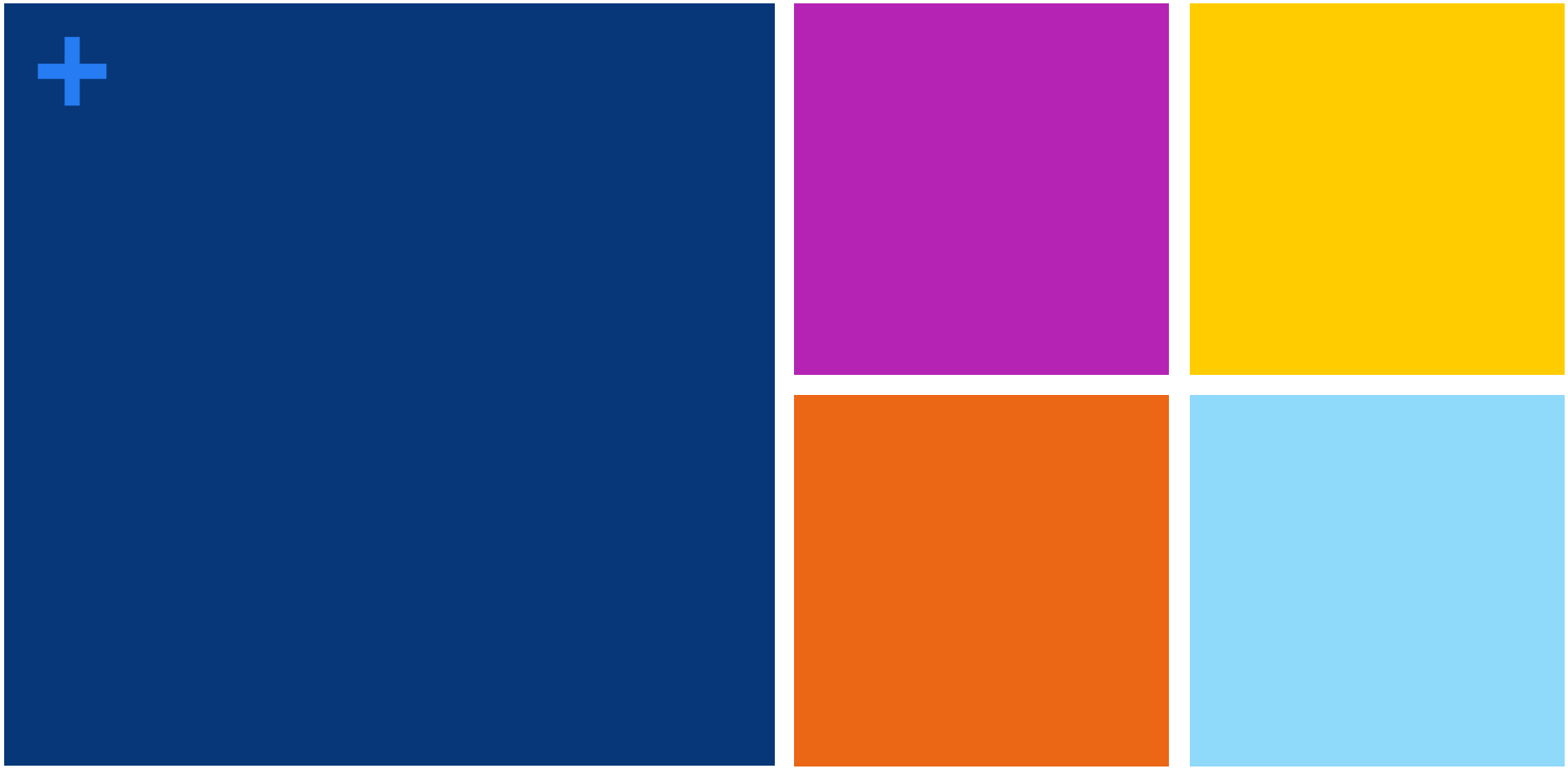


Modeling Distributed Systems

Different models provide an abstract and consistent description of a relevant aspect of DS

Course update (4/11/2019)

- Introduction to Distributed Systems
  - Transparencies
- The Transport Layer
  - UDP
  - TCP
    - *Failure transparency: Omission failures*
    - *Sockets API*
- Physical Time in Distributed Systems
  - Cristian's Algorithm
  - ICMP and NTP
- Distributed System Models
  - C/S Model
  - Interprocess Communication
  - Marshalling
  - Remote Invocation and Distributed Objects
    - Case study: Java RMI (B1 + B3 @ Lab B6)

Introduction.
DS: difficulties and threats

- Varied demands
- Heterogeneity: h/w, OS, Networks
- Non-synchronized clocks
- DoS attacks and attacks on data integrity

- **How to handle complexity? ORGANIZATION**

Modelling Distributed Systems

1. Introduction
2. Physical Models
   - Computers and their interconnection
3. Architectural Models
   - Computational and communication tasks performed by computational elements
4. Fundamental Models
   - Abstract perspective to solve DS issues
   - No global time, types of faults, security

Physical Models

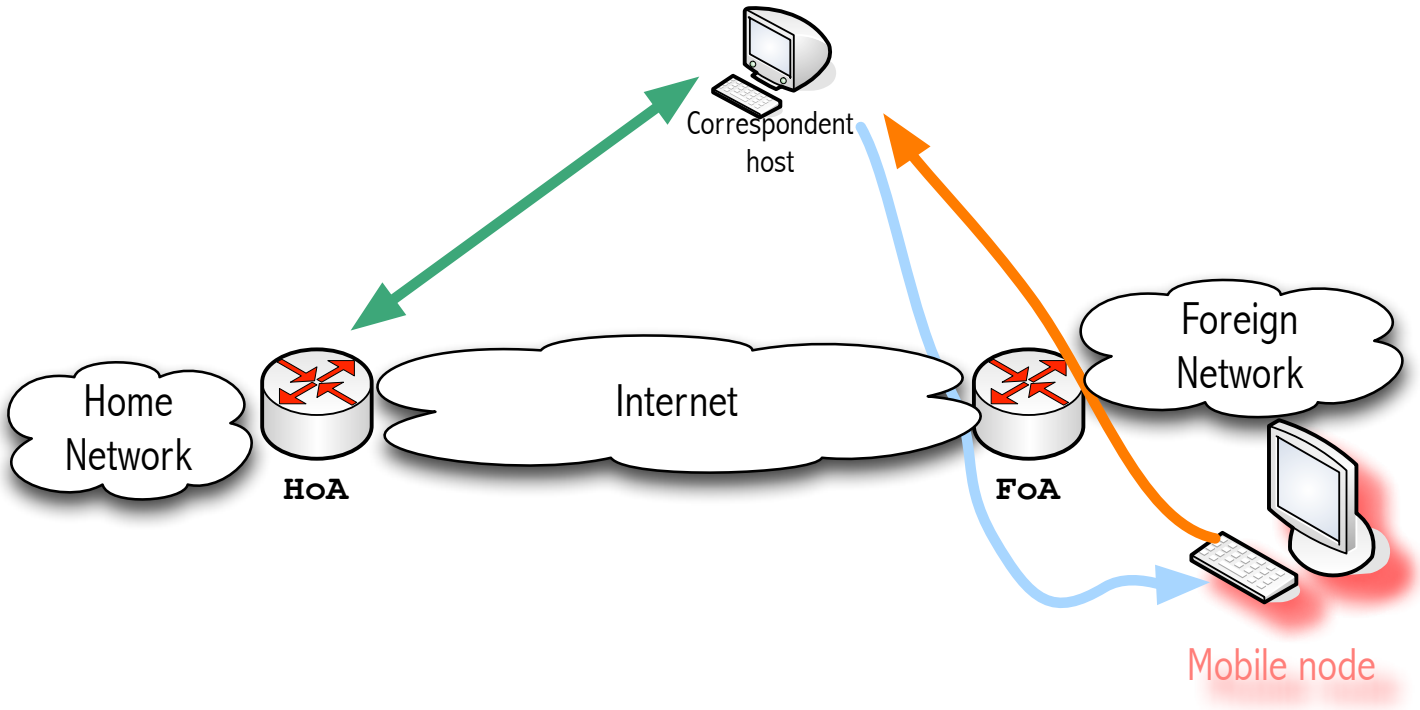
A representation of computers and networks, hardware

