Engineering &
Computer Science
Syracuse University

CIS 657 – Principles of Operating Systems

Topic: Memory – Swapping

Endadul Hoque

Acknowledgement

- Youjip Won (Hanyang University)

- OSTEP book – by Remzi and Andrea Arpaci-Dusseau (University of Wisconsin)

BEYOND PHYSICAL MEMORY: MECHANISMS

Revisit: Assumptions

- An address space is small and fits into physical memory
- The address space of each running process ALL fit into physical memory
- All pages reside in physical memory

Revisit: Assumptions

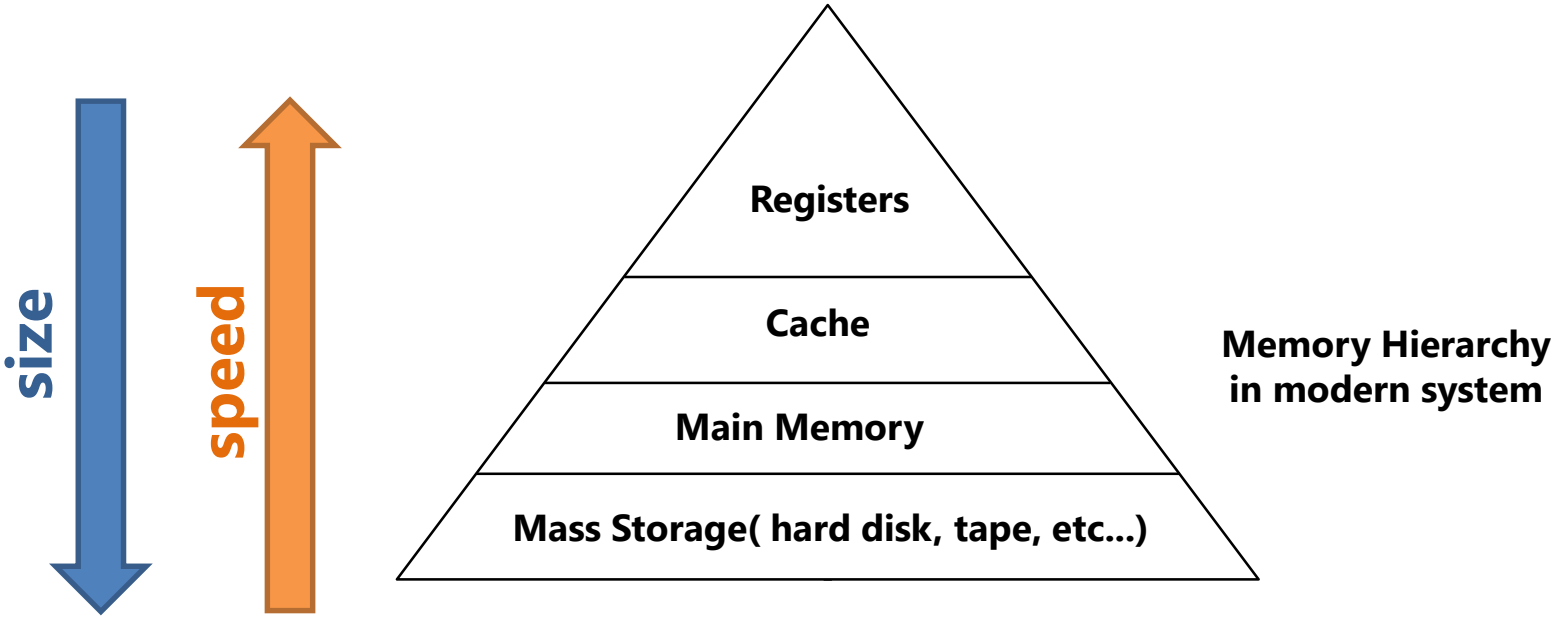
- ~~An address space is small and fits into physical memory~~
- ~~The address space of each running process ALL fit into physical memory~~
- ~~All pages reside in physical memory~~

Focus:

- How to support **many** concurrently-running **large** address spaces?
- How to leverage **a larger (slower) device** to transparently provide the illusion of a large virtual address space?

