TECHNISCHE UNIVERSITÄT DRESDEN

Software Composition – Some Challenges of the Next Decade

Software Composition 2012

Uwe Aßmann
Technische Universität Dresden
Chair of Software Engineering

Prague, June 1, 2012

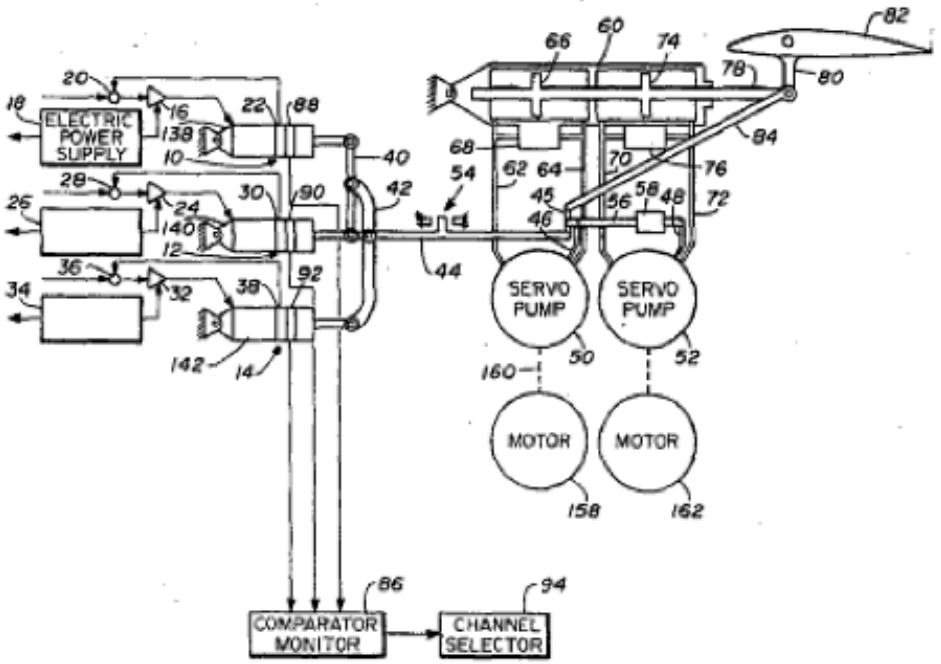
- Software composition has come a long way from the times of the end of the 80s, when the problem of inheritance anomaly stopped the parallel object-oriented languages.
  It is time to look back at some of the old challenges that started the field and discuss, how we can solve them now or in the near future, the next decade.
  These challenges include
- the coordination of parallel objects in pools
- context specifications based on concerns and roles,
- and the quality-based dispatch in multi-objective optimized systems (MOO systems) for the modeling and programming of cyber-physical systems (life-by-wire).
- The talk gives an overview on these challenges for software composition for the next decade, and formulates several research problems (The Prague composition manifesto).

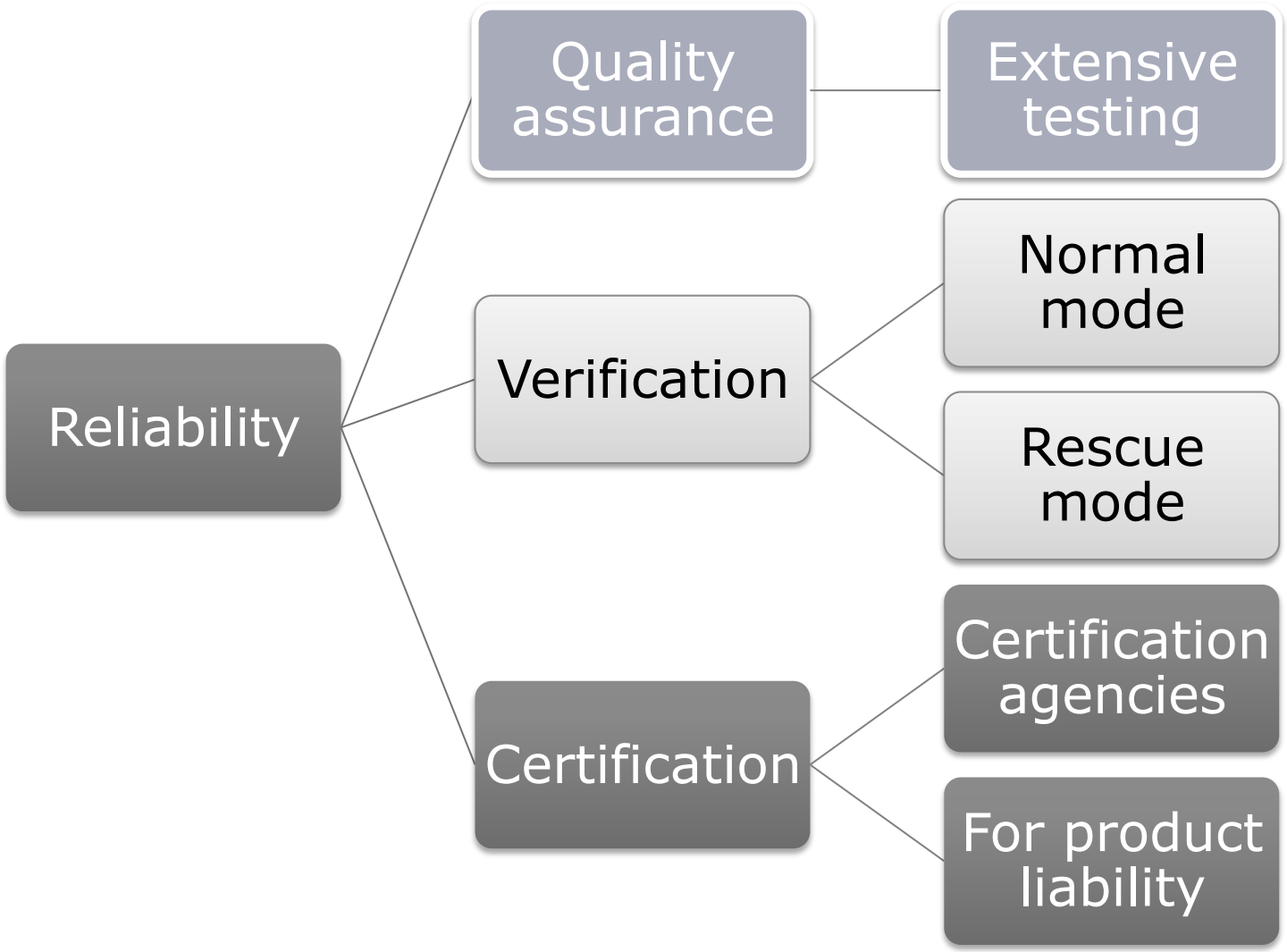
1. Life-by-Wire
2. Separations of Concerns and composition
3. Parallelism:the POOL problem revisited
4. Context-sensitivity: Reliable adaptivity with quality roles
5. Efficiency-based development with MOO architectures
6. The ultimate challenge: Specifying composition systems
7. The Prague composition manifesto

With cyber-physical systems (CPS)

CHAP.1 LIFE-BY-WIRE

- **FLY-BY-WIRE**
- William G. Redmond - US Patent 3,679,156, 1972
- http://www.google.com/patents?hl=de&lr=&vid=USPAT3679156&id=GTswAAAAEBAJ&oi=fnd&dq=fly-by-wire&printsec=abstract#v=onepage&q=fly-by-wire&f=false




Reliability
Quality assurance
Extensive testing
Verification
Normal mode
Rescue mode
Certification
Certification agencies
For product liability

CHAP.1.2 THE NEAR FUTURE: HELP-BY-WIRE

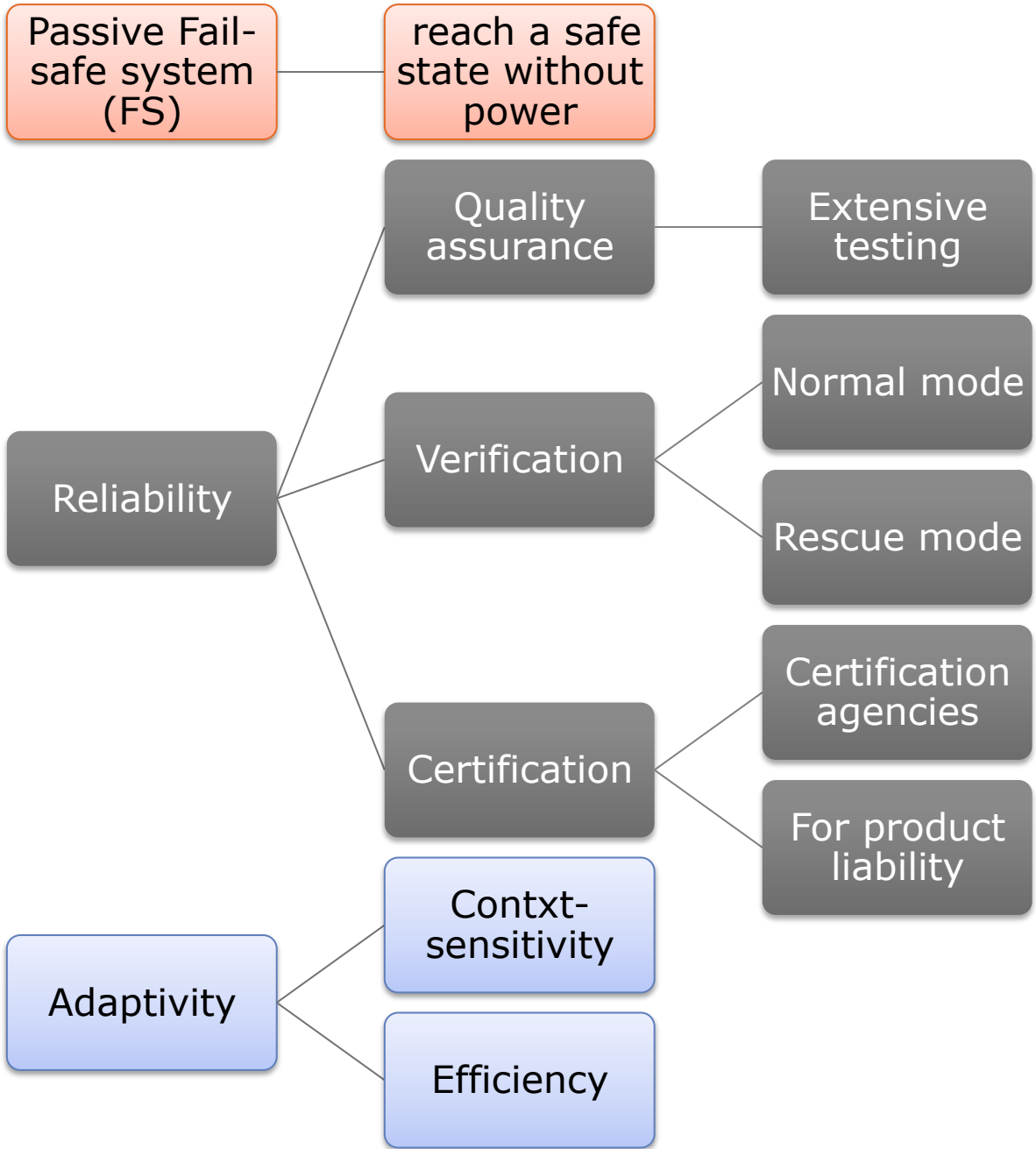
http://de.wikipedia.org/wiki/Ungl%C3%BCck_bei_der_Loveparade_2010

- „In der Europäischen Union werden jährlich ueber 7.000 Fussgänger und 2.000 Fahrradfahrer bei Verkehrsunfällen getätet. Hunderttausende werden darüber hinaus bei Verkehrsunfällen verletzt.“
- [M. Bischoff. Aktive Sicherheitssysteme fuer den Schutz von Fussgaengern im Strassenverkehr]

Um 4.30 Uhr hatte sich Regina F. auf den Weg zur Arbeit gemacht. Von der Britzer Straße in Schöneweide zum Putzen nach Marienfelde. Sie schaffte es nicht mal bis zum S-Bahnhof.

Viel zu schnell kam ein Kombi um die Ecke gefahren. So schnell, dass das Auto aus der Kurve flog und über den Bürgersteig raste. Durch den Aufprall wurde Regina F. auf die Straße geschleudert. Nur kurz hielt der Mann an, dann gab er Gas und raste davon.

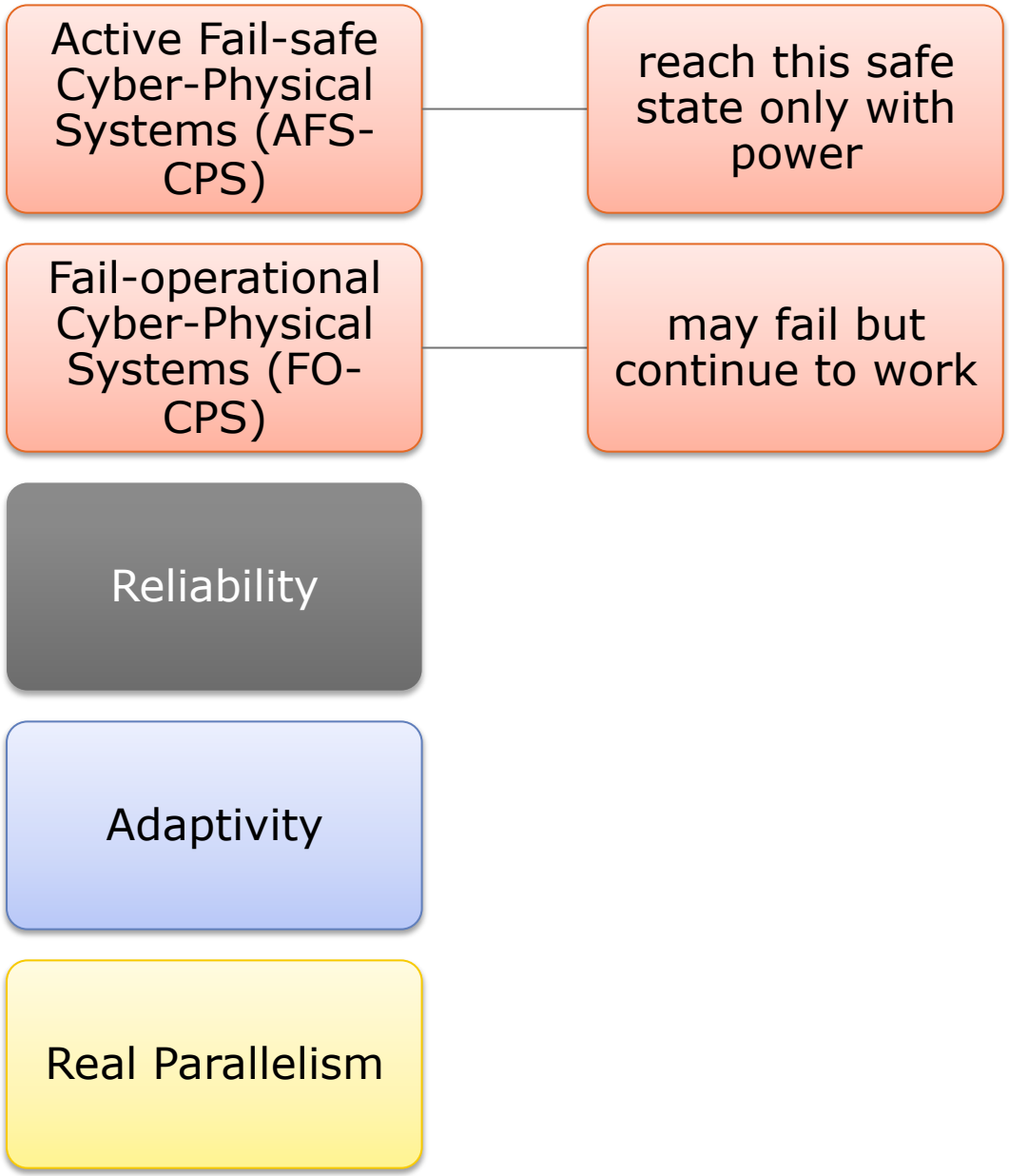
http://www.bz-berlin.de/bezirk/treptow/raser-faehrt-oma-um-und-macht-sich-davon-article1373880.html


Passive Fail-safe system (FS)
reach a safe state without power
Reliability
Quality assurance
Extensive testing
Verification
Normal mode
Rescue mode
Certification
Certification agencies
For product liability
Adaptivity
Contxt-sensitivity
Efficiency

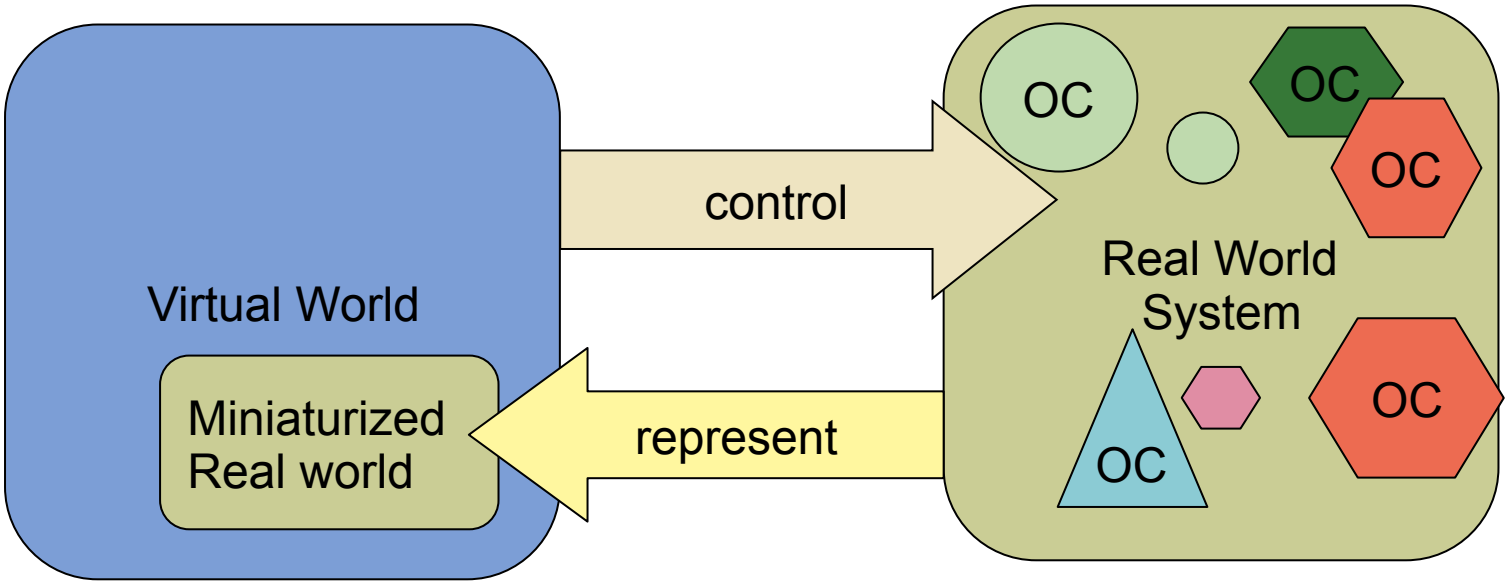
1.3 THE MID-TERM FUTURE: MOVE-BY-WIRE

https://upload.wikimedia.org/wikipedia/commons/thumb/1/14/Google_self_driving_car_at_the_Googleplex.jpg/800px-Google_self_driving_car_at_the_Googleplex.jpg

