

Predictable Composition

Predictable Assembly

M.R.V. Chaudron

www.win.tue.nl/~mchaudro

Dept. of Mathematics and Computing Science
Eindhoven University of Technology

With contributions from Prof. Ivica Crnkovic, Malardalen, Sweden

Agenda

- On testing
 - On binding
 - Reasoning about composition of components
 - Evaluating Architectural Alternatives
 - Concluding remarks CBSE lectures
-
- Next week: demonstration of the assignments
(na overleg met Tom Verhoeff)

CBD testing problems – Buyer

- Vendor has different problems
- Lack of access to source code & design documents makes testing more difficult because:
 - More difficult to understand what happens/can happen
 - What is its state-space?
 - Access to source or documentation is desirable for selecting test-scenarios
 - Consider a set implemented as an array or as a list
 - Test adequacy
 - how much testing is enough? No coverage known!
- Lack of knowledge of component creator

CBD testing problems – Buyer

- Input domain may be too large to cover a significant portion
- If a component provides functionality you are not planning to use, it is probably wise to also test that.
- There may be more assumptions on the working of other components than becomes clear from the specification & interface.

Testability

determined by (o.a.) the following factors

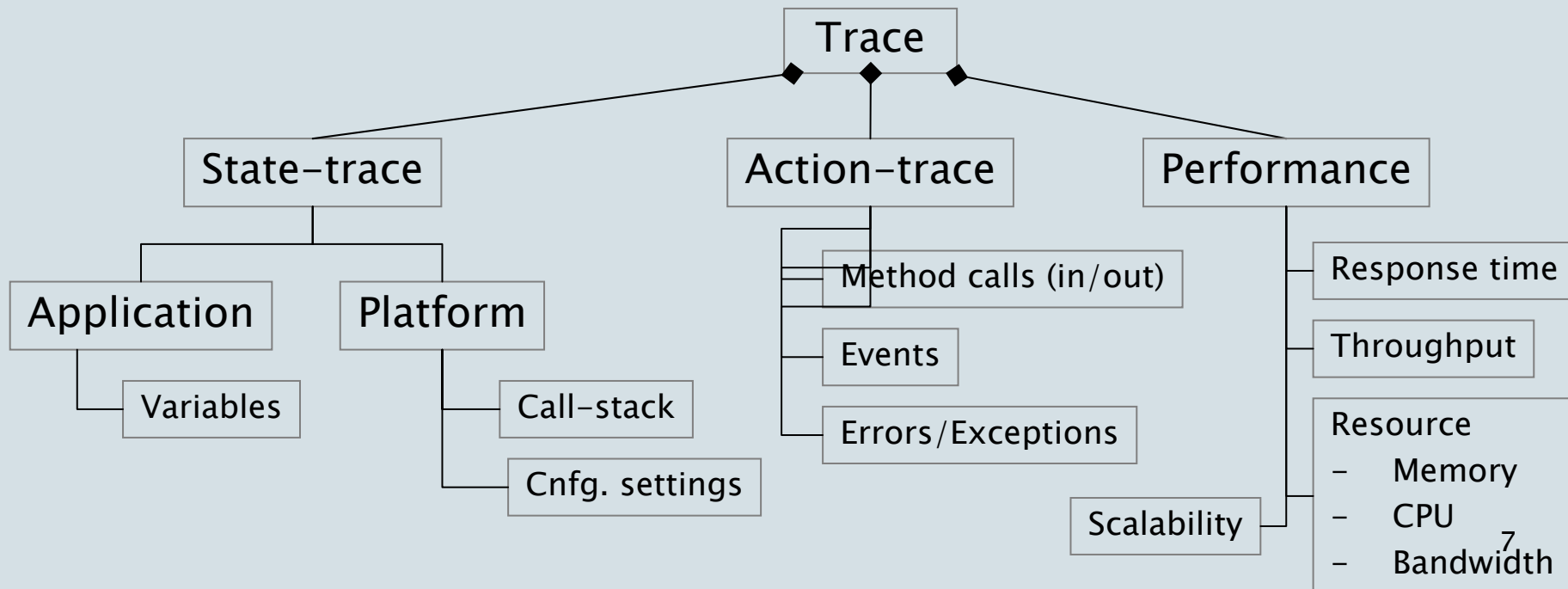
- Understandability
- Observability
- Traceability
- Controllability
- Test support capability

Testability – Understandability

- Availability of component information
 - User documentation
 - Development documentation
 - Testing documentation
 - Test plan
 - Test criteria
 - Test scripts/drivers/stubs
 - Problem reports (?)
 - Coverage data
- Understandability of available information

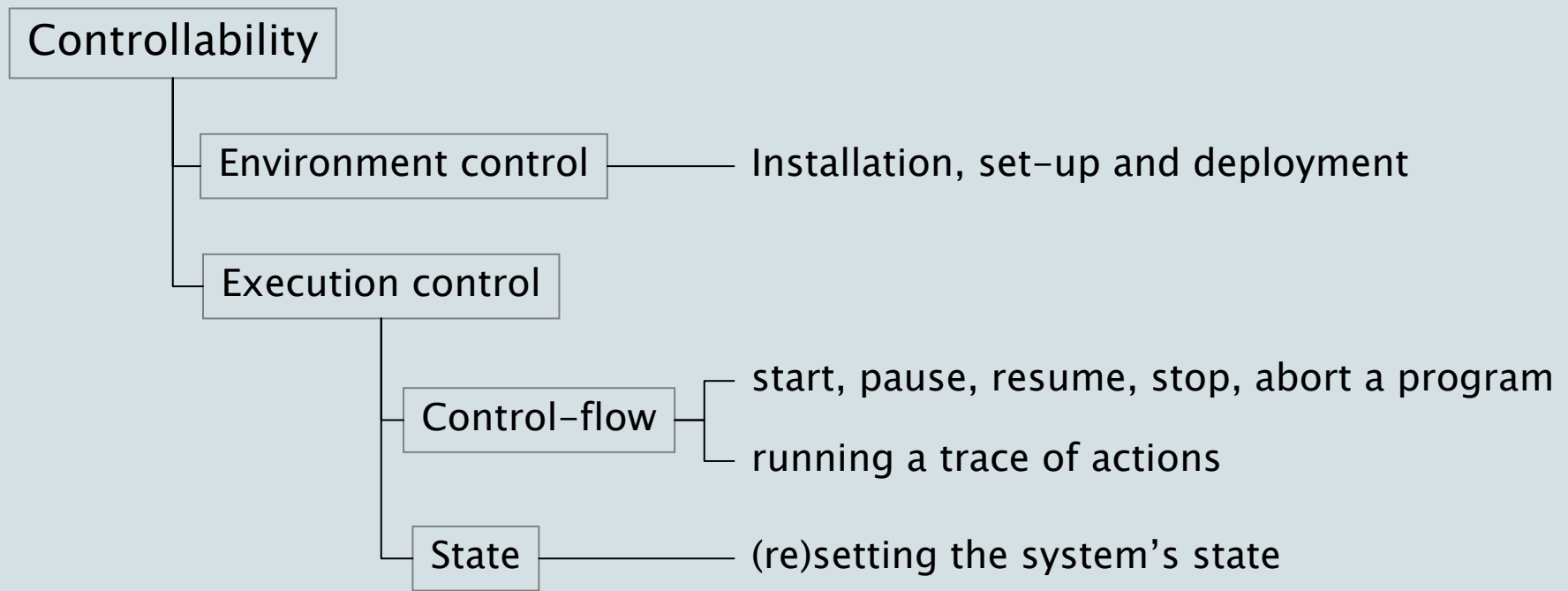
Testability – Observability/Traceability

- How easy it is to observe/monitor the response of a component on stimuli
 - Different inputs lead to different outputs
- Traceability: capabilities that enhance observability



Testability – Controllability

- How easy is it to put a component into a specific state and make it perform a certain action

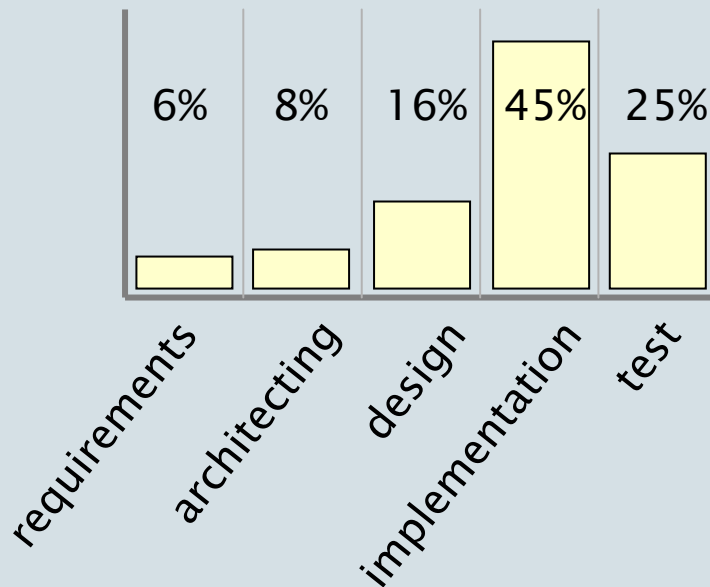


Expected Effects of CBD on the SE Process

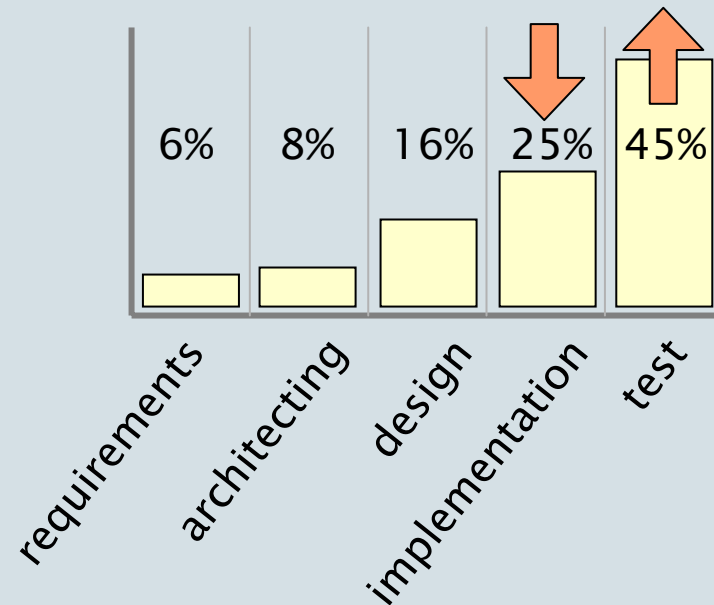
E. Weyuker:

CBD may reduce effort needed for building,
it will increase the effort needed for testing.

