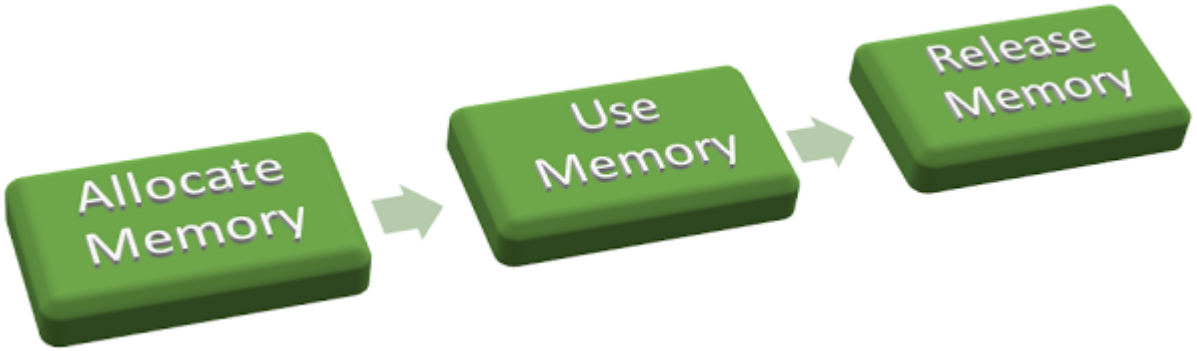
Memory Management. Part 3



MOS4E

https://dzone.com/articles/memory-management-and-garbage-collection-in-javasc

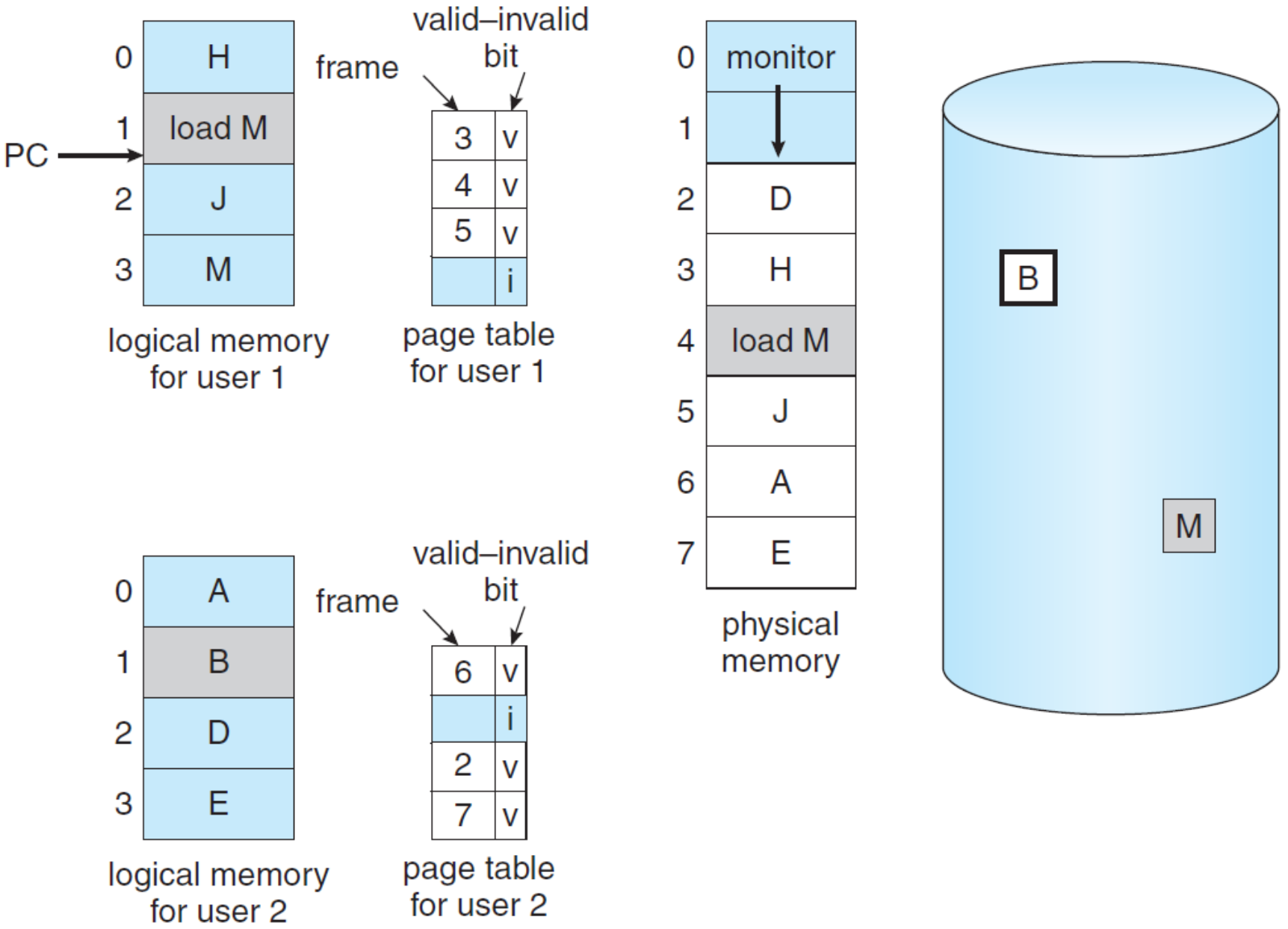


Figure 9.9 Need for page replacement.

OSC9E

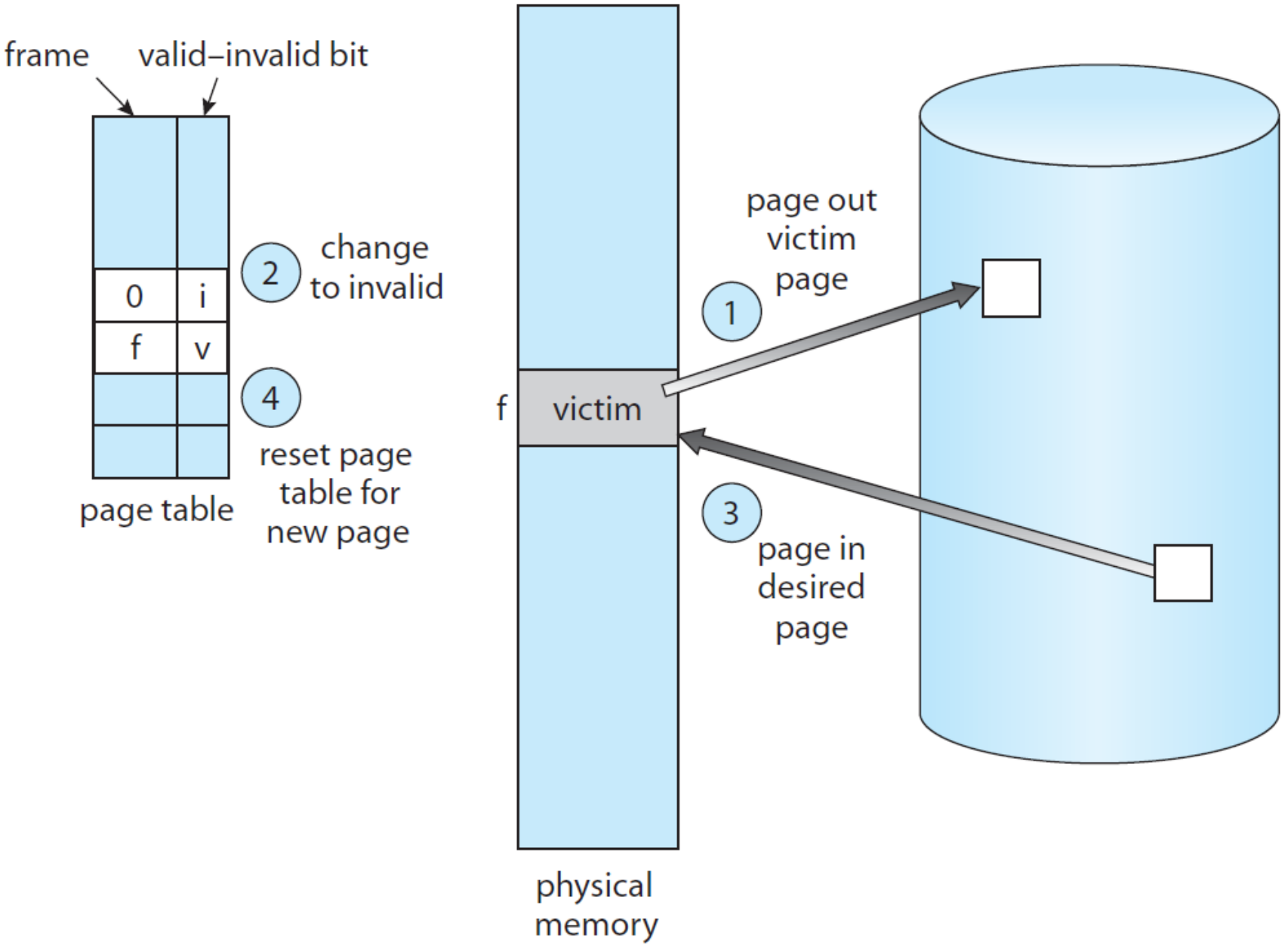


Figure 9.10 Page replacement.

“We evaluate a page-replacement algorithm by running it on a particular string of memory references and computing the number of page faults. The string of memory references is called a reference string.

We can generate reference strings artificially (by using a random-number generator, for example), or we can trace a given system and record the address of each memory reference. The latter choice produces a large number of data (on the order of 1 million addresses per second). To reduce the number of data, we use two facts.

…”

OSC9E

“ … First, for a given page size (and the page size is generally fixed by the hardware or system), we need to consider only the page number, rather than the entire address.

Second, if we have a reference to a page *p*, then any references to page *p* that *immediately* follow will never cause a page fault. Page *p* will be in memory after the first reference, so the immediately following references will not fault.”

OSC9E

“For example, if we trace a particular process, we might record the following address sequence:

0100, 0432, 0101, 0612, 0102, 0103, 0104, 0101, 0611, 0102, 0103, 0104, 0101, 0610, 0102, 0103, 0104, 0101, 0609, 0102, 0105

