Virtual Memory

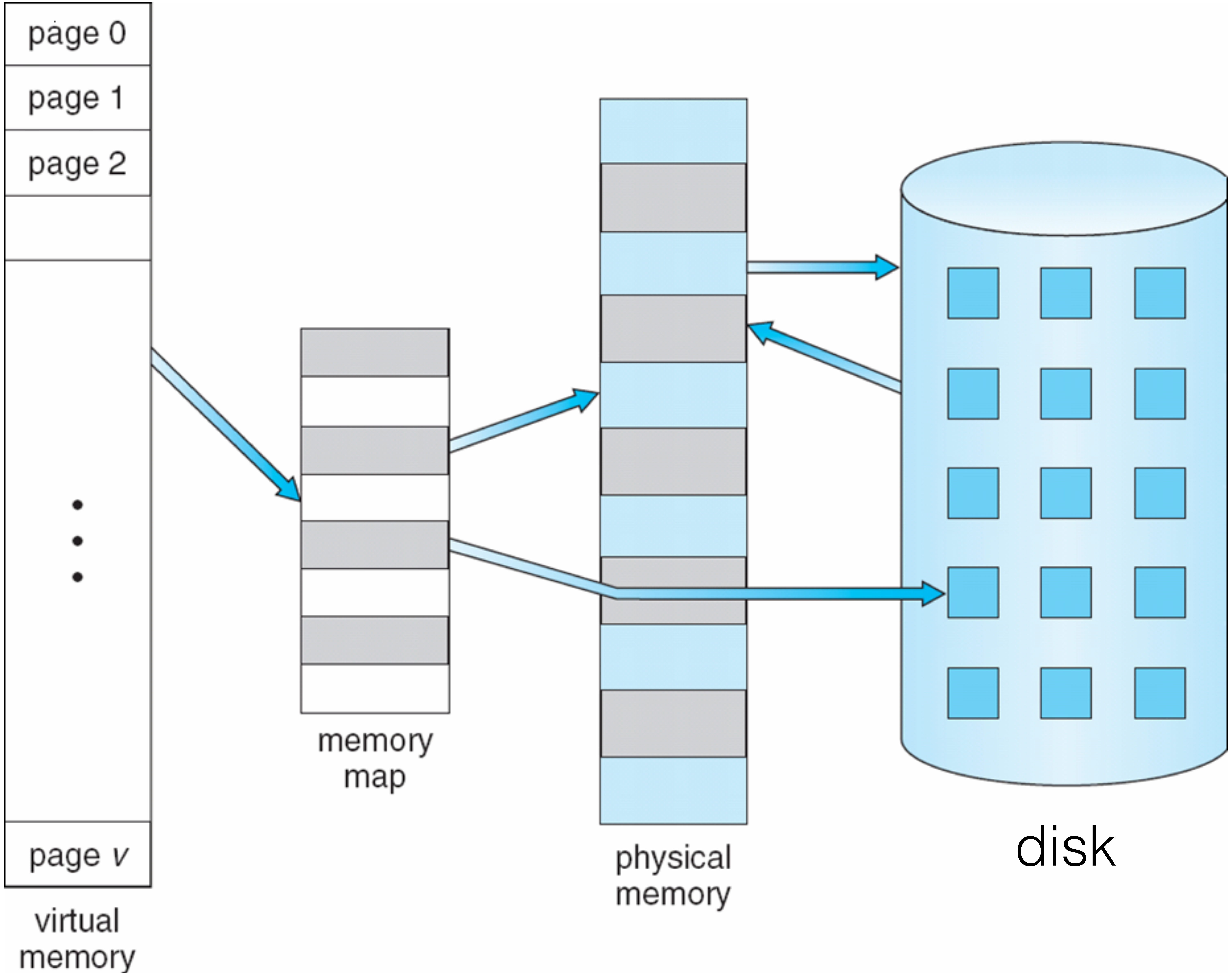
CSE 4001

Content

- Demand paging

Virtual Memory

- Separation of user logical memory from physical memory.
- Programs can be partially in memory for execution
- Logical address space can be much larger than physical address space



Implementation

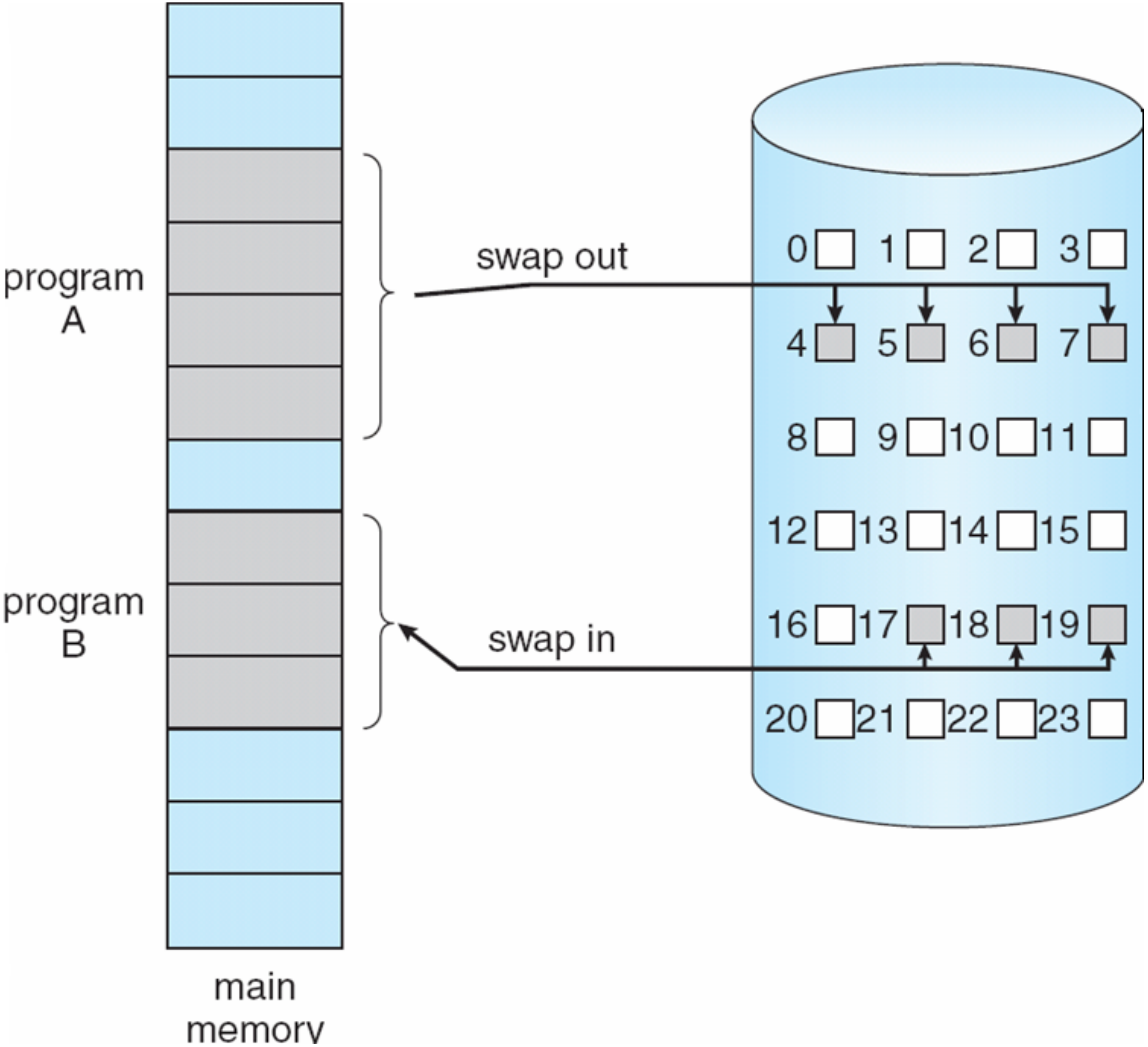
Virtual memory can be implemented via:

- Demand paging
- Demand segmentation

Demand paging

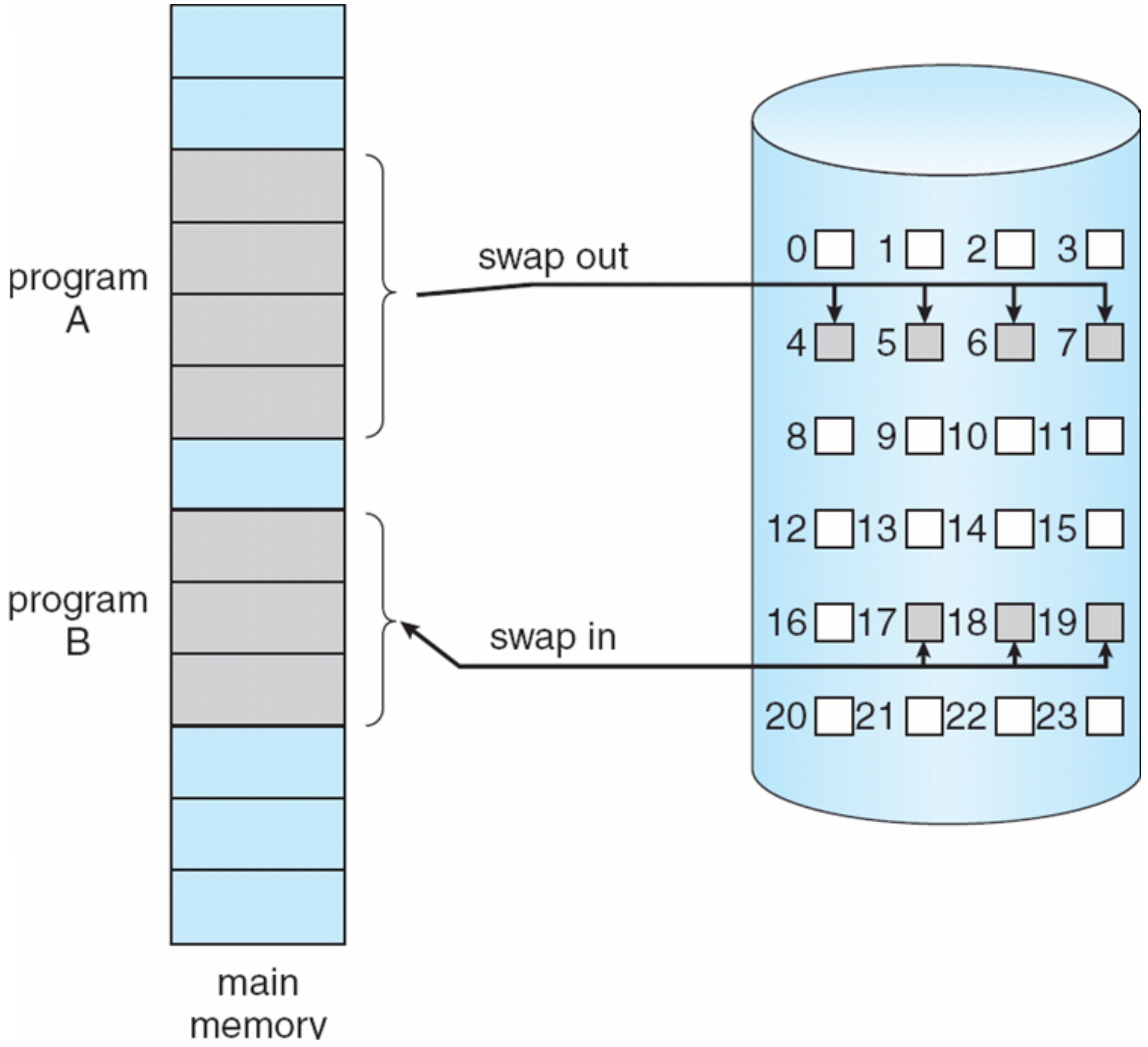
Bring a page into memory only when it is needed:

- Less I/O needed
- Less memory needs
- Faster response
- More users



Demand paging

- Demand paging is similar to a paging system with swapping, where processes reside in secondary memory (e.g., disk).
- Lazy swapper: only bring pages when they are needed.
- In the context of demand paging, we use the term pager instead of swapper.

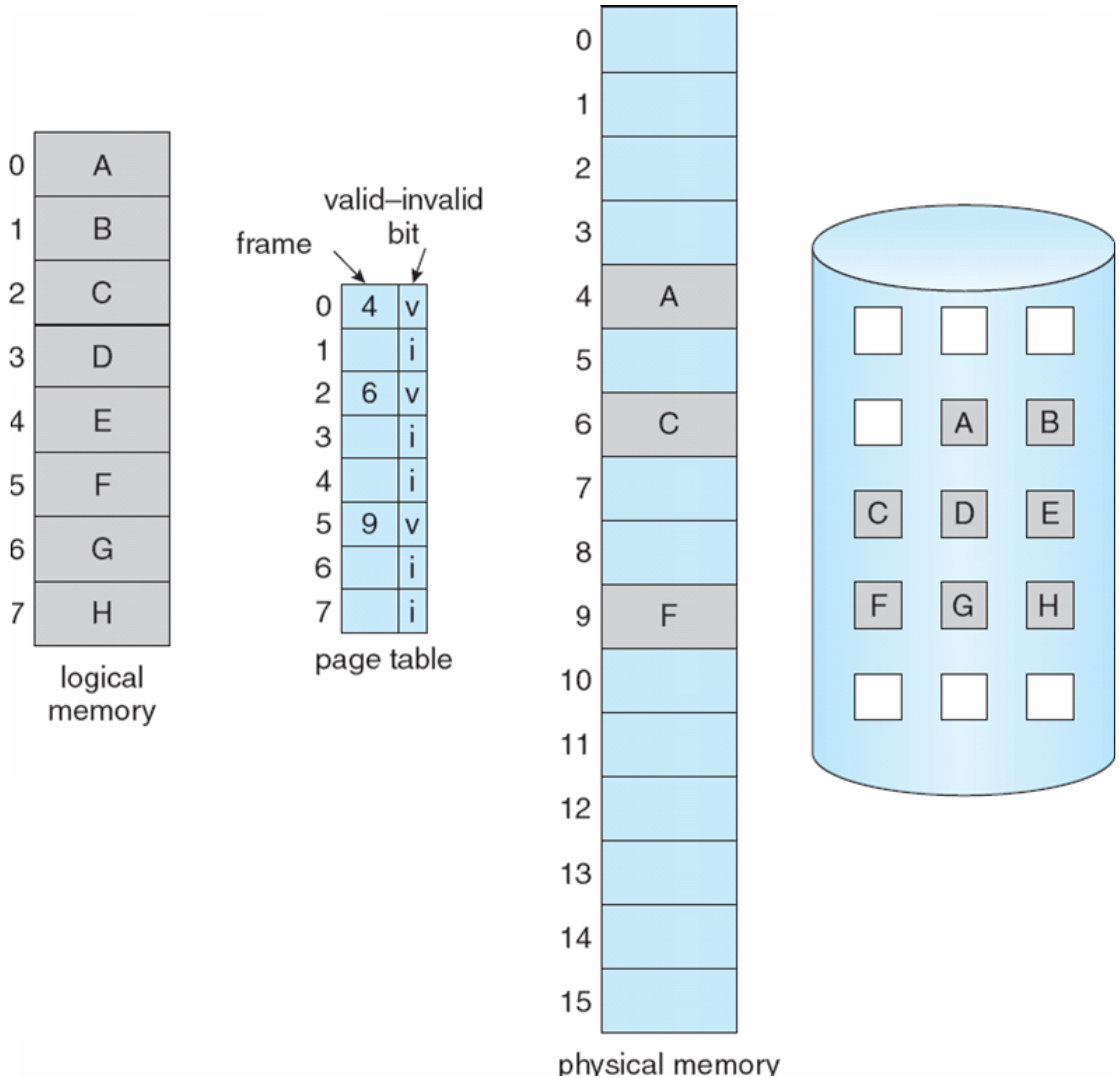


Valid-Invalid Bit

- Hardware support is needed to distinguish between the pages that are in memory and the ones that are on the disk.
- We can re-use the support provided by the valid-invalid bit in the page table.
    - Bit == valid then page is in memory (and is valid).
    - Bit == invalid then page is either not a valid one for that process or is valid but is currently in disk (pager needs to bring it to main memory).

