Tampereen yliopisto
Tampere University

Large Scale Software Design
Interaction of components

Interaction of components

The more dependencies there are between the components, the harder it is to replace a component with another one.

Here we concentrate on some ideas how change dependencies to interactions.

Interaction of components

Role interfaces

Brokers

Facades

Call forwarding

Proxies

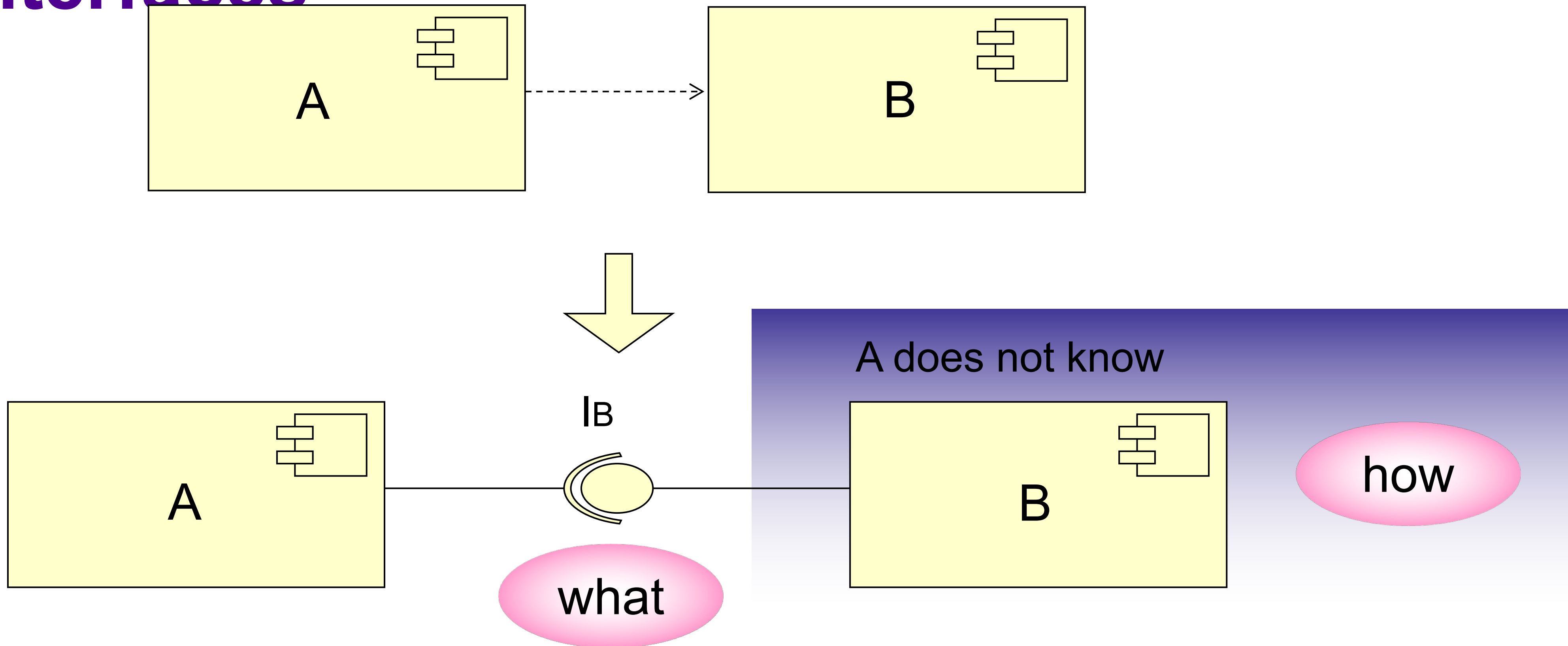
Callbacks

Events

Messages

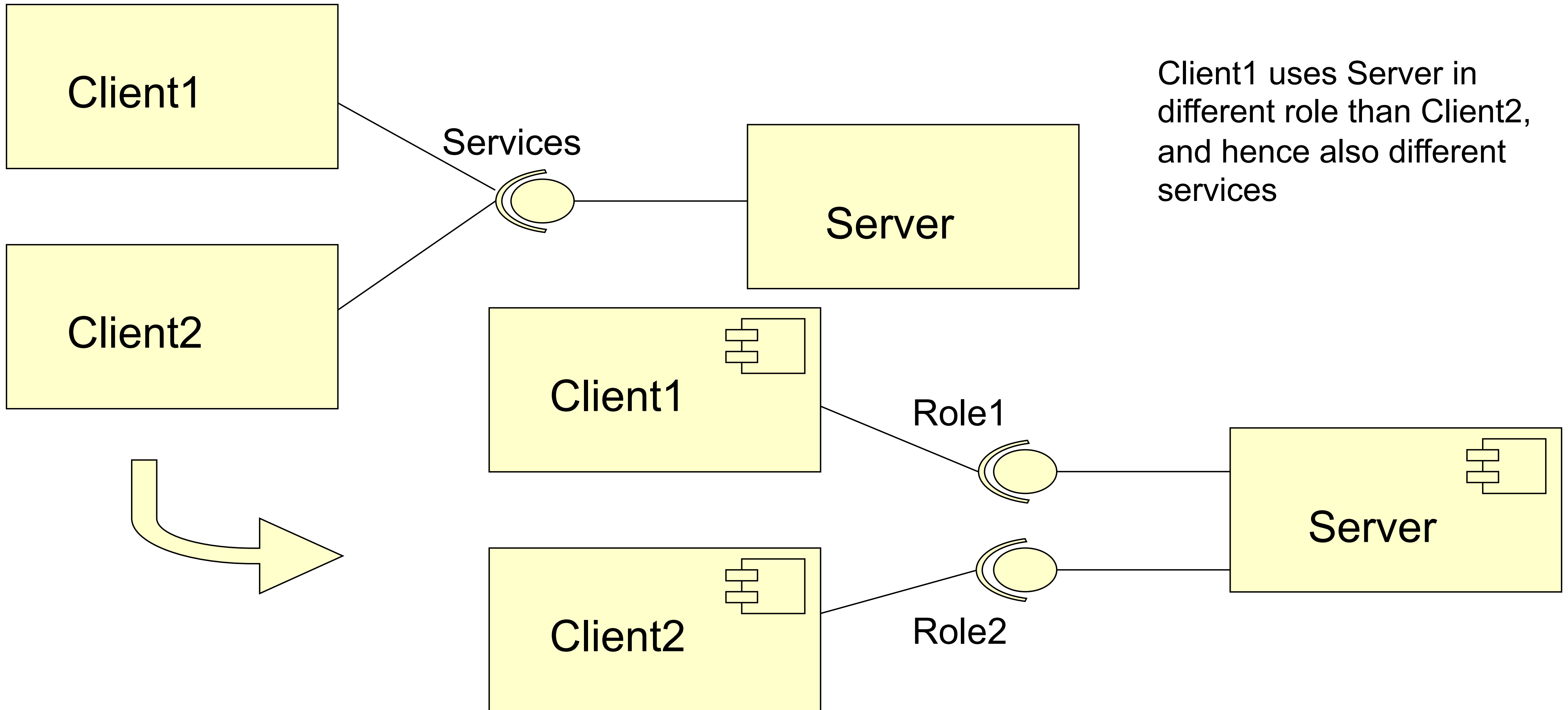
Adapters

Removing implementation dependencies from interfaces



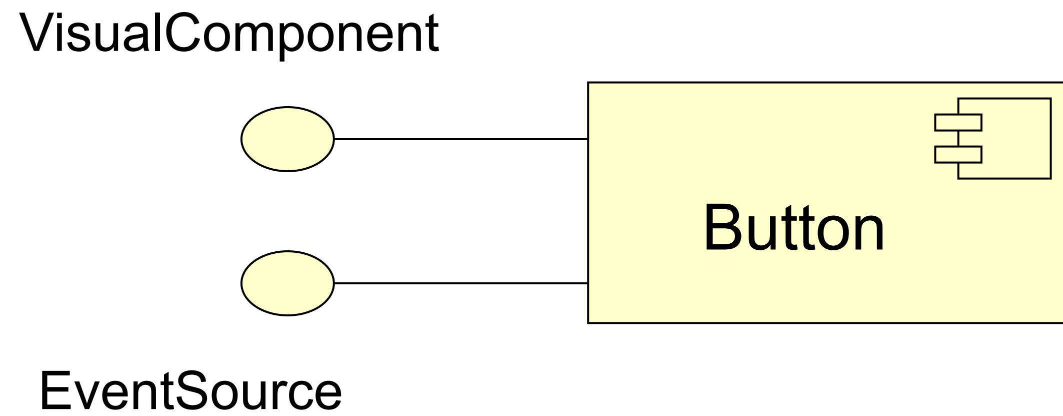
A calls B, i.e. A depends on B. Adding an interface removes dependency.

Role-based interfaces

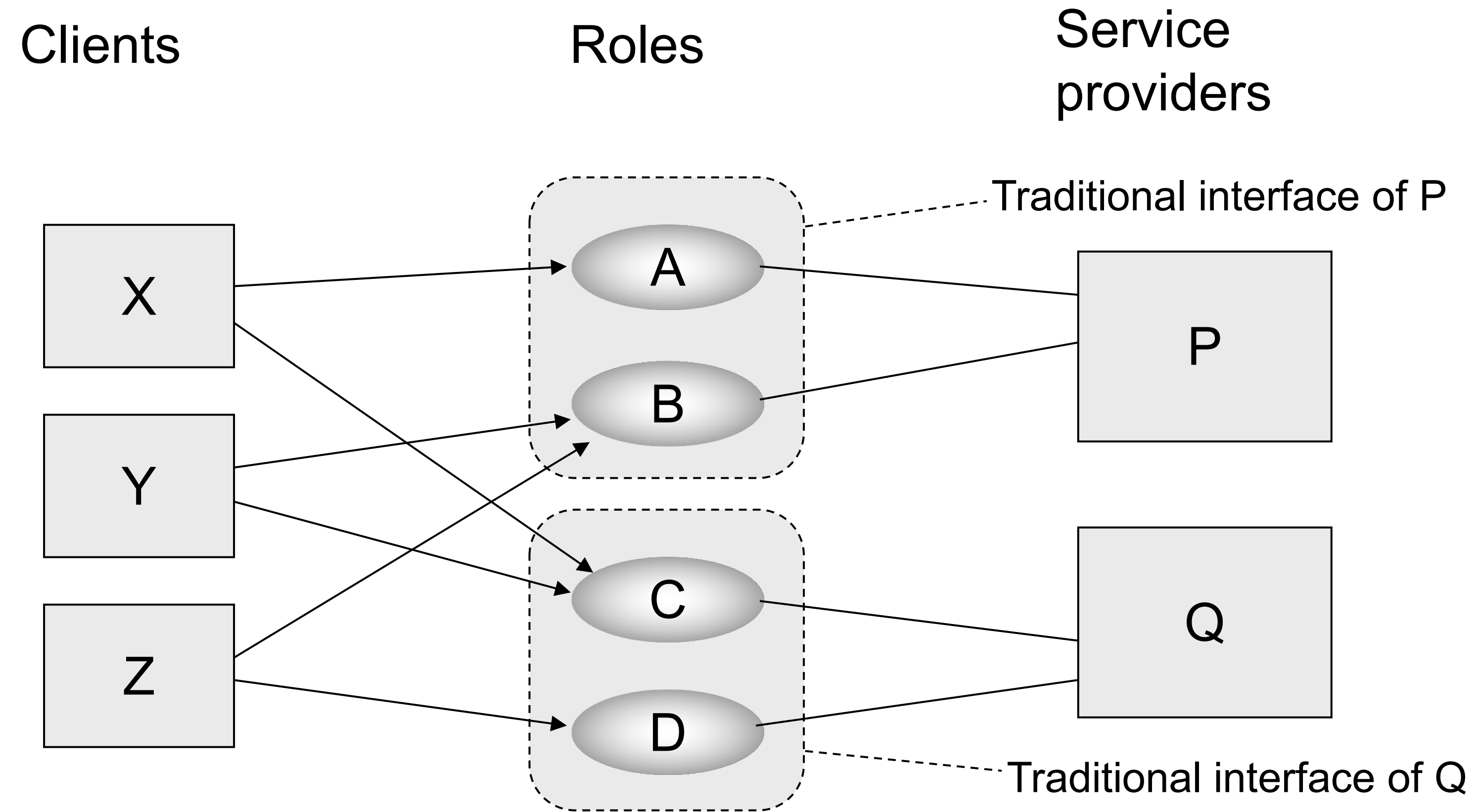


Client1 uses Server in different role than Client2, and hence also different services

