03 - Models

- Models
  - Physical
  - Architectural
  - Fundamental
- Coulouris 1
- Coulouris 2
- Saltzer_84.pdf

Physical models

- Early Distributed Systems
  - Late 70/early 80s
  - 10..100 nodes
  - Local network / Homogeneous / Few services
- Internet scale
  - 90s
  - Internet based / Network of networks / Static
  - Global / Heterogeneous (but server or client) / Open
- Contemporary
  - Mobile nodes (Wifi, GSM)
  - Ubiquitous (embedded in objects and environment)
  - Systems of systems (cloud)
  - New level of heterogeneity (architecture and capabilities)

Physical models

Distributed systems:	*Early*	*Internet-scale*	*Contemporary*
Scale	Small	Large	Ultra-large
Heterogeneity	Limited (typically relatively homogenous configurations)	Significant in terms of platforms, languages and middleware	Added dimensions introduced including radically different styles of architecture
Openness	Not a priority	Significant priority with range of standards introduced	Major research challenge with existing standards not yet able to embrace complex systems
Quality of service	In its infancy	Significant priority with range of services introduced	Major research challenge with existing services not yet able to embrace complex systems

Architectural models

- What entities are communicating in a DS?
  - Communication entities
- How they communicate?
  - Communication paradigms
- What roles and responsibilities they have?
- How are they mapped on the physical infrastructure?

Architectural models

Communicating entities (what is communicating)		*Communication paradigms (how they communicate)*		
System-oriented entities	*Problem-oriented entities*	*Interprocess communication*	*Remote invocation*	*Indirect communication*
Nodes	Objects	Message passing	Request-reply	Group communication
Processes	Components	Sockets	RPC	Publish-subscribe
	Web services	Multicast	RMI	Message queues
				Tuple spaces
				DSM

Architectural models

Communication entities

- DS = Processes + IPC
- Processes
  - Some systems don't have processes
  - Most systems have threads
    - the real communication endpoint
- Objects
  - Migrations of OO to DS
    - OO Designs and OO programming
  - Natural decomposition unit for a problem
  - Accessed by and interface

Architectural models

Communication entities

- Components
  - Provide interface like objects
  - Specify assumption
    - Dependencies are explicit and used to pair components
    - Contracts
- Web services
  - Implementation of Objects and Components
    - On the WEB
  - Identified by a URL
  - Defined/described/discovered by XML

Architectural models

Communication paradigms

- Remote invocation
  - Request-reply protocols
    - Low level
    - Programmer creates/sends messages
    - Example: HTTP

Architectural models

Communication paradigms

- Remote invocation
  - Remote procedure calls
    - Attributed to Birrel and Nelson (84)
    - RPC system hides
      - Distribution
      - Encoding/decoding messages
      - Passing of message
      - Semantic of the procedure call
    - Supports client/server
      - Server offer set of operations (by interfaces)
      - Clients call those operations directly as if local
    - Access and location transparency
      - (minimal)

Architectural models
Communication paradigms

- Remote invocation
  - Remote method invocation
    - Resembles RPC
      - But in a world of objects
    - Client objects invoke methods on remote objects
    - Underling details are hidden
    - May support
      - Object identity
      - Pass objects as parameter
    - Tight integration to OO languages

Architectural models
Communication paradigms

- Remote invocation
  - Two-way relationship (sender ↔ receiver)
    - Receiver identity is know
    - Both parties exist simultaneously at the same time
  - Direct communication
- Indirect communication
  - Space uncoupling
    - Sender does not know who is sending to
  - Time decoupling
    - Senders and receiver do not need to exist at the same time

Architectural models
Communication paradigms

- Indirect communication
  - Group communication
    - Delivery of messages to groups of recipients
    - One-to-many communication
    - Communication relies on groups abstraction
      - Recipients join groups
  - Publish-subscribe
    - Information dissemination/ distributed events
    - Producers(publishers) distribute
      - Information items of interest (events)
    - Consumers register the interest or events to receive

Architectural models

Communication paradigms

- Indirect communication
  - Message Queues
    - Point-to-point communication channel
    - Producer places message on Queue
    - Consumer retrieves message
    - Consumers are notified of message availability
  - Tuples space
    - Communication of performed by the access to shared structured data (tuples)
    - Add tuples to the persistent tuple space
    - Consumers can read or deleted existing tuples
    - Can be client server or P2P

