Client/server and peer-to-peer models: basic concepts

Dmitri Moltchanov

Department of Communications Engineering
Tampere University of Technology
moltchan@cs.tut.fi

September 04, 2013

Slides provided by R. Dunaytsev, http://utopia.duth.gr/rdunayts/

Outline

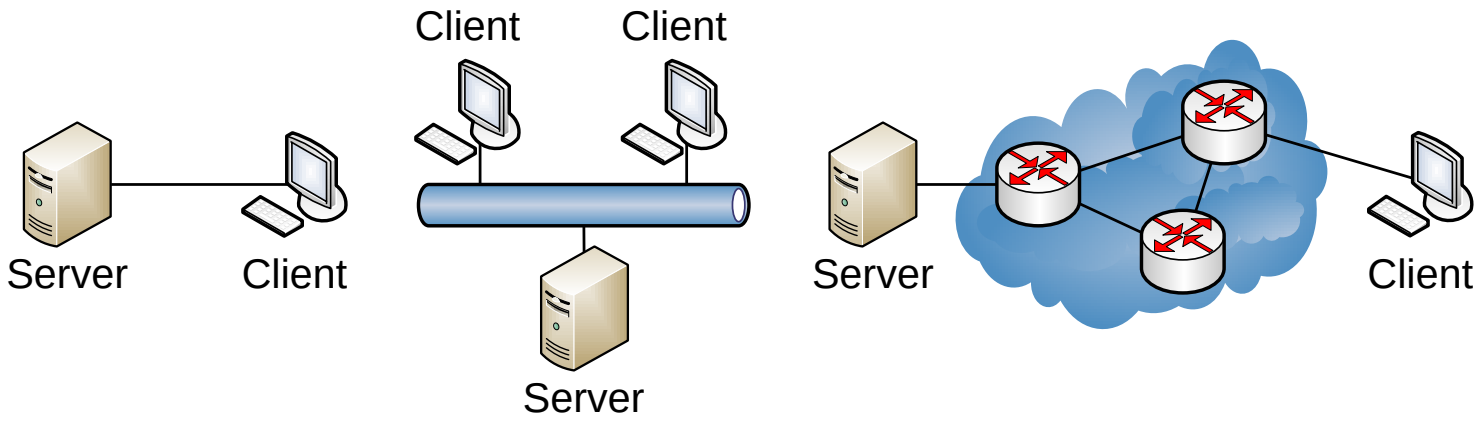
Outline

Introduction

- **Communication network** – a system of interconnected end systems, intermediate systems, and other equipment allowing information to be exchanged
  - A network can be of any size, from 2 to 1000's devices
- **Intermediate system** – a device that operates as a relay element between 2 or more end systems (networks)
  - E.g., a repeater, a hub, a bridge, a switch, a router
- **End system** – a device that uses or provides end-user applications or network services
  - E.g., a desktop PC, a Web server, a DNS server
  - They are labeled "end systems" because they sit at the edge of a network
  - End systems that are connected to the Internet are also referred to as "hosts"; this is because they host (run) Internet applications

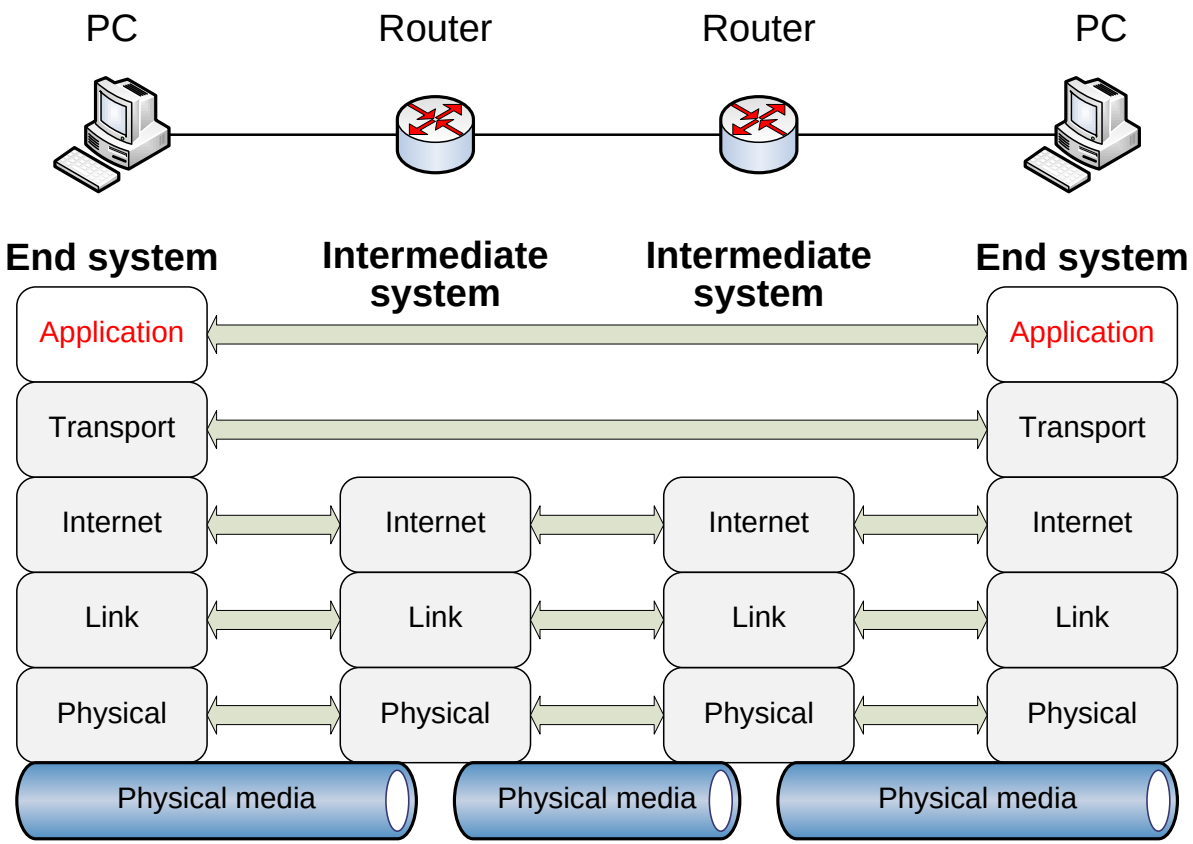
Introduction (cont'd)

