Distributed Systems

Prof. Dr. Schahram Dustdar
Distributed Systems Group
Vienna University of Technology

dustdar@dsg.tuwien.ac.at
dsg.tuwien.ac.at

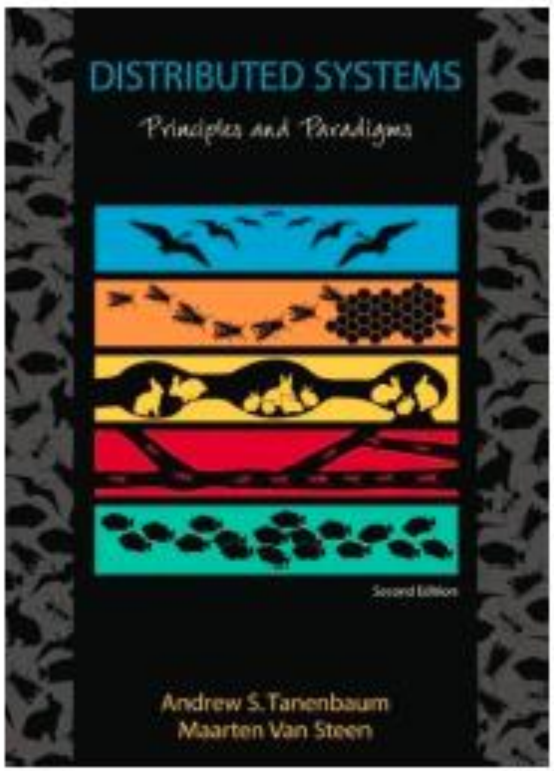
Distributed Systems Group

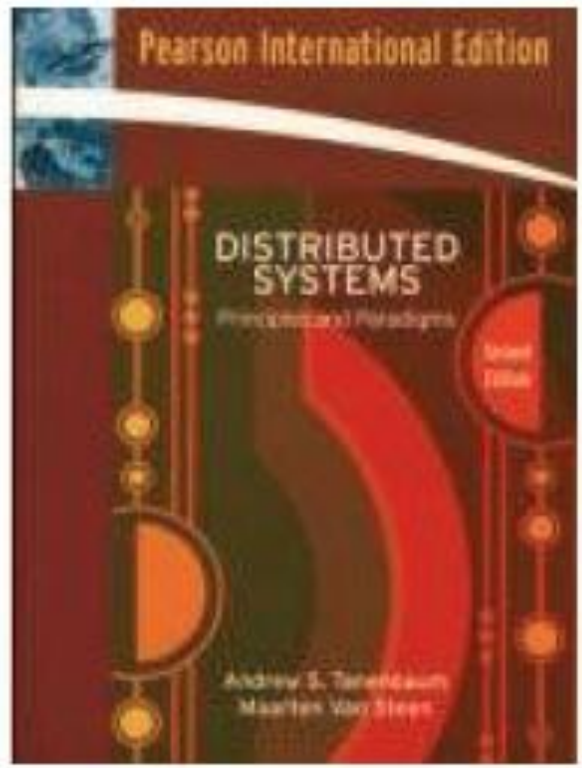
Outline

1. History
2. What is a distributed system?
3. Key concepts and design goals
4. Architectural styles

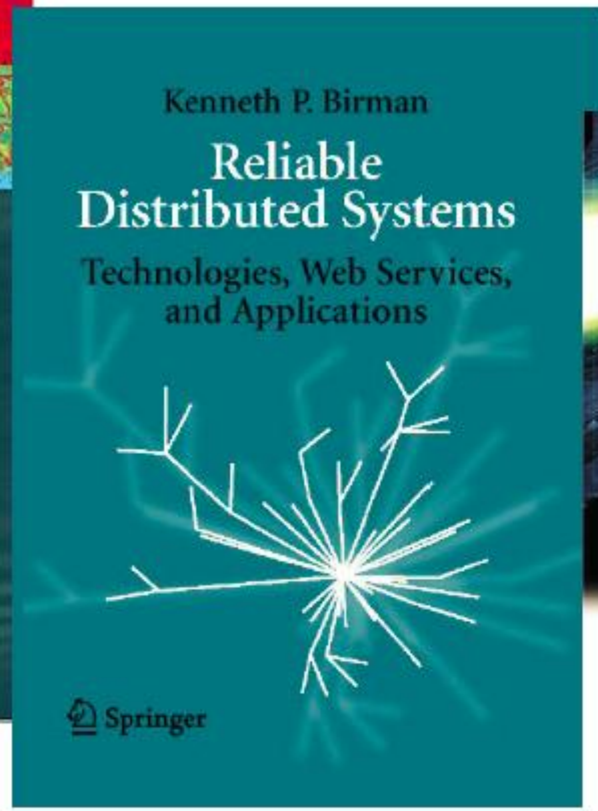
Lecture Material

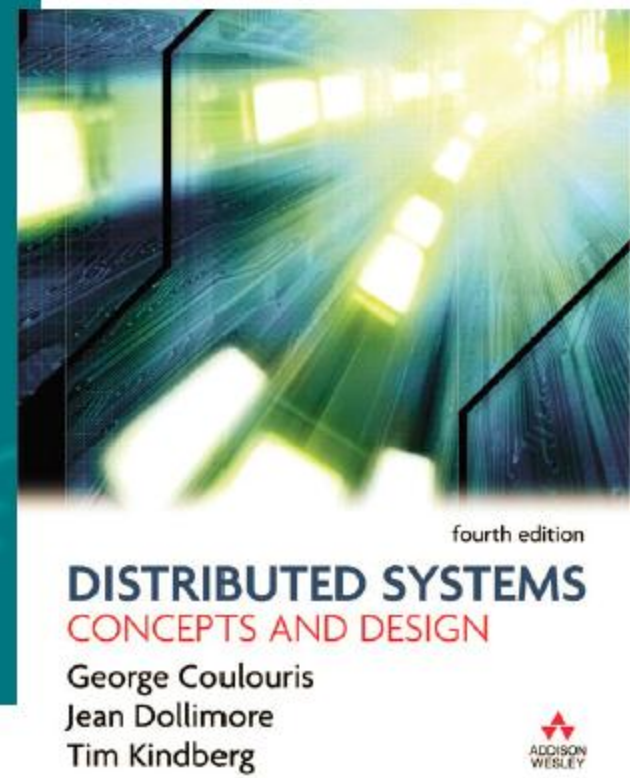
- Slides available for download, but not sufficient for self-study! Please read on…





Recommended additional reading





Prerequisites

- Data structures and algorithms (sequential)
- Operating systems / Systems programming
- Software engineering concepts
- Object-oriented programming

- For the **lab**: Java's support for modularity (packages and interfaces), object orientation, exceptions, distribution (RMI), code mobility (applets, class loader), and concurrency (threads and synchronization)

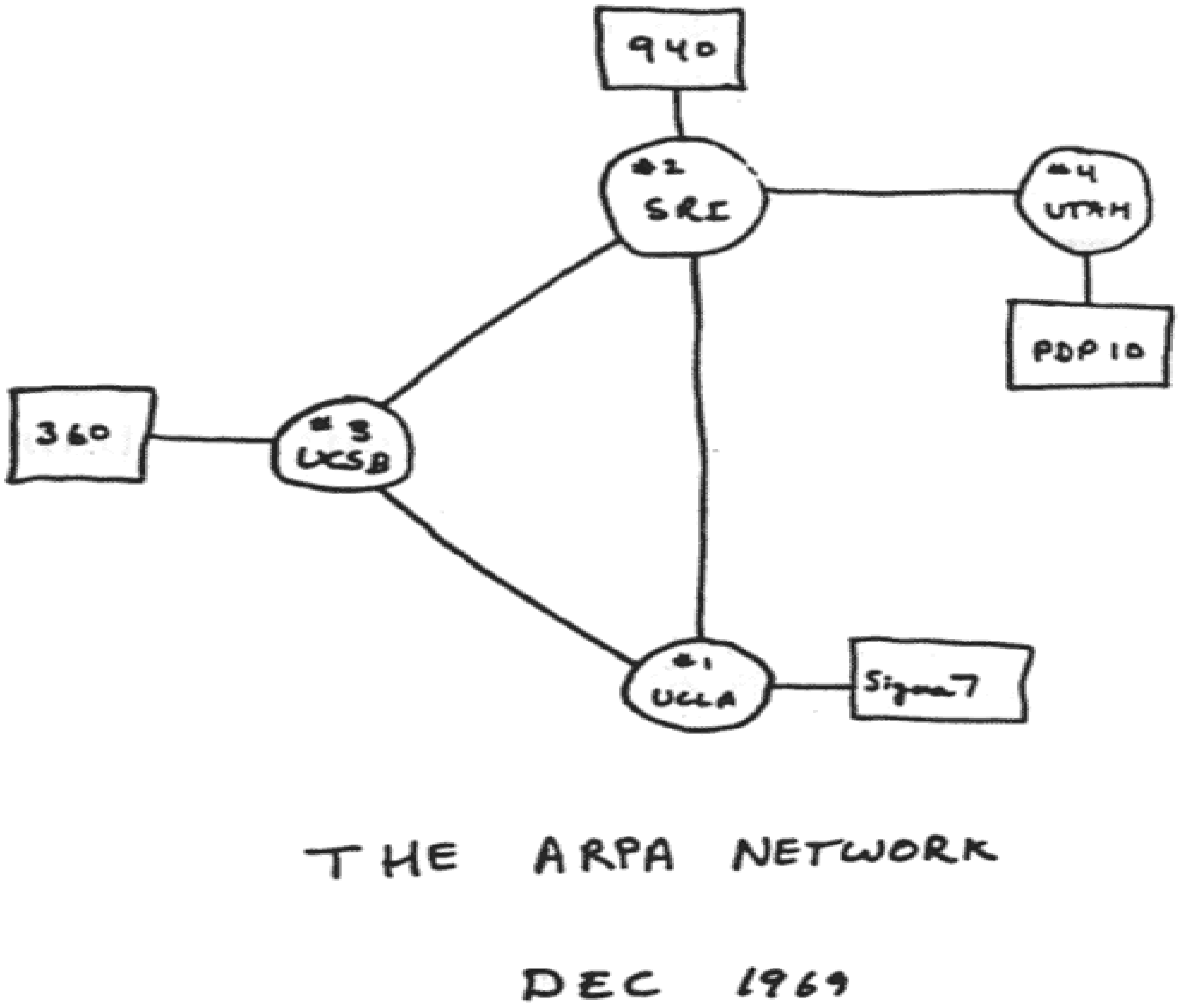
OVERVIEW AND INTRODUCTION

Evolution

- Until 1985 large and expensive stand-alone computers
- Powerful microprocessors (price/performance gain 1012 in 50 years)
- High-speed computer networks (LAN/WAN)

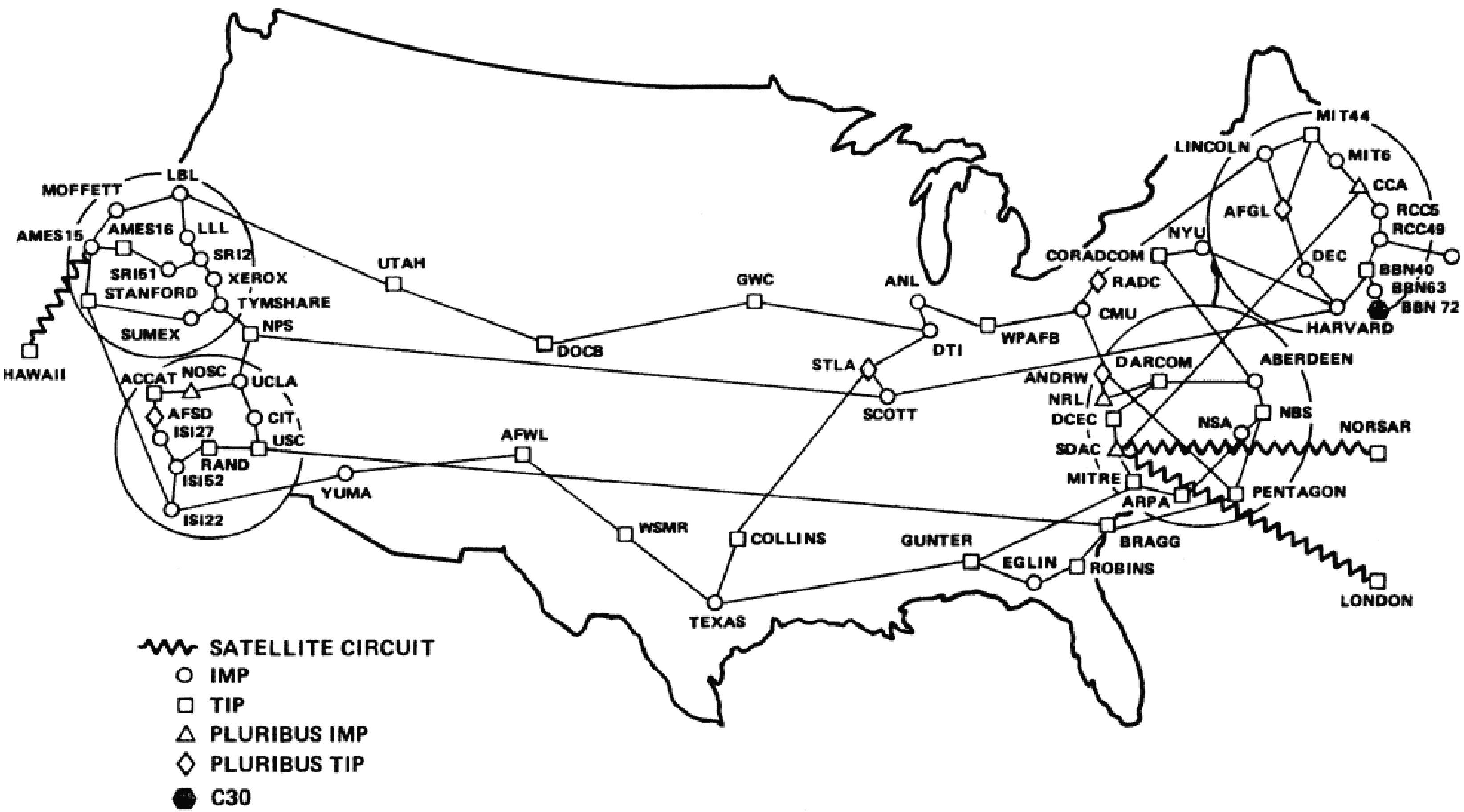
-> composition of computing systems of large numbers of computers connected by a highspeed network increase

