超越物理内存 - 交换策略
（Swapping Policies）

超越物理内存：策略

- 内存压力迫使操作系统开始**分页**，以腾出空间给活跃使用的页面

- 决定要**逐出**的页面由操作系统的替换策略来封装

缓存管理

- 目标是选择一个缓存替换策略，以**最小化缓存未命中（cache miss）的次数**
- 缓存命中（hit）和未命中（miss）的次数让我们计算**平均内存访问时间（AMAT）**

$$AMAT = (P_{Hit} * T_M) + (P_{Miss} * T_D)$$

Arguement	Meaning
T_M	The cost of accessing memory
T_D	The cost of accessing disk
P_{Hit}	The probability of finding the data item in the cache(a hit)
P_{Miss}	The probability of not finding the data in the cache(a miss)

最优替换策略

- 导致最少的整体未命中
  - 替换未来最远将被访问的页面
  - 导致**最少的可能**缓存未命中
- 仅作为比较点，了解我们离**完美**有多近

追踪最优策略

Reference Row

0 1 2 0 1 3 0 3 1 2 1

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

Hit rate is $\frac{Hits}{Hits+Misses} = \mathbf{54.6\%}$

然而，未来的访问是
不可知的

一个简单的策略：FIFO

•页面在进入系统时被放入队列

•当发生替换时，队列尾部的页面（“**最先进入**”的页面）被移除

- 它实现简单，但无法确定块的重要性

追踪FIFO

Reference Row

0 1 2 0 1 3 0 3 1 2 1

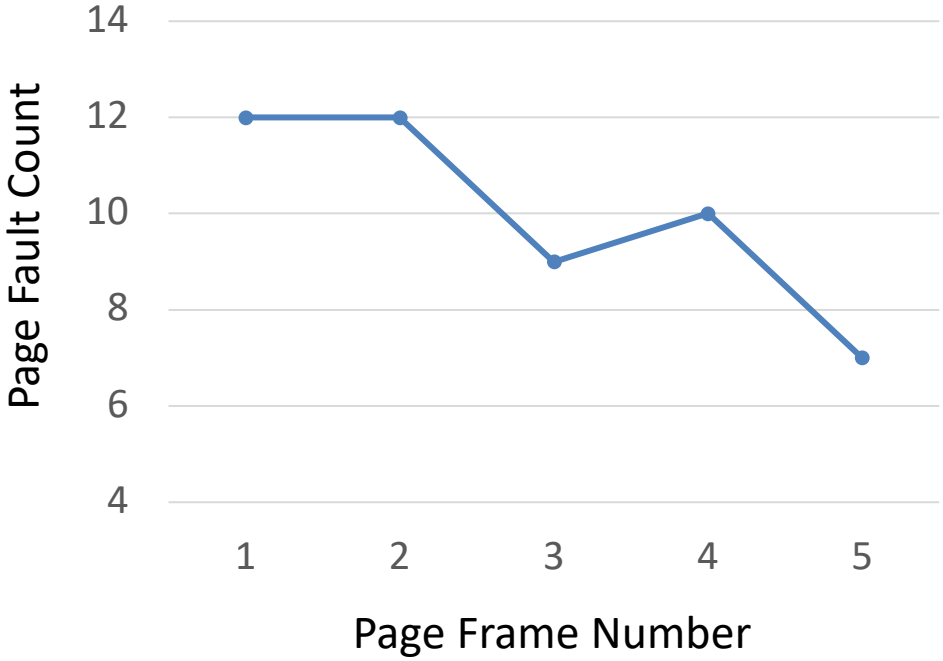
Access	Hit/Miss?	Evict	Resulting Cache State
0	Miss		0
1	Miss		0,1
2	Miss		0,1,2
0	Hit		0,1,2
1	Hit		0,1,2
3	Miss	0	1,2,3
0	Miss	1	2,3,0
3	Hit		2,3,0
1	Miss	2	3,0,1
2	Miss	3	0,1,2
1	Hit		0,1,2

