CS 550:

Advanced Operating Systems

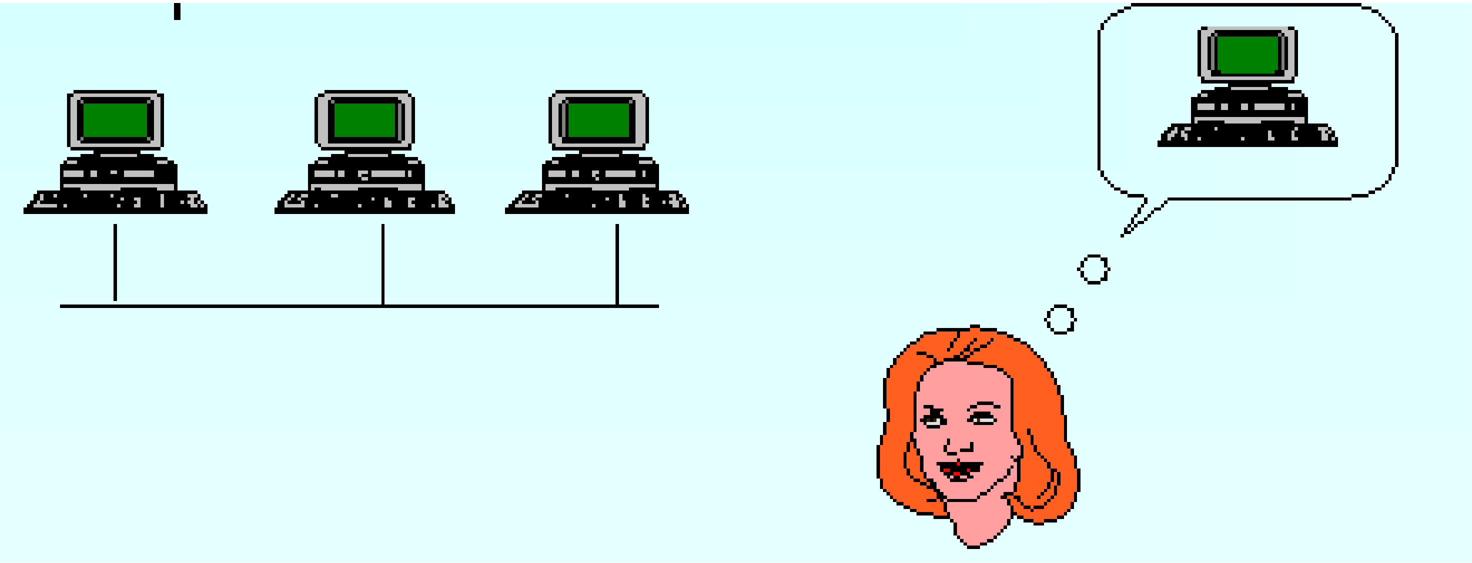
Distributed System Architectures

Ioan Raicu
Computer Science Department
Illinois Institute of Technology

CS 550
Advanced Operating Systems
January 25th, 2011

Last Class: Introduction

- A distributed system is defined as
  - A collection of independent computers that appears to its users as a single coherent system



Last Class: Design Issues

- Resource sharing
- Openness
- Concurrency
- Scalability
- Fault tolerance (reliability)
- Transparence

Outline

- Architectural styles
- System architectures
- Discussion on Client-Server Model

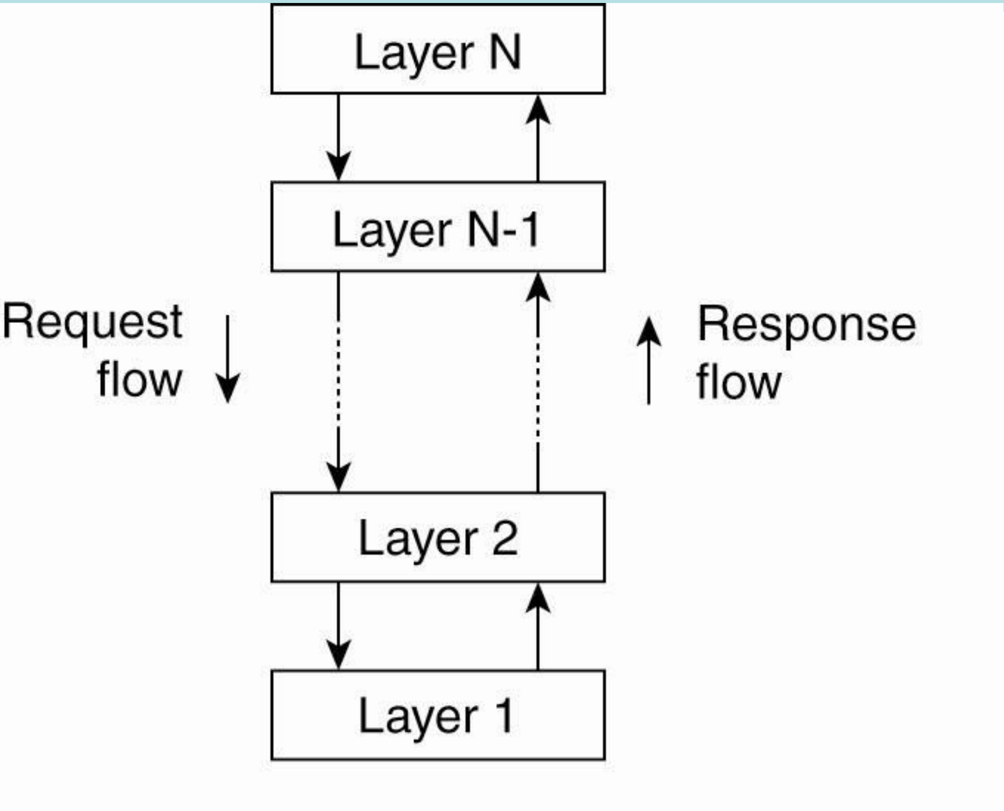
Architectural Styles

- Software architectures
  – Logical organization of distributed systems into software components
- Component