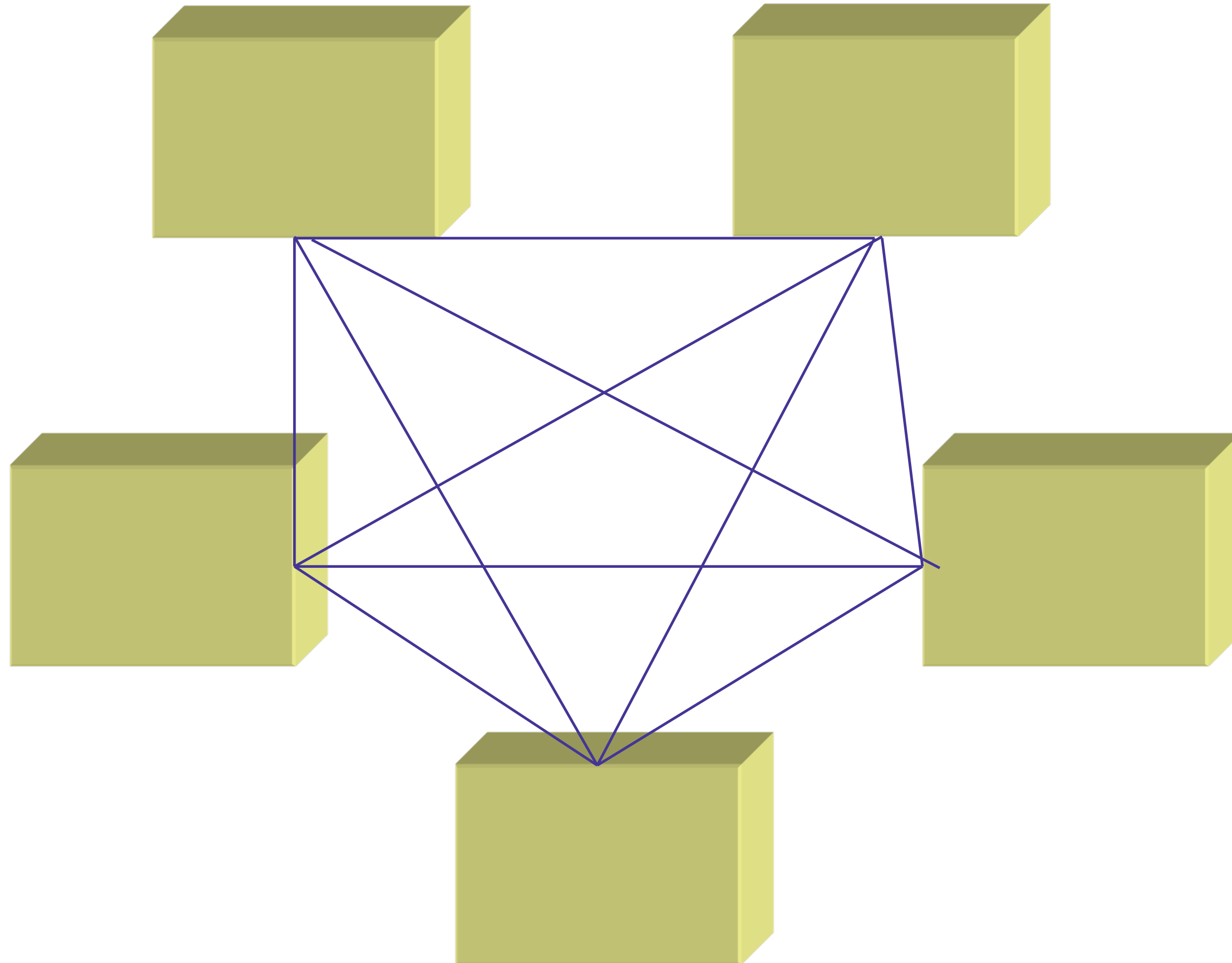
Example



Detailed role interfaces



Managing component interactions



A set of components communicating with each other

Tampereen yliopisto
Tampere University

Managing component interactions

Problems:

- **Dependencies between components are complex and hard to manage**
- **If any connection is changed, all participants have to be changed**
- **It not easy to use components in other context**
- **Following the functionality of the program becomes hard**
- **Dynamic libraries and components are hard to manage**

Example of components

Save As

This PC ▸ Local Disk (C:) ▸ Users ▸ samuel ▸ Dropbox ▸ Ohar ▸ 2015 ▸ Luennot ▸

Search Luennot

Organise ▾ New folder

Desktop
Downloads
ohjsuun_kurssika
easimoln
Recent places
Dropbox

Homegroup
samuel

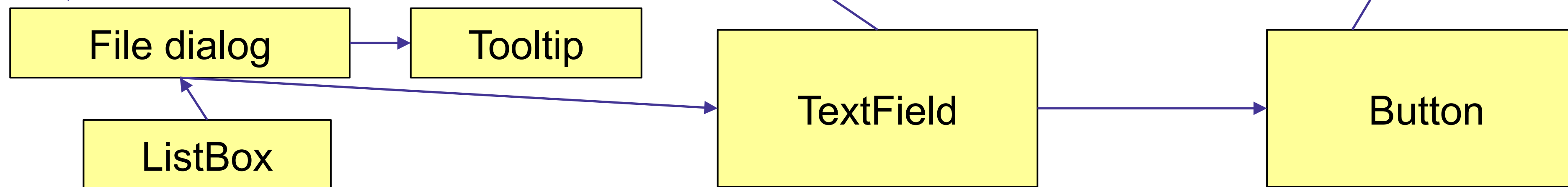
This PC
Desktop
Documents
Downloads
Inteno DG301AL
Music
Pictures
Videos
Local Disk (C:)
vanhan_levyn_va
350GB (G:)
Removable Disk (

Name	Date modified	Type	Size
esimerkki	13.1.2015 20:00	File folder	
luennot_julkinen	13.1.2015 20:00	File folder	
luentokorvike1	13.1.2015 20:00	File folder	
luentokorvike2	27.1.2015 19:45	File folder	
luentokorvike3	13.1.2015 20:00	File folder	
~$Ohar5 Vuorovaikutus.pptx	3.2.2015 18:27	Microsoft PowerP...	1 KB
~$Ohar6 YritysjärjestelmäPatternit.pptx	3.2.2015 21:41	Microsoft PowerP...	1 KB
koneenohjaus.pptx	11.2.2014 18:00	Microsoft PowerP...	30 KB
Ohar0 Aloitus.ppt		Microsoft PowerP...	465 KB
Ohar1 Johdanto.p		Microsoft PowerP...	1 855 KB
Ohar1 Johdanto.p		Microsoft PowerP...	1 855 KB
Ohar1 Johdantoa.		Microsoft PowerP...	848 KB
Ohar1_2_Johdanto.pptx	16.1.2015 14:00	Microsoft PowerP...	1 129 KB
Ohar2 Kuvaus.pptx	16.4.2014 1:15	Microsoft PowerP...	2 122 KB
Ohar2 Kuvaus_dokumentointi.pptx	29.1.2015 15:11	Microsoft PowerP...	843 KB
Ohar2 Kuvaus_uudempi.pptx	21.1.2015 12:16	Microsoft PowerP...	2 385 KB
Ohar4 Komponentit.pptx [Repaired].pptx	22.1.2014 23:54	Microsoft PowerP...	457 KB
Ohar4 Komponentit_bugaava_paskaexcel...	28.1.2015 9:59	Microsoft PowerP...	961 KB
Ohar5 Suunnittelumallit.pptx	2.2.2015 4:12	Microsoft PowerP...	2 947 KB
Ohar5 Vuorovaikutus.pptx	30.1.2014 14:41	Microsoft PowerP...	919 KB

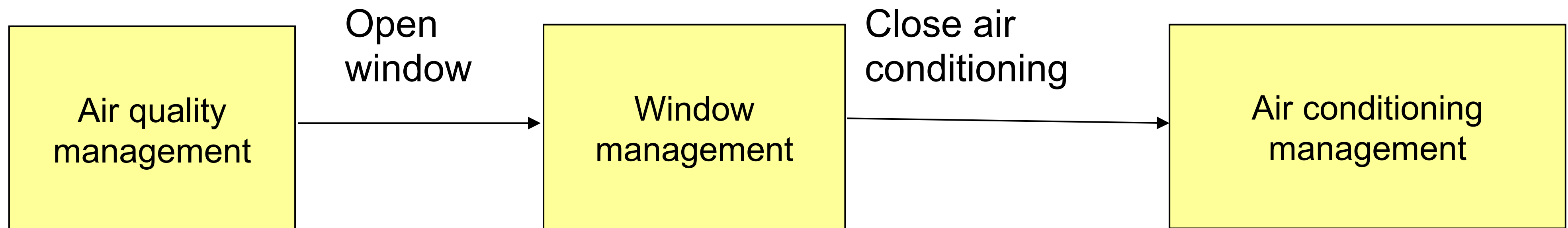
Type: Microsoft PowerPoint Presentation
Authors: samuel
Title: PowerPoint Presentation
Size: 29,8 KB
Date modified: 11.2.2014 18:00

File name: Ohar5 Vuorovaikutus.pptx
Save as type: PowerPoint Presentation (*.pptx)
Authors: Samuel Tags: Add a tag Title: No Slide Title Subject: Specify the subject

