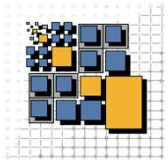
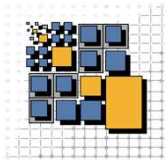


DSG-DSAM-M 2024/25:
- Primer -
- Conceptual Foundations -

Dr. Andreas Schönberger

Lehrstuhl für Praktische Informatik
Fakultät WIAI
Otto-Friedrich-Universität Bamberg



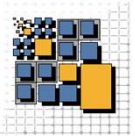
- Primer -

Lehrstuhl für Praktische Informatik
Fakultät WIAI
Otto-Friedrich-Universität Bamberg

About Andreas Schönberger

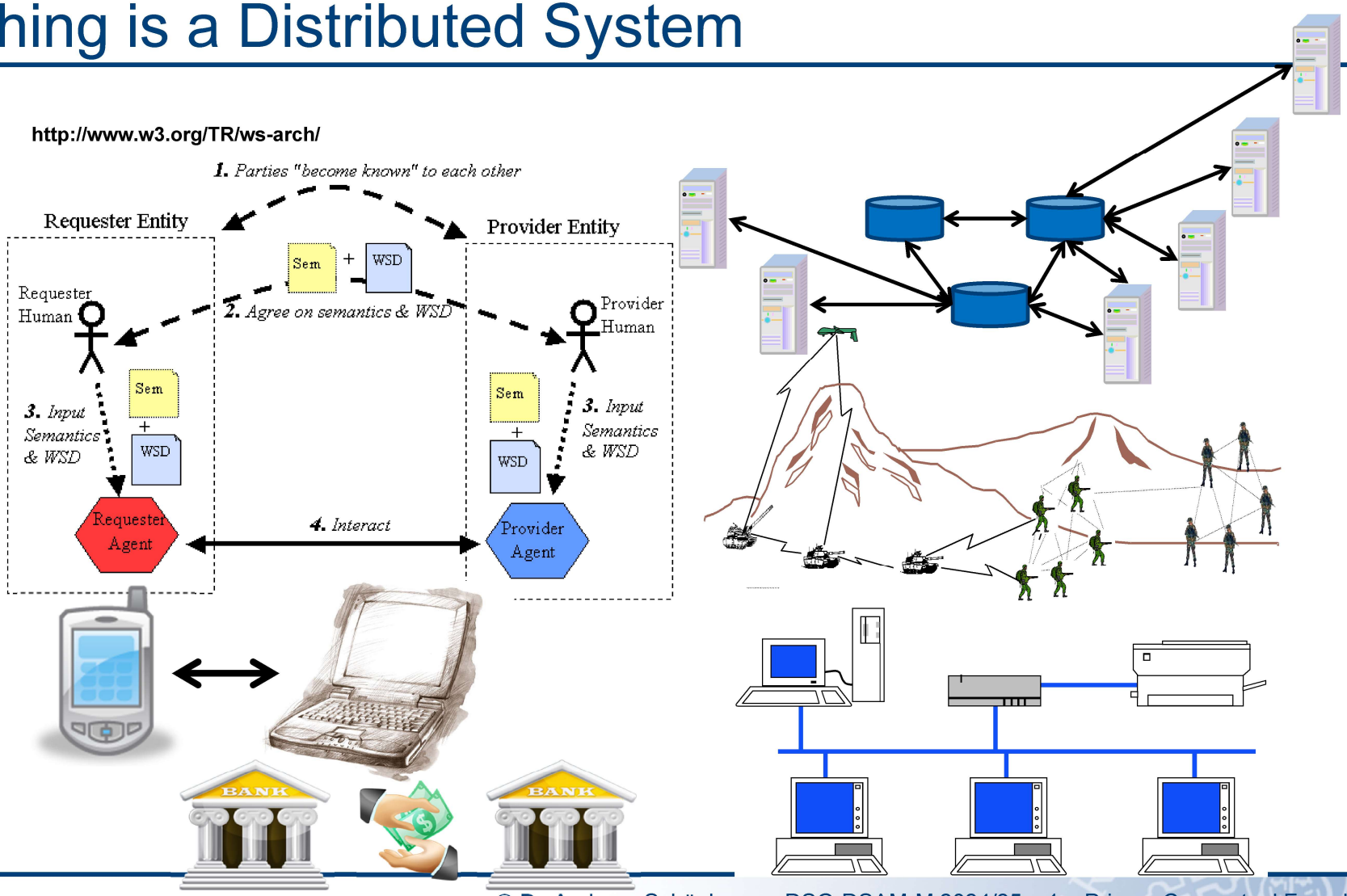
- Born 1979, lives in Bamberg with his family (two children)
- PhD in Computer Science at University of Bamberg
- Previously, trainer and software architect for Java EE Solutions, and architecture consultant at Siemens Corporate Technology
- Founder and CEO of Lion5 GmbH in Bamberg
  - Engages in industrial software platform projects
  - Develops cloud-based services
- Supervisor for more than 20 diploma / master theses
- Author and co-author of more than 30 scientific papers
- Thinks Lion5 is the one and only
  (Disclaimer: this is my personal opinion)
- Thinks Bamberger Beer is the one and only
  (Disclaimer: this is my personal opinion)
- Connect on Facebook, LinkedIn

