University of Stuttgart

Institute of Parallel and
Distributed Systems (IPVS)

Universitätsstraße 38
D-70569 Stuttgart

Net-based Applications: Introduction

Adnan Tariq

(Based on the slides of Dr. Frank Dürr)

Overview

Distributed Systems and Applications

- Definition
- Reasons for Distribution
- Logical Model
- Communication Models
- System Software
- Client/Server Model
- Application Architectures

Definition (1)

- *A distributed system is one I cannot get any work done because a machine I never heard of has crashed* – L. Lamport

- *A distributed system consists of a collection of autonomous computers linked by a computer network and equipped with distributed system software* – Coulouris, Dollimore, Kindberg

- *Distributed applications* are executed on distributed systems
  - Web-based applications
  - Booking and information systems
  - Enterprise resource planning (ERP) systems

Definition (2)

Key concepts of a distributed system are:

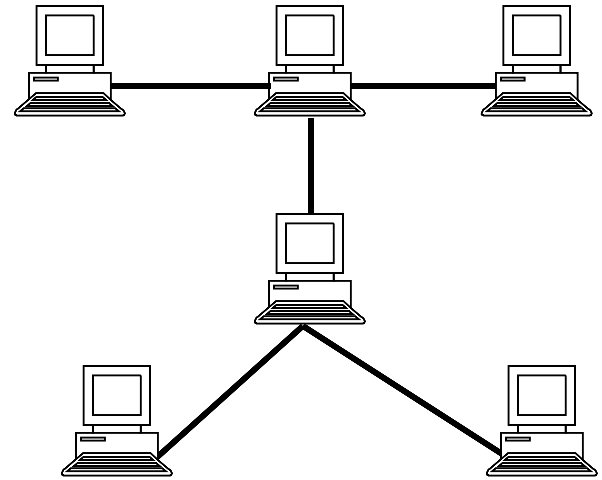
- Distribution transparency
  - **Access transparency:** Local and remote resources are accessed in the same way
  - **Location transparency:** The location of objects is not known to the users
  - **Replication transparency:** The number of copies that exist of an object is not known to the users
  - **Fragmentation transparency:** Objects are accessed without knowledge about any possible fragmentation
- Independent, autonomous machines

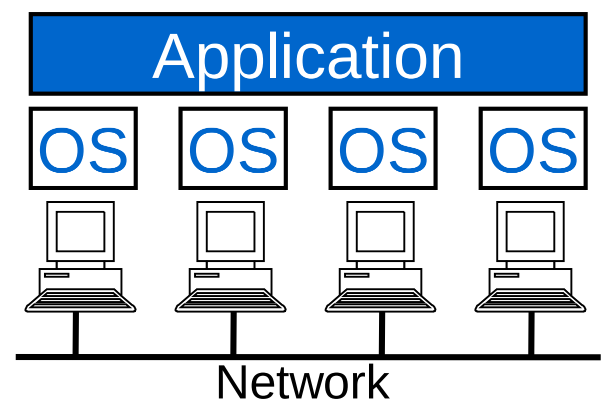
(Some) Reasons for Distribution

- Resource sharing
  - Distributed printer / mainframe access
- Collaboration
  - Distributed file systems
  - Instant messenger
- Scalability
  - Centralized processing and storage does not scale well
  - ➢ Distributed processing and storage
    - ➢ Consistency and coordination problems
- Fault-Tolerance
  - Single computing / storage node results in single point of failure
  - ➢ Replication of data / pooled computing resources

Logical Model

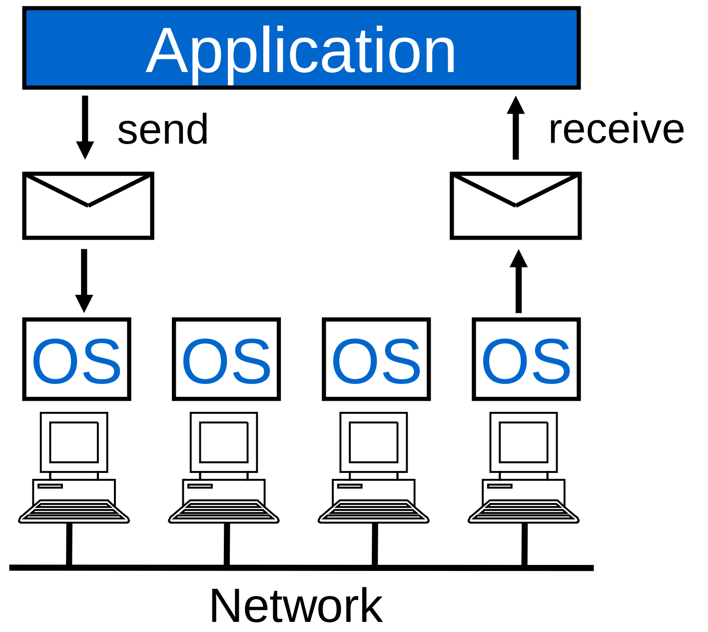
- Distributed system represented as graph with nodes and edges
  - Nodes: autonomous computers
    - No shared physical memory
  - Edges: network connections
- Simplified logical model (no network and topological issues considered)
  - Nodes are connected by some network
  - Nodes may have different hardware and operating systems
  - Applications reside on nodes and communicate across node boundaries





