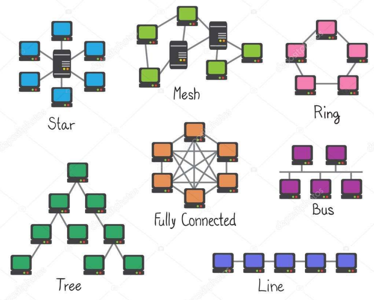
TCSS 558:
APPLIED DISTRIBUTED COMPUTING



Processes:
Threads & Virtualization, Clients & Servers

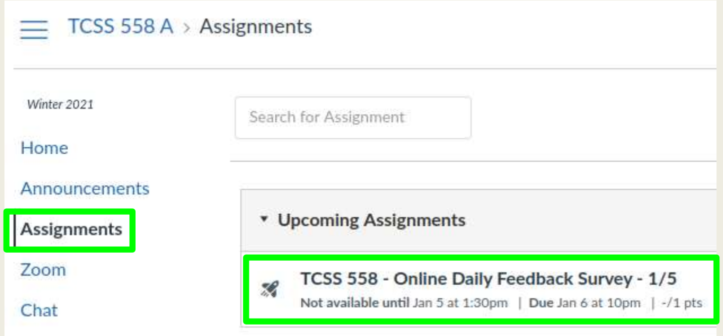
Wes J. Lloyd
School of Engineering
& Technology (SET)
University of Washington - Tacoma

OBJECTIVES – 1/28

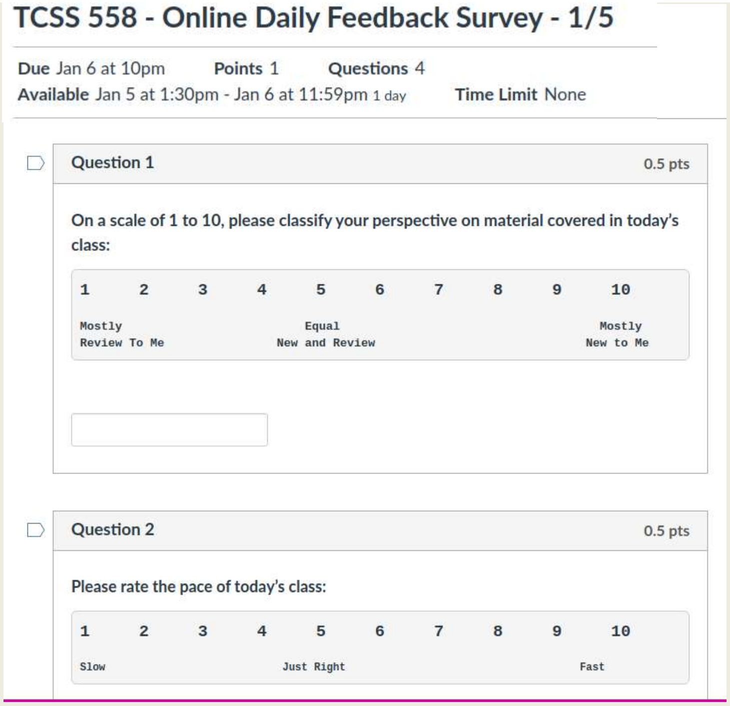
- Questions from 1/26
- Assignment 0: Cloud Computing Infrastructure Tutorial
  - New testFibService.sh script
- Assignment 1: Key/Value Store
- Chapter 3: Processes
  - Chapter 3.1: Threads
    - Context Switches
    - Threading Models
    - Multithreaded clients/servers
  - Chapter 3.2: Virtualization
  - Chapter 3.3: Clients
  - Chapter 3.4: Servers

ONLINE DAILY FEEDBACK SURVEY

- Daily Feedback Quiz in Canvas – Available After Each Class
- Extra credit available for completing surveys ***ON TIME***
- Tuesday surveys: due by ~ Wed @ 10p
- Thursday surveys: due ~ Mon @ 10p



TCSS 558 - Online Daily Feedback Survey - 1/5
Not available until Jan 5 at 1:30pm | Due Jan 6 at 10pm | -/1 pts



TCSS 558 - Online Daily Feedback Survey - 1/5

Due Jan 6 at 10pm **Points** 1 **Questions** 4
Available Jan 5 at 1:30pm - Jan 6 at 11:59pm 1 day **Time Limit** None

Question 1 0.5 pts

On a scale of 1 to 10, please classify your perspective on material covered in today's class:

```
1     2     3     4     5     6     7     8     9     10

Mostly               Equal                       Mostly
Review To Me       New and Review                New to Me
```

Question 2 0.5 pts

Please rate the pace of today's class:

```
1     2     3     4     5     6     7     8     9     10

Slow                 Just Right                  Fast
```

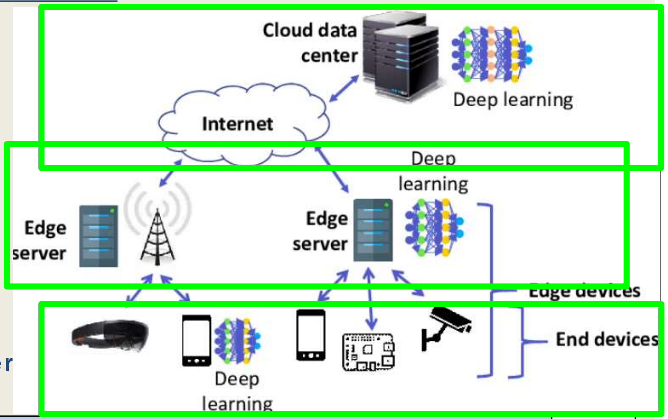
MATERIAL / PACE

- Please classify your perspective on material covered in today's class (22 respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- Average – 7.29 (↑ - *previous 6.73*)

- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- Average – 5.81 (↑ - *previous 5.50*)

FEEDBACK FROM 1/26

- ***The hybrid architecture remains the least clear to me.***
- A **hybrid** architecture combines **more than one** architecture:
- **EDGE COMPUTING EXAMPLE:**
- **End of network:** Unstructured peer-to-peer
- **Edge Server:** Centralized routes to Internet
- **Cloud data center:** Heterogeneous virtual machines: Structured peer-to-peer



FEEDBACK - 2

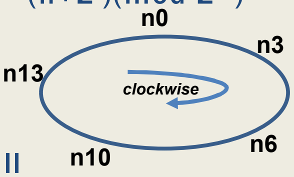
- ***Furthermore, I am wondering about the difference between unstructured and structured peer to peer system.***
- **Structured peer-to-peer:**
  Features deterministic message routes and routing times
  More rigid, changes require reconfiguration
  - Dependable messaging
- **Unstructured peer-to-peer:**
  Message routes need to be discovered
  Topology is constantly changing
  Changes are discovered on-the-fly
  - Common with wireless systems where devices have unreliable networks and power sources
  - Best effort messaging

FEEDBACK - 3

- ***For Dynamic Topology - Chord System, how is the shortest path O(log N)? (N is the number of nodes)***
- Chord provides an alternative to implement a DHT but without a fixed size such as with the four-dimensional hypercube
- Each node keeps a finger table containing m entries
  - m is the number of bits in the hash key
- A query is sent to an arbitrary node
- The node will look up the hash ***k*** in the finger table
- The finger table identifies the node to send the query to
- Nodes in the chord system are responsible for maintaining up-to-date finger tables

HOW TO COMPUTE FINGER TABLE (FT)

- i^{th} entry in ft at peer with id n is ***first node*** >= $(n+2^i)(\text{mod } 2^m)$
- For our example hash has 4 bits (m=4)
- Consider that we have 5 nodes
- Let's compute the finger table for **n3**
- Everytime a node wants to lookup a key it will pass the query to the ***first node*** which is the closest successor or predecessor (going clockwise) of ***k*** in it's finger table

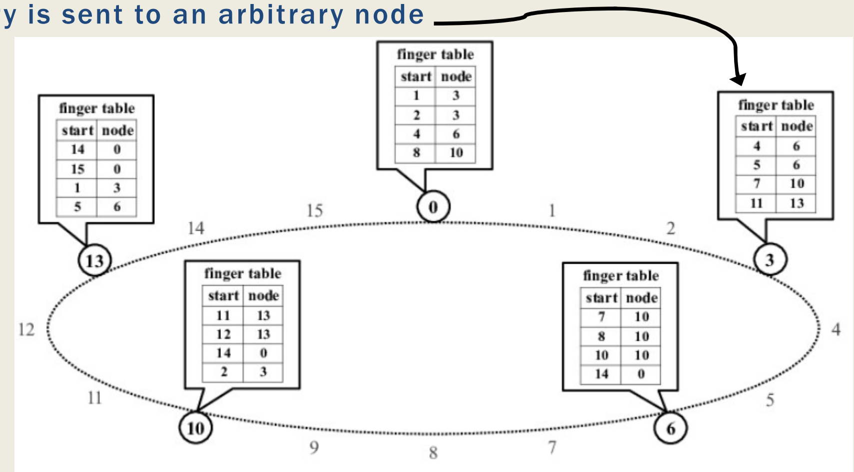


- **N3**

i	ft[i]	
4	**n6**	**$(3+2^0)(\text{mod } 2^4)$ hash i=0**
5	**n6**	**$(3+2^1)(\text{mod } 2^4)$ hash i=1**
7	**n10**	**$(3+2^2)(\text{mod } 2^4)$ hash i=2**
11	**n13**	**$(3+2^3)(\text{mod } 2^4)$ hash i=3**

5-NODE CHORD SYSTEM

- Consider a 5 node Chord system with a 4-bit hash
- A query is sent to an arbitrary node



finger table

start	node
14	0
15	0
1	3
5	6

finger table

start	node
1	3
2	3
4	6
8	10

finger table

start	node
4	6
5	6
7	10
11	13

finger table

start	node
11	13
12	13
14	0
2	3

finger table

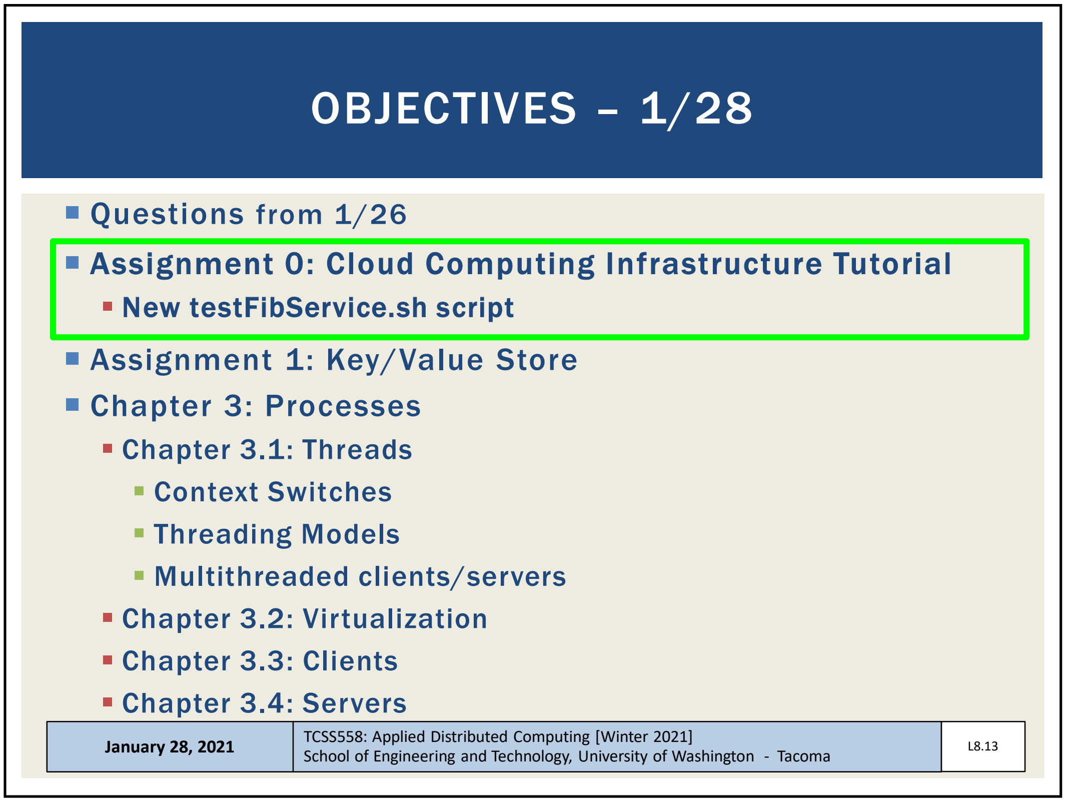
start	node
7	10
8	10
10	10
14	0

TO FIND THE DATA

- To lookup a item with hash key ***k***, the node will pass the query to the closest successor or predecessor of ***k*** in the finger table (the node with the highest ID in the circle whose ID is smaller than ***k***)
- If ***k*** =8 and the query first goes to node n3
- Query is passed to node n10
- Data each node is responsible for storing in this 5-node chord:
  n0 k={14,15,0}
  n3 k= {1,2,3}
  n6 k= {4,5,6}
  n10 k= {7,8,9,10}
  n13 k= {11,12,13}
- Path to data n3 → n10 (data found) – 1 hop ≈ O(log n)

