COMP 9322
Software Service Design and Engineering

Lecture 2 (Part1)– Distributed Information Systems

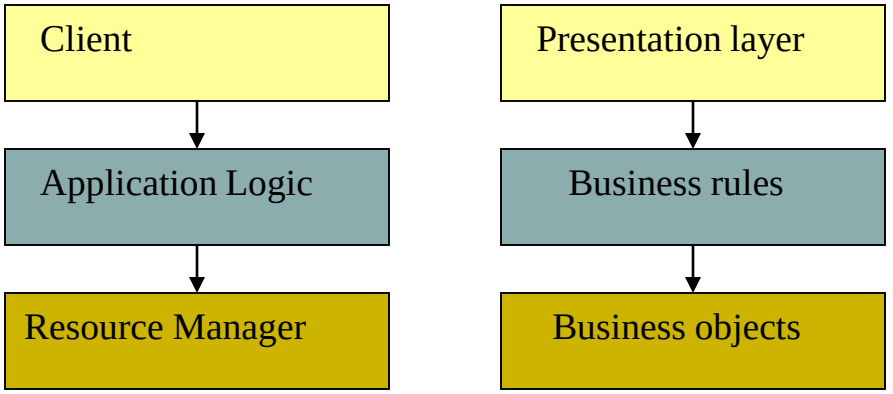
Acknowledgments

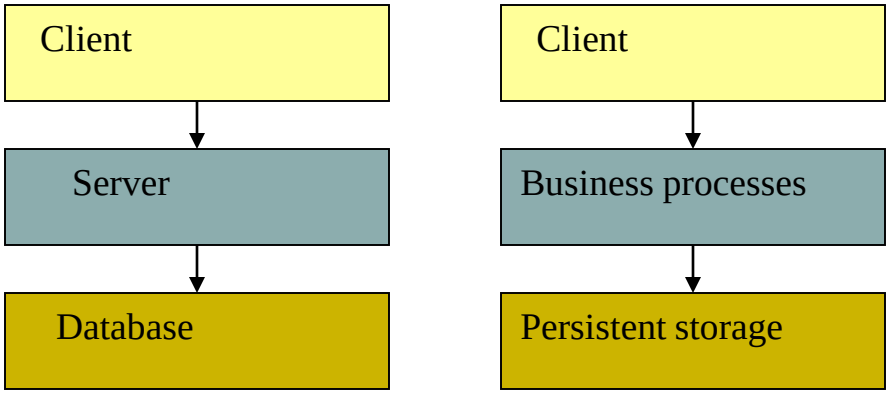
- Parts of some of the slides presented in this course are taken from Dr. Helen Paik's COMP 9322 Course handouts

Layers

- is a construct to logically separate the functionality of an information system.
  - Separation of concerns in relation with the underlying technology
  - If we decompose the software into different conceptual parts we can evolve them separately
- Services and Objects enable us to decompose the software in relation with the concepts of the application domain
- Layers enables us to decompose the software independent from the properties of the application domain.

