	2	4	5	3	2	5	2

2	2*	2*+	2*+	2	2+	2+	2*+	2*	2+*	2*	2+*
	3	3	3	5	5	5	5+	5	5	5+	5+
			1	1*	1*	4	4	3	3	3	3

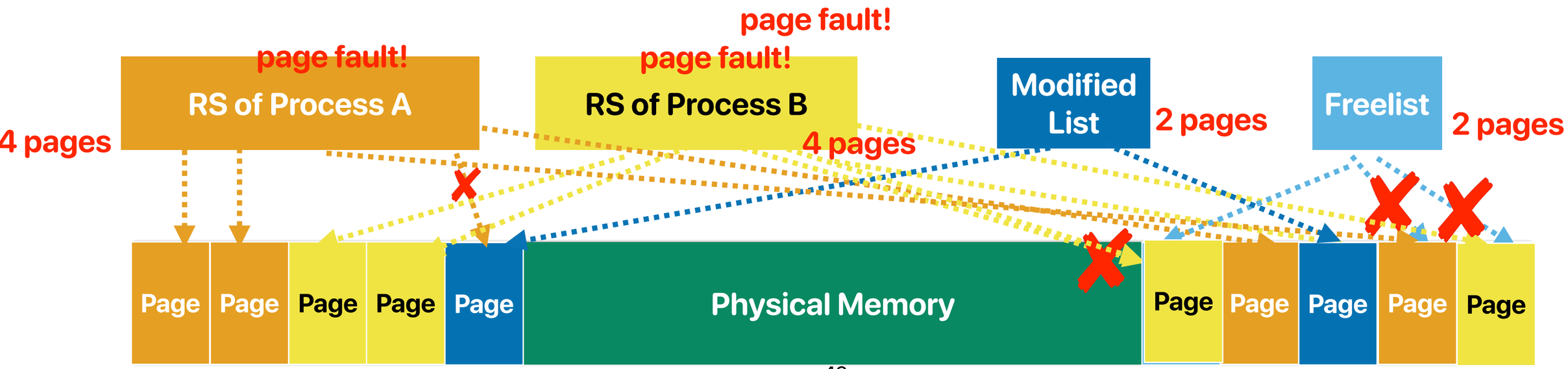
A. 5
B. 6
C. 7
D. 8
E. 9

+ means the reference bit is set

* means the current hand

Recap: Page caching to cover the performance loss

- Evicted pages will be put into one of the lists in DRAM
  - Free list: clean pages
  - Modified list: dirty pages — needs to copy data to the disk
- Page fault to any of the page in the lists will bring the page back
  - Reduces the demand of accessing disks