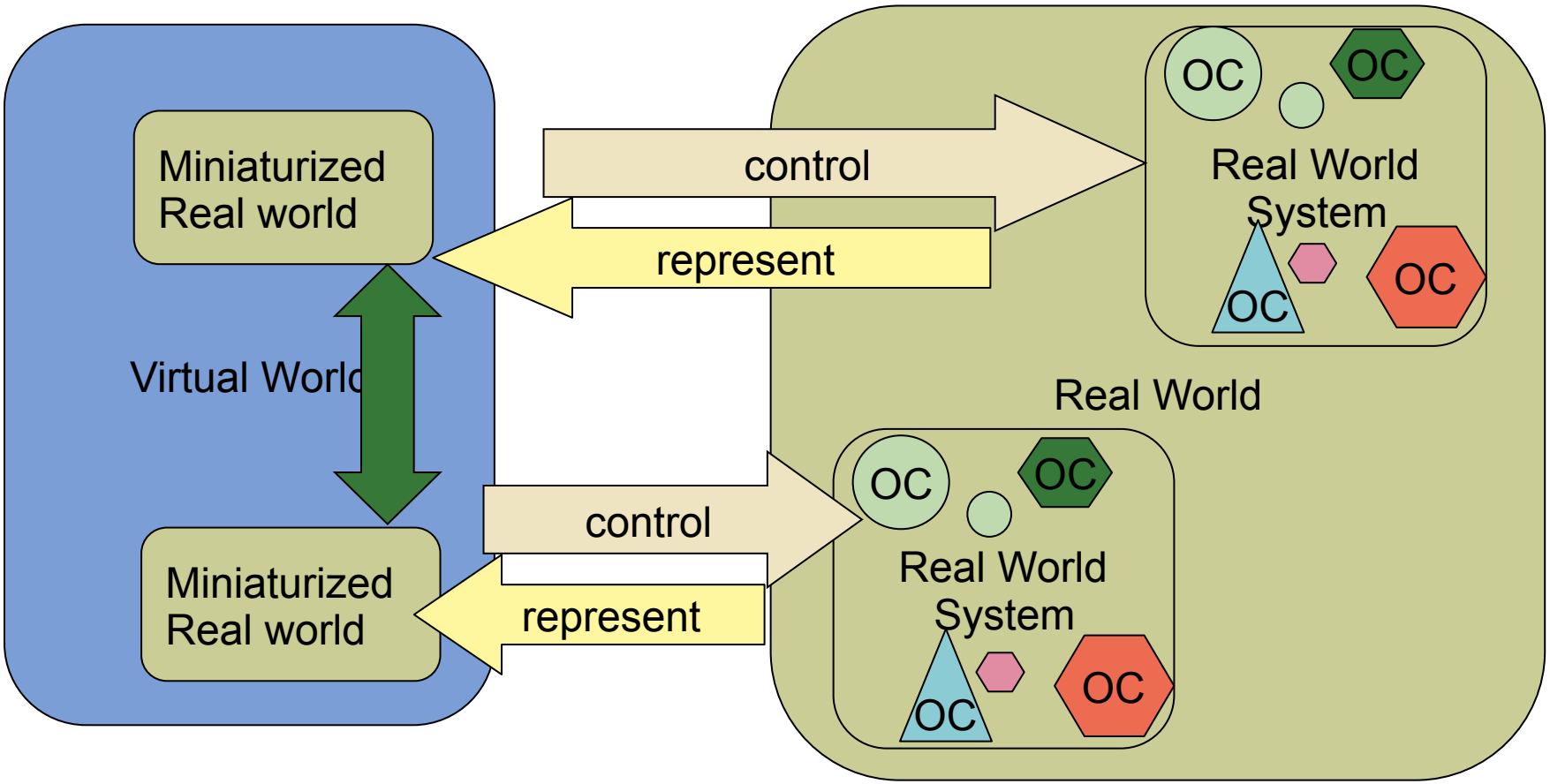
http://commons.wikimedia.org/wiki/File:Traffic_seen_from_top_of_Arc_de_Triomphe.JPG

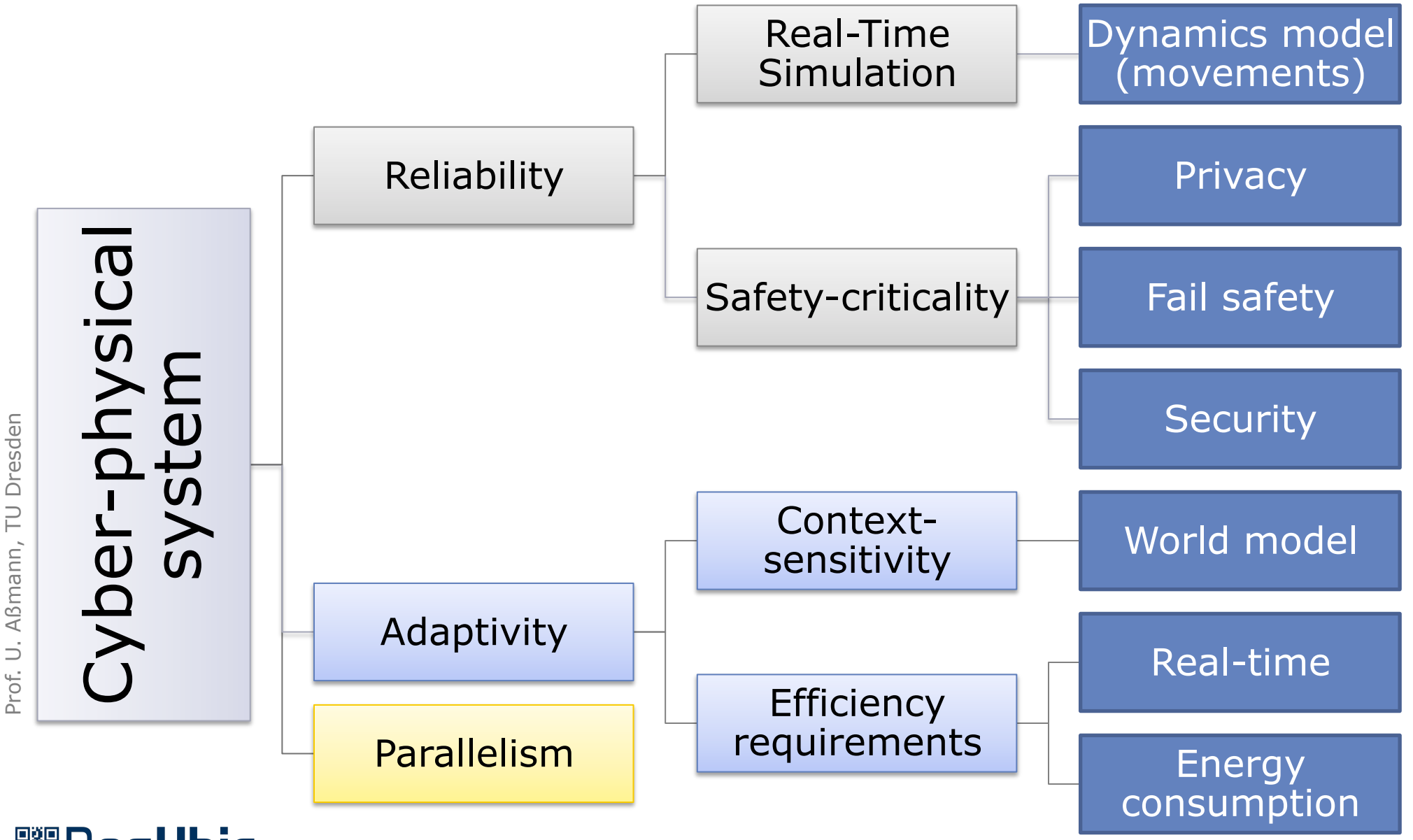

Active Fail-safe Cyber-Physical Systems (AFS-CPS)
reach this safe state only with power
Fail-operational Cyber-Physical Systems (FO-CPS)
may fail but continue to work
Reliability
Adaptivity
Real Parallelism

- Control of the intelligent things in space and time
  - Self regulation
  - Self optimization
  - Self organisation
- Dual reality



- Systems of CPS




Cyber-physical system
Reliability
Real-Time Simulation
Dynamics model (movements)
Safety-criticality
Privacy
Fail safety
Security
Adaptivity
Context-sensitivity
World model
Efficiency requirements
Real-time
Energy consumption
Parallelism

2. SEPARATION OF CONCERNS AND COMPOSITION

- E.W.Dijkstra in EWD 447 „On the role of scientific thought“:

- Let me try to explain to you, what to my taste is characteristic for all intelligent thinking. It is, that one is willing to study in depth an aspect of one's subject matter in isolation for the sake of its own consistency, all the time knowing that one is occupying oneself only with one of the aspects.

- It is what I sometimes have called "*the separation of concerns*", which, even if not perfectly possible, is yet the only available technique for effective ordering of one's thoughts, that I know of. This is what I mean by "focussing one's attention upon some aspect”.

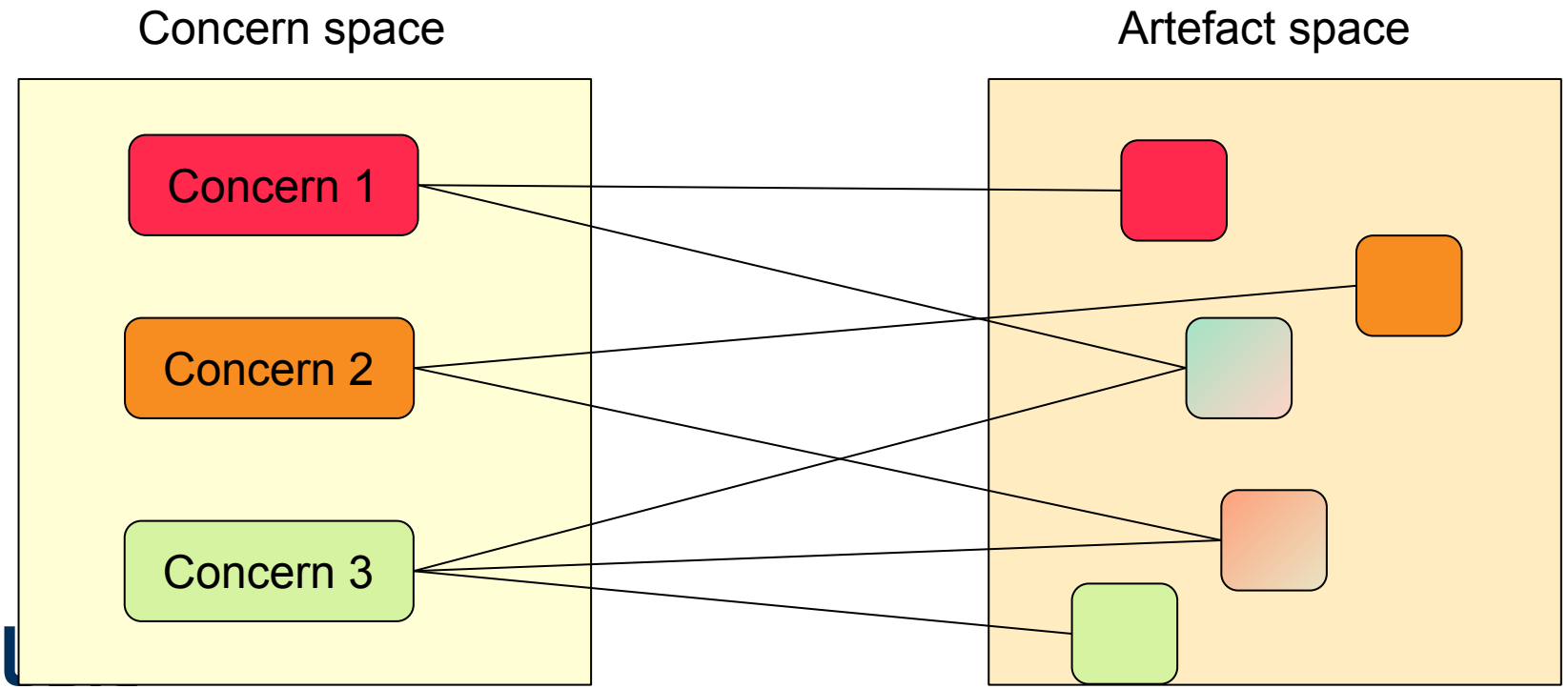
- The current approaches to aspect-orientation are not covering this:
  - SoC must support decomposition **and** composition
  - Changing focus is not supported

- **Ch 1: How can we conceptualize separations of concerns so that decomposition and composition are supported?**

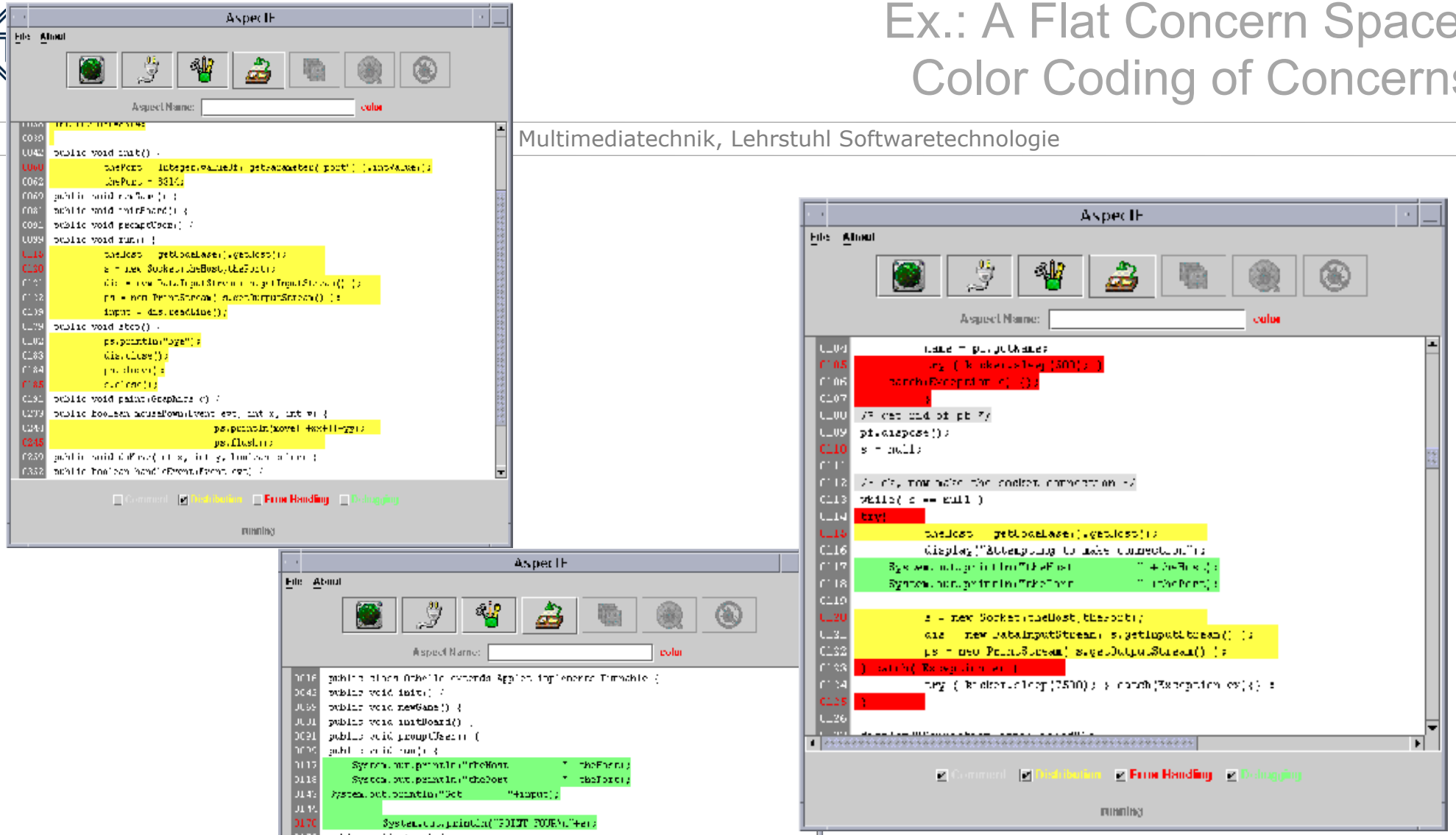
A Simple Conceptualization: Concerns and Artefacts

A **view** is a representation of a whole system from the perspective of a related set of concerns [ISO/IEC 42010:2007, Systems and Software Engineering -- Recommended practice for architectural description of software-intensive systems]

- An **artefact space** (solution space) consists of a set of components, models, or programs
- An **concern (aspect)** is *related* to a **slice** of the components of the artefact space, the **view**
- A **concern space** consists of a set of concerns and an algebraic structure



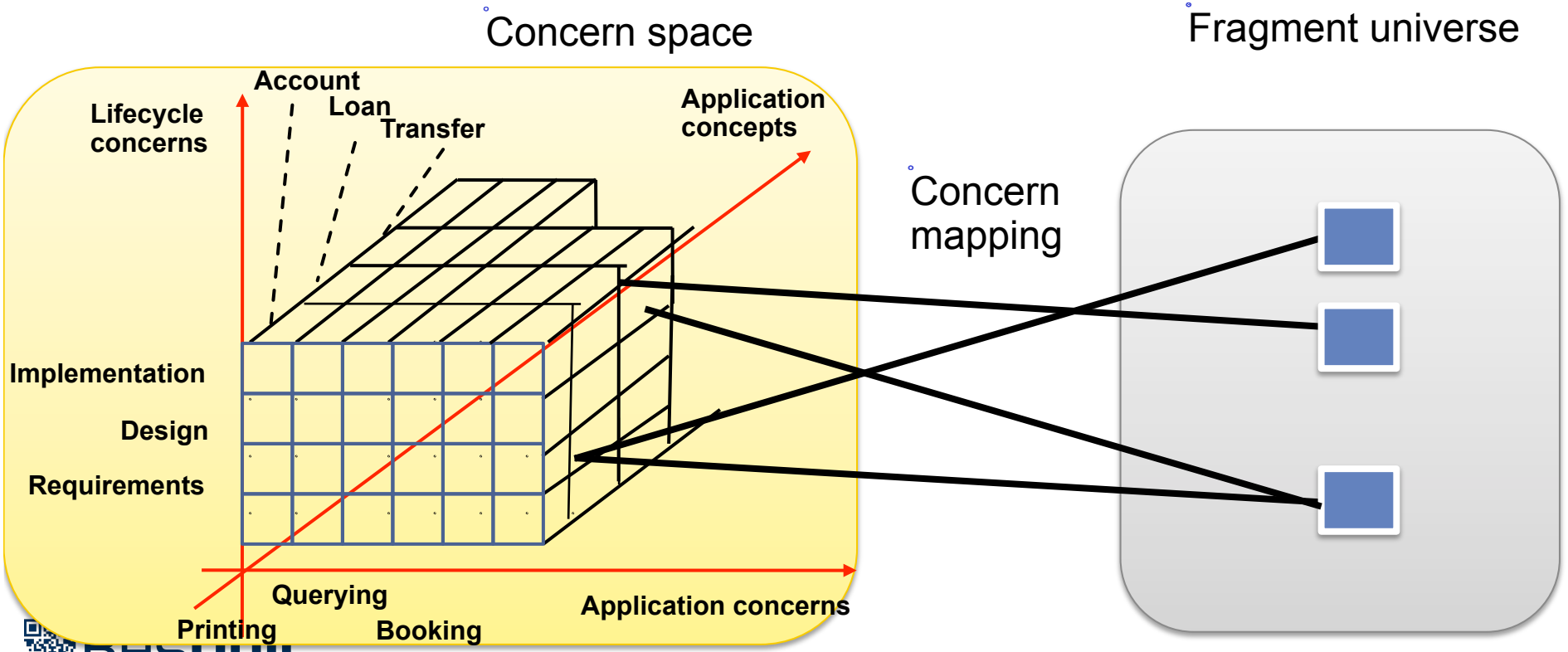
Ex.: A Flat Concern Space: Color Coding of Concerns