Layers

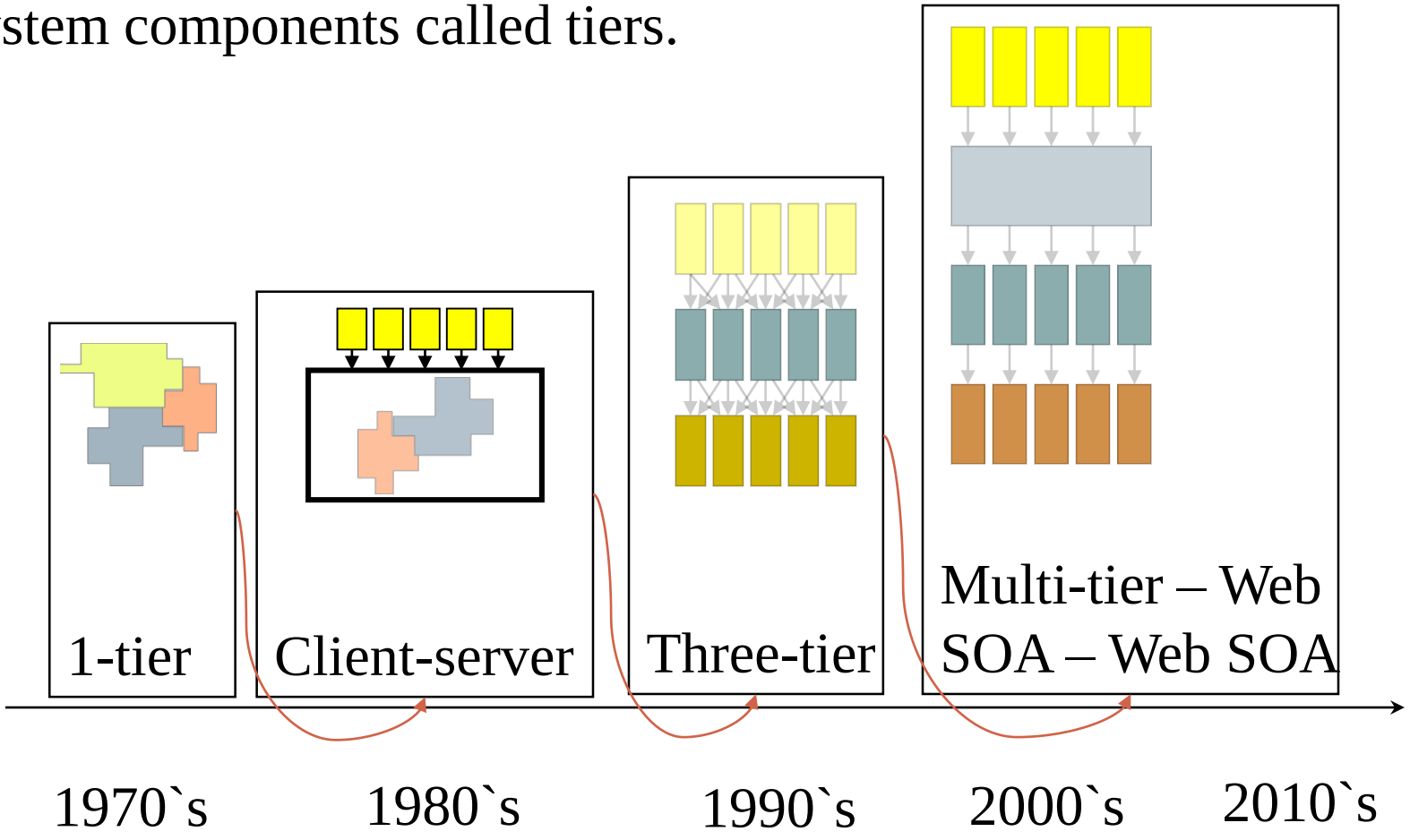




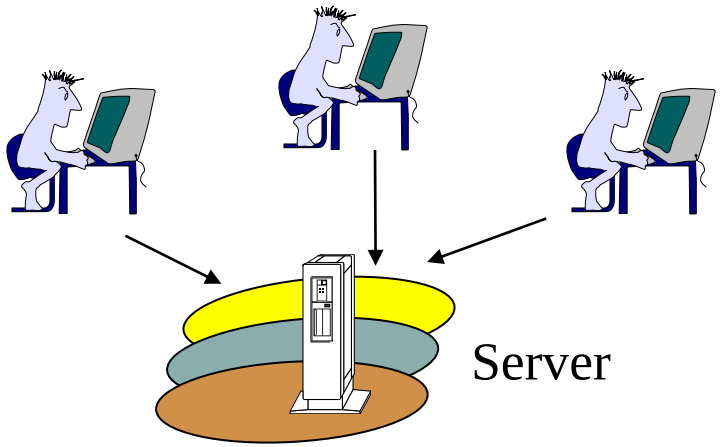
- Presentation:
  - Clients (user or program) wants to perform an operation over the system.
    - Clients interact with the system through a presentation layer.
  - Deals with creating and displaying the (user) interface.
- The application logic:
  - determines what the system actually does.
  - enforcing the business rules and establish the business processes.
  - Can take many forms: programs, constraints, business processes, etc.
- The resource management:
  - deals with the organization (storage, indexing, and retrieval) of the data necessary to support the application logic.
  - Typically a database, a text retrieval system or any other system providing querying capabilities and persistence.

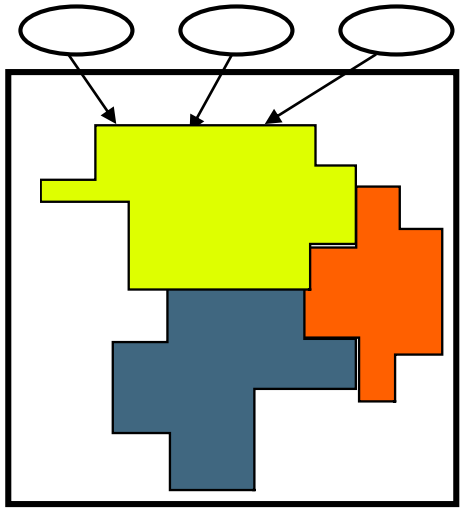
Timeline – Architecture of Information Systems

When implementing real systems
we map layers (logical constructs)
to system components called tiers.



One-tier: Fully Centralized





- PL, AL, RM is a monolithic entity.
- Users/programs access the system through 'dumb' terminals

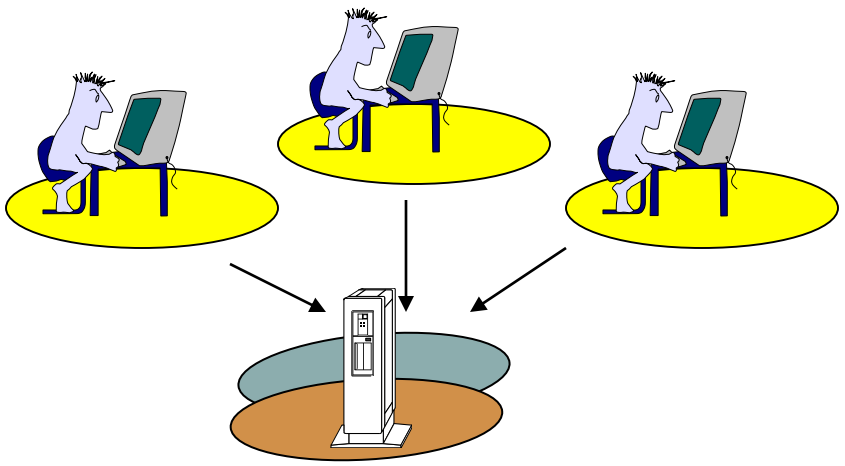
what is displayed and how it appears is controlled by the server.

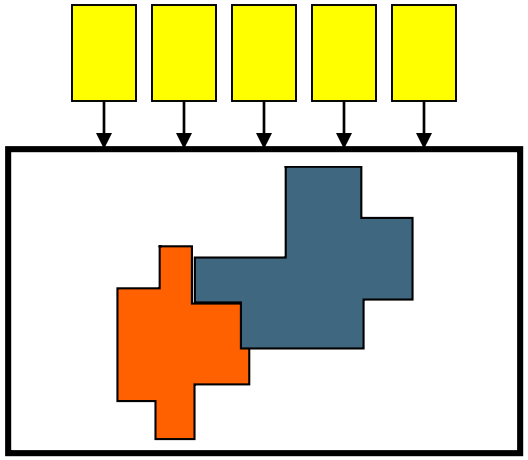
- Typical architecture of mainframes.

Reflections:

- no forced context switches in the control flow,
- centralized, managing and controlling resources is easier,
- the design can be highly optimized no separation between layers.

Two-tier: Client/server





- As computers became cheaper, the presentation layer moved to the client.

Reflections:

- Clients are independent of each other: several presentation layers are possible depending on the client.
- More sophisticated presentation layers: At the same time saving computer resources at the server.
- The server deals with the application and resource: Helps with performance as there are no client connections / sessions to maintain.
- The concept of API (Application Program Interface): An interface to invoke the system from the outside. Federating the systems into a single system.

