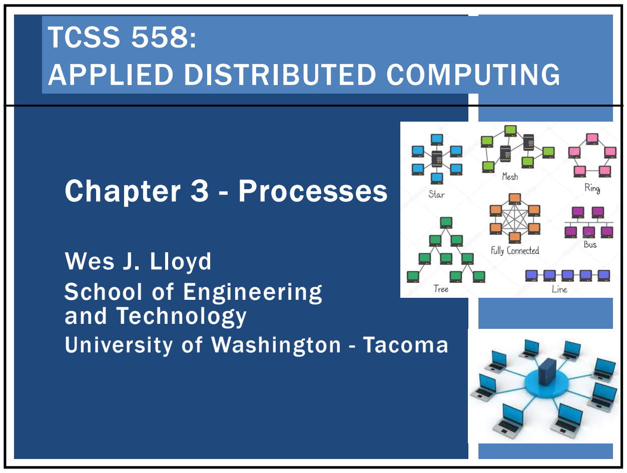
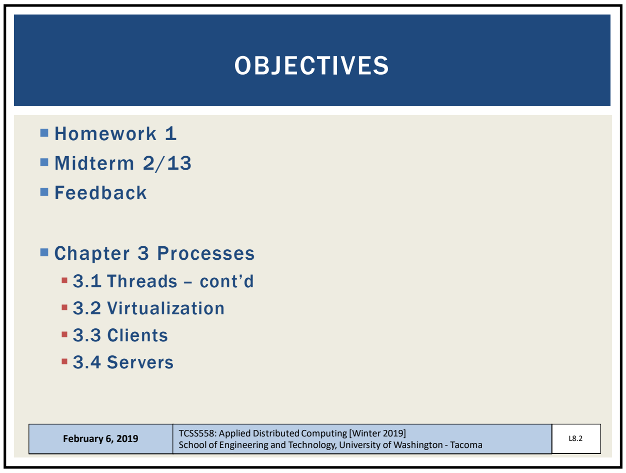


TCSS 558: APPLIED DISTRIBUTED COMPUTING

Chapter 3 - Processes

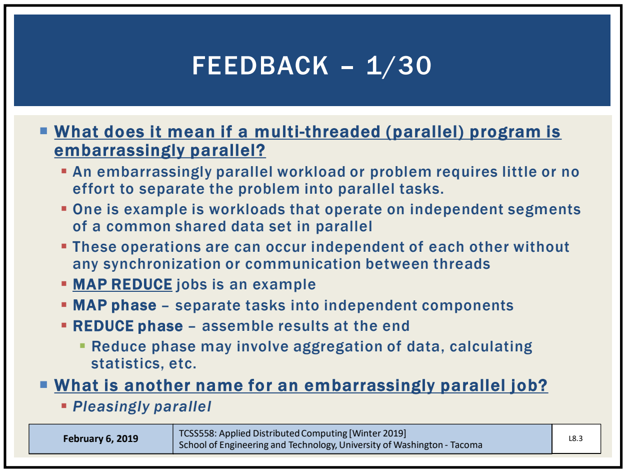
Wes J. Lloyd
School of Engineering and Technology
University of Washington - Tacoma

Star
Mesh
Ring
Fully Connected
Bus
Tree
Line



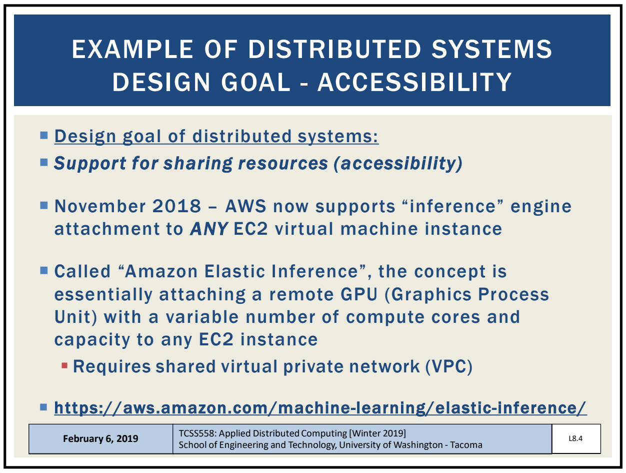
OBJECTIVES

- Homework 1
- Midterm 2/13
- Feedback
- Chapter 3 Processes
  - 3.1 Threads – cont'd
  - 3.2 Virtualization
  - 3.3 Clients
  - 3.4 Servers



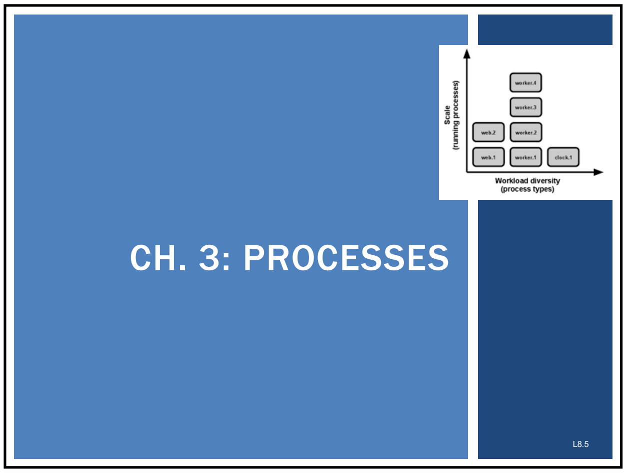
FEEDBACK – 1/30

- What does it mean if a multi-threaded (parallel) program is embarrassingly parallel?
  - An embarrassingly parallel workload or problem requires little or no effort to separate the problem into parallel tasks.
  - One is example is workloads that operate on independent segments of a common shared data set in parallel
  - These operations are can occur independent of each other without any synchronization or communication between threads
  - **MAP REDUCE** jobs is an example
  - **MAP phase** – separate tasks into independent components
  - **REDUCE phase** – assemble results at the end
    - Reduce phase may involve aggregation of data, calculating statistics, etc.
- What is another name for an embarrassingly parallel job?
  - *Pleasingly parallel*



