Fraunhofer IESE

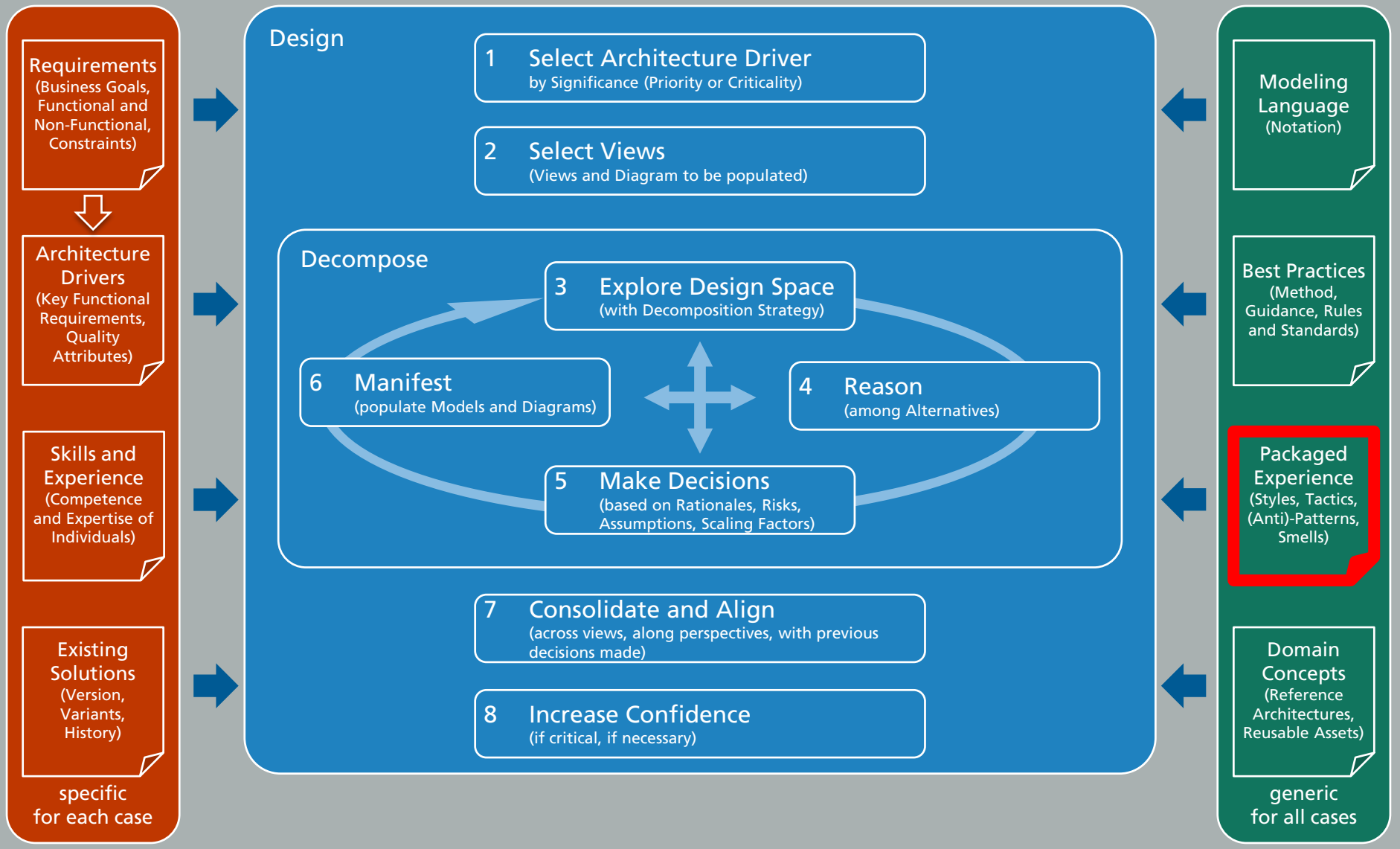
TECHNISCHE UNIVERSITÄT KAISERSLAUTERN

SSA 07 –Architecture Styles and Pattern

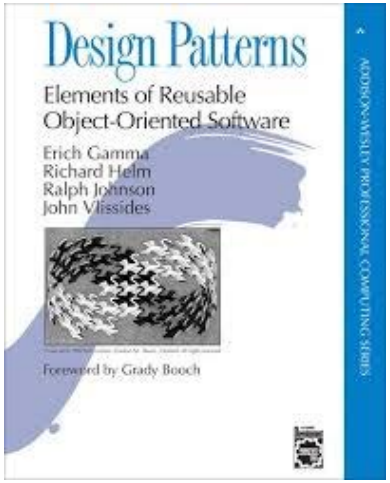
Dr. Andreas Morgenstern
andreas.morgenstern@iese.fraunhofer.de

TU Kaiserslautern, SS2018
Lecture "Software and System Architecture (SSA)"

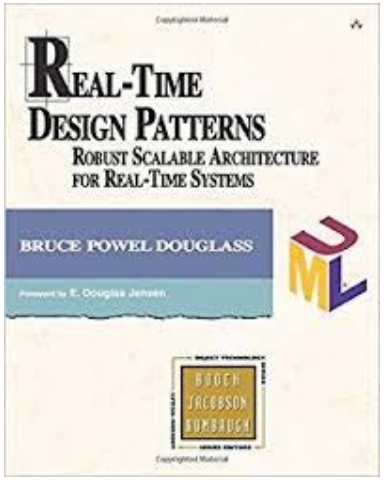
Architecture Design Iteration



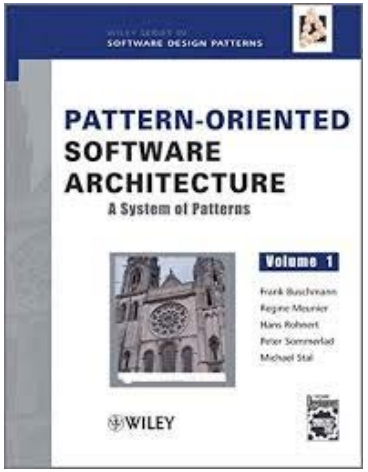
Literature for Software Architecture Pattern, Styles and Tactics



every developer

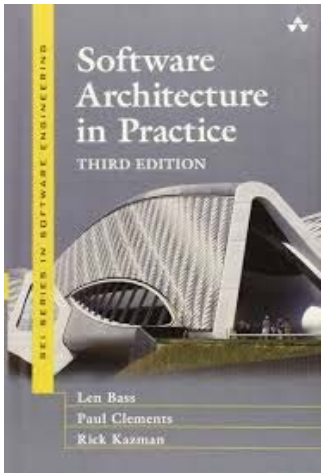


embedded developer

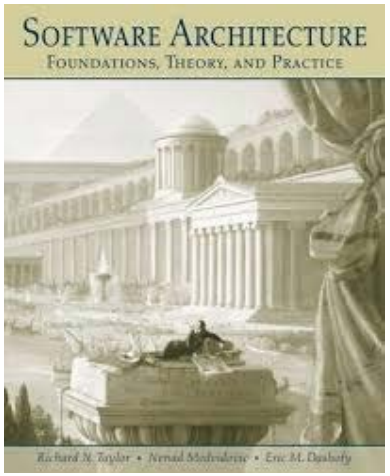


...

interested Software Architect



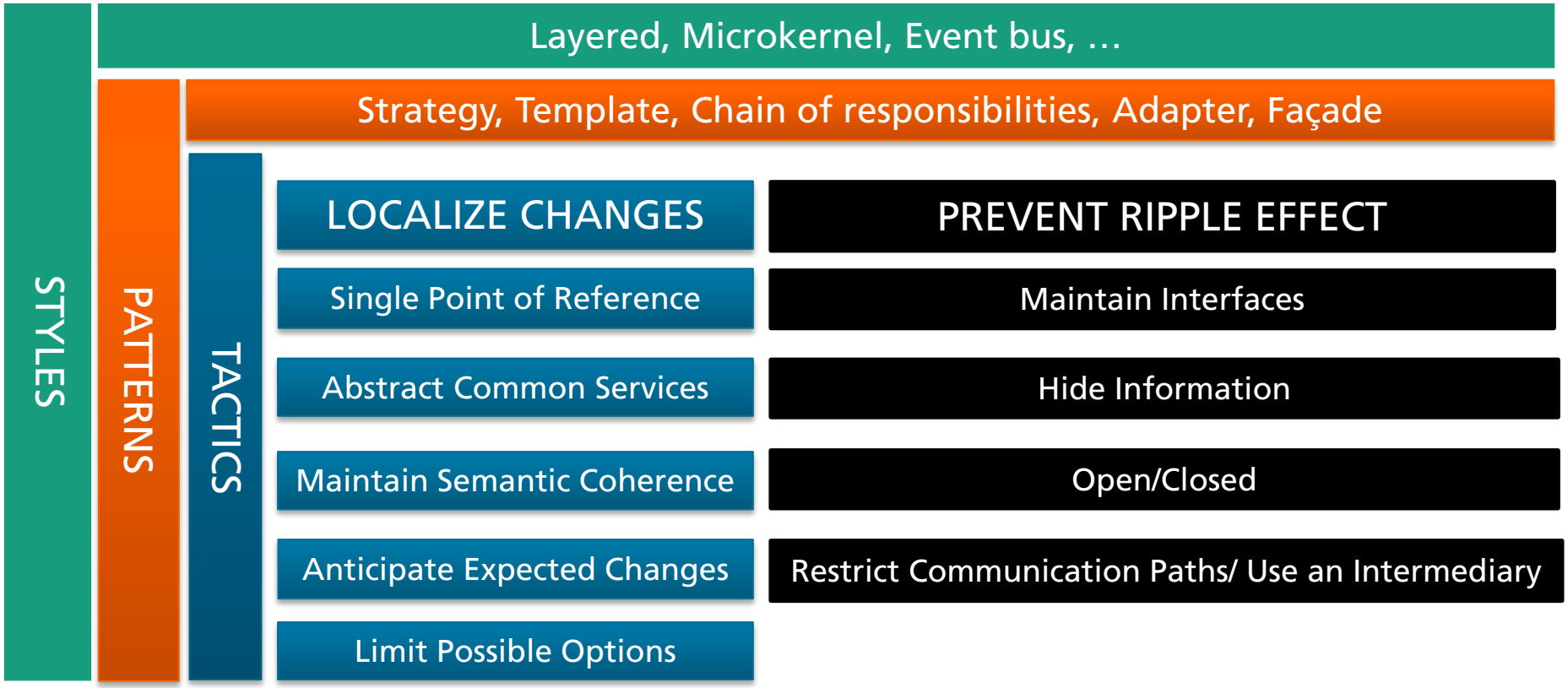
every software Architect



interested Software Architect

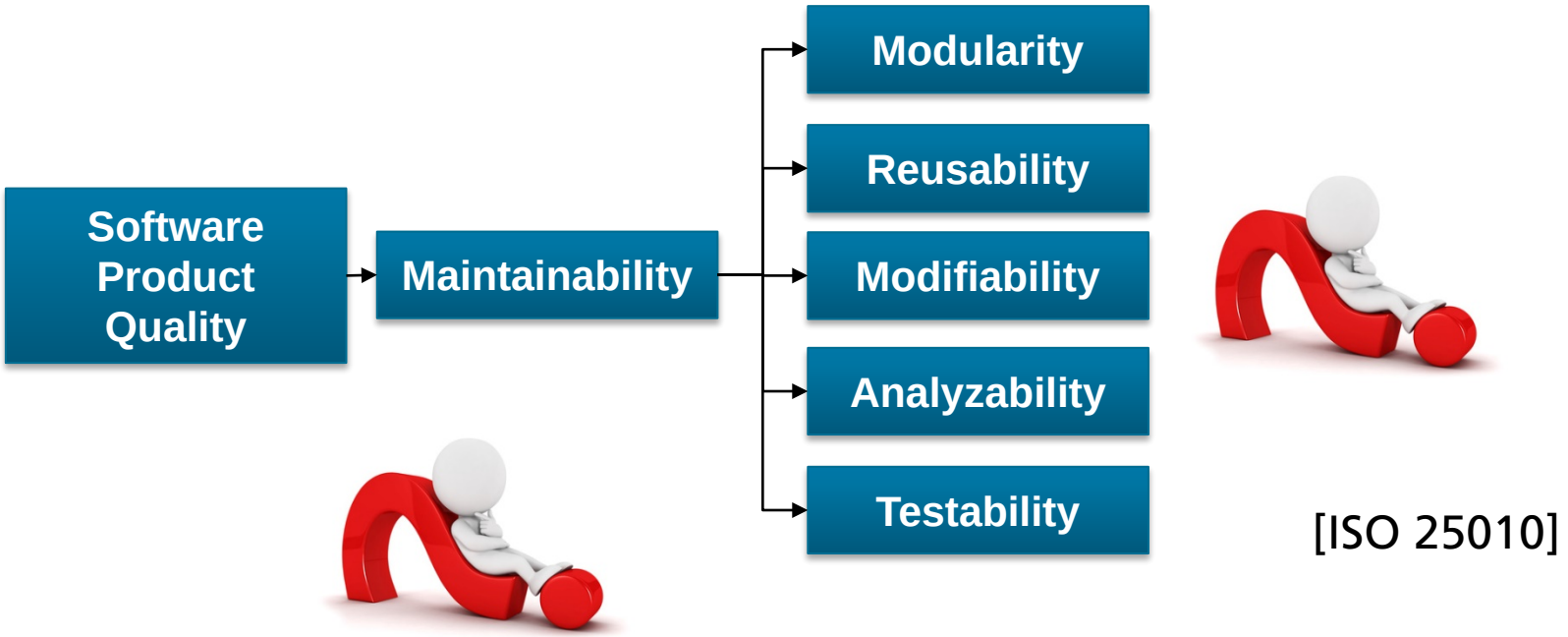
Maintainability

Architecture Tactics, Styles and Patterns: Maintainability



Goal Definition

- Maintainability= „Quality attribute relating to degree of effectiveness to which a software product can be modified by the maintainers.“



[ISO 25010]

Develop a system that is **well-structured**, **easy to change** & **analyze**.

© Fraunhofer IESE

Legend: Quality Attribute — Refinement

Maintainability Strategies

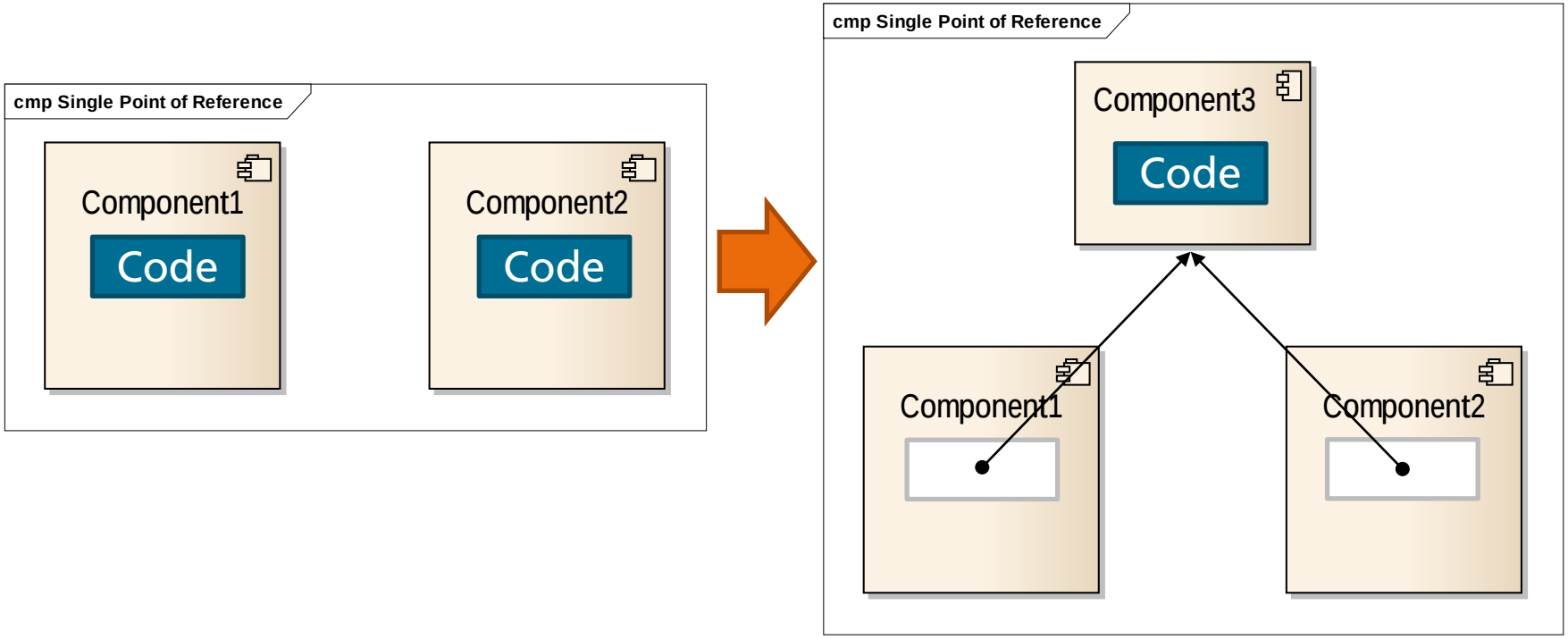
Strategies / Principles	Requ.	Arch.	Design	Impl.	Compil.	V & V	Depl.	Exec.
Localize Changes		☒						
Prevent Ripple Effects		☒						