  \- A modular unit with well-defined required and provided **interfaces** that is **replaceable** within its environment

- Four important styles
  - Layered architectures
  - Object-based architectures
  - Data-centered architectures
  - Event-based architectures

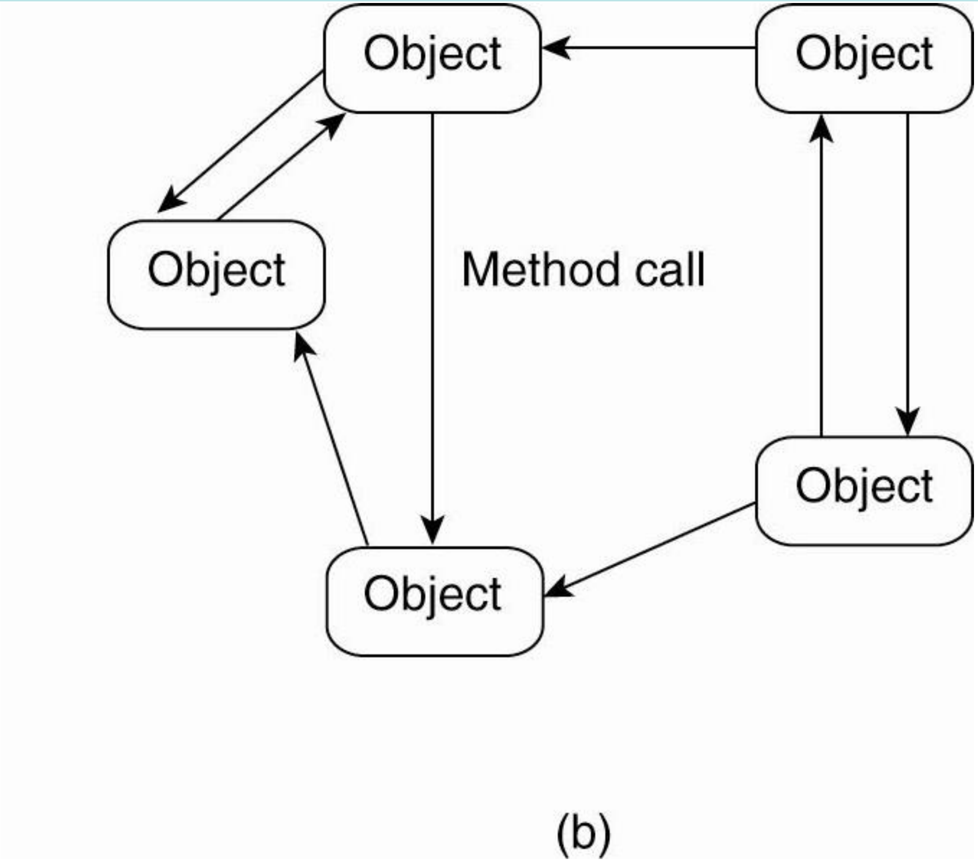
Layered Style



(a)

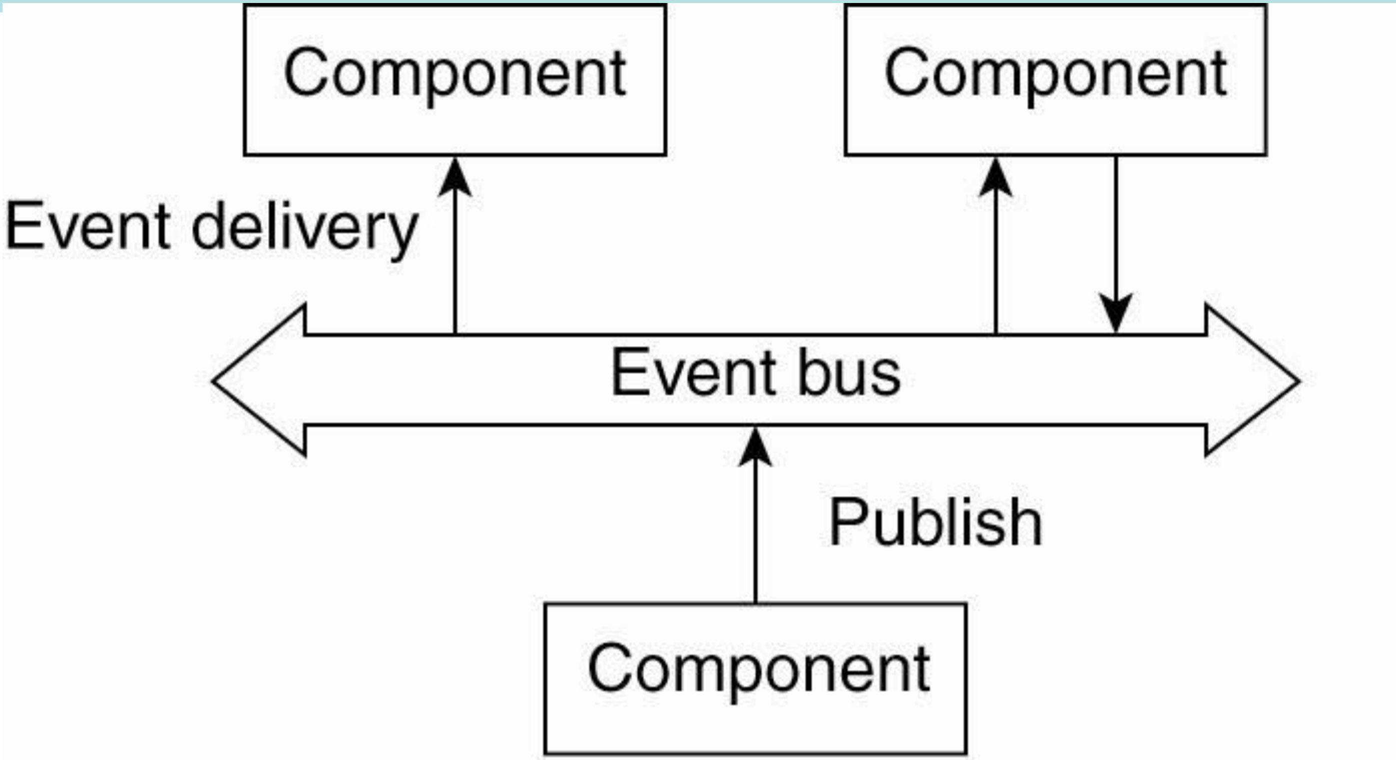
Components are organized in a layered fashion where a component at Layer L_i is allowed to call components at the underlying layer L_{i-1}

