Adapted from:Operating Systems:
Internals and Design Principles, 6/E
William Stallings

CS571
Fall 2010

Client/Server and Distributed Computing

Dave Bremer
Otago Polytechnic, N.Z.

Traditional Data Processing

- Traditionally, data processing was **centralised** (much still is!)
- Typically involving centralised
  - Computers
  - Processing and management
  - Data
- When did this change?
    - …and change again? …and again?

Distributed Data Processing

- Distributed Data Processing (DDP) departs from the centralised model in multiple ways.
- Usually smaller computers, dispersed throughout an organization
  - is this “better”? why or why not?
- May involve many **central node(s)** with satellites, or be a dispersed **peer to peer architecture**
  - Interconnection(s) required
    - Private and “public”
      - Dedicated, intranet, internet

Advantages of DDP (?)

- Responsiveness
- Availability
- Resource Sharing
- Incremental growth
- Increased user involvement and control
- End-user productivity
- Any disadvantages?

Client/Server Computing

- **Client** machines are generally single-user workstations providing a user-friendly (?) interface to the end user
- Each **server** provides a set of shared services to the clients
  - – enables many clients to share access to the same database
  - – enables the use of a high-performance computer system to manage a database

Generic Client/Server Environment

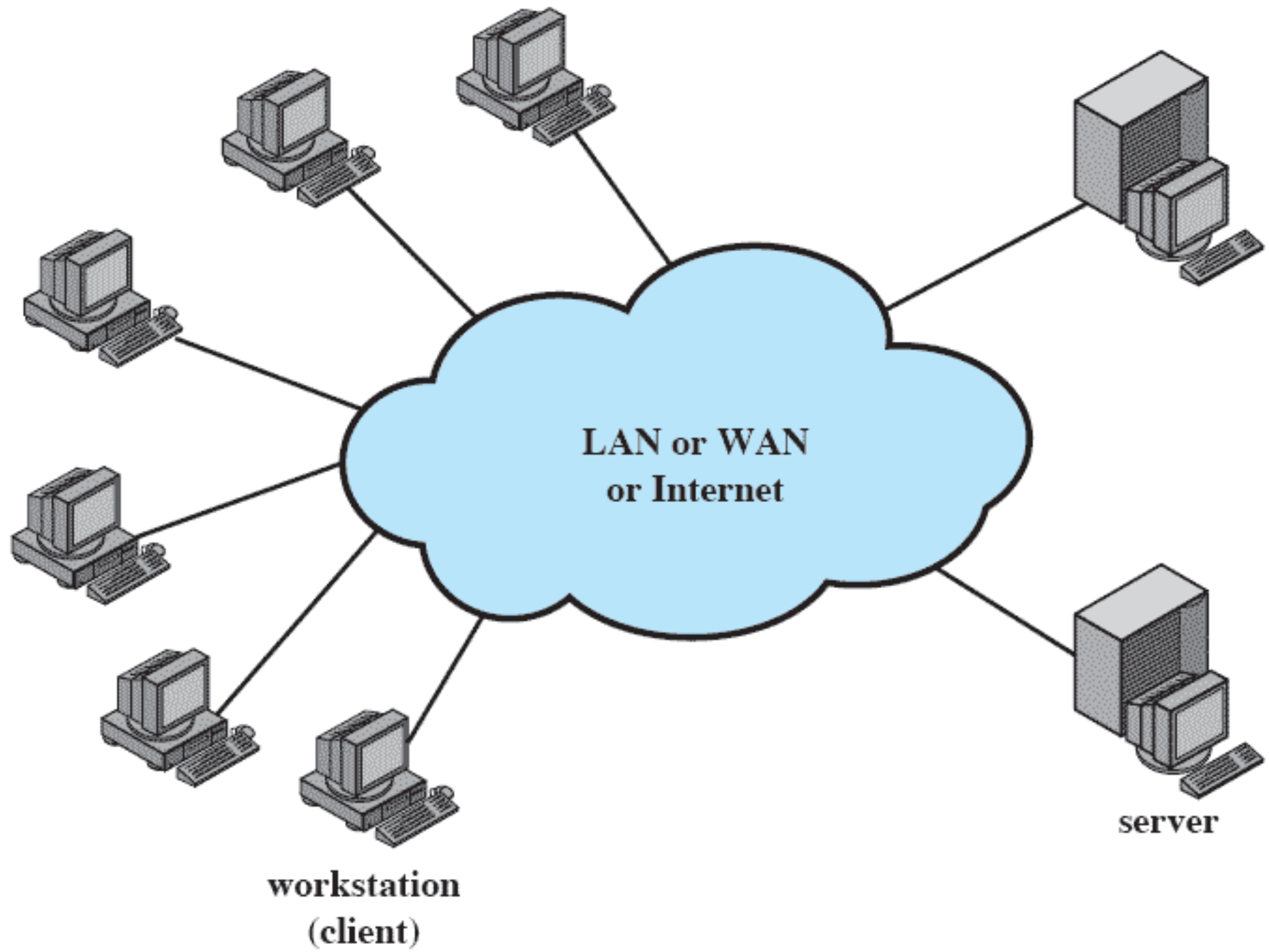


Figure 16.1 Generic Client/Server Environment

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

Generic Client/Server Architecture

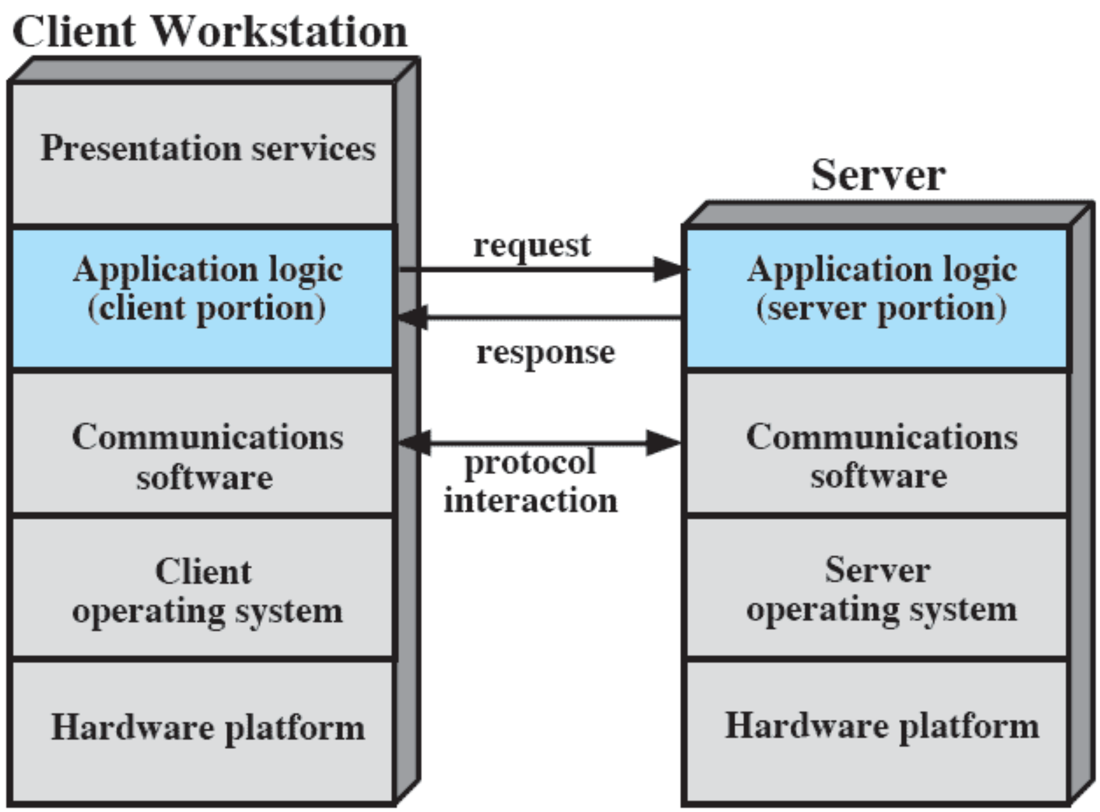


Figure 16.2 Generic Client/Server Architecture

Client/Server Applications

- Bulk of applications software executes on the server
- Application **logic** may be located at the client and/or the server
- Presentation services almost always in the client
- Recall the **MVC design pattern**
- **This is the Web model**

Database Applications

- The server(s) can be database server(s)
  - Most common family of client/server applications
- Interaction is in the form of **transactions**
  - the client makes a database **request** and receives a database **response** from server(s)
- Server(s) responsible for maintaining the database **state (NOT THE CLIENT!!)**