The complete Internet 1969

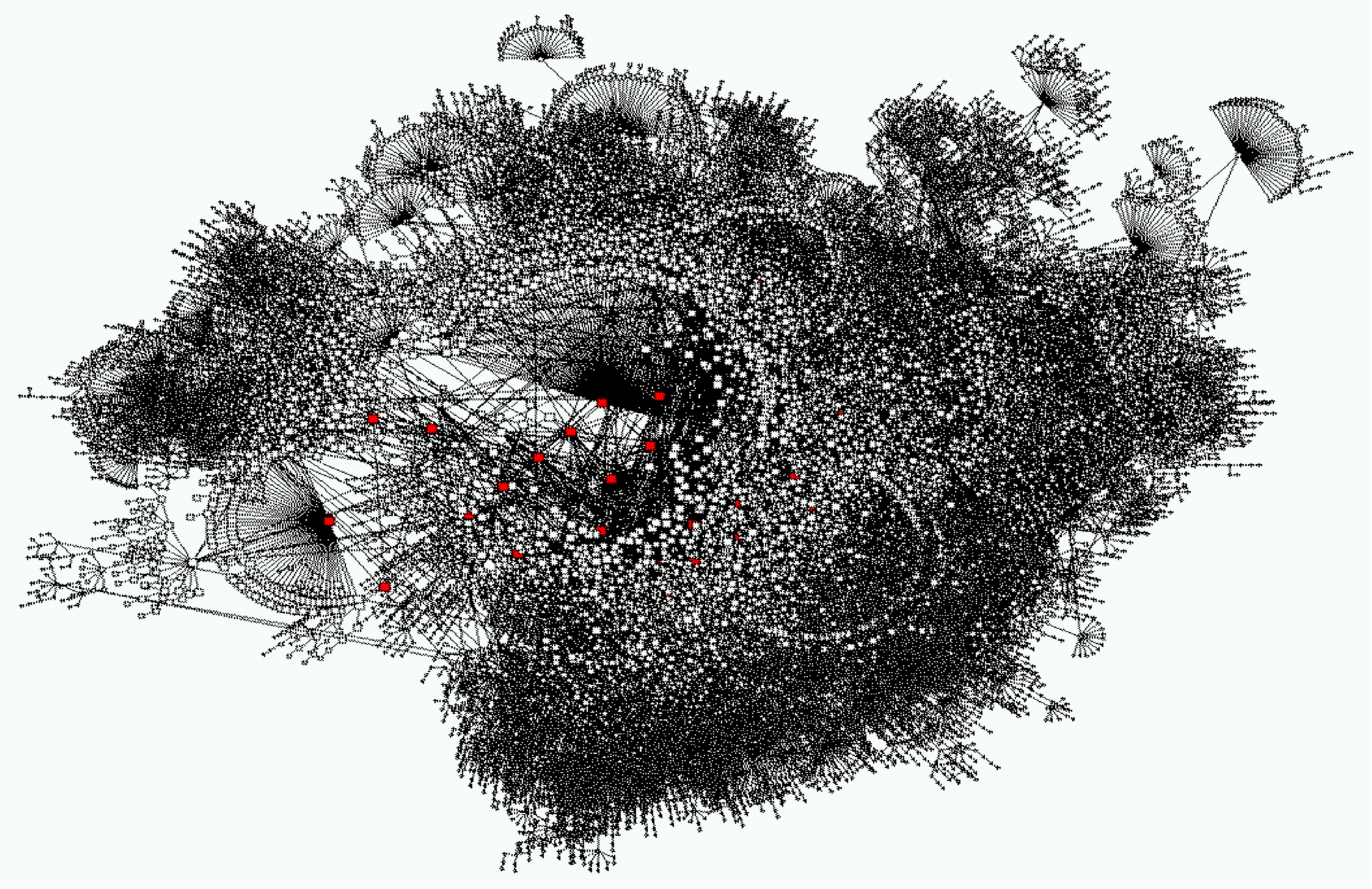


Part of the US Internet 1980

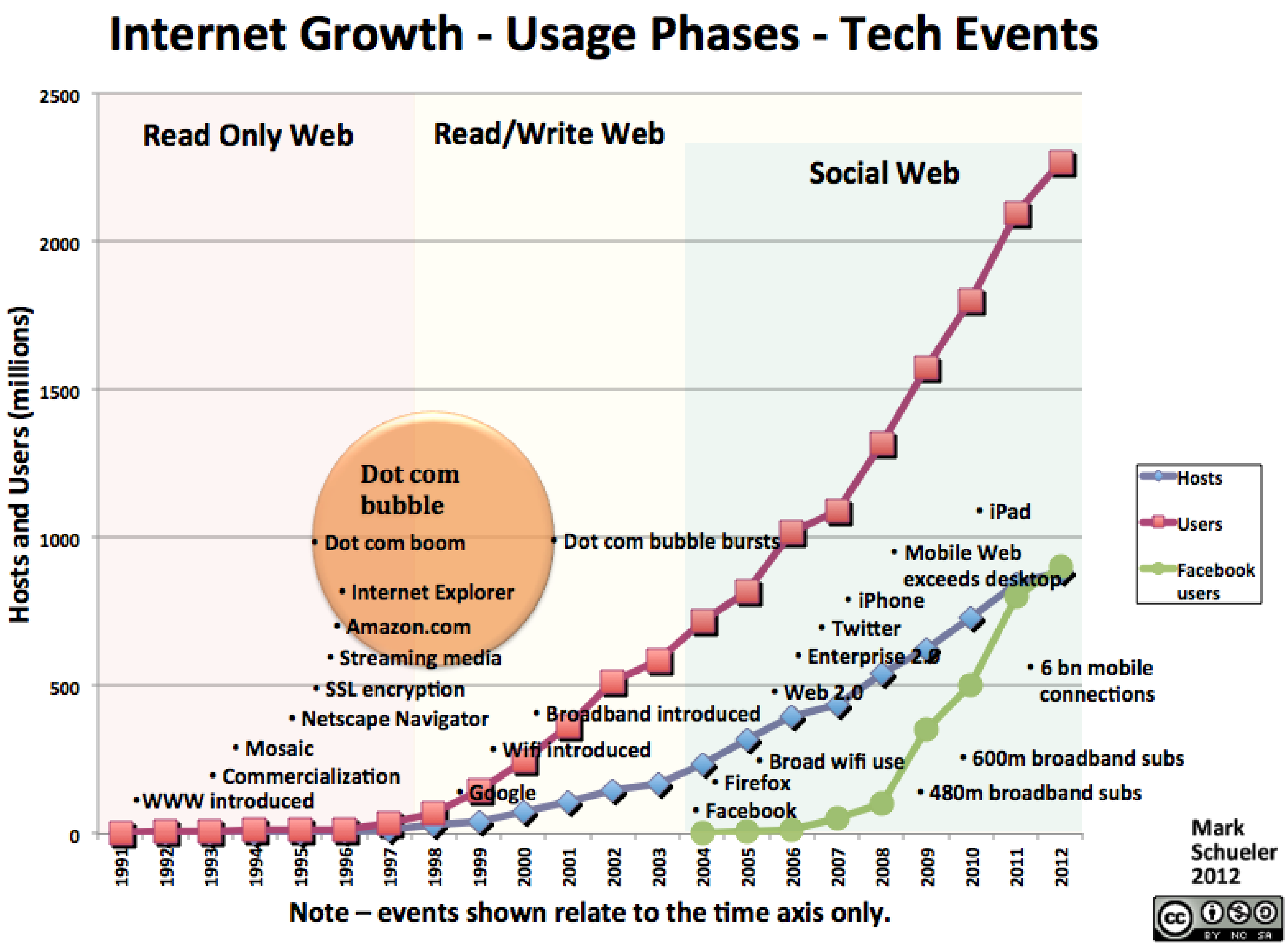
ARPANET GEOGRAPHIC MAP, OCTOBER 1980



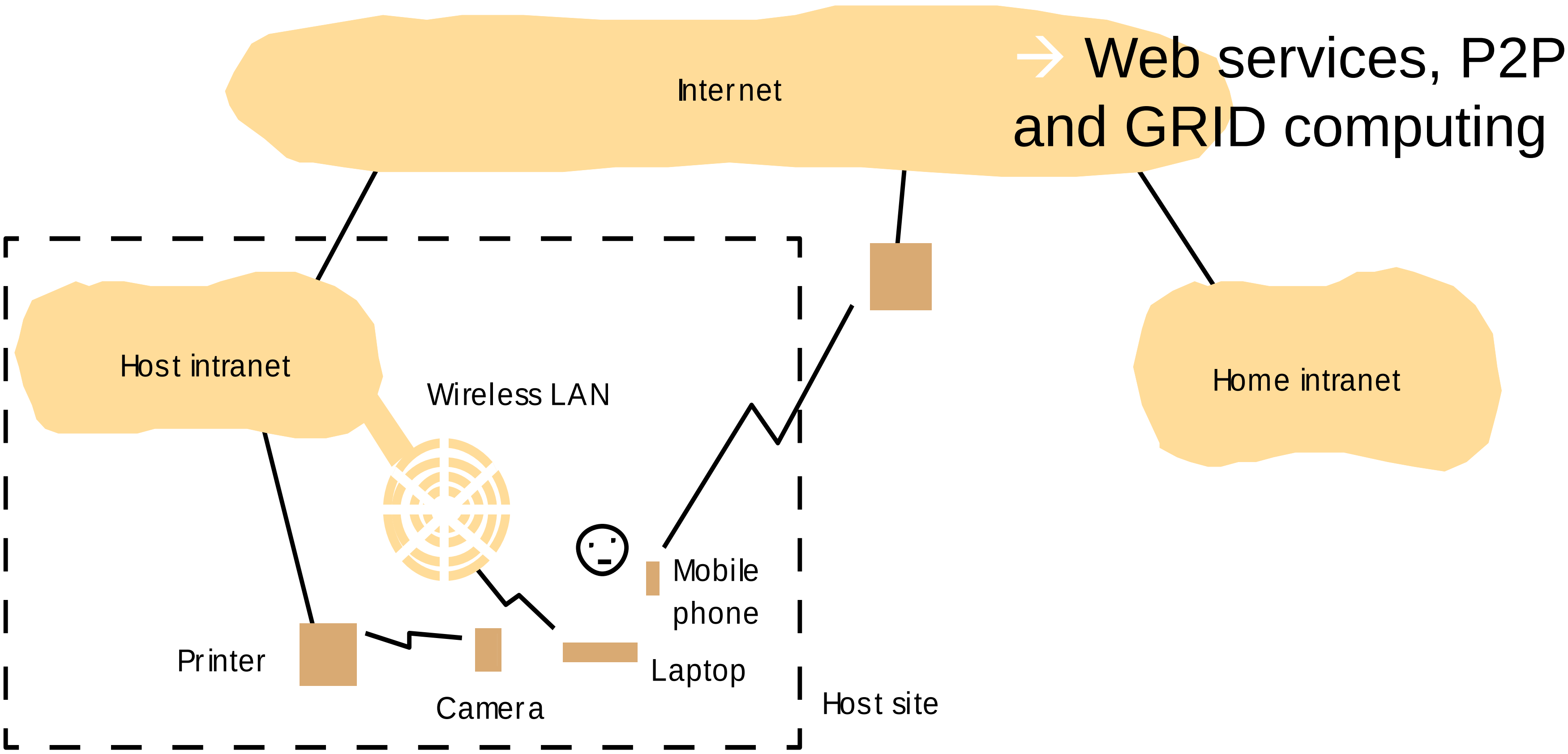
Part of a network today



Growth of the Internet



Portable and handheld devices in a distributed system



Evolution of Distribution technologies

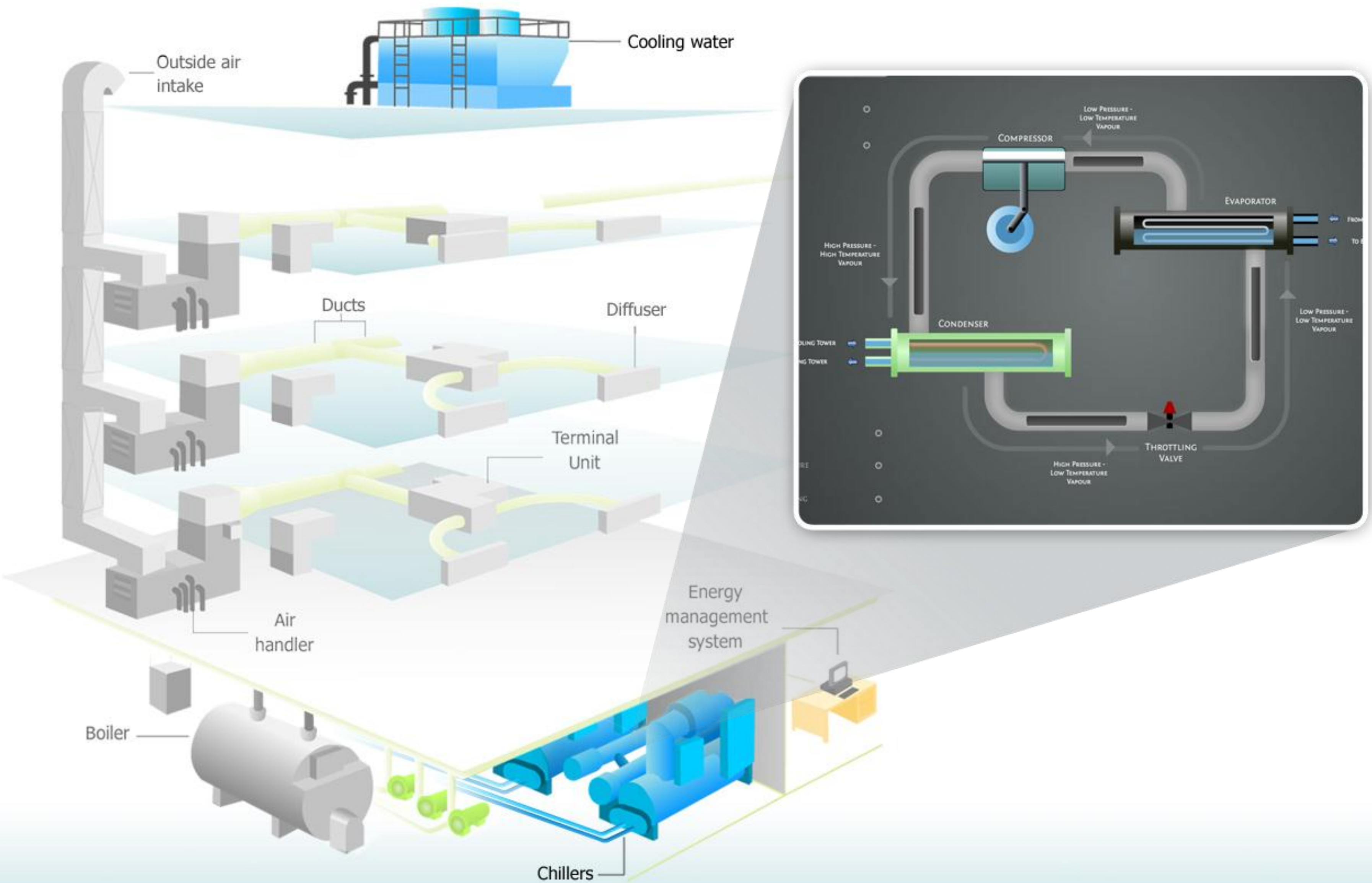
- Mainframe computers
- Workstations and local networks
- Client-server systems
