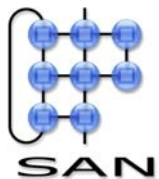


Concepts of Distributed Systems 2006/2007

Introduction & overview

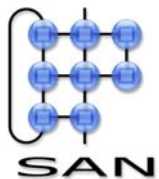
Johan Lukkien



Programme

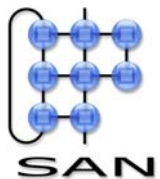
- Introduction & overview
- Communication
- Distributed OS & Processes
- Synchronization
- Security
- Consistency & replication

- Naming
- Fault Tolerance



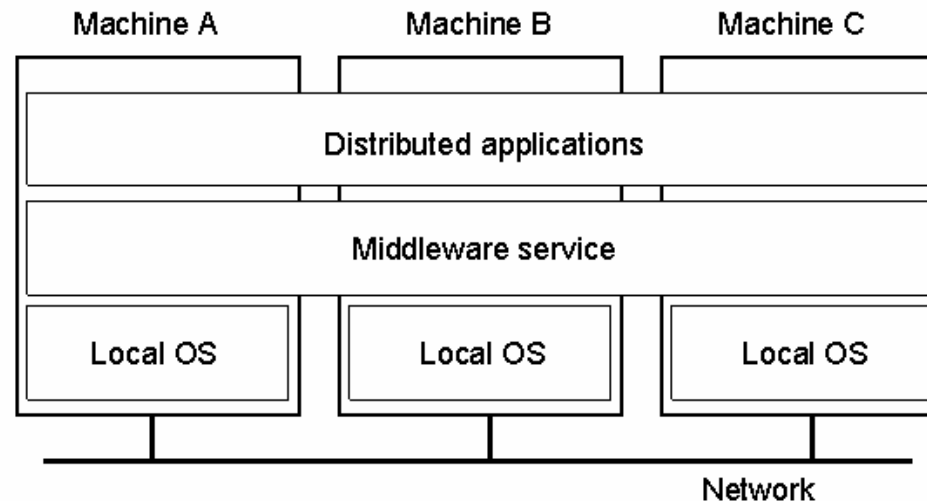
Introduction & overview

- Distributed systems, why and what
- Hardware concepts
- Software
 - Operating Systems
 - Middleware
 - Clients & Servers



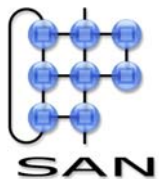
Definition (van Steen)

- **Distributed system:** *A collection of independent computers that appears to its users as a single coherent system.*



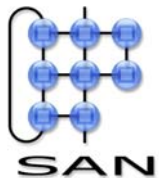
Definition (Lukkien)

- **Distributed system:** *the hard- and software of a collection of independent computers that **cooperate** to realize some functionality*
 - usually, individual parts cannot realize that functionality just by themselves
 - may be: serving a user (...ultimately)
 - ... but, user may not be involved directly



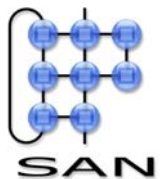
Independent

- Concurrent
- Independent failure
- No shared clock
- No centralized control
- (Spatial distribution)



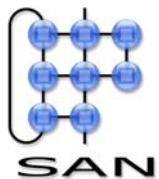
Motivation:

- Share resources
 - connect resources & users
 - compose applications from distributed functionality
- Transparency
 - hide internal details with respect to some property
- Openness
 - explicit boundaries
 - interfaces, protocols
 - policy vs. mechanism
- Scalability
 - size, location, administration
- Performance
 - concurrency, centralized compute power



Transparency in a Distributed System

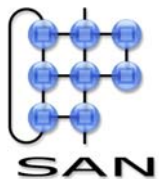
Transparency	Description
Access	Hide differences in data representation and how a resource is accessed
Location	Hide where a resource is located
Migration	Hide that a resource may move to another location
Relocation	Hide that a resource may be moved to another location while in use
Replication	Hide that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



Transparency

- Not always a goal – unawareness
 - can result in performance loss (e.g., caching)
 - can obstruct the use of lower-level information
 - e.g. location, hardware properties

- Not entirely possible
 - never sure of receipt of last sent message
 - fundamental limitation
 - cannot distinguish slow machine from failing one

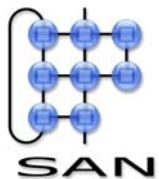


Scalability principles

- Requirement
 - Cost of increased complexity lower than benefit

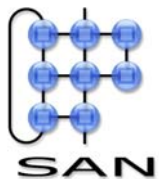
- Rules
 - No machine needs/has complete information
 - Decisions are based on local information
 - Failure of one machine does not ruin results
 - No assumption of a global, shared clock

- Techniques
 - Hide latency ... asynchrony
 - Replicate & relocate
 - data, but also work
 - Distribute tasks
 - locality principle



Scaling

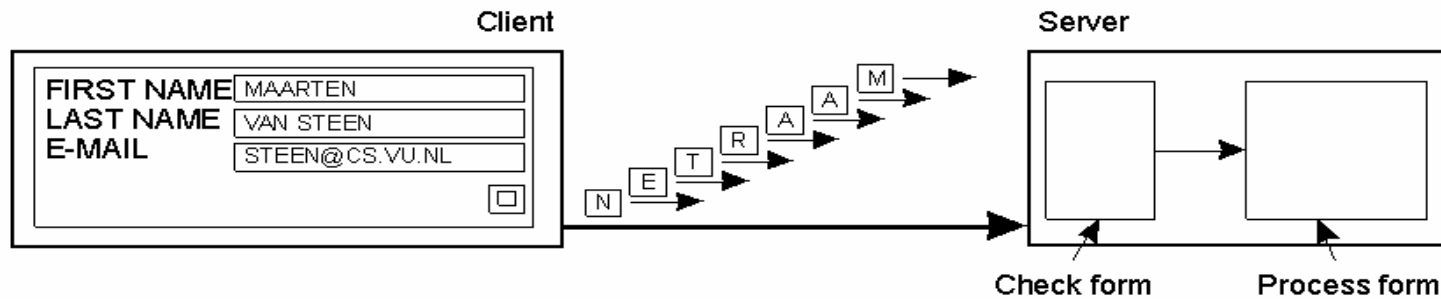
- Changing one or more particular system parameters....
 - typically one that relates to physical construction
 - size, number of nodes, distance, ...
- while keeping derived properties within specified bounds
 - typically, efficiency [= normalized performance]
 - related to a benchmark
- Can extend this to *scalable algorithms*
 - e.g. a distributed matrix multiplication, investigating efficiency as a function of number of processors or problem size



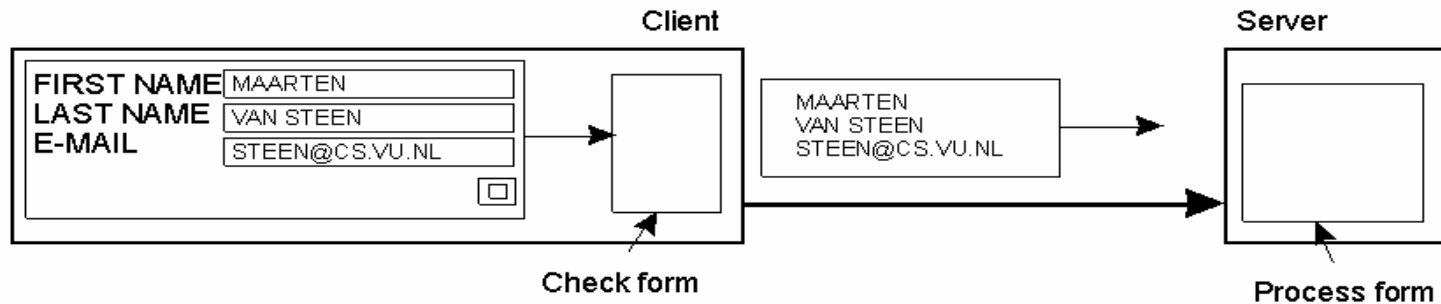
Scalability Problems

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

Scaling Techniques (1)