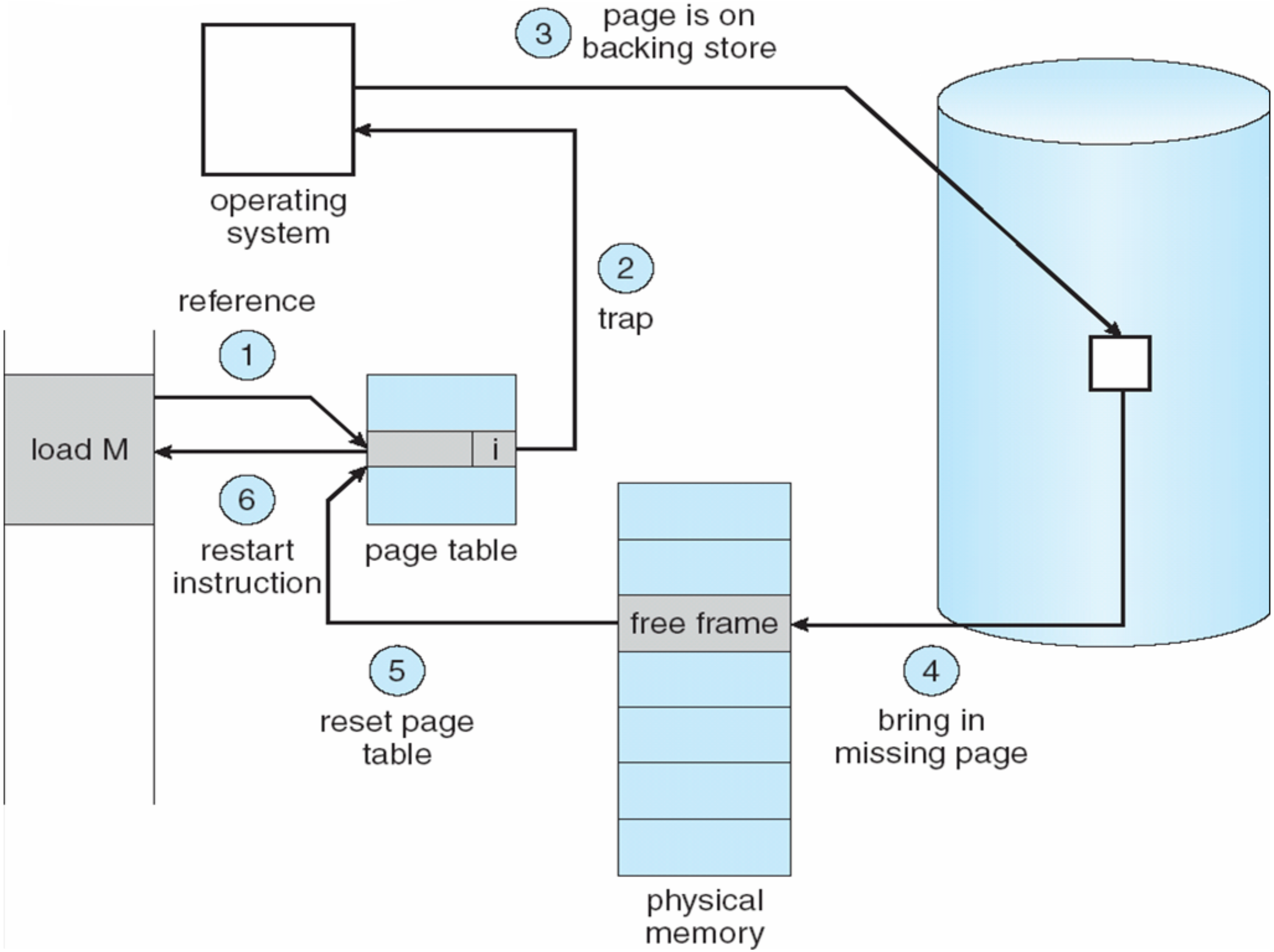
Valid-Invalid Bit

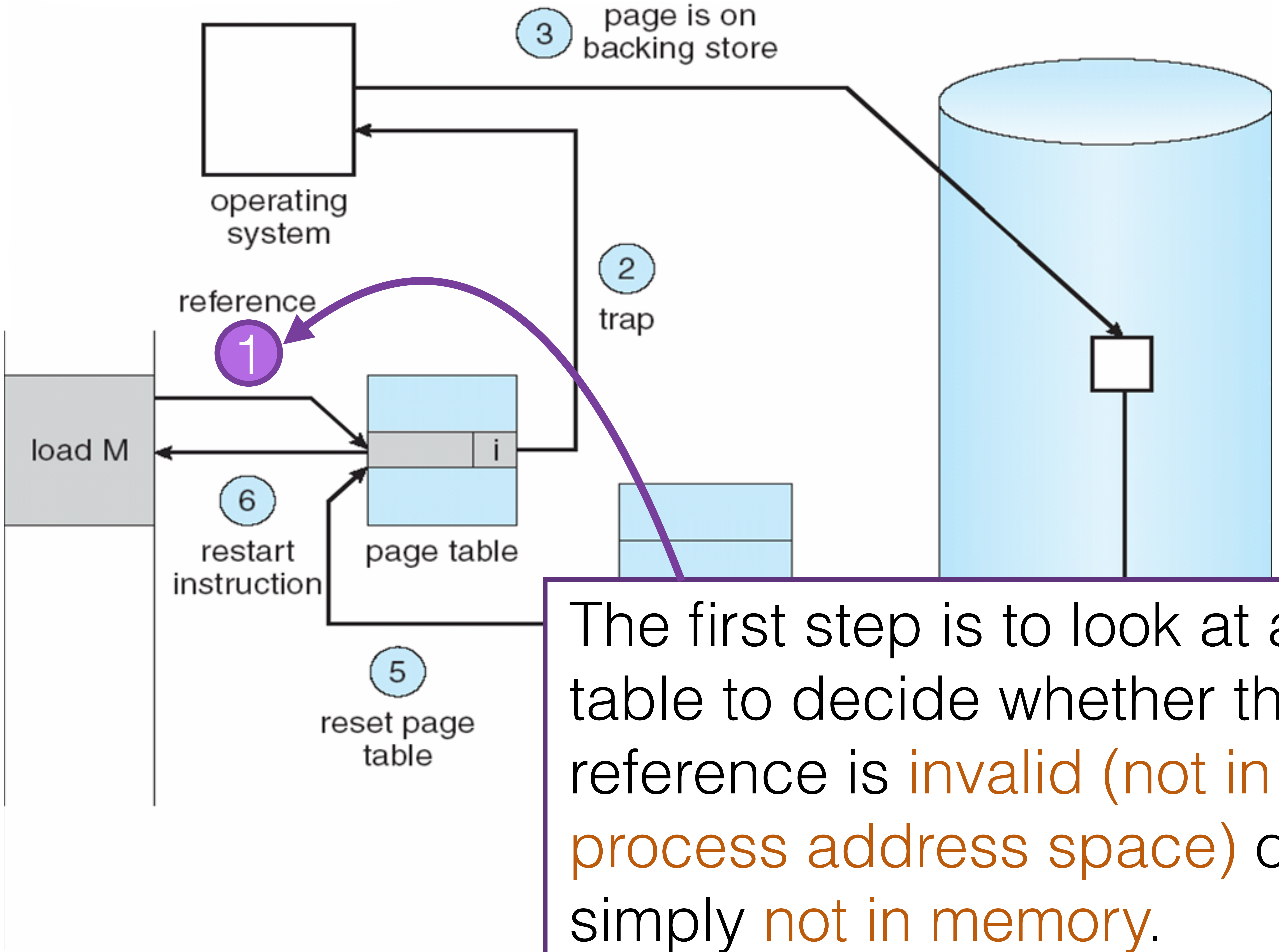
- Marking a page invalid has no effect if the process never attempts to access that page.
- Pages that are in memory are called memory resident.



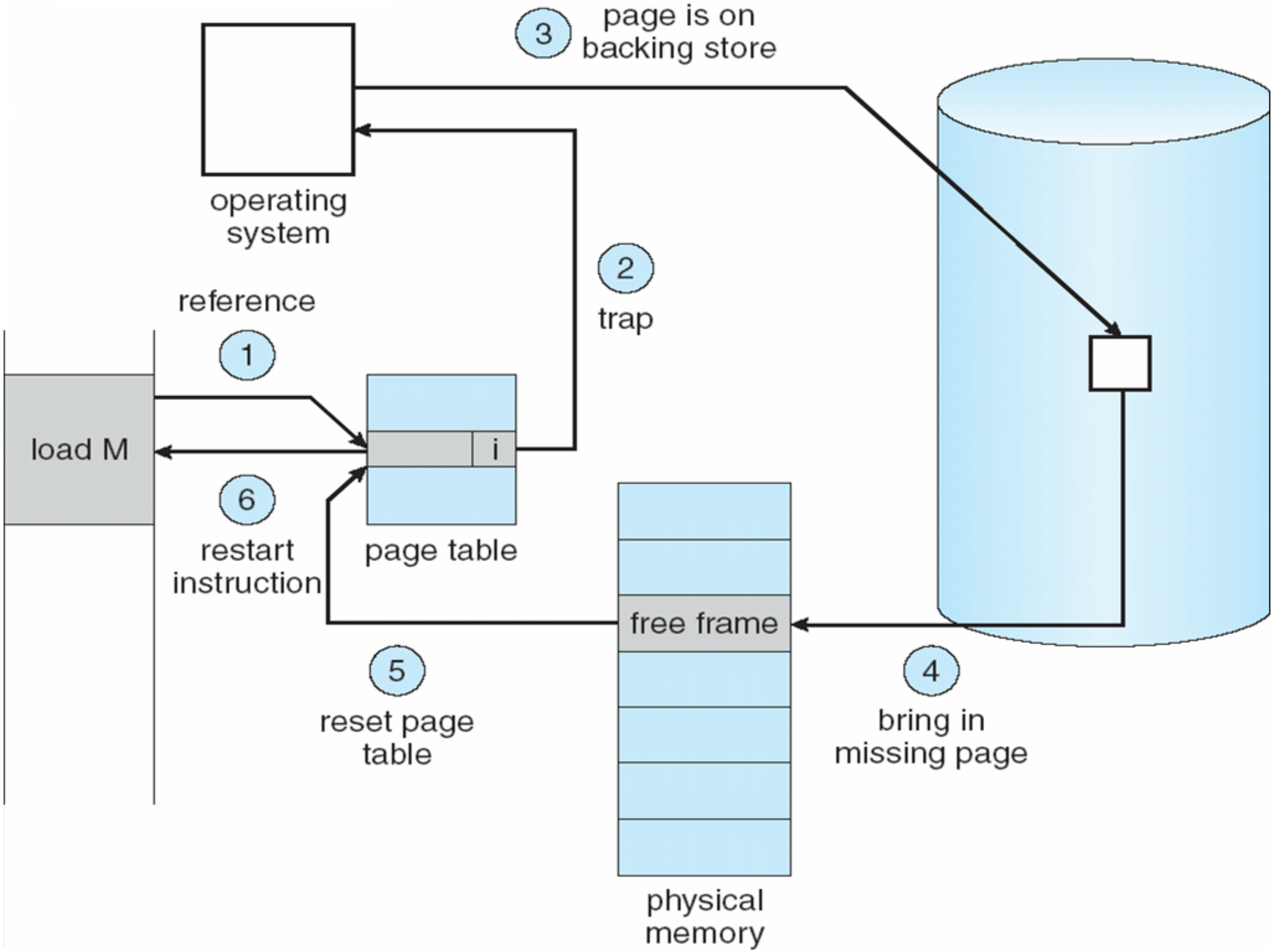
Page Faults

- What happens when a process tries to access non-resident pages?
  - Page Fault: A trap that results because the OS's failed to bring the desired page into memory.