- Internet-scale systems and the Web
- Sensor/actor networks in automation
- Mobile, ad-hoc, and adaptive systems
- Pervasive (ubiquitous) systems
- Today, less than 2% of processors go into personal computers!


Ministry of Interior
Directorate General of Civil Defence
24x7
DUBAI CIVIL DEFENCE
24x7 Direct Alarm System

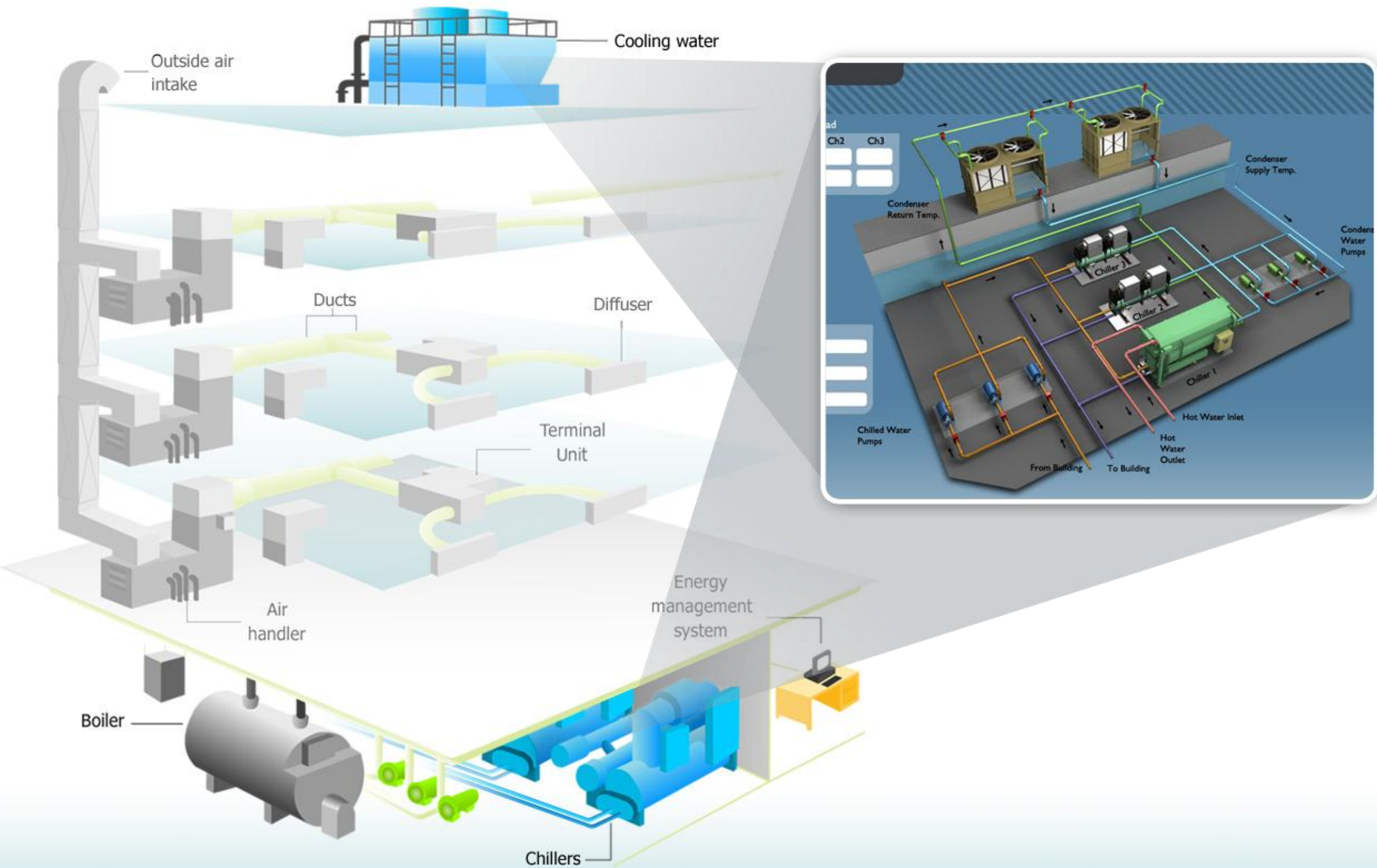
Emergency Hub

Command Control Center for Managed Services

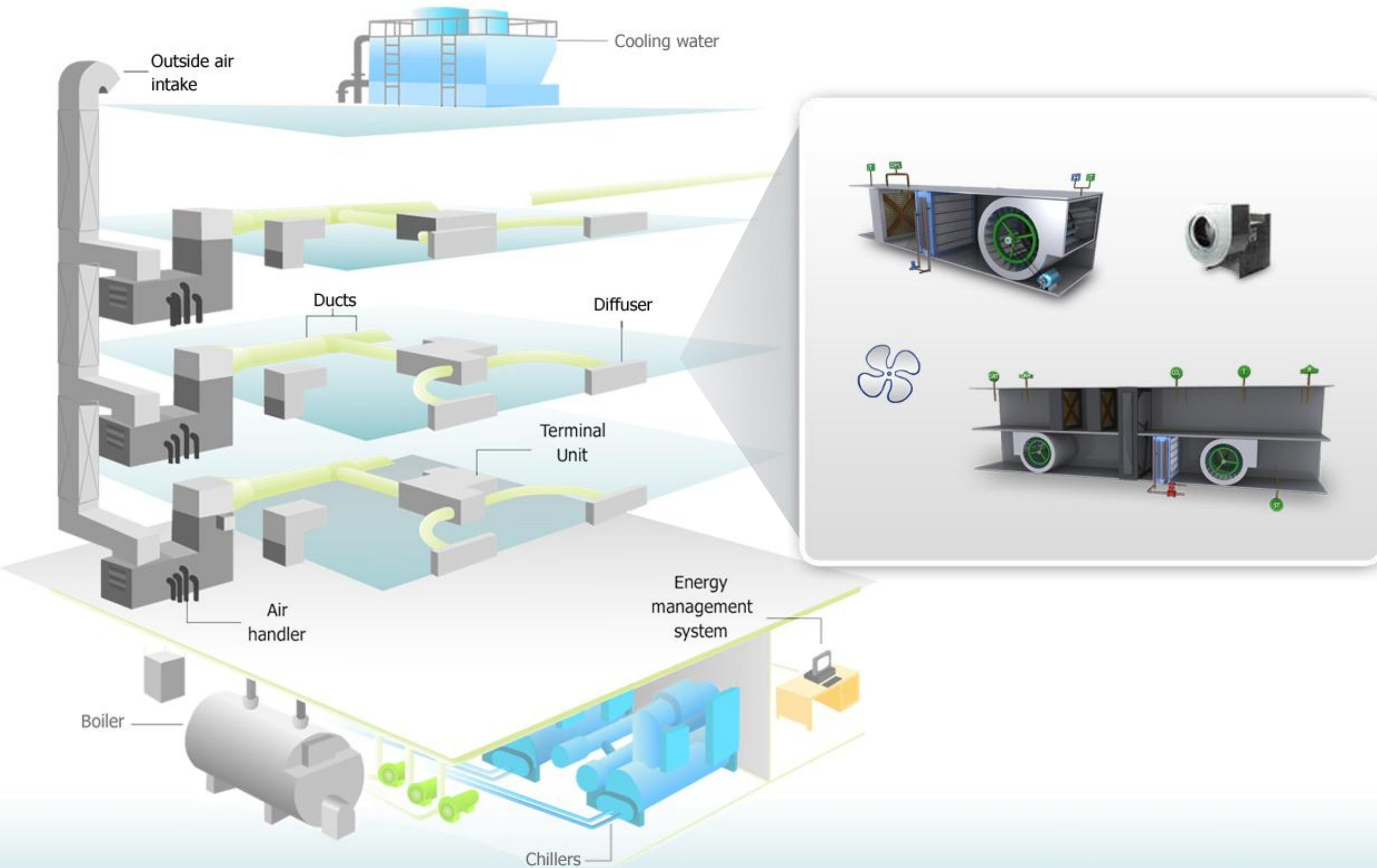
HVAC (Heating, Ventilation, Air Conditioning) Ecosystem