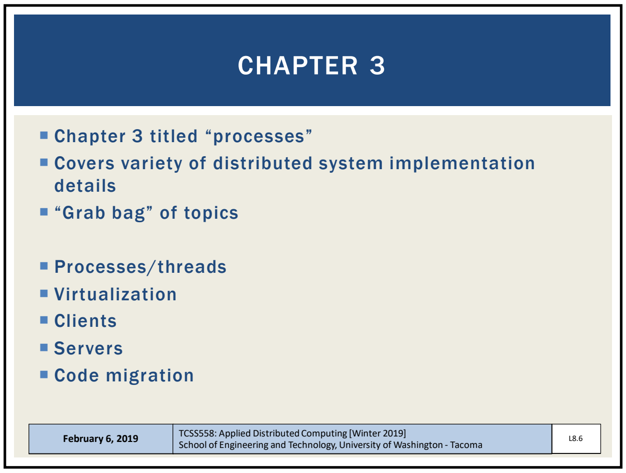
EXAMPLE OF DISTRIBUTED SYSTEMS DESIGN GOAL - ACCESSIBILITY

- Design goal of distributed systems:
- ***Support for sharing resources (accessibility)***
- November 2018 – AWS now supports "inference" engine attachment to ***ANY*** EC2 virtual machine instance
- Called "Amazon Elastic Inference", the concept is essentially attaching a remote GPU (Graphics Process Unit) with a variable number of compute cores and capacity to any EC2 instance
  - Requires shared virtual private network (VPC)
- https://aws.amazon.com/machine-learning/elastic-inference/



CH. 3: PROCESSES

Scale (running processes)
worker.4
worker.3
web.2
worker.2
web.1
worker.1
clock.1
Workload diversity (process types)



CHAPTER 3

- Chapter 3 titled "processes"
- Covers variety of distributed system implementation details
- "Grab bag" of topics
- Processes/threads
- Virtualization
- Clients
- Servers
- Code migration

THREADS

- For implementing a server (or client) threads offer many advantages vs. heavy weight processes
- **What is the difference between a process and a thread?**
  - Review from Operating Systems
- ***Key difference***: **what do threads share amongst each other that processes do not…. ?**
- **What are the segments of a program stored in memory?**
  - Heap segment (dynamic shared memory)
  - Code segment
  - Stack segment
  - Data segment (global variables)

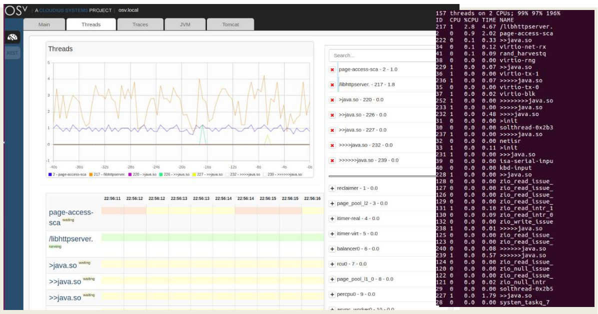
THREADS - 2

- **Do several processes on an operating system share…**
  - **Heap segment?**
  - **Stack segment?**
  - **Code segment?**
- **Can we run multiple copies of the same code?**
- These may be managed as shared pages (across processes) in memory
- Processes are isolated from each other by the OS
  - Each has a separate heap, stack, code segment

THREADS - 3

- Threads avoid the overhead of process creation
- No new heap or code segments required
- **What is a context switch?**
- Context switching among threads is considered to be more efficient than context switching processes
- Less elements to swap-in and swap-out
- Unikernel: specialized single process OS for the cloud
- Example: Osv, Clive, MirageOS (see: **http://unikernel.org/projects/**)
- Single process operating system with many threads
- Developed for the cloud to run only one application at a time

OSV: ONE PROCESS, MANY THREADS



```
157 threads on 2 CPUs; 99% 97% 196%
ID  CPU %CPU TIME NAME
217 1   2.8  4.67 /libhttpserver.
2   0   0.9  2.02 page-access-sca
222 0   0.1  0.33 >>java.so
34  0   0.1  0.12 virtio-net-rx
41  0   0.1  0.09 rand_harvestq
38  0   0.0  0.00 virtio-rng
229 1   0.0  0.07 >>java.so
36  1   0.0  0.00 virtio-tx-1
236 1   0.0  0.07 >>>>>java.so
35  0   0.0  0.00 virtio-tx-0
37  1   0.0  0.02 virtio-blk
252 1   0.0  0.00 >>>>>>>>java.so
233 1   0.0  0.00 >>>>>java.so
232 1   0.0  0.48 >>>>java.so
31  0   0.0  0.00 >init
30  0   0.0  0.00 solthread-0x2b3
237 1   0.0  0.00 >>>>>java.so
32  0   0.0  0.00 netisr
33  1   0.0  0.11 >init
231 1   0.0  0.00 >>>java.so
39  0   0.0  0.00 isa-serial-inpu
40  0   0.0  0.00 kbd-input
228 1   0.0  0.00 >>java.so
128 0   0.0  0.00 zio_read_issue_
127 0   0.0  0.00 zio_read_issue_
126 0   0.0  0.00 zio_read_issue_
129 0   0.0  0.00 zio_read_issue_
131 1   0.0  0.10 zio_read_intr_1
130 0   0.0  0.09 zio_read_intr_0
132 0   0.0  0.00 zio_write_issue
238 1   0.0  0.01 >>>>>java.so
125 0   0.0  0.00 zio_read_issue_
123 0   0.0  0.00 zio_read_issue_
240 0   0.0  0.08 >>>>>>java.so
239 1   0.0  0.57 >>>>>>java.so
124 0   0.0  0.00 zio_read_issue_
120 0   0.0  0.00 zio_null_issue
122 0   0.0  0.00 zio_read_issue_
121 0   0.0  0.02 zio_null_intr
29  0   0.0  0.00 solthread-0x2b5
227 1   0.0  1.79 >>java.so
28  0   0.0  0.00 system_taskq_7
```

