Operating Systems II

Distributed O-Systems

roadmap:

- characteristics of distributed systems
- order in distributed systems
- models of communication and sharing
- distributed shared memory
- distributed file systems

Multi-Processor Systems

Bus-based Multi-Processor with single central memory.
Realization: Hardware.
Problems: Cache coherence and memory consistency.

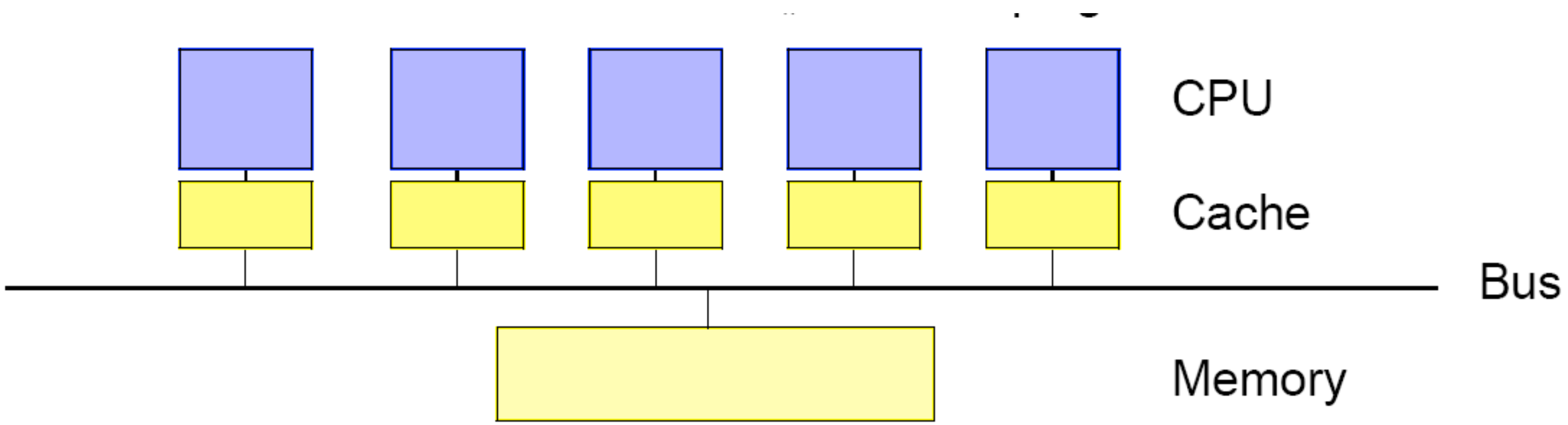
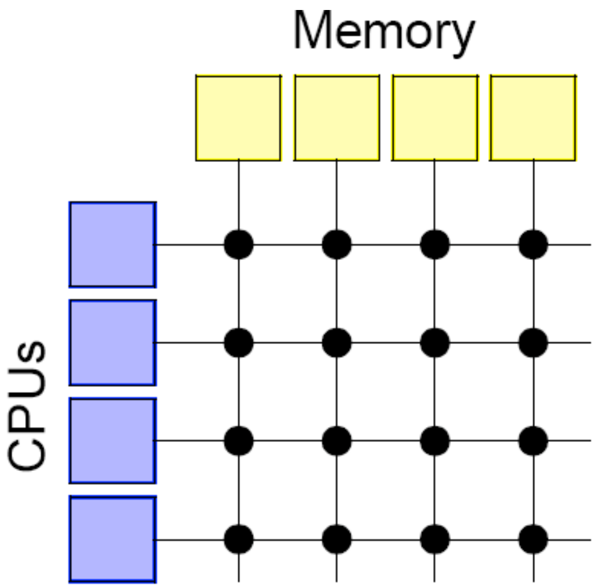
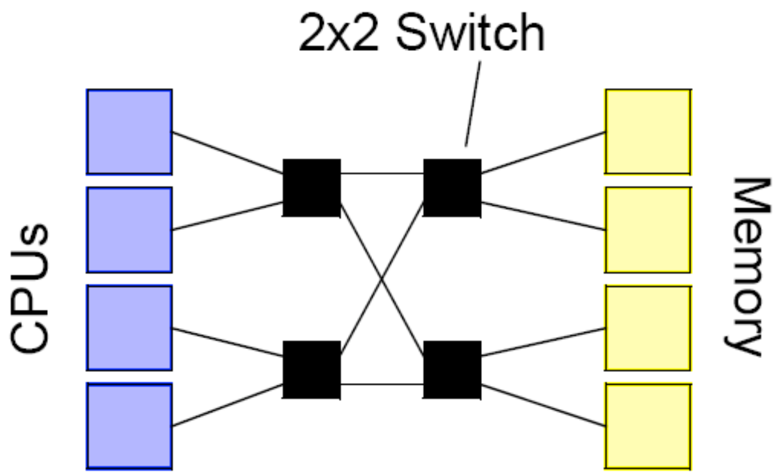


Abb.: F. Hauck: Modernen Konzepte verteilter Systeme (VL U.Ulm, 2002/02)

