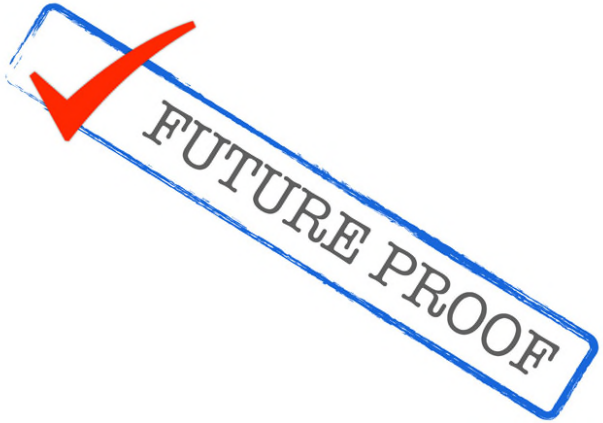


https://static1.squarespace.com

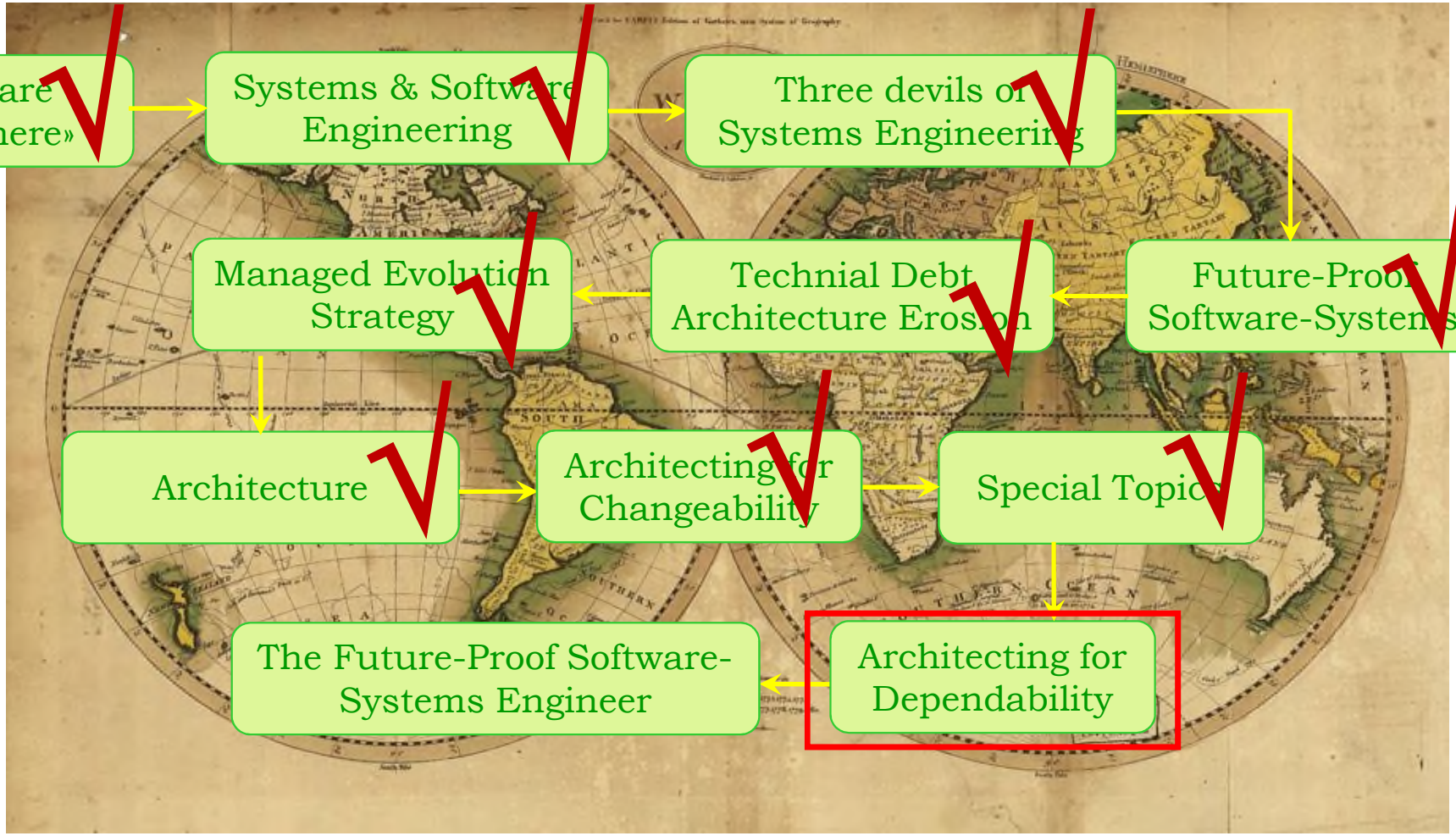
Future-Proof Software-Systems (FPSS)

Part 4A: Architecting for Dependability

Lecture WS 2017/18: Prof. Dr. Frank J. Furrer

Version 0.7

Our journey:



Our objective is:

To build, evolve, and maintain

long-lived, mission-critical IT-systems

with a strong dependability,

an easy changeability,

and a high business value.

Remember

http://www.gettyimages.ch

Content

Part 4A:

1. Introduction
2. Dependability (Repetition)
3. Architecting for Dependability
4. Dependability Architecture Principles
   - ***General*** Dependability Architecture Principles (Resilience)

Part 4B:

- ***Specific*** Dependability Architecture Principles (Examples)

Introduction

Architecting for Changeability:
Part 3 √

Software Changeability

enables success in the competitive business arena

Software Dependability

assures survival in the hostile world

Architecting for Dependability:
Part 4 (this part)

Example:
Weak bank security (1/2)

http://img.deusm.com

February 15, 2015

Carbanak cyber gang:

Hackers in Eastern Europe are bleeding banks dry, stealing as much as ***$1 billion*** from more than 100 financial institutions in a string of attacks that borrow heavily from targeted attacks against sensitive government and industrial targets

https://threatpost.com

Example: Weak bank security (2/2)

⇒ Significant risk for the world banking system

Why is that possible?

Years and years of neglecting dependability!

https://media-cdn.tripadvisor.com

Example:
Industrial espionage (1/2)

http://www.ishn.com

Hackers stole proprietary information from six U.S. and European energy companies, including Exxon Mobil, Royal Dutch Shell, and BP.

Hackers targeted ***computerized topographical maps*** worth "millions of dollars" that locate potential oil reserves.

The **cyberespionage** started in 2009 and went on for years (on-going!)

Example: Industrial espionage (2/2)

⇒ Significant risk for the industrial development

Why is that possible?

https://si.wsj.net

Years and years of neglecting Dependability!

Example:
Infrastructure weakness (1/2)

http://s7d2.scene7.com

In 2007, researchers at the Idaho National Lab conducted the Aurora test, in which a ***virus*** manipulated the computer network systems that controlled ***diesel generators***. The controlled test resulted in an out-of-synchronism condition. Specifically, the out-of-phase synchronism or out-of-phase condition which causes them to fail.

This test is significant because it demonstrated the ability for a ***computer virus to manipulate grid systems*** and cause massive ***physical*** damage.

https://www.scientificamerican.com

Example: Infrastructure weakness (2/2)

⇒ Enormous risk for the power infrastructures of Nations

Why is that possible?

Years and years of neglecting Dependability!

Example: IoT [Internet of Things] (1/2)

September 2016:
Hackers found 47 new vulnerabilities in 23 IoT devices at DEF CON

http://s3-ap-southeast-2.amazonaws.com

The results from this year's IoT hacking contest are in and it's ***not a pretty picture***:

Smart door locks, padlocks, thermostats, refrigerators, wheelchairs and even solar panel arrays were among the internet-of-things devices that fell to hackers during the IoT Village held at the DEF CON security conference in August 2016.

Example: IoT [Internet of Things] (2/2)

⇒ Unbelievable risk for our connected work and life

Why is that possible?

After 20 years in the Security Game: Everything old, is new again

Example:
ICE train real-time problem (1/2)

https://www.ksta.de

June 2013:
The delivery of the new ICE-Trains from SIEMENS to the Deutsche Bahn is *massively delayed*.
Reason: Real-Time performance problems in the train control software. The brake command takes **one second** to travel through the software and start braking. With a full-stop command from 250 km/h the train will stop 70 meters later because of the software latency.

http://www.tagesspiegel.de/wirtschaft/softwareprobleme-ice-bremsen-zu-spaet/7435884.html

Example:
ICE train real-time problem (2/2)

http://deacademic.com

Eisenbahn-Bundesamt

The German train regulator refused to accept the ICE trains

Massive delay and significant monetary losses for both SIEMENS and Deutsche Bahn

Cause:
The dependability criterium «**real-time requirements**» has not been adequately addressed during software development !

Lessons learned:

«**Non-functional aspects** (Dependability properties) *win the game*»

- *Functional aspects* are (in most cases) foremost in the stakeholder's views
- *Non-functional aspects* are (in many cases) added as an afterthought
- *Non-functional aspects* are cross-cutting concerns – they affect the very **core** of architecture and design
- Implementing/fixing *non-functional aspects* in a later phase – especially after deployment – is a tremendously expensive and risky endeavour
- Not meeting *the non-functional aspects* is often the death of a product or service (and possibly: of the company)

Lessons learned:

Non-functional aspects/properties are a central **responsibility** in all phases of the software/system development process

Future-Proof Software-Systems

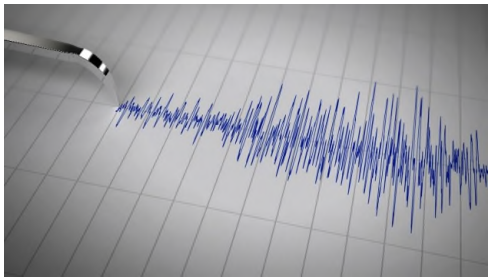
http://mississaugakids.com

The world is a ***dangerous place*** for software and is becoming more dangerous every year

http://www.europeancleaningjournal.com

Where can we build dependability?

On the side of the disruption?



On the side of the impact?

Where can we build dependability?

On the side of the disruption?

Difficult … impossible

On the side of the impact?

Possible … efficient

We need:

Continuous & consistent ***hardening*** of the software-systems

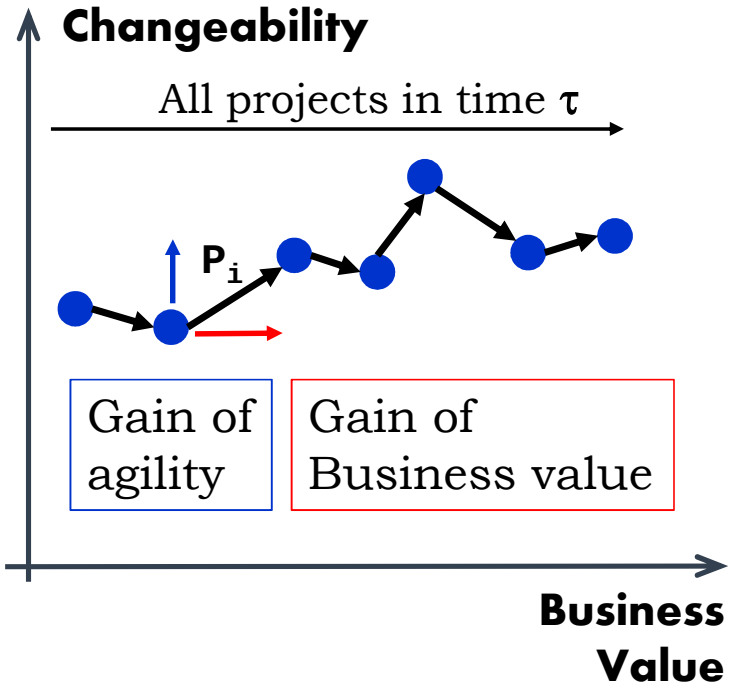
In response to:

- New threats & attacks
- Disruptions in the environment
- Faults and errors (internal & external)
- Increased risks
- Raising hostility

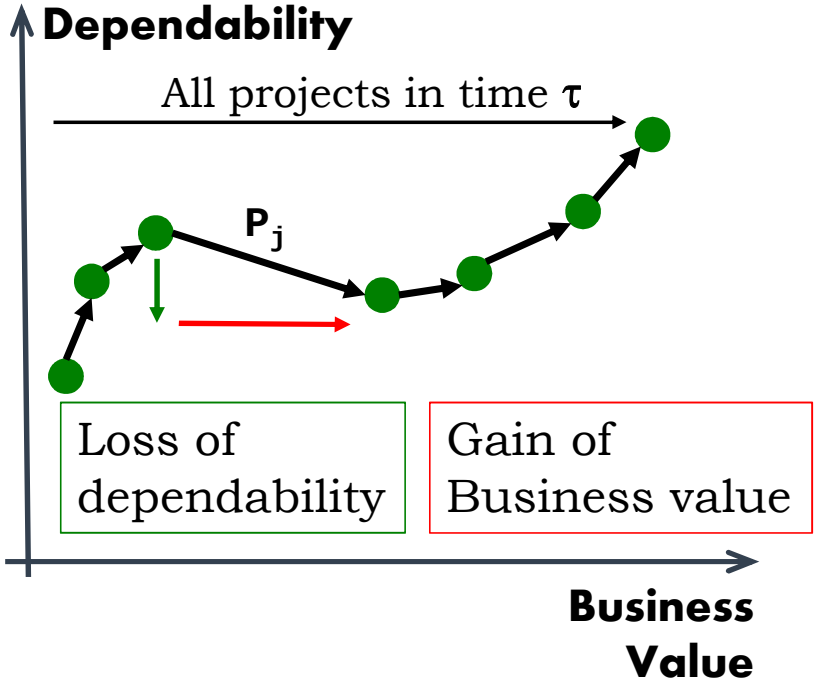
http://www.traiphium.net

Continuous development of **dependability** leads to a sustainable system **(= path to future-proof SW)**

Changeability Evolution Trajectory



Dependability Evolution Trajectory



https://thaguy24.files.wordpress.com

Central questions:

- How do we ***define*** dependability?
- What is ***good*** dependability?
- Can we ***measure*** dependability?
- Which ***principles*** lead to good dependability?

Resilience Example: **Security (Entry protection)**

https://zinatullin.files.wordpress.com

Is that good security?

Why not?

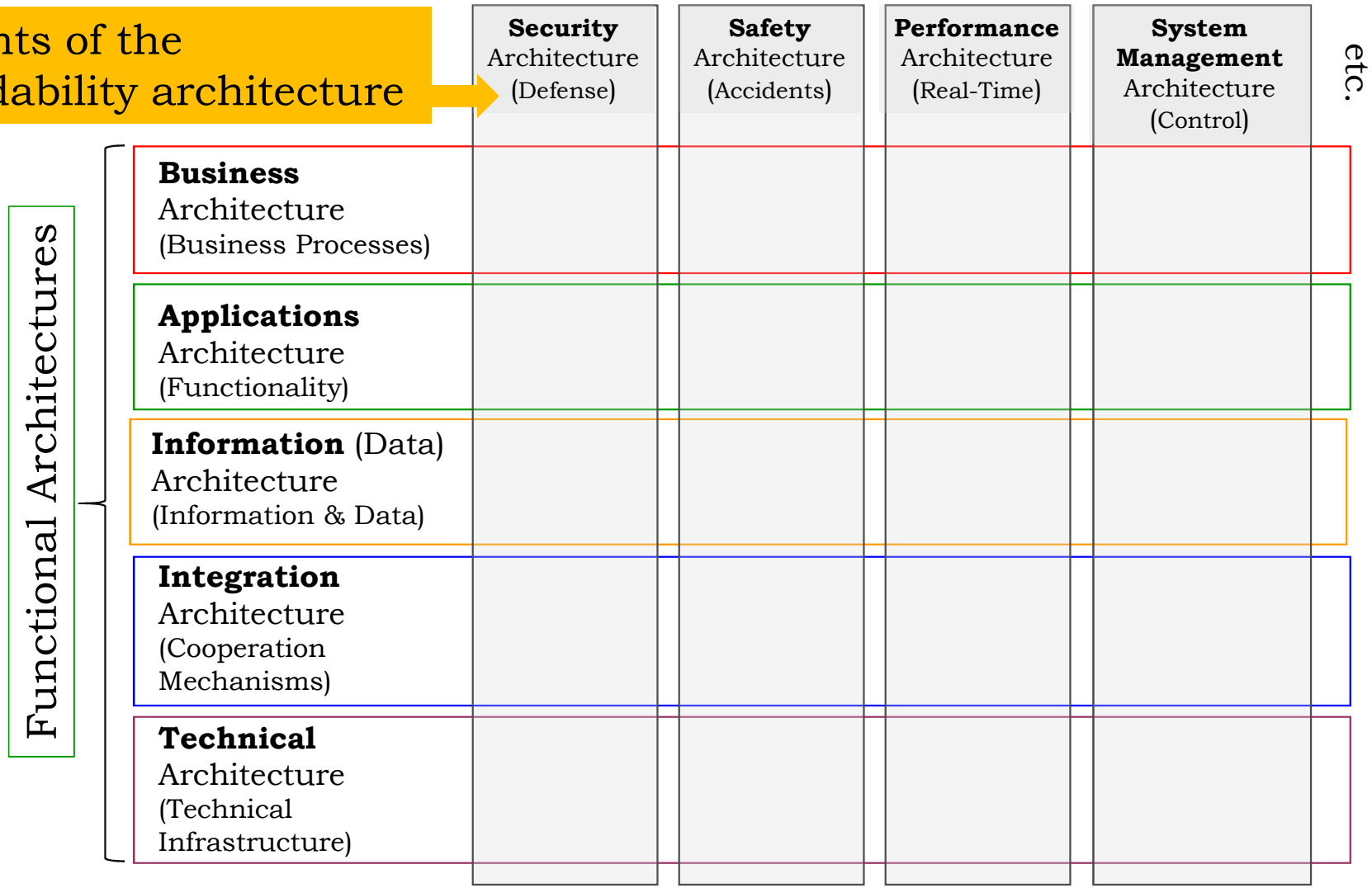
Missing: A Security Architecture!

Foundation of a dependable system

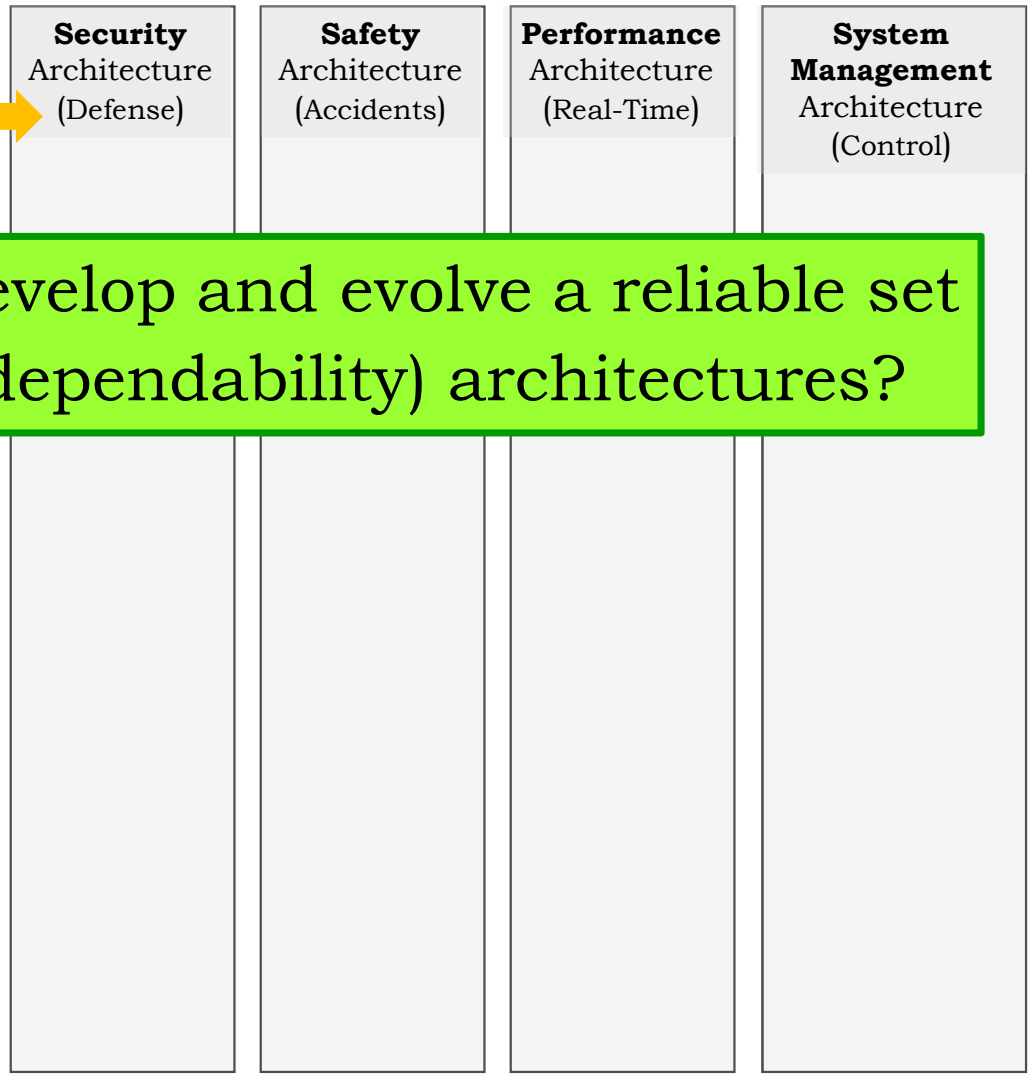
http://easypie.dk

⇒ VERTICAL ARCHITECTUREs (= STRUCTURE)

Elements of the dependability architecture



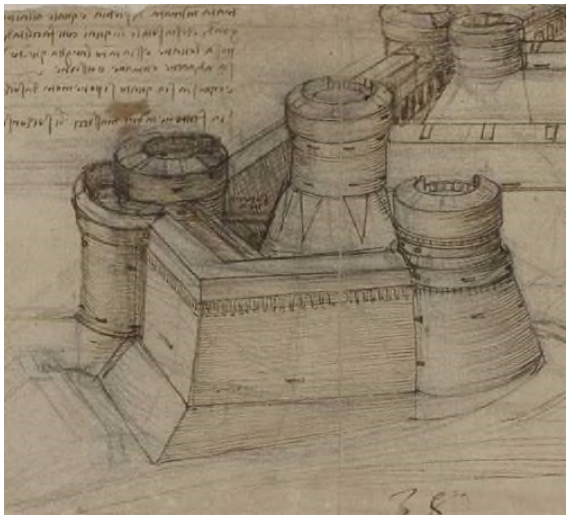
Parts of Dependability Architecture



How do we develop and evolve a reliable set of vertical (dependability) architectures?

By using proven **resilience architecture principles**

Dependability Architecture Principles



http://www.marinabaysands.com

General Resilience Architecture Principles (Overarching Principles)

Safety Principles

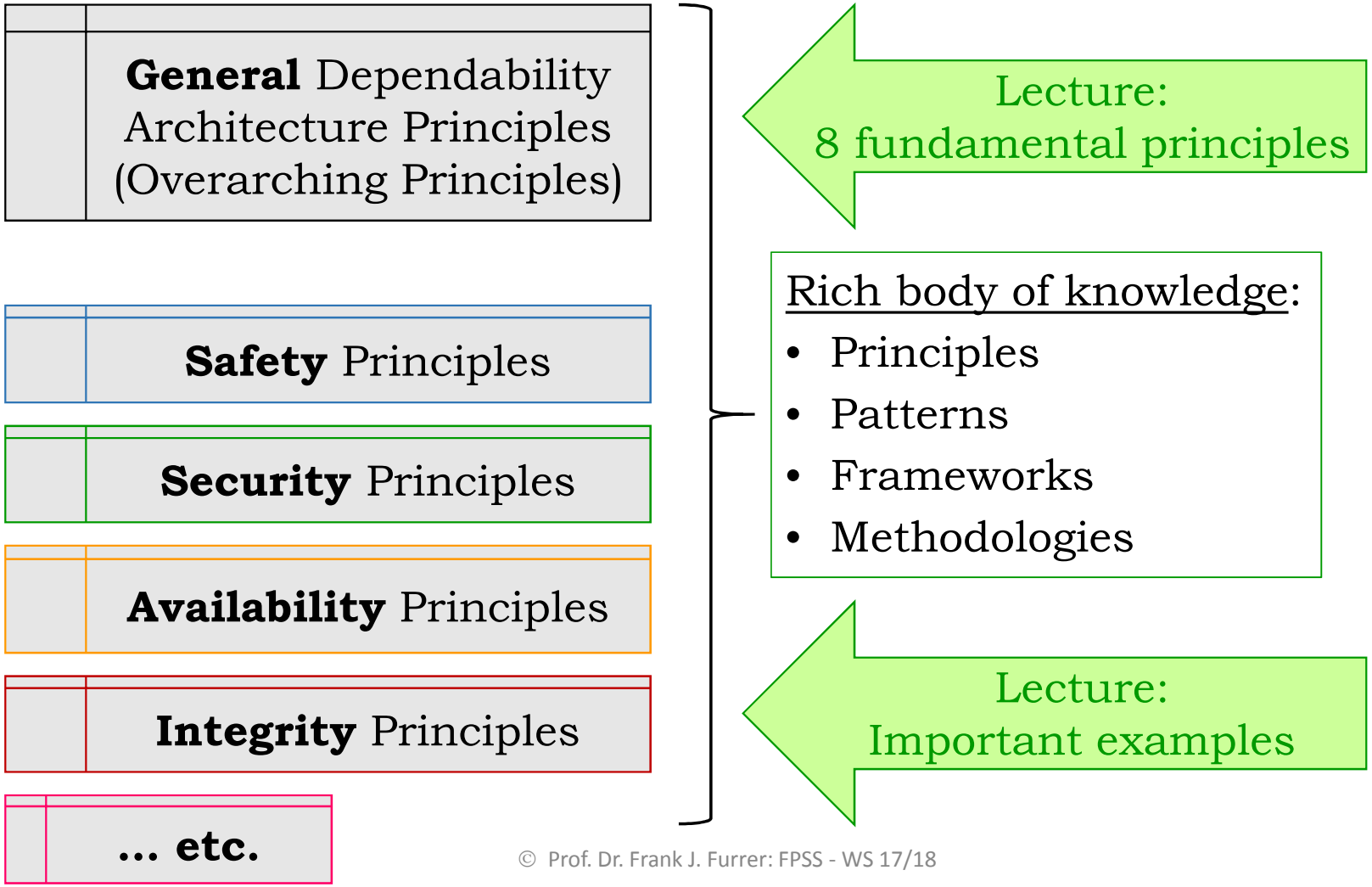
Security Principles

Integrity Principles

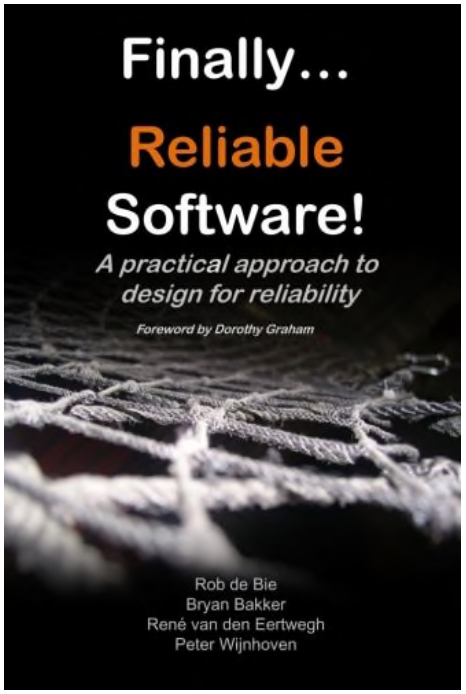
Availability Principles

... etc.

Dependability Architecture Principles

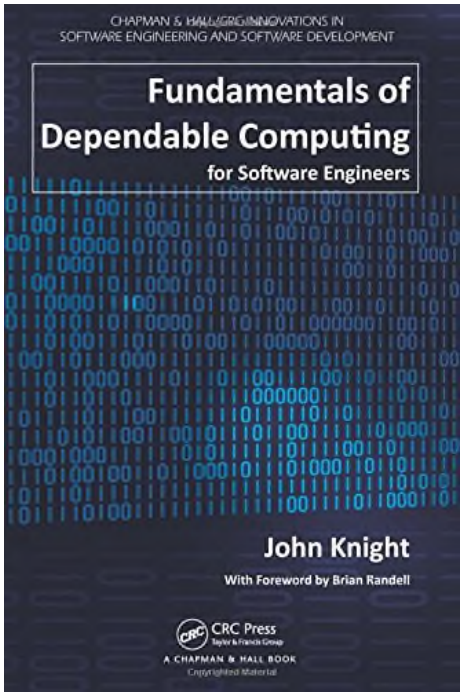


Textbook



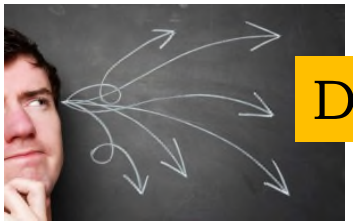
Rob de Bie, Bryan Bakker,Rene van den Eertwegh, Peter Wijnhoven:
Finally... Reliable Software!: *A practical approach to design for reliability*
CreateSpace Independent Publishing Platform, 2015. ISBN 978-1-4992-2666-9

Textbook



John Knight:
Fundamentals of Dependable Computing for Software Engineers
CRC Press (Francis & Taylor), USA, 2012. ISBN 978-1-439862551

Dependability: Repetition



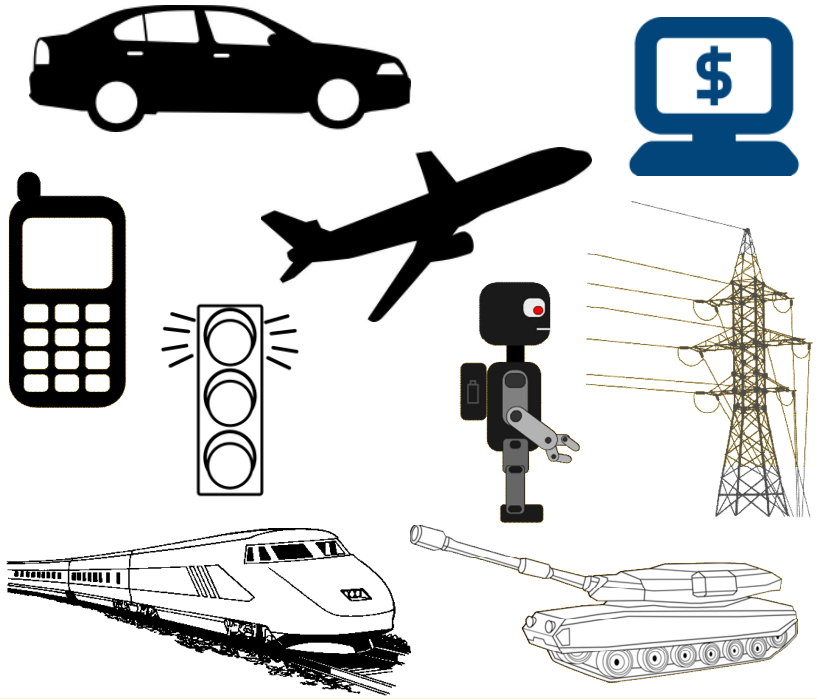
Dependability **expectations** → Dependability **properties**

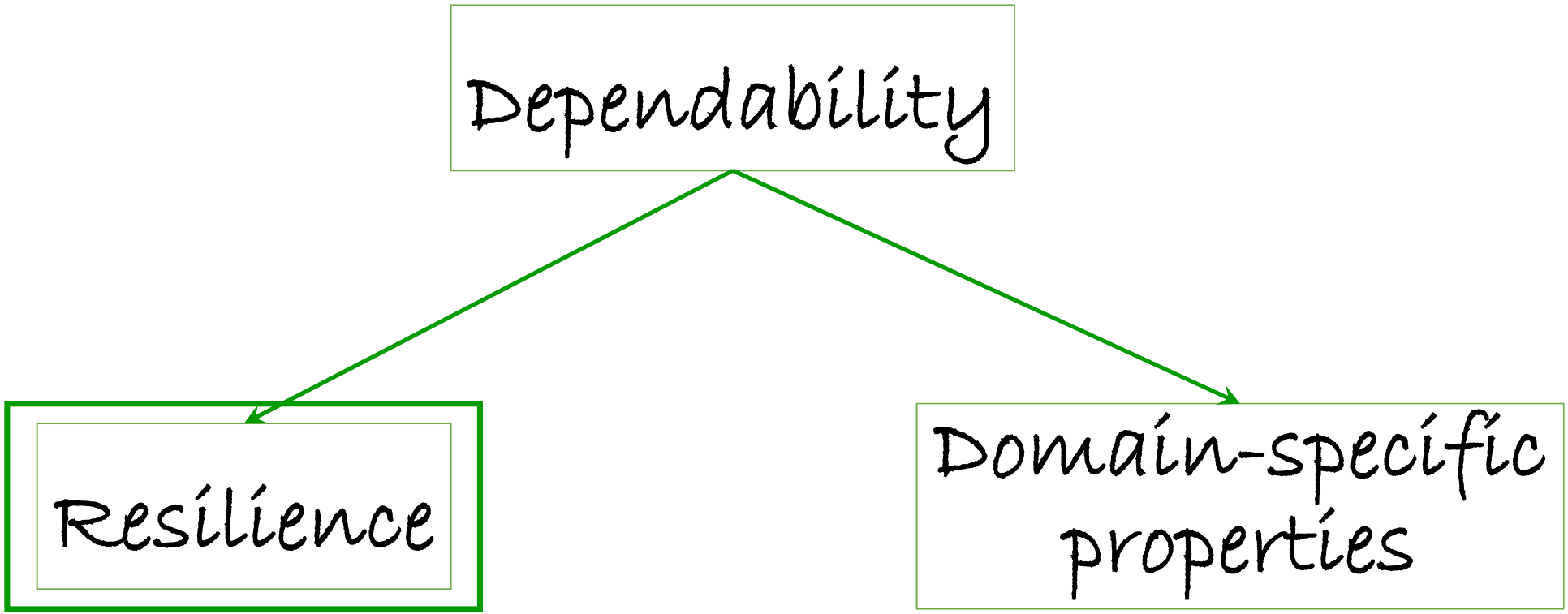
Resilience — General property

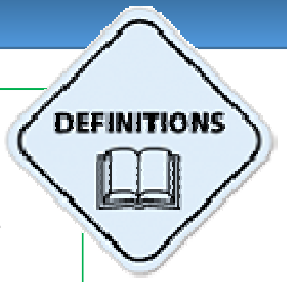
- **Safety**
- **Security**
- **Integrity**
- **Confidentiality**
- **Real-time capability**
- **...**

Specific properties

Application Domain



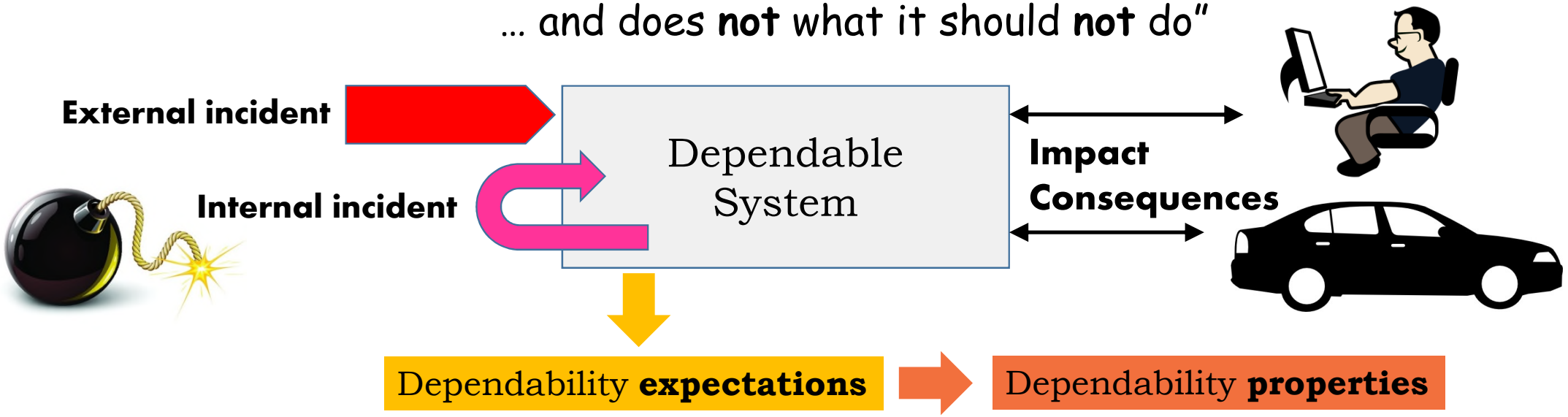

Dependability
Resilience
Domain-specific properties

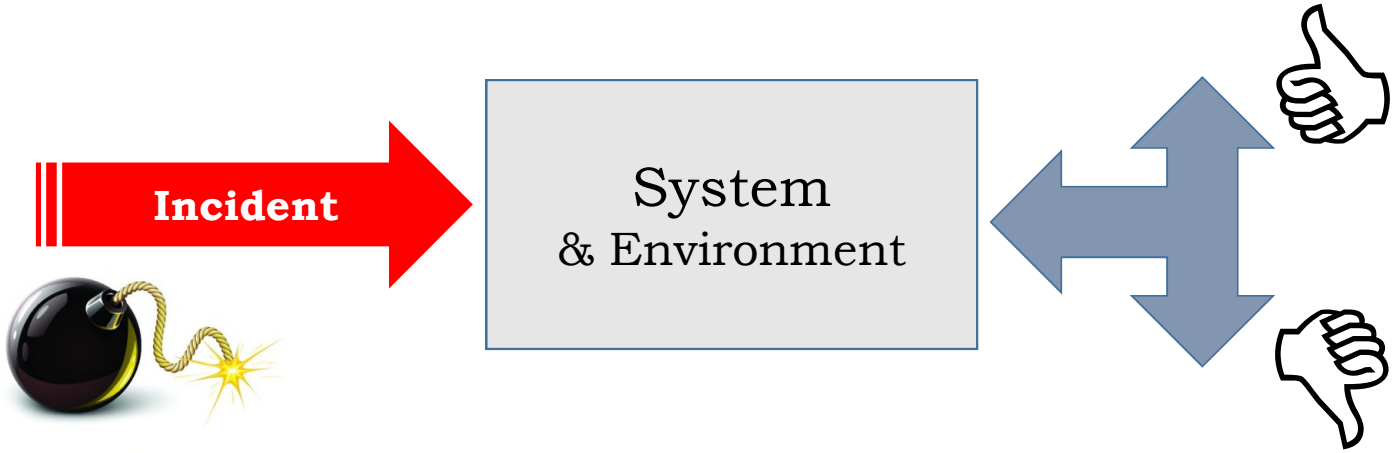


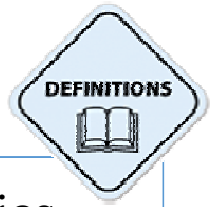
Dependable System

"Dependability" refers to the user's ability to depend on a system in its intended environment, with its intended use, as well as when these assumptions are violated or external events cause disruptions.