THREADS - 4

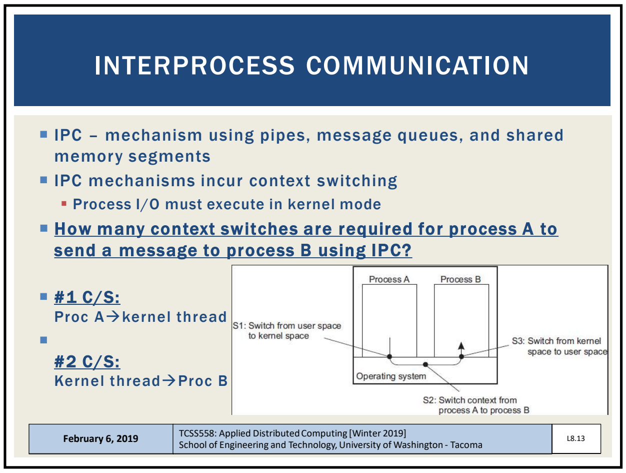
- Important implications with threads:
- (1) multi-threading should lead to performance gains
- (2) thread programming requires additional effort when threads share memory
  - Known as thread **synchronization**, or enabling **concurrency**
- Access to **critical sections** of code which modify shared variables must be **mutually exclusive**
  - No more than one thread can execute at any given time
  - Critical sections must run **atomically** on the CPU

BLOCKING THREADS

- Example: spreadsheet with formula to compute sum of column
- User modifies values in column
- Multiple threads:

1. Supports interaction (UI) activity with user
2. Updates spreadsheet calculations in parallel
3. Continually backs up spreadsheet changes to disk

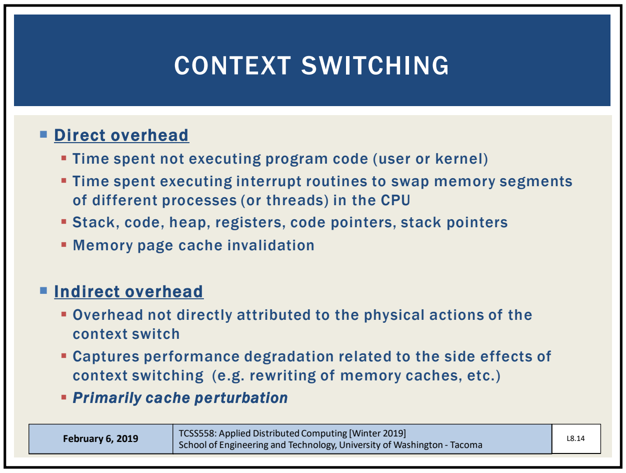
- Single core CPU
  - Tasks appear as if they are performed simultaneously
- Multi core CPU
  - Tasks ***execute*** simultaneously



INTERPROCESS COMMUNICATION

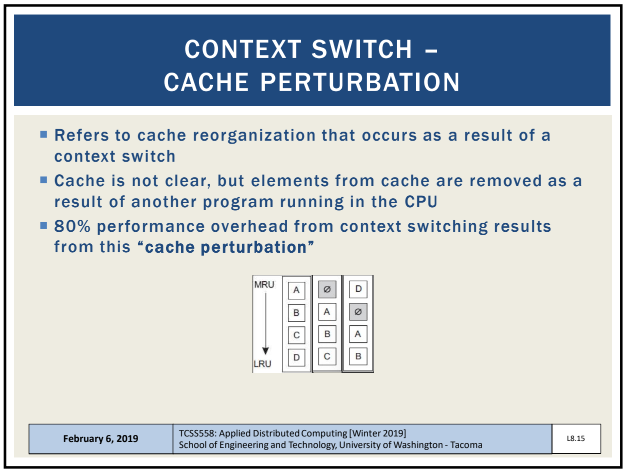
- IPC – mechanism using pipes, message queues, and shared memory segments
- IPC mechanisms incur context switching
  - Process I/O must execute in kernel mode
- **How many context switches are required for process A to send a message to process B using IPC?**
- **#1 C/S:** Proc A→kernel thread
- **#2 C/S:** Kernel thread→Proc B

Process A
Process B
S1: Switch from user space to kernel space
S3: Switch from kernel space to user space
Operating system
S2: Switch context from process A to process B



CONTEXT SWITCHING

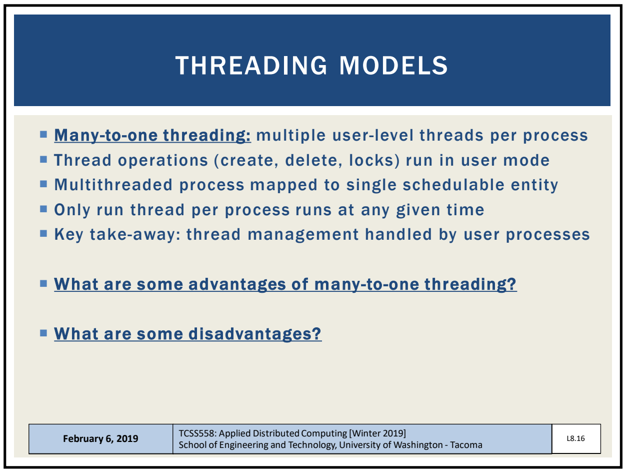
- **Direct overhead**
  - Time spent not executing program code (user or kernel)
  - Time spent executing interrupt routines to swap memory segments of different processes (or threads) in the CPU
  - Stack, code, heap, registers, code pointers, stack pointers
  - Memory page cache invalidation
- **Indirect overhead**
  - Overhead not directly attributed to the physical actions of the context switch
  - Captures performance degradation related to the side effects of context switching (e.g. rewriting of memory caches, etc.)
  - ***Primarily cache perturbation***



CONTEXT SWITCH – CACHE PERTURBATION

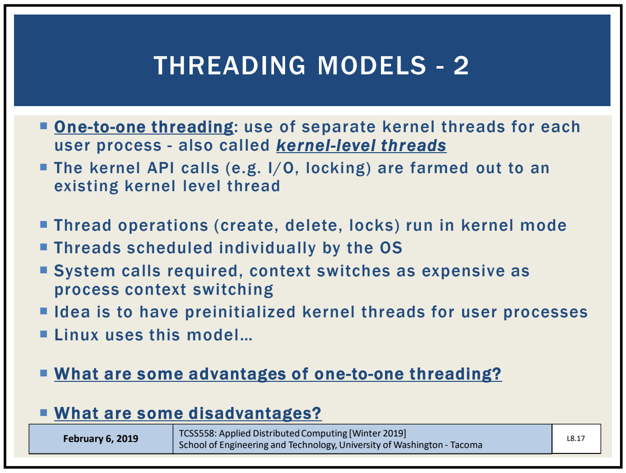
- Refers to cache reorganization that occurs as a result of a context switch
- Cache is not clear, but elements from cache are removed as a result of another program running in the CPU
- 80% performance overhead from context switching results from this "**cache perturbation**"

