System Models

Prof. David Bakken

Cpt. S 464/564 Lecture

Textbook, Chapter 2

Sept 11+13, 2000

Administrative Items

- Handouts today
  - Paper “End-to-End Arguments in System Design” (not testable for 464, but please keep a copy for future reference)
  - Lecture slides for today and Wednesday
  - Updated Syllabus
  - Homework #1 (due in one week)
- CORBA-I lecture and Project #1 delayed a week due to lab problems

Outline of Topics

1. **Overview of System Models**
2. Architectural Models
3. Fundamental Models

Difficulties in Distributed Systems

- Widely varying modes of use (usage patterns)
  - Rate of access
  - Connectiveness of a mobile host
  -
  -
- Wide range of system environments
  - Heterogeneity in multiple dimensions
  - Performance of components can vary widely
  - Load on components can vary widely
- Internal problems (inherent to a DS)
  - Lack of global clock
  - Lots of ways different components can fail
- External threats
  - Security …

Dealing with the Difficulties

- Very hard to deal with all these issues!
- Can't do so in large detail or all possible low-level interactions
- But some common properties and design issues can be abstracted: descriptive model
- A model provides "an abstract, simplified but consistent description of a relevant aspect of distributed system design"
- Idea: focus at the problem components or interactions of interest, abstract away the rest of the details that do not matter for this

Outline of Topics

1. Overview of System Models
2. **Architectural Models**
    1. **Software Layers**
    2. **System Architectures**
    3. **Variations on the Client-Server Model**
    4. **Interfaces and Objects (skip in lecture)**
    5. **Design Requirements for Distributed Architectures**
3. Fundamental Models

Architectural Models

- An architecture specifies
  - Major components of interest
  - Interactions of these components
- Example: building architecture gothic
- Architectural model of a distributed system
  - Defines what the major components should be
  - Defines where to place them across a network
  - Defines how the components should interact
- One way: classify processes
  - Client
  - Server
  - Peer
- Variations on client-server
  - Mobile code
  - Disconnecting and reconnecting clients
- Note: an architecture is not a design!

Software Layers

- Original definition: layers or modules in a single computer
- More recent definition: services offered and requested between processes on same or different computers
- Key idea: service layers

