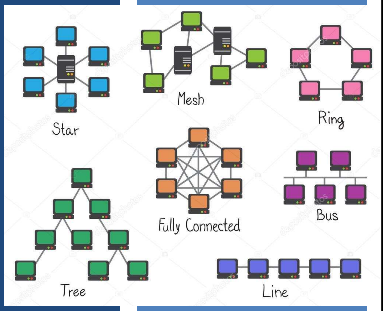
TCSS 558: APPLIED DISTRIBUTED COMPUTING

Distributed Systems: Types and Architectures

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

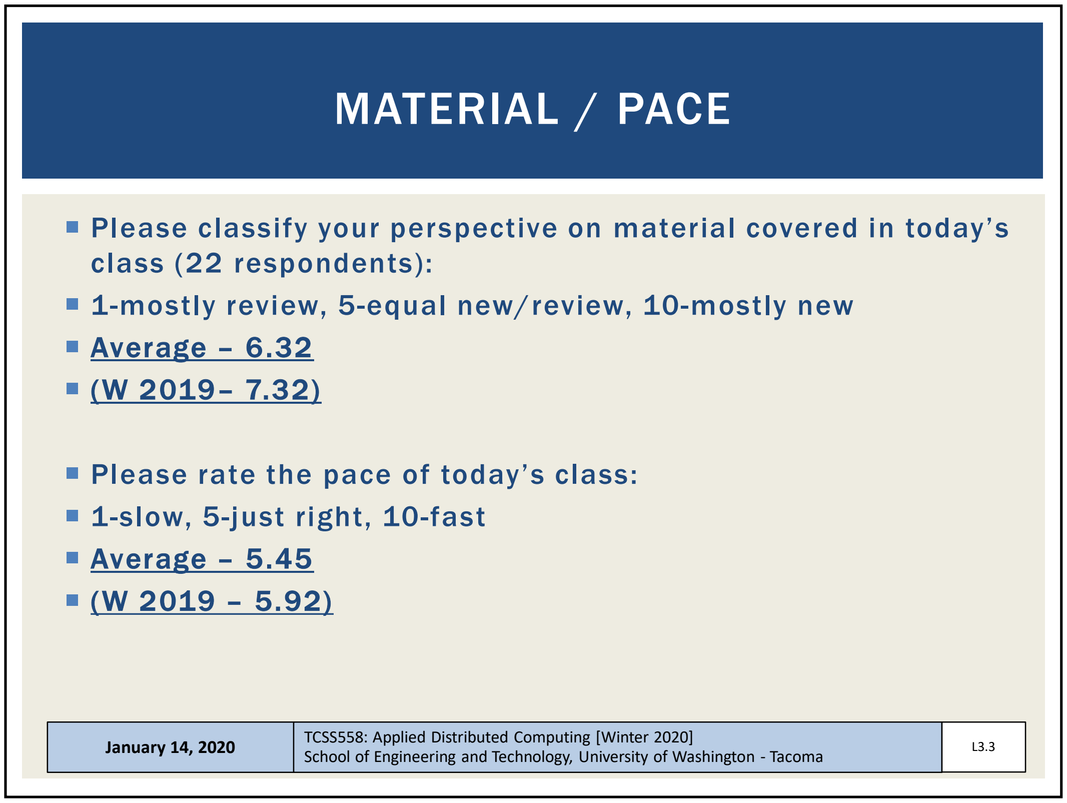


1

OBJECTIVES

- Feedback from 1/9
- Types of distributed systems (Ch. 1.3)
  - HPC, cluster, grid, cloud
  - Distributed information systems
  - Pervasive systems
- Homework 0
- Research directions
  - Serverless Computing
  - Containerization
  - Infrastructure-as-a-Service
  - Others
- Chapter 2: Distributed System Architectures

2



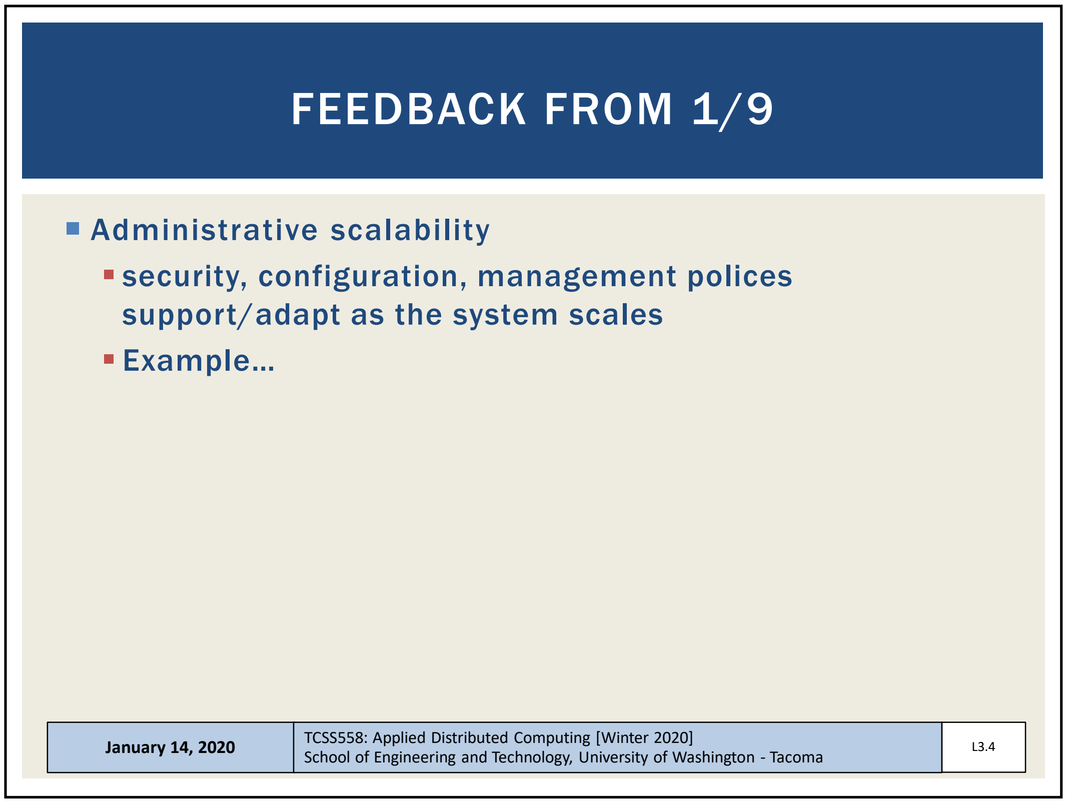
MATERIAL / PACE

- Please classify your perspective on material covered in today's class (22 respondents):
- 1-mostly review, 5-equal new/review, 10-mostly new
- Average – 6.32
- (W 2019– 7.32)

- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- Average – 5.45
- (W 2019 – 5.92)

January 14, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L3.3

3



FEEDBACK FROM 1/9

- Administrative scalability
  - security, configuration, management polices support/adapt as the system scales
  - Example…

January 14, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L3.4

4

ADMINISTRATIVE SCALABILITY EXAMPLE

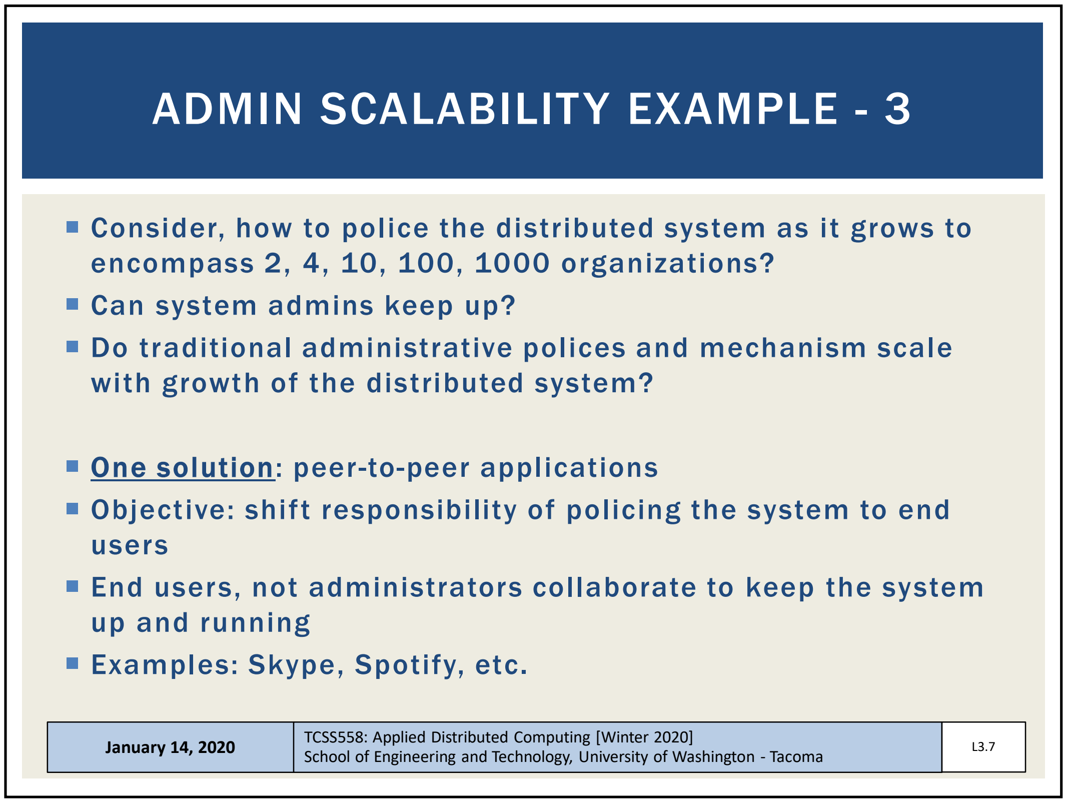
- Scientists share equipment (often expensive) using a computational GRID
- GRID provides global distributed system that federates resources from several organizations local systems:
  - Organization A: University of Washington
  - Organization B: UC Berkeley
- Components within a distributed system can be trusted by users within an organization (e.g. UW)
- Local system admins test and certified applications, and have taken special measures to ensure systems cannot be tampered with
  - Users trust system admins to ensure security
  - However trust does not expand across domain boundaries

5

ADMIN SCALABILITY EXAMPLE - 2

- ***WHAT IS ADMINISTRATIVE SCALABILITY IN THIS CONTEXT?***
- **(1) Distributed system must protect from malicious attacks from new domains**
- For example:
  - Users from new domains may have only read access to file systems
  - Facilities such as high-performance computers may not be made available to unauthorized users
- (2) **New domains must protect from malicious attacks from the distributed system**
- For example:
  - Users in new domain downloading content from distributed system must ensure safety of content
  - E.g. web applets, etc.

6



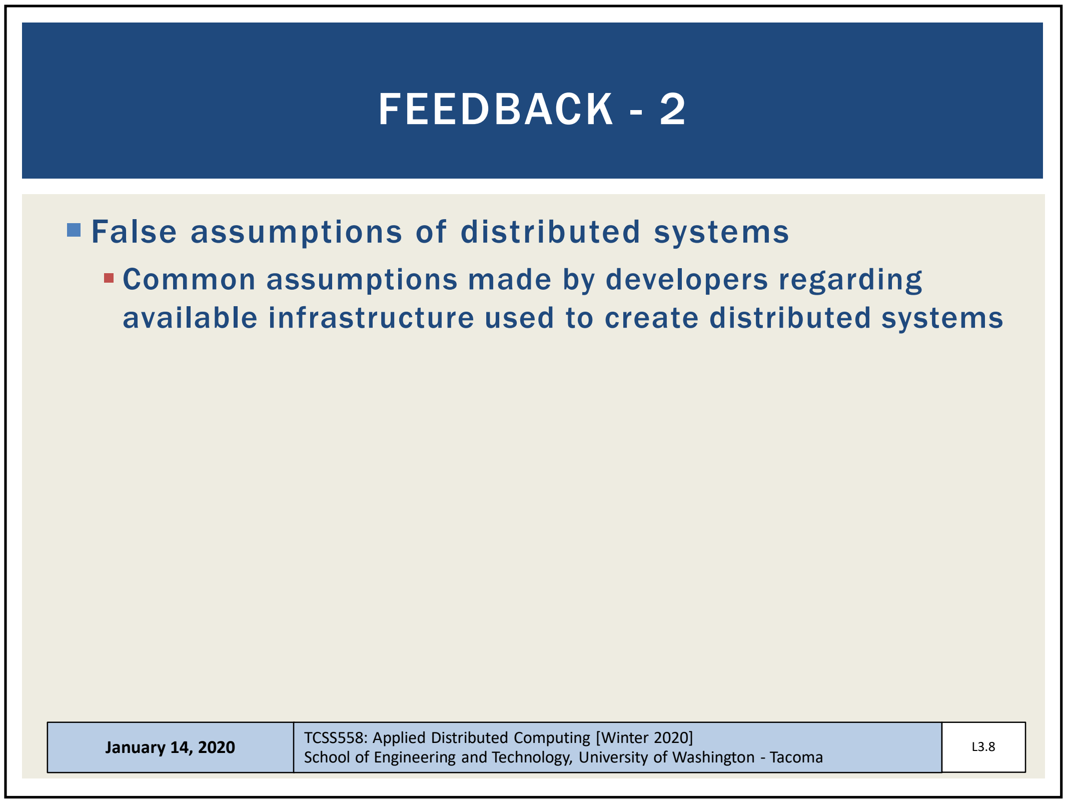
ADMIN SCALABILITY EXAMPLE - 3

- Consider, how to police the distributed system as it grows to encompass 2, 4, 10, 100, 1000 organizations?
- Can system admins keep up?
- Do traditional administrative polices and mechanism scale with growth of the distributed system?

- <u>One solution</u>: peer-to-peer applications
- Objective: shift responsibility of policing the system to end users
- End users, not administrators collaborate to keep the system up and running
- Examples: Skype, Spotify, etc.

January 14, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L3.7

7



FEEDBACK - 2

- False assumptions of distributed systems
  - Common assumptions made by developers regarding available infrastructure used to create distributed systems

January 14, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L3.8

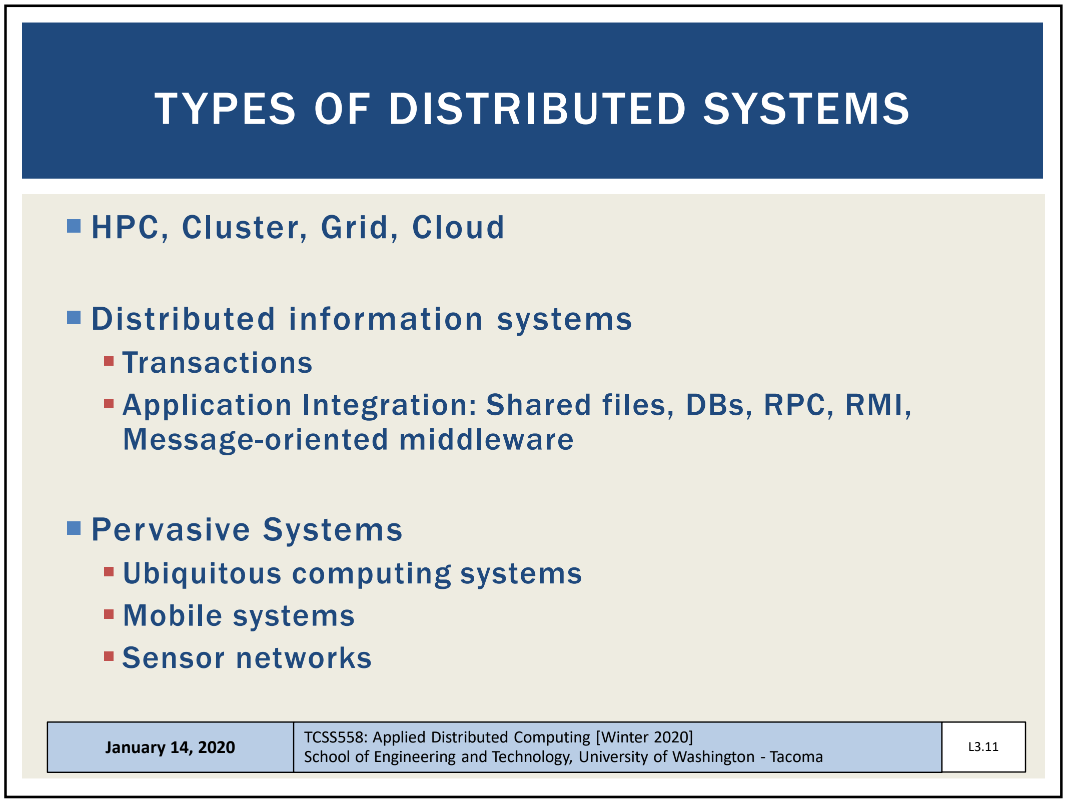
8

FALSE ASSUMPTIONS ABOUT DISTRIBUTED SYSTEMS

- The network is reliable
- The network is secure
- The network is homogeneous
- The topology does not change
- Latency is zero
- Bandwidth is infinite
- Transport cost is zero
- There is one administrator

FEEDBACK - 3

- Can we record the lecture?
  - YES!, certainly

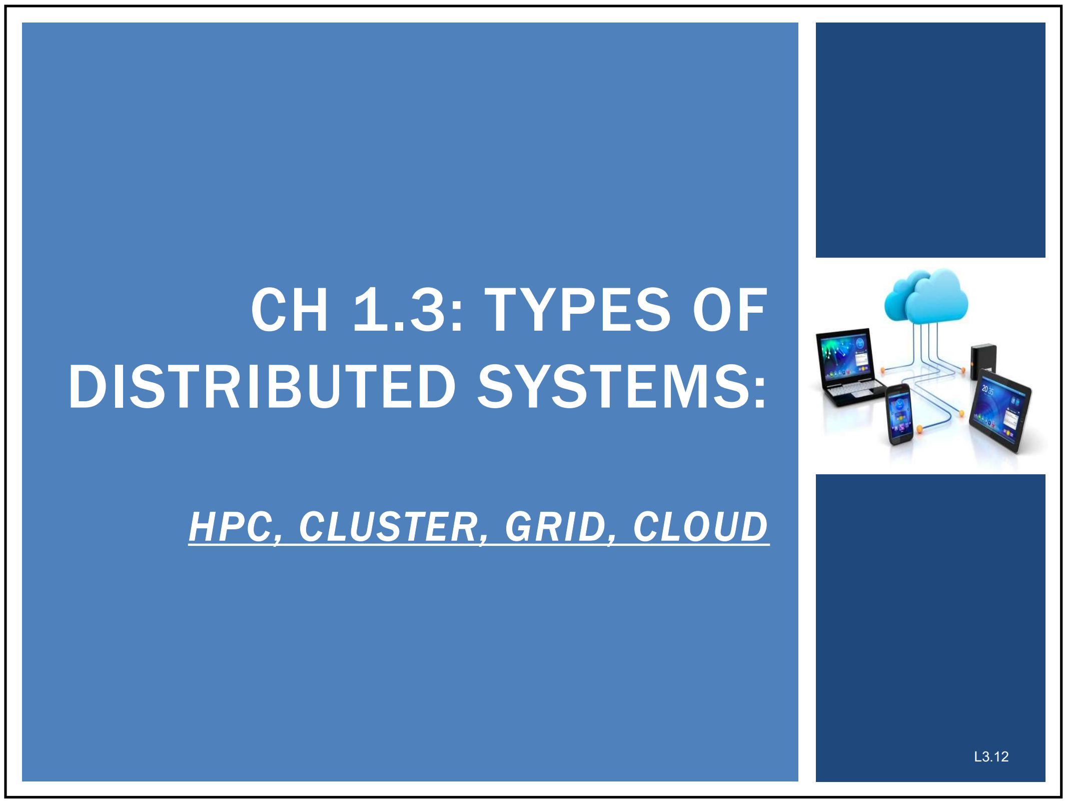


TYPES OF DISTRIBUTED SYSTEMS

- HPC, Cluster, Grid, Cloud
- Distributed information systems
  - Transactions
  - Application Integration: Shared files, DBs, RPC, RMI, Message-oriented middleware
- Pervasive Systems
  - Ubiquitous computing systems
  - Mobile systems
  - Sensor networks

January 14, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L3.11

11



CH 1.3: TYPES OF DISTRIBUTED SYSTEMS:

