ARTIST2 Summer School 2008 in Europe

Autrans (near Grenoble), France

September 8-12, 2008

Enforceable Component-Based Real-Time Contracts

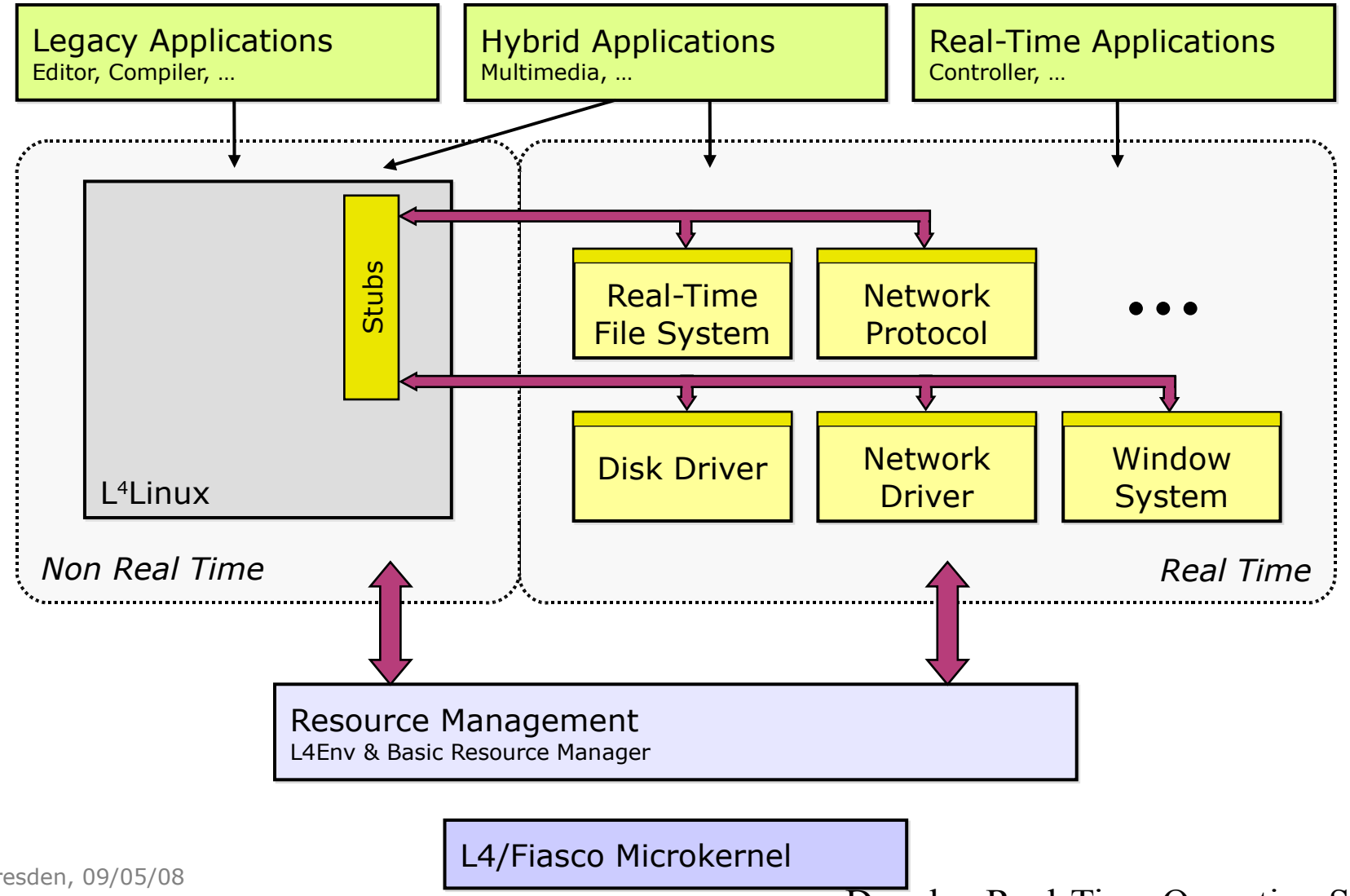
Lecturer: Hermann Härtig

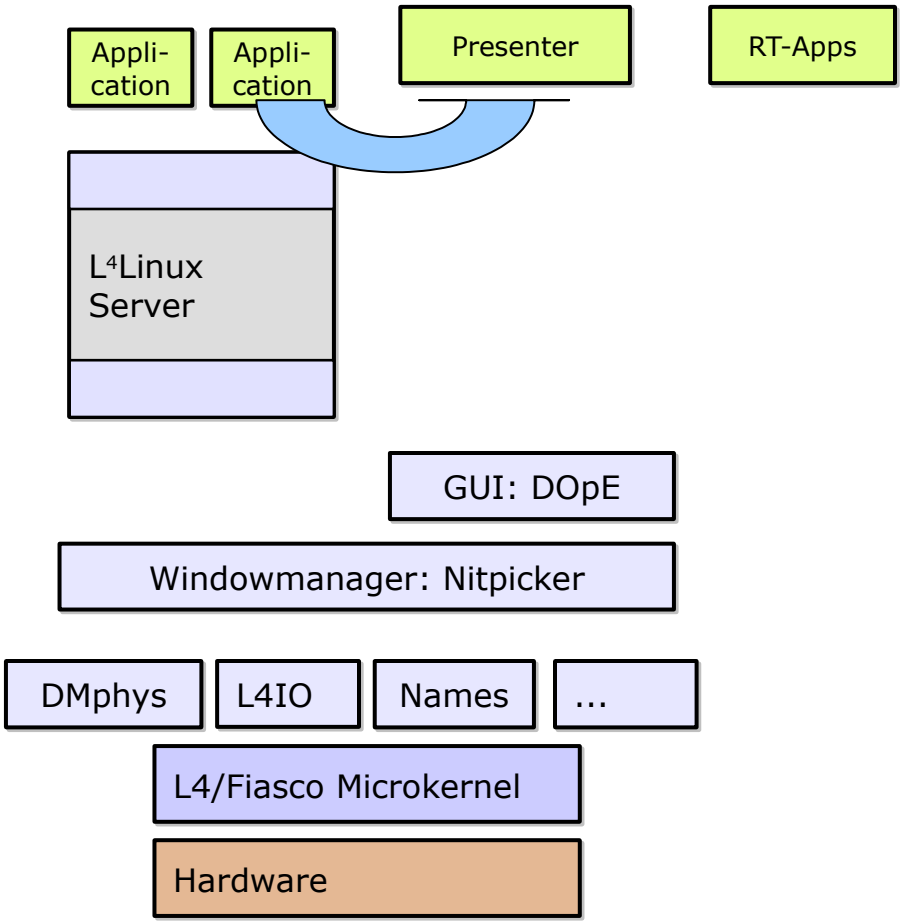
TU Dresden

TECHNISCHE UNIVERSITÄT DRESDEN

Enforceable Component-Based Real-Time Contracts Supporting Real-Time Properties from SW Development to Execution

Hermann Härtig, Steffen Zschaler
Department of Computer Science


Legacy Applications
Editor, Compiler, …
Hybrid Applications
Multimedia, …
Real-Time Applications
Controller, …
Stubs
Real-Time File System
Network Protocol
• • •
Disk Driver
Network Driver
Window System
L4Linux
Non Real Time
Real Time
Resource Management
L4Env & Basic Resource Manager
L4/Fiasco Microkernel


Appli-
cation
Appli-
cation
Presenter
RT-Apps
L4Linux
Server
GUI: DOpE
Windowmanager: Nitpicker
DMphys
L4IO
Names
...
L4/Fiasco Microkernel
Hardware

