第37 - 38讲 页面置换算法

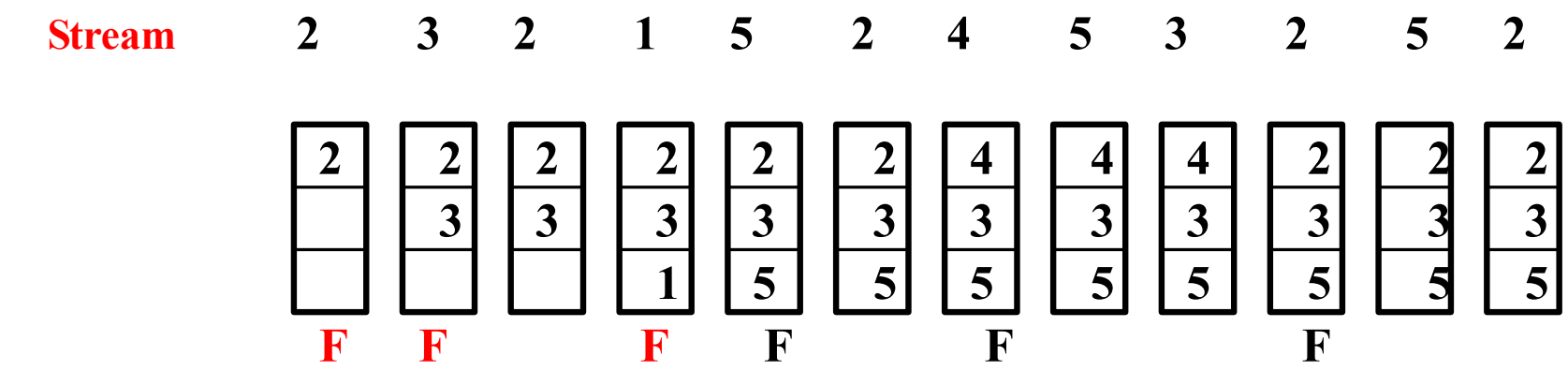
Basic Replacement Algorithms

- Optimal Algorithm(最佳算法)
- Least Recently Used Algorithm(最近最少使用算法)
- First-in First-out Algorithm(先进先出算法)
- Clock Algorithm(时钟算法)

Optimal Algorithm（最佳置换算法）

- 置换**在将来再也不被**访问**的**页**面**
- 置换**在最**远**的将来才被**访问**的**页**面**

Optimal Algorithm



O P T 算 法

Optimal Algorithm

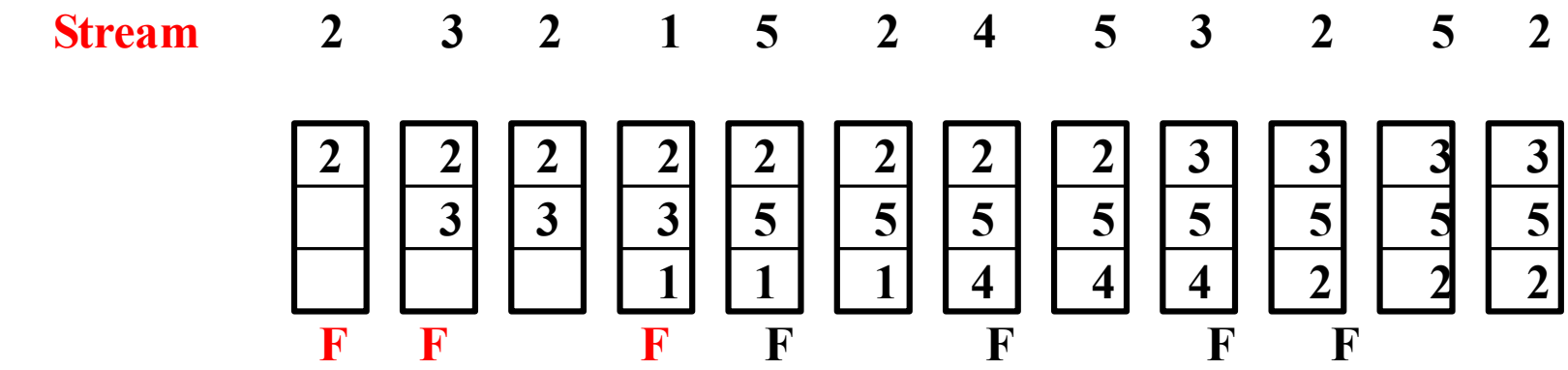
- 从原理上讲， OPT 算法在所有置换算法中具有最佳性能；但， Impossible to have perfect knowledge of future events

- 在 OS 研究领域， OPT 算法常常被用作比较各种置换算法性能的参照标准。

Least Recently Used (LRU) (最近最少使用置换算法)

- Replaces the page that has not been referenced for the longest time.
- By the principle of locality, this should be the page least likely to be referenced in the near future.
- Each page could be tagged with the time of last reference. This would require a great deal of overhead.

