Software (Systems) Architecture Foundations

Supplementary Material

Alar Raabe

Content

- Interfaces
  - Interface as Concept
  - Interfaces in Module Structures
  - Interfaces in Component-and-Connector Structures
  - Principles about Interfaces and the Nature of Interfaces
- Micro-Services vs. Enterprise Service Bus
- Scaling Micro-Services vs. Scaling Monoliths

Interface is ...

Interface is the **Boundary of Interaction**

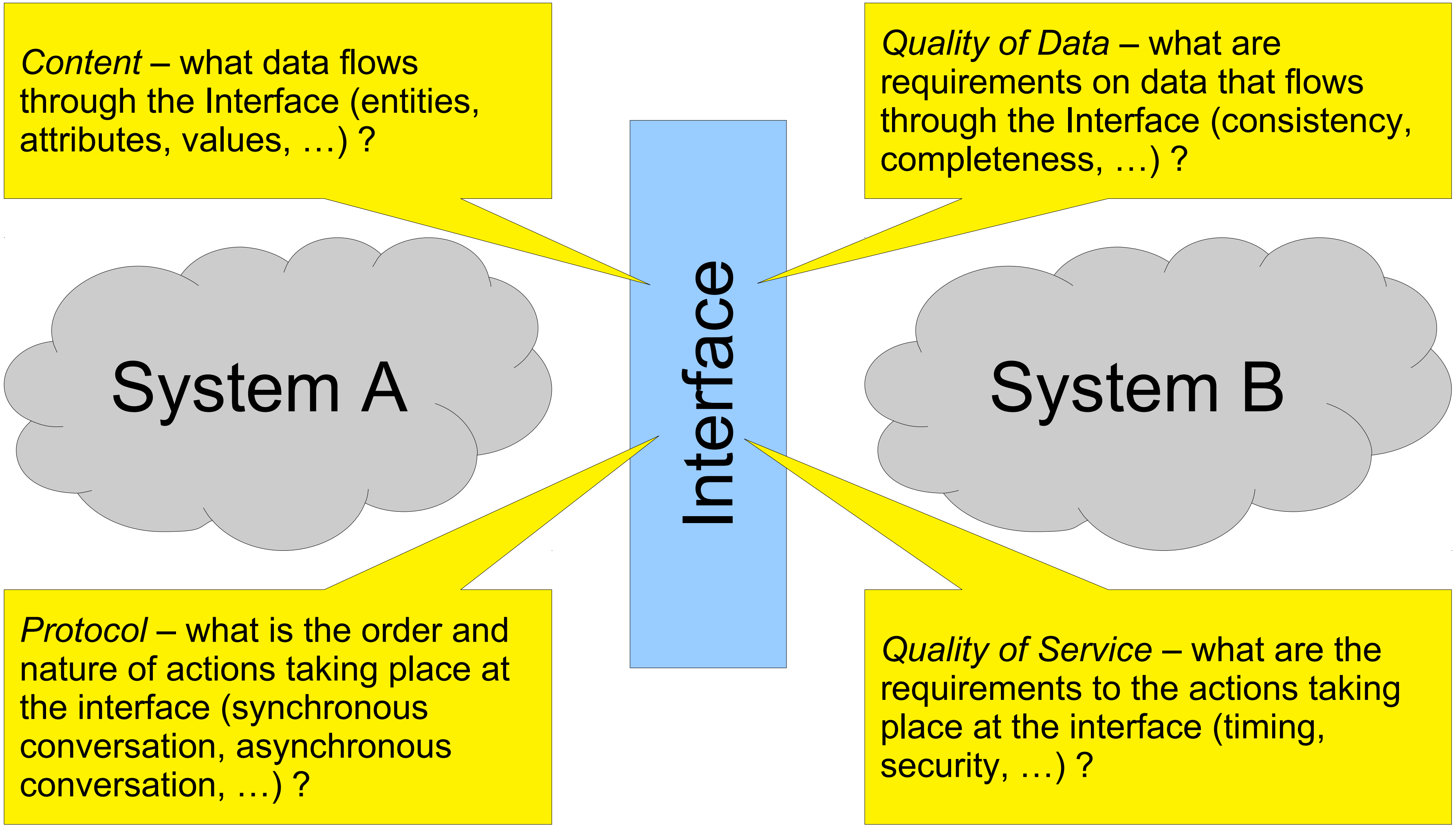
- An interface is a **boundary across which** two **elements** meet and **interact** or communicate with each other (CMU SEI)

- Interface is an **interconnection** and inter-relationships between, for example, people, systems, devices, applications, or the user and an application or device (TOGAF)

- Interface is an external active structure element, that represents **a point of access** where one or more services are provided to the environment (ArchiMate)

- IEEE
  - Interface is a **hardware or software component that connects** two or more other components for the purpose of passing information from one to the other
  - Interface is an **abstraction of the behavior** of an object that consists of a subset of the interactions of that object together with a set of constraints on when they can occur

Interface – as a Boundary where two Systems meet



Content – what data flows through the Interface (entities, attributes, values, …) ?