"The software does what it should do,
... and does **not** what it should **not** do"







Resilience is the *capability* of a system with specific characteristics before, during and after a *disruption* to *absorb* the disruption, *recover* to an acceptable level of performance, and *sustain* that level for an acceptable period of time

http://www.incose.org/practice/techactivities/wg/rswg/

Dependability Profile

For every domain or application a ***dependability profile*** is defined

Application Domain Quality Property	1 (low)	2	3	4	5 (high)
Functionality				x	
Availability			x		
Security					x
Safety					x
Integrity				x	
Efficiency		x			
Performance		x			
Reliability			x		
Recoverability		x			
Traceability (& Forensics)	x				
Accountability		x			

Quality Property Score Card

Example:
Automotive Domain

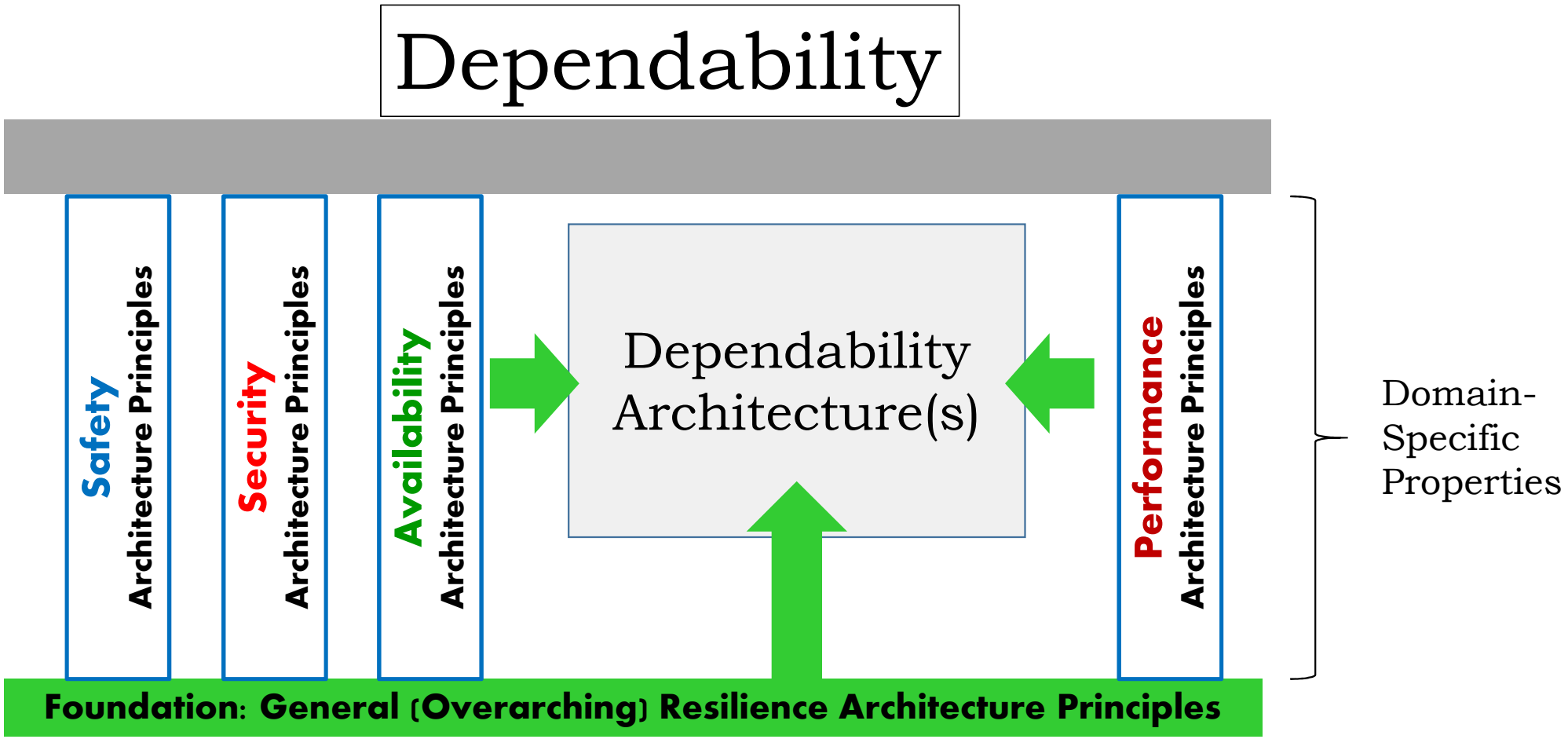
#	System Quality Property	Weight 0: irrelevant 10: highest importance
Primary Characteristics		
1	Business Value	**10**
2	Changeability	**10**
Dependability:		
3	Safety	9
4	Fault-Tolerance	9
5	Compliance to laws & regulations	9
6	Integrity (Sensor Data)	9
7	Availability	8
8	Security	7
9	Diagnosability	6
Secondary Characteristics		
10	Resources (Memory, CPU, …)	8
11	Compliance to industry-standards	7
12	Usability (User Interfaces)	9
etc		

Quality Property Score Card

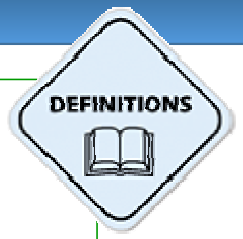
Example:
Automotive Domain

#	System Quality Property	Weight 0: irrelevant 10: highest importance
Primary Characteristics		
1	Business Value	**10**
2	Changeability	**10**
Dependability:		
3	Safety	9
4	Fault-Tolerance	9
5	Compliance to laws & regulations	9
6	Integrity (Sensor Data)	9
7	Availability	8
8	Security	7
9	Diagnosability	6
Secondary Characteristics		
10	Resources (Memory, CPU, …)	8
11	Compliance to industry-standards	7
12	Usability (User Interfaces)	9
etc		

Requirements:


Dependability
Safety Architecture Principles
Security Architecture Principles
Availability Architecture Principles
Dependability Architecture(s)
Performance Architecture Principles
Domain-Specific Properties
Foundation: General (Overarching) Resilience Architecture Principles

Architecting for Dependability



Architecting for Dependability:

Defining and implementing an **IT-structure** providing the optimum defense against incidents, based on a *risk management methodology*

https://www.npmjs.com

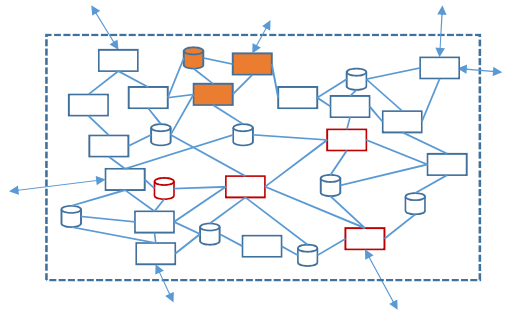
Dependability Engineer:

Responsible for the resilience engineering *process* in a company

- Safety Engineer
- Security Engineeer
- Certification Team
- …

Risk Management Process

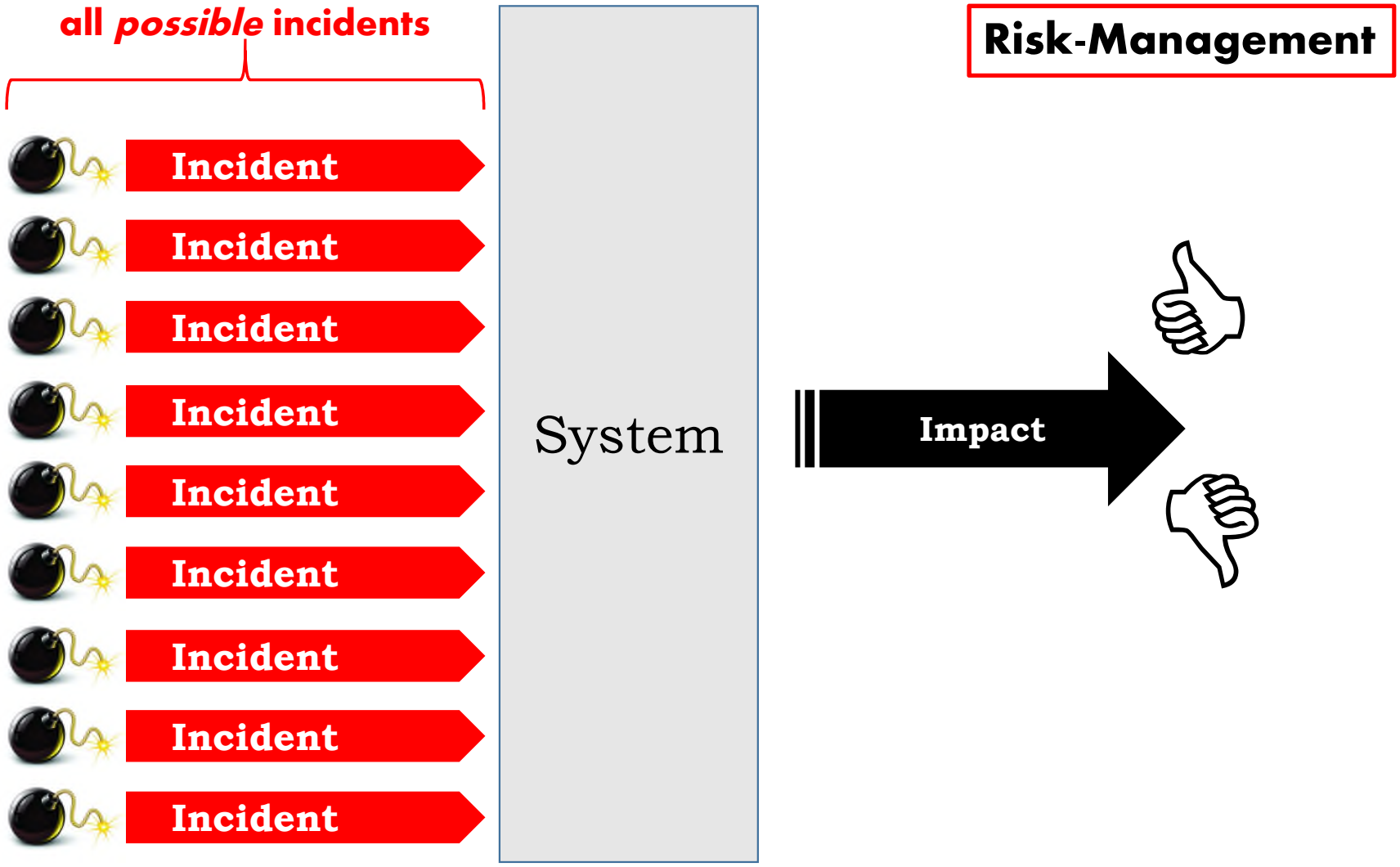
Dependable Architecture

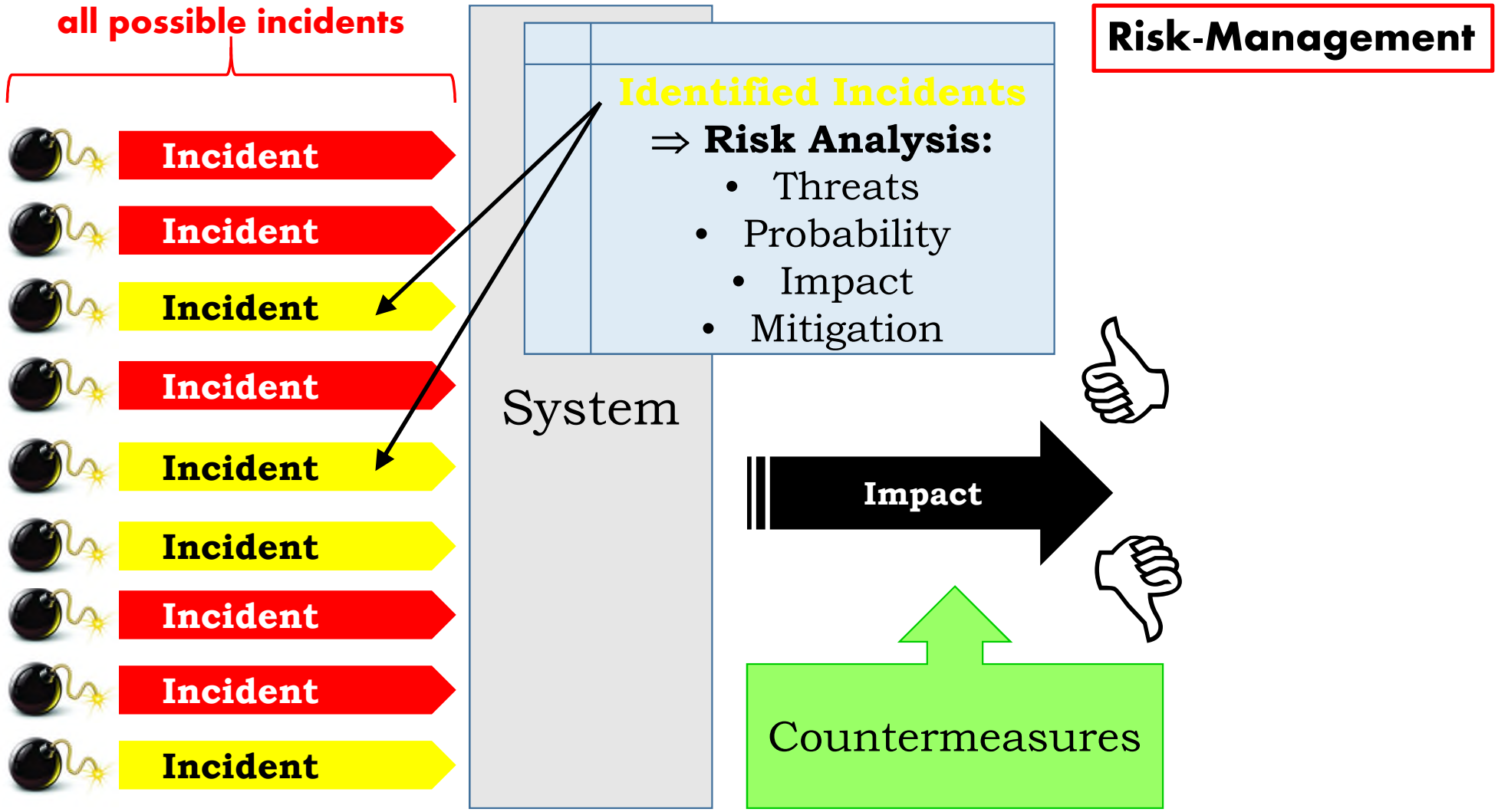


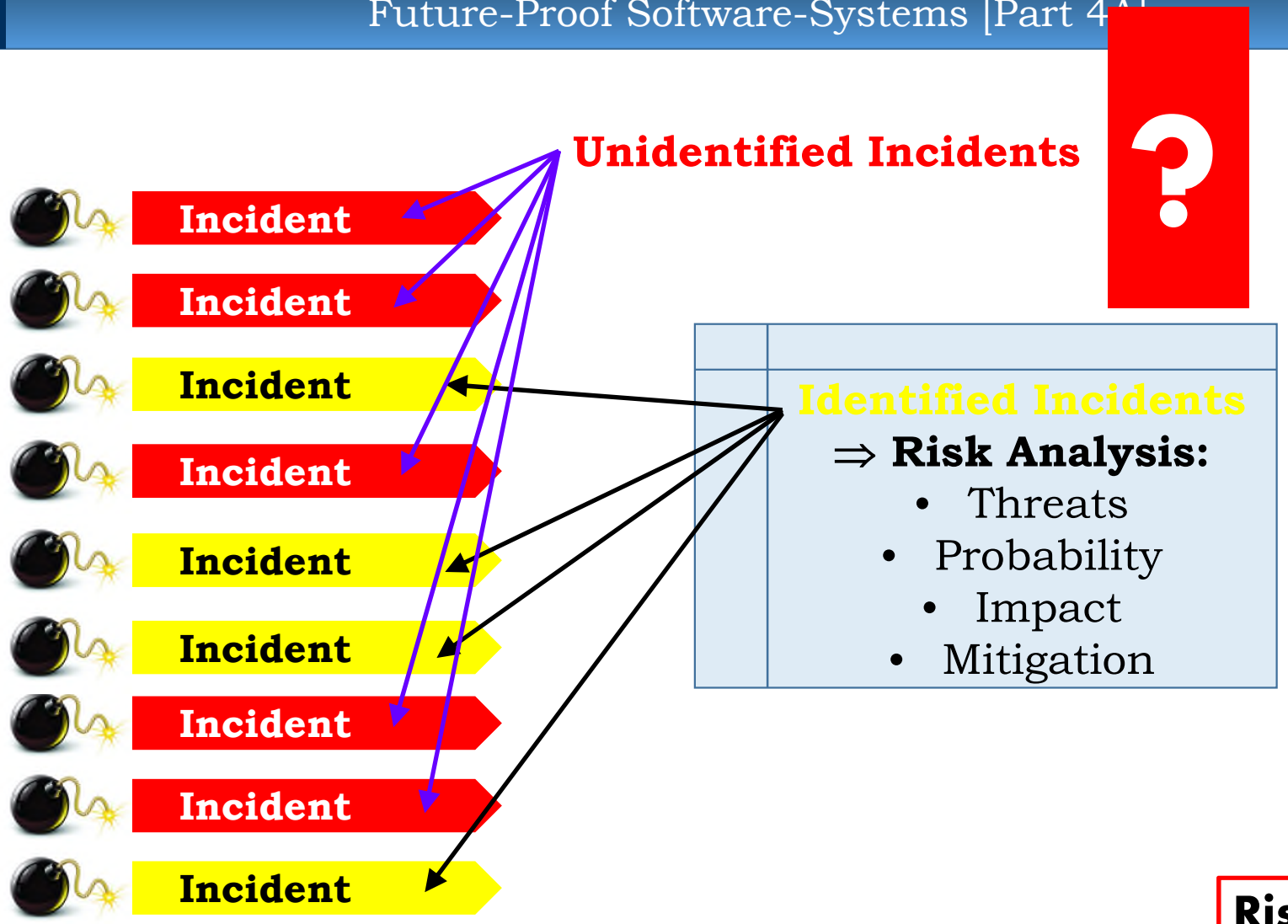
Dependability Building **Process**
(»Architecting for Dependability«)

Resilience Architecture Principles

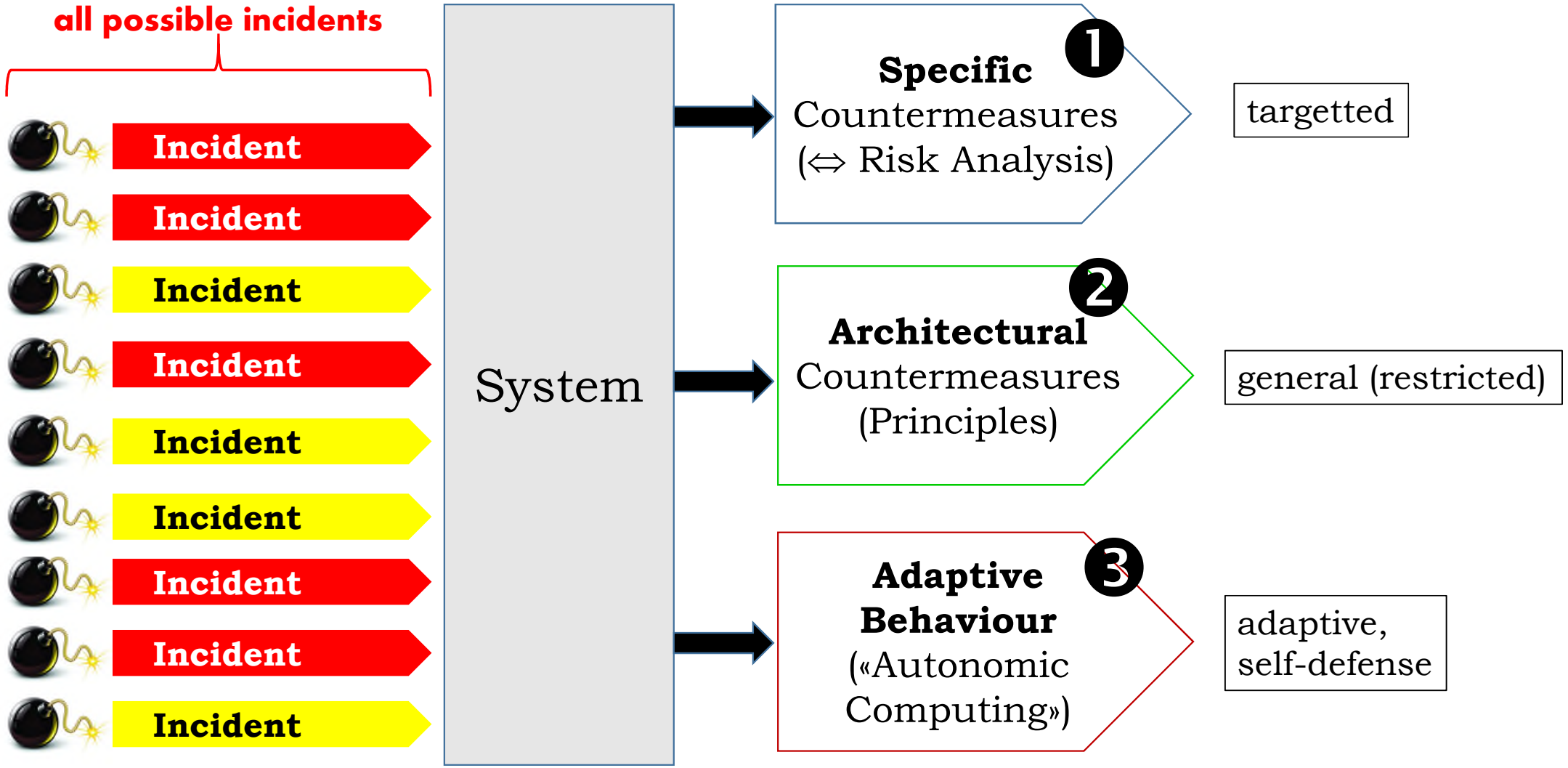
Application-specific Architecture Principles


all *possible* incidents
Incident
Incident
Incident
Incident
Incident
Incident
Incident
Incident
Incident
System
Impact
Risk-Management


all possible incidents
Incident
Incident
Incident
Incident
Incident
Incident
Incident
Incident
Incident
Identified Incidents
⇒ Risk Analysis:
• Threats
• Probability
• Impact
• Mitigation
System
Risk-Management
Impact
Countermeasures

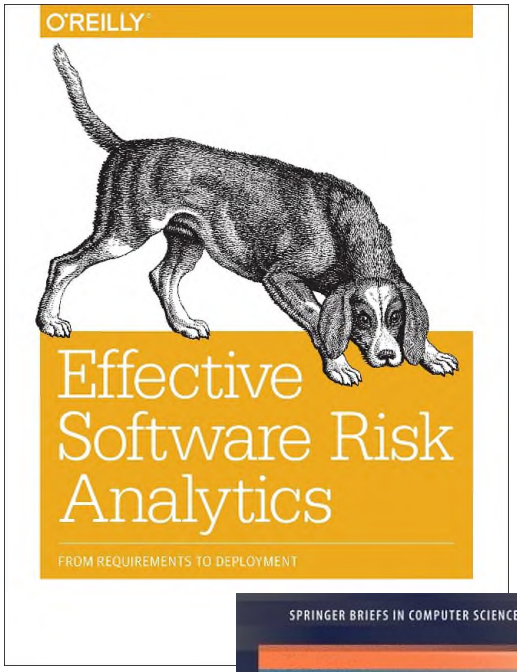


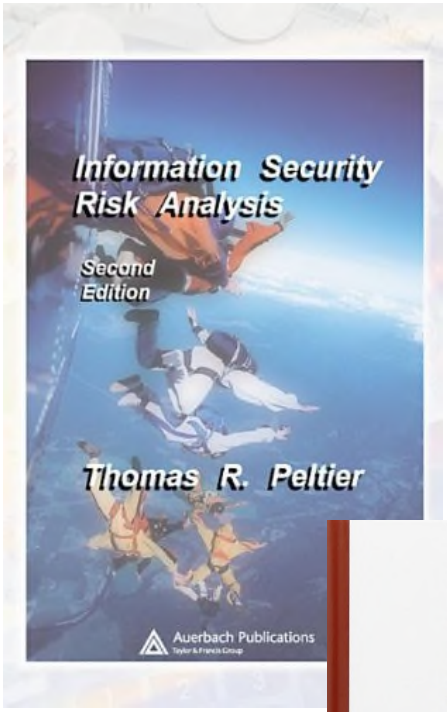
Risk-Management

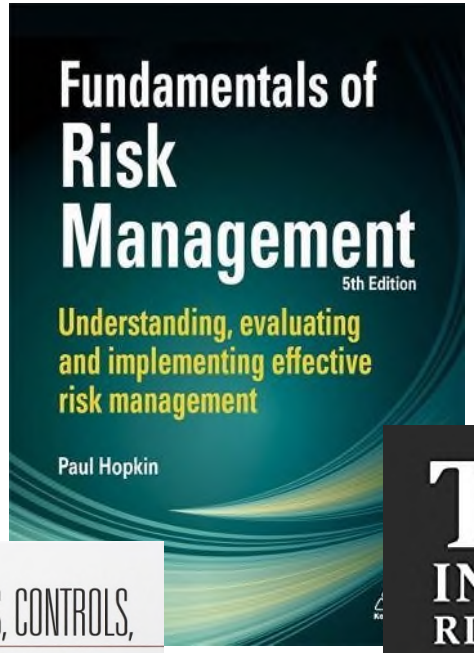

all possible incidents
Incident
Incident
Incident
Incident
Incident
Incident
Incident
Incident
Incident
System
1
Specific
Countermeasures
(⇔ Risk Analysis)
2
Architectural
Countermeasures
(Principles)
3
Adaptive
Behaviour
(«Autonomic
Computing»)
targetted
general (restricted)
adaptive,
self-defense

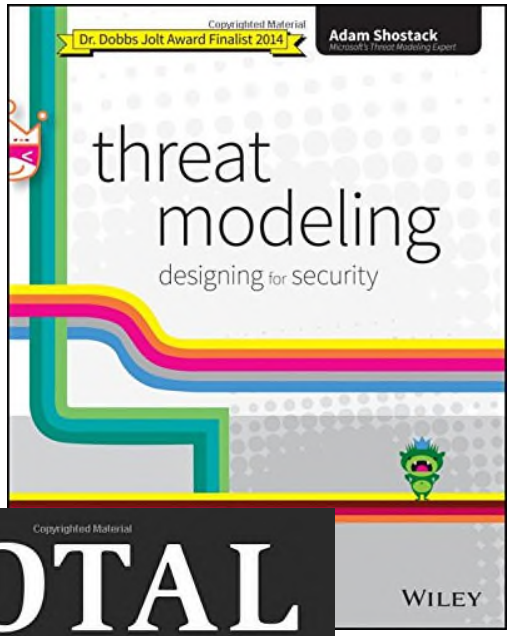
1. **Specific** Countermeasures (⇔ Risk Analysis) → Resilient Code
2. **Architectural** Countermeasures (Principles) → Resilient Architecture
3. **Adaptive Behaviour** («Autonomic Computing») → Resilient System

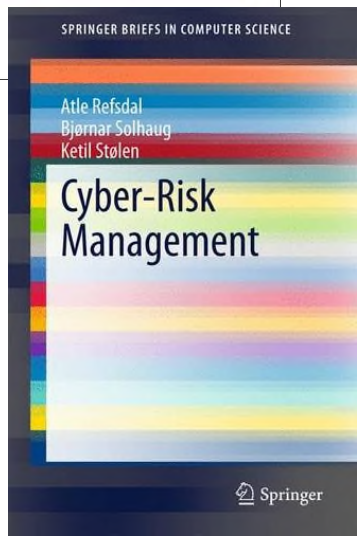
Risk Management

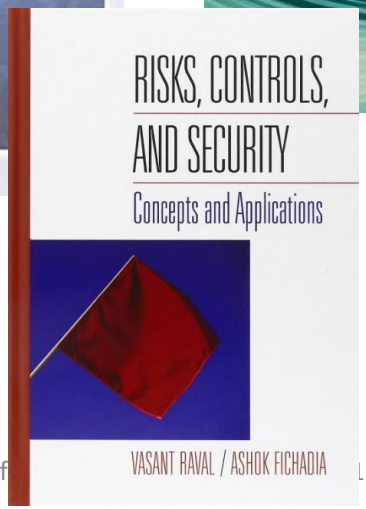


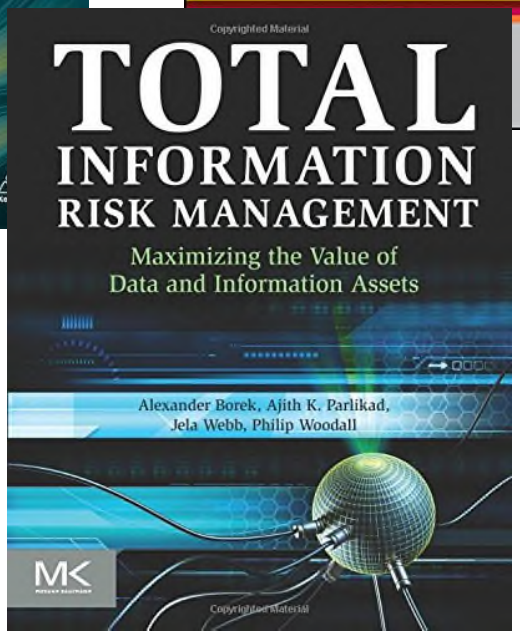












Adaptive approach

Unidentified Incidents

Incident

Incident

But what if the problem

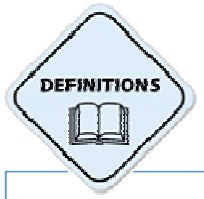
- **is not fully defined**
- **or the environment is uncertain?**

But what if situations

- **are too complex to be predicted**
- **or the environment is changing dynamically?**

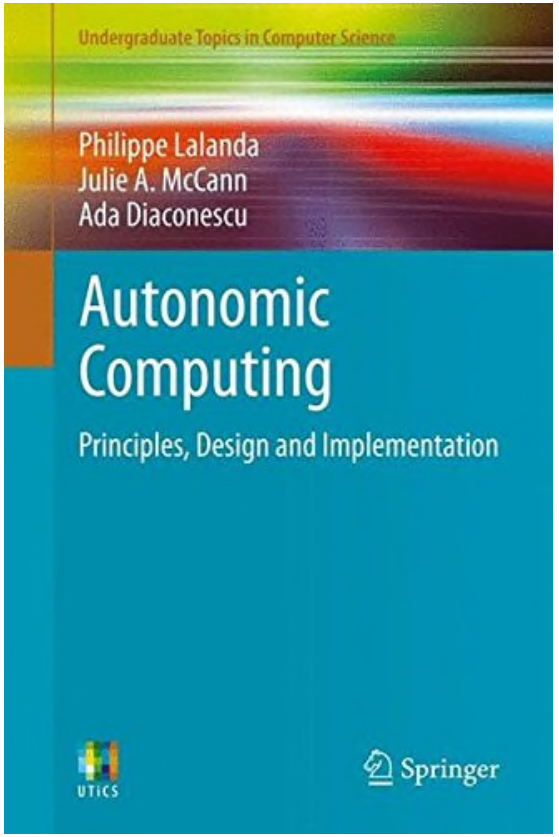
Autonomic Computing

= Specific approach to the engineering of software systems



A type of ***computing model*** in which the system is *self-healing*, *self-configured*, *self-protected* and *self-managed*

⇒ self-* properties



< 2010 → > 2020

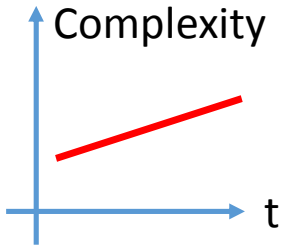
Disruptive Incidents
1. Xxx
2. Yyy
3. Zzz

System

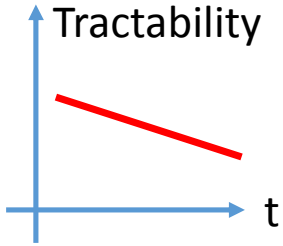
Safety Case

predict

Complexity



Tractability



Self-Protecting System

adapt

How can we achieve *high dependability* in a technical system?

