

CS370 Operating Systems

Colorado State University

Yashwant K Malaiya

Spring 2022 L20

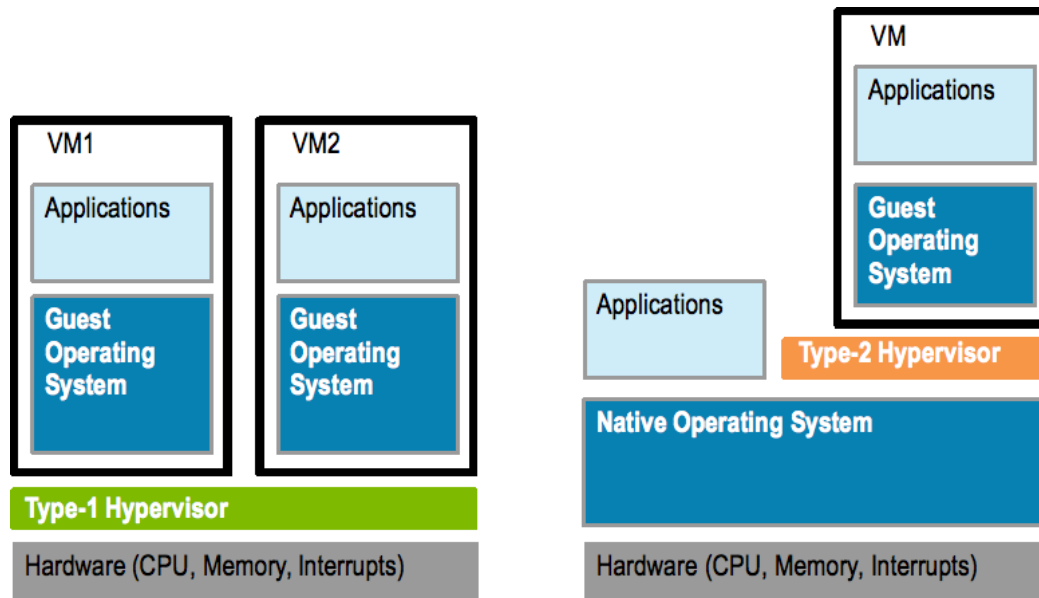
Containers, Virtual Memory



Slides based on

- Text by Silberschatz, Galvin, Gagne
- Various sources

Implementation of VMMs



A higher layer uses services of the lower layers.

Type 1: (ex VMWare esx) Low overhead, choice for data centers

- Run in kernel mode, Implement device drivers, provide traditional OS services

Type2: (ex Virtualbox) Individuals, small organizations

- VMM is simply a process, managed by host OS

Q: Do we really always need multiple copies of the OS?

Full vs Para-virtualization

- Full virtualization: Guest OS is unaware of the hypervisor. It thinks it is running on bare metal.
- Para-virtualization: Guest OS is modified and optimized. It sees underlying hypervisor.
 - Introduced and developed by Xen
 - Modifications needed: Linux 1.36%, XP: 0.04% of code base
 - Does not need as much hardware support
 - allowed virtualization of older x86 CPUs without binary translation
 - Not used by Xen on newer processors

D2 Submission

- Research and Development Canvas groups will be created in a couple of days by us.
 - All members of a team must join an applicable type of Canvas Groups.
- Only one team member (leader) will submit.
Submission will be in the team leaders' section.
 - Grade will automatically apply to all members, if they are in the same section.
 - Please give section numbers of all members in a multi-section team.

CPU Scheduling

- One or more virtual CPUs (vCPUs) per guest
 - Can be adjusted throughout life of VM
- When enough CPUs for all guests
 - VMM can allocate dedicated CPUs, each guest much like native operating system managing its CPUs
- Usually not enough CPUs (CPU overcommitment)
 - VMM can use scheduling algorithms to allocate vCPUs
 - Some add fairness aspect

CPU Scheduling (cont)

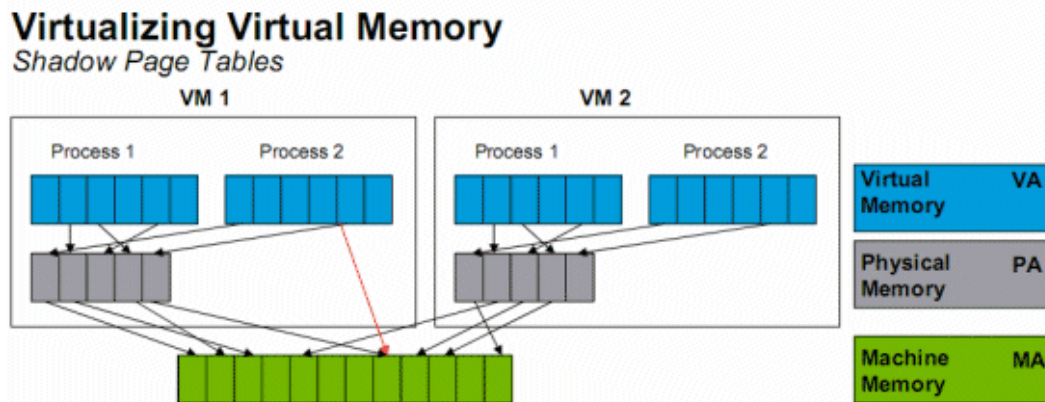
- Oversubscription of CPUs means guests may get CPU cycles they expect
 - Time-of-day clocks may be incorrect
 - Some VMMs provide application to run in each guest to fix time-of-day

Memory Management

Memory mapping:

- On a bare metal machine: OS uses page table/TLB to map Virtual page number (VPN) to Physical page number (PPN) (physical memory is shared). Each process has its own page table/TLB.
 - VPN -> PPN
- VMM: Real physical memory (*machine memory*) is shared by the OSs. Need to map PPN of each VM to MPN (Shadow page table)

PPN -> MPN



Memory Management

- VMM: Real physical memory (*machine memory*) is shared by the OSs. Need to map PPN of each VM to MPN (Shadow page table)

PPN ->MPN

- Where is this done?
 - Has to be done by hypervisor type 1. Guest OS knows nothing about MPN.
 - Page Table/TLB updates are trapped to VMM.
It needs to do VPN->PPN ->MPN.
 - It can do VPN->MPN directly (VMware ESX)

Handling memory oversubscription

Oversubscription solutions:

- *Deduplication* by VMM determining if same page loaded more than once, memory mapping the same page into multiple guests
- Double-paging, the guest page table indicates a page is in a physical frame but the VMM moves some of those to disk.
- Install a **pseudo-device driver** in each guest (it looks like a device driver to the guest kernel but really just adds kernel-mode code to the guest)
 - **Balloon** memory manager communicates with VMM and is told to allocate or deallocate memory to decrease or increase physical memory use of guest, causing guest OS to free or have more memory available.