- End systems can be positioned on a network in different ways relative to each other
  - I.e., they can be made to communicate and share resources according to different interaction models
- **2 fundamental interaction models**:
  - Client/server
  - Peer-to-peer (aka P2P)
- **These models are relevant to end systems only**, regardless of how the end systems are connected to each other



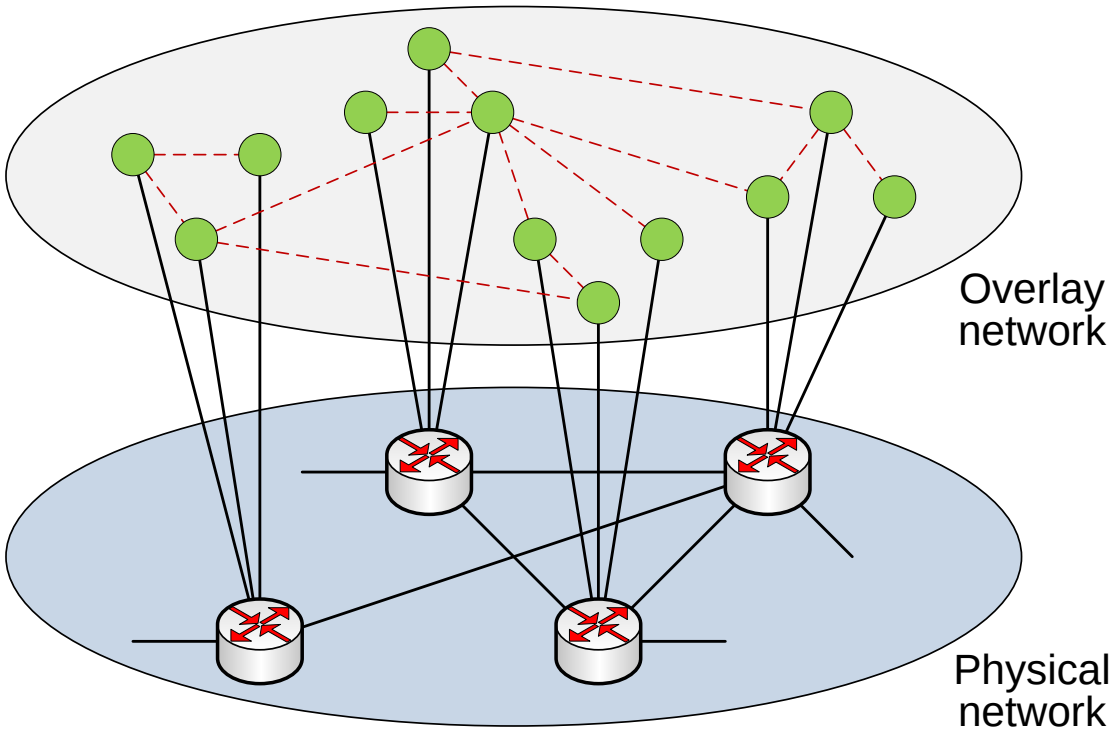
Introduction (cont'd)

- Client/server and P2P protocols operate at **the application layer** of the TCP/IP model
  - Application layer protocols are **end-to-end** protocols



Introduction (cont'd)

- Client/server and P2P systems are implemented as virtual networks of nodes and **logical** links built on top of an existing (aka **underlay**) network, typically the Internet
  - These virtual networks are called **overlay networks**
  - The overlay is a logical view that might not directly mirror the physical network topology



Outline

Client/Server Model

- In the client/server model, all end systems are divided into clients and servers each designed for specific purposes
- **Clients** have an **active** role and initiate a communication session by sending requests to servers
  - Clients must have knowledge of the available servers and the services they provide
  - Clients can communicate with servers only; they cannot see each other
- **Servers** have a **passive** role and respond to their clients by acting on each request and returning results
  - One server generally supports numerous clients

