Dystributed Systems

Most slides have been adapted from
«Distributed systems: concepts and design», 3/E Coulouris et al. (Chapter 1+2)
«*Operating Systems: Internals and Design Principles”, 7/E* W. Stallings (Chapter 16)

Sistemi di Calcolo 2

Riccardo Lazzeretti

INTRODUCTION TO DYSTRIBUTED SYSTEMS

Dystributed system definition

- A distributed system is a set of spatially separated entities, each of them with a certain computational power, that are able to communicate and to coordinate among themselves for reaching a common goal and that appears to its users as a single coherent system

Introduction to dystributed systems

- Why do we develop distributed systems?
  - availability of powerful yet cheap microprocessors (PCs, workstations), continuing advances in communication technology,

- **What is a distributed system?**
- A distributed system is a collection of independent computers that appear to the users of the system as a single system.

- Examples:
  - Network of workstations
  - Distributed manufacturing system (e.g., automated assembly line)
  - Network of branch office computers

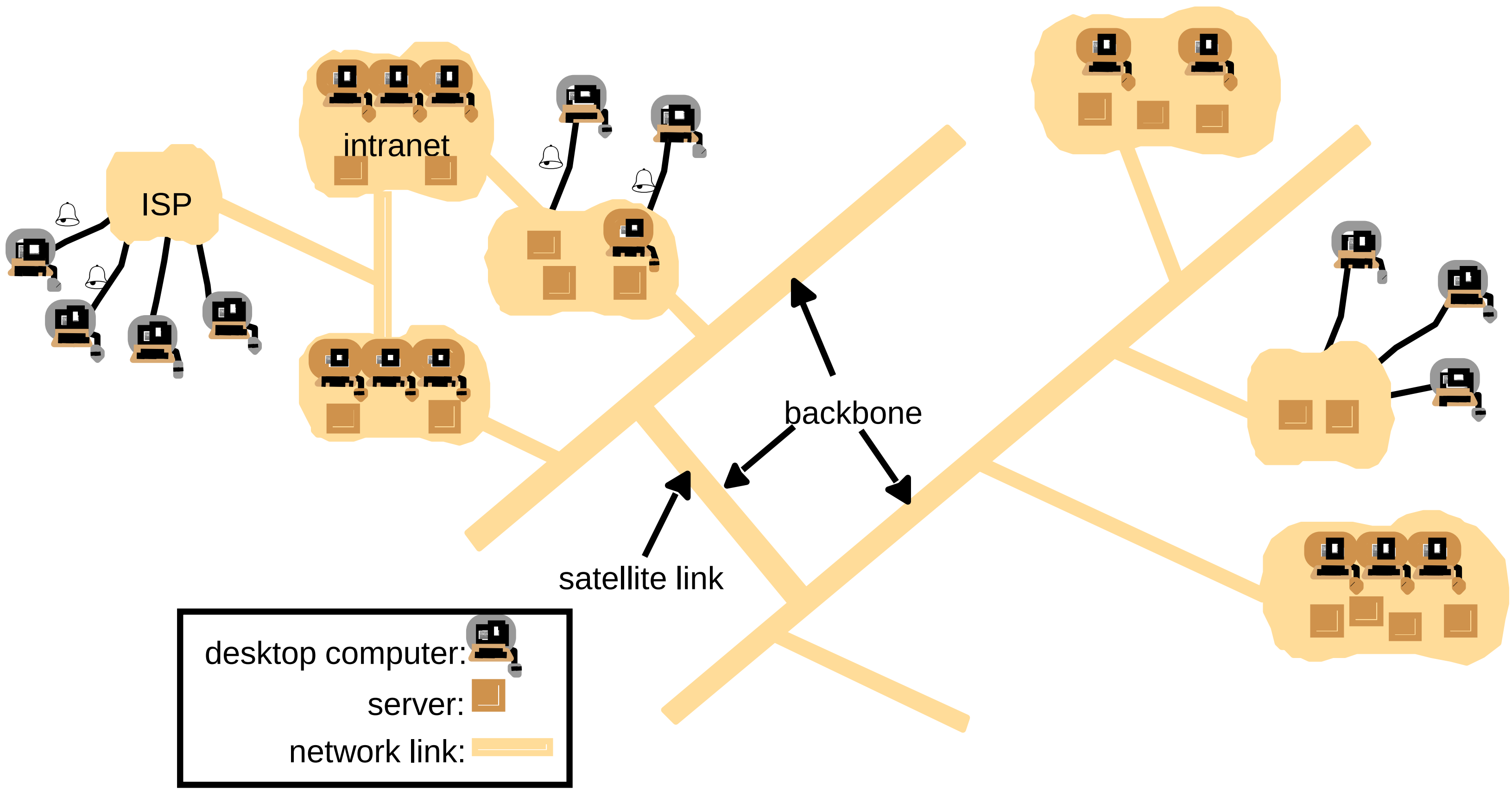
Selected application domains and associated networked applications

Finance and commerce	eCommerce e.g. Amazon and eBay, PayPal, online banking and trading
The information society	Web information and search engines, ebooks, Wikipedia; social networking: Facebook and MySpace.
Creative industries and entertainment	online gaming, music and film in the home, user-generated content, e.g. YouTube, Flickr
Healthcare	health informatics, on online patient records, monitoring patients
Education	e-learning, virtual learning environments; distance learning
Transport and logistics	GPS in route finding systems, map services: Google Maps, Google Earth
Science	The Grid as an enabling technology for collaboration between scientists
Environmental management	sensor technology to monitor earthquakes, floods or tsunamis

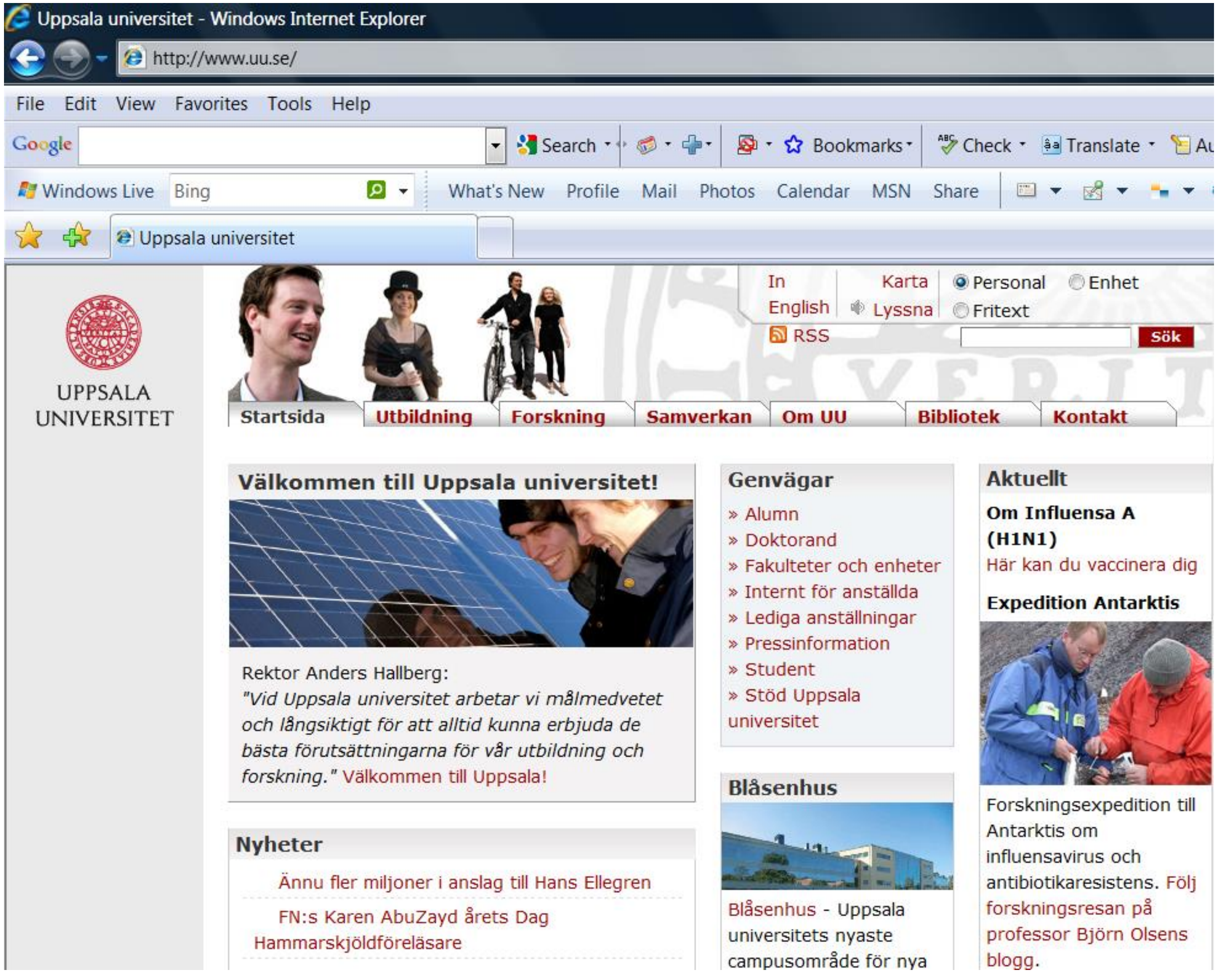
Examples of Distributed Systems

- Local Area Network and Intranet
- Database Management System
- Automatic Teller Machine Network
- Internet/World-Wide Web
- Pervasive Systems and Ubiquitous Computing
- Service Oriented Architecture
- Virtual networks
- Peer-to-peer (P2P)
- Cloud Computing
- Big Data Computing

Internet


intranet
ISP
backbone
satellite link
desktop computer:
server:
network link:

World-Wide-Web



Massively multiplayer online games (MMOGs)

- Offer an immersive experience whereby very large numbers of users interact through the Internet with a persistent virtual world
- Include:
  - Complex playing arenas
  - Social systems
  - Financial systems
- Require:
  - Fast responses
  - Real time propagation of events
  - Consistent view of the entire world

Advantages of Distributed Systems over Centralized Systems

- **Economics**: a collection of microprocessors offer a better price/performance than mainframes. Low price/performance ratio: cost effective way to increase computing power.
- **Speed**: a distributed system may have more total computing power than a mainframe. Ex. 10,000 CPU chips, each running at 50 MIPS. Not possible to build 500,000 MIPS single processor since it would require 0.002 nsec instruction cycle. Enhanced performance through load distributing.
- **Inherent distribution**: Some applications are inherently distributed. Ex. a supermarket chain.
- **Reliability**: If one machine crashes, the system as a whole can still survive. Higher availability and improved reliability.
- **Incremental growth**: Computing power can be added in small increments. Modular expandability
- **Another deriving force**: the existence of large number of personal computers and IoTs, the need for people to collaborate and share information.