Virtual Machine (VM) as a set of files

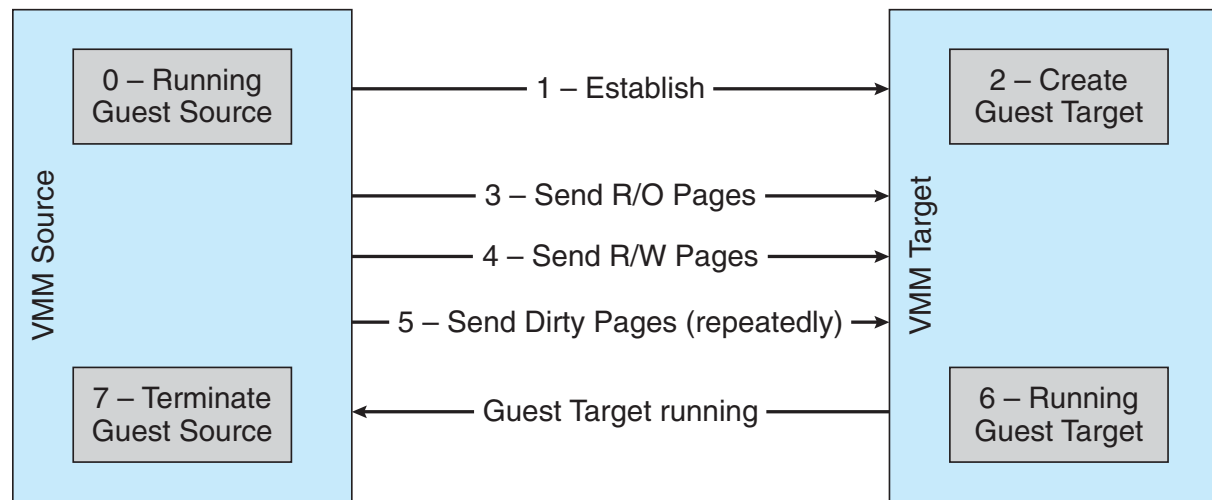
- Configuration file describes the attributes of the virtual machine containing
 - server definition,
 - how many virtual processors (vCPUs)
 - how much RAM is allocated,
 - which I/O devices the VM has access to,
 - how many network interface cards (NICs) are in the virtual server
 - the storage that the VM can access
- When a virtual machine is instantiated, additional files are created for logging, for memory paging etc.
- Copying a VM produces not only a backup of the data but also a copy of the entire server, including the operating system, applications, and the hardware configuration itself

Live Migration

Running guest can be moved between systems, without interrupting user access to the guest or its apps

- for resource management,
- maintenance downtime windows, etc
- Migration from source VMM to target VMM
 - Needs to migrate all pages gradually, without interrupting execution (details in next slide)
 - Eventually source VMM freezes guest, sends vCPU's final state, sends other state details, and tells target to start running the guest
 - Once target acknowledges that guest running, source terminates guest

Live Migration



- Migration from source VMM to target VMM
 - Source establishes a connection with the target
 - Target creates a new guest
 - Source sends all read-only memory pages to target
 - Source starts sending all read-write pages
 - Source VMM freezes guest, sends final stuff,
 - Once target acknowledge that guest running, source terminates guest.

VIRTUAL APPLIANCES: “shrink-wrapped” virtual machines

- Developer can construct a virtual machine with
 - required OS, compiler, libraries, and application code
 - Freeze them as a unit ... ready to run
- Customers get a complete working package
- Virtual appliances: “shrink-wrapped” virtual machines
- Amazon’s EC2 cloud offers many pre-packaged virtual appliances examples of *Software as a service*
- *Question: do we really have to include a whole kernel in a shrink wrapped VM?*

CS370 Operating Systems

Colorado State University

Yashwant K Malaiya

Spring 2022



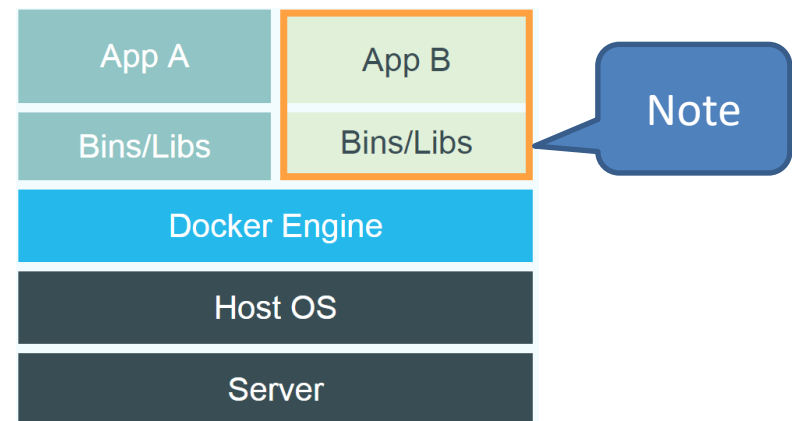
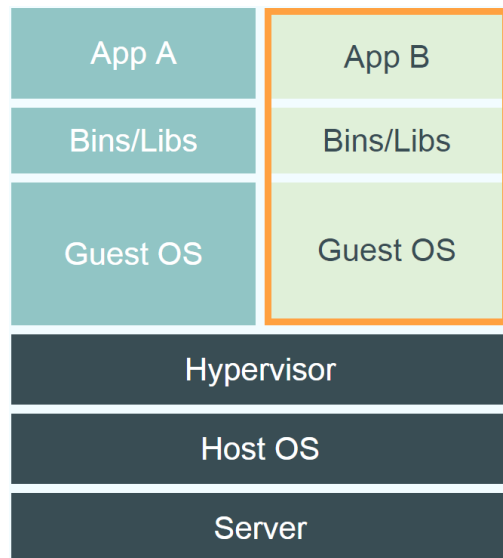
Containers

Slides based on

- Various sources

Linux Containers and Docker

- Linux containers (LXC 2008) are “lightweight” VMs
- Comparison between LXC/docker (2013) and VM



- Containers provide “OS-level Virtualization” vs “hardware level”.
- Containers can be deployed in seconds.
- Very little overhead during execution, even better than Type 1 VMM.