Memory Hierarchy

- Require an additional level in the memory hierarchy.
- OS needs a place to stash away portions of address space that currently aren't in great demand.
- Such a location should have more capacity, but it is generally slower
- In modern systems, this role is usually served by a hard disk drive



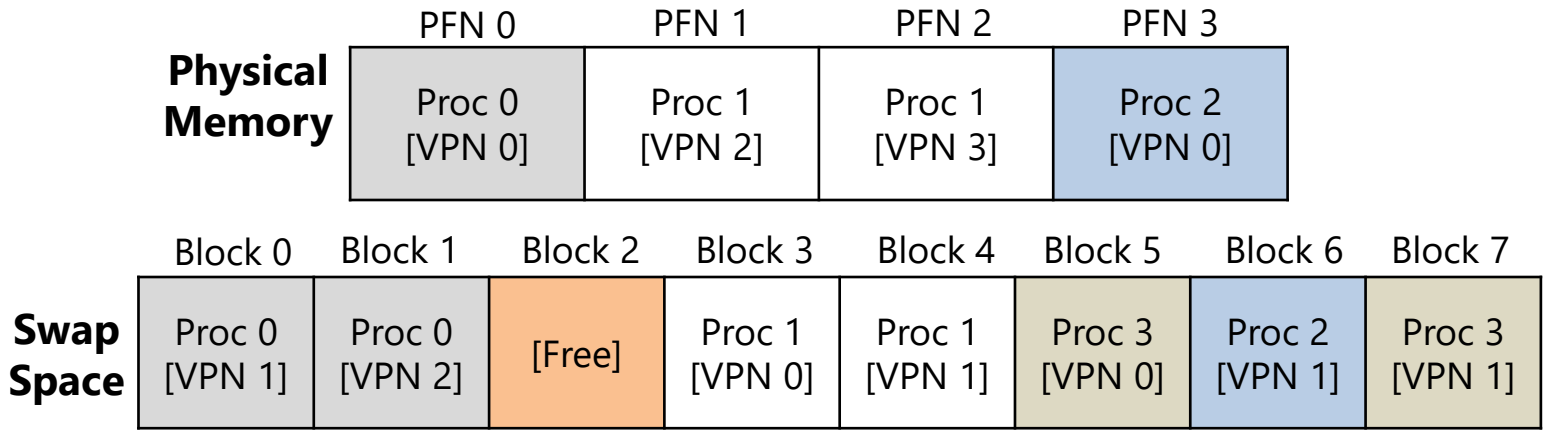
Single large address-space for a process

- Programming convenience and ease-of-use
  - Otherwise, we always need to first arrange for the code or data to be in memory before calling a function or accessing data.

- To Beyond just a single process.
  - The addition of **swap space** allows the OS to support the illusion of a large virtual memory for multiple concurrently-running processes

Swap Space

- Reserve some space on the disk for moving pages back and forth.
  - **Swap pages out** of memory to it and **Swap pages into** memory from it
- OS reads from and writes to the swap space, in **page-sized unit**
  - Need to **remember the disk address** of a given page



Physical Memory and Swap Space

Present Bit

- Add some machinery higher up in the system in order to support swapping pages to and from the disk.
  - When the hardware looks in the PTE, it may find that the page is not present in physical memory.

Value	Meaning
1	page is present in physical memory
0	The page is not in memory but rather on disk.

