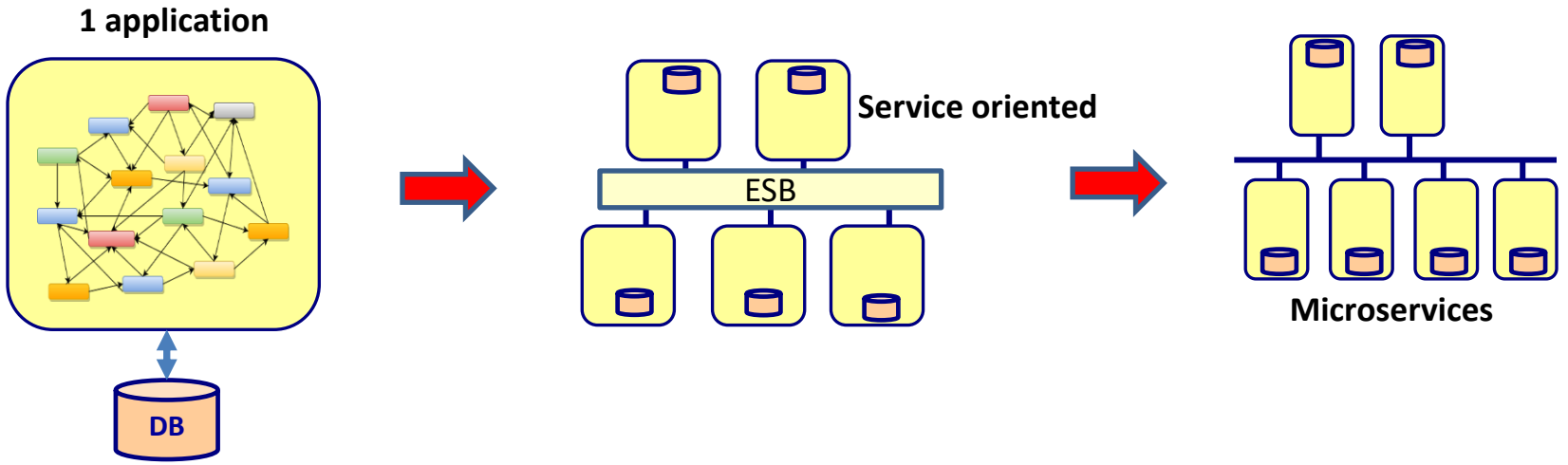
SOFTWARE ARCHITECTURE COURSE OVERVIEW

CS 590 Software Architecture

- Architecture styles
- Architecture patterns
- Architecture principles and best practices



Course agenda

Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
October 27 **Lesson 1** Software architecture introduction	October 28 **Lesson 2** Domain Driven Design	October 29 **Lesson 3** Databases	October 30 **Lesson 4** Component based design	October 31 **Lesson 5** SOA & integration patterns	November 1 **Lesson 6** Microservices 1	November 2
November 3 **Lesson 7** Microservices 2	November 4 **Review**	November 5 **Midterm Exam**	November 6 **Lesson 8** Microservices 3	November 7 **Lesson 9** Microservices 4	November 8 **Lesson 10** Microservices 5	November 9
November 10 **Lesson 11** Kafka	November 11 **Lesson 12** Kafka + Stream based architecture	November 12 **Lesson 13** Finding the right architecture	November 13 **Review**	November 14 **Final Exam**	November 15 **Project**	November 16
November 17 **Project**	November 18 **Project**	November 19 **Project**	November 20 **Project Presentations**			

LESSON 1
SOFTWARE ARCHITECTURE INTRODUCTION

Why architecture?

More complexity asks for:

- More abstraction and decomposition
- More principles and guidelines
- More communication
- More proces
- More powerful tooling

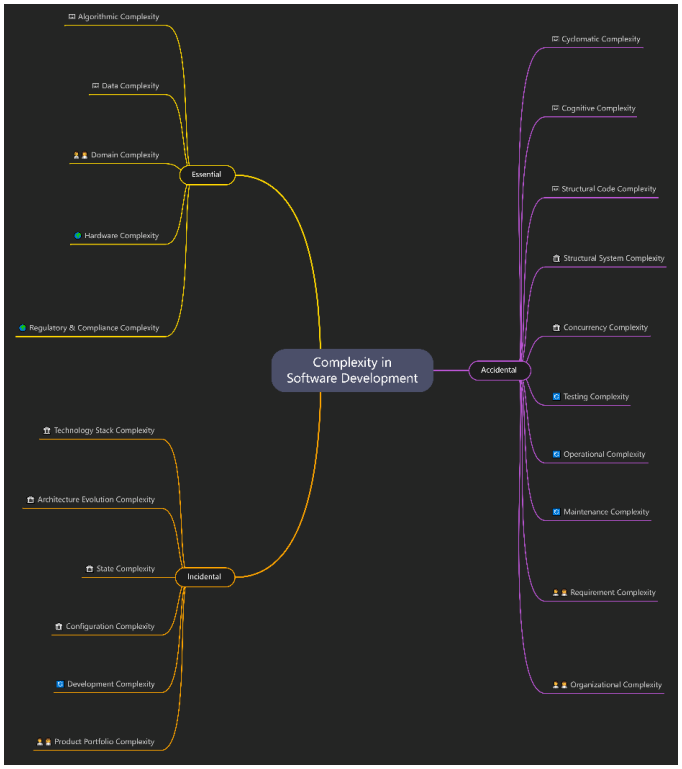
More complexity asks for more architecture

Software complexity

- Essential complexity
  - Inherent to the nature of the problem itself or its environment
- Incidental complexity
  - Accumulated over time as software, project or organization evolves
- Accidental complexity
  - Introduced by human decisions

Software complexity

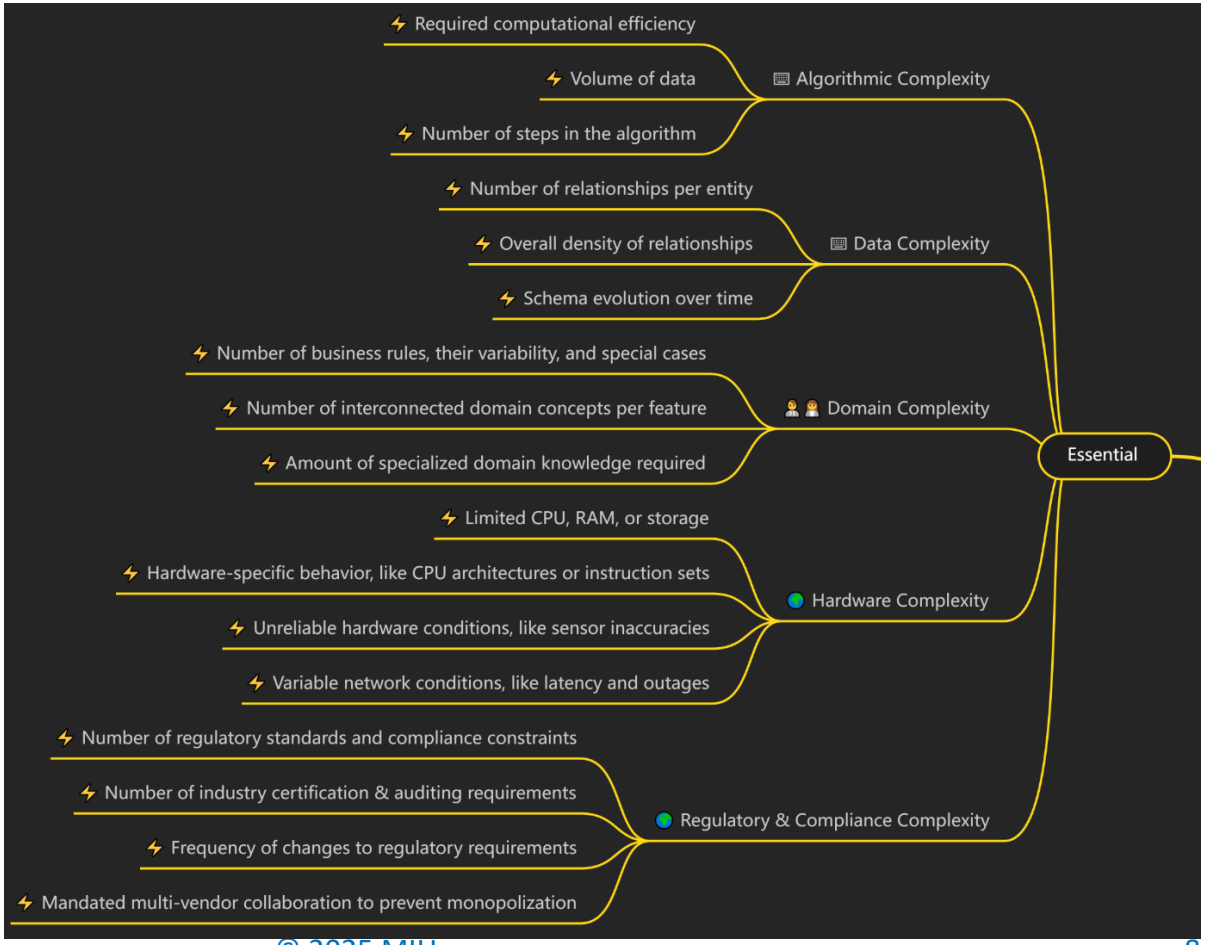
- Essential
- Incidental
- Accidental



- https://github.com/plainionist/AboutCleanCode/tree/main/Complexity

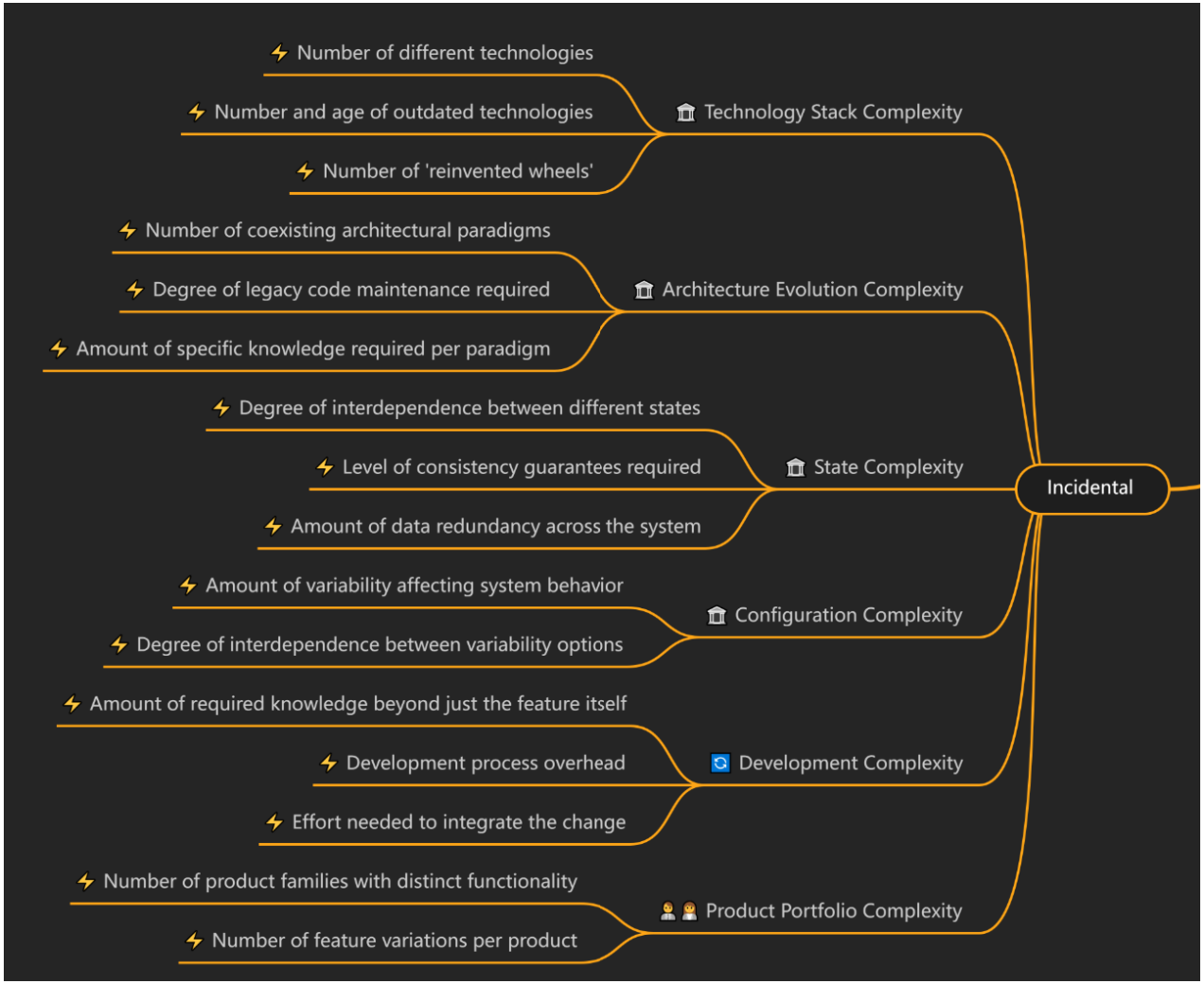
Essential complexity

- Inherent to the nature of the problem itself or its environment



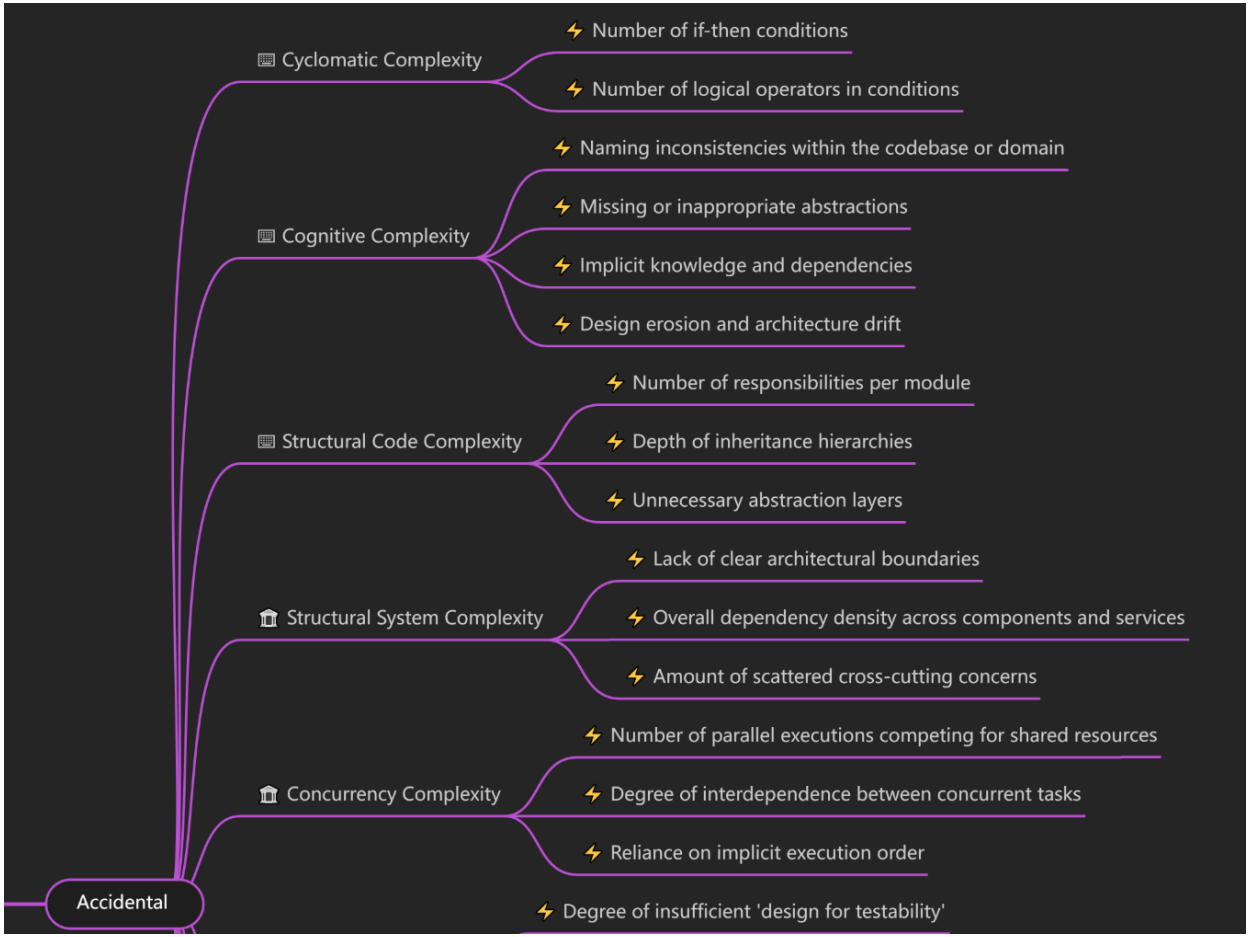
Incidental complexity

- Introduced by human decisions



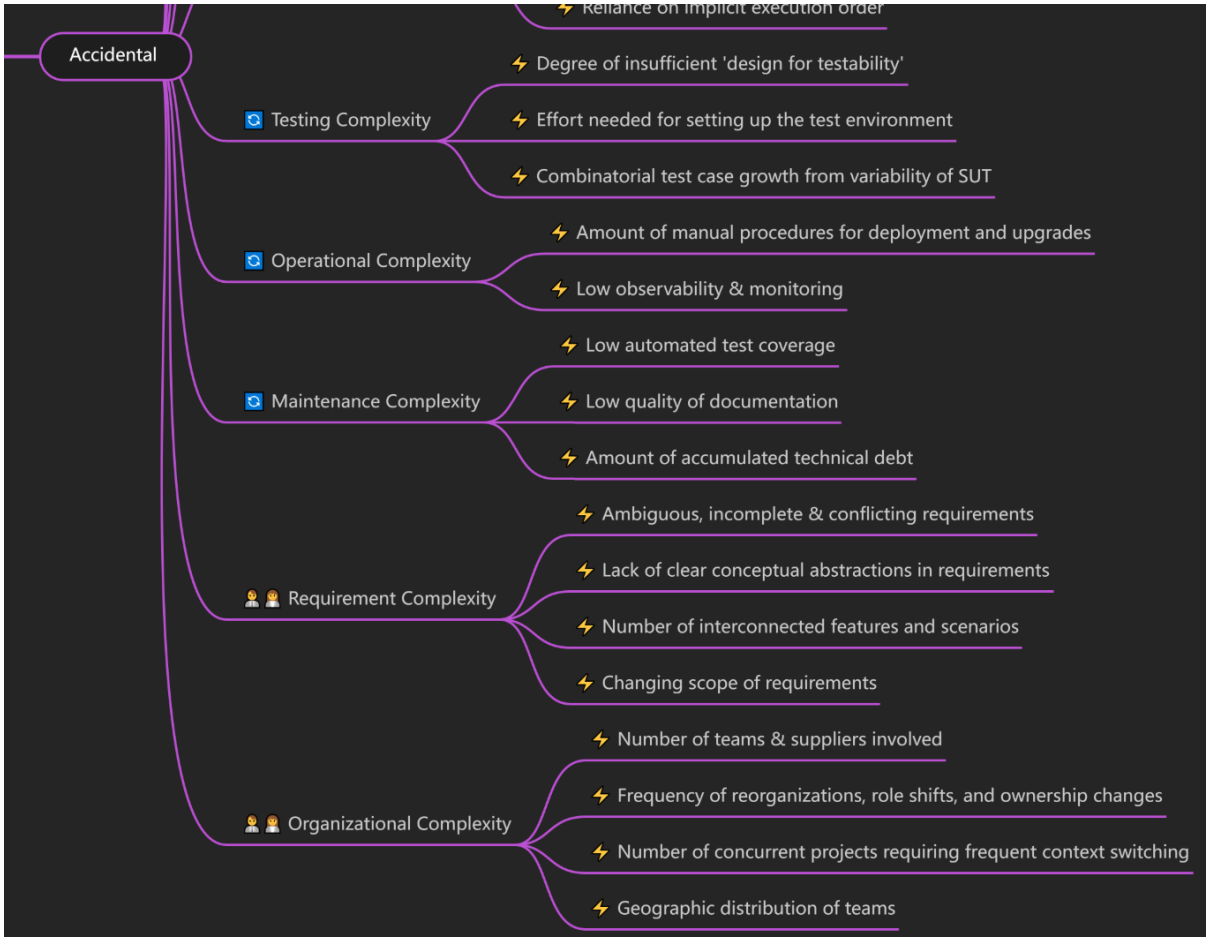
Accidental complexity

- Accumulated over time as software, project or organization evolves



Accidental complexity

- Accumulated over time as software, project or organization evolves



Software complexity

- Essential complexity
  - Inherent to the nature of the problem itself or its environment
  - This complexity cannot be reduced
- Incidental complexity
  - Accumulated over time as software, project or organization evolves
  - Can be reduced with good software engineering principles
- Accidental complexity
  - Introduced by human decisions
  - Can be reduced with good software engineering principles

Why architecture?

- Winchester “mistery” house
- 38 years of construction work– 147 builders 0 architects
- 160 rooms– 40 bedrooms, 6 litchens, 2 basements, 950 doors
- No architecture description
- 65 doors that don’t go anywhere, 13 stairs that don’t go anywhere, 24 skylights where you cannot see the sky

What is software architecture?

- The fundamental concepts or properties of a system in its environment embodied in its elements, relationships, and in the principles of its design and evolution

  - ANSI/IEEE std 1471-2000

What is software architecture?

The things that are difficult to change

A blueprint

Technical leadership

Standards and guidelines

Building blocks

Technical direction

Systems, subsystems, interactions and interfaces

Satisfying non-functional requirements/quality attributes

The big picture

The skeleton/backbone of the product

The system as a whole

Structure (components and interactions)

The things that are expensive to change

What is software architecture?

