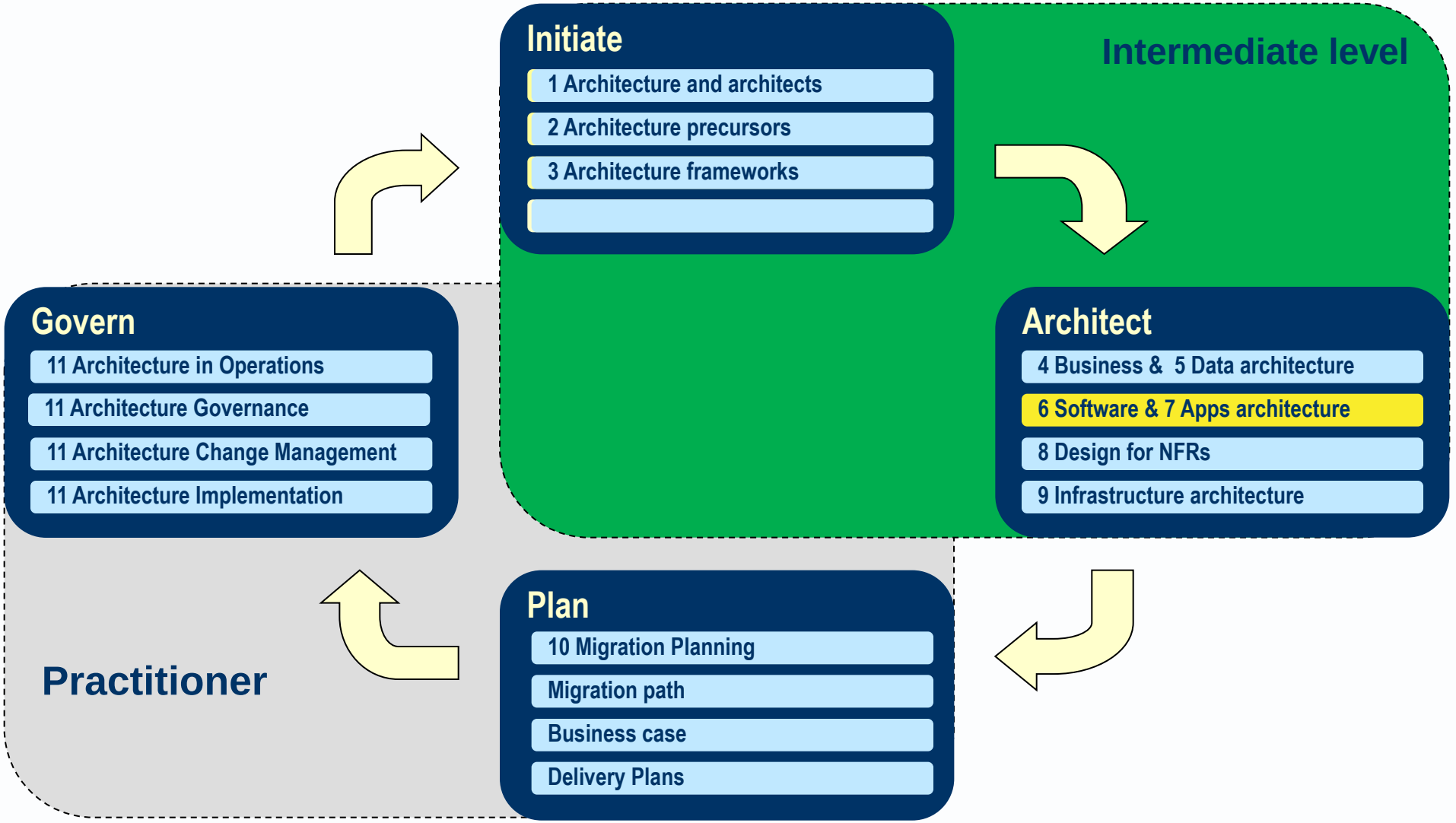
Avancier Reference Model

Software Architecture (ESA 6)

6. Software architecture

Avancier



6: Software domain

- How applications are modularised and interoperate is important to enterprise architecture.

6.1: Foundation (rarely examined)

	Required behaviours	Logical structures	Physical structures
Software	Operation	API	Application component

- **Operation** [a service] that can be requested of a software component. An abstract or virtual operation is an operation for which the signature is defined, but not the internal procedure that implements the operation.

- **API** [an interface] a collection of automated services accessible by software clients; it identifies services, may provide access to them, and hides what performs.

-

- **Application component** [a component] module or object capable of offering automated services. Module is a generic term. Class is the OOP term. An object is an instance of a class; it has identity, behaviour and state; it can perform the general operations of the class and maintain object-specific data item values in its state.

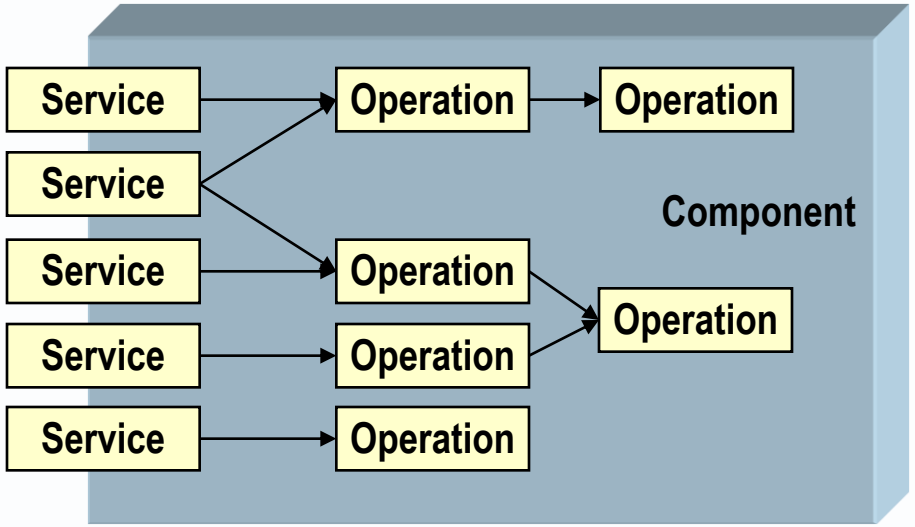
Encapsulation of modules

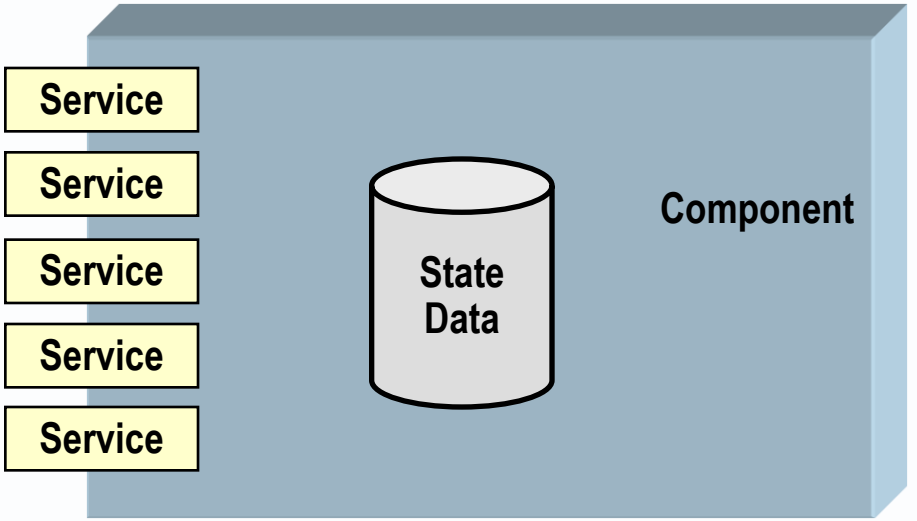
Avancier

Encapsulation

The enclosure within a component of processes, meaning that the inner workings are invisible to outsiders.

Also the enclosure within a component of data, so the only way to access that data is by using the interface of the component.





How well a service should work– the NFRs

- Every client (or client designer) who wants to use a service should know its functional and non-functional characteristics - lest they be unauthorised or unacceptable.

Address Module
Operation: Get Address (Post Code, House Num): Address
Precondition: Post Code is properly formatted
Postcondition: Correct Address is returned
Operation: Get Post Code (Address): Post Code
Precondition: Address is properly formatted
Postcondition: Correct Post Code is returned
Non-functional characteristics – shared by services above
Response time = < 3 seconds
Throughput = 10 per second

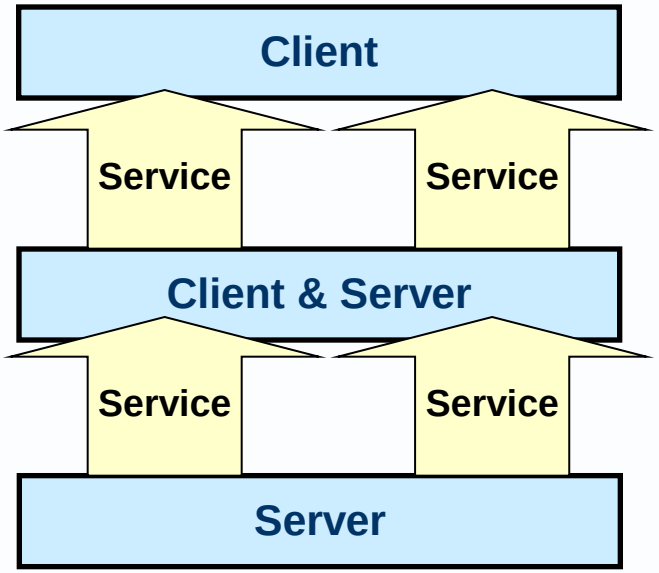
Services can share the same NFRs

Client and server relationships

- **Component role:** a role such as client or server, sender or receiver, prior or successor, played by one component with respect to another.

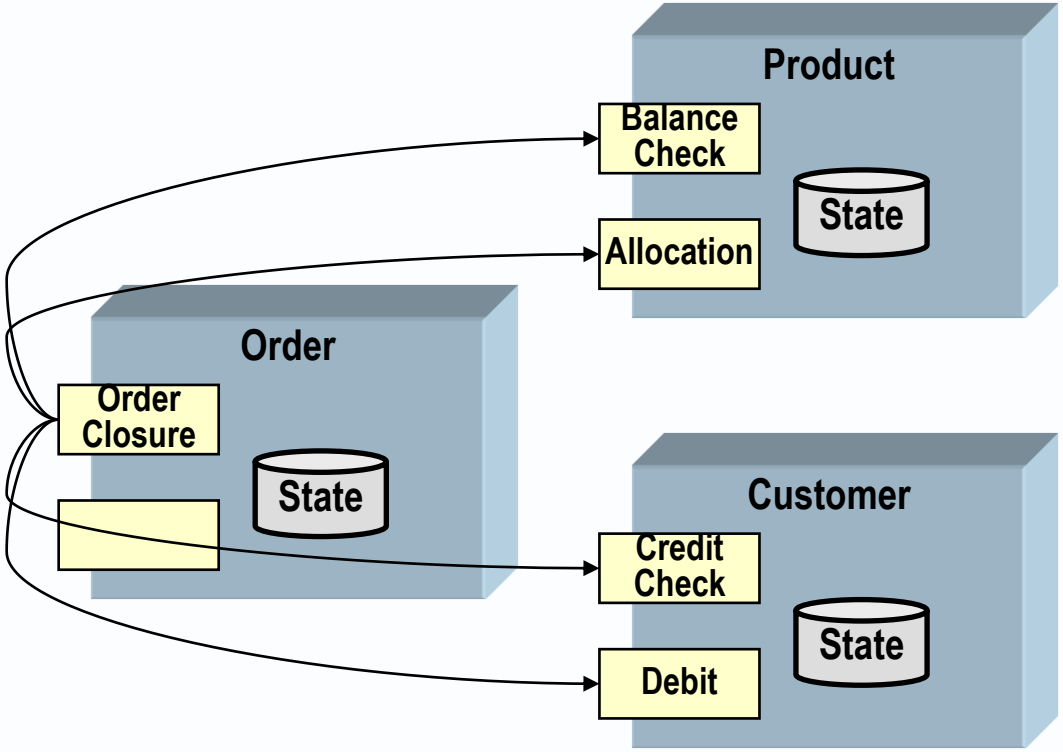
- **Client component** [a component] that requests one or more services from a server component. The request is usually called an invocation.