Organization

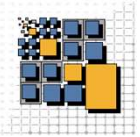


- Who?
  - Lectures 
  - Labs, Assignments (#=2), Tool Introductions 
  - Oral Examination  + 
- Where and When?
  - Lectures: Fridays, 10:15-11:45 am, watch the course calendar
  - Labs and Tool Introductions: Watch VC course
  - Assignment work: self-organized, updates will be provided via VC course
- Support
  - Consulting hours:
    - Andreas Schönberger: get in touch via andreas.schoenberger@uni-bamberg.de
    - Robin Lichtenthäler: https://www.uni-bamberg.de/pi/team/lichtenthaeler-robin/
  - Online: VC course forums, {robin.lichtenthaeler | andreas.schoenberger}@uni-bamberg.de
- More information
  - DSG Homepage: http://www.uni-bamberg.de/pi/
  - VC course: https://vc.uni-bamberg.de/course/view.php?id=70952

Anything is a Distributed System



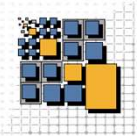
Aims of the Course I



- Aims
  - Understand the characteristics of distributed systems and middleware
  - Know relevant technologies and standards in the field and be able to combine some of these to develop basic middleware solutions.
  - Be able to discuss the benefits and drawbacks of distributed system architectures and middleware technologies.

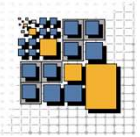
- How will we do that?
  - Domain overview and knowledge through *lectures*
  - Understanding through *hands-on examples* and *discussions*
  - Implementation capabilities through *assignments*

Aims of the Course II



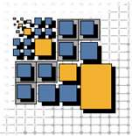
https://www.youtube.com/watch?v=L_E-mHo1Xcs

Who Asks for that?



- Industry
  - _NO_ professional software development without distributed systems knowledge
  - Candidates with theoretical background, analytical skills AND hands-on experience wanted
  - Doers wanted, not windbags
  - A lot of DSG graduates have great jobs today at great companies such as Lion5, small consultancies, but also SAP, Datev, Siemens, Bosch, Allianz …
- Academia
  - Research methods frequently call for prototypic implementations
  - Distributed Systems and, in particular, Cloud Computing in research focus
  - A lot of DSG graduates do their PhD studies now
- Yourself?
  - Still a lot of unsolved problems in a rapidly innovating area
  - Focus on theory and practice possible
  - Good starting point for theses
  - Check out scientific work…find out about interest in PhD studies?
    ➔ Apply for a research oriented thesis or an industry thesis

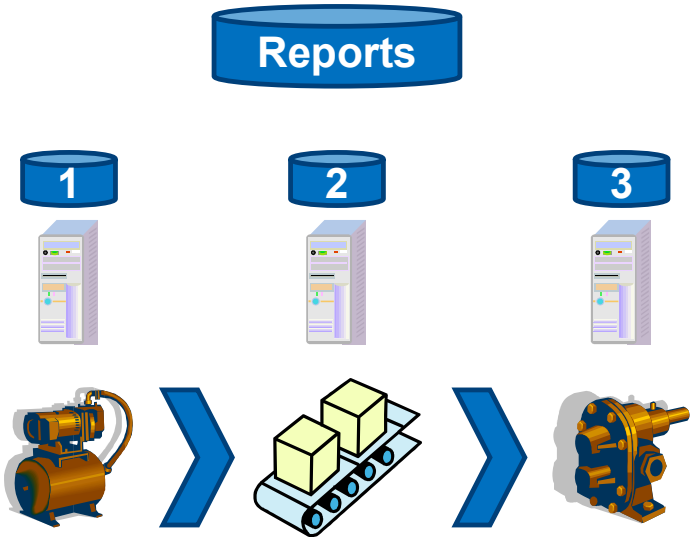
Who Asks for that? – The Production Site Case



The problem:

For a large production site (simplified view) local data storages (1, 2 and 3) had to be updated in a consistent manner and a reports database had to be concurrently written.

Yet, the development team did not implement distributed transactions and mutual exclusion!



The effect:

An expert consultant had to review the control mgt. system, parts of the system had to be reimplemented and the original delivery date was delayed by half a year.

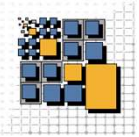
Travel and consulting cost	60	k$
Reimplementation cost	140	k$
Delay penalty	6.5	M$

Your potential role in the game

- System/Software architect
- Expert consultant
- Project manager

This course is at the heart of your future professional life!

How to Succeed in this Course

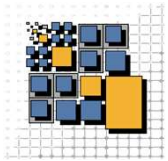


- Computer Science means actively trying out/applying theories, algorithms, modeling and programming languages.
- Practical computer science puts the emphasis on application scenarios, architecture development and software technologies

➔This course is not about learning item lists by heart

- So,…
  - Check presented material against sample scenarios
  - Try out, modify, recompile, test sample code
  - Ask and discuss

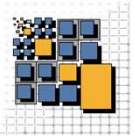
  … on a weekly basis!
- You don't have to be a specialist in Spring, …, Cloud, Databases, but you must be willing to spend some time on it



- Conceptual Foundations -

Lehrstuhl für Praktische Informatik
Fakultät WIAI
Otto-Friedrich-Universität Bamberg