Client/Server Model (cont'd)

- **Hardware roles**:
  - The terms "client" and "server" usually refer to the primary roles played by networked hardware
  - A "client" is usually something like a PC used by an individual, and primarily initiates conversations by sending requests
  - A "server" is usually a powerful machine dedicated to responding to client requests, sitting in a server room somewhere that nobody but its administrator ever sees

Client/Server Model (cont'd)

- **Software roles**:
  - TCP/IP uses different pieces of software for many protocols to implement "client" and "server" roles
  - Client software is usually found on client hardware and server software on server hardware, but not always
  - Some devices may run both client and server software

- **Web clients**: Mozilla Firefox, Internet Explorer, Google Chrome, ...
  - See "Web Statistics" by W3Schools
    www.w3schools.com/browsers/browsers_stats.asp
- **Web servers**: Apache, Microsoft IIS, GWS, ...
  - See "Web Server Survey" by Netcraft Ltd.
    news.netcraft.com/archives/category/web-server-survey/

Client/Server Model (cont'd)

- **Transactional roles**:
  - In any exchange of information, the client is the entity that initiates communication or sends a query; the server responds, usually providing information
  - Again, usually client software on a client device initiates a transaction, but this is not always the case
  - E.g., when 2 SMTP servers communicate to exchange electronic mail, even though they are both server programs running on server hardware, during any transaction one device acts as client, while the other acts as server

Servers

- The purpose of servers is to provide some predefined services for clients
- **2 types of servers**:
  - Iterative
  - Concurrent
- **Iterative servers** iterate through the following steps:
  1. Wait for a client request to arrive
  2. Process the request and send the response back to the client
  3. Go back to Step 1
- Thus, iterative servers handle clients **sequentially**, finishing with one client before servicing the next

Servers (cont'd)

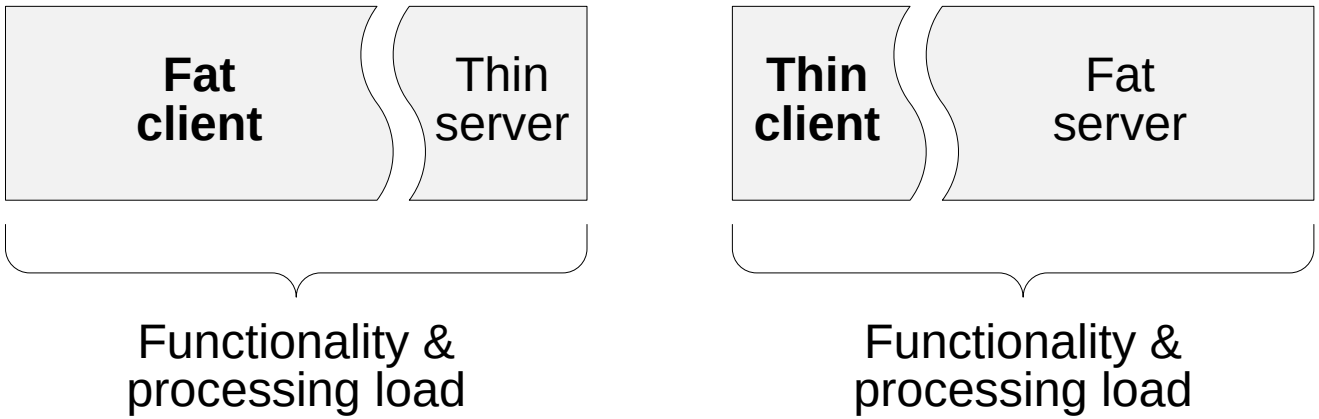
- **Concurrent servers** perform the following steps:
  1. Wait for a client request to arrive
  2. Use a new process/task/thread to handle the request
  3. Go back to Step 1
- Thus, concurrent servers handle client requests **in parallel**
- 2 approaches to implement concurrent servers:
  - **Thread-per-client** – a new thread is spawned to handle each request
  - **Thread pool** – a prespawned set of reusable threads which take turns (round-robin) handling incoming requests

Servers (cont'd)

- Iterative or concurrent?
- Iterative design is quite simple and is most suitable for short-duration services that exhibit relatively little variation in their execution time
  - I.e., if the time to handle a client can be long, the waiting time experienced by subsequent clients may be unacceptable
  - Internet services like **echo** (RFC 862) and **daytime** (RFC 867) are commonly implemented as iterative servers
- Concurrent design is more complex but yields better performance
  - It allows to improve responsiveness and reduce latency when the rate at which requests are processed is less than the rate at which requests arrive at the server
  - Internet services like **HTTP**, **telnet**, and **FTP** are commonly implemented as concurrent servers
- As a rule, TCP-based servers are concurrent and UDP-based servers are iterative

Clients

- Clients are devices/programs that request services from servers
- Clients (and, hence, servers) can be distinguished according to the **functionality** they provide and the amount of **processing load** they carry
- **2 types of clients**:
  - Fat (aka thick or full)
  - Thin (aka slim or lean)