API in client/server

- Client/server systems introduced the notion of service:

the client invokes a service implemented by the server

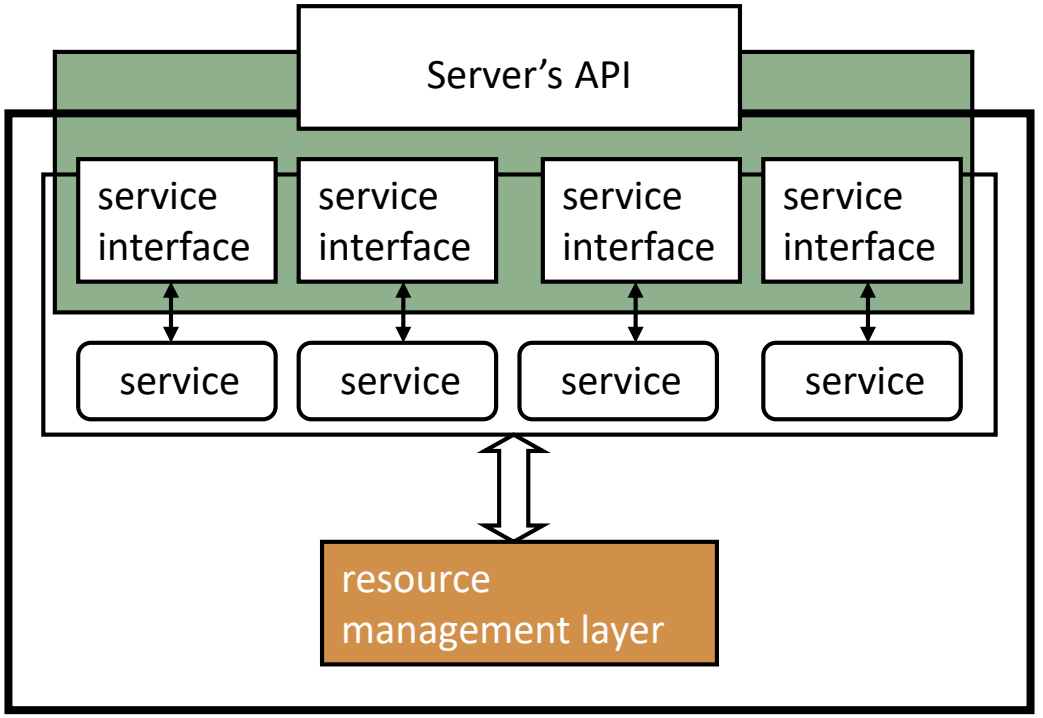
- Client/server systems introduced the notion of service interface:

how the client can invoke a given service

- All together, the interfaces to all the services provided by a server define the server's Application Program Interface (API)

describes how to interact with the server from the outside

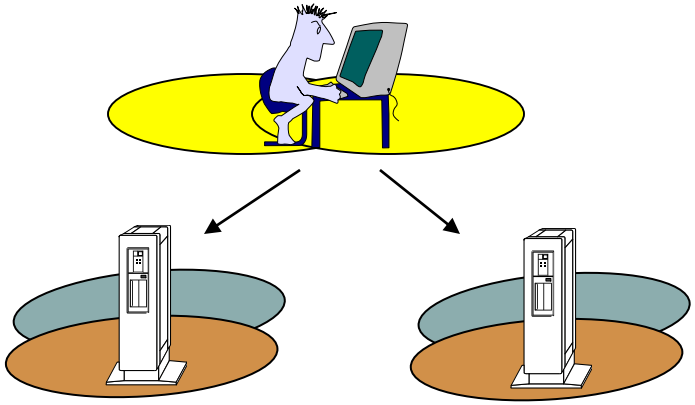
- Standardization efforts were initiated by the need to agree on common APIs for each type of server



Client/Server Architecture

- Advantages:
  - client capacity to off-load work to the clients
  - work within the server takes place within one scope (almost as in 1 tier),
  - the server design is still tightly coupled and can be optimized by ignoring presentation issues
  - relatively easy to manage and control from a software engineering point of view
- Disadvantages:
  - The server has to deal with all possible client 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 server fails, nobody can work. All clients are all competing for the same resources.

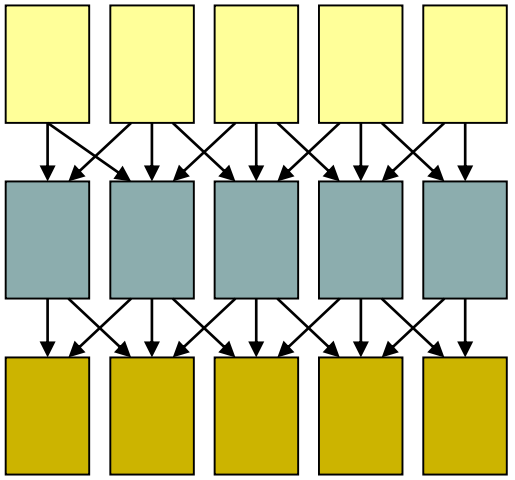
The Main Limitation of Client/Server

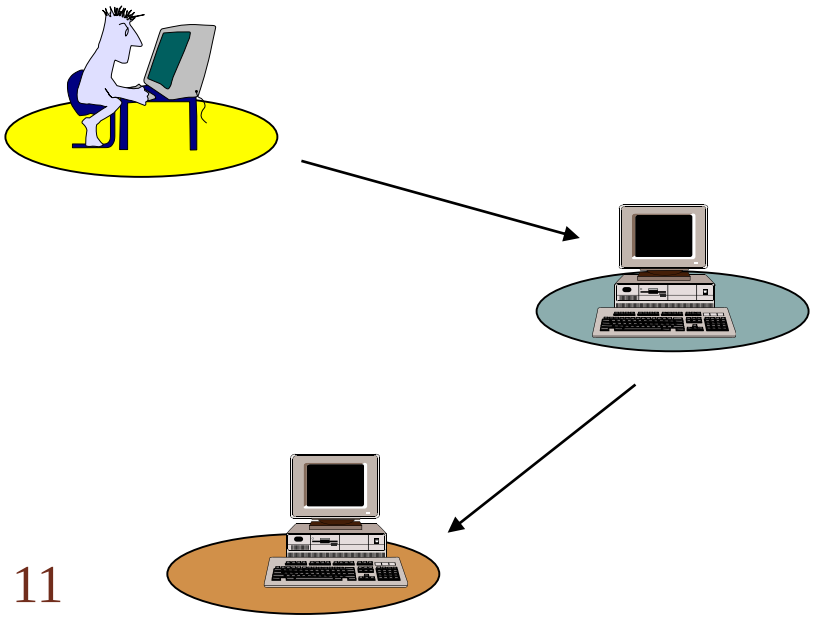


- If clients want to access two or more servers, a 2-tier architecture causes several problems:
  - the underlying systems don't know about each other
  - there is no common business logic
  - the client is the point of integration (increasingly fat clients)

- The responsibility of dealing with heterogeneous systems is shifted to the client.
- The client becomes responsible for knowing where things are, how to get to them, and how to ensure consistency
- This is 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 problem.