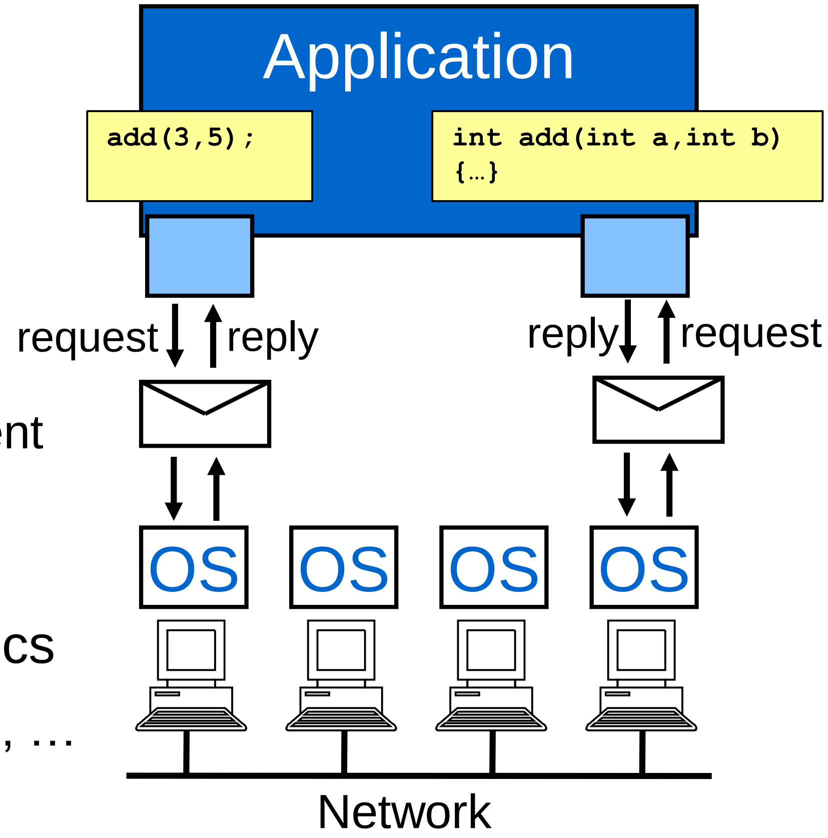
Communication: Message Passing

- Data is exchanged explicitly
  - send and receive primitives
- Decoupling via message buffers at sender and / or receiver
  - Different synchronization points possible (blocking / unblocking)
- Conversion of data between heterogeneous end-systems necessary (marshalling)
- Programming abstraction differs from local interaction
  - Berkeley-Socket API
  - MPI

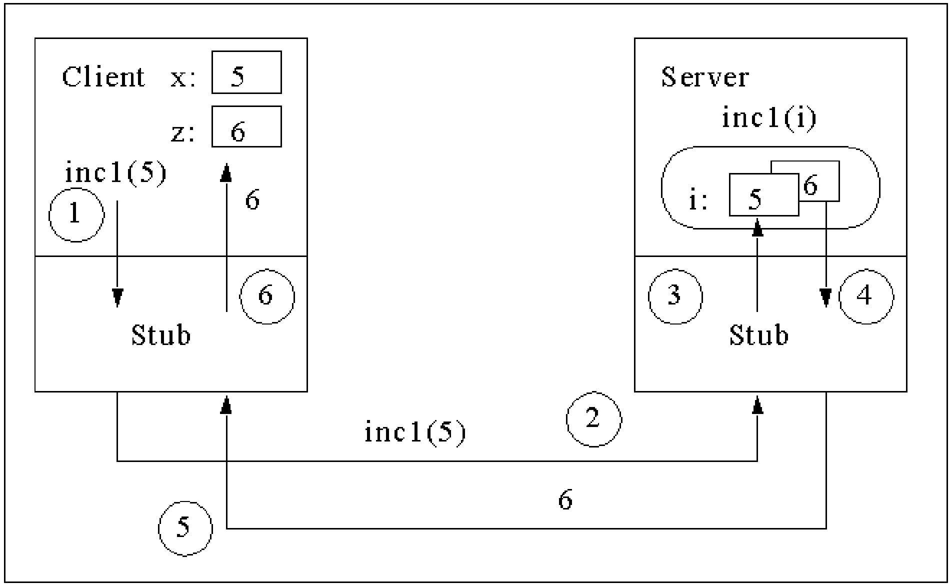


Communication: Remote Procedure Call

- Higher abstraction that mimics local procedure call
  - Similar to non-distributed interaction
  - Local proxies for remote entities
- Automatic data-conversion
  - Specification via machine independent language (Interface Definition Language, IDL)
- Different parameter passing semantics
  - By-value, by-reference, copy-restore, …
- Examples
  - CORBA, DCOM, Java RMI



Call-by-Value



function inc1 (i : integer) : integer
begin
i := i + 1;
return (i);
end;

-- Call (client side):

z := inc1(x);

Client stub : Copies the value of variable *x* as argument value into message

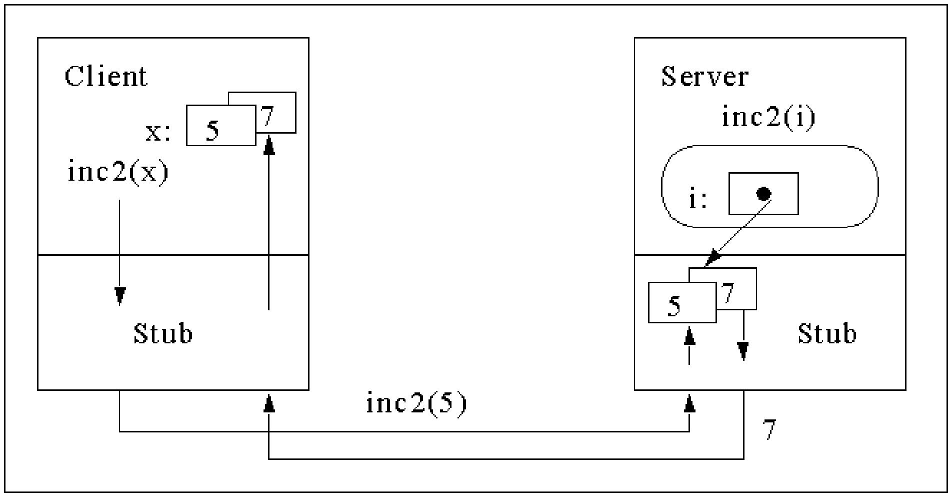
Server stub: Copies argument from message to variable *i*