Typical effort distribution



Expected distribution for CBD

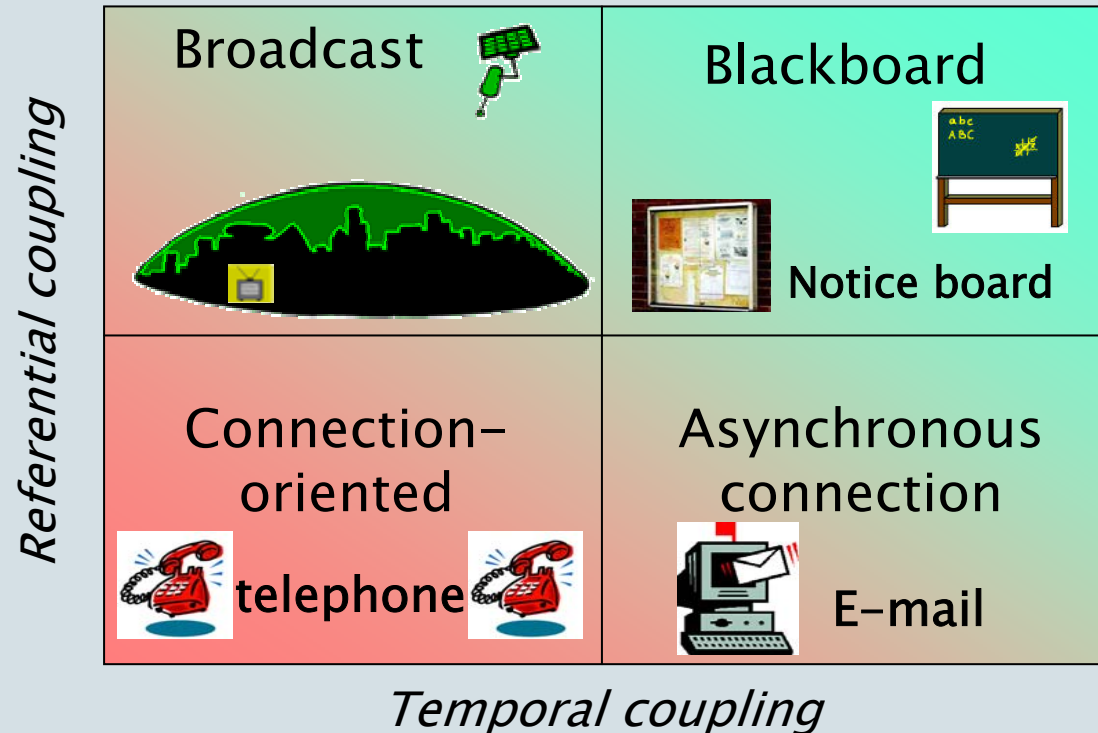


Little empirical measurement is available (see article by W.C. Lim)⁹

Binding & Dependencies

Interaction Style and Binding

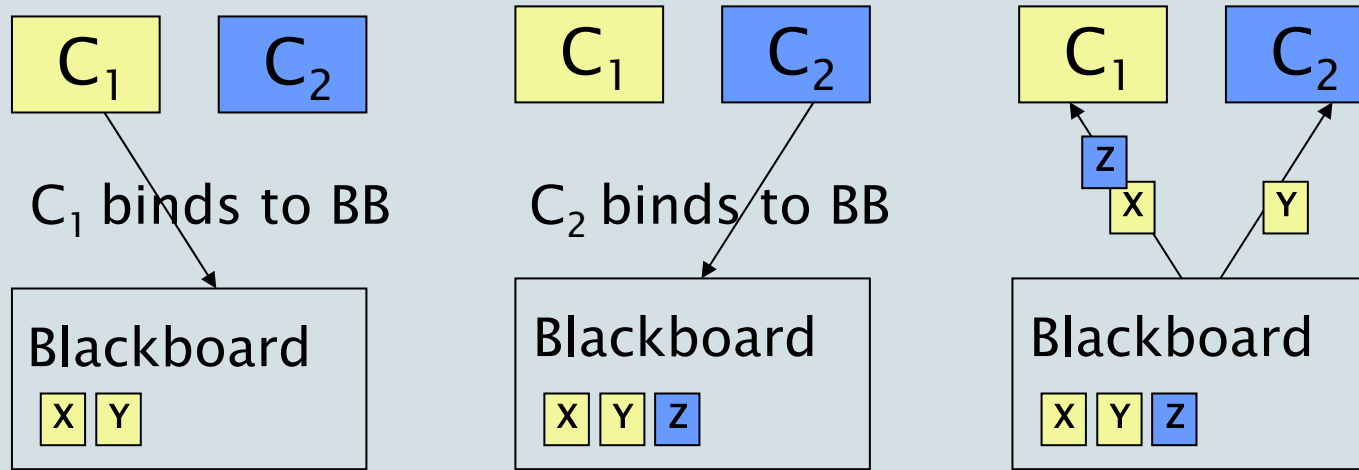
Interaction styles bring
inherent type of binding



Referential coupling : sender has reference to receiver name/address

Temporal coupling: sender and receiver synchronize in time

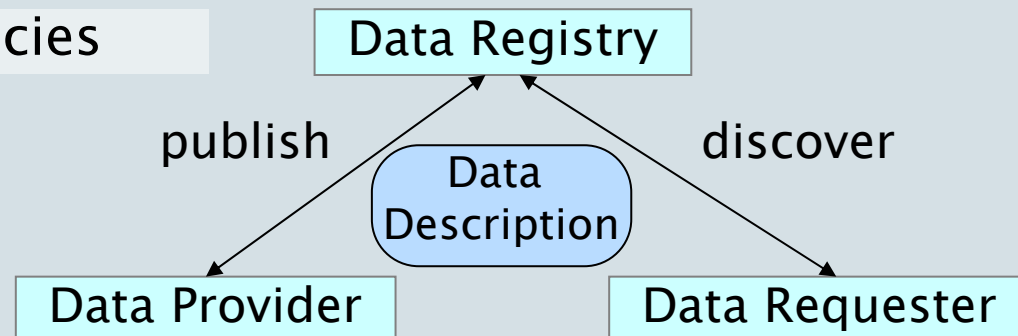
Binding Blackboard style (e.g. JavaSpaces)



C_1 and C_2 read data from the BB – independent of who put it there

C_1 and C_2 do not need references to each other !

No references → no dependencies



Predictable Assembly

- What are the properties of a composition?
- How can properties of a composition be derived from the properties of its parts?
 - Functionality
 - Extra-Functional
- Which information needs to be specified per component in order to construct a model of the composition?

When Predictable Assembly

- Design Time
- Run-time change in components

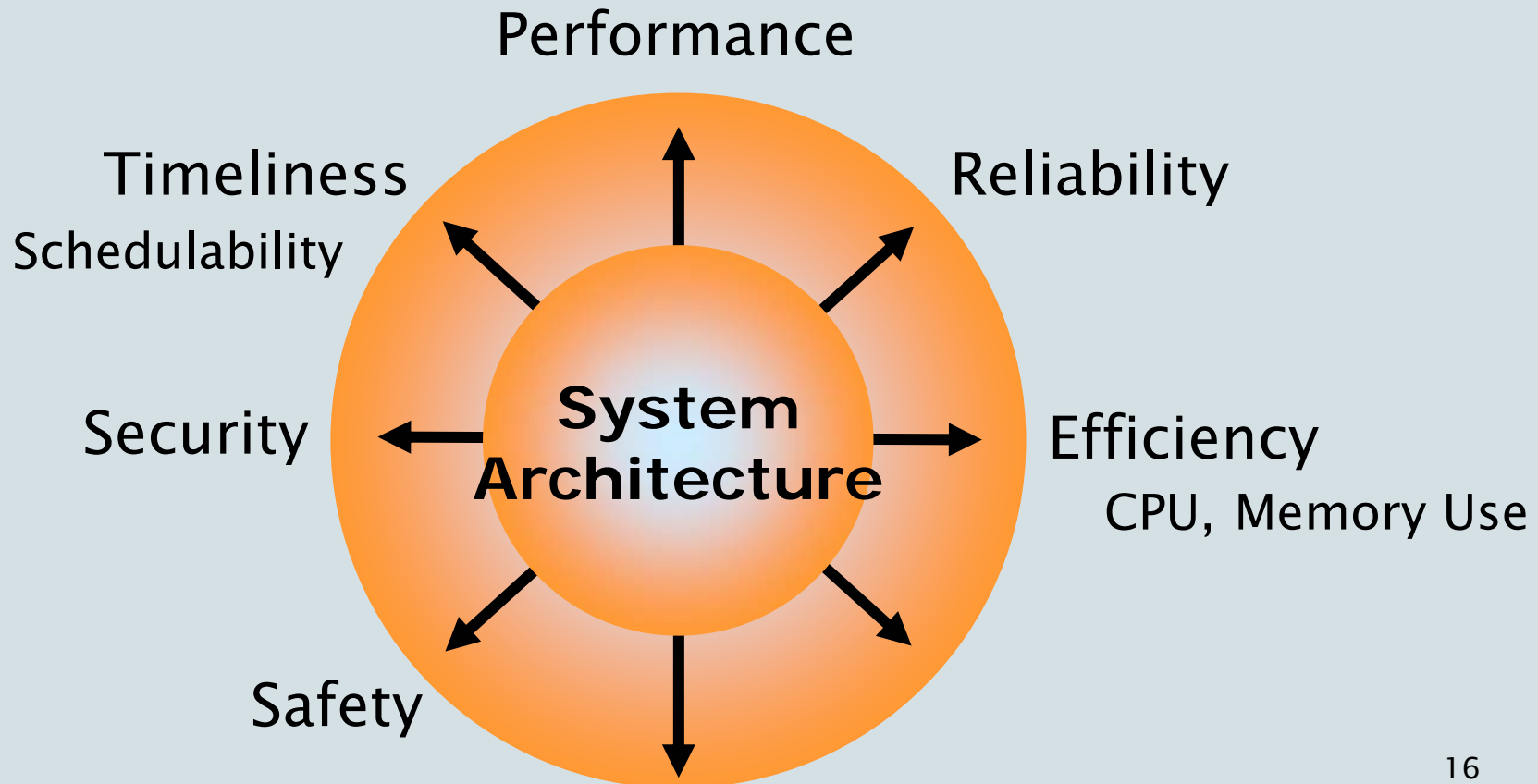
Composition of properties

Hypothesis: properties of the system are a function of properties of the components



Extra-Functional Properties

An architecture needs to balance the extra-functional system properties



What are properties?

Examples:

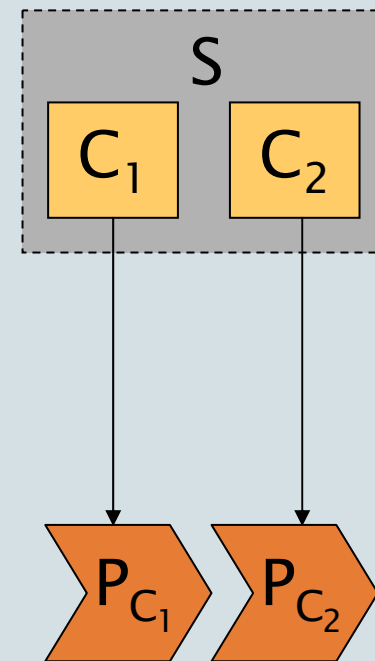
- Timeliness, Responsiveness, Schedulability, CPU utilization, Memory utilization, Latency, Throughput, Concurrency, Footprint
- Availability, Reliability, Safety, Security,
- Cost
- ...

