2020春

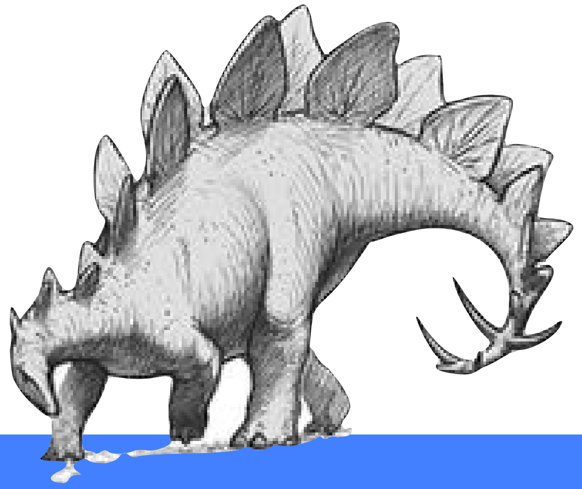
LINUX OPERATING SYSTEM

YANG

LINUX操作系统（双语）

双语课→课件内容中英混排

Lecture 15



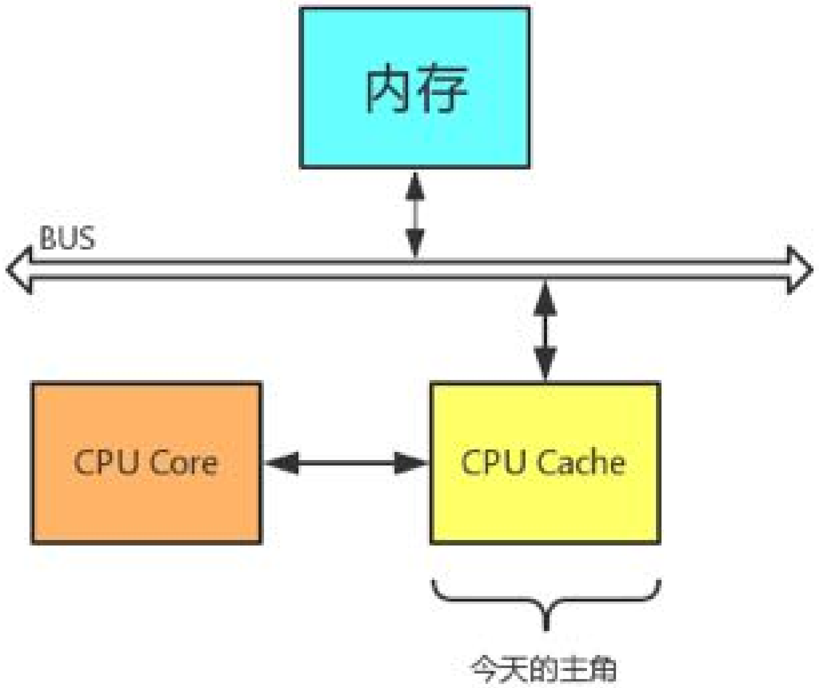
Virtual Memory

本讲内容

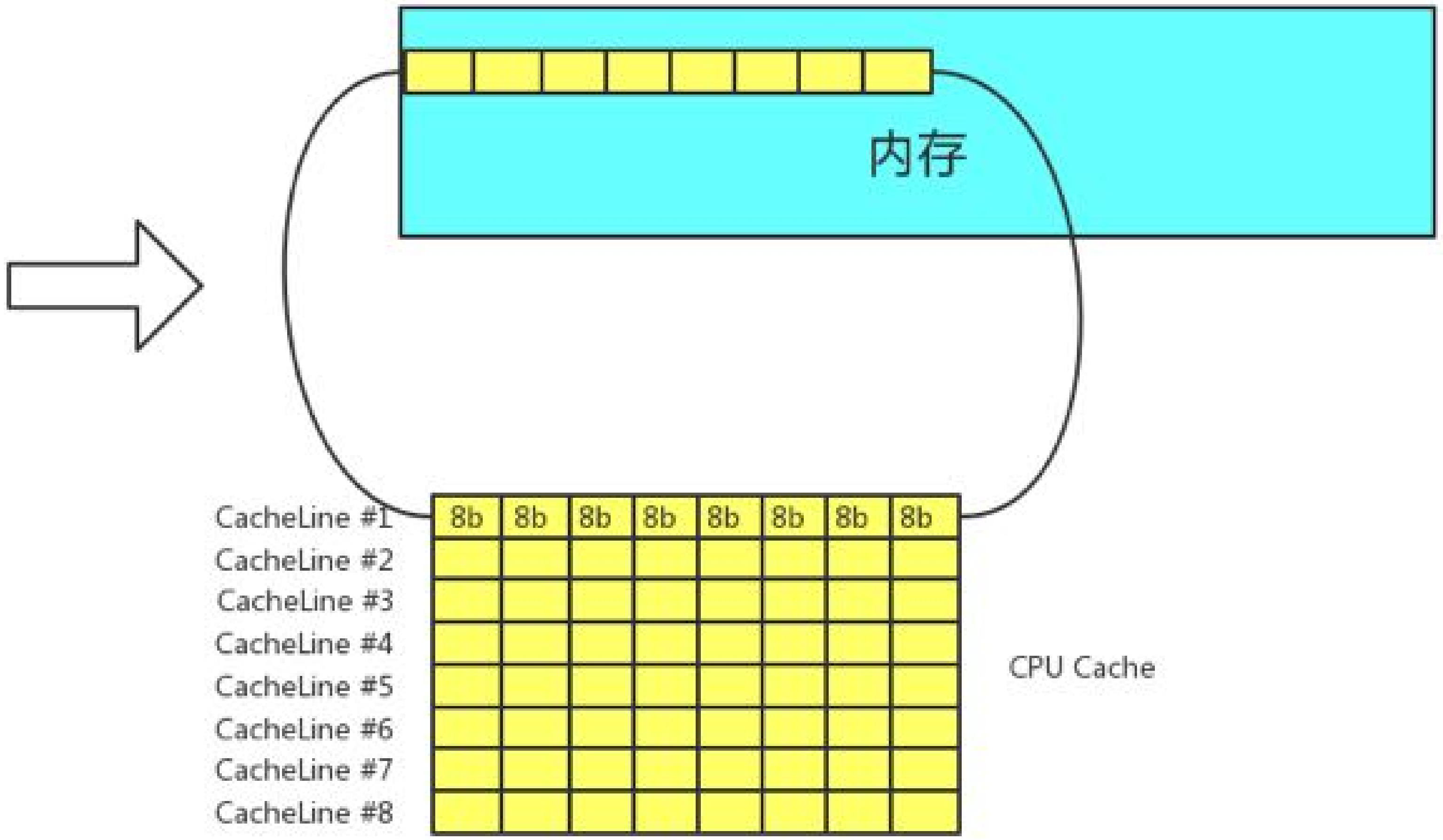
- 局部性原理
- 虚拟内存
- 请求调页
- 页面置换算法
- 系统抖动

局部性原理