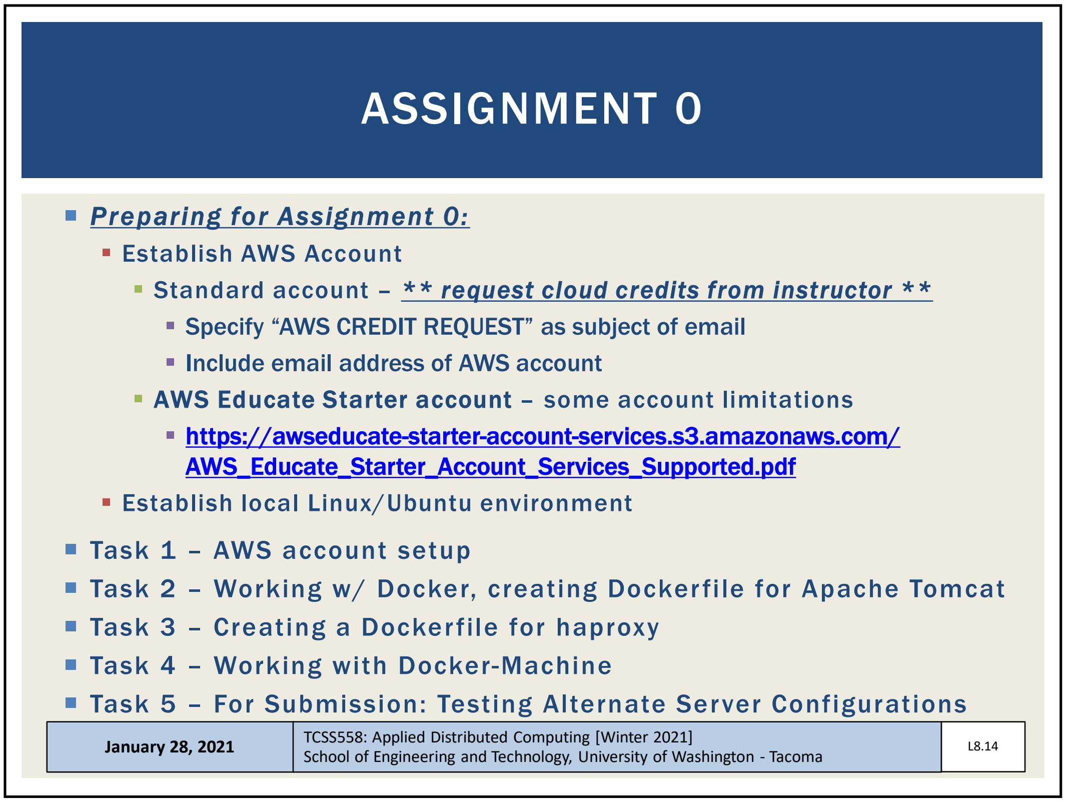
The Instructor is unavailable for next Tuesday's class due to a community service commitment with the National Science Foundation. What is your preference for next Tuesday's class?

Start the presentation to see live content. For screen share software, share the entire screen. Get help at **pollev.com/app**



OBJECTIVES – 1/28

- Questions from 1/26
- Assignment 0: Cloud Computing Infrastructure Tutorial
  - New testFibService.sh script
- Assignment 1: Key/Value Store
- Chapter 3: Processes
  - Chapter 3.1: Threads
    - Context Switches
    - Threading Models
    - Multithreaded clients/servers
  - Chapter 3.2: Virtualization
  - Chapter 3.3: Clients
  - Chapter 3.4: Servers



ASSIGNMENT 0

- *Preparing for Assignment 0:*
  - Establish AWS Account
    - Standard account – *** request cloud credits from instructor ***
      - Specify “AWS CREDIT REQUEST” as subject of email
      - Include email address of AWS account
    - **AWS Educate Starter account** – some account limitations
      - https://awseducate-starter-account-services.s3.amazonaws.com/AWS_Educate_Starter_Account_Services_Supported.pdf
  - Establish local Linux/Ubuntu environment
- Task 1 – AWS account setup
- Task 2 – Working w/ Docker, creating Dockerfile for Apache Tomcat
- Task 3 – Creating a Dockerfile for haproxy
- Task 4 – Working with Docker-Machine
- Task 5 – For Submission: Testing Alternate Server Configurations

TESTING CONNECTIVITY TO SERVER

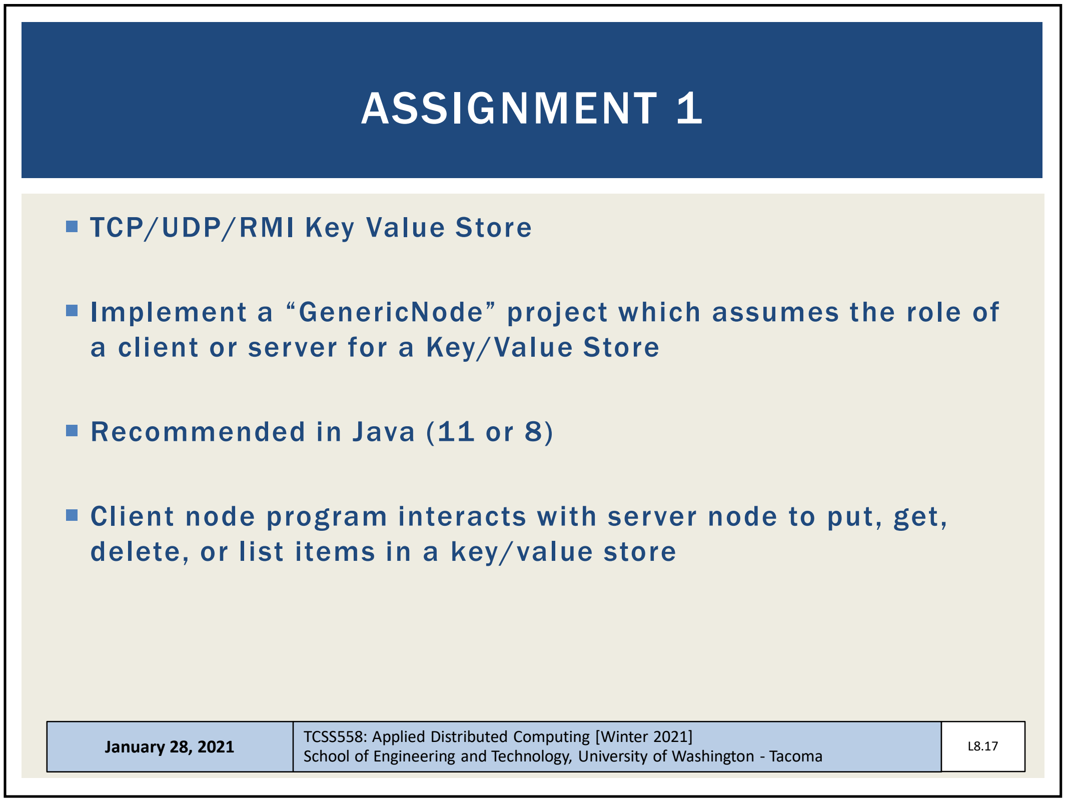
- testFibPar.sh script is a parallel test script
- Orchestrates multiple threads on client to invoke server multiple times in parallel
- To simplify coordinate of parallel service calls in BASH, testFibPar.sh script ignores errors !!!

- To help test client-to-server connectivity, have created a new testFibService.sh script

- TEST 1: Network layer
  - Ping (ICMP)
- TEST 2: Transport layer
  - TCP: telnet (TCP Port 8080) – security group (firewall) test
- TEST 3: Application layer
  - HTTP REST – web service test

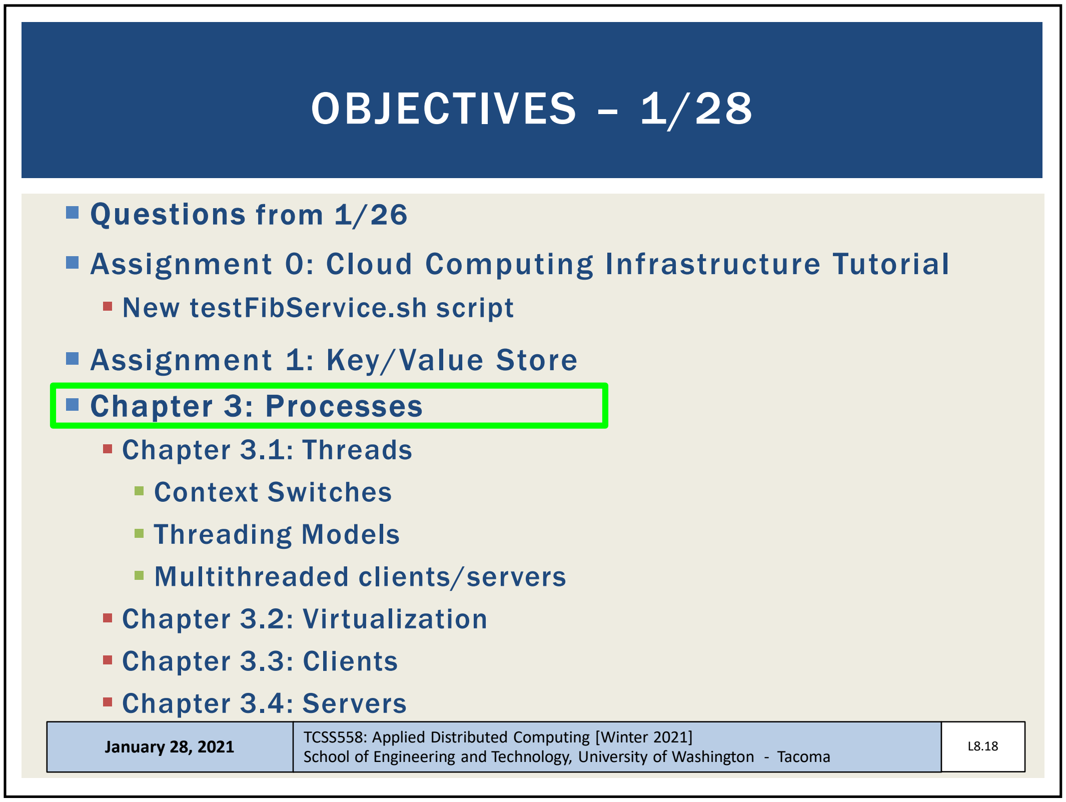
OBJECTIVES – 1/28

- Questions from 1/26
- Assignment 0: Cloud Computing Infrastructure Tutorial
  - New testFibService.sh script
- Assignment 1: Key/Value Store
- Chapter 3: Processes
  - Chapter 3.1: Threads
    - Context Switches
    - Threading Models
    - Multithreaded clients/servers
  - Chapter 3.2: Virtualization
  - Chapter 3.3: Clients
  - Chapter 3.4: Servers



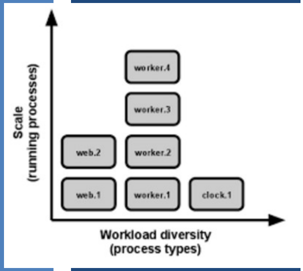
ASSIGNMENT 1

- TCP/UDP/RMI Key Value Store
- Implement a "GenericNode" project which assumes the role of a client or server for a Key/Value Store
- Recommended in Java (11 or 8)
- Client node program interacts with server node to put, get, delete, or list items in a key/value store



OBJECTIVES – 1/28

- Questions from 1/26
- Assignment 0: Cloud Computing Infrastructure Tutorial
  - New testFibService.sh script
- Assignment 1: Key/Value Store
- Chapter 3: Processes
  - Chapter 3.1: Threads
    - Context Switches
    - Threading Models
    - Multithreaded clients/servers
  - Chapter 3.2: Virtualization
  - Chapter 3.3: Clients
  - Chapter 3.4: Servers



CH. 3: PROCESSES

CH. 3.1: THREADS

CHAPTER 3

- Chapter 3 titled "processes"
- Covers variety of distributed system implementation details
- "Grab bag" of topics

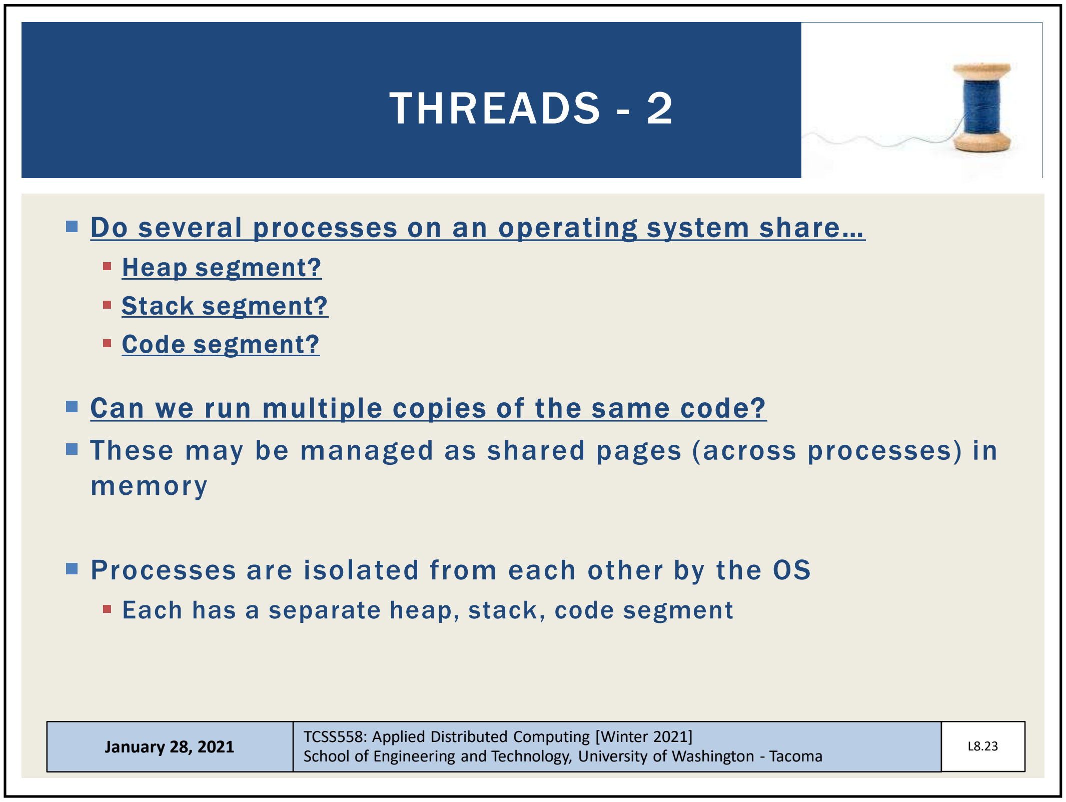
- Processes/threads
- Virtualization
- Clients
- Servers
- Code migration