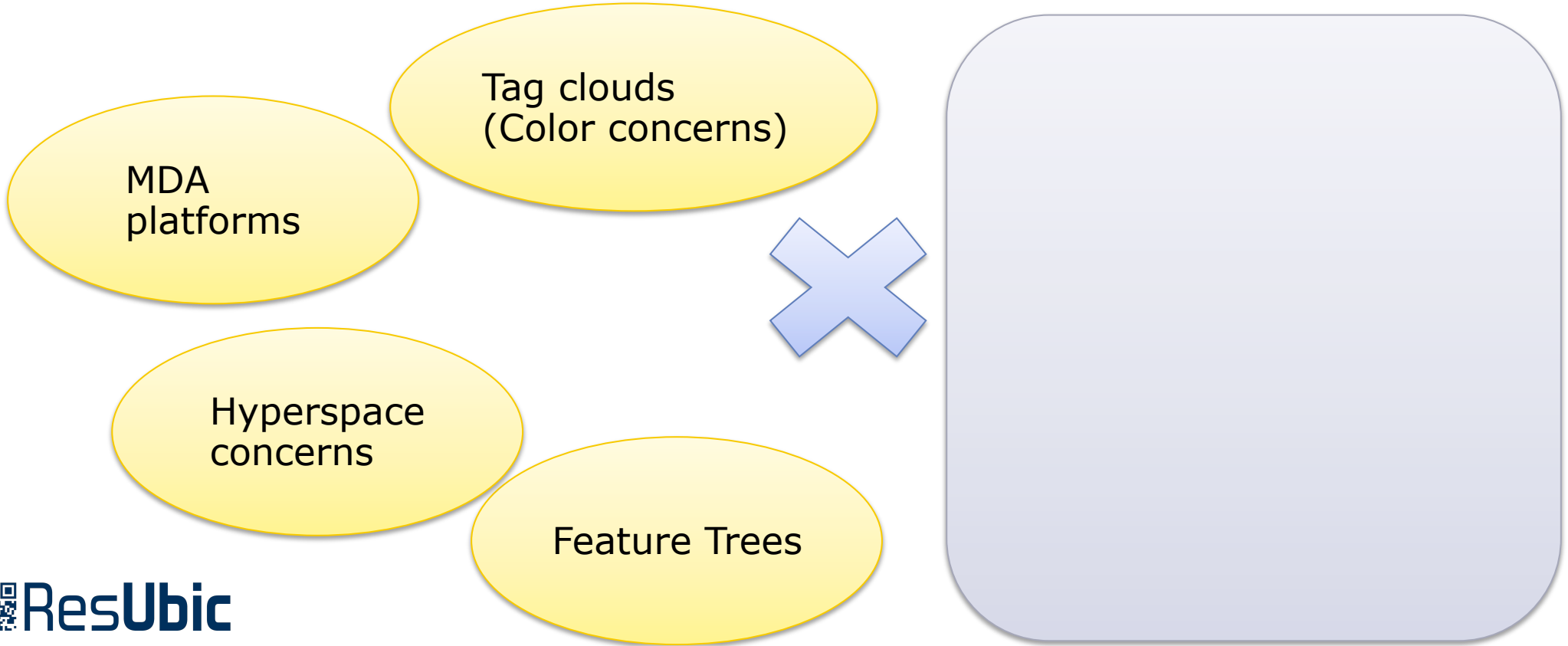
Fragments can be colored with regard to a certain concern (concern mapping)
[Panas, Andersson, Aßmann: The Editing Aspect of Aspects SEA 2002]

Ex. Hyperspaces: The Concern Mapping maps Concern Matrix to the Sets of Fragments

- Concern space and fragment universe connected via a concern mapping (crosscut graph)
- One fragment can relate to many concerns:
  - (concern_1, .., concern_n) x fragment
- The concern mapping results from selection/query expressions

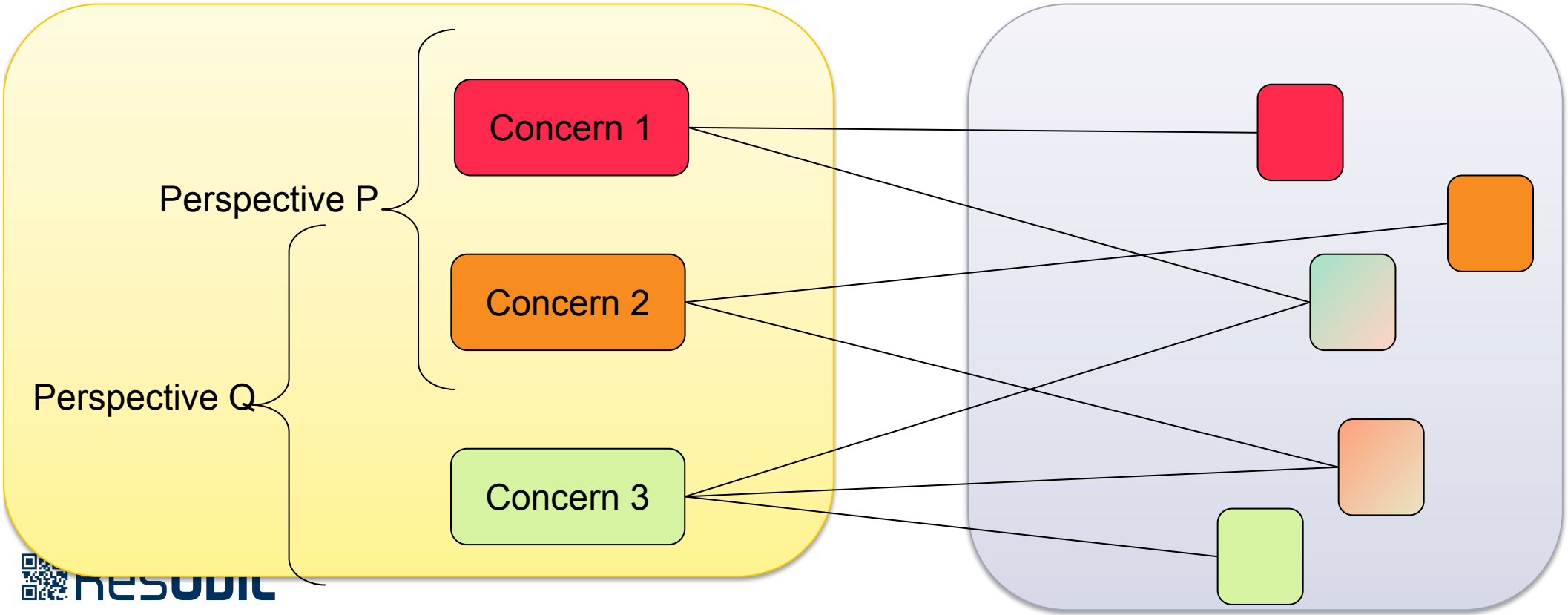


- A **perspective (viewpoint)** consists of a set of concerns in the concern space, with algebraic structure
- A **perspective model** contains a set of perspectives
- Refined: A **concern space** contains a set of perspectives and concerns *with algebraic structure*

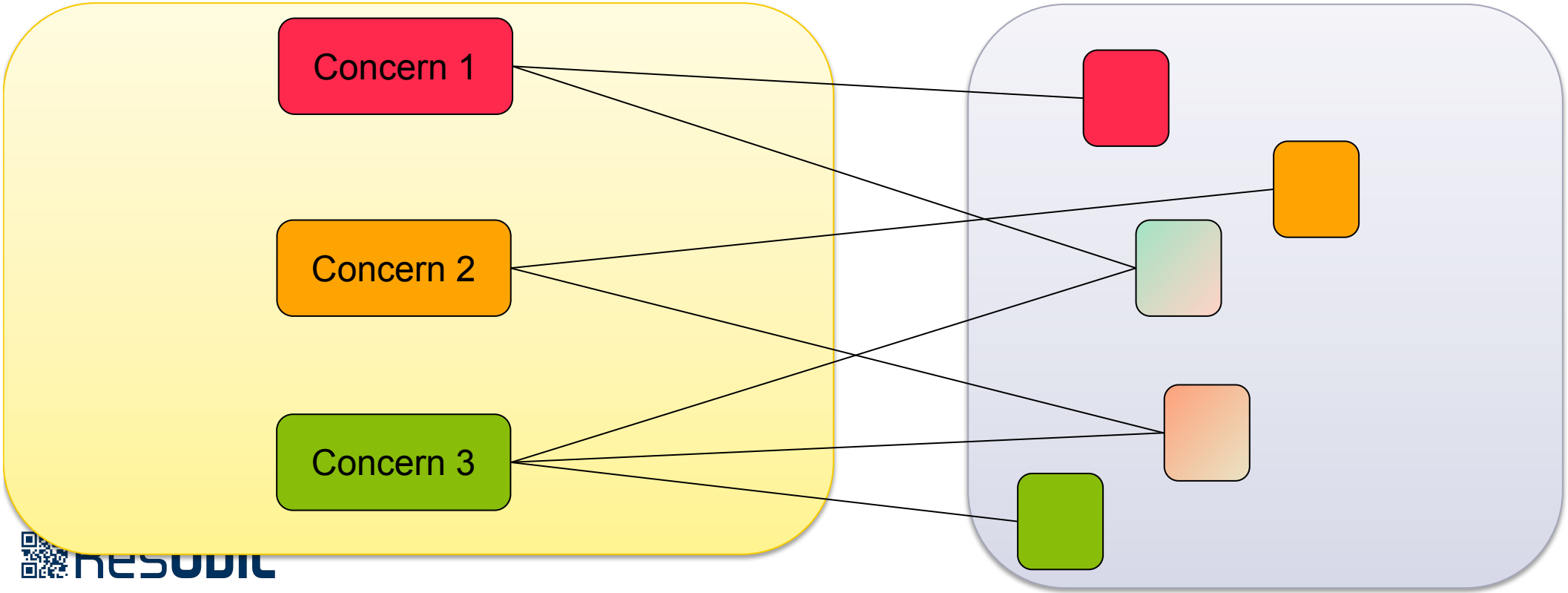


- Perspective models induce systematic aspect-oriented decompositions
  - Hyperspaces
  - Kruchten 4+1 views of software architecture
  - RM-ODP
  - Zachmann framework
- Architctural Decomposition
  - Architecture
  - Application
- Platform-orienied Decomposition in MDA
  - Essenial activities in the PIM
  - Platform 1
  - .. Plattform n

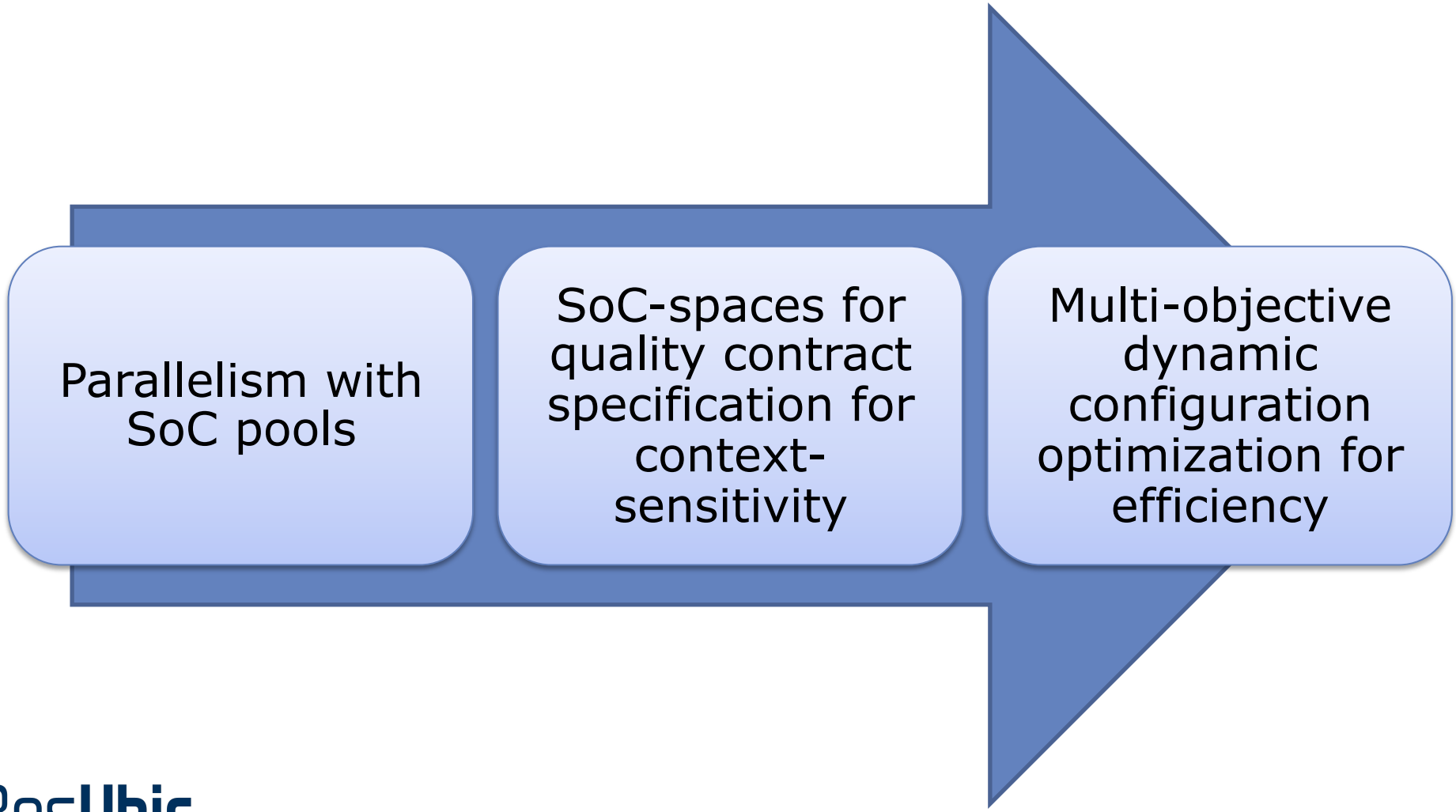
- A **SoC Space** is a triple (C, M, A) with
  - Concern space C
  - Concern mapping M
  - Artefact space A
- Concerns select views for focussing; interpretation by composition



- SoC-space für modules:
  - For all modules: color with a concern
- SoC-space for components:
  - For all components: color with a concern
- SoC-space for arbitrary fragments, such as activities, stores, places
  - For all model elements: color with a concern



- **Ch 1.2: How can we metamodel SoC spaces so that they support composition and decomposition?**



- For CPS, we need Parallel OO Languages

3) PARALLELISM: THE POOL PROBLEM REVISITED

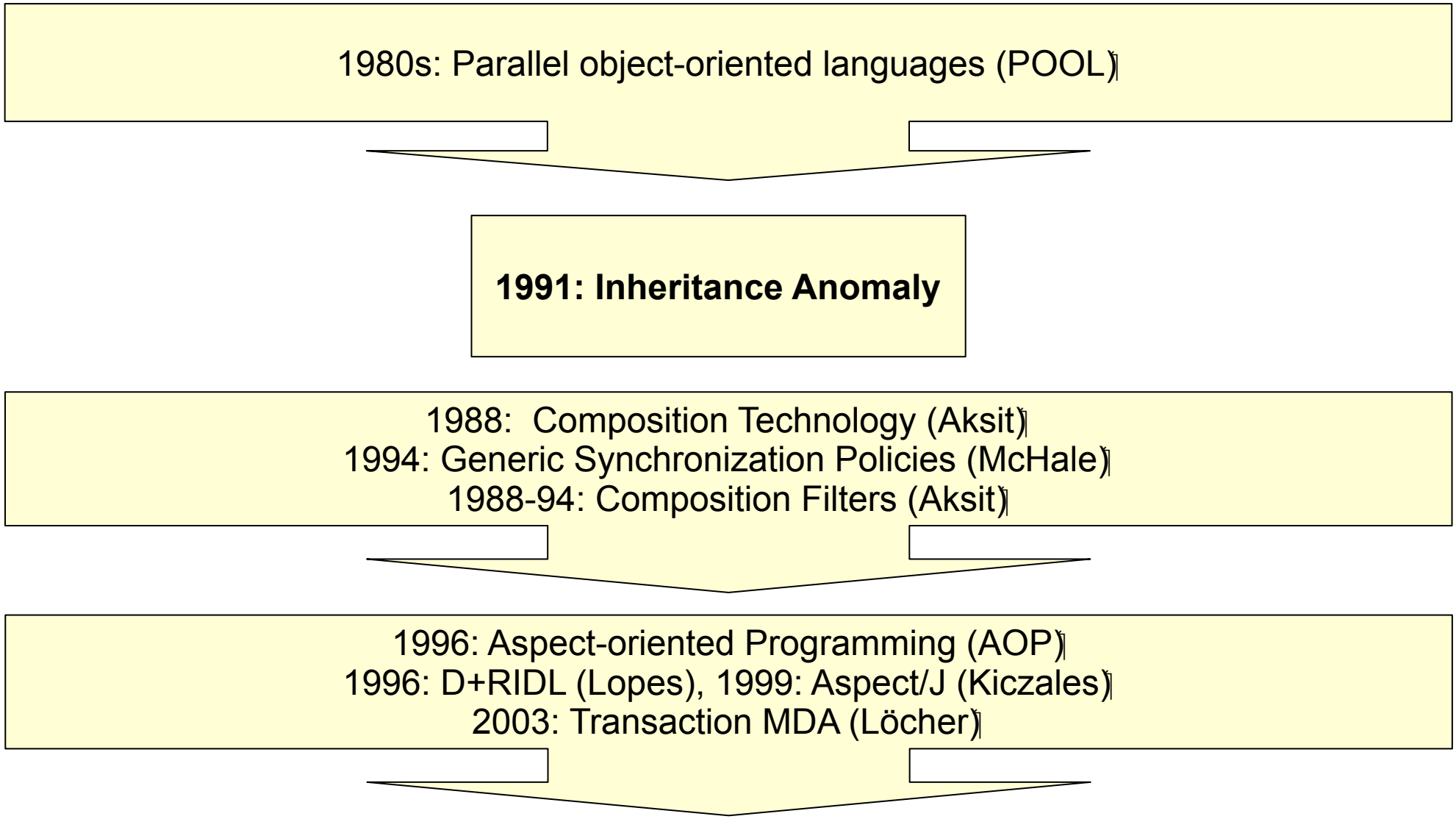
Let me try to explain to you, what to my taste is characteristic for all intelligent thinking.
It is, that one is willing to study in depth an aspect of one's subject matter in isolation for the sake of its own consistency, all the time knowing that one is occupying oneself only with one of the aspects.

It is what I sometimes have called "the separation of concerns" ..