- **Server component** [a component] that can provide one or more services in response to requests from a client.
- Cluster analysis can be used to group services that are related (e.g.) by access to the same data) into one API or one server component.



Delegation

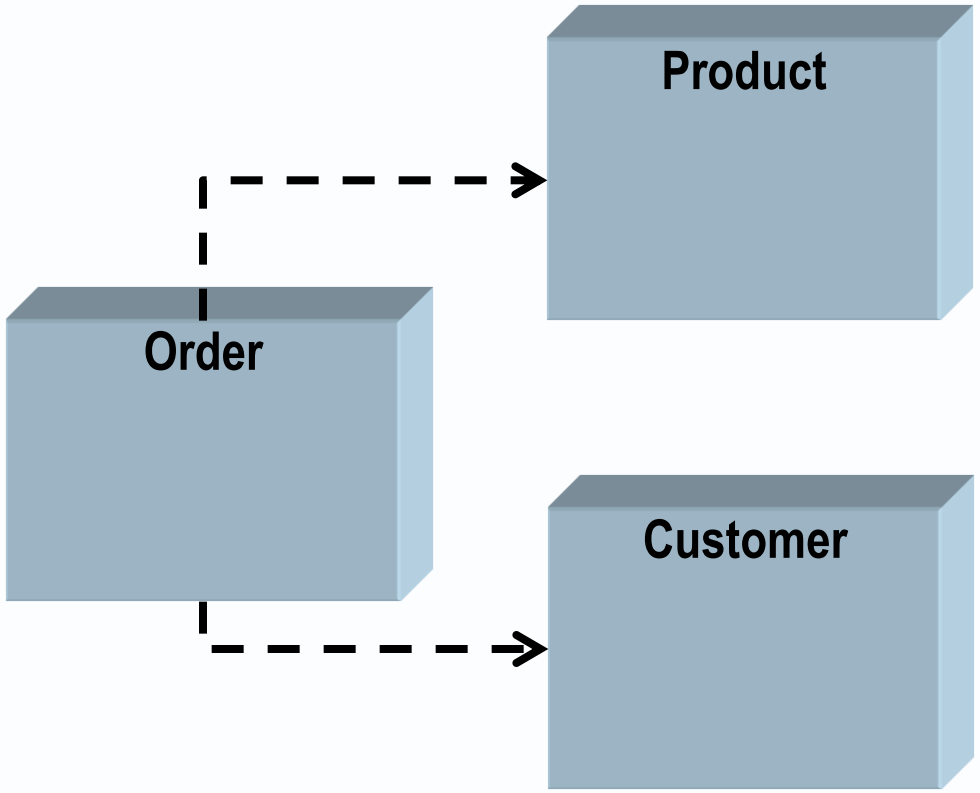
- [a process] whereby on component calls on another to do the work, invoking it by passing a message.



Dependency

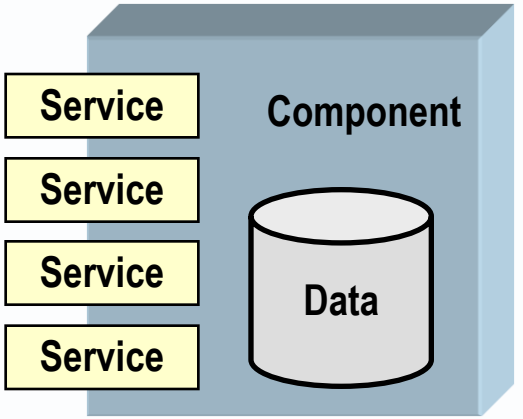
- [a mapping] a relationship between two parties in which any change to the depended-on party implies an impact analysis of the dependent party.

- Often reflects a client-server invocation, or delegation to a server.
- Often represents a run-time relationship.

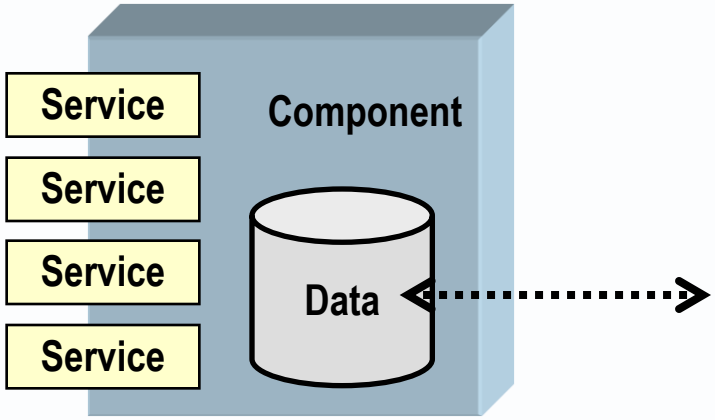


Stateful and stateless components

- **Stateful component** [a component] a component that retains data in its local memory between invocations.
- The state is lost if the component is removed.

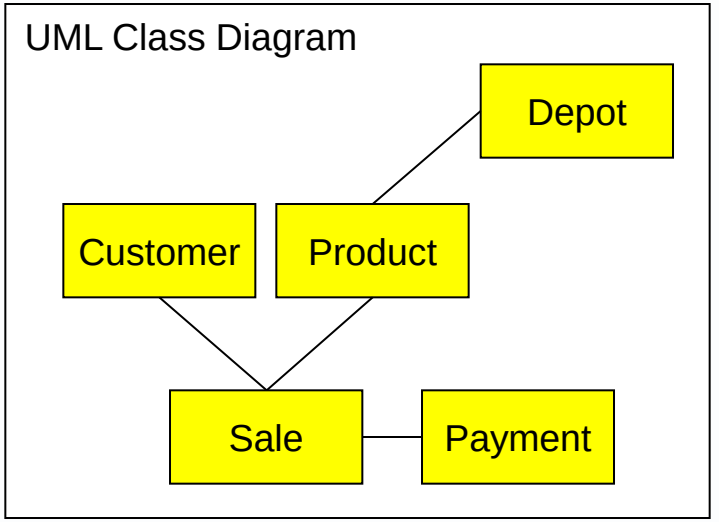


- **Stateless component** [a component] a component that does not retain data between invocations.
- However, its transient state can be copied into a persistent data store.

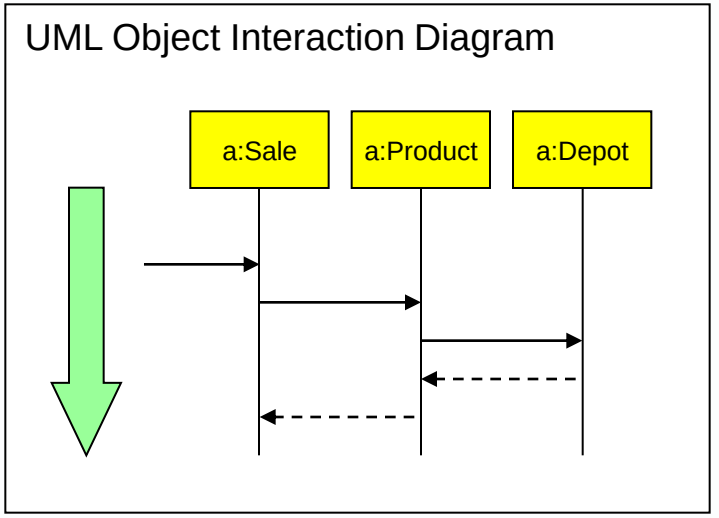


Software engineering diagrams

- **Class diagram** [an artefact] that shows the design-time structure of a software application, showing which components are related and how.



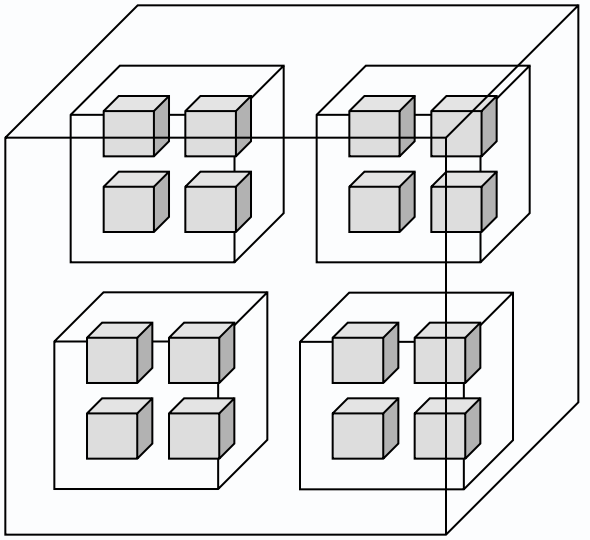
- **Sequence diagram** [an artefact] that shows how components cooperate at run-time to enable a process.
- Knowing how a design-time structure will behave at run time is critical to meeting requirements.