Architecture for Database Applications

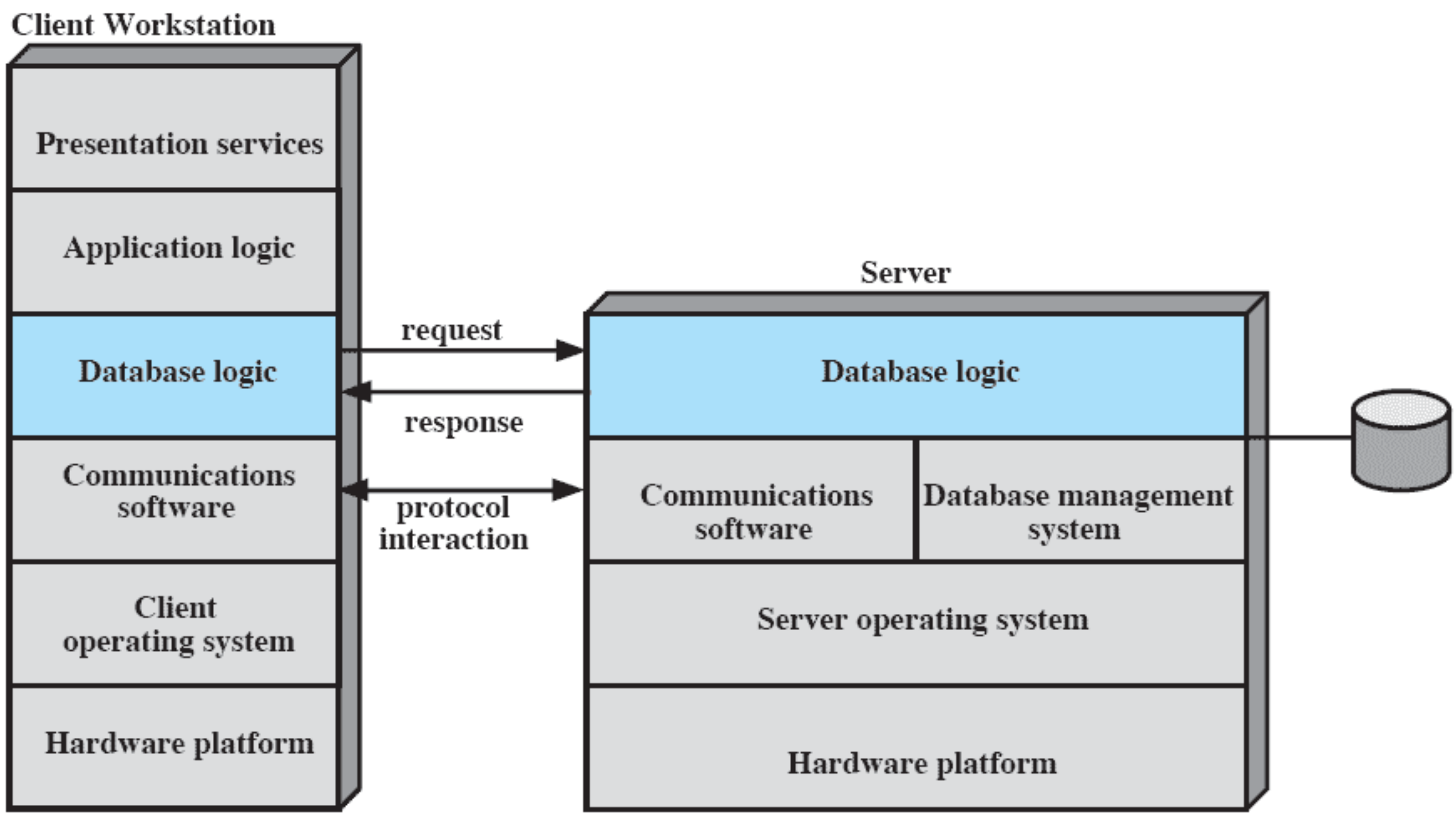


Figure 16.3 Client/Server Architecture for Database Applications

Three-tier (Web) Client/Server Architecture

- Application software distributed among three types of machines
  - User machine (VIEW)
    - Thin client, browser
  - Middle-tier server (CONTROLLER)
    - Gateway
    - Convert protocols
    - Merge/integrate results from different data sources
  - Data server (MODEL)

Three-tier Client/Server Architecture

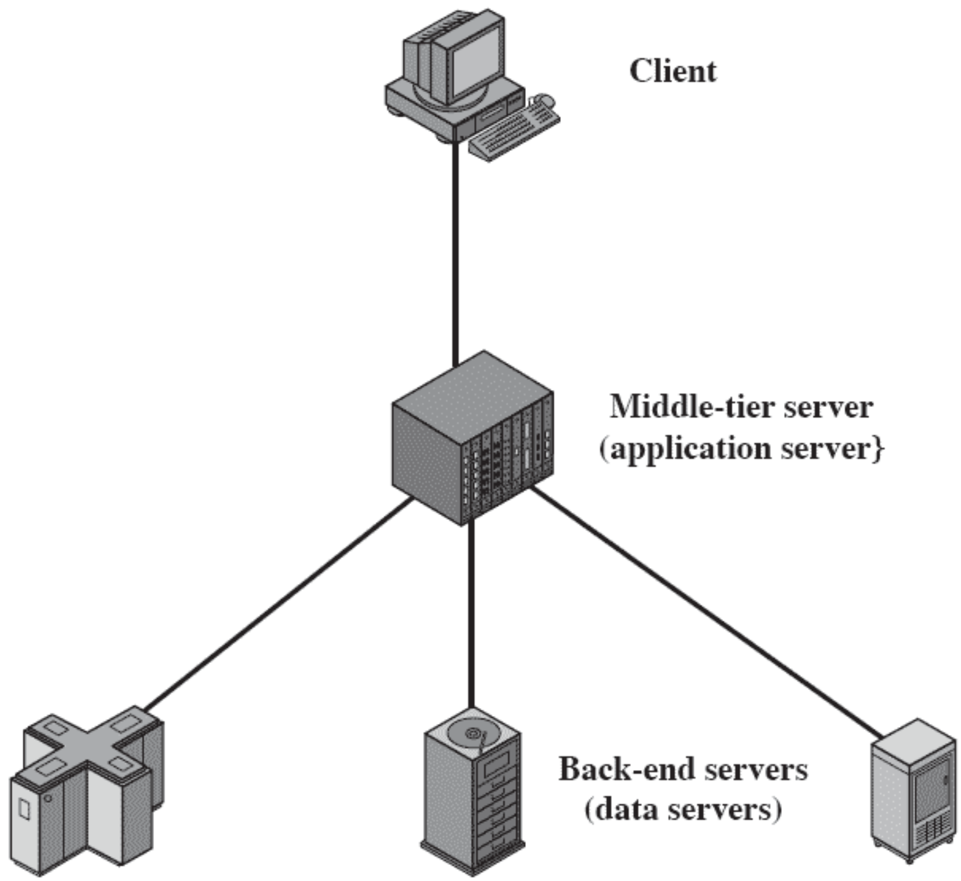


Figure 16.6 Three-tier Client/Server Architecture

File **Cache** Consistency !