Abstracts away specific technology details

- Mobile computing
- Ubiquitous computing
- Cloud computing

Mobile computing

- The mobile node migrates from its Home Network (HN) to a Foreign Network (FN)
- Transparently communicates in Internet as though it were located at its HN



Architectural models

Software architecture: The system's software entities and their interactions

System architecture: In which machines those components run

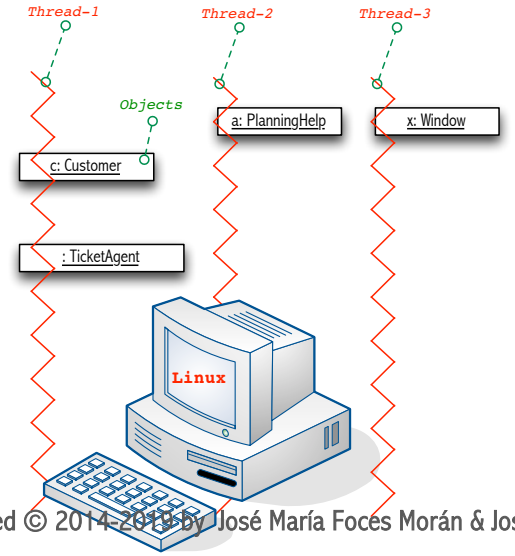
Communicating entities in a DS

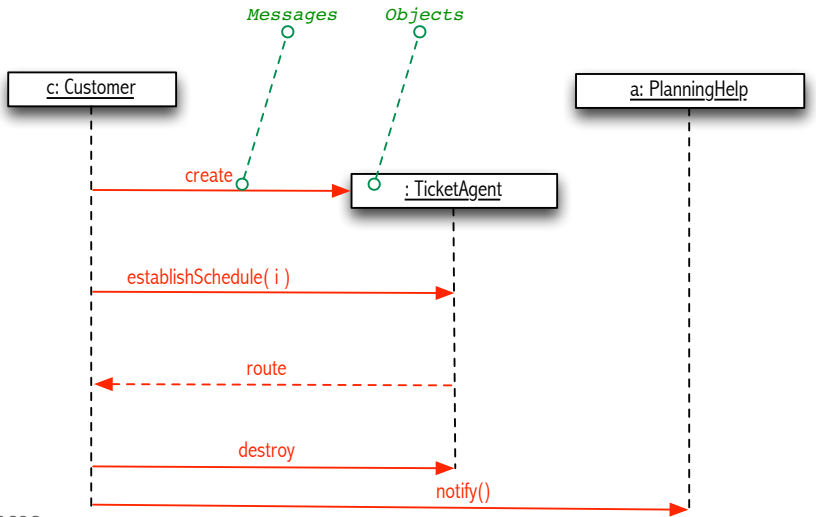
Systems standpoint

- Processes or Threads
- Drawbacks
  - In sensor networks, for example, the OS does not support processes

Programming standpoint

- Objects
- Components
- Web Services

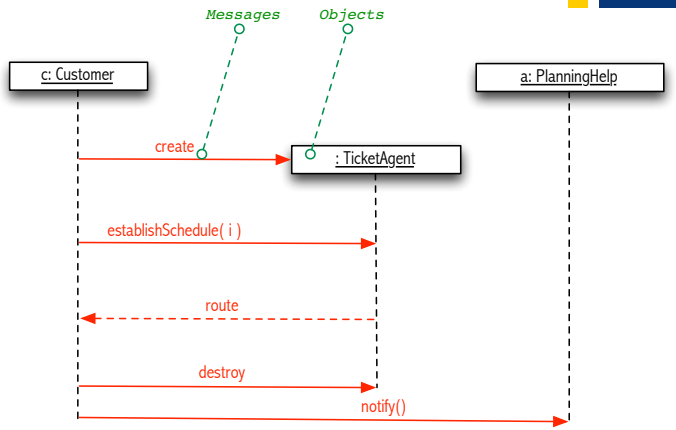




Software entities

Programming standpoint

- Objects: Computation consists of a number of objects interacting
  - Interfaces with IDLs
  - Tightly-coupled applications



- Components: In response to objects weaknesses
  - Specify interfaces (like objects) AND ALSO assumptions regarding the availability of other components
  - A more complete contract for system construction

- Web Services: Encapsulation of behavior + interfaces
  - Integrated in the WWW (Standards, specs, etc.)
  - App identified by an URI, interfaces via XML (Definition, access and discovery)
  - More 'complete services' than objects

Communication Paradigms

How software entities communicate

Direct communication

- Interprocess communication
  - TCP service interface + Sockets API
- Remote invocations
  - The MOST common in DS (Java RMI)

