Tampereen yliopisto
Tampere University

Large Scale Software Design
Components

Components and interfaces

Idea of components: rationalising software engineering

What is a software component?

Components as software units

Interfaces

Tailoring components

Conclusions

Rationalising software engineering

Building products out of components

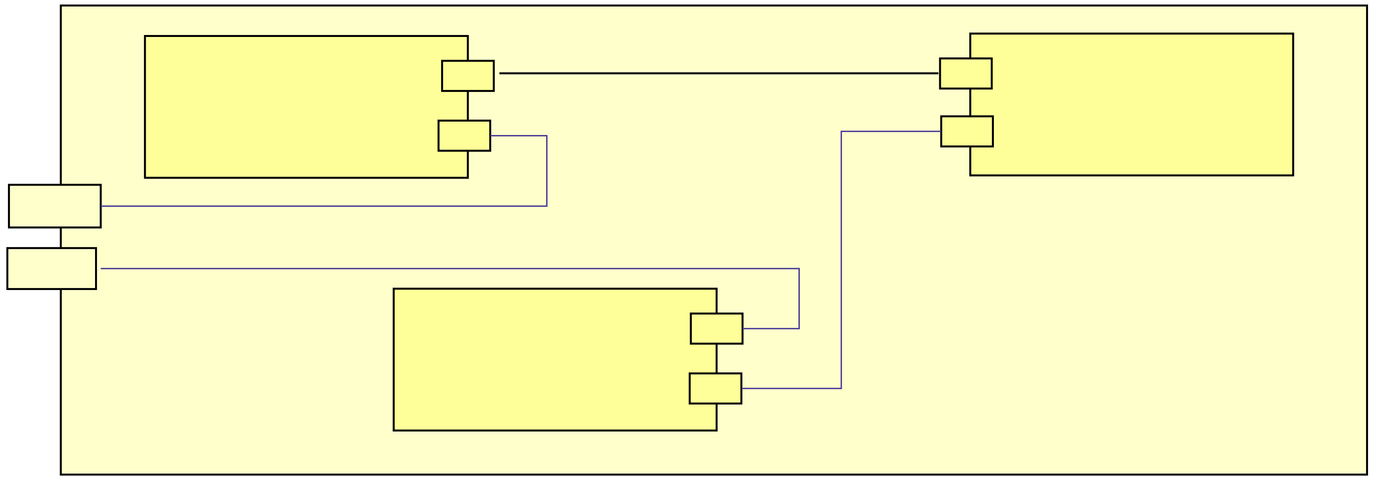
- **Products will be more reliable**
- **Products are easier to make**
- **Makers are easier to educate**
- **Component markets and competition decreases prices**

Applied almost all areas of technology

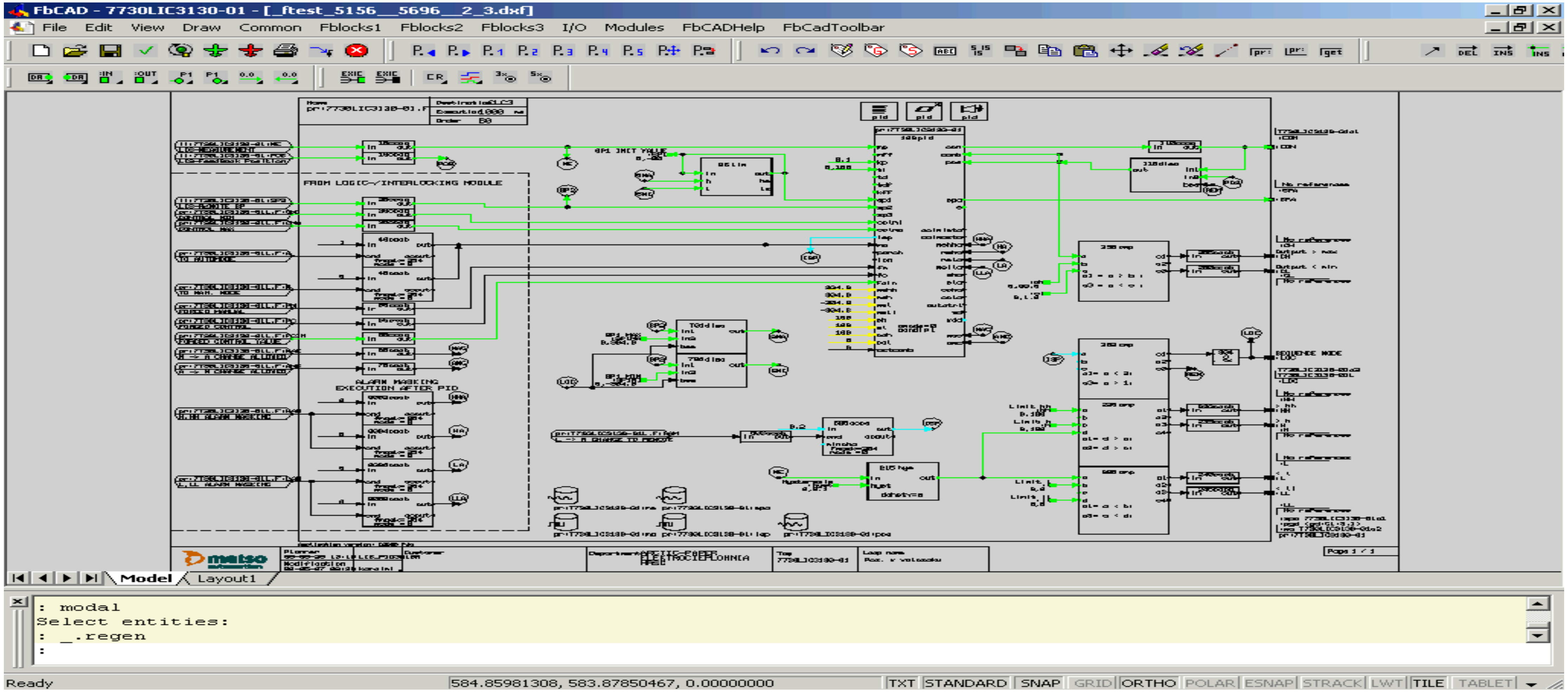
Applications made of components

Visio: the application is composed from existing components using parameters and defining connections.

Tools: scripts, XML, visual tools



Example: Process automation (Valmet)



```
: modal
Select entities:
: _.regen
:
```

```
Ready        584.85981308, 583.87850467, 0.00000000        TXT  STANDARD  SNAP  GRID  ORTHO  POLAR  ESNAP  STRACK  LWT  TILE  TABLET
```

What is a software component?

Component = independent software unit giving services through well-defined interface.

Szyperski:
A software component is a unit of composition with contractually specified interfaces and explicit context dependencies only. A software component can be deployed independently and is subject to composition by third parties.

Properties of components

Level of independency

- **Component typically assume run-time support of given infrastructure**
- **Component can assume given service environment (required interfaces)**

Ways to introduce

- **Can be introduced during development process, linking time, starting time or during usage (source code or binary).**

Size

- **Varies from an object to an subsystem**

Standardisation

- **Standardisation of interface in application area, standardisation of infrastucture.**

Technology-specific properties

- **Metadata**

Tampereen yliopisto
Tampere University

Challengers of components

How is the component implemented?

How is the component interface described?

How to ensure that the component corresponds with its interface, keeps its promises?

How to maintain system made of components?

Challengers continues...

How to combine components, form bigger bodies?

- **Problems of combining components**
- **Types, parameters**
- **Functionality**
- **Concurrency, timing**
- **Using of resources**
- **Security**
- **Safety**

Components as software units

Basic unit of architecture. Unit of:

- **Functionality – what part is responsible on given functionality?**
- **Reuse – which parts are common in different products?**
- **Product configuration – which parts belong to the product?**
- **Introduction – which parts can be utilised separately?**
- **Adaptability – which parts can be substituted?**
- **External development – which parts can be obtained elsewhere?**
- **Task division – which parts are produced by given persons or units?**

Component-based systems

Firefox, plug-ins and add-ons as components

- **Strictly defined interface between Firefox and plugin (does not need information about internal structures of Firefox)**
- **Separated components: failing plugin does not crash Firefox or other plugins.**

Linux and drivers

Bus-based embedded systems

- **E.g. Cars (CAN)**

Example of component use

Scintilla (http://www.scintilla.org/)

- **Text editor component, suitable especially for source code editing**
- **Free**
- **Geany, Komodo, Notepad++, Notepad2, Programmers Notepa, MySQLWorkbench**
- **Wrappers: .NET, Qt, wxWidgets**