The Page Fault

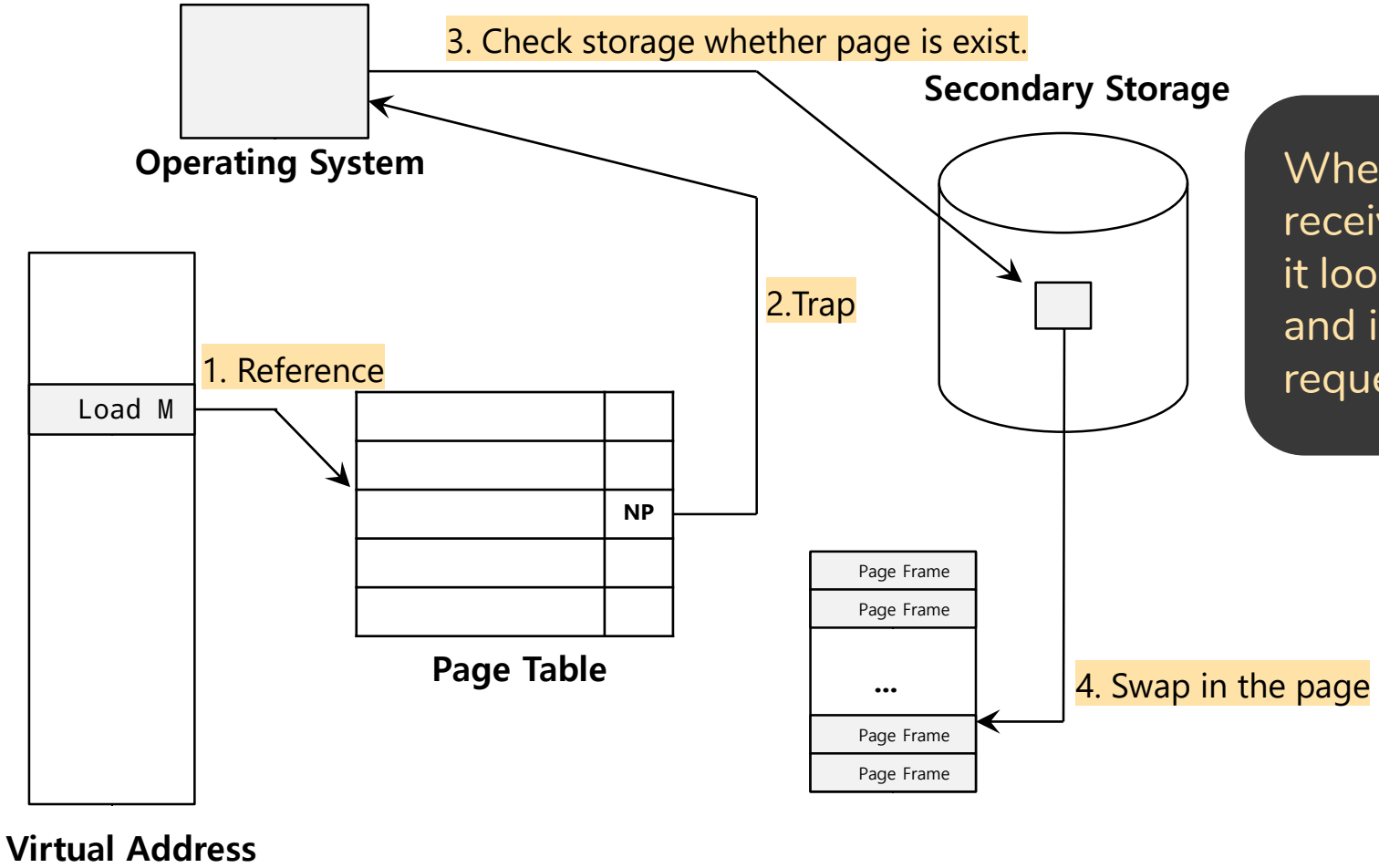
- Accessing page that is not in physical memory.
  - If a page is not present and has been swapped to disk, the OS needs to **swap the page into memory** in order to service the page fault.
    - Using a **page-fault handler**

What if (physical) memory is full ?

