HTTP DELETE

- Removes the resource identified by the URI

- DELETE http://localhost/books/ISBN-0011
  - Delete book identified by ISBN-0011

Representations

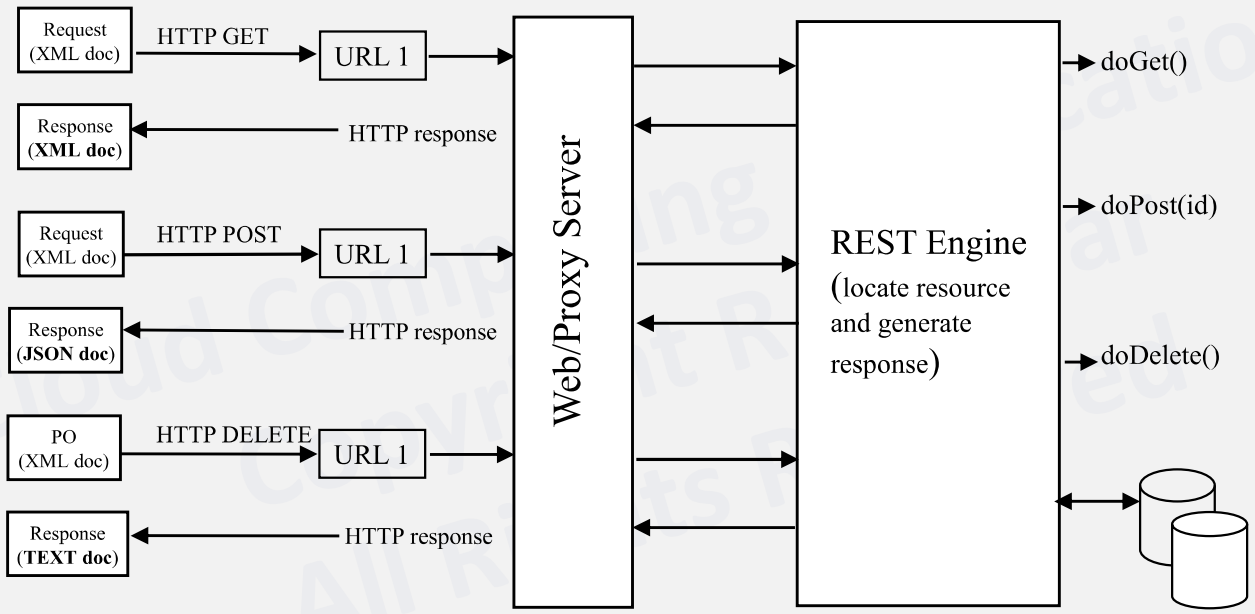
- How data is represented or returned to the client for presentation
- Two main formats:
  - JavaScript Object Notation (JSON)
  - XML
- It is common to have multiple representations of the same data
- XML

```
<COURSE>
        <ID>CS2650</ID>
        <NAME>Distributed Multimedia Software</NAME>
</COURSE>
```

- JSON

```
{course
        {id: CS2650}
        {name: Distributed Multimedia Software}
}
```

Architecture Style



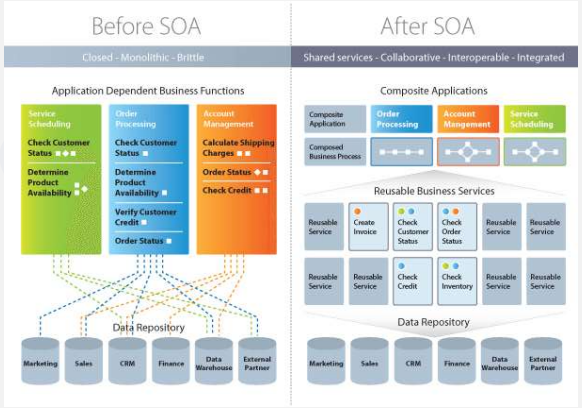
CLOUD COMPUTING APPLICATIONS

Cloud Computing Glue: Service Oriented Architecture and SOAP

Prof. Reza Farivar

Service Oriented Architecture

- Came out of the needs of the business sector, enterprise and B2B applications
  - *"SOA is the philosophy of encapsulating application logic in services with a uniformly defined interface and making these publicly available via discovery mechanisms."*
- Benefits of SOA
  - Reusable Code
  - Interaction
  - Scalability
  - Reduce Costs
- The term "Web Services" typically relates to this type of communication
  - Web Services are one option to implement SOA
  - Other options include: Java Business Integration (JBI), Windows Communication Foundation (WCF) and data distribution service (DDS)
- An example Technology was / is SOAP



Simple Object Access Protocol (SOAP)

- SOAP-based Web APIs use XML validation to ensure structural message integrity
  - XML schemas provisioned with WSDL documents
- Evolved as the successor to XML-RPC
- Characteristics:
  - Extensible
  - Neutrality (from transport layer)
    - Can run on HTTP, WebSocket, even SMTP
  - Independence

```
POST /InStock HTTP/1.1
Host: www.example.org
Content-Type: application/soap+xml; charset=utf-8
Content-Length: 299
SOAPAction: "http://www.w3.org/2003/05/soap-envelope"

<?xml version="1.0"?>
<soap:Envelope xmlns:soap="http://www.w3.org/2003/05/soap-envelope" xmlns:m="http://www.example.org">
  <soap:Header>
  </soap:Header>
  <soap:Body>
    <m:GetStockPrice>
      <m:StockName>T</m:StockName>
    </m:GetStockPrice>
  </soap:Body>
</soap:Envelope>
```

Example Message

** From Wikipedia*

- Use XML validation to ensure structural message integrity
  - XML schemas provisioned with WSDL documents
- Evolved as the successor to XML-RPC
- Characteristics:
  - Extensible
  - Neutrality (from transport layer)
    - Can run on HTTP, WebSocket, even SMTP
  - Independence

Simple Object Access Protocol (SOAP)

- SOAP-based Web APIs use XML validation to ensure structural message integrity
  - XML schemas provisioned with WSDL documents
- Evolved as the successor to XML-RPC
- Characteristics:
  - Extensible
  - Neutrality (from transport layer)
    - Can run on HTTP, WebSocket, even SMTP
  - Independence

```
<!-- Abstract interfaces -->
   <interface name="Interface1">
      <fault name="Error1" element="tns:response"/>
      <operation name="Get" pattern="http://www.w3.org/ns/wsdl/in-out">
         <input messageLabel="In" element="tns:request"/>
         <output messageLabel="Out" element="tns:response"/>
      </operation>
   </interface>

<!-- Concrete Binding Over HTTP -->
   <binding name="HttpBinding" interface="tns:Interface1"
            type="http://www.w3.org/ns/wsdl/http">
      <operation ref="tns:Get" whttp:method="GET"/>
   </binding>

<!-- Concrete Binding with SOAP-->
   <binding name="SoapBinding" interface="tns:Interface1"
            type="http://www.w3.org/ns/wsdl/soap"
            wsoap:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP/"
            wsoap:mepDefault="http://www.w3.org/2003/05/soap/mep/request-response">
      <operation ref="tns:Get" />
   </binding>
```

Example WDSL

** From Wikipedia*

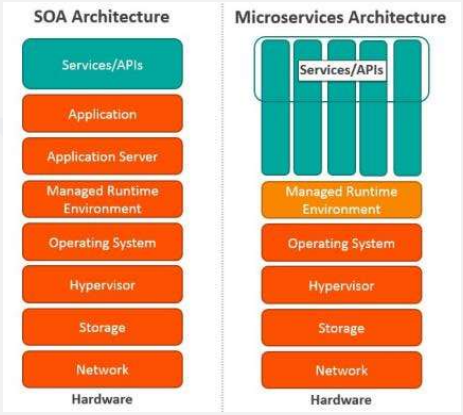
- Use XML validation to ensure structural message integrity
  - XML schemas provisioned with WSDL documents
- Evolved as the successor to XML-RPC
- Characteristics:
  - Extensible
  - Neutrality (from transport layer)
    - Can run on HTTP, WebSocket, even SMTP
  - Independence

Simple Object Access Protocol (SOAP)

- It's popularity has somewhat diminished, but still very relevant in enterprise applications
  - SOAP is still used most often in the enterprise world, where communication between different services needs to *conform to a set of rules and contracts (*)*
  - Because it follows objects, rules, and constraints, SOAP is a more strict *(*)* protocol than REST
  - ** But do enterprises really conform to strict rules? Agile seems to work best in practice*
    - It might have been too rigid for its own good
- E.g. Salesforce SOAP API to create, retrieve, update or delete records, such as accounts, leads, and custom objects
- E.g. The PayPal SOAP API is based on open standards known collectively as web services, which include the Simple Object Access Protocol (SOAP), Web Services Definition Language (WSDL), and the XML Schema Definition language (XSD)

SOA and MicroServices

- MicroService Architecture is very similar, modern reincarnation of SOA
  - SOA mainly 2000~2010
  - MicroServices 2015~…
  - *"Microservices are the kind of SOA we have been talking about for the last decade. Microservices must be independently deployable, whereas SOA services are often implemented in deployment monoliths."* - *Torsten Winterberg*
- What has changed? Adoption of:
  - Containerization
  - Asynchronous Programming
  - Distributed Computing mindset
  - CICD and Agile workflows



CLOUD COMPUTING APPLICATIONS

Cloud Computing Glue: Asynchronous RPC, WebSocket

Prof. Reza Farivar

Asynchronous RPC, aka. Streaming API

- Using old HTTP/0.9 and 1.0, what if the remote server takes a long time?
  - The client can wait, blocked, keeping the HTTP connection open (long polling)
  - It can keep polling the server periodically
  - Ultimately, these are hacks
  - HTTP/1.1 not ready for real-time web
- Web 2.0: Bidirectional client / server communication
  - AJAX
  - Comet
    - Umbrella term: Ajax Push, Reverse Ajax, Two-way-web, HTTP Streaming, and HTTP server push, …
  - Push notifications
  - XMLHttpRequest
    - XMLHttpRequest (XHR) is an API in the form of an object whose methods transfer data between a web browser and a web server.
    - The object is provided by the browser's JavaScript environment.
    - jQuery provides a nice wrapper (it also encapsulates WebSockets and server push)

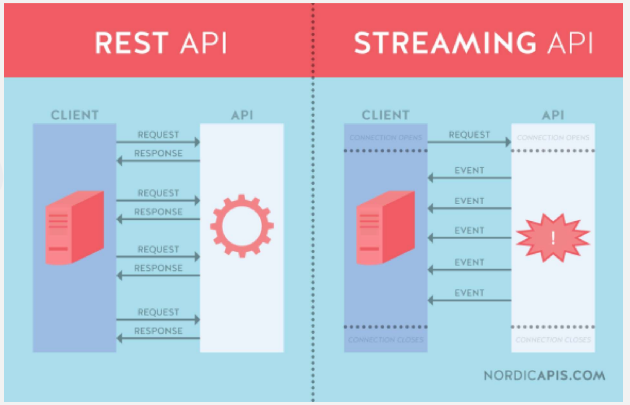


Image from https://nordicapis.com/rest-vs-streaming-apis-how-they-differ/

WebSocket

- RPC libraries can issue asynchronous methods to the server, and the server can inform them of the response later
- Streaming Architecture
  - Minimizing latency
- WebSocket
  - Part of HTML 5 standard
  - Can handle interactive sessions better than RESTful architecture
- Use cases:
  - Chat
  - Stock price update
  - Collaborative Document Editing
  - Location update (I am here now)
  - Multiplayer games

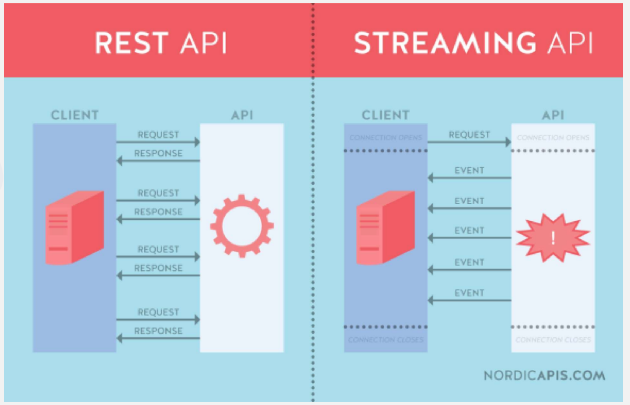


Image from https://nordicapis.com/rest-vs-streaming-apis-how-they-differ/

WebSocket Protocol

- WebSocket is an application protocol, running on top of TCP
  - It uses URIs, but not http://
  - Instead, uses ws://
- It utilizes an initial HTTP session and HTTP port numbers to process handshake phase
- The protocol has 3 phases
  - Opening Handshake
  - Data Transfer
  - Closing Handshake
- Protocol “upgrade”
- Purely event driven
  - Application code listens for events on WebSocket objects to handle incoming data and changes in connection status
- Asynchronous programming
  - Client does not need to do anything (e.g. poll) to receive data

WebSocket 3 phases

1) Opening Handshake
   - HTTP request/response to open WebSocket connection
   - Example client request

```
GET /chat HTTP/1.1
Host: example.com
Upgrade: websocket
Connection: Upgrade
Origin: http://example.com
Sec-WebSocket-Key: dGhlIHNhbXBsZSBub25jZQ==
Sec-WebSocket-Version: 13
```

   - Example server response

```
HTTP/1.1 101 Switching Protocols
        Upgrade: websocket
        Connection: Upgrade
Sec-WebSocket-Accept: s3pPLMBiTxaQ9kYGzzhZRbK+xOo=
```

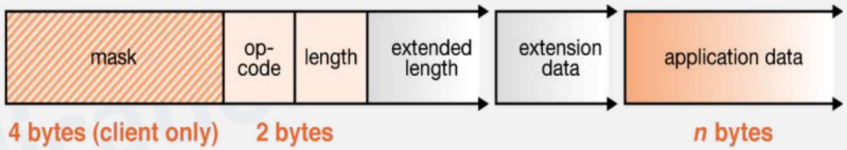
   - HTTP protocol is switched (aka. upgraded) to WebSocket

2) Data Transfer
- Bidirectional communication
- WebSocket frame
- Description of fields:
  - Op-code: Continuation, Text, Binary, Close, Ping, Pong

To keep the connection alive

3) Closing handshake
   - A WebSocket frame with opcode 0x8 is sent

WebSocket Frame



WebSocket API

- Simple Javascript W3C WebSocket API
- four different events:
  - Open - fires to establish a connection
  - Message - contains the data from the server
  - Error - fires in response to an unexpected event (failure)
  - Close - fires when the WebSocket connection is closed

- Primary methods + events
  - `WebSocket(URL, [protocols])` – Create a connection
    - `var ws = new WebSocket("ws://www.websocket.org","SOAP");//"SOAP" is optional`
  - `onOpen()` – WebSocket opened
    - `ws.onOpen = function(e){ console.log(ws.protocol);}`
  - `Send(data)` – Send data (string, Blob or Arraybuffer)
    - `ws.send("Hello WebSocket!");`
  - `onMessage()` – Message received
    - ws.onMessage = function(e) { log("Message received: " + e.data); ws.close(); }
  - `onClose()` – Close message received
    - `ws.onOpen = function(e){ console.log ("Disconnected: "+ e.reason);}`
  - `onError()` - Error

WebSocket

- Note that once you allow asynchronous communication in a networked environment, you should handle faults
- There are wrapper packages, encapsulating WebSockets functionality with additional features
  - Node.js supports WebSockets through plugins
  - Example: Socket.io.
  - Consisting of a Node.js server and a Javascript client library, socket.io provides reliability for handling proxies and load balancers as well as personal firewall and antivirus software and even supports binary streaming.
- Java 11 supports both HTTP as well as WebSocket protocols

WebSocket in Cloud Computing

- Many Cloud providers support WebSocket as the API of choice for interactive sessions with the Cloud service and the client
  - Amazon AWS API Gateway
  - Salesforce
  - Many cloud video vendors
    - E.g. easylive.io
  - Basis of Slack and its "Real Time Messaging API"
    - Slack has a simpler Event API based on HTTP/2 Push, but only the RTM guarantees real time delivery of messages
  - Google App Engine Channel API used WebSocket to allow server to client messaging
    - *Now deprecated, replaced by Firebase (a MBaaS solution)*
    - We can always run a Socket.io instance on Computer Engine (or EC2) instance
      - https://cloud.google.com/solutions/real-time-gaming-with-node-js-websocket
    - Or use AWS API Gateway to handle the deployment of WebSocket servers for us

CLOUD COMPUTING APPLICATIONS

Cloud Computing Glue: HTTP2 Push, Streaming Video

Prof. Reza Farivar

HTTP/2

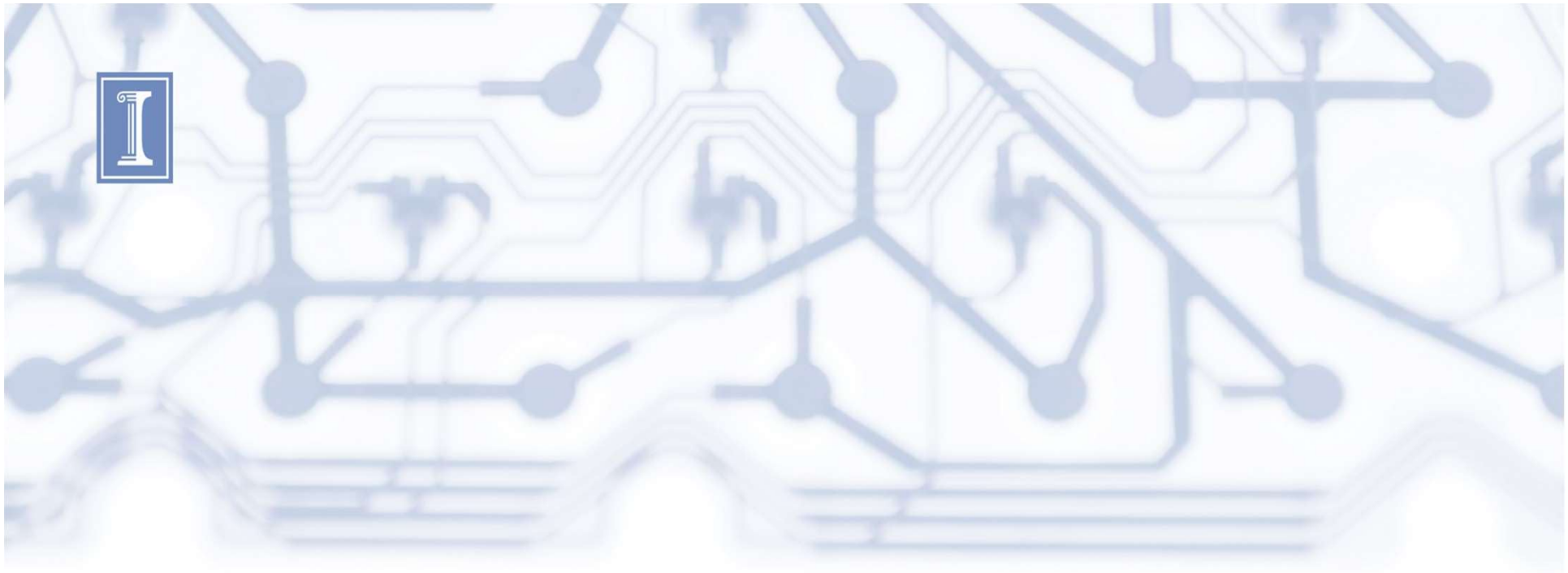
- Published in 2015, most browsers supported it by the end of 2015
  - By 2020, about ~42% of top 10 million websites supported HTTP/2
- One of the main features of HTTP/2 is server push
  - Closely related to another feature of HTTP/2: data streaming
  - HTTP/2 Server push is being progressively implemented, for example Nginx web server implemented it in February 2018.
  - By now, all major servers and browsers support it
- It "proposed" new data in a new stream to the browser, to be stored in a Cache
- It does not send the pushed data directly to the client application itself
- To make the application aware, HTTP/2 utilizes Server-Side Events (SSE)

Amazon AWS API Gateway

- REST API
- WebSocket API
- HTTP API
  - Based on HTTP/2 push and notification

Video Streaming over the Internet

- Streaming Video Content
  - RTMP
    - Still very prevalent, but slowly being phased out
  - RTP (over UDP)
  - HLS (over HTTP)
  - MPEG-DASH
  - WebRTC



CLOUD COMPUTING APPLICATIONS

VPC: Virtual Private Cloud

Prof. Reza Farivar

Virtual Private Clouds

- Most Cloud Providers have some sort of network virtualization solution
  - Arguably the most fundamental building block
  - Amazon Virtual Private Cloud (VPC)
  - Microsoft Azure Virtual Network (VNet)
  - Google Virtual Private Cloud (VPC)
  - Oracle Virtual Cloud Networks
- They somewhat differ in detail, but the concepts are general
- In this lesson we will focus on Amazon VPC
  - Geared towards Cloud Architecture

Solving a Fundamental Problem

- Allow many different users have their own private network in the cloud
- Isolate different customers' network packets from each other
- Solution: VPC

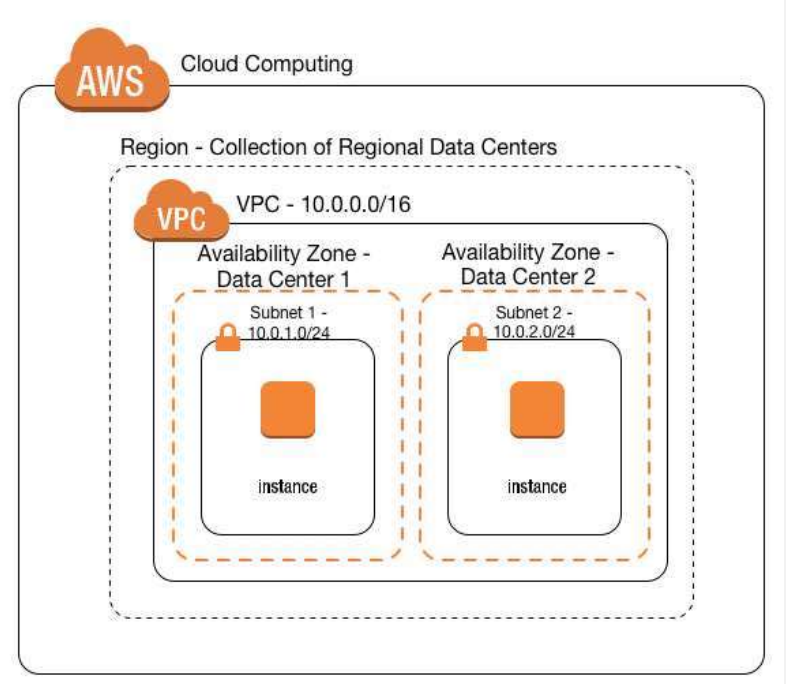
VPCs and Subnets

- You have your “own” network in the cloud
  - VPC
  - In AWS, a VPC is associated with a region, e.g. us-east-1
  - You can have more than one VPC, even in the same region
    - Default 5 quota
  - The IP address range of all the nodes in this VPC can be defined with a CIDR
- Your VPC is subdivided into logically separate segments
  - Subnet
  - Each subnet get a smaller CIDR range
  - Each subnet is associated with one Availability Zone
- You can then launch instances (EC2, RDS, etc.) in a subnet

VPCs, Subnets and Availability Zones

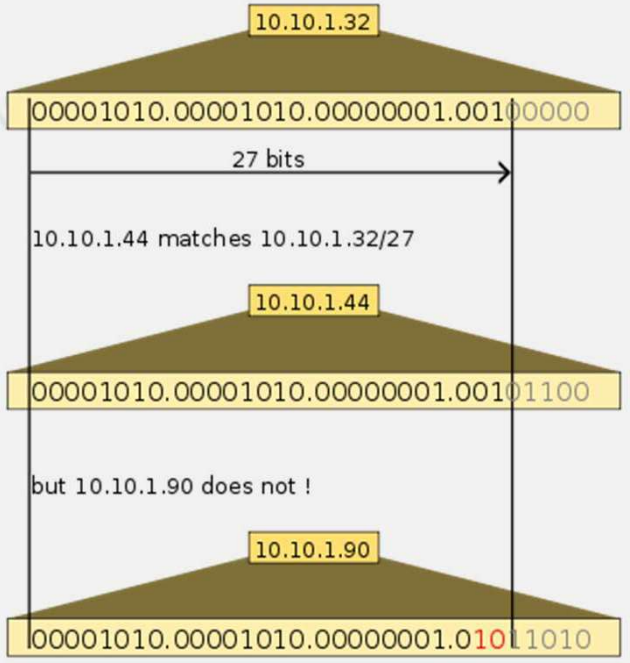
The main routing table, associated with the VPC, has the following route:

Destination	Target
10.0.0.0/16	local



Background Knowledge: CIDR

- CIDR: Classless Inter-Domain Routing
- A method of allocating IP address ranges
- In IPV4, each IP address is a 32 bit value
  - 4 bytes
  - 192.168.0.1
- CIDR notation:
  - 100.101.102.103/24
  - Take the mask (24 bits)
  - Keep the upper 24 bits the same
  - The lower bits can change ➔ range
  - 100.101.102.0 … 100.101.102.255
- IPV6, each address is 128 bits:
  - the IPv6 block *2001:db8::/48* represents the block of IPv6 addresses from *2001:db8:0:0:0:0:0:0* to *2001:db8:0:ffff:ffff:ffff:ffff:ffff*.
  - Note that in IPV6 notation, each segment is written in hex



RFC 1918: Address Allocation for Private Internets

- The Internet Assigned Numbers Authority (IANA) has reserved the following three blocks of the IP address space for private internets
- 10.0.0.0/8
  - 10.0.0.0 - 10.255.255.255
  - Number of addresses: 16,777,216
- 172.16.0.0/12
  - 172.16.0.0 - 172.31.255.255
  - Number of addresses: 1,048,576
- 192.168.0.0/16
  - 192.168.0.0 - 192.168.255.255
  - Number of addresses: 16,777,216
- Why? Because it is guaranteed no other server on the public internet has an IP address in these ranges
- Routing rules do not conflict

RFC 1918 and AWS VPC

- When you create a VPC, you must specify an IPv4 CIDR block for the VPC
- The allowed block size is between a /16 netmask (65,536 IP addresses) and /28 netmask (16 IP addresses)
  - Note that RFC 1918 would allow for 16 million distinct IP addresses in the 10.0.0.0/8, but Amazon would at most accept a /16 netmask in a VPC or subnet

RFC 1918 range	Example AWS VPC CIDR block
10.0.0.0 - 10.255.255.255 (10/8 prefix)	Your VPC must be /16 or smaller, for example, 10.0.0.0/16.
172.16.0.0 - 172.31.255.255 (172.16/12 prefix)	Your VPC must be /16 or smaller, for example, 172.31.0.0/16.
192.168.0.0 - 192.168.255.255 (192.168/16 prefix)	Your VPC can be smaller, for example 192.168.0.0/20.