HPC, CLUSTER, GRID, CLOUD

L3.12

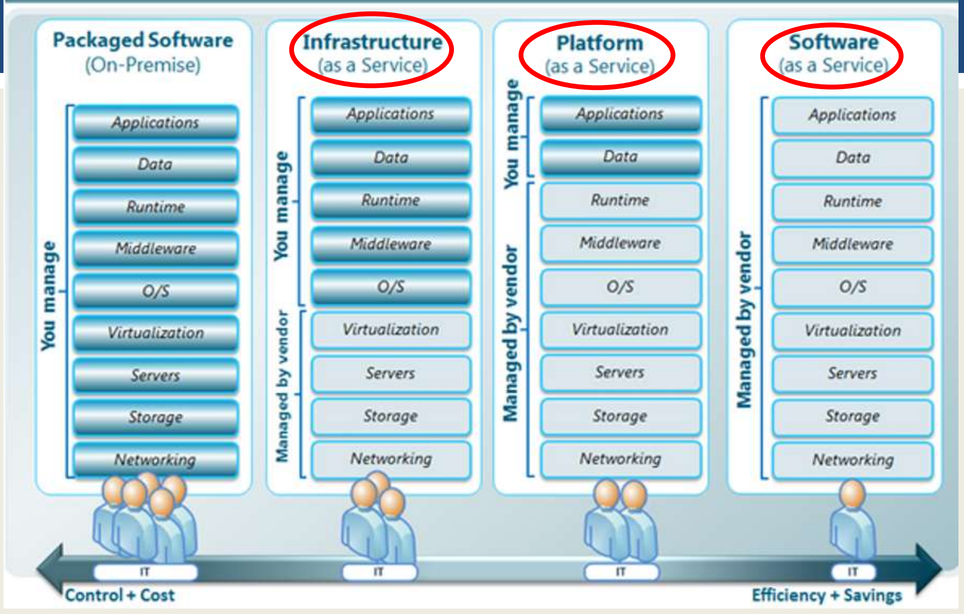
12

TECHNOLOGY INNOVATIONS LEADING TO CLOUD COMPUTING

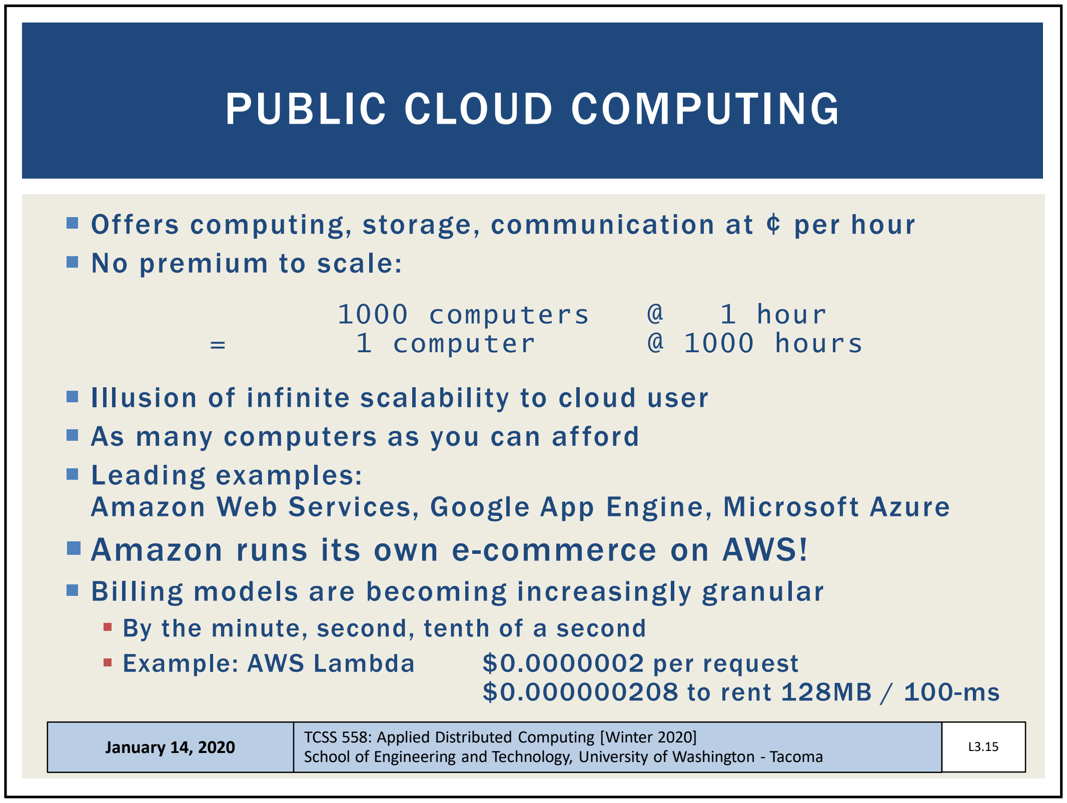
- Super computers
  - Huge multiprocessor system which shares RAM
  - Technically "not distributed"
  - Hardware all in one location
- High performance distributed computing
  - Cluster computing
  - Grid computing
  - Cloud computing
  - Virtualization
  - Others

13

Cloud Services Architecture



14



PUBLIC CLOUD COMPUTING

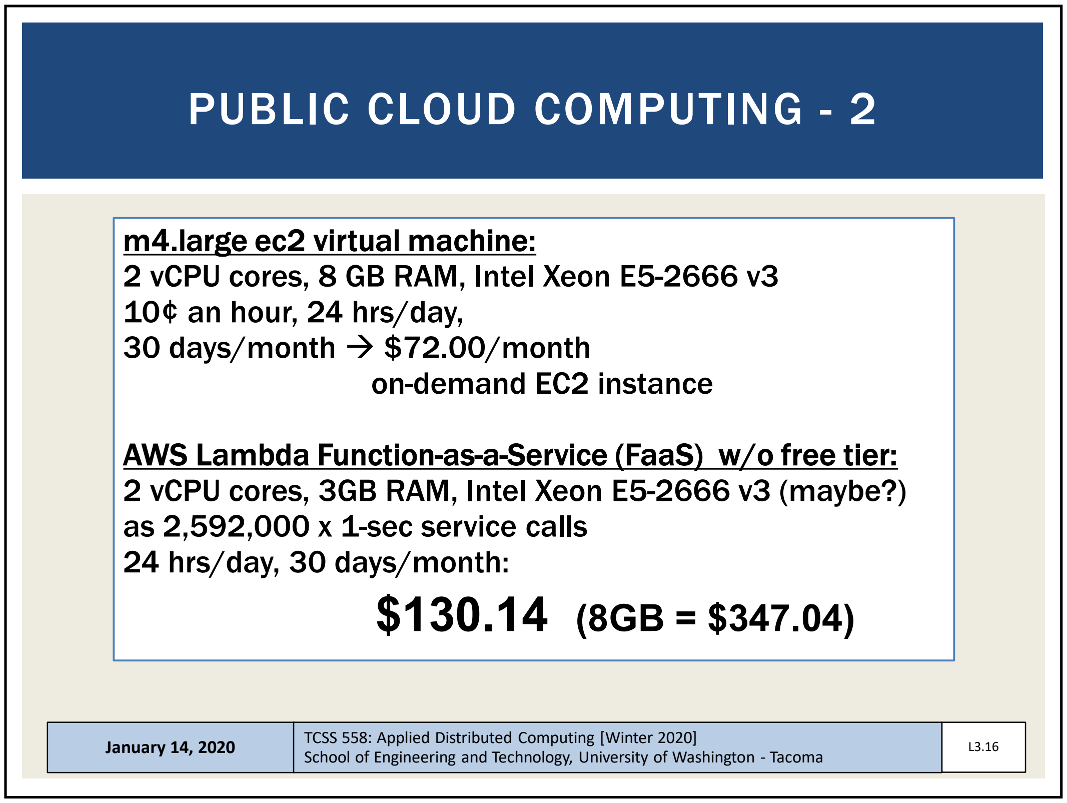
- Offers computing, storage, communication at ¢ per hour
- No premium to scale:

```
         1000 computers   @   1 hour
  =        1 computer     @ 1000 hours
```

- Illusion of infinite scalability to cloud user
- As many computers as you can afford
- Leading examples:
  Amazon Web Services, Google App Engine, Microsoft Azure
- Amazon runs its own e-commerce on AWS!
- Billing models are becoming increasingly granular
  - By the minute, second, tenth of a second
  - Example: AWS Lambda $0.0000002 per request
    $0.000000208 to rent 128MB / 100-ms

January 14, 2020	TCSS 558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L3.15

15



PUBLIC CLOUD COMPUTING - 2

m4.large ec2 virtual machine:
2 vCPU cores, 8 GB RAM, Intel Xeon E5-2666 v3
10¢ an hour, 24 hrs/day,
30 days/month → $72.00/month
on-demand EC2 instance

AWS Lambda Function-as-a-Service (FaaS) w/o free tier:
2 vCPU cores, 3GB RAM, Intel Xeon E5-2666 v3 (maybe?)
as 2,592,000 x 1-sec service calls
24 hrs/day, 30 days/month:
$130.14 **(8GB = $347.04)**

January 14, 2020	TCSS 558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L3.16

16

PAAS SERVICES IMPLEMENTATION

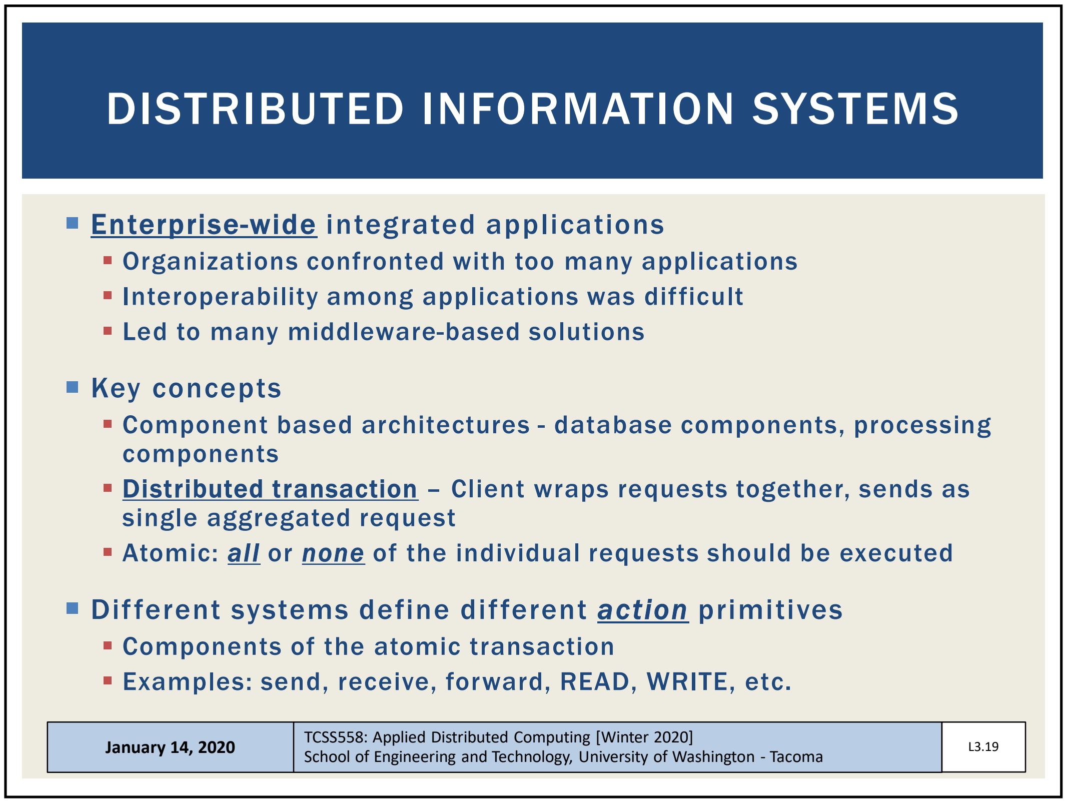
- PaaS services often built atop of IaaS
  - Amazon RDS, Heroku, Amazon Elasticache
- Scalability
  - VM resources can support fluctuations in demand
- Dependability.
  - PaaS services built on highly available IaaS resources

17

CH 1.3: TYPES OF DISTRIBUTED SYSTEMS:

DISTRIBUTED INFORMATION SYSTEMS

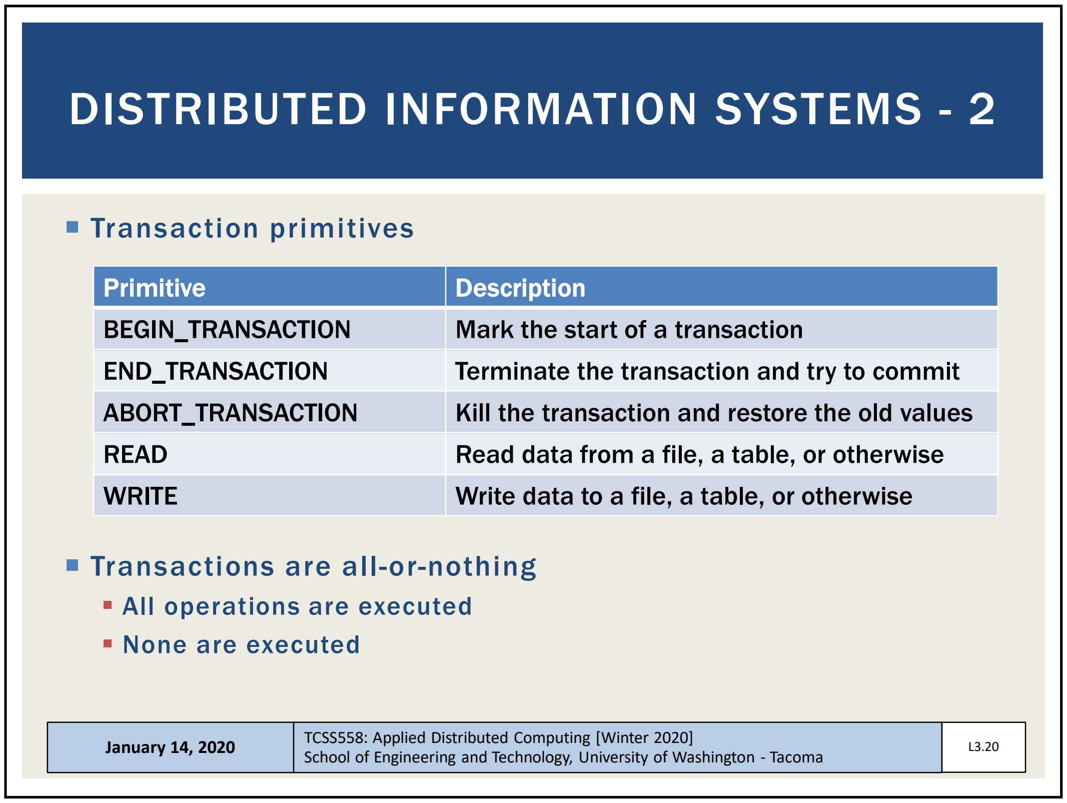
18



DISTRIBUTED INFORMATION SYSTEMS

- <u>**Enterprise-wide**</u> integrated applications
  - Organizations confronted with too many applications
  - Interoperability among applications was difficult
  - Led to many middleware-based solutions
- Key concepts
  - Component based architectures - database components, processing components
  - <u>**Distributed transaction**</u> – Client wraps requests together, sends as single aggregated request
  - Atomic: <u>***all***</u> or <u>***none***</u> of the individual requests should be executed
- Different systems define different <u>***action***</u> primitives
  - Components of the atomic transaction
  - Examples: send, receive, forward, READ, WRITE, etc.

19



DISTRIBUTED INFORMATION SYSTEMS - 2

- Transaction primitives

Primitive	Description
BEGIN_TRANSACTION	Mark the start of a transaction
END_TRANSACTION	Terminate the transaction and try to commit
ABORT_TRANSACTION	Kill the transaction and restore the old values
READ	Read data from a file, a table, or otherwise
WRITE	Write data to a file, a table, or otherwise

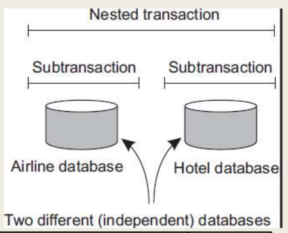
- Transactions are all-or-nothing
  - All operations are executed
  - None are executed

20

TRANSACTIONS: ACID PROPERTIES

- Atomic: The transaction occurs indivisibly
- Consistent: Transaction does not create variant states across nodes during slow updates (e.g. system variants)
  - Replicas remain constant until all updated
  - Two phase commit: data pushed first, then the commit
- Isolated: Transactions do not interfere with each other
- Durable: Once a transaction commits, change are permanent

- Nested transaction: transaction constructed with many sub-transactions
- Follows a logical division of work
- Must support "rollback" of sub-transactions

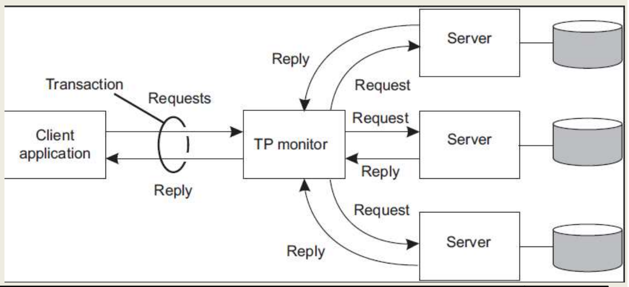


January 14, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L3.21

21

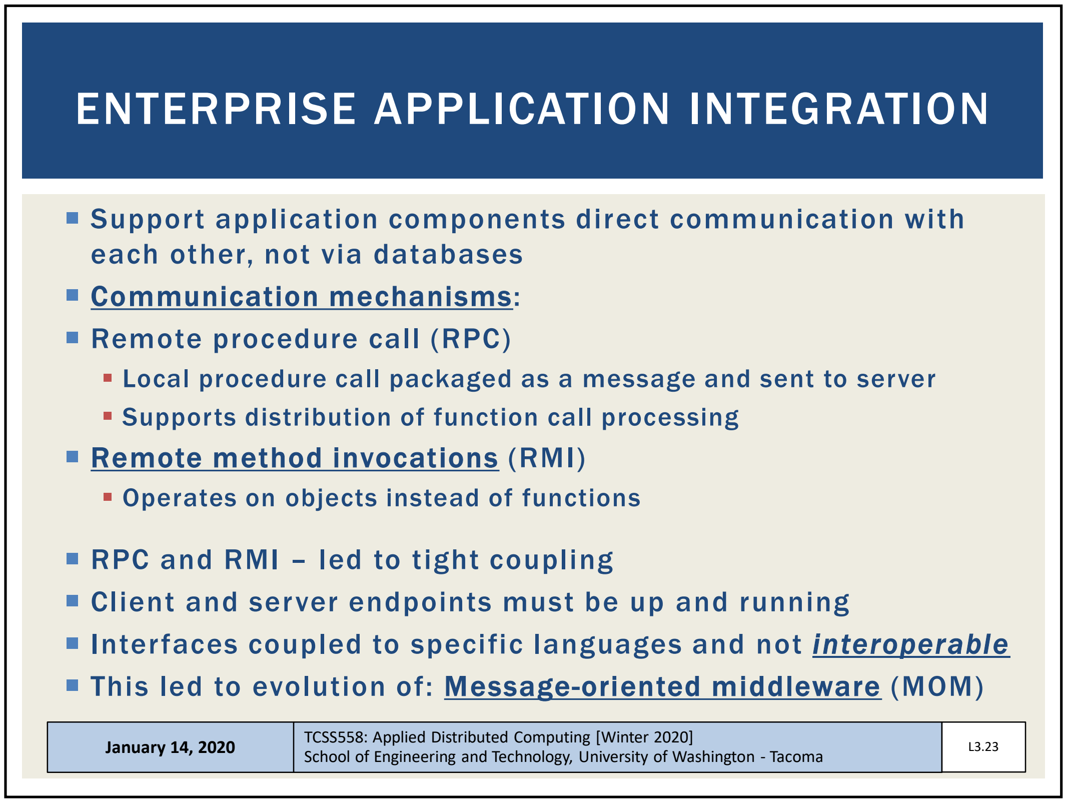
TRANSACTION PROCESSING MONITOR

- Allow an application to access multiple DBs via a transactional programming model
- TP monitor: coordinates commitment of sub-transactions using a distributed commit protocol (Ch. 8)
- Save application complexity from having to coordinate



January 14, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L3.22

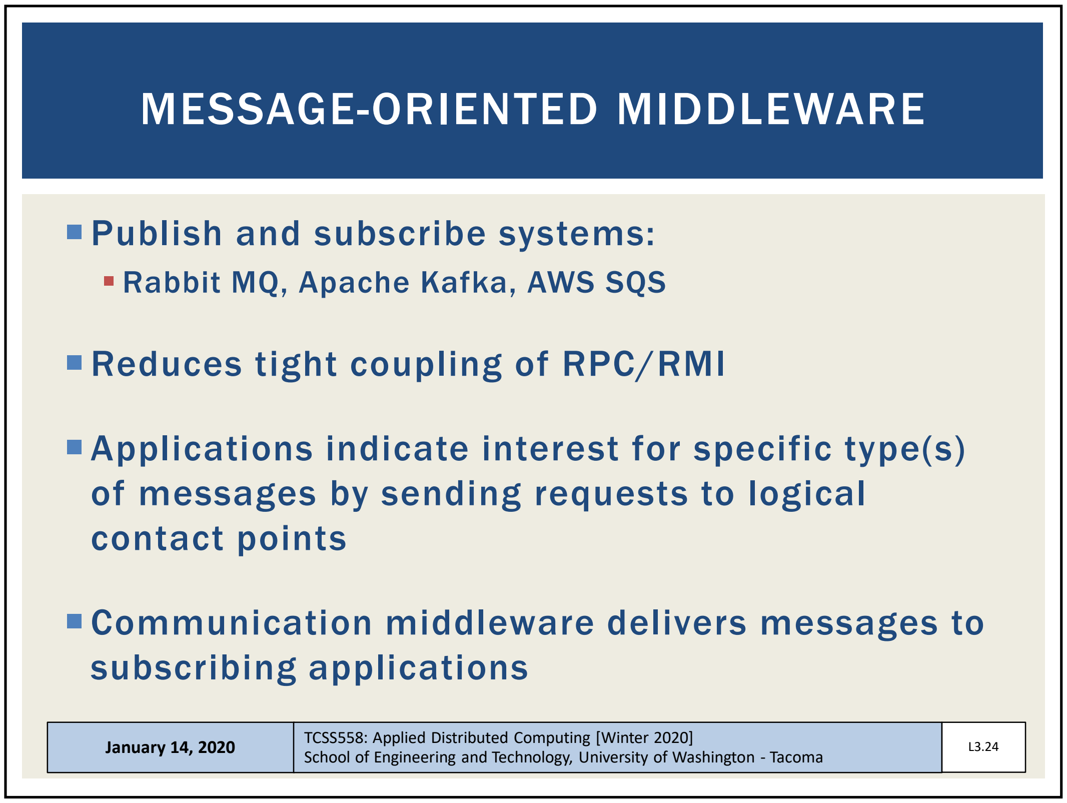
22



ENTERPRISE APPLICATION INTEGRATION

- Support application components direct communication with each other, not via databases
- <u>Communication mechanisms</u>:
- Remote procedure call (RPC)
  - Local procedure call packaged as a message and sent to server
  - Supports distribution of function call processing
- <u>Remote method invocations</u> (RMI)
  - Operates on objects instead of functions
- RPC and RMI – led to tight coupling
- Client and server endpoints must be up and running
- Interfaces coupled to specific languages and not ***<u>interoperable</u>***
- This led to evolution of: <u>Message-oriented middleware</u> (MOM)