- The OS like to swap out pages to **make room** for the new pages the OS is about to bring in.
  - The process of picking a page to kick out, or replace is known as page-replacement policy

Page Fault Control Flow

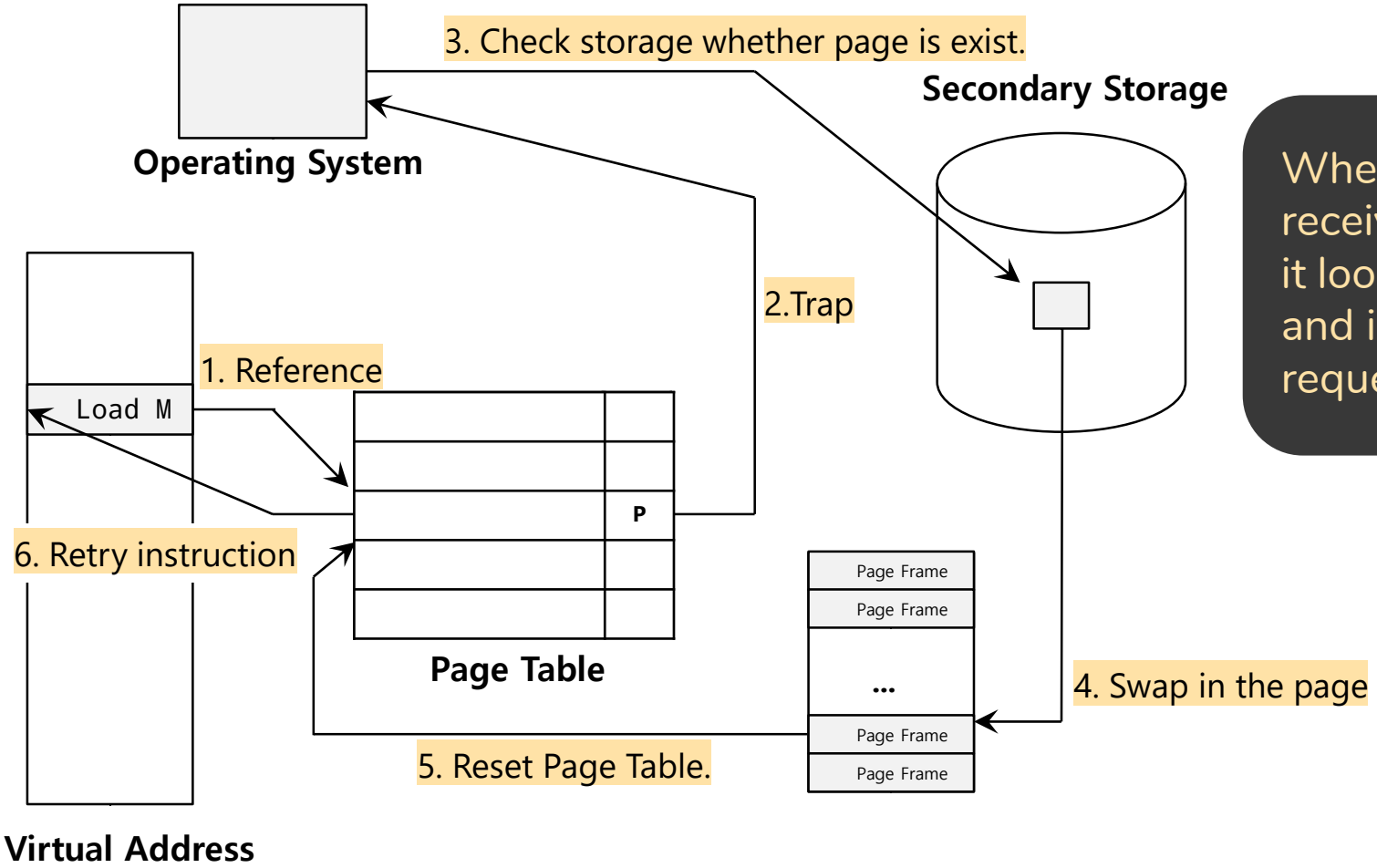
- PTE's PFN is used to hold the disk address of the page in swap space.



When the OS receives a page fault, it looks in the PTE and issues the request to disk.