© Fraunhofer IESE

Legend: ☒ Application of Strategy/Pattern
☑ Resolution of Strategy/Pattern

The Localize Changes Strategy

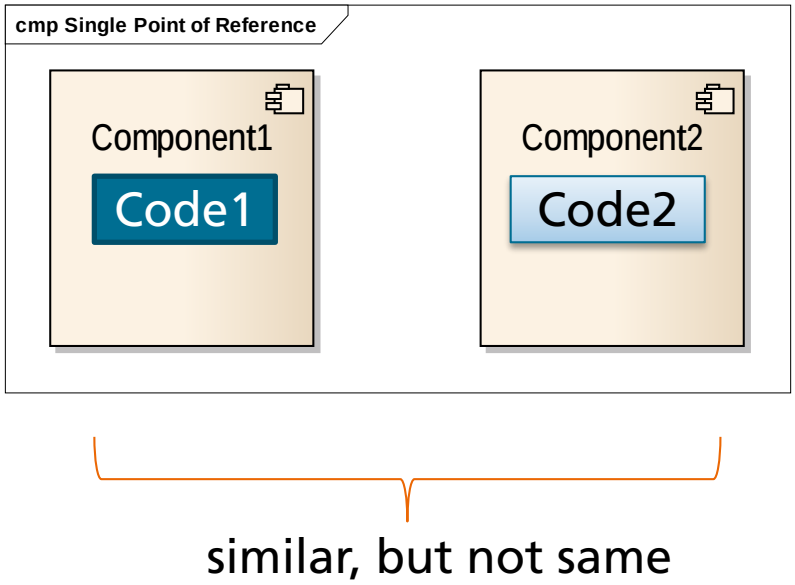
- **Purpose**
  - Reduce number of modules directly affected by changes
- **Tactic: Single Point of Reference**

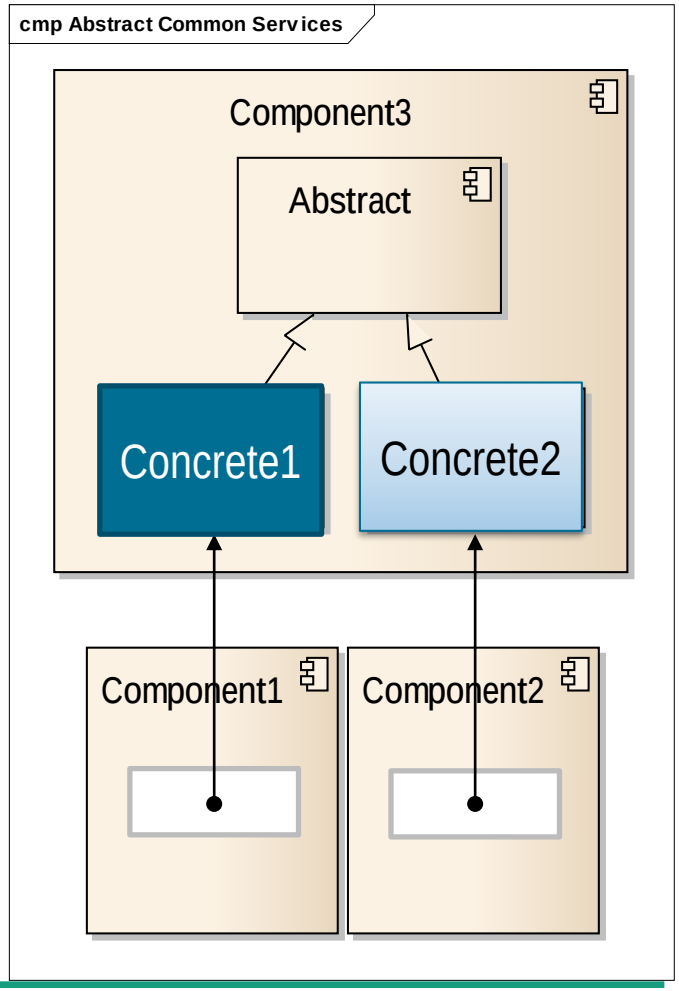


© Fraunhofer IESE

The Localize Changes Strategy

- **Purpose**
  - Reduce number of modules directly affected by changes
- **Tactic: Abstract Common Services**



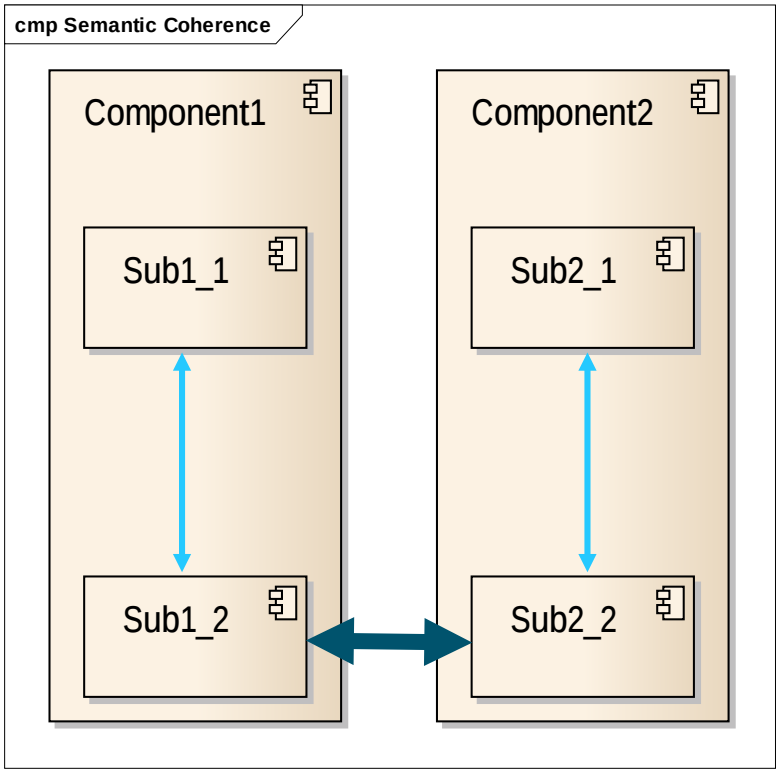


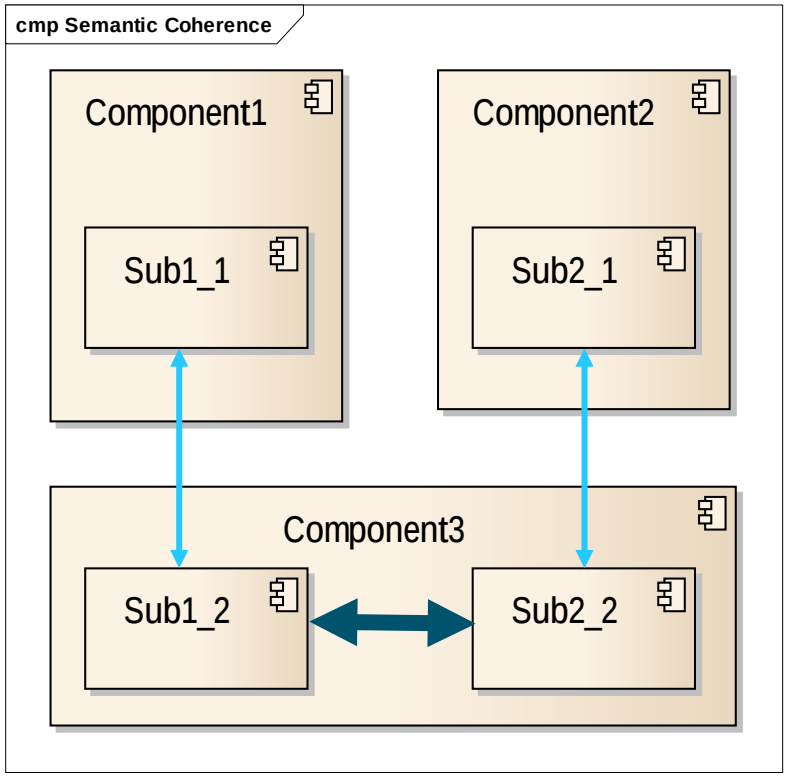
© Fraunhofer IESE

Fraunhofer IESE

The Localize Changes Strategy

- **Purpose**
  - Reduce number of modules directly affected by changes
- **Tactic: Maintain Semantic Coherence**



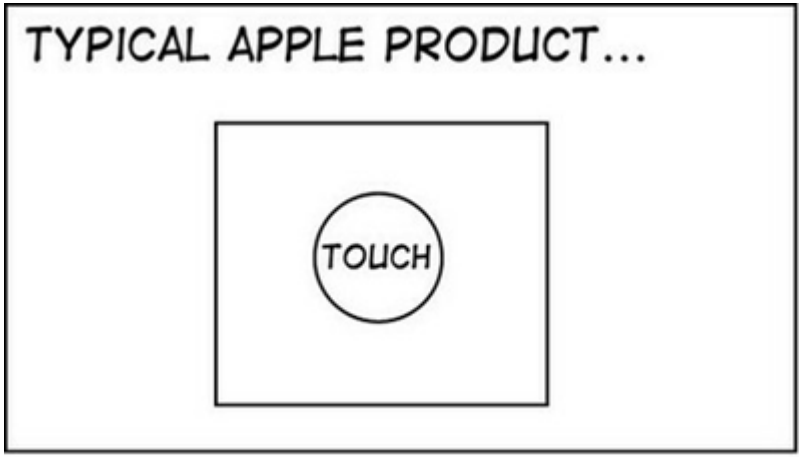


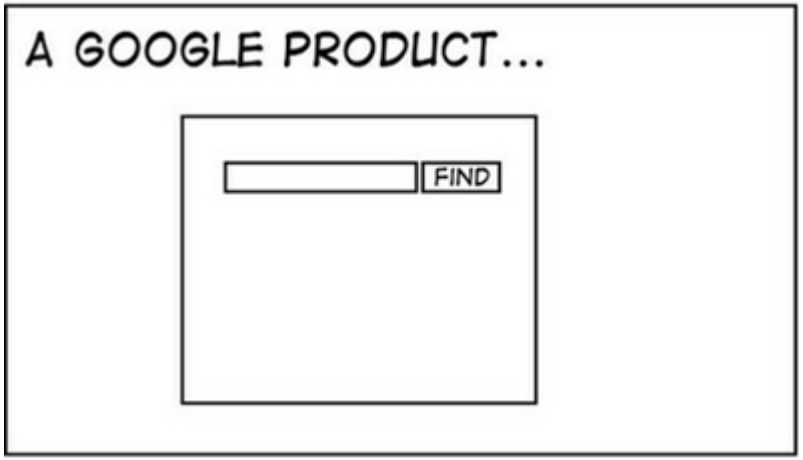
© Fraunhofer IESE

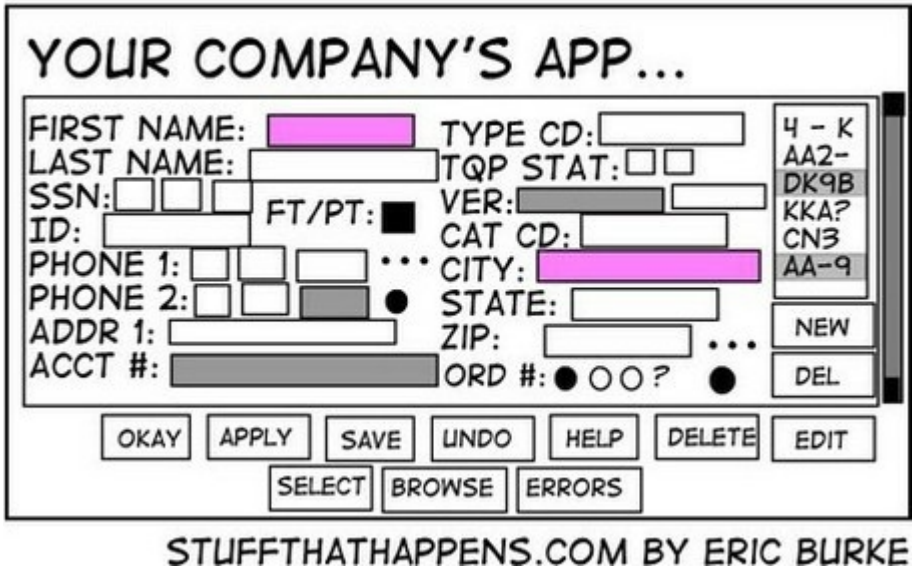
Weak coupling Strong coupling

The Localize Changes Strategy

- **Purpose**
  - Reduce number of modules directly affected by changes
- **Tactic: Limit Possible Options**







STUFFTHATHAPPENS.COM BY ERIC BURKE

© Fraunhofer IESE

Fraunhofer IESE

The Localize Changes Strategy

Strategy ID	Localize Changes

Purpose	▪ Reduce number of modules directly affected by changes
	▪ Tactics to implement the strategy
	  ▪ Single point of reference
	  ▪ Abstract common services
	  ▪ Maintain semantic coherence
	  ▪ Limit possible options
	  ▪ Anticipate expected changes (especially in product lines)

Advantages	▪ Cost reduction for implementing changes ▪ System is more understandable
Drawbacks	▪ Difficulty for determining “right” place of components for semantic coherence

© Fraunhofer IESE

The Prevent Ripple Effects Strategy

Strategy ID	Prevent Ripple Effects

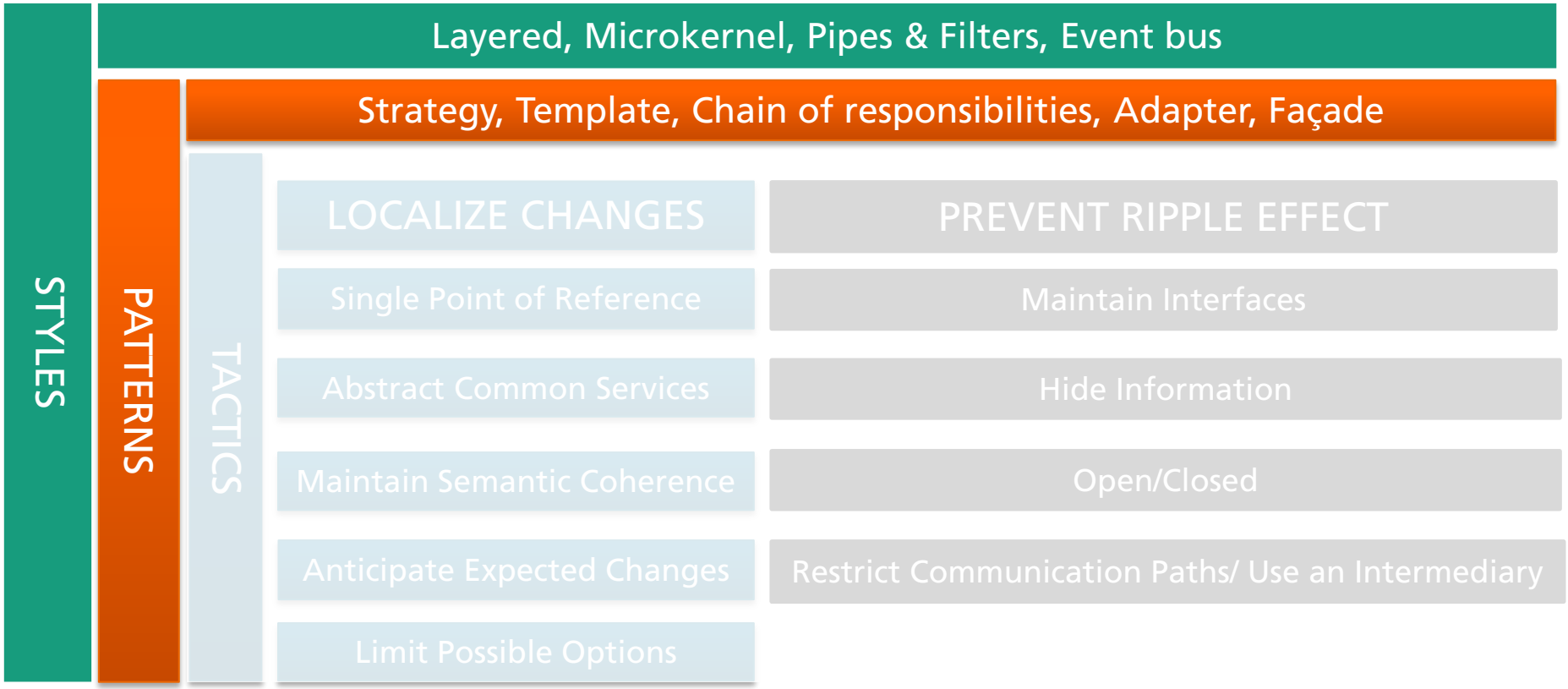
Purpose	▪ Reduce number of modules indirectly affected by changes
	▪ Tactics to implement the strategy
	  ▪ Maintain Interfaces
	  ▪ Hide Information
	  ▪ Open/Closed
	  ▪ Restrict Communication Paths
	  ▪ Use an Intermediary

