

COMP7940

Distributed and Cloud

Computing

Chapter 03

Communication Paradigms in

Distributed Systems

Reading: Textbook Chapter 3, 4, and external references

Unless specified, all diagrams are assumed from the textbook.

Learning Outcomes

- Having a essential knowledge in computer network.
- Be able to describe the characteristics of the three major communication paradigms in DS:
 - Interprocess communication
 - Remote invocation
 - Indirect communication
- Give examples of the different communication paradigms, such as
 - Socket
 - MPI
 - Java RMI
 - publish-subscribe systems
 - SOA

Outline

- Network Essentials

Communication Paradigms

- External data representation
 - Java Serialization
- Interprocess communication
 - MPI
- Remote invocation
 - Java RMI
- Indirect communication
 - Publish-subscribe systems
- Service-oriented architecture (SOA)
 - Web services/SOAP
 - Restful API

Computer Networks

- Different Types of Networks:
 - PAN: Personal Area Networks enable various digital devices carried by a user are connected by a low-cost, low-energy network. (e.g. smart home, iPad, TV)
 - LAN (Ethernet): **L**ocal **A**rea **N**etworks carry message at relative high speeds (10/100/1000Mbps) between computers connected CAT5/CAT6 cables, fibers.
 - WAN: **W**ide **A**rea **N**etworks carry message across different organizations by large distances. Routers are installed to communicate between networks

External Data Representation and Marshalling

- Different systems may store the same information in different ways
 - Big-endian system vs. little-endian system
 - Different formats for floating-point numbers
 - ASCII code vs. Unicode

Big Endian

12	34	56	78
0x00400000	0x00400001	0x00400002	0x00400003

Little Endian

78	56	34	12
0x00400000	0x00400001	0x00400002	0x00400003

sign exponent mantissa

0	1	1	0	1	0	1	1
---	---	---	---	---	---	---	---

↓

$$+ (0.1011)_2 * 10^{(110)_{\text{3bit excess-k}}}$$

$$= (0.1011)_2 * 10^2$$

$$= (10.11)_2$$

$$= (1*2)^1 + (1*2)^0 + (0*2)^{-1} + (1*2)^{-2} = 2.75$$

External Data Representation and Marshalling

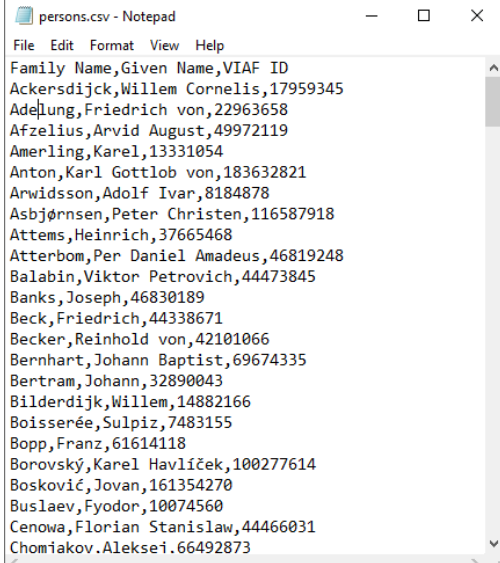
- External data representation is an agreed standard for the representation of data structures and primitive values
 - It enables any two systems to exchange binary data values.
 - **Marshalling**: the process of taking a collection of data items and assembling them into a form suitable for transmission in a message
 - **Unmarshalling**: the process of disassembling a message to produce an equivalent collection of data items at the destination

External Data Representation and Marshalling

- Some popular data exchange format:
 - Binary/raw: save space
 - CSV: comma separated values
 - XML: eXtensible Markup Language
 - JSON: JavaScript Objection Notation

```
- <Parts>
- <Part>
  <Id>4478</Id>
  <Part_Name>1000 Ohm Resistor</Part_Name>
  <Total_Available>25000</Total_Available>
  <Price>0.01</Price>
</Part>
- <Part>
  <Id>3328</Id>
  <Part_Name>15000 Ohm Resistor</Part_Name>
  <Total_Available>75000</Total_Available>
  <Price>0.02</Price>
</Part>
- <Part>
  <Id>4725</Id>
  <Part_Name>555 Timer IC</Part_Name>
  <Total_Available>1500</Total_Available>
  <Price>0.25</Price>
</Part>
</Parts>
```

```
[
  {
    "description": "quarter",
    "mode": "REQUIRED",
    "name": "qtr",
    "type": "STRING"
  },
  {
    "description": "sales representative",
    "mode": "NULLABLE",
    "name": "rep",
    "type": "STRING"
  },
  {
    "description": "total sales",
    "mode": "NULLABLE",
    "name": "sales",
    "type": "INTEGER"
  }
]
```



persons.csv - Notepad

File Edit Format View Help

Family Name,Given Name,VIAF ID

Ackersdijck,Willem Cornelis,17959345

Adelung,Friedrich von,22963658

Afzelius,Arvid August,49972119

Amerling,Karel,13331054

Anton,Karl Gottlob von,183632821

Arwidsson,Adolf Ivar,8184878

Asbjørnsen,Peter Christen,116587918

Attems,Heinrich,37665468

Atterbom,Per Daniel Amadeus,46819248

Balabin,Viktor Petrovich,44473845

Banks,Joseph,46830189

Beck,Friedrich,44338671

Becker,Reinhold von,42101066

Bernhart,Johann Baptist,69674335

Bertram,Johann,32890043

Bilderdijk,Willem,14882166

Boisserée,Sulpiz,7483155

Bopp,Franz,61614118

Borovský,Karel Havlíček,100277614

Bosković,Jovan,161354270

Buslaev,Fyodor,10074560

Cenowa,Florian Stanislaw,44466031

Chomiakov,Aleksei,66492873

Images from:

- <https://nodegoat.net/guides/csvfile>
- <https://dimestorerocket.com/read-a-xml-file-fast-with-csharp/>
- <https://cloud.google.com/bigquery/docs/loading-data-cloud-storage-json>

Example:

Java Object Serialization

- In Java, an object is an instance of a Java class.
- Java serialization can flatten an object or a set of objects into a serial form that is suitable for storing on disk or transmitting in a message.
 - By stating that a class implements the ***Serializable*** interface (provided in ***java.io*** package), its instances can be serialized.
 - E.g., ***public class Person implements Serializable { ... }***
 - ***ObjectOutputStream.writeObject(X)*** is used to serialize an object ***X***
- Java deserialization can restore the state of an object or a set of objects from their serialized form.
 - ***X = ObjectInputStream.readObject()*** is used to deserialize an object and store into ***X***

Example:

Python Object Serialization

- Default data type like list/dict can be serialized to JSON, which is more ready for data exchange
 - > `serialized_data = json.dumps(value)`
- For Python Object or better efficiency, use *pickle*
 - > Marshalling: `pickle.dump(object, file)`
 - > Unmarshalling: `restored_object = pickle.load(file)`

Extended reading:

<https://www.journaldev.com/15638/python-pickle-example>

MAC and IP Address

- To join a network, each computer needs to have a NIC (Network Interface Card), could be a network card, a WiFi adapter, a mobile network modem.
- Each NIC has a unique **MAC address**, like everyone has a HKID.
- The internet service provider will assign an **IP address** to each NIC which act like a mailing address or a phone number of a person.
 - Others can contact this NIC via its IP address.
- Mac : for LAN/Switching ; IP : for WAN/Routing

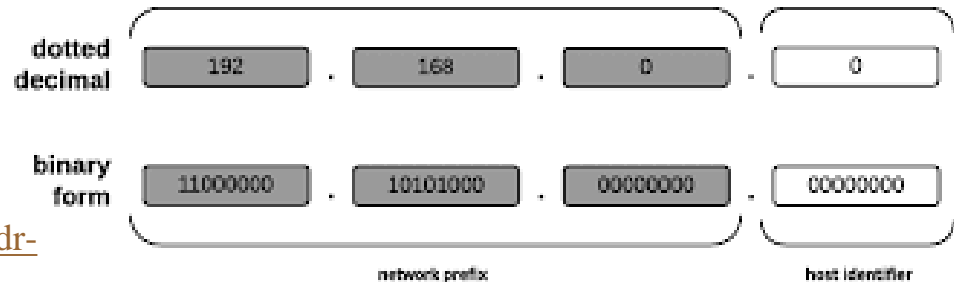
IPv4

- IPv4 is represented by 4 segments of 8bits decimal string, namely
—xxx . xxx. xxx. xxx where xxx is between 0 to 255
- Total number of IP address = 2^{32} , < 1 IP/human.
- CIDR : to refer a group of IP address, a notation <IP>/<Mask Bit> is used.

