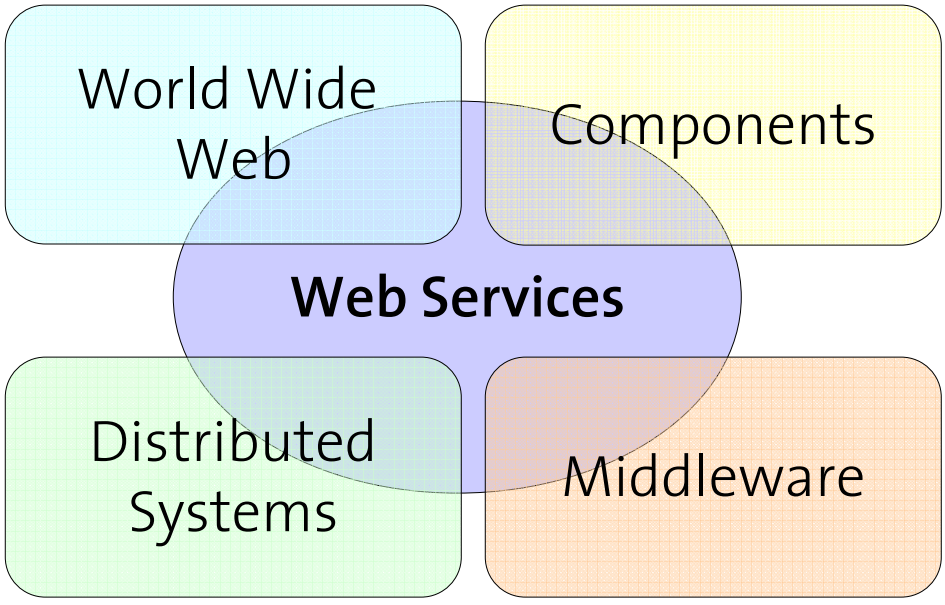
IKS Information and Communication Systems Research Group

ETH Eidgenössische Technische Hochschule Zürich Swiss Federal Institute of Technology Zurich

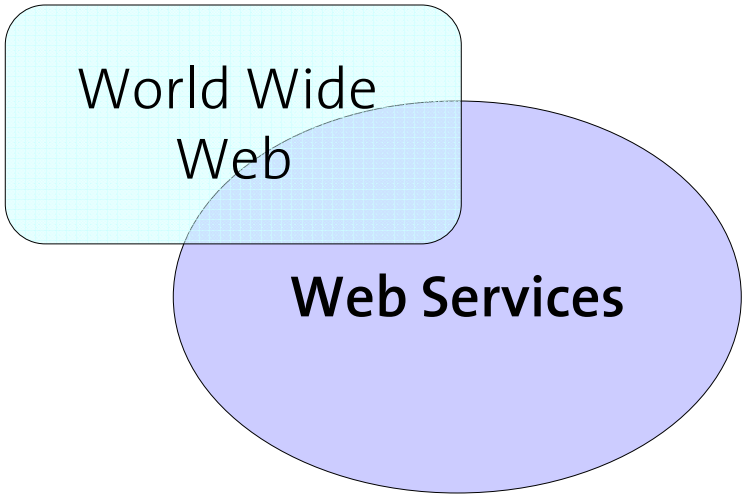
Distributed Systems
Web Services

Dr. Cesare Pautasso
Computer Science Department
Swiss Federal Institute of Technology (ETHZ)
pautasso@inf.ethz.ch
http://www.inf.ethz.ch/~pautasso

Web Services in Context



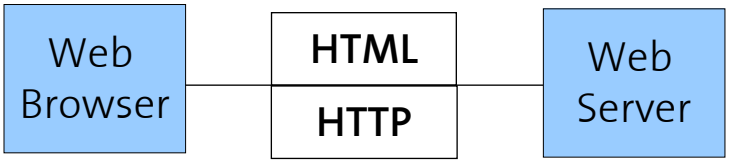
Web Services in Context



Is this a Web Service?