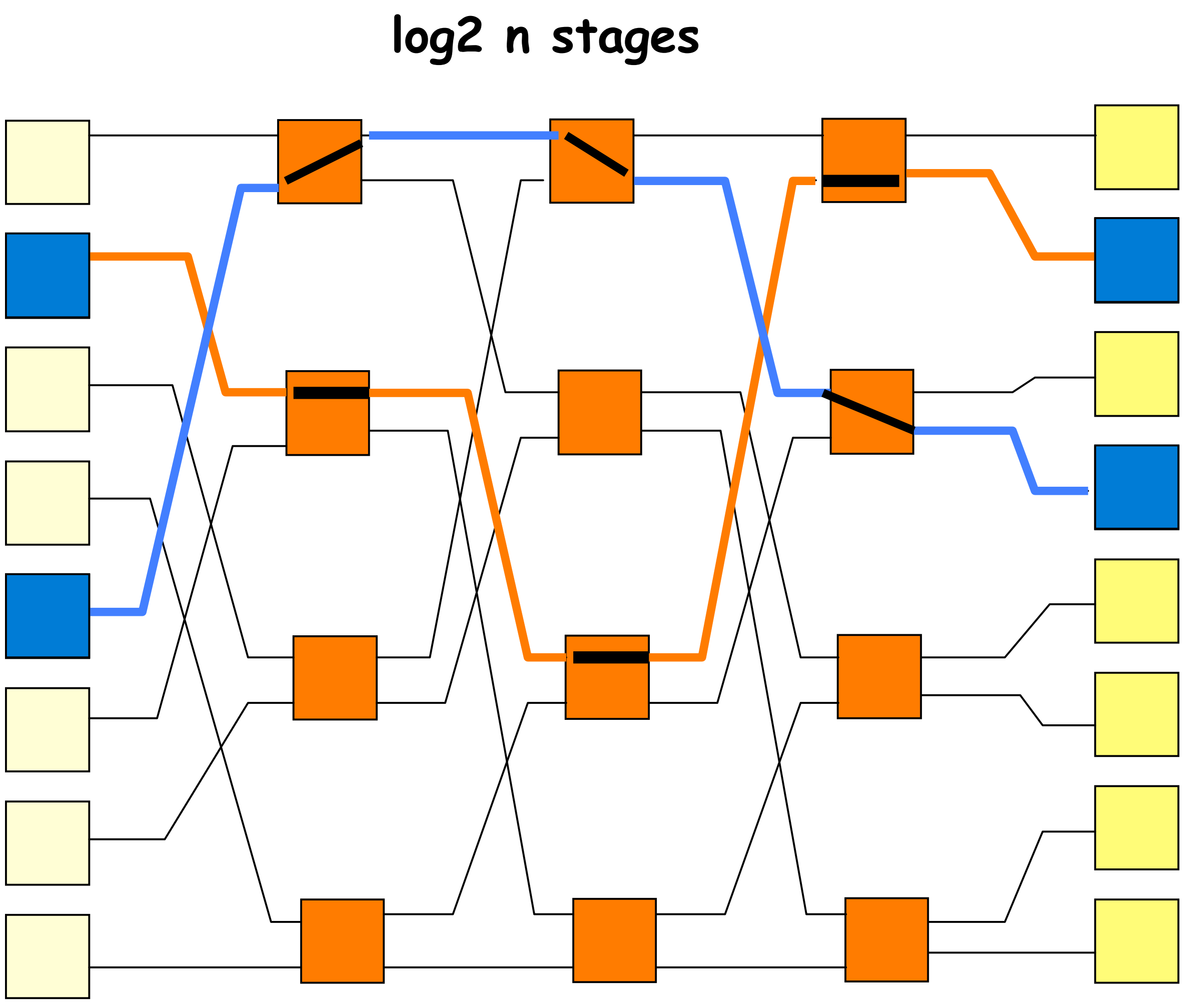
Multi-Processor Systems

Connection-based Multi-Processor with multiple memories.
Realization: Special switching network hardware (Omega networks, Banyan trees,..)
Problems: Complexity of the switching network.





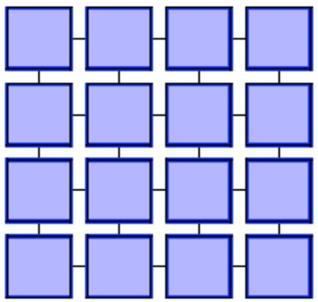
An Omega switching network



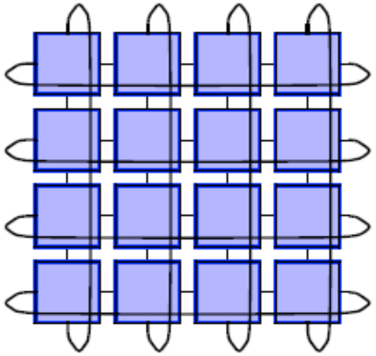
$2^k = N$
inputs

Multi-Processor Systems

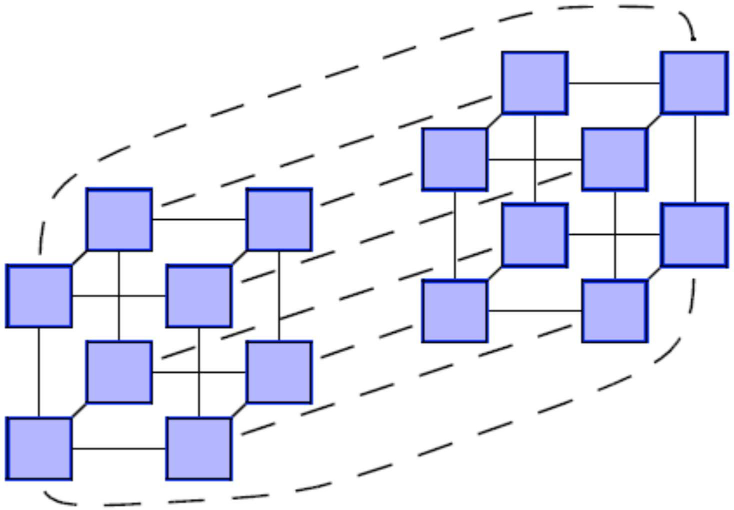
Grid



Torus



Hypercube



	max. distance
Grid	6
Torus	3
Hyperc.	3

Types of Multi-Processor Systems

	data	control	
shared memory multiproc.	c	c	tight coordination of multiple execution engines
computer cluster	d	c	central coordination of proc/mem pairs working on distributed data
distributed system	d	d	no central component.

What is a distributed system?

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

Andrew Tanenbaum: A distributed system is composed from multiple autonomous computers which appear as a single computer for a user.

George Coulouris: A distributed system is composed from multiple autonomous computers which coodinate actions by exchanging messages.

What is a distributed system?

Essential properties:

- multiple computers (local CPU-/memory-/network-/I-O-components)
- computers are autonomous, i.e. they have an independent local control
- computers are connected by a network and basically communicate by exchanging messages
- there is no special central control and coordination facility

Distributed Data + Distributed Control

What is a distributed system?

