Distributed Computer Systems Engineering

CIS 508: Lecture 2
Architecture

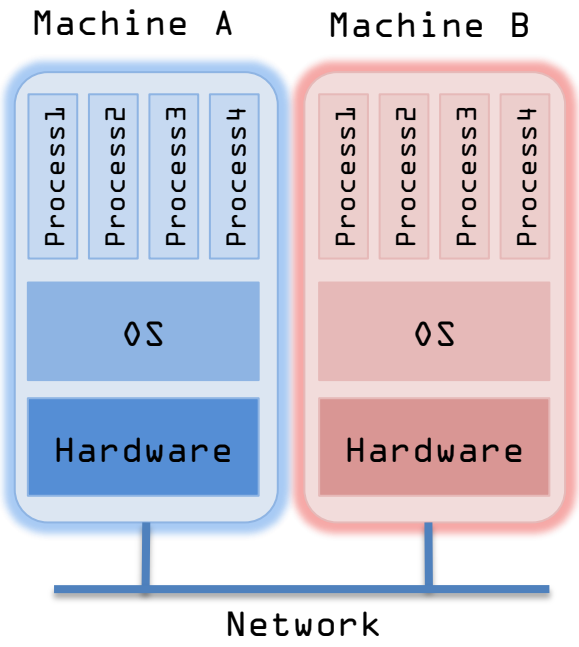
Lecturer: Sid C.K. Chau
Email: ckchau@masdar.ac.ae

Masdar INSTITUTE CIS

Part I

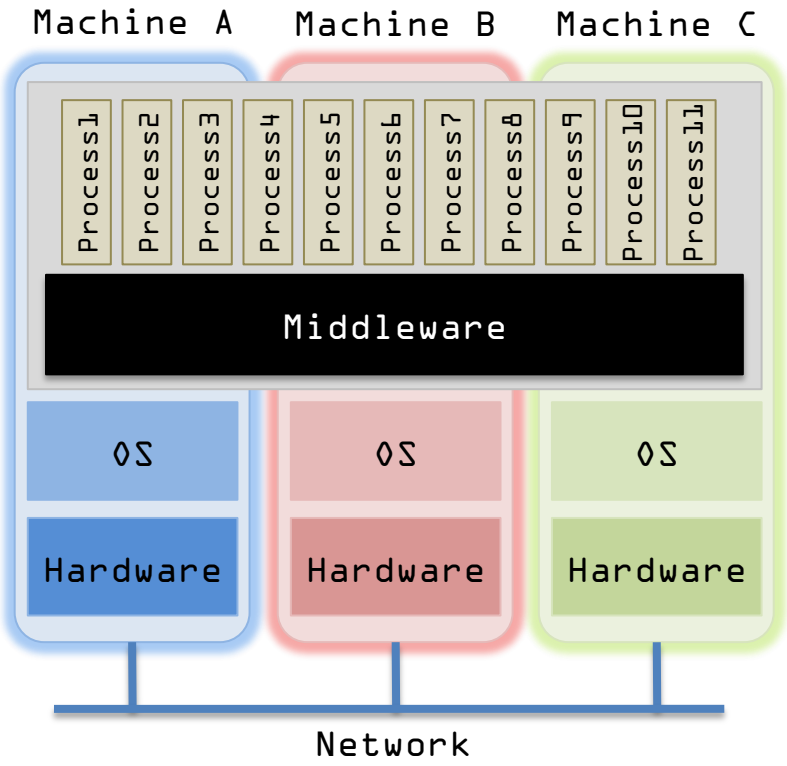
Systems

What is a Computer System?



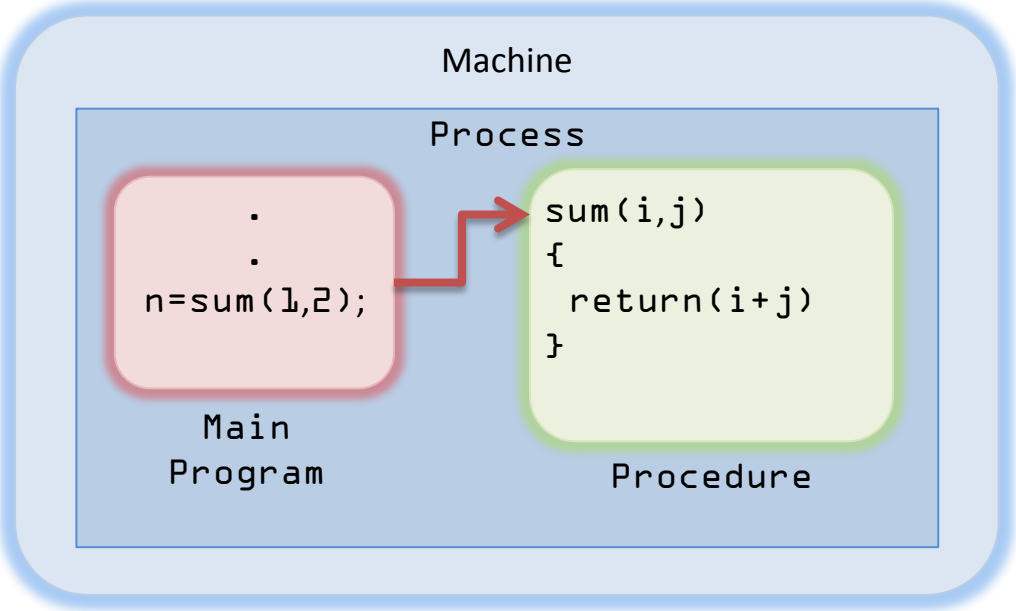
- Process (as a Computer System)
  - Minimal unit of an autonomous computer system, an instance of computer program execution
  - Each has independent (or apparently independent) access to its computational resources (CPU, memory, storage) from other processes
  - Operating system provides separation among simultaneous processes over hardware
  - Processes can communicate with other within the same machine, or across different machines (over a communication network)
- Thread vs. Process
  - Thread may not have independent access to computational resources
  - Threads are aware of simultaneous access themselves