The important stuff

Whatever that might be

Defining the architecture

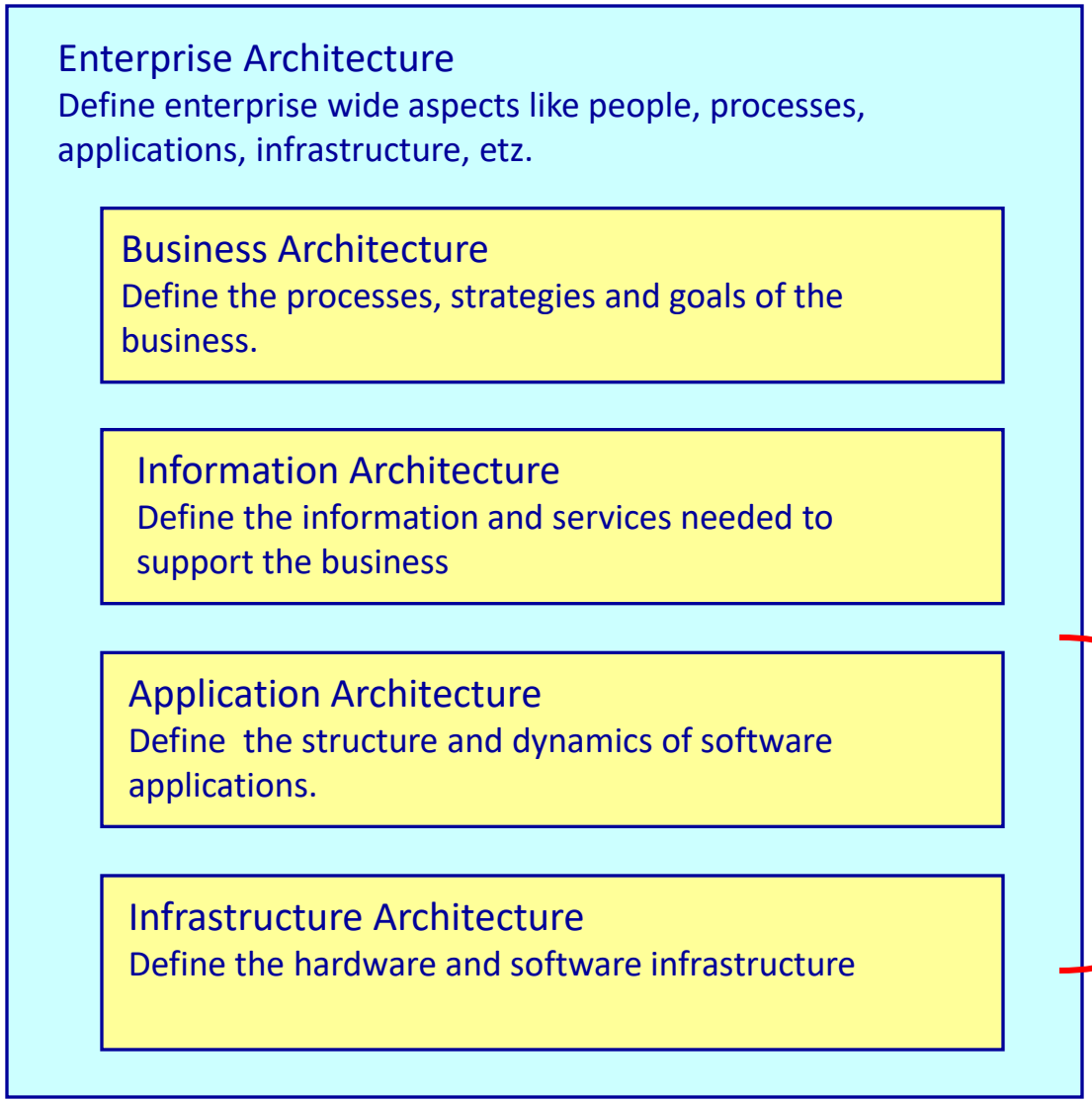
- Define components
- Define component interfaces
- Define platform and language(s)
- Define architectuur styles
- Define architectuur patterns
- Define layers and packages
- Define presentation architecture
- Define persistency architecture
- Define security architecture
- Define transaction architecture
- Define distribution architecture
- Define integration architecture
- Define the deployment architecture
- Define the clustering architecture
- Define the hardware
- Define tools to use

- Decide on solutions for
  - Logging
  - Error management
    - Error detection
    - Error reporting
    - Fault tolerance
  - Event management
  - File handling
  - Printing
  - Reporting
  - Resource management
  - Internationalization
  - Licence management
  - Debugging
  - …

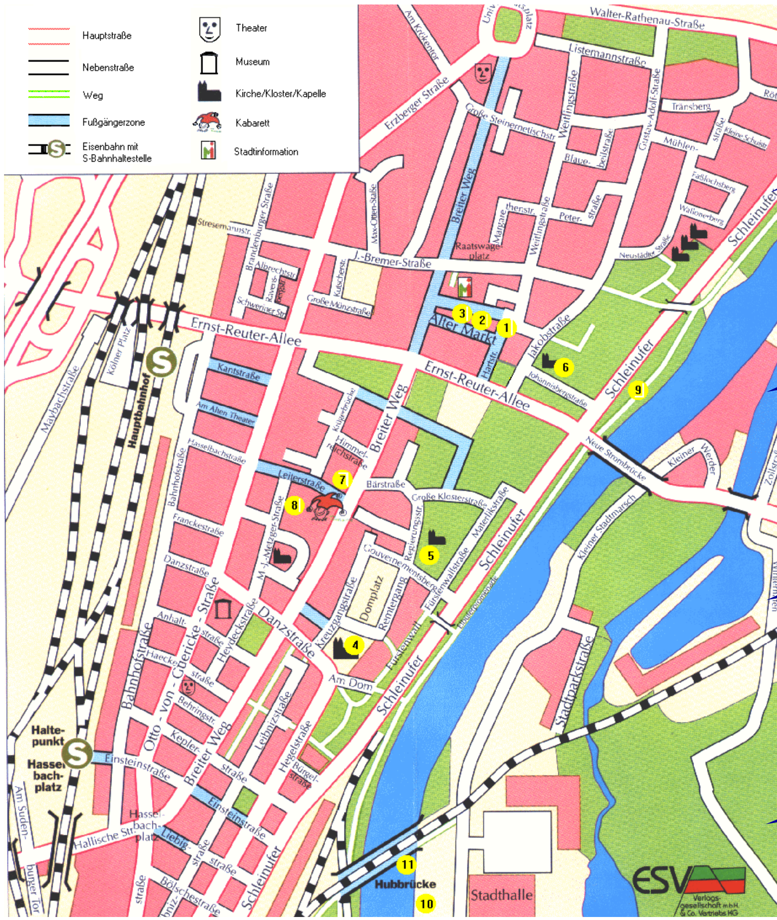
Different kinds of architecture

- Infrastructure
- Security
- Technical
- Solution
- Network
- Data
- Hardware
- Enterprise
- Application
- System
- Integration
- IT
- Database
- Information
- Process
- Business
- Software

Different kinds of architecture



City planning



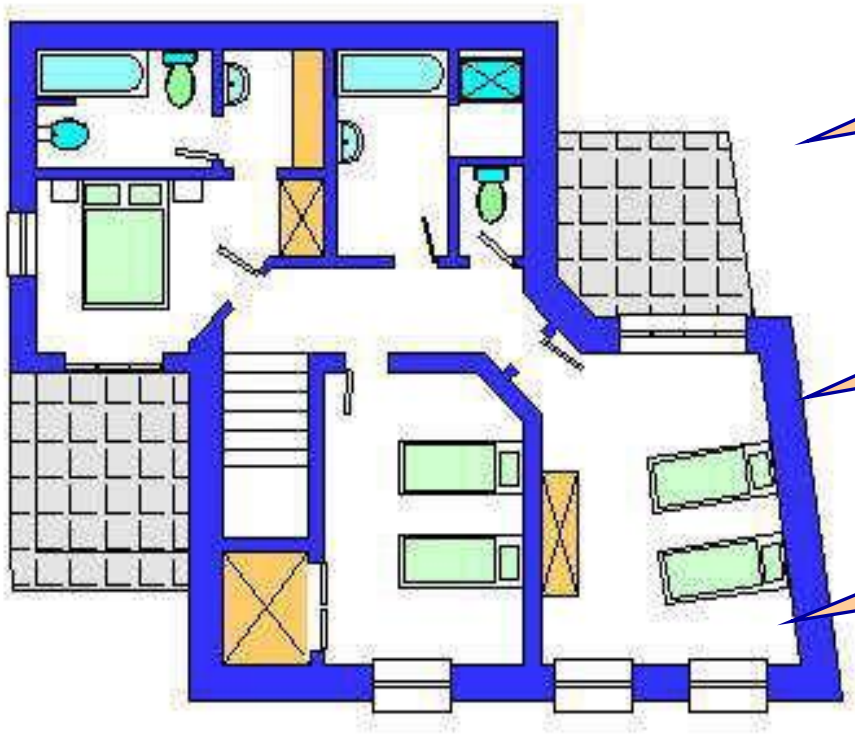
Business Architecture :
Goals of the city and the processes in a city

Information Architecture:
healthcare, education, water, electricity, transport, …

Application Architecture :
hospital, police, library, schools, …

Infrastructure Architecture :
roads, railroads, harbour, airport, …

House planning



Information Architecture:
water, electricity, heating, …

Application Architecture :
Living room, bedrooms, kitchen, …

Infrastructure Architecture :
Electrical wires, water pipes, driveway,…

Warship Vasa

- Customer
  - King of Sweden
    - Gustav II Adolf
- Requirements:
  - 70 m long
  - 300 soldiers
  - 64 guns
  - 2 decks
- Architect
  - Hendrik Hybertson

Software architecture is hard!

- Complexity
  - No physical limitations
  - Huge state-space
- Constant change of
  - Business
  - Technology
- The architecture is never ideal

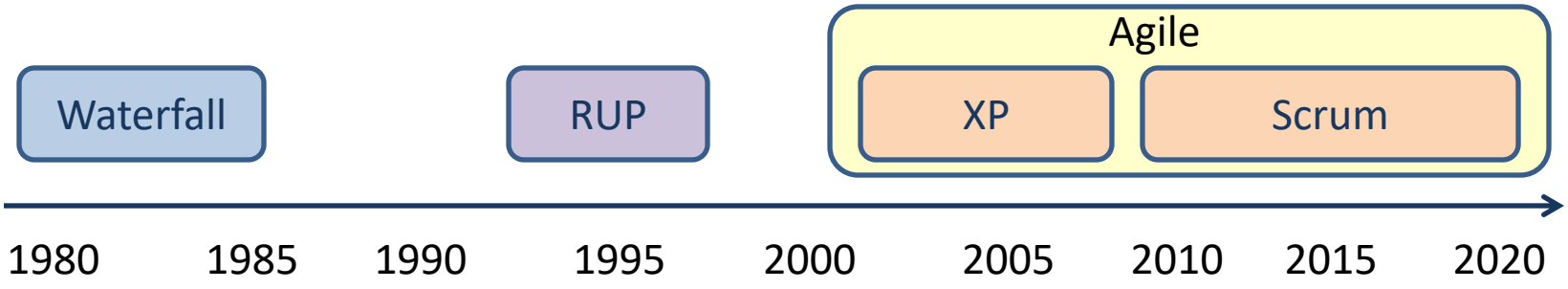
The work of an architect: Make non-optimal decisions in the dark.

Main point

- Software architecture is defining all the important stuff in a software development project.
- The human physiology has the same structure as the structure of the Veda and Vedic literature who are expressions of the structure of pure consciousness.

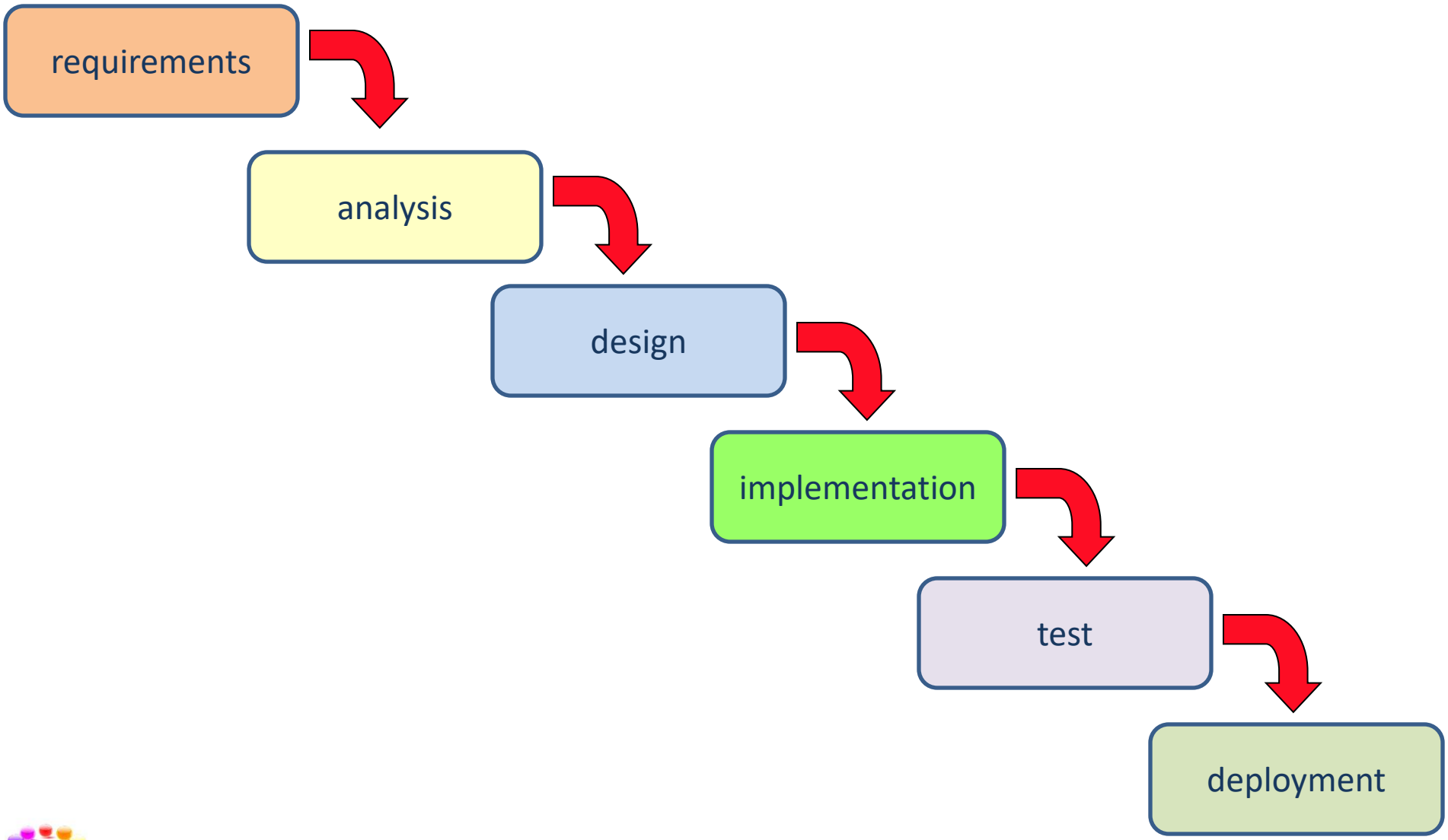
HOW DOES SOFTWARE ARCHITECTURE FIT IN THE PROCESS

Software development methods



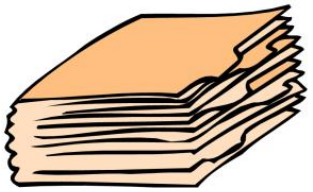
Linear	Iterations
Different roles	Cross-functional team
Document driven	Face-to-face
Customer is outside the project	Customer inside project
Large projects(time, nr. of people)	Small projects
Req. statements / Use cases	User stories

Waterfall



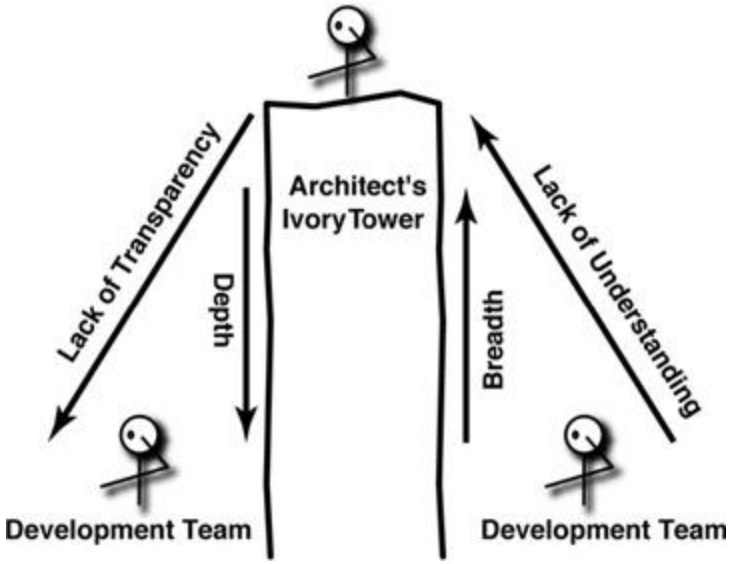
Waterfall software architecture

- The architect designs the system
  - Big upfront architecture
    - Large Software Architecture Document (SAD)
  - Ivory tower architect
    - The architecture is not understood and implemented by the developers
    - The architect does not understand the current technology
  - The architect is only available in the beginning of the project



Ivory tower architect

- It is very hard to truly know the best solutions for design problems if you are not working (coding) on the project
- It takes many iterations of a solution before it finally works - so you can’t suggest a solution and then leave



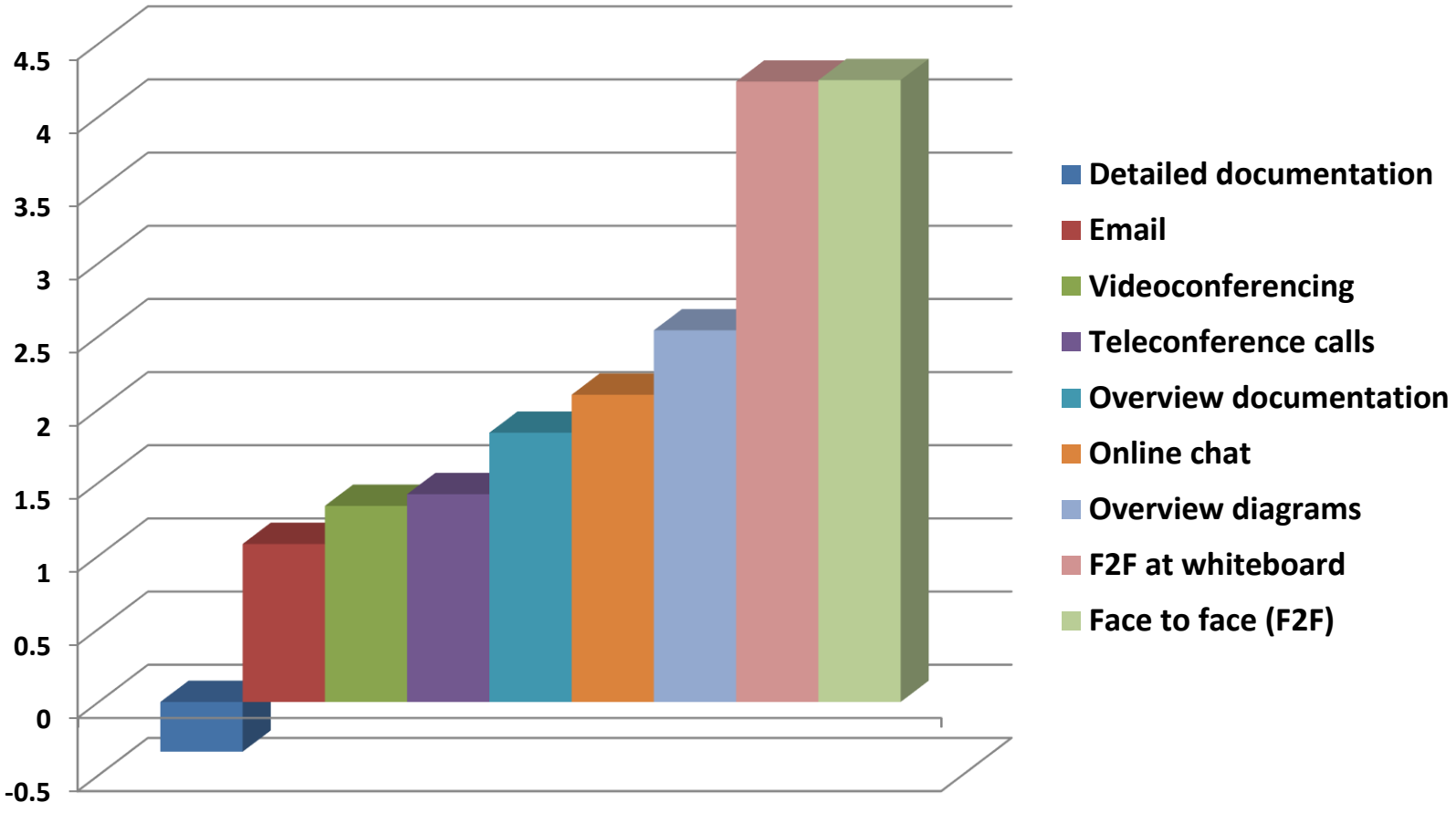
The Agile Manifesto

Individuals and interactions	over	Processes and tools
Working software	over	Comprehensive documentation
Customer collaboration	over	Contract negotiation
Responding to change	over	Following a plan

Agile principles

- Early and continuous delivery of valuable software.
- Welcome changing requirements.
- Business people and developers must work together daily.
- Give the team the environment and support they need, and trust them to get the job done.
- Prefer face-to-face conversation.
- Working software is the primary measure of progress.
- Continuous attention to technical excellence and good design
- Simplicity is essential.
- Self-organizing teams.