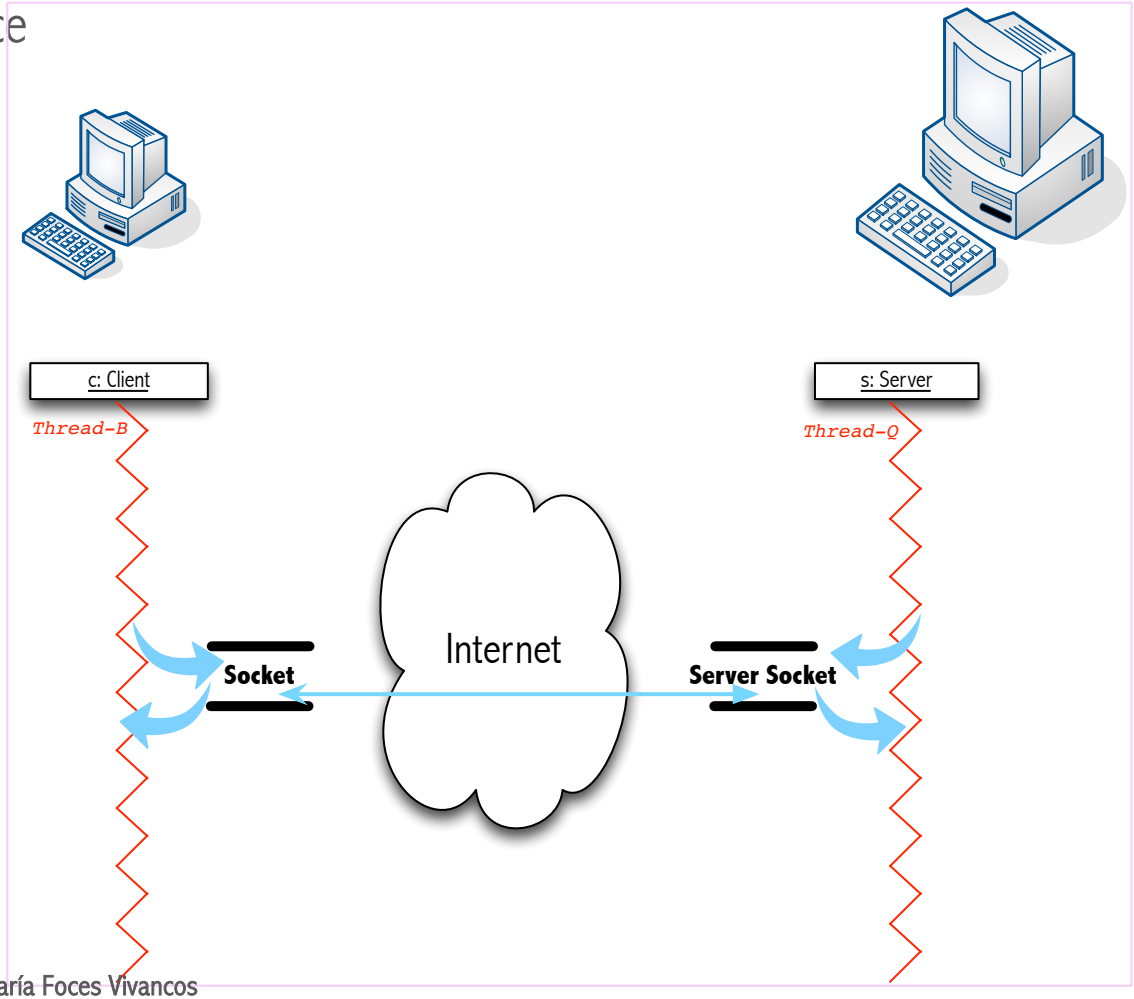
Indirect communication

- Communication is through a third entity

Communication Paradigms

IPC (Inter Process Communication)

- Access to the TCP service interface API: Sockets
  - From a systems standpoint it is threads that are the endpoints of communication
- Multicast communication via Multicast sockets



Communication Paradigms

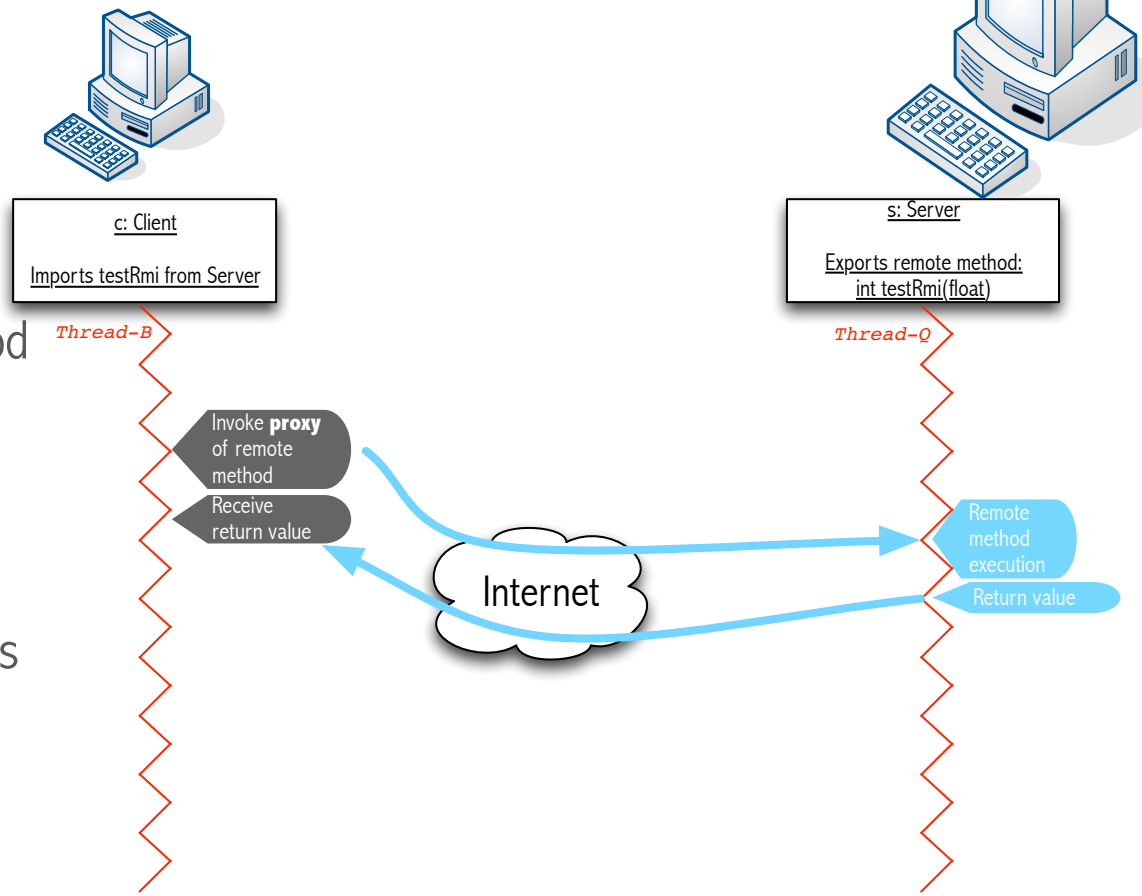
Remote Invocation (I/II)

- The most common communication paradigm in DS
- Consists of calling **remote procedures (RPC)** or **methods (RMI)**
- Based on two-way exchange between entities
  - **Request-reply** protocols
    - Basic for Client/Server computing
    - HTTP
- RPC (Remote Procedure Call)
  - A major breakthrough in DS
  - Hides a lot of aspects of distribution, encoding of parameters, passing of messages
  - Directly and elegantly supports C/S
  - Access and location transparency

Communication Paradigms

Remote Invocation (II/II)

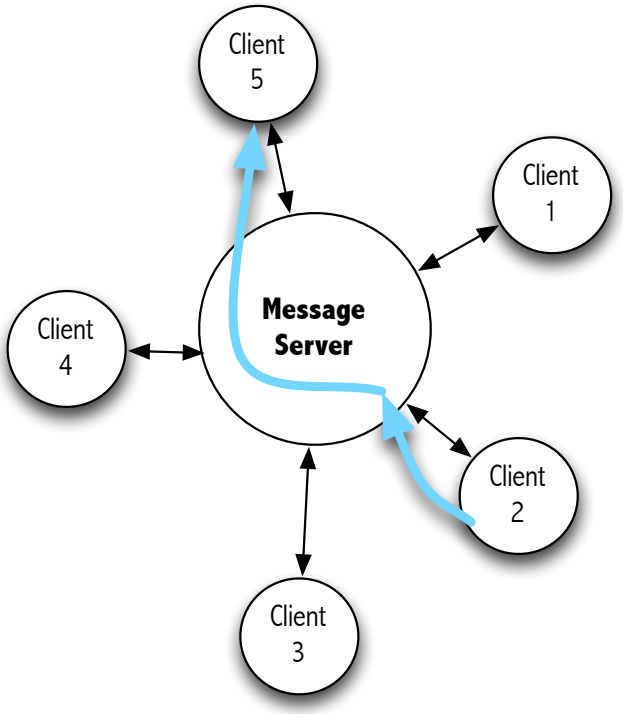
- **RMI (Remote Method Invocation)**
  - The RPC of the OO world
  - Java RMI
  - Programming against interfaces
- A calling object can invoke a method on a remote object
- Object identity is supported
- Object references can be passed as parameters in remote calls