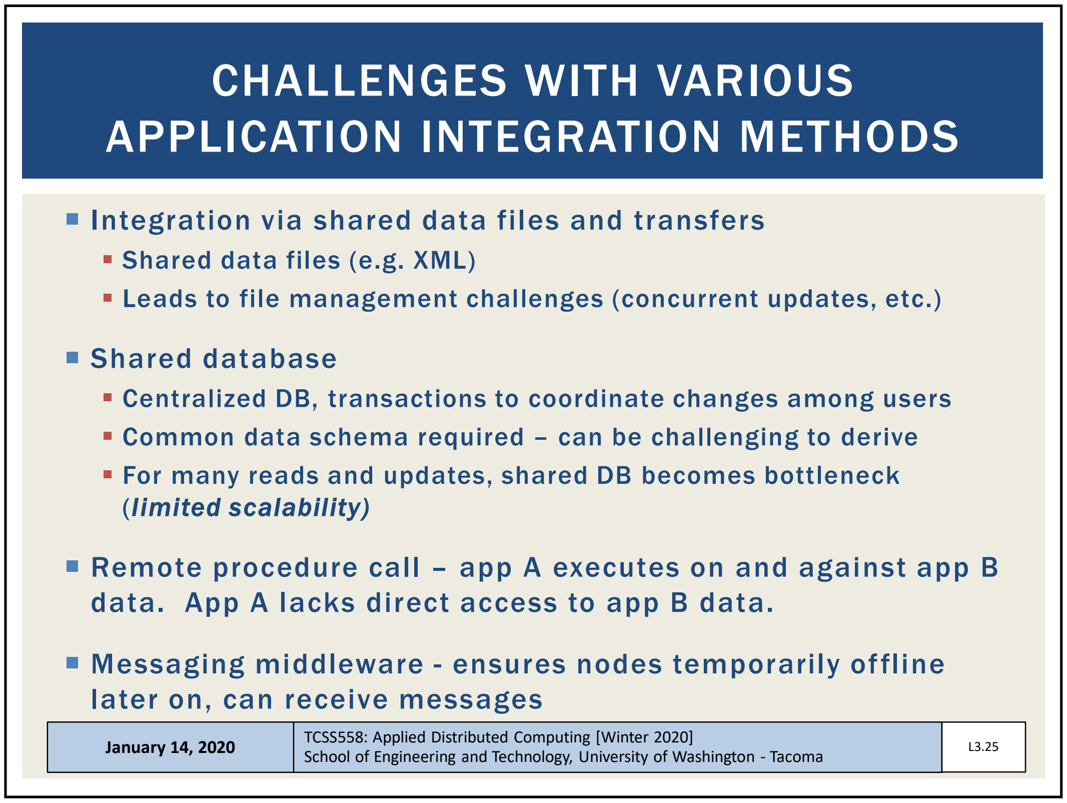
23



MESSAGE-ORIENTED MIDDLEWARE

- Publish and subscribe systems:
  - Rabbit MQ, Apache Kafka, AWS SQS
- Reduces tight coupling of RPC/RMI
- Applications indicate interest for specific type(s) of messages by sending requests to logical contact points
- Communication middleware delivers messages to subscribing applications

24

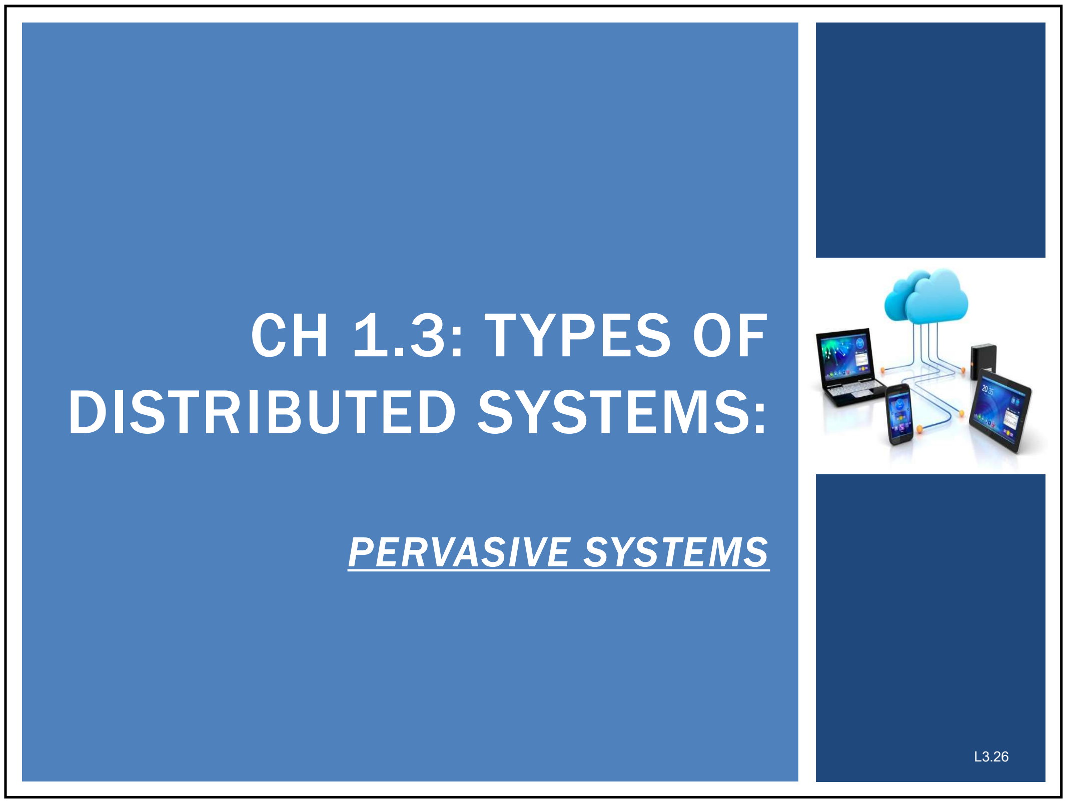


CHALLENGES WITH VARIOUS APPLICATION INTEGRATION METHODS

- Integration via shared data files and transfers
  - Shared data files (e.g. XML)
  - Leads to file management challenges (concurrent updates, etc.)
- Shared database
  - Centralized DB, transactions to coordinate changes among users
  - Common data schema required – can be challenging to derive
  - For many reads and updates, shared DB becomes bottleneck (***limited scalability***)
- Remote procedure call – app A executes on and against app B data. App A lacks direct access to app B data.
- Messaging middleware - ensures nodes temporarily offline later on, can receive messages

January 14, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L3.25

25

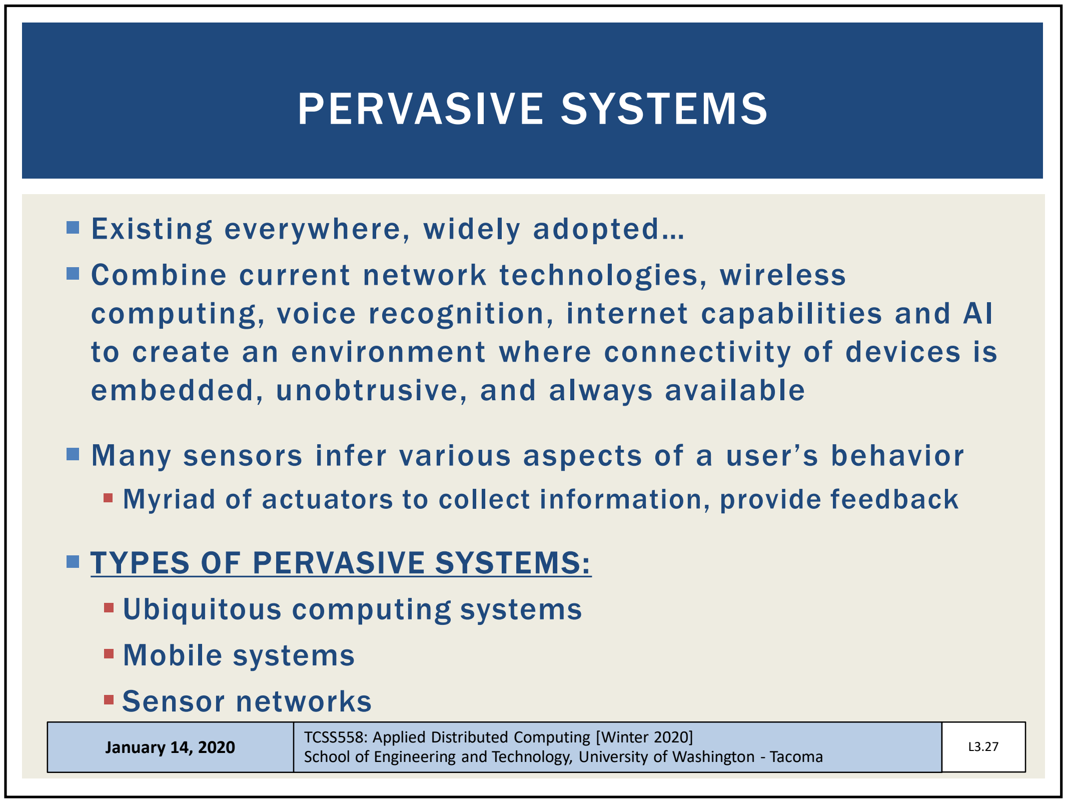


CH 1.3: TYPES OF DISTRIBUTED SYSTEMS:

PERVASIVE SYSTEMS

L3.26

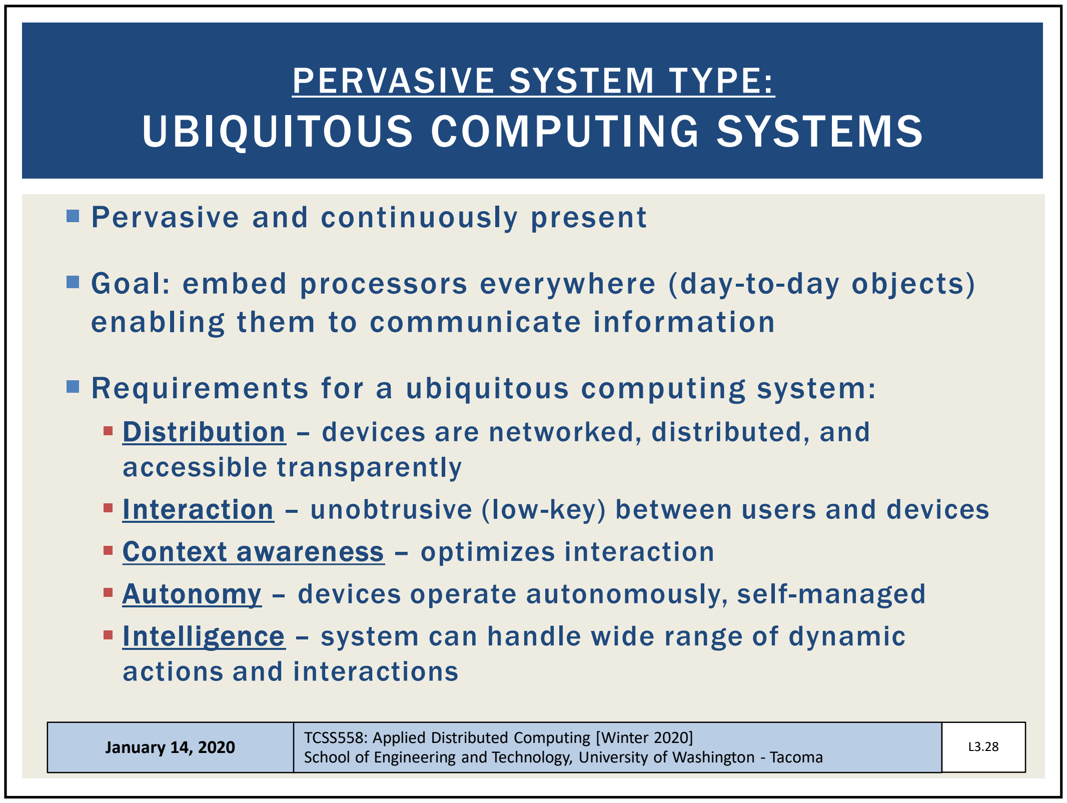
26



PERVASIVE SYSTEMS

- Existing everywhere, widely adopted…
- Combine current network technologies, wireless computing, voice recognition, internet capabilities and AI to create an environment where connectivity of devices is embedded, unobtrusive, and always available
- Many sensors infer various aspects of a user's behavior
  - Myriad of actuators to collect information, provide feedback
- **TYPES OF PERVASIVE SYSTEMS:**
  - Ubiquitous computing systems
  - Mobile systems
  - Sensor networks

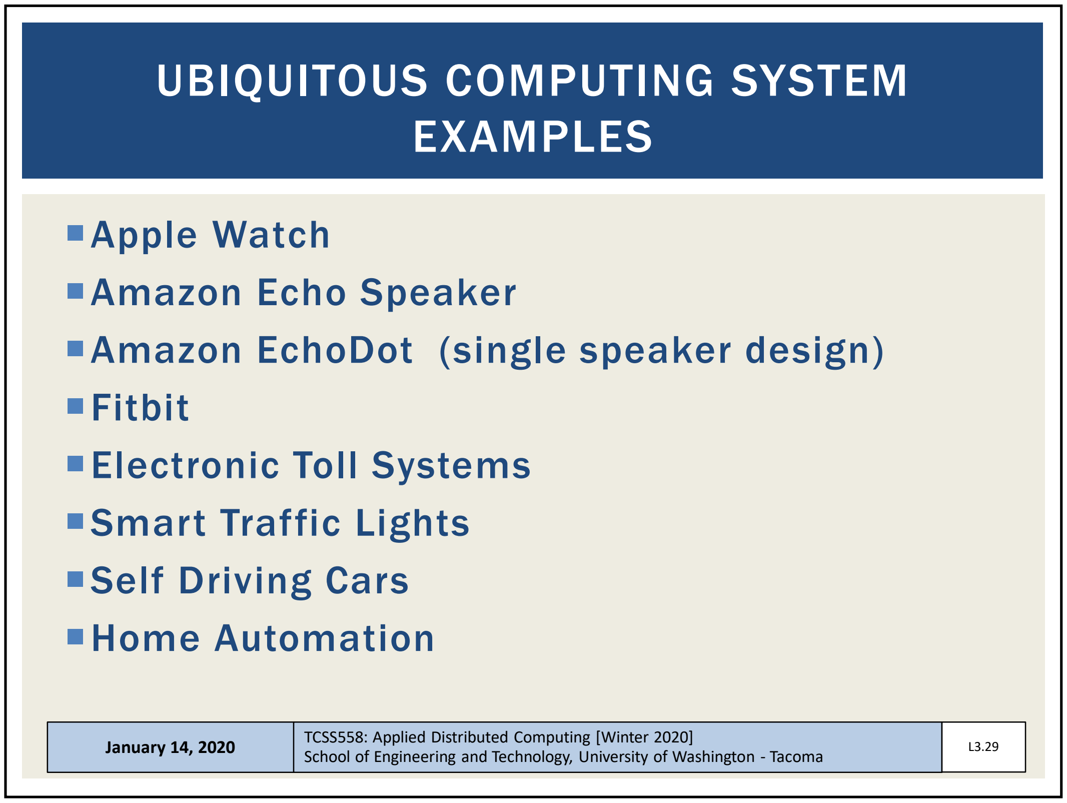
27



PERVASIVE SYSTEM TYPE: UBIQUITOUS COMPUTING SYSTEMS

- Pervasive and continuously present
- Goal: embed processors everywhere (day-to-day objects) enabling them to communicate information
- Requirements for a ubiquitous computing system:
  - **Distribution** – devices are networked, distributed, and accessible transparently
  - **Interaction** – unobtrusive (low-key) between users and devices
  - **Context awareness** – optimizes interaction
  - **Autonomy** – devices operate autonomously, self-managed
  - **Intelligence** – system can handle wide range of dynamic actions and interactions

28

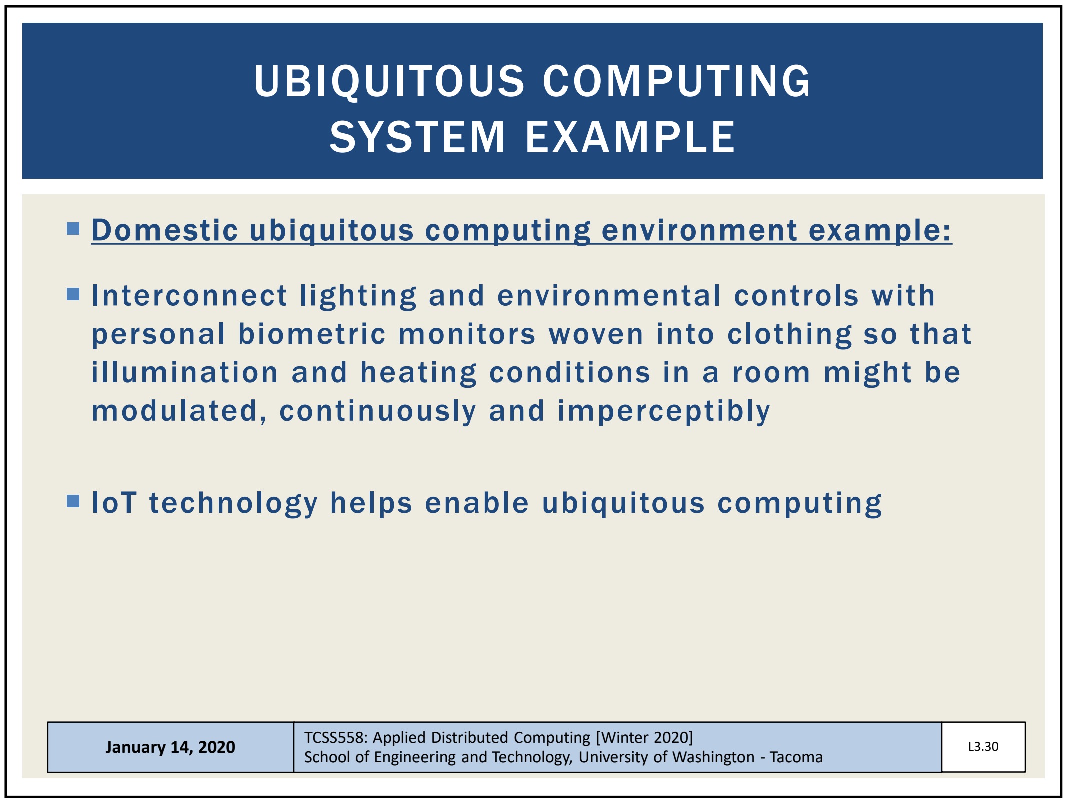


UBIQUITOUS COMPUTING SYSTEM EXAMPLES

- Apple Watch
- Amazon Echo Speaker
- Amazon EchoDot (single speaker design)
- Fitbit
- Electronic Toll Systems
- Smart Traffic Lights
- Self Driving Cars
- Home Automation

January 14, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L3.29

29

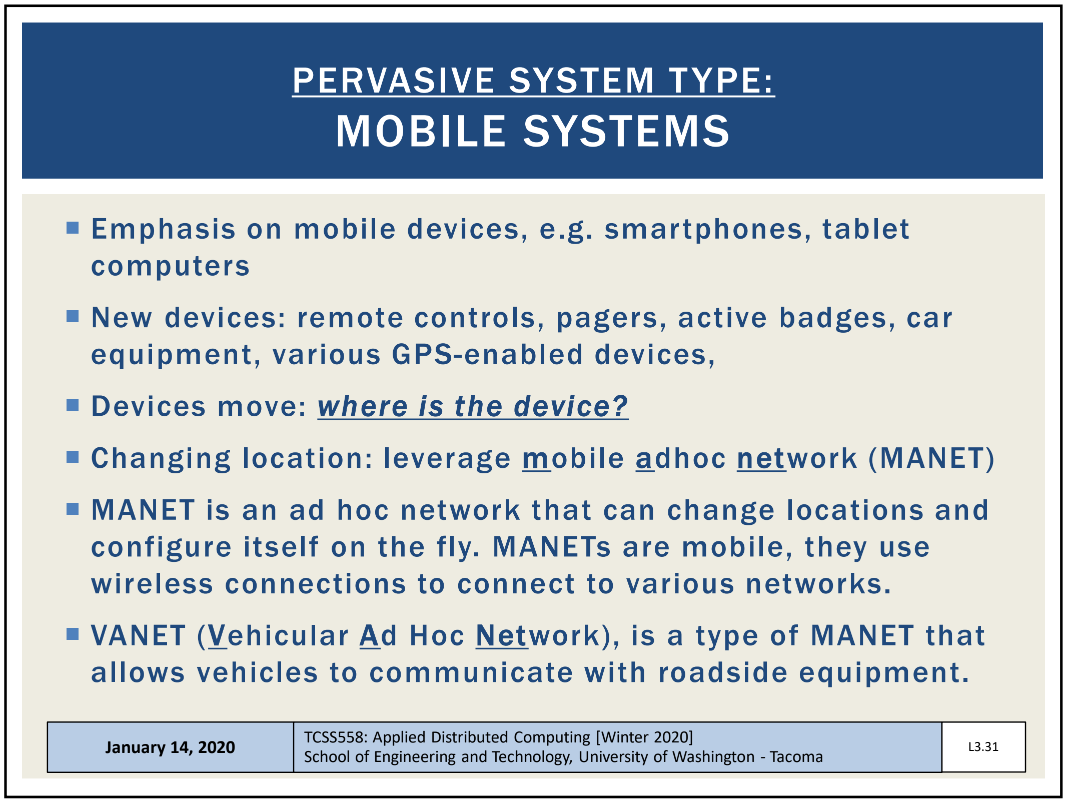


UBIQUITOUS COMPUTING SYSTEM EXAMPLE

- Domestic ubiquitous computing environment example:
- Interconnect lighting and environmental controls with personal biometric monitors woven into clothing so that illumination and heating conditions in a room might be modulated, continuously and imperceptibly
- IoT technology helps enable ubiquitous computing

January 14, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L3.30

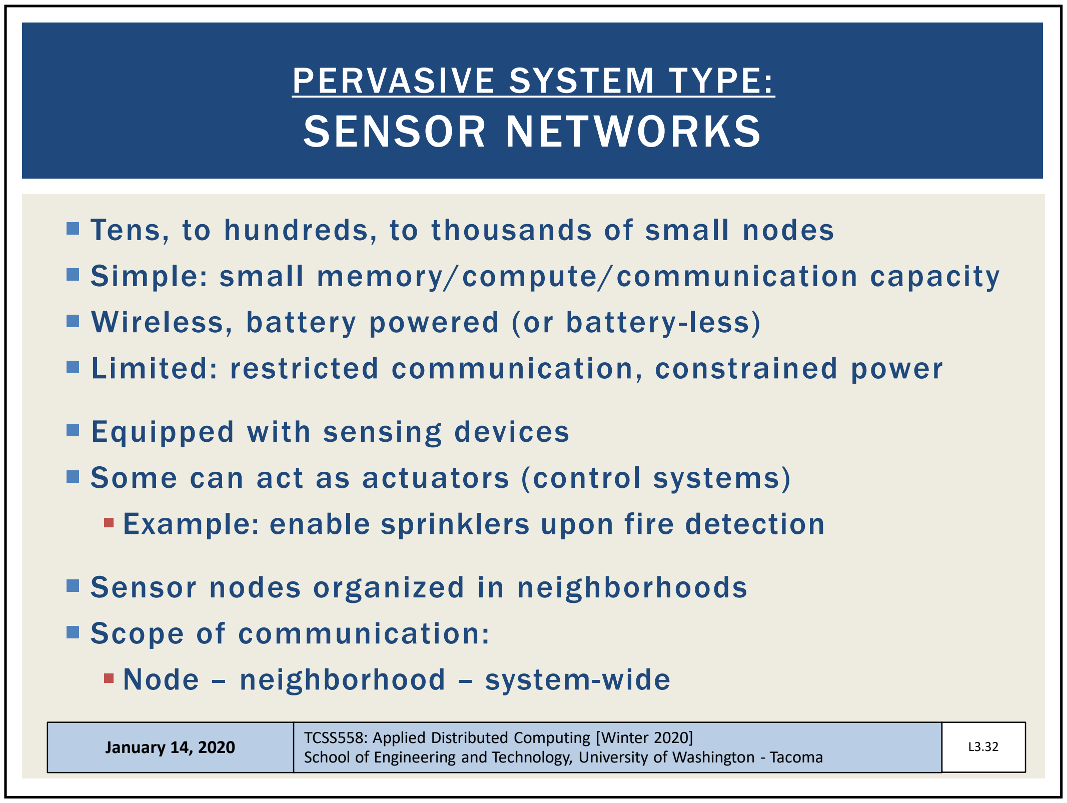
30



PERVASIVE SYSTEM TYPE: MOBILE SYSTEMS

- Emphasis on mobile devices, e.g. smartphones, tablet computers
- New devices: remote controls, pagers, active badges, car equipment, various GPS-enabled devices,
- Devices move: ***where is the device?***
- Changing location: leverage mobile adhoc network (MANET)
- MANET is an ad hoc network that can change locations and configure itself on the fly. MANETs are mobile, they use wireless connections to connect to various networks.
- VANET (Vehicular Ad Hoc Network), is a type of MANET that allows vehicles to communicate with roadside equipment.