- File **caches** hold recently accessed file records
- Caches are consistent when they contain **exact copies** of remote data
- **File-locking** prevents simultaneous access to a file
  - However…
    - remember cache consistency problem!
      - and its performance impact

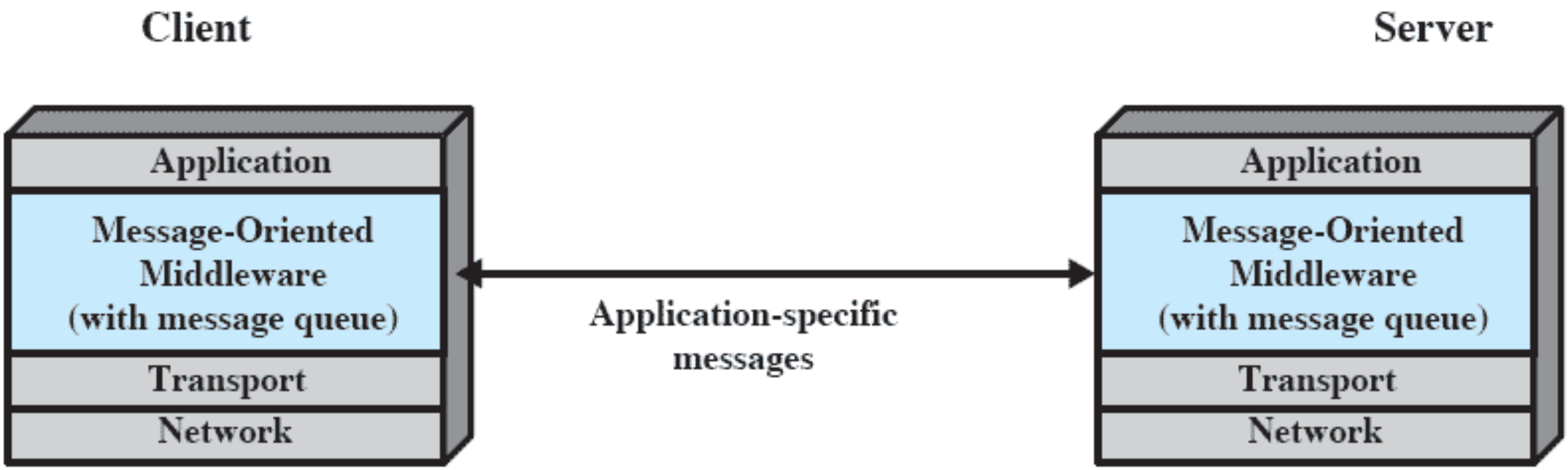
Is Caching **Scalable**?

- As # of systems, users, processes grows, file & cache locking become bottlenecks
- Brewer's **CAP Theorem**:
  - **C**onsistency, **A**vailability, **P**artitionability…
  - …choose any TWO, **can't do the third !!**…
  - …leads to idea of ***eventual consistency***
    - Given a "sufficiently long period of time", over which no updates are sent, we expect that during this period, all updates will, **eventually**, propagate through the system and all the replicas will be consistent
    - In database terminology, this is known as **BASE** (**B**asically **A**vailable, **S**oft state, **E**ventual consistency), as opposed to the database concept of ACID (*__A__tomicity, __C__onsistency, __I__solation, __D__urability)*

Interprocess Communication (IPC)

- *Usually* computers involved in DDP do not share a main memory
  – They are *isolated* computers
  – But some database and other applications do use shared memory services
- IPC relies on *message passing*

