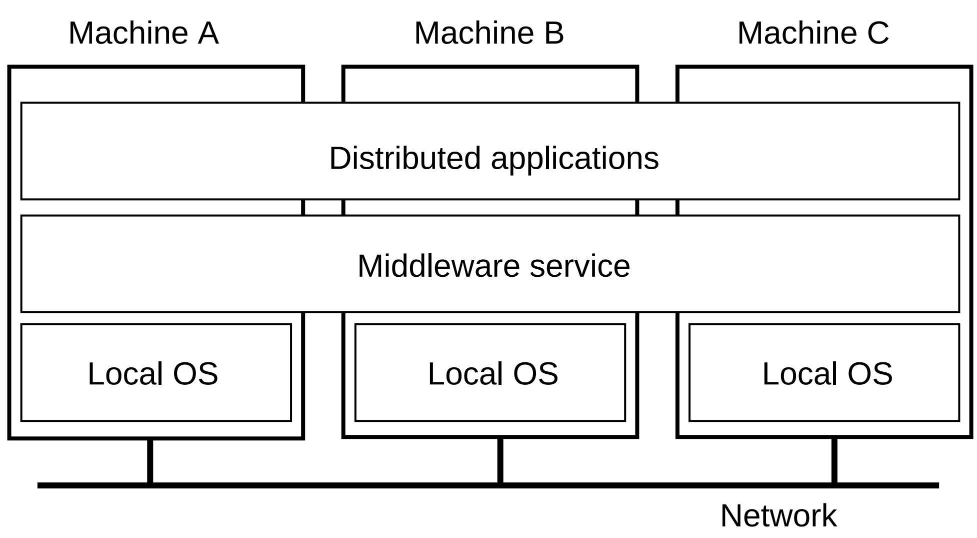
Distributed System: Definition

A distributed system is a piece of software that ensures that:

> *A collection of independent computers that appears to its users as a single coherent system*

Two aspects: (1) independent computers and
(2) single system ⇒ **middleware**.



Goals of Distributed Systems

- Connecting resources and users
- Distribution transparency
- Openness
- Scalability

Background

Developing Collaborative applications over a collection of mobile heterogeneous devices and data stores.

- Autonomous and mobile data stores
- Wireless (or wired) networks of various characteristics
- Devices of varying capabilities (pagers, cell phones, PDAs, PCs etc.)

Limitations of Current Technology

- Explicit and tedious (data and network) programming of applications on each device.
- Multiple types of heterogeneity of data stores
- Poor support for maintaining global consistency of data stores
- Poor Middleware support
    - Difficult peer-to-peer interaction (no data serving capabilities)
    - Poor or no support for dis-connectivity, location independence, group collaboration, atomic transaction, QoS

System on Mobile Devices (SyD)

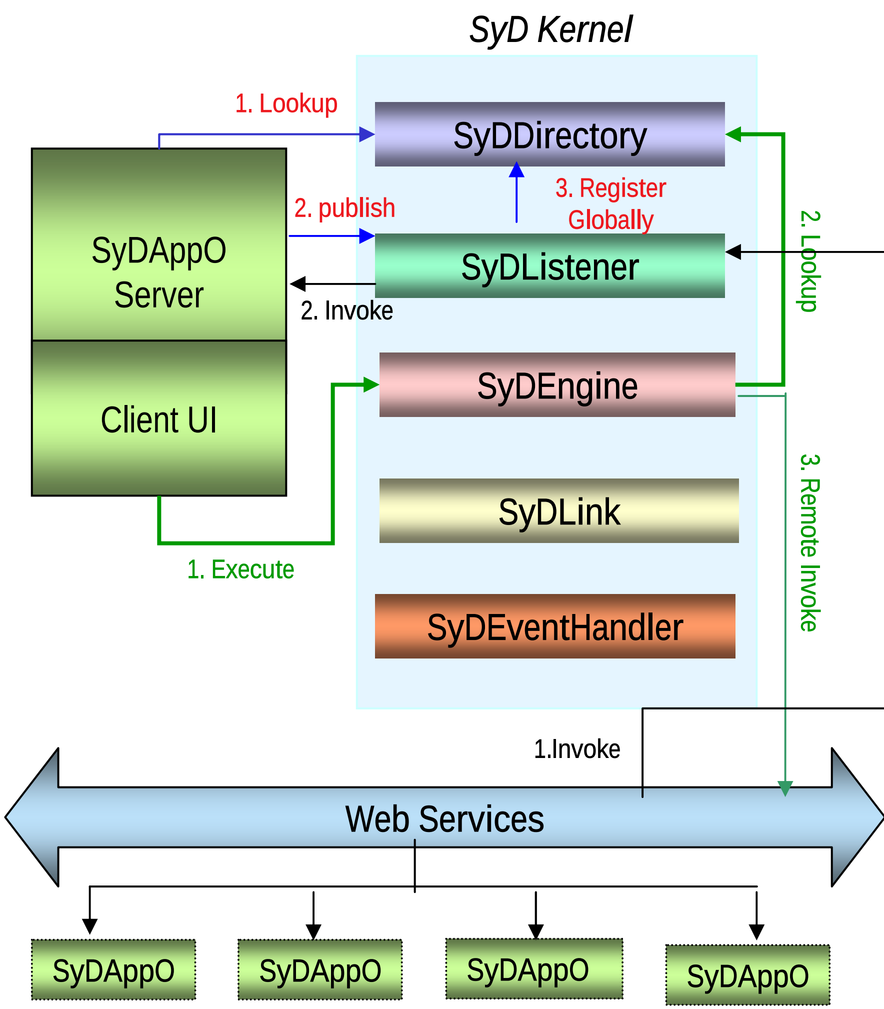
An Integrated Programming and Deployment Platform

