Freie Universität Berlin

Course introduction & basics of distributed systems

Netzprogrammierung
(Algorithmen und Programmierung V)

Our topics today

1 Motivating example

2 Class organization

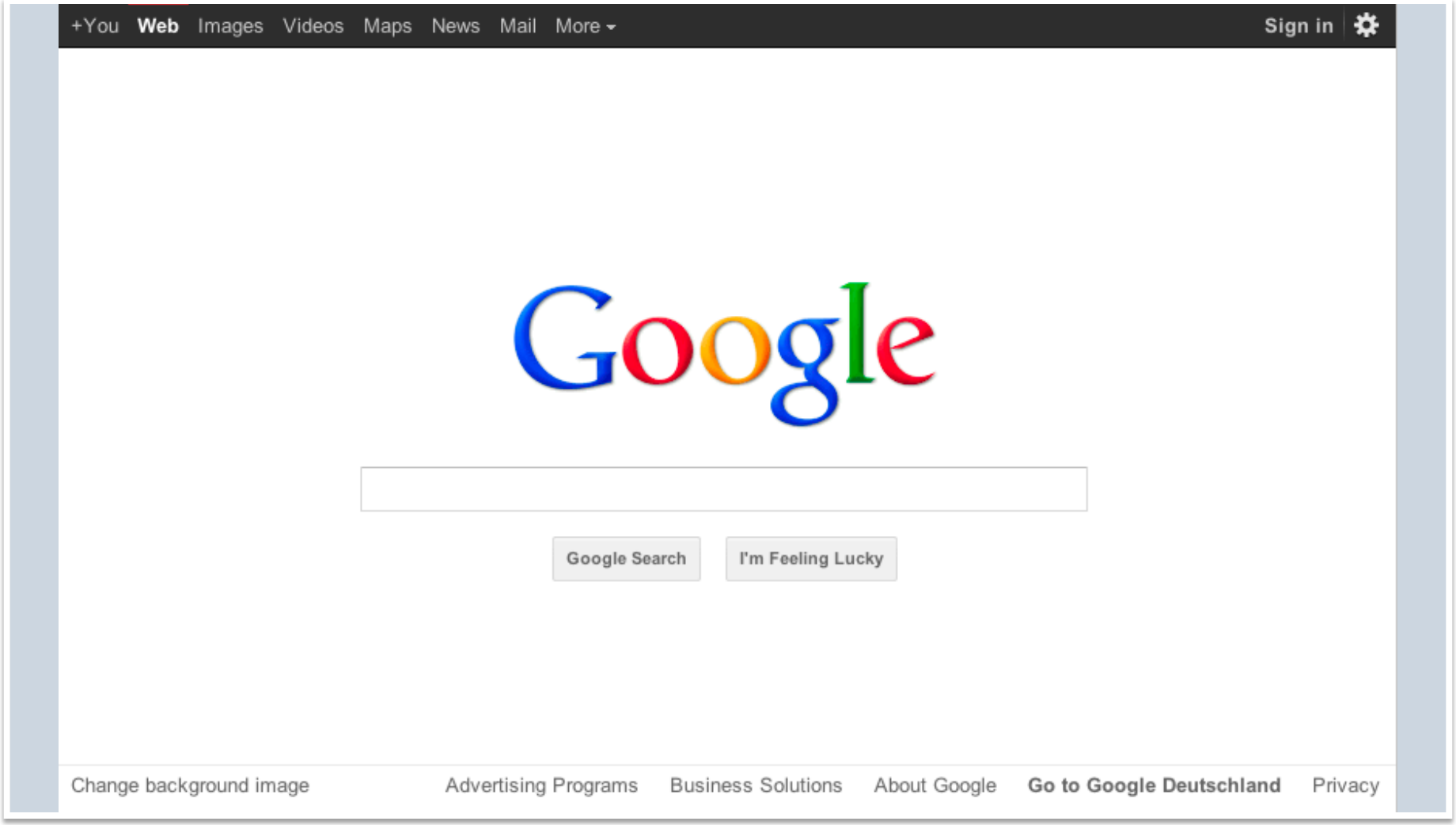
3 Class schedule

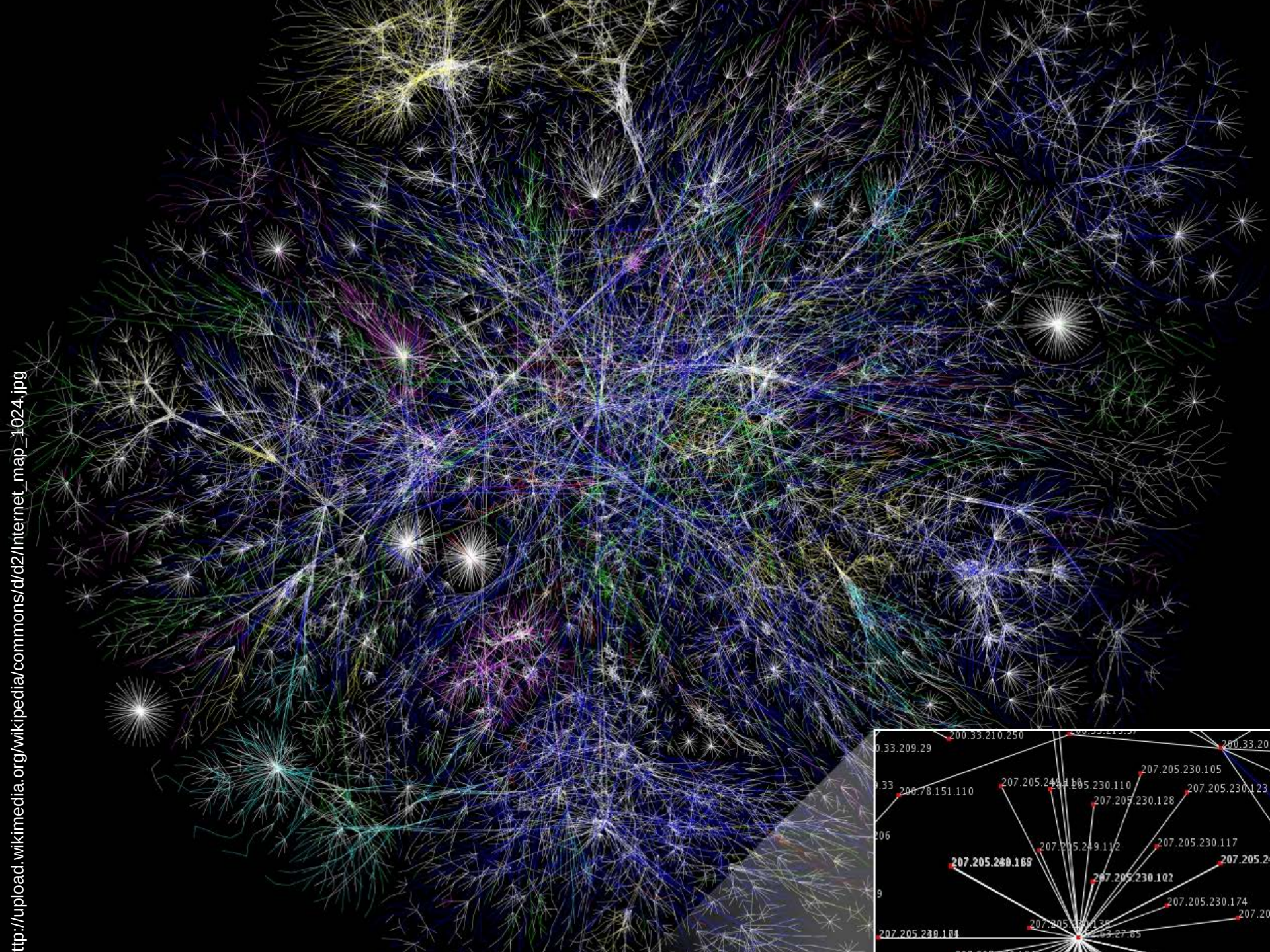
4 Terminology and introduction to distributed systems

5 Levels of supporting distributed systems

6 Brief summary

One perspective





http://upload.wikimedia.org/wikipedia/commons/d/d2/Internet_map_1024.jpg

Web Applications on the Internet

Significance of the internet increased as the population of computers connected to it and the range of software supporting its use has grown

Internet supports a number of distributed applications and services, examples are

- Mail
- Usenet
- World wide web (WWW)

… and today there is a bit more…


flogr
FireAnt
simply
Renkoo
standpoint
oyogi
YEDDA
dabble
SHOUTWIRE
Karma
IBM
kojeet
Findory
gOFFICE
Squeet
wink
wal-mart
blogbeat
vSocial
chuquet
MyFrogs
gumtree
FeedTier
POD2INGER
flickr
Ning
Oddes
CASTPOST
ScoopGO!
FeedBurner
BlogLines
ourmedia
ICEROCKET
Yub.com
EchoSign
Gcast
ajchat
jambo
ROLLYO
ClipShack
XmailHardDrive
LinkedIn
looklater
Noodly
box
Jots
diigo
writely
Simpy
Gtalkr
moo flex
ebay
TRUVEO
egoSurf
pegasus
SQUIDOO
picturecloud
listal
newsvine
Cliqfire