Reserved IP Adresses

- The first four IP addresses and the last IP address in each subnet CIDR block are not available for you to use, and cannot be assigned to an instance; E.g. for 10.0.0.0/24:
  - 10.0.0.0: Network address
  - 10.0.0.1: The VPC router
  - 10.0.0.2: The IP address of the DNS server is the base of the VPC network range plus two
  - 10.0.0.3: Reserved for future use
  - 10.0.0.255: Network broadcast address
    - AWS does NOT support broadcast in a VPC, therefore this address is reserved

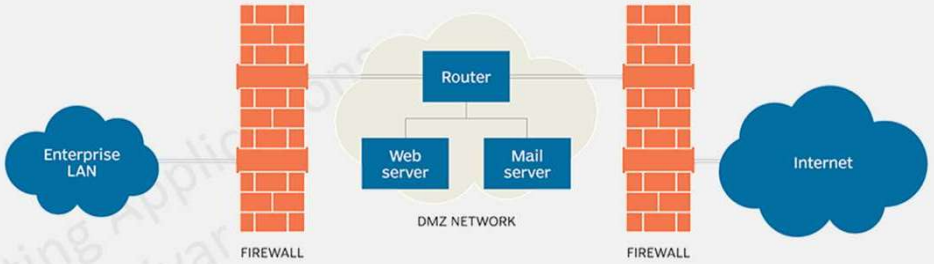
CLOUD COMPUTING APPLICATIONS

VPC: Subnets

Prof. Reza Farivar

Subnets

- Create subnets to isolate resources per the project requirement
  - DMZ/Proxy
  - Load balancer
  - web applications
  - Mail servers
  - Databases



- E.g. have a public subnet to host internet-facing resources and a private subnet for databases that accept web requests
- Create multiple subnets (public or private) in multiple AZs to host a high availability multi-AZ infrastructure and avoid a single point of failure
  - each subnet can communicate with every other subnet in the same VPC

Private Subnets

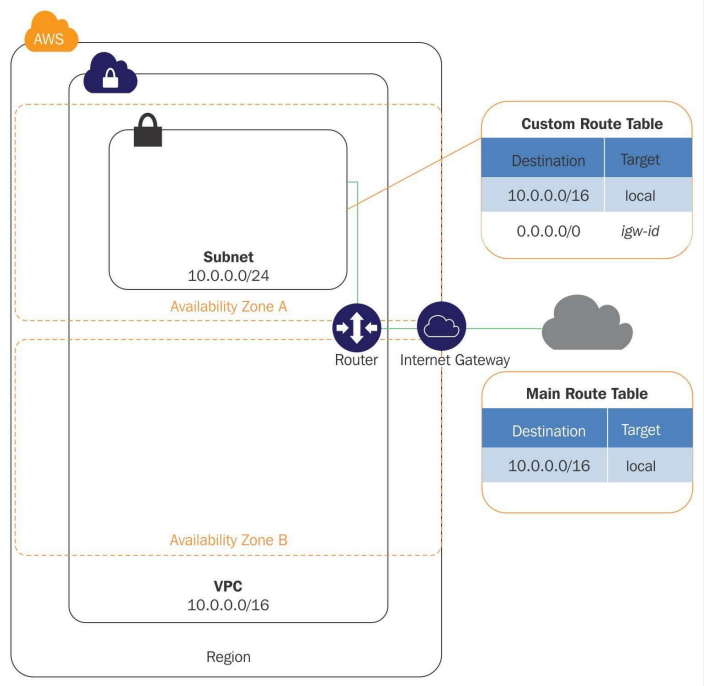
- Any incoming traffic from the internet cannot directly access the resources within a private subnet
- Outgoing traffic from a private subnet cannot directly access the internet
  - Restricted; or
  - Routed through a NAT
- Each resource (instance) gets a private IP
  - From the CIDR range associated with the subnet
- Technically, a subnet is private if there is no route in the routing table to an internet gateway

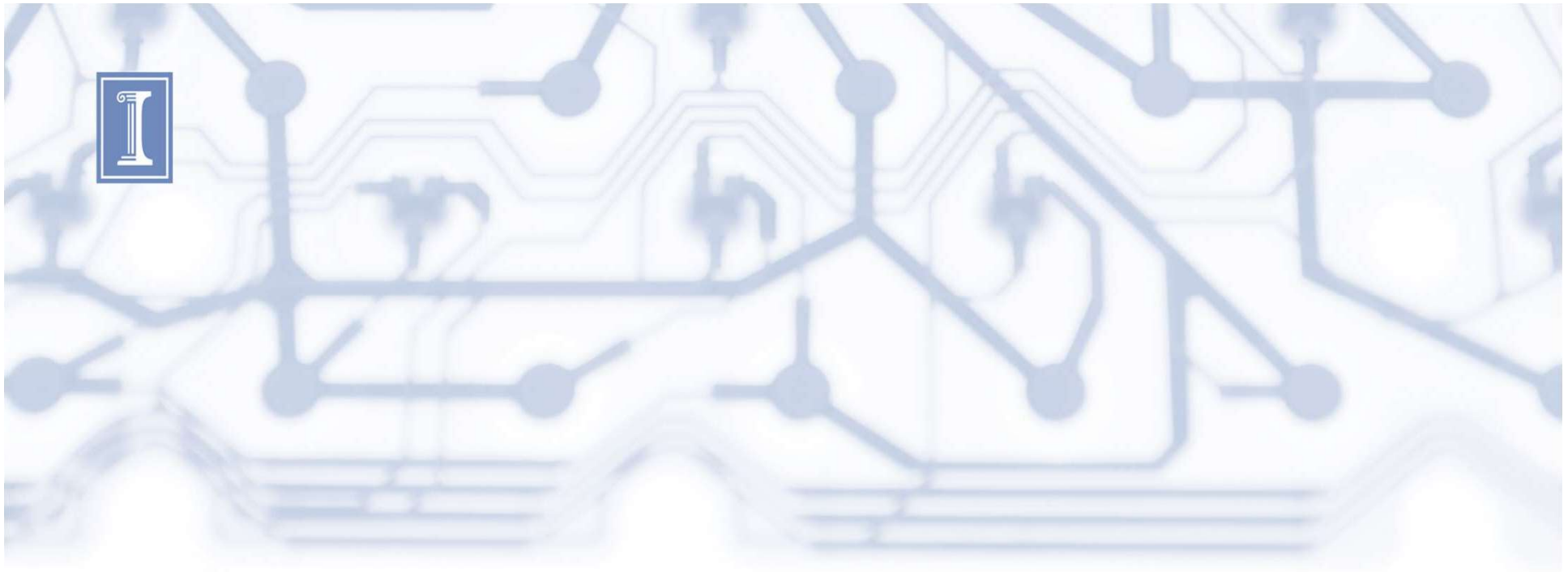
Public Subnet

- A subnet that has access to an internet gateway defined in the routing table
- Each resource in a public subnet gets a private IP within the CIDR range, AND a public IP accessible from the internet
  - the public IP can be dynamic (only remains valid while the instance is alive, and then AWS reclaims it), or
  - It can be an elastic IP, where you pay for it, and it will remain yours even if the instance shuts down
- Outgoing traffic can directly access internet
  - Unlike Private, which needs a NAT to access internet

Route Tables

- Each VPC has an associated “Main Route Table”
  - The Default VPC has a route in its “Main Route Table” to an internet gateway
  - Custom VPCs usually only have the local route in the MRT
- Subnets can have their own custom route table
  - If no custom route table is explicitly associated with a subnet, then it is associated with the VPC's main route table
- Possible route table targets
  - Local, IGW, a NAT device, A VGW, a peering connection, or a VPC endpoint (e.g. S3)
- Selection of the optimum route for network traffic is done based on the longest prefix match
  - the most specific routes that match the network traffic





CLOUD COMPUTING APPLICATIONS

VPC: Gateways

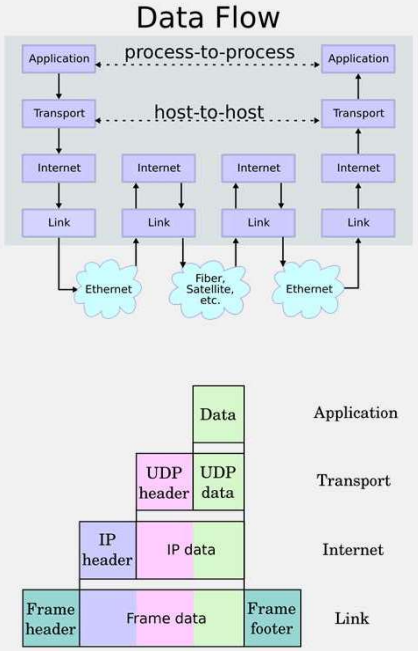
Prof. Reza Farivar

Internet Gateways

- Internet Gateway is a logical construct, not a specific instance or resource
- AWS does quite a bit of behind the scene work to allow highly available internet to all the required Availability Zones in the VPC
- Is attached to a VPC
- Highly available, redundant, and horizontally scaled
- → in the route tables, it is referred to by its name (e.g. igw-05ae7f551a8154d1a), not an IP address

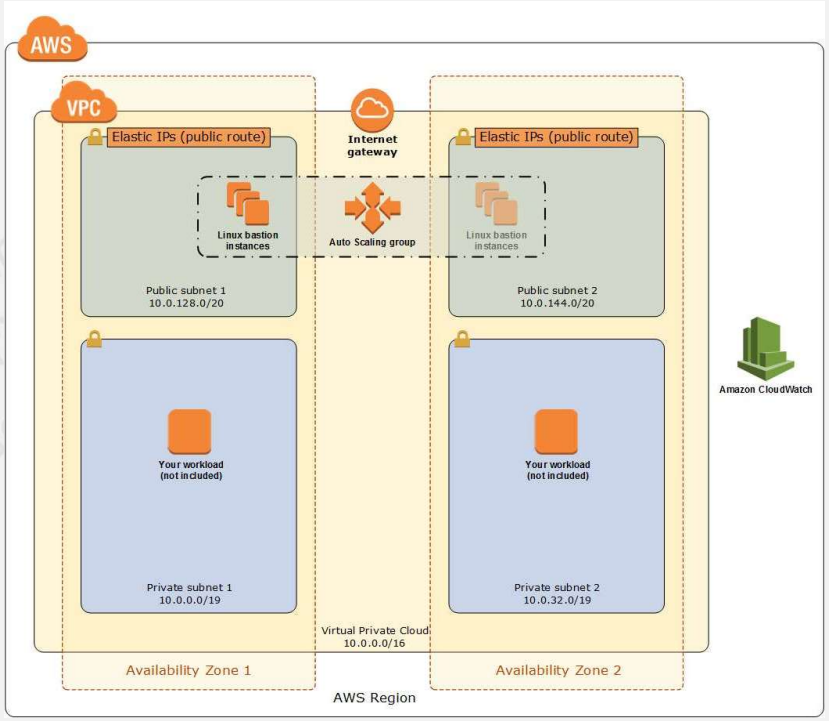
NAT Gateways

- Network Address Translation
  - Just like your home wireless router
- Virtual router or a gateway in a public subnet that enables instances in a private subnet to interact with the internet
  - IPV4 only
- Modifies the network address information in the IP header
  - It receives traffic from an EC2 instance residing in a private subnet
  - before forwarding the traffic to the internet, replaces the reply-to IPv4 address with its own public or Elastic IP address
  - When a reply is received from internet, it changes the reply-to address from its IP address to the EC2 instance private IP address
- Two types of NAT
  - NAT Instance → Runs as as EC2 instance
  - NAT Gateway → fully managed by AWS, requires elastic IP
    - Better availability and higher bandwidth

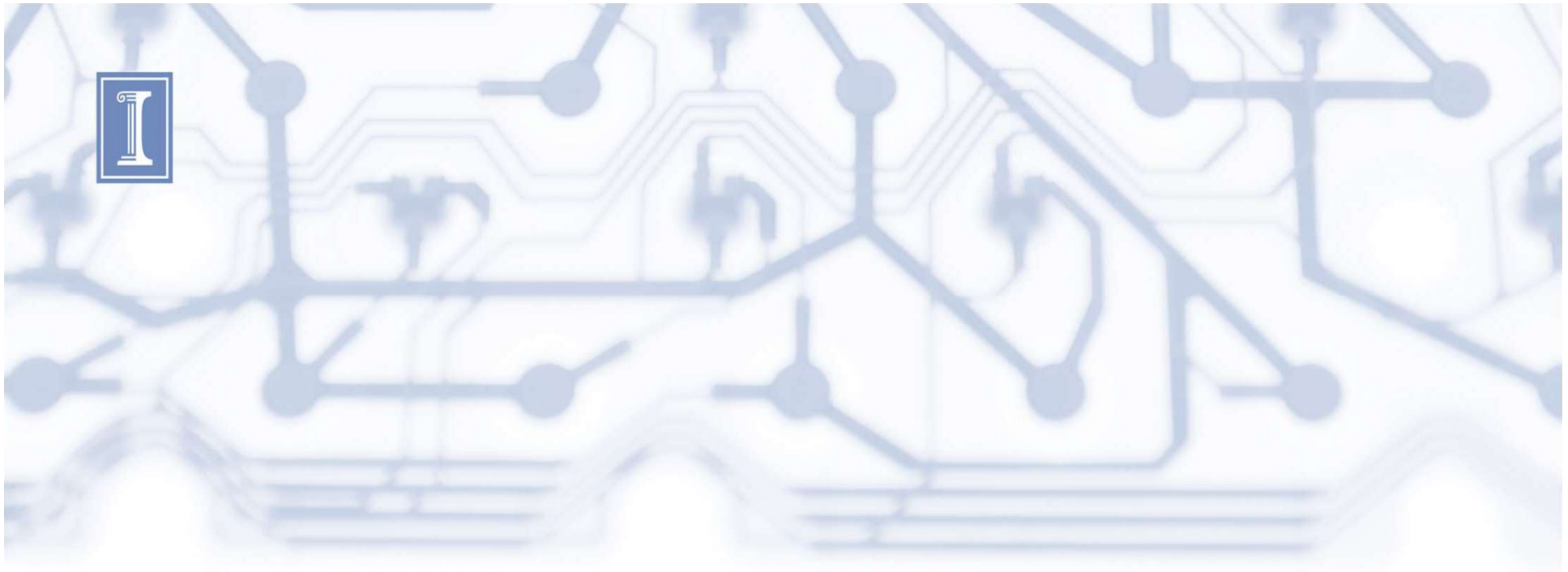


Bastion Host

- Use a bastion host to access private machines hosted in a private network in a VPC
- Bastion host: “a server whose purpose is to provide access to a private network from an external network, such as the Internet. Because of its exposure to potential attack, a bastion host must minimize the chances of penetration”



Further reading: https://cloudacademy.com/blog/aws-bastion-host-nat-instances-vpc-peering-security/



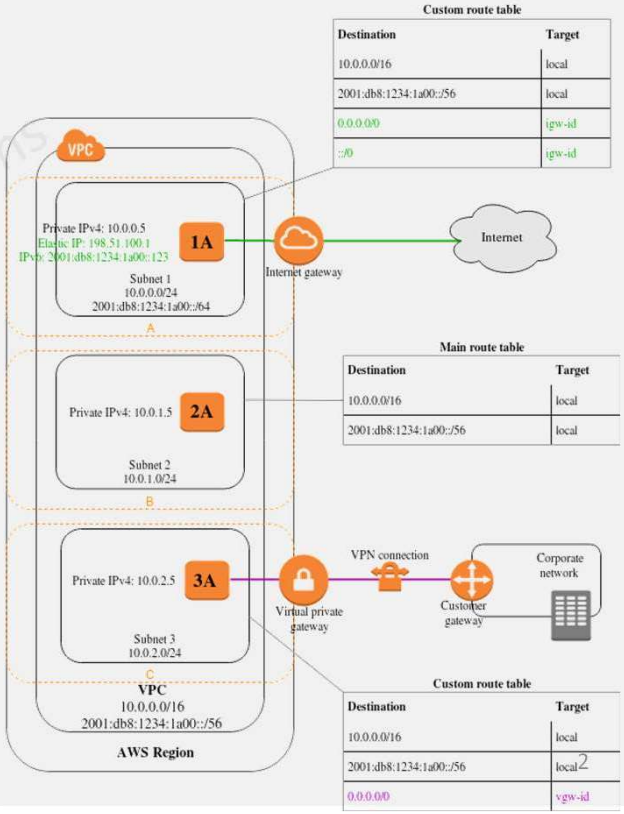
CLOUD COMPUTING APPLICATIONS

VPC: Advanced VPC

Prof. Reza Farivar

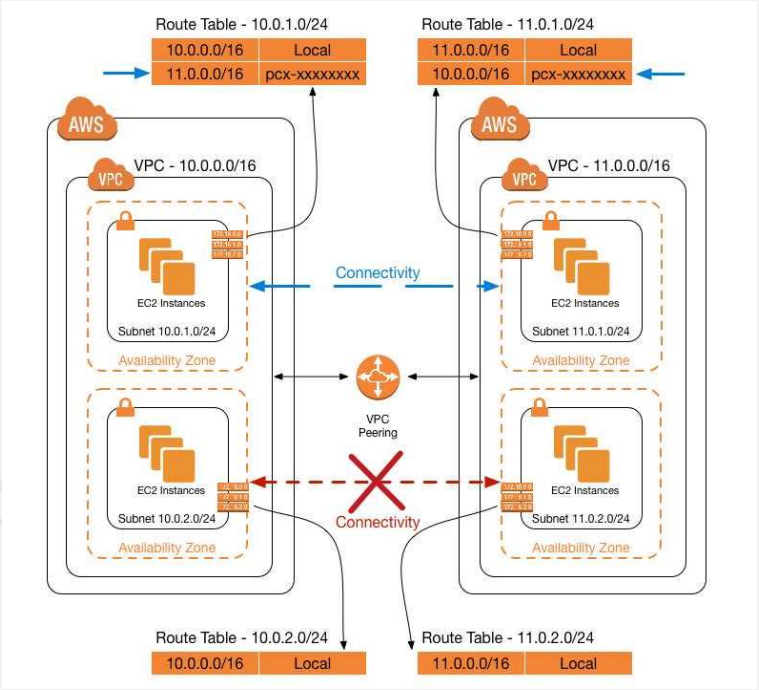
Virtual Private Gateway

- If a subnet doesn't have a route to the internet gateway, but has its traffic routed to a virtual private gateway for a Site-to-Site VPN connection, the subnet is known as a *VPN-only subnet*
- In this diagram, subnet 3 is a VPN-only subnet
- Concepts
  - **VPN connection**: A secure connection between your on-premises equipment and your VPCs.
  - **VPN tunnel**: An encrypted link where data can pass from the customer network to or from AWS.
    - Each VPN connection includes two VPN tunnels which you can simultaneously use for high availability.
  - **Customer gateway**: An AWS resource which provides information to AWS about your customer gateway device.
  - **Customer gateway device**: A physical device or software application on your side of the Site-to-Site VPN connection.
  - **Virtual private gateway**: The VPN concentrator on the Amazon side of the Site-to-Site VPN connection. You use a virtual private gateway or a transit gateway as the gateway for the Amazon side of the Site-to-Site VPN connection.
  - **Transit gateway**: A transit hub that can be used to interconnect your VPCs and on-premises networks. You use a transit gateway or virtual private gateway as the gateway for the Amazon side of the Site-to-Site VPN connection.
- As of 2020, VPN connections into AWS are IPV4 only
- It is recommended that you use non-overlapping CIDR blocks for your networks



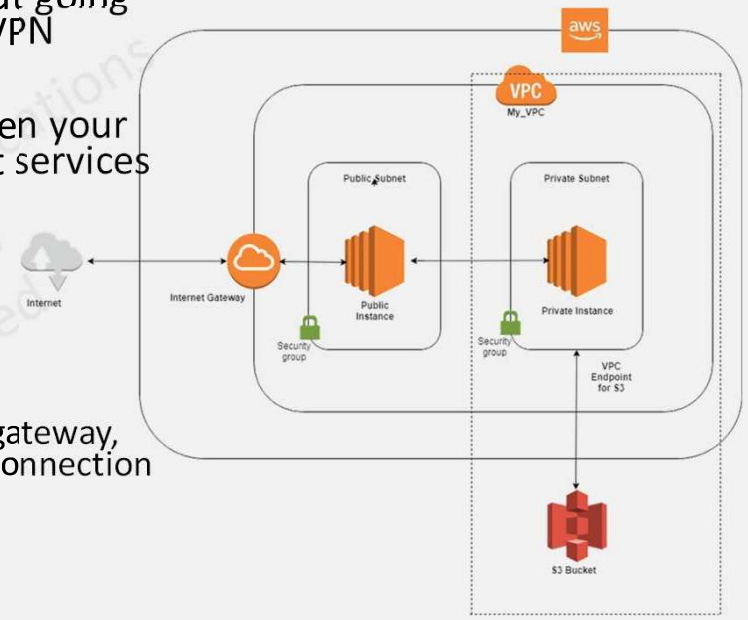
VPC Peering

- VPC peering can be used to make communication between VPCs within the same account, different AWS accounts, or any two VPCs within the same region or different regions
- Initially, VPC peering was supported only within the same region, but later AWS added support for VPC peering across regions
- The two VPCs cannot have CIDR blocks that overlap with each other.