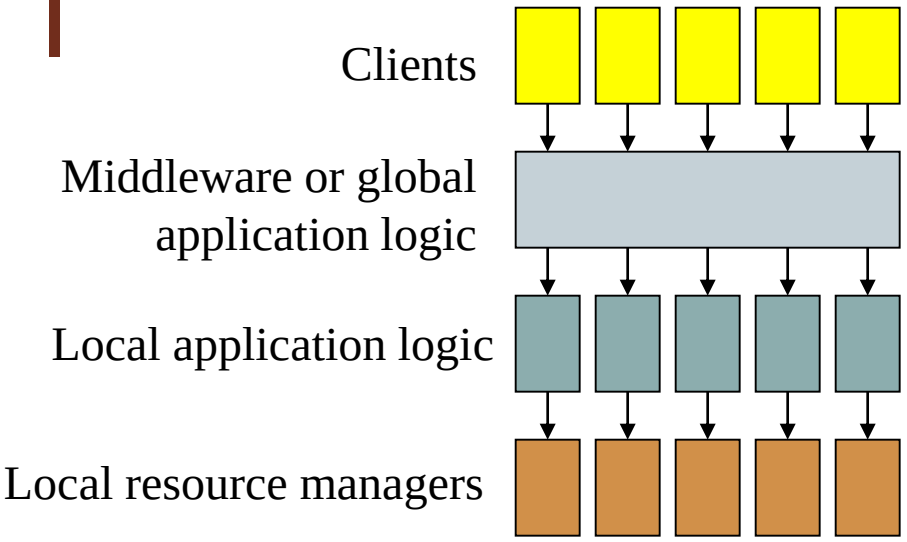
Three tier: middleware

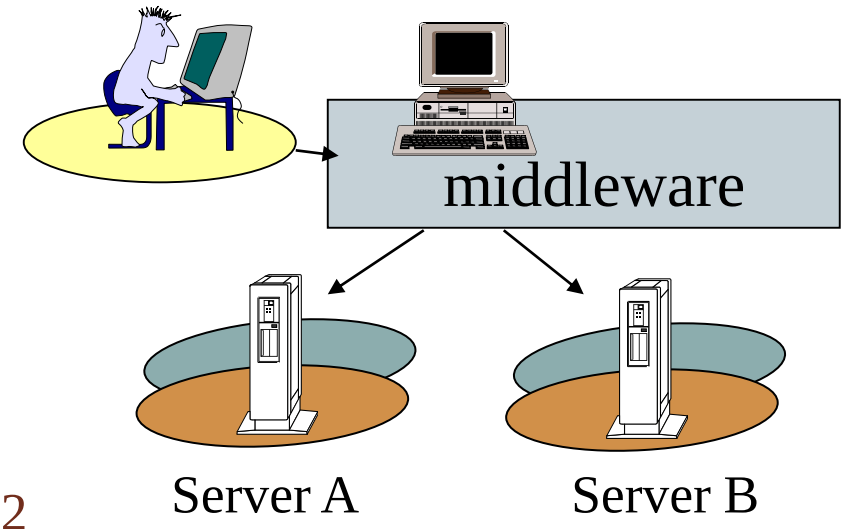




- The three layers are fully separated.
- The layers are typically distributed
  - an advantage of the complete modularity of the design
  - in two tier systems, the server is typically centralized
- A middleware based system is a 3 tier architecture.
  - 3 tier makes only sense in the context of middleware systems (otherwise the client has the same problems as in a 2 tier system).

