https://blog.thebrain.com

Key Points FPSS
Lecture: 06.01.2016

January 6, 2016:

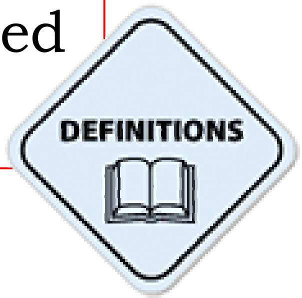
- Legacy System Migration/Modernization
- Software Product Lines
- Domain Software Engineering

- Part 4A: Architecting for Resilience

  General Resilience Architecture Principles

Legacy Systems Migration/Modernization

A *legacy system* refers to outdated computer systems, eroded software architectures, old programming languages or badly supported applications software



A very large part of systems in today's use are legacy systems!

http://www.123rf.com

http://www.nzz.ch/aktuell

Liability — Asset

bad:

- very low agility
- weak resilience
- eroded architecture
- badly or not documented
- obsolete technology (HW & SW)
- large technical debt
- lost knowledge (people left)

good:

- invaluable *implicit* knowledge of the domain and the business processes
- stable operation (mature)
- good solutions/algorithms
- often: suprisingly good code

Legacy systems must (at some time) be replaced or migrated

- High resistance to change for new business requirements (low agility)
- Weak resilience (attacks, faults, ...)
- Technology pressure
- New architecture paradigms
- Knowledge shortage

Legacy Software-System

How do we replace or migrate legacy systems?

⇒ By applying a **migration strategy**

Type of Migration	Current State	Target State
Replacing ▪ Completely new development starting from systems requirements	Operational software. Cost, time and risk for a migration to high	Software has completely been rewritten, starting from the initial requirements
Re-Architecting ▪ Transforming to new architecture paradigm (Considerable functional change)	Operational software. Architecture paradigm has changed [e.g. monolithic architecture ⇒ service-oriented architecture]	Software runs under the new architecture paradigm
Re-Engineering ▪ Transforming to new technology base, e.g. new infrastructure or software technology (limited functional change)	Operational code running on an outdated execution platform or using an obsolete software technology	Code runs on the modern execution platform or uses modern software technology
Re-Factoring ▪ Improving existing code (no functionality change)	Operational code, deficiencies in the program implementation	Improved code (quality criteria)
Reverse Engineering ▪ No or insufficient information (code + doc)	Operational code, massive lack of documentation, of knowledge and of source code	System is sufficiently understood and documented to start migration

Legacy SW-System Modernization

http://www.midcontinent.org/press/press_rivets.html

http://www.dcctrain.com

Our world runs on Billions of legacy lines of code which must be migrated/modernized in the next decades

⇒ Enormous challenge for the SW-community

R Legacy

Architecture Recommendations for Legacy System Modernization

1. Unambigously specify the boundary of the system (Code & Data) to be migrated/modernized
2. Clearly assess the state of the legacy system (code, data, documentation, value)
3. Precisely define the migration/modernization goals (for code & data)
4. Choose a migration/modernization strategy based on risk, fit-for-future, cost & time and quality attributes (e.g. certification or validation etc.)
5. Select optimum tool support [Note: Many excellent tools available, search www]

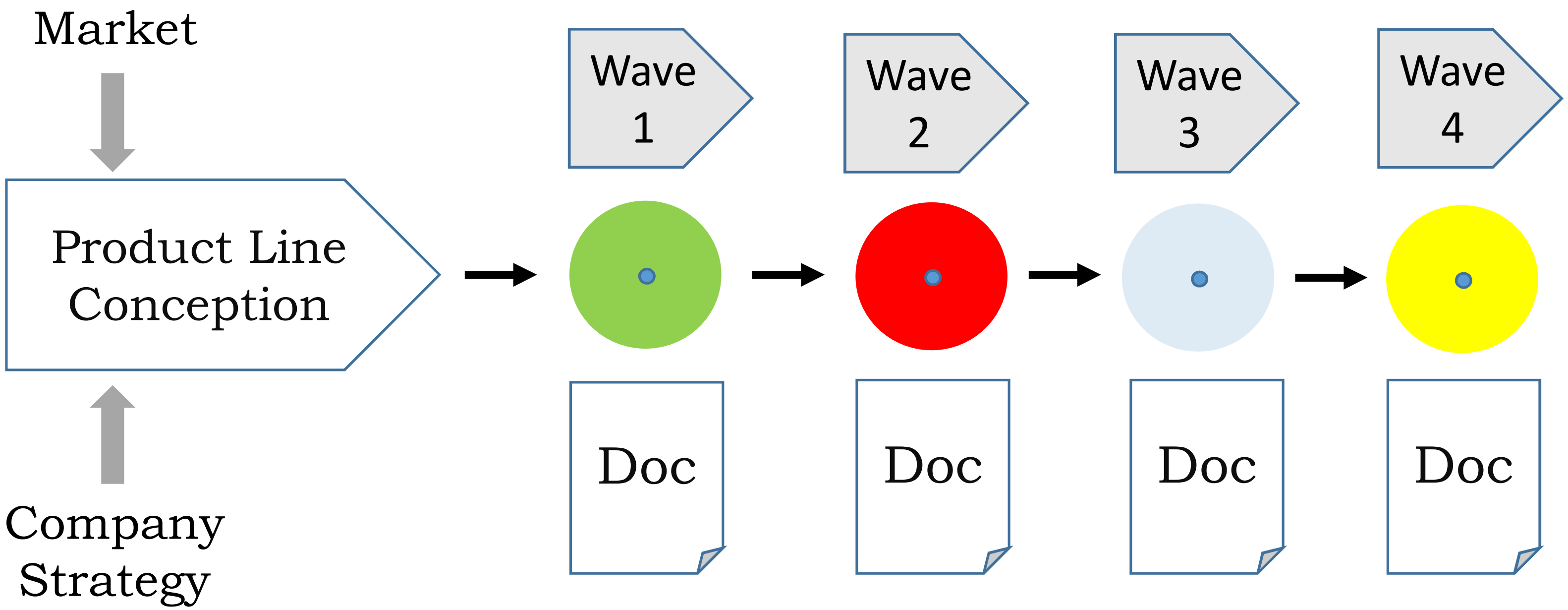
http://www.prodpad.com/2013

Software Product Lines

DEFINITIONS

A software product line is a *set* of software-intensive systems sharing a *common*, managed set of features, that satisfy the specific needs of a particular *market* or *mission* and that are developed from a common set of *core assets*

[Clements02]