OBJECTIVES – 1/28

- Questions from 1/26
- Assignment 0: Cloud Computing Infrastructure Tutorial
  - New testFibService.sh script
- Assignment 1: Key/Value Store
- Chapter 3: Processes
  - Chapter 3.1: Threads
    - Context Switches
    - Threading Models
    - Multithreaded clients/servers
  - Chapter 3.2: Virtualization
  - Chapter 3.3: Clients
  - Chapter 3.4: Servers

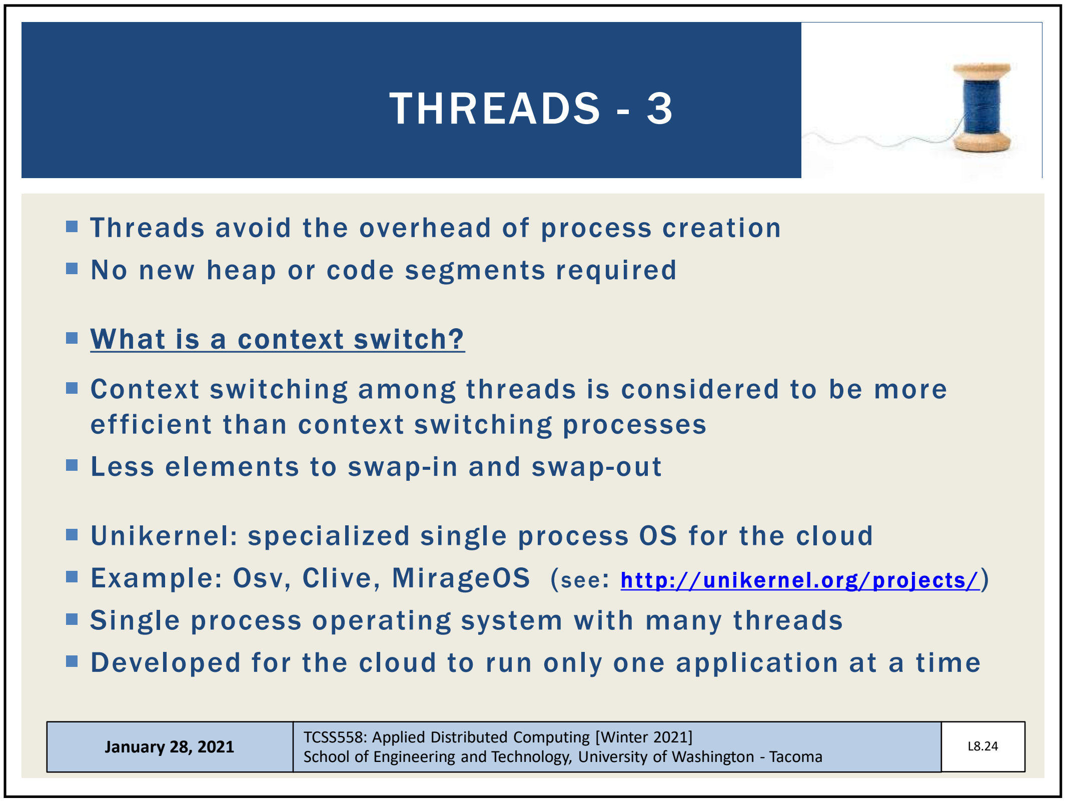
CH. 3.1 - THREADS

- 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

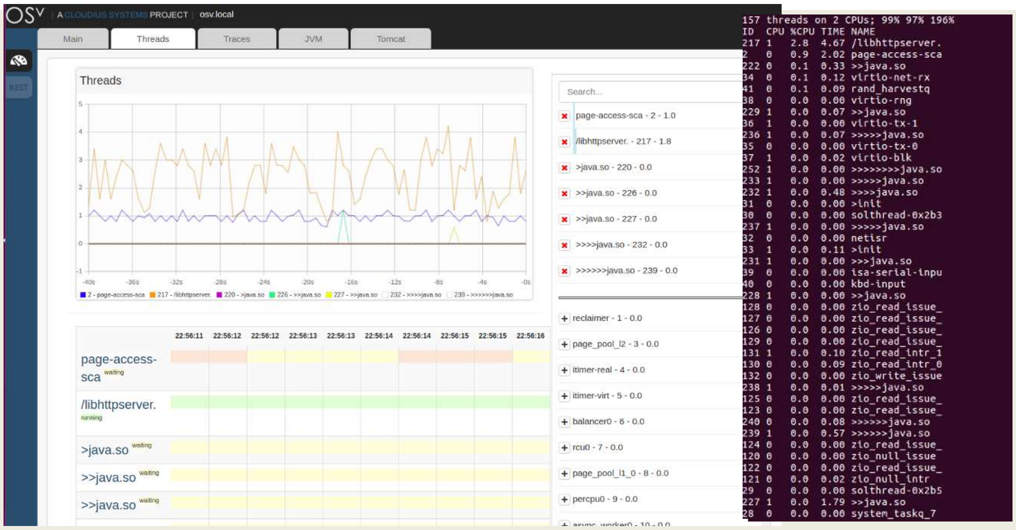
- Do several processes on an operating system share…
  - Heap segment?
  - Stack segment?
  - Code segment?
- Can we run multiple copies of the same code?
- These may be managed as shared pages (across processes) in memory
- Processes are isolated from each other by the OS
  - Each has a separate heap,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

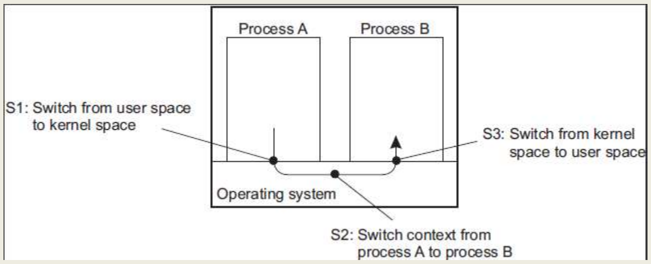
1. Supports interaction (UI) activity with user
2. Updates spreadsheet calculations in parallel
3. Continually backs up spreadsheet changes to disk

- 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



OBJECTIVES – 1/28

- Questions from 1/26
- Assignment 0: Cloud Computing Infrastructure Tutorial
    - New testFibService.sh script
- Assignment 1: Key/Value Store
- Chapter 3: Processes
    - Chapter 3.1: Threads
        - **Context Switches**
        - Threading Models
        - Multithreaded clients/servers
    - Chapter 3.2: Virtualization
    - Chapter 3.3: Clients
    - Chapter 3.4: Servers

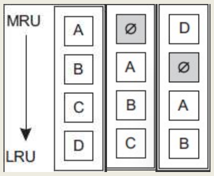
CONTEXT SWITCHING

- 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*

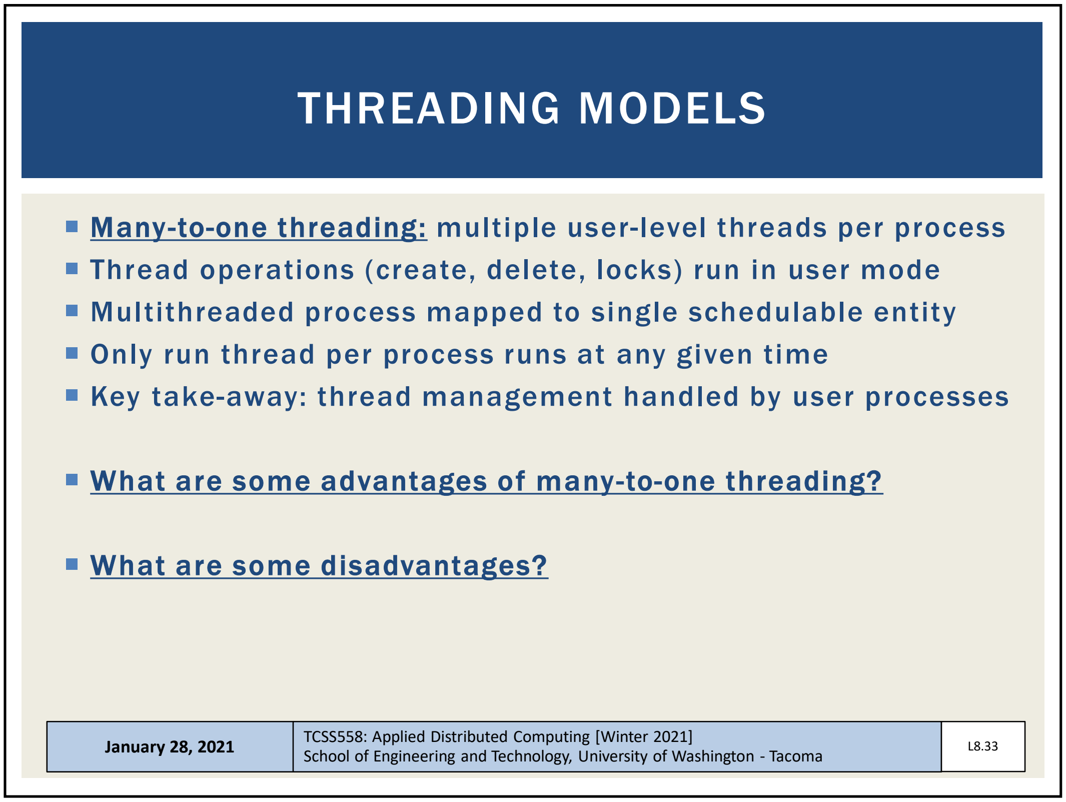
CONTEXT SWITCH – CACHE PERTURBATION

- 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**"



OBJECTIVES – 1/28

- Questions from 1/26
- Assignment 0: Cloud Computing Infrastructure Tutorial
  - New testFibService.sh script
- Assignment 1: Key/Value Store
- Chapter 3: Processes
  - Chapter 3.1: Threads
    - Context Switches
    - Threading Models
    - Multithreaded clients/servers
  - Chapter 3.2: Virtualization
  - Chapter 3.3: Clients
  - Chapter 3.4: Servers

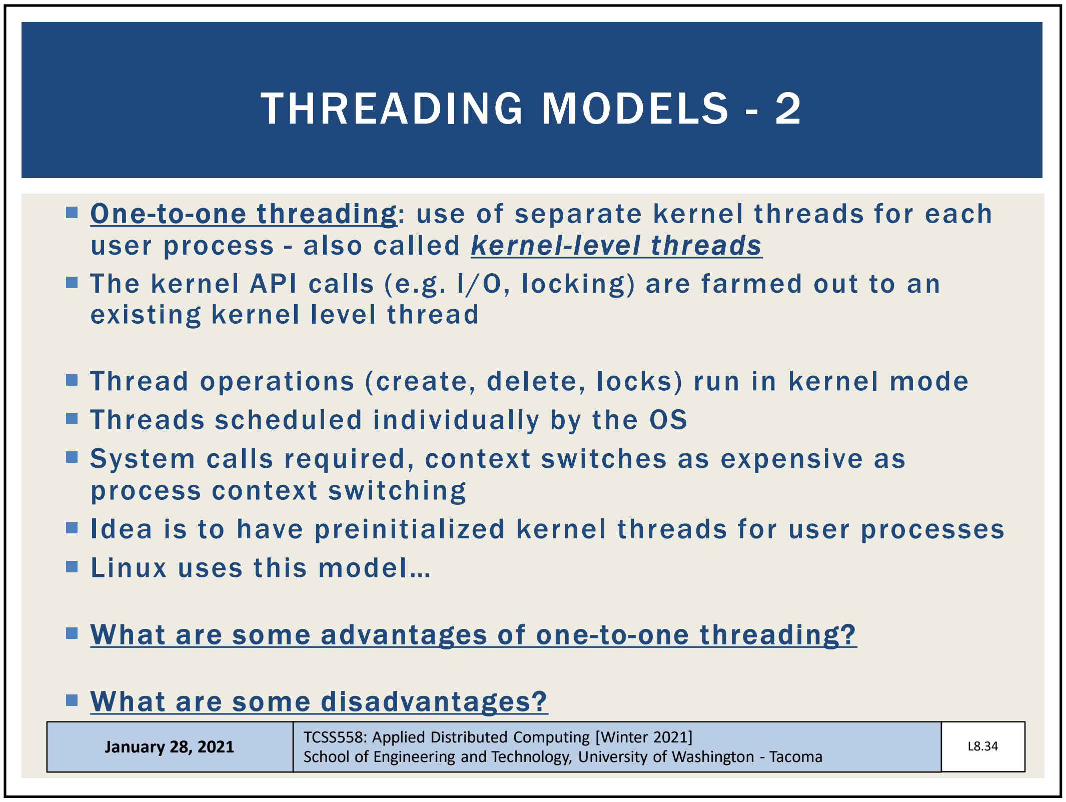


THREADING MODELS

- **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?**

WE WILL RETURN AT 2:46PM

OBJECTIVES – 1/28

- Questions from 1/26
- Assignment 0: Cloud Computing Infrastructure Tutorial
  - New testFibService.sh script
- Assignment 1: Key/Value Store
- Chapter 3: Processes
  - Chapter 3.1: Threads
    - Context Switches
    - Threading Models
    - Multithreaded clients/servers
  - Chapter 3.2: Virtualization
  - Chapter 3.3: Clients
  - Chapter 3.4: Servers

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

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)

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

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

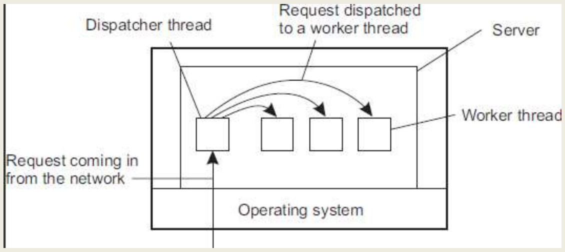
- Metric – measures degree of parallelism realized by running system, by calculating average utilization:

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

- 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**

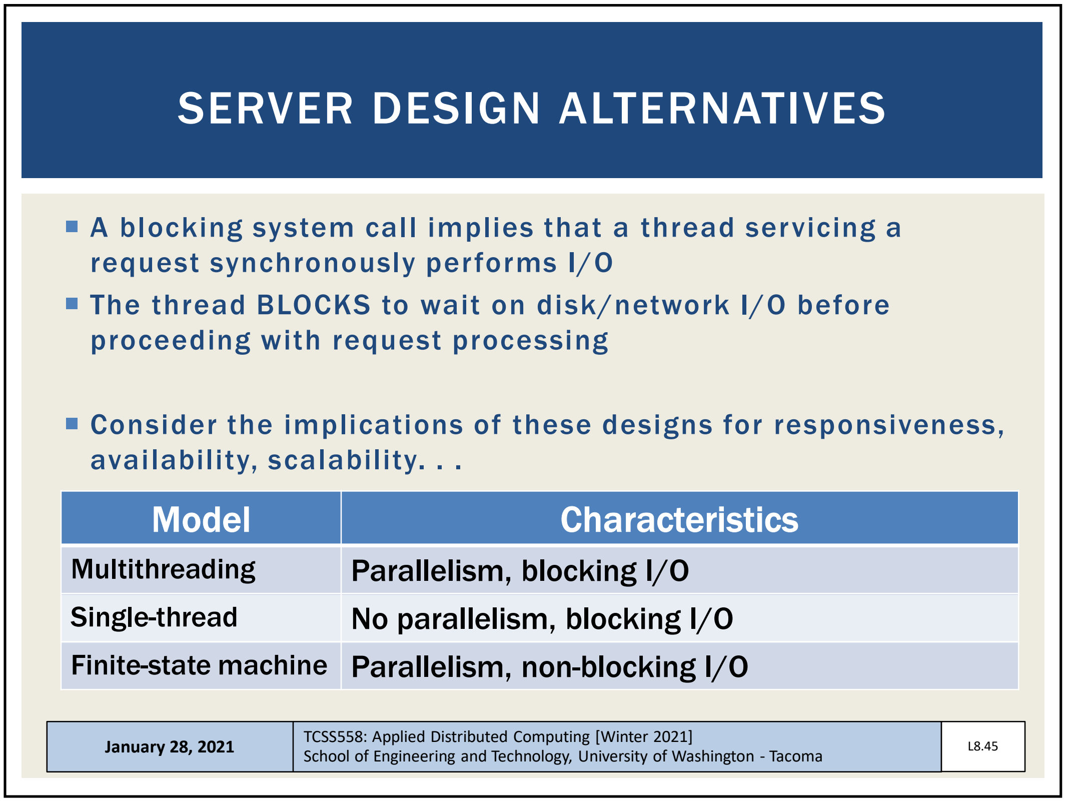
MULTITHREADED SERVERS

- 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



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

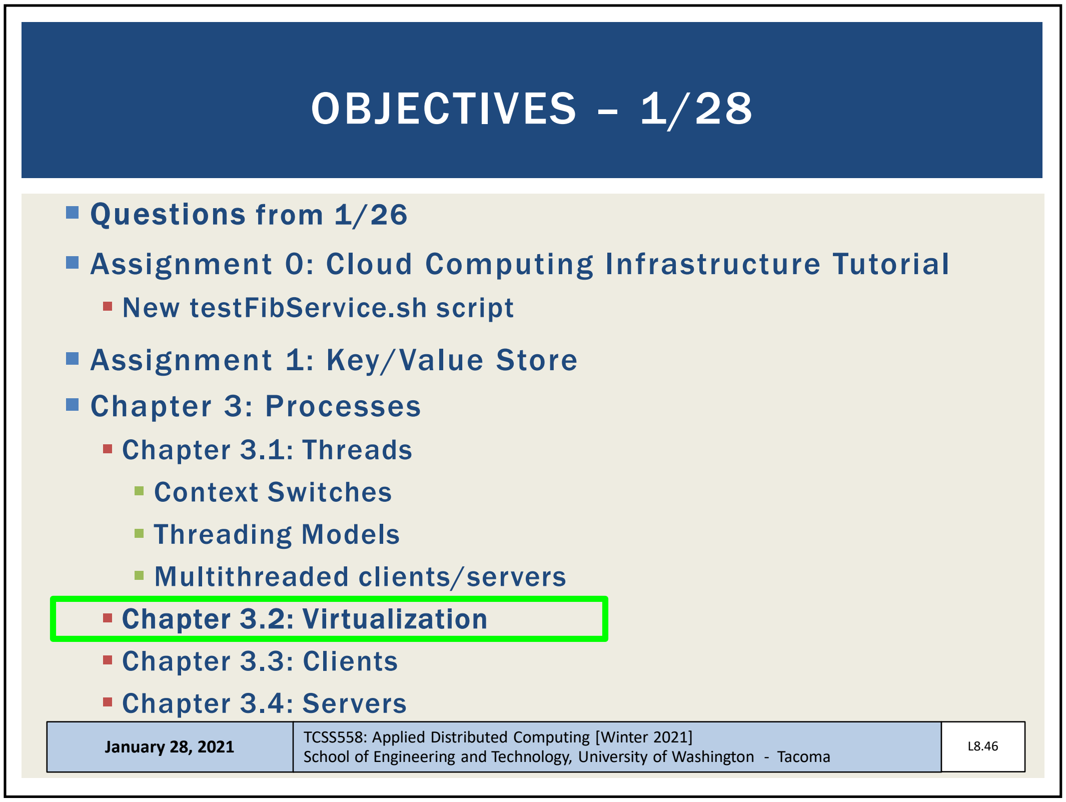


SERVER DESIGN ALTERNATIVES

- 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

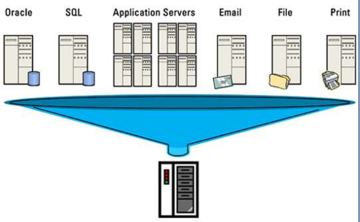
- Consider the implications of these designs for responsiveness, availability, scalability. . .

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



OBJECTIVES – 1/28

- Questions from 1/26
- Assignment 0: Cloud Computing Infrastructure Tutorial
  - New testFibService.sh script
- Assignment 1: Key/Value Store
- Chapter 3: Processes
  - Chapter 3.1: Threads
    - Context Switches
    - Threading Models
    - Multithreaded clients/servers
  - Chapter 3.2: Virtualization
  - Chapter 3.3: Clients
  - Chapter 3.4: Servers

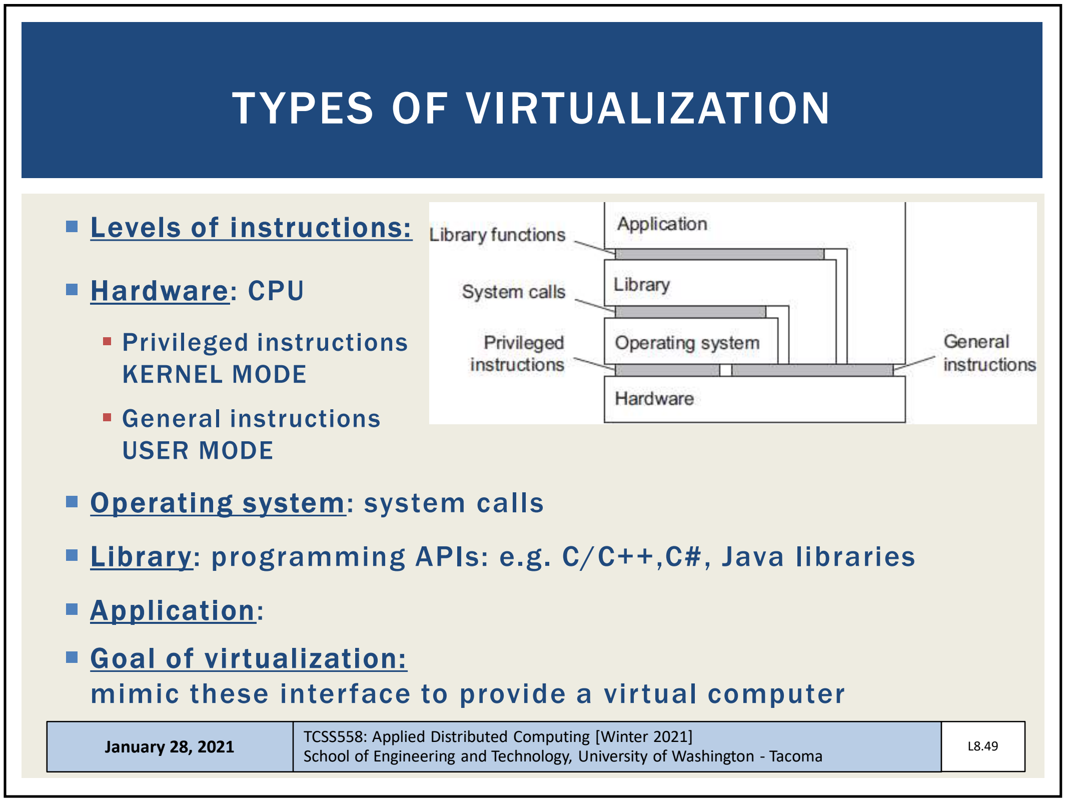


CH. 3.2: VIRTUALIZATION

VIRTUALIZATION

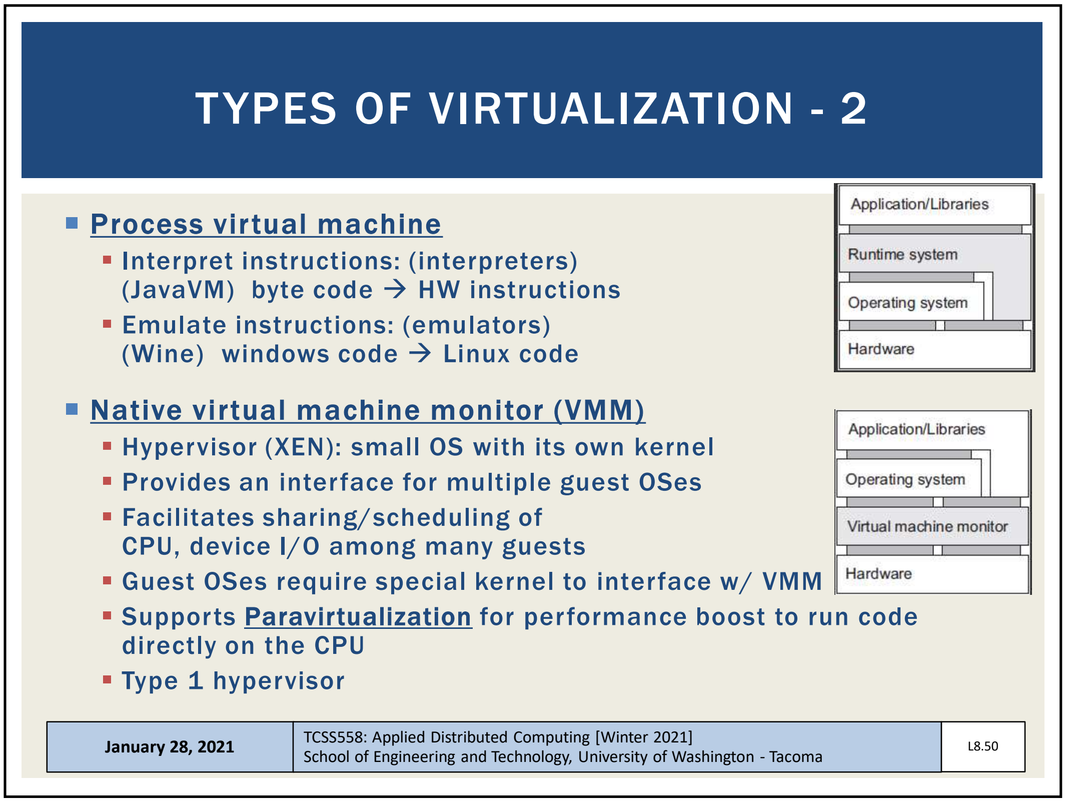
- Initially introduced in the 1970s on IBM mainframe computers
- Legacy operating systems run in mainframe-based VMs
- Legacy software could be sustained by virtualizing legacy OSes
- 1970s virtualization went away as desktop/rack-based hardware became inexpensive

- 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)



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

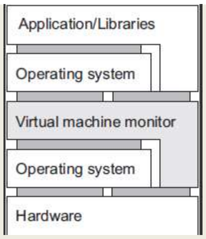


TYPES OF VIRTUALIZATION - 2

- **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

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

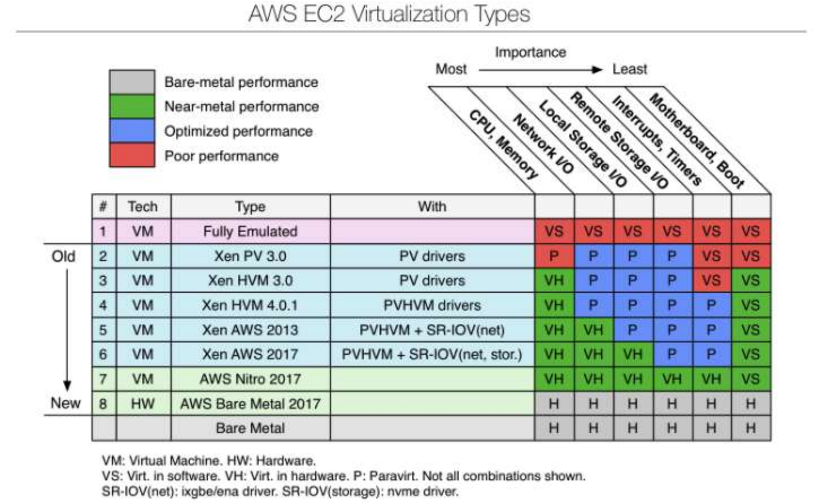
From http://www.brendangregg.com/blog/2017-11-29/aws-ec2-virtualization-2017.html