This is what I mean by "focussing one's attention upon some aspect"“

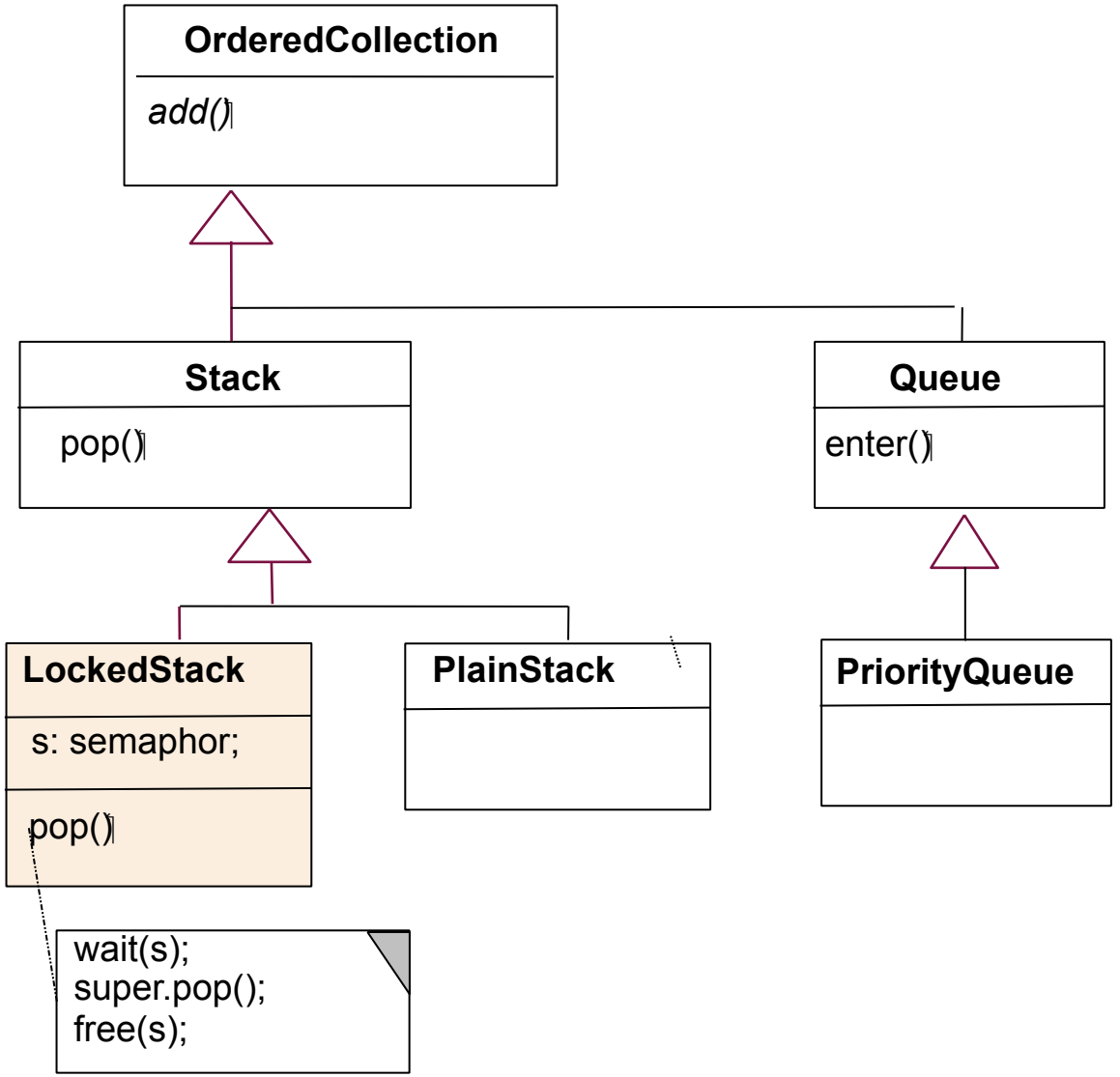
Parallelism results from the fact that all threads can focus on the data of their concern.

A Tale from History: Inheritance Anomaly and how Software Composition Came All Along



Inheritance Anomaly - Why Software Composition Is Necessary

- In a parallel program or library, where should synchronization code be inserted?
  - Stack?
  - Queue?
  - OrderedCollection?
  - Collection?
  - Object?



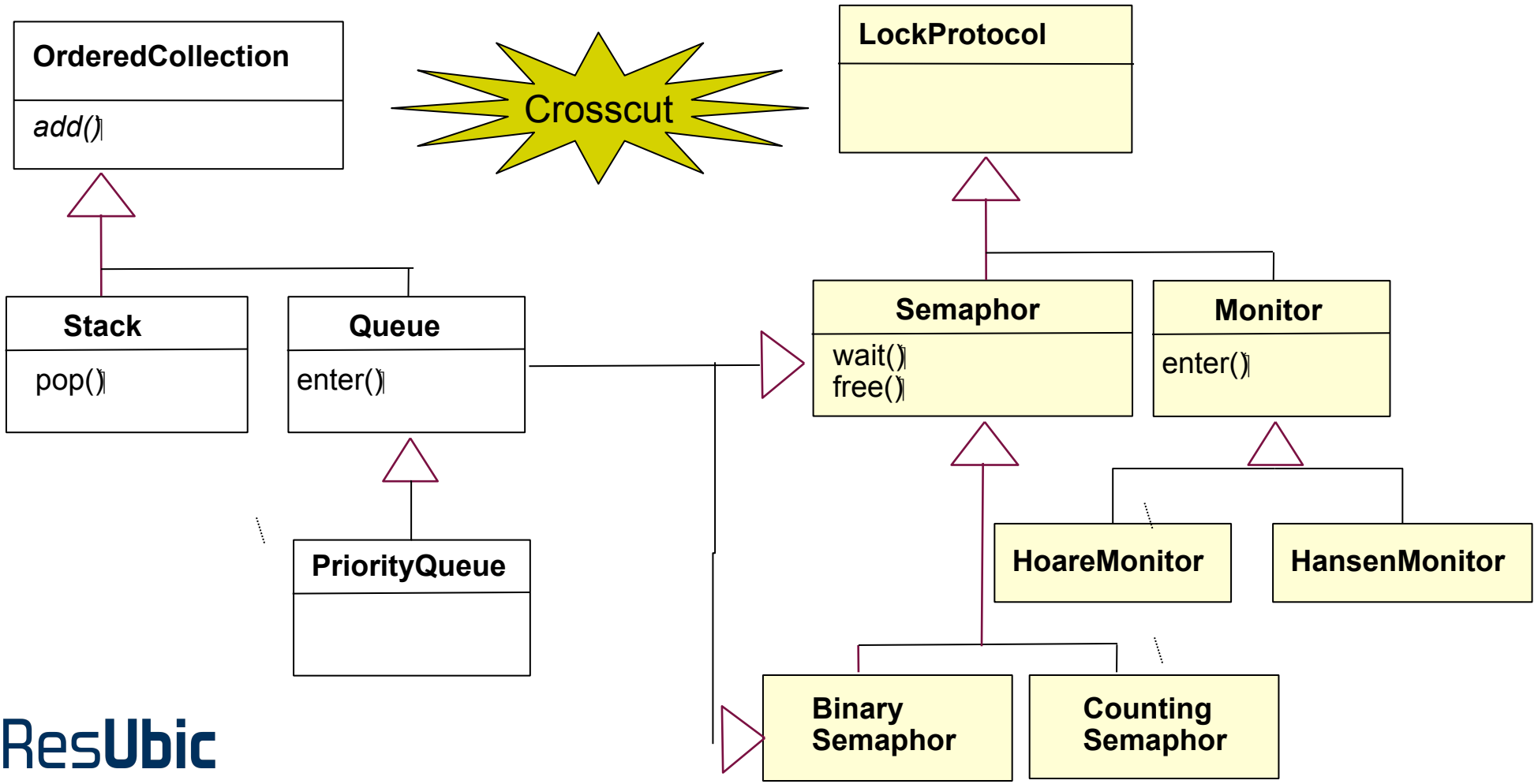
- At the beginning of the 90s, parallel object-oriented languages failed, due to the inheritance anomaly problem

Inheritance anomaly: In inheritance hierarchies, synchronization code is intermingled with the algorithm and cannot be easily exchanged

Synchronization tangling: Because synchronization code *braces* code, it is *tangling*

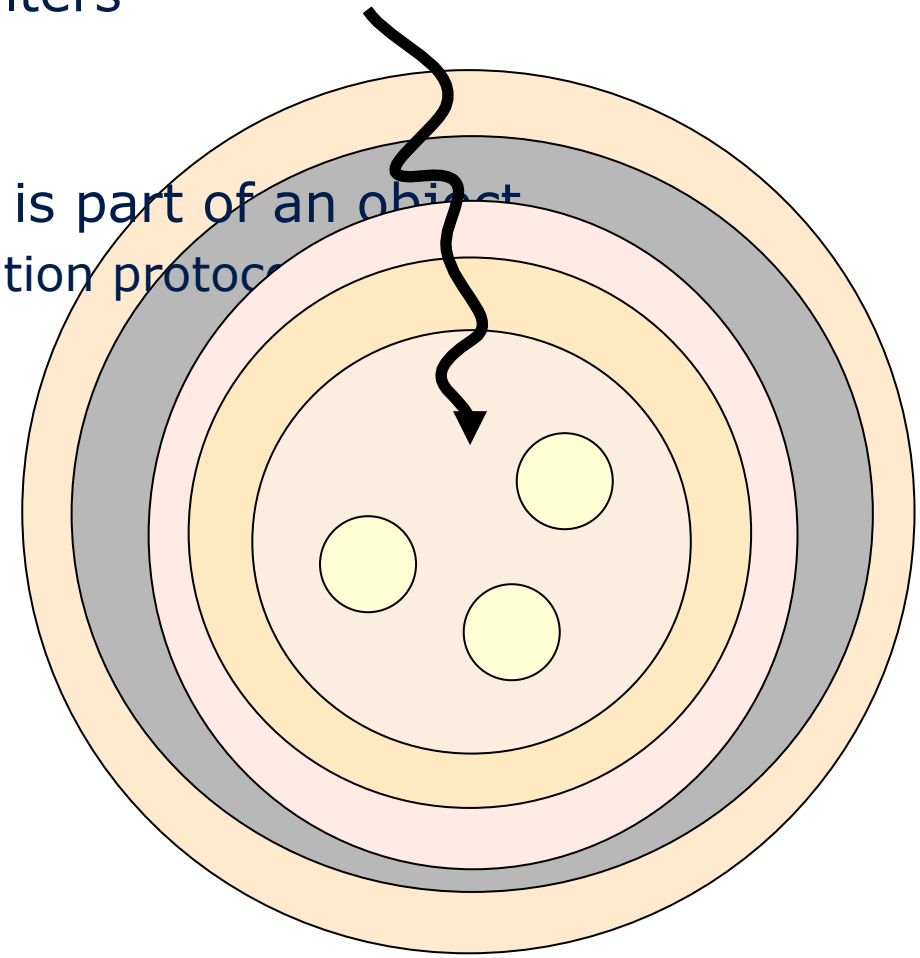
Synchronization crosscut: Because synchronization code *is reused* code, it is *crosscutting*

- Composition fixes crosscut between core and aspect



1988: Composition Filters are Basic Composition Constructs

- Composition Filters (CF) wraps objects with *filters*
- Messages flow through the filters
  - are accepted or rejected
  - are modified
- A filter is an *interceptor* that is part of an object
  - superimposes a synchronization protocol



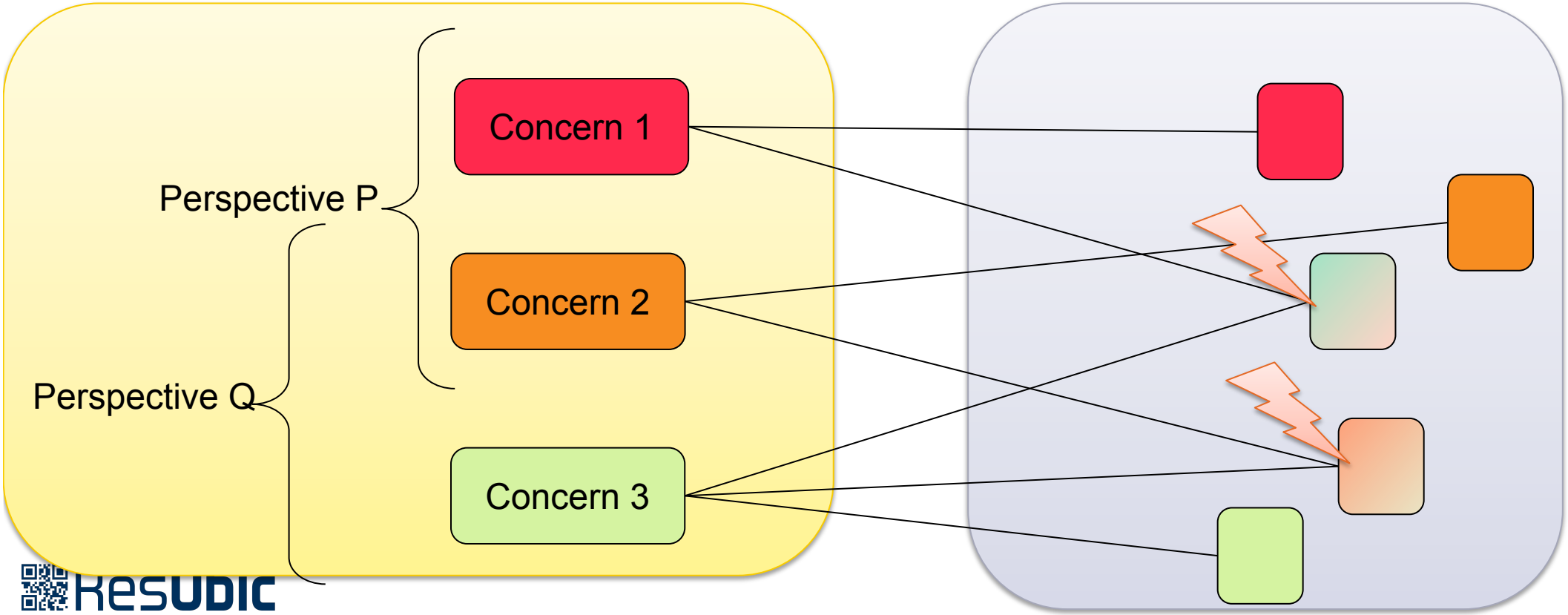
- Inheritance anomaly is solved by aspect composition over crosscut graphs
- But data dependencies are not covered
  - DOALL loops (without memory dependencies and alias-freedom) were the most successful form of parallelization because they map easily to MIMD

The relation of objects to memory is never modeled

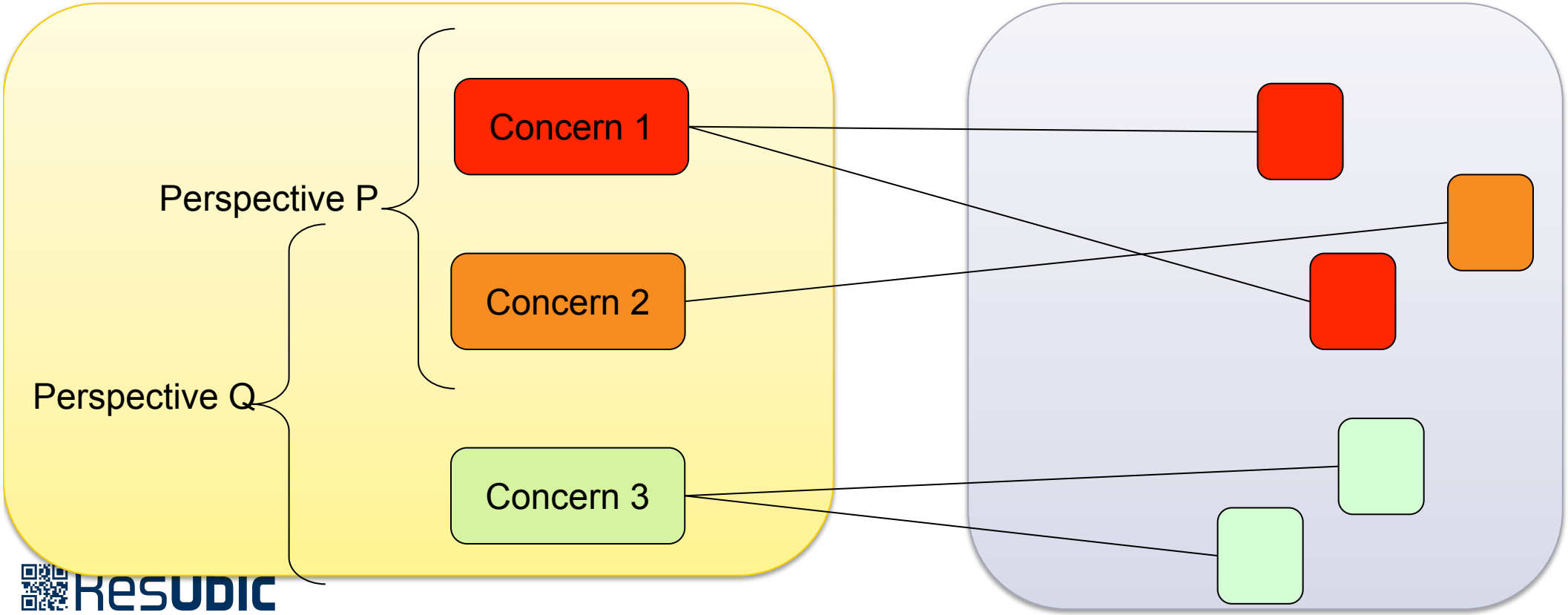
- **Ch 2: How can software composition help to scale parallelism?**

- **Ch 2.1: For a scalable parallism, how can we express alias absence with aspects and crosscutting?**
- **Ch 2.2: Hierarchical objects ensure alias disjointness to others. How to express this?**

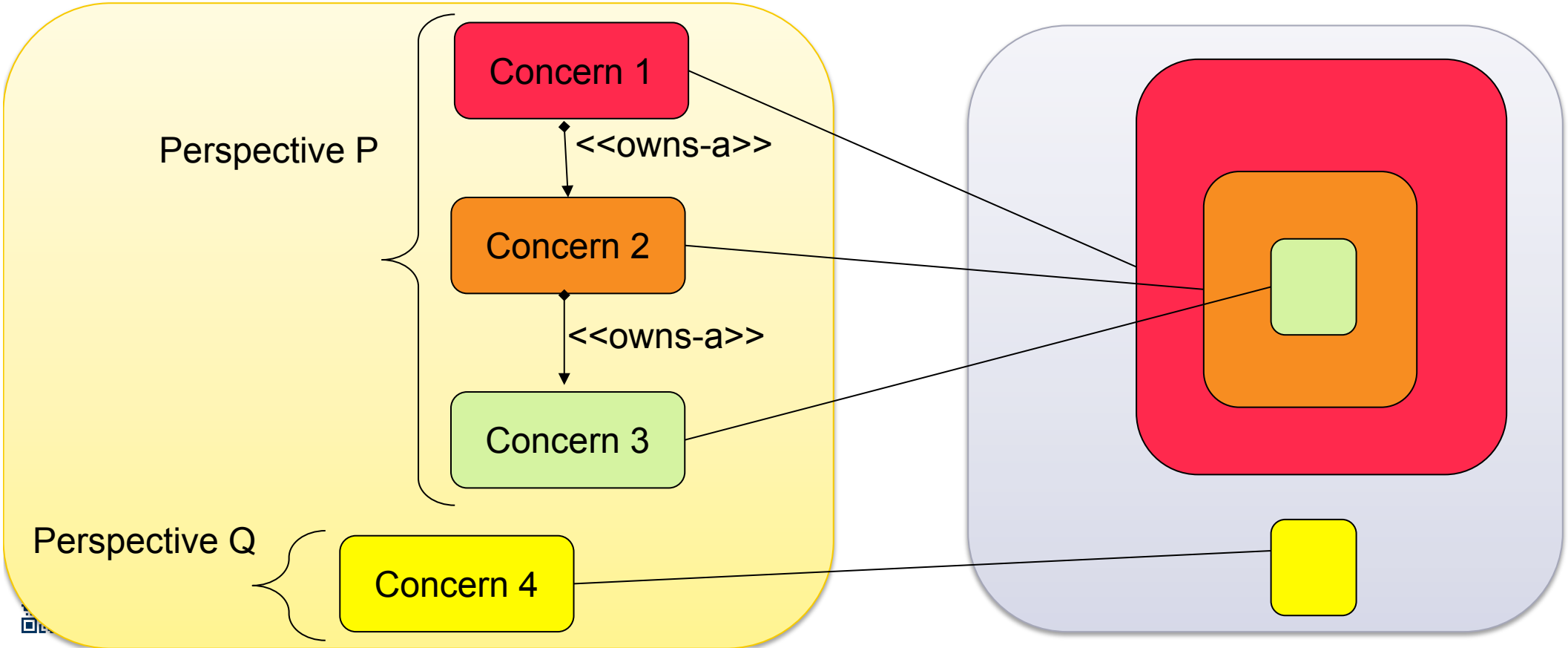
- Non-injectiveness of the concern mapping means aliases and dependencies (overlapping memory slices)



- Injective concern mapping yields alias absence and scalable slice-based parallelism
  - Already in HPF/95 patterns for non-overlapping (alias absense) were introduced for arrays as comment aspects



- Injective mappings of a owns-a concern hierarchy result in large objects without aliases for different perspectives
- (Business objects)



- **Ch 2.3: Which forms of alias-absence can be specified in SoC-Spaces with concern structure and concern mapping?**

- Whenever concern mappings are injective, parallelism scales

- SoC Pools belong to the composition level.
  - They should not be mixed with the algorithmic level.
  - They are independent of the algorithmic language and belong to the composition level
- Don‘t mix it like in classic parallel programming!