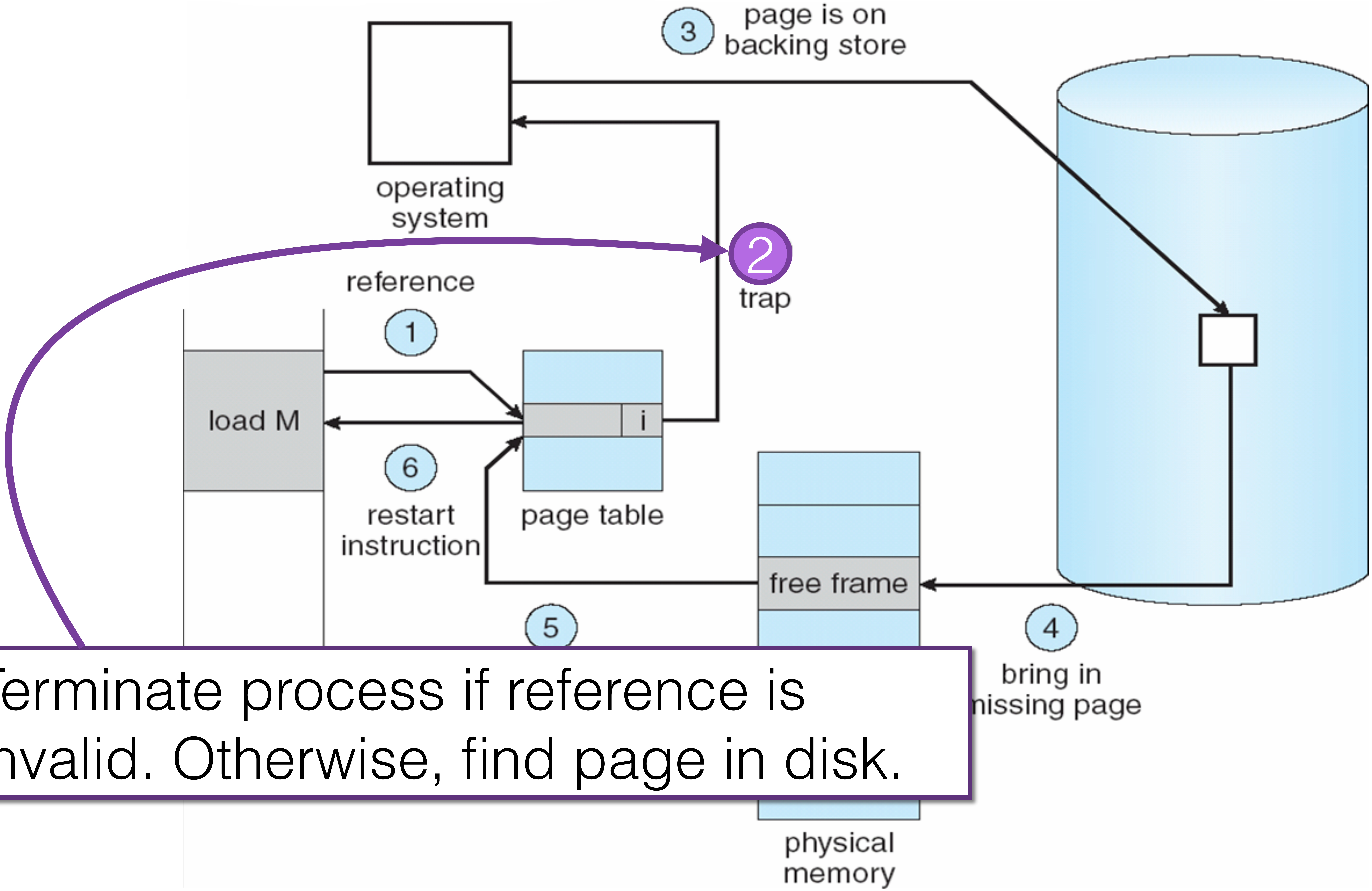

page is on
backing store
3
operating
system
2
trap
reference
1
load M
i
6
restart
instruction
page table
free frame
5
reset page
table
4
bring in
missing page
physical
memory



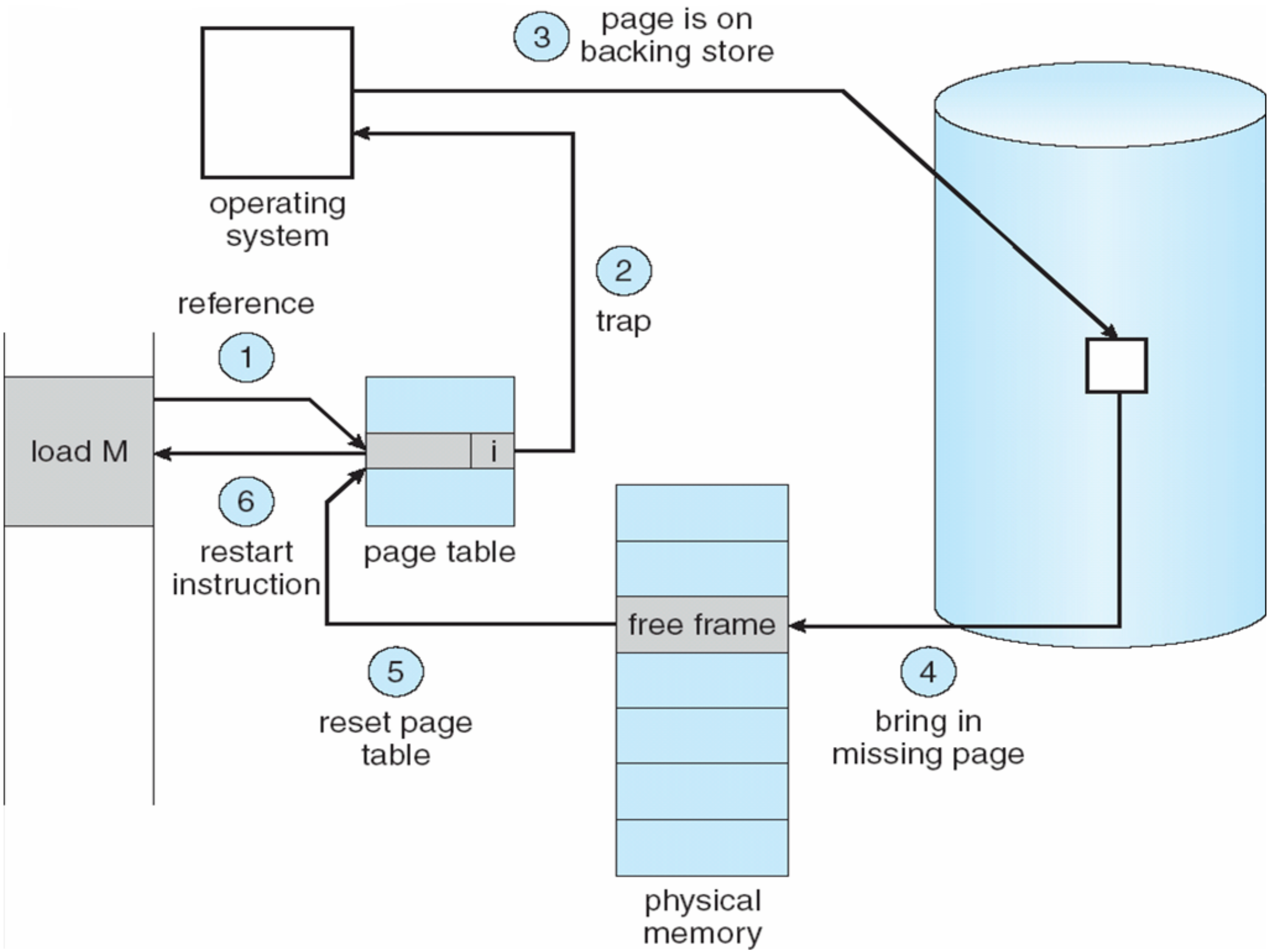
The first step is to look at another table to decide whether the actual reference is invalid (not in the process address space) or is simply not in memory.



3 page is on backing store
operating system
2 trap
reference
1
load M
i
6
restart instruction
page table
free frame
5
reset page table
4
bring in missing page
physical memory