Middleware



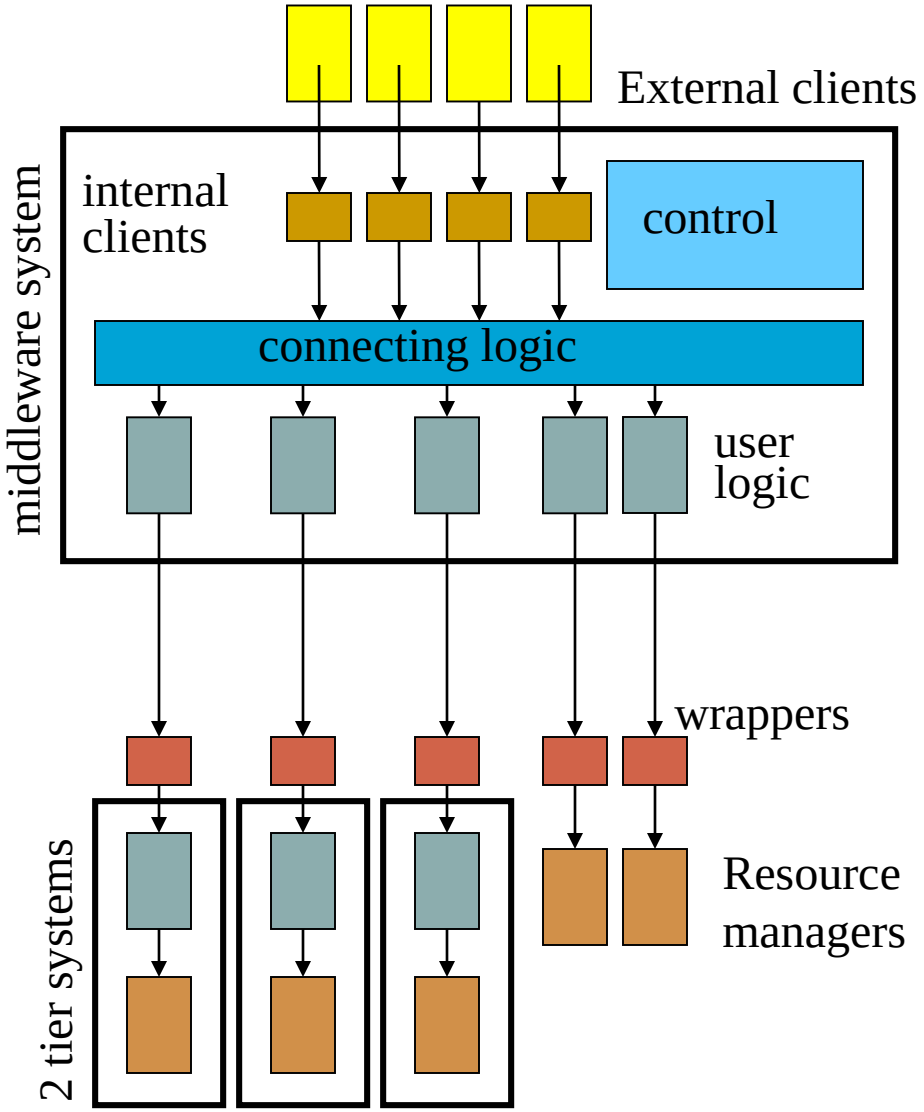


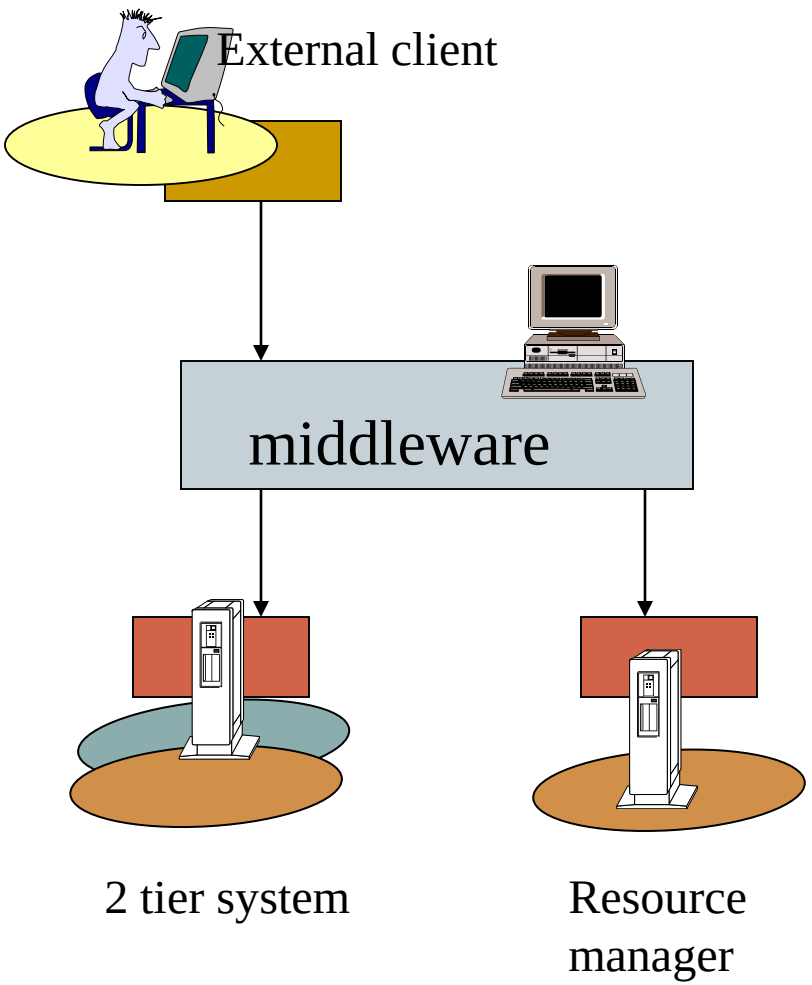
- Middleware is a level of indirection between clients and other layers.
- An additional layer of business logic for all.
  - simplifies the design of the clients by reducing the number of interfaces,
  - provides transparent access to the underlying systems,
  - acts as the platform for inter-system functionality and high level application logic, and
  - takes care of locating resources, accessing them, and gathering results.
- A middleware system is another system! It can also be 1 tier, 2 tier, ...

Middleware Based Architecture

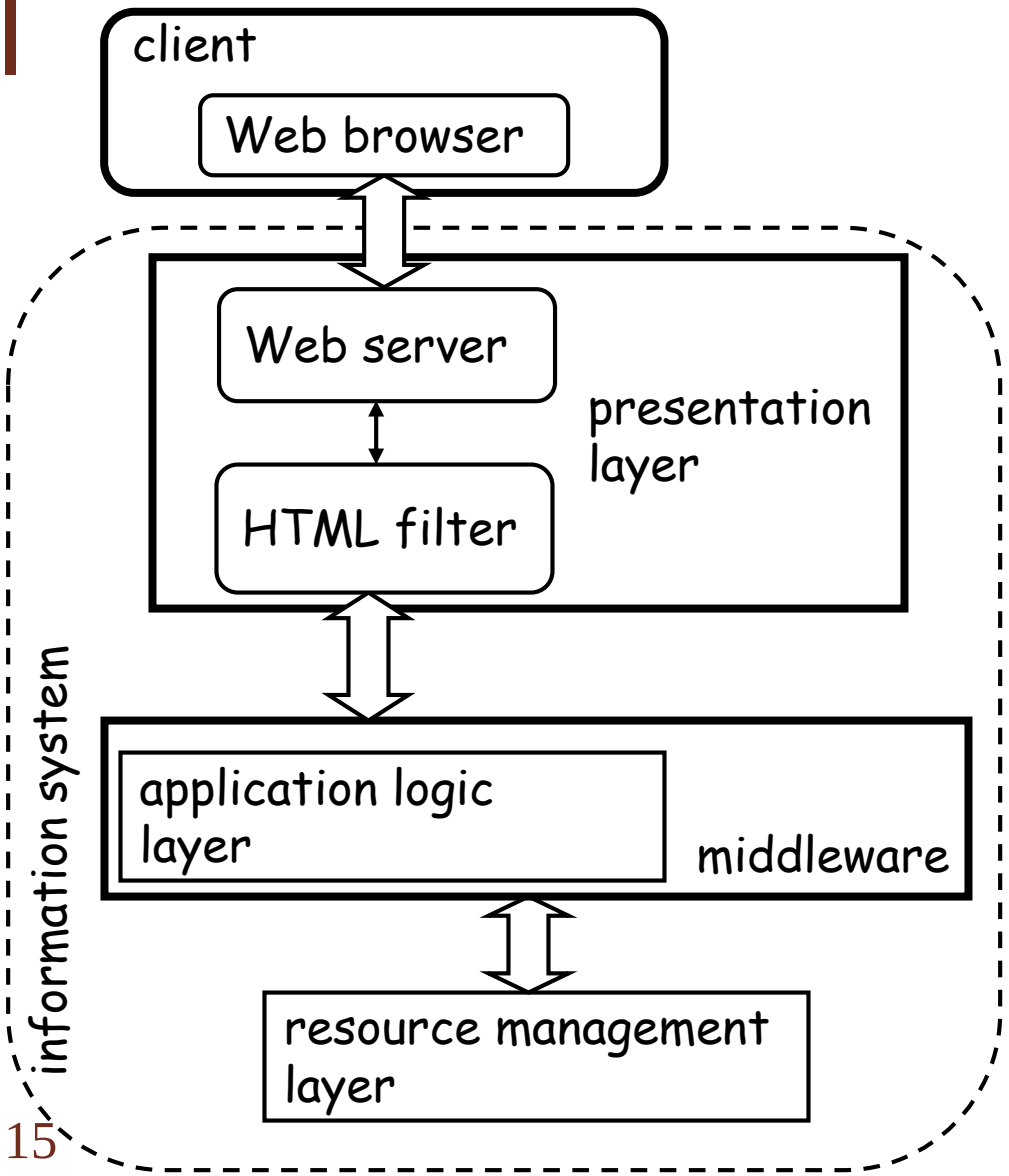
- Advantages:
  - the number of necessary interfaces is greatly reduced:
    - clients see only one system (the middleware),
    - local applications see only one system (the middleware),
  - centralizes control,
  - makes necessary functionality widely available to all clients,
  - a first step towards dealing with application heterogeneity
- Disadvantages:
  - Another indirection level,
  - Complex software,
  - A development platform, not a complete system