Communication Paradigms

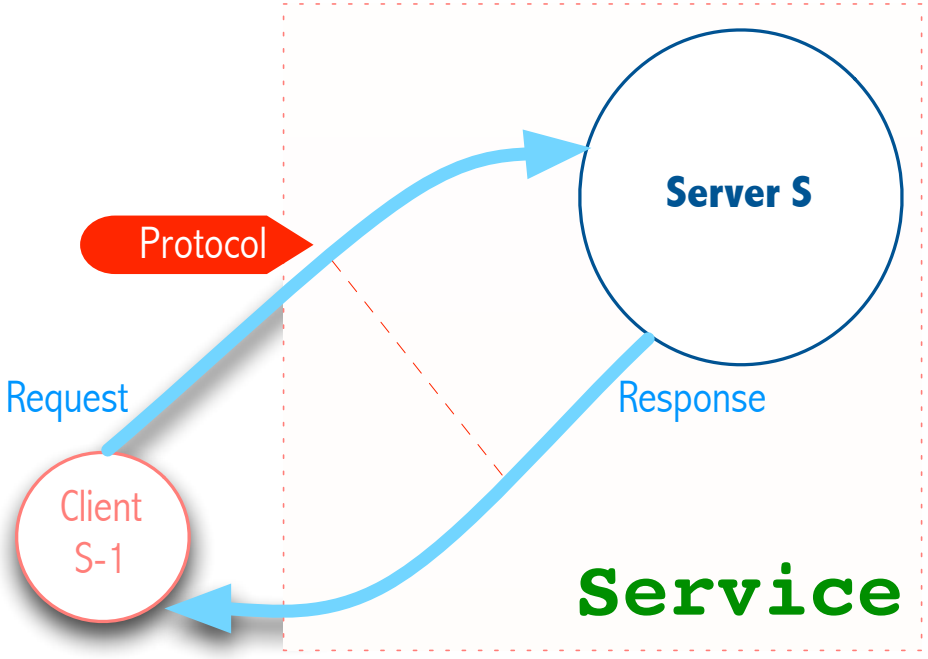
Indirect Communication Techniques

- Senders and receivers are decoupled
  - In space: Senders do not need to know who they are sending to
  - In time: Senders and receivers do not need to be executing at the same time
- Group communication: Delivering a message to a set of recipients
- Publish-subscribe systems: Producers (publishers) distribute items of interest (events) to a large number of consumers (One to many)
- Message queues: Producer sends to a specified queue
- Distributed shared memory



Abstract Client/Server model
What is it?

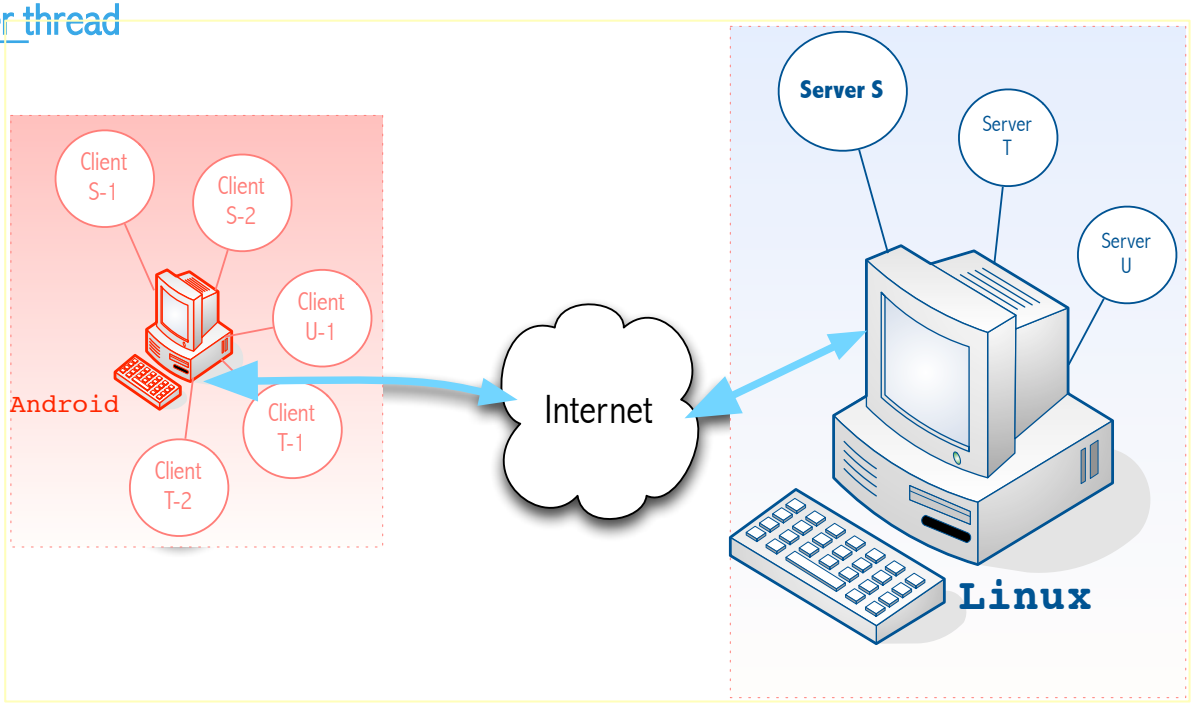
- The most important model in DS
- It's an abstract concept
  - WHAT?
    - Client sends Request
    - Server sends Reply
- It may be implemented in various ways
  - HOW?
  - WHERE?



Abstract Client/Server model

How is it implemented?