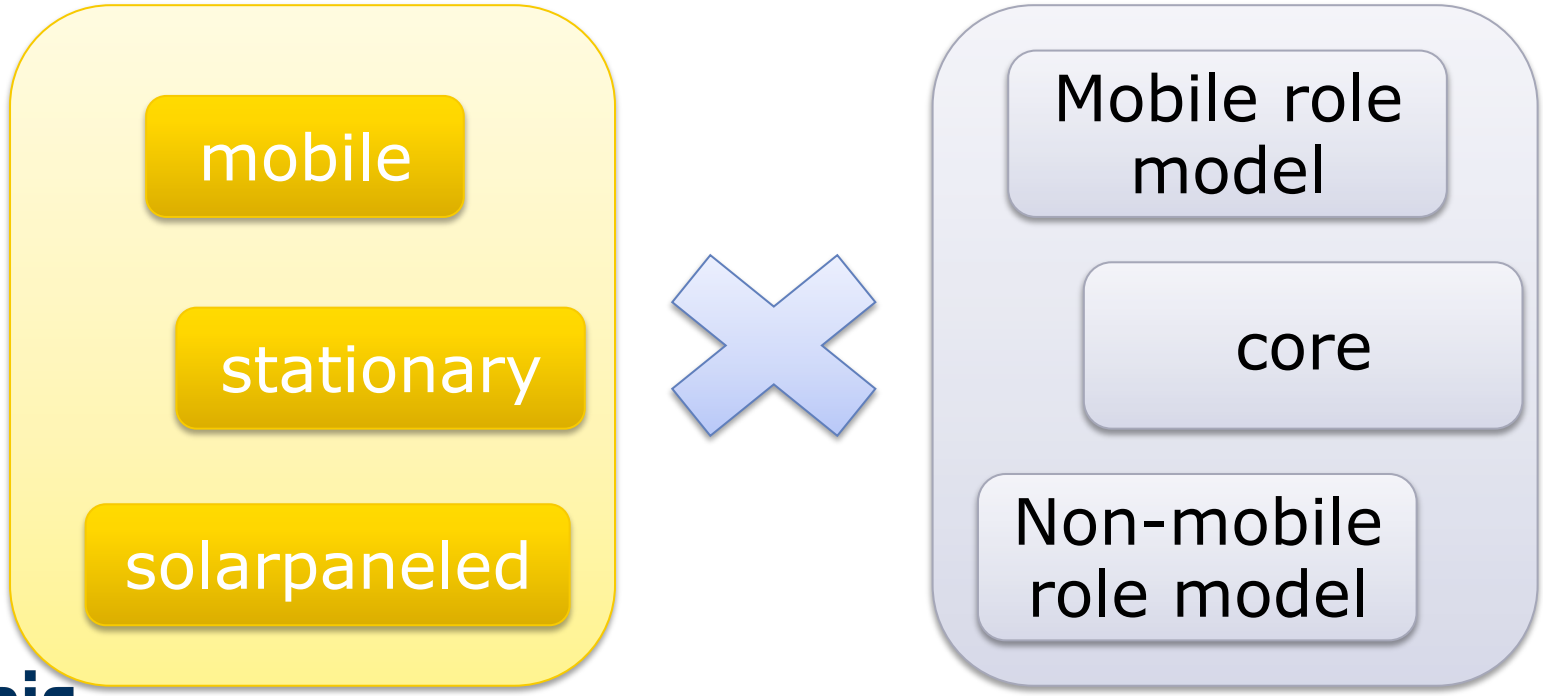
4. CHALLENGES FOR CONTEXT-SENSITIVITY

- Bachman's work (1987) on role models in databases allowed for context-sensitive querying, because roles model context.
- In database graphs, objects carry roles, so contextual querying is very important.

Because the standard data models did not provide roles, the query OO languages failed.

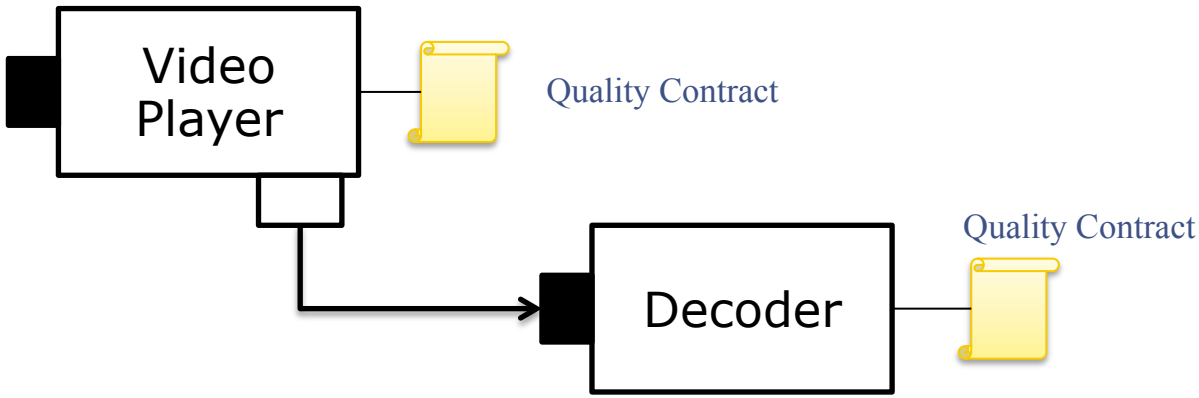

CPS are context-sensitive
• Mobile
• World models
• User profiles
• Energy aware
CPS need roles and context specification
CPS require reliable roles
• Verification
• Adaptivity to contexts

- [Context-aware programming of Hirschfeld et. al.]
- [Context-aware composition (Växjö, Linköping)]
- Roles belong to the artefact space
- Contexts to the concern space
- Switching on and off contexts adapts the application by enabling and disabling role models



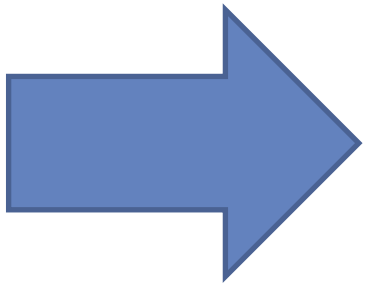
Heterogeneous Rich Components (Damm et.al, 2005) offer Quality Contracts in Compositional Systems

- **Quality contracts** are temporal-logic constracts by which the test coverage can be decided statically. They allow for writing statements over time that are verified
  - „in 5ms the system is in a safe state"
  - „the system is life and will never get stuck"
  - „After the door is closed, the train will eventially run"
  - „This robot cannot be functionally replaced – function of the swarm fails
- Quality contracts assure modular verification of compositions

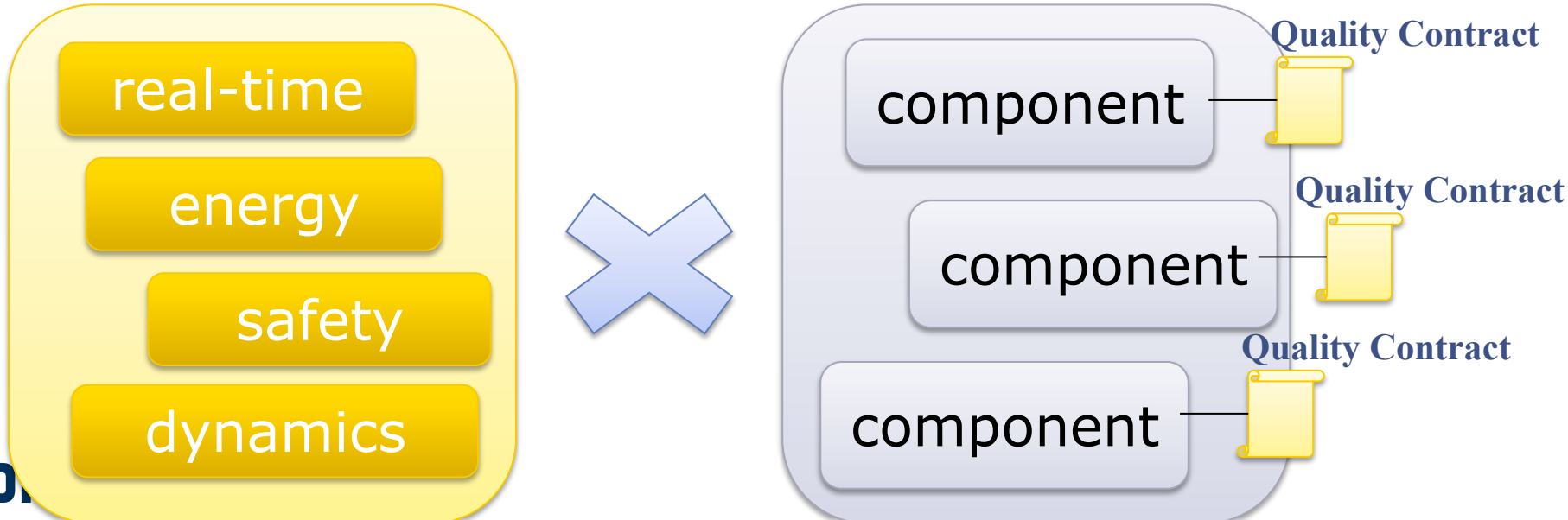


Heterogeneous Rich Components (Damm et.al, 2005) form a Flat SoC Space

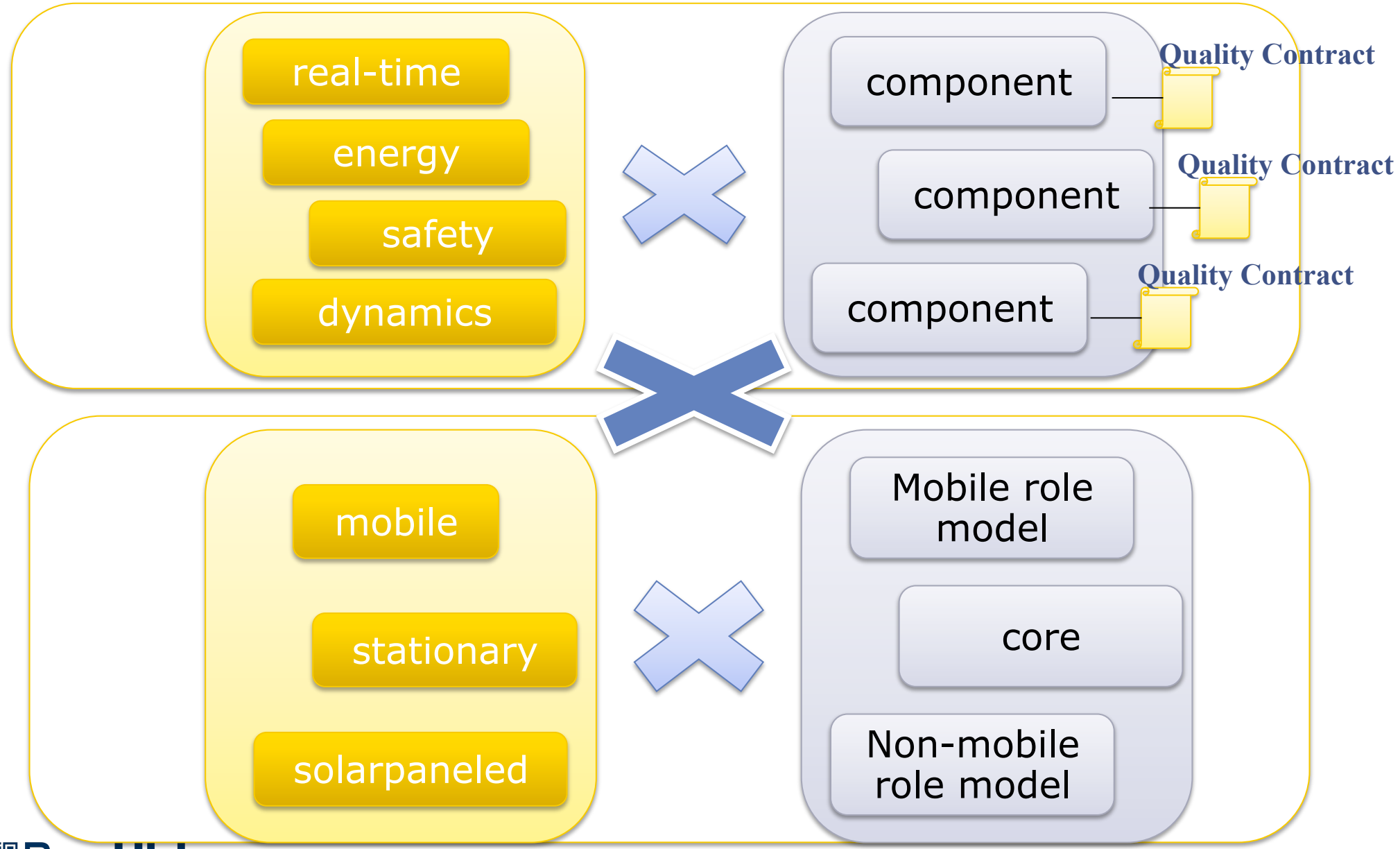
- CPS must be reliably composable
  - Functional Contracts
  - Quality Contracts
    - Real-time contracts
    - Energy contracts
    - Safety contracts
    - Spatial dynamics contracts



- Contract Checking of Quality Properties, e.g.
- Energy reserve for active fail-safety (AFS)
- „The robot swarm can still run 25s, so the shutdown to safe state has start in 5s“

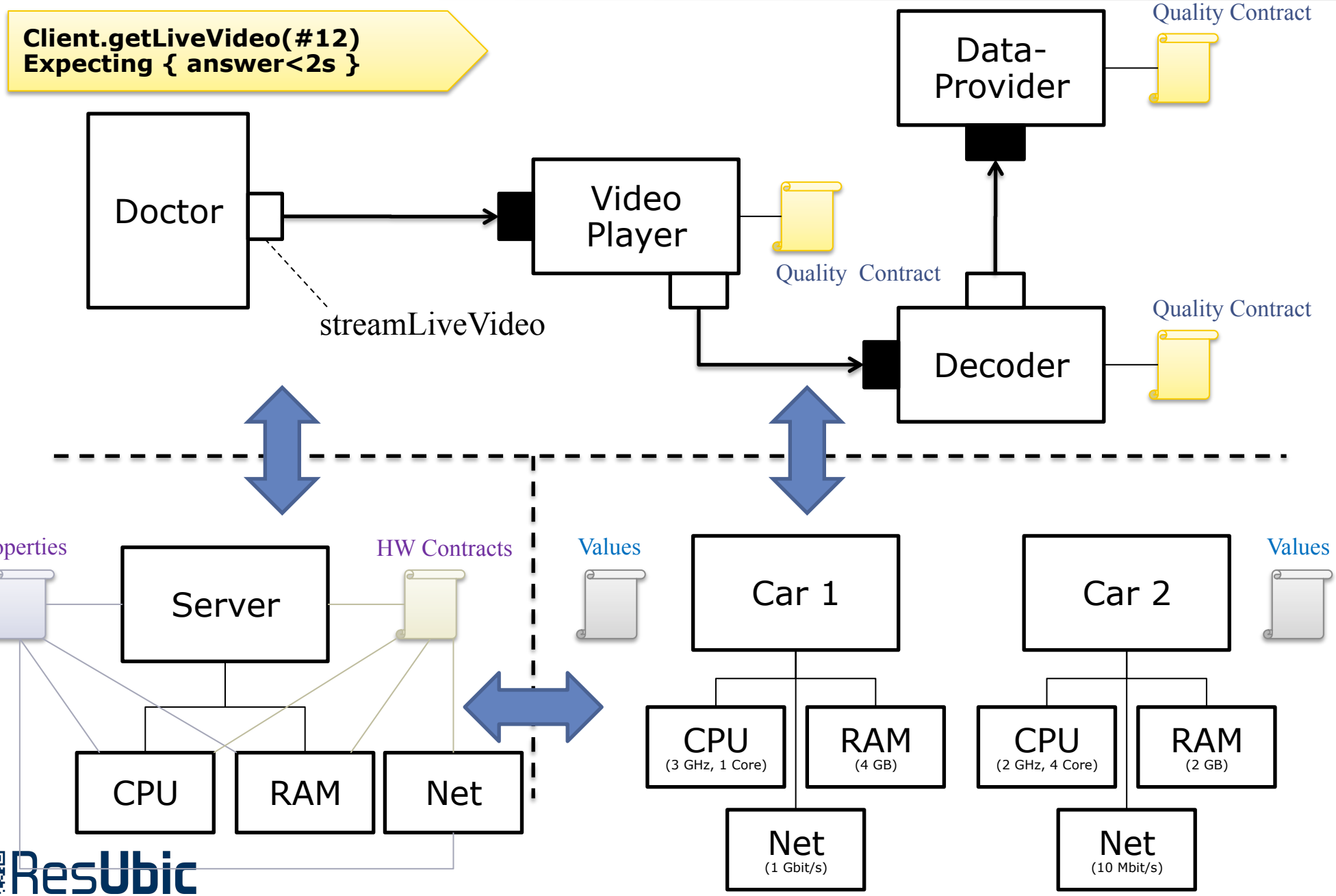


Reliability needs Quality Contracts for Role Models

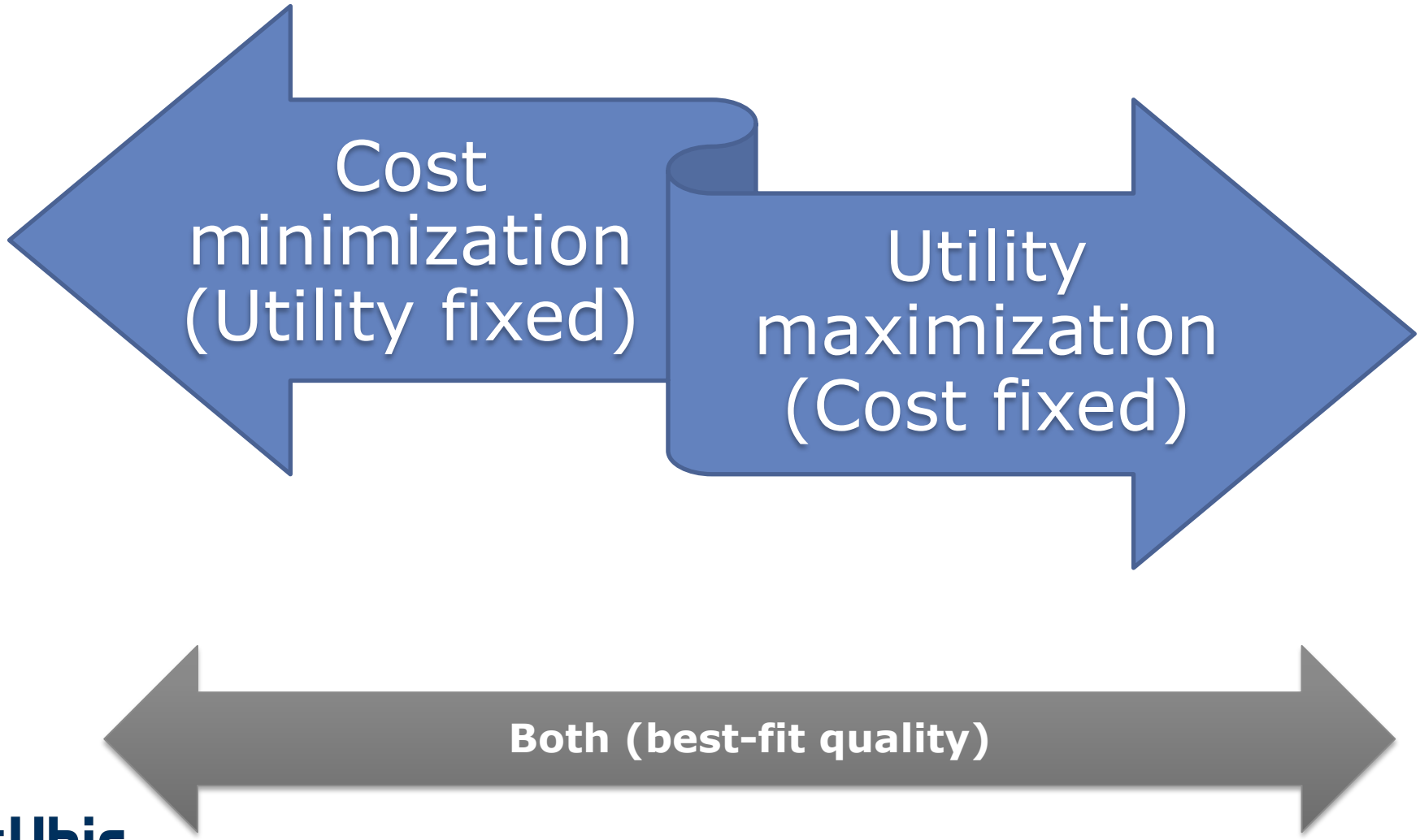


- **Ch 3: How can role models and contextual programming be combined with quality contracts?**
- **Ch 3.1 How can SoC spaces be applied systematically to reliable context-sensitivity?**