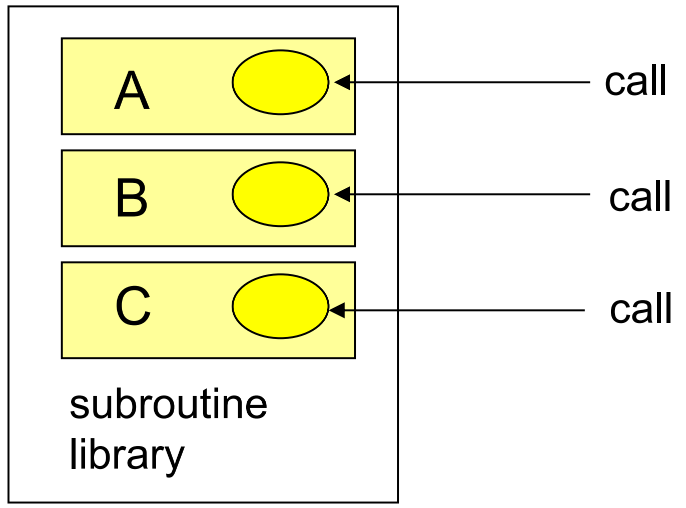
Documentation examples: http://www.scintilla.org/ScintillaDoc.html

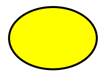
Interfaces

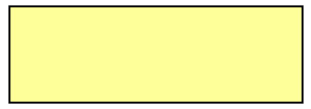
Component / interface concept evolution to object paradigm

- **Subprograms: abstraction of functionality**
- **Modules: hiding information**
- **Classes: extendable modules**
- **Abstract classes: classed without implementation**
- **Multiple inheritance: several abstractions in one implementation**
- **Components: different interfaces**

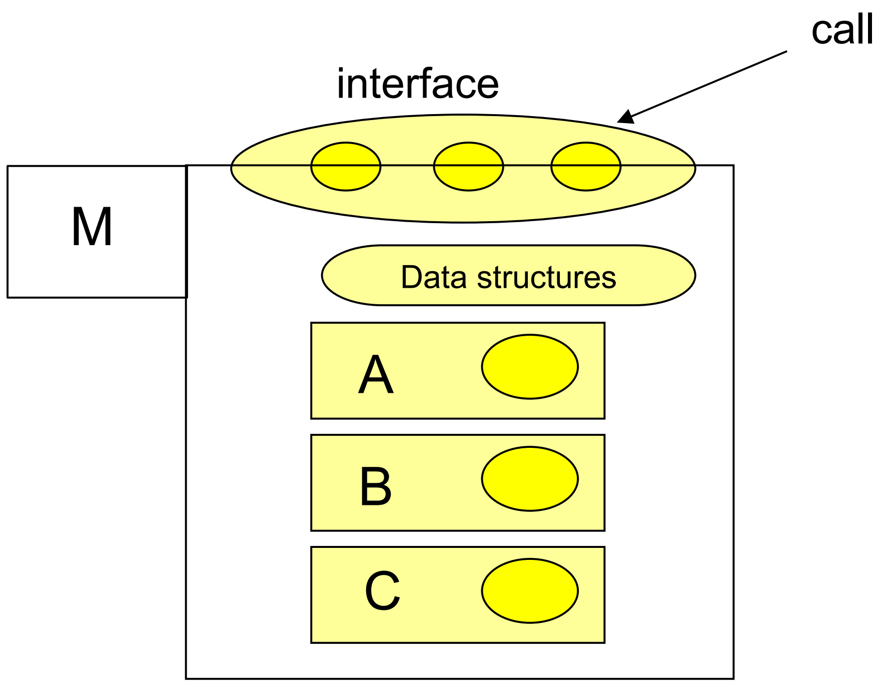
Subroutines: abstraction of functionality




How to call the service (signature)

