DISTRIBUTED SYSTEMS CS6421
DISTRIBUTED ARCHITECTURE

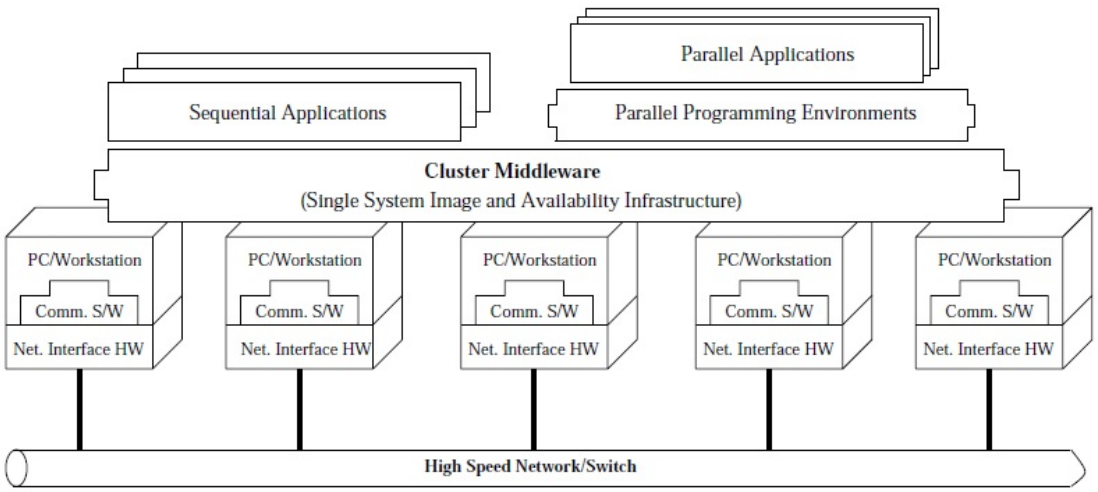
Prof. Tim Wood and Prof. Roozbeh Haghnazar

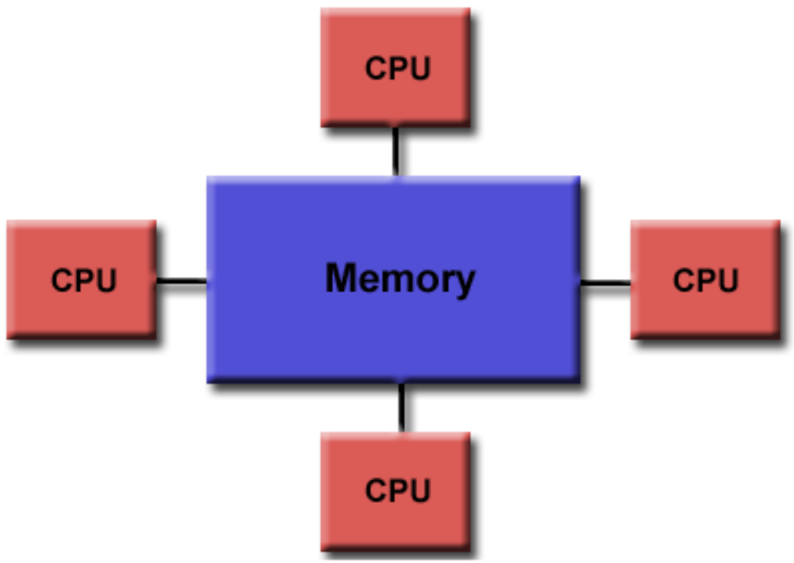
Includes material adapted from Van Steen and Tanenbaum's Distributed Systems book

TYPES OF DISTRIBUTED SYSTEMS

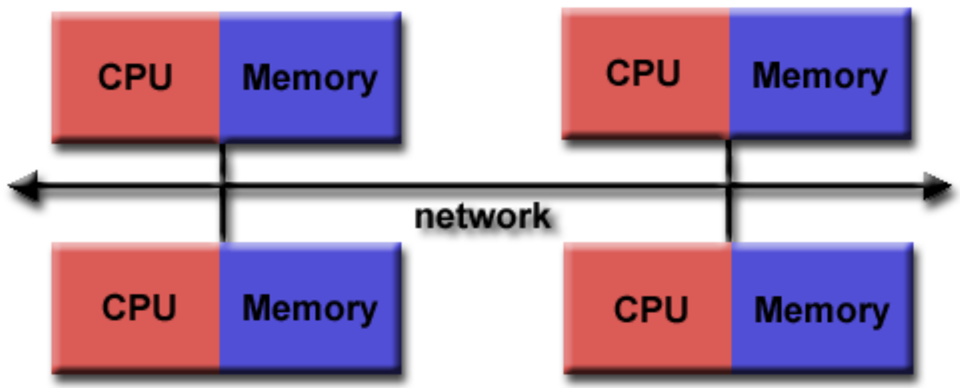
- Distributed Computing Systems
  - Clusters
  - Grids
- Distributed Information Systems
  - Transaction Processing Systems
  - Enterprise Application Integration
- Distributed Embedded Systems
  - Home systems
  - Health care systems
  - Sensor networks

CLUSTER COMPUTING





Shared Memory: Uniform Memory Access Obtained from www.computing.llnl.gov



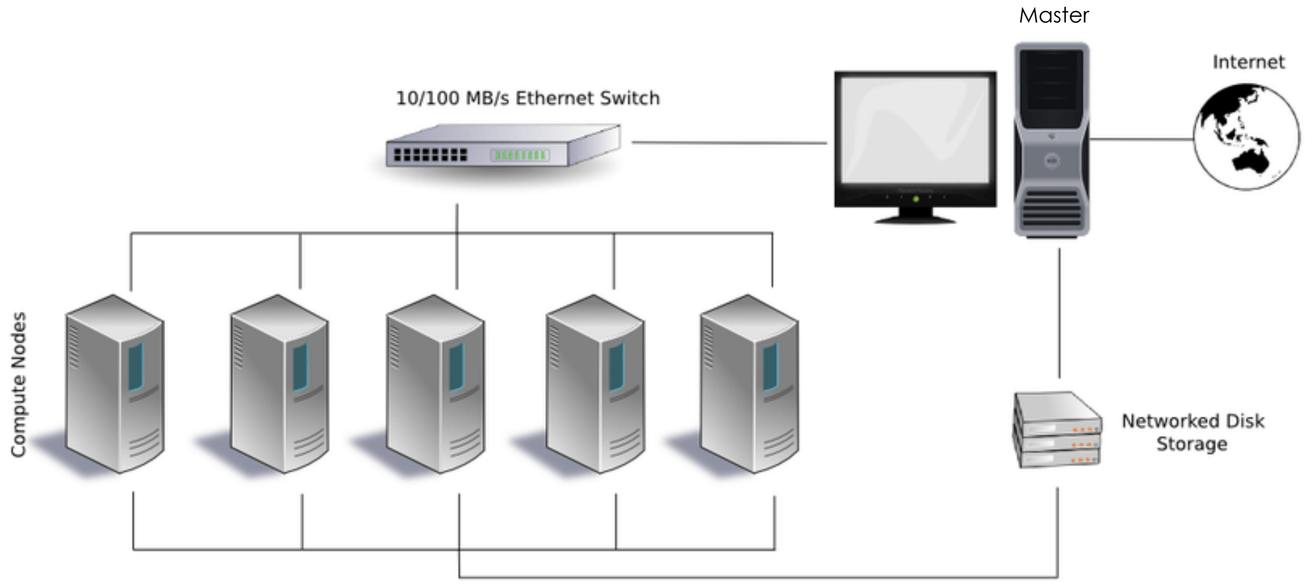
Distributed Memory System Obtained from www.computing.llnl.gov

CLUSTERS CLASSIFICATIONS

- High Performance
- Expandability and Scalability
- High Throughput
- High Availability

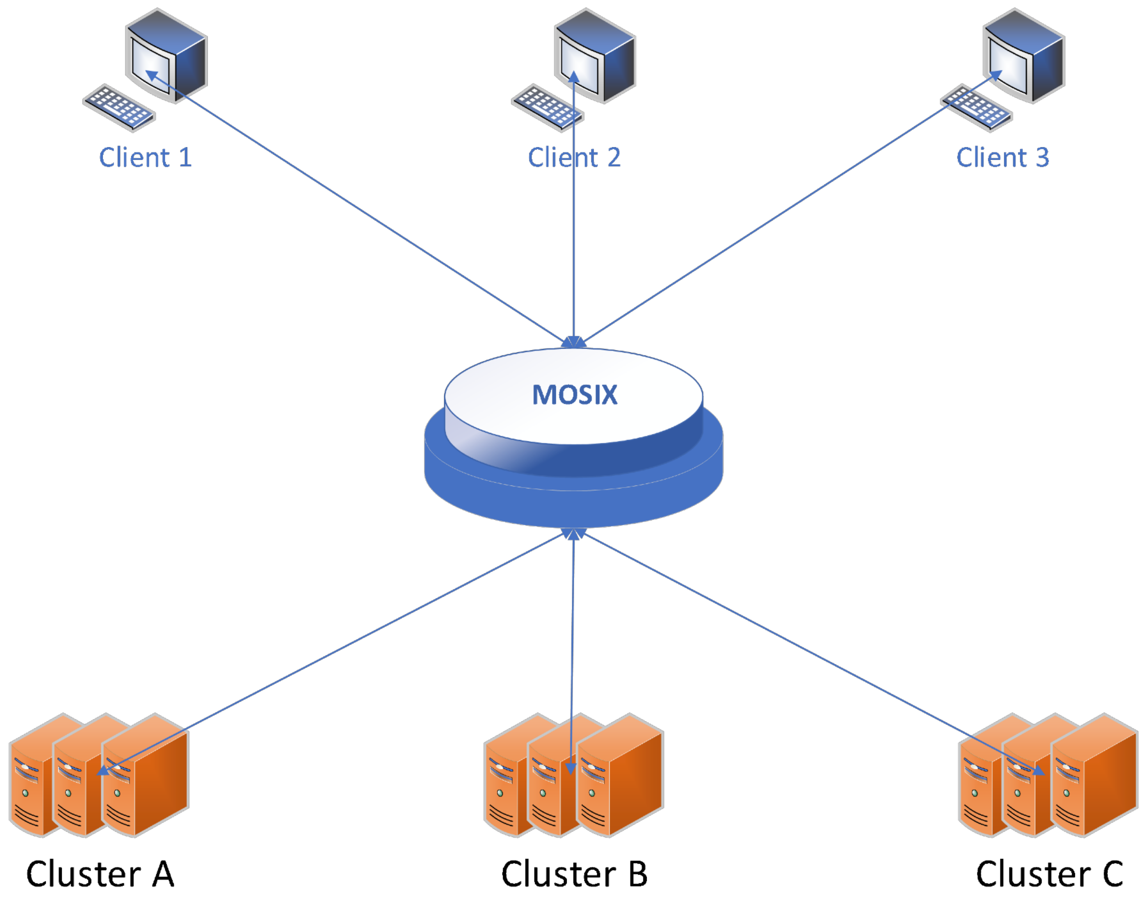
CLUSTERS – BEOWULF MODEL

- Master-slave paradigm
  - One processor is the master; allocates tasks to other processors, maintains batch queue of submitted jobs, handles interface to users
  - Master has libraries to handle message-based communication or other features (the middleware).
- Proper for parallel programs



CLUSTERS – MOSIX MODEL

- Provides a **symmetric**, rather than **hierarchical** paradigm
    - Single system image simplifies deployment
    - Processes can migrate between nodes dynamically
- "Operating-system-like"; looks & feels like a single computer with multiple processors
    - Provides resource discovery and and automatic workload distribution among clusters