Processes over Middleware



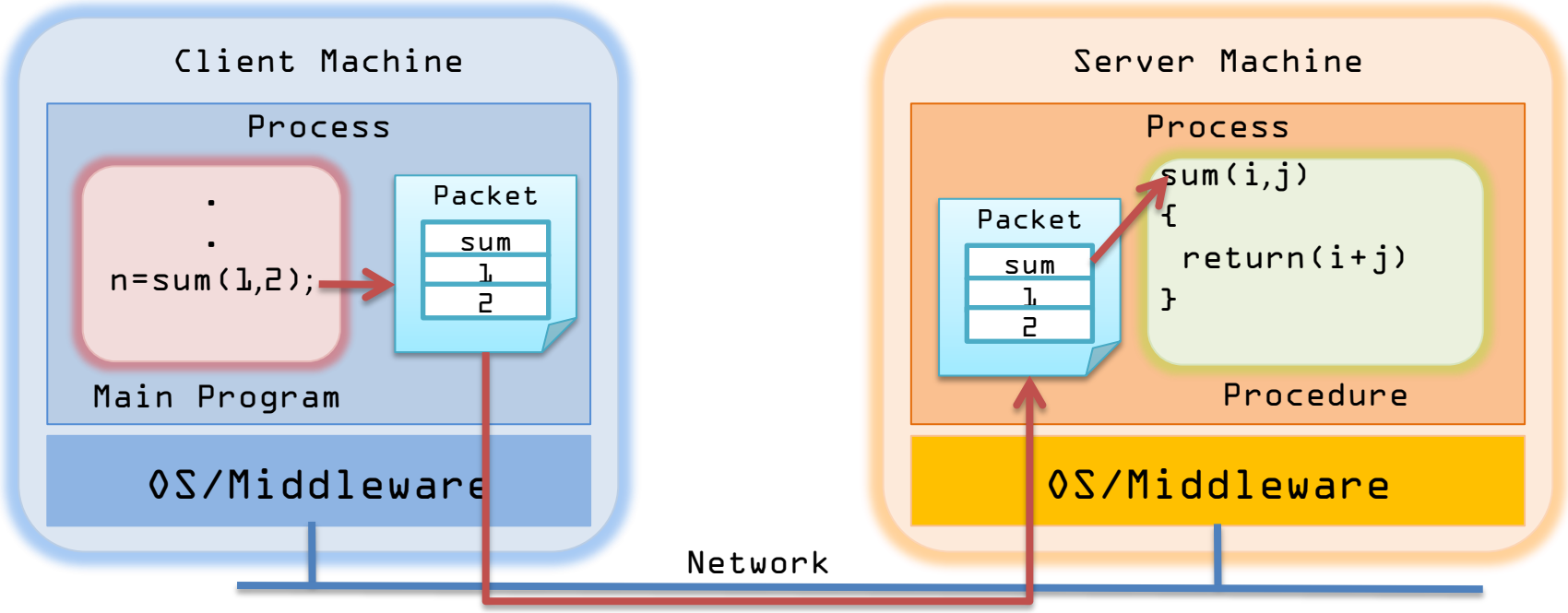
- About middleware
  - Processes may span multiple machines
  - Middleware provides transparency to hardware resources
  - Enable process migration across machines
- Advantages
  - Dynamical and scalable resource allocation
  - Improving reliability and availability
  - Bridging heterogeneous physical resources
- Disadvantages
  - Expensive in cost, high overhead
  - Slow in performance
  - Proprietary middleware lock-in

Conventional Procedure Call



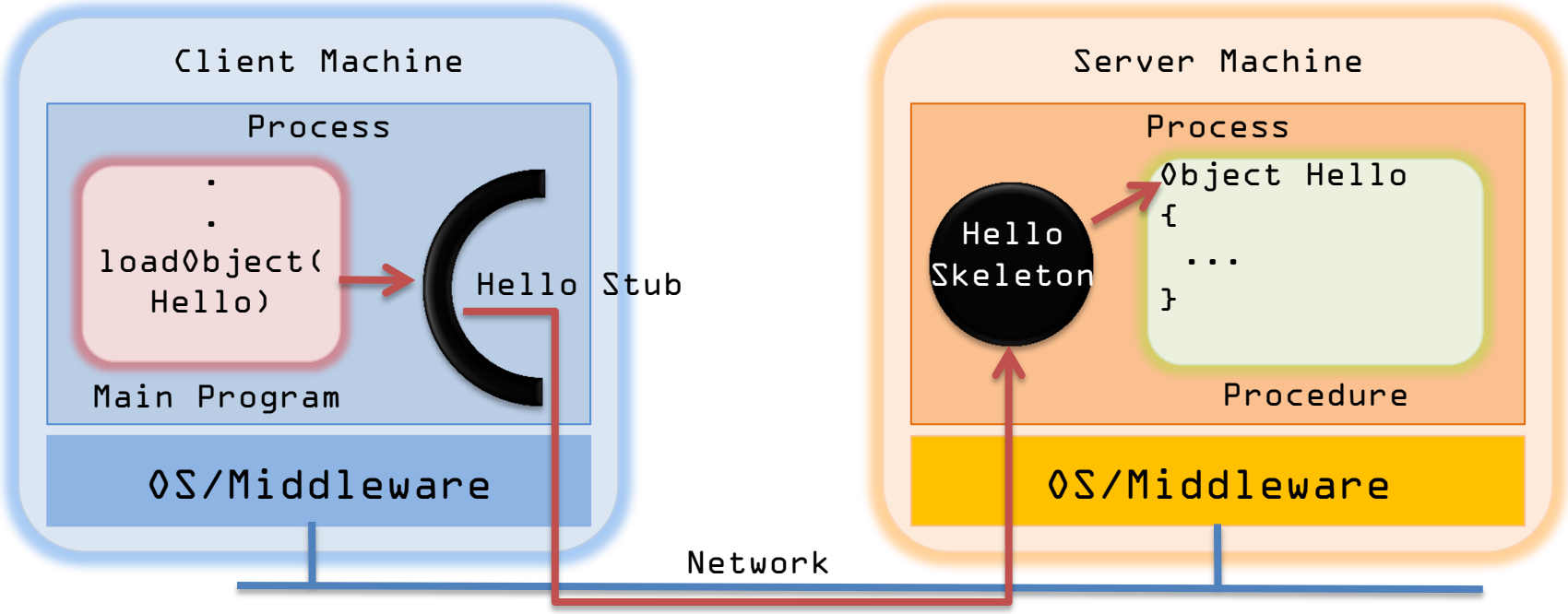
- Calling a procedure within a local machine
  - Storing the parameters in a memory stack before the call to invoke
  - The procedure is executed and the return value is stored in memory call
  - The main program is continued, copying the return value from memory