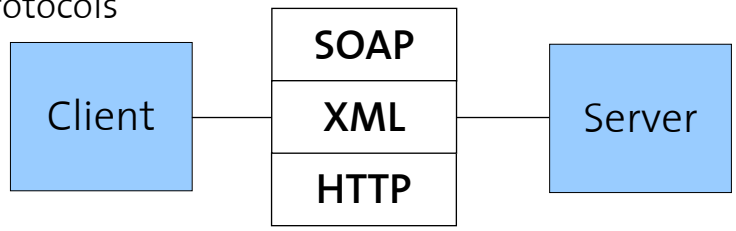
Web-based Services

- Services offered through a Web site

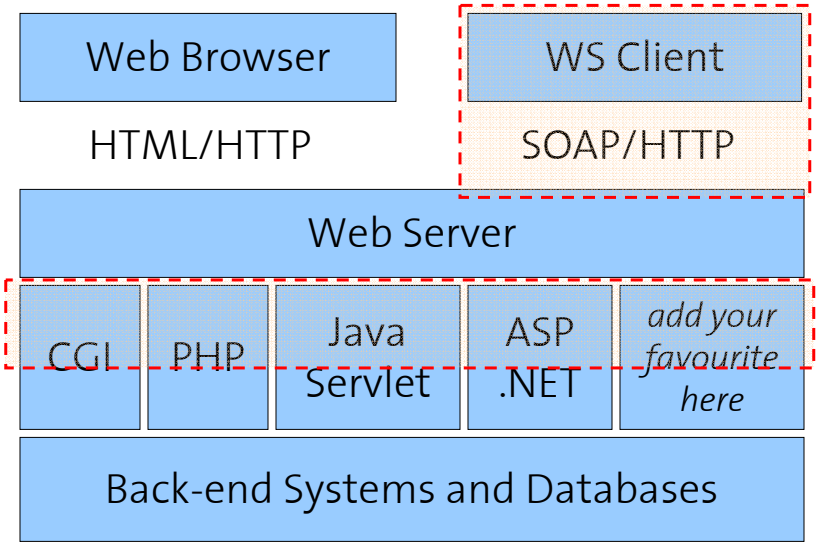


Web Services

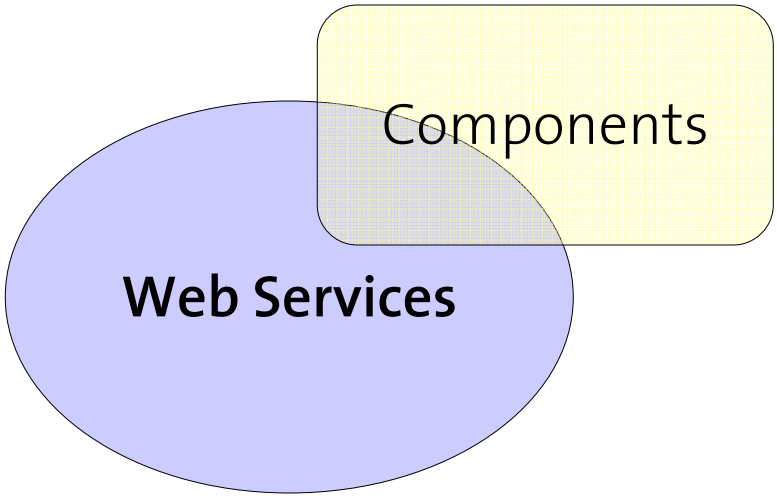
- Services offered through Web-wide standardized protocols



Extending the Web with Services



Web Services in Context



Services and Components

- What is a component?
- What is a service?

- **Component Based Software Engineering** (CBSE). Define system architectures in terms of the dependencies connecting a set of reusable components (Spatial dimension)
- **Service Oriented Architectures** (SOA). The architecture of a distributed system is defined in terms of the interactions among its component services (Temporal dimension)

Components

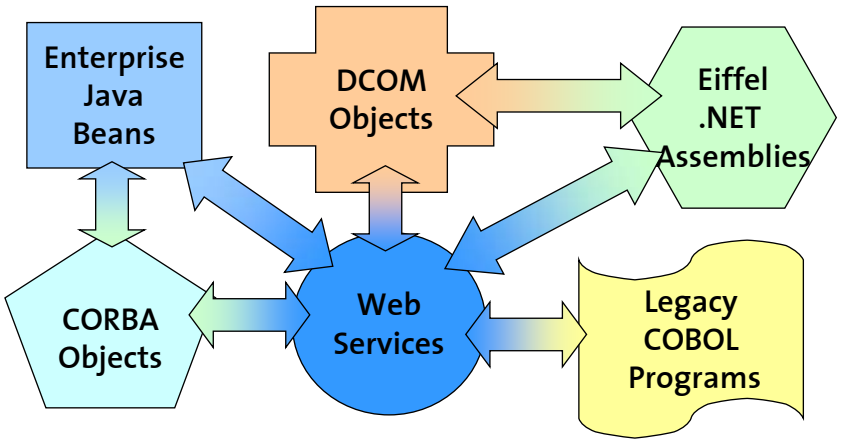
- Software components are reusable
- To be used a component must:
  - be packaged to be deployed as part of some larger application system
  - fit with the existing framework used to develop the system (as an exercise try to use a .NET assembly to make an Eclipse plug-in and see what happens)
- Components can be sold.
  - Component developers charge on a per-deployment basis: whenever a new client downloads the component.
- There are many component frameworks available for building distributed systems (e.g., J2EE, DCOM, .NET, CORBA).
- The problem is: they are not compatible.

Web Services are Components

- Web services are reusable too.
- To be used a service must:
  - be published on the Web (once)
  - advertise its description and location to potential clients across the Web so that they can access it using standard protocols
- Web Services can be sold too.
  - Service providers can charge on a per-call basis: each time an existing client interacts with a service by exchanging a new message.
- Like components, Web services can be reused, composed into larger systems and (of course) they can be found on the Web.
- Unlike components, Web services do not have to be downloaded and deployed in order to be used by clients. Instead, a client may discover and access their functionality by using standard protocols (WSDL, SOAP, UDDI) based on XML.

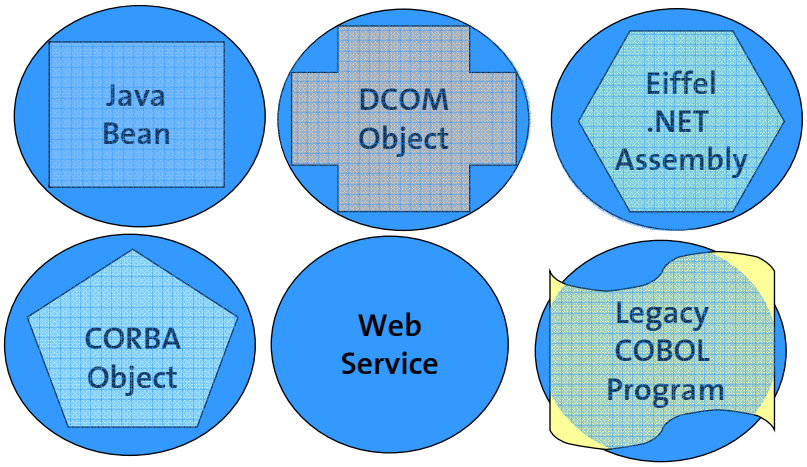
Component Interoperability

- Due to lack of interoperability, it is not always possible to build a distributed system using heterogeneous components

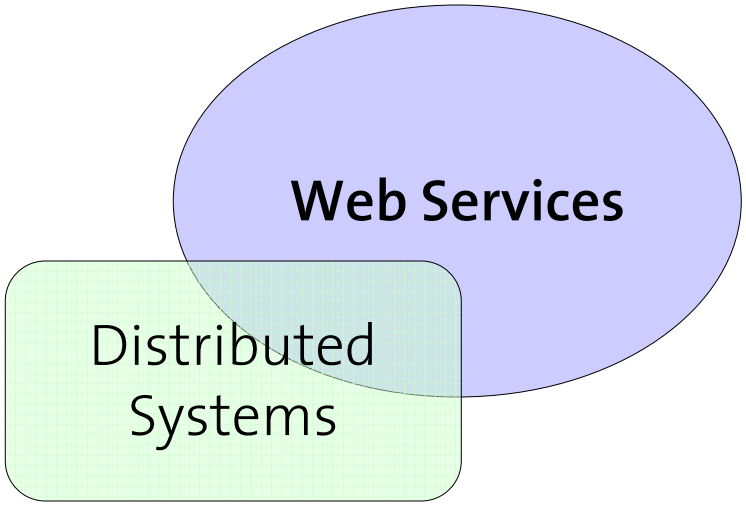


Web Services for Interoperability

- If the components are published as Web services, they can interoperate across different component frameworks. (Interoperability through Wrapping)