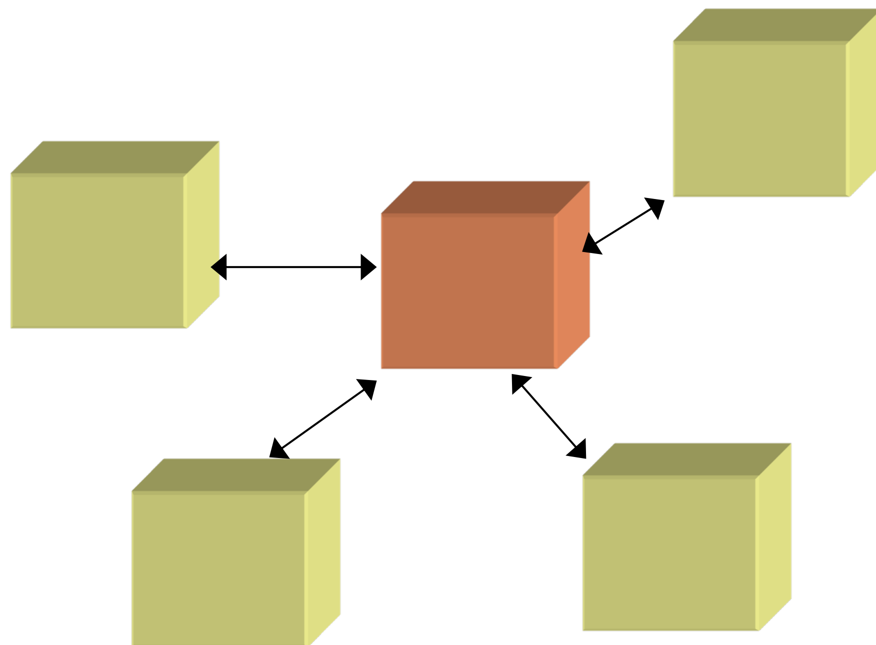
Hide Folders Tools ▾ Save Cancel



Example



Centralising dependencies: Mediator



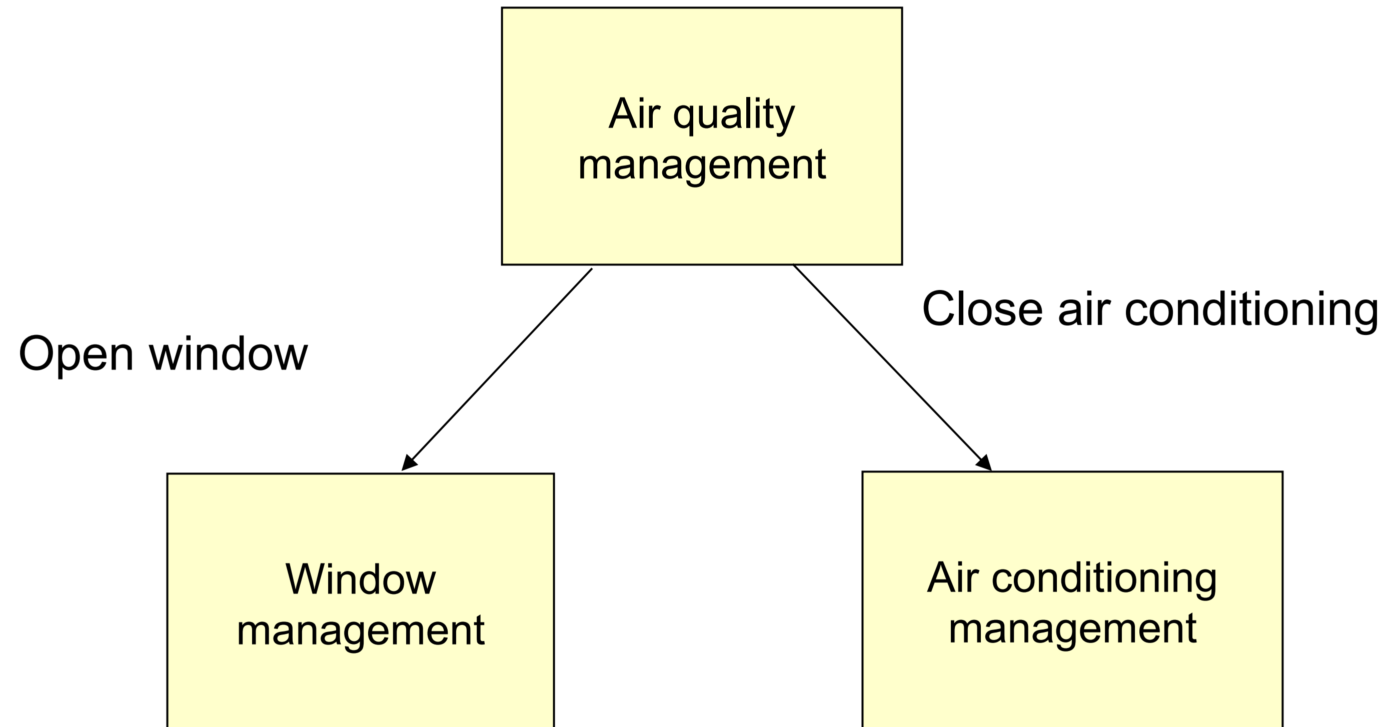
The control of interactions is centralised by limiting the responsibilities of components and taking in use a new component that is responsible for interaction management

Benefits:

- **Interaction as its own whole (broker), can be changed or tailored without changes to other components.**
- **Makes components independent of each other.**
- **Simplifies communication (one-to-many, not many-to-many)**

Problem: the centralised component may itself grow complex.

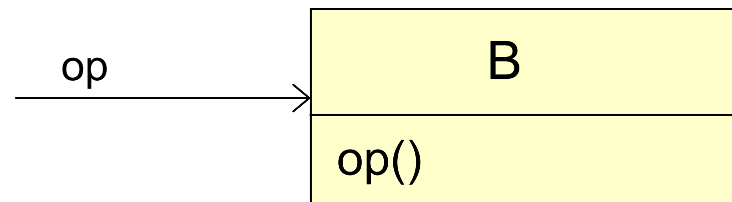
Example

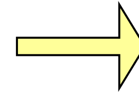


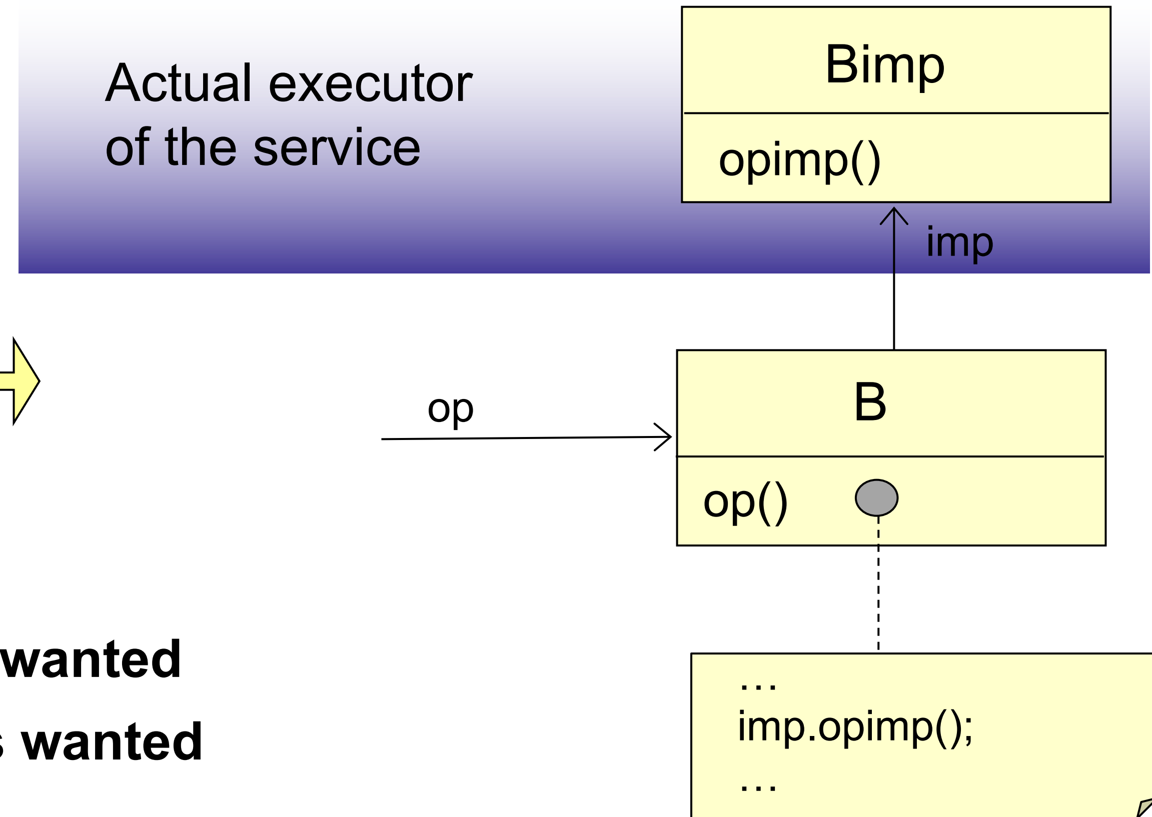
Call forwarding (delegation)

Common basic mechanism in many standard solutions

B gets a service request:



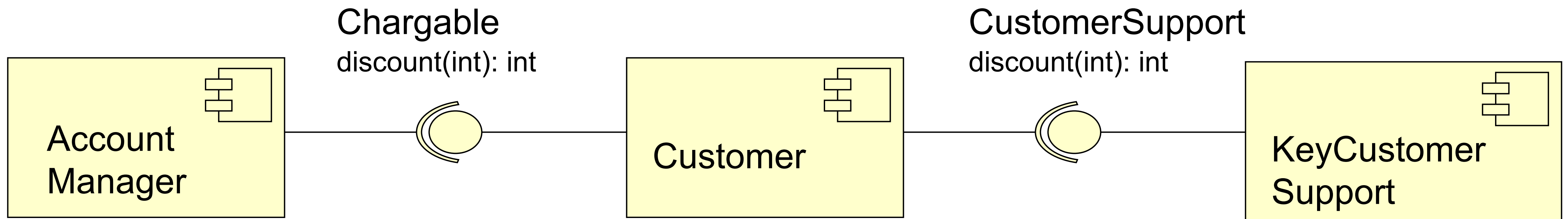




Different kinds of reasons:

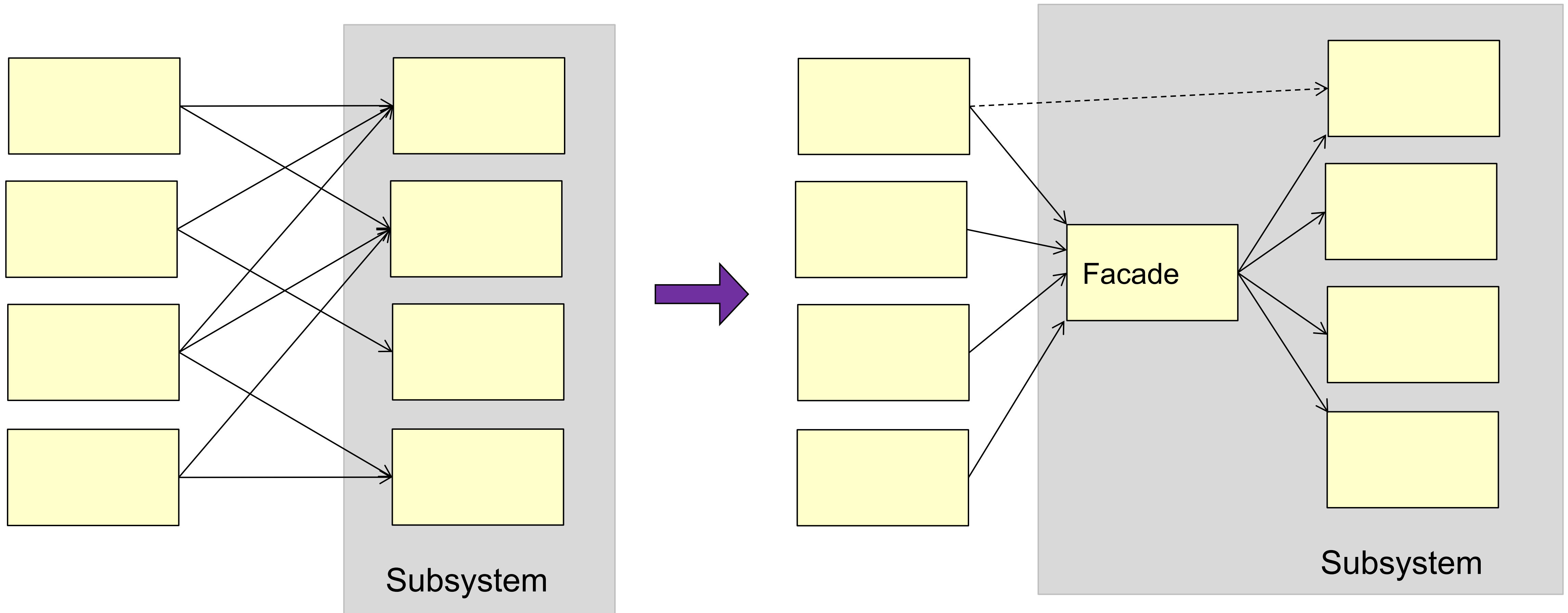
- **Side function not visible for the requester is wanted**
- **Easy dynamical change of implementation is wanted**
- **Call mode change is wanted**
- **Hiding the actual executor is wanted**