Advantages of Distributed Systems over Independent PCs

- **Data sharing**: allow many users to access to a common data base
- **Resource Sharing**: expensive peripherals like color printers
- **Communication**: enhance human-to-human communication, e.g., email, chat
- **Flexibility**: spread the workload over the available machines

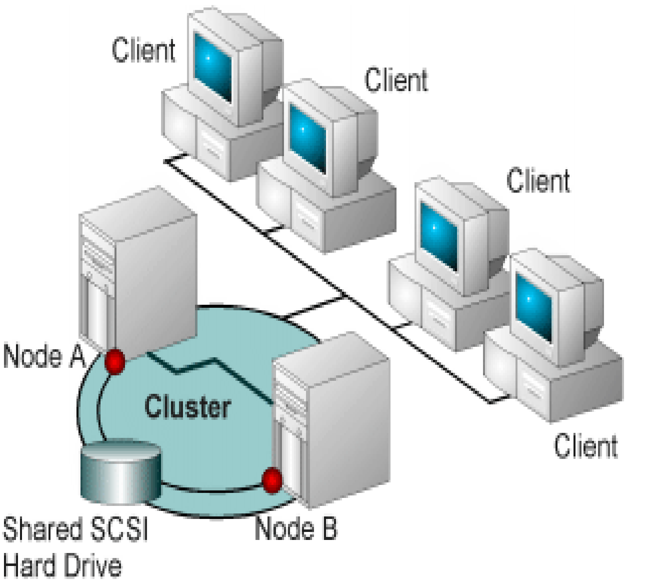
Disadvantages of Distributed Systems

- **Software**: difficult to develop software for distributed systems
- **Network**: saturation, lossy transmissions
- **Security**: easy access also applies to secrete data

Primary Goal: sharing data/resources

Problems

- Synchronization
- Coordination



Coordination

take into account the following condition that deviates from centralized systems:

1. Temporal and spatial concurrency
2. No global Clock
3. Failures
4. Unpredictable latencies

These limitations restrict the set of coordination problems we can solve in a distributed setting

Design Issues of Distributed Systems

- Heterogeneity
- Openness
- Security
- Scalability
- Reliability
- Concurrency
- Flexibility
- Performance
- Transparency

Heterogeneity

- networks
- computer hardware
- operating systems
- programming languages
- implementations by different developers
- mobile code

Opennes

- Dystributed services may be usable
- Their interfaces must be published
- Open distributed systems are based on the provision of
  - a uniform communication mechanism
  - published interfaces for access to shared resources.
- Open distributed systems can be constructed from heterogeneous hardware and software
  - the conformance of each component to the published standard must be carefully tested

Security

- confidentiality
  - protection against disclosure to unauthorized individuals
- integrity
  - protection against alteration or corruption
- availability
  - protection against interference with the means to access the resources

Scalability

- Systems grow with time or become obsolete.
  - No linear growth
- Challenges:
  - Controlling the cost of physical resources
  - Controlling the performance loss
  - Preventing software resources running out
  - Avoiding performance bottlenecks
- Examples of bottlenecks
  - Centralized components: a single mail server
  - Centralized tables: a single URL address book
  - Centralized algorithms: routing based on complete information

Reliability

- Distributed system should be more reliable than single system.
- Example: 3 machines with .95 probability of being up. 1-.05^3 probability of being up.

– Fault tolerance
  - Detecting failures
  - Masking failures
  - Tolerating failures
  - Fault recovery
  - Redundancy

Concurrency

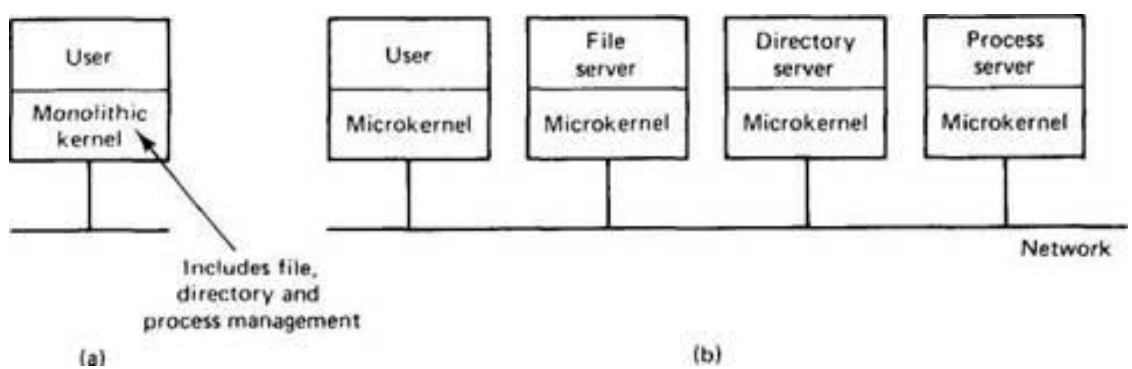
- Do I really need to explain this one?
- Yes, I do
- Spatial concurrency
- Time concurrency

Flexibility

- Make it easier to change

\- Monolithic Kernel: systems calls are trapped and executed by the kernel. All system calls are served by the kernel, e.g., UNIX.

\+ Microkernel: provides minimal services.
1) InterProcess Communication
2) some memory management
3) some low-level process management and scheduling
4) low-level I/O



Performance

- Without gain on this, why bother with distributed systems.
- Performance loss due to communication delays:
  – fine-grain parallelism: high degree of interaction
  + coarse-grain parallelism
- Performance loss due to making the system fault tolerant.

Transparency

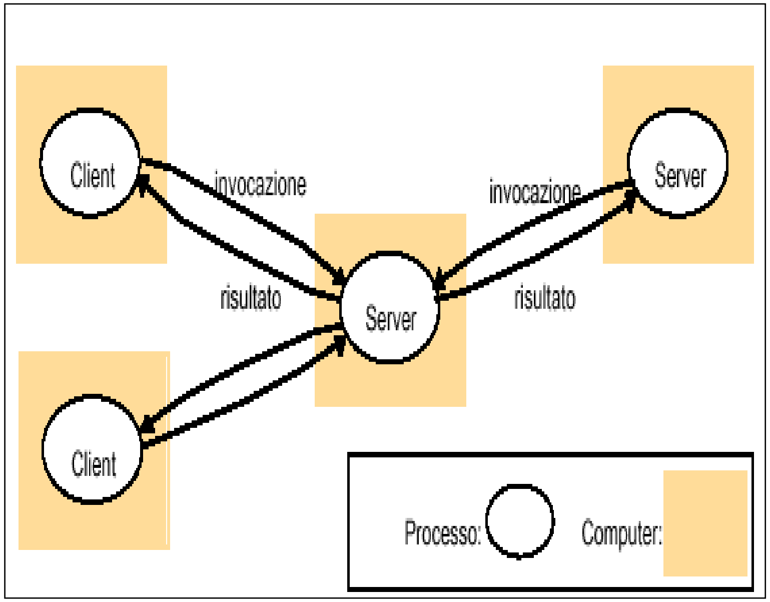
- How to achieve the single-system image, i.e., how to make a collection of computers appears as a single computer.
  - **Access Transparency**: local and remote resources are accessed using the same operations
  - **Location Transparency**: users cannot tell where hardware and software resources such as CPUs, printers, files, data bases are located.
  - **Concurrency Transparency**: The users are not aware of the existence of other users. Need to allow multiple users to concurrently access the same resource. Lock and unlock for mutual exclusion.
  - **Replication Transparency**: OS can make additional copies of files and resources without users noticing.

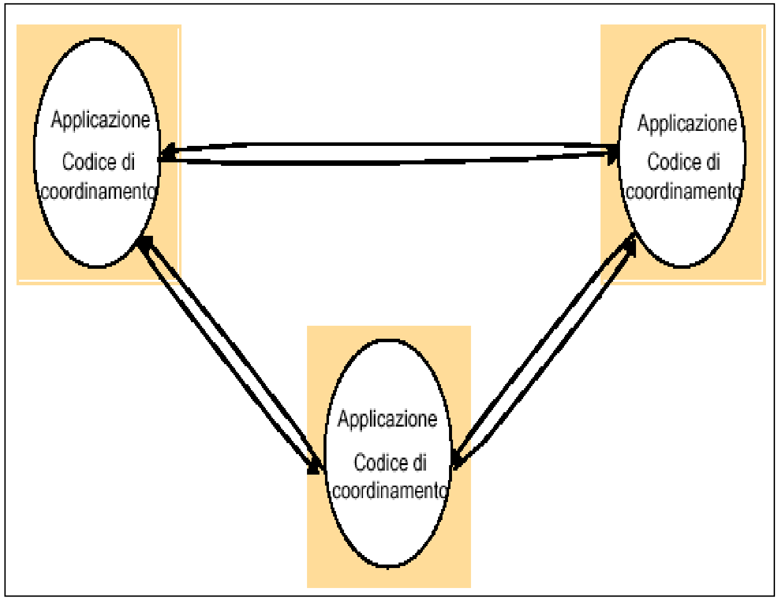
Types of transparency

- **Failure transparency:** enables the concealment of faults, allowing users and application programs to complete their tasks despite the failure of hardware or software components
- **Migration Transparency**: resources and clients must be free to move from one location to another without affecting the operations or requiring their names changed.
  E.g., /usr/lee, /central/usr/lee
- **Performance transparency** allows the system to be reconfigured to improve performance as loads vary.
- **Scaling transparency** allows the system and applications to expand in scale without change to the system structure or the application algorithms.
- **Parallelism Transparency**: Automatic use of parallelism without having to program explicitly. The holy grail for distributed and parallel system designers.