… and even more…

Selected application domains and associated networked applications

Finance and commerce	eCommerce e.g. Amazon and eBay, PayPal, online banking and trading
The information society	Web information and search engines, ebooks, Wikipedia; social networking: Facebook, Twitter.
Creative industries and entertainment	Online gaming, music and film in the home, user-generated content, e.g. YouTube, Flickr
Healthcare	Health informatics, online patient records, monitoring patients
Education	E-learning, virtual learning environments; distance learning
Transport and logistics	GPS in route finding systems, map services: Google Maps, Google Earth
Science	Grid as an enabling technology for collaboration between scientists
Environmental management	Sensor technology to monitor earthquakes, floods or tsunamis

Main drivers

Increasing need for collaboration and connectivity

- Connecting a vast quantities of geographically distributed information and services, such as e-commerce sites, multimedia content, and encyclopedias
- Popularity of social networks, instant messaging or chat rooms is another driver for distributed systems

Increasing availability of different platforms

- Computer networks that incorporate PDAs, laptops, PCs, and servers often offer a better price/performance ratio than centralized mainframe computers
- Selected application components and services can be delegated to run on nodes with specialized processing attributes, such as high-performance disk controllers or large amounts of memory

(Tanenbaum, 1995)

Benefits of distribution

Performance and **scalability**

- Performance of distributed systems should scale up to handle the increased load and capabilities
- Significant performance increases can be gained by using the combined computing power of networked computing nodes

Failure tolerance

- Distributed computing must tolerate partial system failures
- For example, although all the nodes in a network may be live, the network itself may fail
- A common way to implement fault tolerance is to replicate services across multiple nodes and/or networks
- Replication helps minimize single points of failure, which can improve system reliability in the face of partial failures

(Tanenbaum, 1995)

Scope of this course

„In this class you will learn about principles, methods, languages and middleware for developing distributed systems, especially web-based applications.“

We will not talk about

- Theory of computer networks
- Telematics
- Theory of distributed systems
- Design of distributed algorithms
- Design of distributed databases

Course organization

Context of this course

Module is part of the *algorithms and programming slot* in the bachelor program.

Normally in the 5th semester of your study.

The class material is now in English.

This year we slightly changed the structure of this course, and therefore, home assignments might differ from last year‘s course.

Your feedback is appreciated!

Goal of this course

At the end of this course, you should be able to

- Differentiate relevant interaction paradigms such as client/server or peer-to-peer
- Knowing the different levels of support for distributed computing
- Develop distributed software based on local inter-process communication (remote procedure calls) as well as socket-based network communication
- Implement distributed software based on Java RMI
- Knowing middleware technologies and understanding their differences
- Describe the main design principles of cloud computing and its application areas
- Development of web-based, distributed software based on relevant standards

Our team

Instructor: Barry Linnert

Teaching assistants

- Thierry Meurers
- Marco Ziener

Communication:

via forum of KVV course

If you have specific questions regarding a tutorial contact the respective teaching assistant.

General course organization

The lecture takes place each Tuesday, 12 - 14 PM, in room HS Großer Hörsaal (Takustraße 9)

Our website: http://www.inf.fu-berlin.de/w/SE/VorlesungALPVNetzprogrammierung2015

and KVV

Additionally, you have to attend one of the offered labs which take place every Wednesday. **The lab will start next week (2015-10-21)!** The registration is mandatory. Do not switch between labs.

Lab organization

Please check out our webpage/KVV for details.

Lab 1: Wed 08-10 – SR 005/T9 Übungsraum, Thierry Meurers

Lab 2: Wed 10-12 – 046/T9 Seminarraum, Thierry Meurers

Lab 3: Wed 10-12 – SR 009/A6 Seminarraum, Marco Ziener

Lab 4: Wed 12-14 – SR 006/T9 Seminarraum, Thierry Meurers

Lab 5: Wed 14-16 – 049/T9 Seminarraum, Marco Ziener

Lab 6: Wed 16-18 – 055/T9 Seminarraum, Marco Ziener

Grading

Your final grade is only based on the result of your written exam.

But

in order to actively participate in this course, you need to fulfill **ALL** of the following requirements

- you have to submit (n-2) of all assignments that are distributed in the labs,
- you need to get at least 50 % of all points in each assignment,
- you must present at least one assignment,
- the mean (= average) of all your assignments need to be above 60 %.