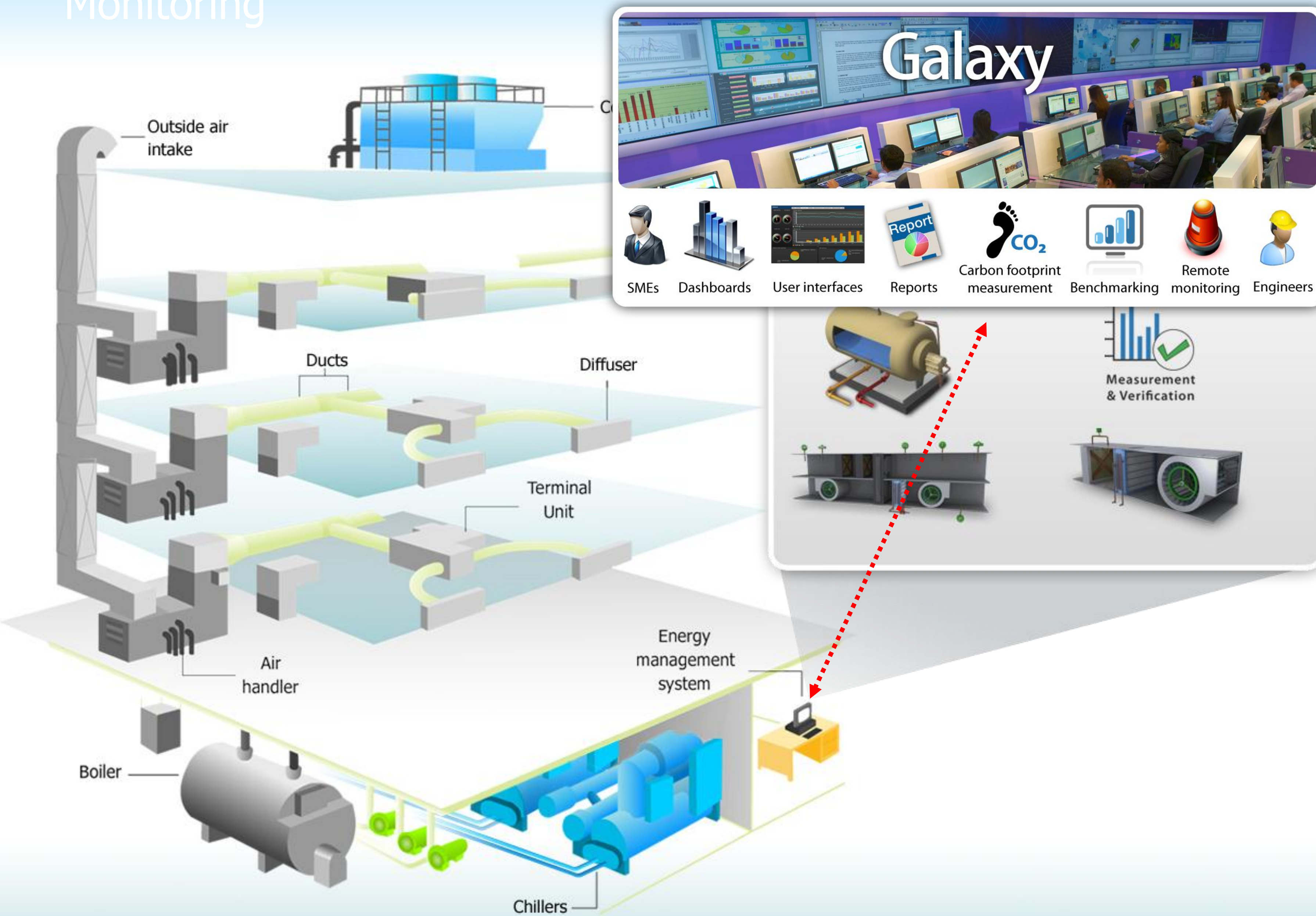
Water Ecosystem



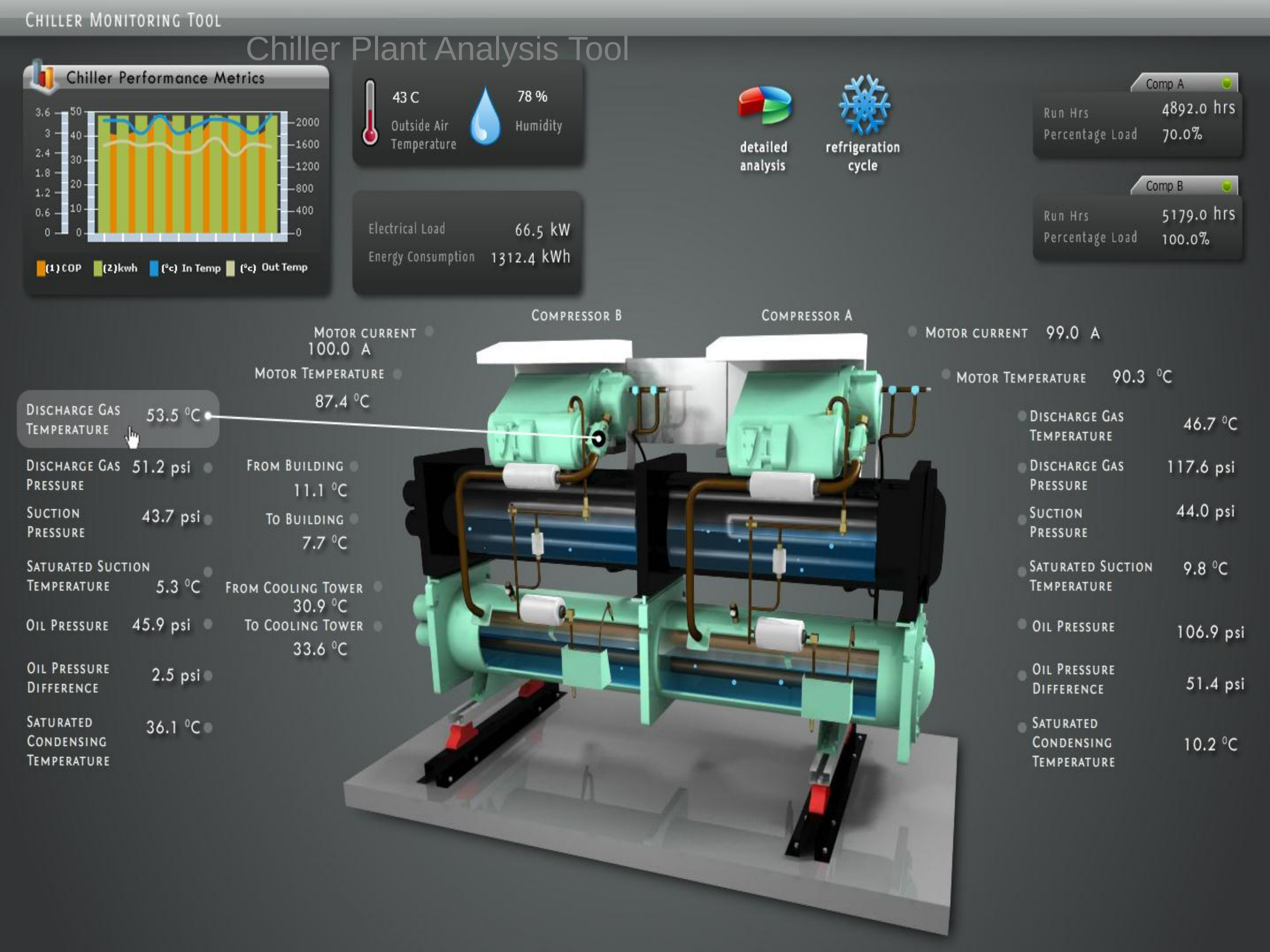
Air Ecosystem



Monitoring


Outside air intake
Ducts
Diffuser
Terminal Unit
Air handler
Energy management system
Boiler
Chillers
Galaxy
SMEs
Dashboards
User interfaces
Report
Reports
CO_2
Carbon footprint measurement
Benchmarking
Remote monitoring
Engineers
Measurement & Verification

Chiller Plant Analysis Tool



ICT for energy savings in buildings

Remote Service Maintainence

Definition of a Distributed System (1)

A **collection** of

independent computers

that appears to its users

as a **single coherent**

system.

Definition of a Distributed System (2)

A collection of **autonomous** computers linked by a computer **network** and supported by **software** that enables the collection to operate as an **integrated** facility.

Definition of a Distributed System (3)

You know you have one
when the **crash** of a computer
you have **never heard of**
stops you from getting any
work done. (Leslie Lamport)

Types of Distributed Systems (1)

- Object/component based (CORBA, EJB, COM)
- File based (NFS)
- Document based (WWW, Lotus Notes)
- Coordination (or event-) based (Jini, JavaSpaces, publish/subscribe, P2P)
- Resource oriented (GRID, Cloud, P2P, MANET)
- Service oriented (Web services, Cloud, P2P)

Types of Distributed Systems (2)

- Distributed Computing (cluster, GRID, cloud)
- Distributed Information Systems (EAI, TP, SOA)
- Distributed Pervasive Systems (often P2P, UPnP in home systems, sensor networks, ...)

Concepts of Distributed Systems

- Communication
- Concurrency and operating system support (competitive, cooperative)
- Naming and discovery
- Synchronization and agreement
- Consistency and replication
- Fault-tolerance
- Security

KEY CONCEPTS AND DESIGN GOALS

Why distribute at all?

- Connecting users to resources and services

  □ Basic function of a distributed system

- Dependability and Security

  □ Availability, Fault Tolerance (FT), Intrusion Tolerance, ...

- Performance

  □ Latency, throughput, ...

Otherwise: Don’t distribute, its far more complex hence expensive, error-prone, ...

Design goals in Distributed Systems

- Resource sharing (collaborative, competitive)
- Transparency
- Hiding internal structure, complexity
  - Openness, Portability, Interoperability, ...
- Services provided by standard rules
- Scalability
- Ability to expand the system easily
- Concurrency
  - inherently parallel (not just simulated)
- Fault Tolerance (FT), availability

The 8 Fallacies of Distributed Computing

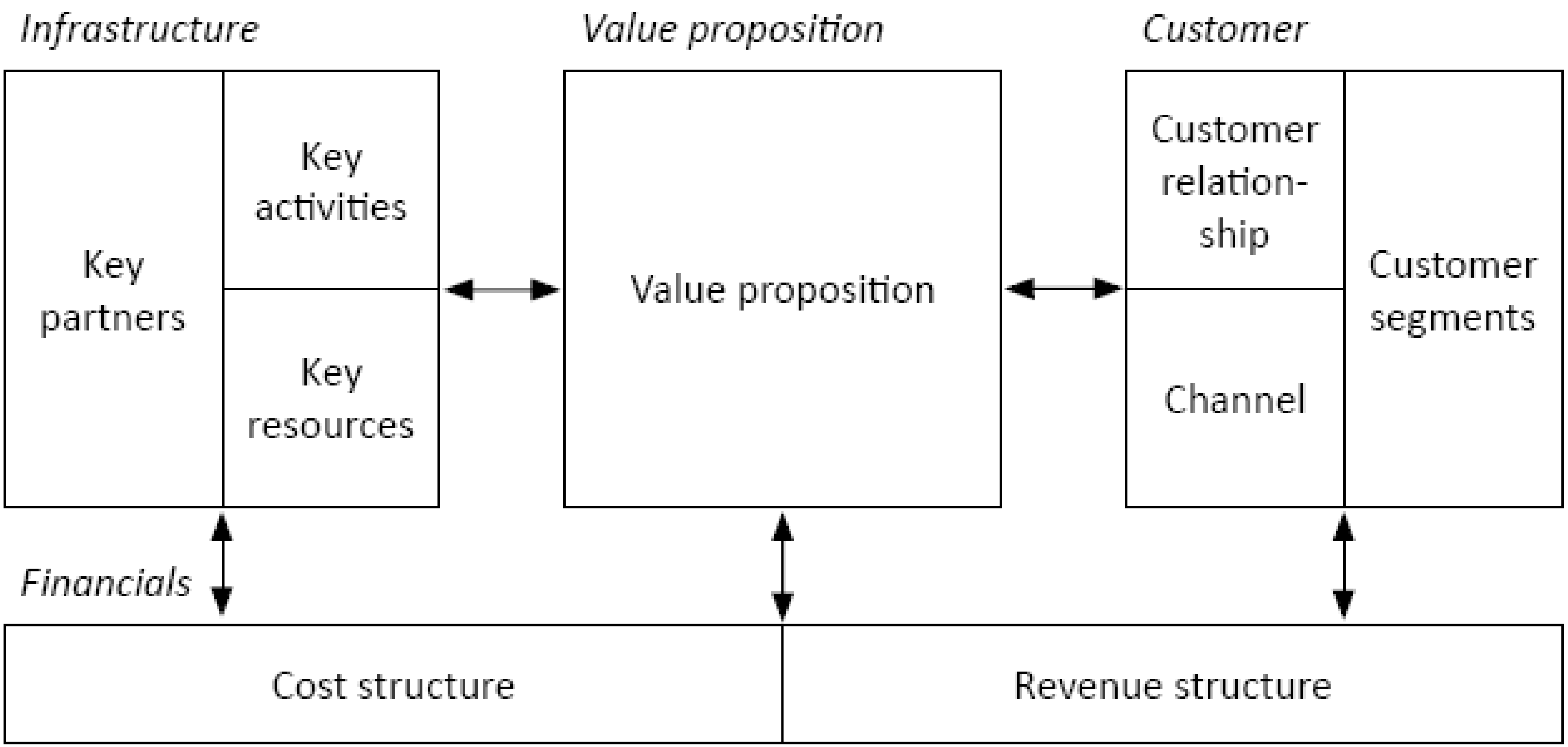
1. The network is reliable
2. Latency is zero
3. Bandwidth is infinite
4. The network is secure
5. Topology doesn't change
6. There is one administrator
7. Transport cost is zero
8. The network is homogeneous

Essentially everyone, when they first build a distributed application, makes the above eight assumptions. All prove to be false in the long run and all cause big trouble and painful learning experiences. (Peter Deutsch)

Connecting Users and Services

- Access and share (remote) resources
- Business Models and policies (see next slide)
- Collaboration by information exchange
- Communication (Convergence, VoIP)
- Groupware and virtual organizations
- Electronic and mobile commerce
- Sensor/actor networks in automation and pervasive computing (fine grained distribution)
- May compromise security (tamper proof HW) and privacy (tracking, spam)

Business Models



Quality of Service (QoS)

- QoS is a concept with which clients can indicate the level of service (SLA) they require

Examples:

- For real-time voice communication, the client prefers reliable delivery times over guaranteed delivery
- In financial applications, a client may prefer encrypted communication in favor of faster communication
- You can't have it all -> **Trade-offs!**

Transparency

- Concept: **Hide** different aspects of distribution from the client. It is the ultimate goal of many distributed systems.
- It can be achieved by providing lower-level (system) services (i.e. use another **layer**).
- The client uses these services instead of hardcoding the information.
- The **service layer provides** a service with a certain Quality of Service.

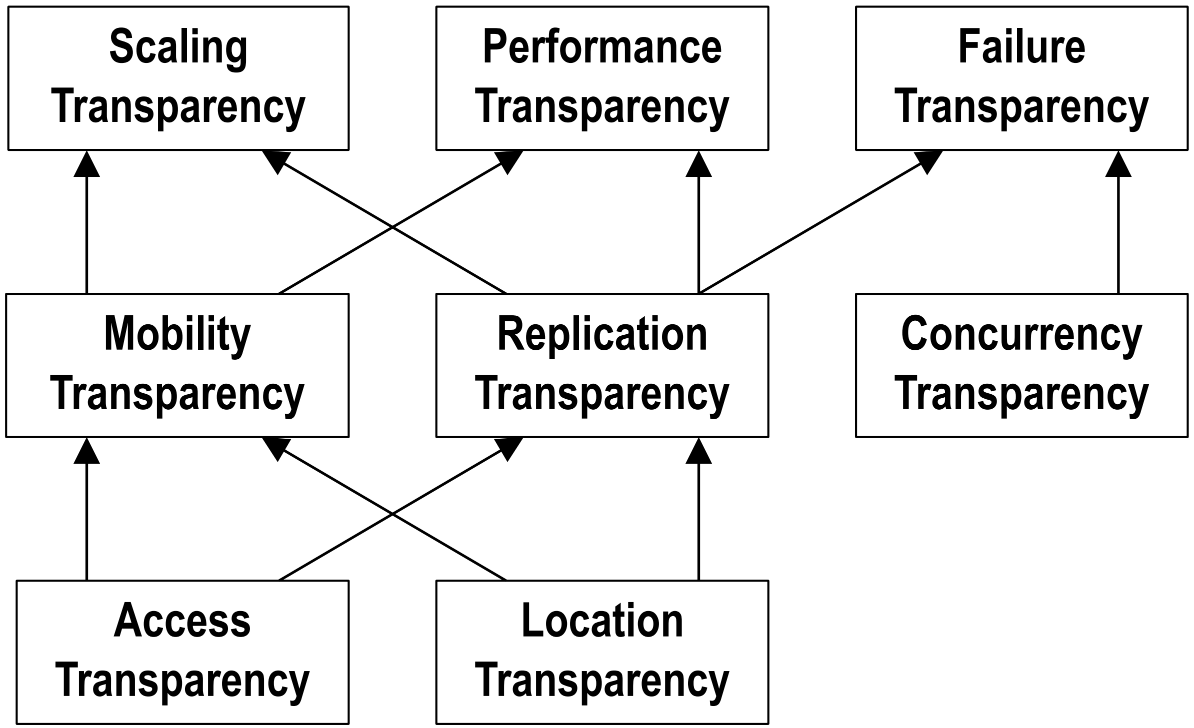
Transparency in a Distributed System

Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that a resource is replicated
Concurrency	Hide that a resource may be shared by several competitive users
Failure	Hide the failure and recovery of a resource

Different forms of transparency in a distributed system (ISO, 1995).