VMs vs Containers

VMs	Containers (“virtual environment”)
Heavyweight several GB	Lightweight tens of MB
Limited performance	Native performance
Each VM runs in its own OS	All containers share the host OS
<i>Hardware-level virtualization</i>	<i>OS virtualization</i>
Startup time in minutes	Startup time in milliseconds
Allocates required memory	Requires less memory space
Fully isolated and hence more secure	Process-level isolation, possibly less secure

Container: basis

Linux kernel provides

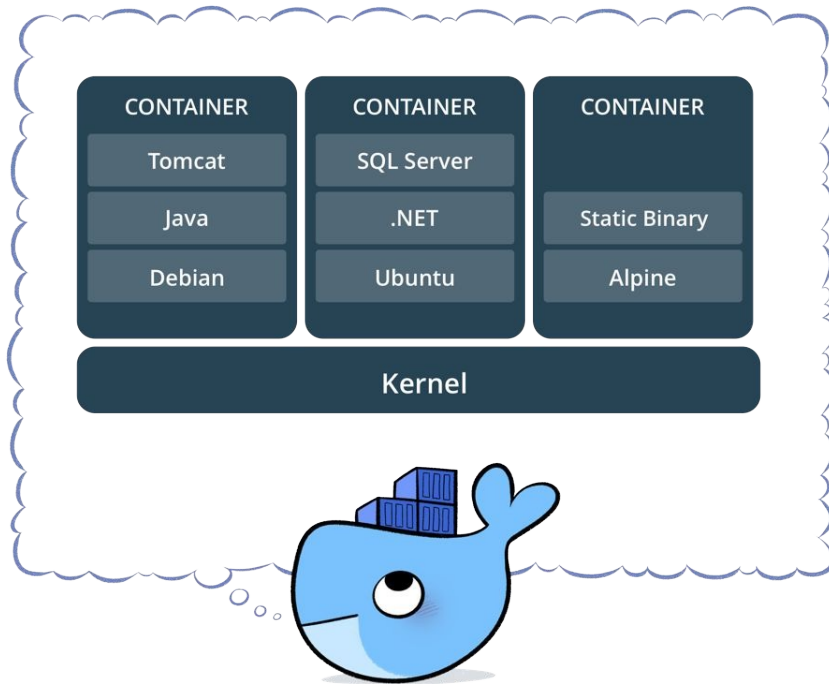
- “control groups” (cgroups) functionality for a set of processes
 - allows allocation and prioritization of resources (CPU, memory, block I/O, network, etc.) without the need for starting any VM
- “namespace isolation” functionality
 - allows complete isolation of an applications' view of the operating environment including Process trees, networking, user IDs and mounted file systems.
- Managed by
 - Docker (or competitors): build, share, run containerized apps.
 - Kubernetes (or competitors): orchestration platform for managing, automating, and scaling containerized applications

Docker – podman/buildah

Docker swarm – Kubernetes, OPENSIFT

Container

What is a container?



- Standardized packaging for software and dependencies
- Isolate apps from each other
- Share the same OS kernel
- Works for all major Linux distributions
- Docker Desktop for Windows uses Windows-native Hyper-V virtualization (Win10)
- Containers native to Windows Server 2016
- Docker: a popular container management service technology.

Alternatives: Podman etc

Some Docker vocabulary

- **Docker Image**
 - The basis of a Docker container. Represents a full application
- **Docker Container**
 - The standard unit in which the application service resides and executes
- **Docker Engine**
 - Creates, ships and runs Docker containers deployable on a physical or virtual, host locally, in a datacenter or cloud service provider
- **Registry Service (Docker Hub(Public) or Docker Trusted Registry(Private))**
 - Cloud or server based storage and distribution service for images (can be **pulled** or **pushed**)
- **Dockerfile** is a text document that contains all the commands a user could call on the command line to assemble an image using **docker build** command.

19

Correspondence: executable:image container:process

Some Docker vocabulary: Analogies

Containers have their own jargon. Here are some analogous terms. Note that some analogies can be questionable.

	Docker	Non-containerized code
What is executed	Docker Image	executable
Isolation unit	Docker Container	process
to create what is executed	Dockerfile	makefile
	Docker engine	OS/JVM
	Registry Service	code repository

- Only a high-level look here. For details see documentation and videos.
- Help Session this Wed 5:30 PM.
- Several interrelated technologies. Significant experience needed to gain expertise.

Some Docker vocabulary

- **Dockerfile** is a text document that contains all the commands a user could call on the command line to assemble an image using **docker build** command.
- Ex:

```
# syntax=docker/dockerfile:1
FROM ubuntu:18.04
COPY . /app
RUN make /app
CMD python /app/app.py
```

Each instruction creates one layer:

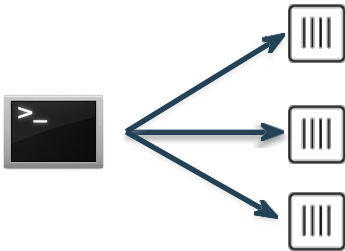
- FROM creates a layer from the ubuntu:18.04 Docker image.
- COPY adds files from your Docker client's current directory.
- RUN builds your application with make.
- CMD specifies what command to run within the container.

Docker Volumes

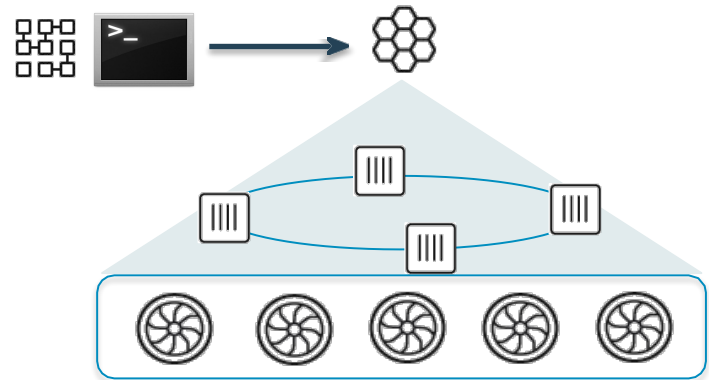
- Volumes mount a directory on the host into the container at a specific location
- Can be used to share (and persist) data between containers
 - Directory persists after the container is deleted
 - Unless you explicitly delete it
- Can be created in a Dockerfile or via CLI

Docker Compose: Multi Container Applications

- Build and run one container at a time
- Manually connect containers together
- Must be careful with dependencies and start up order