VS:
Virtualization
In software

P:
Paravirtual

VH:
Virtualization
In Hardware

H:
Hardware



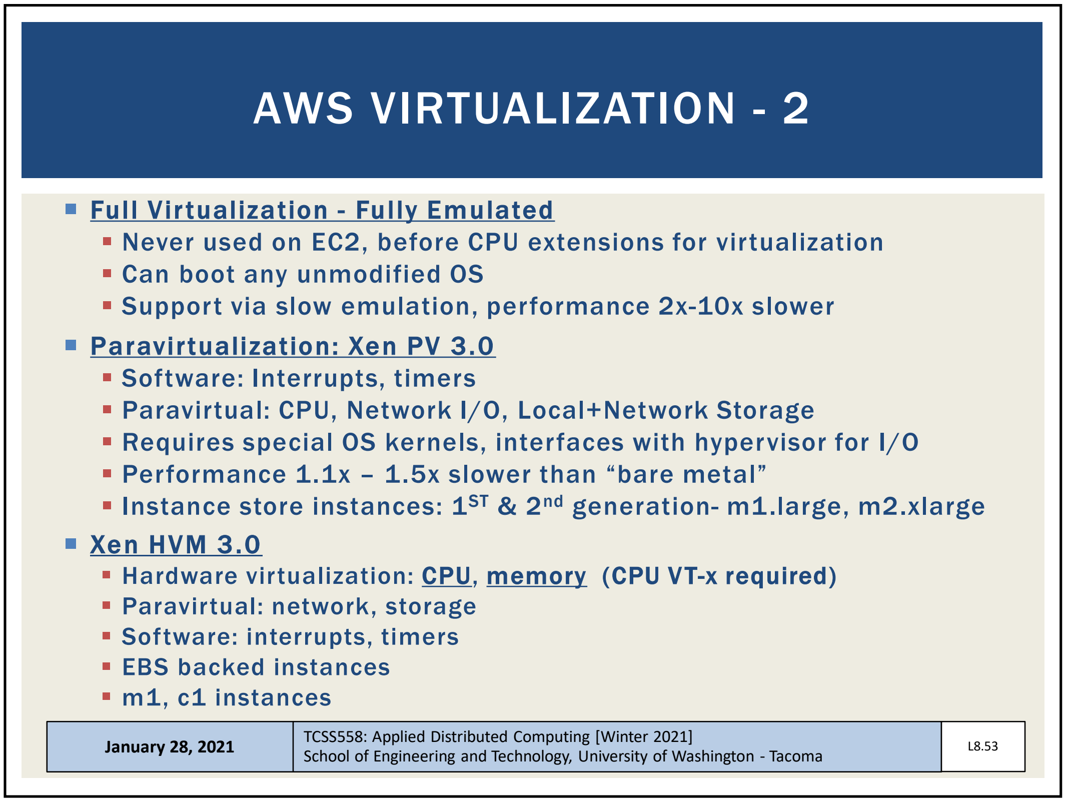
AWS EC2 Virtualization Types

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

Importance: Most → Least

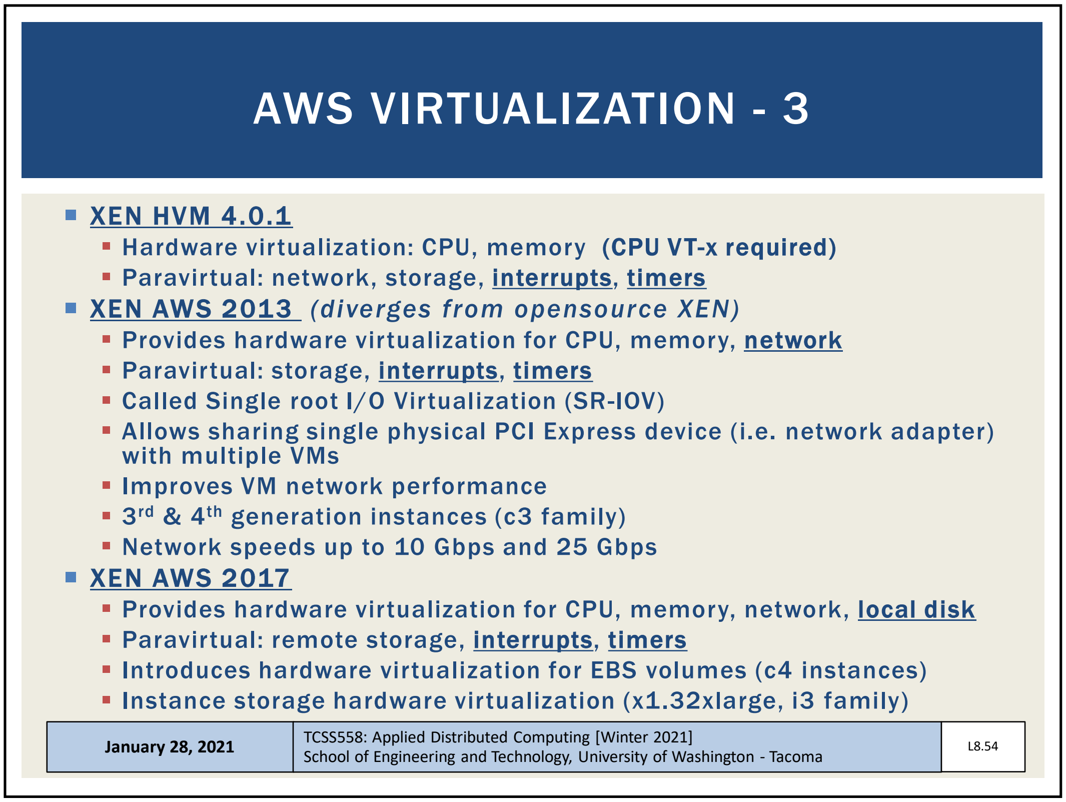
	#	Tech	Type	With	CPU, Memory	Network I/O	Local Storage I/O	Remote Storage I/O	Interrupts, Timers	Motherboard, Boot
	1	VM	Fully Emulated		VS	VS	VS	VS	VS	VS
Old	2	VM	Xen PV 3.0	PV drivers	P	P	P	P	VS	VS
	3	VM	Xen HVM 3.0	PV drivers	VH	P	P	P	VS	VS
	4	VM	Xen HVM 4.0.1	PVHVM drivers	VH	P	P	P	P	VS
	5	VM	Xen 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
New	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.



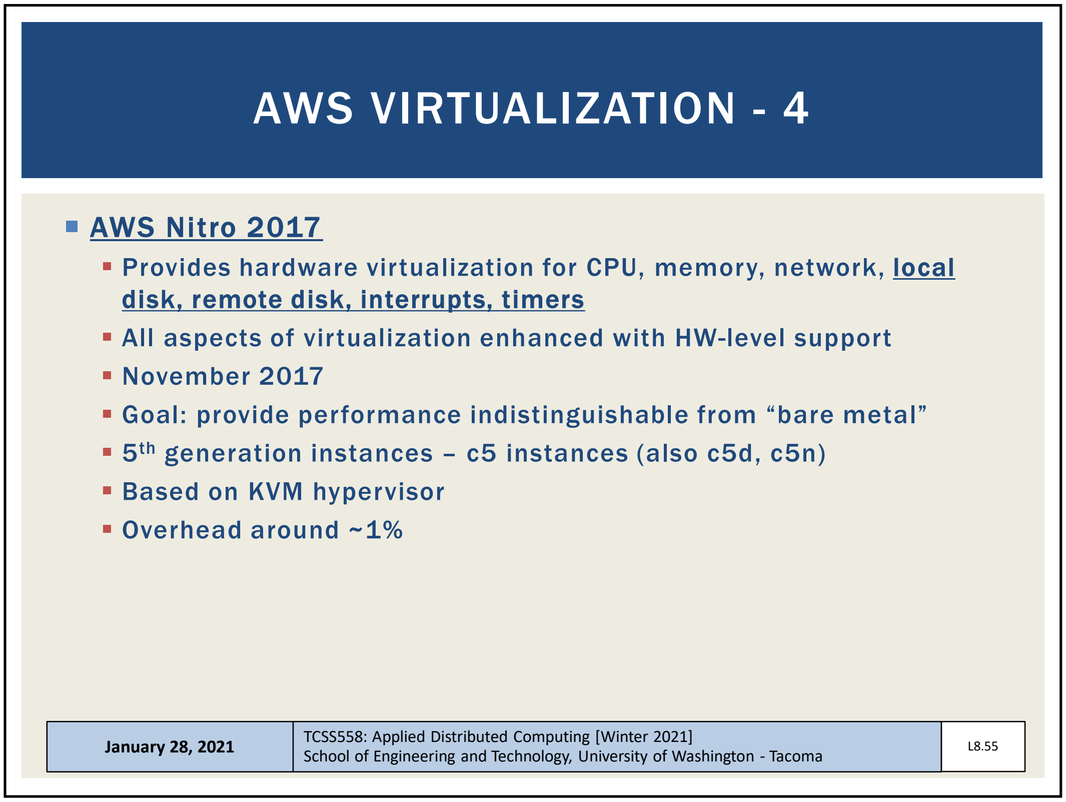
AWS VIRTUALIZATION - 2

- 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



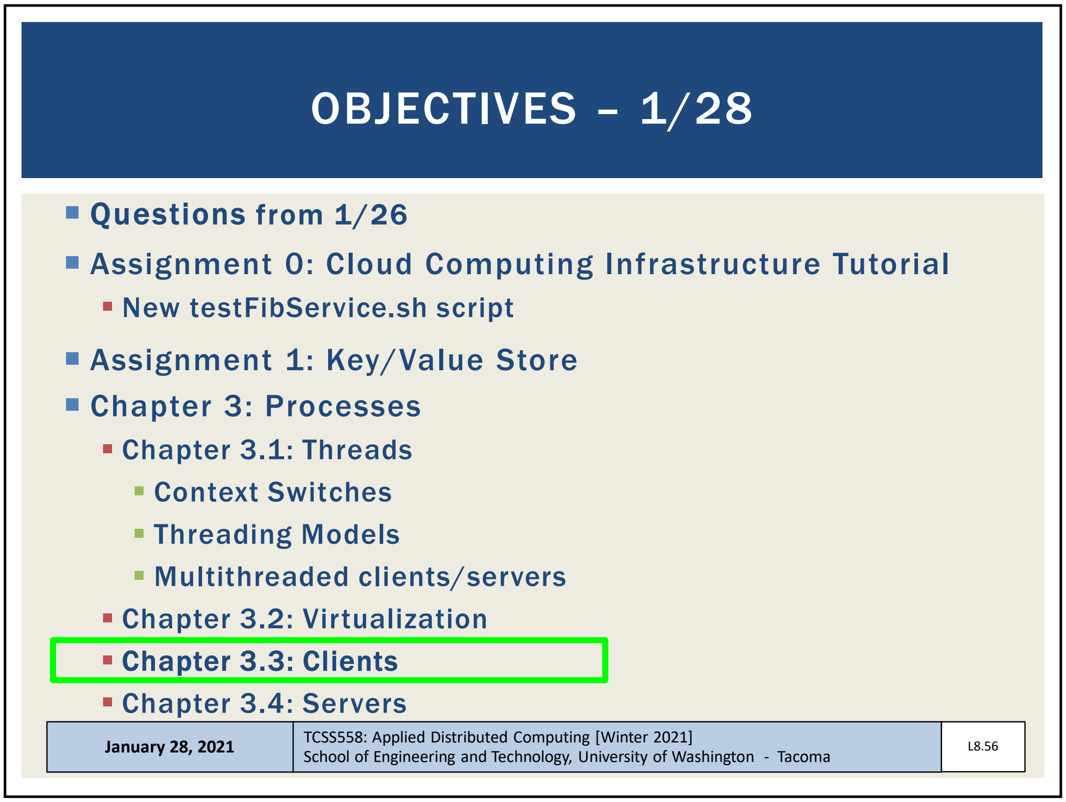
AWS VIRTUALIZATION - 3

- 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%



OBJECTIVES – 1/28

- Questions from 1/26
- Assignment 0: Cloud Computing Infrastructure Tutorial
  - New testFibService.sh script
- Assignment 1: Key/Value Store
- Chapter 3: Processes
  - Chapter 3.1: Threads
    - Context Switches
    - Threading Models
    - Multithreaded clients/servers
  - Chapter 3.2: Virtualization
  - Chapter 3.3: Clients
  - Chapter 3.4: Servers

CH. 3.3: CLIENTS

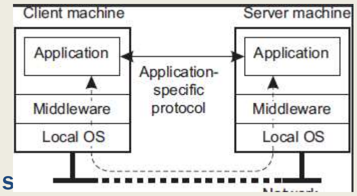
TYPES OF CLIENTS

- Thick clients
  - Web browsers
    - Client-side scripting
  - Mobile apps
  - Multi-tier MVC apps

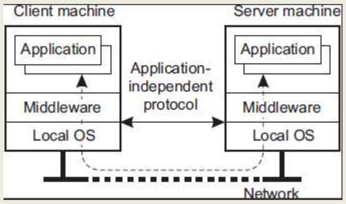
- Thin clients
  - Remote desktops/GUIs (very thin)