Distributed Message Passing



(a) Message-Oriented Middleware

Basic Message-Passing Primitives

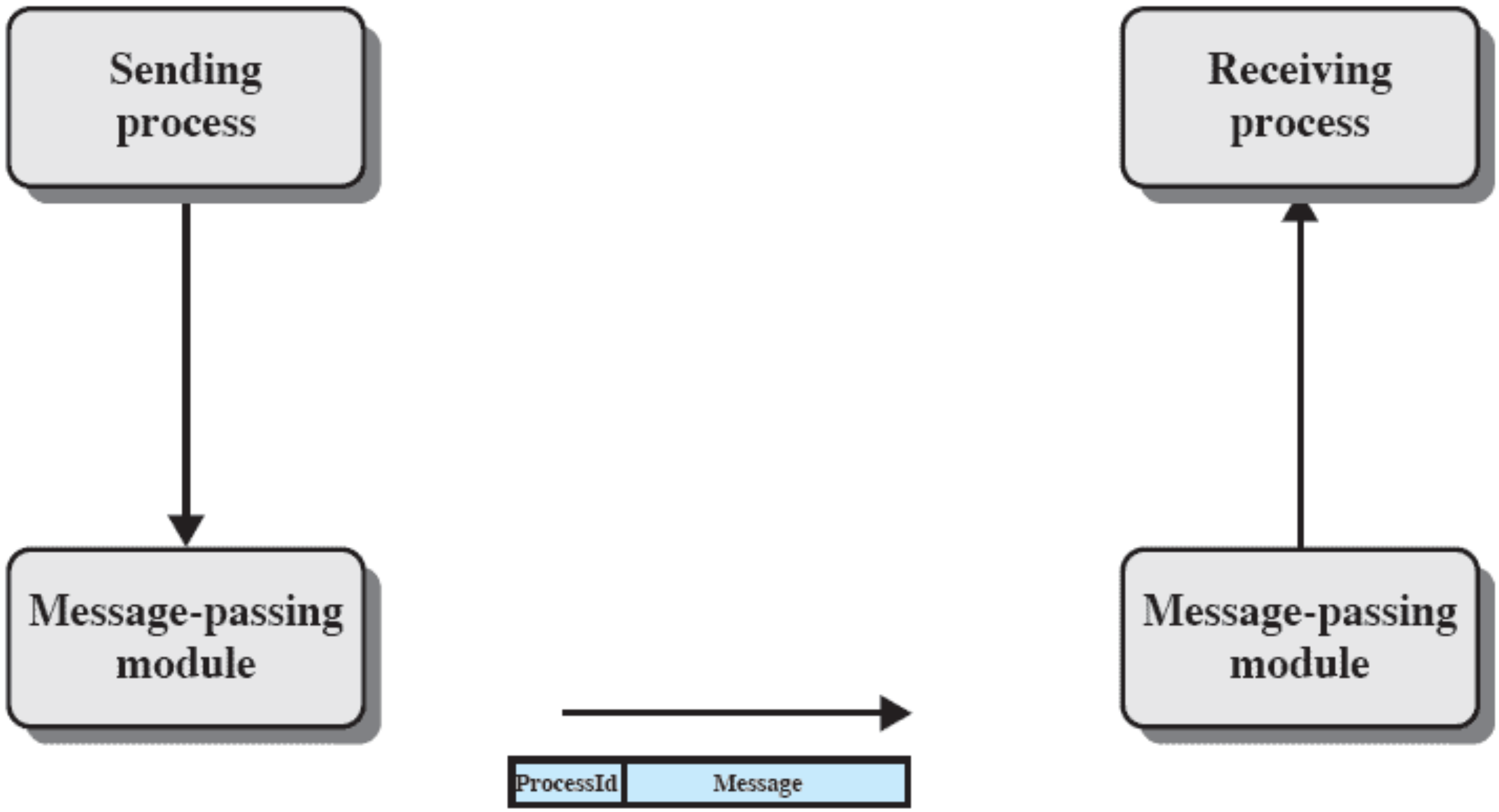


Figure 16.11 Basic Message-Passing Primitives

Reliability vs.. Unreliability

- **Reliable** message-passing *guarantees delivery* if possible
  - But **acknowledgement** is a *performance issue*
- "**Unreliable**": Send the message out into the communication network *without reporting success or failure*
  - Reduces complexity and overhead
  - Like the **UDP** protocol

Blocking vs.. Nonblocking

- Nonblocking
  - Process is **not suspended** as a result of issuing a Send or Receive
  - Efficient and flexible
  - **Difficult to debug!**

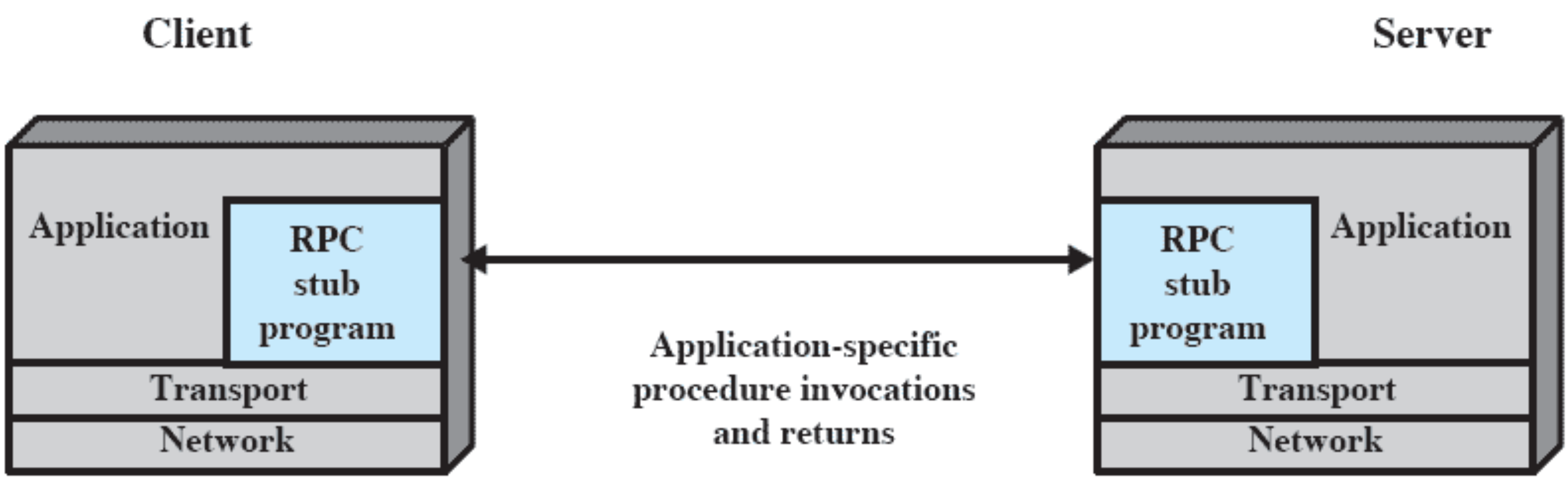
Blocking vs.. Nonblocking

- Blocking
  - Send does not return control to the sending process until the message has been transmitted
  - OR does not return control until an acknowledgment is received
  - Receive does not return until a message has been placed in the allocated buffer
- Blocking and NonBlocking protocols used many places in computer architectures

Remote Procedure Calls

- Allow programs on different machines to interact using simple procedure call/return semantics
- Widely accepted
- Standardized
  - Client and server modules can be moved among computers and operating systems easily