- Define multi container app in compose.yml file
- Single command to deploy entire app
- Handles container dependencies
- Works with Docker Swarm, Networking, Volumes, Universal Control Plane



Docker Compose: Multi Container Applications



`version: '2' # specify docker-compose version`

`# Define the services/containers to be run`
`services:`

`angular: # name of the first service`

`build: client # specify the directory of the Dockerfile`

`ports:`

`- "4200:4200" # specify port forwarding`

`express: #name of the second service`

`build: api # specify the directory of the Dockerfile`

`ports:`

`- "3977:3977" #specify ports forwarding`

`database: # name of the third service`

`image: mongo # specify image to build container from`

`ports:`

`- "27017:27017" # specify port forwarding`

Terms









- **Docker** technology used for containers and can deploy single, containerized applications.
- **Docker Compose** for configuring and starting multiple Docker containers on the same host.
- **Docker swarm** is a container orchestration tool that allows you to run and connect containers on multiple hosts.
- **Kubernetes** is a container orchestration tool that is similar to Docker swarm, but has ease of automation and ability to handle higher demand.

Some Docker Commands

- **docker — — version** get the currently installed version of docker
- **docker build <path to docker file>** build an image from a specified docker file
- **docker login** login to the docker hub repository
- **docker pull <image name>** pull images from the **docker repository** hub.docker.com
- **docker push <username/image name>**
- **docker run -it -d <image name>** create a container from an image
- **docker stop <container id>** stops a running container
- **docker kill <container id>** kills the container by stopping its execution immediately
- **docker rm <container id>** delete a stopped container
- **docker ps** list the running containers
- **docker exec -it <container id> bash** to access the running container
- **docker commit <container id> <username/imagename>** creates a new image of an edited container
- **docker images** lists all the locally stored docker images

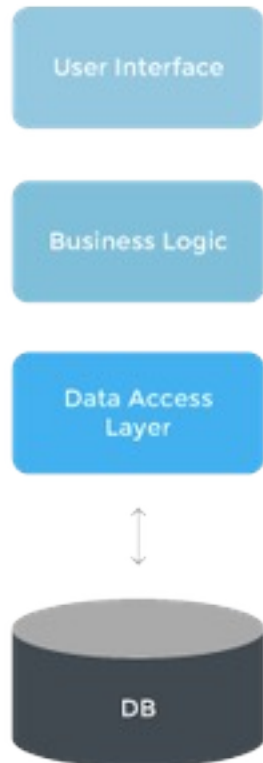
Unique features

- Containers run in the user space
- Each container has its own: process space, network interface, booting mechanism with configuration
- Share kernel with the host
- Can be packaged as Docker images to provide *microservices*.