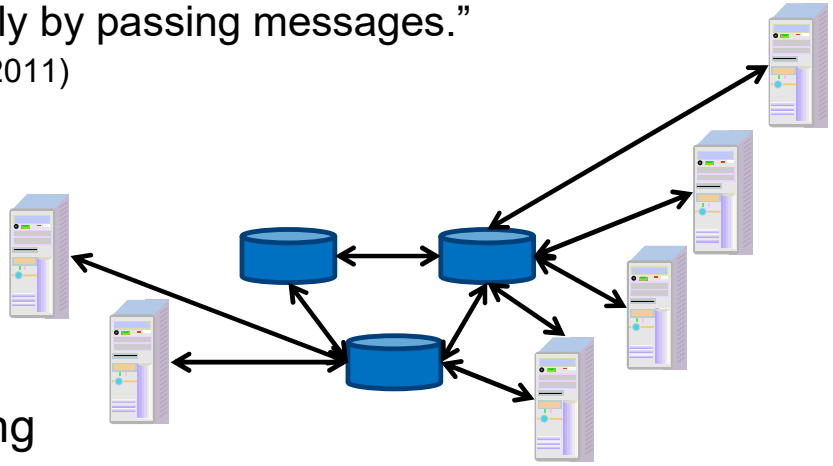
Distributed Systems



- Definition:

  “A distributed system is a collection of independent computers that appears to its users as a single coherent system.”
  (Tanenbaum, *Distributed Systems*, Prentice Hall, 2016)

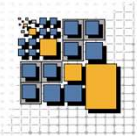
  “We define a distributed system as one in which hardware or software components located at networked computers communicate and coordinate their actions only by passing messages.”
  (Coulouris, Dollimore, Kindberg, *Distributed Systems*, Addison Wesley, 2011)



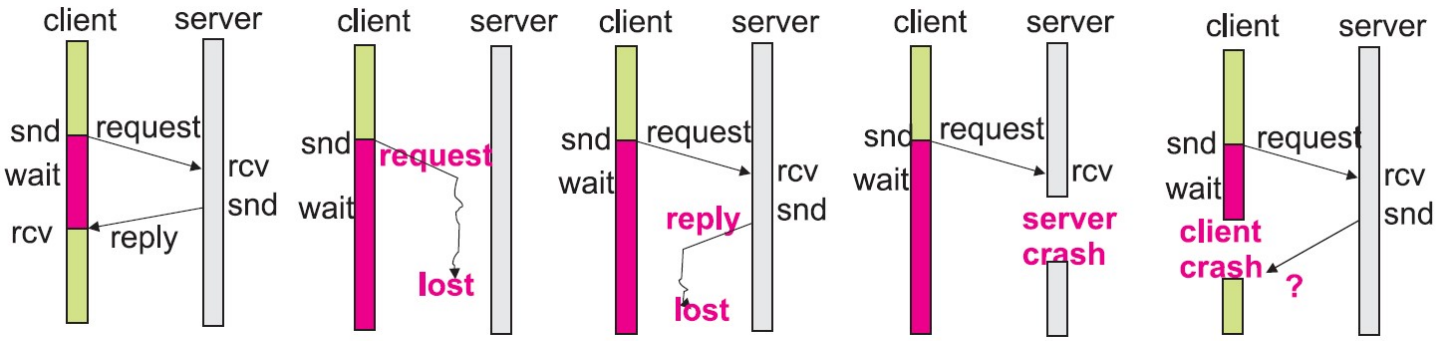
➔ Distributed Computing =

“The task of engineering, developing and running programs on distributed systems”