A Three-tier Middleware Based System



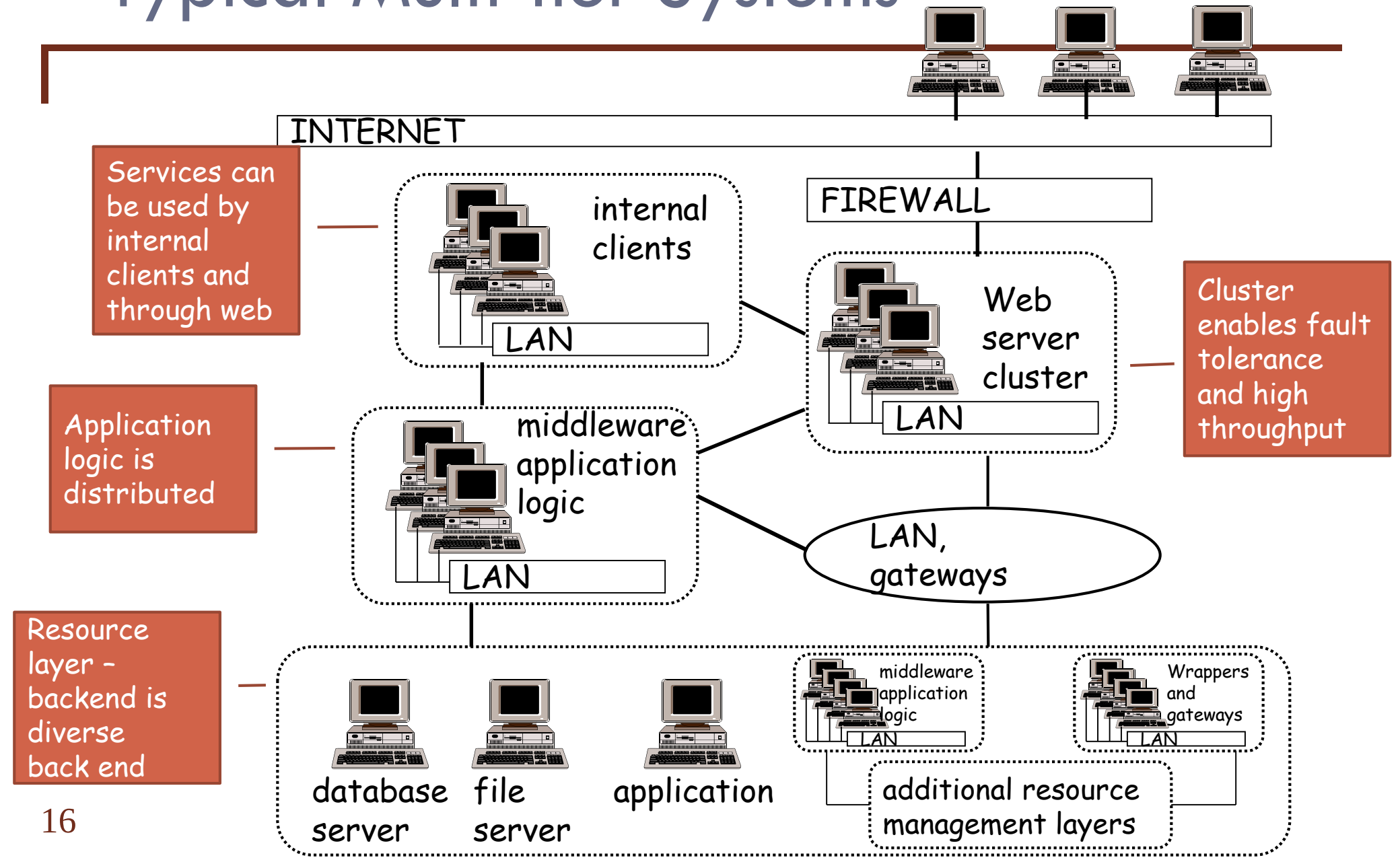


Multi-tier: The Web



- Multi tier connect Three tier systems to each other
  - Add a web layer for clients to access the system through a Web server
- Web layer can be external to the system; Mostly, it is in the presentation layer that resides on the server side
- The Web layer led to the notion of "application servers", which was used to refer to middleware platforms supporting access through the Web

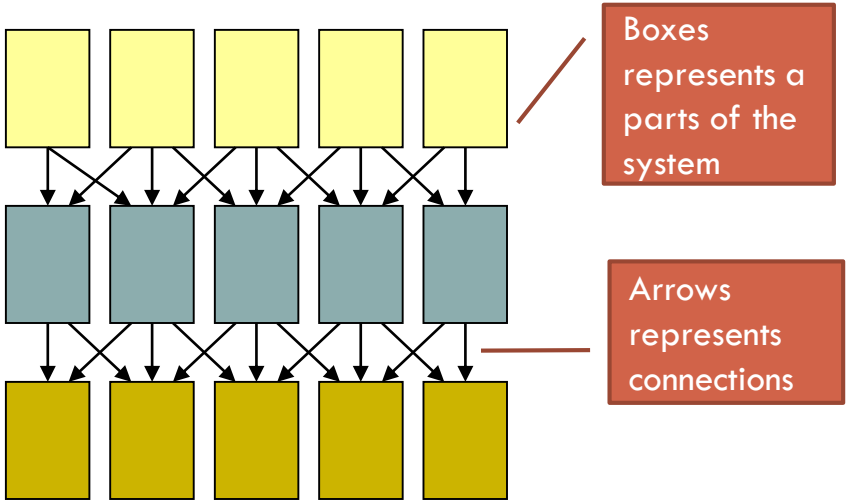
Typical Multi-tier Systems



Multi-tier Architecture Main Disadvantage

- Complexity
  - Too much middleware is involved frequently with redundant functionality
  - As tiers increase the cost of maintenance increase exponentially

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.

- The more boxes the more modular:

More opportunities for distribution and parallelism. This allows encapsulation, component based design, reuse.