Outline

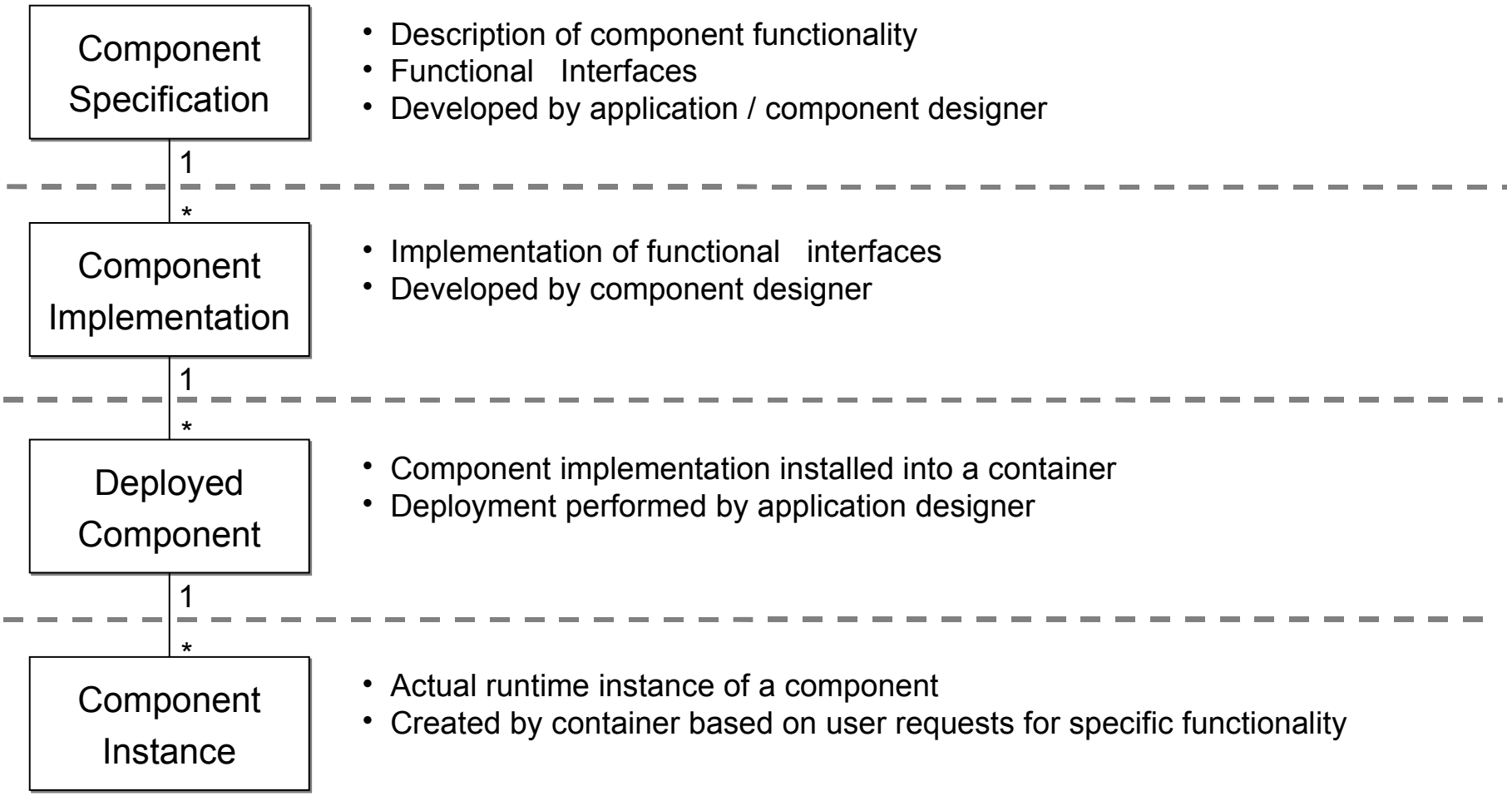
- "Components" in software engineering
- Objectives and limitations
- Short intro into DROPS RT system
- Real-Time Specifications
- Real-Time "Container"
- Lessons learned

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

Clemens Szyperski (with Dominik Gruntz and Stephan Murer):
Component Software – Beyond Object-Oriented Programming – 2nd Edition
Addison-Wesley / ACM Press, 2002. ISBN 0-201-74572-0.

- **Packaged for reuse**
  - Specifications including context dependencies
  - Component repositories / Component markets
  - Standardized run-time infrastructure (“Containers”)
  - “Transparency” goals (location, communication, language, ...)
  - Large chunks of application-level software (business SW, GUI)
- **Examples**
  - Corba
  - Enterprise Java Beans (JBoss)