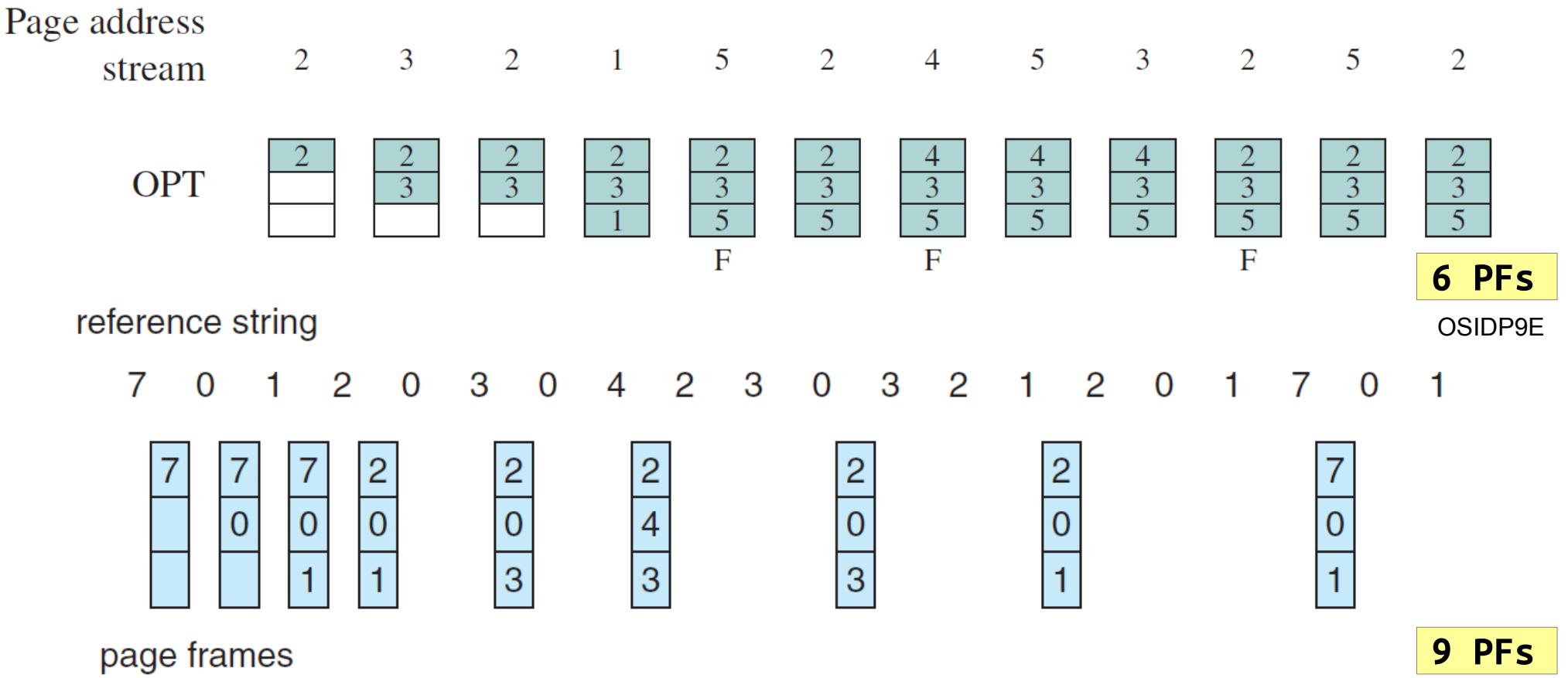
At 100 bytes per page, this sequence is reduced to the following reference string:

1, 4, 1, 6, 1, 6, 1, 6, 1, 6, 1”

OSC9E

The Optimal Page Replacement Algorithm:

Replace the page that will not be used for the longest period of time.



OSIDP9E

Figure 9.14 Optimal page-replacement algorithm.

OSC9E

The Not Recently Used Page Replacement Algorithm

In order to allow the operating system to collect useful page usage statistics, most computers with virtual memory have two status bits, *R* and *M*, associated with each page. *R* is set whenever the page is referenced (read or written). *M* is set when the page is written to (i.e., modified). The bits are contained in each page table entry. It is important to realize that these bits must be updated on every memory reference, so it is essential that they be set by the hardware. Once a bit has been set to 1, it stays 1 until the operating system resets it.

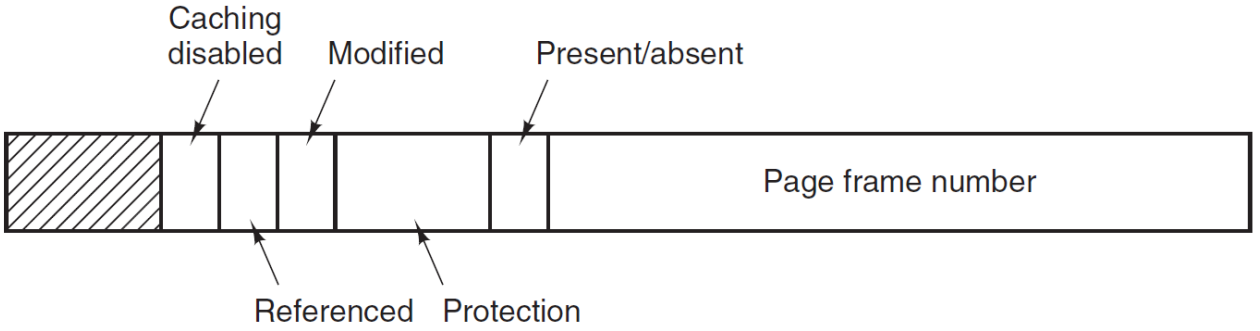


Figure 3-11. A typical page table entry.

MOS4E

The Not Recently Used Page Replacement Algorithm

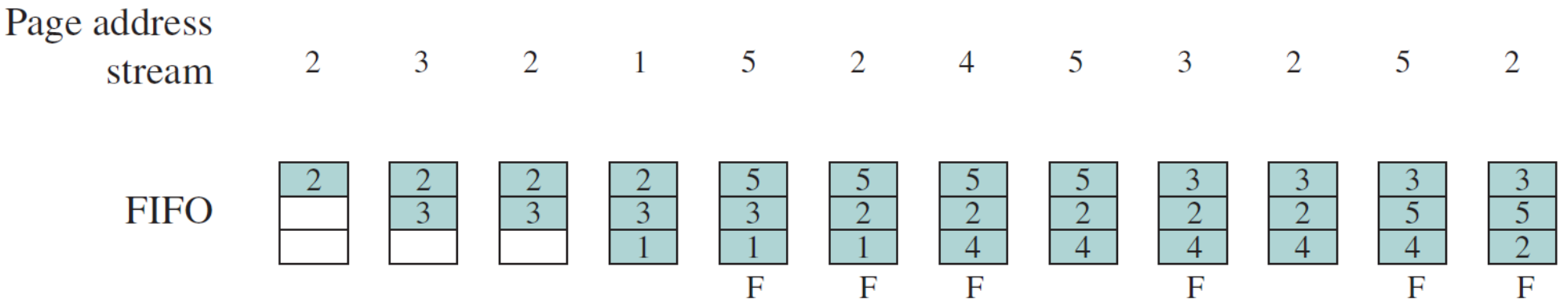
The *R* and *M* bits can be used to build a simple paging algorithm as follows. When a process is started up, both page bits for all its pages are set to 0 by the operating system. Periodically (e.g., on each clock interrupt), the *R* bit is cleared, to distinguish pages that have not been referenced recently from those that have been. When a page fault occurs, the operating system inspects all the pages and divides them into four categories based on the current values of their *R* and *M* bits:

Class 0: not referenced, not modified.
Class 1: not referenced, modified.
Class 2: referenced, not modified.
Class 3: referenced, modified.

The NRU (Not Recently Used) algorithm removes a page at random from the lowest-numbered nonempty class.

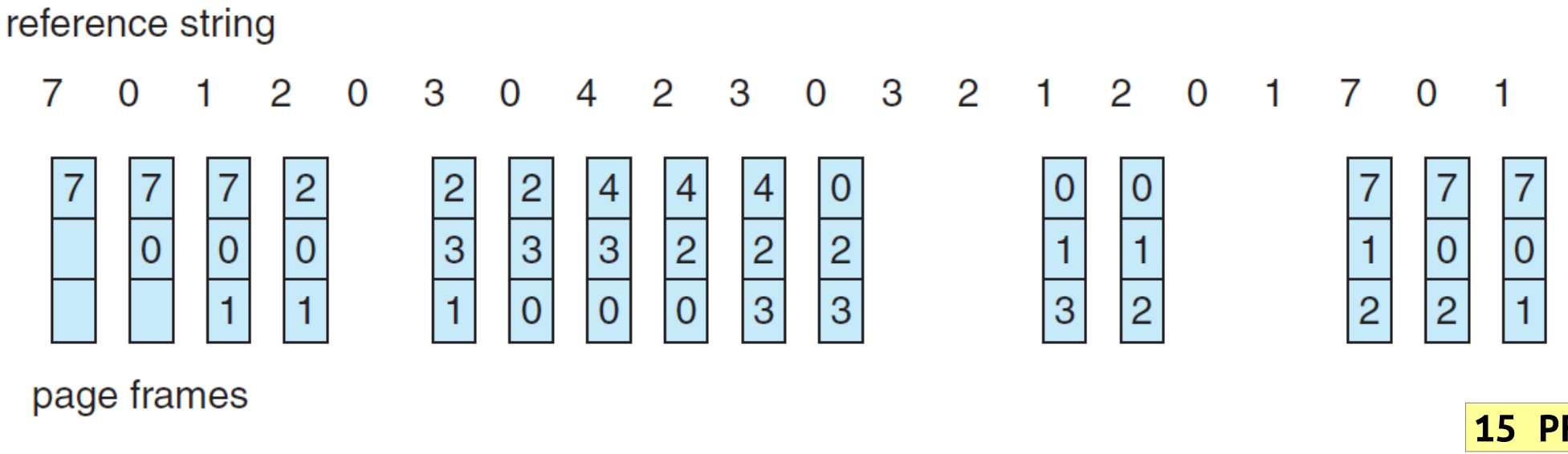
MOS4E

The FIFO Page Replacement Algorithm



9 PFs

OSIDP9E



15 PFs

Figure 9.12 FIFO page-replacement algorithm.

OSC9E

Belady's Anomaly

Consider the following reference string:

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

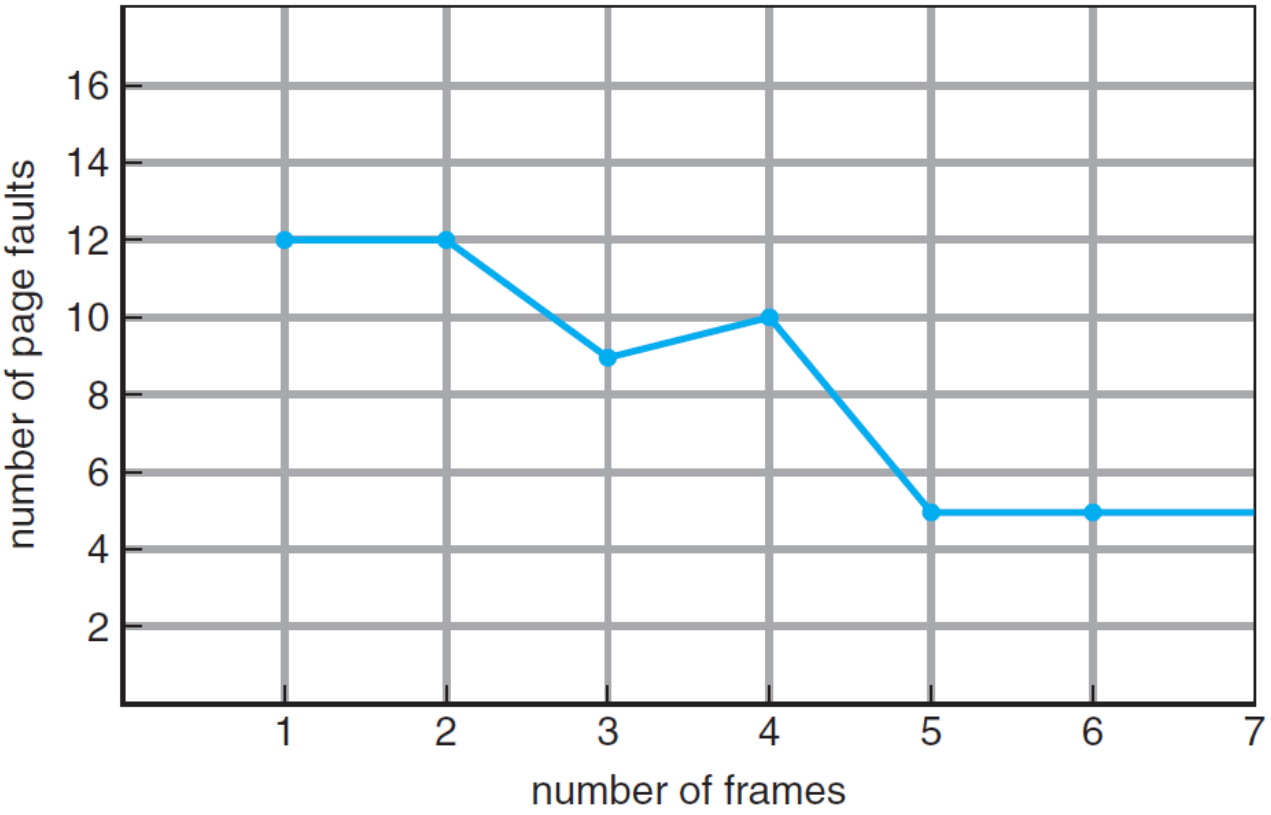


Figure 9.13 Page-fault curve for FIFO replacement on a reference string.