Remote Procedure Call



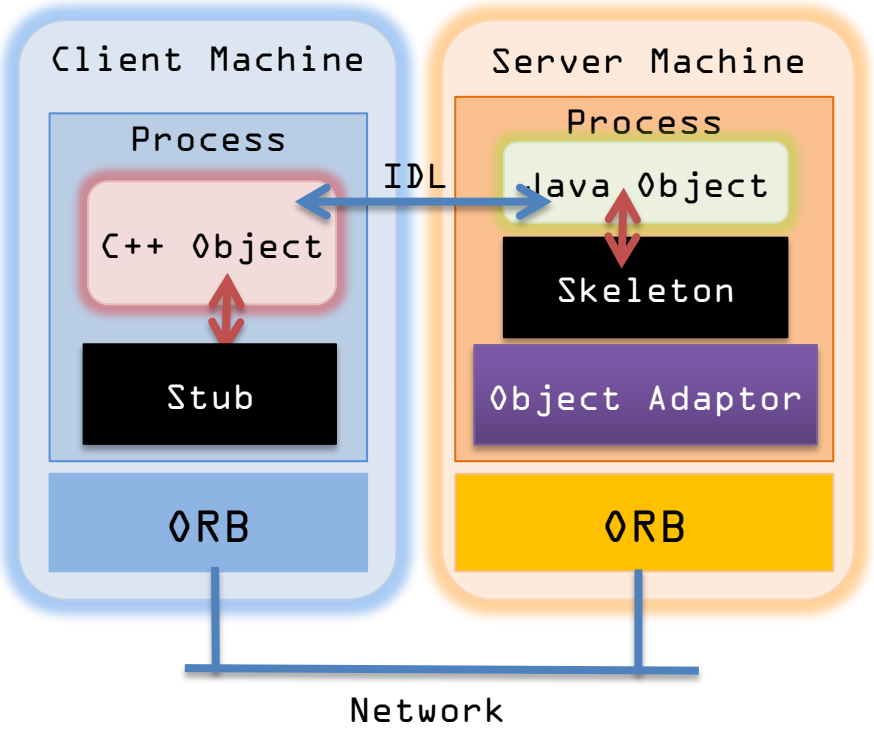
- Calling a remote procedure in another machine
  - Packaging machine ID, procedure ID, parameters as a packet
  - Packet is sent via network to reach the targeted machine
  - OS/Middleware responsible for inter-communication and interoperation

Distributed Objects



- Distributed objects are object-oriented modules
  - More programmable than procedure calls
  - Support persistent state, replication, and object migration
  - Examples: CORBA, DCOM, Java RMI

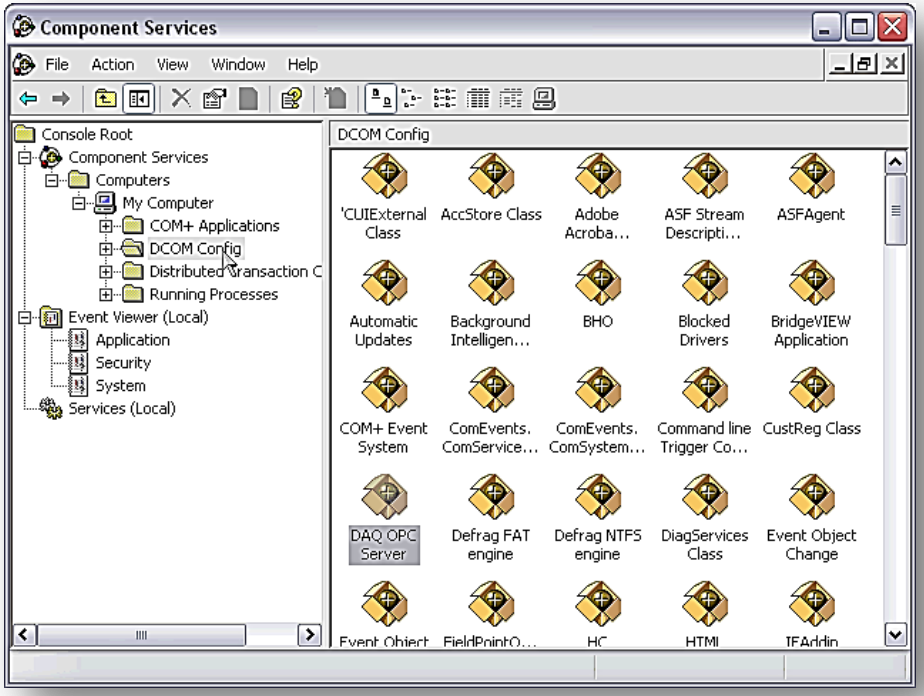
CORBA



- Common Object Request Broker Architecture (CORBA)
- A standard of distributed objects defined by the Object Management Group (OMG)
- Interoperable with C++, Java, Python
  - Via interface definition language (IDL)
  - Describes interface
  - Language independent
  - Client and server platform independent