Transparency:
Information Hiding Applied to Distributed Systems

Transparency



Degree of Transparency

- Not blindly try to hide every aspect of distribution
- Performance transparency difficult (LAN/WAN)
- **Trade-off transparency/performance**
  - Failure masking
  - Replica consistency
- ➔ Transparency is an important goal, but has to be considered together with all other non-functional requirements and with respect to particular demands

Openness

- Offer services according to standard rules (syntax and semantics: format, contents, and meaning)
- Formalized in protocols
- Interfaces (IDL): semantics often informal

  Complete → Interoperability: Communication between processes

  Neutral → Portability: Different implementations of interface
- Flexibility: composition, configuration, replacement, extensibility (CBSE)

Separating Policy from Mechanism

- Granularity: objects vs. applications?
- Component interaction and composition standards (instead of closed/monolithic)
- E.g. Web browser provides facility to store cached documents, but caching policy can be plugged in arbitrarily (parameters or algorithmic).

Achieving openness

Web examples

- Different Web servers and Web browsers interoperate
- New browsers may be introduced to work with existing servers (and vice versa)
- Plugin interface allows new services to be added

Scalability

- A distributed system's **ability to grow** to meet increasing demands along several dimensions:
  1. Size (users and resources)
  2. Geographically (topologically)
  3. Administratively (independent organizations/domains)

- System remains effective
- System and application software should not need to change
- Trade-Off scalability/security

Scalability Challenges (size)

- **Controlling the cost of physical resources**: The quantity required should be O(n), i.e., the algorithm's performance is directly proportional to the size of the data set being processed
- **Controlling the performance loss**: In hierarchical system should be no worse than O(*log* n), i.e., the algorithm deals with a data set that is iteratively partitioned, like a balanced binary tree.
- **Preventing software resources running out**, but over-compensation may be even worse: Internet Addresses or Oracle7 2TB restriction
- **Avoiding performance bottlenecks** (centralized services, data, or algorithms)

Performance Bottlenecks

Concept	Example
Centralized services	A single server for all users
Centralized data	A single on-line telephone book, central DNS
Centralized algorithms	Doing routing based on complete information

Decentralized Algorithms - Principles

1. No machine has complete system state information
2. Machines make decisions based only on local (surrounding) information
3. Failure of one machine does not ruin the algorithm (no single point of failure)
4. No implicit assumption that a global clock exists

Geographical Scalability

- LAN:
  - Synchronous communication
  - Fast
  - Broadcast
  - Highly reliable
- WAN:
  - Asynchronous communication
  - Slow
  - Point to point (e.g. problems with location service)
  - Unreliable

Administrative Scalability

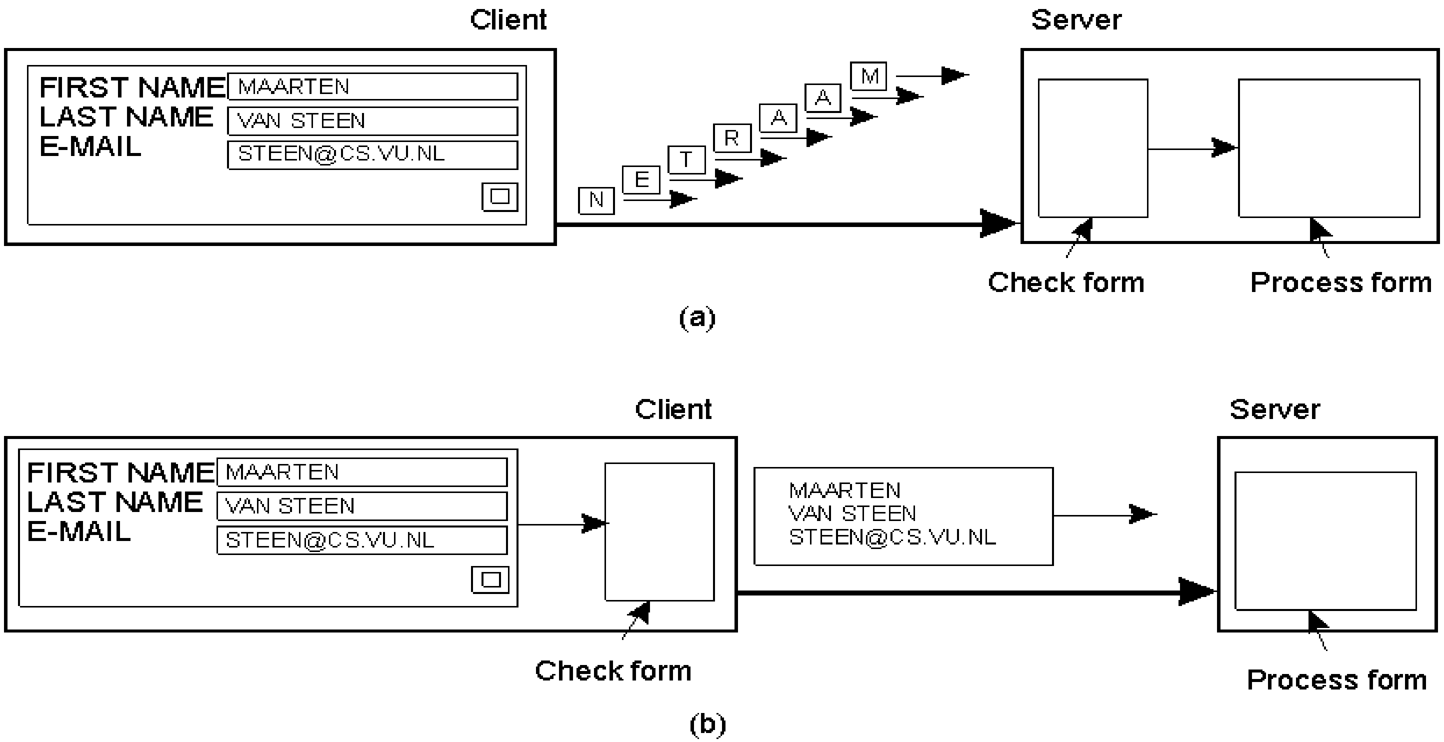
- e.g., Mobile Number portability, conflicting (orthogonal) policies:

  1. Resource usage
  2. Billing
  3. Management
  4. Security: Protection between the administrative domains – trusted domains – enforced limitations

Scaling Techniques

- **Hiding communication latencies**
  - Asynchronous communication (batch processing, parallel applications)
  - Reduce overall communication (HMI)
- **Distribution**
  - Hierarchies, domains, zones, … → split
- **Replication**
  - Availability, load balance, reduce communication
  - Caching: proximity, client decision
  - Consistency issues may adverse scalability!

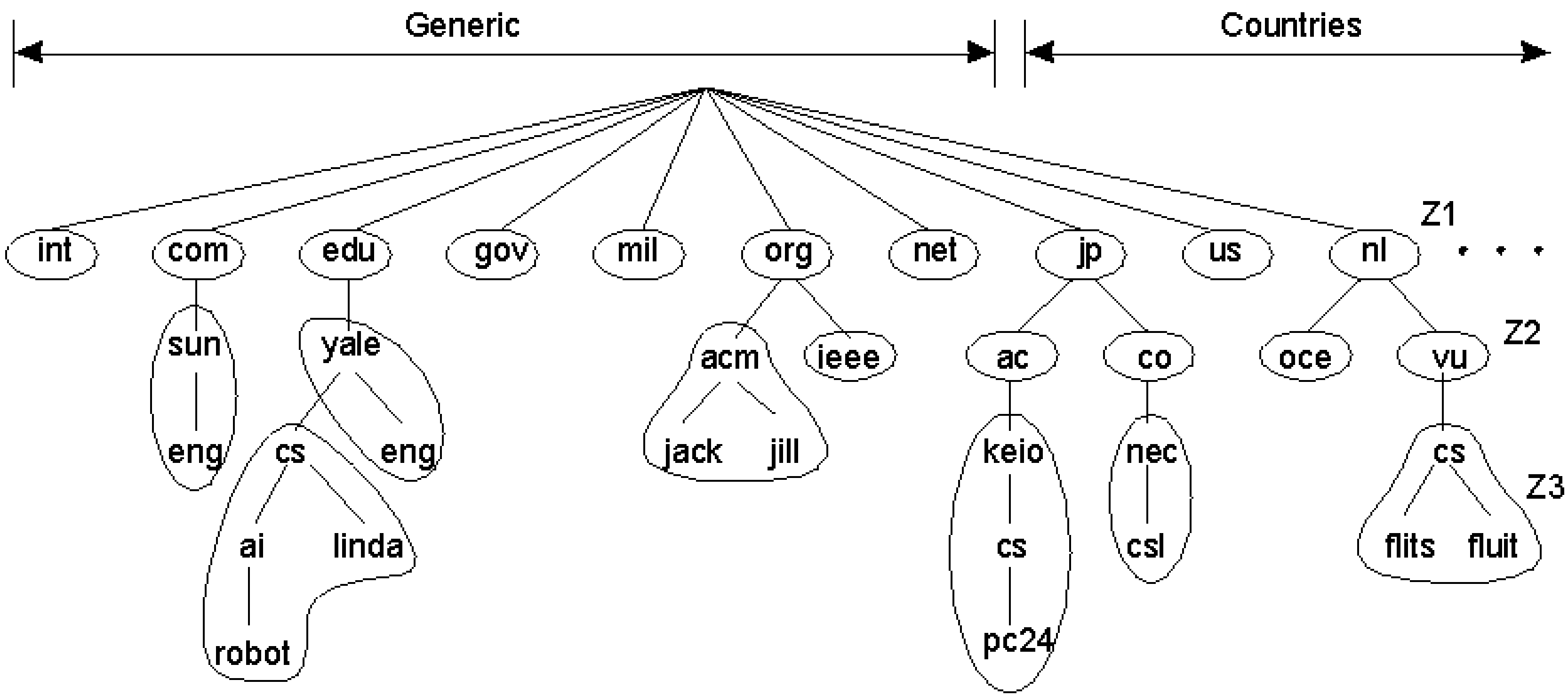
Scaling Techniques (2)



The difference between letting:

(a) a server or (b) a client check forms as they are being filled

Scaling Techniques (3)



ARCHITECTURAL STYLES

Dealing with complexity

- **Abstraction (and modeling)**
  - Client, server, service
  - Interface versus implementation
- **Information hiding (encapsulation)**
  - Interface design
- **Separation of concerns**
  - Layering (filesystem example: bytes, disc blocks, files)
  - Client and server
  - Components (granularity issues)

Communication models

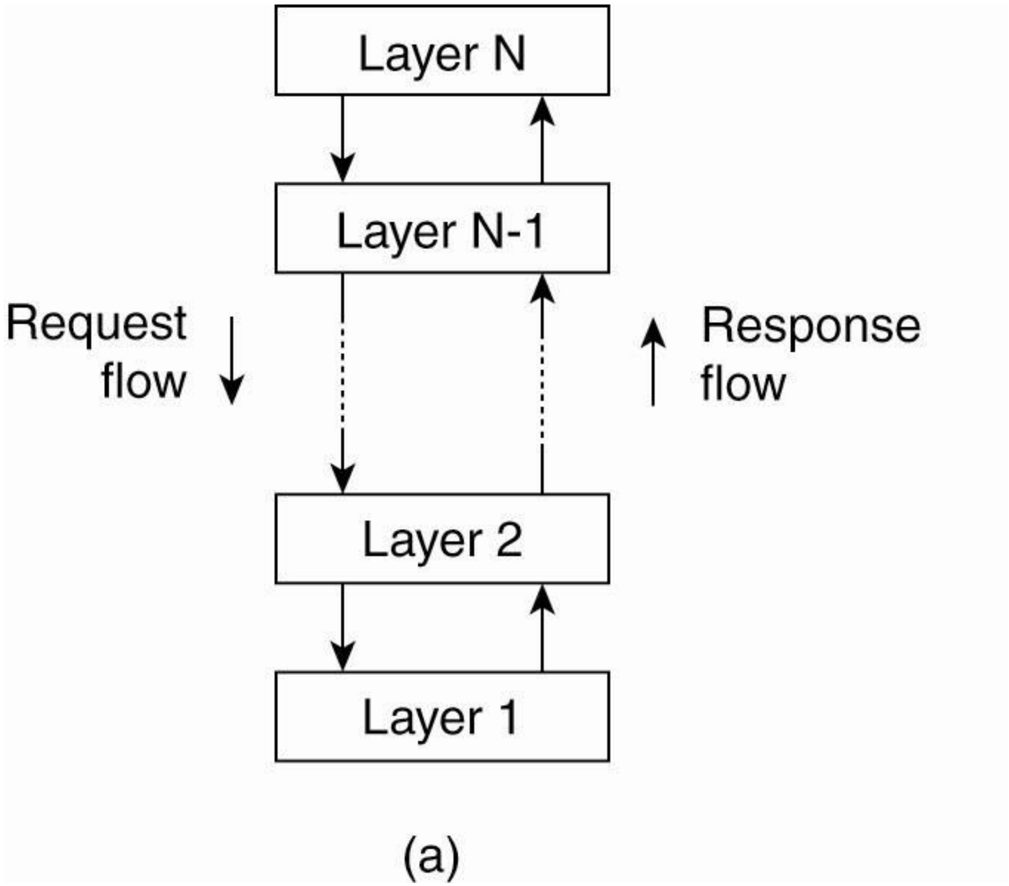
- Multiprocessors: shared memory (requires protection against concurrent access)
- Multicomputers: message passing
- Synchronization in shared memory
  - Semaphores (atomic mutex variable)
  - Monitors — an abstract data type whose operations may be invoked by concurrent threads; different invocations are synchronized
- Synchronization in multicomputers: blocking in message passing