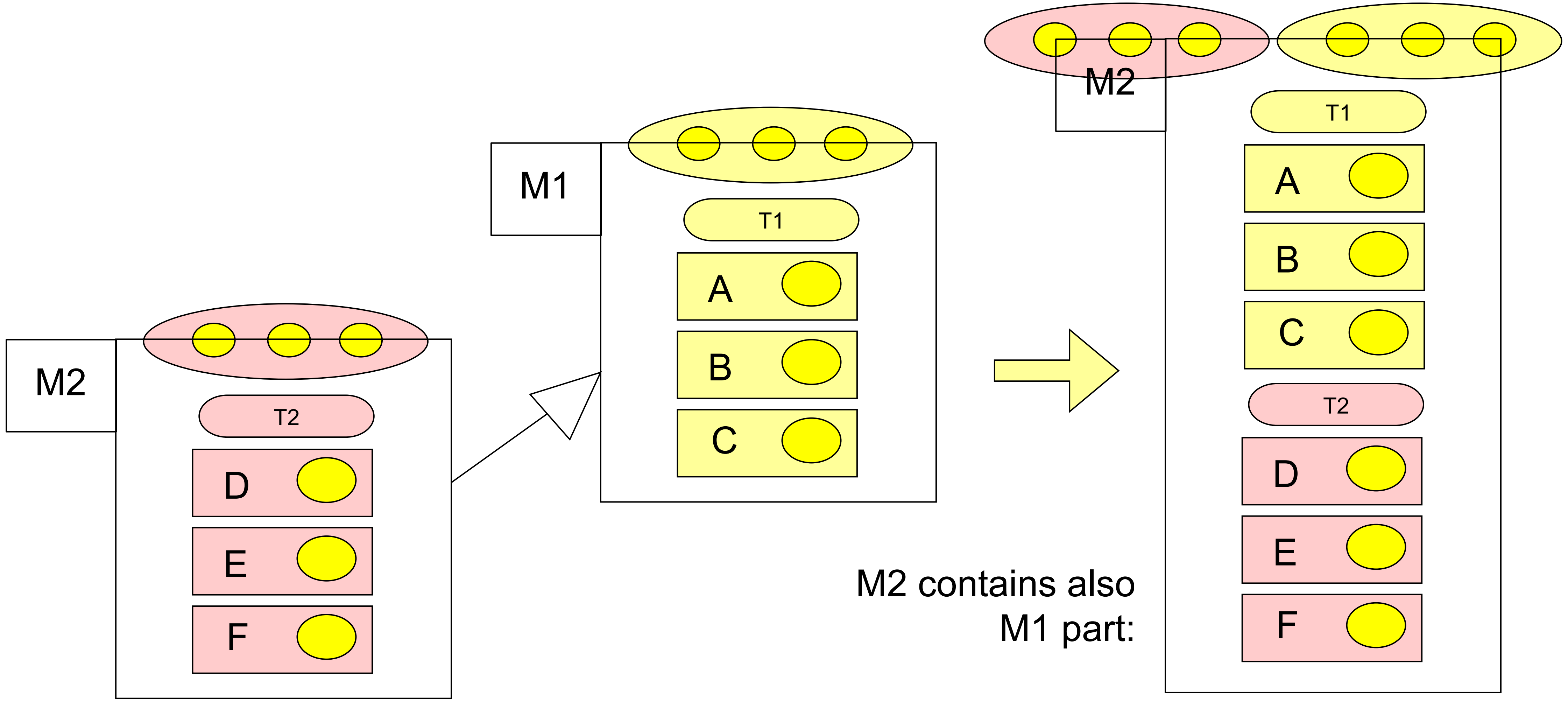

Implementation of service

Modules: hiding information



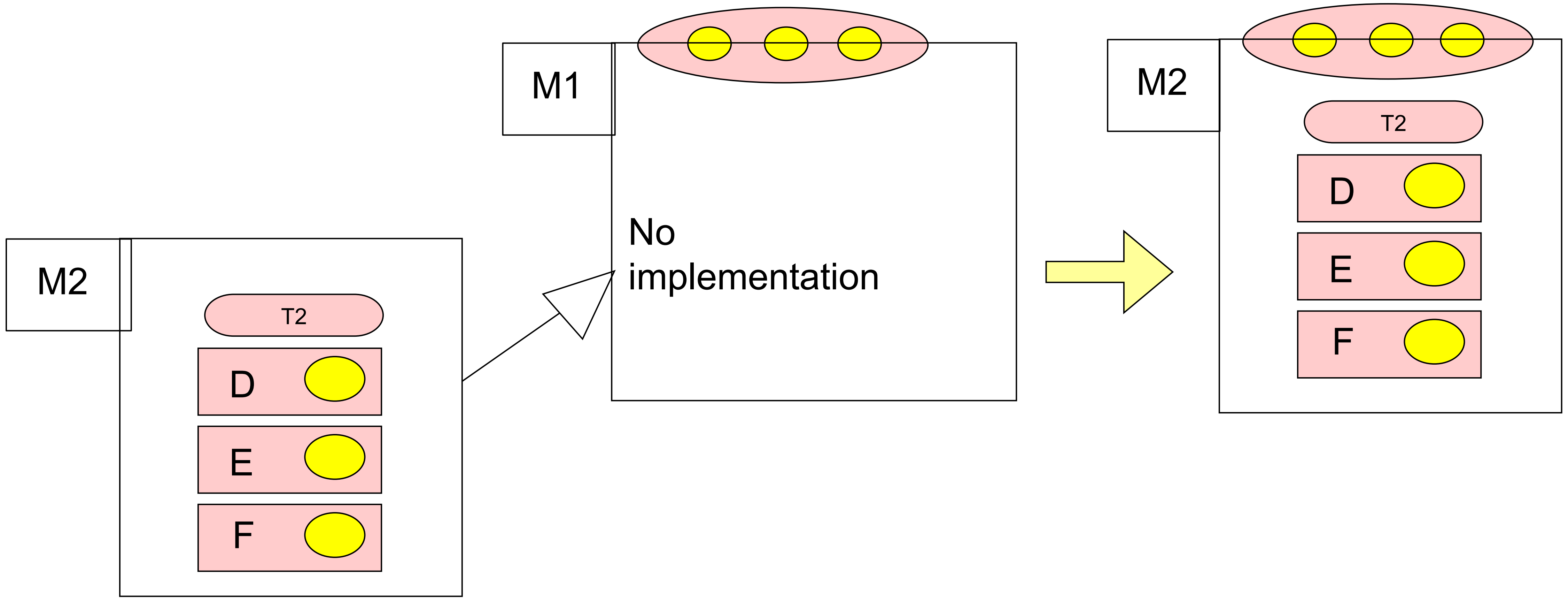
E.g. Ada programming language

Classes: extendable modules

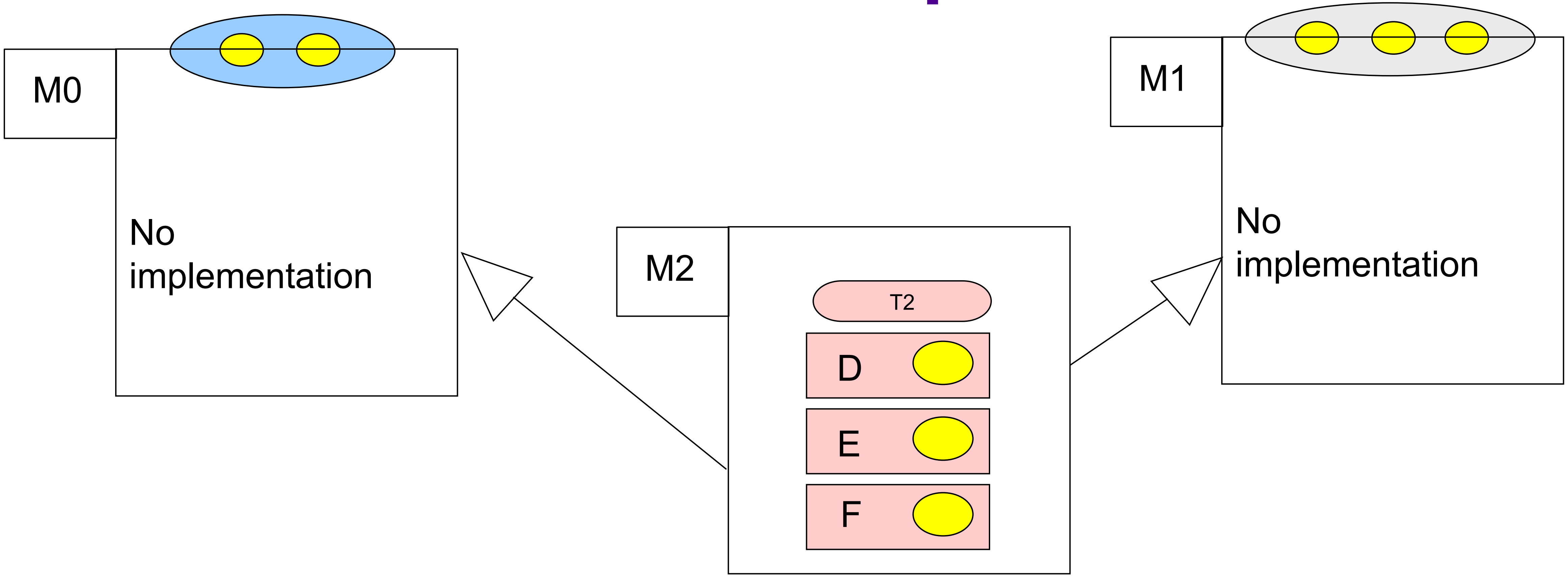


M2 contains also M1 part:

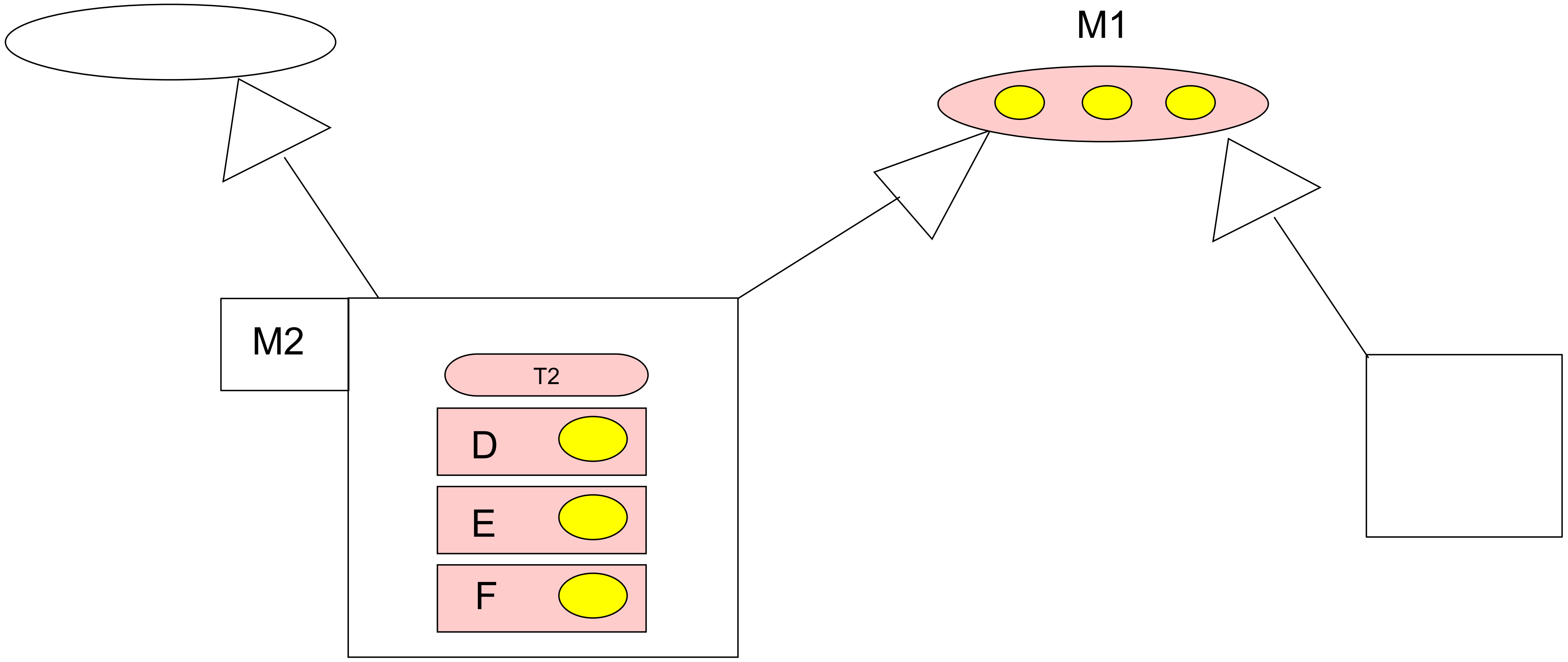
Abstract base classes: classes without implementation