Page Fault Control Flow

- PTE's PFN is used to hold the disk address of the page in swap space.



When the OS receives a page fault, it looks in the PTE and issues the request to disk.

Page Fault Control Flow – Hardware

```
1:      VPN = (VirtualAddress & VPN_MASK) >> SHIFT
2:      (Success, TlbEntry) = TLB_Lookup(VPN)
3:      if (Success == True) // TLB Hit
4:      if (CanAccess(TlbEntry.ProtectBits) == True)
5:              Offset = VirtualAddress & OFFSET_MASK
6:              PhysAddr = (TlbEntry.PFN << SHIFT) | Offset
7:              Register = AccessMemory(PhysAddr)
8:      else RaiseException(PROTECTION_FAULT)
```

Page Fault Control Flow – Hardware

```
9:      else // TLB Miss
10:     PTEAddr = PTBR + (VPN * sizeof(PTE))
11:     PTE = AccessMemory(PTEAddr)
12:     if (PTE.Valid == False)
13:             RaiseException(SEGMENTATION_FAULT)
14:     else
15:     if (CanAccess(PTE.ProtectBits) == False)
16:             RaiseException(PROTECTION_FAULT)
17:     else if (PTE.Present == True)
18:     // assuming hardware-managed TLB
19:             TLB_Insert(VPN, PTE.PFN, PTE.ProtectBits)
20:             RetryInstruction()
21:     else if (PTE.Present == False)
22:             RaiseException(PAGE_FAULT)
```

Page Fault Control Flow – Software

```
1:        PFN = FindFreePhysicalPage()
2:        if (PFN == -1) // no free page found
3:                PFN = EvictPage() // run replacement algorithm
4:        DiskRead(PTE.DiskAddr, PFN) // sleep (waiting for I/O)
5:        PTE.present = True // update page table with present
6:        PTE.PFN = PFN // bit and translation (PFN)
7:        RetryInstruction() // retry instruction
```

- The OS must find a free physical frame for the soon-be-swapped-in page.
- If there is no such page, waiting for the replacement algorithm to run and kick some pages out of memory.

When Replacements Really Occur

- OS **waits** until memory is entirely full, and only then replaces a page to make room for some other page
  - This is a little bit unrealistic, and there are many reason for the OS to keep a small portion of memory free more **proactively**.

- Swap Daemon, Page Daemon
  - There are fewer than LW pages available, a background thread that is responsible for freeing memory (page frames) runs.
  - The thread evicts pages until there are HW pages available.

Swapping: policies

Beyond Physical Memory: Policies

- Memory pressure forces the OS to start paging out pages to make room for actively-used pages.
- Deciding which page to **evict** is encapsulated within the replacement policy of the OS.

In this discussion, physical memory can "technically" be viewed as a **cache** for virtual pages

Cache Management

- Goal in picking a replacement policy for this cache is to **minimize** the number of cache misses.
- The number of cache hits and misses let us calculate the *average memory access time* (***AMAT***).

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

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

The Optimal Replacement Policy

- Leads to the fewest number of misses overall
  - Replaces the page that will be accessed furthest in the future
  - Resulting in the fewest-possible cache misses
- Serve only as a comparison point, to know how close we are to perfect

Tracing the Optimal Policy

Reference Row

0 1 2 0 1 3 0 3 1 2 1

Physical memory holds **3** frames

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

Future is not known.

A Simple Policy: FIFO

- Pages were placed in a queue when they enter the system.
- When a replacement occurs, the page at the tail of the queue (the "**First-in**" pages) is evicted.
  - It is simple to implement, but can't determine the importance of blocks.

