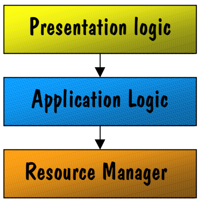
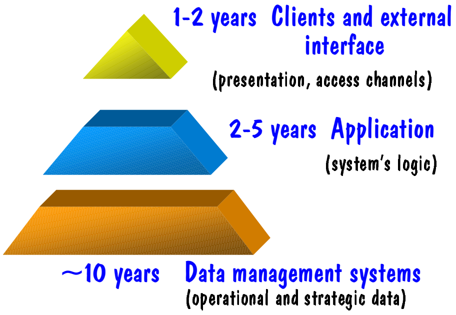
Distributed Information Systems
From Middleware to Web services (Part 1)
VS - WS 2002/2003

Prof. Dr. Gustavo Alonso
Computer Science Department
ETH Zürich
alonso@inf.ethz.ch

Understanding the layers

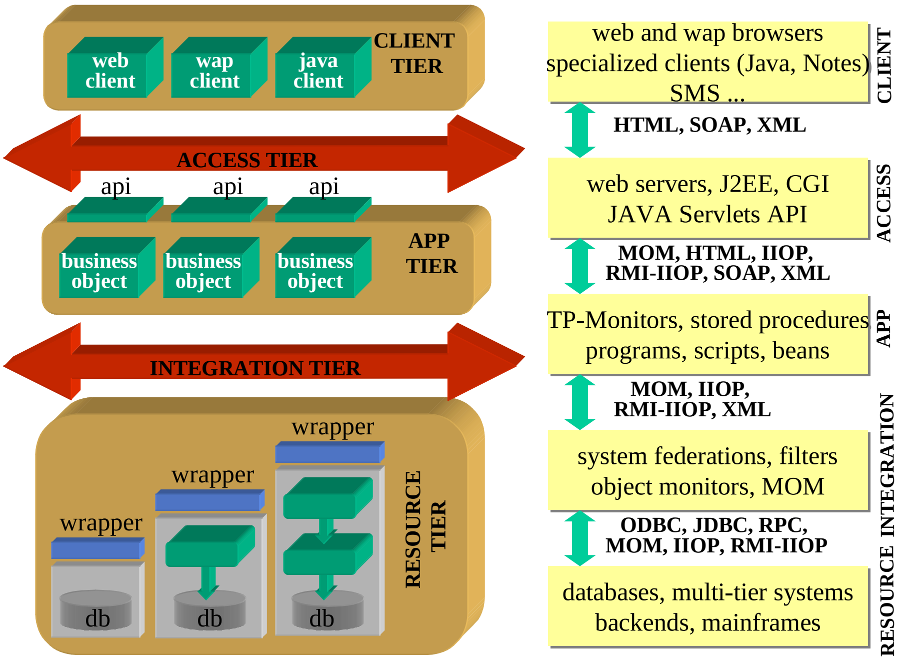

Presentation logic
Application Logic
Resource Manager


1-2 years Clients and external interface
(presentation, access channels)

2-5 years Application
(system's logic)

~10 years Data management systems
(operational and strategic data)

- ❑ 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.



CLIENT

web and wap browsers
specialized clients (Java, Notes)
SMS ...

HTML, SOAP, XML

ACCESS

web servers, J2EE, CGI
JAVA Servlets API

MOM, HTML, IIOP,
RMI-IIOP, SOAP, XML

APP

TP-Monitors, stored procedures
programs, scripts, beans

MOM, IIOP,
RMI-IIOP, XML

INTEGRATION

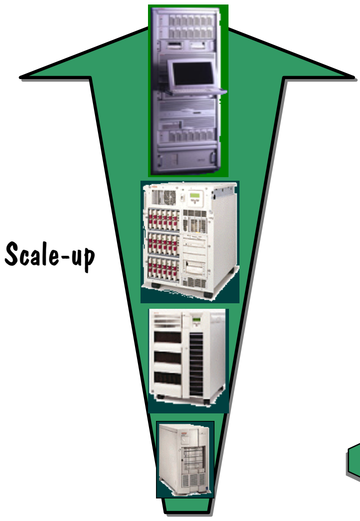
system federations, filters
object monitors, MOM

ODBC, JDBC, RPC,
MOM, IIOP, RMI-IIOP

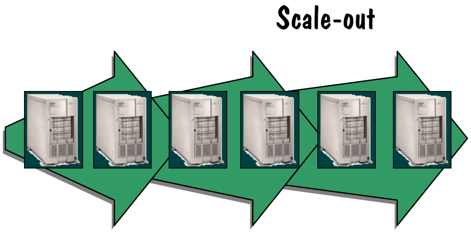
RESOURCE

databases, multi-tier systems
backends, mainframes

Understanding the context

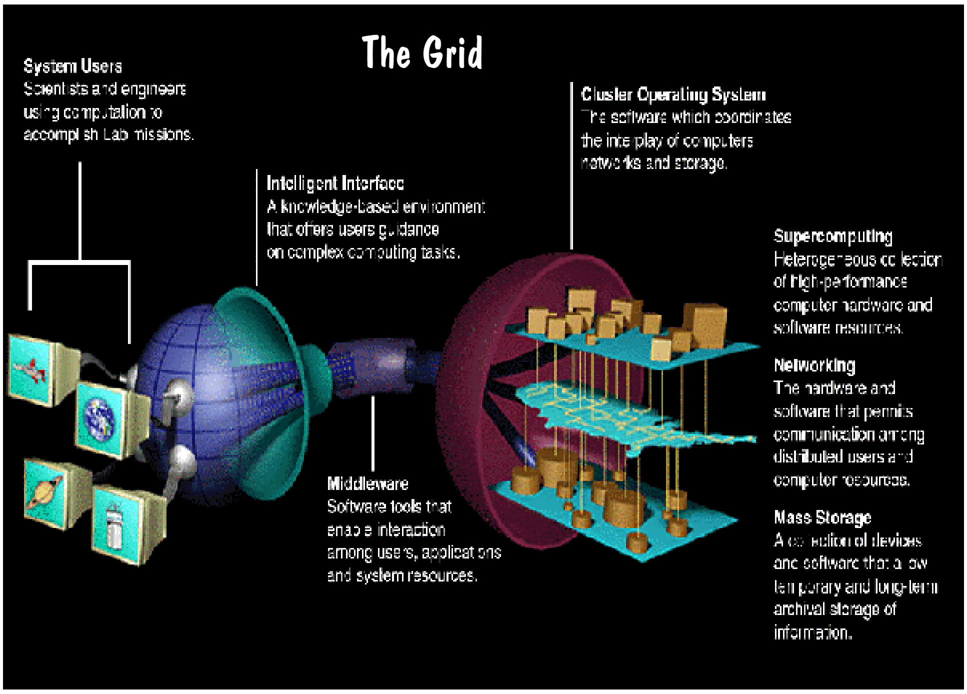


- Scale up is based on using a bigger computer as the load increases. This requires to use parallel computers (SMP) with more and more processors.
- Scale out is based on using more computers as the load increases instead of using a bigger computer.
- Both are usually combined! Scale out can be applied at any level of the scale up.