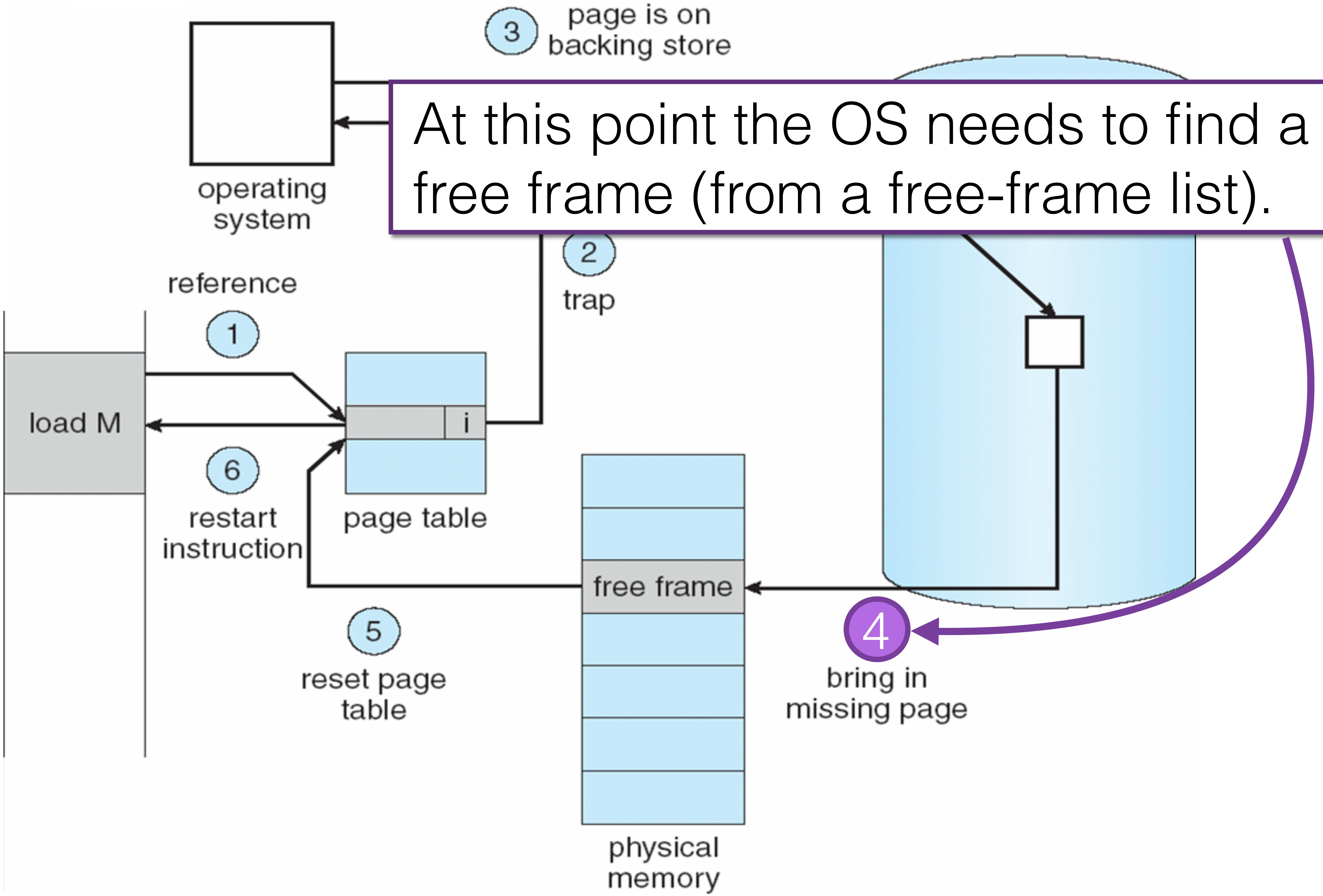


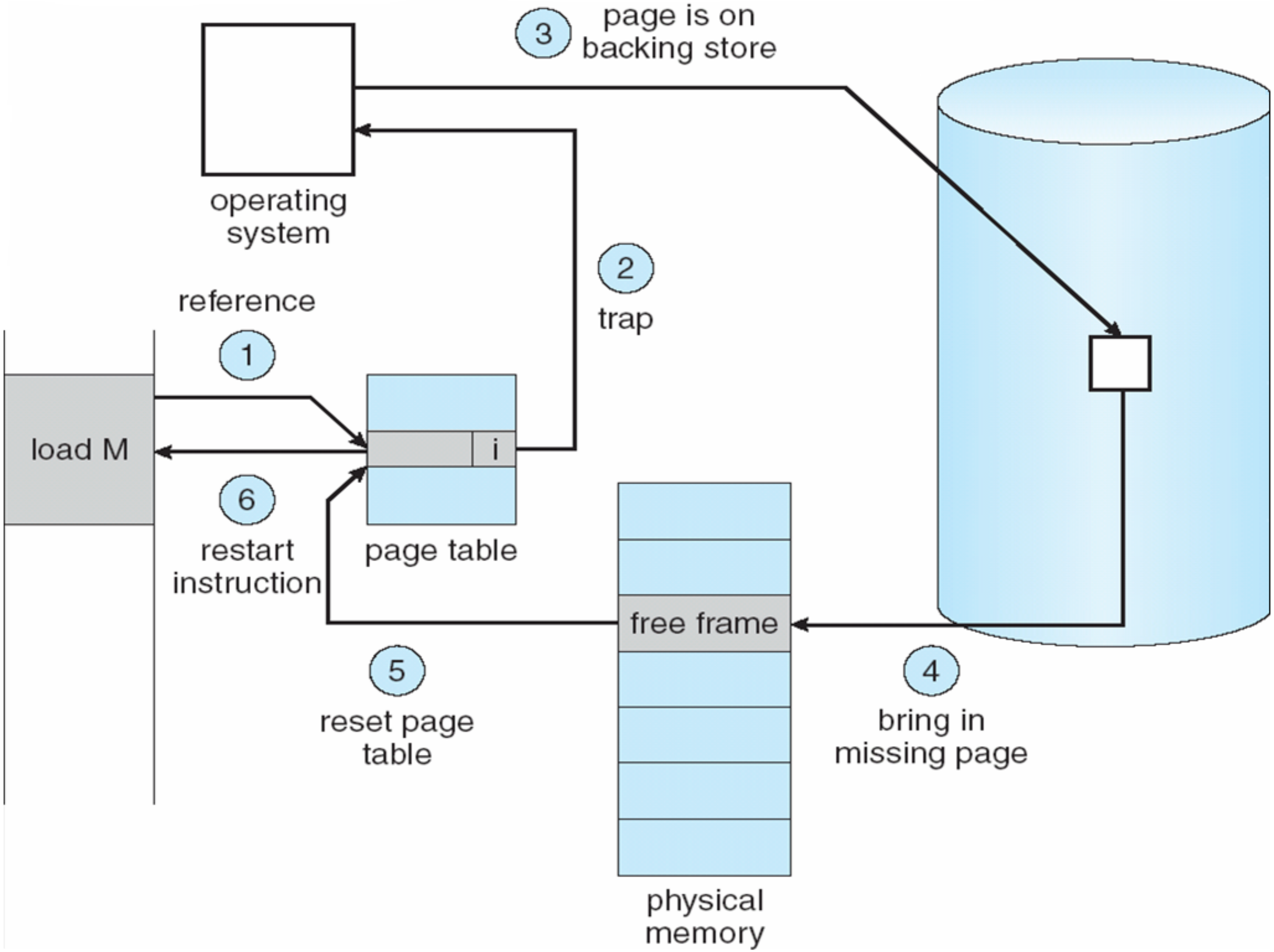
Terminate process if reference is invalid. Otherwise, find page in disk.



3
page is on
backing store
operating
system
2
trap
reference
1
load M
i
6
restart
instruction
page table
free frame
5
reset page
table
4
bring in
missing page
physical
memory

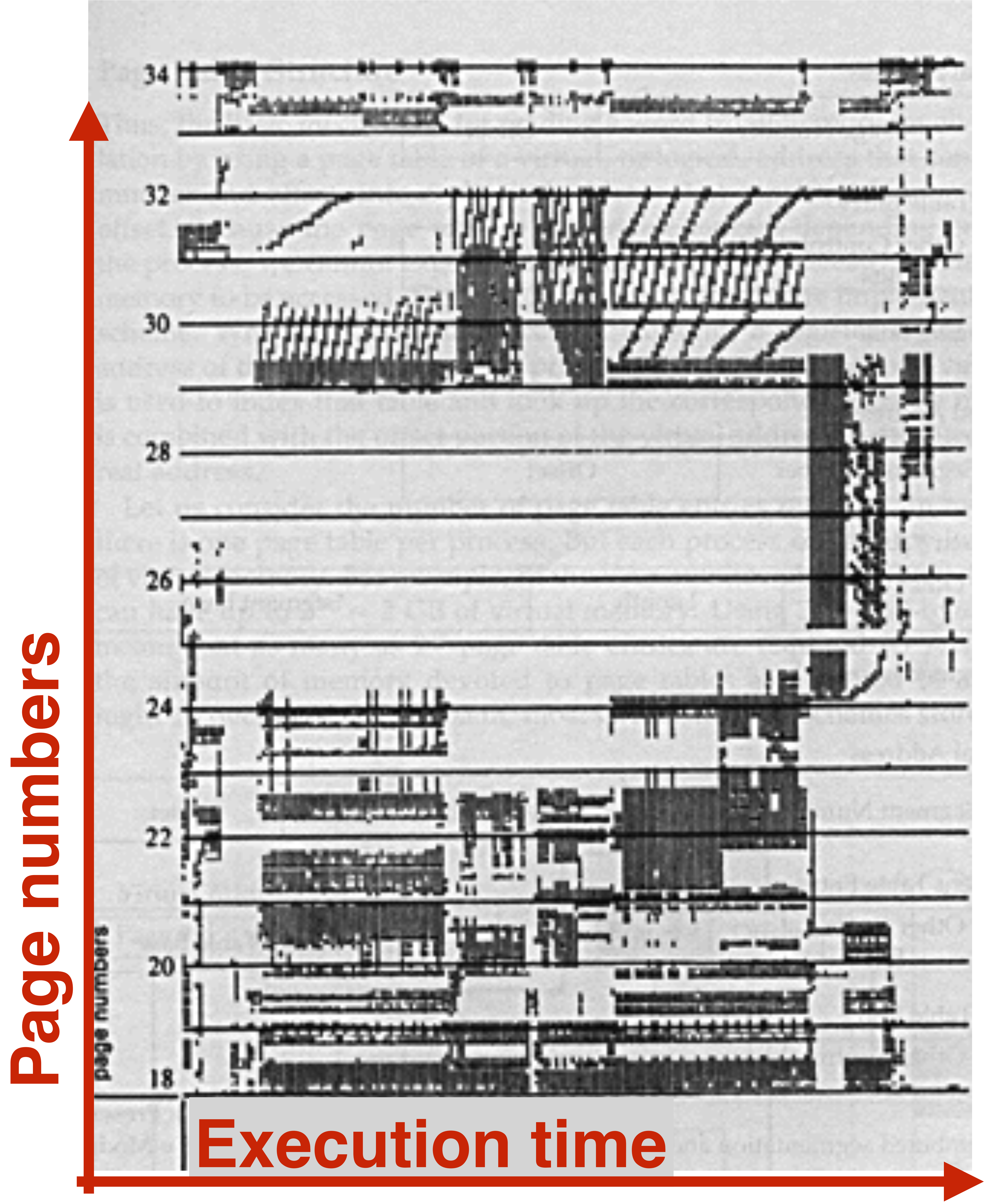
At this point the OS needs to find a free frame (from a free-frame list).




page is on
backing store
3
operating
system
2
trap
reference
1
load M
i
6
restart
instruction
page table
free frame
5
reset page
table
4
bring in
missing page
physical
memory

Locality in memory-reference pattern

- Theoretically, some programs could access several new pages with a single instruction.
- In this case, system performance could be seriously degraded.
- Luckily, this behavior is unlikely.