5. EFFICIENCY-BASED DEVELOPMENT WITH MULTI-OBJECTIVE OPTIMIZED ARCHITECTURES (MOO)

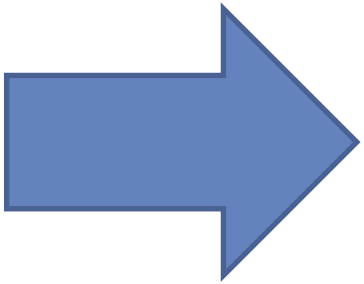

Client.getLiveVideo(#12)
Expecting { answer<2s }
Data-Provider
Quality Contract
Doctor
Video Player
Quality Contract
streamLiveVideo
Decoder
Quality Contract
Properties
Server
HW Contracts
Values
Car 1
Car 2
Values
CPU
RAM
Net
CPU
(3 GHz, 1 Core)
RAM
(4 GB)
Net
(1 Gbit/s)
CPU
(2 GHz, 4 Core)
RAM
(2 GB)
Net
(10 Mbit/s)

- Configuration is here a special form of composition
- All efficiency requirements can be handled



Multi-Objective Optimization (MOO) of Configurations

- CPS need adaptability
  - QoS adaptation
  - Context adaptation

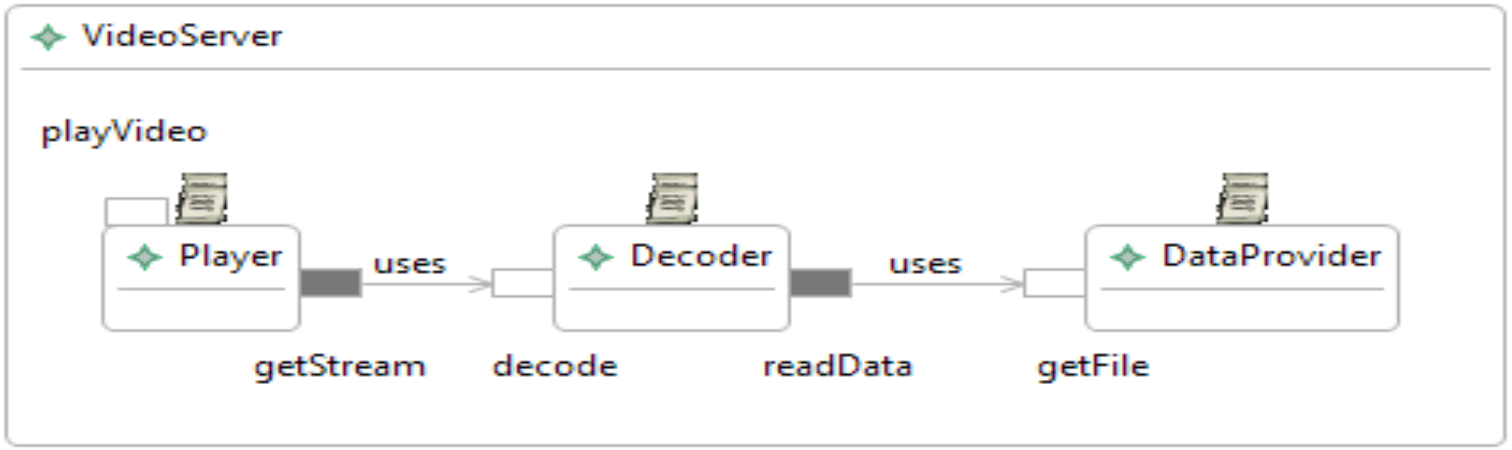


- Configuration Optimization
  - Determining Optimal Mappings of Software Components onto Hardware Resources
  - Resource minimization (energy)
  - Computation of pricing

- **Ch 4: How can we reconfigure architectures with different cost and utility criteria (efficiency)?**

- **Ch 4.1: Can we integrate energy consumption as resource?**
- **Ch 4.2: Can we optimize with regard to multiple objectives?**

- Software is described using a CCM Structural Model
  - approx. SysML plus QCL contract language



- SW Components have provided/required ports (e.g. methods of classes)
- Quality-Modes and Implementations of SW Components are described using QCL contracts

VideoServer_Player.ecl

```
1  import ccm [../VideoServer.structure]
2  import ccm [../Server.structure]
3
4  contract VLC implements software Player {
5      mode highQuality {
6          requires component Decoder {
7              bitrate min: 5
8          }
9          requires resource CPU {frequency min: 1500}
10
11         provides framerate min: 20
12     }
13     mode lowQuality {
14         requires component Decoder {
15             bitrate min: 2
16         }
17         requires resource CPU {frequency
18
19         provides framerate min: 10
20     }
21 }
```

Quality Modes

- SW Dependencies
- HW Dependencies
- Provided Properties

Where do the properties come from?

Where and how to define HW ?

VideoServer_Player.ecl | VideoServer_DataProvider.ecl

```
1  import ccm [../VideoServer.structure]
2  import ccm [../Server.structure]
3
4  contract FileReader implements software DataProvider {
5      mode file {
6          provides throughput min: 20
7          requires resource HDD {
8              throughput min: 20
9          }
10     }
11 }
12
13 contract URLReader implements software DataProvider {
14     mode url {
15         provides throughput min: 5
16         requires resource Network {
17             bandwidth min: 5
18         }
19     }
20 }
```

Efficiency Challenge:

- Which implementations of the quality contracts
- in which quality mode
- on which resources
- give the most efficient configuration?

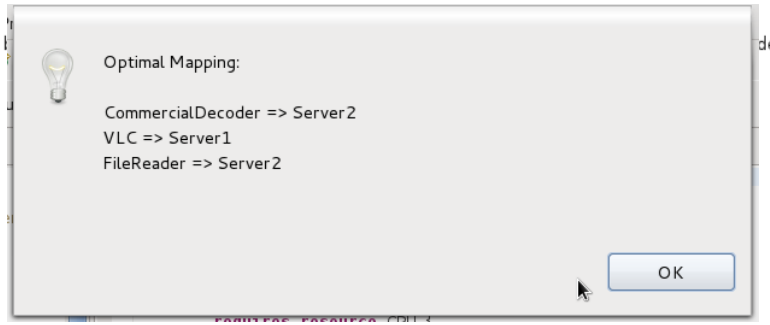
Multi-Objective Optimization (MOO) with:

- Integer Linear Programming (exact)
- Pseudo-Boolean Optimization (exact)
- Ant Colony Optimization (approx.)
- Simulated Annealing (approx.)
- ...

- Solution of the ILP gives an optimal configuration

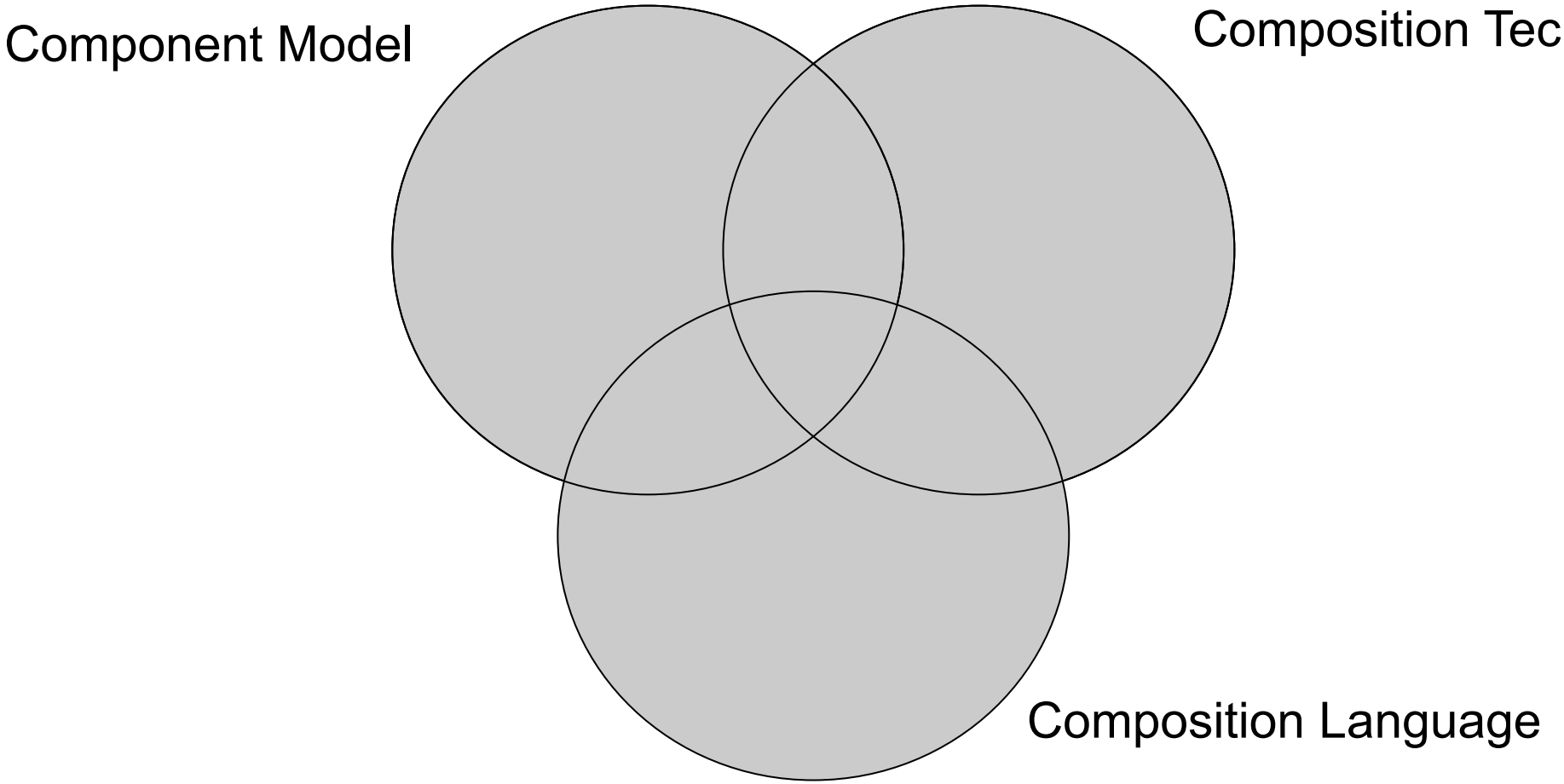
VideoServer_Decoder.ecl | player.request.lp

```
min: usage#Server1#CPU_[s1]#frequency + usage#Server2#CPU#frequency + usage#Server1#Net_[s1]#bandwidth + usage#Server1#HDD_[s1]#size + usage#Server2#Network#bar

b#VLC#highQuality#Server2 + b#VLC#highQuality#Server1 + b#VLC#lowQuality#Server2 + b#VLC#lowQuality#Server1 = 1;
b#FreeDecoder#fast#Server2 + b#FreeDecoder#fast#Server1 + b#FreeDecoder#slow#Server2 + b#FreeDecoder#slow#Server1 + b#CommercialDecoder#ultrafast#Server2 + b#Cc
b#FileReader#file#Server2 + b#FileReader#file#Server1 + b#URLReader#url#Server2 + b#URLReader#url#Server1 = 1;

usage#Server1#CPU_[s1]#frequency <= 2000.0;
usage#Server1#CPU_[s1]#frequency >= 0;
usage#Server1#CPU_[s1]#frequency = 1500.0 b#VLC#highQuality#Server1 + 1500.0 b#FreeDecoder#fast#Server1 + 1000.0 b#FreeDecoder#slow#Server1 + 800.0 b#Commercial
usage#Server1#RAM_[s1]#size <= 512.0;
usage#Server1#RAM_[s1]#size >= 0;
usage#Server1#RAM_[s1]#size = 512.0 b#FreeDecoder#fast#Server1 + 256.0 b#FreeDecoder#slow#Server1 + 128.0 b#CommercialDecoder#slow#Server1 + 512.0 b#CommercialD
usage#Server1#Net_[s1]#bandwidth <= 10.0;
usage#Server1#Net_[s1]#bandwidth >= 0;
usage#Server1#Net_[s1]#bandwidth = 5.0 b#URLReader#url#Server1;
usage#Server1#HDD_[s1]#size <= 500.0;
usage#Server1#HDD_[s1]#size >= 0;
usage#Server1#HDD_[s1]#throughput_w <= 5.0;
usage#Server1#HDD_[s1]#throughput_w >= 0;
usage#Server2#CPU#frequency <= 1000.0;
usage#Server2#CPU#frequency >= 0;
usage#Server2#CPU#frequency = 1500.0 b#VLC#highQuality#Server2 + 1500.0 b#FreeDecoder#fast#Server2 + 1000.0 b#FreeDecoder#slow#Server2 + 800.0 b#CommercialDecoc
usage#Server2#RAM#size <= 2048.0;
usage#Server2#RAM#size >= 0;
usage#Server2#RAM#size = 512.0 b#FreeDecoder#fast#Server2 + 256.0 b#FreeDecoder#slow#Server2 + 128.0 b#CommercialDecoder#slow#Server2 + 512.0 b#CommercialDecode
usage#Server2#Network#bandwidth <= 100.0;
usage#Server2#Network#bandwidth >= 0;
usage#Server2#Network#bandwidth = 5.0 b#URLReader#url#Server2;
usage#Server2#HDD#size <= 150.0;
usage#Server2#HDD#size >= 0;
usage#Server2#HDD#throughput_w <= 25.0;
usage#Server2#HDD#throughput_w >= 0;
bitrate = 10.0 b#CommercialDecoder#fast#Server1 + 8.0 b#FreeDecoder#fast#Server2 + 3.0 b#FreeDecoder#slow#Server2 + 10.0 b#CommercialDecoder#fast#Server2 + 3.0
throughput = 5.0 b#URLReader#url#Server1 + 20.0 b#FileReader#file#Server2 + 5.0 b#URLReader#url#Server2 + 20.0 b#FileReader#file#Server1;
framerate = 10.0 b#VLC#lowQuality#Server1 + 10.0 b#VLC#lowQuality#Server2 + 20.0 b#VLC#highQuality#Server2 + 20.0 b#VLC#highQuality#Server1;
bitrate >= 2.0 b#VLC#lowQuality#Server1 + 2.0 b#VLC#lowQuality#Server2 + 5.0 b#VLC#highQuality#Server2 + 5.0 b#VLC#highQuality#Server1;
throughput >= 5.0 b#CommercialDecoder#fast#Server1 + 5.0 b#FreeDecoder#fast#Server2 + 2.0 b#FreeDecoder#slow#Server2 + 5.0 b#CommercialDecoder#fast#Server2 + 2.

framerate >= 20.0;

binary b#CommercialDecoder#fast#Server1, b
```



Prof. U. Aßmann, TU Dresden

6. THE ULTIMATE CHALLENGE: SPECIFYING COMPOSITION SYSTEMS


Component Model
Composition Technique
Composition Language

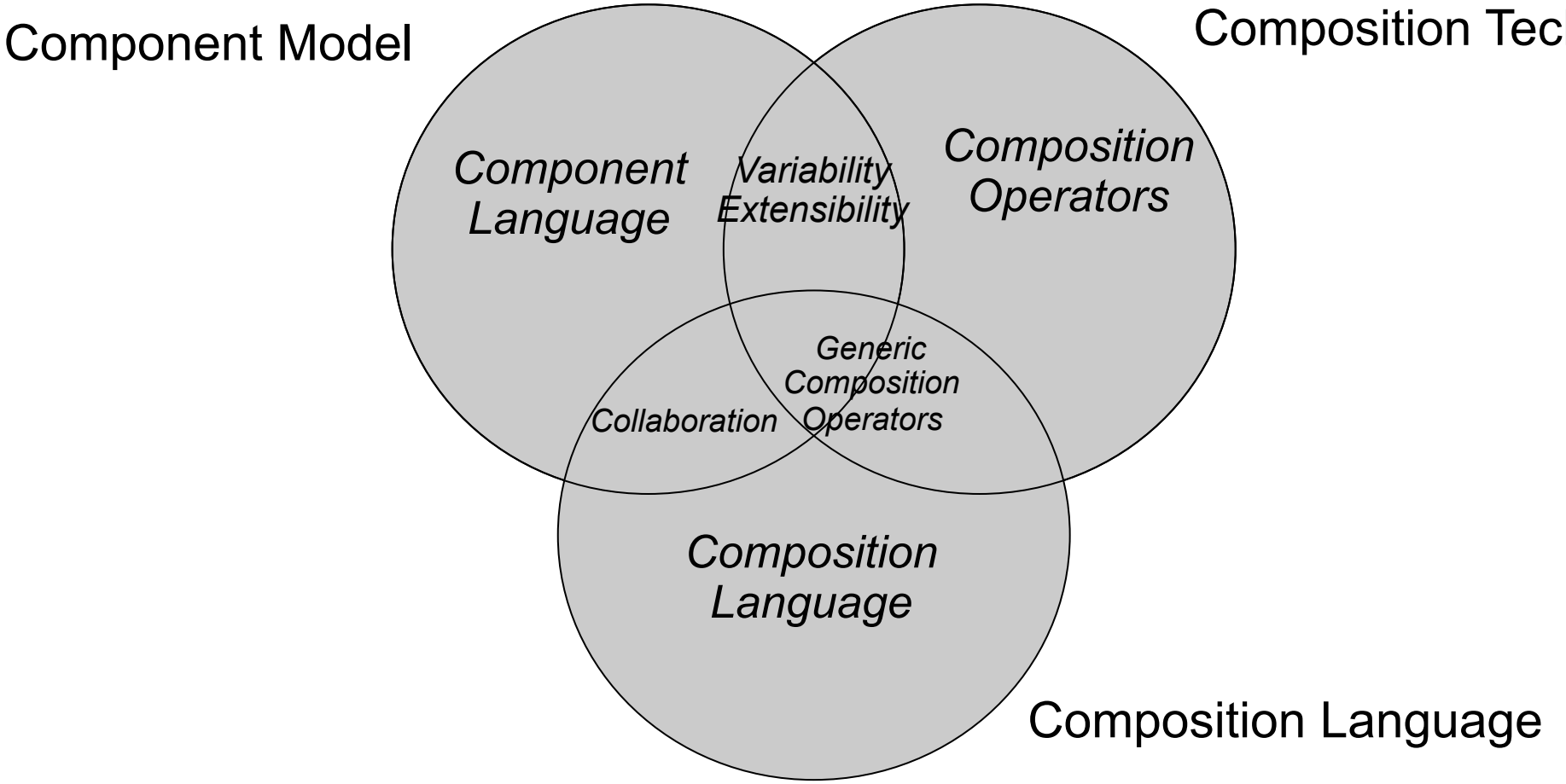
All issues above do not belong to the component or algorithmic level, but to the composition level:

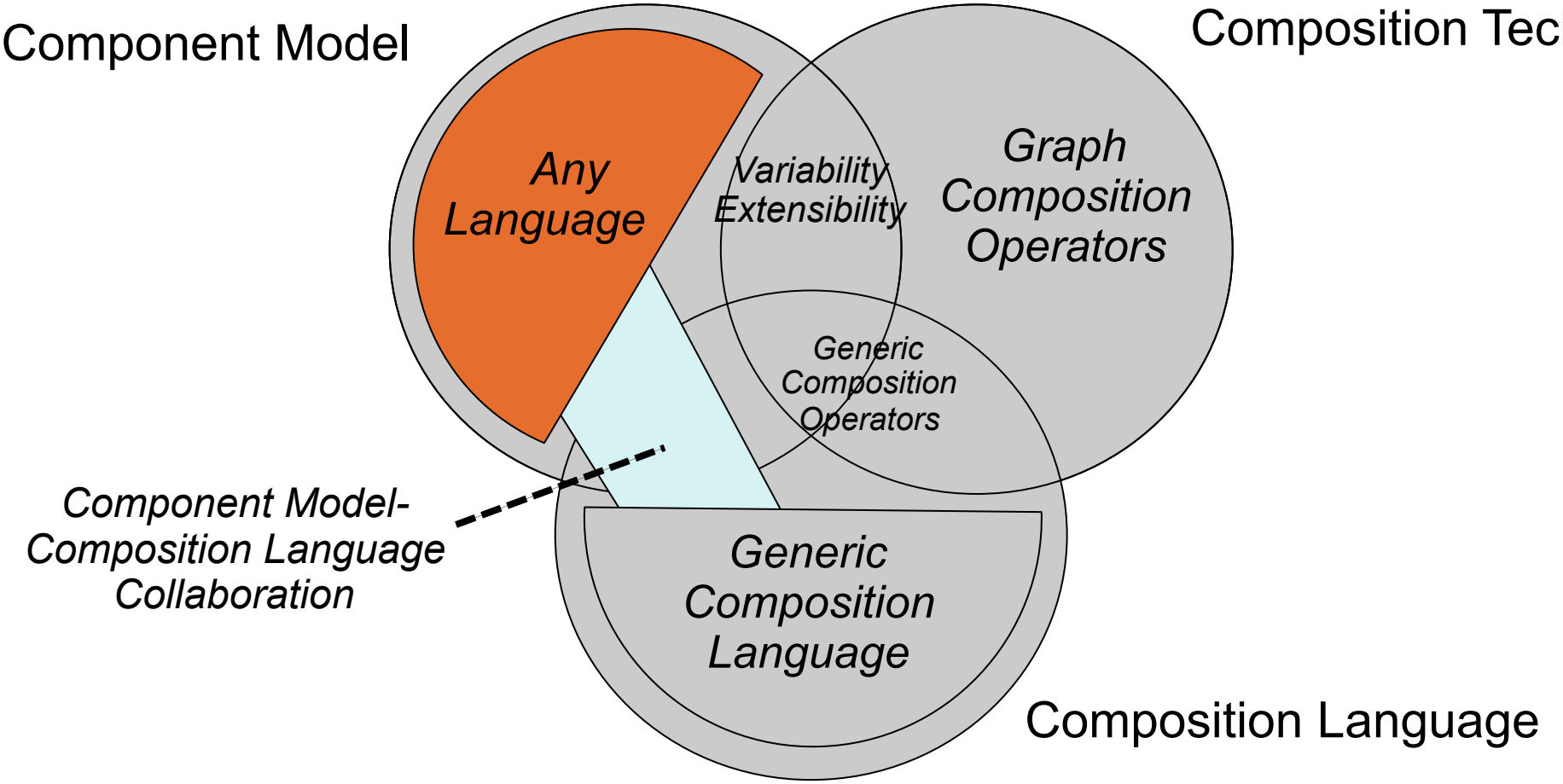
- SoC spaces
  - Hierarchical pools
  - SoC Pools
  - Context specifications
- Quality contracts
- Efficiency-based development with MOO

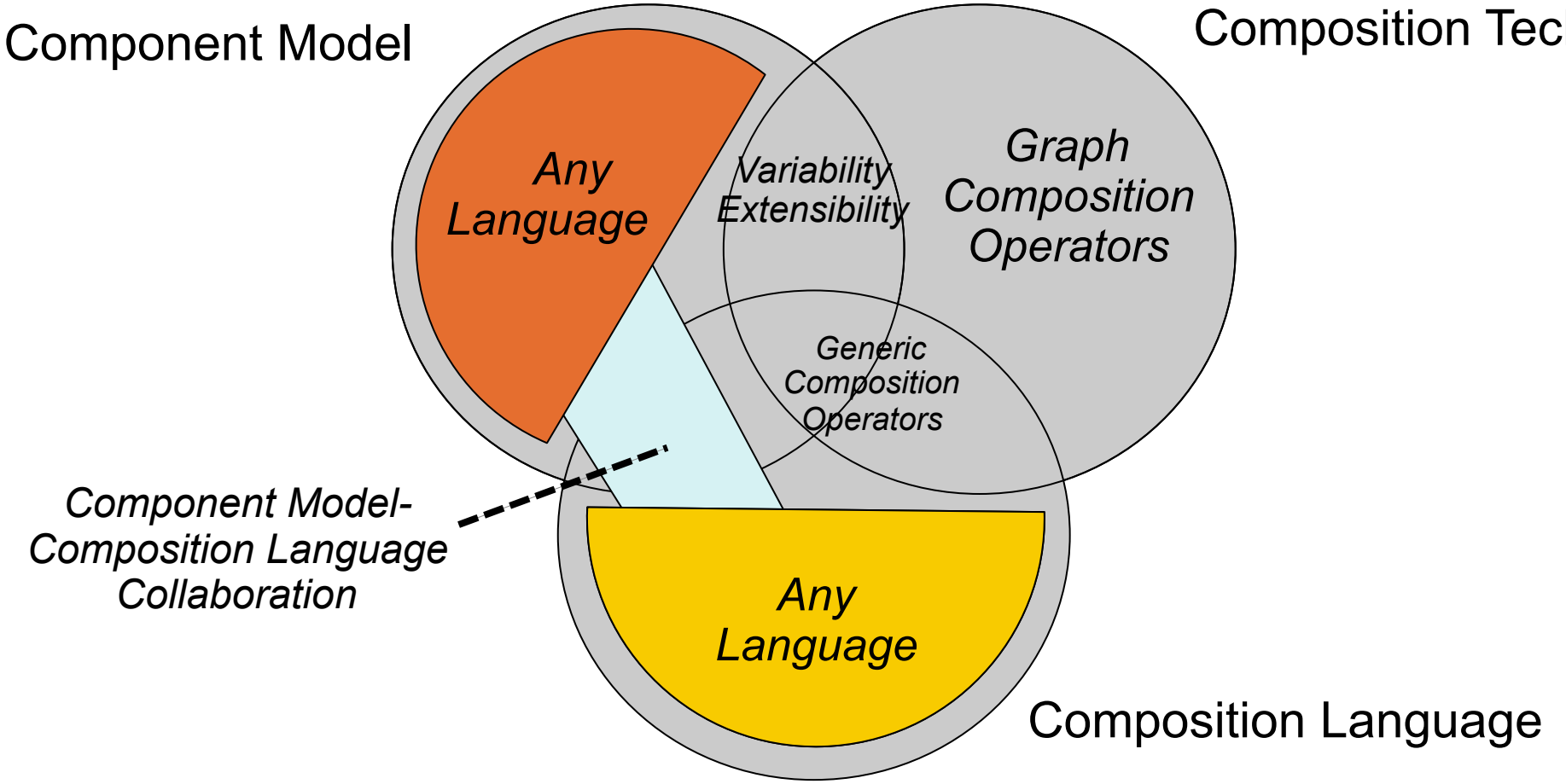
We want to reuse composition languages for **any** component language

- Mixing composition level and component level has a lot of disadvantages:
  - Composition language cannot be reused for several algorithmic languages
  - Tools, editors, compilers, checkers, cannot be reused, but have to be constructed

Composition languages should be separate (Principle of separate composition)


Component Model
Composition Technique
Component Language
Variability Extensibility
Composition Operators
Generic Composition Operators
Collaboration
Composition Language
Composition Language

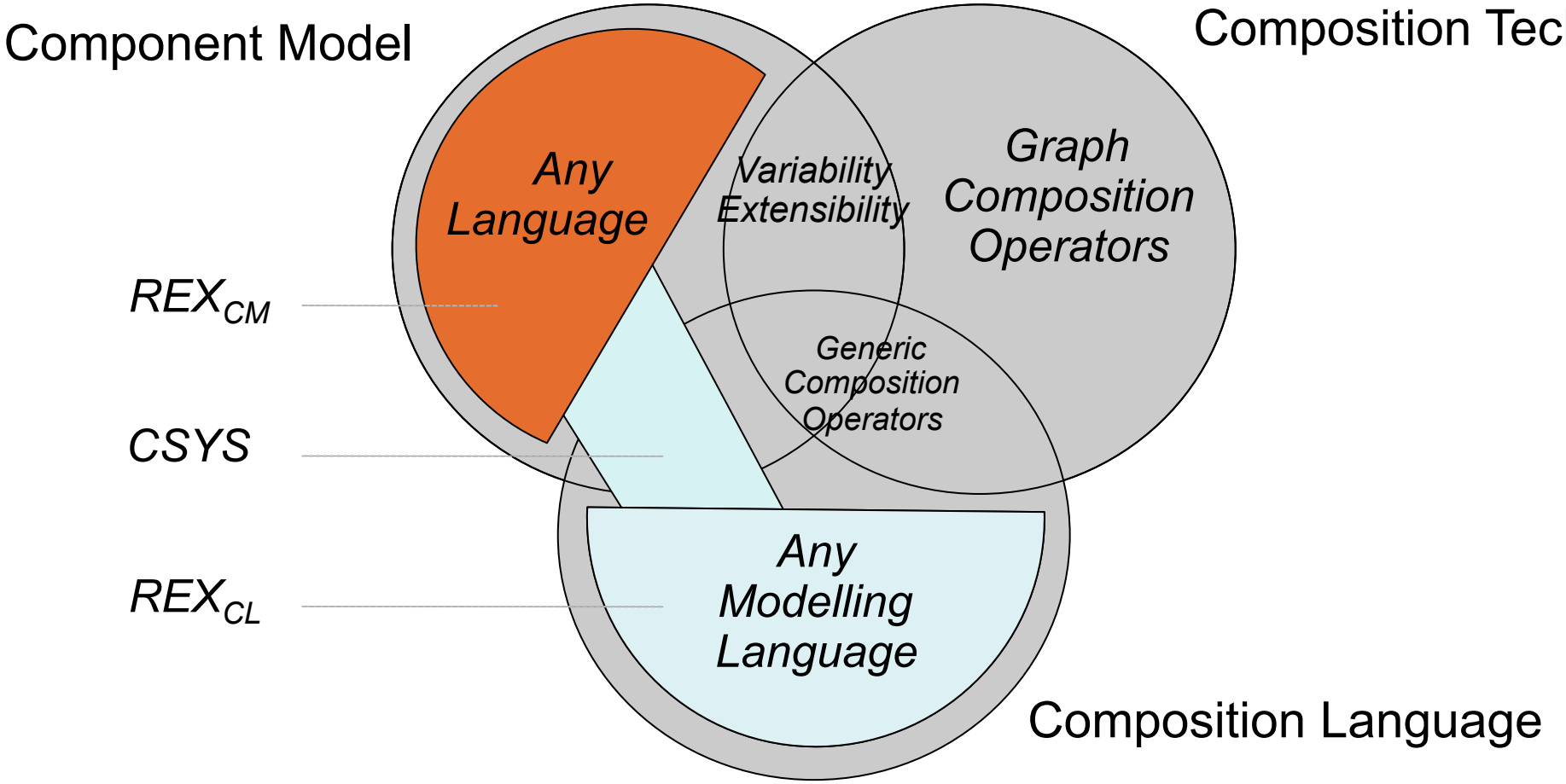

Component Model
Composition Technique
Any Language
Variability Extensibility
Graph Composition Operators
Generic Composition Operators
Component Model-Composition Language Collaboration
Generic Composition Language
Composition Language


Component Model
Composition Technique
Any Language
Variability Extensibility
Graph Composition Operators
Generic Composition Operators
Component Model-Composition Language Collaboration
Any Language
Composition Language

- **Ch 5: How do we specify composition systems?**

- **Ch 5.1: How can composition languages be separated from components (separate composition)?**
- **Ch 5.2: In-line compositions in components (write-easy) must be round-tripped with off-line, separate compositions (read-easy).**

Ex.: Reuseware: a Composition Meta-System



[Johannes 2010]

Life-by-Wire based on CPS will only be possible

if software composition answers the following research questions:

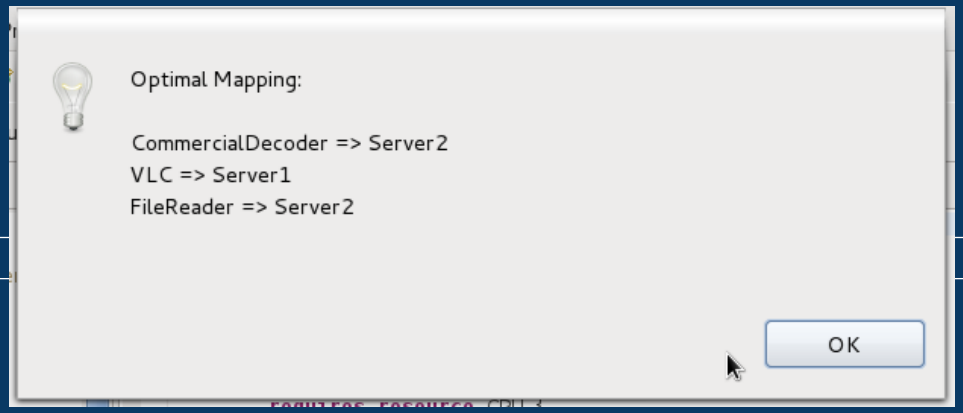
- **Ch 1: How can we conceptualize separations of concerns so that decomposition and composition are supported?**
- **Ch 2: How can software composition help to scale parallelism?**
- **Ch 3: How can role models and contextual programming be combined with quality contracts?**
- **Ch 4: How can we reconfigure architectures with different cost and utility criteria (efficiency)?**
- **Ch 5: How do we specify composition systems?**

- Jendrik Johannes and Uwe Aßmann. Concern-based (de)composition of model-driven software development processes. In Dorina C. Petriu, Nicolas Rouquette, and Øystein Haugen, editors, MoDELS (2), volume 6395 of LNCS, pages 47-62. Springer, 2010.
- Jendrik Johannes. Component-Based Model-Driven Software Development. PhD thesis, Dresden University of Technology, December 2010.
- Christian Wende. Language Family Engineering. PhD thesis, Dresden University of Technology, March 2012.
- Sebastian Götz, Claas Wilke, Sebastian Cech, and Uwe Aßmann. Sustainable Green Computing: Practices, Methodologies and Technologies, chapter Architecture and Mechanisms for Energy Auto Tuning. IGI Global, 2011.
- [Acatech] Agenda CPS. Acatech Studie März 2012.

Contact Info

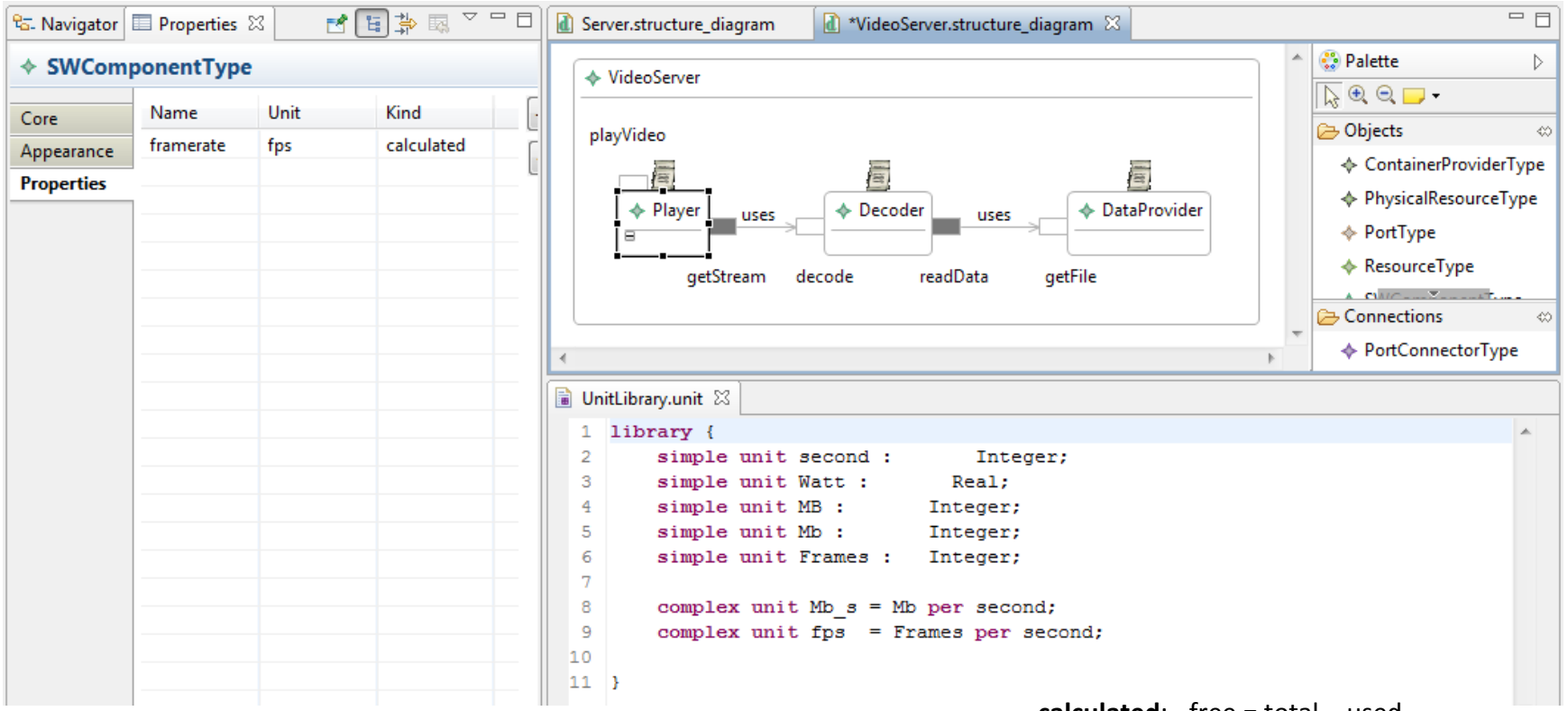
- http://www.resubic.org
- Uwe.assmann@tu-dresden.de
- Sebastian.goetz@tu-dresden.de
- Claas.wilke@tu-dresden.de
- Sebastian.cech@tu-dresden.de
- www.devboost.org
- www.reuseware.org

TECHNISCHE UNIVERSITÄT DRESDEN



Backup. The Software for Life-by-Wire: Multi-Objective Optimization Architectures (MOOA)