Advantages	▪ Separation of interface and implementation ▪ More structured system ▪ More stable system
Drawbacks	▪ Difficulty of defining interfaces in advance

… more about strategies and tactics in Software Architecture in Practice

© Fraunhofer IESE

Maintainability Styles and patterns

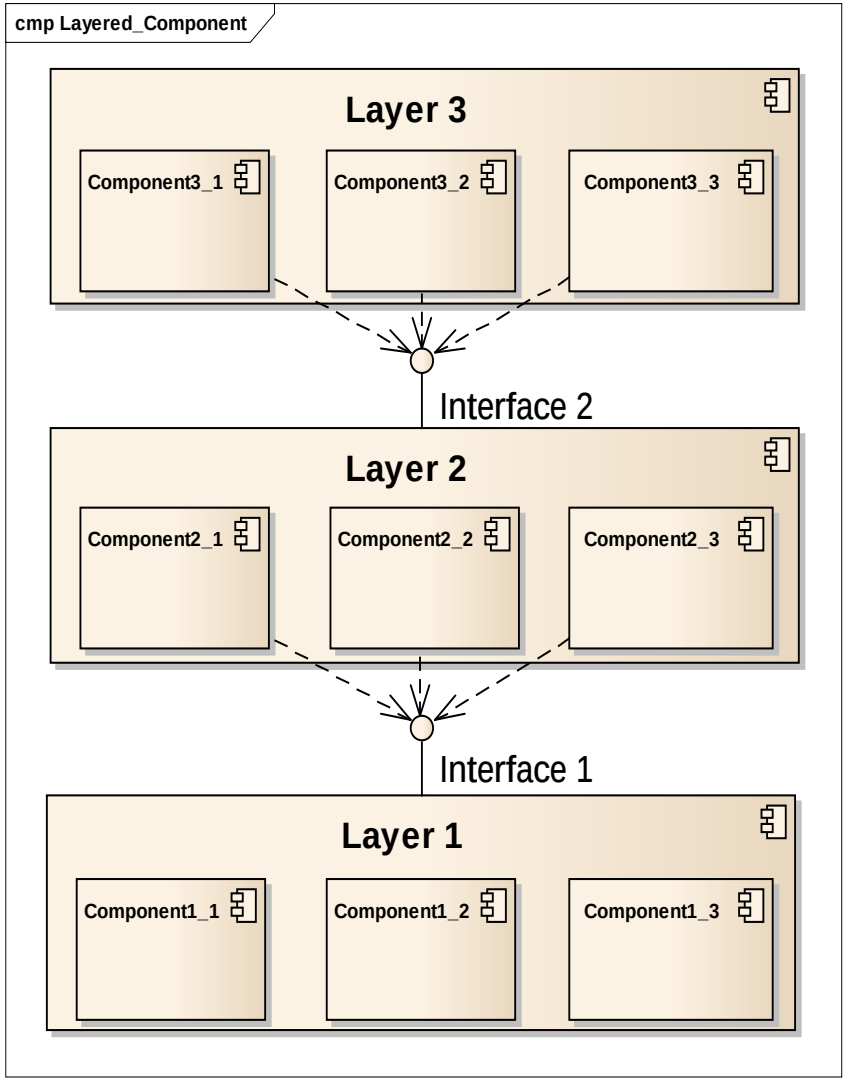


© Fraunhofer IESE

Fraunhofer IESE

The Layered Style

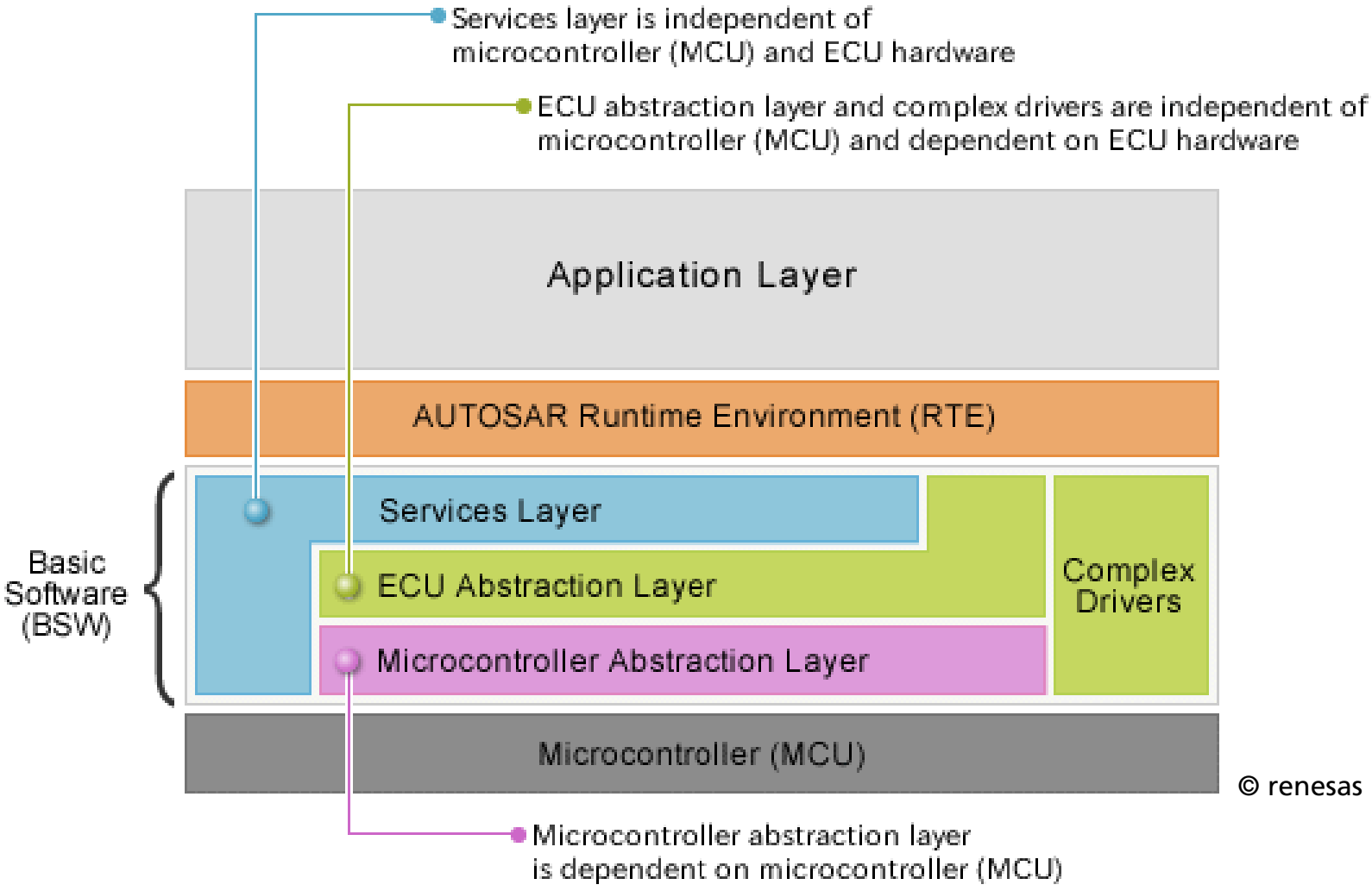
- **Problem/Context**
  - System is decomposed into components
  - Components are defined at different abstraction levels
  - Components depend on each other
- **Solution**
  - Group components into ordered layers
  - Components in a layer may only use services from the layer below it
  - Communication over fixed interfaces



© Fraunhofer IESE

Fraunhofer IESE

The Layered Style: Example AUTOSAR



© Fraunhofer IESE

Fraunhofer IESE

The Layered Style

- **Advantages**
  - Reuse of layers
  - Dependencies are kept local
  - Exchangeability of layer implementation
    (but external behavior must be the same, otherwise potential cascade of changing behavior)

- **Drawbacks**
  - Communication overhead
  - Difficulty to decide on granularity of a layer

© Fraunhofer IESE

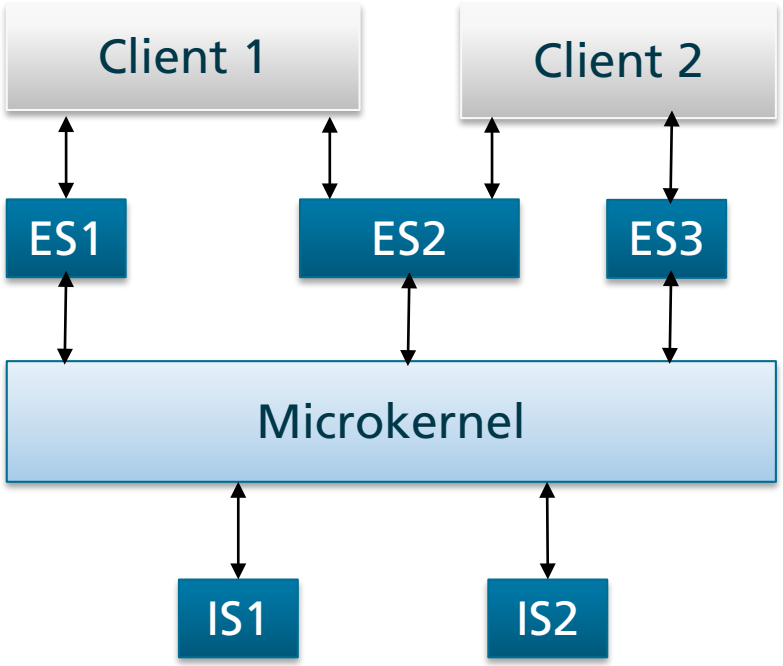
The Layered Style

Pattern ID	Layered
Problem/ Context	▪ System is decomposed into subtasks ▪ Subtasks are defined at different abstraction levels ▪ Subtasks depend on each other
Solution	▪ Group subtasks into ordered layers ▪ Subtasks in a layer may only use services from the layer below it ▪ Communication over fixed interfaces
Advantages	▪ Reuse of layers ▪ Dependencies are kept local ▪ Exchangeability of layer implementation
Drawbacks	▪ Cascade of changing behavior ▪ Communication overhead ▪ Difficulty to decide on granularity of a layer
Impact on Safety	▪ Improvement of safety in a layer trickles through it dependents. ▪ Difficulty handling mixed criticality when there is a high degree of dependencies in a layer. ▪ Safety mechanisms, e.g. failure detection/management, tend to crosscut across the functional layers. ▪ Layering overhead increase latencies and timing uncertainties.
Example	AUTOSAR, TCP/IP

© Fraunhofer IESE

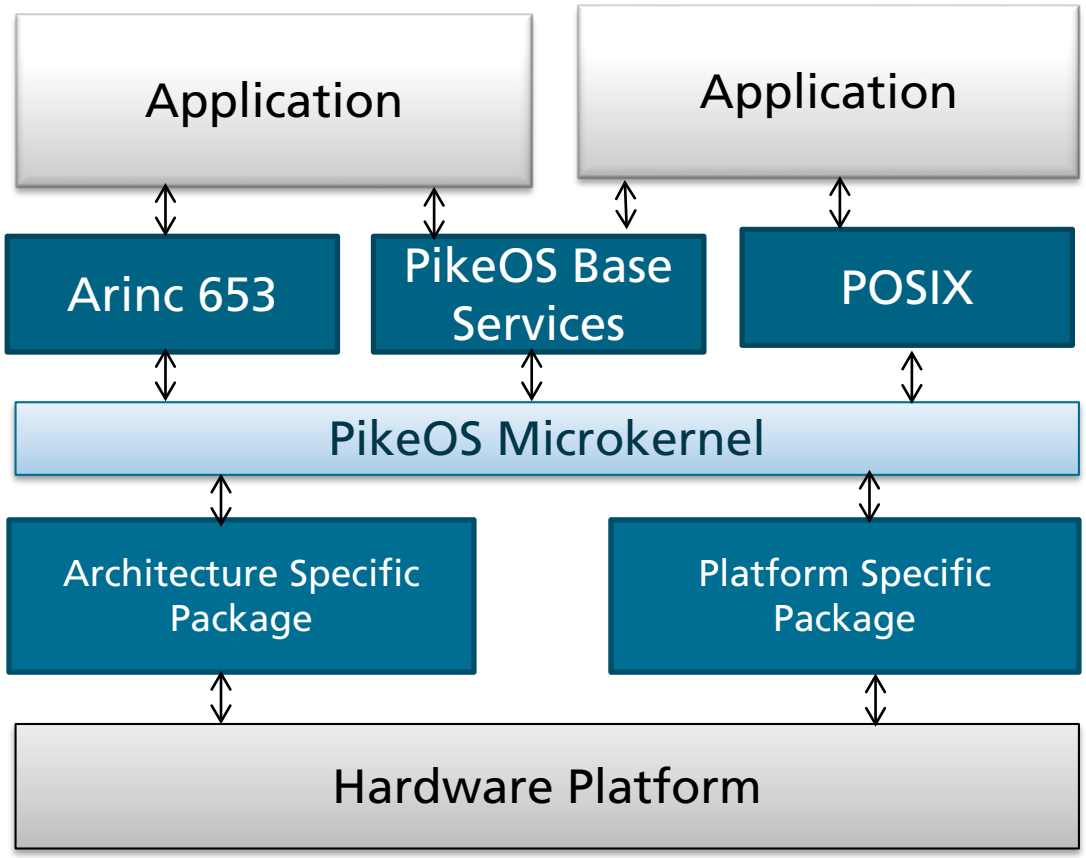
The Microkernel Style

- **Problem/Context**
  - Several applications use same core functionality
  - Core functionality implementation should be safe and maintainable
  - Core functionality is expected to remain stable
  - System should be still modifiable
- **Solution**
  - Separate a minimal functional core from extended services
  - Microkernel (stable, maintainable)
  - External services (application specific)
  - Internal services (platform specific)



© Fraunhofer IESE

The Microkernel Style Example: PikeOS



© Fraunhofer IESE

The Microkernel Style

- **Advantages**
  - Maintainable / modular
  - Ease of adding / removing services
  - Separation of policy and mechanisms
- **Drawbacks**
  - Performance decreases (compared to monolithic system)
  - Complexity of design and implementation increase