OSC9E

The Second-Chance Page Replacement Algorithm

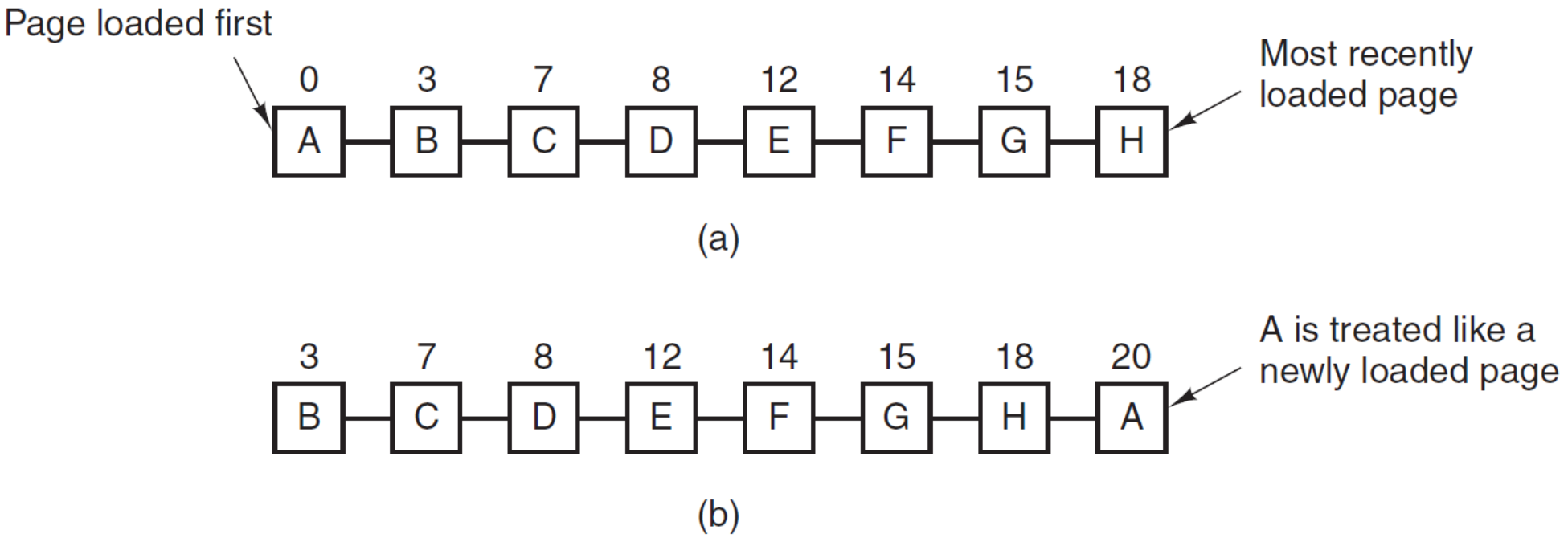


Figure 3-15. Operation of second chance. (a) Pages sorted in FIFO order. (b) Page list if a page fault occurs at time 20 and *A* has its *R* bit set. The numbers above the pages are their load times.

MOS4E

The Second-Chance Page Replacement Algorithm

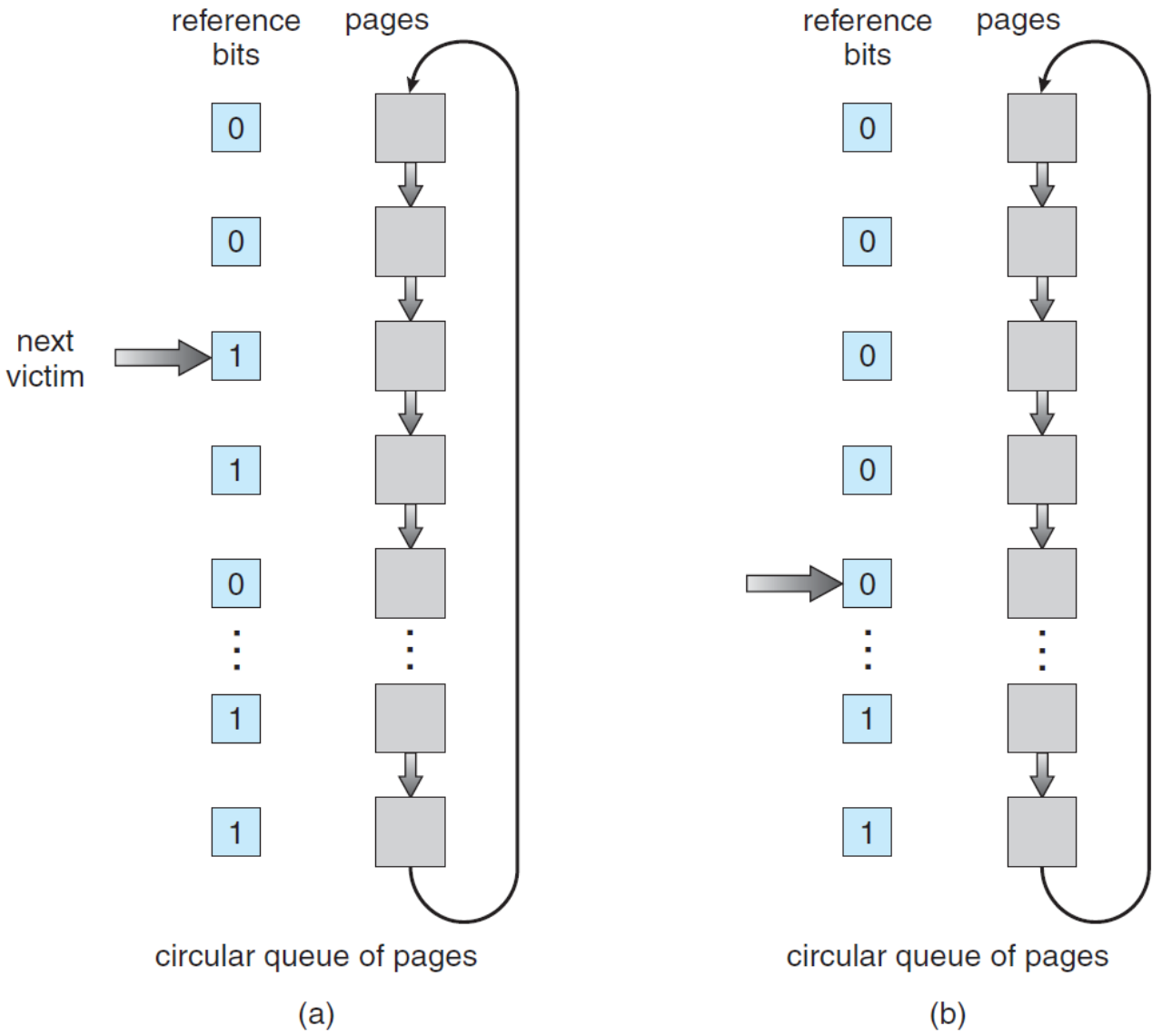
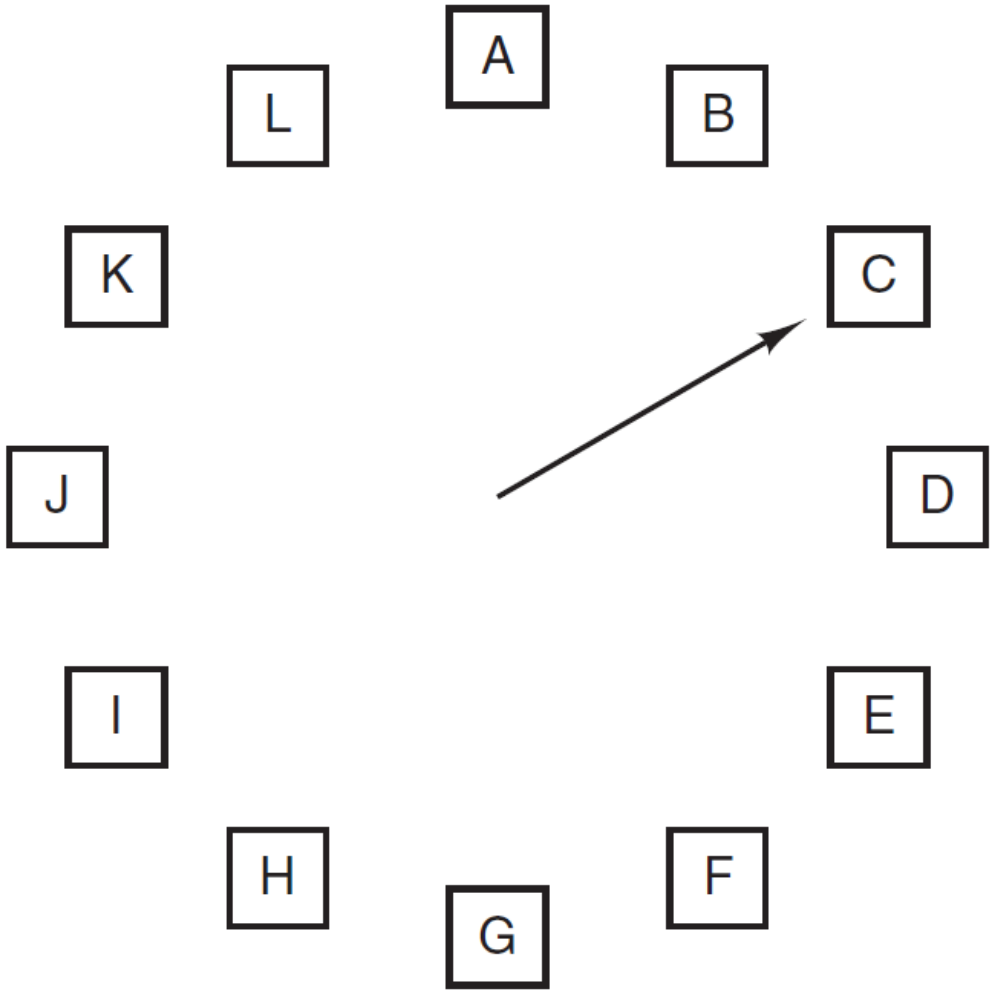


Figure 9.17 Second-chance (clock) page-replacement algorithm.