Influence Factors on Distributed Computing

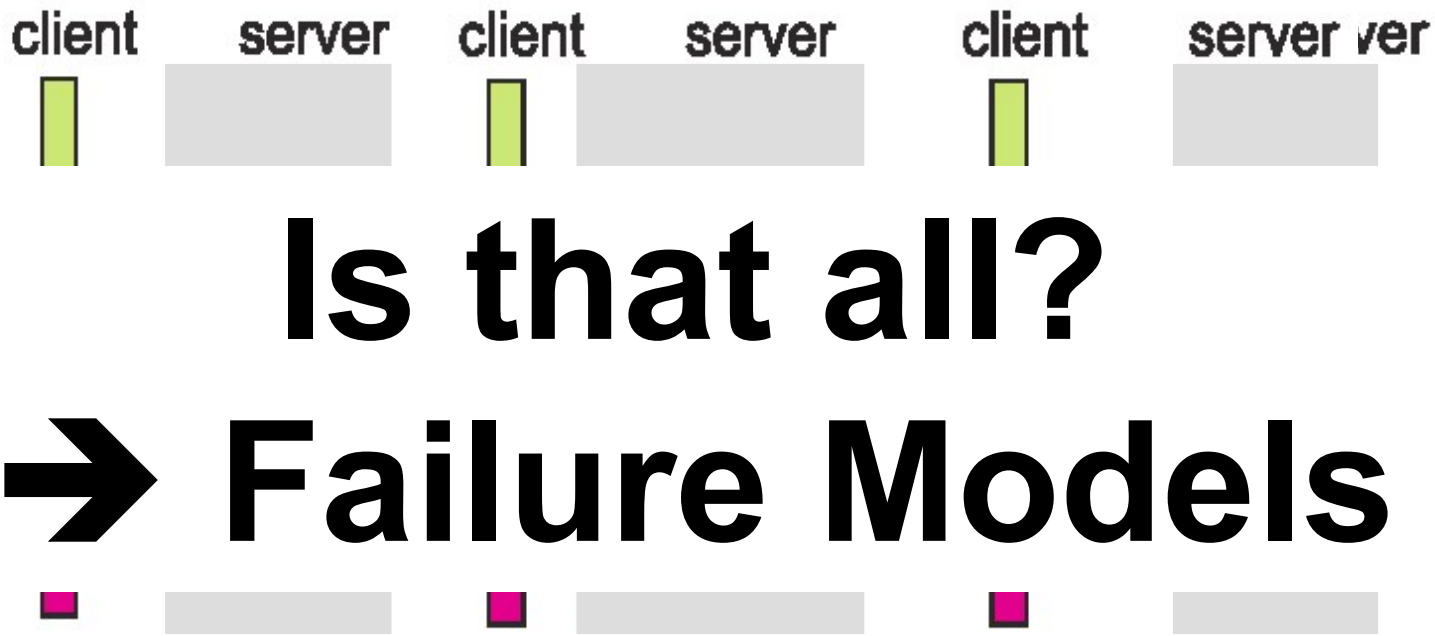


- Core characteristics of distributed systems determine the programming paradigm
  - Autonomous Entities / Partial Failures
  - No Global Time
  - No Global Memory
  - Communication Errors
  - Heterogeneity (technical, semantical)
  - Complex Associations (dynamic bindings, multi-party)
- How do those characteristics influence the scenario below?



Visualization by courtesy of Guido Wirtz

Typical Errors



Is that all?

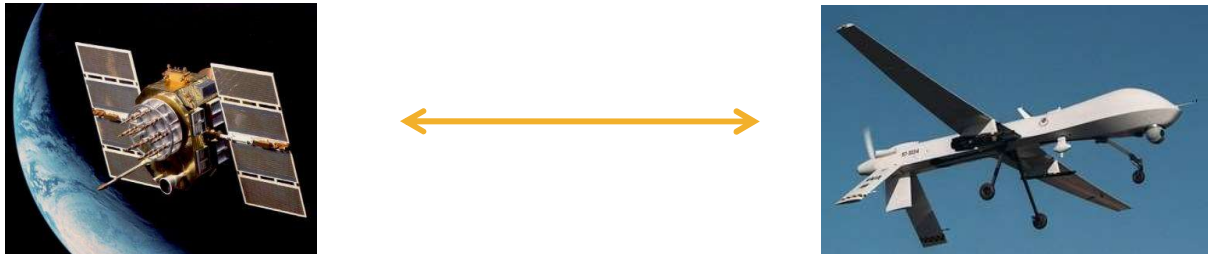
➔ Failure Models

- But also local problems:
  - Synchronization
  - Scheduling
  - …

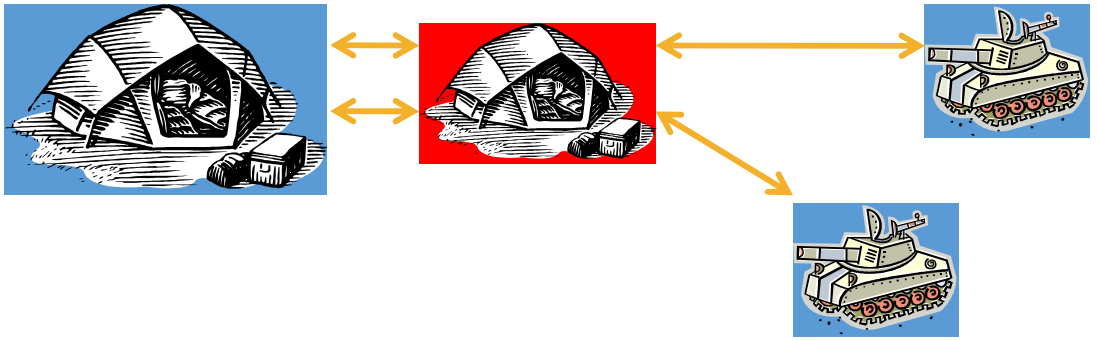
Visualization by courtesy of Guido Wirtz

Failure Models

- Make Assumptions about your Environment
- Examples
  - Resource Discipline: Reservation of cinema tickets
  - Spoiling: Control of UAVs

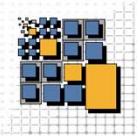


  - Man-In-The-Middle:



  - Similarly:
    - Activation of email account
    - Electronic tax declaration with personal certificate…

Use Distributed Systems in spite of Complexity?