GRID COMPUTING SYSTEMS

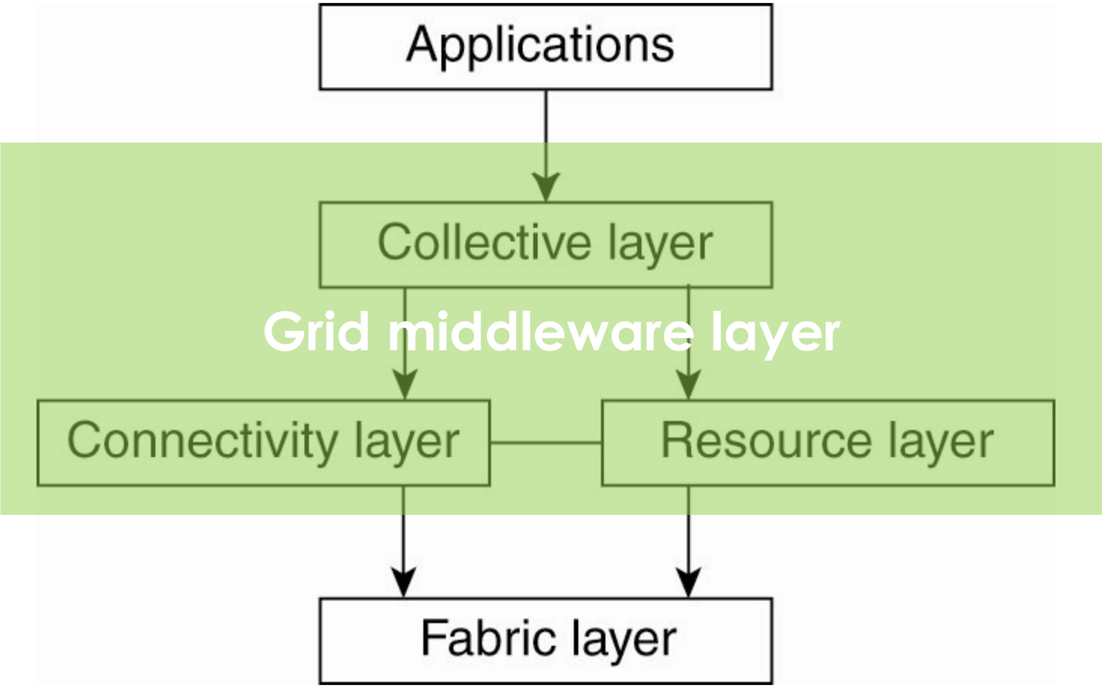
Highly heterogeneous with respect to hardware, software, networks, security policies, etc.

Grids support **virtual organizations**: a collaboration of users who pool resources (servers, storage, databases) and share them

Grid software is concerned with managing sharing across administrative domains.

A PROPOSED ARCHITECTURE FOR GRID SYSTEMS

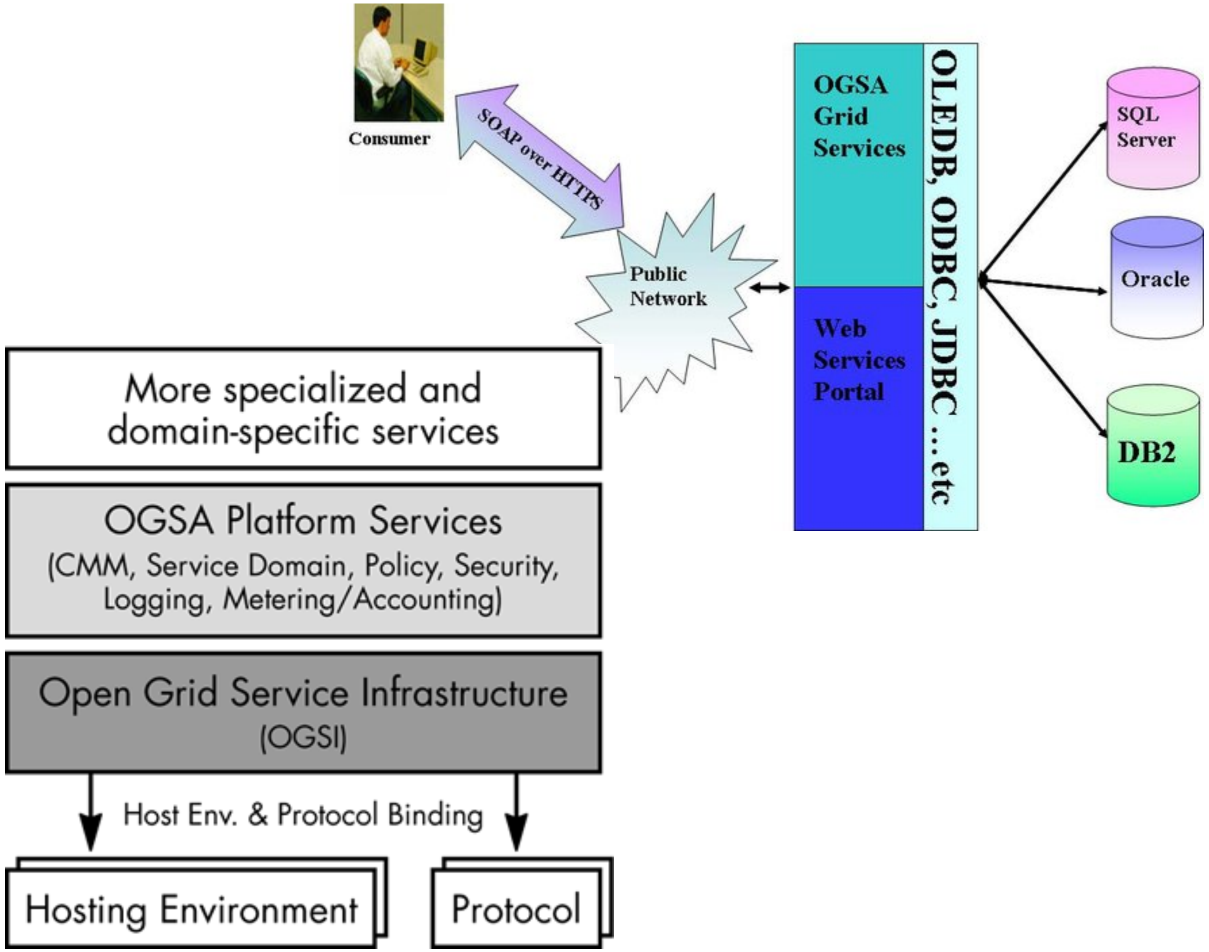
- **Fabric layer**: interfaces to local resources
- **Connectivity layer**: supports usage of *multiple resources* for a single application; e.g., access a remote resource or transfer data between sites
- **Resource layer** manages a *single resource*
- **Collective layer:** resource discovery, allocation, etc.
- **Applications**: use the grid resources
- The collective, connectivity and resource layers together form the middleware layer for a grid



. A layered architecture for grid computing systems

OGSA – ANOTHER GRID ARCHITECTURE

- Open Grid Services Architecture (OGSA) is a service-oriented architecture
  - Sites that offer resources to share do so by offering specific Web services.
- The architecture of the OGSA model is more complex than the previous layered model.



TYPES OF DISTRIBUTED SYSTEMS

- Distributed Computing Systems
  - Clusters
  - Grids
- Distributed Information Systems
  - Transaction Processing Systems
  - Enterprise Application Integration
- Distributed Embedded Systems
  - Home systems
  - Health care systems
  - Sensor networks

DISTRIBUTED INFORMATION SYSTEMS

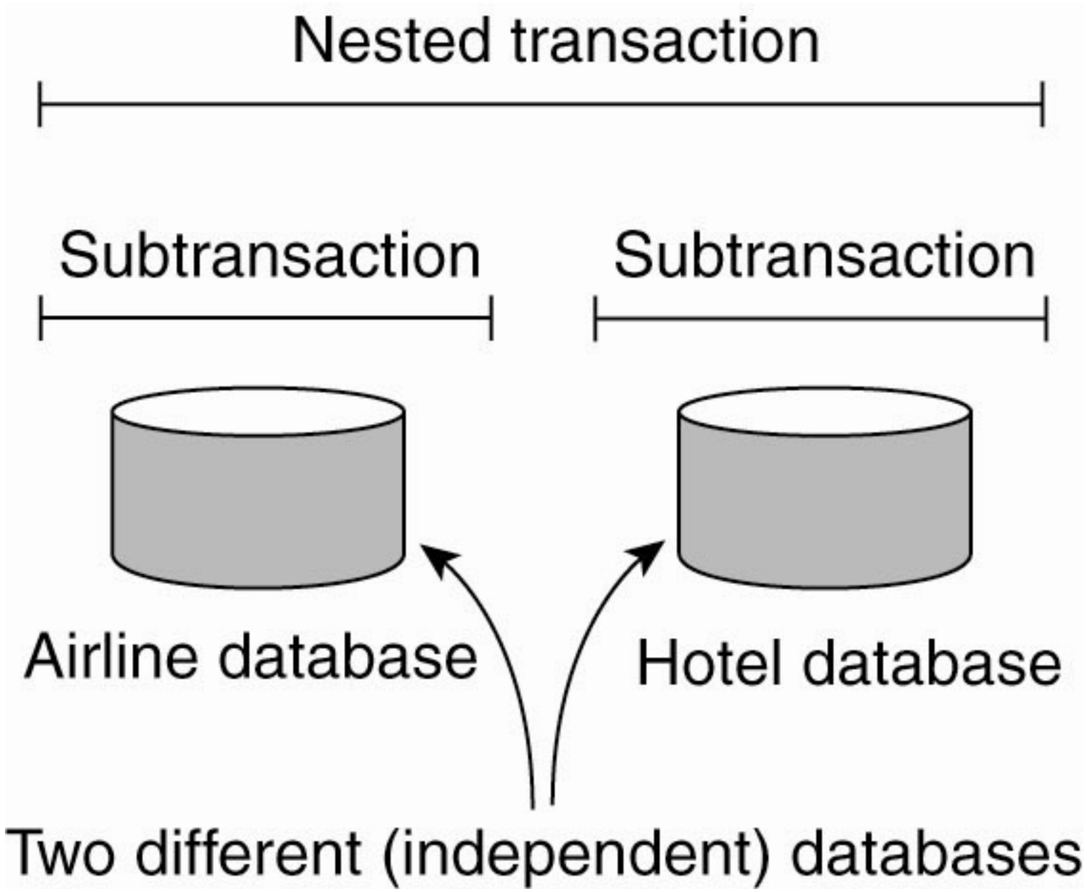
- Business-oriented
- Systems to make a number of separate network applications interoperable and build “enterprise-wide information systems”.
- Two types discussed here:
  - Transaction processing systems
  - Enterprise application integration

TRANSACTION PROCESSING SYSTEMS

- Provide a highly structured client-server approach for database applications
- Transactions are the communication model
- Obey the ACID properties:
  - Atomic: all or nothing
  - Consistent: invariants are preserved
  - Isolated (serializable)
  - Durable: committed operations can't be undone

Primitive	Description
BEGIN_TRANSACTION	Mark the start of a transaction
END_TRANSACTION	Terminate the transaction and try to commit
ABORT_TRANSACTION	Kill the transaction and restore the old values
READ	Read data from a file, a table, or otherwise
WRITE	Write data to a file, a table, or otherwise

TRANSACTION PROCESSING SYSTEMS



A nested transaction.