Clients

- **Fat clients** are devices/programs that are powerful enough and operate with limited dependence on their server counterparts
- **Fat clients as devices** – a user workstation that is powerful and fully-featured in its own right
  - E.g., a desktop PC, a laptop, a netbook

- **Fat clients as programs** – a client carries a relatively large proportion of the processing load
  - E.g., the Lineage II gaming client (more than 2 GB in size)

Clients (cont'd)

- **Thin clients** are devices/programs that have very limited functionality and depend heavily on their server counterparts
- **Thin clients as devices** – a user workstation that contains a minimal operating system and little or no data storage
  - E.g., Sun Ray thin clients in Lintula, room TC215

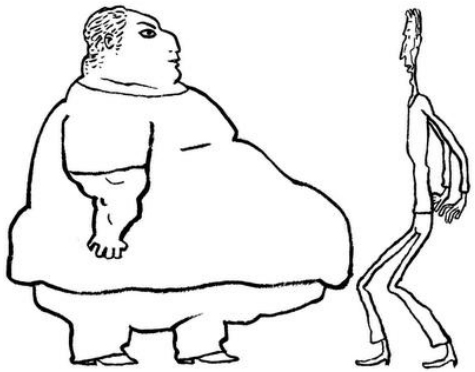
- **Thin clients as programs** – a client mainly provides a user interface, while the bulk of processing occurs in the server
  - E.g., the OnLive gaming client (about 10 MB in size)

Clients (cont'd)

- Fat or thin?
- Both technologies have their positive and negative aspects
- **Benefits** of thin-client systems:
  - No viruses, spyware, spam, thefts, etc.
  - Easy to keep the software properly configured and patched
  - Lower TCO (Total Cost of Ownership)
  - Fewer points of failure
- **Shortcomings** of thin-client systems:
  - Servers are the central point of failure
  - Systems tend to be proprietary
  - Multimedia-rich applications require a significant amount of bandwidth to function to their maximum potential (e.g., OnLive recommends a 5 Mbit/s connection speed or higher)

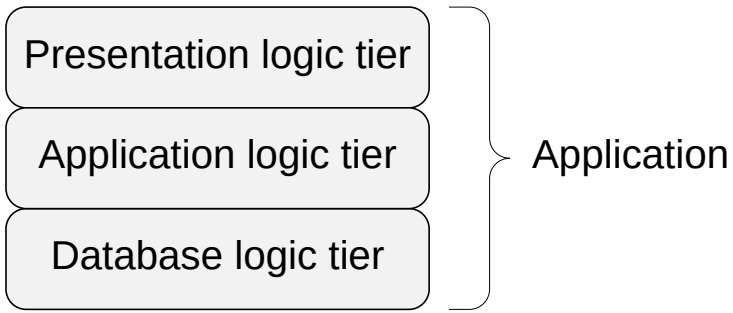
Clients (cont'd)

- Today, there are a lot of factors driving IT towards thin clients:
  - Desktop virtualization and increase in Web-based applications
  - High-bandwidth LAN and WAN connections
  - Data security concerns
  - Energy savings
  - Advances in low-cost computers (i.e., netbooks and, possibly, nettops)
- "`Thin clients can reduce TCO but the trick is using them for the right people and applications.`"
  - www.wgpeople.com/itn_thinisin.htm



Logical Tiers

- In general, application software can be divided into **3 logical tiers**:
  - Presentation logic
  - Application logic (aka business rules)
  - Database logic
- Each tier in the software is responsible for a specific task in the application
  - This layering is a reference model; in practice, the boundaries may be not so sharp
- The logical layering of application software does not need to be the same as its physical layering

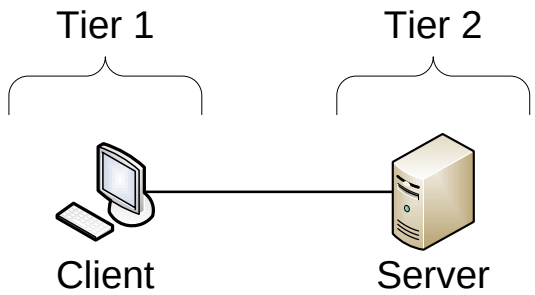


Logical Tiers (cont'd)

- **Presentation logic** is responsible for displaying the information and interacting with the user (i.e., user interface)
  - It must make the information available in a suitable form to the user and must respond appropriately to input from the user
- **Application logic** processes commands, makes logical decisions, performs calculations, and coordinates the application
  - It also moves and processes data between the presentation logic and database logic tiers
- **Database logic** refers to the management of underlying databases
  - It is responsible for storing and retrieving the data according to the requirements of the application logic tier
  - A static content is typically stored in a **file system**