Object-Based Style



Each object corresponds to a component, and these components are connected through a procedure call mechanism

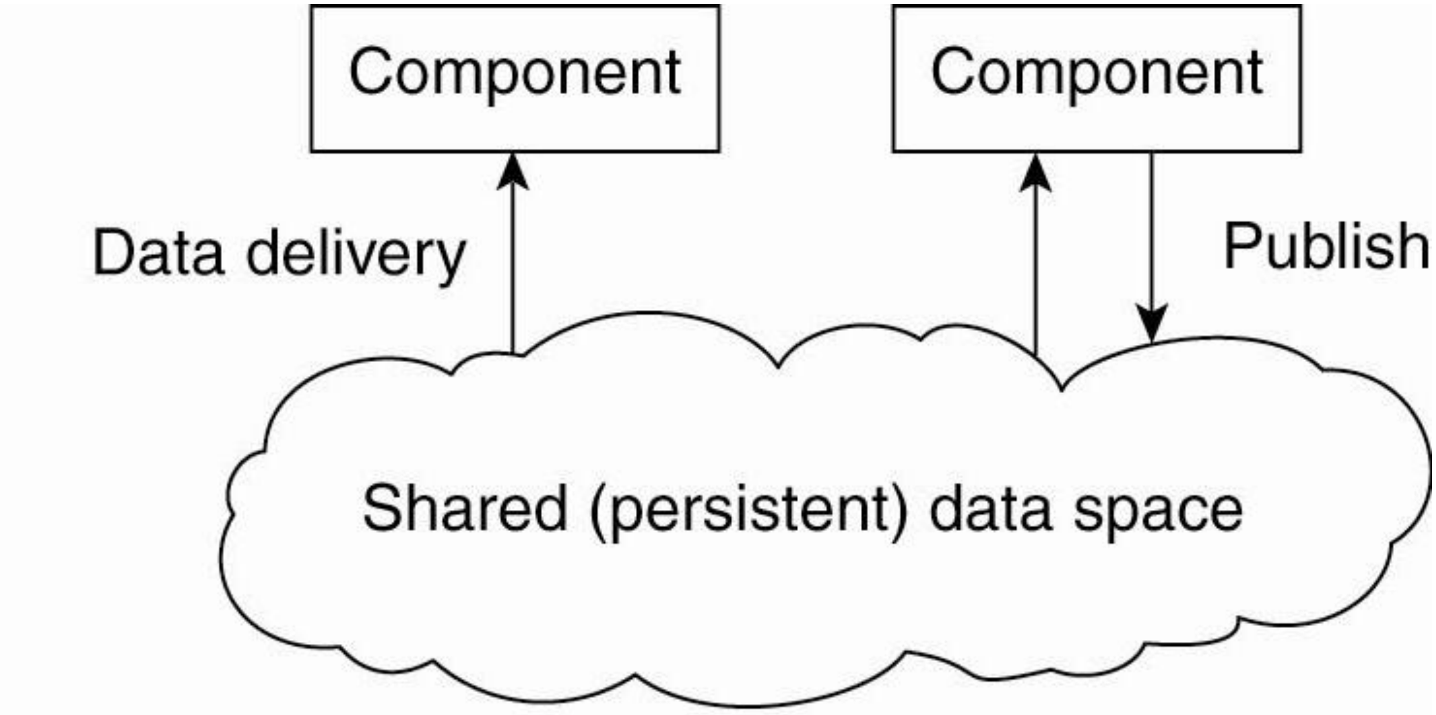
Event-Based Style



(a)