Call forwarding: example



Customer moves the request directly to KeyCustomer; AccountManager is not aware of this

Choking dependencies: Facade



Facade (traditional, recorded)

Lecture

- Check the phone (silent mode)
- Open lecture room system, select lighting, source of display, etc.
- Log in lecture room computer
- Open Powerpoint, start display mode
- Remember to use the microphone (otherwise voice is not recorded)
- Wait for recording to start, then start lecturing
- Hold lecturing
- Mute the microphone
- Hold recording (if possible)
- Unmute the microphone
- Continue recording (if held)
- Continue lecturing
- End lecturing
- Mute the microphone, put it in charger
- Stop recording (closes automatically, so don't worry; publishing is done automatically)
- Log out the computer
- Turn off lights, videos, etc.
- Phone to normal mode

With facade:

- **Start the lecture**
- **Hold the lecture**
- **Continue the lecture**
- **End lecture**

Facade (online lecture)

Lecture

- Check the phone (silent mode)
- Log in the lecturing computer
- Sign in Zoom home page to get the host privilege
- Open Zoom, mute microphone and disable camera; open chat
- Open Powerpoint, start display mode
- Wait until starting time
- Unmute, enable camera, start recording
- Share the Powerpoint
- Hold lecturing
- Mute the microphone, disable camera
- Hold recording
- Unmute the microphone, enable camera
- Continue recording
- Continue lecturing
- End lecturing, stop sharing Powerpoint
- Mute the microphone, disable camera and stop recording
- End the session for all; recording is converted and saved on local computer
- Phone to normal mode
- Open Panopto. Upload the recording (takes some time).
- When recording is uploaded, close the connection. Panopto processes the video (2-6 hours) and publish it.

With facade:

- **Start the lecture**
- **Hold the lecture**
- **Continue the lecture**
- **End lecture**

Façade pattern

Is used to give a subsystem a simple default interface, suitable for most of the users, to perform higher level services that can connect several functionalities.

Façade does not completely hide the components of the subsystem from direct use.

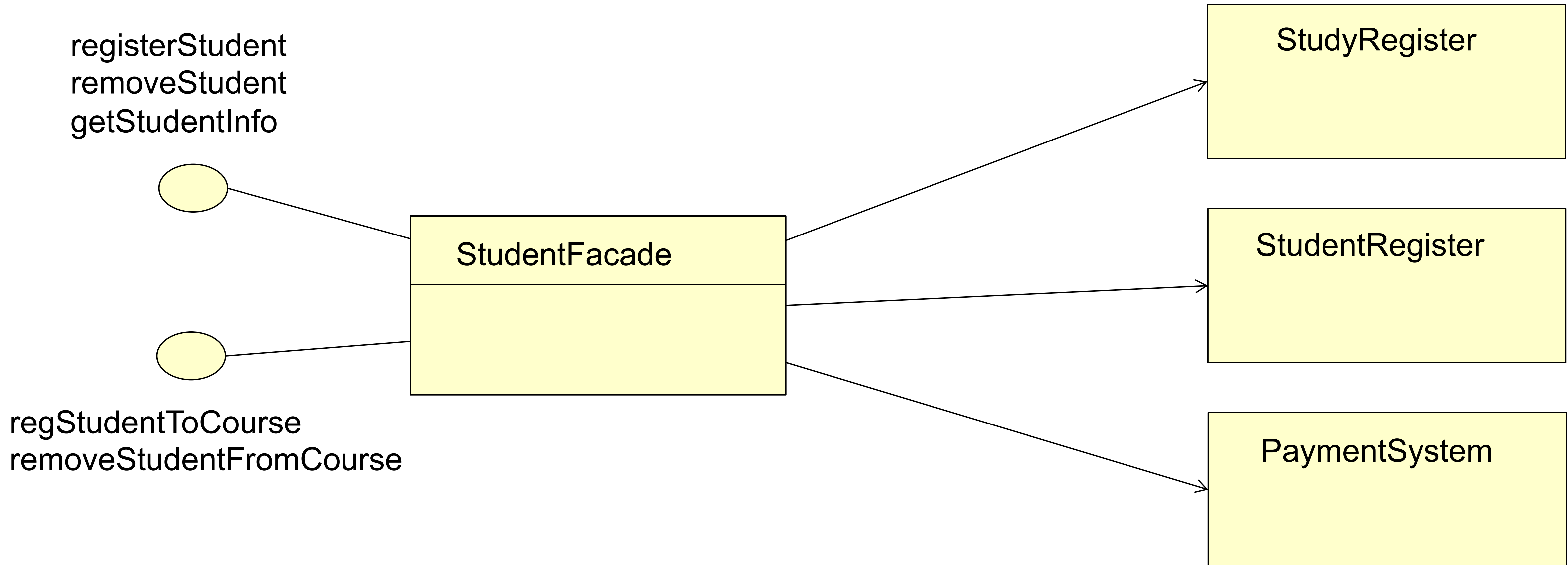
Can be used in layered architecture to give each layer simple connection point and interface.

Compare to broker: façade is a one-way service, typically façade only delivers the calls to the correct component(s).

The use of façade can be further restricted with roles (different façade for different users).

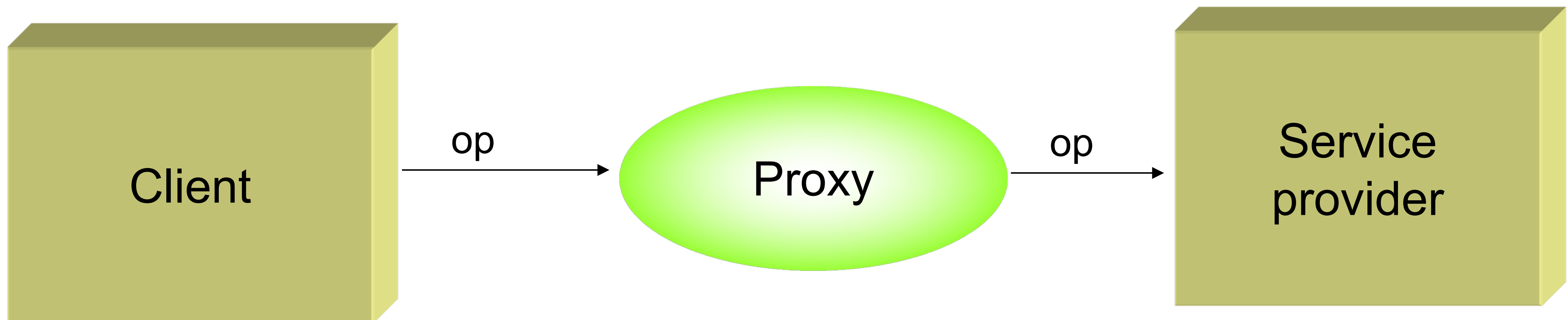
Tampereen yliopisto
Tampere University

Example



Removing component dependencies using proxy

Proxy: a component that represents another component in some context so that clients of the component do not know about this. Typically the proxy makes some side functions when fulfilling the service request.



Applications

When direct connection to the resource is not wanted

Distributed systems

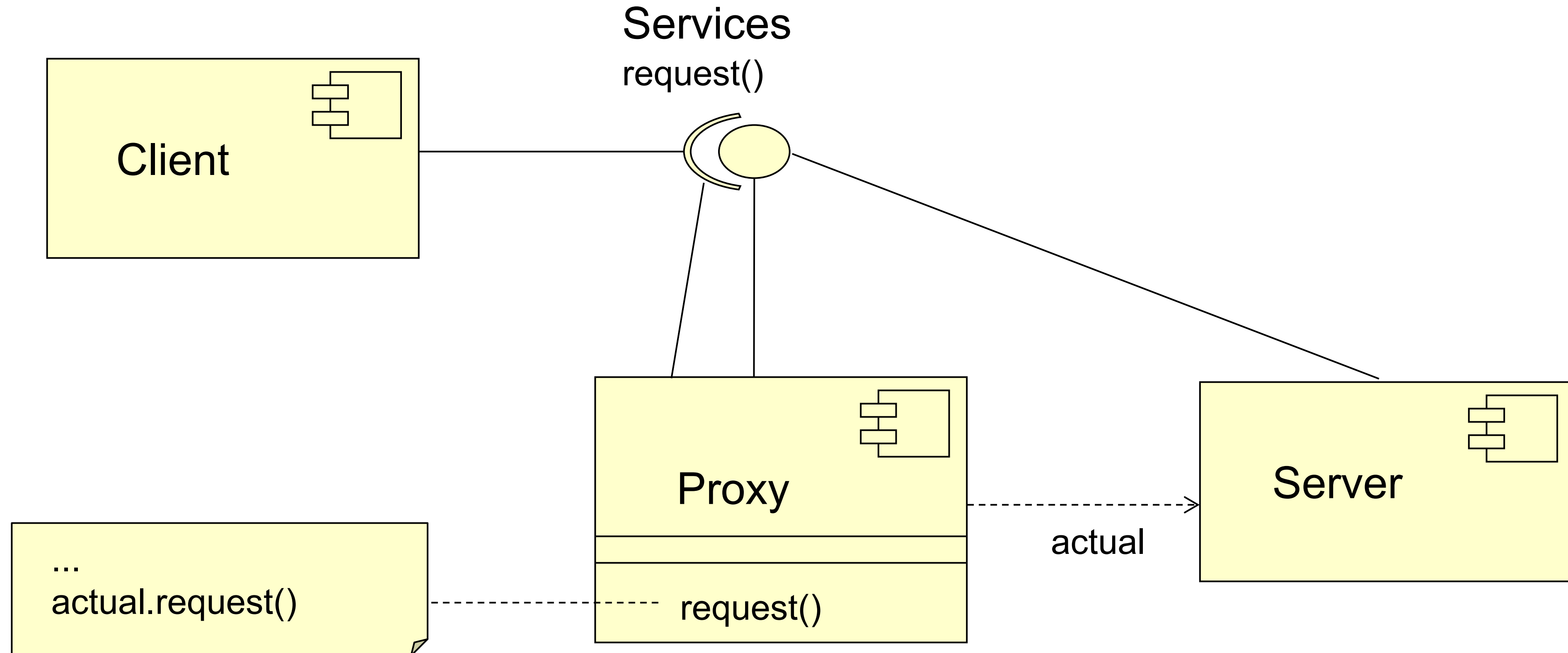
Delayed loading (e.g. object bases), partitioned loading.

Intelligent pointers

Security, authentication of clients

Smart proxy: smart selection of the service provider.

Proxy pattern



Different kinds of proxies, usages

Virtual proxy: delayed loading, etc. Works in-between and ensures that heavy operations are not started in vain.

Protection proxy: the user can’t use the service or the resource directly, the proxy in-between may contain e.g. authentication.

Remote proxy: usage of network resources through an interface, the proxy takes care of connections (e.g. directs calls to correct places, gives additional security etc.)

Proxy vs. Mediator vs. Broker

Proxy hides the service from its users

Mediator acts as a centralised connection manager.