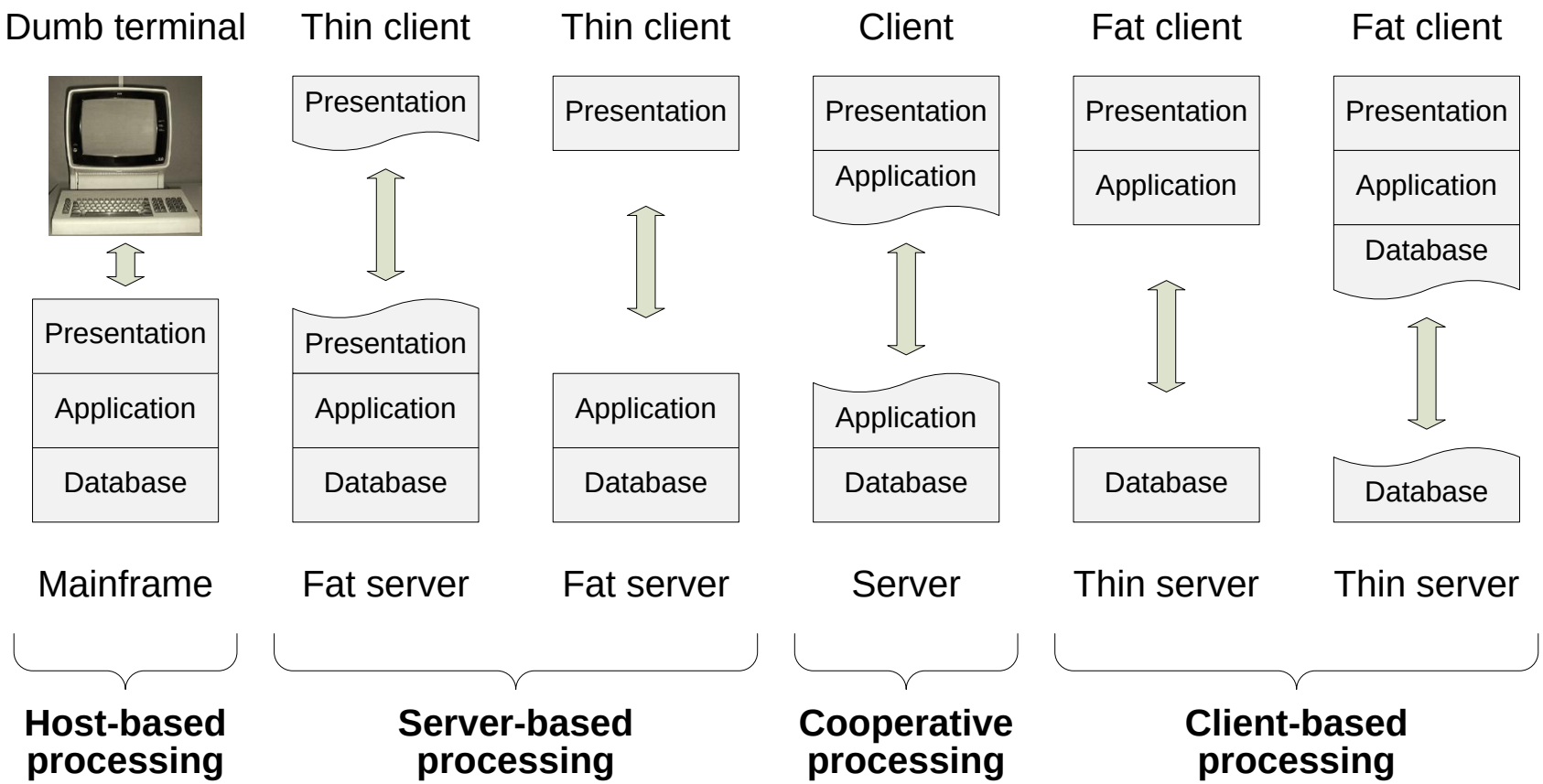
Physical Tiers

- **1-tier architecture** is used to describe systems in which all of the processing is done on a single host
  - Users can access such systems (aka **mainframes**) through display terminals (aka **dumb terminals**) but what is displayed and how it appears is controlled by the mainframe
- **2-tier architecture** (aka **flat**) is used to describe client/server systems, where clients request resources and servers respond directly to these requests, using there own resources



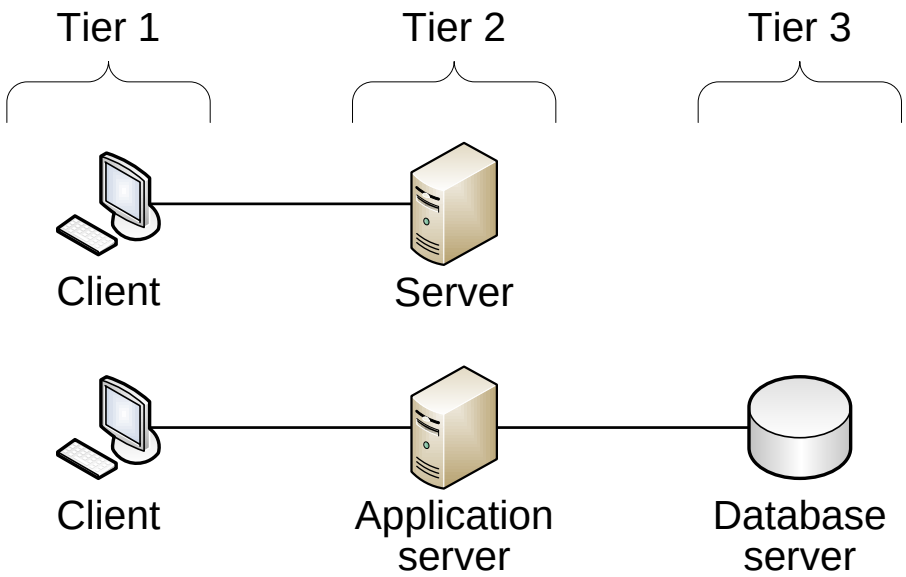
Physical Tiers (cont'd)

