Avancier

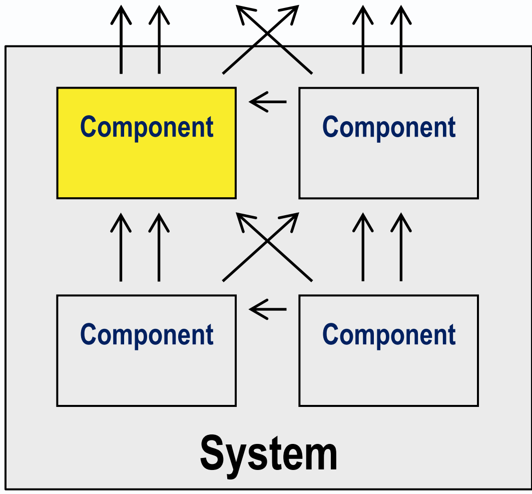
Avancier Methods
Conceptual framework – part two

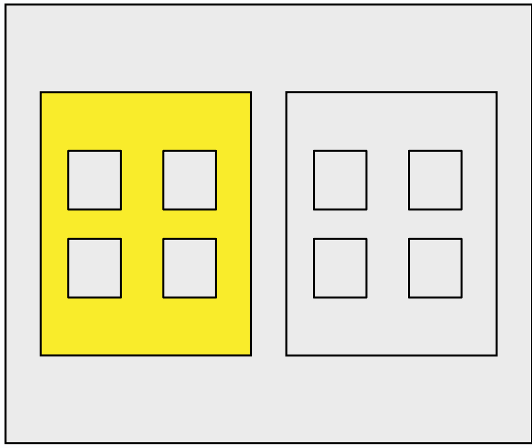
Component-based design
underpins SOA, enterprise and solution architecture

TOGAF says

Avancier

- "Systems are built from collections of building blocks"
  - In its operation, a human and/or computer activity system is a collection of actors and/or components cooperating in processes.
- “A building block is generally recognizable as "**a thing**" by domain experts”
  - Roles and components are *structural* elements rather than behavioural..
- “A building block may be assembled from other building blocks; may be a subassembly of other building blocks”
  - Roles and components are recursively composable and decomposable

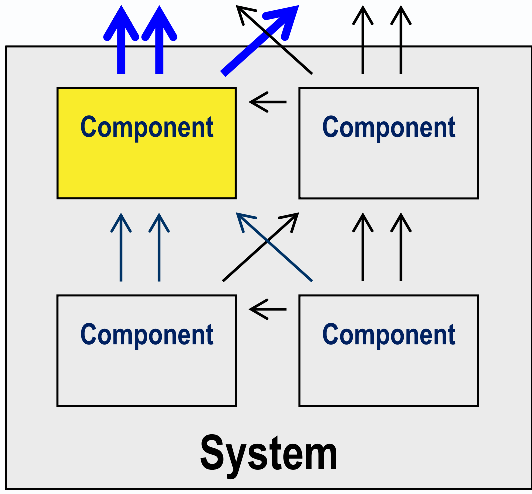




TOGAF building block = component in CBD

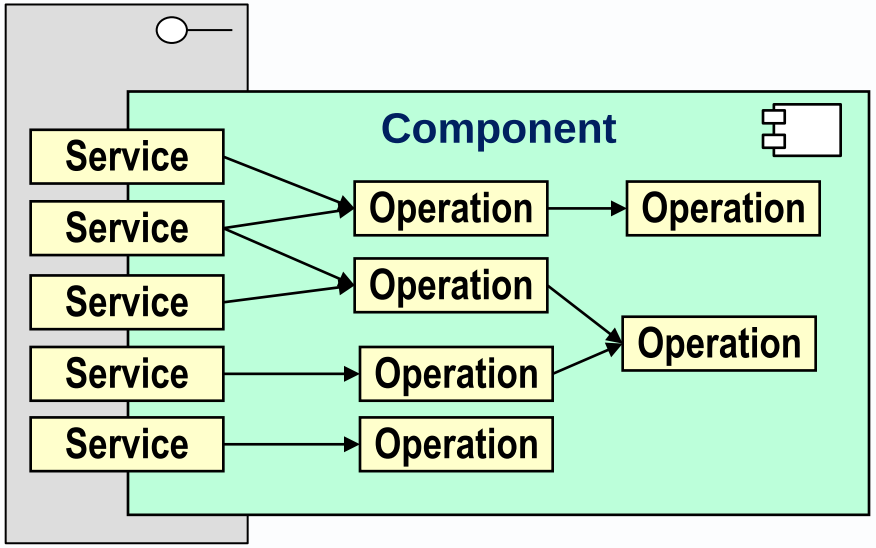
- “is a package of functionality defined to meet the business needs across an organization.”
  - A component type groups *behaviours* performable by an actor or component instance, that is, groups services requested of it and/or processes performed by it

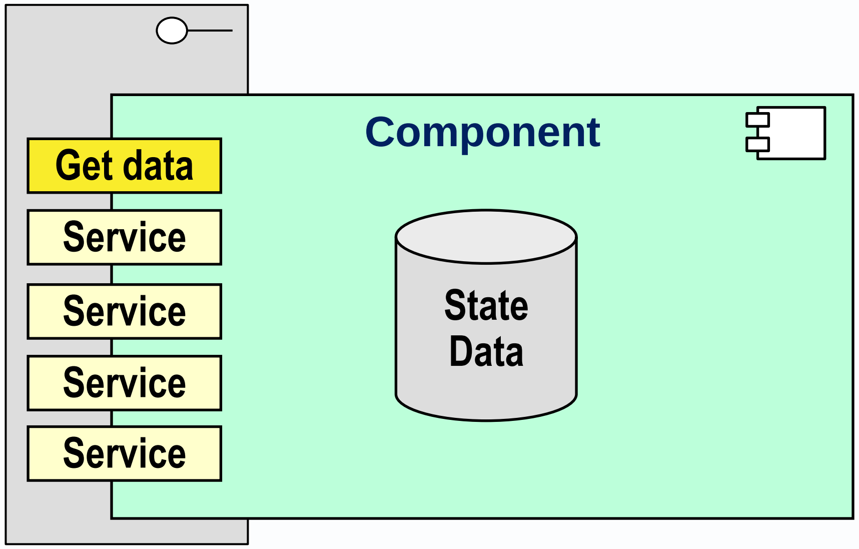
- A component is sometimes called
  - A performer (DoDAF)
  - An actor, function or role



TOGAF building block = component in CBD

- “has a defined boundary”
  - is *encapsulatable* behind an interface specification.
- A component
  - encloses **processes**, meaning that its inner workings are invisible to outsiders.
  - encloses **data**, so the only way to access that data is by using the interface of the component.

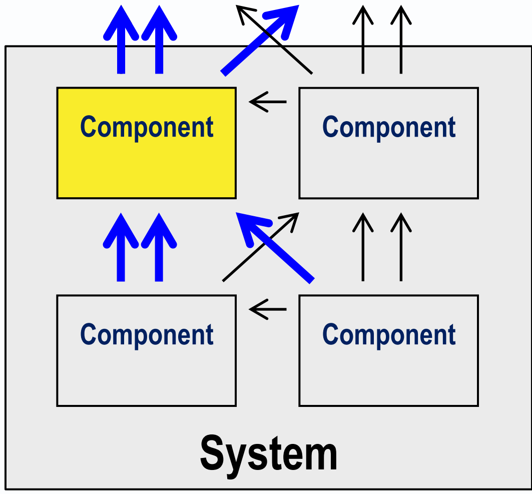




TOGAF building block = component in CBD

- “may interoperate with other, inter-dependent, building blocks.”
  - Is related to other components by requesting or delivering services.