Web Services in Context



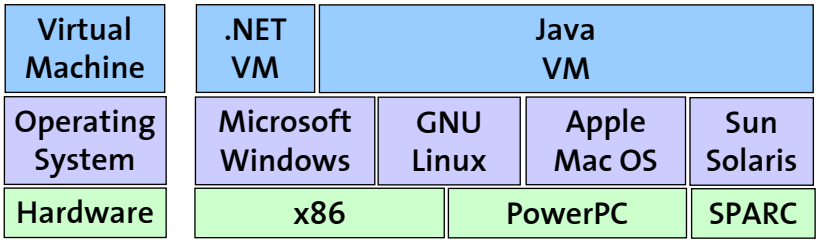
Distributed Systems & Web Services

- Web services provide standards for developing large scale distributed systems
- One example: “the Grid” is adopting Web services as standard protocols to build a distributed infrastructure for utility based computing
- Web services on the path of success while CORBA distributed objects failed (This is nothing technical, only a matter of widespread industry acceptance)

The Internet	The WWW	Web Services	Semantic Web
1973	1992	2000	?
Standard Network	*Standard User Interface*	*Standard API*	*Standard API Metadata*

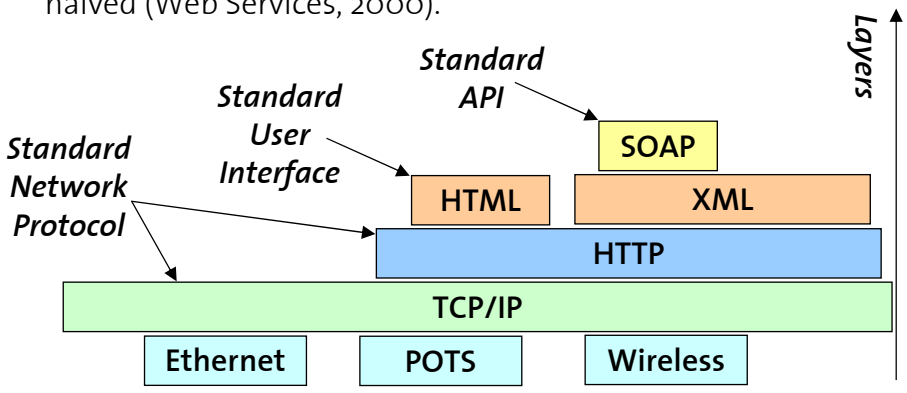
Standards, Platforms and Layers

- A layer with a standard interface becomes a stable platform on which to build the higher layers
- The purpose of a platform is also to hide the complexity of the lower layers
- The OS/VM platform example shows that controlling the standard can bring a great competitive advantage in the marketplace

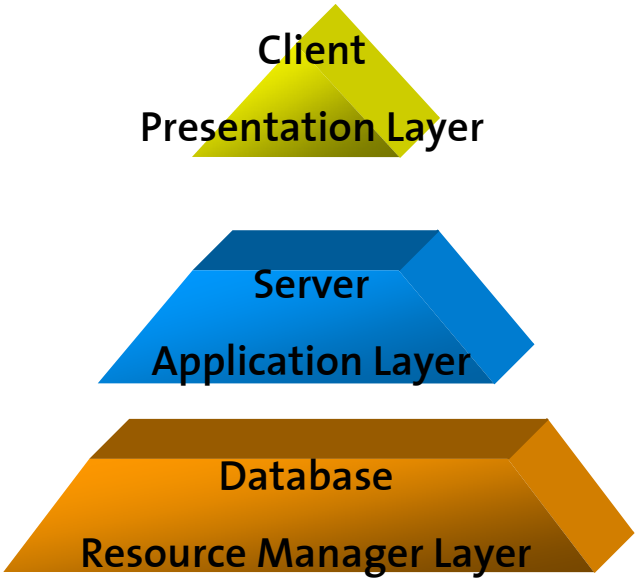


Standards for Distributed Systems

- Distributed systems are built using standardized layers of increasingly higher abstraction levels.
- It took 20 years to go from the TCP/IP (Internet, 1973) standard to the HTTP/HTML (World Wide Web, 1992) standards.
- By reusing HTTP, the time to standardize SOAP/XML was halved (Web Services, 2000).



