# CIS 630
# Distributed Systems

Lecture 2

# Distributed System Models

- Architectural models
  - Concerned with placement of parts and their relationships
  - Defines how these parts map down onto the network and computers.
- Fundamental models formalize properties of the systems (e.g.: correctness, reliability, etc…)
- Distributed system characteristics addressed by:
  - Interaction model
  - Failure model
  - Security model

# Difficulties For / Threats To Dist. Sys.

- Widely varying models of use
  - Workload has wide variation
  - Poor connectivity of some parts of system
  - Applications have different requirements
    - Bandwidth and latency
- Wide range of system environments
  - Heterogeneous hardware, OS, networks
  - Varying network performance
  - Widely differing system scales
- Internal problems – clocks, data, component failure
- External problems – attacks, data integrity, secrecy

# Lamport's Definition of a DS

- Lamport once defined a distributed system as:
  - "One on which I cannot get any work done because some system I never heard of has crashed."

- Applications need to adapt gracefully in the face of partial failure.
- An example of a distributed system technology that will lead to Lamport's issue is NFS. How many of us have ever seen a set of workstations freeze because the NFS server failed?
  - Distributed file systems are hard, especially with respect to adaptation to failure.

# Architectural Models

- Ensure that the **structure** meets requirements.
- Simplify and abstract functions of individual components of a distributed system. Then consider:
  - How these are placed amongst a set of networked computers. We seek to define useful patterns to drive data distribution, workload distribution.
  - Inter-relationships between components, their functional roles and communication patterns.
- Classification aids in simplification.
  - Servers, clients, peers.
  - Classification identifies responsibilities, behavior, workload and failure properties.
  - Analysis is used to specify placement based to meet objectives.

# System Architectures

- This is concerned with the division of responsibilities.
  - Between system components (apps, servers, processes)
  - Placement on computers in the network
- Implications for performance, reliability, and security.
- Types
  - Client-server model
  - Services provided by multiple servers
  - Proxy servers and caches
  - Peer processes
  - Mobile code / agents / spontaneous networking
  - Networked computers / thin clients

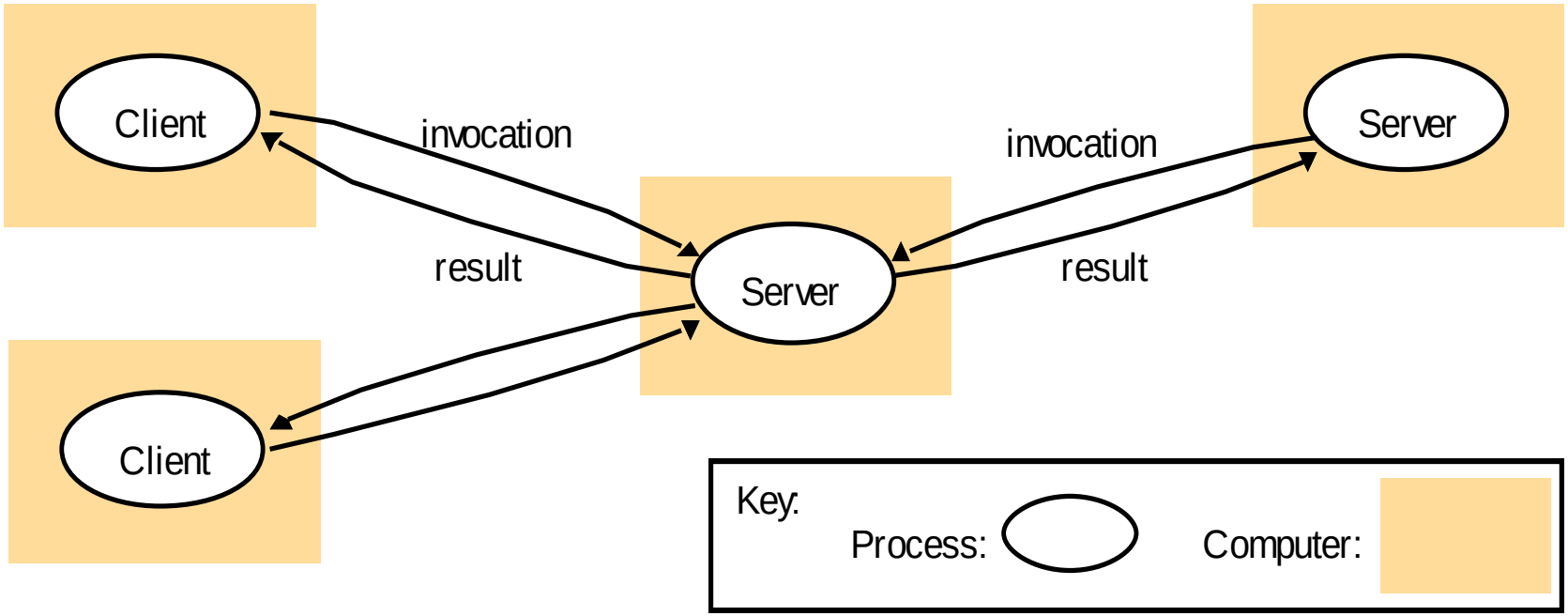
# Client/Server Model, Multiple Servers

- We're all familiar with this one. The web is the most widespread with browsers (clients) and web servers (servers).
- The model defines the interaction relationship.
  - Service: A task a machine can perform
  - Server: A machine that performs that task when requested
  - Client: A machine that requests the service
- The model allows chaining and hierarchy
  - Servers may be clients of other servers.
    - Example: WWW server using files provided by a file server.
- Service types
  - Directory service, print service, file service, …