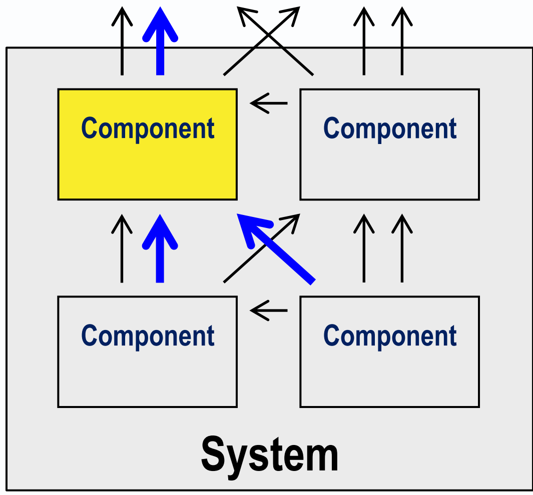
- In CBD, a component
  - interoperates with other components
  - provides services through published interfaces
  - requires services of other components
  - The interface offered by one component should match the interface required by another



Rationalisation by de-duplication of service provision

- We don’t want two components to provide the same service, so the principles are

- 1 component <offers> 1 or more services
- 1 service <is assigned to> 1 component interface
  - As though that component does all the work
  - Though to complete the service may require other components

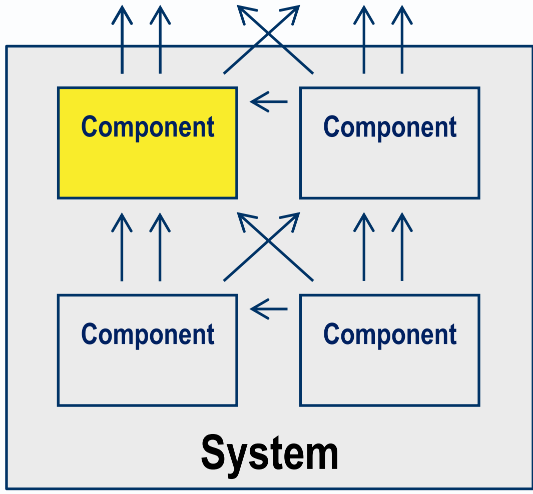


TOGAF building block = component in CBD

- “Ideally, is re-usable and replaceable, and well specified.
  - should offer generally useful services, and be free of implementation detail

- In CBD, the ideal component is
  - re-usable by several clients
  - replaceable by any component with the same interface(s)
  - portable between platform environments
  - extensible with new services

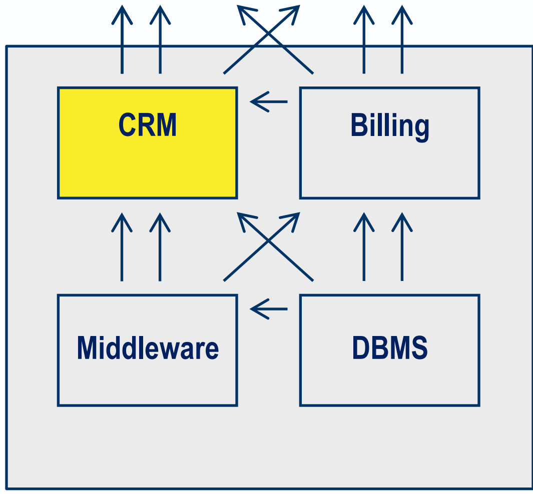
- Skilful componentisation should help system integration, reuse and agile development

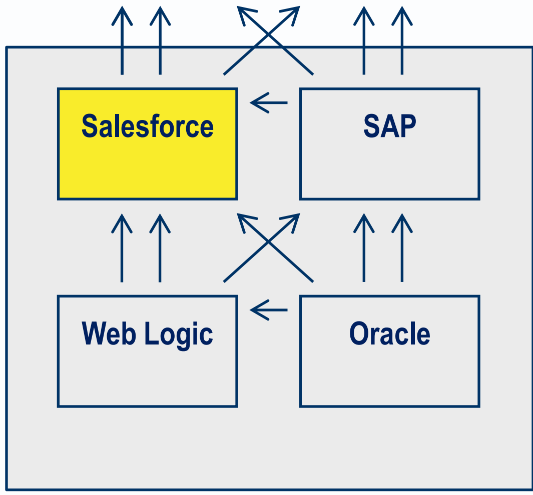


TOGAF building block = component in CBD

- "considers implementation and usage, and evolves to exploit technology and standards."

- Although components are specified logically in terms of services required, they can
  - be specified with particular technology-specific components in mind
  - be reverse-engineered





TOGAF building block = component in CBD

Avancier

TOGAF Building Block	CBD Component
“is generally recognizable as "a thing" by domain experts”	is a *structural* element rather than behavioural.
“may be assembled from other building blocks; may be a subassembly of other building blocks”	is recursively composable and decomposable.
“is a package of functionality defined to meet the business needs across an organization.”	groups *behaviours* performable by an actor or component instance, that is, groups services requested of it and/or processes performed by it.
“has a defined boundary”	is *encapsulatable* behind an interface specification.
“may interoperate with other, inter-dependent, building blocks.”	is related to other components by requesting or delivering services.
“Ideally, is re-usable and replaceable, and well specified.”	should offer generally useful services via an interface that is free of implementation detail.
“considers implementation and usage, and evolves to exploit technology and standards.	can be specified with particular technology-specific components in mind, or reverse-engineered from them.

Our kind of component

Avancier

- is a package of capability defined to meet client needs
- provides services through published interfaces
- may interoperate with other components
- may be assembled from other components
- may be placed in a client or server layer (later)

- is ideally
  - **re-usable** by several clients
  - **replaceable** by any component with the same interface(s)
  - **portable** between platform environments
  - **extensible** with new services