Organization of labs

Assignments

- This semester we have weekly assignments
- Each assignment is solved by one or a group of two students
- Assignments are published **every Tuesday after the lecture**
  - *First assignment 15-10-20 published in KVV with deadline 15-10-30*
- Deadline for assignments is **Friday 10 AM** mostly of the following week

Typical structure of a lab meeting

Presentation of Assignment 1
(submitted last Friday)

Discussing of Assignment 2
(published last Tuesday)

Preparing Assignment 3
(submission deadline next Friday)

Submission of assignments

Please send your assignments in an electronic format per email to **your** teaching assistant AND for the printed version use the physical mail boxes in the institute!

Your email subject **MUST** have the following structure

```
[ALPV] Übungsblatt XX – Tutorium X - Gruppennummer XX
```

Beispiel

```
[ALPV] Übungsblatt 1 – Tutorium 3 - Gruppe 6
```

In the case of multiple submissions from one group, only the most recent zip file will be marked. Please be sure to include everything necessary within one zip file. Files not included cannot be marked. **Usual rules apply: late submissions get zero.**

The marks for each assignment will be returned by the next lab meeting.

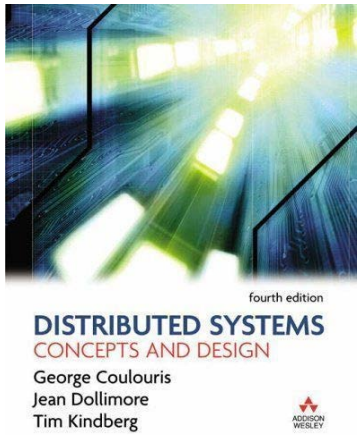
Preliminary lecture schedule

13.10.2015	Introduction and overview of class Technologies for supporting distribution
20.10.2015	Architectures of distributed systems
27.10.2015	Ad hoc network programming (communication over sockets)
03.11.2015 10.11.2015	Structured communication (RPC) Structured communication (RMI)
17.11.2015	Java RMI
24.11.2015 01.12.2015	Indirect Communication I Indirect Communication II
09.12.2015	Distributed Event Based Systems and Complex Event Processing
15.12.2015	Distributed Object and Component middleware (OMG with CORBA Component Model)

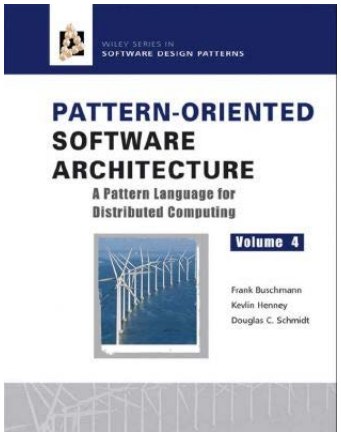
Preliminary lecture schedule

05.01.2016	Service-oriented architectures and web services
12.01.2016	Web application development I: dynamic web applications (e.g., CGI, Servlets, JSP)
19.01.2016	Web application development II: dynamic web applications
26.01.2016	Applications of distributed computing I: Peer to Peer
02.02.2016	Applications of distributed computing II: Cloud computing
09.02.2016	Exam

Recommended literature *(part 1)*



Distributed Systems: Concepts and Design
George Coulouris, Jean Dollimore, Tim Kindberg
4th edition, 2005



Pattern-Oriented Software Architecture Volume 4: A Pattern Language for Distributed Computing

Frank Buschmann, Kevlin Henney, Douglas C. Schmidt

1st edition, 2007

Questions?

Introduction

Basics of distributed systems

Interaction of Processes - two **paradigms**:

- ☛ (ALP 4:) Access to *shared data*
  → **centralized program**

- ☛ (ALP 5:) Communication via *messages*
  → **distributed program**

- ☛ (mixed forms)