Processes communicate through the propagation of events

Data-Centered Style



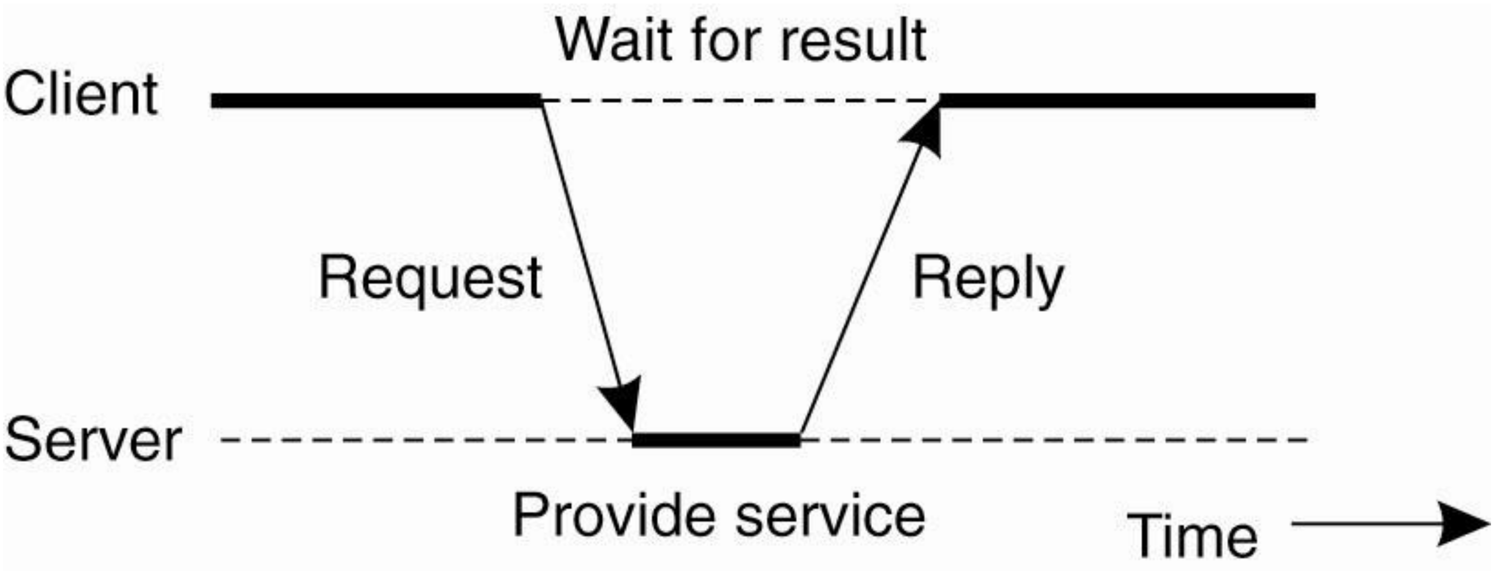
(b)

Processes communicate through a common repository; When combining with event-based architectures, it is also known as shared data spaces

System Architecture

- Instantiate and place software components on real machines
- Important architectures
  - Centralized
  - Decentralized
  - hybrid

Centralized Architectures

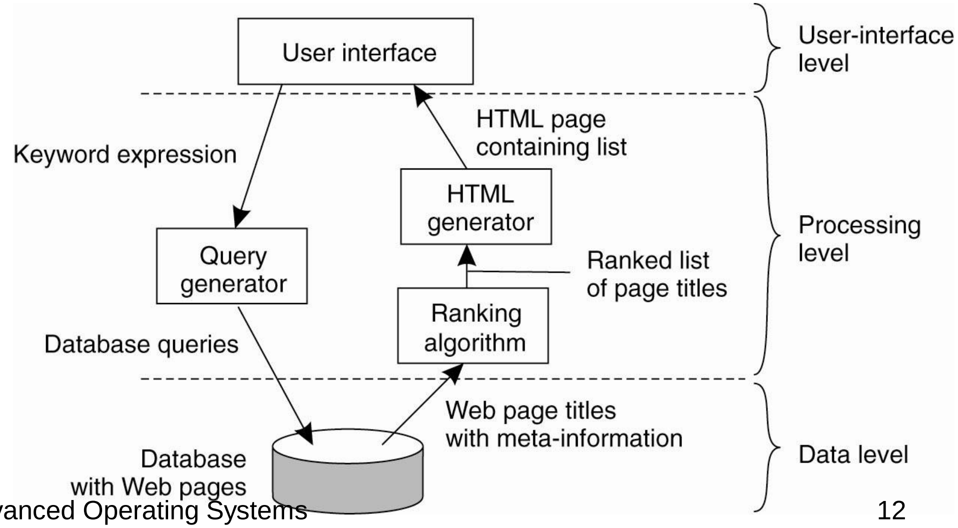


Client-server model; Two process groups:

- a server is <u>a process implementing a specific service</u>
- a client is <u>a process requesting a service from a server</u>
- aka **request-reply behavior**

Centralized Architectures

- Application Layering
  - There is no clear distinction between a client and a server
- Since many client-server applications are targeted toward supporting user access to DB
  – The user-interface level
  – The processing level
  – The data level