Multiple inheritance: several abstractions in one implementation

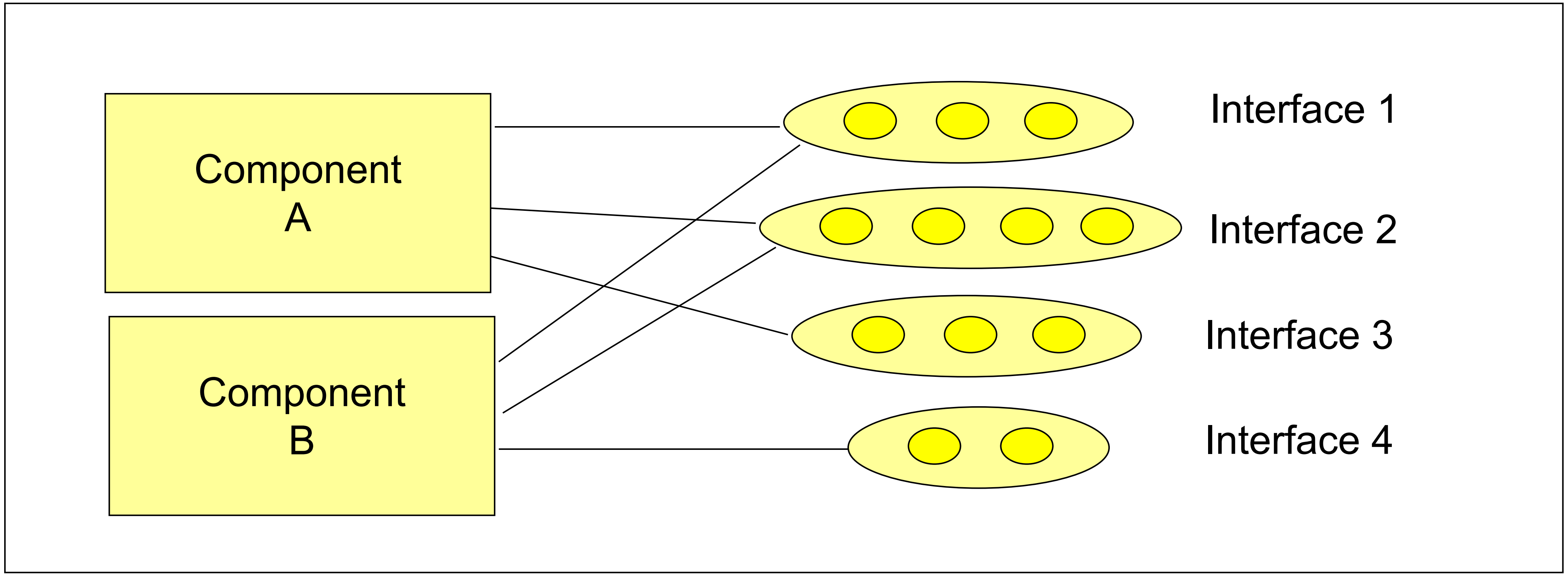


Components: separate interfaces



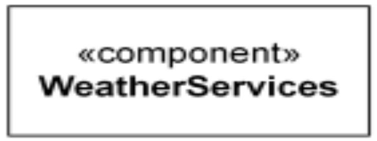
Tampereen yliopisto
Tampere University

Interfaces as separate software units

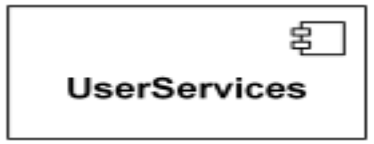


Components and UML

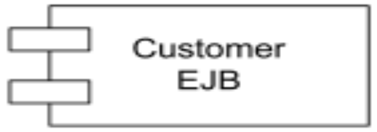
<<component>> –stereotype



Using component symbol



Component before UML 2, still OK



Provided and required interfaces

Component may have two different relationships with an interface:

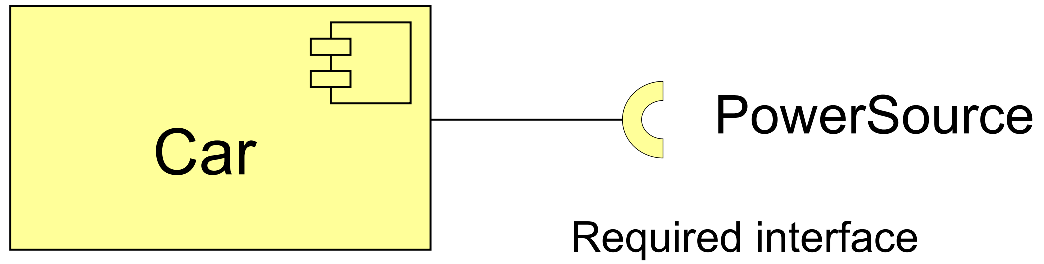
Interface provided by the component

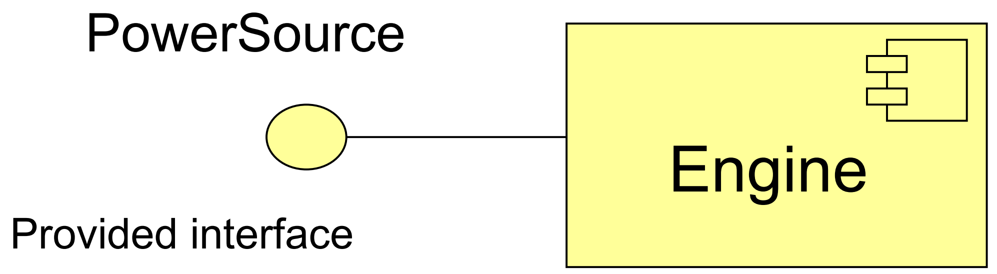
- **The component provides the services of the interface**

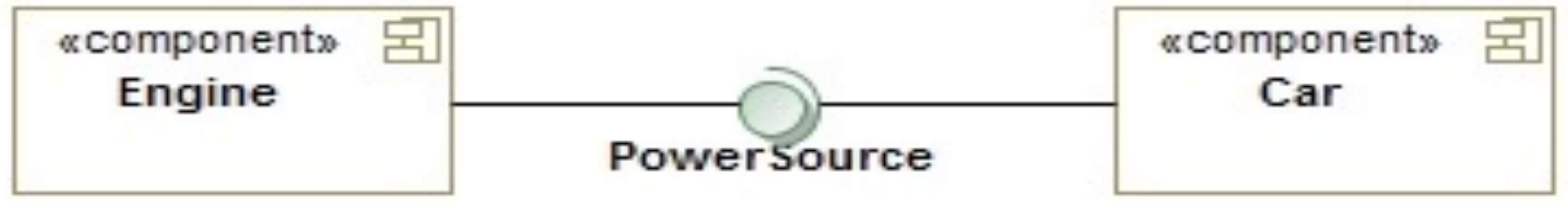
Interface required by the component

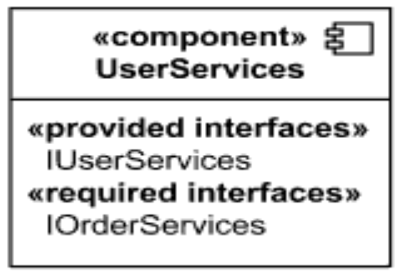
- **The component needs the services of the interface.**

Provided and required interfaces in UML 2.x

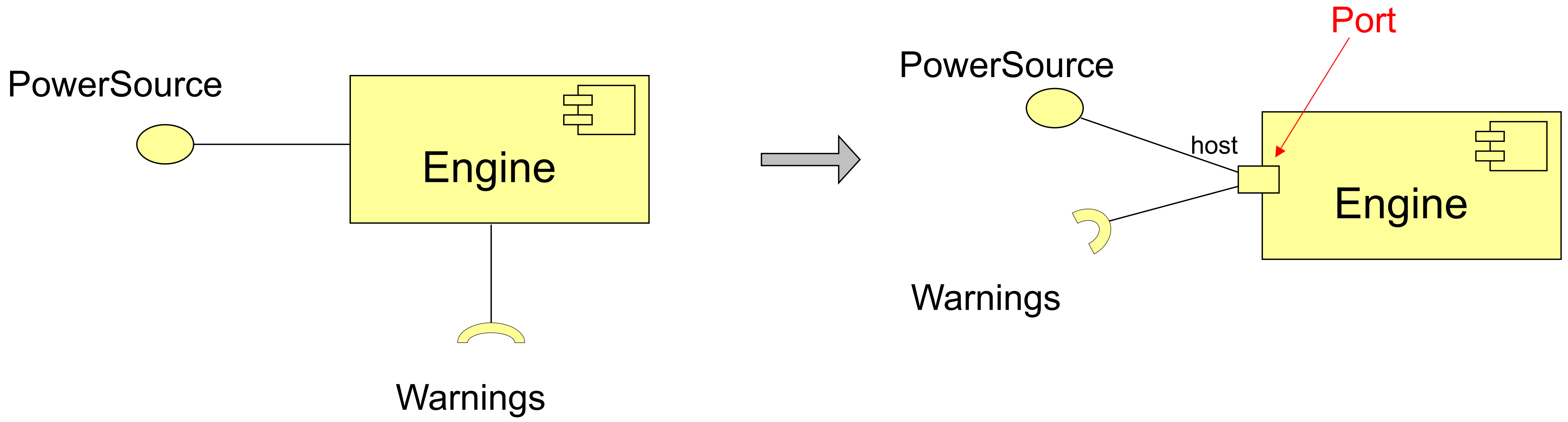








Ports in UML 2.x



Port is a contact point enabling interaction between a component and its environment