Server stub: Copies value from variable *i* as result value into message

Client stub : Copies result from message to variable *z*

Call-by-Reference

Possible implementation using a *call-by-copy/restore* approach:



function inc2 (var i : integer) **begin**

i := i + 2;

end;

-- Call (client side):

inc2(x);

Call:

Client stub : Copies variable *x* as argument into message
Server stub: Copies argument from message to stub variable
Passes *pointer to* stub variable to procedure

Reply:

Server stub: Copies value from stub variable as result value into message
Client stub : Copies result from message to variable *x*

Difficulties with the Call-by-Copy/Restore Approach

```
program test (output)
var a : integer;

procedure doubleinc
                (var x, y : integer)
begin
    x := x + 1;
    y := y + 1;
end;

begin
   a := 0;
   doubleinc(a,a);
   writeln(a);
end.
```

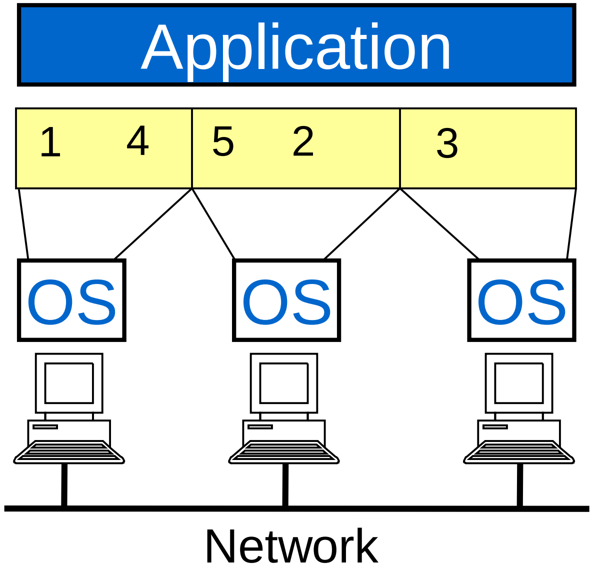
Problem:

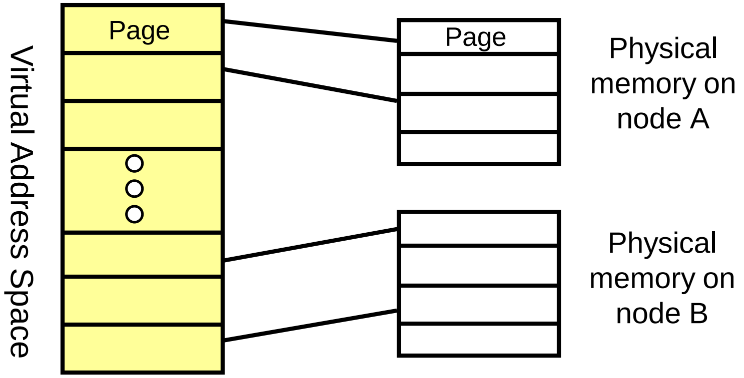
- Different results for local (a=2) and remote (a=1) execution
- Reason: Different copies for a single reference variable

Alternative Implementation: Server accesses parameter value remotely ⇨ huge overhead

Communication: Distributed Shared Memory

- Distributed application shares a common data space
  - System realizes a global view on memory (granularity is page)
  - Communication via read and write
- Required memory pages are copied to local address space
  - Similar to demand-paging in OS
- Scalability and efficiency
  - Size bound by virtual address space size
  - Remote access requires copying of data → separation of execution and data-storage
  - Many optimizations possible (e.g. replicated read-only pages)





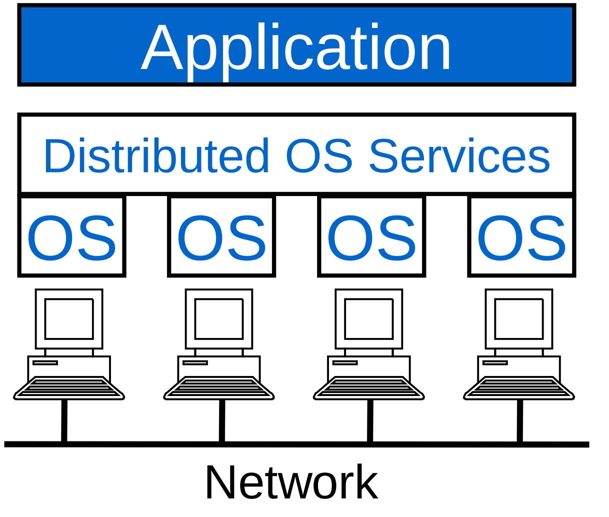
System Software

Many possible solutions to provide system software support for distributed applications

- Distributed operating system
  - Amoeba, Clouds
  - RPC, DSM
- Network operating system (OS with network extensions)
  - Unix, Windows
  - Sockets
- Middleware (on top of network operating system)
  - CORBA, DCOM, Java RMI, DCE
  - Remote procedure / method call
  - Distributed transactions

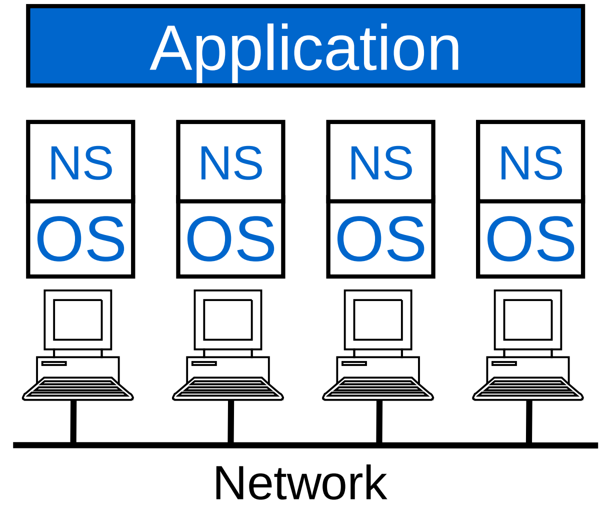
Distributed Operating System

- OS provides distributed virtual machine
  - Single system image
- Local kernel for local resource management enhanced by distributed operating system services
  - Distributed shared memory,
  - Task assignment (to machines),
  - Transparent file access,
  - Failure masking, …
- Programming abstractions
  - High-level primitives as system calls
  - Used by applications and OS