Risk-based approach

Use of a risk-management methodology

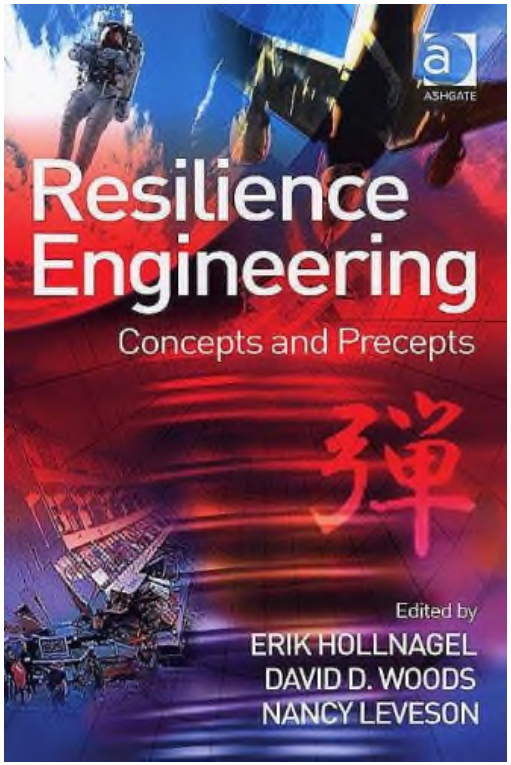
Score-card approach

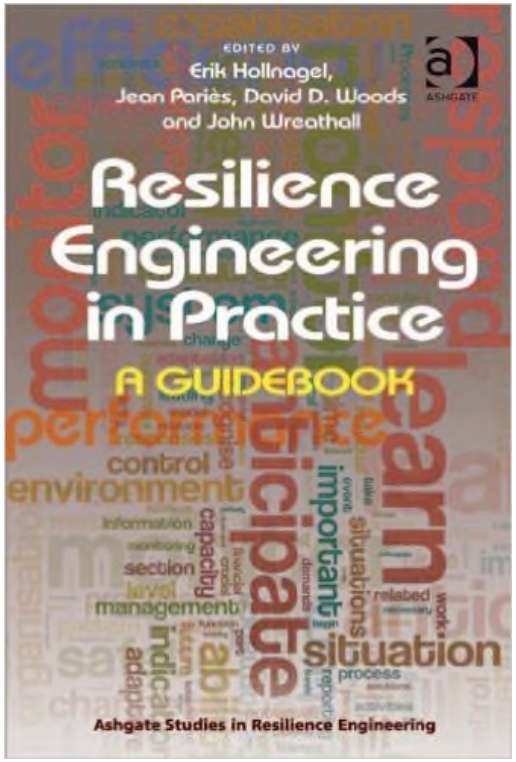
Specify the domain-specific dependability properties

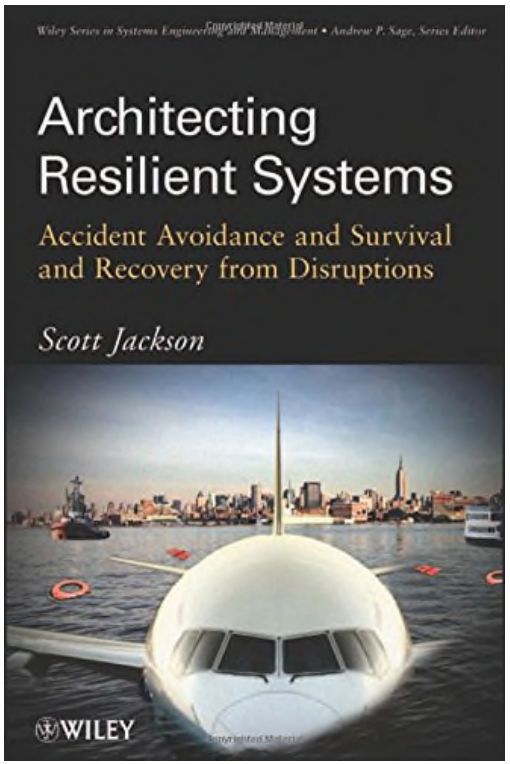
Adaptive approach

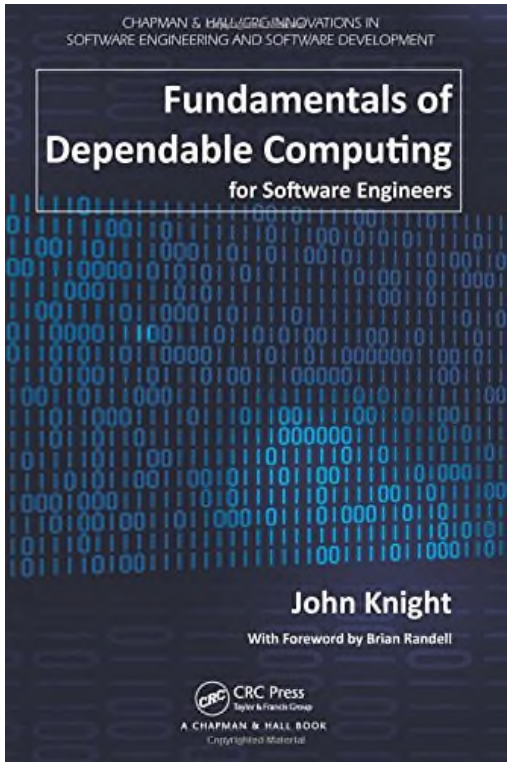
Use self-* capabilities [Autonomic Computing]

⇒ Dependability Engineering

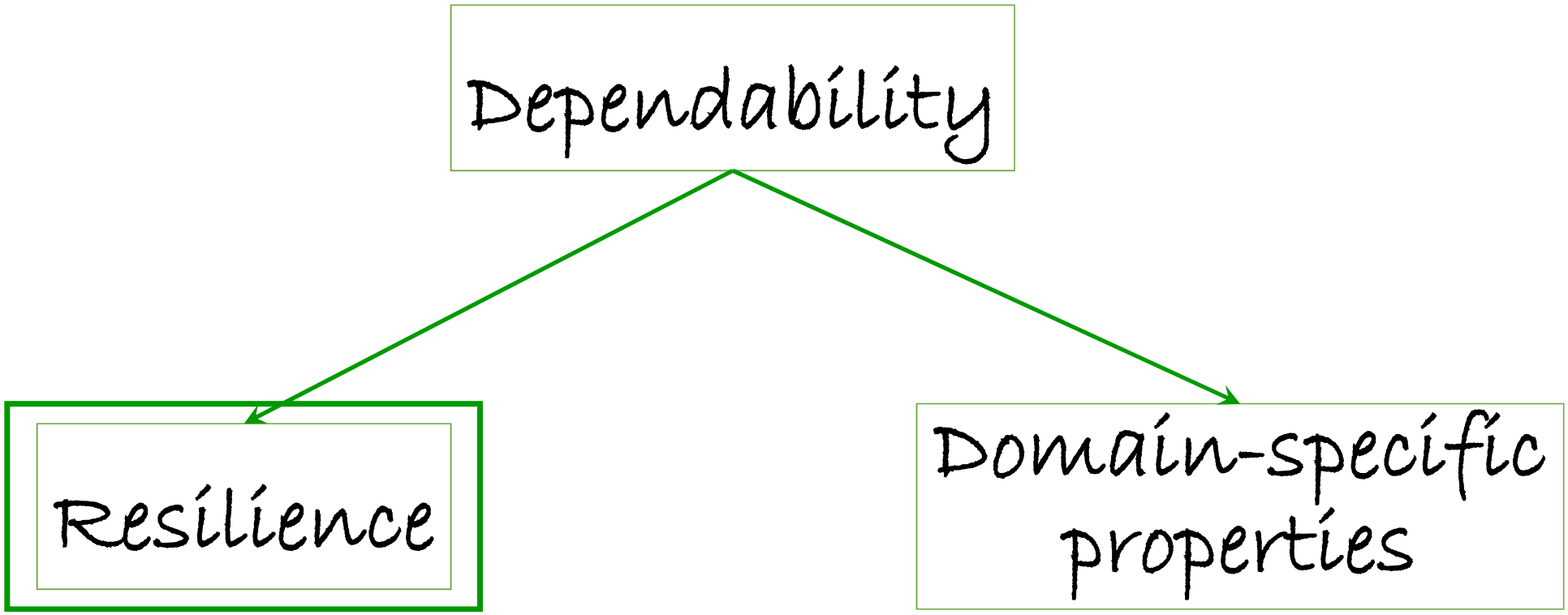








… an interesting research & applications field


Dependability
Resilience
Domain-specific properties

Resilience

Resilience: Reaction

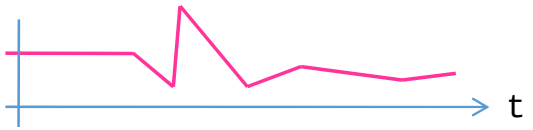
Incident

https://soundcloud.com

1 Crash — **NOT acceptable**

2 Degraded operation — **Acceptable in some cases**

3 Recovery — **Acceptable in some cases**



4 Malfunction — **NOT acceptable**

http://kanto.stripes.com

5 Immunity — **Desirable**

3 phases of resilience:

https://soundcloud.com

http://www.clipartpanda.com

Incident

http://www.joioseareh.com

http://kanto.stripes.com

before | during | after

t

How can we achieve *resilience* in a technical system?

Resilience (from the definition):

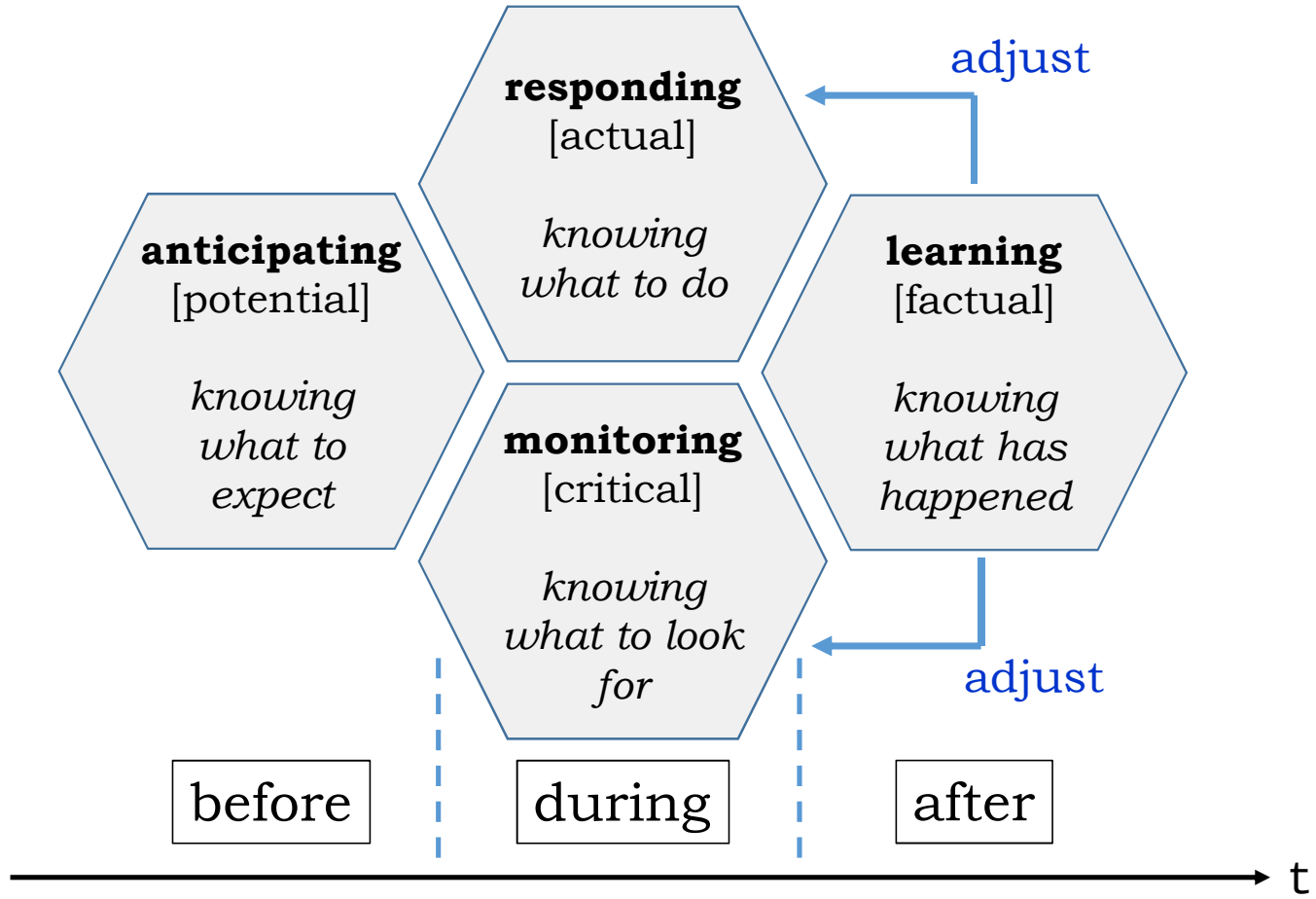
- *Before* – Allows anticipation and corrective action to be considered
- *During* – How the system survives the impact of the disruption
- *After* – How the system recovers from the disruption

Mode(s) of operation
(Forensic) Analysis
Diagnosis
Repair procedures
Improvements
…

Operating procedures
Monitoring
Logging/Audit trail
Recovery mechanisms
…

Resiliency objectives (Profile)
Risk & hazard analysis
Resilient architecture
Full planning
Careful Engineering
Safe development (Process)
Recovery procedures
Scenario testing
…

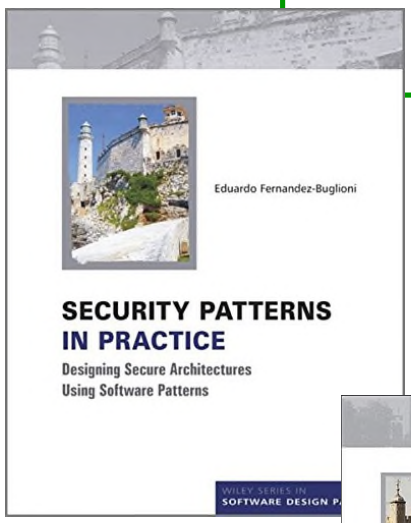
The four cornerstones of resilience

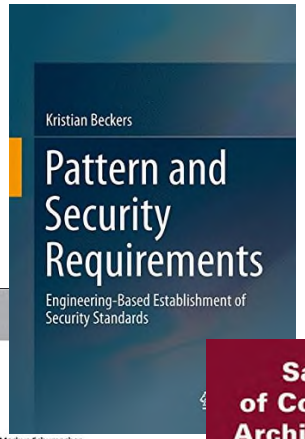


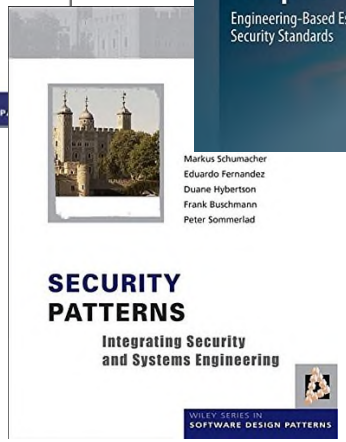
Hollnagel, 2011, ISBN 978-1-4724-2074-9

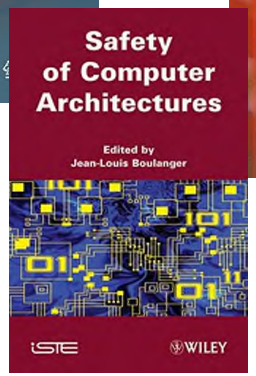
Resilience Patterns

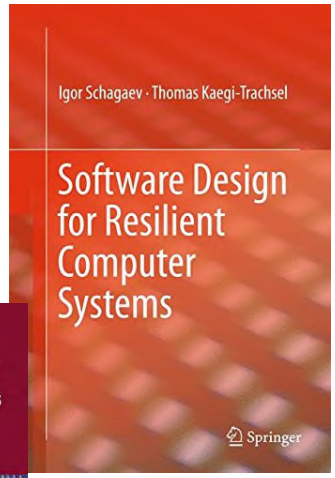
Patterns are the finest-granular form of knowledge
to convey valuable, proven information
about ***resilient architecture/designs***

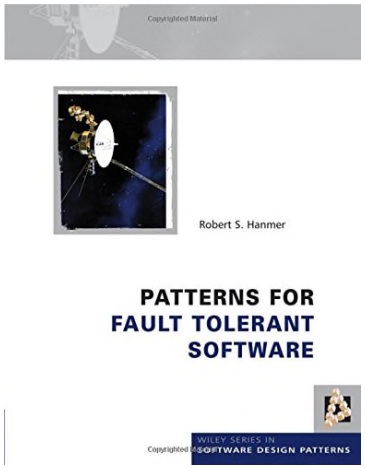


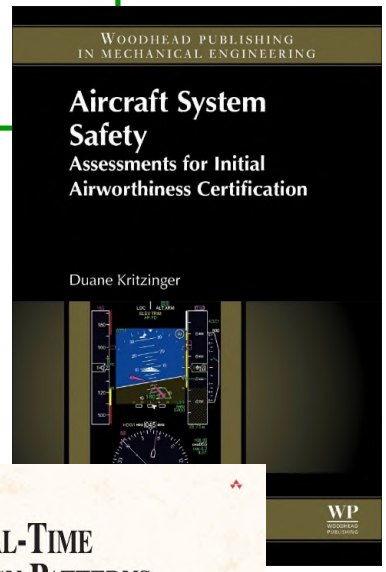


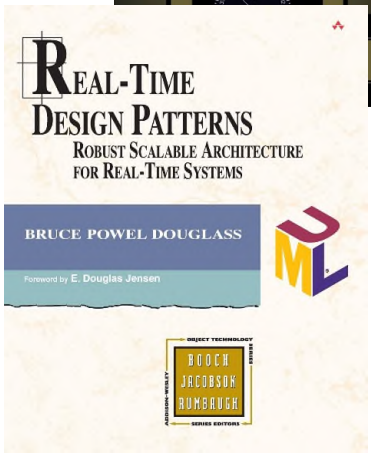












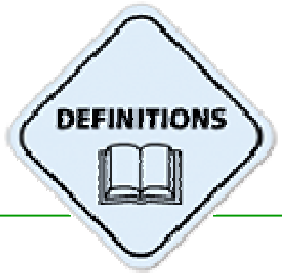
General (Overarching) Architecture Principles for Resilience

- R1: Policies
- R2: Vertical Architectures
- R3: Fault Containment Regions
- R4: Single Points of Failure
- R5: Multiple Lines of Defense
- R6: Fail-Safe States
- R7: Graceful Degradation
- R8: Dependable Foundation (Infrastructure)
- R9: Monitoring

General Resilience

Architecture Principles

http://www.differencebetween.info

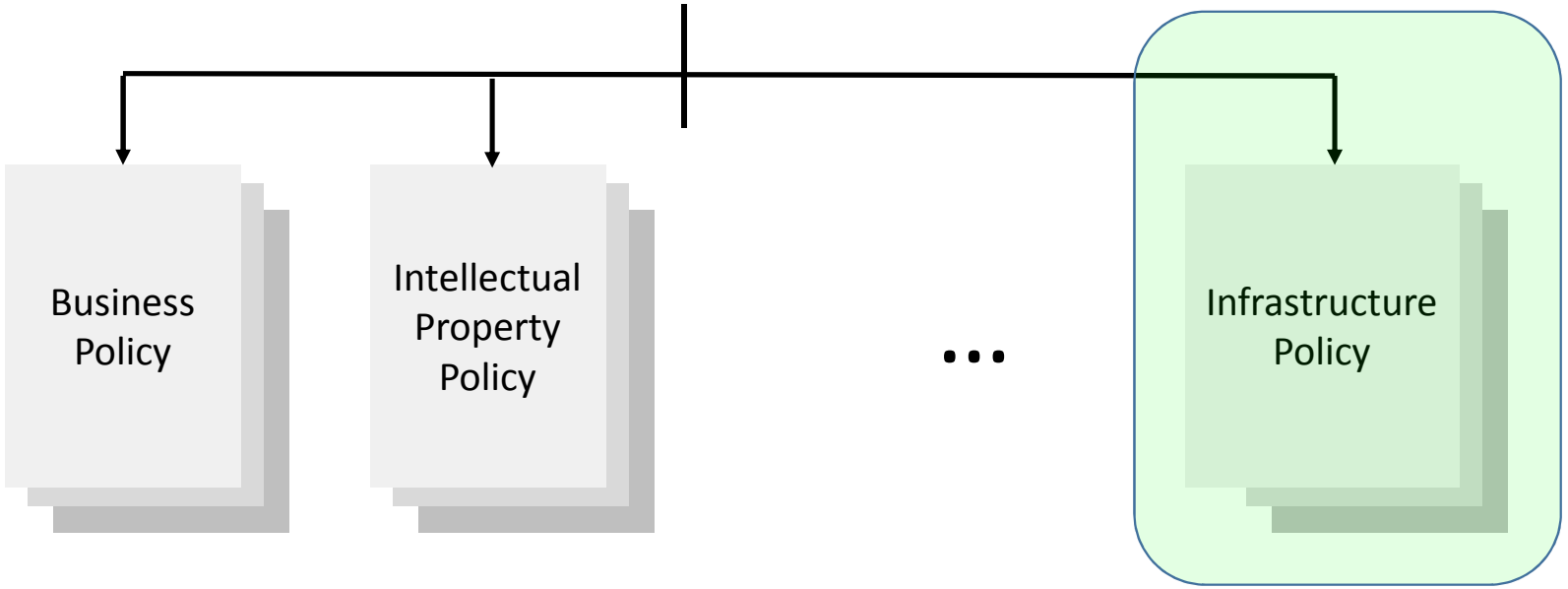


Policy:

The set of *basic principles* and *associated guidelines*, formulated and enforced by the governing body of an organization, to direct and limit its actions in pursuit of ***long-term goals***

http://www.businessdictionary.com/definition/policy.html

Defines the targets, direction and restraints of all activities of the organization

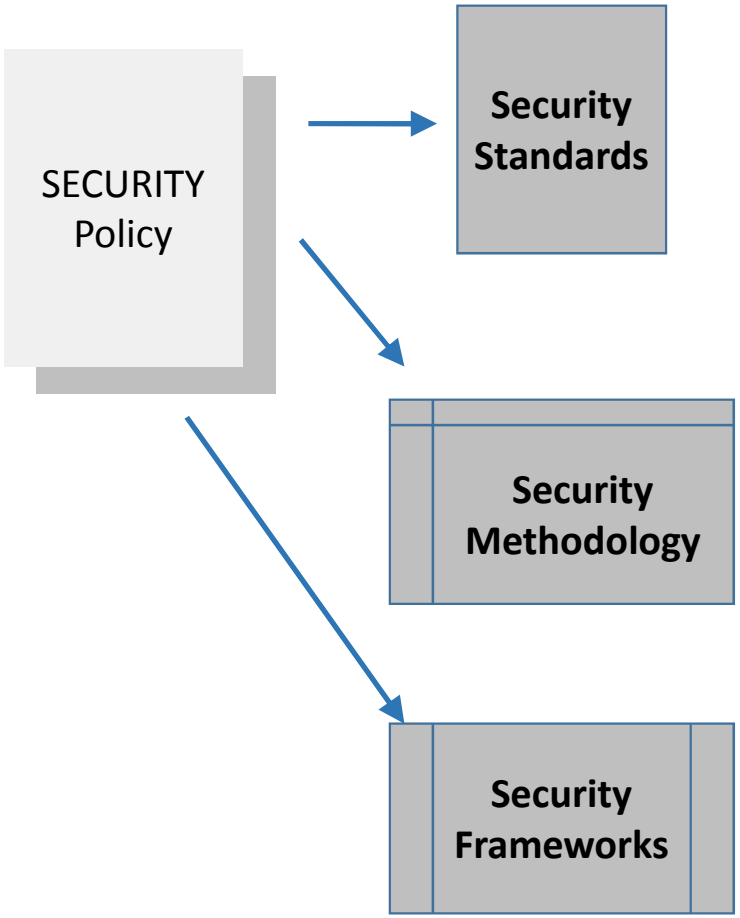


http://www.lighthousecounselingomaha.com

Good policies guide the course of a company in all relevant areas towards sustainable success

Good policies are a great help for the people implementing company objectives, especially ***infrastructure projects***

Policy Implementation:



TOGAF 9
Security Architecture
Summarised – 2010

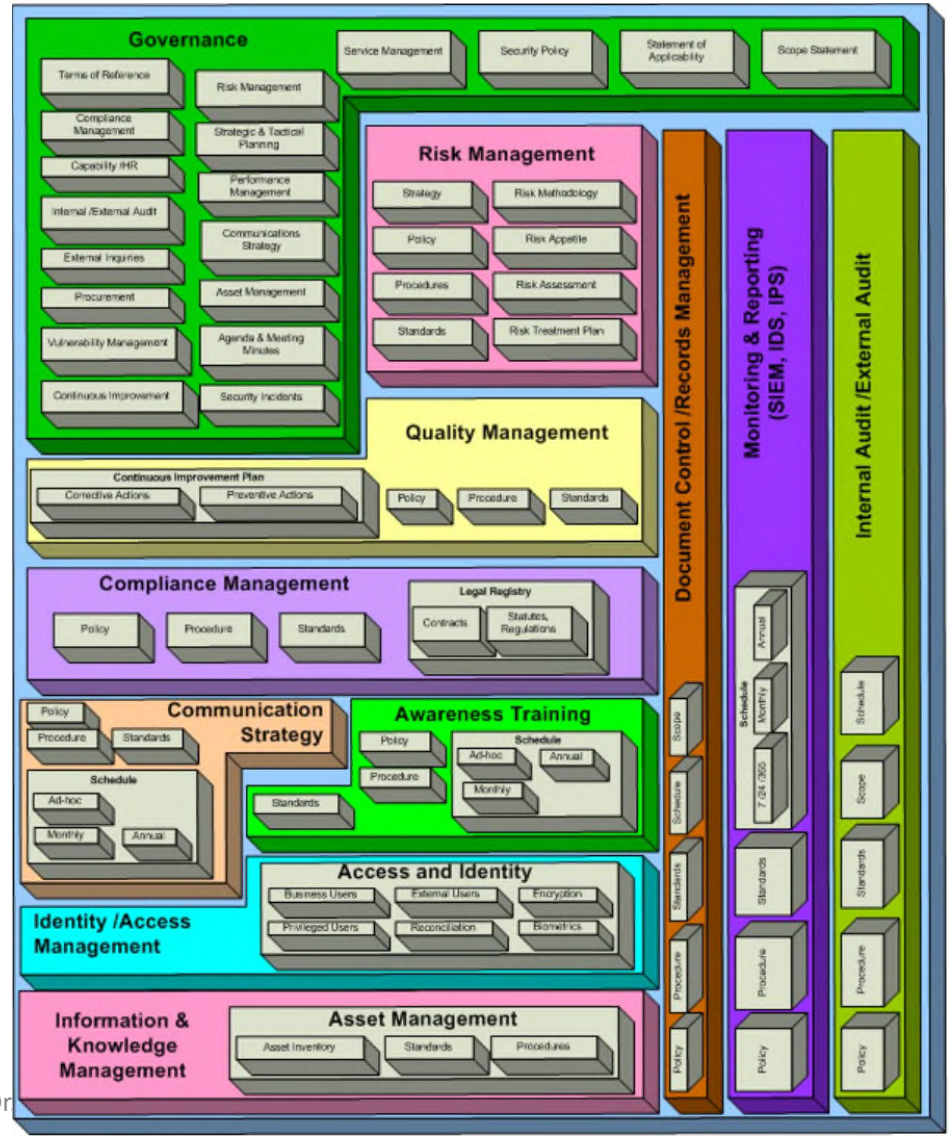
Example:

ISO27001

Information Security

Framework

http://www.iso.org/iso/iso27001



Example:

CC – Common Criteria

Common Criteria for Information Technology Security Evaluation (CC), and the companion Common ***Methodology*** *for Information Technology Security Evaluation (CEM)*

https://www.commoncriteriaportal.org/files/ccfiles/CEMV3.1R4.pdf

R1

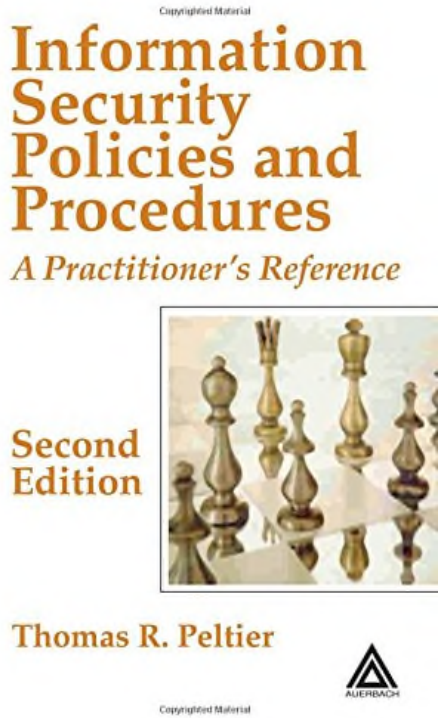
Resilience Architecture Principle R1:

Policies

1. Develop and enforce good, comprehensive, consistent policies for the infrastructure of the company or organization
2. Support the policies by carefully selected standards, methodologies and frameworks
3. Keep all policies and supporting material up-to-date and adapted to the changing environment
4. Consequently apply the policies to the evolution of the infrastructure (enforcement)

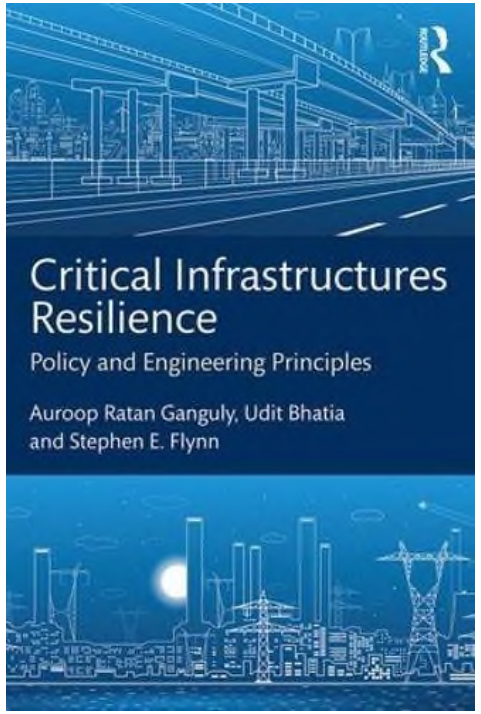
Justification: Technical infrastructure has become so complex and important that it needs to be governed by a consistent set of policies – otherwise the resulting divergence is a massive risk for the company

Textbook



Thomas R. Peltier:
Information Security Policies and Procedures: *A Practitioner's Reference*
Taylor & Francis Ltd, USA, 2nd edition, 2004.
ISBN 978-0-8493-1958-7