缓存CACHE

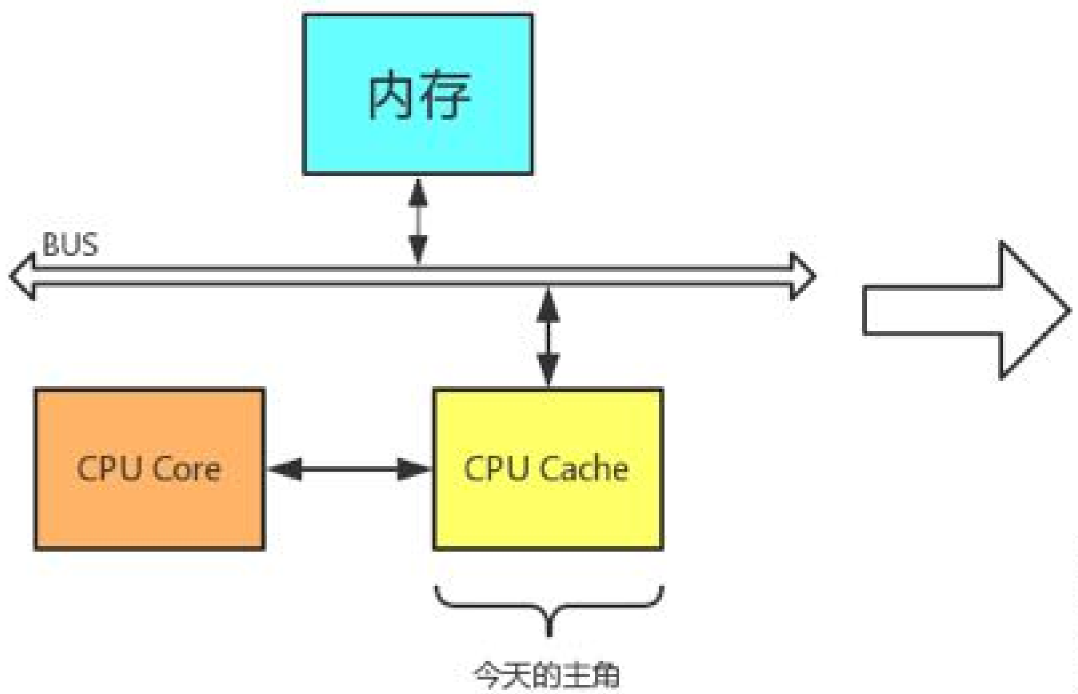


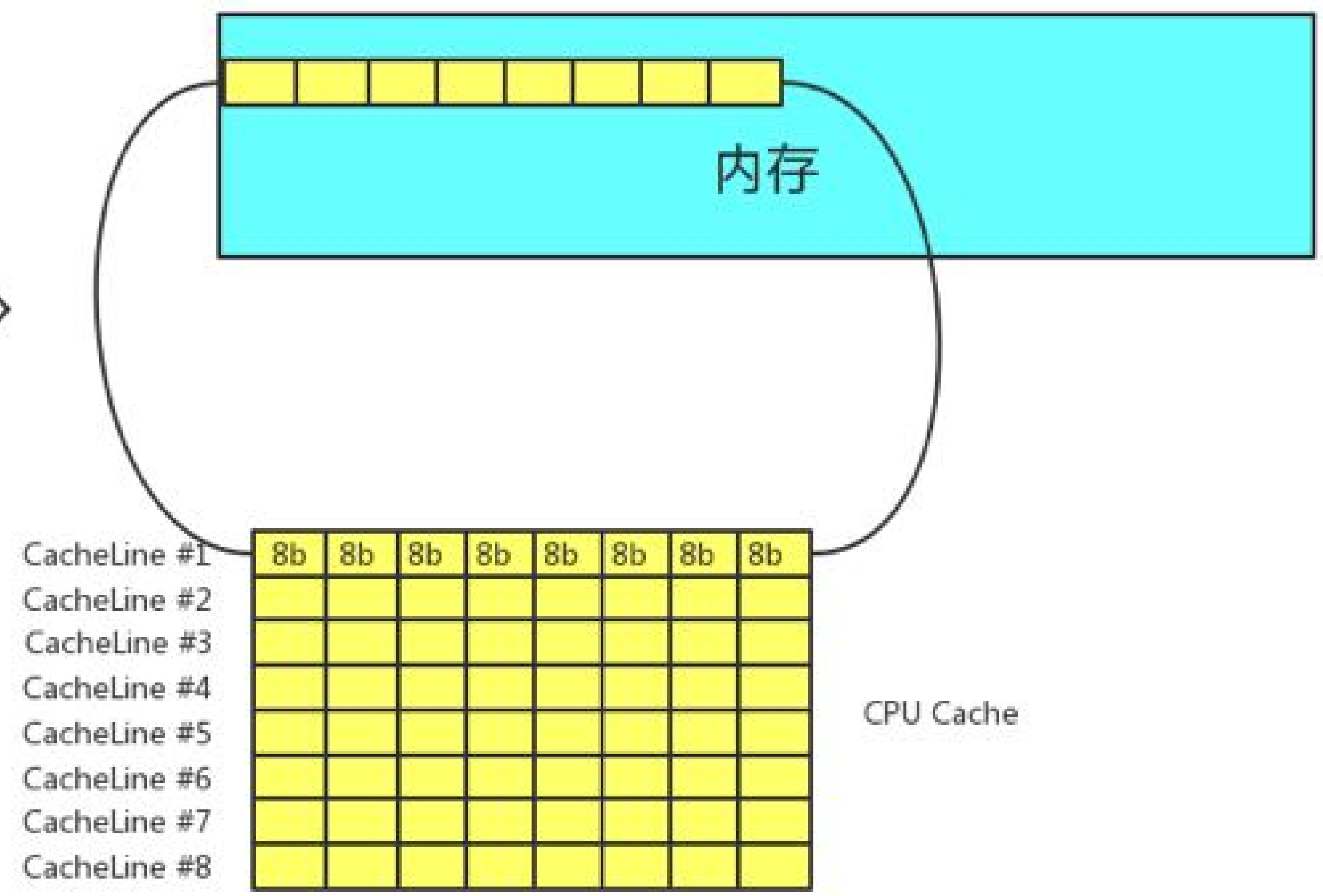
内存
BUS
CPU Core
CPU Cache
今天的主角



内存
CacheLine #1
CacheLine #2
CacheLine #3
CacheLine #4
CacheLine #5
CacheLine #6
CacheLine #7
CacheLine #8
8b 8b 8b 8b 8b 8b 8b 8b
CPU Cache