6.2: Modular design and integration

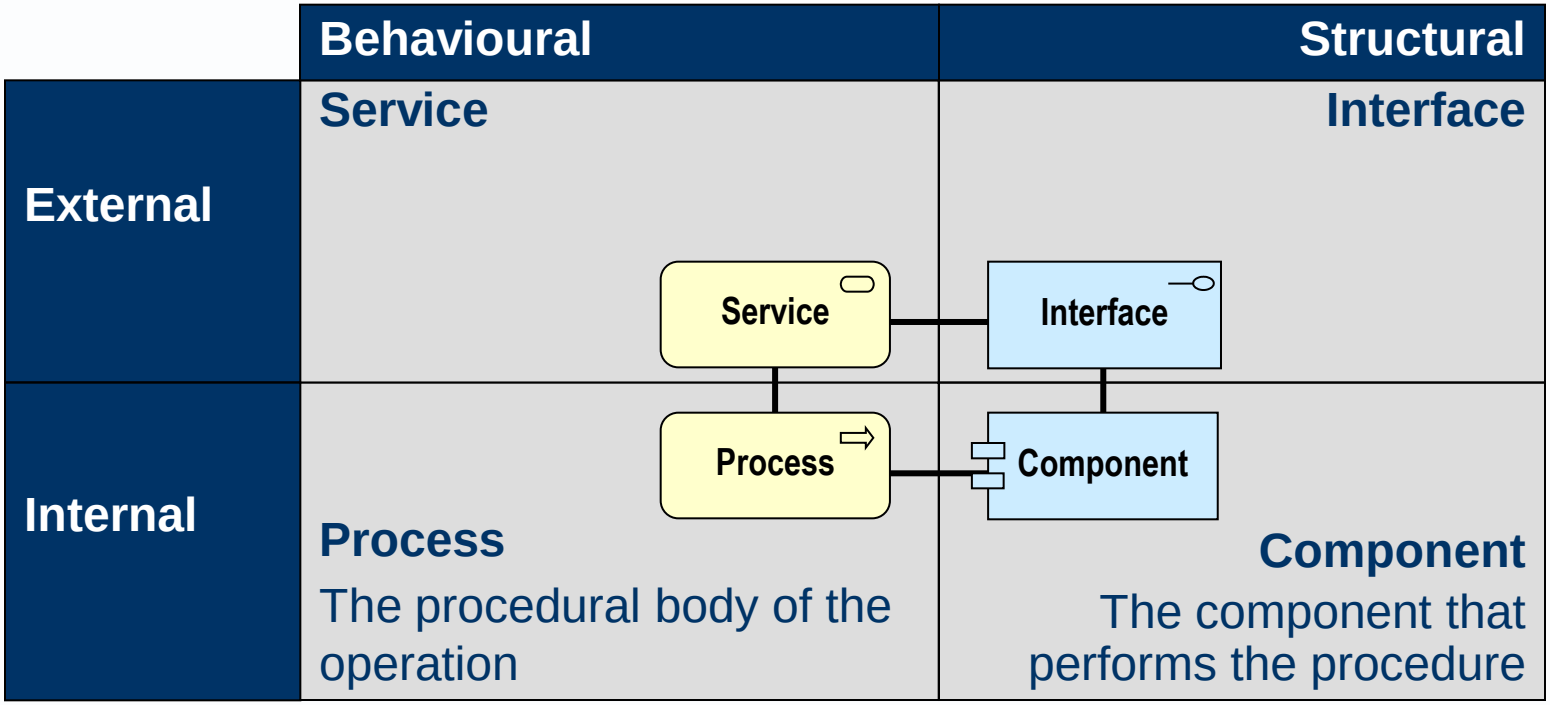
- This section reflects a history in which there has been a drift from **closely-coupled** to **loosely-coupled** software architecture.

- **Modular design** [a technique] for dividing a “monolithic” system into self-contained, encapsulated components.
- Concerns include:
- scoping a component,
- avoiding duplication between components,
- separating or distributing components, and
- integrating components.



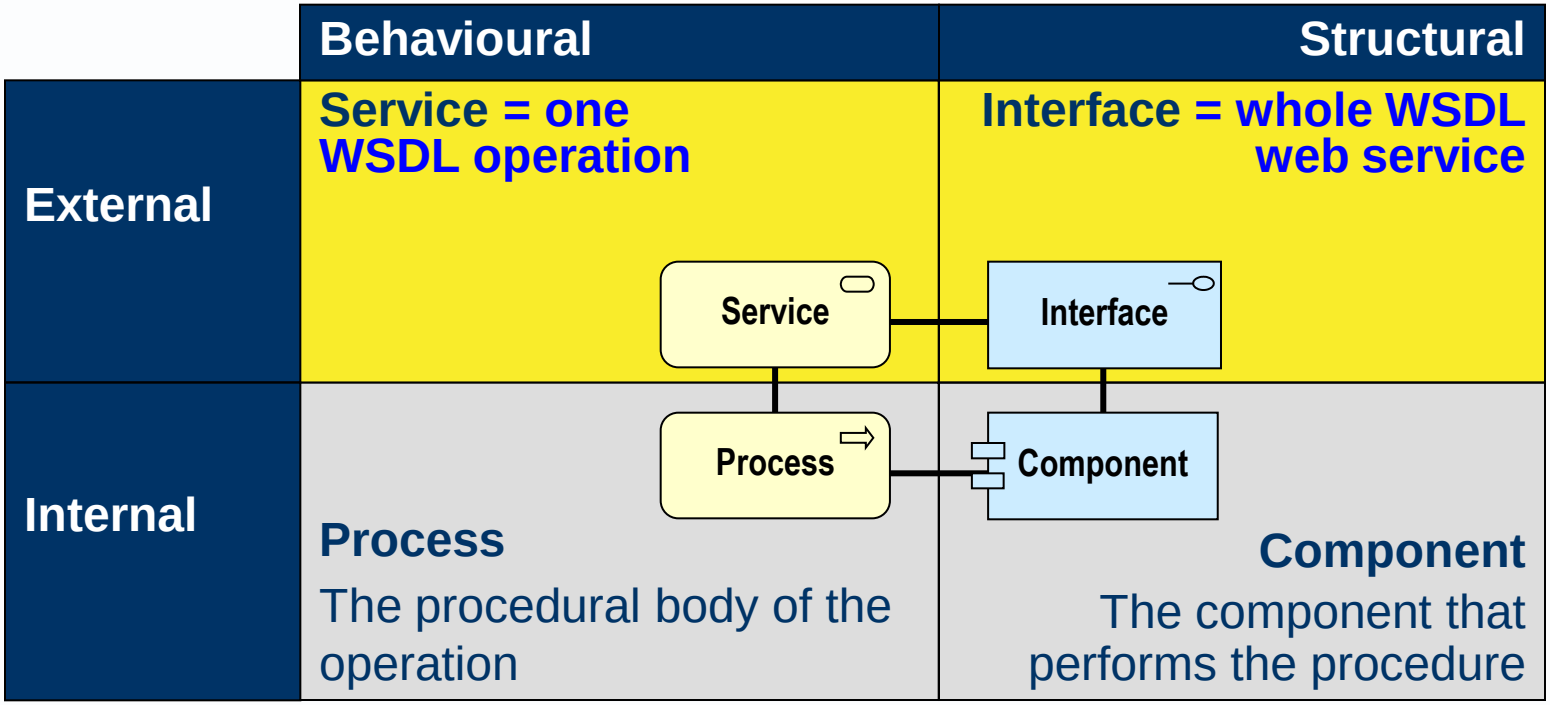
This history of software architecture led us to

- Encapsulate components behind interfaces
- Encapsulate processes behind service



Mapping web services to our generic meta model

The essence of the WSDL
is the abstract part, the list of operation



The first client-server invocations

- **Local Procedure Call** [a process] by which one component calls another component running on the same computer. It is simpler, quicker and more secure than a remote procedure call.

Local Procedure Call

- Simple
- Fast
- Available
- Secure

- **Remote Procedure Call** [a process] by which a process on one computer calls a process on another computer. It is more complex, slower and less secure than a local procedure call. The term usually implies a synchronous request-reply style of interoperation

Remote Procedure Call

- More complex
- Slower
- Less available
- Less secure

OO Programming Language

- a language in which components differ from more traditional modules in three ways.

1. It is possible to deploy **many instances** (objects) of a module type (class).

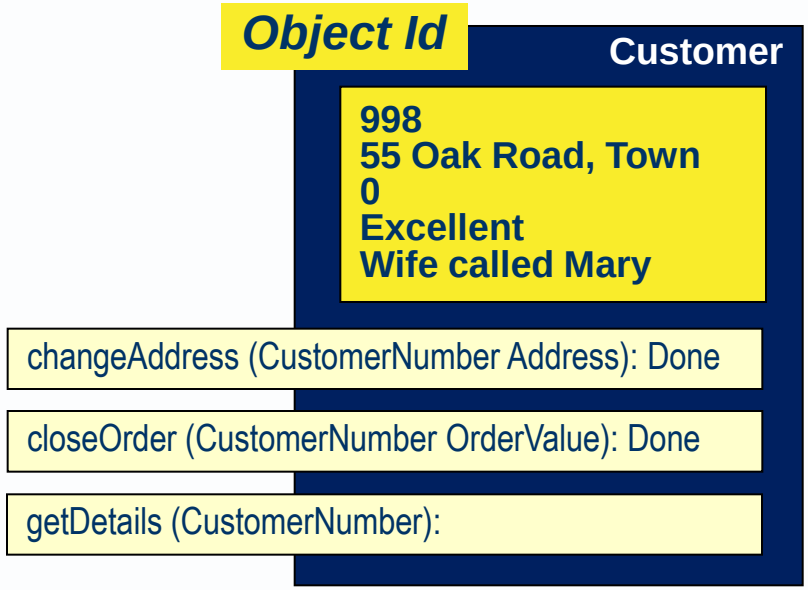
2. A client uses an **object identifier** to locate the module instance (object) it wants.

3. Objects of one class can **inherit/extend** the operations of a more generic class.

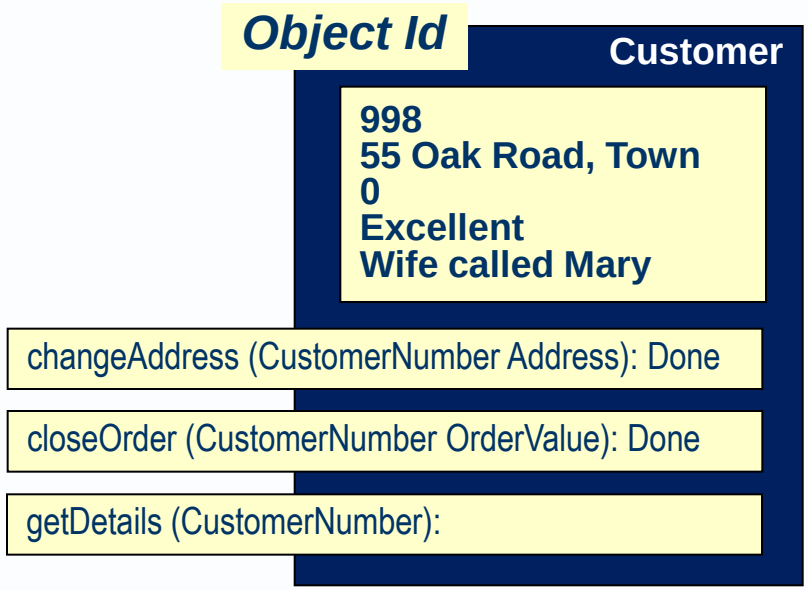
OO Design

- [a technique] a kind of modular design and integration that *initially* tended to assume the following features.
  - a component is an object, an instance of a class, identified by an object identifier
  - client/sender and server/receiver objects work in the same name space
  - an object is stateful
  - an object can reuse behaviour by inheriting it from a super type class
  - intelligent domain objects (rather than intelligent process/control objects)
  - client objects communicate with server objects by request-reply invocations
  - a server object can accept only one invocation at once (so blocks others)

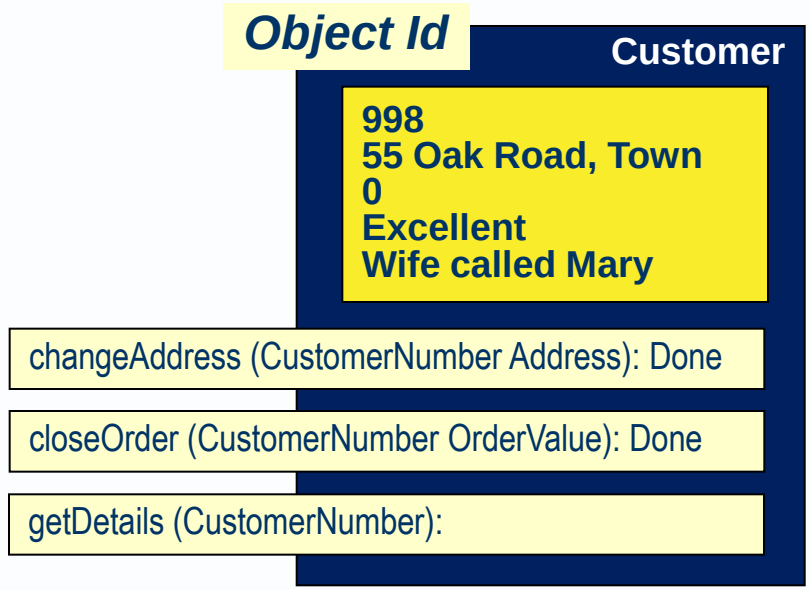
- a component is an object, an instance of a class, identified by an object identifier



