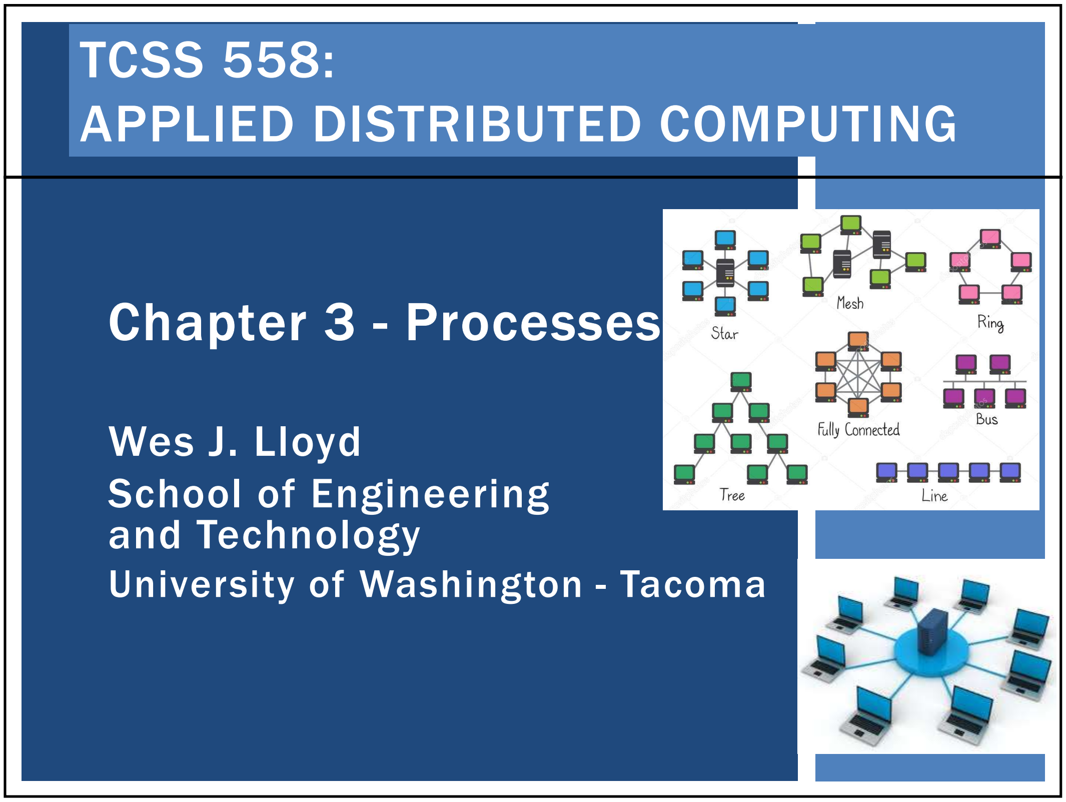
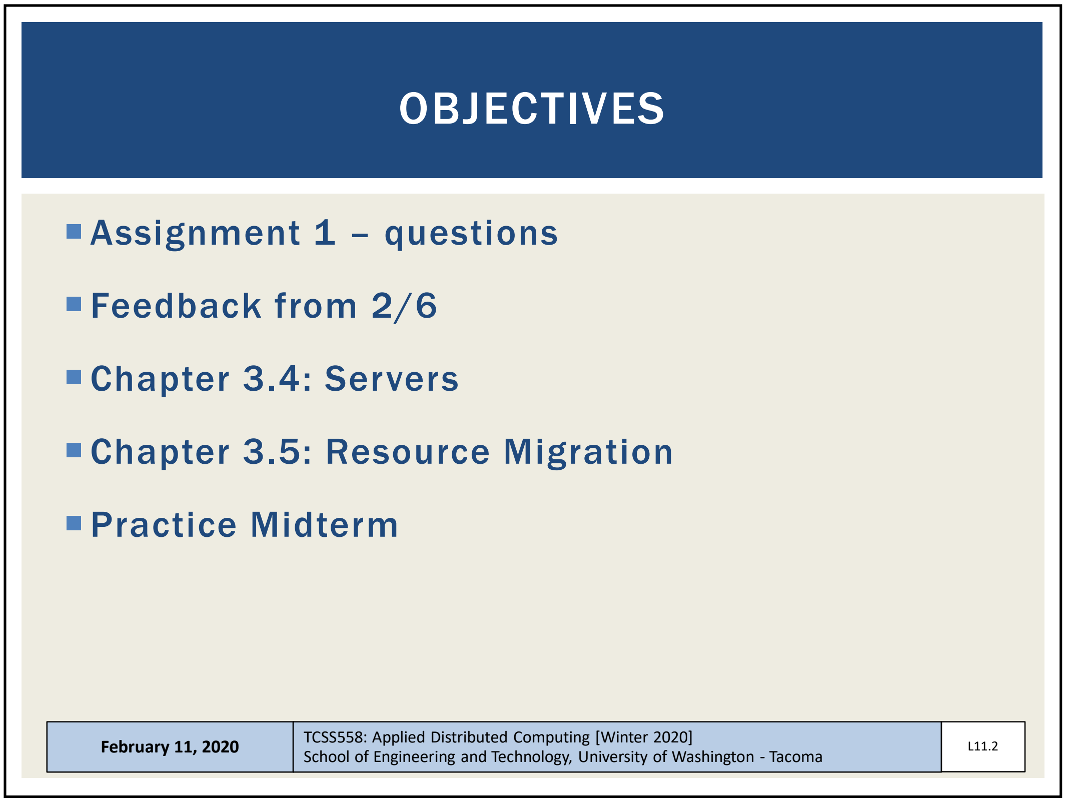


# TCSS 558:
# APPLIED DISTRIBUTED COMPUTING

## Chapter 3 - Processes

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

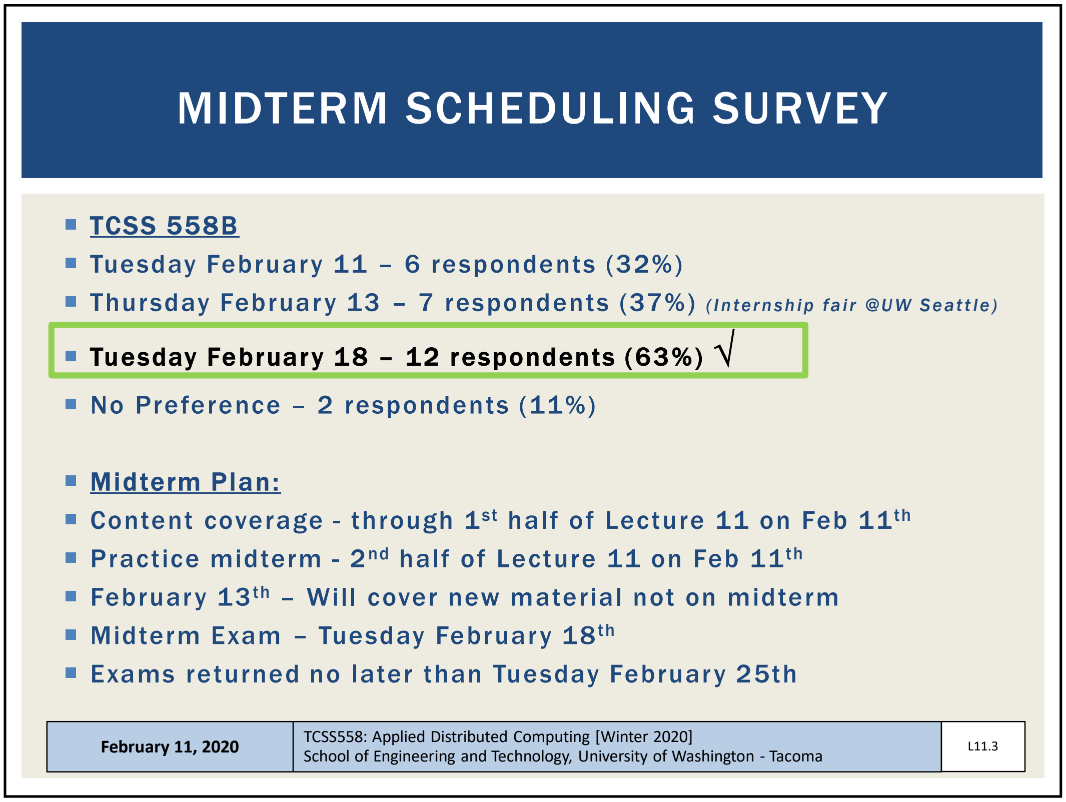
1



## OBJECTIVES

- Assignment 1 – questions
- Feedback from 2/6
- Chapter 3.4: Servers
- Chapter 3.5: Resource Migration
- Practice Midterm

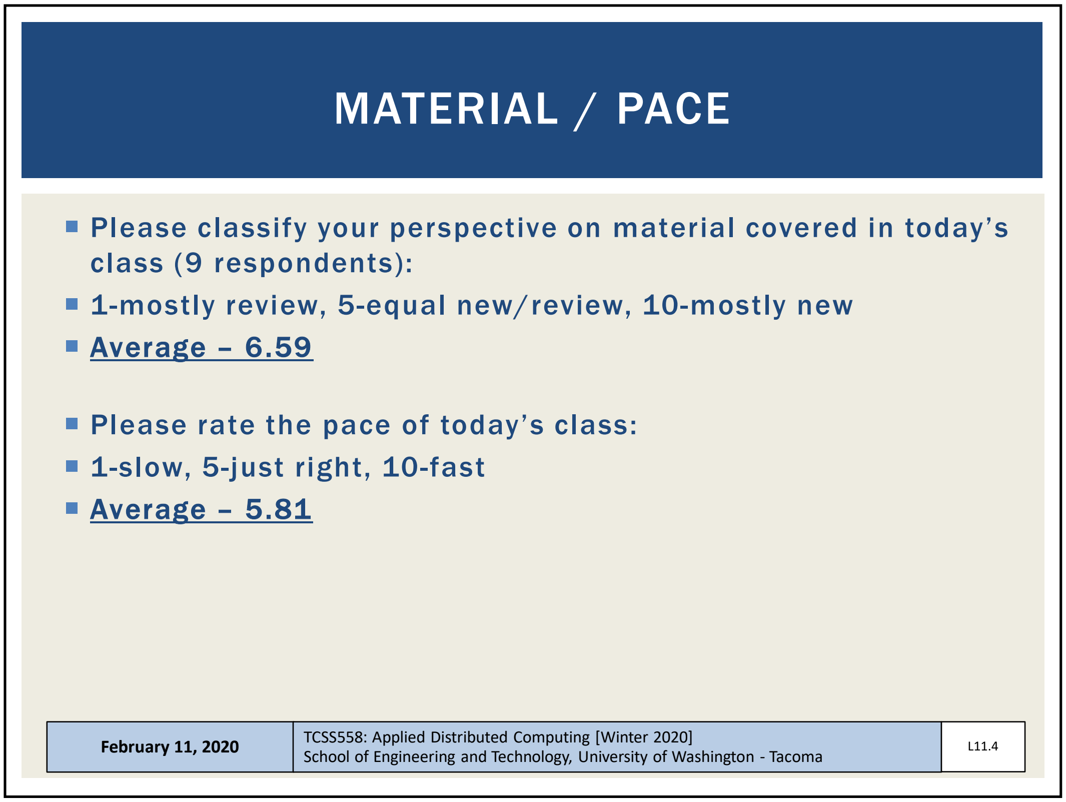
2



# MIDTERM SCHEDULING SURVEY

- **TCSS 558B**
- Tuesday February 11 – 6 respondents (32%)
- Thursday February 13 – 7 respondents (37%) *(Internship fair @UW Seattle)*
- **Tuesday February 18 – 12 respondents (63%)** √
- No Preference – 2 respondents (11%)