• Users *do not always want* complete transparency

Interaction Models

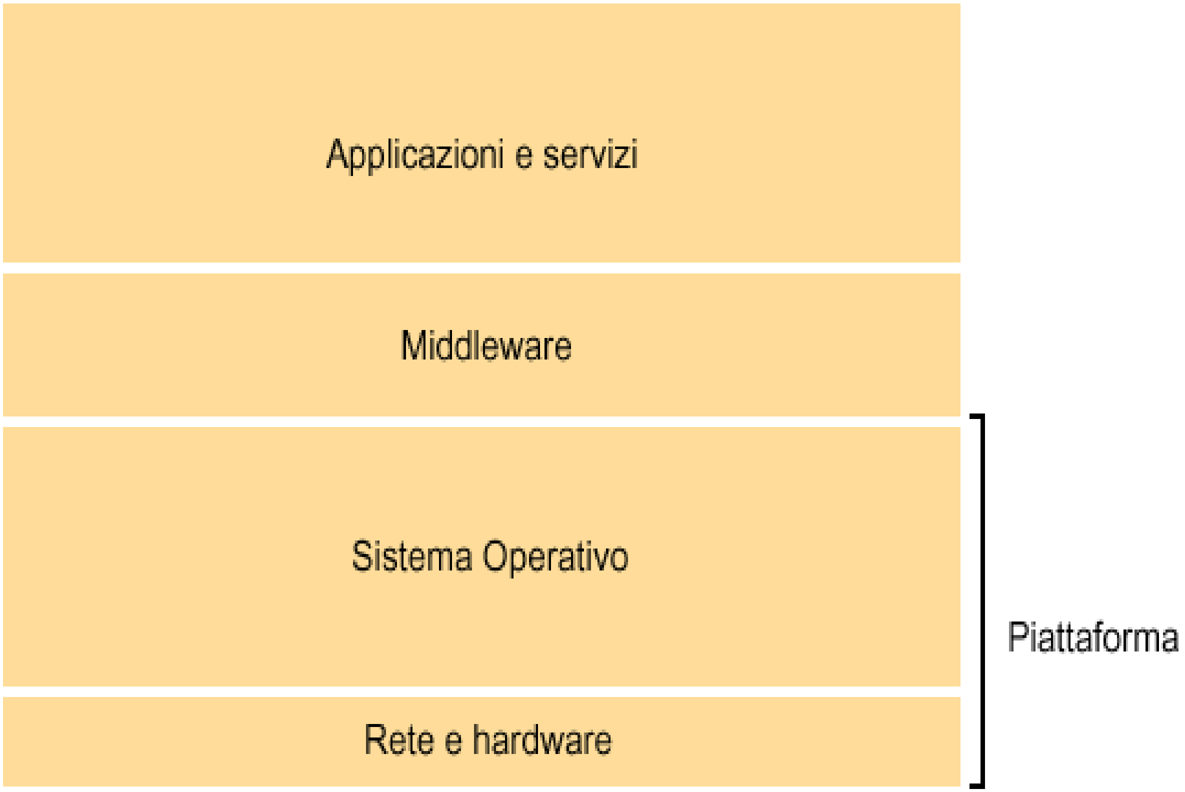


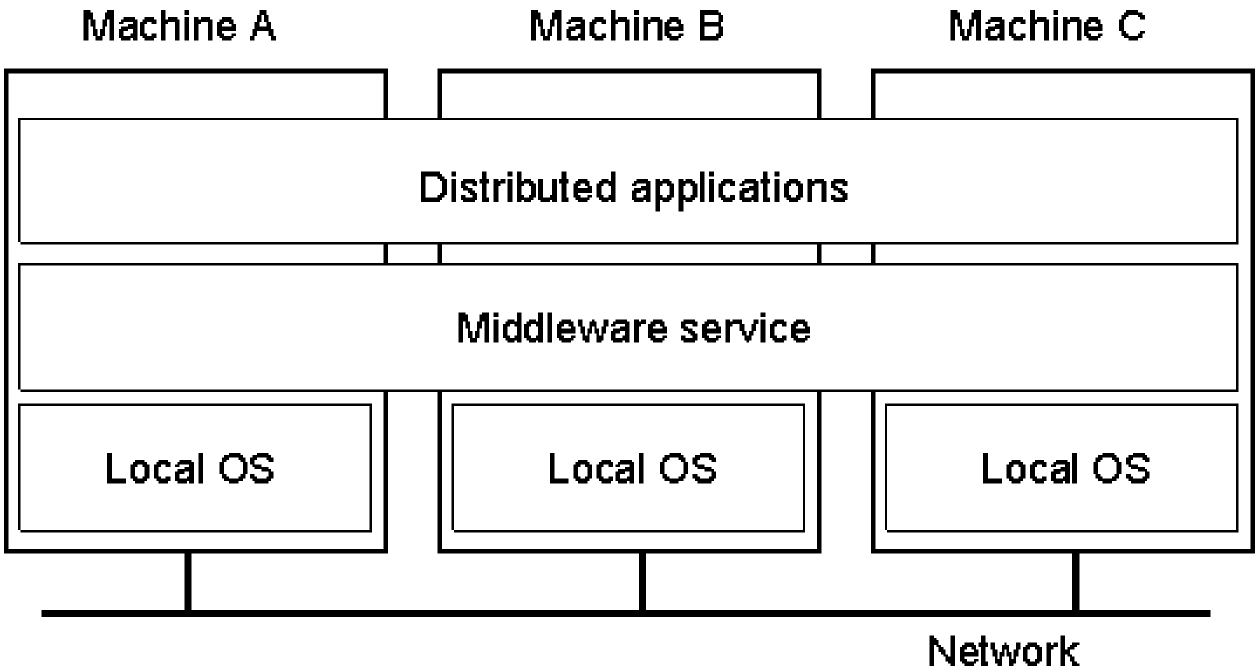


client/server

peer-to-peer

Layering hw and sw





A distributed system organized as middleware.
Note that the middleware layer extends over multiple machines.

Middleware problems

- **Heterogeneity**: OS, clock speeds, data representation, memory, architecture HW
- **Local Asynchrony**: load on a node, different HW, Interrupts
- **Lack of global knowledge:** knowledge propagates through messages whose propagation time will be much slower than the time taken by the execution of an internal event
- **Network Asynchrony**: propagation times of message can be unpredictable
- **Failures of nodes or network partitions**
- **Lack of a global order of events**
- **Consistency vs Availability vs Network Partitions**

This limits the set of problems that can be solved through deterministic algorithms on some distributed systems

Middleware

- To achieve the true benefits of the distributed system, developers must have a set of tools that provide a uniform means and style of access to system resources across all platforms
- This would enable programmers to build applications that look and feel the same
- Enable programmers to use the same method to access data regardless of the location of that data
- The way to meet this requirement is by the use of standard programming interfaces and protocols that sit between the application (above) and communications software and operating system (below)

Logical View of Middleware

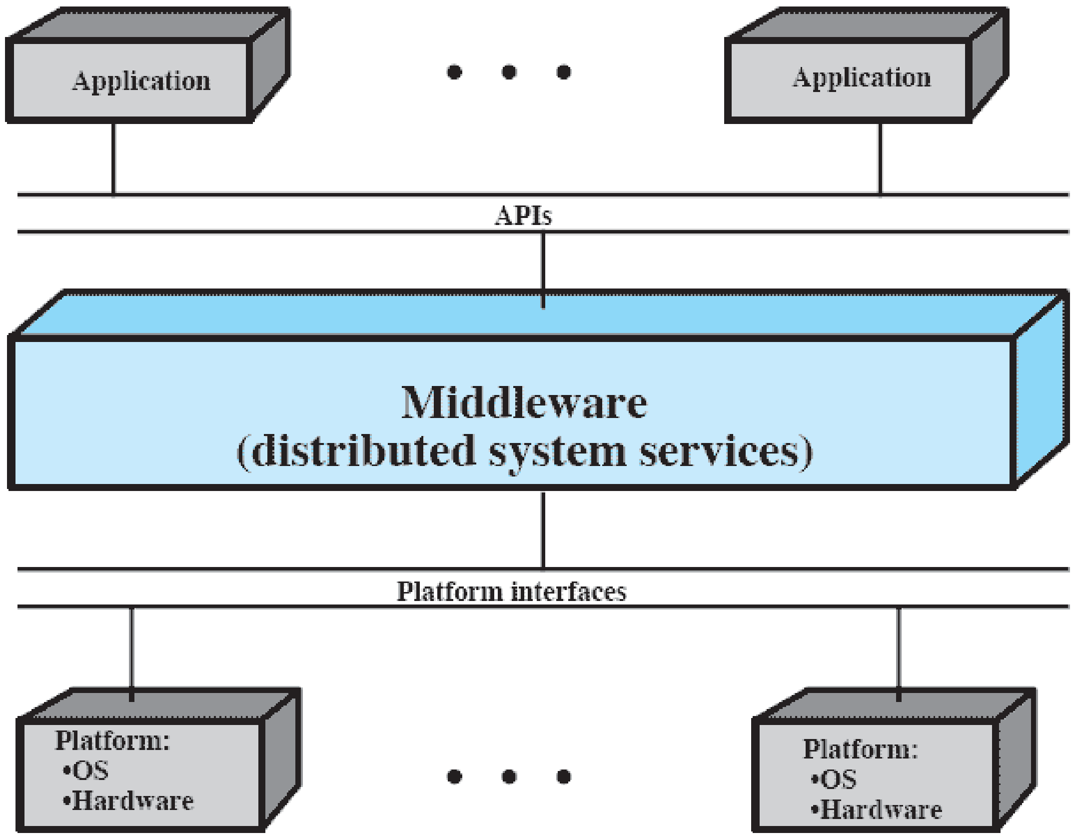


Figure 16.9 Logical View of Middleware

CLIENT/SERVER COMPUTING

Client/Server Computing

- Client machines are generally single-user PCs or workstations that provide a highly user-friendly interface to the end user
- Each server provides a set of shared services to the clients
- The server enables many clients to share access to the same database and enables the use of a high-performance computer system to manage the database