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

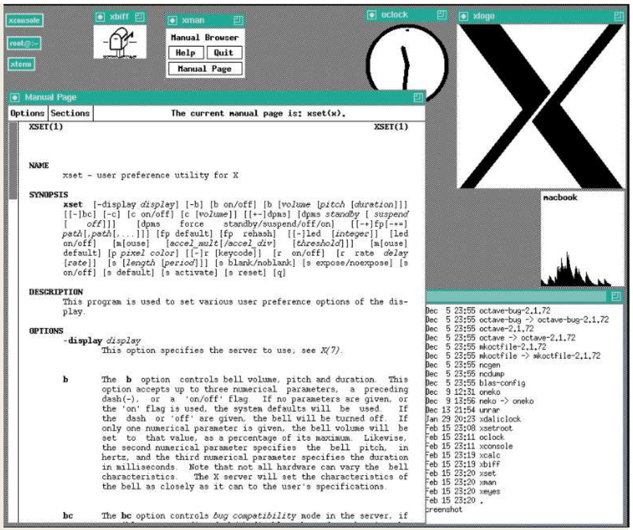


X WINDOWS

- 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

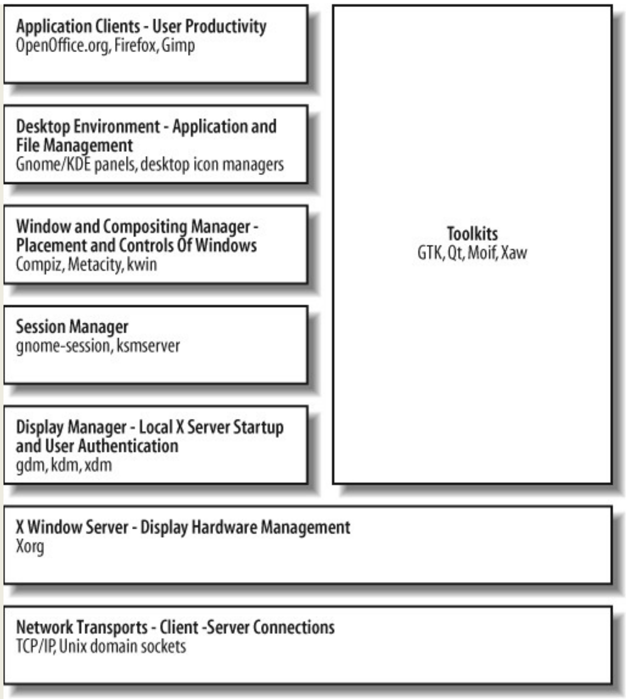


The current manual page is: xset(x).

```
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

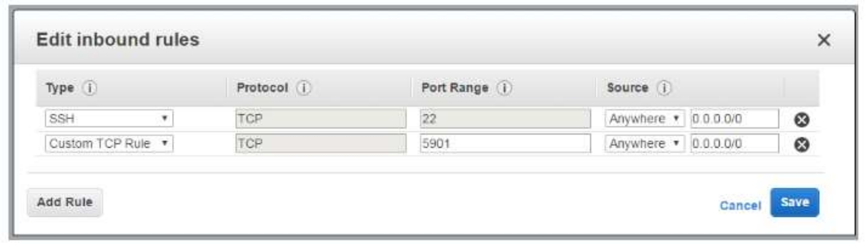
- On the VM:
- Edit config file: `nano ~/.vnc/xstartup`
- Replace contents as below (Ubuntu 18.04):

```
#!/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:



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

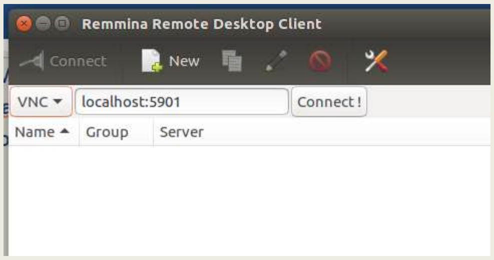
```
ssh -i <ssh-keyfile> -L 5901:127.0.0.1:5901 -N
-f -l <username> <EC2-instance ip_address>
```

- 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

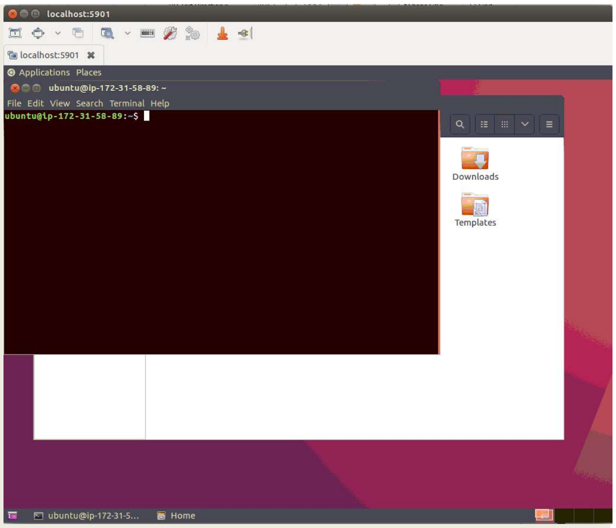
- On the client (e.g. laptop):
- Use a VNC Client to connect
- Remmina is provided by default on Ubuntu 16.04
- Can "google" for many others
- Remmina login:
- Chose "VNC" protocol
- Log into "localhost:5901"



localhost:5901

REMOTE COMPUTER IN THE CLOUD

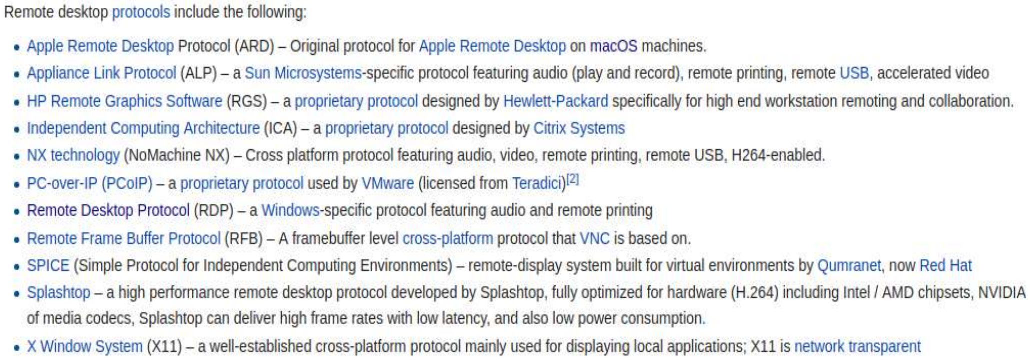
- EC2 instance with a GUI. . .!!!



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

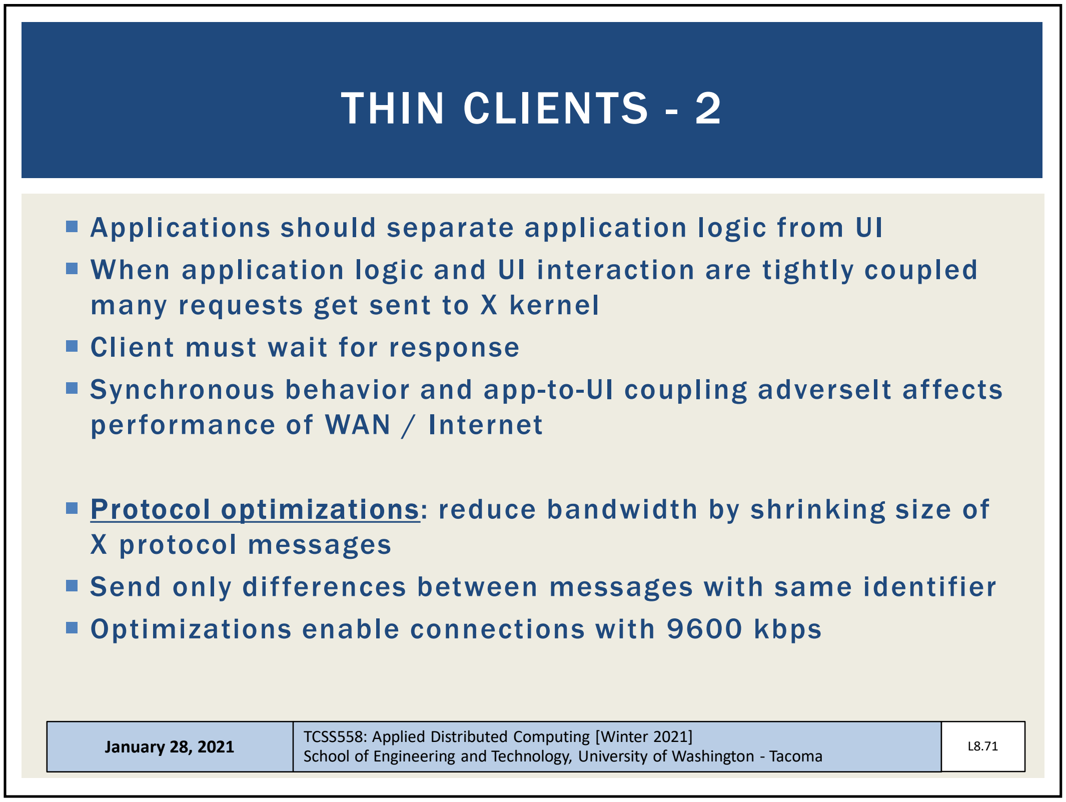
THIN CLIENTS

- Thin clients
  - X windows protocol
  - A variety of other remote desktop protocols exist:



Remote desktop protocols include the following:

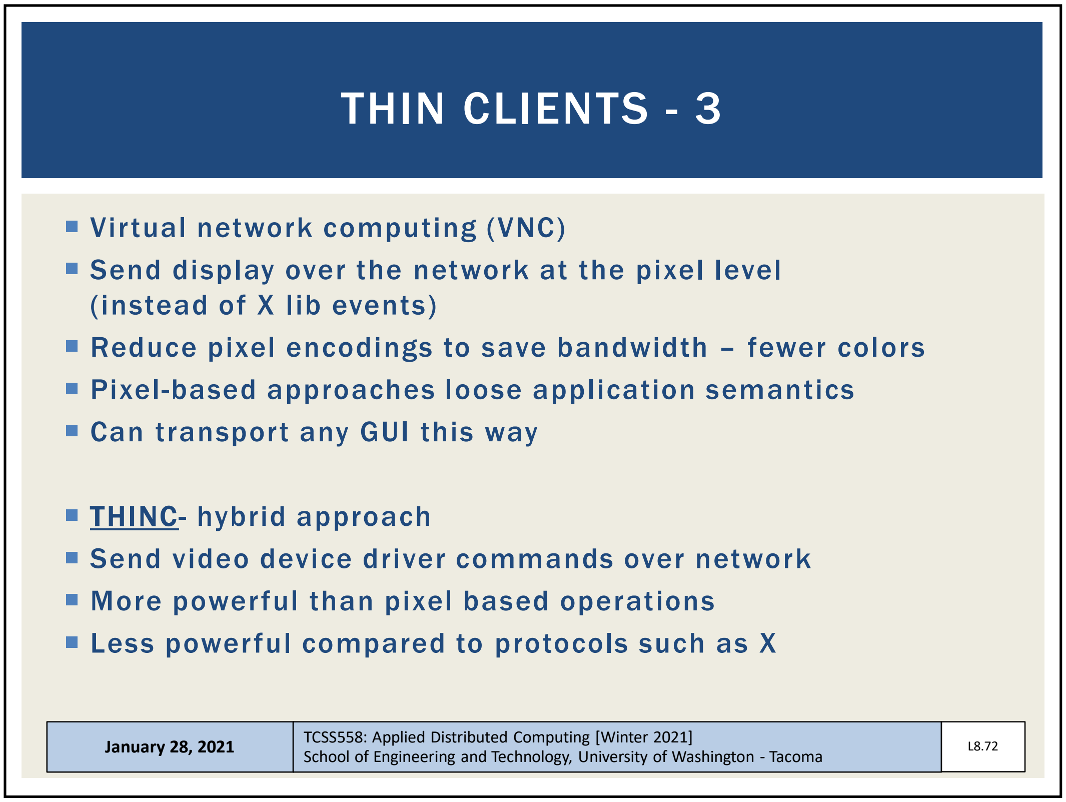
- Apple Remote Desktop Protocol (ARD) – Original protocol for Apple Remote Desktop on macOS machines.
- Appliance Link Protocol (ALP) – a Sun Microsystems-specific protocol featuring audio (play and record), remote printing, remote USB, accelerated video
- HP Remote Graphics Software (RGS) – a proprietary protocol designed by Hewlett-Packard specifically for high end workstation remoting and collaboration.
- Independent Computing Architecture (ICA) – a proprietary protocol designed by Citrix Systems
- NX technology (NoMachine NX) – Cross platform protocol featuring audio, video, remote printing, remote USB, H264-enabled.
- PC-over-IP (PCoIP) – a proprietary protocol used by VMware (licensed from Teradici)[2]
- Remote Desktop Protocol (RDP) – a Windows-specific protocol featuring audio and remote printing
- Remote Frame Buffer Protocol (RFB) – A framebuffer level cross-platform protocol that VNC is based on.
- SPICE (Simple Protocol for Independent Computing Environments) – remote-display system built for virtual environments by Qumranet, now Red Hat
- Splashtop – a high performance remote desktop protocol developed by Splashtop, fully optimized for hardware (H.264) including Intel / AMD chipsets, NVIDIA of media codecs, Splashtop can deliver high frame rates with low latency, and also low power consumption.
- X Window System (X11) – a well-established cross-platform protocol mainly used for displaying local applications; X11 is network transparent



THIN CLIENTS - 2

- Applications should separate application logic from UI
- When application logic and UI interaction are tightly coupled many requests get sent to X kernel
- Client must wait for response
- Synchronous behavior and app-to-UI coupling adverselt affects performance of WAN / Internet

- **Protocol optimizations**: reduce bandwidth by shrinking size of X protocol messages
- Send only differences between messages with same identifier
- Optimizations enable connections with 9600 kbps



THIN CLIENTS - 3

- Virtual network computing (VNC)
- Send display over the network at the pixel level (instead of X lib events)
- Reduce pixel encodings to save bandwidth – fewer colors
- Pixel-based approaches loose application semantics
- Can transport any GUI this way

- **THINC**- hybrid approach
- Send video device driver commands over network
- More powerful than pixel based operations
- Less powerful compared to protocols such as X