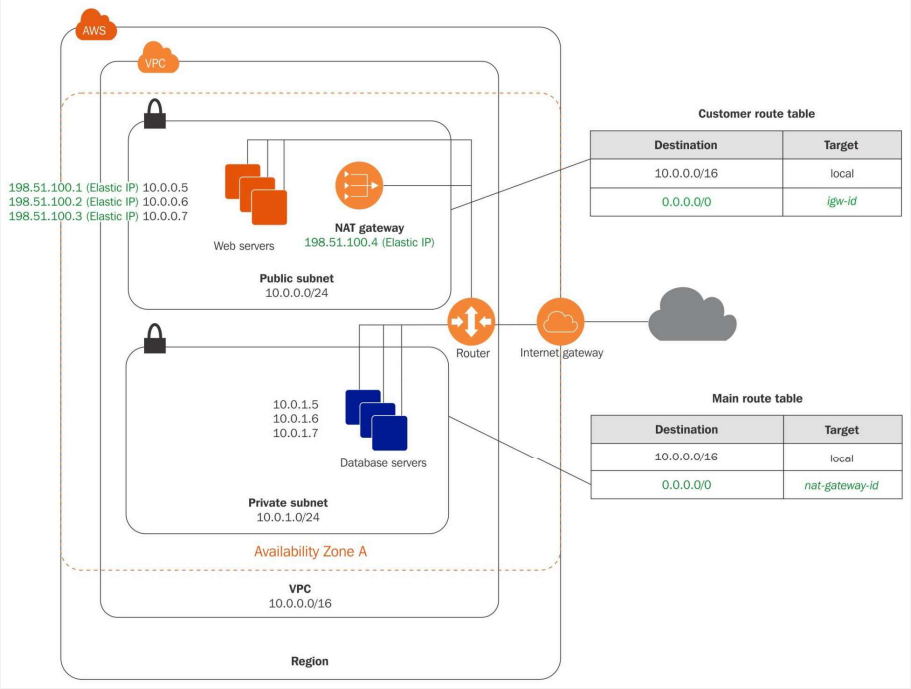
VPC Endpoints

- Generally, AWS services are different entities and do not allow direct communication with each other without going through either an IGW, a NAT gateway/instance, a VPN connection, or AWS Direct Connect
- A VPC endpoint enables private connections between your VPC and supported AWS services and VPC endpoint services
  - S3
  - DynamoDb
- AWS PrivateLink
  - Private IP addresses
  - Traffic does not leave the Amazon network
  - Does not require an internet gateway, virtual private gateway, NAT device, VPN connection, or AWS Direct Connect connection



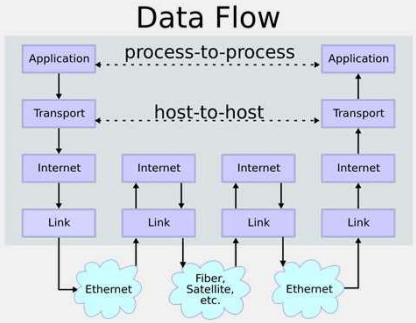
** Interesting reading: https://www.bluematador.com/blog/s3-endpoint-connectivity-in-aws-vpc*

VPC with Private and Public Subnets



Routing in VPC vs. Physical Network

- Physical Ethernet Network
  - Link Layer
    - Lowest layer in the Internet Protocol
      - Layer 2 in OSI model
  - In a physical traditional network, this layer uses MAC address and ARP messaging (to discover unknown MAC addresses)
- VPC Network
  - Amazon backend intercepts any MAC ARP request
  - Looks up routing tables, and returns the destination without implementing ARP



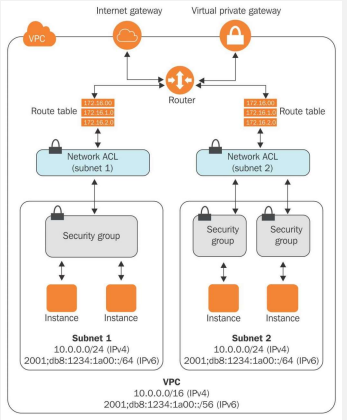
CLOUD COMPUTING APPLICATIONS

VPC: Security and Firewalls

Prof. Reza Farivar

Security and Firewalls

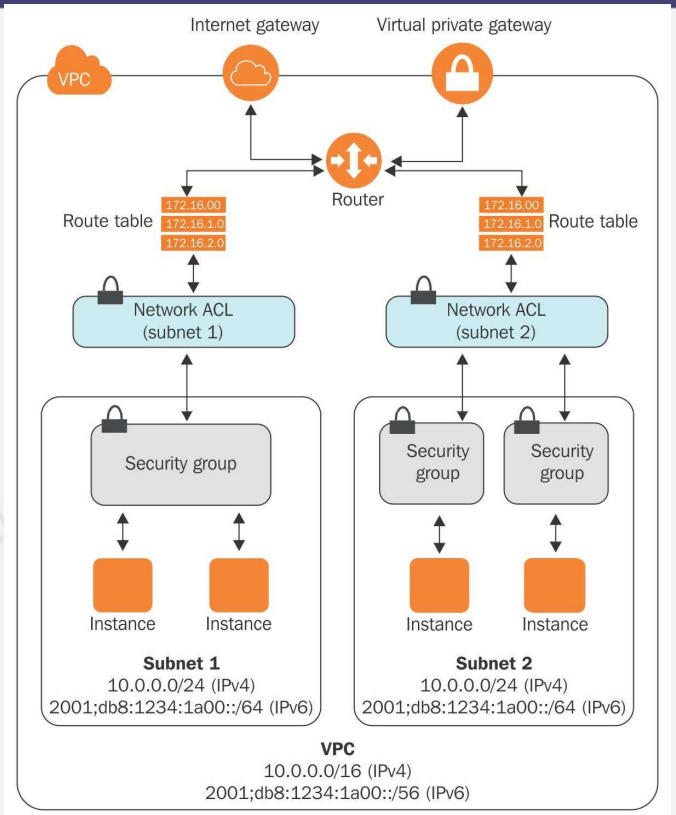
- Security
  - Security Groups
    - EC2 instance-level firewall
  - Network Access Control Lists (NACL)
    - Subnet firewall
- Monitoring
  - Flow Logs
  - Enable VPC flow logs for audit purposes
  - Study flow logs from time to time
    - highlights unauthorized attempts to access the resources



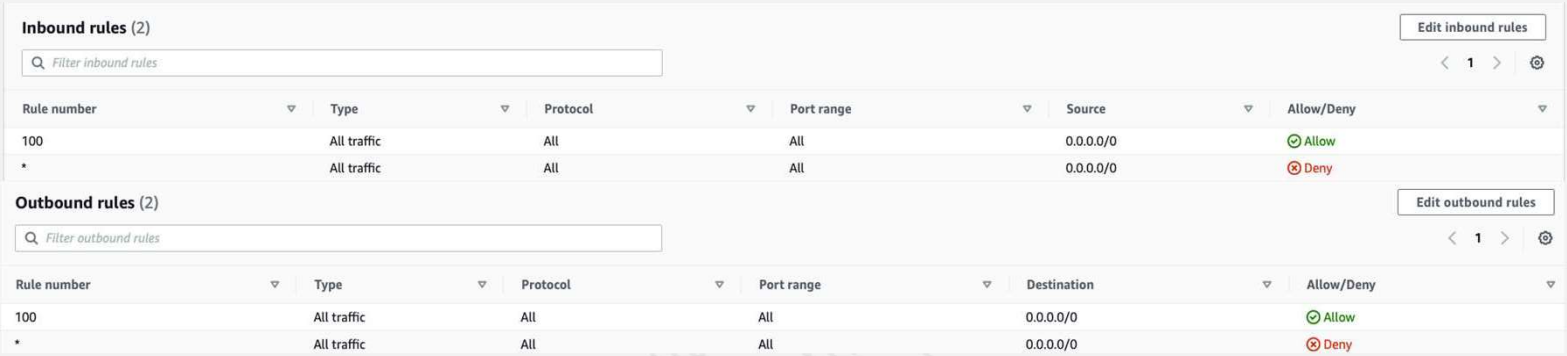
Security

- Make sure that only required ports and protocols from trusted sources can access AWS resources using security groups and NACLs
- Make sure that unwanted outgoing ports are not open in security groups
  - A security group for a web application does not need to open incoming mail server ports

Security and Firewalls



Network Access Control List



Inbound rules (2)

Edit inbound rules

Filter inbound rules

1

Rule number	Type	Protocol	Port range	Source	Allow/Deny
100	All traffic	All	All	0.0.0.0/0	Allow
*	All traffic	All	All	0.0.0.0/0	Deny

Outbound rules (2)

Edit outbound rules

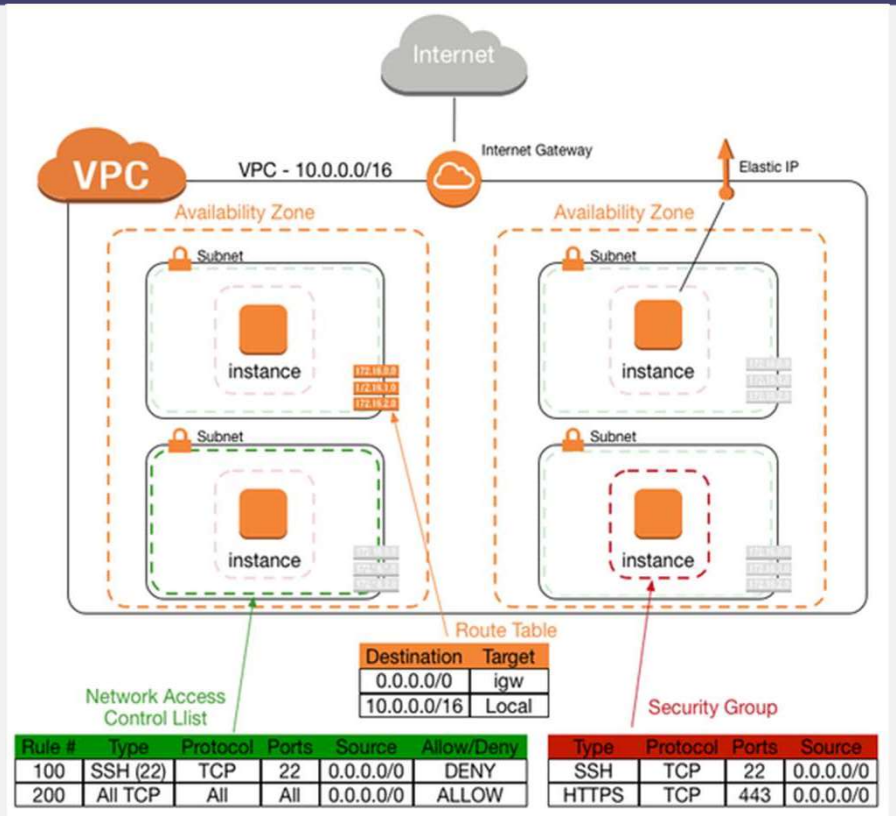
Filter outbound rules

1

Rule number	Type	Protocol	Port range	Destination	Allow/Deny
100	All traffic	All	All	0.0.0.0/0	Allow
*	All traffic	All	All	0.0.0.0/0	Deny

- NACL acts as a virtual firewall at the subnet level
- Every VPC has a default NACL
- Every subnet, whether it is private or public in a VPC, must be associated to one NACL
- One NACL can be associated with one or more subnets; but each subnet can have ONE NACL associated with it
- NACL rules are evaluated based on its rule numbers. It evaluates the rule starting from the lowest number to the highest number
- NACL is stateless:
  - Separate rules to allow or deny can be created for inbound and outbound traffic
  - If a port is open for allowing inbound traffic, it does not automatically allow outbound traffic
- The default NACL for any VPC contains a rule numbered as * in both inbound and outbound rules
  - This rule appears and executes last

Anatomy of a VPC with Route Table, Network ACL and Security Group



Security Group

Inbound rules — Edit inbound rules

Type	Protocol	Port range	Source	Description - optional
HTTP	TCP	80	0.0.0.0/0	-
HTTP	TCP	80	::/0	-
Custom TCP	TCP	8080	0.0.0.0/0	-
Custom TCP	TCP	8080	::/0	-
SSH	TCP	22	0.0.0.0/0	-
SSH	TCP	22	::/0	-
HTTPS	TCP	443	0.0.0.0/0	-
HTTPS	TCP	443	::/0	-

Outbound rules — Edit outbound rules

Type	Protocol	Port range	Destination	Description - optional
All traffic	All	All	0.0.0.0/0	-

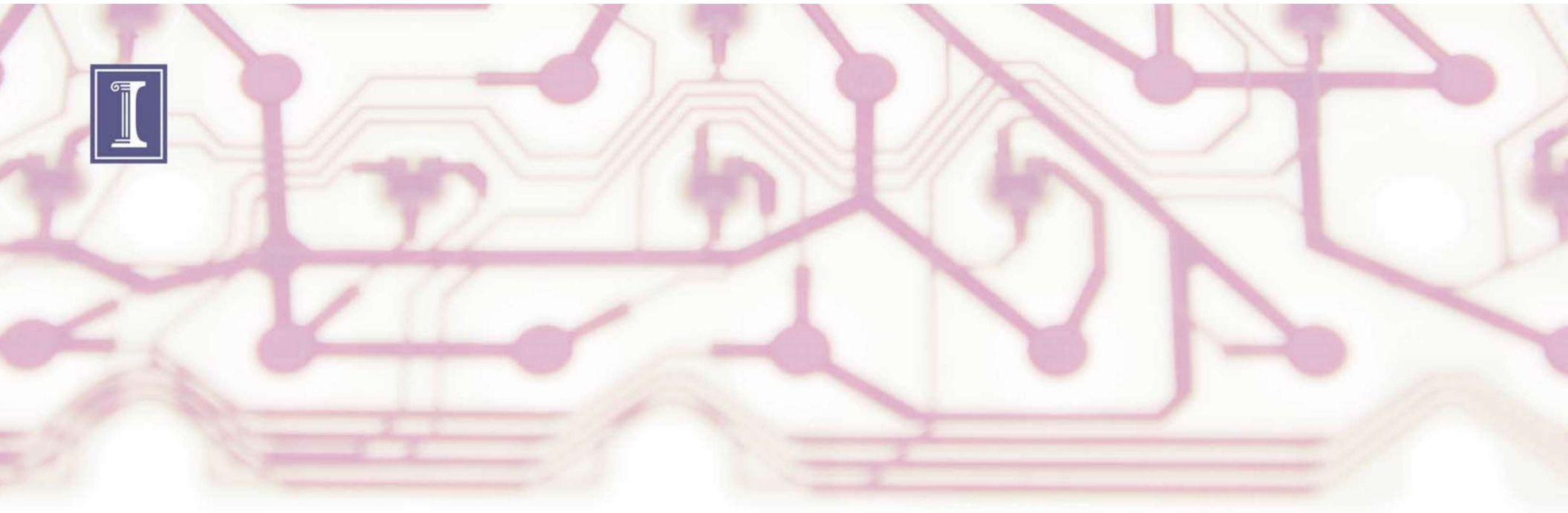
- Firewall at the instance level
- One or more security groups can be associated with each EC2 instance
- A security group can be attached to many EC2 instances
- Each SG contains rules allowing inbound and outbound traffic
- Using CIDR notation, a source IP can be fixed to a particular IP, such as 10.108.20.107/32
- Any source IP can be allowed by a 0.0.0.0/0

Security Group as Source IP

Inbound rules Edit inbound rules

Type	Protocol	Port range	Source	Description - optional
Custom TCP	TCP	4003 - 65535	sg-00fbbeba74e062d16 (nodes.dev.k8s.mp3-k8.in)	-
Custom TCP	TCP	2382 - 4000	sg-00fbbeba74e062d16 (nodes.dev.k8s.mp3-k8.in)	-
All traffic	All	All	sg-003e7c9121913dca3 (masters.dev.k8s.mp3-k8.in)	-
SSH	TCP	22	0.0.0.0/0	-
Custom UDP	UDP	1 - 65535	sg-00fbbeba74e062d16 (nodes.dev.k8s.mp3-k8.in)	-
Custom TCP	TCP	1 - 2379	sg-00fbbeba74e062d16 (nodes.dev.k8s.mp3-k8.in)	-
HTTPS	TCP	443	0.0.0.0/0	-

- A security group ID can be specified as a source IP to allow communication from all the instances that are attached to that security group
- For example, in the case of autoscaling, the number of EC2 instances and their IP addresses keeps changing.
- In such situations, it is best practice to attach a security group to such EC2 instances with the help of an autoscaling template and place a security group ID as a source IP in another security group.



CLOUD COMPUTING APPLICATIONS

SOFTWARE DEFINED ARCHITECTURE

Roy Campbell & Reza Farivar

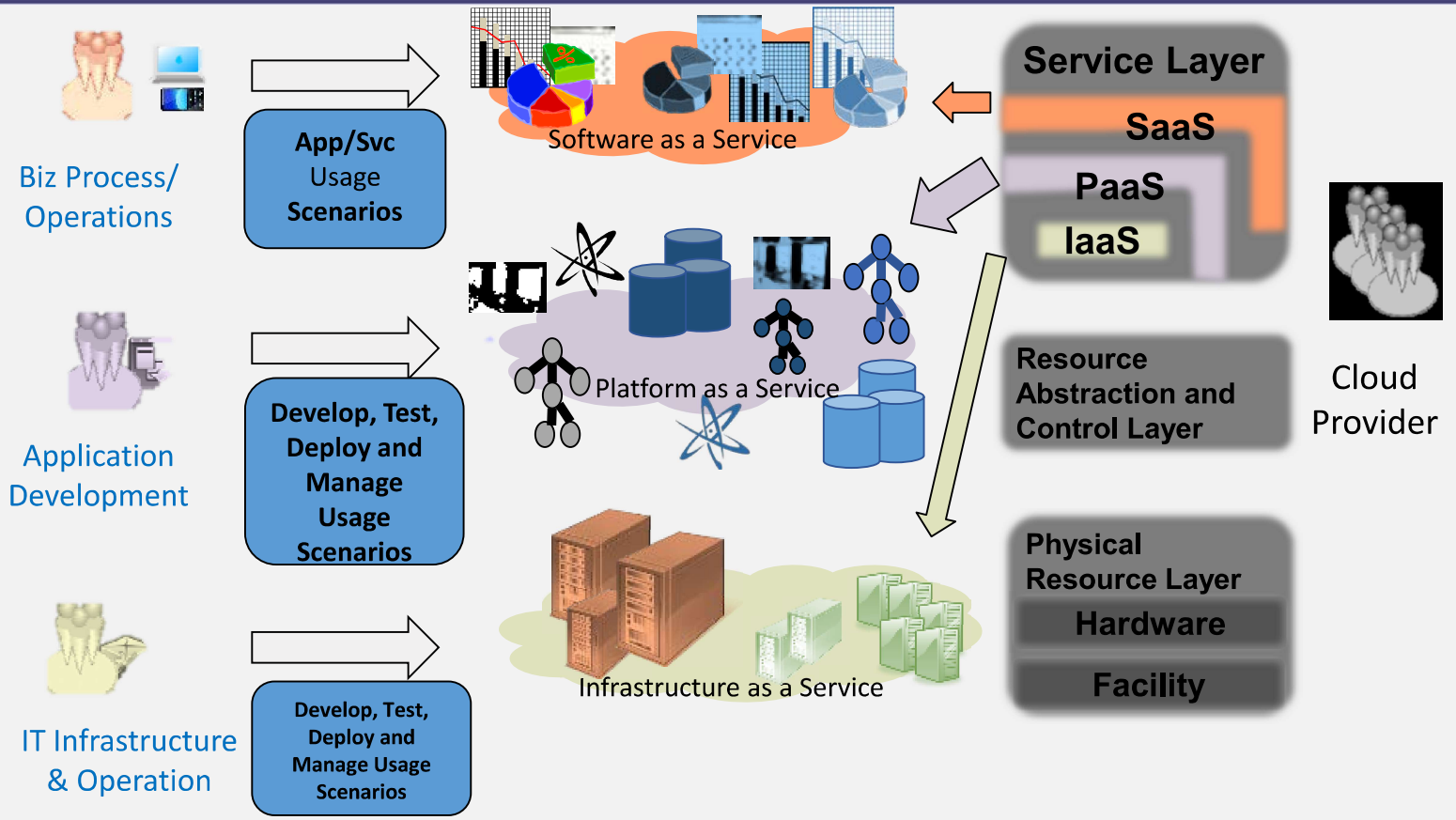
Learning Objectives

- How services are created
- How services can control other services
- The principal architectural components of a cloud and their organization
- How services and orchestration play a role in each layer of a production cloud: IAAS, PAAS, SAAS

Software Defined Architecture

- Cloud provides services, service orchestration, and provisioning
- A Cloud may provide IAAS, PAAS, SAAS and have both internal and external Application Programming Interfaces
- The mechanisms and concept of providing services, orchestration, and provisioning is called a Software Defined Architecture
- A Cloud may contain other software defined entities:
  - Software Defined Network
  - Software Defined Storage
  - Software Defined Compute