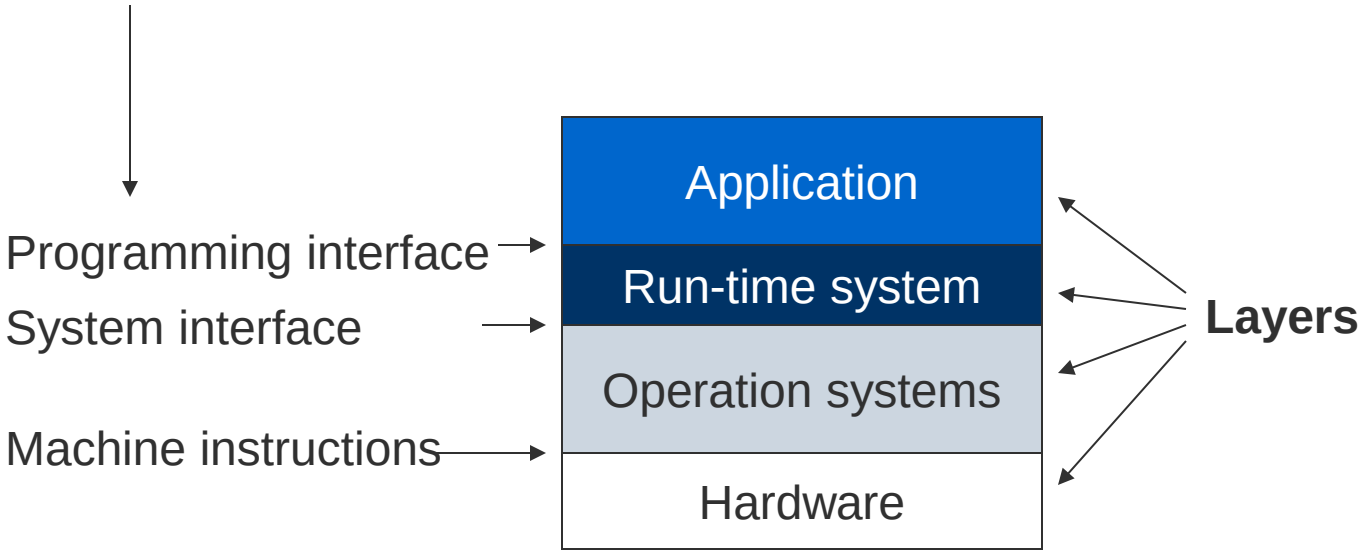
Note: the participating processes need to know about an interaction media

Distribution – Basic Terminology:

- **Local** resource of a process, system, …
- **Remote** resource
- *communication:* interchange of *messages* between processes, systems, …
- *Transport service*
  provides communication operations with particular syntax and semantics (see TI 3)
- *Network* *[will be defined later]*

Distribution and Abstraction

Non-trivial software spans several
abstraction levels of a **functional hierarchy** , *e.g.*



What looks centralized on a higher abstraction level, can be distributed on another layer !

Defining a distributed system

“A distributed system consists of a **collection of autonomous computer** linked by a **computer network** and equipped with distributed system software. Distributed system **software enables** computers **to coordinate their activities** and **to share the resources** of the system – hardware, software, and data – ” (Coulouris et al., 1994)

“[…] so that users perceive the system as a **single, integrated** computing facility.”

“Most computer software today runs in distributed systems, where the interactive presentation, application business processing, and data resources reside **in loosely-coupled computing nodes** and **service tiers connected together by networks**.” (Buschmann et al., 2007)

Characteristics of …

Centralized System

- One component with non-autonomous parts
- Component shared by users all the time
- All resources accessible
- Software runs in a single process
- Single point of control
- Single point of failure

Distributed System

- Multiple autonomous components
- Components are not shared by all users
- Resources may not be accessible
- Software runs in concurrent processes on different processors
- Multiple points of control
- Multiple points of failure

Aspects – Design of distributed systems

Heterogeneity

Openness

Security

Scalability

Failure handling

Transparency

Heterogeneity

(= **variety and difference**) apply to the following

- Networks
- Computer hardware
- Operating systems
- Programming languages
- Implementation by different developers

Possible solutions that address heterogeneity

- **Middleware**
  - Software layer that provides a programming abstraction as well as masking the heterogeneity of the underlying networks, hardware, operating system and programming language
- **Virtual machine**
  - Approach to make code executable on any hardware, for example the Java compiler produces code for the Java virtual machine

Openness

Openness is concerned with **extensions and improvements** of distributed systems. It is primarily determined by the degree to which new resource-sharing services can be added and be made available for use by a variety of client programs.

Requirements

- Key interfaces of components need to be published
- New components have to be integrated with existing components
- Differences in data representation of interface types on different processors (of different vendors) have to be resolved by published standards

Request for comments (RFC) - http://www.ietf.org/rfc.html

- Document collection published by the Internet Engineering Task Force (IETF) describing methods, behaviors, research, or innovations applicable to the working of the Internet and Internet-connected systems

Security

Security for information resources has three components

- **Confidentiality** – protection against disclosure to unauthorized individuals
- **Integrity** – protection against alteration or corruption
- **Availability** – protection against interference with the means to access the recourses

Examples

- Doctors request access to health care information of their patients
- Users send credit card numbers across the Internet

What are security threads here? And how can we solve them?