© Fraunhofer IESE

The Microkernel Style

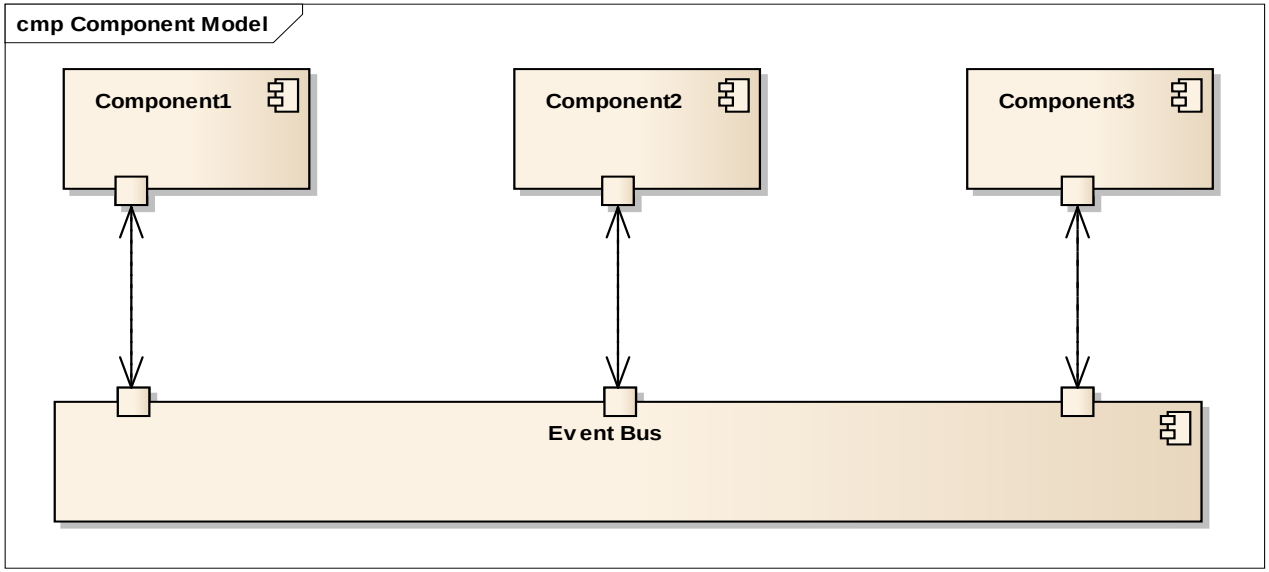
Pattern ID	Microkernel
Problem/ Context	▪ Several applications use same core functionality ▪ Core functionality implementation: safe and maintainable ▪ Core functionality is expected to remain stable ▪ System should be still modifiable
Solution	Separate a minimal functional core from extended functionality and customer-specific parts
Advantages	▪ Maintainable / modular ▪ Ease of adding / removing services ▪ Separation of policy and mechanisms
Drawbacks	▪ Performance decreases (compared to monolithic system) ▪ Complexity of design and implementation increase
Impact on Safety	▪ Minimum code base eases qualification. ▪ Reuse of certified kernels. ▪ Expand upon infrastructural safety mechanisms (e.g. safety kernel). ▪ Common cause of failures / single point of failure of kernel.
Example	PikeOS, EB tresos Safety OS, Linux Kernel,…

© Fraunhofer IESE

The Event-Bus Style

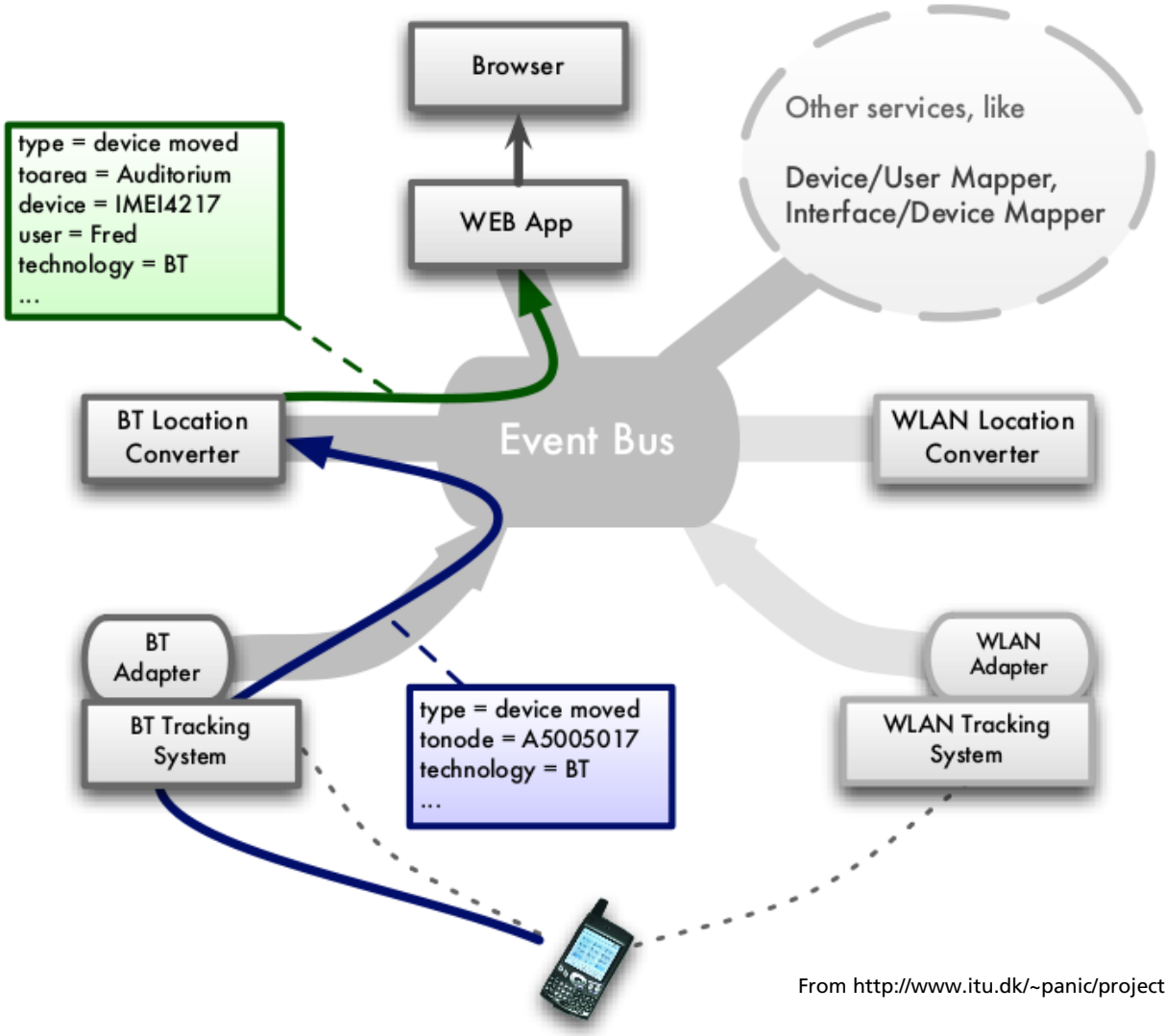
- **Problem/Context**
  - Distributed system where many components communicate
  - Connections between each communicating component not feasible/ desirable

- **Solution**
  - Components asynchronously emit and receive events over event busses
  - Components communicate with the event-bus, not directly with each other



© Fraunhofer IESE

The Event-Bus Style Example: Android



From http://www.itu.dk/~panic/projects/SeamlessTracking.html

© Fraunhofer IESE

The Event-Bus Style

- **Advantages**
  - Easy to add new components to the system
- **Drawbacks**
  - No guarantee that a message arrives at the destination
  - Message delay possible

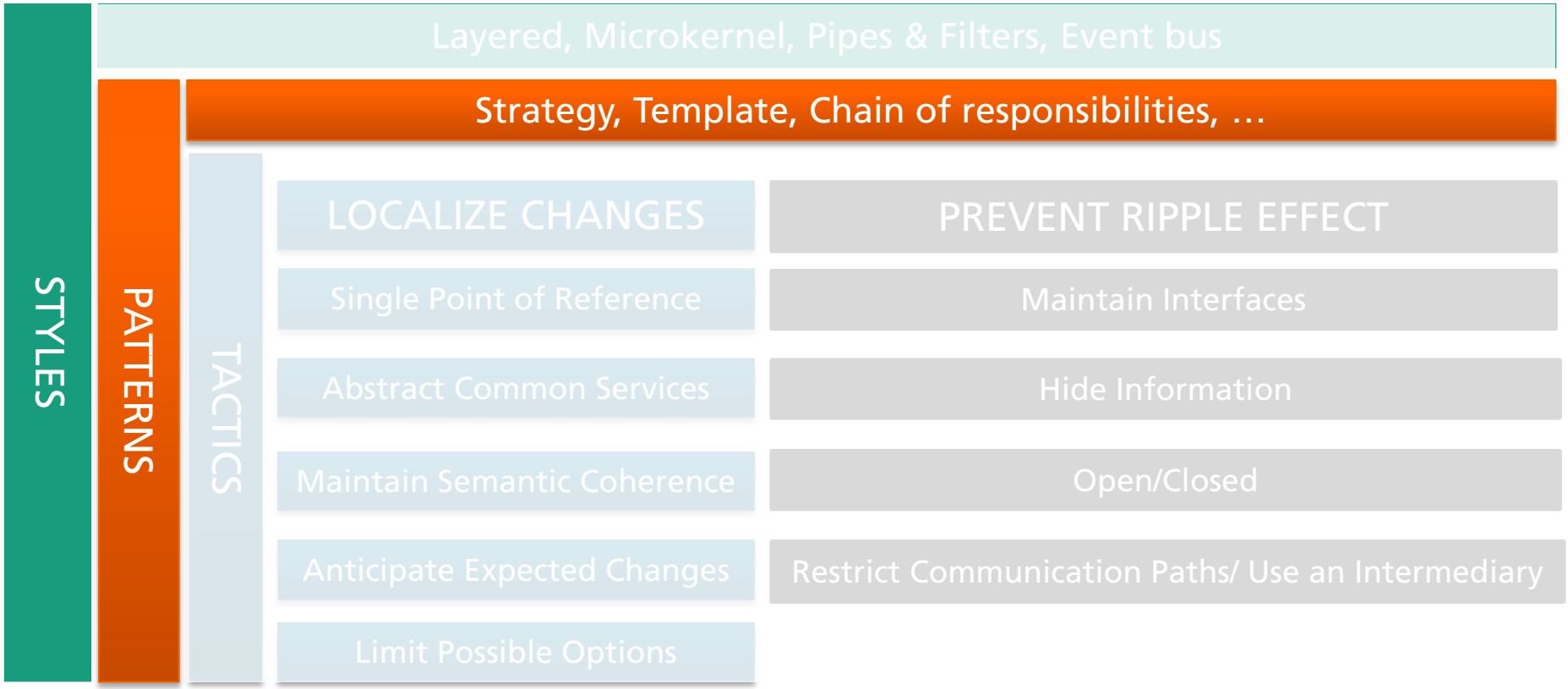
© Fraunhofer IESE

The Event-Bus Style

Pattern ID	Event-Bus
Problem	▪ Distributed system where many components communicate ▪ Connections between each communicating component not feasible
Solution	▪ Components asynchronously emit and receive events over event busses ▪ Components communicate with the event-bus, not with each other
Advantages	▪ Easy to add new components to the system
Drawbacks	▪ No guarantee that a message arrives at the destination ▪ Message delay possible
Impact on Safety	▪ Entry point for hooking safety mechanisms to failure events ▪ Separates failure detection from failure recovery. ▪ Manage redundancies using activation/deactivation events. ▪ Dynamic execution complicates analysis. ▪ CCF/SPF of the bus.
Example	Satellite constellation communication, Event Bus in android

© Fraunhofer IESE

Styles and patterns

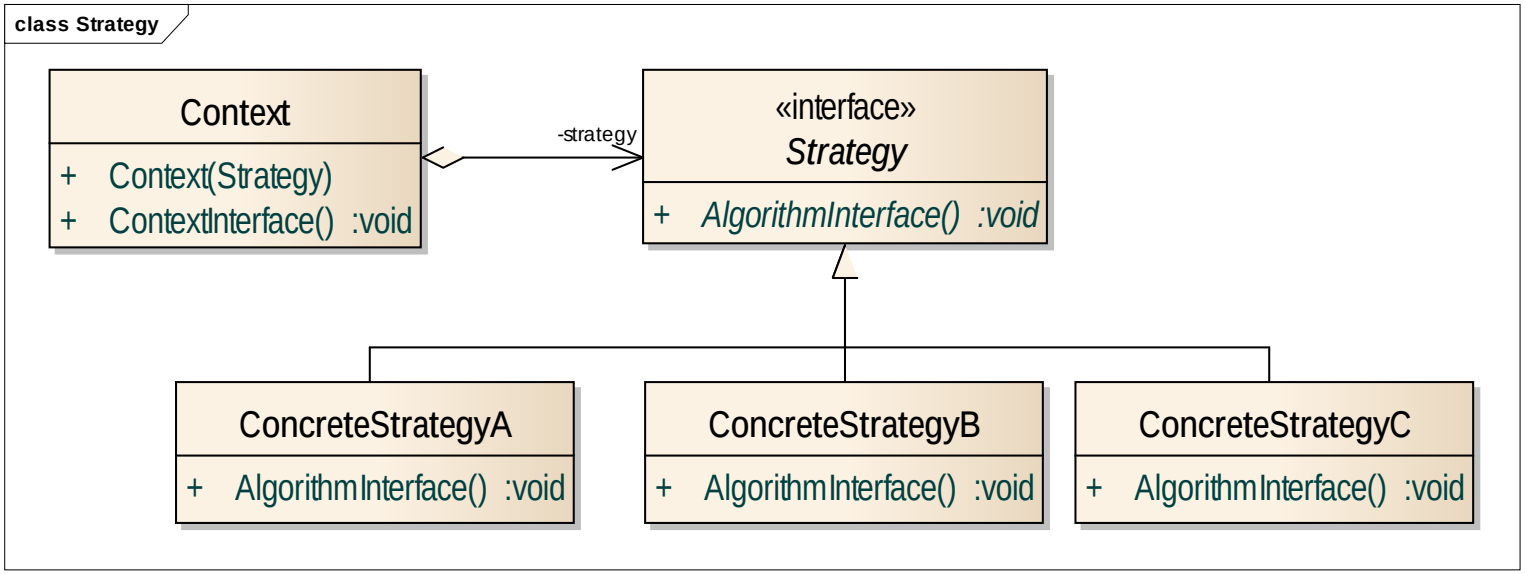


© Fraunhofer IESE

Fraunhofer IESE

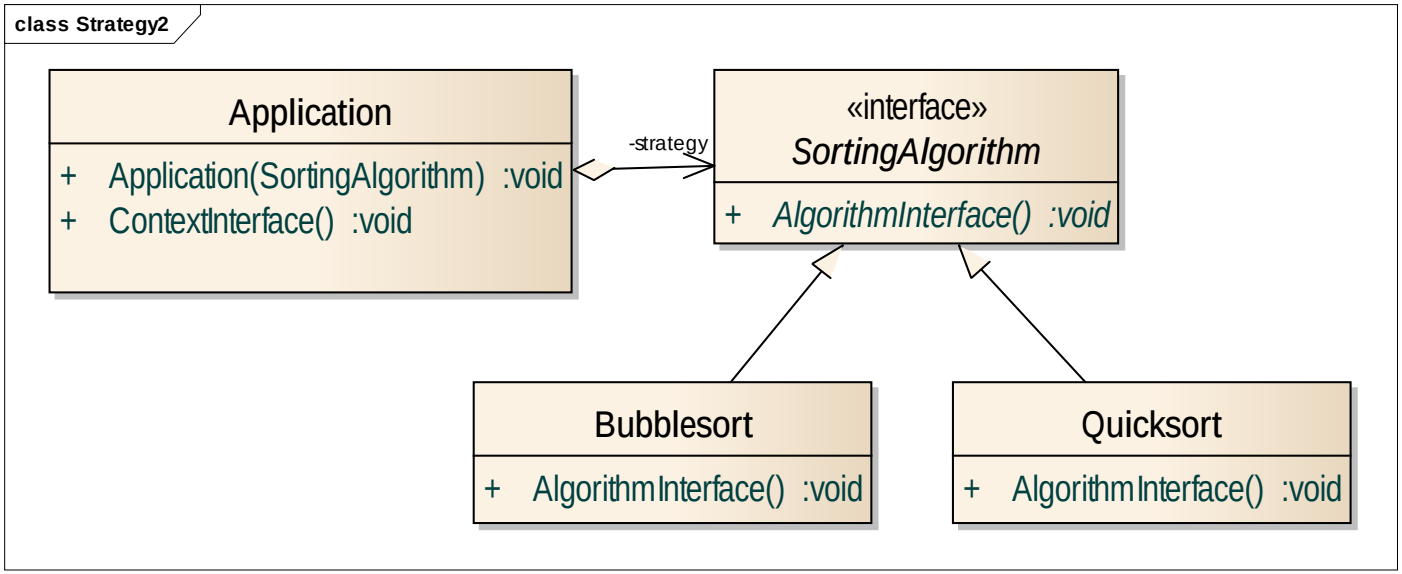
The Strategy Pattern

- **Problem**
  - Different variants of an algorithm are needed
  - Selection of variant depends on the context
- **Solution**
  - Define an interface for a family of algorithms to make them interchangeable



© Fraunhofer IESE

The Strategy Pattern Example: Sorting Algorithm



- Bubble Sort: (-) slower (+) no additional memory
- Quick Sort : (+) faster (–) additional memory
- Compile time instantiation: depending on system properties
- Runtime instantiation: depending on currently available memory

© Fraunhofer IESE

The Strategy Pattern

- **Advantages**
  - Strategies can provide different implementations of the same behavior with a clean interface
  - Client can choose among implementations with different time and space trade-offs
  - Strategy eliminates conditional statements that are hard to maintain
  - Strategies can be selected at runtime