- The more boxes, the more arrows:

More sessions 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:

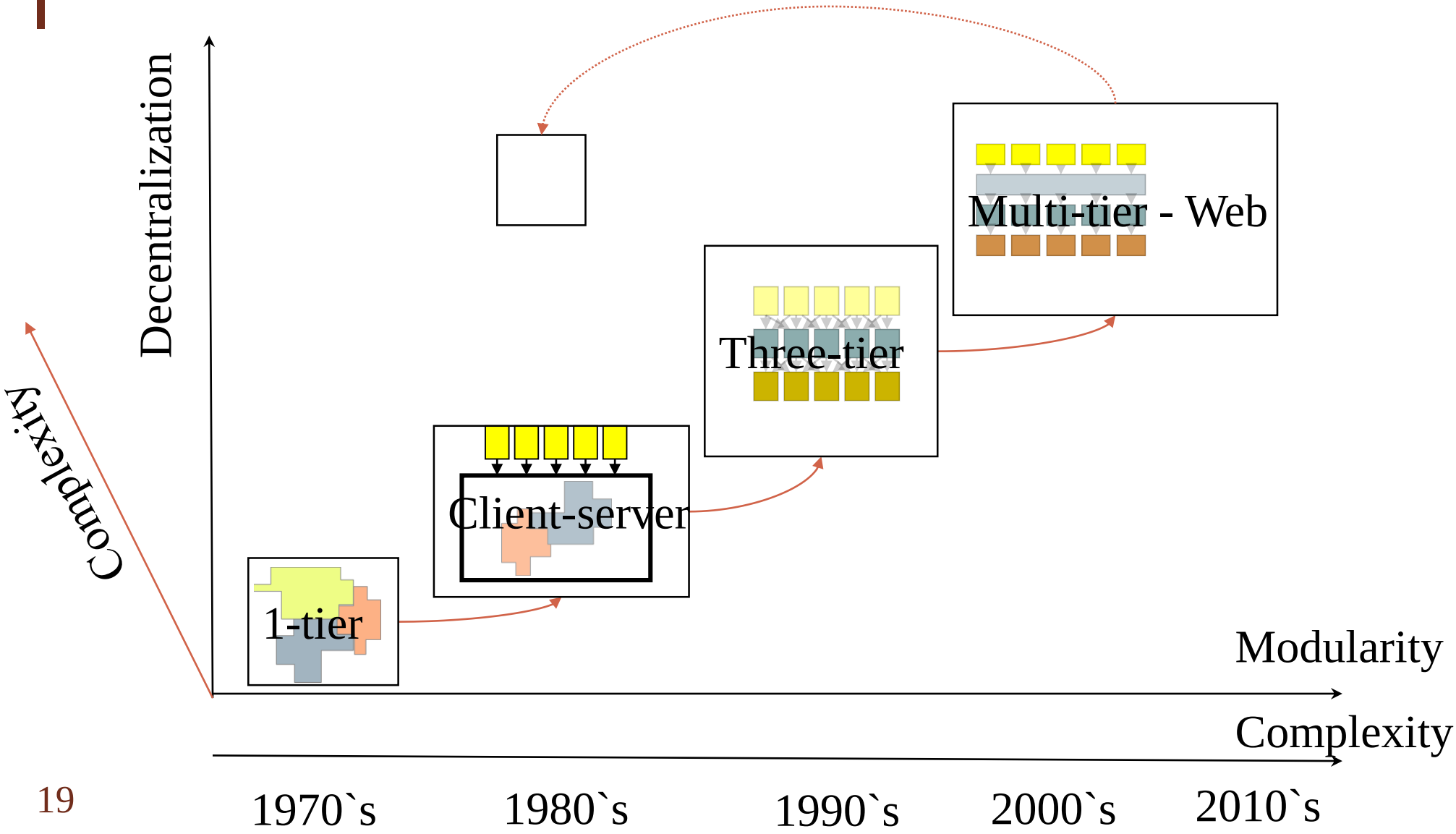
increased intermediate steps to go through to access data.

Performance suffers considerably.

- System designers try to balance:

the flexibility of modular design vs performance demands

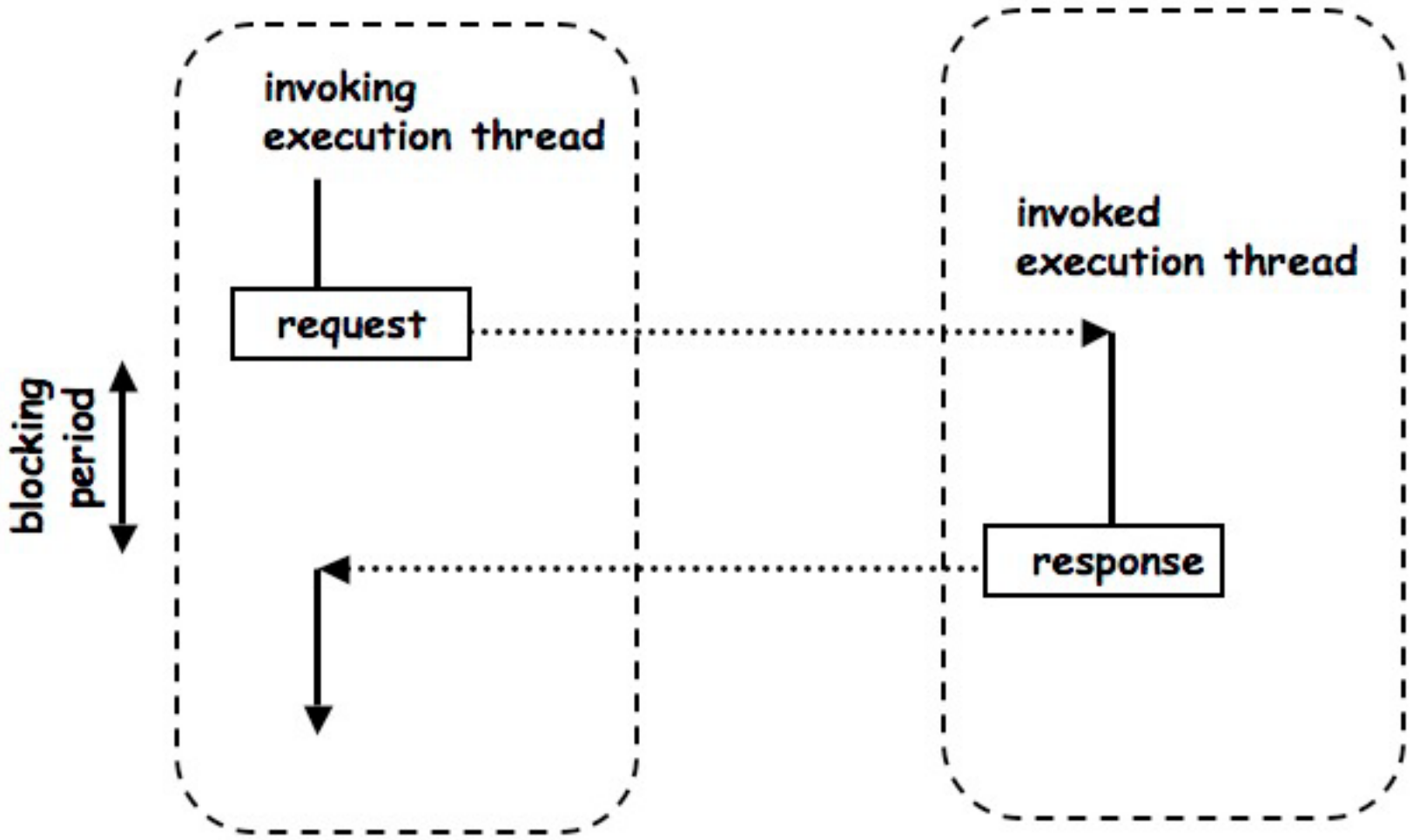
The Evolution



Forms of Communication

- When we separate layers and tiers in an information system, we assume that there is some form of communication between all these elements.
- There are two widely used communication patterns:
  - Blocking Synchronous
  - Non-blocking Asynchronous

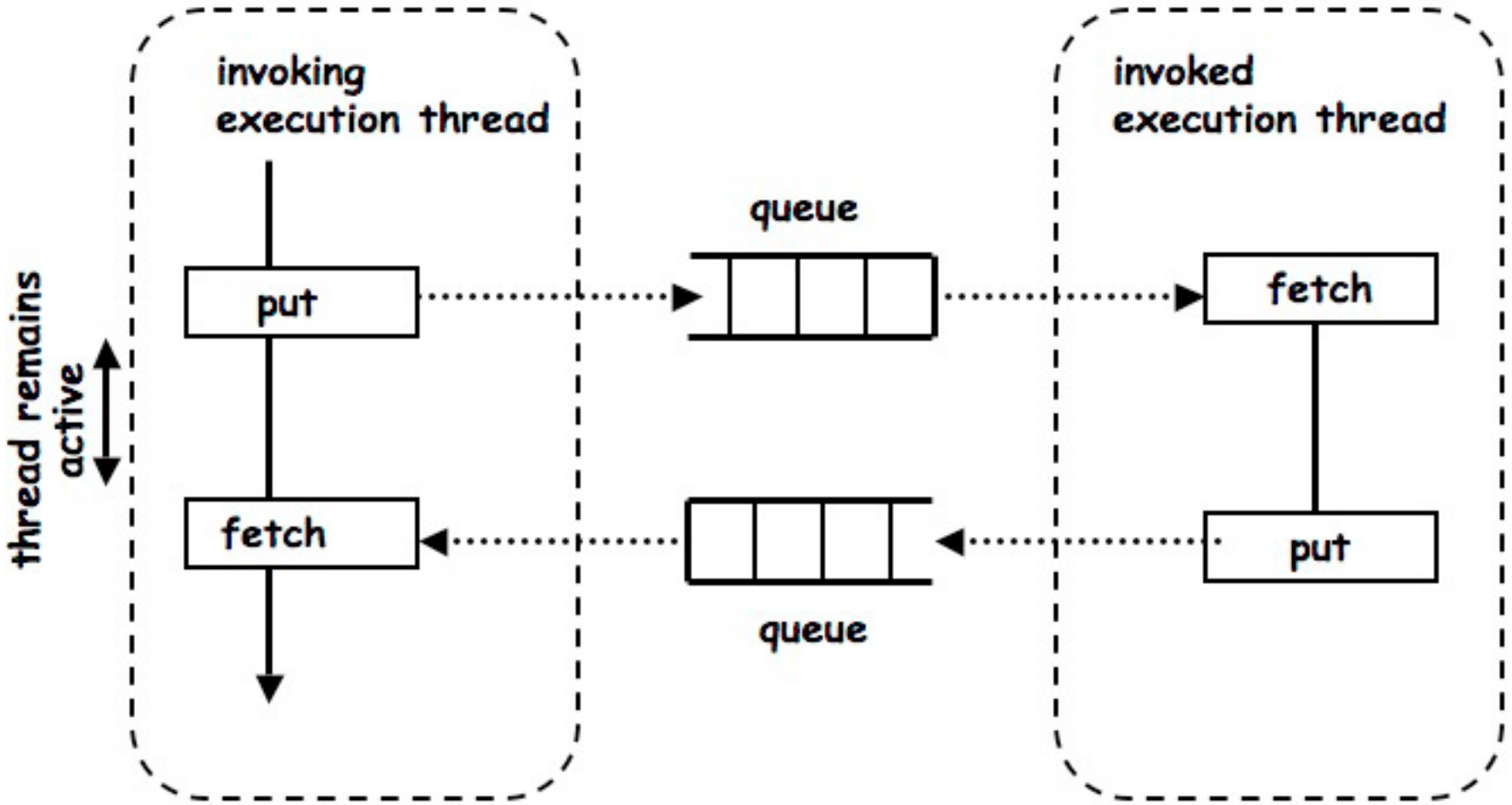
Blocking Synchronous



Blocking-synchronous Interaction

- The main advantage: Simple to understand and implement
  - easier, as the code follow traditional method calls
  - easier to debug (relation between invoking and invoked is clear)
  - widely used in traditional middleware
- The main disadvantage: expensive and wastes resources
  - connection overhead (new connection for each request)
  - calling thread must wait
  - requires both parties to be on-line
    - higher probability of failures
  - not suitable if the number of tiers increases

Non-blocking Asynchronous



Non-blocking Asynchronous Interaction

- Main advantage: less dependency between communicating parties
  - Easier to implement complex interactions between heterogeneous systems
  - Very suitable for non request-response type communications
    - Multicast, Publish/Subscribe
  - Client can continue to run and occasionally check with server to see if a response is ready
- Main disadvantage: adds complexity to client
  - Better to be utilized in less known approaches like reactive systems and event based modelling

References

- Alonso, Casati, Kuno, Machiraju, Web Services: Concepts, Architectures and Applications, Springer 2004
  - The evolution content is partially based on the book
    - An essential reading to understand the evolution