192.168.0.0 / 24
address network prefix length

192.168.0.0/24 =

192.168.0.0 – 192.168.0.255



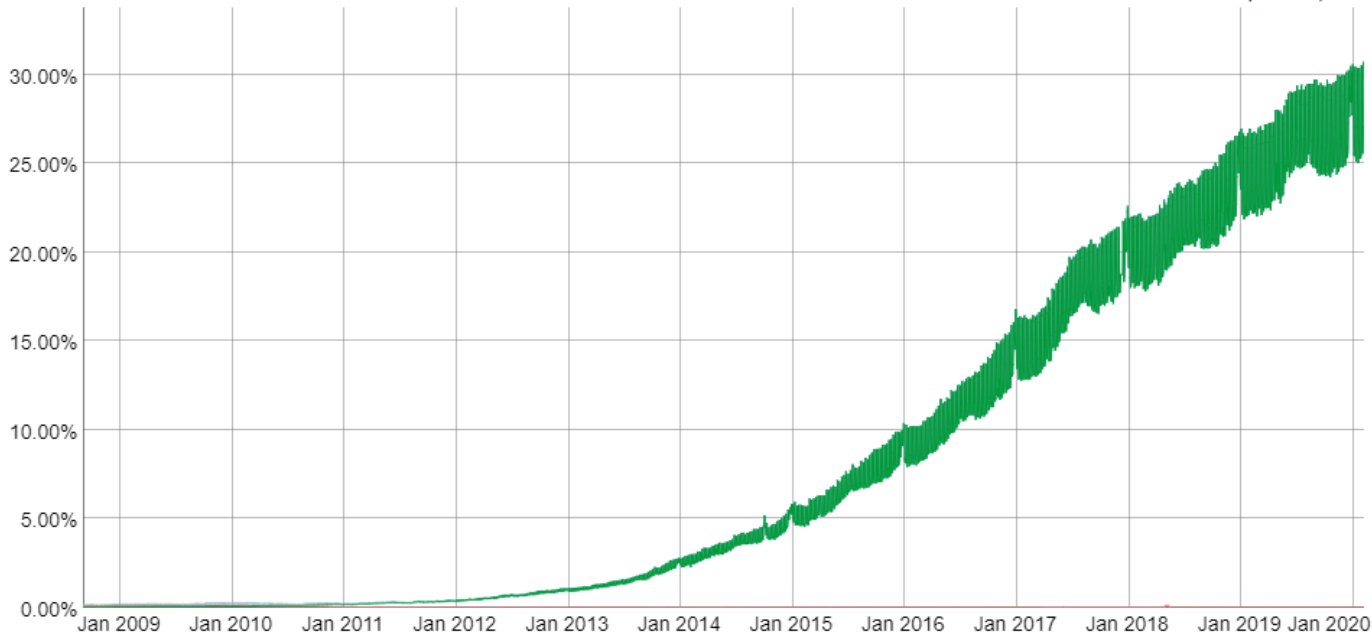
IPv6

- IPv6 is to resolve insufficient IP problems.
- Use 128bits
- All hardware, software are ready, but...

IPv6 Adoption

We are continuously measuring the availability of IPv6 connectivity among Google users. The graph shows the percentage of users that access Google over IPv6.

Native: 30.16% 6to4/Teredo: 0.00% Total IPv6: 30.16% | Dec 29, 2019

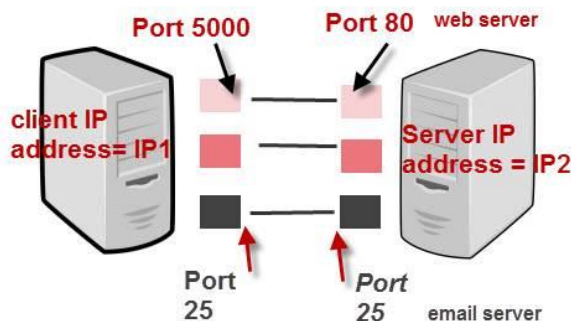


IP Routing

- IP address is globally managed by IANA and ICANN.
 - Allocates IP blocks to the five regional internet registries (RIRs)
 - Each RIR allocates smaller IP blocks to national internet registries (NIRs) and local internet registries (LIRS) and eventually to your internet service provider (ISP).
 - Using Classless Inter-Domain Routing (CIDR) hierarchical addressing scheme, a NIC can be reachable through its ISP.
- Routers based on IP address to route a packet from a source to a destination

IP and Port

- A NIC may have many connections at a time (e.g. open many web browser tabs), each connection needs to have a dedicated **port**.
- Port is a 16-bit unsigned integer ranging from 0 to 65535. Usually ports <1023 are reserved for system used.



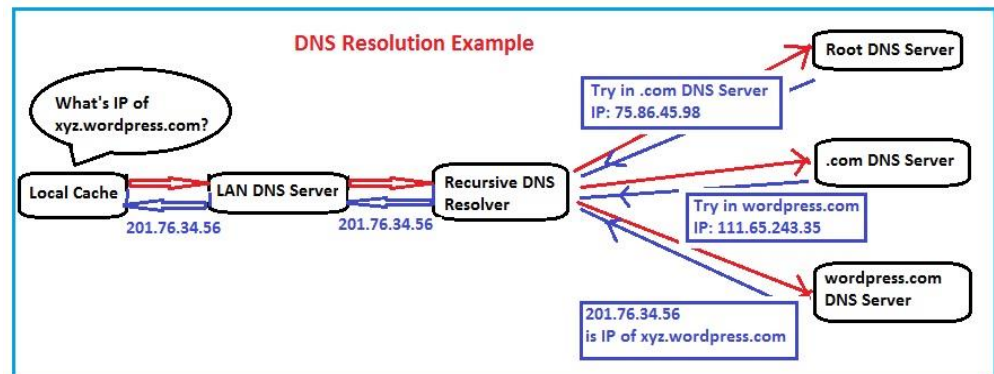
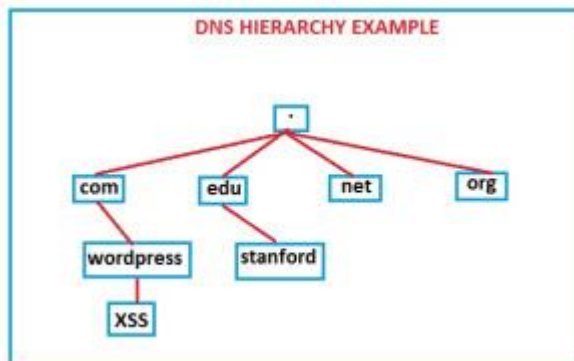
IP Address + Port number = Socket

TCP/IP Ports And Sockets

Port Number	Protocol	Application
20	TCP	FTP data
21	TCP	FTP control
22	TCP	SSH
25	TCP	SMTP
53	UDP, TCP	DNS
80	TCP	HTTP (WWW)
110	TCP	POP3
443	TCP	SSL

DNS Server

- A domain name is composed by different namespaces in a hierarchical pattern, e.g. `www . domain . com . hk`
- A domain name server helps a client to resolve the IP address of a domain name.
- Anyone can register a domain name from a TLD Manager, of course, with \$.