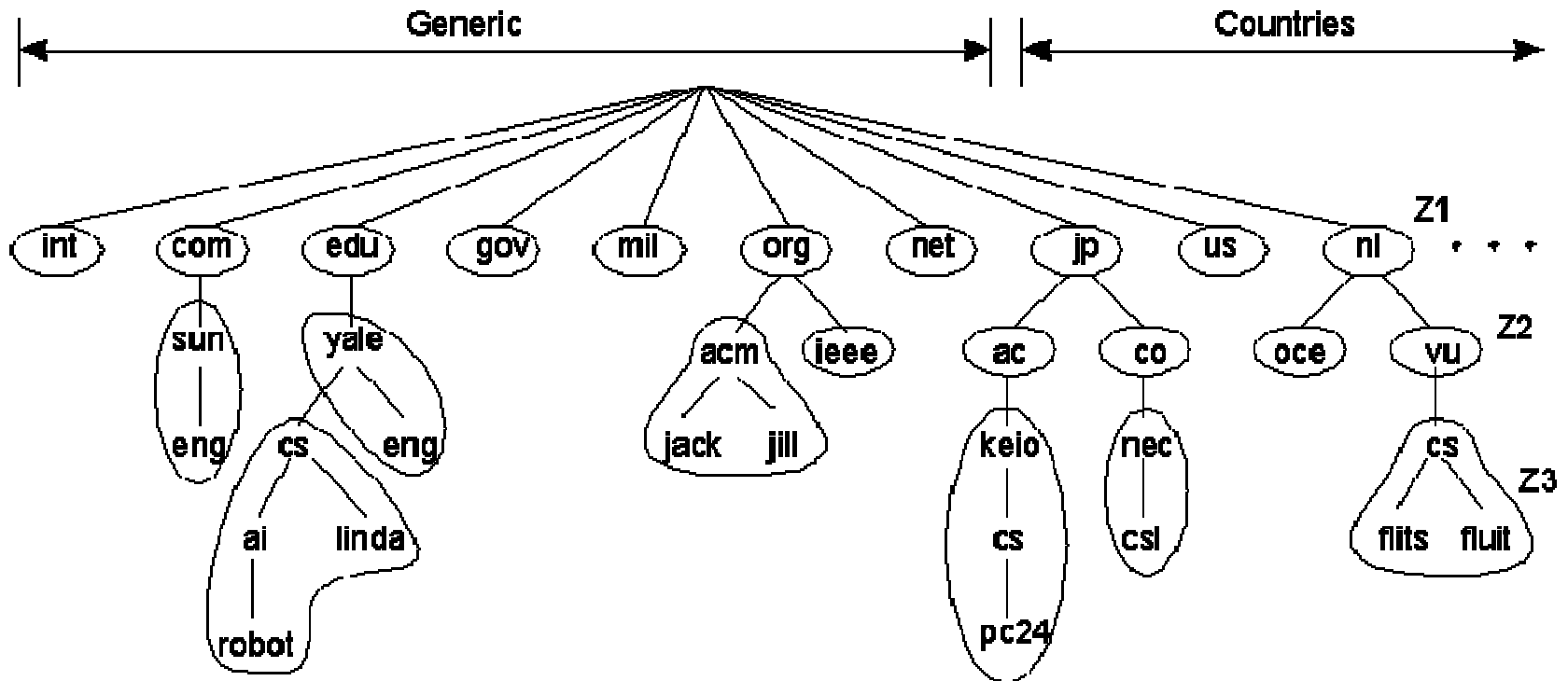


(a)



(b)

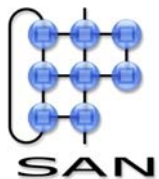
Scaling Techniques (2)



Dividing the DNS name space into zones.

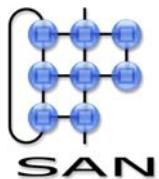
Introduction & overview

- Distributed systems, why and what
- Hardware concepts
- Software
 - Operating Systems
 - Middleware
 - Clients & Servers



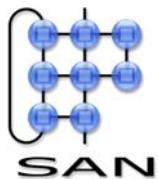
Hardware concepts

- A network is modeled by a graph
 - Nodes: Processors, Memories, P+M
 - P+M may also be a dedicated network component
 - Network purpose:
 - connect P with M: tightly coupled
 - connect P+M's: loosely coupled
 - P+M identical: homogeneous
 - Edges represent communication medium
 - broadcast, i.e. multiple access
 - e.g. bus, ethernet
 - point-to-point
 - e.g. “switched” ethernet



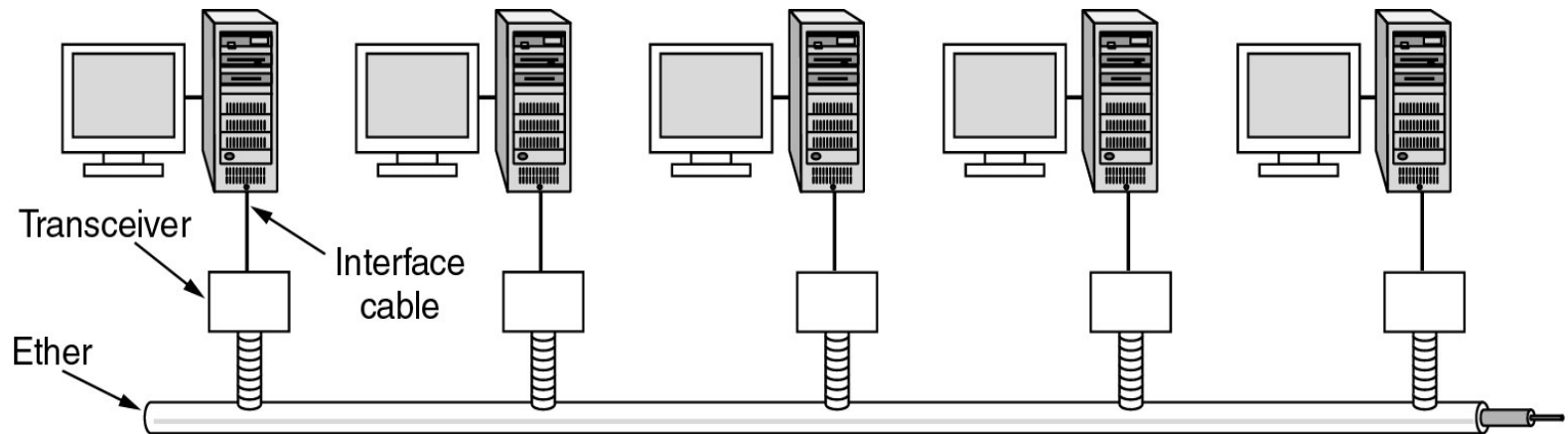
Broadcast medium

- All access points see the same physical signals
 - ...though not at the same time
 - introduces dependence on propagation delays
 - need to deal with “collisions” (in fact: interference)
 - arbitration: centralized / distributed
 - back-off and try again....
- Implementation:
 - Several access points on a single “wire”
 - e.g., bus, coax ethernet, wireless LAN
 - Access points connected by repeaters & hubs
 - **hub**: repeat and amplify signal (as well as noise) on all lines, except where it came from
 - “layer 1 switch”, e.g. for ethernet
 - **repeater**: two-party hub (copy signal from one onto other)
 - The above two are functionally identical