OSC9E

The Clock Page Replacement Algorithm



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

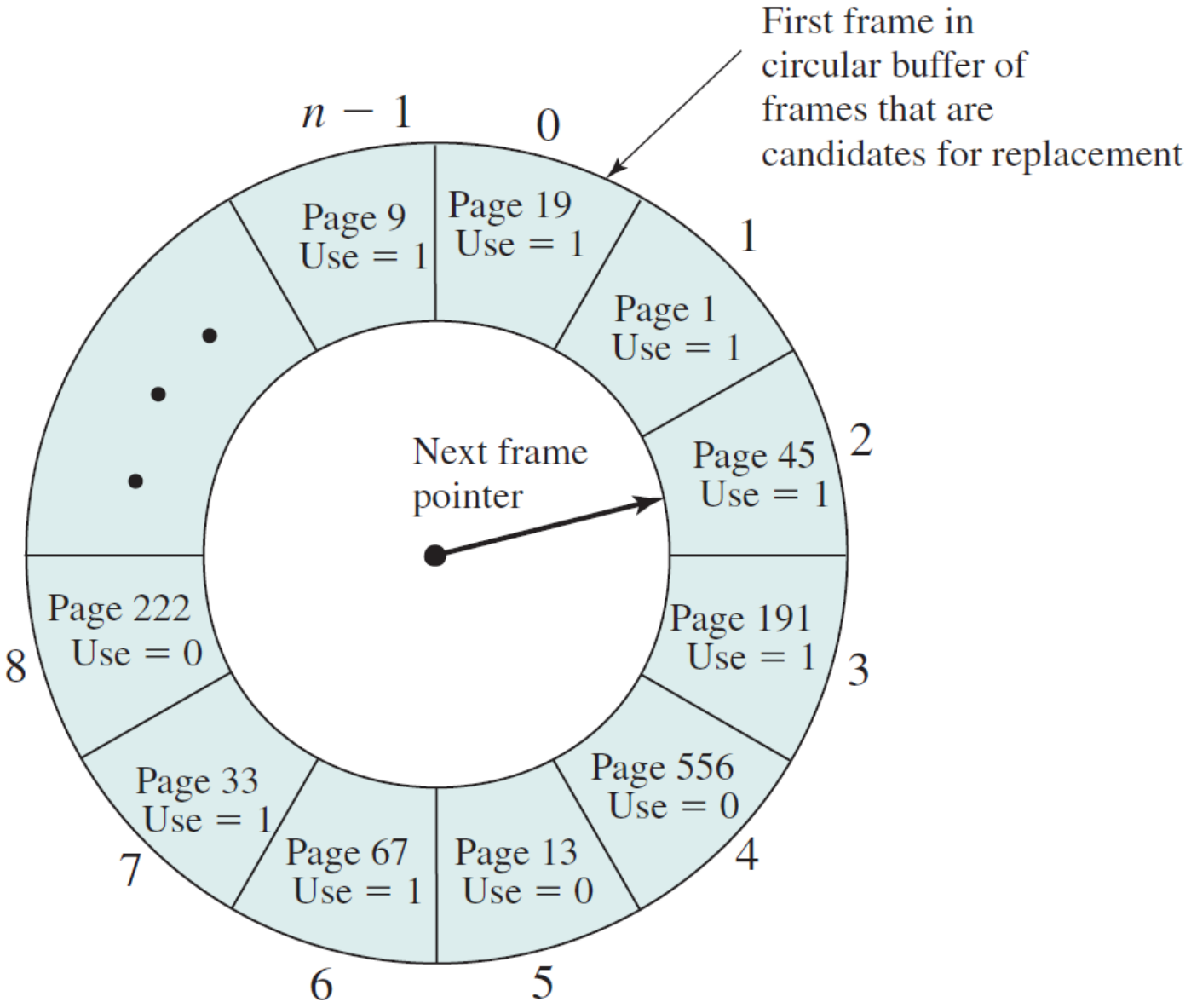
R = 0: Evict the page

R = 1: Clear R and advance hand

Figure 3-16. The clock page replacement algorithm.