Scalability

A system is described as scalable if it will remain **effective when** there is a **significant increase** in the number of resources and the number of users.

Adaption of distributed systems to

- accommodate more users
- respond faster (this is the hard one)

Usually done by adding more and/or faster processors.

Components should not need to be changed when scale of a system increases.

Design components in a way that they are scalable!

Failure handling

Hardware, software and networks fail!

Distributed systems must maintain availability even at low levels of hardware/software/network reliability.

Fault tolerance is achieved by

- **Recovery**
- **Redundancy**

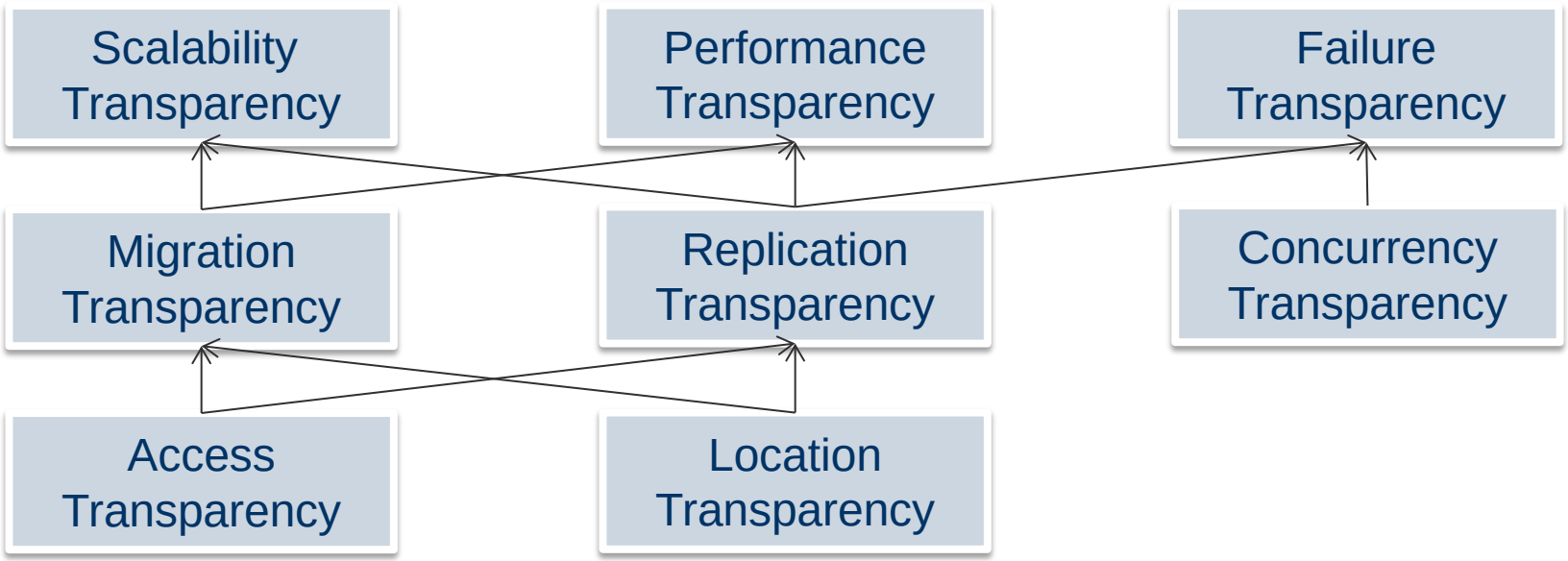
Recovery from failures involves the design of software so that the state of permanent data can be recovered or ‘rolled back’ after a server has crashed.

Redundancy means that services can be made to tolerate failures by the use of redundant components.

Transparency

Distributed systems should be **perceived** by users and application programmers **as a whole** rather than as a collection of cooperating components.

Transparency has different dimensions that were identified by ANSA Reference Manual (ANSA 1998). These represent various properties that distributed systems should have.



Transparency (cont.)

Access transparency

- Enables local and remote resources to be accessed using identical operations
- Examples: File system operations in NFS, navigation in the Web, SQL Queries

Location transparency

- Enables resources to be accessed without knowledge of their physical or network location (for example, which building or IP address)
- Examples: File system operations in NFS, pages in the Web, tables in distributed databases

Concurrency transparency

- Enables several processes to operate concurrently using shared resources without interference between them
- Examples: NFS, automatic teller machine, database management system

Transparency (cont.)