Diagrams courtesy of Jim Gray, Microsoft

The Grid



System Users
Scientists and engineers using computation to accomplish Lab missions.

Intelligent Interface
A knowledge-based environment that offers users guidance on complex computing tasks.

Cluster Operating System
The software which coordinates the interplay of computers networks and storage.

Supercomputing
Heterogeneous collection of high-performance computer hardware and software resources.

Networking
The hardware and software that permits communication among distributed users and computer resources.

Middleware
Software tools that enable interaction among users, applications and system resources.

Mass Storage
A collection of devices and software that allow temporary and long-term archival storage of information.

Understanding the applications

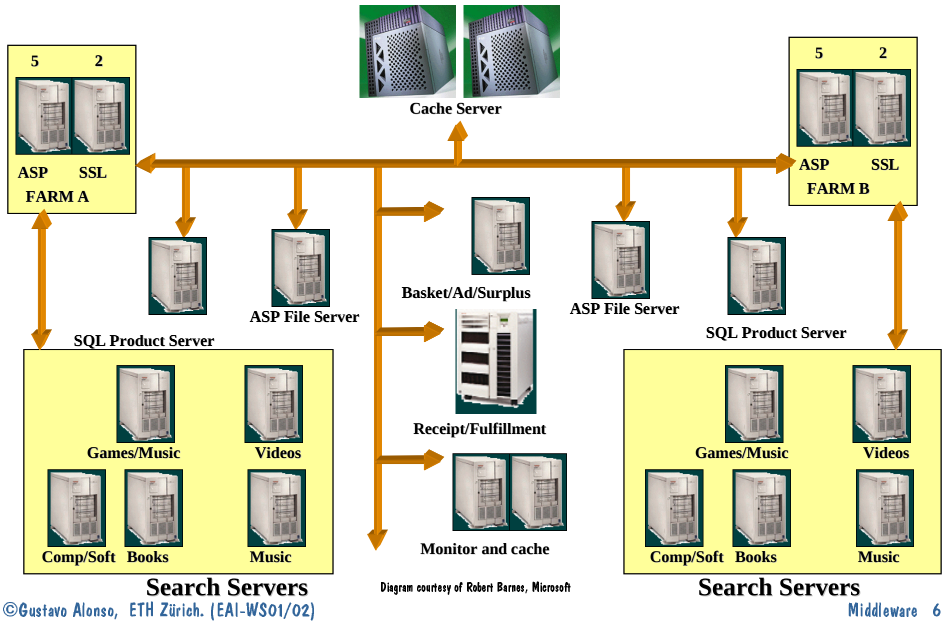
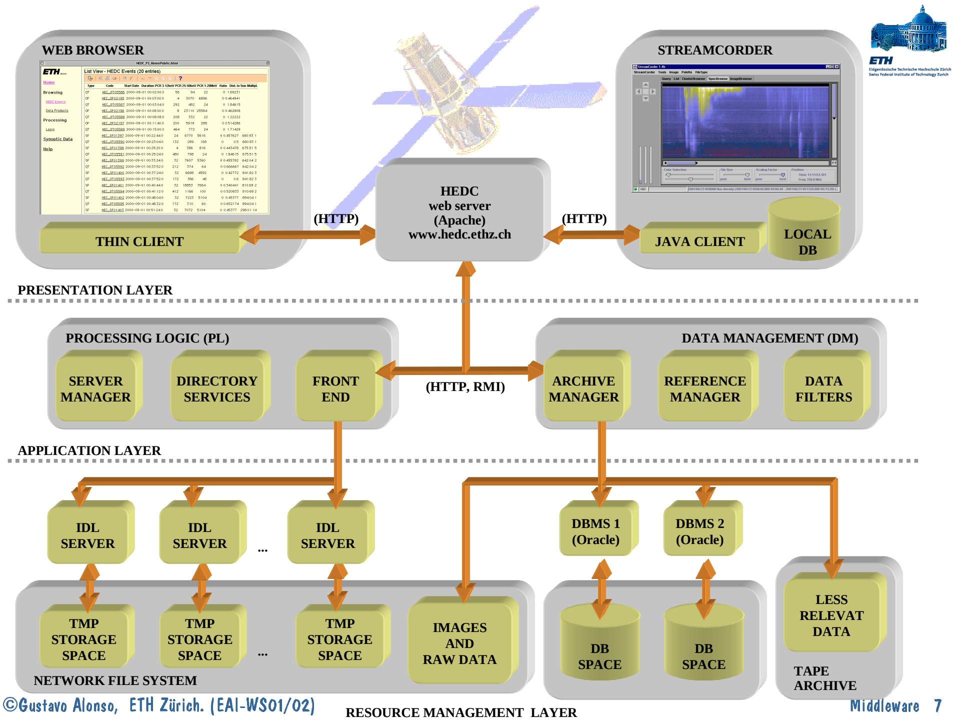
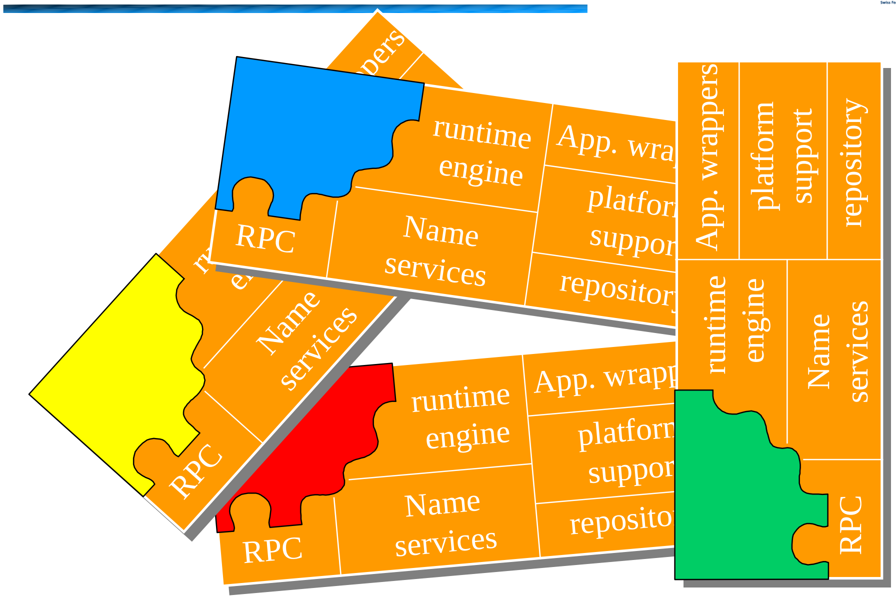