Generic Client/Server Environment

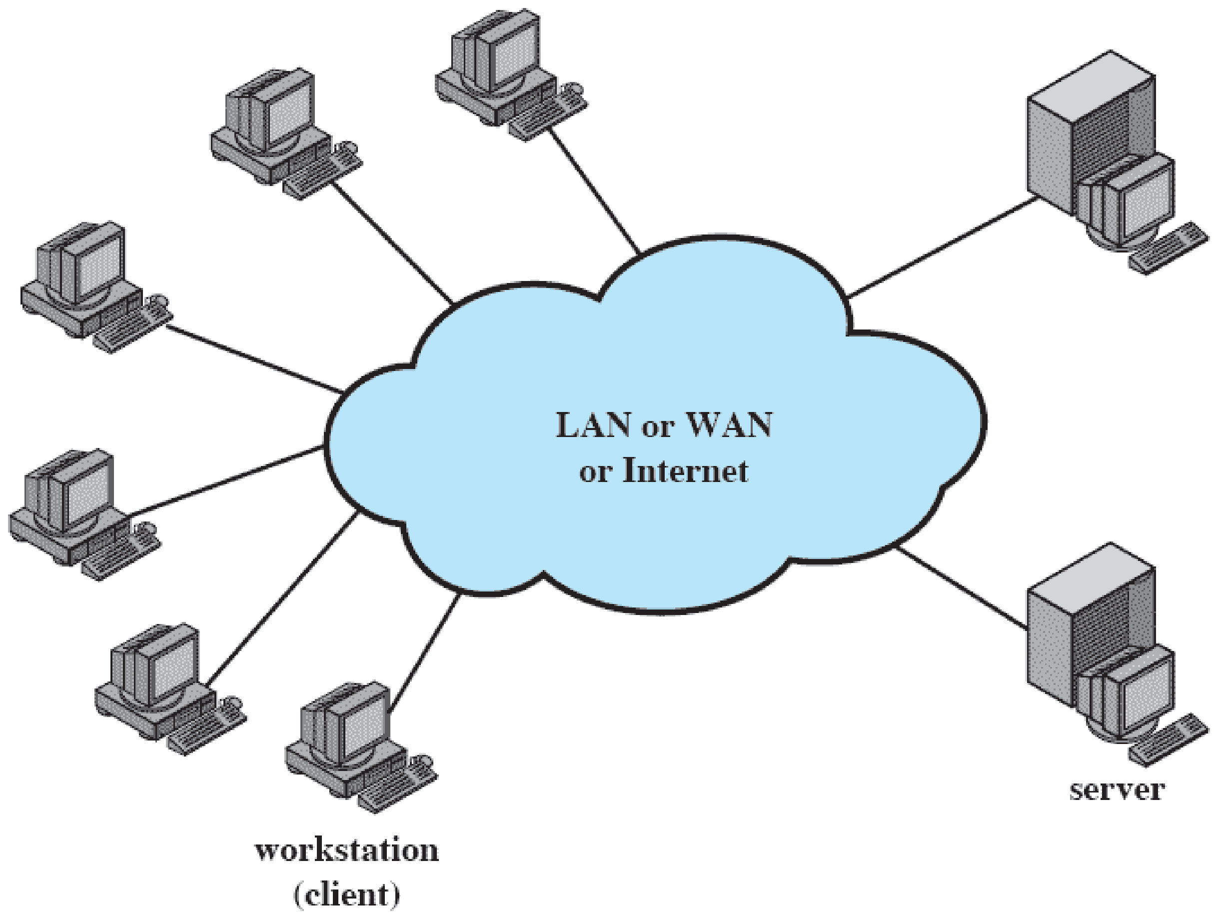


Figure 16.1 Generic Client/Server Environment

Client/Server Characteristics

- A client/server configuration differs from other types of distributed processing:
  - there is a heavy reliance on bringing user-friendly applications to the user on his or her own system
  - there is an emphasis on centralizing corporate databases and many network management and utility functions
  - there is a commitment, both by user organizations and vendors, to open and modular systems
  - networking is fundamental to the operation

Client/Server Applications

- Bulk of applications software executes on the server
- Application logic is located at the client
- Presentation services in the client

Client/Server Applications

- The key feature of a client/server architecture is the allocation of application-level tasks between clients and servers
- Hardware and the operating systems of client and server may differ
- These lower-level differences are irrelevant as long as a client and server share the same communications protocols and support the same applications

Client/Server Applications

- It is the communications software that enables client and server to interoperate
  - principal example is TCP/IP
- Actual functions performed by the application can be split up between client and server in a way that optimizes the use of resources
- The design of the user interface on the client machine is critical
  - there is heavy emphasis on providing a graphical user interface (GUI) that is easy to use, easy to learn, yet powerful and flexible

Generic Client/Server Architecture

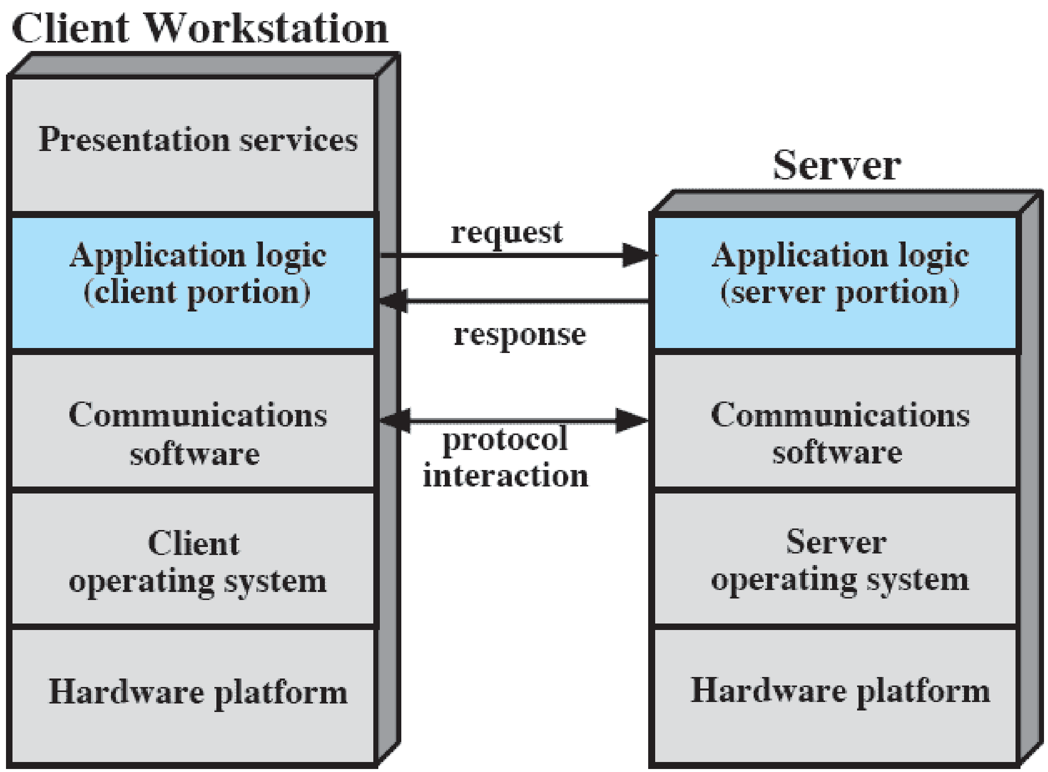


Figure 16.2 Generic Client/Server Architecture

Distributed Application Components

- Business-oriented applications (payroll, order entry, customer tracking, inventory control, etc.) contain four general components:
  - **Presentation logic**: user interface.
  - **I/O processing logic**: data validation.
  - **Business processing logic**: business rules and calculations.
  - **Data storage logic**: constraints such as primary keys, referential integrity, and actual data retrieval.

While almost all applications contain those four general components, for any given application those components need not be part of the same program, resident on the same computer, written in the same language, nor written by the same group of programmers.

Questions in application component development

- Decisions to make:
  - What language should a component be written in?
  - What hardware resource should a component reside upon?
- Information needed to make the decision:
  - How often will the component change?
    - Language changes.
    - Platform changes.
    - Business changes.
  - Who is responsible for maintaining the component?
  - How long is the application supposed to last?

Role of Middleware in Client/Server Architecture

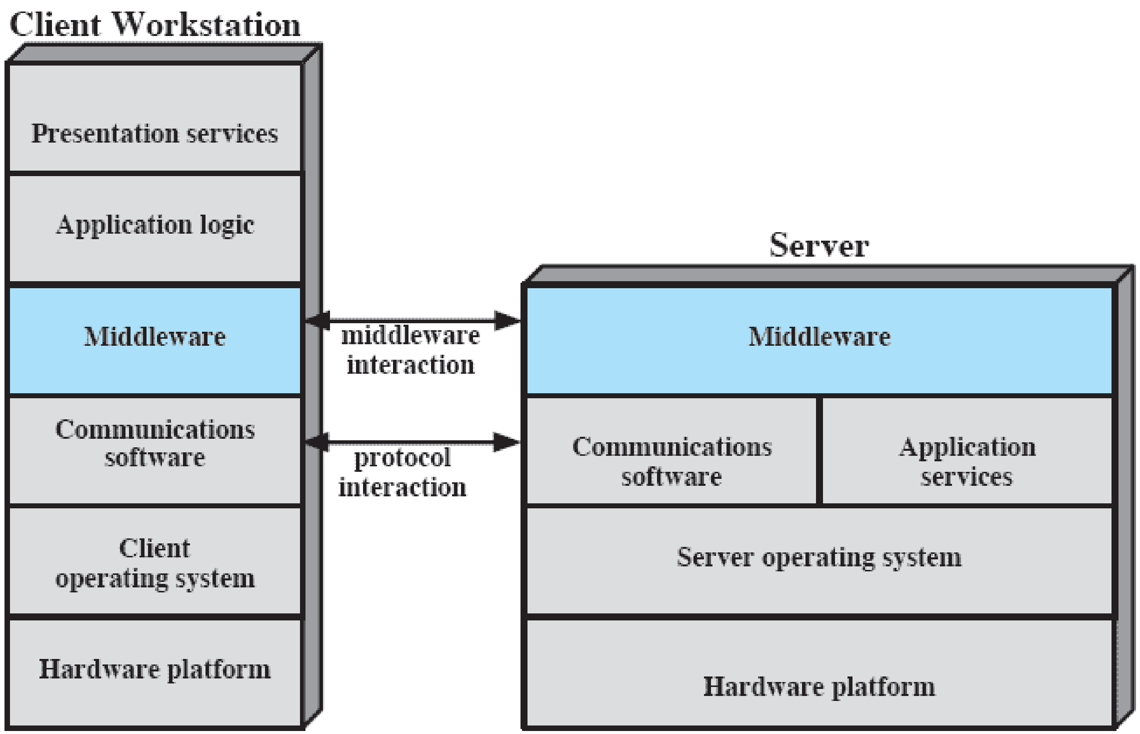


Figure 16.8 The Role of Middleware in Client/Server Architecture

Classes of Client/Server Applications

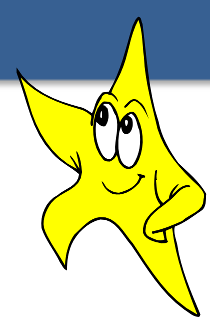
Four general classes are:

Host-based processing

Server-based processing

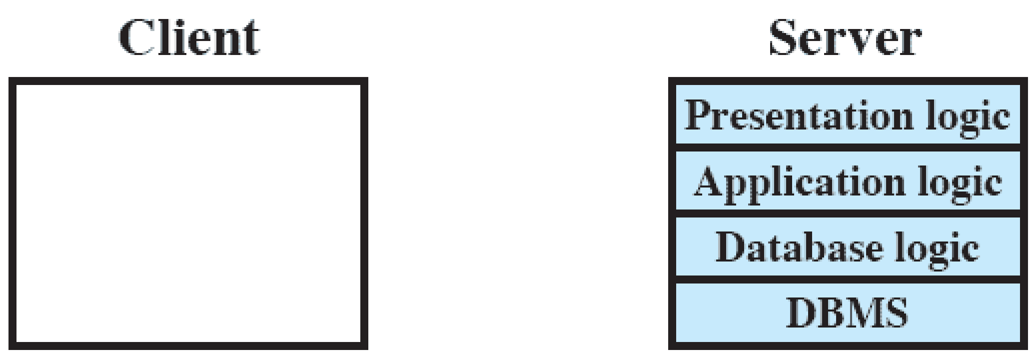
Cooperative processing

Client-based processing



Classes of Client/Server Applications

- Host-based processing
  - Not true client/server computing
  - Traditional mainframe environment



(a) Host-based processing

Classes of Client/Server Applications

- Server-based processing
  - Server does all the processing
  - Client provides a graphical user interface