- **Drawbacks**
  - Client code must be aware of different strategies
  - Strategies should implement **same** behavior, otherwise functional outcome depends on instantiation and the overall behavior is hard to predict
  - Communication overhead

© Fraunhofer IESE

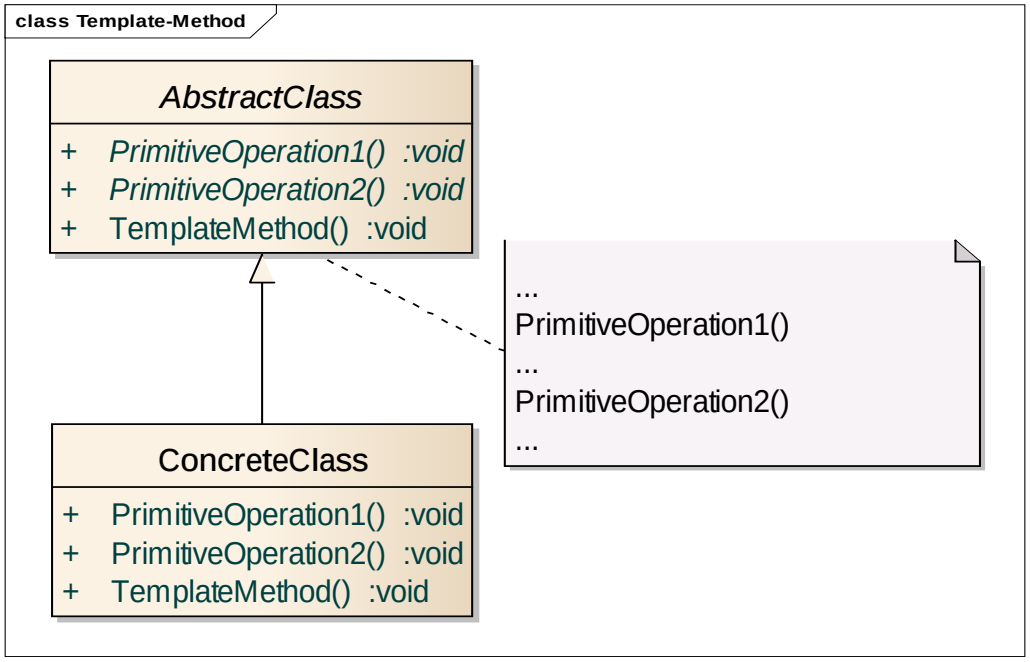
The Strategy Pattern

Pattern ID	Strategy
Problem	▪ Different variants of an algorithm are needed ▪ Selection of variant depends on the context
Solution	▪ Define an interface for a family of algorithms to make them interchangeable
Advantages	▪ Strategies can provide different implementations of the same behavior with a clean interface ▪ Client can choose among implementations with different time and space trade-offs ▪ Strategy eliminates conditional statements that are hard to maintain ▪ Strategies can be selected at runtime
Drawbacks	▪ Client code must be aware of different strategies ▪ Strategies should implement same behavior, otherwise functional outcome depends on instantiation ▪ Communication overhead
Impact on Safety	▪ Entry point for expressing (safe) mode-dependent algorithms as different strategies. ▪ Eases qualification by facilitating a per-strategy (e.g. per-mode) safety analysis. ▪ Dynamic execution binding complicates analysis.
Example	Selecting sorting functions based on free memory availability

© Fraunhofer IESE

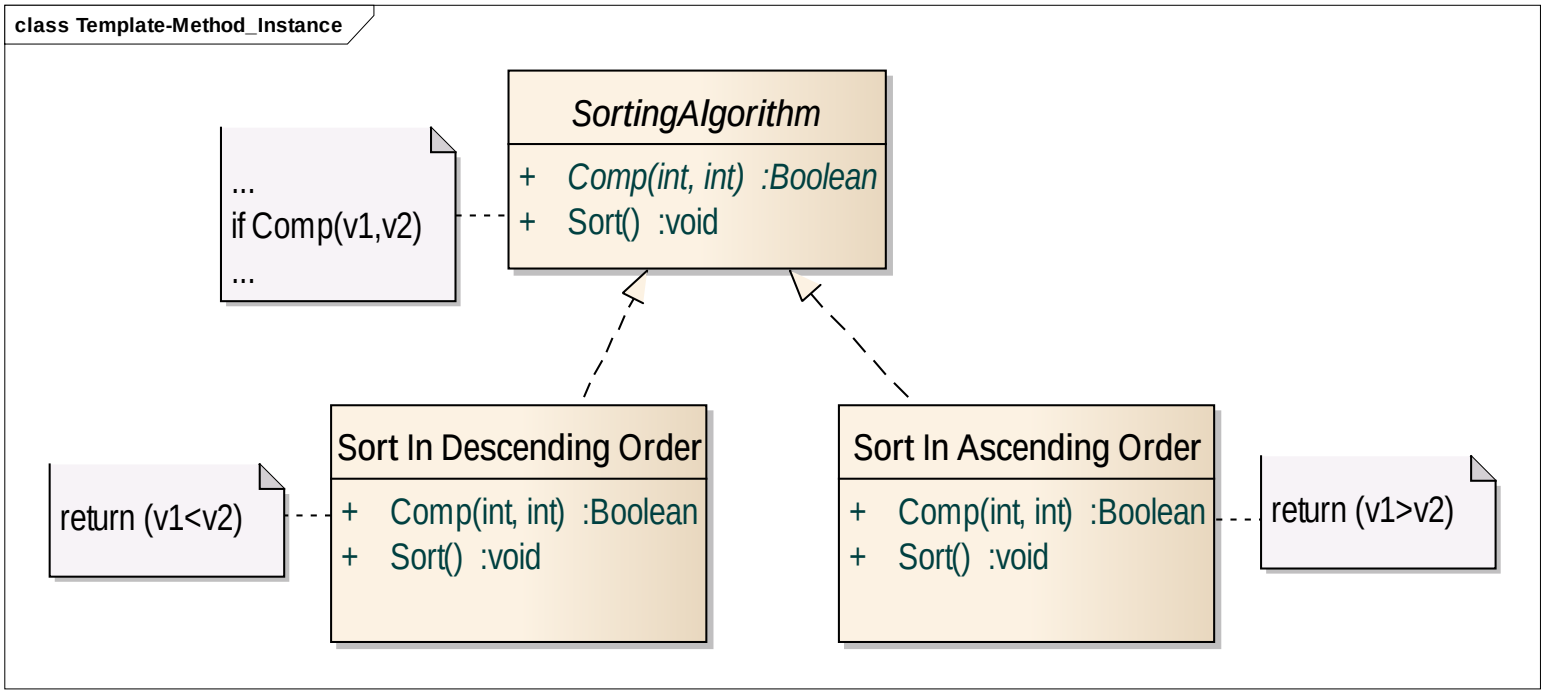
The Template Method Pattern

- **Problem**
  - Different variants of an algorithm are needed
  - Variants differ only in small steps
- **Solution**
  - Define a skeleton of an algorithm in a function, deferring some steps to other functions that may vary



© Fraunhofer IESE

The Template Method Pattern Example: Sorting



- Difference to Strategy
  - in scope: strategy replaces whole algorithm, template-method only parts
  - intention: strategy intended to select algorithm, template-method to build
- Concrete strategies may be implemented using template-method !

© Fraunhofer IESE

The Template Method Pattern

- **Advantages**
  - Template Method eliminates conditional statements that are hard to maintain
  - Supports Code Reuse
- **Drawbacks**
  - Must be well documented, otherwise hard to understand

© Fraunhofer IESE

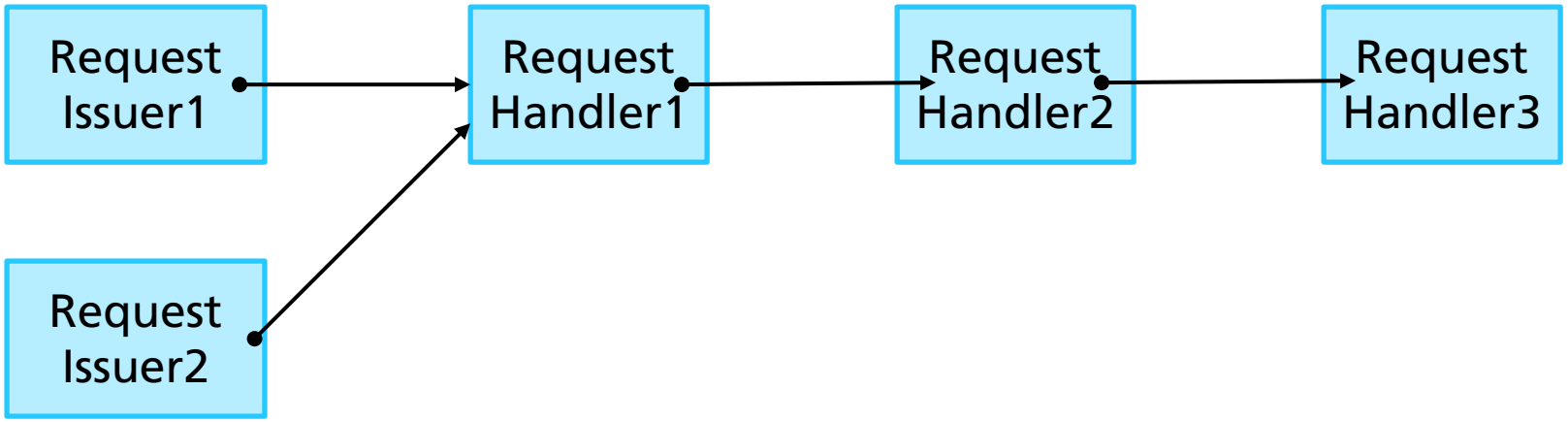
The Template Method Pattern

Pattern ID	Template Method
Problem	▪ Different variants of an algorithm are needed ▪ Variants differ only in small steps
Solution	Define a skeleton of an algorithm in a function, deferring some steps to other functions that may vary
Advantages	▪ Template method eliminates conditional statements that are hard to maintain ▪ Supports code reuse
Drawbacks	Must be well documented, otherwise hard to understand
Impact on Safety	n.a.
Example	Sorting in descending/ascending order

© Fraunhofer IESE

The Chain of Responsibility Pattern

- **Problem**
  - A request (like error handling) is issued but it is unclear at which part of the system the request should be handled
- **Solution**
  - Request are passed along a chain of possible handlers until one handles it



© Fraunhofer IESE

Request Communication Path

The Chain of Responsibility Pattern Example: Satellite Fault Management

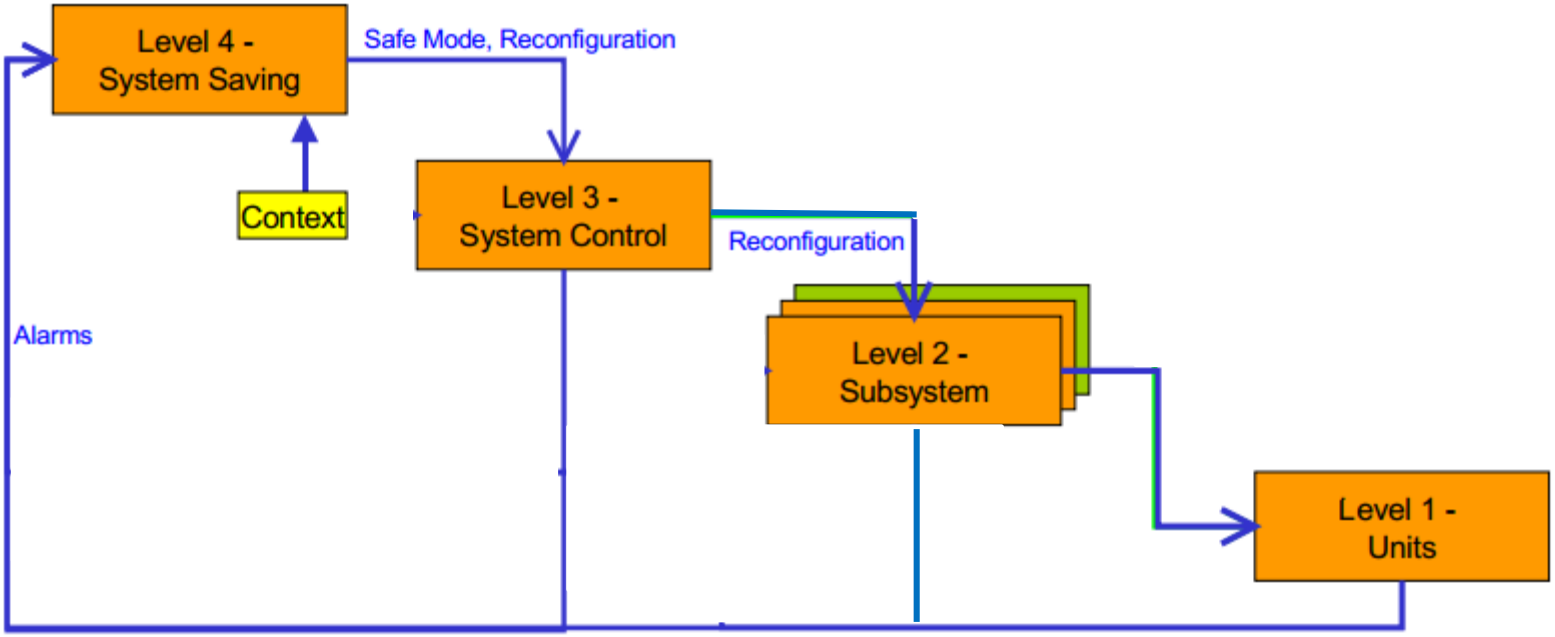


Figure 4: FDIR Hierarchical Breakdown

© Fraunhofer IESE

The Chain of Responsibility Pattern

- **Advantages**
  - Reduced coupling
  - Added flexibility for handling requests
- **Drawbacks**
  - No guarantee that request is handled if not pre-planned

© Fraunhofer IESE

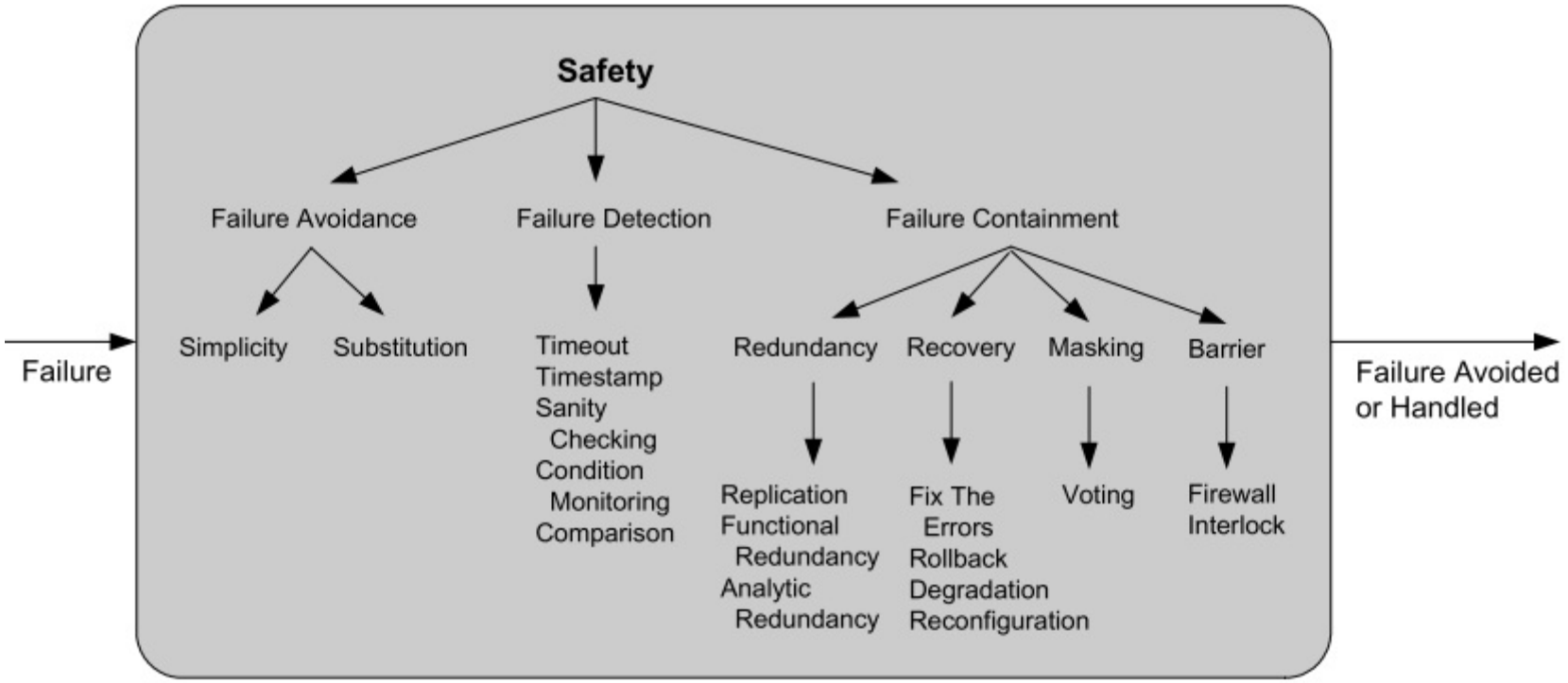
The Chain of Responsibility Pattern

Pattern ID	Chain of Responsibility
Problem	A request (like error handling) is issued but it is unclear at which part of the system the request should be handled
Solution	Request are passed along a chain of possible handlers until one handles it
Advantages	▪ Reduced coupling ▪ Added flexibility for handling requests
Drawbacks	No guarantee that request is handled if not pre-planned
Impact on Safety	▪ Organizational scheme for managing failures across crosscutting abstractions or levels, i.e. classical way for implementing fault management. ▪ Dynamic execution binding complicates analysis.
Example	Satellite fault management