Writing code with demand-paging in mind…

- Program structure
  - `Int[128,128] data;`
  - Each row is stored in one page
  - Program 1

```
for (j = 0; j <128; j++)
    for (i = 0; i < 128; i++)
        data[i,j] = 0;
```

128 x 128 = 16,384 page faults

- Program 2

```
for (i = 0; i < 128; i++)
    for (j = 0; j < 128; j++)
        data[i,j] = 0;
```

128 page faults

Thrashing

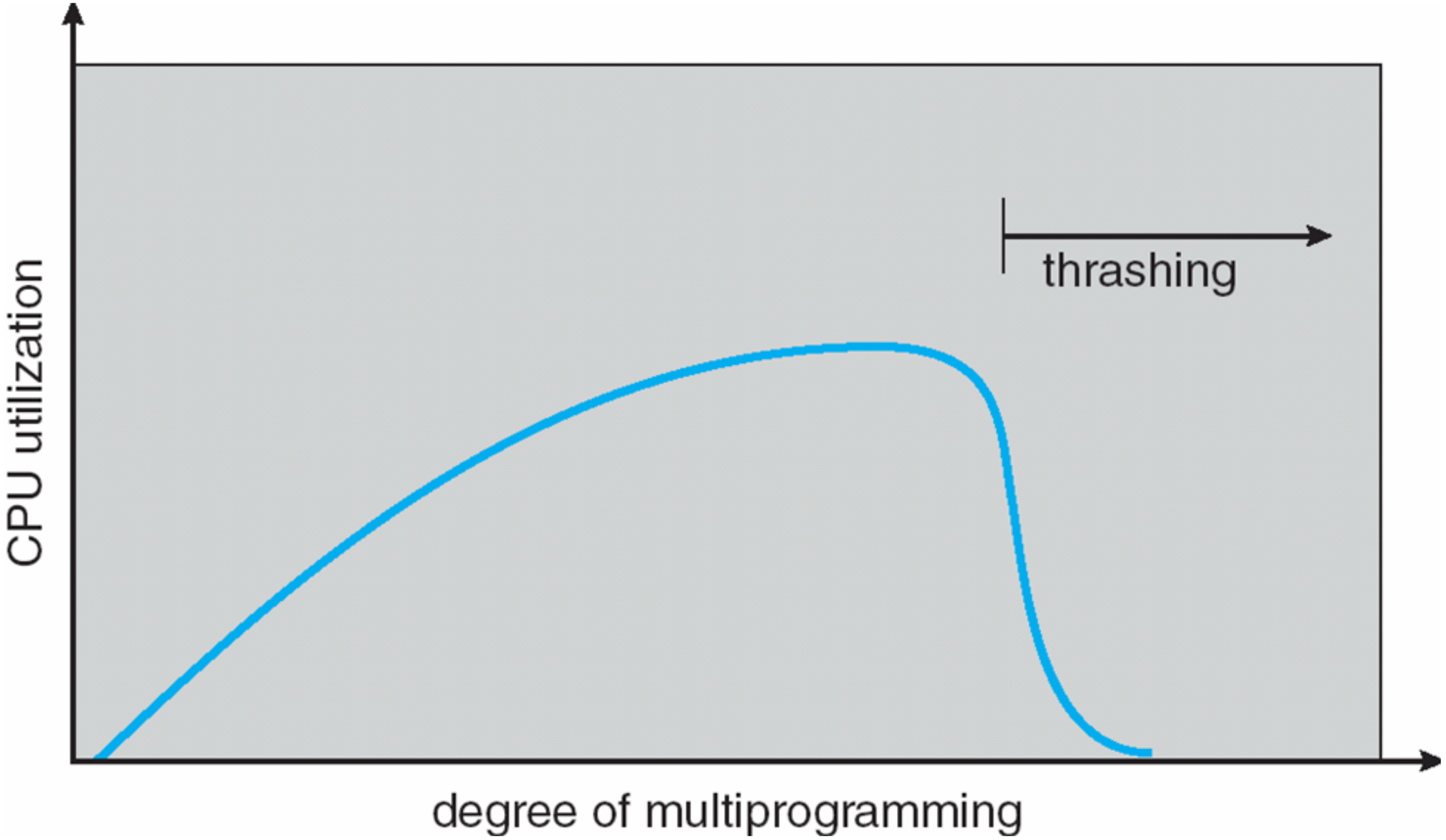
- The process does not have “enough” pages, the page-fault rate is very high and CPU becomes sub-utilized.

- The OS wants to maximize CPU utilization. As a result, it decides that it is a good idea to increase the degree of multiprogramming by adding new processes to the system.

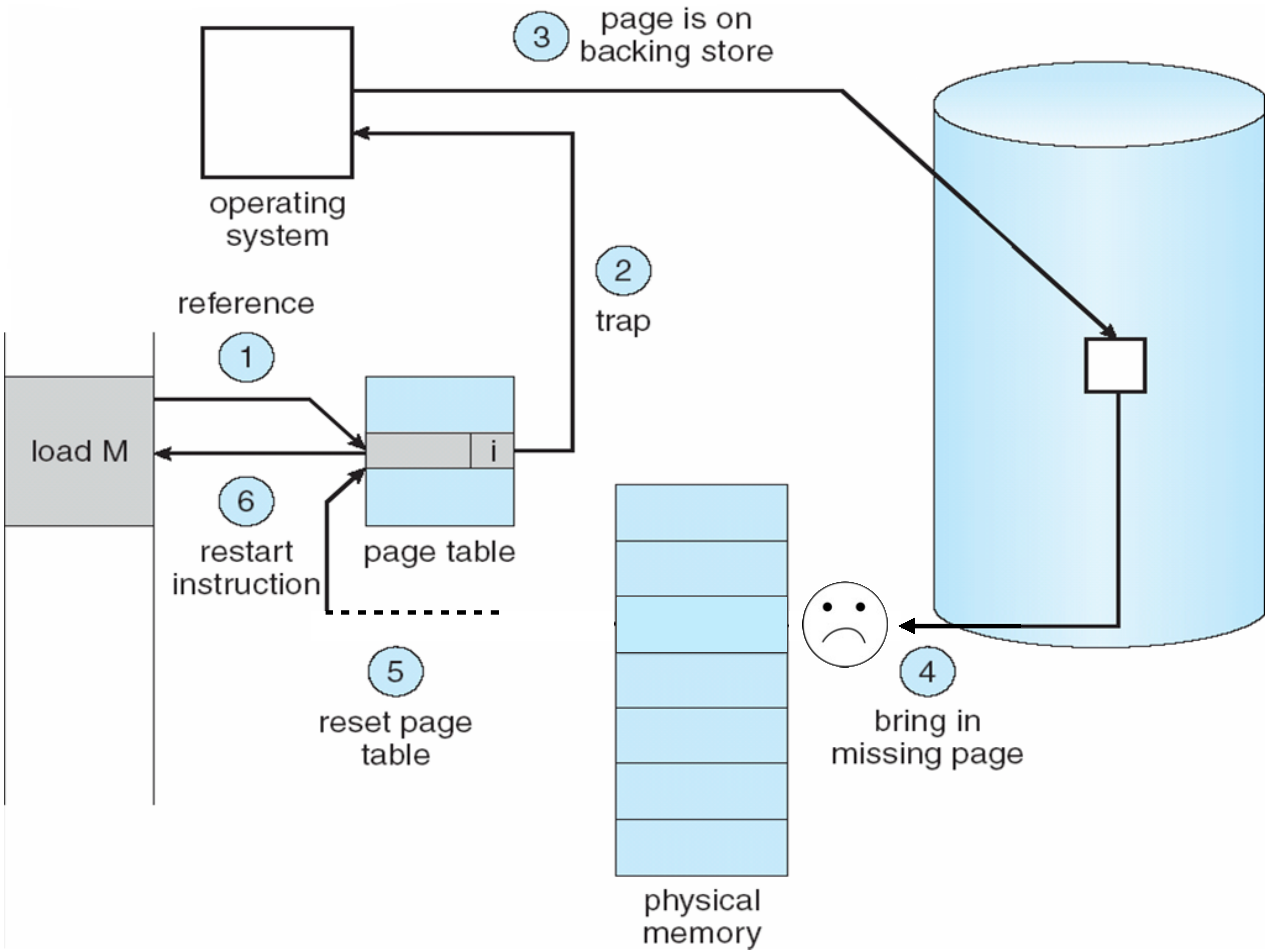
- Thrashing: A process is spending more time paging that executing.

Thrashing

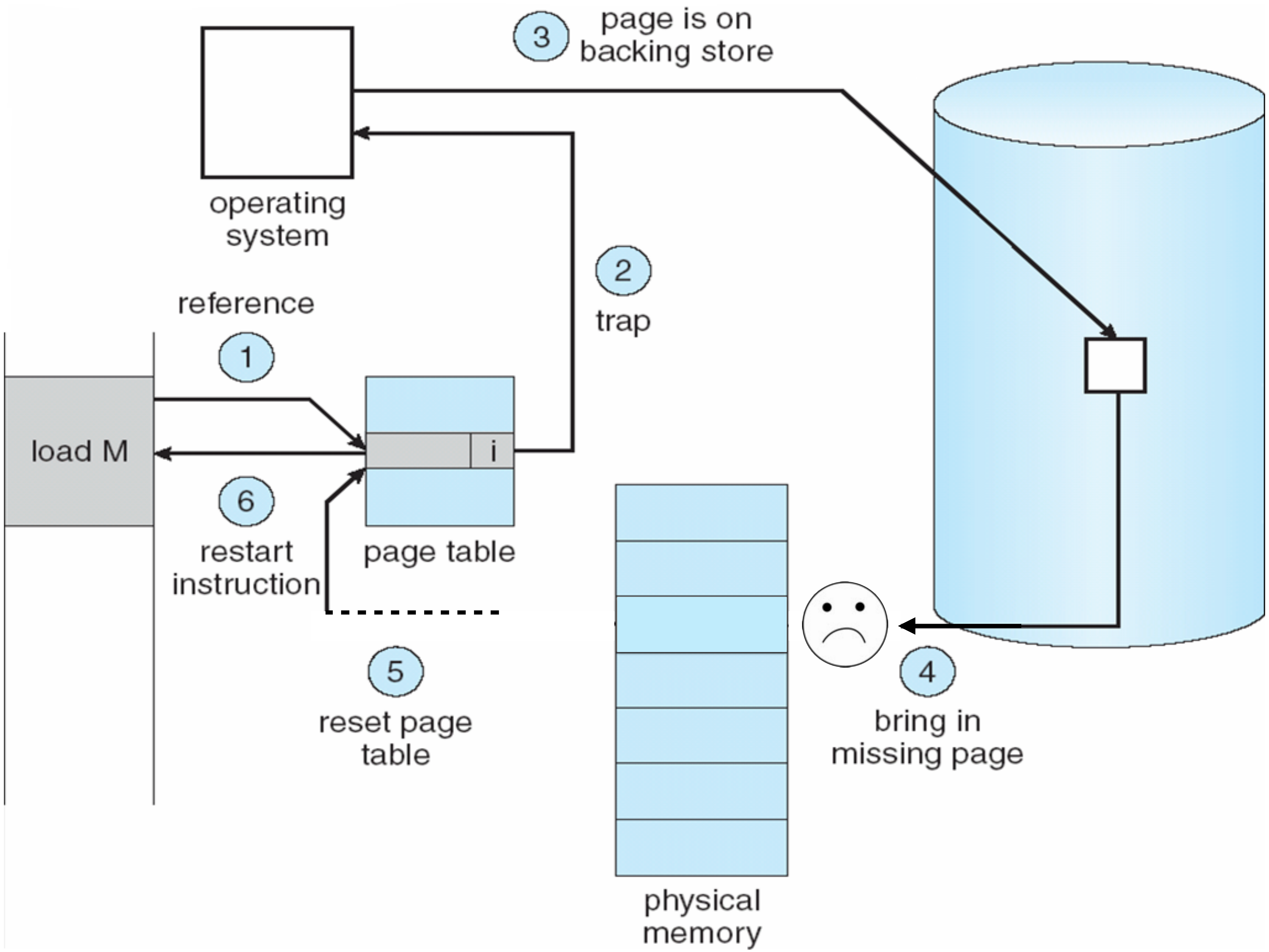
- The OS wants to maximize CPU utilization. As a result, it decides that it is a good idea to increase the degree of multiprogramming by adding new processes to the system.

