Virtual Memory (IV) — Policies

Hung-Wei Tseng

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

Swapping policies

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

	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

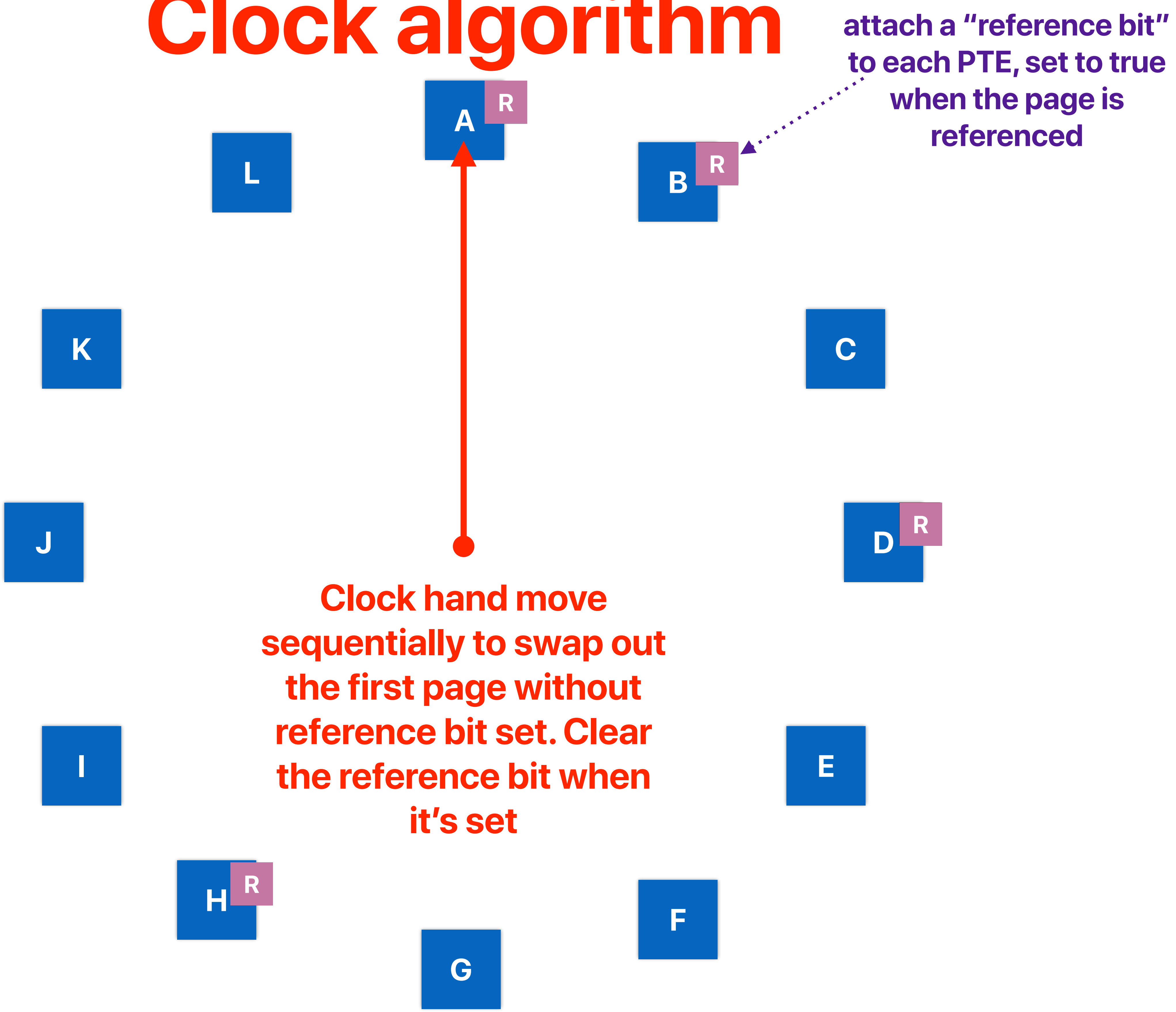
Converting a Swap-Based System to do Paging in an Architecture Lacking Page-Reference Bits