Tracing the FIFO Policy

Reference Row

0 1 2 0 1 3 0 3 1 2 1

Physical memory holds **3** frames

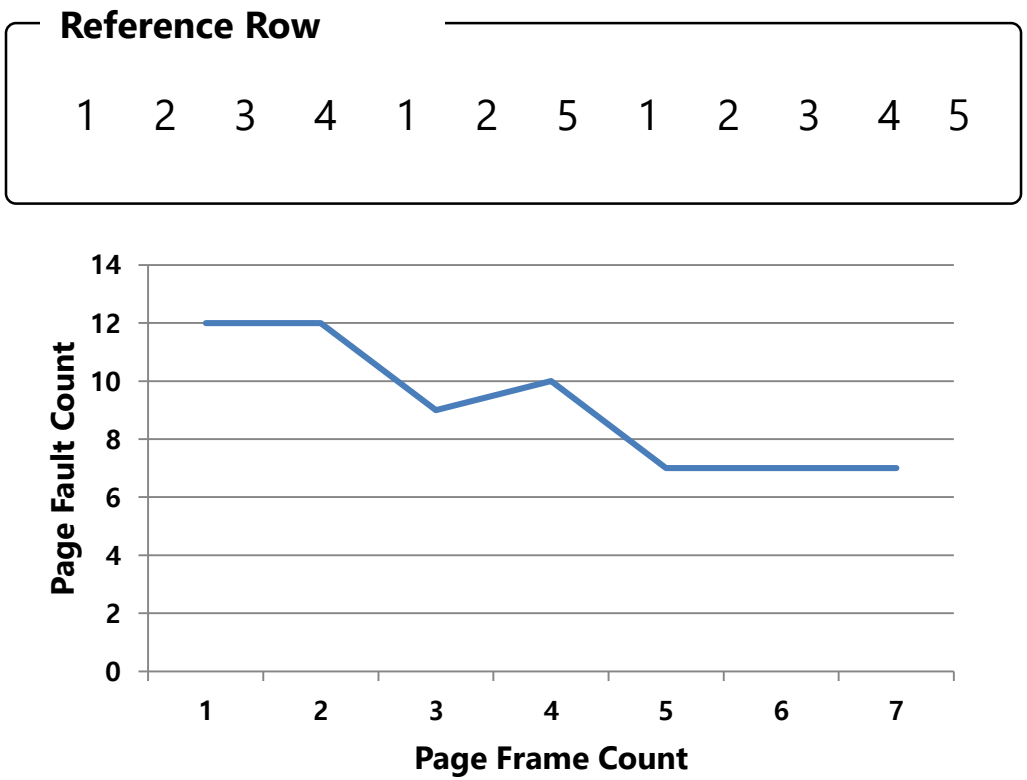
Access	Hit/Miss?	Evict	Resulting Cache State
0	Miss		0
1	Miss		0,1
2	Miss		0,1,2
0	Hit		0,1,2
1	Hit		0,1,2
3	Miss	0	1,2,3
0	Miss	1	2,3,0
3	Hit		2,3,0
1	Miss		3,0,1
2	Miss	3	0,1,2
1	Hit		0,1,2

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

Even though page 0 had been accessed a number of times, FIFO still kicks it out.

BELADY'S Anomaly

- We would expect the cache hit rate to increase when the cache gets larger. But in this case, with FIFO, it gets worse.



Another Simple Policy: Random

- Picks a random page to replace under memory pressure.
  - It doesn't really try to be too intelligent in picking which blocks to evict.
  - Random does depends entirely upon how lucky Random gets in its choice.

