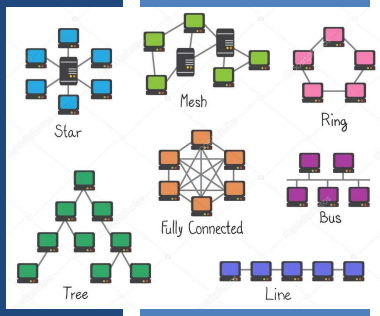
TCSS 558: APPLIED DISTRIBUTED COMPUTING

Distributed Systems: Types and Architectures



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

1

OBJECTIVES

- Feedback from 1/14
- Homework 0
- Chapter 2: Distributed System Architectures
- Research directions
  - Serverless Computing
  - Containerization
  - Infrastructure-as-a-Service
  - Others

2

MATERIAL / PACE

- Please classify your perspective on material covered in today's class (8 respondents, some missing?):
- 1-mostly review, 5-equal new/review, 10-mostly new
- **Average – 7.25**

- Please rate the pace of today's class:
- 1-slow, 5-just right, 10-fast
- **Average – 5.5**

FEEDBACK FROM 1/14

- Can ppt be uploaded before class?
  - Will try to accommodate
  - There may be minor changes made after initial posting

- Questions from 1/14?

5

TYPES OF DISTRIBUTED SYSTEMS

- HPC, Cluster, Grid, Cloud
- Distributed information systems
  - Feature transactions (all –or- nothing)
  - Feature Application Integration methods: Shared files, DBs, RPC, RMI, Message-oriented middleware
- Pervasive Systems
  - Ubiquitous computing systems
  - Mobile systems
  - Sensor networks

6

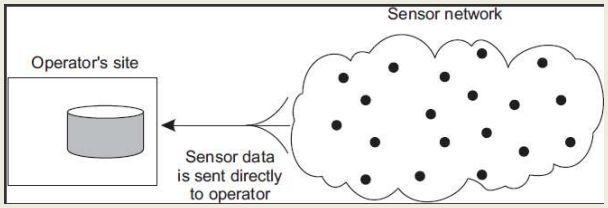
EXAMPLES OF DISTRIBUTED SYSTEMS

- **Classify the following types of distributed systems:**
- Web search engine
- Assisted living home monitoring system for elderly
- Ecommerce websites: e.g. eBay, Amazon
- Wikipedia: online encyclopedia
- Amazon Elastic Compute Cloud (EC2)
- Massively multiplayer online games (MMOG)
- Seismic monitoring network: warning system for earthquakes
- Worldwide Large Hadron Collider (LHC) Computing Grid
- Hospital health informatics and records system
- Canvas: web based learning environment
- Modern automobile with self-driving features

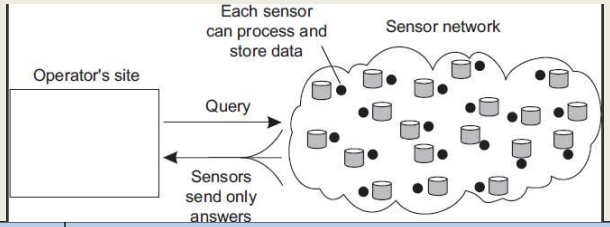
7

WHAT ARE SOME TRADEOFFS FOR CENTRALIZED VS. DECENTRALIZED DATA STORAGE? EXAMPLE: SENSOR NETWORKS

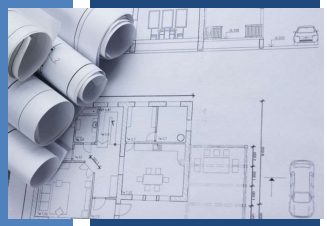
- Centralized:



- Decentralized:



8



CH. 2.1: ARCHITECTURAL STYLES

L4.9

9

DISTRIBUTED SYSTEM ARCHITECTURES

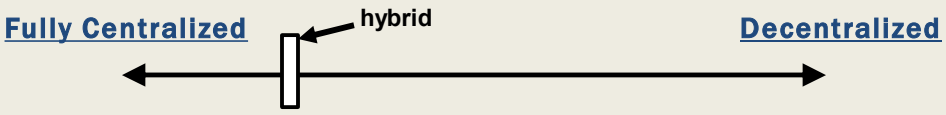
- Provides logical organization of a distributed system into software **components**
- **Logical**: How system is perceived, modeled
  - *The OO/component abstractions*
  - *The "idealists" view of the system*
- **Physical** – how it really exists
  - The "realist" view of the system
- Middleware
  - Helps separate application from platforms
  - Helps organize and assemble distributed components
  - Helps components communicate
  - Enables system to be extended
  - Supports replication within the distributed system
  - Provides "realization" of the architecture

January 16, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L4.10

10

CENTRALIZED VS. DECENTRALIZED DISTRIBUTED SYSTEM ARCHITECTURE

- Tradeoff space: degree of distribution of the system



- Single point-of-failure
- No nodes: vertical scaling
- Always consistent
- Less available (fewer 9s)
- Immediate updates
- No data partitions

- Multiple failure points
- Nodes: horizontal scaling
- Eventually consistent
- More available (more 9s)
- Rolling updates
- Data partitioned or replicated

11

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

12

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)

13

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)

14

DISTRIBUTED SYSTEM GOALS TO CONSIDER

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

15

LAYERED ARCHITECTURES

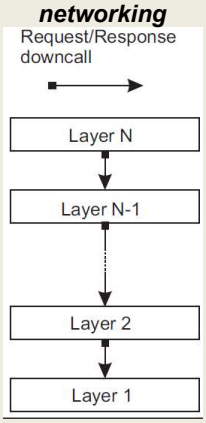
- Components organized in layers
- Component at layer L_j downcalls to lower-level components at layer L_i (where $i < j$)
- Calls go down
- Exceptional cases may produce upcalls

16

LAYERED ARCHITECTURES - 2

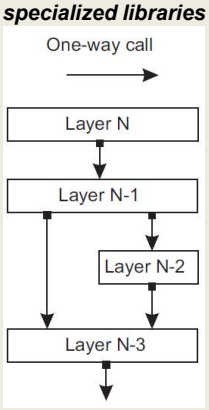
Pure-layered Organization

networking



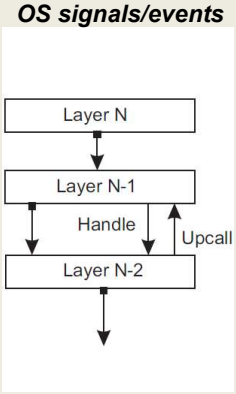
Mixed-layered organization

specialized libraries



Layered w/ upcalls organization

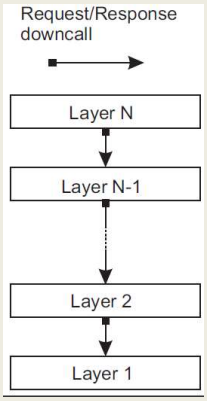
OS signals/events



17

LAYERED ARCHITECTURES - 3

- Consider an architecture with 5 layers
- Does a client interacting with "Layer 5" of the distributed system need to be concerned with details regarding the implementation of lower layers *(layers 1, 2, 3, 4)* ?