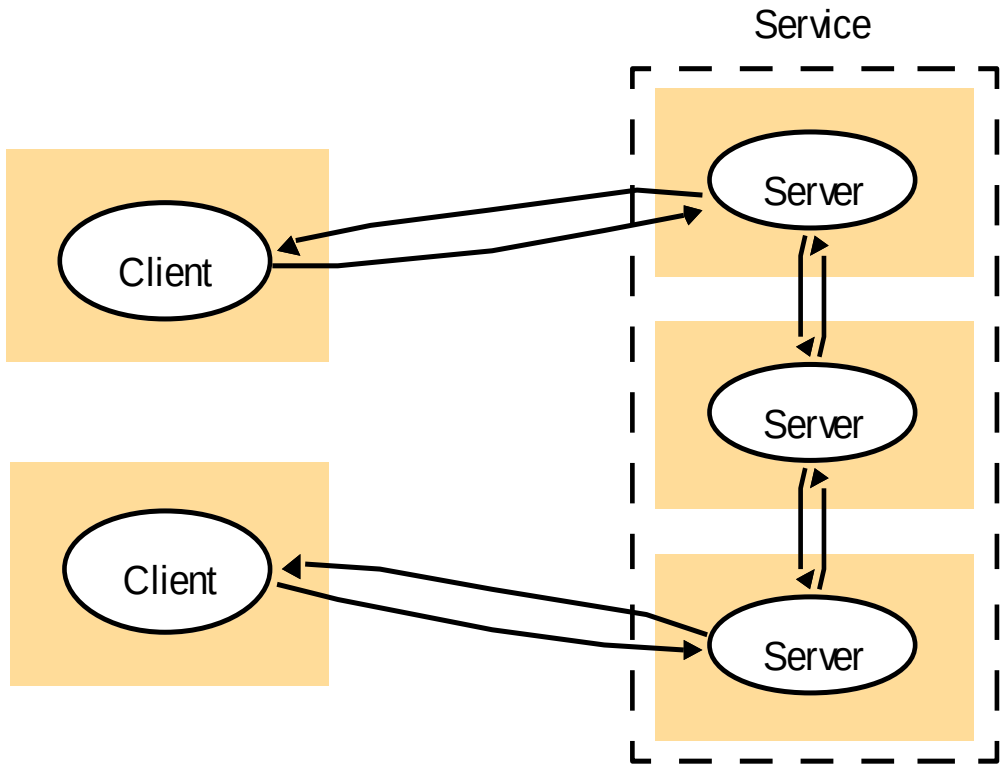
# Client/Server Model, Multiple Servers

- Services may be implemented by distributed processes.
  - May require distributed resources (such as the WWW)
  - May choose to partition and distribute for reliability
  - Replication can be used to:
    - Increase performance
    - Increase availability
    - Improve fault tolerance

# Clients Invoke Individual Servers



# A Service Provided by Multiple Servers




- *Example*: load balancing very heavily used web servers by delegating clients to different servers based on individual server load or client proximity.