MRU
LRU
A
B
C
D
∅
A
B
C
D
∅
A
B



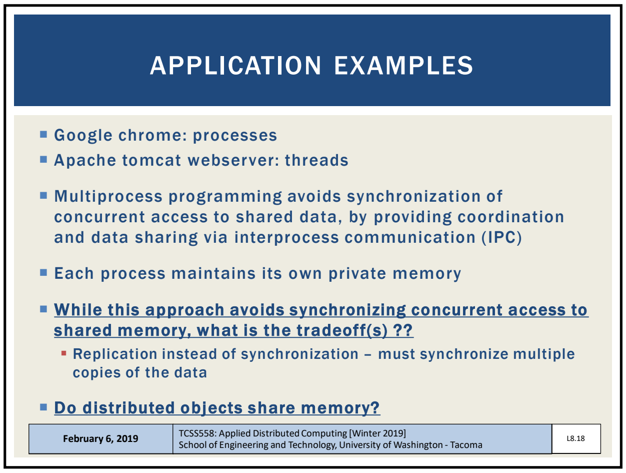
THREADING MODELS

- **Many-to-one threading:** multiple user-level threads per process
- Thread operations (create, delete, locks) run in user mode
- Multithreaded process mapped to single schedulable entity
- Only run thread per process runs at any given time
- Key take-away: thread management handled by user processes
- **What are some advantages of many-to-one threading?**
- **What are some disadvantages?**



THREADING MODELS - 2

- **One-to-one threading**: use of separate kernel threads for each user process - also called ***kernel-level threads***
- The kernel API calls (e.g. I/O, locking) are farmed out to an existing kernel level thread
- Thread operations (create, delete, locks) run in kernel mode
- Threads scheduled individually by the OS
- System calls required, context switches as expensive as process context switching
- Idea is to have preinitialized kernel threads for user processes
- Linux uses this model…
- **What are some advantages of one-to-one threading?**
- **What are some disadvantages?**



APPLICATION EXAMPLES

- Google chrome: processes
- Apache tomcat webserver: threads
- Multiprocess programming avoids synchronization of concurrent access to shared data, by providing coordination and data sharing via interprocess communication (IPC)
- Each process maintains its own private memory
- **While this approach avoids synchronizing concurrent access to shared memory, what is the tradeoff(s) ??**
  - Replication instead of synchronization – must synchronize multiple copies of the data
- **Do distributed objects share memory?**

MULTITHREADED CLIENTS

- **Web browser**
- Uses threads to load and render portions of a web page to the user in parallel
- A client could have dozens of concurrent connections all loading in parallel

- **testFibPar.sh**
- Assignment 0 client script (GNU parallel)

- **Important benefits:**
- Several connections can be opened simultaneously
- Client: dozens of concurrent connections to the webserver all loading data in parallel

MULTIPLE THREADS

- In Linux, threads also receive a process ID (PID)
- To display threads of a process in Linux:

- Identify parent process explicitly:

- top –H –p <pid>
- htop –p <pid>
- ps –iT <pid>

- Virtualbox process ~ 44 threads
- No mapping to guest # of processes/threads

PROCESS METRICS

Disk
- dsr: disk sector reads
- dsreads: disk sector reads completed
- drm: merged adjacent disk reads
- readtime: time spent reading from disk
- dsw: disk sector writes
- dswrites: disk sector writes completed
- dwm: merged adjacent disk writes
- writetime: time spent writing to disk

CPU
- cpuUsr: CPU time in user mode
- cpuKrn: CPU time in kernel mode
- cpuIdle: CPU idle time
- cpuIoWait: CPU time waiting for I/O
- cpuIntSrvc: CPU time serving interrupts
- cpuSftIntSrvc: CPU time serving soft interrupts
- cpuNice: CPU time executing prioritized processes
- cpuSteal: CPU ticks lost to virtualized guests
- contextsw: # of context switches
- loadavg: (avg # proc / 60 secs)

Network
- nbs: network bytes sent
- nbr: network bytes received

LOAD AVERAGE

- Reported by: `top`, `htop`, `w`, `uptime`, and `/proc/loadavg`
- Updated every 5 seconds
- Average number of processes using or waiting for the CPU
- Three numbers show exponentially decaying usage for 1 minute, 5 minutes, and 15 minutes
- One minute average: exponentially decaying average
- Load average = 1 ▪ (avg last minute load) – 1/e ▪ (avg load since boot)

- 1.0 = 1-CPU core fully loaded
- 2.0 = 2-CPU cores
- 3.0 = 3-CPU cores . . .

THREAD-LEVEL PARALLELISM

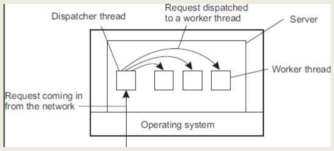
- Metric – measures degree of parallelism realized by running system, by calculating average utilization:

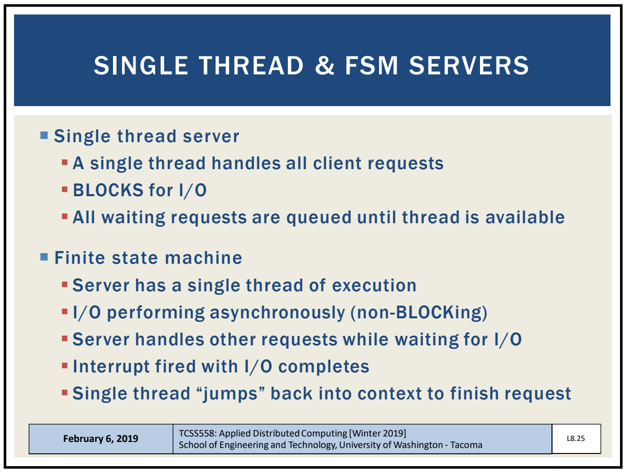
$$TLP = \frac{\sum_{i=1}^{N} i \cdot c_i}{1 - c_0}$$