- **Uniform Web Service view of device, data & network**
  - persistent object-view of mobile data and services.
- **Rapid development** of reliable and portable group applications
  - high-level programming and deployment environment
  - Leverage off existing server applications
- **Peer-to-peer and distributed applications**
- **Group** creation, maintenance, and manipulation
- **Quality of Service**, while handling mobility and dis-connectivity.
- **Footprint:** 112 KB, only **42 KB** device resident

GEORGIA

yamacraw futureforward

SyD Kernel Architecture and Interactions



- SyD Kernel modules developed in Java.
- SyD Directory provides user, group and service publishing, lookup service, and intelligent proxy management.
- SyD Listener sitting on device enables devices to act as servers by listening to remote invocation requests.
- SyD Engine allows users to execute services (can be group) remotely and aggregate results.
- SyD Event Handler handles local and global events.
- SyD Link enables an application to create and enforce interdependencies, constraints, and automatic updates among groups of SyD entities

Distribution Transparency

Transparency	**Description**
Access	Hides differences in data representation and invocation mechanisms
Location	Hides where an object resides
Migration	Hides from an object the ability of a system to change that object's location
Relocation	Hides from a client the ability of a system to change the location of an object to which the client is bound
Replication	Hides the fact that an object or its state may be replicated and that replicas reside at different locations
Concurrency	Hides the coordination of activities between objects to achieve consistency at a higher level
Failure	Hides failure and possible recovery of objects
Persistence	Hides the fact that an object may be (partly) passivated by the system

Note: Distribution transparency may be set as a goal, but achieving it is a different story.

Degree of Transparency

Observation: Aiming at full distribution transparency may be too much:

- Users may be located in different continents; distribution is apparent and not something you want to hide
- Completely hiding failures of networks and nodes is (theoretically and practically) impossible
  - You cannot distinguish a slow computer from a failing one
  - You can never be sure that a server actually performed an operation before a crash
- Full transparency will cost performance, exposing distribution of the system
  - Keeping Web caches *exactly* up-to-date with the master copy
  - Immediately flushing write operations to disk for fault tolerance

Openness of Distributed Systems

Open distributed system: Be able to interact with services from other open systems, irrespective of the underlying environment:

- Systems should conform to well-defined **interfaces**
- Systems should support **portability** of applications
- Systems should easily **interoperate**

Achieving openness: At least make the distributed system independent from **heterogeneity** of the underlying environment:

- Hardware
- Platforms
- Languages

Policies versus Mechanisms

Implementing openness: Requires support for different **policies** specified by applications and users:

- What level of consistency do we require for client-cached data?
- Which operations do we allow downloaded code to perform?
- Which QoS requirements do we adjust in the face of varying bandwidth?
- What level of secrecy do we require for communication?

Implementing openness: Ideally, a distributed system provides only **mechanisms**:

- Allow (dynamic) setting of caching policies, preferably per cachable item
- Support different levels of trust for mobile code
- Provide adjustable QoS parameters per data stream
- Offer different encryption algorithms

Scale in Distributed Systems

Observation: Many developers of modern distributed system easily use the adjective “scalable” without making clear *why* their system actually scales.

Scalability: At least three components:

- Number of users and/or processes **(size scalability)**
- Maximum distance between nodes **(geographical scalability)**
- Number of administrative domains **(administrative scalability)**

Most systems account only, to a certain extent, for size scalability. The (non)solution: powerful servers.