Architectural models

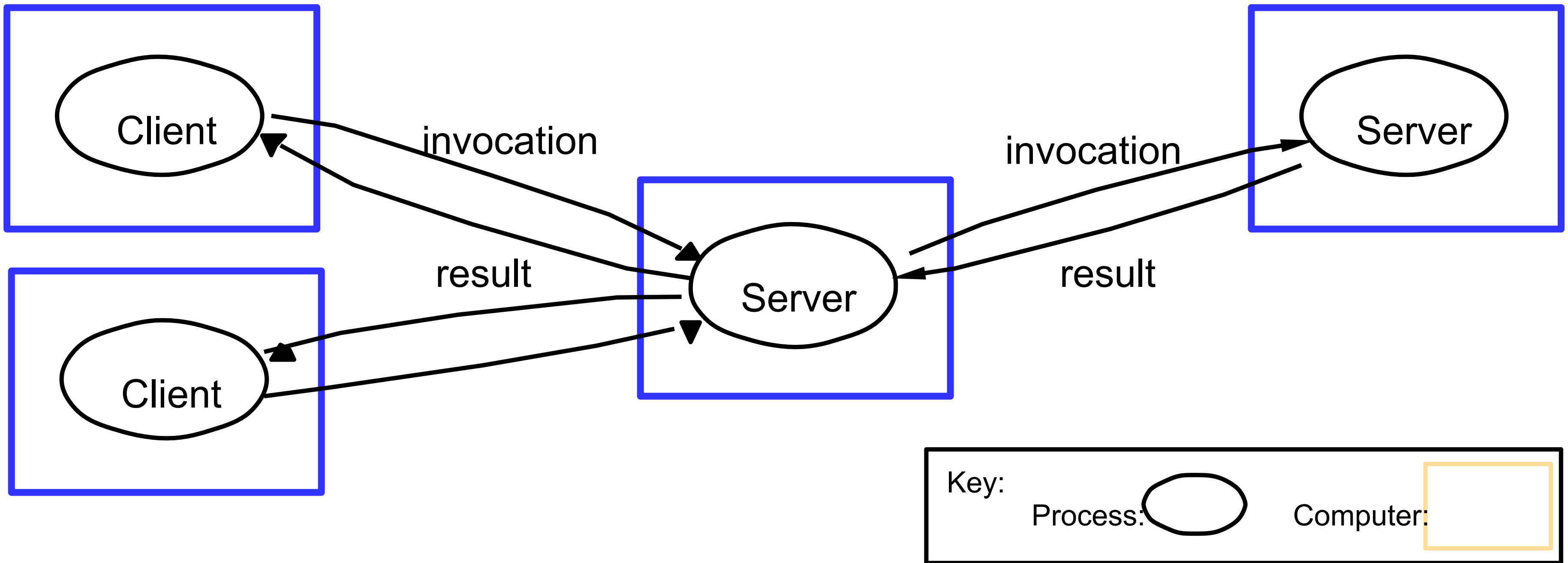
Communication paradigms

- Indirect communication
  - Distributed Shared Memory
    - DSM systems provide a view of a shared memory space
      - Composed of data on multiple remote nodes
    - Programmers are presented with the abstraction of reading/writing local memory
      - All accesses are to local address space
      - Although data can be on a remote node
    - Infrastructure guarantees
      - Copies of data are provided in a timely manner
      - Synchronization and consistency of data