MOS4E

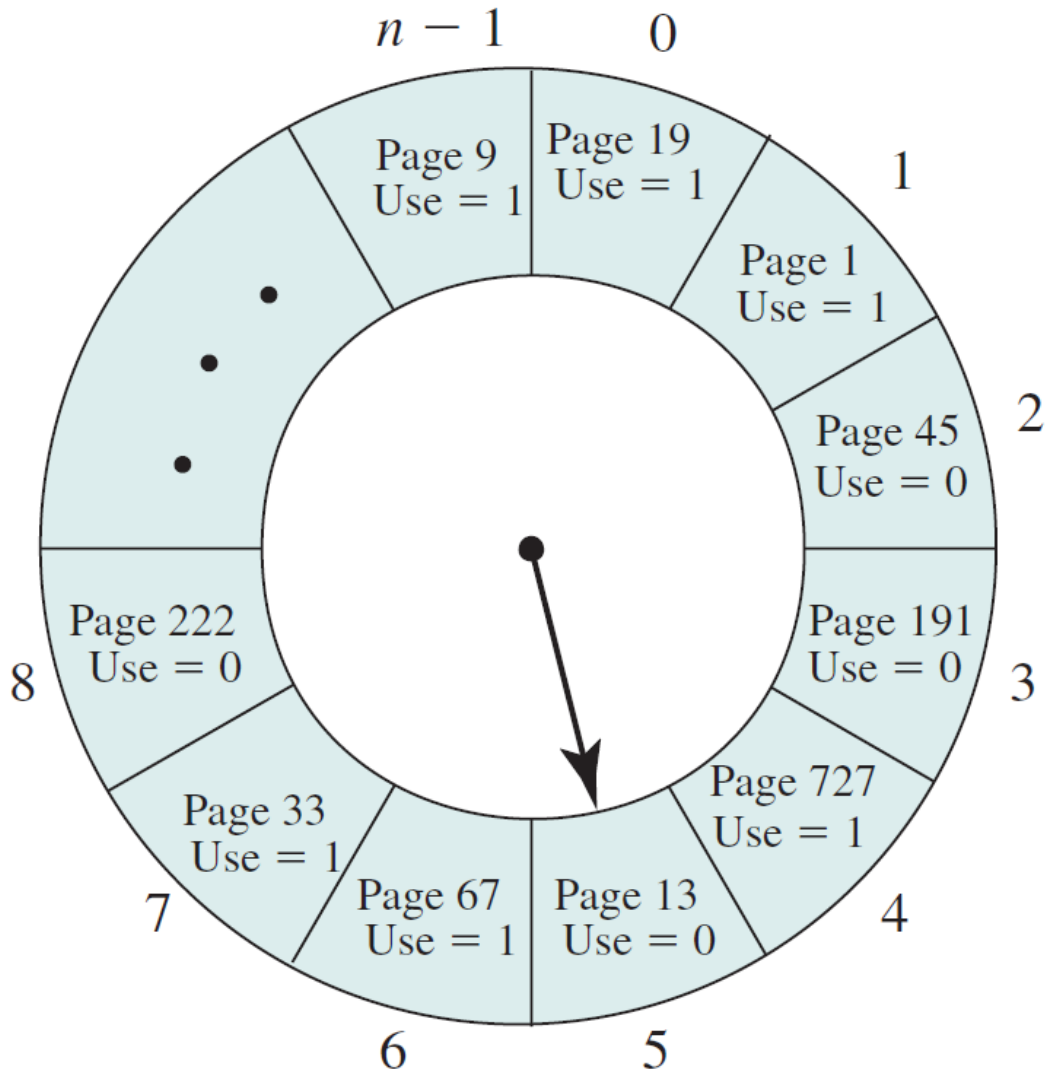
The Clock Page Replacement Algorithm



(a) State of buffer just prior to a page replacement

OSIDP9E

The Clock Page Replacement Algorithm

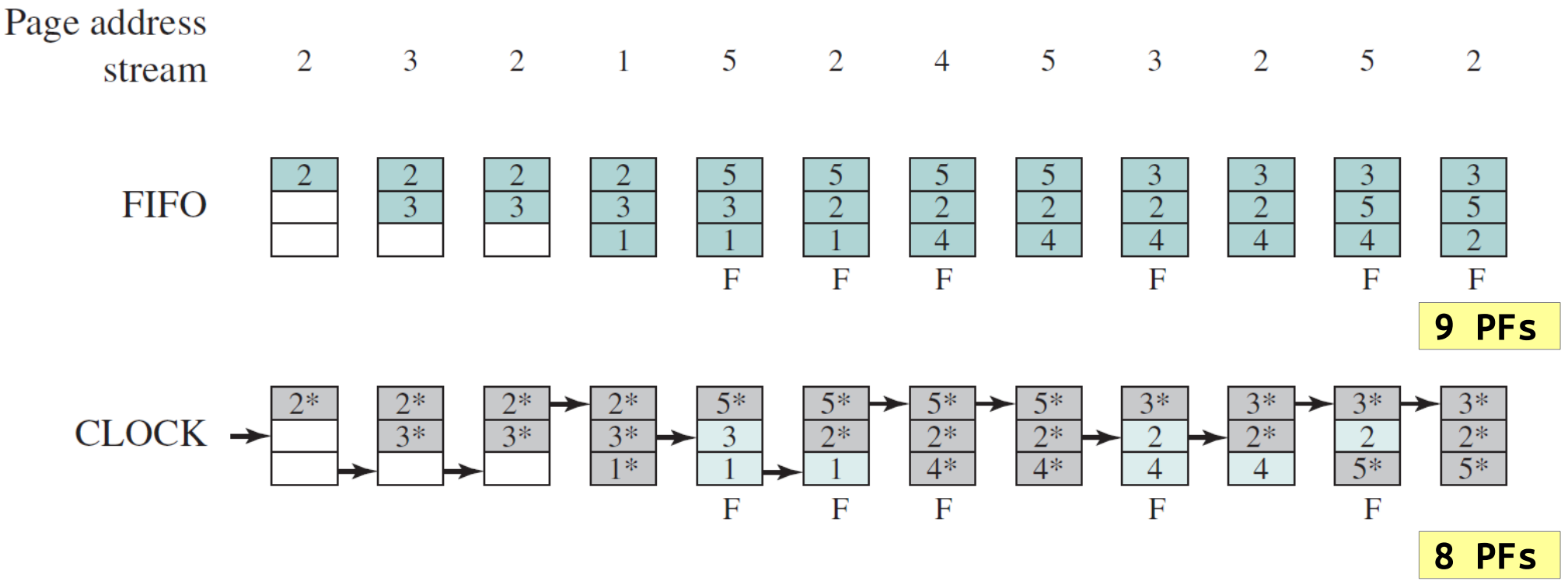


(b) State of buffer just after the next page replacement

Figure 8.15 Example of Clock Policy Operation

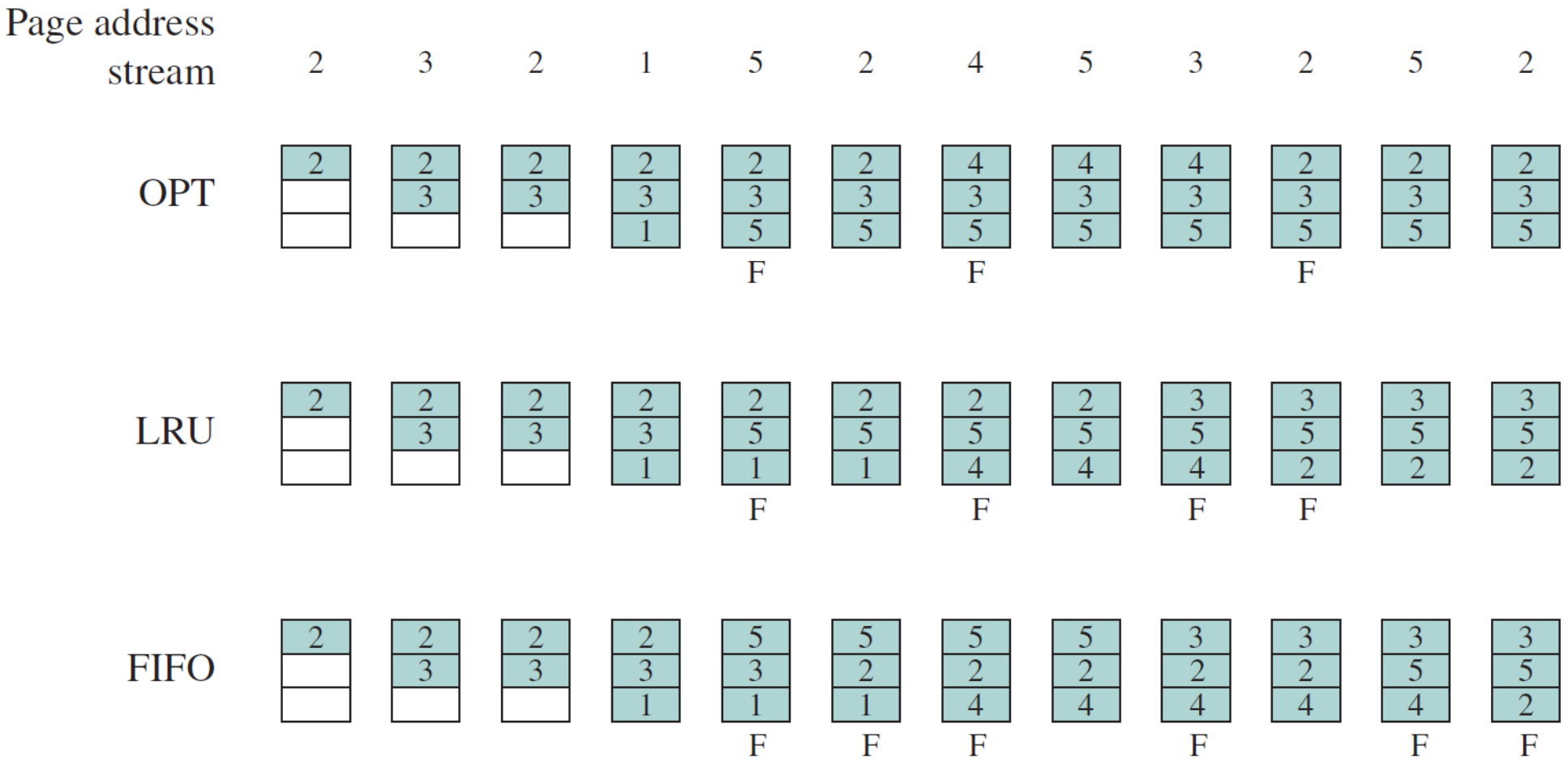
OSIDP9E

The Clock Page Replacement Algorithm



OSIDP9E

The Least Recently Used Page Replacement Algorithm



OSIDP9E

The Least Recently Used Page Replacement Algorithm

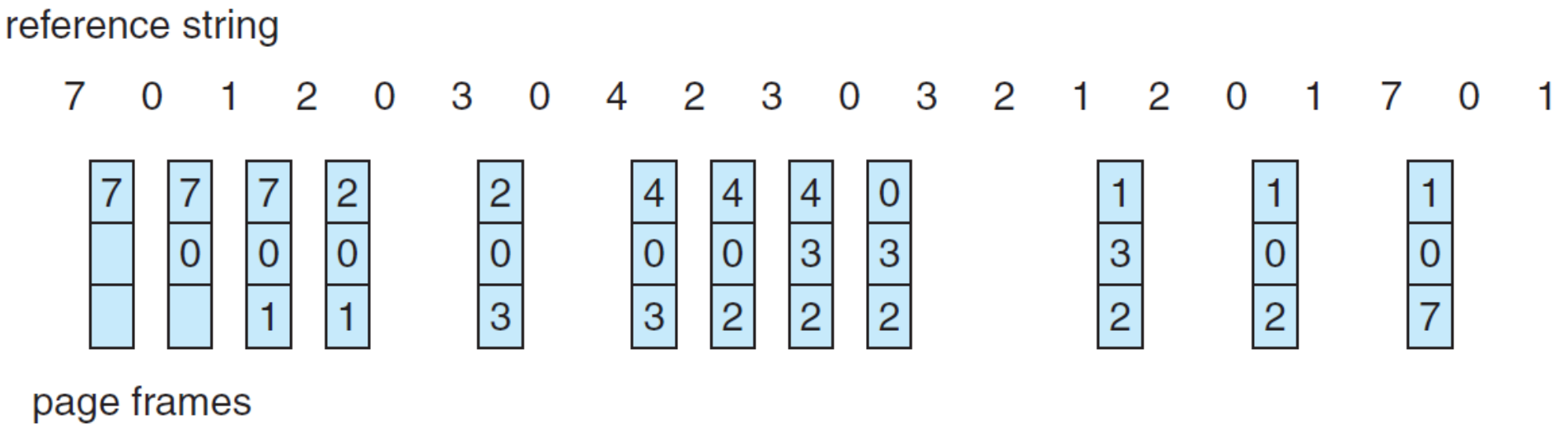


Figure 9.15 LRU page-replacement algorithm.

LRU: 12 PFs
vs
FIFO: 15 PFs

OSC9E

The LRU Implementation

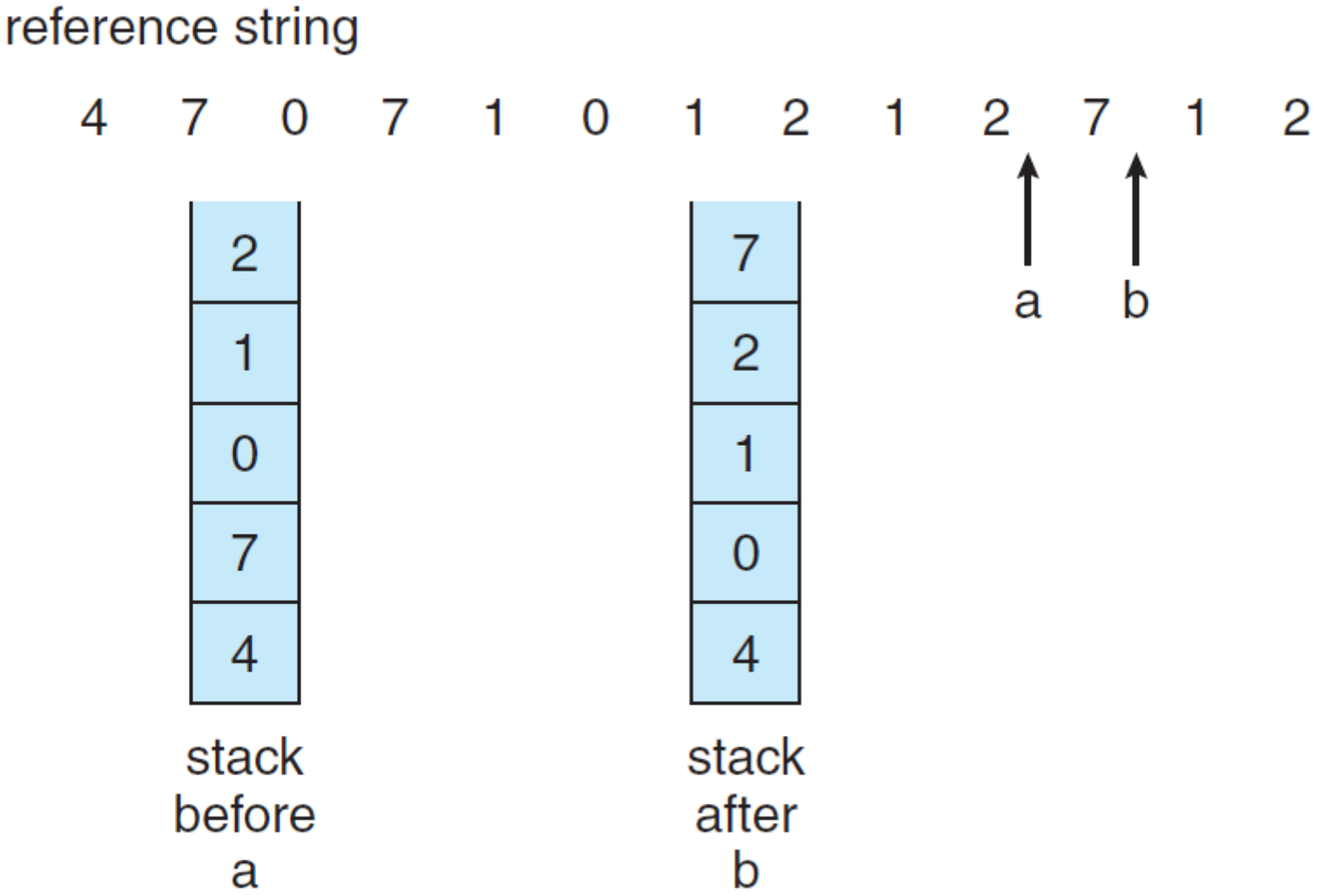


Figure 9.16 Use of a stack to record the most recent page references.

OSC9E

Simulating LRU in Software (NFU)

One possibility is called the NFU (Not Frequently Used) algorithm.

It requires a software counter associated with each page, initially zero.

At each clock interrupt, the operating system scans all the pages in memory. For each page, the R bit, which is 0 or 1, is added to the counter. The counters roughly keep track of how often each page has been referenced. When a page fault occurs, the page with the lowest counter is chosen for replacement.

MOS4E

Simulating LRU in Software (Aging)

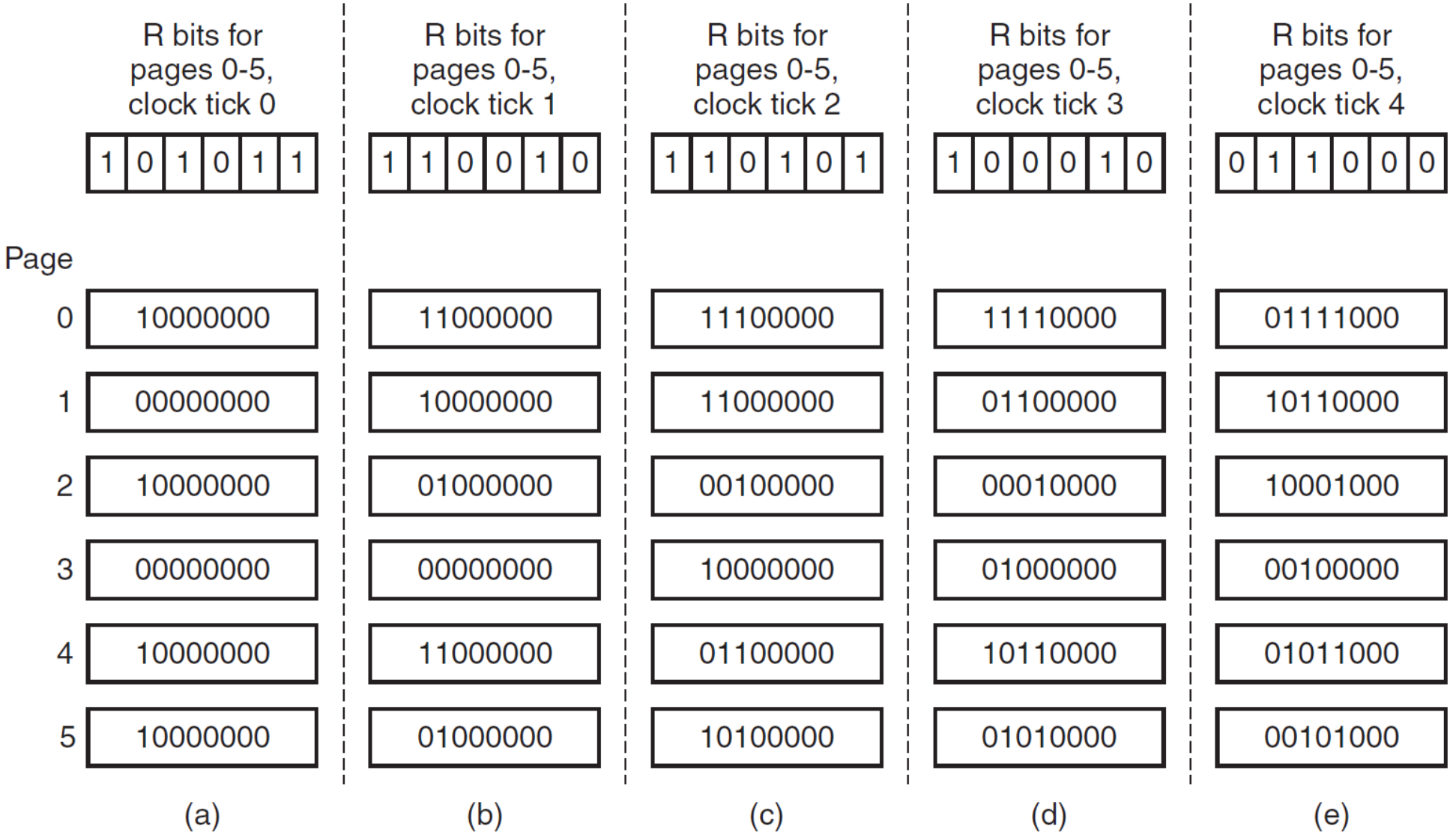


Figure 3-17. The aging algorithm simulates LRU in software. Shown are six pages for five clock ticks. The five clock ticks are represented by (a) to (e).