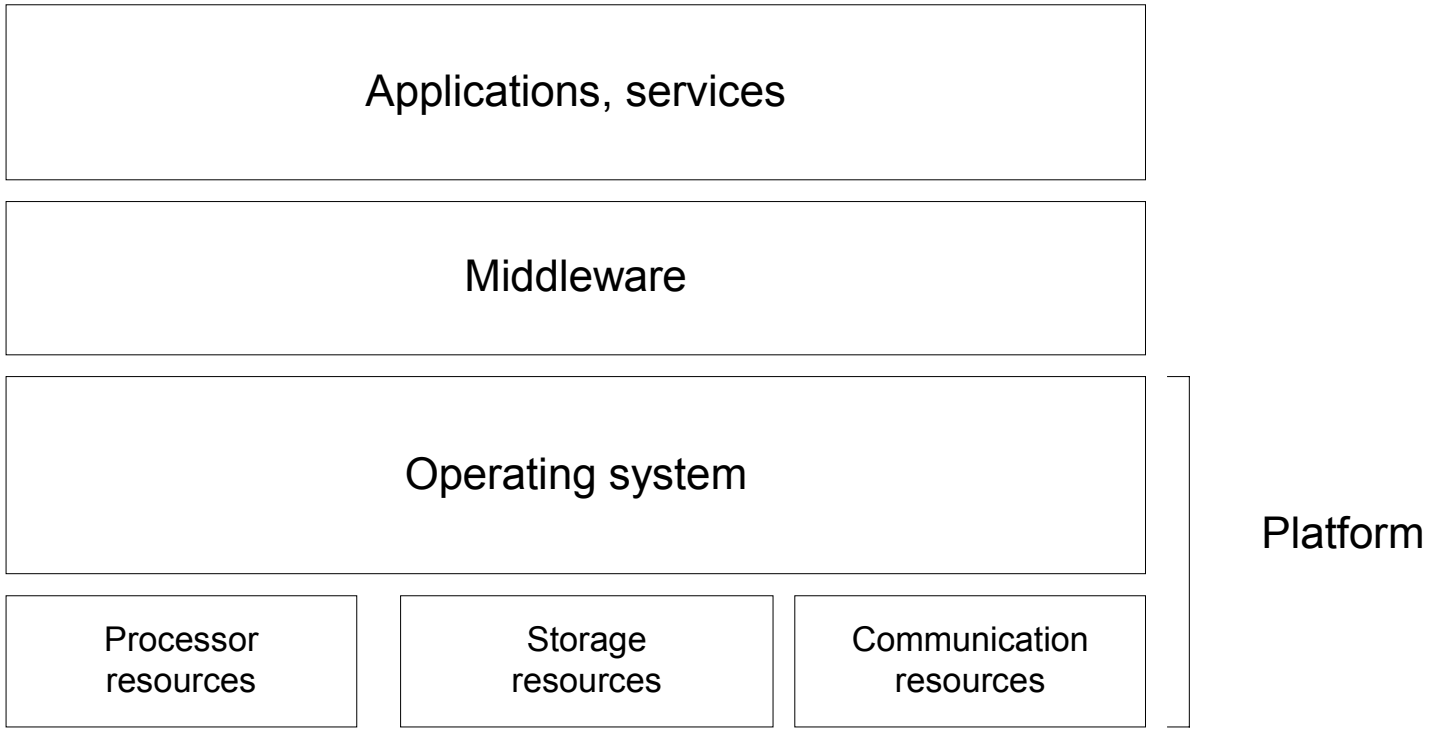
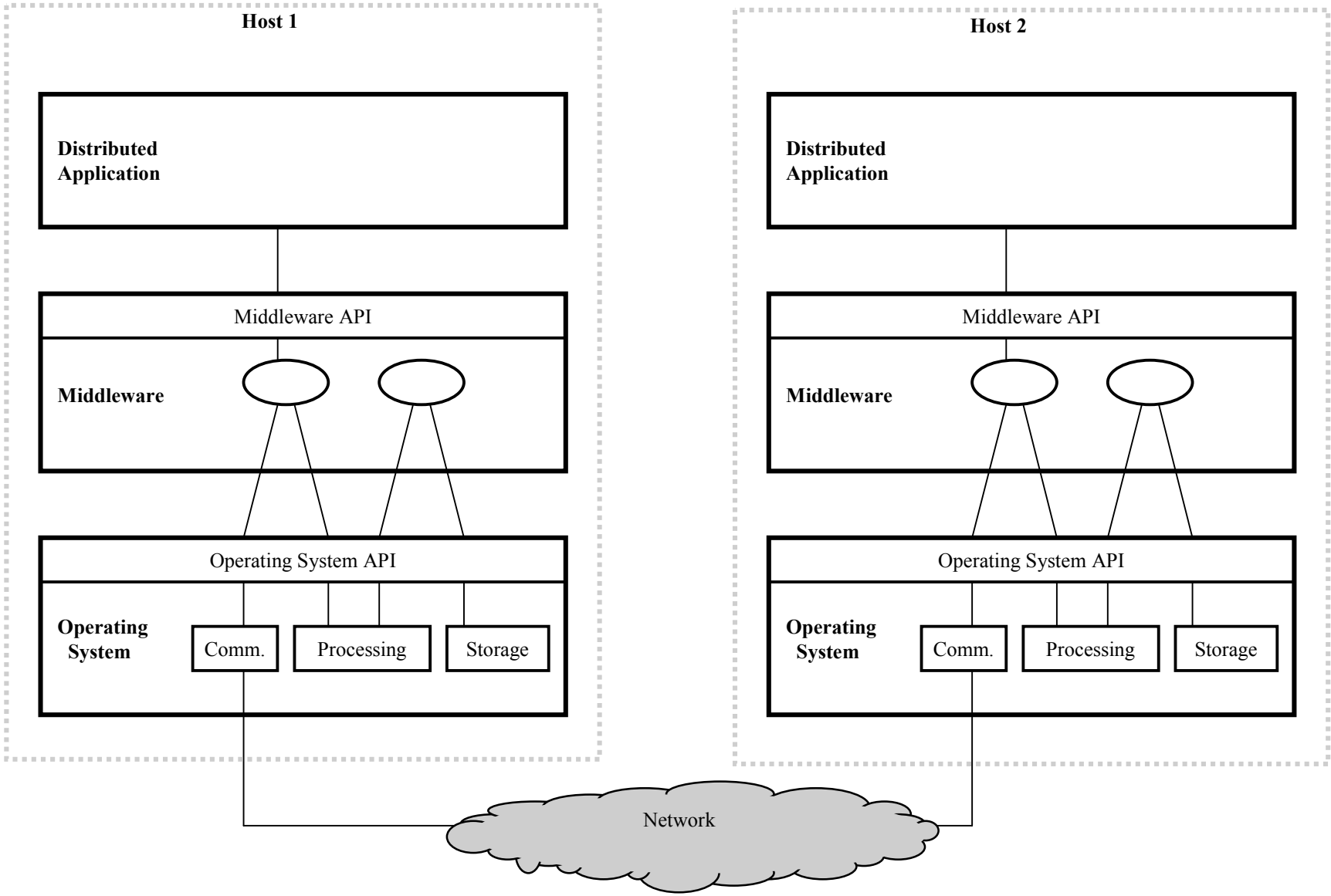


Figure 2.1 Software and hardware service layers in distributed systems

Middleware


Host 1
Distributed Application
Middleware API
Middleware
Operating System API
Operating System
Comm.
Processing
Storage
Host 2
Distributed Application
Middleware API
Middleware
Operating System API
Operating System
Comm.
Processing
Storage
Network

Limitations of Middleware

- Some aspects of dependability (and other issues) require application-level support
  - get_state() and set_state()
  - Merging partitioned replicas
  - Text example: TCP error correction versus big email message and SMTP
- "End-to-End" paper (a classic):
  - Some communication-related functions can be completely and reliably implemented only with the knowledge and help of the application standing at the end points of the communication system. Therefore, providing the function as a feature of the communication system itself is not always sensible.
- Checks, correction, etc. must be done at many levels
  - Which layer to put a given one at is an art, not a science!
  - Some must be done at the application layer
- Cannot abstract away everything in middleware! (But some try.)
- Einstein: "Make it as simple as possible, but no simpler."

System Architectures

- Issues:
  - How to map division of responsibilities/functionalities into components
  - How to place components in a system
  - Note: decisions here have **HUGE** impact on system
    - Performance
    - Security
    - Reliability
    - Price
    - Extensibility
- Architectures
  - Client-server
  - Replicated Servers
  - Proxy servers and caches
  - Peer processes

Client-Server Model

- Most widely used and cited
- Q: How to place the processes onto hosts?
- Q: What do you base this placement decision on?