- Systems standpoint
  - Client operations are executed in a thread
  - Server ops are executed in another thread
- Programming standpoint
  - Client and Server
    - Object
    - Component
    - Web service

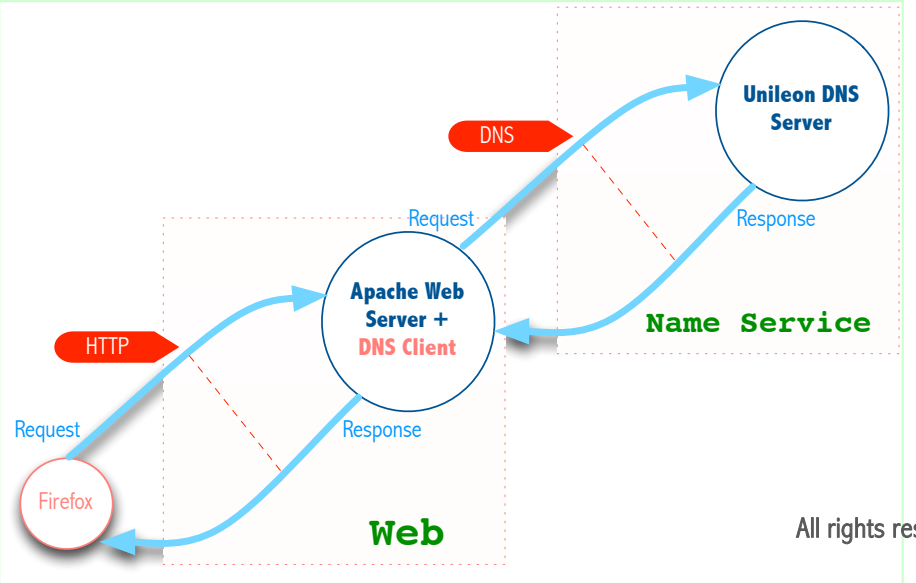


Client/Server model

Characteristics

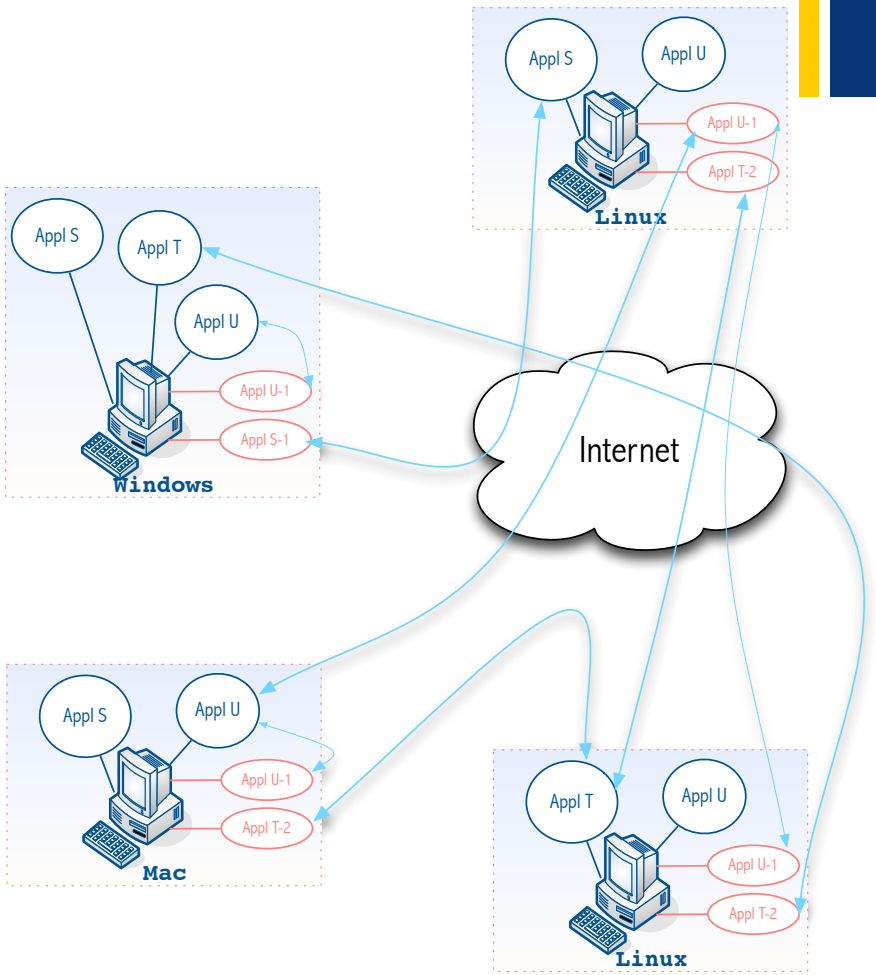
```
JUSTIFICATION FOR THIS SLIDE
APACHE WEB SERVER CAN LOOKUP EVERY RECEIVED IP IN THE DNS
FOR LOGGING THE DNS NAME INSTEAD OF THE IP ITSELF IN THE APAHCE ACCESS
LOG
# HostnameLookups: Log the names of clients or just their IP addresses
# e.g., www.apache.org (on) or 204.62.129.132 (off).
# The default is off because it'd be overall better for the net if people
# had to knowingly turn this feature on, since enabling it means that
# each client request will result in AT LEAST one lookup request to the
# nameserver.
#
HostnameLookups ON
```

- The most widely employed in DS
- Chained servers
  - It may delegate some functions to other, more specialized Servers
  - *The Server assumes the role of Client*
- This model does not scale well due to the centralized nature of the server



The peer-to-peer model

- Both entities involved assume similar roles
  - They offer the *same service interfaces*
- User network and computing resource utilization is maximized
  - Napster, BitTorrent
- Objects replicated in several computers: much more complex than C/S



Placement

Where are the software entities located?

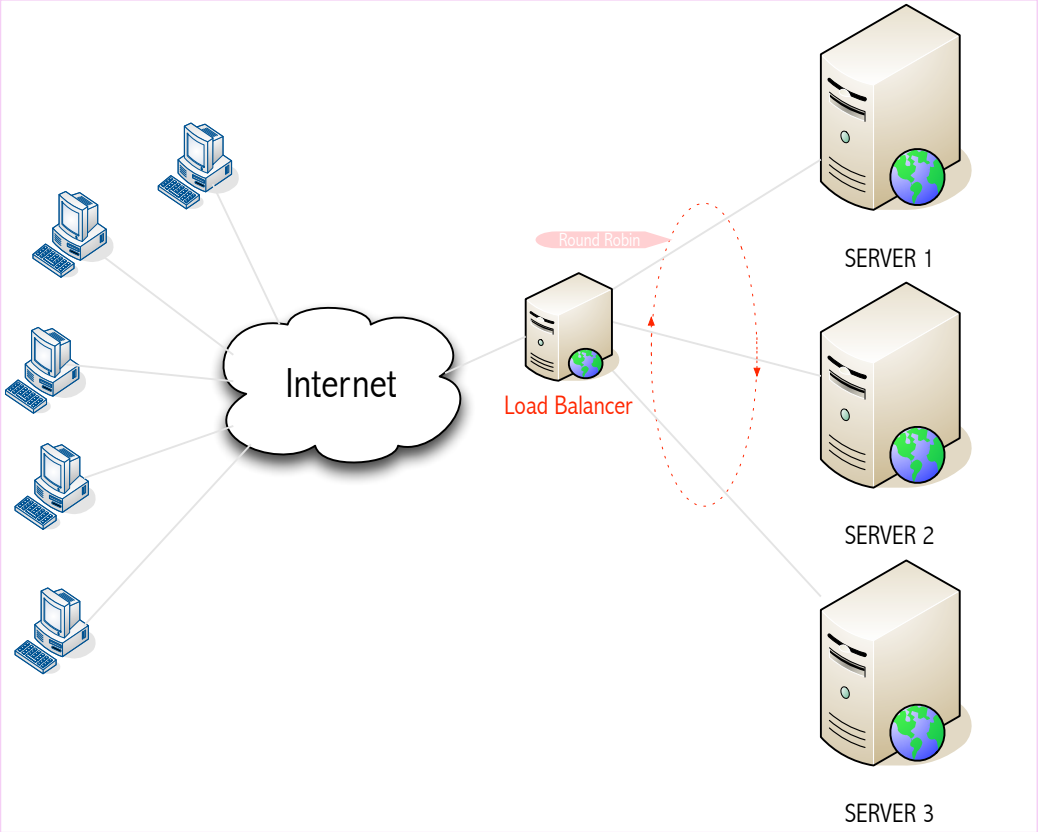
Affects the quality of networks, the performance of servers, reliability and depends on certain properties of the application