MOS4E

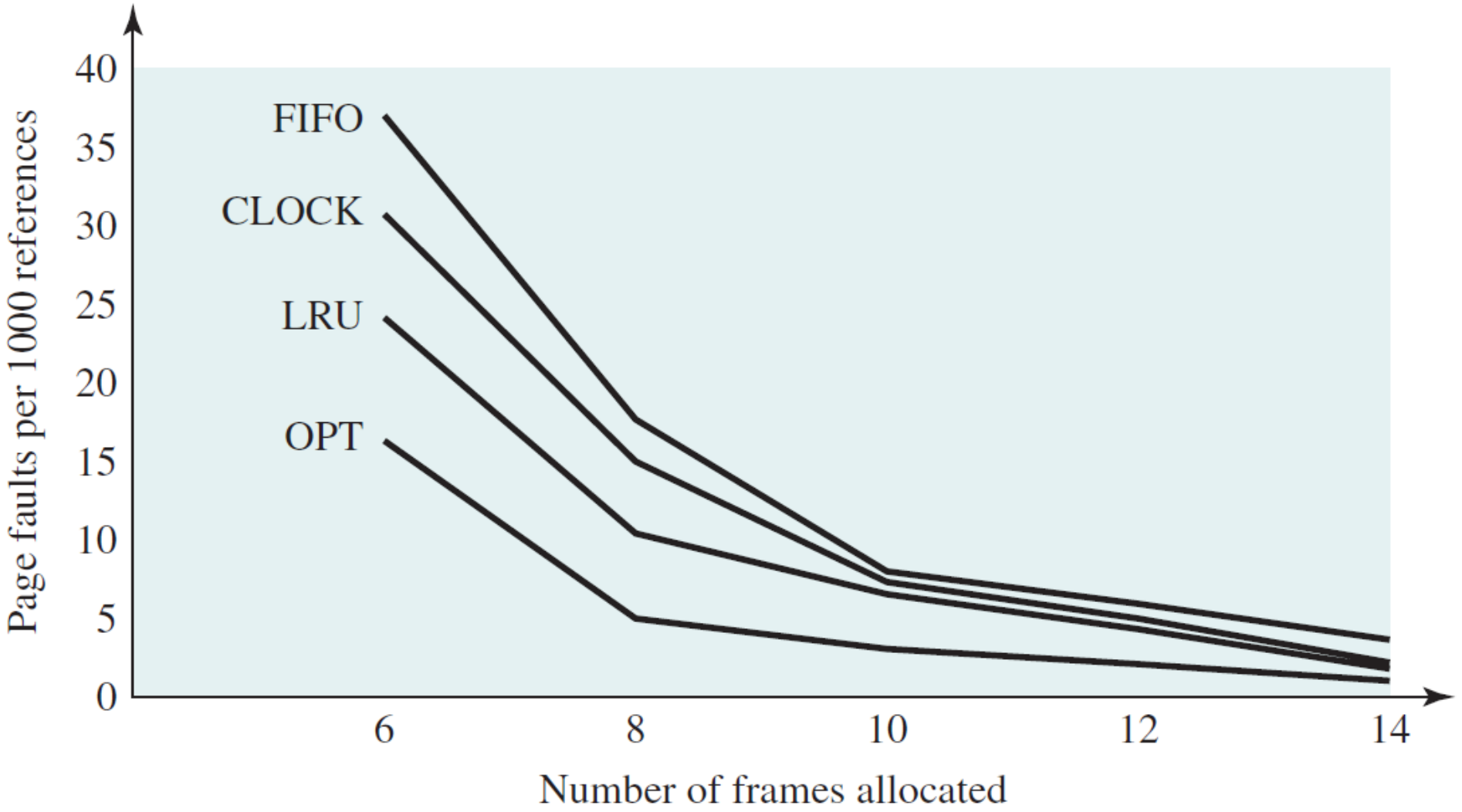


Figure 8.16 **Comparison of Fixed-Allocation, Local Page Replacement Algorithms**

OSIDP9E

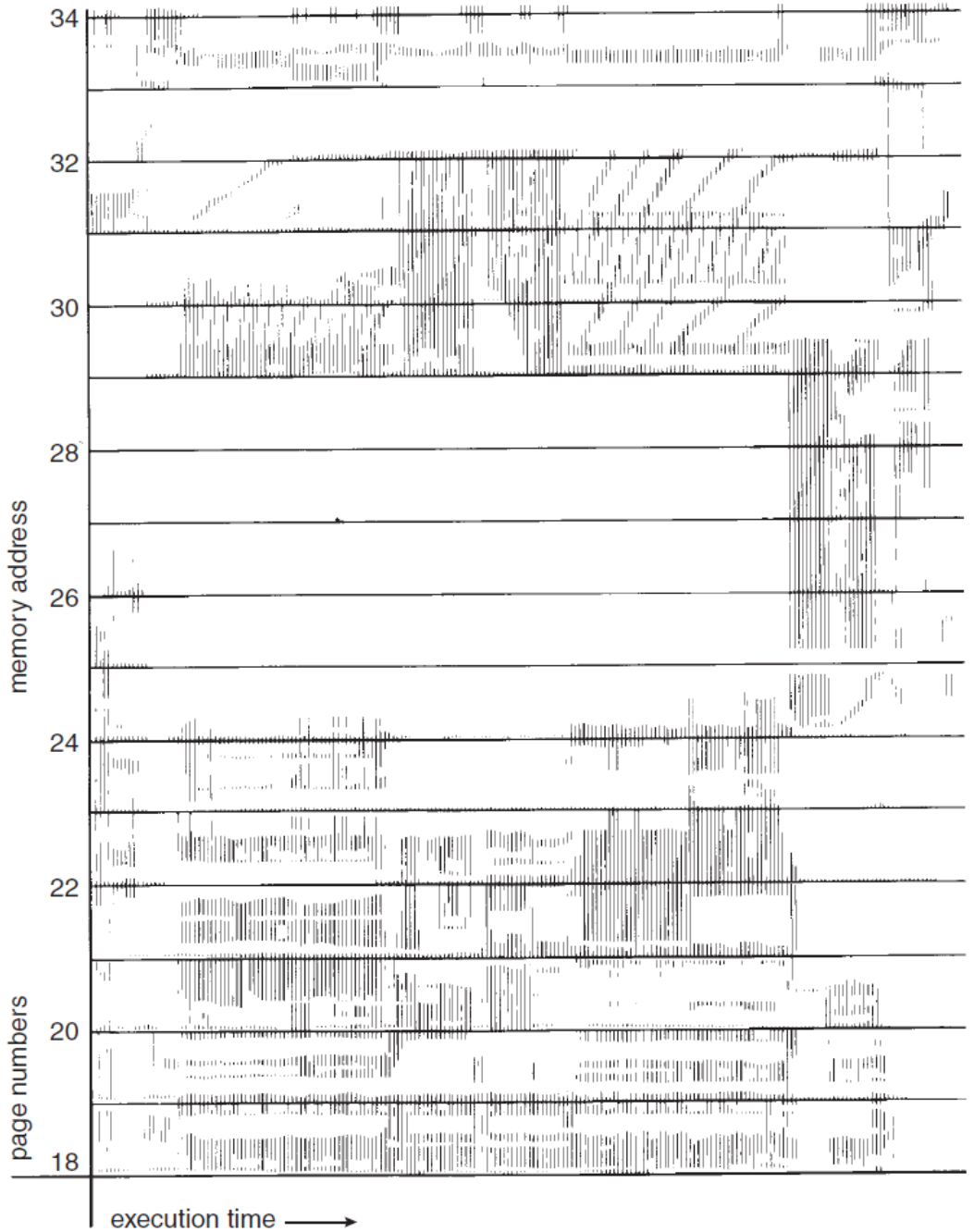


Figure 9.19 Locality in a memory-reference pattern.

OSC9E

The Working Set Model

At any instant of time, *t*, there exists a set consisting of all the pages used by the *k* most recent memory references. This set, *w*(*k*, *t*), is the working set.

Because the *k* = 1 most recent references must have used all the pages used by the *k* > 1 most recent references, and possibly others, *w*(*k*, *t*) is a monotonically nondecreasing function of *k*.

The limit of *w*(*k*, *t*) as *k* becomes large is finite because a program cannot reference more pages than its address space contains, and few programs will use every single page.

MOS4E

The Working Set Model

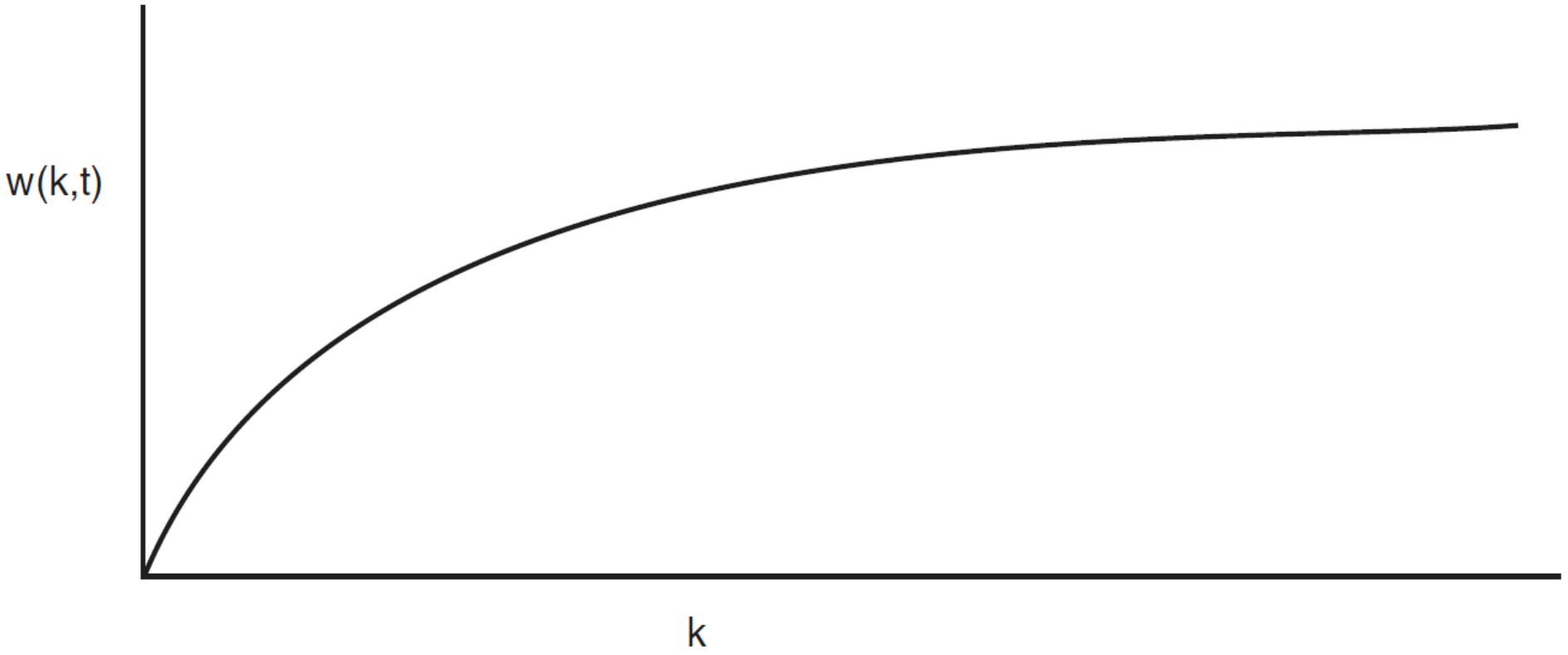


Figure 3-18. The working set is the set of pages used by the k most recent memory references. The function $w(k, t)$ is the size of the working set at time t.