(b) Server-based processing

Classes of Client/Server Applications

- Client-based processing
  - All application processing done at the client
  - Data validation routines and other database logic functions are done at the server

(d) Client-based processing

Classes of Client/Server Applications

- Cooperative processing
  - Application processing is performed in an optimized fashion
  - Complex to set up and maintain
  - Fat vs Thin clients

(c) Cooperative processing

Three-tier Client/Server Architecture

- Application software distributed among three types of machines
  - User machine
    - Thin client
  - Middle-tier server
    - Gateway
    - Convert protocols
    - Merge/integrate results from different data sources
  - Backend server

Three-tier Client/Server Architecture

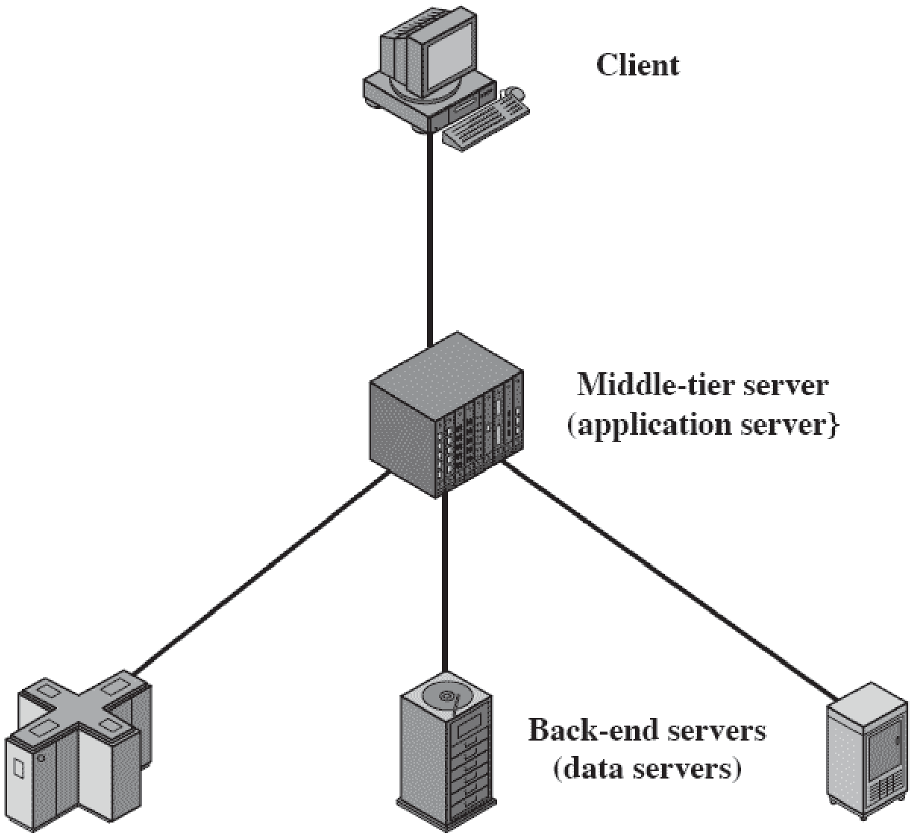
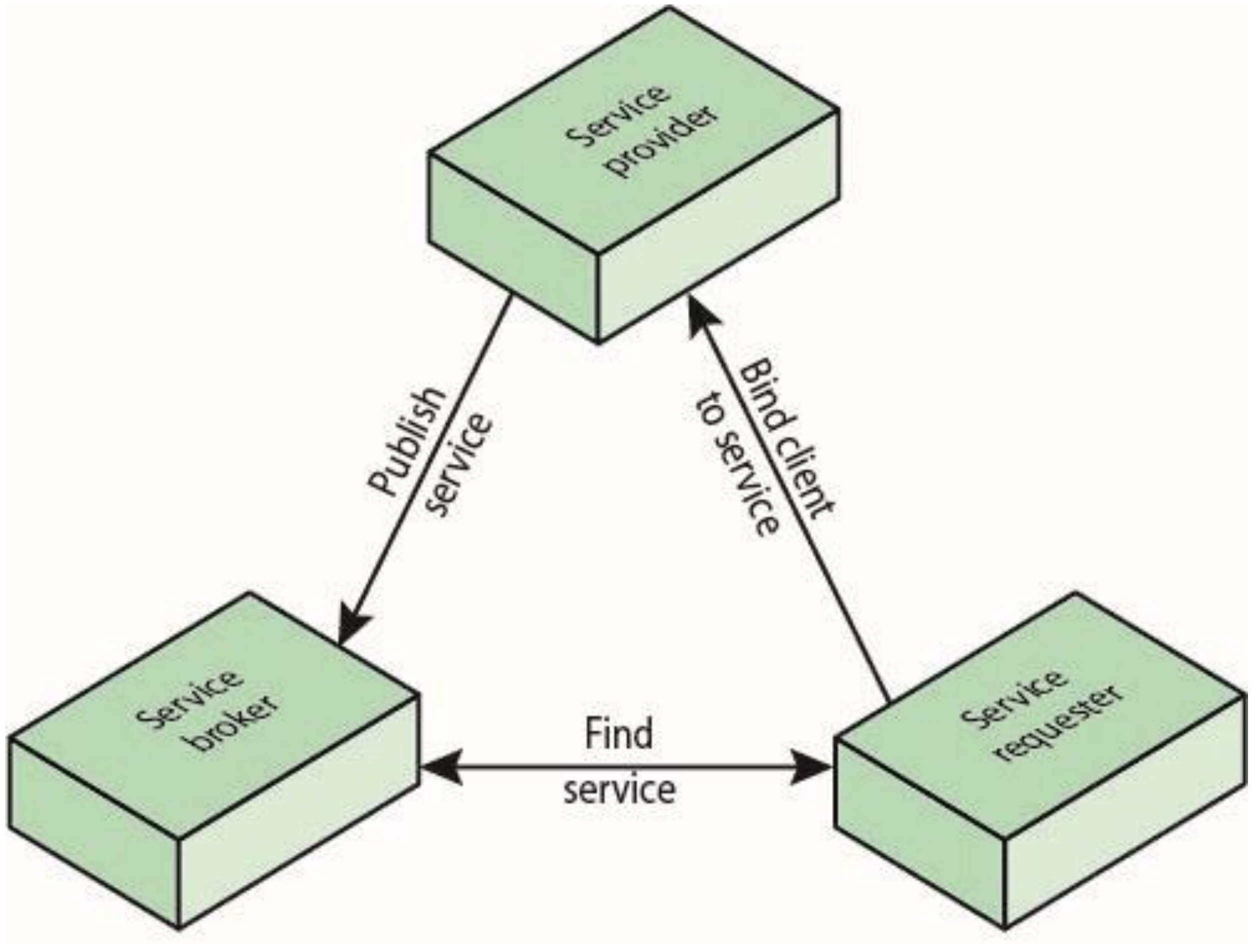


Figure 16.6 Three-tier Client/Server Architecture

Service-Oriented Architecture (SOA)

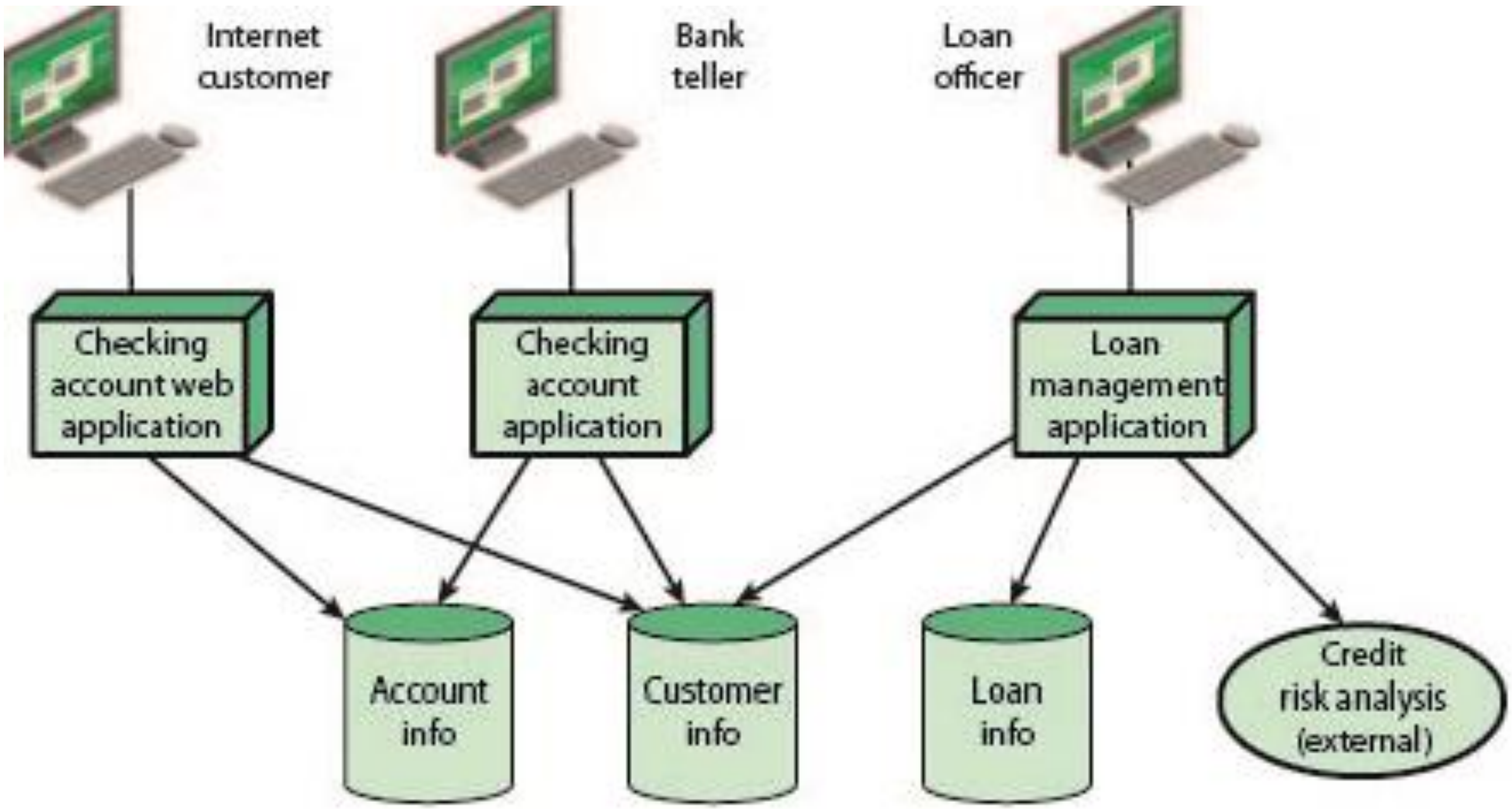
- A form of client/server architecture used in enterprise systems
- Organizes business functions into a modular structure rather than as monolithic applications for each department
  - as a result, common functions can be used by different departments internally and by external business partners as well
- Consists of a set of services and a set of client applications that use these services
- Standardized interfaces are used to enable service modules to communicate with one another and to enable client applications to communicate with service modules
  - most popular interface is XML (Extensible Markup Language) over HTTP (Hypertext Transfer Protocol), known as ***Web services***