Cloud Provider: Service Orchestration

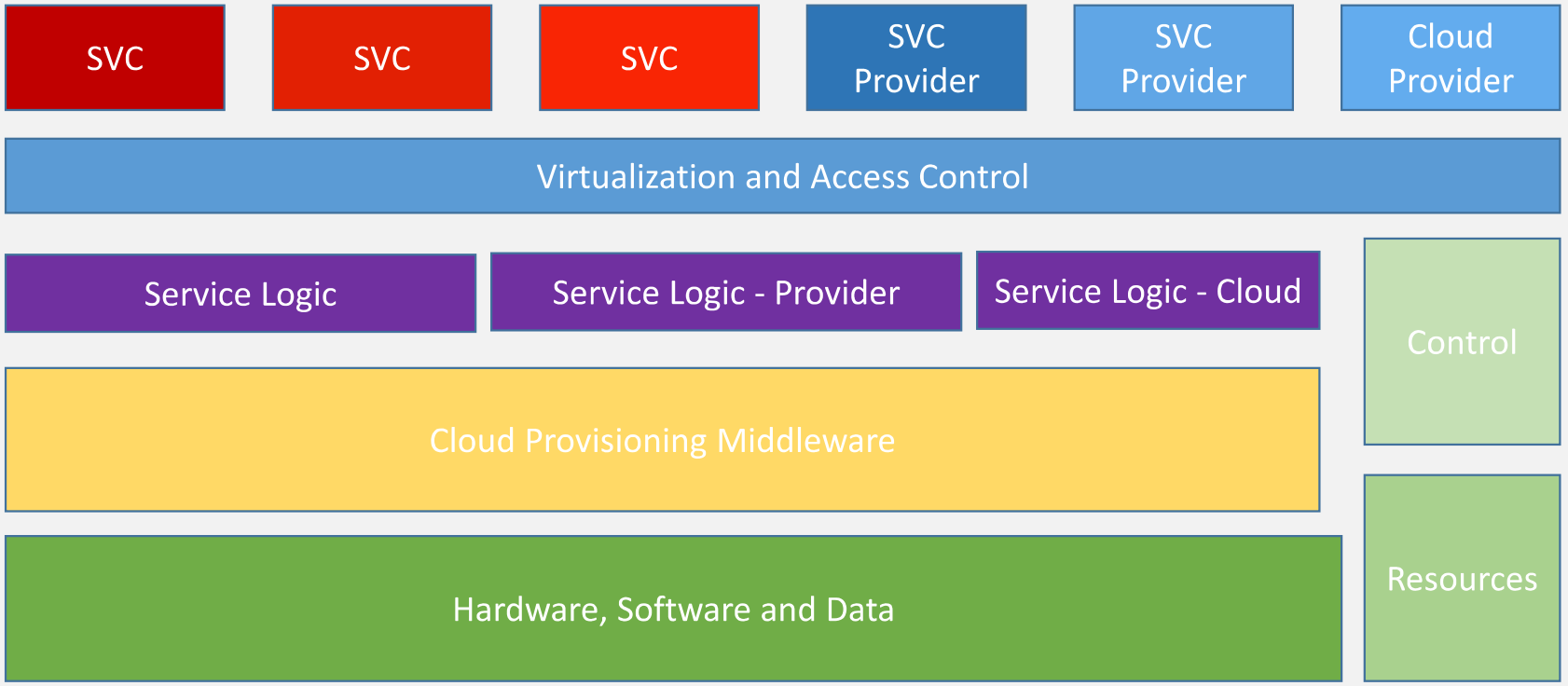


Orchestration

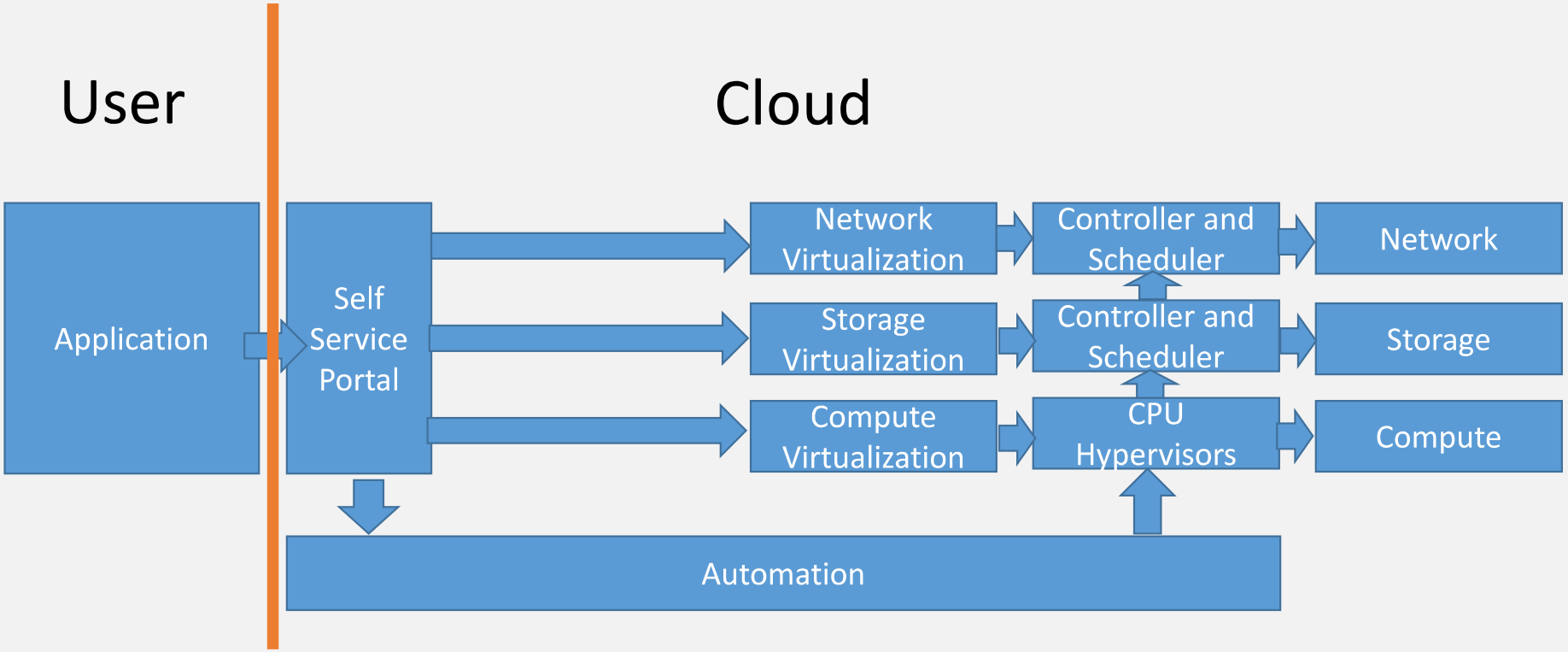
Cloud service orchestration is the:

- *Composing* of architecture, tools and processes used by humans to deliver a defined Service.
- *Stitching* of software and hardware components together to deliver a defined Service.
- *Connecting* and *Automating* of work flows when applicable to deliver a defined Service.
- Provides: up and down scaling, assurance, billing, workflows

Software Defined Architecture



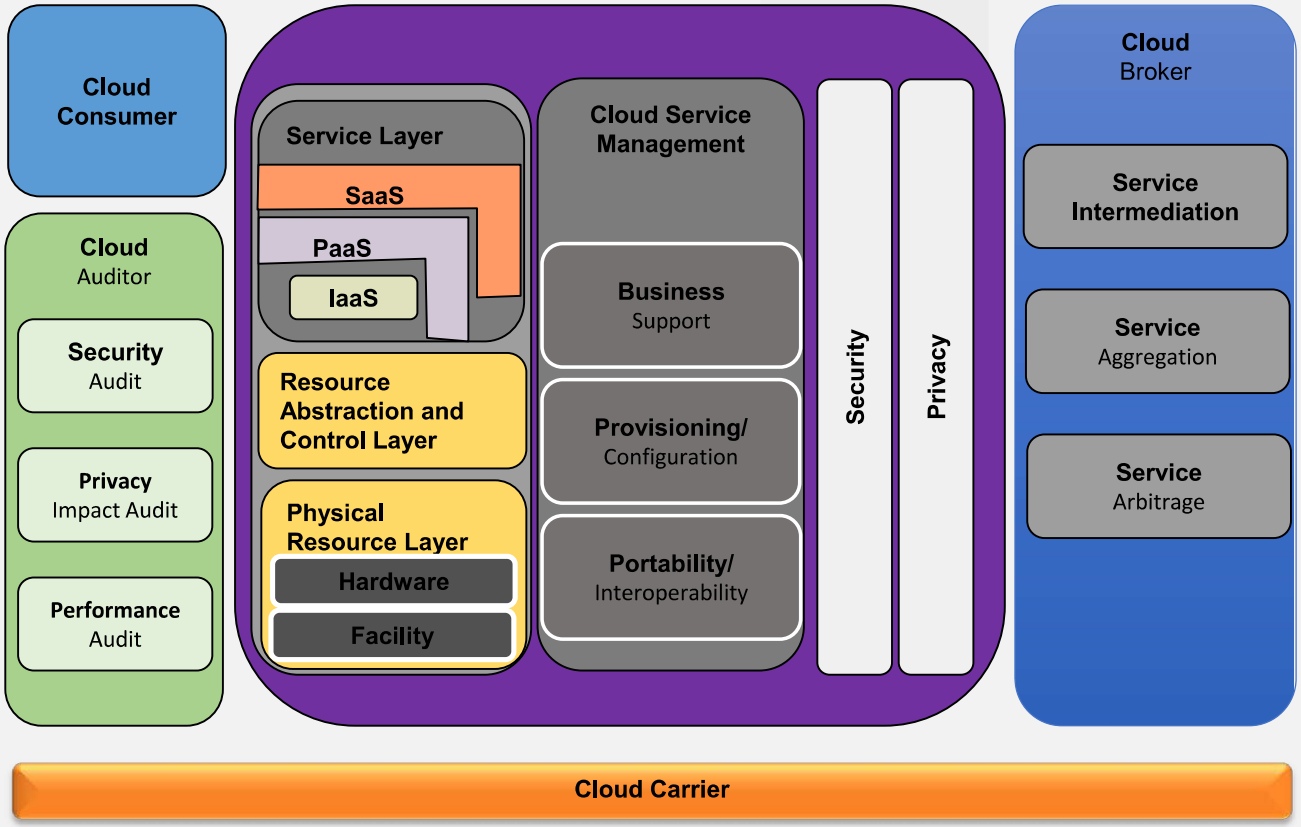
Software Defined Data Center

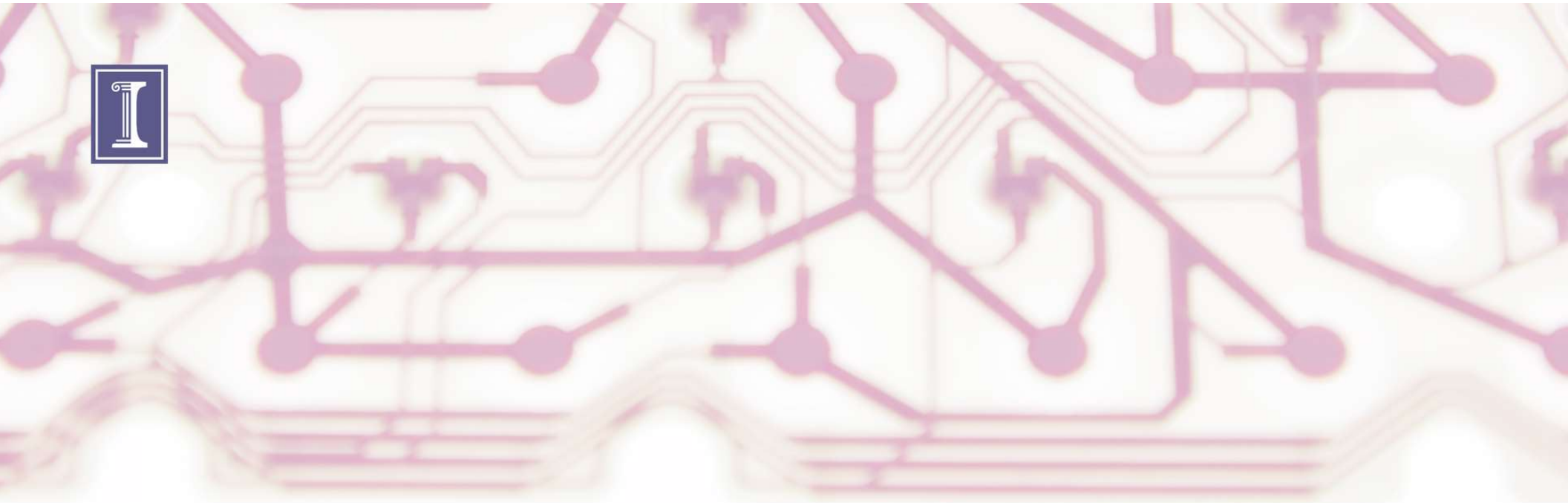


Content and Learning Objectives

1) Virtualization is a key abstraction in building software defined architectures:
    1) Software Defined Networks
    2) Software Defined Storage
    3) Software Defined Compute
2) Web Service: A Simple Application built on a Data Center
3) Load Balancing: A simple scheme to distribute the load of multiple servers
4) Infrastructure as a Service
5) Mirantis and OpenStack
6) How systems are structured with orchestration

The Combined Conceptual Reference Diagram





CLOUD COMPUTING APPLICATIONS

Cloud Services

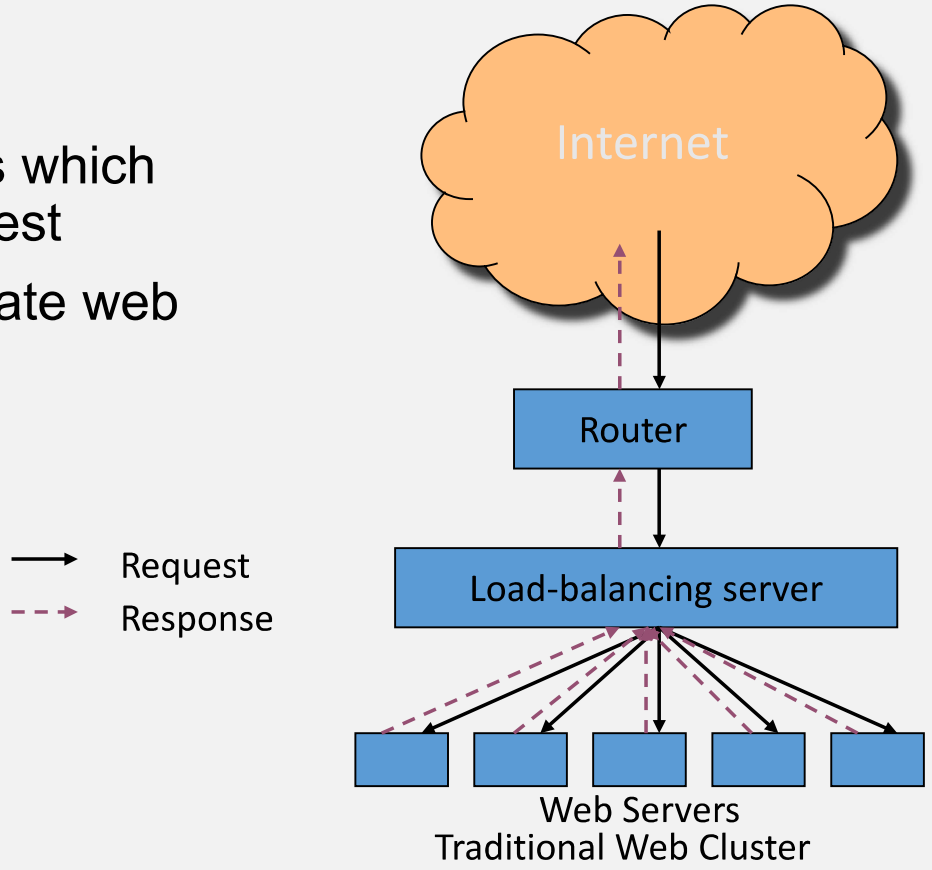
Prof. Roy Campbell

Contents

- Web Services
- Remote Procedure Calls
  - RMI, SOAP
- HTTP, REST
- JSON, XML
- Load Balancing

Introduction to Web Servers and Load Balancing

- Request enters a router
- Load balancing server determines which web server should serve the request
- Sends the request to the appropriate web server



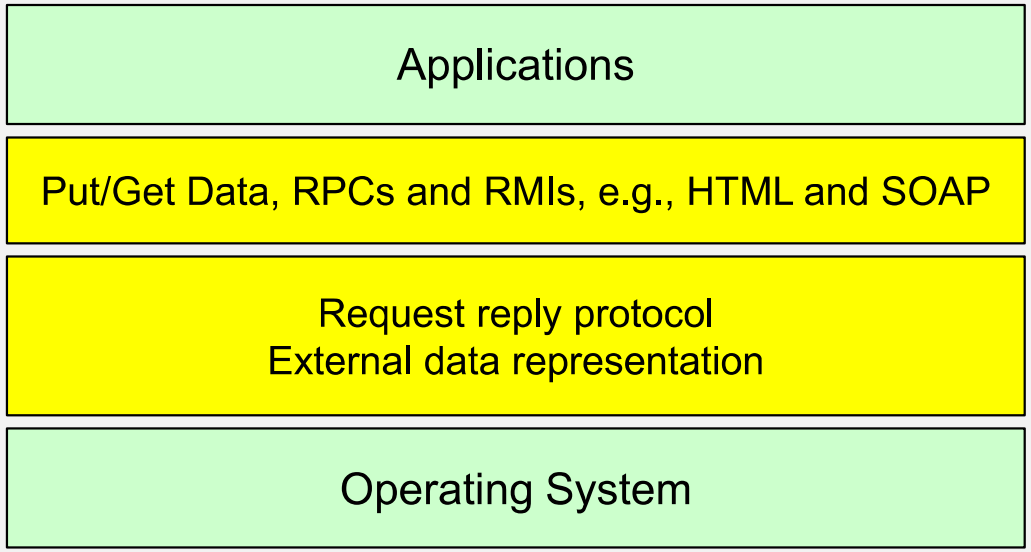
Middleware Layer Definition

- *Software that provides services to applications beyond those generally available at the operating system*

- Middleware implements functionalities that are common across many different applications
  - No need to reinvent the wheel (e.g., message parsing) every time you need to do something

- Building distributed systems while maintaining our code is not very different from a single-node program

Middleware Layers



Middleware layers

Provide support to the application

Run at all servers at user level

RMI = Remote Method Invocation
CORBA = Common Object Request Brokerage Architecture
SOAP = Simple Object Access Protocol

SOAP — Simple Object Access Protocol

- Transmitted by HTTP or SMTP (or many others)
- Coded in XML (Can be decoded on any machine)
- Return value: Any XML document
- Underlies Web Services Description Language (WSDL)

Common Abstractions

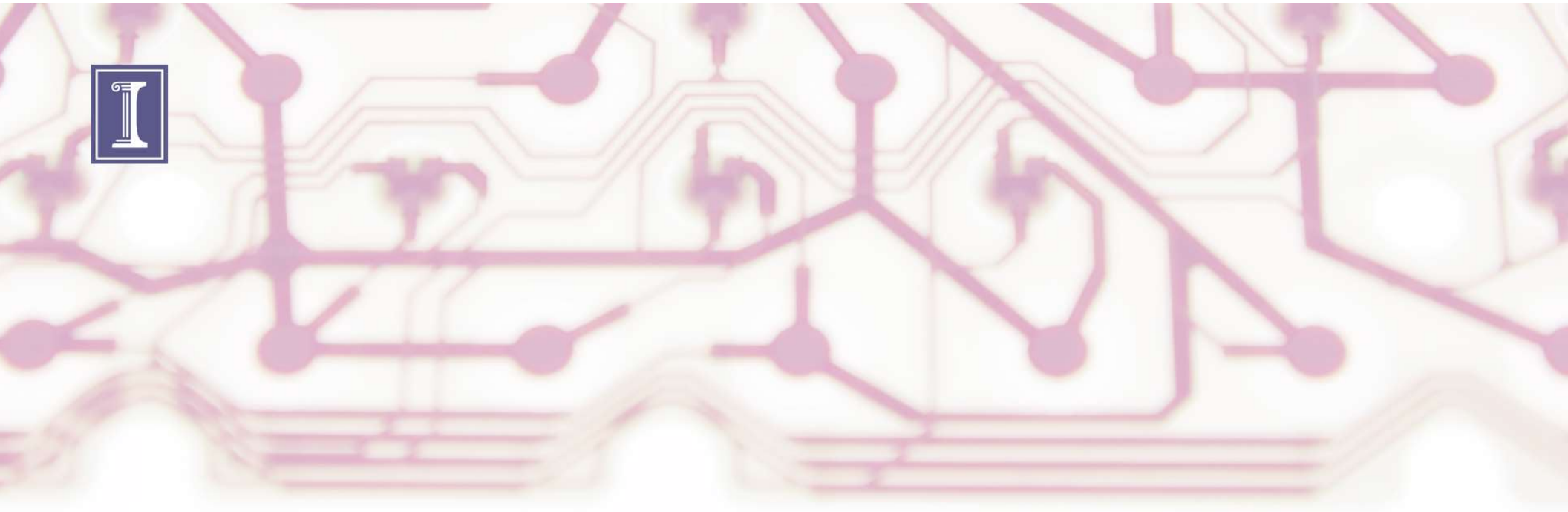
- In single-node programs, we often rely on concepts such as
  - Procedure calls
  - Objects
  - Shared memory
  - …
- A Middleware Layer can provide the same abstractions to distributed applications!
- We'll start with objects and procedure calls

Local Objects

- Within one process' address space
- Object
  - Consists of a set of data and a set of methods
  - For example, C++ object, Chord object at a client (Chord data structures + functions at a node)
- Object reference
  - An identifier by which objects can be accessed, i.e., a *pointer*
- Interface
  - Provides a definition of the signatures of a set of methods (i.e., the types of their arguments, return values, and exceptions) without specifying their implementation
  - For example, {put(objectname), get(objectname)} interface for Chord object. Same interface also applies to other P2P objects such as Gnutella, Kazaa, etc.

Remote Objects

- May cross multiple process' address spaces
- Remote method invocation
  - Method invocations between objects *in different processes (processes may be on the same or different host)*
  - Remote Procedure Call (RPC): procedure call between functions on different processes in non-object-based system
- Remote objects
  - Objects that can receive remote invocations
- Remote object reference
  - An identifier that can be used globally *throughout a distributed system* to refer to a particular unique remote object
- Remote interface
  - Every remote object has a remote interface that specifies which of its methods can be invoked remotely, e.g., CORBA interface definition language (IDL)



CLOUD COMPUTING APPLICATIONS

RPC IMPLEMENTATION

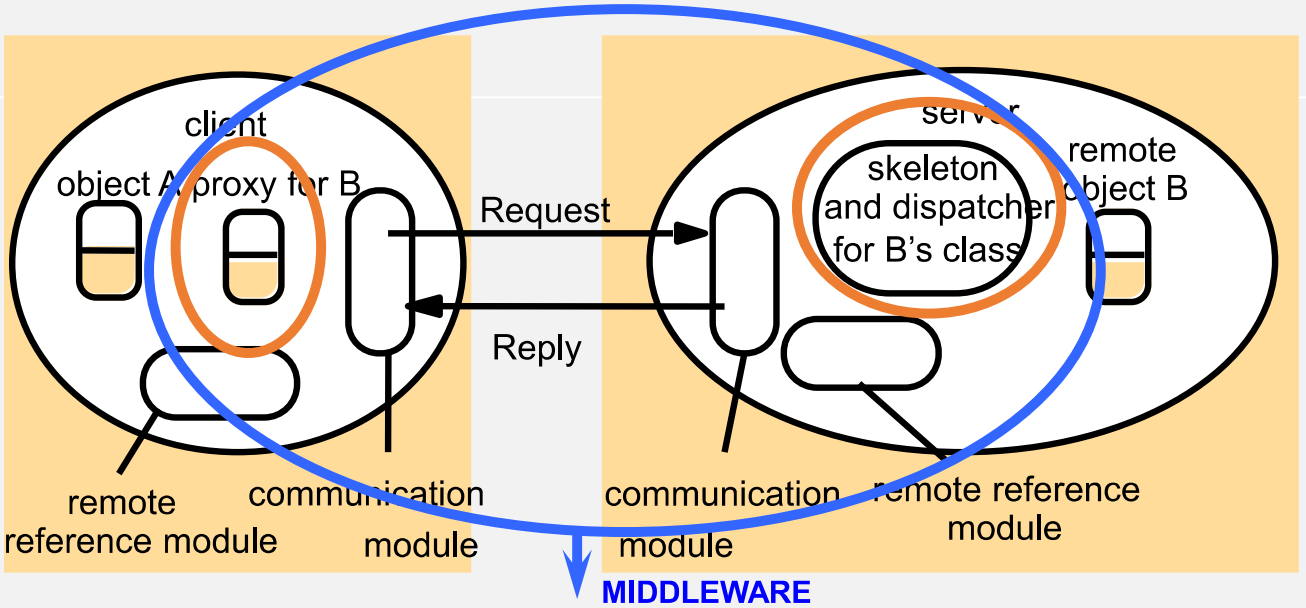
Prof. Roy Campbell

Contents

- Remote Procedure Calls
  - RMI, SOAP

How Do We Implement the Abstractions?

What should the middleware do to make the call appear similar to a local call?