Architectural models

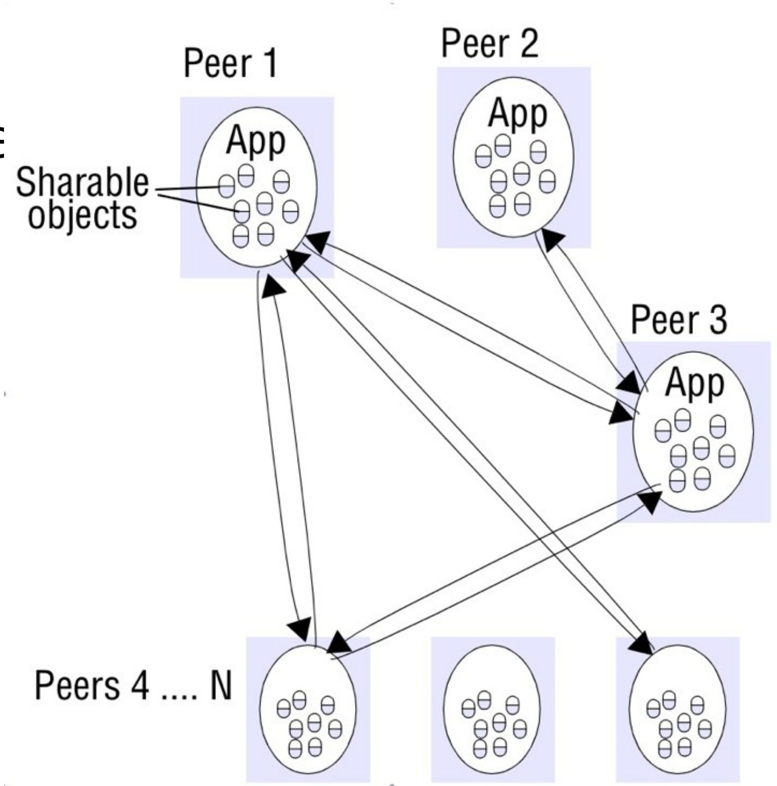
Roles and responsibilities

- Client-Server
  - Client processes interact with individual servers
    - (pottenctially) In separate hosts
    - To access shared resources
  - Servers can also be clients



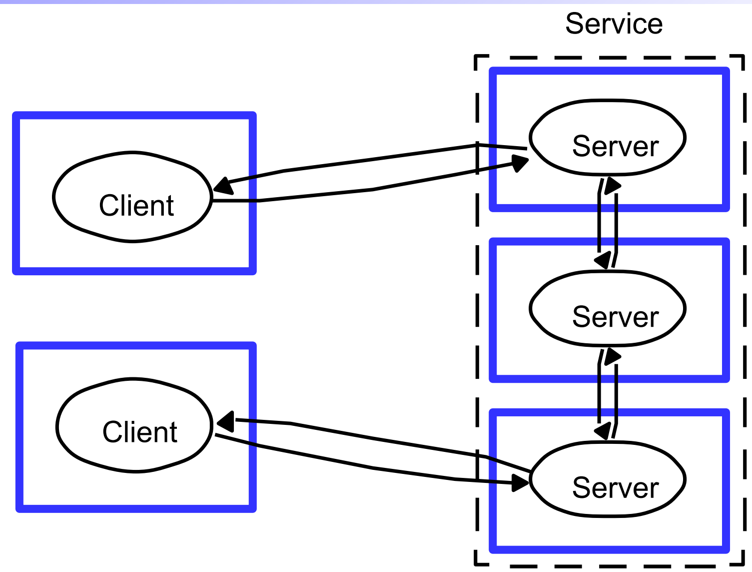
Architectural models
Roles and responsibilities

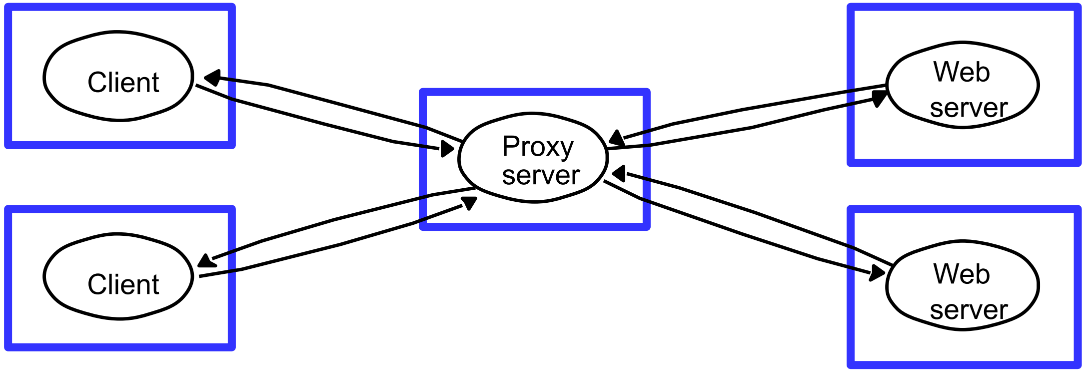
- Peer-to-peer
  - All processes have similar roles
    - Interacting cooperatively as *peers*
    - No distinctions between client or server
  - Resources owned by users
    - Can be put to use to support the service
    - Resources increase with the number of users
  - Data objects are shared and distributed
    - Distribution and replication increases complexity



Architectural models
Placement

- Where to place entities on the physical model?
  - Distribute service among several servers
    - Data may be partitioned
    - Data may be replicated
  - Caching
    - Store of recently used data
      - Locally to the client
      - On a separate server
    - Objects are retrieved from cache
      - If available