IMPLEMENTING TRANSACTIONS

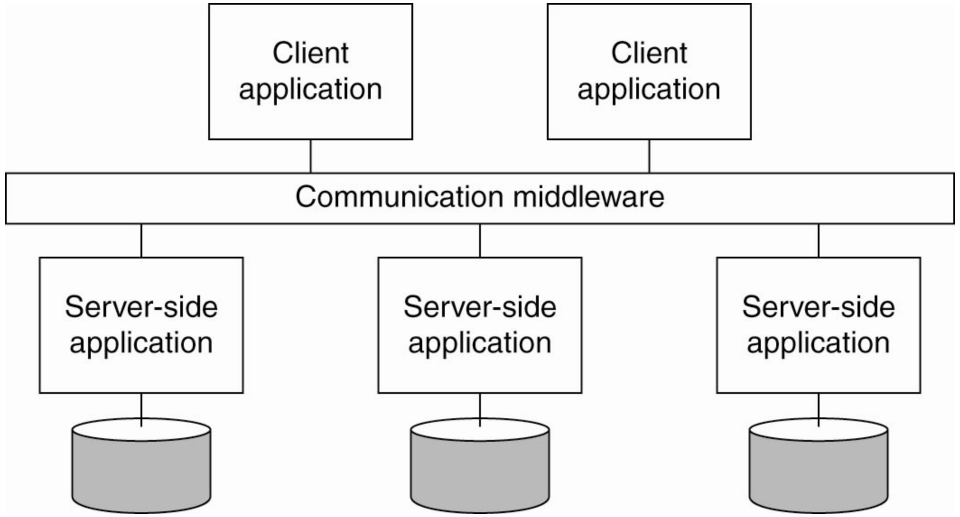
- Conceptually, private copy of all data
- Actually, usually based on logs
- Multiple sub-transactions – commit, abort
  - Durability is a characteristic of top-level transactions only
- Nested transactions are suitable for distributed systems
  - Transaction processing monitor may interface between client and multiple data bases.

ENTERPRISE APPLICATION INTEGRATION

- Supports a less-structured approach (as compared to transaction-based systems)
- Application components are allowed to communicate directly
- Communication mechanisms to support this include CORBA, Remote Procedure Call (RPC), Remote Method Invocation (RMI), and Message-Oriented middleware (MOM).

Examples?

Tell some software architectures that can be applied on this model



Middleware as a communication facilitator in enterprise application integration.

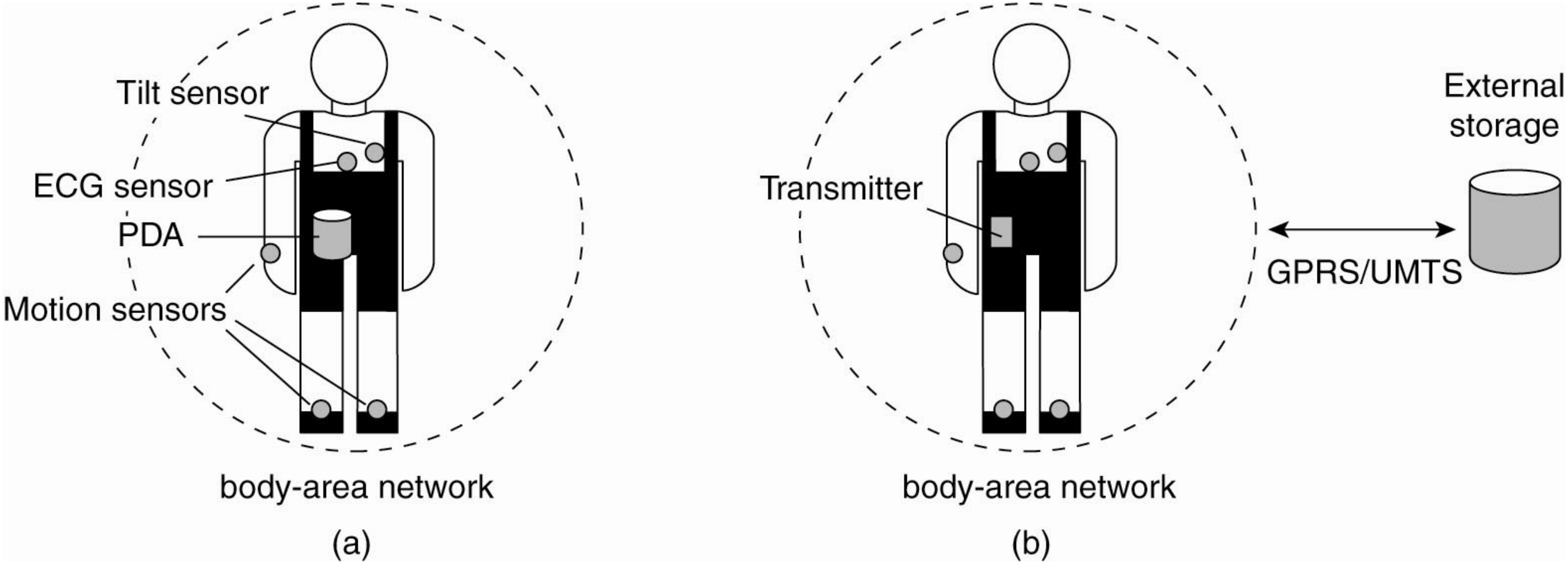
TYPES OF DISTRIBUTED SYSTEMS

- Distributed Computing Systems
  - Clusters
  - Grids
- Distributed Information Systems
  - Transaction Processing Systems
  - Enterprise Application Integration
- Distributed Embedded Systems
  - Home systems
  - Health care systems
  - Sensor networks

DISTRIBUTED PERVASIVE SYSTEMS

- The first two types of systems are characterized by their stability: nodes and network connections are more or less fixed
- This type of system is likely to incorporate small, battery-powered, mobile devices
  - Home systems
  - Electronic health care systems – patient monitoring
  - Sensor networks – data collection, surveillance

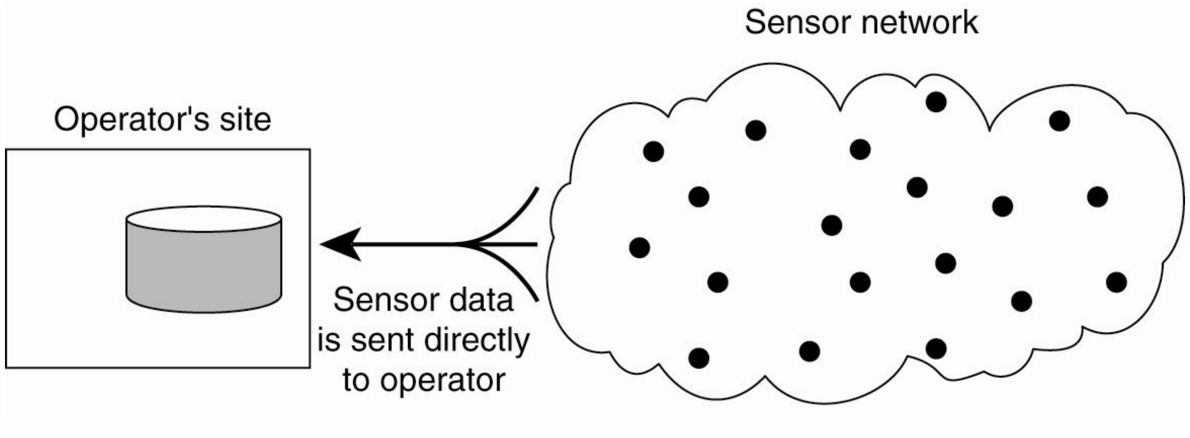
ELECTRONIC HEALTH CARE SYSTEMS



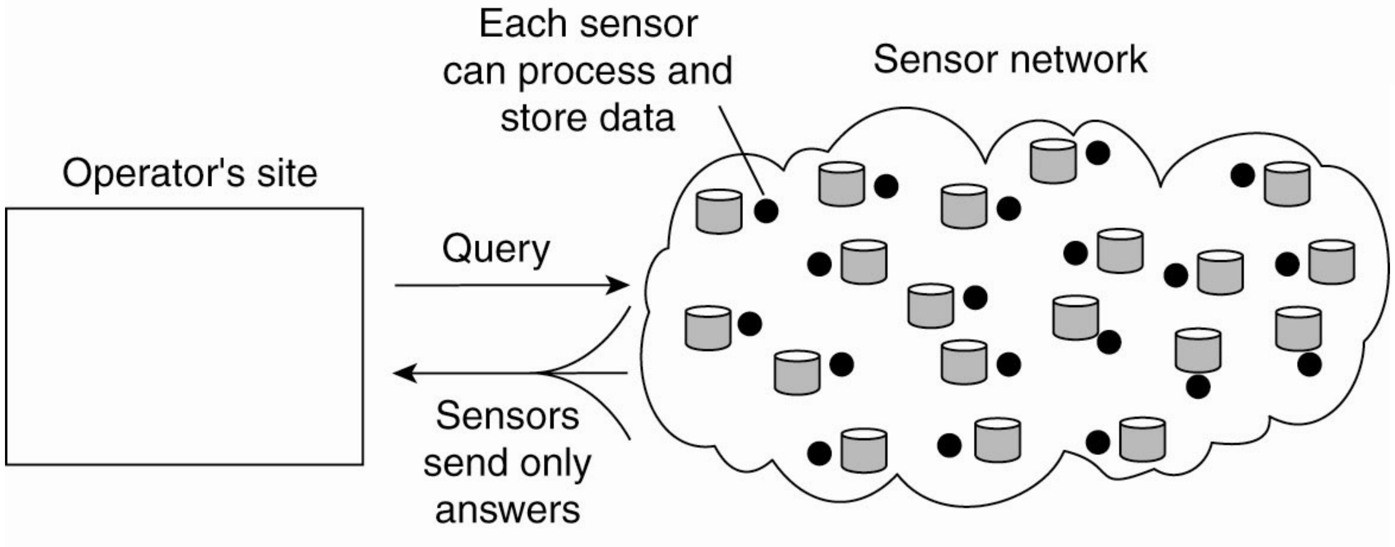
Monitoring a person in a pervasive electronic health care system, using (a) a local hub or (b) a continuous wireless connection.

SENSOR NETWORKS

Organizing a sensor network database, while storing and processing data (a) only at the operator's site or (b) only at the sensors.



(a)



(b)

ARCHITECTURES

DEFINITION OF *ARCHITECTURE*

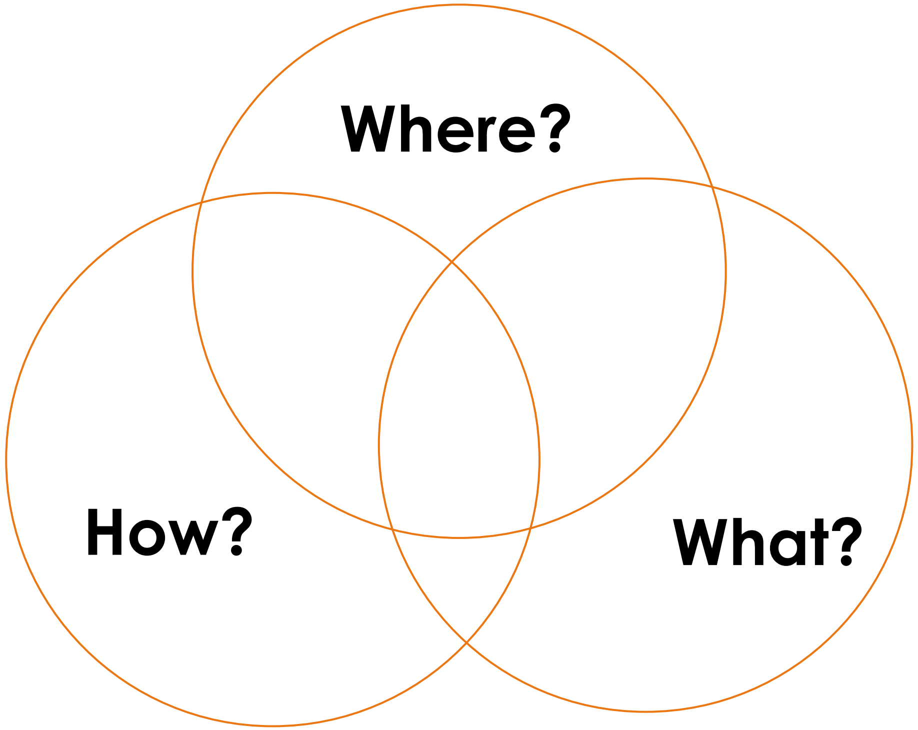
- The art or science of building
  - *specifically* **:** the art or practice of designing and building structures and especially habitable ones
- Formation or construction resulting from or as if from a conscious act or a unifying or coherent form or structure
- A method or style of building
- The manner in which the components of a computer or computer system are organized and integrated