- **Proxy on the "client" (process P1) side**
- **Skeleton and dispatcher on the "server" (process P2) side**

Proxy

- Is responsible for making RMI transparent to clients by behaving like a local object to the invoker
  - The proxy *implements* (Java term, not literally) the methods in the interface of the remote object that it represents. But…
- Instead of executing an invocation, the proxy forwards it to a remote object
  - On invocation, a method of the proxy *marshals* the following into a request message: (i) a reference to the target object, (ii) its own method id and (iii) the argument values. Request message is sent to the target, then proxy awaits the reply message, *unmarshals* it, and returns the results to the invoker
  - Invoked object unmarshals arguments from request message, and when done, marshals return values into reply message

Marshalling / Unmarshalling

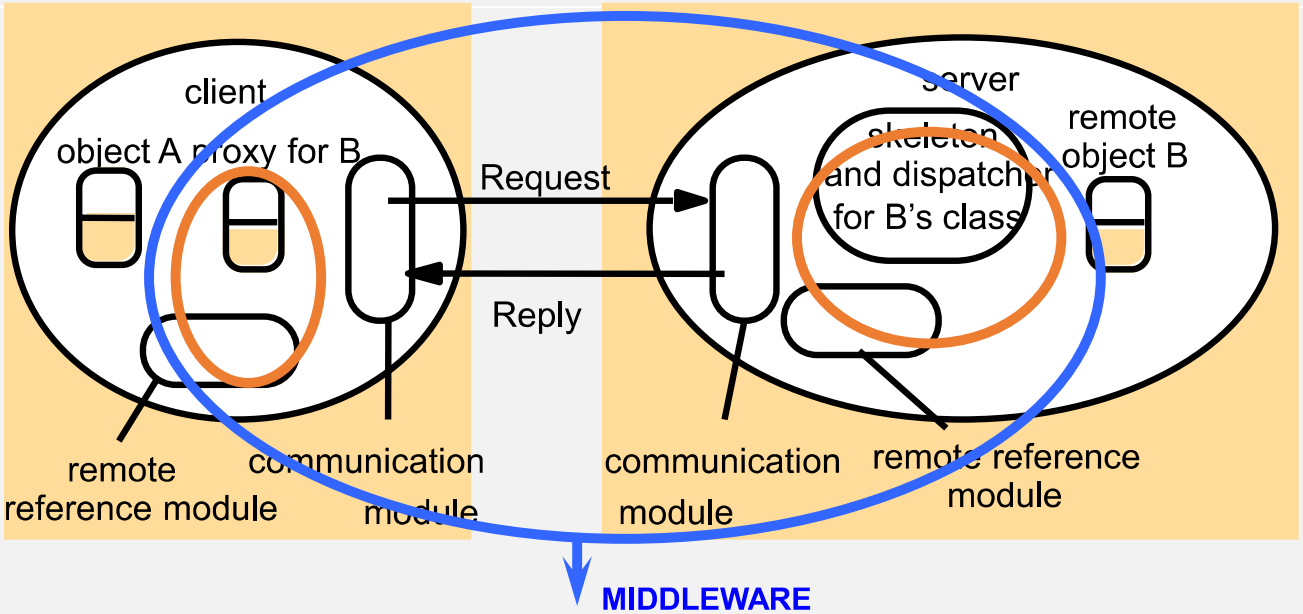
- External data representation: an agreed, platform-independent, standard for the representation of data structures and primitive values
  - CORBA Common Data Representation (CDR)
  - Allows an ARM client (possibly big endian) to interact with a x86 Unix server (little endian).
- Marshalling: the act of taking a collection of data items (platform dependent) and assembling them into the external data representation (platform independent)
- Unmarshalling: the process of disassembling data that is in external data representation form into a locally interpretable form

Remote Reference Module

- Is responsible for translating between local and remote object references and for creating remote object references
- Has a *remote object table*
  - An entry for each remote object held by any process (e.g., B at P2)
  - An entry for each local proxy (e.g., proxy-B at P1)
- When the remote reference module sees a new remote object, it creates a remote object reference and adds it to the table
- When a remote object reference arrives in a request or reply message, the remote reference module is asked for the corresponding local object reference, which may refer to either a proxy or to a remote object
- In case the remote object reference is not in the table, the RMI software creates a new proxy and asks the remote reference module to add it to the table

How Do We Implement the Abstractions?

What should the middleware do to make the call appear similar to a local call?

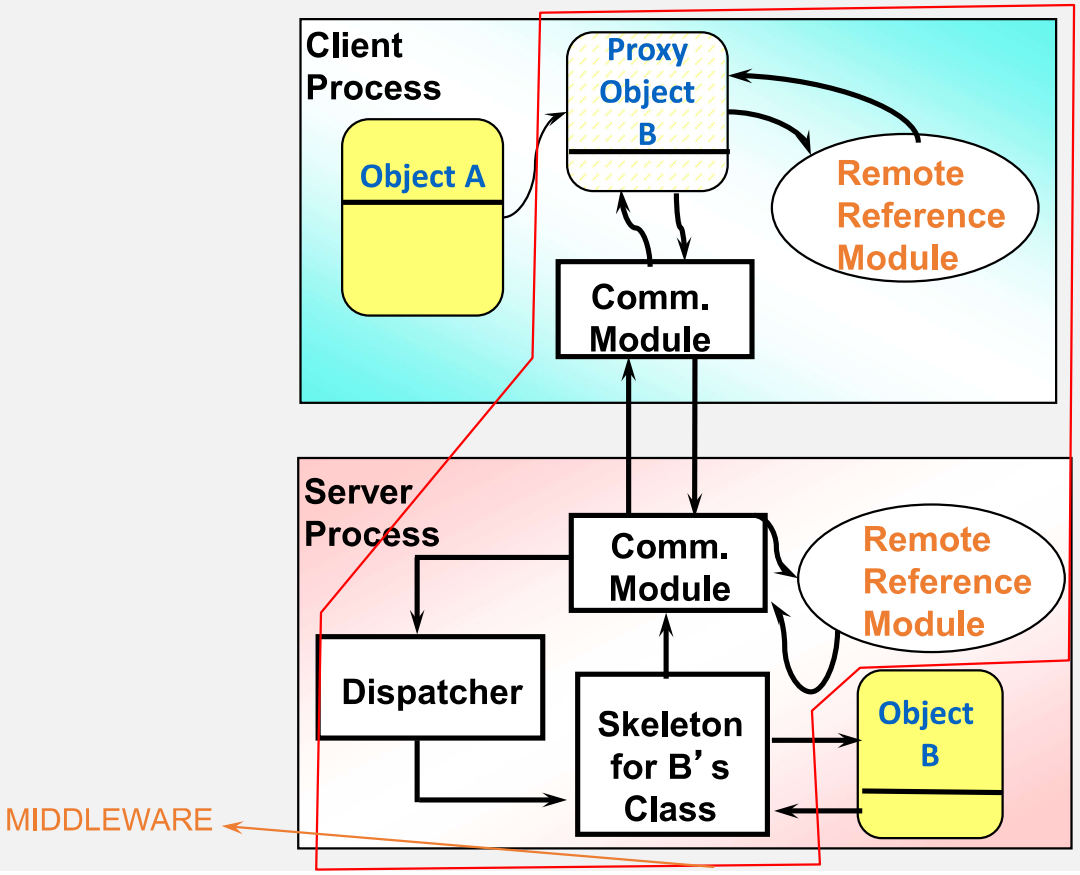


- **Proxy on the "client" (process P1) side**
- **Skeleton and dispatcher on the "server" (process P2) side**

What About Server Side? Dispatcher and Skeleton

- Each process has one dispatcher and a skeleton for each local object (actually, for the class)
- The dispatcher receives all request messages from the communication module
  - For the request message, it uses the method id to select the appropriate method in the appropriate skeleton, passing on the request message
- Skeleton “implements” the methods in the remote interface
  - A skeleton method unmarshals the arguments in the request message and invokes the corresponding method in the remote object (the actual object)
  - It waits for the invocation to complete and marshals the result, together with any exceptions, into a reply message

Summary of Remote Method Invocation (RMI)

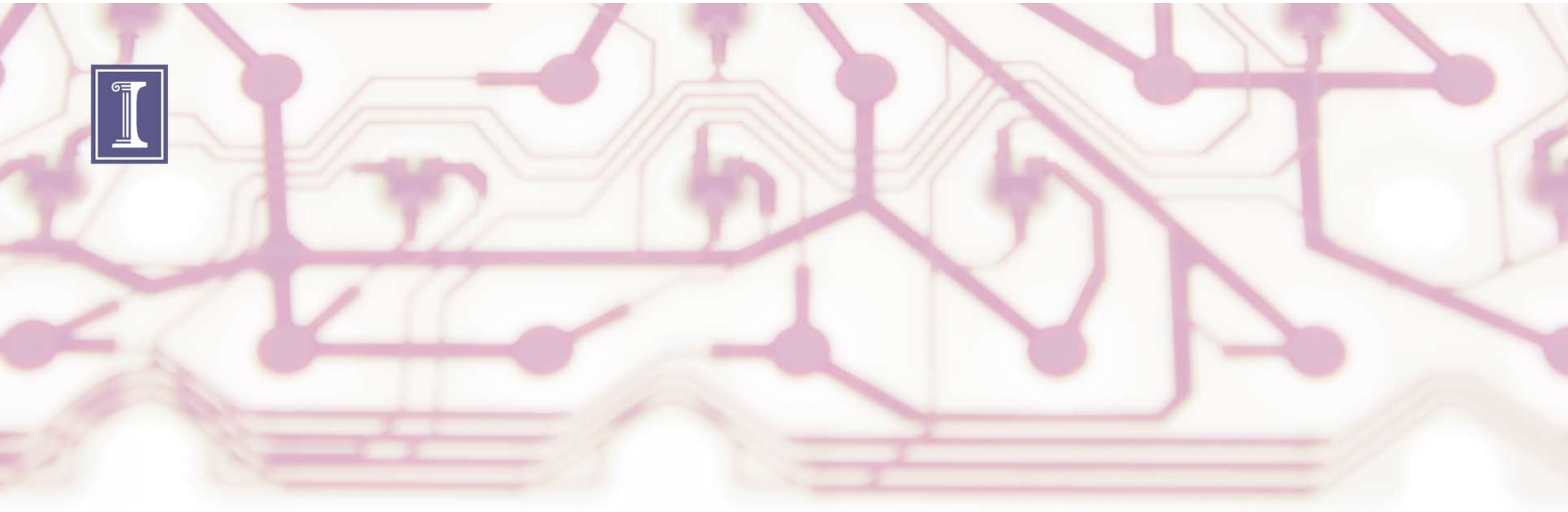


Proxy object is a hollow container of method names

Remote Reference Module translates between local and remote object references

Dispatcher sends the request to Skeleton Object

Skeleton unmarshals parameters, sends it to the object, and marshals the results for return



CLOUD COMPUTING APPLICATIONS

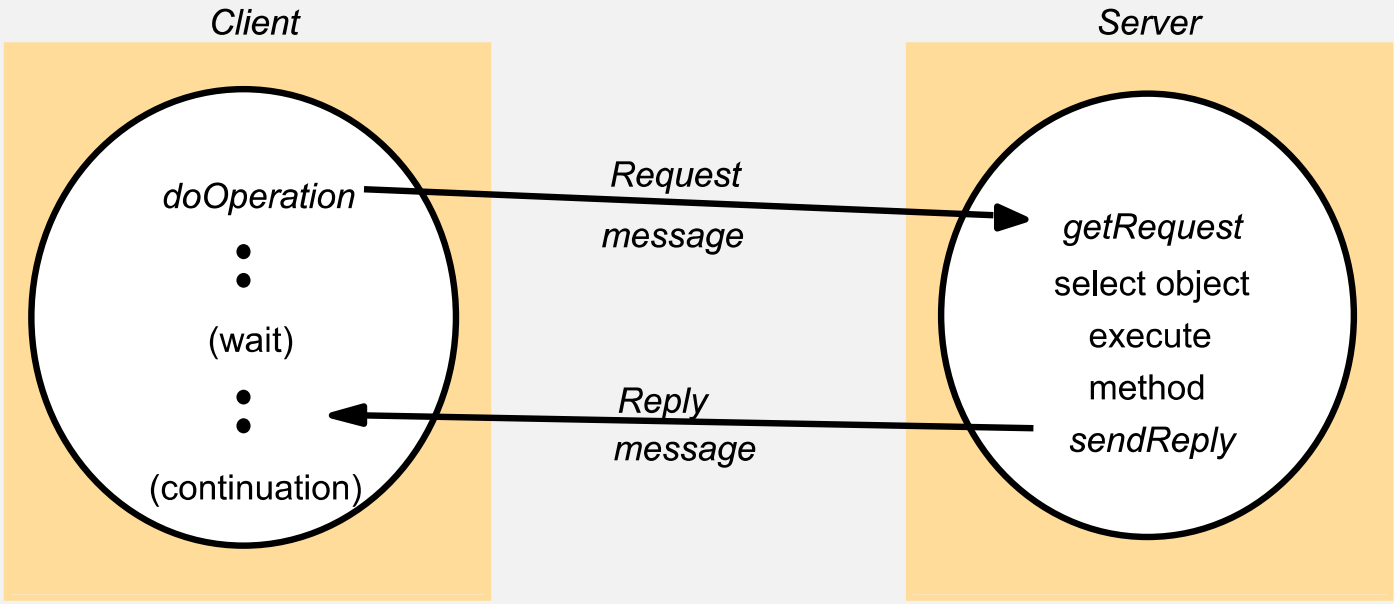
RPC Semantics

Prof. Roy Campbell

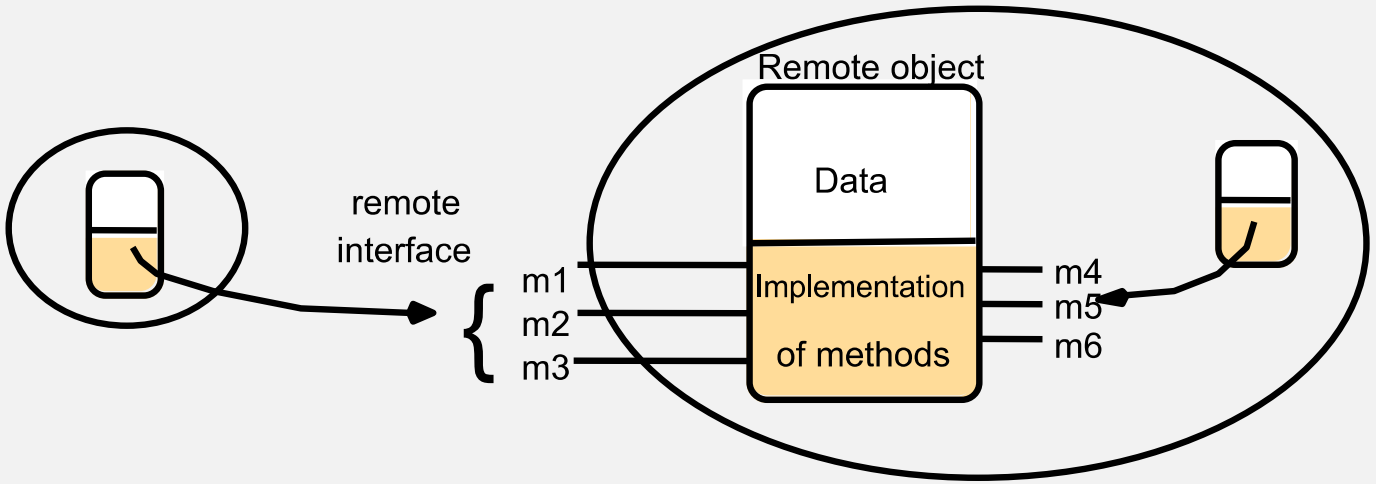
Contents

- Remote Procedure Call Semantics
  - RMI, SOAP

Request–Reply Communication

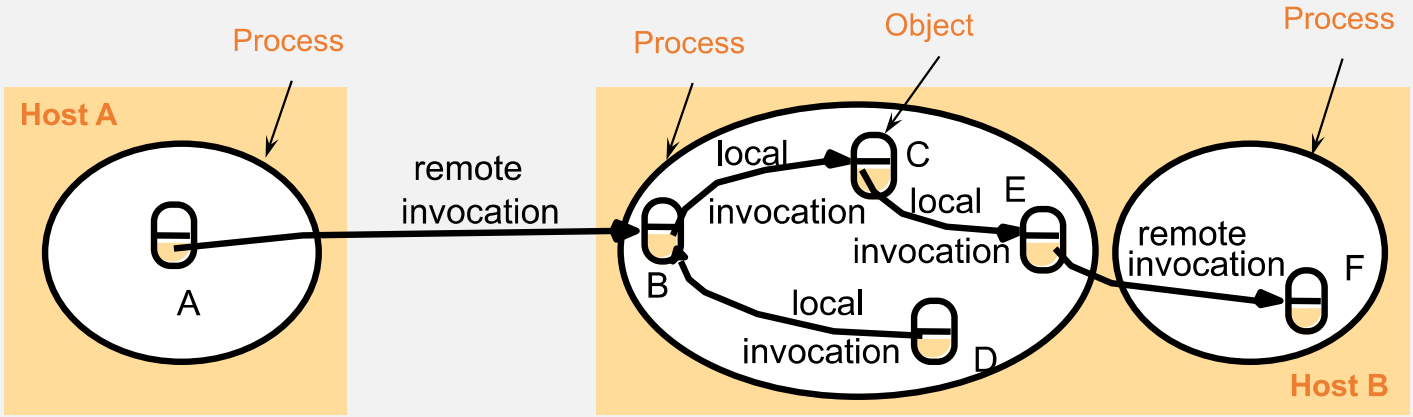


A Remote Object and Its Remote Interface



Example remote object reference = (IP, port, object number, signature, time)

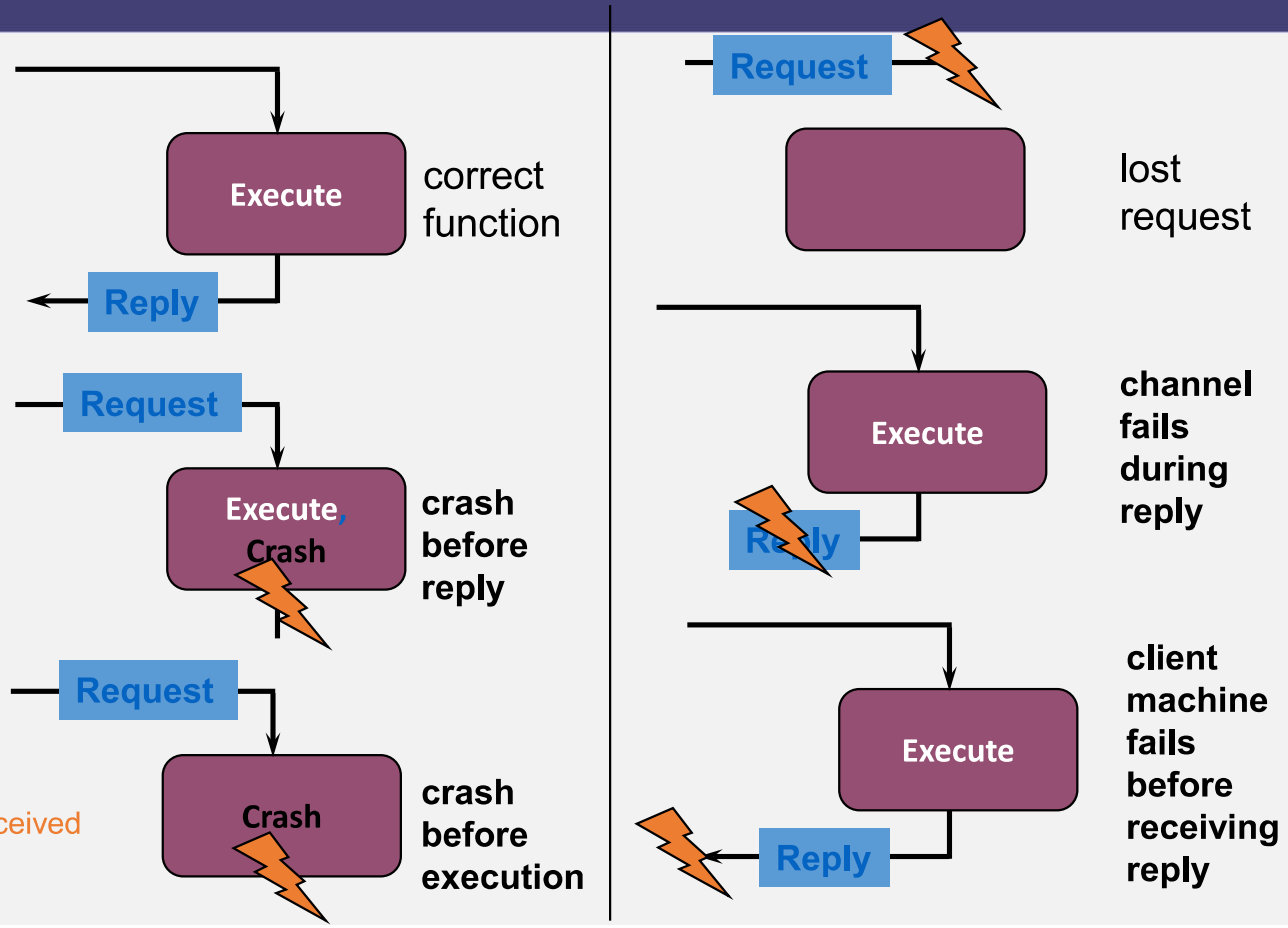
Remote and Local Method Invocations



Local invocation = between objects on same process; has *exactly once* semantics

Remote invocation = between objects on different processes. Ideally, you'd also want *exactly once* semantics for remote invocations, but it's difficult in distributed systems (why?)

Failure Modes of RMI RPC



(and if request is received more than once?)