Wireless Network WLAN

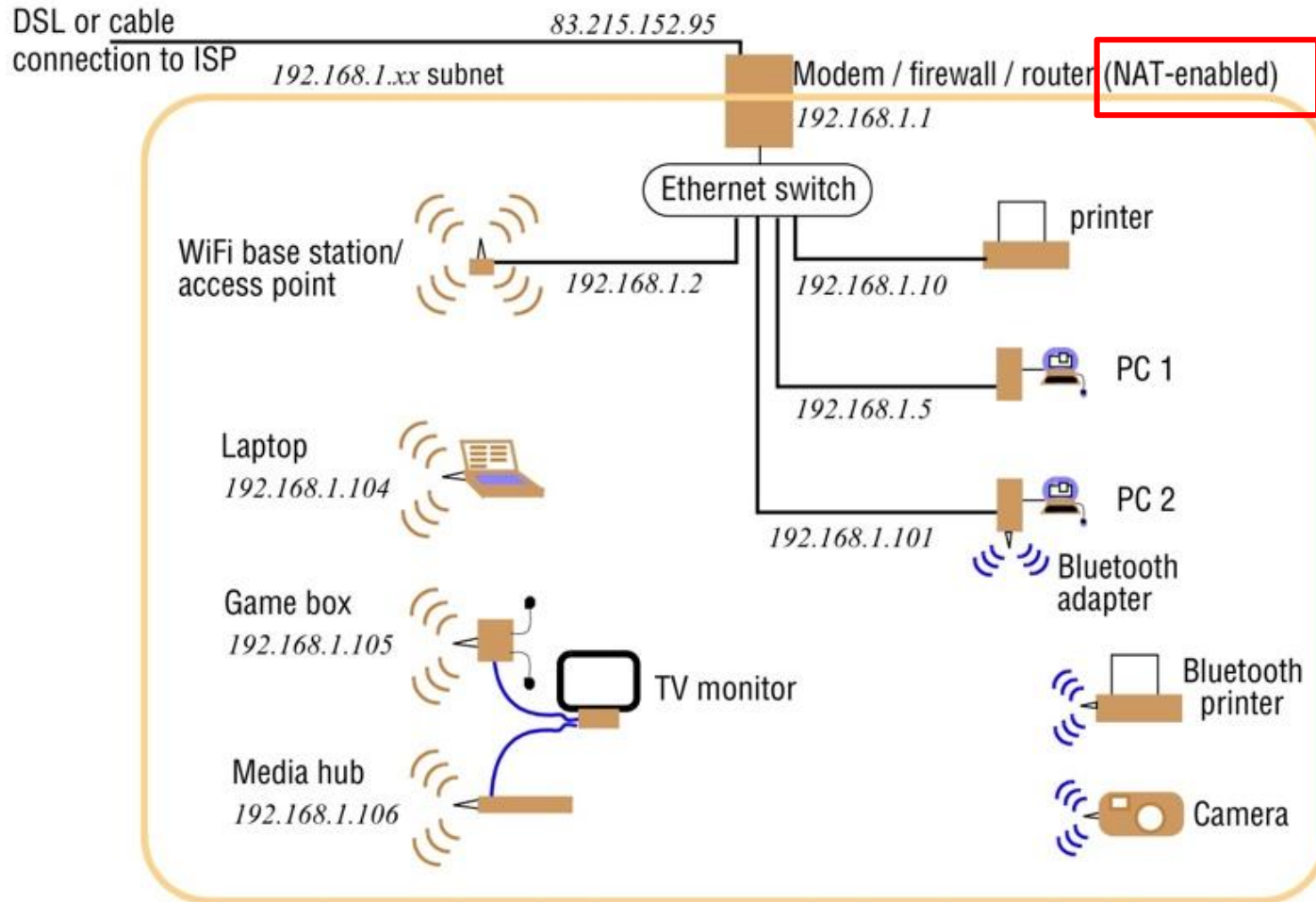
- Commonly known as WiFi, specified in IEEE standard 802.11, evolves since 1999.
- Some protocols specific the physical layer of communication (e.g. 802.11a/b/g/n/ac)
- Some protocols specific other perspectives like roaming (802.11v), security (802.11i)

IEEE Standard	802.11a	802.11b	802.11g	802.11n	802.11ac	802.11ax
Year Released	1999	1999	2003	2009	2014	2019
Frequency	5Ghz	2.4GHz	2.4GHz	2.4Ghz & 5GHz	2.4Ghz & 5GHz	2.4Ghz & 5GHz
Maximum Data Rate	54Mbps	11Mbps	54Mbps	600Mbps	1.3Gbps	10-12Gbps

Typical WLAN structure

- **Modem:** A machine at home that connects to ISP via a telephone cable, a CAT5 cable, a optical fiber, or cellular network.
- **Router:** A device that allocates IP addresses to all devices at home. Usually with wireless connectivity.
- **Access Points (AP):** Devices that connected to a router that has wireless interface to allow devices to connect to. Usually connected to a router using Ethernet cable.
- **Signal Extender:** Amplify the WiFi signal to eliminate blind spot.
- **Home-plug:** two pairs of devices that communicate over power socket. Usually connected to an AP and a router.

A typical WLAN

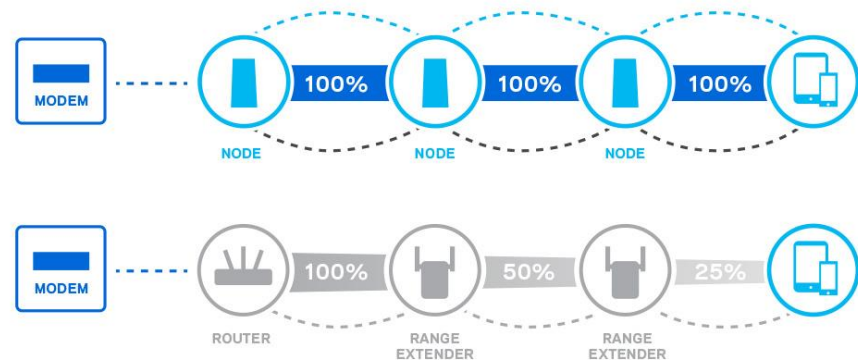


NAT

- Network Address Translation (NAT): not all devices can be assigned with a globally unique IP addresses (not enough in IPv4)
- IPv6 has no such problem but we have not yet fully migrated to IPv6.
- A NAT-enabled router does the following:
 - Give each devices in the LAN/WLAN a “fake” IP.
 - Each device attempts to connect to outside will “borrow” the router IP and a random port.
 - Server outside reply to the borrowed IP/port and the router catches the message.
 - Router relay the message to the “borrower”.

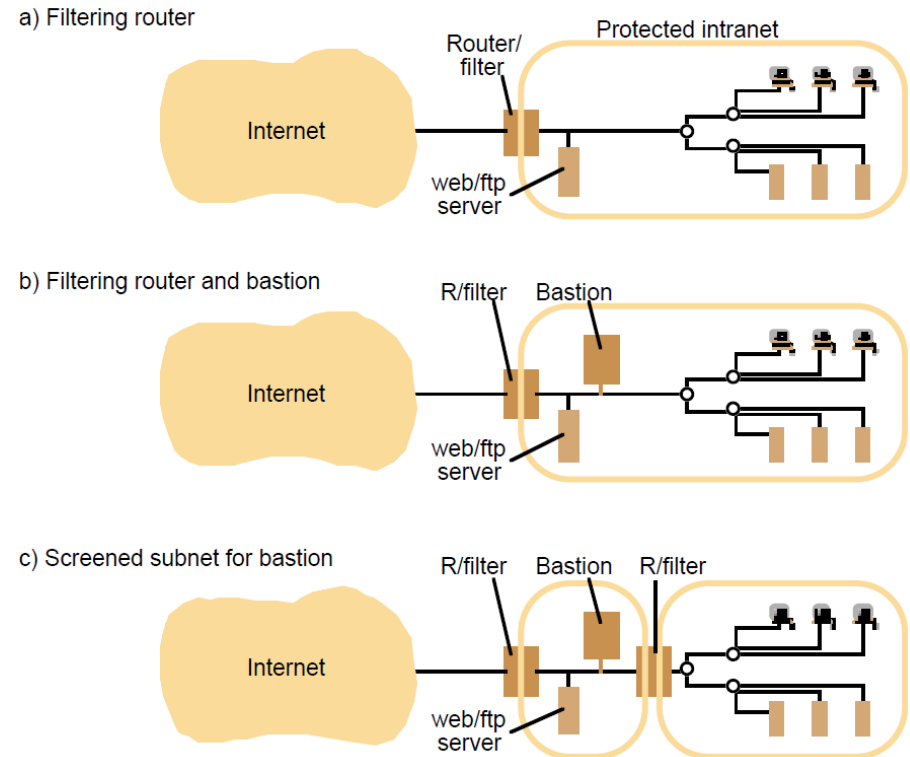
Recent Advancement in WLAN

- Problems: blind spot of WiFi where signal extender is not sufficient.
- Mesh Network refers a network with more than one access points and these access points are interconnected over wireless.
- Commercial Mesh for WLAN
 - Seamless roaming
 - Easy installation



Firewall

- A firewall is a machine that filters external attacks.
 - An Intrusion Detection System (IDS) detects network attacks and rings alarm.
- Networks attacks:
 - Port Scanning
 - DoS/DDoS
 - Phishing



Interprocess Communication (IPC)

- Interprocess communication provides low-level support for communication between *processes* (or *threads*) in DSs, such as
 - Socket programming
 - Message Passing Interface (MPI)
- Message passing between a pair of processes can be supported by two operations: *send* and *receive*
 - One process sends a message to a destination
 - Another process at the destination receives the message

Characteristics of IPC

- Synchronous and asynchronous
 - In synchronous communication, the sending and receiving processes synchronize at every message
 - Both send and receive are blocking operations.
 - Whenever a send is issued, the sending process is blocked until the corresponding receive is issued.
 - Whenever a receive is issued by a process, it blocks until a message arrives.
 - In asynchronous communication, the send operation is non-blocking, while the receive operation can be blocking or non-blocking
 - The sending process can proceed as soon as the message has been copied to a local buffer

Characteristics of IPC (Cont.)