Essential properties:

- Concurrency of computations
- No global time (approximations possible)
- Components fail independently

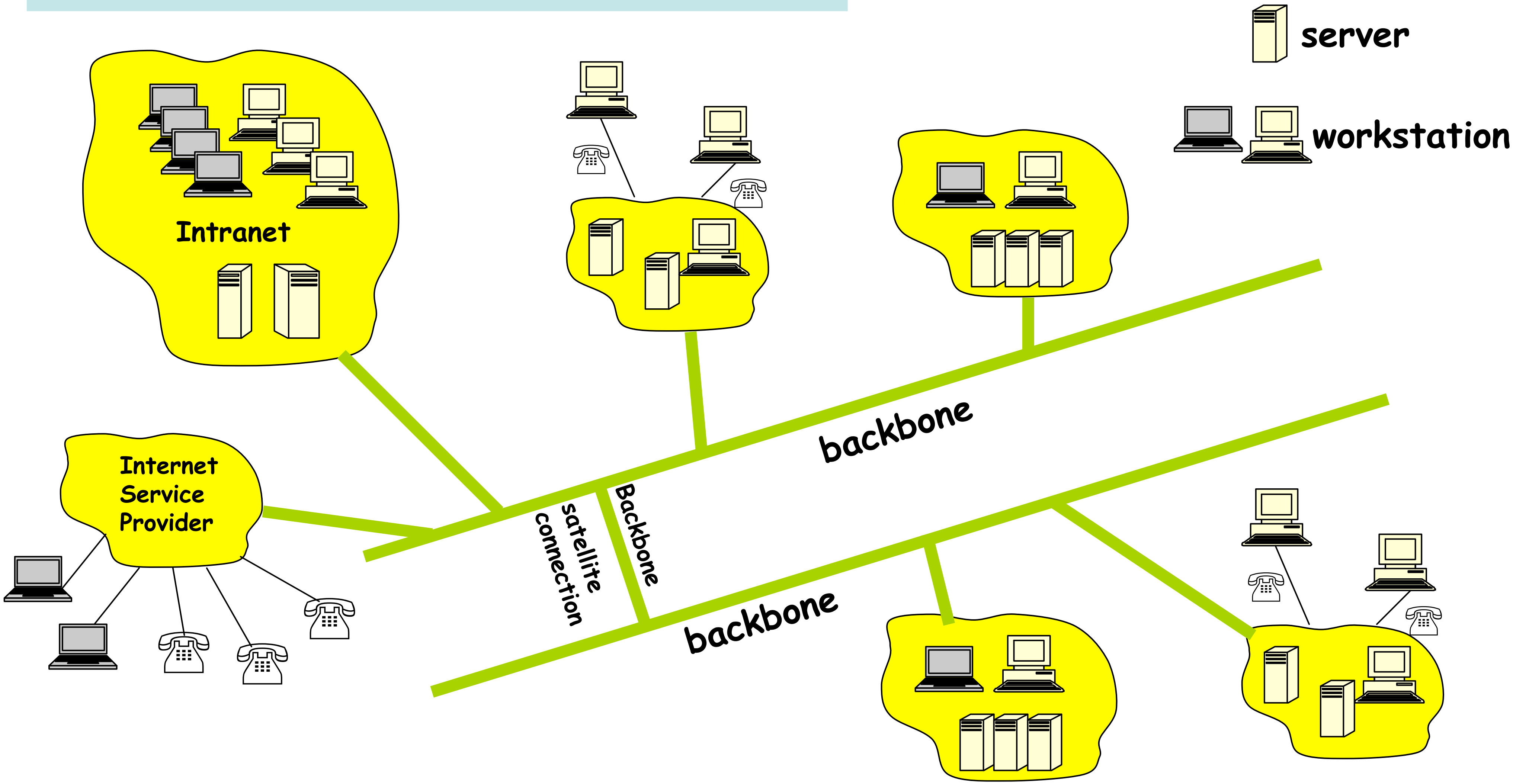
Why a distributed system?

- Performance
- Sharing of resources
- Independence of failure and no single point of failure
- Distributed nature of application
- Distributed data
- Extensibility and Scalability