- Strategies:
  - Multiple servers copperating in the provision of a service
  - Caching
  - Mobile code
  - Mobile agents

Placement

Replicating servers: Load Balancers

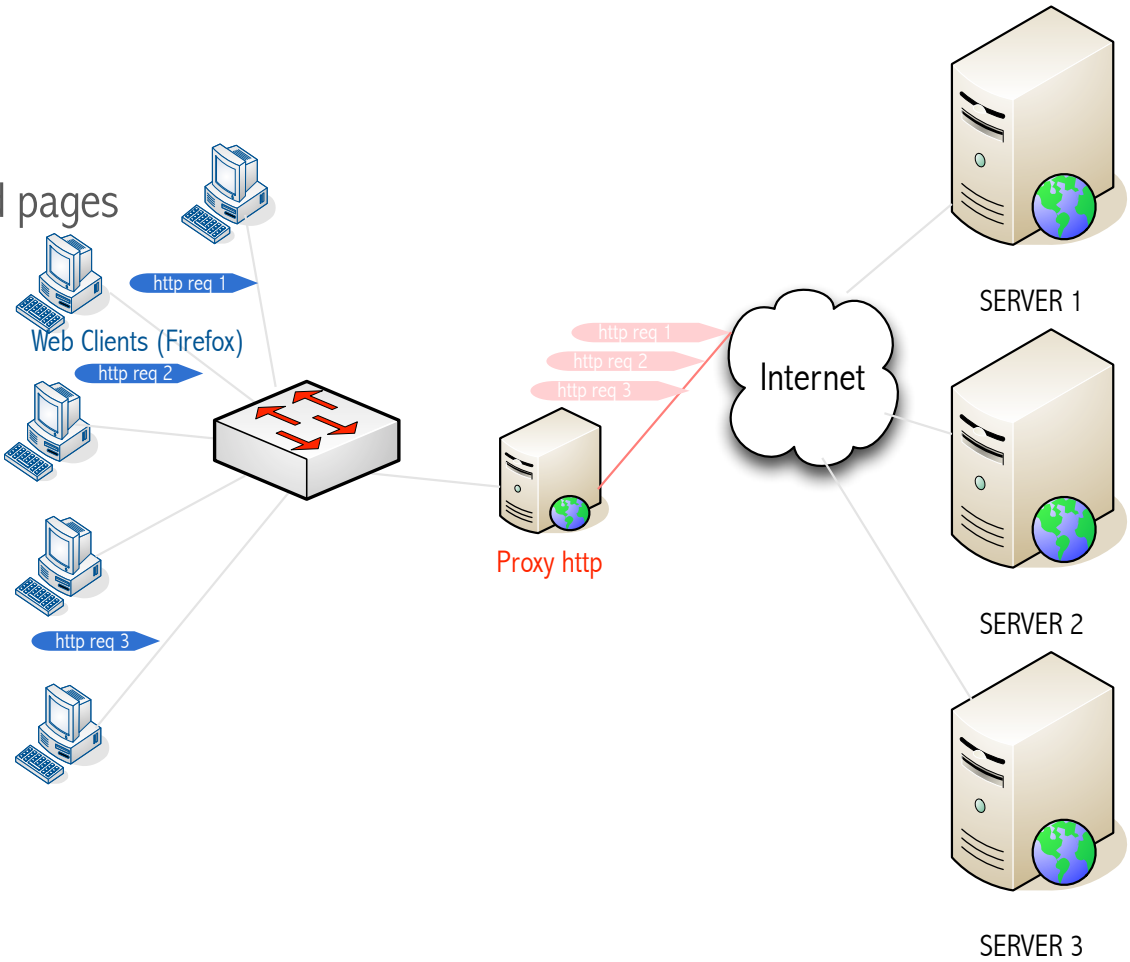
- Multiple servers collaborate to provide a single service
  - Improves availability
  - Improves performance
    - Load balancers
    - Cluster computing



Placement

Replicating servers

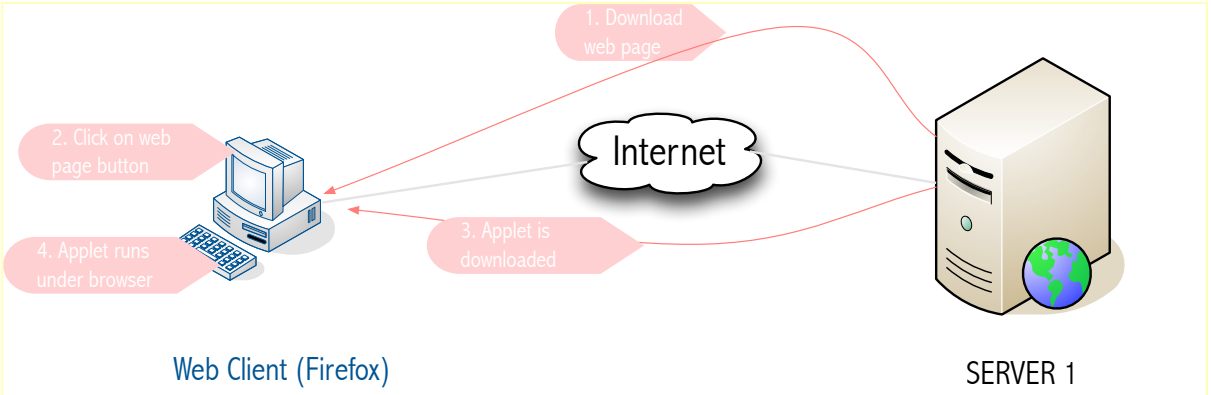
- Caching
  - Proxy http
  - Stores frequently accessed pages
  - Reduces traffic offered
  - Improves security



Placement

Mobile code

- Java Applets
  - A Java Object known as *Applet* is dynamically loaded into the client Web browser when the user requests it
  - Applet runs under control of the browser
    - Security can be fine-tuned
  - Extends the functionality of the http protocol and browser
    - Push model: *Server* needs to contact client randomly –not possible by default
- Mobile agents
  - Complete program migrates to another computer
  - Good for reducing consumed bandwidth
  - Security is an issue



Architectural patterns

Composite recurring structures

Based on the elements just studied

Architectural Patterns

Composite recurring structures

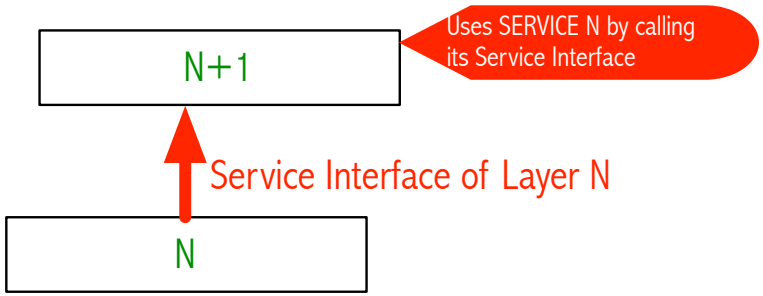
Not final solutions, just partial insights that must be combined