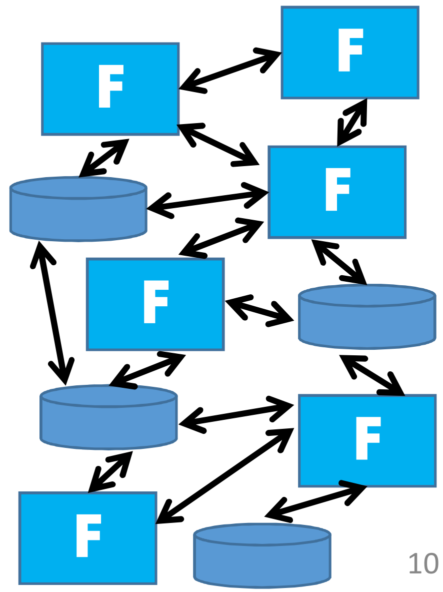
Feature Model

Wave 1:
- X
- Y

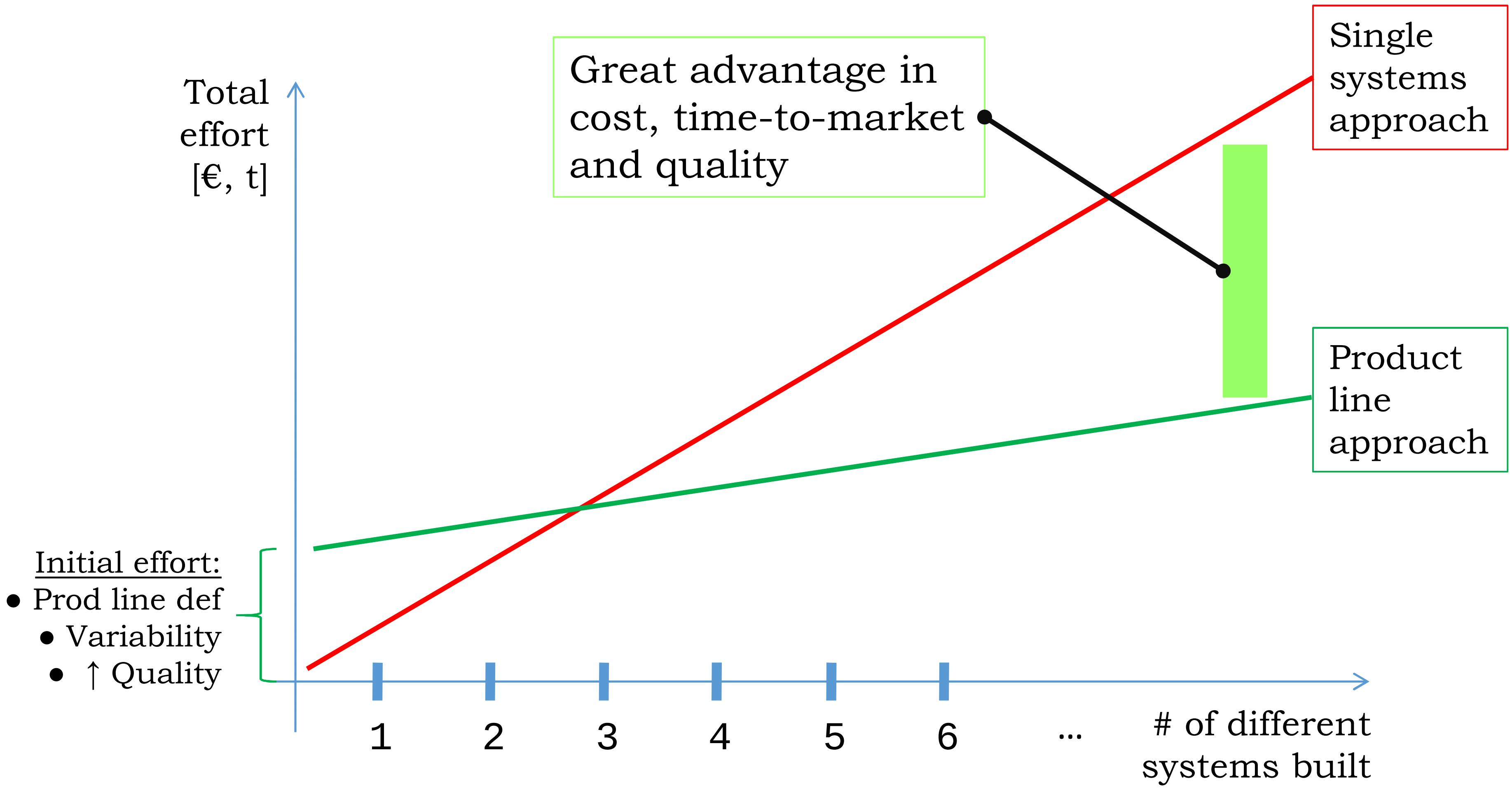
Wave 2:
- Z
- R

…

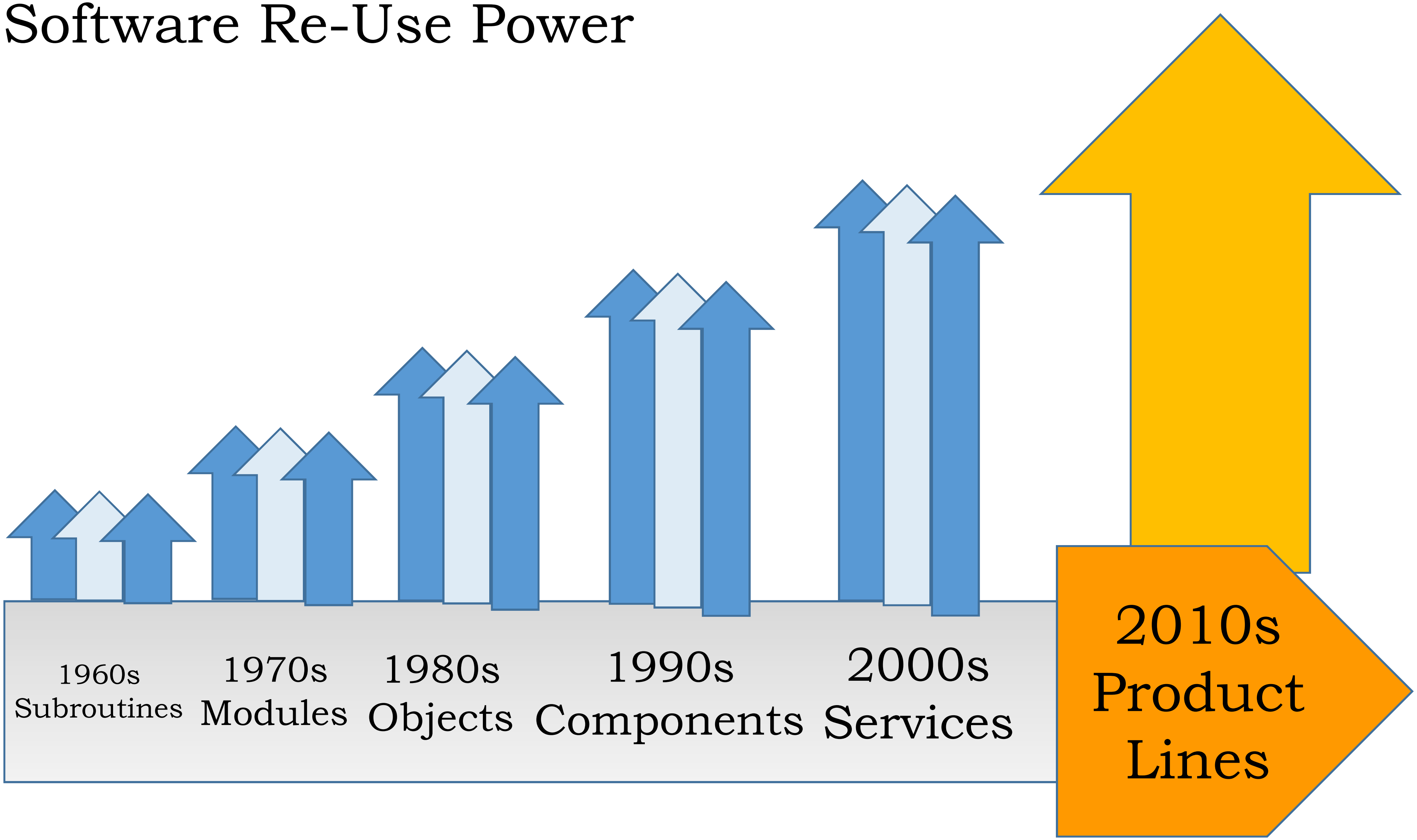
Product Line **Architecture**



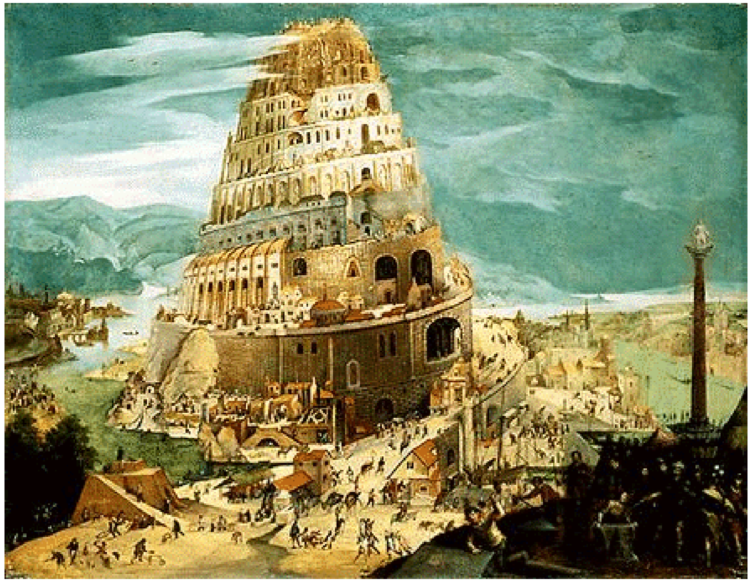
Economics of Product Line Development:



Software Re-Use Power



Linda Northrop, SEI, 2008



http://joseph_berrigan.tripod.com

Domain Software Engineering

Domain Software Engineering [DSE] =

an architectural ***methodology***

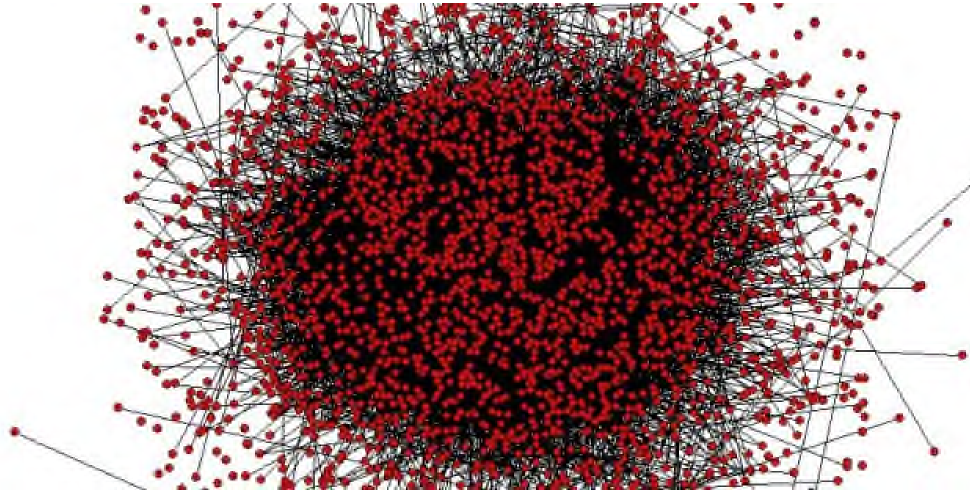
for evolving a *software* system

that closely aligns to *business* domains

DEFINITIONS

The Challenge: Divergence & Complexity

http://www.sutherlandweston.com



http://blog.digital.telefonica.com

Divergence =
Mismatch between
Business Needs
and *IT-Implementation*

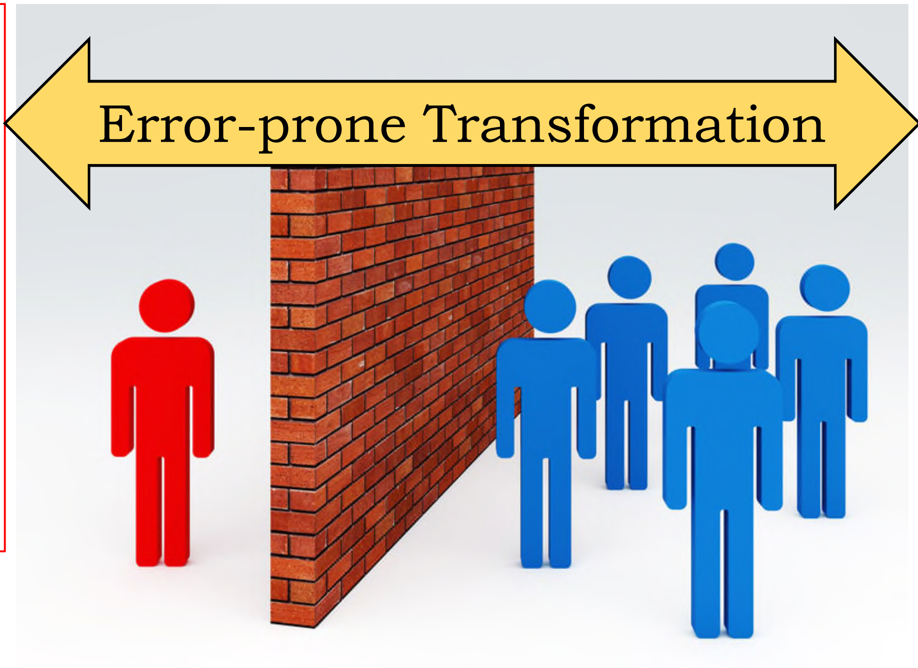
Complexity =
Property of an IT-system which makes it difficult to formulate its overall behaviour, even when given complete information about its parts and their relationships

http://www.sutherlandweston.com

Divergence

Business Units:

- Vocabulary
- Concepts
- Customs
- Knowledge
- Tradition
- …



http://mayrsom.com

IT Units:

- Terminology
- Classes
- Programming
- Constraints
- Experience
- …

http://www.sutherlandweston.com

Business Units

Divergence

IT Units

Business needs
Requirements

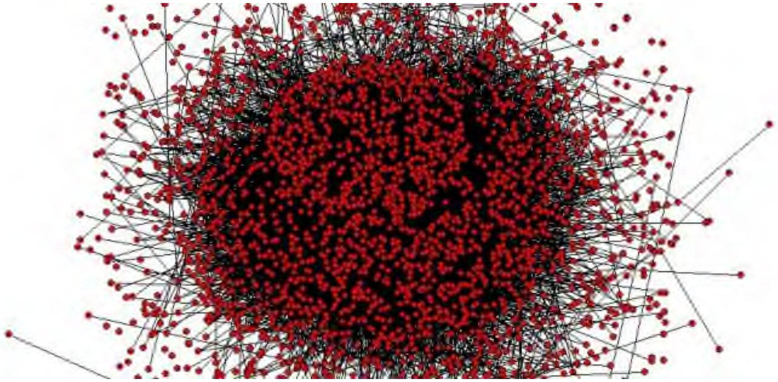
Misunderstandings, Loss of Precision

Specifications

Development

Integration

Deployment



http://blog.digital.telefonica.com

Complexity

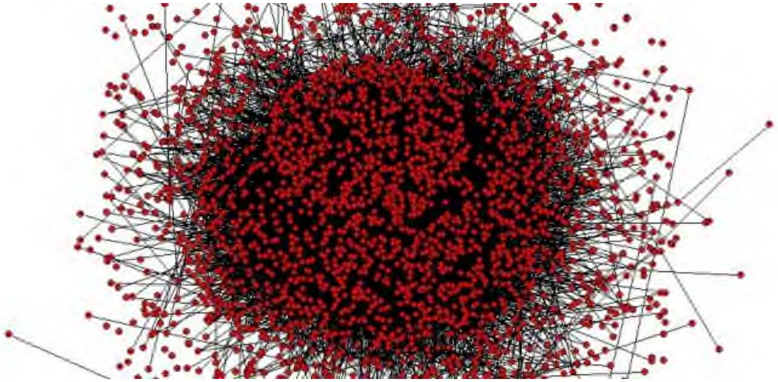
Essential complexity

... is the *inherent* complexity of the system to be built

Essential complexity for a given problem *cannot* be reduced.

Accidental Complexity

... is *introduced* by our development activities or by constraints from our environment



http://blog.digital.telefonica.com

Complexity

Business Units

IT Units

Business needs
Requirements

Translation: Accidental Complexity

Specifications

Development

Integration

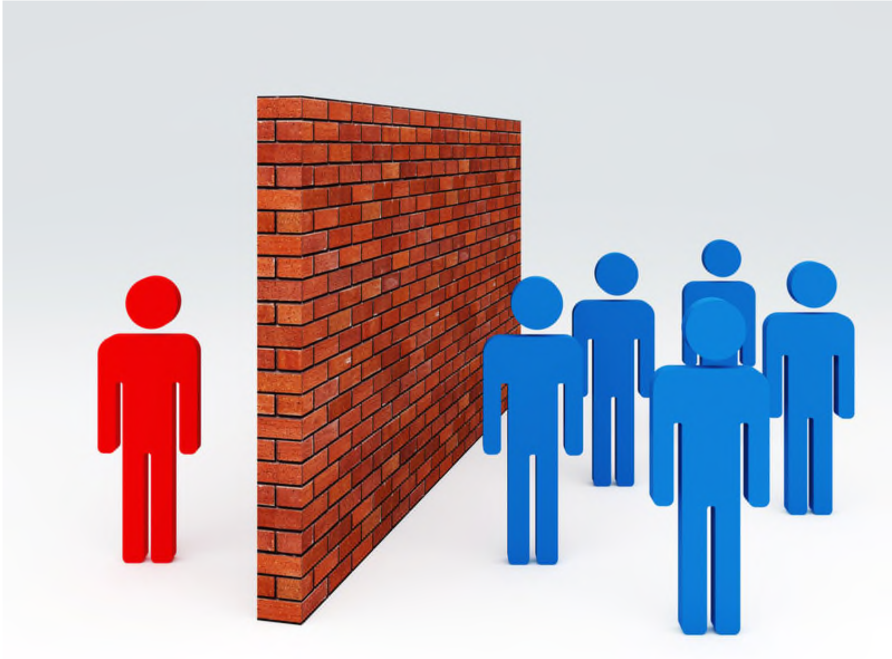
Deployment

Modeling — Accidental Complexity

Implementation — Accidental Complexity

Integration — Accidental Complexity

Business Units



IT Units

We need to improve communication and understanding!

⇒ The promise and value of domain software engineering (DSE)

Future-Proof Software-Systems: Summary

http://clipartzebraz.com

DSE

https://cimx.wordpress.com

Customer/Business Needs

IT Implementation

Which are the key elements of DSE (Domain Software Engineering?)

1. Understanding the Business/Application Domain in terms of the business
   (⇒ Domain Model)
2. Use of an ubiquitous language
   (Business ⇔ IT alignment)
3. Software: Implementation of Business Domain concepts
   (Concepts ⇒ Business objects ⇒ Programm objects)

Universale Ausdrucksform

The DSE concepts:

- Business/Application Domain — A *Domain* is a Sphere of Knowledge, Influence or Activity
- Bounded Context — The *Bounded Context* is the Boundary of a Model
- Domain Model — A *Domain Model* is a representation of the Entities, Relationships and their Properties in a specific Application Domain
- Anticorruption Layer — An *Anti-Corruption Layer* is a method to isolate two domains or systems, allowing systems to be integrated without knowledge of each other