MOS4E

The Working Set Model

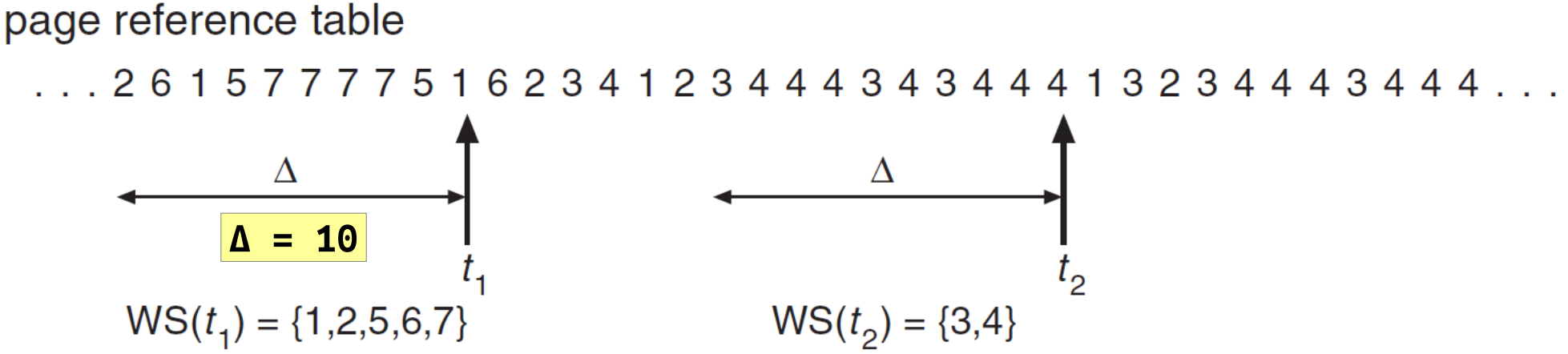


Figure 9.20 Working-set model.

OSC9E

Sequence of Page References	Window Size, Δ			
W	2	3	4	5
24	24	24	24	24
15	24 15	24 15	24 15	24 15
18	15 18	24 15 18	24 15 18	24 15 18
23	18 23	15 18 23	24 15 18 23	24 15 18 23
24	23 24	18 23 24	•	•
17	24 17	23 24 17	18 23 24 17	15 18 23 24 17
18	17 18	24 17 18	•	18 23 24 17
24	18 24	•	24 17 18	•
18	•	18 24	•	24 17 18
17	18 17	24 18 17	•	•
17	17	18 17	•	•
15	17 15	17 15	18 17 15	24 18 17 15
24	15 24	17 15 24	17 15 24	•
17	24 17	•	•	17 15 24
24	•	24 17	•	•
18	24 18	17 24 18	17 24 18	15 17 24 18

Figure 8.17 **Working Set of Process as Defined by Window Size**

OSIDP9E

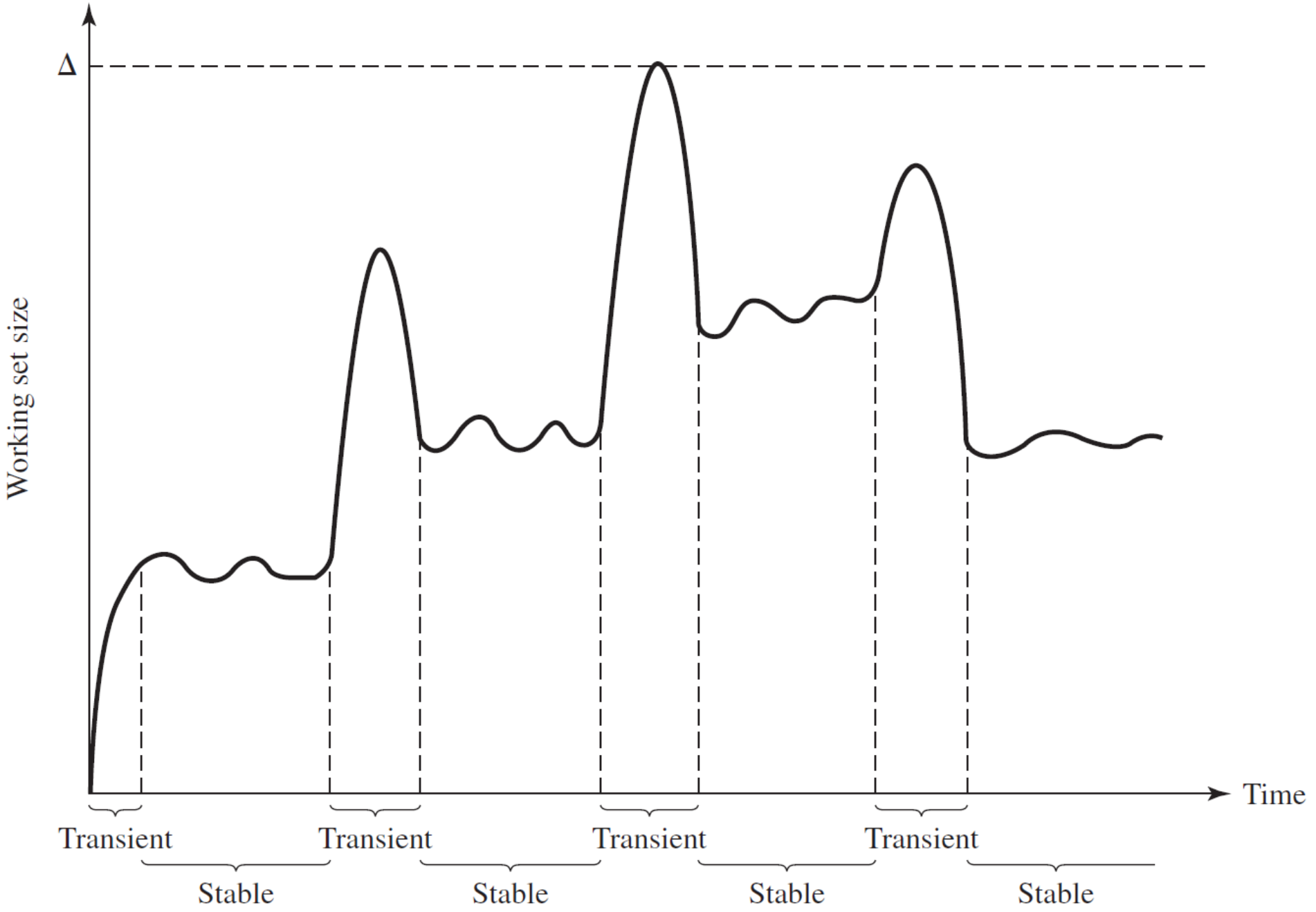
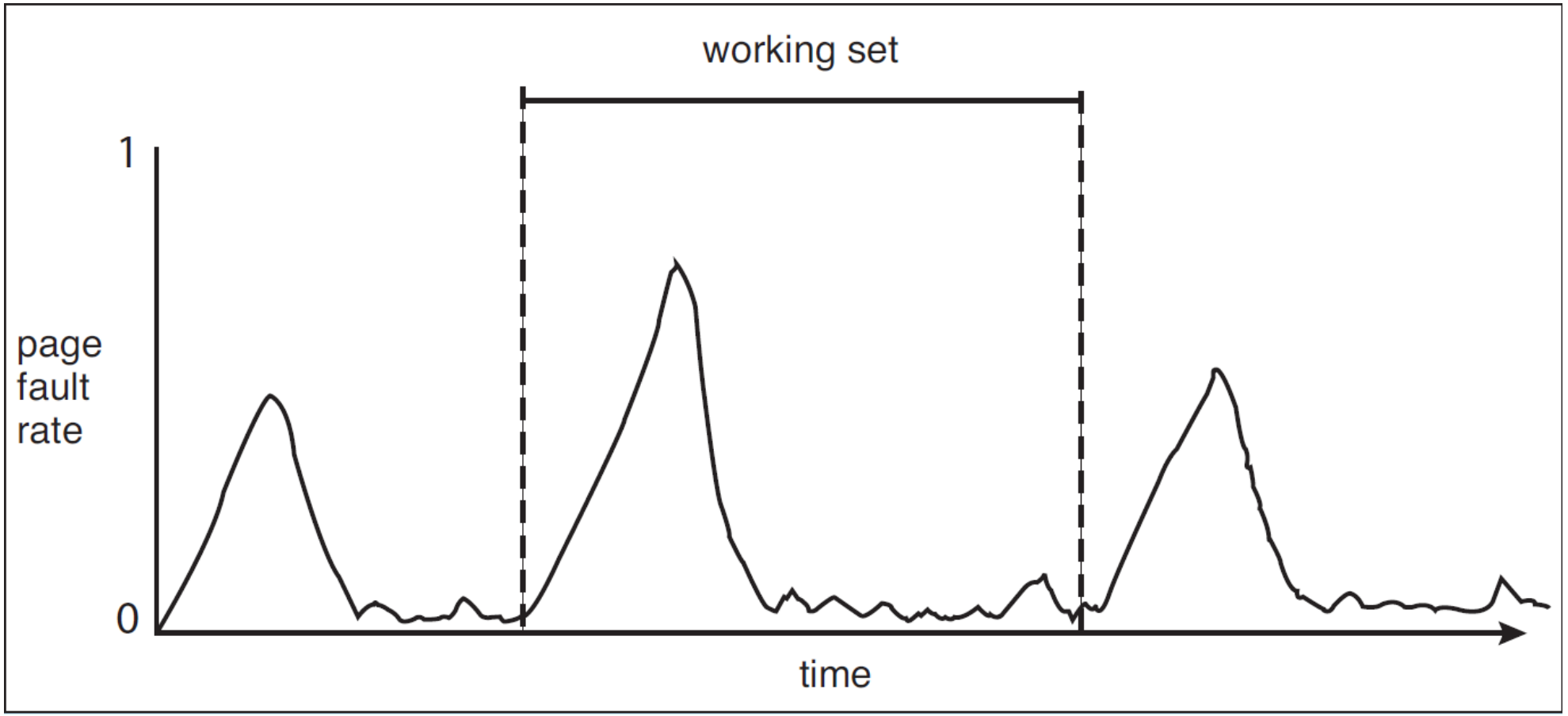


Figure 8.18 **Typical Graph of Working Set Size [MAEK87]**

OSIDP9E

The Working Set Model



OSC9E

The Working Set Algorithm

Instead of defining the working set as those pages used during the previous 10 million memory references, we can define it as the set of pages used during the past 100 msec of execution time. In practice, such a definition is just as good and much easier to work with. Note that for each process, only its own execution time counts. Thus if a process starts running at time *T* and has had 40 msec of CPU time at real time *T* + 100 msec, for working set purposes its time is 40 msec.

The amount of CPU time a process has actually used since it started is often called its current virtual time. With this approximation, the working set of a process is the set of pages it has referenced during the past τ seconds of virtual time.