18

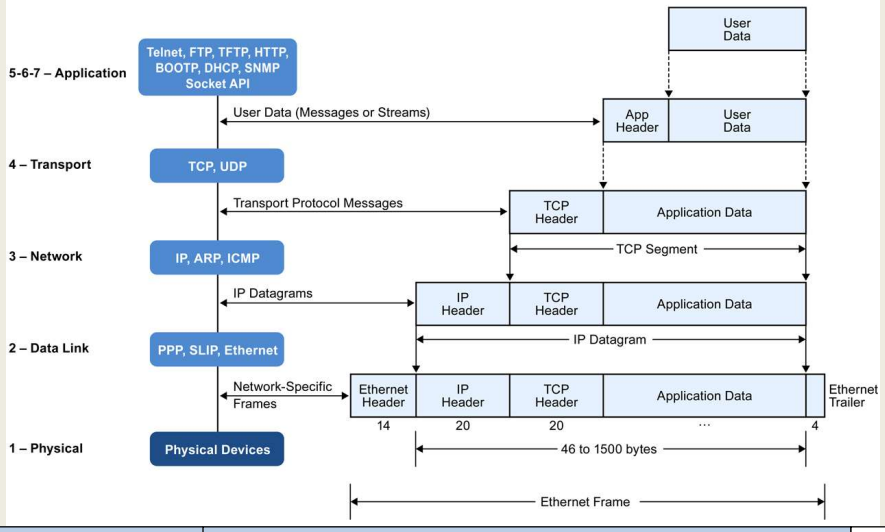
COMMUNICATION-PROTOCOL STACKS

- 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 **service**
- **Interface** makes service available
- **Protocol** implements communication for a layer

- **New services can be built atop of existing layers to reuse lower level implementation(s)**
- Abstractions make it easier to reuse existing layers which already implement communication basics

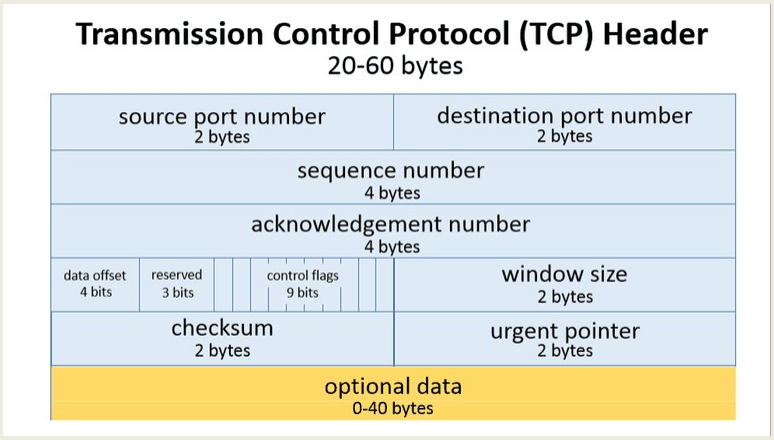
19

HOW A NETWORK PACKET IS BUILT



20

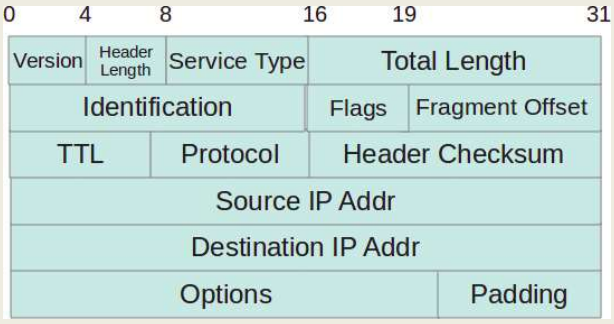
TCP HEADER



21

IP HEADER

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



22

TRANSMISSION CONTROL PROTOCOL (TCP)

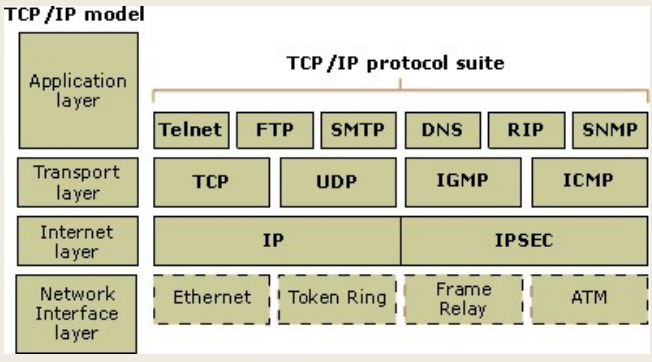
- TCP (layer 4) provides easy to use API
- API supports:
  - setup, tear down of connection(s)
  - sending and receiving of messages
- TCP preserves ordering of transferred data
- TCP detects and corrects lost data

- But TCP is "protocol" agnostic
  - A protocol is a language of messages exchanged to enable communication
  - Application layer communication is programming language agnostic
  - Code can be written in many programming languages to "speak" the "language" of a custom protocol known as an ***APPLICATION PROTOCOL***

- What should the application protocol say?