Today, the challenge lies in geographical and administrative scalability.

Techniques for Scaling

Distribution: Partition data and computations across multiple machines:

- Move computations to clients (Java applets)
- Decentralized naming services (DNS)
- Decentralized information systems (WWW)

Replication: Make copies of data available at different machines:

- Replicated file servers (mainly for fault tolerance)
- Replicated databases
- Mirrored Web sites
- Large-scale distributed shared memory systems

Caching: Allow client processes to access local copies:

- Web caches (browser/Web proxy)
- File caching (at server and client)

Scaling – The Problem

Observation: Applying scaling techniques is easy, except for one thing:

> *Having multiple copies (cached or replicated), leads to* ***inconsistencies****: modifying one copy makes that copy different from the rest.*
>
> *Always keeping copies consistent and in a general way requires* ***global synchronization*** *on each modification.*
>
> *Global synchronization precludes large-scale solutions.*

Observation: If we can tolerate inconsistencies, we may reduce the need for global synchronization.

Observation: Tolerating inconsistencies is application dependent.

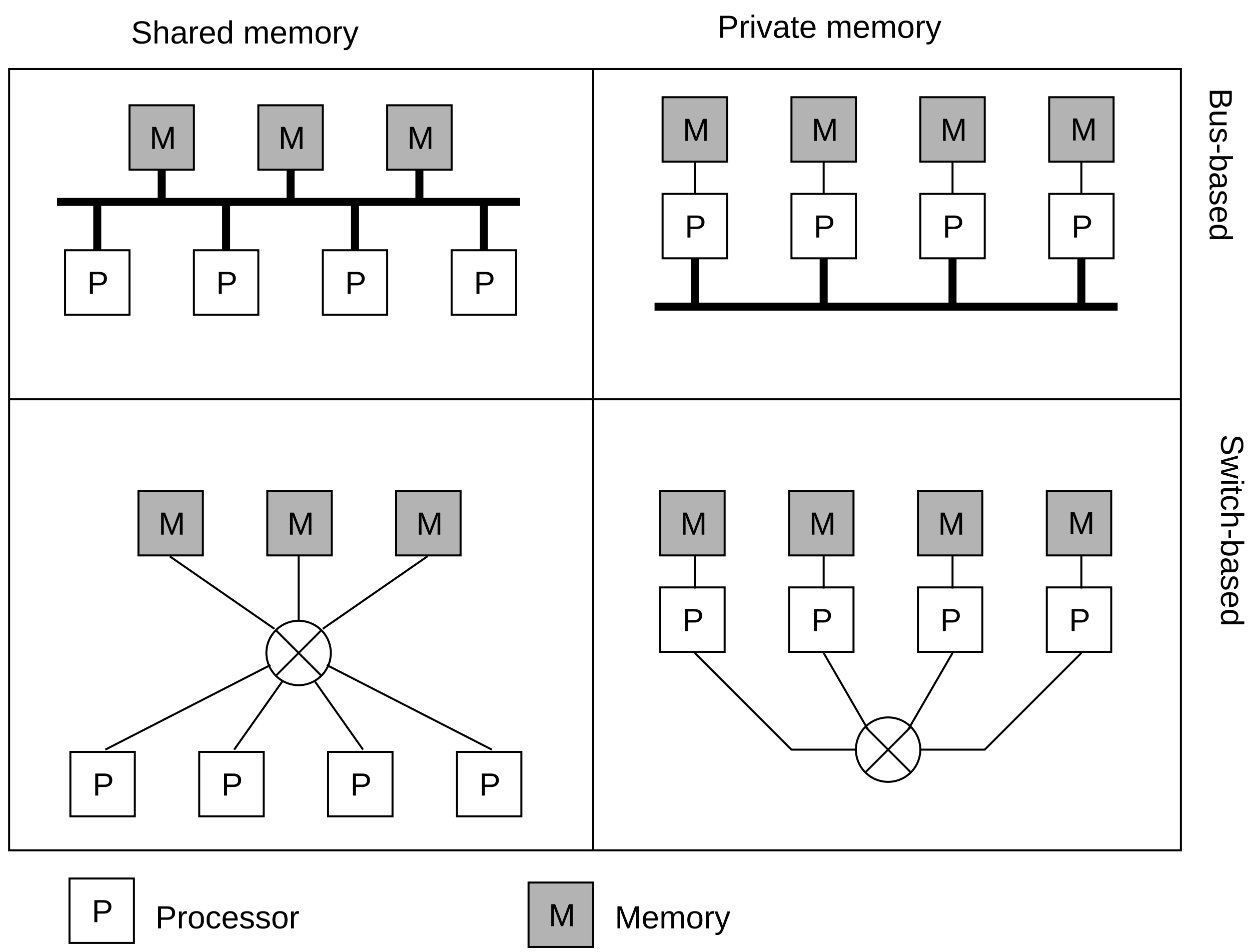
Distributed Systems: Hardware Concepts

- Multiprocessors
- Multicomputers
- Networks of Computers

Multiprocessors and Multicomputers

Distinguishing features:

- Private versus shared memory
- Bus versus switched interconnection



Networks of Computers

High degree of node heterogeneity:

- High-performance parallel systems (multiprocessors as well as multicomputers)
- High-end PCs and workstations (servers)
- Simple network computers (offer users only network access)
- Mobile computers (palmtops, laptops)
- Multimedia workstations

High degree of network heterogeneity:

- Local-area gigabit networks
- Wireless connections
- Long-haul, high-latency connections
- Wide-area switched megabit connections

Observation: Ideally, a distributed system hides these differences

Distributed Systems: Software Concepts

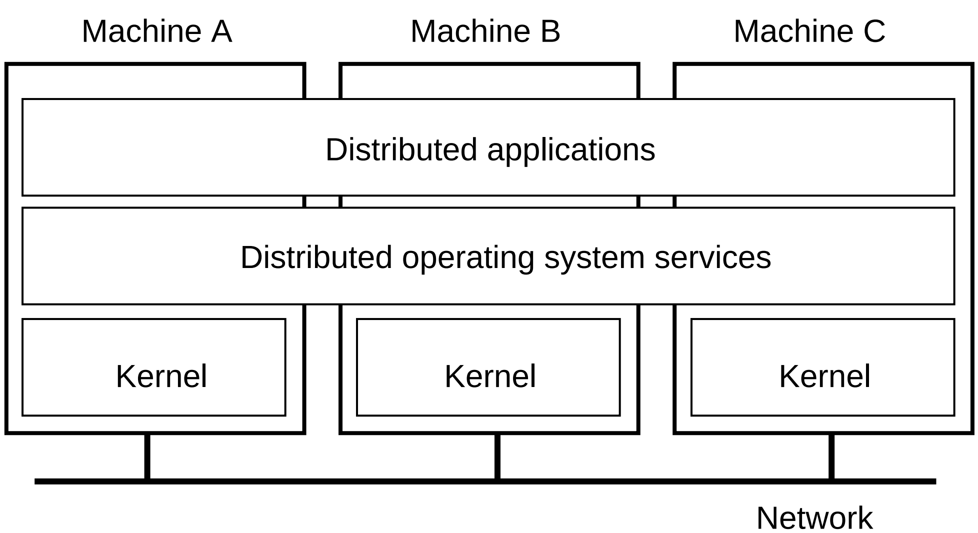
- Distributed operating system
- Network operating system
- Middleware

System	Description	Main goal
DOS	Tightly-coupled OS for multiprocessors and homogeneous multicomputers	Hide and manage hardware resources
NOS	Loosely-coupled OS for heterogeneous multicomputers (LAN and WAN)	Offer local services to remote clients
Middle-ware	Additional layer atop of NOS implementing general-purpose services	Provide distribution transparency

Distributed Operating System

Some characteristics:

- OS on each computer knows about the other computers
- OS on different computers generally the same
- Services are generally (transparently) distributed across computers



Multicomputer Operating System

Harder than traditional (multiprocessor) OS: Because memory is not shared, emphasis shifts to processor communication by message passing:

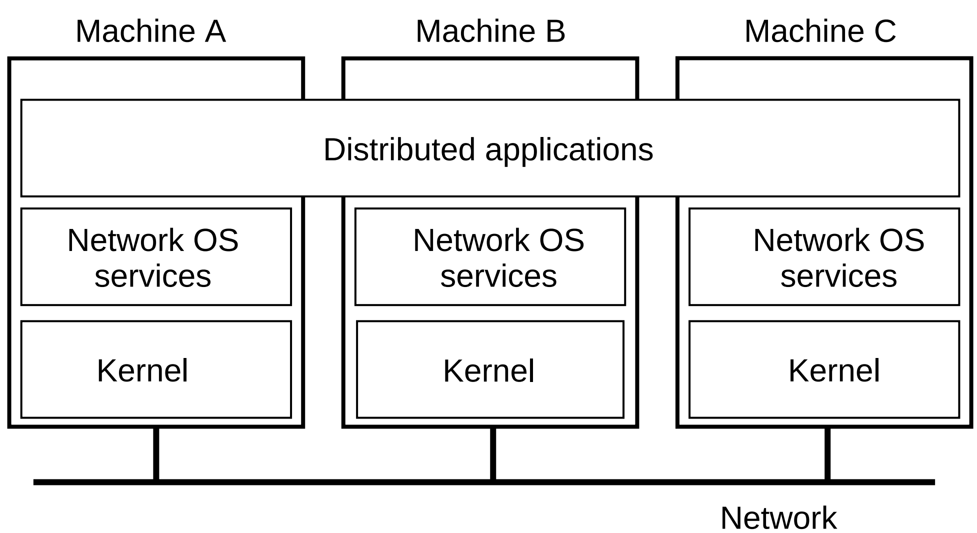
- Often no simple global communication:
  - Only bus-based multicomputers provide hardware broadcasting
  - Efficient broadcasting may require network interface programming techniques
- No simple systemwide synchronization mechanisms
- Virtual (distributed) shared memory requires OS to maintain global memory map in software
- Inherent distributed resource management: no central point where allocation decisions can be made

Practice: Only very few truly multicomputer operating systems exist (example: Amoeba)

Network Operating System

Some characteristics:

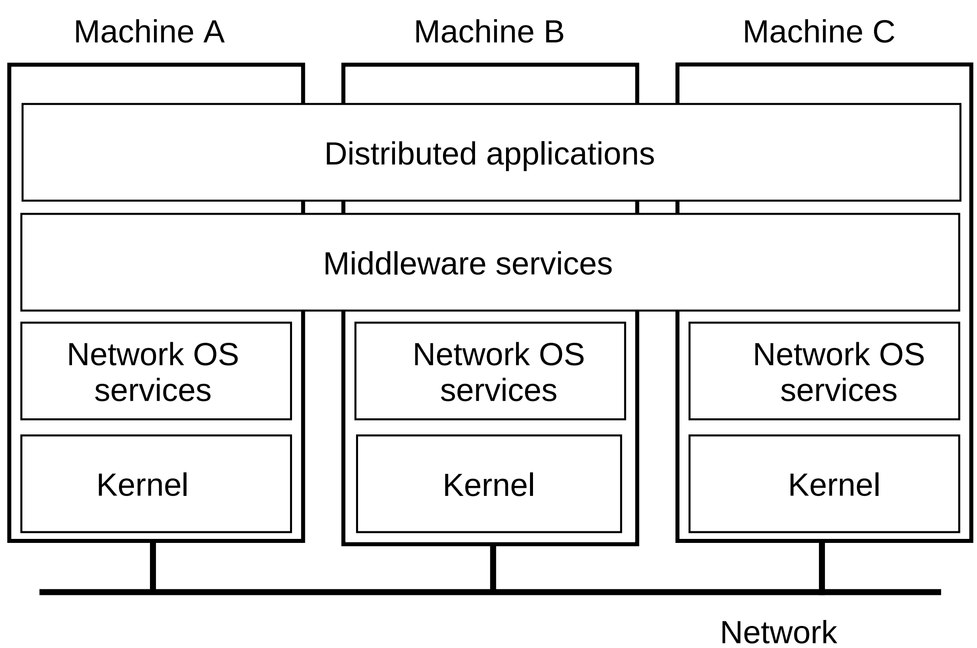
- Each computer has its own operating system with networking facilities
- Computers work independently (i.e., they may even have different operating systems)
- Services are tied to individual nodes (ftp, telnet, WWW)
- Highly file oriented (basically, processors share *only* files)



Distributed System (Middleware)

Some characteristics:

- OS on each computer need not know about the other computers
- OS on different computers need not generally be the same
- Services are generally (transparently) distributed across computers



Need for Middleware

Motivation: Too many networked applications were hard or difficult to integrate:

- Departments are running different NOSs
- Integration and interoperability only at level of primitive NOS services
- Need for federated information systems:
  - Combining different databases, but providing a single view to applications
  - Setting up enterprise-wide Internet services, making use of existing information systems
  - Allow transactions across different databases
  - Allow extensibility for future services (e.g., mobility, teleworking, collaborative applications)
- Constraint: use the existing operating systems, and treat them as the underlying environment (they provided the basic functionality anyway)

Middleware Services (1/2)

Communication services: Abandon primitive socket-based message passing in favor of:

- Procedure calls across networks
- Remote-object method invocation
- Message-queuing systems
- Advanced communication streams
- Event notification service