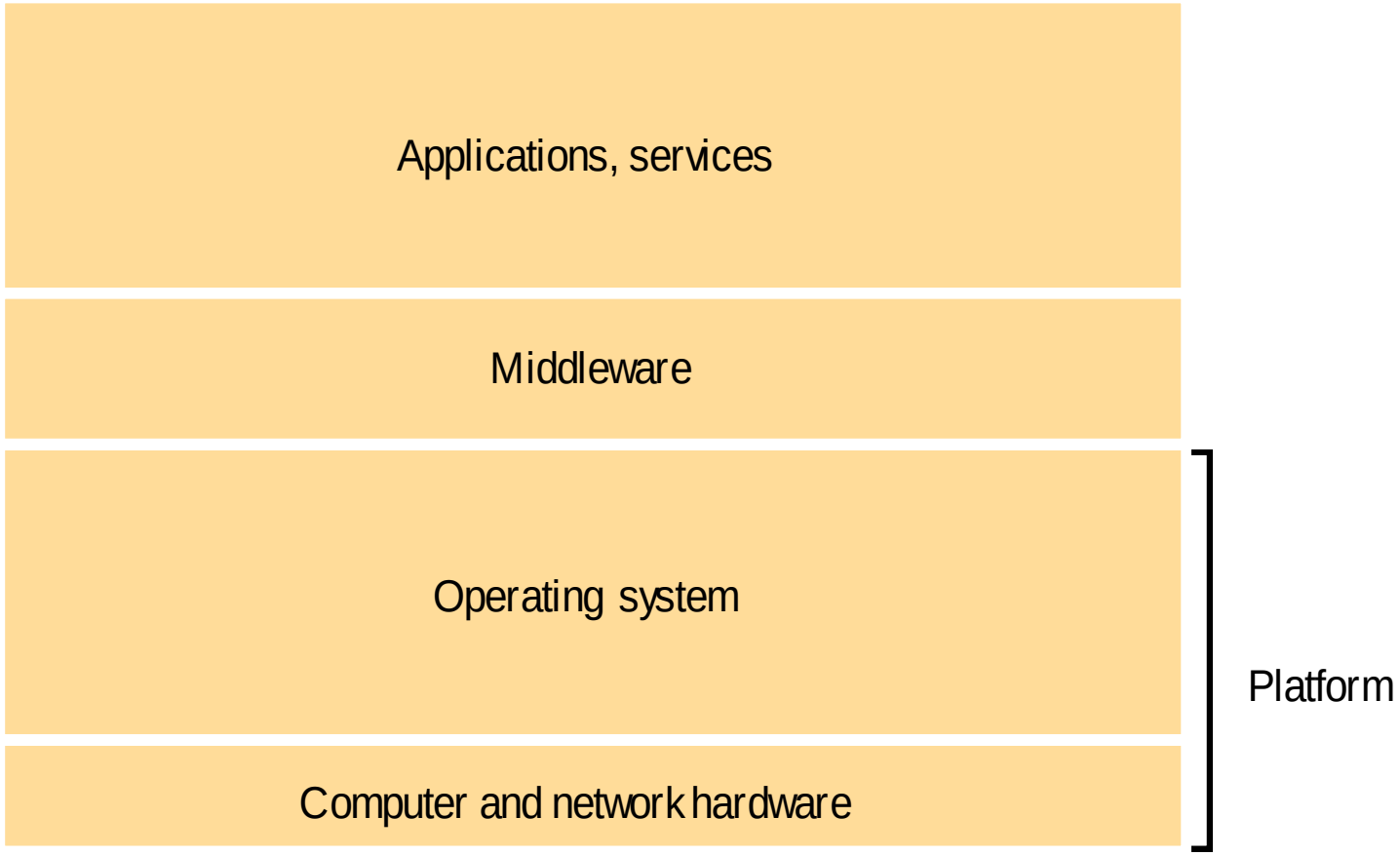
# More on Client/Server Model

- Clients
  - Generally block until server responds or a timeout occurs.
  - Typically invoked by end users when they require service.
  - Interacts with users through a user interface.
  - Interacts with client middleware through middleware API to abstract above underlying network connectivity to server.
- Server
  - Implements services.
  - Usually waits for incoming requests.
  - Usually a program with special privileges.
  - Invoked by server middleware.
  - Provides error recovery and failure handling services.

# Software Layers

- Software architectures refers to the structuring of software
  - Layers and services (“service layers”).
  - We will see an instance of this soon with the networking middleware.
- Platform
  - Lowest-level hardware and software layers (e.g.: OS).
- Middleware
  - Layer of software that provides abstraction above potential heterogeneity via a convenient programming model.
  - Building blocks for building software.
  - Raises the level of communication activities through communication abstractions and mechanisms.
  - Makes distributed nature of system transparent.

# Software and Hardware Layers



# Common middleware packages

- Remote procedure call (RPC)
- Group communication (Isis)
- Object-oriented
  - CORBA: Common Object Request Broker Architecture
  - Java RMI: Remote Method Invocation
  - Microsoft DCOM: Distributed Common Object Model
- Packages provide higher-level application services
  - Naming, security, transactions
  - Persistent storage, event notification

# Middleware limitations

- End-to-end argument (Saltzer, Reed, Clarke, 1984)
  - Some communications-related functions can be completely and reliably implemented only with the knowledge and participation of the application standing at the endpoints of the communication system. Therefore, providing that function as a feature of the communication system itself is not always sensible.
- This runs counter to the view that all communication activities can be abstracted away by middleware layers.
- Correct behavior in distributed programs depends upon error measures and security at all levels.
  - Example: fault tolerant, reliable, end-to-end transfer

# Functional View of Middleware

- Information exchange services
  - Message passing
- Application-specific services
  - Specialized services
    - Example: Transaction, replication services for distributed DB.
    - Example: Groupware services for collaborative applications.
- Management and support services
  - Name services and registries for locating distributed resources dynamically.
  - Administration of resources distributed over a network.
  - Monitoring performance and behavior of distributed set of resources.

# Production Middleware

- Single-service components
  - HTTP for retrieving documents remotely
  - Sun RPC for remote procedure call
  - SSL for secure socket layer
- Integrated middleware environments
  - Integrates multiple components into a single coherent package.
  - Examples: CORBA, DCOM, .NET, Java