- Ci – fraction of time that exactly I threads are executed
- N – maximum threads that can execute at any one time
- Web browsers found to have TLP from 1.5 to 2.5
- Clients for web browsing can utilize from 2 to 3 CPU cores
- Any more cores are redundant, and potentially wasteful
- **Measure TLP to understand how many CPUs to provision**

MULTITHREADED SERVERS

- Multiple threads essential for servers in distributed systems
- Even on single-core machines greatly improves performance
- Take advantage of idle/blocking time
- Two designs:
  - Generate new thread for every request
  - Thread pool – pre-initialize set of threads to service requests

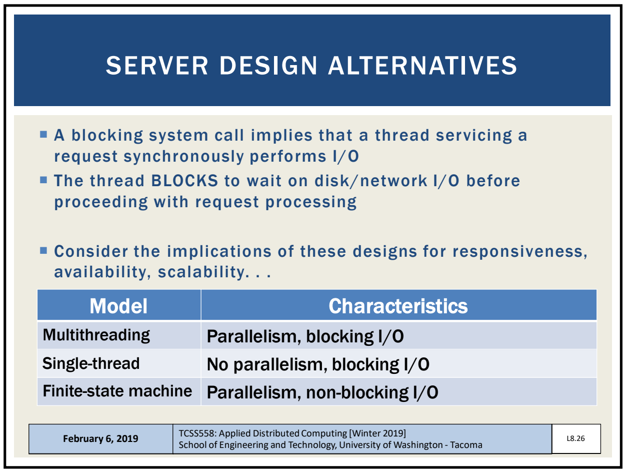

Dispatcher thread
Request dispatched to a worker thread
Server
Worker thread
Request coming in from the network
Operating system



SINGLE THREAD & FSM SERVERS

- Single thread server
  - A single thread handles all client requests
  - BLOCKS for I/O
  - All waiting requests are queued until thread is available
- Finite state machine
  - Server has a single thread of execution
  - I/O performing asynchronously (non-BLOCKing)
  - Server handles other requests while waiting for I/O
  - Interrupt fired with I/O completes
  - Single thread "jumps" back into context to finish request

February 6, 2019 | TCSS558: Applied Distributed Computing [Winter 2019] School of Engineering and Technology, University of Washington - Tacoma | L8.25

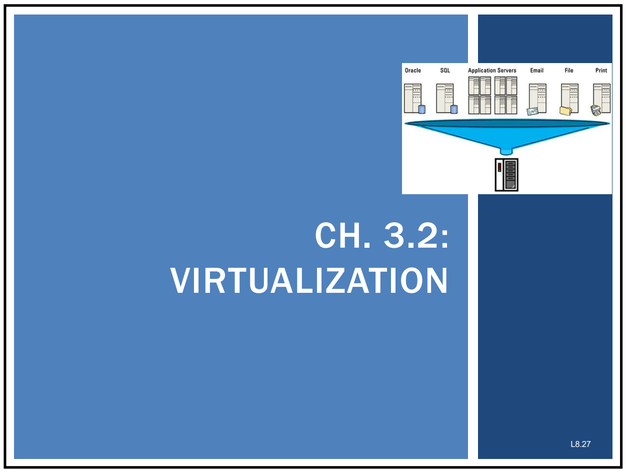


SERVER DESIGN ALTERNATIVES

- A blocking system call implies that a thread servicing a request synchronously performs I/O
- The thread BLOCKS to wait on disk/network I/O before proceeding with request processing
- Consider the implications of these designs for responsiveness, availability, scalability. . .

Model	Characteristics
Multithreading	Parallelism, blocking I/O
Single-thread	No parallelism, blocking I/O
Finite-state machine	Parallelism, non-blocking I/O

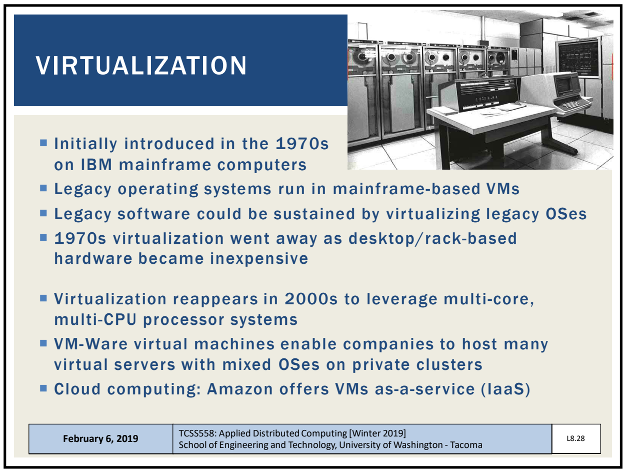
February 6, 2019 | TCSS558: Applied Distributed Computing [Winter 2019] School of Engineering and Technology, University of Washington - Tacoma | L8.26



Oracle SQL Application Servers Email File Print

CH. 3.2: VIRTUALIZATION

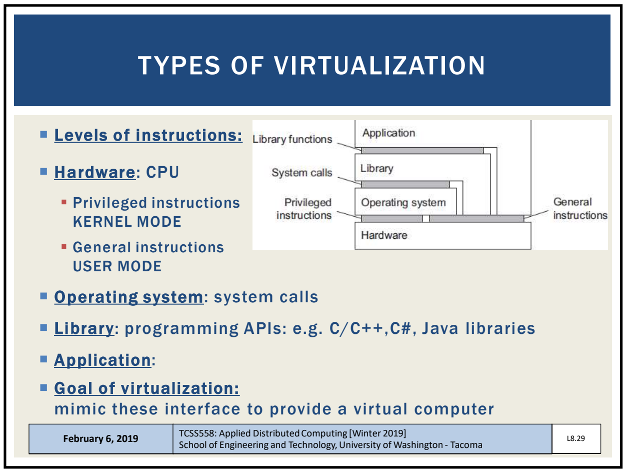
L8.27



VIRTUALIZATION

- Initially introduced in the 1970s on IBM mainframe computers
- Legacy operating systems run in mainframe-based VMs
- Legacy software could be sustained by virtualizing legacy OSes
- 1970s virtualization went away as desktop/rack-based hardware became inexpensive
- Virtualization reappears in 2000s to leverage multi-core, multi-CPU processor systems
- VM-Ware virtual machines enable companies to host many virtual servers with mixed OSes on private clusters
- Cloud computing: Amazon offers VMs as-a-service (IaaS)