RPC Architecture



(b) Remote Procedure Calls

Remote Procedure Call Mechanism

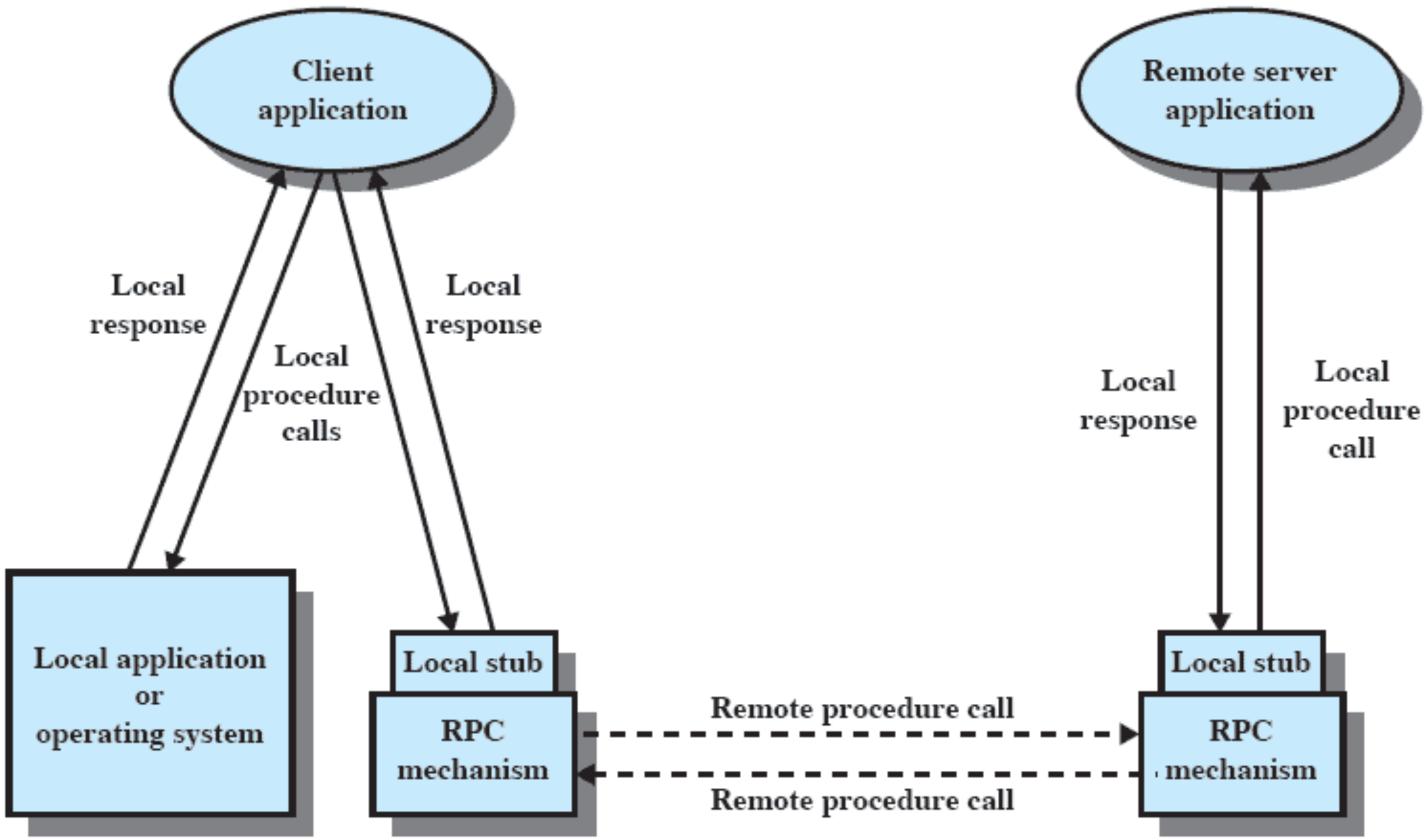


Figure 16.12 Remote Procedure Call Mechanism

Synchronous versus Asynchronous

- Synchronous RPC
  - Behaves much like a subroutine call
- Asynchronous RPC
  - **Does not block** the caller
  - Enable a client execution to proceed locally **in parallel** with server invocation

Clusters

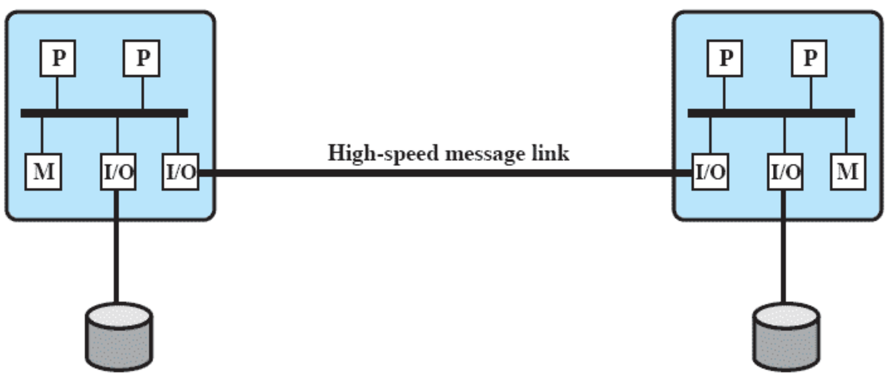
- Alternative to symmetric multiprocessing (SMP)
- Group of interconnected, whole computers working together as a unified computing resource
  - Illusion of one machine
  - Each system can run on its own
- Digital's early VAX/VMS Cluster is archetype
  - took many years for UNIX/Linux to catch up