31



PERVASIVE SYSTEM TYPE: SENSOR NETWORKS

- Tens, to hundreds, to thousands of small nodes
- Simple: small memory/compute/communication capacity
- Wireless, battery powered (or battery-less)
- Limited: restricted communication, constrained power
- Equipped with sensing devices
- Some can act as actuators (control systems)
  - Example: enable sprinklers upon fire detection
- Sensor nodes organized in neighborhoods
- Scope of communication:
  - Node – neighborhood – system-wide

32

PERVASIVE SYSTEM TYPE: SENSOR NETWORKS - 2

- Collaborate to process sensor data in app-specific manner
- Provide mix of data collection and processing

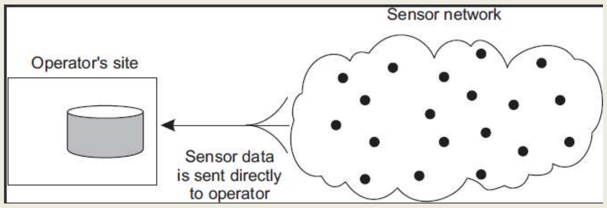
- Nodes may implement a distributed database
- Database organization: centralized to decentralized
- In network processing: forward query to all sensor nodes along a tree to aggregate results and propagate to root

- Is aggregation simply data collection?
- Are all nodes homogeneous?
- Are all network links homogeneous?
- How do we setup a tree when nodes have heterogeneous power and network connection quality?

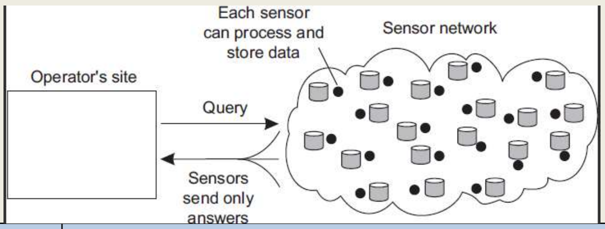
33

CENTRALIZED VS. DECENTRALIZED DATA STORAGE

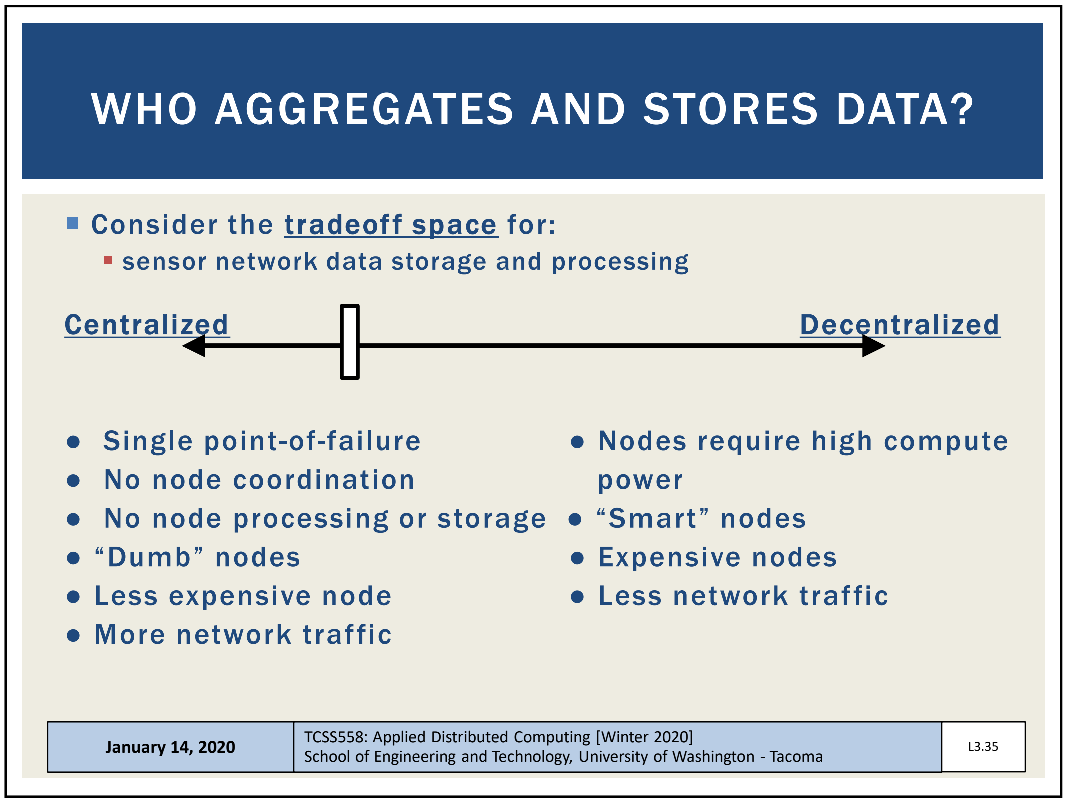
- Centralized:



- Decentralized:


Each sensor can process and store data
Sensor network
Operator's site
Query
Sensors send only answers

34



WHO AGGREGATES AND STORES DATA?

- Consider the **tradeoff space** for:
  - sensor network data storage and processing

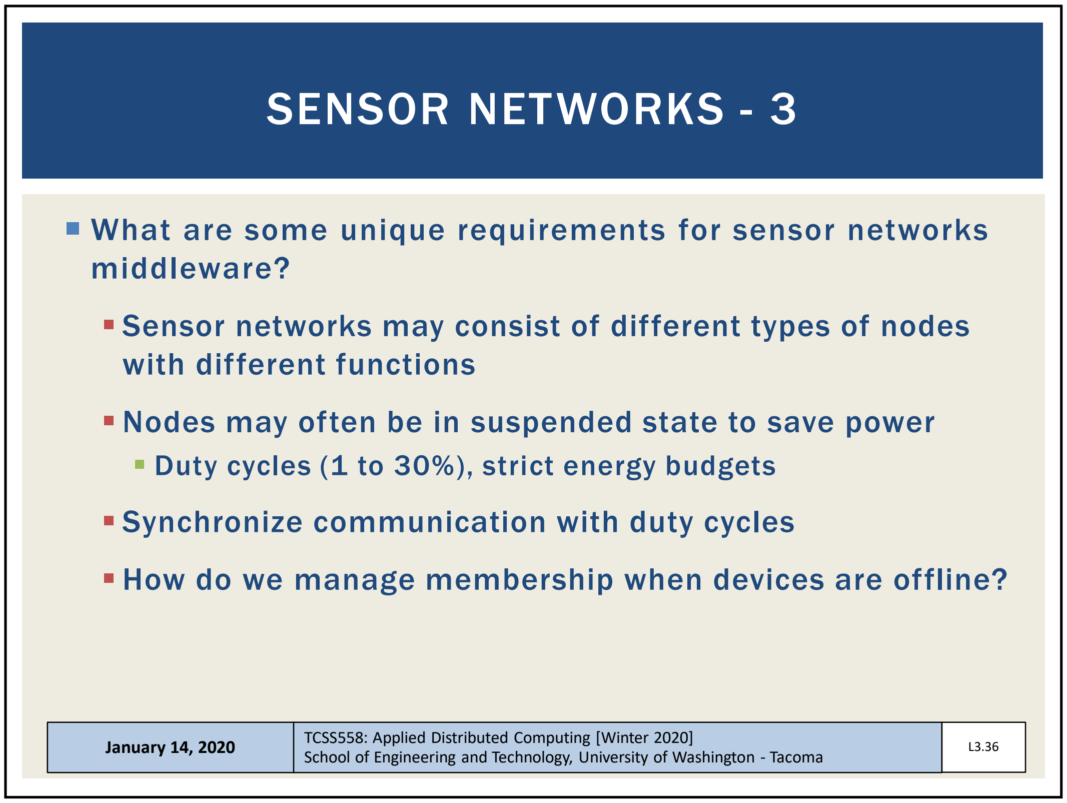
Centralized ⟵ ▯ ⟶ **Decentralized**

- Single point-of-failure
- No node coordination
- No node processing or storage
- "Dumb" nodes
- Less expensive node
- More network traffic

- Nodes require high compute power
- "Smart" nodes
- Expensive nodes
- Less network traffic

January 14, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L3.35

35



SENSOR NETWORKS - 3

- What are some unique requirements for sensor networks middleware?
  - Sensor networks may consist of different types of nodes with different functions
  - Nodes may often be in suspended state to save power
    - Duty cycles (1 to 30%), strict energy budgets
  - Synchronize communication with duty cycles
  - How do we manage membership when devices are offline?

January 14, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L3.36

36

TYPES OF DISTRIBUTED SYSTEMS

- HPC, Cluster, Grid, Cloud
- Distributed information systems
  - Transactions
  - Application Integration: Shared files, DBs, RPC, RMI, Message-oriented middleware
- Pervasive Systems
  - Ubiquitous computing systems
  - Mobile systems
  - Sensor networks

37

RESEARCH DIRECTIONS

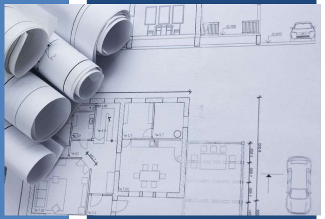
38

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*

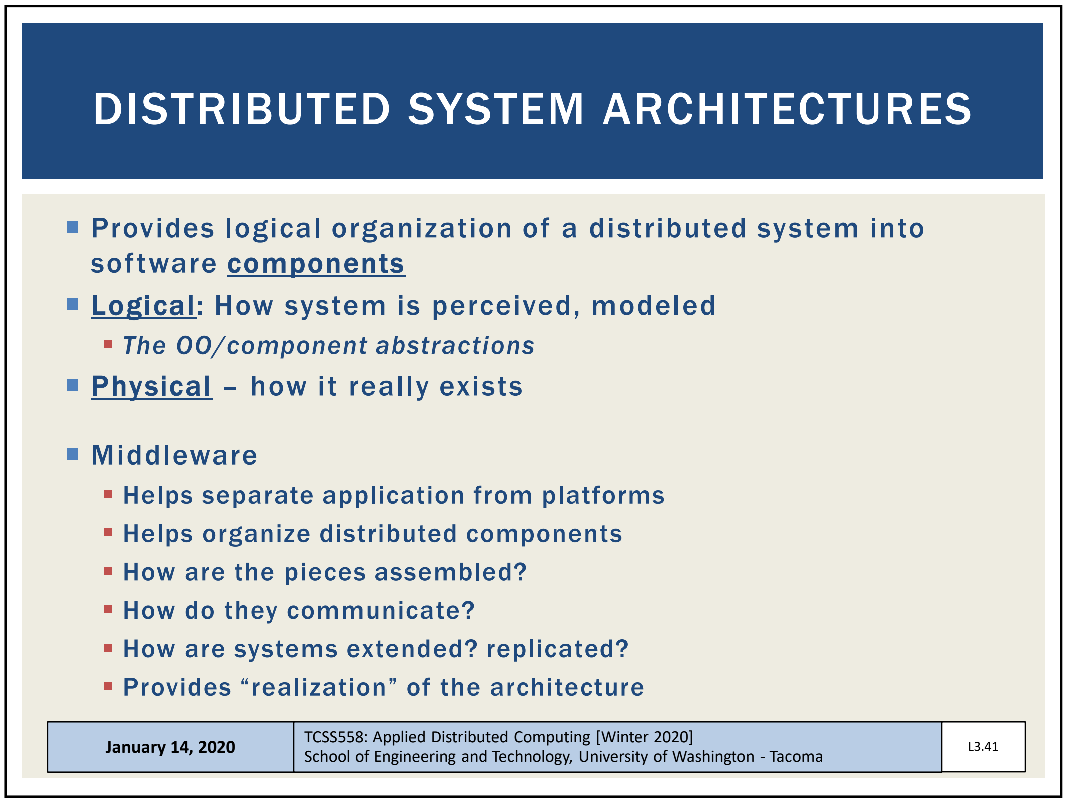
The CDS group collaborates on research projects spanning Serverless computing (FaaS), Containerization, Infrastructure-as-a-Service (IaaS) cloud, virtualization, infrastructure management, and performance and cost modeling of application deployments. Our research aims to demystify the myriad of options 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.

39



CH. 2.1: DISTRIBUTED SYSTEMS ARCHITECTURES

40

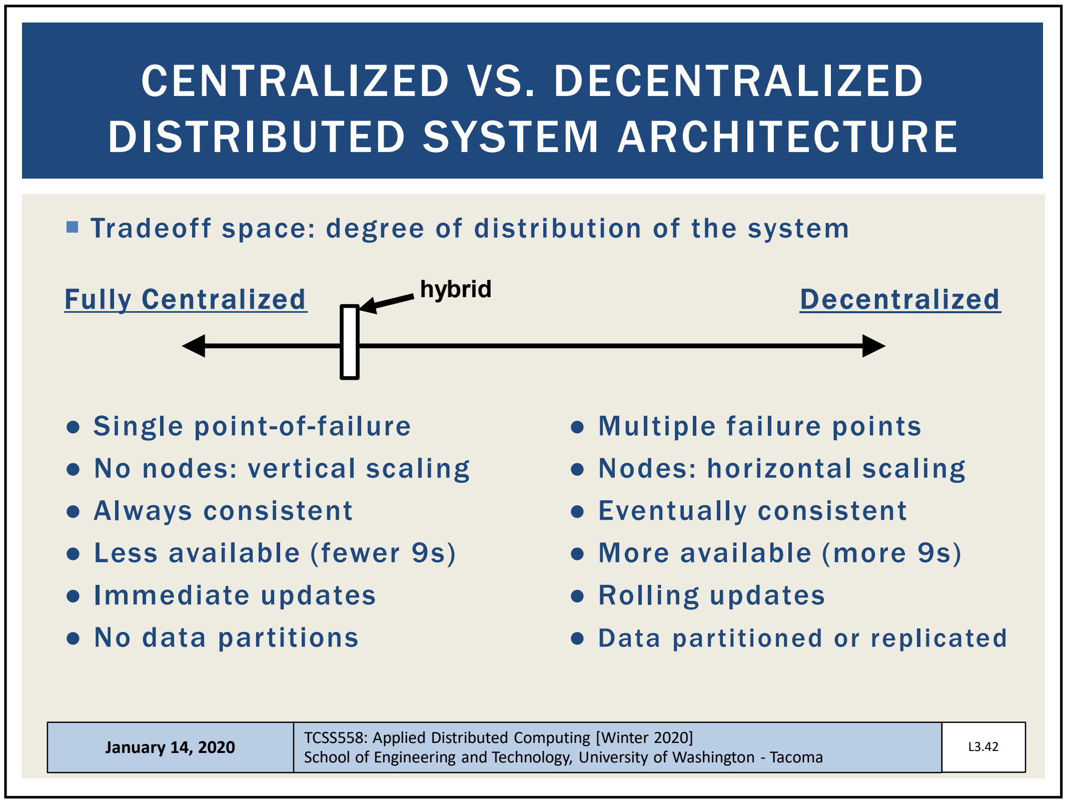


DISTRIBUTED SYSTEM ARCHITECTURES

- Provides logical organization of a distributed system into software **components**
- **Logical**: How system is perceived, modeled
  - *The OO/component abstractions*
- **Physical** – how it really exists
- Middleware
  - Helps separate application from platforms
  - Helps organize distributed components
  - How are the pieces assembled?
  - How do they communicate?
  - How are systems extended? replicated?
  - Provides "realization" of the architecture

January 14, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L3.41

41



CENTRALIZED VS. DECENTRALIZED DISTRIBUTED SYSTEM ARCHITECTURE

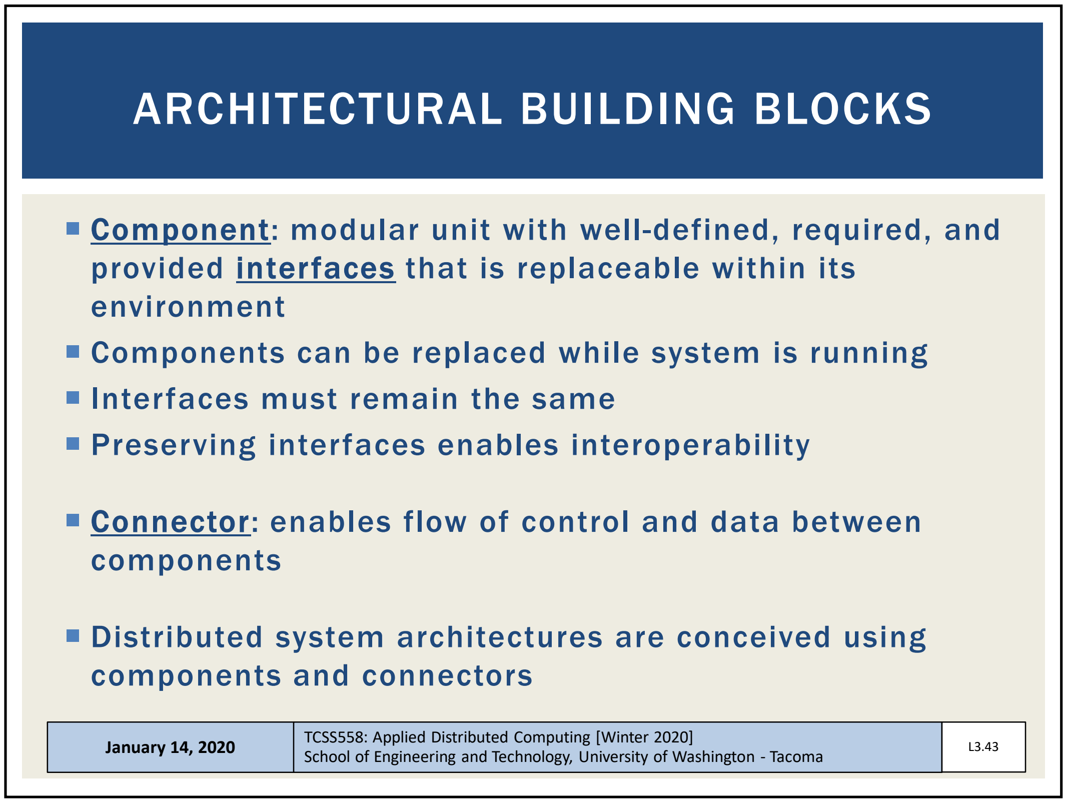
- Tradeoff space: degree of distribution of the system

Fully Centralized | **Decentralized**

Fully Centralized	Decentralized
● Single point-of-failure	● Multiple failure points
● No nodes: vertical scaling	● Nodes: horizontal scaling
● Always consistent	● Eventually consistent
● Less available (fewer 9s)	● More available (more 9s)
● Immediate updates	● Rolling updates
● No data partitions	● Data partitioned or replicated