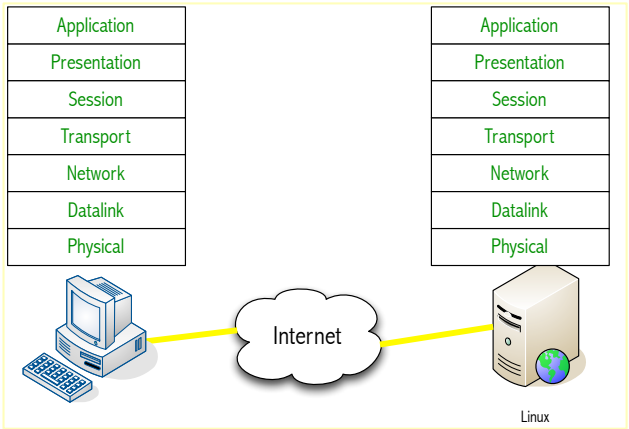
- Layering
- Tiered architectures
- Web Services constitute also a pattern

Layering in CN

A layer uses the services provided by the layer below

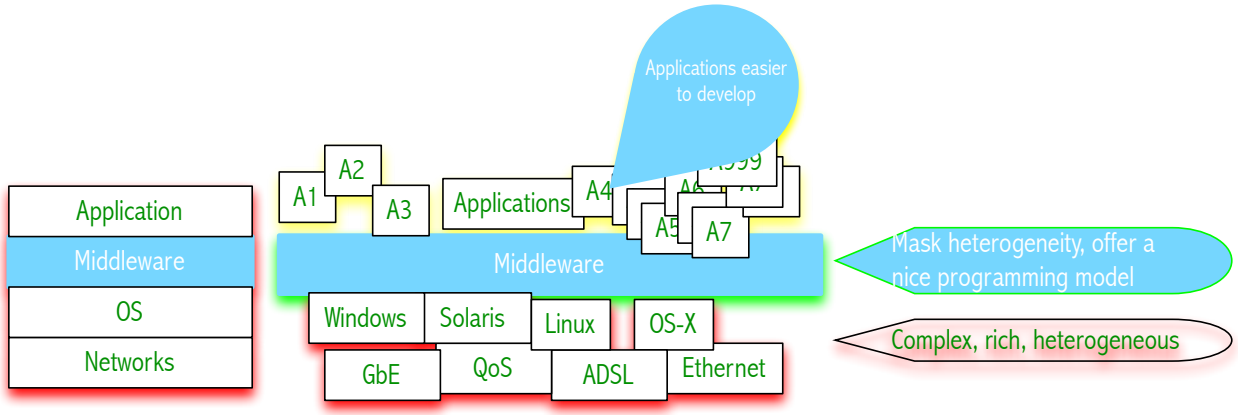
- Implementation details remain hidden
  - Layer N+1 knows nothing about the implementation of layer N
- Service use via Service Interface, only
- OSI 7-layer model

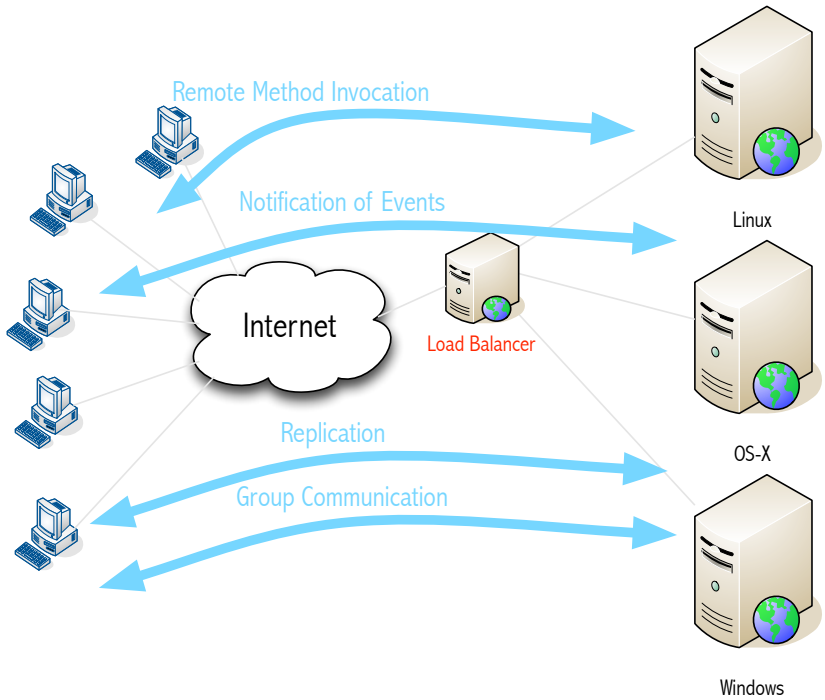




Layering in Distributed Systems

Middleware





Tiered architectures

Complementary to layering

- Organize a layer by placing its functionality into several servers

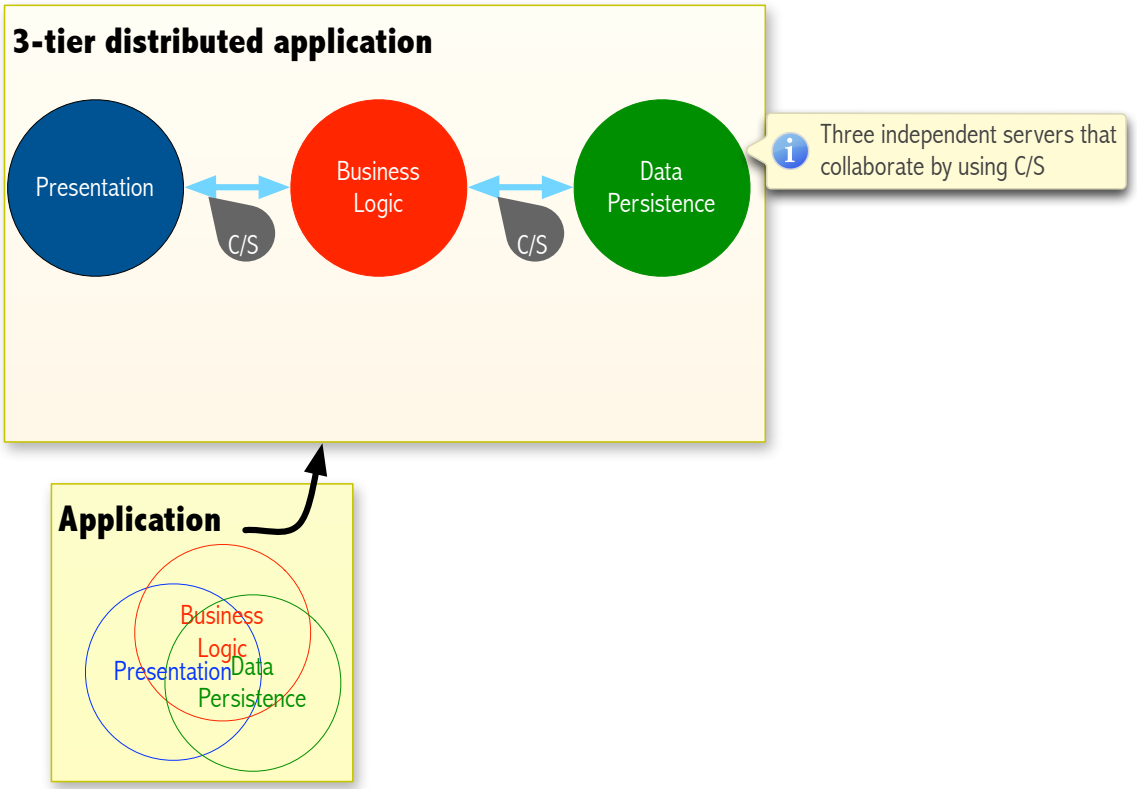
3-tier: An application is partitioned into 3 responsibilities