Invocation Semantics

- Middleware aims at providing transparency
  - Sometimes obtaining transparency is expensive and unnecessary
  - Because of faults, obtaining “exactly once” semantics is expensive
- Other invocation semantics are possible
  - Maybe: Caller cannot determine whether the remote method has been executed
  - At-least-once: Caller either receives a result (in which case the user knows the method was executed at least once) or an exception
  - At-most-once: Caller either receives a result (in which case the user knows the method was executed at most once) or an exception
- The right invocation semantic depends on the application
  - For example, at-least-once semantic + idempotent operations can provide a semantic that would be equivalent to exactly-once applications

Idempotent = same result if applied repeatedly, w/o side effects

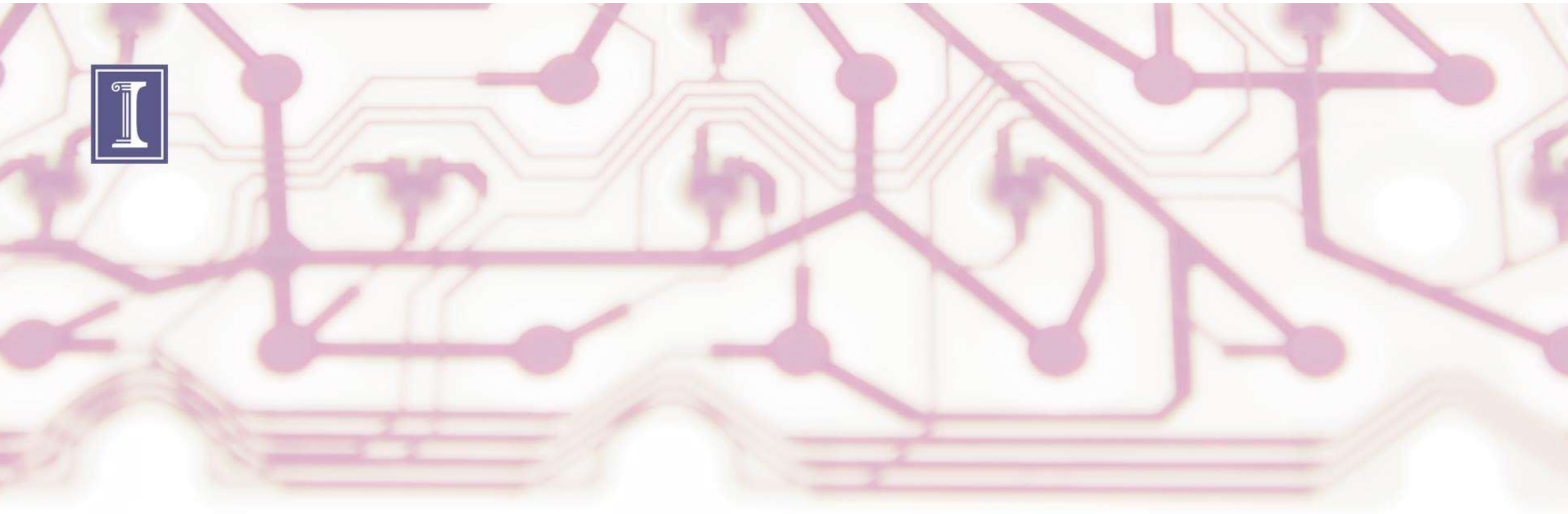
Invocation Semantics

Retransmit request message	Duplicate filtering	Re-execute procedure or retransmit reply	Invocation sematic	Used in …
No	N/A	N/A	*Maybe*	CORBA
Yes	No	Re-execute procedure	*At-least-once*	Sun-RPC
Yes	Yes	Retransmit old reply	*At-most-once*	RMI;CORBA

Retransmit request message: Whether to retransmit the request message until either a reply is received or the server is assumed to be failed

Duplicate filtering: When retransmissions are used, whether to filter out duplicate requests at the server

Re-execute procedure or retransmit reply: Whether to keep a history of result messages to enable lost results to be retransmitted without re-executing the operations



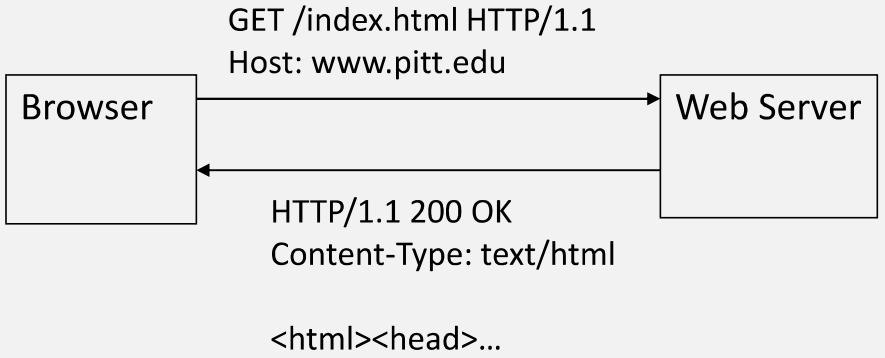
CLOUD COMPUTING APPLICATIONS

HTTP SOAP REST

Prof. Roy Campbell

Hypertext Transfer Protocol (HTTP)

- A communications protocol
- Allows retrieving inter-linked text documents (hypertext)
  - World Wide Web
- HTTP verbs
  - HEAD
  - **GET**
  - **POST**
  - PUT
  - DELETE
  - TRACE
  - OPTIONS
  - CONNECT



SOAP – Simple Object Access Protocol

- Transmitted by HTTP or SMTP (or many others)
- Coded in XML (can be decoded on any machine)
- Return value: any XML document
- Underlies Web Services Description Language (WSDL)

Representational State Transfer (REST)

- A style of software architecture for distributed hypermedia systems such as the World Wide Web
- Introduced in the doctoral dissertation of Roy Fielding
  - One of the principal authors of the HTTP specification
- A collection of network architecture principles that outline how resources are defined and addressed

REST and HTTP

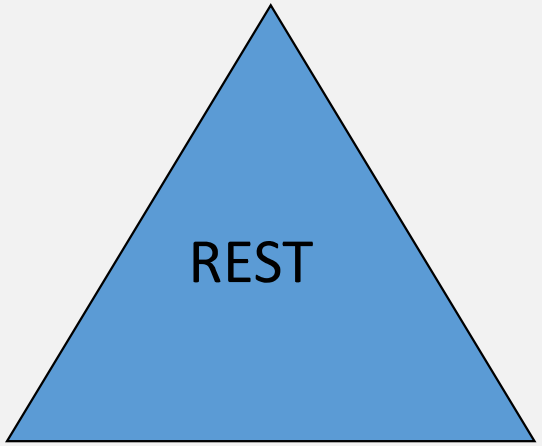
- The motivation for REST was to capture those characteristics of the Web that made the Web successful
  - URI-addressable resources
  - HTTP
  - Make a request – receive response – display response
- Exploits the use of the HTTP beyond HTTP POST and HTTP GET
  - HTTP PUT, HTTP DELETE

REST – Not a Standard

- REST is not a standard
  - JSR 311: JAX-RS: The Java™ API for RESTful Web Services
- But it uses several standards:
  - HTTP
  - URL
  - XML/HTML/GIF/JPEG/etc. (resource representations)
  - Text/xml, text/html, image/gif, image/jpeg, etc. (resource types, MIME types)

Main Concepts

Nouns (resources)
unconstrained
i.e., http://example.com/employees/12345



Verbs
constrained
i.e., GET

Representations
constrained
i.e., XML

Resources

- The key abstraction of information in REST is a resource
- A resource is a conceptual mapping to a set of entities
  - Any information that can be named can be a resource: a document or image, a temporal service (e.g., "today's weather in Los Angeles"), a collection of other resources, a non-virtual object (e.g., a person), and so on
- Represented with a global identifier (URI in HTTP)
  - http://www.boeing.com/aircraft/747

Naming Resources

- REST uses URI to identify resources
  - http://localhost/books/
  - http://localhost/books/ISBN-0011
  - http://localhost/books/ISBN-0011/authors
  - http://localhost/classes
  - http://localhost/classes/cs2650
  - http://localhost/classes/cs2650/students
- As you traverse the path from more generic to more specific, you are navigating the data

Verbs

- Represent the actions to be performed on resources

- HTTP GET
- HTTP POST
- HTTP PUT
- HTTP DELETE

HTTP GET

- How clients ask for the information they seek
- Issuing a GET request transfers the data from the server to the client in some representation

- GET http://localhost/books
  - Retrieve all books
- GET http://localhost/books/ISBN-0011021
  - Retrieve book identified with ISBN-0011021
- GET http://localhost/books/ISBN-0011021/authors
  - Retrieve authors for book identified with ISBN-0011021

HTTP POST, HTTP PUT

- HTTP POST creates a resource
- HTTP PUT updates a resource

- POST http://localhost/books/
  - Content: {title, authors[], …}
  - Creates a new book with given properties

- PUT http://localhost/books/isbn-111
  - Content: {isbn, title, authors[], …}
  - Updates book identified by isbn-111 with submitted properties

HTTP DELETE

- Removes the resource identified by the URI

- DELETE http://localhost/books/ISBN-0011
  - Delete book identified by ISBN-0011

Representations

- How data is represented or returned to the client for presentation
- Two main formats:
  - JavaScript Object Notation (JSON)
  - XML
- It is common to have multiple representations of the same data

Representations

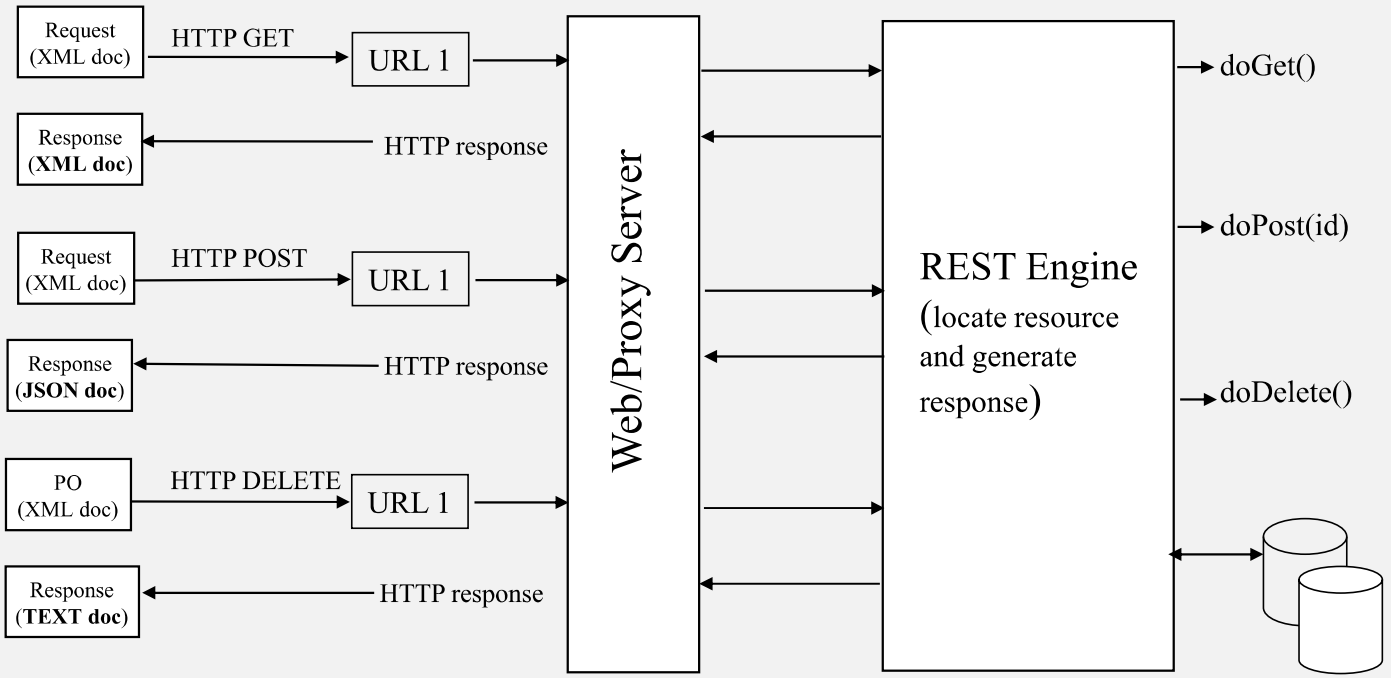
- XML

```
<COURSE>
    <ID>CS2650</ID>
    <NAME>Distributed Multimedia Software</NAME>
</COURSE>
```

- JSON

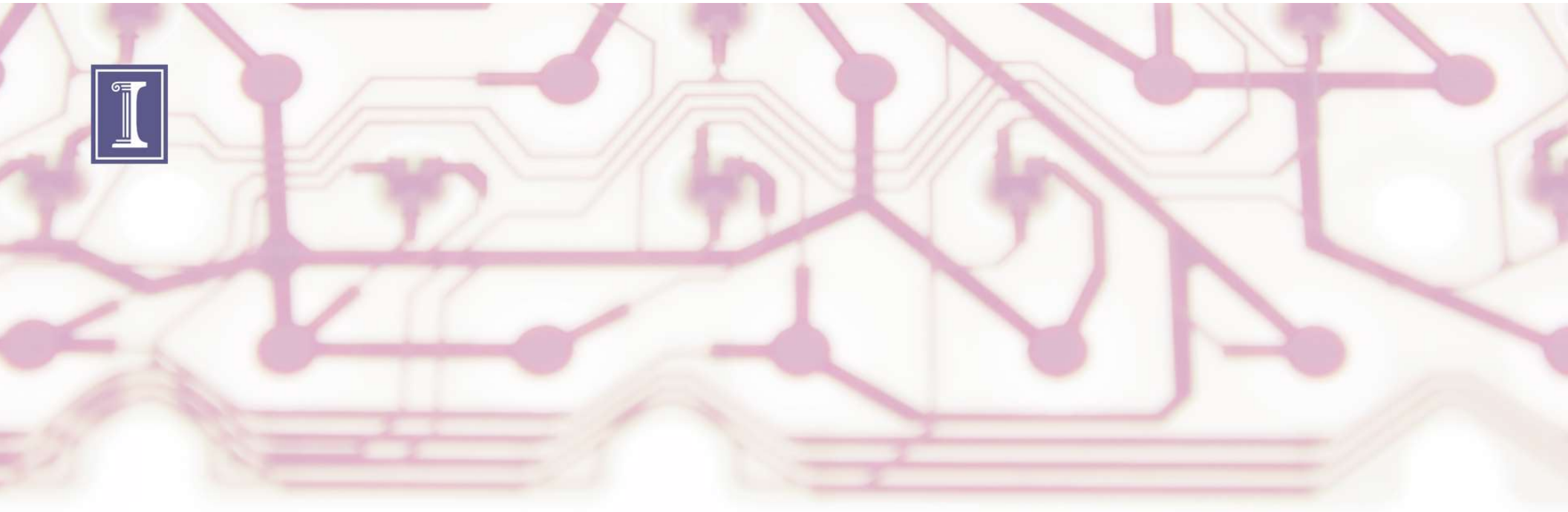
```
{
  "course":{
    "id":"CS2650",
    "name":"Distributed Multimedia Software"
  }
}
```

Architecture Style



Real Life Examples

- Google Maps
- Google AJAX Search API
- Yahoo Search API
- Amazon WebServices



CLOUD COMPUTING APPLICATIONS

JSON

Prof. Roy Campbell

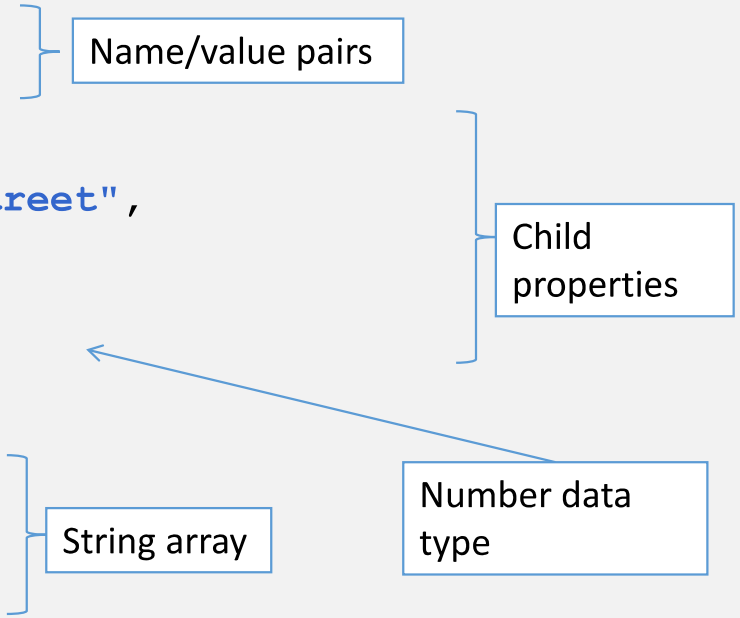
JSON – What Is It?

- *"JSON (JavaScript Object Notation) is a **lightweight data interchange format**. It is easy for humans to read and write. It is easy for machines to parse and generate."*
  - JSON.org

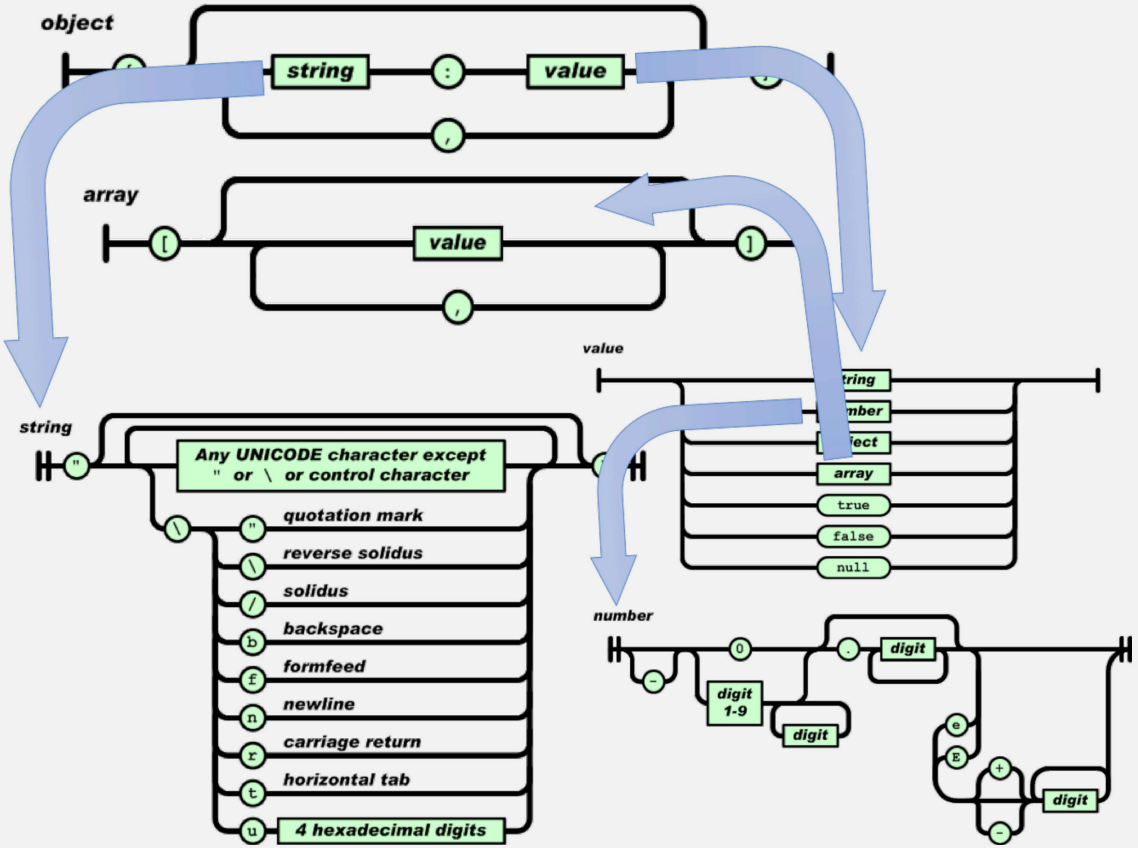
- Importantly: JSON is a subset of JavaScript

JSON – What Does It Look Like?

```
{
    "firstName": "John",
    "lastName": "Smith",
    "address": {
        "streetAddress": "21 2nd Street",
        "city": "New York",
        "state": "NY",
        "postalCode": 10021
    },
    "phoneNumbers": [
        "212 555-1234",
        "646 555-4567"
    ]
}
```

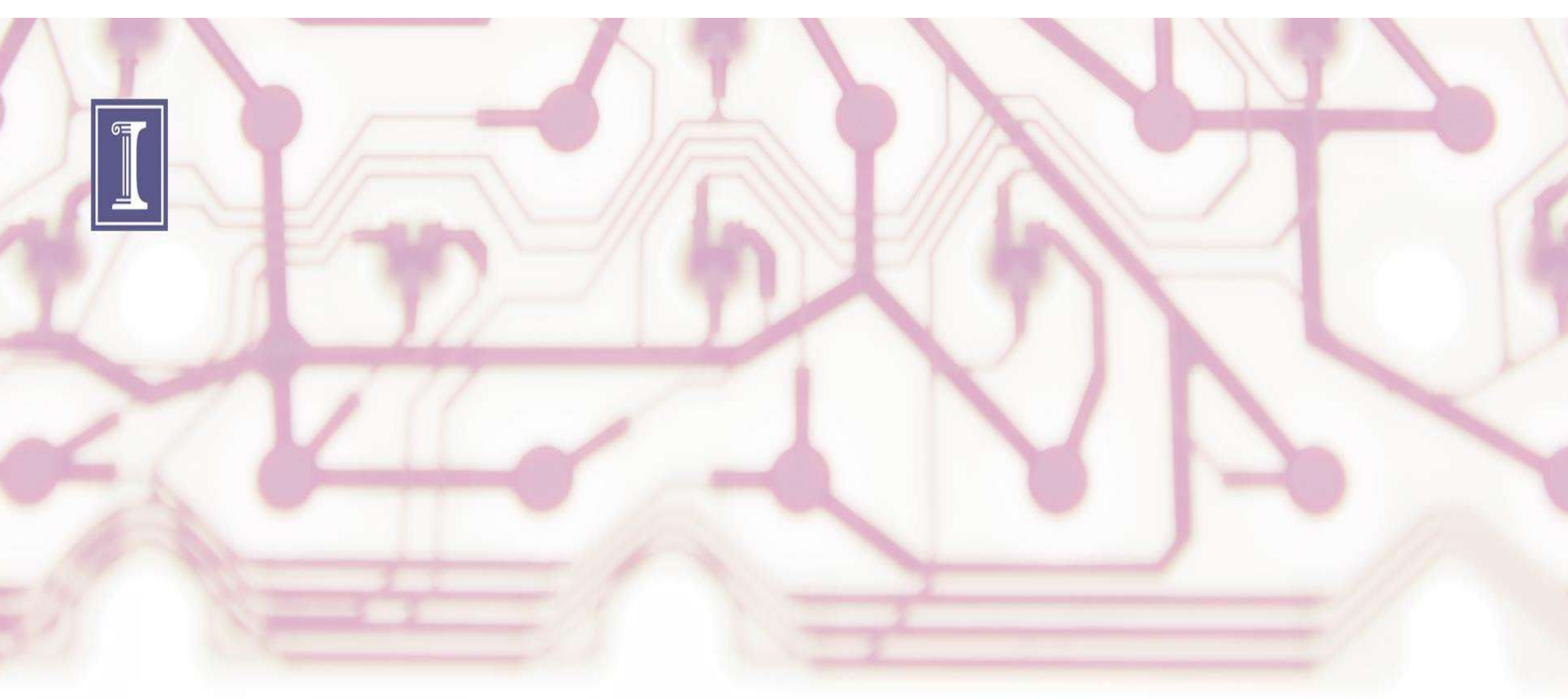


JSON Data Structures



JSON vs. XML

JSON	XML
Data Structure	Data Structure
No validation system	XSD
No namespaces	Has namespaces (can use multiples)
Parsing is just an eval •Fast •Security issues	Parsing requires XML document parsing using things like XPath
In JavaScript you can work with objects – runtime evaluation of types	In JavaScript you can work with strings – may require additional parsing
Security: Eval() means that if the source is not trusted anything could be put into it. Libraries exist to make parsing safe(r)	Security: XML is text/parsing – not code execution