Problems

- There are different system/components properties
 - Different abilities to (formally) specify them
 - Different abilities to measure them
 - Different complexity (relations to other attributes...)
 - Different ways (and efforts) to achieve them
- Different properties are of interest in different domains

Compositional Reasoning (Semantics)

- Calculating properties of a system by combining properties of its constituents.
- If $S = C_1 \oplus C_2$
- Then $P(S) = P(C_1) \otimes P(C_2)$
- ‘Traditionally’ $P(C_i)$ denotes the functional meaning of C_i



Predictable Assembly: Functionality

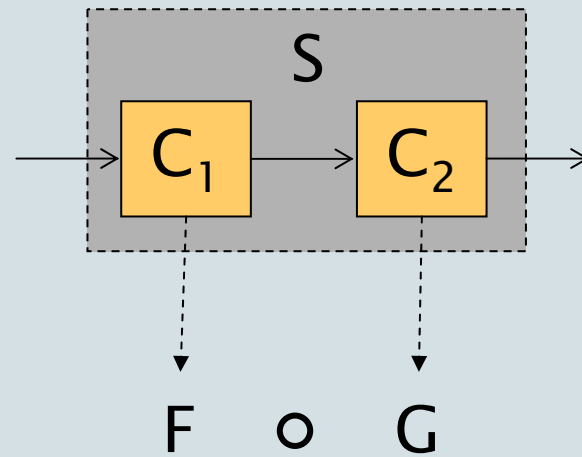
- Can be fairly adequately specified using
 - predicate logic (pre- and post-conditions)
 - using functions (lambda expressions)
- However, the **behaviour** of components is typically not sufficiently well specified;
typically because this is not part of the program text.
- Extra-functional properties?

Is a component autonomous or reactive?

- If autonomous, then when does it take action?
- What happens if two reactive components are composed?

Compositional Reasoning: Functionality

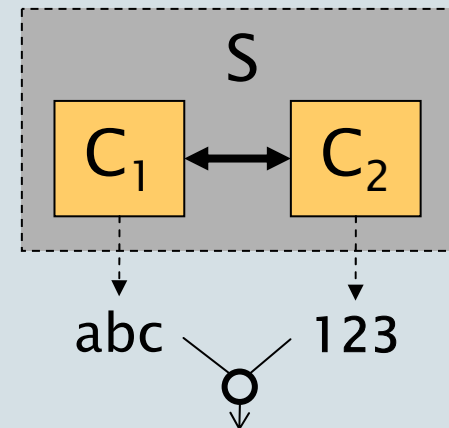
- Meaning $M(C)$ of program C can be a function from inputs to outputs; e.g. $M(C) = \lambda x.x^2$
- Then composition is nicely modelled by function composition



Compositional Reasoning: Behaviour

e.g. Traces

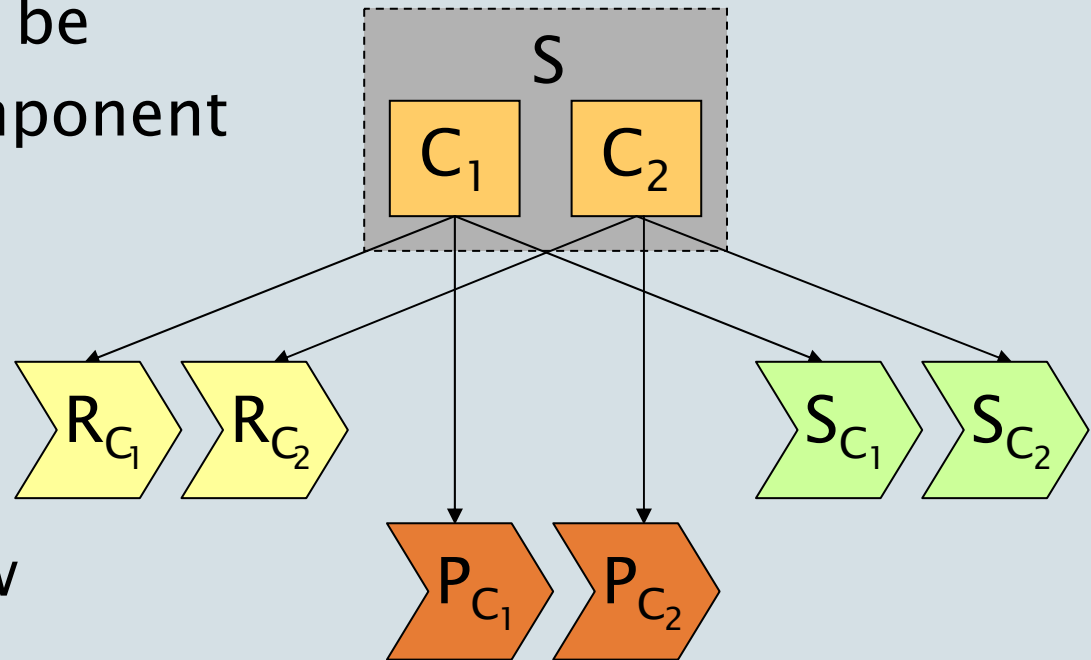
- Meaning $P(C)$ of program C can be the (set of) traces of actions that C can perform
- then sequential composition can be modelled by concatenation
- parallel composition can be modelled by interleaving
- complications arise if components synchronize



$abc123, ab1c23, a1bc23, \dots, 123abc_2$

Compositional Reasoning

Different models may be associated with a component



some examples follow

Robocop Component

Robocop Component

Resource Model

Simulation Model

Documentation

Functionality Model

Source Code

...

Executable Component

A **Robocop Component** is a set of Models

- Model can be machine and/or human readable / executable
- Typically: one of those is executable on a target system
 - Called the (executable) component

Example

- “Physical characteristics”
 - Static memory

$$M(C) = \sum_{i=1}^n M(c_i)$$

M = memory size

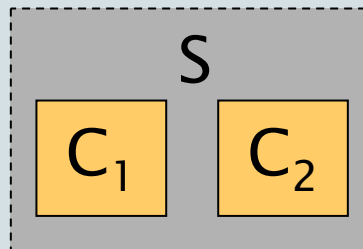
C = composition

c_i = components

$M(c_i)$ is known for all c_i

Predictable Assembly

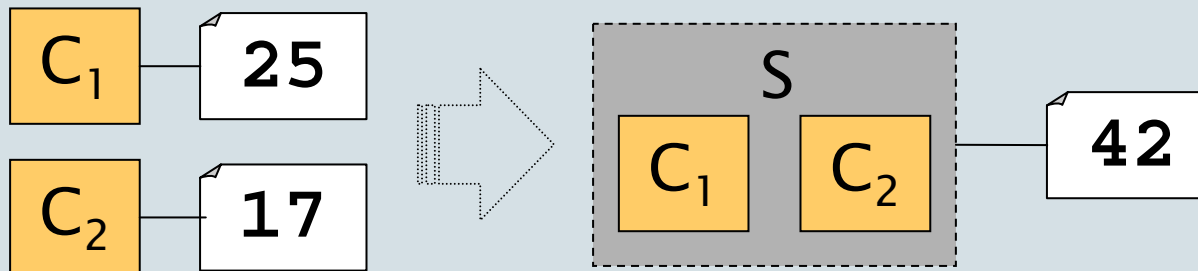
- Same question, now for extra-functional properties.
- Static properties vs. Dynamic properties
- Let's consider dynamic memory-use
- Given the dynamic memory-use of C_1 and C_2 .
- Now what is the dynamic memory use of S ?



?

Problem Instance: Cost

Derive cost of a system from cost of its parts.

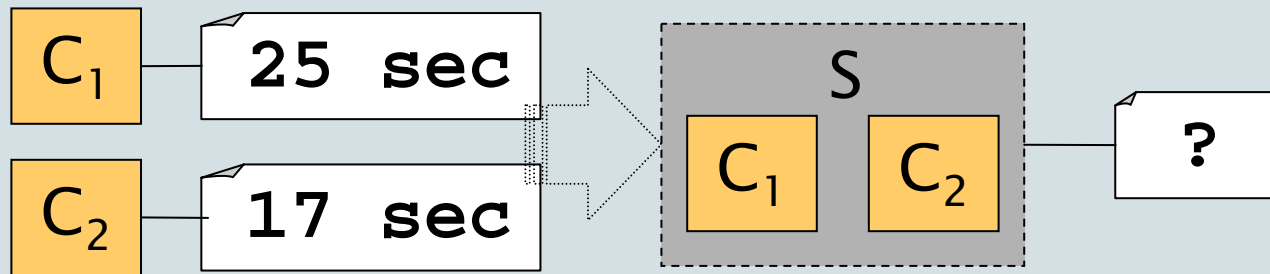


Other example: static memory use

Let's move onto real-time / timeliness properties

Problem Instance: Timeliness

Derive timing of a system from timing of its parts.



There is interaction via shared resources.

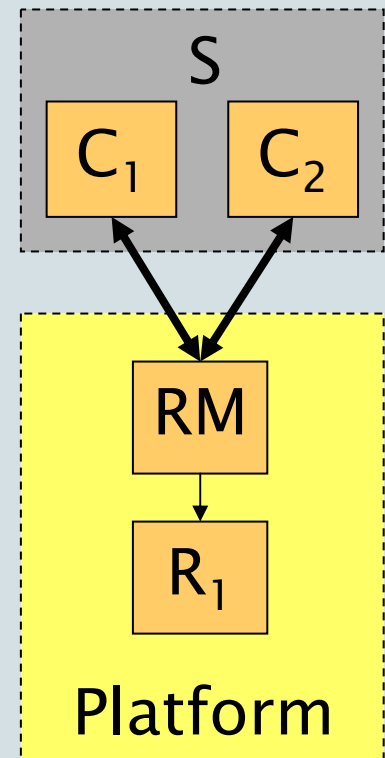
Discussion

- Take 4 minutes to think about a solution
- What are the complications

- break

Complicating Factors in Compositional Reasoning about Extra-Functional Properties

- The property is not determined by the application software only
- But also by the platform
 - platform may be OS + run-time environment
 - in particular the resource management
 - Scheduling
 - Memory management
- The information supplied by C_1 and C_2 is not sufficient to reason about the composition of their extra-functional properties.