Effectiveness of communication



Scrum team

Is responsible that the team works in the most optimized and efficient way

Scrum Master

Product Owner

Is responsible that the business gets what they need

Developer

Developer

Devel oper

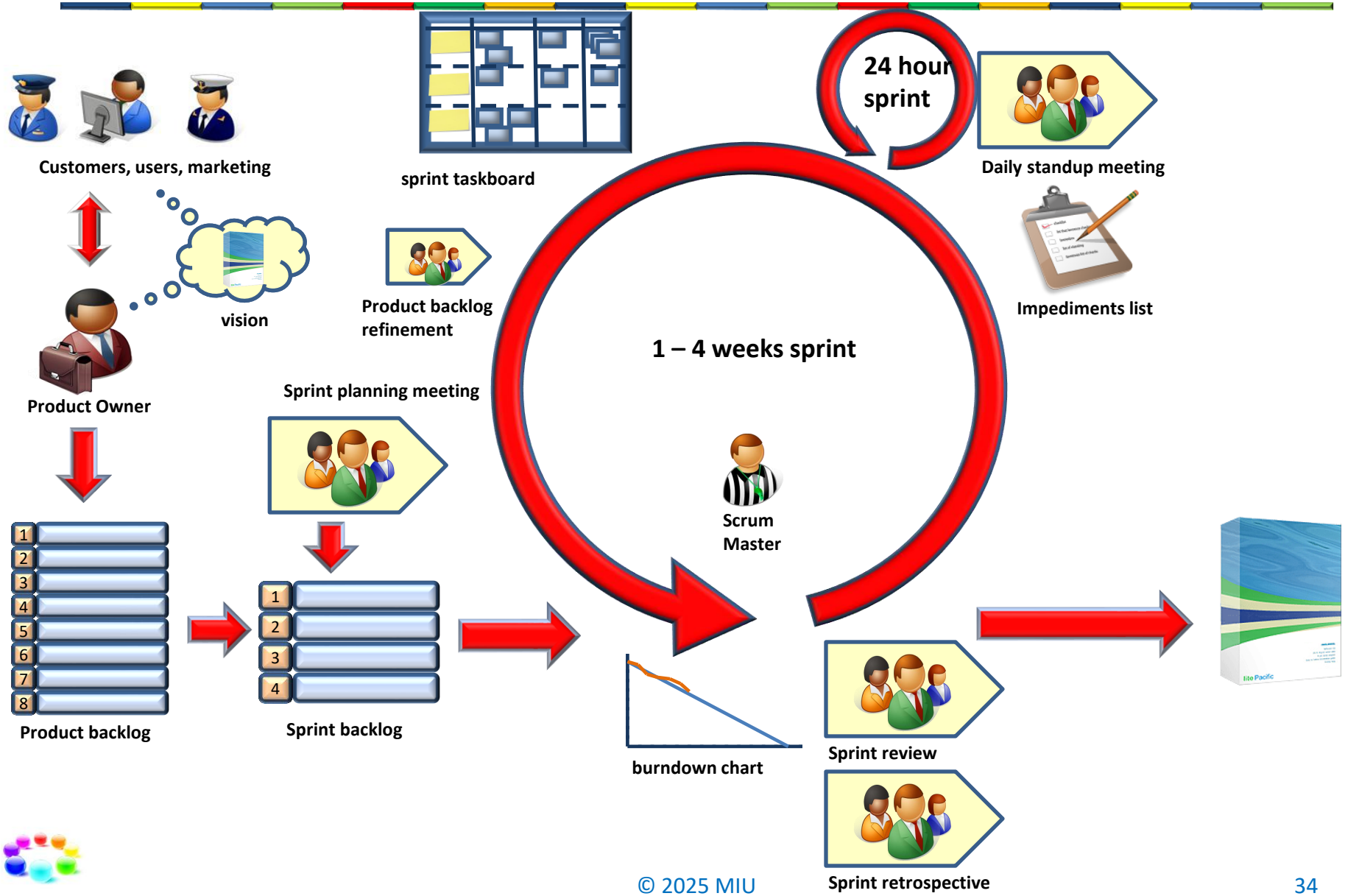
Developer

Developer

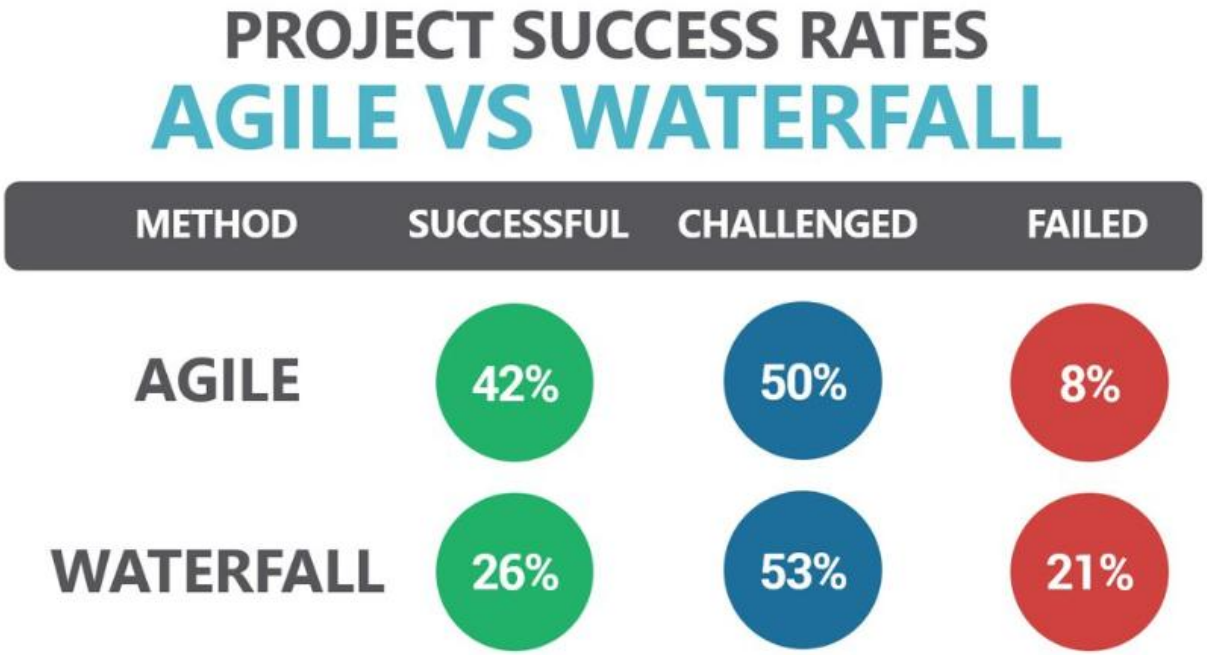
Developer

Developer

Scrum in action



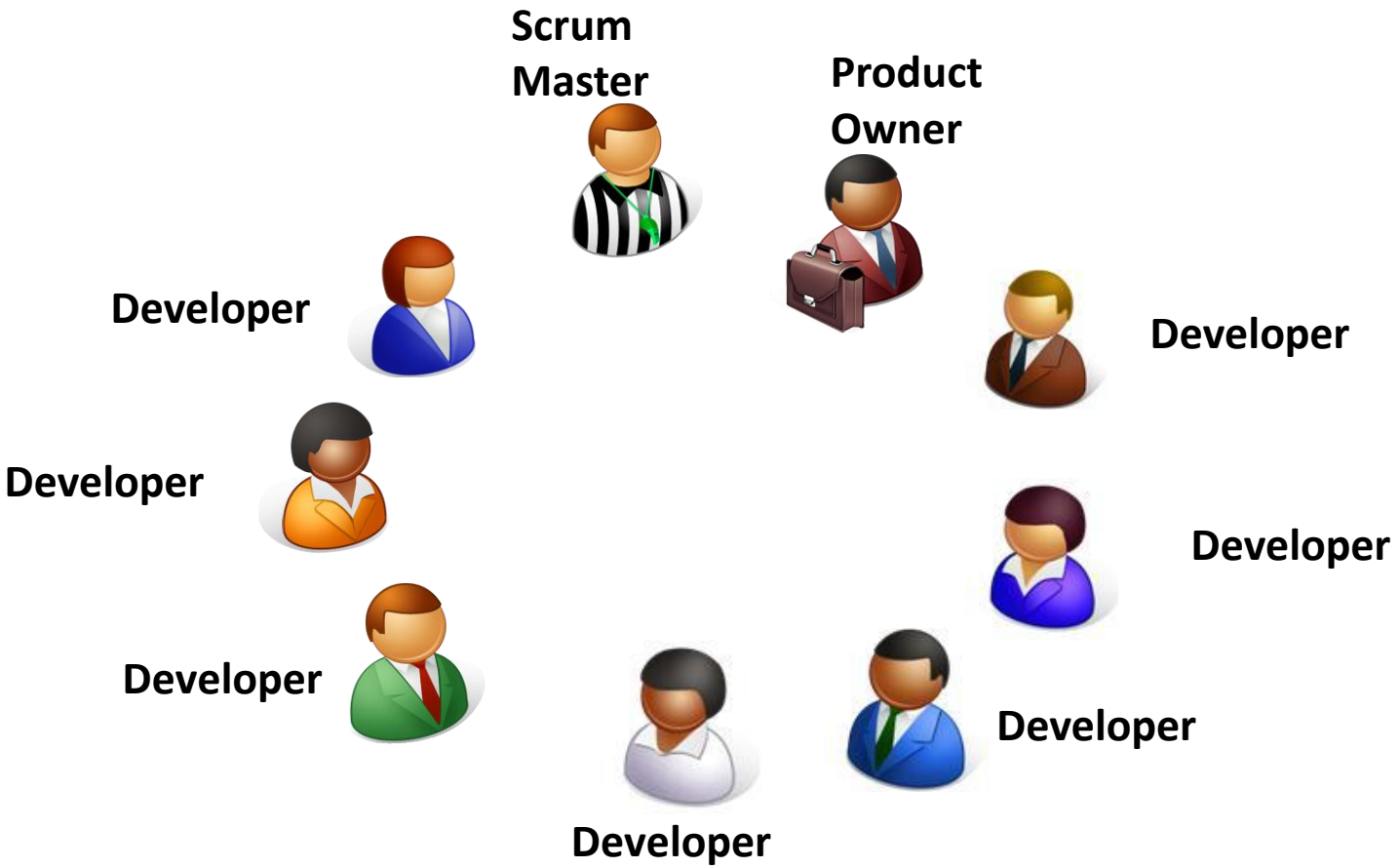
Comparing waterfall and agile



The chaos manifesto 2017

Agile architecture

- Architecture is a task, not a role
  - Team is responsible for architecture



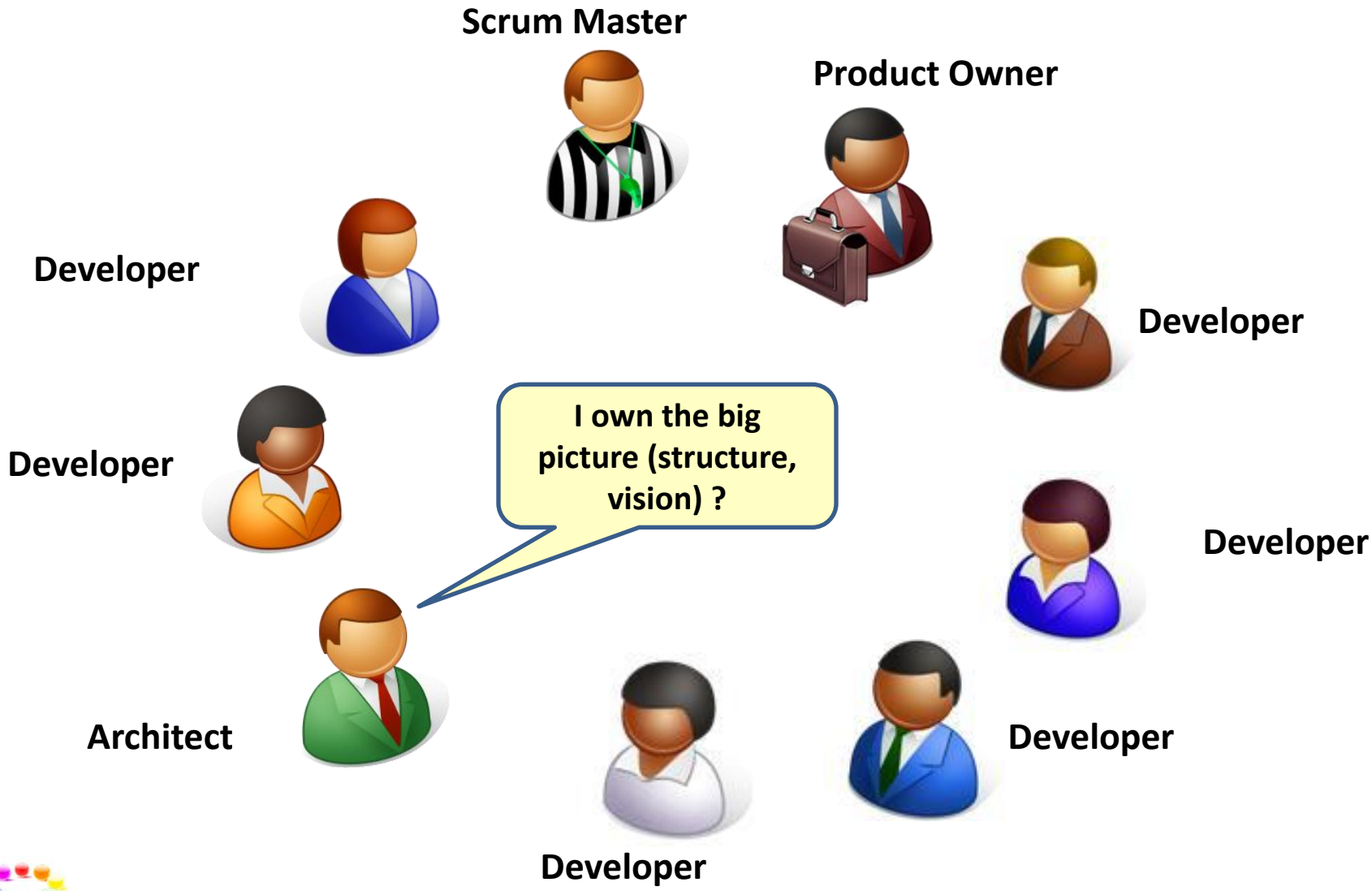
Scrum team


Scrum Master
Product Owner
Focus on the code
Developer
Developer
Developer
Who owns the big picture (structure, vision) ?
Developer
Developer
Developer
Developer

Why software architecture in agile?

- We need a clear vision and roadmap for the team to follow.
- We need to identifying and mitigating risk.
- We have to communicate our solution at different levels of abstraction to different audiences.
- We need technical leadership
- We have to make sure our architecture is consistent, correct and fits within the context

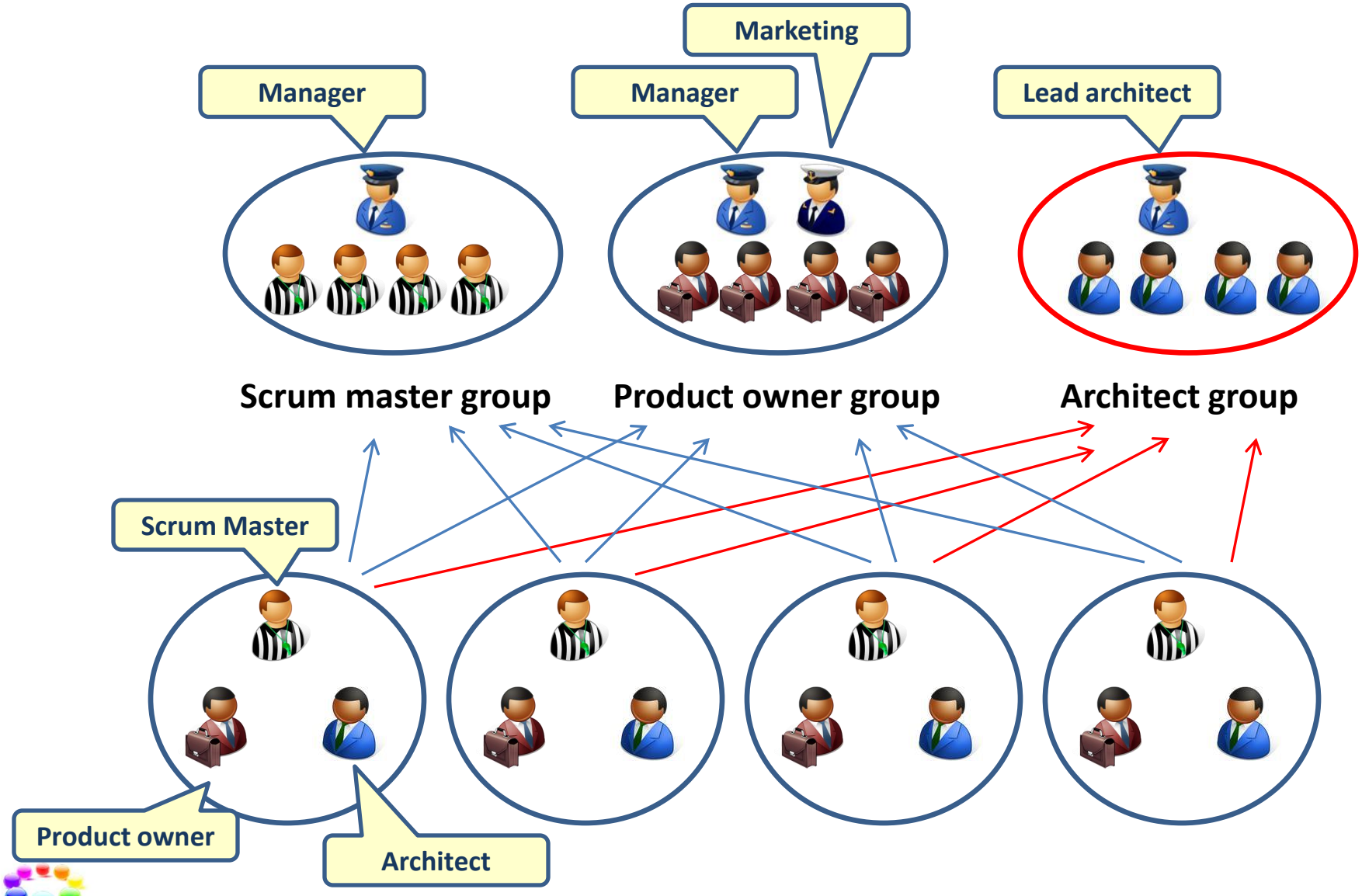
You need an architect



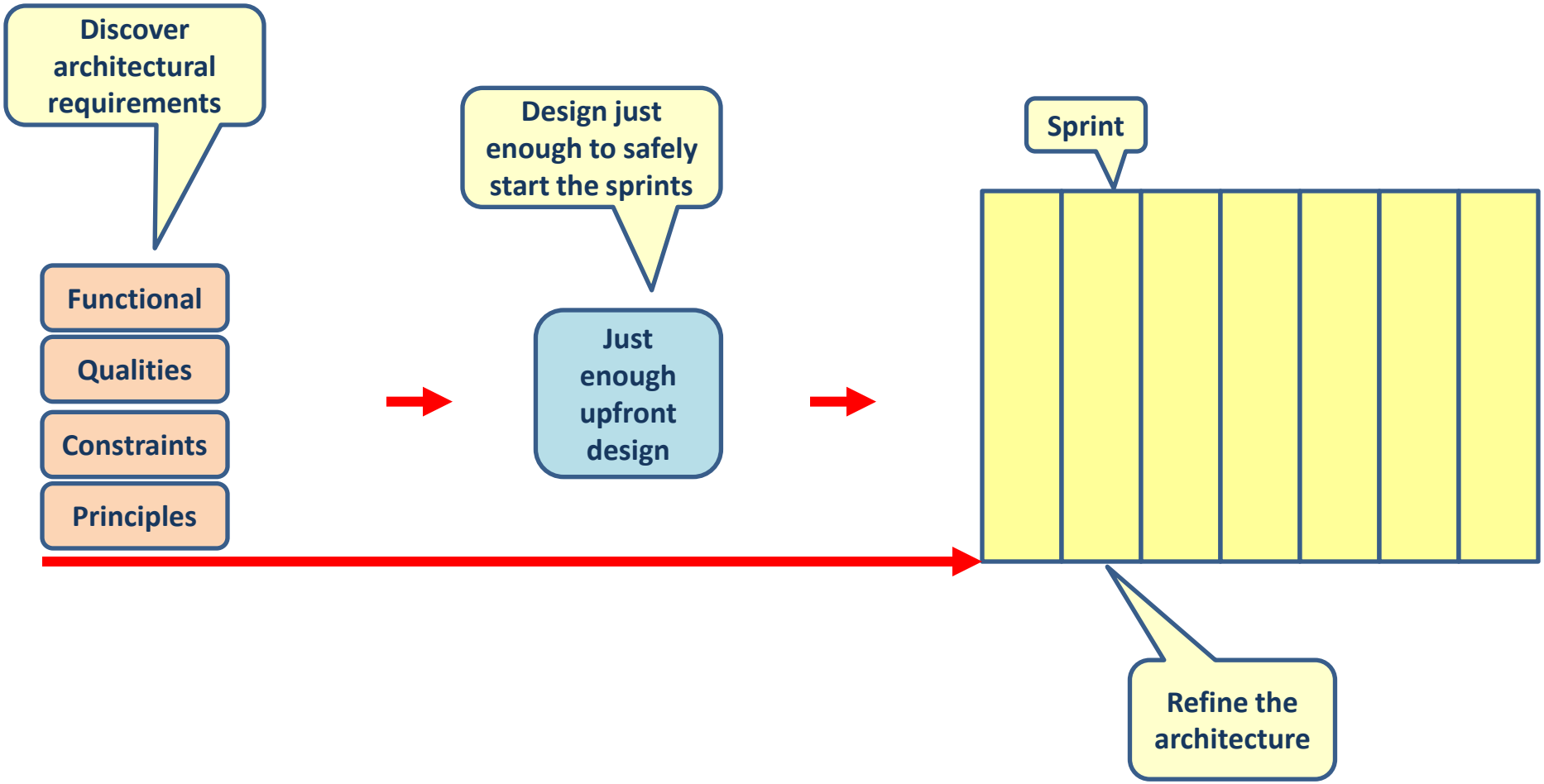
Agile architecture

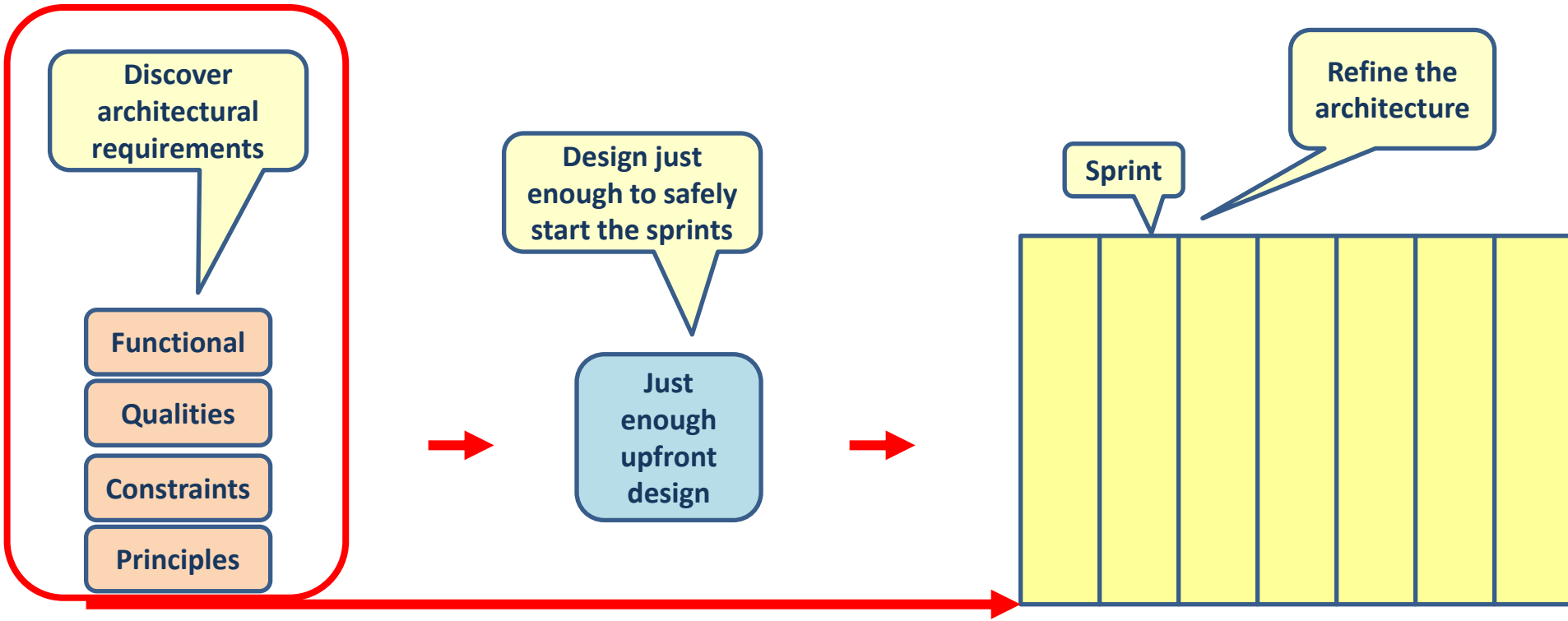
- Just enough upfront architecture
  - Refine the architecture later
- Keep your architecture flexible
- The architect is available during the whole project
- The architect also writes code
  - But not all the time
  - The architecture is grounded in reality
  - Works together with the developers
  - Architects should be master builders
- Proof the architecture in the first iteration(s)

Architect group



Agile architecture

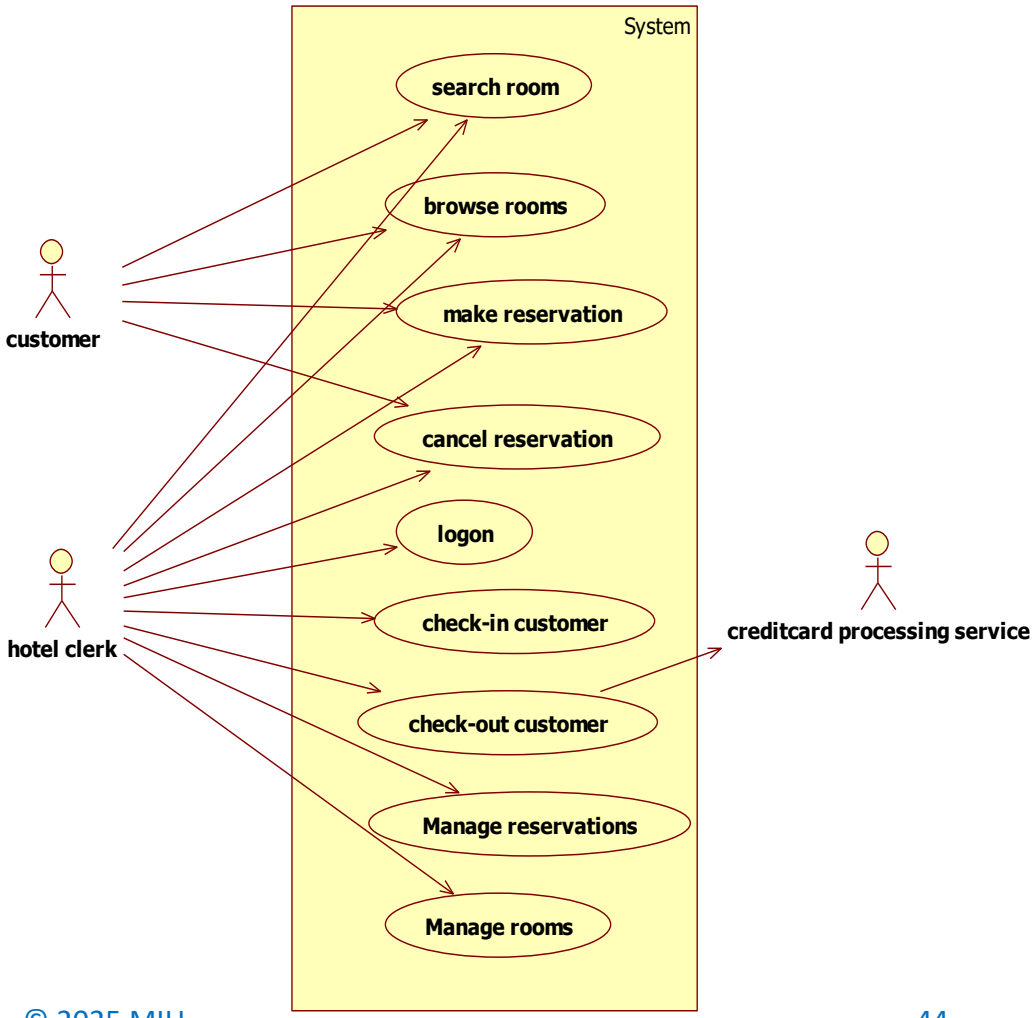




ARCHITECTURAL REQUIREMENTS

Functional requirements

- What should the system functionally do?
  - Use cases
  - User stories



As a customer

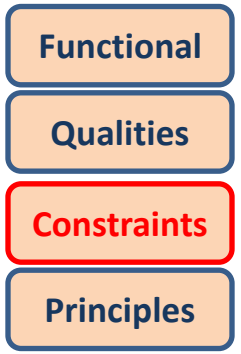
I can view my account history

so that I know all transactions on my account

- **Functional**
- **Qualities**
- **Constraints**
- **Principles**

Architectural constraints

- Constraints from the business (or enterprise architecture
  - We do everything in .Net
  - We always use an Oracle database
  - Our maintenance engineers all know Java
  - All applications talk with the Oracle ESB
- Budget
- Deadlines



SOFTWARE QUALITIES

Wikipedia software qualities

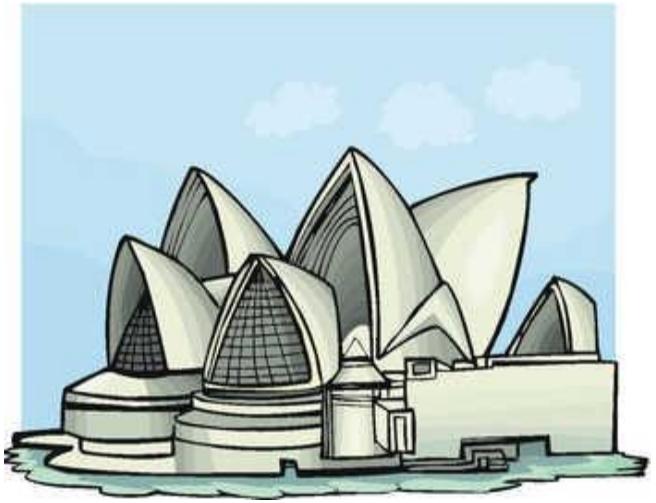
- accessibility
- accountability
- accuracy
- adaptability
- administrability
- affordability
- agility (see Common subsets below)
- auditability
- autonomy [Erl]
- availability
- compatibility
- composability [Erl]
- configurability
- correctness
- credibility
- customizability
- debuggability
- degradability
- determinability
- demonstrability
- dependability (see Common subsets below)
- deployability
- discoverability [Erl]
- distributability
- durability
- effectiveness
- efficiency
- evolvability
- extensibility
- failure transparency
- fault-tolerance
- fidelity
- flexibility
- inspectability
- installability
- integrity
- interchangeability
- interoperability [Erl]
- learnability
- localizability
- maintainability
- manageability
- mobility
- modifiability
- modularity
- observability
- operability
- orthogonality
- portability
- precision
- predictability
- process capabilities
- producibility
- provability
- recoverability
- relevance
- reliability
- repeatability
- reproducibility
- resilience
- responsiveness
- reusability [Erl]
- robustness
- safety
- scalability
- seamlessness
- self-sustainability
- serviceability (a.k.a. supportability)
- securability (see Common subsets below)
- simplicity
- stability
- standards compliance
- survivability
- sustainability
- tailorability
- testability
- timeliness
- traceability
- transparency
- ubiquity
- understandability
- upgradability
- usability
- vulnerability

ARCHITECTURE PRINCIPLES

Architecture (design) principles

- Keep it simple
- Keep it flexible
- Loose coupling
  - High cohesion, low coupling
- Separation of concern
- Information hiding
- Principle of modularity
- Open-closed principle