- Distributing logical tiers between 2 physical tiers



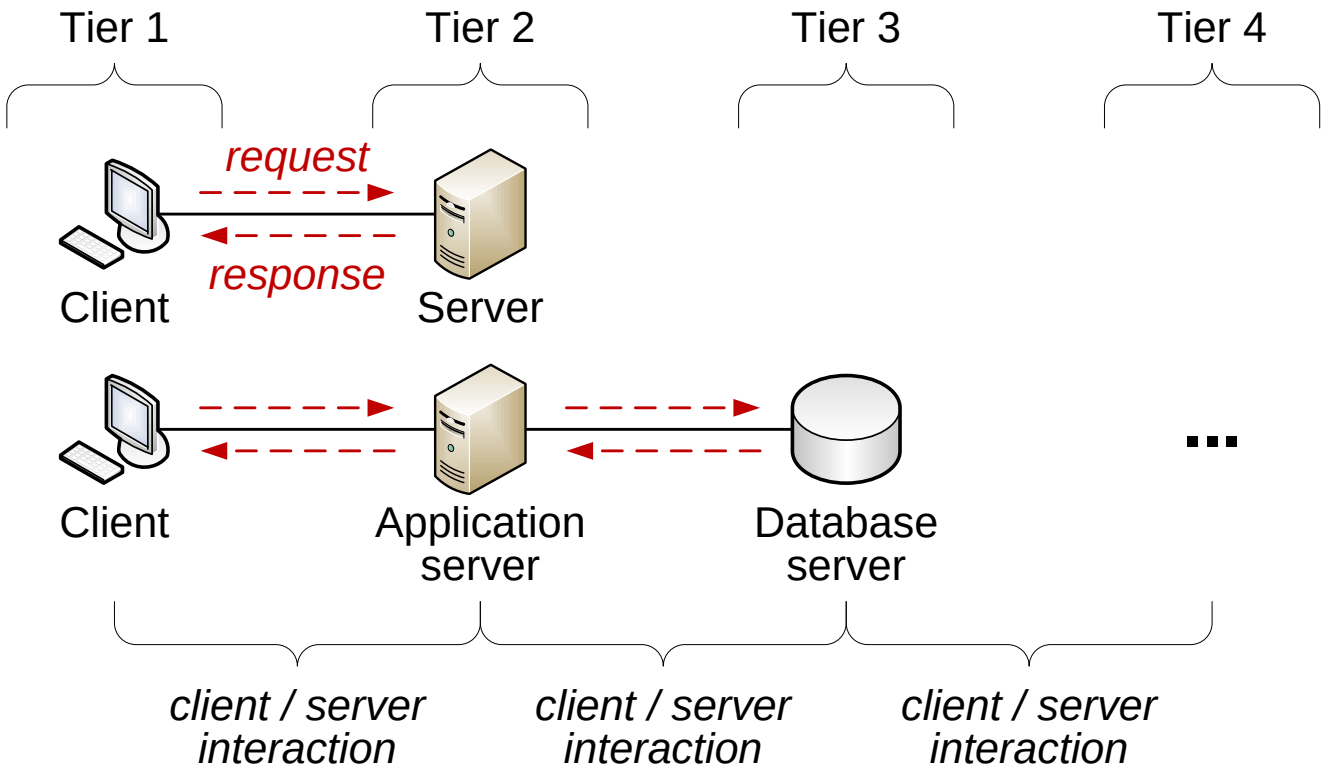
Physical Tiers (cont'd)

- **3-tier architecture** is used to describe client/server systems consisting of:
  - **Clients** which request services
  - **Application servers** whose task is to provide the requested resources, but by calling on database servers
  - **Database servers** which provide the application servers with the data they require



Physical Tiers (cont'd)

- **N-tier architecture** is used to describe client/server systems consisting of more than 3 tiers



Physical Tiers (cont'd)

- **Benefits** of 3- and N-tier (aka **multi-tier** or **hierarchical**) architectures:
  - Increased flexibility, as changes to the presentation/application/database logic are largely independent from one another
  - Increased security, as security can be defined for each service and at each tier
  - Increased performance, as tasks are shared between tiers
- **Shortcomings** of multi-tier architectures:
  - Increased complexity of development and testing
  - Increased need for load balancing and fault tolerance