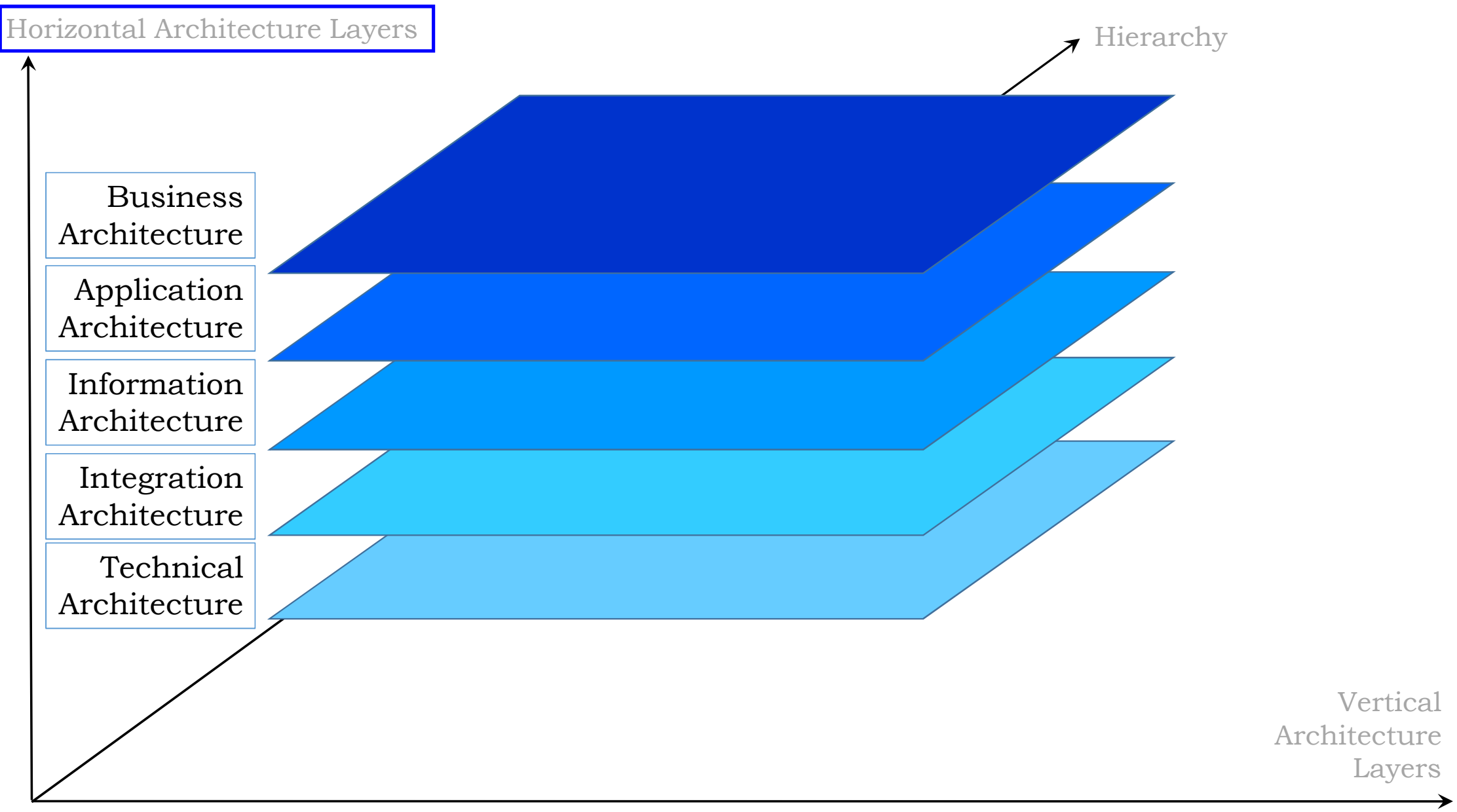
Textbook

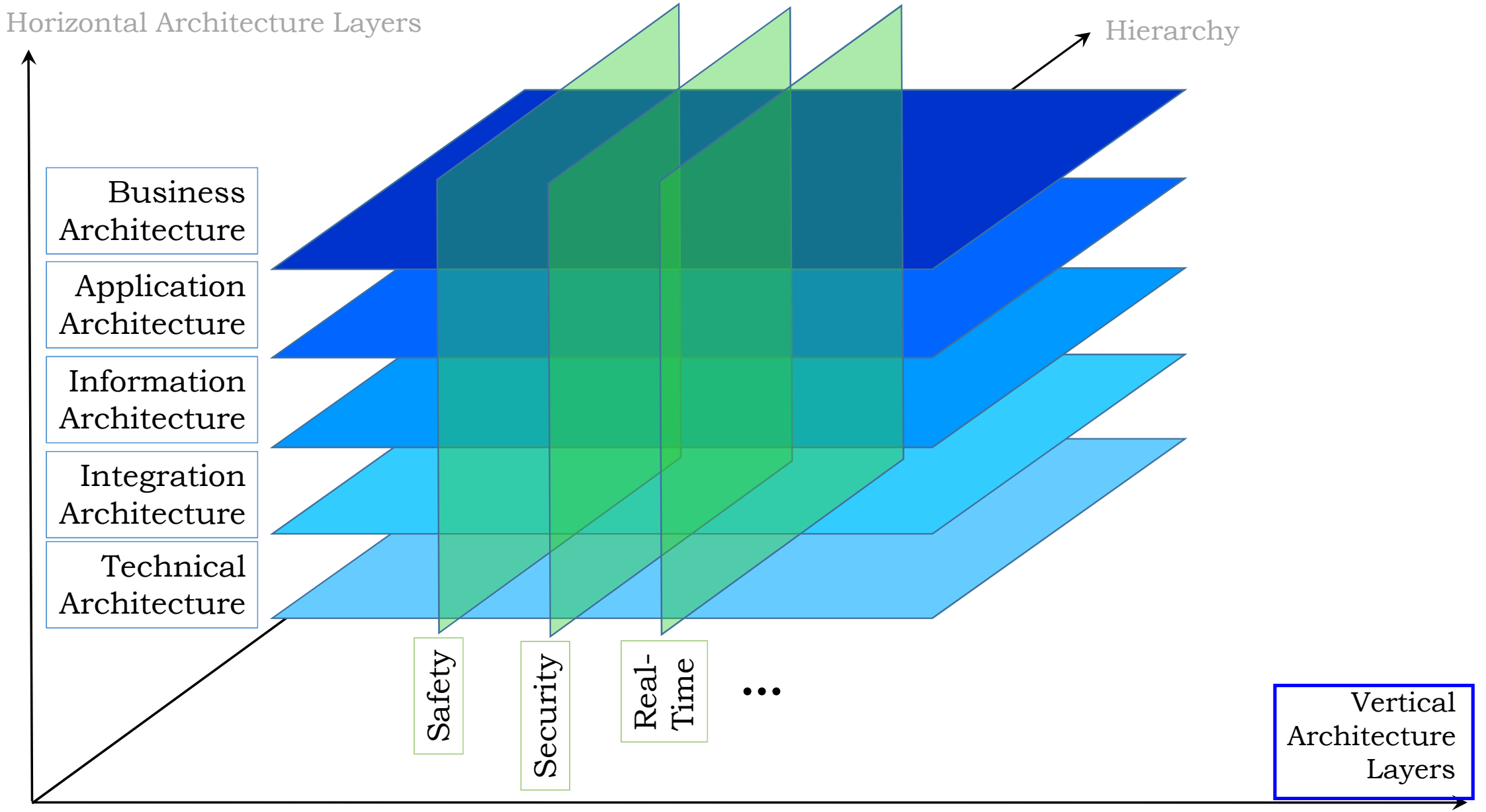


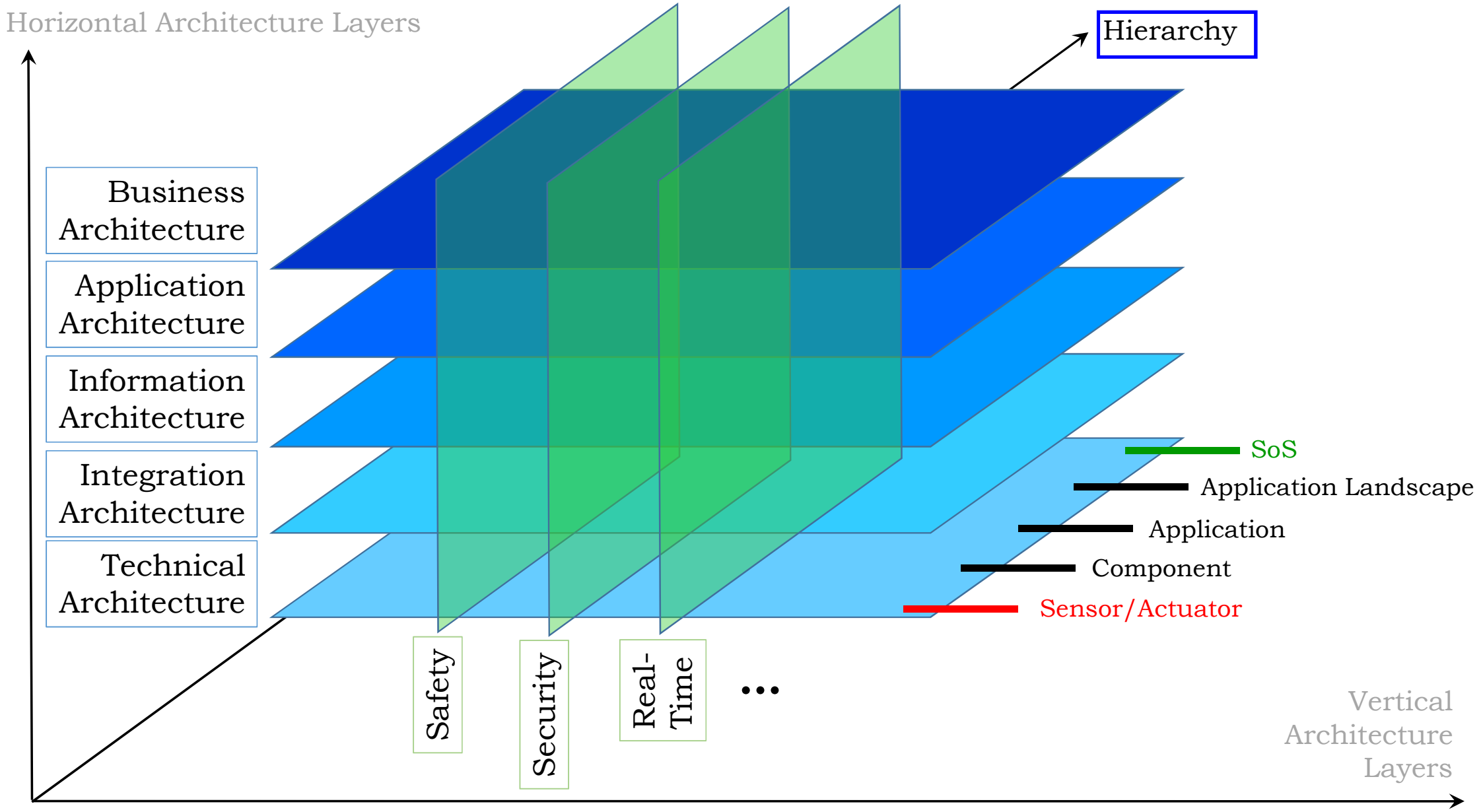
Auroop Ratan Ganguly, Stephen E. Flynn, Udit Bhatia:
Critical Infrastructures Resilience: *Policy and Engineering Principles*
Productivity Press, 2018. ISBN 978-1-4987-5863-5

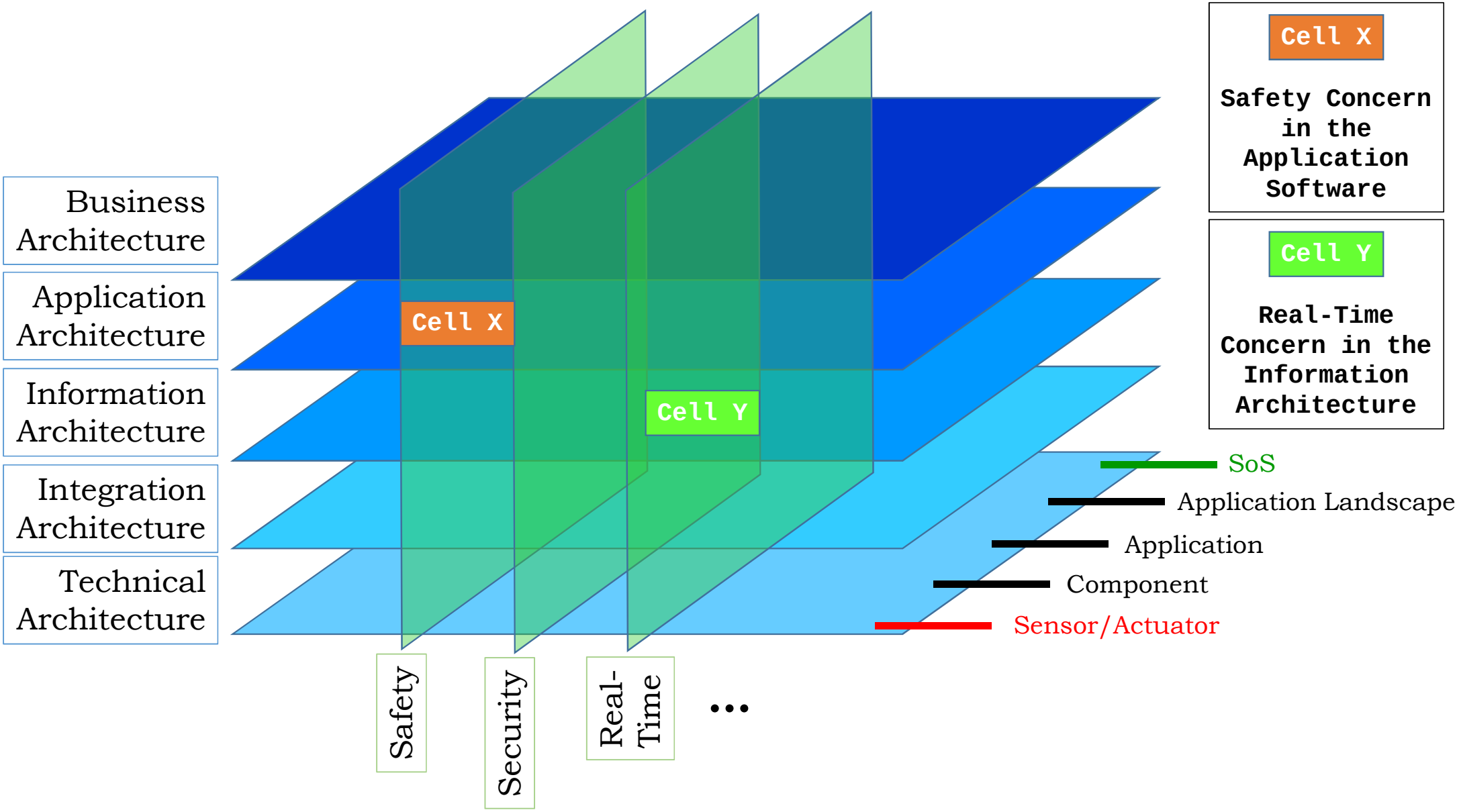
General Resilience

Architecture Principles


Horizontal Architecture Layers
Hierarchy
Business Architecture
Application Architecture
Information Architecture
Integration Architecture
Technical Architecture
Vertical Architecture Layers


Horizontal Architecture Layers
Hierarchy
Business Architecture
Application Architecture
Information Architecture
Integration Architecture
Technical Architecture
Safety
Security
Real-Time
...
Vertical Architecture Layers


Horizontal Architecture Layers
Hierarchy
Business Architecture
Application Architecture
Information Architecture
Integration Architecture
Technical Architecture
SoS
Application Landscape
Application
Component
Sensor/Actuator
Safety
Security
Real-Time
...
Vertical Architecture Layers


Business Architecture
Application Architecture
Information Architecture
Integration Architecture
Technical Architecture
Cell X
Cell Y
Safety
Security
Real-Time
...
Cell X
Safety Concern in the Application Software
Cell Y
Real-Time Concern in the Information Architecture
SoS
Application Landscape
Application
Component
Sensor/Actuator

Cell X = **Safety Concern in the Application Software**

Architecture Framework Cells =

Allow assignment, structuring, and separating of the functionality and of the quality properties of IT-systems to enable partitioning and life-cycle management.

⇒ **Formulation of Powerful Set of Architecture Principles**, e.g.:

NEVER implement security functionality in the applications software

… but only allow calls to the security functionality

«Canon of Orthogonality»

https://www.nprnis.com

Parts of Resilience Architecture

Security architecture
⇒ Protection of IT Assets

Safety architecture
⇒ Protection of people and property

Real-Time architecture
⇒ Assurance of hard timing properties

+ ...

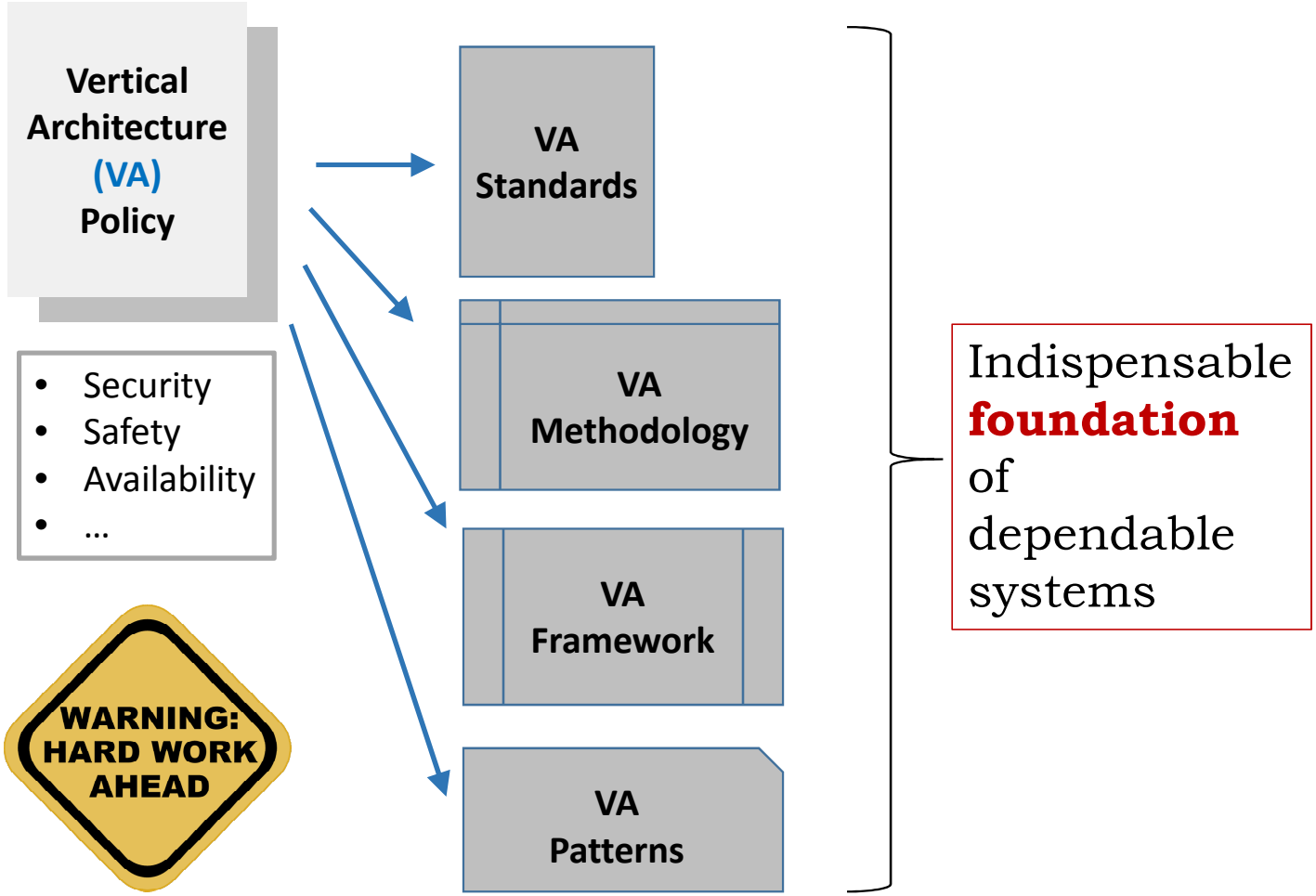
Security Architecture (Defense)

Safety Architecture (Accidents)

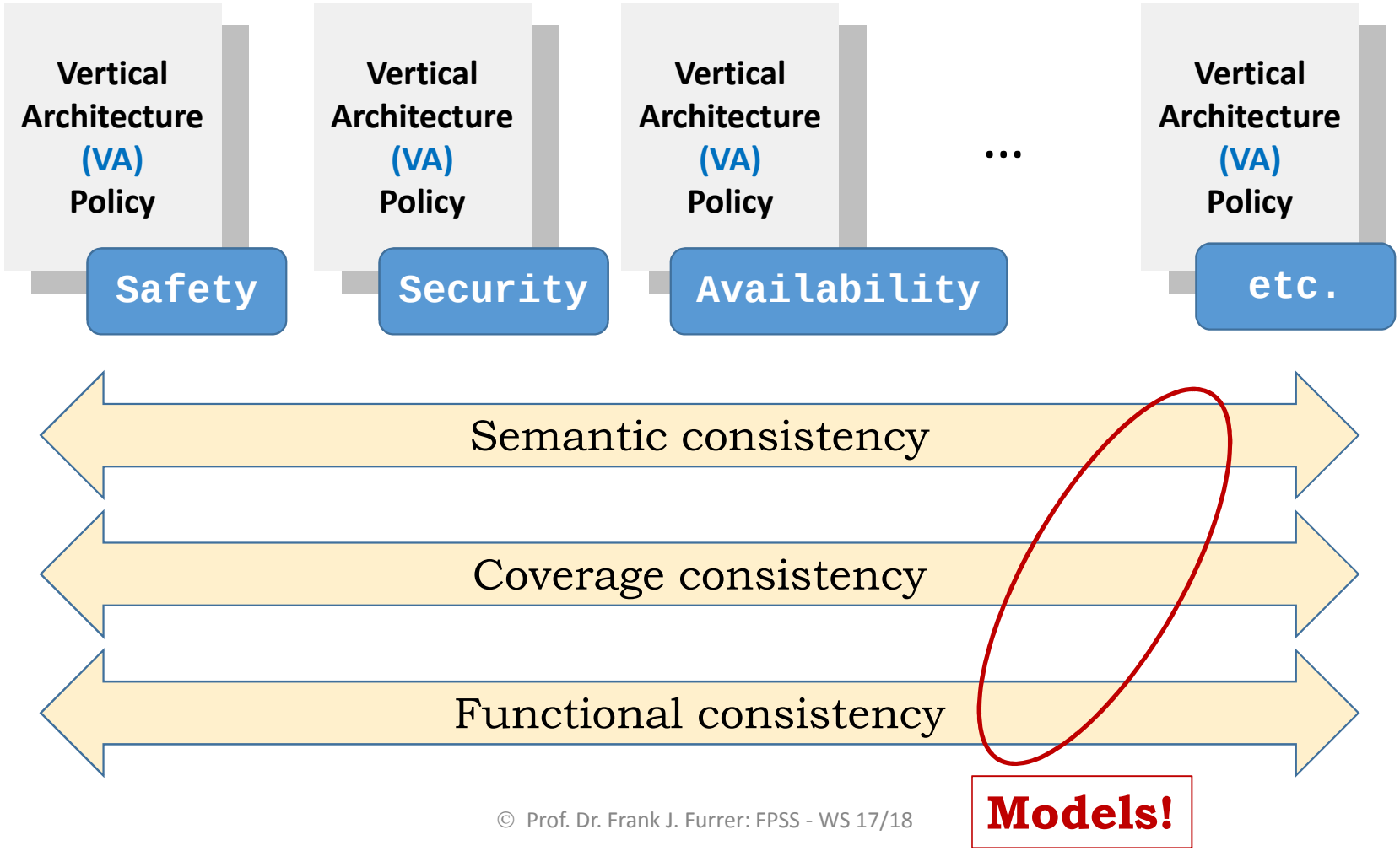
Performance Architecture (Real-Time)

System Management Architecture (Control)

etc.


Vertical Architecture (VA) Policy
• Security
• Safety
• Availability
• …
WARNING: HARD WORK AHEAD
VA Standards
VA Methodology
VA Framework
VA Patterns
Indispensable **foundation** of dependable systems

Consistency of Vertical Architectures



Example: Authentication/Authorization Architecture (*Security*)

Authentication (Digital Identity)

UserName

Password

#Cert

1. 3-element identification
2. Forced PW Changes (3 months)
3. Secure Registration Process
4. Regular Audits

Authorization (Access Control)

Access Control

Protection Asset
(Customer Information)

5. Assignment to a Role
6. Strict, time-limited access control
7. Access rights based on **least** privilege
8. Secure Rights Management

https://www.laneterralever.com

The quality and consistency of the vertical architectures determines the **dependability** of the system

However, … this represents a large amount of very difficult work

https://upload.wikimedia.org

Criticality
- Loss of life, property
- Reputation Damage
- Legal consequences
- …

high

low

VA effort

Criticality
- Loss of life, property
- Reputation Damage
- Legal consequences
- …

high

low

Answer:
⇒ **Risk-based approach**

High risk: High VA-effort
Low risk: Low VA-effort

VA effort

R2

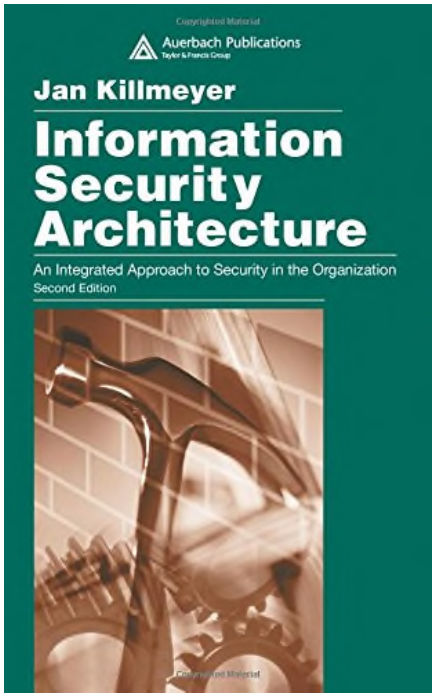
Resilience Architecture Principle R2:

Vertical Architectures

1. The dependability of the system is directly dependent on the quality of the vertical architectures (and the quality of their implementation)
2. Match the quality of the vertical architectures to the risk/damage potential of your application (Caution: be on the safe side)
3. Continuously maintain/evolve your vertical architectures in sync with changing environments and requirements

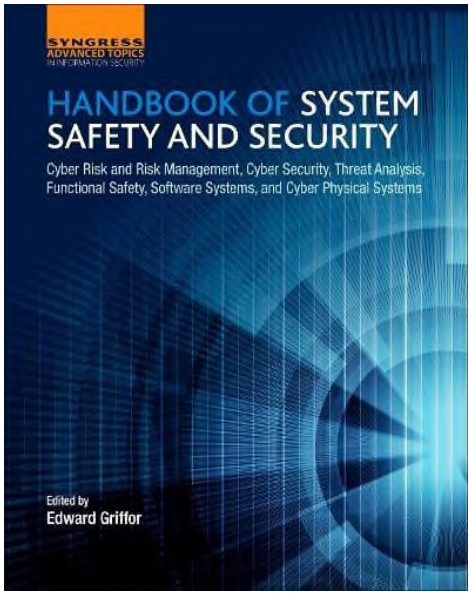
Justification: Vertical architectures are at the core of dependability. A great bandwith in quality of vertical architectures exists. If the quality is too low, your systems/applications may be at risk.

Textbook



Jan Killmeyer:
Information Security Architecture: *An Integrated Approach To Security in The Organization*
Auerbach Publishers Inc., USA, 2nd edition 2006. ISBN 978-0-849-31549-7

Textbook



Edward Griffor (Editor):
Handbook of System Safety and Security: *Cyber Risk and Risk Management, Cyber Security, Threat Analysis, Functional Safety, Software Systems, and Cyber Physical Systems*
Syngress Media, USA, 2016. ISBN 978-0-128-03773-7

General Resilience

Architecture Principles

R_3

Fault Containment Regions

http://ems12lead.com

http://www.auto-repair-help.com

Fault
Incident

Error
Failure
Malfunction

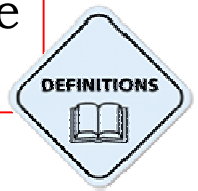
Faults and *incidents* are inevitable in today's complex systems

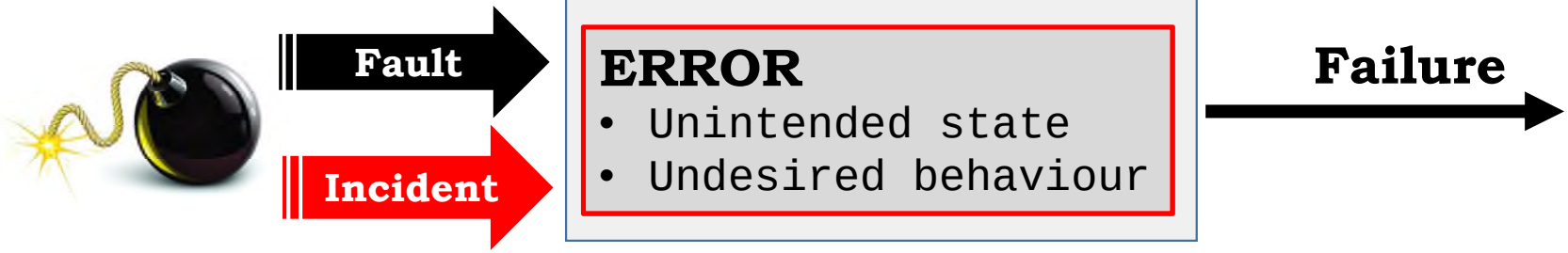
… they are the normal case

Accident

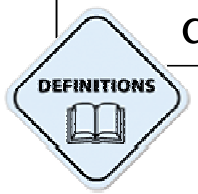
http://pb-attorneys.com

A **fault** or **incident** is the cause of an error in the system part ("Input")

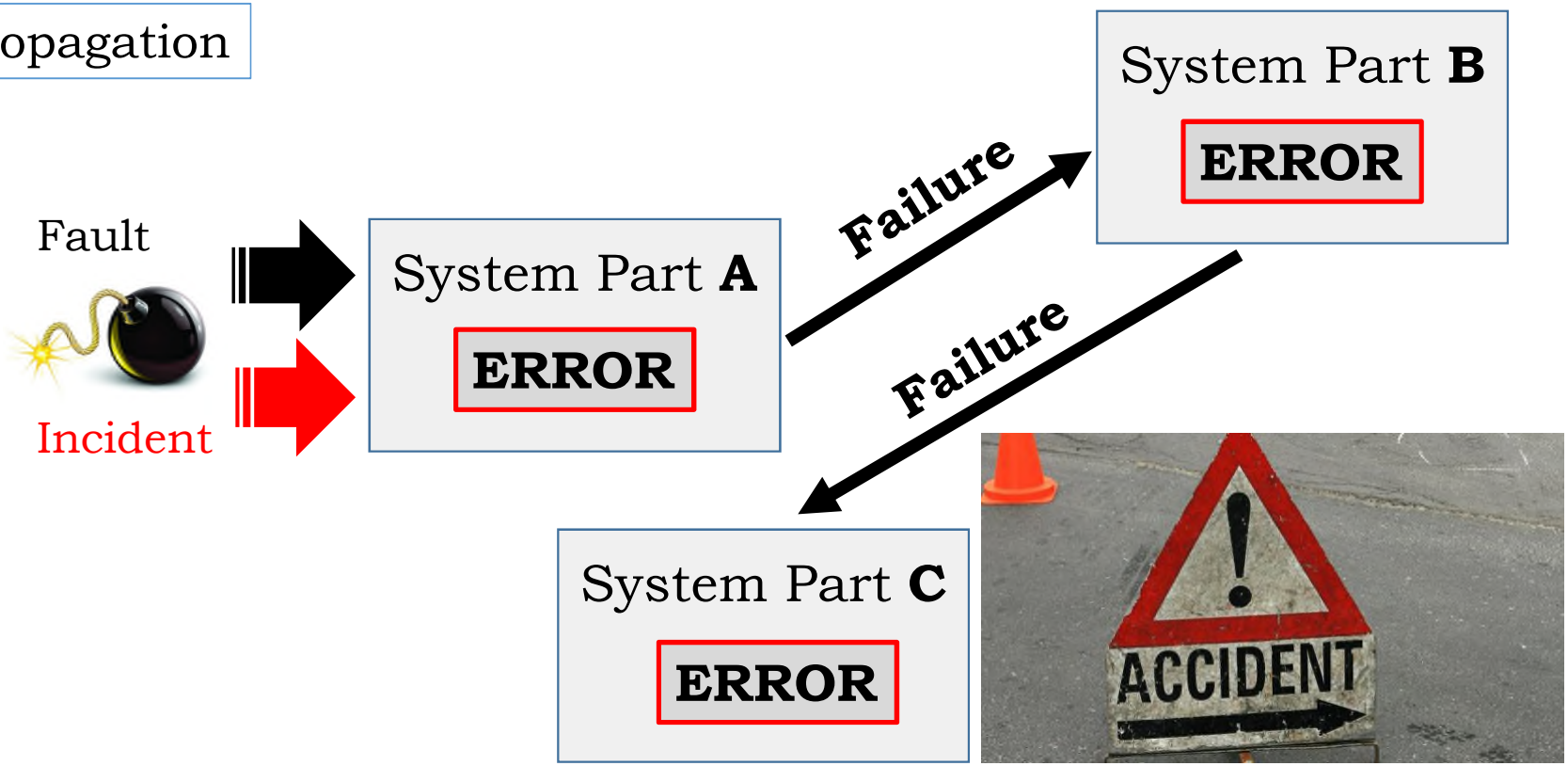




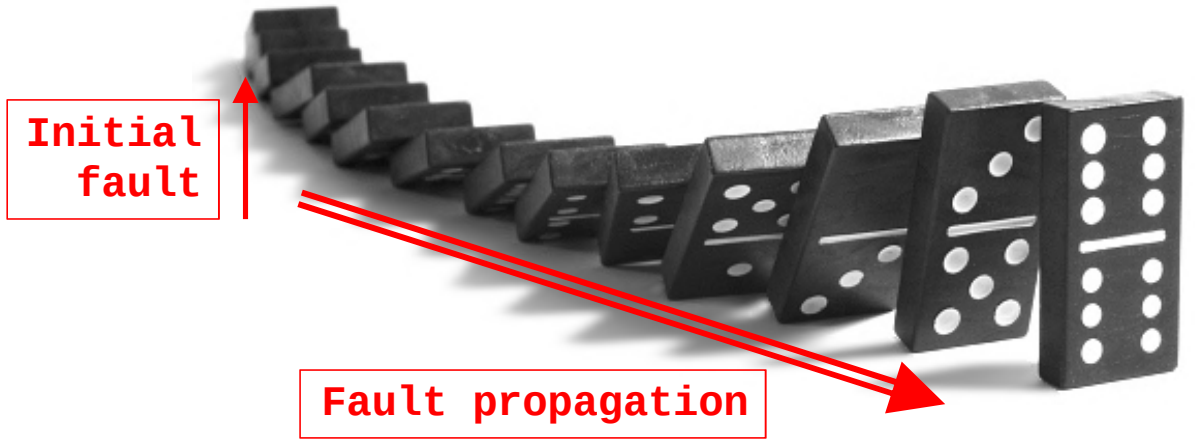
A **failure** is an event that occurs when the delivered service deviates from correct service



Fault Propagation



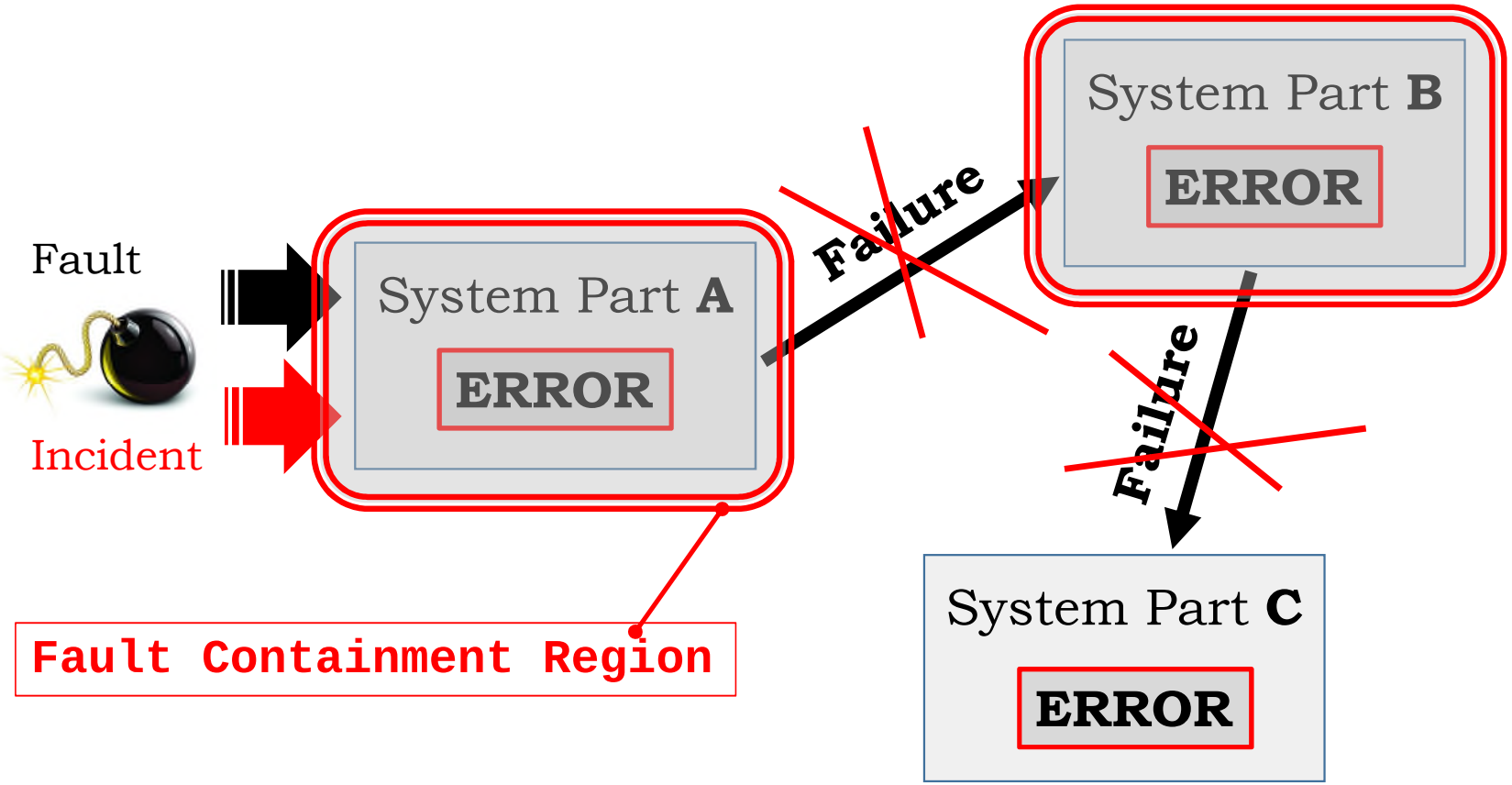
The consequences of a *fault* – the ensuing *error* – can **propagate** either by an erroneous message or by an erroneous output action of the faulty part



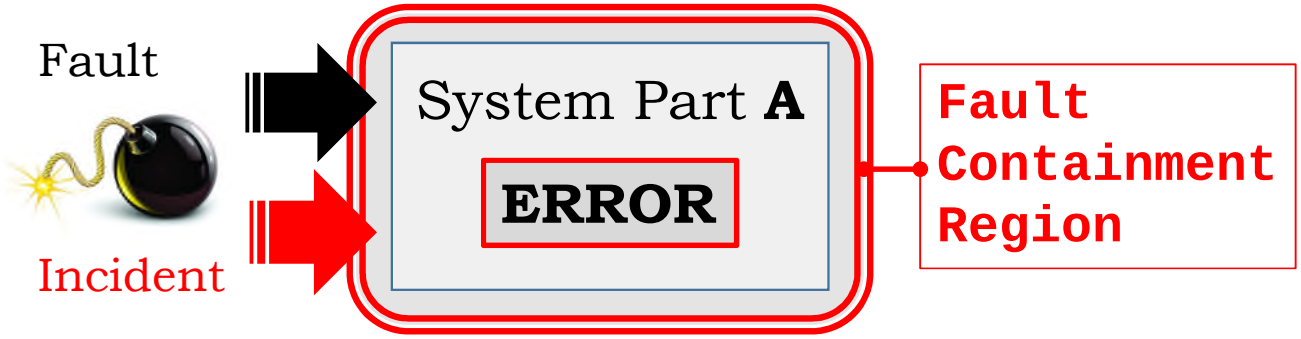
Fault propagation can consecutively affect system parts (⇒ Domino effect)

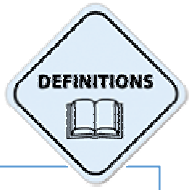
The result may be severe malfunctions or the loss of the system

Fault propagation is difficult to predict



Build **error** propagation boundaries around each system part





Fault tolerant systems are often built around the concept of ***fault containment regions*** (FCRs).