- Presentation: Present results, take user input
- Application logic: Domain-specific logic operations
- Data Access: Provide persistence to the data by using a DBMS, for example

- N-tier: Wikipedia

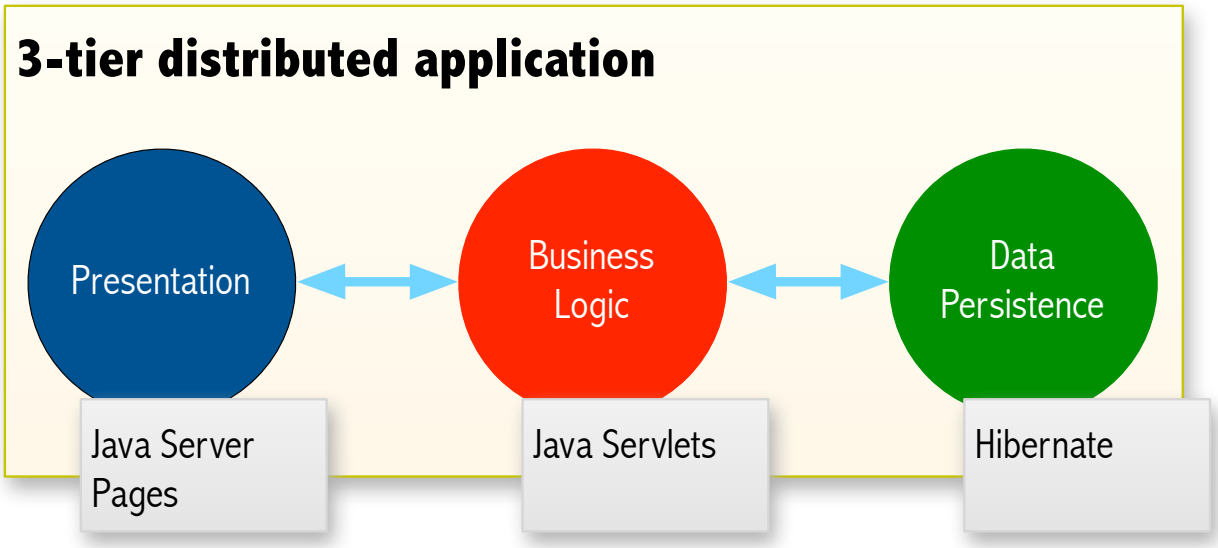
3-tier

- C/S between each two consecutive tiers
- Chained C/S



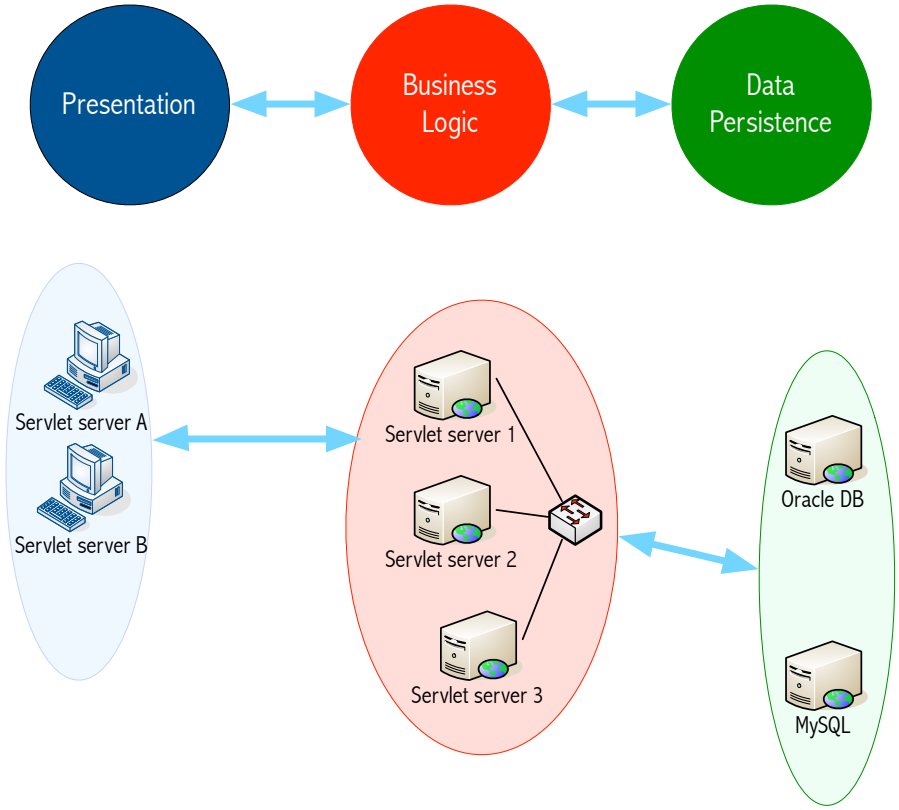
3-tier

- Java Server Pages: Java executed on a server that calculates the html necessary to paint the presentation on a browser
- Servlets: Multithreaded Java programs capable of encapsulating http protocol primitives into Java types
- Hibernate: Allows to encapsulate SQL queries into Java programs



3-tier: physical servers?

- How many *physical servers* must be used to host each of the three responsibilities (Logical servers)?
- Could one physical server suffice?
- And two?
- Are three the least number of physical servers needed?
- Then, can we have three servers implement the Business Logic tier?
  - Discuss the tradeoffs in this case



Middleware solutions

Major categories:	*Subcategory*	*Example systems*
Distributed objects (Chapters 5, 8)	Standard	RM-ODP
	Platform	CORBA
	Platform	Java RMI
Distributed components (Chapter 8)	Lightweight components	Fractal
	Lightweight components	OpenCOM
	Application servers	SUN EJB
	Application servers	CORBA Component Model
	Application servers	JBoss
Publish-subscribe systems (Chapter 6)	-	CORBA Event Service
	-	Scribe
	-	JMS
Message queues (Chapter 6)	-	Websphere MQ
	-	JMS
Web services (Chapter 9)	Web services	Apache Axis
	Grid services	The Globus Toolkit
Peer-to-peer (Chapter 10)	Routing overlays	Pastry
	Routing overlays	Tapestry
	Application-specific	Squirrel
	Application-specific	OceanStore
	Application-specific	Ivy
	Application-specific	Gnutella