Diagram courtesy of Robert Barnes, Microsoft

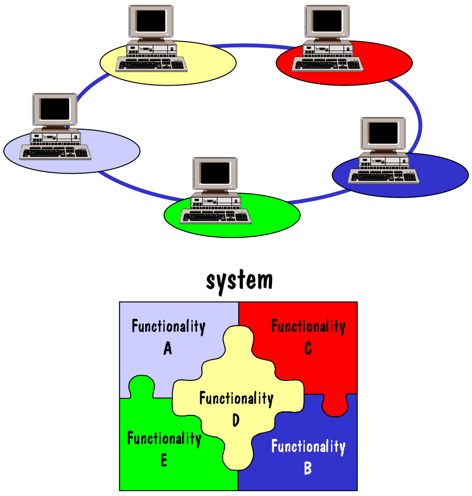


Understanding products



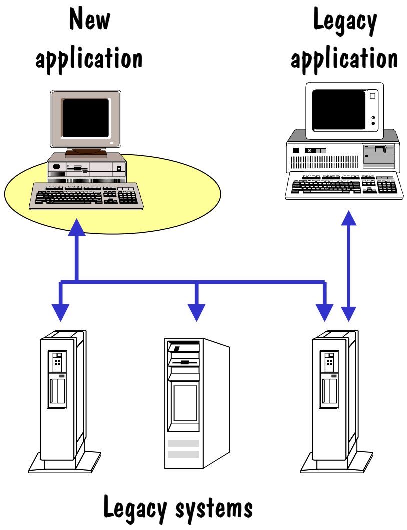
The evolving nature of the architecture of distributed information systems

Distributed applications (top down design)



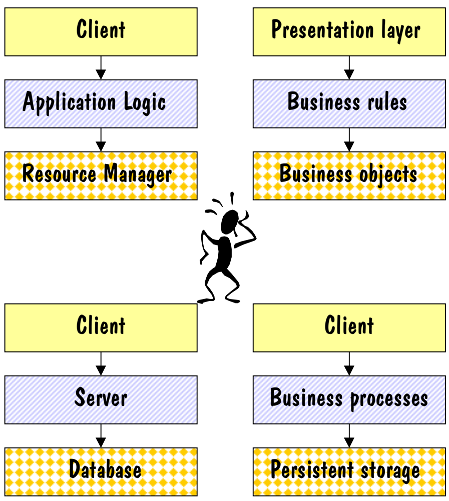
- ❑ The functionality of a system is divided among several distributed nodes which work exclusively for the system.
- ❑ Each node cannot act as a separate component, its functionality depends on the functionality implemented at other nodes.
- ❑ Nodes are typically homogeneous and the system is designed to be distributed from the beginning.
- ❑ This is the architecture of many applications (for instance databases), however, they do not allow to incorporate other applications, legacy systems, and are difficult to extend.

Distributed applications (bottom up design)



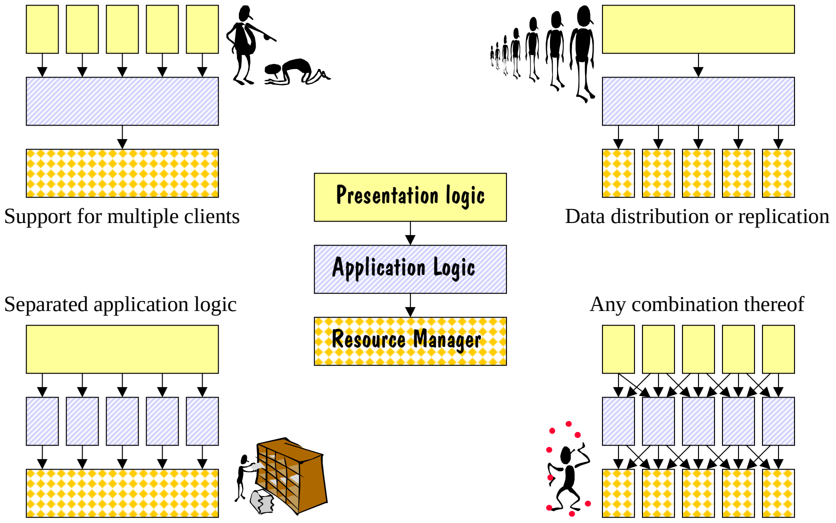
- ❑ The basic components already exist, stand alone systems which need to be linked in order to provide better functionality.
- ❑ The components do not necessarily cease to work as stand alone components. Often old applications continue running at the same time as new applications.
- ❑ This approach has a wide application because the underlying systems already exist and cannot be easily replaced.
- ❑ Much of the work and products in this area are related to middleware, the intermediate layer used to provide a common interface, bridge heterogeneity, and cope with distribution.

Basic concepts and notation



- ❑ 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.

Distribution at the different layers



Support for multiple clients

Presentation logic

Application Logic

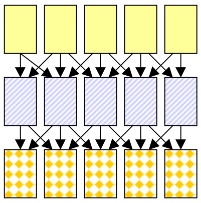
Resource Manager

Data distribution or replication

Separated application logic

Any combination thereof

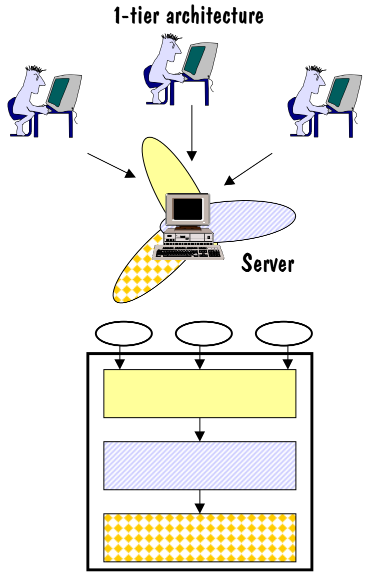
A game of boxes and arrows



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.

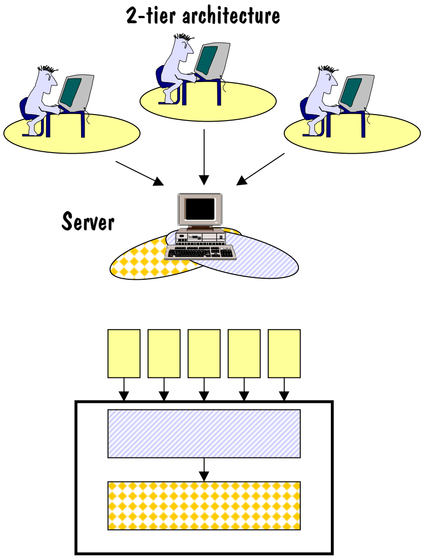
- ❑ Each box represents a part of the system.
- ❑ Each arrow represents a connection between two parts of the system.
- ❑ The more boxes, the more modular the system: more opportunities for distribution and parallelism. This allows encapsulation, component based design, reuse.
- ❑ The more boxes, the 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 capacity of the computers involved and the advantages and disadvantages of the different architectures.