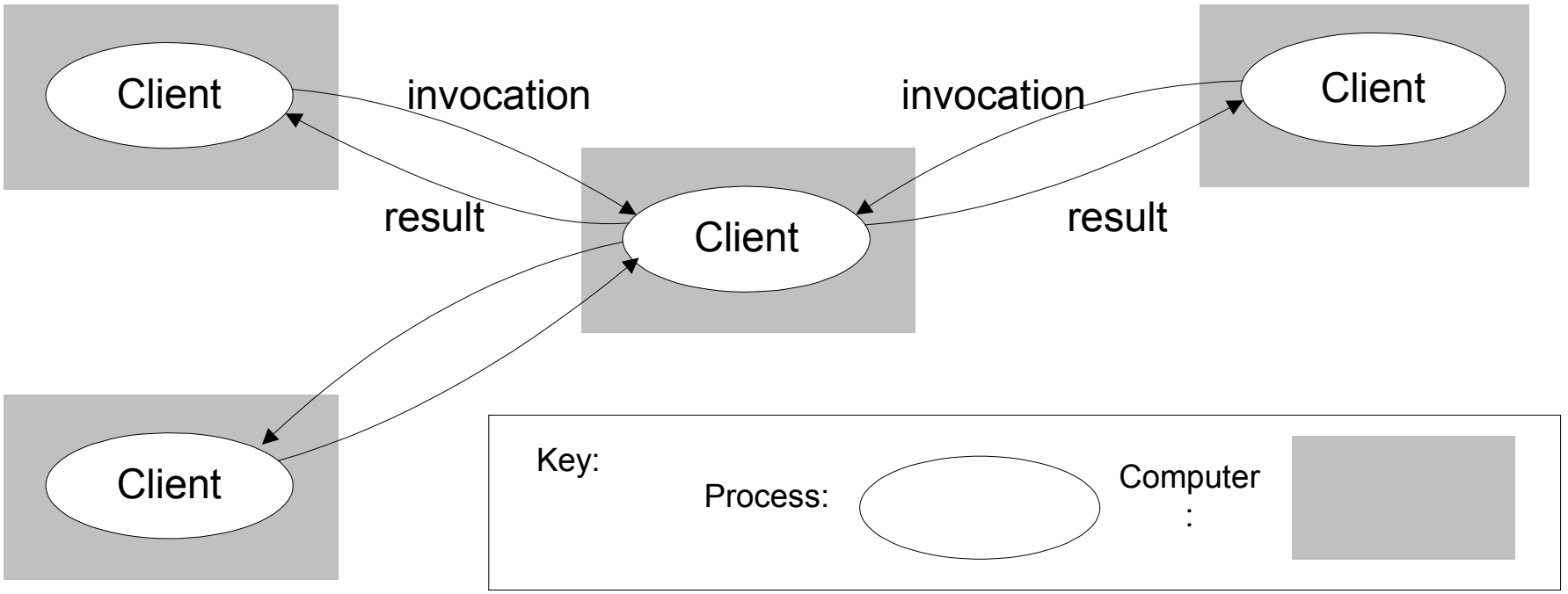


Figure 2.2 Clients invoke individual servers

Replicated Servers

- Choices:
  - All data on each server (completely replicated) for fault tolerance or performance of data frequently read
  - Data split between computers (1/3 on host1 of 3, …) for load balancing
  - Something inbetween (tradeoff between performance and load balancing)

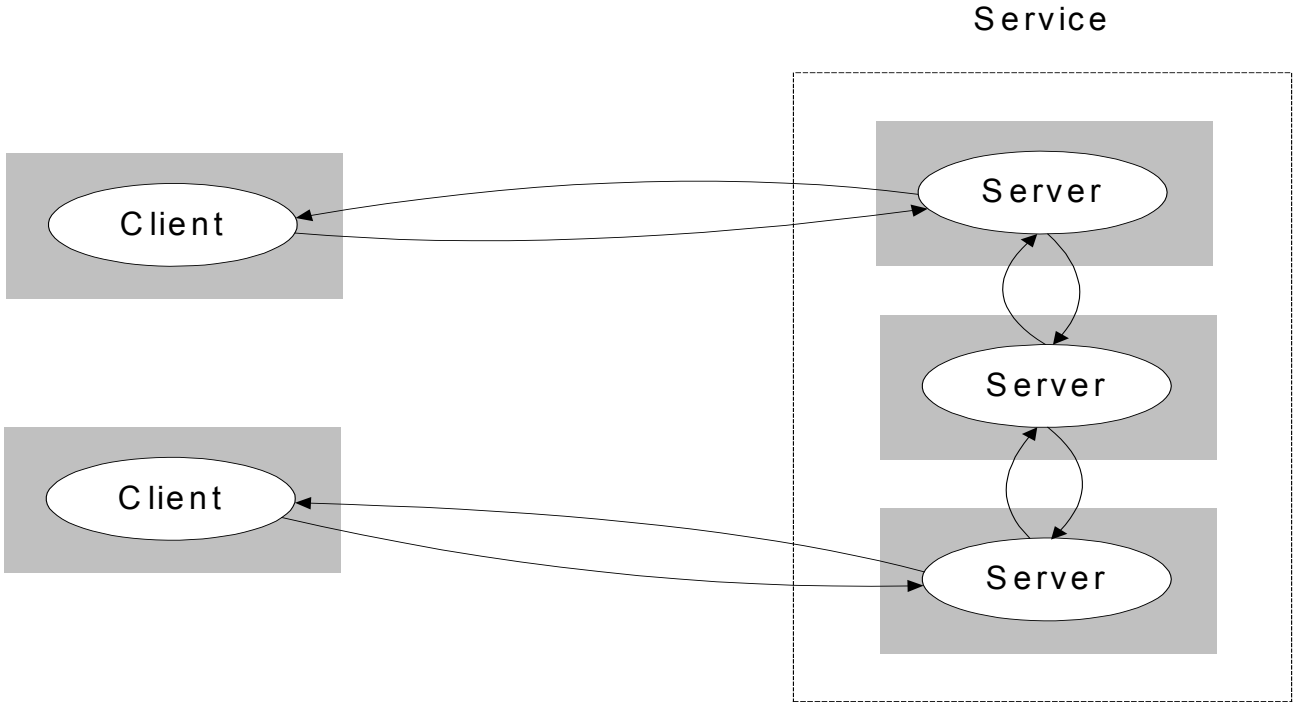


Figure 2.3 A service provided by multiple servers

Proxy servers and caches

- Cache: store of most recently used data objects closer to client than the objects themselves
- Caches are widely used (examples???)