Avancier Methods

Conceptual framework – part two

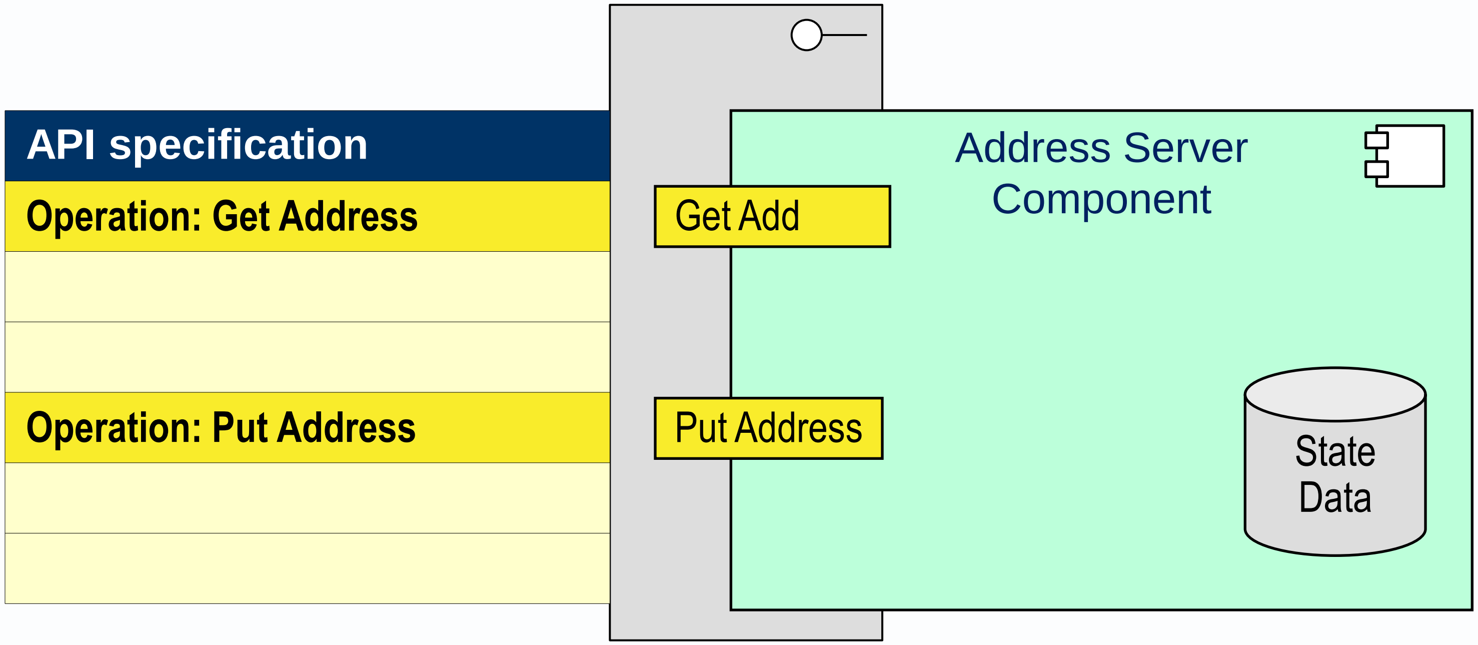
Component specification

underpins SOA, enterprise and solution architecture

Interface specification

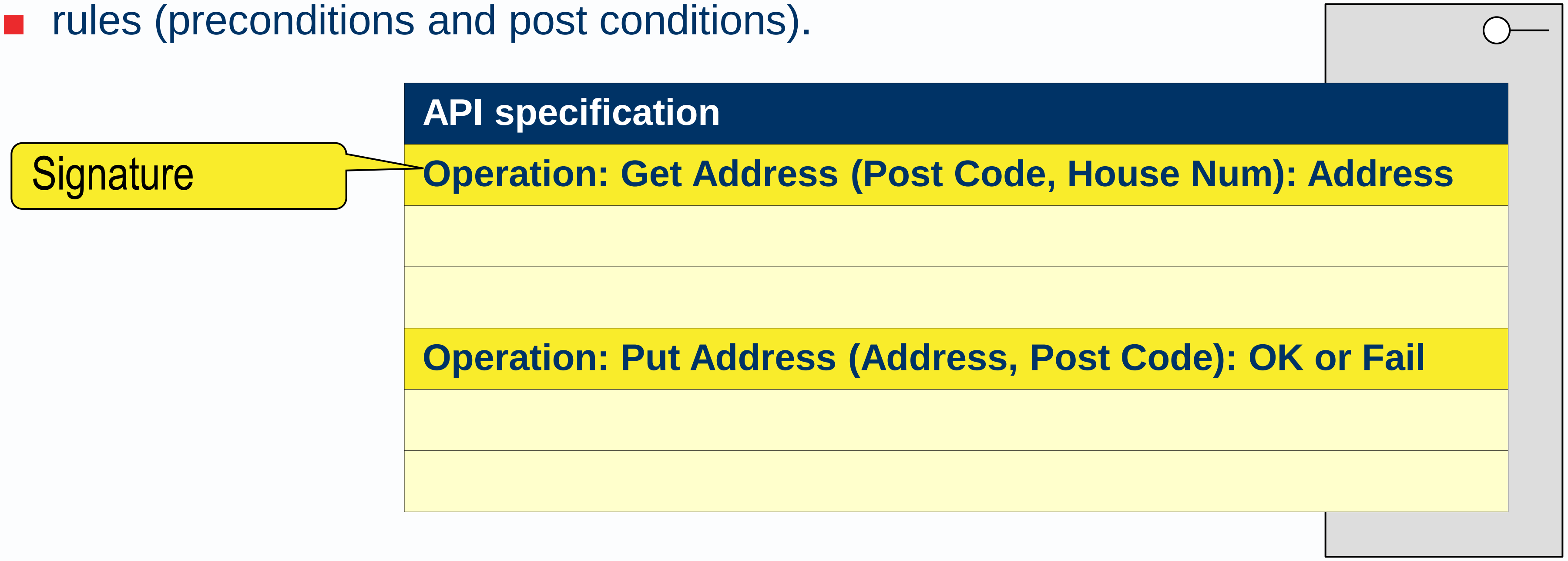
Avancier

- To encapsulate means hiding the internal data and procedures of a system (component or object) behind its interface.
- The interface is a collection of accessible services.



Operation signature

- Each operation is defined by its
  - signature (name, inputs and outputs) and
  - rules (preconditions and post conditions).

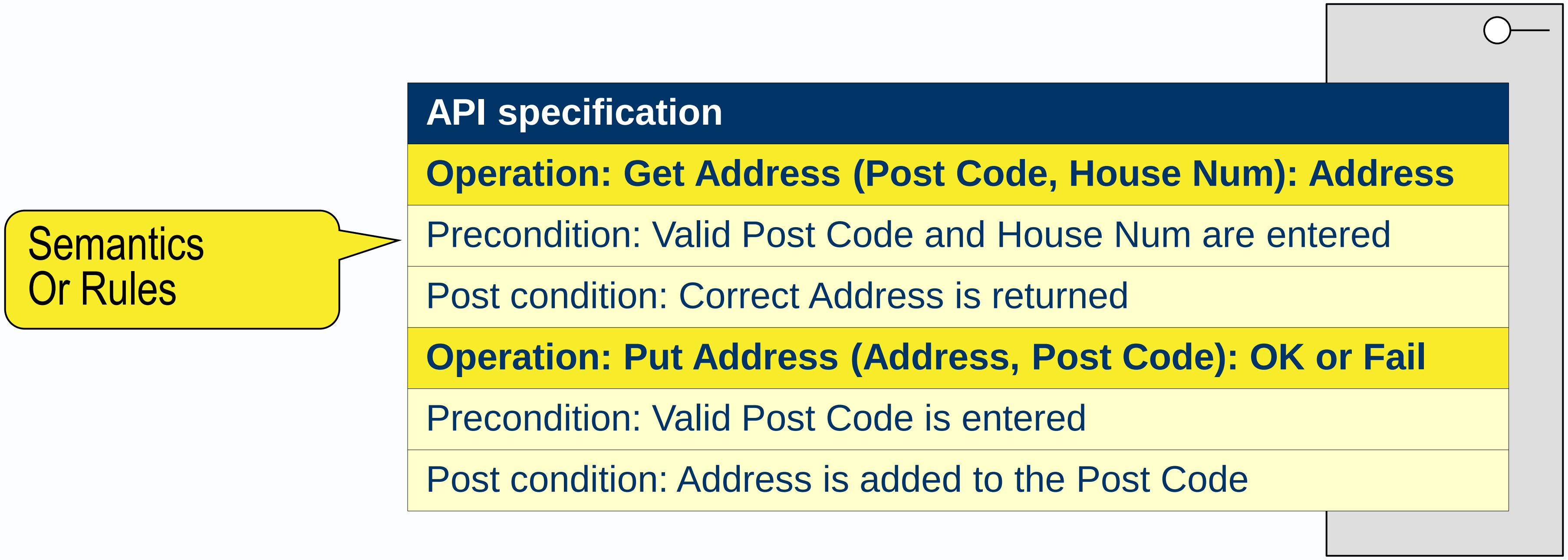

Signature
API specification
Operation: Get Address (Post Code, House Num): Address
Operation: Put Address (Address, Post Code): OK or Fail

- You can make it work using only the signature

Operation semantics or rules

Avancier

- In addition, a designer should understand, and ideally document, the preconditions and post conditions of an operation.



API specification
Operation: Get Address (Post Code, House Num): Address
Precondition: Valid Post Code and House Num are entered
Post condition: Correct Address is returned
Operation: Put Address (Address, Post Code): OK or Fail
Precondition: Valid Post Code is entered
Post condition: Address is added to the Post Code

- If the preconditions are true, and the operation proceeds to completion, then the post conditions will be true.

How well a service should work– the NFRs

Avancier

- 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.