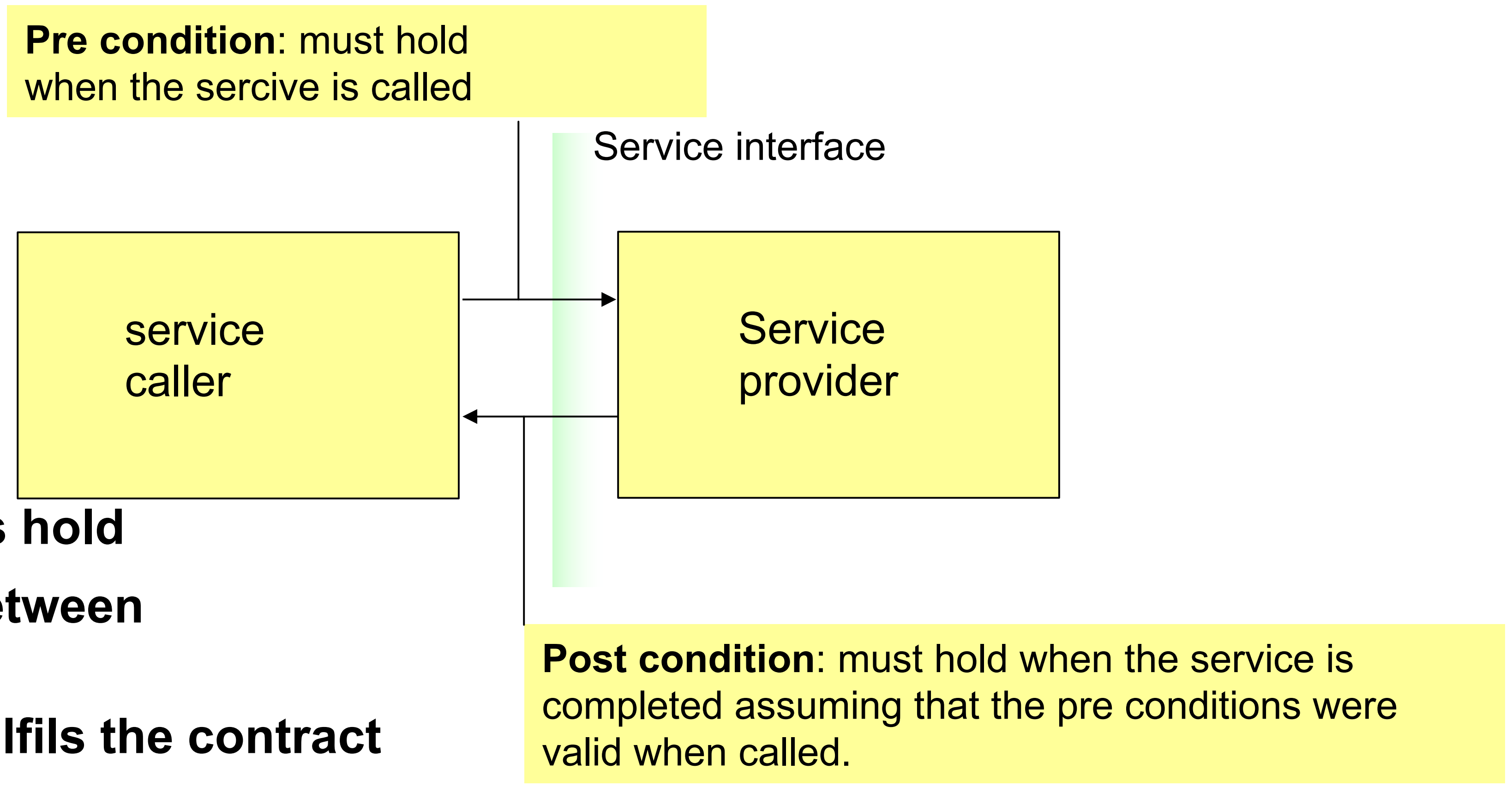
On left, the warning is sent somewhere, on right, it is sent to the same component that used PowerSource

Contract-base design of interfaces

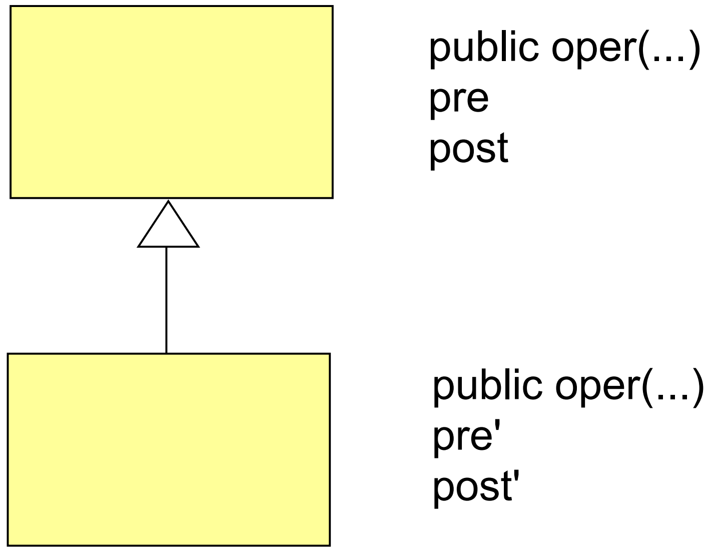
design-by-contract

Pre and post conditions

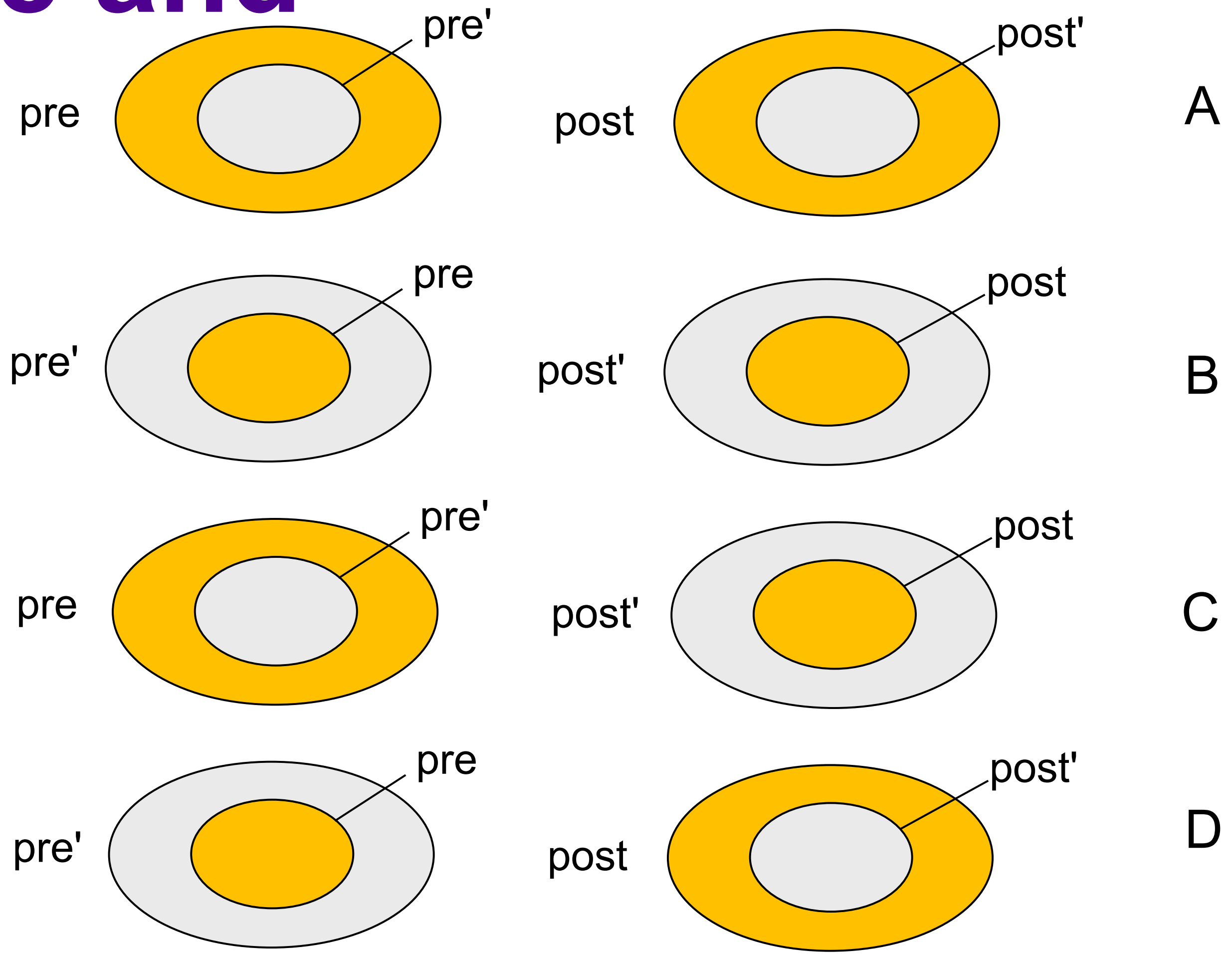
- **Define the meaning of service**
- **Allow changing of participants, if conditions hold**
- **Define responsibilities between participants**
- **Service provider either fulfils the contract or causes an exception**


Pre condition: must hold when the sercive is called
Service interface
service caller
Service provider
Post condition: must hold when the service is completed assuming that the pre conditions were valid when called.