SOFTWARE/SYSTEM ARCHITECTURE

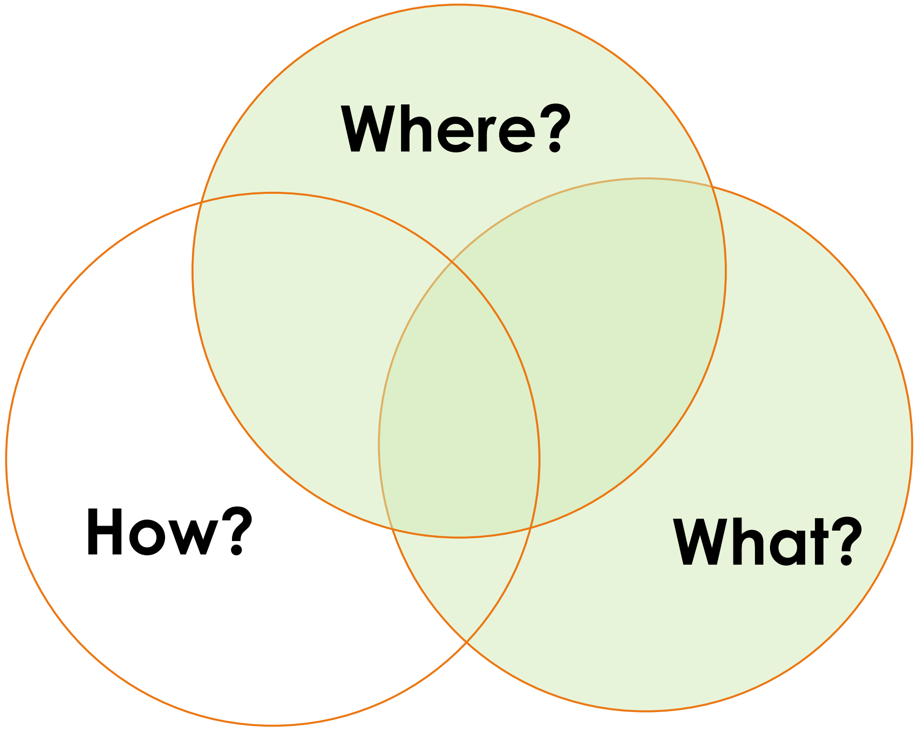
Software Architectures – describe the organization and interaction of software components; focuses on logical organization of software (component interaction, etc.)

System Architectures - describe the communication and placement of software components on physical machines

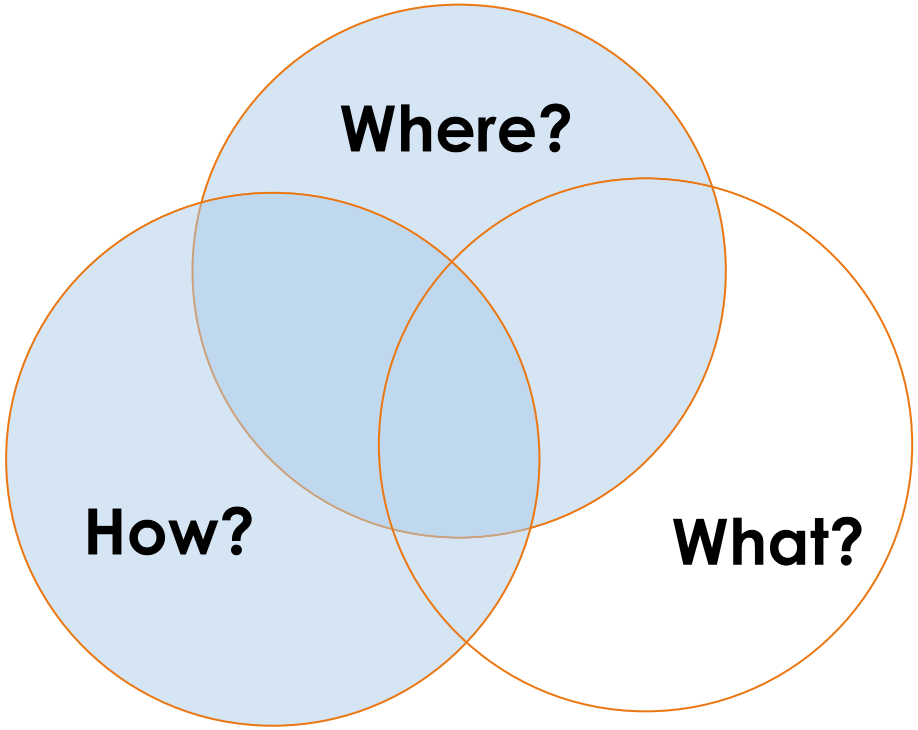
ARCHITECTURE VS DESIGN



ARCHITECTURE VS DESIGN



ARCHITECTURE VS DESIGN



COMPONENT

A component is an encapsulated part of a software system

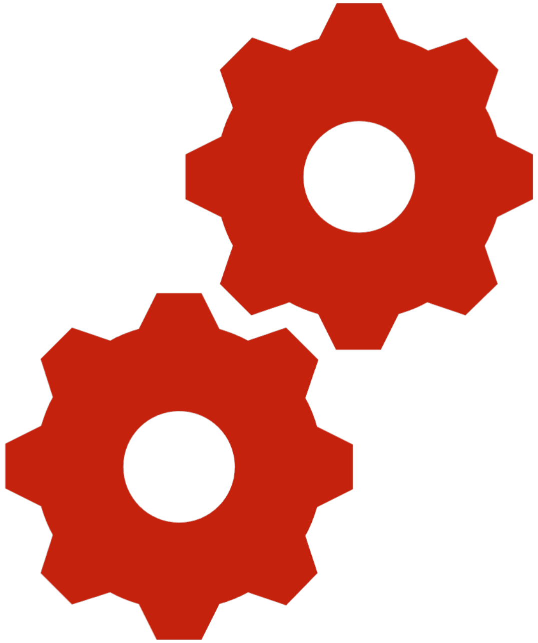
A component has an interface

Components serve as the building blocks for the structure of a system

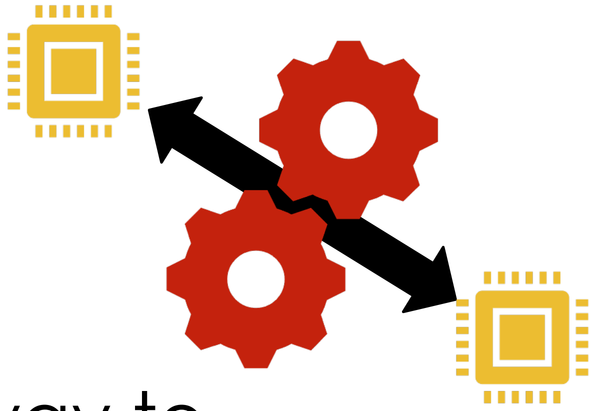
At a programming-language level, components may be represented as modules, classes, objects or as a set of related functions

SUBSYSTEM

- A subsystem is a set of collaborating components performing a given task
- A subsystem is considered a separate entity within a software architecture
  - It performs its designated task by interacting with other subsystems and components…



ARCHITECTURAL STYLES



- An **architectural style** describes a particular way to configure a collection of components and connectors.
  - **Component** - a module with well-defined interfaces; reusable, replaceable
  - **Connector** – communication link between modules

- An architectural style is a coordinated set of architectural **constraints** that restricts the relationships among those elements

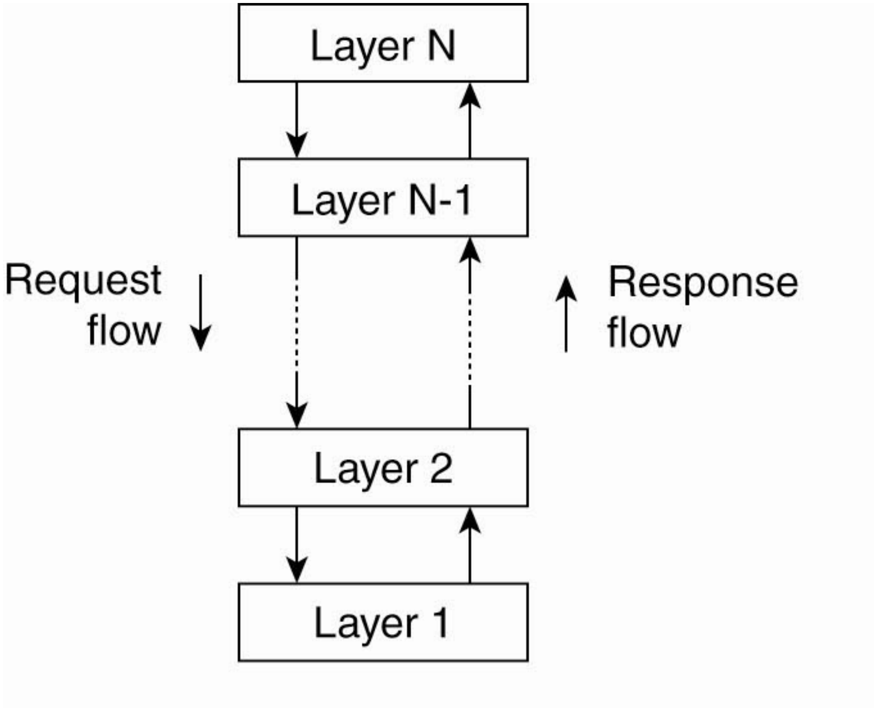
ARCHITECTURAL STYLES

1. Layered architectures
2. Object-based architectures
3. Data-centered architectures
4. Event-based architectures

1. LAYERED ARCHITECTURES

- Components of layer N_i is only allowed to call components at the underlying layer N_{i-1}

Why? Example?

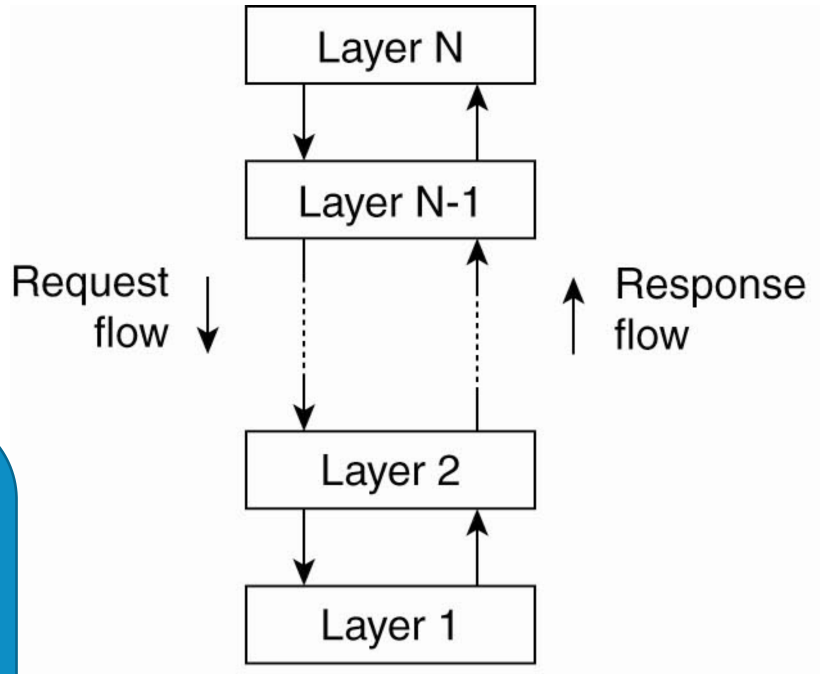


(a)

1. LAYERED ARCHITECTURES

- Components of layer N_i is only allowed to call components at the underlying layer N_{i-1}