Component Life Cycle and Development Steps

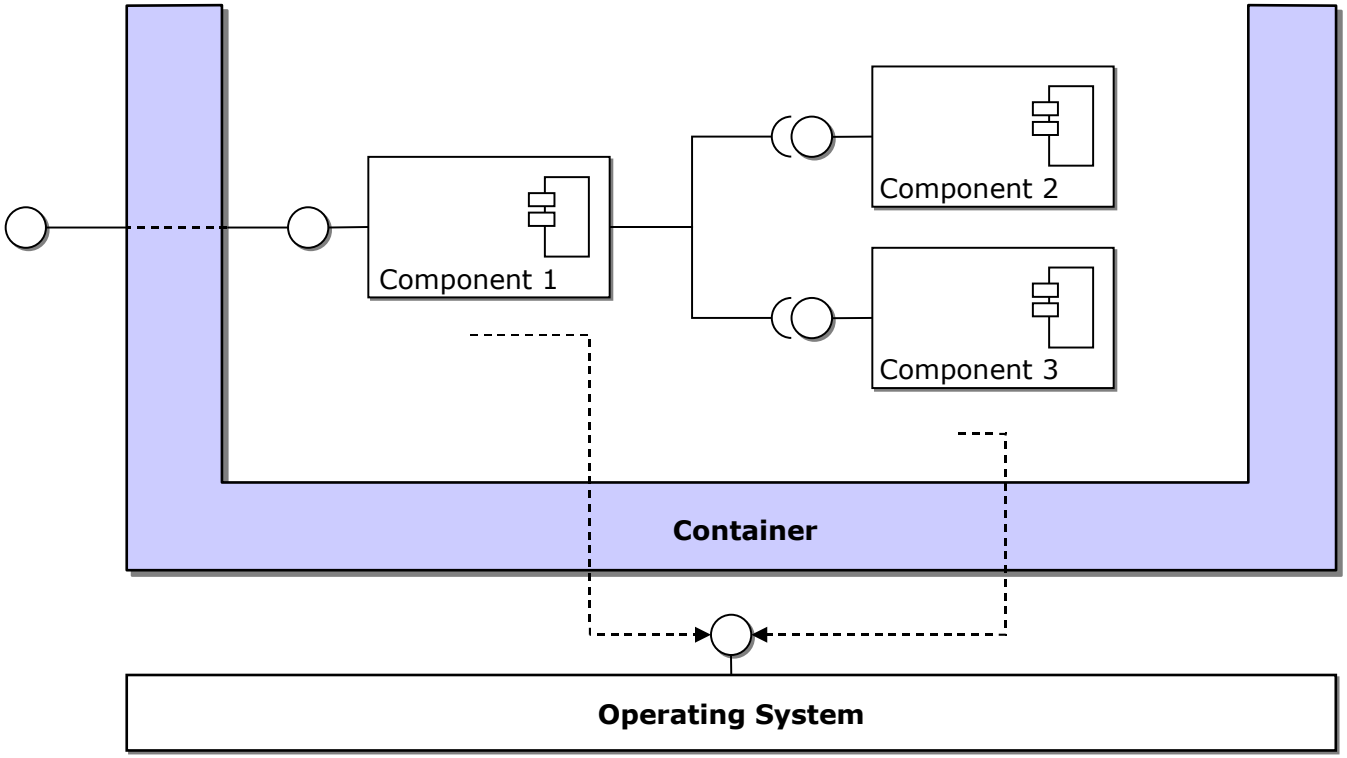


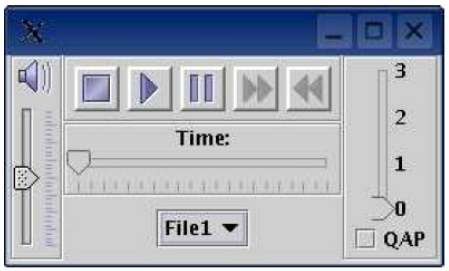
- Description of component functionality
- Functional Interfaces
- Developed by application / component designer

- Implementation of functional interfaces
- Developed by component designer

- Component implementation installed into a container
- Deployment performed by application designer