TRADEOFFS: ABSTRACTION OF REMOTE DISPLAY PROTOCOLS

- Tradeoff space: abstraction level of remote display protocols

Pixel-level
VNC

Graphics lib
X11

TRADEOFFS: ABSTRACTION OF REMOTE DISPLAY PROTOCOLS

- Tradeoff space: abstraction level of remote display protocols

Pixel-level
VNC

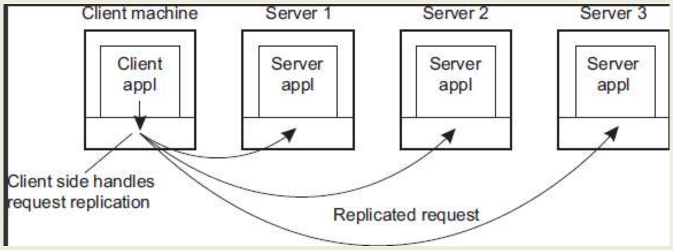
- Generic – no app context
- Graphics data
- Higher network bandwidth
- Fewer colors
- Utilize graphics compression
- More network traffic

Graphics lib
X11

- Application context is available
- UI data/operations
- Lower network bandwidth
- More colors

CLIENT ROLES IN PROVIDING DISTRIBUTION TRANSPARENCY

- Clients help enable distribution transparency of servers
- Replication transparency
  - Client aggregates responses from multiple servers
  - Only the client knows of replicas



CLIENT ROLES IN PROVIDING DISTRIBUTION TRANSPARENCY - 2

- Location/relocation/migration transparency
  - Harness convenient naming system to allow client to infer new locations
  - Server inform client of moves / Client reconnects to new endpoint
  - Client hides network address of server, and reconnects as needed
  - May involve temporary loss in performance
- Replication transparency
  - Client aggregates responses from multiple servers
- Failure transparency
  - Client retries, or maps to another server, or uses cached data
- Concurrency transparency
  - Transaction servers abstract coordination of multithreading

OBJECTIVES – 1/28

- Questions from 1/26
- Assignment 0: Cloud Computing Infrastructure Tutorial
  - New testFibService.sh script
- Assignment 1: Key/Value Store
- Chapter 3: Processes
  - Chapter 3.1: Threads
    - Context Switches
    - Threading Models
    - Multithreaded clients/servers
  - Chapter 3.2: Virtualization
  - Chapter 3.3: Clients
  - Chapter 3.4: Servers

CH. 3.4: SERVERS

SERVERS

- Cloud & Distributed Systems – rely on **Linux**
- http://www.zdnet.com/article/it-runs-on-the-cloud-and-the-cloud-runs-on-linux-any-questions/
- IT is moving to the cloud. And, what powers the cloud?
    - **Linux**
- Uptime Institute survey - 1,000 IT executives (2016)
    - 50% of IT executives – plan to migrate majority of IT workloads to off-premise to cloud or colocation sites
    - 23% expect the shift in 2017, 70% by 2020…
- Docker on Windows / Mac OS X
    - Based on **Linux**
    - Mac: Hyperkit Linux VM
    - Windows: Hyper-V Linux VM

SERVERS - 2

- Servers implement a specific service for a collection of clients
- Servers wait for incoming requests, and respond accordingly

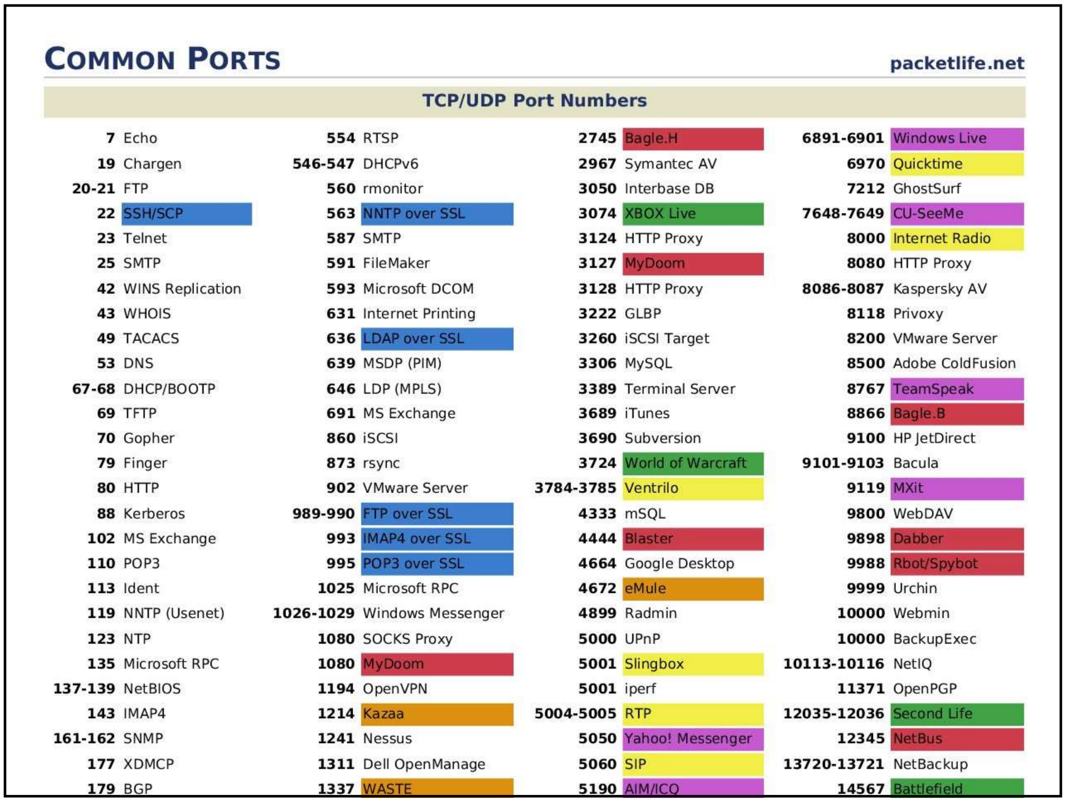
- **Server types**
- **Iterative**: immediately handle client requests
- **Concurrent**: Pass client request to separate thread

- Multithreaded servers are concurrent servers
    - E.g. Apache Tomcat

- ***Alternative***: fork a new process for each incoming request
- ***Hybrid***: mix the use of multiple processes with thread pools

END POINTS

- Clients connect to servers via:
  IP Address and **Port Number**
- How do ports get assigned?
  - Many protocols support “default” port numbers
  - Client must find IP address(es) of servers
  - A single server often hosts multiple end points (servers/services)
  - When designing new TCP client/servers must be careful not to repurpose ports already commonly used by others



COMMON PORTS

packetlife.net

TCP/UDP Port Numbers

7	Echo	**554**	RTSP	**2745**	🟥 Bagle.H	**6891-6901**	🟪 Windows Live
19	Chargen	**546-547**	DHCPv6	**2967**	Symantec AV	**6970**	🟨 Quicktime
20-21	FTP	**560**	rmonitor	**3050**	Interbase DB	**7212**	GhostSurf
22	🟦 SSH/SCP	**563**	🟦 NNTP over SSL	**3074**	🟩 XBOX Live	**7648-7649**	🟪 CU-SeeMe
23	Telnet	**587**	SMTP	**3124**	HTTP Proxy	**8000**	🟨 Internet Radio
25	SMTP	**591**	FileMaker	**3127**	🟥 MyDoom	**8080**	HTTP Proxy
42	WINS Replication	**593**	Microsoft DCOM	**3128**	HTTP Proxy	**8086-8087**	Kaspersky AV
43	WHOIS	**631**	Internet Printing	**3222**	GLBP	**8118**	Privoxy
49	TACACS	**636**	🟦 LDAP over SSL	**3260**	iSCSI Target	**8200**	VMware Server
53	DNS	**639**	MSDP (PIM)	**3306**	MySQL	**8500**	Adobe ColdFusion
67-68	DHCP/BOOTP	**646**	LDP (MPLS)	**3389**	Terminal Server	**8767**	🟪 TeamSpeak
69	TFTP	**691**	MS Exchange	**3689**	iTunes	**8866**	🟥 Bagle.B
70	Gopher	**860**	iSCSI	**3690**	Subversion	**9100**	HP JetDirect
79	Finger	**873**	rsync	**3724**	🟩 World of Warcraft	**9101-9103**	Bacula
80	HTTP	**902**	VMware Server	**3784-3785**	🟨 Ventrilo	**9119**	🟪 MXit
88	Kerberos	**989-990**	🟦 FTP over SSL	**4333**	mSQL	**9800**	WebDAV
102	MS Exchange	**993**	🟦 IMAP4 over SSL	**4444**	🟥 Blaster	**9898**	🟥 Dabber
110	POP3	**995**	🟦 POP3 over SSL	**4664**	Google Desktop	**9988**	🟥 Rbot/Spybot
113	Ident	**1025**	Microsoft RPC	**4672**	🟧 eMule	**9999**	Urchin
119	NNTP (Usenet)	**1026-1029**	Windows Messenger	**4899**	Radmin	**10000**	Webmin
123	NTP	**1080**	SOCKS Proxy	**5000**	UPnP	**10000**	BackupExec
135	Microsoft RPC	**1080**	🟥 MyDoom	**5001**	🟨 Slingbox	**10113-10116**	NetIQ
137-139	NetBIOS	**1194**	OpenVPN	**5001**	iperf	**11371**	OpenPGP
143	IMAP4	**1214**	🟧 Kazaa	**5004-5005**	🟨 RTP	**12035-12036**	🟩 Second Life
161-162	SNMP	**1241**	Nessus	**5050**	🟪 Yahoo! Messenger	**12345**	🟥 NetBus
177	XDMCP	**1311**	Dell OpenManage	**5060**	🟨 SIP	**13720-13721**	NetBackup
179	BGP	**1337**	🟧 WASTE	**5190**	🟪 AIM/ICQ	**14567**	🟩 Battlefield

TYPES OF SERVERS

- Daemon server
  - Example: NTP server
- Superserver
- Stateless server
  - Example: Apache server
- Stateful server
- Object servers
- EJB servers

NTP EXAMPLE

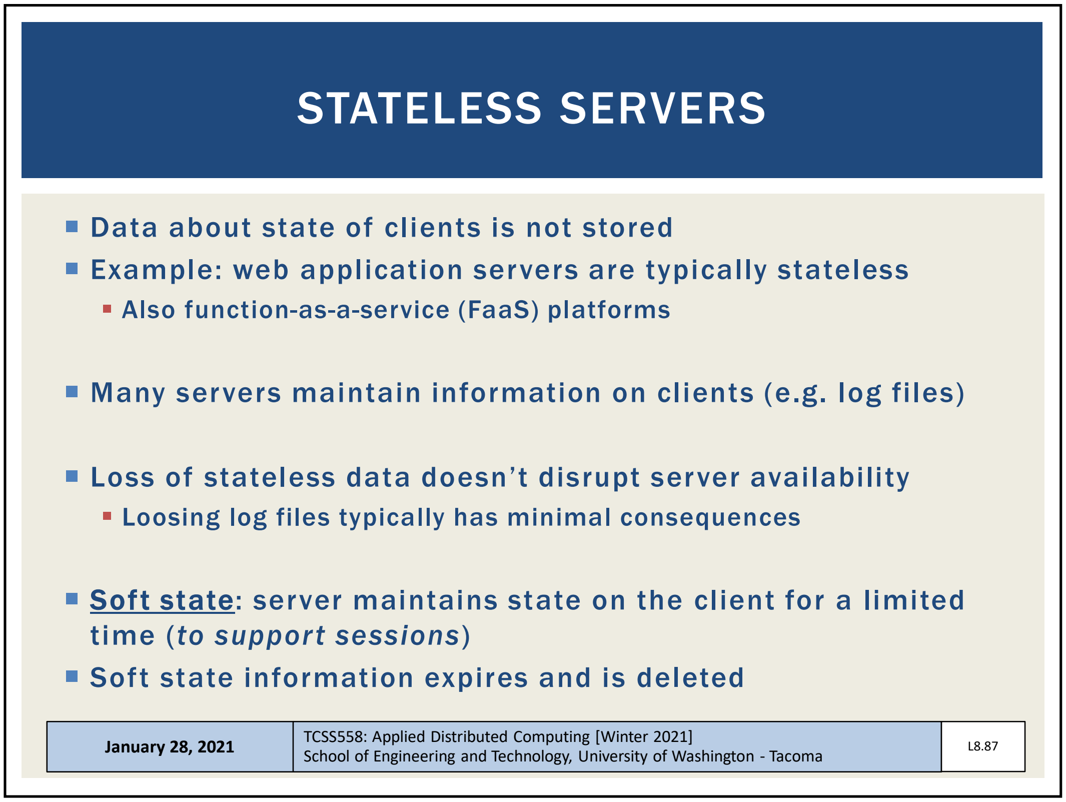
- Daemon servers
  - Run locally on Linux
  - Track current server end points (outside servers)
  - Example: network time protocol (ntp) daemon
    - Listen locally on specific port (ntp is 123)
    - Daemons routes local client traffic to the configured endpoint servers
    - University of Washington: time.u.washington.edu
    - Example "`ntpq -p`"
      - Queries local ntp daemon, routes traffic to configured server(s)

SUPERSERVER

- Linux inetd / xinetd
  - Single superserver
  - Extended internet service daemon
  - Not installed by default on Ubuntu
  - Intended for use on server machines
  - Used to configure box as a server for multiple internet services
    - E.g. ftp, pop, telnet
  - inetd daemon responds to multiple endpoints for multiple services
  - Requests fork a process to run required executable program
- Check what ports you're listening on:
  - `sudo netstat -tap | grep LISTEN`

INTERRUPTING A SERVER

- Server design issue:
  - Active client/server communication is taking place over a port
  - How can the server / data transfer protocol support interruption?
- Consider transferring a 1 GB image, how do you pass a unrelated message in this stream?
  1. <u>Out-of-band</u> data: special messages sent in-stream to support interrupting the server (*TCP urgent data*)
  2. Use a separate connection (different port) for admin control info
- Example: sftp secure file transfer protocol
  - Once a file transfer is started, can't be stopped easily
  - Must kill the client and/or server