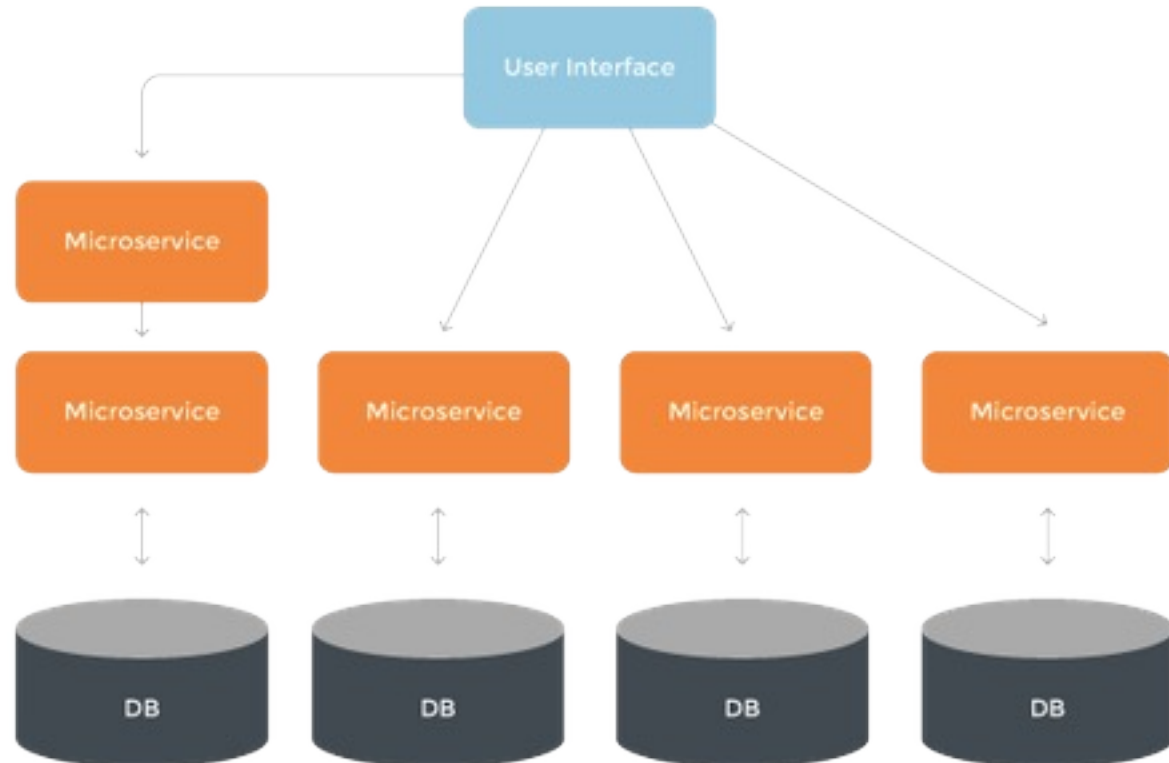
		
Size		
Startup		
Integration		

Monolithic architecture vs microservices

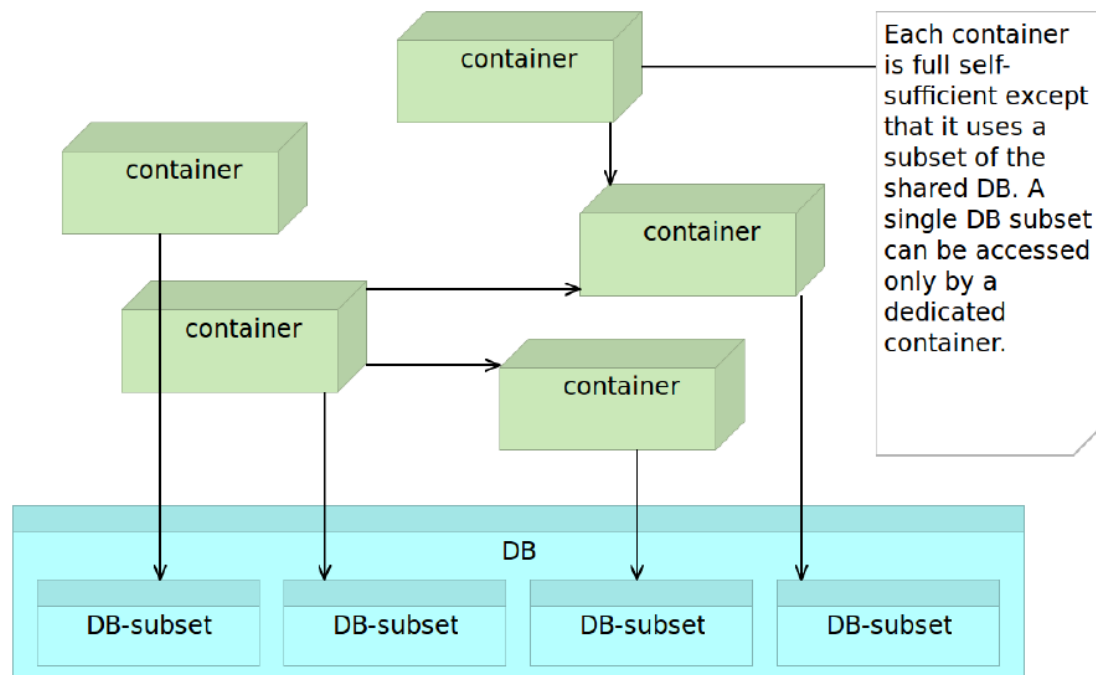
MONOLITHIC ARCHITECTURE



MICROSERVICES ARCHITECTURE



Microservices Accessing the Shared Database



Microservices Characteristics

- Many smaller (fine grained), clearly scoped services
 - Single Responsibility Principle
 - Independently Managed
- Clear ownership for each service
 - Typically need/adopt the “DevOps” model
- 100s of MicroServices
 - Need a Service Metadata Registry (Discovery Service)
- May be replicated as needed
- A microservice can be updated without interruption

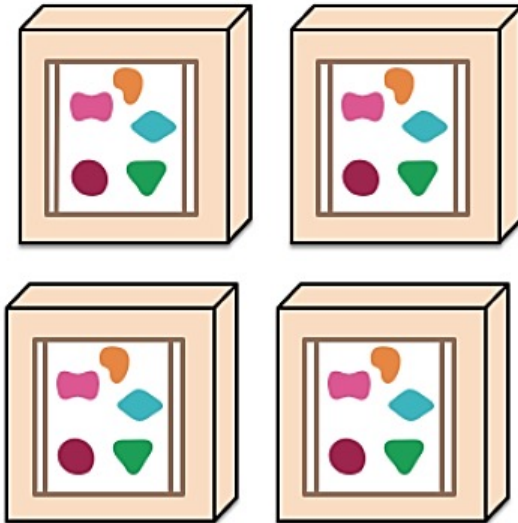


Microservices. Scalability

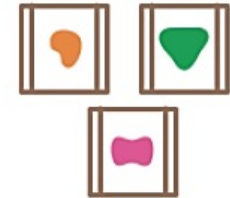
A monolithic application puts all its functionality into a single process...



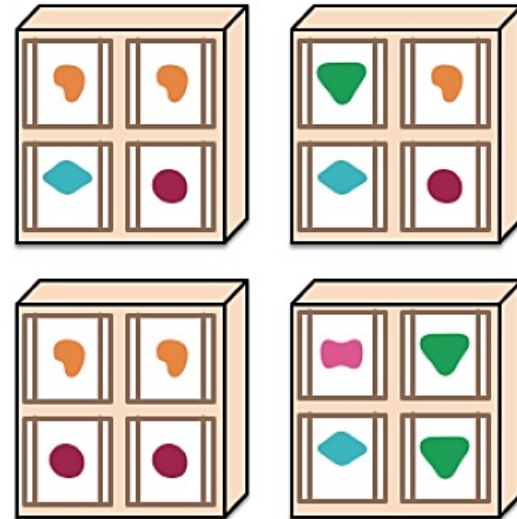
... and scales by replicating the monolith on multiple servers



A microservices architecture puts each element of functionality into a separate service...



... and scales by distributing these services across servers, replicating as needed.



Back from VMs & Containers

- We need to do a context switch back here.