© Fraunhofer IESE

Safety

Safety Tactics



Safety Tactics for Software Architecture Design
W. Wu and T. Kelly

Safety Pattern

Redundancy

- Homogeneous Redundancy
- Triple Modular Redundancy

Pattern for Error Detection

- Monitor-Actuator Pattern
- Watchdog

Achieving Quality Attributes

Redundancy

- Homogeneous Redundancy
- Triple Modular Redundancy

Pattern for Error Detection

- Monitor-Actuator Pattern
- Watchdog

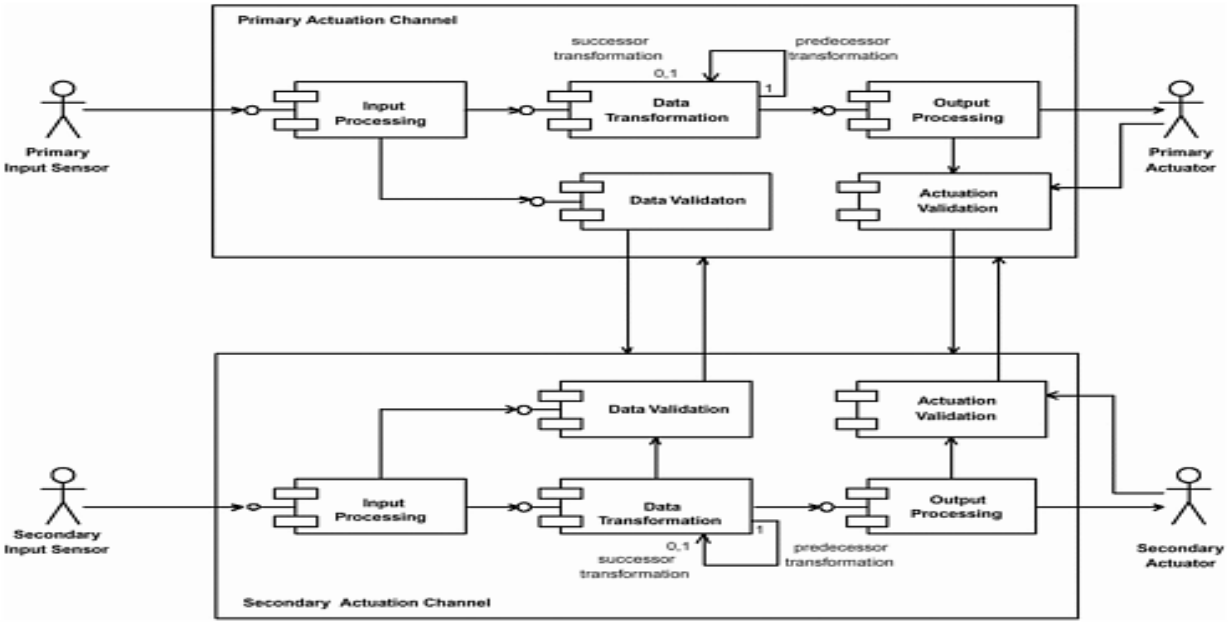
Homogeneous Redundancy Pattern (1/3)

"An obvious approach to solving the problem of things breaking is to provide multiple copies of that thing." B.P. Douglass

Pattern goal

- Improve reliability by offering multiple channels that can operate in sequence or in parallel.

Homogeneous Redundancy Pattern (2/3)



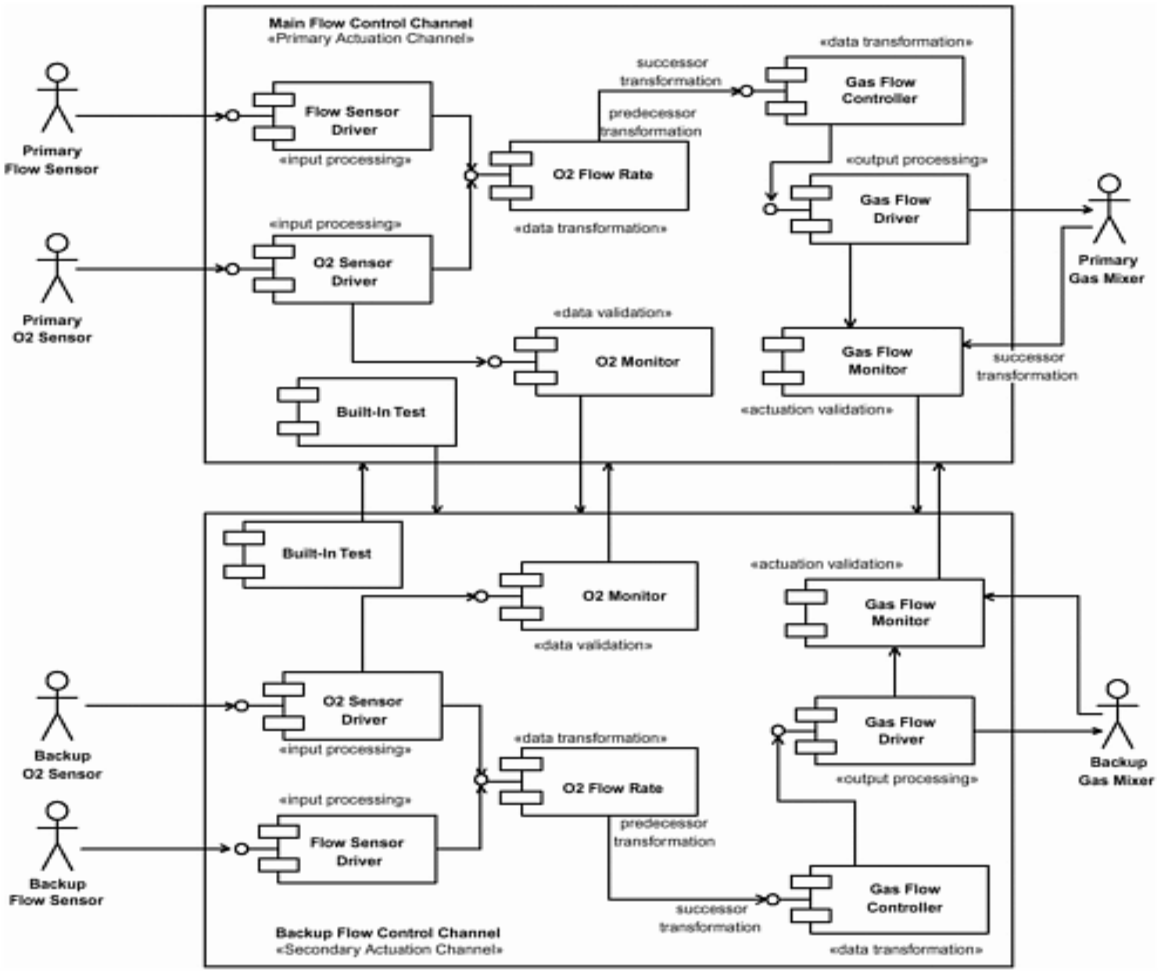
Solution

- Improves reliability by addressing random faults (failures) in the system execution.

Drawback

- Any systematic fault in one copy of the system is replicated in its clones, thus, not proving, protection against systematic faults (errors).

Homogeneous Redundancy Pattern (3/3)



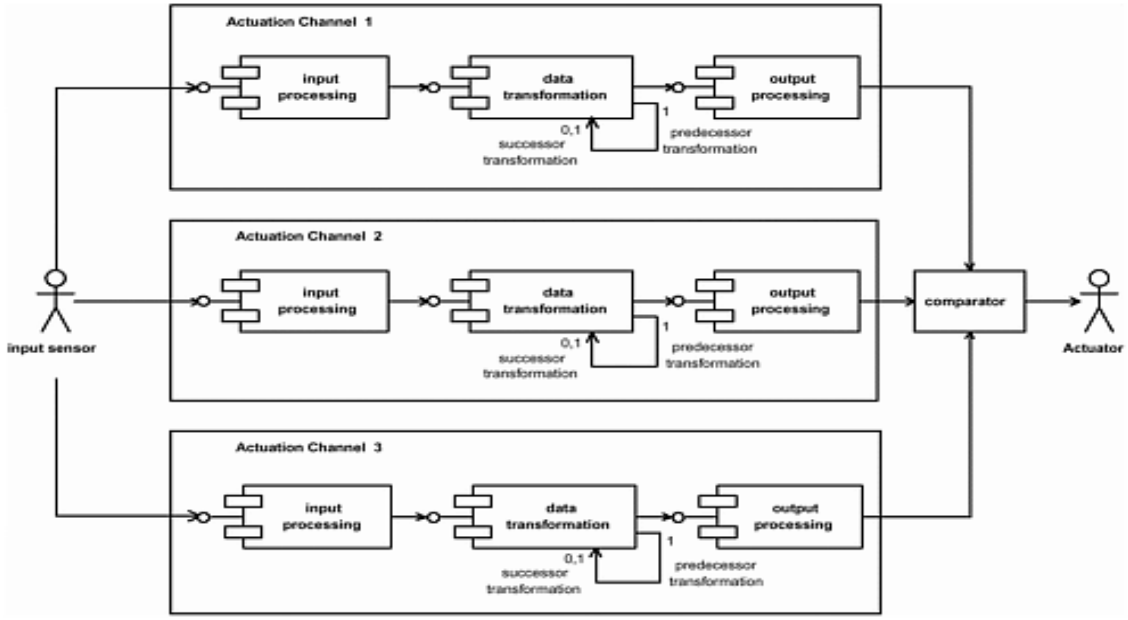
Triple Modular Redundancy (1/2)

Pattern goal

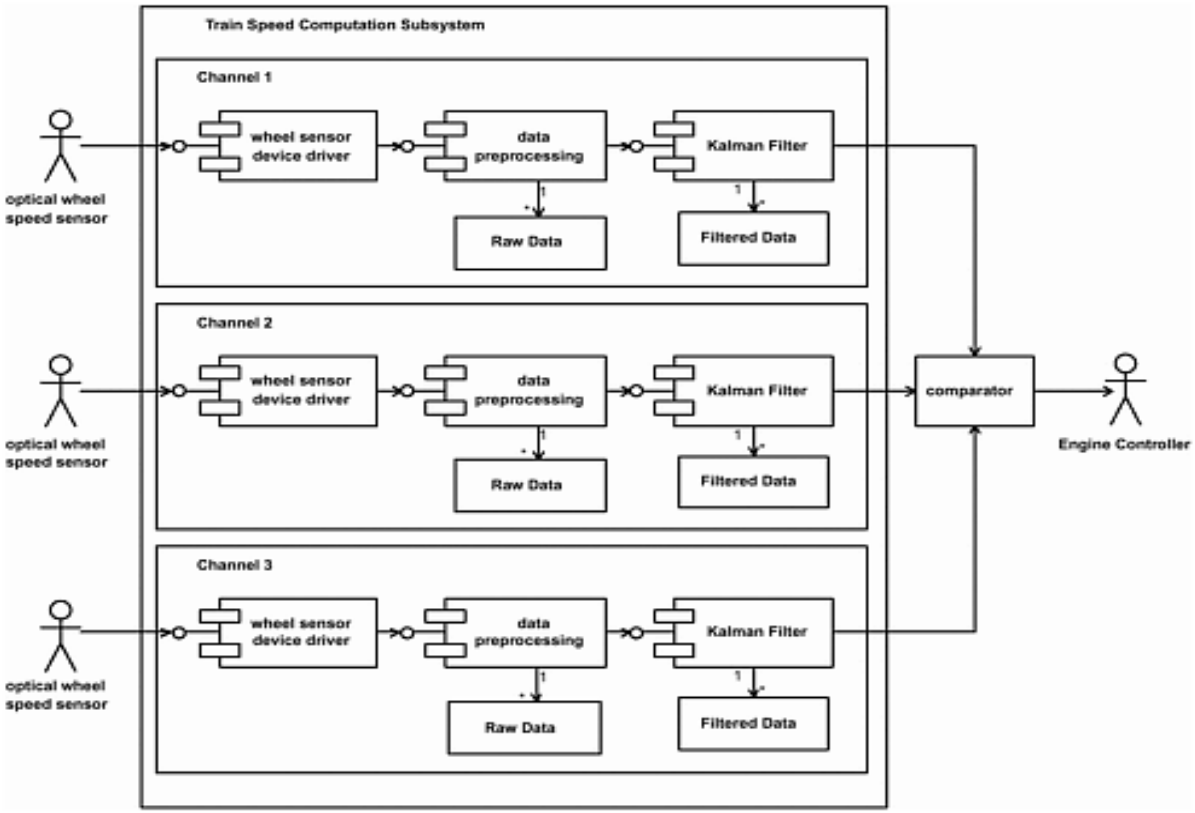
- Enhance reliability and safety in situations where there is no **fail-safe state**.

Solution

- Odd number of channels operating in parallel, each in effect checking the results of all the others.
- The computational results or resulting actuation signals are compared, and if there is a disagreement, then a *"two-out-of-three majority wins"* policy is invoked.



Triple Modular Redundancy (2/2)



Drawbacks

- High recurring cost because the hardware and software in the channels must be replicated.