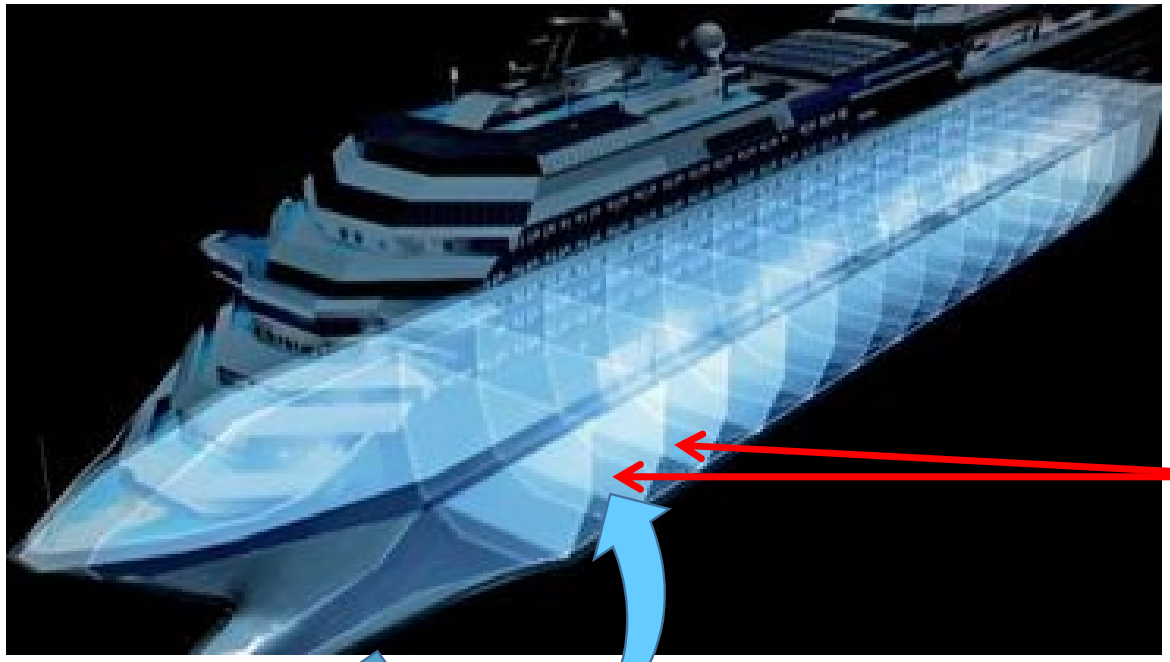
The primary goal of a FCR is to limit the effects of a fault and *prevent the propagation* of errors from one region of the system to another.

A FCR is a subsystem that will operate correctly regardless of any arbitrary fault outside the region.

FCRs may be physically separated, electrically isolated, and have independent power supplies.

https://shemesh.larc.nasa.gov/fm/papers/Butler-TM-2008-215108-Primer-FT.pdf

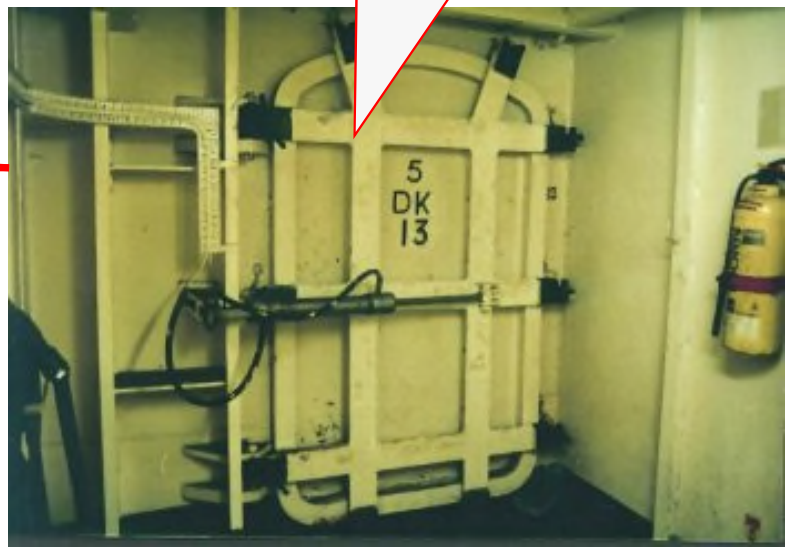
Example: Watertight Ship Compartments



http://www.daserste.de

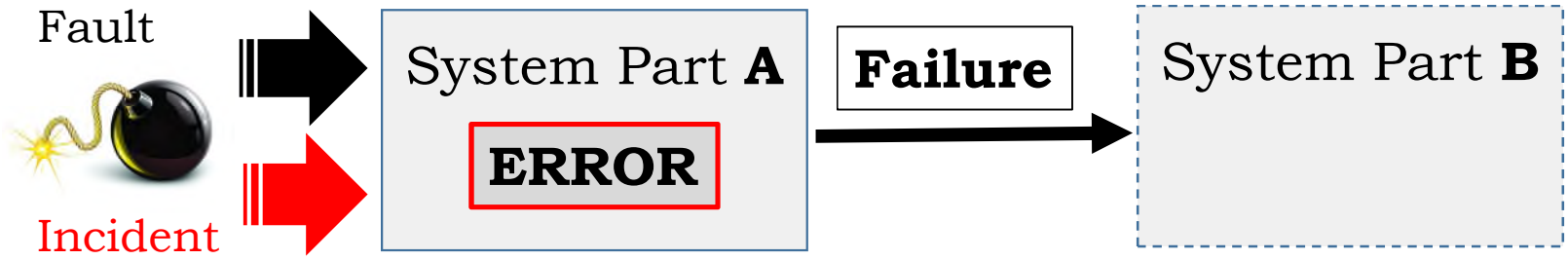
Incident = Hull Damage

Water inrush is *contained* to few compartments due to **bulkheads**



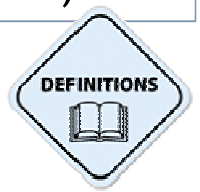
http://www.dieselduck.info

⇒ The «fault» *cannot* propagate

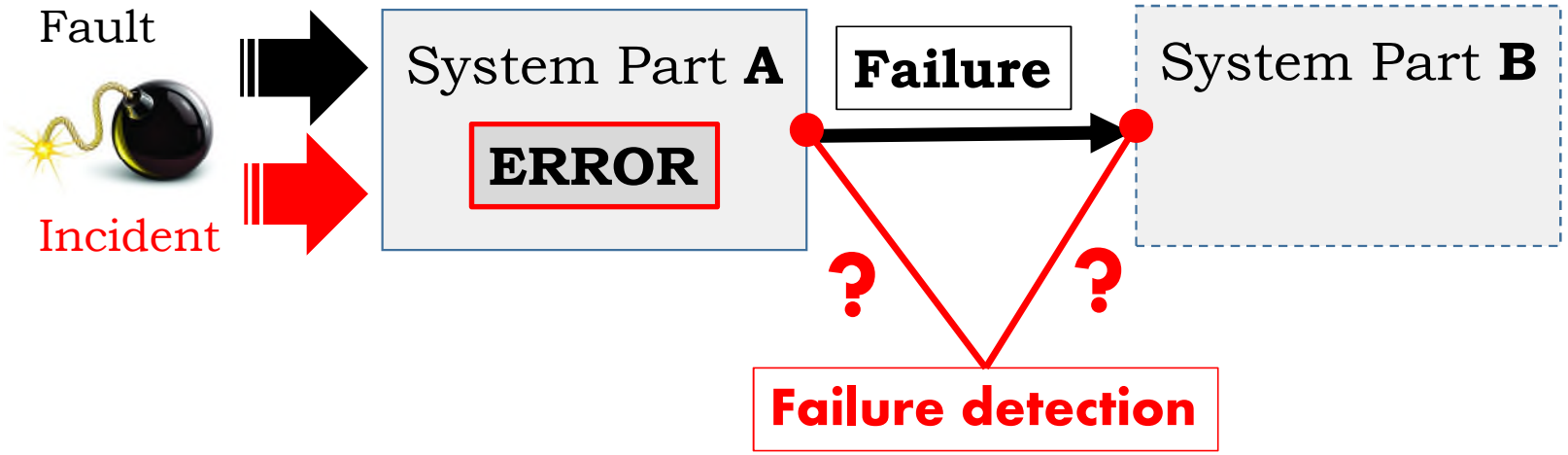


Typology of Failures:

- *Temporal* domain (too early, too late, not at all)
- *Value* domain (incorrect or out of range values)
- *Content* domain (wrong, faked or unallowed content)



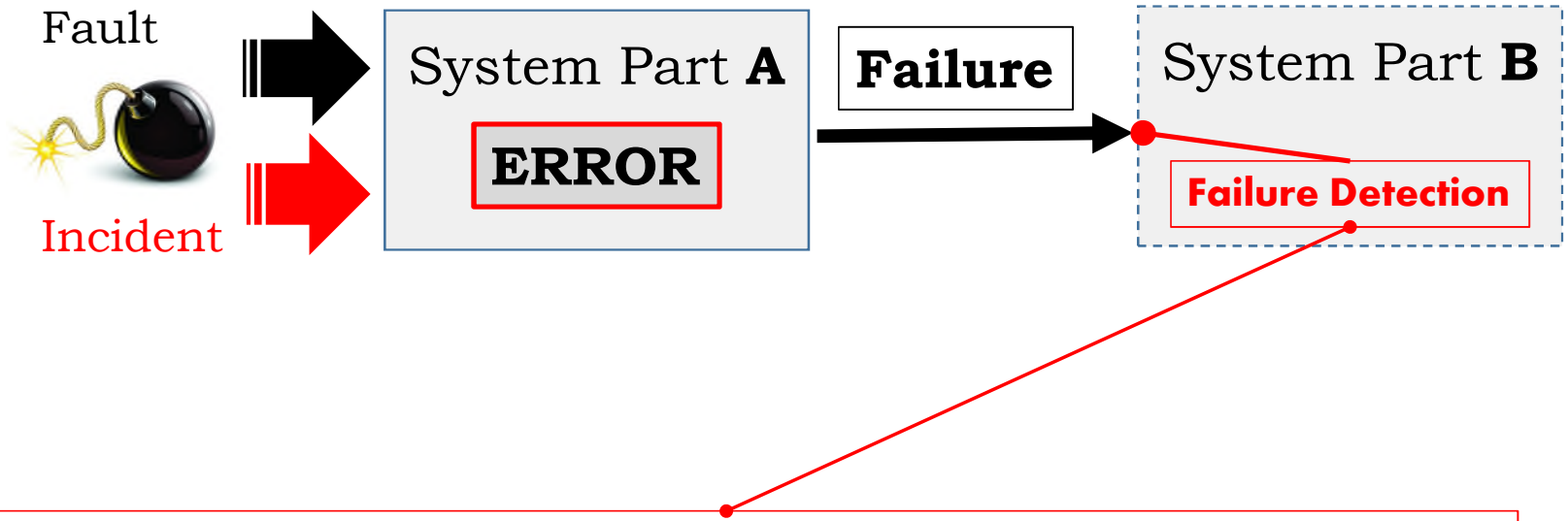
Failure Detection



A failure can only be detected:

- If the *observed* behaviour of a component can be judged in relation to the *intended* behaviour
- If the system contains some form of *redundant* information about the intended behaviour

Failure Handling

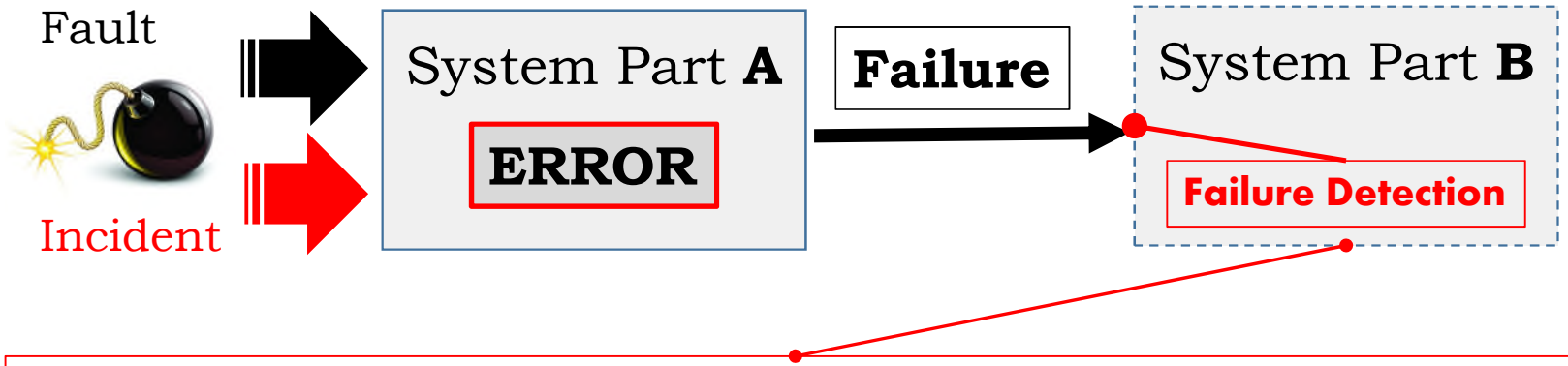


- *Temporal* domain (too early, too late, not at all)
- *Value* domain (incorrect or out of range values)

`Tables Calculations`

- *Content* domain (wrong, faked or unallowed content) ⇐ `difficult`

Failure Handling



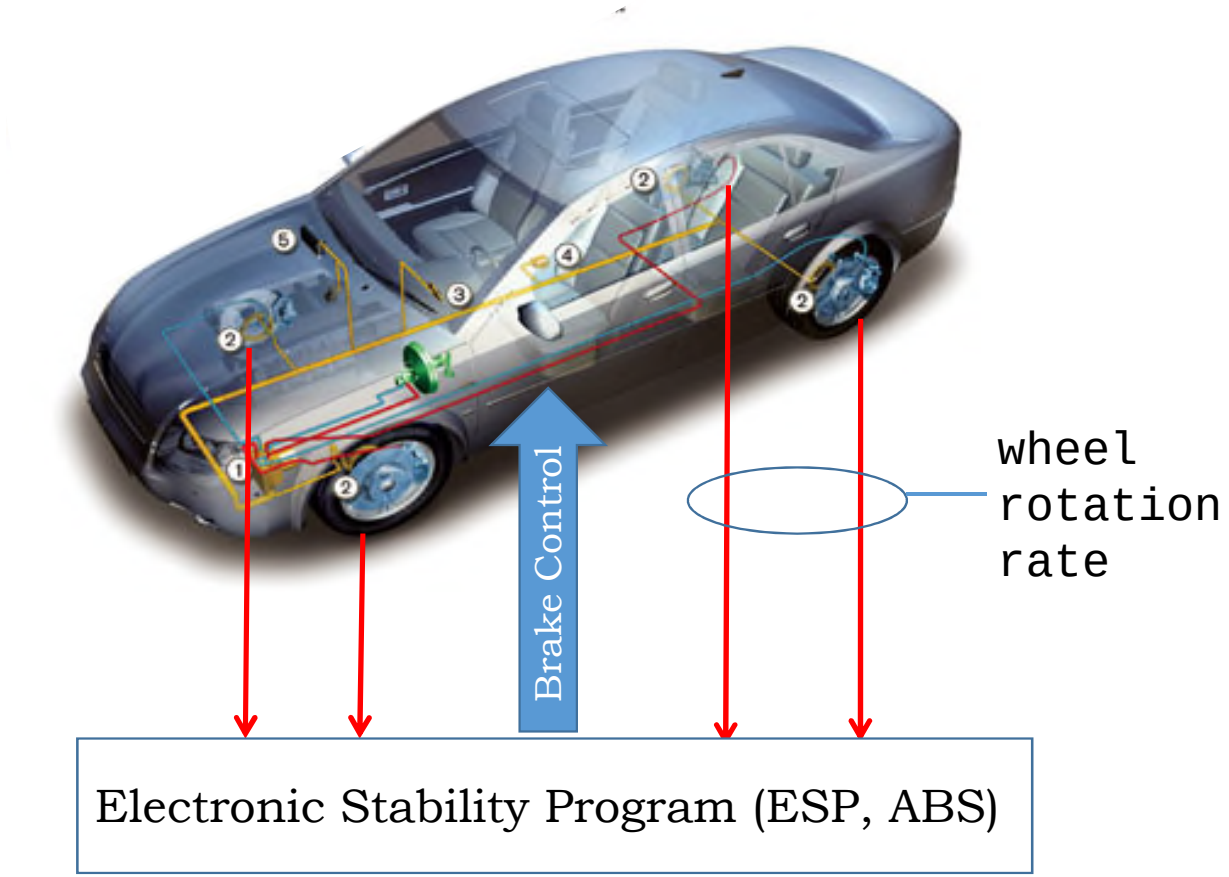
Reaction?

- Recovery (Fault tolerance)
- Degraded operation
- Safe state
- [Malfunction or Crash]

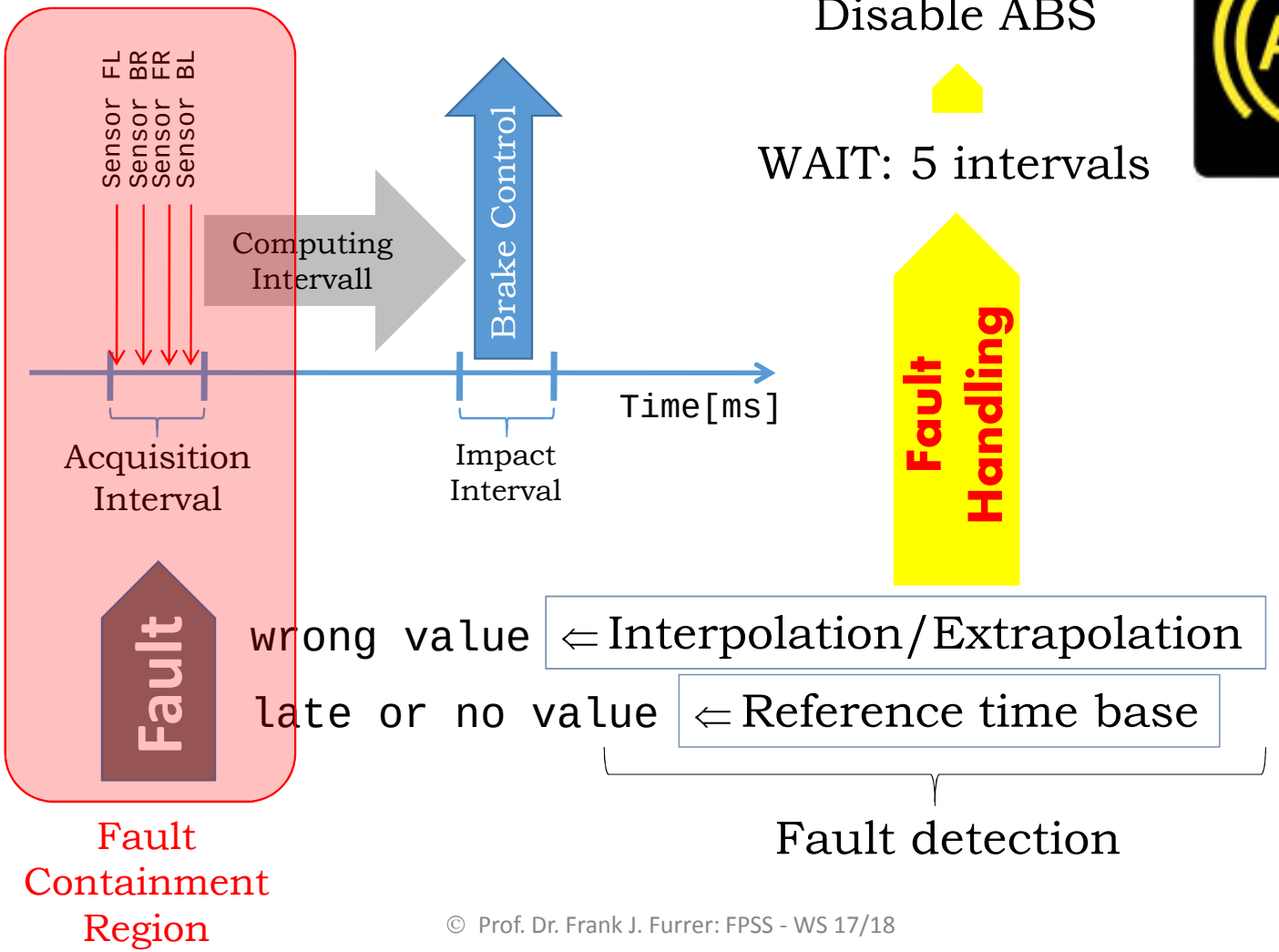
Recovery

Degraded operation

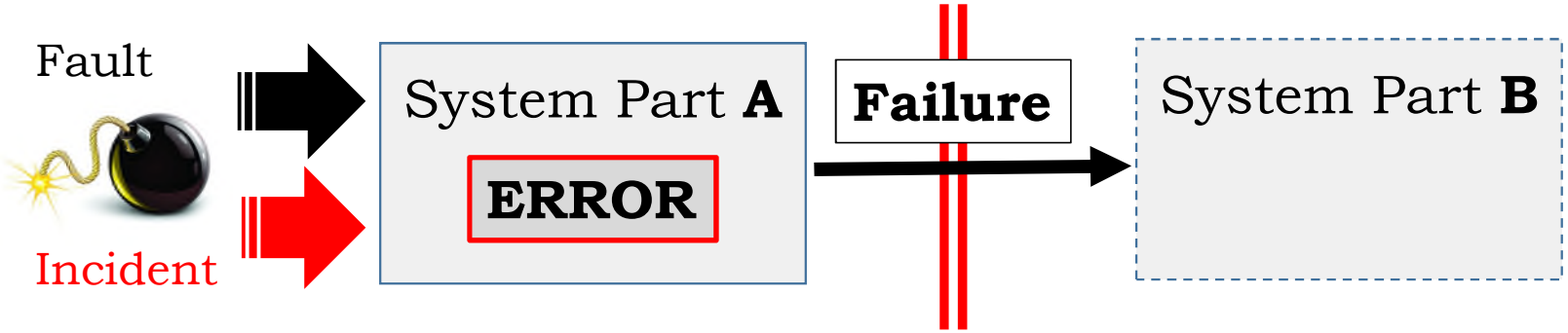
Example: Car ABS (1/2)



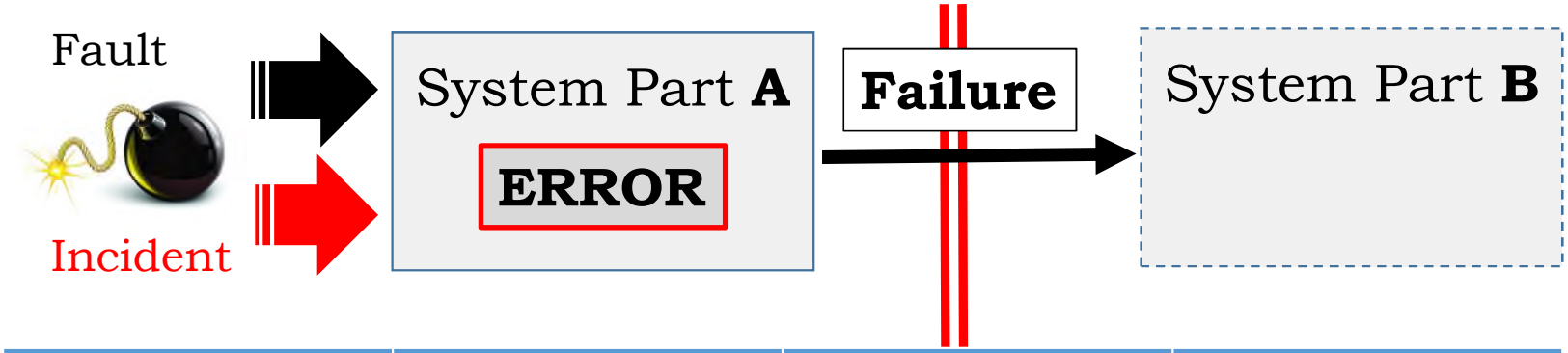
Example: Car ABS (2/2)



Constructing Fault Containment:

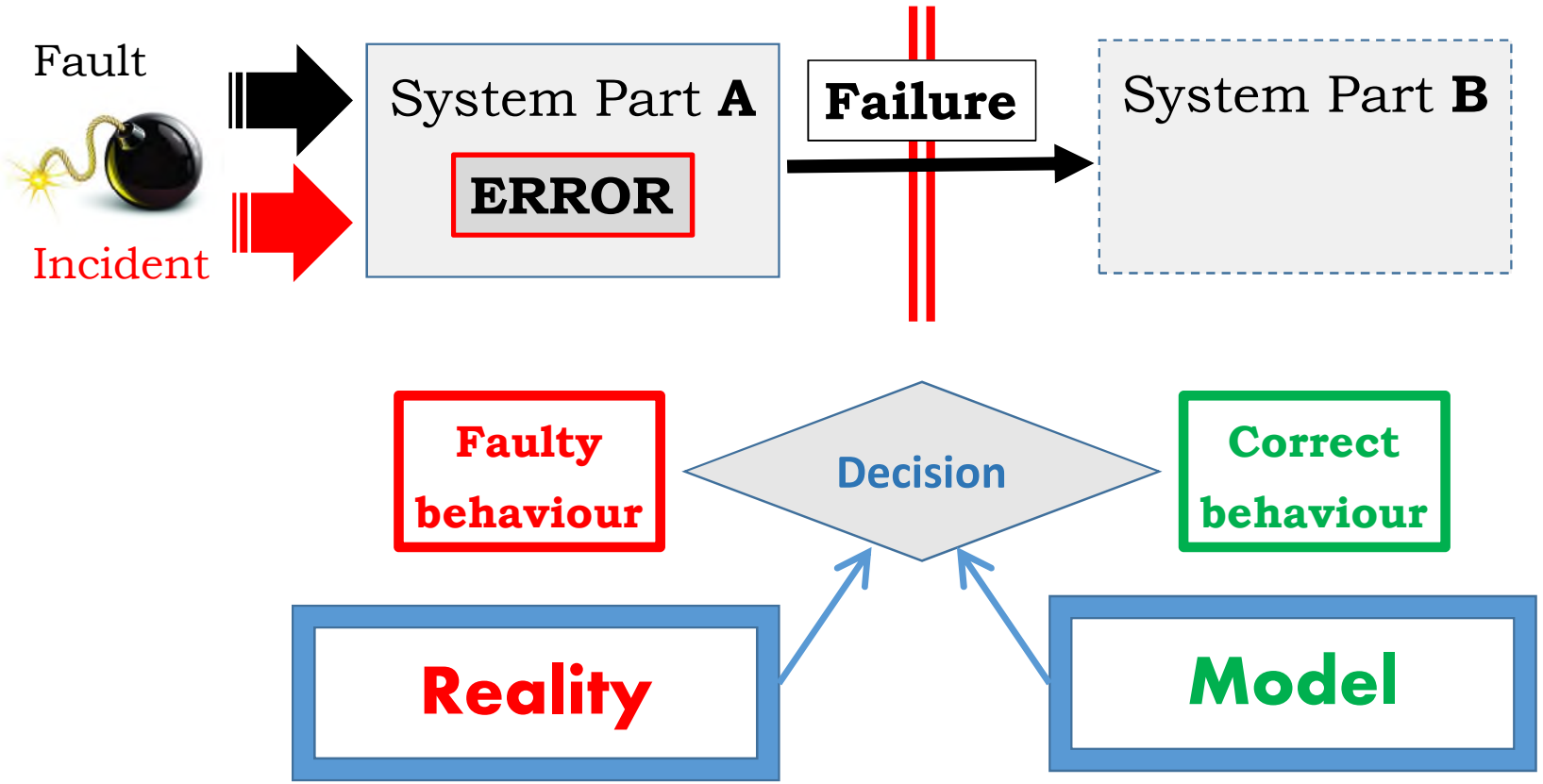


- DETECT erroneous output of system A
- Inhibit transfer to system B
- Failure handling in system B
- Corrective action in system A (← human)

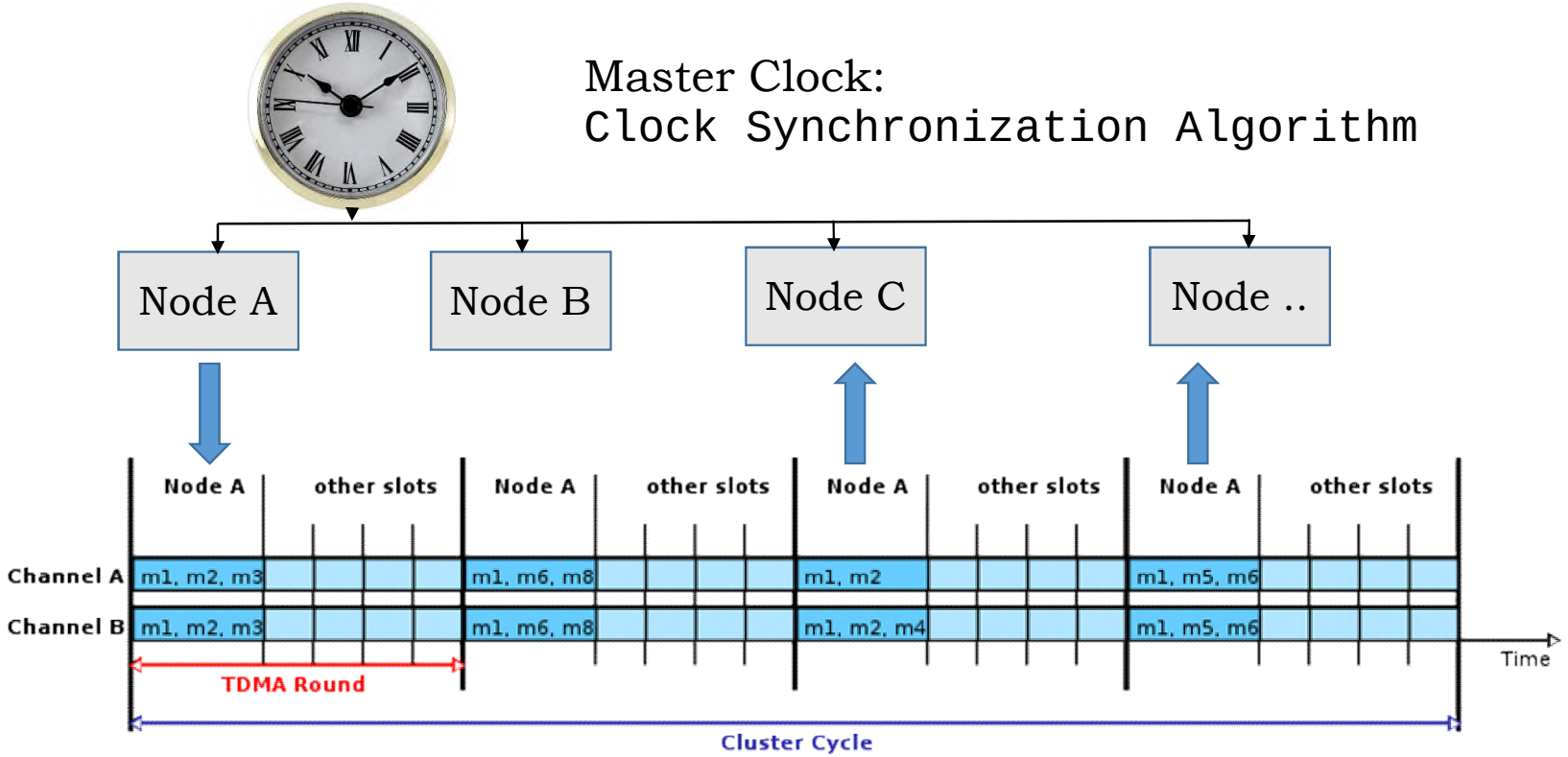


Type	Detection	Failure Handling	Required Info
Temporal fault (too early, too late, not at all)	Timing frame (global clock) & timing model	Repeat (N x) Redundancy Fail safe state	Time reference
Value fault (incorrect or out of range values)	Interface contract violation	Repeat (N x) Algorithms Fail safe state	Formal contract
Content fault (wrong, faked or unallowed content)	Integrity protection (e.g. # certs)	Repeat request Redundancy Fail safe state	Model