February 6, 2019 | TCSS558: Applied Distributed Computing [Winter 2019] School of Engineering and Technology, University of Washington - Tacoma | L8.28

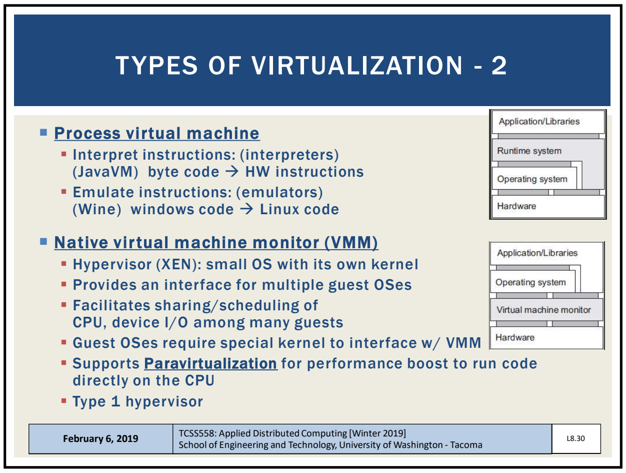


TYPES OF VIRTUALIZATION

- **Levels of instructions:**
- **Hardware**: CPU
  - Privileged instructions KERNEL MODE
  - General instructions USER MODE
- **Operating system**: system calls
- **Library**: programming APIs: e.g. C/C++,C#, Java libraries
- **Application**:
- **Goal of virtualization:** mimic these interface to provide a virtual computer

Library functions
Application
System calls
Library
Privileged instructions
Operating system
General instructions
Hardware

February 6, 2019 | TCSS558: Applied Distributed Computing [Winter 2019] School of Engineering and Technology, University of Washington - Tacoma | L8.29



TYPES OF VIRTUALIZATION - 2

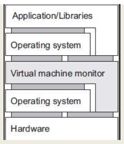
- **Process virtual machine**
  - Interpret instructions: (interpreters) (JavaVM) byte code → HW instructions
  - Emulate instructions: (emulators) (Wine) windows code → Linux code
- **Native virtual machine monitor (VMM)**
  - Hypervisor (XEN): small OS with its own kernel
  - Provides an interface for multiple guest OSes
  - Facilitates sharing/scheduling of CPU, device I/O among many guests
  - Guest OSes require special kernel to interface w/ VMM
  - Supports **Paravirtualization** for performance boost to run code directly on the CPU
  - Type 1 hypervisor

Application/Libraries
Runtime system
Operating system
Hardware

Application/Libraries
Operating system
Virtual machine monitor
Hardware

February 6, 2019 | TCSS558: Applied Distributed Computing [Winter 2019] School of Engineering and Technology, University of Washington - Tacoma | L8.30

TYPES OF VIRTUALIZATION - 3



- **Hosted virtual machine monitor (VMM)**
  - Runs atop of hosted operating system
  - Uses host OS facilities for CPU scheduling, I/O
  - Full virtualization
  - Type 2 hypervisor
  - **Virtualbox**
- *Textbook: note 3.5–good explanation of full vs. paravirtualization*
- **GOAL**: run all user mode instructions directly on the CPU
- x86 instruction set has ~17 privileged user mode instructions
- **Full virtualization**: scan the EXE, insert code around privileged instructions to divert control to the VMM
- **Paravirtualization**: special OS kernel eliminates side effects of privileged instructions

EVOLUTION OF AWS VIRTUALIZATION

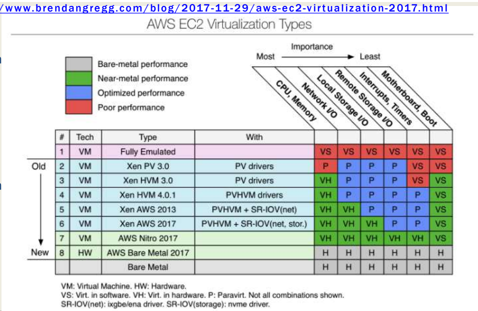
From http://www.brendangregg.com/blog/2017-11-29/aws-ec2-virtualization-2017.html

VS: Virtualization In software

P: Paravirtual

VH: Virtualization In Hardware

H: Hardware


AWS EC2 Virtualization Types

⬜ Bare-metal performance
🟩 Near-metal performance
🟦 Optimized performance
🟥 Poor performance

Importance
Most → Least
Old
New

#	Tech	Type	With	CPU, Memory	Network I/O	Local Storage I/O	Remote Storage I/O	Interrupts, Timers	Motherboard, Boot
1	VM	Fully Emulated		VS	VS	VS	VS	VS	VS
2	VM	Xen PV 3.0	PV drivers	P	P	P	P	VS	VS
3	VM	Xen HVM 3.0	PV drivers	VH	P	P	P	VS	VS
4	VM	Xen HVM 4.0.1	PVHVM drivers	VH	P	P	P	P	VS
5	VM	Xen AWS 2013	PVHVM + SR-IOV(net)	VH	VH	P	P	P	VS
6	VM	Xen AWS 2017	PVHVM + SR-IOV(net, stor.)	VH	VH	VH	P	P	VS
7	VM	AWS Nitro 2017		VH	VH	VH	VH	VH	VS
8	HW	AWS Bare Metal 2017		H	H	H	H	H	H
		Bare Metal		H	H	H	H	H	H

VM: Virtual Machine. HW: Hardware.
VS: Virt. in software. VH: Virt. in hardware. P: Paravirt. Not all combinations shown.
SR-IOV(net): ixgbe/ena driver. SR-IOV(storage): nvme driver.