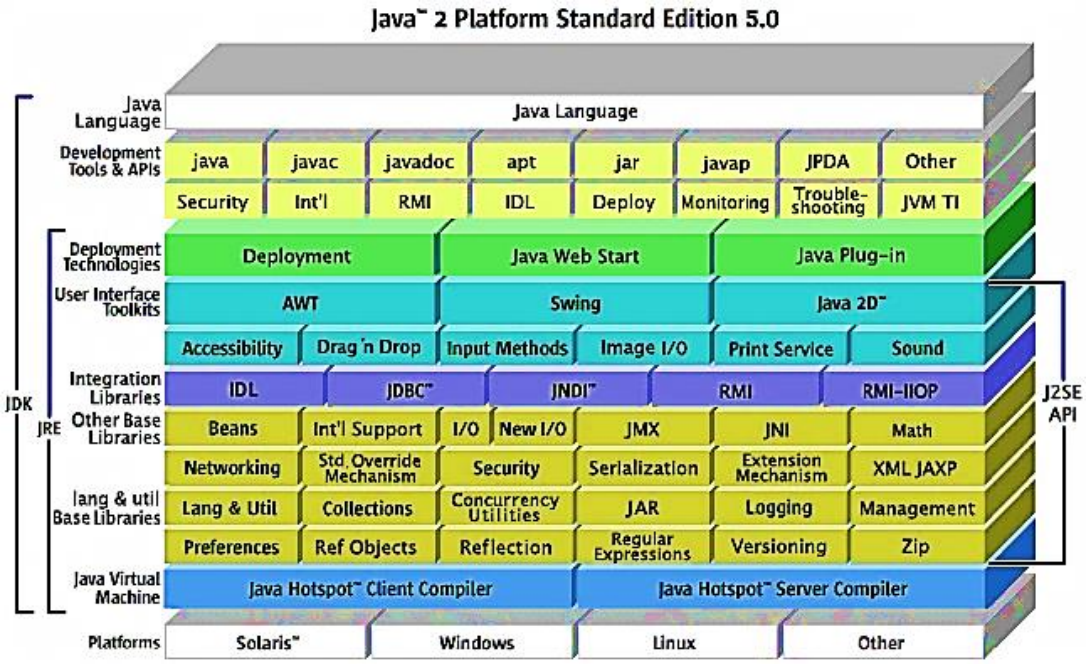
Microsoft DCOM

- Distributed Component Object Model (DCOM)
- Based on Component Object Model (COM)
- Interoperability
  - Different applications, Microsoft platforms, languages
- Versioning
  - Compatibility between new version of server and old versions of clients
  - New interfaces should preserve the old interface
- Naming: Use Globally unique identifiers



Java RMI

1. Define remote interface
    1. Extend `java.rmi.Remote`
2. Create server code
    1. Implements interface
    2. Creates security manager, registers with registry
3. Create client code
    1. Define object as instance of interface
    2. Lookup object in registry
    3. Call object
4. Compile and run
    1. Run `rmic` on compiled classes to create stubs
    2. Start registry
    3. Run server then client



Example: Java RMI

```
import java.rmi.Remote;
import java.rmi.RemoteException;

public interface Hello extends Remote {

          String sayHello() throws RemoteException;

}
```

Interface

```
import java.rmi.Naming;
import java.rmi.RemoteException;

public class HelloClient {
public static void main(String args[]) {
    String message = "blank";
    Hello obj = null;

    try {  obj = (Hello)Naming.lookup("//myhost/HelloServer");

             message = obj.sayHello();

               System.out.println(message);

        } catch (Exception e) {
            System.out.println("HelloClient exception:" +
               e.getMessage());
            e.printStackTrace();
}    }   }
```

Client

Example: Java RMI

```
import java.rmi.Naming;
import java.rmi.RemoteException;
import java.rmi.RMISecurityManager;
import java.rmi.server.UnicastRemoteObject;

public class HelloServer extends UnicastRemoteObject implements Hello {

    public HelloServer() throws RemoteException { super(); }

    public String sayHello() { return "Hello World!"; }

    public static void main(String args[]) {
          if (System.getSecurityManager() == null) {
              System.setSecurityManager(new RMISecurityManager()); }
          try { HelloServer obj = new HelloServer();
                 Naming.rebind("//myhost/HelloServer", obj);
                 System.out.println("HelloServer bound in registry");
          } catch (Exception e) {
              System.out.println("HelloServer err: " + e.getMessage());
              e.printStackTrace();
} } }
```

Server

Compile and Run

```
•   javac Hello.java HelloServer.java
    HelloClient.java
•   rmic -d `pwd` HelloServer
•   rmiregistry &   # not in same directory
•   java -Djava.rmi.server.codebase=file:///`pwd`/
    -Djava.security.policy=opensocket HelloServer
    -opensocket: grant { permission
    java.net.SocketPermission "*", "connect"; };
•   java HelloClient
```

Web Services

- *Web services*
  - XML based (not binary) interoperation scheme for distributed systems
  - Defined by W3C
- *Web Services Description Language (WSDL)*
  - XML-based language that is used for describing the functionality offered by a Web service
  - Like IDL in CORBA
- *Simple Object Access Protocol (SOAP)*
  - Protocol specification for exchanging structured information in the implementation of Web services

SOAP

- Client (like web browser):
  - Generate http calls
  - Data represented in XML
  - Listen for response
- Server (like web server):
  - Listen for HTTP
  - Bind to procedure
  - Respond with HTTP

SOAP Call

```
<soap:Envelope>
        <soap:Body>

        <xmlns:m="http://www.stock.org/" />
                <m:GetStockPrice>

        <m:StockName>Google</m:StockName>
                </m:GetStockPrice>
        </soap:Body>
</soap:Envelope>
```

SOAP Response

```
<soap:Envelope>
        <soap:Body>

        <xmlns:m="http://www.stock.org/" />
                <m:GetStockPriceResponse>

        <m:Price>44.5</m:Price>
                </m:GetStockPriceResponse>
        </soap:Body>
</soap:Envelope>
```