API specification
Operation: Get Address (Post Code, House Num): Address
Precondition: Valid Post Code and House Name are entered
Post condition: Correct Address is returned
Operation: Put Address (Address, Post Code): OK or Fail
Precondition: Valid Post Code is entered
Post condition: Address is recorded against the Post Code
Non-functional characteristics – shared by services above
Response time = < 3 seconds
Throughput = 10 per second

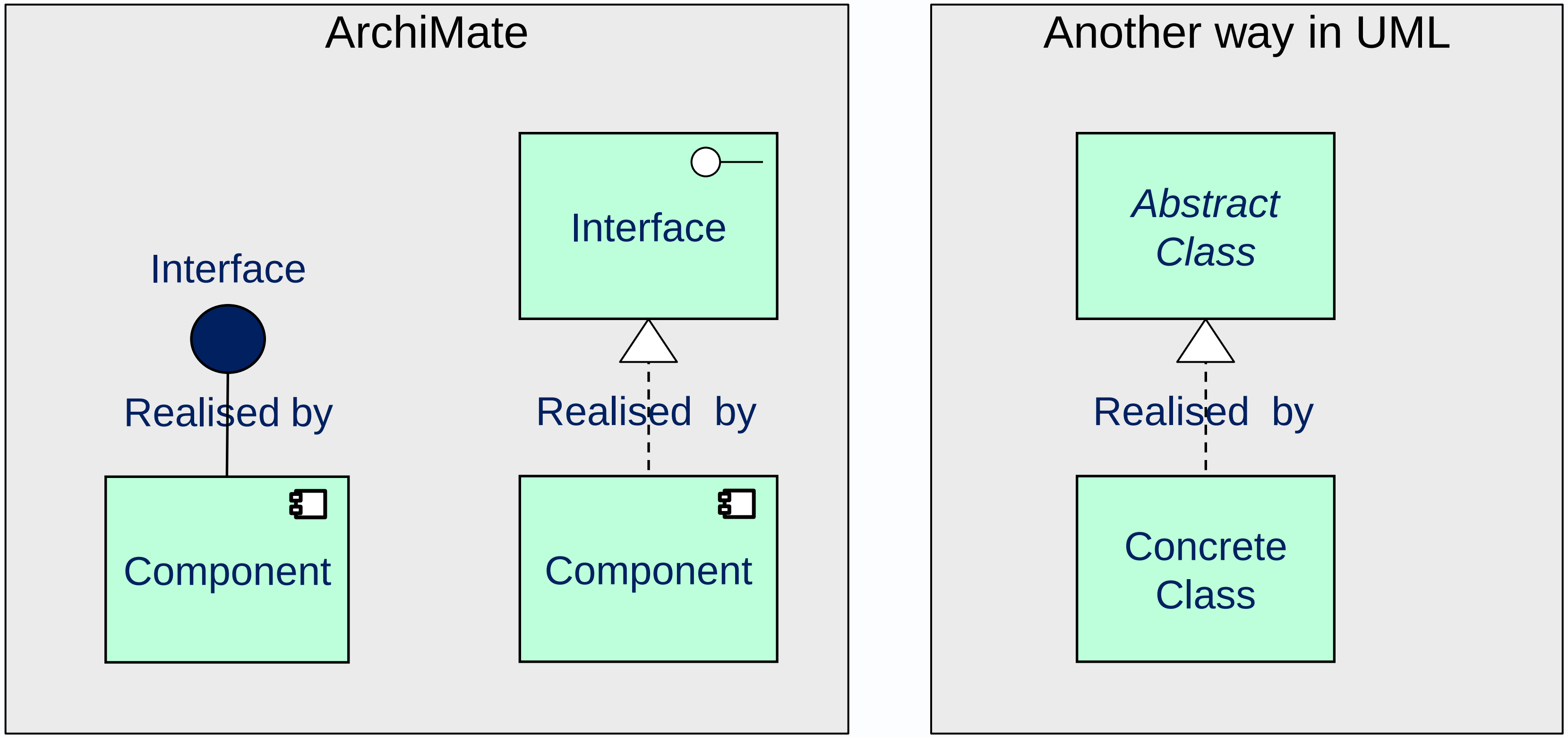
Services can share the same NFRs

Aside: on using preconditions and post conditions

- An operation’s preconditions and post conditions are important
- “Test-driven design” requires specification of preconditions and post conditions
- Some programming languages
  - include features for specifying preconditions and post conditions
  - enable the runtime system to verify them

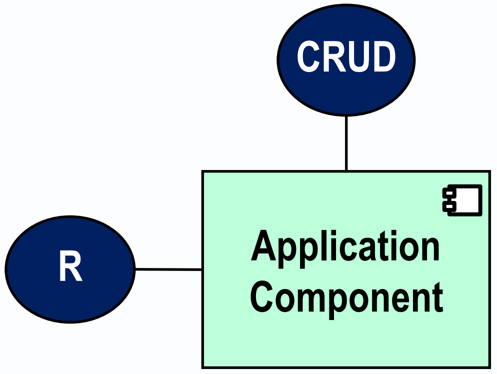
Realisation

- The act of providing implementations for services accessible via an interface.
- May be represented using modelling notations as below



Separation of component, interface and service

- One component may realise more than one interface.
  - older and newer versions, or
  - full and restricted list of services.
- One service can appear in several interfaces. E.g.
  - 1 component
  - 2 interfaces
  - 4 services (one duplicated)



	Behaviour	Structure
External	App/IS Service	API
Internal		Application Component

Incremental component specification

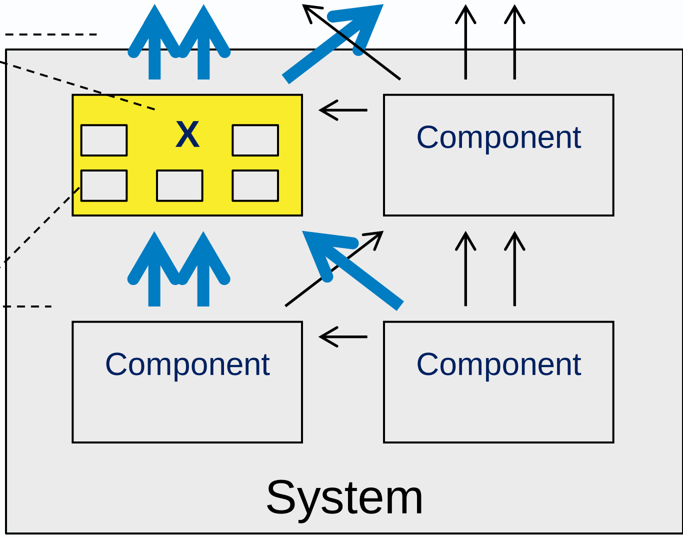
1. Name the component

2. Define provided services
   - Service name, inputs and outputs
   - Service pre and post conditions
   - Service non-functional characteristics

3. Define required services or components

4. Define required resources
   - Space, power, etc

5. Decompose into subcomponents if this is deemed "architecturally significant"



Avancier Methods

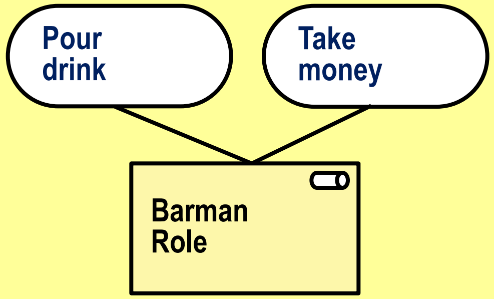
Conceptual framework – part two

Client-server layering

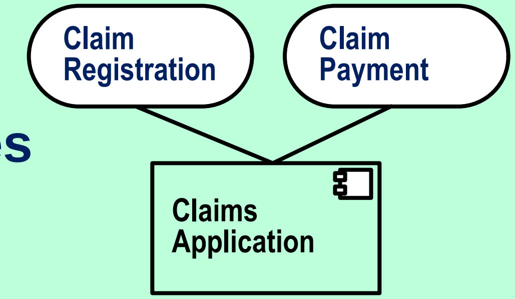
underpins SOA, enterprise and solution architecture

Components in different architecture domains

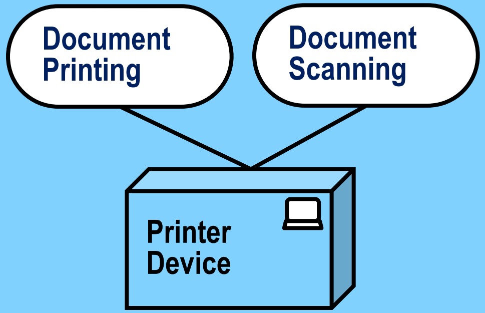
Business (usually human) layer
"offers products and **services** to external customers realized in the organization by business processes performed by business **actors**." ArchiMate