Hit rate is $\frac{Hits}{Hits+Misses} = 36.4\%$

即使页面0被多次访问（重要），FIFO仍然会将其驱逐

Belady异常

- 我们预期缓存命中率会**随着缓存变大而增加**，从而减少页错误。但在有些例子下，使用FIFO（先进先出），情况反而变差。



Reference Row											
1	2	3	4	1	2	5	1	2	3	4	5

另一种简单策略：随机

- 随机选择一个页面进行替换以缓解内存压力
  - 在选择要驱逐的块时并不试图过于智能
  - 依赖运气来做出选择

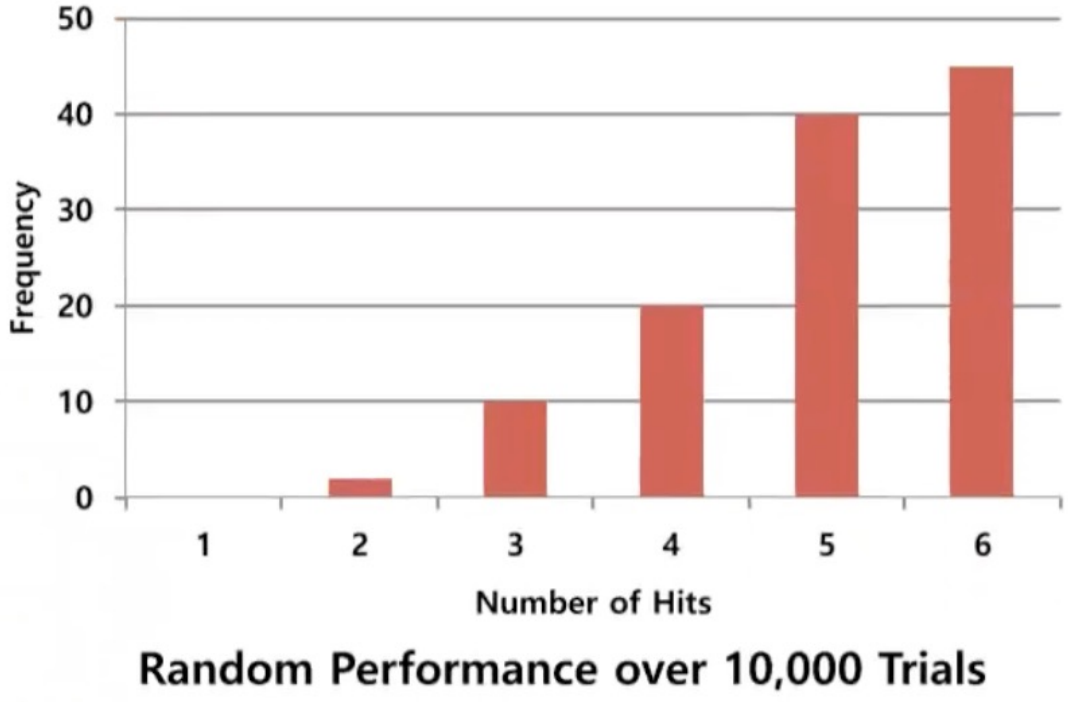
Access	Hit/Miss?	Evict	Resulting Cache State
0	Miss		0
1	Miss		0,1
2	Miss		0,1,2
0	Hit		0,1,2
1	Hit		0,1,2
3	Miss	0	1,2,3
0	Miss	1	2,3,0
3	Hit		2,3,0
1	Miss	3	2,0,1
2	Hit		2,0,1
1	Hit		2,0,1

Reference Row

0 1 2 0 1 3 0 3 1 2 1

随机性能

- 有时候，随机方法和最优方法一样好，在示例轨迹中可以达到6次命中。



使用历史

- 依赖过去并使用历史
  - 两种类型的历史信息

历史信息	含义	算法
最近性 (recency)	越是最近访问过的页面，越有可能再次被访问	LRU
频率 (frequency)	如果一个页面被访问了很多次，它不应该被轻易替换，因为它显然有一定的价值	LFU