Why: hides information, interchangeable layers
Example: Network stack, LAMP

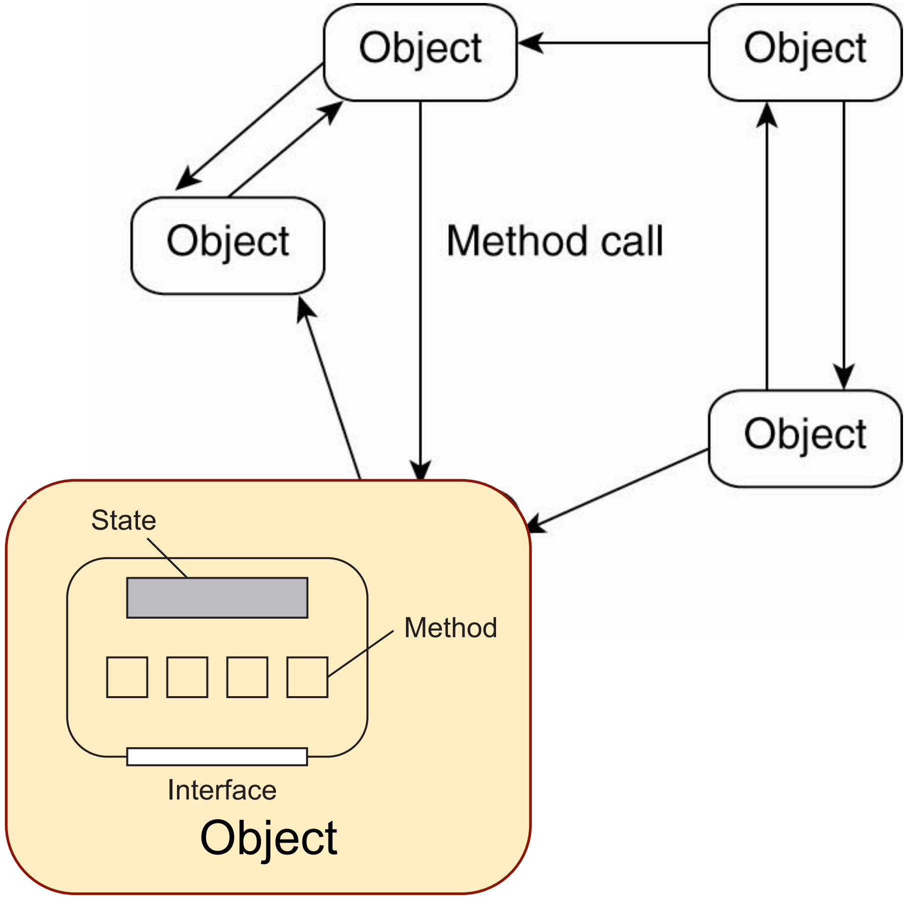


(a)

2. OBJECT-BASED ARCHITECTURES

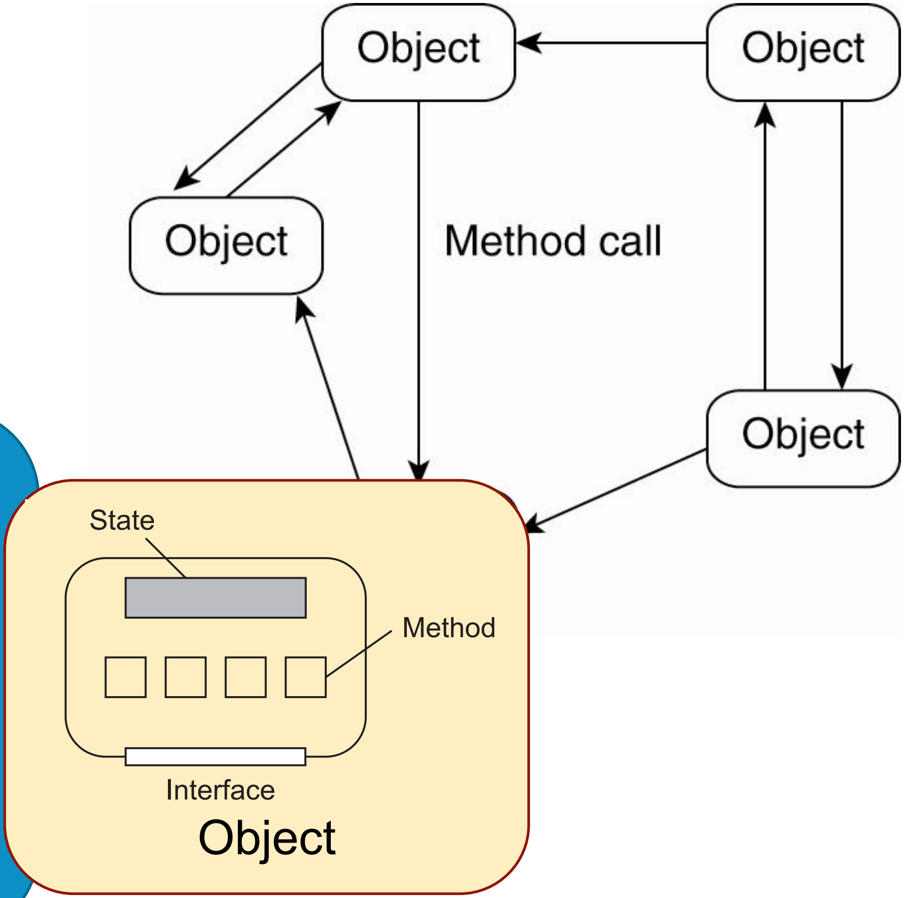
- Each object is a component that encapsulates data and methods
- They are connected through a well defined remote API that hides internals

Why? Example?



2. OBJECT-BASED ARCHITECTURES

- Each object is a component that encapsulates data and methods
- They are connected through a well defined remote API that hides internals

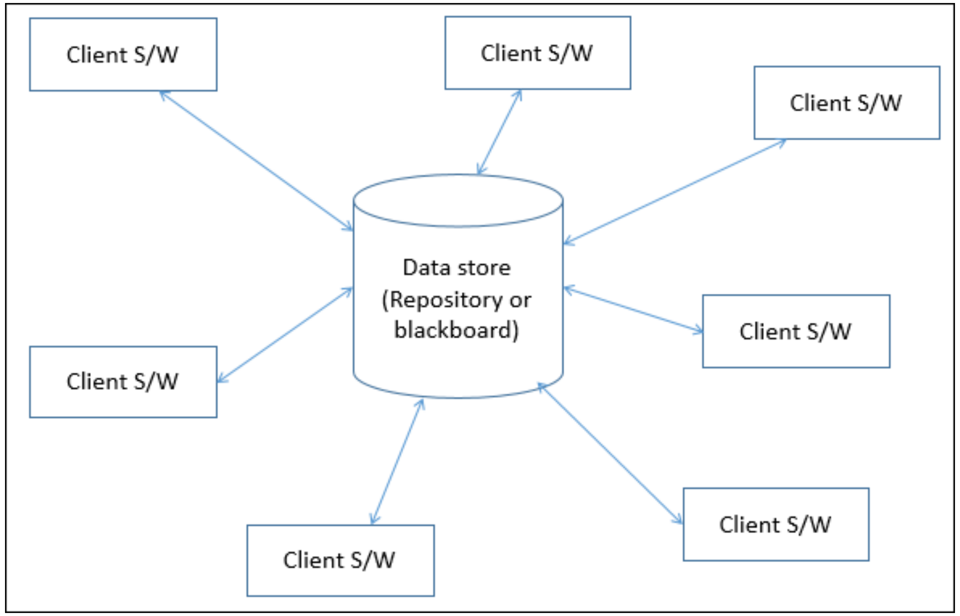


Why: components can be individually scaled/developed/managed
Example: MapReduce, microservice web architectures

3. DATA-CENTERED ARCHITECTURES

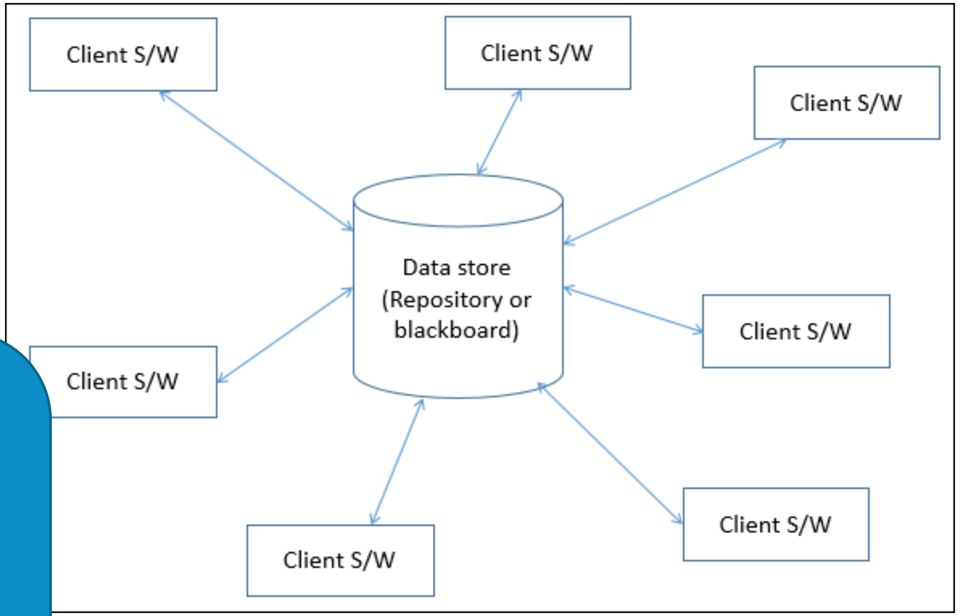
- Main purpose: data access and update
- Processes interact by reading and modifying data in a **centralized** shared repository

Why? Example?



3. DATA-CENTERED ARCHITECTURES

- Main purpose: data access and update
- Processes interact by reading and modifying data in a **centralized** shared repository

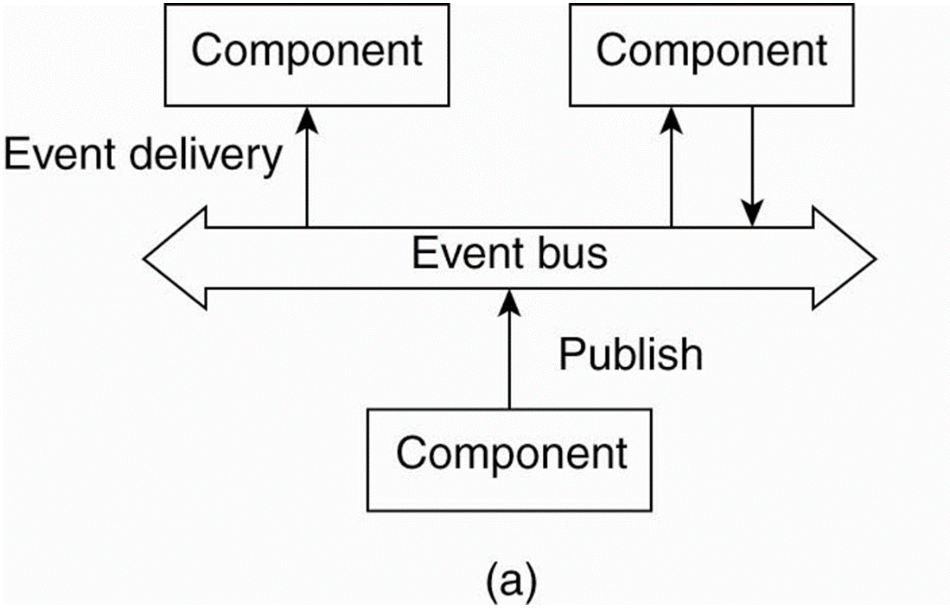


Why: simplifies data management
Example: Dropbox, Message board systems, Email

4. EVENT-BASED ARCHITECTURES

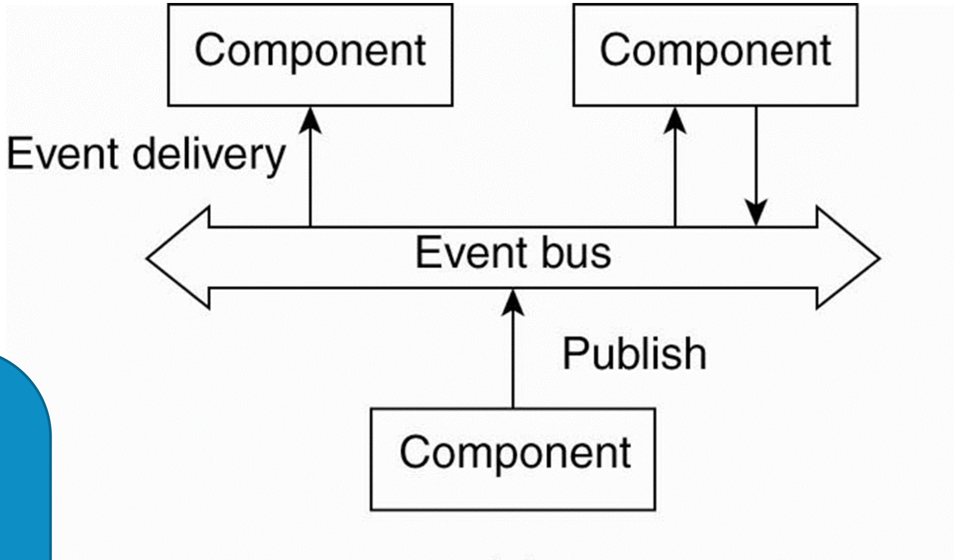
- Communication via event propagation
  - Publish-subscribe
  - Broadcast
  - Point-to-point

Why? Example?