A Scenario-based Approach to Predictable Assembly

Based on research with

Johan Muskens & Egor Bondarev

as part of the ITEA projects Robocop and Space4U

Component behaviour model

```

component c2
  requires I2
  requires I3
  provides I1{
    operation f
      uses I2.g
      uses I3.h
      behaviour
        operation f calls:
          I2.g*
          I3.h
  }
  
```

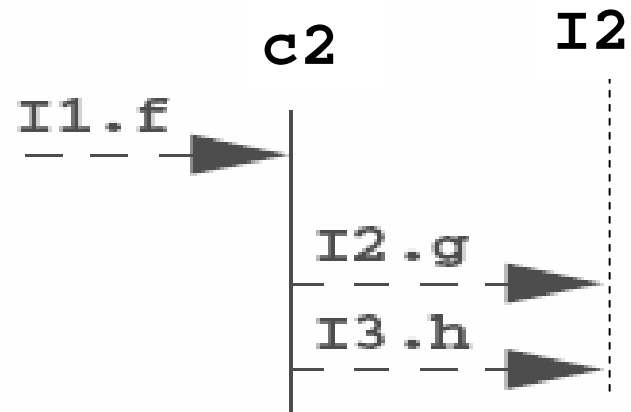
Component

Behaviour

Variables

Resource claims / releases are modeled explicitly

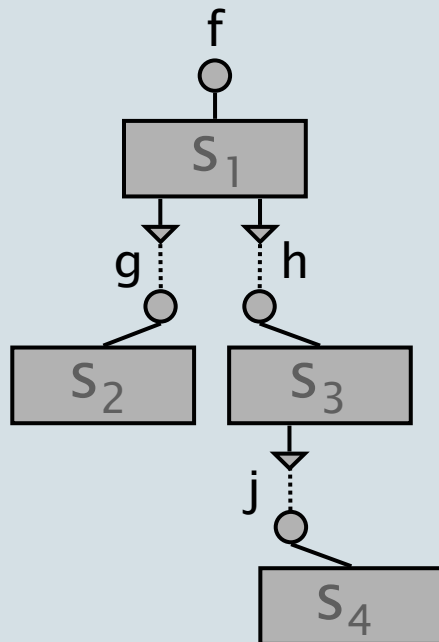
Behaviour forms a partial call graph



Operation f resource use:
 claim 100
 release 100

Model Assembly: Phase 1: Structure

- At bind-time, the decision is made which services are composed to provide the implementation.
- Press button 'f'



- f is provided by s_1
- s_1 needs g and h
 - g is provided by s_2
 - s_2 is done
 - h is provided by s_3
 - s_3 needs j
 - j is provided by s_4
 - s_4 is done

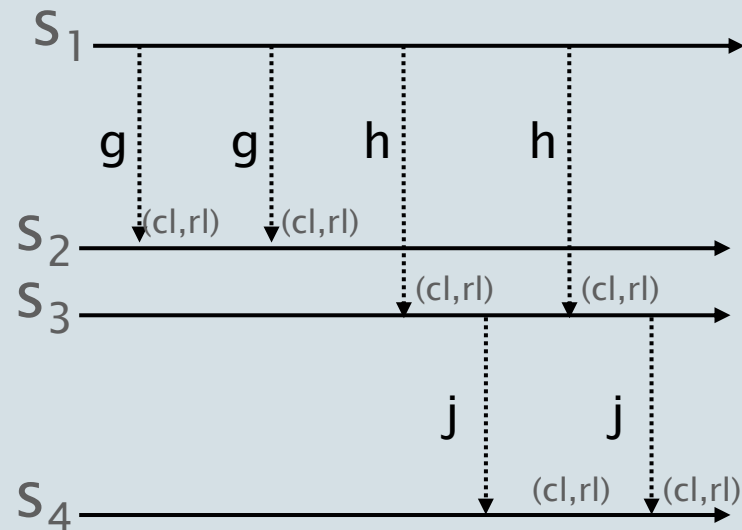
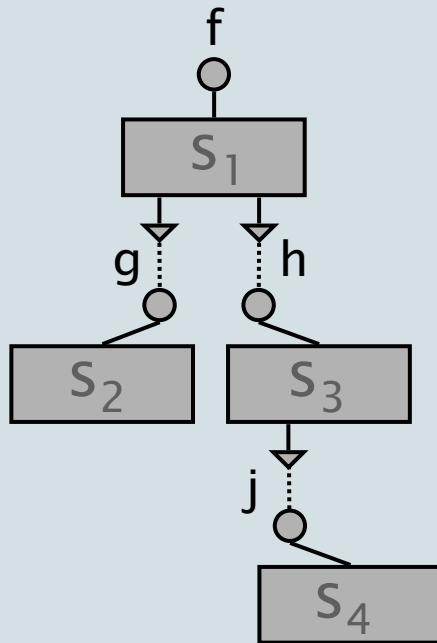
Model Assembly: Phase 2: Logical Behaviour

Combine the behaviour models of the operations used

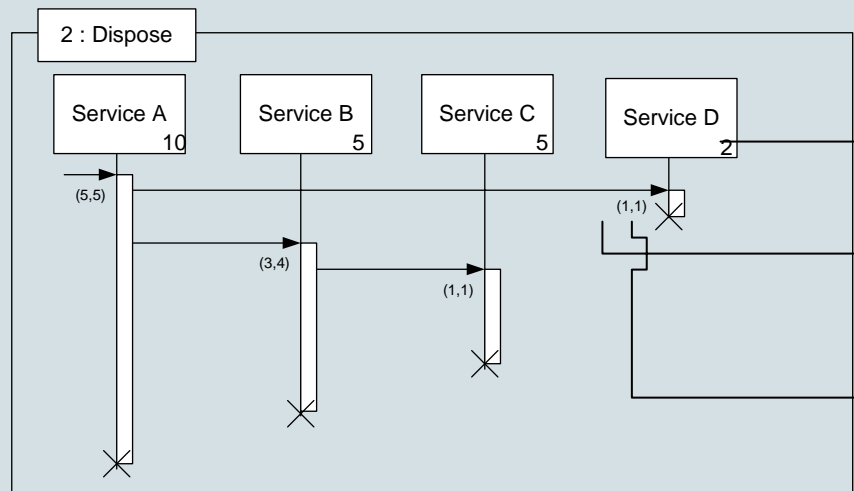
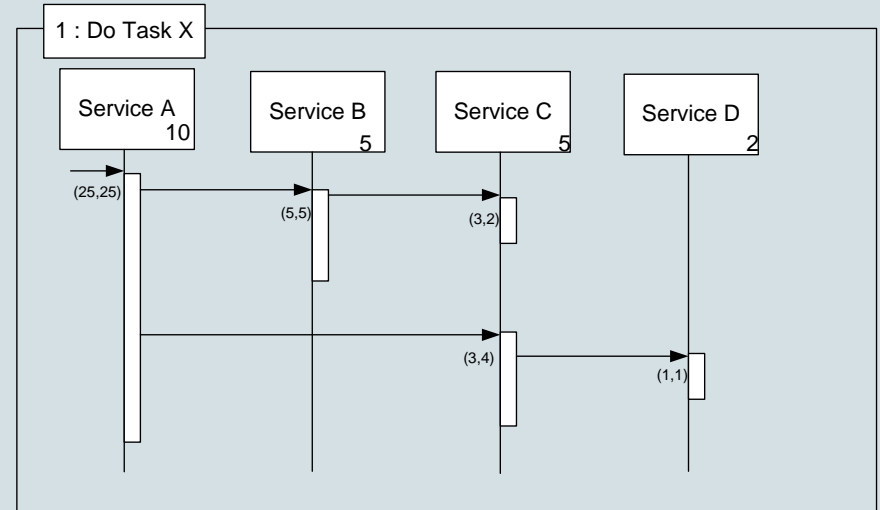
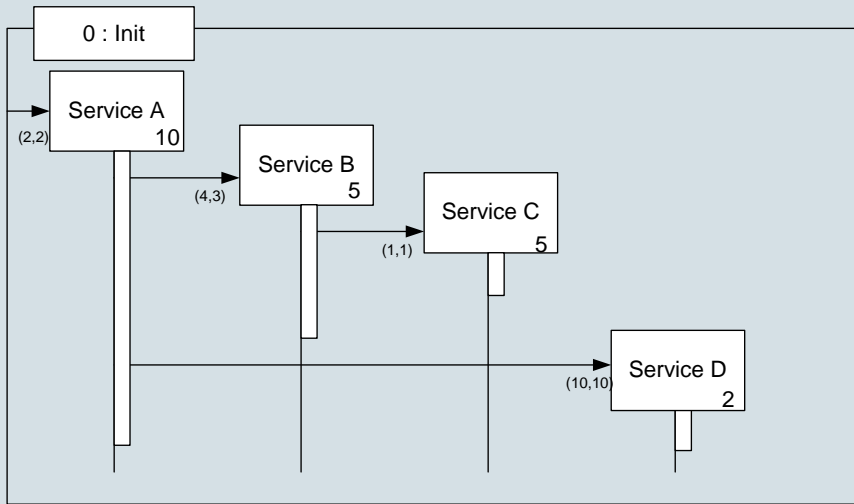
Press button 'f'

operation f calls:

S2.g; S2.g; S3.h; S3.h



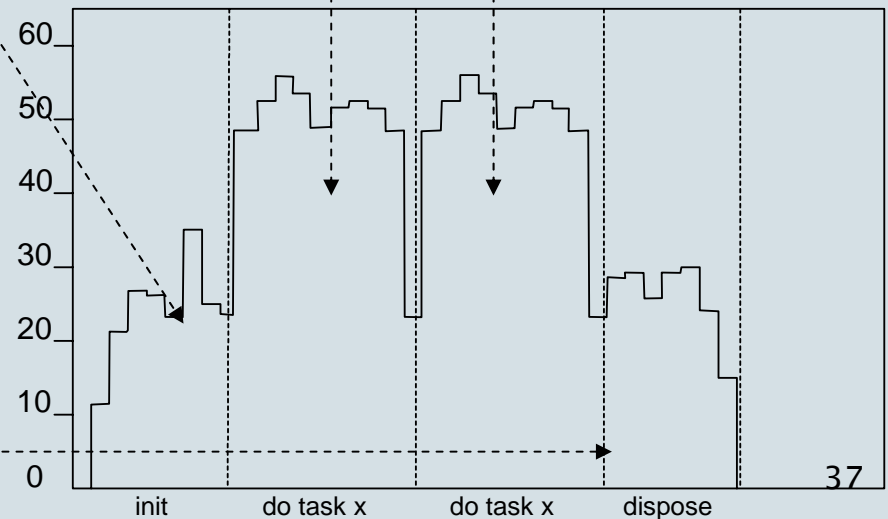
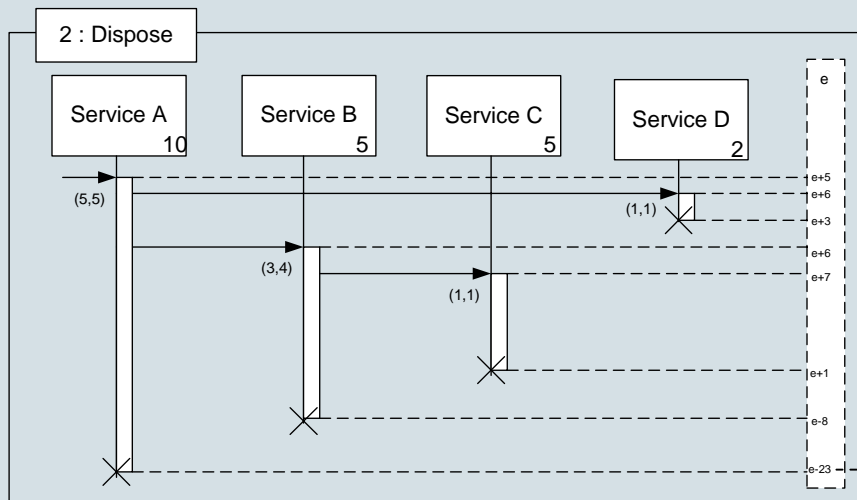
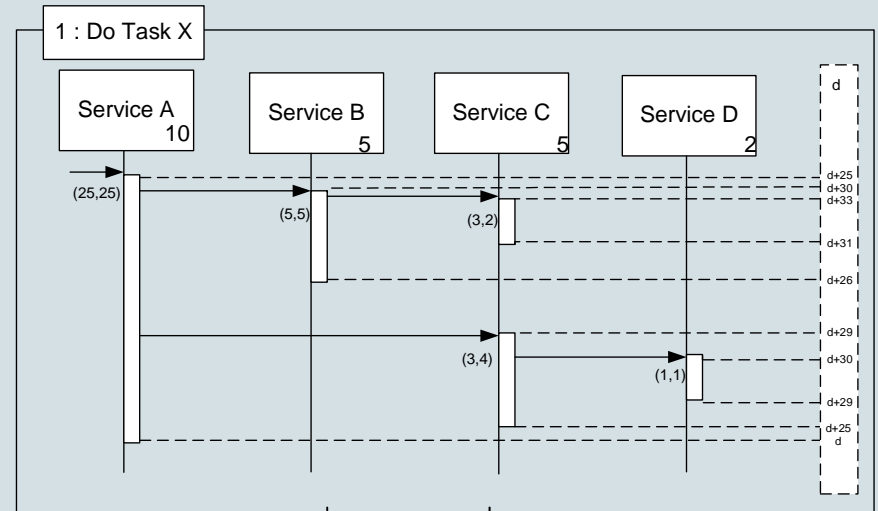
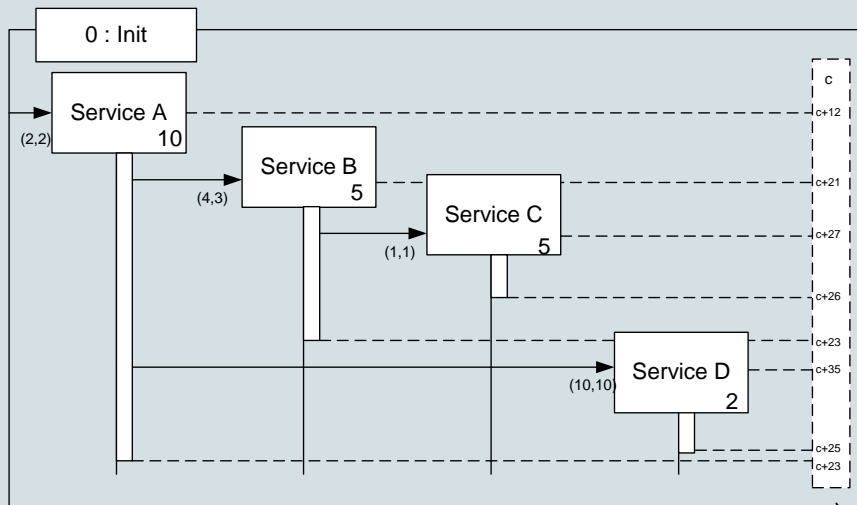
Annotate Scenario using Resource Use



Annotations:

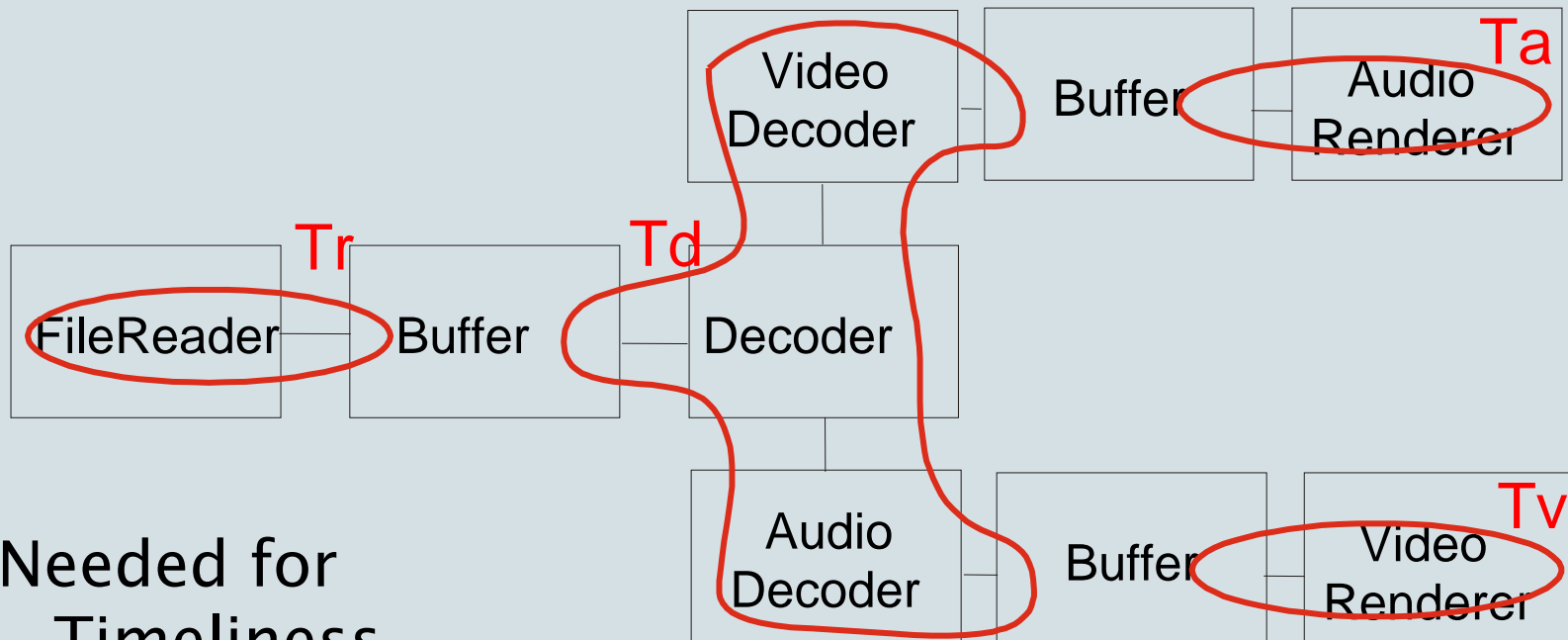
- Nr of Resources needed for creation of instance
- Nr of Resources that are claimed before / during operation execution
- Nr of Resources that are released after / during operation execution

Concatenate Scenarios



Dealing with Concurrent Tasks

- Task based accumulation of resource consumption
- Accumulation of per task resource consumption based on scheduling policy