Network Operating System

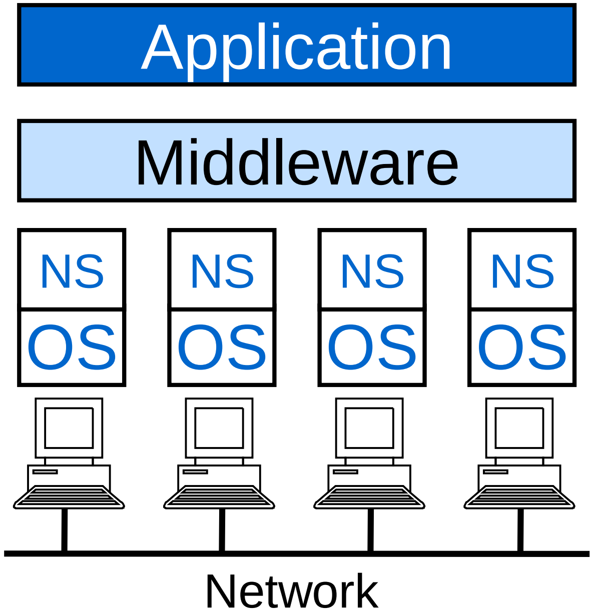
- Operating system with network extensions
- Local kernel for local resource management with basic network services
  - Message exchange, e.g. sockets
- Additional services
  - User process
    - Remote procedure calls
  - Kernel extensions
    - Distributed file systems
- Programming abstractions
  - Basic primitives for message exchange
  - Cumbersome and error-prone



NS: Network Services

Communication Middleware

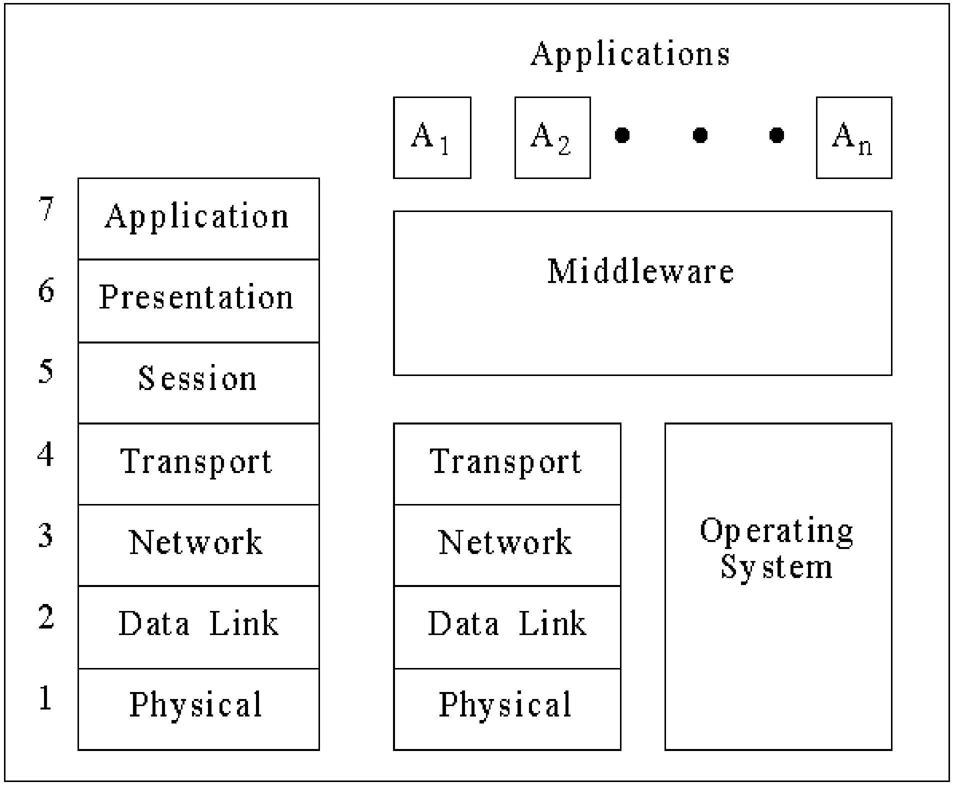
- Software layer between OS and application
  - Based on OS with network extension
  - Uses message exchange, e.g. sockets
- Objectives
  - Uniform programming abstraction
  - Mask heterogeneity
  - Interoperability (common protocol)
  - (Language independency)
- Programming abstractions
  - RPC, RMI
  - Message-oriented middleware (MOM)



NS: Network Services

➢ Brief RPC example

The Typical Architecture of a Distributed System with Middleware



The typical architecture of a distributed system in terms of the seven-layer OSI model.