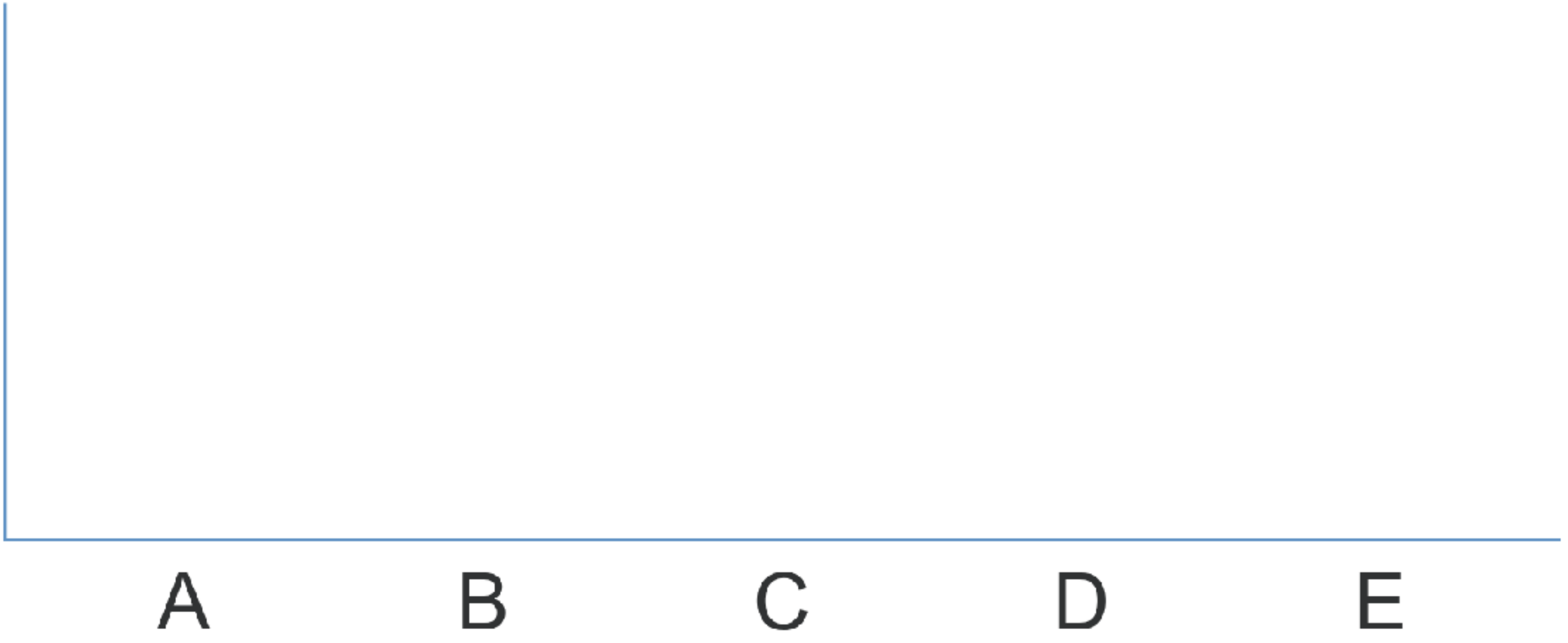
https://www.pollev.com/hungweitseng close in 1:00

Free list in Babaoglu's UNIX

- How many of the following statements regarding the "free list" is/are correct?
  - ① It can improve the latency of a page fault
  - ② It can reduce the latency of swapping out a page