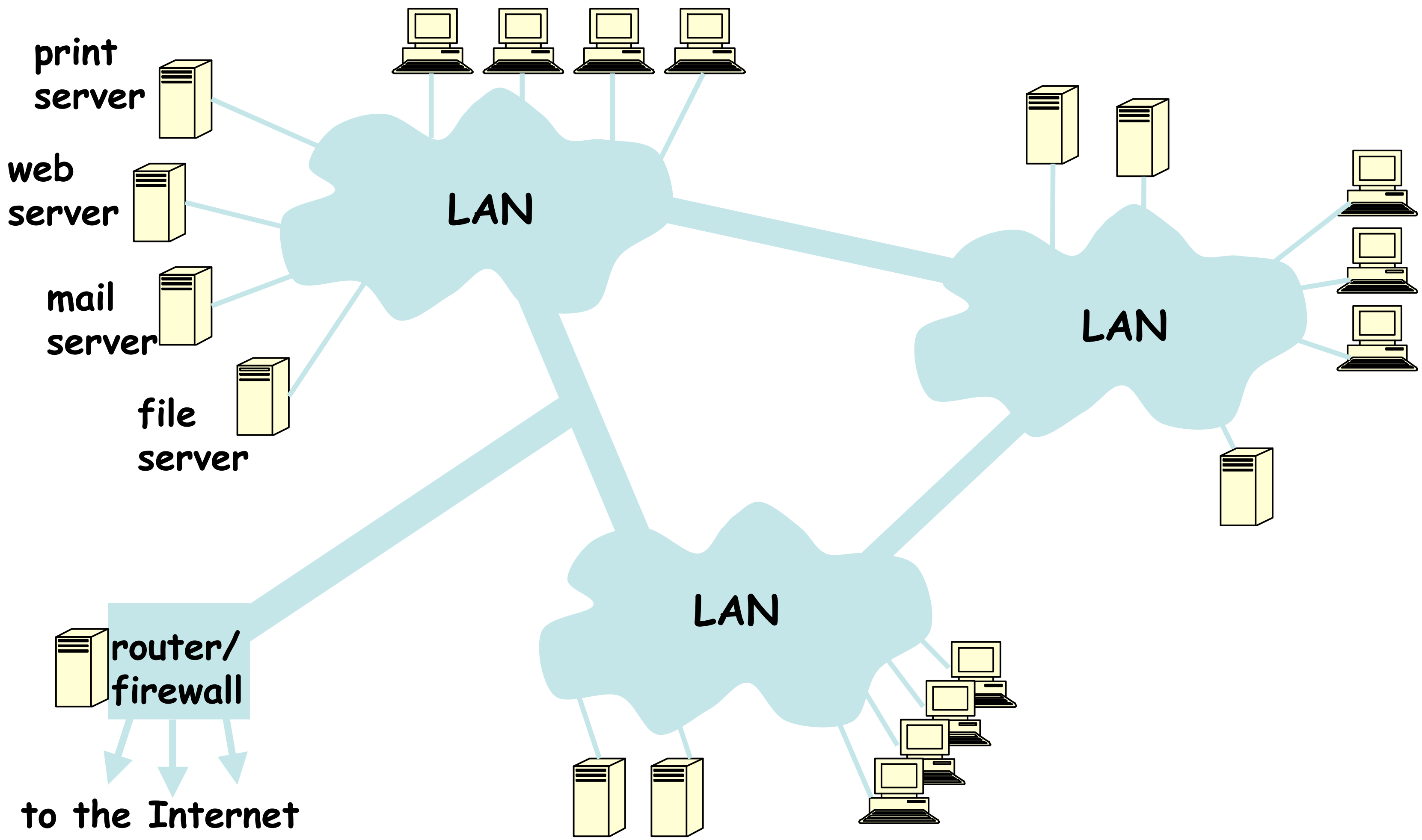
Examples

- The Internet
- An Intranet
- Distributed Control Systems
- Ubiquitous and mobile computing environments