CS370 Operating Systems

Colorado State University

Yashwant K Malaiya

Spring 2022 Lecture 20



Virtual Memory

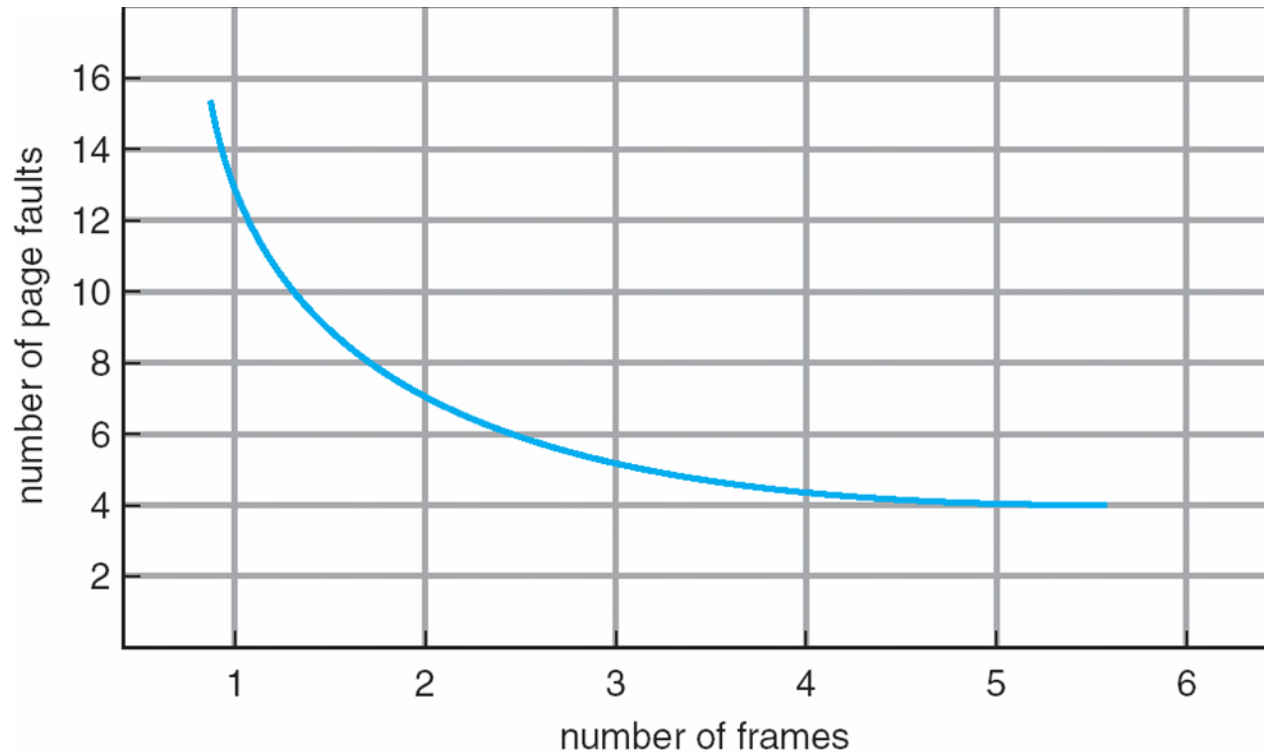
Slides based on

- Text by Silberschatz, Galvin, Gagne
- Various sources

Page Replacement Algorithms

- **Page-replacement algorithm**
 - Which frames to replace
 - Want lowest page-fault rate
- **Evaluate algorithm** by running it on a particular string of memory references (reference string) and computing the number of page faults on that string
 - String is just page numbers, not full addresses
 - Repeated access to the same page does not cause a page fault
 - Results depend on number of frames available
- In all our examples, we use **3** frames, and the **reference string** of referenced page numbers is
7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1

Graph of Page Faults Versus The Number of Frames



What we would generally expect

Page Replacement Algorithms

Algorithms

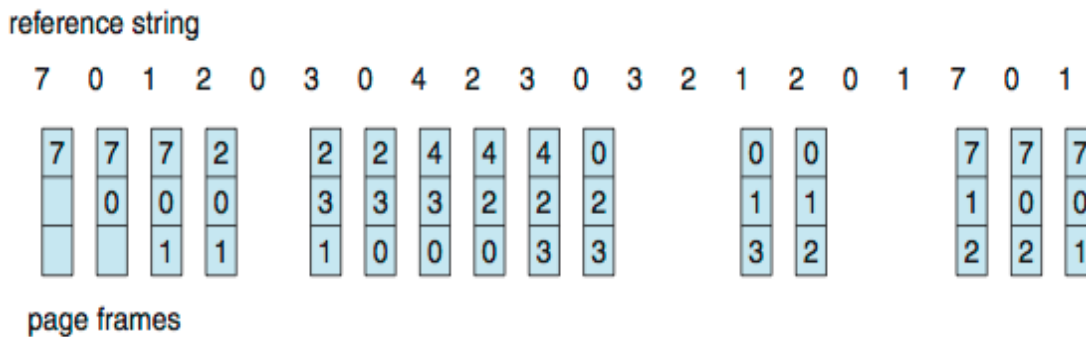
- FIFO
- “Optimal”
- The Least Recently Used (LRU)
 - Exact Implementations
 - Time of use field, Stack
 - Approximate implementations
 - Reference bit
 - Reference bit with shift register
 - Second chance: clock
 - Enhanced second chance: dirty or not?
- Other

FIFO page replacement algorithm: Out with the old; in with the new

- When a page must be replaced
 - Replace the oldest one
- OS maintains list of all pages currently in memory
 - Page at head of the list: Oldest one
 - Page at the tail: Recent arrival
- During a page fault
 - Page at the head is removed
 - New page added to the tail

First-In-First-Out (FIFO) Algorithm

- Reference string:
7,0,1,2,0,3,0,4,2,3,0,3,2,1,2,0,1,7,0,1
- 3 frames (3 pages can be in memory at a time per process)

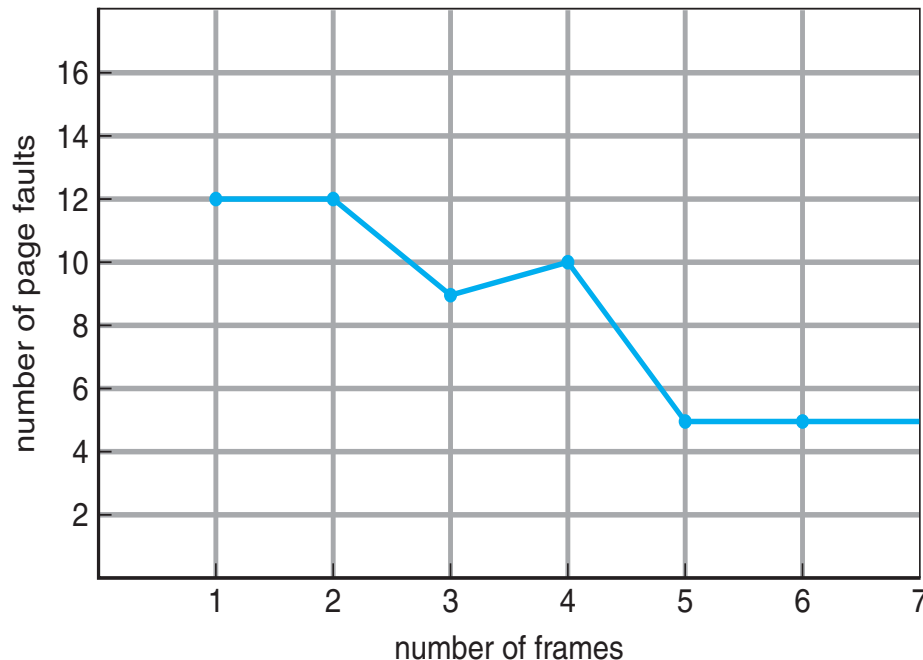


- 15 page faults (out of 20 accesses)
- Sometimes a page is needed soon after replacement 7,0,1,2,0,**3 (0 out),0, ..**

Belady's Anomaly

- Consider Page reference string **1,2,3,4,1,2,5,1,2,3,4,5**
 - 3 frames, 9 faults, 4 frames 10 faults! Try yourself.
 - Sometimes adding more frames can cause more page faults!

- **Belady's Anomaly**



Lazlo Belady was here at CSU. Guest in my CS530!