  - ③ It can incur disk accesses without page faults
  - ④ It doesn't allow a page in the list to be used for other purpose

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

Freelist



Total Results: 0

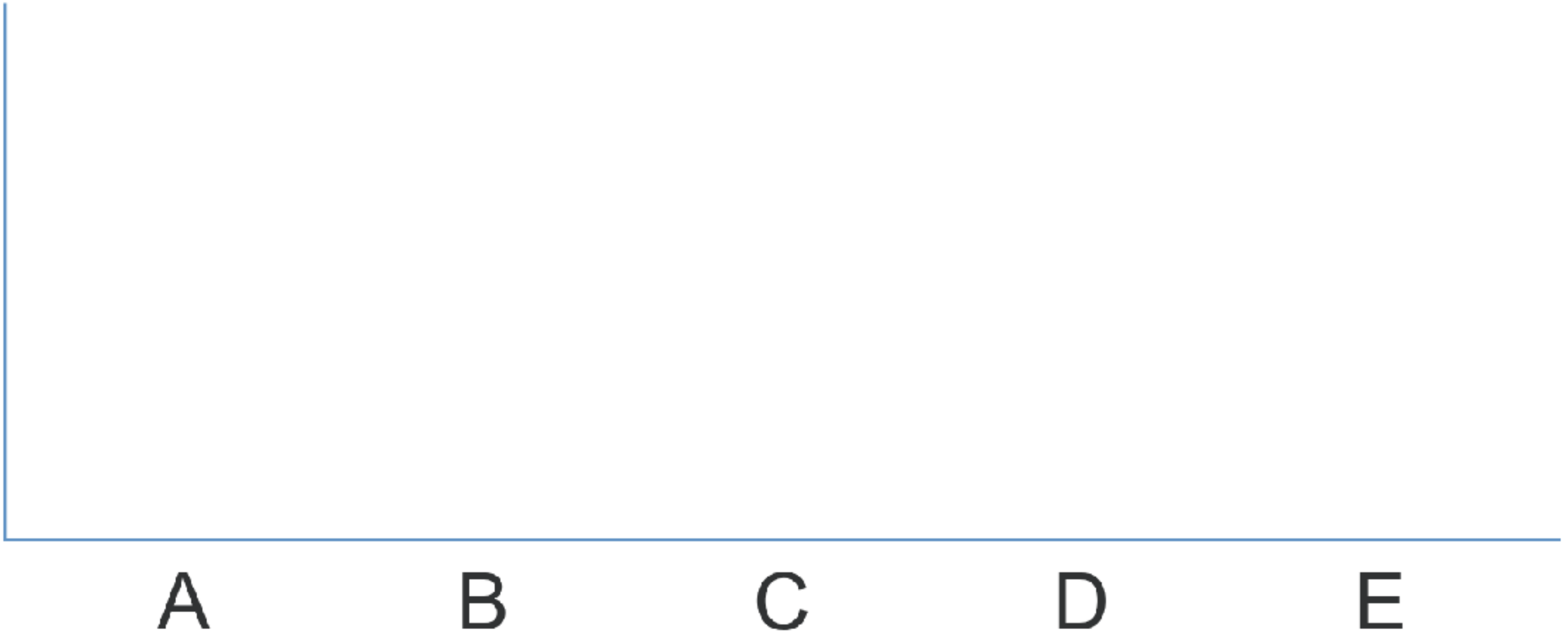
https://www.pollev.com/hungweitseng close in 1:00