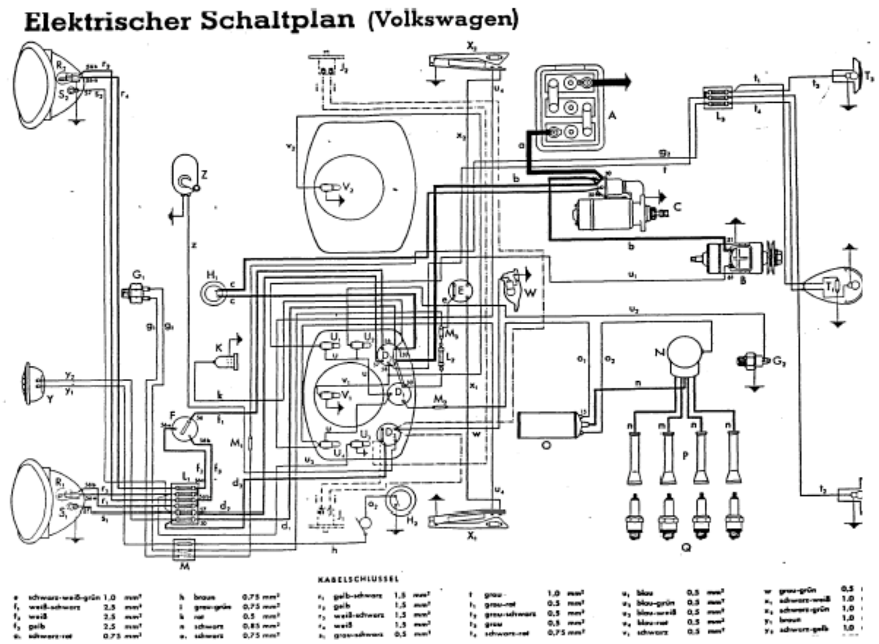
Example: Internet



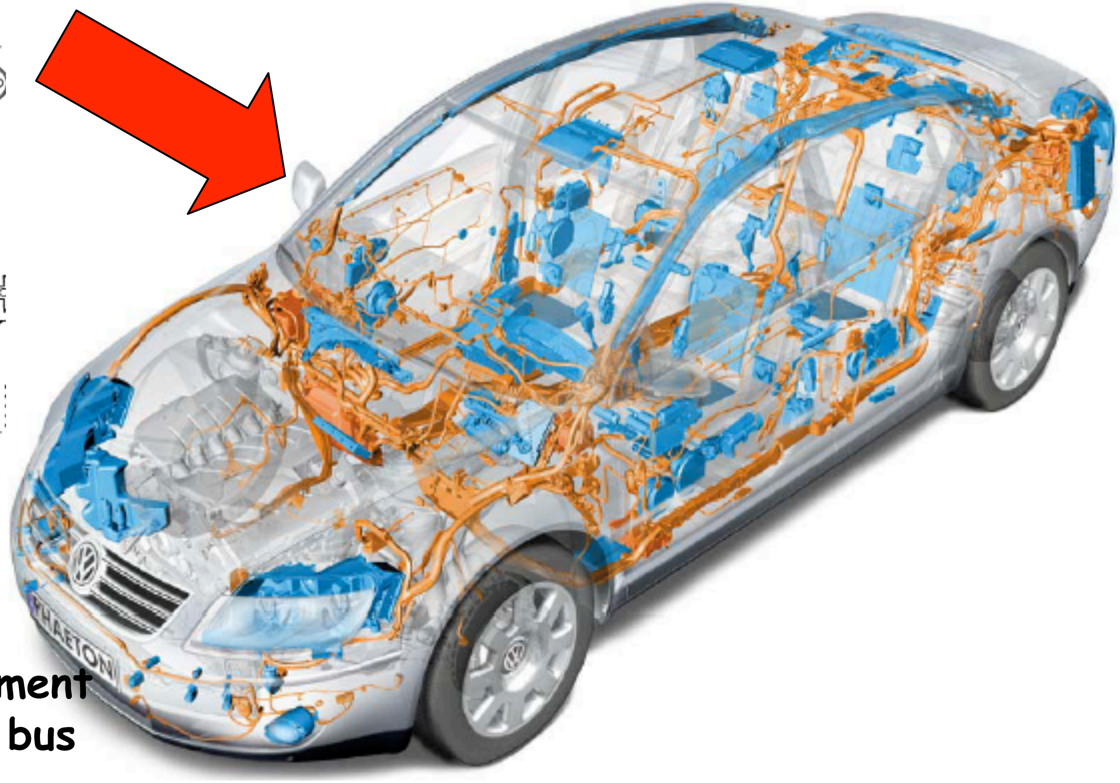
Example: Intranet



Example: Control Networks

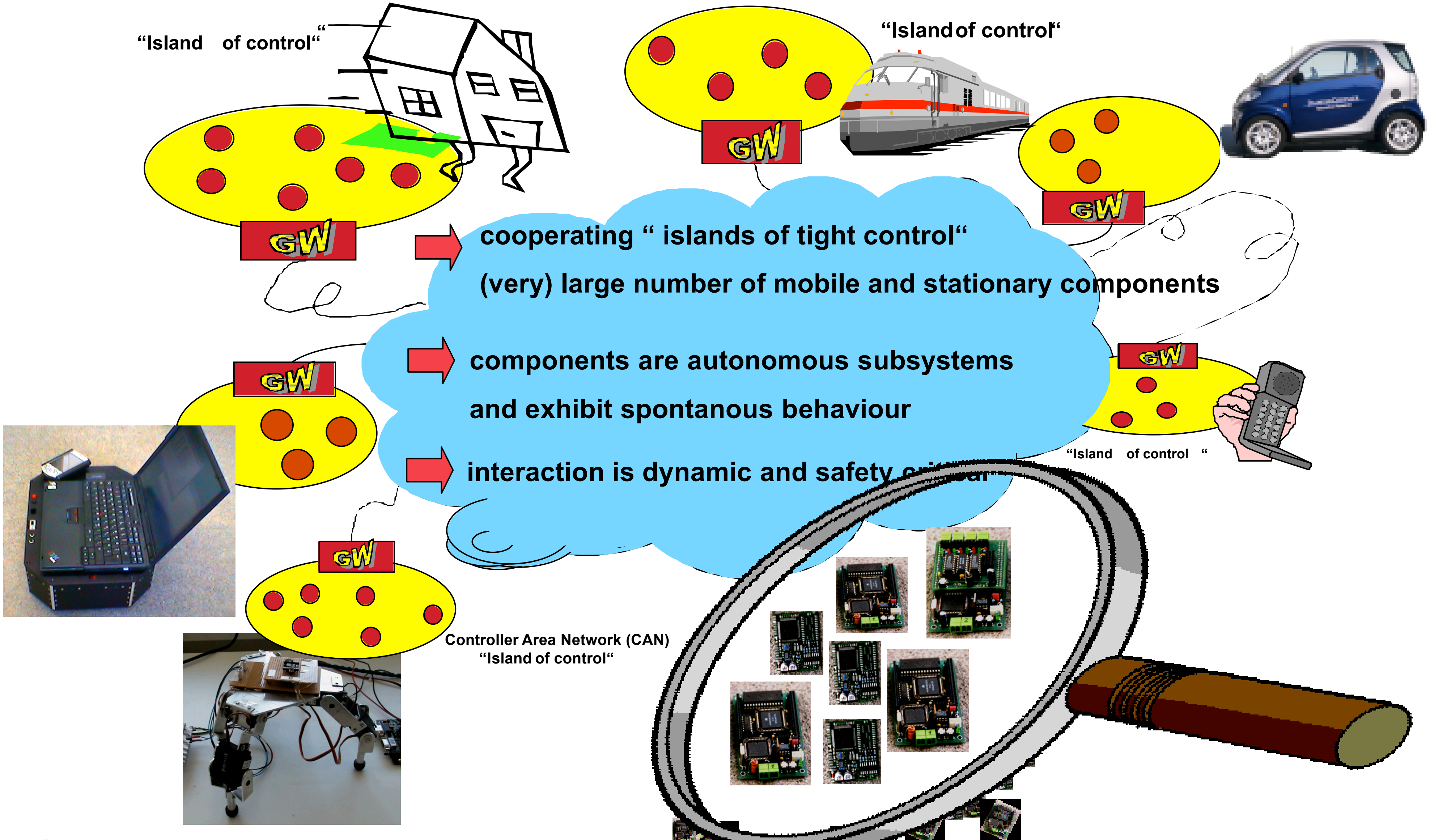


drastically increasing complexity



- 11.136 electrical parts
- 61 ECUs
- Optical bus for information and entertainment
- Sub networks based on proprietary serial bus
- 35 ECUs connected to 3 CAN-Busses
- 2500 signals in 250 CAN messges

Example: A networked physical world



Problems and desirable properties

- general problems: concurrency, delays, faults
- more problems: heterogeneity, openess, scalability
- desirable properties:

A distributed system should be programmable like a local, centralized computer (→ see Tanenbaum).

???

- Support to deal with the above problems in an application specific way on an adequate level of abstraction. → Find a better definition!

Transparencies:

- Access transparency
- Location transparency
- Concurrency transparency
- Migration transparency
- Relocation transparency
- Replication tranparency
- Fault transparency
- Persistency transparancy

?

Qos transparency

Types of distributed operating systems

Network operating systems: basic support for communication between homogeneous local OS, individual computing nodes are visible
Examples: Windows NT, UNIX, Linux, distributed file systems (NFS)

Distributed operating systems: transparent IPC mechanism, no difference between local and remote interaction, unified name space, integrated file system, unified user admin and protection/security mechanisms.
Examples: Amoeba, Emerald, Chorus, Clouds

Middleware: builds on top of heterogeneous local OS, provides unified programming model, communication and cooperation mechanisms, maintains autonomy of local nodes but supports transparent access to shared resources.
Examples: CORBA, Java RMI, .NET, DCE