使用历史：LRU

- 替换最近最少使用的页面

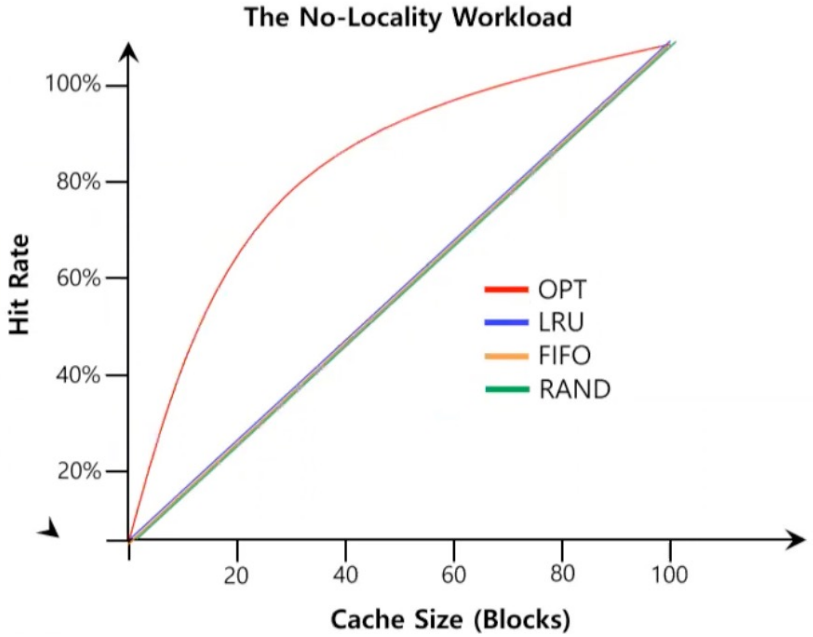
Reference Row

0 1 2 0 1 3 0 3 1 2 1

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

工作负载示例：无局部性工作负载

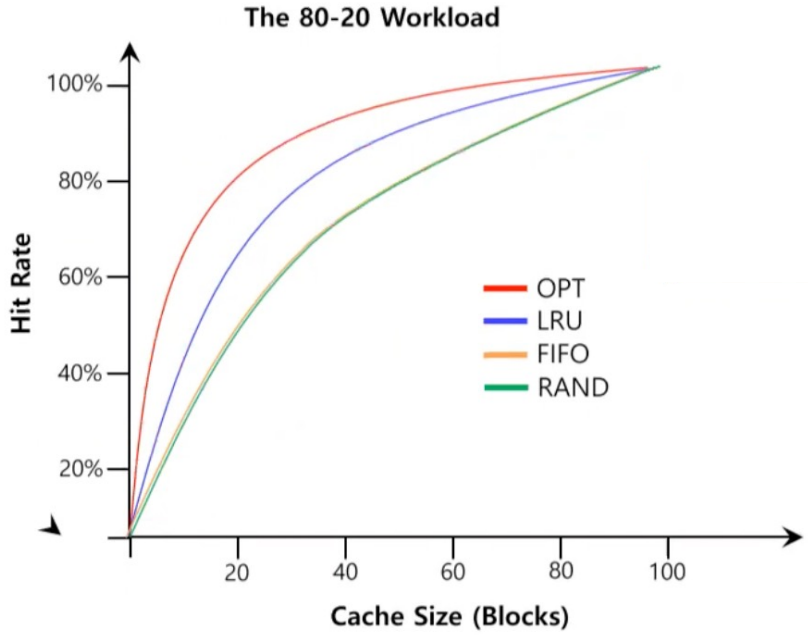
- 每次引用是访问页面集中的一个随机页面。
  - 工作负载随时间访问100个唯一页面。
  - **无局部性**：选择下一个要引用的页面是随机的。