Architectures (1): fully centralized



- 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. (This are the "dumb" terminals).
- This was the typical architecture of mainframe applications, 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.
- This is not as unfashionable as one may think: network computing is based on similar ideas!

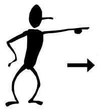
Architecture (2): 2 tier system



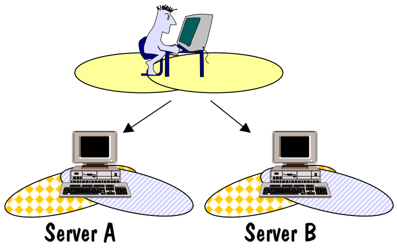
- As computers became more powerful, it was possible to move the presentation layer to the client. This has several advantages:
  - → Clients are independent of each other: one could have several presentation layers depending on what each client wants to do.
  - → One can take advantage of the computing 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 to think about federating these systems by linking several of them.
  - → The resource manager only sees one client: the application logic. This greatly helps with performance since there are no connections/sessions to maintain.

Technical aspects of the 2 tier architecture

- There are clear technical advantages: work within the server takes place within one scope (almost as in 1 tier), the design is tighter and can be optimized, less race conditions to worry about (still easy to manage and control), etc.
- However, not all are advantages:
  - →The system has to deal with all possible connections. The maximum number of clients is given by the number of connections supported by the server.
  - →Clients are "tied" to the system since there is no standard presentation layer. If one wants to connect to two systems, then the client needs two presentation layers.
  - →There is no failure or load encapsulation. If the system fails, nobody can work. Similarly, the load created by a client will directly affect the work of others since they are all competing for the same resources.
  - →The design of the application logic and the resource manager is tightly coupled, making it very difficult to change or separate but also more efficient.
  - →The design is complex and difficult to port to other platforms.

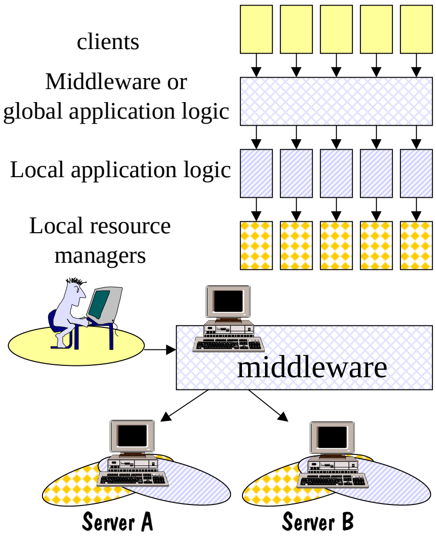


The client is always right



- Clients end up wanting to access two or more systems. With a 2-tier architecture, this creates several problems:
  - →the underlying systems don't know about each other; there is no common business logic. If it is necessary, it needs to be implemented at the client.
  - →The underlying systems are probably different. The complexity of dealing with two heterogeneous systems needs to be addressed by the client.
  - →The client becomes responsible for knowing where things are, how to get to them, and how to ensure consistency!
- This is tremendously inefficient from all points of view (very fat clients are not a solution).
- There is very little that can be done to solve this problems if staying within the 2 tier model. It can be solved by adding a level of indirection: MIDDLEWARE

Middleware

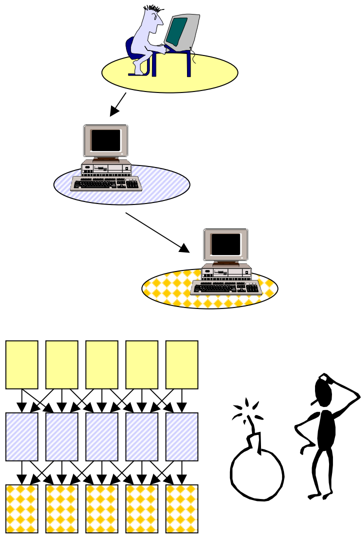


- ❑ Middleware is just a level of indirection between clients and other layers of the system.
- ❑ It introduces an additional layer of business logic encompassing all underlying systems.
- ❑ By doing this, a middleware system:
  - → simplifies the design of the clients by reducing the number of interfaces,
  - → 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 ...

Technical aspects of middleware based systems

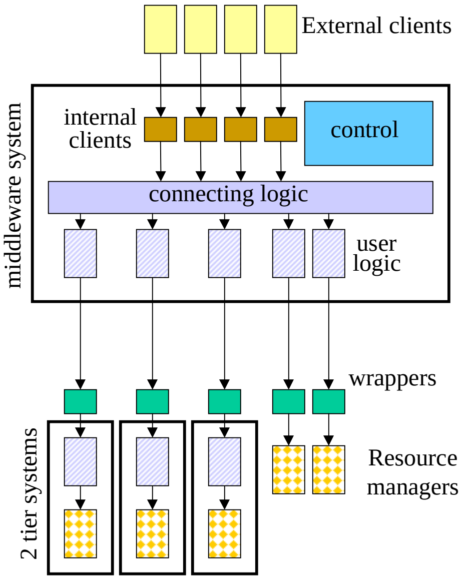
- ❑ The introduction of a middleware layer helps in that:
  - → the number of necessary interfaces is greatly reduced:
    - • clients see only one system (the middleware),
    - • local applications see only one system (the middleware),
  - → it centralizes control (middleware systems themselves are usually 2 tier),
  - → it makes necessary functionality widely available to all clients,
  - → it allows to implement functionality that otherwise would be very difficult to provide, and
  - → it is a first step towards dealing with application heterogeneity (some forms of it).
- ❑ The middleware layer does not help in that:
  - → it is another indirection level,
  - → it is complex software,
  - → it is a development platform, not a complete system, and
  - → middleware functionality is poorly understood.

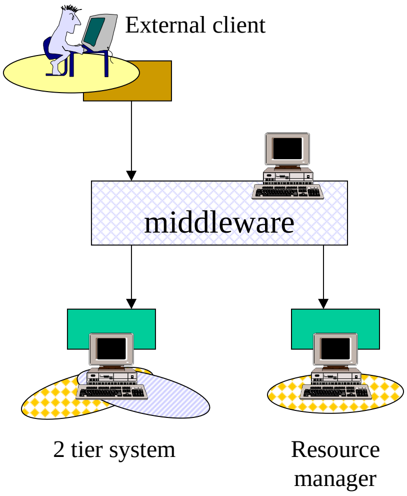
Architecture (3): 3 tier system



- In a 3 tier system, the three layers are fully separated.
- For some people, a middleware based system is a 3 tier architecture. This is a bit oversimplified but conceptually correct since the underlying systems can be treated as black boxes. In fact, 3 tier makes only sense in the context of middleware systems (otherwise the client has the same problems as in a 2 tier system!).
- We will see examples of this architecture when concrete middleware systems are discussed.
- A 3 tier systems has the same advantages as a middleware system and also its disadvantages.
- In practice, things are not as simple as they seem … there are several hidden layers that are not necessarily trivial: the wrappers.