Quality of Data – what are requirements on data that flows through the Interface (consistency, completeness, …) ?

Protocol – what is the order and nature of actions taking place at the interface (synchronous conversation, asynchronous conversation, …) ?

Quality of Service – what are the requirements to the actions taking place at the interface (timing, security, …) ?

Architecture Structures and Interface (CMU SEI)

All elements of Architecture have Interfaces

- **Module structures** – embody decisions as to how the system is to be structured as a set of code or data units that have to be constructed or procured

Module Interface defines what is available (visible) to other Modules

- **Component-and-connector structures** – embody decisions as to how the system is to be structured as a set of elements that have run-time behavior (components) and interactions (connectors)

Component Interface is called **Port** and it defines the potential interactions (behavior) of Component with its environment

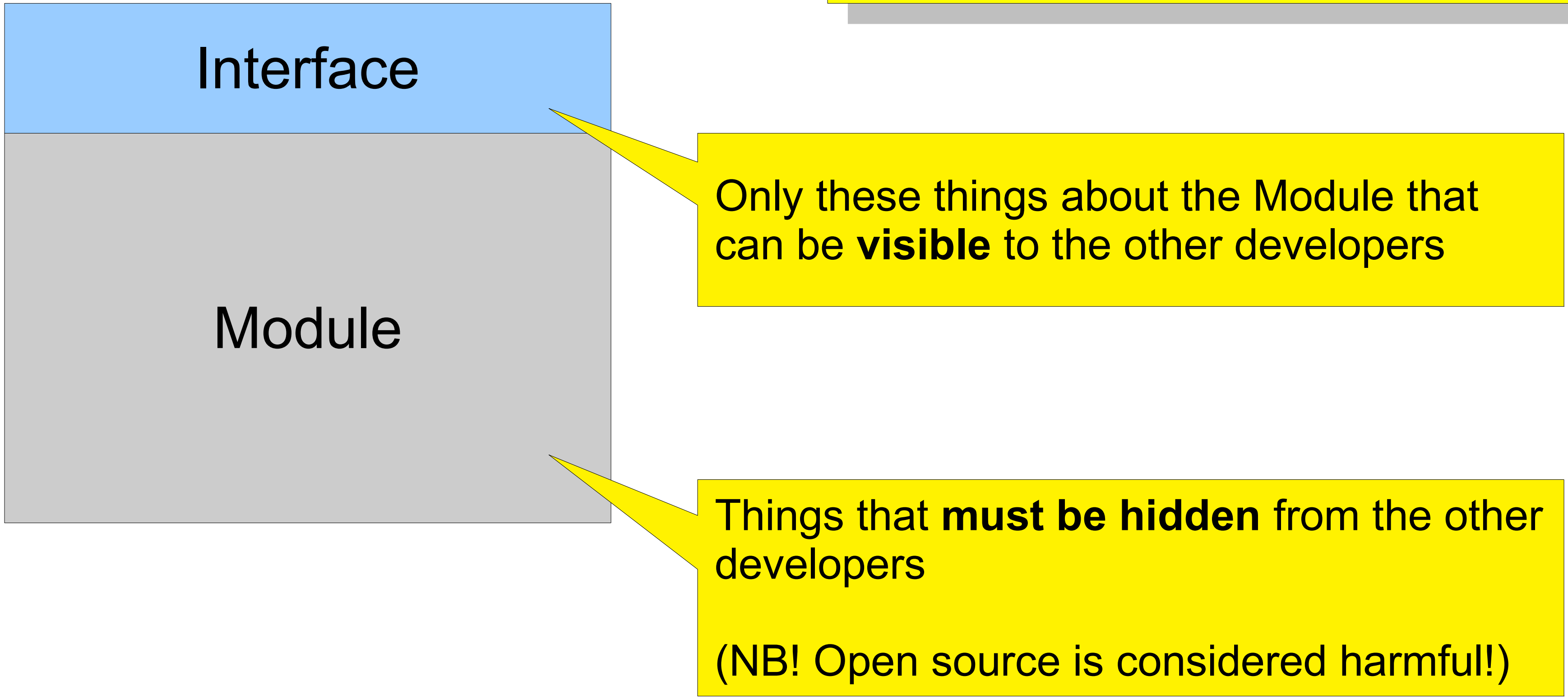
Connector Interface is called **Role** and it defines the ways how Connector can be used (specifying protocol of interaction – i.e. prescribing what patterns of events or actions are allowed to take place over the Connector)