Applications layer
"supports the business layer with **application services** realized by (software) **applications."** ArchiMate

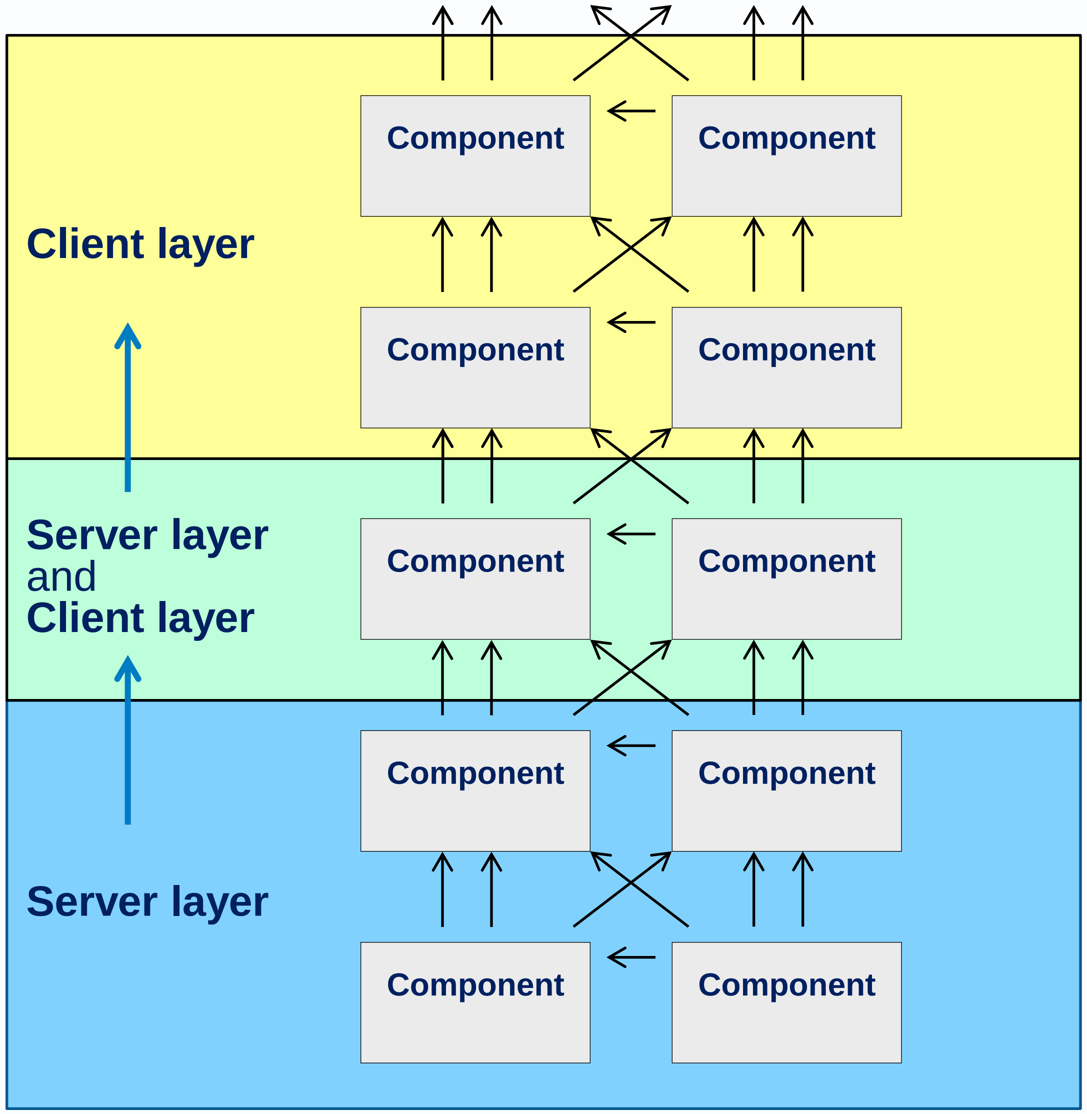


Technology layer
"offers **infrastructure services** needed to run applications realized by computer and communication **hardware and system software**" ArchiMate



Delegation by client-server laying

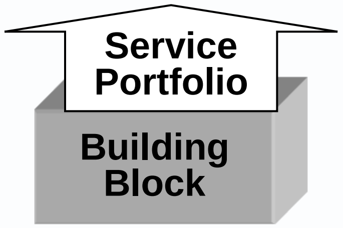
- Components can be arranged in client-server layers
- Client components **delegate** work to server components

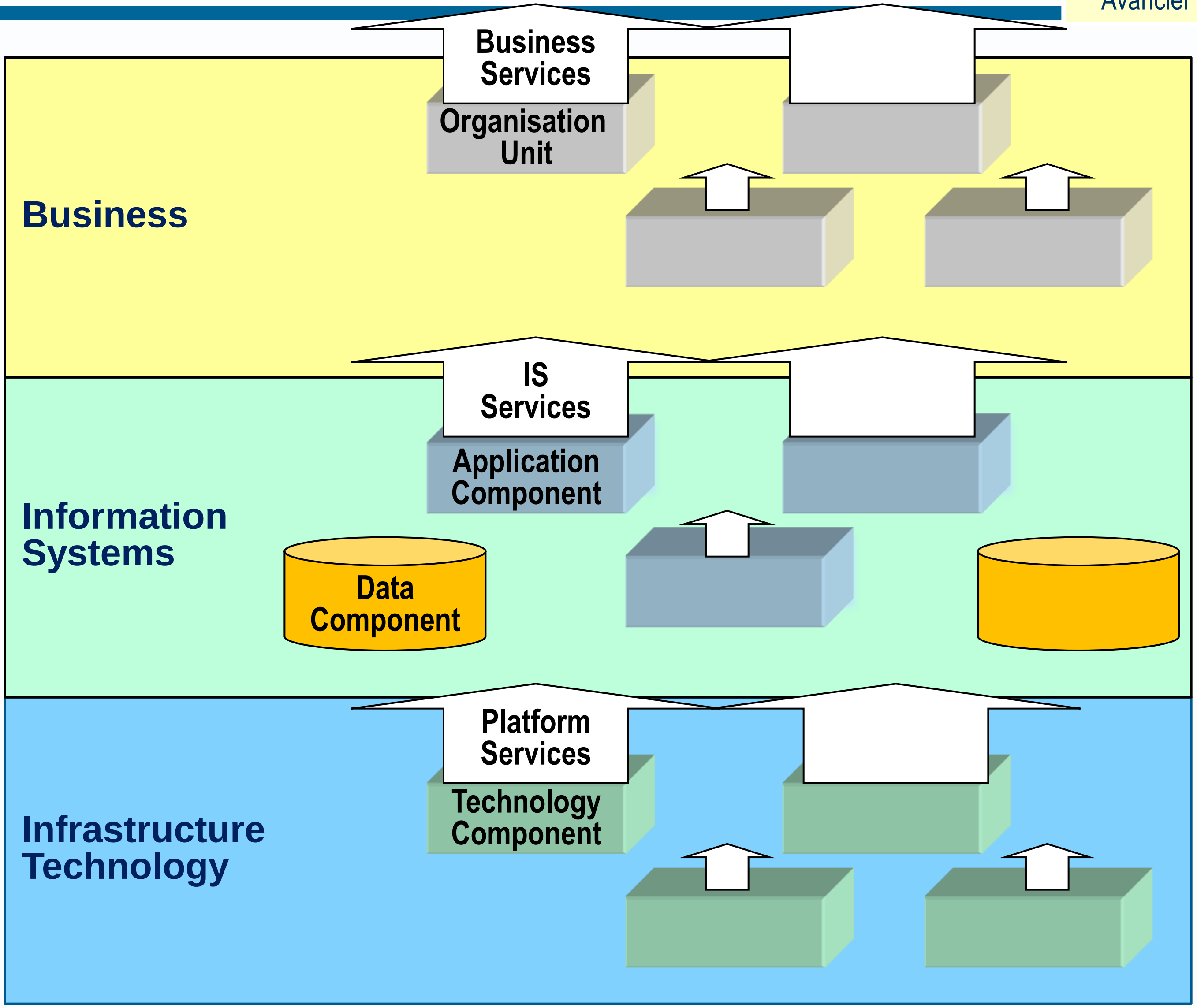


Client-server layering (using TOGAF terms)