January 14, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L3.42

42

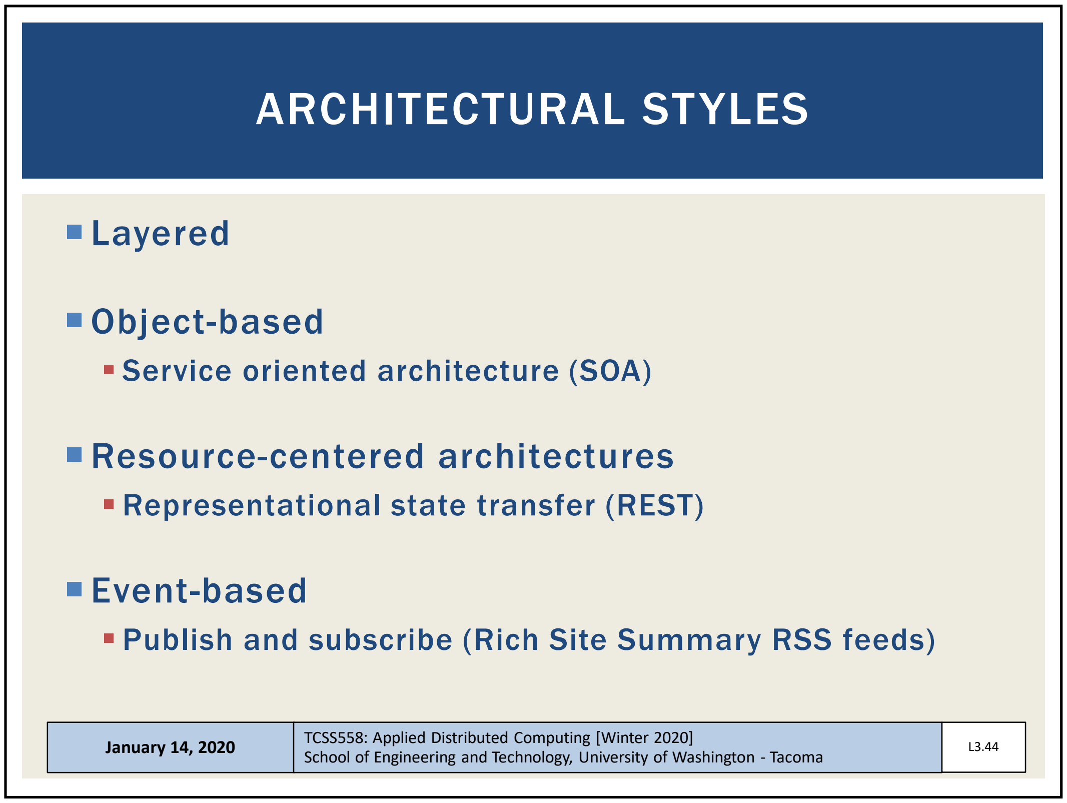


ARCHITECTURAL BUILDING BLOCKS

- **Component**: modular unit with well-defined, required, and provided **interfaces** that is replaceable within its environment
- Components can be replaced while system is running
- Interfaces must remain the same
- Preserving interfaces enables interoperability
- **Connector**: enables flow of control and data between components
- Distributed system architectures are conceived using components and connectors

January 14, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L3.43

43

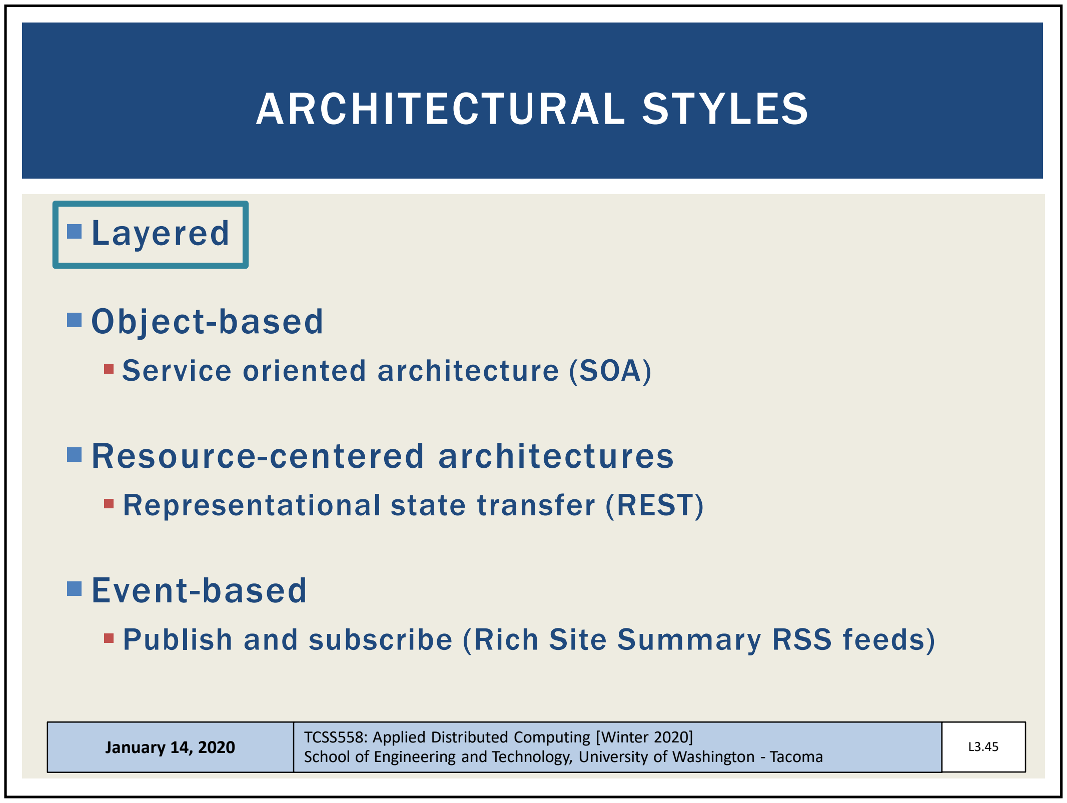


ARCHITECTURAL STYLES

- Layered
- Object-based
  - Service oriented architecture (SOA)
- Resource-centered architectures
  - Representational state transfer (REST)
- Event-based
  - Publish and subscribe (Rich Site Summary RSS feeds)

January 14, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L3.44

44

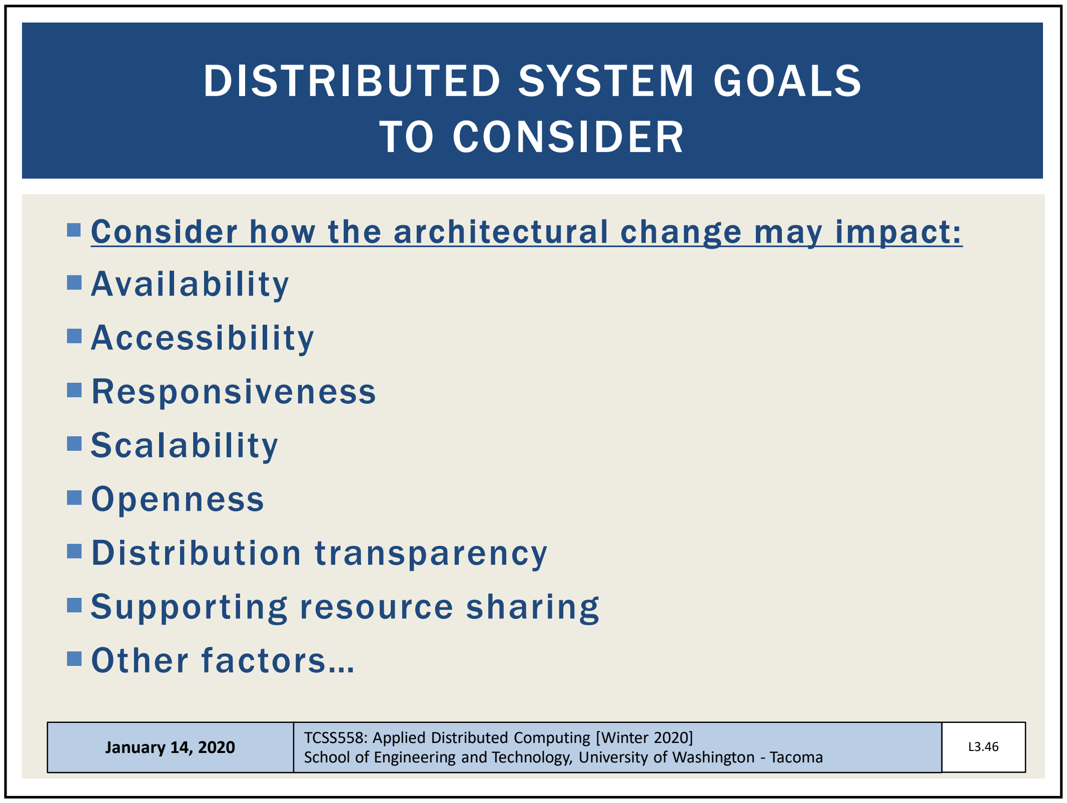


ARCHITECTURAL STYLES

- Layered
- Object-based
  - Service oriented architecture (SOA)
- Resource-centered architectures
  - Representational state transfer (REST)
- Event-based
  - Publish and subscribe (Rich Site Summary RSS feeds)

January 14, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L3.45

45

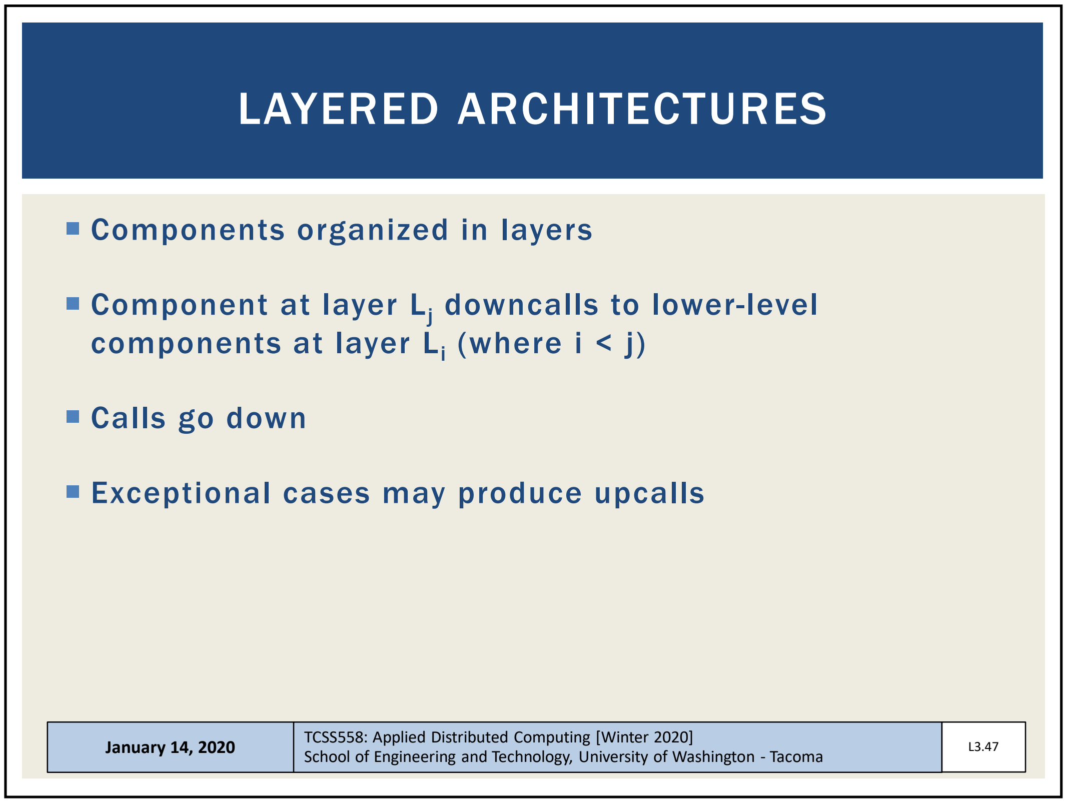


DISTRIBUTED SYSTEM GOALS TO CONSIDER

- Consider how the architectural change may impact:
- Availability
- Accessibility
- Responsiveness
- Scalability
- Openness
- Distribution transparency
- Supporting resource sharing
- Other factors…

January 14, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L3.46

46

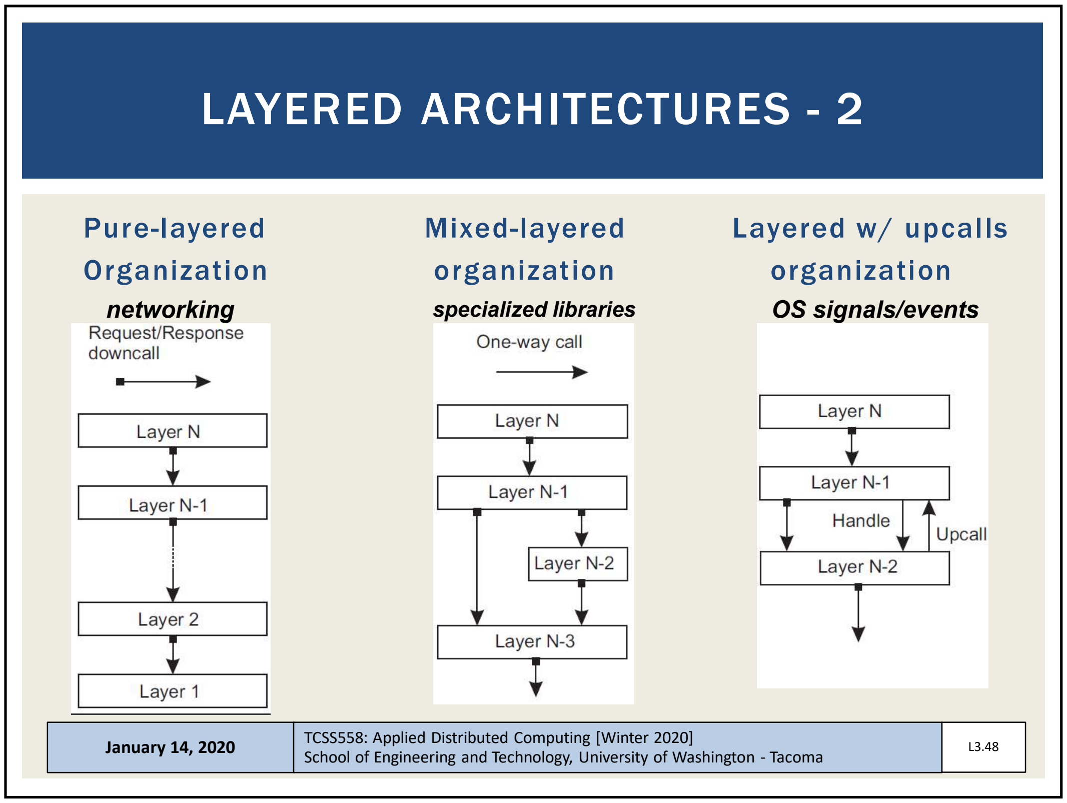


LAYERED ARCHITECTURES

- Components organized in layers
- Component at layer L_j downcalls to lower-level components at layer L_i (where $i < j$)
- Calls go down
- Exceptional cases may produce upcalls

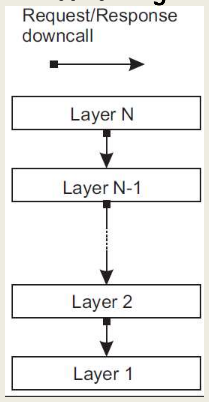
January 14, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L3.47

47

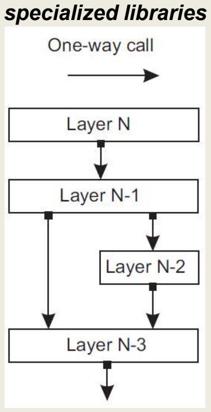


LAYERED ARCHITECTURES - 2

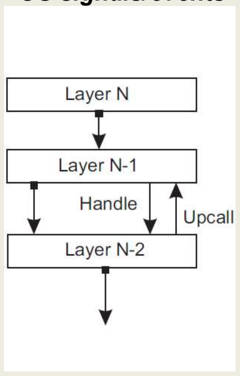
Pure-layered Organization
networking



Mixed-layered organization
specialized libraries



Layered w/ upcalls organization
OS signals/events



January 14, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L3.48

48

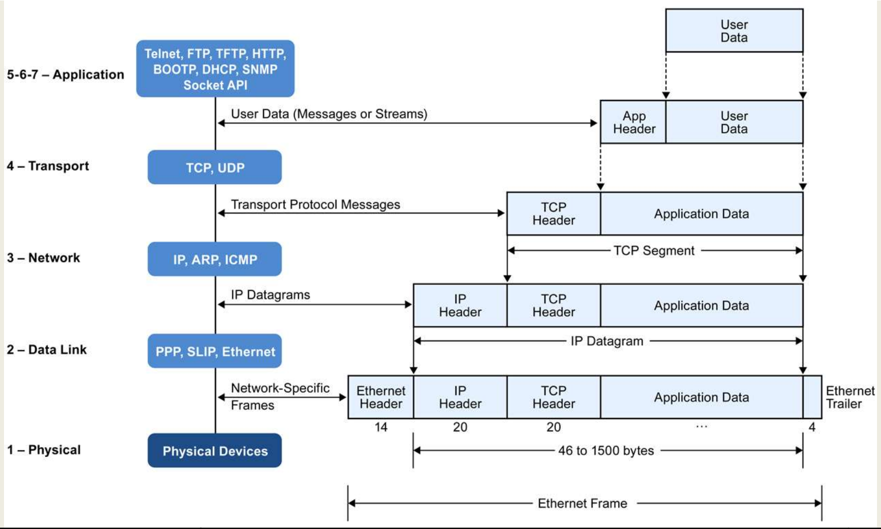
COMMUNICATION-PROTOCOL STACKS

- Example: pure-layered organization
- Each layer offers an interface specifying functions of the layer
- Communication protocol: rules used for nodes to communicate
- Layer provides a <u>service</u>
- <u>Interface</u> makes service available
- <u>Protocol</u> implements communication for a layer

- New services can be built atop of existing layers to reuse low level implementation
- Abstractions make it easier to reuse existing layers which already implement communication basics

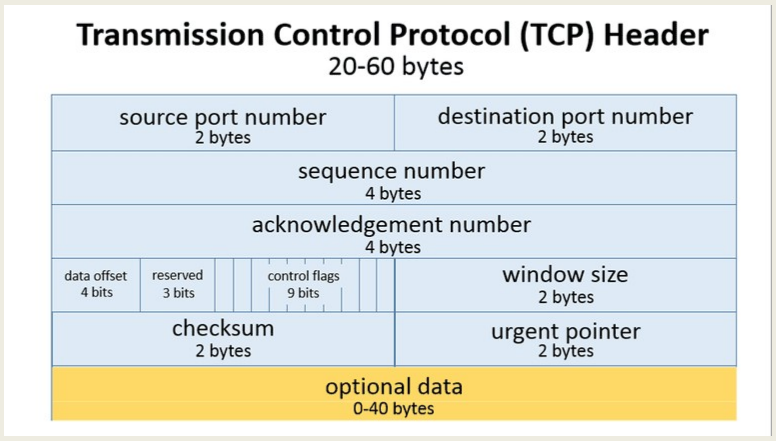
49

HOW A NETWORK PACKET IS BUILT



50

TCP HEADER

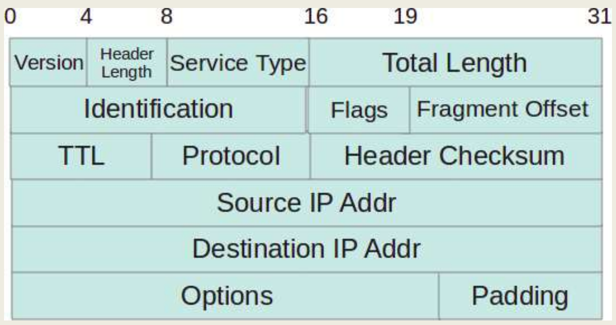


January 14, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L3.51

51

IP HEADER

- Source / Destination IP Addr
- IPv4: 32bits / 4 bytes
- IPv6: 128bits / 16 bytes



January 14, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L3.52

52

TRANSMISSION CONTROL PROTOCOL (TCP)

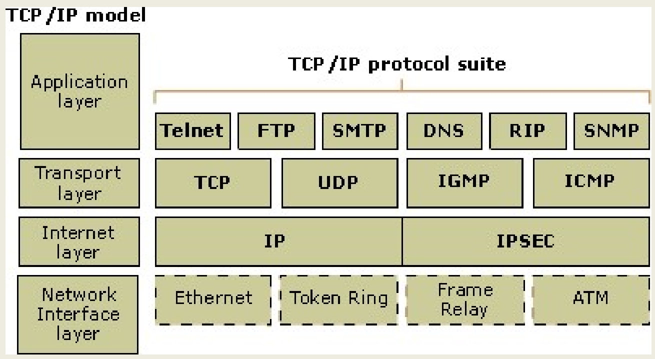
- TCP provides easy to use API
- API supports: setup, tear down of connection(s)
- API supports: sending and receiving of messages
- TCP preserves ordering of transferred data
- TCP detects and corrects lost data

- But TCP is “protocol” agnostic
  - E.g. language agnostic

- What are we going to say?