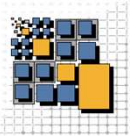


- You have to, the potential is too big!
  - → Flexibility, Robustness, Availability, Cost Savings…
  - Connectivity for actors of all sizes, data centers and mobile devices
  - Integrate existing infrastructure and applications (EAI)
  - Implement business processes across different company locations (EAI)
  - Implement business processes across company boundaries (B2Bi)
  - Only option for really dependable systems

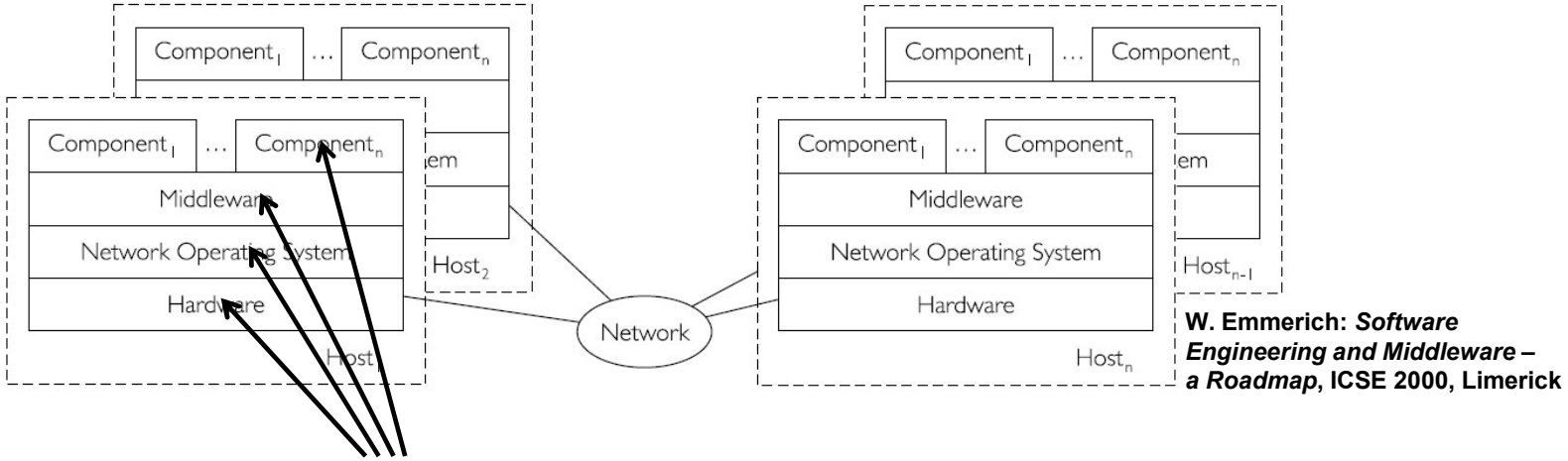
➔Almost every system is a distributed system!

BEWARE: There's no such thing like a free lunch!

Middleware to the Rescue



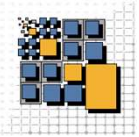
- Middleware is the classical means to manage the complexity of distributed systems, but what is middleware?
- Middleware is the software between
  - …application and operating system (local view)
  - …service user and service provider (global view)



W. Emmerich: *Software Engineering and Middleware – a Roadmap*, ICSE 2000, Limerick

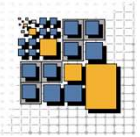
Classical Layering Strategy

Typical Middleware Services



- Middleware differs in which of the below services are offered
  - Naming (Local vs. Remote References; Reference Injection)
  - Transactions
  - Persistency
  - Security
  - Lifecycle Management
  - Scalability
  - Replication (Consistency vs. Availability)
  - Interoperability
  - Vertical/Horizontal Clustering
- Middleware differs in how services are offered
  - Explicit Use
  - Transparent Use

Sample Discussion: Clustering



- Aims
  - Failover
  - Load-Balancing
  - Resource Usage
  - Response Time
  - Scalability
- Implementation Options
  - Vertical: Multiple services on the same machine
  - Horizontal: Multiple services on multiple machines

Discuss: Which option satisfies which aim?

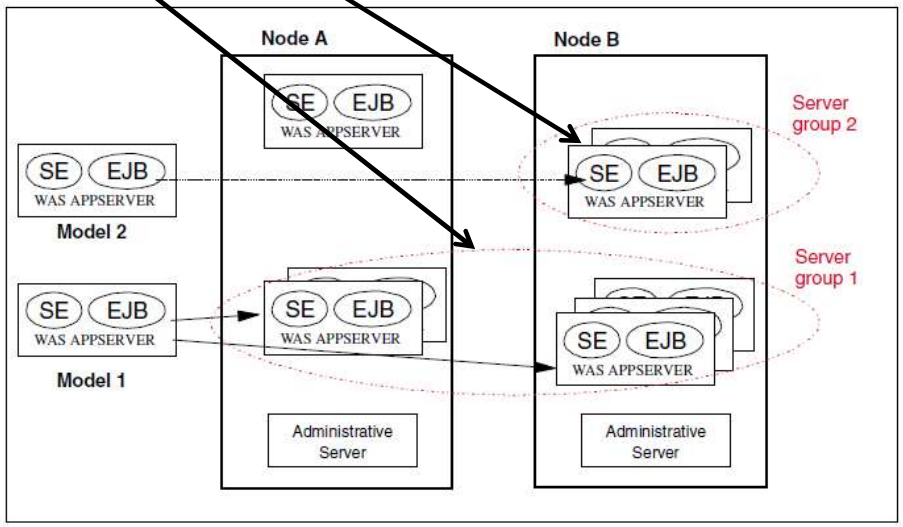
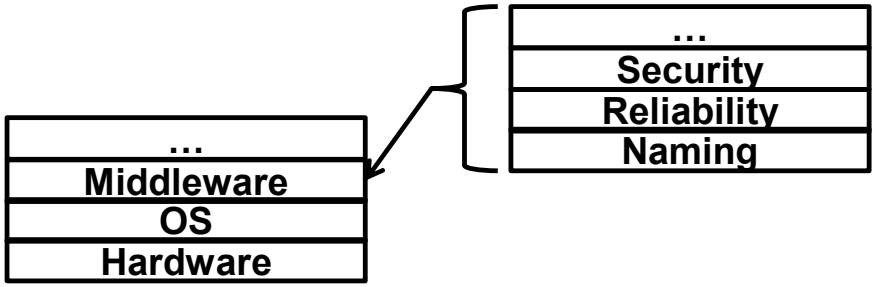