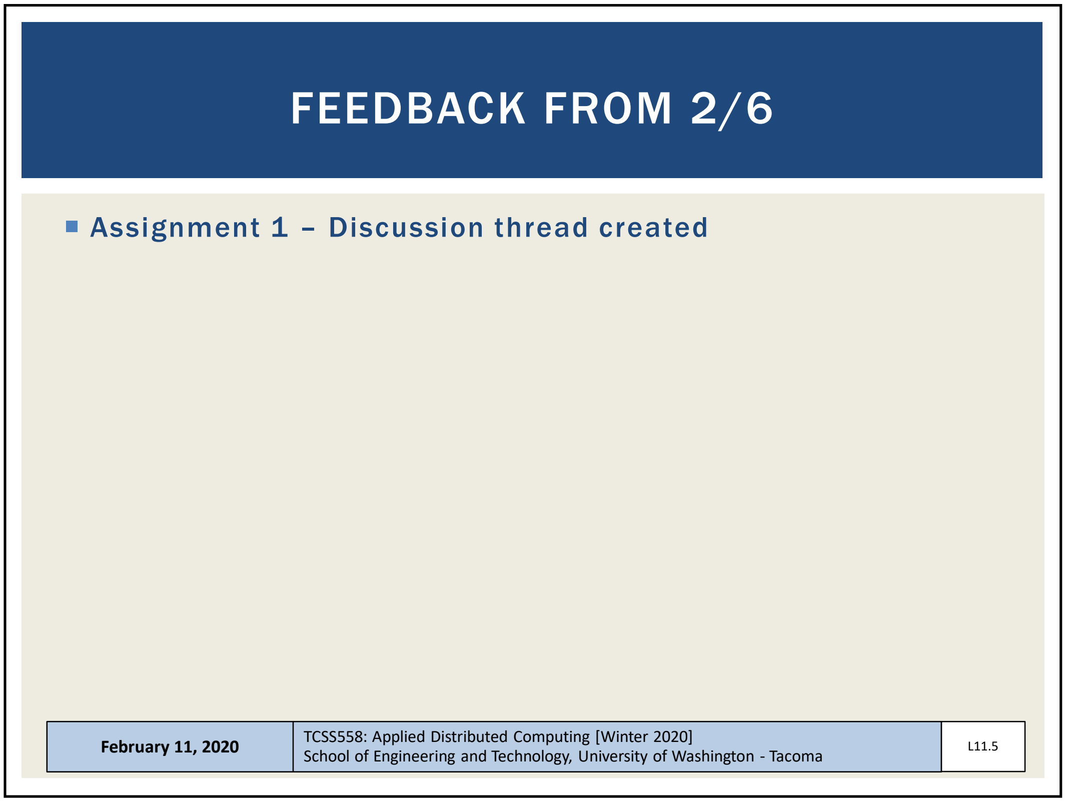
- **Midterm Plan:**
- Content coverage - through 1st half of Lecture 11 on Feb 11th
- Practice midterm - 2nd half of Lecture 11 on Feb 11th
- February 13th – Will cover new material not on midterm
- Midterm Exam – Tuesday February 18th
- Exams returned no later than Tuesday February 25th



# MATERIAL / PACE

- Please classify your perspective on material covered in today's class (9 respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 6.59**

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



# FEEDBACK FROM 2/6

- Assignment 1 – Discussion thread created

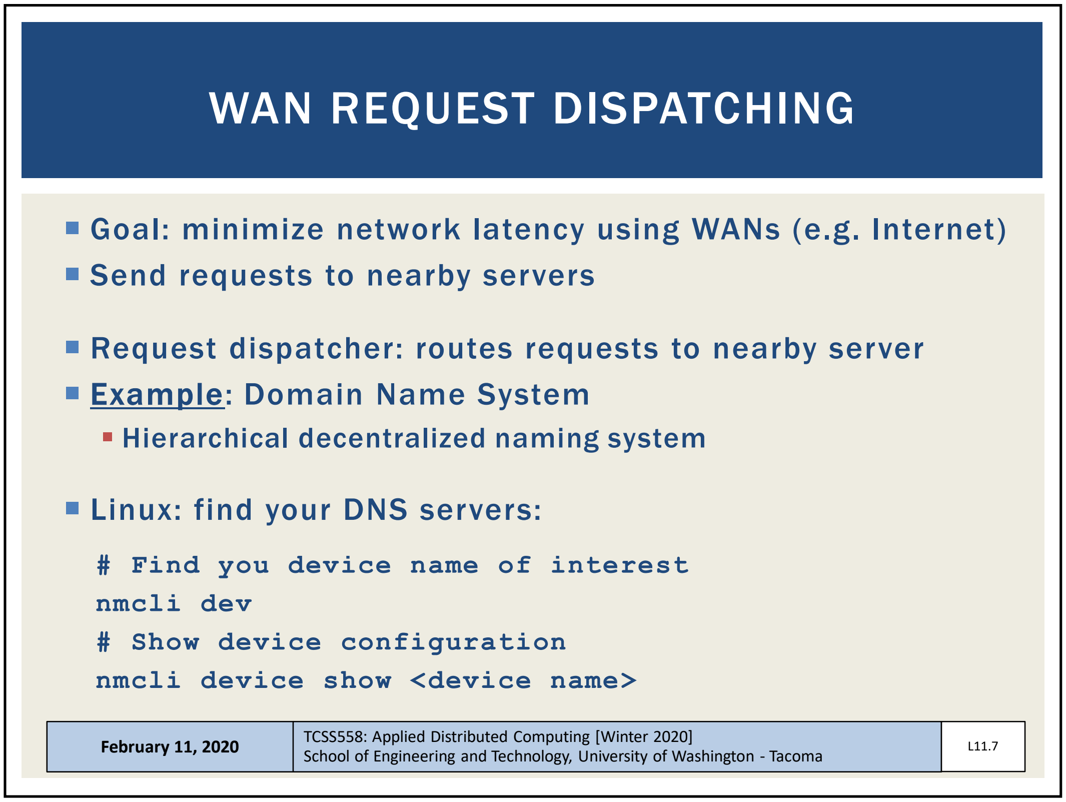
| February 11, 2020 | TCSS558: Applied Distributed Computing [Winter 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L11.5 |
|---|---|---|

5

# CH. 3.4: SERVERS

L11.6

6



# WAN REQUEST DISPATCHING

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

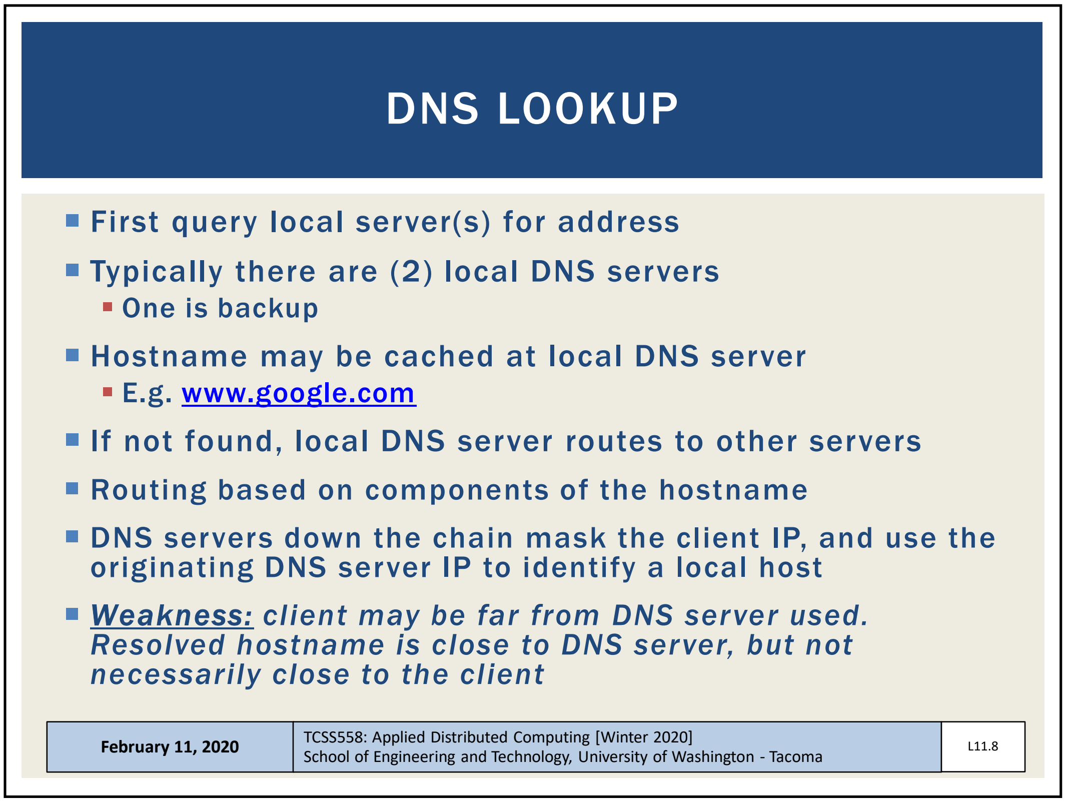
- Request dispatcher: routes requests to nearby server
- **Example**: Domain Name System
  - Hierarchical decentralized naming system

- Linux: find your DNS servers:

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

February 11, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L11.7

7

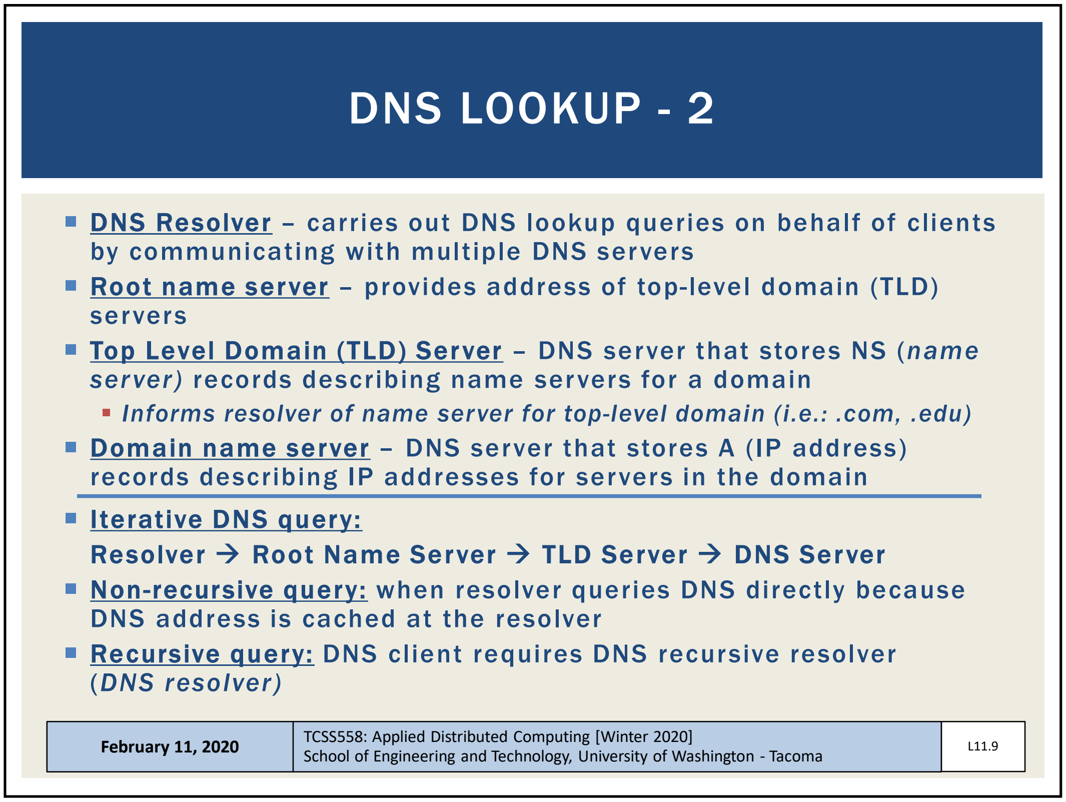


# DNS LOOKUP

- 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*

February 11, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L11.8

8

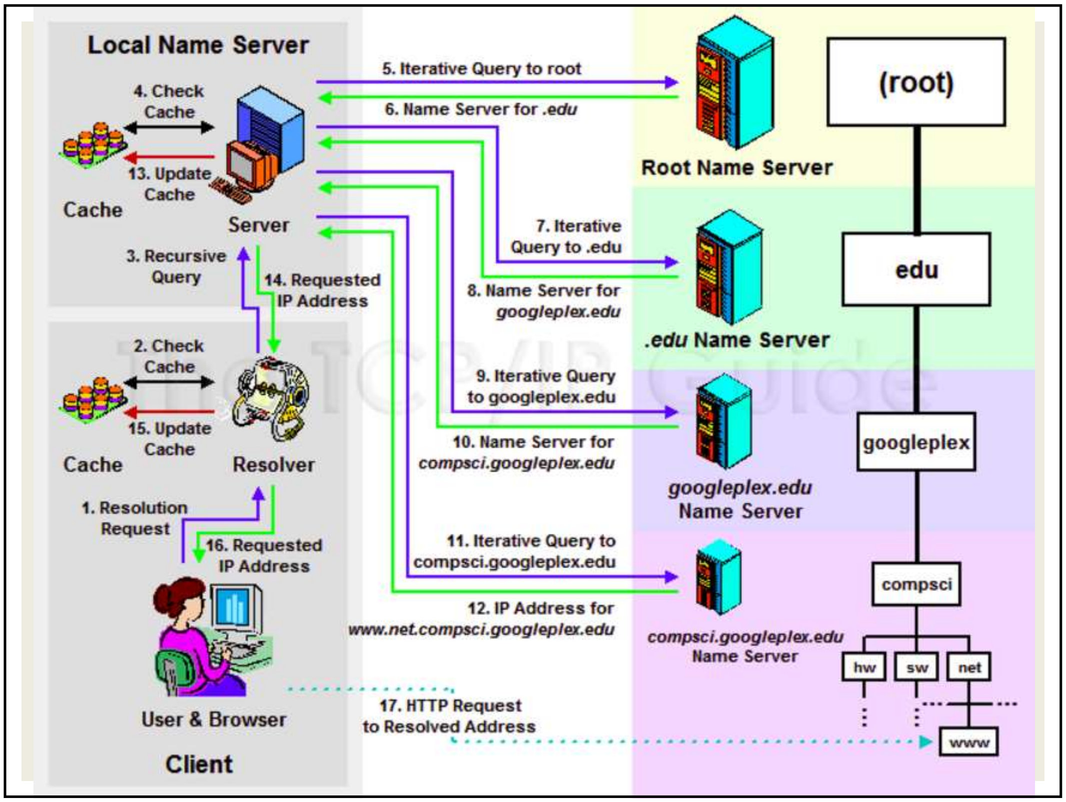


# DNS LOOKUP - 2

- **DNS Resolver** – carries out DNS lookup queries on behalf of clients by communicating with multiple DNS servers
- **Root name server** – provides address of top-level domain (TLD) servers
- **Top Level Domain (TLD) Server** – DNS server that stores NS (*name server)* records describing name servers for a domain
  - *Informs resolver of name server for top-level domain (i.e.: .com, .edu)*
- **Domain name server** – DNS server that stores A (IP address) records describing IP addresses for servers in the domain
- **Iterative DNS query:**
  Resolver → Root Name Server → TLD Server → DNS Server
- **Non-recursive query:** when resolver queries DNS directly because DNS address is cached at the resolver
- **Recursive query:** DNS client requires DNS recursive resolver (*DNS resolver)*

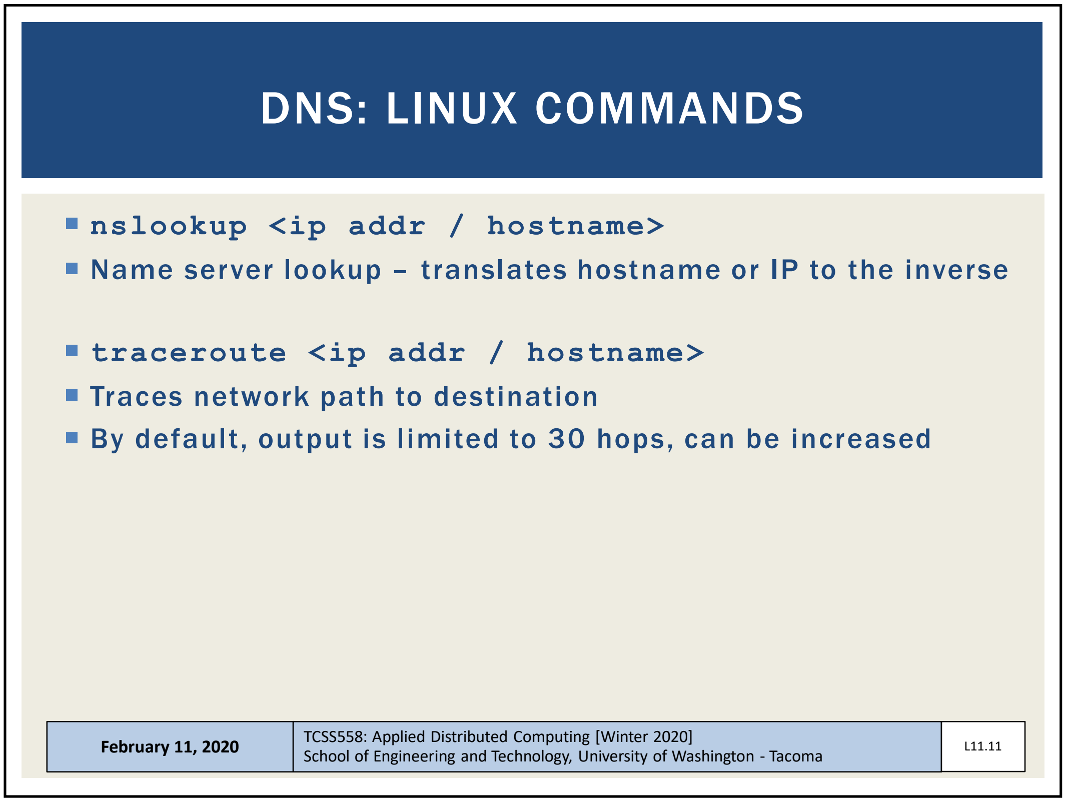
| February 11, 2020 | TCSS558: Applied Distributed Computing [Winter 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L11.9 |
|---|---|---|

9



Local Name Server
4. Check Cache
13. Update Cache
Cache
Server
3. Recursive Query
14. Requested IP Address
2. Check Cache
15. Update Cache
Cache
Resolver
1. Resolution Request
16. Requested IP Address
User & Browser
Client
5. Iterative Query to root
6. Name Server for .edu
7. Iterative Query to .edu
8. Name Server for googleplex.edu
9. Iterative Query to googleplex.edu
10. Name Server for compsci.googleplex.edu
11. Iterative Query to compsci.googleplex.edu
12. IP Address for www.net.compsci.googleplex.edu
17. HTTP Request to Resolved Address
Root Name Server
.edu Name Server
googleplex.edu Name Server
compsci.googleplex.edu Name Server
(root)
edu
googleplex
compsci
hw
sw
net
www
The TCP/IP Guide

10

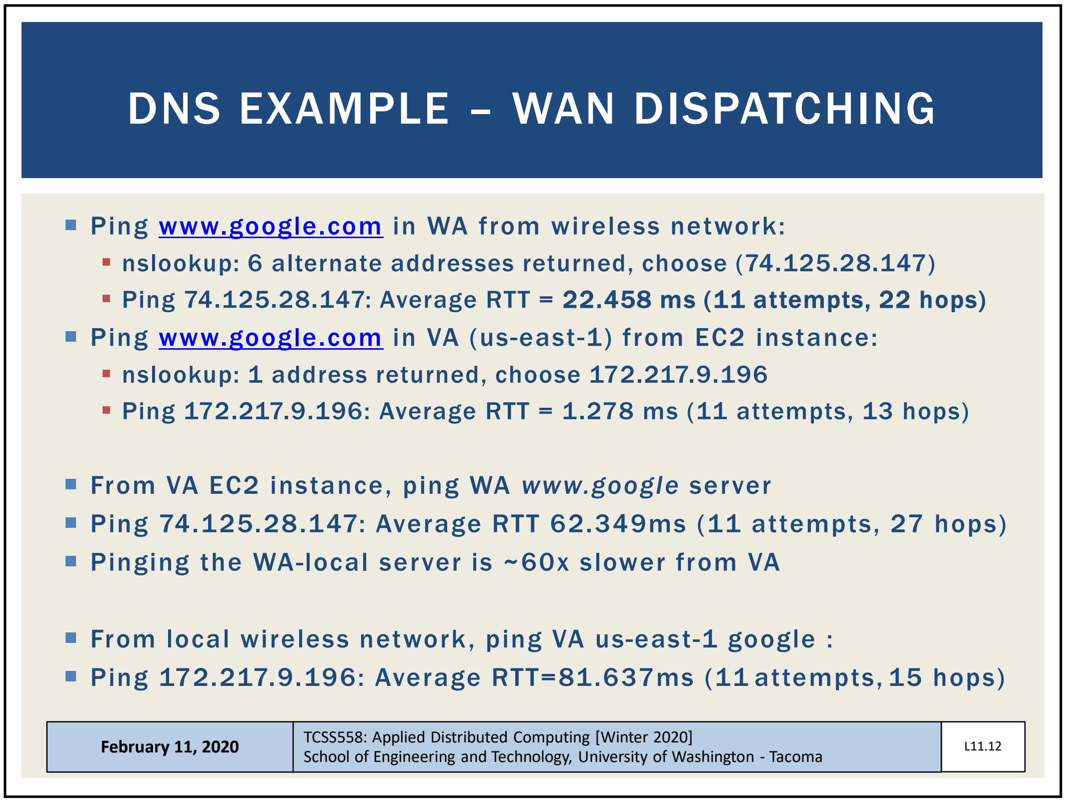


# DNS: LINUX COMMANDS

- `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

11



# 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)

12

# 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" google in WA: ~3.63x**

**WA laptop: local-google 22.458ms to VA-google 81.637ms**

**Latency to ping "WA" google in VA: ~48.7x**

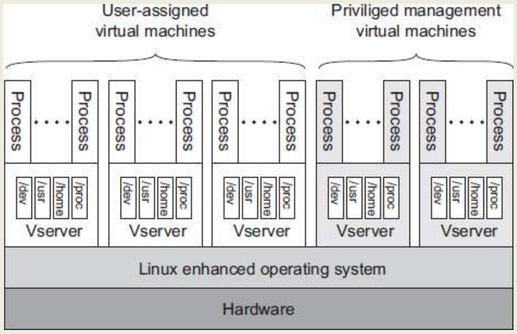
**Virginia ec2 VM: local-google 1.278ms to WA-google 62.349!**

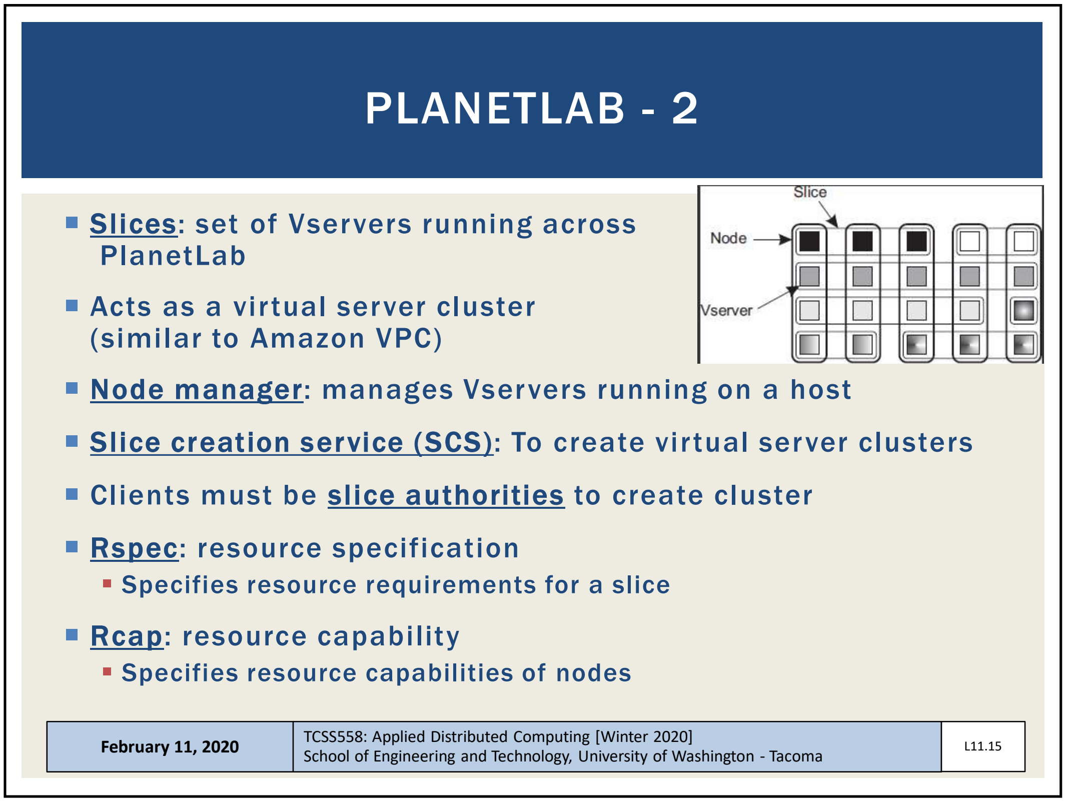
- From local wireless network, ping VA us-east-1 google :
- Ping 172.217.9.196: Average RTT=81.637ms (11 attempts, 15 hops)