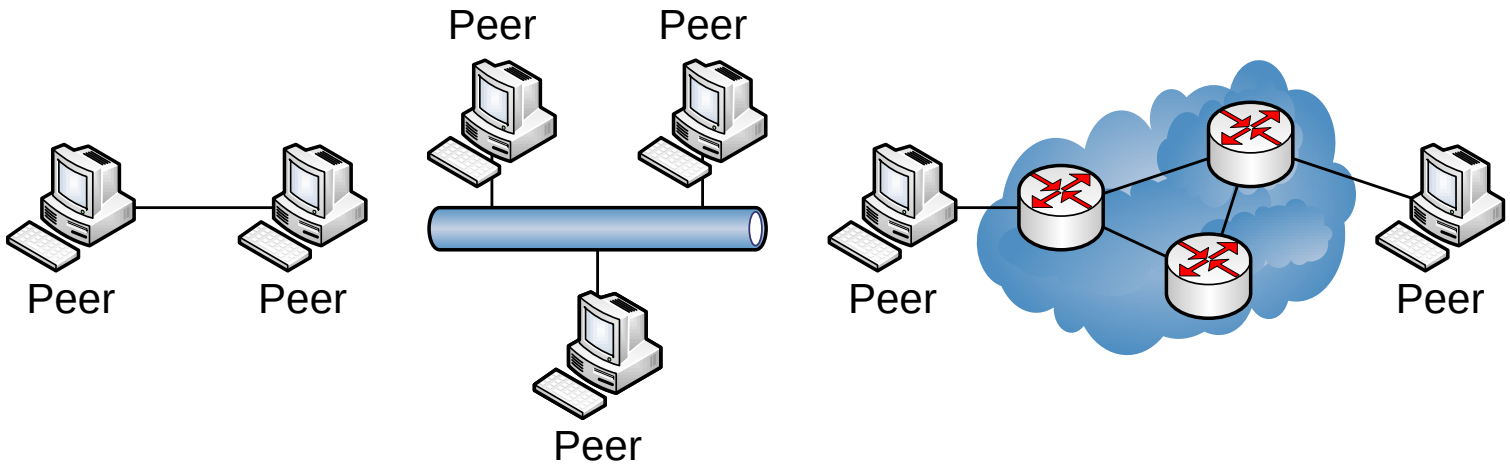
Outline

P2P Model

- In the P2P model, all end systems have equivalent capabilities and responsibilities and either party can initiate a communication session
- The participants share a part of their own hardware resources
  - E.g., storage capacity, link capacity, CPU power
  - These shared resources are necessary to provide the service or content offered by the P2P network
- Thus, the participants are **both resource providers and resource requestors** and use similar networking programs to connect with each other
- In P2P networks, downlink and uplink data flows tend to be (but not necessarily) **symmetric**
  - This is because each connected host simultaneously operates as both client and server, thus receiving and transmitting on average the same amount of data

P2P Model (cont'd)

- The P2P model does not have the notion of clients or servers but only equal **peers** (aka **servents**, servent = SERVer + cliENT) that simultaneously function as both clients and servers
  - But for any communication session we can still distinguish between requesting peers as "clients" and responding peers as "servers"
- Again, **this model is relevant to end systems only**, regardless of how the end systems are connected to each other



P2P Model (cont'd)

- **Benefits** of P2P:
  - No need for dedicated application and database servers
  - Improved scalability and reliability (no single point of failure)
- **Shortcomings** of P2P:
  - Poor security
  - Lack of centralized control
  - Computers with shared resources may suffer from sluggish performance
- P2P networking allows easily to share and download copyrighted files
  - Is it a benefit or a shortcoming? :-)

Pure P2P

- In P2P systems, cooperative peers self-organize themselves into overlay networks and store or relay data for each other
- **2 types of P2P systems**:
  - Pure
  - Hybrid
- **Pure P2P system** – a P2P system that has **no central service** of any kind
  - I.e., the entire communication occurs among connected peers without any assistance from any server
- **Pure P2P systems represent the reference type of P2P design**
- Examples of pure P2P systems:
  - Workgroups in Microsoft Windows Network
  - Gnutella v0.4
  - Freenet

Hybrid P2P

- The major challenge of P2P systems is how to achieve efficient resource search
  - Pure P2P systems work well only in a small-scale environment
- **Hybrid P2P system** – a P2P system which depends partially on central servers or allocates selected functions to a subset of dedicated peers
  - **Central servers** act as central directories where either connected users or indexed content can be mapped to the current location
  - **Dedicated peers** direct control information among other peers
- Thus, **client/server and P2P systems are not mutually exclusive**
- Examples of hybrid P2P systems:
  - Usenet
  - Napster
  - Gnutella v0.6
  - eDonkey2000
  - BitTorrent

Outline

Summary

- Client/server vs. pure P2P

	Client/server	**Pure P2P**
Participants	Clients and servers	Equal peers
Networking software	Different for clients and servers	Similar for all
Active role (requester)	Client	Any participant
Passive role (provider)	Server	Any participant
Interaction	Clients with servers	Arbitrary
Service/content/resource provider	Servers	Active participants
Data flows (**in theory**)	Asymmetric	Symmetric

Summary (cont'd)