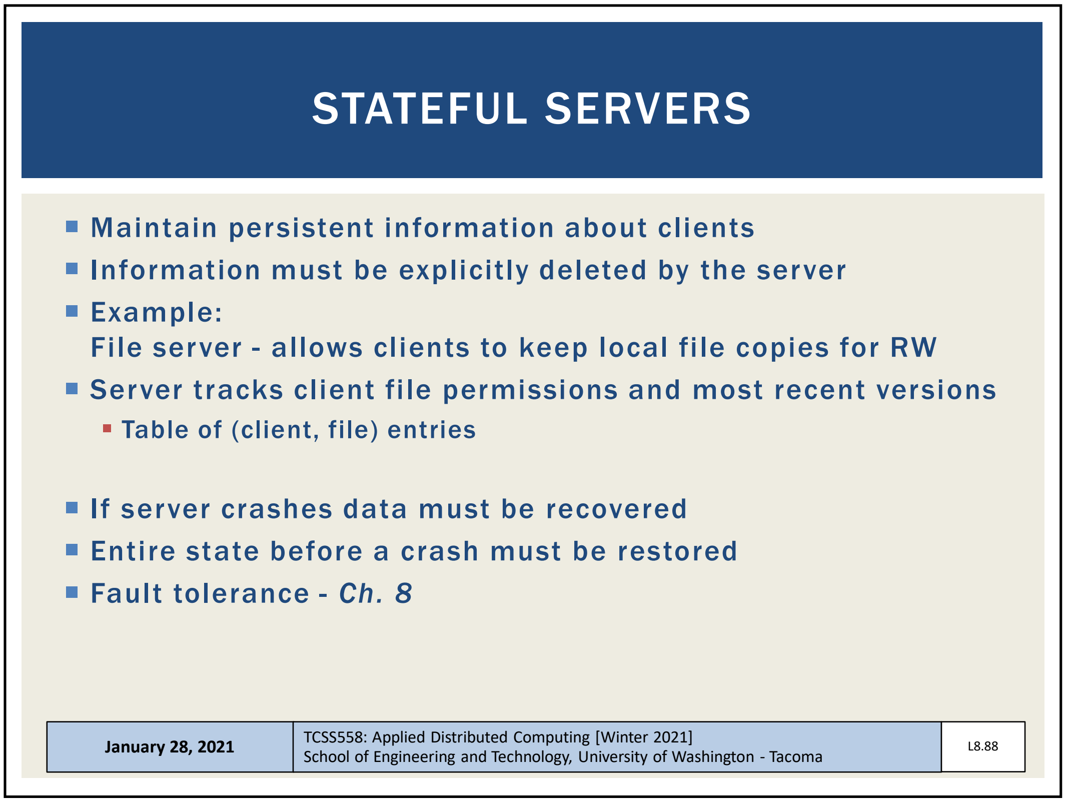
STATELESS SERVERS

- Data about state of clients is not stored
- Example: web application servers are typically stateless
  - Also function-as-a-service (FaaS) platforms

- Many servers maintain information on clients (e.g. log files)

- Loss of stateless data doesn't disrupt server availability
  - Loosing log files typically has minimal consequences

- **Soft state**: server maintains state on the client for a limited time (*to support sessions*)
- Soft state information expires and is deleted



STATEFUL SERVERS

- Maintain persistent information about clients
- Information must be explicitly deleted by the server
- Example:
  File server - allows clients to keep local file copies for RW
- Server tracks client file permissions and most recent versions
  - Table of (client, file) entries

- If server crashes data must be recovered
- Entire state before a crash must be restored
- Fault tolerance - *Ch. 8*

STATEFUL SERVERS - 2

- Session state
  - Tracks series of operations by a single user
  - Maintained temporarily, not indefinitely
  - Often retained for multi-tier client server applications
  - Minimal consequence if session state is lost
  - Clients must start over, reinitialize sessions
- Permanent state
  - Customer information, software keys
- Client-side cookies
  - When servers don't maintain client state, clients can store state locally in "cookies"
  - Cookies are not executable, simply client-side data

OBJECT SERVERS

- <u>**OBJECTIVE:**</u> Host objects and enable remote client access
- Do not provide a specific service
  - Do nothing if there are no objects to host
- Support adding/removing hosted objects
- Provide a home where objects live
- Objects, *themselves*, provide "<u>services</u>"
- Object parts
  - State data
  - Code (methods, etc.)
- <u>Transient object(s)</u>
  - Objects with limited lifetime (< server)
  - Created at first invocation, destroyed when no longer used (i.e. no clients remain "bound").
  - Disadvantage: initialization may be expensive
  - Alternative: preinitialize and retain objects on server start-up

OBJECT SERVERS - 2

- **Should object servers isolate memory for object instances?**
  - Share neither code nor data
  - May be necessary if objects couple data and implementation

- Object server threading designs:
  - Single thread of control for object server
  - One thread for each object
  - Servers use separate thread for client requests

- Threads created on demand *vs.* Server maintains pool of threads

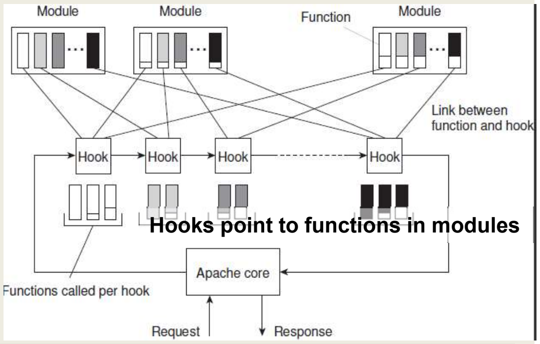
- **What are the tradeoffs for creating server threads on demand vs. using a thread pool?**

EJB – ENTERPRISE JAVA BEANS

- EJB- specialized Java object hosted by a EJB web container
- 4 types: stateless, stateful, entity, and message-driven beans
- Provides "middleware" standard (framework) for implementing back-ends of enterprise applications
- EJB web application containers integrate support for:
  - Transaction processing
  - Persistence
  - Concurrency
  - Event-driven programming
  - Asynchronous method invocation
  - Job scheduling
  - Naming and discovery services (JNDI)
  - Interprocess communication
  - Security
  - Software component deployment to an application server

APACHE WEB SERVER

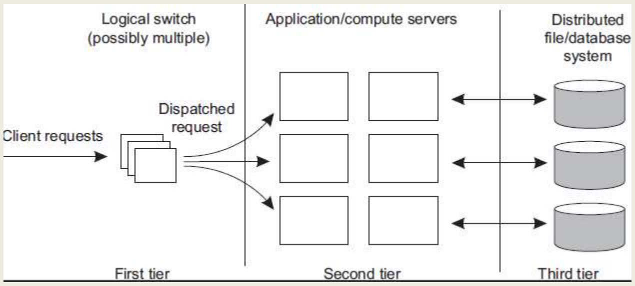
- Highly configurable, extensible, platform independent
- Supports TCP HTTP protocol communication
- Uses hooks – placeholders for group of functions
- Requests processed in phases by hooks
- Many hooks:
  - Translate a URL
  - Write info to log
  - Check client ID
  - Check access rights
- Hooks processed in order enforcing flow-of-control
- Functions in replaceable modules



Hooks point to functions in modules

SERVER CLUSTERS

- Hosted across an LAN or WAN
- Collection of interconnected machines
- Can be organized in tiers:
  - Web server → app server → DB server
  - App and DB server sometimes integrated

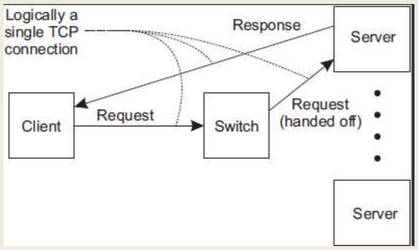


LAN REQUEST DISPATCHING

- Front end of three tier architecture (logical switch) provides distribution transparency – hides multiple servers
- Transport-layer switches: switch accepts TCP connection requests, hands off to a server
  - Example: hardware load balancer (F5 networks – Seattle)
  - HW Load balancer - OSI layers 4-7
- Network-address-translation (NAT) approach:
  - All requests pass through switch
  - Switch sits in the middle of the client/server TCP connection
  - Maps (rewrites) source and destination addresses
- Connection hand-off approach:
  - TCP Handoff: switch hands of connection to a selected server

LAN REQUEST DISPATCHING - 2

- Who is the best server to handle the request?
- Switch plays important role in distributing requests
- Implements load balancing
- **Round-robin** – routes client requests to servers in a looping fashion



- **Transport-level** – route client requests based on TCP port number
- **Content-aware request distribution** – route requests based on inspecting data payload and determining which server node should process the request

WIDE AREA CLUSTERS

- Deployed across the internet
- Leverage resource/infrastructure from Internet Service Providers (ISPs)
- Cloud computing simplifies building WAN clusters
- Resource from a single cloud provider can be combined to form a cluster

- **For deploying a cloud-based cluster (WAN), what are the implications of deploying nodes to:**
- (1) a single availability zone (e.g. us-east-1e)?
- (2) across multiple availability zones?

WAN REQUEST DISPATCHING

- Goal: minimize network latency using WANs (e.g. Internet)
- Send requests to nearby servers

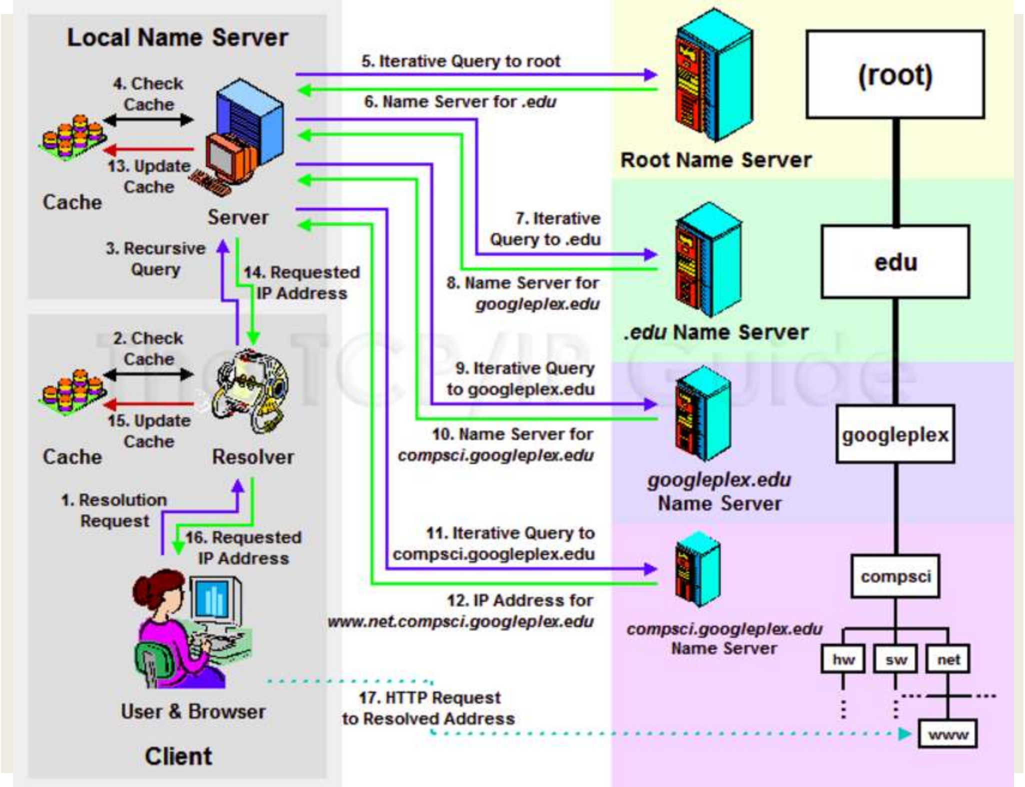
- Request dispatcher: routes requests to nearby server
- <u>**Example**</u>: Domain Name System
  - Hierarchical decentralized naming system

- Linux: find your DNS servers:

```
# Find you device name of interest
nmcli dev
# Show device configuration
nmcli device show <device name>
```

DNS LOOKUP

- First query local server(s) for address
- Typically there are (2) local DNS servers
  - One is backup
- Hostname may be cached at local DNS server
  - E.g. www.google.com
- If not found, local DNS server routes to other servers
- Routing based on components of the hostname
- DNS servers down the chain mask the client IP, and use the originating DNS server IP to identify a local host
- ***Weakness:*** *client may be far from DNS server used. Resolved hostname is close to DNS server, but not necessarily close to the client*



DNS: LINUX COMMANDS

- `nslookup <ip addr / hostname>`
- Name server lookup – translates hostname or IP to the inverse

- `traceroute <ip addr / hostname>`
- Traces network path to destination
- By default, output is limited to 30 hops, can be increased

DNS EXAMPLE – WAN DISPATCHING

- Ping www.google.com in WA from wireless network:
  - nslookup: 6 alternate addresses returned, choose (74.125.28.147)
  - Ping 74.125.28.147: Average RTT = **22.458 ms (11 attempts, 22 hops)**
- Ping www.google.com in VA (us-east-1) from EC2 instance:
  - nslookup: 1 address returned, choose 172.217.9.196
  - Ping 172.217.9.196: Average RTT = 1.278 ms (11 attempts, 13 hops)

- From VA EC2 instance, ping WA *www.google* server
- Ping 74.125.28.147: Average RTT 62.349ms (11 attempts, 27 hops)
- Pinging the WA-local server is ~60x slower from VA

- From local wireless network, ping VA us-east-1 google :
- Ping 172.217.9.196: Average RTT=81.637ms (11 attempts, 15 hops)

DNS EXAMPLE – WAN DISPATCHING

- Ping www.google.com in WA from wireless network:
  - nslookup: 6 alternate addresses returned, choose (74.125.28.147)

Latency to ping VA server in WA: ~3.63x

WA client: local-google 22.458ms to VA-google 81.637ms

Latency to ping WA server in VA: ~48.7x

VA client: local-google 1.278ms to WA-google 62.349!

- From local wireless network, ping VA us-east-1 google :
- Ping 172.217.9.196: Average RTT=81.637ms (11 attempts, 15 hops)

QUESTIONS