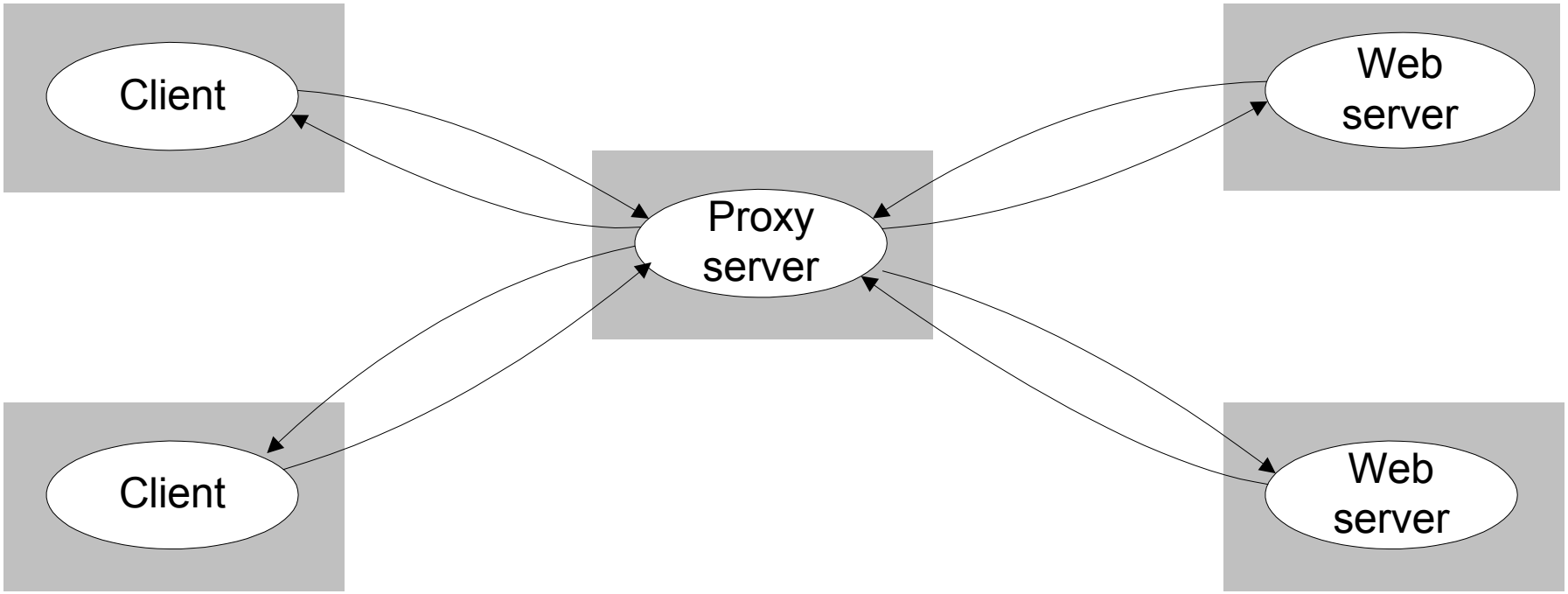


Figure 2.4 Web proxy server

Peer Processes

- All processes play similar or identical role
- Examples
  - Multi-user editor
  - Multi-user game
  - Shared whiteboard