- This 3-layer view is commonplace in architecture frameworks like TOGAF

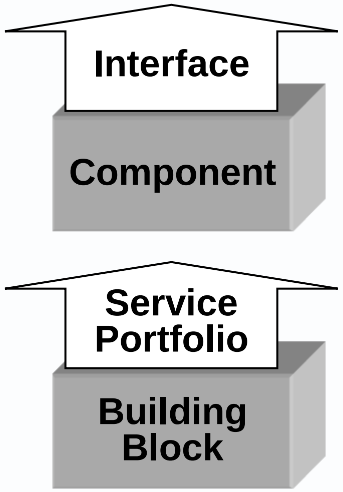
- Each BB is defined by the service portfolio it provides.

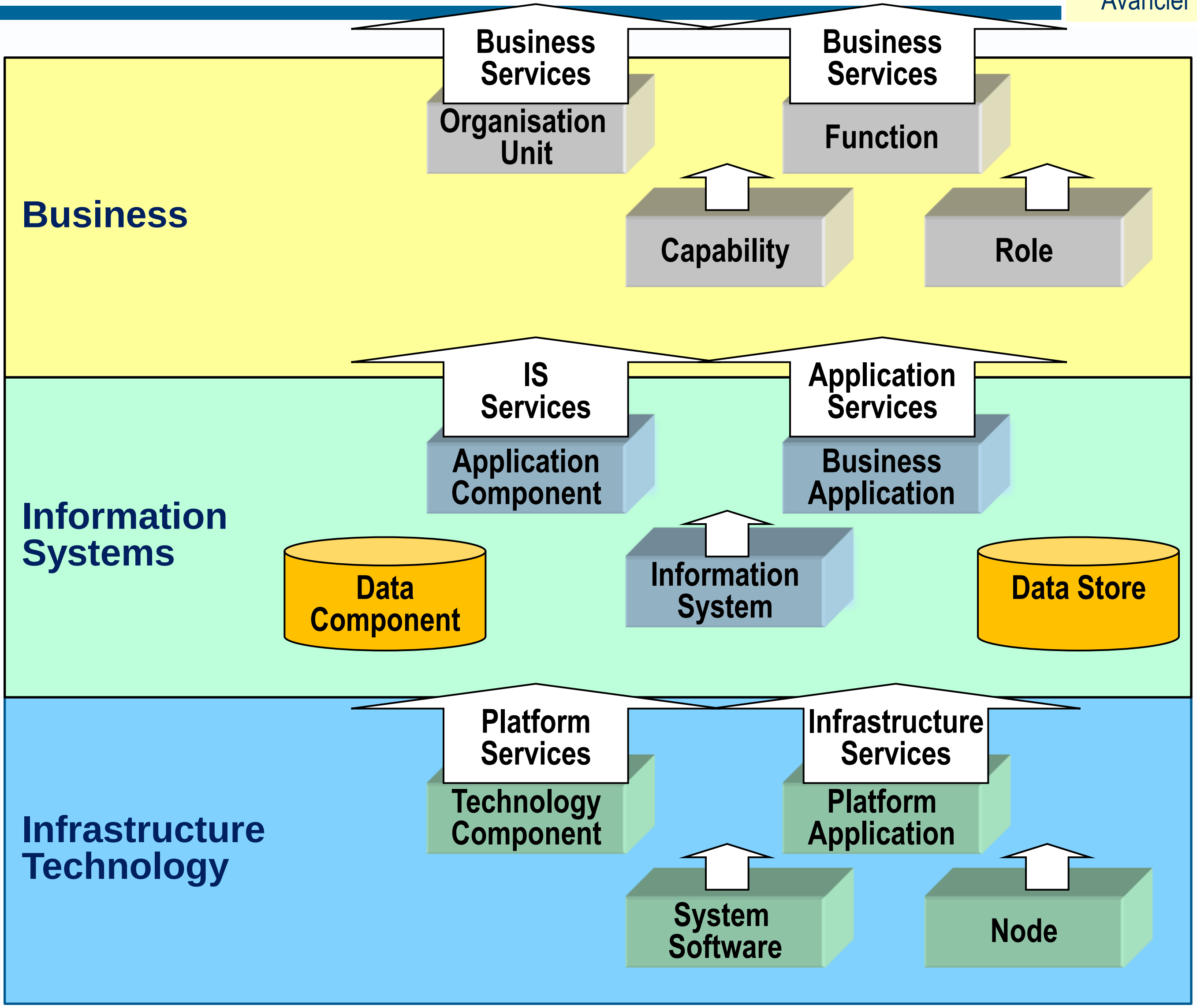




The vocabulary challenge

- Different frameworks use different terms for the same and very similar concepts