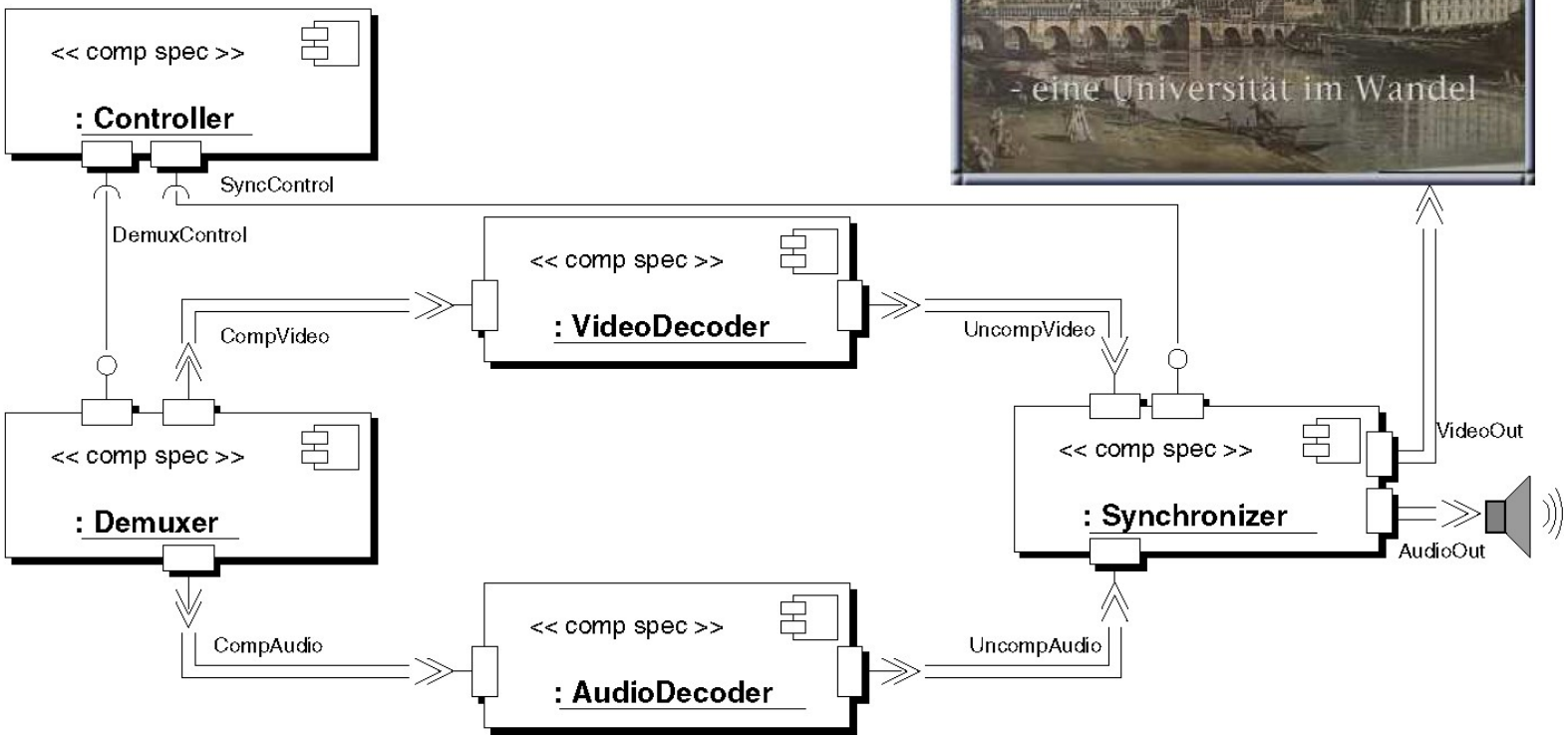
- Actual runtime instance of a component
- Created by container based on user requests for specific functionality