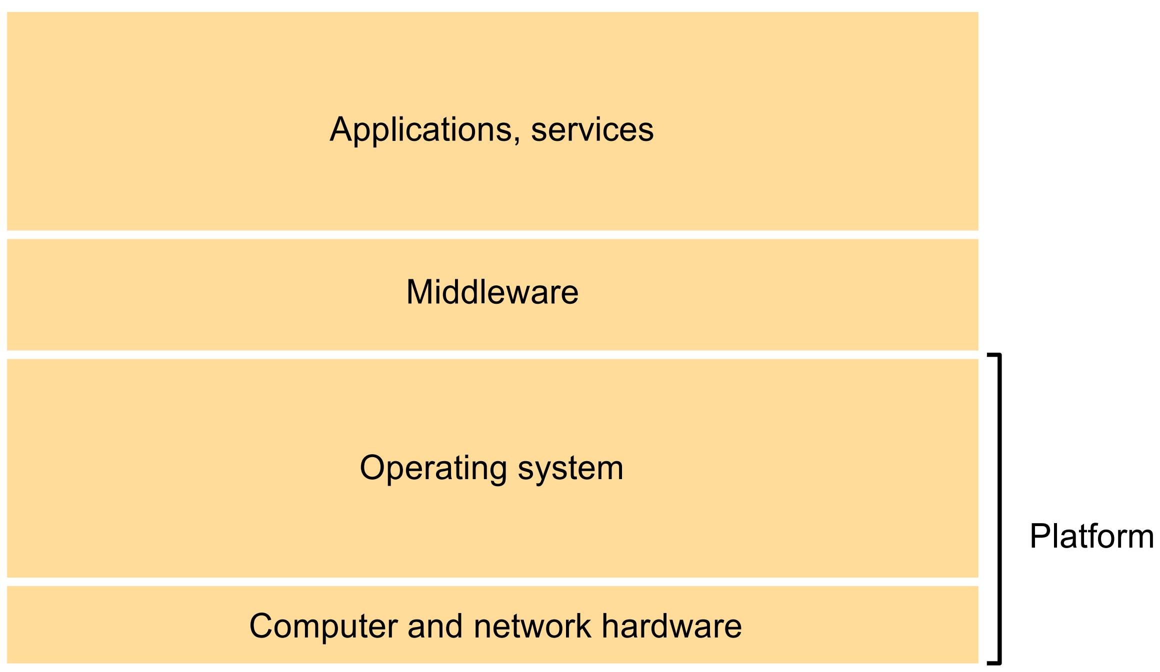
Architectural models

Placement

- Mobile code
  - Code is downloaded from the server
    - Executed on the client
    - No network delays
- Mobile agents
  - Code (and data) roams on computers
    - Executes on behalf of other
    - Invokes local services
    - Lower execution time
      - Code transfer + local invocation vs remote invocation

Architectural models
Architectural patterns

- Layering
  - Related to abstraction
  - Each layer offers an abstraction
  - Higher layers not aware of “lower” implementations