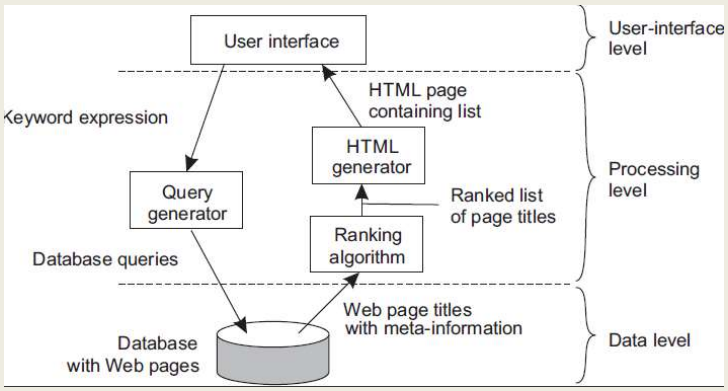
COMMON APPLICATION LAYER PROTOCOLS

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



APPLICATION LAYERING

- Distributed application example: Internet search engine



25

APPLICATION LAYERING

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

26

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

27

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)

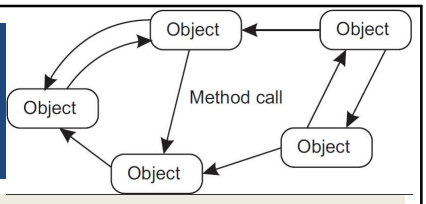
28

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

29

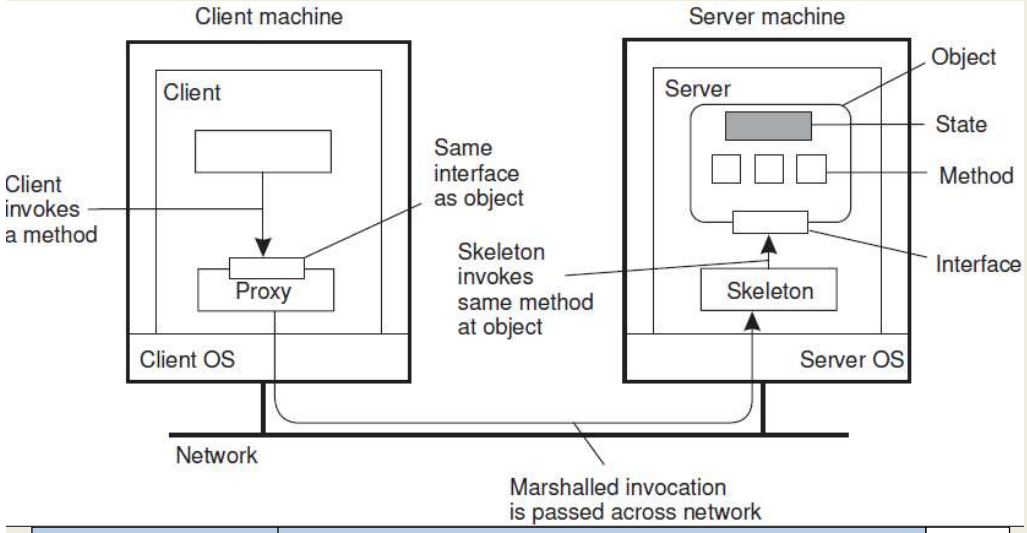
OBJECT-BASED ARCHITECTURES - 2



- 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

30

DISTRIBUTED OBJECTS



31

DISTRIBUTED OBJECTS - 2

- A counterintuitive feature 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 …. *(does this make sense?)*

32

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

SERVICE ORIENTED ARCHITECTURE - 2

- Are more easily developed independently 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

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)

RESOURCE BASED ARCHITECTURES

- Motivation:
  - Increasing number of services available online
  - Each with specific protocol(s), methods of interfacing
  - Connecting services w/ different TCP/IP protocols → integration nightmare
    - Need for specialized client for each service that speaks the application protocol "language"…
- Need standardization of interfaces
  - Make services/components more pluggable
  - Easier to adopt and integrate
  - Common architecture