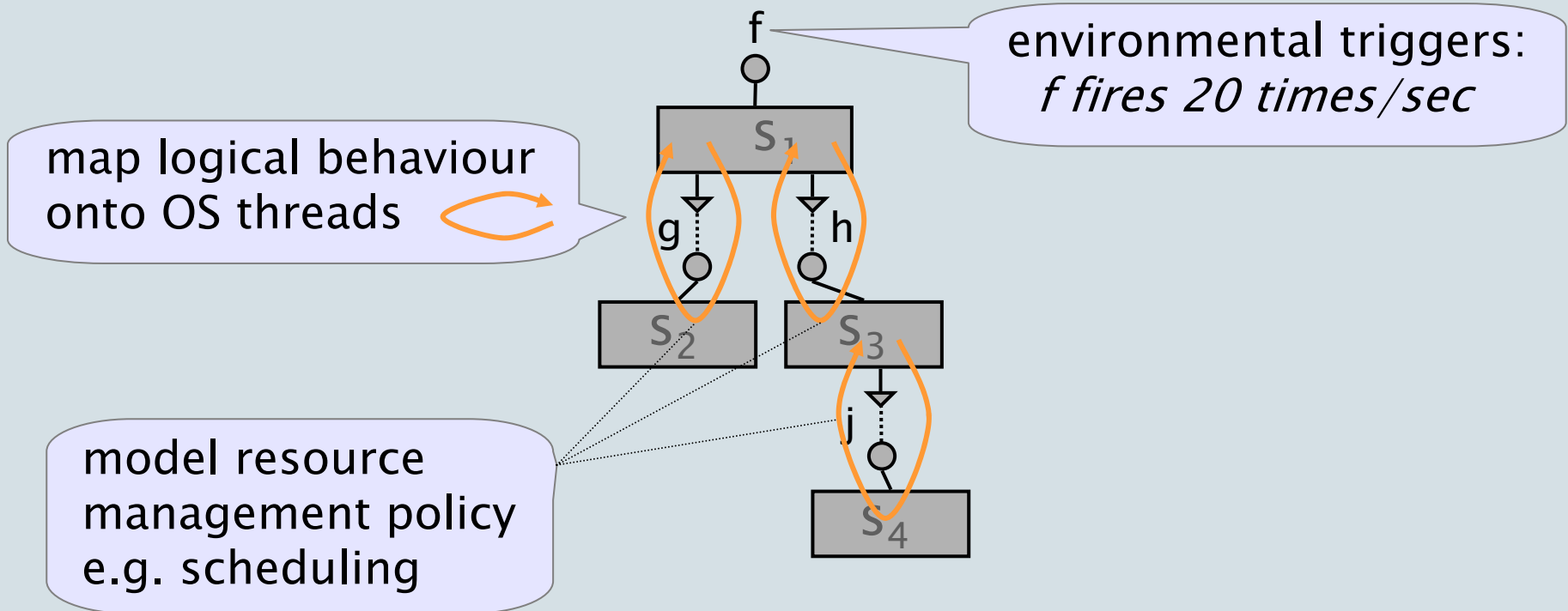


Needed for

- Timeliness
- CPU utilization

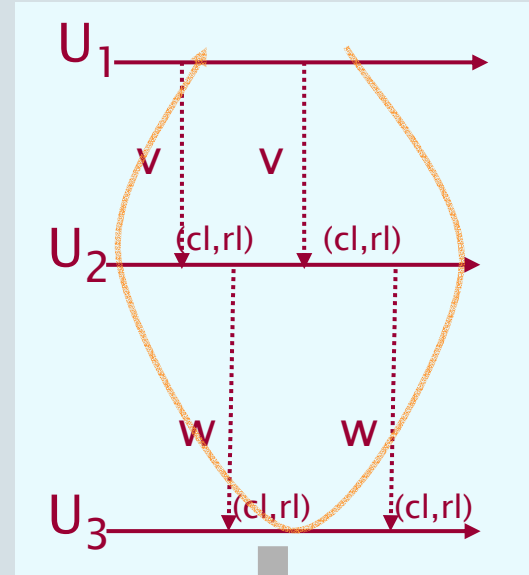
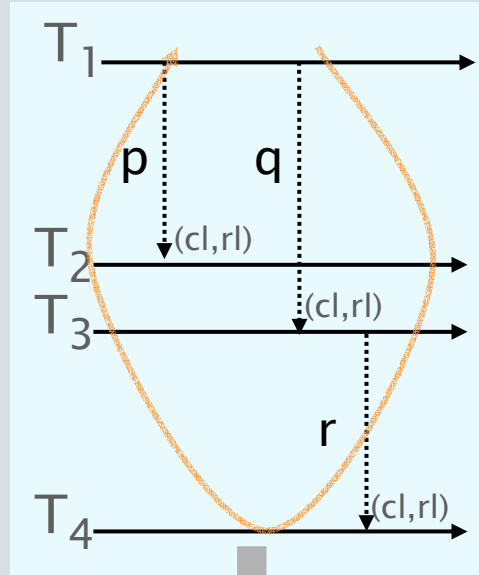
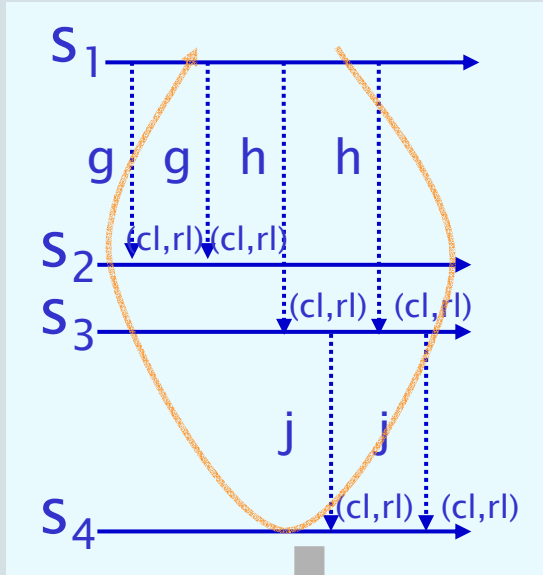
Composition of Execution Behaviour

- Map 'logical' behaviour onto concurrent tasks
- model triggers from the environment (statistically)



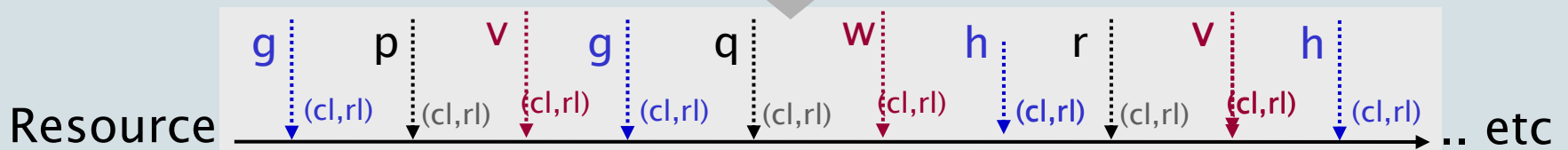
- additional complexity: synchronization between tasks 39

Phase 3: Execution Behaviour



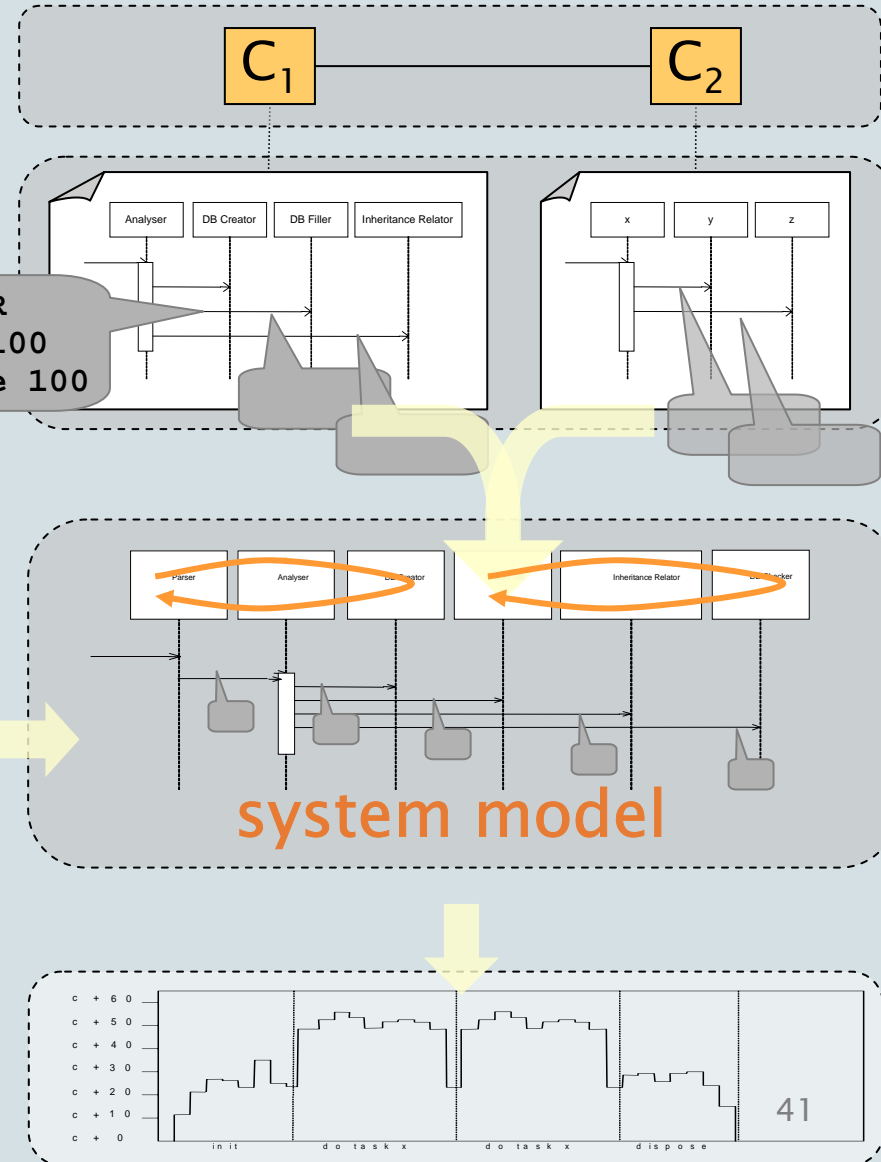
Resource Management Policy
e.g. scheduler

e.g. Round Robin,
EDF, ...



Recipy for Predictable Assembly

1. Composition of System Structure
2. Composition of Logical Behaviour of whole system
3. Composition of Execution Behaviour
 - threads
 - resource mng policies
4. Analyse



Tooling

The screenshot displays the Eclipse Platform interface for editing a scenario. The main window is titled "Resource - CarNav_Arch5_Scen1.scenario - Eclipse Platform". The interface includes a menu bar (File, Edit, View, Navigate, Search, Project, Run, Window, Help), a toolbar, and a Navigator on the left showing a project structure with folders like "ScenarioDemo" and "UseCase".

The main editing area is titled "Edit the whole scenario" and contains a diagram with three cyan-colored components. The components are connected by lines representing relationships. The connections are labeled "IPParameters" and "IRDSDecoder". A box labeled "Node1" is visible in the background of the editing area.

Below the diagram is a "Scenario" section with a "Properties" table:

Property	Value
id	w11012943433160
Name	CarNav_Arch5_Scen1

At the bottom of the interface, there are tabs for "Tasks" and "Properties".

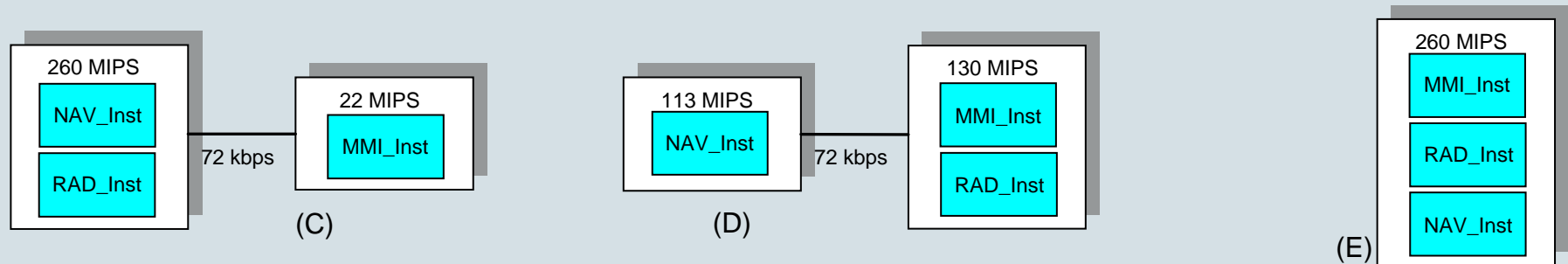
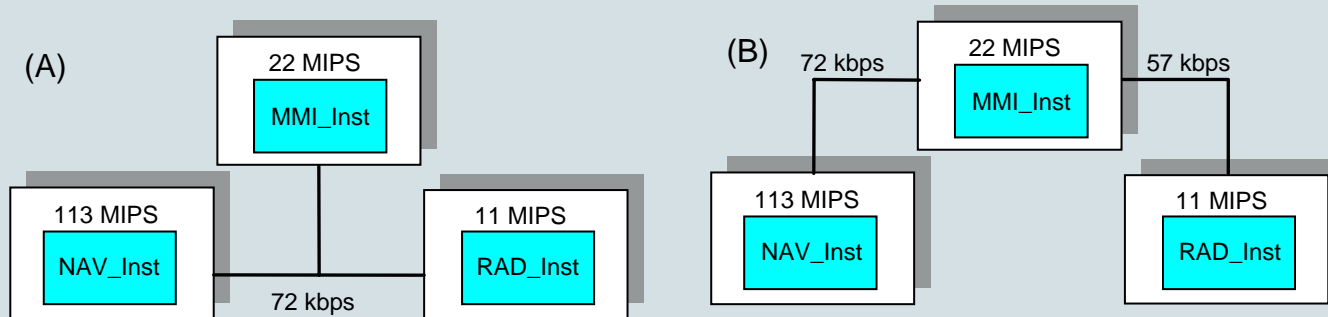
Evaluation of the prediction method

- Compositional (supports third-party binding)
- Can be applied to different types of resources
- Support different types of analyses (Worst, Best, Avg, ...)
- Can be used throughout development cycle (design / implementation / deployment).
- Scenario's
 - Do not give 100% guarantees as usual in formal methods,
 - Focus on what is important
 - Allow incremental accuracy & trade-off with effort invested
 - Can be used in 'lightweight' manner