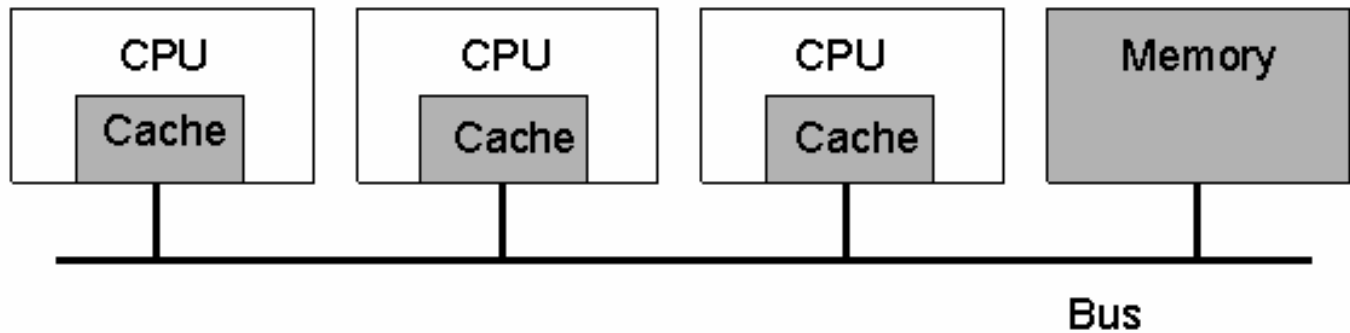
10Base2 ethernet

- No arbitration: detect and avoid collision CSMA/CD
 - 10Base2: 10 Mbps, 200meter, coax
- Loosely coupled

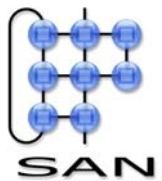
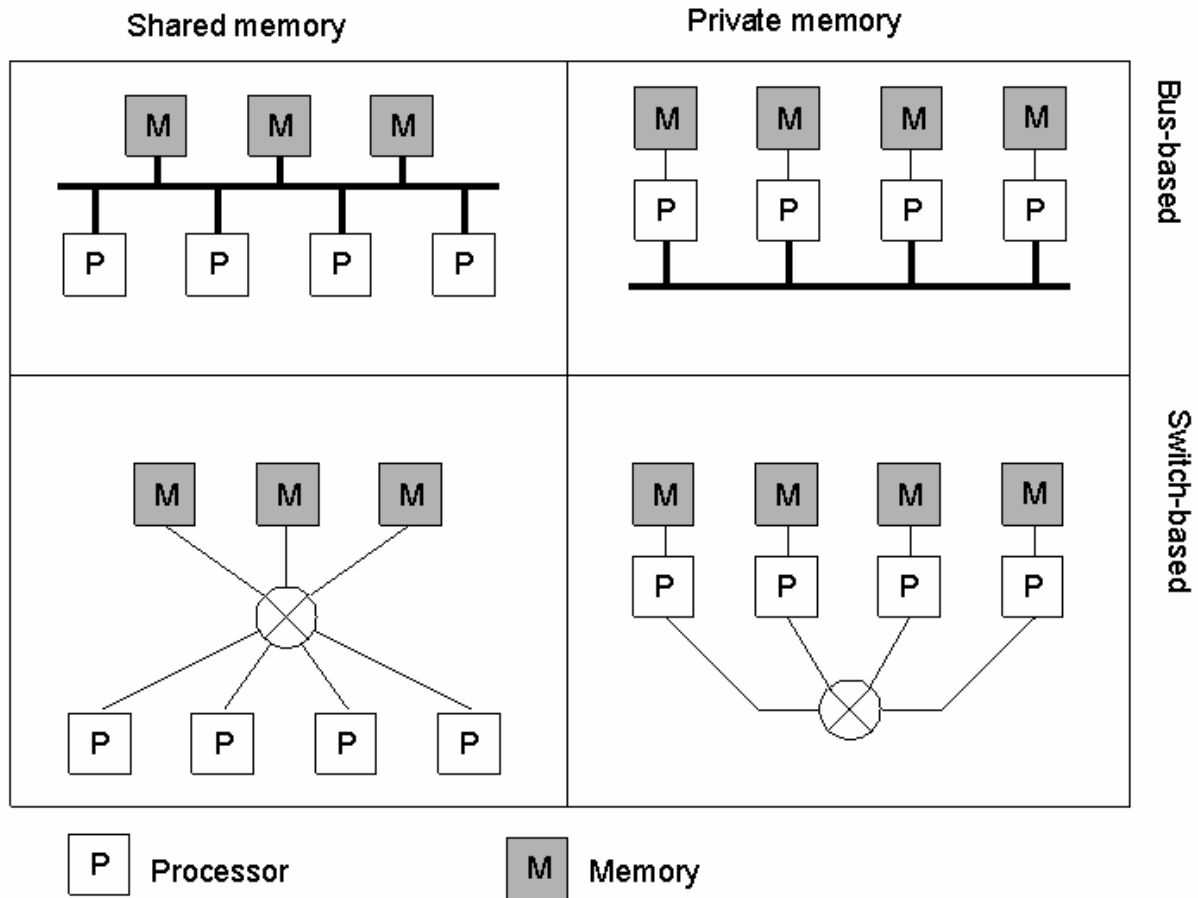


A bus-based multiprocessor

- Arbitration, centralized (bus master)

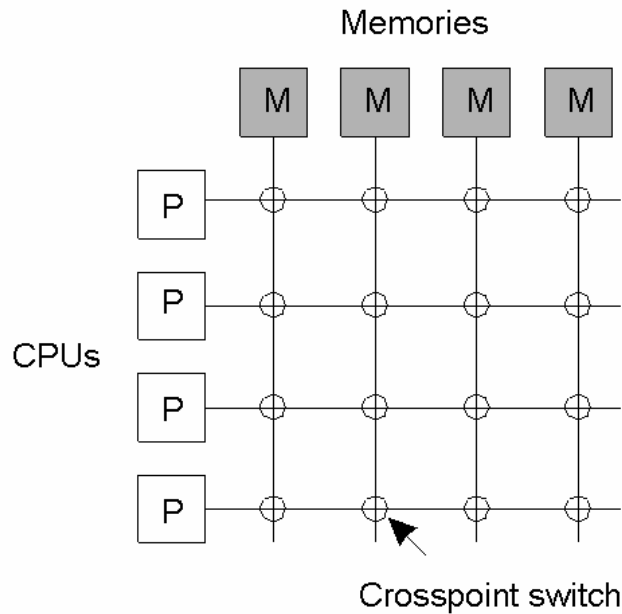


Bus and switch

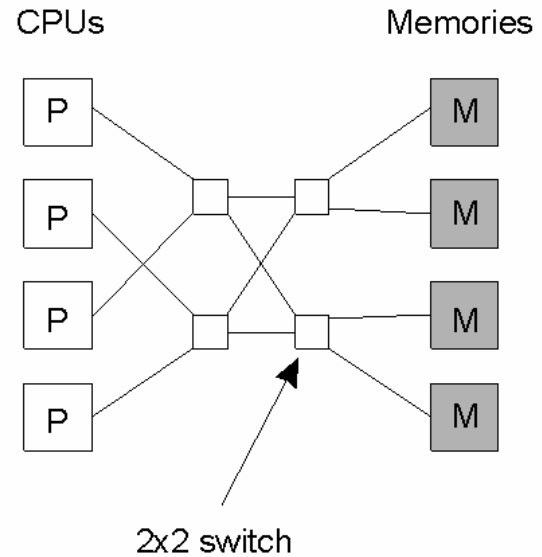


Switch implementations

- Crosspoint (crossbar) & multistage crossover
 - multistage crossover: $n/2 \log n$