Achieving Quality Attributes

Redundancy

- Homogeneous Redundancy
- Triple Modular Redundancy

Tactics for Error Detection

- Monitor-Actuator Pattern
- Watchdog

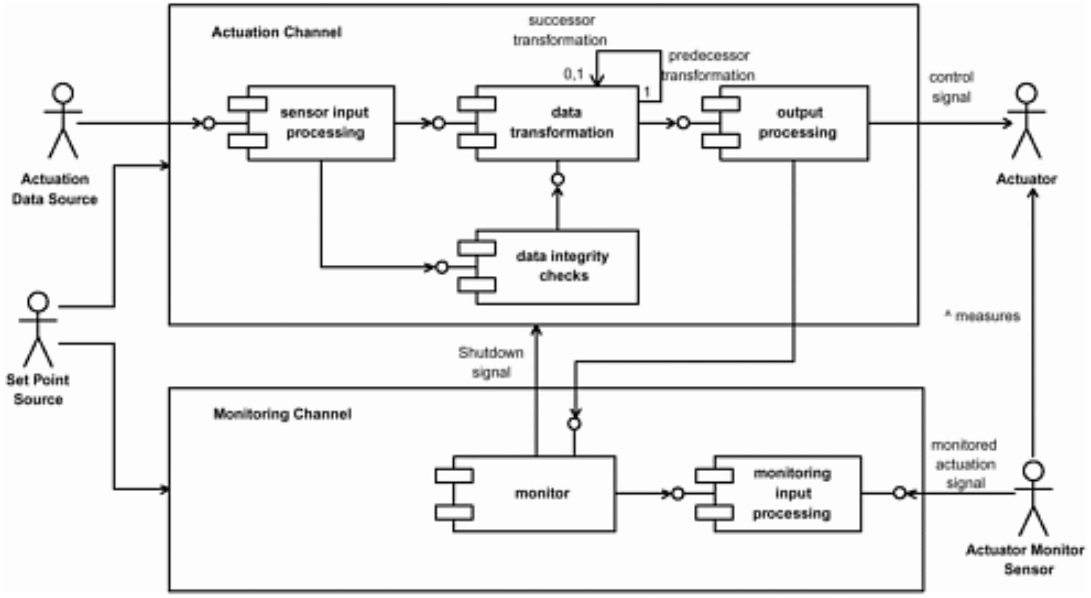
Monitor-Actuator (1/2)

Pattern goal

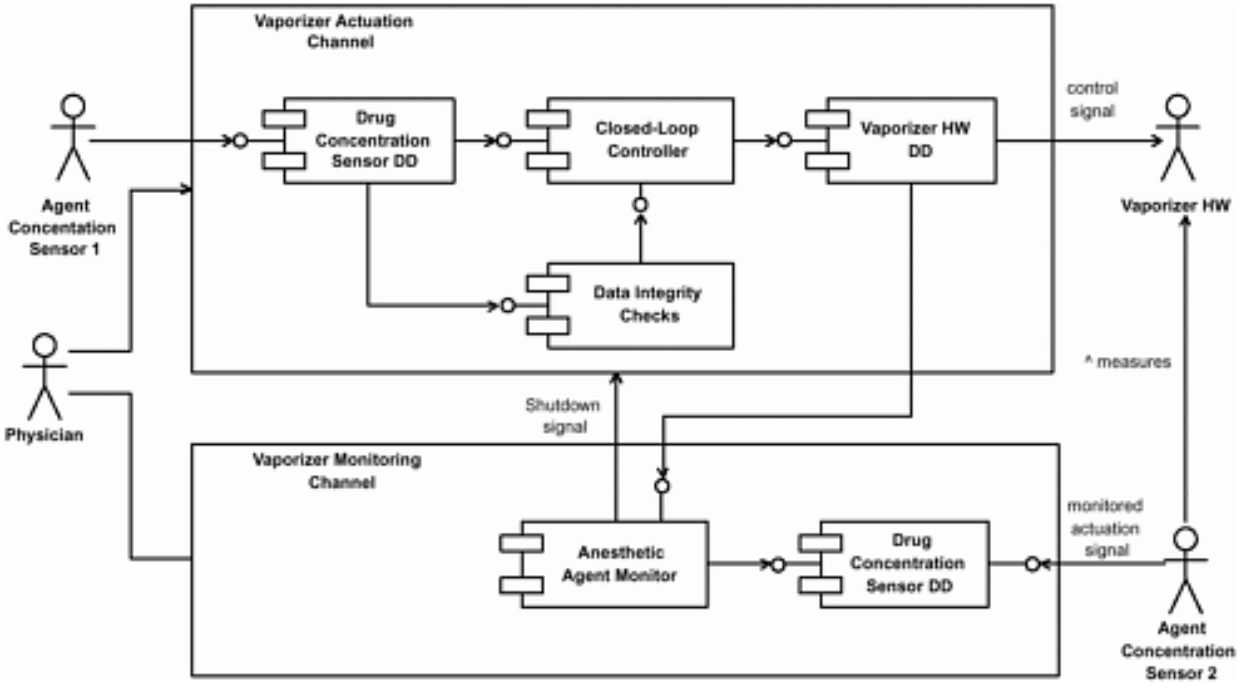
- Improve safety in a system with moderate to low availability requirements at a low cost.

Solution

- Redundant channel differs from the primary actuation channel by providing monitoring.
- *Monitor* maintains a watch on the actuation channel looking for an indication that the system should be commanded into its fail-safe state.



Monitor-Actuator (2/2)



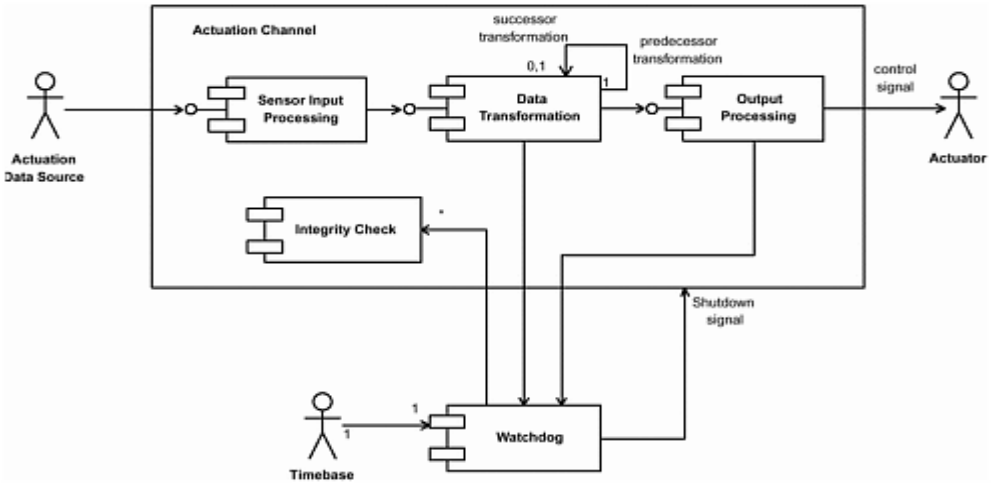
Drawback

- Because there is minimal redundancy, the system cannot continue to function when a fault is identified.

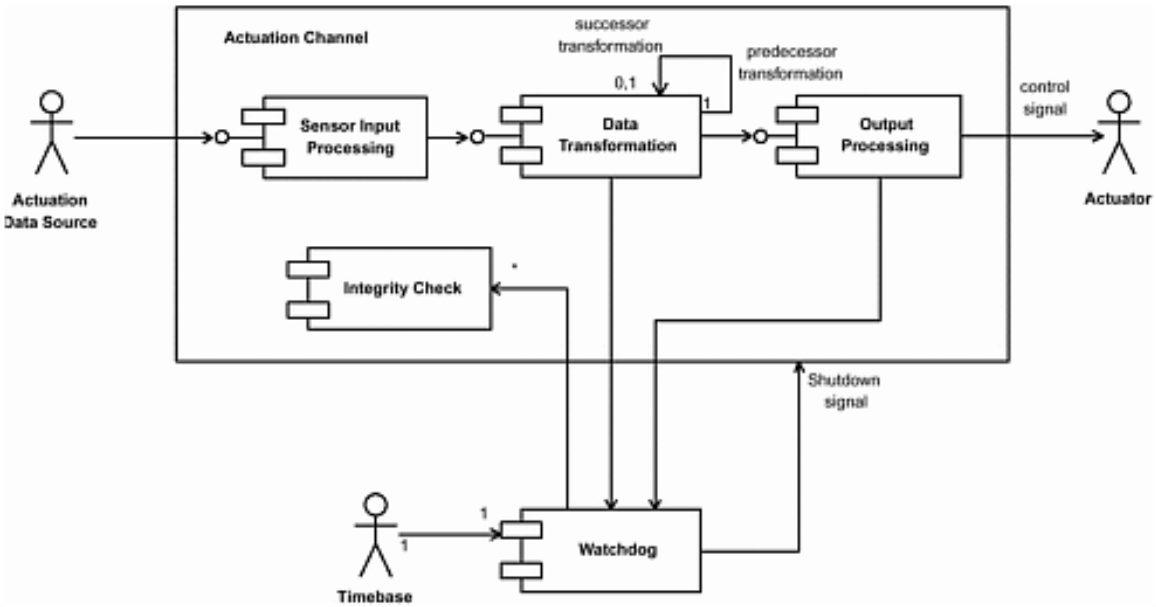
Watchdog (1/3)

Pattern goal

- Checks that the internal computational processing is proceeding as expected.
  - Check a computation timebase;
  - Ensure that computation steps are proceeding in a predefined order.



Watchdog (2/3)



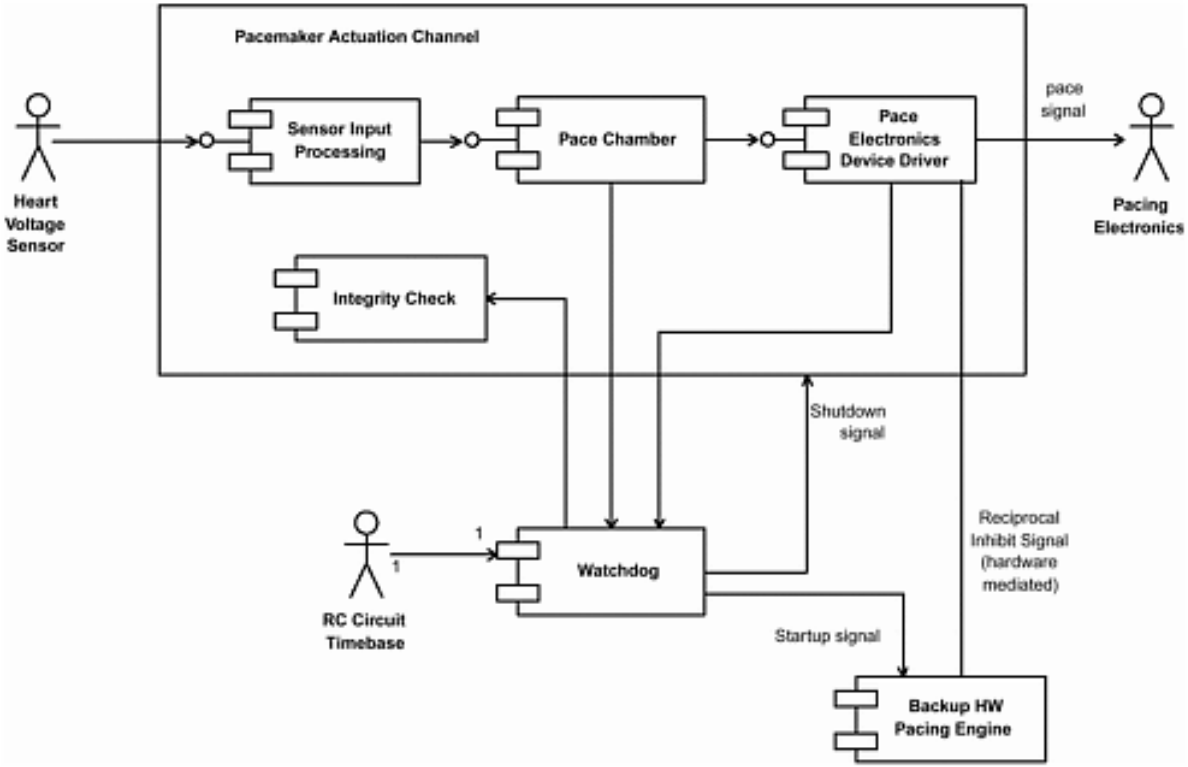
Solutions

- Watches out over processing of another component, ensuring that nothing is *obviously* wrong.
- The *Actuator Channel* operates independently of the watchdog, often sending a *liveness* message to the watchdog.

Drawback

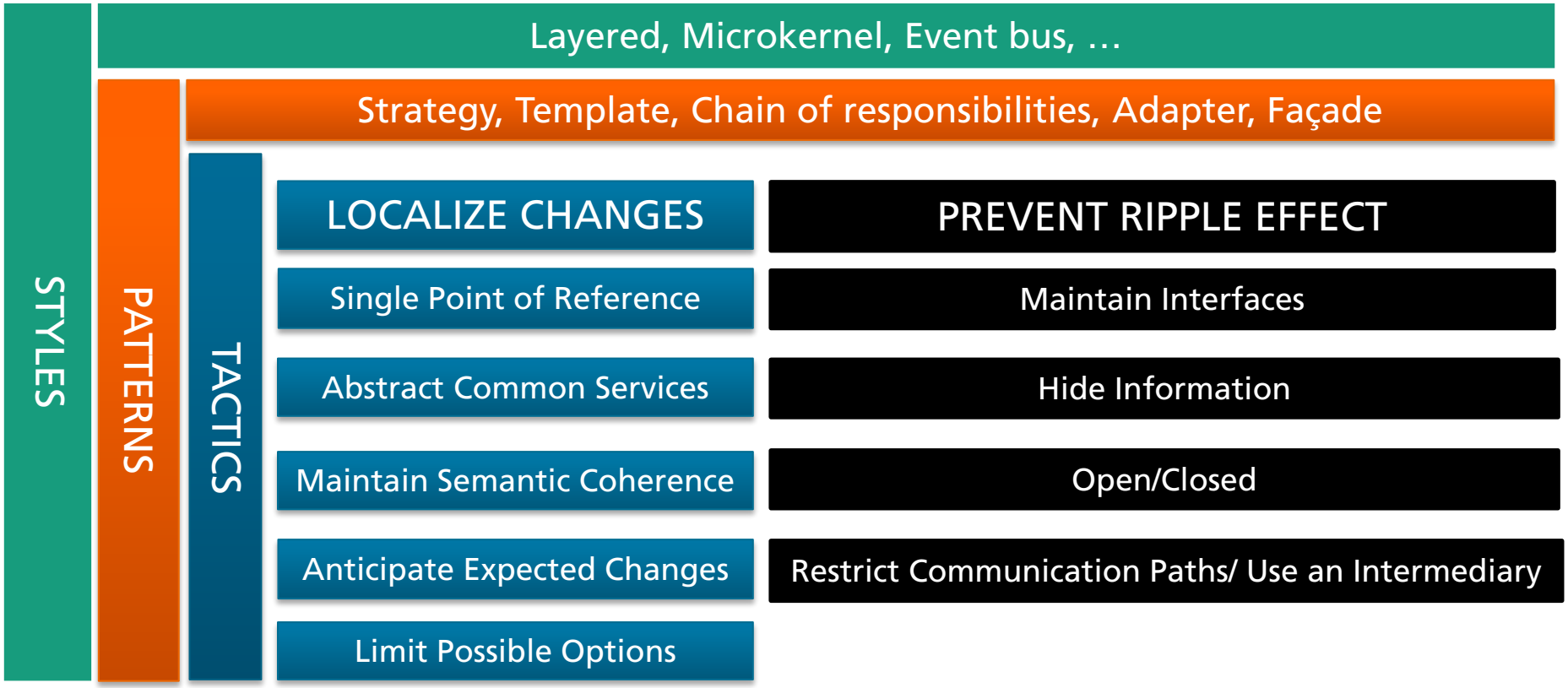
- Because of the minimal coverage of the Watchdog Pattern, it is rarely used alone, but combined with other tactics.

Watchdog (3/3)

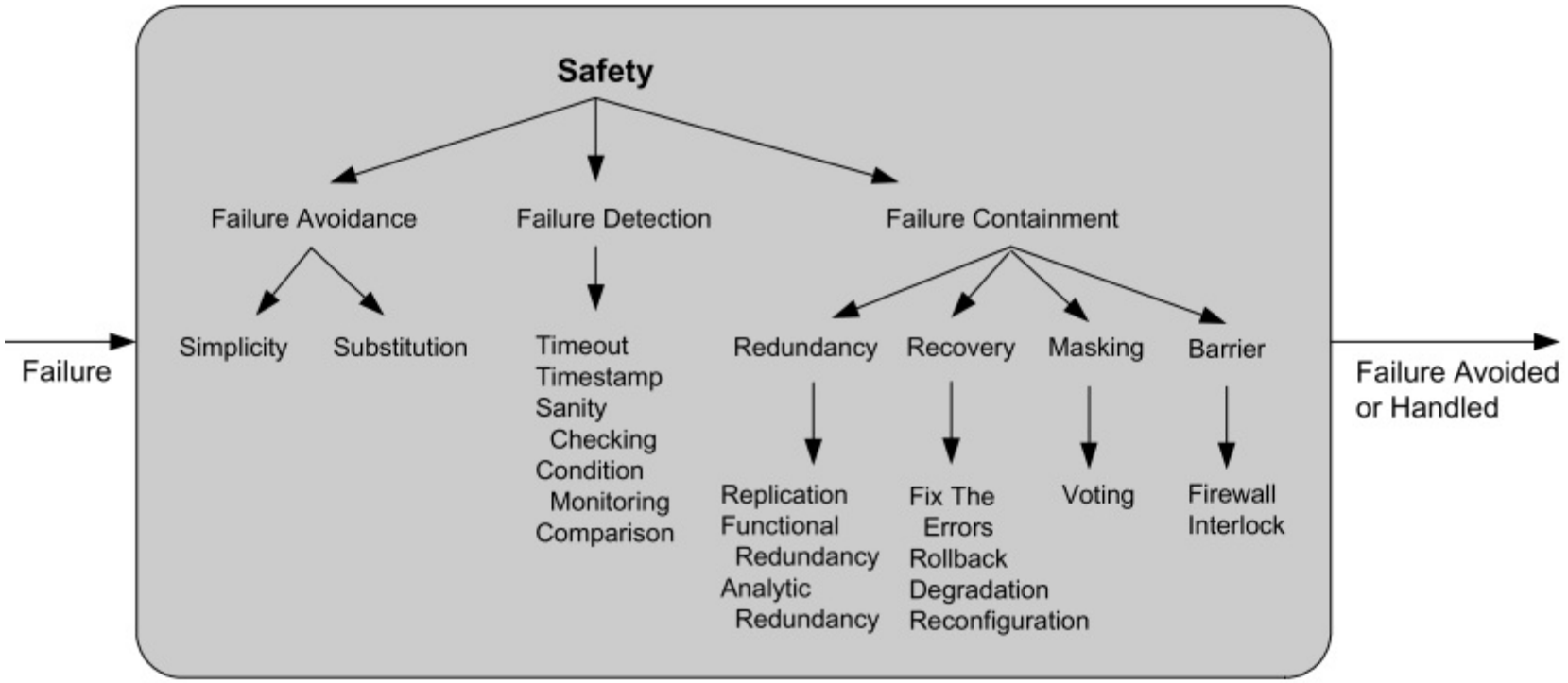

Pacemaker Actuation Channel
Heart Voltage Sensor
Sensor Input Processing
Pace Chamber
Pace Electronics Device Driver
pace signal
Pacing Electronics
Integrity Check
Shutdown signal
RC Circuit Timebase
1
1
Watchdog
Reciprocal Inhibit Signal (hardware mediated)
Startup signal
Backup HW Pacing Engine