Architectural Styles (1)

Important styles of architecture for distributed systems

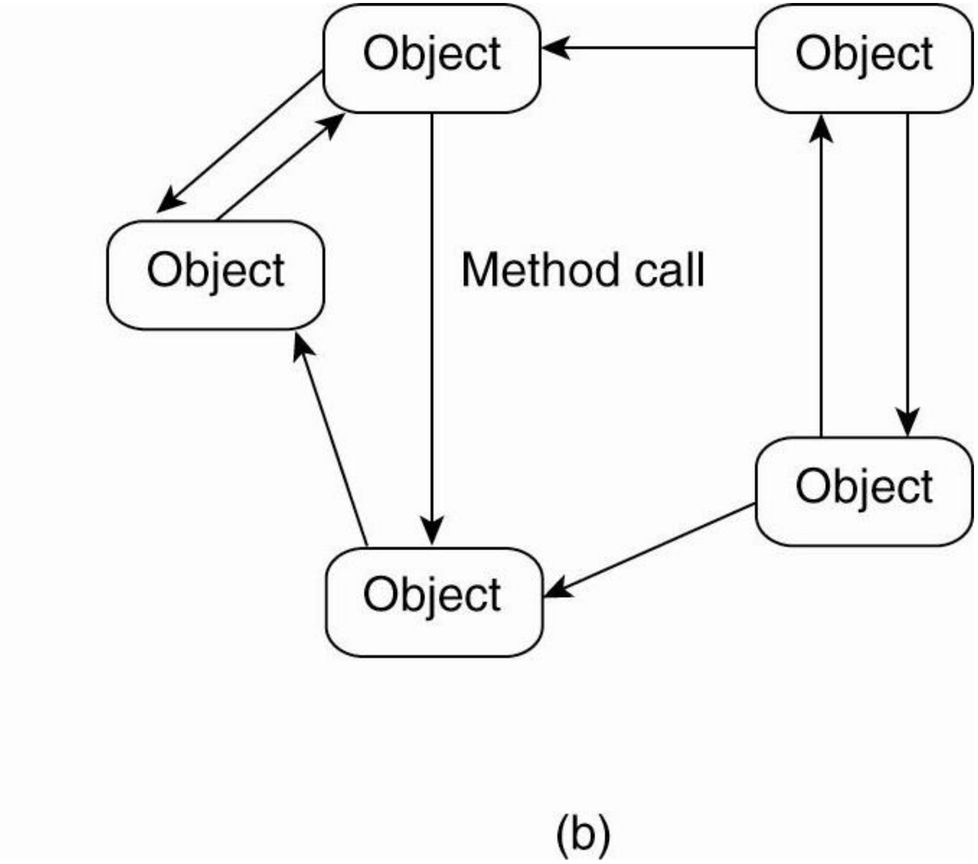
- Layered architectures
- Object-based architectures
- Data-centered architectures
- Event-based architectures

Architectural Styles (2)



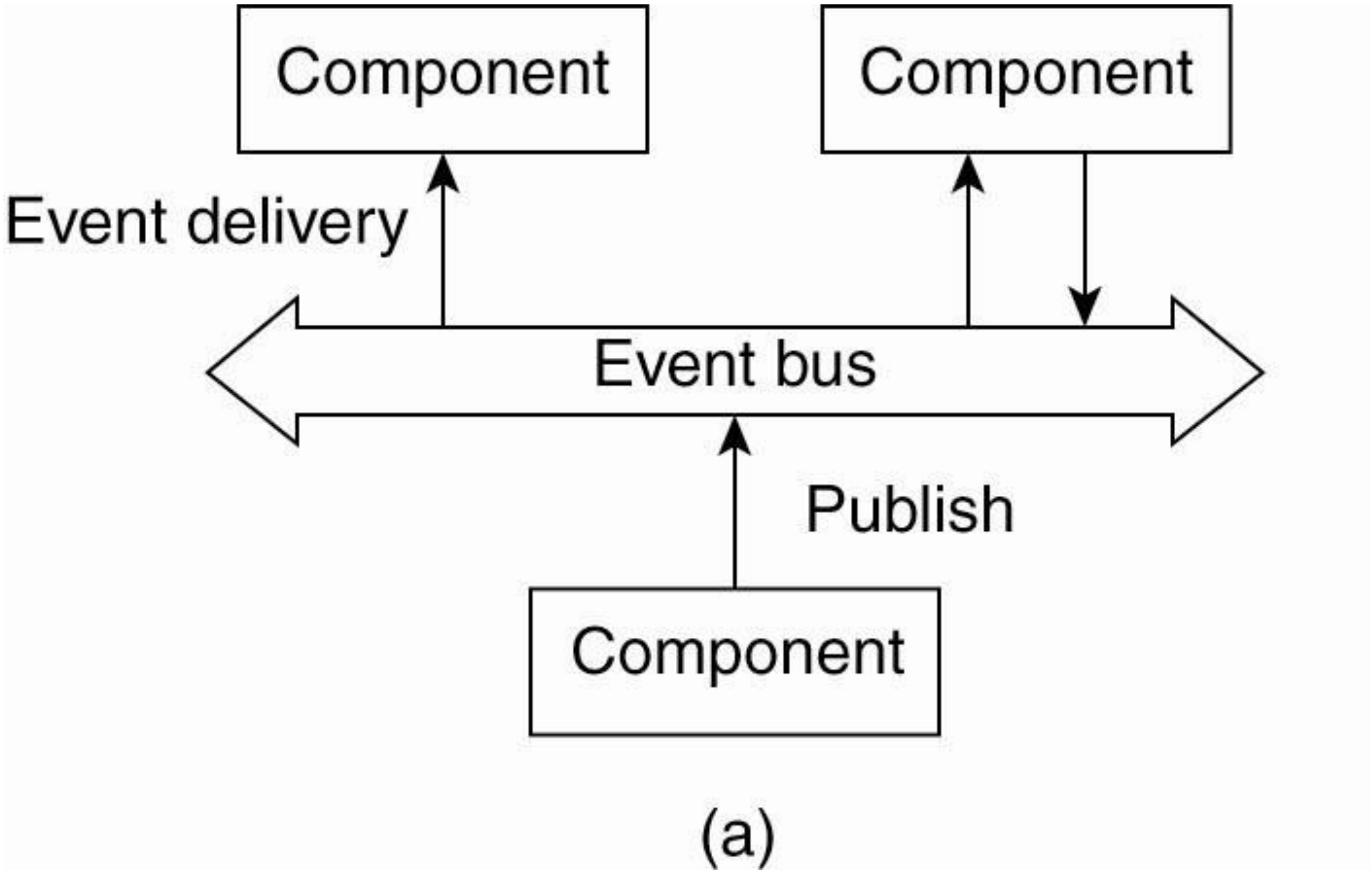
The **layered** architectural style

Architectural Styles (3)



The **object-based** architectural style

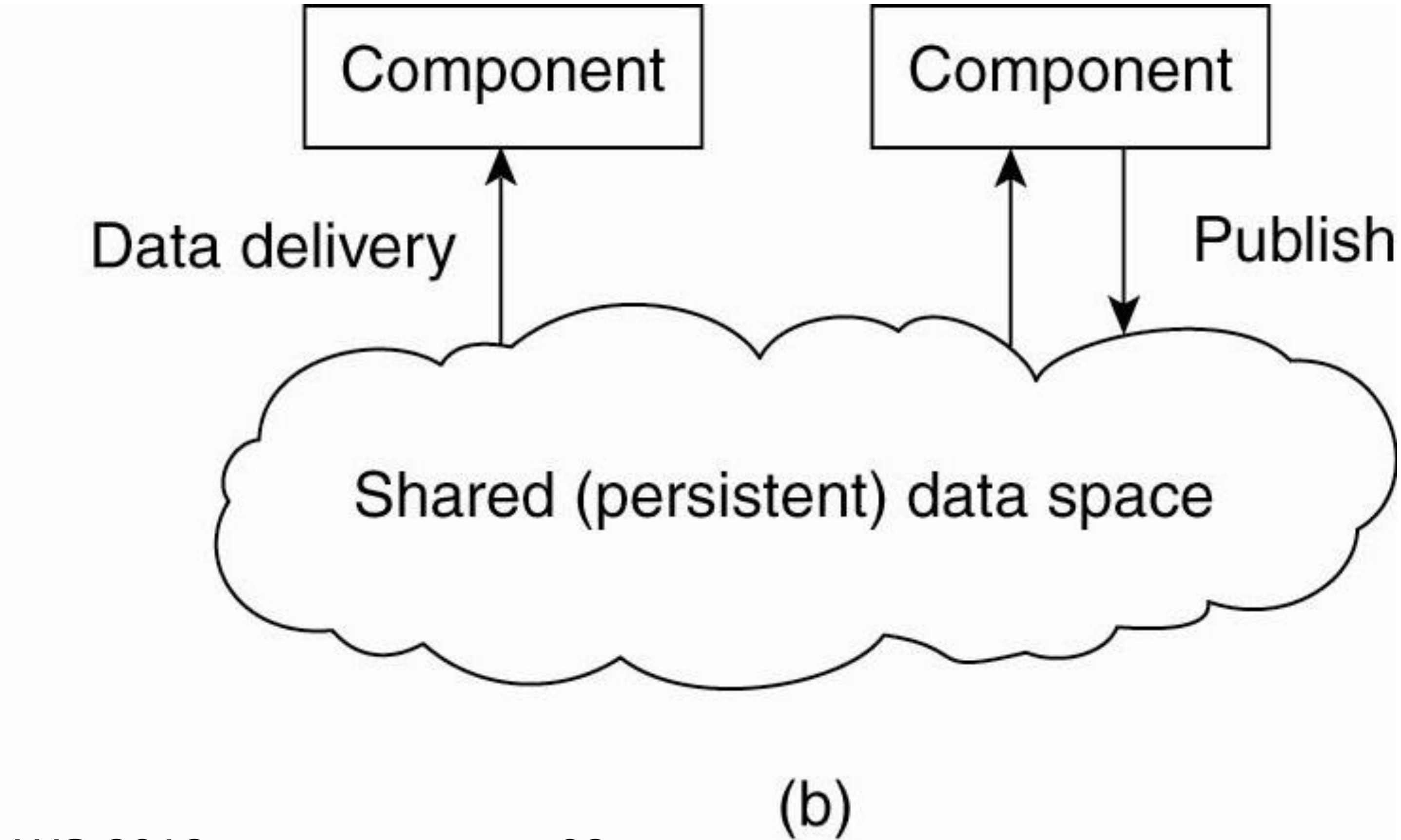
Architectural Styles (4)



The **event-based** architectural style

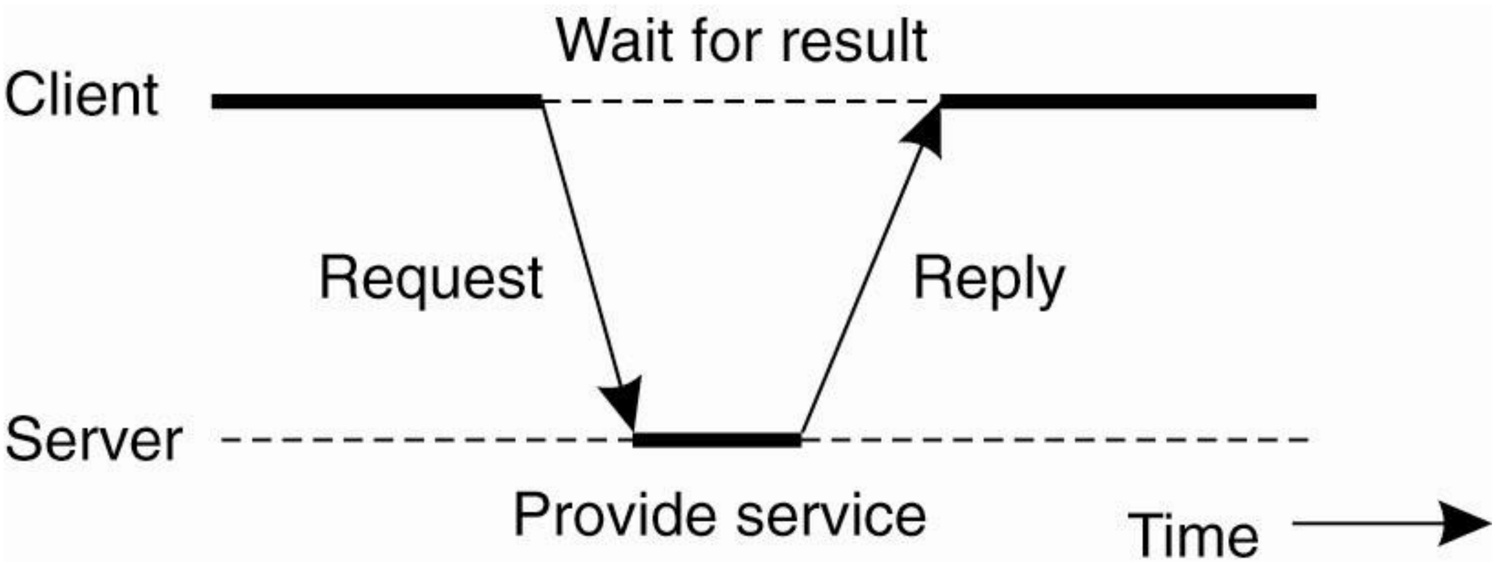
Architectural Styles (5)

The **shared data-space** architectural style.