Information system services: Services that help manage data in a distributed system:

- Large-scale, systemwide naming services
- Advanced directory services (search engines)
- Location services for tracking mobile objects
- Persistent storage facilities
- Data caching and replication

Middleware Services (2/2)

Control services: Services giving applications control over when, where, and how they access data:

- Distributed transaction processing
- Code migration

Security services: Services for secure processing and communication:

- Authentication and authorization services
- Simple encryption services
- Auditing service

Comparison of DOS, NOS, and Middleware

1: Degree of transparency
2: Same operating system on each node?
3: Number of copies of the operating system
4: Basis for communication
5: How are resources managed?
6: Is the system easy to scale?
7: How open is the system?

Item	Distributed OS		Network OS	Middleware DS
	multiproc.	multicomp.		
1	Very High	High	Low	High
2	Yes	Yes	No	No
3	1	N	N	N
4	Shared memory	Messages	Files	Model specific
5	Global, central	Global, distributed	Per node	Per node
6	No	Moderately	Yes	Varies
7	Closed	Closed	Open	Open

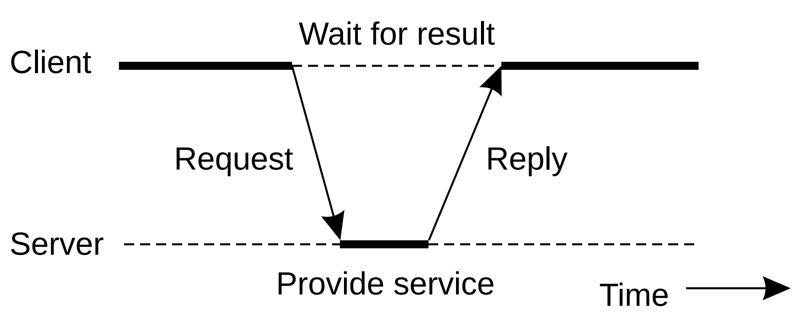
Client–Server Model

- Basic model
- Application layering
- Client–Server architectures

Basic Client–Server Model (1/2)

Characteristics:

- There are processes offering services (**servers**)
- There are processes that use services (**clients**)
- Clients and servers can be distributed across different machines
- Clients follow request/reply model with respect to using services



Basic Client–Server Model (2/2)

Servers: Generally provide services related to a *shared resource*:

- Servers for file systems, databases, implementation repositories, etc.
- Servers for shared, linked documents (Web, Lotus Notes)
- Servers for shared applications
- Servers for shared distributed objects

Clients: Allow remote service access:

- Programming interface transforming client's local service calls to request/reply messages
- Devices with (relatively simple) digital components (barcode readers, teller machines, hand-held phones)
- Computers providing independent user interfaces for specific services
- Computers providing an integrated user interface for related services (compound documents)

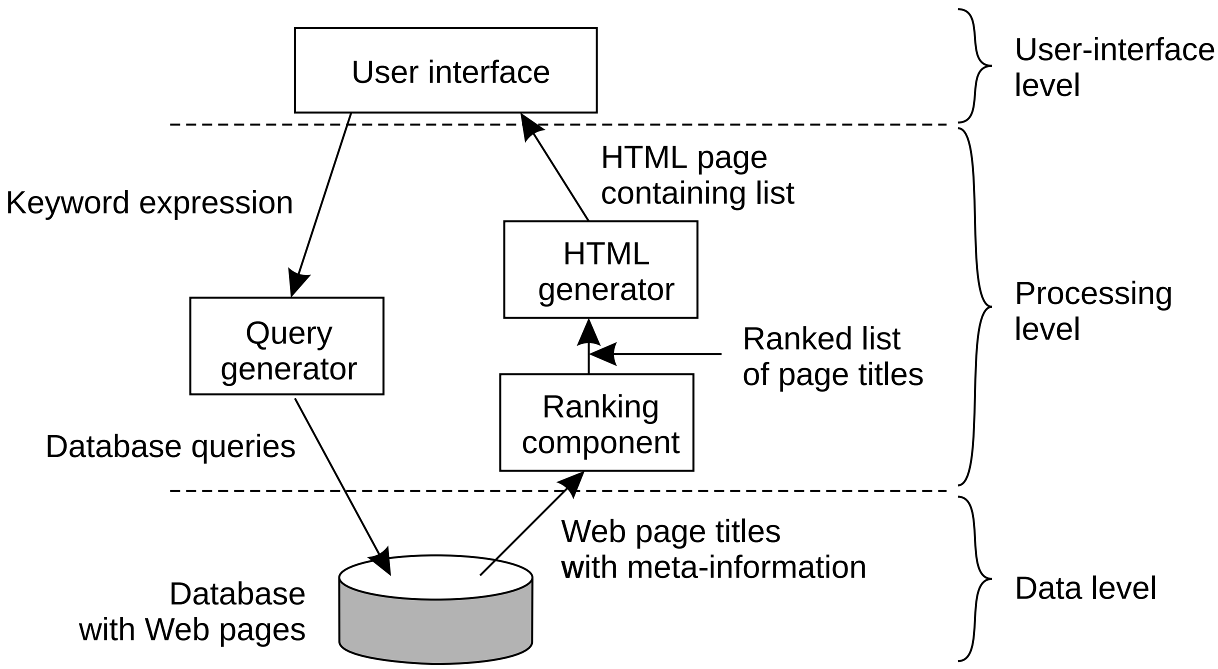
Application Layering (1/2)