Wrap Up

Architecture Tactics, Styles and Patterns: Maintainability



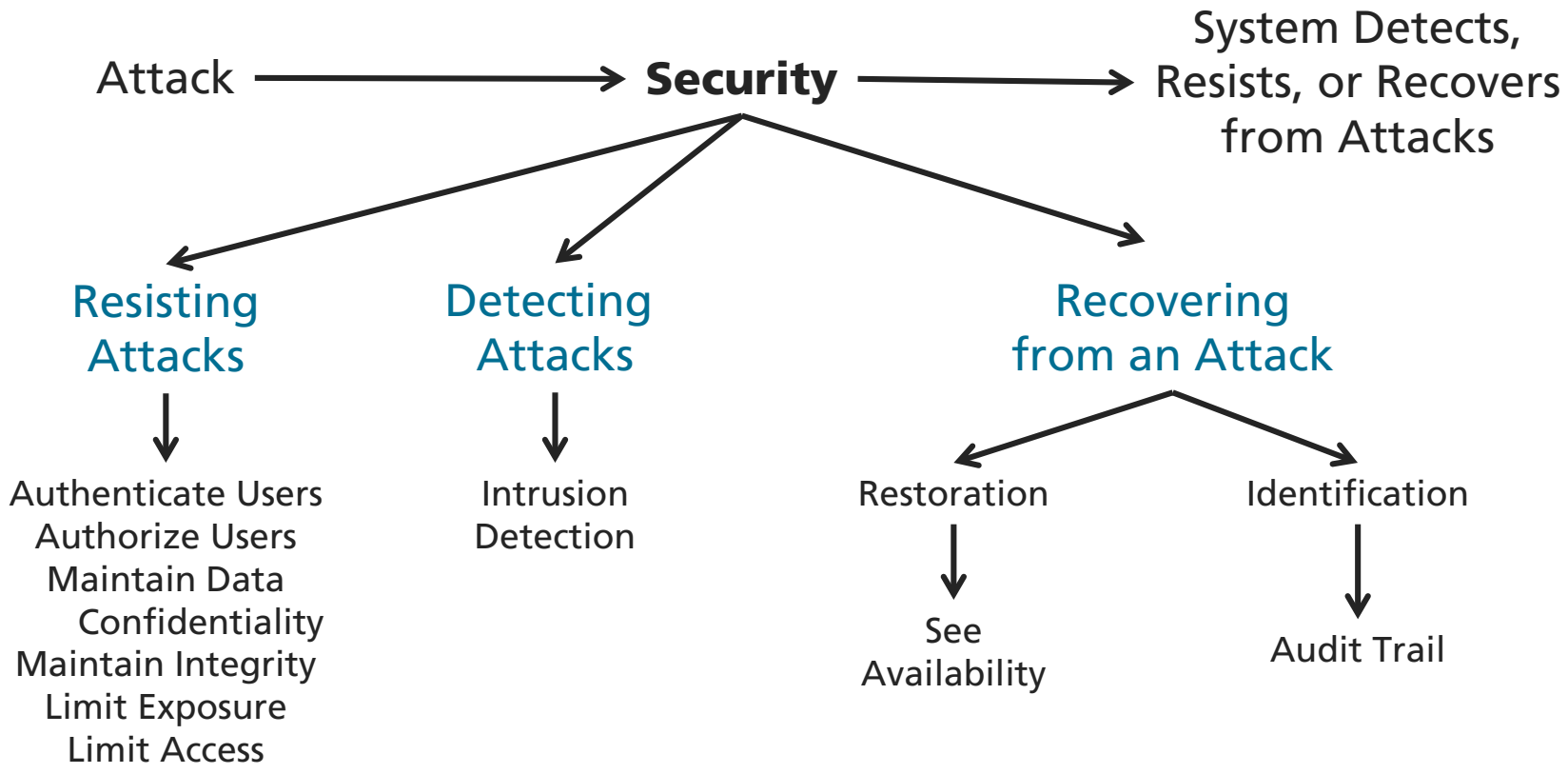
Safety Tactics



Safety Tactics for Software Architecture Design
W. Wu and T. Kelly

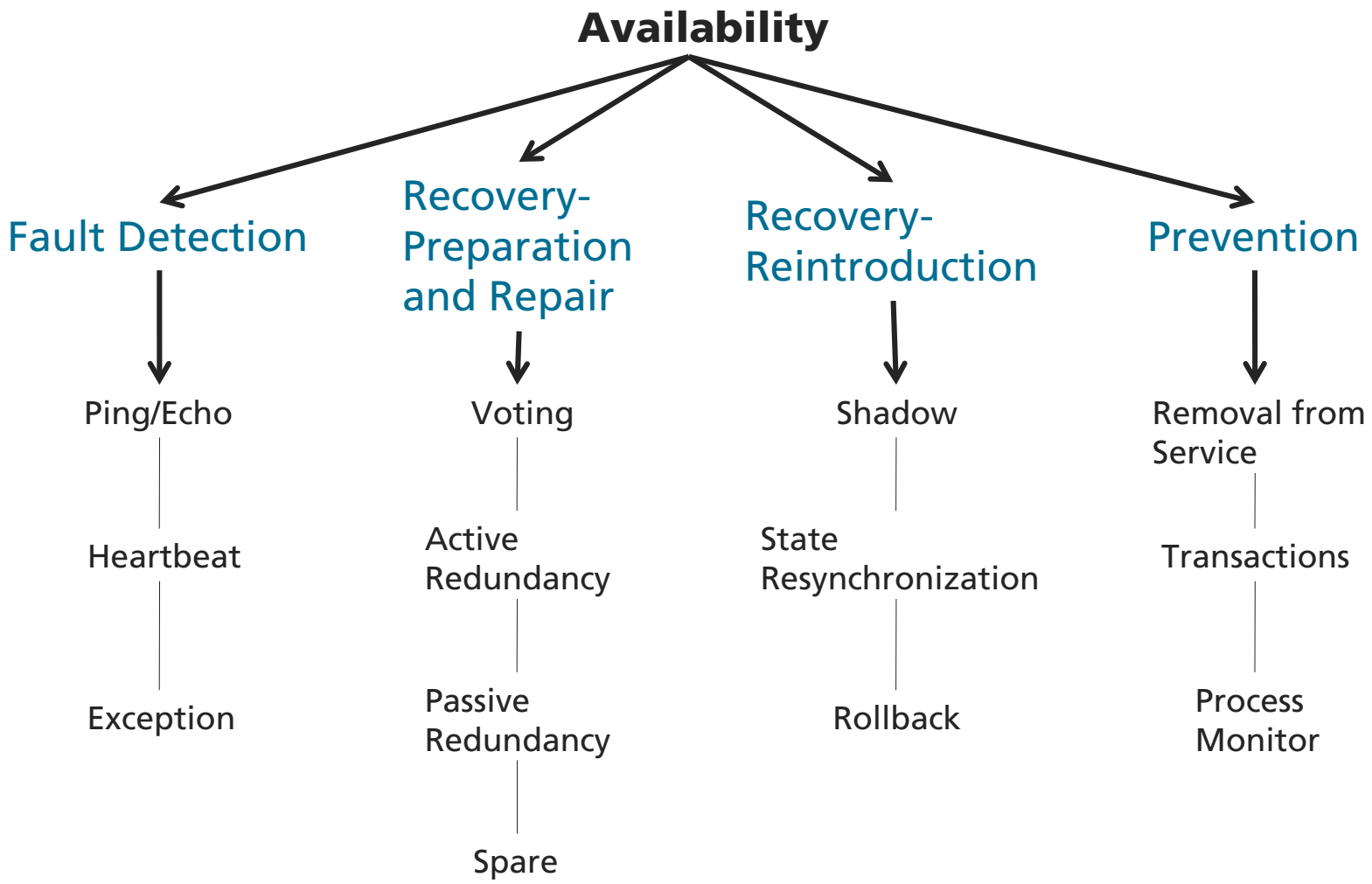
Tactics for other Quality Attributes

Security Tactics



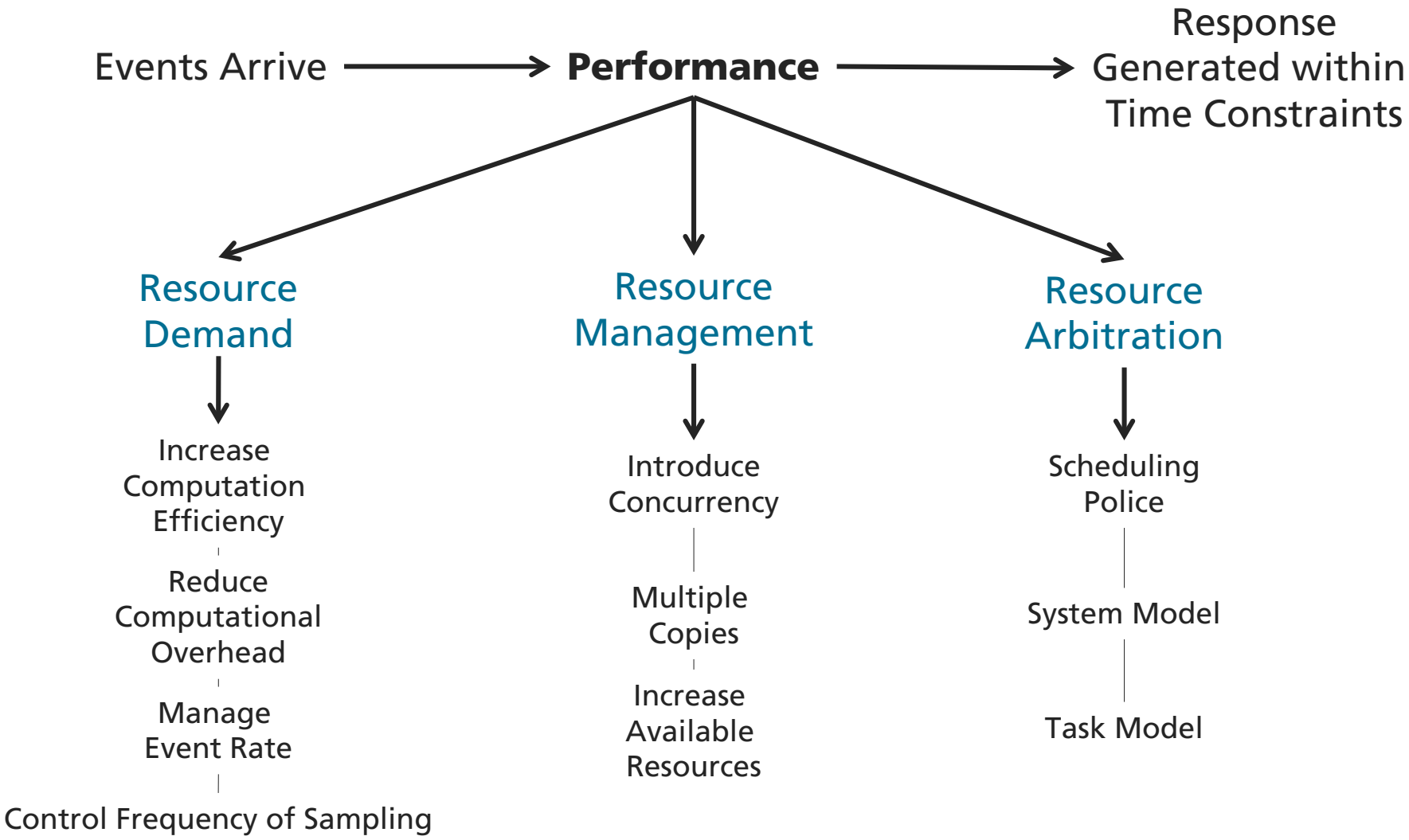
© Fraunhofer IESE

Availability Tactics



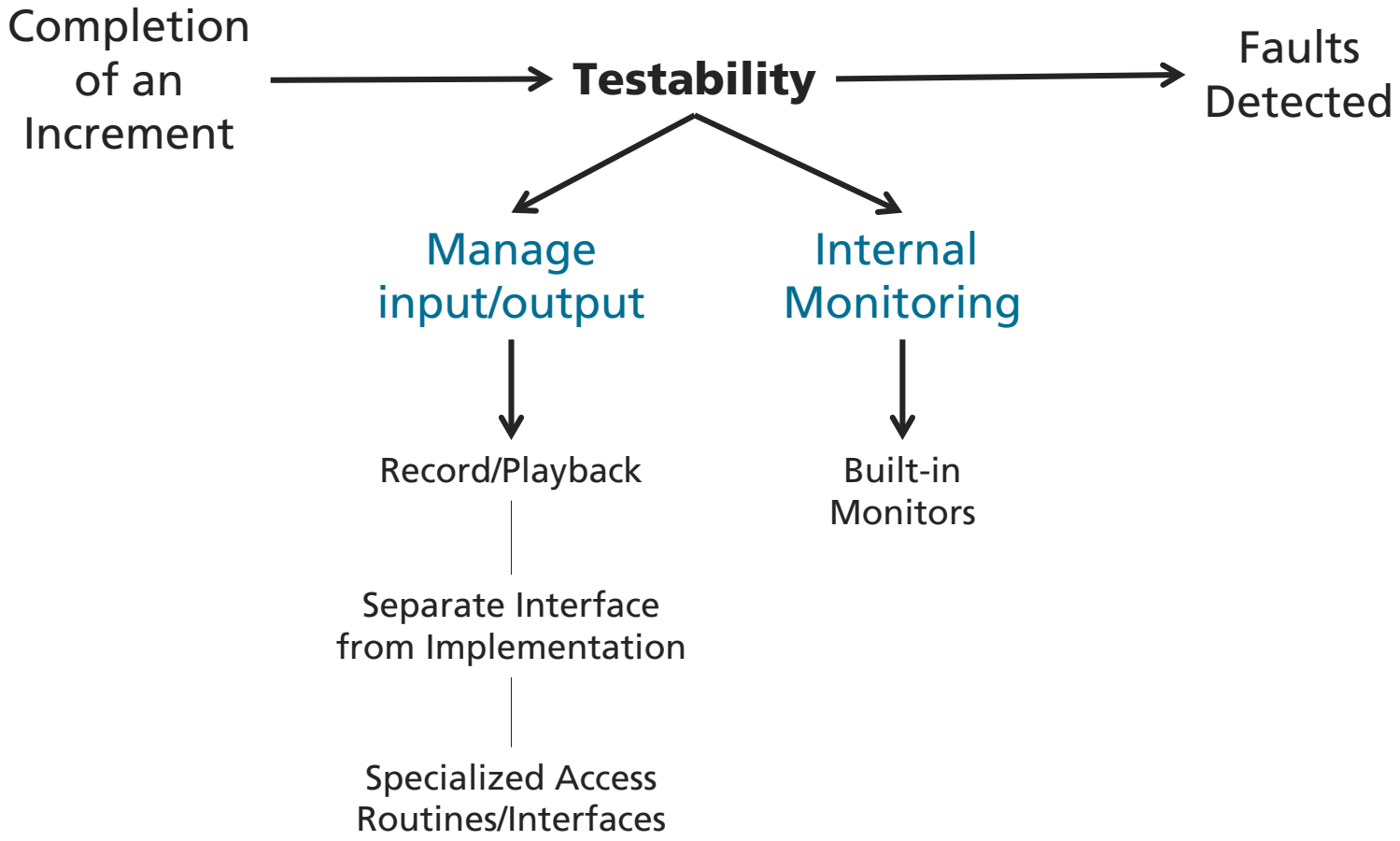
© Fraunhofer IESE

Performance Tactics



© Fraunhofer IESE

Testability Tactics



© Fraunhofer IESE

© Fraunhofer IESE