注释： 当缓存足够大以容纳整个工作负载时，使用的策略就不再重要。

工作负载示例：80-20工作负载

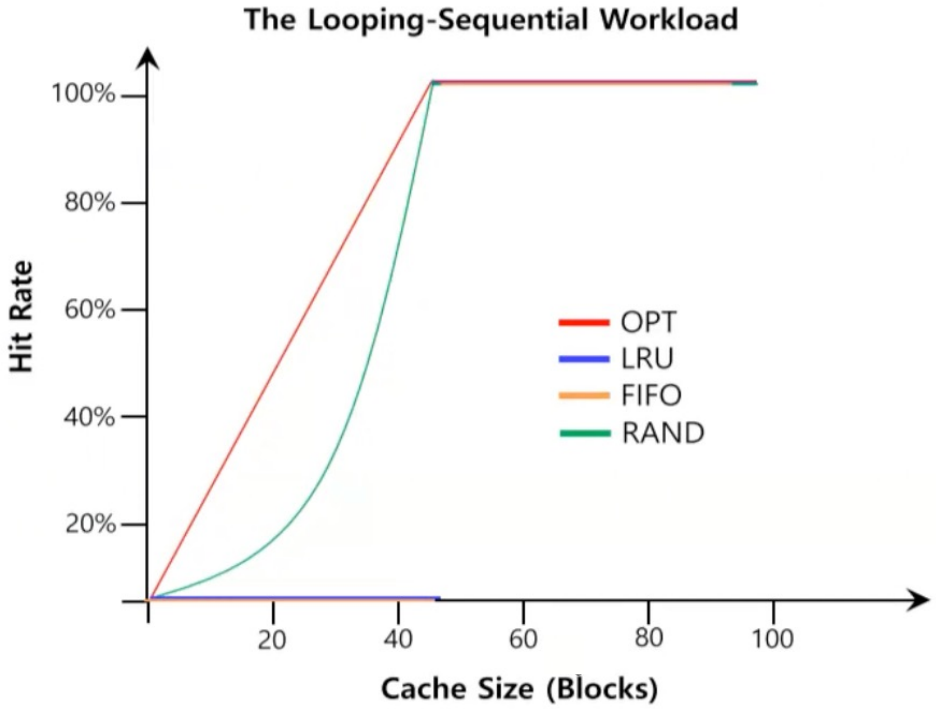
- **表现出局部性：**80%的引用指向20%的页面（热门页面）。
- 剩余20%的引用指向剩余80%的页面（冷门页面）。