REST

- *REST (or ReST)*
  - Representational State Transfer
  - Use simple HTTP
  - REST uses HTTP for all four CRUD (Create/Read/Update/Delete) operations
  - Lightweight alternative to mechanisms like RPC (Remote Procedure Calls) and Web Services (SOAP, WSDL, et al.)
  - Supported by standard library features in languages like Perl, Java, or C#

REST Examples

REST Request1

```
GET /
Accept: application/json+userdb
```

REST Response1

```
200 OK
Content-Type: application/json+userdb
{
    "version": "1.0",
    "links": [
        {
            "href": "/user",
            "rel": "list",
            "method": "GET"
        },
        {
            "href": "/user",
            "rel": "create",
            "method": "POST"
        }
    ]
}
```

REST Request2

```
GET /user
Accept: application/json+userdb
```

REST Response2

```
200 OK
Content-Type: application/json+userdb
{
    "users": [
        {
            "id": 1,
            "name": "Sam",
            "country: "UAE",
            "links": [
                {
                    "href": "/user/1",
                    "rel": "self",
                    "method": "GET"
                },
                {
                    "href": "/user/1",
                    "rel": "edit",
                    "method": "PUT"
                },
                {
                    "href": "/user/1",
                    "rel": "delete",
                    "method": "DELETE"
                }
            ]
        },
    "links": [
        {
            "href": "/user",
            "rel": "create",
            "method": "POST"
        }
    ]
}
```

REST Examples

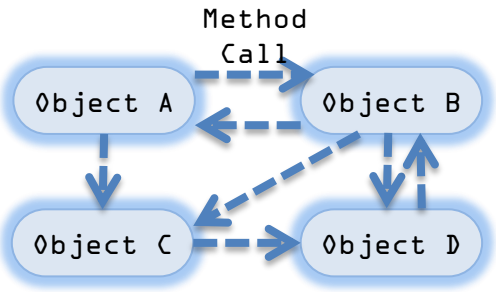
REST Request3

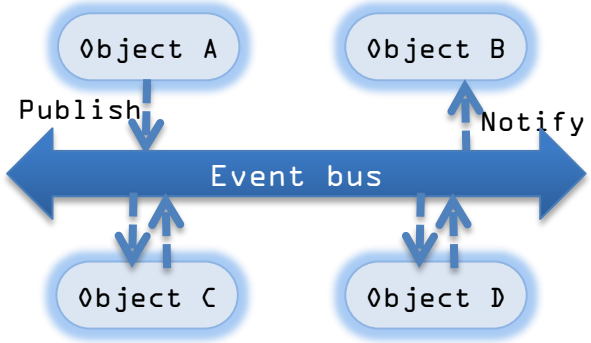
```
POST /user
Accept: application/json+userdb
Content-Type: application/json+userdb
{
    "name": "Dave",
    "country": "USA"
}
```

REST Response3

```
201 Created
Content-Type: application/json+userdb
{
    "user": {
        "id": 3,
        "name": "Karl",
        "country": "Austria",
        "links": [
            {
                "href": "/user/3",
                "rel": "self",
                "method": "GET"
            },
            {
                "href": "/user/3",
                "rel": "edit",
                "method": "PUT"
            },
            {
                "href": "/user/3",
                "rel": "delete",
                "method": "DELETE"
            }
        ]
    },
    "links": {
       "href": "/user",
       "rel": "list",
       "method": "GET"
    }
}
```