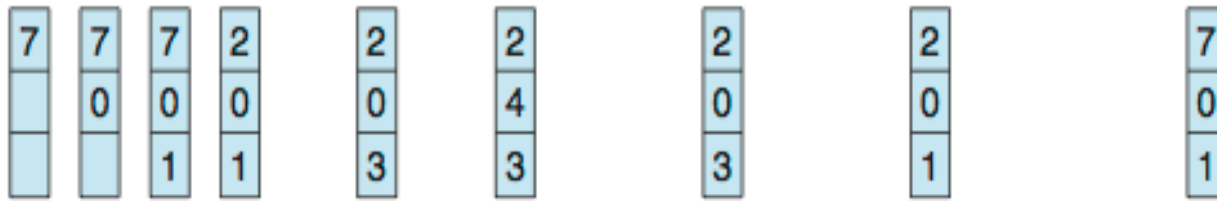
Budapest, 1928

“Optimal” Algorithm Belady 66

- Replace page that will not be used for longest period of time

reference string

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



page frames

- 4th access: replace 7 because we will not use it for the longest time...
- 9 page replacements is optimal for the example
- But how do we know the future pages needed?
 - Can't read the future in reality.
- Used for *measuring* how well an algorithm performs.

Least Recently Used (LRU) Algorithm

- Use past knowledge rather than future
- Replace page that has not been used in the most amount of time (4th access – page 7 is least recently used ..._)
- Associate time of last use with each page

Track usage carefully!

reference string

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

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

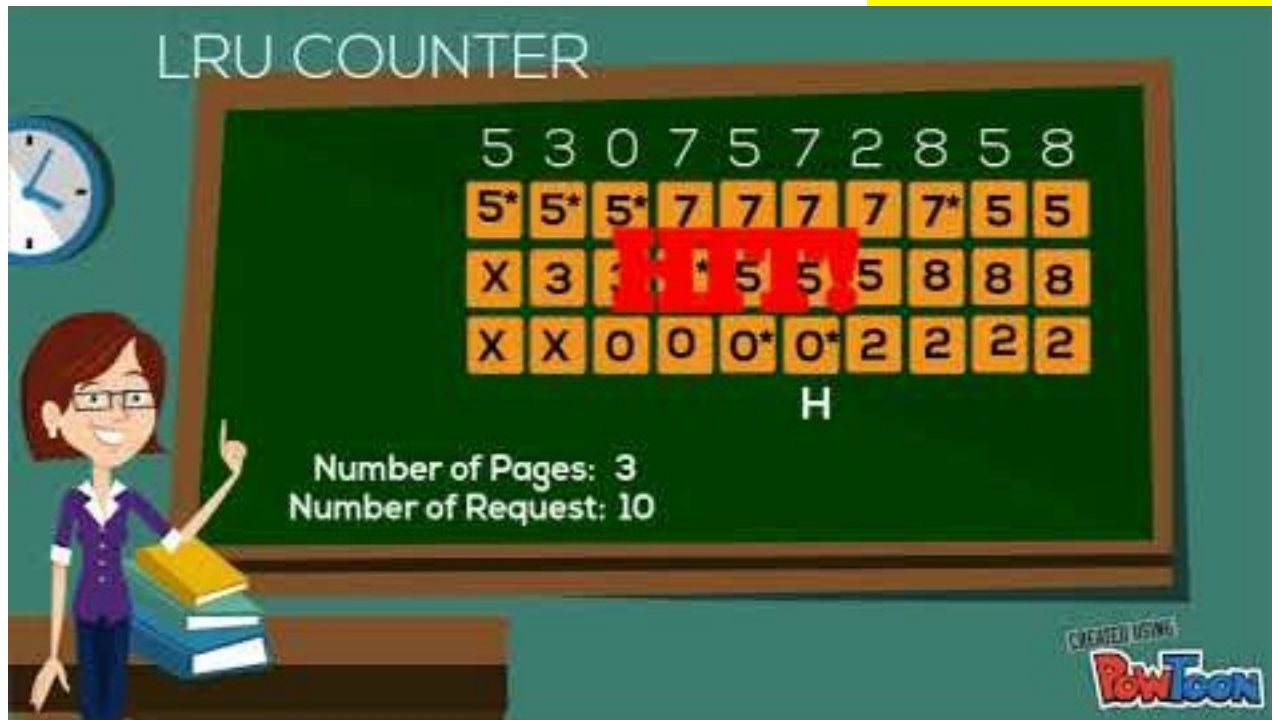
page frames

- 12 faults – better than FIFO (15) but worse than OPT (9)
- Generally good algorithm and frequently used
- But how to implement it by tracking the page usage?

LRU and OPT are cases of *stack algorithms* that don't have Belady's Anomaly

Least Recently Used (LRU) Algorithm

LRU page number is marked (*).
Unmarked if that page is accessed.



LRU applied to cache memory.

Least Recently Used (LRU) Algorithm

- * Use past knowledge rather than future
- 12 faults – better than FIFO (15) but worse than OPT (9)
- Tracking the page usage. One approach: mark least recently used page each time.

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

- Other approach: use stack for tracking (soon)

LRU Algorithm: Implementations

Possible tracking implementations

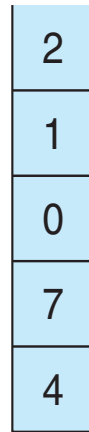
- Counter implementation
 - Every page entry has a counter; every time page is referenced through this entry, copy the clock into the counter
 - When a page needs to be changed, look at the counters to find smallest value
 - Search through table needed
- Stack implementation
 - Keep a stack of page numbers in a double link form:
 - Page referenced:
 - move it to the top
 - requires 6 pointers to be changed
 - Each update expensive
 - No search for replacement needed (bottom is least recently used)

Use Of A Stack to Record Most Recent Page References

reference string

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

Most recently used ->



stack
before
a



stack
after
b



This shows tracking stack,
not actual frames.

Too slow if done in software

Use Of A Stack to Record Most Recent Page References

Examine this at home.

	4	7	0	7	1	0	1	2	1	2	7	1	2
Most recently used ->	4	7	0	7	1	0	1	2	1	2	7	1	2
		4	7	0	7	1	0	1	2	1	2	7	1
			4	4	0	7	7	0	0	0	1	2	7
					4	4	4	7	7	7	0	0	0
Least recently used ->								4	4	4	4	4	4

Detailed version of previous slide.
This shows tracking stack, not actual frames.

Use Of A Stack to Record Most Recent Page References

reference string

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

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

page frames

Earlier problem (upper) revisited.
This shows tracking stack, not actual frames.