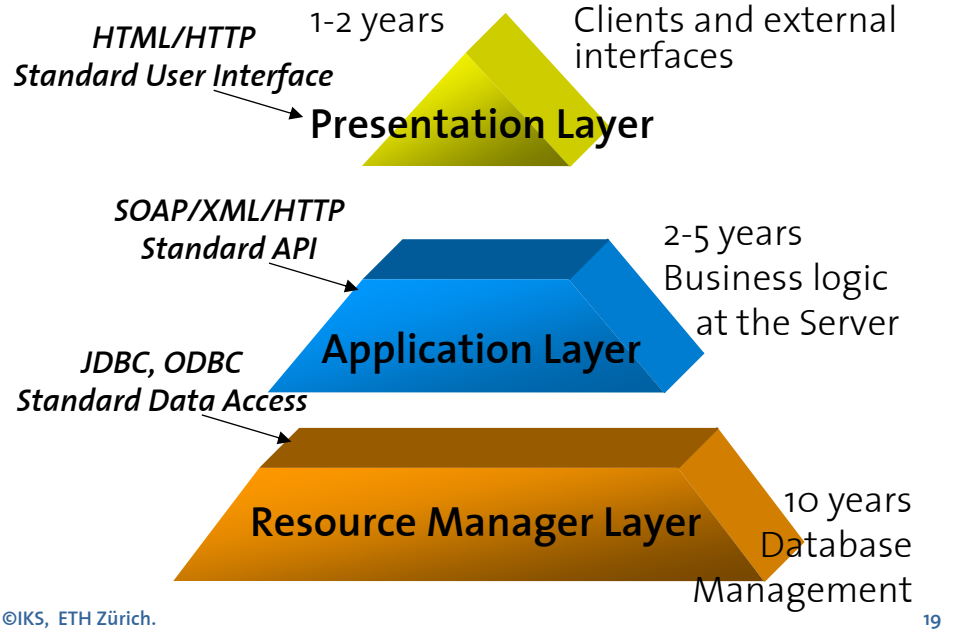
Layers in Distributed Systems

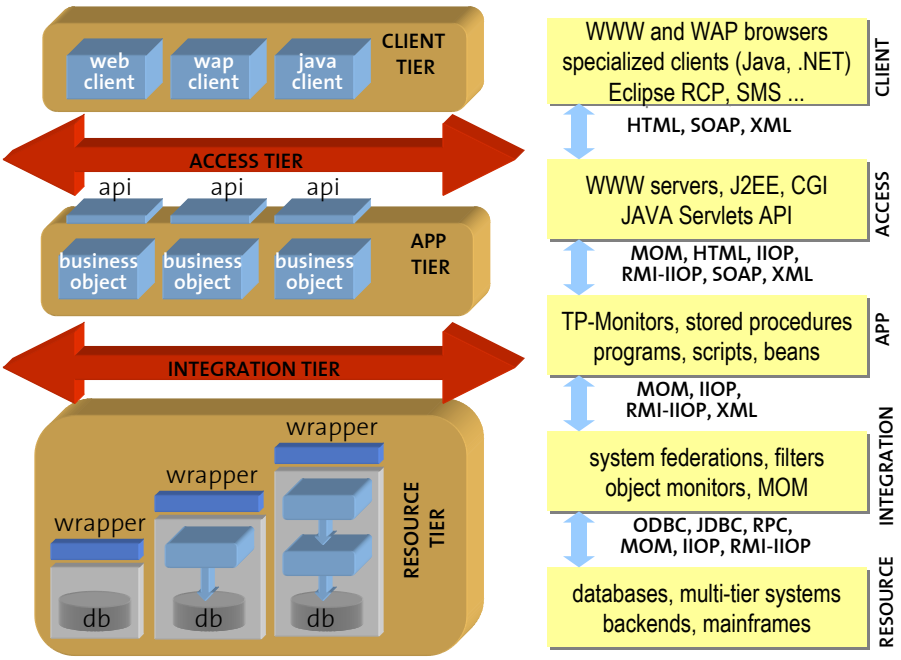


Layers in Distributed Systems

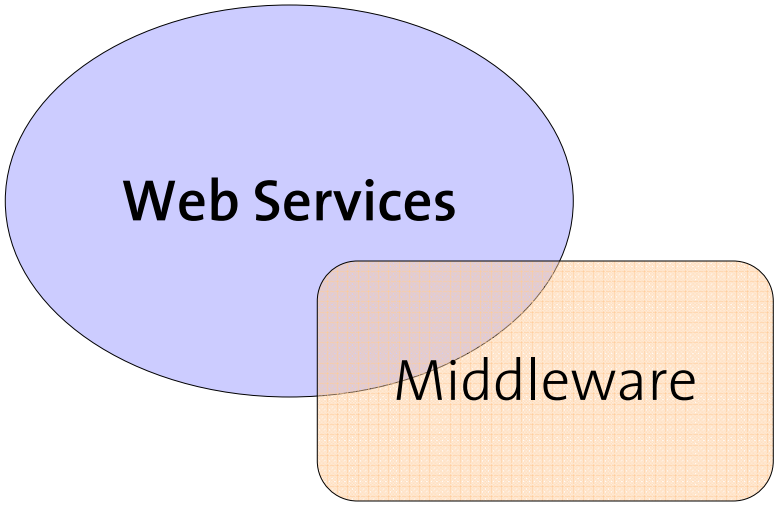
- **Client** is any user or program that wants to perform an operation over the system. To support a client, the system needs to have a **presentation layer** through which the user can submit operations and obtain a result.
- The **application logic** establishes what operations can be performed over the system and how they take place. It takes care of enforcing the business rules and establish the business processes. The application logic can be expressed and implemented in many different ways: constraints, business processes, server with encoded logic ...
- The **resource manager** deals with the organization (storage, indexing, and retrieval) of the data necessary to support the application logic. This is typically a **database** but it can also be a text retrieval system or any other data management system providing querying capabilities and persistence.

Layers in Distributed Systems

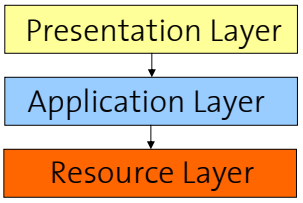


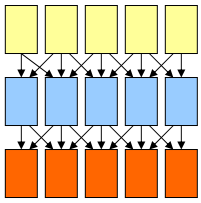


Web Services in Context



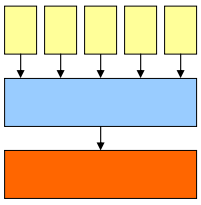
Distributing the Layers



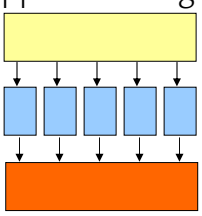


In general, all layers can be distributed

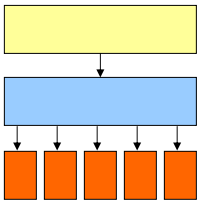
Support for multiple clients



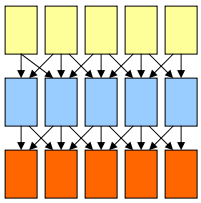
Modular Application Logic



Data Partitioning or Replication



A game of boxes and arrows



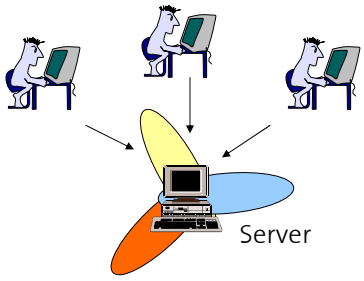
There is no problem in system **design** that cannot be solved by adding a level of indirection.

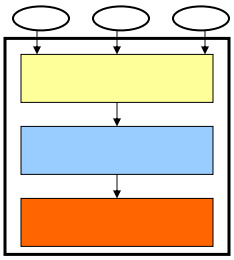
There is no **performance** problem that cannot be solved by removing a level of indirection.

- Each box represents a part of the system.
- Each arrow represents a connection between two parts of the system.
- The more boxes, the more modular the system: more opportunities for distribution and parallelism.
  This allows encapsulation, component based design, reuse.
- The more boxes, the more arrows: more sessions (connections) need to be maintained, more coordination is necessary. The system becomes more complex to monitor and manage.
- The more boxes, the greater the number of context switches and intermediate steps to go through before one gets to the data. Performance suffers considerably.
- System designers try to balance the flexibility of modular design with the performance demands of real applications. Once a layer is established, it tends to migrate down and merge with lower layers.