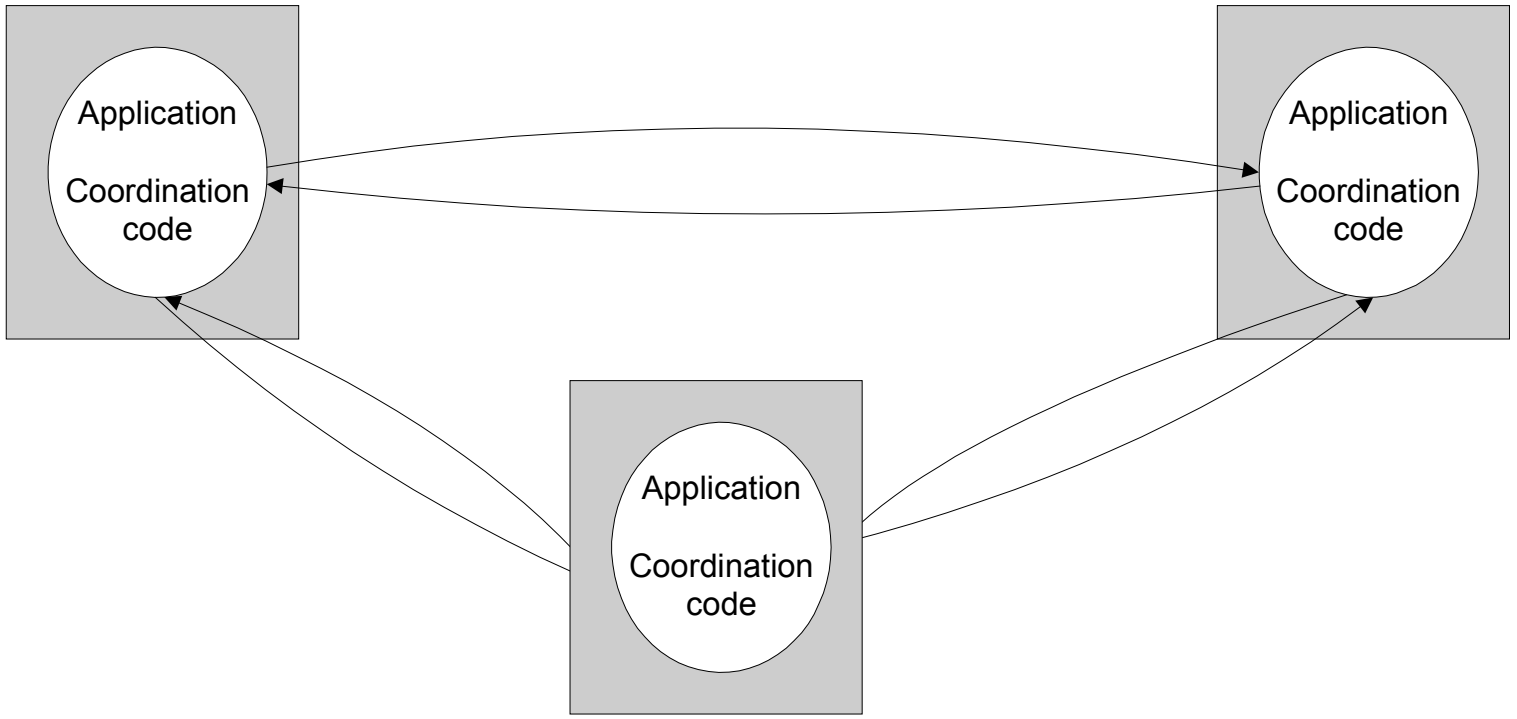
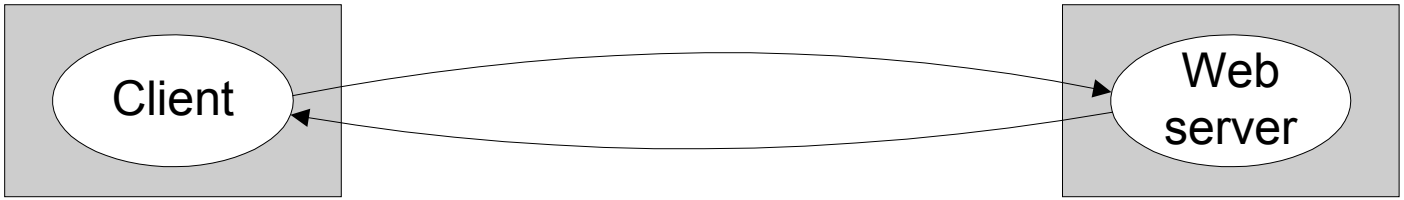


Figure 2.5 A distributed application based on peer processes

Mobile Code

- Variation on client-server model
- Code is downloaded and executed on client machine
- Advantage: quick interactivity and good error checking
- Disadvantages: download time and security risks

a) client request results in the downloading of applet code



b) client interacts with the applet

Figure 2.6 Web applets

Mobile Agents

- Mobile agent is a program that roams around a network doing work for someone
  - Collect information
  - Make decisions (purchases)
- Can have much greater autonomy than client-server applications
- Problems
  - Agents are a security threat to the hosts they run on
  - The hosts are a security threat to the agents

Network Computers and Thin Clients

- Network computers
  - Downloads the OS (!!) and application from remote file server
  - Apps run locally
  - Files managed remotely
  - Disks on computer hold very little software, mainly a cache
- Thin clients
  - Computer with a layer with a GUI for a remote application
  - Apps don't run locally, but on a powerful compute server
  - E.g., X-Windows system

Mobile Devices and Spontaneous Networking

- Lots of small and tiny portable devices being built and sold, networkable
- Spontaneous networking: connect to available devices nearby
- Key features
  - Easy connection to a local network (almost always wireless)
  - Easy integration with local services
- Problems
  - Limited connectivity while traveling
  - Security
- Discovery services: services to let clients discover what services are available and what their properties are
  - Registration service
  - Lookup Service
- Commercial examples today
  - Jini
  - Microsoft Unversal Plug and Play
  - Microsoft Research Aladdin (April 28 talk at WSU)