AWS VIRTUALIZATION - 2

- **Full Virtualization - Fully Emulated**
  - Never used on EC2, before CPU extensions for virtualization
  - Can boot any unmodified OS
  - Support via slow emulation, performance 2x-10x slower
- **Paravirtualization: Xen PV 3.0**
  - Software: Interrupts, timers
  - Paravirtual: CPU, Network I/O, Local+Network Storage
  - Requires special OS kernels, interfaces with hypervisor for I/O
  - Performance 1.1x – 1.5x slower than "bare metal"
  - Instance store instances: 1ST & 2nd generation- m1.large, m2.xlarge
- **Xen HVM 3.0**
  - Hardware virtualization: **CPU**, **memory** **(CPU VT-x required)**
  - Paravirtual: network, storage
  - Software: interrupts, timers
  - EBS backed instances
  - m1, c1 instances

AWS VIRTUALIZATION - 3

- **XEN HVM 4.0.1**
  - Hardware virtualization: CPU, memory **(CPU VT-x required)**
  - Paravirtual: network, storage, **interrupts**, **timers**
- **XEN AWS 2013** *(diverges from opensource XEN)*
  - Provides hardware virtualization for CPU, memory, **network**
  - Paravirtual: storage, **interrupts**, **timers**
  - Called Single root I/O Virtualization (SR-IOV)
  - Allows sharing single physical PCI Express device (i.e. network adapter) with multiple VMs
  - Improves VM network performance
  - 3rd & 4th generation instances (c3 family)
  - Network speeds up to 10 Gbps and 25 Gbps
- **XEN AWS 2017**
  - Provides hardware virtualization for CPU, memory, network, **local disk**
  - Paravirtual: remote storage, **interrupts**, **timers**
  - Introduces hardware virtualization for EBS volumes (c4 instances)
  - Instance storage hardware virtualization (x1.32xlarge, i3 family)

AWS VIRTUALIZATION - 4

- **AWS Nitro 2017**
  - Provides hardware virtualization for CPU, memory, network, **local disk, remote disk, interrupts, timers**
  - All aspects of virtualization enhanced with HW-level support
  - November 2017
  - Goal: provide performance indistinguishable from "bare metal"
  - 5th generation instances – c5 instances (also c5d, c5n)
  - Based on KVM hypervisor
  - Overhead around ~1%

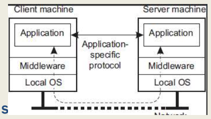
CH. 3.3: CLIENTS

TYPES OF CLIENTS

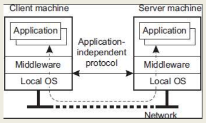
- Thick clients
  - Web browsers
    - Client-side scripting
  - Mobile apps
  - Multi-tier MVC apps
- Thin clients
  - Remote desktops/GUIs (very thin)

CLIENTS

- Application specific protocol
  - Thick clients
  - Clients maintain local data
  - Middleware (APIs)
  - Clients synchronize data with remote nodes
  - Example: shared calendar application



- Application independent
  - Thin clients
  - Client acts as a remote terminal
  - Provides interface to user (GUI / UI)
  - Server houses entire application stack

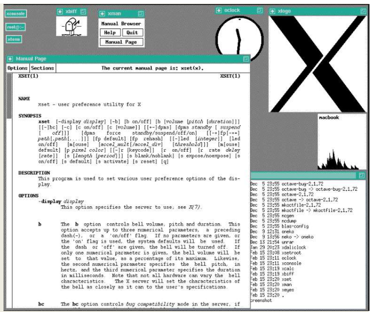


X WINDOWS

- Layered architecture to transport UI over network
- Remote desktop functionality for Linux/Unix systems
- X kernel acts as a server
  - Provides the **X protocol**: application level protocol
  - Xlib instances (client applications) exchange data and events with X kernels (servers)
  - Clients and servers on single machine → Linux GUI
  - Client and server communication transported over the network → remote Linux GUI

X WINDOWS - 2

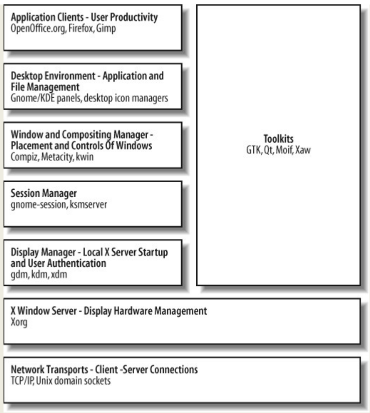
- Window manager:
  - Application running atop of X-windows which provides flair
  - Many variants
  - Without X windows is quite bland



```
The current manual page is: xset(x).
XSET(1)                                                        XSET(1)

NAME
       xset - user preference utility for X

SYNOPSIS
       xset  [-display display] [-b] [b on/off] [b [volume [pitch [duration]]]
       [[-]bc] [-c] [c on/off] [c [volume]] [[+-]dpms] [dpms standby [ suspend
       [    off]]]    [dpms    force    standby/suspend/off/on]    [[-+]fp[-+=]
       path[,path[,...]]] [fp default] [fp rehash]  [[-]led  [integer]]  [led
       on/off]    [m[ouse]    [accel_mult[/accel_div]    [threshold]]]    [m[ouse]
       default] [p pixel color] [[-]r [keycode]]  [r  on/off]  [r  rate  delay
       [rate]]   [s [length [period]]] [s blank/noblank] [s expose/noexpose] [s
       on/off] [s default] [s activate] [s reset] [q]

DESCRIPTION
       This program is used to set various user preference options of the dis-
       play.

OPTIONS
       -display display
               This option specifies the server to use; see X(7).

       b       The  b  option  controls bell volume, pitch and duration.  This
               option accepts up to three numerical  parameters,  a  preceding
               dash(-),  or  a  'on/off' flag.  If no parameters are given, or
               the 'on' flag is used, the system defaults will  be  used.   If
               the  dash  or  'off' are given, the bell will be turned off.  If
               only one numerical parameter is given, the bell volume will  be
               set  to  that value, as a percentage of its maximum.  Likewise,
               the second numerical parameter specifies  the  bell  pitch,  in
               hertz, and the third numerical parameter specifies the duration
               in milliseconds.  Note that not all hardware can vary the  bell
               characteristics.   The X server will set the characteristics of
               the bell as closely as it can to the user's specifications.

       bc      The bc option controls bug compatibility mode in the server.  if
```

```
Dec  5 23:55 octave-bug-2.1.72
Dec  5 23:55 octave-bug -> octave-bug-2.1.72
Dec  5 23:55 octave-2.1.72
Dec  5 23:55 octave -> octave-2.1.72
Dec  5 23:55 mkoctfile-2.1.72
Dec  5 23:55 mkoctfile -> mkoctfile-2.1.72
Dec  5 23:55 ncgen
Dec  5 23:55 ncdump
Dec  5 23:55 blas-config
Dec  9 12:31 oneko
Dec  9 13:56 neko -> oneko
Dec 13 21:54 unrar
Jan 29 20:23 xdaliclock
Feb 15 23:08 xsetroot
Feb 15 23:11 oclock
Feb 15 23:11 xconsole
Feb 15 23:19 xcalc
Feb 15 23:19 xbiff
Feb 15 23:20 xset
Feb 15 23:20 xman
Feb 15 23:20 xeyes
Feb 15 23:20 .
creenshot
```