Keep it simple



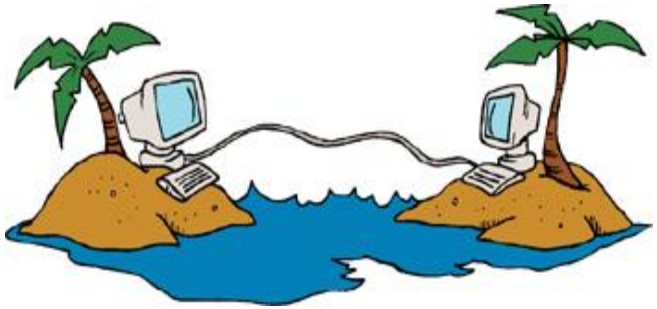
- The more complexity, the more change on failure
- Simple applications, frameworks, tools, etc. remain to be used
  - Complex ones will be replaced by something simple
- Gold plating

Keep it flexible

- Everthing changes
  - Business
  - Technical
- More flexibility leads to more complexity

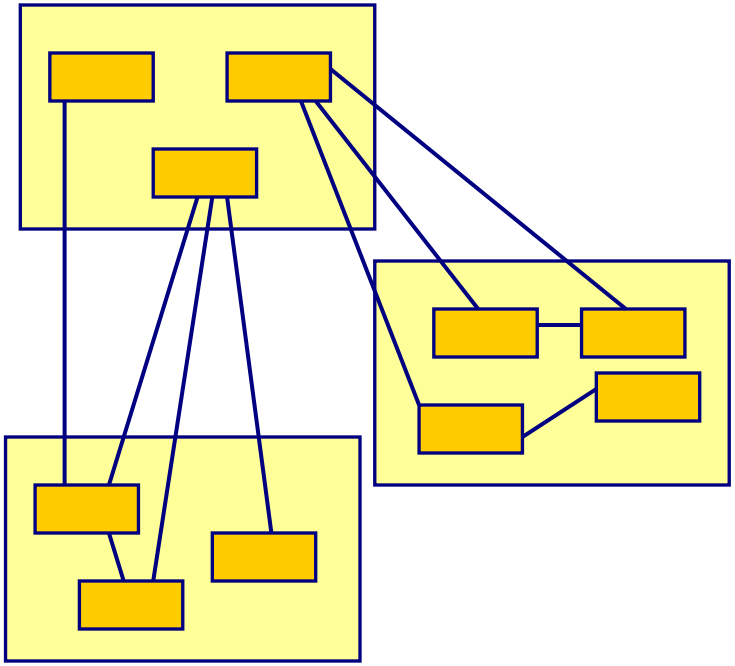
Loose coupling

- Different levels of coupling
  - Technology
  - Time
  - Location
  - Data structure
- You need coupling somewhere
  - Important is the level of coupling

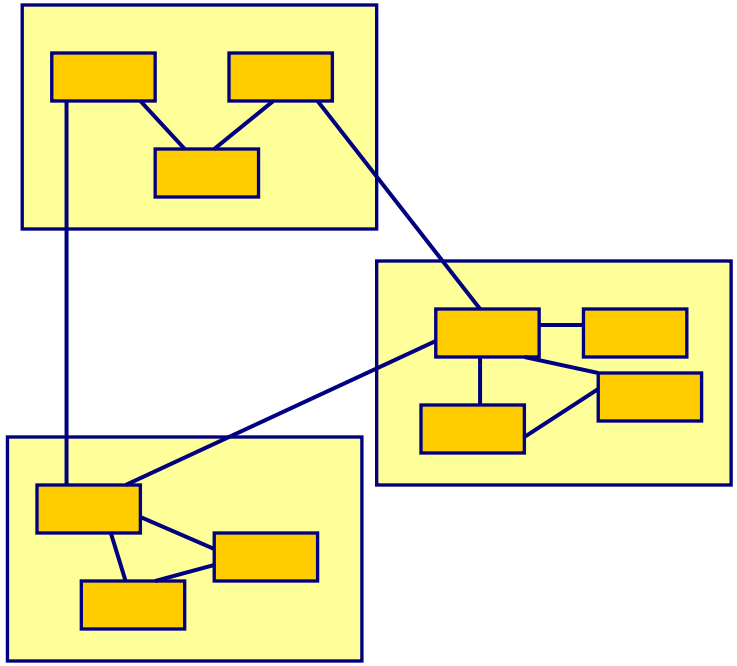


High cohesion, low coupling

- High coupling, low cohesion



- High cohesion, low coupling



Separation of concern

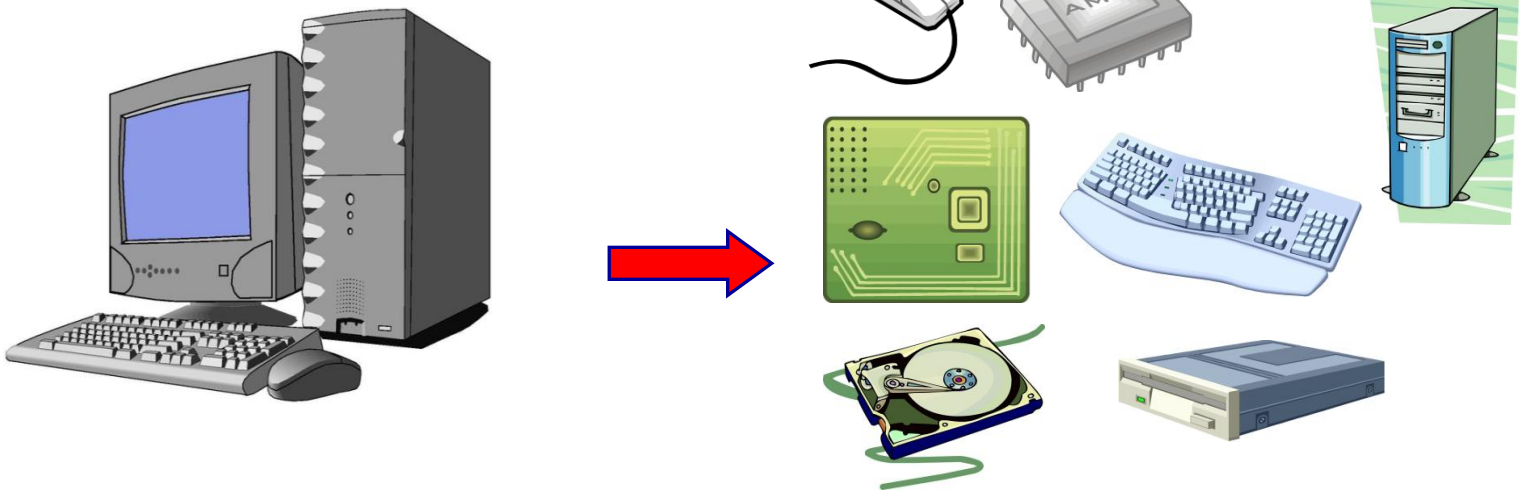
- Separate technology from business
- Separate stable things from changing things
- Separate things that need separate skills
- Separate business process from application logic
- Separate implementation from specification

Information hiding

- Black box principle
- Hide implementation behind an interface
- Hide the data structure behind stored procedures
- Hide the data structure behind business logic

Principle van modularity

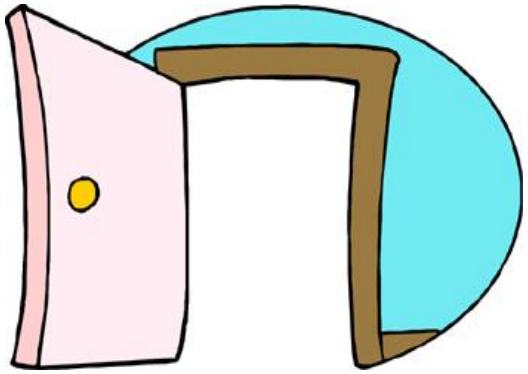
- Decomposition
- Devide a big complex problem is smaller parts
- Use components that are
  - Better understandable
  - Independent
  - Reusable
- Leads to more flexibility
- Makes finding and solvings bugs easier



Open- closed principle

- The design should be “open” for extension, but “closed” for change.

- You want to add new functionality instead of changing existing, working and tested code.

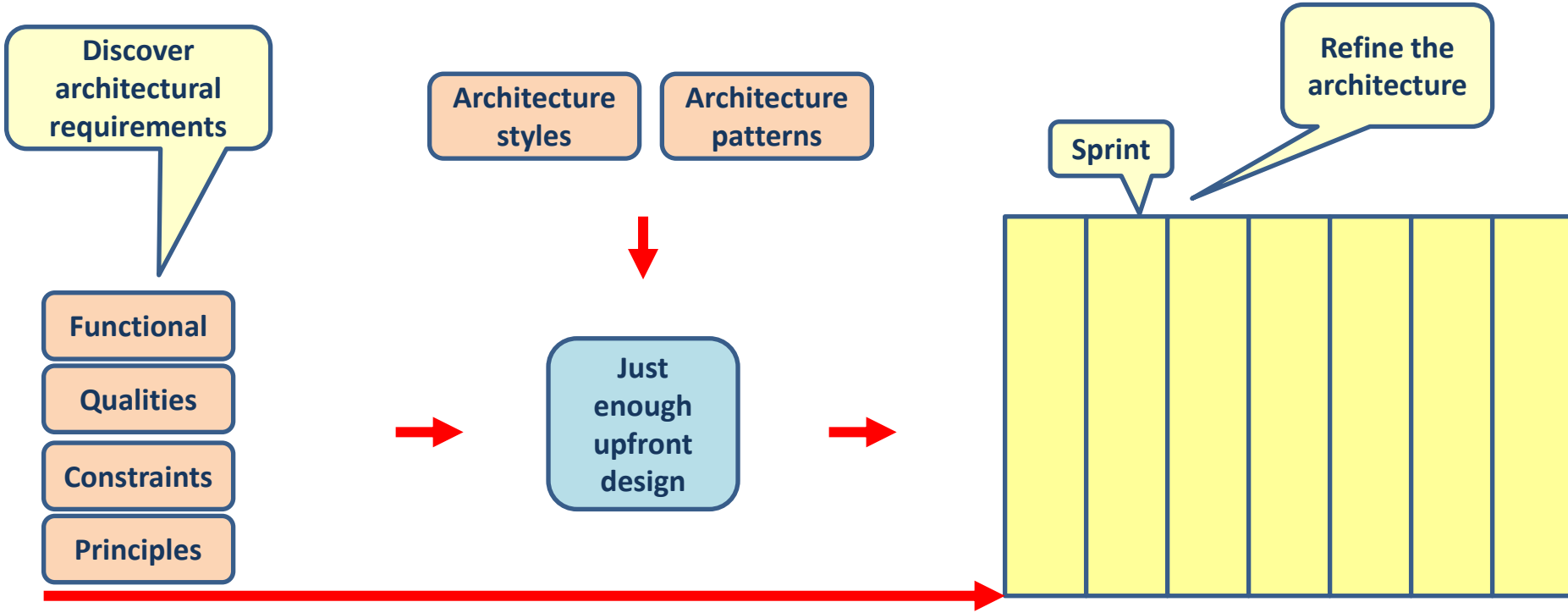


Most important architecture principles

- Keep it simple
- Keep it flexible
- Loose coupling
  - High cohesion, low coupling
- Separation of concern
- Information hiding
- Principle of modularity
- Open-closed principle

Main point

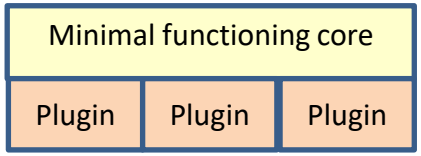
- Software architecture is never ideal. We have to find the right balance between the different software qualities and architecture principles
- Nature always takes the path of least resistance so that the perfection of the unified field can express itself in the relative creation



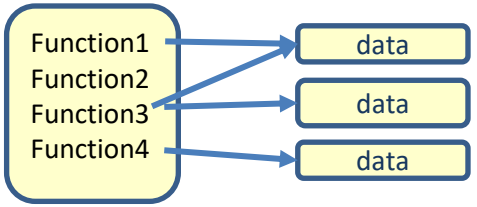
ARCHITECTURAL STYLES

Architecture styles within an application

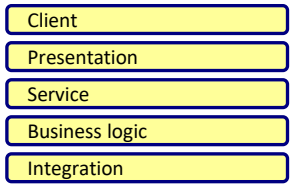
Client-server



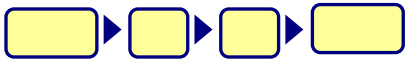
Microkernel



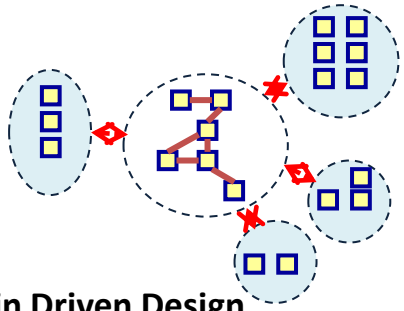
Procedural



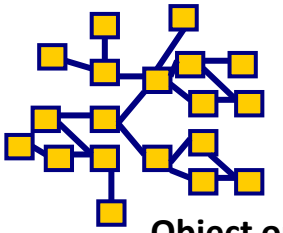
Layering



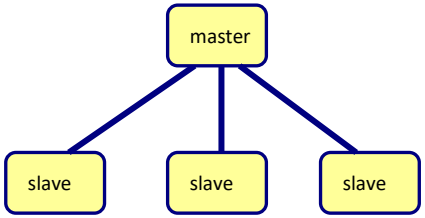
Pipe-and-Filter



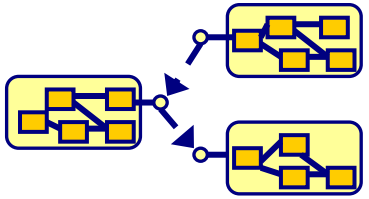
Domain Driven Design



Object oriented

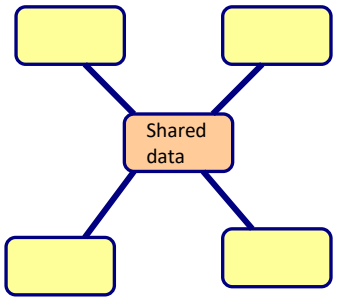


Master-Slave

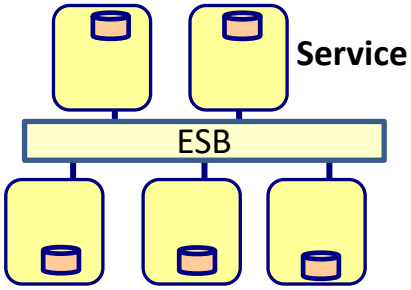


Component based

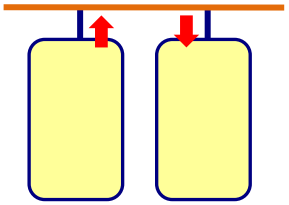
Architecture styles to connect applications



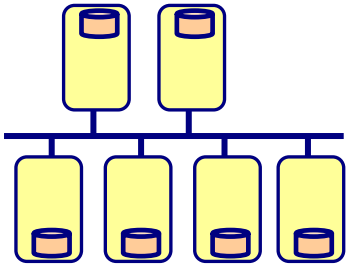
Blackboard



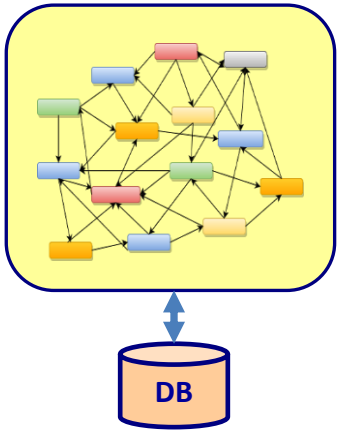
Service oriented



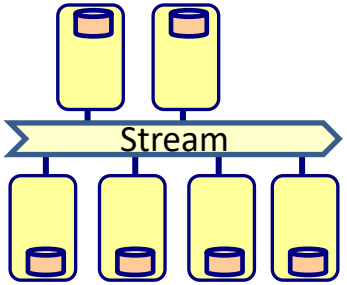
Event Driven



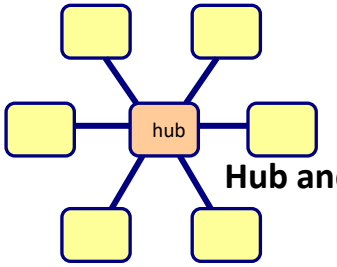
Microservices



Monolith



Stream based

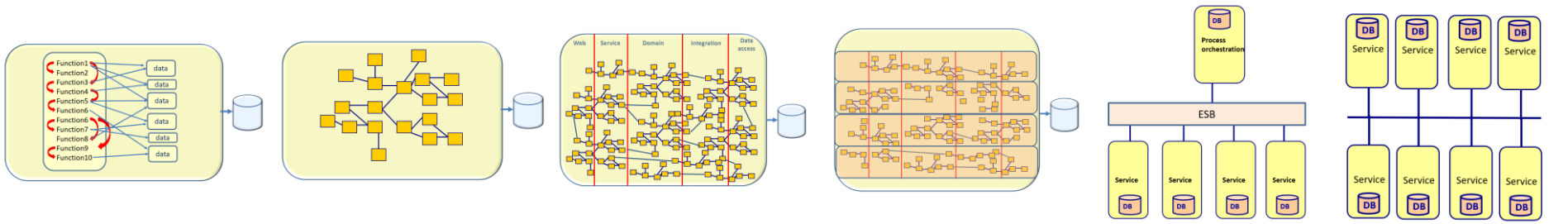


Hub and Spoke

Architecture evolution

Monolith

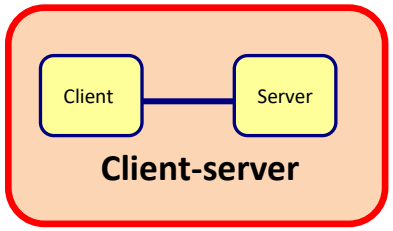
Distributed system



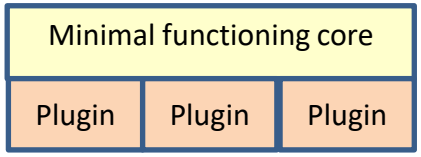
procedural object-oriented layering components SOA microservices

- Smaller and simpler parts
- More separation of concern
- More abstraction
- Less dependencies

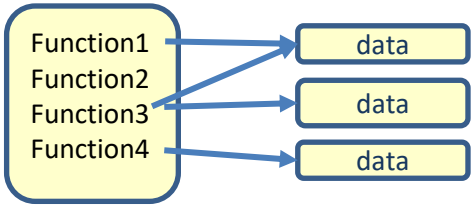
Architecture styles within an application



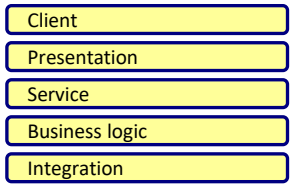
Client-server



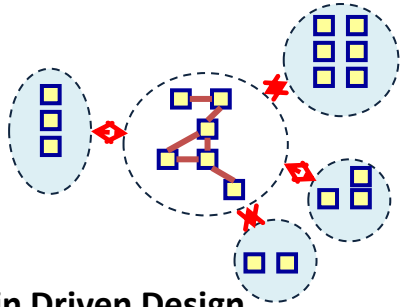
Microkernel



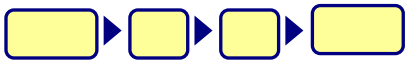
Procedural



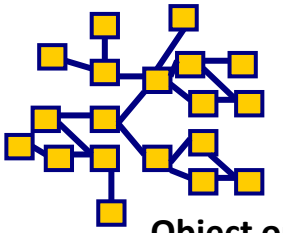
Layering



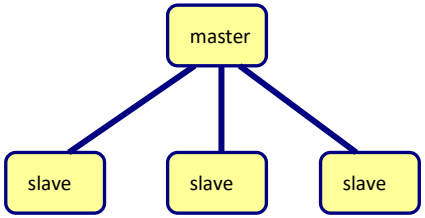
Domain Driven Design



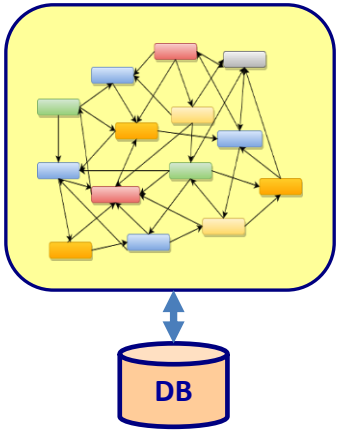
Pipe-and-Filter



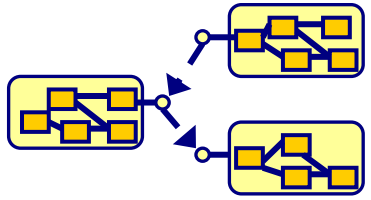
Object oriented



Master-Slave



Monolith

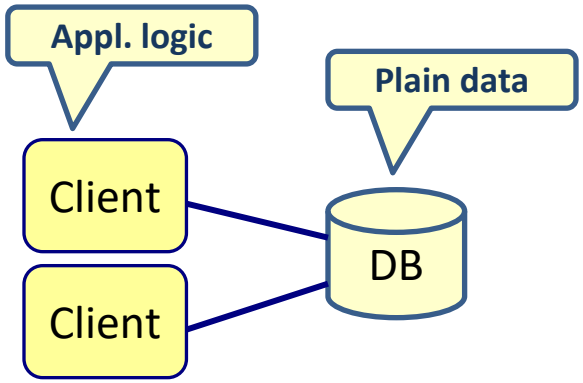


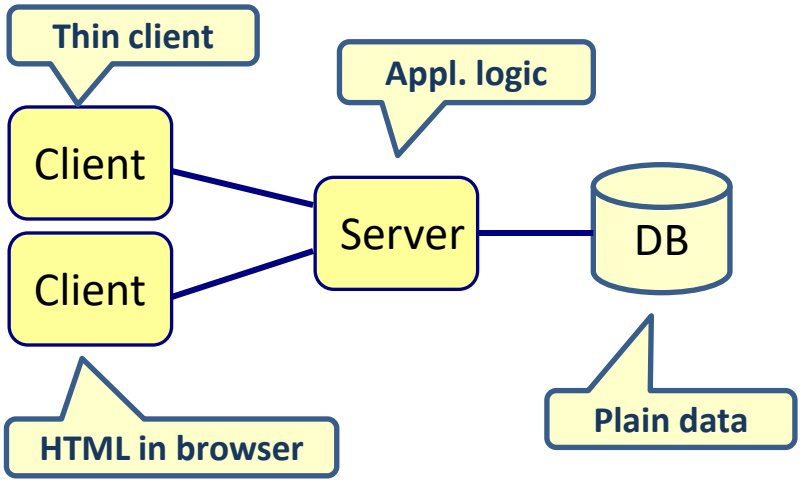
Component based

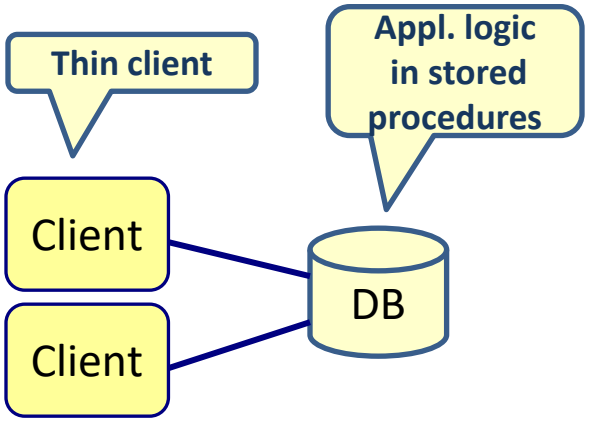
Client-Server

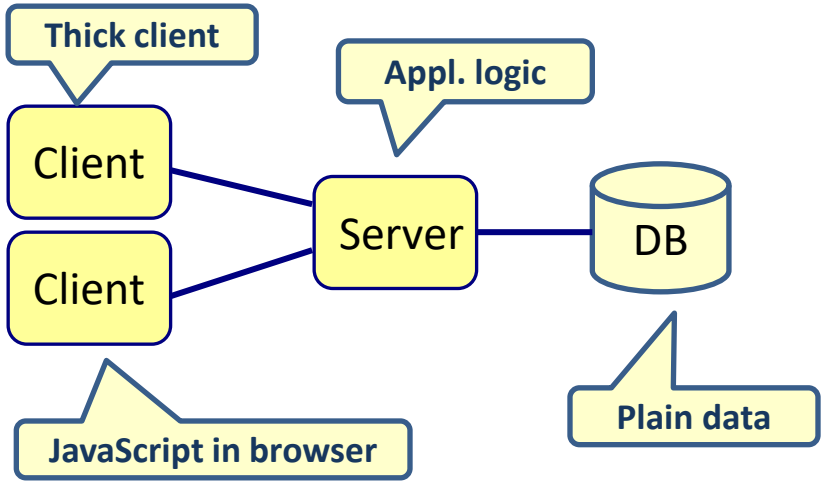
- Multiple clients per server
- Thin-client/Thick client
- Stateless / stateful server
- Multiple tiers
- Requests typically handled in separate threads

Client-server architectures

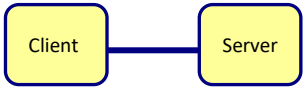




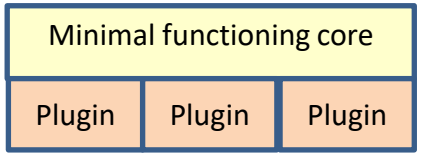




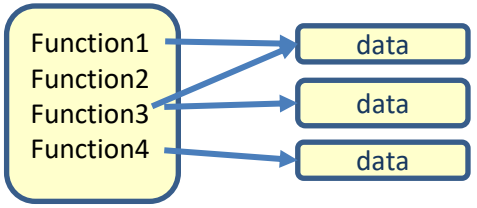
Architecture styles within an application