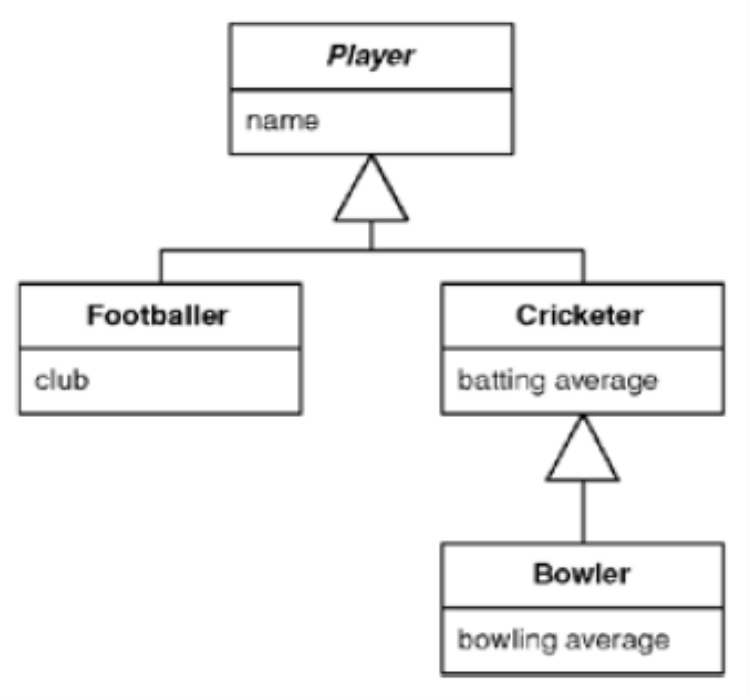
- client/sender and server/receiver objects work in the same name space

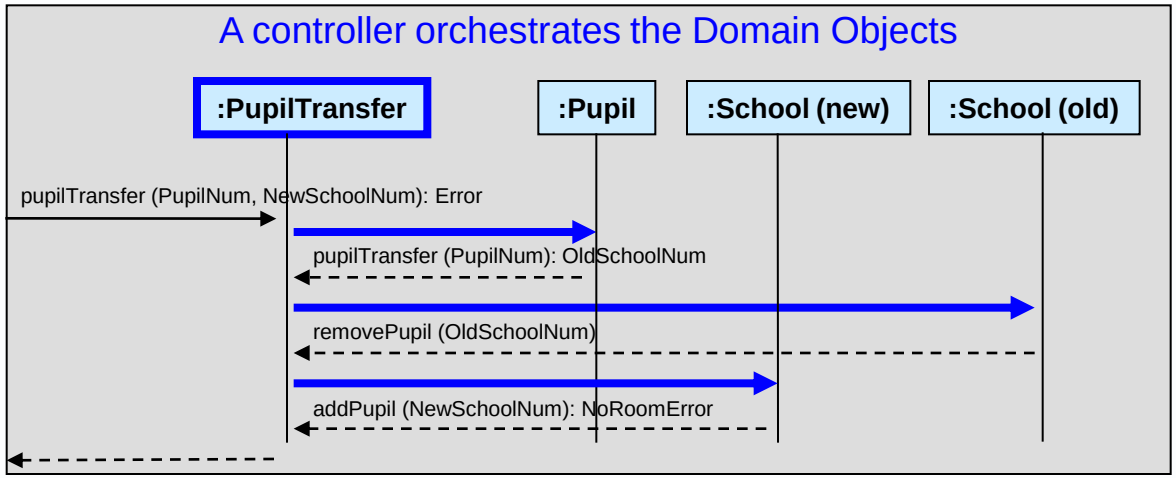


- an object is stateful

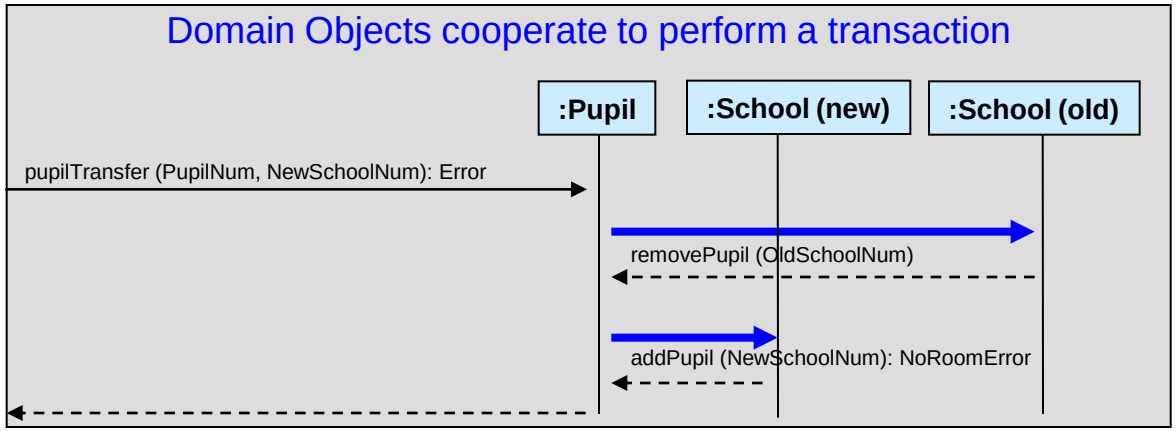


- an object can reuse behaviour by inheriting it from a super type class



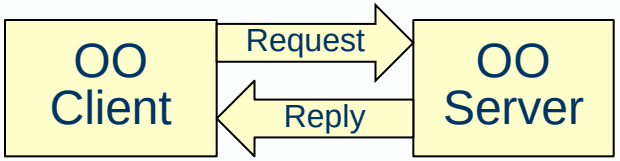


- intelligent domain objects (rather than intelligent process/control objects)



Synchronous

- 1. client objects communicate with server objects by request-reply invocations



- 2. a server object can accept only one invocation at once (so blocks others)

Garbage collection

- [a process] that deletes unused stateful objects from memory. (Multi-user business systems, handling millions of entities and events, are unlike the earliest OO programs, which needed to maintain only few stateful objects in memory.)

DO: Distributed Objects

- [a technique] a distributed modular design and integration style that employs an Object Request Broker (a kind of middleware).

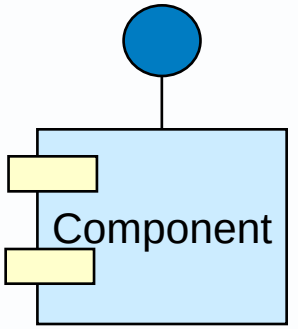
- **Object request broker (ORB):** Like RPC with extra features. It enables the objects of an OO program to be distributed. Software is coded as though all objects are on one computer. The ORB handles the distribution of objects between computers. So (in theory) the distributed system behaves like one OO program. It may add transaction management, security and other features. OMG's CORBA emerged as the standard.

DO **SOAP** **REST** **EDA**

Tightly-coupled **Loosely-coupled**

CBD: Component-Based Design

- [a technique] a modular design and integration style that involves dividing a system into encapsulated components that are much more coarse-grained than the objects in conventional OO design, perhaps in different places and technologies.



- **IDL: Interface Definition Language:** a language for defining an API that is supposed to be independent of the technology used to implement the component behind the interface.
- It enables communication between components in different languages and running on different operating systems.

E.g. Sun's ONC RPC
The Open Group's Distributed Computing Environment
IBM's System Object Model
Object Management Group's CORBA,
WSDL for Web services

SOA according to Microsoft

- **WSDL: Web Service Description Language:** [a standard] from the W3C for defining an interface, including a signature, protocol and web address for each discretely accessible operation/service. Initially, related to XML and SOAP, now usable with JSON and HTTP.

- **SOAP:** A W3C standard protocol, initially devised by Microsoft as a means to connect remote components without using an ORB.
- It allows components deployed on disparate operating systems (e.g.Windows and Linux) to communicate by sending XML messages over HTTP.

DO **SOAP** **REST** **EDA**

Tightly-coupled **Loosely-coupled**

Web Service

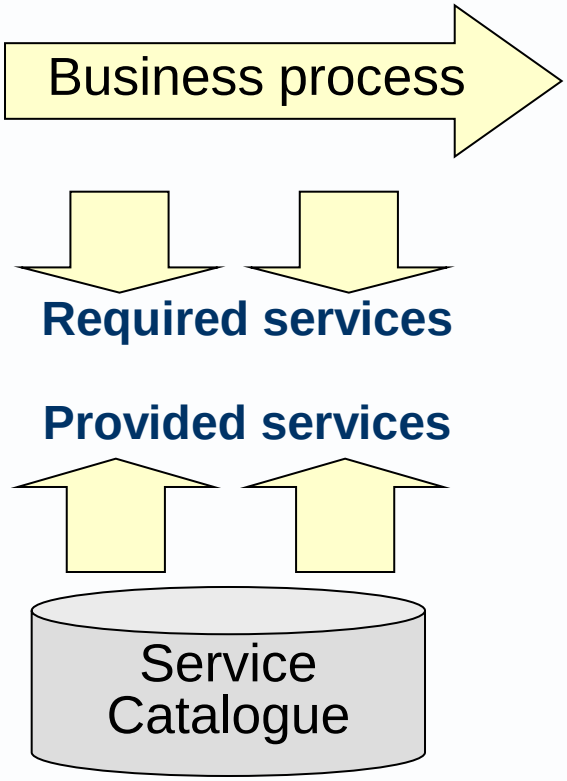
- [a component] that can be invoked over “the web” using an internet protocol and a published interface.
- It uses open standards like WSDL, XML and SOAP, but no particular standard is widely agreed as constraining what the term means.

DO **SOAP** **REST** **EDA**

Tightly-coupled **WS?** **Loosely-coupled**

SOD

- **SOA**: **Service–Oriented Architecture** [a technique]
- a modular design and integration style that is a
- more loosely-coupled than Distributed Objects and
- facilitates the reuse of remotely accessible services.
- It is often associated with the use of Web Services, but does not have to be.



DO — **SOAP** — **REST** — **EDA**

Tightly-coupled — **SOA?** — **Loosely-coupled**

REST: Representational State Transfer

- [a technique] a modular design and integration style devised by Roy Fielding as a means to connect remote components using standard internet protocols.
- It decouples distributed components so that client/sender components need minimal information about server/receiver components.
- A RESTful client invokes a remotely accessible service using a domain name and an operation type available in an internet protocol, usually HTTP.
- A REST-compliant server is identified by a domain name and offers only one service in response to each operation type in an internet protocol, usually HTTP.

DO **SOAP** **REST** **EDA**

Tightly-coupled **Loosely-coupled**

OData:

- [an IDL] an evolution of REST that supports clients wanting to invoke operations on entities in a remote data store.
- Client apps that speak OData can easily connect to data server apps that provide CRUD operations on a logical data model.

DO SOAP REST EDA

Tightly-coupled Loosely-coupled

EDA: Event-Driven Architecture

- [a technique] a modular design and integration style decouples the senders of news or update events from the receivers.
- Any component can receive or read any event/message published or posted by any other component.
- It uses either the publish and subscribe functions of a middleware technology, or else a shared data space.