Constructing Fault Containment:



Example: *Temporal* Failures (Time-Triggered Architecture **TTA**)



The messages are transported in exactly defined and assigned time slots, based on precise clock synchronization in all nodes

R3

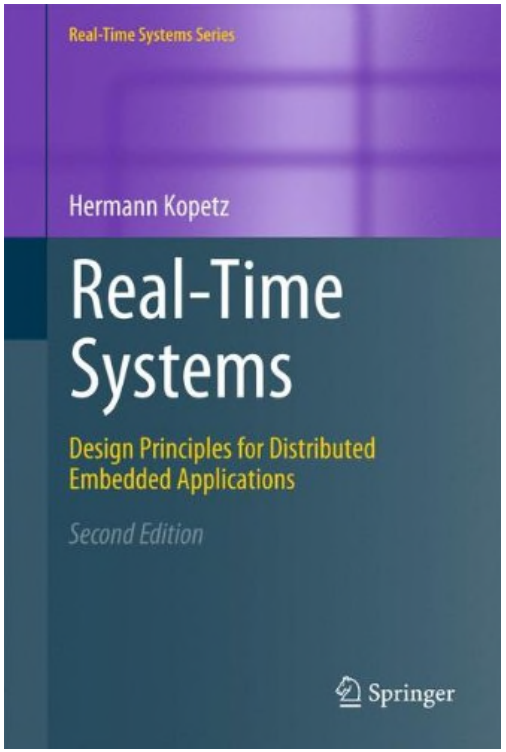
Resilience Architecture Principle R3:

Fault Containment Regions

1. Partition the system into fault containment regions
2. Build error propagation boundaries around each system part (⇒ Interfaces)
3. Provide sufficient redundant information about the intended behavior of the system parts (components)

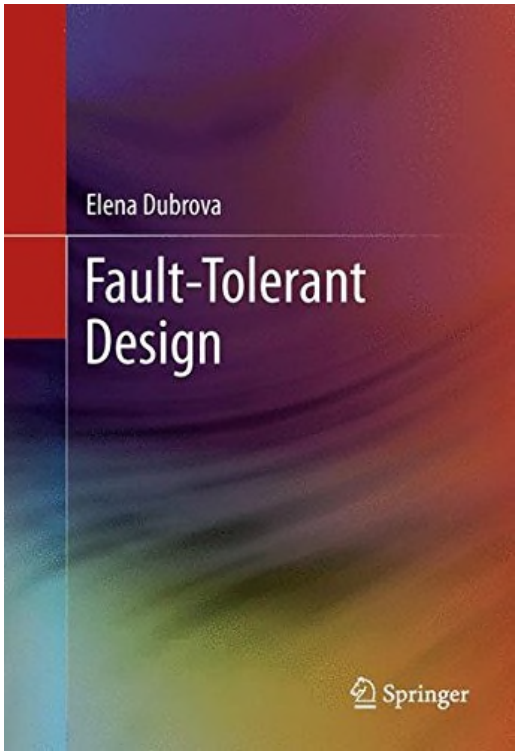
Justification: A fault or incident causing an error or disruption in one part (component) of the system should not propagate to other parts of the system and thus cause a sequence of errors and failures

Textbook



Hermann Kopetz:
Real-Time Systems: *Design Principles for Distributed Embedded Applications*
Springer-Verlag, Germany, 2nd edition, 2011.
ISBN 978-1-441-98236-0

Textbook

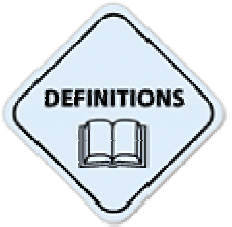


Elena Dubrova
Fault-Tolerant Design
Springer-Verlag, Germany, 2013. ISBN 978-1-461-42112-2

General Resilience

Architecture Principles

Single Points of Failure



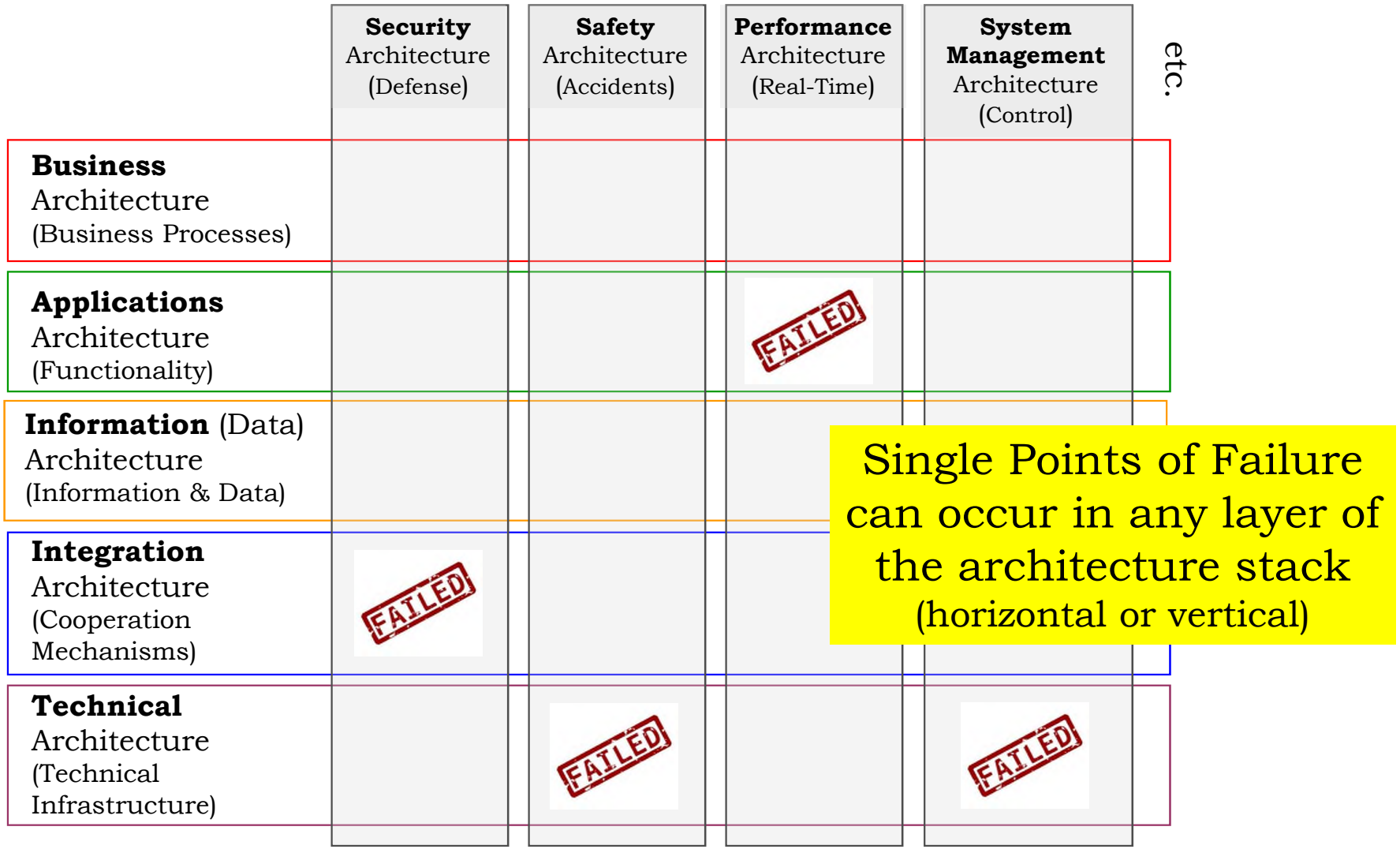
A single point of failure (SPOF)

is a part of a system that,

if it *fails*,

will stop the *entire* system from working

Example: Computer Network


Security Architecture (Defense)
Safety Architecture (Accidents)
Performance Architecture (Real-Time)
System Management Architecture (Control)
etc.
Business Architecture (Business Processes)
Applications Architecture (Functionality)
Information (Data) Architecture (Information & Data)
Integration Architecture (Cooperation Mechanisms)
Technical Architecture (Technical Infrastructure)
FAILED
FAILED
FAILED
FAILED
Single Points of Failure can occur in any layer of the architecture stack (horizontal or vertical)

Single Points of Failure (SPOF) are a high risk for dependabillity

http://favim.com

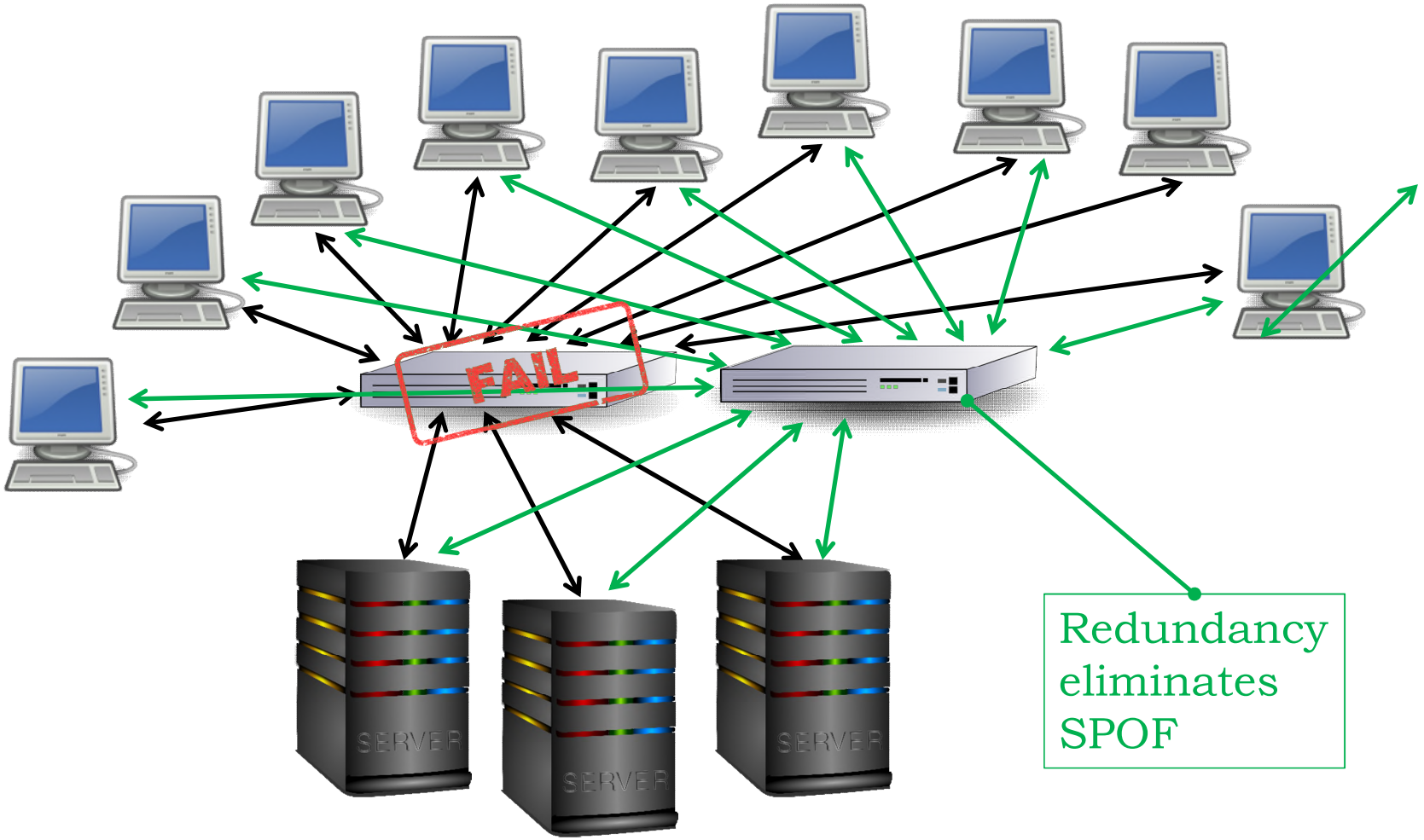
Single Points of Failure (SPOF) may be well hidden in a system and sometimes difficult to find

Single Points of Failure (SPOF) are <u>eliminated</u> by:

a) Intelligent architecture/design

b) Introduction of *redundancy*

Example: Computer Network



R4

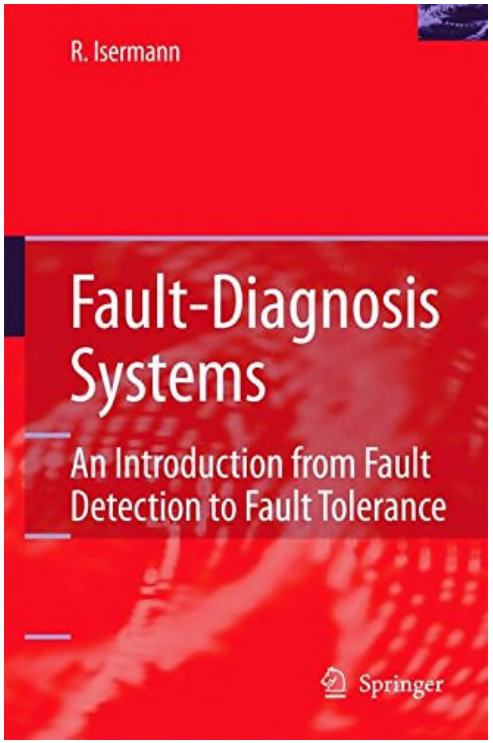
Resilience Architecture Principle R4:

Single Points of Failure

1. Identify possible single points of failure early in the architecture/design process (Note: single points of failure can occur on all levels of the architecture stack)
2. Eliminate single points of failure, e.g. by introducing redundancy

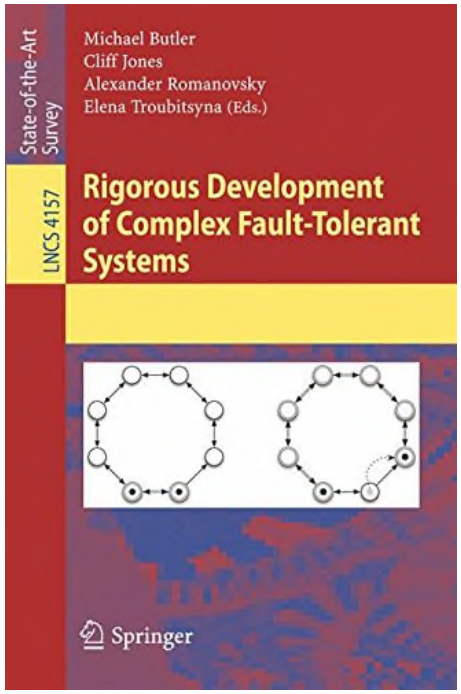
Justification: Any single point of failure is a great risk for a dependable system. They must therefore be avoided

Textbook



Rolf Isermann
Fault-Diagnosis Systems: *An Introduction from Fault Detection to Fault Tolerance*
Springer-Verlag, Germany, 2006. ISBN 978-3-540-24112-6

Textbook



Michael Butler, Cliff Jones, Alexander Romanovsky, Elena Troubitsyna (Editors):
Rigorous Development of Complex Fault-Tolerant Systems
Springer Berlin, Heidelberg, 2008. ISBN 978-3-540-48265-9

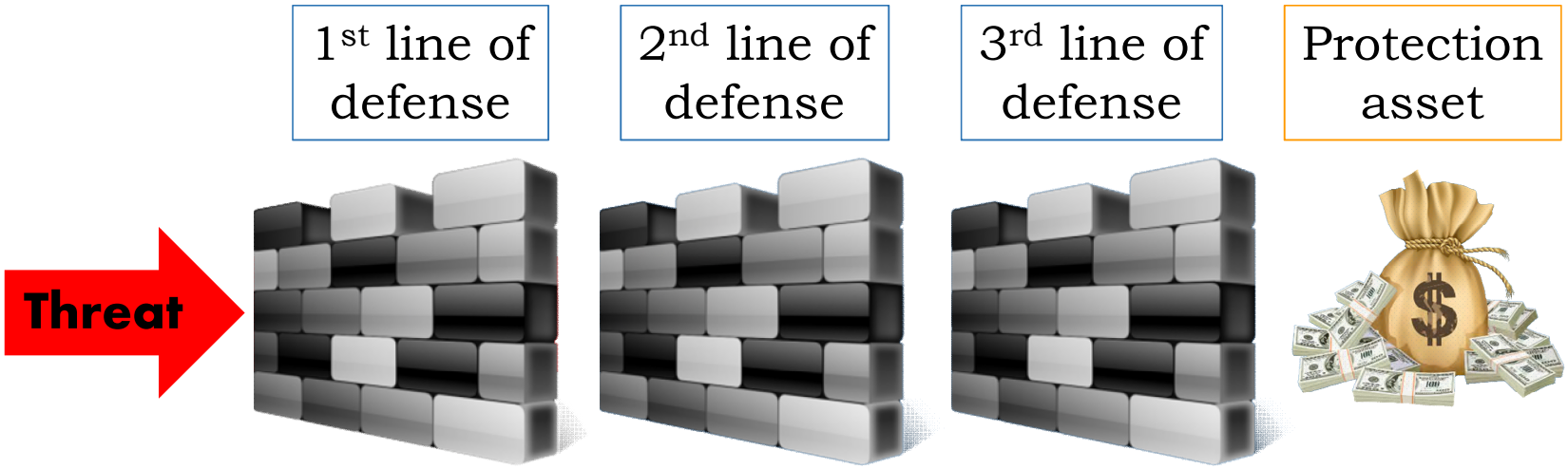
General Resilience

Architecture Principles

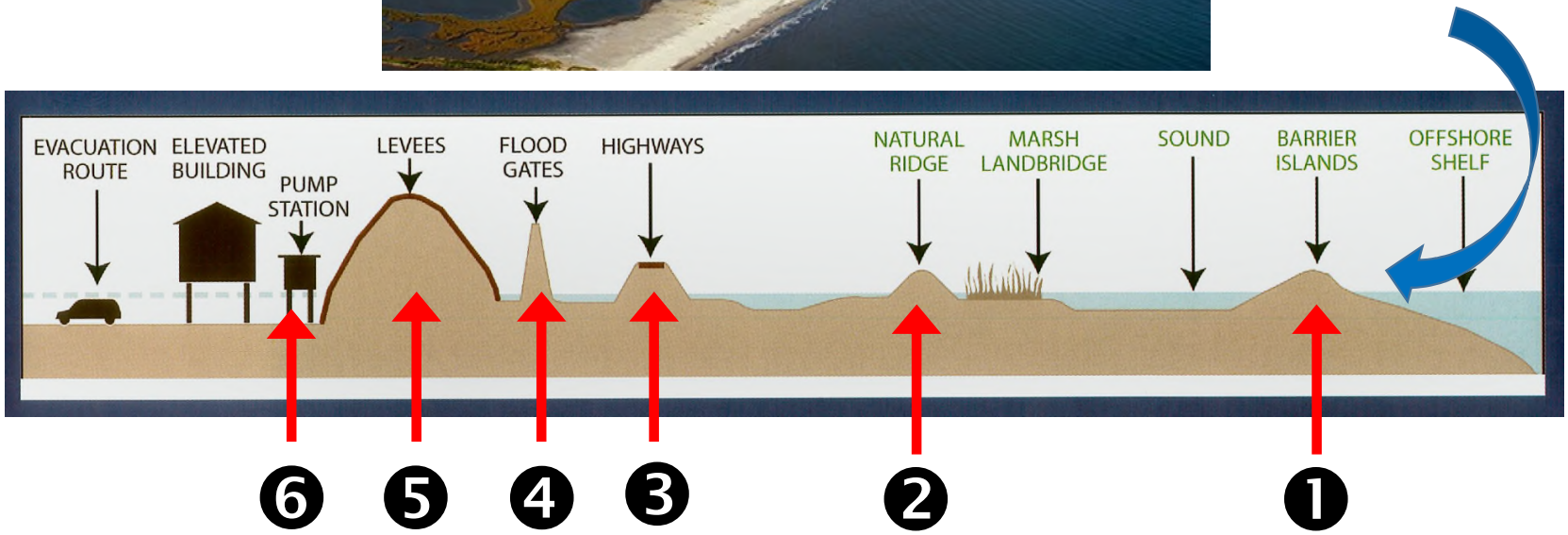
R_5

Multiple Lines of Defense

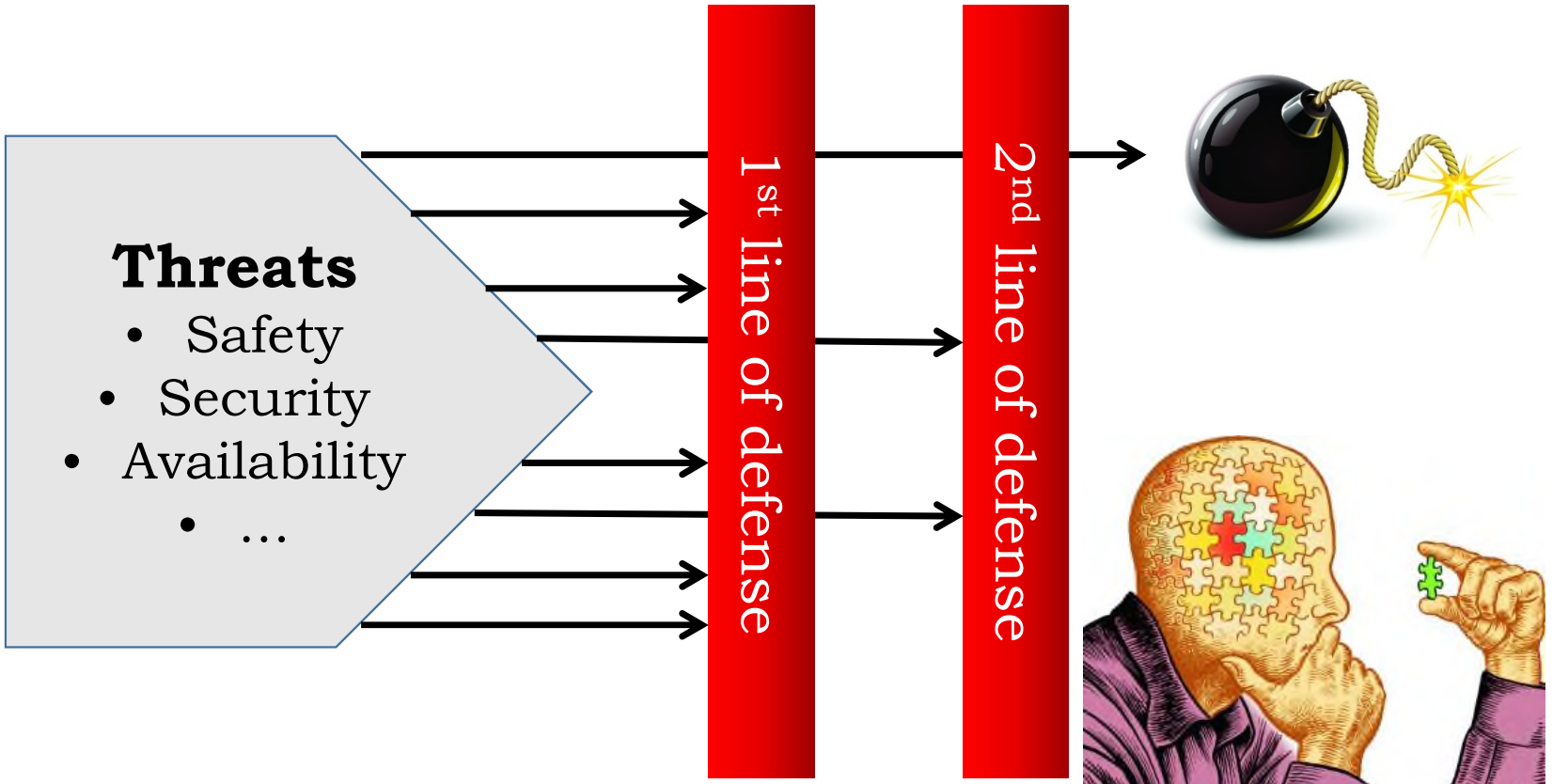
Multiple lines of defense represents the use of *multiple* computer techniques to help mitigate the risk of one component of the defense being compromised or circumvented



Example: South-East Louisiana Coast Hurricane Flood Protection



Multiple Lines of Defense Strategy



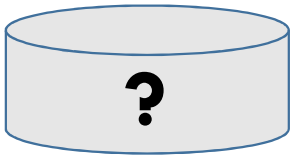
http://www.thinkingbusinessblog.com

Example: Security in Fund Transfer

1st line of defense

Step 1: Login

Step 2: Input Transfer Order



2nd line of defense

Step 3: Bank Checks **receiving account**

3rd line of defense

Step 4: Bank asks for signature

R5

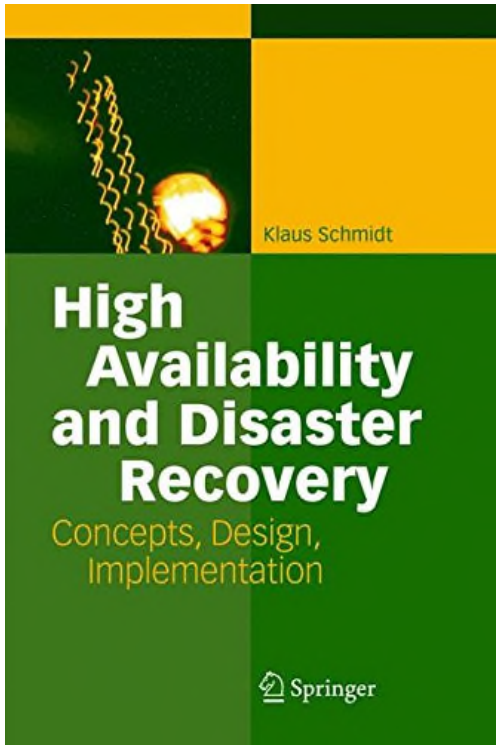
Resilience Architecture Principle R5:

Multiple Lines of Defense

1. For each threat and incident implement multiple, independent lines of defense
2. For each line of defense use different methods, techniques and technologies

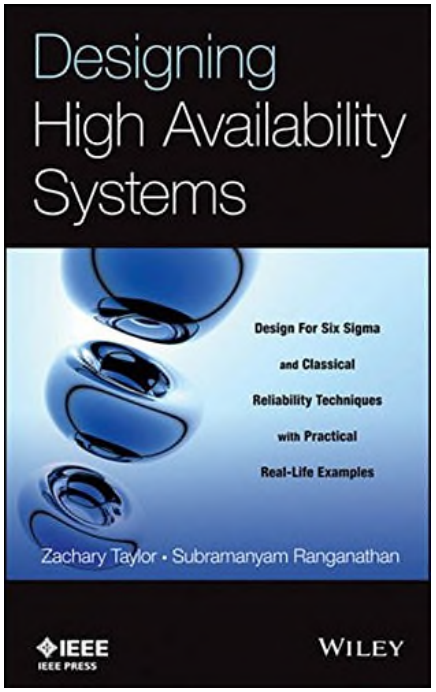
Justification: If a line of defense is overcome as a consequence of an incident, the second (third, …) line of defense may mitigate the impact of the incident

Textbook



Klaus Schmidt:
High Availability and Disaster Recovery: *Concepts, Design, Implementation*
Springer-Verlag, Germany, 2006. ISBN 978-3-540-24460-8

Textbook



Zachary Taylor, Subramanyam Ranganathan:
Designing High Availability Systems: *DFSS and Classical Reliability Techniques with Practical Real Life Examples*
Wiley-IEEE Press, USA, 2013. ISBN 978-1-118-55112-7

General Resilience

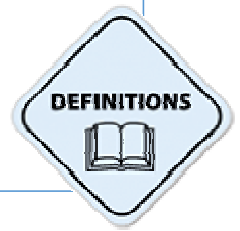
Architecture Principles

Fail-Safe States

https://media.licdn.com

Fail-safe means that a system will not endanger lives or property when it fails.

It will go into a *fail-safe state* and stop working.



A fail-safe system does not mean that failure is impossible or improbable – but that the system's design and implementation prevent ***unsafe consequences*** of the failure

https://img.clipartxtras.com

"As engineers we sometimes find designing equipment to be well-built is much easier than designing it to fail predictably"

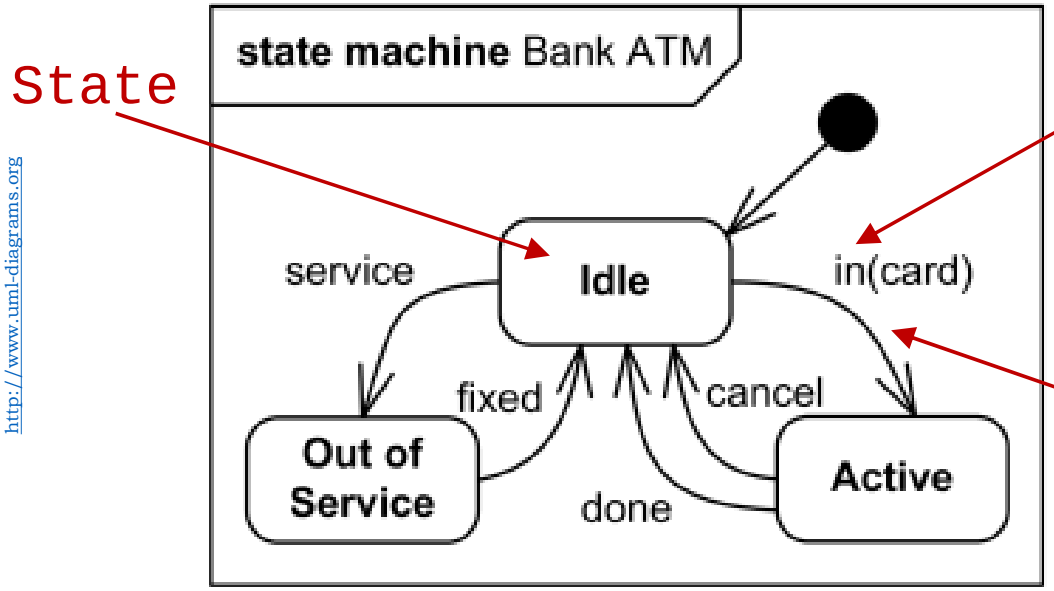
Peter Herena, 2011

Fail-safe means that a system will not endanger lives or property when it **fails**

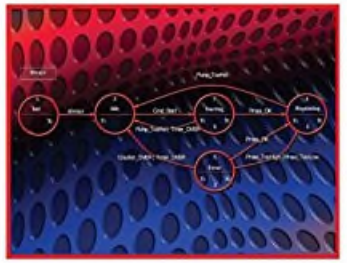
What is the fail-safe state for?:

Fail-Safety is a Property of the End-Product

Systems as State-Machines



State-Chart Diagram



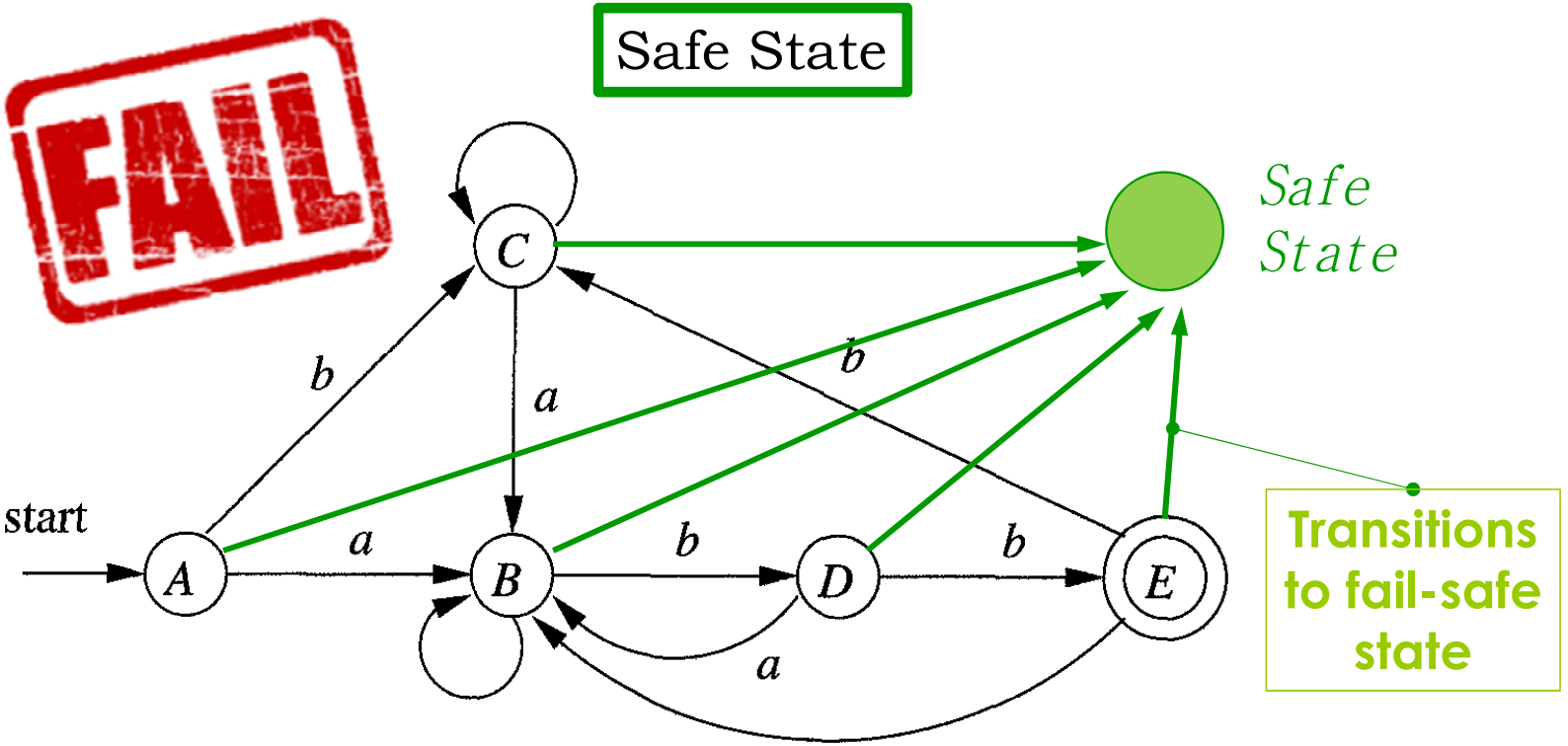
Modeling Software with Finite State Machines

A Practical Approach

Ferdinand Wagner
Ruedi Schmuki
Thomas Wagner
Peter Wolstenholme

http://www.uml-diagrams.org

The operation of the system is a sequence of states.
A state change is triggered by an event.