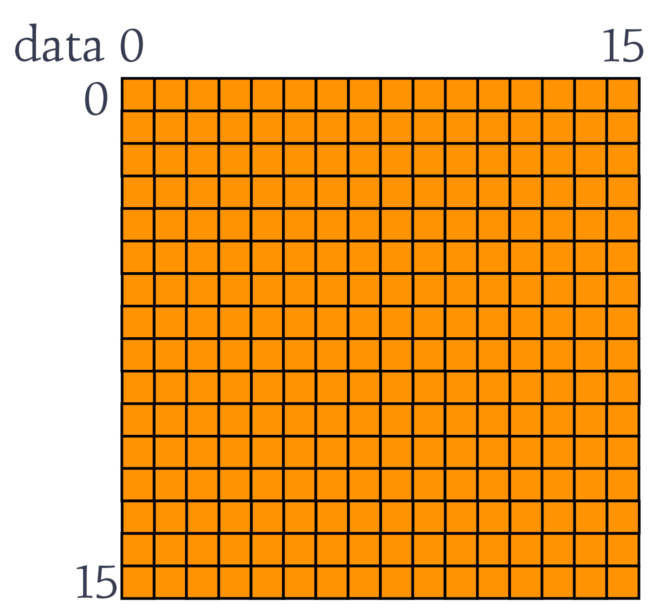
缓存CACHE



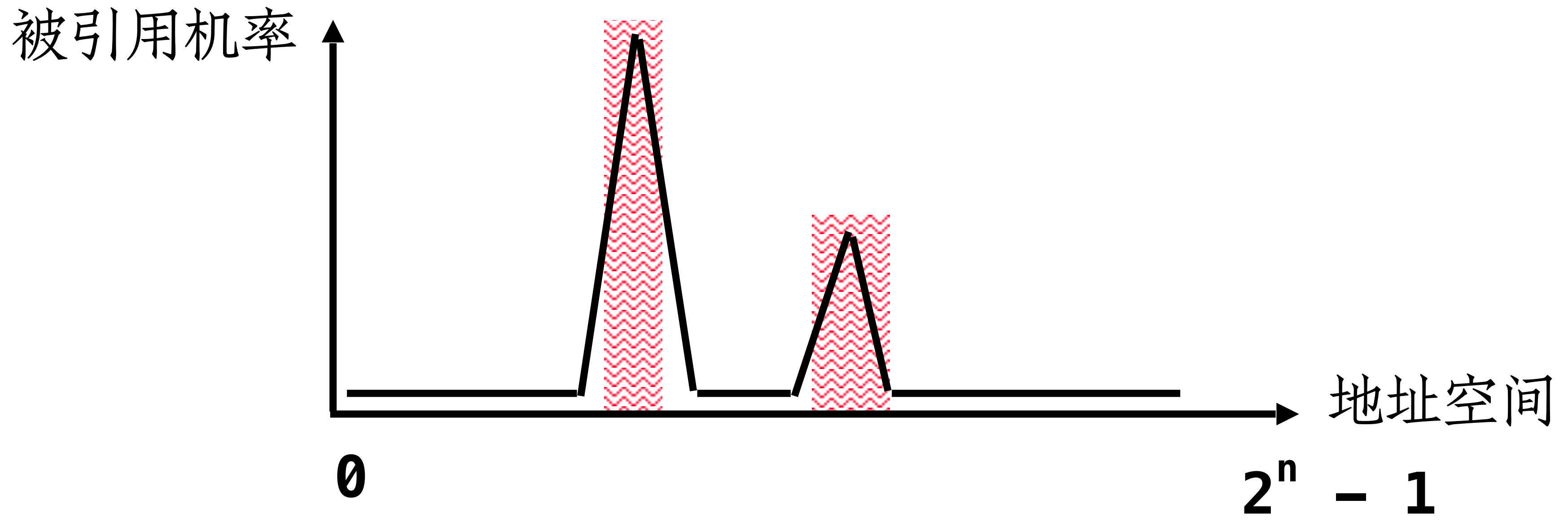



```
int data[16][16];
for(i=0;i<16;i++)
  for(j=0;j<16;j++)
     read data[i][j];
```

```
int data[16][16];
for(j=0;i<16;i++)
  for(i=0;j<16;j++)
     read data[i][j];
```



哪些数据放在缓存?



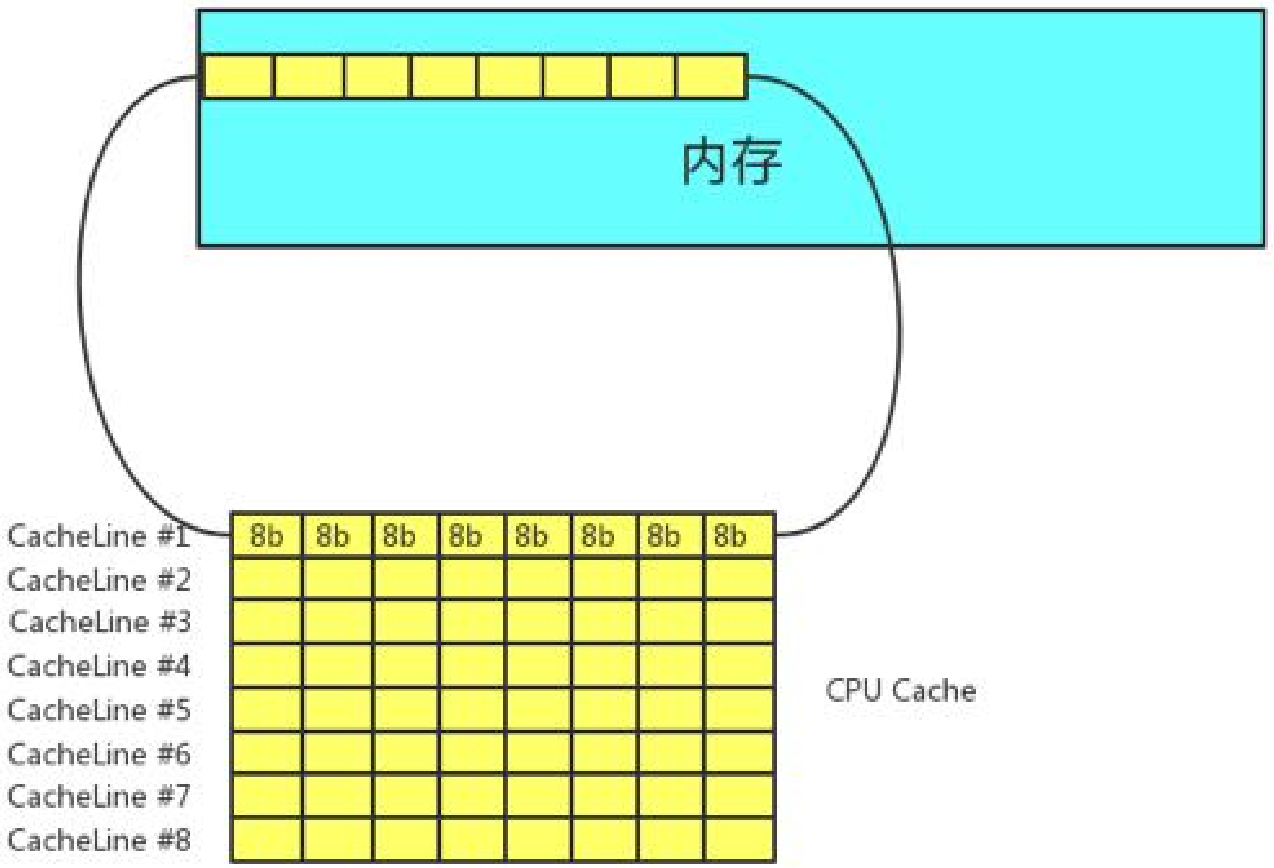
你读懂了这张图的含义么?

局部性原理

- 时间局部性(Temporal locality)
  - 如果某个信息这次被访问，那它有可能在不久的未来被多次访问。
- 空间局部性(Spatial locality)
  - 如果某个位置的信息被访问，那和它相邻的信息也很有可能被访问到。
- 内存局部性(Memory locality)
  - 访问内存时，大概率会访问连续的块，而不是单一的内存地址，其实就是空间局部性在内存上的体现。
- 分支局部性(Branch locality)
  - 计算机中大部分指令是顺序执行，顺序执行和非顺序执行的比例大致是5:1。
- 等距局部性(Equidistant locality)
  - 等距局部性是指如果某个位置被访问，那和它相邻等距离的连续地址极有可能会被访问到。