SOA Model



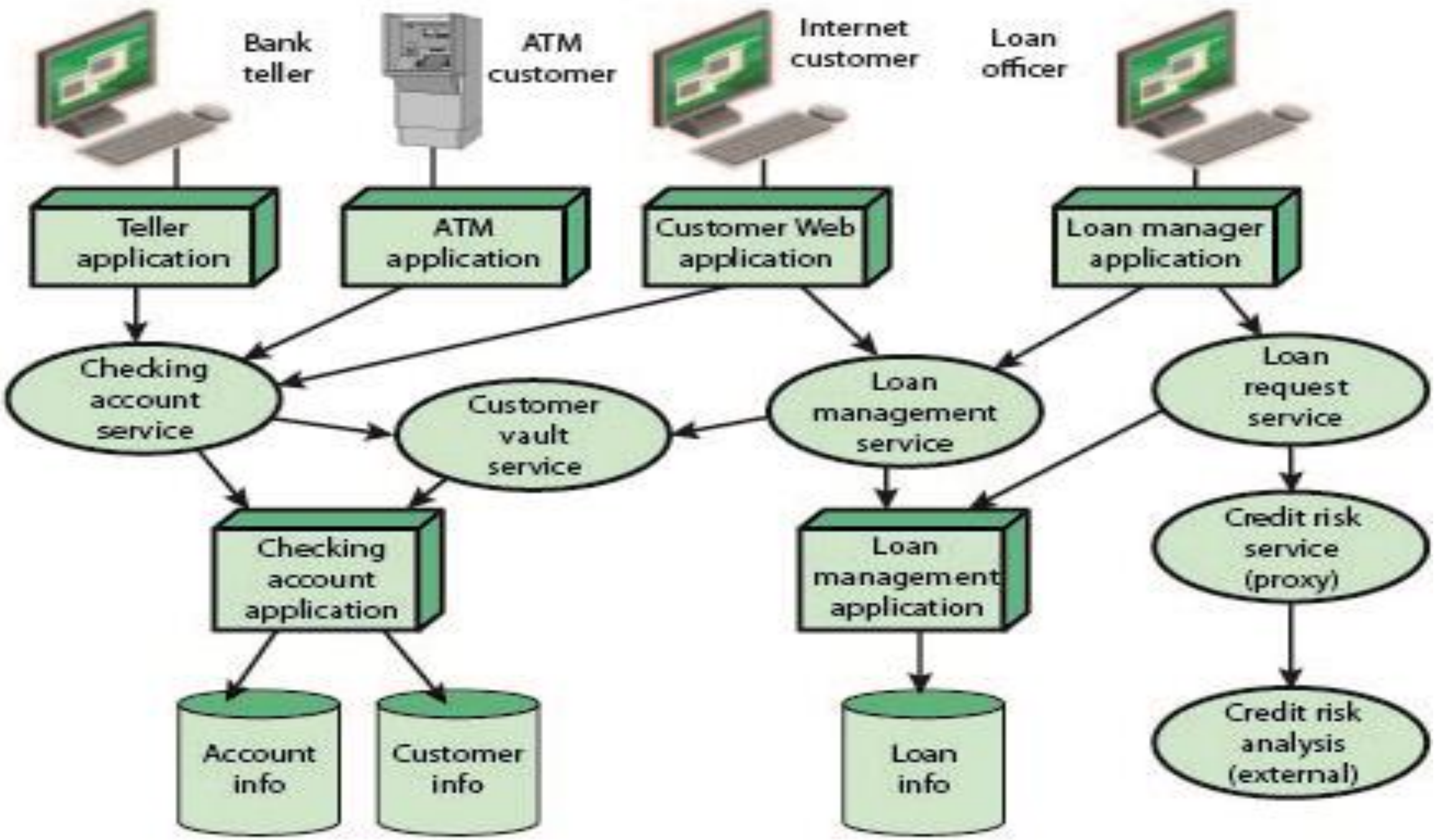
Service provider
Publish service
Bind client to service
Service broker
Find service
Service requester