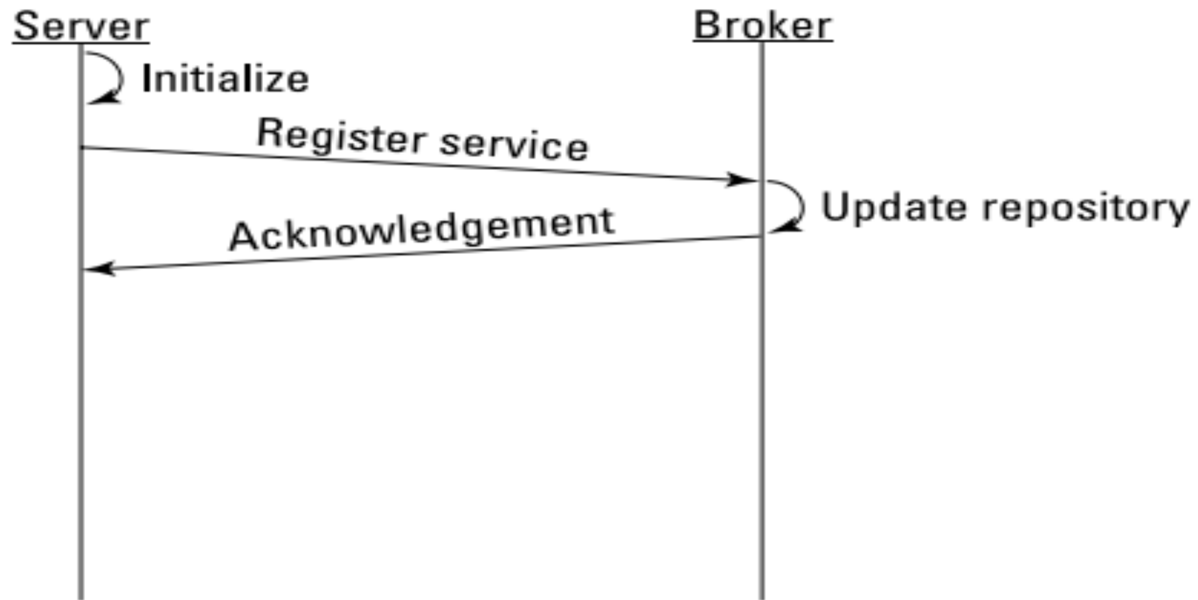
Broker, as above, but conceptual difference. Resources are distributed, can alter the location, etc.

- **Broker knows where to find the services**
- **Can execute the services and return the results to the client or work as a library: give a service**

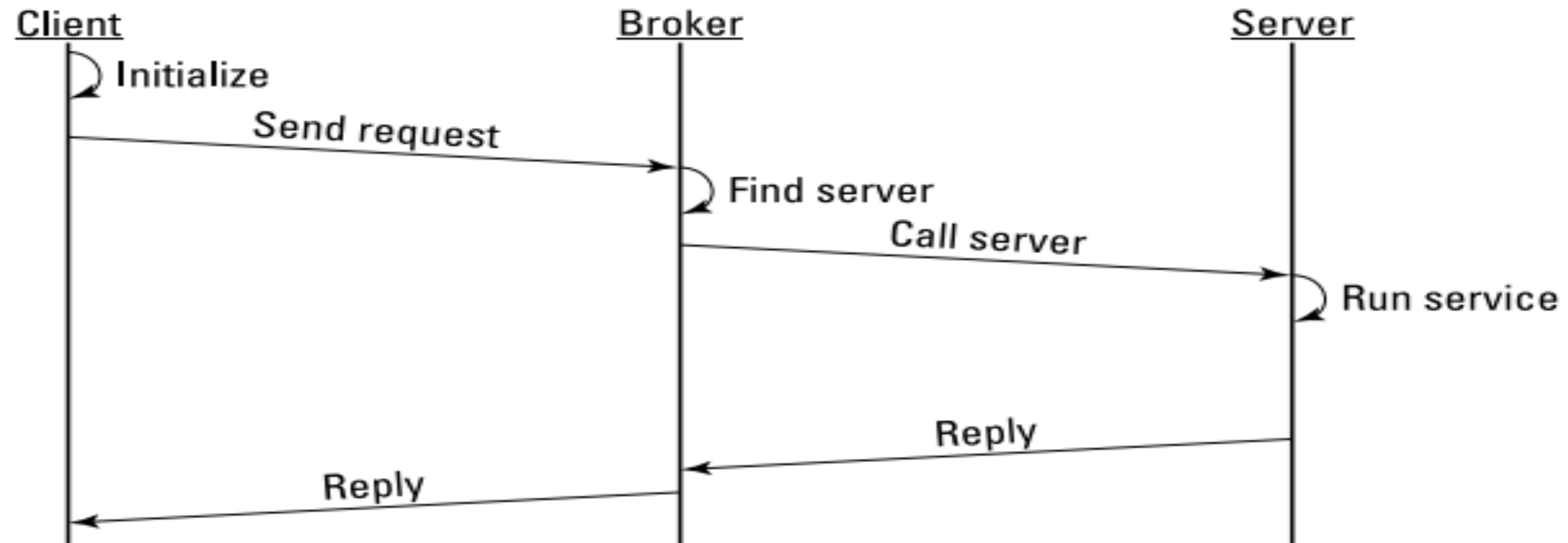
Brokers, proxies, adapters etc. cooperating.

Several brokers; may know each other

Broker (service registering)



Broker (deliver request)



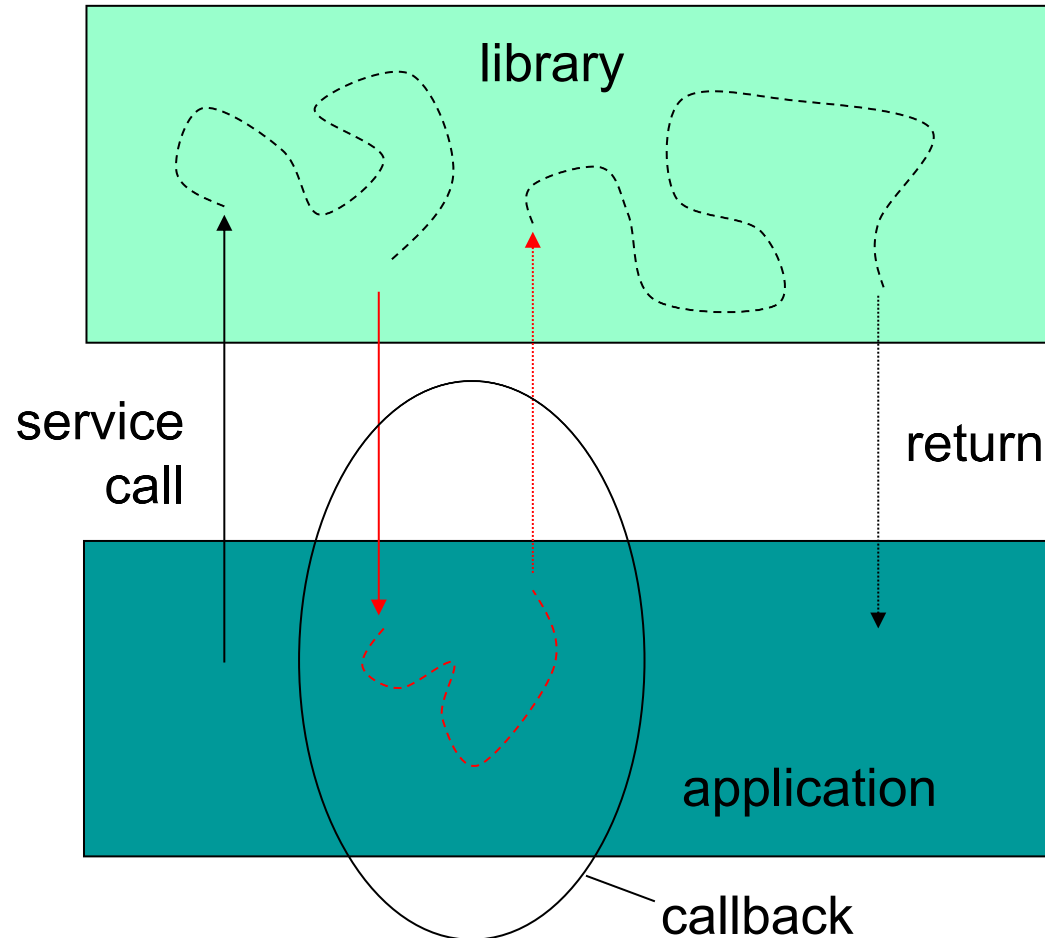
Removing dependencies using callbacks

Callback: A call by the service provider to the service requester during the service execution. It is a technique to allow the caller to get control in the middle of service.

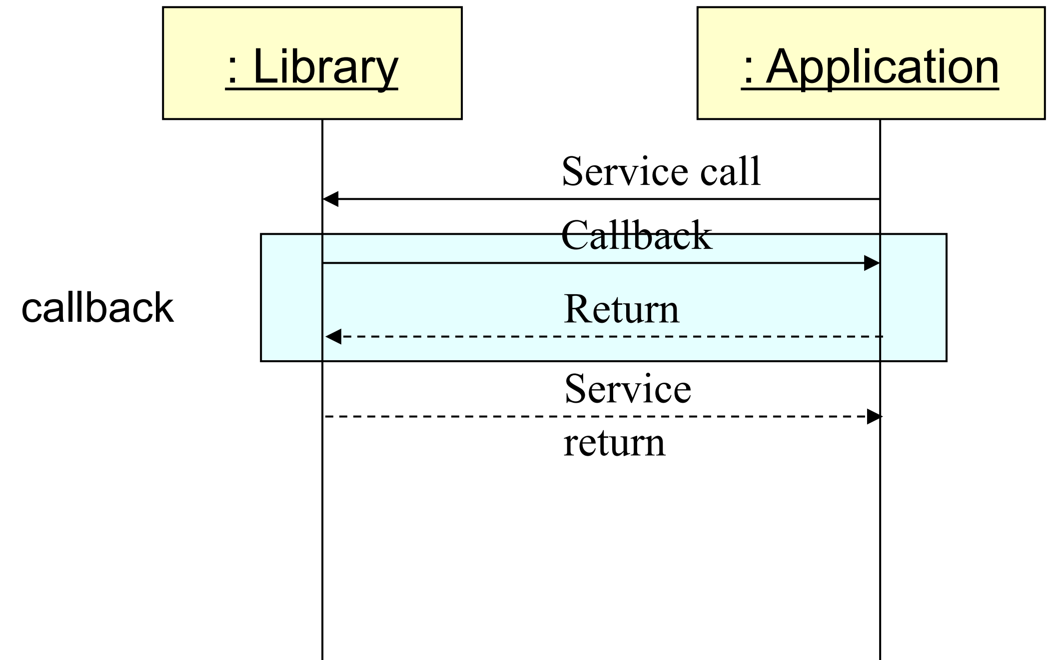
Typically the service is part of a general library that shall not become dependent on applications using the library.

Callback makes application-specific tailoring for a general service possible.