- **Layered architecture**
- X-kernel: low level interface/APIs for controlling screen, capturing keyboard and mouse events (X window Server)
- Provided on Linux as Xlib
- Provides network enabled GUI
- Layering allows for use for custom window managers



Application Clients - User Productivity
OpenOffice.org, Firefox, Gimp

Desktop Environment - Application and File Management
Gnome/KDE panels, desktop icon managers

Window and Compositing Manager - Placement and Controls Of Windows
Compiz, Metacity, kwin

Session Manager
gnome-session, ksmserver

Display Manager - Local X Server Startup and User Authentication
gdm, kdm, xdm

Toolkits
GTK, Qt, Moif, Xaw

X Window Server - Display Hardware Management
Xorg

Network Transports - Client -Server Connections
TCP/IP, Unix domain sockets

EXAMPLE: VNC SERVER

- **How to Install VNC server on Ubuntu EC2 instance VM:**
- `sudo apt-get update`
- `# ubuntu 16.04`
- `sudo apt-get install ubuntu-desktop`
- `sudo apt-get install gnome-panel gnome-settings-daemon metacity nautilus gnome-terminal`
- `# on ubuntu 18.04`
- `sudo apt install xfce4 xfce4-goodies`
- `sudo apt-get install tightvncserver  # both`
- Start VNC server to create initial config file
- `vncserver :1`

EXAMPLE: VNC SERVER – UBUNTU 16.04

- On the VM: edit config file: `nano ~/.vnc/xstartup`
- Replace contents as below (Ubuntu 16.04):

```
#!/bin/sh

export XKL_XMODMAP_DISABLE=1
unset SESSION_MANAGER
unset DBUS_SESSION_BUS_ADDRESS

[ -x /etc/vnc/xstartup ] && exec /etc/vnc/xstartup
[ -r $HOME/.Xresources ] && xrdb $HOME/.Xresources
xsetroot -solid grey

vncconfig -iconic &
gnome-panel &
gnome-settings-daemon &
metacity &
nautilus &
gnome-terminal &
```

EXAMPLE: VNC SERVER – UBUNTU 18.04

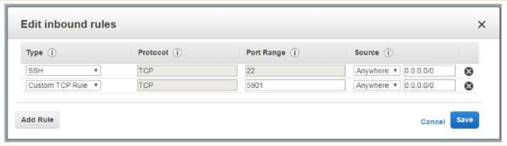
- On the VM:
- Edit config file: `nano ~/.vnc/xstartup`
- Replace contents as below (Ubuntu 18.04):

```
#!/bin/bash
xrdb $HOME/.Xresources
startxfce4 &
```

EXAMPLE: VNC SERVER - 3

- On the VM: reload config by restarting server
- `vncserver -kill :1`
- `vncserver :1`

- Open port 22 & 5901 in EC2 security group:



Edit inbound rules

Type	Protocol	Port Range	Source	
SSH	TCP	22	Anywhere	0.0.0.0/0
Custom TCP Rule	TCP	5901	Anywhere	0.0.0.0/0

EXAMPLE: VNC CLIENT

- On the client (e.g. laptop):
- Create SSH connection to securely forward port 5901 on the EC2 instance to your localhost port 5901
- This way your VNC client doesn't need an SSH key

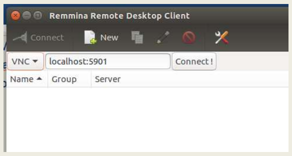
```
ssh –i <ssh-keyfile> -L 5901:127.0.0.1:5901 -N
-f -l <username> <EC2-instance ip_address>
```

- For example:

```
ssh –i mykey.pem -L 5901:127.0.0.1:5901 -N -f -
l ubuntu 52.111.202.44
```

EXAMPLE: VNC CLIENT - 2

- On the client (e.g. laptop):
- Use a VNC Client to connect
- Remmina is provided by default on Ubuntu 16.04
- Can "google" for many others
- Remmina login:
- Chose "VNC" protocol
- Log into "localhost:5901"

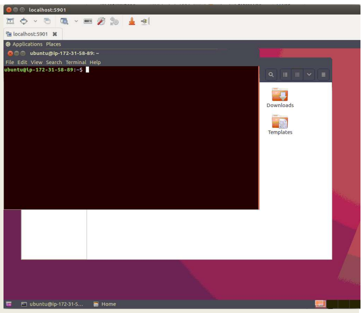


Remmina Remote Desktop Client

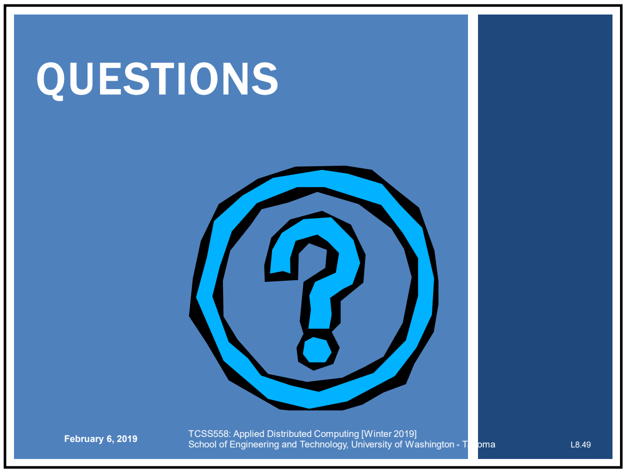
VNC: localhost:5901

REMOTE COMPUTER IN THE CLOUD

- EC2 instance with a GUI. . .!!!



```
ubuntu@ip-172-31-58-89:~$
```



QUESTIONS

February 6, 2019

TCSS558: Applied Distributed Computing [Winter 2019]
School of Engineering and Technology, University of Washington - Tacoma

L8.49