LRU更有可能保留热门页面

工作负载示例：循环序列

- 引用50个页面：从0开始，然后是1，……，一直到49，然后循环，重复这些访问



实现历史算法

•在LRU中，为了跟踪哪些页面是最近最少使用的，系统必须在每次内存引用时进行一些记录工作

•添加一点硬件支持，加速性能（例如，每个页面维护一个时间字段/计数器）

•然而，随着页面数量的增长，扫描所有页的时间字段的代价较大

由于实现完美的LRU 代价非常昂贵，我们能否实现一个近似的LRU 算法

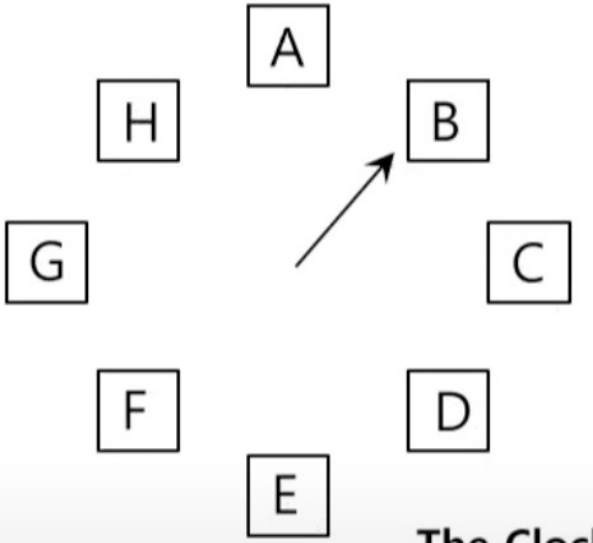
近似LRU

- **需要一些硬件支持，以使用位的形式**
  - 每当**一个页面被引用**时，使用位由硬件置为1
  - 硬件**从不**清除该位，这由操作系统负责

- **时钟算法：***利用使用位来实现近似LRU*
  - 系统的所有页面排列成一个循环列表
  - 一个时钟指针指向某个特定页面开始

时钟算法

- 算法会持续运行，直到找到一个使用位（Use bit）为0的页面

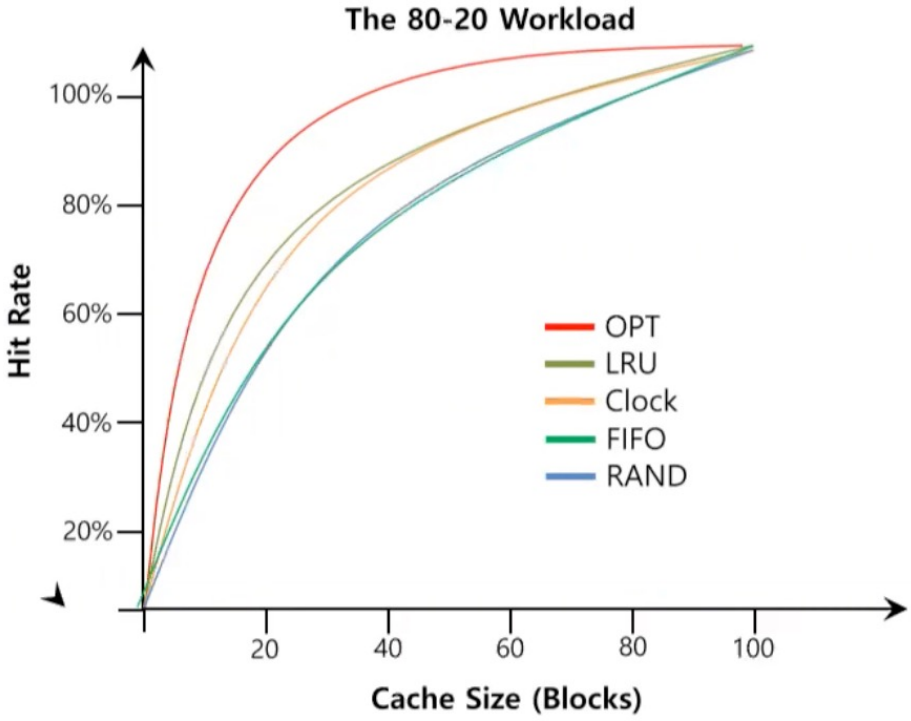


Use bit	Meaning
0	Evict the page
1	Clear **Use bit** and advance hand

The Clock page replacement algorithm

使用时钟算法处理工作负载

- 时钟算法的表现不如完美的LRU，但比完全不考虑历史的策略要好。



考虑脏页

•硬件包含一个**修改位**（也称为脏位）

- 页面已被修改，因此是脏的，必须写回磁盘，再将其逐出
- 页面未被修改，不需要写回磁盘，可以直接逐出
- 时钟算法可以优先逐出未使用的干净页；否则查找未使用的脏页