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

37

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*

38

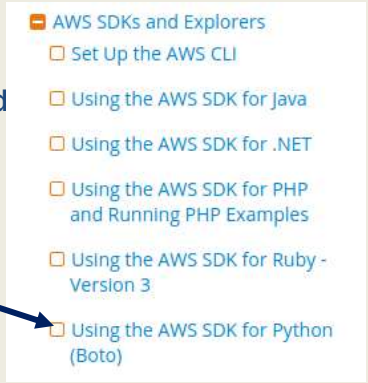
REST-FUL OPERATIONS

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

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



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/

41

```
// 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>
```

L4.42

42

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

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

L4.43

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 16, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L4.44

44

PUBLISH-SUBSCRIBE ARCHITECTURES

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

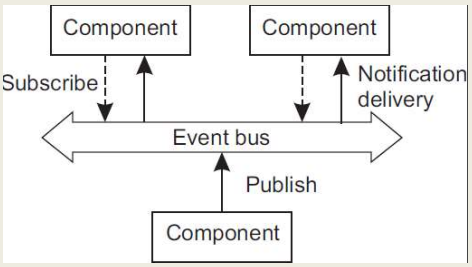
	Temporally coupled (at the same time)	Temporally decoupled (at different times)
Referentially coupled (*dependent on name*)	**Direct** Explicit synchronous service call	**Mailbox** Asynchronous by name (address)
Referentially decoupled (*name not required*)	**Event-based** Event notices published to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

Publish and subscribe architectures

45

PUBLISH-SUBSCRIBE ARCHITECTURES - 2

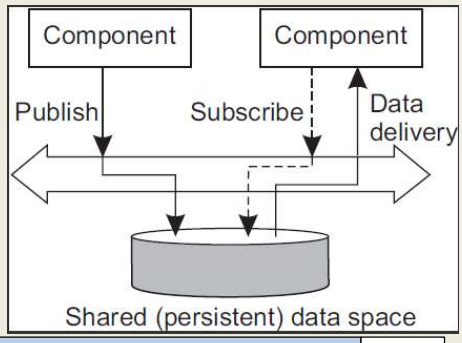
- **Event-based coordination**
- Processes do not know about each other explicitly
- **Processes:**
  - **Publish:** a notification describing an event
  - **Subscribe:** to receive notification of specific kinds of events
- Assumes subscriber is presently up (*temporally coupled*)
- Subscribers must actively **MONITOR** event bus



46

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



47

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?**

48

RESEARCH DIRECTIONS

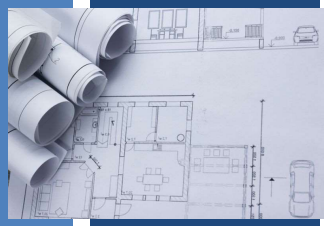
49

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.