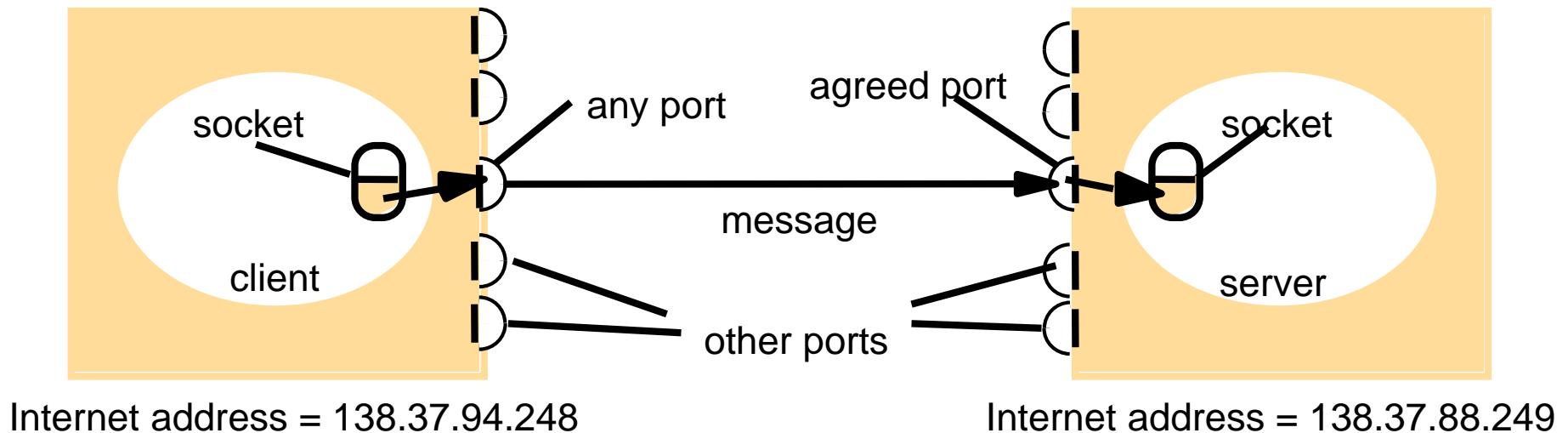
- Reliability: validity and integrity
 - Validity: a message service is reliable if messages are guaranteed to be delivered, despite some packets being dropped or lost.
 - Integrity: messages must arrive uncorrupted and without duplication.
- Ordering: some applications require that messages be delivered in sender order.

Sockets (~1980s)

- Interprocess communication consists of transmitting a message between a *socket* in one process and a *socket* in another process.
 - Specify IP address and Port
 - Using a UDP socket: without acknowledgement or retries; faster.
 - Using a TCP socket: with acknowledgement, flow control, speed control, error detection;
- Still using today, as a primitive level of network communication implementation

IPC Example 1: Java Socket

- Java Socket API supports
 - UDP datagrams
 - TCP stream communication
 - IP multicast



Example of Python Socket

Client

```
import socket
```

```
HOST = '127.0.0.1'
```

```
# The server's hostname or IP address
```

```
PORT = 65432
```

```
# The port used by the server
```

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
```

```
    s.connect((HOST, PORT))
```

```
    s.sendall(b'Hello, world')
```

```
    data = s.recv(1024)
```

```
print('Received', repr(data))
```

Server

```
import socket
```

```
HOST = '127.0.0.1'
```

```
# Standard loopback interface address (localhost)
```

```
PORT = 65432
```

```
# Port to listen on (non-privileged ports are > 1023)
```

```
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
```

```
    s.bind((HOST, PORT))
```

```
    s.listen()
```

```
    conn, addr = s.accept()
```

```
    with conn:
```

```
        print('Connected by', addr)
```

```
        while True:
```

```
            data = conn.recv(1024)
```

```
            if not data:
```

```
                break
```

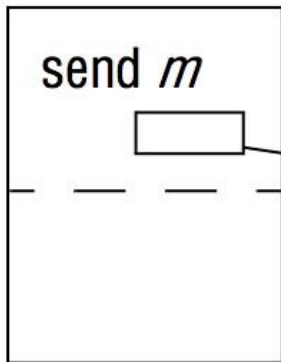
```
            conn.sendall(data)
```

AF_INET : connect via IP

SOCK_STREAM: TCP

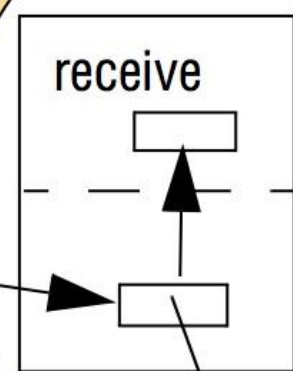
IPC Example 2: MPI

Process p



Message

Process q



MPI library buffer

MPI (~1990s)

- MPI is a widely used standard for writing message-passing programs
 - <http://www.mpi-forum.org>
 - It's a specification, not an implementation
- It is implemented as a library, not a programming language
 - Examples include MPICH, Open MPI, Microsoft MPI (MS-MPI), Intel MPI
 - MPI has been supported by C, C++, Fortran, Java, Python, etc.

MPI Process and Message Passing

- An MPI program consists of many processes
 - These processes are executed on a set of physical processors which exchange data (by internal bus or a network).
- The processes executing in parallel have separate **address spaces**.
 - Assume your program has a statement " $y = a + b$ ".
 - When process A and process B both execute the above statement, each process has its own set of variables $\{a, b, y\}$.
- Message-passing: a portion of one process's address space can be copied into another process's address space
 - "message" means "data"
 - "message-passing" means "data transfer"
 - It's usually done by **send** operation and **receive** operation

Example:

Matrix-Vector Multiplication

a_{00}	a_{01}	\cdots	$a_{0,n-1}$		y_0
a_{10}	a_{11}	\cdots	$a_{1,n-1}$	x_0	y_1
\vdots	\vdots		\vdots	x_1	\vdots
a_{i0}	a_{i1}	\cdots	$a_{i,n-1}$	\vdots	$y_i = a_{i0}x_0 + a_{i1}x_1 + \cdots a_{i,n-1}x_{n-1}$
\vdots	\vdots		\vdots	x_{n-1}	\vdots
$a_{m-1,0}$	$a_{m-1,1}$	\cdots	$a_{m-1,n-1}$		y_{m-1}

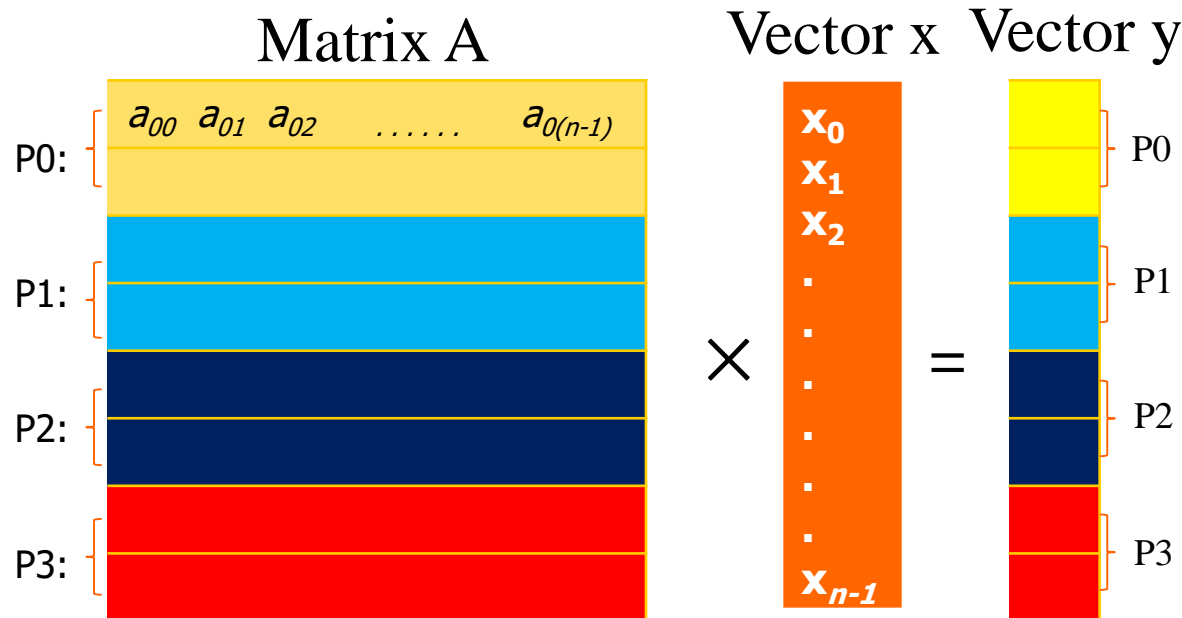
```