DO **SOAP** **REST** **EDA**

Tightly-coupled **Loosely-coupled**

Component interoperation style

Avancier

Tightly-coupled Loosely-coupled

LPC RPC **DO** SOAP **REST** **EDA**

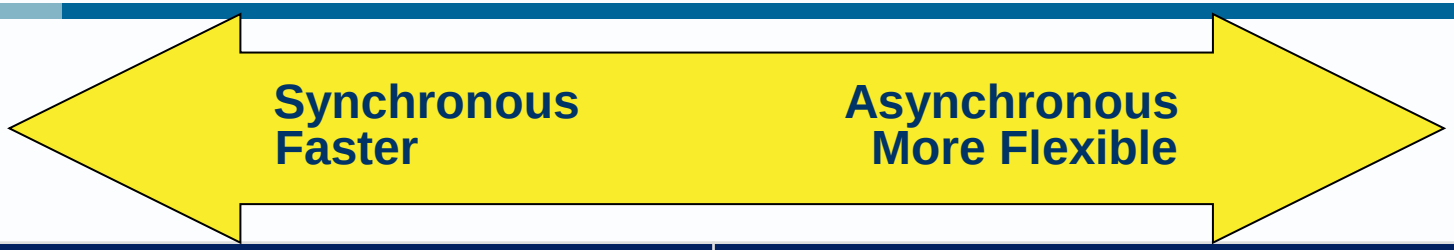
Interoperation styles

SOA

6.3: Decoupling

- **Loose coupling factor** [a property] considered in deciding whether components should be tightly or loosely coupled.

Loose coupling mechanisms



Earliest OO design presumptions	Recent SOA design presumptions
Object identifiers	Internet domain names or URIs
One name space	Several name spaces
Stateful objects	Stateless modules
Reuse by Inheritance	Reuse by Delegation
Intelligent domain objects	Intelligent process controllers
Request-reply invocations	Message passing
Blocking servers	Non-blocking servers

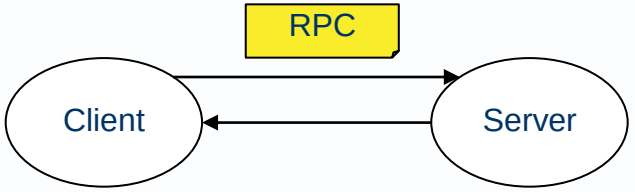
COBOL modules and Java objects

CORBA

Web Services

Coupling by time: synchronous

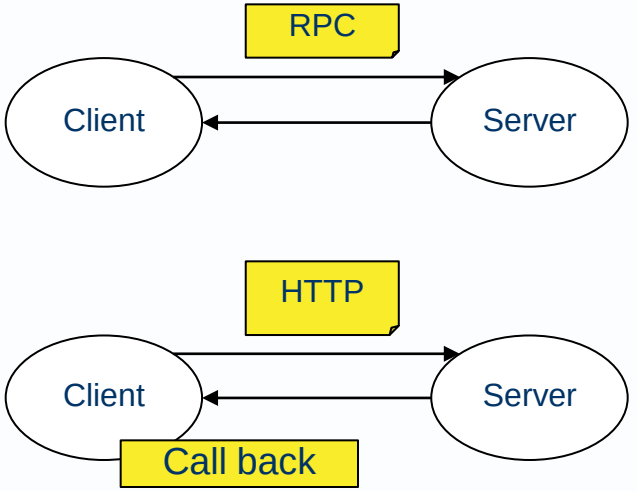
- 1: **A request-reply style**; a client must wait for a server to reply before continuing.
- 2: A **blocking style**; a server serves one client at a time, turning away any other client who attempts to request a service.
- The caller and responder hold a channel open, blocking others from using it.



- Aside:
- 1: is the usual invocation from one COBOL module to another or one Java object to another.
- 2: is the usual invocation style in CORBA.

Decoupling by time: asynchronous

- 1: A so-called **fire-and-forget style** in which a client does not wait for a server to reply, and can carry on to do other things.
- 2: A **non-blocking style** in which a server can accept requests from several clients before responding to the first.
- Typically, the responder has a queue of incoming messages and releases the channel after a message is received.



- Aside:
- 1. is the usual style in email conversations.
- 2. is the usual style of Web Services.

General coupling factors

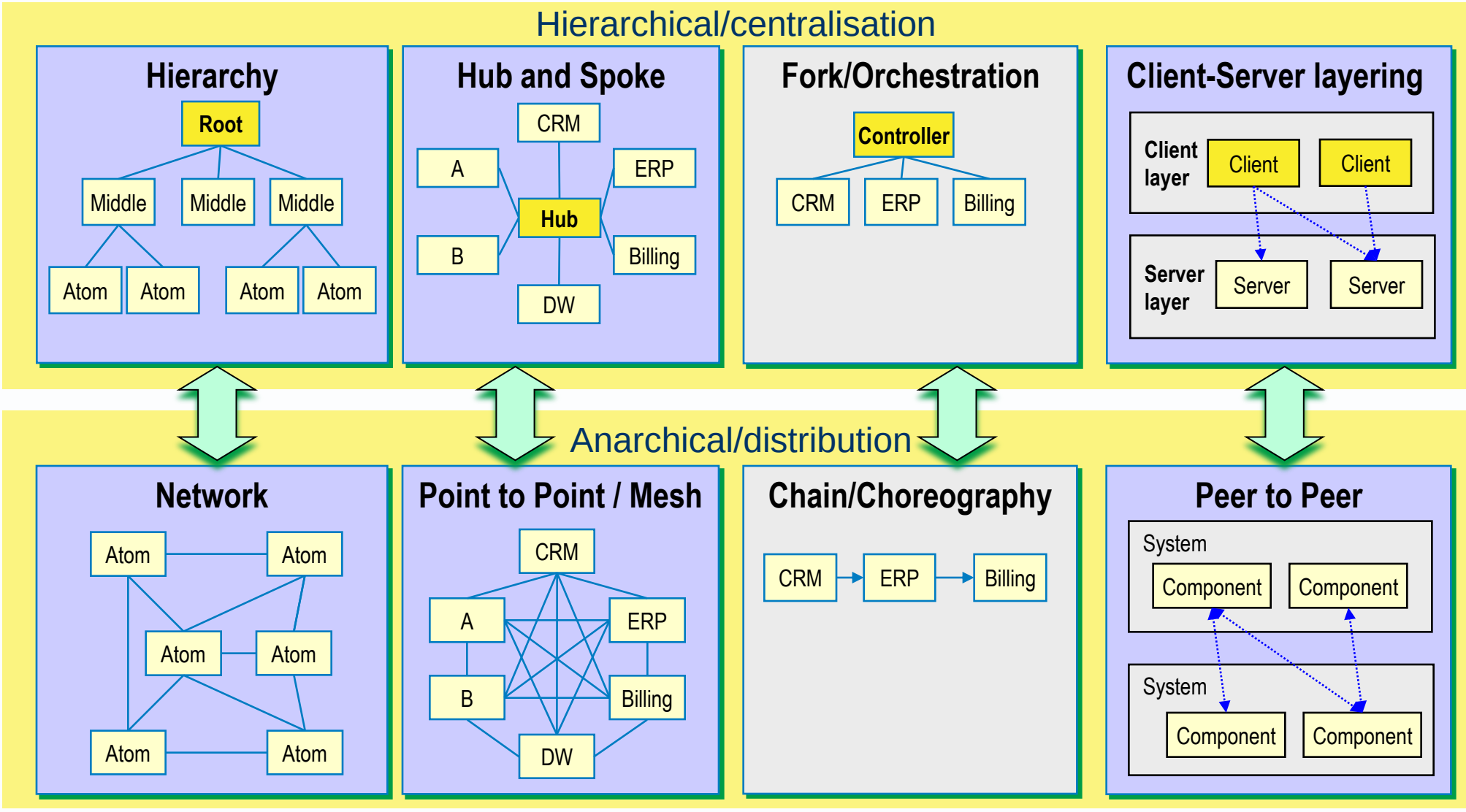
Faster simpler

More flexible

Avancier

Factor	Tight coupling	Decoupling techniques
Naming	Clients use object identifiers One name space	Clients use domain names Multiple name spaces behind interfaces
Paradigm	Stateful objects/modules Reuse by OO inheritance Intelligent domain objects	Stateless objects/modules Reuse by delegation Intelligent process controllers
Time	Synchronous request-reply Blocking servers	Asynchronous messaging Non-blocking servers
Location	Remember remote addresses	Use brokers/directories/facades
Data types	Complex data types	Simple data types
Version	Version dependency	Design to avoid version dependence Apply the open-closed principle
Protocol	Protocol dependency	Design for multiple protocols
Integrity constraints	ACID transactions	BASE: compensating transactions and eventual consistency

6.4: Component structures and patterns



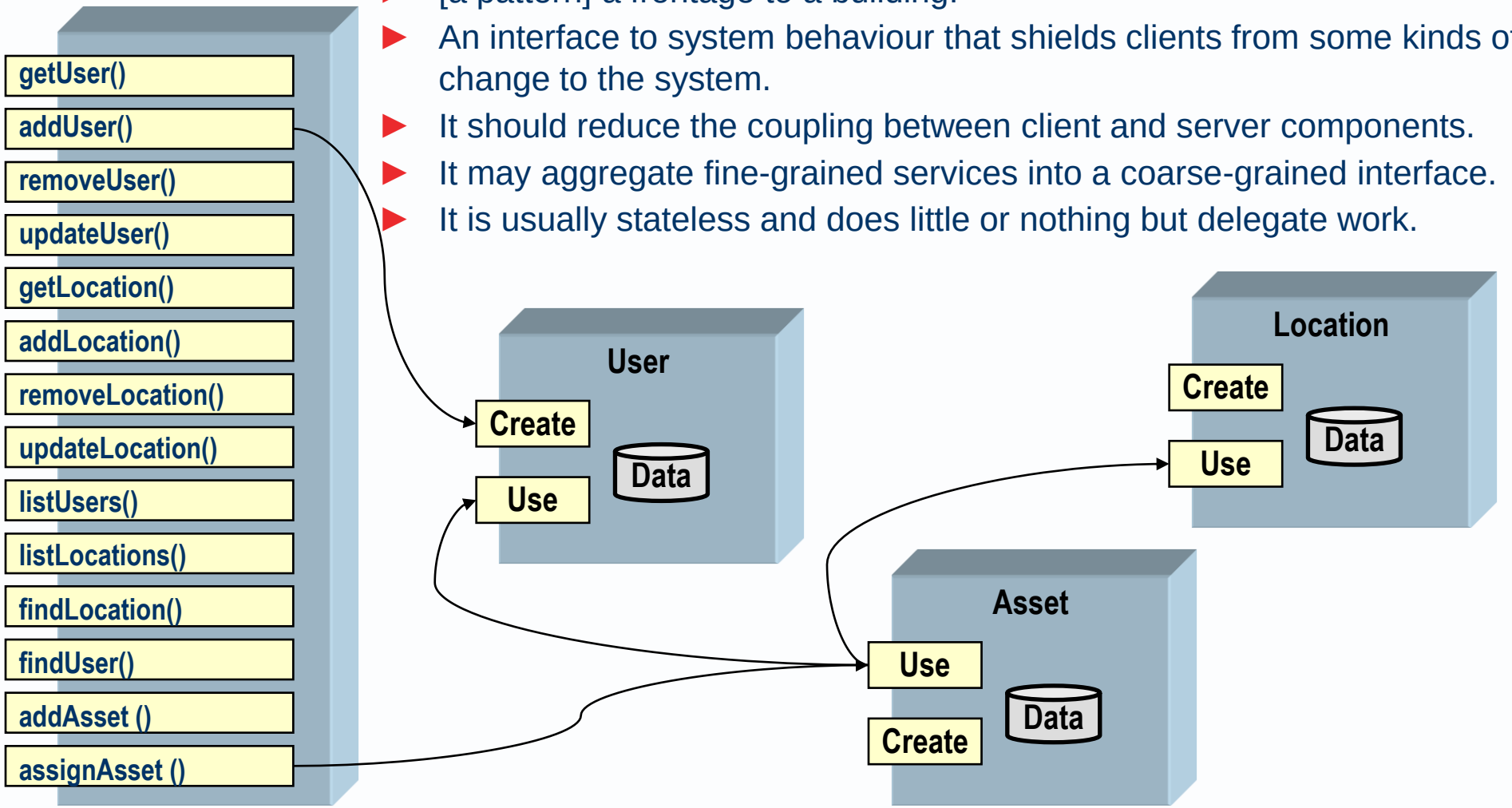
23 design patterns in the GoF's famous book