Avancier Methods

Conceptual framework – part two

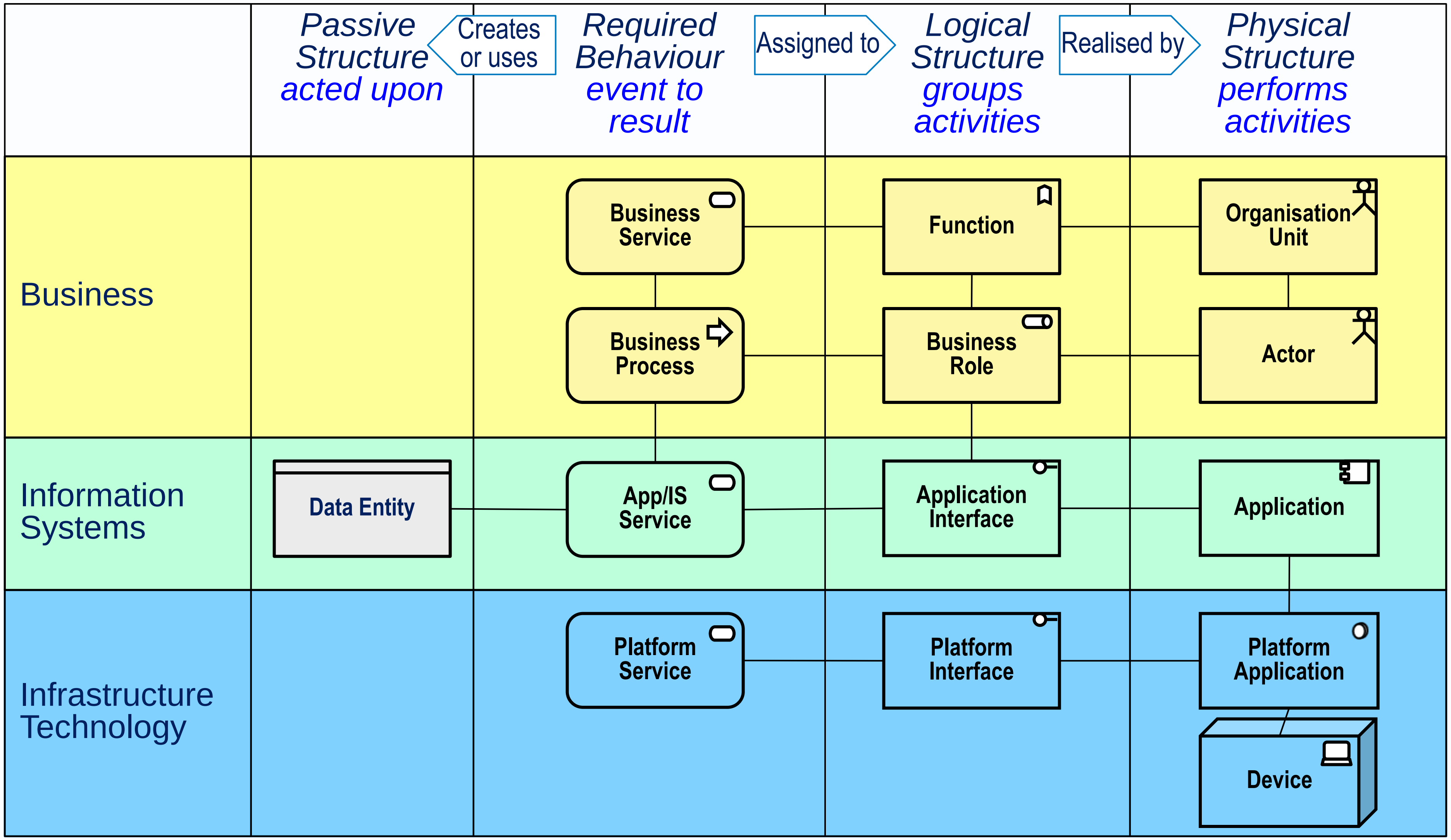
The core framework

ArchiMate’s Core Framework – 3 layers * 3 aspects

	Passive Structure	Creates or uses	*Required Behaviour*	Assigned to	*Active Structure*
Business layer					
Applications layer					
Technology layer					

Avancier

A TOGAF/ArchiMate compromise



Avancier Methods Core Framework

- Similar to that in countless of EA frameworks, including TOGAF

	Passive Structure	*Required Behaviour*	*Logical Structure*	*Physical Structure*
Business		Business Service Business Process	Function Role	Org Unit Actor
Data / Information	Data Entity	Data Flow	Log Data Model	Data Store
Applications		IS Service	Application Interface	Application
Infrastructure Technology		Platform Service	Platform Interface	Platform Application

Questions

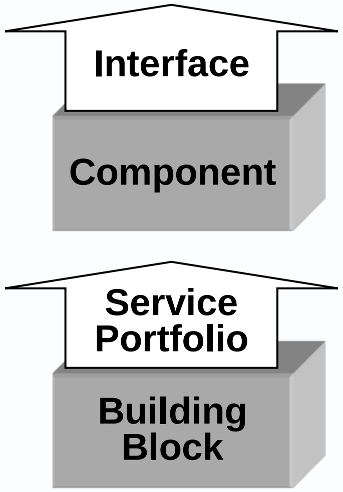
Avancier

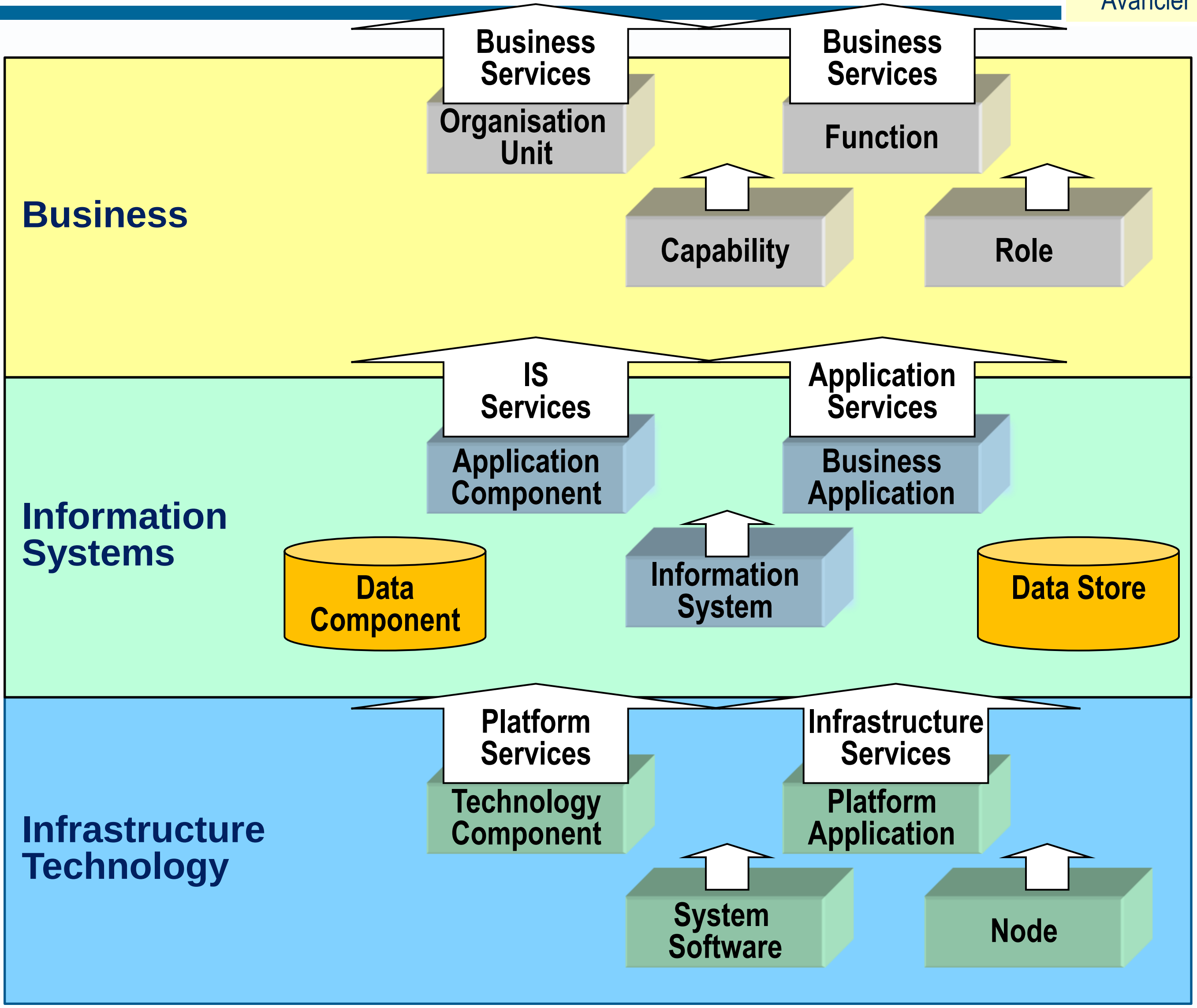
Which is true?

1. A component is a package of capability that offers one or more services
2. Ideally, a service is assigned to one component
3. A component is a subsystem
4. A component is elementary (cannot be decomposed)
5. A component can be classified on scales
   - from Business to Technical and
   - from Logical to Physical

Remember the vocabulary challenge?

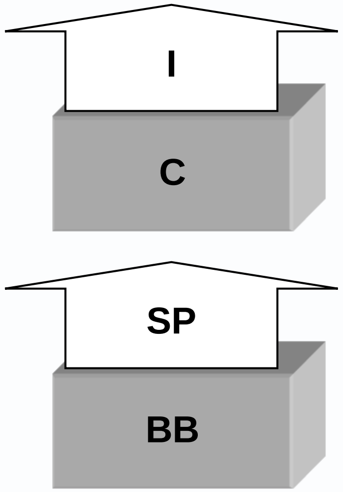
- Different frameworks use different terms for the same and very similar concepts