Free list in Babaoglu's UNIX

- How many of the following statements regarding the "free list" is/are correct?
  - ① It can improve the latency of a page fault
  - ② It can reduce the latency of swapping out a page
  - ③ It can incur disk accesses without page faults
  - ④ It doesn't allow a page in the list to be used for other purpose

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

Freelist (Group)



Total Results: 0

Free list in Babaoglu's UNIX

- How many of the following statements regarding the "free list" is/are correct?
  - ① It can improve the latency of a page fault
  - ② It can reduce the latency of swapping out a page
  - ③ It can incur disk accesses without page faults
  - ④ It doesn't allow a page in the list to be used for other purpose
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

— instead of swapping a page during the page fault, just take one from the free list

— No! This completely depend on how fast your disk/storage is!

— Do you remember how UNIX page replacement is triggered?

— No! You can use those pages as disk caches!

Free list

- So far, we need to trigger clock policy and swap in/out on each page fault
- Why don’t we prepare more free pages each time so that we can feed page faults with pages from the list?
- Free list
  - When we need a page, take one from the free list
  - Have a daemon running the background, managing this free list — you can do this when system is not loaded
  - If size of free list gets too small, trigger the clock algorithm to add pages into the free list (by swapping out to disk)
  - Free list can be used as a disk cache

The page replacement policy proposed

- How many of following statements fit the page replacement policy that the paper implements?
  - ① It uses LRU ~~(least-recently-used) as~~ clock the page replacement policy
  - ② Page replacement policy are only triggered whenever ~~a page fault occurs~~ free list is under a threshold
  - ③ It attaches a ~~timestamp~~ reference bit to each page table entry instead of using the reference bit from hardware
  - ④ Processes are allocated a fixed set of pages and swap in/out to/from those pages
  - ⑤ The page replacement policy helps to guarantee the response time of short programs
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

The page replacement policy proposed

- How many of following statements fit the page replacement policy that the paper implements?
  - ① It uses LRU ~~(least-recently-used) as~~ clock the page replacement policy
  - ② Page replacement policy are only triggered whenever a ~~page fault occurs~~ free list is under a threshold
  - ③ It attaches a ~~timestamp~~ reference bit to each page table entry instead of using the reference bit from hardware
  - ④ ~~Processes are allocated a fixed set of pages and swap in/out~~ Process just get a page from the free list whenever it needs to/from those pages
  - ⑤ The page replacement policy helps to guarantee the response time of short programs

A. 0
B. 1
C. 2
D. 3
E. 4

formance implications. Lazowska [LAZO 79] reports that in his measurements based on a real workload, system performance was significantly improved by increasing the minimum size of the free list (a system generation parameter). An unfortunate

(ii) The projected workload for the system had no requirement of guaranteed response times as in real-time applications.