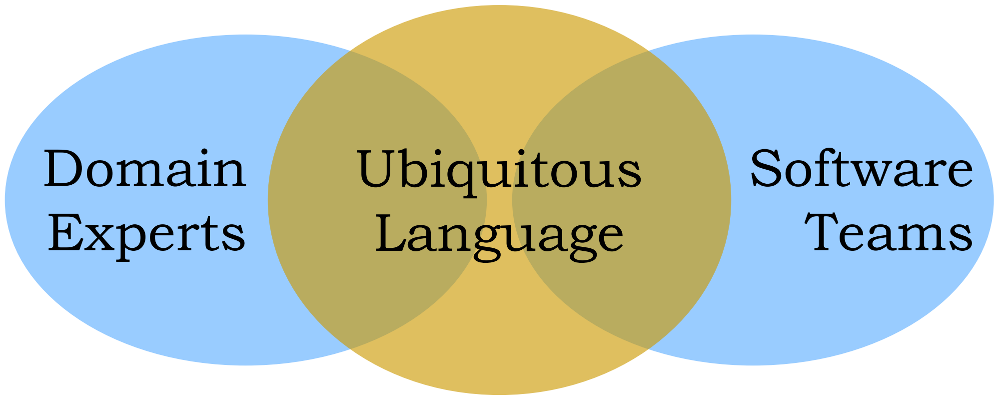
Customer/Business

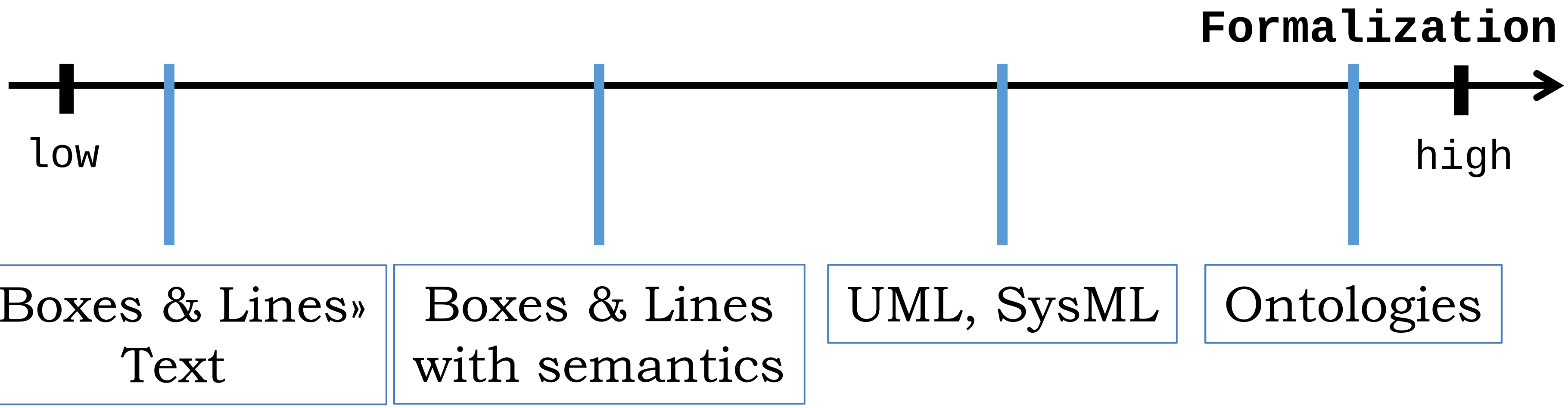
Ubiquitious Language UL

IT Organization

http://clipartzebraz.com

https://cimx.wordpress.com





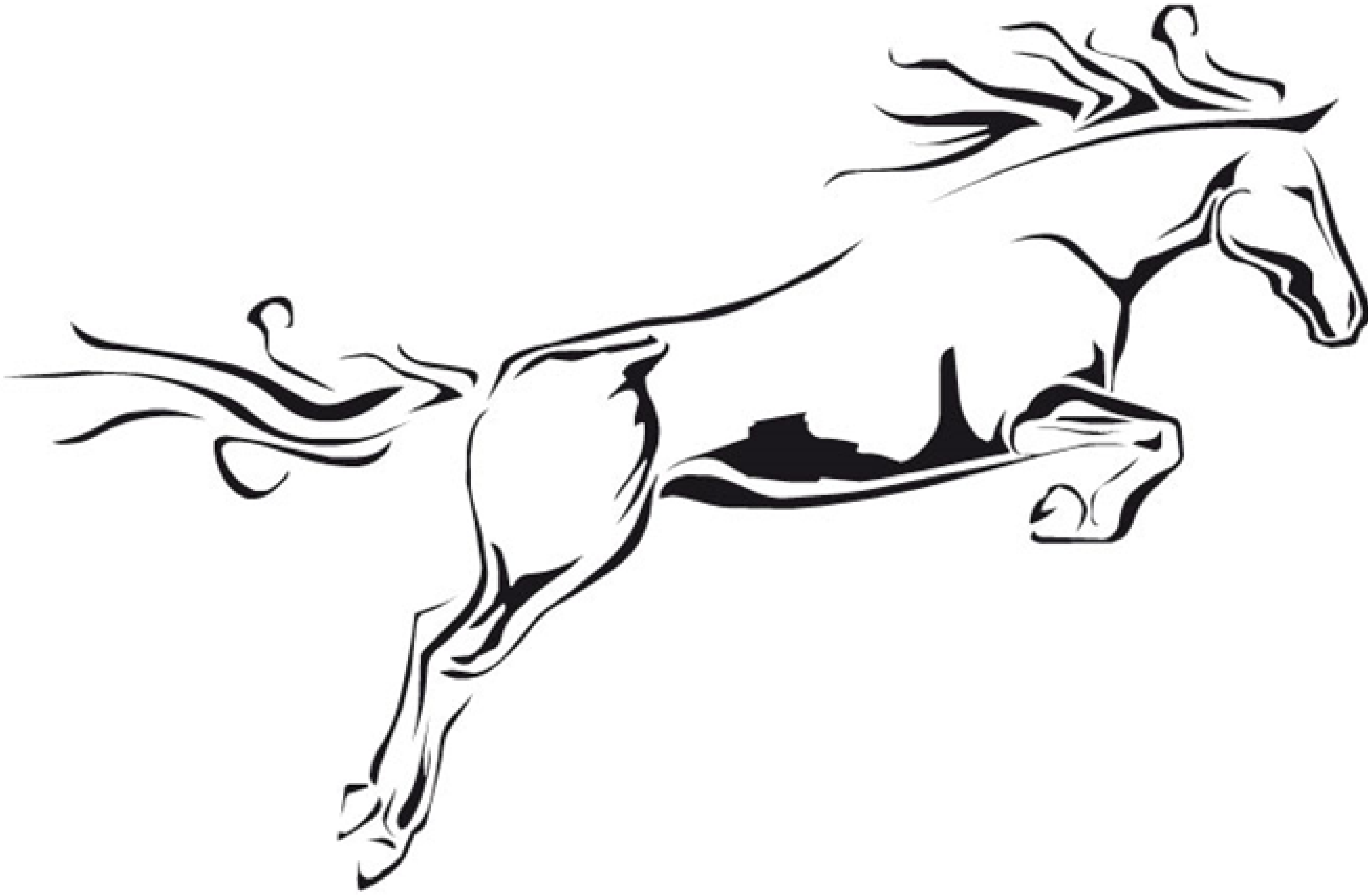
Conclusions:

- Domain Software Engineering (DSE) is a very promising *software development methodology*
- It has the potential to massively reduce *divergence* and *accidental* complexity
- It forces a close *cooperation* between business units (domain experts) and IT experts
- The base for success is an accurate, current domain model in business terms (business concepts & objects)
- The software implementation must strongly reflect the elements of the domain model

RDSE

Architecture Recommendations for Domain Software Engineering (DSE)

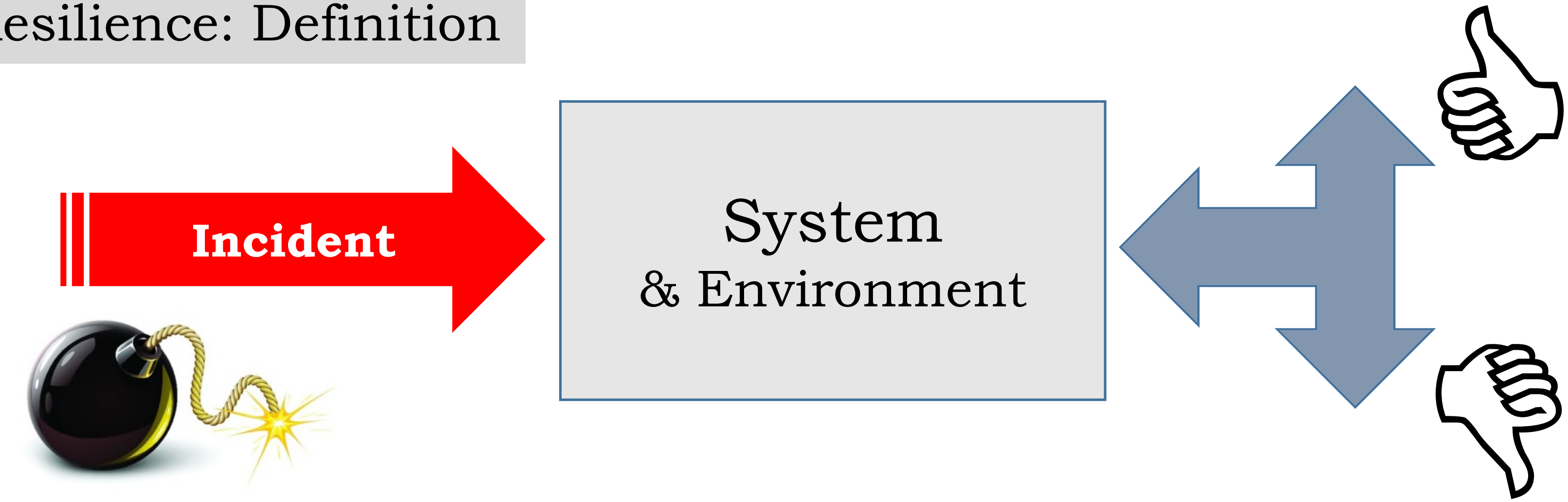
1. Gracefully build up an Ubiquituous Language between Business/Customer and IT (Implementer)
2. Define a consistent and complete domain model (hierarchical because of the size)
3. Push the formalization as far as possible (without losing the business/customer)
4. Use the terminology from the domain model/ubiquitous language in the code
5. Keep the domain model and the code implementation strictly synchronized at all times



https://www.wall-art.com

Part 4: Architecting for Resilience

Resilience: Definition



Resilience is the *capability* of a system with specific characteristics before, during and after a *disruption*

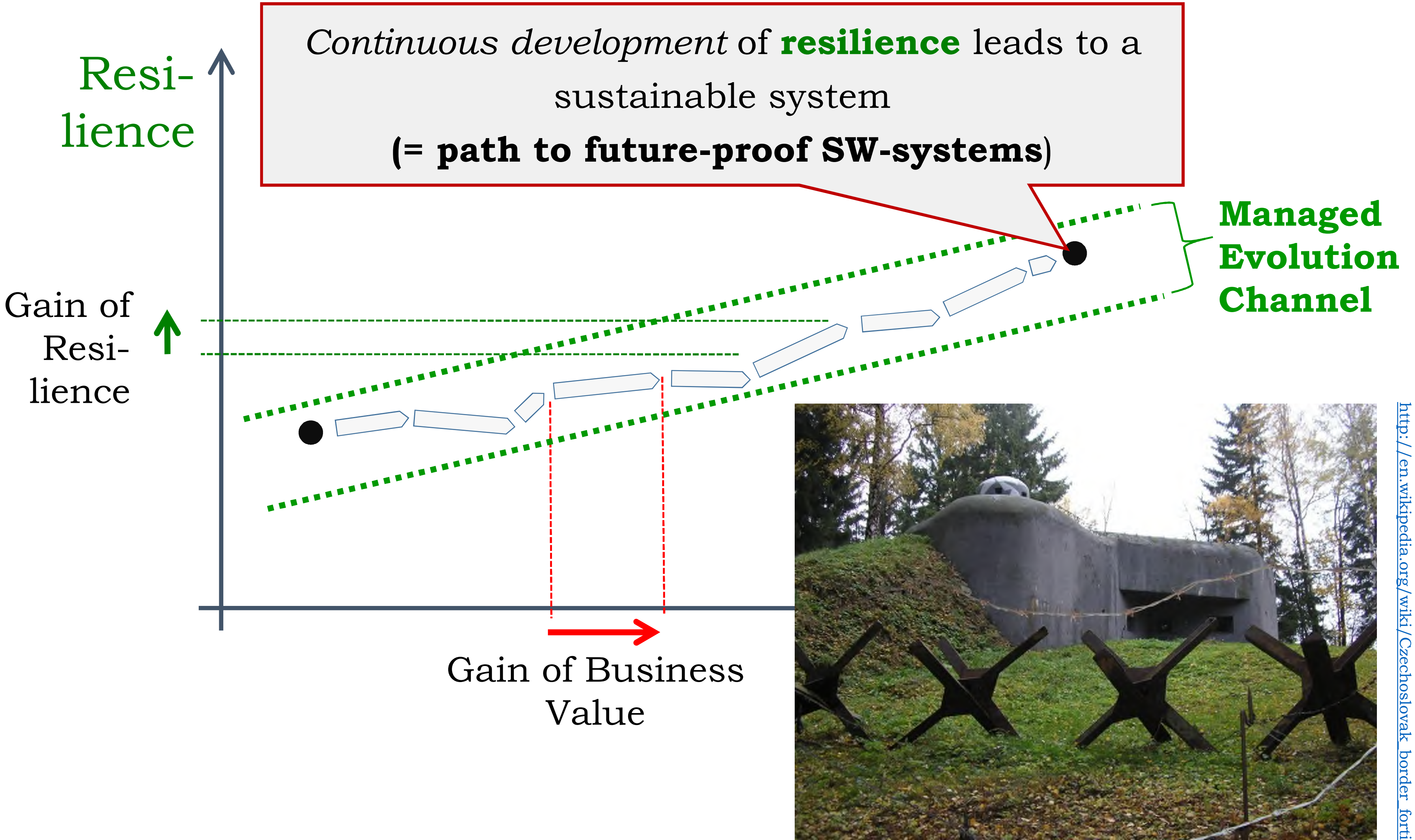
to *absorb* the disruption, *recover* to an acceptable level of performance,