Figure 3. Models and clones

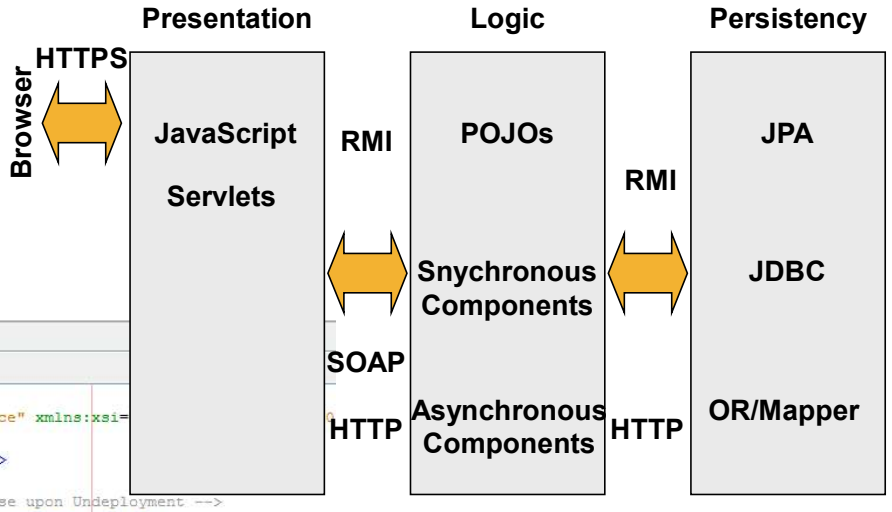
IBM Redbooks:
WebSphere Scalability: WLM and Clustering

How to Provide Middleware Services I

- Layering
  - ➔ Precise assumptions needed!