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

The Why of Babaoglu new UNIX VM

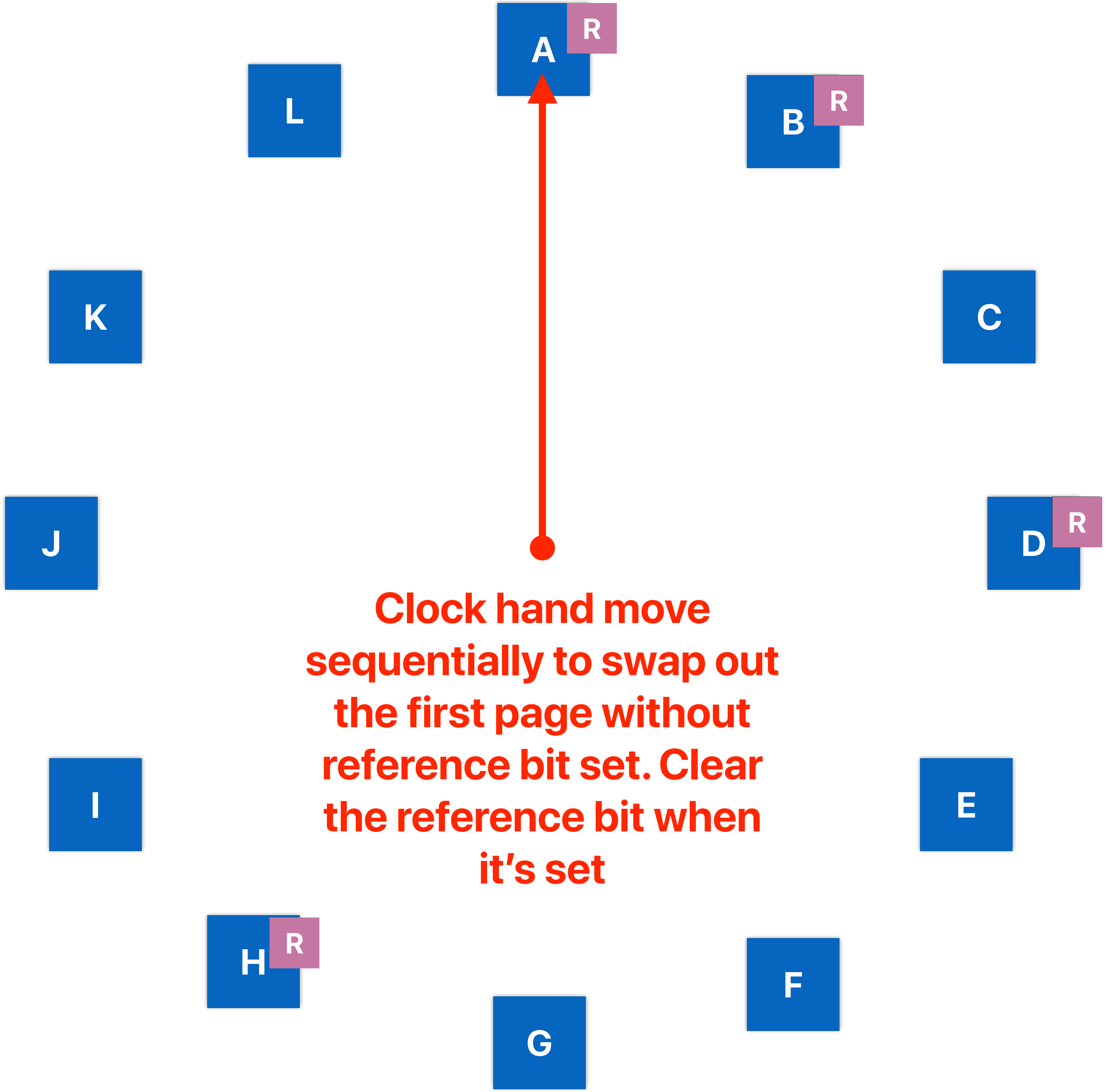
- The original UNIX is a “swap-based” system
  - Whenever you have a context switch, swap the whole process out from the memory
  - Really inefficient if you have frequent context switches or if you have many applications in-fly
- Efficient page replacement policies and other virtual optimization techniques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