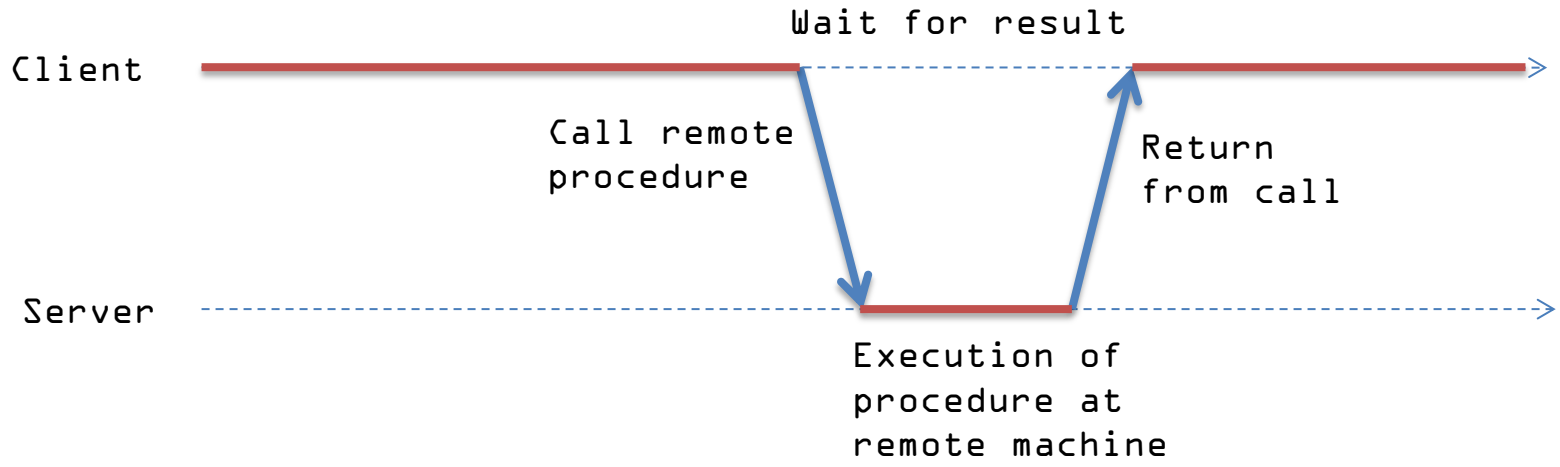
Modes of Calling





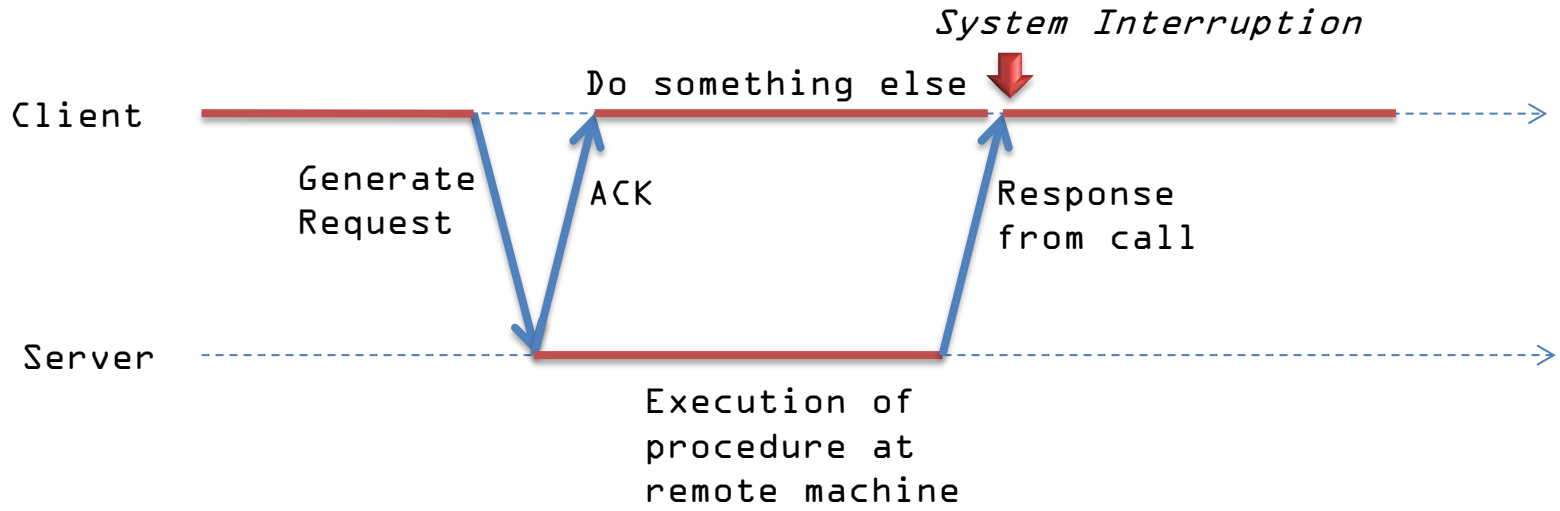
- Method Call / Remote Procedure Call (RPC)
  - Synchronous communication
  - Caller is blocked until callee returns
  - Easily creating deadlocks
- Messaging / Event based Communication
  - Asynchronous communication
  - Invokers can proceed to other tasks, until waiting for replies
  - There can be arbitrary long latency between the message arrivals
  - Invokers may not be aware that a target is unavailable or crashed
  - Explicitly timeout mechanism is required

Synchronous Call



- Synchronous procedure call
  - When the client calls the server
  - Client is blocked and has to wait for the return of result until server finishes execution
  - If server does not return (crashed), client may wait forever (deadlock)

Asynchronous Call



- Asynchronous message call
  - When the client invokes the server
  - Client only waits for acknowledge of receipt from server
  - When server returns the response, the client system interrupts

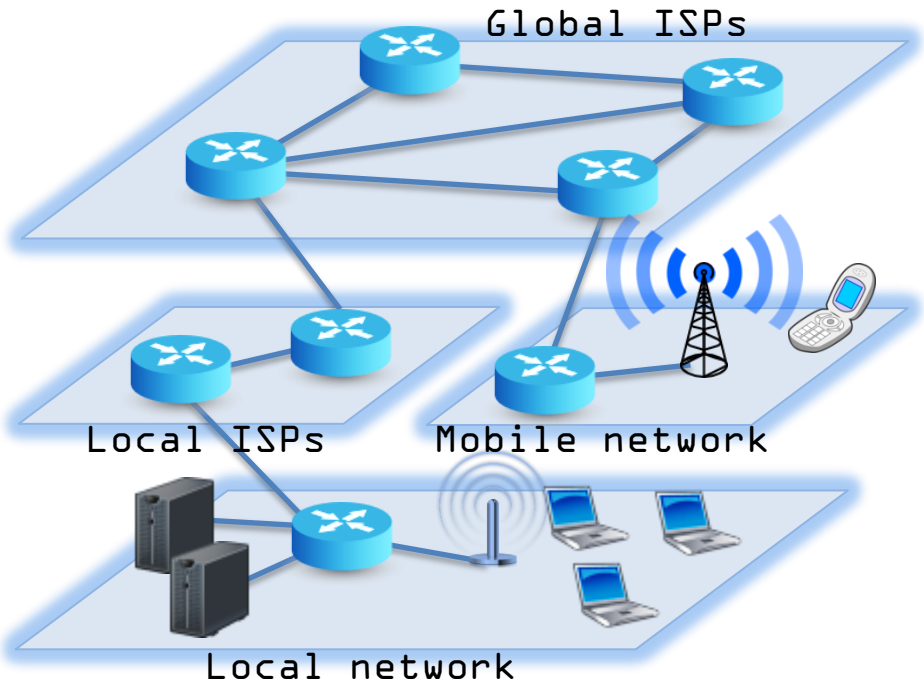
Ensure Transparency to Systems

Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource may be shared by several competitive users
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource
Persistence	Hide whether a (software) resource is in memory or on disk