Types of middleware

Document-based middleware: model: distributed data	Documents which contain (hyper-) links to other documents. Examples: World-Wide-Web
File-based middleware: model: distributed data	Transparent access to remote files. Examples: Andrew File System, NFS
Object-based middleware: model: distrib. functions	Transparent invocation of remote objects. Examples: CORBA, DCOM(windows only)
Service-based middleware: model: distrib. functions	Discovery and use of remote services. Examples: Jini, JXTA, UPnP
Coordination-based middleware: model: distrib. functions	Coordination through a shared information space. Examples: Linda, Java Spaces, Lime

Shared Data Spaces

Immutable Data Storage:

- no write operation!
- "out" always adds a data element to the storage
- destructive "in" and non-destructive "read"
- consistency is preserved by ordering accesses
- examples: Linda, JavaSpaces

An example of a JINI service

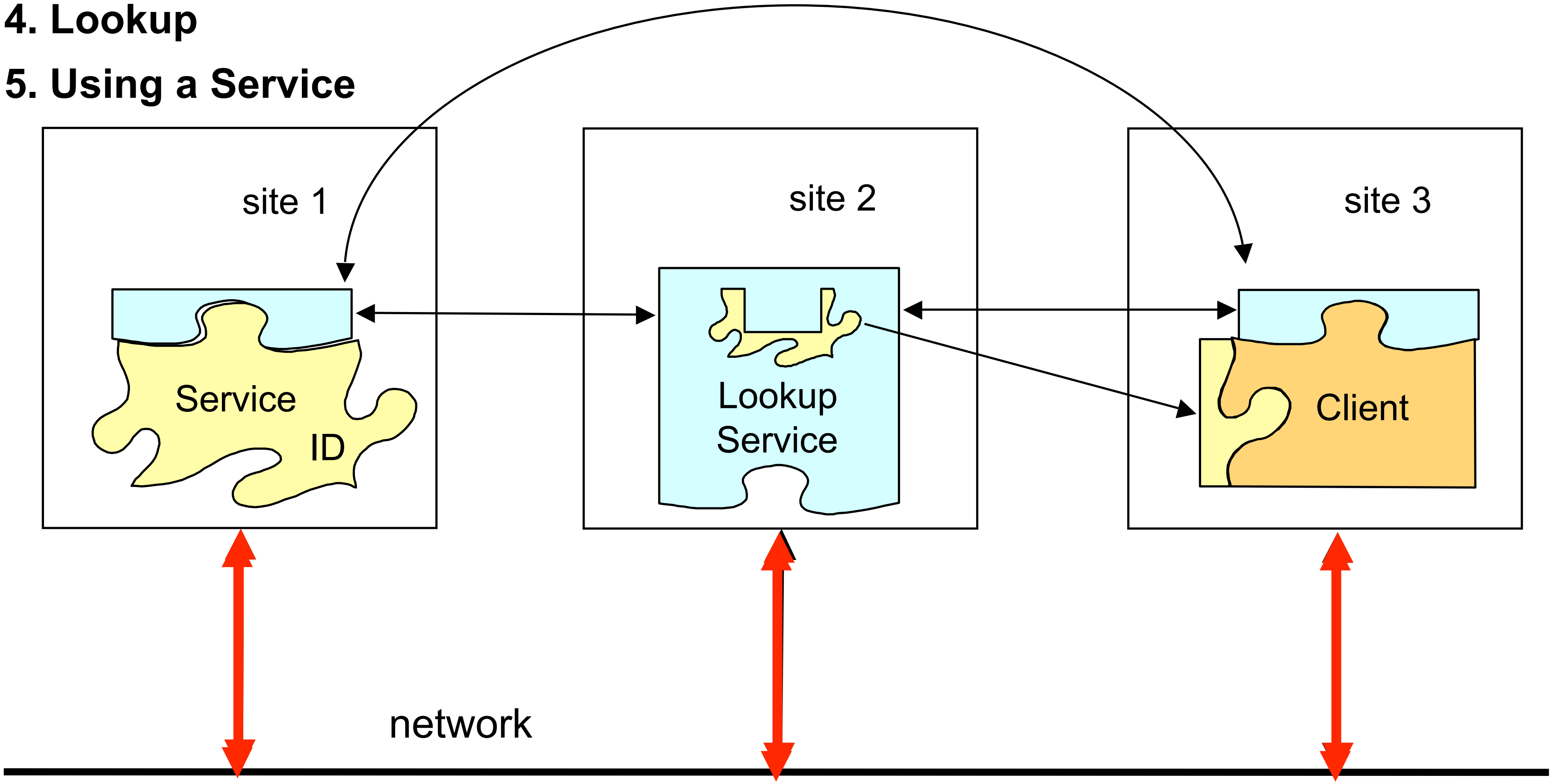
1. Discovery - Finding Lookup Services

2. Join - Service Registration

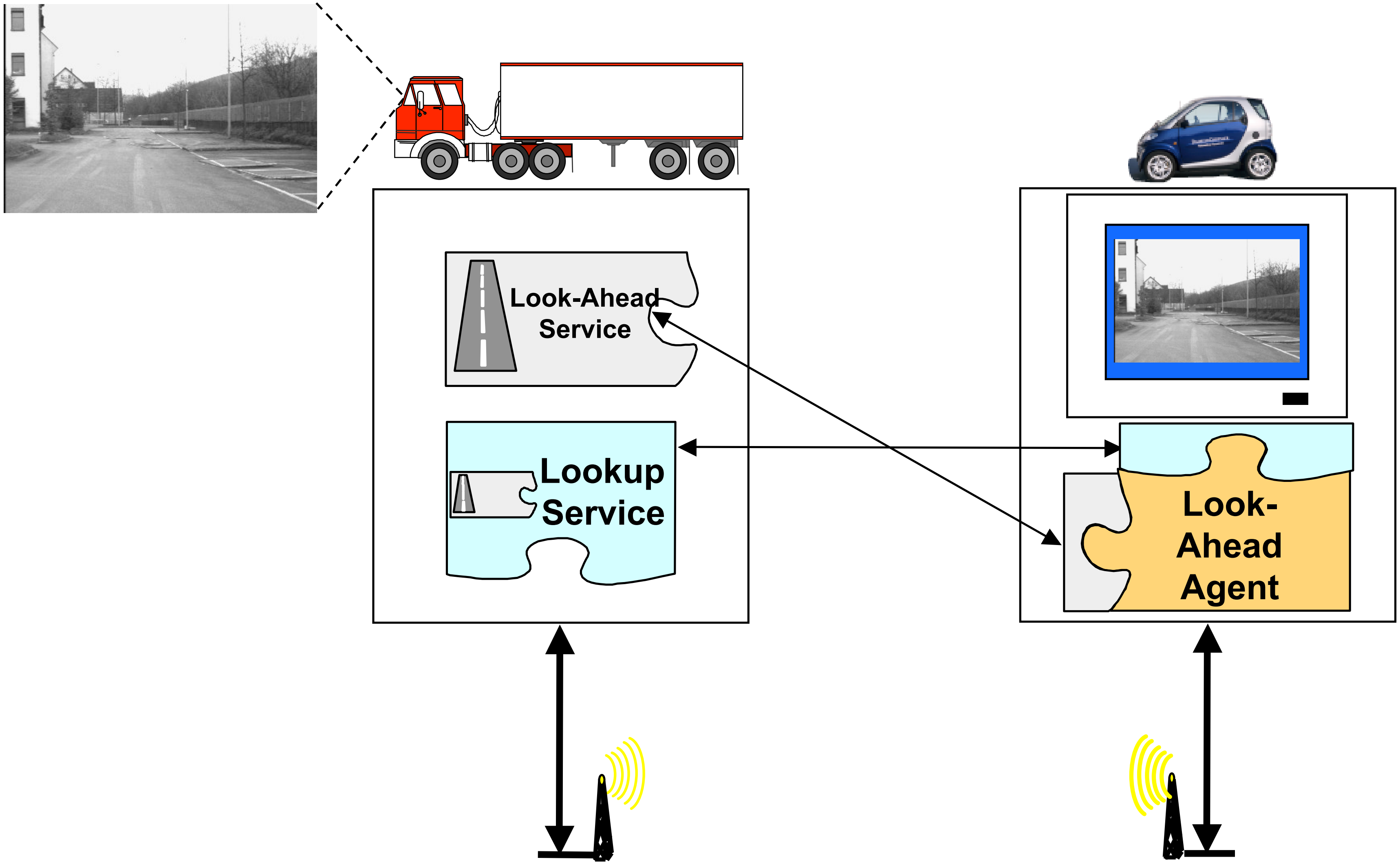
3. Discovery - Finding Lookup Services

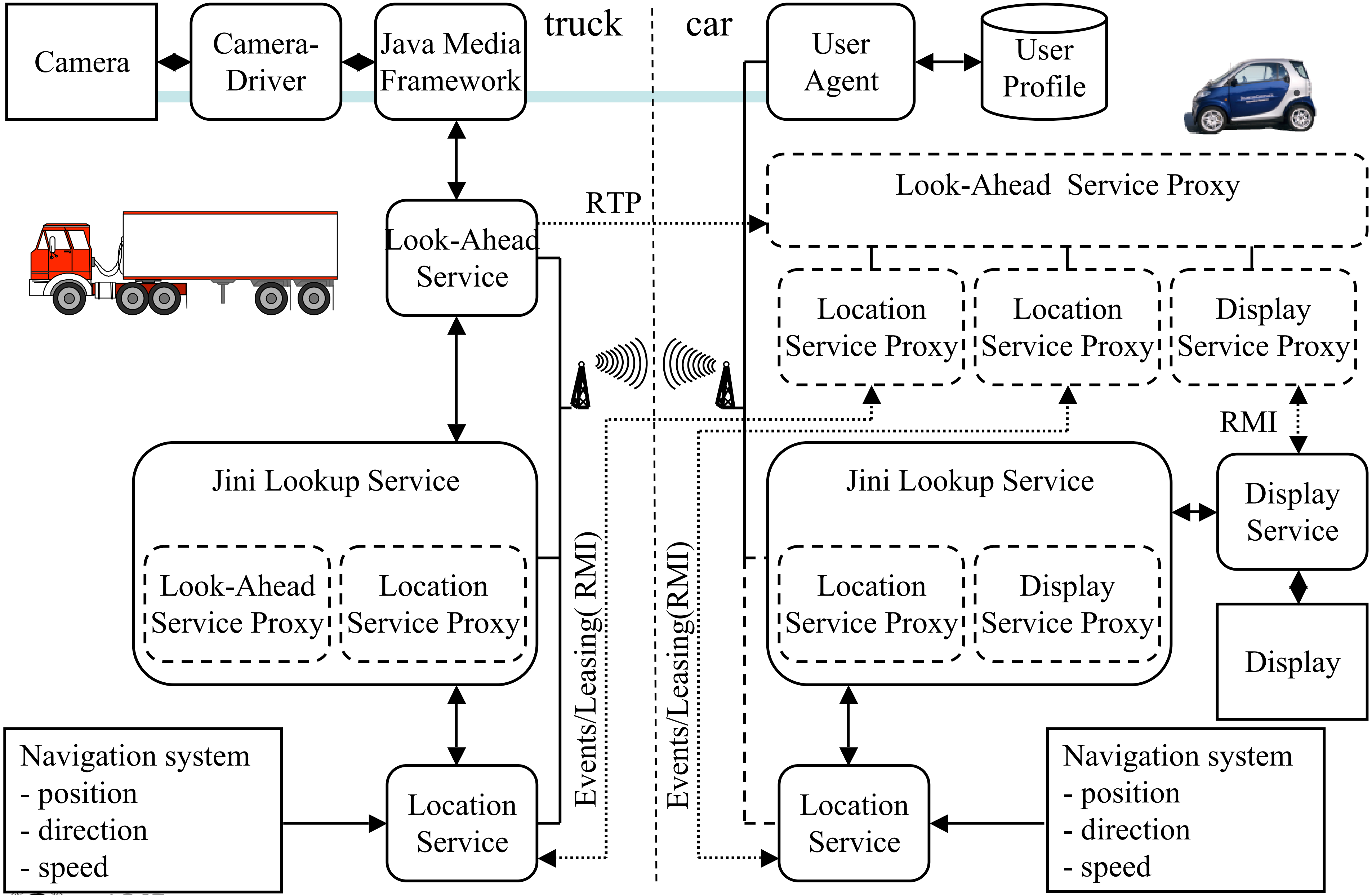
4. Lookup

5. Using a Service



The Demo Scenario: A proactive car-to-car service





Camera
Camera-Driver
Java Media Framework
truck
car
User Agent
User Profile
Look-Ahead Service Proxy
RTP
Look-Ahead Service
Location Service Proxy
Location Service Proxy
Display Service Proxy
RMI
Jini Lookup Service
Look-Ahead Service Proxy
Location Service Proxy
Events/Leasing(RMI)
Events/Leasing(RMI)
Jini Lookup Service
Location Service Proxy
Display Service Proxy
Display Service
Display
Navigation system
- position
- direction
- speed
Location Service
Location Service
Navigation system
- position
- direction
- speed