Centralized Architectures

General interaction between a client and a server.



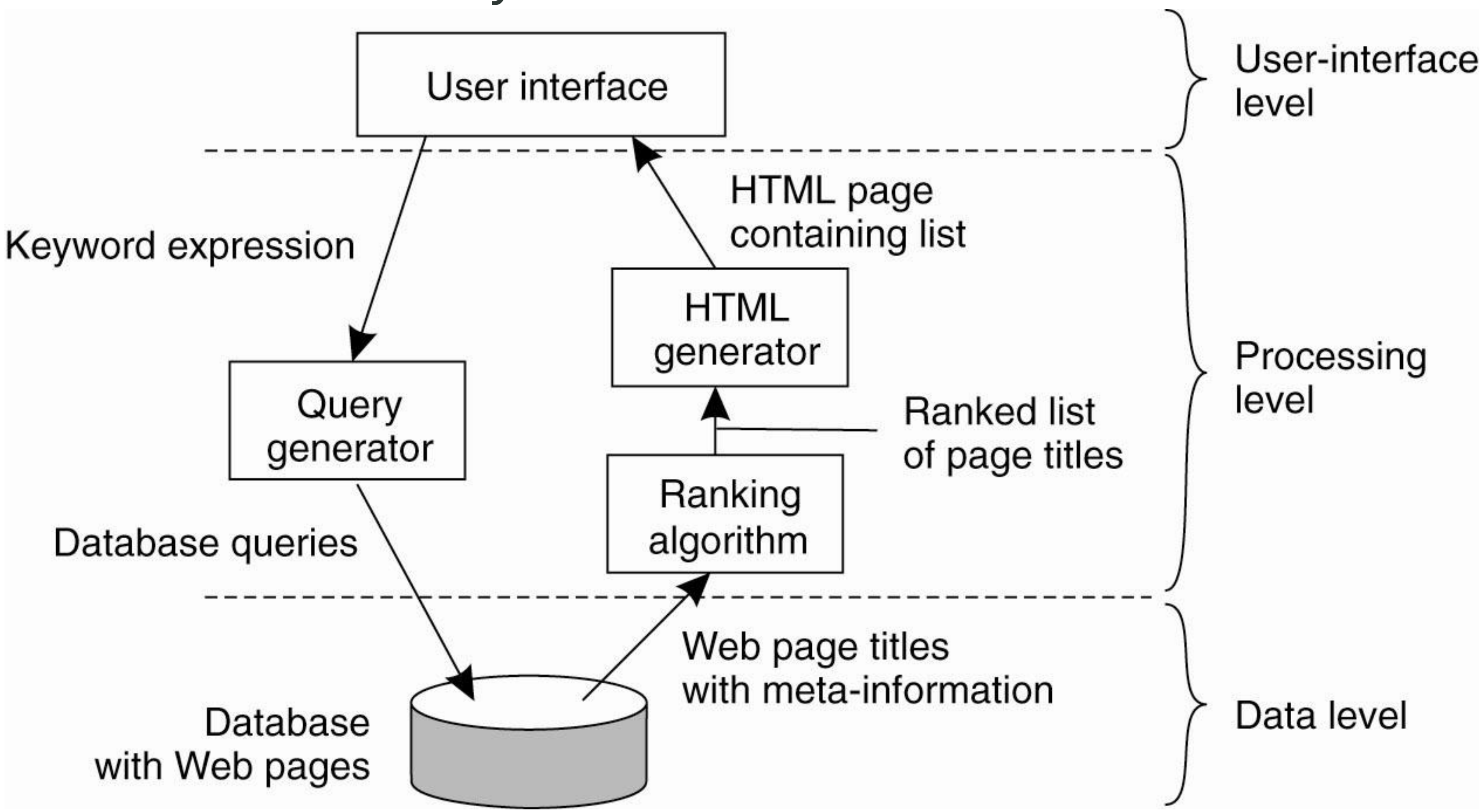
Application Layering (1)

Recall previously mentioned layers of architectural style

- The user-interface level
- The processing level
- The data level

Application Layering (2)

The simplified organization of an Internet search engine into three different layers.

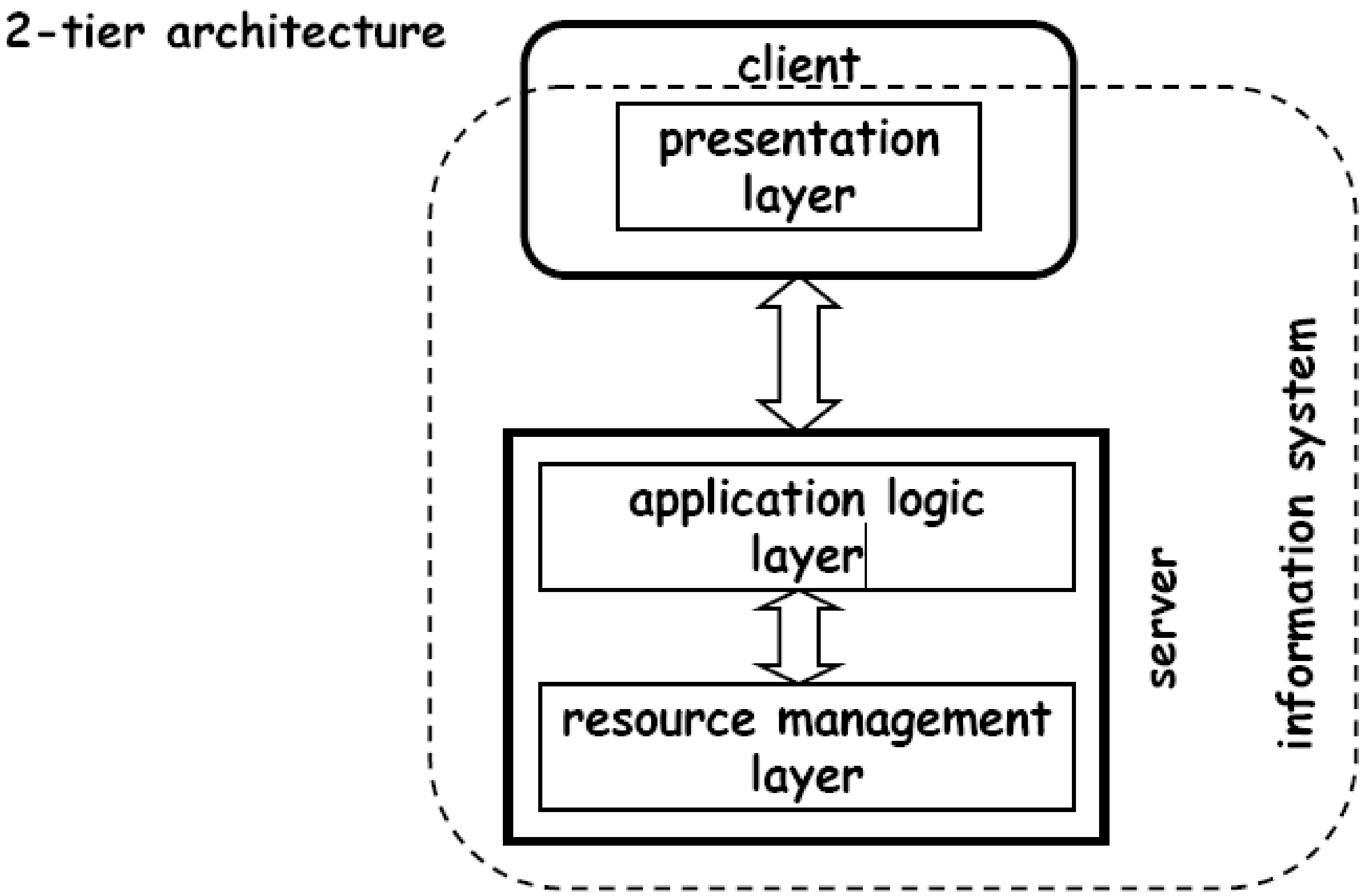


Multitiered Architectures (1)

The **simplest organization is to have only two types of machines**:

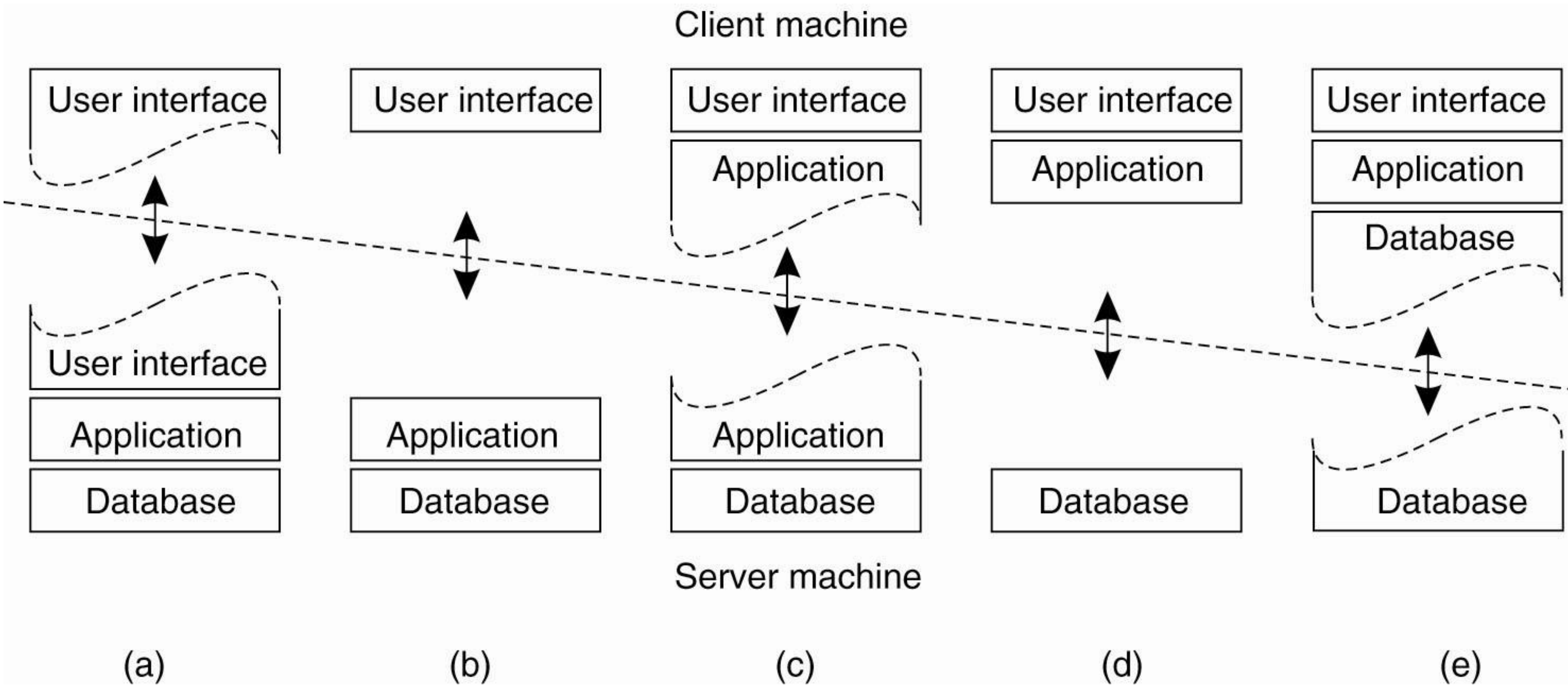
- A **client** machine containing only the programs implementing (part of) the user-interface level
- A **server** machine containing the rest, the programs implementing the processing and data level

Two-tier Architecture



Multitiered Architectures (2)

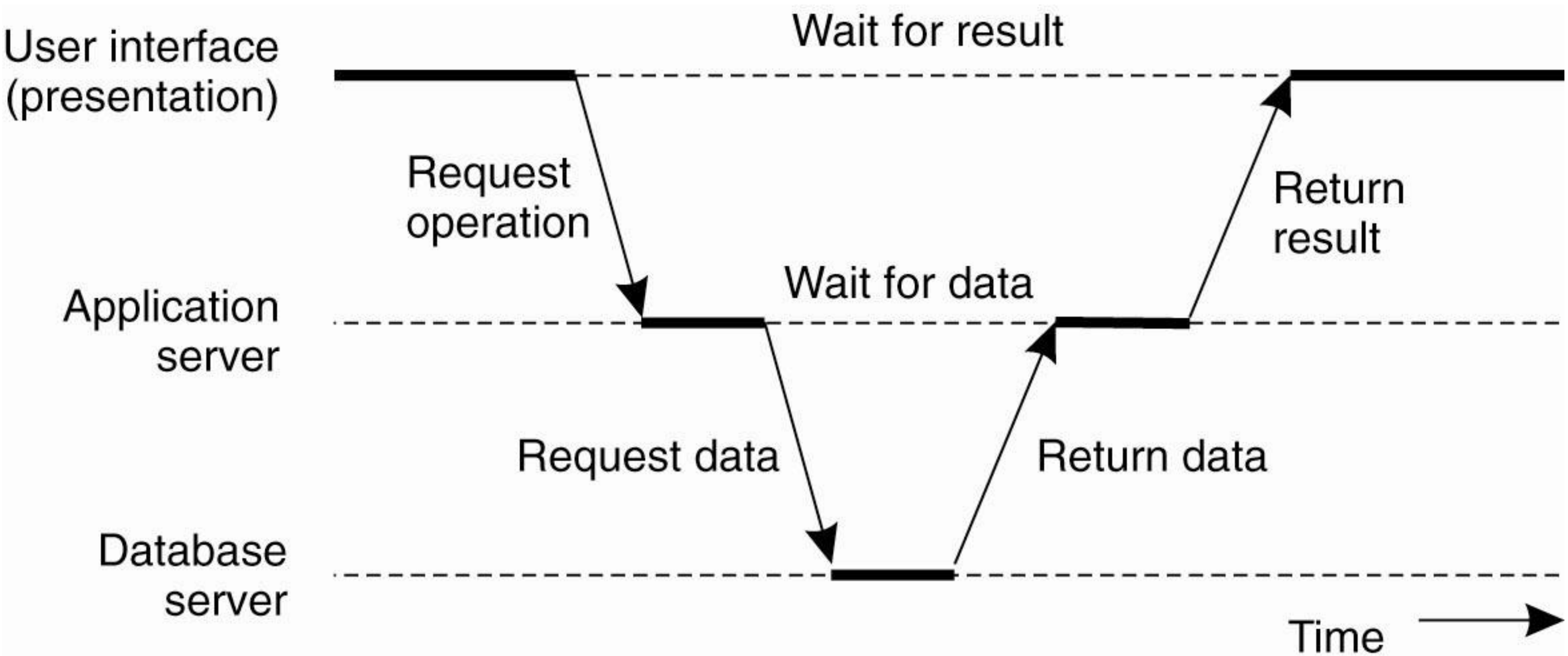
Vertical Distribution: Alternative client-server organizations (a)–(e).