How to implement a simple shell?

- Say, we want to implement a shell that interprets command line commands and executes "./a"
- The following program can serve for this purpose:

```
int main(int argc, char *argv[]) {
    int child_pid;
    char cmd[1024];
    memset(cmd, 0 , 1024);
    fprintf(stderr,"CSC501-myshell$ ");
    while(fgets_wrapper(cmd,1024,stdin)) {
         if(strcmp("exit",cmd)==0)
             exit(1);
         child_pid = fork();
         if (child_pid == 0)
             execvp(cmd,NULL);
         else {
             fprintf(stderr,"CSC501-myshell$ ");
             memset(cmd, 0 , 1024);
         }
    }
    return 0;
}
```

- Do we actually need the code segment of the parent?

What happens on a fork?

- Create a new page table
- Copy data page-by-page
- 80% of fork occurs in shell command interpreter

Poll close in 1:30

virtual fork — vfork

- How many of the following statements regarding “vfork” is/are correct?
  - ① vfork can improve the response time of the parent process
  - ② vfork does not create a new address space upon the creation of new process
  - ③ vfork allows the child process to execute within the parent’s memory address
  - ④ The child process created by vfork can potentially corrupt the parent process
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

Poll close in 1:30

virtual fork — vfork

- How many of the following statements regarding “vfork” is/are correct?
  - ① vfork can improve the response time of the parent process
  - ② vfork does not create a new address space upon the creation of new process
  - ③ vfork allows the child process to execute within the parent’s memory address
  - ④ The child process created by vfork can potentially corrupt the parent process
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

virtual fork — vfork

- How many of the following statements regarding “vfork” is/are correct?
  - ① vfork can ~~improve~~ the response time of the parent process
    hurt, because it delays the parent until exec()
  - ② vfork does not create a new address space upon the creation of new process
  - ③ vfork allows the child process to execute within the parent’s memory address
  - ④ The child process created by vfork can potentially corrupt the parent process

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4

WSClcok - A Simple and Effective Algorithm for Virtual Memory Management

Richard Carr and John Hennessy

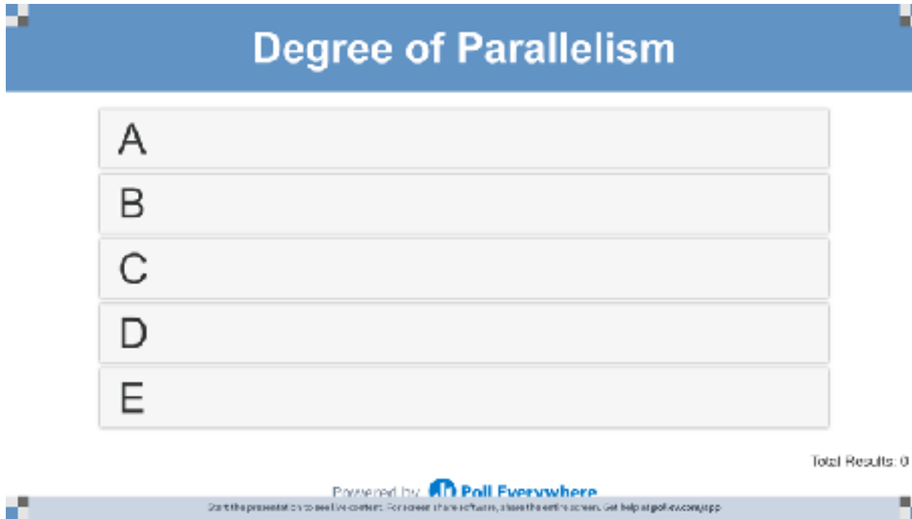
Brief recap: what policies are used?