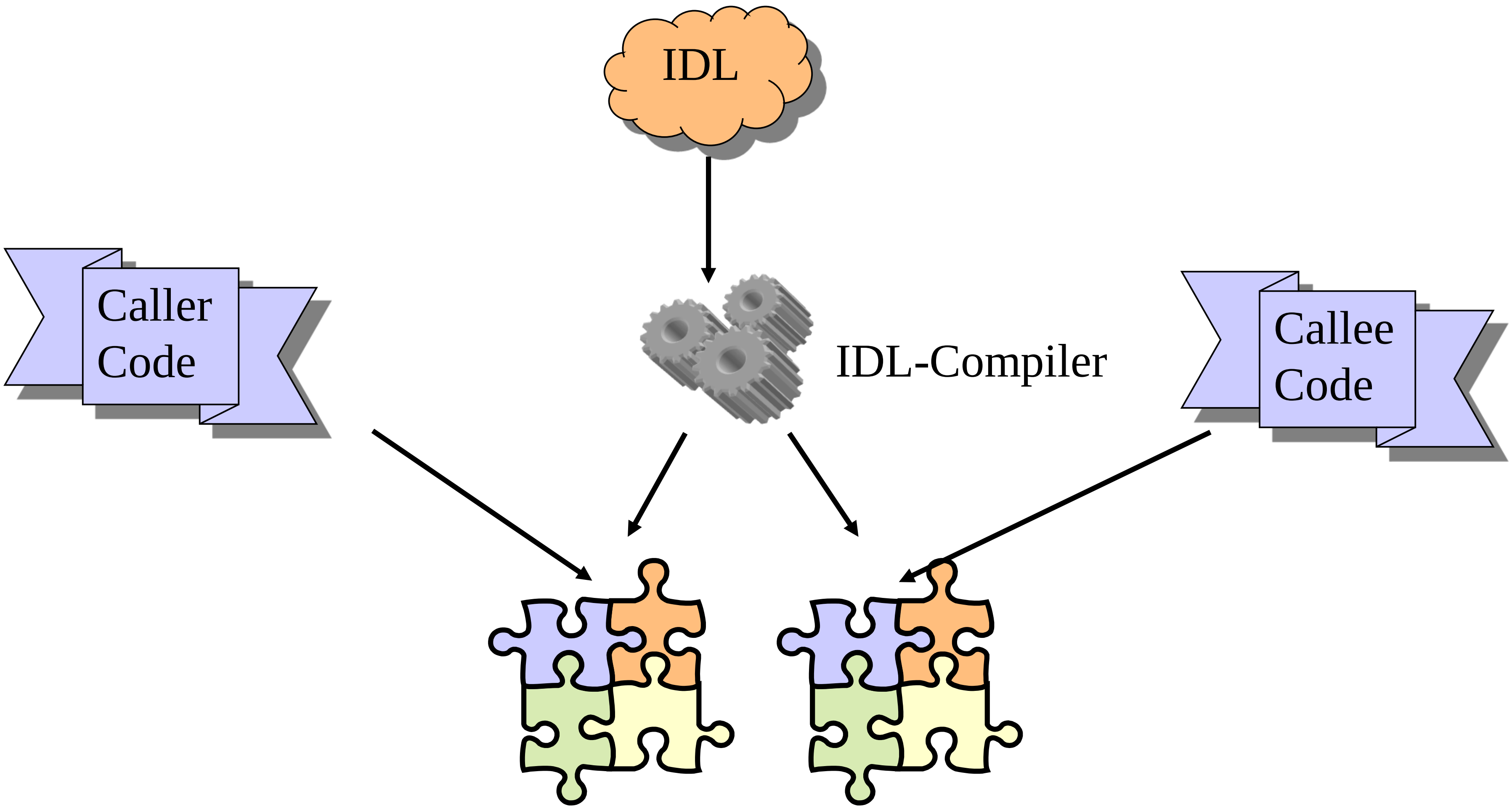
Transport System:

- Consists of layers 1 through 4
- Message-based inter-process communication
- No location transparency

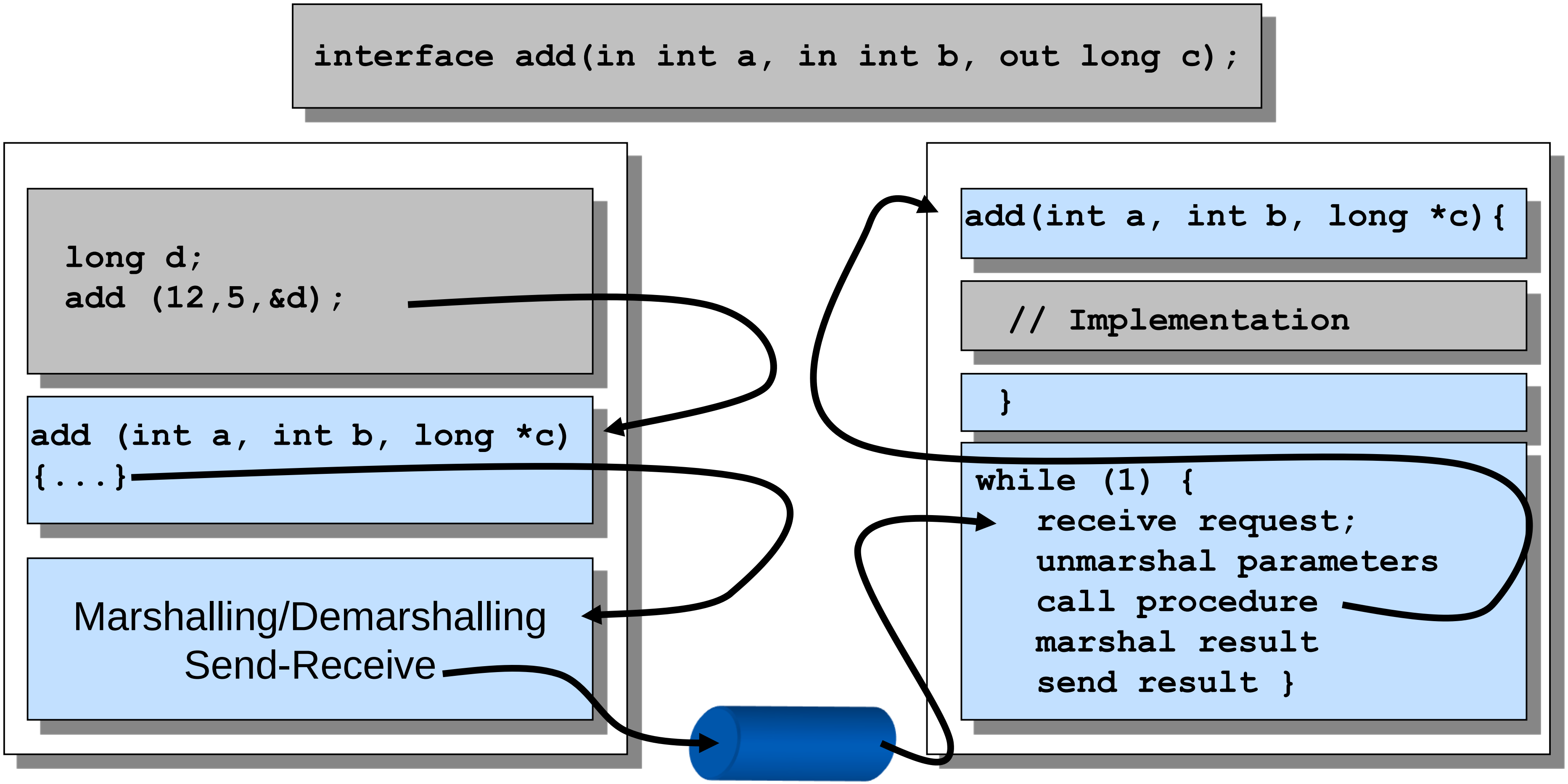
Middleware:

- Infrastructure for distributed applications
- Distribution transparency
- Examples: *Communication services*, security services, directory services, time services, file services, transaction services, etc.

Example: Remote Procedure Call

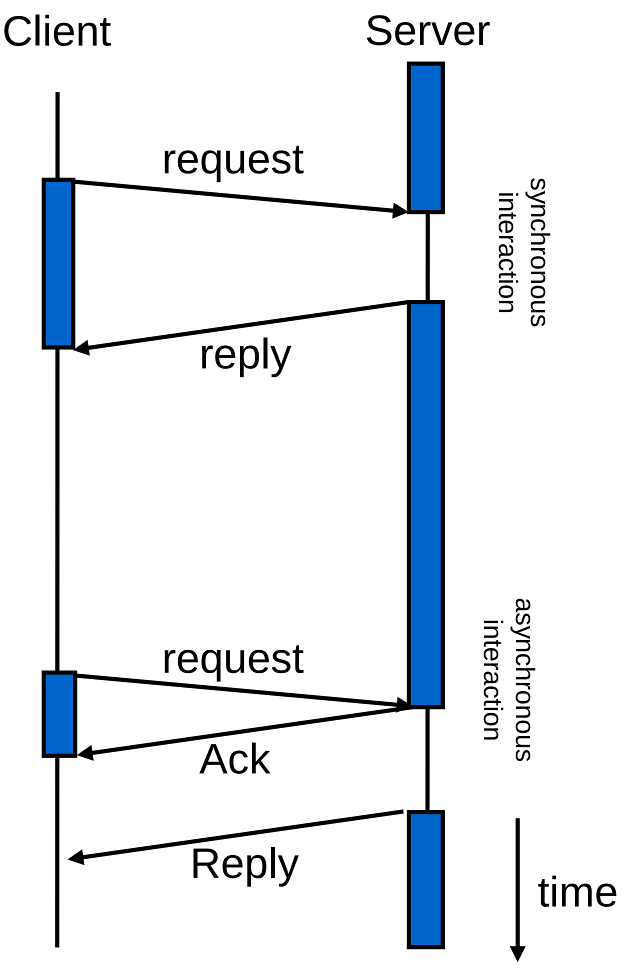


Example: Remote Procedure Call



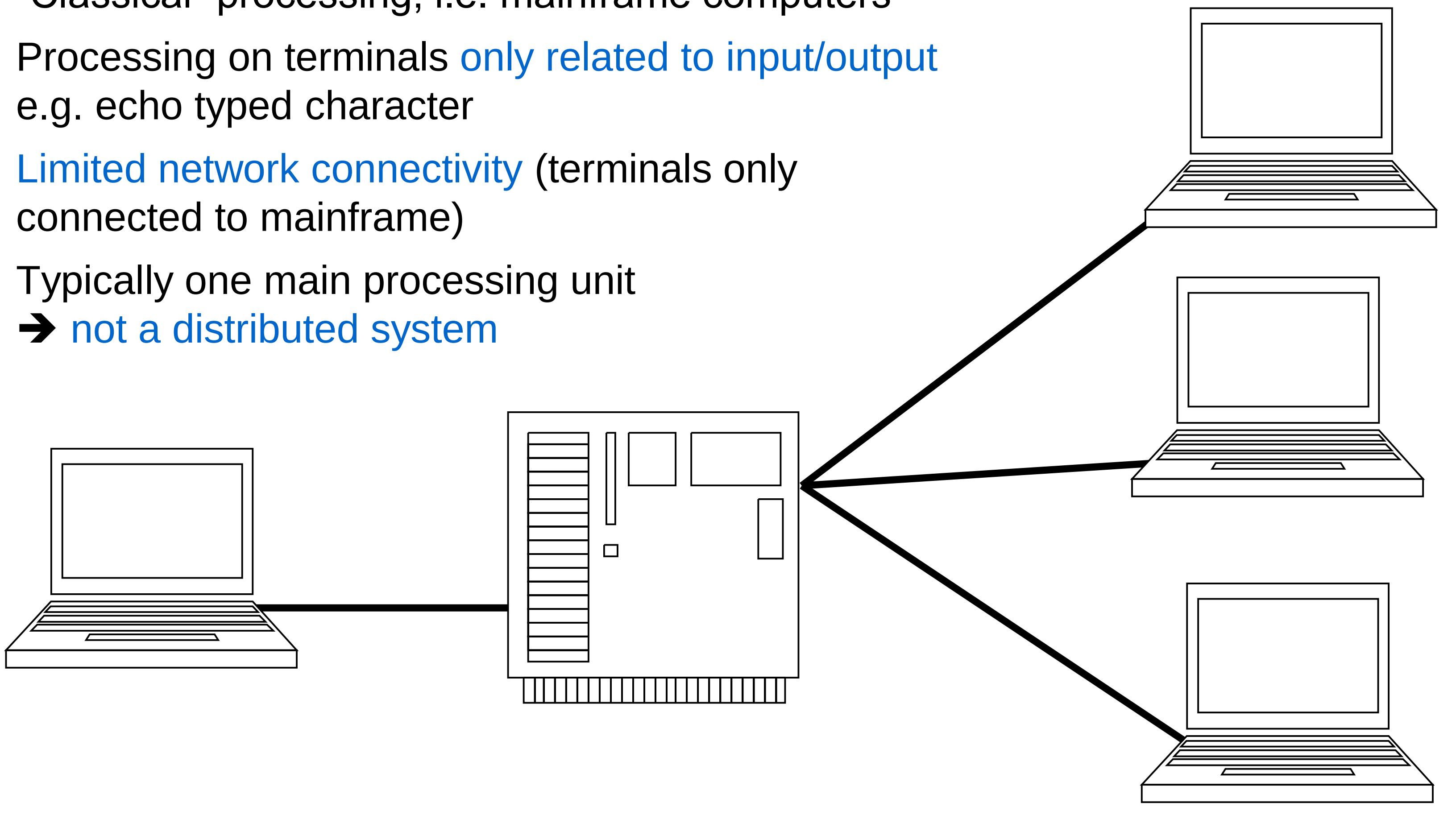
Structuring Applications: Client-Server-Model