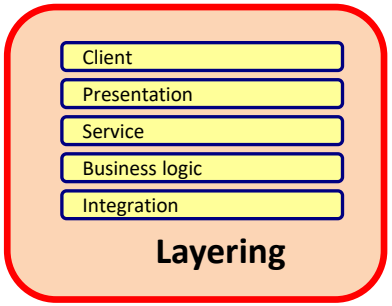
Client-server



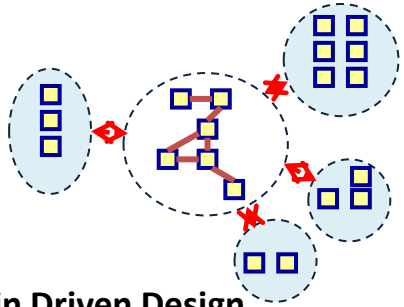
Microkernel



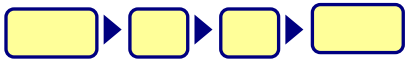
Procedural



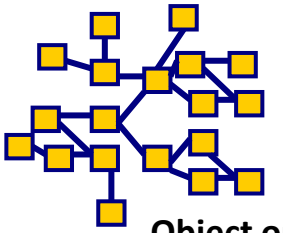
Layering



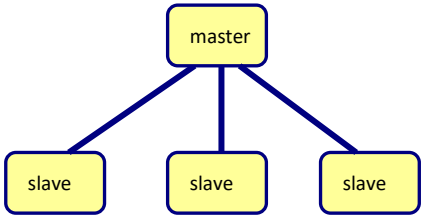
Domain Driven Design



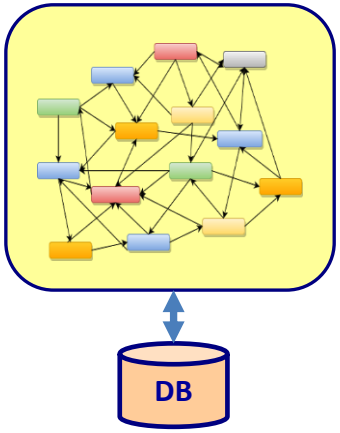
Pipe-and-Filter



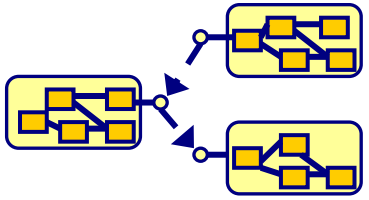
Object oriented



Master-Slave

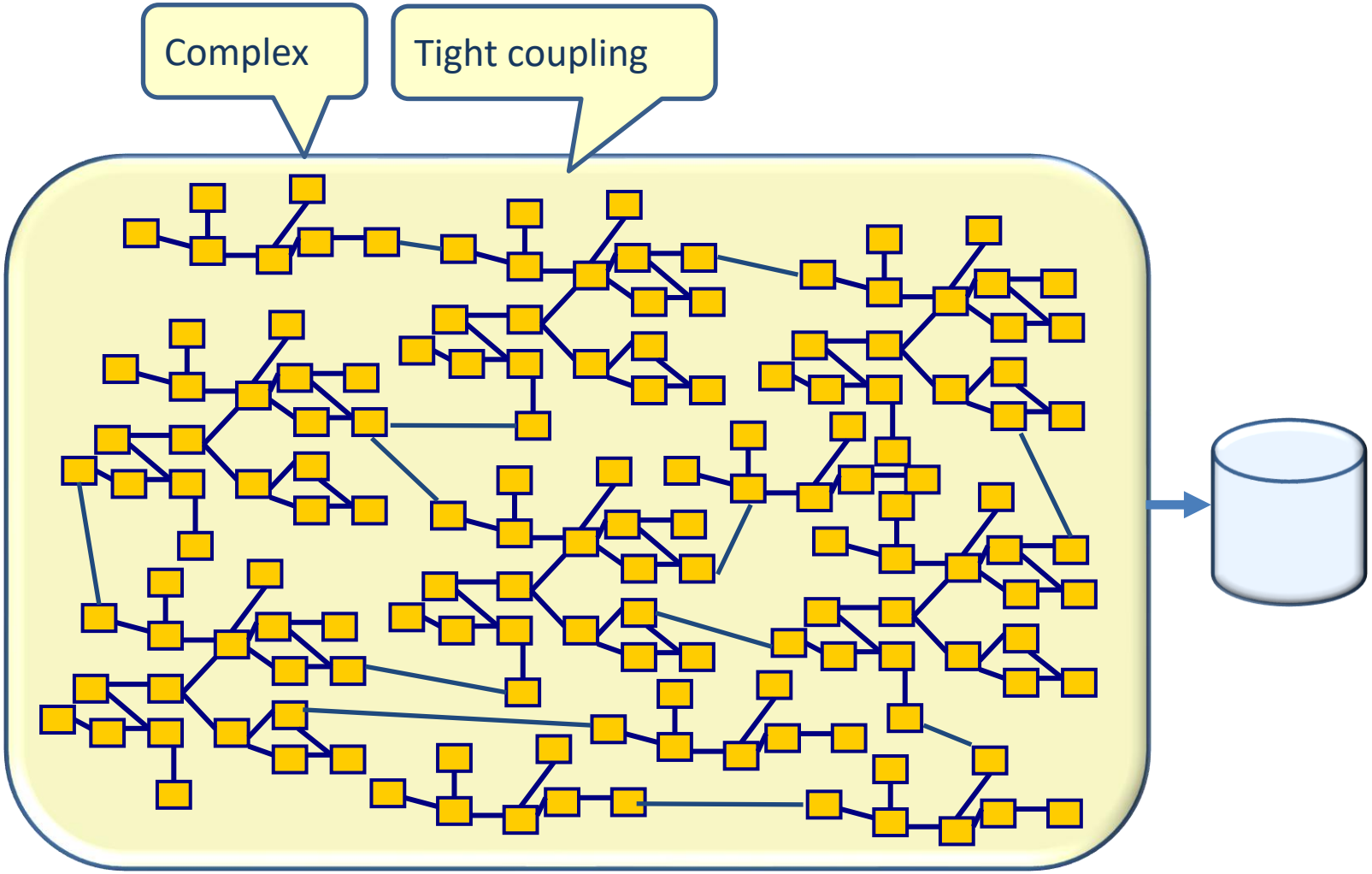


Monolith



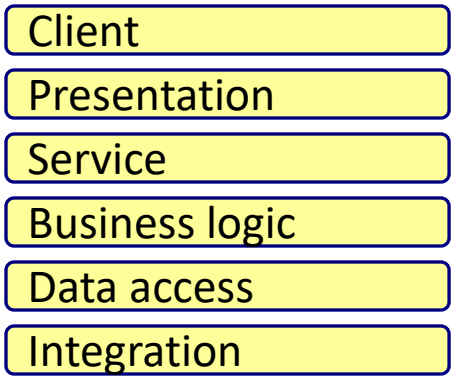
Component based

Object orientation

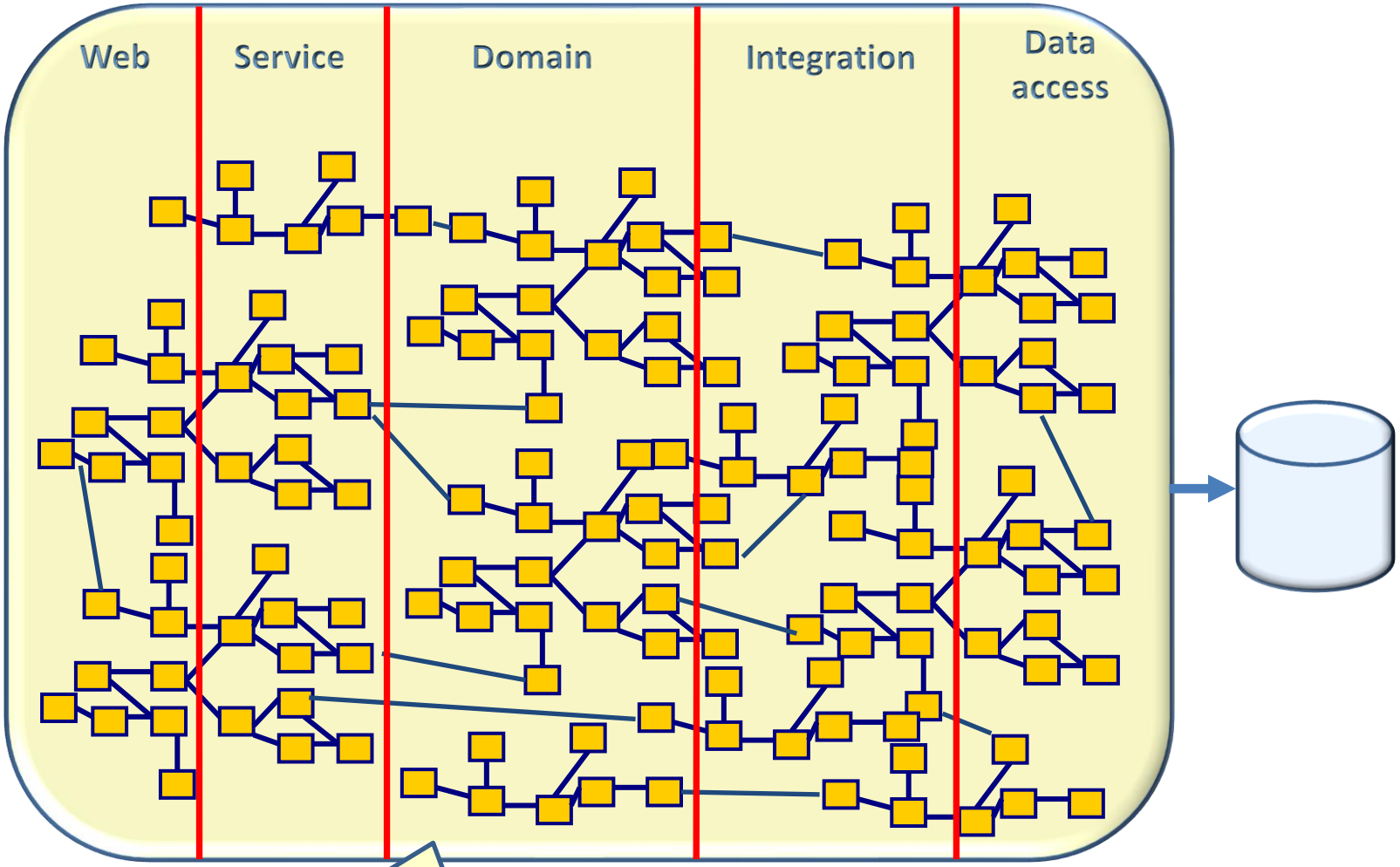


Layering

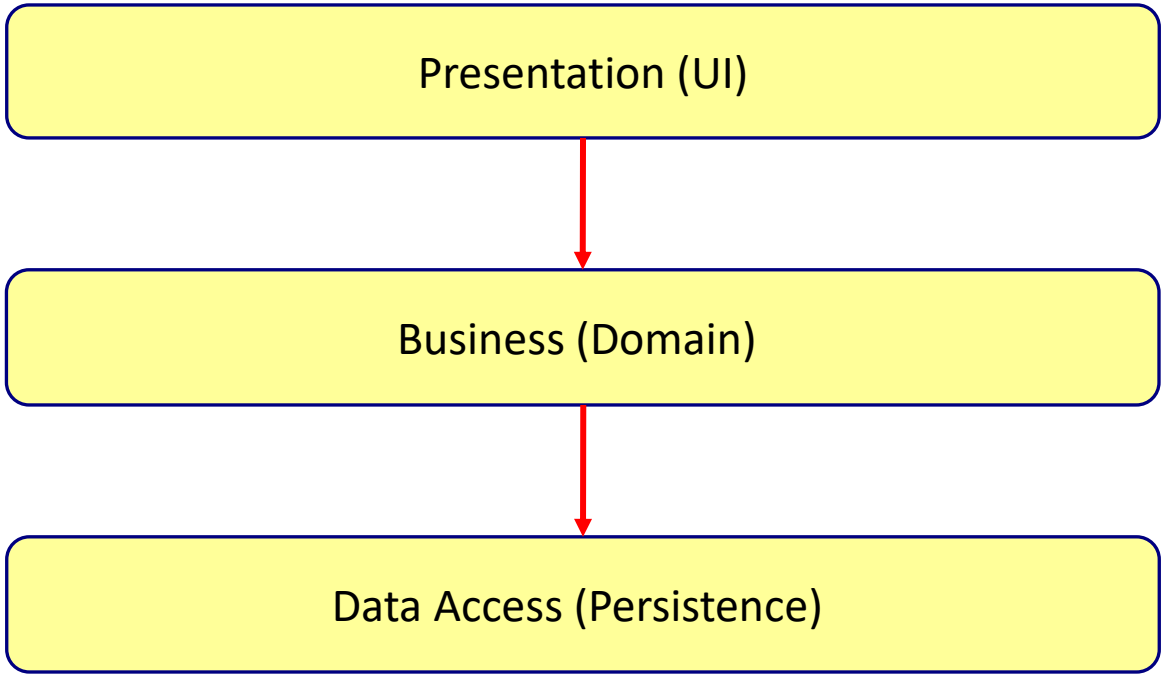
- Separation of concern
- Layers are independent
- Layers can be distributed
- Layers use different techniques