and *sustain* that level for an acceptable period of time

- *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

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

Resilience Evolution Trajectory



http://en.wikipedia.org/wiki/Czechoslovak_border_fortifications

Resilience = Result of the relevant *properties* of the system

Safety

Security

Integrity

Availability

Confidentiality

Certifiability

Reliability

Diagnosability

Standards adherence

Accountability

Traceability

Non-Repudiation

Fault-Tolerance

Business Continuity

Recoverability

Graceful Degradation

Auditability

Performance

Real-TimeCapability

Fail-Save Behaviour

Survivability

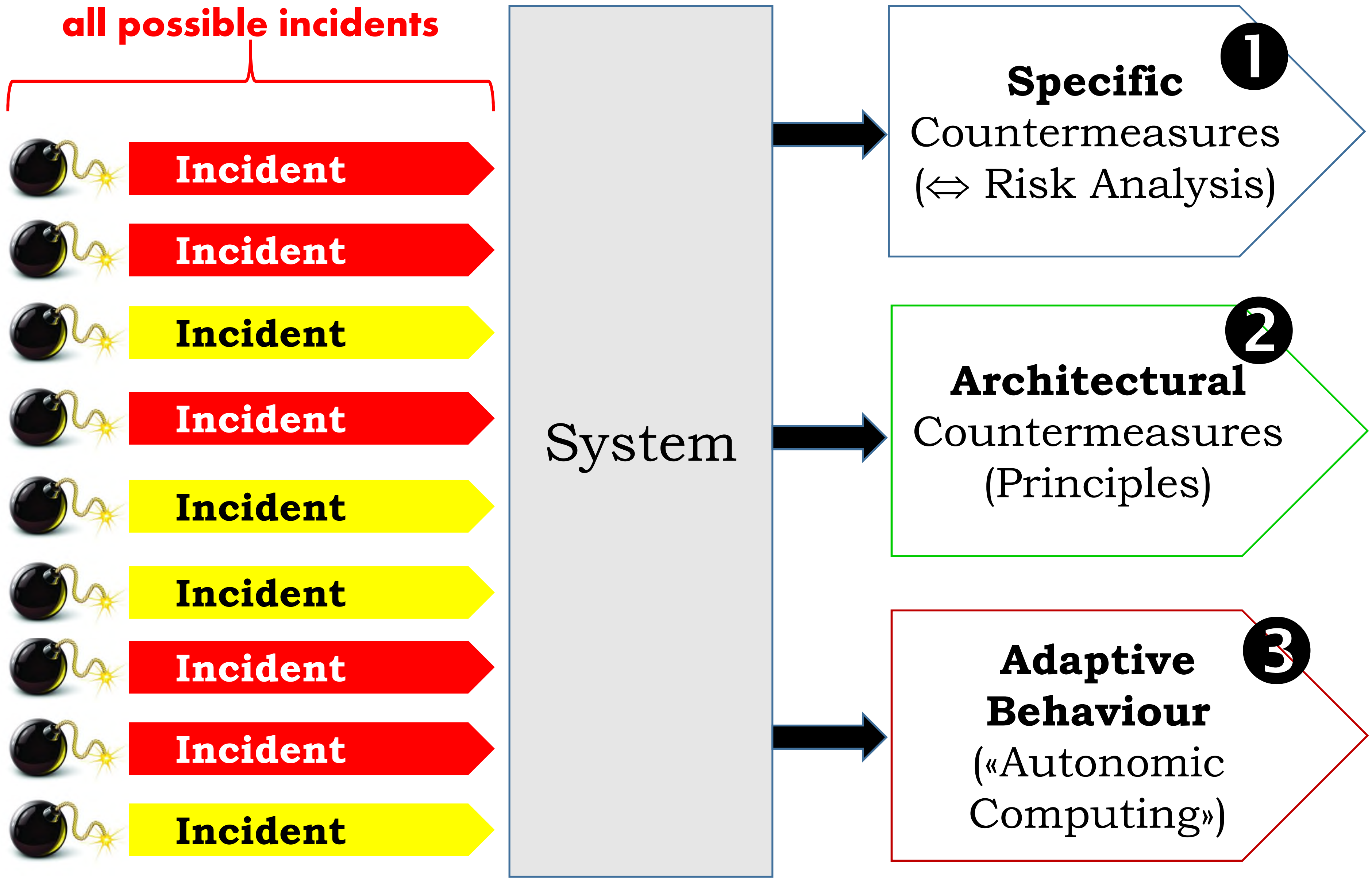
etc.

Quality Property Score Card

Example:
Automotive Domain

Resilience defined

#	System Quality Property	Weight 0: irrelevant 10: highest importance
	Primary Characteristics	
1	Business Value	**10**
2	Agility	**10**
Resilience:		
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		


all possible incidents
Incident
Incident
Incident
Incident
Incident
Incident
Incident
Incident
Incident
System
❶ Specific Countermeasures (⇔ Risk Analysis)
❷ Architectural Countermeasures (Principles)
❸ Adaptive Behaviour («Autonomic Computing»)

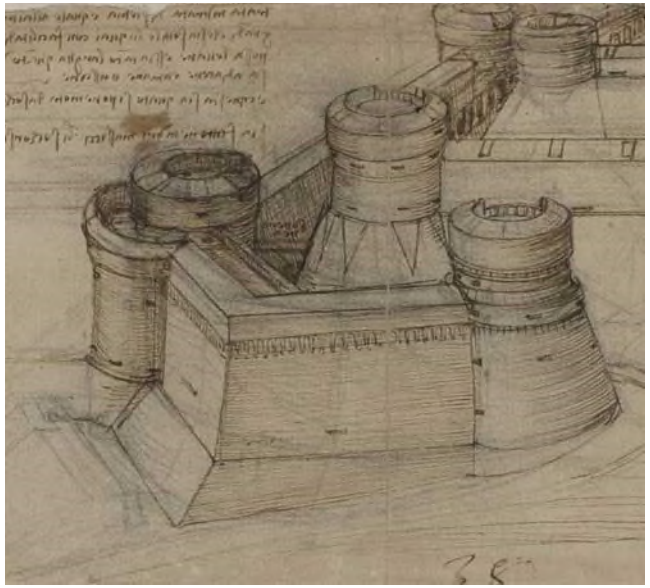
http://travelinos.com/castles

Software Resilience

⇒ **Architecture** (Structure)

- governed by **resilience architecture principles**

Resilience Architecture Principles



http://www.marinabaysands.com

General Resilience Architecture Principles (Overarching Principles)

Safety Principles

Security Principles

Integrity Principles

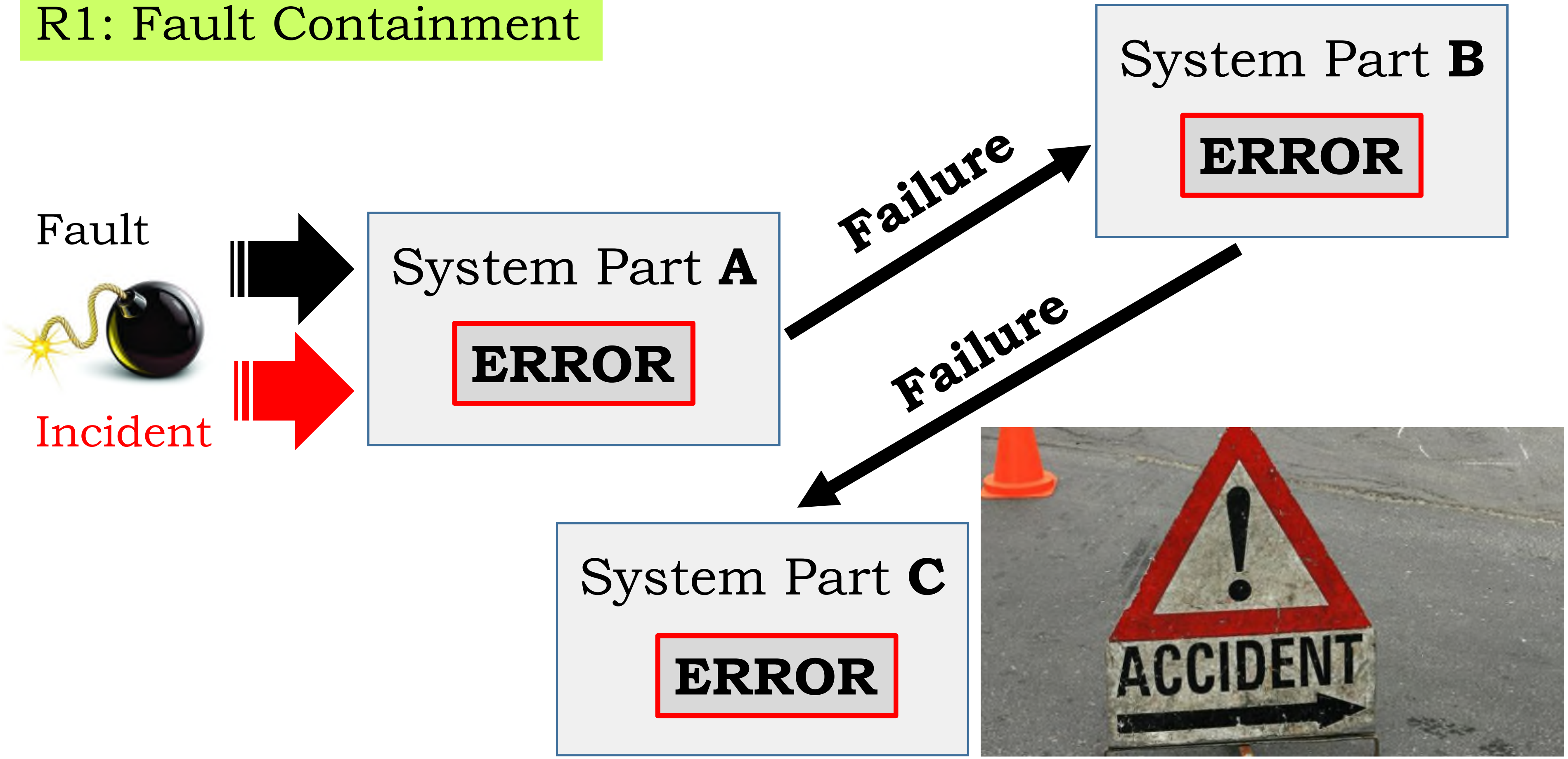
Availability Principles

... etc.