One tier: fully centralized

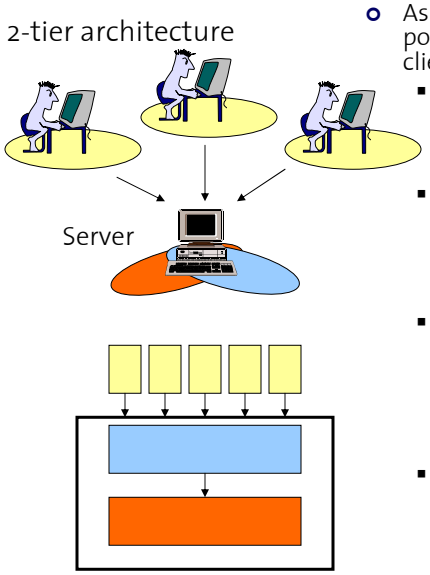
1-tier architecture





- The presentation layer, application logic and resource manager are built as a monolithic entity.
- Users/programs access the system through display terminals but what is displayed and how it appears is controlled by the server.
  (These are “dumb” terminals).
- This was the typical architecture of mainframes, offering several advantages:
  - no forced context switches in the control flow (everything happens within the system),
  - all is centralized, managing and controlling resources is easier,
  - the design can be highly optimized by blurring the separation between layers.

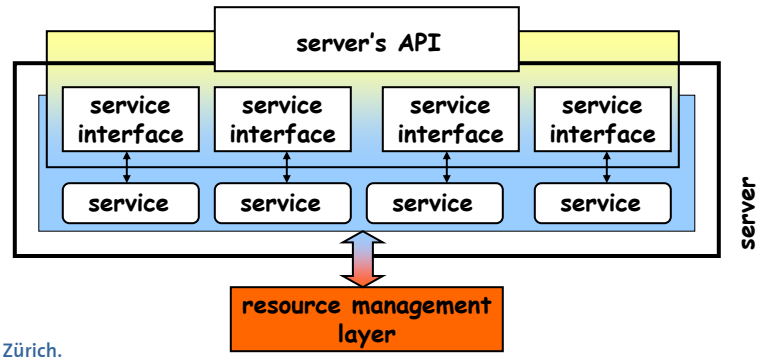
Two tier: Client/Server



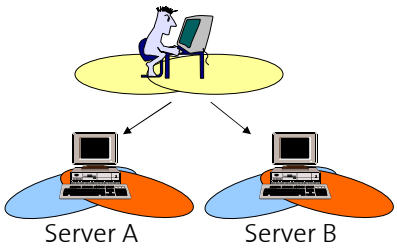
- As computers became more powerful, it was possible to move the presentation layer to the client. This has several advantages:
  - Clients are independent of each other: one could have several presentation layers depending on what each client wants to do.
  - One can take advantage of the computing power at the client machine to have more sophisticated presentation layers. This also saves computer resources at the server machine.
  - It introduces the concept of API (Application Program Interface). An interface to invoke the system from the outside. It also allows designers to think about federating the systems into a single system.
  - The resource manager only sees one client: the application logic. This greatly helps with performance since there are no client connections/sessions to maintain.

Standard Client/Server APIs

- Client/server systems introduced the notion of service (the client invokes a service implemented by the server)
- Together with the notion of service, client/server introduced the notion of service interface (how the client can invoke a given service)
- Taken all together, the interfaces to all the services provided by a server define the server's Application Program Interface (API) that describes how to interact with the server from the outside
- Web Services standardize the mechanisms used to describe, discover and access the API offered by a server

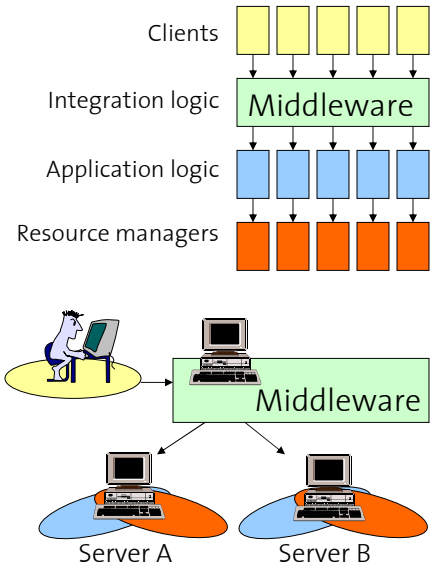


The problem of Client/Server



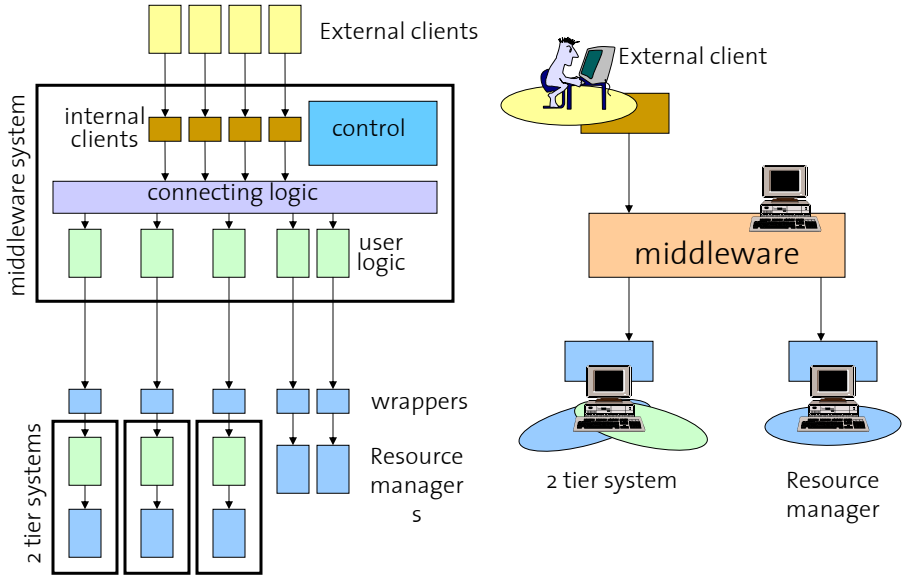
- If clients want to access two or more servers, a 2-tier architecture causes several problems:
  - the underlying systems don't know about each other
  - there is no common business logic
  - the client is the point of integration (increasingly fat clients)
  - The responsibility of dealing with heterogeneous systems is shifted to the client.
- This is tremendously inefficient from all points of view (software design, portability, code reuse, performance since the client capacity is limited, etc.).
- There is very little that can be done to solve this problems if staying within the 2 tier model.