# CH 3.2 - EXAMPLE: PLANETLAB

- Unstructured heterogeneous cluster of servers
- Similar to grid but organized as cluster (no grid middleware)
- Testbed established in 2002 for computer networking and distributed systems research
- Organizations share nodes in the cluster

**Leverages Linux Vservers Early "containers" similar to Docker**


User-assigned virtual machines
Priviliged management virtual machines
Process
....
Process
/dev
/usr
/home
/proc
Vserver
Linux enhanced operating system
Hardware



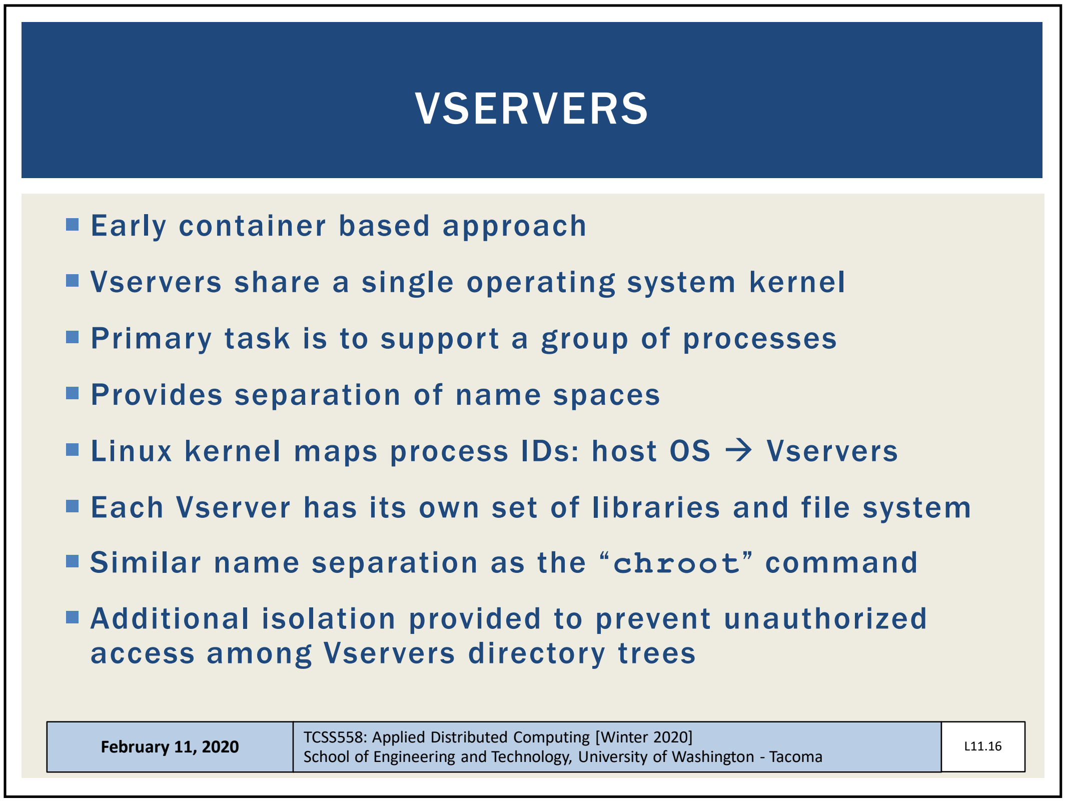
# PLANETLAB - 2

Slice
Node
Vserver

- **Slices**: set of Vservers running across PlanetLab
- Acts as a virtual server cluster (similar to Amazon VPC)
- **Node manager**: manages Vservers running on a host
- **Slice creation service (SCS)**: To create virtual server clusters
- Clients must be **slice authorities** to create cluster
- **Rspec**: resource specification
  - Specifies resource requirements for a slice
- **Rcap**: resource capability
  - Specifies resource capabilities of nodes

| February 11, 2020 | TCSS558: Applied Distributed Computing [Winter 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L11.15 |
|---|---|---|

15



# VSERVERS

- Early container based approach
- Vservers share a single operating system kernel
- Primary task is to support a group of processes
- Provides separation of name spaces
- Linux kernel maps process IDs: host OS → Vservers
- Each Vserver has its own set of libraries and file system
- Similar name separation as the "`chroot`" command
- Additional isolation provided to prevent unauthorized access among Vservers directory trees

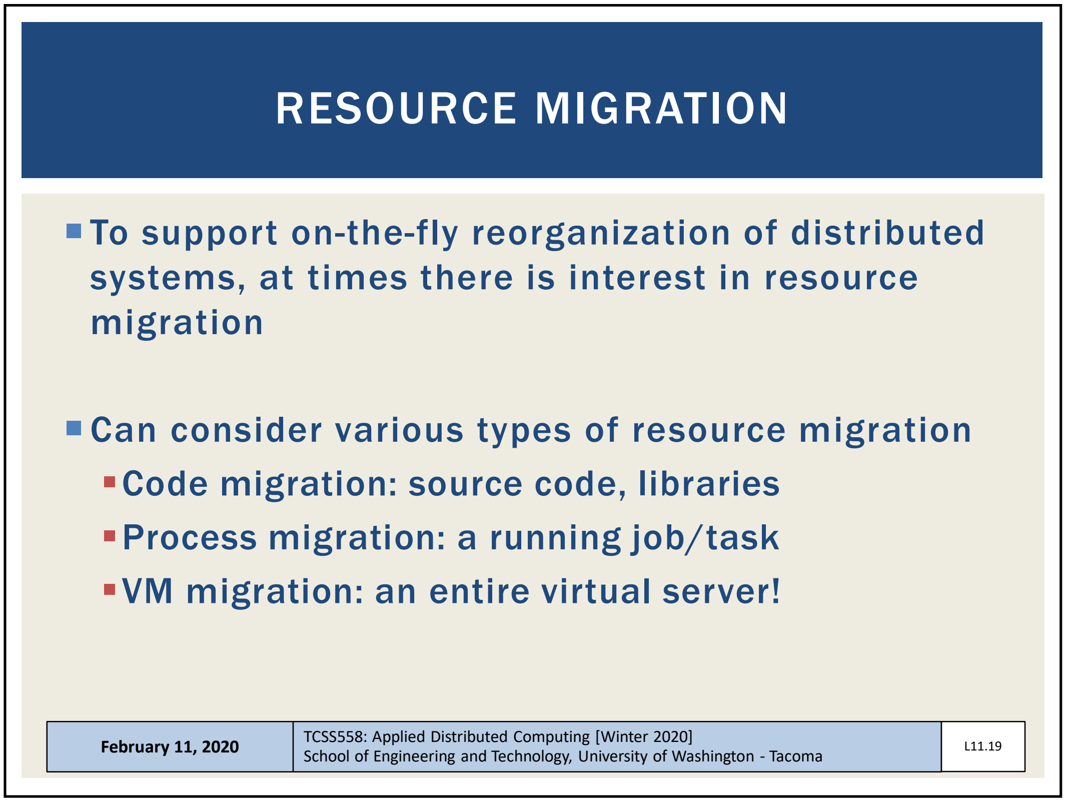
| February 11, 2020 | TCSS558: Applied Distributed Computing [Winter 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L11.16 |
|---|---|---|

16

## VSERVERS - 2

- **Advantages of Vservers (containers) vs. VMs:**
- Simpler resource allocation
- Possible to overbook resources by leveraging dynamic resource allocation - **Example: CPU or RAM** *(assignment 0, config 2)*
- VMs reserve a block of memory
- Containers can oversubscribe memory
  - Memory not formally reserved
  - Linux kernel shares memory among processes
  - Swap filesystem can use disk as extended RAM
- Memory sharing important for PlanetLab
  - Early nodes had limited memory (e.g. 4 GB)
- Vserver hogging most memory reset when out of swap space

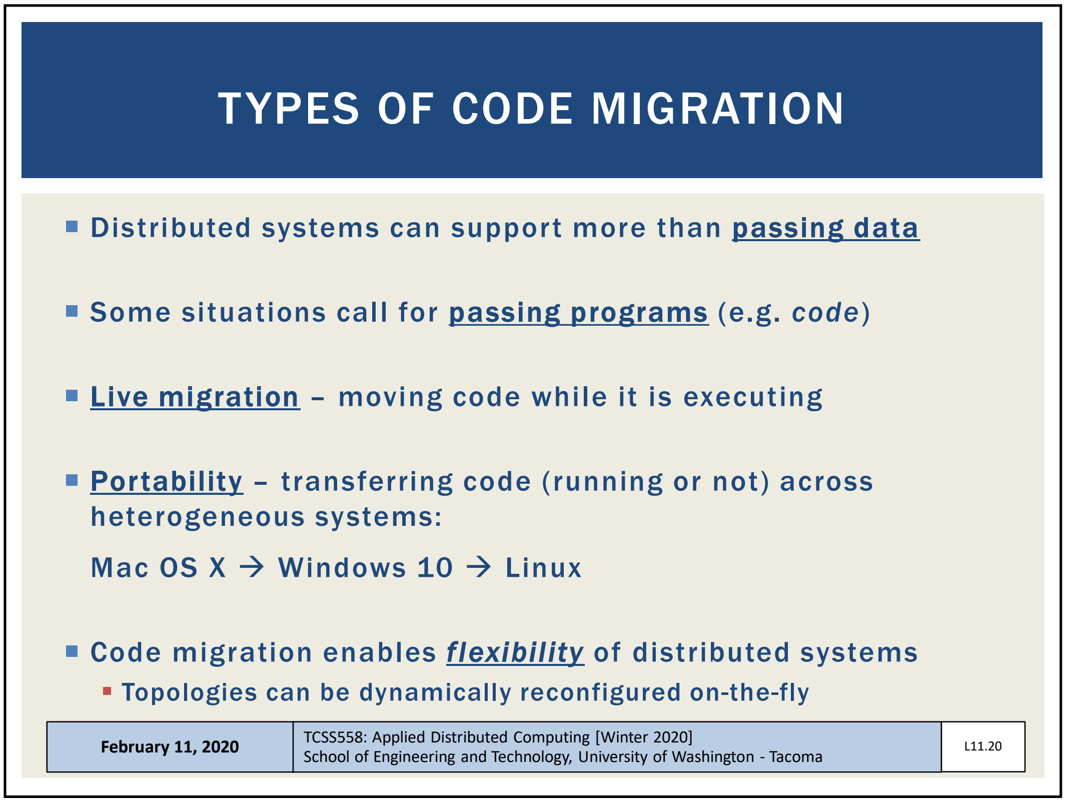
## CH. 3.5: RESOURCE (CODE) MIGRATION



## RESOURCE MIGRATION

- To support on-the-fly reorganization of distributed systems, at times there is interest in resource migration
- Can consider various types of resource migration
  - Code migration: source code, libraries
  - Process migration: a running job/task
  - VM migration: an entire virtual server!