Composability Property Classification

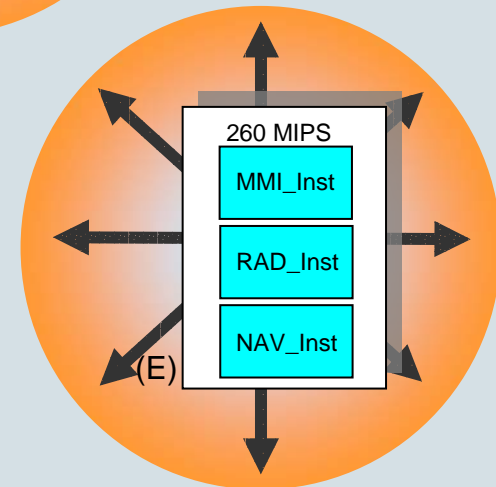
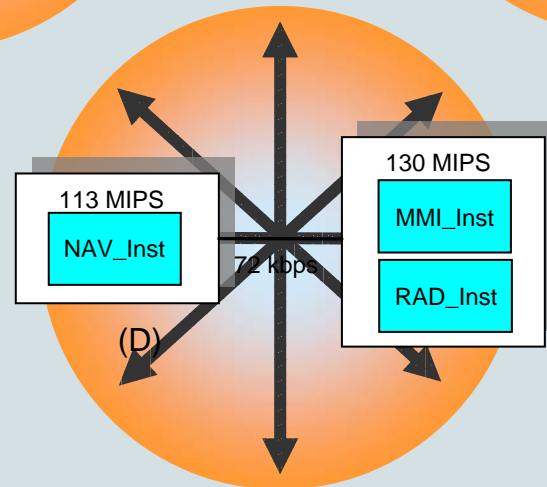
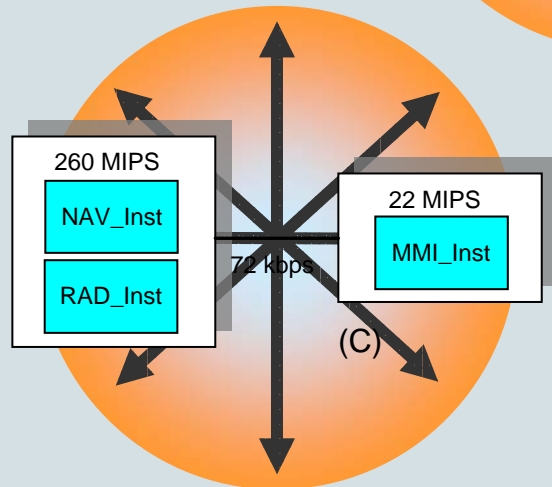
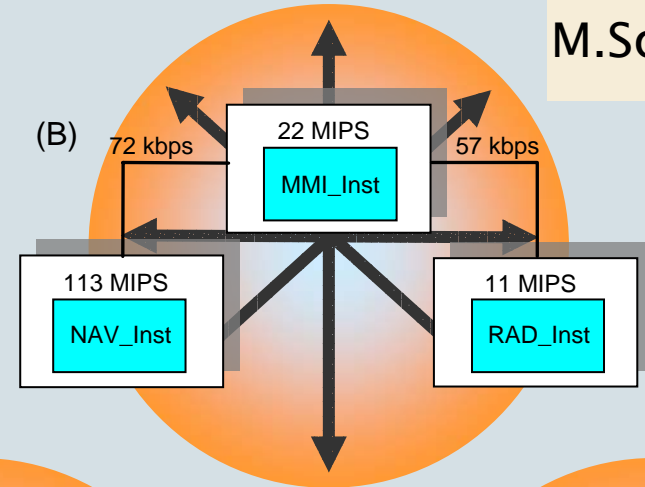
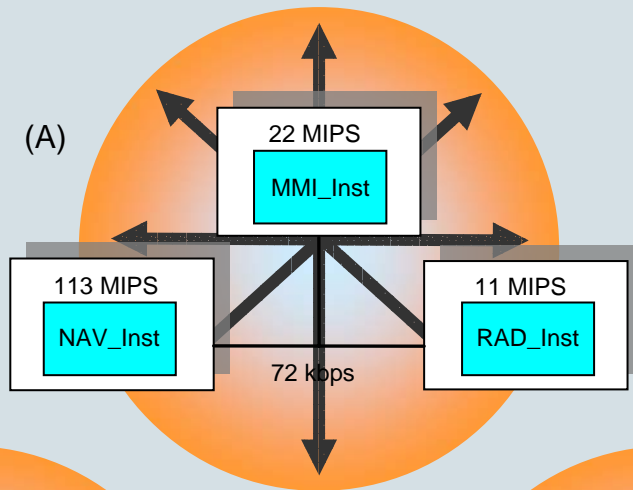
1. *Directly composable properties.* A property of an assembly which is a function of, and only of the same property of the components involved. E.g. cost.
2. *Architecture-related properties.* A property of an assembly which is a function of the same property of the components and of the software architecture.
3. *Derived (emerging) properties.* A property of an assembly which depends on several different properties of the components.
4. *Usage-depended properties.* A property of an assembly which is determined by its usage profile. E.g. CPU load.
5. *System context properties.* A property which is determined by other properties and by the state of the system environment.
E.g. safety.

Evaluating Architectural Alternatives

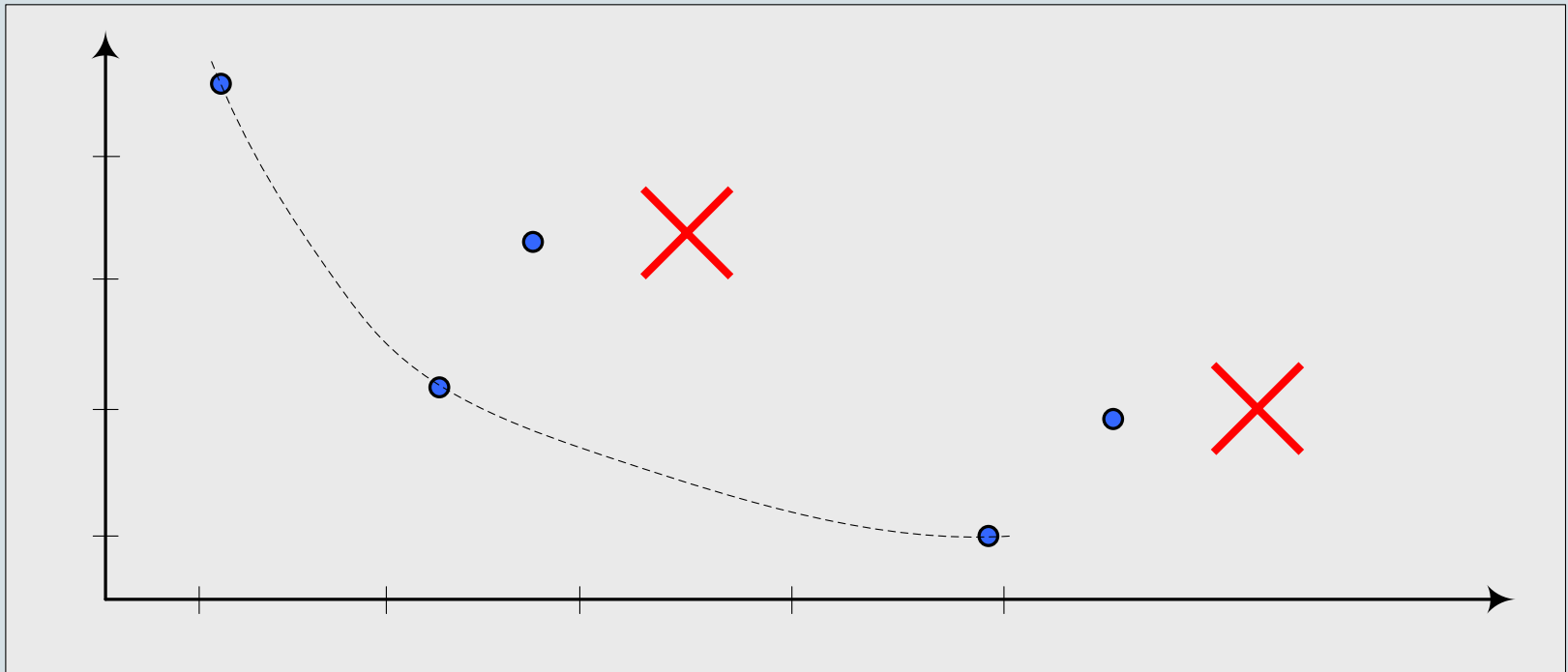


Evaluating Architectural Alternatives

M.Sc. projects



Pareto-analysis of Alternatives



Concluding Remarks

- Predictable Assembly
 - requires
 - the composition of independently developed models
 - the explicit modeling of the composition logic and the composition framework.
 - enables
 - design exploration

Concluding Remarks CBSE

- CBSE is a paradigm for design and construction of software
- CBSE is aimed at improving development–productivity (reduce time and effort)
- Components are building blocks that are part of a component model. May or may not be OO.
- If you have components, you can reuse them
- CBSE requires changes in development processes and development techniques
 - Separate system development from component development
 - Validation & certification are open problems
- There are different approaches for composition of software
 - Techniques for reasoning about composition are being developed