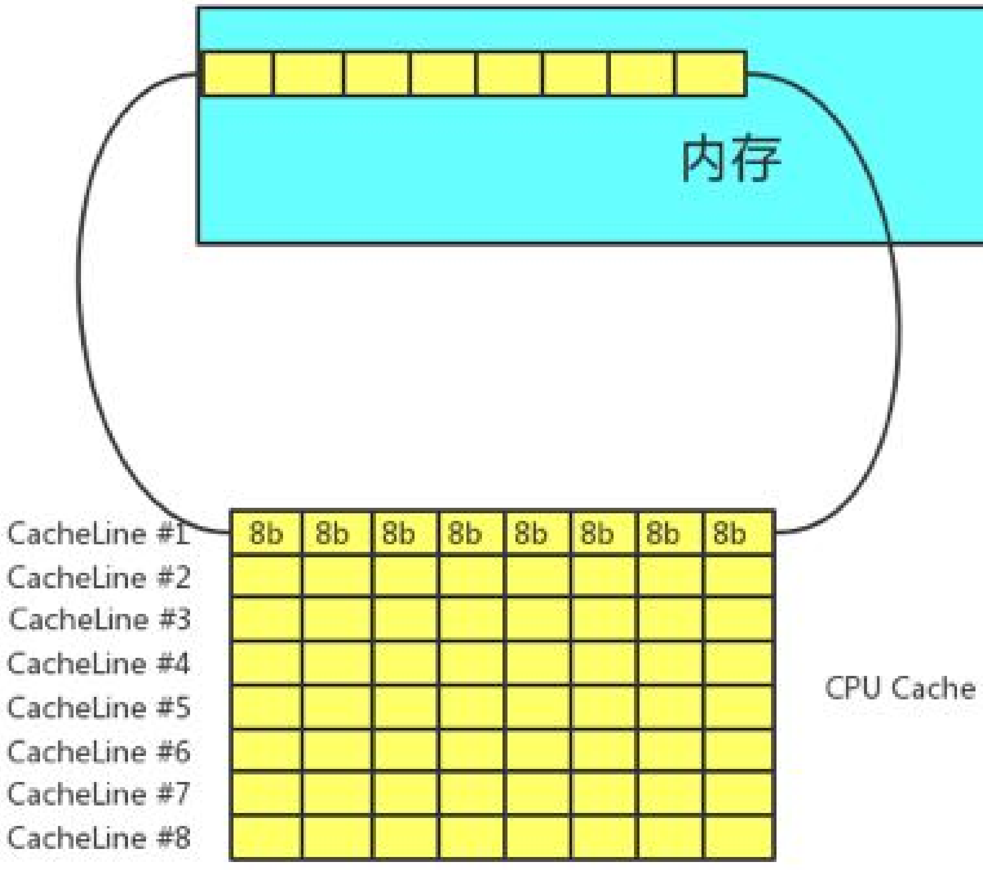
修改缓存数据

- Write through
  - 修改缓存数据的同时修改内存数据
- Write back
  - 只修改缓存数据，直到该数据要被清除出缓存再修改内存中的数据