Architectural models

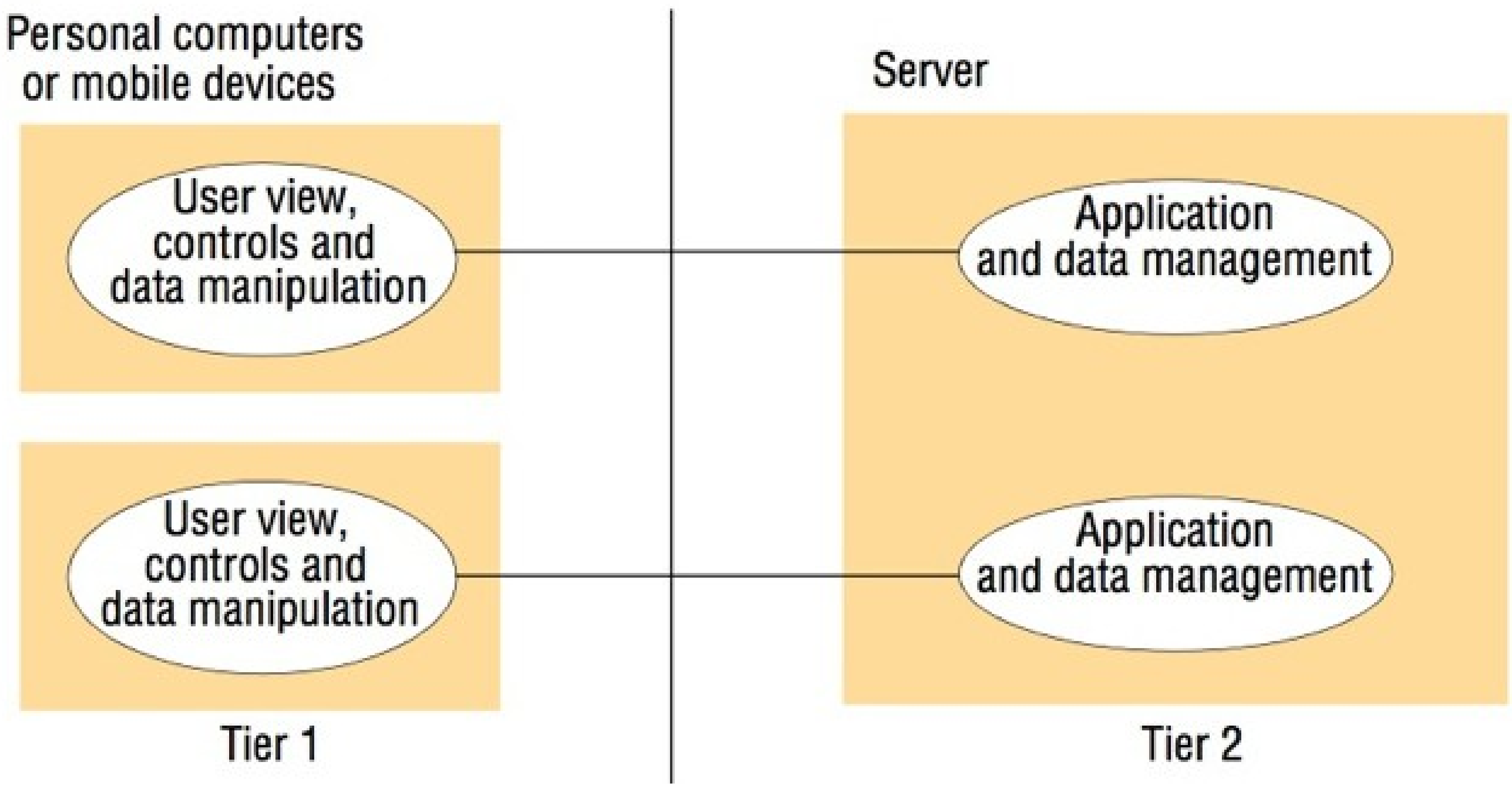
Architectural patterns

- Tiered
  - Related to composition
  - Complements layerings
    - Layering: vertical organization of services into layers
    - Tiering: Distribution of a given layer into appropriate servers
  - Presentation logic
  - Application logic
  - Data logic