Attachments can be made only between **compatible Ports and Roles** – Components can be attached only to Connectors, not to other Components an vice versa

In case of **Interface delegation** Component ports can be associated with one or more Ports in an “internal” structure (similarly for the roles of a connector)

Module Interface → External Definition of Module (Signature & Invariants)

> Interface is the set of assumptions that each programmer needs to make about the other program in order to demonstrate the correctness of his own program
>
> D. L. Parnas



Interface

Module

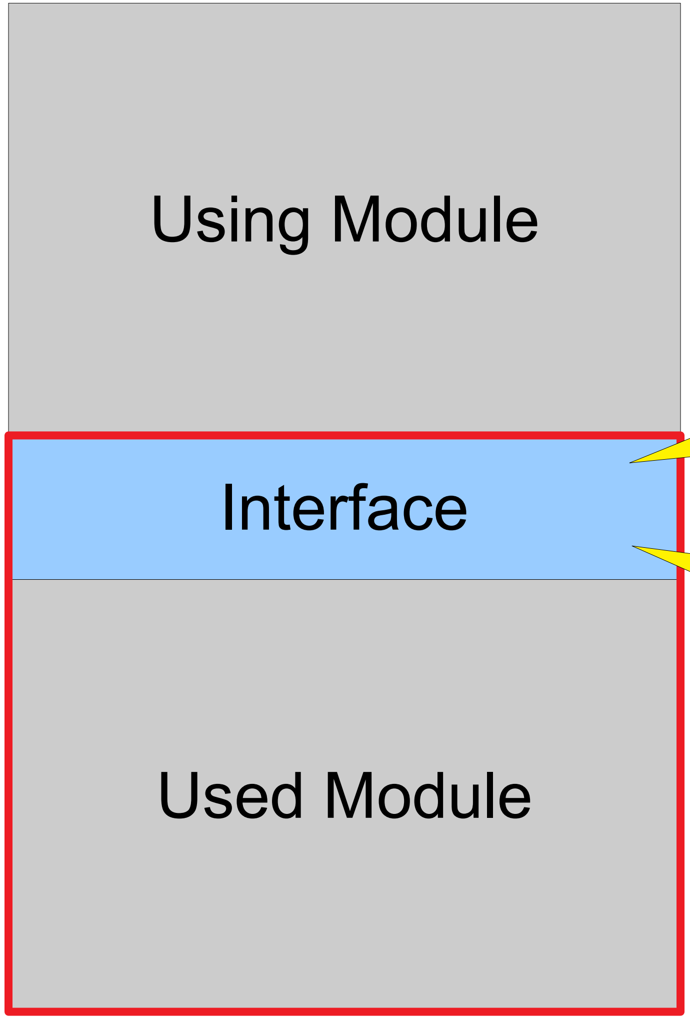
Only these things about the Module that can be **visible** to the other developers

Things that **must be hidden** from the other developers

(NB! Open source is considered harmful!)