50



IN-CLASS ACTIVITY: DISTRIBUTED SYSTEMS ARCHITECTURES

L5.51

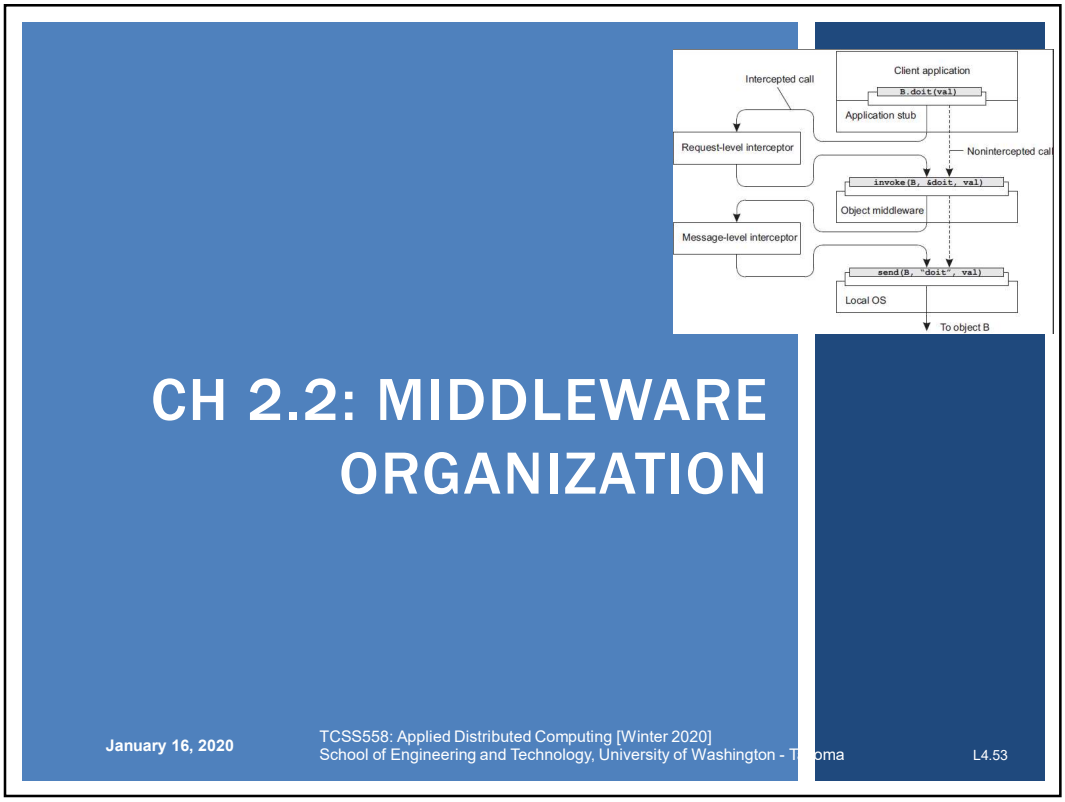
51

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 16, 2020	TCSS558: Applied Distributed Computing [Winter 2020] School of Engineering and Technology, University of Washington - Tacoma	L4.52

52



53

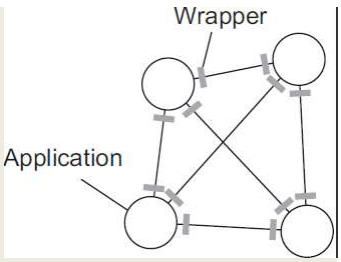
MIDDLEWARE: WRAPPERS

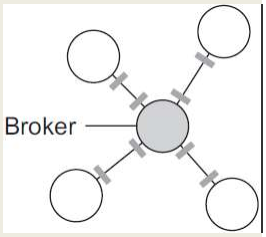
- **Wrappers (adapters)**
  - Special "frontend" components that provide interfaces to client
  - Interface wrappers transform client requests to "implementation" at the component-level
  - Provide modern services interfaces for legacy code/systems
  - Enable meeting all preconditions for legacy code 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

54

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





55

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.

56

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

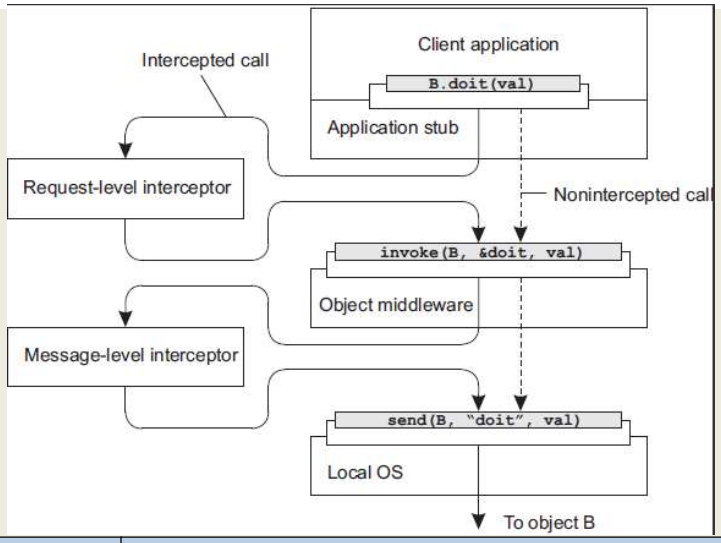
57

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 ?**

58

MIDDLEWARE: INTERCEPTORS - 2



59

QUESTIONS

60

EXTRA SLIDES

61

61

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

62

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?

65