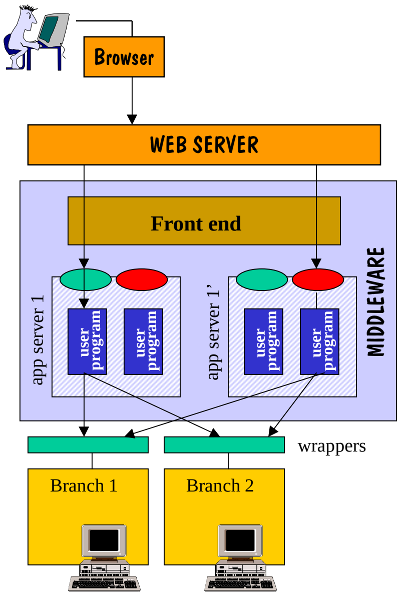
A real 3 tier middleware based system ...



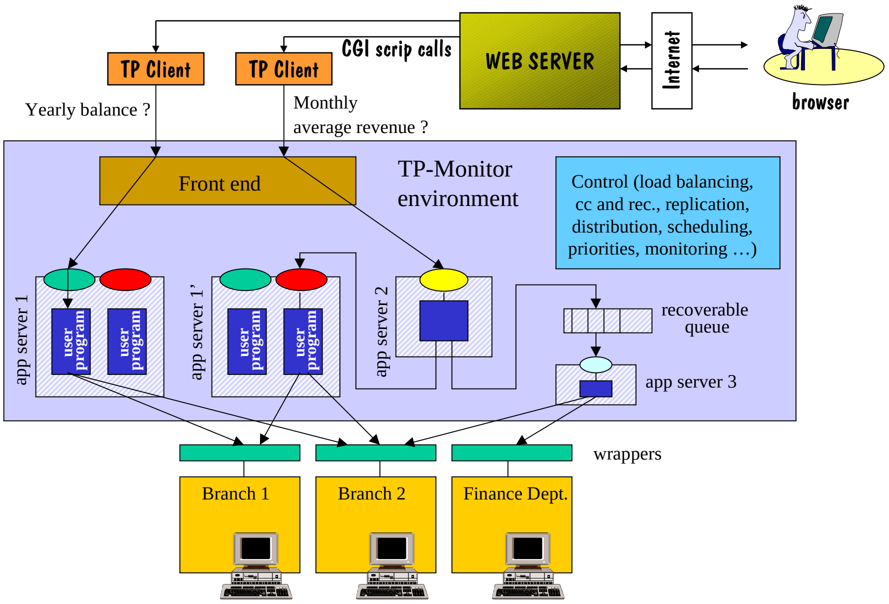


The Web as software layer ...

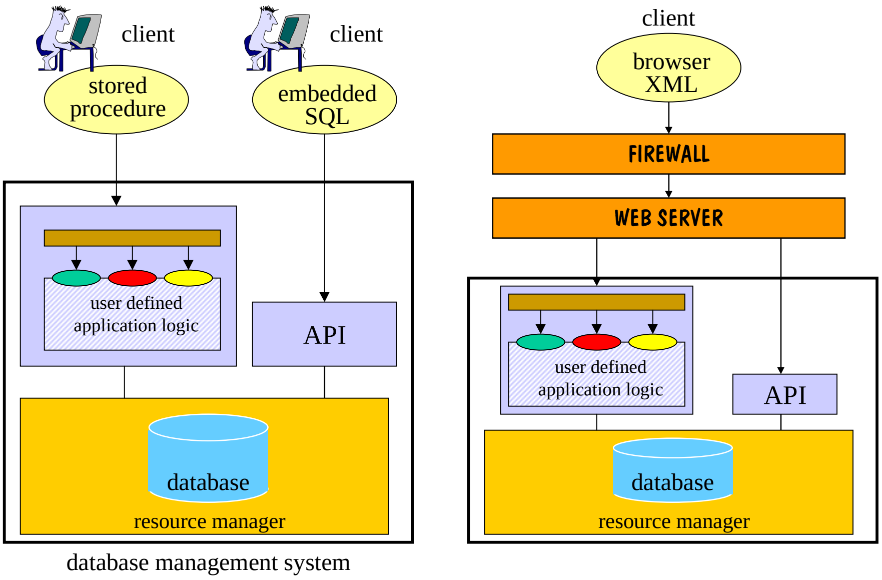
- ❑ The WWW suddenly opened up software systems that had remained hidden within the IT organization of a company
- ❑ It is not that new types of interactions were possible. Behind the WWW there is the same client/server model as in basic RPC. However, the WWW made everything much easier, cheaper and efficient
  - →integration at the level of user interface became possible
  - →services could be accessed from anywhere in the world
  - →the clients could now be not just an internal or selected user but anybody with a browser