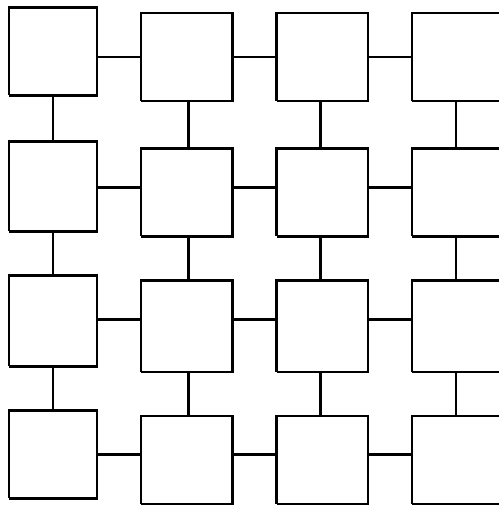


(a)

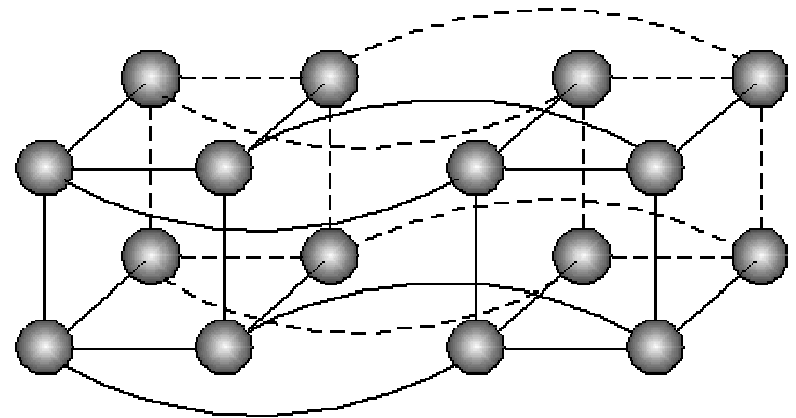


(b)

Homogeneous p2p: Grid and Hypercube



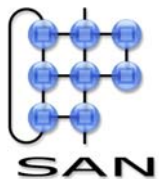
(a)



(b)

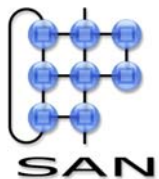
Broadcasting & scaling

- Networks based on broadcasting are limited in size (geographical, number of nodes)
 - **Collision domain:** the domain in which physical broadcasting is done (derived from collision-based methods)
- Communication based on packets for
 - scaling in network size
 - decoupling



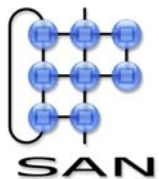
Current networks: heterogeneous

- Connect multi-computers, PC's, palm-tops
 - using a variety of physical connections
- Just connect arbitrary networks
 - e.g. wire-less and regular LAN, bluetooth
- Generally,
 - physical connections do not connect all pairs directly
 - some nodes need to transport information explicitly
 - these nodes need to interpret the passed information to some extent
-need hiding of these details (transparency)



Introduction & overview

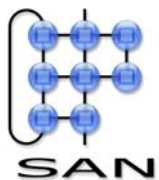
- Distributed systems, why and what
- Hardware concepts
- Software
 - Operating Systems
 - Middleware
 - Clients & Servers



Software Concepts

System	Description	Main Goal
DOS	Tightly-coupled operating system for multi-processors and homogeneous multicomputers	Hide and manage hardware resources
NOS	Loosely-coupled operating system for heterogeneous multicomputers (LAN and WAN)	Offer local services to remote clients
Middleware	Additional layer on top of NOS implementing general-purpose services	Provide distribution transparency

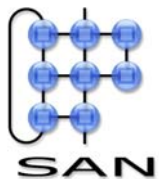
- DOS: Distributed Operating Systems
- NOS: Network Operating Systems



Uniprocessor Operating Systems

- Provide “virtual machine” to user: transparent
 - multi-tasking, multi-user, communication & synchronization primitives, i/o,
 - OS-support for shared-memory concurrency: synchronization primitives
 - shared variables, semaphores, monitors
 - however, no real concurrency, hence simple implementations
- Architecture: monolithic or microkernel
- Micro-kernel as a software architecture:
 - “*separates a minimal functional core from*
 - *extended functionality and*
 - *customer-specific parts*
 - *and serves as a socket for plugging-in these extensions and coordinate their collaboration*”

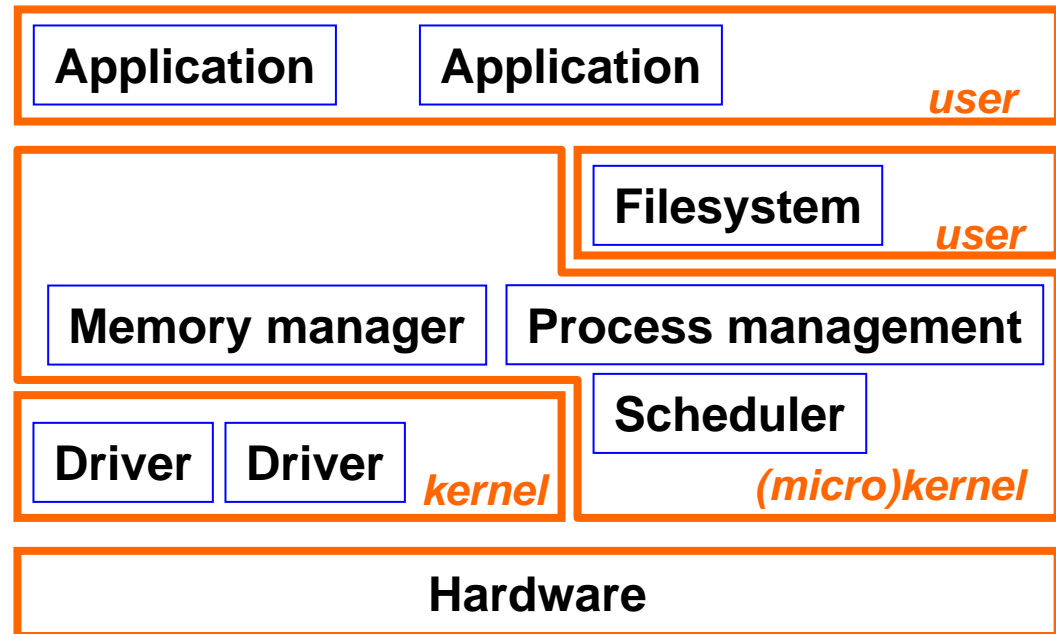
(from “*Pattern-oriented software architecture*”, Buschmann et.al.)



Micro-kernel OS

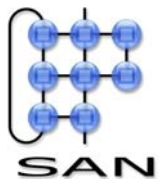
- Drivers, filesystem, IO and many other typical OS tasks go outside the kernel as *external* or *internal* services
 - easy to change ('hot' pluggable) or to remove
 - standard

memory
protection:
no system
crash in case
of error



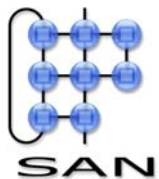
Micro-kernel (cnt'd)

- Performance penalty: all communication through kernel
- Typical communication facility between system components: message passing
 - natural view on distribution



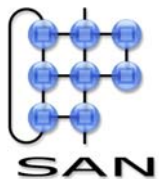
Extend the virtual machine to multiple processors

- Shared memory models:
 - UMA, (cc)NUMA
- OS-support for shared-memory concurrency: synchronization primitives
 - shared variables, semaphores, monitors
 - now: implementations must deal with real concurrency
- Question: what about scalability?