Least Recently Used (LRU)



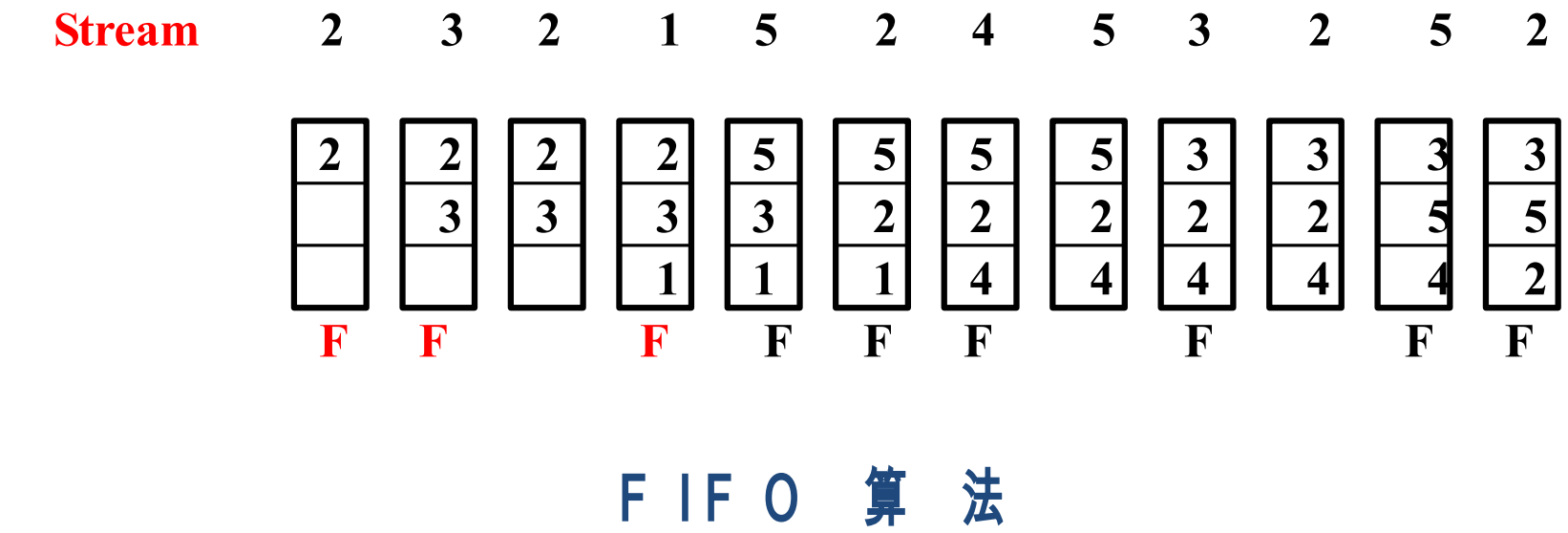
Least Recently Used (LRU)

- 从原理上讲， LRU 算法可以被实现；但任何一种实现方法都将产生很大的系统开销。因此，许多 OS 均使用近似 LRU 算法。
- 一些应用程序具有很强的非局部存储引用（比如顺序存储引用）特征。对于此类应用程序， LRU 算法将有很差的表现。

First-in, first-out (FIFO)

- Treats page frames allocated to a process as a circular buffer.
- Pages are removed in round-robin style.
- Simplest replacement policy to implement.
- Page that has been in memory the longest is replaced.
- These pages may be needed again very soon.