General (Overarching) Architecture Principles for Resilience

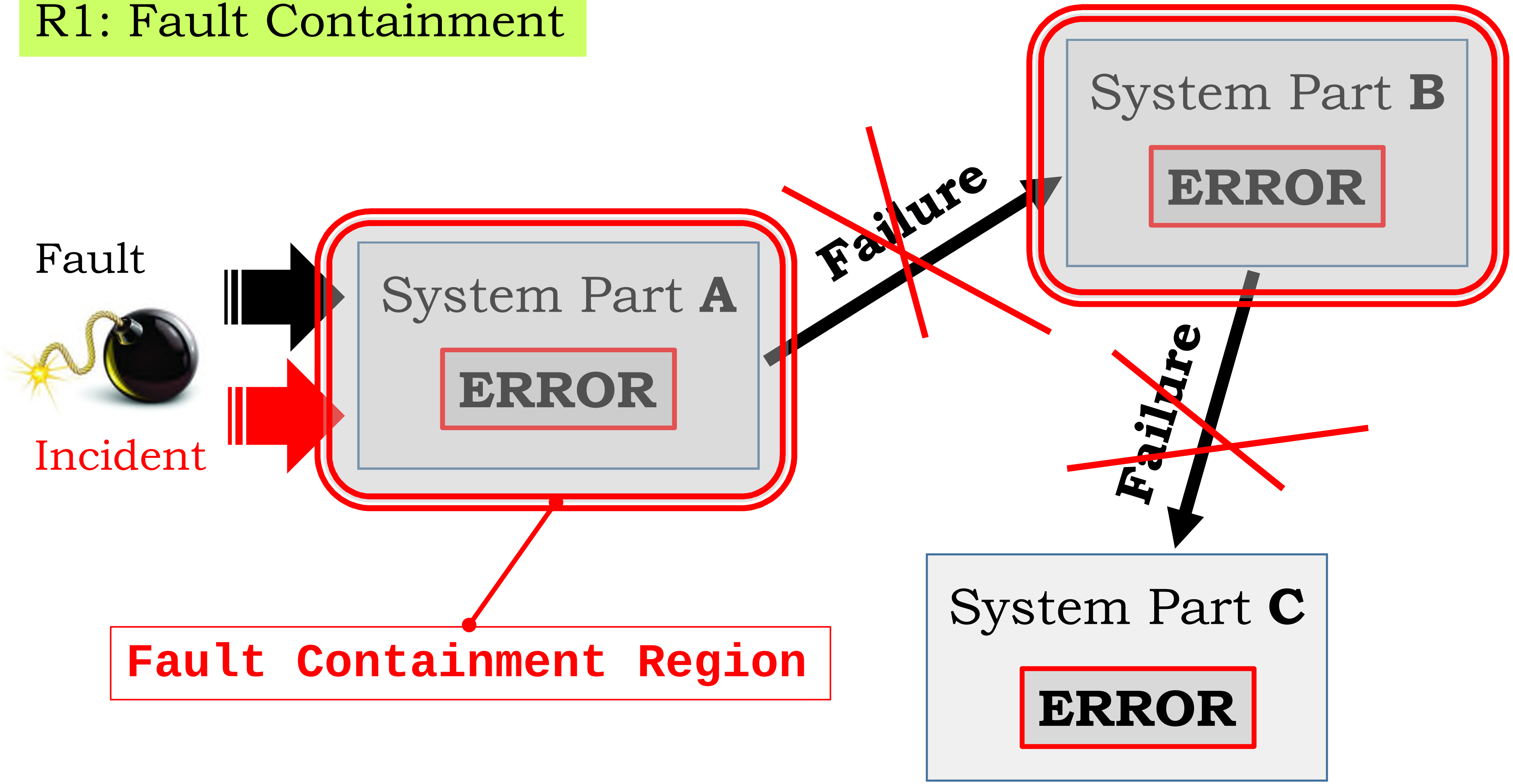
- R1: Fault Containment Regions
- R2: Single Points of Failure
- R3: Multiple Lines of Defense
- R4: Fail-Safe States
- R5: Graceful Degradation
- R6: Dependable Foundation (Infrastructure)

R1: Fault Containment



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

R1: Fault Containment



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

R1: Fault Containment

R1

Resilience Architecture Principle R1:

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

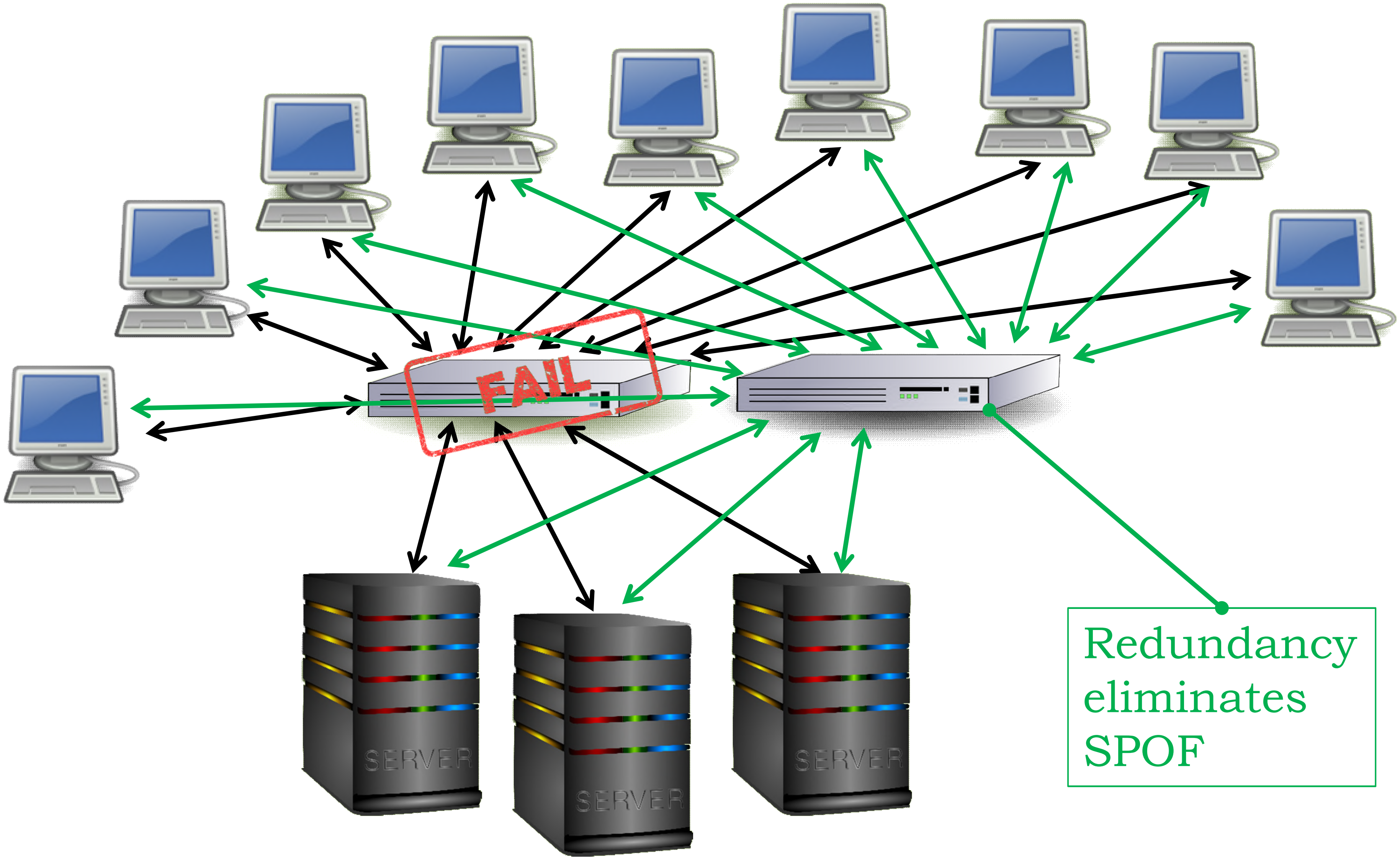
R2: SPoF

	Security Architecture (Defense)	**Safety** Architecture (Accidents)	**Performance** Architecture (Real-Time)	**System Management** Architecture (Control)	etc.
Business Architecture (Business Processes)					
Applications Architecture (Functionality)	FAILED			FAILED	
Information (Data) Architecture (Information & Data)			FAILED		
Integration Architecture (Cooperation Mechanisms)	FAILED				
Technical Architecture (Technical Infrastructure)					

A single point of failure (SPOF)
is a part of a system that,
if it *fails*,
will stop the *entire* system from working

R2: SPoF

Example: Computer Network



Redundancy eliminates SPOF

R2: SPoF

R2

Resilience Architecture Principle R2:

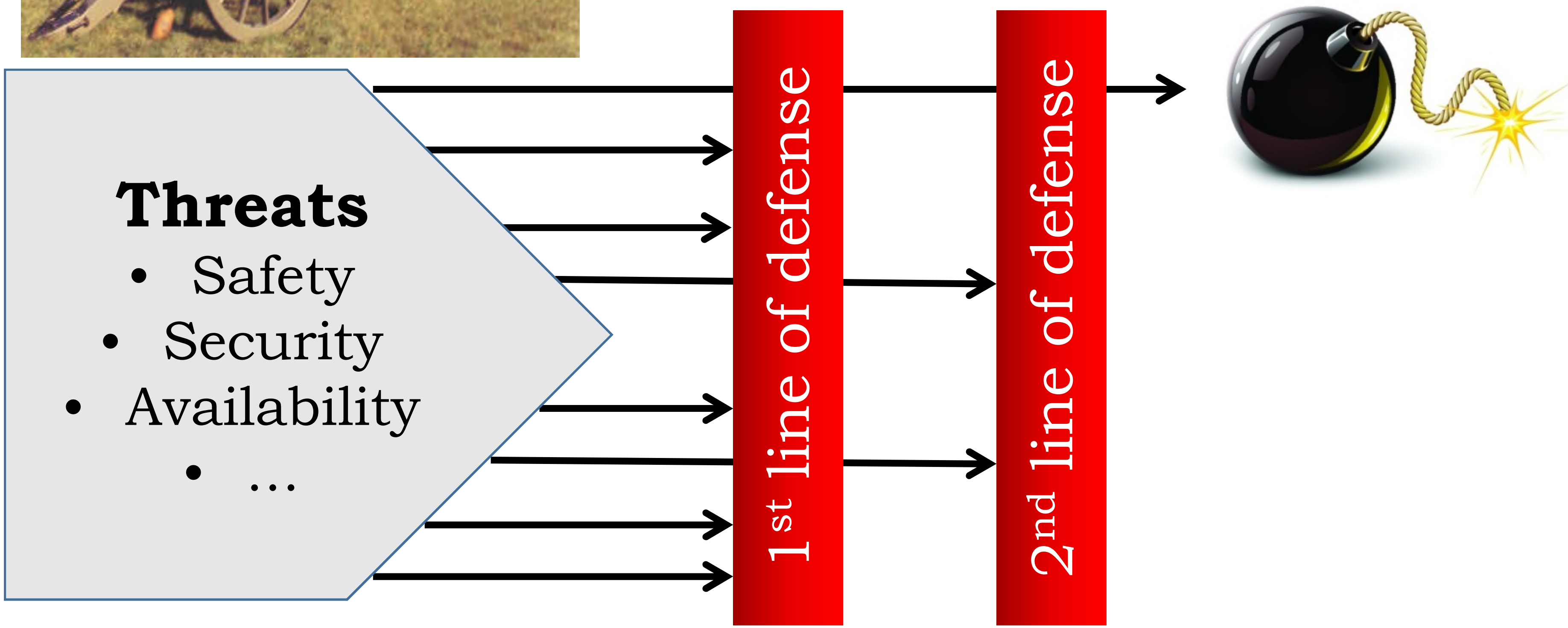
Single Points of Failure

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 or improving the architecture

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

R3: Multiple Lines of Defense

http://www.buckstix.com



R3: Multiple Lines of Defense

R3

Resilience Architecture Principle R3:

Multiple Lines of Defense

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

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

R4: Fail Safe States

DEFINITIONS

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

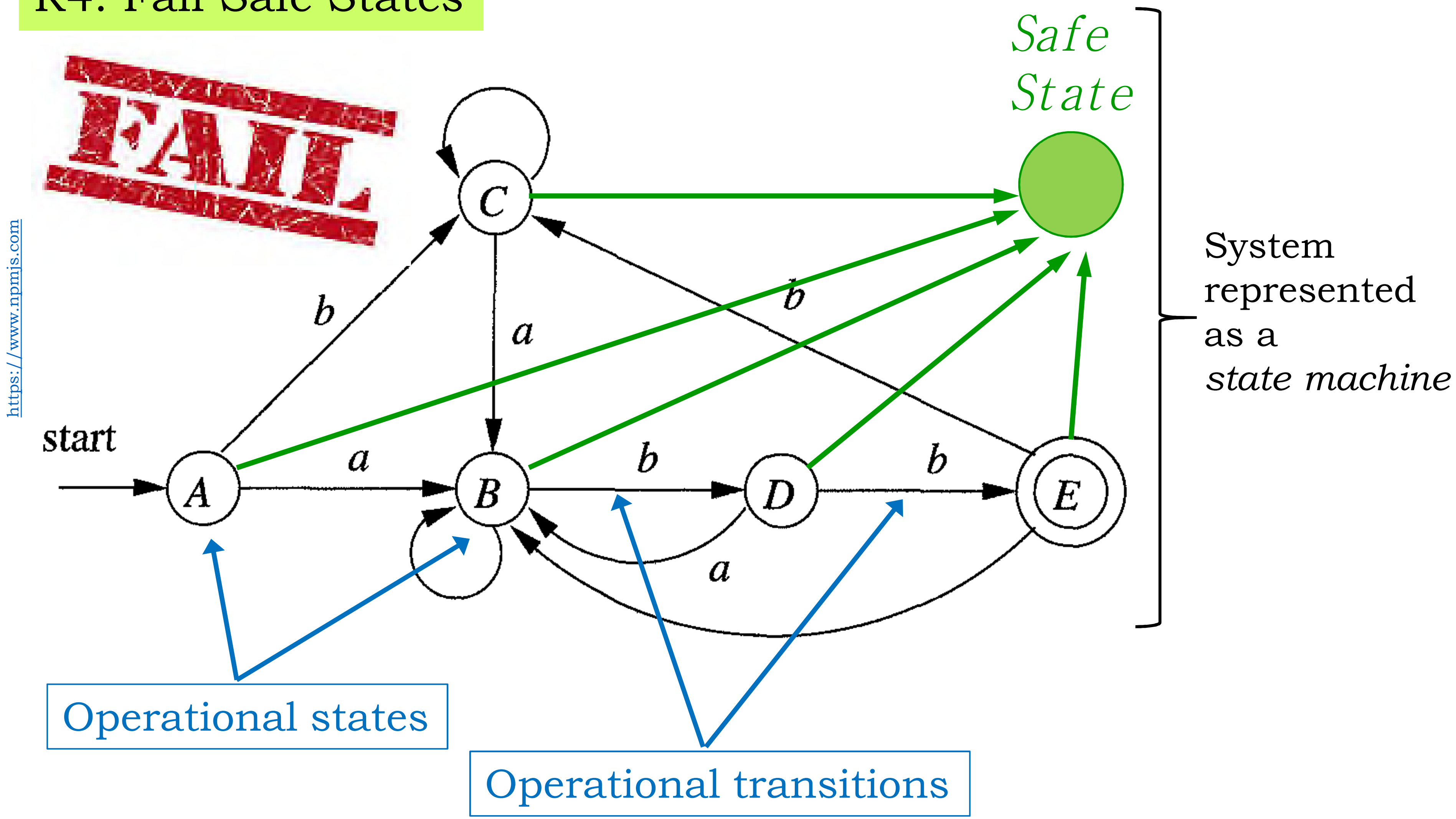
It will go into a *fail-safe state* and stop working.

http://www.viewsender.com

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

Peter Herena, 2011

R4: Fail Safe States



R4: Fail Safe States

R4

Resilience Architecture Principle R4:

Fail-Safe States

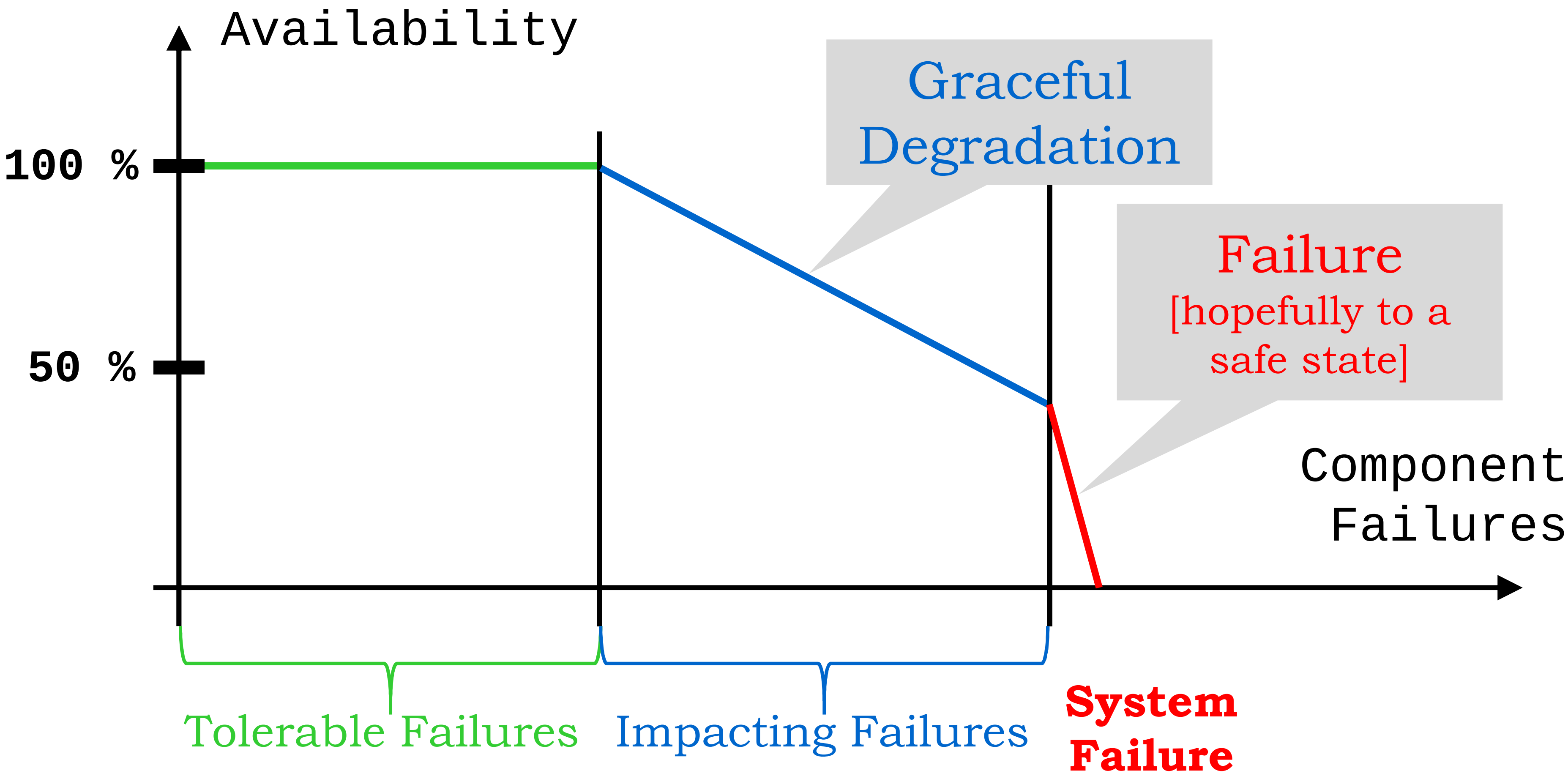
1. Define and specify fail-safe states in the operation of the system (= states which do not endanger lives or property)
2. Define and implement trajectories to fail-safe states from all possible states in case of irrecoverable faults, errors or failures
3. The trajectories to fail-safe states must be automatic, short and dependable