Inheritance and pre and post conditions



- Correct alternative? A, B, C, D or F?



Pre and post conditions of the subclass have to be identical to the ones of the base class. F

Tampereen yliopisto
Tampere University

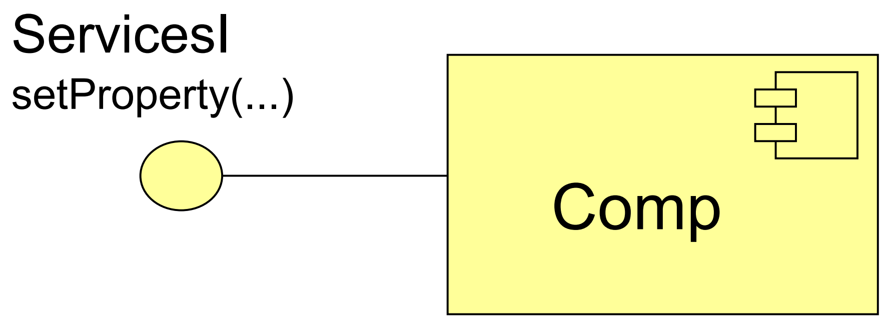
Tailoring components

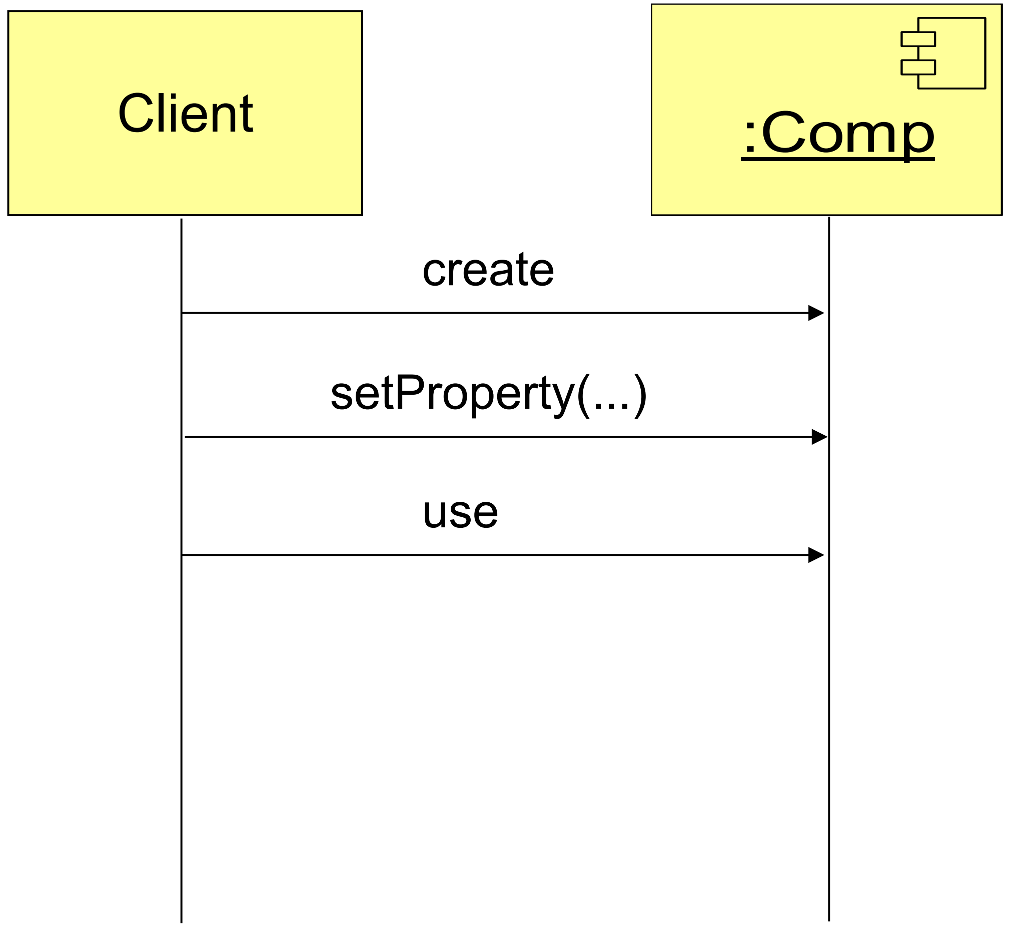
Changing the initial state of a component

Providing or changing the implementation of required interfaces

Subclassing.

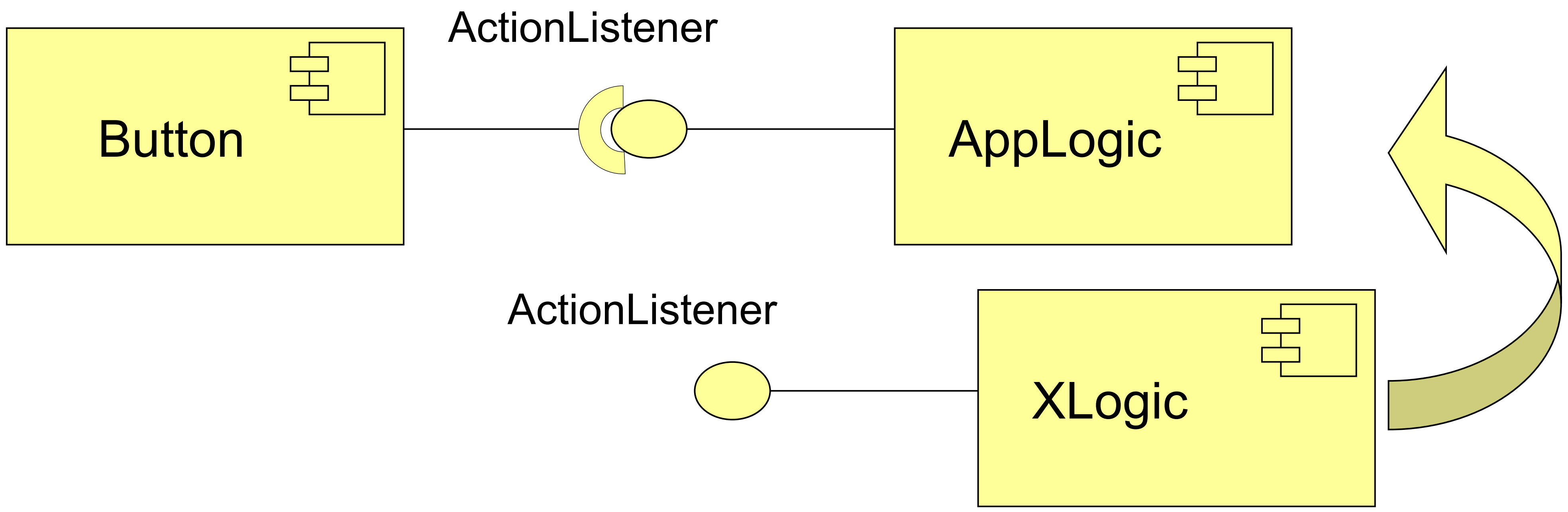
Changing the initial state of a component