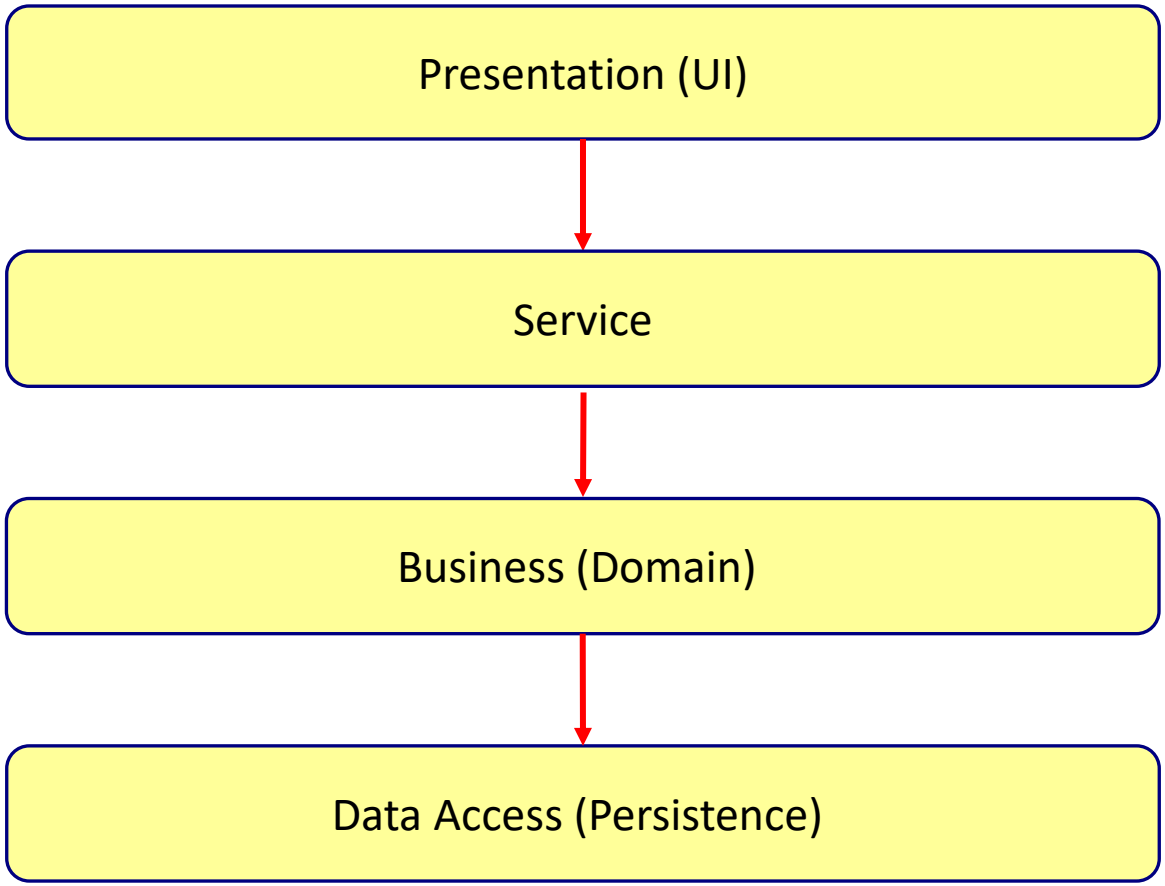
Layering



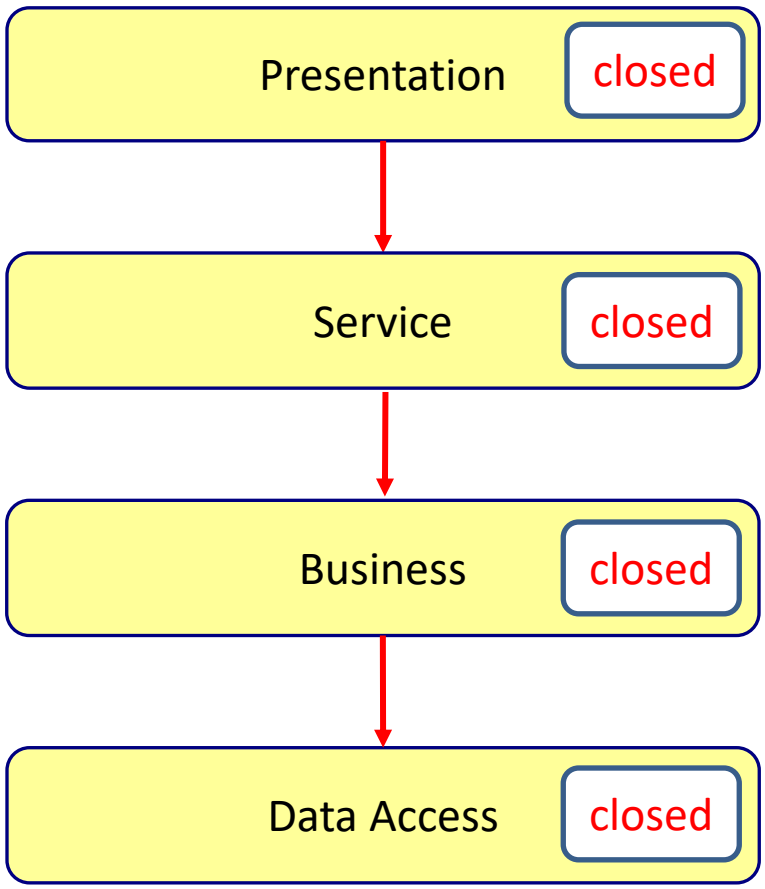
3 layered architecture

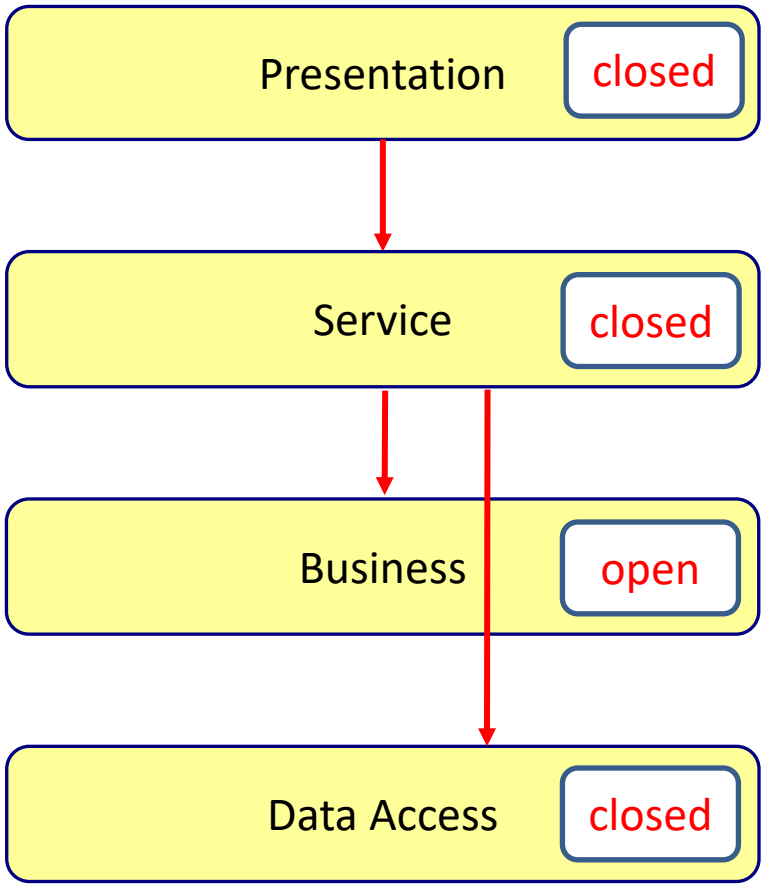


4 layered architecture

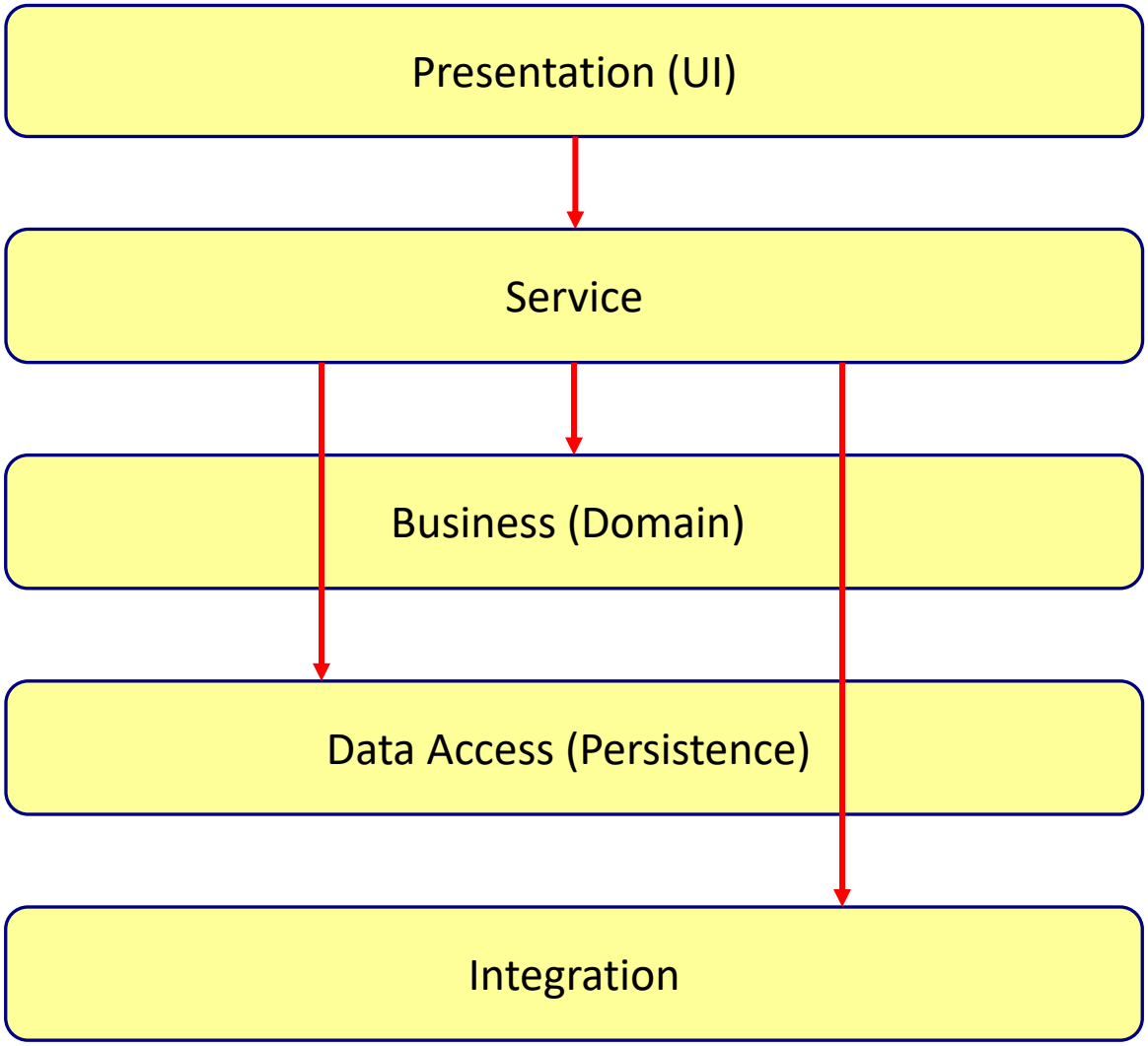


Open and closed layers

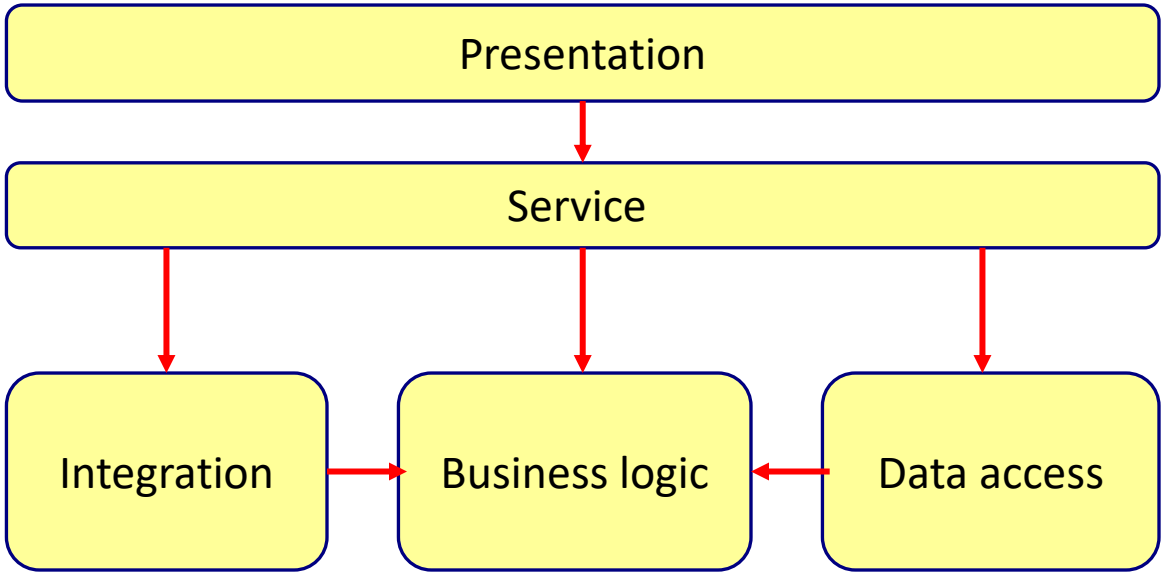




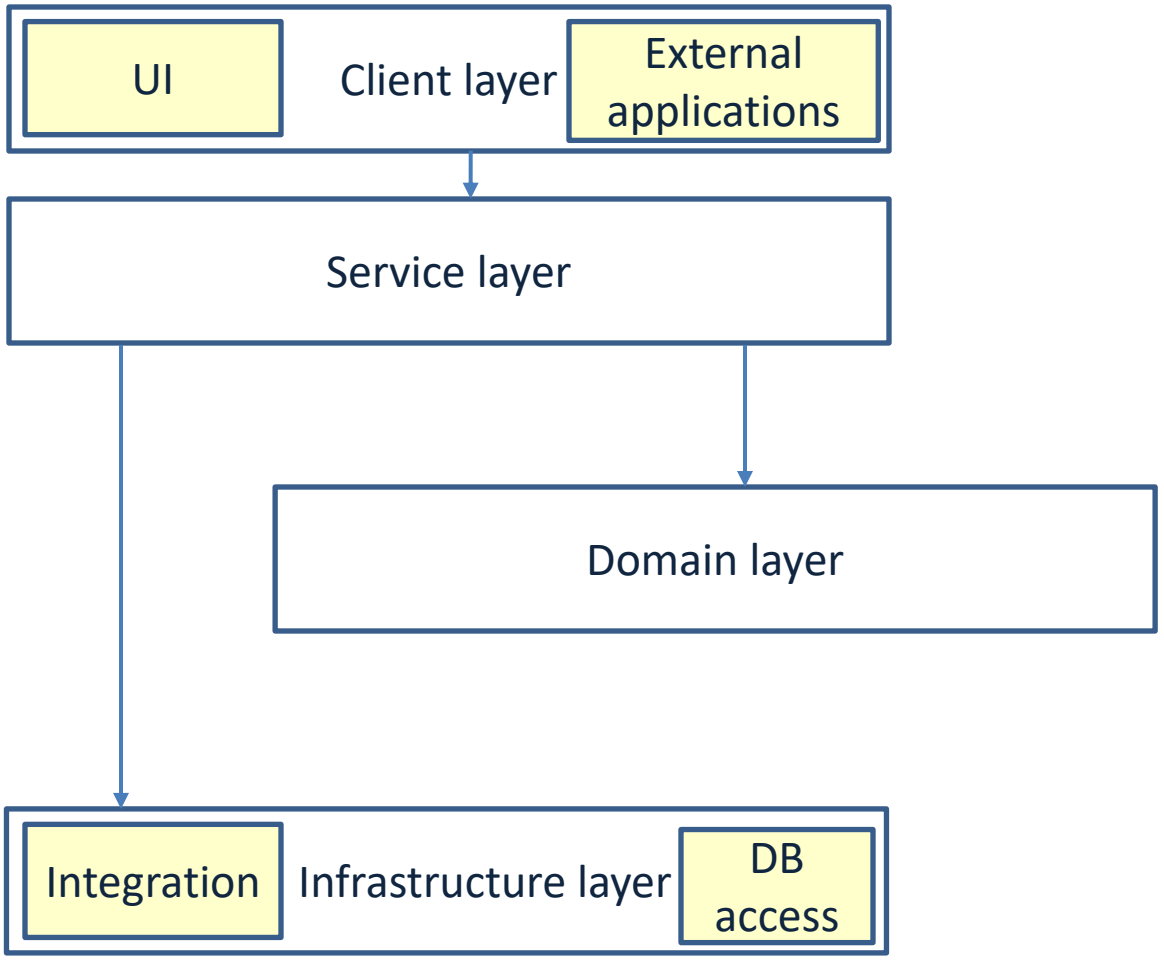
5 layered architecture



Layered architecture



Layered architecture



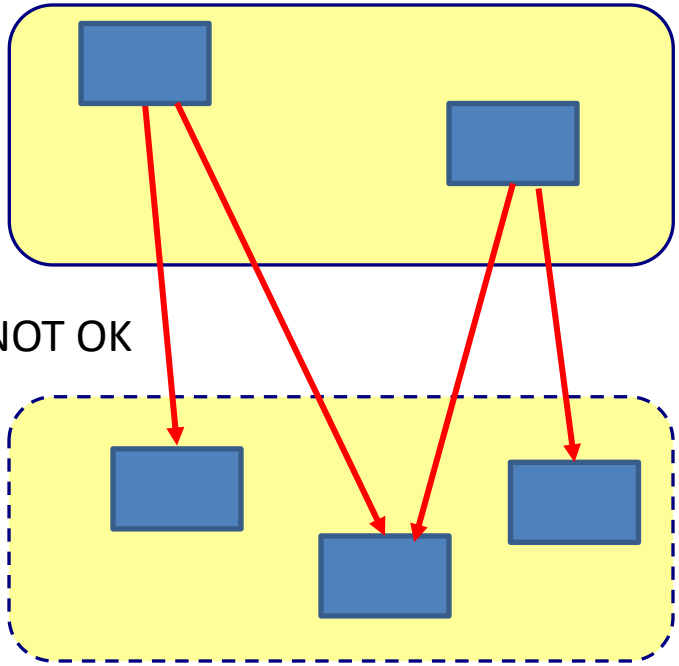
Layering

- Benefits
  - Layers can be distributed
  - Separation if concern
    - Different skills required in each layer
    - Easy to modify
    - Easy to test
- Drawbacks
  - Development effort can increase
  - Performance can become an issue

Layering anti patterns

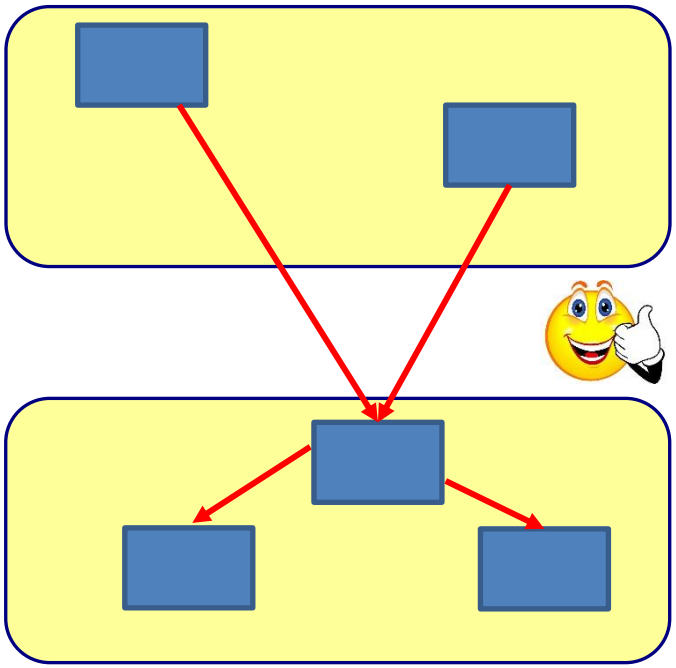
- Too much layers
- No logic in layers
- No encapsulation of layers