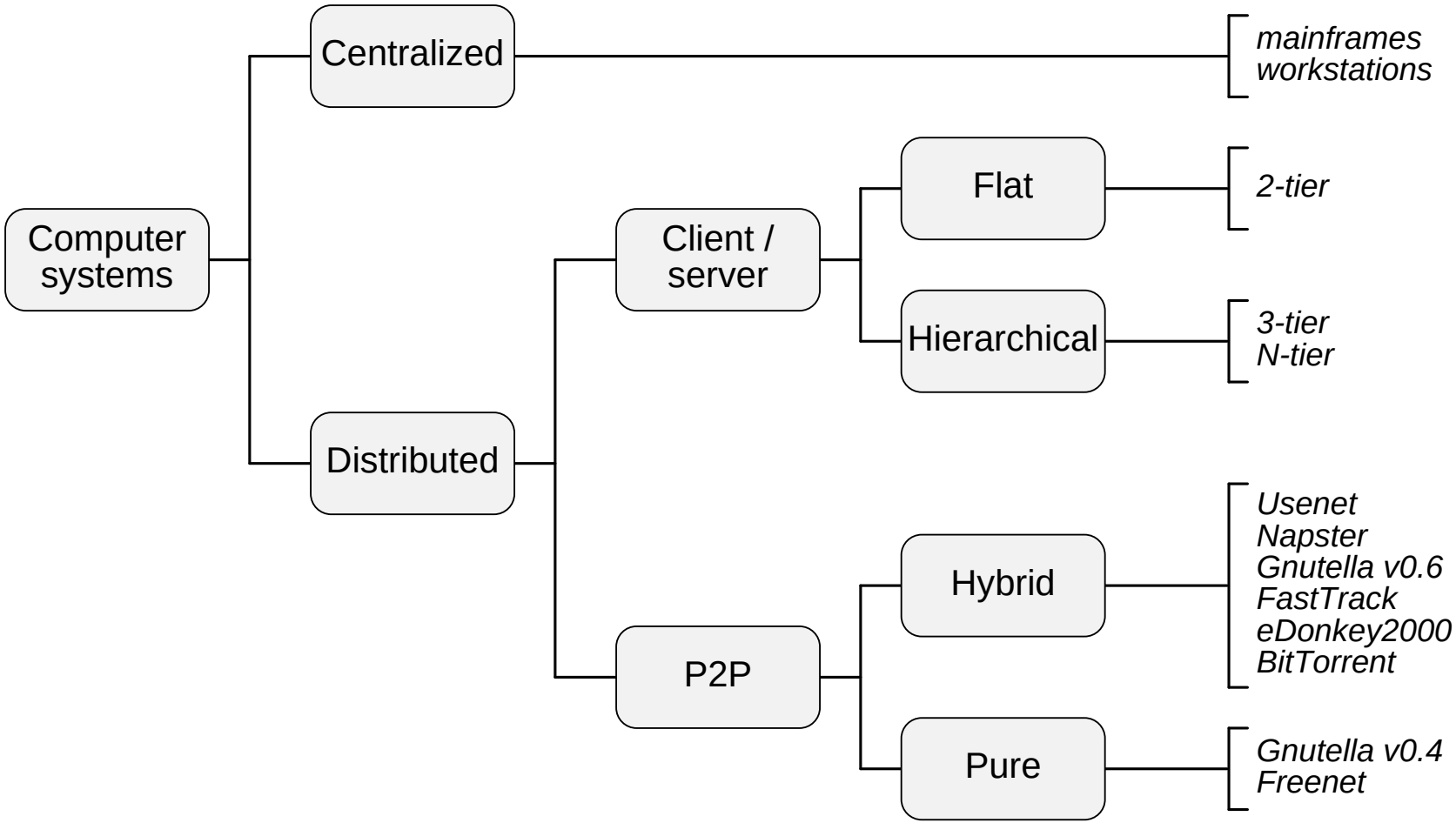
- Client/server vs. P2P as lecture-based vs. project-based learning

Summary (cont'd)

- Client/server vs. P2P as eating at restaurant vs. eating at home

Summary (cont'd)

- Taxonomy of computer systems architectures



Summary (cont'd)

- **Centralized systems** represent single-unit solutions, including single- and multi-processor machines, as well as high-end machines, such as supercomputers and mainframes
- **Distributed systems** are those in which components located at networked computers communicate and coordinate their actions by passing messages
- **Flat client/server systems** – all clients only communicate with a single server (possibly replicated for improved reliability)
- **Hierarchical client/server systems** – servers of one level are acting as clients to higher-level servers
- **Pure P2P systems** – there are no central servers
- **Hybrid P2P systems** – a server is approached first, then the P2P communication is performed

Outline

Learning Outcomes

- Things to know:
  - Fundamental models (client/server, P2P)
  - Types of servers (iterative, concurrent)
  - Types of clients (fat, thin)
  - Logical tiers (presentation, application, database)
  - Physical tiers (1-, 2-, 3-, N-tier architectures)
  - Types P2P systems (pure, hybrid)
- Be ready to explain/compare/give examples