First-in, first-out (FIFO)



First-in, first-out (FIFO)

- FIFO 算法最简单，最容易实现
- 如果应用程序具有顺序存储引用特征，那么使用 FIFO 算法将获得很好的性能。
- 不过，大多数应用程序具有局部存储引用特征。对于此类应用程序， FIFO 算法的性能是很差的。

First-in, first-out (FIFO)

- 容易产生抖动。
- 可能存在 Belady 现象。
- Belady 现象：虚拟存储系统中的一种异常现象，即增加进程的页框数，缺页率反而上升。

First-in, first-out (FIFO)

若某进程的页面访问序列为： 1, 2, 3, 4, 1, 2, 5, 1, 2, 3, 4, 5 。若为进程分配的页框数为 3 ，则缺页 9 次。

FIFO	1	2	3	4	1	2	5	1	2	3	4	5
页 0	1	2	3	4	1	2	5	5	5	3	4	4
页 1		1	2	3	4	1	2	2	2	5	3	3
页 2			1	2	3	4	1	1	1	2	5	5
缺页	x	x	x	x	x	x	x	√	√	x	X	√

First-in, first-out (FIFO)

若为进程分配的页框数为 4，则缺页 10 次。

FIFO	1	2	3	4	1	2	5	1	2	3	4	5
页 0	1	2	3	4	4	4	5	1	2	3	4	5
页 1		1	2	3	3	3	4	5	1	2	3	4
页 2			1	2	2	2	3	4	5	1	2	3
页 3				1	1	1	2	3	4	5	1	2
缺页	x	x	x	x	√	√	x	x	x	x	x	x

思考：为什么 FIFO 算法会出现 Belady 现象？

Clock Policy (时钟置换算法)

- Additional bit called a use bit.
- When a page is first loaded in memory, the use bit is set to 1.
- When the page is referenced, the use bit is set to 1.
- When it is time to replace a page, the first frame encountered with the use bit set to 0 is replaced.
- During the search for replacement, each use bit set to 1 is changed to 0.