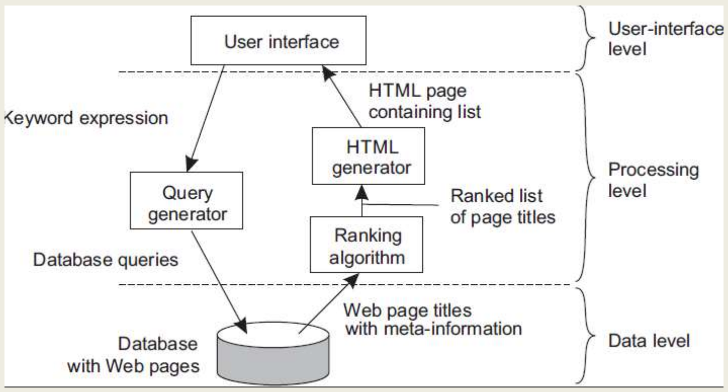
COMMON APPLICATION LAYER PROTOCOLS

- Telnet, FTP, TFTP, HTTP, DHCP, DNS, NTP, POP, RTP, SMTP, Telnet, RPC, LDAP



APPLICATION LAYERING

- Distributed application example: Internet search engine

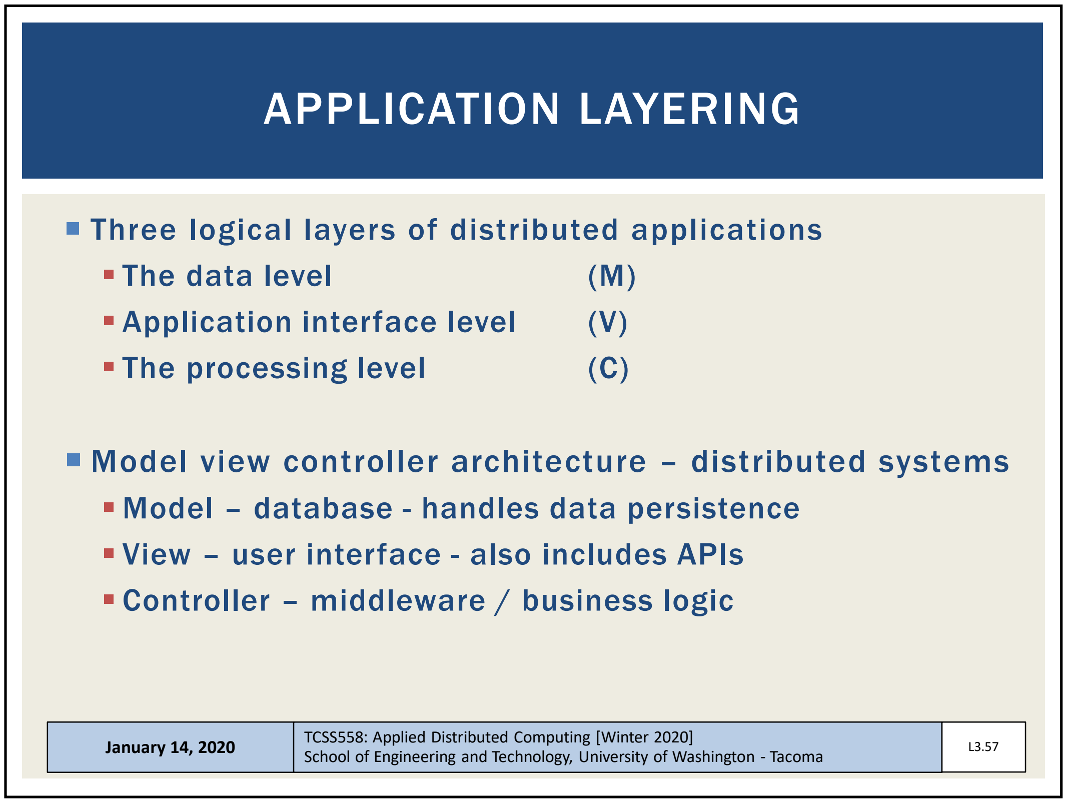


55

APPLICATION LAYERING

- Three logical layers of distributed applications
  - The data level
  - Application interface level
  - The processing level

56

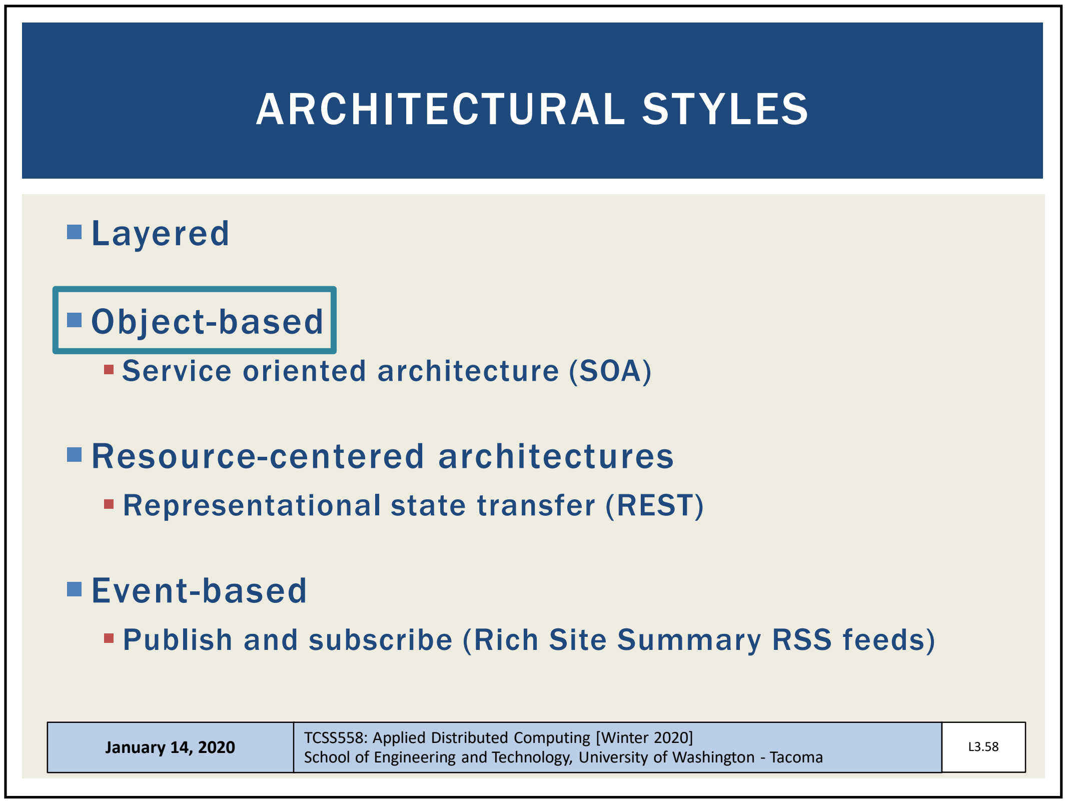


APPLICATION LAYERING

- Three logical layers of distributed applications
  - The data level (M)
  - Application interface level (V)
  - The processing level (C)
- Model view controller architecture – distributed systems
  - Model – database - handles data persistence
  - View – user interface - also includes APIs
  - Controller – middleware / business logic

January 14, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L3.57

57



ARCHITECTURAL STYLES

- Layered
- Object-based
  - Service oriented architecture (SOA)
- Resource-centered architectures
  - Representational state transfer (REST)
- Event-based
  - Publish and subscribe (Rich Site Summary RSS feeds)

January 14, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L3.58

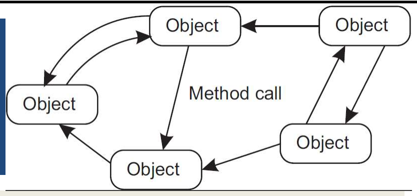
58

OBJECT-BASED ARCHITECTURES

- Enables loose and flexible component organization
- Objects == components
- Enable distributed node interaction via function calls over the network
- Began with C - Remote Procedure Calls (RPC)
  - Straightforward: package up function inputs, send over network, transfer results back
  - Language independent
  - In contrast to web services, RPC calls originally were more intimate in nature
  - Procedures more "coupled", not as independent
  - The goal was not to decouple and widgetize everything

59

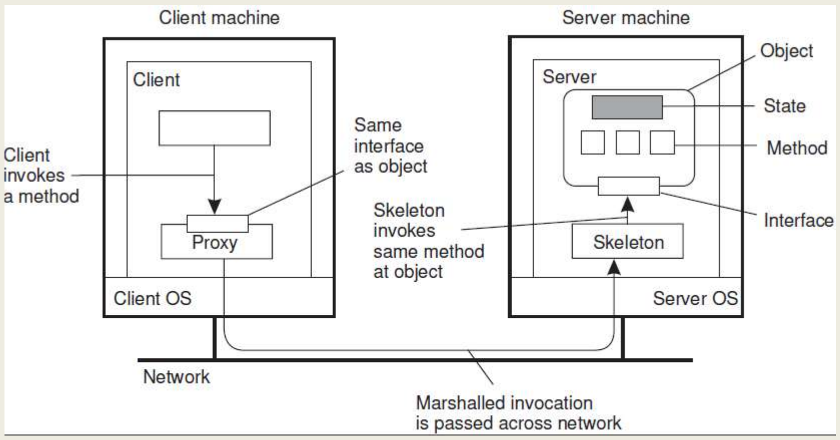
OBJECT-BASED ARCHITECTURES - 2



- Distributed objects Java- Remote Method Invocation (RMI)
  - Adds object orientation concepts to remote function calls
  - Clients bind to proxy objects
  - Proxy provide an object interface which transfers method invocation over the network to the remote host
- How do we replicate objects?
  - Object marshalling – serialize data, stream it over network
  - Unmarshalling- create an object from the stream
  - Unmarshall local object copies on the remote host
  - JSON, XML are some possible data formats

60

DISTRIBUTED OBJECTS

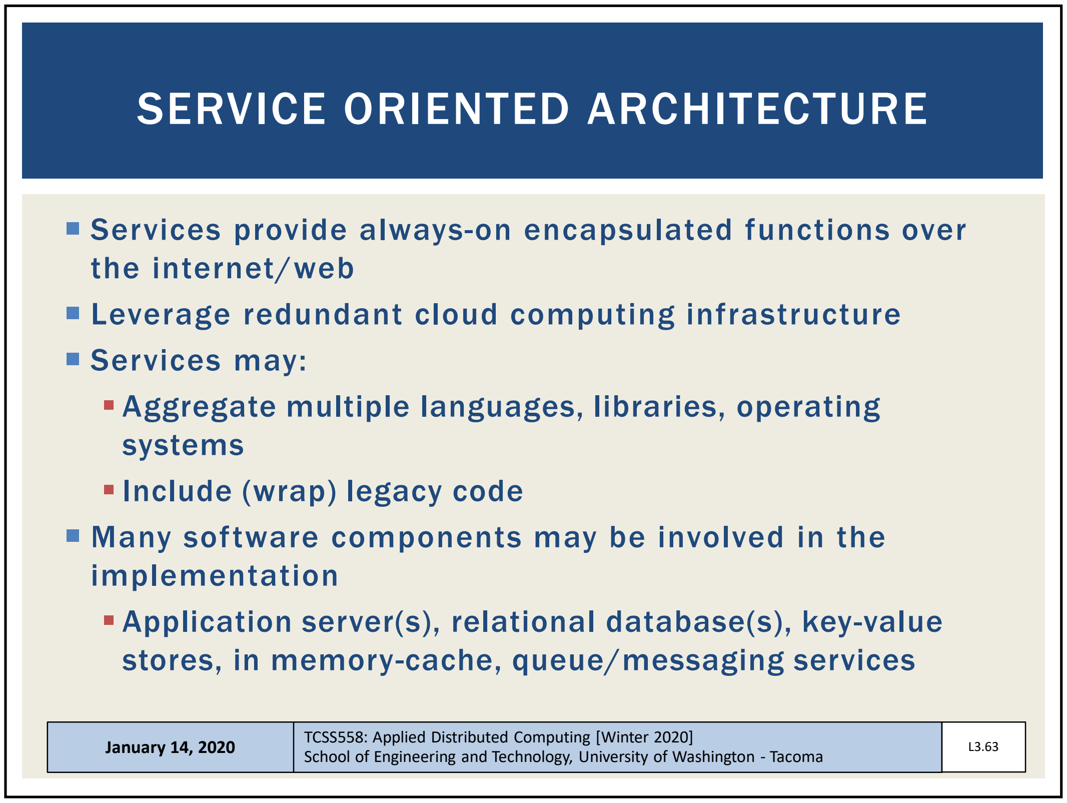


61

DISTRIBUTED OBJECTS - 2

- A counterintuitive features is that state is not distributed
- Each "remote object" maintains its own state
- Remote objects may not be replicated
- Objects may be "mobile" and move around from node to node
  - Common for data objects
- For distributed (remote) objects consider
  - Pass by value
  - Pass by reference

62

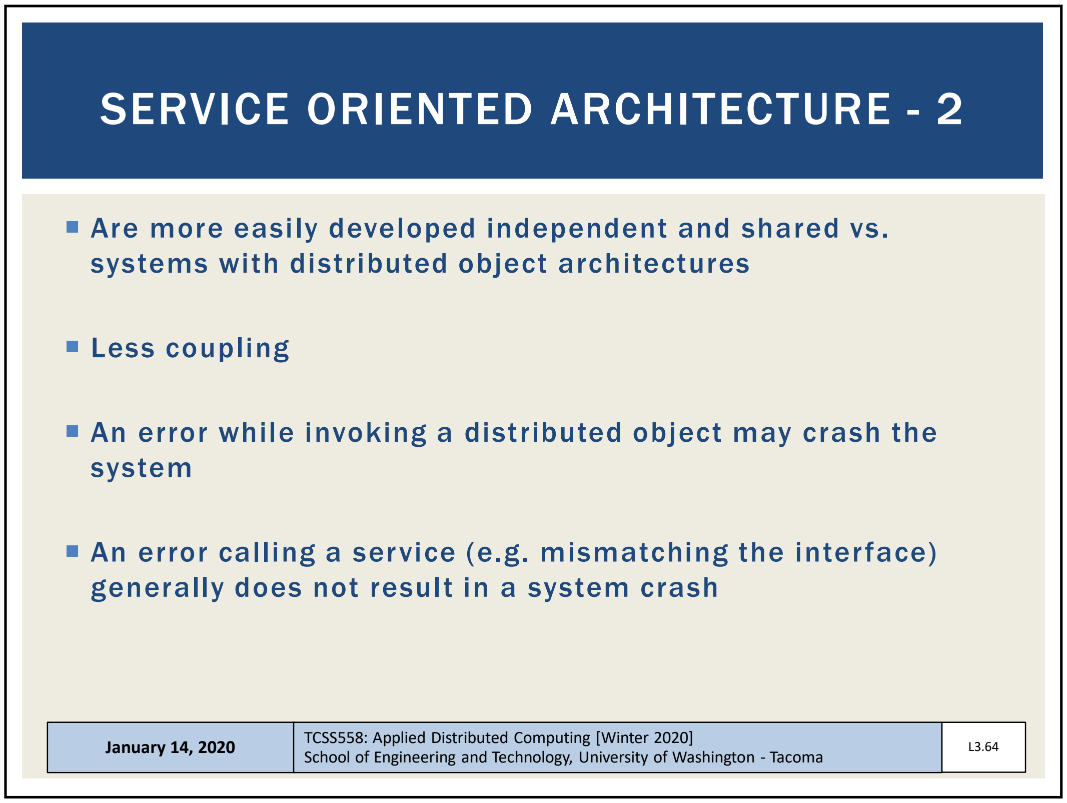


SERVICE ORIENTED ARCHITECTURE

- Services provide always-on encapsulated functions over the internet/web
- Leverage redundant cloud computing infrastructure
- Services may:
  - Aggregate multiple languages, libraries, operating systems
  - Include (wrap) legacy code
- Many software components may be involved in the implementation
  - Application server(s), relational database(s), key-value stores, in memory-cache, queue/messaging services

January 14, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L3.63

63

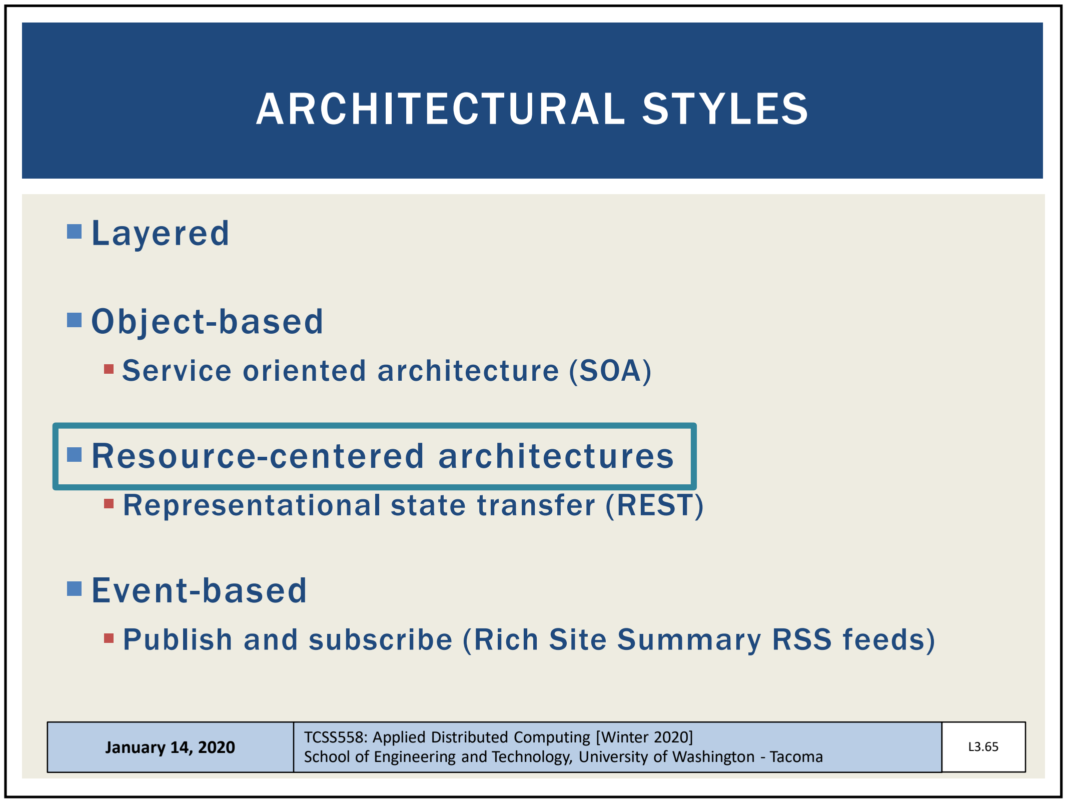


SERVICE ORIENTED ARCHITECTURE - 2

- Are more easily developed independent and shared vs. systems with distributed object architectures
- Less coupling
- An error while invoking a distributed object may crash the system
- An error calling a service (e.g. mismatching the interface) generally does not result in a system crash

January 14, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L3.64

64

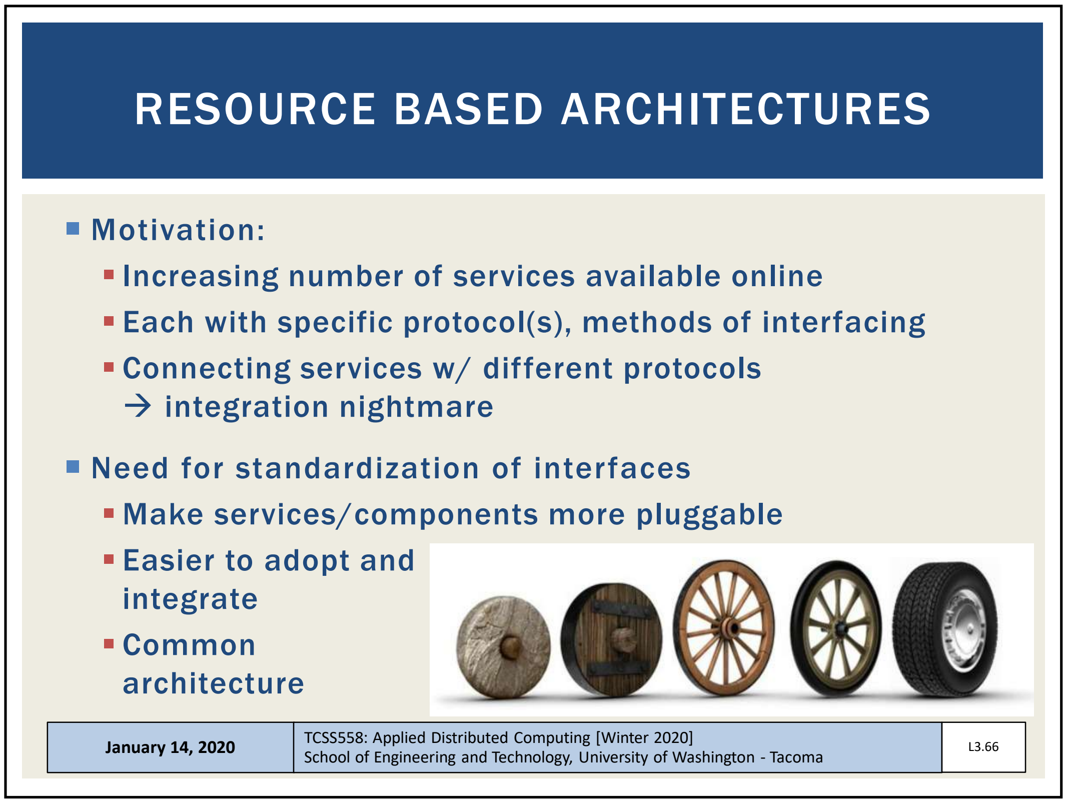


ARCHITECTURAL STYLES

- Layered
- Object-based
  - Service oriented architecture (SOA)
- Resource-centered architectures
  - Representational state transfer (REST)
- Event-based
  - Publish and subscribe (Rich Site Summary RSS feeds)