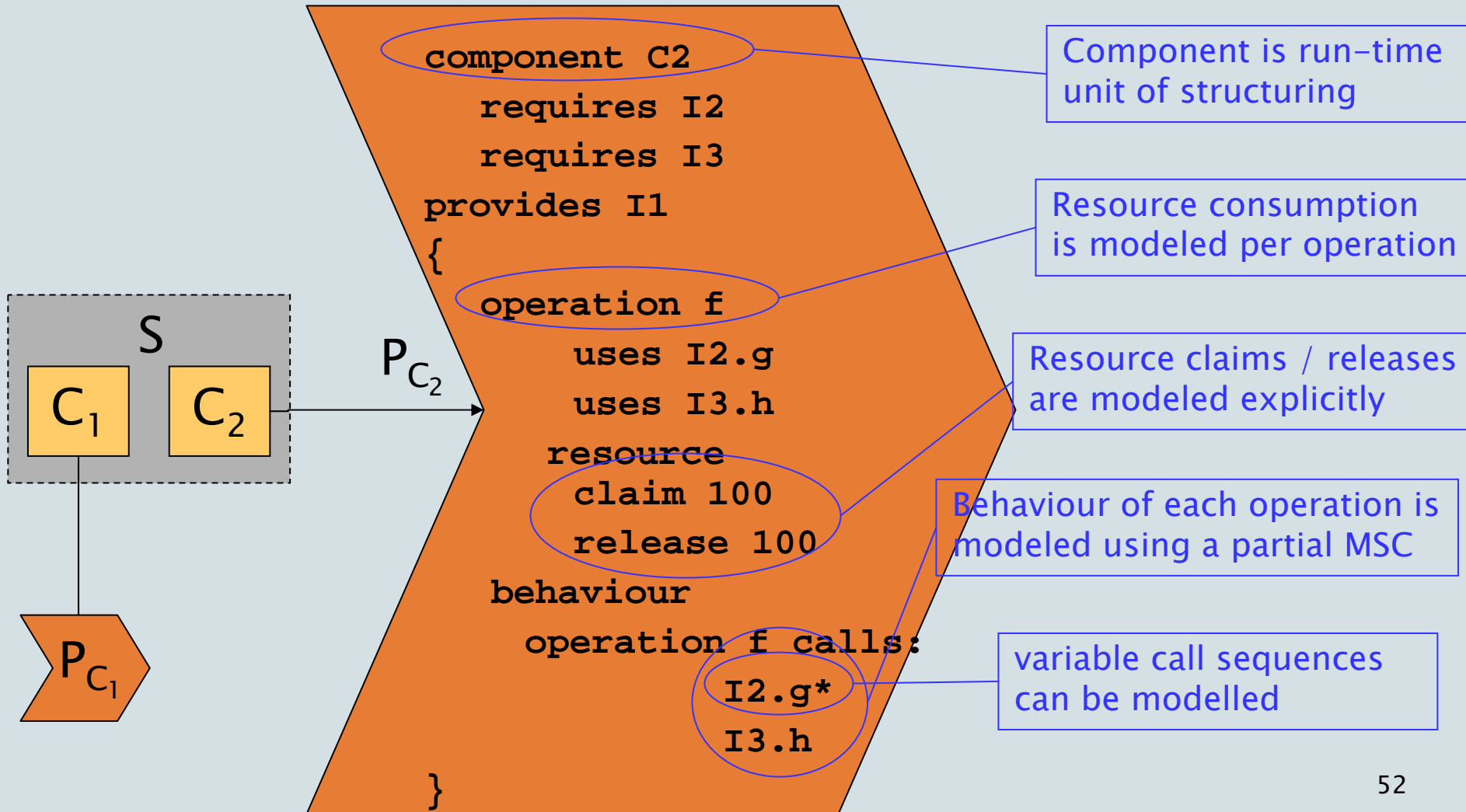
Concluding Remarks

- What have you learned?
- What would you like to see improved?
 - Book/notes/slides/...

Discussion / Questions

?

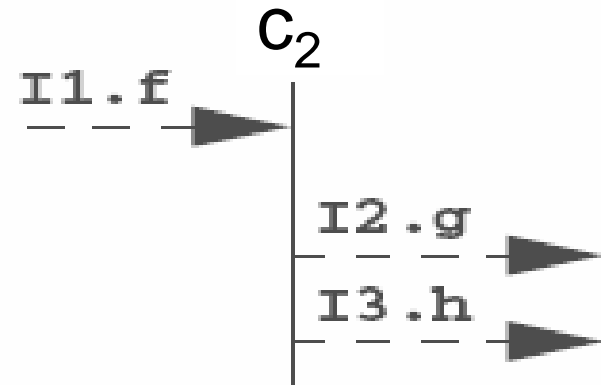
Component specification



Service specification

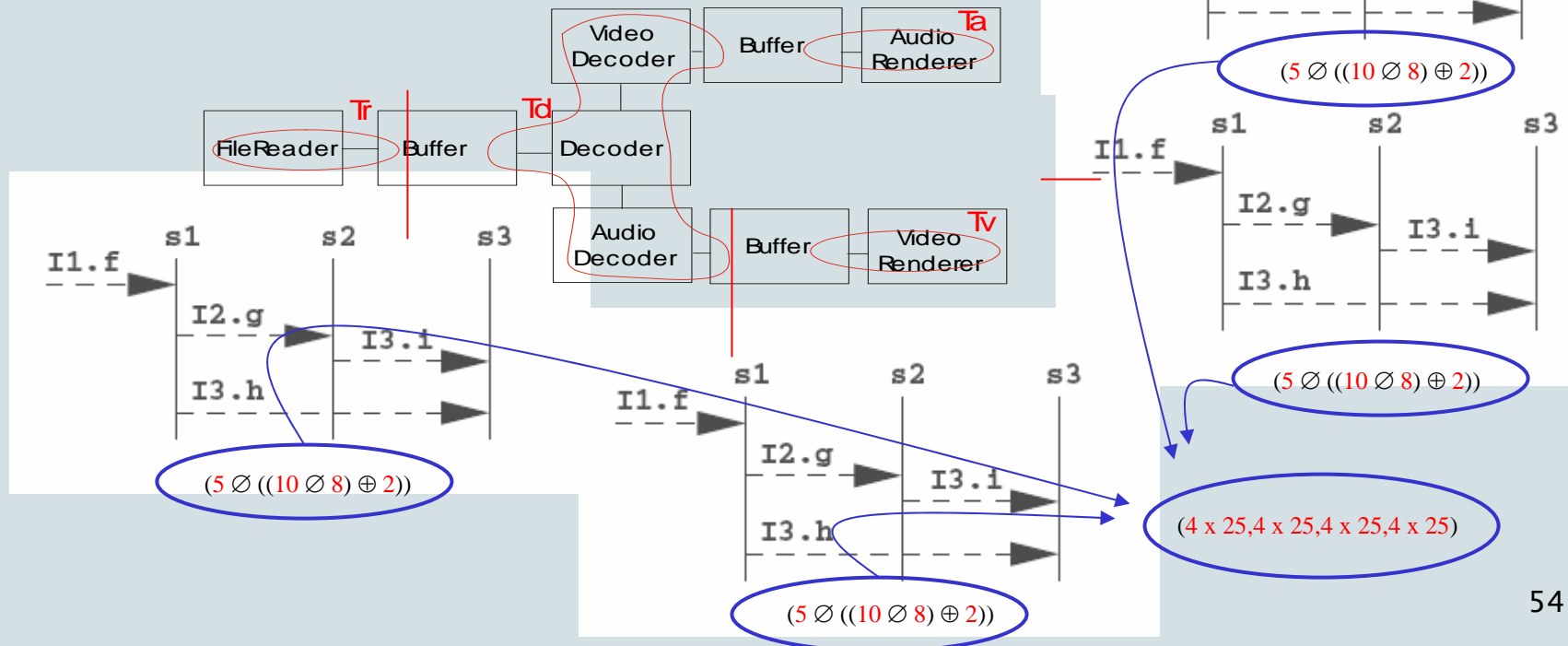
```
component c2
  requires I2
  requires I3
  provides I1{
    operation f
      uses I2.g
      uses I3.h
      resource claim 100
      release 100
    behaviour
      operation f calls:
        I2.g*
        I3.h
  }
```

Operational behavior
forms **partial**
call graph or MSC



Dealing with Concurrent Tasks

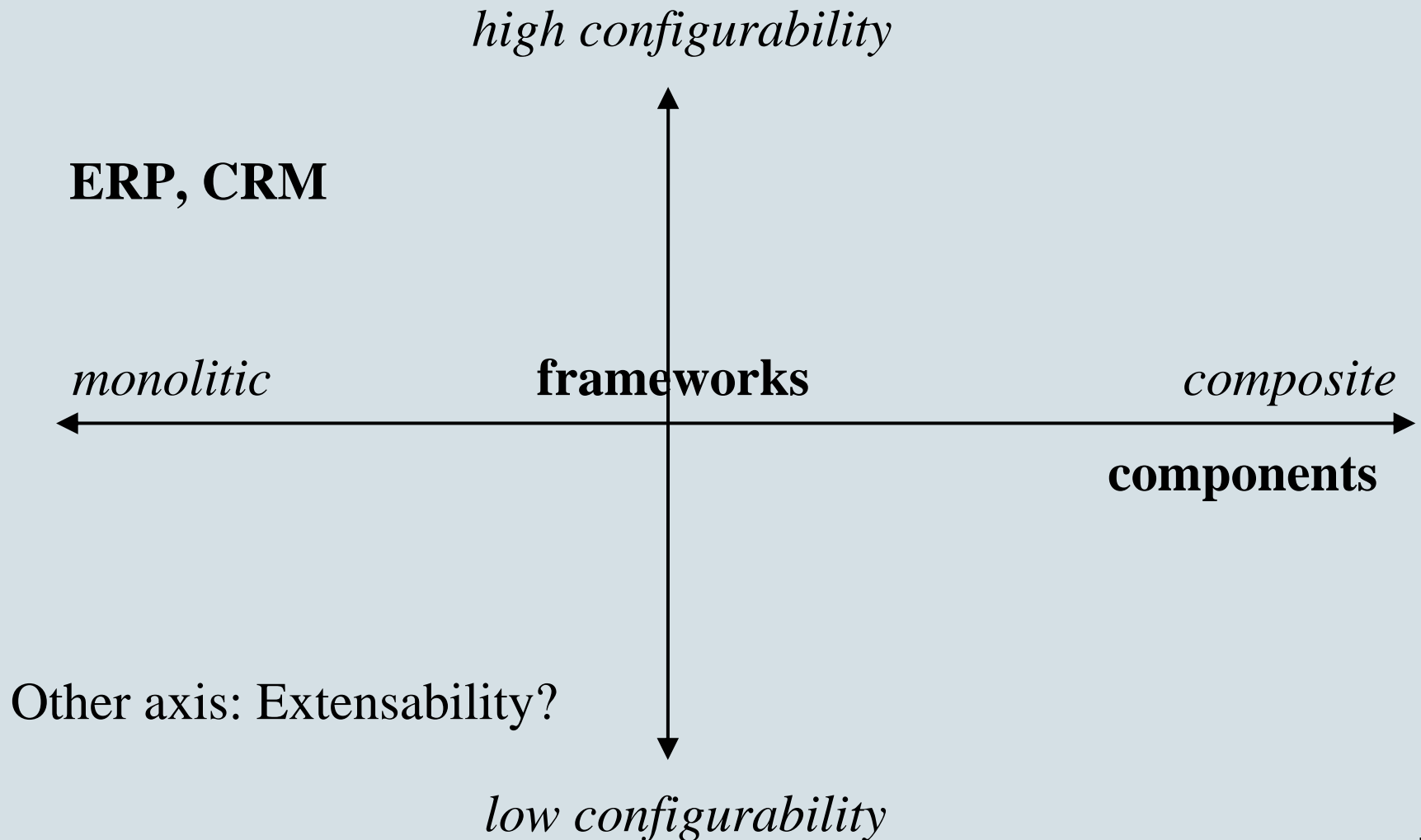
- Task based accumulation of resource consumption
- Accumulation of per task resource consumption based on scheduling policy



Evaluation of the method

- Compositional (supports third-party binding)
- The method can be applied to different types of resources
- The method can support different types of analyses (WC, BC, ...)
- Method can be used throughout development cycle (design / implementation / deployment).
- Scenario's
 - Do not give 100% guarantees as usual in formal methods
 - Do allow incremental accuracy: focus on what is important

Spectrum



Composition time trade-offs

The later composition takes place, the more flexibility the component model/system provides.

Run-time composition requires

- more functionality in the framework
 - more complex framework
 - more expensive to develop
 - use more resources (important in embedded systems)
- more meta-information in the component
 - more expensive to develop
 - more expensive to download
- indirection during run-time
 - performance penalty

Run-time binding

Run-time binding needs run-time support from the component framework. Hence, this needs to be foreseen in the component model.

This design decision whether or not to support run-time binding, affects all systems that are built using the resulting component model.

Many ‘component approaches’ that try to migrate from existing technology are not able to integrate run-time binding.