Justification: When a system encounters an irrecoverable fault, error or failure a planned transition into a safe state will avoid damage

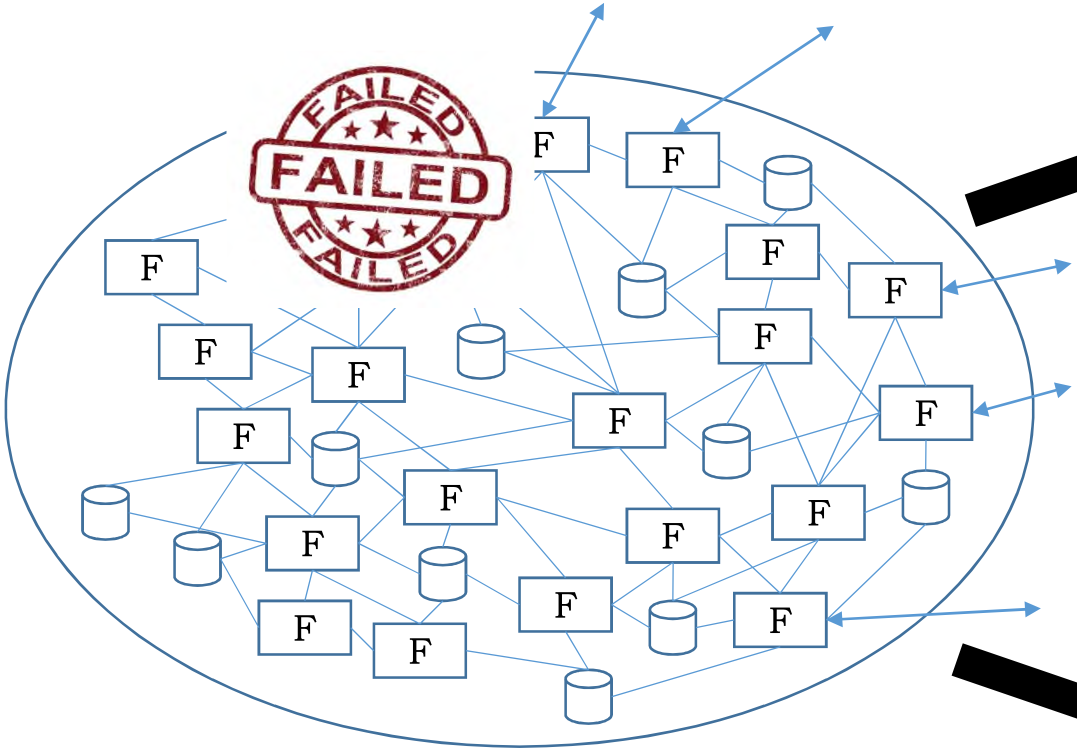
R5: Graceful Degradation

Availability

Graceful Degradation is with respect to a *specific* resilience property



R5: Graceful Degradation



reduced level of **functionality** or **resilience**

Fault Tolerance: succesful **redundancy**

Graceful Degradation = Fault Tolerance Engineering

R5: Graceful Degradation

R5

Resilience Architecture Principle R5:

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 failures by 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

R6: Dependable Foundation (Infrastructure)

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

DEFINITIONS

Resilience Infrastructure:

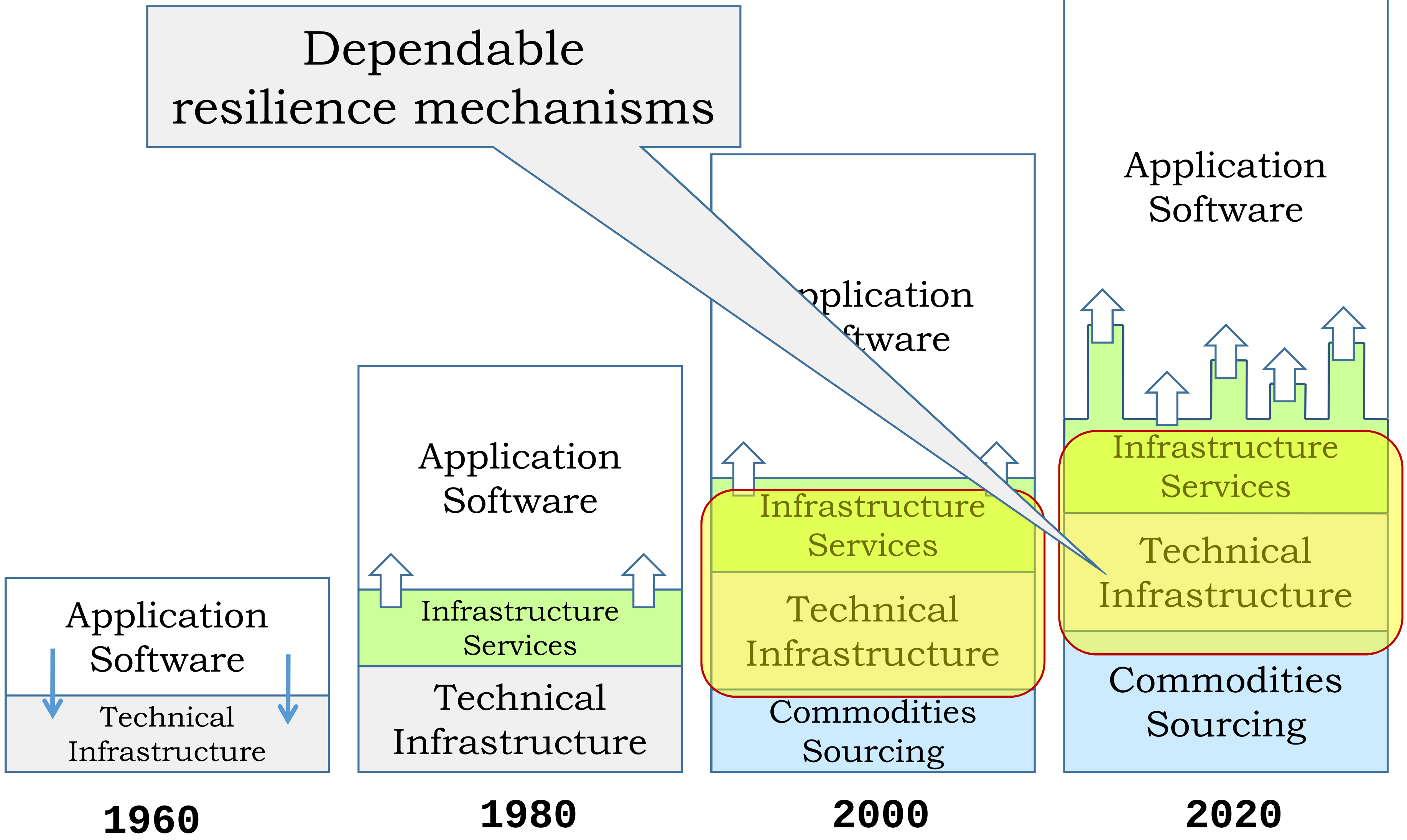
Set of proven resilience technologies and services supporting the resilience properties (availability, security, performance, …) of software systems

https://www.npmjs.com

Resilience Engineer Roles:

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

R6: Dependable Foundation (Infrastructure)



R6: Dependable Foundation (Infrastructure)

R6

Resilience Architecture Principle R6:

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 (But avoid vendor lock-in)

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

Exams Winter Term 2015/16

Planned dates (oral exams):

Wednesday, February 3, 15:00 – 18:00

Thursday, Februar 4, 09:00 – 11:30

Please write an email to katrin.heber@tu-dresden.de (Secretary of the Chair of Software Technology). She will schedule your timing of the exam.

Alternate date: If none of these dates suits you, there will be another exam time after the beginning of the Summer Term 2016. The date will be announced later. Please register with katrin.heber@tu-dresden.de

Antrittsvorlesung Prof. Dr. Frank J. Furrer
21. Janaury 2016 / 13:00 – 14:15
http://st.inf.tu-dresden.de/fjfurrer/

Reminder

13:00	**BEGRÜSSUNG**
	Dekan der Fakultät Informatik &
	Prof. Dr. Uwe Aßmann | Technische Universität Dresden
13:15	**ANTRITTSVORLESUNG**
	SOFTWARE
	– Gratwanderung zwischen Erfolgsgeschichten und Katastrophen?
	Prof. Dr. Frank J. Furrer | Technische Universität Dresden
14:15	**KAFFEEPAUSE**
15:00	**Emergenz in Cyber-Physical System of Systems (CPSoS)**
	Was führt zu emergentem Verhalten in Cyber-Physical System of Systems?
	Prof. Dr. Hermann Kopetz | Technische Universität Wien
15:45	**Gotische Kathedralen und Software-Architekturen**
	– Gibt es da irgendeinen Zusammenhang?
	Prof. Dr. Manfred Nagl | RWTH Aachen
16:30	**Building correct cyberphysical systems**
	– and how to improve current practice
	Prof. Dr. Susanne Graf | VERIMAG Grenoble

January 20, 2016:

- **Part 4B**: Architecting for Resilience
    - *Specific Resilience Architecture Principles*
    - *Autonomic Computing*