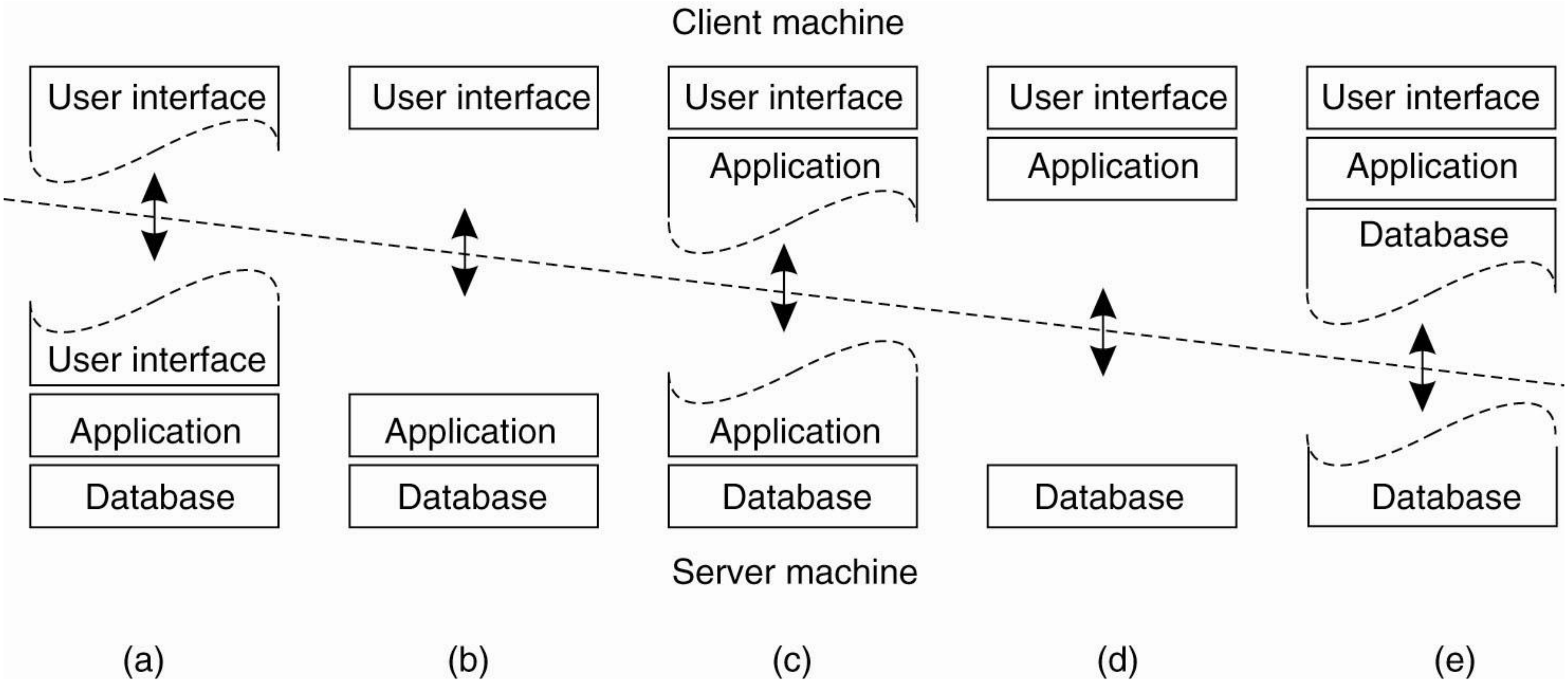
Centralized Architectures

- Physically distribute a client-server application across several machines => **Multi-tiered architectures**
- The simplest organization is to have only two types of machines:
  - A client machine containing only the programs implementing (part of) the user-interface level
  - A server machine containing the rest, i.e., the programs implementing the processing and data level

Centralized Architectures

To distribute the programs in the **application layers** across different machines

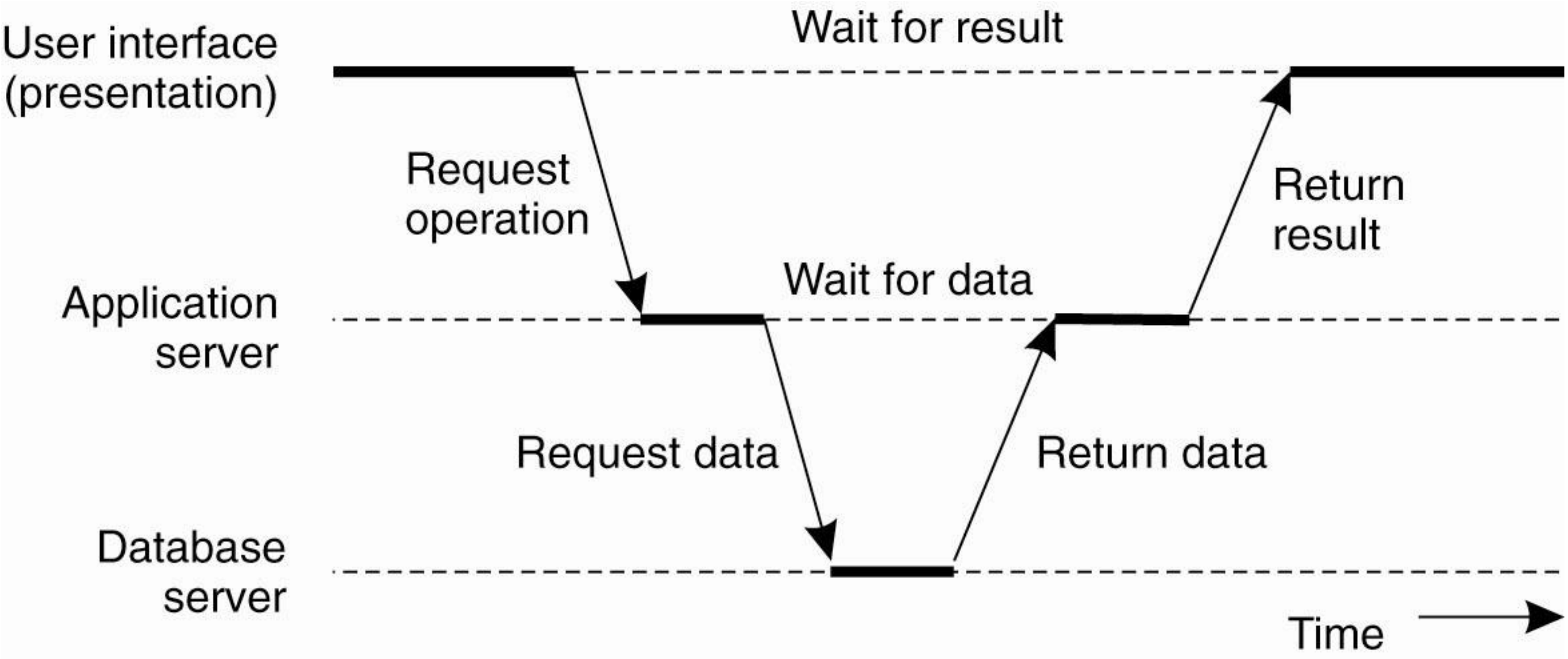
Examples of **two-tiered architectures**:



Centralized Architectures

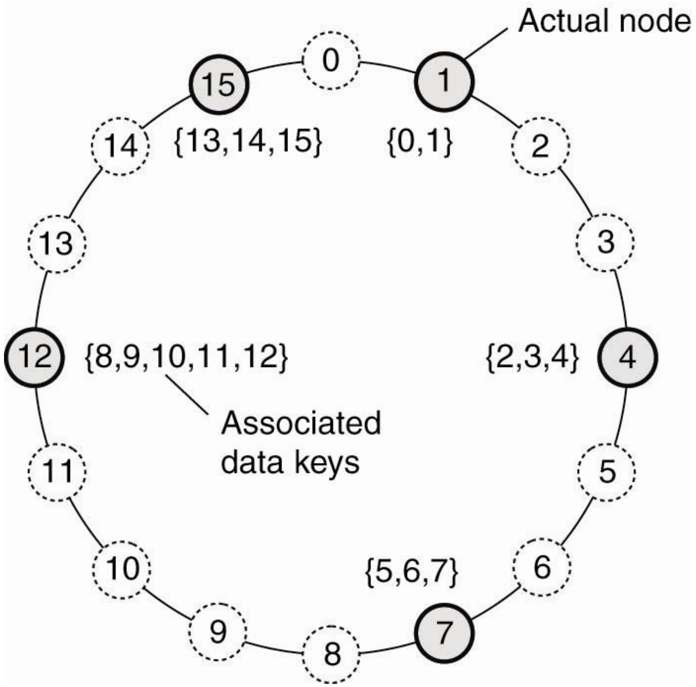
Examples of **multi-tiered architectures**:

a single server is being replaced by multiple servers running on different machines



Decentralized Architectures

- **Peer-to-peer systems**
  - The processes that constitute a p2p system are all equal
  - Much of the interaction between processes is symmetric--- each proc will act as a client and a server
  - Focuses on how to organize the processes in **an overlay network**
    - Structured vs. unstructured



- **Structured P2P architectures**
  - The overlay network is constructed using a deterministic procedure, e.g., DHT

Decentralized Architectures

- **Unstructured P2P architectures**
  - Each node maintains a list of neighbors
  - When a node needs to locate a specific data item, the only thing it can effectively do is to flood the network with a search query

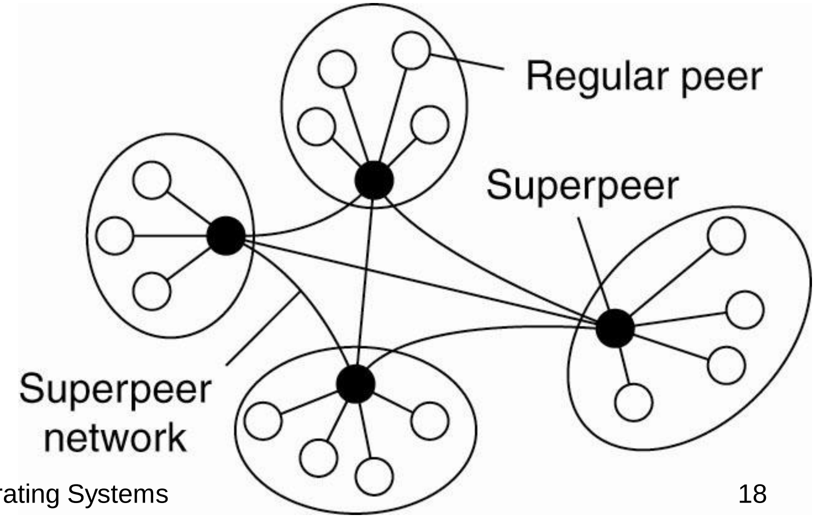
Actions by active thread (periodically repeated):

```
select a peer P from the current partial view;
if PUSH_MODE {
    mybuffer = [(MyAddress, 0)];
    permute partial view;
    move H oldest entries to the end;
    append first c/2 entries to mybuffer;
    send mybuffer to P;
} else {
    send trigger to P;
}
if PULL_MODE {
    receive P's buffer;
}
construct a new partial view from the current one and P's buffer;
increment the age of every entry in the new partial view;
```

(a)