Callback

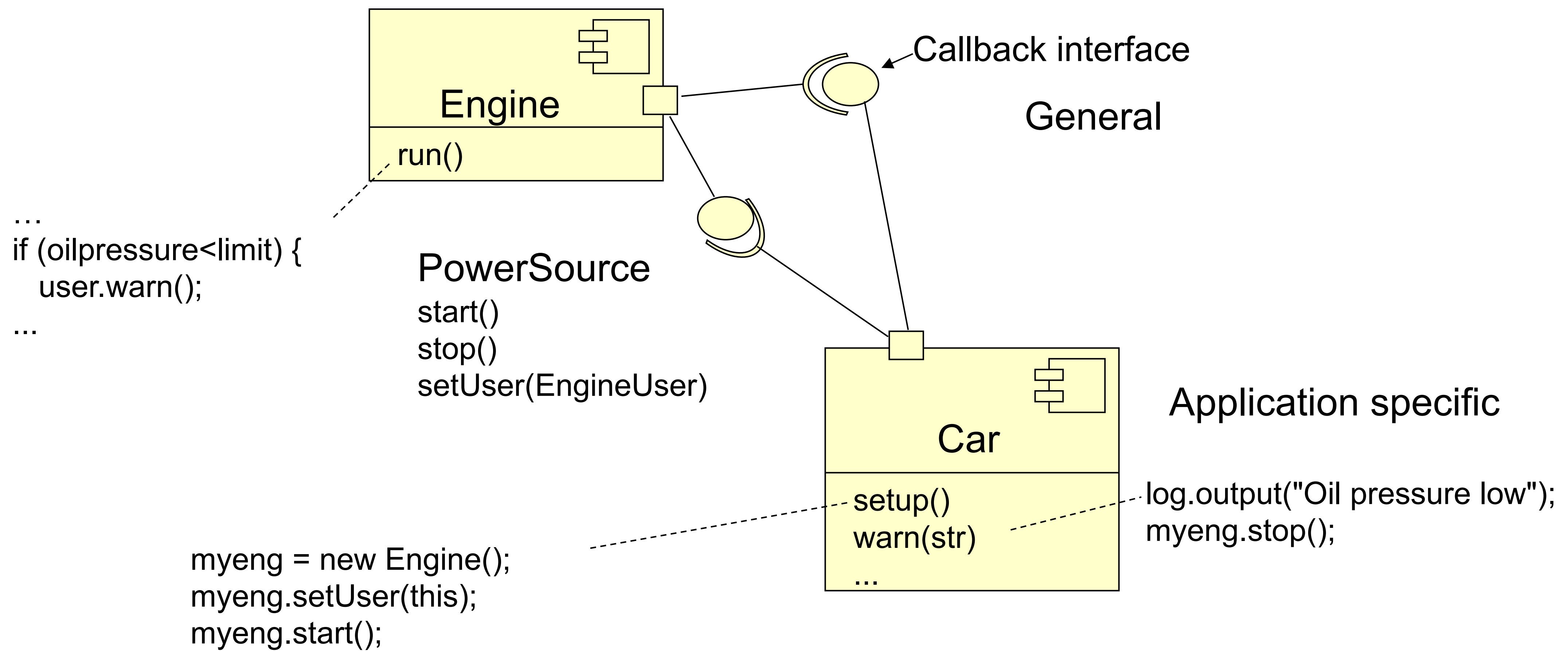


Sequence chart:



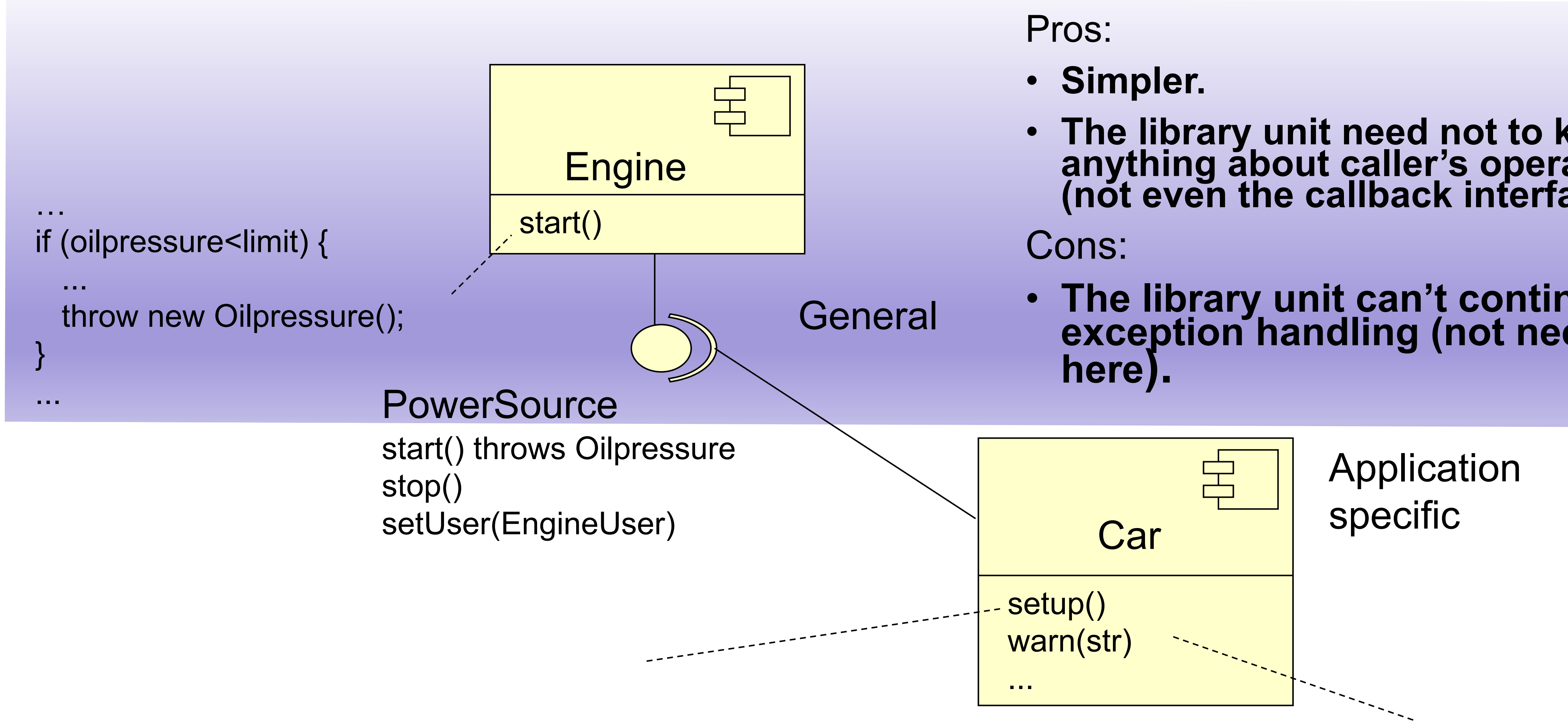
Callback interface

Note: in real car system engine control etc. are separate systems than physical devices (messaging through a bus).



Tampereen yliopisto
Tampere University

With exceptions



Differences:

- **No callback interface**
- **Calling unit (Car) has to expect exceptions.**

Pros:

- **Simpler.**
- **The library unit need not to know anything about caller's operations (not even the callback interface).**

Cons:

- **The library unit can't continue after exception handling (not needed here).**

Decreasing dependencies using events

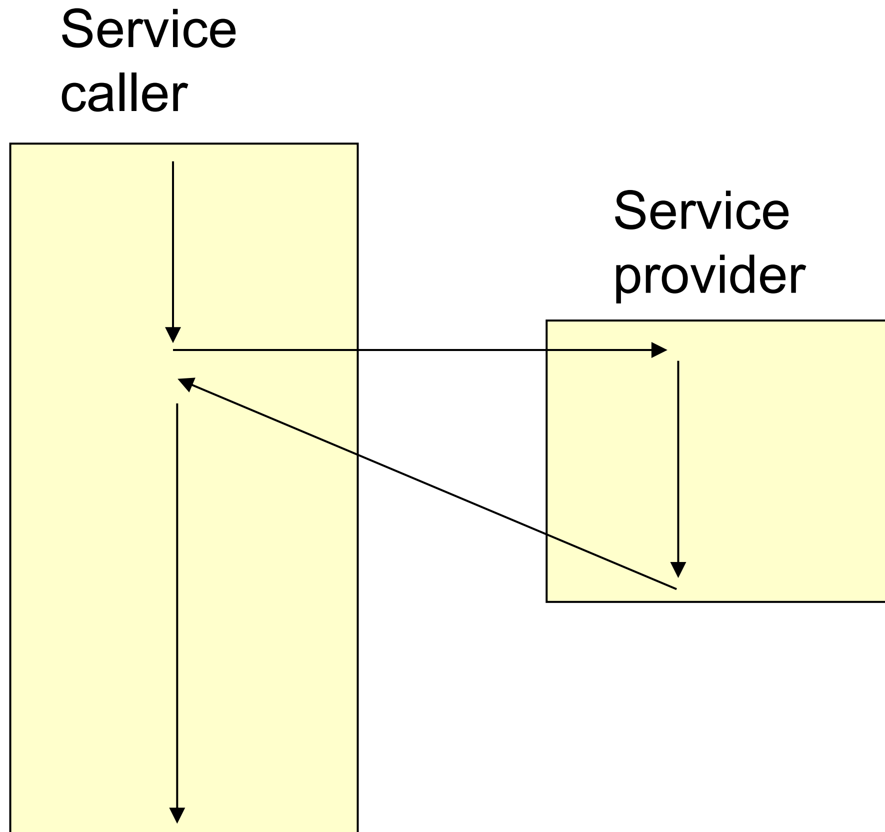
An event is a run-time data object of the program (in this context).

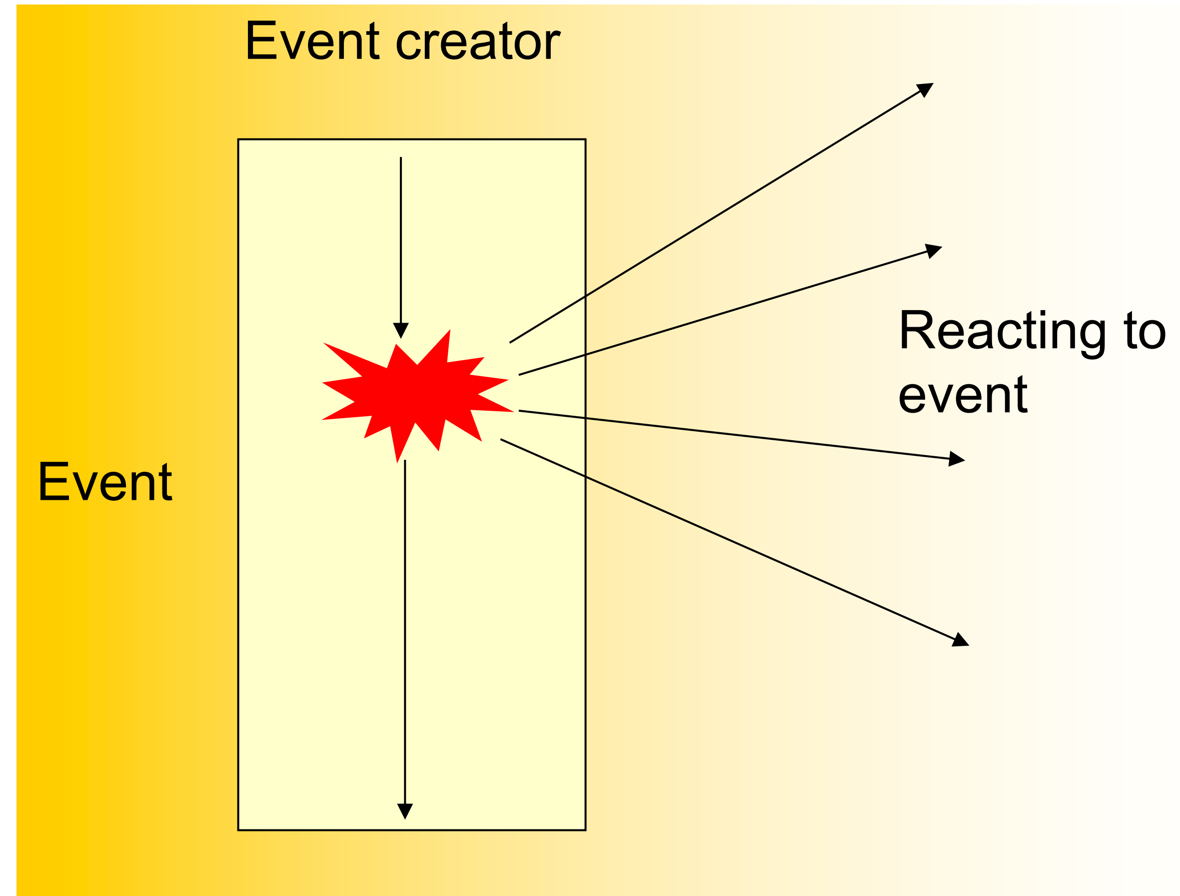
- **It is created by a component,**
- **One or more components react to its creation.**
- **It vanishes when there are no components that should react to its creation.**

The creator of an event does not know units reacting to the creation.