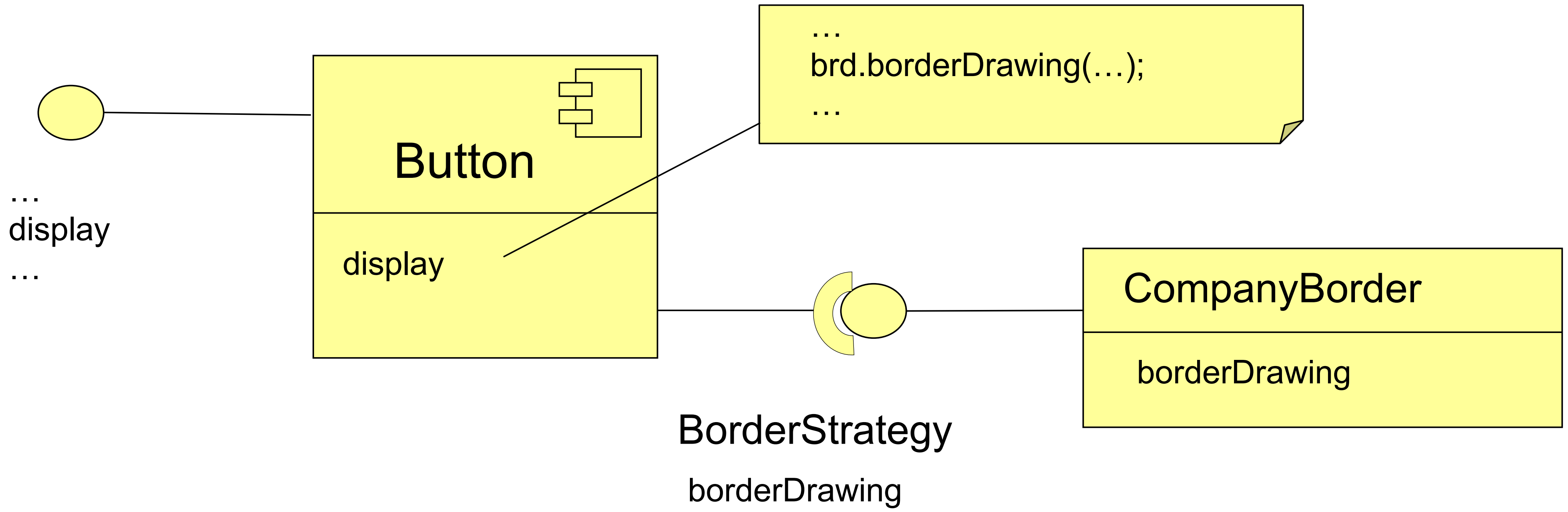
The component's user sets the initial state before use.

Providing or changing the implementation of required interfaces (dependency injection)



Service used by the component is changed either in development or run time (static or dynamic)

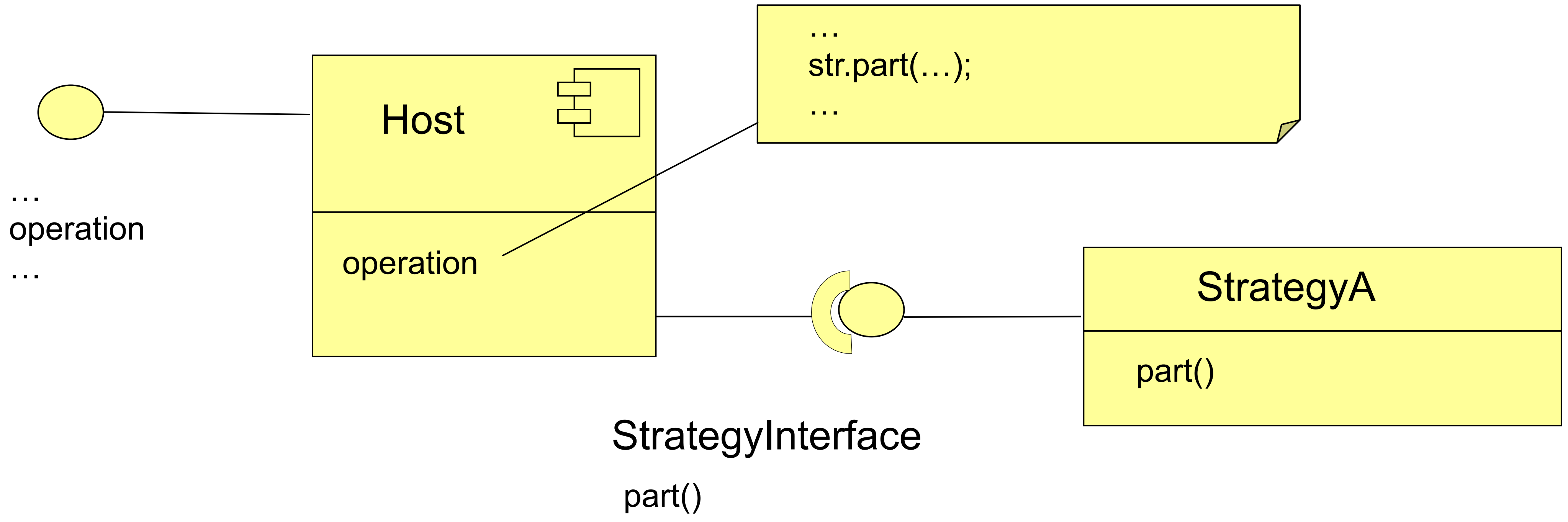
Tailoring one operation



One operation is concerned

Can be changed in run time

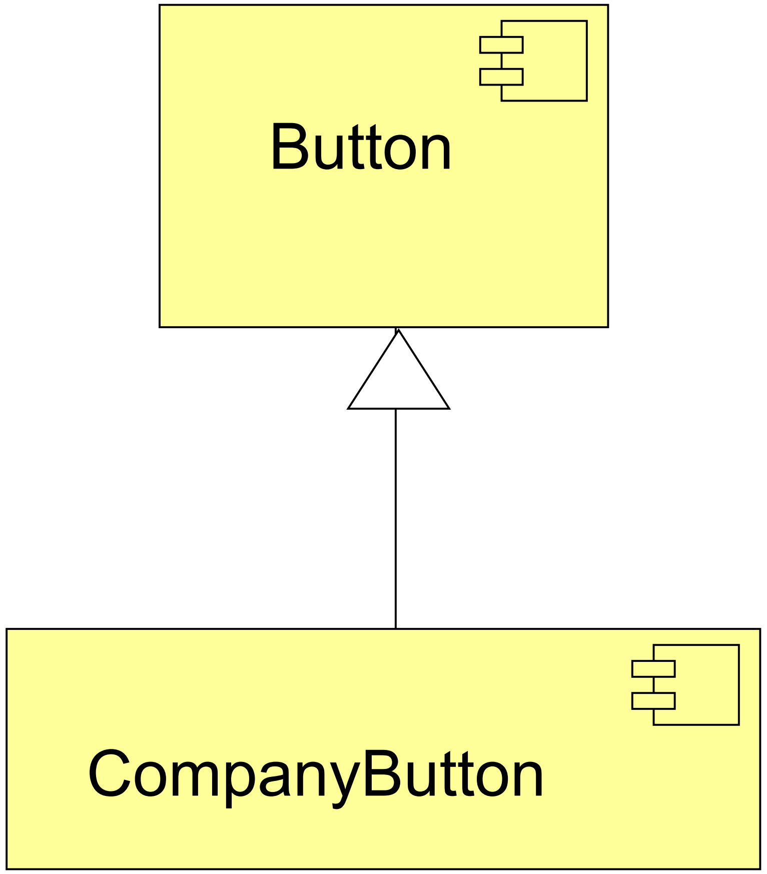
Strategy pattern



Note: only one operation in the interface.