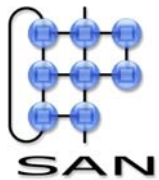
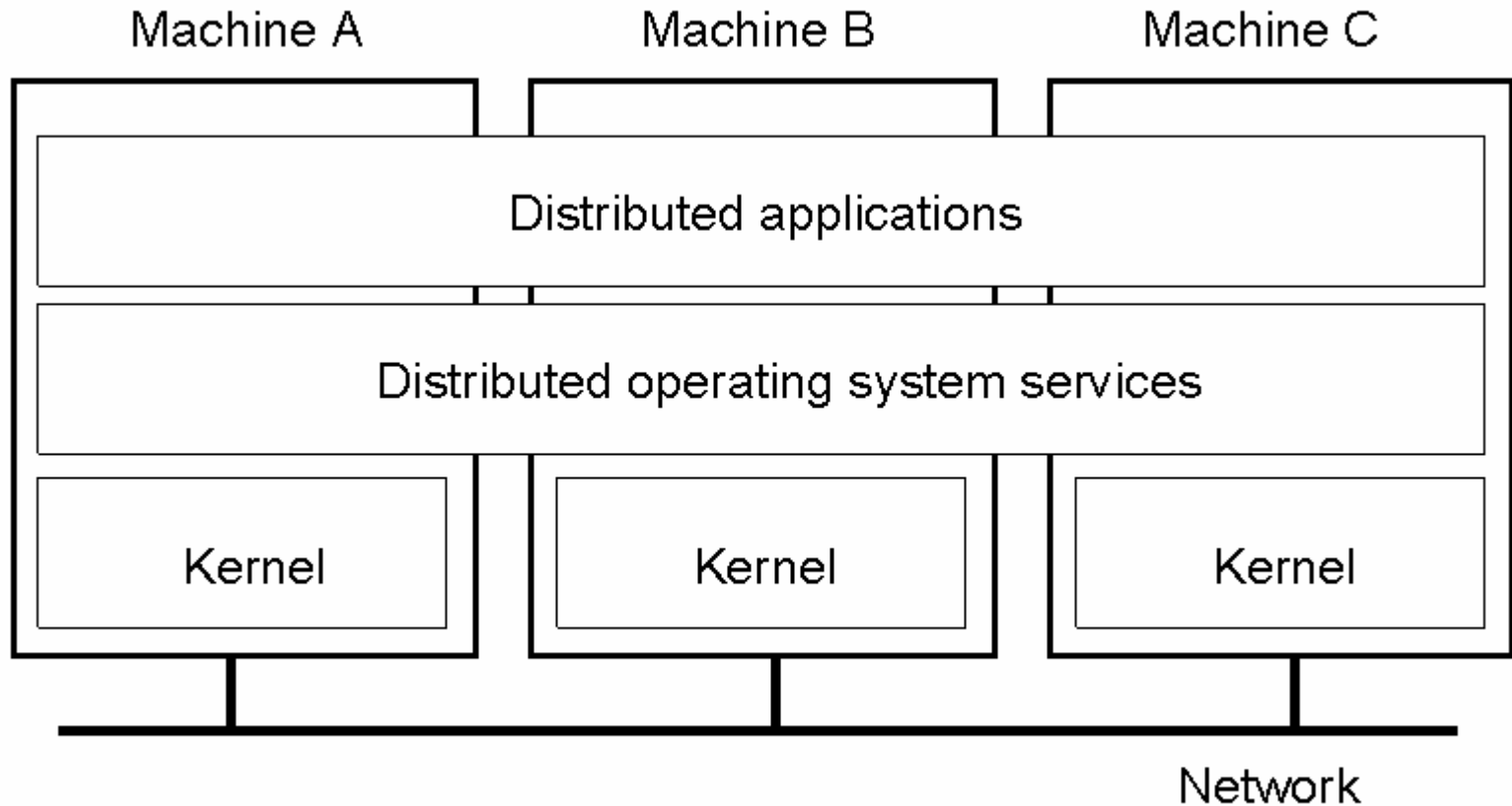


Distributed Operating System

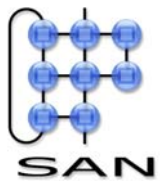
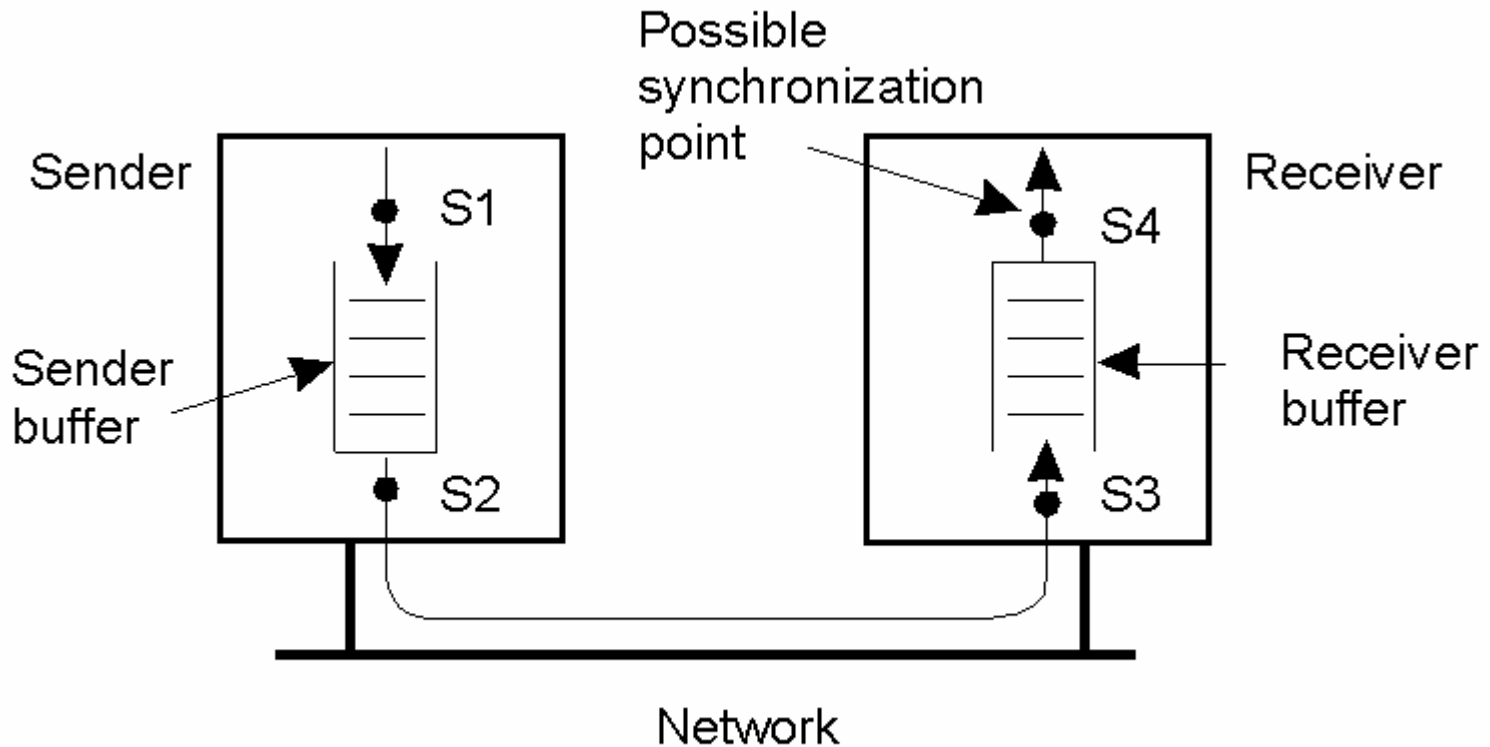
- Multicomputer: distributed memory
 - message passing is the basic primitive
 - no simple global synchronization
- Generally, OS kernel per machine
- Symmetric: kernels are (almost) the same
- System view: kernels “know” about each other
- Services: transparently distributed
- May emulate shared memory
 - but usually message passing is provided only