Traditional three-layered view:

- User-interface layer contains units for an application's user interface
- Processing layer contains the functions of an application, i.e. without specific data
- Data layer contains the data that a client wants to manipulate through the application components

Observation: This layering is found in many distributed information systems, using traditional database technology and accompanying applications.

Application Layering (2/2)



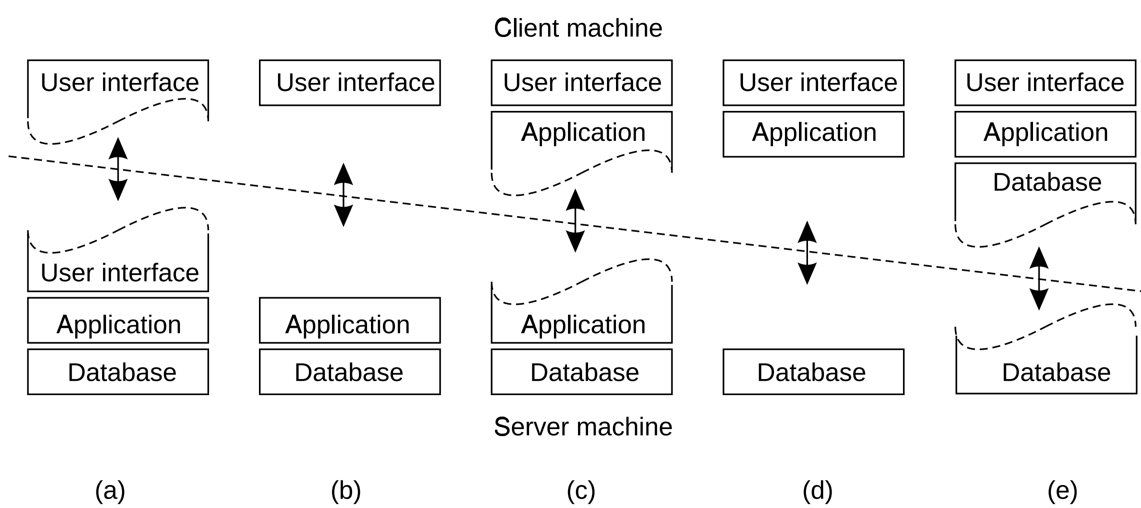
Client-Server Architectures

Single-tiered: dumb terminal/mainframe configuration

Two-tiered: client/single server configuration

Three-tiered: each layer on separate machine

Traditional two-tiered configurations:



Alternative C/S Architectures (1/2)

Observation: Multi-tiered architectures seem to constitute buzzwords that fail to capture many modern client–server organizations.

Cooperating servers: Service is physically distributed across a collection of servers:

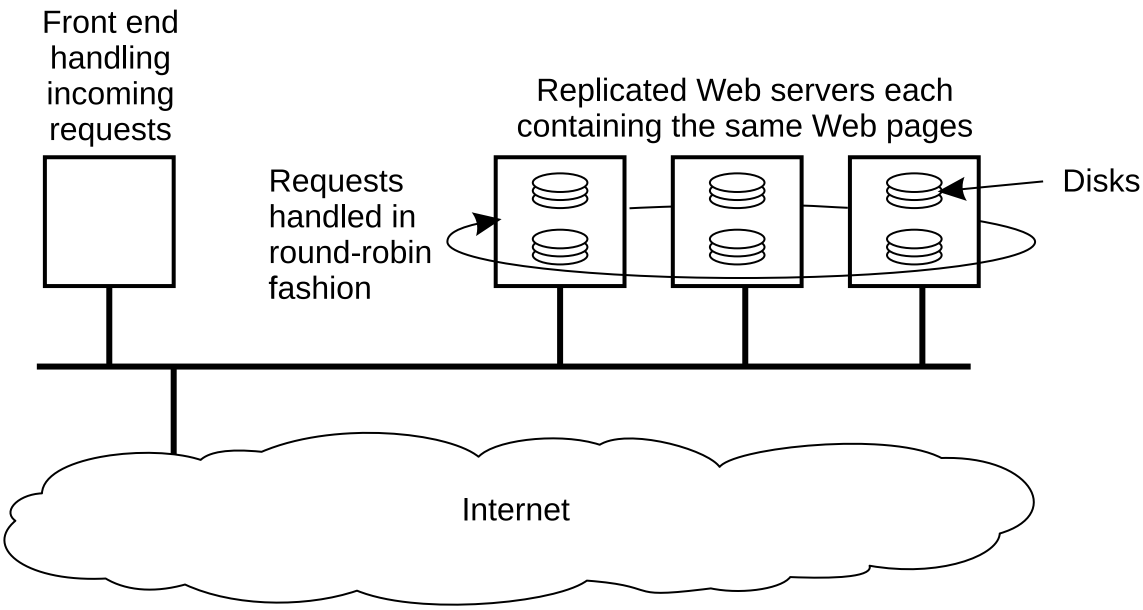
- Traditional multi-tiered architectures
- Replicated file systems
- Network news services
- Large-scale naming systems (DNS, X.500)
- Workflow systems
- Financial brokerage systems

Cooperating clients: Distributed application exists by virtue of client collaboration:

- Teleconferencing where each client owns a (multimedia) workstation
- Publish/subscribe architectures in which role of client and server is blurred

Alternative C/S Architectures (2/2)

Essence: Make distinction between vertical and horizontal distribution



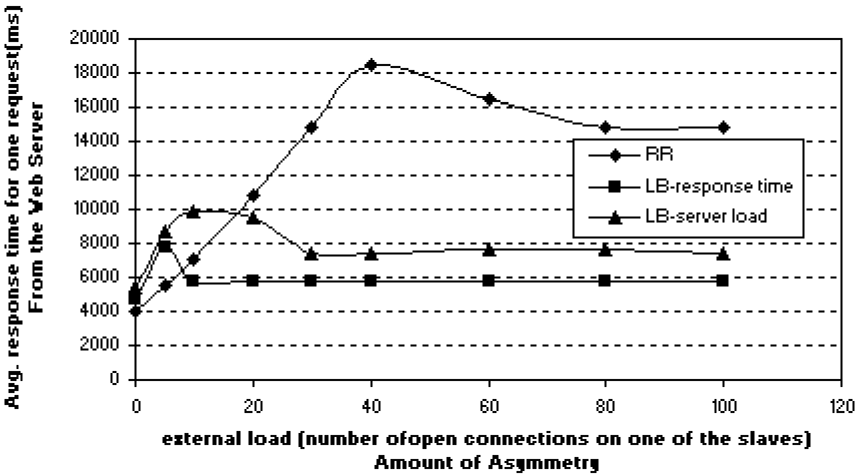


Figure 5: Comparison with asymmetric load on the application servers

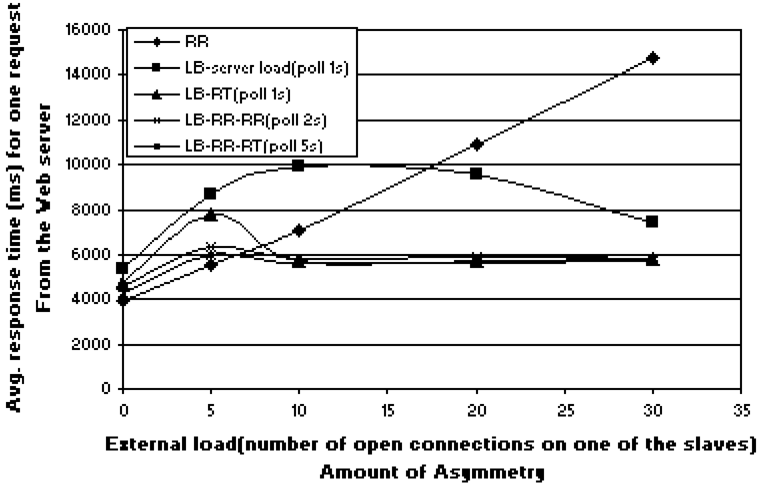


Figure 9: Comparison of all the algorithms