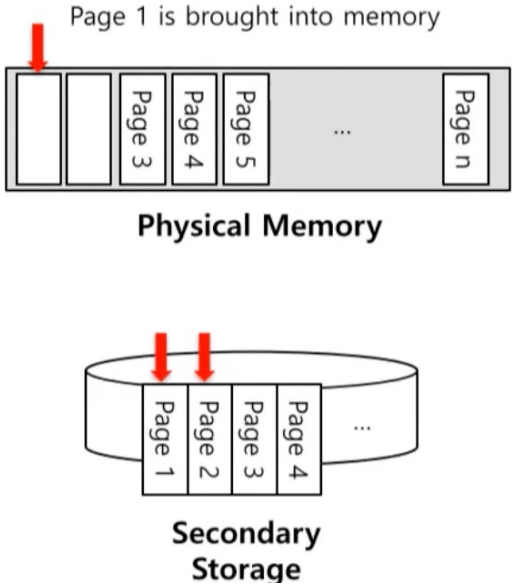
页选择策略

- 操作系统需要决定何时将页面加载到内存中

- 为操作系统提供了一些**不同的选项**

策略1: 预取（Prefetching）

- 操作系统会猜测某个页面即将被使用，并提前将其加载到内存中。

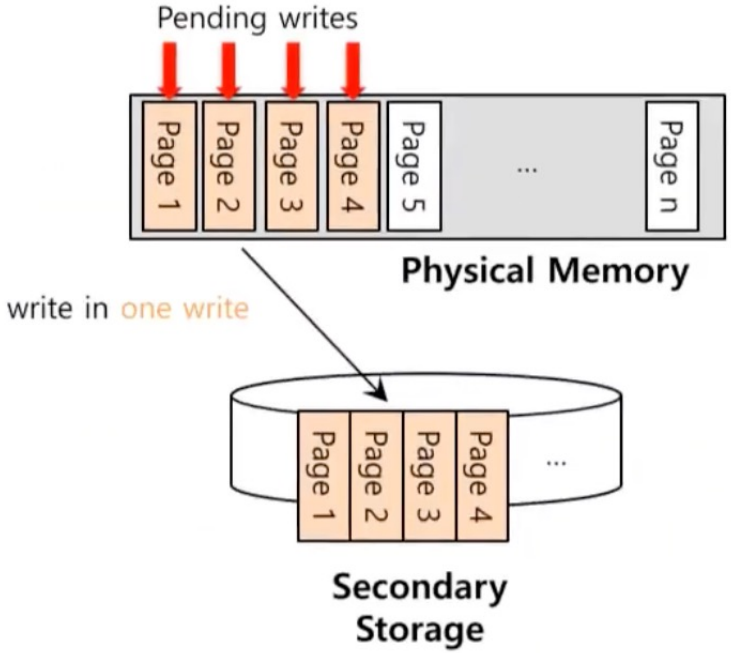


页面2很可能会很快被访问，因此也应该被加载到内存中

策略2: 聚类，分组

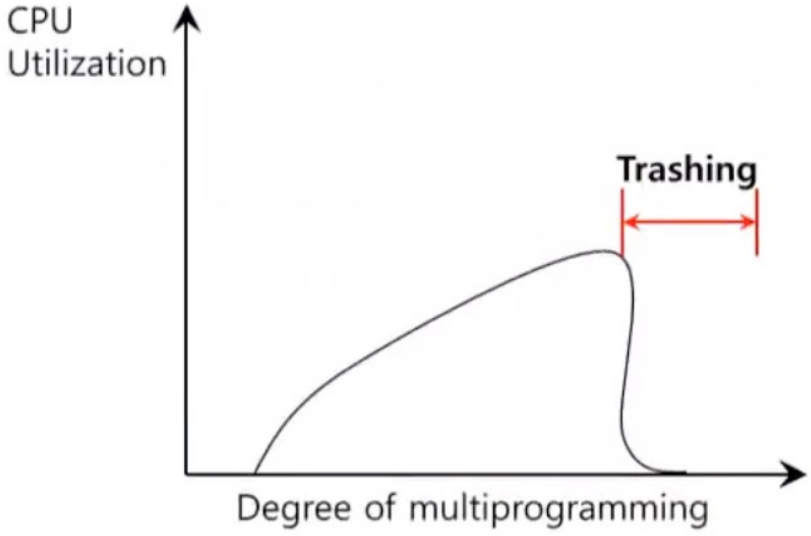
- 收集多个**待写入的写操作**，在内存中将它们聚在一起，然后一次性写入磁盘

  •进行**一次大的写操作**比**多次小的写操作**更有效率。



抖动（Thrashing）

- 内存被**过度订阅**，运行进程的内存需求**超过了可用物理内存**
  - 准入控制：决定不运行一部分进程的子集。
  - Out-of-memory killer: 减少内存中进程的工作集。



小结

- 我们介绍了页面替换的一些策略
  - 最优替换、FIFO替换、随机替换、LRU替换
- 我们介绍了三种工作负载情况下，上述页面替换策略的表现对比
  - 无局部性工作负载、80-20工作负载、循环序列
- 我们介绍了一种近似LRU方法：时钟算法
- 我们介绍了页选择策略（预取、聚类）、抖动等内容

内存虚拟化内容总结

- 地址空间
- 内存分配和释放
- 地址转换
- 基址-界限寄存器方法、分段、分页
- 快速地址转换：**TLB**
- 减少页表大小：混合、多级页表
- 内存和磁盘间交换的机制和策略