/* For each row of A */
for (i = 0; i < m; i++) {
    /* Form dot product of ith row with x */
    y[i] = 0.0;

    for (j = 0; j < n; j++)
        y[i] += A[i][j]*x[j];
}
  
```

Rowwise 1-D Partitioning

- Given p processes, Matrix A ($m \times n$) is partitioned into p smaller matrices, each with dimension ($m/p \times n$).
 - For simplicity, we assume p divides m (or, m is divisible by p).



Matrix-Vector Multiplication by MPI

- Assumptions
 - A total of p processes
 - Matrix A ($m \times n$) and vector x ($n \times 1$) are created at **process 0**
 - called “master process” because it coordinates the work of other processes (i.e., “slave processes”)
- 1. Message passing:
 - Process 0 will send $(p-1)$ sub-matrices to corresponding processes
 - Process 0 will send vector x to all other $p-1$ processes
- 2. Calculations:
 - Each process carries out its own matrix-vector multiplication
- 3. Message passing:
 - Processes 1 to $(p-1)$ send the results (i.e., part of vector y) back to process 0

Remote Invocation

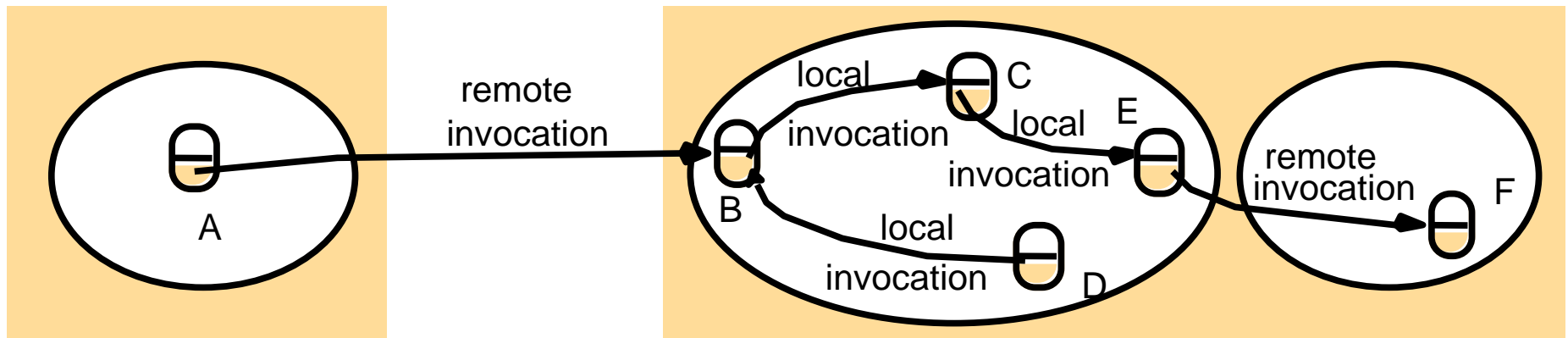
- Remote invocation is the most common communication paradigm in DSs.
 - Request-reply protocols: a pattern of two-way message exchange on top of message passing
 - Such as **HTTP**
 - Remote procedure call (RPC): a client program can call procedures in a server program transparently
 - Such as **Sun RPC**
 - Remote method invocation (RMI): extends the concept of RPC to object-oriented programming model
 - Such as **Java RMI (~2000s)**

Traditional Object Model

- An object consists of a set of data and a set of methods.
- An object-oriented program consists of a collection of interacting objects.
- An object communicates with other objects by invoking their methods, passing arguments and receiving results.
- Interfaces: an interface provides a definition of the signatures of a set of methods without specifying their implementations.
 - We say “an object provides a particular interface” if its class contains code that implements the methods of that interface.
- Actions: an action is initiated by an object invoking a method in another object.
 - As an invocation can lead to further invocations of methods in other objects, an action can be a chain of related invocations.

Distributed Objects

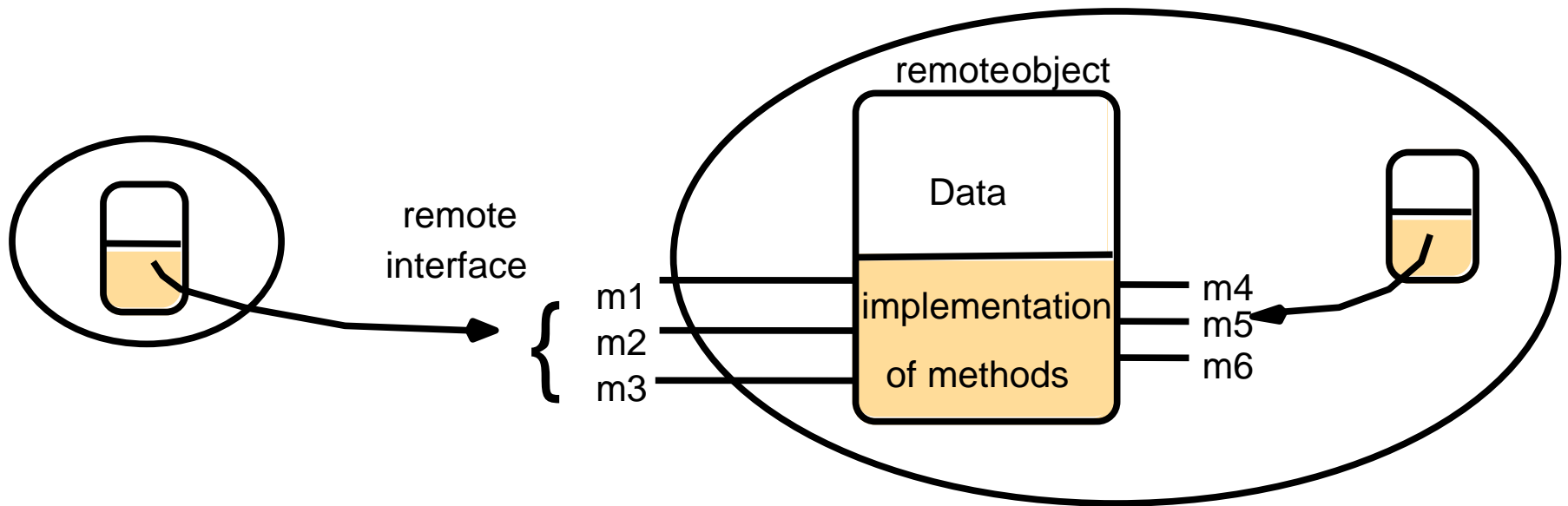
- In distributed systems, objects can be physically distributed into different processes or computers.
- Remote objects: objects that can receive remote invocations



REF1 Fig. 5.12: Remote and local invocations

Distributed Objects (Cont.)

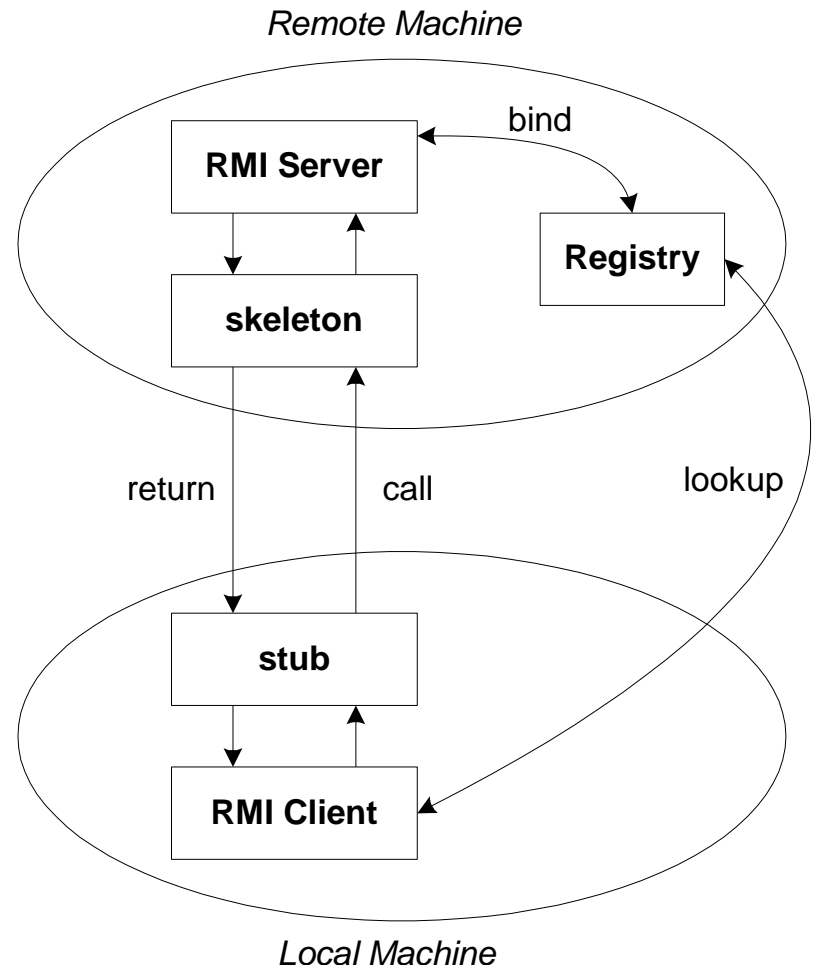
- Two fundamental concepts
 - **Remote object references**: an identifier that can be used to refer to a remote object. Object references can be passed as parameters.
 - **Remote interfaces**: every remote object has a remote interface that specifies which of its methods can be invoked remotely.



REF1 Fig. 5.13: A remote object and its remote interface

The General RMI Architecture

- The server must first bind its name to the registry
- The client looks up the server name in the registry to establish remote references.
- The Stub serializing the parameters to skeleton, the skeleton invoking the remote method and serializing the result back to the stub.



A Typical Java RMI Application

- A typical Java RMI application may include the following programs:
 - Interface program: define the remote interface
 - Servant program: define and implement the remote object
 - Server program: maintain the remote objects
 - Client program: look up remote object references and make remote invocations

Indirect Communication

- Both IPC and RMI are based on direct communication.
 - Direct coupling between the sender and receiver.
 - It may not well handle the scenario that clients or servers are temporally disconnected from the distributed system.
- Indirect communication: communication between entities in a DS through an intermediary with no direct coupling between the sender and the receiver(s). Two key properties are:
 - Space uncoupling: the sender doesn't know or need to know the identify of the receiver(s)
 - Time uncoupling: the send and receiver(s) can have independent lifetimes

Pros and Cons of Indirect Communication

- Advantages:
 - With space uncoupling, system developer has more degree of freedom in dealing with system changes such as failure, replacement, upgrade, migration of system participants (senders or receivers).
 - With time uncoupling, sender and receiver(s) don't need to exist at the same time to communicate. This is good for volatile environments where senders and receivers may come and go.
- Disadvantages:
 - Performance overhead by the added level of indirection
 - The system becomes more difficult to manage precisely