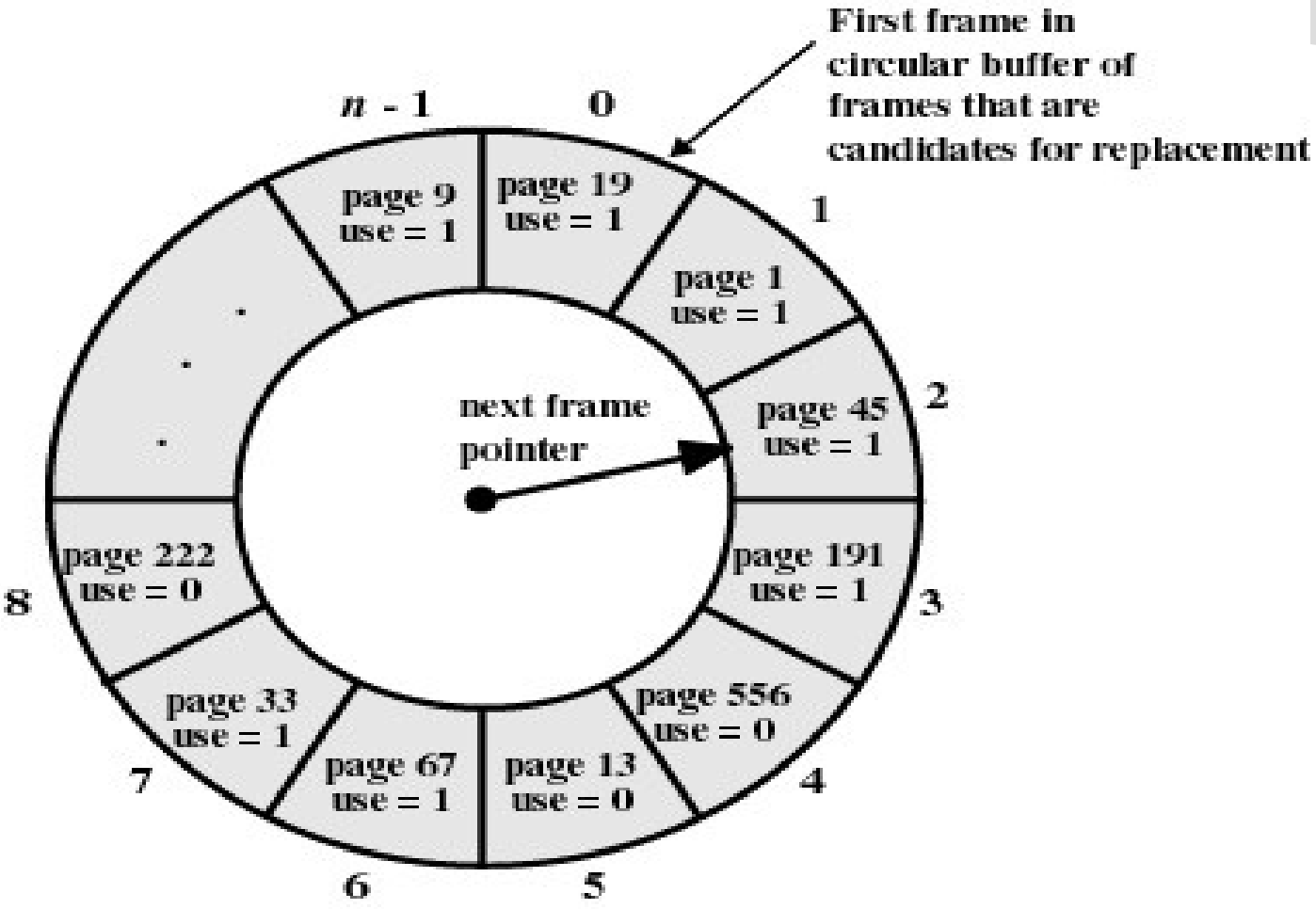
Clock Policy

- CLOCK 算法中，系统将置换范围内的所有 frame 组成一 个环形缓冲区，并为其设置一个*扫描指针*

- 没有进行页面置换时，扫描指针总是指向上一次进行页面置换时被置换页面所在位置的下一个位置。

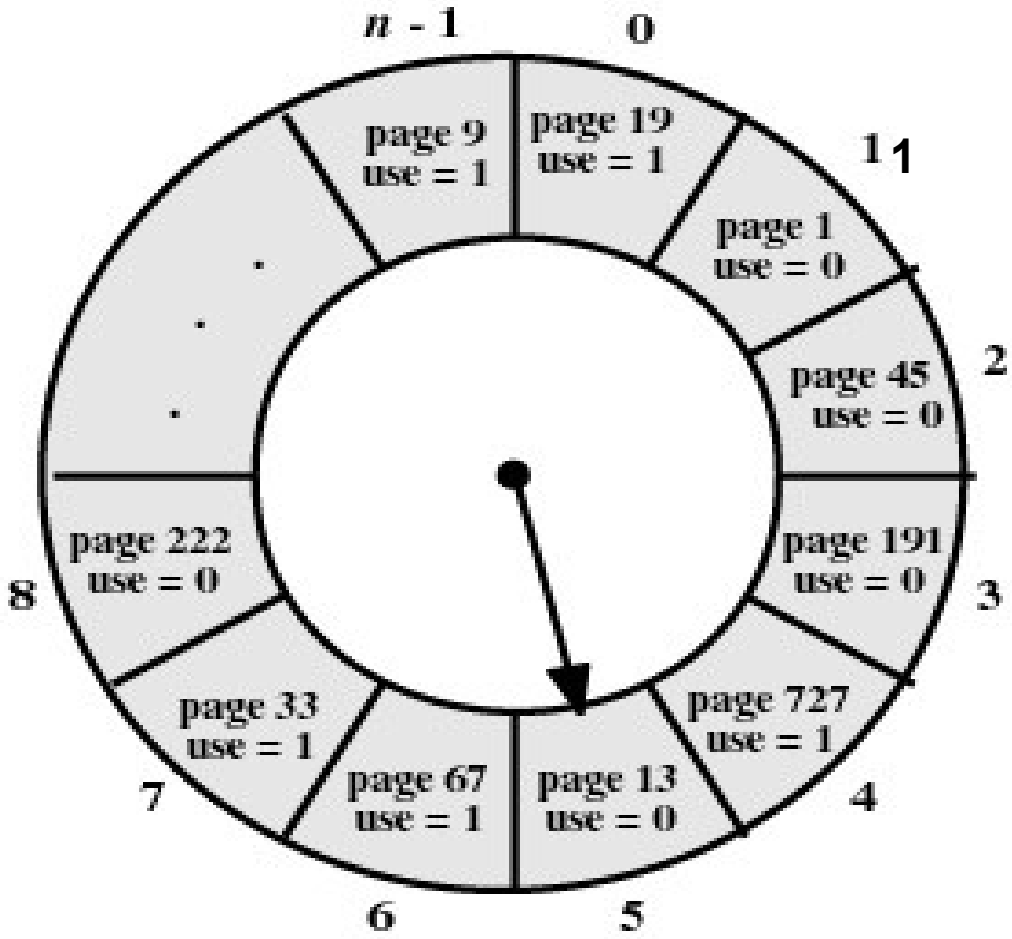
Clock Policy

- 当需要进行页面置换时，系统将移动扫描指针搜索置换范围内的各个 frame 以便找到一个 U 位为 0 的 frame：
  - 如果当前扫描指针所指向的 frame 其 U 位为 1，那么系统将该 frame 的 U 位设置为 0，扫描指针移到下一个位置，继续搜索；
  - 如果当前扫描指针所指向的 frame 其 U 位为 0，则系统将该 frame 中的页面作为被置换页面，同时把扫描指针移到下一个位置，停止搜索。