MOS4E

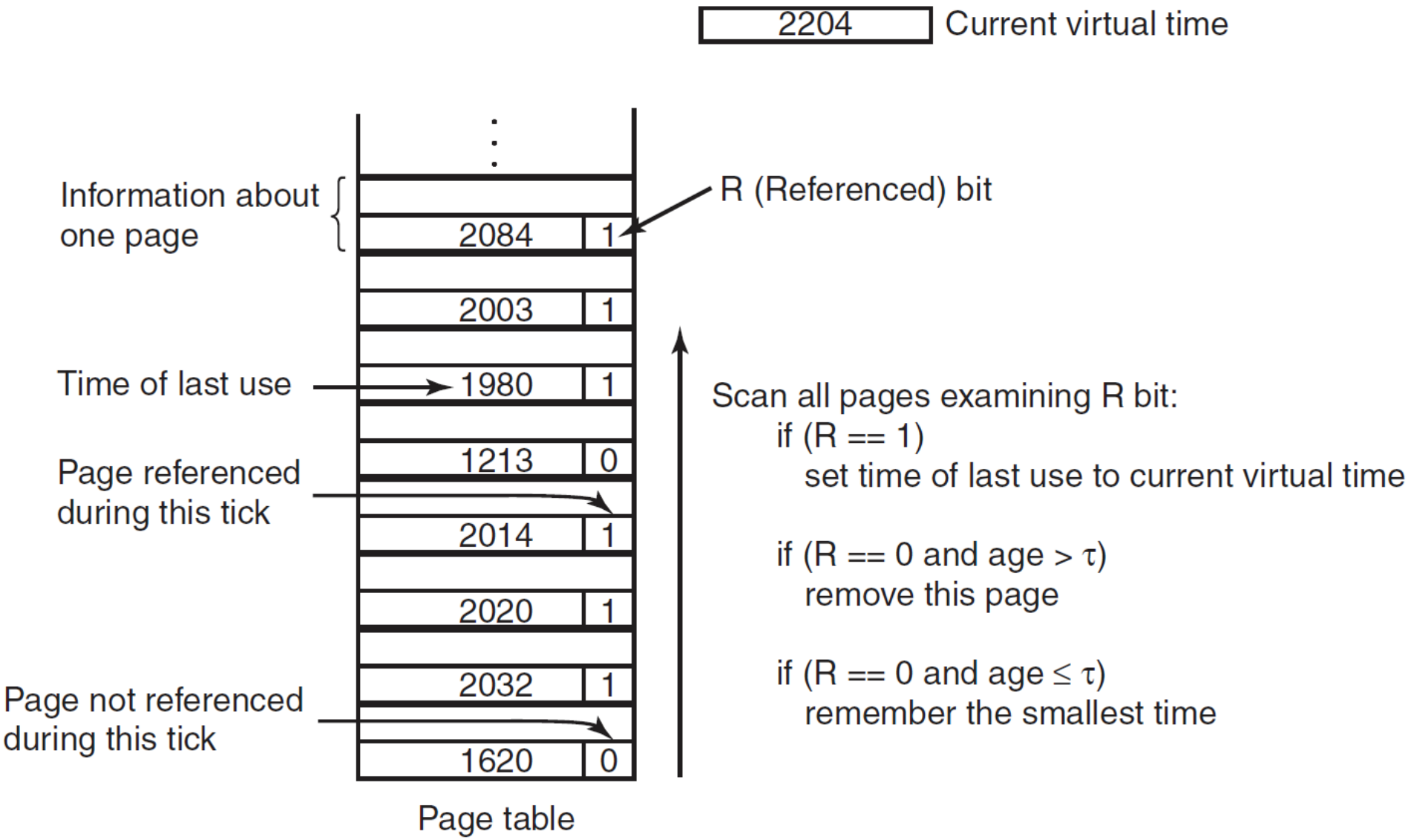


Figure 3-19. The working set algorithm.

MOS4E

The Working Set Algorithm

The hardware is assumed to set the *R* and *M* bits. Similarly, a periodic clock interrupt is assumed to cause software to run that clears the *Referenced* bit on every clock tick. On every page fault, the page table is scanned to look for a suitable page to evict.

As each entry is processed, the *R* bit is examined. If it is 1, the current virtual time is written into the *Time of last use* field in the page table, indicating that the page was in use at the time the fault occurred. Since the page has been referenced during the current clock tick, it is clearly in the working set and is not a candidate for removal (τ is assumed to span multiple clock ticks).

MOS4E

The Working Set Algorithm

If *R* is 0, the page has not been referenced during the current clock tick and may be a candidate for removal.

To see whether or not it should be removed, its age (the current virtual time minus its *Time of last use*) is computed and compared to τ.

If the age is greater than τ, the page is no longer in the working set and the new page replaces it. The scan continues updating the remaining entries.

MOS4E

The Working Set Algorithm

However, if *R* is 0 but the age is less than or equal to τ, the page is still in the working set.

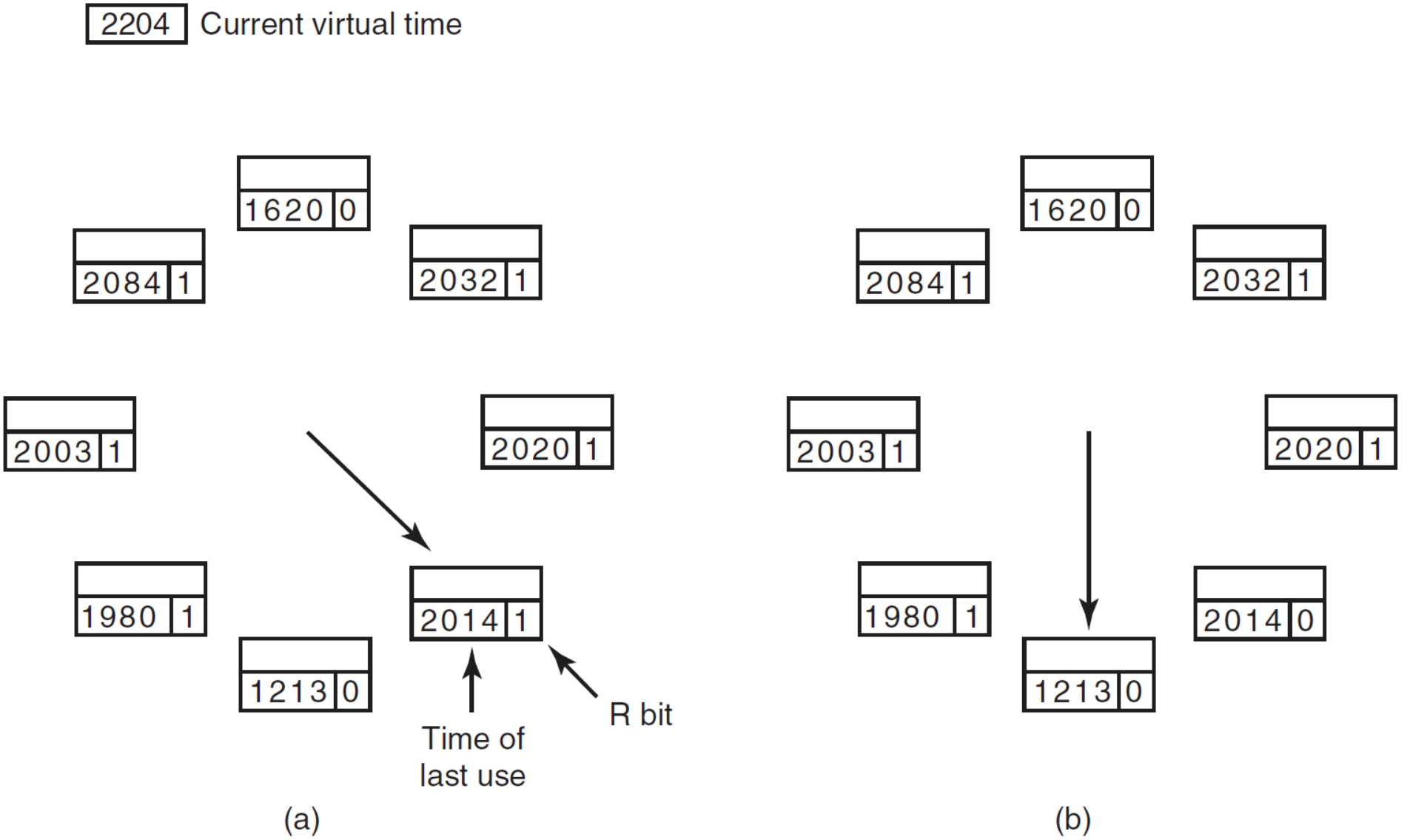
The page is temporarily spared, but the page with the greatest age (smallest value of *Time of last use*) is noted.

If the entire table is scanned without finding a candidate to evict, that means that all pages are in the working set. In that case, if one or more pages with *R* = 0 were found, the one with the greatest age is evicted.

In the worst case, all pages have been referenced during the current clock tick (and thus all have *R* = 1), so one is chosen at random for removal, preferably a clean page, if one exists.

MOS4E

The WSClock Algorithm



MOS4E

The WSClock Algorithm

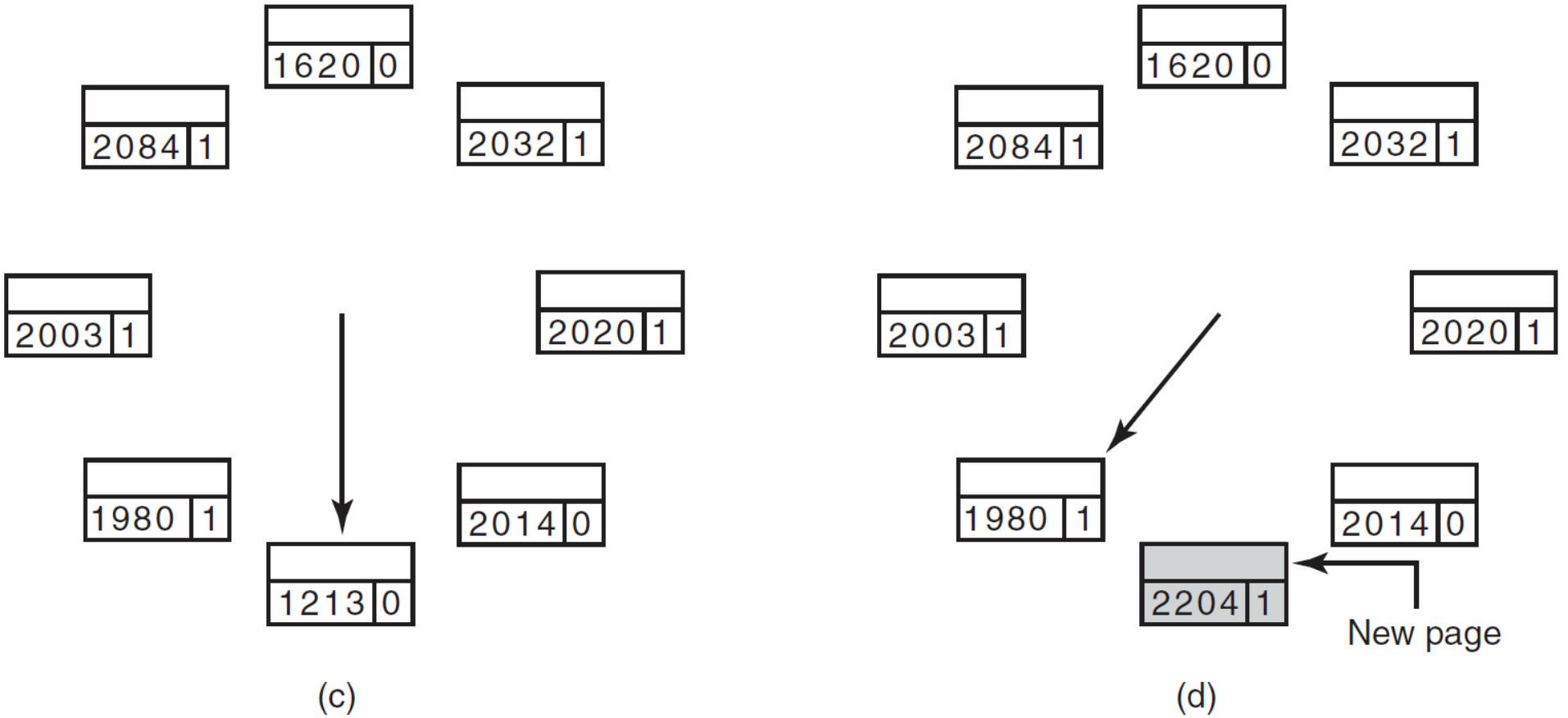


Figure 3-20. Operation of the WSClock algorithm. (a) and (b) give an example of what happens when $R = 1$. (c) and (d) give an example of $R = 0$.

MOS4E

Algorithm	Comment
Optimal	Not implementable, but useful as a benchmark
NRU (Not Recently Used)	Very crude approximation of LRU
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

Figure 3-21. Page replacement algorithms discussed in the text.

MOS4E