Distributed Operating System

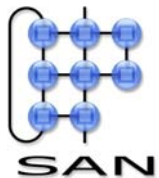
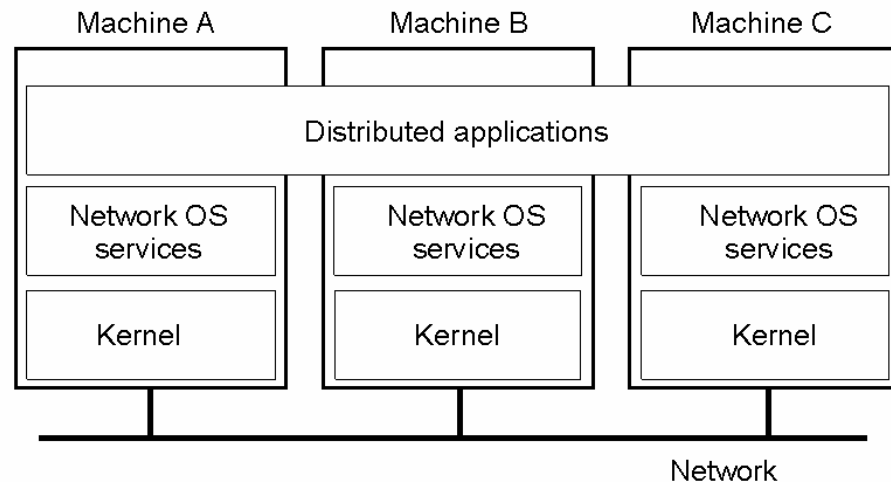


Message passing synchronization

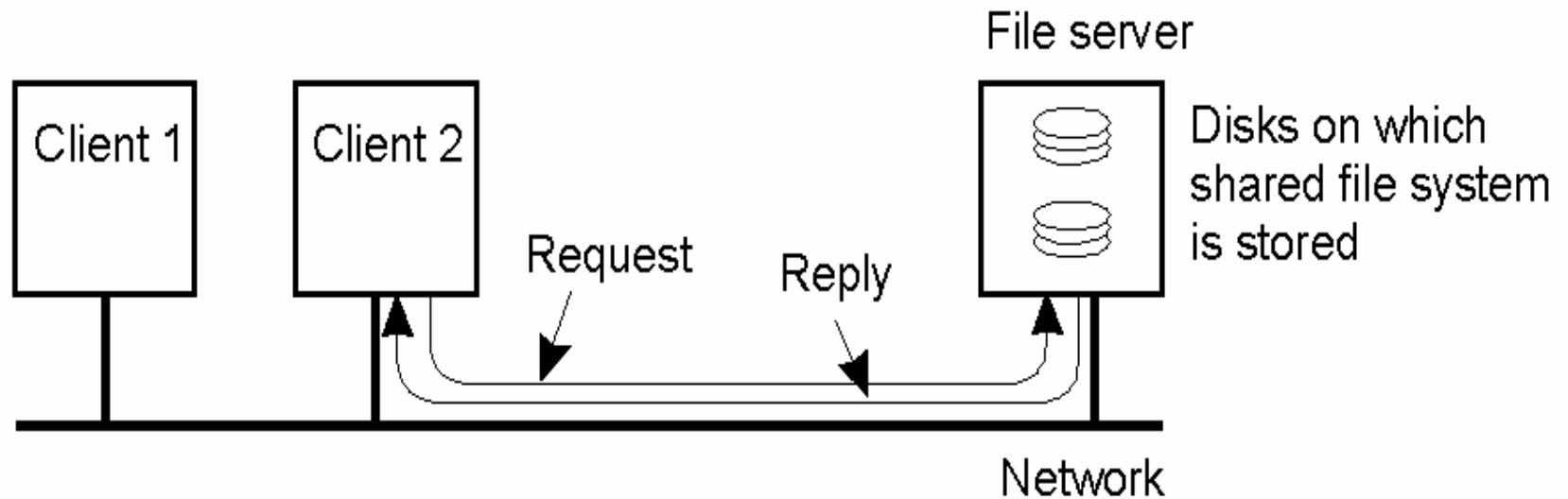


Network Operating System

- Machines independent:
 - own OS
 - heterogeneous
- Services: tied to individual machines
- File oriented

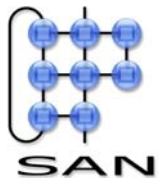
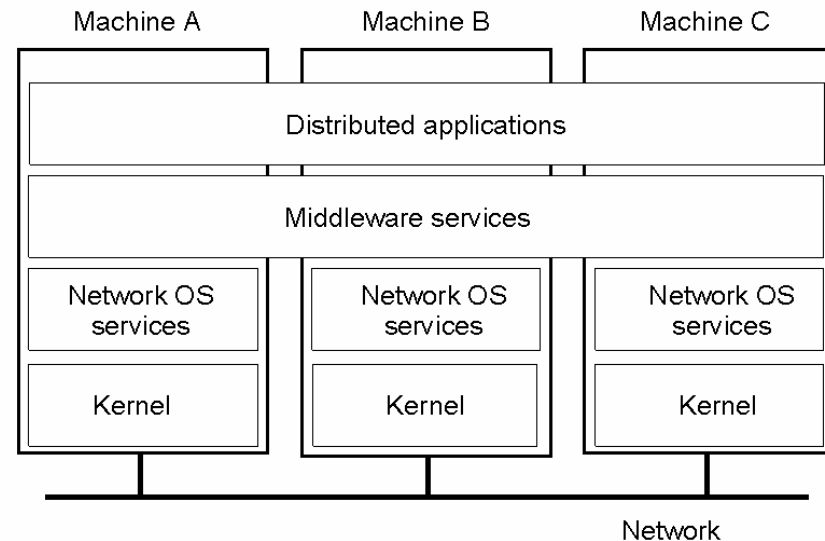


Two clients and a server in a NOS



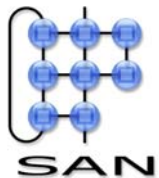
Middleware

- OS's don't know each-other, can be entirely different
 - transparency, in particular w.r.t. heterogeneity
- Still: distributed, general services needed
 - factor out common functionality
- NOS's services too basic and too diverse