Architectural models

Architectural patterns

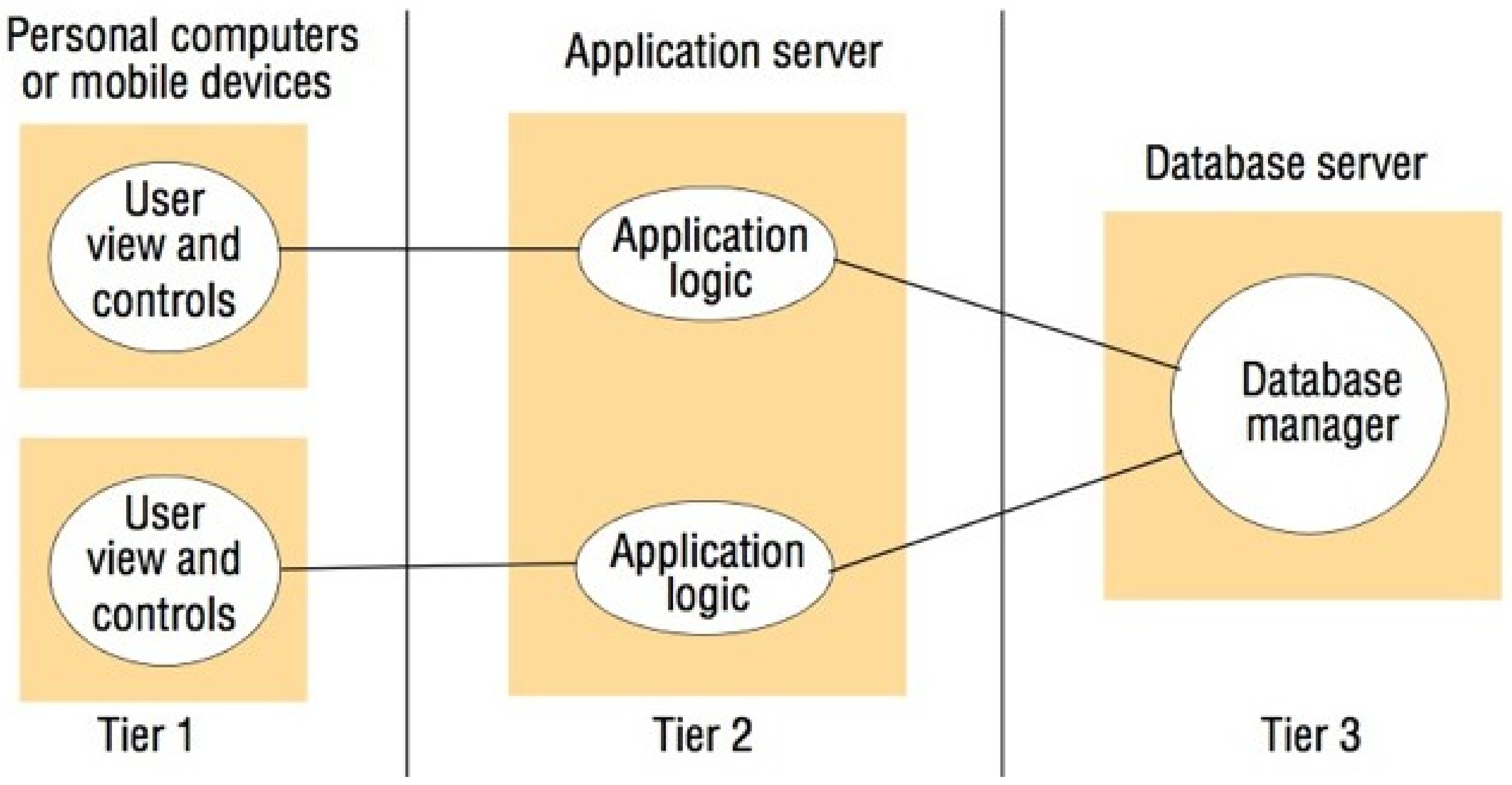
- 2-Tiered`



Architectural models

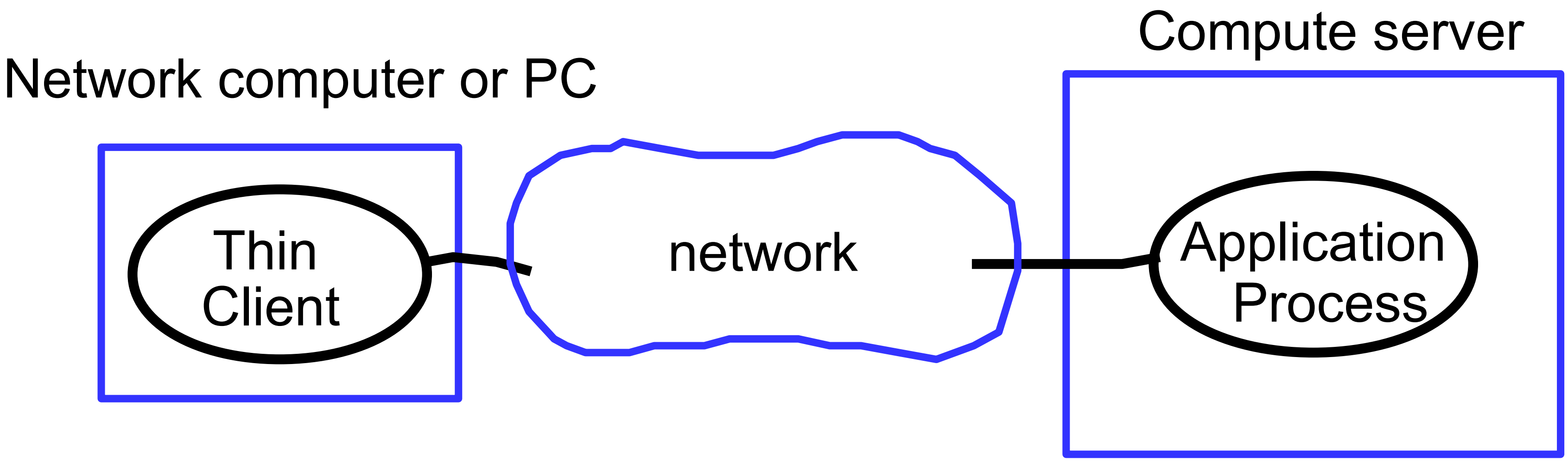
Architectural patterns

- 3-Tiered



Architectural models
Architectural patterns

- Thin-Client
  - Moves complexity away from end-user
    - Client has no logic only presentation