Spontaneous Networking Example

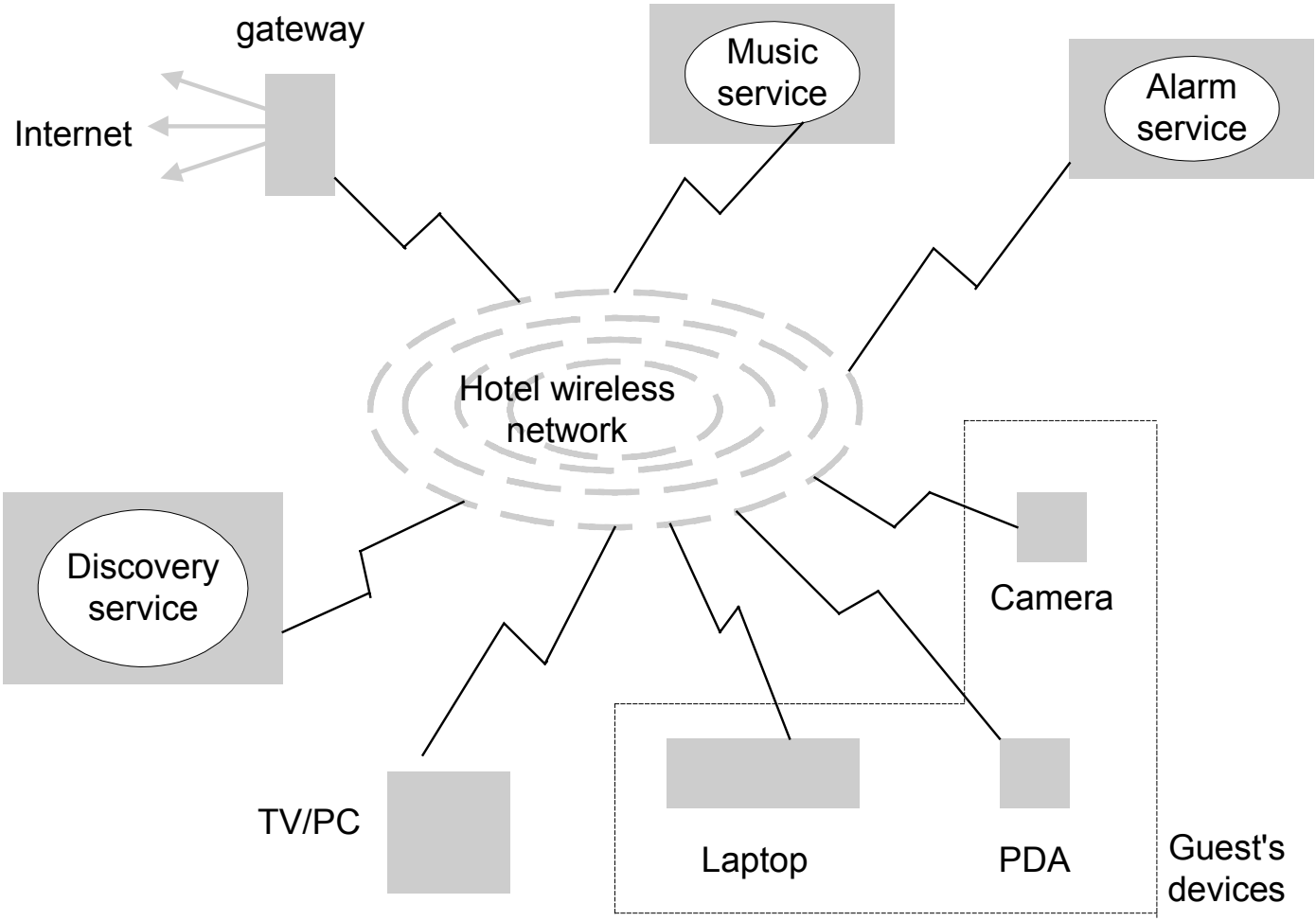


Figure 2.8 Spontaneous networking in a hotel

Design Requirements for Distributed Arch's

- Performance issues
  - Responsiveness
  - Throughput
  - Load balancing: spreading the load
- Quality of Service (QoS)
  - Reliability
  - Security
  - Performance
  - Adaptability
- Caching and replication: becoming very common
- Dependability issues
  - Correctness (certifiably)
  - Fault tolerance: requires redundancy (dispersion of data)
  - Security: keep sensitive data nearby

Outline of Topics

1. Overview of System Models
2. Architectural Models
3. **Fundamental Models**

Fundamental Models

- Very different architectural models shown (client/server, etc)
- But they share some fundamental properties
- Models contain only essential info we need to understand and reason about some aspect of a system
- A system model must address
  - What are the main entities in the system?
  - How do they interact?
  - What are the characteristics that affect their individual and collective behavior?
- Purpose of the model
  - Make explicit all relevant assumptions
  - Be able to make generalizations about what is possible, given the assumptions
  - Generalizations are algorithms or properties
- Kinds of fundamental models
  - Interaction
  - Failure
  - Security

Interaction models

- Algorithm: sequence of steps on one computer
- Distributed algorithm: sequences of steps at each computer plus message passing between them
- Hard problem: cannot predict message latencies well
- Major factors affecting interaction processes
  - Communication performance is often a bottleneck
  - Cannot maintain a single global notion of time
- Communication channels
  - Latency
  - Bandwidth
  - Jitter
- Clock drift and synchronization
- Variations on the interaction model (opposites)
  - Synchronous DS
  - Asynchronous DS