Module Interface → External Definition of Module

Module can’t have several Interfaces of same type

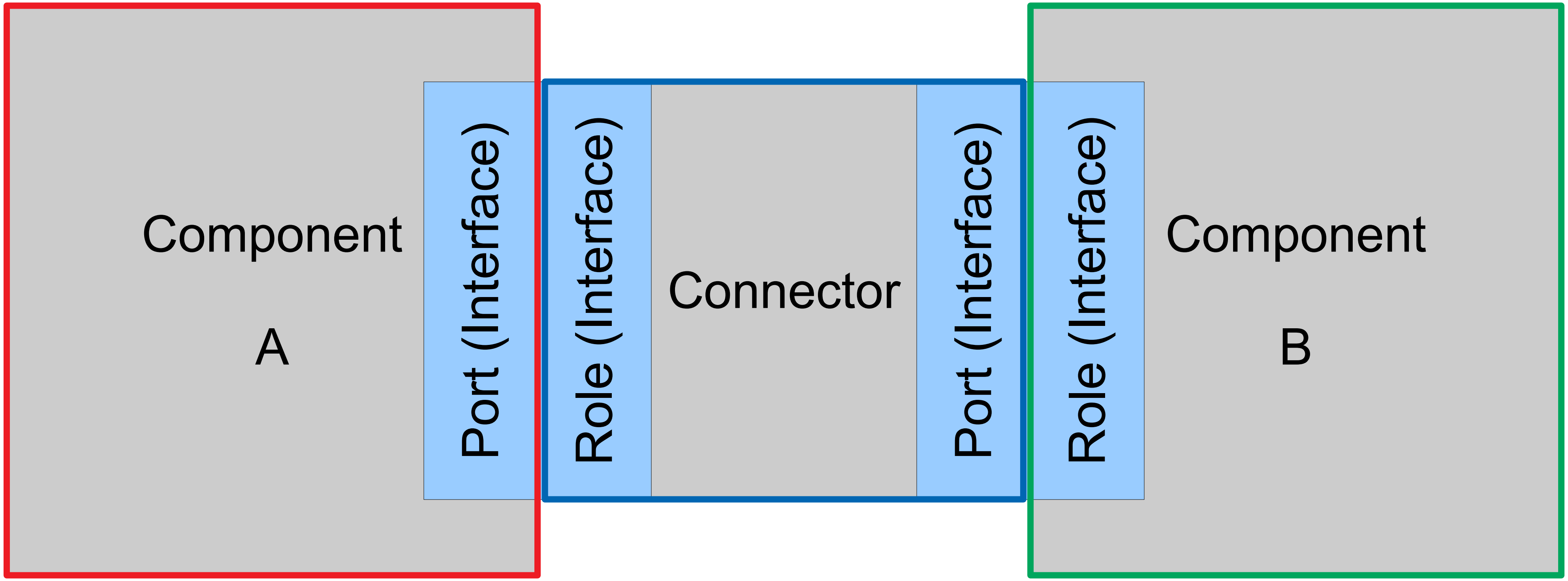


Interface is the Shared Resource – a Contract between the Using Module and Used Module

Change of Interface can break the Contract !

Component Interface and Connector Interfaces → Possible Behavior / Interactions & Data

Components and Connectors can have several Interfaces of same type



Some Principles about Interfaces

CMU SEI Views & Beyond

- All elements have interfaces – all software elements interact with their environment
- An element’s interface is separate from its implementation
- An element can have multiple interfaces
  - Each interface contains a separate collection of resources (functions, data, message end points, event triggers, ...) that have a related logical purpose, or represent a role that the element could fulfill
  - Multiple interfaces provide a separation of concerns – a specific actor might require only a subset of the resources
  - Evolution can be supported by keeping the old interface and adding a new one
- Elements not only provide interfaces but also require interfaces
  - An element interacts with its environment by making use of resources or assuming that its environment behaves in a certain way – without these required resources, the element cannot function correctly
- Multiple actors may interact with an element through its interface at the same time (if interface allows multiple concurrent interactions)
- Interfaces can be extended by generalization
  - Examples of resources often shared by several interfaces include: an initialization operation, a set of exceptions, …
- Sometimes it’s useful to distinguish interface types from interface instances in the architecture (if components can provide multiple instances of the same interface)

Nature of Interface

Asynchronous ≠ Event-Driven

- Nature of Usage
  - Provided Interface (the services offered to others) vs. Required Interface (services needed from others)
- Nature of Interactions
  - Synchronous (requester waits until result is delivered) vs. Asynchronous (requester doesn’t wait the result)
  - Transactional Protocol (stateless) vs. Conversational Protocol (stateful)
- Nature of Behavior
  - Fine-Grained (small actions) vs. Coarse Grained (large actions)
- Nature of Actor
  - User Interface (Human Interface) vs. Programming Interface (API)
- Nature of Operation(s)
  - REST API (requesting a resource representation) vs RPC API (requesting a service)

Content

- Interfaces
  - Interface as Concept
  - Interfaces in Module Structures
  - Interfaces in Component-and-Connector Structures
  - Principles about Interfaces and the Nature of Interfaces
- Micro-Services vs. Enterprise Service Bus
- Scaling Micro-Services vs. Scaling Monoliths