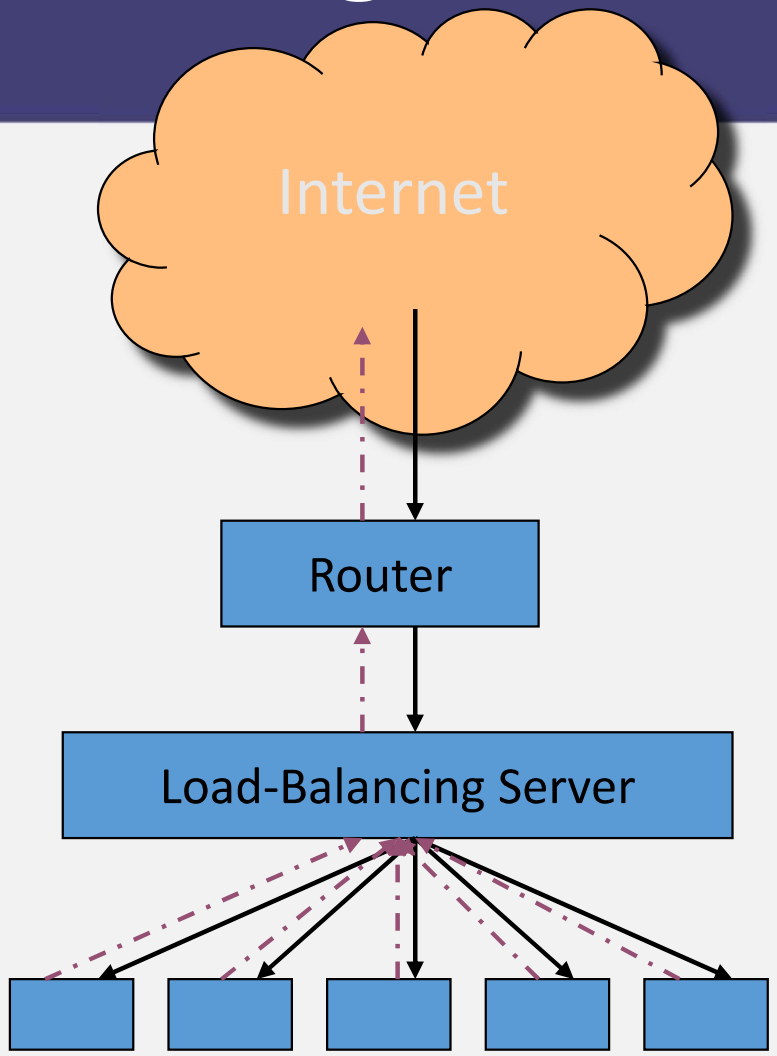
CLOUD COMPUTING APPLICATIONS

LOAD BALANCER INTRO

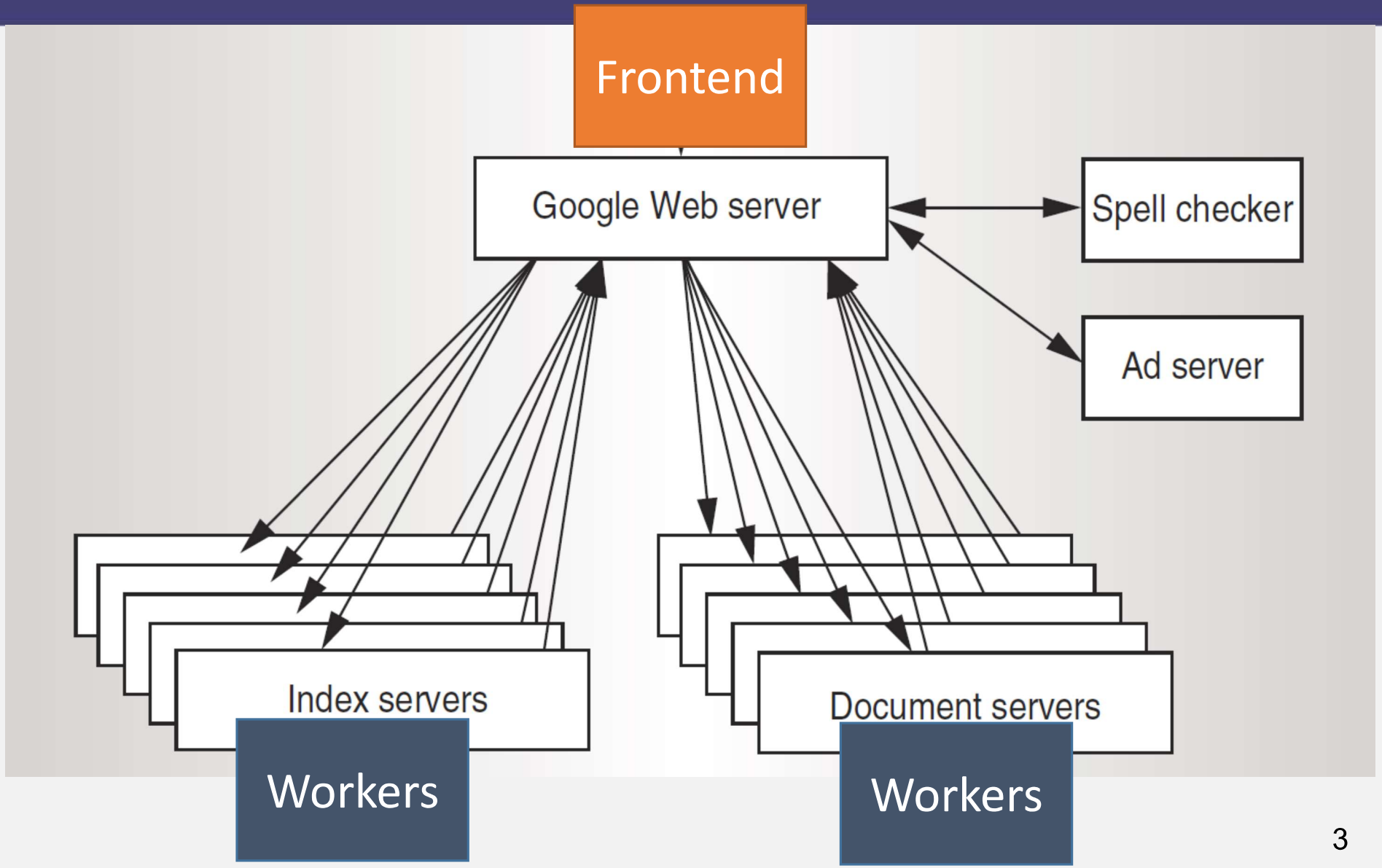
Prof. Roy Campbell

Introduction to Load Balancing

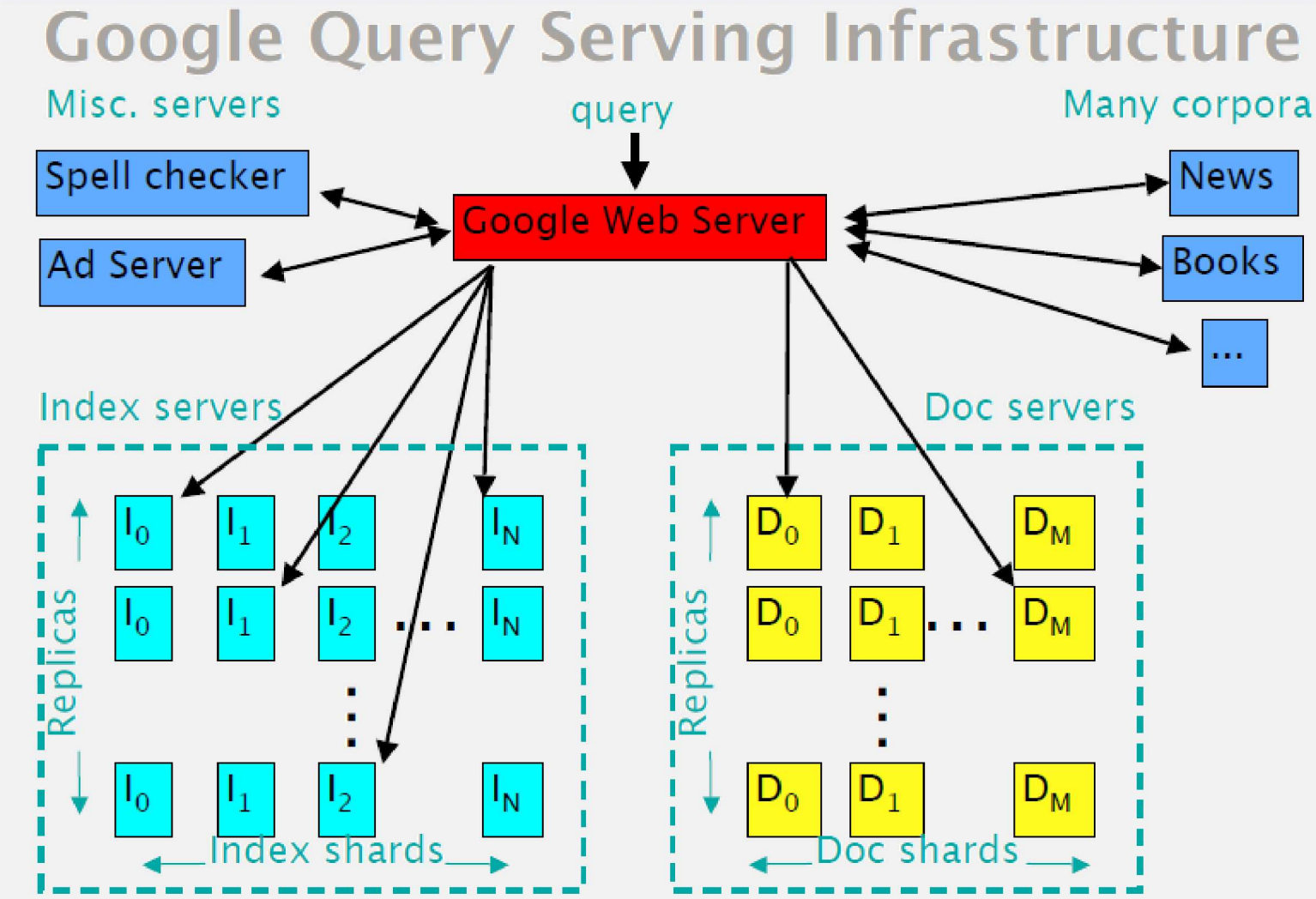
- Request enters a router
- Load balancing server determines which web server should serve the request
- Sends the request to the appropriate web server



Web Search for a planet: The Google Cluster Architecture (2003)



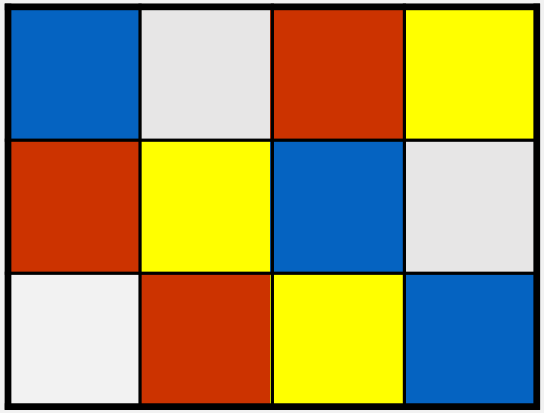
Google: A Behind-the-Scenes Tour



Elapsed time: 0.25s, machines involved: 1000s+

How do we split up information?

Content



?

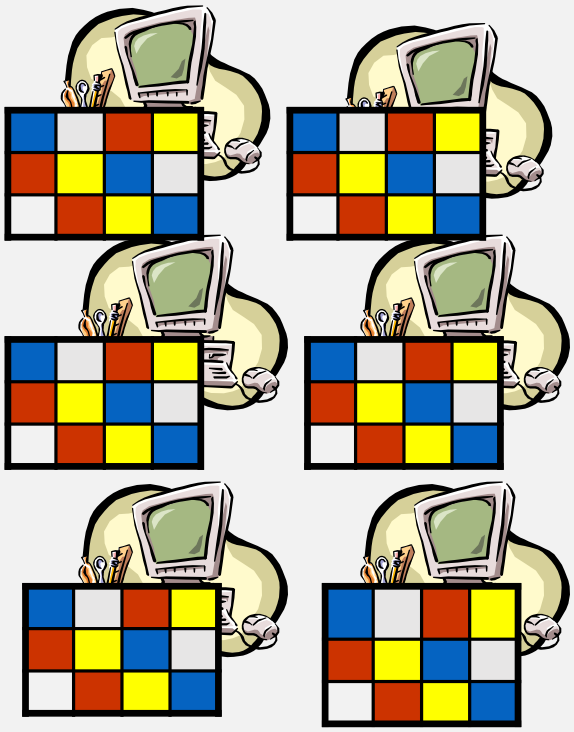
Server Farm



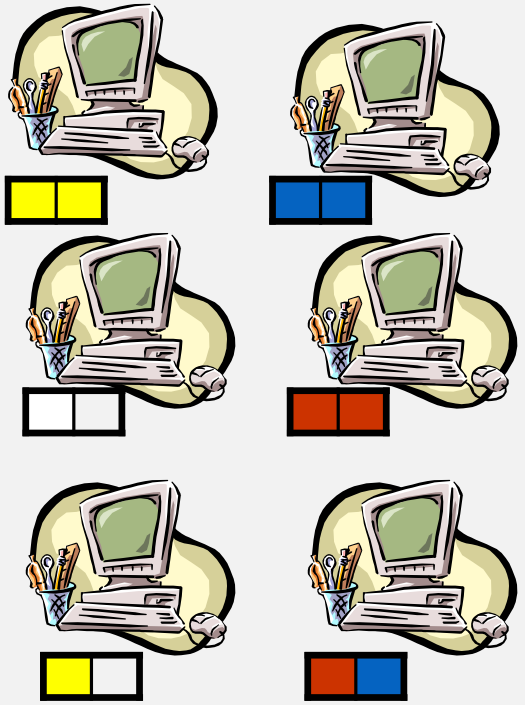



Information Strategies

Replication



Partition

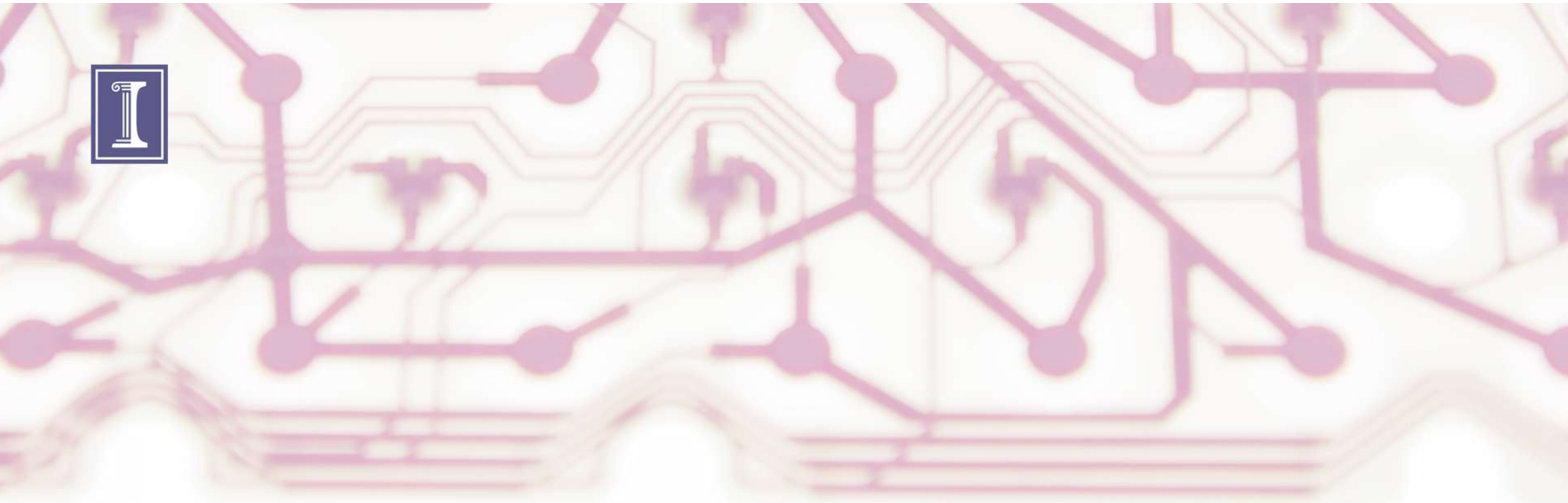


Load Balancing Approaches

File Distribution	Routing
Content/Locality Aware	DNS Server
Size Aware	Centralized Router
Workload Aware	Distributed Dispatcher

Issues

- Efficiently processing requests with optimizations for load balancing
  - Send and process requests to a web server that has files in cache
  - Send and process requests to a web server with the least amount of requests
  - Send and process requests to a web server determined by the size of the request



CLOUD COMPUTING APPLICATIONS

LOAD BALANCER SCHEMES

Prof. Roy Campbell

What Does a Server Load Balancer (SLB) Do?

- Gets user to needed resource
  - Server must be available
  - User's "session" must not be broken
    - If user must get to the same resource over and over, the SLB device must ensure that happens (i.e., session persistence)
- In order to do work, SLB must
  - Know servers – IP / port, availability
  - Understand details of some protocols (e.g., FTP, SIP)
- Network Address Translation (NAT)
  - Packets are rewritten as they pass through the SLB device

Reasons to Load-Balance

- Scale applications / services
- Ease of administration / maintenance
  - Easily and transparently remove physical servers from rotation in order to perform any type of maintenance on that server
- Resource sharing
  - Can run multiple instances of an application / service on a server; could be running on a different port for each instance; can load-balance to different port based on data analyzed

Load-Balancing Algorithms

- Most predominant
  - **Least connections**: Server with fewest number of flows gets the new flow request
  - **Weighted least connections**: Associate a weight / strength for each server and distribute load across server farm based on the weights of all servers in the farm
  - **Round robin**: Round robin through the servers in server farm
  - **Weighted round robin**: Give each server “weight” number of flows in a row; weight is set just like it is in weighted least flows
- There are other algorithms that look at or try to predict server load in determining the load of the real server

How SLB Devices Make Decisions

- The SLB device can make its load-balancing decisions based on several factors
  - Some of these factors can be obtained from the packet headers (i.e., IP address, port numbers)
  - Other factors are obtained by looking at the data beyond the network headers. Examples:
    - HTTP cookies
    - HTTP URLs
    - SSL client certificates
- The decisions can be based strictly on flow counts, or they can be based on knowledge of application
- For some protocols, like FTP, you must have knowledge of protocol to correctly load-balance (i.e., control and data connection must go to same physical server)

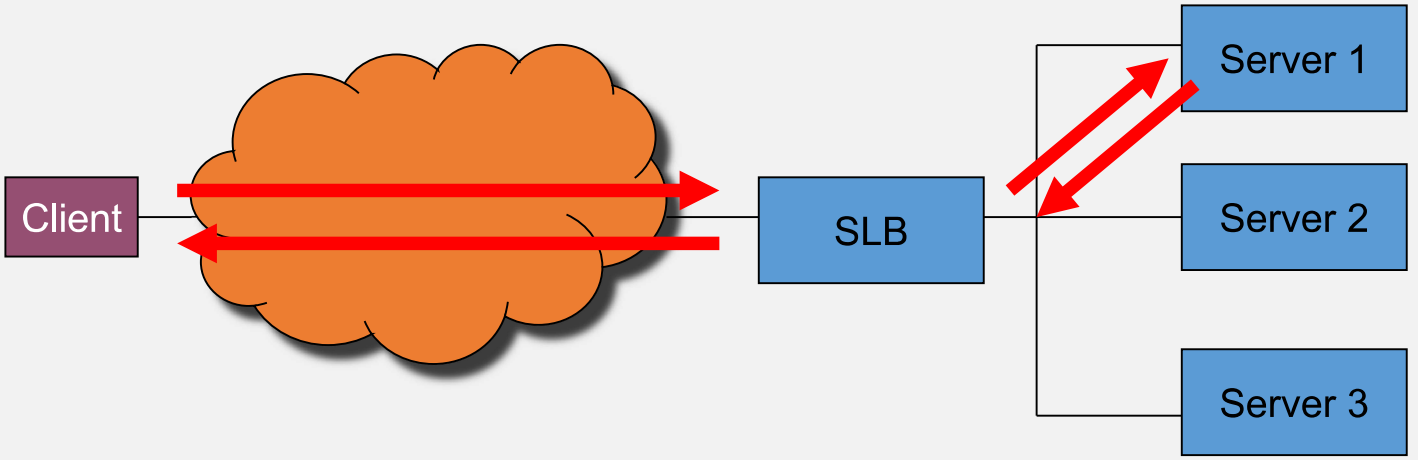
When a New Flow Arrives

- Determine whether virtual server exists
  - If so, make sure virtual server has available resources
  - If so, then determine level of service needed by that client to that virtual server
    - If virtual machine is configured with particular type of protocol support of session persistence, then do that work
  - Pick a real server for that client
    - The determination of real server is based on flow counts and information about the flow
    - In order to do this, the SLB may need to proxy the flow to get all necessary information for determining the real server; this will be based on the services configured for that virtual server
- If not, the packet is bridged to the correct interface based on Layer 2

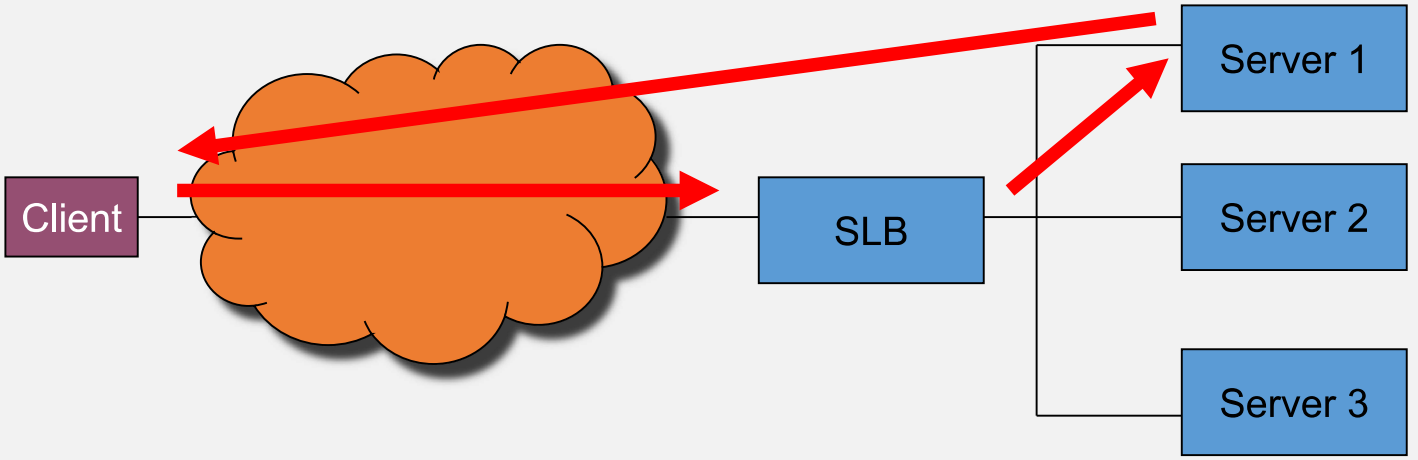
SLB: Architectures

- Traditional
  - SLB device sits between the Clients and the Servers being load-balanced
- Distributed
  - SLB device sits off to the side and only receives the packets it needs to, based on flow setup and teardown