Synchronous DSs

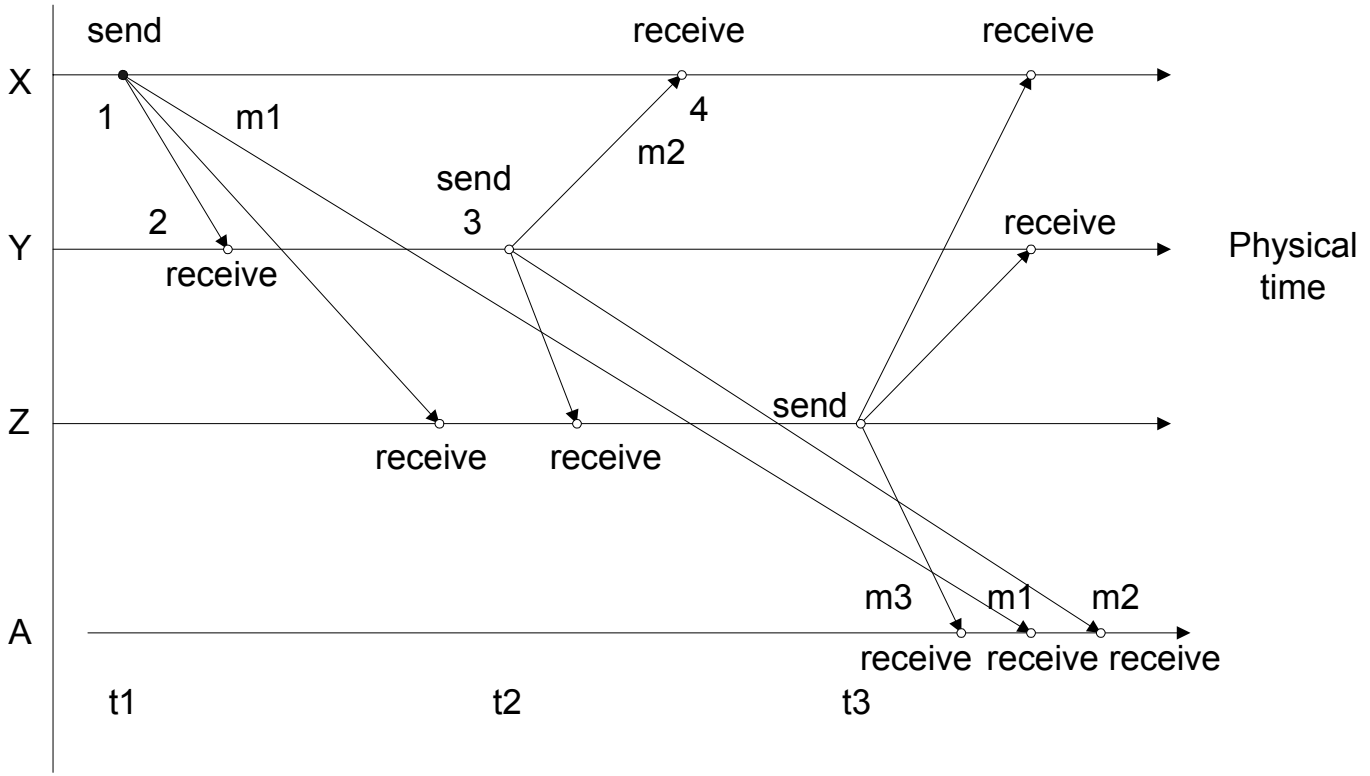
- Definition:
  - Time to execute each step of a process has known lower and upper bounds
  - Message delivery times have a known lower and upper bound
  - Clock drift times have a known lower and upper bound
- Strong assumptions about time!
- Q: how to set the bounds?
- Synchronous systems can be built… (how?)

Asynchronous DSs

- Definition: no bounds on
  - Process execution speeds
  - Message transmission delays
  - Clock drift rates
- Q: how would each affect your programming?
- Pepperland example:
  - Two Pepperland army divisions camped on hills
  - Blue meanies in the valley between
  - Pepperland armies must sent couriers which can get delayed arbitrariliy
  - Pepperland armies need to agree on who leads charge and when
  - Possible to agree who will lead charge: send (and resend) #troops
  - Impossible to agree when (at least in time to do it)

Event Ordering

- Sometimes useful to know about Event A and Event B
  - A happened before B, or vice versa
  - Both at same time (to some granularity)
- Example: displaying newsgroup messages: reply after original posting
- Possible to make inferrences even if not all orderings can be known:



Failure Models

- Lots of low-level ways a component can fail, in general
- Failure models generalize them into higher-level categories
- Failure models also extend a component's specification to include legal ways in which it can fail
  - Allows these contingencies to be handled by system designers
  - E.g., a system can tolerate some (kind and number of) failures
  - If failures outside model occur, then results undefined (usually catestrophic failure)

Omission and Arbitrary Failures

Class of failure	*Affects*	*Description*
Fail-stop	Process	Process halts and remains halted. Other processes may detect this state.
Crash	Process	Process halts and remains halted.Other processes may not be able to detect this state.
Omission	Channel	A message inserted I an outgoing message buffer never arrives at the other end's incoming message buffer.
Send-omission	Process	A process completes a *send*, but the message is not put in its outgoing message buffer.
Receive-omission	Process	A message is put in a process's incoming message buffer, but that process does not receive it.
Arbitrary (Byzantine)	Process or channel	Process/channel exhibits arbitrary behavior:it may send/transmit arbitrary messages at arbitrary times, commit omissions;a process may stop or take an incorrect step.

Timing Failures

Class of Failure	*Affects*	*Description*
Clock	Process	Process's local clock exceeds the bounds on its rate of drift from real time.
Performance	Process	Process exceeds the bounds on the interval between two steps.
Performance	Channel	A message's transmission takes longer than the stated bound.

Agreement in Pepperland with Failures

- If failures can happen (couriers get captured), then agreement cannot be reached on who should charge!
- Last message may not arrive, and sender never knows, so a correct algorithm has to work without it
- But this applies back to all previous messages!

Security Models

- Security of a distributed system can be achieved by
  - Securing the processes and channels they use for interacting
  - Protecting the objects they encapsulate against unauthorized use
- Definitions: principal and access rights