19

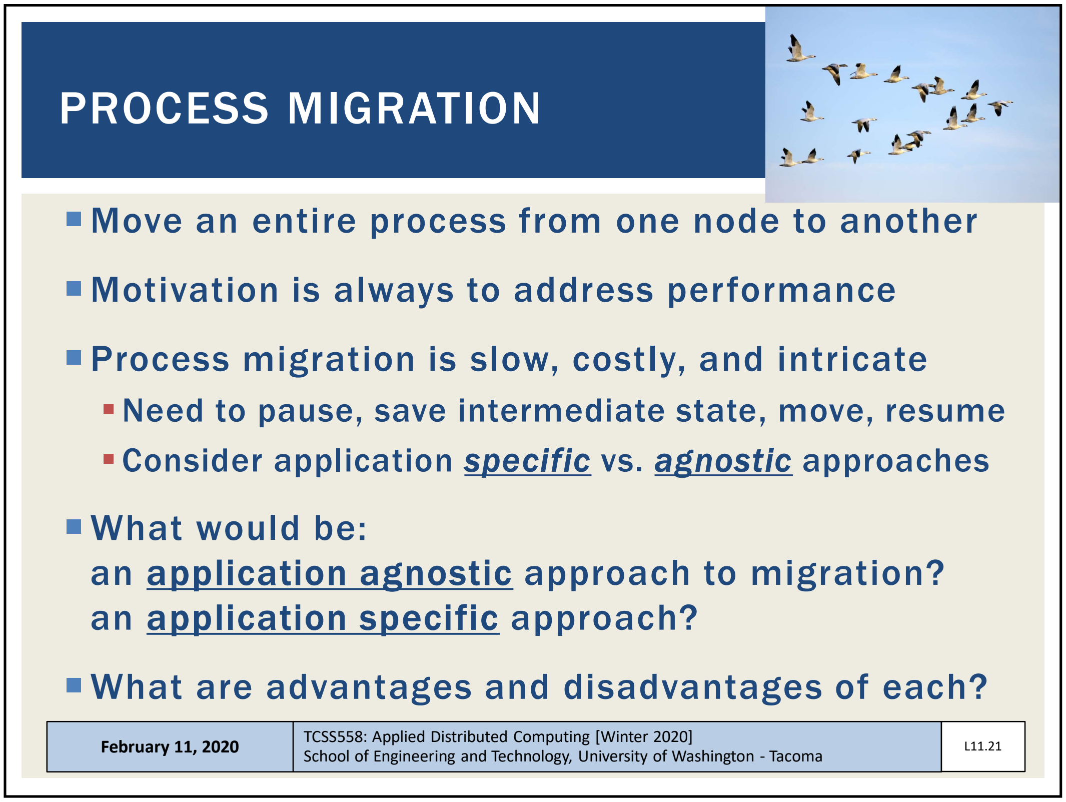


## TYPES OF CODE MIGRATION

- Distributed systems can support more than **passing data**
- Some situations call for **passing programs** (e.g. *code*)
- **Live migration** – moving code while it is executing
- **Portability** – transferring code (running or not) across heterogeneous systems:

  Mac OS X → Windows 10 → Linux
- Code migration enables ***flexibility*** of distributed systems
  - Topologies can be dynamically reconfigured on-the-fly

20

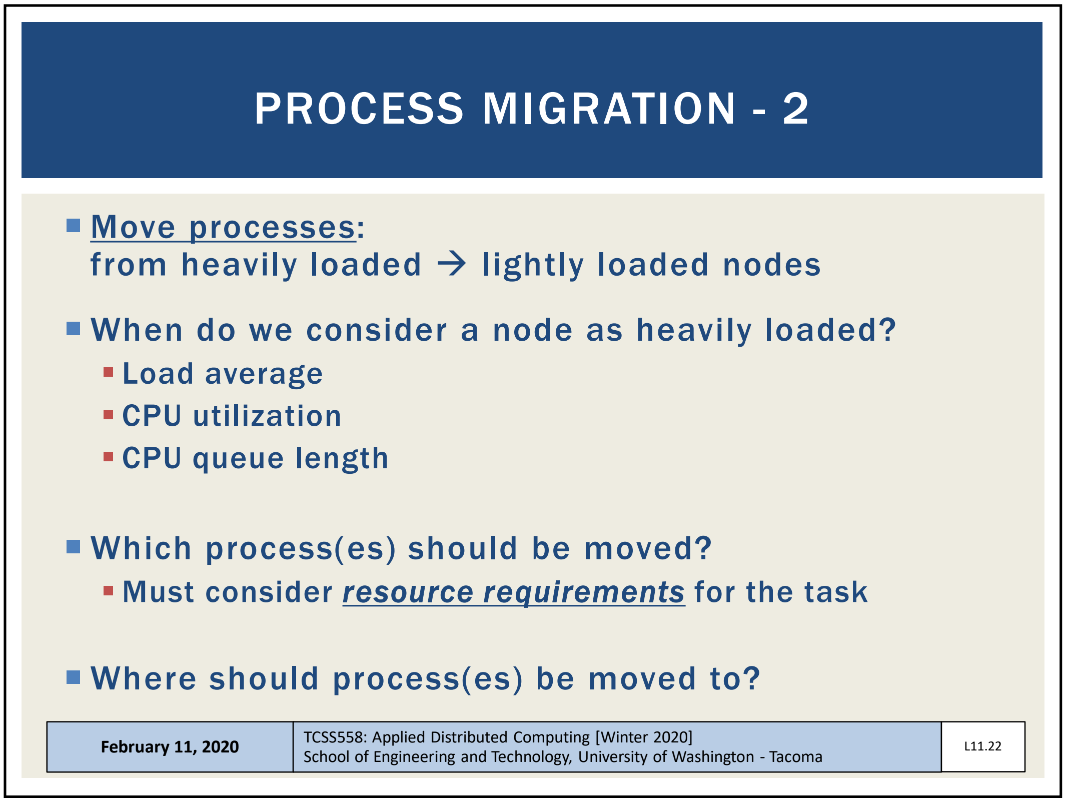


## PROCESS MIGRATION

- Move an entire process from one node to another
- Motivation is always to address performance
- Process migration is slow, costly, and intricate
  - Need to pause, save intermediate state, move, resume
  - Consider application ***specific*** vs. ***agnostic*** approaches
- What would be:
  an **application agnostic** approach to migration?
  an **application specific** approach?
- What are advantages and disadvantages of each?

| February 11, 2020 | TCSS558: Applied Distributed Computing [Winter 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L11.21 |
|---|---|---|

21

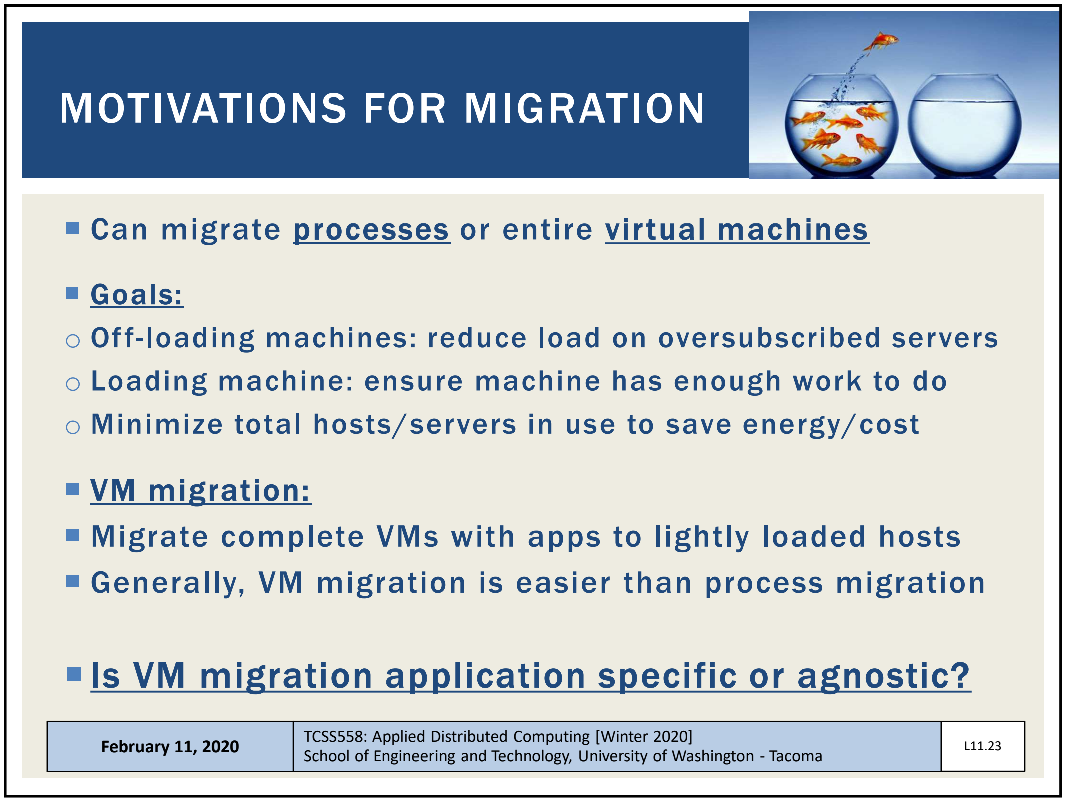


## PROCESS MIGRATION - 2

- Move processes:
  from heavily loaded → lightly loaded nodes
- When do we consider a node as heavily loaded?
  - Load average
  - CPU utilization
  - CPU queue length
- Which process(es) should be moved?
  - Must consider ***resource requirements*** for the task
- Where should process(es) be moved to?

| February 11, 2020 | TCSS558: Applied Distributed Computing [Winter 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L11.22 |
|---|---|---|

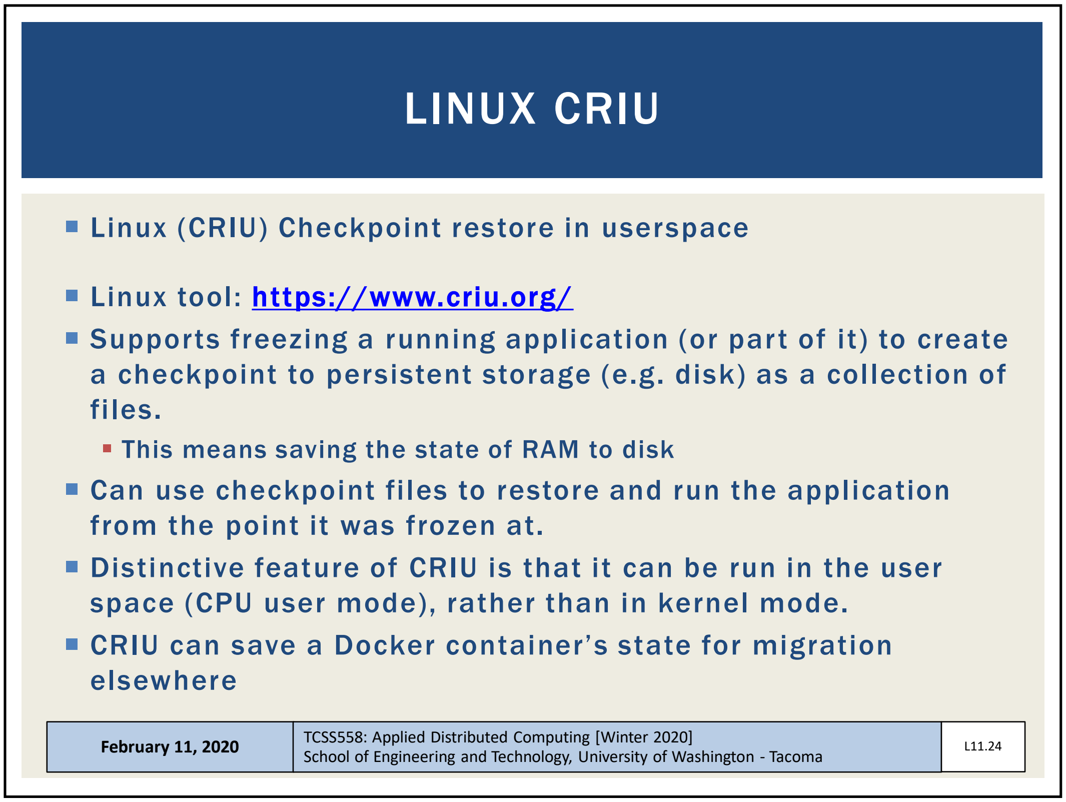
22



## MOTIVATIONS FOR MIGRATION

- Can migrate processes or entire virtual machines
- Goals:
  - ○ Off-loading machines: reduce load on oversubscribed servers
  - ○ Loading machine: ensure machine has enough work to do
  - ○ Minimize total hosts/servers in use to save energy/cost
- VM migration:
- Migrate complete VMs with apps to lightly loaded hosts
- Generally, VM migration is easier than process migration
- Is VM migration application specific or agnostic?