"Design Patterns: Elements of Reusable Object-Oriented Software"

Name	Description or purpose
Façade	Consistent interface coarse-grained services that encapsulate a suibsystem
Adapter	A wrapper that converts a provided service into a required service. Facilitates the reuse of existing technologies.
Proxy	A surrogate for a distributed component. Used in distribution of code between different name spaces.
Singleton	A component (or class) with only one instance (or object).
Observer	A component that monitors the state of another component.
Composite	Enables a class to process operations on every level of a hierarchical structure, including composite and leaf nodes
Template method	Offers several variations of an algorithm.
Strategy	Adds a façade to a template method.
Manager	Often used to manage a set of objects.
Factory method	Create the right server object for the client - hiding how the server object is initialized.
Abstract object factory	Create the right factory object for the client - hiding which factory class is used.

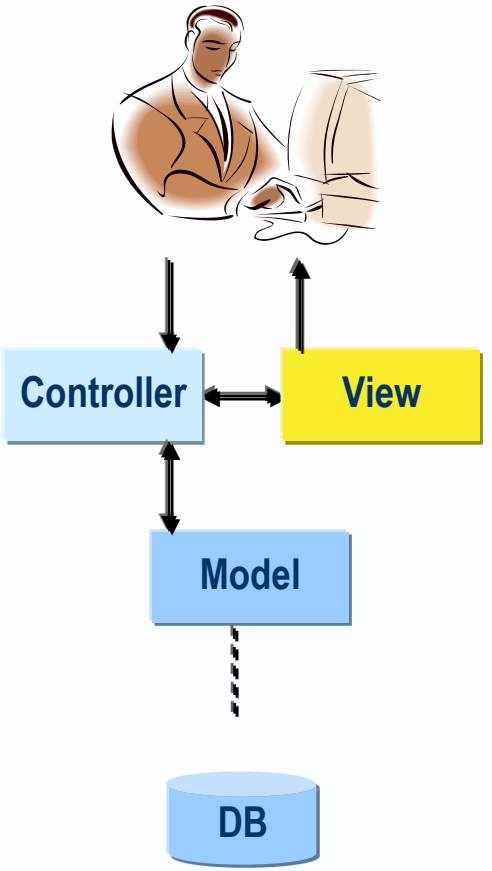
Façade

- [a pattern] a frontage to a building.
- An interface to system behaviour that shields clients from some kinds of change to the system.
- It should reduce the coupling between client and server components.
- It may aggregate fine-grained services into a coarse-grained interface.
- It is usually stateless and does little or nothing but delegate work.



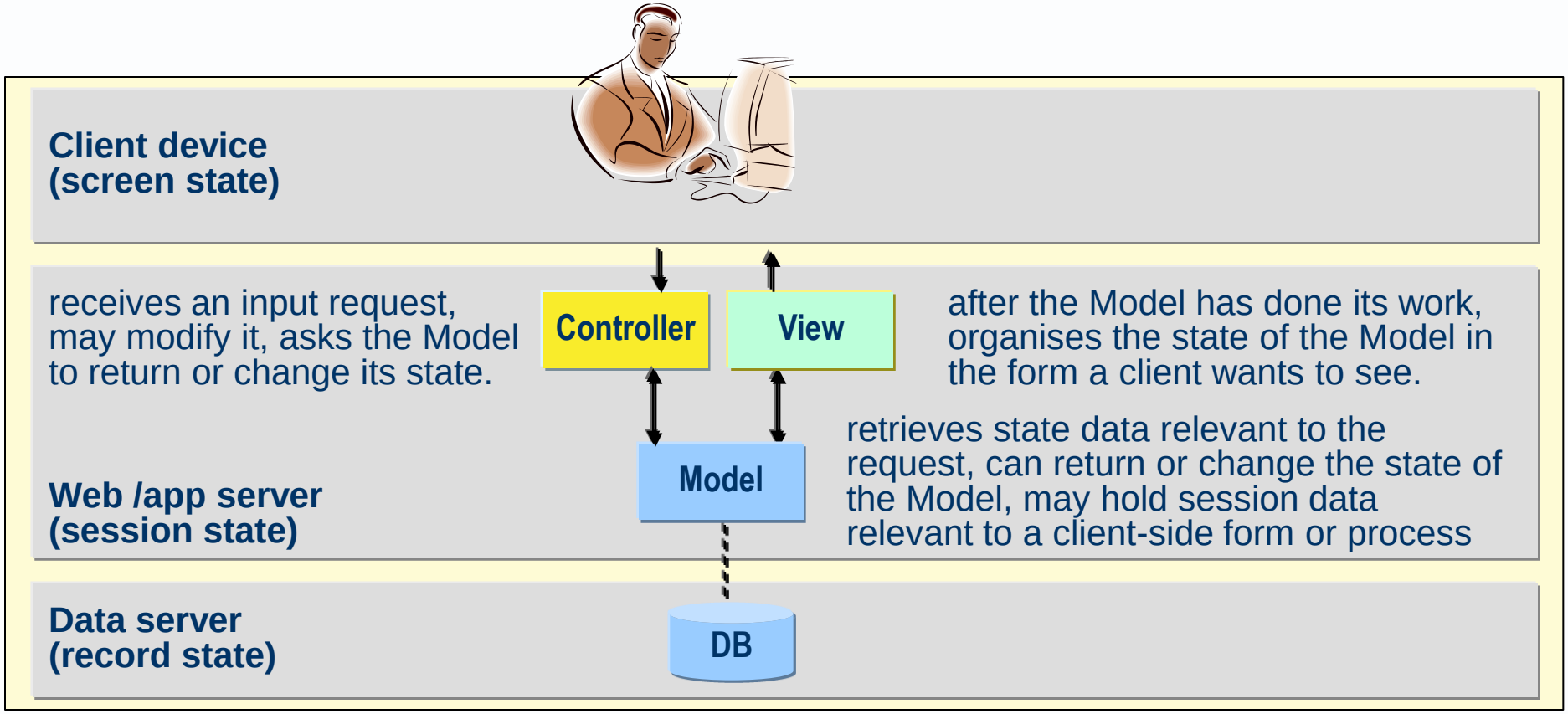
Model View Controller

- [a pattern] that separates
- the processing of an input message (controller),
- the display of a user interface (view) and
- processing of persistent data (model).



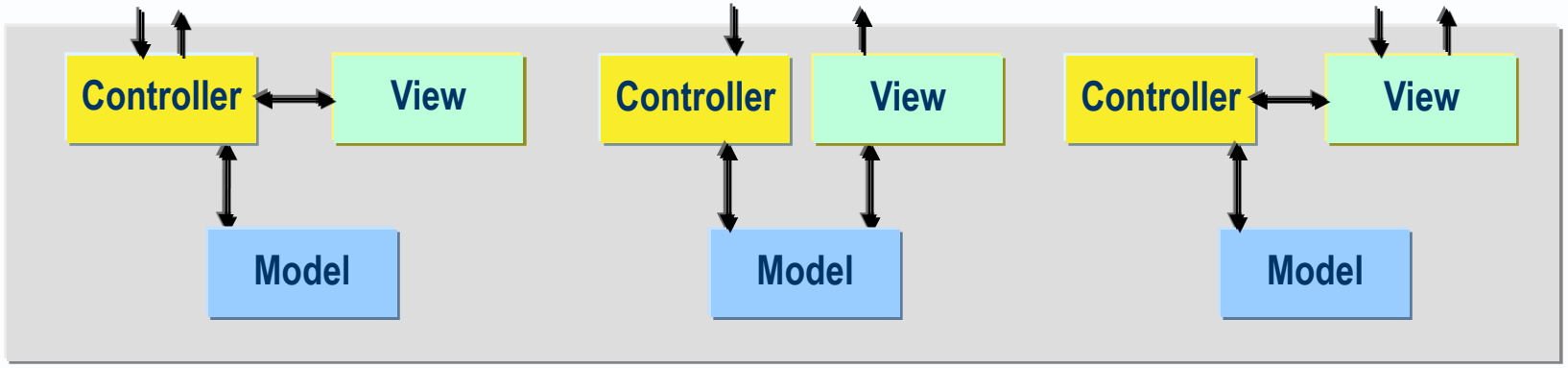
How the MVC pattern may be used in enterprise applications

- Separates modules that handle client-side data structures from modules that handle server-side data structures



Variations of the MVC pattern

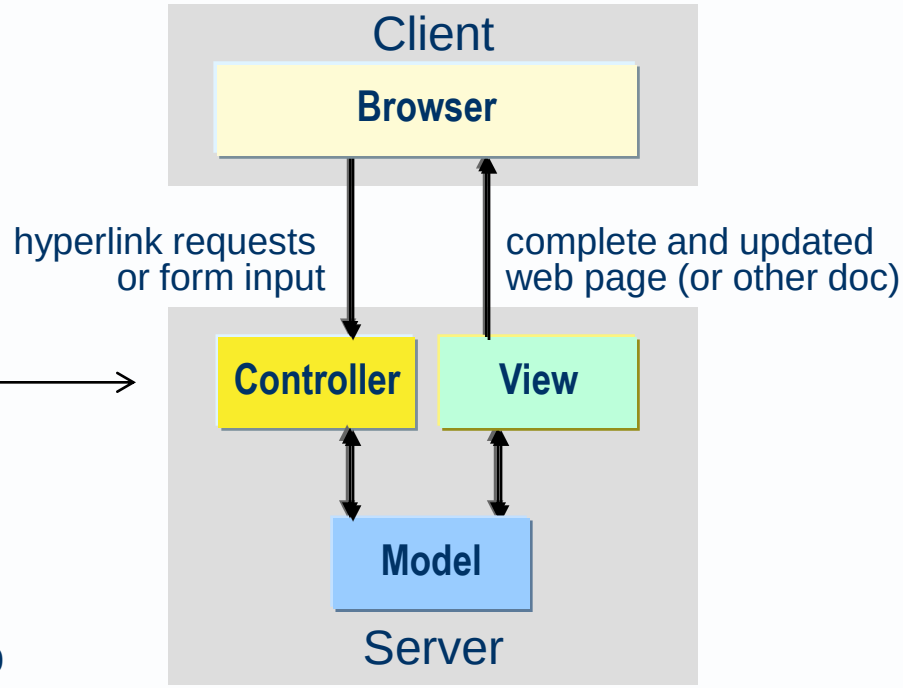
- Three variations



- Other MVC variants include
  - hierarchical model–view–controller (HMVC),
  - model–view–adapter (MVA)
  - model–view–presenter (MVP),
  - model–view–viewmodel (MVVM)

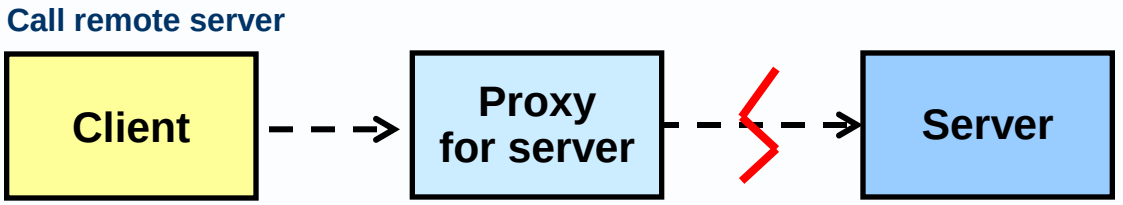
MVC in web frameworks (after Wikipedia)

- Web frameworks divide MVC differently between client and server tiers.
- Early web frameworks mostly put MVC components **on the server**.
  - Ruby on Rails,
  - Django,
  - ASP.NET MVC and
  - Express
- Other frameworks allow MVC components to execute partly **on the client**
  - AngularJS,
  - EmberJS,
  - JavaScriptMVC and B
  - ackbone (also see Ajax).