Programming models

<u>**Object:**</u> Incarnation of an abstract data type (characeristics of o-o: class, inheritance, polymorphism)

<u>**Component:**</u> encapsulated unit of functionality and deployment that interact with other components <u>only</u> via well defined interfaces.

- *Interfaces:* defining sets of operations and the associated data types.
- *Receptacles:* special (required) interfaces that explicitly define the dependencies on other components. On deployment this describes which other components must be present.
- *Binding:* association between one single interface and one single receptacle.
- *Capsule:* container providing the run-time API, e.g. a process

Programming models

Clemens Szyperski, Component Software, ACM Press/Addison-Wesley, England, (1998).

A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.

A component is a unit of independent deployment.
A component is a unit of third-party composition.
A component has no persistent state.

An object is a unit of instantiation; it has a unique identity.
An object has state; this state can be persistent state.
An object encapsulates its state and behavior.

Programming models

What is a service?

<u>Service:</u>

"a mechanism to enable access to one or more capablities, where access is provided using a prescribed interface and is exercised consistently with the constraints and policies as specified by the service description." (OASIS)

"a software system designed to support interoperable machine-to-machine interaction over a network. It has an interface described in a machine-processable format." (W3C)

Properties of a service

- A service can be used as an independent and self-contained entity.
- A service is available within a network.
- Every service has a published interface that is sufficient to use the service.
- The use of services is platform and language independent.
- A service is registered in some directory.
- Binding to a services is dynamic. At design time of an application existence of a respective service is not required. It will be discovered and used dynamically.

Programming models

Actors and Agents

.. components [that] are concurrent objects that communicate via messaging, rather than abstract data structures that interact via procedure calls. ... We call them *actor-oriented* languages.... Actor-oriented languages, like object-oriented languages, are about modularity of software.

Edward A. Lee, UCB, 2004

The term “actors” was introduced in the 1970’s by Carl Hewitt of MIT to describe autonomous reasoning agents.

The term evolved through the work of Gul Agha and others to refer to a family of concurrent models of computation, irrespective of whether they were being used to realize autonomous reasoning agents.

Programming models

Actors are complex, physical, possibly distributed architectural objects that interact with their surroundings through one or more *signal-based* boundary objects called ports.

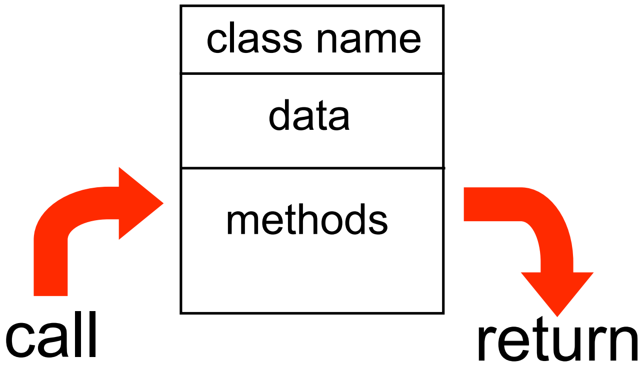
A *port* is a physical part of the implementation of a actor that mediates the interaction of the actor with the outside world. It is an object that implements a specific interface.

Bran Selic, ObjecTime Limited, Jim Rumbaugh, Rational Software Corporation: "Using UML for Modeling Complex Real-Time Systems, March 11, 1998

Objects vs. Actors

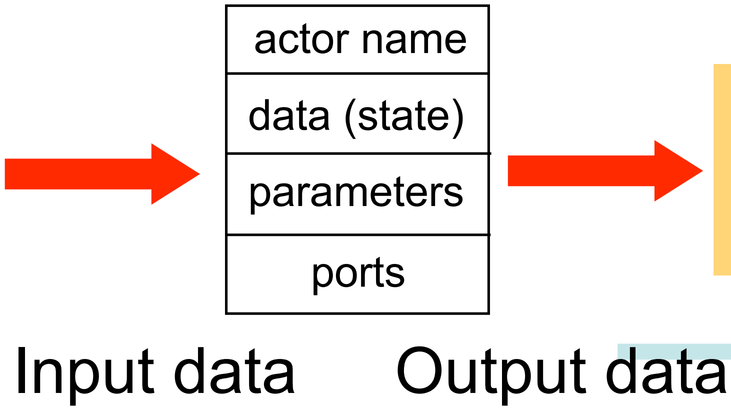
Edward A. Lee, UCB, 2004

Object orientation:



What flows through an object is sequential control

Actor orientation:



What flows through an actor is streams of data

Actors and Agents

Properies of Agents:

- Activity: an agent is an actor
- Automomy: agents behave according a plan
- Social behaviour: ability to communicate (with humans)
- Reactivity: an agent reacts on perceived events
- Proactivity: agents are able to take initiative

Actors and Agents

"Agents are autonomous, computational entitities that can be viewed as perceiving their environment through sensors and acting upon their environment through effectors. To say that agents are computational entities simply means that they physically exist in the form of programs that run on computing devices. To say that they are autonomous means that to some extent they have control over their behavior and can act without the intervention of humans and other systems. Agents pursue goals or carry out tasks in order to meet their design objectives, and in general these goals and tasks can be suplementary as well as conflicting." (Gerhard Weiß)

....will be continued