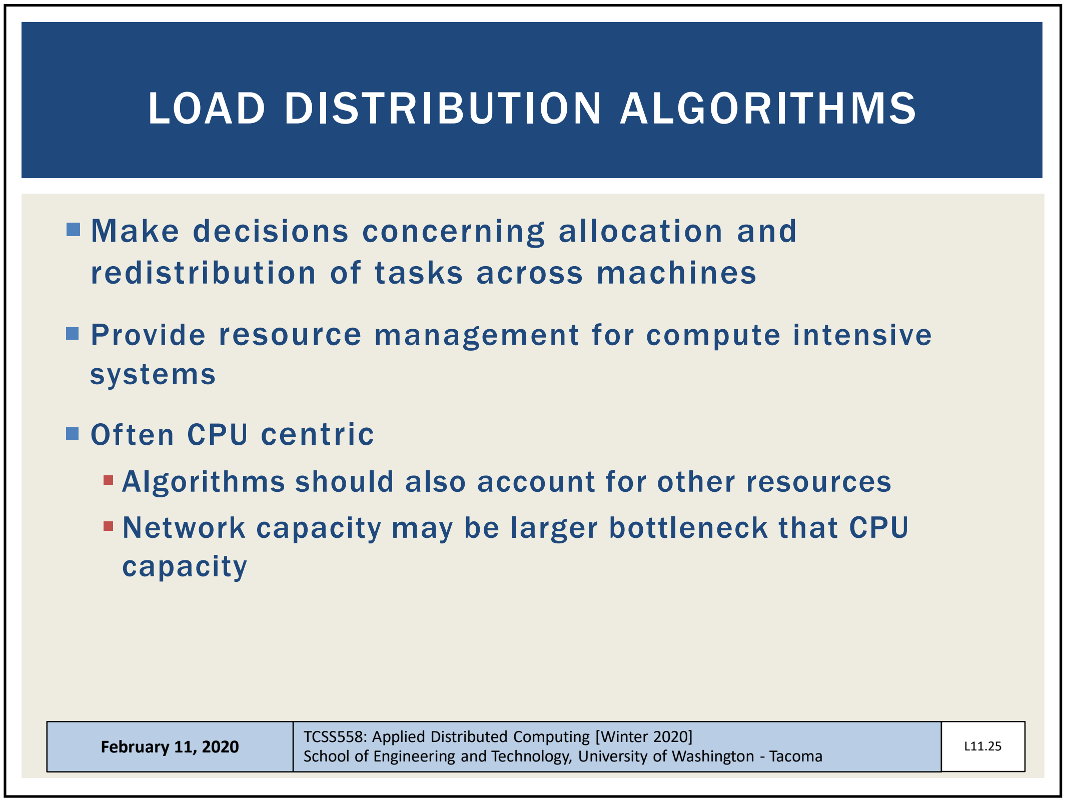
23



## LINUX CRIU

- Linux (CRIU) Checkpoint restore in userspace
- Linux tool: https://www.criu.org/
- Supports freezing a running application (or part of it) to create a checkpoint to persistent storage (e.g. disk) as a collection of files.
  - This means saving the state of RAM to disk
- Can use checkpoint files to restore and run the application from the point it was frozen at.
- Distinctive feature of CRIU is that it can be run in the user space (CPU user mode), rather than in kernel mode.
- CRIU can save a Docker container's state for migration elsewhere

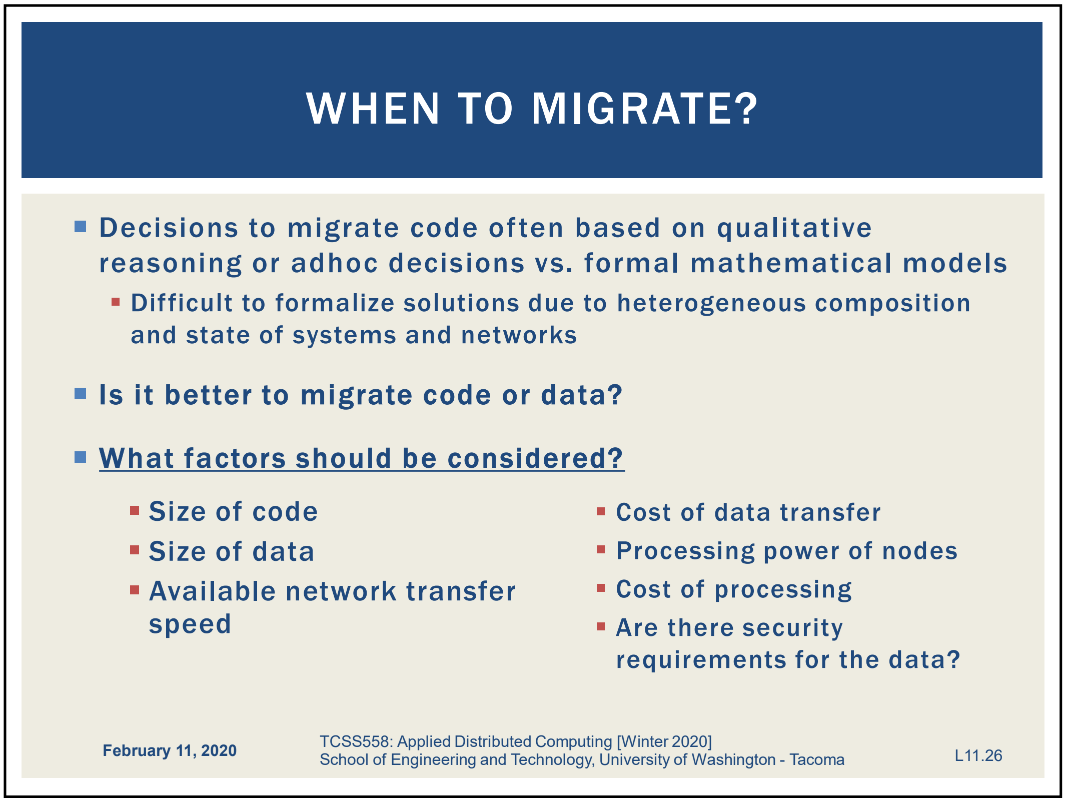
24



# LOAD DISTRIBUTION ALGORITHMS

- Make decisions concerning allocation and redistribution of tasks across machines
- Provide resource management for compute intensive systems
- Often CPU centric
  - Algorithms should also account for other resources
  - Network capacity may be larger bottleneck that CPU capacity

25



# WHEN TO MIGRATE?

- Decisions to migrate code often based on qualitative reasoning or adhoc decisions vs. formal mathematical models
  - Difficult to formalize solutions due to heterogeneous composition and state of systems and networks
- Is it better to migrate code or data?
- <u>What factors should be considered?</u>
  - Size of code
  - Size of data
  - Available network transfer speed
  - Cost of data transfer
  - Processing power of nodes
  - Cost of processing
  - Are there security requirements for the data?

26

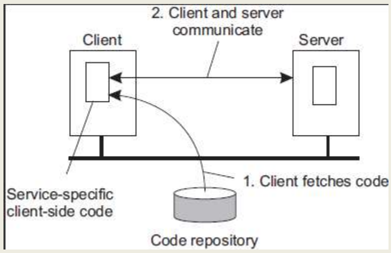
# APPROACHES TO CODE MIGRATION

- Traditional clients
  - Client interacts with server using specific protocol
  - Tight coupling of client->server limits system flexibility
  - Difficult to change protocol when there are ***<u>many</u>*** clients
- Dynamic web clients
  - Web browser downloads client code immediately before use
  - New versions can readily be distributed

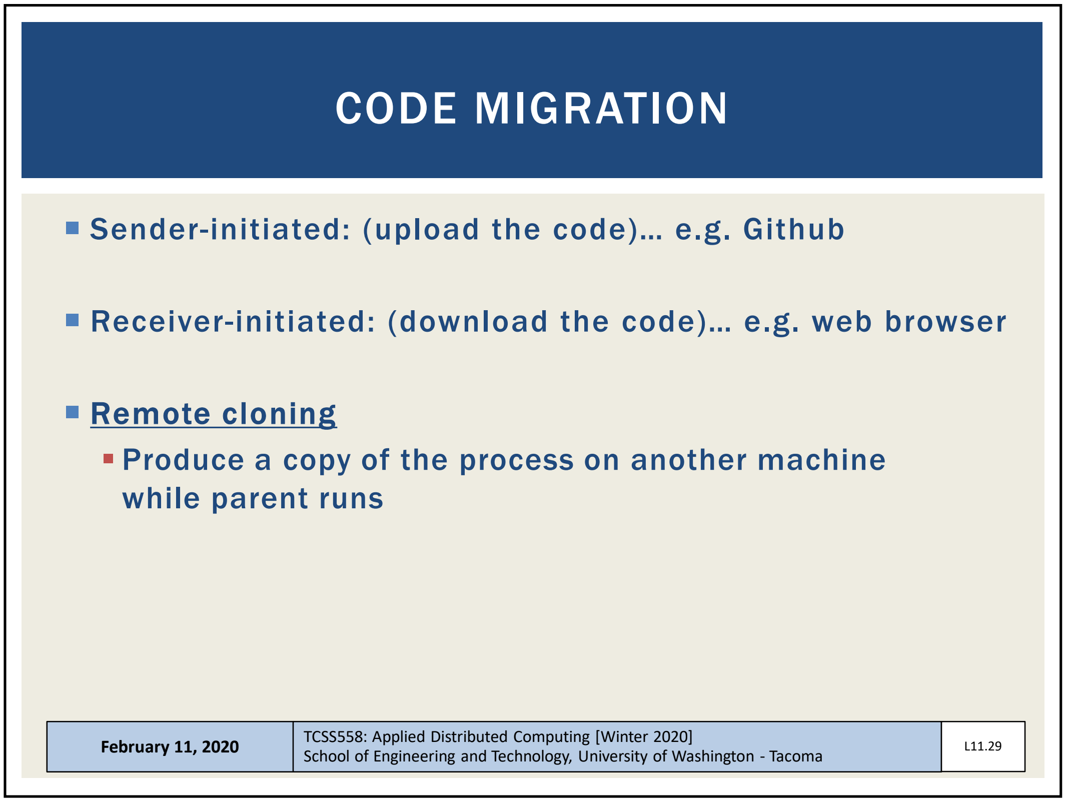
27

# DYNAMIC WEB CLIENTS

- Advantages
  - Client code loaded in as necessary
  - Discarded when no longer needed
  - Can easily change the client/server protocol
- Disadvantages
  - Security: we have to trust the code
  - Downloading client requires network bandwidth & time


2. Client and server communicate
Client
Server
1. Client fetches code
Service-specific client-side code
Code repository

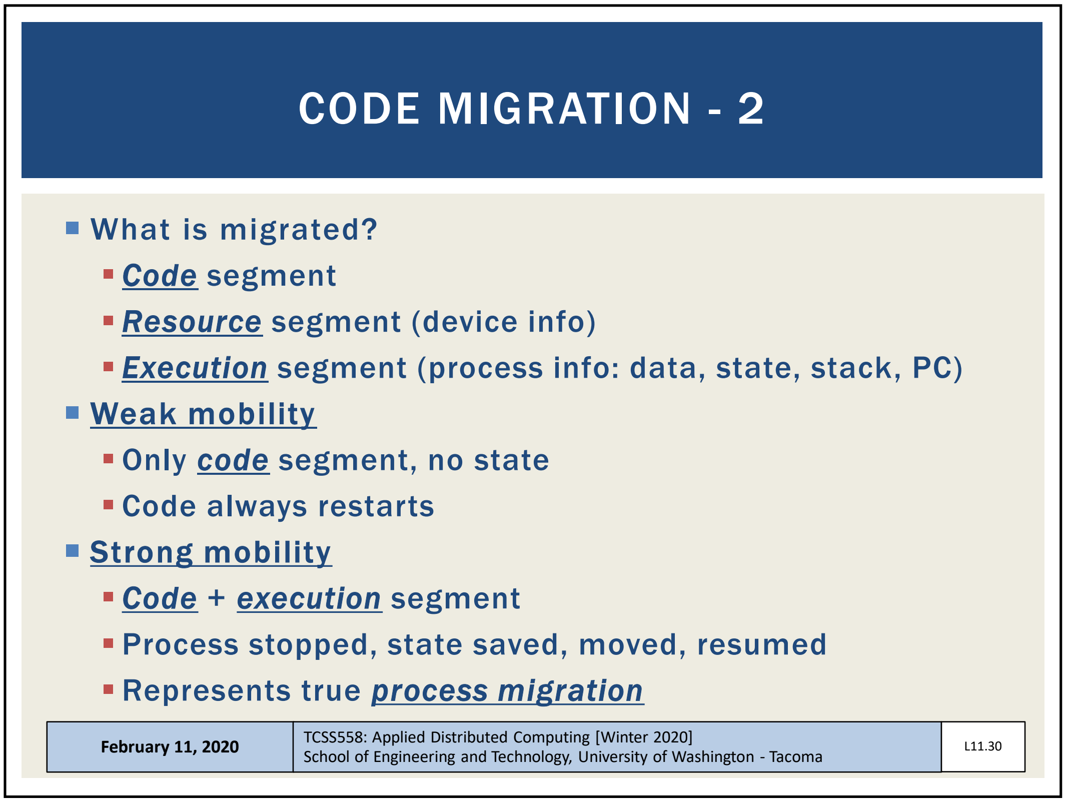
28



## CODE MIGRATION

- Sender-initiated: (upload the code)… e.g. Github
- Receiver-initiated: (download the code)… e.g. web browser
- Remote cloning
  - Produce a copy of the process on another machine while parent runs