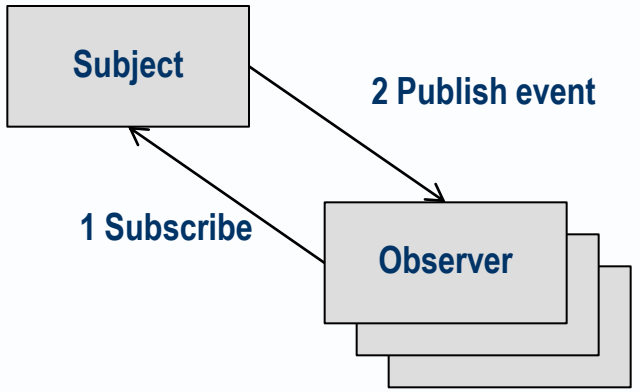
Proxy

- [a pattern] a surrogate for a distributed component.
- It is used in distribution of code between different name spaces, as in “Distributed Objects”



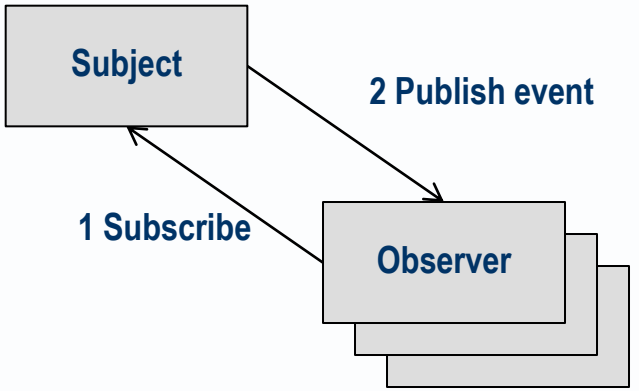
Observer

- [a pattern] a component that monitors the state of another component.
- A primitive kind of “Event-Driven Architecture”.

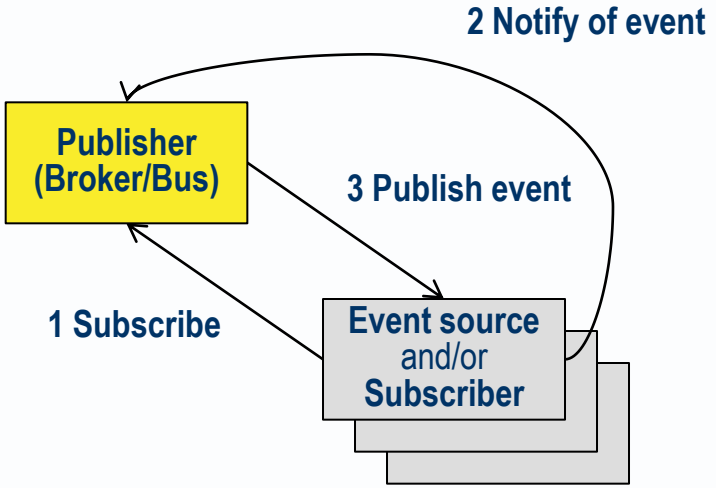


Moving notification into a separate publisher

- **Observer**
  - A subject
    - notifies observers of changes to its state
  - Observers
    - register with the subject to be notified of changes.
    - unregister when no longer interested



- **Event-Driven Architecture (EDA)**
  - Inserts a publisher (broker) between Subjects and Observers, and so decouples



Singleton

- [a pattern] a component (or class) with only one instance (or object).

- We only need one copy of:
  - Global variables – e.g. current date and time
  - A commonly required table – e.g. exchange rates
  - A façade – e.g. a stateless controller
- So, a single-object class can hold the data or do the job

In Java, static fields (aka class variables)
exist independently of any instances of the class
and one copy is shared among all instances

Utility class = a stateless singleton?

Commonly required mathematical functions

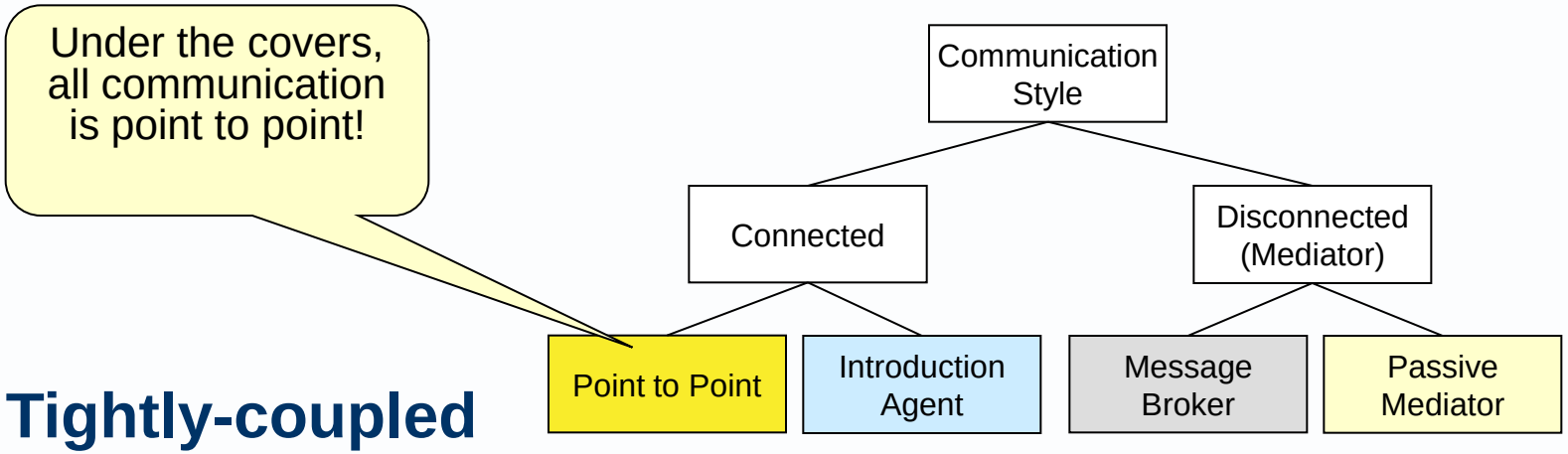
Length, weight and temperature conversions

Circumference, area and volume calculations

4 communication styles

Avancier

Component communication style	The manner in which components interact (as client and server, or sender and receiver), which can be direct or via intermediaries. There are three broad categories: •Point to point •Introduction agent •Mediator (active or passive)



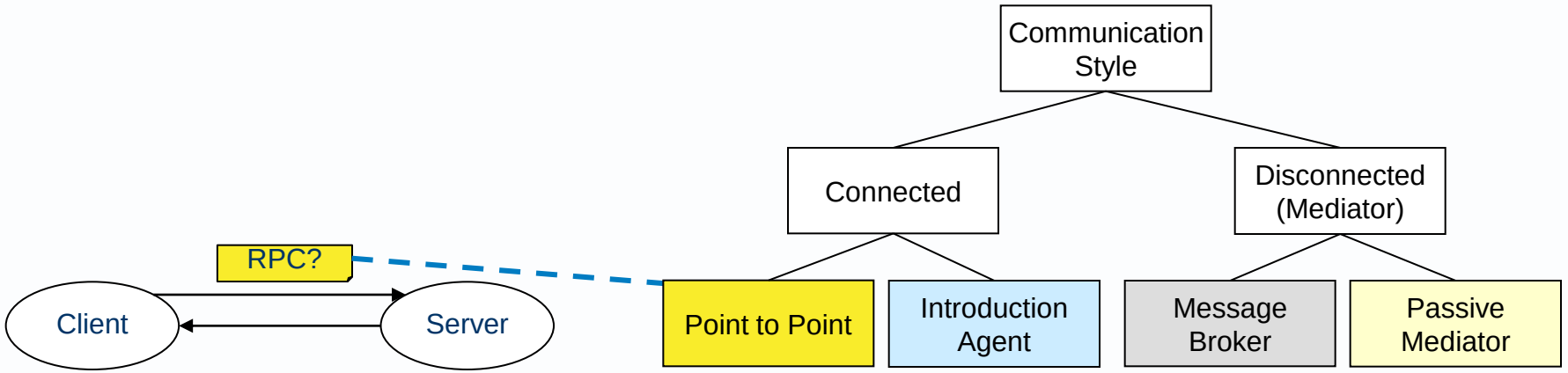
Tightly-coupled

Loosely-coupled

Point-to-point communication

Avancier

Point-to-point communication	A sender (or client) is coupled to a receiver (or server). More precisely defined by two features: 1) The sender is responsible for knowing or determining both the location of the receiver, and a protocol and data format the receiver understands. 2) A message is sent by one sender and received by one receiver. Strength: simple to implement. Weaknesses: potential duplication of data transformation and routing code, high configuration cost of receiver address changes.

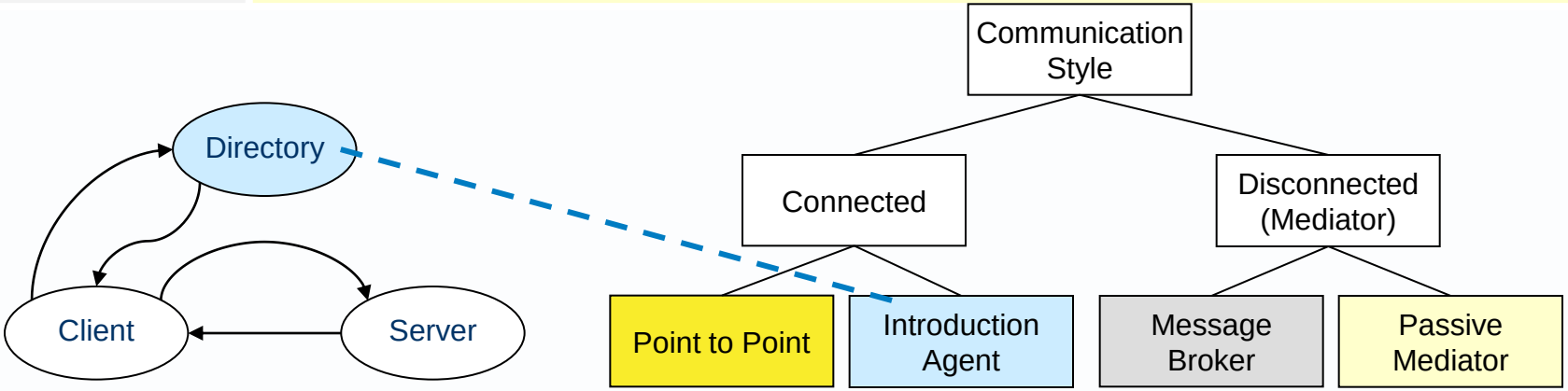


Introduction agent (direct broker)

Avancier

Introduction agent (direct broker) communication

A direct broker helps parties to communicate.
It decouples clients and servers, at least to the extent that the two parties can work in different places.
It hides some complexities of the communication transport.
The broker must register parties willing to communicate (end point registration).
The broker can then establish initial connection when a client requests a service from a server.
Subsequently, the parties communicate point-to-point via client-side and server-side proxies, which is faster than via a mediator broker.