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

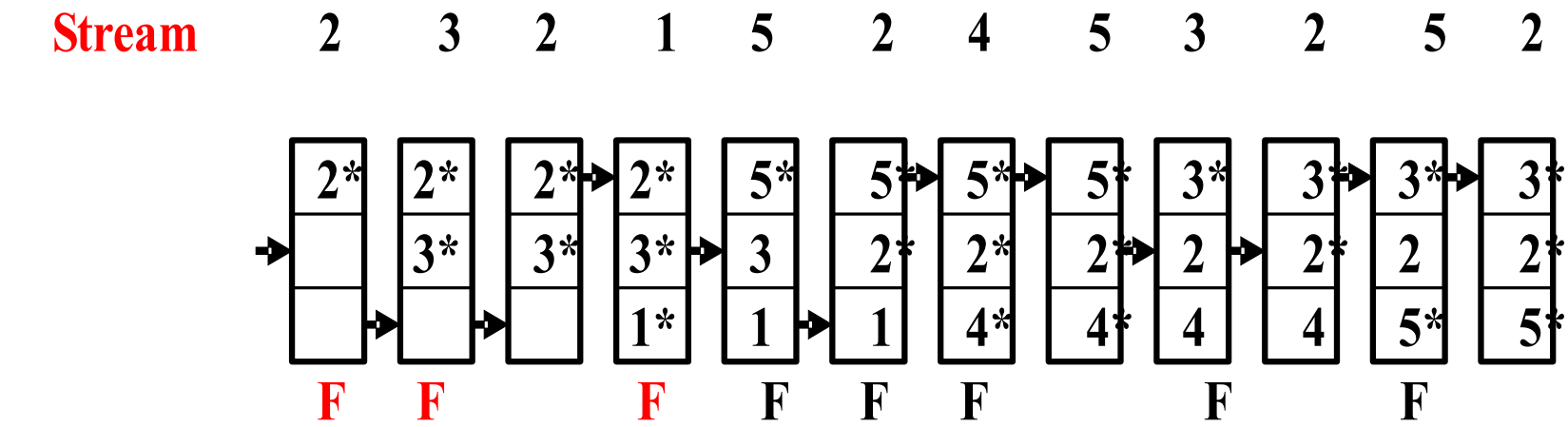
Figure 8.16 Example of Clock Policy Operation



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

Figure 8.16 Example of Clock Policy Operation

Clock Policy



CLOCK 算法

时钟置换算法的改进

- 系统把一个页面移出内存时，如果该页面驻留内存期间没有被修改过，那么不必把它写回辅存，否则系统必须把它写回辅存。这表明，换出未修改过的页面比换出被修改过的页面开销小。

- 显然，可以依据上述结论改进 CLOCK 算法。改进后的 CLOCK 算法将在置换范围内*首选*在最近没有使用过、没有被修改过的页面作为被置换页面。

时钟置换算法的改进

- 系统为内存的每个 frame 配置一个 Modify Bit (简称为 M 位）。
- 改进后的 CLOCK 算法在选择被置换页面时将同时考虑 U 位和 M 位。
- 一个页面的 U 位与 M 位共有四种组合：
  - U=0， M=0 ：最近没有被使用过，也没有被修改过；
  - U=1， M=0 ：最近被使用过，但没有被修改过；
  - U=0， M=1 ：最近没有被使用过，但被修改过；
  - U=1， M=1 ：最近被使用过，也被修改过；

时钟置换算法的改进

- 改进后的 CLOCK 算法按下列步骤选择被置换页面：

  1、从当前位置开始搜索 *U=0 且 M=0* 的 frame。但不修改任何 U 位。若找到第一个 U=0 且 M=0 的 frame，那么系统将该 frame 中的页面置换出去，算法终止。

时钟置换算法的改进

- 2、如果第一步没有成功，那么扫描指针将回到原位。再次搜索 *U=0 但 M=1* 的 frame。此搜索过程中，如果遇到 U 位为 1 的 frame，则将其 U 位修改为 0。若找到第一个 U=0 但 M=1 的 frame，那么系统将该 frame 中的页面选作被置换页面，算法终止。
- 3、如果第二步也没有成功，那么扫描指针将再次回到原位且置换范围内的所有 frame 其 U 位均为 0。此时，算法将返回第一步继续执行。