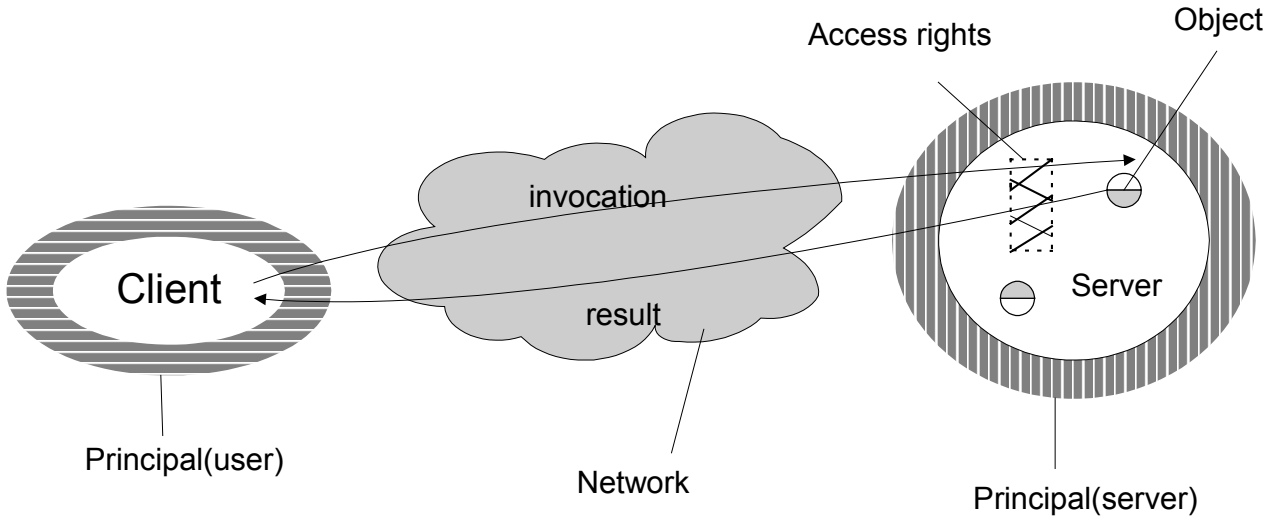


Figure 2.13 Objects and principals

Protecting Objects

- Access rights
  - Defn: who can access an object, and in what way
  - Example: “joe” can read this object (call methods that do not write state)
- Principal: entity with rights to do something
  - A user or a process typically
  - Rights can be delegated
- Server must
  - Verify identity of the principal calling it
  - Verify that principal has the right to do what its invocation calls for

Security Terms for our Model

- Enemy/adversary which can
  - Send a message to any process
  - Read any message sent by anyone
  - Copy any message sent by anyone

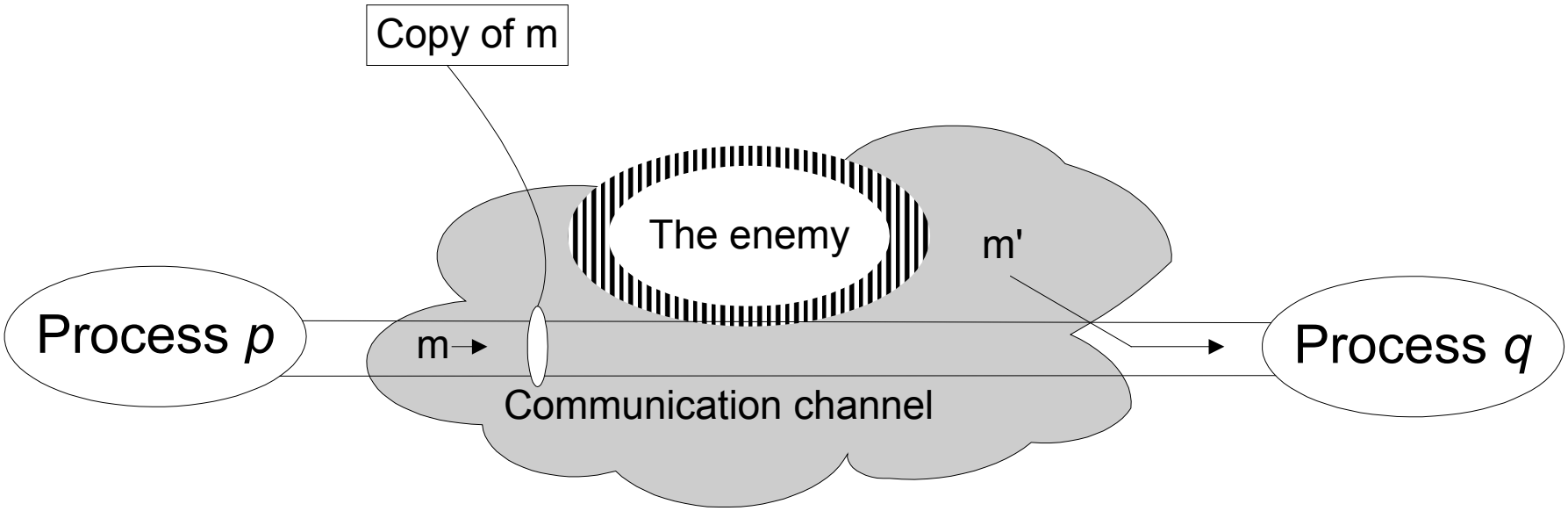


Figure 2.14 The enemy

Threats in the Security Model

- Threats to processes
  - Can't always determine identity of sender of a message
  - So servers cannot know to perform an operation or reject it
  - Clients can't tell if a reply ("result") message is from the server
- Threats to communication channels
  - Enemy can
    - Copy a message
    - Alter a message
    - Inject new messages
    - Replay a message later
  - All can be defeated with secure channels

Defeating Security Threats

- Cryptography: science of keeping messages secure
- Encryption: scrambling messages to hide their contents (very difficult to unscramble without the key)
- Authentication: using shared secrets and encryption to verify identities
- Secure channels: layer on top of existing communication service where
  - Each process reliably knows identity of principal on other side
  - Data transmitted have privacy maintained
  - Data transmitted have integrity maintained: nobody can alter it
  - Messages have timestamps (physical or logical) to prevent replaying or reordering of messages
- Other threats
  - Denial of service: overloading resource to prevent legitimate use
  - Mobile code is another source of vulnerability
- Security causes a lot of processing and management overhead!
- But “Do you feel lucky today?”