- N-Tier-Architectures and Containers

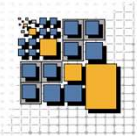


- Deployment Descriptors

persistence.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<persistence version="1.0" xmlns="http://java.sun.com/xml/ns/persistence" xmlns:xsi=
        <persistence-unit name="ejb3DemoEJB">
                <jta-data-source>jdbc/ejb3DemoRoomMgt</jta-data-source>
        <properties>
                <!-- Creates relations upon deployment and deletes these upon Undeployment -->
                <property name="toplink.ddl-generation" value="drop-and-create-tables"/>
                <property name="toplink.logging.logger" value="DefaultLogger"/>
                <property name="toplink.logging.level" value="FINE"/>
                <property name="toplink.logging.thread" value="true"/>
                <property name="toplink.logging.exceptions" value="true"/>
                <property name="toplink.create-ddl-jdbc-file-name" value="create_ejb3Demo.jdbc"/>
            <property name="toplink.drop-ddl-jdbc-file-name" value="drop_ejb3Demo.jdbc"/>
        </properties>
        </persistence-unit>
</persistence>
```

How to Provide Middleware Services II



Distinguish between synchrony and asynchrony…

- as a system classification:
  - A distributed system is synchronous iff
    - known upper time for message transmission
    - known upper time for clock shift
    - known upper time for processing jobs

  ➔ asynchronous otherwise

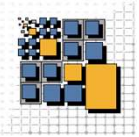
see Prof. Mendler's courses

- as a communication quality:
  - synchronous: sender blocks on call until transmission is done
  - asynchronous: sender (potentially) is done before receive event fires

really implementable? → see next slide

- as a classification of interaction:
  - synchronous: sender waits for a reply
  - asynchronous: after transmission, the sender does something else

Excursus: Coordinated Attack Problem

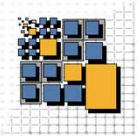


Consider this:

- Two allied generals occupy a fortified city
- Both have agreed to attack the city, but they have not agreed upon time
- The attack will only be successful if both generals attack at exactly the same time
- The ONLY way of communication is sending a messenger
  (NO mobile phones, NO fireworks, NO smoke signal!).
  However, messengers may be intercepted (and killed).

Can the two generals agree upon a time for attack such that each of them can be sure about the other general‘s participation?
(The generals and messengers do not lie!)

How to Provide Middleware Services III

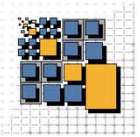


Distinguish between the type of coupling:

- Pure signal: the message does not contain any data;
  computing is done based on the existence of the message
  → *Object.notify()/notifyAll() in Java (Thread synchronization)*
- Structured/Unstructured data: the message contains data
  and the receiver knows how to process it
  → *send some text to a translation service*
- Data and Command: the message contains data and a reference to a processing style
  → *send address data and whether to create/update an entry*
- Data and Processor: the message contains data and executable code for processing it
  → *compute job or downloadable functionality*

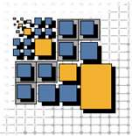
Discuss coupling in terms of interface, platform and application!
Coupling := How many assumptions does the sender make about the receiver and vice versa?

Middleware is Driven by Usage Scenarios



- If you want high performance and dedicated functionality
  - ➔ server/enterprise component technologies
    - Centralized, consistent offering of functionality
    - Failsafe, scalable, 24/7 service provision
- If you want interaction between independent entities
  - ➔ Bus technologies, Peer-to-Peer systems
    - Frameworks and infrastructure for bridging heterogeneity
    - Provision of advanced communication facilities
- If you want something in between
  - ➔ Choose a mix that fits!

Where to Apply Middleware



→Be clear about which LAYER you are talking about!

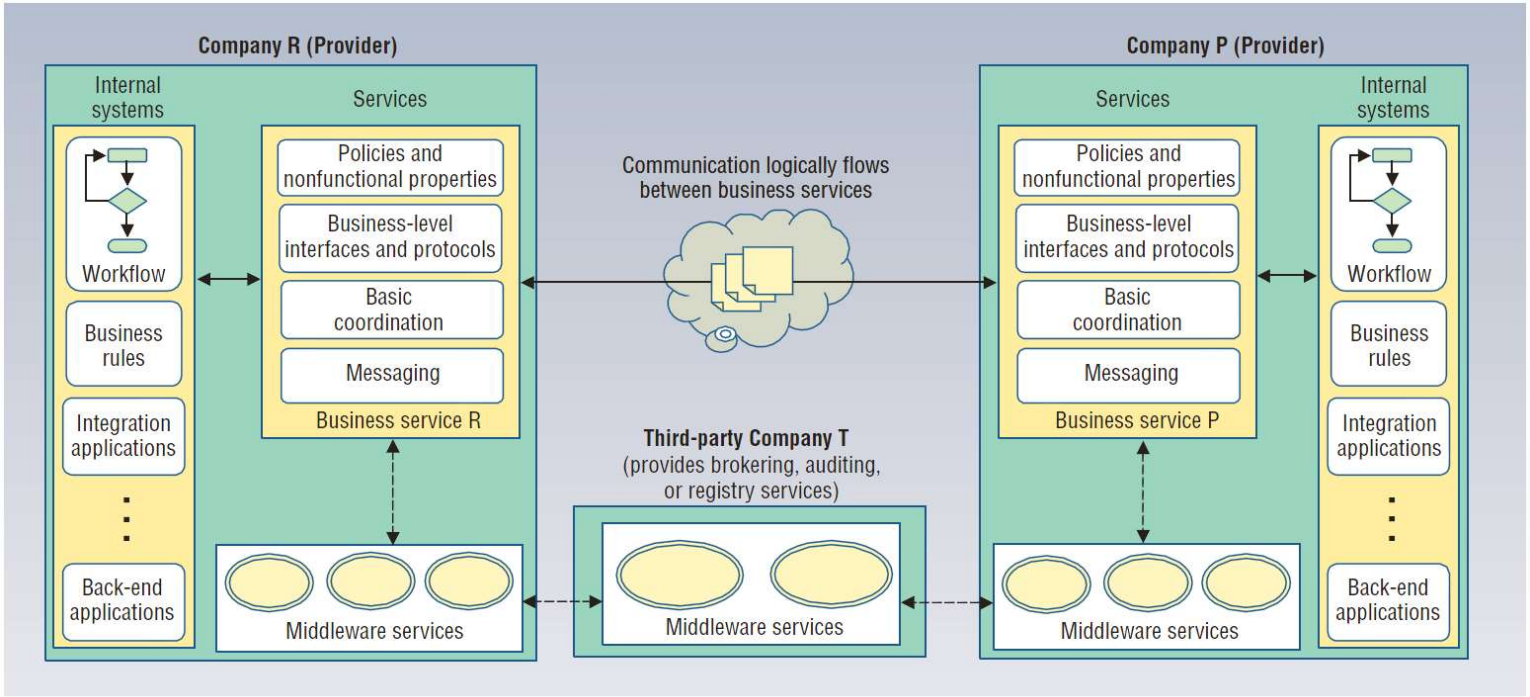