Benefits of Clusters

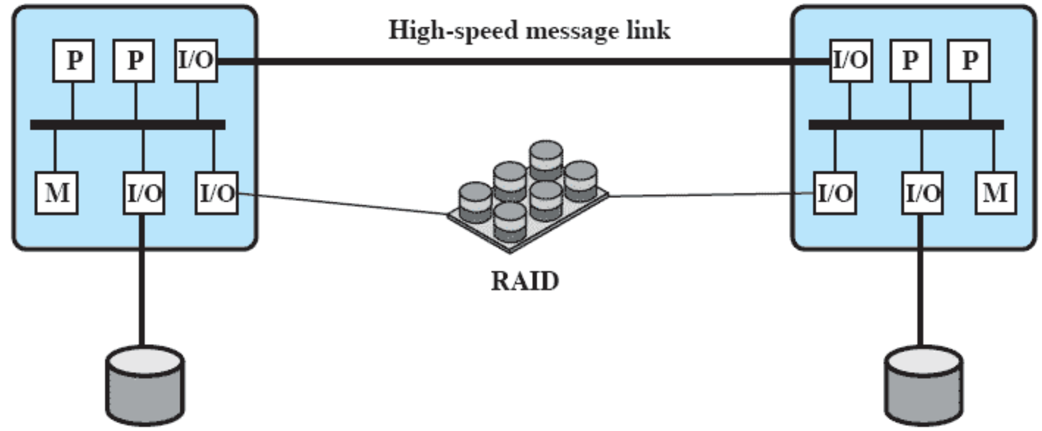
- Absolute Scalability (?)
  - Larger than any single device is possible
- Incremental scalability
  - System can grow by adding new nodes
- High availability
  - Failure of one node is not critical to system
- Superior price/performance
  - Using commodity equipment

Cluster Classification

- Numerous approaches to classification.
  - Simplest is based on shared disk access



(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.

Clustering Methods: Benefits and Limitations

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.

Beowulf and Linux Clusters

- Initiated in 1994 by NASA's High Performance Computing and Communications project
- To investigate the potential for clustered PC's to perform computational tasks beyond the capacity of typical workstations at minimal cost
- The project was a success!

Beowulf and Linux Clusters

- Key features
  - Mass market commodity components
  - Dedicated processors (rather than scavenging cycles from idle workstations)
  - A dedicated, private network (LAN or WAN or internetted combination)
  - No custom components
  - Easy replication from multiple vendors

Beowulf Features

- Dedicated processors and network
- Scalable I/O (**Lustre** file system)
- A freely available software base
  - Beowulf, Sun Grid Engine, IBM Globus, …
- Use freely available distribution computing tools with minimal changes
- Open Source (Community Developed):
  - Return of the design and improvements to the community

Generic Beowulf Configuration

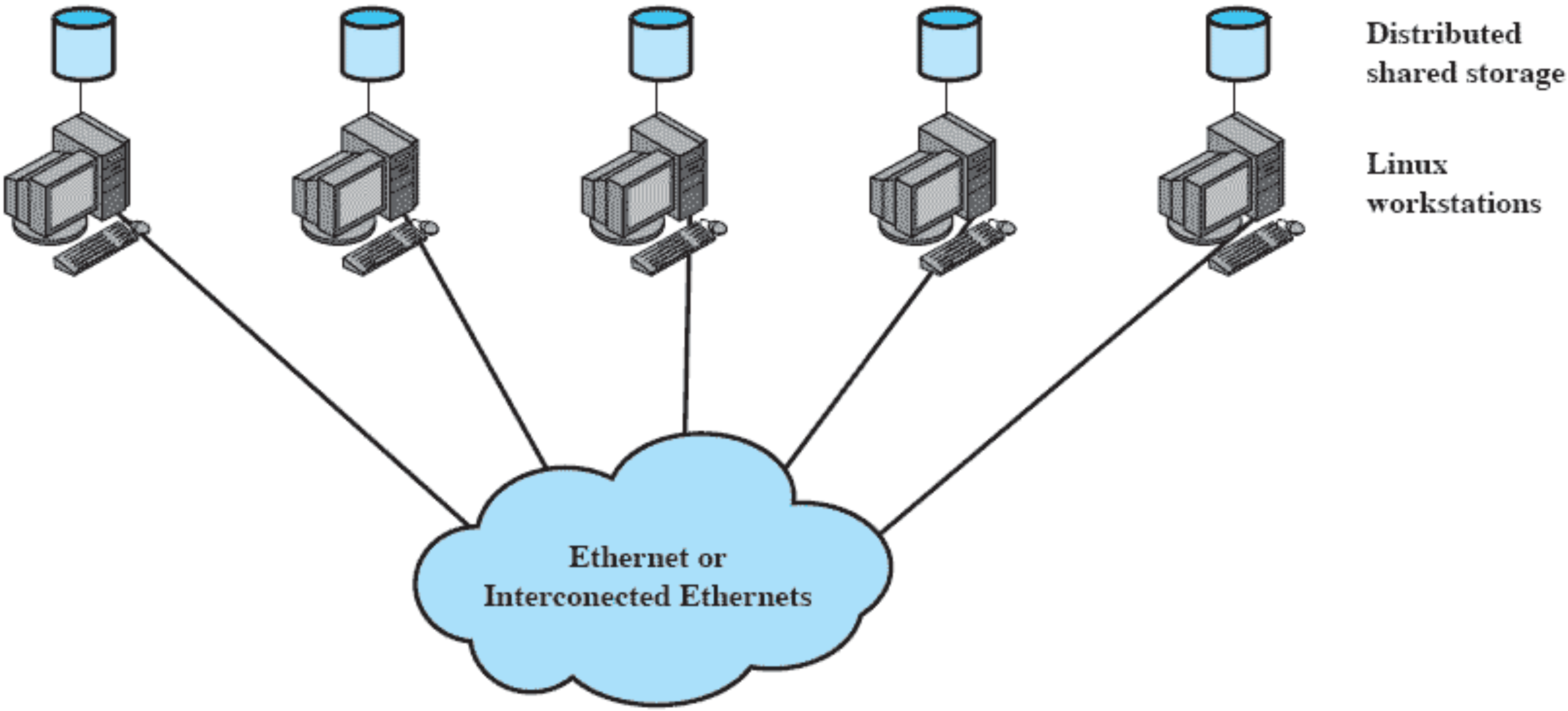
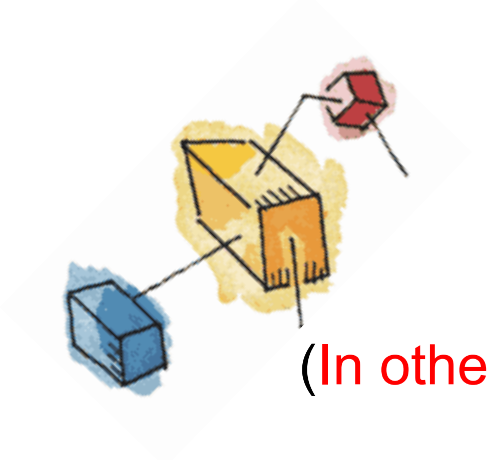