- Distributed applications can be structured in two roles
  - Server
    - passive entity (awaits incoming requests)
    - processes requests and returns results
  - Client
    - active entity
    - contacts server to request a service
- Different interaction models
  - Synchronous
  - Asynchronous
- Clear distinction between client and server in practice sometimes hard
  - Entities might change their role
  - There may be thousands of clients and servers (e.g. distributed objects)



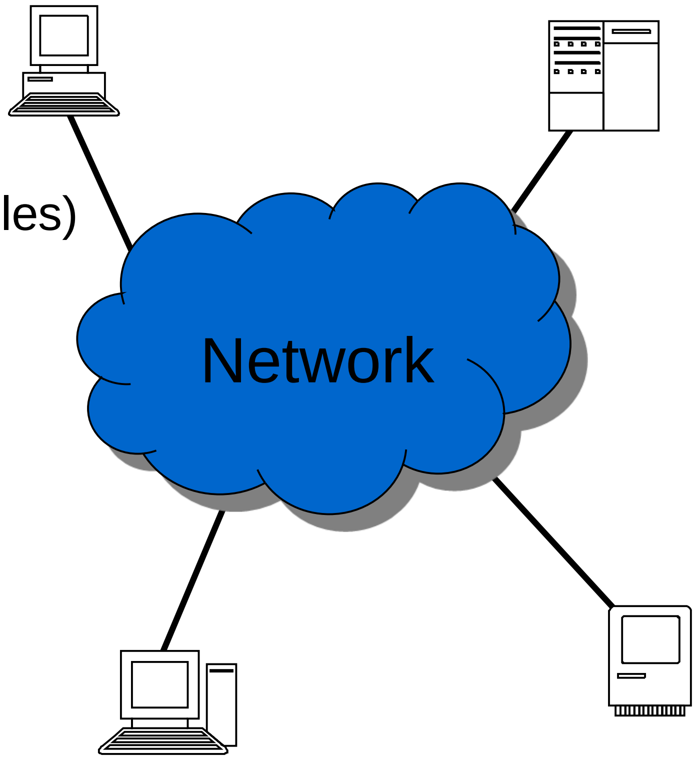
Centralized Application Architecture

- “Classical“ processing, i.e. mainframe computers
- Processing on terminals only related to input/output
  e.g. echo typed character
- Limited network connectivity (terminals only
  connected to mainframe)
- Typically one main processing unit
  ➔ not a distributed system



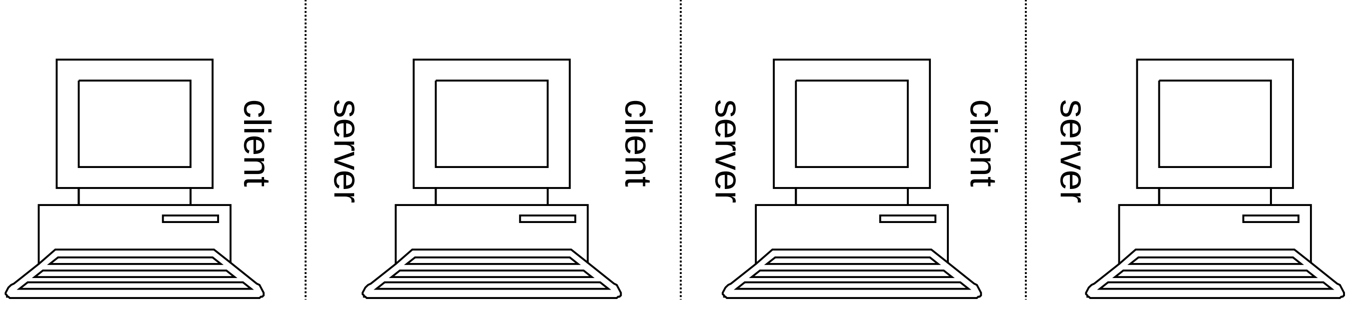
Client/Server Application Architecture

- Heterogeneous system
  - Nodes: hardware, OS, language, ...
  - Edges: protocols, physical link, ...
- Many clients and servers (with changing roles)
  - No main processing system
  - Nodes may have equal rights (peer-to-peer)
- Possible (design) problems
  - Coordination hard to achieve
  - Performance bottlenecks
  - Single points of failure
  - No clear separation of logical tasks
    (e.g. presentation, processing, storage)