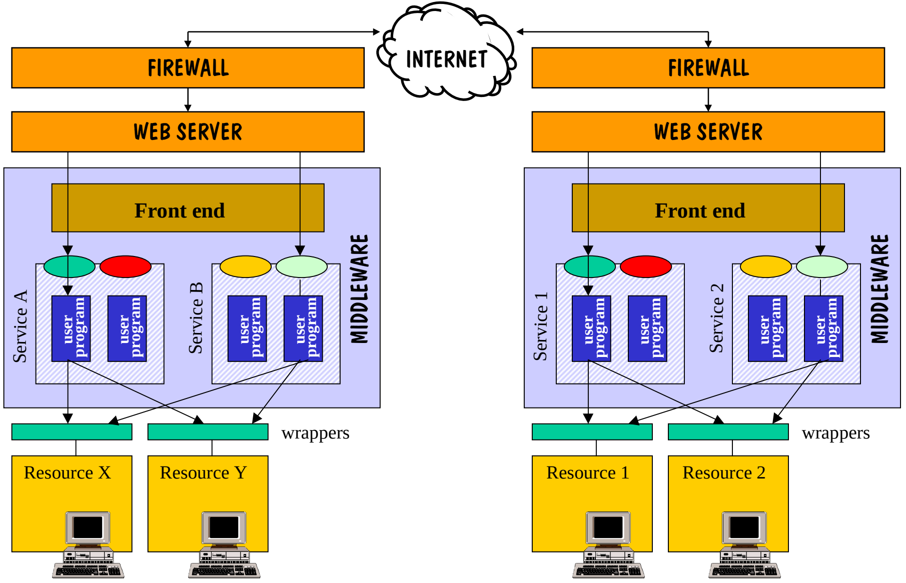
... on top of existing systems



Remote clients

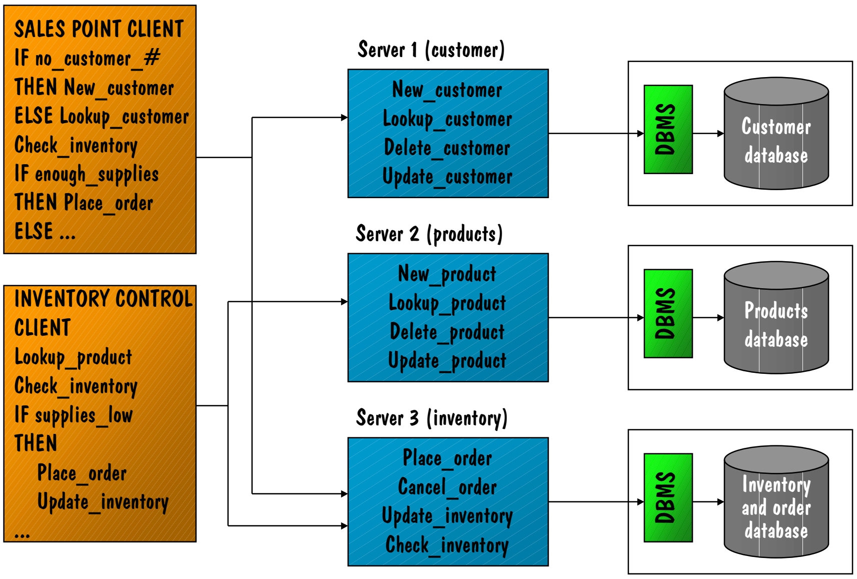


Business to Business (B2B)



Middleware platforms:
From RPC to Enterprise Application Integration

Generic distributed application



```
SALES POINT CLIENT
IF no_customer_#
THEN New_customer
ELSE Lookup_customer
Check_inventory
IF enough_supplies
THEN Place_order
ELSE ...
```

```
INVENTORY CONTROL
CLIENT
Lookup_product
Check_inventory
IF supplies_low
THEN
    Place_order
    Update_inventory
...
```

Server 1 (customer)

New_customer
Lookup_customer
Delete_customer
Update_customer

Server 2 (products)

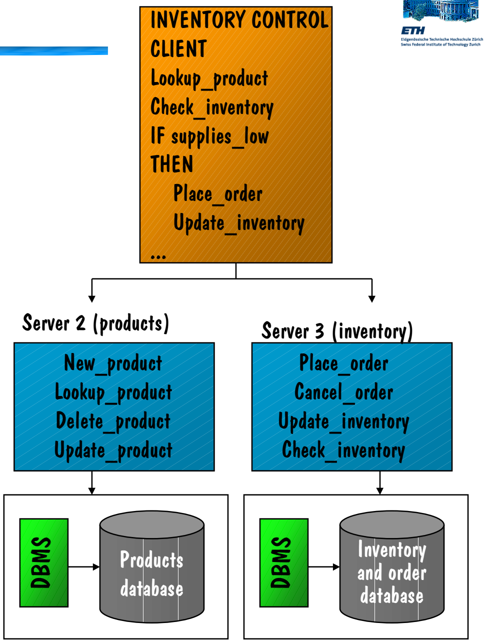
New_product
Lookup_product
Delete_product
Update_product

Server 3 (inventory)

Place_order
Cancel_order
Update_inventory
Check_inventory

What can go wrong here?

- ❑ RPC is a point to point protocol in the sense that it supports the interaction between two entities (the client and the server)
- ❑ When there are more entities interacting with each other (a client with two servers, a client with a server and the server with a database), RPC treats the calls as independent of each other. However, the calls are not independent
- ❑ Recovering from partial system failures is very complex. For instance, the order was placed but the inventory was not updated, or payment was made but the order was not recorded …
- ❑ Avoiding these problems using plain RPC systems is very cumbersome



INVENTORY CONTROL
CLIENT
Lookup_product
Check_inventory
IF supplies_low
THEN
Place_order
Update_inventory
...

Server 2 (products)
New_product
Lookup_product
Delete_product
Update_product

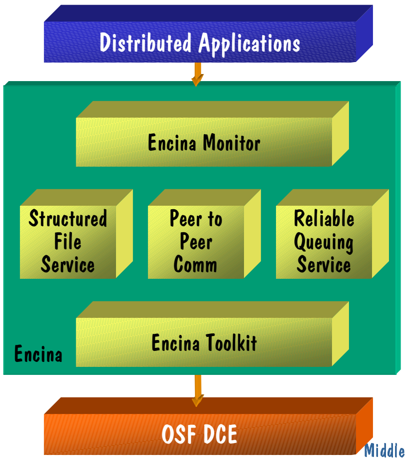
Server 3 (inventory)
Place_order
Cancel_order
Update_inventory
Check_inventory

DBMS
Products database

DBMS
Inventory and order database

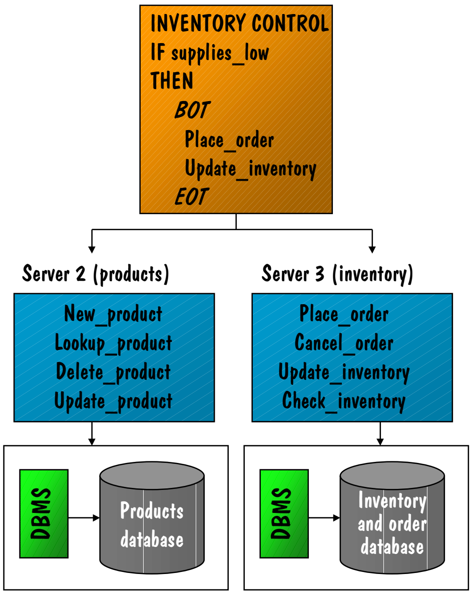
Transactional RPC

- ❑ The solution to this limitation is to make RPC calls transactional, that is, instead of providing plain RPC, the system should provide TRPC
- ❑ What is TRPC?
  - →same concept as RPC plus …
  - →additional language constructs and run time support (additional services) to bundle several RPC calls into an atomic unit
  - →usually, it also includes an interface to databases for making end-to-end transactions using the XA standard (implementing 2 Phase Commit)
  - →and anything else the vendor may find useful (transactional callbacks, high level locking, etc.)
- ❑ Simplifying things quite a bit, one can say that, historically, TP-Monitors are RPC based systems with transactional support. An example: Encina