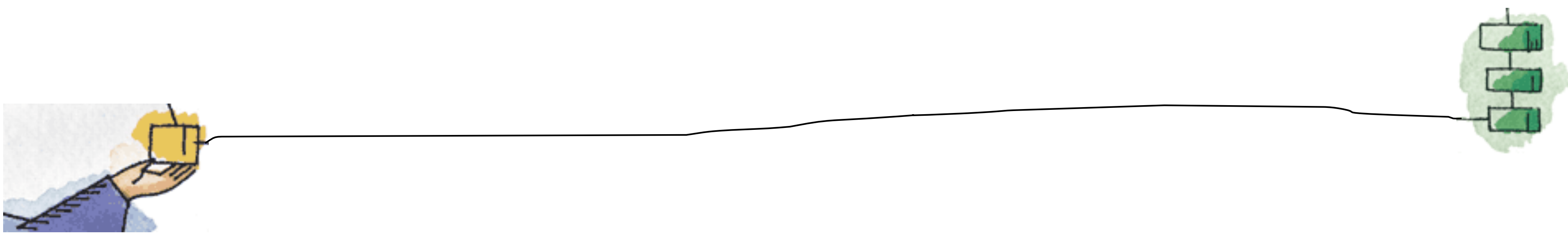
Figure 16.18 Generic Beowulf Configuration

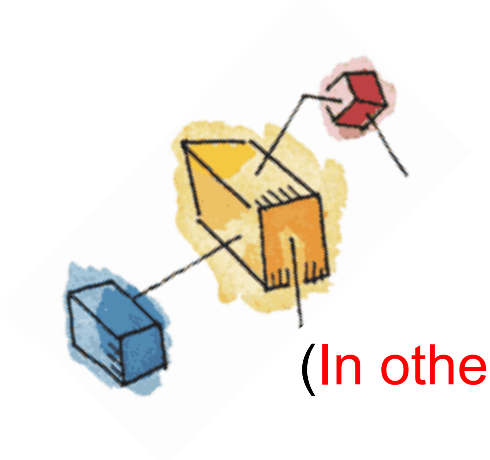


The **Fallacies** of Distributed Computing

(In other words, **don’t make these mistaken assumptions!**)

- The Network is **Reliable**
- **Latency** is Zero
- **Bandwidth** is Infinite
- The Network is **Secure**
- **Topology** doesn’t change
- There is One **administrator**
- Transport **cost** is Zero
- The Network is **Homogeneous** (Gosling)
- Location is **Irrelevant** (Foxwell)
- All **system clocks are synchronized** (Unknown)





The **Fallacies** of Distributed Computing

(In other words, **don't make these mistaken assumptions!**)

- The Network is **Reliable:** things break (HW &SW); design for failure
- **Latency** is Zero: Speed of Light limit! 30+ ms RT US to Europe
- **Bandwidth** is Infinite: No, due to packet loss (Shannon 1948!)
- The Network is **Secure:** 50% enterprises secure *only* their perimeter
- **Topology** doesn't change: changes constantly! new devices, routes
- There is One **administrator:** multiservice apps (mashups)
- Transport **cost** is Zero: *someone* is paying for all this!
- The Network is **Homogeneous** (Gosling): multiple OS, apps, browsers..
- Location is **Irrelevant** (Foxwell): *Jurisdiction* is important!
- All **system clocks are synchronized** (Unknown): what time is it *really*?