Clock algorithm



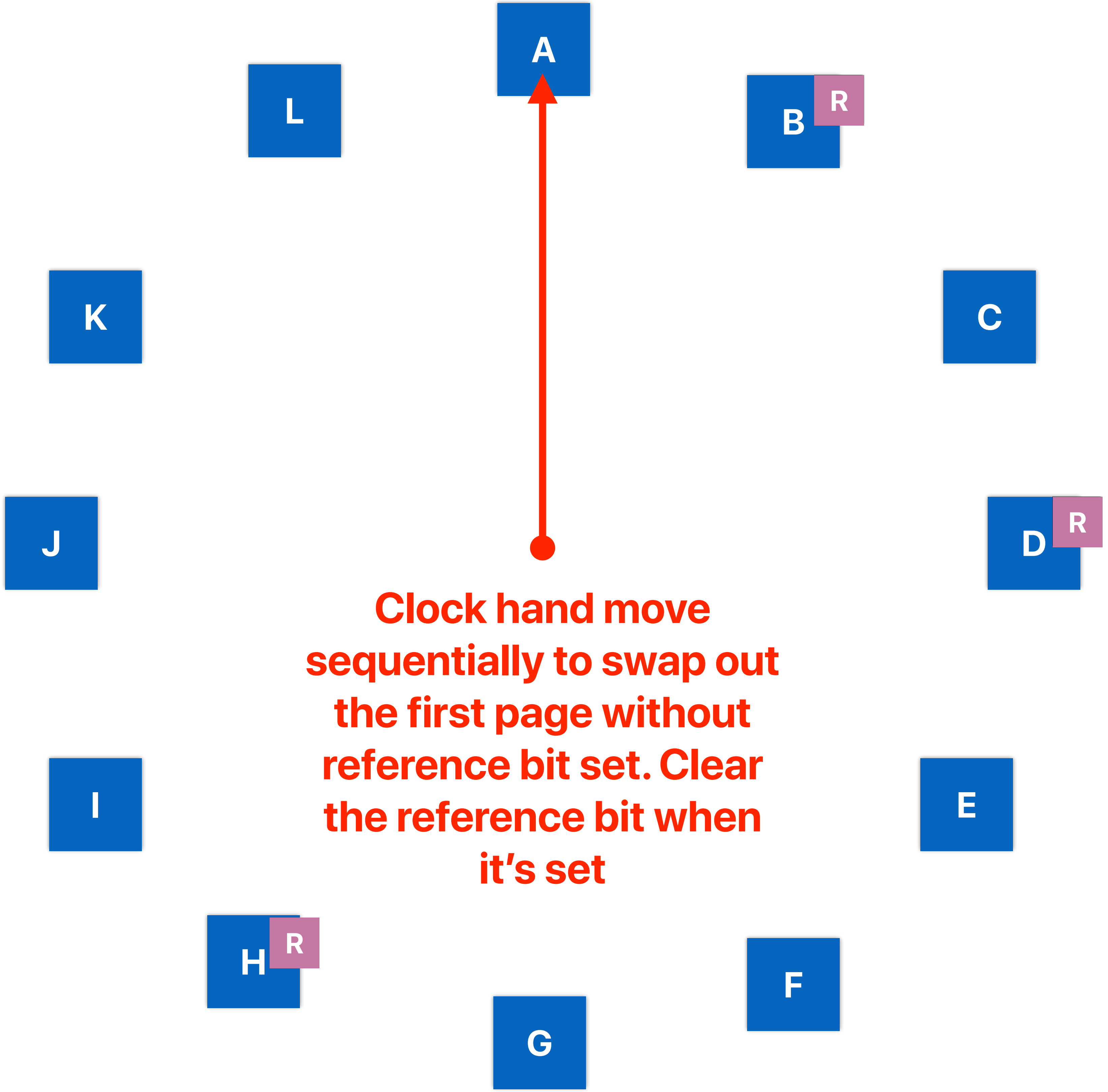
Clock algorithm in motion

Where to put M ?