缓存数据的淘汰

- 缓存的容量很小，当缓存满的时候，就需要将缓存中的部分数据淘汰，装入新的数据。
- 对于淘汰算法，元芳们，你们怎么看？

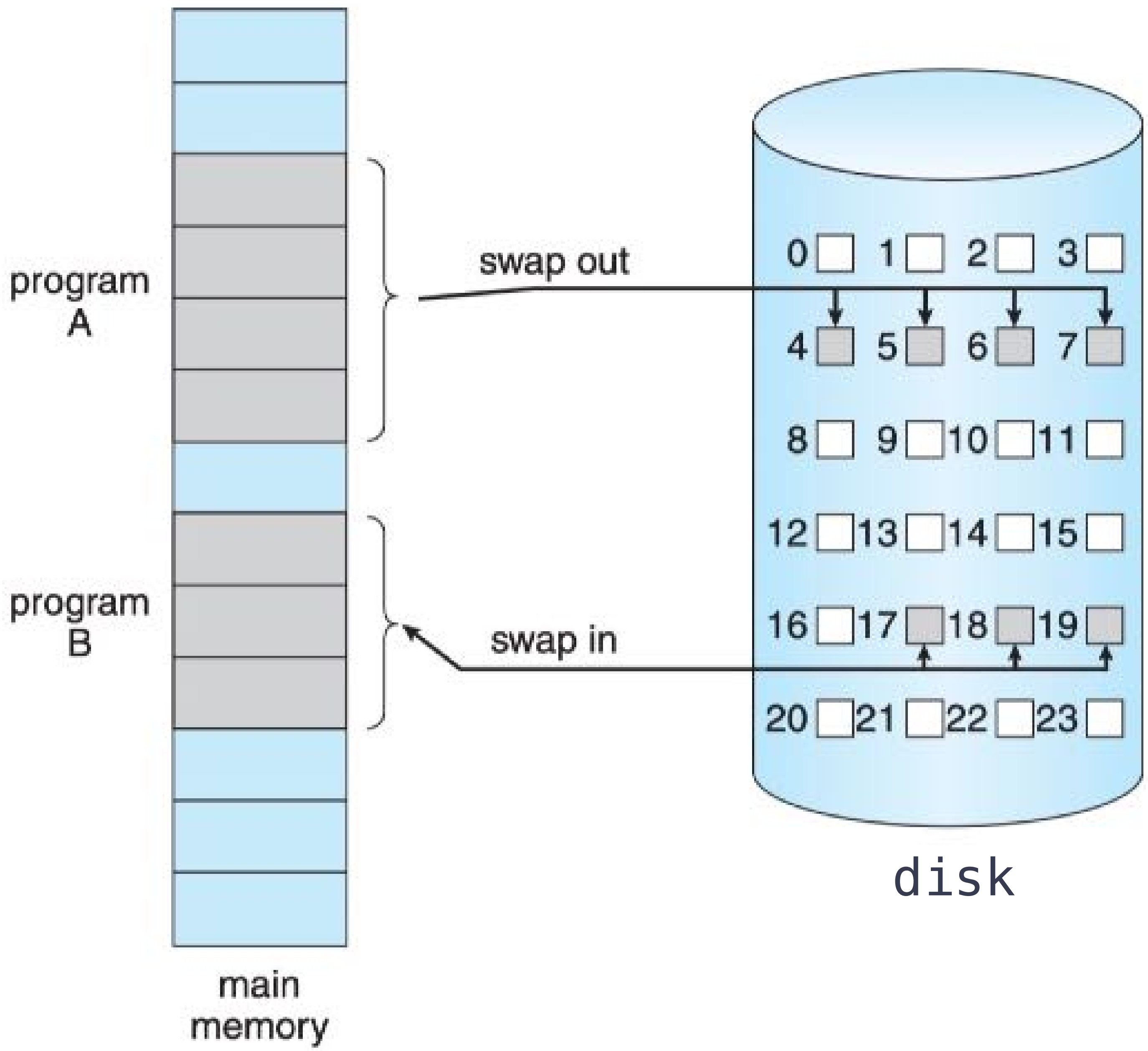


虚拟内存

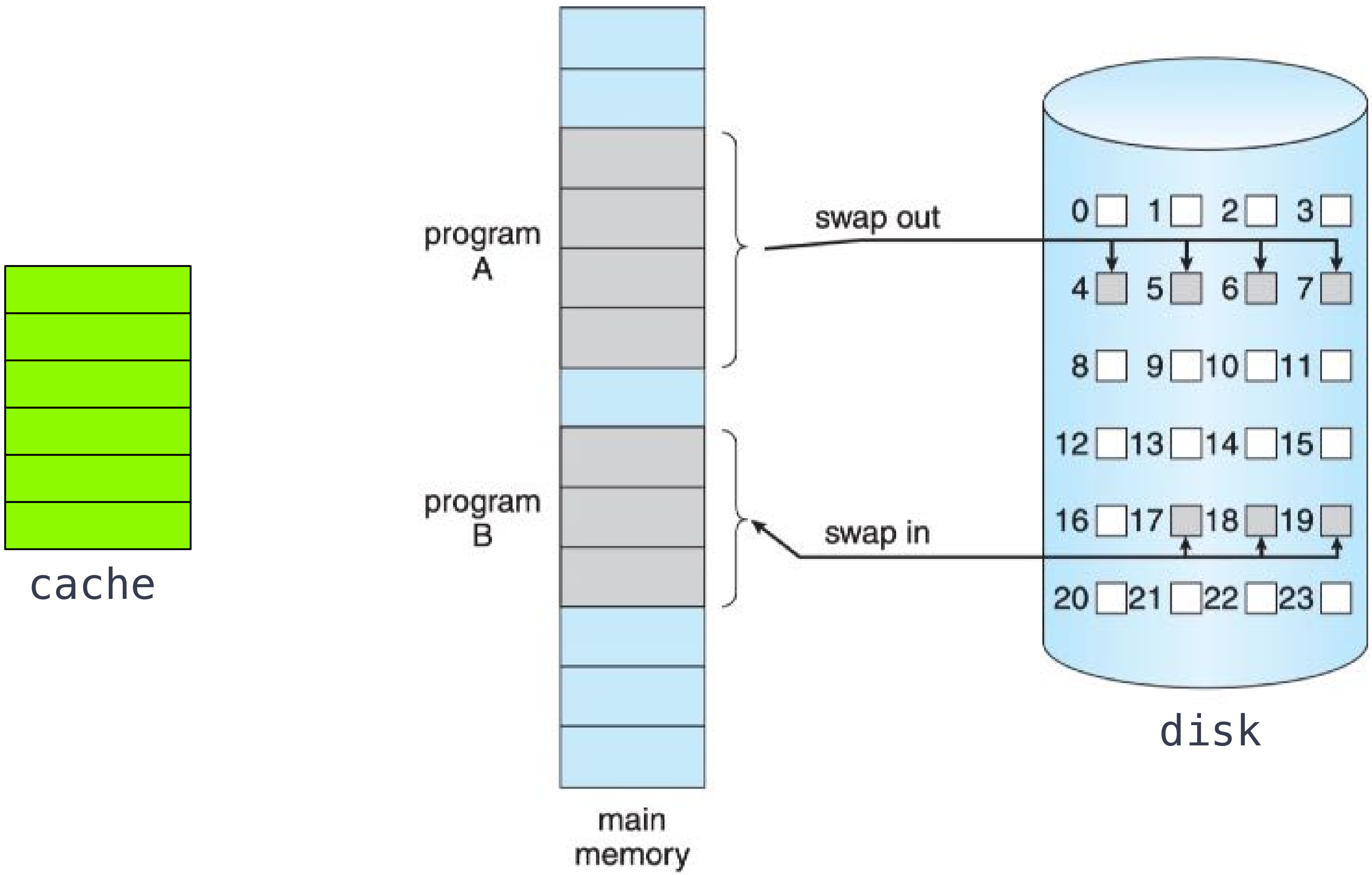
部分装入和部分对换

- 部分装入
  - 进程运行时仅加载部分进入内存，而不必全部装入
  - 其余部分暂时放在swap space
- 部分对换
  - 可以将进程部分对换出内存，用以腾出内存空间
  - 对换出的部分暂时放在swap space

SWAP SAPCE


program
A
program
B
main
memory
swap out
swap in
0 1 2 3
4 5 6 7
8 9 10 11
12 13 14 15
16 17 18 19
20 21 22 23
disk

STORAGE HIERARCHY



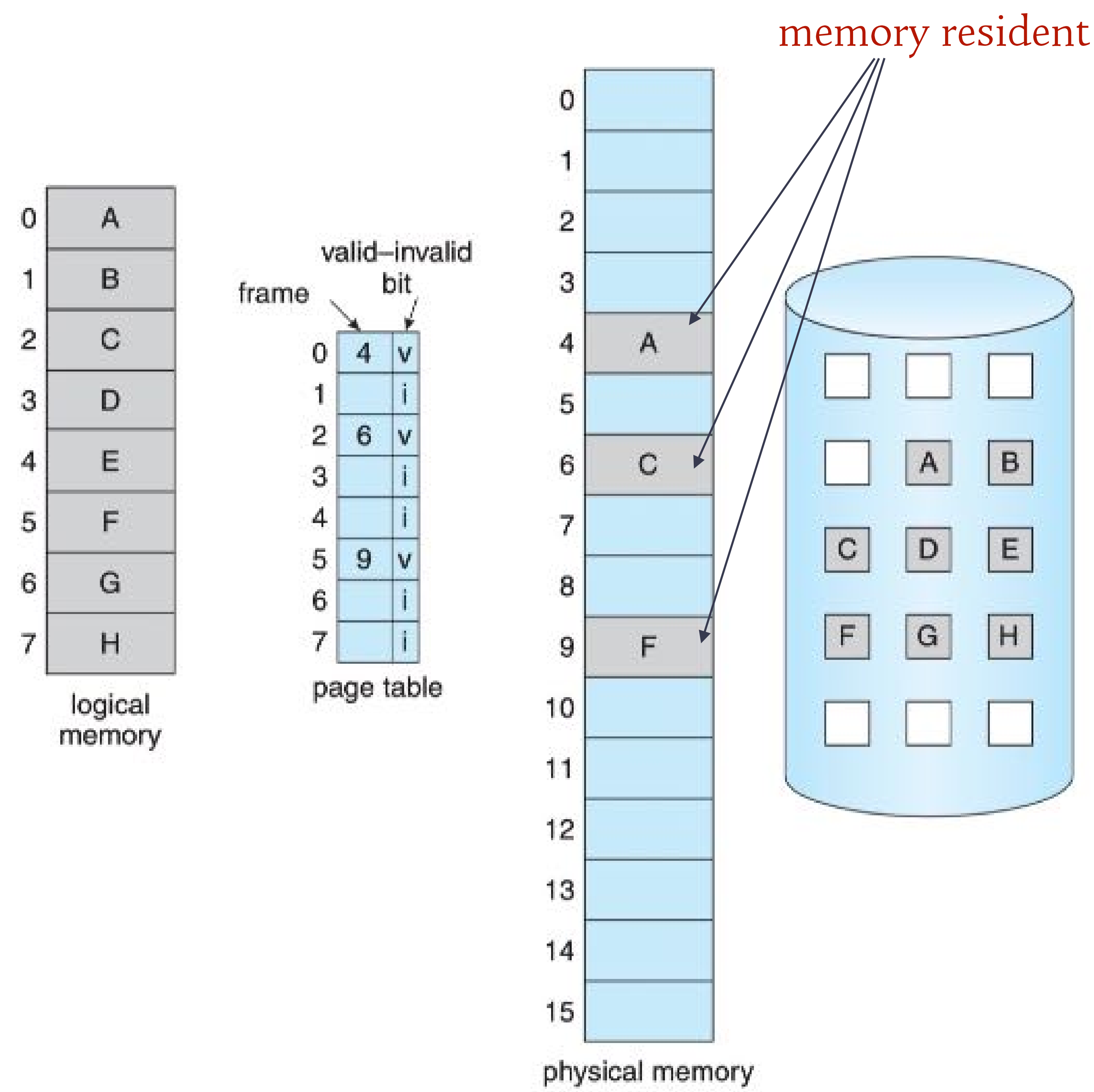
VIRTUAL MEMORY

- Virtual memory is a technique that allows the execution of processes that are not completely in memory.
- One major advantage of this scheme is that programs can be larger than physical memory.
- Further, virtual memory abstracts main memory into an extremely large, uniform array of storage, separating logical memory as viewed by the user from physical memory.
- This technique frees programmers from the concerns of memory-storage limitations.