Part II

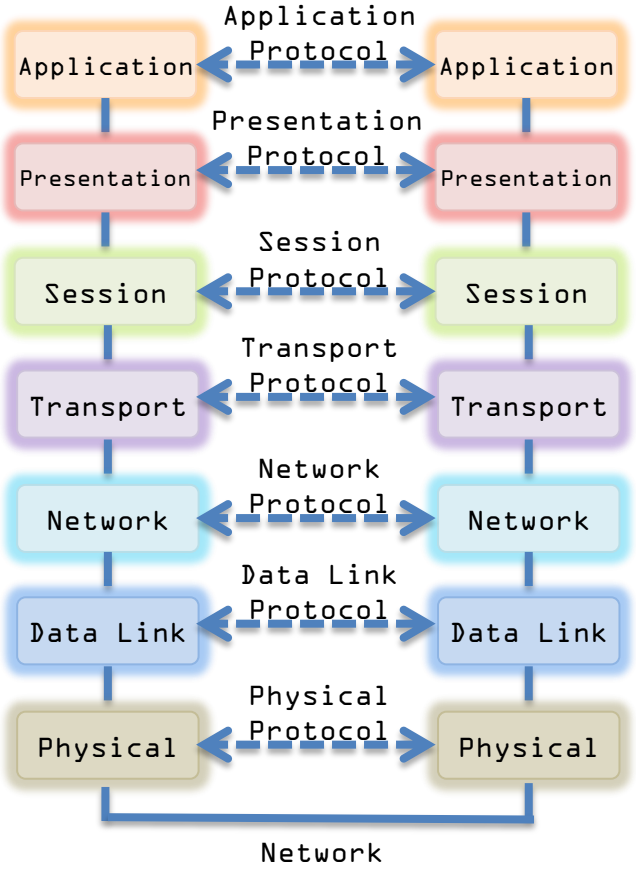
Networks

Networks of Computers



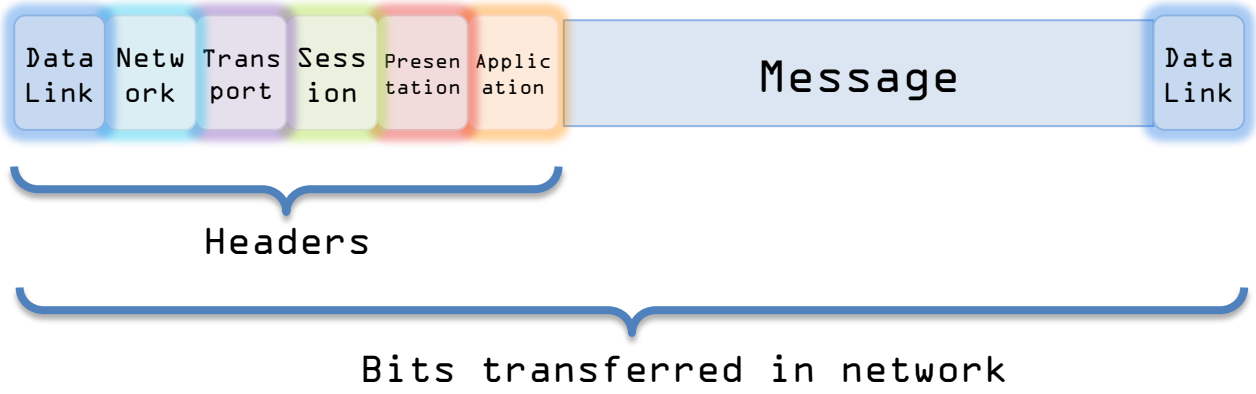
- Network core
  - Interconnected global ISPs
  - Well connected topology
  - High bandwidth capacity
- Network edge
  - Local ISPs
  - Local networks in homes, institutes
  - Mobile networks
  - Usually spoke, hub topology
  - Lower bandwidth capacity
    - Last-mile congestion
    - Throttling

Layers of Communication Stack



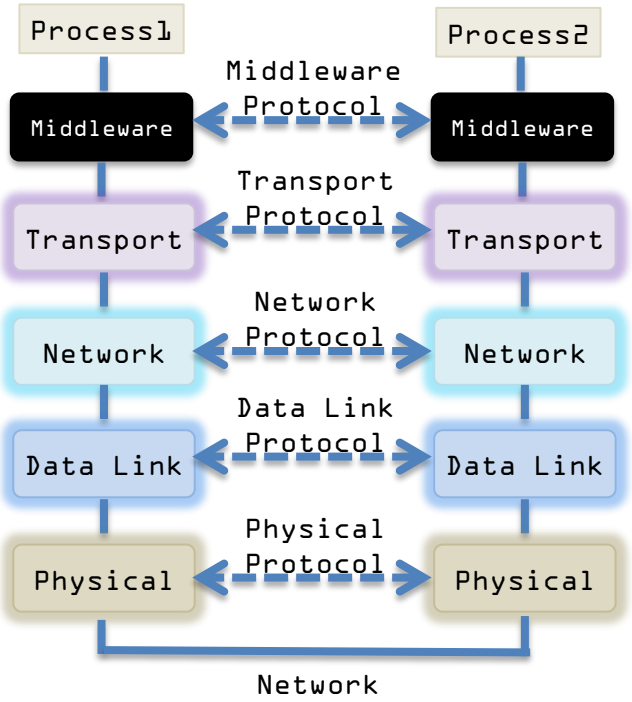
- **Application layer:** Supporting network applications, e.g. FTP, SMTP, HTTP, RPC (binary)
- **Presentation layer:** Allowing proper interpretation of data, e.g., encryption, compression
- **Session layer:** Synchronization, checkpointing, recovery of data exchange
- **Transport layer:** System-to-system data transfer, e.g. TCP, UDP
- **Network layer:** Routing of datagrams from source to destination, e.g. IP, routing protocols
- **Data link layer:** Data transfer between neighboring network elements, e.g. Ethernet, 802.11 (WiFi), PPP
- **Physical layer:** Encoding, modulating bits, e.g. on wireless, optics, cable, underwater media

Packet Headers



- Headers provide encapsulation to packet
  - Only the respective header will be processed in the respective layer
  - Other content in packet will be ignored
  - Headers enable separation of operations
  - Overriding the headers can alter the behavior

Partial Communication Stack



- Full communication stack is not required
- Middleware layer can integrate application, presentation and session layers:
  - Handling remote procedure calls
  - Translating binary data to proper format
  - Dynamical library linkage
  - Exception and error reporting
  - Debugging support