29



## CODE MIGRATION - 2

- What is migrated?
  - *Code* segment
  - *Resource* segment (device info)
  - *Execution* segment (process info: data, state, stack, PC)
- Weak mobility
  - Only *code* segment, no state
  - Code always restarts
- Strong mobility
  - *Code* + *execution* segment
  - Process stopped, state saved, moved, resumed
  - Represents true *process migration*

30

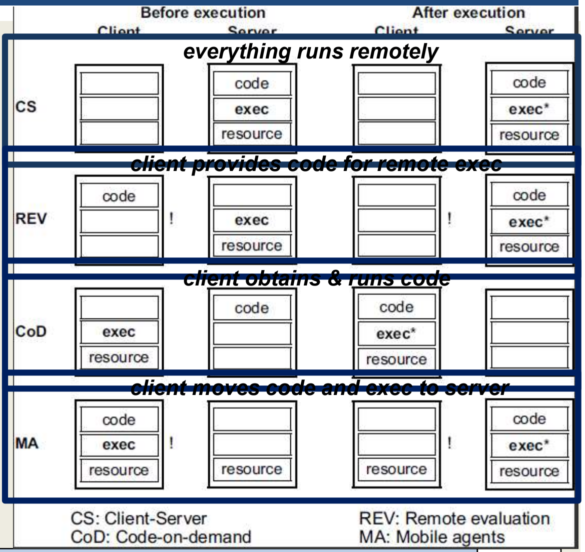
# CODE MOBILITY TYPES

* indicates what is modified

- CS: Client-Server
- REV: Remote Evaluation
- CoD: Code-on-demand
- MA: Mobile agents

- Where does state get modified?

- State is stored in <u>exec</u>



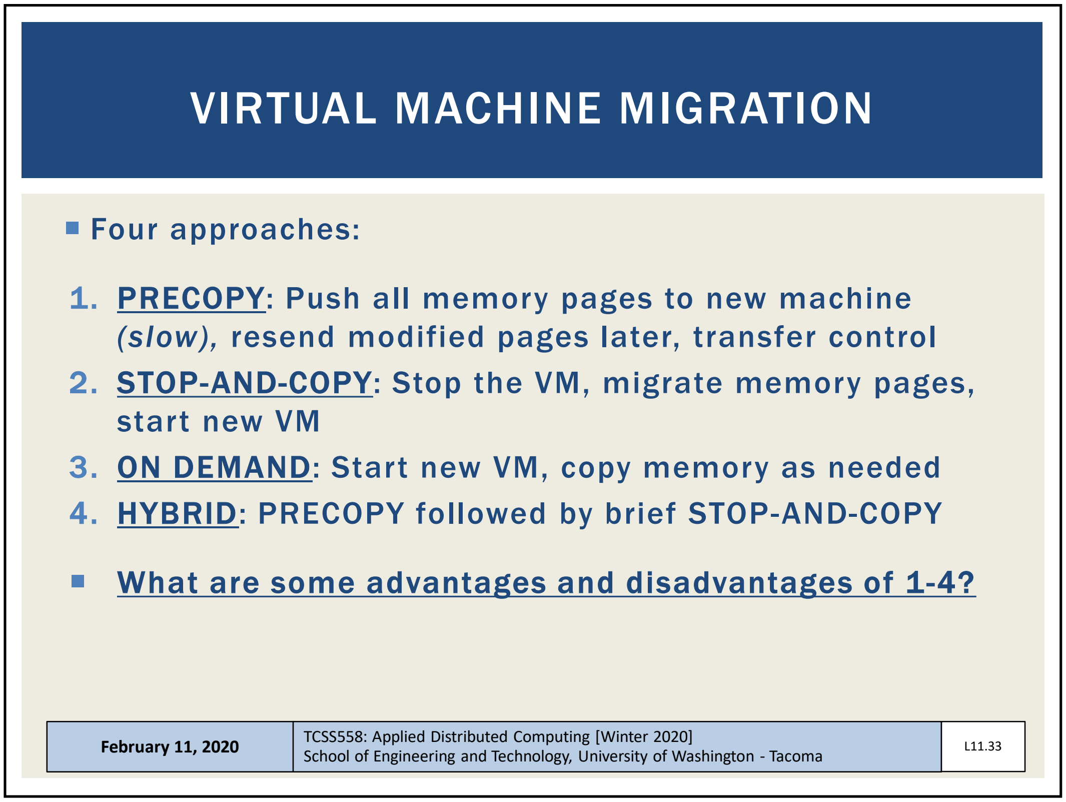
# MIGRATION OF HETEROGENEOUS SYSTEMS

- Assumption: code will always work at new node
- Invalid if node architecture is different (*heterogeneous*)

- What approaches are available to migrate code across heterogeneous systems?

- Intermediate code
  - 1970s Pascal: generate machine-independent intermediate code
  - Programs could then run anywhere
  - <u>Today:</u> web languages: Javascript, Java

- VM Migration



# VIRTUAL MACHINE MIGRATION

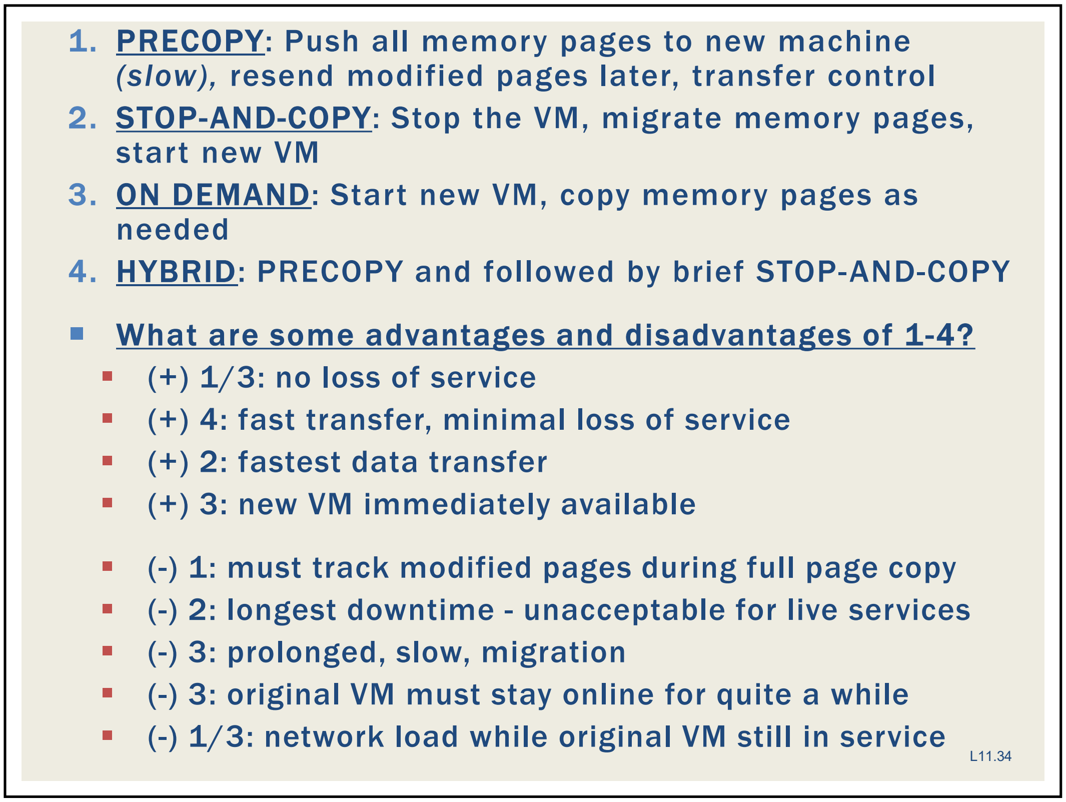
- Four approaches:

1. PRECOPY: Push all memory pages to new machine *(slow)*, resend modified pages later, transfer control
2. STOP-AND-COPY: Stop the VM, migrate memory pages, start new VM
3. ON DEMAND: Start new VM, copy memory as needed
4. HYBRID: PRECOPY followed by brief STOP-AND-COPY

- What are some advantages and disadvantages of 1-4?

February 11, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L11.33

33

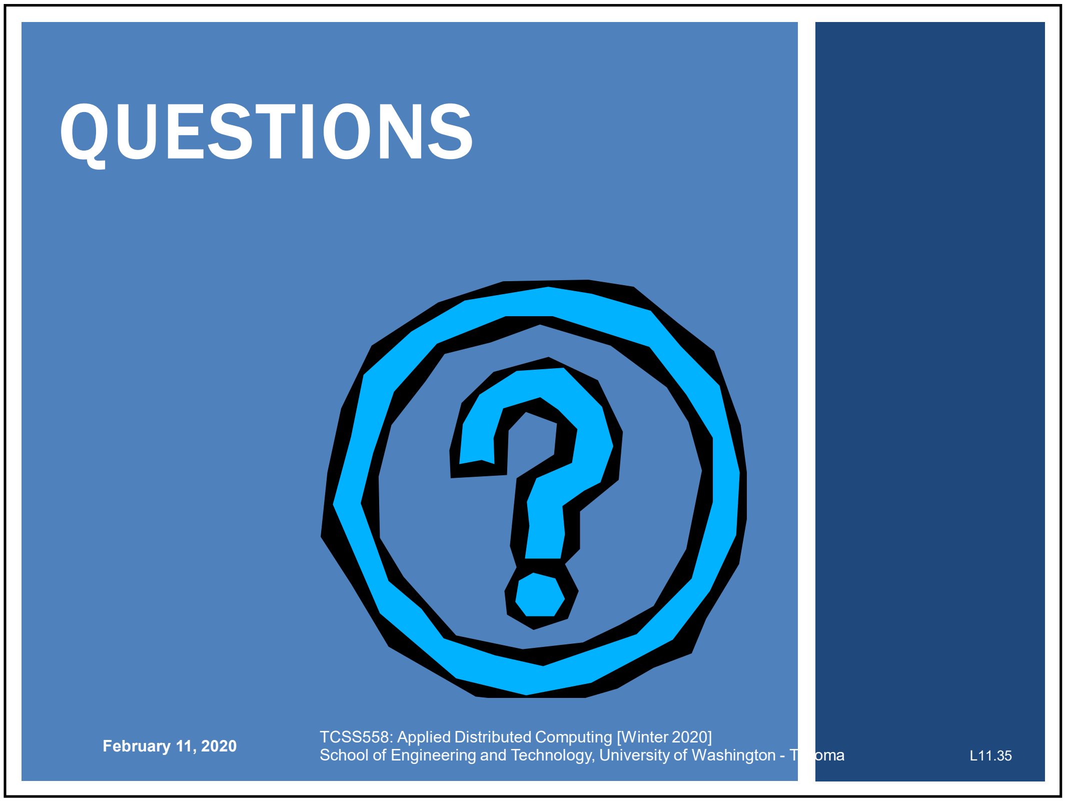


1. PRECOPY: Push all memory pages to new machine *(slow)*, resend modified pages later, transfer control
2. STOP-AND-COPY: Stop the VM, migrate memory pages, start new VM
3. ON DEMAND: Start new VM, copy memory pages as needed
4. HYBRID: PRECOPY and followed by brief STOP-AND-COPY

- What are some advantages and disadvantages of 1-4?
  - (+) 1/3: no loss of service
  - (+) 4: fast transfer, minimal loss of service
  - (+) 2: fastest data transfer
  - (+) 3: new VM immediately available
  - (-) 1: must track modified pages during full page copy
  - (-) 2: longest downtime - unacceptable for live services
  - (-) 3: prolonged, slow, migration
  - (-) 3: original VM must stay online for quite a while
  - (-) 1/3: network load while original VM still in service