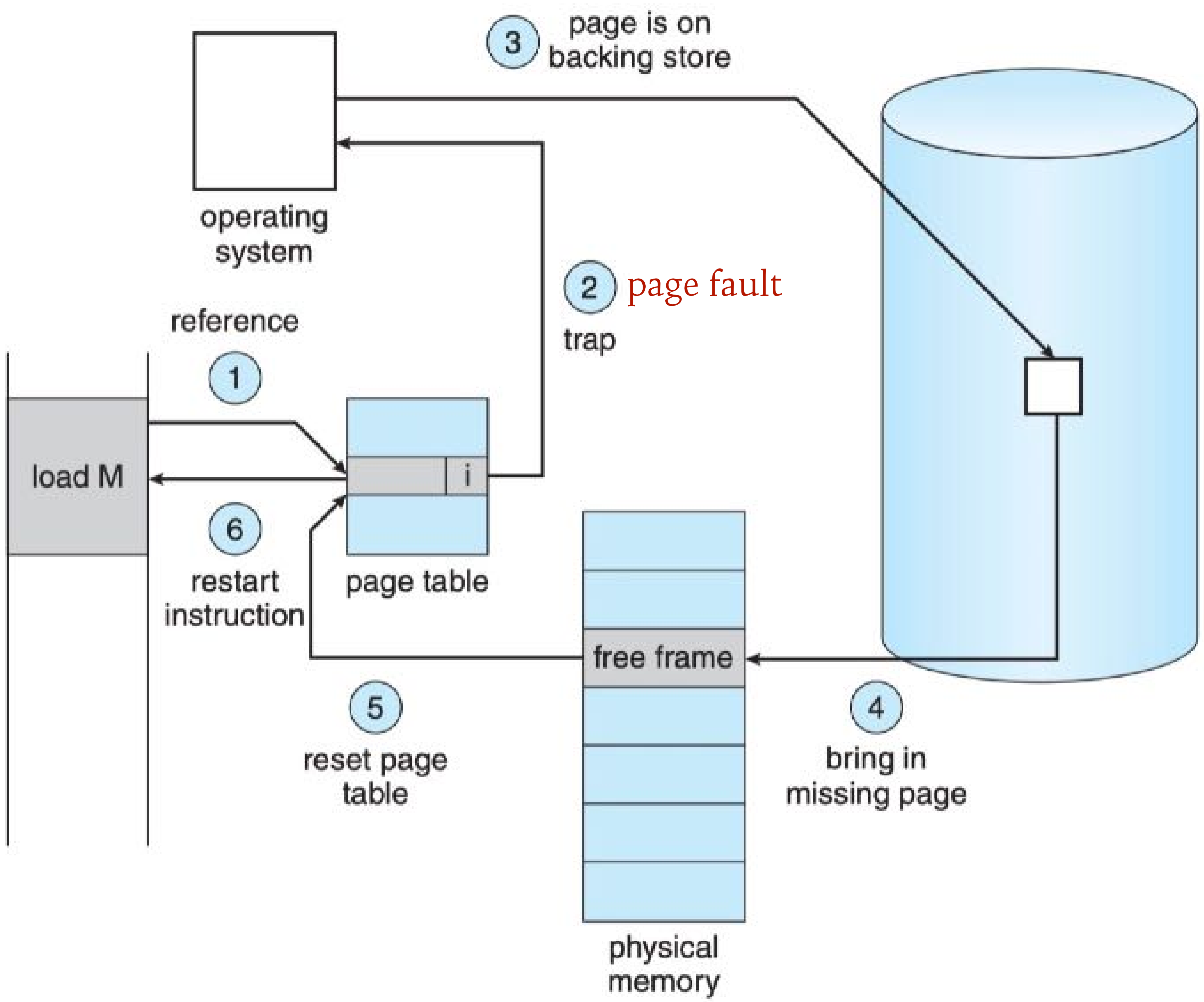
请求调页

DEMAND PAGING

- With demand-paged virtual memory, pages are loaded only when they are demanded during program execution.
- Pages that are never accessed are thus never loaded into physical memory.



请求调页步骤



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

请求调页的性能

- 假设访问内存时间为ma，处理一次缺页中断的时间记作page fault time，令p为缺页中断的出现几率，则有效访问时间的计算公式为：

```
effective access time = (1 – p) × ma + p × page fault time
```