January 14, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L3.65

65

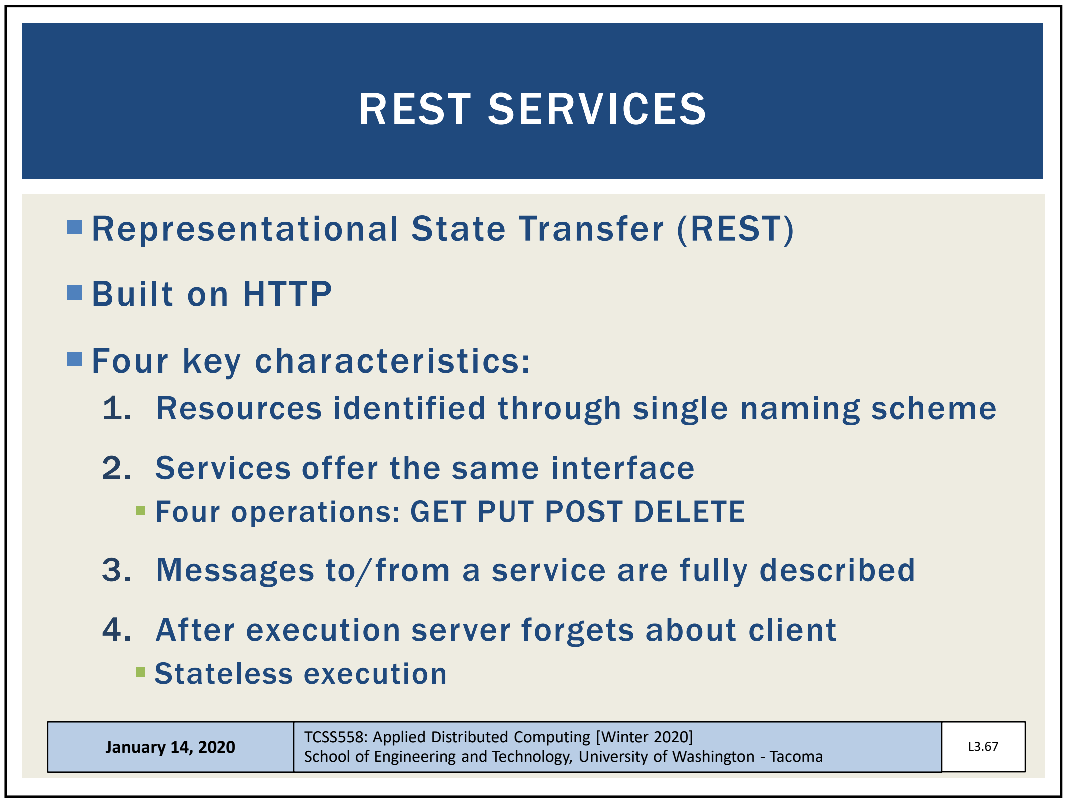


RESOURCE BASED ARCHITECTURES

- Motivation:
  - Increasing number of services available online
  - Each with specific protocol(s), methods of interfacing
  - Connecting services w/ different protocols → integration nightmare
- Need for standardization of interfaces
  - Make services/components more pluggable
  - Easier to adopt and integrate
  - Common architecture

January 14, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L3.66

66

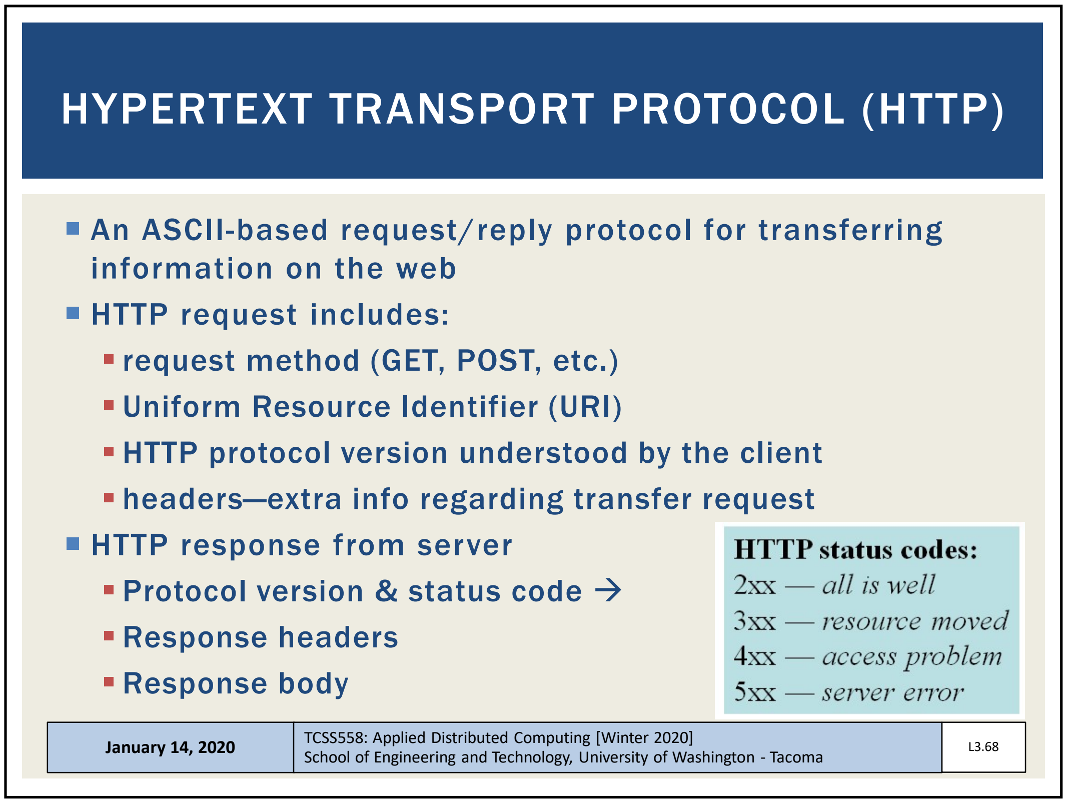


REST SERVICES

- Representational State Transfer (REST)
- Built on HTTP
- Four key characteristics:
  1. Resources identified through single naming scheme
  2. Services offer the same interface
     - Four operations: GET PUT POST DELETE
  3. Messages to/from a service are fully described
  4. After execution server forgets about client
     - Stateless execution

January 14, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L3.67

67



HYPERTEXT TRANSPORT PROTOCOL (HTTP)

- An ASCII-based request/reply protocol for transferring information on the web
- HTTP request includes:
  - request method (GET, POST, etc.)
  - Uniform Resource Identifier (URI)
  - HTTP protocol version understood by the client
  - headers—extra info regarding transfer request
- HTTP response from server
  - Protocol version & status code →
  - Response headers
  - Response body

HTTP status codes:
2xx — *all is well*
3xx — *resource moved*
4xx — *access problem*
5xx — *server error*

January 14, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L3.68

68

REST-FUL OPERATIONS

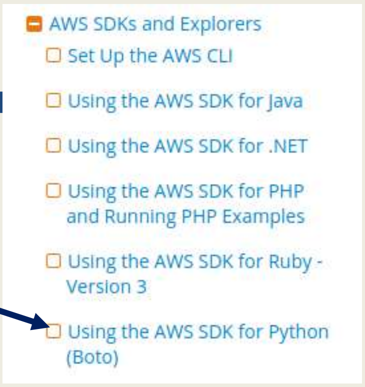
Operation	Description	
PUT	Create a new resource	(C)reate
GET	Retrieve state of a resource in some format	(R)ead
POST	Modify a resource by transferring a new state	(U)pdate
DELETE	Delete a resource	(D)elete

- Resources often implemented as objects in OO languages
- REST is weak for tracking state
- Generic REST interfaces enable ubiquitous *"so many"* clients

69

EXAMPLE: AMAZON S3

- Amazon S3 offers a REST-based interface
- Requires signing HTTP authorization header or passing authentication parameters in the URL query string
- REST: GET/PUT/POST/DELETE
- SOAP: 16 operations, moving toward deprecation
- Python boto ~50 operations (SDK for Python)
- SDKs for other languages


AWS SDKs and Explorers
Set Up the AWS CLI
Using the AWS SDK for Java
Using the AWS SDK for .NET
Using the AWS SDK for PHP and Running PHP Examples
Using the AWS SDK for Ruby - Version 3
Using the AWS SDK for Python (Boto)

70

REST - 2

- Defacto web services protocol
- Requests made to a URI – uniform resource identifier
- Supersedes SOAP – Simple Object Access Protocol
- Access and manipulate web resources with a predefined set of stateless operations (known as web services)
- Responses most often in JSON, also HTML, ASCII text, XML, no real limits as long as text-based
- curl – generic command-line REST client: https://curl.haxx.se/

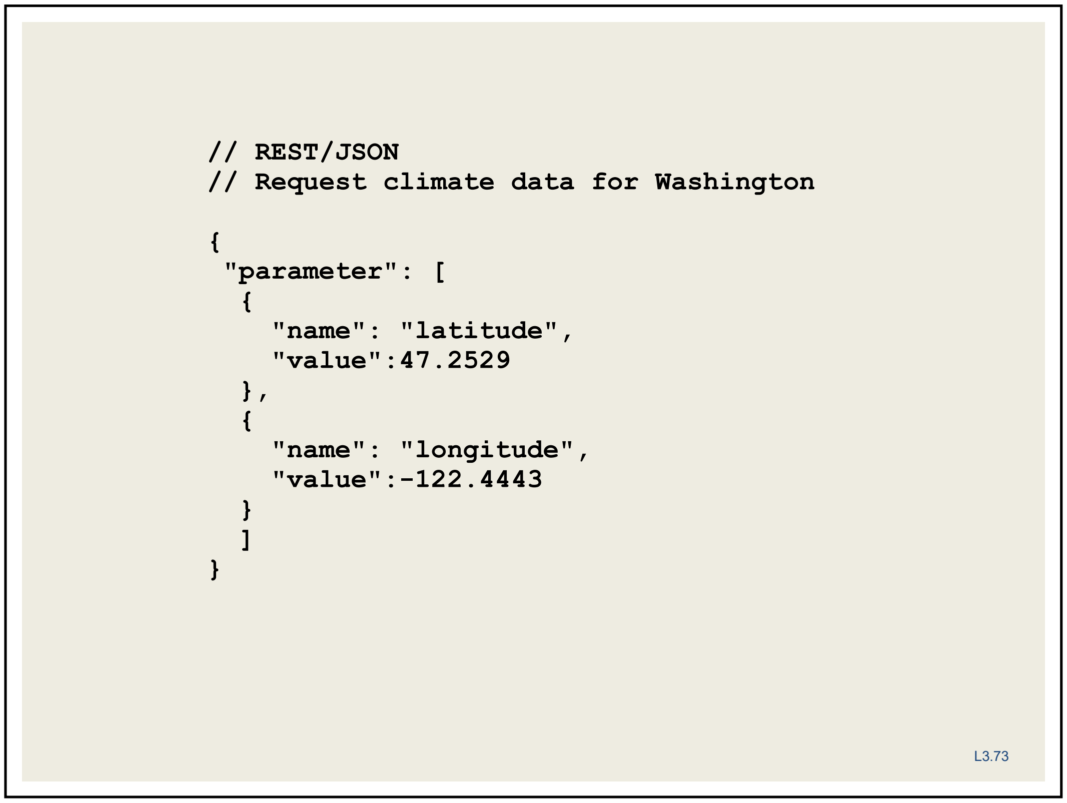
January 14, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L3.71

71

```
// WSDL Service Definition
<?xml version="1.0" encoding="UTF-8"?>
<definitions  name ="DayOfWeek"
  targetNamespace="http://www.roguewave.com/soapworx/examples/DayOfWeek.wsdl"
  xmlns:tns="http://www.roguewave.com/soapworx/examples/DayOfWeek.wsdl"
  xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
  xmlns="http://schemas.xmlsoap.org/wsdl/">
  <message name="DayOfWeekInput">
    <part name="date" type="xsd:date"/>
  </message>
  <message name="DayOfWeekResponse">
    <part name="dayOfWeek" type="xsd:string"/>
  </message>
  <portType name="DayOfWeekPortType">
    <operation name="GetDayOfWeek">
      <input message="tns:DayOfWeekInput"/>
      <output message="tns:DayOfWeekResponse"/>
    </operation>
  </portType>
  <binding name="DayOfWeekBinding" type="tns:DayOfWeekPortType">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="GetDayOfWeek">
      <soap:operation soapAction="getdayofweek"/>
      <input>
        <soap:body use="encoded"
          namespace="http://www.roguewave.com/soapworx/examples"
          encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </input>
      <output>
        <soap:body use="encoded"
        namespace="http://www.roguewave.com/soapworx/examples"
        encodingStyle="http://schemas.xmlsoap.org/soap/encoding/"/>
      </output>
    </operation>
  </binding>
  <service name="DayOfWeekService" >
    <documentation>
      Returns the day-of-week name for a given date
    </documentation>
    <port name="DayOfWeekPort" binding="tns:DayOfWeekBinding">
      <soap:address location="http://localhost:8090/dayofweek/DayOfWeek"/>
    </port>
   </service>
</definitions>
```

L3.72

72

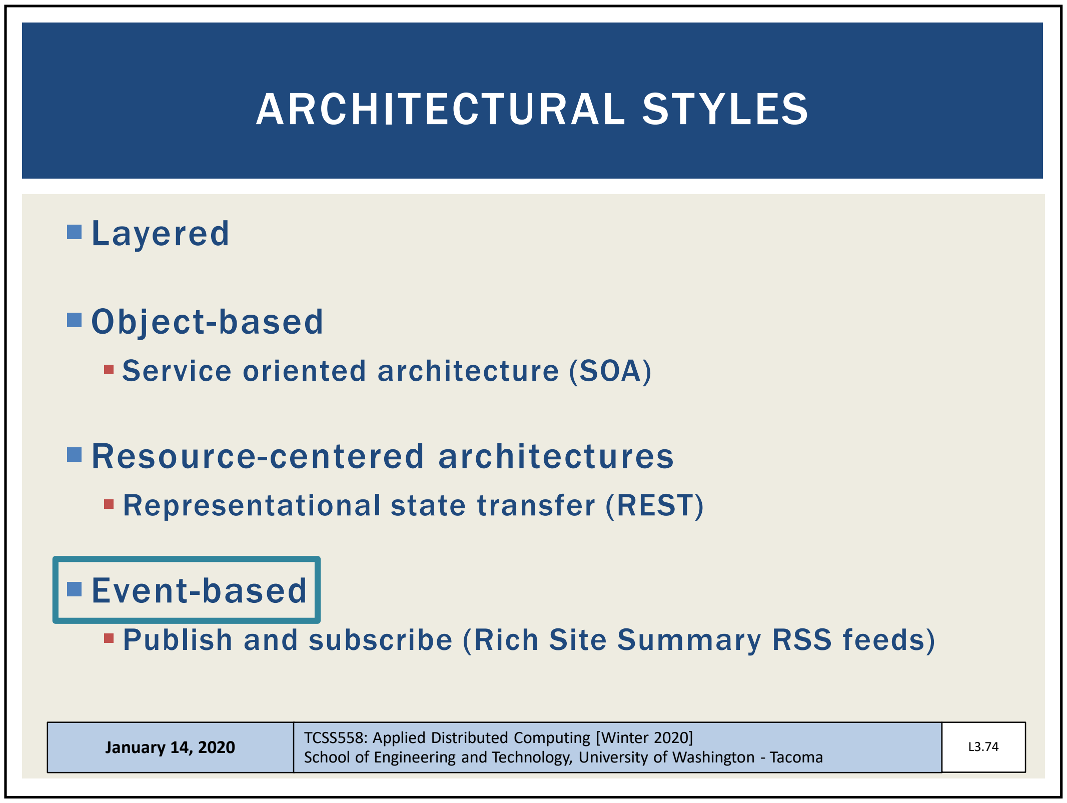


```
// REST/JSON
// Request climate data for Washington

{
 "parameter": [
  {
    "name": "latitude",
    "value":47.2529
  },
  {
    "name": "longitude",
    "value":-122.4443
  }
  ]
}
```

L3.73

73



ARCHITECTURAL STYLES

- Layered
- Object-based
  - Service oriented architecture (SOA)
- Resource-centered architectures
  - Representational state transfer (REST)
- Event-based
  - Publish and subscribe (Rich Site Summary RSS feeds)

January 14, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L3.74

74

PUBLISH-SUBSCRIBE ARCHITECTURES

- Enables separation between processing and coordination
- Types of coordination:

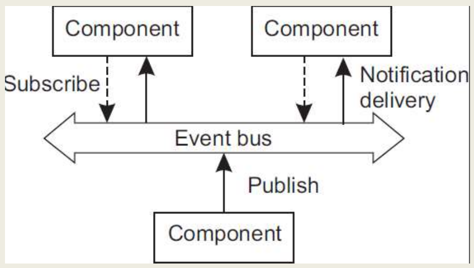
	Temporally coupled (at the same time)	Temporally decoupled (at different times)
Referentially coupled (*dependent on name*)	**Direct** Explicit synchronous service call	**Mailbox** Asynchronous by name (address)
Referentially decoupled (*name not required*)	**Event-based** Event notices published to shared bus, w/o addressing	**Shared data space** Processes write tuples to a shared data space

Not publish and subscribe

75

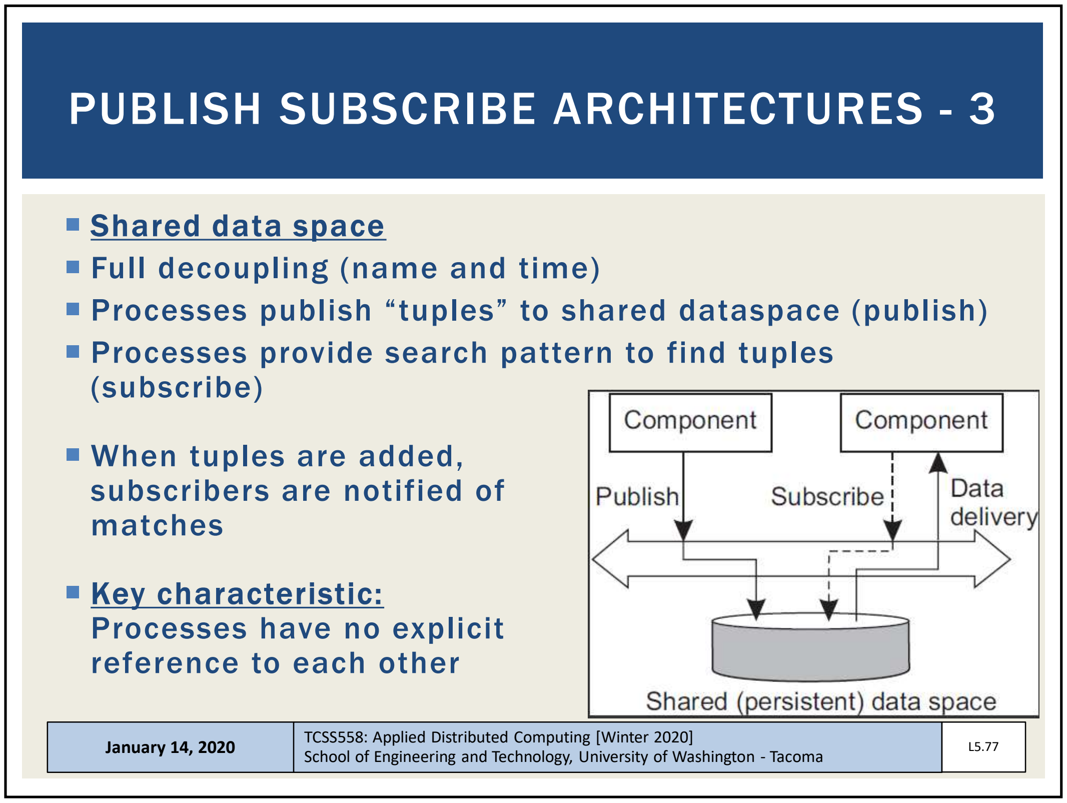
PUBLISH-SUBSCRIBE ARCHITECTURES - 2

- Event-based coordination
- Processes do not know about each other explicitly



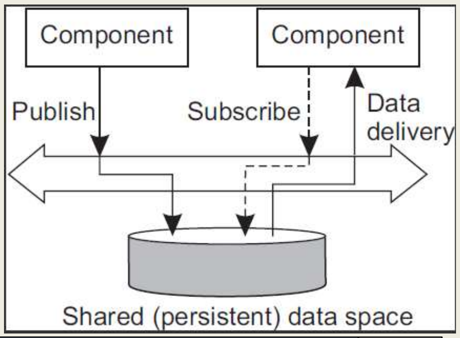
- Processes:
  - Publish: a notification describing an event
  - Subscribe: to receive notification of specific kinds of events
- Assumes subscriber is presently up (*temporally coupled*)

76

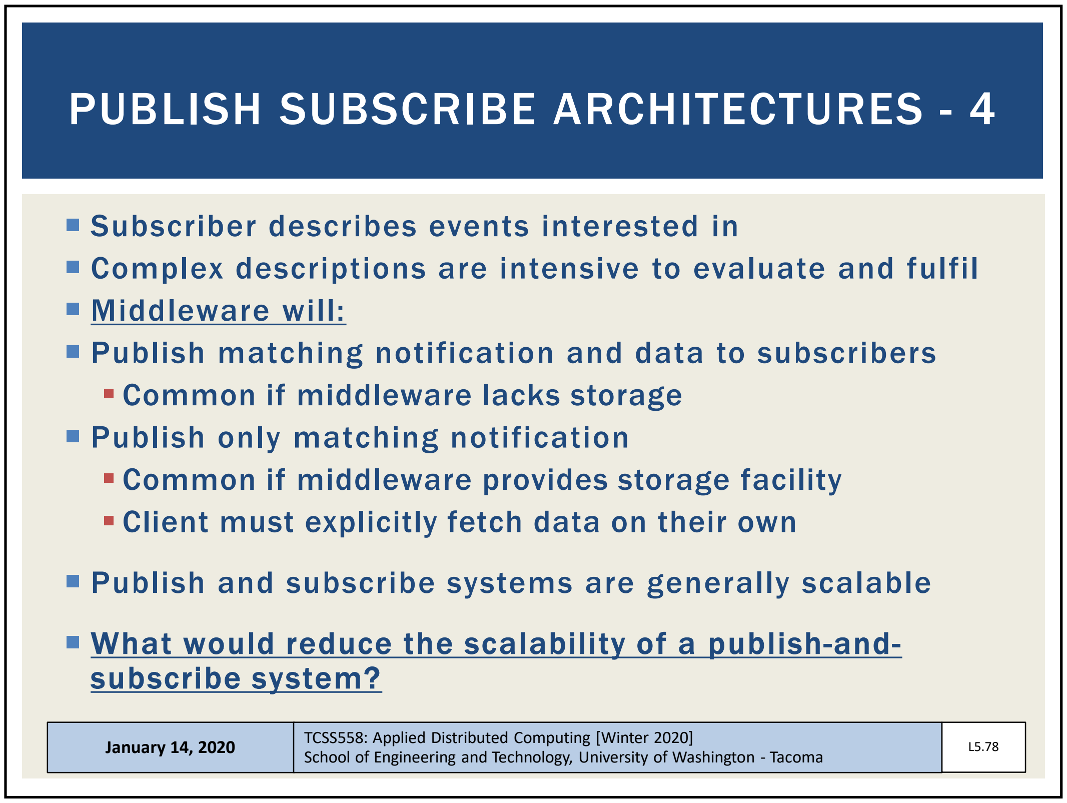


PUBLISH SUBSCRIBE ARCHITECTURES - 3

- Shared data space
- Full decoupling (name and time)
- Processes publish “tuples” to shared dataspace (publish)
- Processes provide search pattern to find tuples (subscribe)
- When tuples are added, subscribers are notified of matches
- Key characteristic: Processes have no explicit reference to each other



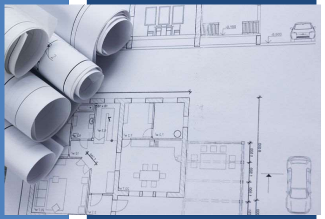
77



PUBLISH SUBSCRIBE ARCHITECTURES - 4

- Subscriber describes events interested in
- Complex descriptions are intensive to evaluate and fulfil
- Middleware will:
- Publish matching notification and data to subscribers
  - Common if middleware lacks storage
- Publish only matching notification
  - Common if middleware provides storage facility
  - Client must explicitly fetch data on their own
- Publish and subscribe systems are generally scalable
- What would reduce the scalability of a publish-and-subscribe system?