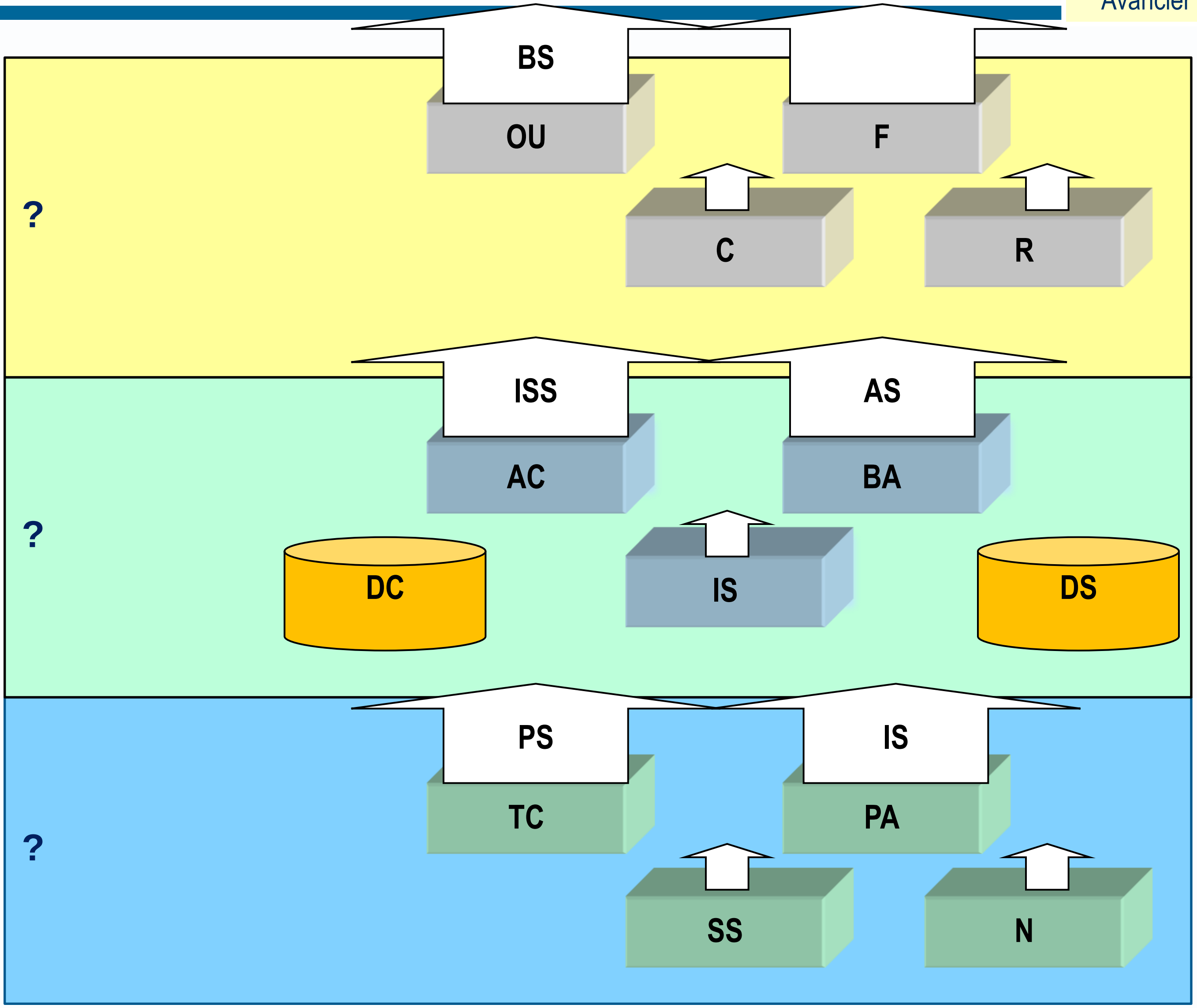




The vocabulary challenge

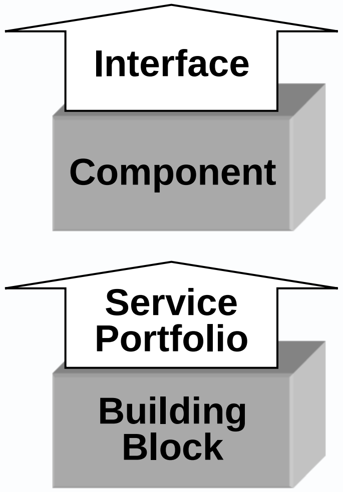
- How many box names can you complete?

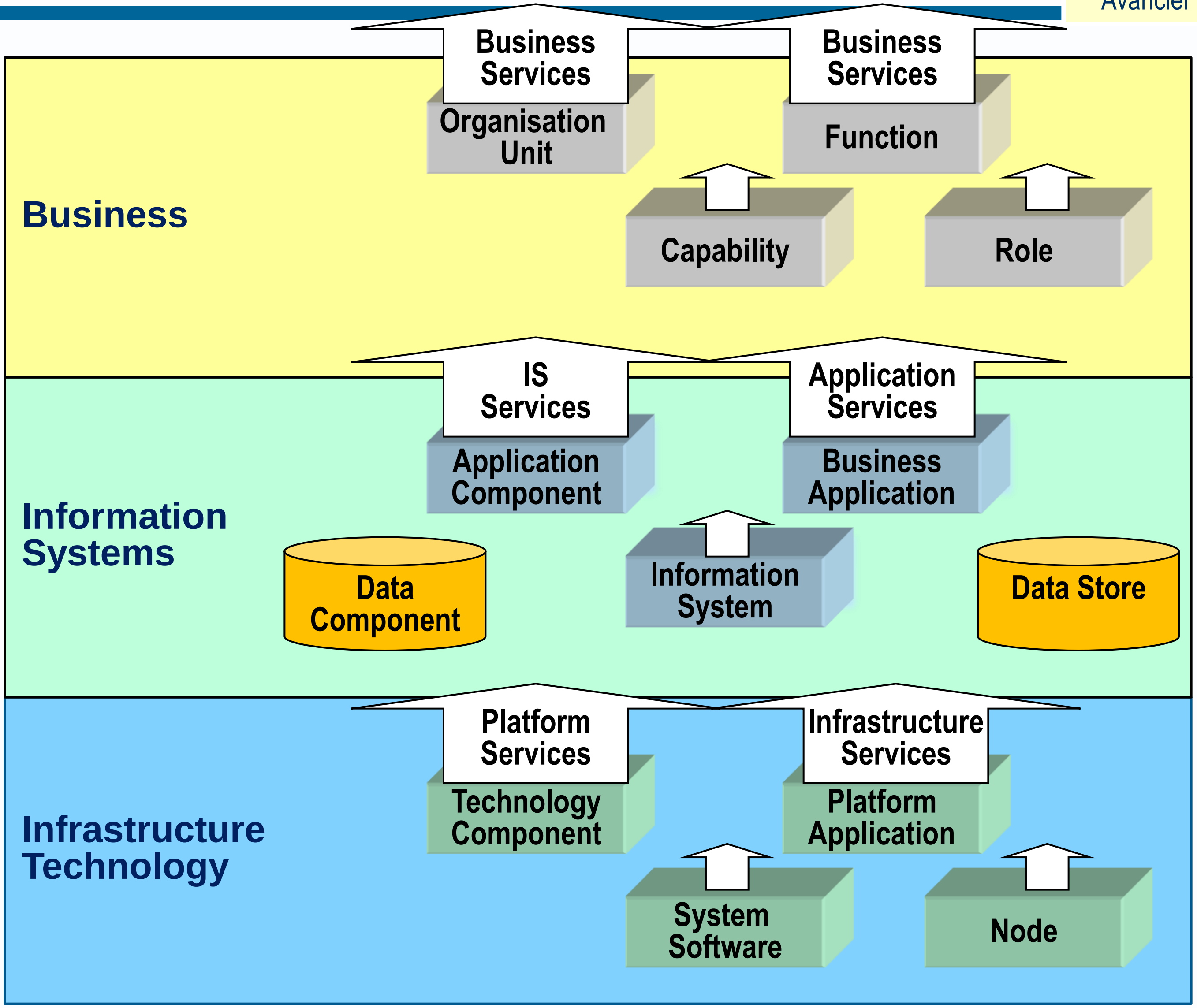




The vocabulary challenge

- Different frameworks use different terms for the same and very similar concepts


Interface
Component
Service Portfolio
Building Block


Business
Business Services
Organisation Unit
Business Services
Function
Capability
Role
Information Systems
IS Services
Application Component
Application Services
Business Application
Data Component
Information System
Data Store
Infrastructure Technology
Platform Services
Technology Component
Infrastructure Services
Platform Application
System Software
Node