- 若ma=200ns，page fault time=8ms，p=0.001，则

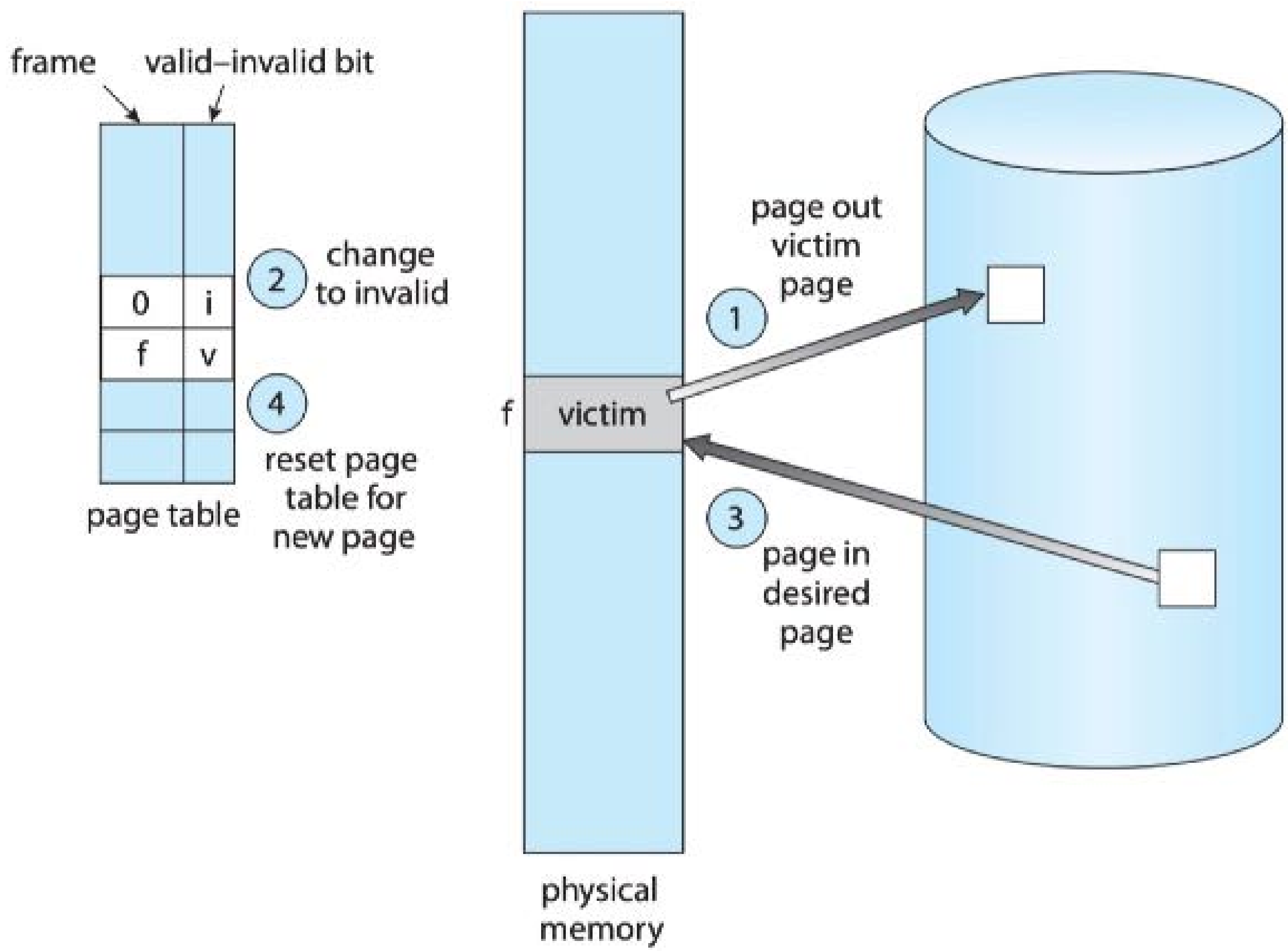
```
effective access time = 8200ns
```

- 缺页中断率p对性能影响重大

页面置换算法

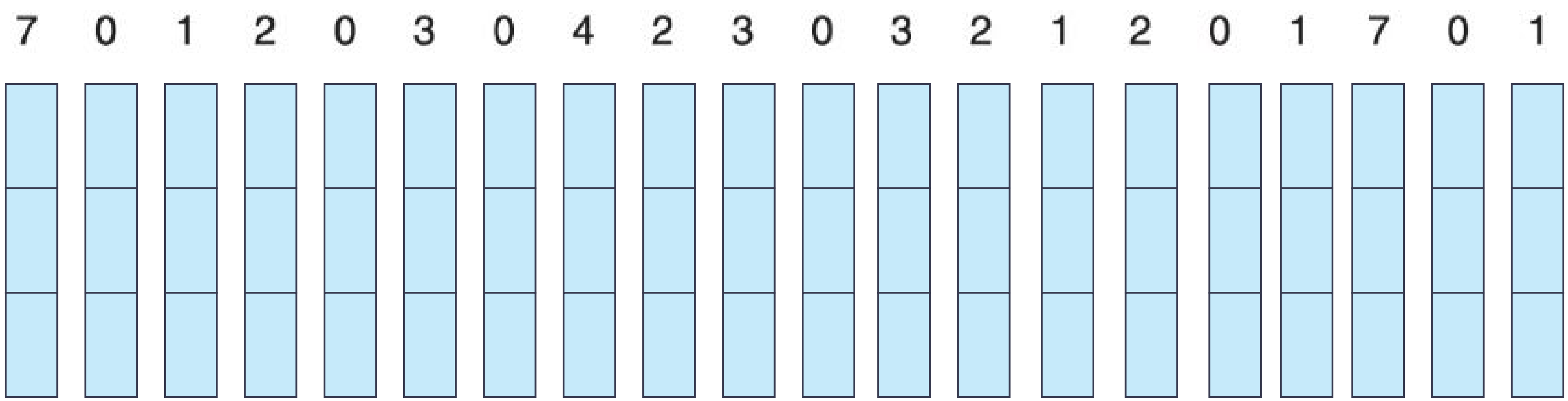
页面置换

- 当进程在执行过程中发生了缺页，在请求调页的时候发现内存已经没有空闲页框可用，操作系统在此时会做出一个处理：页面置换。



FIFO

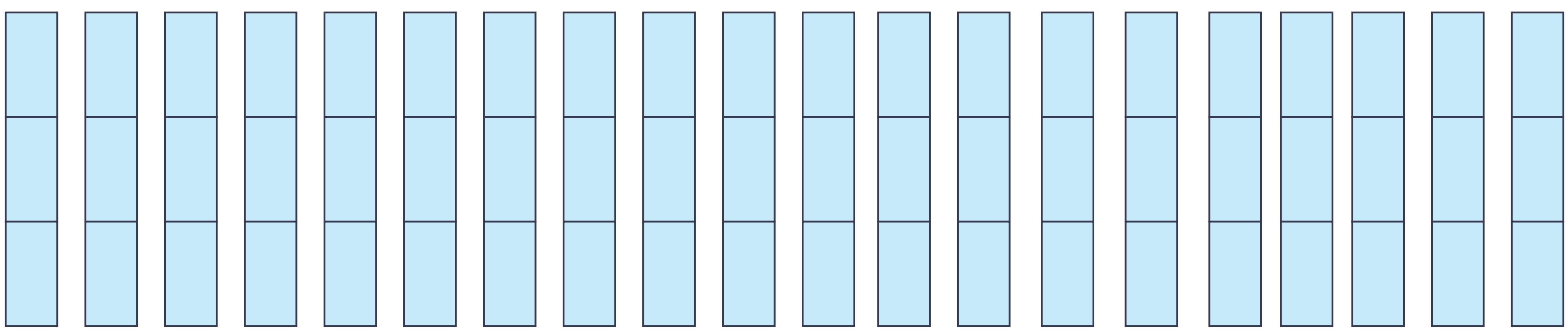
- 总是淘汰最先进入内存的页面，因为它在内存中待的时间最久。
- 假设分配给一个进程3个页框，按照下面的页面访问顺序，计算缺页中断次数。



OPTIMAL

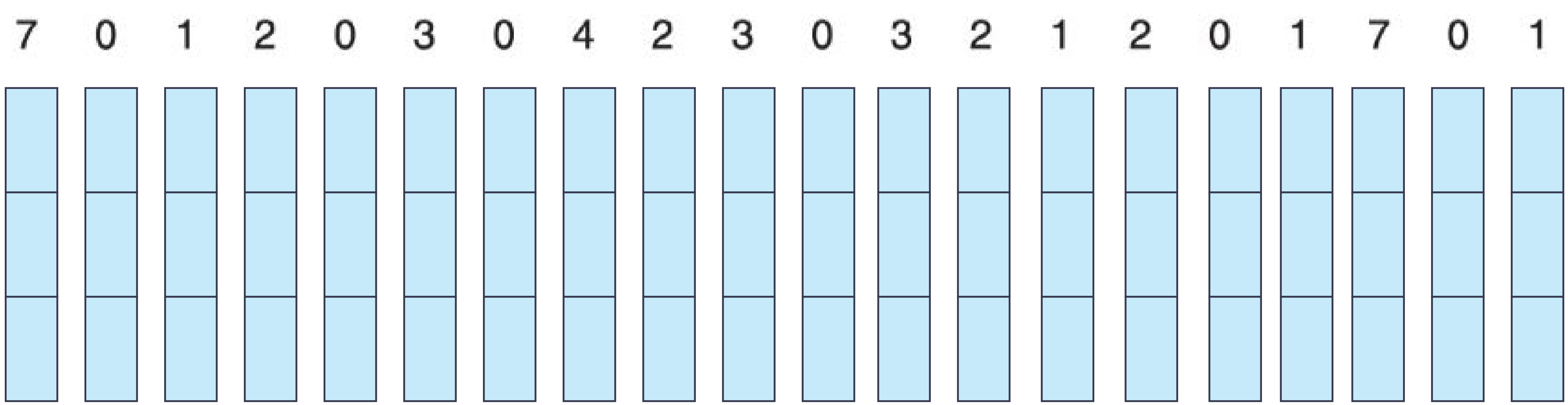
- 总是淘汰最长时间不会再使用的页面。
- 假设分配给一个进程3个页框，按照下面的页面访问顺序，计算缺页中断次数。