Component 1
Component 2
Component 3
Container
Operating System

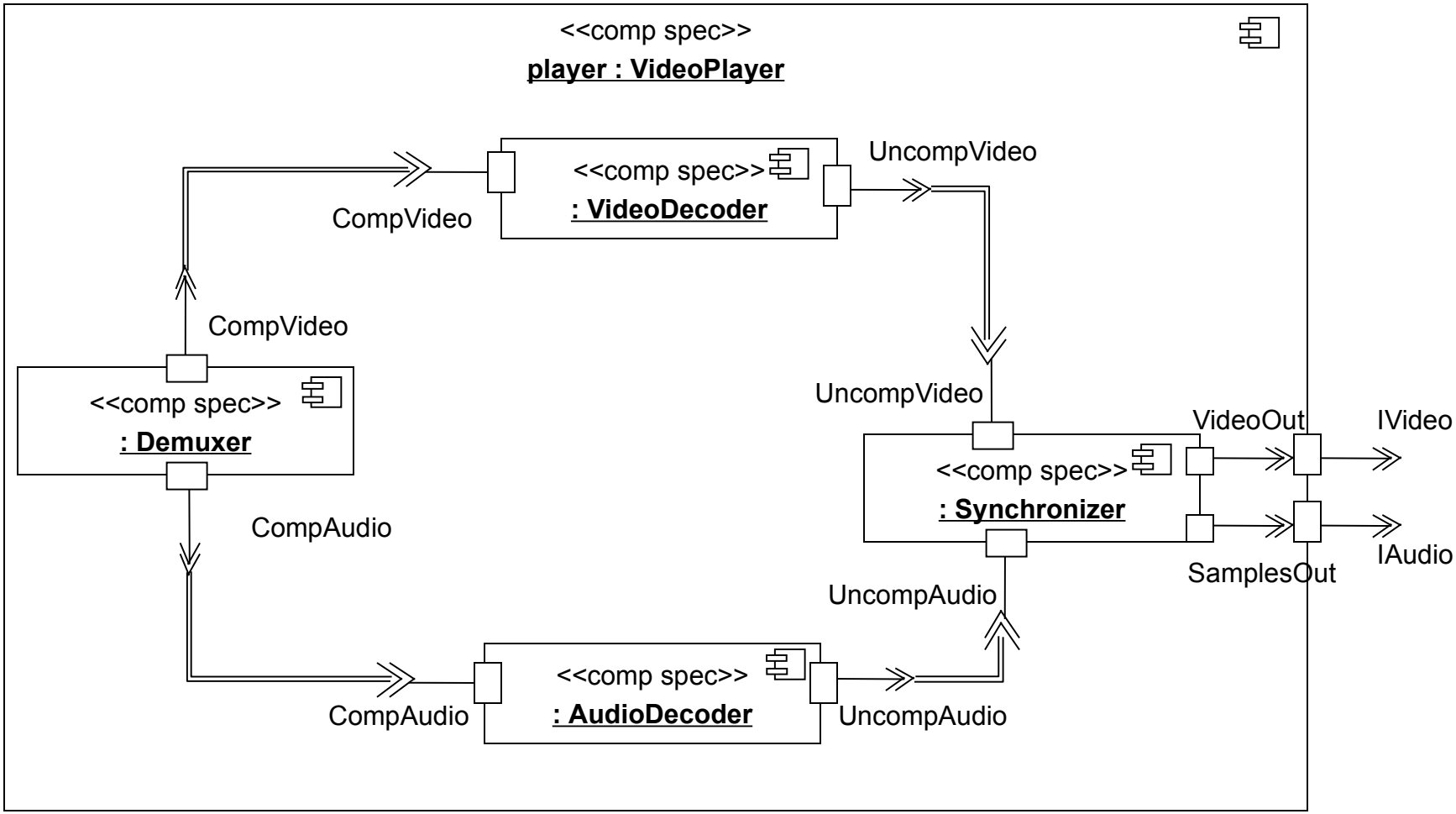

Time:
File1
QAP
3
2
1
0

:GUI


WISSEN
SCHAFFT
BRÜCKEN
- eine Universität im Wandel


<< comp spec >>
: Controller
SyncControl
DemuxControl
<< comp spec >>
: VideoDecoder
CompVideo
UncompVideo
<< comp spec >>
: Demuxer
<< comp spec >>
: Synchronizer
VideoOut
AudioOut
CompAudio
<< comp spec >>
: AudioDecoder
UncompAudio

Application Architecture: Assembly Diagram



```
<template homeplacementref="Synchronizer">
  <instance id="vdec" homeplacementref="VideoDecoder"/>
  <instance id="adec" homeplacementref="AudioDecoder"/>
  <instance id="mux" homeplacementref="Demuxer"/>

  <connectinstance id="this">
    <connect type="stream"
      usesport="uncompressedVideoIn"
      providesport="uncompressedVideoOut"
      instanceref="vdec"/>
    <connect type="stream"
      usesport="uncompressedAudioIn"
      providesport="uncompressedAudioOut"
      instanceref="adec"/>
  </connectinstance>

  <connectinstance id="vdec">
    <connect type="stream"
      usesport="compressedVideoIn"
      providesport="uncompressedVideoOut"
      instanceref="mux"/>
  </connectinstance>
  ...
</template>
```