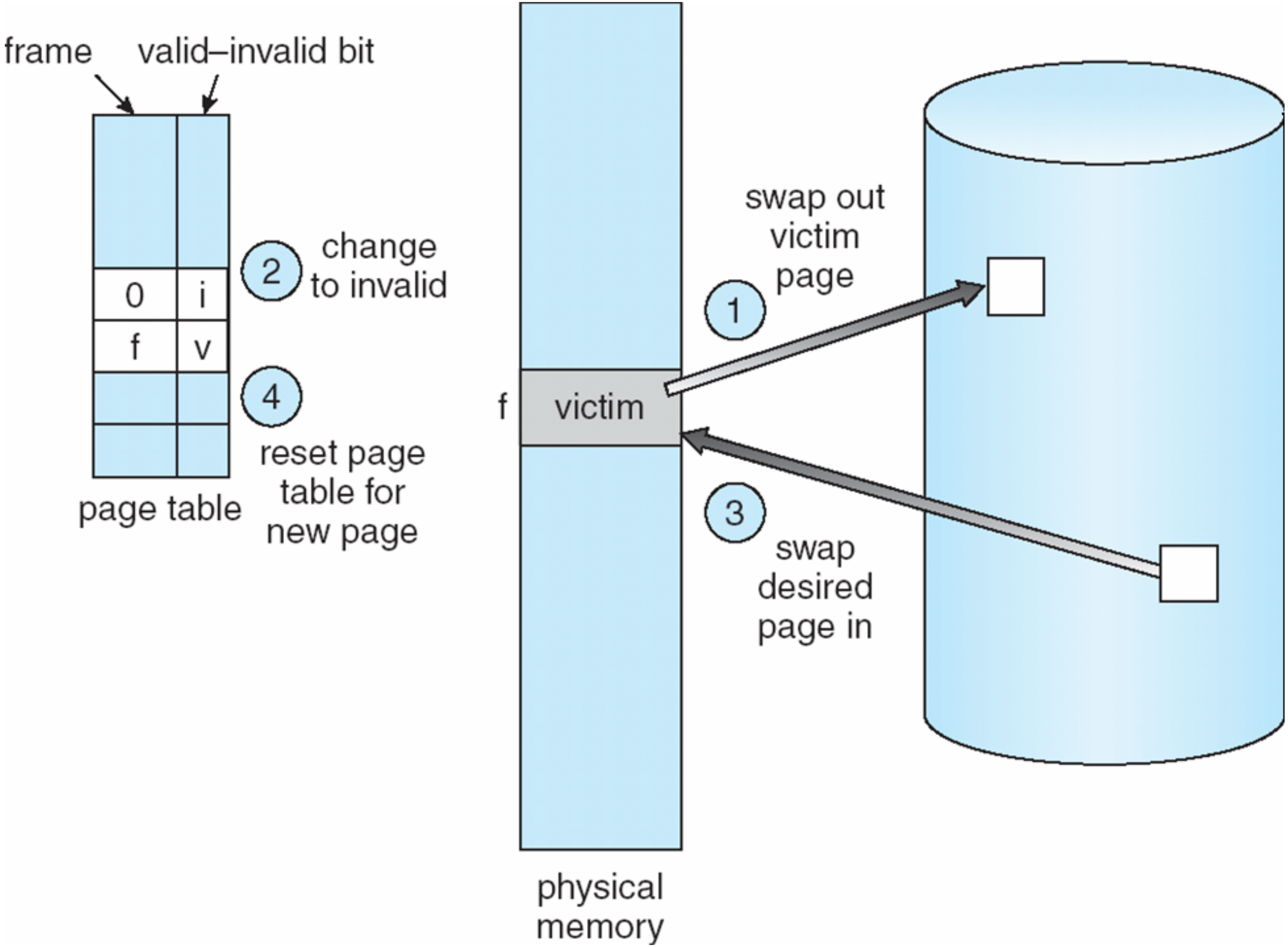


What happens is there is no free frame?



What happens is there is no free frame?





frame
valid–invalid bit
0
i
f
v
2
change to invalid
4
reset page table for new page
page table
f
victim
physical memory
1
swap out victim page
3
swap desired page in

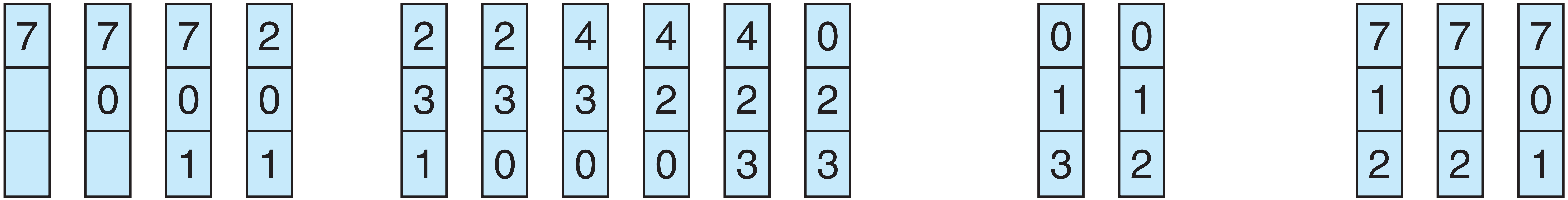
Page-Replacement Algorithms

- FIFO algorithm
- Optimal page-replacement algorithm
- Least-recently used (LRU) algorithm
- Second-chance algorithm (clock)