- Local: select one page from the same process' physical pages for storing the demanding page when swapping is necessary
  - VAX/VMS
  - Original UNIX
- Global: select any page that was previously belong to any process when swapping is necessary
  - UNIX after Babaoglu
  - Mach

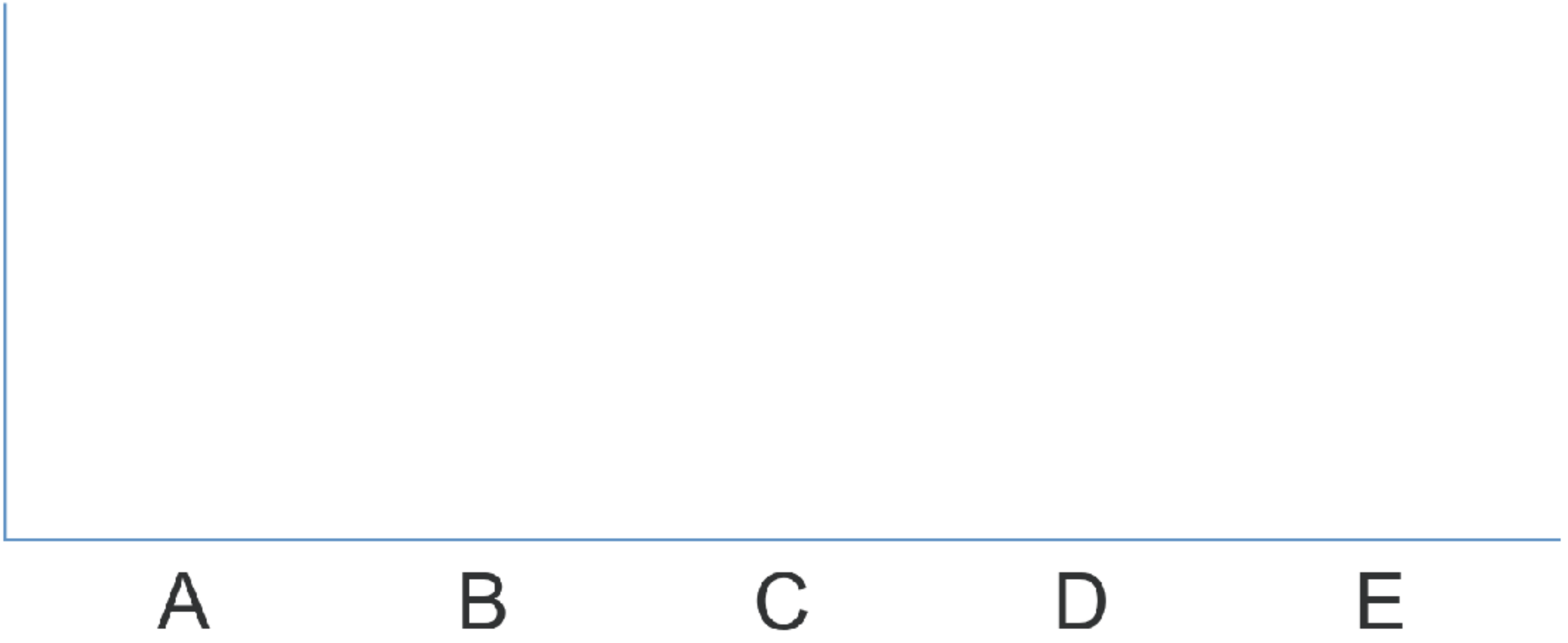
Degree of parallelism and performance

accomplish most tasks-- processes are cheap. These

- How many of the following would happen in Babaoglu's UNIX VM if we keep increase the amount of concurrent processes and assume each process uses some virtual memory in the system?
  - ① The CPU utilization will keep increasing and stay at 100%
  - ② The system may spend more time in context switching than real computation
  - ③ The system may spend more time in swap in/out than real computation
  - ④ Some process may not respond due to the high paging overhead
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4