4. EVENT-BASED ARCHITECTURES

- Communication via event propagation
  - Publish-subscribe
  - Broadcast
  - Point-to-point



(a)

Why: decouples sender/receiver, asynchronous
Example: Slack, Security monitoring

ARCHITECTURAL STYLES

1. Layered architectures
2. Object-based architectures
3. Data-centered architectures
4. Event-based architectures

Each style constrains how we will build the system. Following a style makes development and extensibility easier.

But sometimes we need a **hybrid** style!

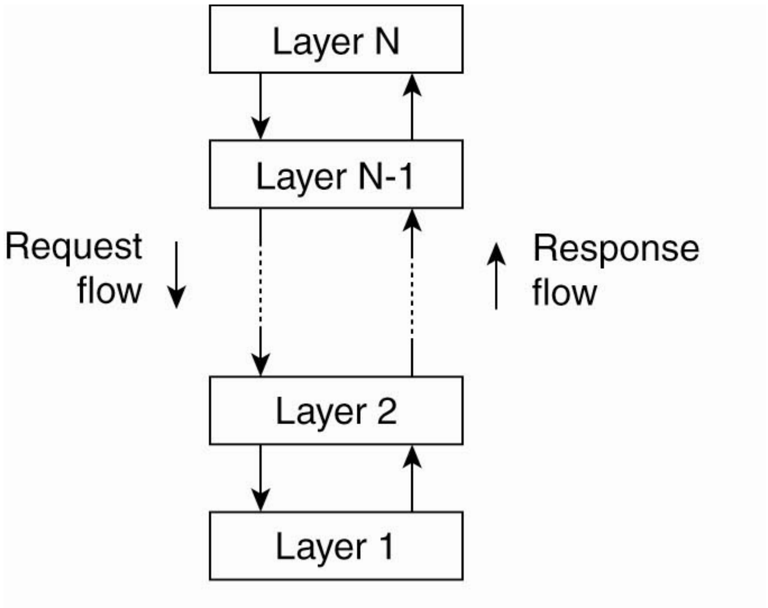
SYSTEM CHARACTERISTICS

- **Centralized**: A single component/subsystem is "in charge"
  - **Vertical** (or hierarchical) organization of communication and control paths
  - Logical separation of functions into client (requester) and server (responder)
- **Decentralized**: multiple components/subsystems interact as peers
  - **Horizontal** rather than hierarchical communication and control
  - Communication paths may be less structured; symmetric functionality
- **Hybrid:** combine elements of both

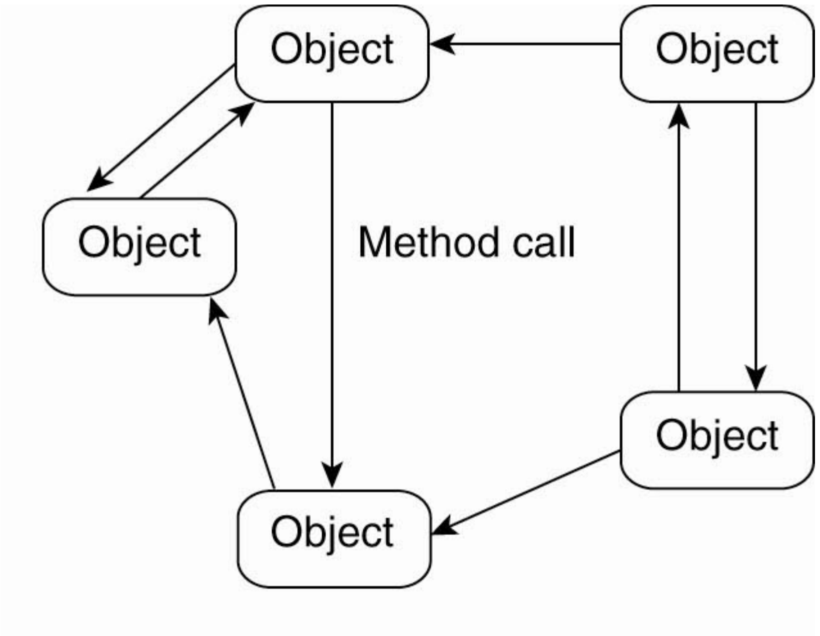
Classification of a system as **centralized** or **decentralized** primarily refers to **communication** and **control organization**

VERTICAL VS HORIZONTAL

- Vertical: Layers with different functionality. Restricted communication

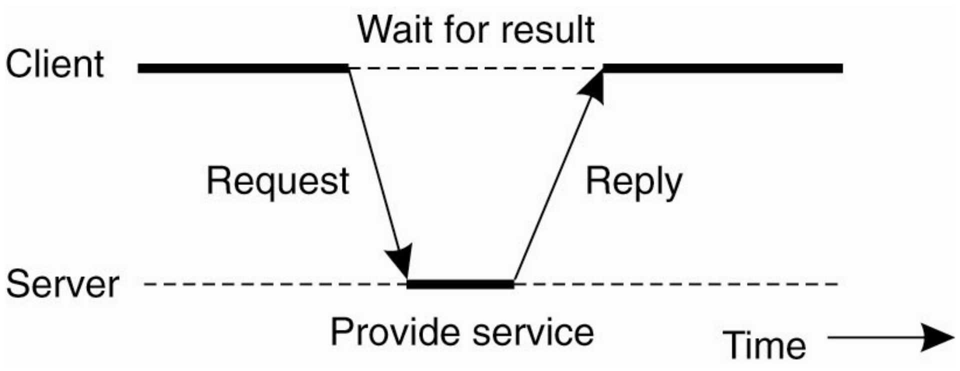


- Horizontal: Components with similar functionality or more diverse communication



TRADITIONAL CLIENT-SERVER

- Processes are divided into two groups (clients and servers).
- Synchronous communication: request-reply protocol
  - Could be message oriented or RPC



- Note: even in this simple example, lots could go wrong!

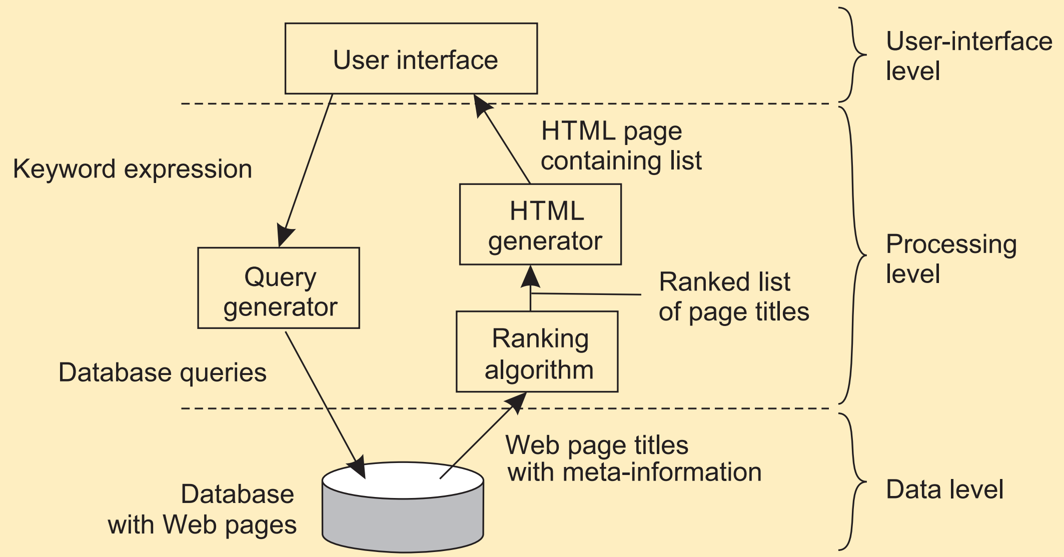
CLIENT ARCHITECTURE

- Server provides processing and data management; client provides simple graphical display (**thin-client**)
  - Pro: Easier to manage, more reliable, client machines don’t need to be so large and powerful
  - Con: Potential performance loss at client
- At the other extreme, all application processing and some data resides at the client (**fat-client** approach)
  - Pro: reduces workload at server; more scalable
  - Con: harder to manage by system admin, less secure

LAYERED SERVER EXAMPLE

- **User-interface level**: GUI's (usually) for interacting with end users
- **Processing level**: data processing applications – the core functionality
- **Data level**: interacts with data base or file system

Example: a simple search engine



TIERS, LAYERS, NODES, COMPONENTS

- ***Layer*** and ***tier*** are roughly equivalent terms, but *layer* typically implies software and *tier* is more likely to refer to component deployment on HW.
  - Several software layers might comprise a subsystem deployed as a single "tier" in a multi-tier web application
- **Components** are generally software, whereas a **node** could refer to a component deployed on a particular server