Middleware

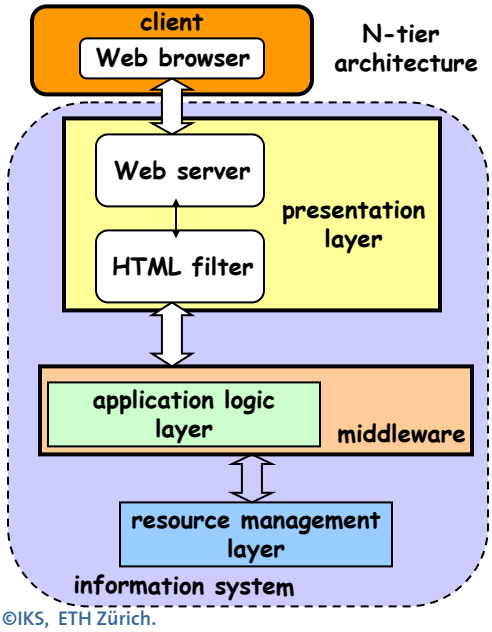


- Middleware is just a level of indirection between clients and other layers of the system.
- It introduces an additional layer of business logic encompassing all underlying systems.
- By doing this, a middleware system:
  - simplifies the design of the clients by reducing the number of interfaces,
  - provides transparent access to the underlying systems,
  - acts as the platform for inter-system functionality and high level application logic, and
  - takes care of locating resources, accessing them, and gathering results.
- But a middleware system is just a system like any other! It can also be 1 tier, 2 tier, 3 tier ...

Middleware 3-tier Architectures



N-tier: connecting to the Web

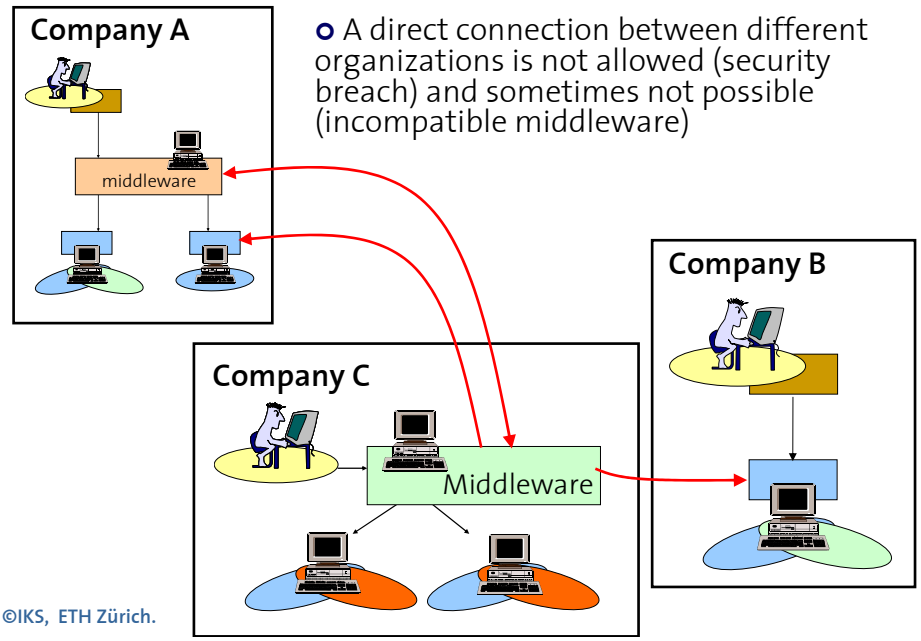


- N-tier architectures result from connecting several three tier systems to each other and/or by adding an additional layer to allow clients to access the system through a Web server
- The Web layer was initially external to the system (a true additional layer); today, it is slowly being incorporated into a presentation layer that resides on the server side (part of the middleware infrastructure in a three tier system, or part of the server directly in a two tier system)
- The addition of the Web layer led to the notion of "application servers", which was used to refer to middleware platforms supporting access through the Web

Limitations of Middleware

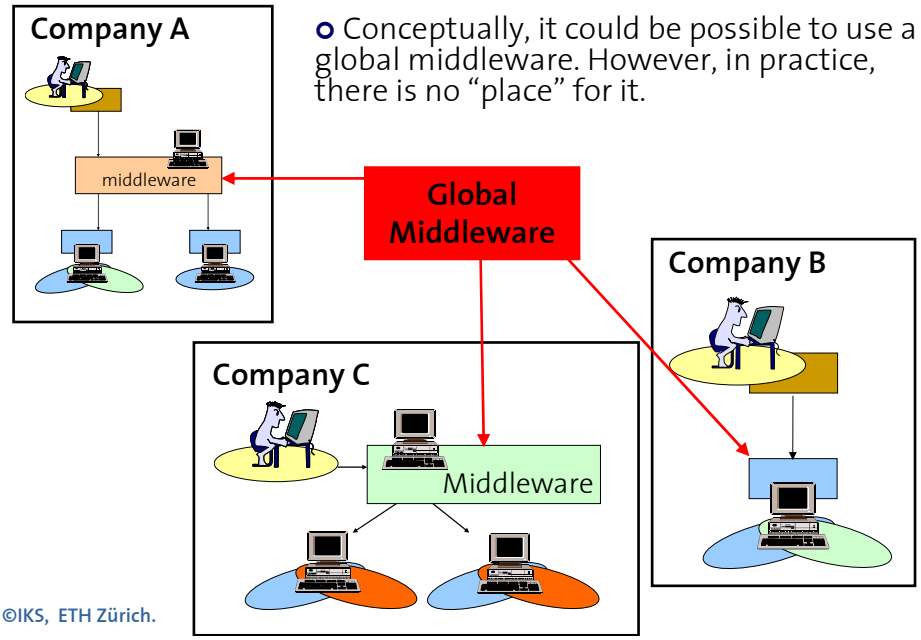
- Non-standard interfaces. Traditional middleware systems and tools suffer from lack of standardization: they are not compatible. Thus, it is very expensive to build integrated distributed systems across different middleware platforms.
- Lack of trust. With Web Services the internal "API" of a company is exposed to the Internet. How to trust the clients? Building integrated systems spawning across different trust domains can be difficult.
- Middleware systems are (logically) centralized. Thus, there is no place for them in B2B Integration scenarios as they should be distributed across all partners. Point to Point integration does not scale.
- Interactions across organizational boundaries may be slow and should be handled asynchronously.