Events remove visible dependency, but if used incorrectly they don’t alter the functional dependencies (understanding and modifying the behaviour can be even harder).

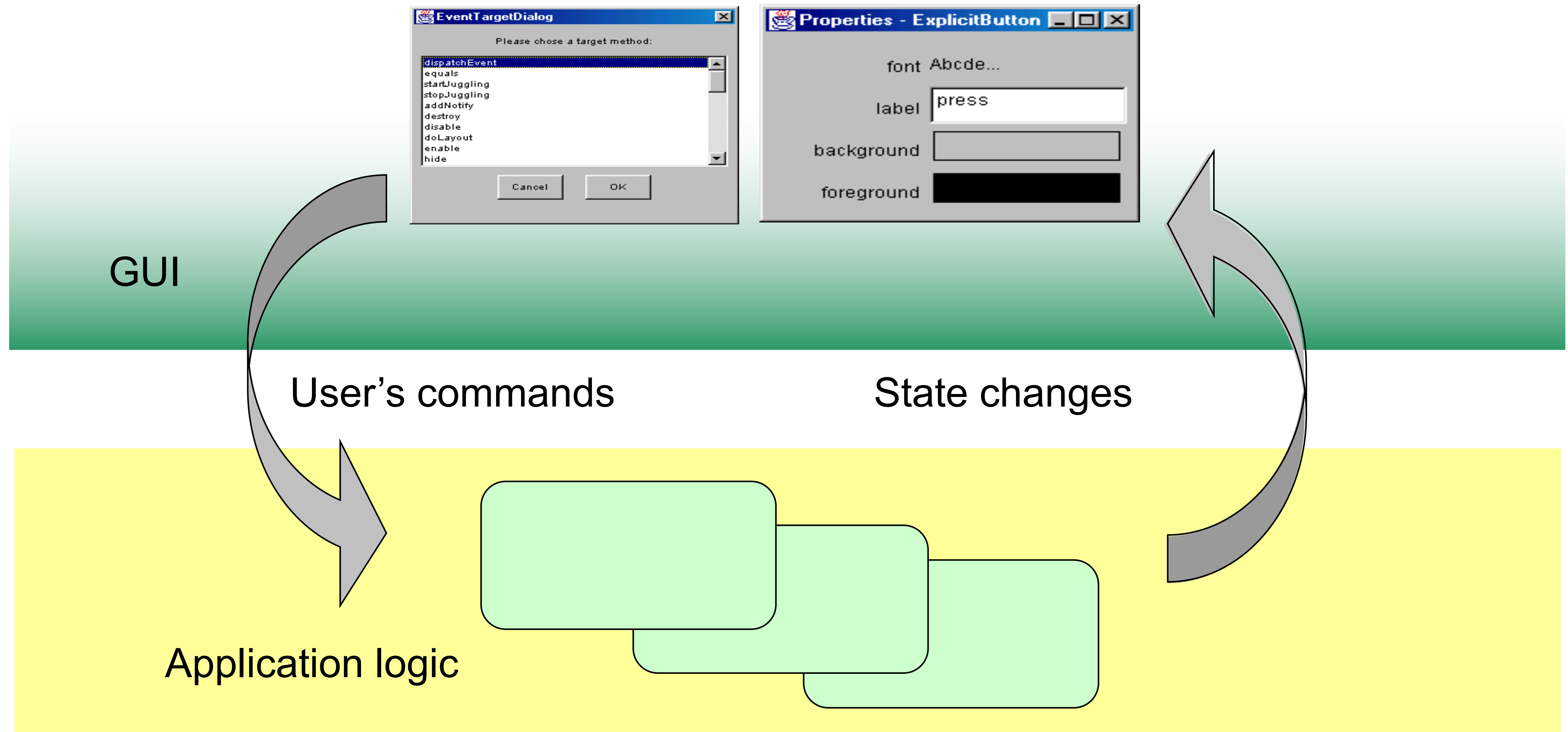
Traditional call vs. Event



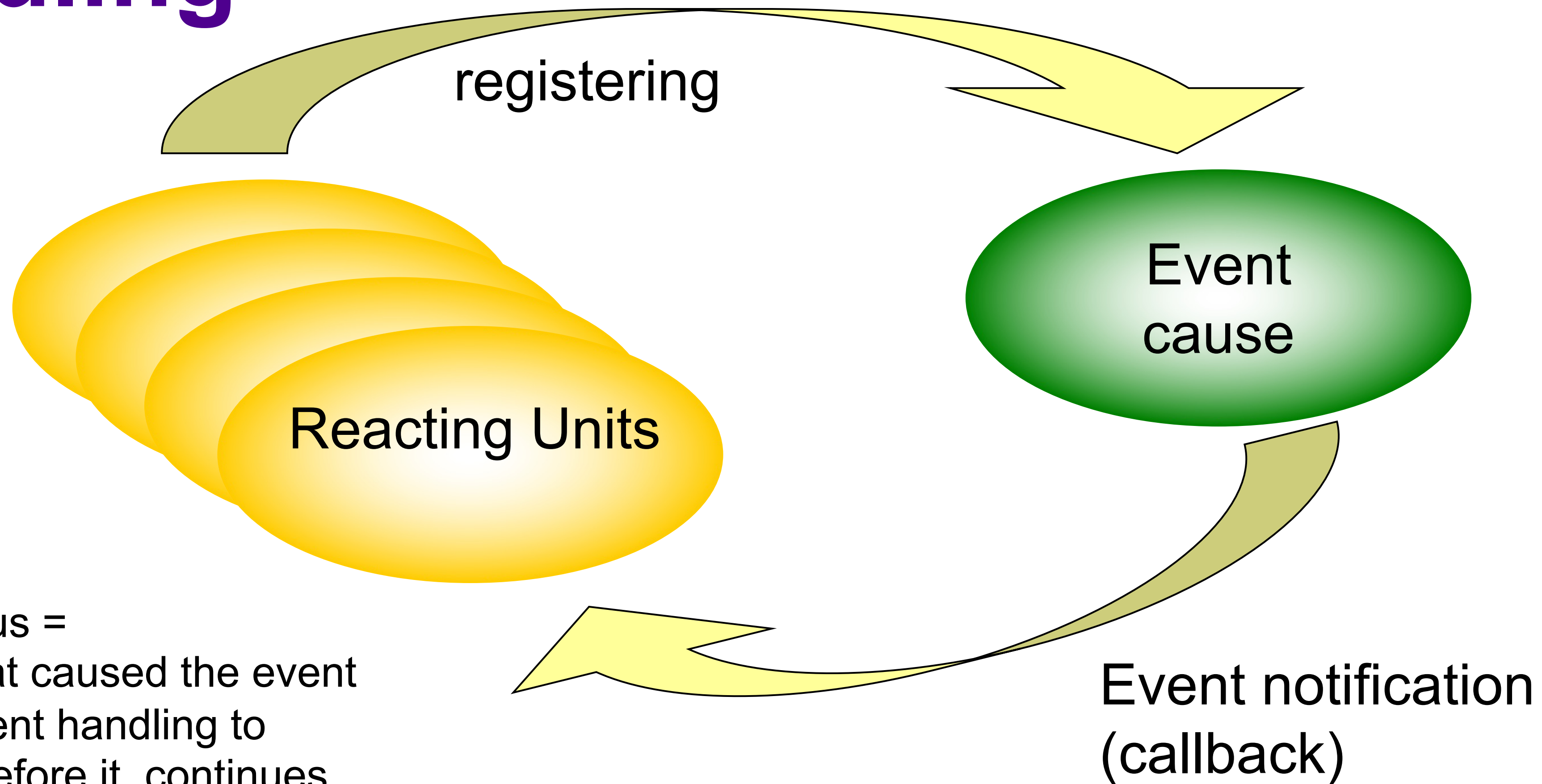


Tampereen yliopisto
Tampere University

Events in user interfaces

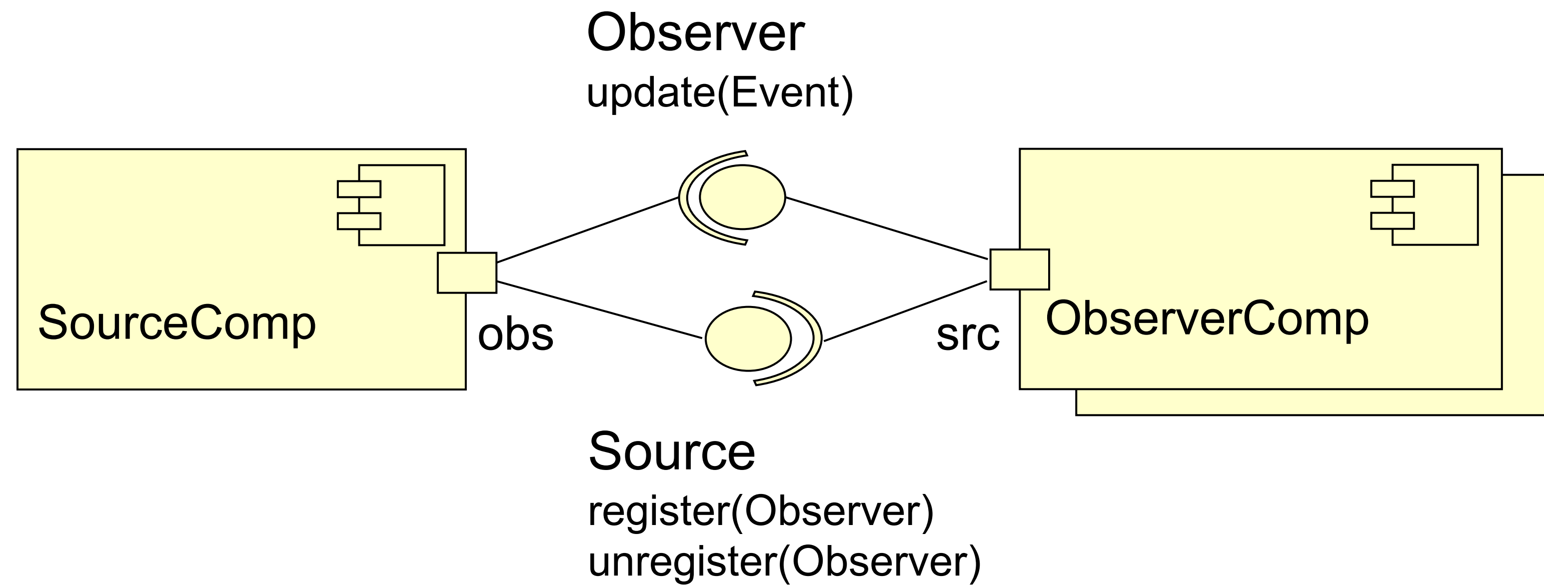


Synchronic callback-based event handling

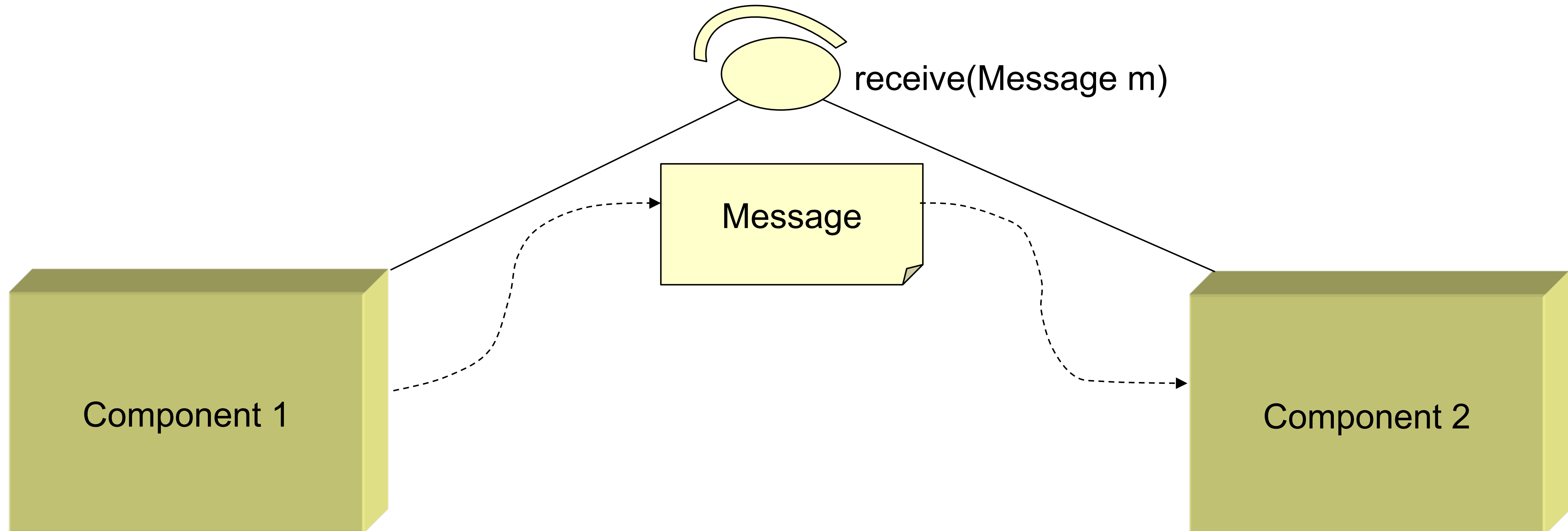


Synchronous =
The unit that caused the event will wait event handling to complete before it continues.

Observer pattern



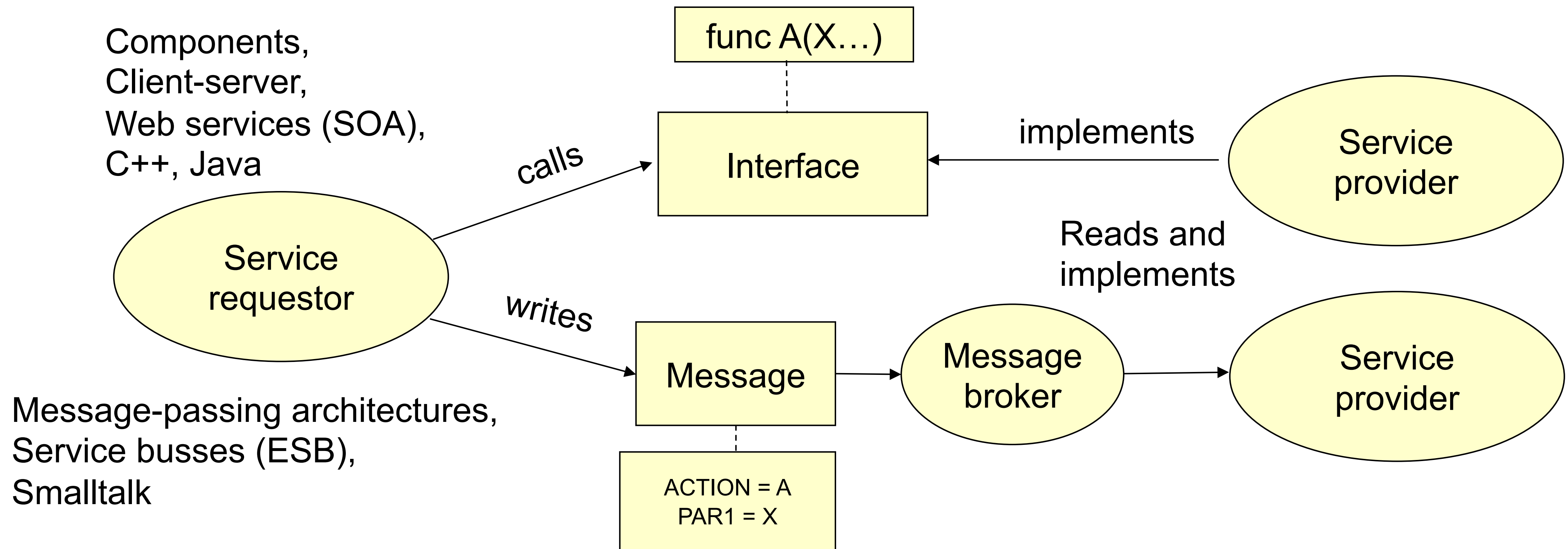
Message-based communication



Components implement generic receive interface (and use it either directly or through a broker).
Components communicate with each other sending messages
Message dispatcher takes care of message delivery.

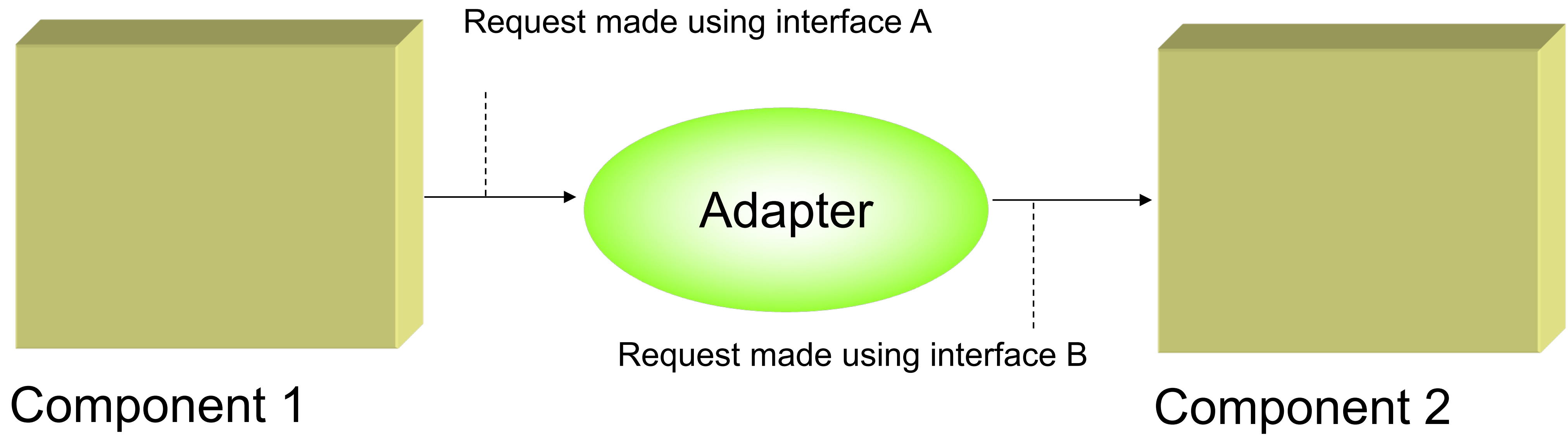
Tampereen yliopisto
Tampere University

Interfaces vs. Messages



The interface tells what is done and by which information, message can tell what ever (what is done, who does, which information).

Removing interface dependencies using adapters

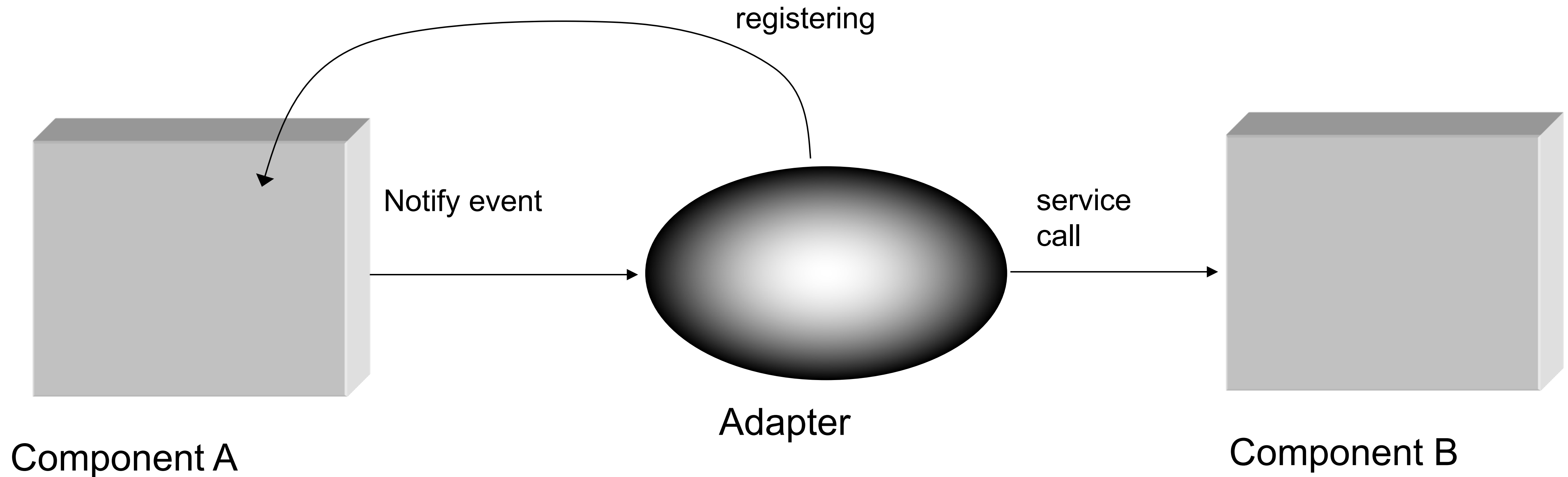


Adapter: program unit between the requester and provider of a service. Makes requester independent of the provider's interface.

Adapter and events

Using adapters in event-based communication between components

Adapter fulfils observer interface



Conclusions

Role interfaces give architecture that is more precise.

Using a broker concentrates on centralised interaction.

A Façade concentrates usage of subsystems.

Side functionality can be added by proxies.

Control is temporarily transferred back to the caller by callbacks.

Observer is a common solution for event-based interaction.

Message-based communication loosens ties.

Interfaces can be modified by adapters.

Links and reading

A set of Java patterns: http://www.java-forums.org/ocmjea/57996-tutorial-review-java-design-patterns-java-architect-exam.html

About broker: Pattern-oriented Software architecture for dummies (chapter 12).

Mediator and message passing example Java Message Service

http://docs.oracle.com/javaee/6/tutorial/doc/bncdr.html

Mediator, simple example code: http://www.journaldev.com/1730/mediator-design-pattern-in-java-example-tutorial

Proxy pattern: http://www.oodesign.com/proxy-pattern.html