L11.34

34



# QUESTIONS

February 11, 2020

TCSS558: Applied Distributed Computing [Winter 2020]
School of Engineering and Technology, University of Washington - T oma

L11.35

35

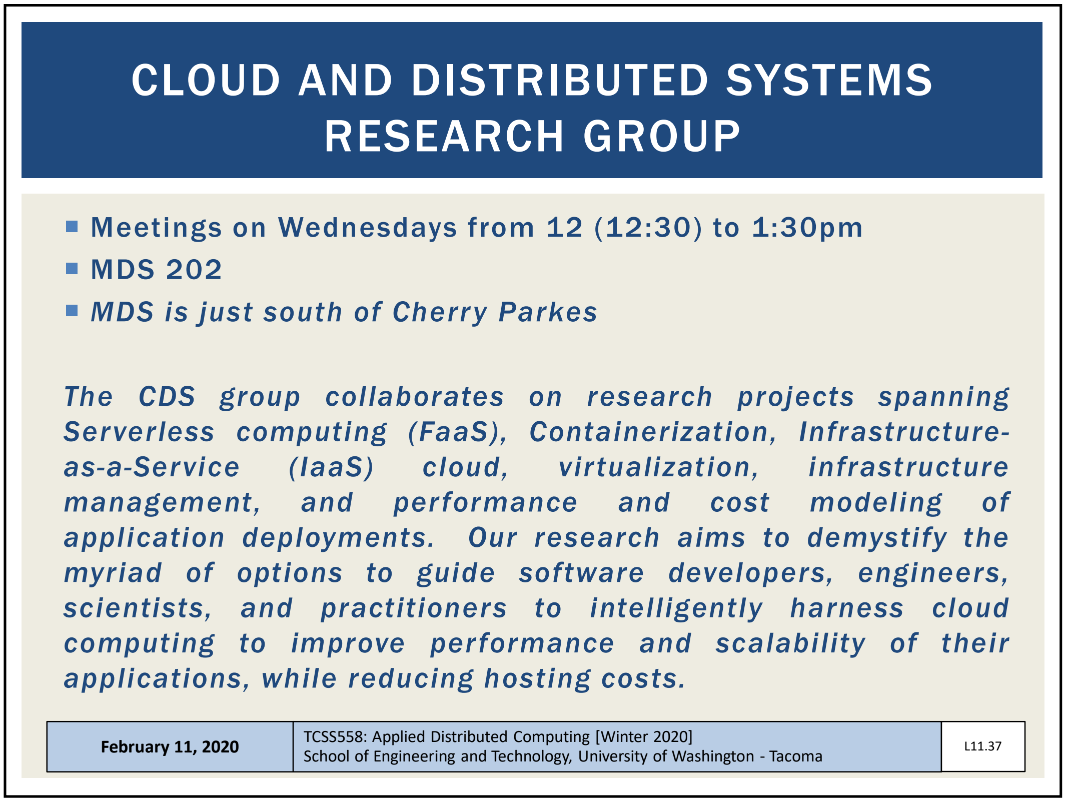
# RESEARCH DIRECTIONS

October 5, 2017

TCSS558: Applied Distributed Computing [Winter 2020]
School of Engineering and Technology, University of Washington - T oma

L11.36

36



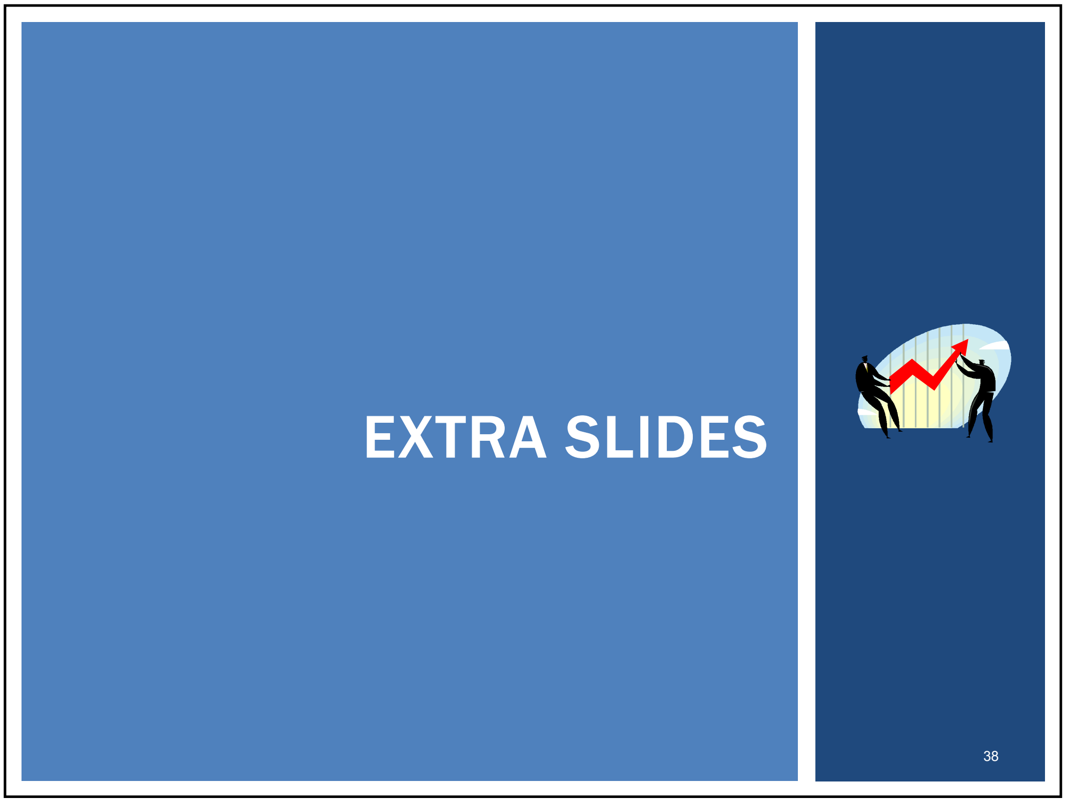
## CLOUD AND DISTRIBUTED SYSTEMS RESEARCH GROUP

- Meetings on Wednesdays from 12 (12:30) to 1:30pm
- MDS 202
- *MDS is just south of Cherry Parkes*

*The CDS group collaborates on research projects spanning Serverless computing (FaaS), Containerization, Infrastructure-as-a-Service (IaaS) cloud, virtualization, infrastructure management, and performance and cost modeling of application deployments. Our research aims to demystify the myriad of options to guide software developers, engineers, scientists, and practitioners to intelligently harness cloud computing to improve performance and scalability of their applications, while reducing hosting costs.*

| **February 11, 2020** | TCSS558: Applied Distributed Computing [Winter 2020]<br>School of Engineering and Technology, University of Washington - Tacoma | L11.37 |
|---|---|---|

37



## EXTRA SLIDES

38

38

# CHORD SYSTEM – FINGER TABLE

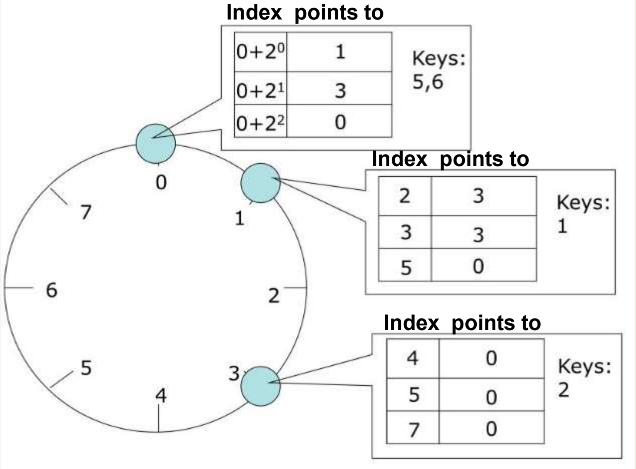
- Each node keeps maintains a finger table with m entries
  - m is the number of bits in the hash key
  - Distance of the entries increases exponentially
- Contents of each node's finger table:

```
for i=0 to m-1
        finger table entry for node n:
        index: $n+2^i$ → points to: $n+2^i \bmod 2^m$
```

- The first entry of finger table is the node's immediate successor (an extra successor field is not needed).
- Each time a node looks up a key k, it passes the query to the closest node to k in the finger table that is not greater than k
- With finger tables, the number of nodes contacted to find a successor in an N-node network is O(log N).

# CHORD SYSTEM – 2

- Keys have m-bits
- m=3
- Always pass query for key k to index in the finger table that is not greater than k
- Example: key (k=7)
- Query arrives at (0)
  - 0: → (index=4, pass to 0), key 7 is adjacent



| Index | points to |
|---|---|
| $0+2^0$ | 1 |
| $0+2^1$ | 3 |
| $0+2^2$ | 0 |

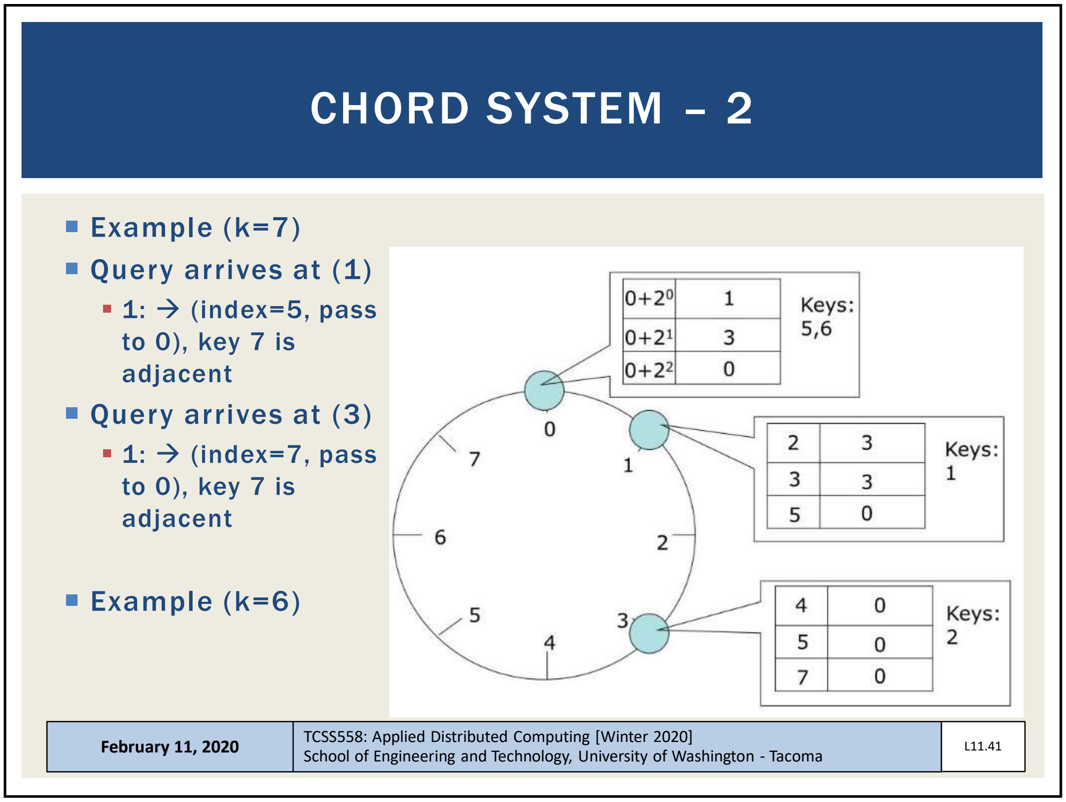
Keys: 5,6

| Index | points to |
|---|---|
| 2 | 3 |
| 3 | 3 |
| 5 | 0 |

Keys: 1

| Index | points to |
|---|---|
| 4 | 0 |
| 5 | 0 |
| 7 | 0 |

Keys: 2



# CHORD SYSTEM – 2

- Example (k=7)
- Query arrives at (1)
  - 1: → (index=5, pass to 0), key 7 is adjacent
- Query arrives at (3)
  - 1: → (index=7, pass to 0), key 7 is adjacent
- Example (k=6)

| | |
|---|---|
| $0+2^0$ | 1 |
| $0+2^1$ | 3 |
| $0+2^2$ | 0 |

Keys: 5,6

| | |
|---|---|
| 2 | 3 |
| 3 | 3 |
| 5 | 0 |

Keys: 1

| | |
|---|---|
| 4 | 0 |
| 5 | 0 |
| 7 | 0 |

Keys: 2