Limitations of Middleware



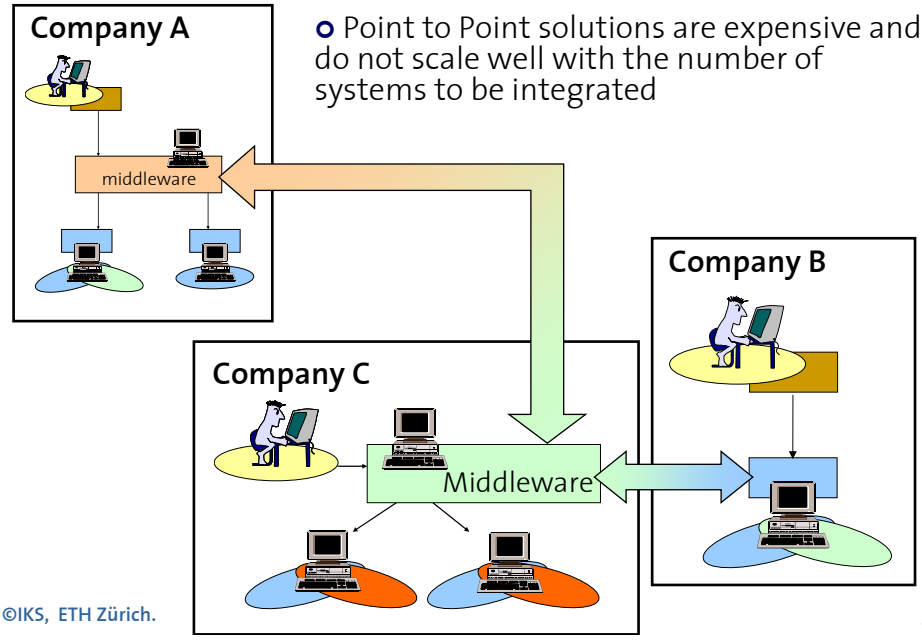
- A direct connection between different organizations is not allowed (security breach) and sometimes not possible (incompatible middleware)

Limitations of Middleware



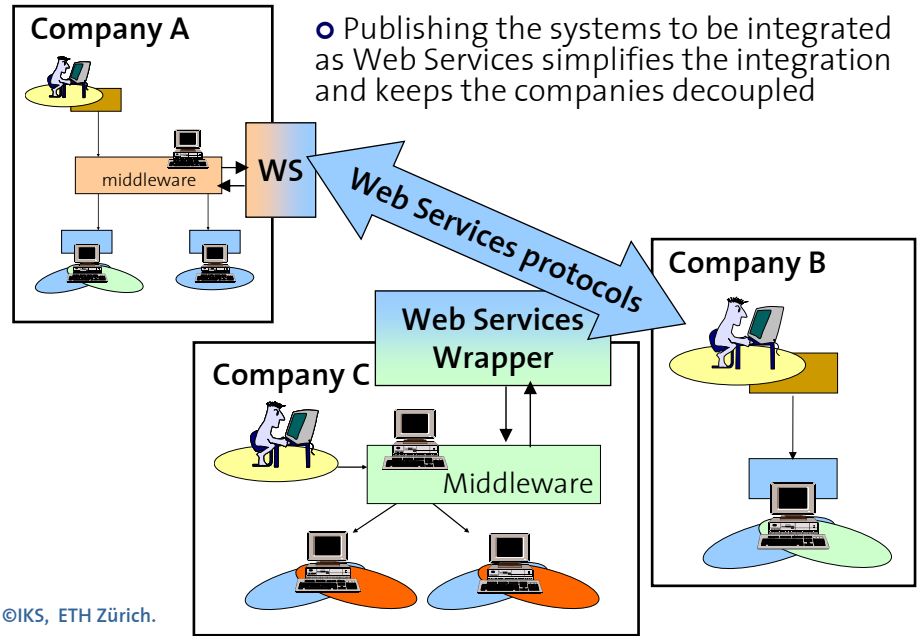
- Conceptually, it could be possible to use a global middleware. However, in practice, there is no "place" for it.

Limitations of Middleware



- Point to Point solutions are expensive and do not scale well with the number of systems to be integrated

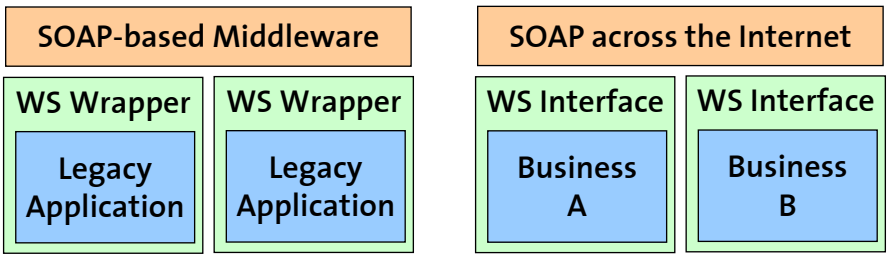
Web Services for integration



- Publishing the systems to be integrated as Web Services simplifies the integration and keeps the companies decoupled

Web Services and Middleware

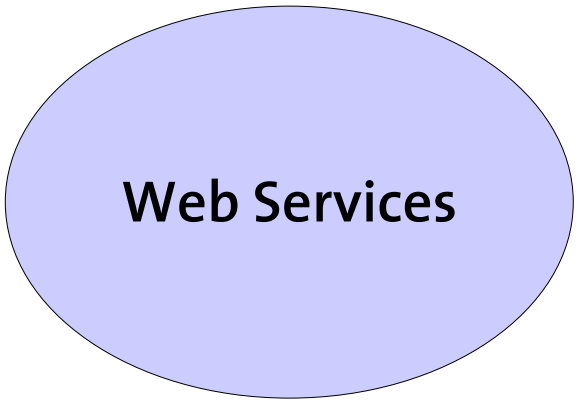
- Web Services can be seen as the natural evolution of existing Middleware systems:
  - Web Services standards enable the interoperability of existing Middleware platforms and tools
  - Enterprise Application Integration made easier by using Web Services
  - Business to Business integration enabled by common interface standards
    ("Unlike CORBA protocols, SOAP goes through firewalls")



Web Services and Middleware

- The Web services architecture represented by SOAP, UDDI, and WSDL is a direct descendant of conventional middleware platforms. They can be seen as the most basic extensions that are necessary to allow conventional synchronous (RPC based) middleware to achieve interoperability.
- The model and even the notation followed in this architecture mimics to a very large extent what has already been done in RPC, RMI, CORBA, etc.
- This dependency gives a very good hint of what can be done with these technologies today and what is missing to obtain a complete distributed systems platform
- First implementations are just extensions of existing platforms to accept invocations through a Web service interface (e.g., database stored procedure published as Web services)

Defining Web Services



Web services are not...