Publish-subscribe Systems

- A widely used indirect communication technique
 - A one-to-many indirect communication paradigm
- *Publishers* publish structured events to an event service.
- *Subscribers* express interest in particular events through subscriptions.
- The publish-subscribe system matches subscriptions against published events and ensure the correct delivery of event notifications.

Example:

A Dealing Room System

- A dealing room system: to allow dealers see the latest information about the market prices of the stocks they deal in.
 - Market prices come from many information providers.
 - A dealer is only interested in his own specialist stocks.
- The system can be implemented by two types of process:
 - Information provider process
 - Dealer process

Example:

A Dealing Room System

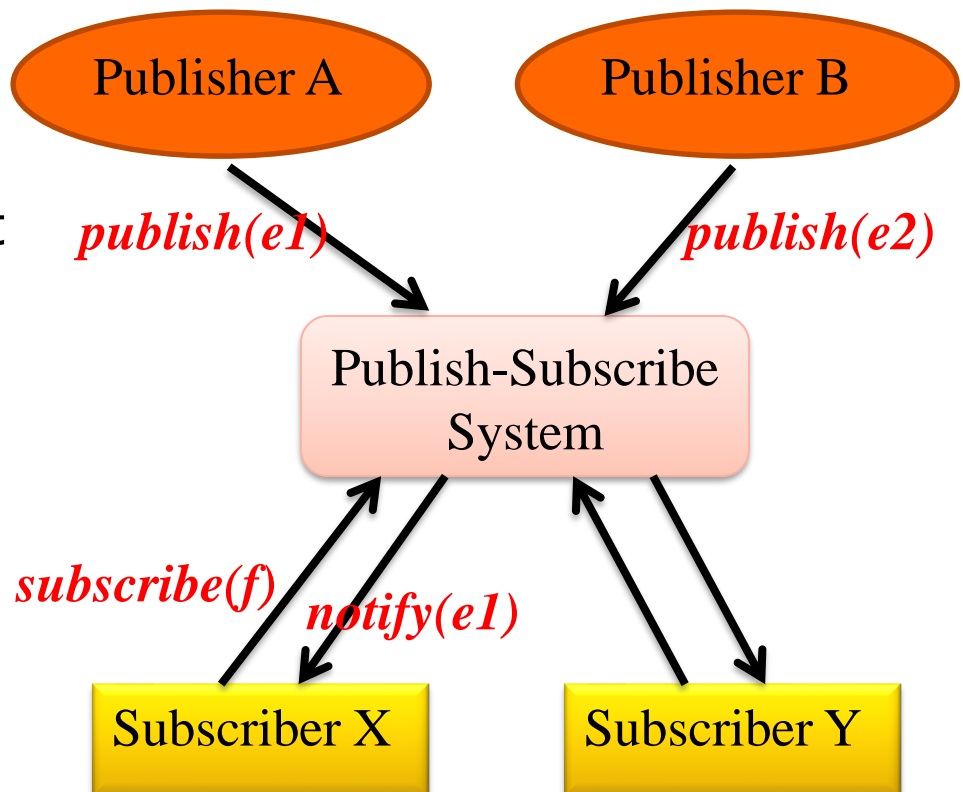
- Information provider process
 - Continuously receives new trading information
 - Each of the updates is regarded as an event.
 - Publishes such events to the publish-subscribe system for delivery
- Dealer process
 - Create a subscription representing each named stock that the dealer is interested in
 - Each subscription expresses an interest in events.
 - Receives all the information sent to it and displays it to the dealer.

Characteristics of Publish-Subscribe Systems

- Heterogeneity:
 - Heterogeneous entities in a distributed system can work together with event notifications as a means of communication.
- Asynchronicity:
 - Notifications are sent asynchronously by event-generating publishers to all subscribers

Programming Model of Publish-Subscribe Systems

- Publishers disseminate an event ***e*** through a ***publish(e)*** operation.
- Subscribers express an interest in a set of events through ***subscribe(f)*** operation, where ***f*** is a filter that express the subscriber's interest.
 - ***unsubscribe(f)*** is used to revoke this interest
- When events arrive at a subscriber, the events are delivered using ***notify(e)*** operation.



Subscription Model of Publish-Subscribe Systems

- How to express a subscriber's interest?
 - **Channel-based**: publishers publish events to named channels, and subscribers subscribe to one of the named channels to receive all events in that channel
 - **Topic-based** (or subject-based): each notification includes a field that denotes the "topic"; and subscriptions are defined by the "topic" of interest. A "topic" can be hierarchical, such as "sport/basketball/NBA", "sport/soccer/PremierLeague".
 - **Content-based**: each notification has many attributes; a content-based filter is a query defined in terms of compositions of constraints over the values of event attributes.
 - **Type-based**: linked with object-based approaches where objects have a specified type. Subscriptions are defined in terms of types of events and matching is defined in terms of types or subtypes of the given filter.

Service-Oriented Architecture

- SOA is a loosely-coupled architecture that component works together by service provision.
 - Loosely-coupled: replaceable, upgradable
 - Service provision: can be internal or external
- This implies open and standards-based interoperability.
- Popular Implementations:
 - SOAP (~2000s)
 - RESTful (~2000s)

Some SOA Principles

- **Standardized Service Contract**
 - Service within the same service inventory are in compliance with the same contract design standards.
- **Service Loose Coupling**
 - Service contracts impose low consumer coupling requirements and are themselves decoupled from their surrounding environments.
- **Service Abstraction**
 - Service contracts only contain essential information and information about service is limited to what is published in service contracts
- **Service Reusability**
 - Services contain and express agnostic logic and can be positioned as reusable enterprise resources.

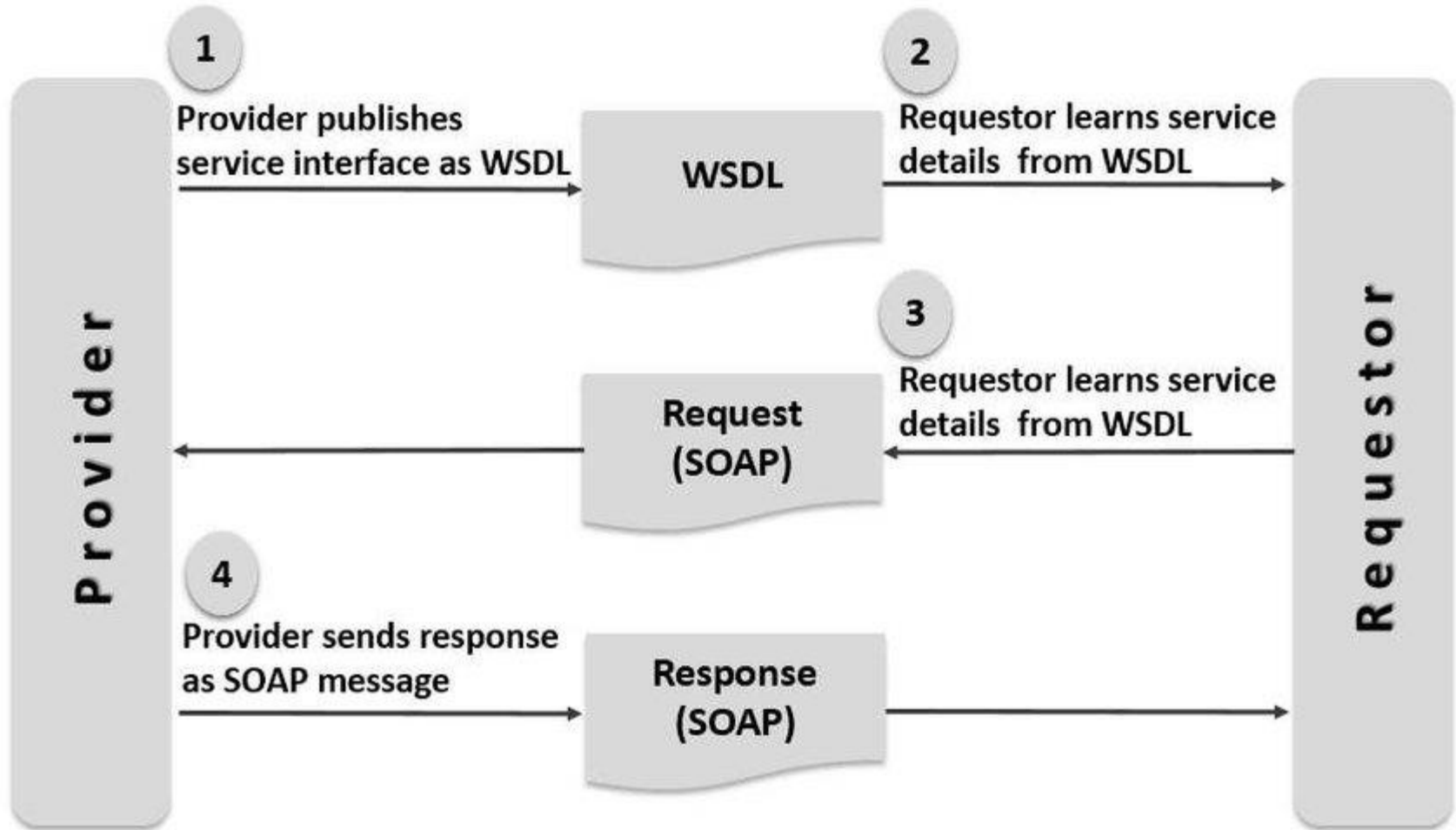
Some SOA Principles (con't)

- Service Autonomy
 - Services exercise a high level of control over their underlying runtime execution environment
- Service Statelessness
 - Service minimize resource consumption by deferring the management of state information when necessary.
- Service Discoverability
 - Services are supplemented with communicative meta data by which they can be effectively discovered and interpreted
- Service Composability
 - Services are effective composition participants, regardless of the size and complexity of the composition.

SOAP

- Simple Object Access Protocol is a message exchange protocol that implements SOA.
- Assume the use of WSDL (XML), commonly over HTTP.
- SOAP allows
 - Over internet
 - Cross operating systems (Win/Mac/Linux)
 - Cross language (C++/Java/Python)
- It is a W3C standard

SOAP



SOAP Message Structure