Safe State

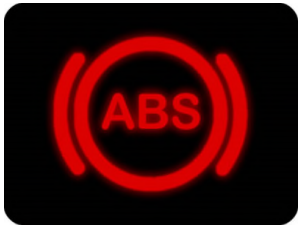
Which is a safe state?

How can we find a safe state?

Difficult Engineering Task

Safe State

STOP while avoiding an accident
⇒ Human intervention



NO safe state necessary (**uncritical**)

Safe-states only for subsystems
Revert to human intervention

NO safe state (**critical**)

Safe State – **Example 1**: Train Signalling System (1/2)

http://www.theguardian.com

A train signalling system controls the lights and switches to assure an accident-free train traffic

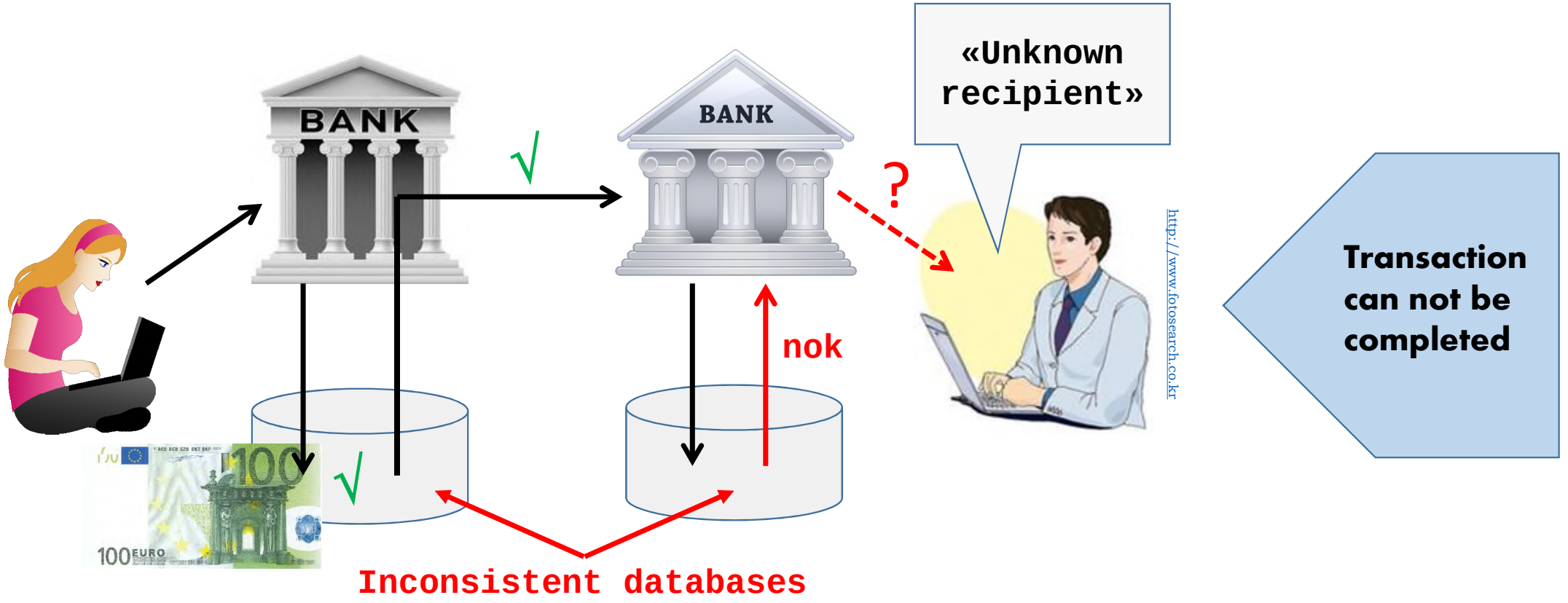
Safe State – Example 1: Train Signalling System (2/2)

Safe State Which is the safe state for this system?

Immediately STOP all traffic
→ **Human Operation**

Affected Area

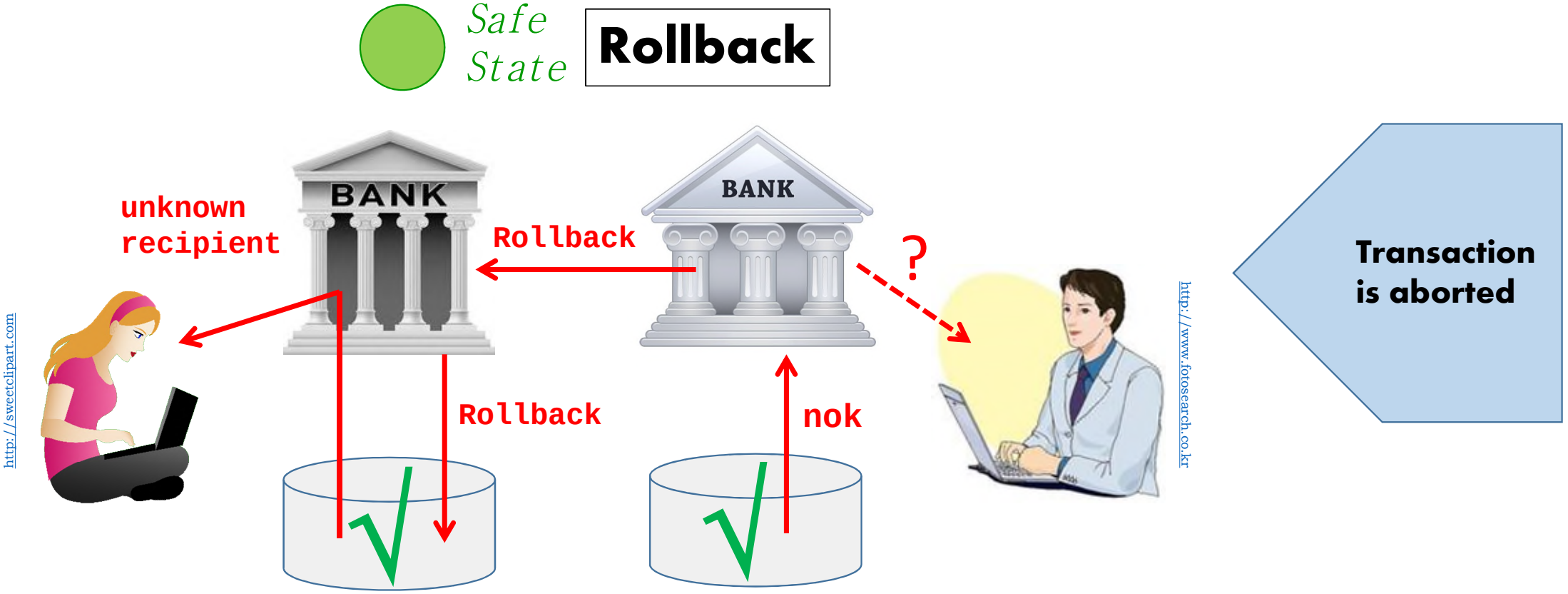
Safe State – Example 2: Transaction Rollback (1/2)



Safe State

Aborted Transaction: Which is the Safe State?

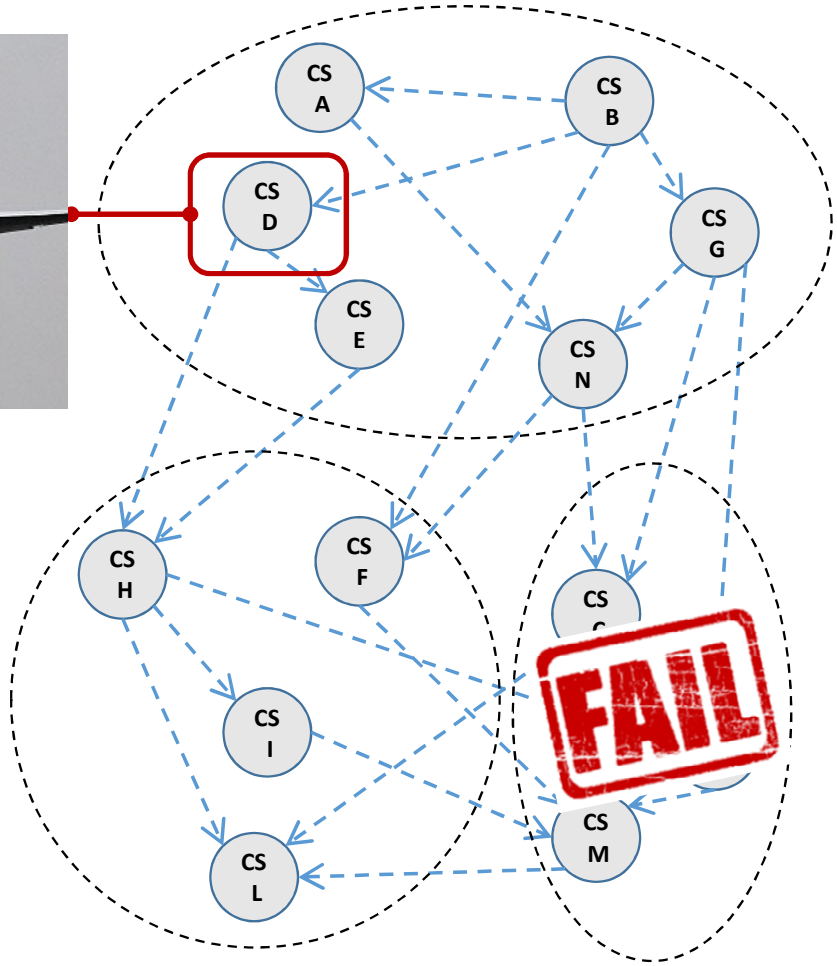
Safe State – Example 2: Transaction Rollback (2/2)



Safe State: All databases are correct and consistent

Most applications = Systems-of-systems (SoS)

http://www.flight.org



Constituent system failure

Partial Fail-Safe State

R6

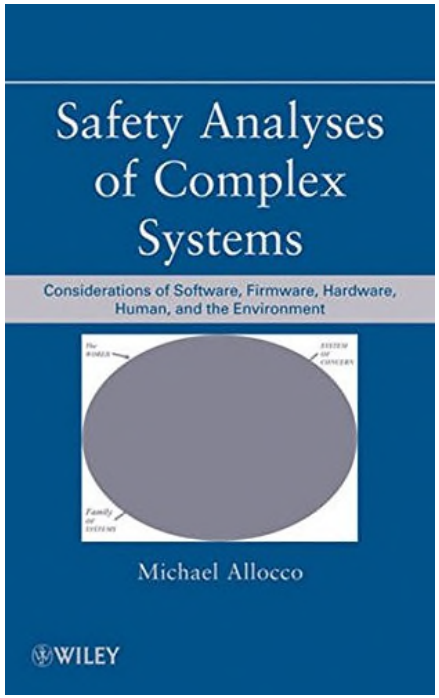
Resilience Architecture Principle R6:

Fail-Safe States

1. Execute a careful hazard analysis of your *full system* to identify all (= goal) critical or harmful states
2. Document all paths to the critical or harmful states in a formal way, such as state chart diagrams
3. Model your application (or the software part of it) as a finite state machine
4. Define fail-safe state(s)
5. Implement reliable paths from all nodes to the fail safe state(s)

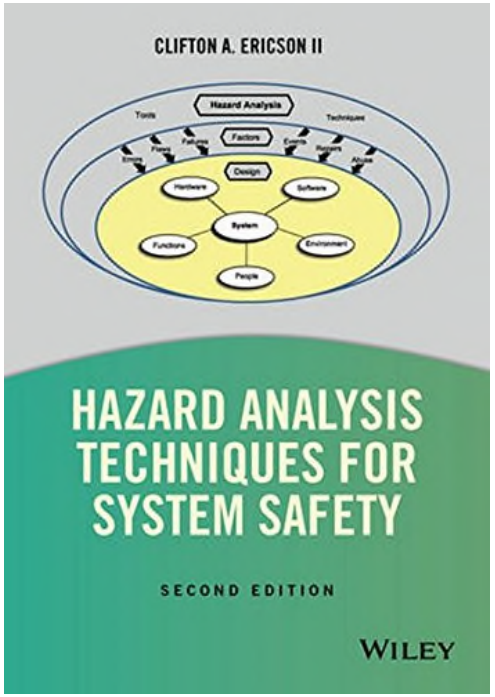
Justification: If a failed system can transition into a fail-save state, then damage, loss of life or property or other negative consequences may be avoidable (or minimized)

Textbook



Michael Allocco:
Safety Analyses of Complex Systems:
Considerations of Software, Firmware, Hardware, Human, and the Environment
John Wiley & Sons, USA, 2010. ISBN 978-0-470-58770-6

Textbook



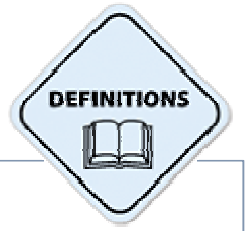
Clifton A. Ericson
Hazard Analysis Techniques for System Safety
John Wiley & Sons, USA, 2nd edition, 2015. ISBN 978-1-118-94038-9

General Resilience

Architecture Principles

Graceful Degradation

Degraded Mode of Operation:
Drive slowly and carefully



Graceful Degradation:

Property of a system to continue operation at some reduced level of **functionality**, **performance** or **dependability** after one or several of its components failed

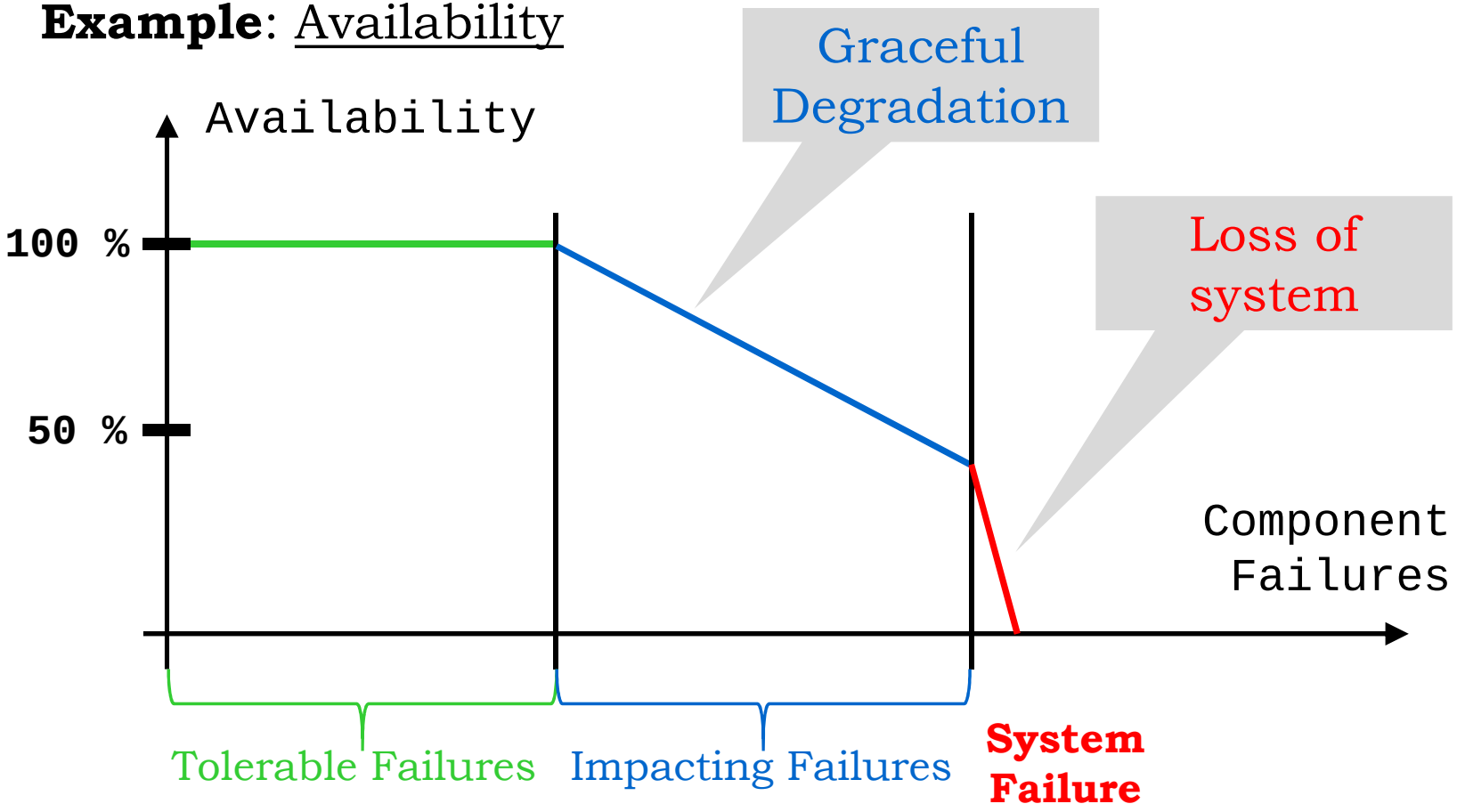
Example: Degraded airworthiness after engine failure

http://www.flight.org

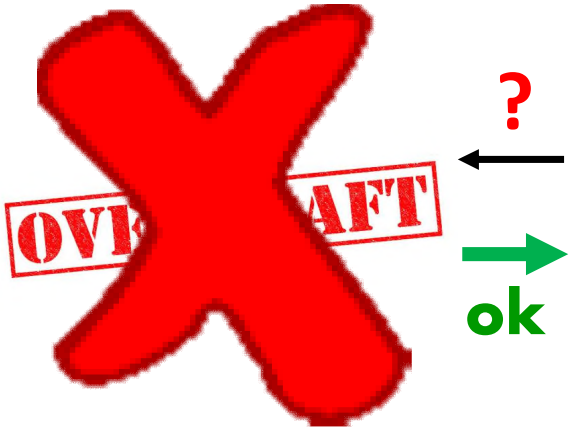
- Stop right engine
- Engine fire-extinguisher kills the fire
- **Degraded operation**: Fly with one engine
- **Safe-State**: Safe landing at nearest airport

Graceful Degradation = Specific to Resilience Properties

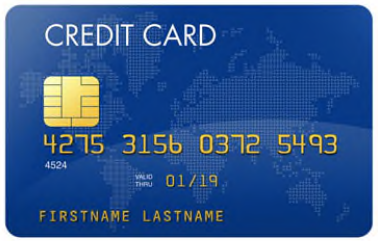
Example: Availability



Example: Graceful Degradation in Automatic Teller Machines



http://www.vendingroutesforsale.net

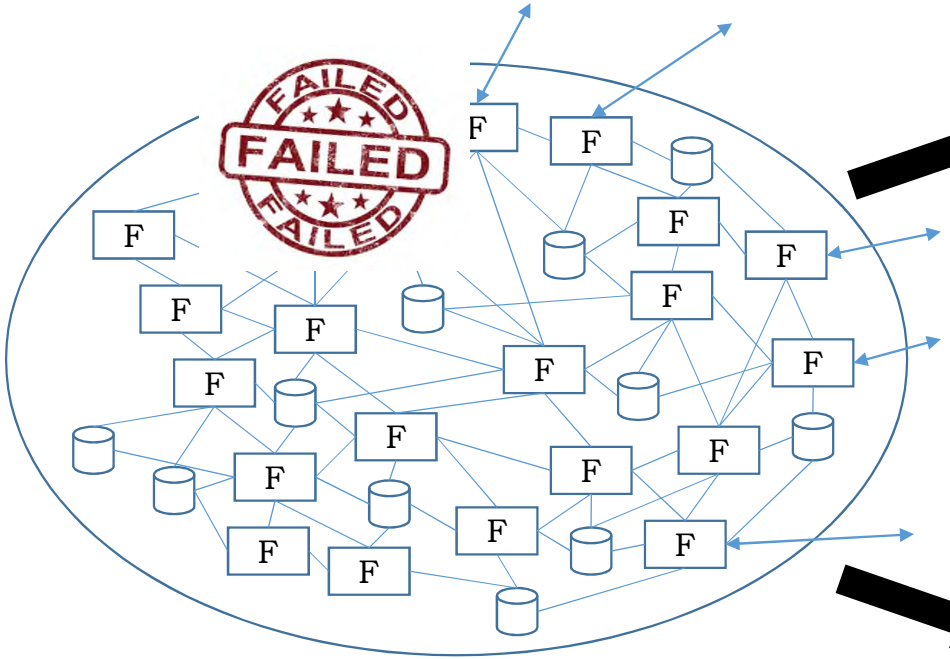


https://github.com

Daily withdrawals stored on the card

Dependability property impacted: **Security**

Degraded Operation: Limit cash dispensing to 100 € per day



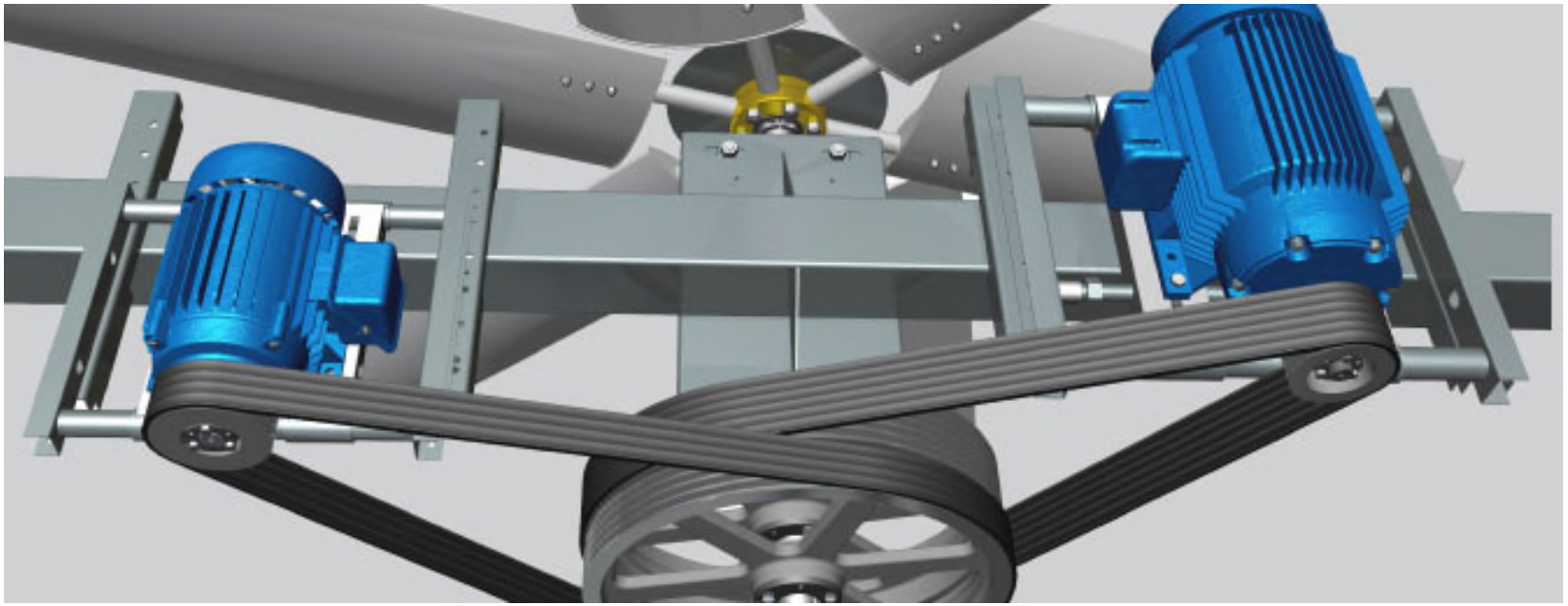
reduced level of **functionality** or **dependability**

Fault Tolerance: succesful **redundancy**

Graceful Degradation = Fault Tolerance Engineering

Graceful Degradation = Fault Tolerance Engineering

Fault tolerance: Providing functionality or service that are consistent with its specification in spite of faults



⇒ Addition of planned **redundancy** to our systems

https://www.baltimoreaircoil.com

R7

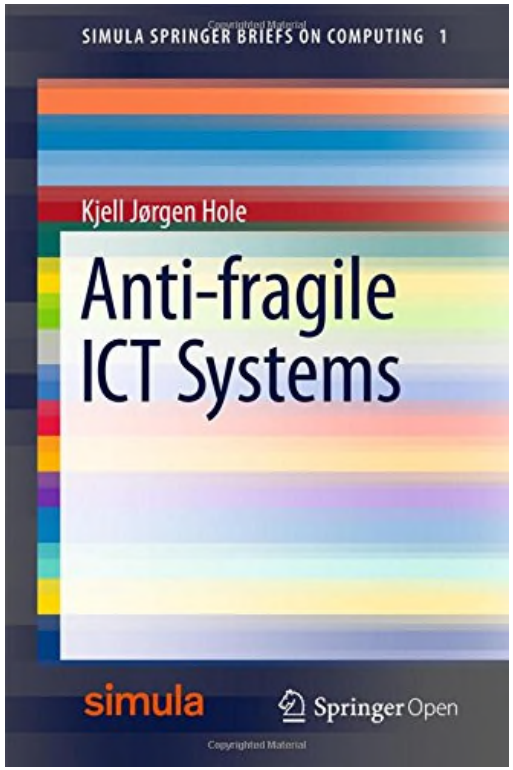
Resilience Architecture Principle R7:

Graceful Degradation

1. Investigate the possibility for graceful degradation in your planned system (= Business task)
2. Architect and implement proven graceful degradation technologies (for specific resilience properties, such as availability, performance, safety, security, …)
3. Compensate component failures by carefully planned *redundancy*

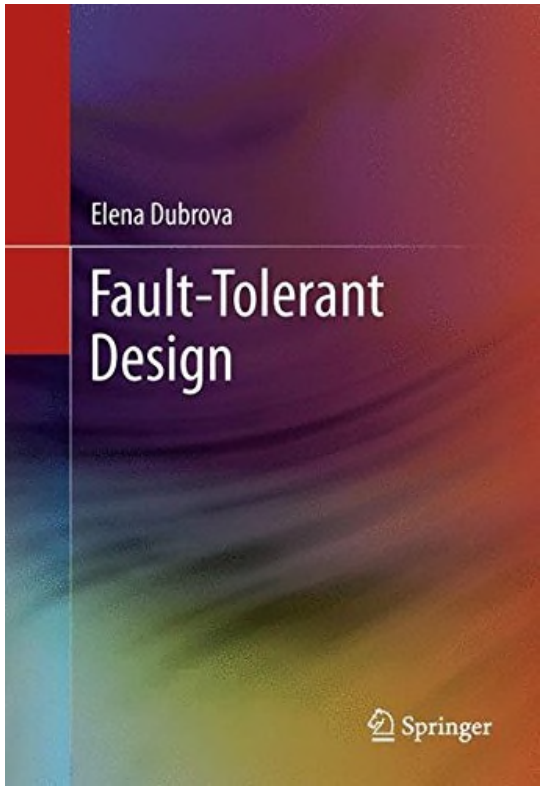
Justification: The value of many systems is significantly improved if after a failure of a component the system operates in a (planned) degraded mode instead of stopping service

Textbook



Kjell Jørgen Hole:
Anti-fragile ICT Systems
Springer-Verlag, Germany, 2016. ISBN 978-3-319-30068-9

Textbook



Elena Dubrova:
Fault-Tolerant Design
Springer-Verlag, Germany, 2013. ISBN 978-1-461-42112-2

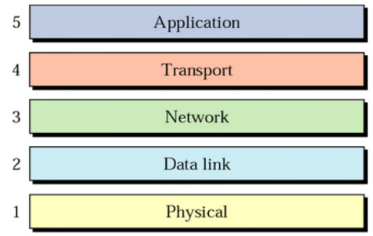
General Resilience

Architecture Principles

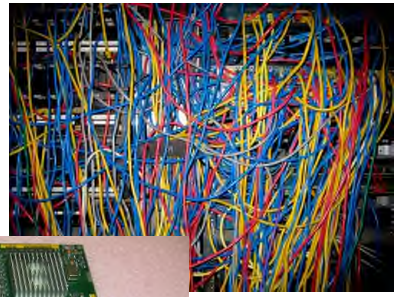
Dependable Foundation (Infrastructure)

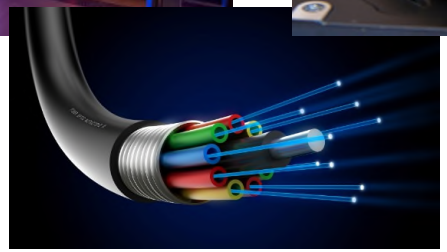
APPLICATIONS

Execution Infrastructure: *System Software*



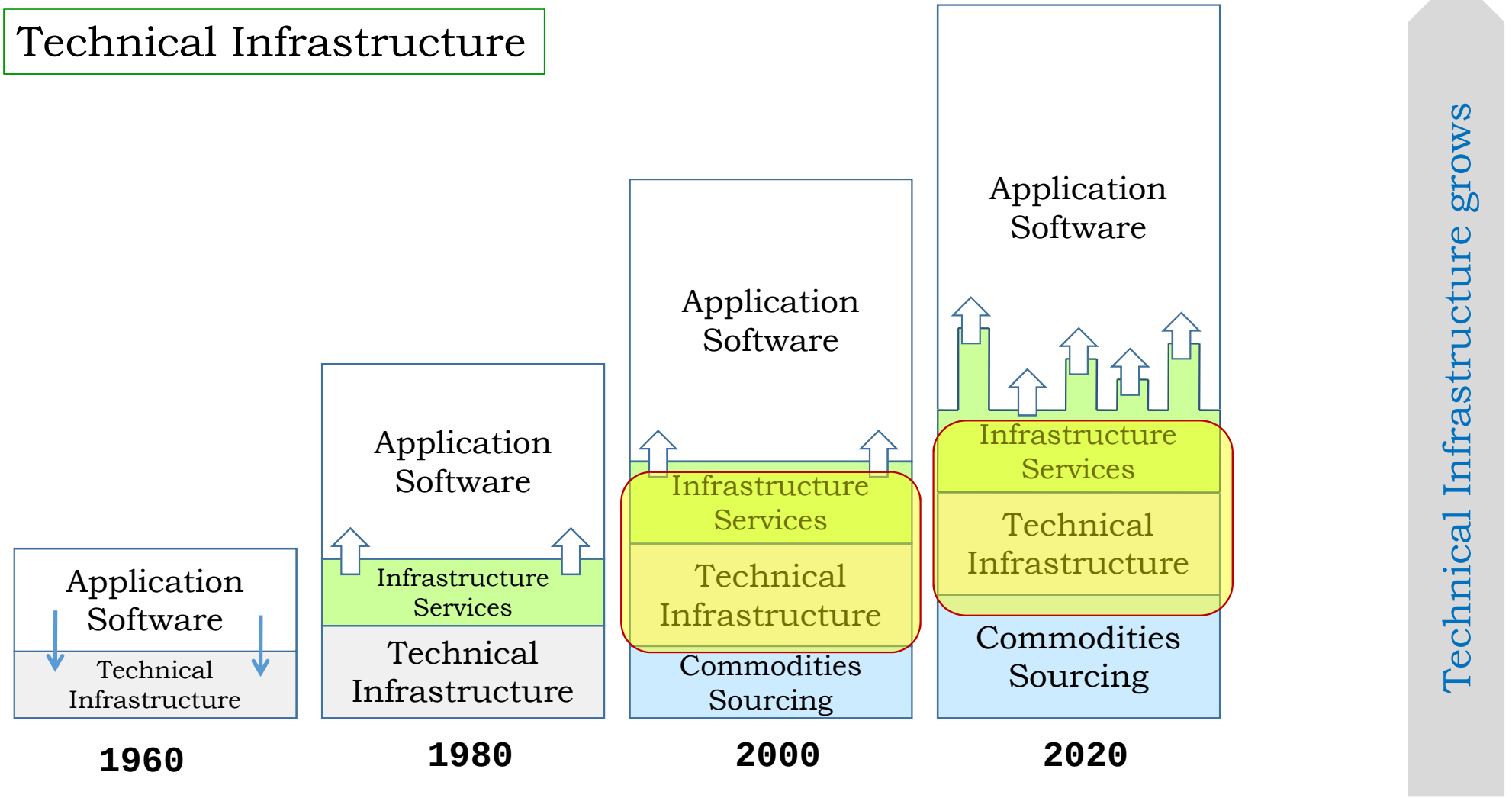
Execution Infrastructure: *System Hardware*



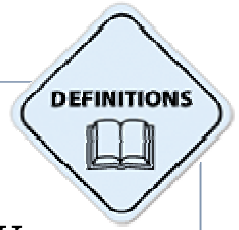


System Stack

Technical Infrastructure



Use a *resilience infrastructure* as part of a reliable foundation for dependable software-systems



Resilience Infrastructure:

Set of proven resilience technologies and services supporting the dependability properties (availability, security, performance, …) of software systems

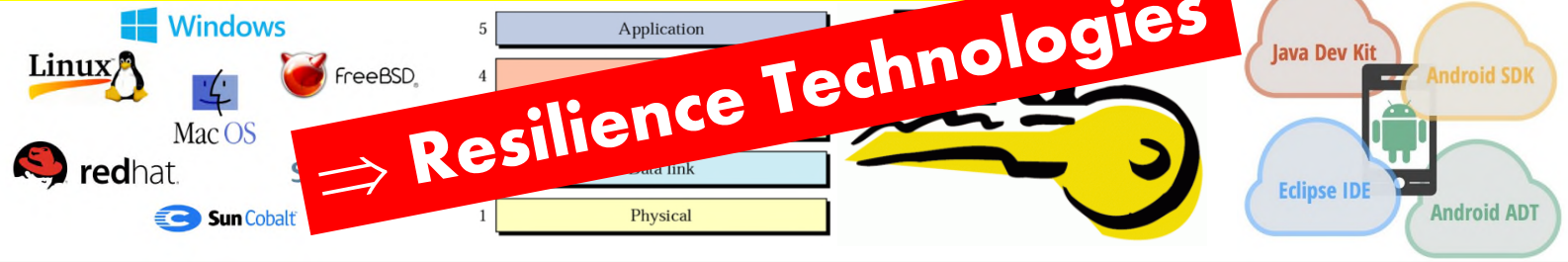
Note 1: Remember «Industry Standards» ⇐ Do NOT get boxed in by vendor-specific features