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



LRU（LEAST RECENT UNUSED）

- 总是淘汰最近最少使用的页面。
- 假设分配给一个进程3个页框，按照下面的页面访问顺序，计算缺页中断次数。

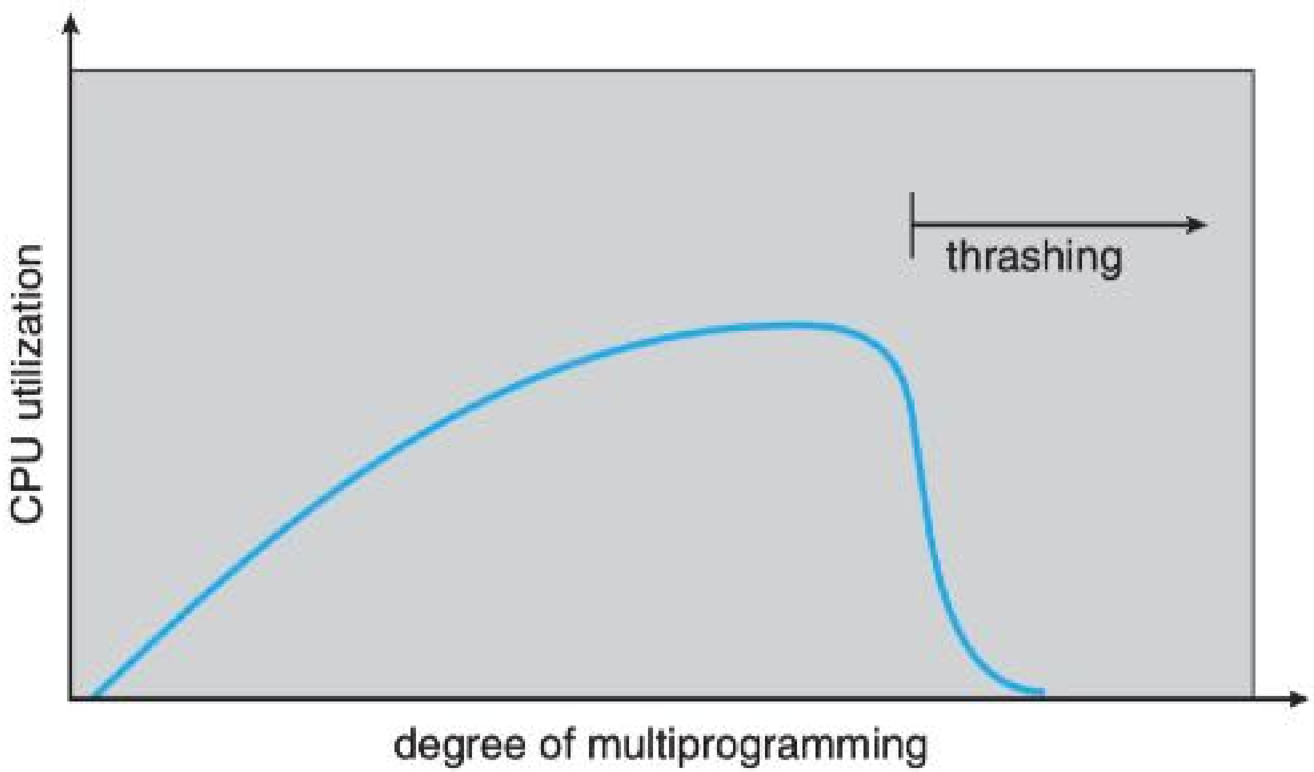


系统抖动

THRASHING

- If the process does not have the number of frames it needs to support pages in active use, it will quickly page-fault. At this point, it must replace some page. However, since all its pages are in active use, it must replace a page that will be needed again right away. Consequently, it quickly faults again, and again, and again, replacing pages that it must bring back in immediately.
- This high paging activity is called thrashing. A process is thrashing if it is spending more time paging than executing.

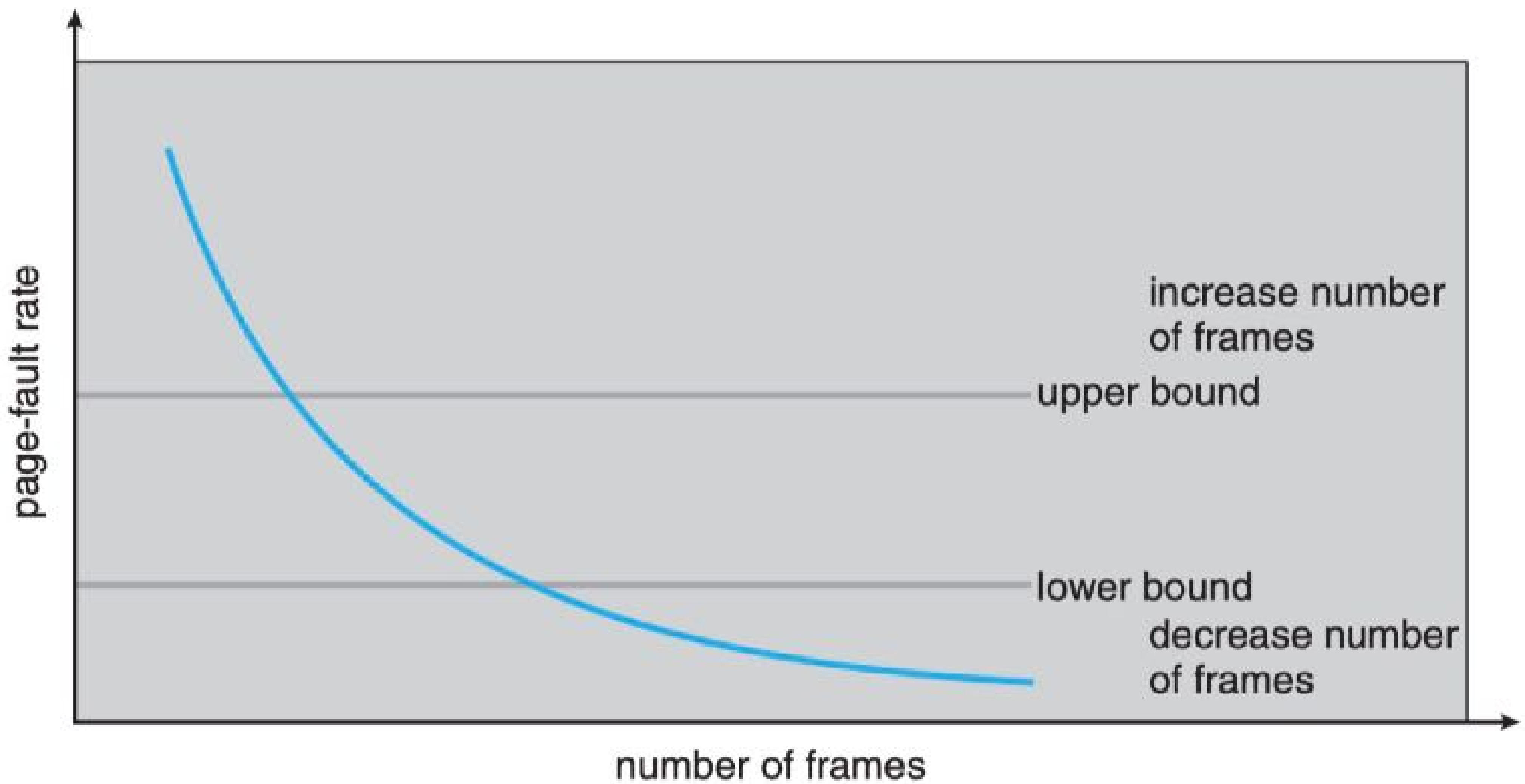
抖动的原因



- 并发进程数量过多
- 进程页框分配不合理

PAGE FAULT FREQUENCY

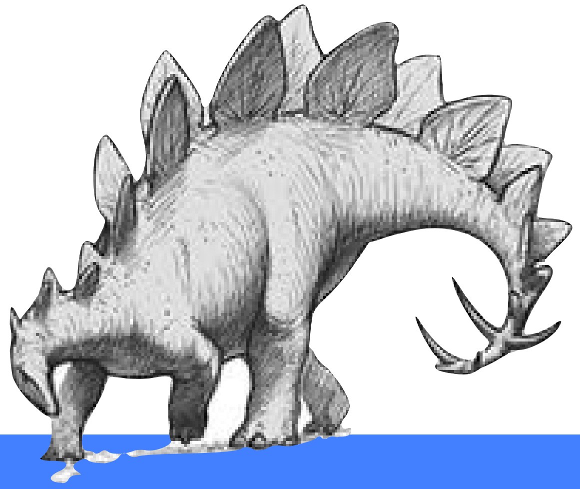
- PFF称作页面故障(频)率，基于这个数据可以实施一个防止抖动的策略：动态调节分配给进程的页框数量。



CONCLUDING REMARKS

- Practically speaking, thrashing and the resulting swapping have a disagreeably large impact on performance.
- The current best practice in implementing a computer facility is to include enough physical memory, whenever possible, to avoid thrashing and swapping.
- From smartphones through mainframes, providing enough memory to keep all working sets in memory concurrently, except under extreme conditions, gives the best user experience.

Lecture 15



The End

下期预告

- 下次直播时间：3月31日 上午9:30
- 课程内容
  - Lecture 16 Mass Storage
- Q&A