```
<?xml version = "1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV = "http://www.w3.org/2001/12/soap-envelope"
  SOAP-ENV:encodingStyle = "http://www.w3.org/2001/12/soap-encoding">

  <SOAP-ENV:Header>
    ...
    ...
  </SOAP-ENV:Header>
  <SOAP-ENV:Body>
    ...
    ...
    <SOAP-ENV:Fault>
      ...
      ...
    </SOAP-ENV:Fault>
    ...
  </SOAP-ENV:Body>
</SOAP_ENV:Envelope>
```

3 parts: Envelope, Header, Body

SOAP Message

<?xml version = "1.0"?>

<SOAP-ENV:Envelope>

.....

<SOAP-ENV:Body>

<m:GetQuotationResponse xmlns:m =
"http://www.tp.com/Quotation">

<m:Quotation>This is Qutation</m:Quotation>

</m:GetQuotationResponse>

</SOAP-ENV:Body>

</SOAP-ENV:Envelope>

Another Example

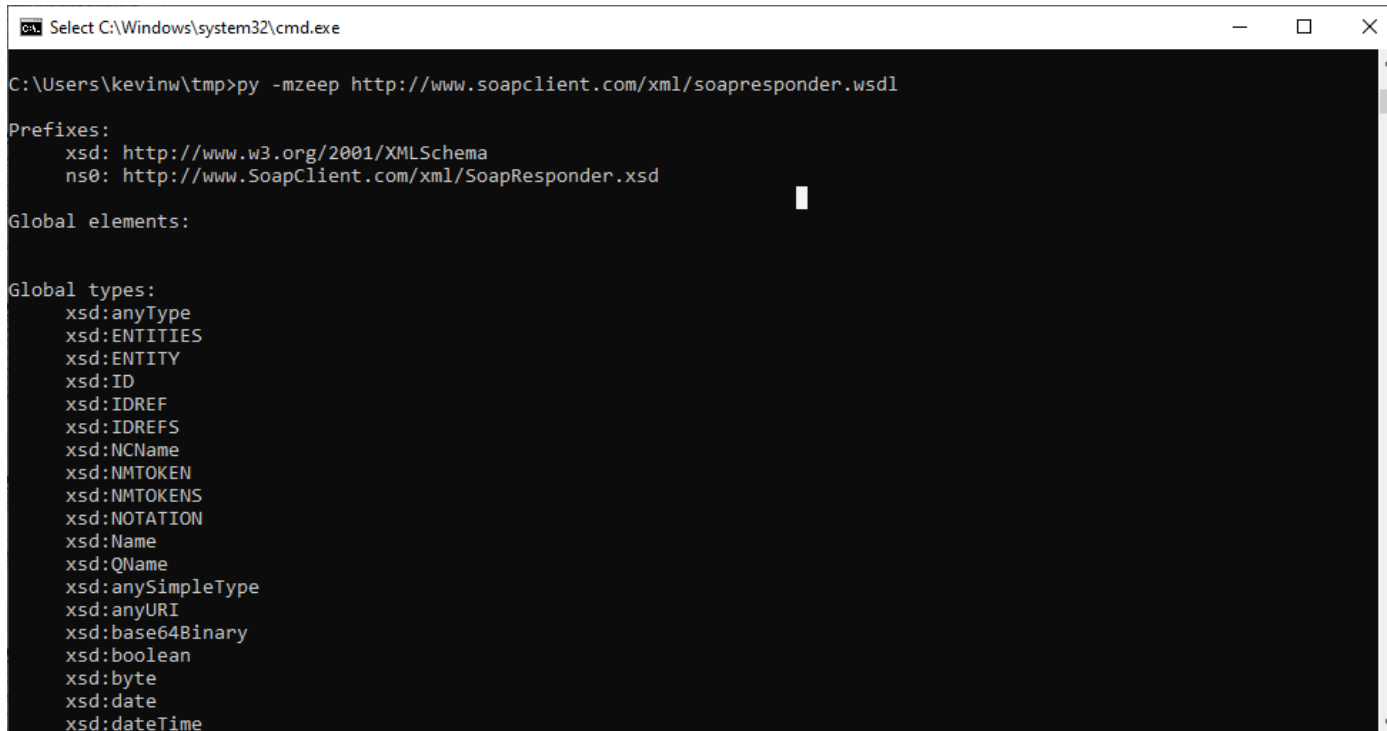
← → ↺ ⓘ Not secure | soapclient.com/xml/soapresponder.wsdl

This XML file does not appear to have any style information associated with it. The document tree is shown below.

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns:tns="http://www.SoapClient.com/xml/SoapResponder.wsdl" xmlns:xsd1="http://www.SoapClient.com/xml/SoapResponder.xsd" xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:soap="http://schemas.xmlsoap.org/soap/" name="SoapResponder" targetNamespace="http://www.SoapClient.com/xml/SoapResponder.wsdl">
  <types>
    <schema xmlns="http://www.w3.org/1999/XMLSchema" targetNamespace="http://www.SoapClient.com/xml/SoapResponder.xsd"> </schema>
  </types>
  <message name="Method1">
    <part name="bstrParam1" type="xsd:string"/>
    <part name="bstrParam2" type="xsd:string"/>
  </message>
  <message name="Method1Response">
    <part name="bstrReturn" type="xsd:string"/>
  </message>
  <portType name="SoapResponderPortType">
    <operation name="Method1" parameterOrder="bstrparam1 bstrparam2 return">
      <input message="tns:Method1"/>
      <output message="tns:Method1Response"/>
    </operation>
  </portType>
  <binding name="SoapResponderBinding" type="tns:SoapResponderPortType">
    <soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http"/>
    <operation name="Method1">
      <soap:operation soapAction="http://www.SoapClient.com/SoapObject"/>
      <input>
        <soap:body use="encoded" namespace="http://www.SoapClient.com/xml/SoapResponder.xsd" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </input>
      <output>
        <soap:body use="encoded" namespace="http://www.SoapClient.com/xml/SoapResponder.xsd" encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
      </output>
    </operation>
  </binding>
  <service name="SoapResponder">
    <documentation>
      A SOAP service that echoes input parameters in the response
    </documentation>
    <port name="SoapResponderPortType" binding="tns:SoapResponderBinding">
      <soap:address location="http://www.soapclient.com/xml/soapresponder.wsdl"/>
    </port>
  </service>
</definitions>
```