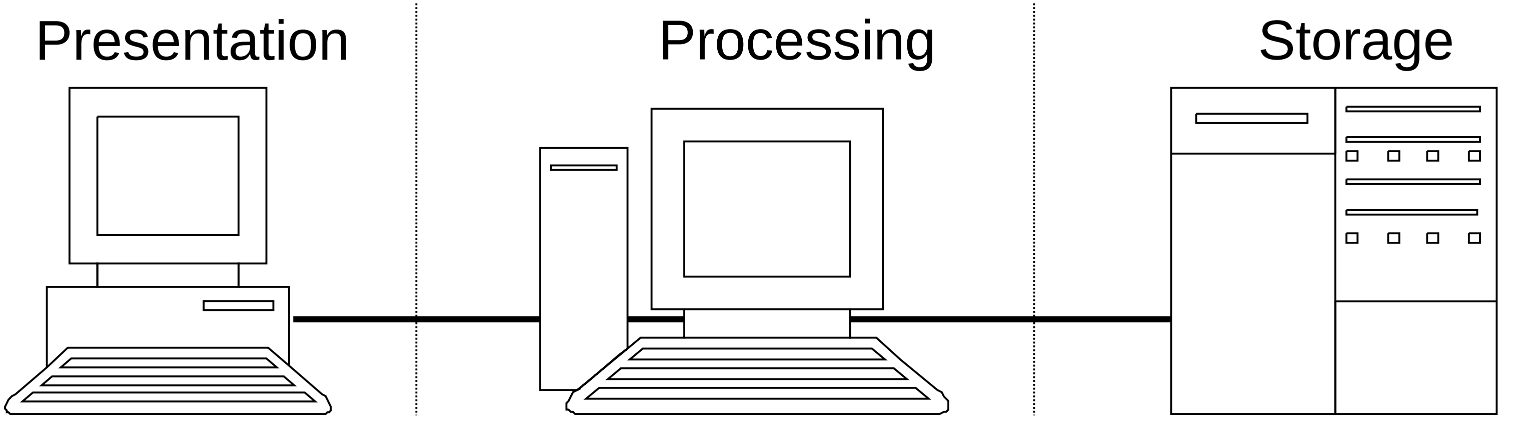
N-Tier Application Architecture

- Logical (and physical) application structuring in N tiers (aka multi-tier)
- Tiers provide a distinct functionality (e.g. store data)
- Tier M is server for tier M-1 and client of tier M+1
- Interaction only between “neighboring” tiers
  - Replacement of tiers less complex
  - Different implementations for individual tiers possible
- Design decision: How many tiers are needed? What is a good separation?



3-Tier Application Architecture

- Widely used logical and physical separation
  - Tiers: presentation, processing, storage
- Still heterogeneous (e.g. multiple presentation tiers)
- Often centralization with respect to processing and storage
- How to distribute functionality among nodes in a distributed system?
  ➔ performance and fault-tolerance issues



Presentation Tier

- Responsibility
  - User interface to clients
  - Text based: shells, host masks
  - Graphical user interface (GUI)
    - Mouse and keyboard interaction
    - Windows, buttons, scrollbars, etc.
- Processing
  - Only related with presentation
    - Echoing keystrokes, requesting data on push button activity
- No data storage
- Clear distinction from application logic and data storage

Logic (Application) Tier

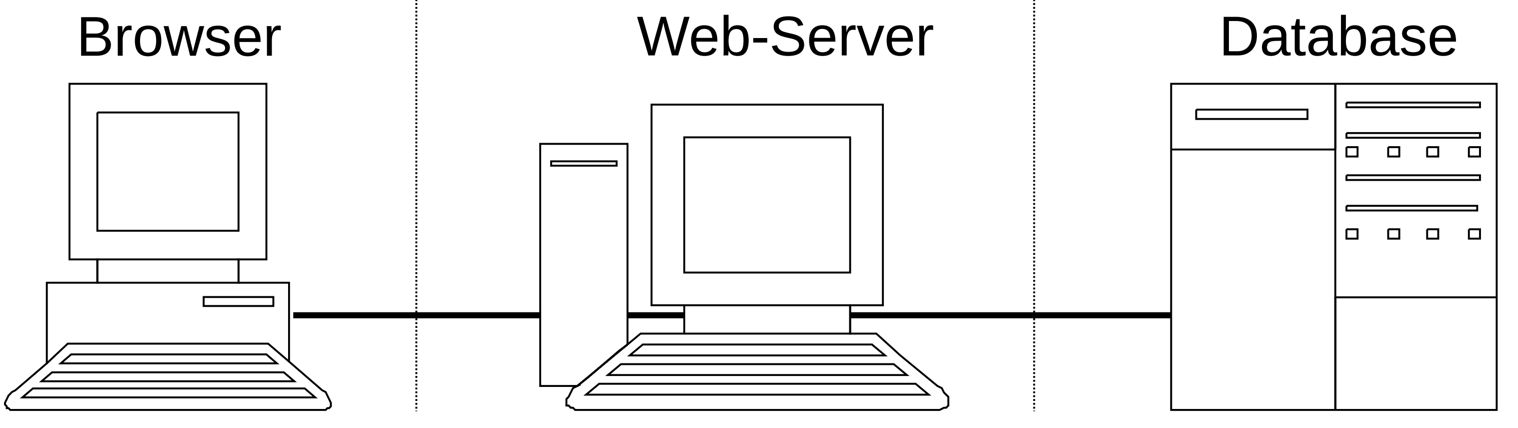
- Responsibility
  - Manipulation of data
  - Executing “business logic“
  - Access of persistent data from the database tier
  - Provides data to presentation tier
- No presentation of data
- Data storage
  - No persistent storage of data
  - Intermediate state may be captured for further processing

Database Tier

- Responsibility
  - Persistent storage of data
  - Concurrency control, i.e. transaction processing
  - Provides an appropriate interface to application tier
- No presentation
- Processing
  - Queries are processed and executed
  - No workflow or business logic

Web-based Application Architecture

- Special case of N–tier architecture
- Based on Internet technology, i.e. TCP/IP, HTTP
- Standard Components (browser, web server, DBMS)
- Customization: applets, servlets, database tables
- Suitable for Internet and intranets



Net-based Applications …

- In this class we present and discuss important aspects of net-based applications:
  - Basic network programming: sockets, client/server …
  - Web technologies: HTTP, JSP, Java Servlets, …
  - XML Technology: specification, parsing, processing, …
  - Basics of security: cryptography, certificates, …