- ...the latest revolutionary technology which will enable seamless interoperability and solve all integration problems across the entire World Wide Web because all major software vendors are going to support Web services related standards with a new wave of powerful automatic tools

[Hint: Try to recognize **hype** when you see it]

What are Web services?

- The term Web services has become nowadays very popular and it is not always used with the same meaning.
- At one extreme, a Web service is any application program which is accessible through the World Wide Web
- More precisely, W3C defines Web services as: a *software application identified by a URI, whose interfaces and bindings are capable of being defined, described and discovered as XML artifacts.*
  A Web service supports direct interactions with other software agents using XML-based messages exchanged via the Internet
- The long-term goal is *just-in-time* integration of applications by discovering and orchestrating Web services available on the network

Properties of Web Services

- The W3C definition emphasizes different aspects:
  - In order to be accessed by clients, a Web Service should be defined, **described and discovered**.
  - **XML** is the foundation for all **standards** that are going to be used (SOAP, WSDL, UDDI) to do so.
  - Web services are intended to be used as **components** that can be readily integrated into more complex distributed applications.
  - Web services are meant for software based consumption (clients are programs)
  - Web-based applications are meant to be used by humans equipped with a WWW browser (clients are users)

Benefits of Web services

- One important difference with conventional middleware is related to the standardization efforts at the W3C that should guarantee:
  - Platform independence (Hardware, Operating System)
  - Reuse of existing networking infrastructure (HTTP has become ubiquitous)
  - Programming language neutrality (.NET talks with Java, and vice versa)
  - Portability across Middleware tools of different Vendors
  - Web services are loosely coupled, reusable and can be adopted incrementally

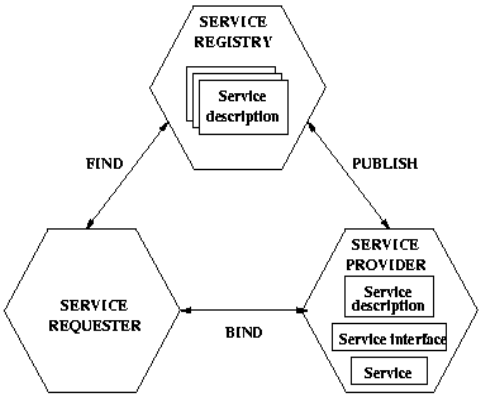
Problems of Web services

- What is the price to pay for interoperability?
- Currently Web services standards are still rapidly evolving
- A good thing about standards is that there are so many to choose from: Many WS-* standards are competing and overlapping.
- Not all standards are supported by tools. Tools must play catch-up with new standard versions.
- The performance of some of the available tools and protocols is quite poor.

Web Services Architecture

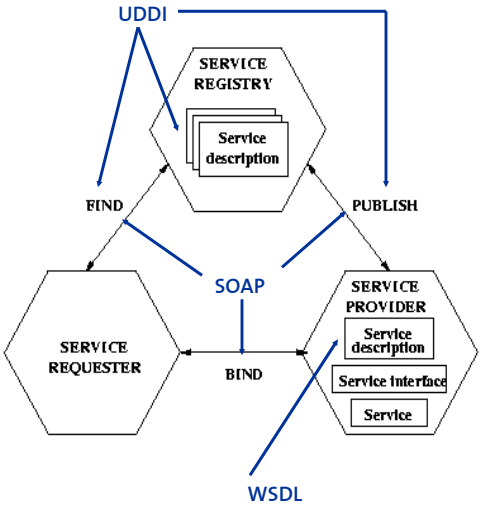
- A popular interpretation of Web services is based on IBM's *Web service architecture* based on three elements:

1. Service **requester**: The potential user of a service (the client)
2. Service **provider**: The entity that implements the service and offers to carry it out on behalf of the requester (the server)
3. Service **registry**: A place where available services are listed and that allows providers to advertise their services and requesters to lookup and query for services



Main Web Services Standards

- The Web service architecture proposed by IBM is based on two key concepts:
  - architecture of existing synchronous middleware platforms
  - current specifications of SOAP, UDDI and WSDL
- The architecture has a remarkable client/server flavor
- It reflects only what can be done with
  - SOAP (Simple Object Access Protocol)
  - UDDI (Universal Description and Discovery Protocol)
  - WSDL (Web Services Description Language)



Plan

- Context: What is the problem? Interoperability in distributed systems
- Solution: Standardization
  - Web Service Invocation: SOAP
  - Web Service Description: WSDL
  - Web Service Discovery: UDDI
- Advanced Topics
  - Web Service Coordination
  - Web Service Composition

The Web services stack

	WSDL-based		Semantic Web	ebXML
Messaging	**SOAP**			ebXML MSS
Description	**WSDL**		RDF	ebXML CPP
Nonfunctional description	WSEL		DAML-S	
Conversations	WSCL	WSCI		ebXML BPSS
Choreography	WS-Coordination			
Business processes	BPEL4WS WSFL/XLANG	BPML		BPML
Contracts				ebXML CPA
Discovery	**UDDI**			ebXML registries
Transactions	WS-Transactions	BTP		BTP
Security	WS-Security			SAML S/MIME

More information

- Take the EAI lecture, if you are interested in doing a big project using Web services
- Read the book:
  - G. Alonso et al., **Web Services. Concepts, Architectures and Applications**, Springer, 2004
  - ISBN 3-540-44008-9
- ETH-BIB 783322
- ETH-INFK IK.04.1