Prof. Uwe Aßmann
Sebastian Götz

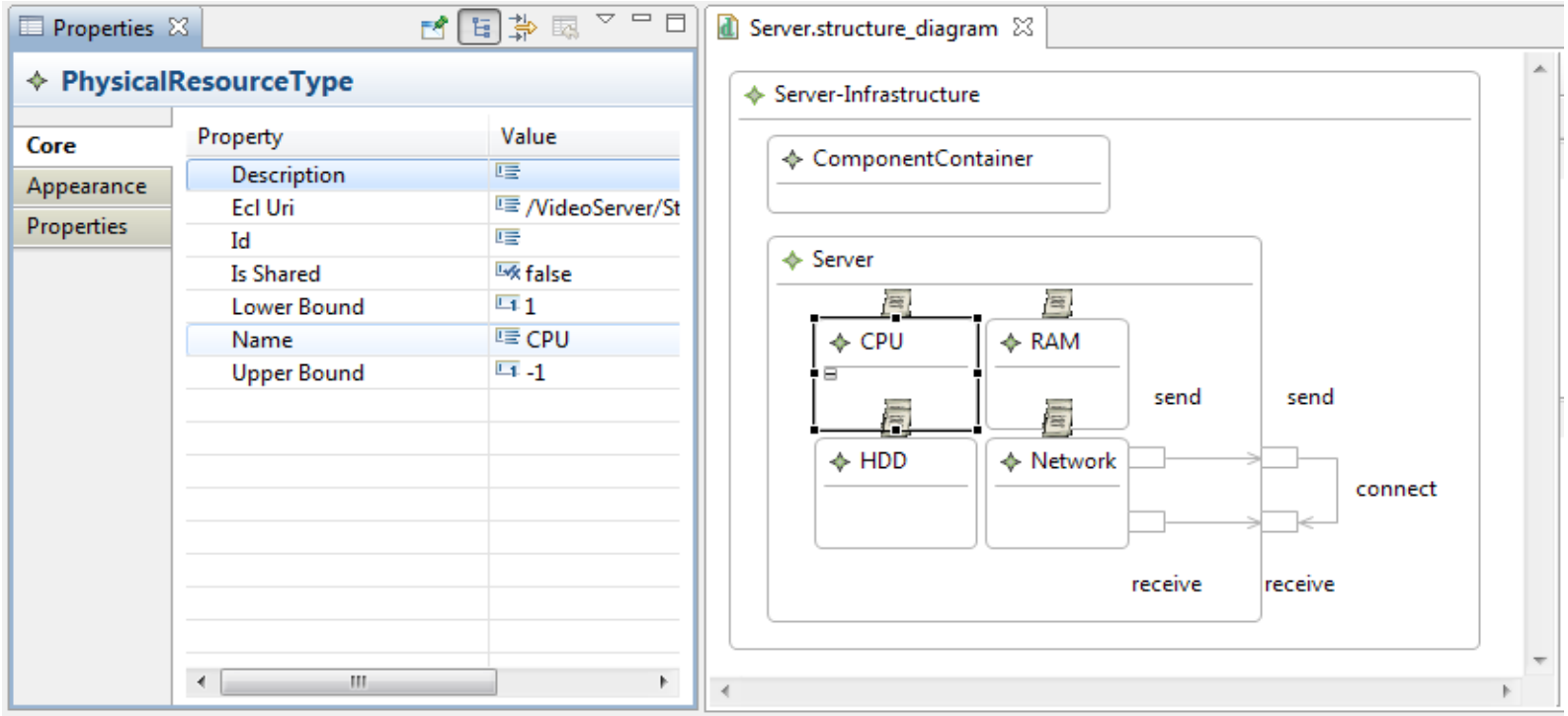


calculated: free = total – used
monitored: ResourceMgr.getX()
static: fixed value for instance (e.g., size of HDD)

- Properties are defined for each ComponentType.
- Units can be defined
- They are either calculated, monitored or static.

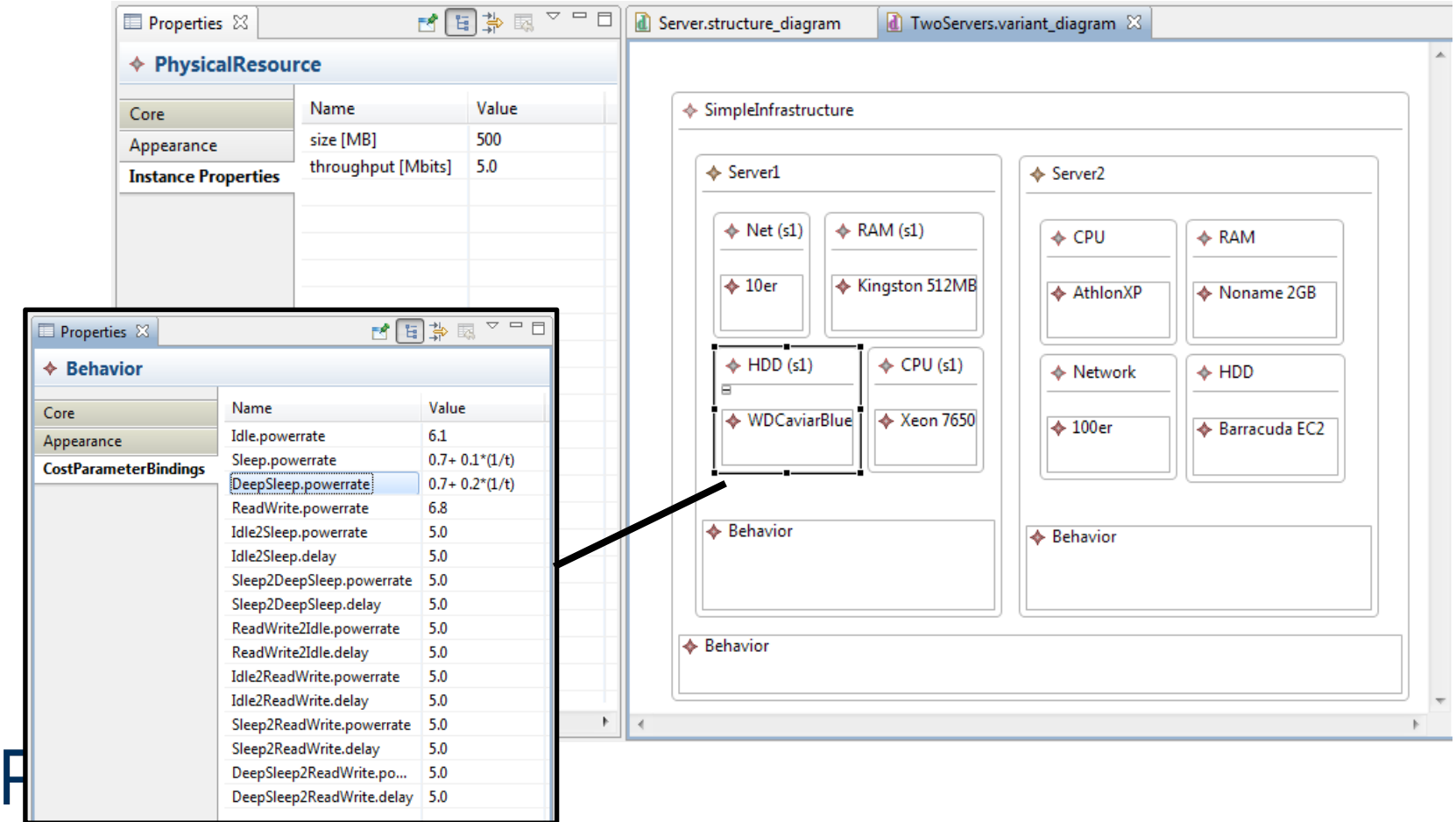
- Which ResourceTypes exist
- How they are connected
- Multiplicities - Contracts

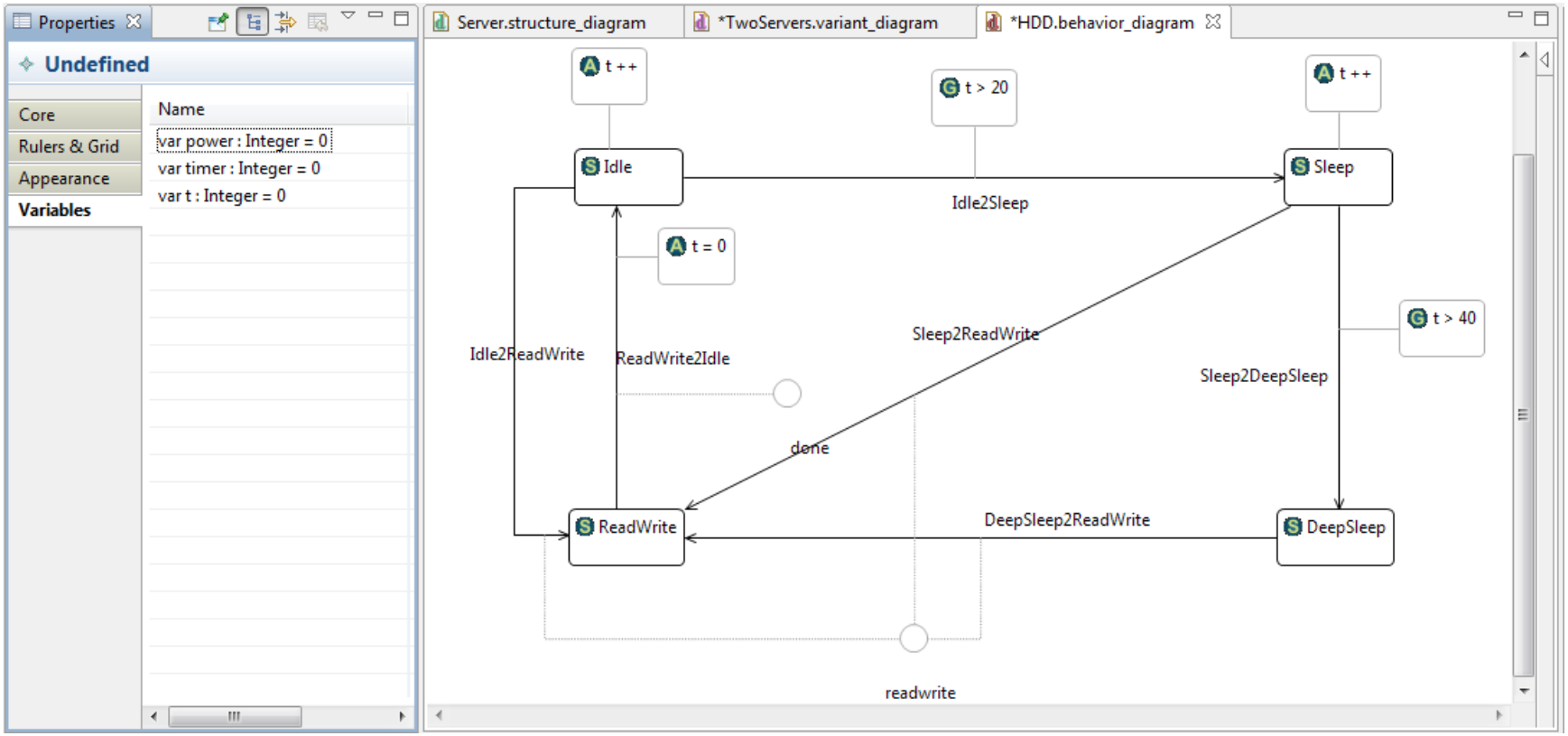
- Hardware Structural Model



- Hardware Variant Model Concrete Resources (next Slide)
- Hardware Behavior using Energy State Charts

- concrete resources are modeled and connected to structural model
- values of static properties are assigned per resource
- behavior templates are assigned per resource,
- Cost parameters are assigned with values or formulas





- use cost parameters instead of concrete values
  - reuse for resources of same kind
- can be executed using workload descriptions

ResUbic

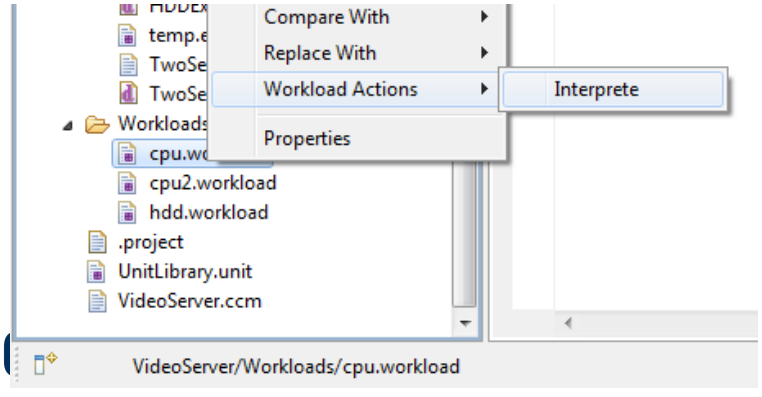
Workload Descriptions

*HDD.behavior_dia... | hdd.workload | HDDExampleTest.va...

```
1  import ccm [./VideoServer/Variants/HDDExampleTest.variant]
2
3  Workload TestHDD for WDHDD {
4      readwrite at 2
5      done at 5
6      readwrite at 20
7      done at 30
8      readwrite at 60
9      done at 100
10     readwrite at 150
11     done at 155
12 }
```

hdd.workload | cpu2.workload | cpu.workload

```
1  import ccm [./VideoServer/Variants/CPUExampleTest.variant]
2
3  Workload someName for Xeon {
4      work at 5
5      work at 20 for 15
6      work at 50 for 20
7  }
```

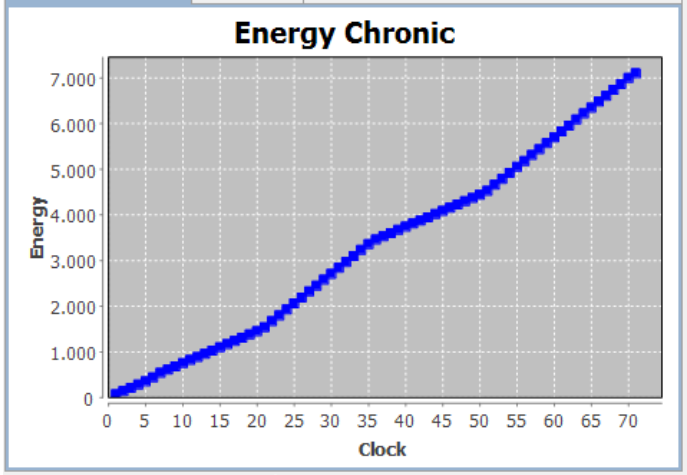


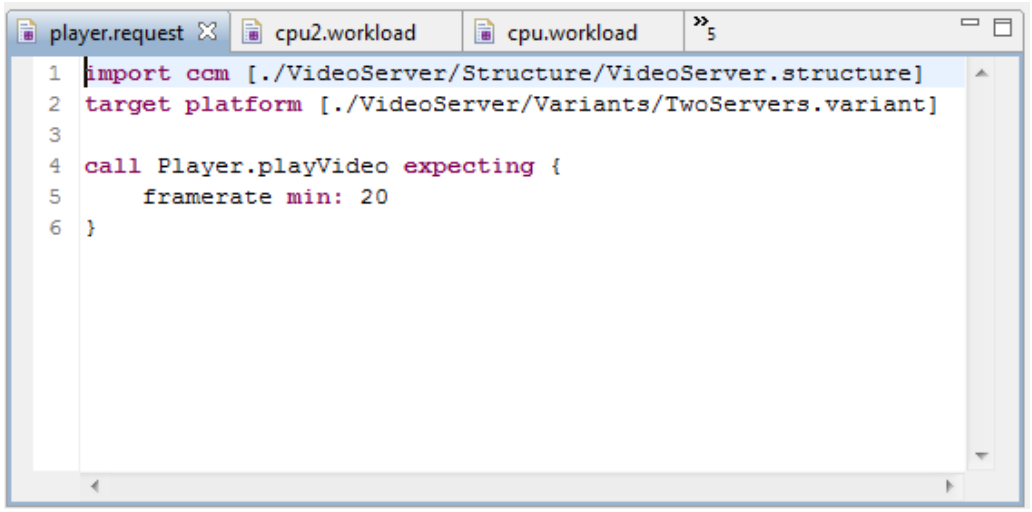
Simulation Results

CPU

Clock	Power	State	Config
1	70	Idle	t = 1, x =
2	140	Idle	t = 2, x =
3	210	Idle	t = 3, x =
4	280	Idle	t = 4, x =
5	350	Idle	t = 5, x =
6	440	Busy	t = 5, x = 1
7	540	Idle	t = 1, x = 1
8	610	Idle	t = 2, x = 1
9	680	Idle	t = 3, x = 1
10	750	Idle	t = 4, x = 1
11	820	Idle	t = 5, x = 1
12	890	Idle	t = 6, x = 1
13	960	Idle	t = 7, x = 1
14	1030	Idle	t = 8, x = 1
15	1100	Idle	t = 9, x = 1
16	1170	Idle	t = 10, x = 1
17	1240	Idle	t = 11, x = 1
18	1310	Idle	t = 12, x = 1
19	1380	Idle	t = 13, x = 1
20	1450	Idle	t = 14, x = 1
21	1540	Busy	t = 14, x = 2
22	1670	Busy	t = 14, x = 3
23	1800	Busy	t = 14, x = 4
24	1930	Busy	t = 14, x = 5
25	2060	Busy	t = 14, x = 6
26	2190	Busy	t = 14, x = 7
27	2320	Busy	t = 14, x = 8
28	2450	Busy	t = 14, x = 9
29	2580	Busy	t = 14, x = 10
30	2710	Busy	t = 14, x = 11
31	2840	Busy	t = 14, x = 12

Energy Chronic | Problems





- Users call methods on SW Components
- They state their (quality) expectations explicitly
- HW infrastructure is selected

- Then the software configuration is automatically optimized with Multi-Objective-Optimization (MOO)

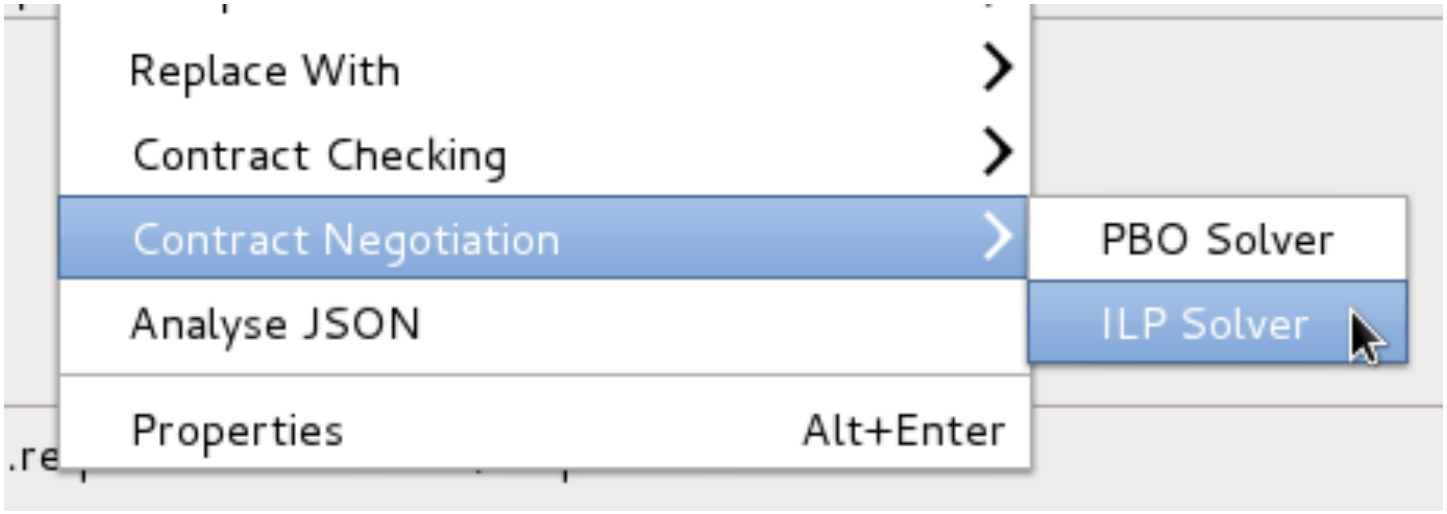
Multi-Objective Optimization (MOO) of Configurations

Question:

- Which implementations of the contracts
- in which quality mode
- on which resources
- give the most efficient configuration?

Multi-Objective Optimization (MOO) with:

- Integer Linear Programming (exact)
- Pseudo-Boolean Optimization (exact)
- Ant Colony Optimization (approx.)
- Simulated Annealing (approx.)
- ...



- An ILP is generated for a certain request on a software component and a hardware variant.