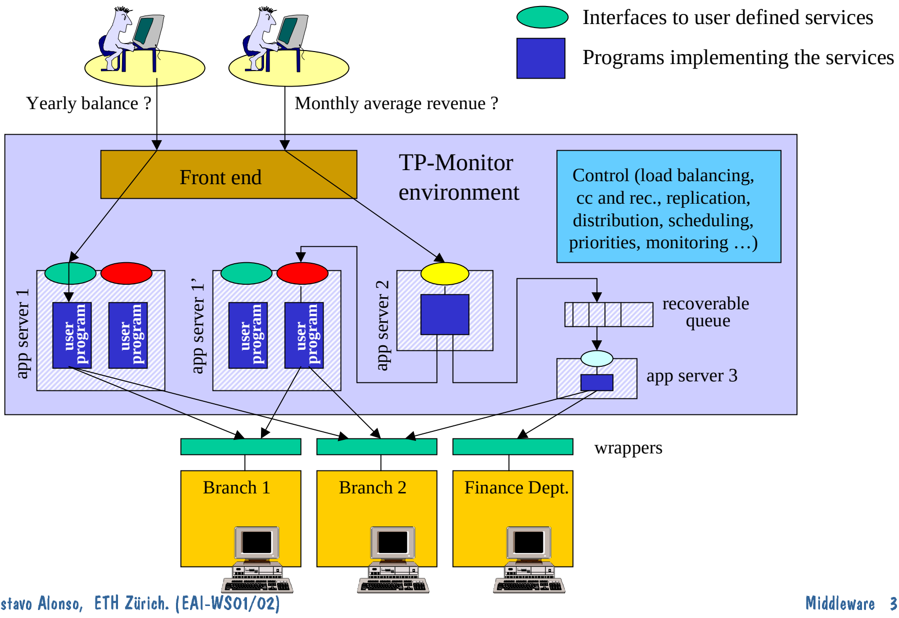
Distributed Applications
Encina Monitor
Structured File Service
Peer to Peer Comm
Reliable Queuing Service
Encina
Encina Toolkit
OSF DCE

TP-Monitors



- ❑ The design cycle with a TP-Monitor is very similar to that of RPC:
  - →define the services to implement and describe them in IDL
  - →specify which services are transactional
  - →use an IDL compiler to generate the client and server stubs
- ❑ Execution requires a bit more control since now interaction is no longer point to point:
  - →transactional services maintain context information and call records in order to guarantee atomicity
  - →stubs also need to support more information like transaction id and call context
- ❑ Complex call hierarchies are typically implemented with a TP-Monitor and not with plain RPC

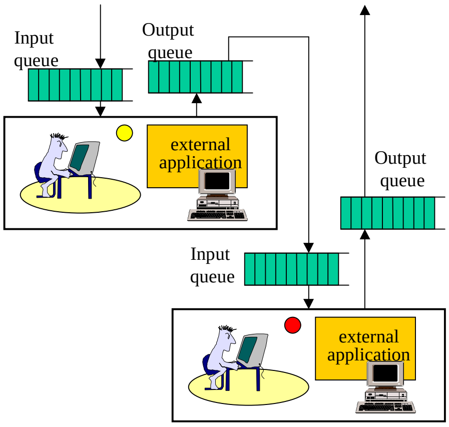
TP-Monitor Example



Asynchronous calls

- The solution is to entirely separate the client and the server. Instead of the client calling a procedure at the server and waiting for the server to execute the procedure and return a result, the client simply sends the call to the server without waiting. The server processes the call in due time and sends a message with the response
- The exchange typically occurs using persistent queues where these messages are stored until the client or the server pick them up
- These queues can be treated as separate entities and can be made persistent, transactional, indexed, priority based, multi-user, etc.
- Asynchronous calls are, but their very nature, point to point. Although m:n relations are feasible, the system supports each individual call as a separate entity (much like RPC). There is no easy way to link several asynchronous calls into an atomic unit
- Asynchronous calls and transactional calls are complementary aspects of distributed systems:
  - →transactional calls are for highly consistent on-line operations
  - →asynchronous calls are for information dissemination across different systems
- Note: TP-Monitors were the first systems to provide the type of persistent queuing used today

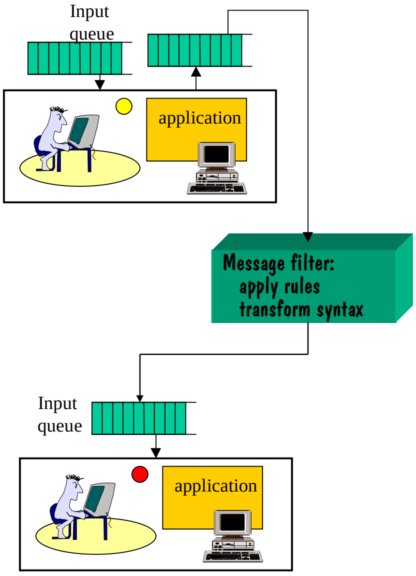
Queuing systems



- Queuing systems implement the asynchronous interaction.
- Each element in the system communicates with the rest via persistent queues. These queues store messages transactionally. The flow of control from queue to queue is defined by the user as limited form of process and constitutes the application logic.
- Queuing systems offer significant advantages over traditional solutions in terms of fault tolerance and overall system flexibility: applications do not need to be there at the time a request is made!
- Queues provide a way to communicate across heterogeneous networks and systems while still being able to make some assumptions about the behavior of the messages.
- They can be used embedded (workflow, TP-Monitors) or by themselves (MQSeries, Tuxedo/Q).

Message brokers

- Message brokers are the latest incarnation of queuing systems
  - →it is the same concept as a queuing system (often the same implementation) plus ...
  - →additional services that permit filtering and transforming messages as they move from queue to queue
- Why are they needed?
  - →In conventional queuing systems, message formatting must be done by all senders individually so that the receiver understands the message
  - →A message broker centralizes this formatting in a single point, thereby facilitating the design



Input queue

application

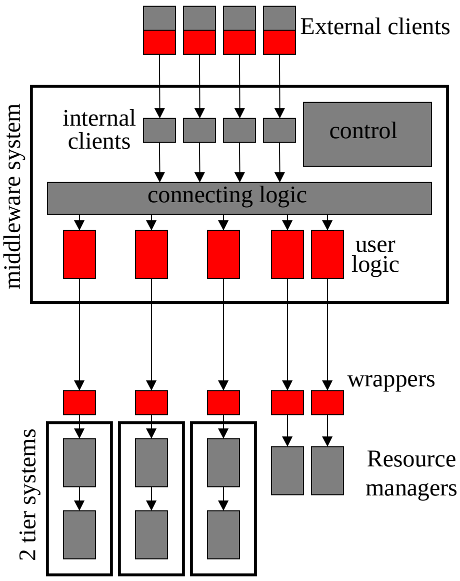
Message filter:
apply rules
transform syntax

Input queue

application

... and JAVA?