Instance declaration

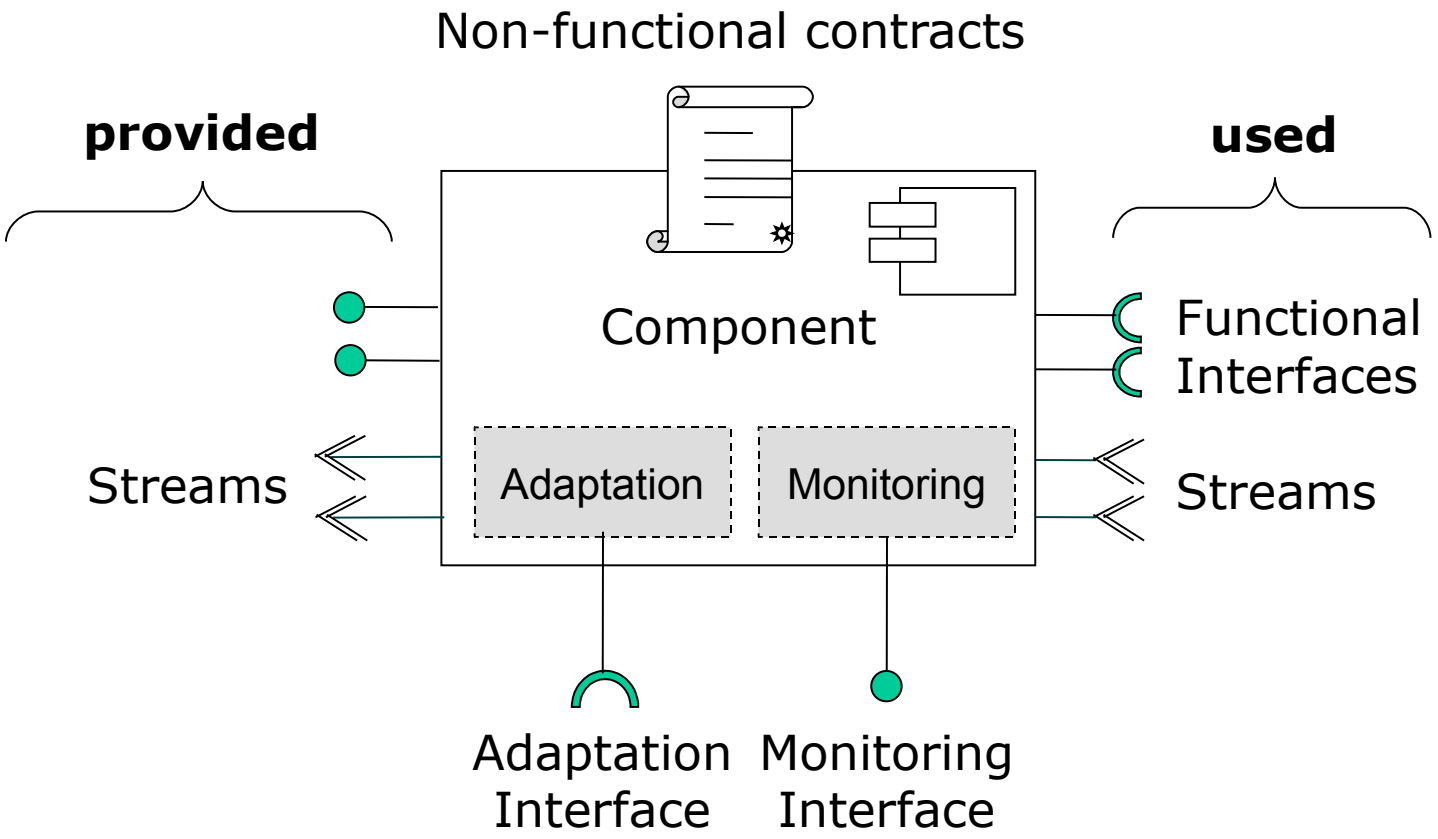
Instance connection

Objectives:

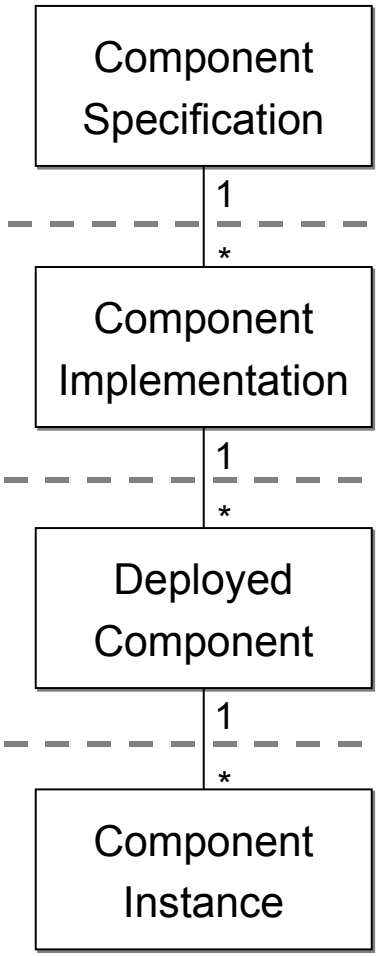
- full fledged components: specs, repositories, containers
- hard real-time + soft real-time + non-real-time
- adaptation
- application-specific description of non-functional properties