78



IN-CLASS ACTIVITY: DISTRIBUTED SYSTEMS ARCHITECTURES

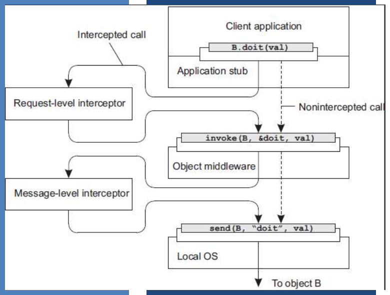
L5.79

79

DISTRIBUTED SYSTEM GOALS TO CONSIDER

- Consider how the architectural change may impact:
- Availability
- Accessibility
- Responsiveness
- Scalability
- Openness
- Distribution transparency
- Supporting resource sharing
- Other factors…

80



CH 2.2: MIDDLEWARE ORGANIZATION

81

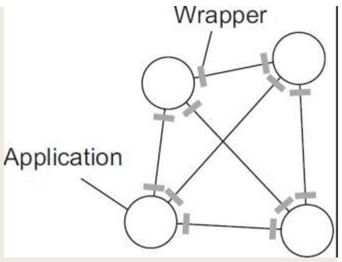
MIDDLEWARE: WRAPPERS

- Wrappers (adapters)
  - Special "frontend" components that provide interfaces to client
  - Interface wrappers transform client requests to "implementation" at the component-level
  - Provide modern services interfaces for legacy code/systems
  - Enable meeting all preconditions for legacy code to operate
  - Parameterization of functions, configuration of environment
- Contributes towards system openness
- Example: Amazon S3
- Client uses REST interface to GET/PUT/DELETE/POST data
- S3 adapts and hands off REST requests to system for fulfillment

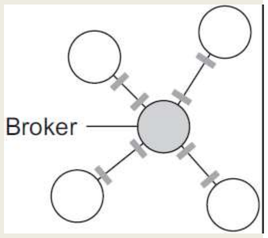
82

MIDDLEWARE: WRAPPERS - 2

- Inter-application communication
  - Application provides unique interface for every application
- Scalability suffers
  - N applications → O(N^2) wrappers

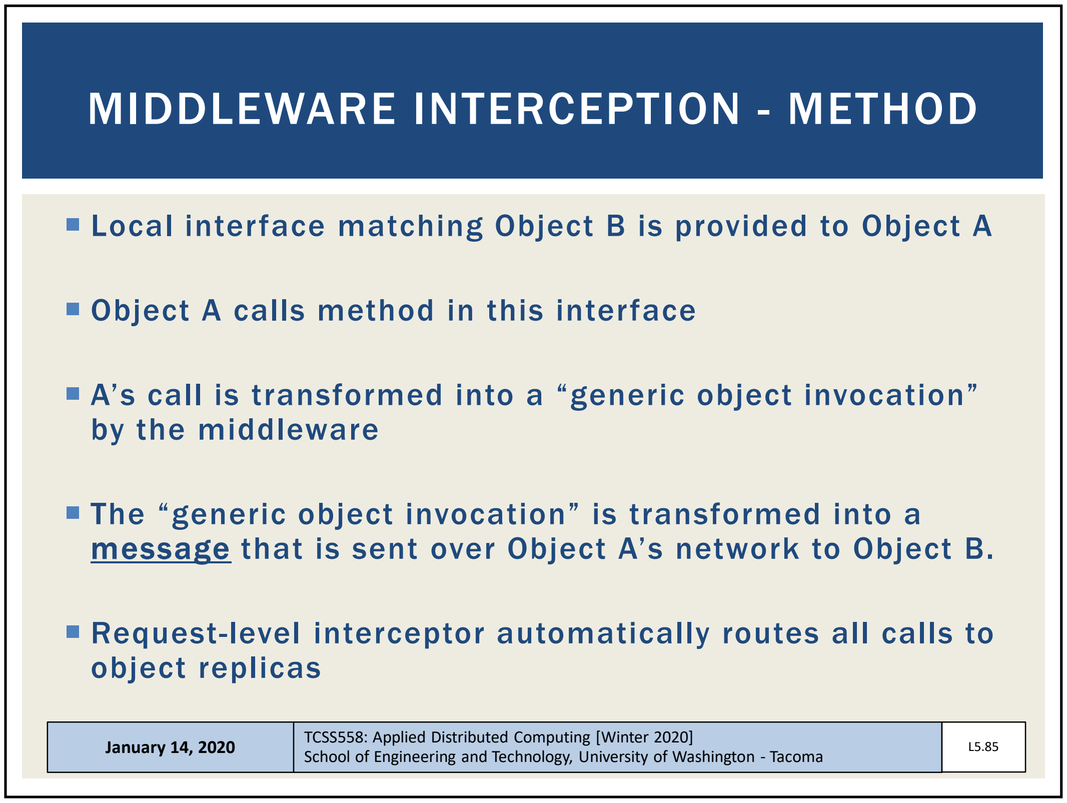


- Broker
  - Provide a common intermediary
  - Broker knows how to communicate with every application
  - Applications only know how to communicate with the broker



MIDDLEWARE: INTERCEPTORS

- Interceptor
- Software construct, breaks flow of control, allows other application code to be executed
- Enables remote procedure calls (RPC), remote method invocation (RMI)
- Object A can call a method belonging to object B on a different machine than A.

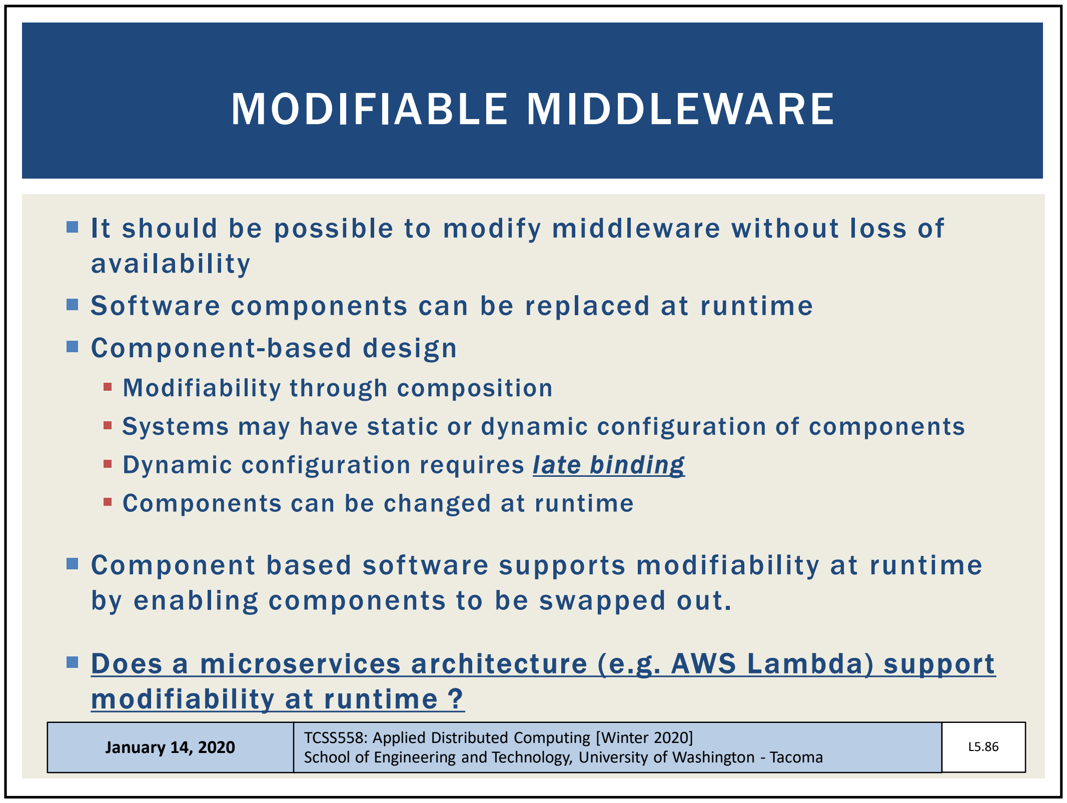


MIDDLEWARE INTERCEPTION - METHOD

- Local interface matching Object B is provided to Object A
- Object A calls method in this interface
- A's call is transformed into a "generic object invocation" by the middleware
- The "generic object invocation" is transformed into a **message** that is sent over Object A's network to Object B.
- Request-level interceptor automatically routes all calls to object replicas

January 14, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L5.85

85

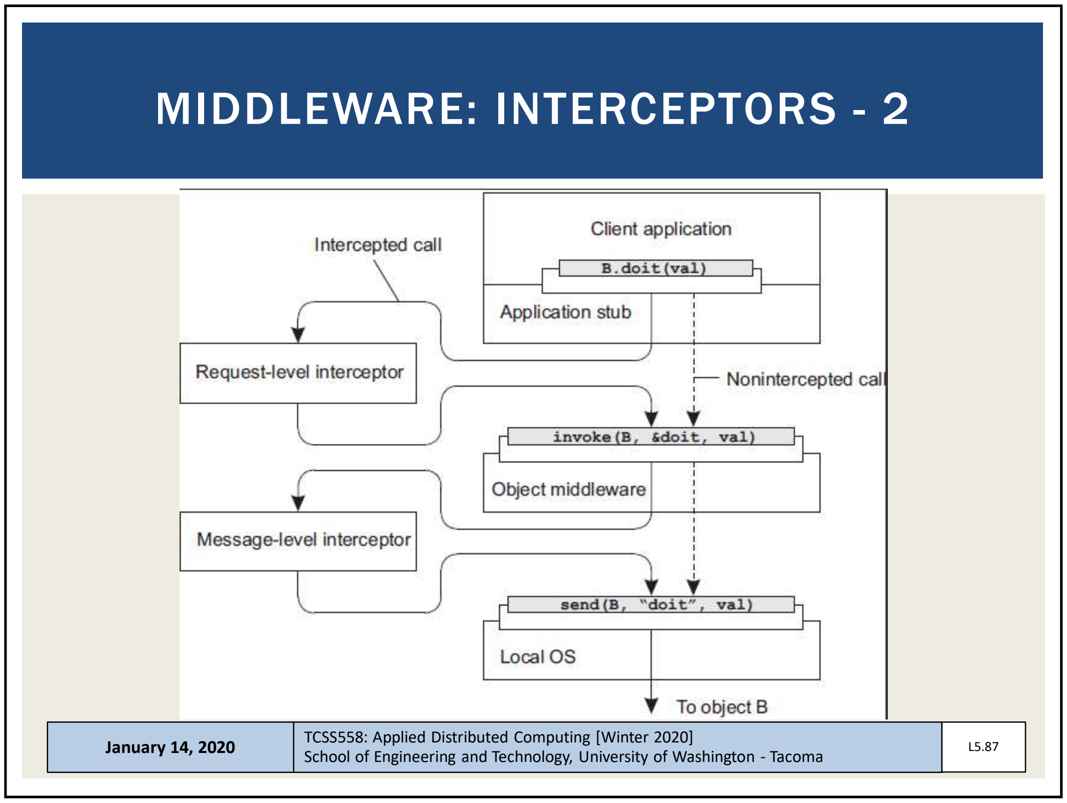


MODIFIABLE MIDDLEWARE

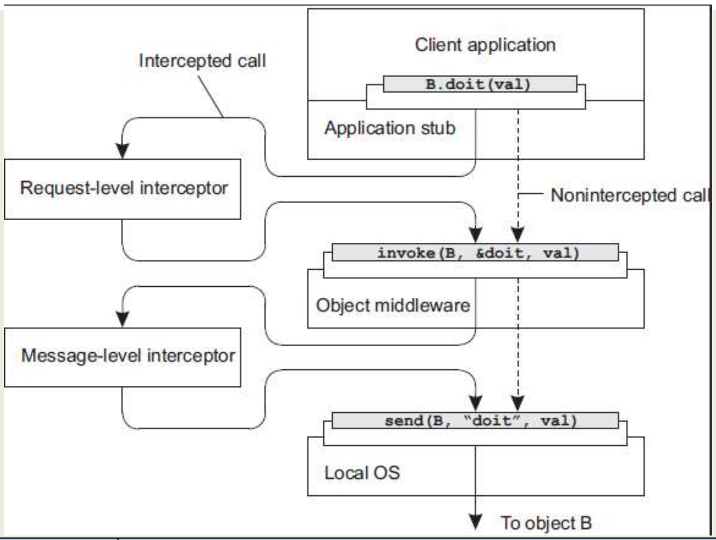
- It should be possible to modify middleware without loss of availability
- Software components can be replaced at runtime
- Component-based design
  - Modifiability through composition
  - Systems may have static or dynamic configuration of components
  - Dynamic configuration requires ***late binding***
  - Components can be changed at runtime
- Component based software supports modifiability at runtime by enabling components to be swapped out.
- **Does a microservices architecture (e.g. AWS Lambda) support modifiability at runtime ?**

January 14, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L5.86

86

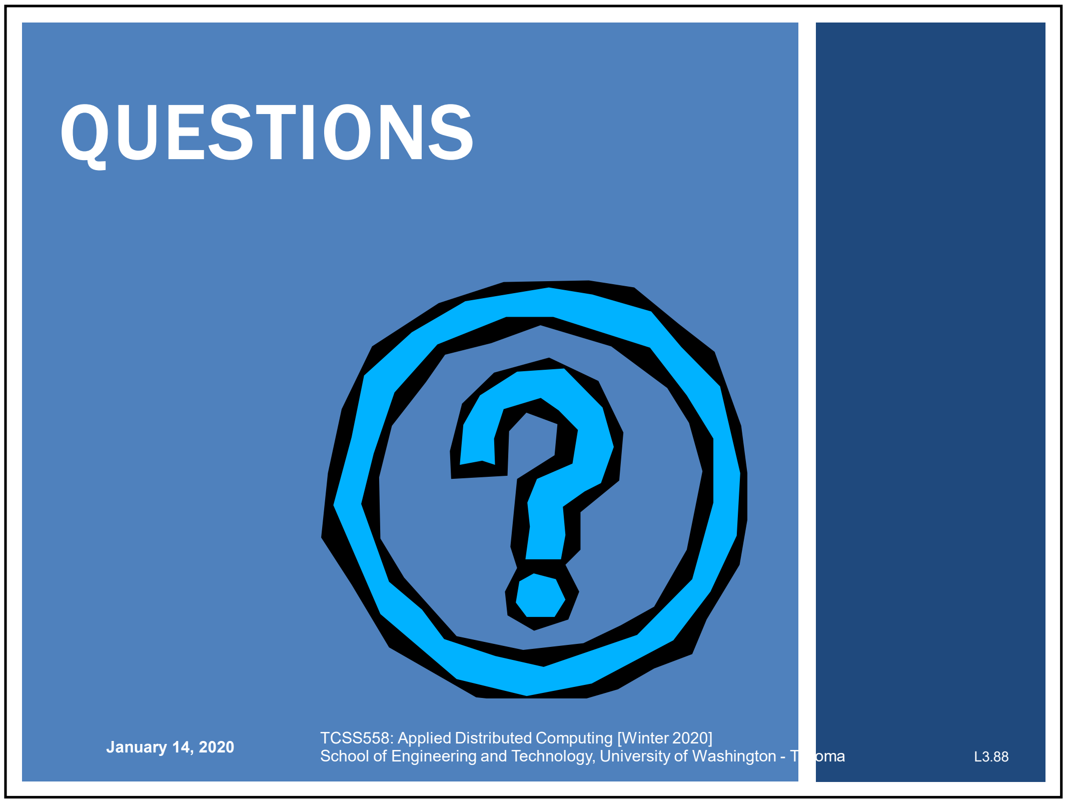


MIDDLEWARE: INTERCEPTORS - 2



January 14, 2020 | TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma | L5.87

87



QUESTIONS

January 14, 2020 TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma L3.88

88

EXTRA SLIDES

FEEDBACK – 9/28

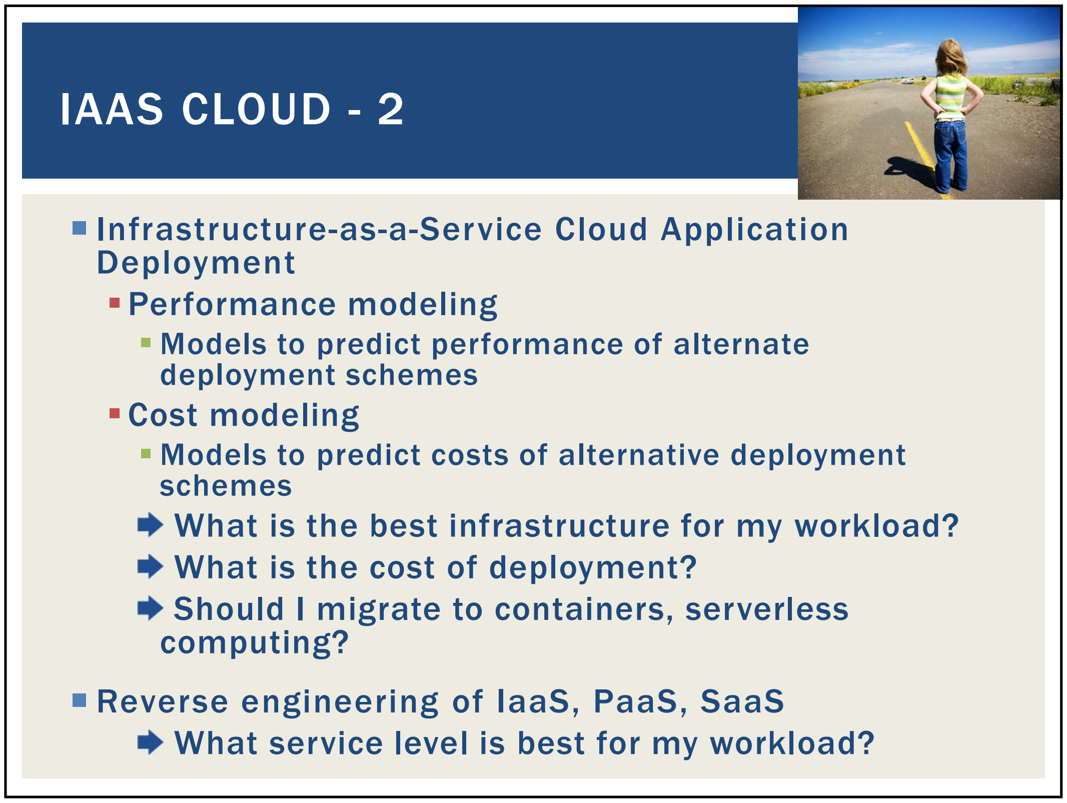
- What is the difference between extensibility and scalability?
  - Extensibility – ability for a system implementation to be extended with additional functionality
  - Scalability – ability for a distributed system to scale (up or down) in response to client demand
- What is the loss of availability in a distributed system?
  - Availability refers to "uptime"
  - How many 9s
  - (1 – (down time/ total time)) * 100%
- Transparency: term is confusing
  - Generally means "exposing everything", obfuscation is better
  - Distribution transparency means the implementation of the distribution cannot be seen

FEEDBACK - 2

- What do we mean by replication transparency?
  - Resources are automatically replicated (by the middleware/framework)
  - That fact that the distributed system has replica nodes is unbeknownst to the users
- How does replication improve system performance?
  - By replicating nodes, system load is “distributed” across replicas
  - Distributed reads – many concurrent users can read
  - Distributed writes – when replicating data, requires synchronization of copies

RESEARCH DIRECTIONS

- Serverless Computing: FaaS, CaaS, DBaaS
- Containerization, Container Platforms
- Infrastructure-as-a-Service (IaaS) Cloud
- Resource profiling, Measurement, Cloud System Data Analytics
- Application performance and cost modeling
- Autonomic infrastructure management to optimize cost and performance
- Cloud Federation, Workload Consolidation, Green Computing
- Virtualization / Unikernel operating systems
- Domains:
- Bioinformatics (genomic sequencing)
- Environmental modeling (USDA, USGS modeling applications)



IAAS CLOUD - 2

- Infrastructure-as-a-Service Cloud Application Deployment
  - Performance modeling
    - Models to predict performance of alternate deployment schemes
  - Cost modeling
    - Models to predict costs of alternative deployment schemes
    - ➡ What is the best infrastructure for my workload?
    - ➡ What is the cost of deployment?
    - ➡ Should I migrate to containers, serverless computing?
- Reverse engineering of IaaS, PaaS, SaaS
  - ➡ What service level is best for my workload?

93