Degree of Parallelism



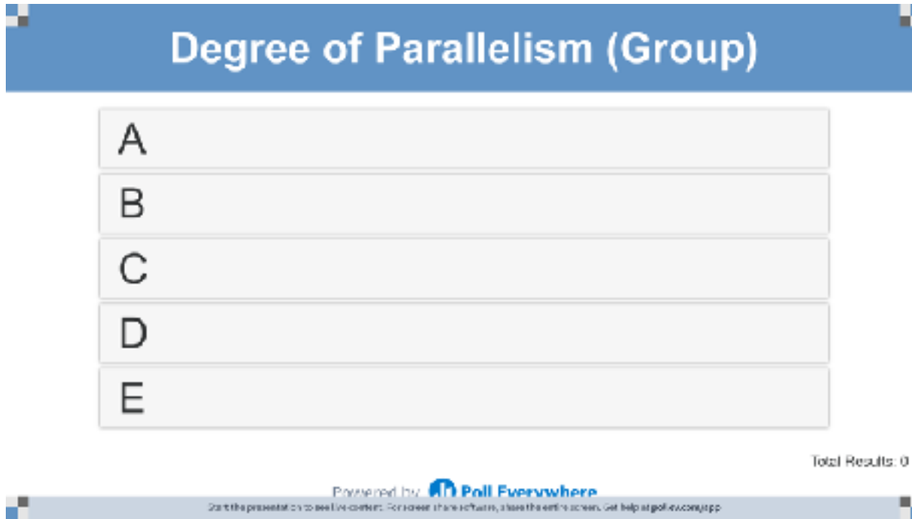
Total Results: 0

https://www.pollev.com/hungweitseng close in 1:00

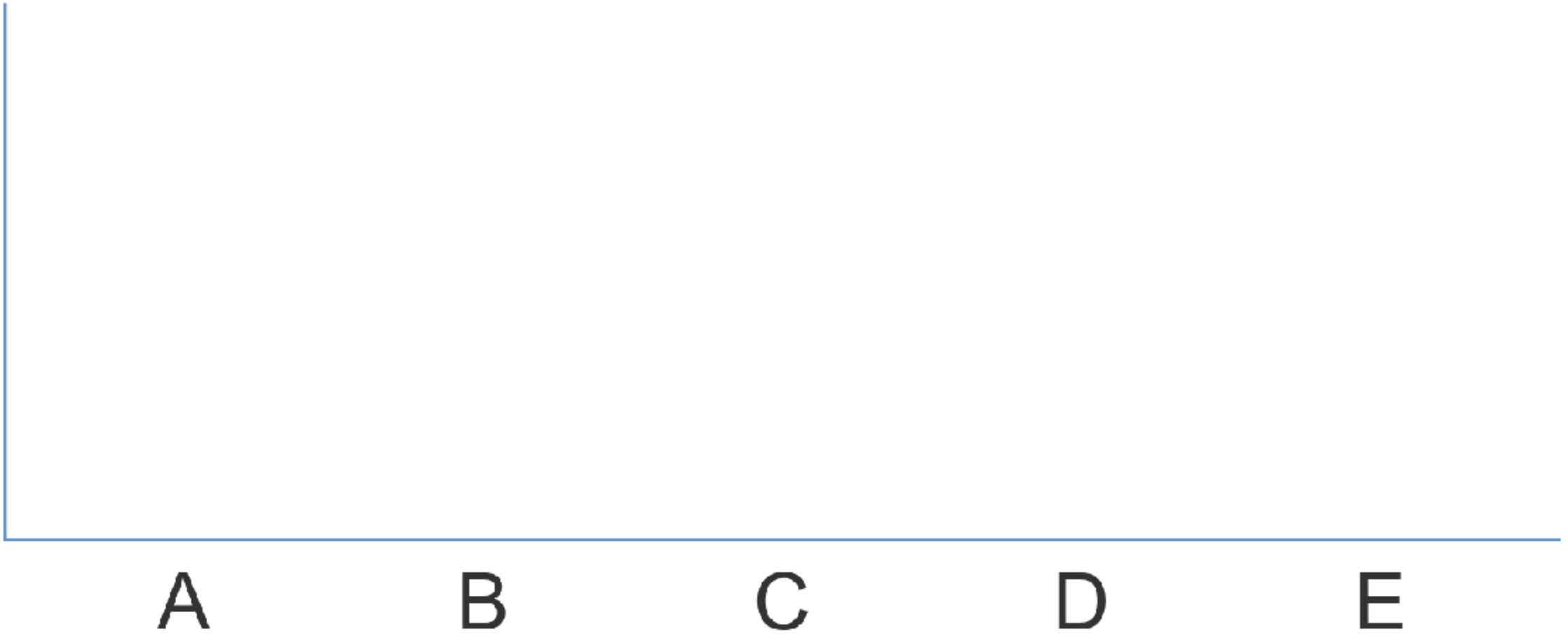
Degree of parallelism and performance

- How many of the following would happen in Babaoglu's UNIX VM if we keep increase the amount of concurrent processes and assume each process uses some virtual memory in the system?
  - ① The CPU utilization will keep increasing and stay at 100%
  - ② The system may spend more time in context switching than real computation
  - ③ The system may spend more time in swap in/out than real computation
  - ④ Some process may not respond due to the high paging overhead

  A. 0
  B. 1
  C. 2
  D. 3
  E. 4



Degree of Parallelism (Group)