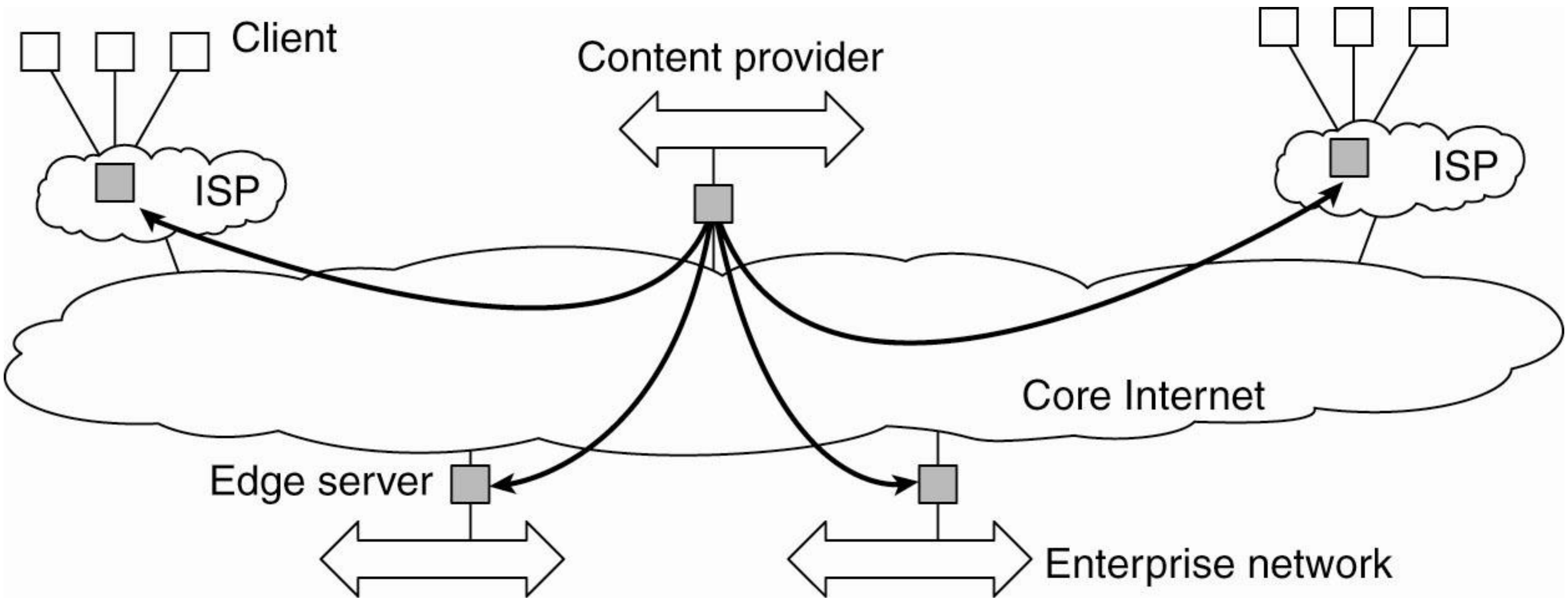
Decentralized Architectures

- In unstructured P2P systems, locating relevant data items can become problematic as the network grows
- One solution:
  - Superpeers to maintaini an index or acting as a broker
  - Superpeers are often organized as in a P2P network, leading to a hierarchical organization



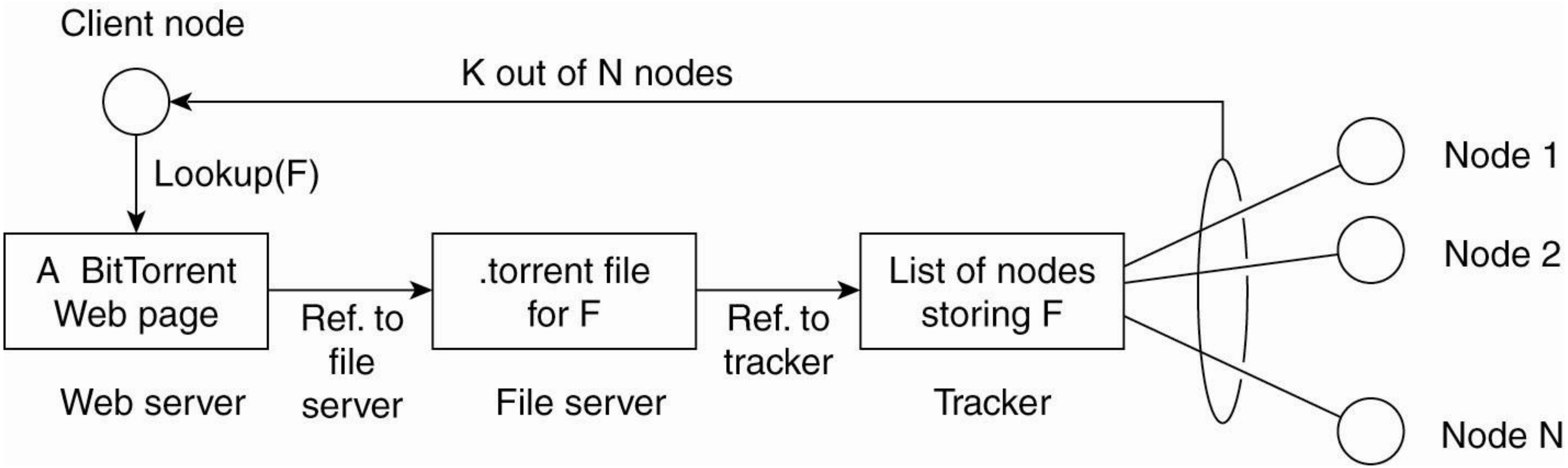
Hybrid Architectures

- Client-server solutions are combined with decentralized architectures
- **Edge-server systems**



Hybrid Architecures

- **Collaborative distributed systems**
  - BitTorrent example

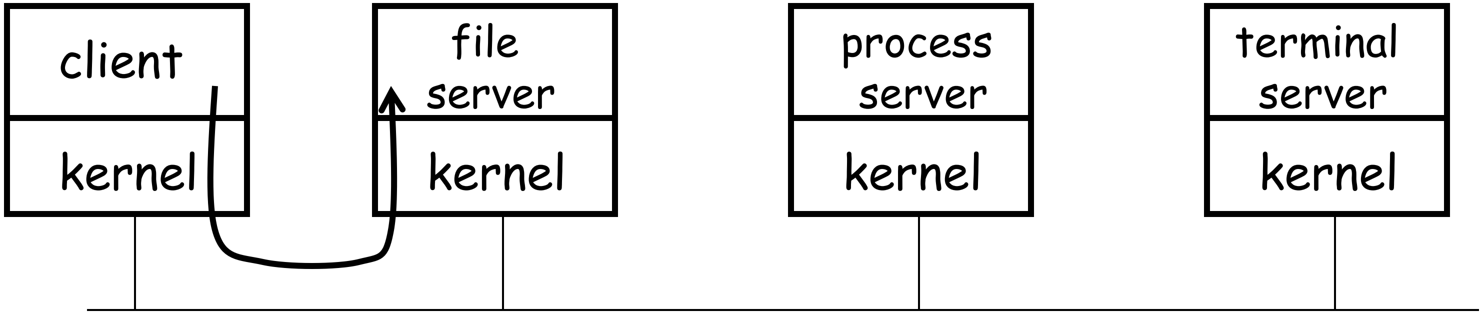


Self-Management

- Adaptability in distributed systems can be achieved by having the system monitor its own behavior and taking appropriate measures when needed
  - Autonomic systems or self-* systems
- Three basic approaches to adaptive software:
  - Separation of concerns
  - Computational reflection
  - Component-based design
- Feedback control model