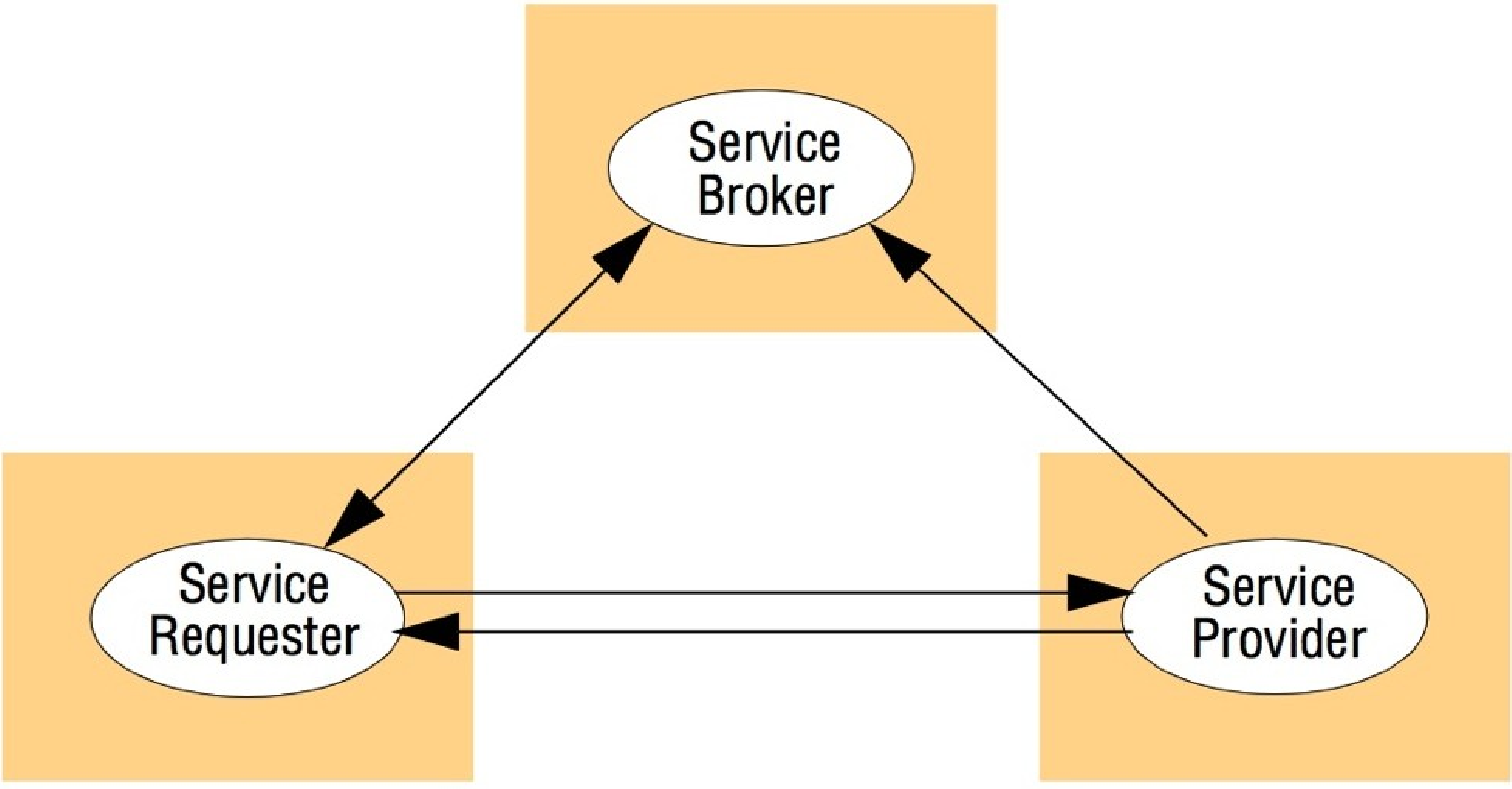
Architectural models
Architectural patterns

- Proxy
  - Offers the same interface as the server
  - Located on the local client
  - Is contacted and redirect calls to remote node
  - Offers location transparency
  - Can encapsulate other functionalities
    - Placement policies of replication
    - Caching
    - authentication, ...

Architectural models

Architectural patterns

- Brokerage
  - Supporting interoperability



Fundamental models

- Describe the general and fundamental characteristics of a DS
  - Not how it is implemented
- Define the assumptions about the system
- Allows the generalization about (imp)possibilities
  - General purpose algorithms
  - Desirable properties that are guaranteed
- Interaction
- Failure
- Security

Fundamental models

Interaction

- Assumptions about the communication channels
  - Latency: delay between start of message transmission and receiving
    - Time taken on transmission
    - Delay assessing network
    - Time of message processing (on the OS)
  - Bandwidth: Amount of information that can be transmitted on a give time
  - Jitter: variation in time take to deliver a series of messages

Fundamental models

Interaction

- Assumptions about computer clocks
  - Each computer has local clock
    - Used to obtain current time
  - Simultaneous clock read render different values
    - Clocks on computers drift at different rates
  -