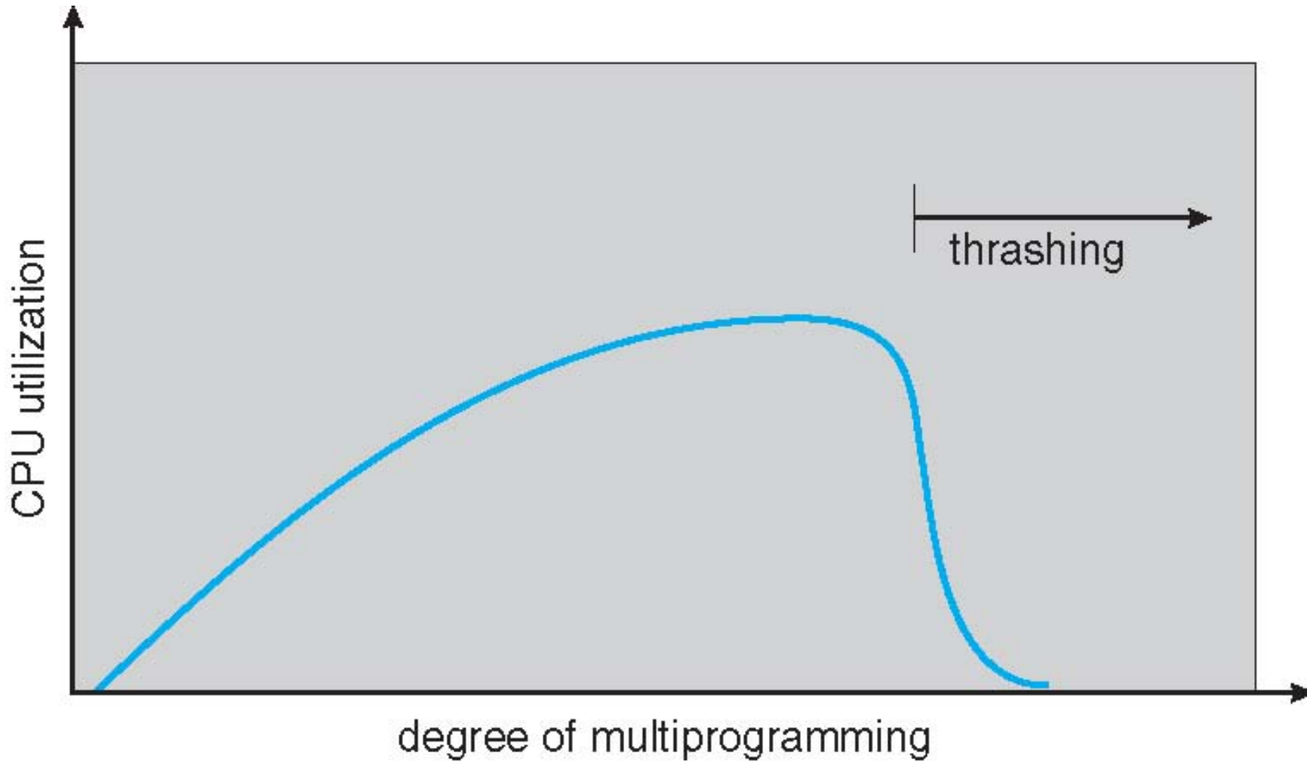


Total Results: 0

Thrashing: Paging overhead

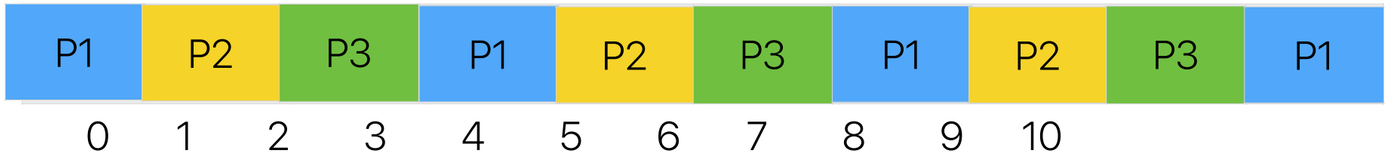
- The system overcommitted memory to tasks
- The system spends most time in paging, instead of making meaningful progress

Previously, we have seen how scheduling policies can help improving "saturation". Now, let's see how page replacement policies can address this "thrashing"



Saturation: Context Switch Overhead

You think round robin should act like this —



But the fact is —

- **Your processor utilization can be very low if you switch frequently**
- **No process can make sufficient amount of progress within a given period of time**
- **It also takes a while to reach your turn**

Thrashing v.s. Saturation

- Thrashing — when memory are overcommitted
  - The system is busy paging
  - The processor is idle waiting
- Saturation — when processors are overcommitted
  - The system is busy context switching and scheduling
  - The processor is busy but not contributing to the running program

Degree of parallelism and performance

- How many of the following would happen in Babaoglu's UNIX VM if we keep increase the amount of concurrent processes and assume each process uses some virtual memory in the system?
  - ① The CPU utilization will keep increasing and stay at 100%
  - ② The system may spend more time in context switching than real computation
  - ③ The system may spend more time in swap in/out than real computation
  - ④ Some process may not respond due to the high paging overhead
  - A. 0
  - B. 1
  - C. 2
  - D. 3
  - E. 4

Why WS-Clock

- Take advantages from both local and global page replacement policies
  - Global — simplicity, adaptive to process demands
  - Local — prevent thrashing

Working Set Algorithm

- Working set: the set of pages used in a certain number of recent accesses
- Assume these recently referenced pages are likely to be referenced again soon (temporal locality)
- Evict pages that are not referenced in a certain period of time
  - Swap out may occur even if there is no page faults
- A process is allowed to be executed only if the working set size fits in the physical memory

WSClock

- Use **working set** policy to decide how many pages can a process use
  - Return a page to the free list if there exists a page in the process' working set that hasn't been access for a certain period of time
- If the free list is lower than a threshold
  - Trigger the **clock** policy to select pages from any process
- On a page fault
  - Take a page from the free list

WSClock

- Wherever you need to reclaim a page —
    1. Examine the PTE pointed to by clock hand.
    2. If reference bit is set
        1. Clear reference bit;
        2. Advance clock hand;
        3. Goto Done.
    3. If reference bit is not set
        1. If the timestamp of the PTE is older than a threshold
            1. Write the page to disk if it's dirty and use this page
            2. Goto Done
        2. Otherwise
            1. Advance clock hand
            2. Goto 1.
    4. Done
    5. If no victim page is chosen, randomly pick one

The impact of WSClock

- One of the most important page replacement policies in practice

Announcement

- No lecture this Thursday — relocate those 80 minutes to anytime you like before **2/11 11:59:00pm**.
- Reading quizzes due next **Tuesday**
- Start working on your project **ASAP**

Computer
Science &
Engineering

202

つづく