Protocols

- Protocols define
  - Format of messages sent and received among network entities
  - Order of messages sent and received among network entities
  - Actions taken on message transmission
  - Functions of acknowledge
  - Exceptional situations (e.g. error, loss of messages)
- Internet standards define protocols
  - RFC: Request for comments
  - IETF: Internet Engineering Task Force
- Examples
  - *TCP* (Transport Control Protocol) regulates the transmission rate in a distributed manner
  - *BGP* (Border Gateway Protocol) enables information among ISPs to route data among global Internet backbone
- Require research in performance analysis, correctness proof, stability proof

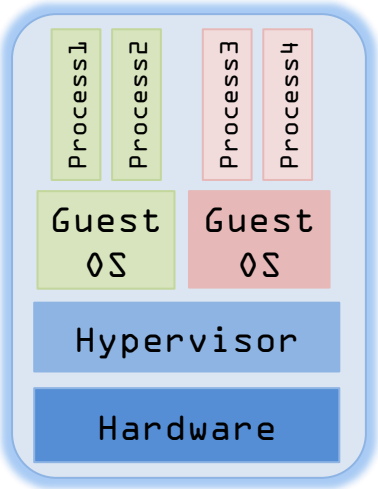
Layered System Architecture

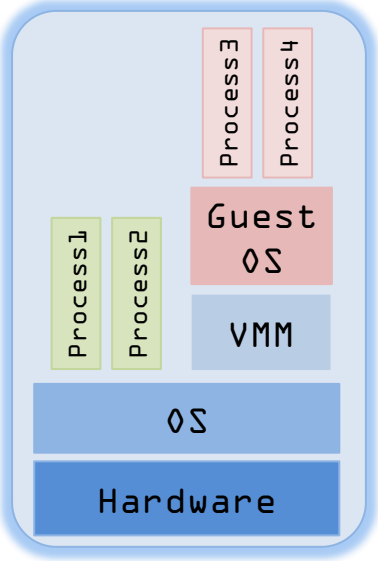
- Layered system architecture is a common paradigm
  - Encapsulating nested organization
  - Enabling flexible and extensible organization
  - Allowing overriding the underlying properties
  - Hiding dependence from underlying systems
- **Virtualization**

  Operating systems on top of other (different) operating systems
- **Overlay Networks**

  Deploy another stack on top of conventional communication stack

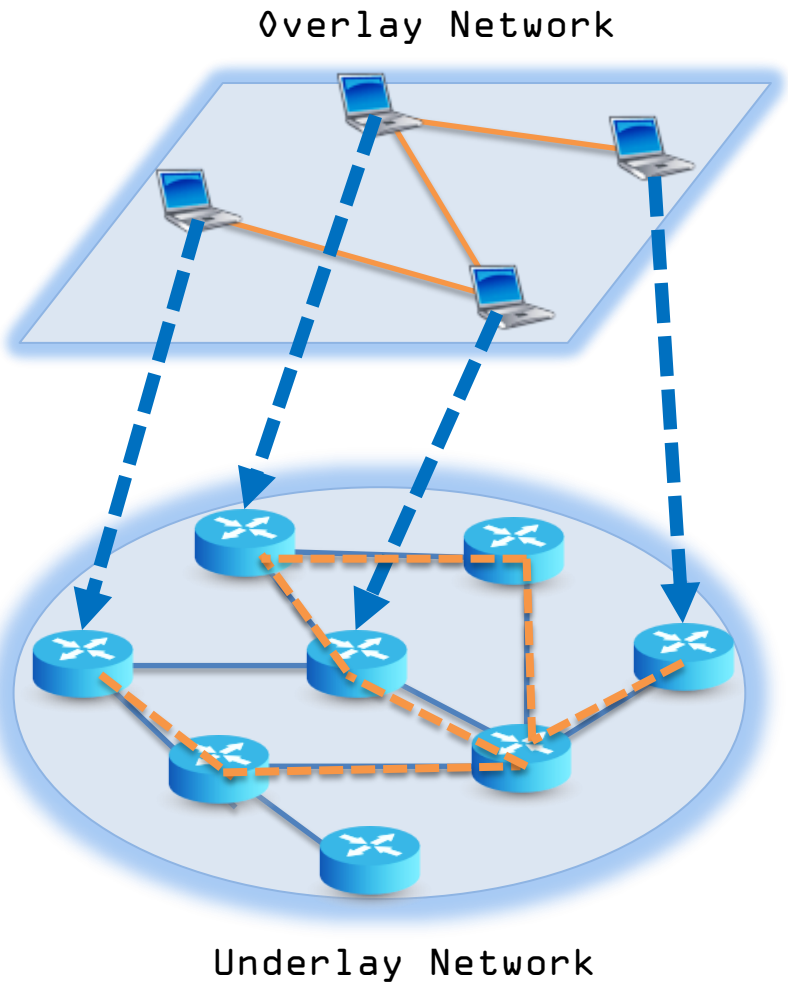
Virtualization





- Virtualization broadly refers to abstraction of resources in many aspects of computing
- One physical machine to support multiple operating systems that run in parallel
  - Hypervisor: Isolate multiple guest OSes
  - Hardware virtualization (IBM pSeries Servers)
  - Software virtualization (VMware Server, Xen)
- Reduce operation costs by consolidating services to fewer number of physical machines
- Aggregate lowly utilized machines
- Make better use of existing capacity
- Support heterogeneous environments

Overlay Networks



- A virtual network of nodes and logical links built on top of existing network
  - To implement a network service that is not available in the existing network
- Adding an additional layer of indirection or virtualization
- Altering the properties in one or more areas of underlying network
- Applications:
  - Robust networking, multicasting, VoIP (skype)

References

- Textbooks
  - Distributed Systems: Principles and Paradigms
    Andrew Tanenbaum and Maarten van Steen; Prentice Hall
  - Computer Networking
    James F. Kurose and Keith W. Ross; Pearson Addison-Wesley