Python Implementation on SOAP

- Using the package zeep
- Read a WSDL document by
—python -mzeep <URL_OF_WSDL>



```

C:\Users\kevinw\tmp>py -mzeep http://www.soapclient.com/xml/soapresponder.wsd
Prefixes:
  xsd: http://www.w3.org/2001/XMLSchema
  ns0: http://www.SoapClient.com/xml/SoapResponder.xsd

Global elements:

Global types:
  xsd:anyType
  xsd:ENTITIES
  xsd:ENTITY
  xsd:ID
  xsd:IDREF
  xsd:IDREFS
  xsd:NCName
  xsd:NMTOKEN
  xsd:NMTOKENS
  xsd:NOTATION
  xsd:Name
  xsd:QName
  xsd:anySimpleType
  xsd:anyURI
  xsd:base64Binary
  xsd:boolean
  xsd:byte
  xsd:date
  xsd:dateTime

```


Con't

```
C:\Users\kevinw\tmp>more soap.py
import zeep

wsdl = 'http://www.soapclient.com/xml/soapresponder.wsdl'
client = zeep.Client(wsdl=wsdl)
print(client.service.Method1('Zeep', 'is cool'))

C:\Users\kevinw\tmp>py soap.py
Your input parameters are Zeep and is cool

C:\Users\kevinw\tmp>
```

RESTful API

- REST is an acronym for Representational State Transfer.
- PhD dissertation of Roy Fielding in 2000
- An architectural style for distributed system in implementing SOA.
- An API that satisfy REST specification is called a RESTful API
- May work with different platforms, different languages, different data format (XML, JSON)

RESTful Idea

- The idea of a RESTful service is that usually service involves operations like CRUD:
 - Create Data
 - Read Data
 - Update Data
 - Delete Data
- In HTTP protocol, it defines the functions
 - GET
 - POST
 - PUT
- Add a new command DELETE

RESTful Idea

- Access a resource via a URL
 - <https://resource.pro/devices/list> all devices list
 - <https://resource.pro/devices/1331/> device id 1331
 - To get the resource, use HTTP GET; to update use POST or PUT; to delete use DELETE
- e.g. GET /devices/list get all devices list
- e.g. DELETE /devices/1331 delete device id 1331

RESTful

- **Client–server** – By separating the user interface concerns from the data storage concerns, we improve the portability of the user interface across multiple platforms and improve scalability by simplifying the server components.
- **Stateless** – Each request from client to server must contain all of the information necessary to understand the request, and cannot take advantage of any stored context on the server. Session state is therefore kept entirely on the client.
- **Cacheable** – Cache constraints require that the data within a response to a request be implicitly or explicitly labeled as cacheable or non-cacheable. If a response is cacheable, then a client cache is given the right to reuse that response data for later, equivalent requests.
- **Uniform interface** – By applying the software engineering principle of generality to the component interface, the overall system architecture is simplified and the visibility of interactions is improved. In order to obtain a uniform interface, multiple architectural constraints are needed to guide the behavior of components. REST is defined by four interface constraints: identification of resources; manipulation of resources through representations; self-descriptive messages; and, hypermedia as the engine of application state.
- **Layered system** – The layered system style allows an architecture to be composed of hierarchical layers by constraining component behavior such that each component cannot “see” beyond the immediate layer with which they are interacting.
- **Code on demand (optional)** – REST allows client functionality to be extended by downloading and executing code in the form of applets or scripts. This simplifies clients by reducing the number of features required to be pre-implemented.

RESTful Example

- An RESTful API example that returns JSON



img; <https://uchi.kz/zapis-v-bloge/patch-vs-post-rest-api>

RESTful Python Implementation

C:\Windows\system32\cmd.exe - py rest.py

```
C:\Users\kevinw\tmp>more rest.py
from flask import Flask, jsonify

app = Flask(__name__)

tasks = [
    {
        'id': 1,
        'title': u'Buy groceries',
        'description': u'Milk, Cheese, Pizza, Fruit, Tylenol',
        'done': False
    },
    {
        'id': 2,
        'title': u'Learn Python',
        'description': u'Need to find a good Python tutorial on the web',
        'done': False
    }
]

@app.route('/todo/api/v1.0/tasks', methods=['GET'])
def get_tasks():
    return jsonify({'tasks': tasks})

if __name__ == '__main__':
    app.run(debug=True)

C:\Users\kevinw\tmp>py rest.py
* Serving Flask app "rest" (lazy loading)
```

Con't

localhost:5000/todo/api/v1.0/tasks

```
{
  "tasks": [
    {
      "description": "Milk, Cheese, Pizza, Fruit, Tylenol",
      "done": false,
      "id": 1,
      "title": "Buy groceries"
    },
    {
      "description": "Need to find a good Python tutorial on the web",
      "done": false,
      "id": 2,
      "title": "Learn Python"
    }
  ]
}
```

C:\Windows\system32\cmd.exe

in.</p>

C:\Users\kevinw>curl -i localhost:5000/todo/api/v1.0/tasks

HTTP/1.0 200 OK

Content-Type: application/json

Content-Length: 317

Server: Werkzeug/1.0.0 Python/3.8.1

Date: Tue, 11 Feb 2020 12:00:20 GMT

```
{
  "tasks": [
    {
      "description": "Milk, Cheese, Pizza, Fruit, Tylenol",
      "done": false,
      "id": 1,
      "title": "Buy groceries"
    },
    {
      "description": "Need to find a good Python tutorial on the web",
      "done": false,
      "id": 2,
      "title": "Learn Python"
    }
  ]
}
```


More Reading Materials

- MPI:
 - <https://computing.llnl.gov/tutorials/mpi/>
- Java RMI:
 - <https://docs.oracle.com/javase/tutorial/rmi/>
- Publish-Subscribe System:
 - Y. Liu and B. Plate, "Survey of Publish Subscribe Event Systems," TR574, Indiana University.
 - <ftp://www.cs.indiana.edu/pub/techreports/TR574.pdf>
- SOA:
 - <https://docs.microsoft.com/en-us/dotnet/architecture/microservices/architect-microservice-container-applications/service-oriented-architecture>
 - <https://patterns.arcitura.com/>
- RESTful:
 - <https://restfulapi.net/>