- Developing a large information system has never been easy. With the advent of the Internet and the scalability problems it creates, today's information systems are becoming incredibly complex. Nevertheless, and in spite of the help that middleware provides, most development is entirely ad-hoc. That means, it is expensive, non portable, a legacy problem ... (this is also true of CORBA since with CORBA the code developed is platform dependent!)
- For vendors, to provide off-the-shelf solutions is not a realistic option since there are too many platforms that would fragment the market and make development considerable difficult (note that parallelism of what SAP has meant for databases and what could be done for middleware and three tier architectures in general).
- A step in this direction is J2EE (and the competing proposal .NET)



External clients

middleware system

internal clients

control

connecting logic

user logic

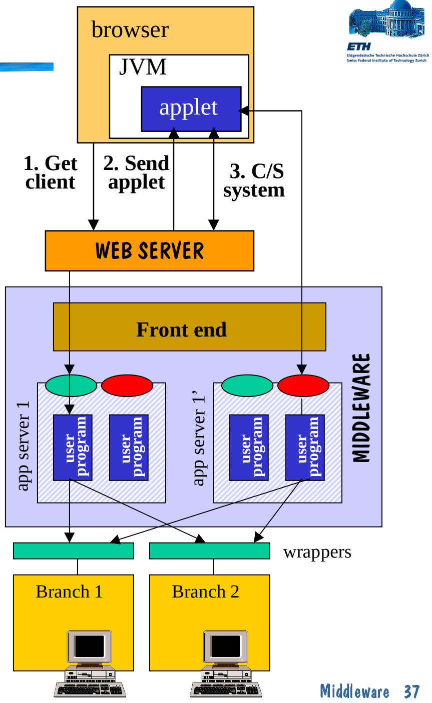
wrappers

Resource managers

2 tier systems

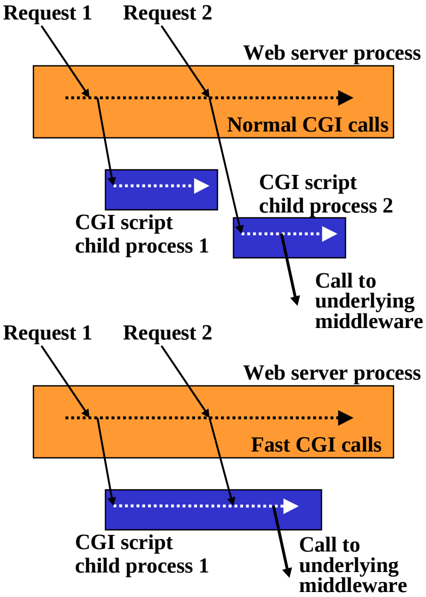
Applets and clients

- ❑ The problem of the using a web browser as universal client is that it does not do much beyond displaying data (it is a thin client):
  - →multiple interactions are needed to complete complex operations
  - →the same operations must be done over and over again for all clients
  - →the processing power at the client is not used
- ❑ By adding a JVM (Java Virtual Machine) to the browser, now it becomes possible to dynamically download the client functionality (an applet) every time it is needed
- ❑ The client becomes truly independent of the operating system and is always under the control of the server



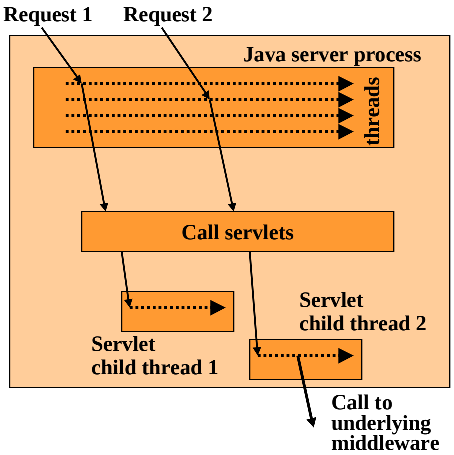
Web server as a client of a EAI system

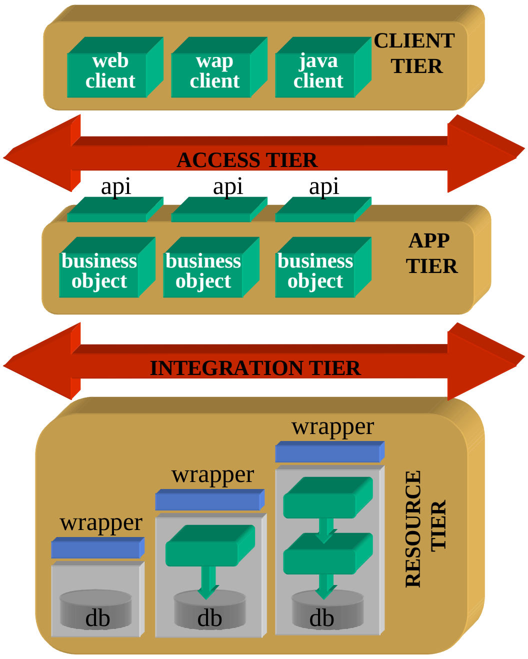
- ❑ CGI scripts were initially widely used as there was no other way of connecting the web server with the IT system so that it could do something beyond sending static documents
- ❑ However, CGI scripts have several problems that are not easy to solve:
  - →CGI scripts are separate processes, requiring additional context switches when a call is made (and thereby adding to the overall delay)
  - →Fast-CGI allows calls to be made to a single running process but it still requires two context switches
  - →CGI is really a quick hack not designed for performance, security, scalability, etc.

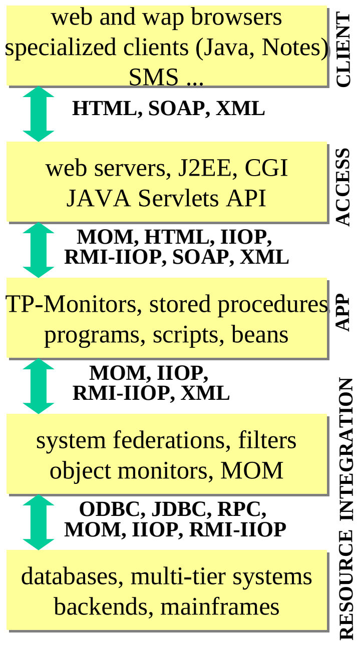

Request 1
Request 2
Web server process
Normal CGI calls
CGI script child process 2
CGI script child process 1
Call to underlying middleware
Request 1
Request 2
Web server process
Fast CGI calls
CGI script child process 1
Call to underlying middleware

Servlets

- ❑ Servlets fulfill the same role as CGI scripts: they provide a way to invoke a program in response to an http request.
- ❑ However:
  - →Servlets run as threads of the Java server process (not necessarily the web server) not as separate OS processes
  - →unlike CGI scripts, that can be written in any language, Servlets are always written in Java (and are, therefore, portable)
  - →can use all the mechanisms provided by the JVM for security purposes
- ❑ The merge of Java based tools and middleware platforms leads to the so called application servers.







CLIENT: web and wap browsers specialized clients (Java, Notes) SMS ...

HTML, SOAP, XML

ACCESS: web servers, J2EE, CGI JAVA Servlets API

MOM, HTML, IIOP, RMI-IIOP, SOAP, XML

APP: TP-Monitors, stored procedures programs, scripts, beans

MOM, IIOP, RMI-IIOP, XML

INTEGRATION: system federations, filters object monitors, MOM

ODBC, JDBC, RPC, MOM, IIOP, RMI-IIOP

RESOURCE: databases, multi-tier systems backends, mainframes