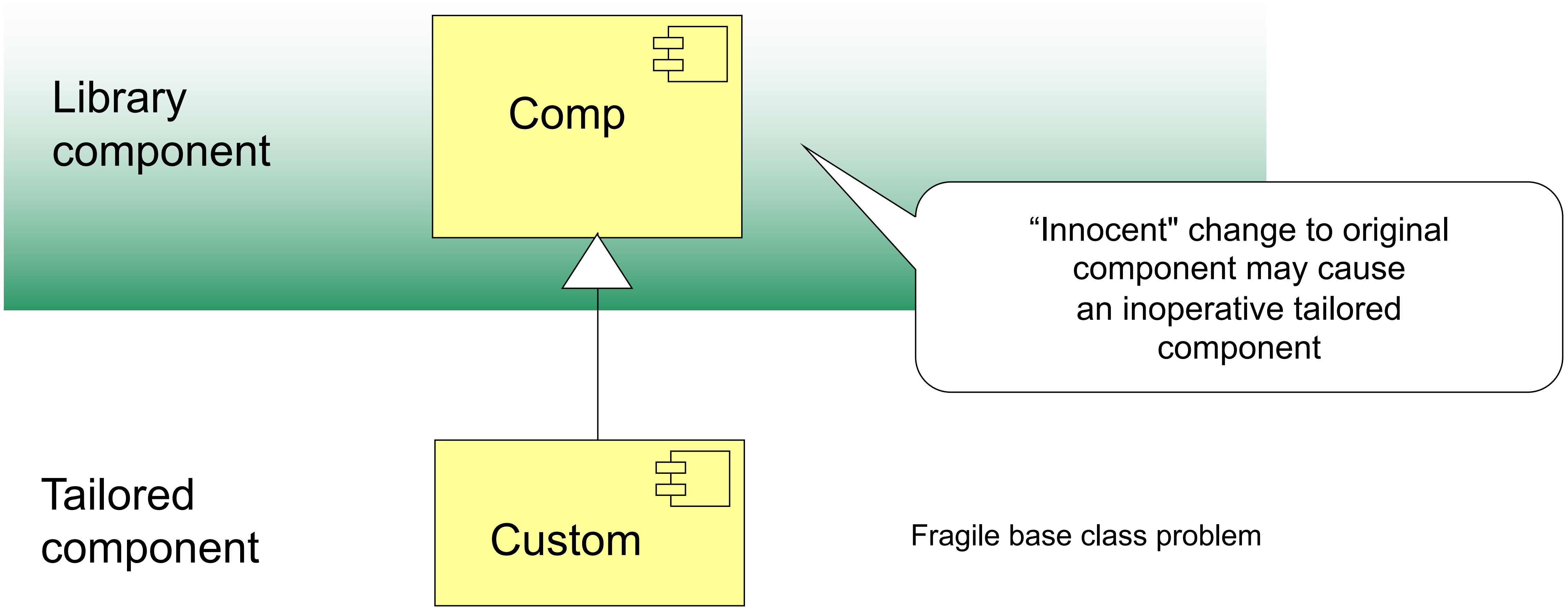
Tampereen yliopisto
Tampere University

Subclassing



Component class is specialised to meet application needs using subclassing (static tailoring).

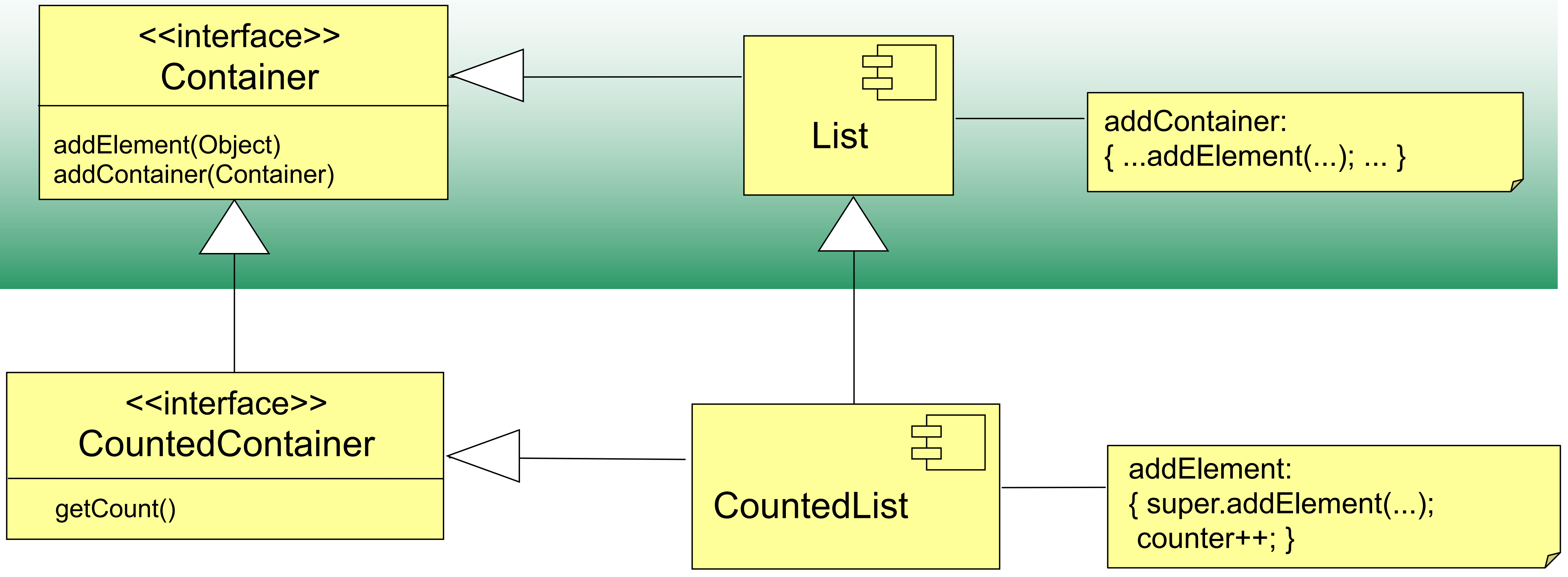
Fragile base class problem


Library
component
Comp
"Innocent" change to original component may cause an inoperative tailored component
Tailored
component
Custom

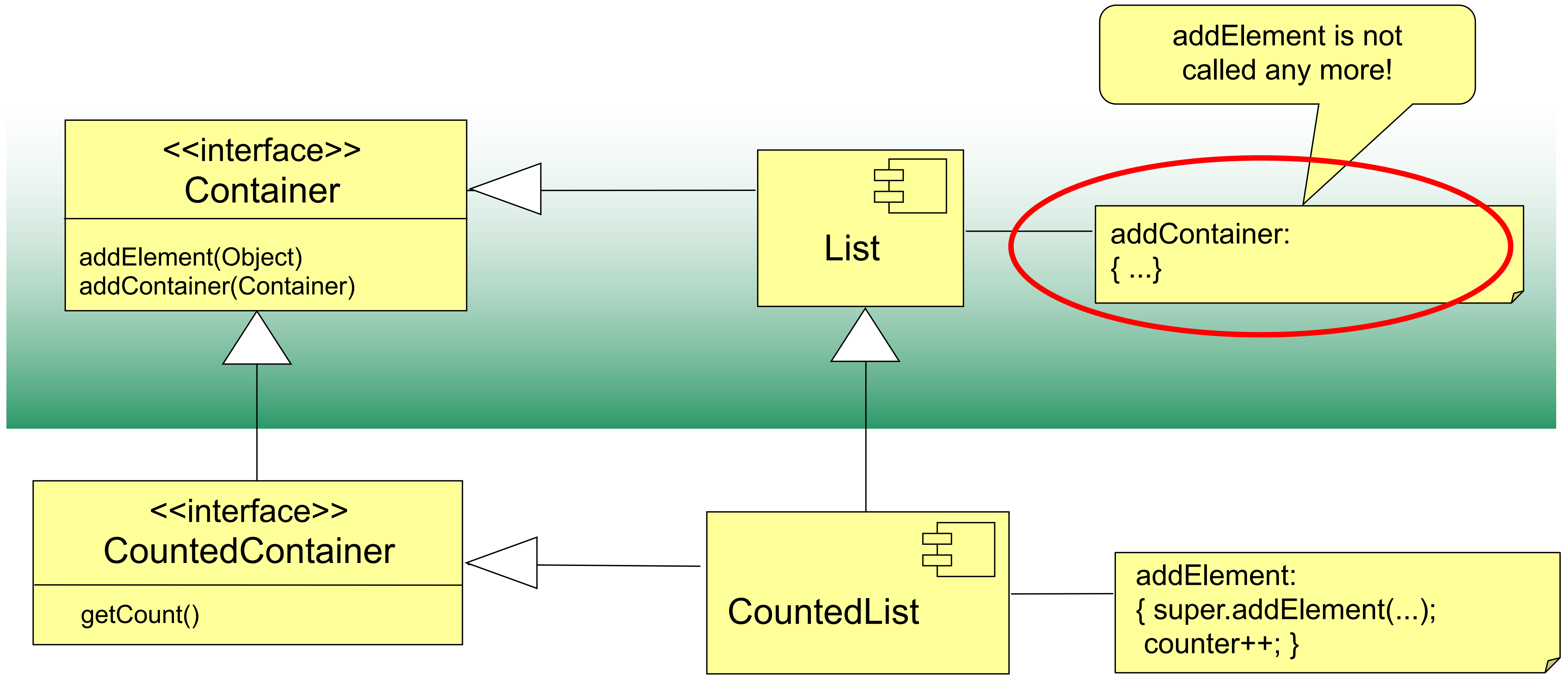
Fragile base class problem

Tampereen yliopisto
Tampere University

Example 1: original list



Example 2: new list


addElement is not called any more!
<<interface>> Container
addElement(Object)
addContainer(Container)
List
addContainer: { ...}
<<interface>> CountedContainer
getCount()
CountedList
addElement: { super.addElement(...); counter++; }

Disadvantages

Problem of fragile base class: it may be risky to inherit the component.

Licensing problems (open source vs. commercial)

Changing interface may break components using the interface.

It is hard to obtain changes to components, even if the interface is not changed.

- **Changing the system is not agile.**

Tampereen yliopisto
Tampere University

Conclusions

Components are basic units of architecture that are connected to each other by provided and required interfaces.

Interfaces should define not only the calling mode but agreement of the usage of the interface (pre- and post-conditions).

Components can be tailored by changing its initial state, changing the components connected to its required interfaces or by inheriting a new specialised component