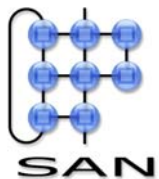
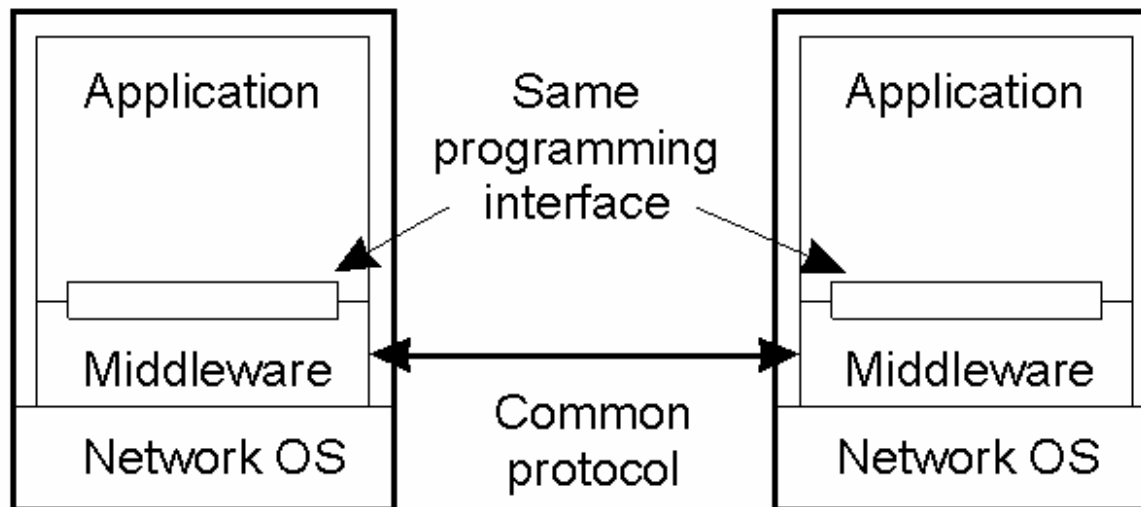
Middleware services

- Communication
 - RPC, message oriented,
- Information
 - database access, directory, naming, ...
- Control
 - transaction processing, code migration
- Security
 - authentication, encryption, ...



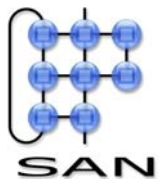
Middleware and Openness

- Openness concerns both
 - interface
 - protocol



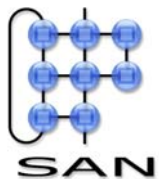
Comparison between Systems

Item	Distributed OS		Network OS	Middleware-based OS
	Multiproc.	Multicomp.		
Degree of transparency	Very High	High	Low	High
Same OS on all nodes	Yes	Yes	No	No
Number of copies of OS	1	N	N	N
Basis for communication	Shared memory	Messages	Files	Model specific
Resource management	Global, central	Global, distributed	Per node	Per node
Scalability	No	Moderately	Yes	Varies
Openness	Closed	Closed	Open	Open



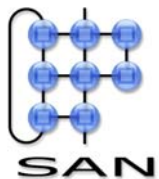
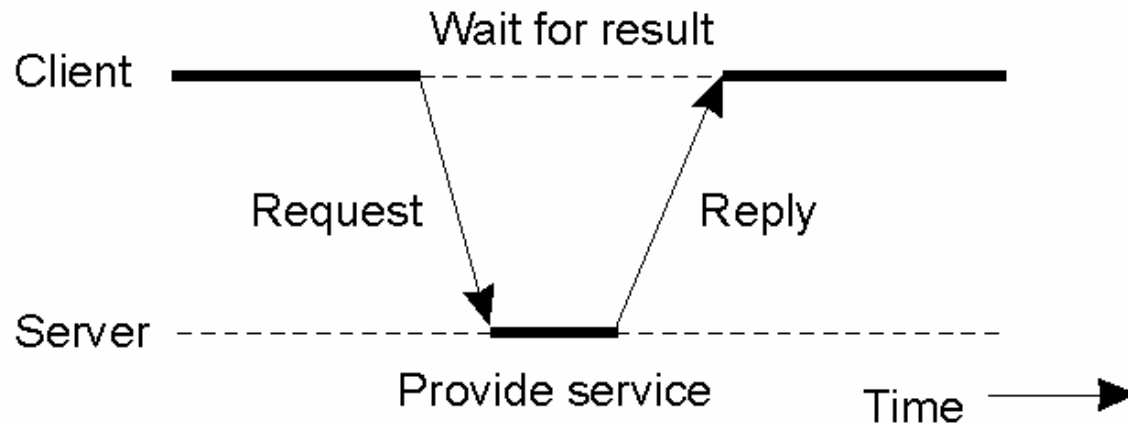
Introduction & overview

- Distributed systems, why and what
- Hardware concepts
- Software
 - Operating Systems
 - Middleware
 - Clients & Servers

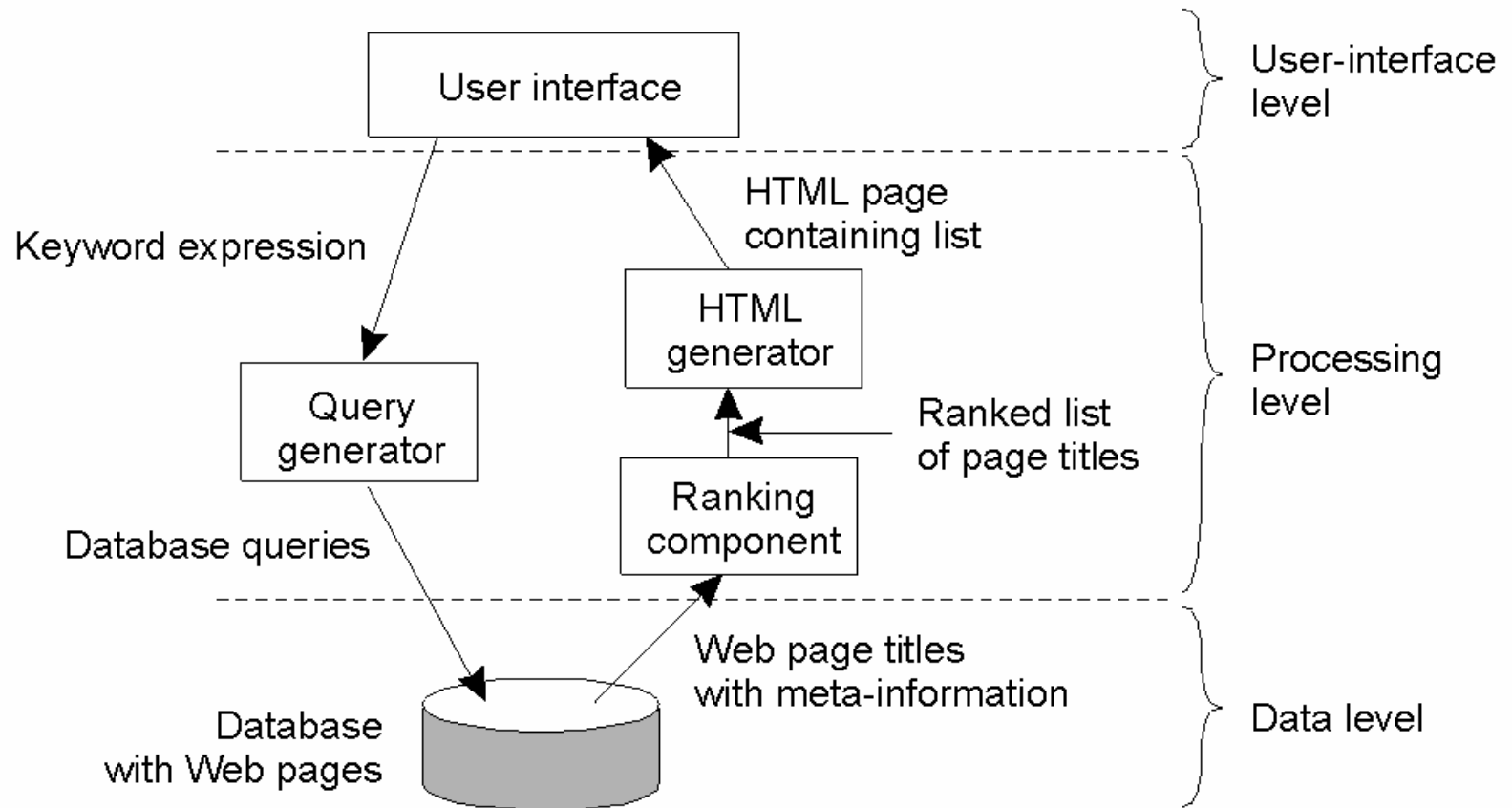


Clients and Servers

- Server: offers service
 - often: access to a shared resource
- Client: uses service
- Floating roles!