Clock algorithm in motion

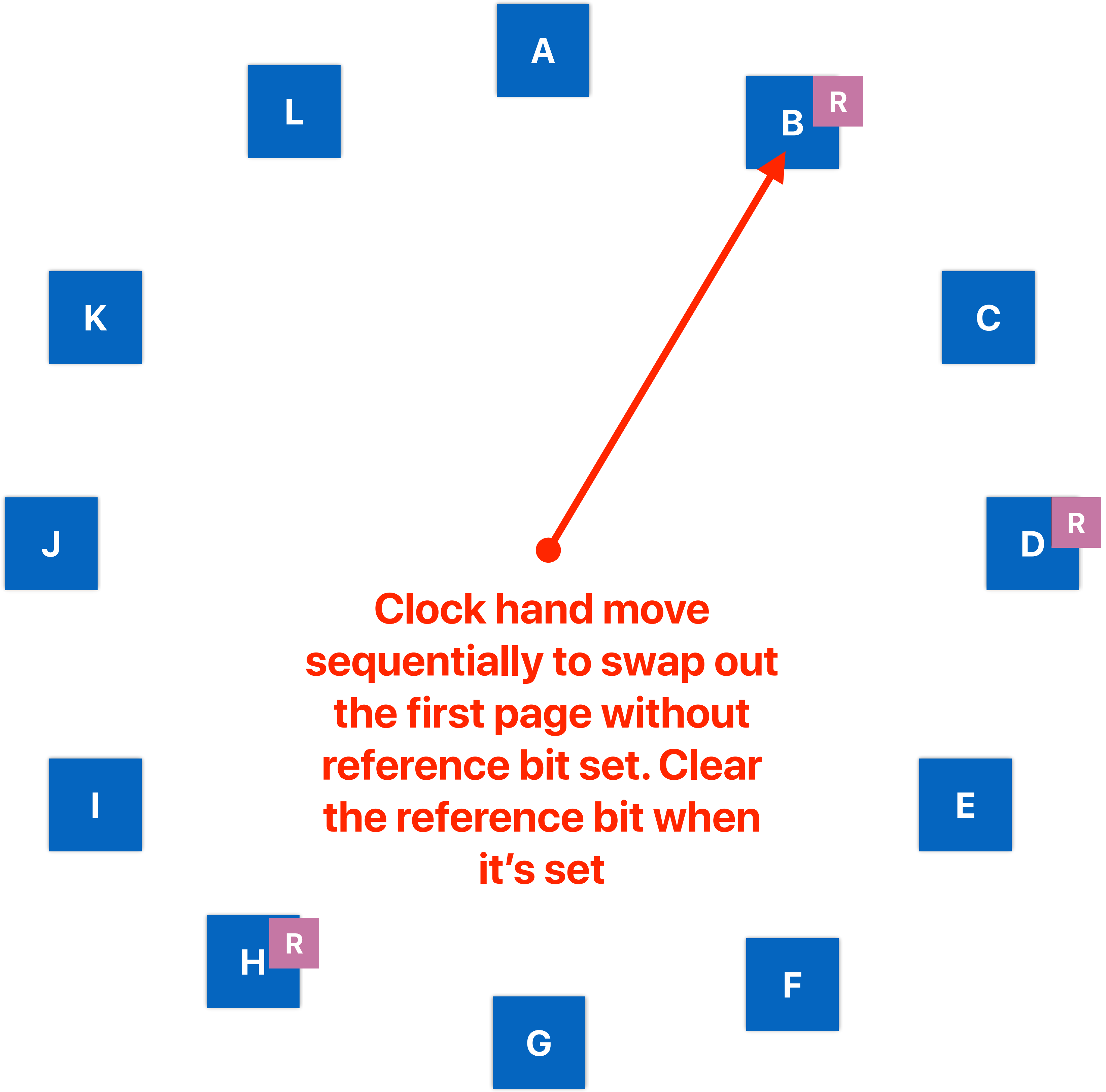
Where to put M ?