White box layering



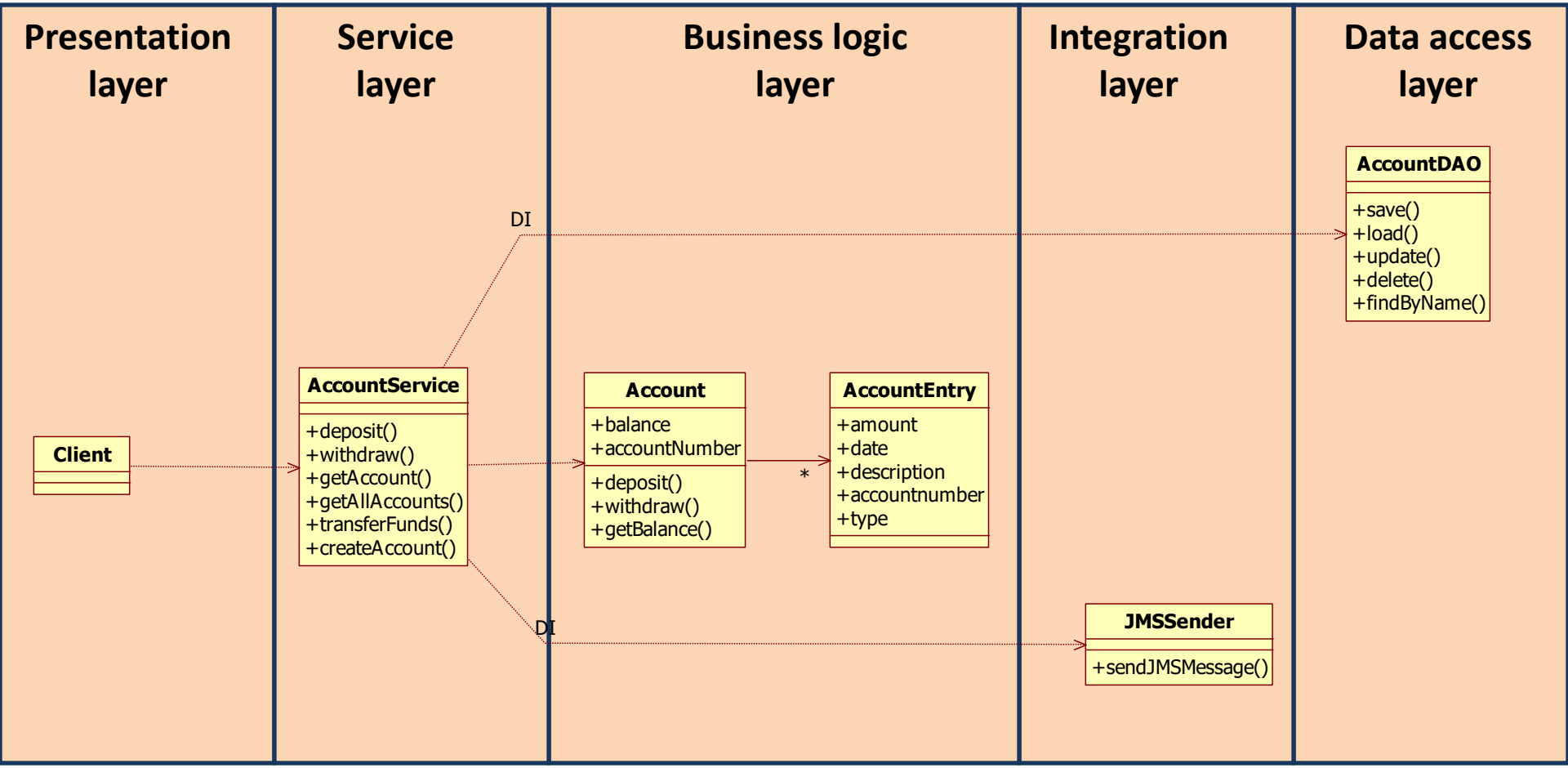
NOT OK

Black box layering

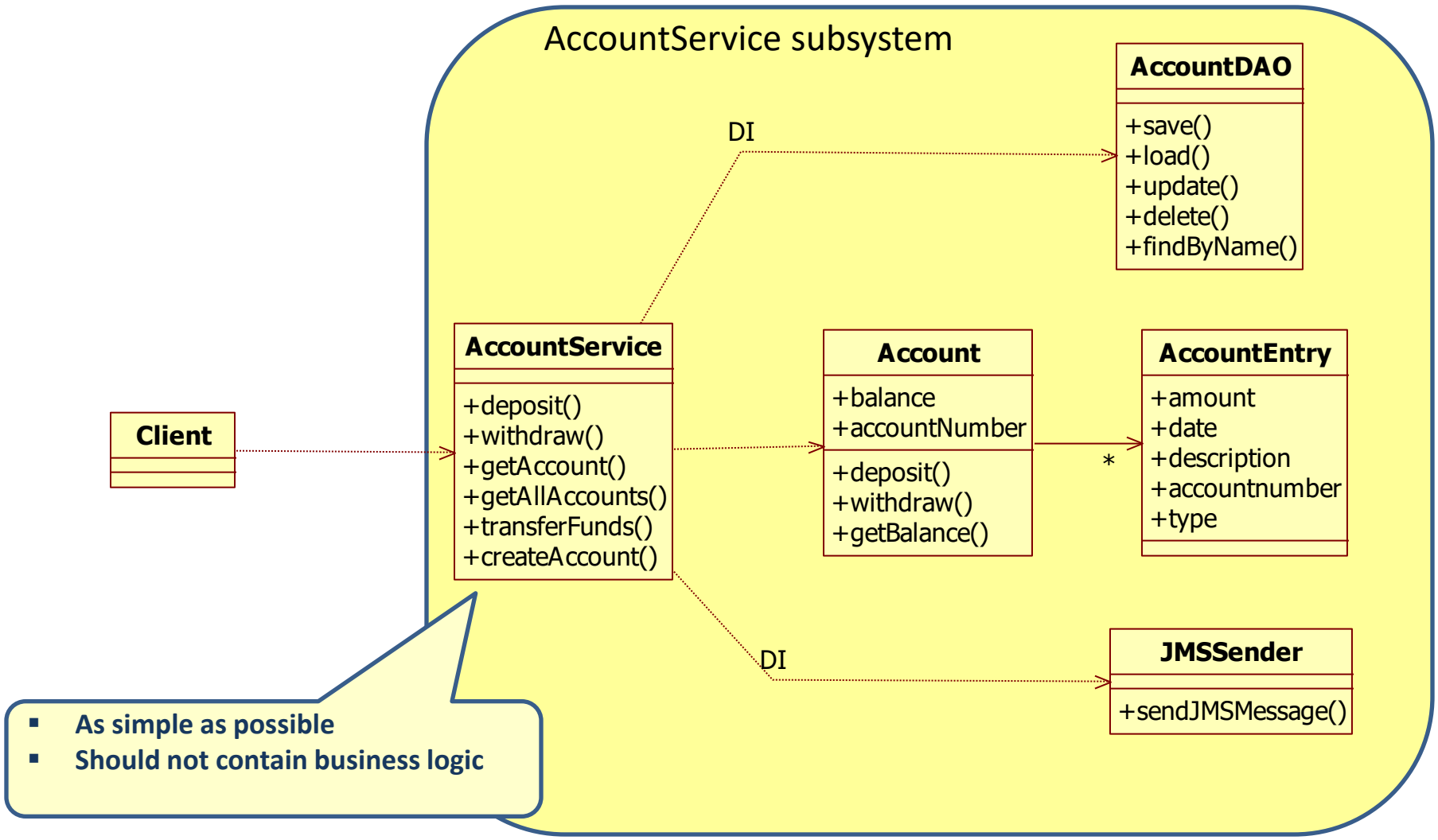


OK

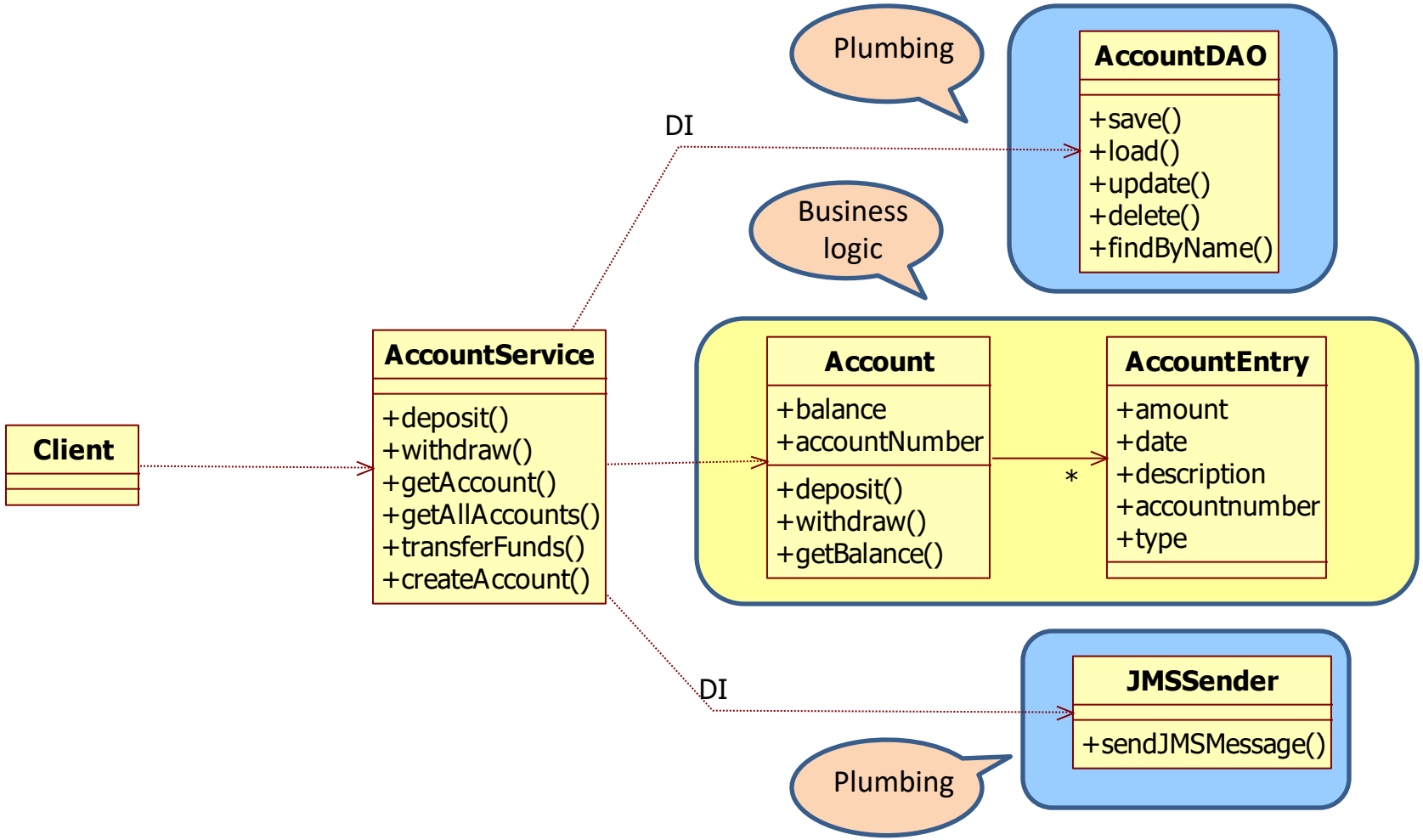
Application layers



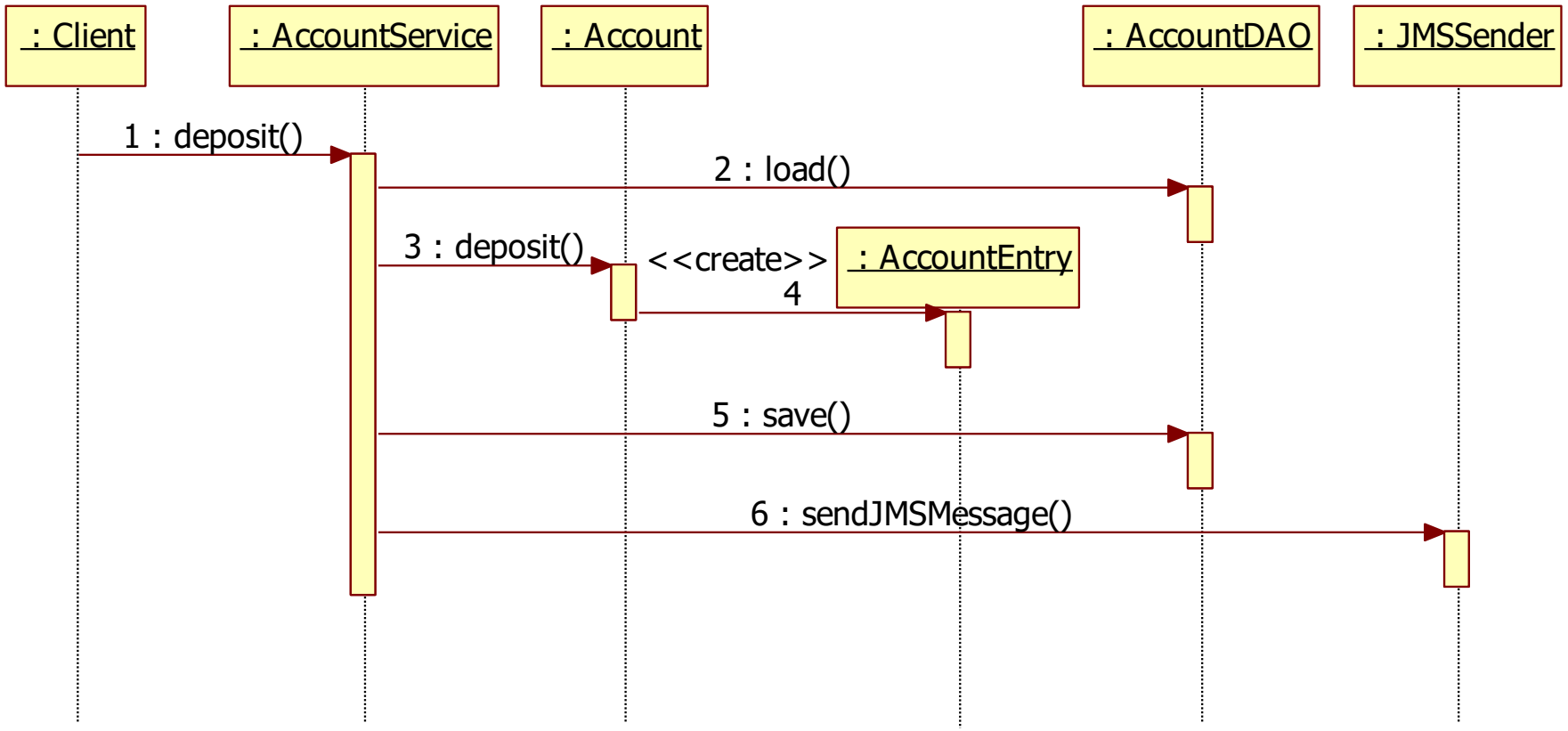
Entry of a complex subsystem



Separation of concern



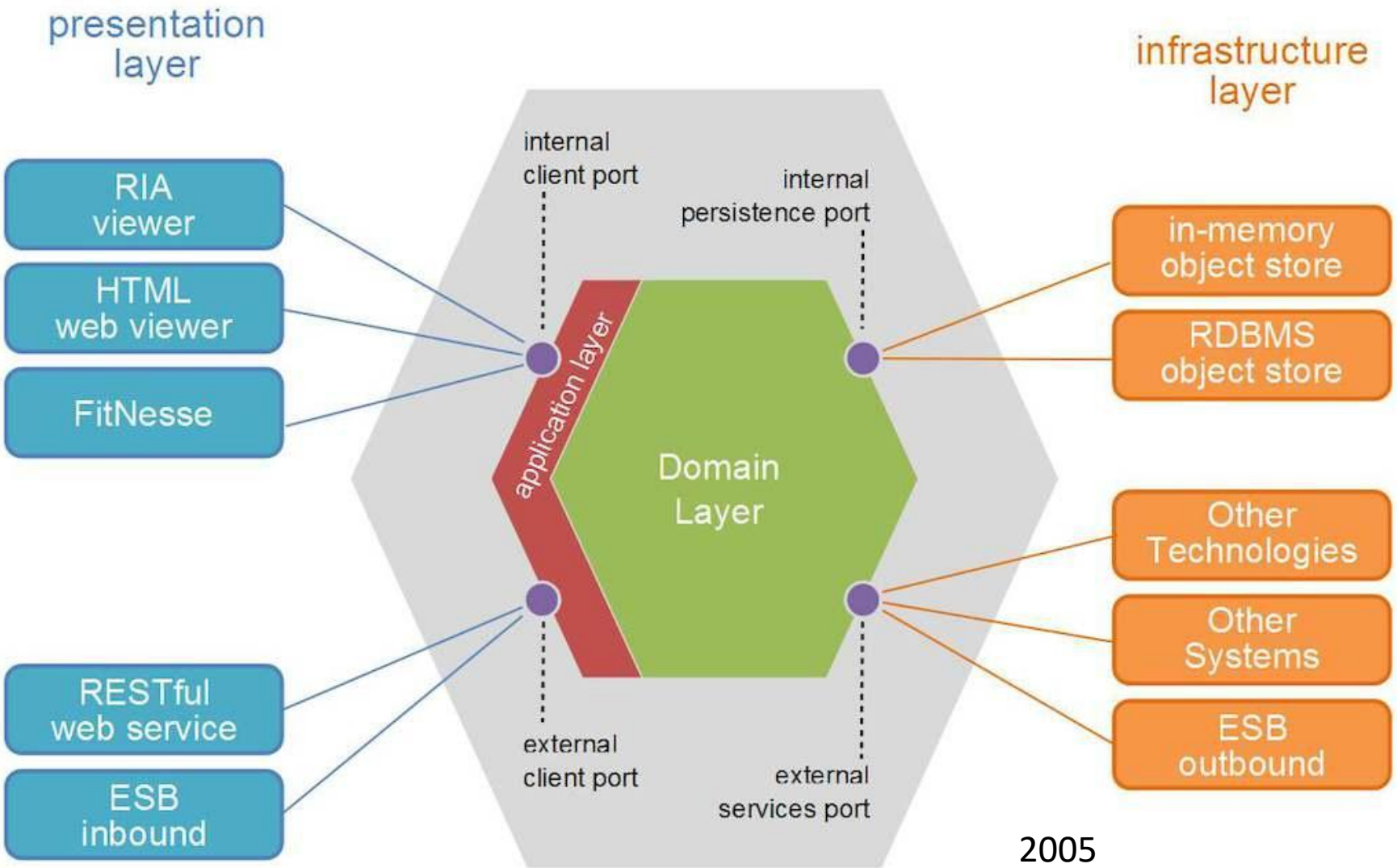
Service object



Main point

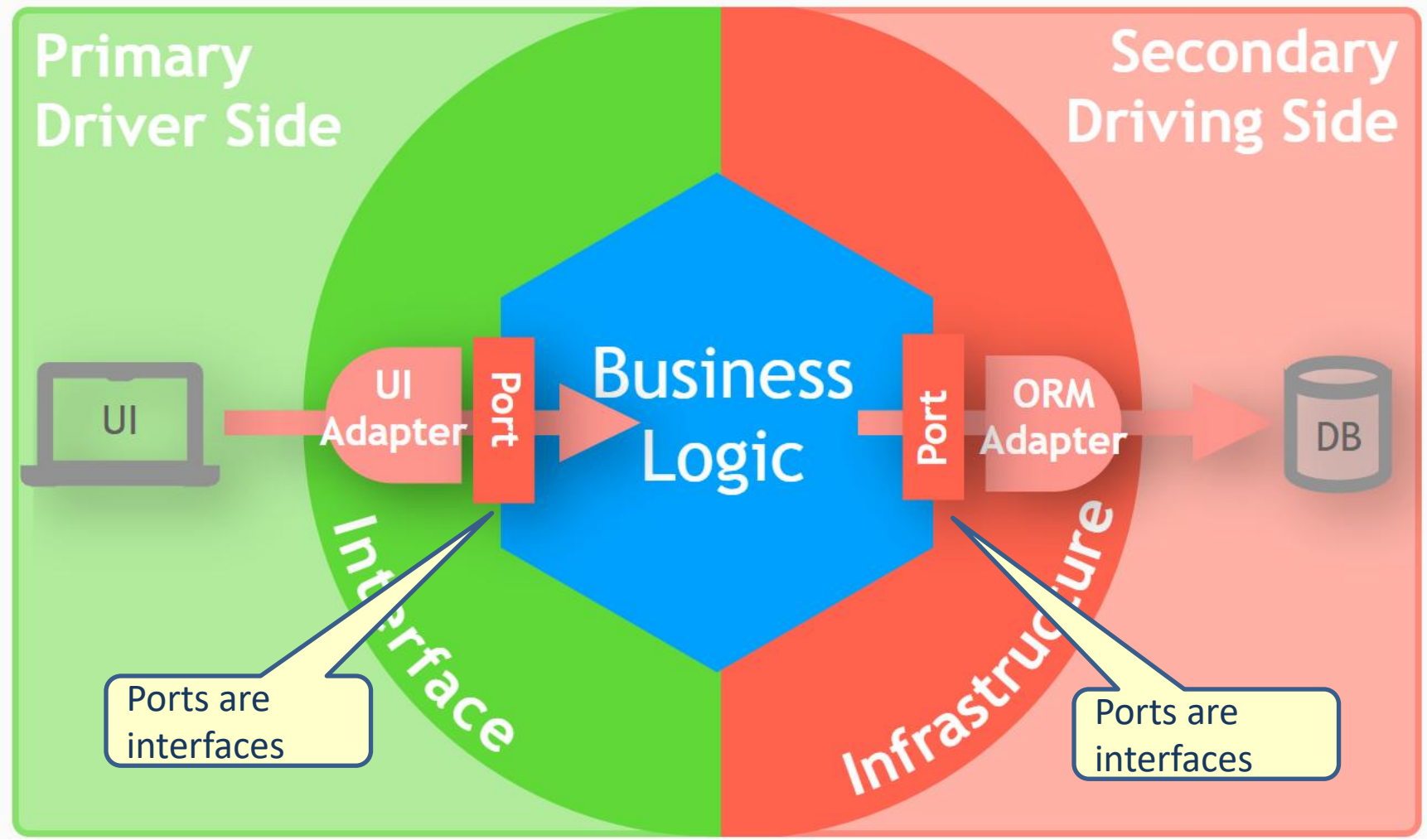
- An enterprise back-end system is typically divided in different layers.
- Life is found in layers

Hexagonal architecture

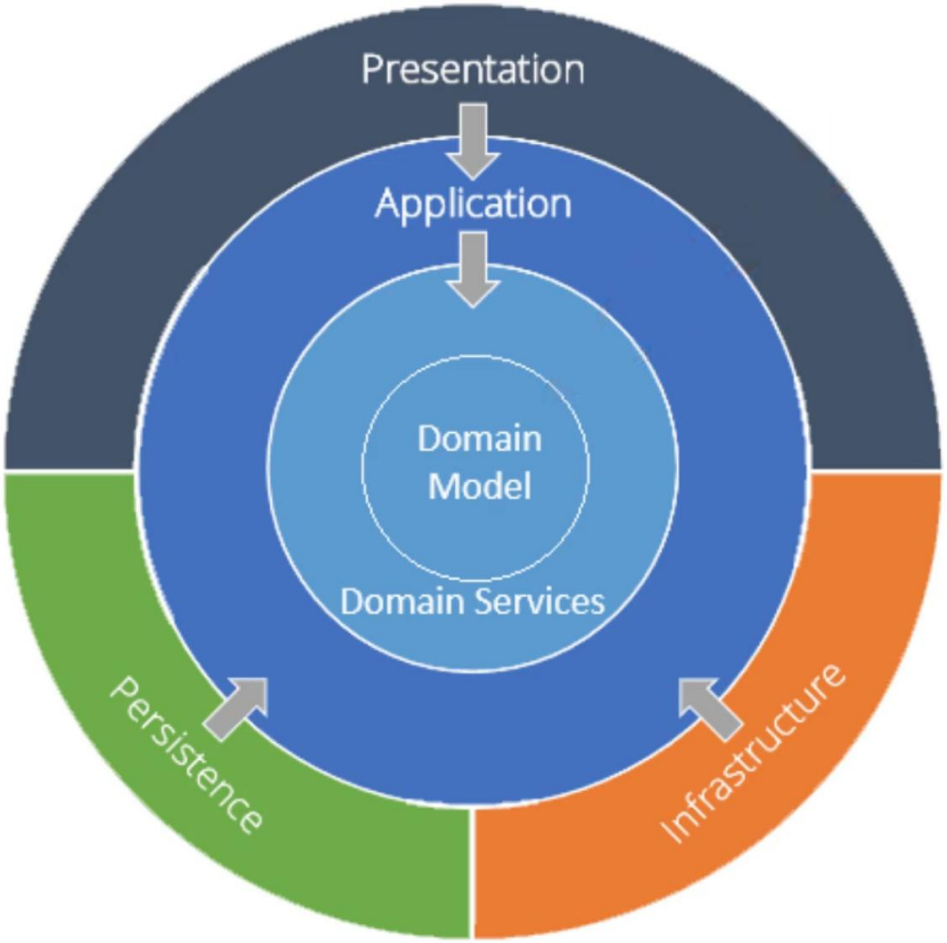


2005

Port and adapter architecture

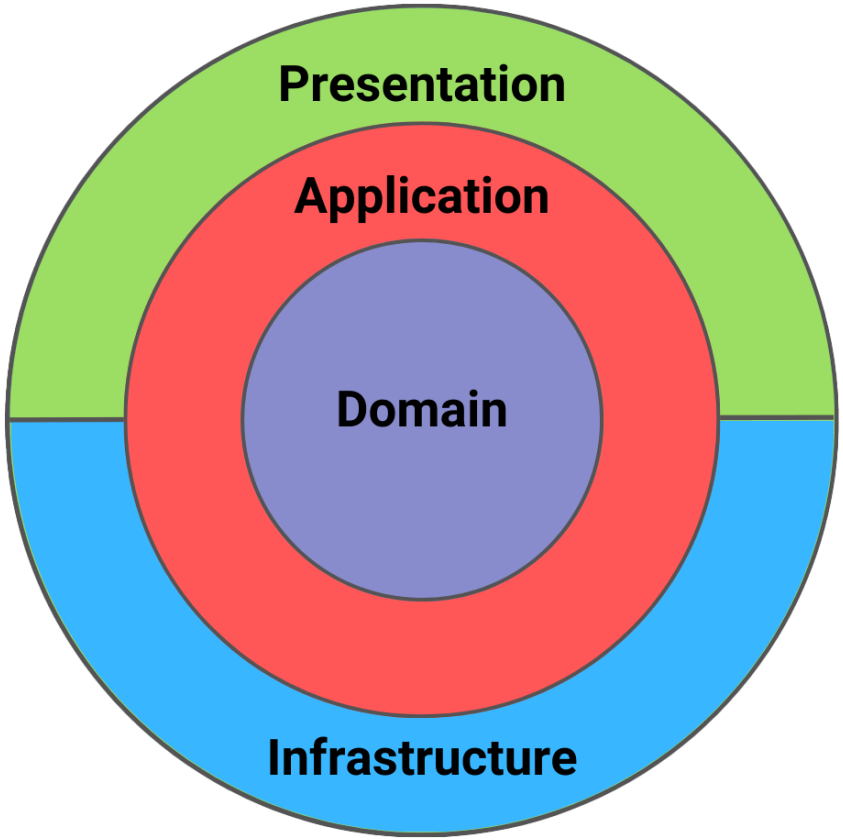


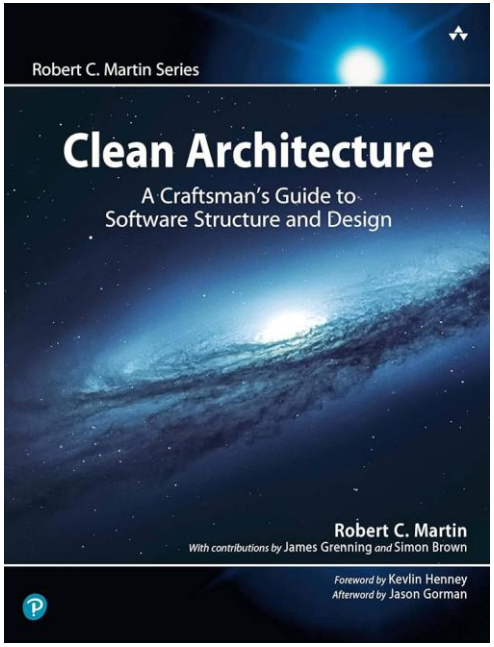
Union architecture



2008

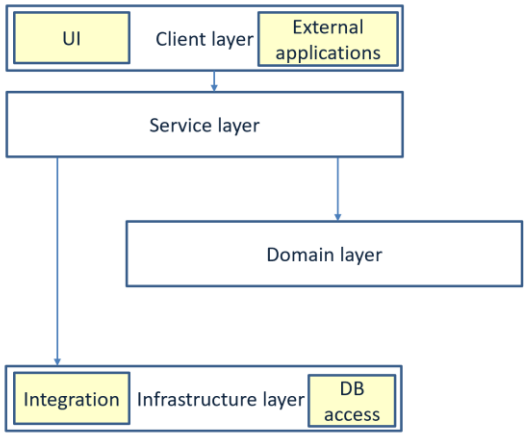
Clean architecture

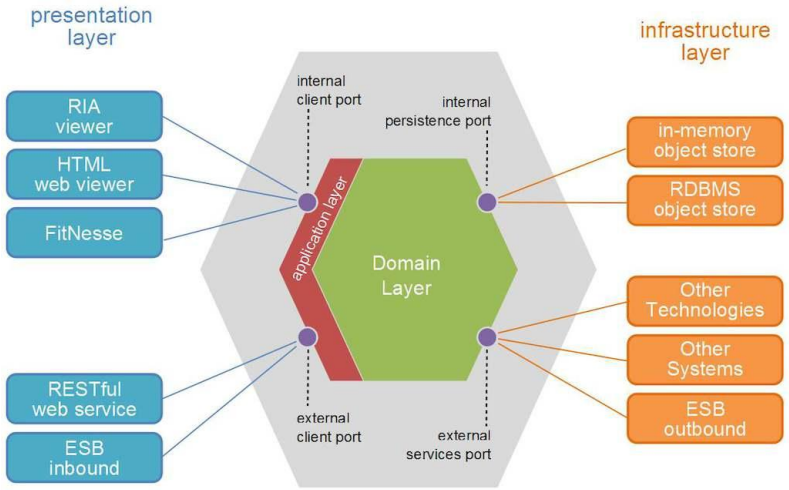




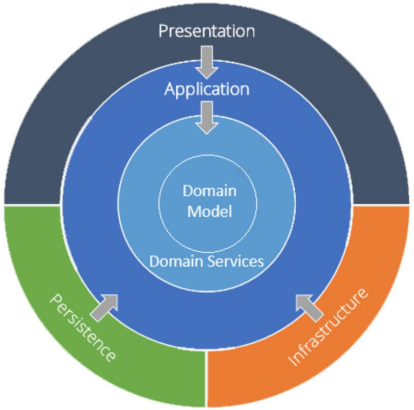
2017

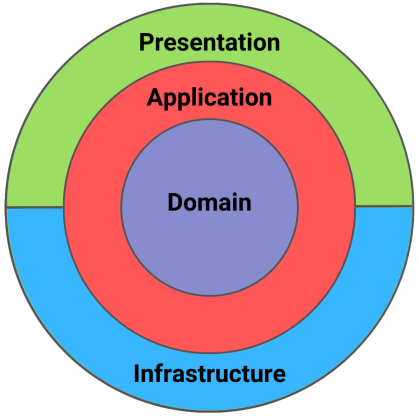
Layered, hexagonal, onion and clean architecture



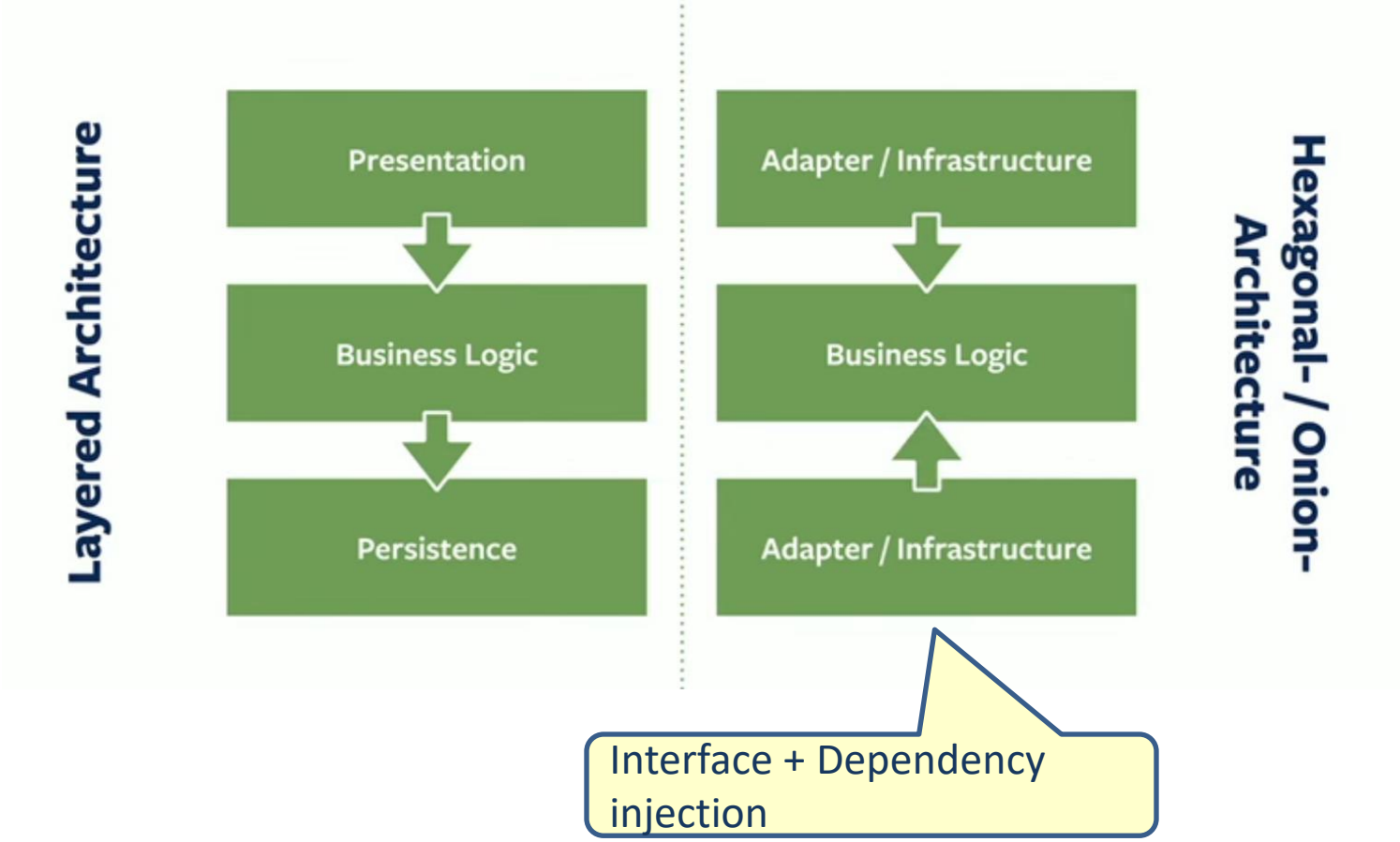


Domain is the core





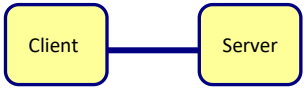
Layered, hexagonal, onion and clean architecture



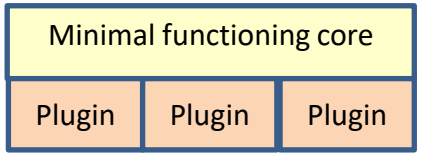
Main point

- The domain classes are never aware of technical “plumbing” classes. This gives many different advantages.
- By diving deep into pure consciousness, one gains support of all the laws of nature without needing to know or to be aware of all different details of your life and your world.

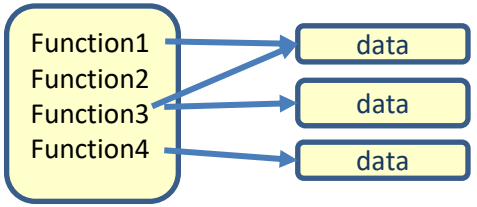
Architecture styles within an application



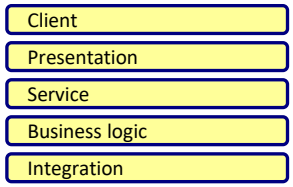
Client-server



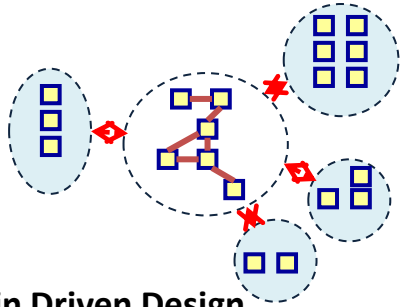
Microkernel



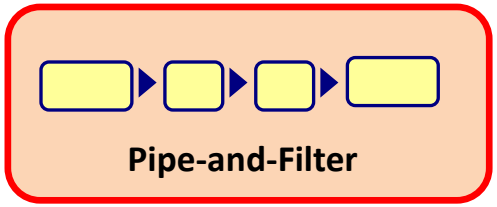
Procedural



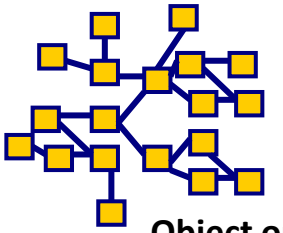
Layering



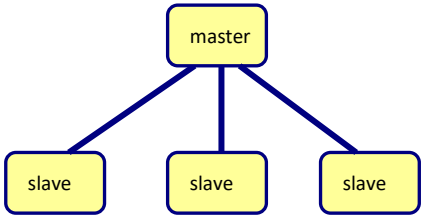
Domain Driven Design



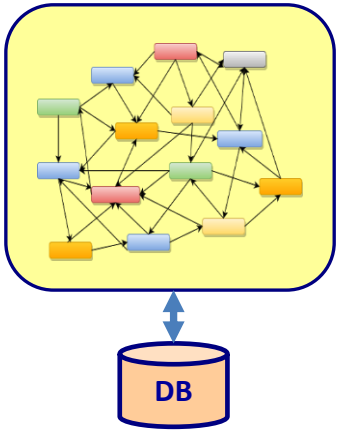
Pipe-and-Filter



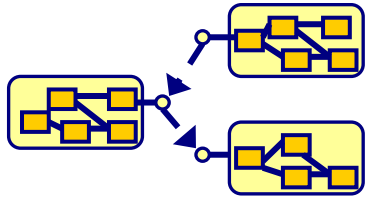
Object oriented



Master-Slave

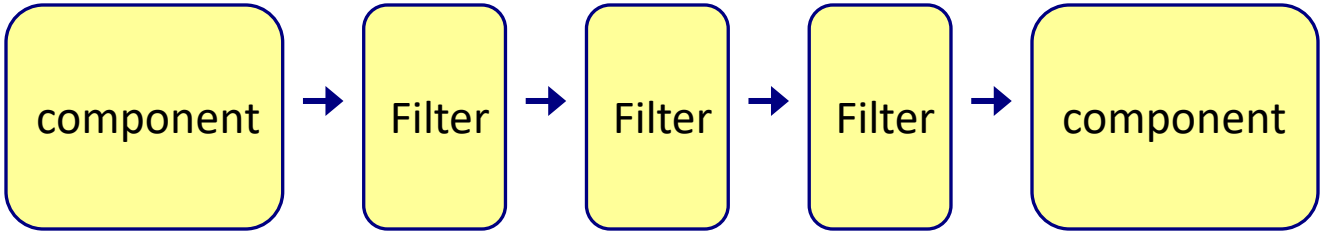


Monolith

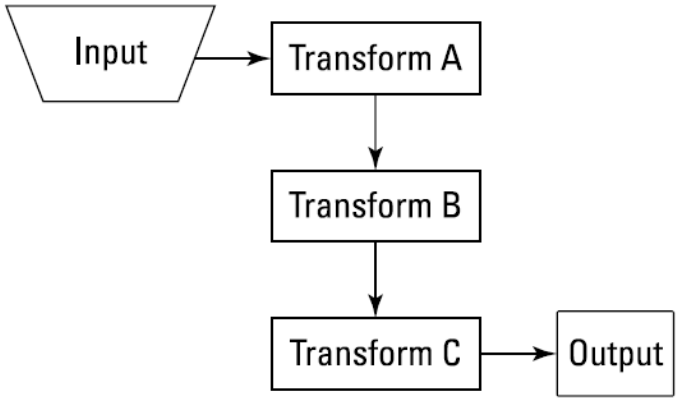


Component based

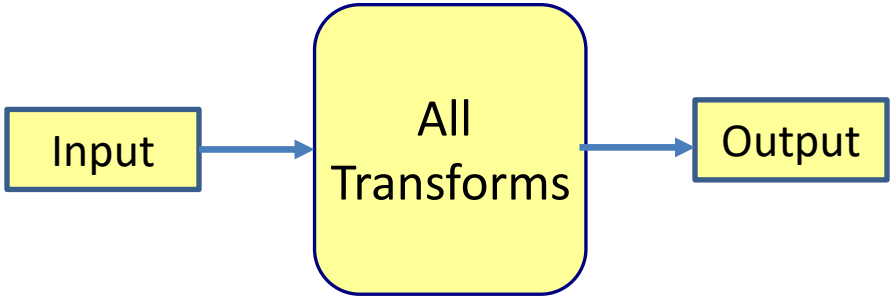
Pipe and Filter



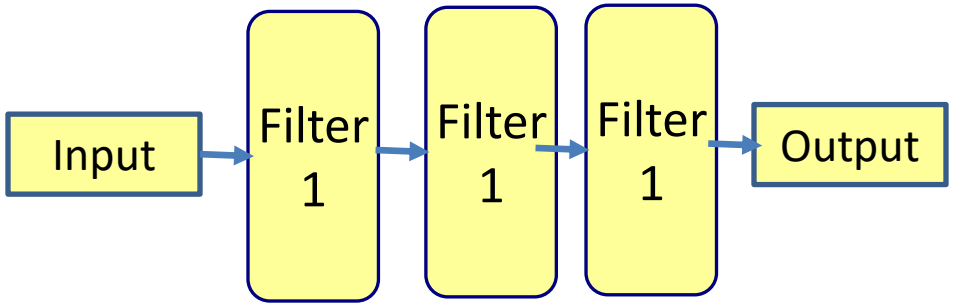
Analyzing an Image Stream



One big transformation



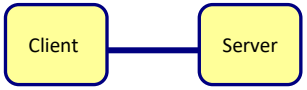
Pipe and Filter



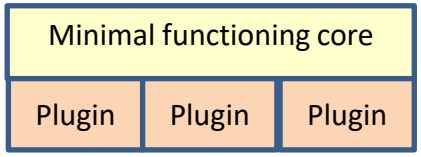
Pipe and Filter

- Benefits
  - Filters are independent
  - Filters are reusable
  - Order of filters can change
  - Easy to add new filters
  - Filters can work in parallel
- Drawbacks
  - Works only for sequential processing
  - Sharing state between filters is difficult