Replication transparency

- Enables multiple instances of resources to be used to increase reliability and performance without knowledge of the replicas by users or application programmers
- Examples: Distributed DBMS, Mirroring Web Pages

Failure transparency

- Enables the concealment of faults, allowing users and application programs to complete their tasks despite the failure of hardware or software components
- Examples: Database Management System

Transparency (cont.)

Migration transparency

- Allows the movement of resources and clients within a system without affecting the operation of users or programs
- Examples are NFS, Web pages

Performance transparency

- Allows the system to be reconfigured to improve performance as loads vary
- Examples are distributed make

Scaling transparency

- Allows the system and applications to expand in scale without change to the system structure or the application algorithms
- Examples are World Wide Web, distributed database

Examples of distributed systems

Clusters [3]

- Enable coordinated sharing and aggregation of distributed, autonomous, heterogeneous resources based on users' QoS (Quality of Service) requirements
- Generally used in high performance scientific engineering and business applications

Grids [4]

- Are commonly used to support applications emerging in the areas of e-Science and e-Business, which commonly involve geographically distributed communities of people who engage in collaborative activities to solve large scale problems and require sharing of various resources such as computers, data, applications and scientific instruments

Other examples of distributed systems

P2P (Peer-to-Peer) networks [5]

- Decentralized distributed systems, which enable applications such as file-sharing, instant messaging, online multi- user gaming and content distribution over public networks.

Distributed storage systems

- Distributed storage systems such as NFS (Network File System) provide users with a unified view of data stored on different file systems and computers which may be on the same or different networks

Challenges of distribution

Inherent complexities

- Components of a distributed system often reside in separate address spaces on separate nodes, so inter-node communication needs different mechanisms, policies, and protocols
- Synchronization and coordination is more complicated since components may run in parallel and network communication can be asynchronous and non-deterministic
- Networks that connect components introduce additional forces, such as latency, jitter, transient failures, and overload

Accidental complexities

- Limitations with software tools and development techniques, such as non-portable programming APIs and poor distributed debuggers
- New layers of distributed infrastructure are conceived and released, not all of which are equally mature or capable, which complicates development, integration, and evolution of working systems

(Schmidt, et al. 2002)

Challenges of distribution (cont.)

Inadequate methods and techniques

- Popular software analysis methods and design techniques such as UML (Dennis et al., 2004) have focused on constructing single-process, single-threaded applications with ‘best-effort’ QoS requirements
- Development of high-quality distributed systems (e.g., video-conferencing, air traffic control systems), has been left to the expertise of skilled software architects and engineers

Continuous re-invention and re-discovery of core concepts and techniques

- Software industry has a long history of recreating incompatible solutions to problems that have already been solved
- If effort had instead been focused on enhancing a smaller number of solutions, developers of distributed system software would be able to innovate more rapidly by reusing common tools and standard platforms and components

(Schmidt, et al. 2002)

Basics of distributed systems

Technologies for supporting distribution

Levels of support for distributed computing

Ad hoc network programming

Structured communication

Middleware

(Buschmann, et al., 2007)

Ad hoc network programming

Inter-process communication (**IPC**) mechanisms, such as ***shared memory***, ***pipes***, and ***sockets***, allow distributed components to connect and exchange information

IPC mechanisms enable components from different address spaces to cooperate with one another

Drawbacks when developing distributed systems only using ad hoc network programming support

- Using **sockets** directly within application code tightly couples this code to the socket API -> porting this code to another IPC mechanism or redeploying components to different nodes in a network becomes a costly manual programming effort
- Programming directly to an IPC mechanism can also cause a paradigm mismatch, for example, local communication uses object-oriented classes and method invocations, whereas remote communication uses the function-oriented socket API and message passing

(Buschmann, et al., 2007)

Structured communication

Overcomes limitations with ad hoc network programming by not coupling application code to low-level IPC mechanisms

Offers higher-level communication mechanisms to distributed systems

Encapsulates machine-level details, such as bits and bytes and binary reads and writes

Provides a programming model that embodies types and a communication style closer to their application domain

Significant examples of structured communication are ***Remote Procedure Call*** (RPC) platforms

(Buschmann, et al., 2007)

RPC platforms

Allow distributed applications to cooperate with one another much like they would in a local environment

Invoke functions on each other, pass parameters along with each invocation, and receive results from the functions they called

Shields functions from details of specific IPC mechanisms and low-level operating system APIs

For more complex distributed systems structured communication does not fulfill all properties needed