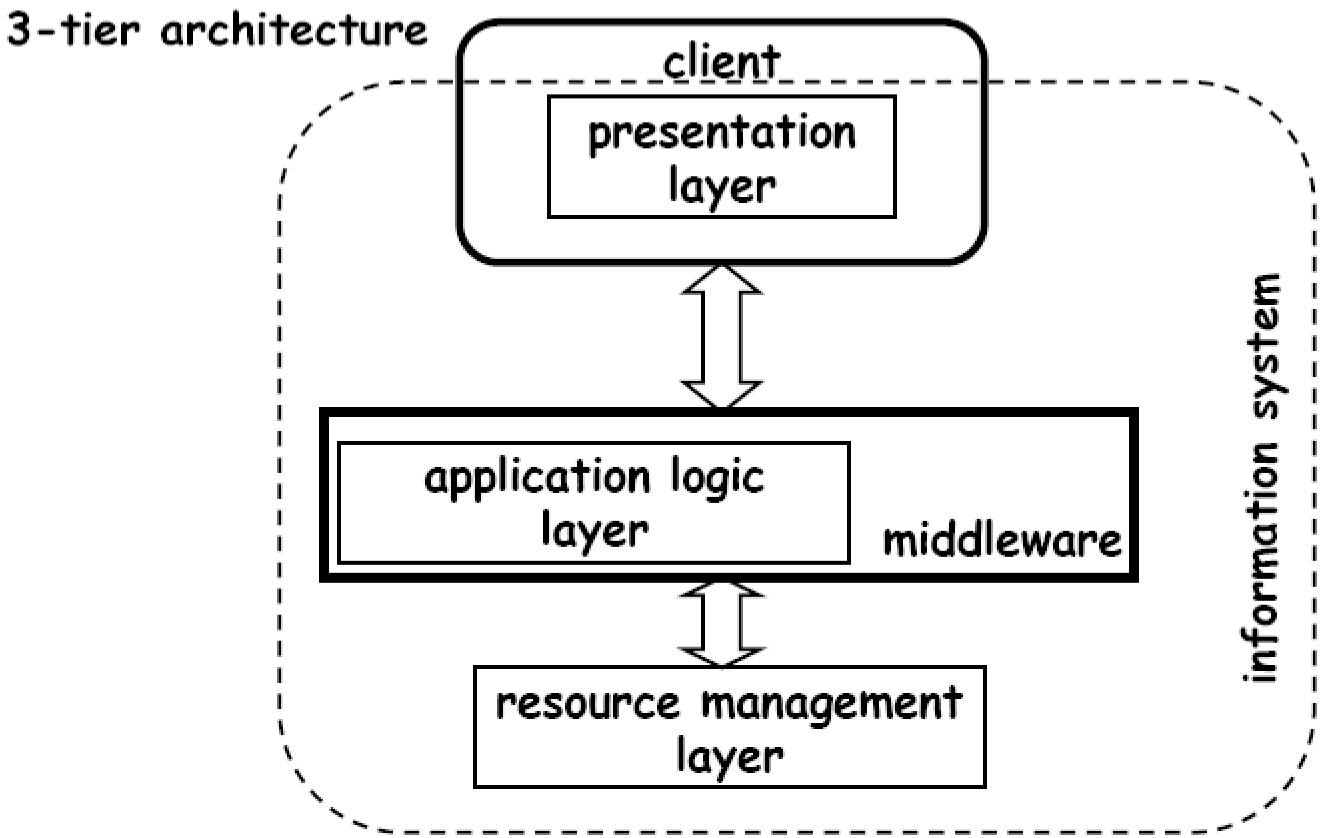
Distributed Systems Group

Multi-tiered Architectures (3)

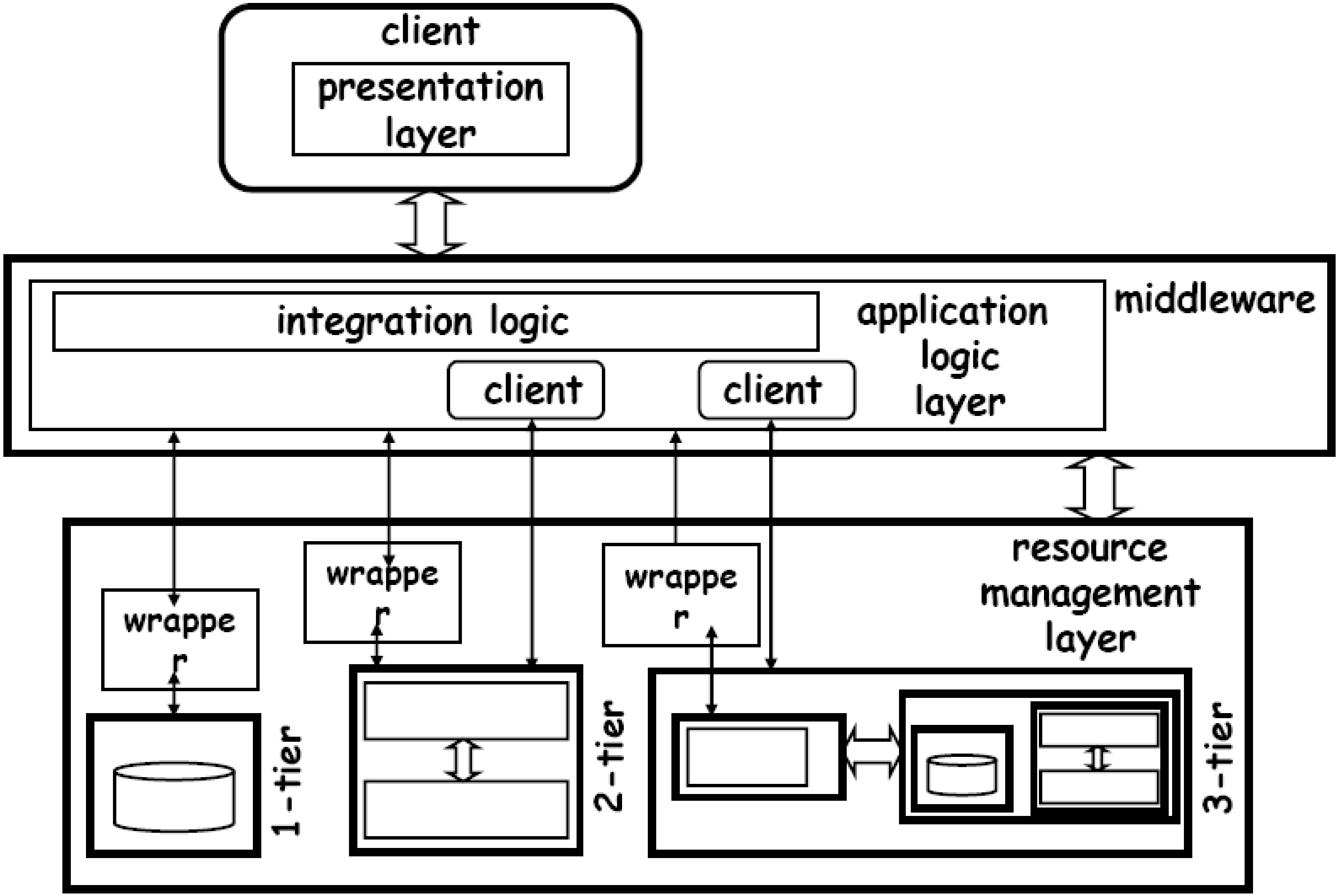
An example of a server acting as client.



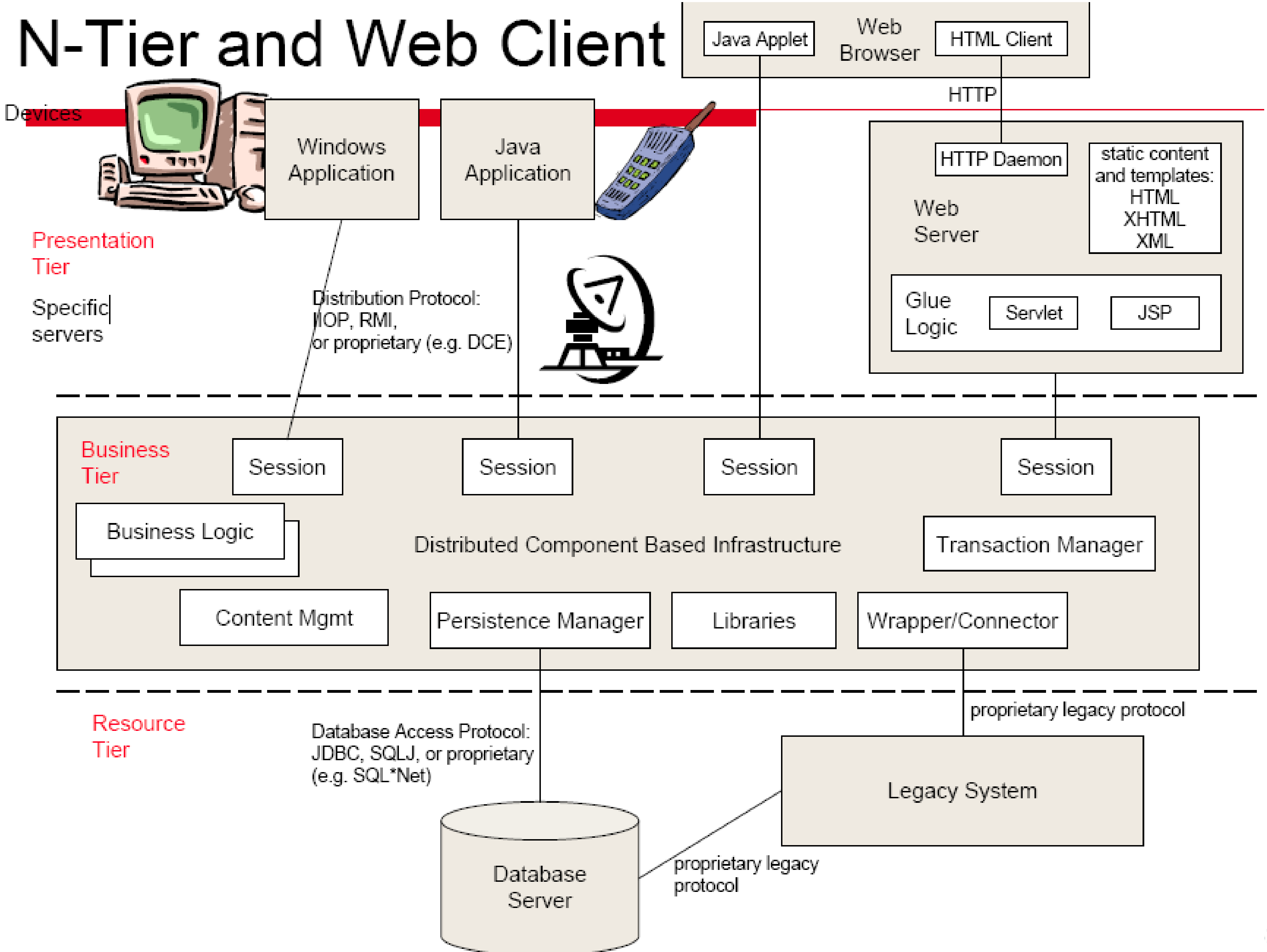
Three-tier Architecture



Application (System) Integration



N-Tier and Web Client



Summary

- A distributed system is a collection of computers working seamlessly together (single-system image – pro/con!)
- Distributed systems have evolved to be pervasive
- Principles and techniques are needed to cope with the complexity of distributed systems (openness, scalability, architectural styles, ...)
- Basic abstractions and concepts for distributed systems: client/server, layering (multitier), middleware, service, QoS, ...

Thanks for your attention

Prof.Dr. Schahram Dustdar
Distributed Systems Group
Vienna University of Technology

dustdar@dsg.tuwien.ac.at
dsg.tuwien.ac.at