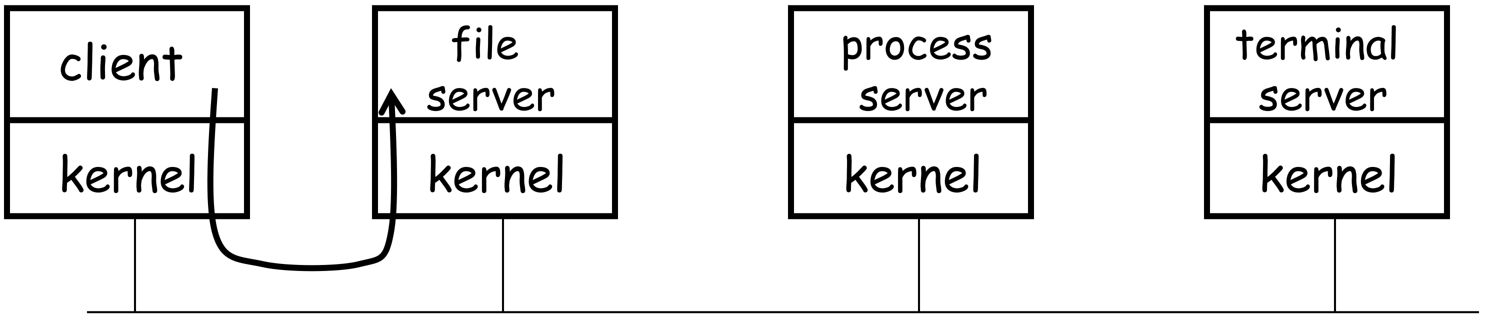
Client-Server Communication Model

- Structure: group of servers offering service to clients
  - Servers: offer services to the users called "clients"
  - Clients: applications requiring services from servers
  - Example: Web Server/clients, File server …
- Why use client-server model
  - simplicity
  - low(er) overheads (why?)



Client-Server Comm. Model

- Based on a request/response paradigm
  - Clients send a request asking for service (e.g., a file block)
  - Server processes and replies with result (or error)
- Techniques:
  - Socket, remote procedure calls (RPC), Remote Method Invocation (RMI)



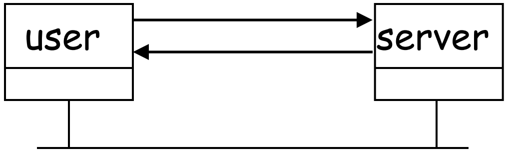
Issues in Client-Server Communication

- Addressing
- Blocking versus non-blocking
- Buffered versus unbuffered
- Reliable versus unreliable
- Server architecture: concurrent versus sequential
- Scalability

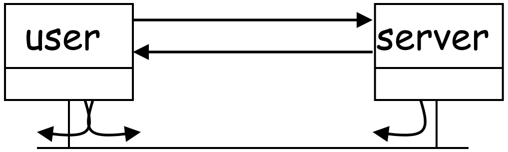
Addressing Issues

- *Question: how is the server located?*

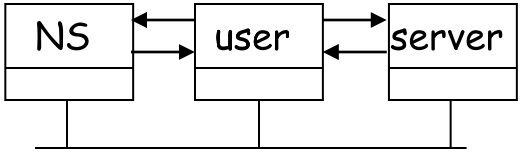
- Hard-wired address



- Broadcast-based



- Locate address via name server



Blocking versus Non-blocking

- Blocking communication (synchronous)
  - Sender blocked until msg sent
- Non-blocking communication (asynchronous)
  - Returns control to sender once msg copied into buffer
    - Pro?
    - Con?
  - Sender may not modify msg until msg sent
- How does the sender know it can use the buffer?
  - copy into kernel space (overhead)
  - interrupt sender to inform msg sent (=> buffer available)

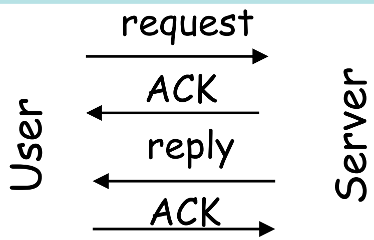
Buffering Issues

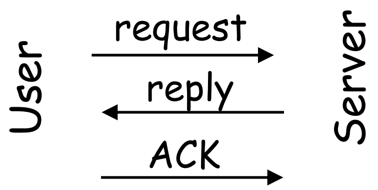
- Unbuffered communication
  - works well if “server calls receive before client calls send”!!!

- Buffered communication
  - Client send to a mailbox
  - Server receives from a mailbox

Reliability

- Unreliable channel
  - Need acknowledgements (ACKs)
  - Applications handle ACKs
  - ACKs for both request and reply





- Reliable channel
  - Reply acts as ACK for request
  - Explicit ACK for response

Reliability

- Reliable communication on unreliable channels
  - Transport protocol handles lost messages
- Reliability introduces overhead
  - Why?

Reliability

- Large size messages must be split and sent
- Packets get lost, arrive out-of-order
  - A packet number is assigned (seq no) - used to reassemble msgs
- How does the sender know if msg received?
- Acknowledgment-Options
  - Ack each packet
    - Pro?
    - Con?
  - Ack each message
    - Pro?
    - Con?
- Options depend on network characteristics

Server Architecture

- Sequential
  - Serve one request at a time
  - Can serve multiple requests by employing events and asynchronous communication
- Concurrent
  - Server spawns a process or thread to service each request
  - Can also use a pre-spawned pool of threads/processes (apache)
- Thus servers could be
  - Pure-sequential, event-based, thread-based, process-based

Scalability

- *Question:How can you scale the server capacity?*
  - Buy bigger machine!
  - Hide communication latency
  - Distribution
  - Replication (caching)
  - …

To Push or Pull ?

- Client-pull architecture
  - Clients pull data from servers (by sending requests)
  - Example: HTTP
  - Pro: ?
  - Con: ?

- Server-push architecture
  - Servers push data to client
  - Example: video streaming, stock tickers
  - Pro: ?
  - Con:?

- When/how-often to push or pull?

Summary

- Architectural styles
- System architectures
- Discussion on Client-Server Model
- Readings
  - Chapter

Questions

?