Micro-Services vs. Enterprise Service Bus

Micro-Services → One Application
ESB → Many Applications

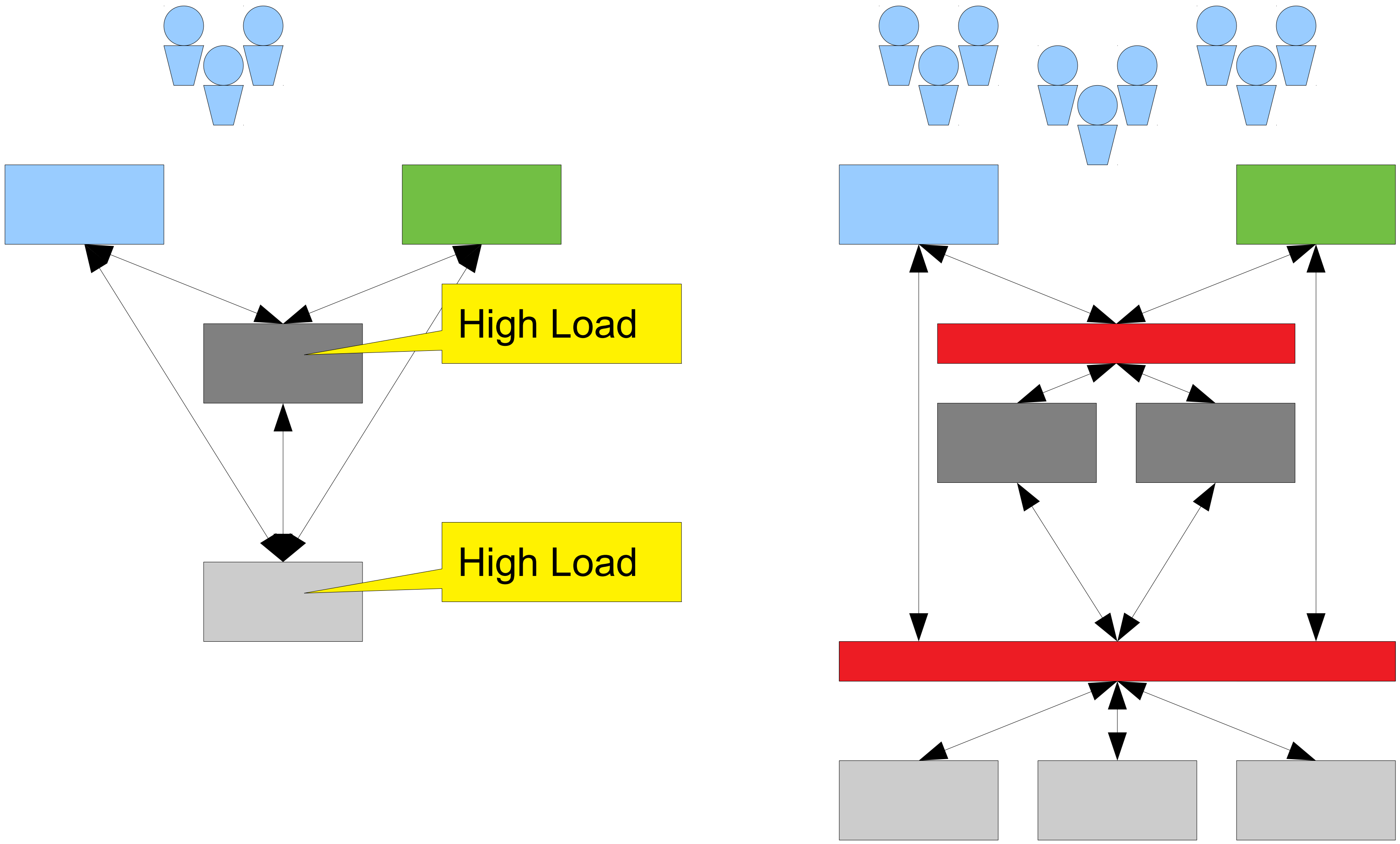
- Micro-Services
  - For modularization of applications into loosely coupled components
  - No centralized service management
  - Focuses on decoupling (decomposition of application)
  - No centralized communication infrastructure (better error tolerance)
  - Usually single simple communication protocol
  - Limits integration choices
  - Usually smaller granularity of components
  - Multiple independent data-stores
  - Relaxed governance
  - Better suited **for compact and well-partitioned applications**