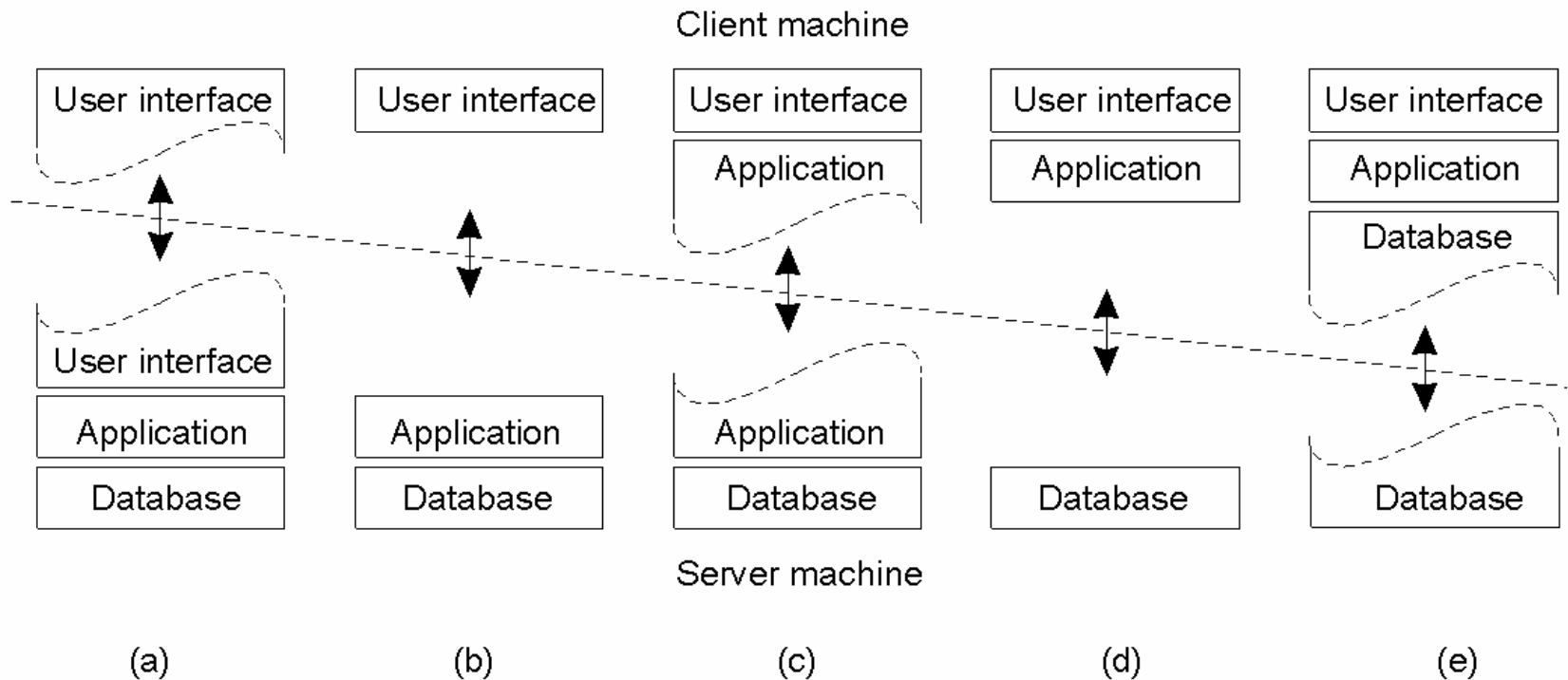


Application layering



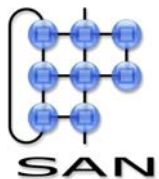
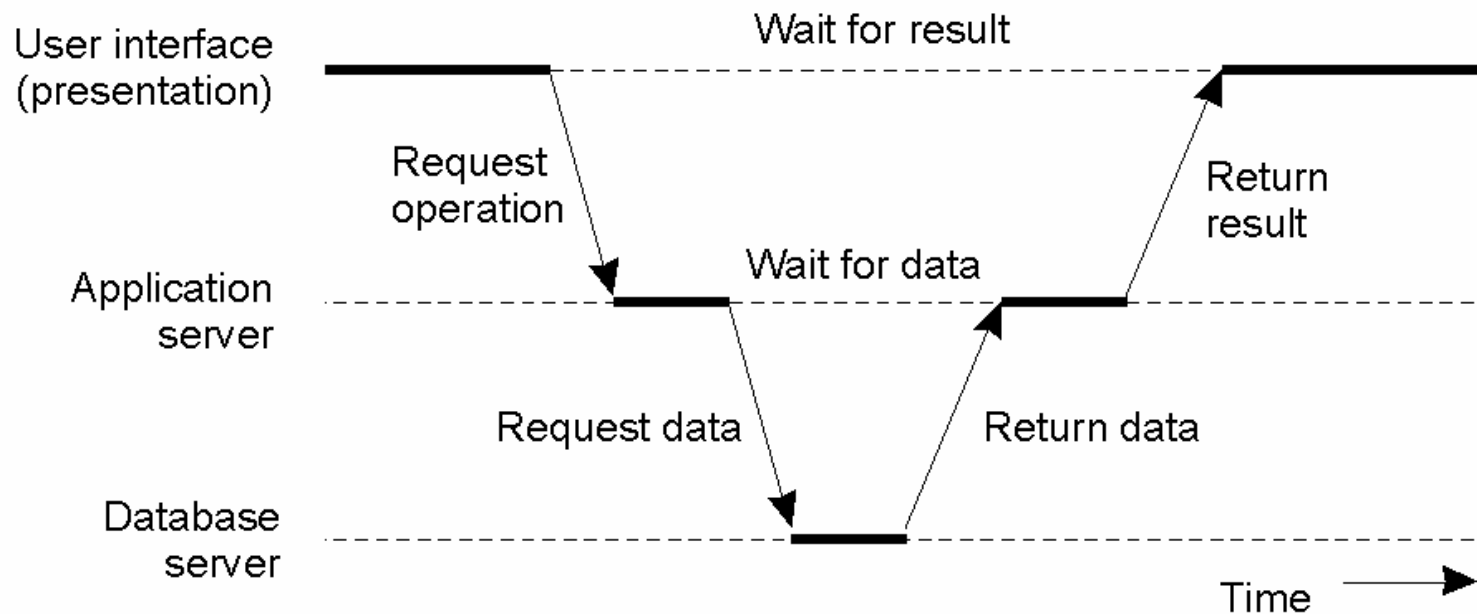
Multitiered Architectures

- Network cuts possible at many places
 - even multiple cuts!



Multitiered Architectures

- Each task a separate machine
- Server/client roles change



Modern Architectures

- Distinguish *horizontal* and *vertical* distribution
 - horizontal: distribute single layer
 - vertical: distribute layers