Physical memory holds **3** frames

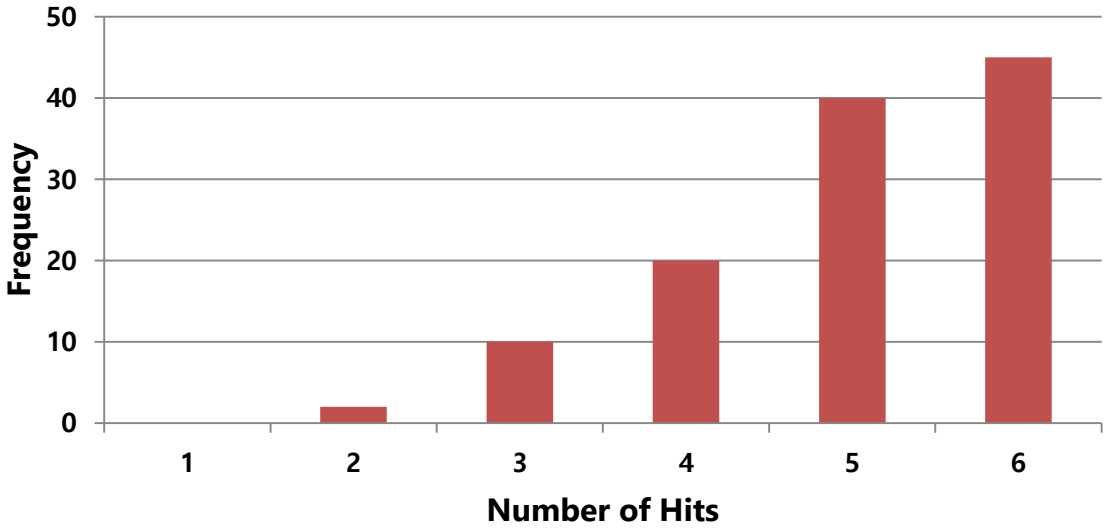
Reference Row

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	3	2,0,1
2	Hit		2,0,1
1	Hit		2,0,1

Performance of Random Policy

- Sometimes, Random is as good as optimal, achieving 6 hits on the example trace.



Random Performance over 10,000 Trials

Using History

- Lean on the past and use **history**.
  - Two type of historical information.

Historical Information	Meaning	Algorithms
recency	The **more recently** a page has been accessed, the more likely it will be accessed again	LRU
frequency	If a page has been **accessed many times**, it should not be replaced as it clearly has some value	LFU

Using History : LRU

- Replaces the least-recently-used page.

Reference Row

0 1 2 0 1 3 0 3 1 2 1

Physical memory holds **3** frames

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

Implementing Historical Algorithms

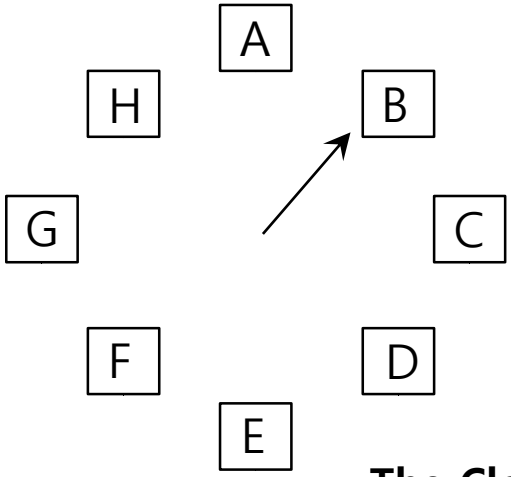
- To keep track of which pages have been least-and-recently used, the system has to do some accounting work on **every memory reference.**
  - Add a little bit of hardware support.

Approximating LRU

- Require some hardware support, in the form of a **use bit**
  - Whenever a page is referenced, the use bit is set by hardware to 1.
  - Hardware never clears the bit, though; this is the responsibility of the OS
- Clock Algorithm
  - All pages of the system arranges in a circular list.
  - A clock hand points to some particular page to begin with.

Clock Algorithm

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



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

The Clock page replacement algorithm

- The algorithm continues until it finds a use bit that is set to 0.

Considering Dirty Pages

- The hardware include a **modified bit** (a.k.a **dirty bit**)
  - Page has been **modified** and is thus **dirty**, it must be written back to disk to evict it.
  - Page has not been modified, the eviction is free.

Reading Material

- **Chapter 21-22** of OSTEP book – by Remzi and Andrea Arpaci-Dusseau (University of Wisconsin)
  http://pages.cs.wisc.edu/~remzi/OSTEP/vm-beyondphys.pdf
  http://pages.cs.wisc.edu/~remzi/OSTEP/vm-beyondphys-policy.pdf

Questions?