	7	0	1	2	0	3	0	4	2	3	0	3	2	1	2	0	1	7	0	1
MRU->	7	0	1	2	0	3	0	4	2	3	0	3								
		7	0	1	2	0	3	0	4	2	3	0								
LRU->			7	0	1	2	3	0	4	2	2									

LRU Approximation Algorithms

- LRU needs special hardware and still slow
- **Reference** 1 bit per frame to track history
 - With each page associate a bit, initially = 0
 - When the page is referenced, bit set to 1
 - Replace any page with reference bit = 0 (if one exists)
 - 0 implies not used since initialization
 - We do not know the order, however.
- Advanced schemes using more bits: preserve more information about the order

Ref bit + history shift register

LRU approximation 9 bits per frame to track history

Ref bit: 1 indicates used, Shift register records history. Examples:

Ref Bit	Shift Register	Shift Register after OS timer interrupt
1	0000 0000	1000 0000
1	1001 0001	1100 1000
0	0110 0011	0011 0001

- Interpret 8-bit bytes as **unsigned integers**
- Page with the lowest number is the LRU page: replace.

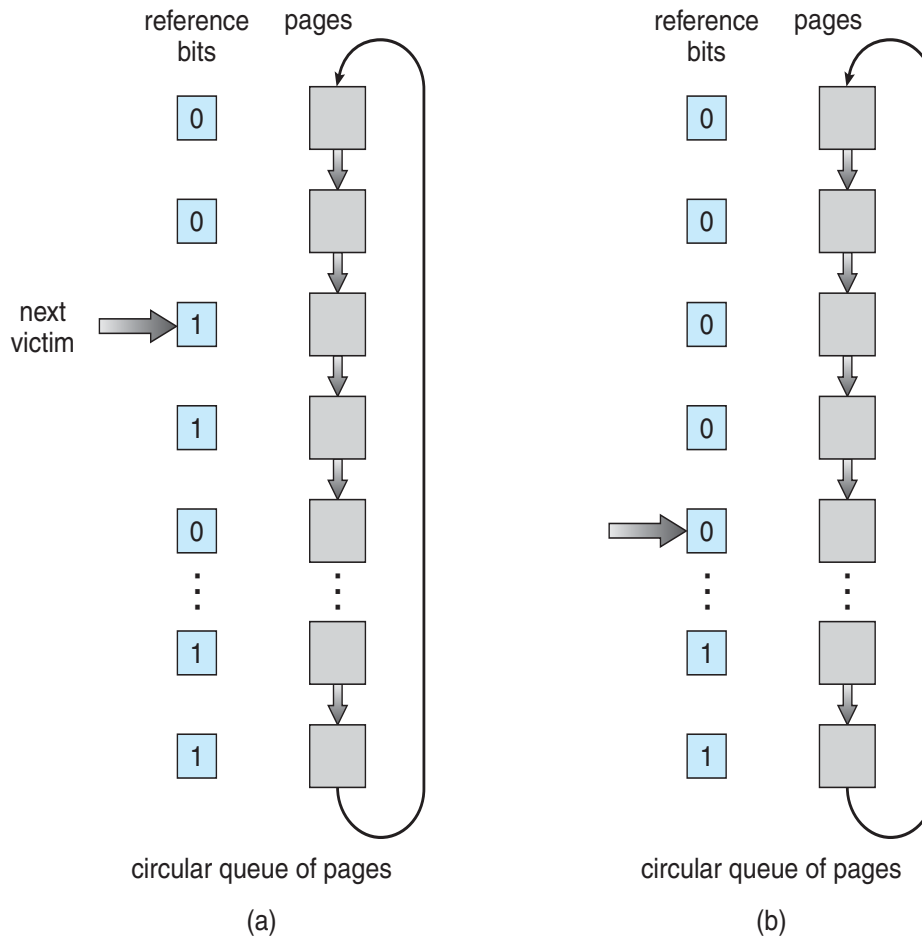
Examples:

- 00000000 : Not used in last 8 periods
- 01100101 : Used 4 times in the last 8 periods
- 11000100 used more recently than 01110111

Second-chance algorithm

- **Second-chance algorithm**
 - Generally FIFO, plus hardware-provided reference bit
 - Avoid throwing out a heavily used page
 - **“Clock”** replacement (using circular queue): hand as a pointer
 - Consider next page
 - Reference bit = 0 -> replace it
 - reference bit = 1 then: give it another chance
 - set reference bit 0, leave page in memory
 - consider next page, subject to same rules

Second-Chance (clock) Page-Replacement Algorithm



- **Clock** replacement: hand as a pointer
- Consider next page
 - Reference bit = 0 -> replace it
 - reference bit = 1 then:
 - set reference bit 0, leave page in memory
 - consider next page, subject to same rules

Example:

(a) Change to 0, give it another chance

(b) Already 0. Replace page

Enhanced Second-Chance Algorithm

Improve algorithm by using reference bit and modify bit (if available) in concert [clean page: better replacement candidate](#)

- Take ordered pair (reference, [modify](#))
 1. (0, [0](#)) neither recently used not modified – best page to replace
 2. (0, [1](#)) not recently used but modified – not quite as good, must write out before replacement
 3. (1, [0](#)) recently used but clean – probably will be used again soon
 4. (1, [1](#)) recently used and modified – probably will be used again soon and need to write out before replacement
- When page replacement called for, use the clock scheme but use the four classes replace page in lowest non-empty class
 - Might need to search circular queue several times

Counting Algorithms

- Keep a counter of the number of references that have been made to each page
 - Not common
- **Least Frequently Used (LFU) Algorithm:**
replaces page with smallest count
- **Most Frequently Used (MFU) Algorithm:**
based on the argument that the page with the smallest count was probably just brought in and has yet to be used

Clever Techniques for enhancing Perf

- Keep a buffer (pool) of free frames, always
 - Then frame available when needed, not found at fault time
 - Read page into free frame and select victim to evict and add to free pool
 - When convenient, evict victim
- Keep list of modified pages
 - When backing store is otherwise idle, write pages there and set to non-dirty (being proactive!)
- Keep free frames' previous contents intact and note what is in them
 - If referenced again before reused, no need to load contents again from disk
 - Generally useful to reduce penalty if wrong victim frame selected

Buffering and applications

- Some applications (like databases) often understand their memory/disk usage better than the OS
 - Provide their own buffering schemes
 - If both the OS and the application were to buffer
 - Twice the I/O is being utilized for a given I/O
 - OS may provide “raw access” disk to special programs without file system services.