Fundamental models

Interaction

- Synchronous Distributed Systems
  - Time to execute each step
    - Has lower bound
    - Has higher bound
  - Messages
    - Are received within a known bounded time
  - Process clocks
    - Drift rate has known bound
  - Can be built if
    - It is possible to guarante previous bounds

Fundamental models

Interaction

- Asynchronous Distributed Systems
  - There are no bounds for
    - Process execution speed
    - Message transmission delays
    - Clock drift rates
  - Internet
    - Good example
    - No limits to server or network load
  - Email
    - May take days to arrive
    - Servers can have drifted clock

Fundamental models

Failure

- In a DS processes and communication can fail
- Failure model defines how failures can occur
  - In order to provide understanding of their effects
- Omission failures
  - Process omission failures:
    - process does not execute the task (crashes)
    - Can be detected using timeouts
  - Communication omission failures
    - A sent message is not delivered to the receptor
  - Both benign

Fundamental models

Failure

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 in 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.

Fundamental models

Failure

- Timing failures
  - Applicable to synchronous systems
    - With limits to execution, delivery times and clock drifts

Class of failure	Affects	Description
Clock	Process	Process's local clock exceeds the bounds on its
Performance	Process	Process exceeds the bounds on the interval between two steps.
Performance	Channel	A message's transmission takes longer than the stated bound.

Fundamental models
Failure

- Arbitrary (Bynzatine) failures
  - Any error can occur
  - A process set wrong values in data items
  - A process answers a wrong values
  - A process arbitraly fails
    - No way to distinguish Process fail from no answer :(
  - A message gets corrupted
  - A bogus messages is created
  - A messages is intercepted from delivery
  - A messages is delivered twice

Fundamental models
Masking failures

- A reliable system can be composed of
  - Unreliable components that exhibit failures
- Knowledge of each component failures
  - Allows a service to mask them
- Hiding a failure
  - e.g. retrying
- Converting to a more acceptable failure
  - Recovering an old version of the file

Fundamental models

Reliability on one-to-one communication

- Basic communication channels
  - can exhibits failures (e.g. omission)
  - But can be used to build a service that masks some failures
    - Provides reliable communication
- Reliable communication
  - Validity: any message is eventually delivered
  - Integrity: the received message is identical to the sent one

Network Platform

- Networking issues
  - Performance
  - Scalability
  - Reliability
  - Security
  - Mobility
  - QoS
- Types of networks
  - Personal Area
  - (wireless) Local Area
  - (wireless) Wide Area
  - (wireless) Metropolitan Area
- Handled by the platform
  - OS + HW
  - Low level protocols

Middleware

- SW layer that
  - Provides a programming abstraction
  - Masks underlying heterogeneity
    - Network, HW, OS, programming languages
  - Provides a uniform computational model
    - To be used by programmers of distributed applications
    - RMI, remote event notification, distributed transactions, ...

Middleware

Major categories:	*Subcategory*	*Example systems*
Distributed objects (Chapters 5, 8)	Standard	RM-ODP
	Platform	CORBA
	Platform	Java RMI
Distributed components (Chapter 8)	Lightweight components	Fractal
	Lightweight components	OpenCOM
	Application servers	SUN EJB
	Application servers	CORBA Component Model
	Application servers	JBoss
Publish-subscribe systems (Chapter 6)	-	CORBA Event Service
	-	Scribe
	-	JMS
Message queues (Chapter 6)	-	Websphere MQ
	-	JMS
Web services (Chapter 9)	Web services	Apache Axis
	Grid services	The Globus Toolkit
Peer-to-peer (Chapter 10)	Routing overlays	Pastry
	Routing overlays	Tapestry
	Application-specific	Squirrel
	Application-specific	OceanStore
	Application-specific	Ivy
	Application-specific	Gnutella

Middleware
Limitations

- Some application rely only on middleware
  - If suitable for client-serves can use RPC
    - Name/Address database
- Not all issues can be handled by middleware
  - Some dependability aspects
- Large file transfer over unreliable link
  - TCP offers some error detection and correction
  - TCP does not recover from major network interruption
  - If service offers a new level of fault tolerance
    - Must maintain a progress level
    - Must resumes transmission with a new TCP connection

Middleware

Limitations

- End-to-end argument
  - Saltzer et al (84)
- Some communication functions can only be completely and reliably implemented
  - With knowledge and help of the application standing at the end points of the communication system
- In the previous case:
  - TCP does not how to restart a file transmission
  - Client must know where to restart
  - Server must receive information about restart