Example Use of SOA



(a) Typical application structure

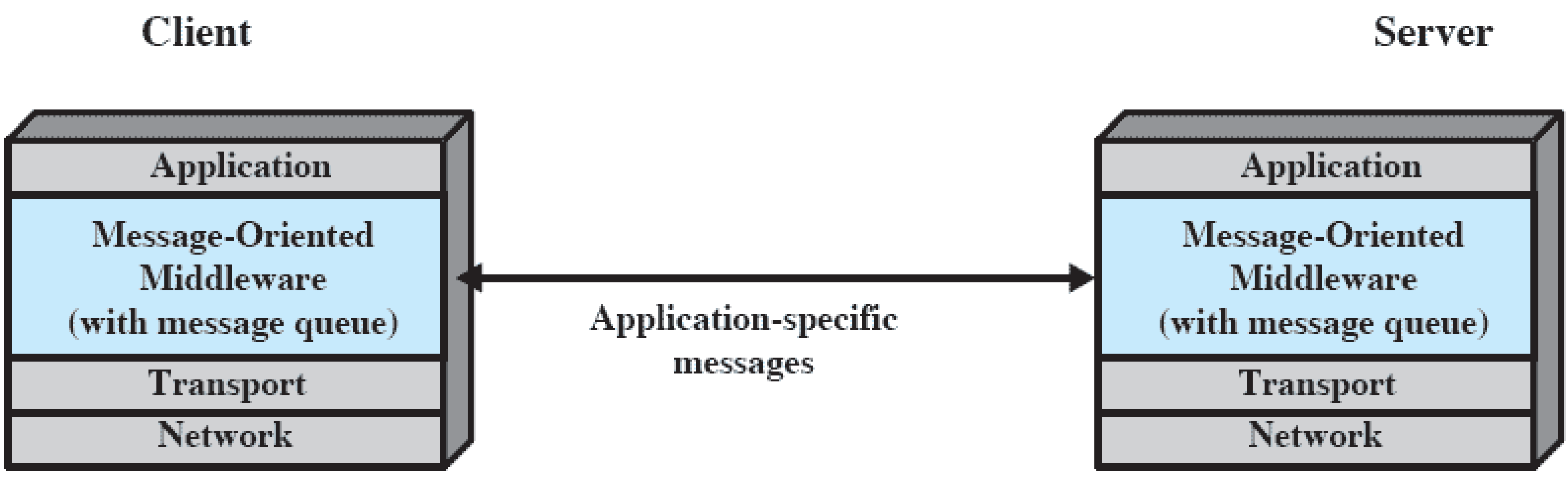
Example Use of SOA



(b) An architecture reflecting SOA principles

MESSAGE PASSING

Distributed Message Passing



(a) Message-Oriented Middleware

Basic Message-Passing Primitives

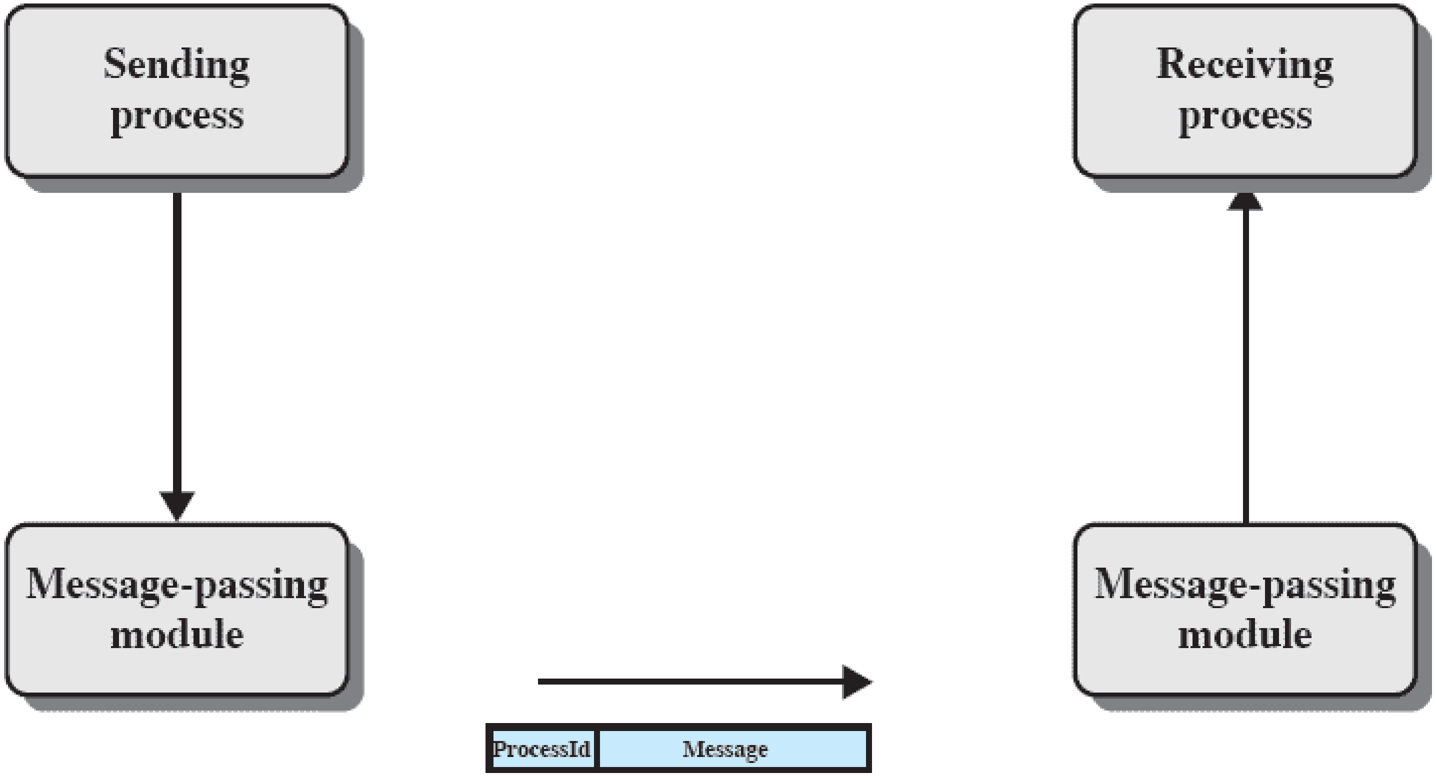


Figure 16.13 Basic Message-Passing Primitives

REMOTE PROCEDURE CALLS

Remote Procedure Calls

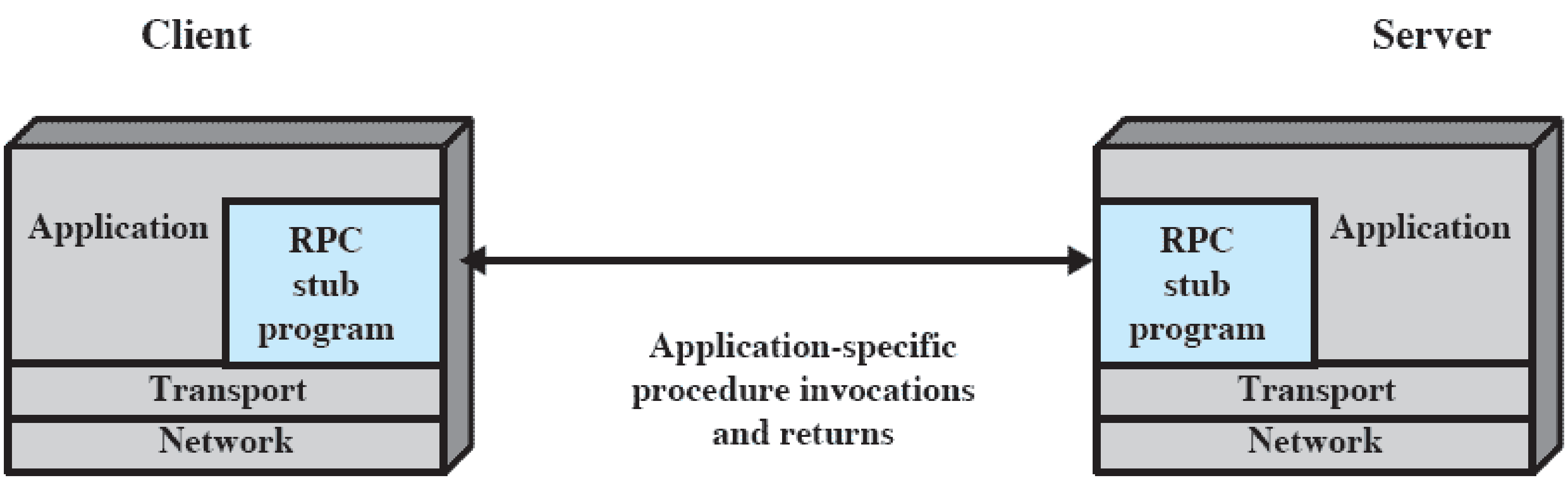
- Allow programs on different machines to interact using simple procedure call/return semantics
- Used for access to remote services
- Widely accepted and common method for encapsulating communication in a distributed system

RPC is standard technology

Advantages of standardization

- the communication code for an application can be generated automatically
- client and server modules can be moved among computers and OSs with little modification and recoding

Remote Procedure Call Architecture



(b) Remote Procedure Calls

Remote Procedure Call Mechanism

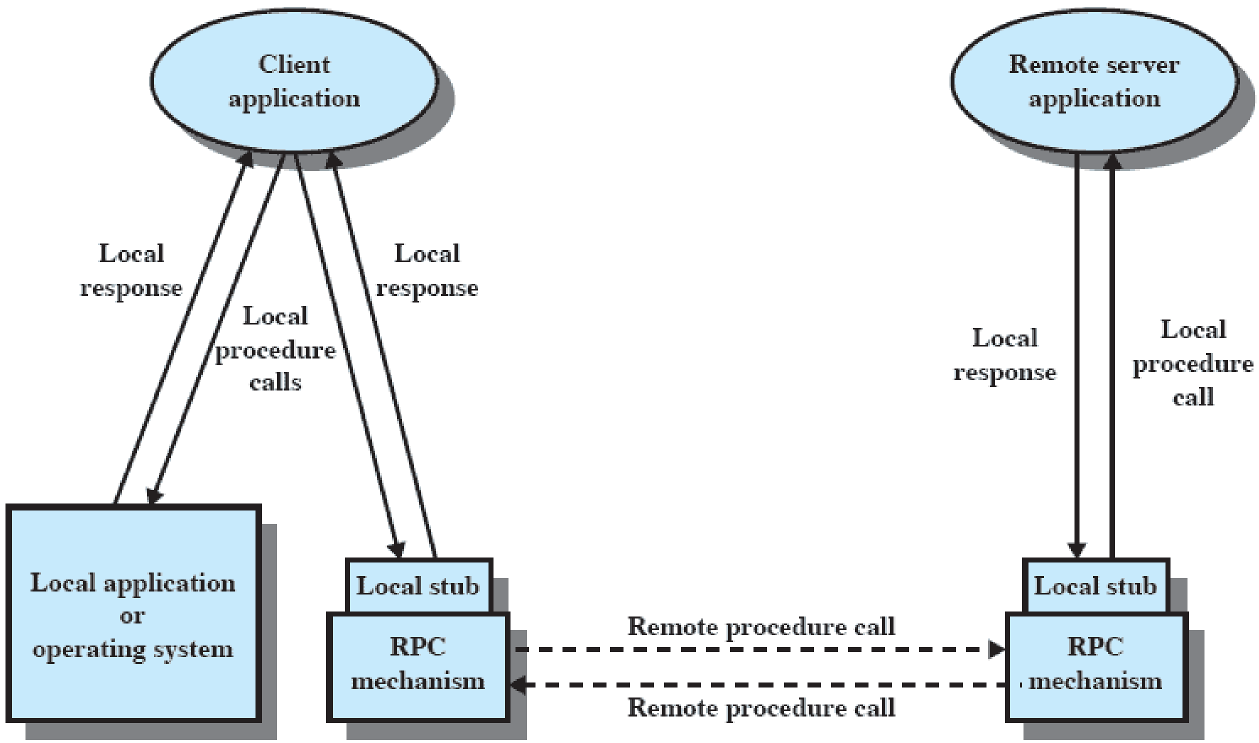


Figure 16.14 Remote Procedure Call Mechanism

Parameter Passing/ Parameter Representation

- Passing a parameter by ***value*** is easy
- Passing by ***reference*** is more difficult
  - a unique system wide pointer is necessary
  - the overhead for this capability may not be worth the effort
- The representation/format of the parameter and message may be difficult if the programming languages differ between client and server

Client/Server Binding

A binding is formed when two applications have made a logical connection and are prepared to exchange commands and data

Nonpersistent Binding

- Nonpersistent binding means that a logical connection is established between the two processes at the time of the remote procedure call and that as soon as the values are returned, the connection is dismantled
- The overhead involved in establishing connections makes nonpersistent binding inappropriate for remote procedures that are called frequently by the same caller

Client/Server Binding

Persistent Binding

- A connection that is set up for a remote procedure call is sustained after the procedure return
- The connection can then be used for future remote procedure calls
- If a specified period of time passes with no activity on the connection, then the connection is terminated
- For applications that make many repeated calls to remote procedures, persistent binding maintains the logical connection and allows a sequence of calls and returns to use the same connection

Synchronous versus Asynchronous

Synchronous RPC

- behaves much like a subroutine call
- behavior is predictable
- however, it fails to exploit fully the parallelism inherent in distributed applications
- this limits the kind of interaction the distributed application can have, resulting in lower performance

Synchronous versus Asynchronous

Asynchronous RPC

- does not block the caller
- replies can be received as and when they are needed
- allow client execution to proceed locally in parallel with server invocation

RPC Call Semantics

Normal RPC functioning may get disrupted

- call or response message is lost
- caller node crashes and is restarted
- callee node crashes and is restarted

The call semantics determines how often the remote procedure may be executed under fault conditions

Credits: http://www.ques10.com/p/2159/write-a-short-note-on-call-semantics-of-rpc-1/
Next few slides: http://www.cs.unc.edu/~dewan/242/f97/notes/ipc/node27.html

At least once

- This semantics guarantees that the call is executed one or more times but does not specify which results are returned to the caller.
- Very little overhead & easy to implement
  - using timeout based retransmission without considering orphan calls (i.e., calls on server machines that have crashed)
  - the client machine continues to send call requests to the server machine until it gets an acknowledgement. If one or more acknowledgements are lost, the server may execute the call multiple times
- Works only for idempotent operations

At most once

- This semantics guarantees that the RPC call is executed at most once
  - either it does not execute at all or it executes exactly once depending on whether the server machine goes down
- Unlike the previous semantics, this semantics require the detection of duplicate packets, but works for non-idempotent operations.

Exactly once

- The RPC system guarantees the *local* call semantics assuming that a server machine that crashes will eventually restart.
- It keeps track of orphan calls and allows them to later be adopted by a new server
- Requires a very complex implementation!

CLUSTERS

Clusters

- Alternative to symmetric multiprocessing (SMP) as an approach to providing high performance and high availability
- Group of interconnected, whole computers working together as a unified computing resource that can create the illusion of being one machine
- ***Whole computer*** means a system that can run on its own, apart from the cluster
- Each computer in a cluster is called a ***node***

Benefits of Clusters

Absolute scalability

it is possible to create large clusters that far surpass the power of even the largest stand-alone machines

Incremental scalability

configured in such a way that it is possible to add new systems to the cluster in small increments

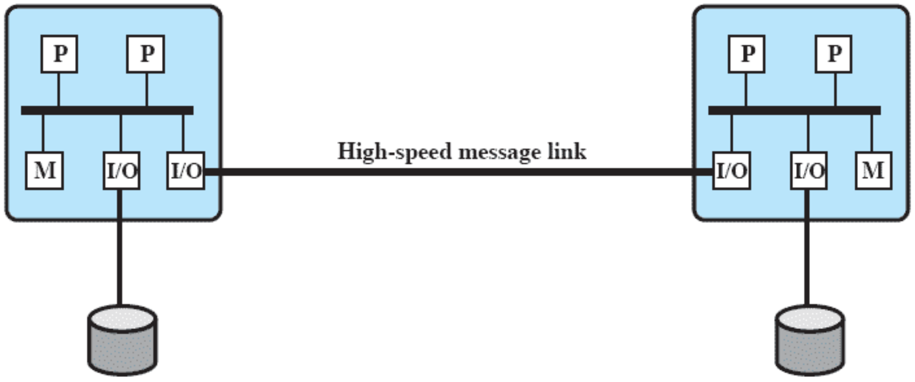
High availability

failure of one node is not critical to system

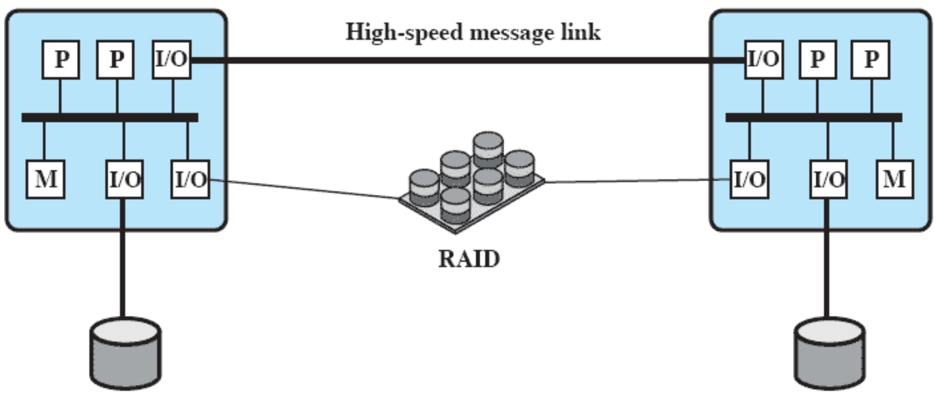
Superior price/performance

by using commodity building blocks, it is possible to put together a cluster at a much lower cost than a single large machine