- Enterprise Service Bus (SOA)
  - For composition of application from independent components
  - Centralized service management
  - Focuses on reuse (of business functionality)
  - Usually centralized communication infrastructure (single point of failure)
  - Supports multiple communication protocols
  - Focuses on interoperability
  - Usually larger granularity of components
  - Usually share data-store
  - Common governance
  - Better suited **for large complex enterprise applications (sets of applications)**

Content

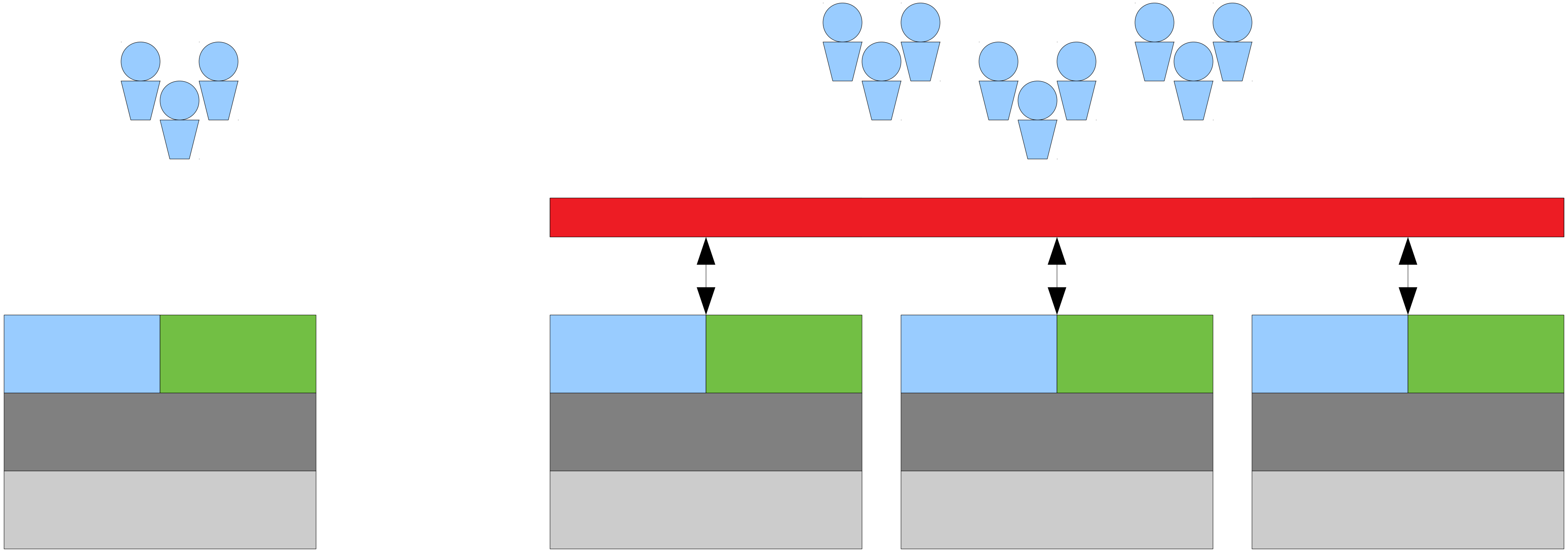
- Interfaces
  - Interface as Concept
  - Interfaces in Module Structures
  - Interfaces in Component-and-Connector Structures
  - Principles about Interfaces and the Nature of Interfaces
- Micro-Services vs. Enterprise Service Bus
- Scaling Micro-Services vs. Scaling Monoliths

Scaling Micro-Services
→ fine control



Scaling the Monolith (“cookie cutter” scaling) → simpler to manage

> If you can’t build a monolith, what makes you think micro-services are the answer?
>
> S. Brown



38’. When the process is lost, there is good practice
When good practice is lost, there are rules
When rules are lost, there is ritual
Ritual is the beginning of chaos

Lao Tsu (by Philippe Kruchten)

Thank You!

Literature

-
- https://flylib.com/books/en/2.121.1/
  - https://flylib.com/books/en/2.121.1/3366_overview.html
- http://blog.robertelder.org/interfaces-most-important-software-engineering-concept/
- http://roy.gbiv.com/untangled/2008/rest-apis-must-be-hypertext-driven
- http://www.codingthearchitecture.com/2014/07/06/distributed_big_balls_of_mud.html
- https://martinfowler.com/articles/microservices.html#MicroservicesAndSoa
- ...
-
- … Google ...