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

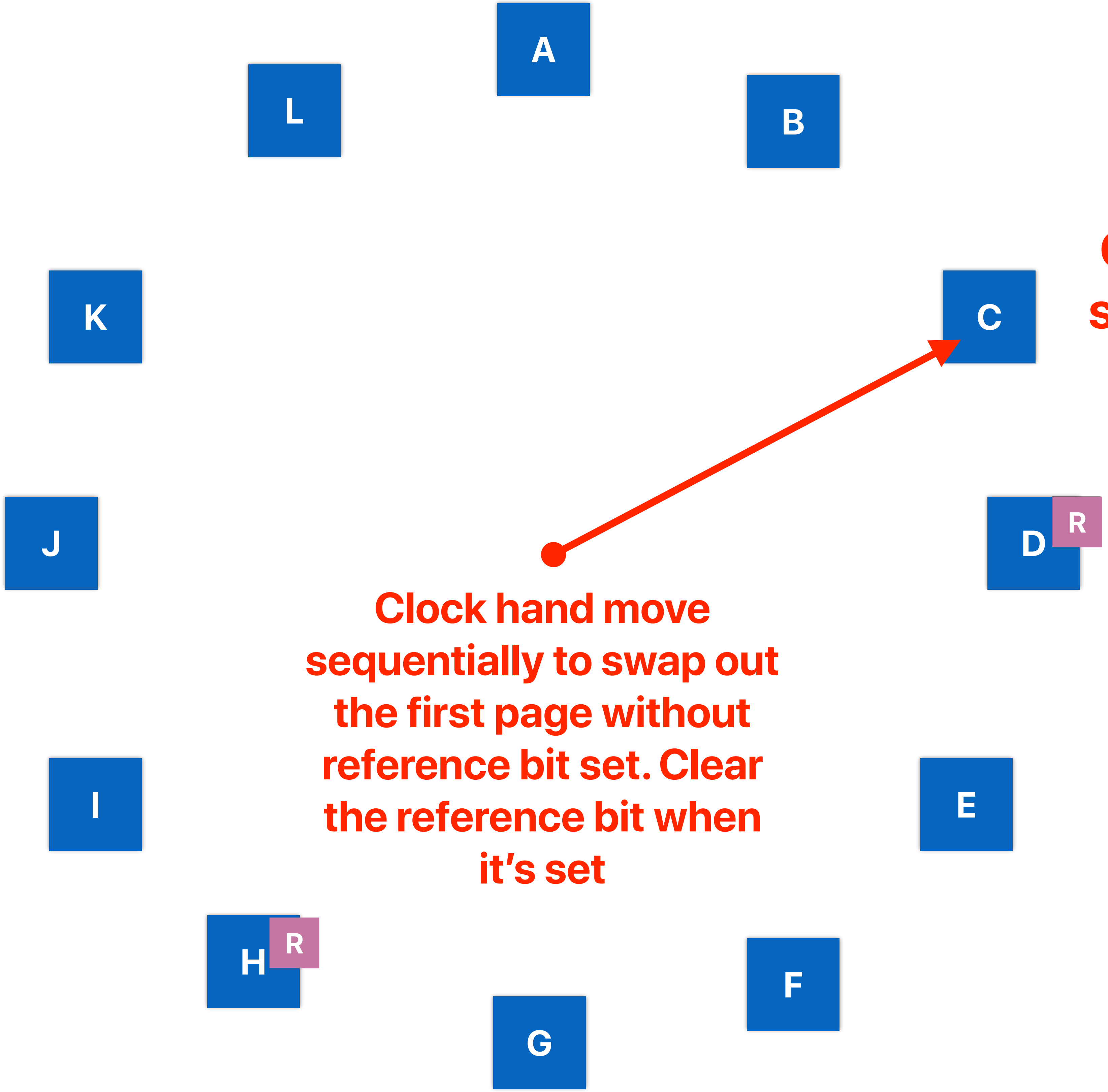
Clock algorithm in motion

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

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?

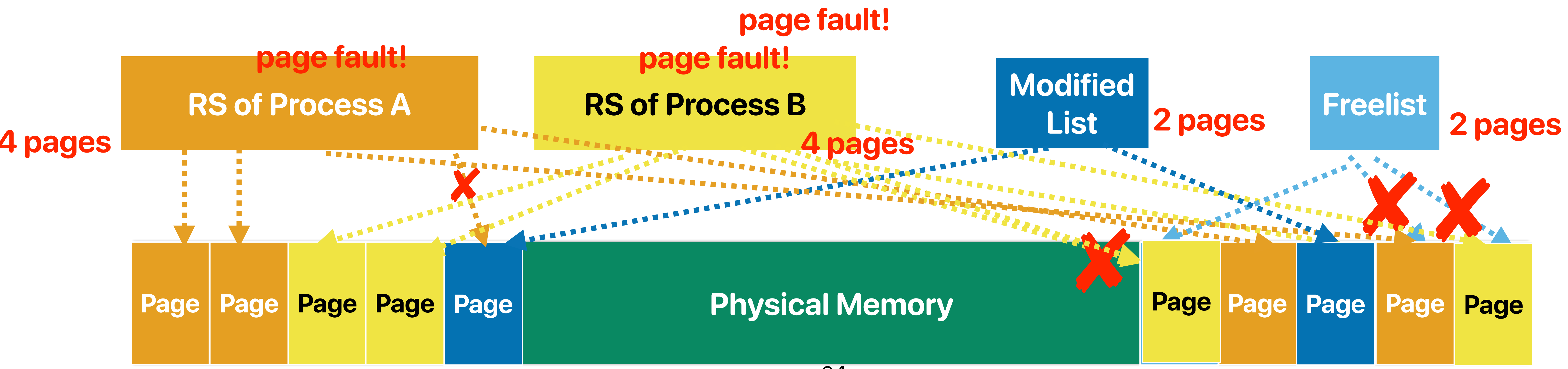
	0	1	2	3	4	5	6	7	8	9	10	11
Page #	2	3	2	1	5	2	4	5	3	2	5	2

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

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

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



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

What happens on a fork?