- Location-independence of components
- Flexible component (re)deployment
- Integration of legacy code
- Heterogeneous components

(Buschmann, et al., 2007)

Middleware

Distribution infrastructure software that resides between an application and the operating system, network, or database underneath it

Middleware allow application developers to focus on their primary responsibility: implementing their domain-specific functionality

Different parties developed technologies for distributed computing

- companies such as Microsoft, IBM, and Sun
- consortia, such as the Object Management Group (OMG) and the World Wide Web Consortium (W3C)

(Buschmann, et al., 2007)

Middleware technologies

Distributed object computing middleware

Component middleware

Publish/subscribe middleware

Service-oriented architectures and Web Services

(Buschmann, et al., 2007)

Distributed Object Computing Middleware

Emerged in late 1980s and early 1990s

Represented the confluence of two major information technologies:

- RPC-based distributed computing systems
- object-oriented design and programming

Used object-oriented techniques to distribute reusable services and applications efficiently, flexibly, and robustly over multiple, often heterogeneous, computing and networking elements

Examples

- CORBA 2.x
- Java RMI

(Buschmann, et al., 2007)

Component Middleware

Starting in the mid to late 1990s to overcome limitations (we will talk about it in the following lecture in detail) of DOC middleware

Allows a group of cohesive component objects to interact with each other through multiple provided and required interfaces and defines standard runtime mechanisms needed to execute these component objects in generic applications servers

component middleware also often specifies the infrastructure to package, customize, assemble, and disseminate components throughout a distributed system

Examples

- Enterprise JavaBeans
- CORBA Component Model (CCM)

(Buschmann, et al., 2007)

Message-Oriented Middleware

RPC platforms, DOC middleware, and component middleware are all based on a request/response communication model

- Requests flow from client to server and responses flow back from server to client

<u>Problem:</u> certain types of distributed applications are not well-suited certain aspects of the request/response communication model

- Synchronous communication between the client and server, which can underutilize the parallelism available in the network and end systems
- Designated communication, where the client must know the identity of the server, which tightly couples it to a particular recipient
- Point-to-point communication, where a client talks with just one server at a time, which can limit its ability to convey its information to all interested recipients

Message-Oriented Middleware are mostly proprietary systems.

(Buschmann, et al., 2007)

Service-Oriented Architectures

‘SOA’ was originally coined in the mid-1990’s

Generalize interoperability middleware standards available at the time

Is a style of organizing and utilizing distributed capabilities that may be controlled by different organizations or owners

Provides a uniform means to offer, discover, interact with and use capabilities of loosely coupled and interoperable software services to support the requirements of the business processes and application users

Includes protocols or specifications such as

- SOAP (Simple Object Access Protocol)
- Web Services and WSDL (Web Service Description Language)

(Buschmann, et al., 2007)

Summary

We talked about

- Typical application areas for distributed systems
- Terminology and definition for distributed systems
  - distributed program vs. distributed system
- Centralized vs. distributed systems
- Design principles of distributed systems
- Technologies for supporting distribution
  - Ad hoc network programming
  - Structured communication
  - Middleware

Questions

- What is a distributed system? Name two typical example application areas.
- Discriminate the characteristics of decentralized systems from centralized systems.
- Name and briefly define/describe six typical design principles of distributed systems. How can they be implemented in distributed systems?
- Describe four typical challenges of distribution.
- Discriminate ad hoc network programming, structured communication and middleware. What are the characteristics and benefits?
- Describe four middleware technologies and name a concrete example middleware product / standard for each.

Next lecture:

Communication in and architecture of distributed systems

References

Dennis, B. Haley Wixom, D. Tegarden: Systems Analysis and Design with UML Version 2.0: An Object-Oriented Approach, John Wiley & Sons, 2004

George Coulouris, Jean Dollimore, Tim Kindberg: Distributed Systems: Concepts and Design. 4th edition, Addison Wesley, 2005

Frank Buschmann, Kevlin Henney, Douglas C. Schmidt: Pattern-Oriented Software Architecture Volume 4: A Pattern Language for Distributed Computing, Wiley, 2007

[3] R. Buyya (editor), High Performance Cluster Computing, Prentice Hall, USA, 1999

[4] I. Foster and C. Kesselman (editors), The Grid: Blueprint for a Future Computing Infrastructure, Morgan Kaufmann Publishers, USA, 1999. 5.

[5]R. Subramanian and B. Goodman (editors), Peer-to-Peer Computing: Evolution of a Disruptive Technology, Idea Group Inc., Hershey, PA, USA, 2005.