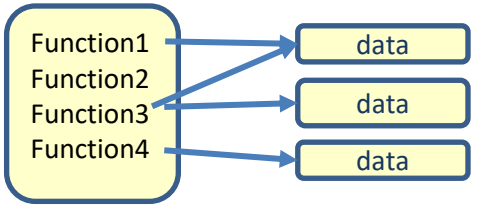
Architecture styles within an application



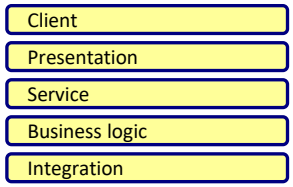
Client-server



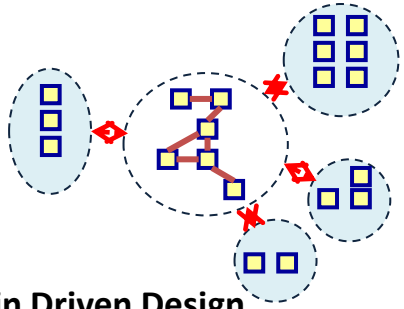
Microkernel



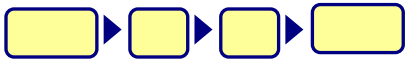
Procedural



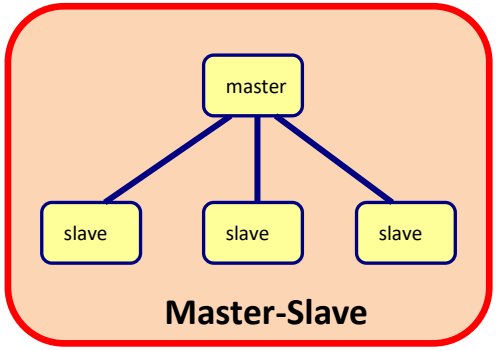
Layering



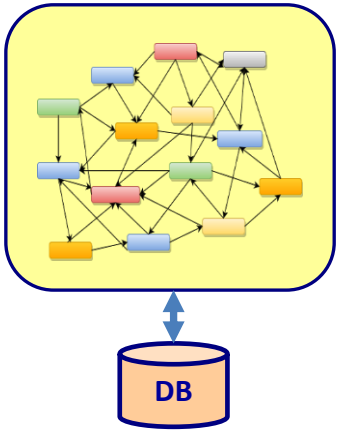
Domain Driven Design



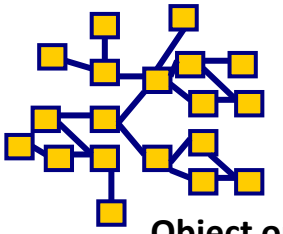
Pipe-and-Filter



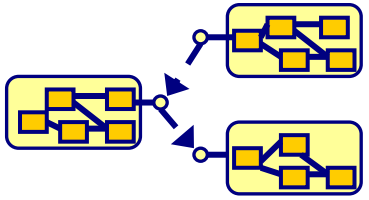
Master-Slave



Monolith



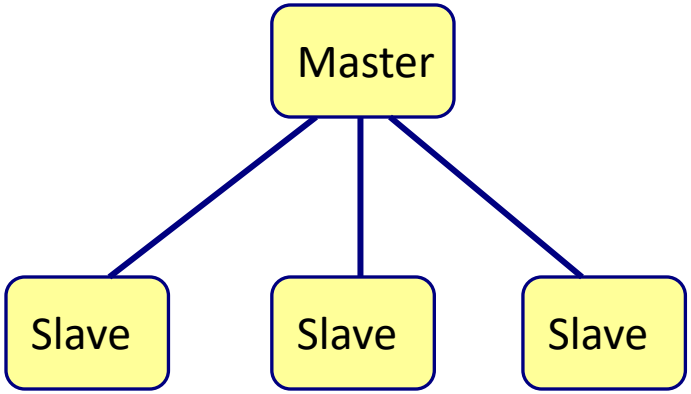
Object oriented



Component based

Master-Slave

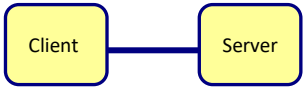
- Master organizes work into distinct subtasks
- Subtasks are allocated to isolated slaves
- Slaves report their result to the master
- Master integrates results



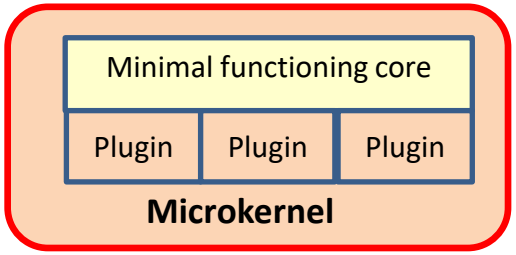
Master slave

- Benefits
  - Separation of coordination and actual work
  - Master has complete control
  - Slaves are independent
    - No shared state
  - Easy to add new slaves
  - Slaves can work in parallel
  - Slaves can be duplicated for fault tolerance
- Drawbacks
  - Problem must be decomposable
  - Master is single point of failure

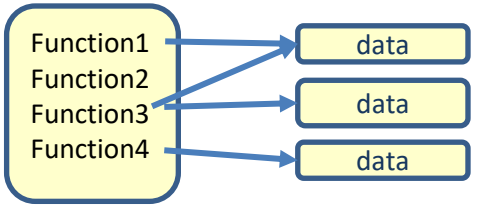
Architecture styles within an application



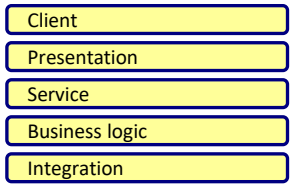
Client-server



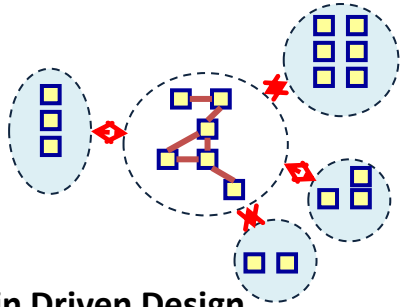
Microkernel



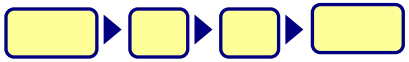
Procedural



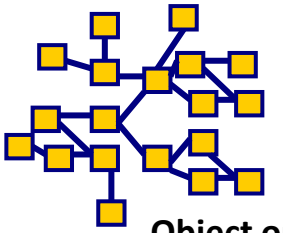
Layering



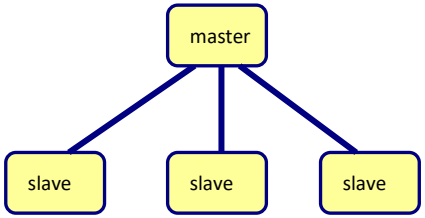
Domain Driven Design



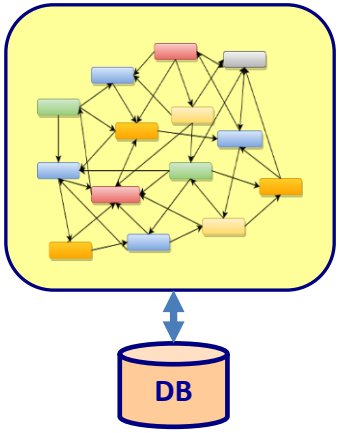
Pipe-and-Filter



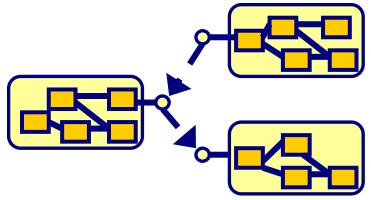
Object oriented



Master-Slave



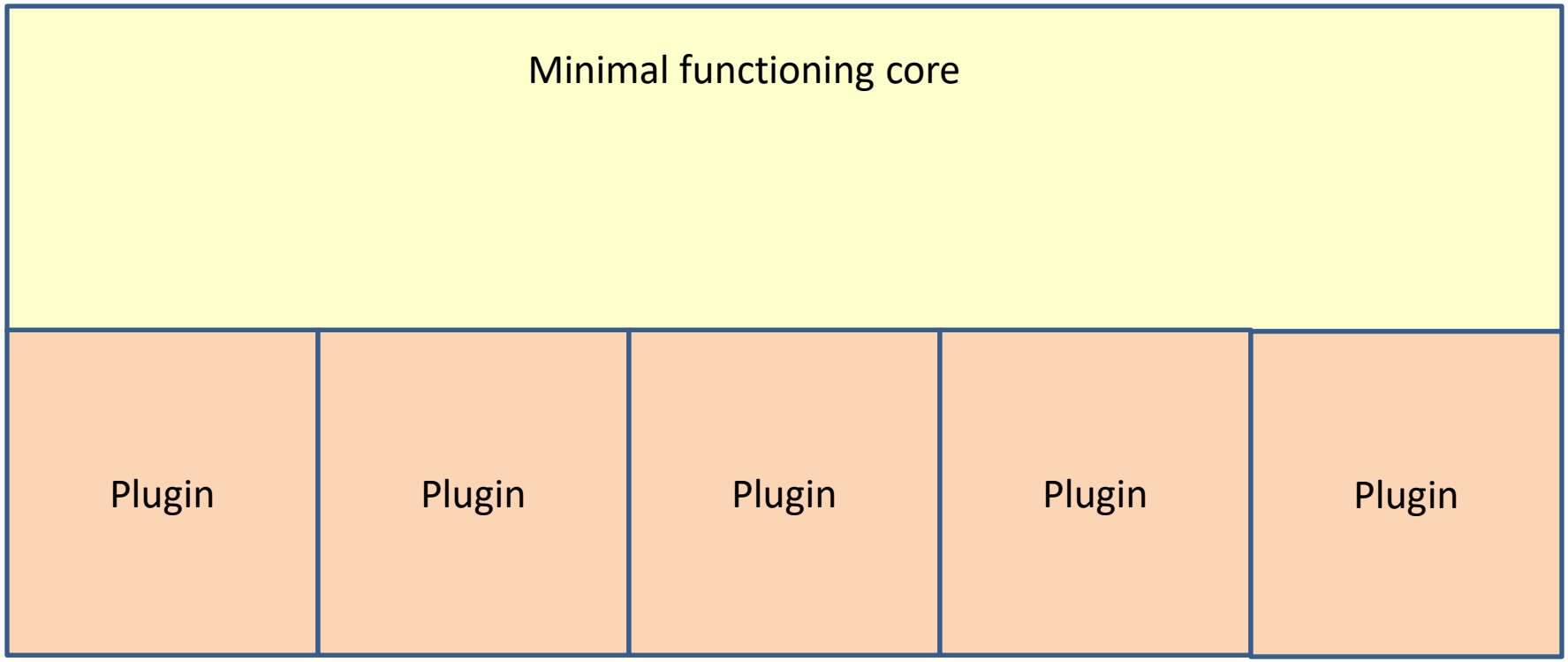
Monolith



Component based

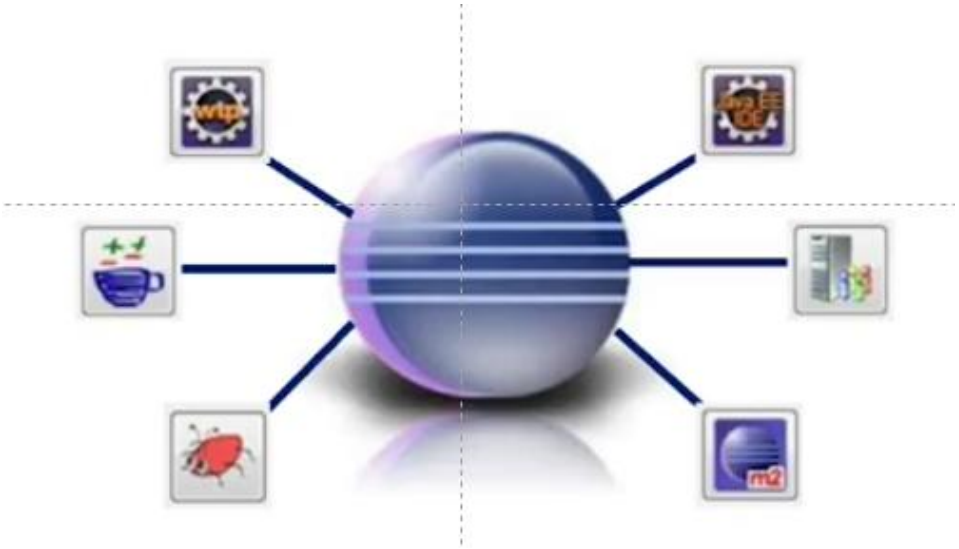
Microkernel

- Plugin application/framework



Microkernel examples

- Eclipse
  - With plugins



- Operating system
  - With drivers

Microkernal

- Benefits
  - Natural for product based apps
  - Extensibility
  - Flexibility
  - Separation of concern
- Drawbacks
  - Complexity

SUMMARY

What is software architecture?

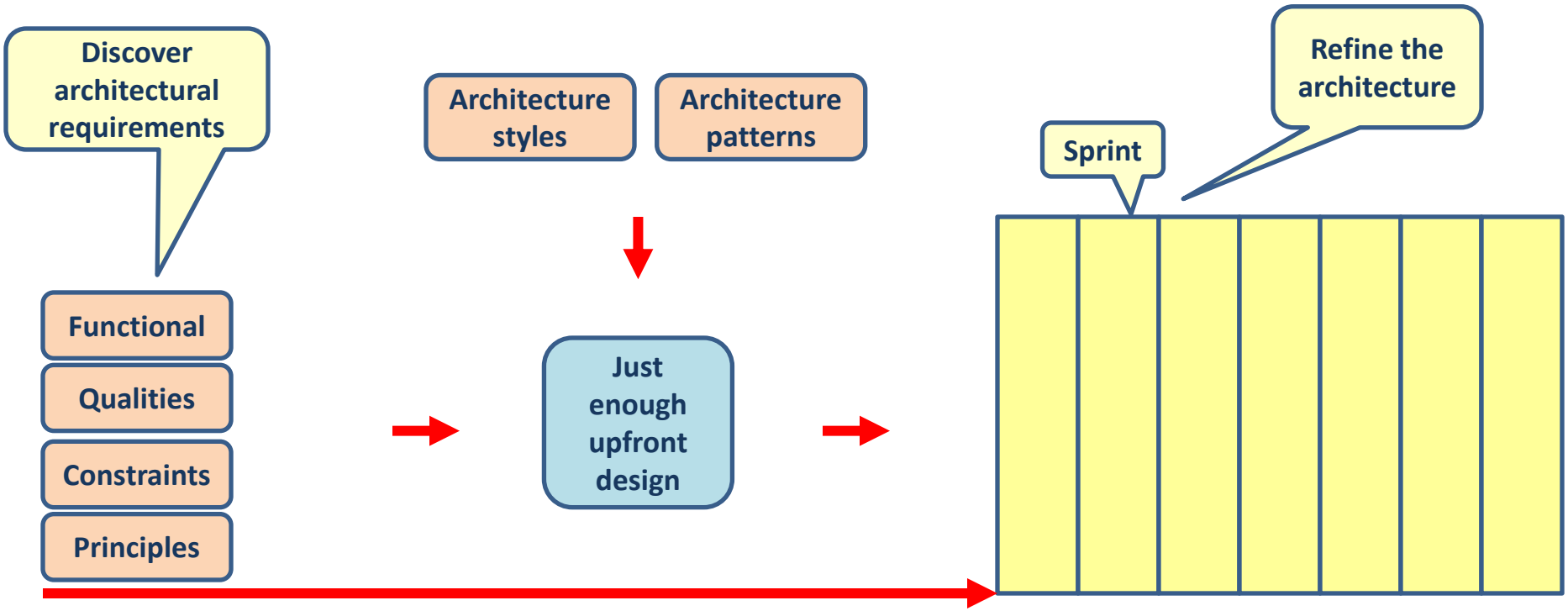
The important stuff

Whatever that might be

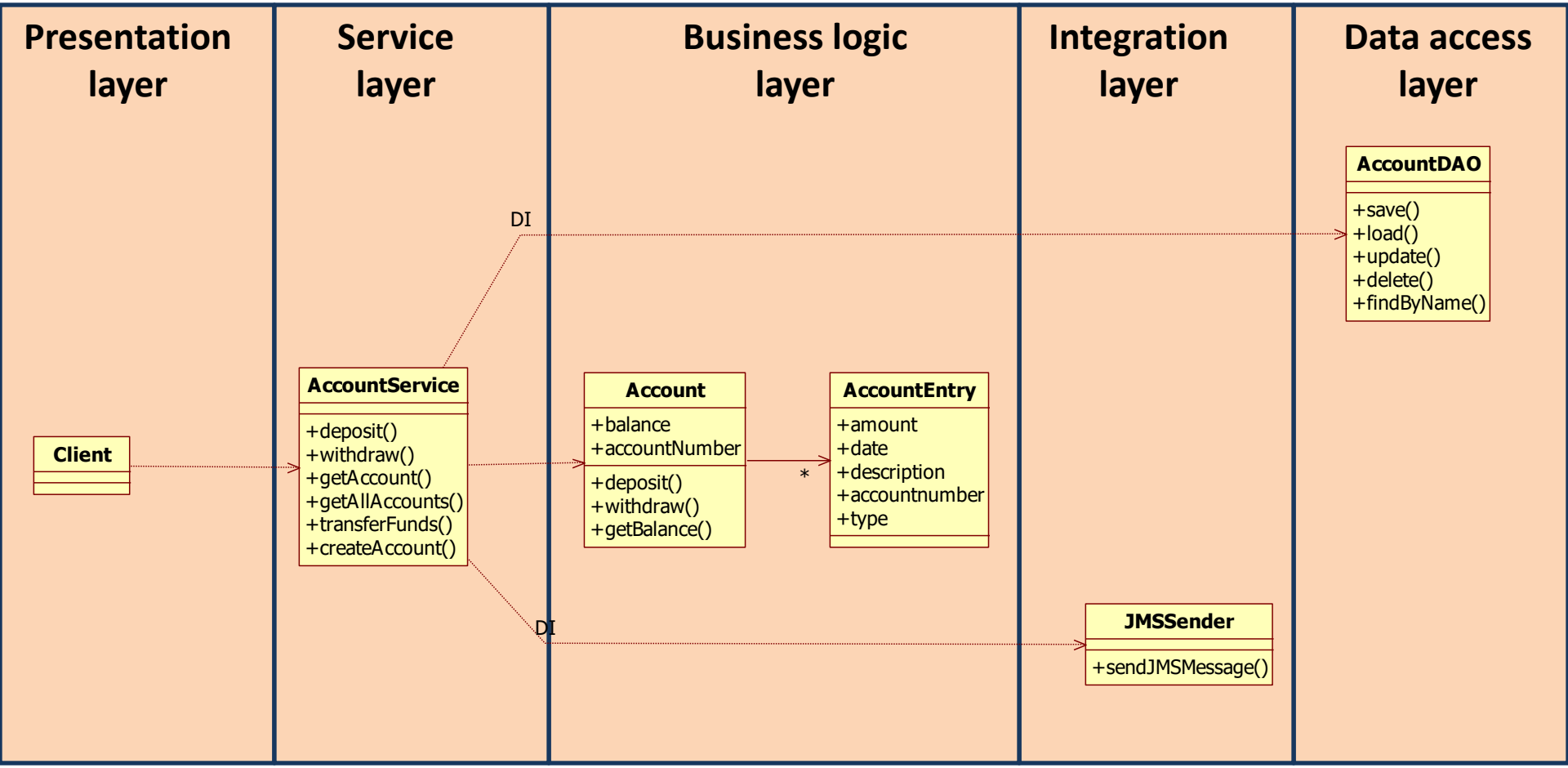
Agile architecture

- Just enough upfront architecture
  - Refine the architecture later
- Keep your architecture flexible
- The architect is available during the whole project
- The architect also writes code
  - But not all the time
  - The architecture is grounded in reality
  - Works together with the developers
  - Architects should be master builders
- Proof the architecture in the first iteration(s)

Agile software architecture



Application layers



SOFTWARE ARCHITECTURE KEY PRINCILES

Key principle 1

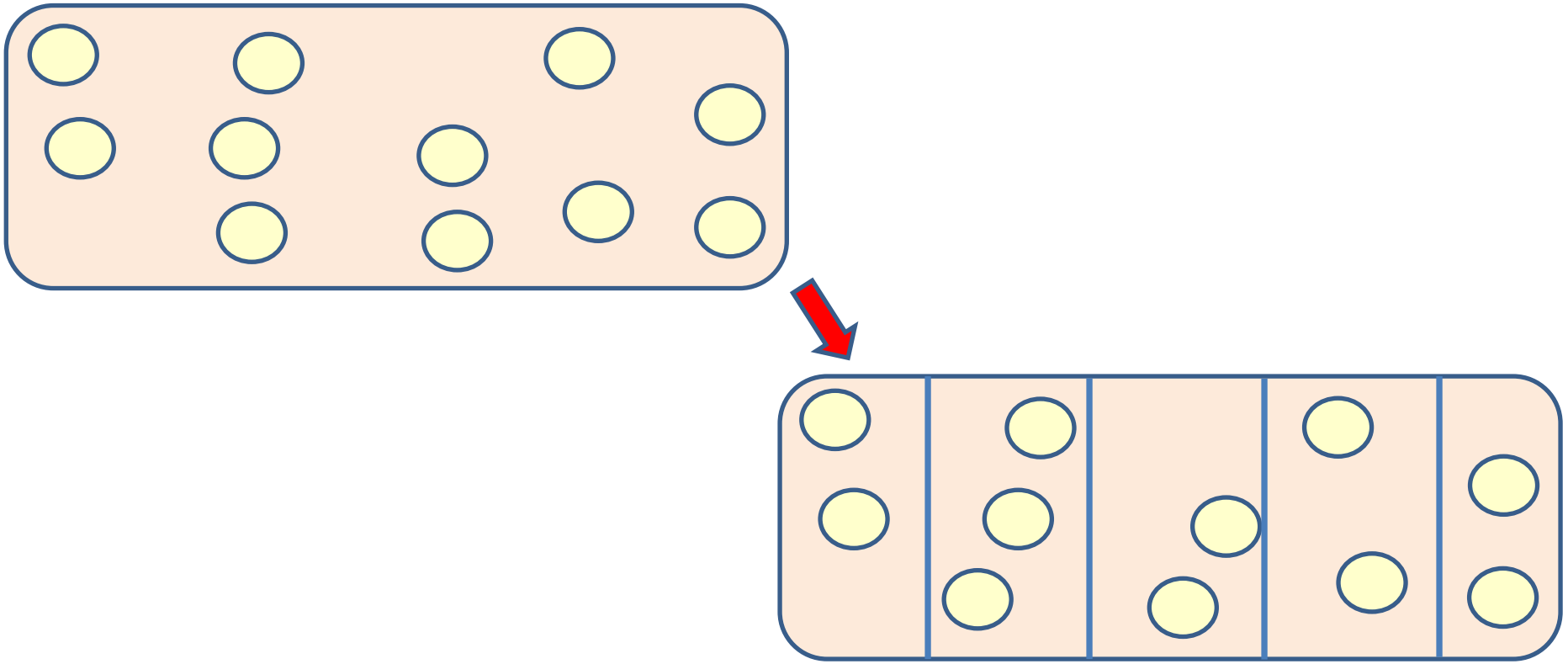
- Software architecture is about making tradeoffs
  - Every decision has advantages and disadvantages
    - There is no silver bullet
    - The architecture is never ideal
  - You cannot read this is a book
    - There is no fixed template you can follow
  - The answer is always: It depends

Key principle 2

- Most important architecture principles
  - Keep it simple
  - Keep it flexible
  - Loose coupling
    - High cohesion, low coupling
  - Separation of concern
  - Information hiding
  - Principle of modularity
  - Open-closed principle

Key principle 3

- When something becomes too complex, divide it into simpler parts



Connecting the parts of knowledge with the wholeness of knowledge

1. Software architecture defines all important aspects of a system.
2. The architecture decisions are based on the functional requirements, the qualities, the business constraints and the architecture principles

---

3. **Transcendental consciousness** is the natural experience of pure consciousness, the home of all the laws of nature.
4. **Wholeness moving within itself:** In unity consciousness, one appreciates and enjoys the underlying blissful nature of life even in all the abstract expressions of pure consciousness.