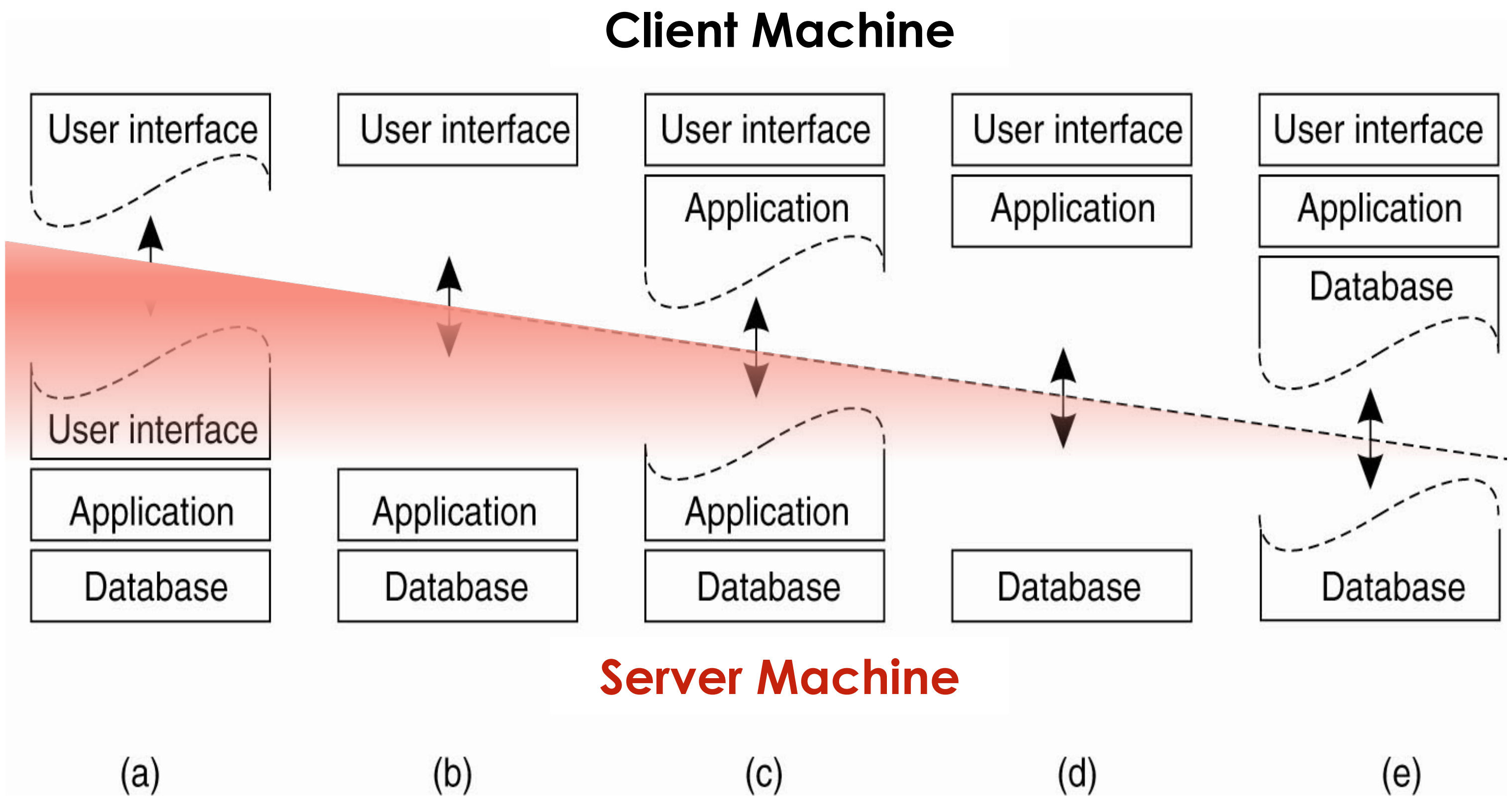
Layers / Components = Software

Tiers / Nodes = Software deployed on hardware

(usually*)

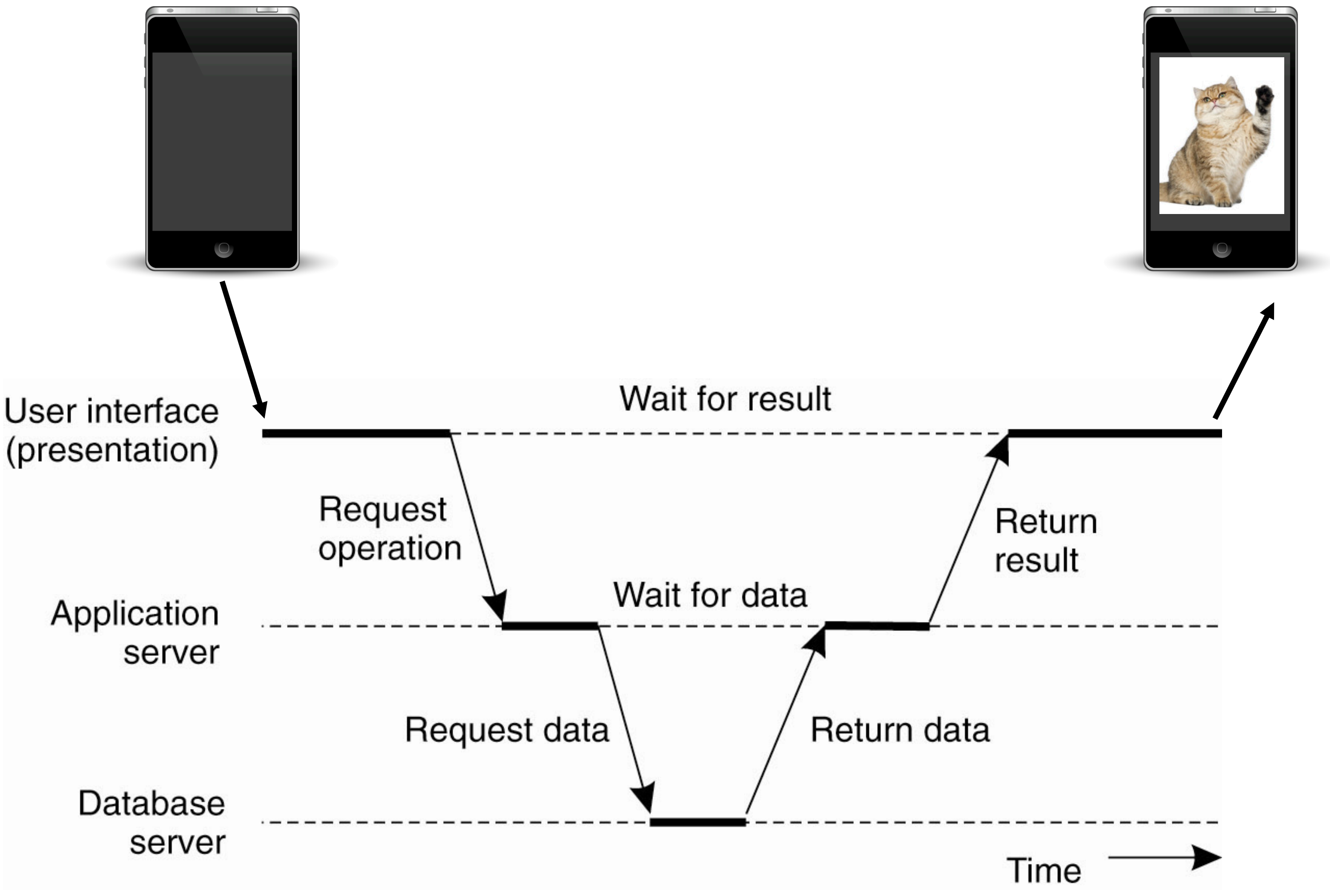
CLIENT-SERVER SPLIT

Can you come up with an example service/application which uses each of these architectures?



THREE-TIERED WEB ARCHITECTURE

- Tiers can be spread across multiple servers
- Simplifies deployment, performance management, reliability
- Servers also can play the role of client



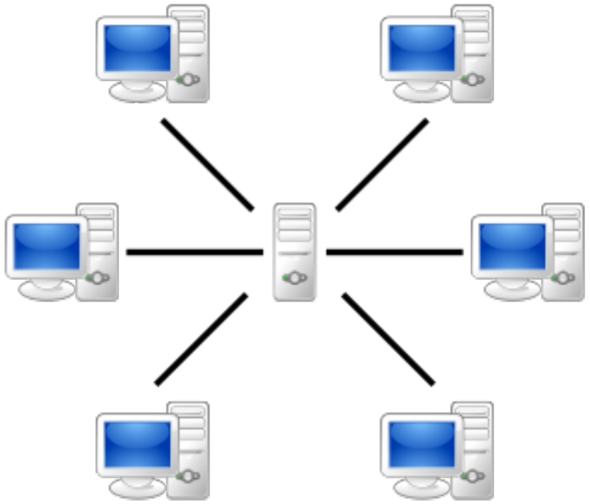
"LAMP Stack" (Linux, Apache, MySQL, PHP) was the 3-tier standard

Prof. Tim Wood & Prof. Roozbeh Haghnazar

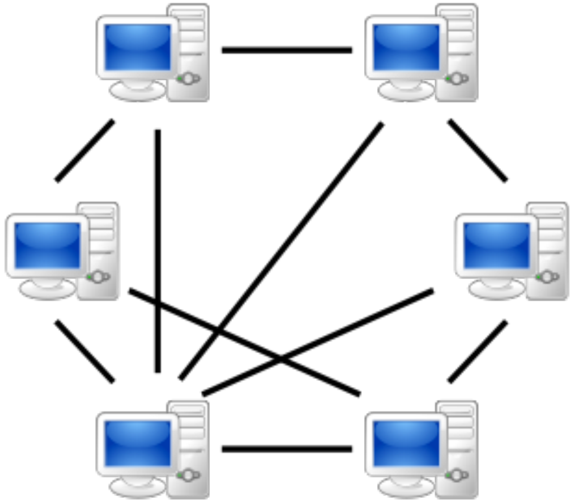
DECENTRALIZED ARCHITECTURES

PEER TO PEER SYSTEMS

- A distributed system that does not rely on centralized coordination
- Peers are *equipotent* and work together to provide a service



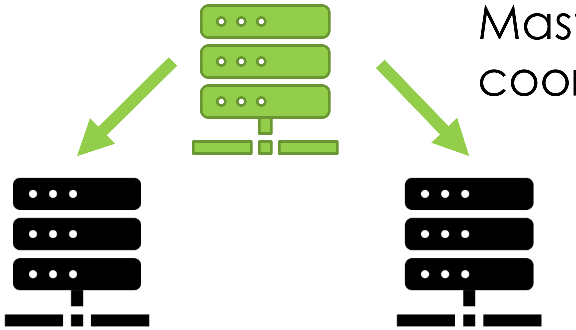
Centralized



P2P

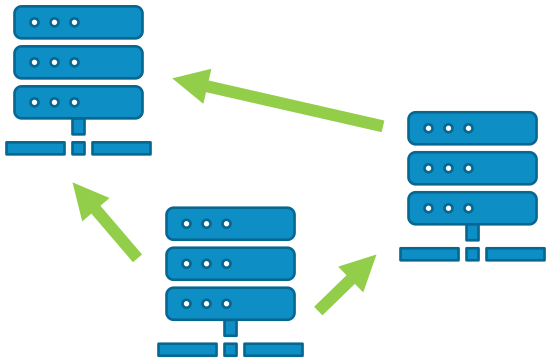
DECENTRALIZATION BENEFITS

- Centralized systems may have **a single point of failure**
  - Affects reliability and may be a performance bottleneck



Master/Leader node coordinates followers

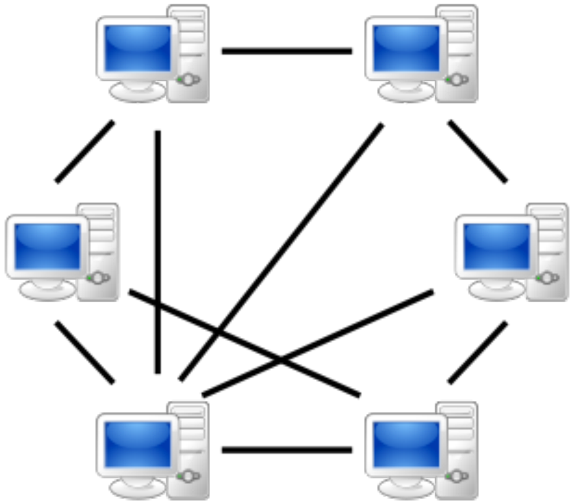
- Decentralization can make a system **more robust and performant**
  - But only if it is well designed!



Peer-to-Peer system can self-organize

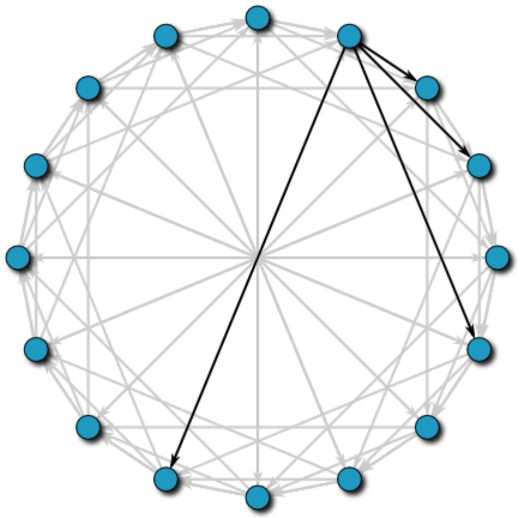
P2P CHALLENGES

- Routing and Discovery
  - How to reach other nodes?
  - How to find out what other nodes exist?
  - How to bootstrap when you first join?
- Consistency
  - How to keep information consistent across the network?
- Reliability / Failure Handling
  - What happens when nodes crash and rejoin?
- Performance
  - How to get predictable performance with limited control?