SLB: Traditional View with NAT



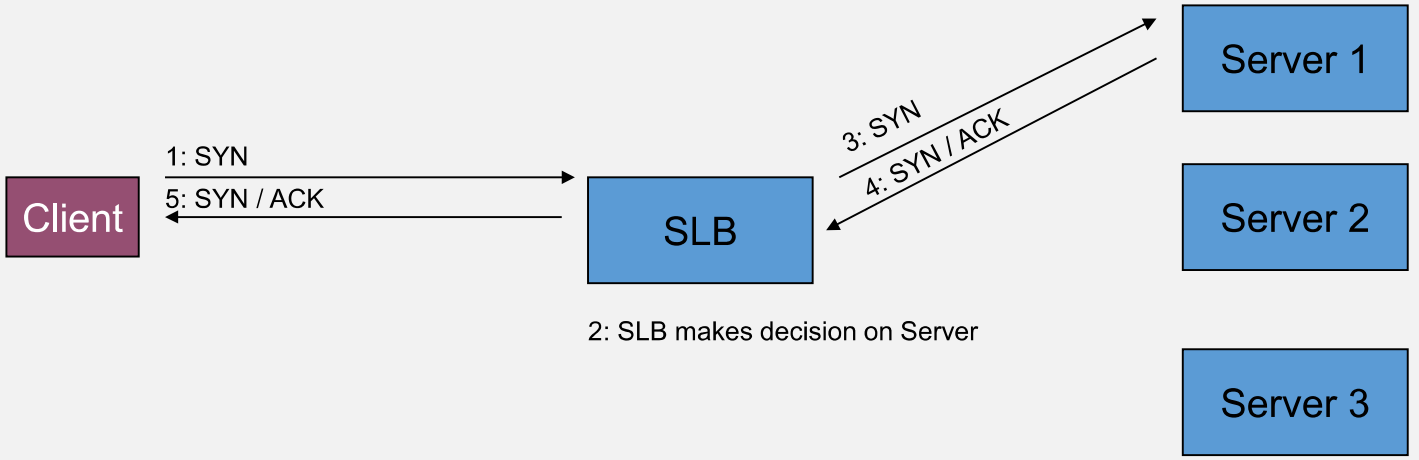
SLB: Traditional View without NAT



Load-Balance: Layer 3 / 4

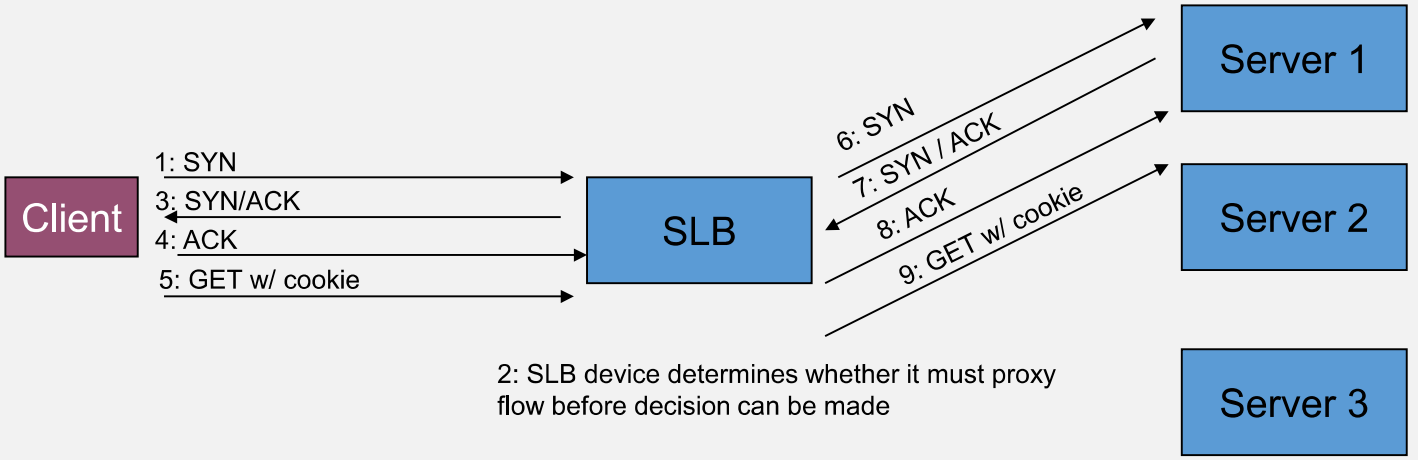
- Look at the destination IP address and port to make a load-balancing decision
- In order to do that, you can determine a real server based on the first packet that arrives

Layer 3 / 4: Sample Flow



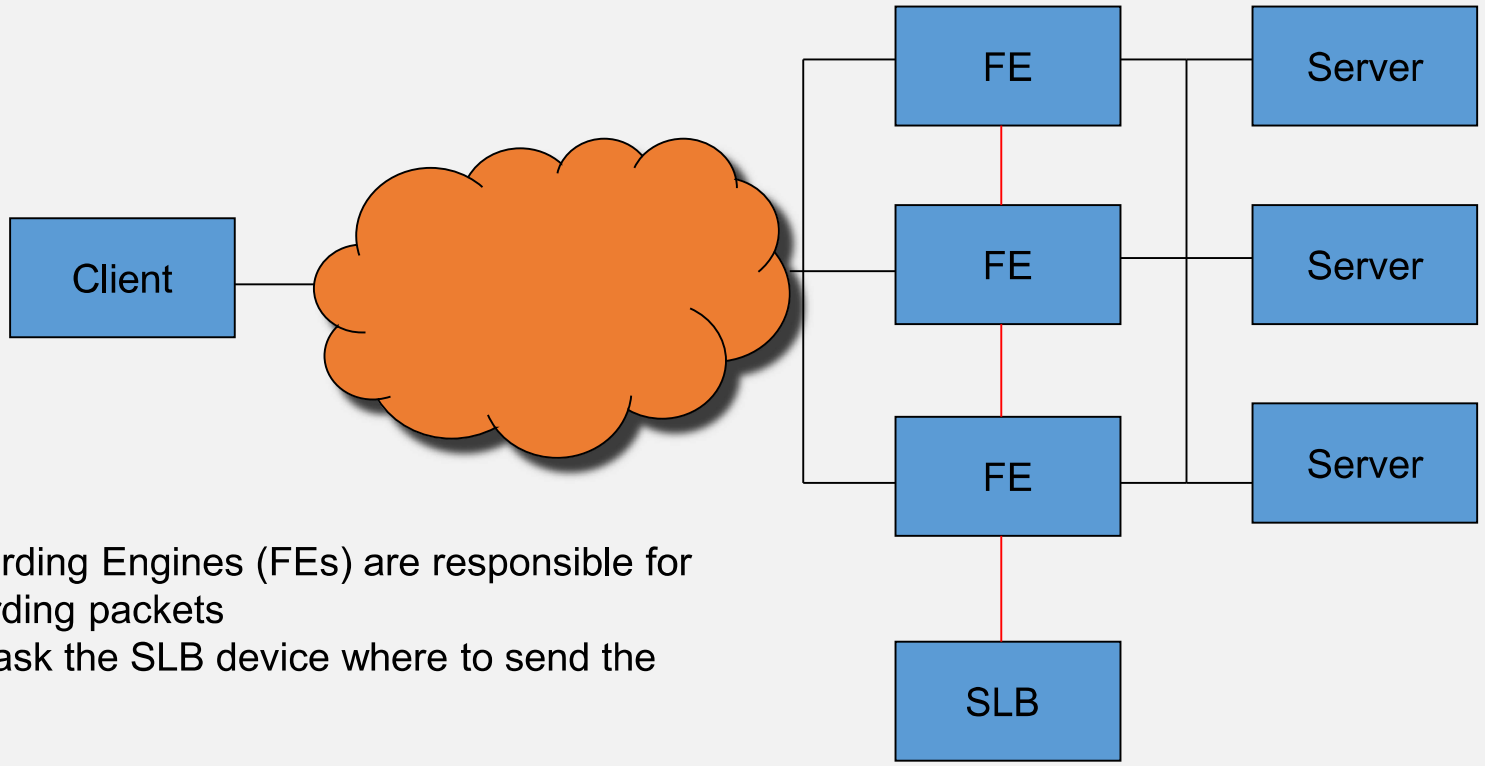
Rest of flow continues through HTTP GET and Server response

Layer 5+: Sample Flow



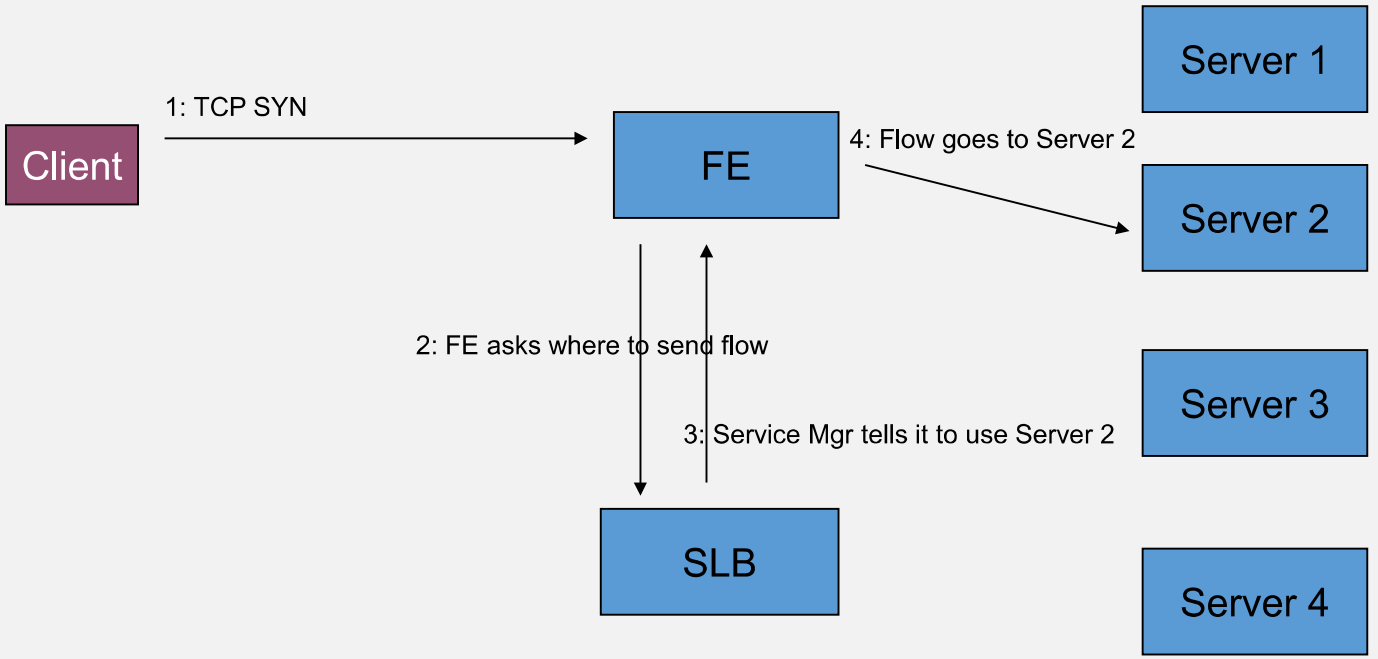
- Rest of flow continues with Server response
- Note that the flow can be unproxied at this point for efficiency

SLB: Distributed Architecture

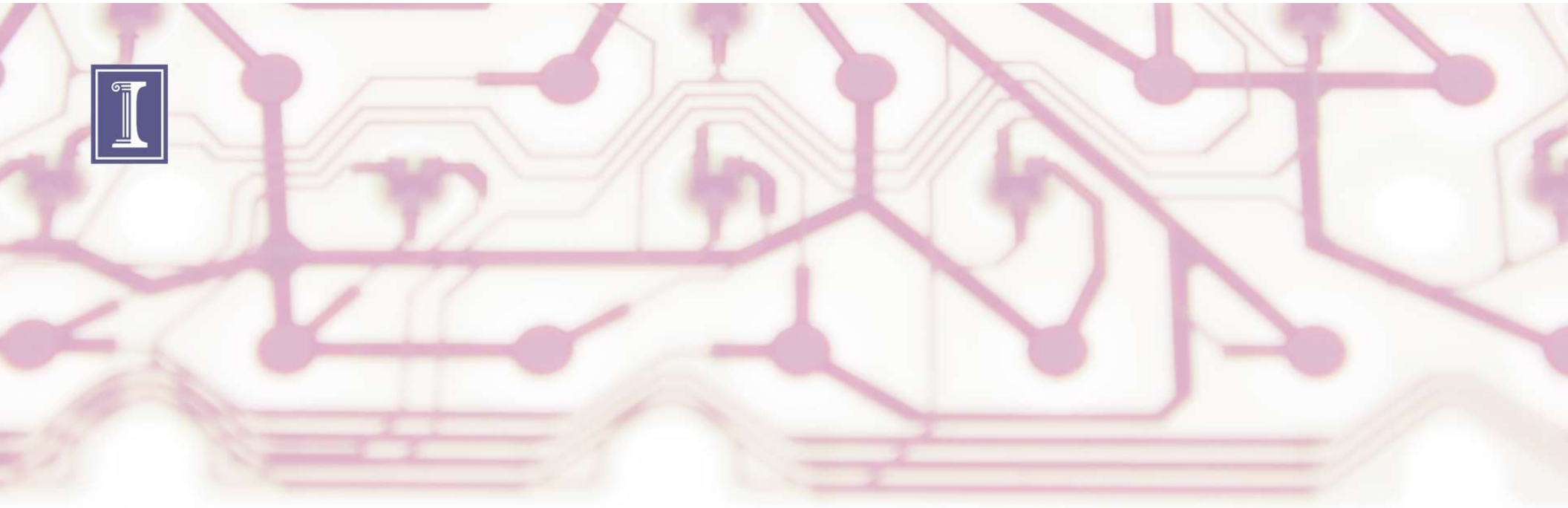


- Forwarding Engines (FEs) are responsible for forwarding packets
- They ask the SLB device where to send the flow

Distributed Architecture: Sample Flow



- Subsequent packets flow directly from Client to Server 2 through the FE
- The FE must notify the SLB device when the flow ends



CLOUD COMPUTING APPLICATIONS

Protocol Buffers and Thrift

Prof. Roy Campbell

Protocol Buffers

- Invented by Google
- Build on RPCs
- Language-neutral, platform-neutral
- Extensible mechanism
- Serializing structured data
- For distributed services

Example Schema

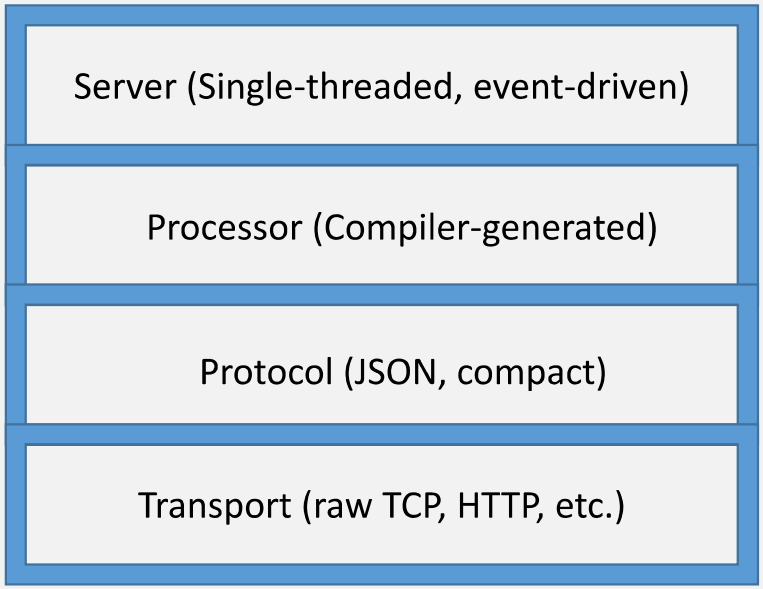
```
1. Message Person{
2.     required int32 id = 1;
3.     required string name = 2;
4.     optional string email = 3;
5. }
```

Example Code (Ruby for example)

```
1.  #!/usr/bin/env ruby
2.  # Generated by the protocol buffer compiler. DO NOT EDIT!
3.  require “protocol_buffers”

4.  # forward declarations
5.  class Person < ::ProtocolBuffers::Message; end

6.  class Person < ::ProtocolBuffers::Message set_fully_qualified_name “Person”

7.  required :int32, :id, 1
8.  required :string, :name, 2
9.  optional :string, :email, 3
10. end
```

Thrift Network Stack

Interface Definition Language. Creates files for clients and servers from needs to serialize structured data (Facebook)



https://thrift.apache.org/static/files/thrift-20070401.pdf

Transport Methods

- Transport
  - open
  - close
  - read
  - write
  - flush
- Server transport also has open, listen, accept, and close allowing
  - **Read / write to / from a file on disk**
  - **http**

Example File Transports

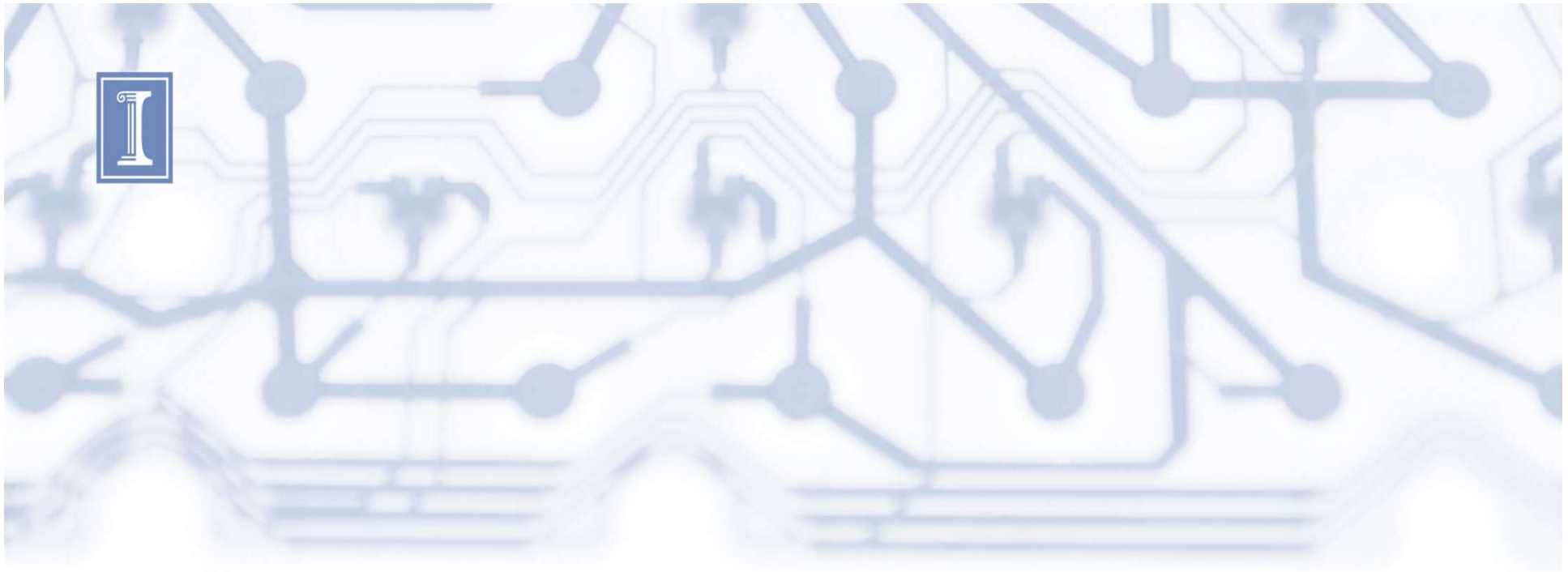
- TFileTransport – This transport writes to a file
- TFramedTransport – Transport for non-blocking server
- TMemoryTransport – Uses memory for I/O
- TSocket – Uses blocking socket I/O for transport
- TZlibTransport – Performs compression using zlib

Example server codes

- TNonblockingServer – A multi-threaded server using non-blocking I/O
- TSimpleServer – A single-threaded server using standard blocking I/O. Useful for testing.
- TThreadPoolServer – A multi-threaded server using standard blocking I/O.

Example Schema

```
1.  struct Person{
2.      int32 id = 0,
3.      string name,
4.      optional string email
5.  }
```



CLOUD COMPUTING APPLICATIONS

Serverless Cloud Computing: Landscape

Prof. Reza Farivar

Serverless Cloud Computing Landscape

- Different Categories of Serverless Computing
  - Compute
    - Platform as a Service (Apps)
    - Function as a Service (Functions)
    - Container as a Service (Containers)
  - Storage
    - Blobs (Binary Large Objects)
    - Key/Value Datastores
  - Analytics
  - AI and ML

Compute: PaaS

- The unit of compute is a full App
- You still select which server / platform you need, only that you don't need to manage it anymore

- Amazon Elastic BeanStalk
- Google AppEngine
- Microsoft Azure App Service
- IBM CloudFoundry
  - Based on Open Source CloudFoundry
    - Originally developed by VMware, transferred to Pivotal Software (a joint venture by EMC, VMware and General Electric), brought back into VMware at the end of 2019
- Oracle Java Cloud Service

Function as a Service

- This is what most people think of as “Serverless”
- Unit of compute is a function
  - Functions running when “events” are triggered

- Amazon AWS Lambda
- Microsoft Azure Functions
- Google Cloud Functions
- IBM Cloud Functions
  - Based on Apache Open Whisk
- Oracle Functions
  - Apache Fn

- Open Source Open Lambda

Container as a Service

- Function as a Service on HEAVY steroids!
  - Containers have to be Stateless
  - Unit of compute is a whole container
  - Container running when “events” are triggered

- Amazon Elastic Container Service (ECS), Elastic Kubernetes Service (EKS), Fargate
- Microsoft Azure Kubernetes Service (AKS)
  - Built on top of Open Source KEDA
- Google Cloud Run, Anthos
  - Built on top of Open Source Knative
    - Kubernetes-based platform to deploy and manage modern serverless workloads.
- IBM Cloud Kubernetes Service
- Oracle Container Engine for Kubernetes

- Open Source Kubernetes
  - Google Borg

Serverless Storage: Blobs

- Blob Storage

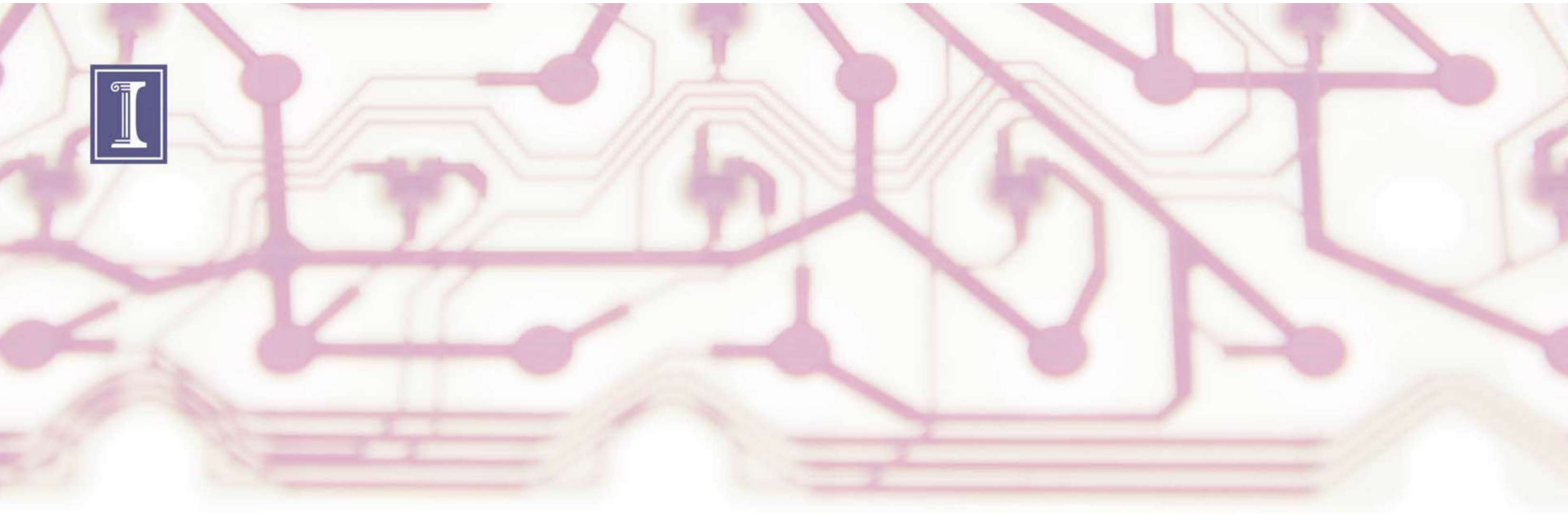
- Amazon Simple Storage Service (S3)
- Microsoft Azure Blob Storage
- IBM Object Storage
- Google Cloud Storage
- Oracle Object Storage

Serverless Storage: Key/Value Database

- Distributed NOSQL key/value storage service

- Amazon DynamoDB
- Microsoft Azure Cosmos DB
- Google Cloud Datastore, Firestore, Cloud BigTable
- IBM Cloudant

- Open Source: Kassandra



CLOUD COMPUTING APPLICATIONS

Serverless Architecture

Roy Campbell & Reza Farivar

Introduction to Serverless Architecture

- “Applications where some amount of server-side logic is still written by the application developer but unlike traditional architectures is run in stateless compute containers that are event-triggered, ephemeral (may only last for one invocation), and fully managed by a 3rd party”
- ‘Functions as a service / FaaS’
- AWS Lambda is one of the most popular implementations of FaaS at present, but there are others