FIFO Algorithms

reference string

7 0 1 2 0 3 0 4 2 3 0 3 2 1 2 0 1 7 0 1

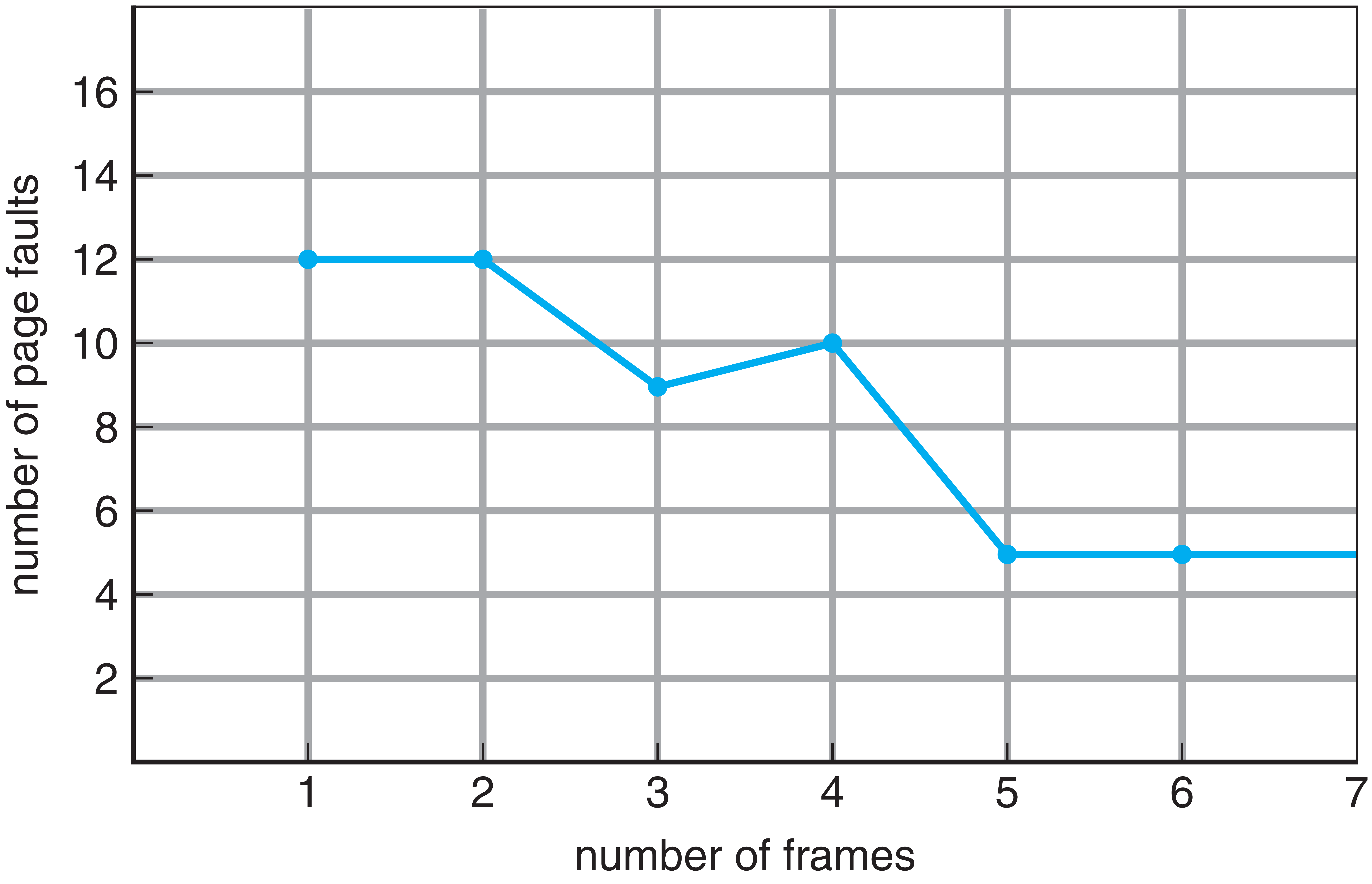


page frames

FIFO Algorithms

Access	Hit/Miss?	Evict	Resulting Cache State	
0	Miss		First-in→	0
1	Miss		First-in→	0, 1
2	Miss		First-in→	0, 1, 2
0	Hit		First-in→	0, 1, 2
1	Hit		First-in→	0, 1, 2
3	Miss	0	First-in→	1, 2, 3
0	Miss	1	First-in→	2, 3, 0
3	Hit		First-in→	2, 3, 0
1	Miss	2	First-in→	3, 0, 1
2	Miss	3	First-in→	0, 1, 2
1	Hit		First-in→	0, 1, 2

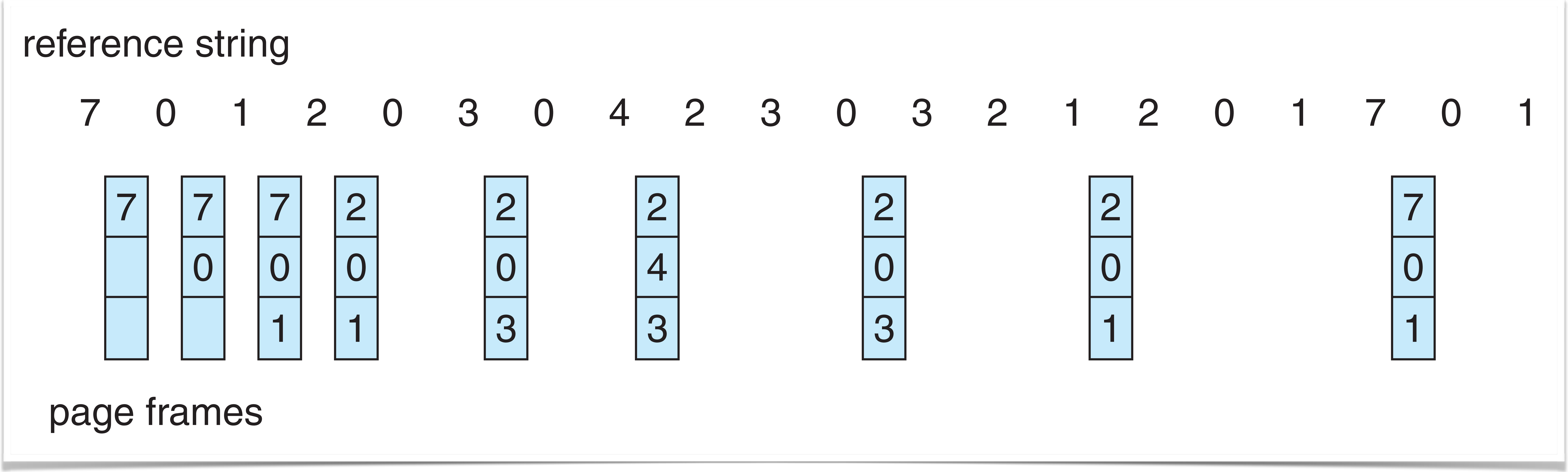
Belady’s anomaly



Paper: L. A. Belady, R. A. Nelson, G. S. Shedler, An anomaly in space-time char- acteristics of certain programs running in paging machine, Comm. ACM , 12, 1 (1969) 349–353.

Optimal Algorithm

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



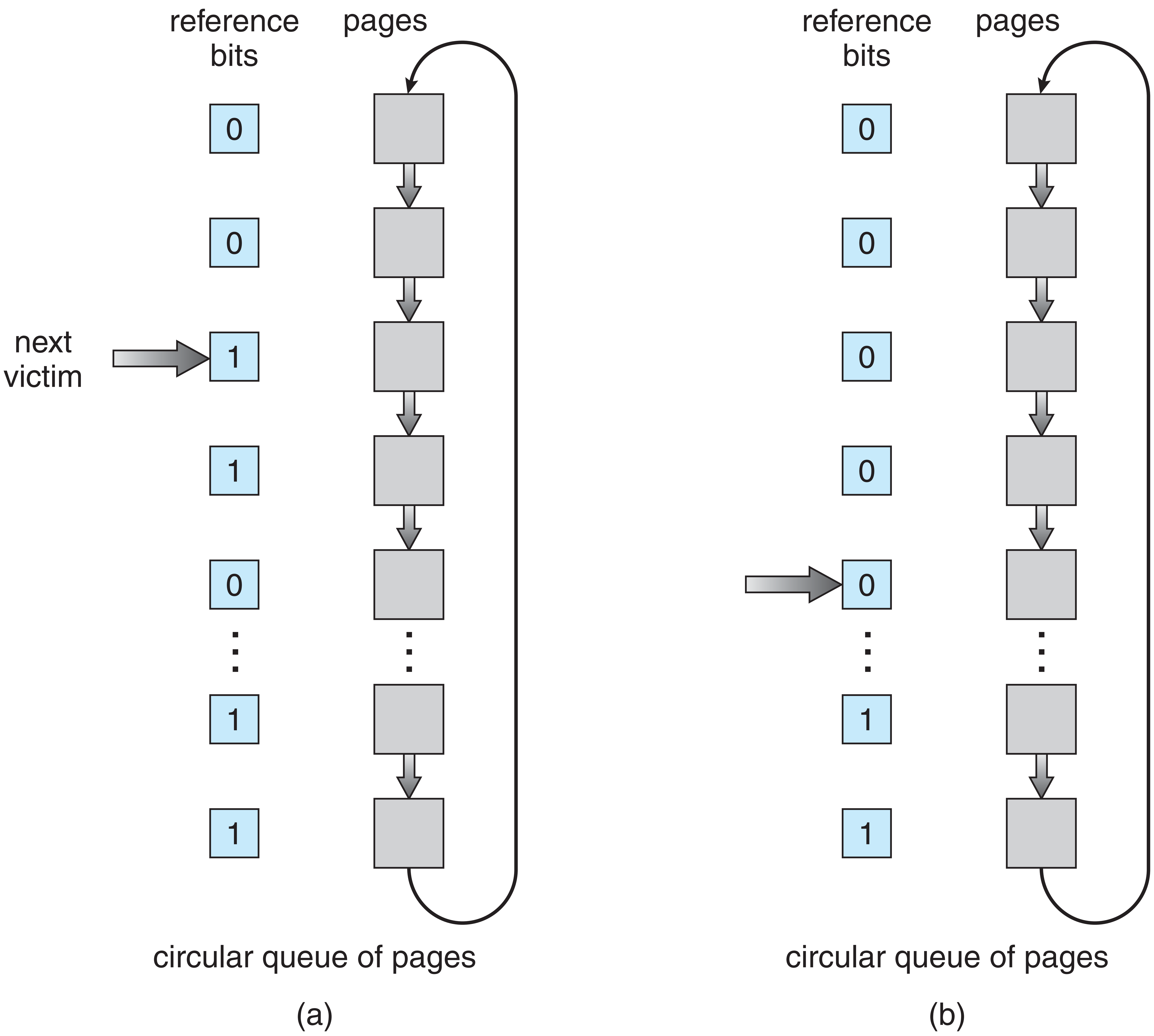
Optimal Algorithm

Access	Hit/Miss?	Evict	Resulting Cache State
0	Miss		0
1	Miss		0, 1
2	Miss		0, 1, 2
0	Hit		0, 1, 2
1	Hit		0, 1, 2
3	Miss	2	0, 1, 3
0	Hit		0, 1, 3
3	Hit		0, 1, 3
1	Hit		0, 1, 3
2	Miss	3	0, 1, 2
1	Hit		0, 1, 2

LRU Algorithm

Access	Hit/Miss?	Evict	Resulting Cache State	
0	Miss		LRU→	0
1	Miss		LRU→	0, 1
2	Miss		LRU→	0, 1, 2
0	Hit		LRU→	1, 2, 0
1	Hit		LRU→	2, 0, 1
3	Miss	2	LRU→	0, 1, 3
0	Hit		LRU→	1, 3, 0
3	Hit		LRU→	1, 0, 3
1	Hit		LRU→	0, 3, 1
2	Miss	0	LRU→	3, 1, 2
1	Hit		LRU→	3, 2, 1

Second-chance Algorithm



Working-set Model

page reference table

. . . 2 6 1 5 7 7 7 7 5 1 6 2 3 4 1 2 3 4 4 4 3 4 3 4 4 4 1 3 2 3 4 4 4 3 4 4 4 . . .

$WS(t_1) = \{1,2,5,6,7\}$

$WS(t_2) = \{3,4\}$

Working Sets and Page-fault frequency