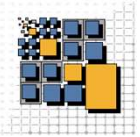


Figure 1. Service integration layers. The lower levels incorporate specifications that most interactions require, while the need for specifications in the higher levels varies depending on the application.

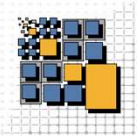
Nezhad et al., *"Web services Interoperability Specifications"*, IEEE Computer, May 2006

Well-Known Types of Middleware



- Message Queueing systems: Websphere MQ, MSMQ,…
  → typical EAI technology for decoupled interaction
- Transaction Processing Systems: IBM CICS, Oracle Tuxedo,…
  → natural evolution of database-centric computing on mainframes
- Bus technologies, most notably CORBA
  → distributed objects on arbitrary platforms/prog. languages
- Domain specific technologies, EDIINT, AS2...
  → support the paradigm of the domain, e.g., business document exchanges
- Web Services, SOA, RESTful services, and Microservices
  → Interface technology for bridging heterogeneity
  (originally for light-weight, stateless interactions)
- Server-Centric systems: EJBs, Servlets, JSF, ASP .NET, …
  → Provision of functionality for different types of clients
- THE CLOUD
  → Solves all problems you will ever have ;-)
- … and others: Grid, P2P, …

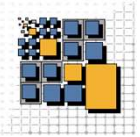
Distributed Systems Programming



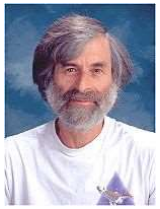
Programming is hard, programming a DS is harder

- You have to consider more than one processing entity and specify the interactions between the entities
- You have to deal with concurrency errors
- You have to respect the features/limitations of your platform
- You have to define a failure model
- You may not be able to test your application in a realistic environment
  - How many users will you have?
  - What will be the peak performance required?
  - What kind of scalability features do you have?

Transparent Middleware Services?



➔ Does the user realize she is using middleware?



http://research.microsoft.com/en-us/um/people/lamport/pubs/distributed-system.txt

Leslie Lamport:

"A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable."

So, does the user have to think about the following features of a particular service?

- Location: Can you use a remote service offline?
- Time: Do you have to be aware of when your job is scheduled?
- Availability: Do you have to care about failover strategies?
- Performance: Do you have to think about where your data is?
- Sharing: Do you have to think about concurrent users?

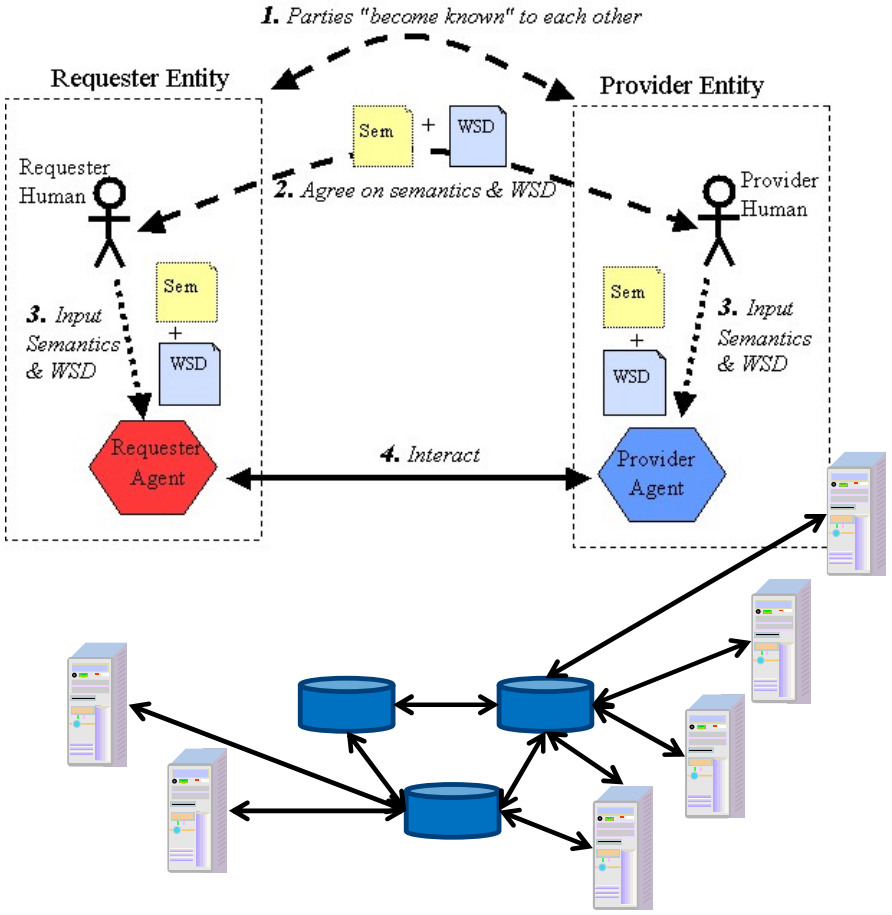
SOC as Distributed Computing Discipline

- SOC = Service Oriented Computing
- The basic service interaction style implies an underlying distributed system.

➔Service interactions are subject to all typical distributed computing problems which are driven by distributed system characteristics.

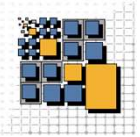
NOTE: SOC != SOA
SOC → Computing based on services
SOA → Architectural paradigm

http://www.w3.org/TR/ws-arch/



See DSG-SOA-M for More Information!

SOC Doubles the Fun!

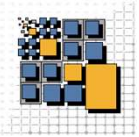


- Building secure/reliable/available… distributed systems is hard!
  → you have to know
  - Business logic
  - Server frameworks
  - Middleware configuration
- SOC is even harder!
  → you have to know
  - Business Logic
  - Server frameworks
  - Middleware configuration
  - Service Wrappers/Exposition of logic as services
  - Service Composition
- And all that in the face of distributed computing tasks
  - Data conversion
  - Distributed commit
  - Distributed transactions
  - …

Create applications and provide middleware services using a dedicated middleware technology!

Reuse applications or build new applications from existing applications no matter what the underlying technology is!

Next Topics for this Course



- (Synchronization)
- Software Architecture
- Server Centric Applications
  - Spring Boot
  - JPA
- Cloud Computing, in particular platform as a service
  - Characteristics of Cloud Computing
  - Google App Engine
  - Serverless / FaaS
  - Similarities with component technologies
- Classical Middleware