Cluster Configurations



(a) Standby server with no shared disk



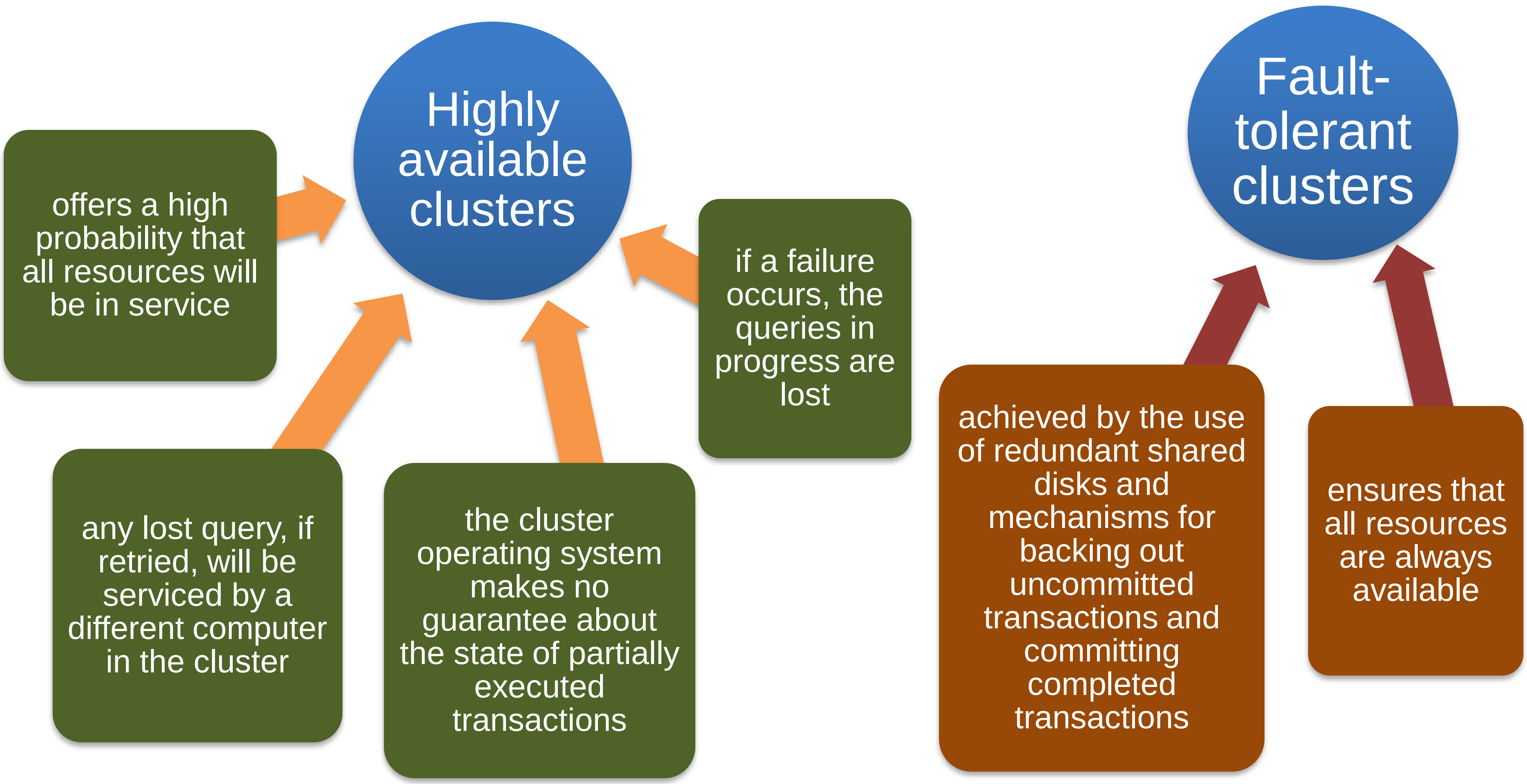
(b) Shared disk

Clustering Methods: Benefits and Limitations

Clustering Method	Description	Benefits	Limitations
Passive Standby	A secondary server takes over in case of primary server failure.	Easy to implement.	High cost because the secondary server is unavailable for other processing tasks.
Active Secondary	The secondary server is also used for processing tasks.	Reduced cost because secondary servers can be used for processing.	Increased complexity.
Separate Servers	Separate servers have their own disks. Data is continuously copied from primary to secondary server.	High availability.	High network and server overhead due to copying operations.
Servers Connected to Disks	Servers are cabled to the same disks, but each server owns its disks. If one server fails, its disks are taken over by the other server.	Reduced network and server overhead due to elimination of copying operations.	Usually requires disk mirroring or RAID technology to compensate for risk of disk failure.
Servers Share Disks	Multiple servers simultaneously share access to disks.	Low network and server overhead. Reduced risk of downtime caused by disk failure.	Requires lock manager software. Usually used with disk mirroring or RAID technology.

OS Design Issues: Failure Management

Two approaches can be taken to deal with failures:



OS Design Issues: Failure Management

- The function of switching an application and data resources over from a failed system to an alternative system in the cluster is referred to as ***fallover***
- The restoration of applications and data resources to the original system once it has been fixed is referred to as ***fallback***
- Fallback can be automated but this is desirable only if the problem is truly fixed and unlikely to recur
- Automatic failback can cause subsequently failed resources to bounce back and forth between computers, resulting in performance and recovery problems

Load Balancing

- A cluster requires an effective capability for balancing the load among available computers
- This includes the requirement that the cluster be incrementally scalable
- When a new computer is added to the cluster, the load-balancing facility should automatically include this computer in scheduling applications
- Middleware must recognize that services can appear on different members of the cluster and may migrate from one member to another

Parallelizing Computation

Parallelizing compiler

- determines, at compile time, which parts of an application can be executed in parallel
- performance depends on the nature of the problem and how well the compiler is designed

Parallelized application

- the programmer designs the application to run on a cluster and uses message passing to move data, as required, between cluster nodes
- this places a high burden on the programmer but may be the best approach for exploiting clusters for some applications

Parametric computing

- this approach can be used if an application is an algorithm or program that must be executed a large number of times, each time with a different set of starting conditions or parameters
- for this approach to be effective, parametric processing tools are needed to organize, run, and manage the jobs in an orderly manner

Cluster Computer Architecture

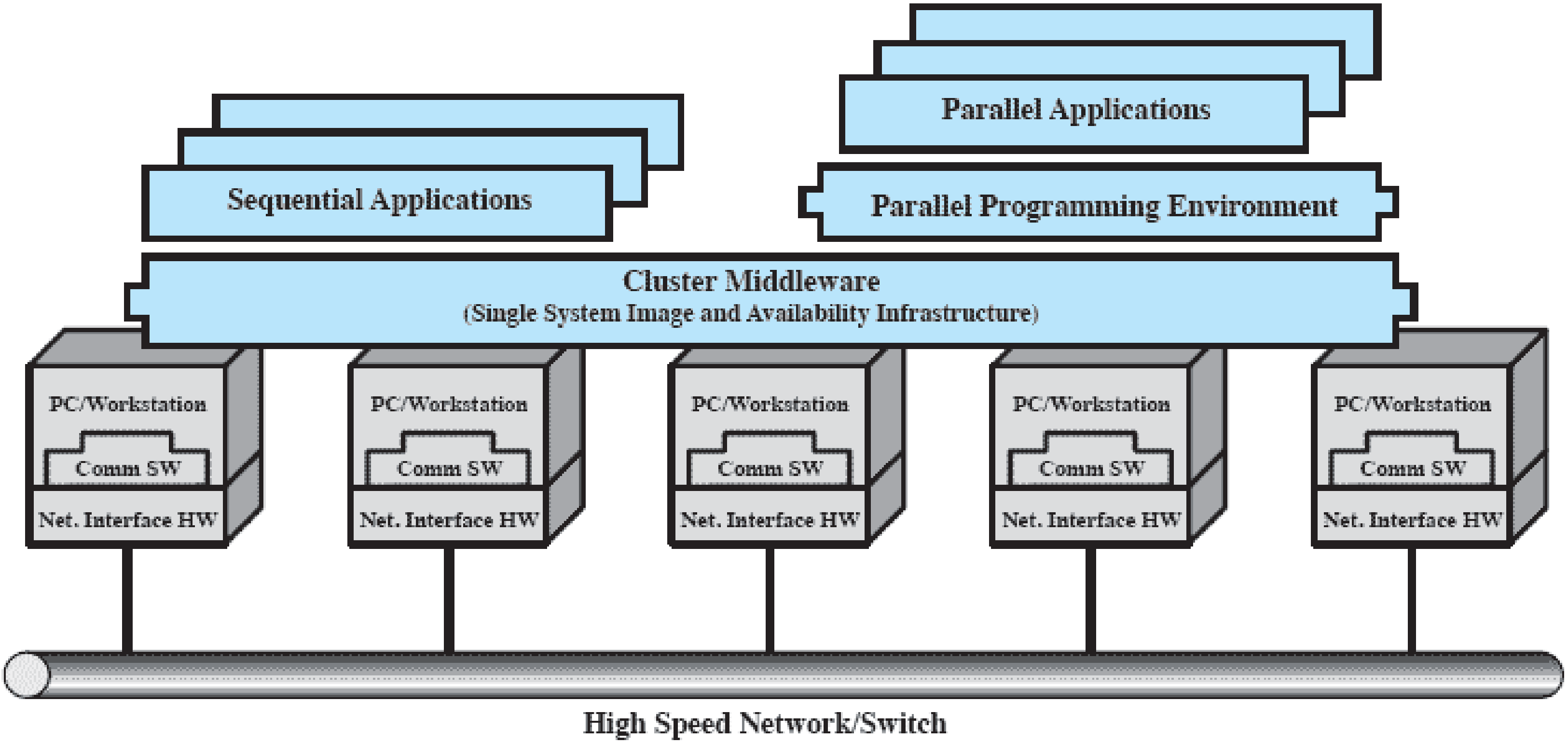


Figure 16.14 Cluster Computer Architecture [BUYY99a]