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

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?

WSClcok - A Simple and Effective Algorithm for Virtual Memory Management

Richard Carr and John Hennessy

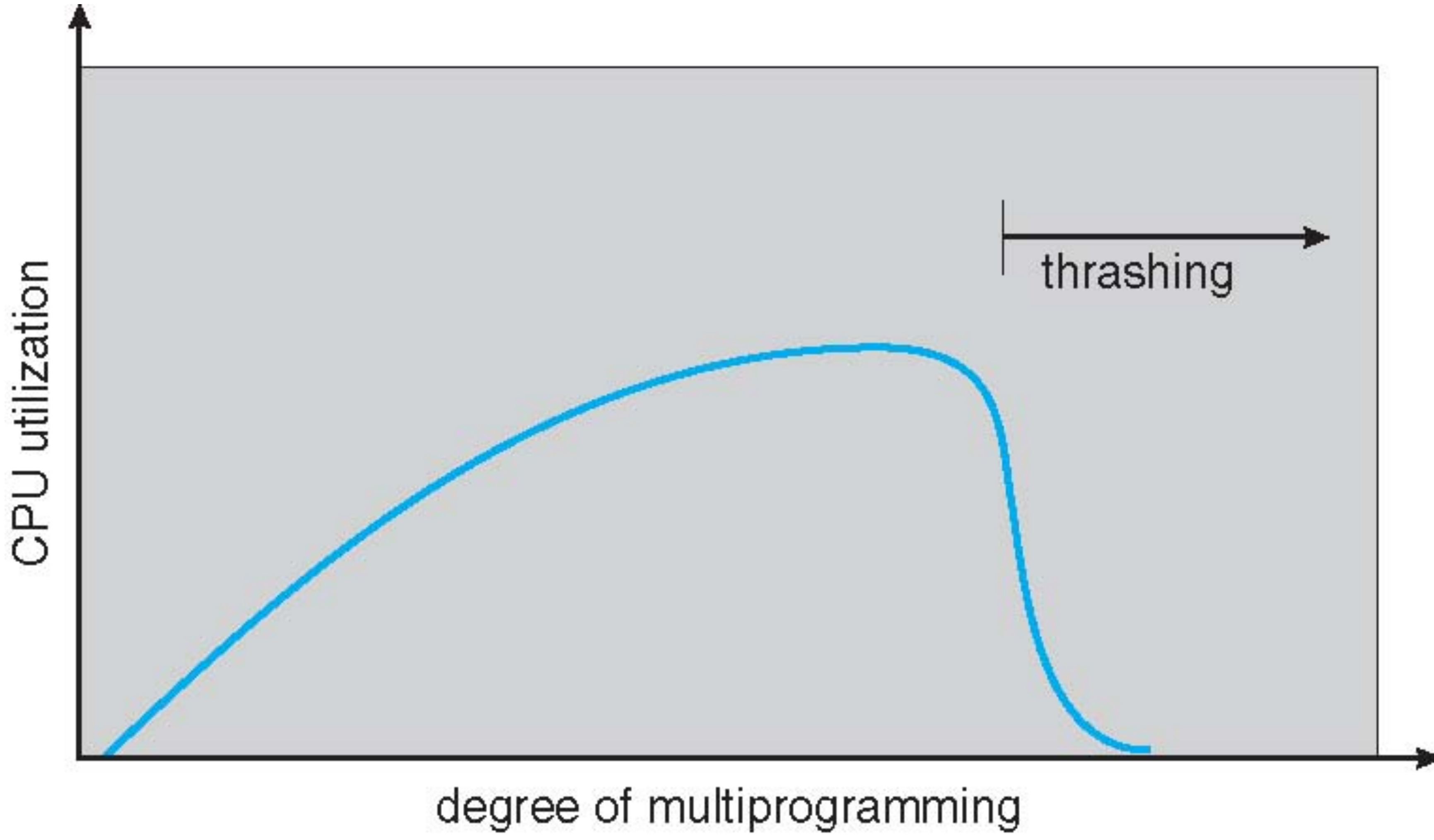
Brief recap: what policies are used?

- 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

Thrashing!

- 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"



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

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