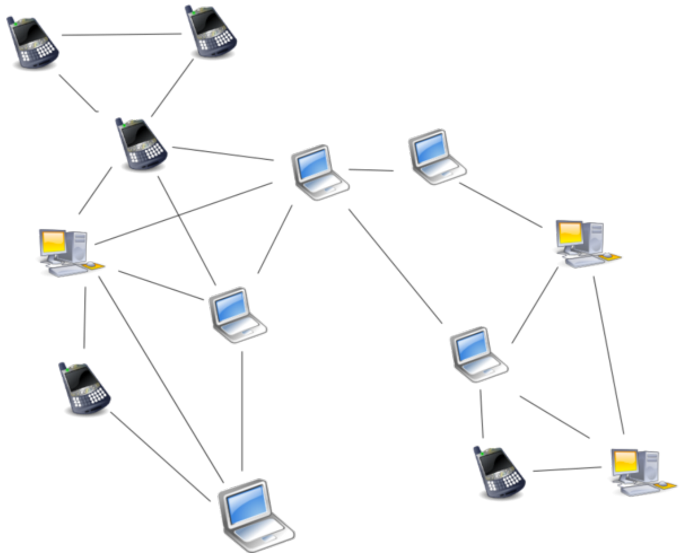


P2P ARCHITECTURES

Structured



Unstructured

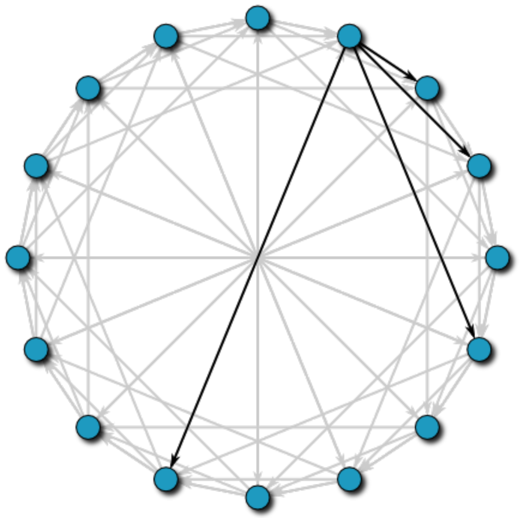


- The **peers** can be connected in a organized (**structured**) manner or in an ad-hoc (**unstructured**) manner
  - **Why/When might you choose one over the other?**

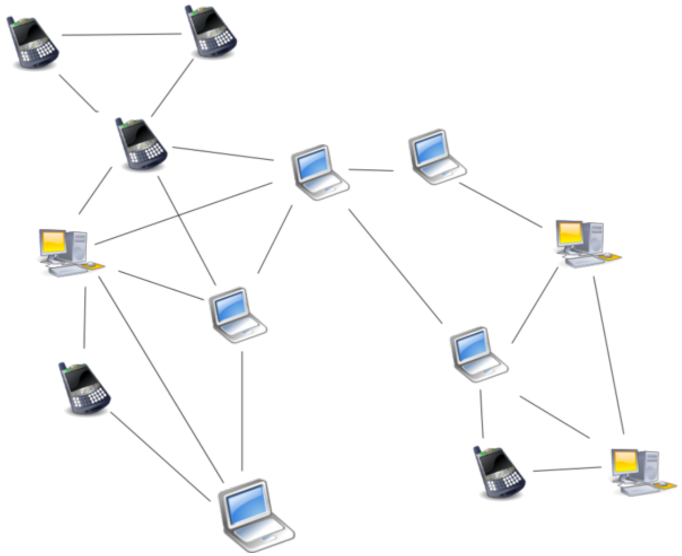
Figures from wikipedia

BOOTSTRAPPING

Structured



Unstructured

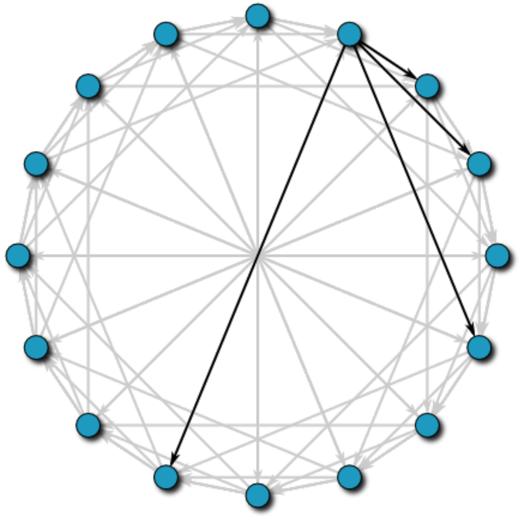


- How can a node join a P2P network if there is no centralized server to connect to?

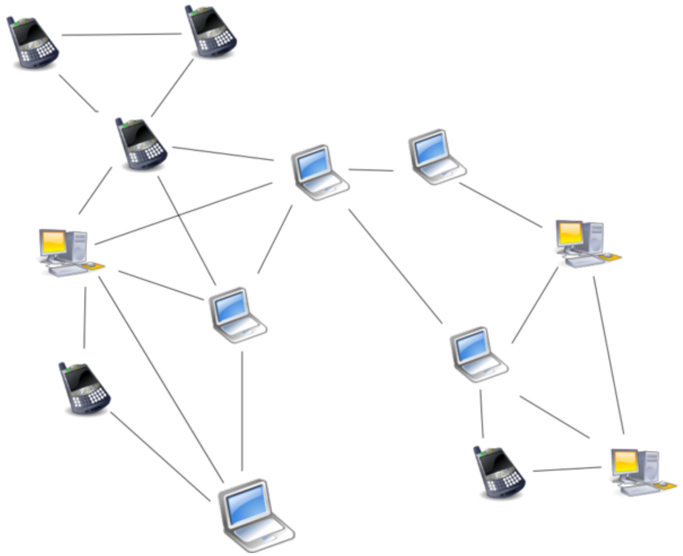
Figures from wikipedia

BOOTSTRAPPING

Structured



Unstructured

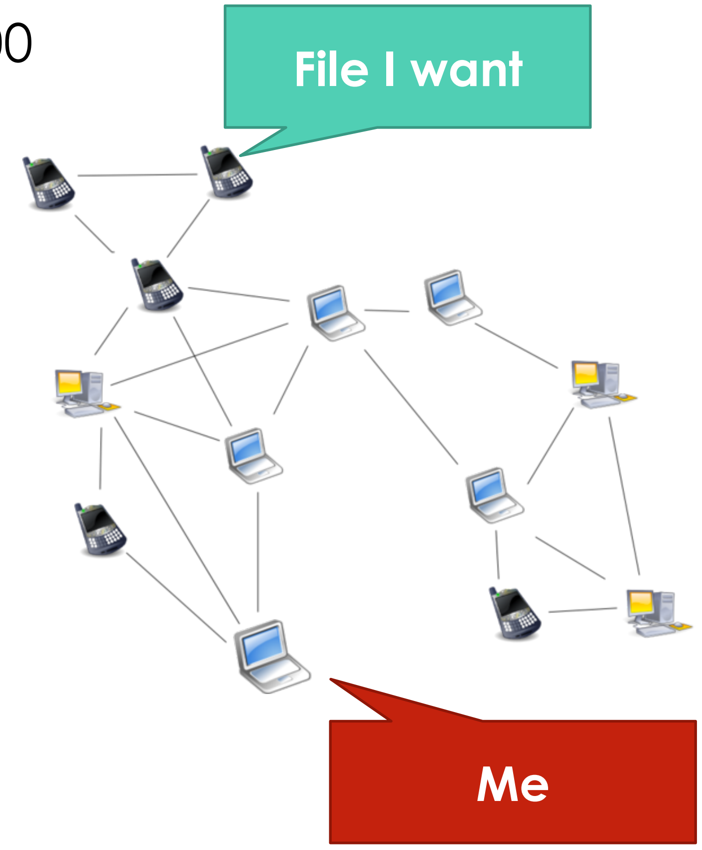


- Common Assumptions:
  - New nodes have address of at least one other active node
  - Special peers store extra information

- New nodes can broadcast on their radio to find close neighbors

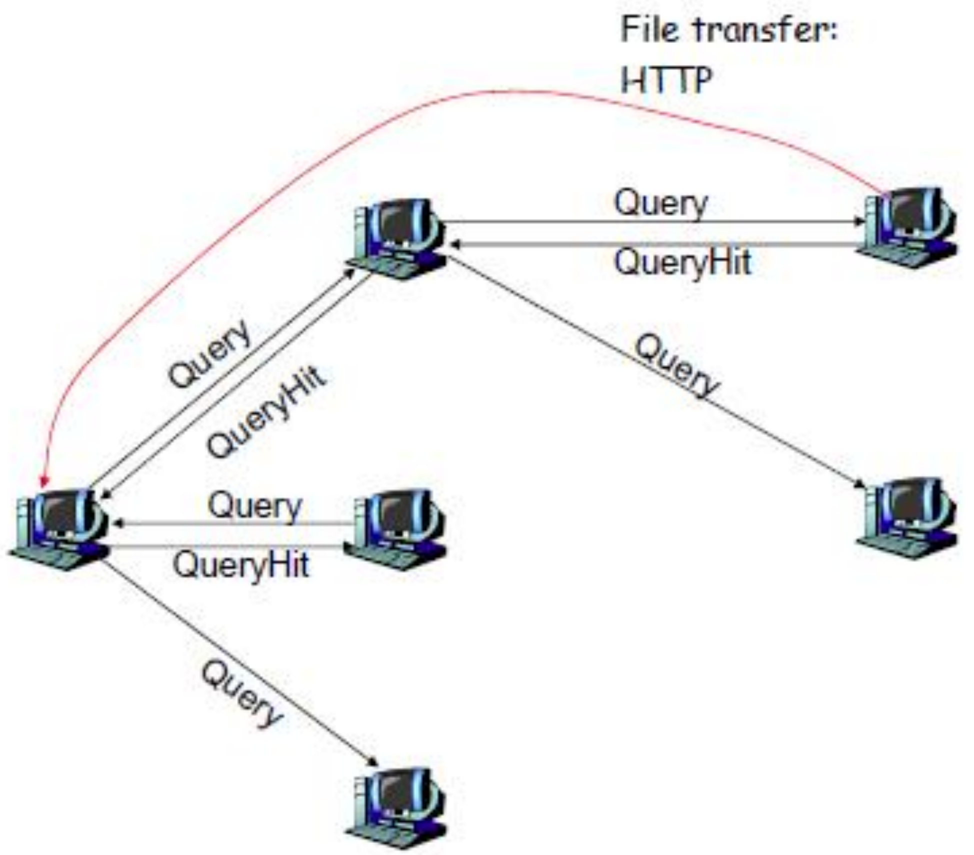
GNUTELLA P2P FILE SHARING

- Peer-to-Peer file sharing service
  - Released by a company owned by AOL on March 14, 2000
  - AOL shut down the company the next day…
- **Unstructured** P2P system
  - Bootstrap using pre-defined addresses of starter nodes
  - Randomly pick a set of N neighbors (N=5)
  - Search for files by querying neighbors
  - Neighbors propagate searches up to H hops total (H=7)
  - Responses travel back the same path
- Once file is found, transfer over direct connection



GNUTELLA

- At most how many neighbors will this search?
  - 5 neighbors per node
  - 7 hop max path
- This is a form of *flooding*
- What could make this more efficient?



Figures from wikipedia

NOT ALL PEERS ARE EQUAL

- Gnutella v0.6 added Ultra Peers and Leaves
- Leaf Node:
  - Connects to 3 Ultra Peers
  - Maintains an index of all its content
  - Send queries to Ultra Peer
- Ultra Peer:
  - Connects to 32 Ultra Peers
  - Forwards queries at most 4 hops (not 7)
  - Merges the content indexes of all leaf nodes
  - Shares content index with all adjacent Ultra Peers
  - Only send to an Ultra peer on the 4th hop if query is in index
- How does this change things?

How to Pick Neighbors?

- Want to avoid disconnected components and weak connectivity between groups!

- This is why some networks enforce a structure or hierarchy