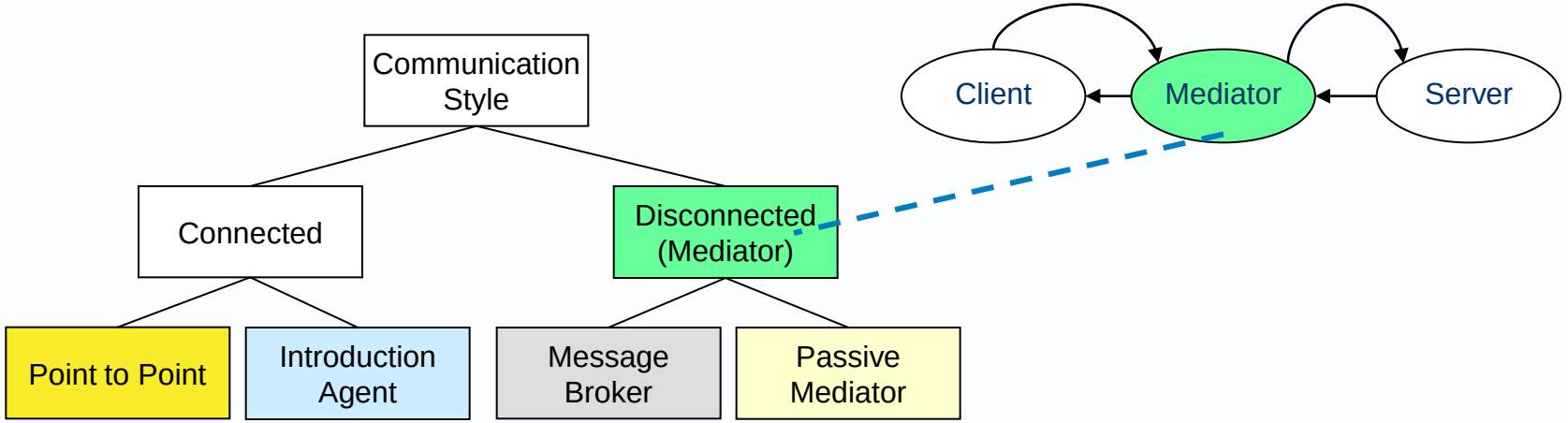


Mediator (indirect broker)

Mediator communication

An indirect broker helps parties to communicate. It:

- decouples clients and servers by sitting between them.
- means the parties can work at different places and times (asynchronously).
- can shield one party from some effects of some changes to the other.
- does for components what email infrastructure does for people, that is, enable them to communicate asynchronously via messages - rather than talk directly over an end-to-end network connection kept open for that conversation.



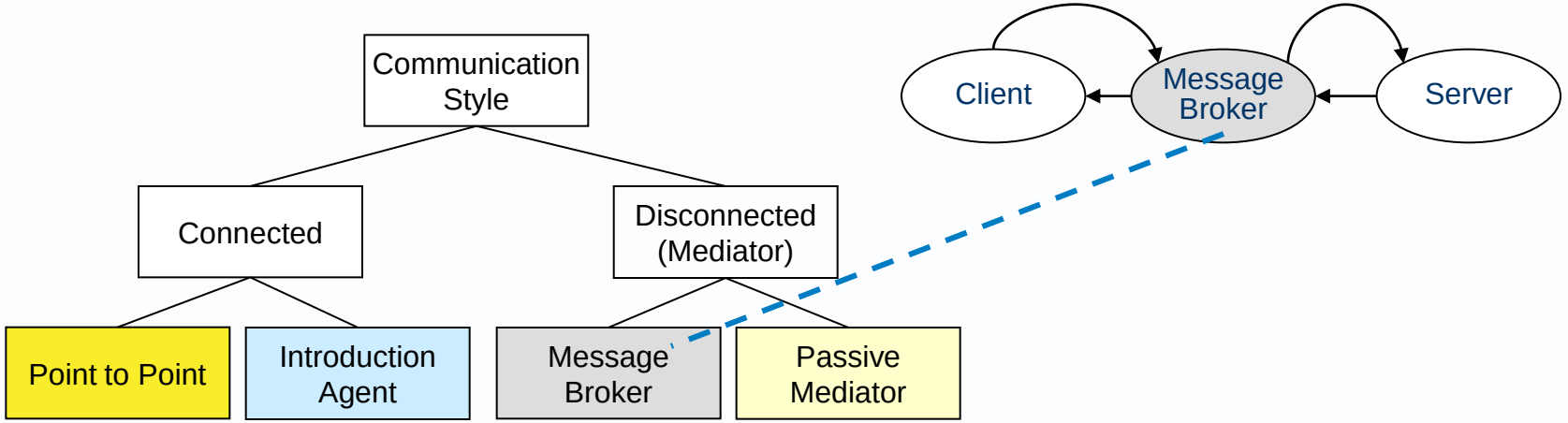
Active Mediator

Avancier

Message broker (indirect broker) distribution

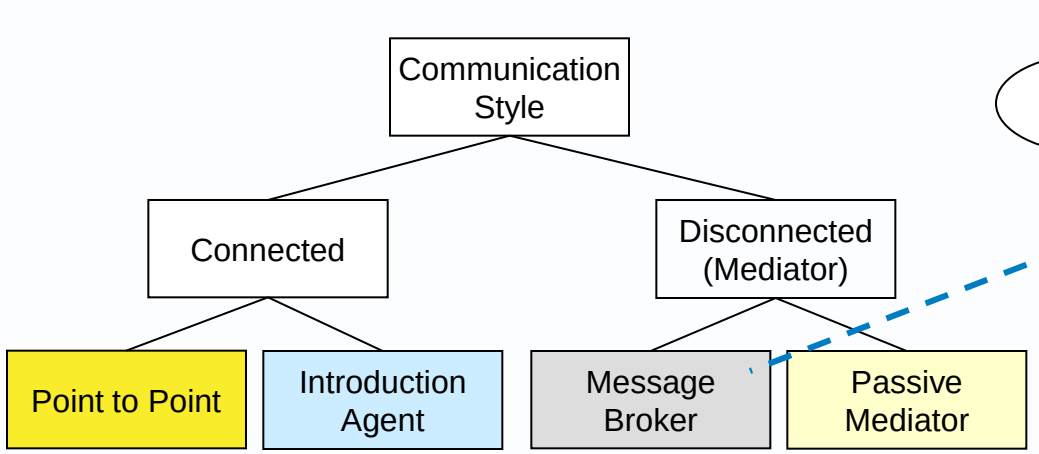
A message broker is a mediator that forwards communications. It:

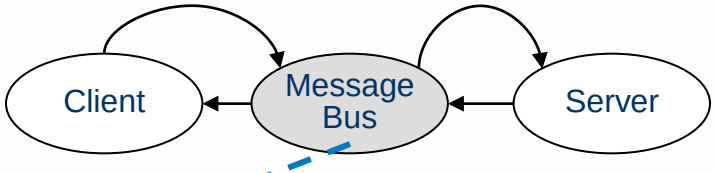
- provides a shared infrastructure for sending messages to recipients.
- hides some complexities of the communication transport.
- offers common command messages.
- enables end point registration: the broker registers parties willing to communicate.
- enables routing: the broker locates parties and sends them messages
- enables transformation: the broker converts data formats.



Mediator-based distribution

Message Router Distribution	The message broker sends a message where the sender directs. The client communicates the logical name of the server to the broker. The broker looks up the server that is registered under the logical name and passes the communication to the server.
Schema-based Distribution	The message broker uses a common data format to reduce the cost of adding and removing communicating parties. It provides a common data model, a set of agreed-upon message schemas.



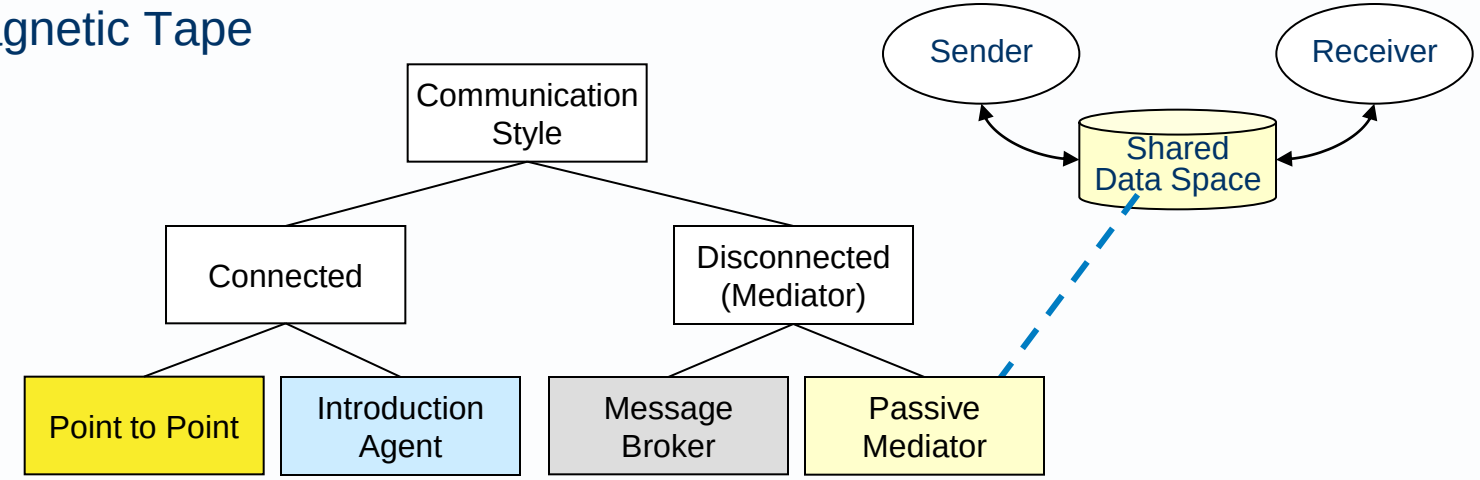


- Message bus = message broker + schema-based distribution

Passive Mediator

Passive Mediator	Any shared memory space can act as a mediator, since communicating parties can post messages to the shared memory and read messages from it – as in the blackboard pattern.

- "Shared memory" could be
  - Memory / scratch pad / bulletin boar
  - Message queue
  - Database
  - Magnetic Tape



Communication styles – summary overview