Note 2: Only use proven technology and isolate it via services

APPLICATIONS

⇒ Resilience Principles

Execution Infrastructure: *System Software*



⇒ Resilience Technologies

Execution Infrastructure: *System Hardware*

⇒ Resilience Technologies

https://www.npmjs.com

Dependability Engineer Roles:

- Security Engineer
- Safety Engineer
- Availability/Performance Engineer
- … (all required resilience properties)

Architecting a Resilience Infrastructure:

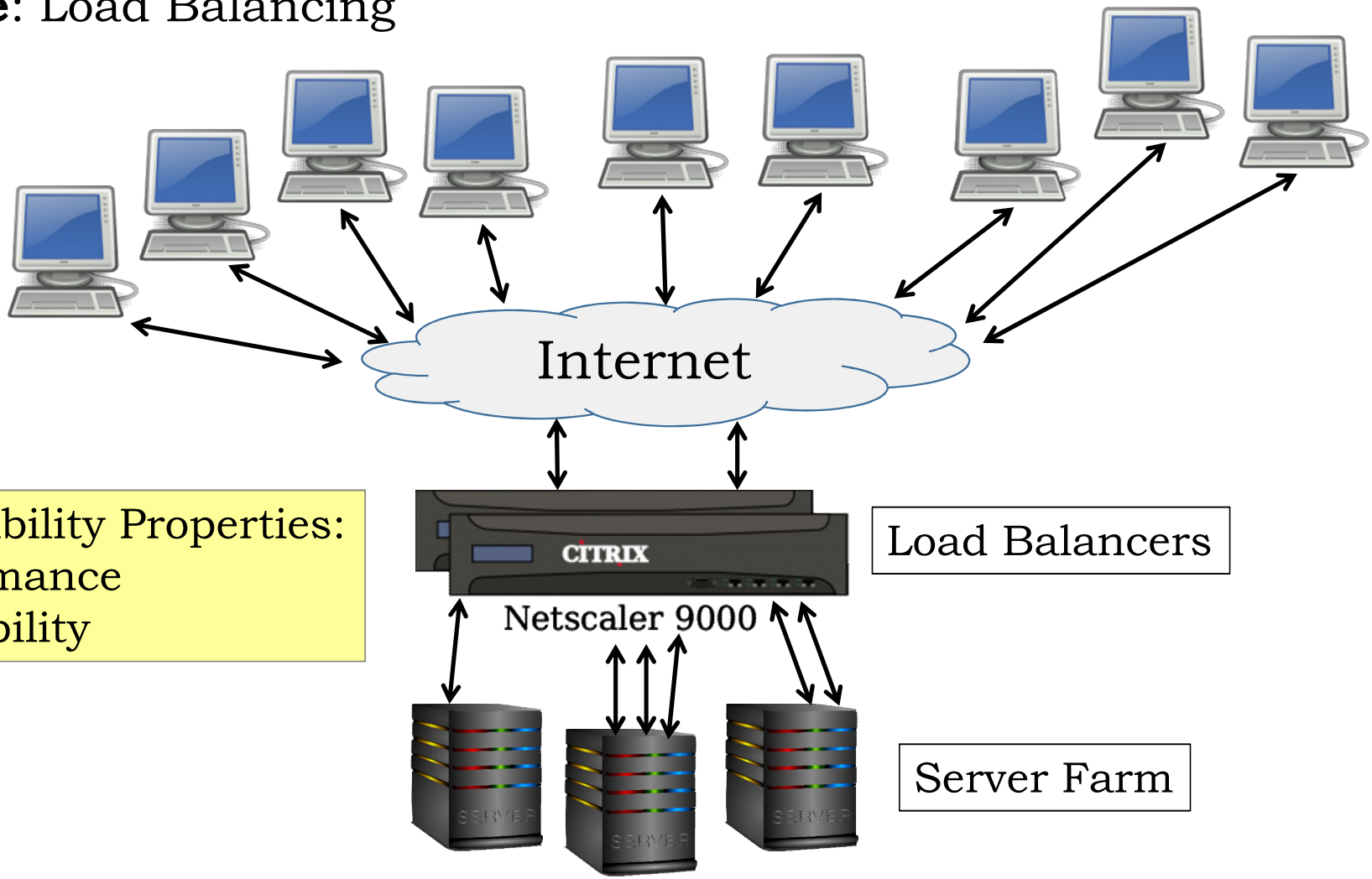
✓ Defining the specifications

✓ Selecting the vendors

✓ Integrating into the existing company/product infrastructure

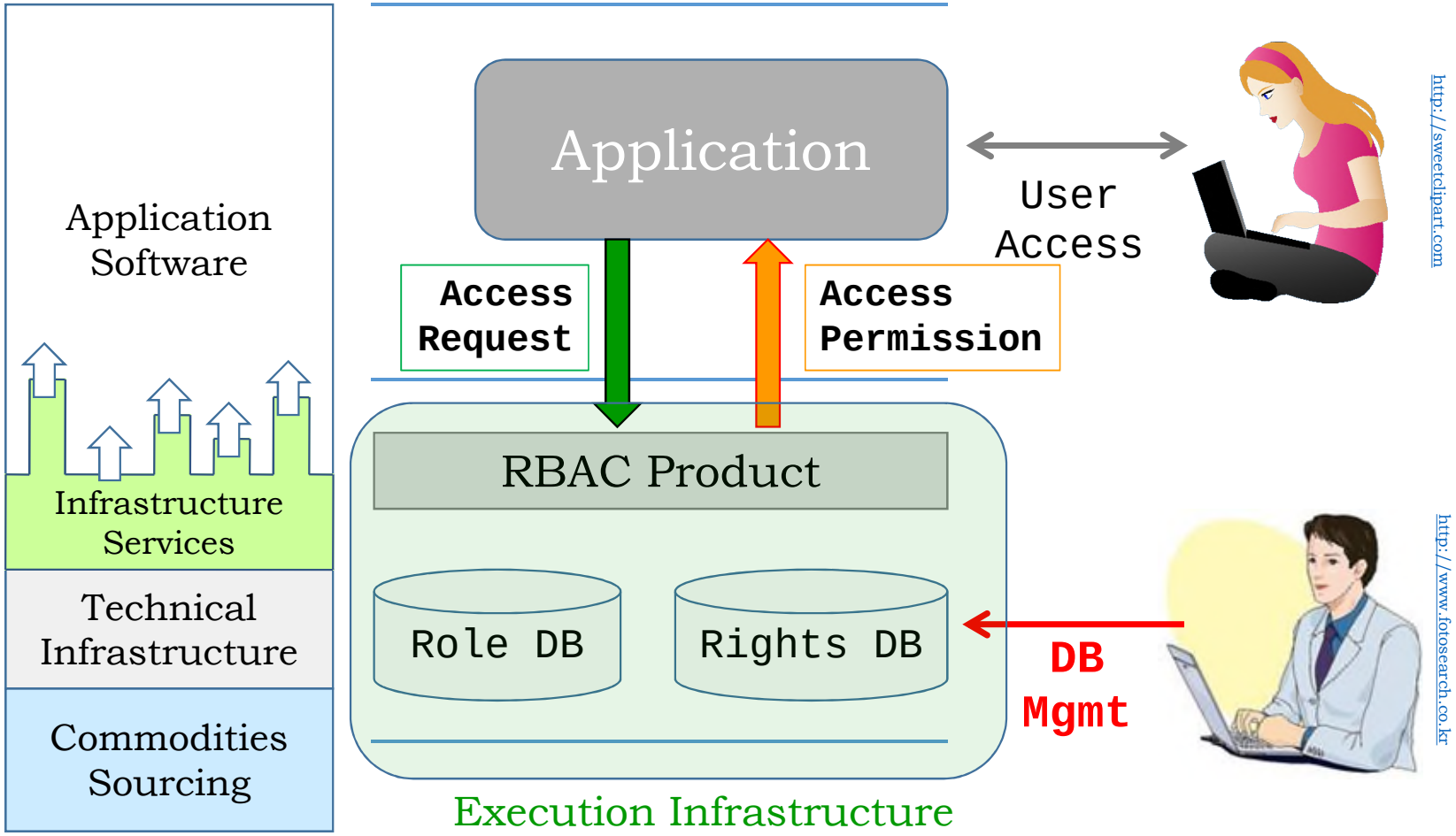
✓ Maintenance (Updates, Patches)

Example: Load Balancing



Dependability Properties:
- Performance
- Availability

Example: Role-Based Access Control (RBAC)



R8

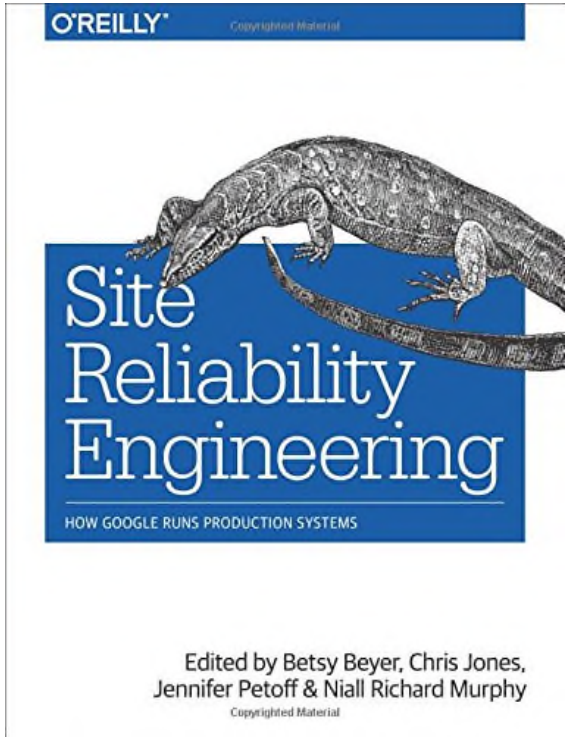
Resilience Architecture Principle R8:

Dependable Foundation (Infrastructure)

1. Use a resilience infrastructure as part of a dependable foundation for resilient software-systems
2. Only use *proven* resilience technologies and services supporting the resilience properties (availability, security, performance, …)
3. Whenever possible use industry-standard based resilience techniques (Avoid vendor lock-in)

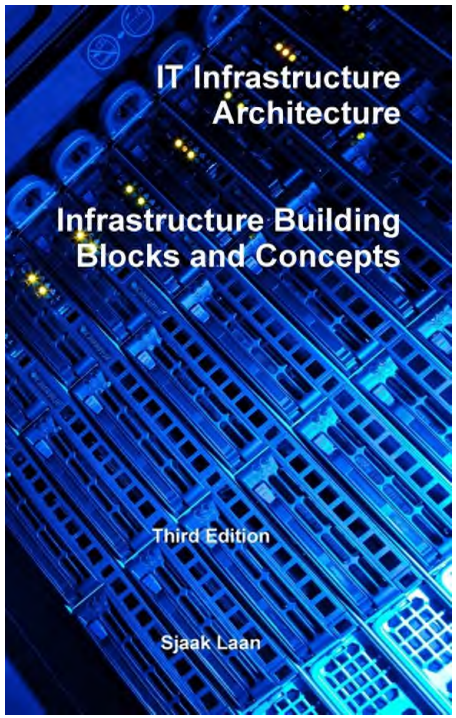
Justification: An implementation of proven resilience techniques in the form of industry-standard products forms a valuable, trustable resilience foundation

Textbook



Betsy Beyer, Chris Jones, Jennifer Petoff, Niall Richard Murphy:
Site Reliability Engineering: *How Google Runs Production Systems*
O'Reilly UK Ltd., 2016. ISBN 978-1-491-92912-4

Textbook



Sjaak Laan:
IT Infrastructure Architecture - *Infrastructure Building Blocks and Concepts*
Lulu.com, 3rd edition, 2017. ISBN 978-1-326-91297-0

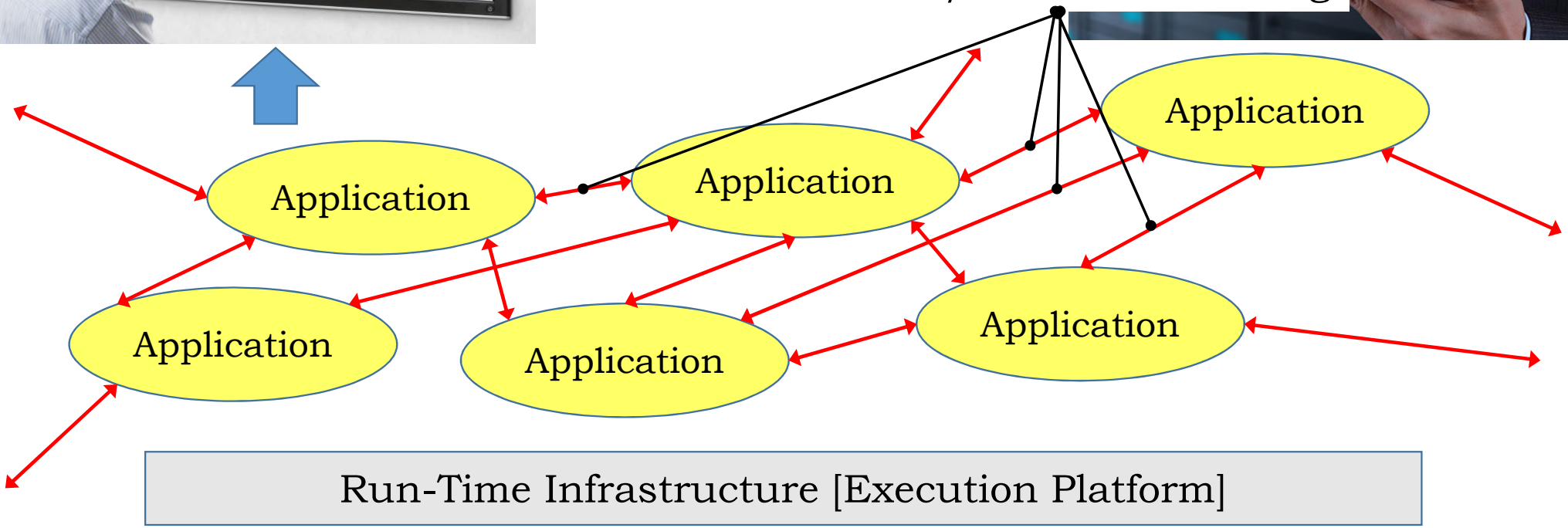
General Resilience

Architecture Principles

Operating Parameter Monitoring

Monitoring

Interface/Service Monitoring

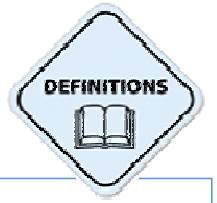


Run-Time Infrastructure [Execution Platform]

http://www.mcleanit.ca

Operating Parameter Monitoring

Monitoring



An **IT system monitor** is a hardware and software component used to measure resource consumption and performance in a computer system.

https://raw.githubusercontent.com

Any ***anomaly*** in operating parameters (load, response time, …) is automatically detected and an alarm is triggered

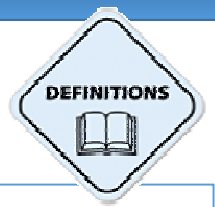
⇒ **Automatic** or **human** intervention

http://www.digicor.com.au

Interface/Service Monitoring

https://www.nethosting.com

Monitoring



Interface/service monitoring is a continuous, real-time activity which assures that:

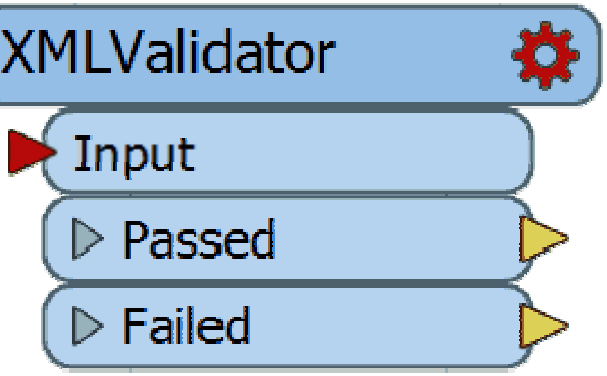
- An application accepts only syntactically and semantically correct information,
- The interface/service contract conditions are met by the delivering party,

The receiving application cannot be crashed, damaged or mislead by accidental or malicious information.

http://www.digicor.com.au

Interface/Service Monitoring

Syntax/Semantic Checking



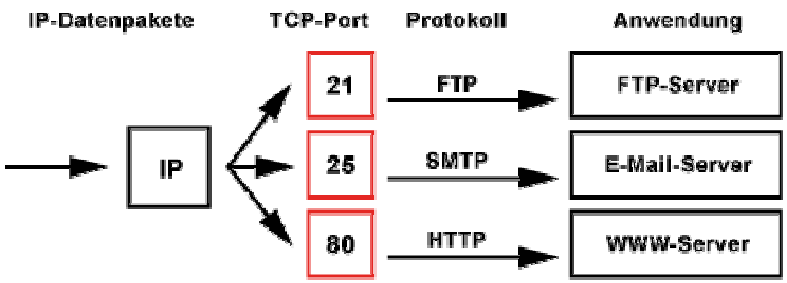
https://cdn.safe.com

Monitoring

Content Filtering

Malware Checker

http://www.psdgraphics.com



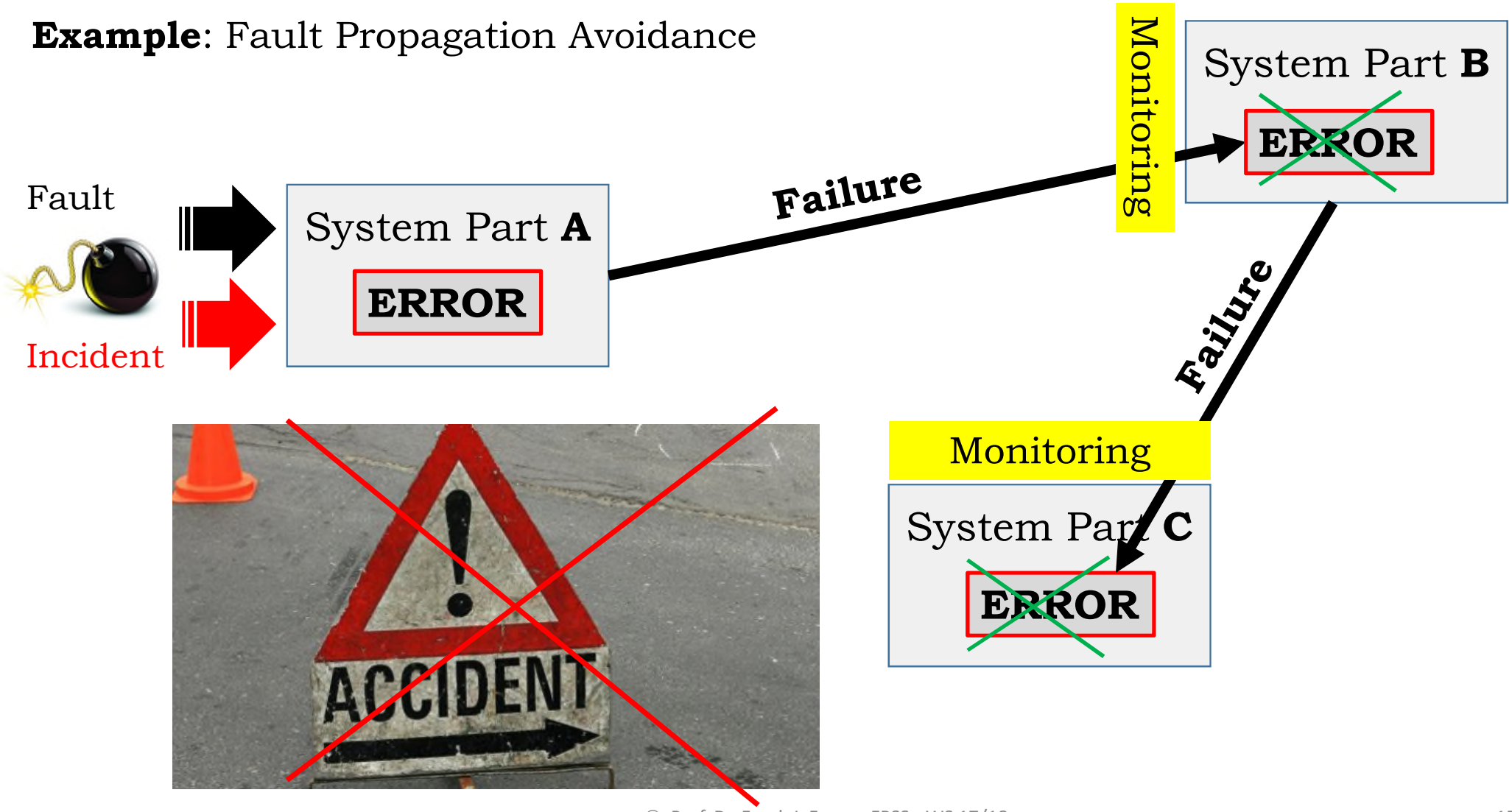
Entry/Port Checking

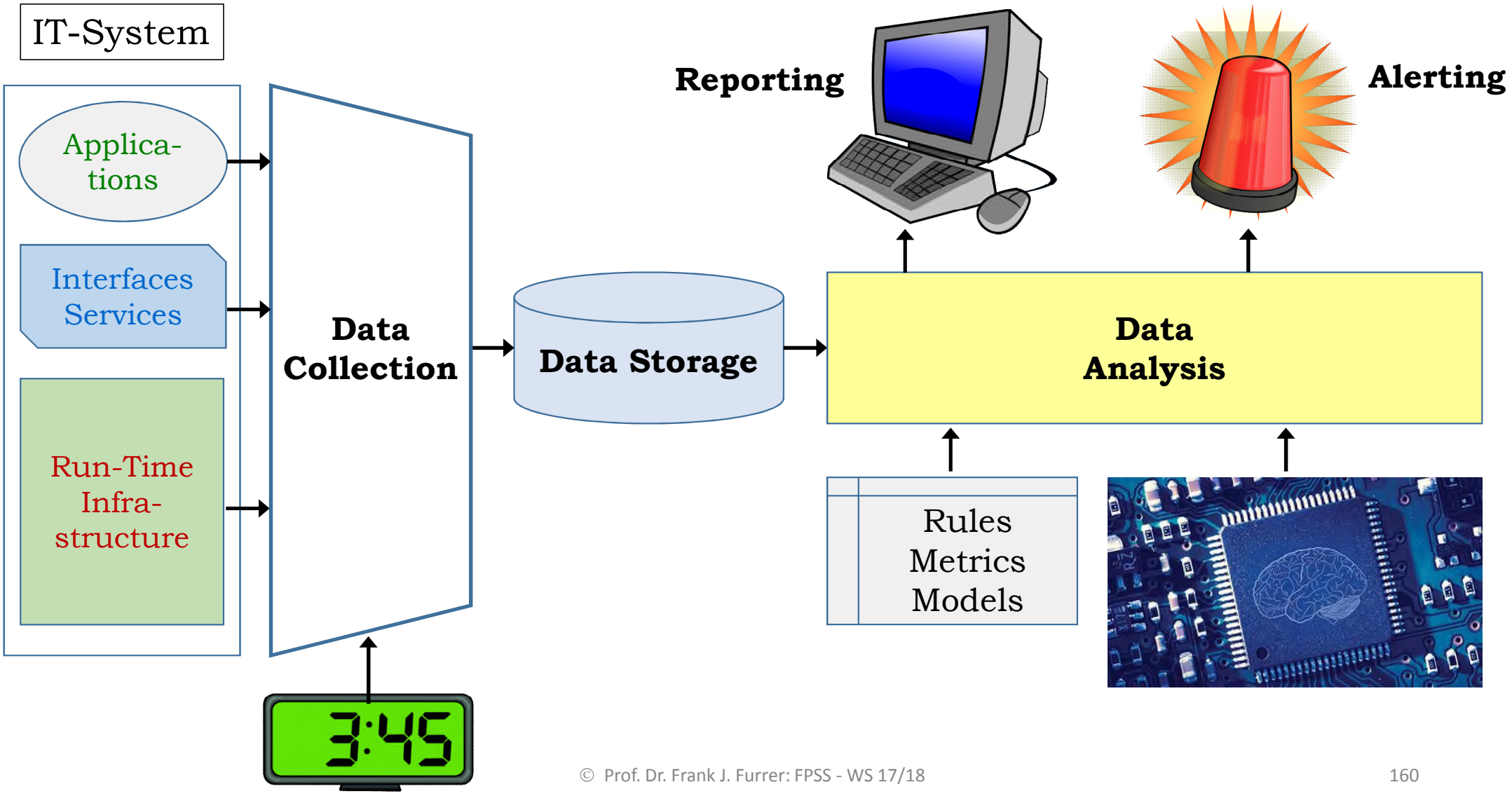
https://www.elektronik-kompendium.de

Standards Conformance Checking

https://www.sepastop.eu

Example: Fault Propagation Avoidance

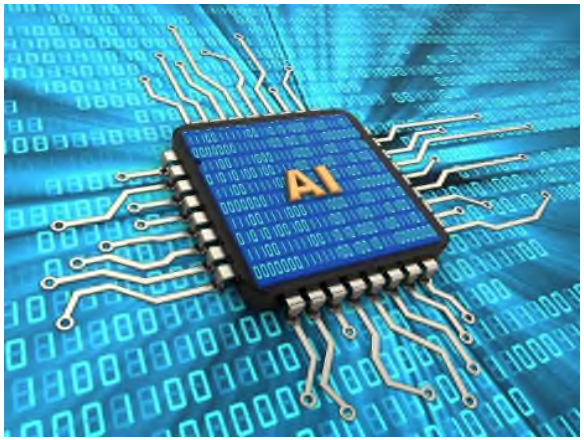



IT-System
Applica-tions
Interfaces Services
Run-Time Infra-structure
Data Collection
Data Storage
Data Analysis
Reporting
Alerting
Rules
Metrics
Models
3:45

Monitoring

Alerting

Human Intervention



Automatic Intervention

What should be monitored ?

https://thumbs.dreamstime.com

Technical Information

- Network: Operational Parameters
- Infrastructure: Operational Parameters
- Interfaces/Services: Timing, Syntax & Semantics
- Configuration: Changes

Business Information

- Business KPI's: Statistics
- Applications: Operational Parameters
- Service Level Agreements: Operational Parameters
- Dependability Properties: Activity & Parameters

What should be monitored ?

Technical Information

Objectives:

- Early problem warning
- System defense
- System optimization
- System intelligence information
- Failure tracing

⇒ *Assure the **non-functional** requirements*

https://www.istockphoto.com

Business Information

Objectives:

- Customer satisfaction
- Financial optimization
- Contract (SLA) supervision
- Audit/Compliance
- Business intelligence information
- ⇒ Assure the ***functional*** requirements

https://www.jobdiagnosis.com

R9

Resilience Architecture Principle R9:

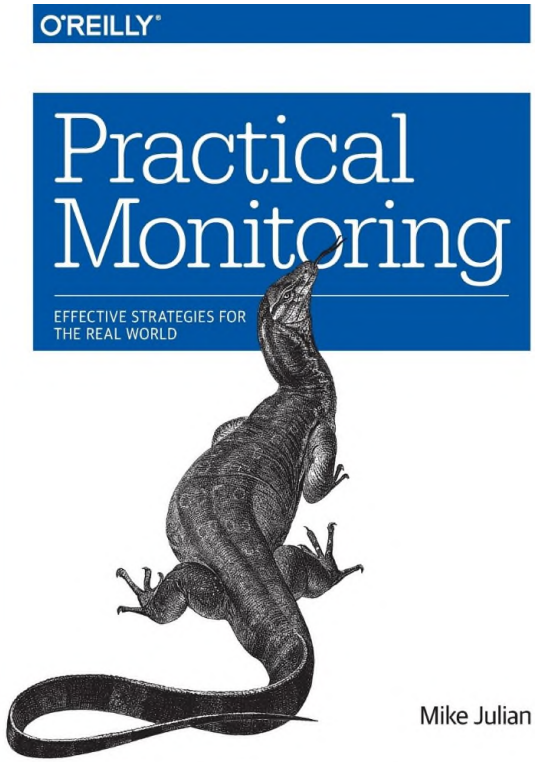
Monitoring

1. Define the objectives of monitoring, both for technical monitoring and the business monitoring
2. Carefully specify the metrics, analytics, results, and alerts to be extracted from monitoring
3. Define the processes for data analysis, including incident/emergency response
4. Specify the actions following alerts – whenever fully automated responses
5. Recommendation: Use commercial monitoring tools whenever possible

Justification:

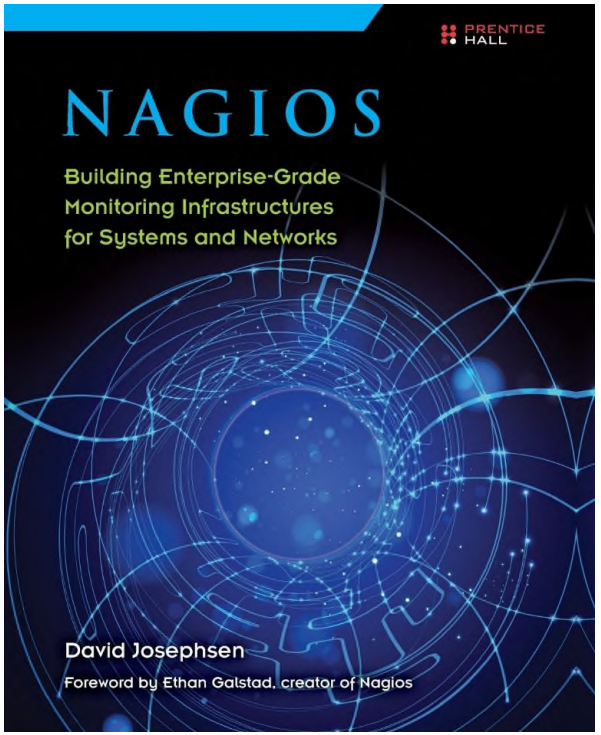
- Technical monitoring is a strong weapon for assuring the non-functional properties of the system and for defending the system against incidents
- Business monitoring strongly contributes to customer satisfaction

Textbook

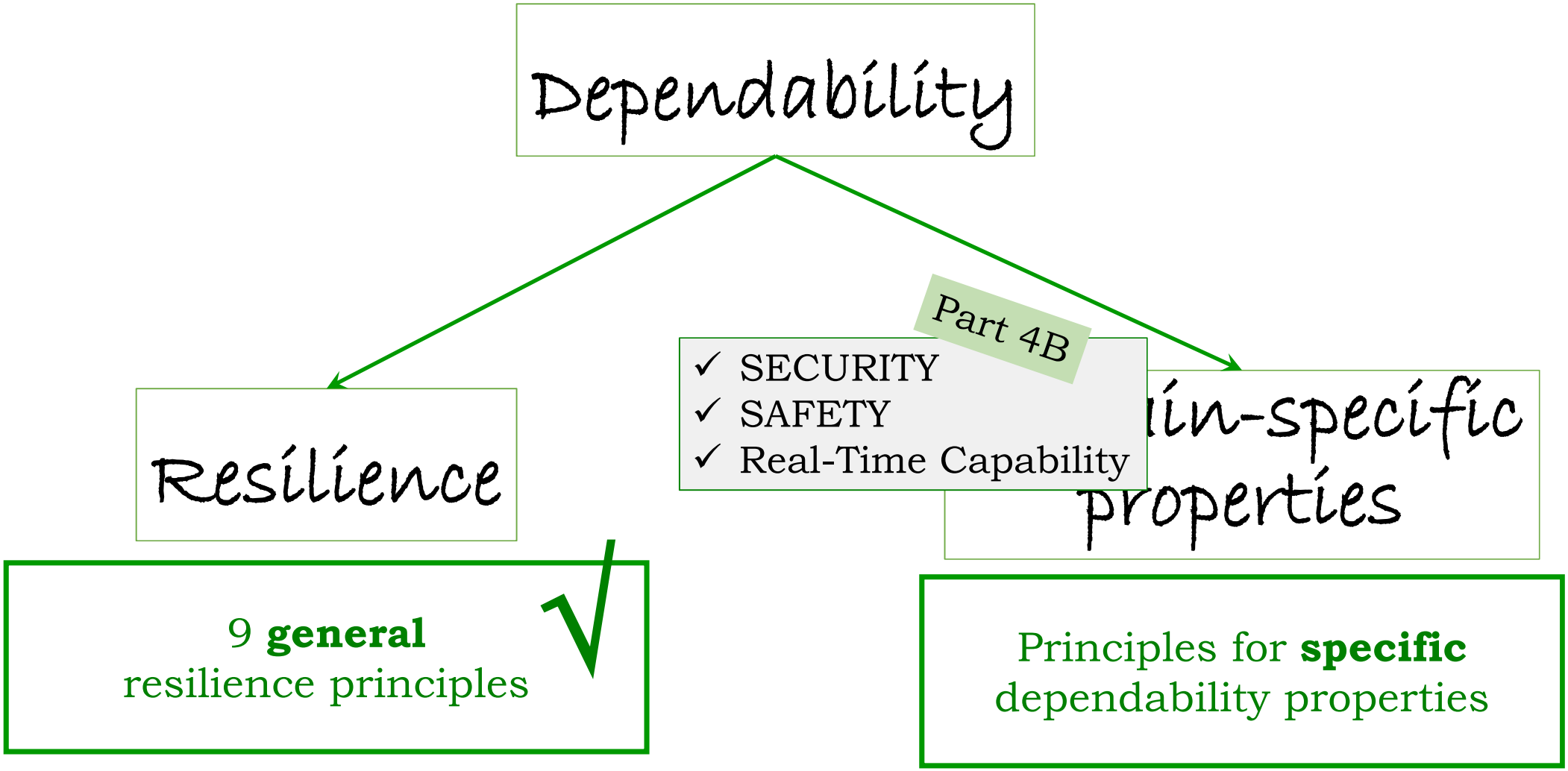


Mike Julian:
Practical Monitoring: *Effective Strategies for the Real World*
O'Reilly UK Ltd., 2017. ISBN 978-1-491-95735-6

Textbook



David Josephsen:
Nagios: *Building Enterprise-Grade Monitoring Infrastructures for Systems and Networks*
Prentice Hall Inc., USA, 2nd edition, 2013. ISBN 978-0-133-13573-2


Dependability
Resilience
Part 4B
✓ SECURITY
✓ SAFETY
✓ Real-Time Capability
ain-specific properties
9 general resilience principles
√
Principles for specific dependability properties

Part 4A