Simplifications:

- streams only (only few thoughts on other interfaces)
- no dependencies except stream concatenation
- pipelining assumed
- simple scheduling


Non-functional contracts
provided
used
Component
Functional Interfaces
Streams
Adaptation
Monitoring
Streams
Adaptation Interface
Monitoring Interface

Component Life Cycle and Development Steps

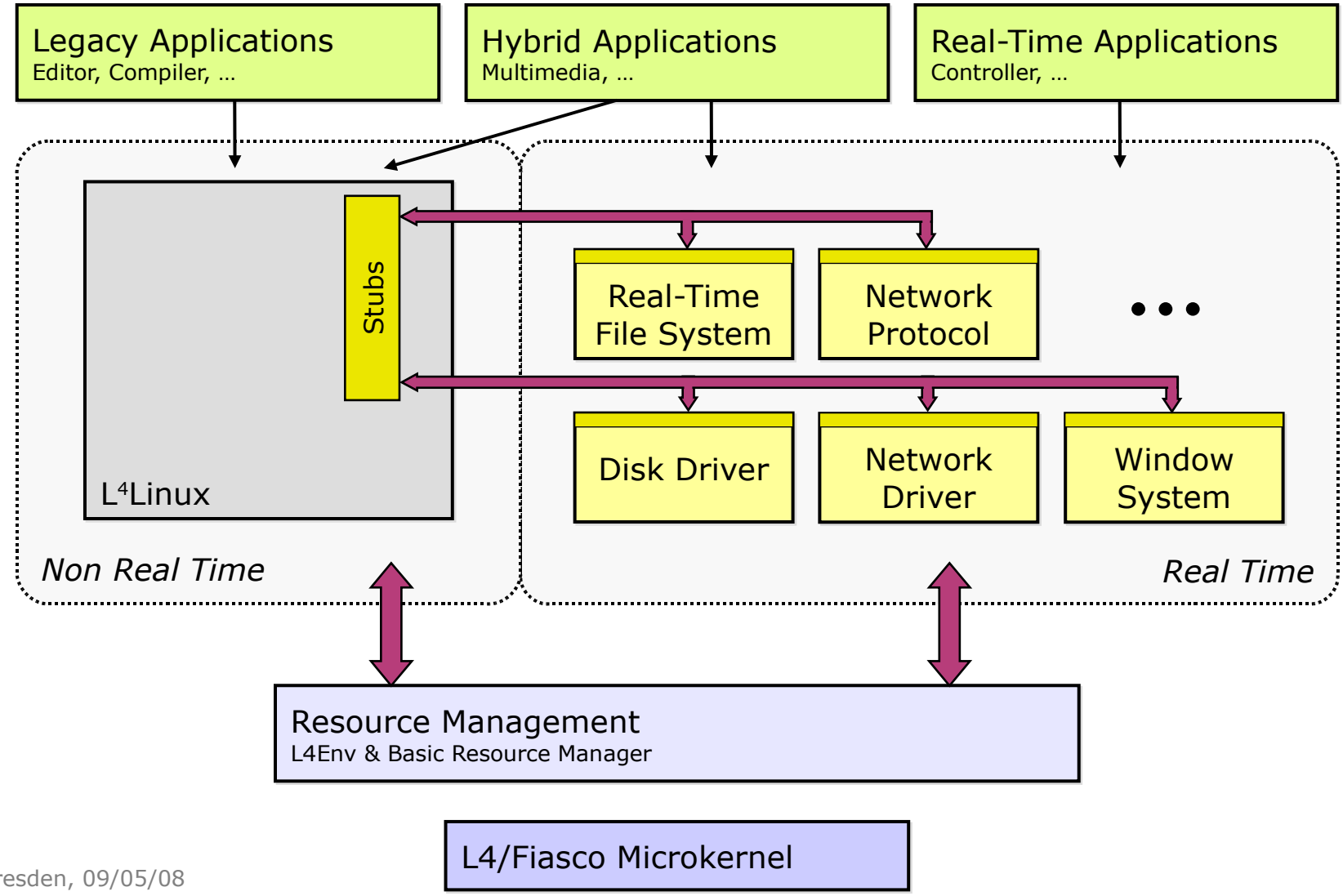


- Description of component functionality
- Functional and stream Interfaces
- Developed by application / component designer

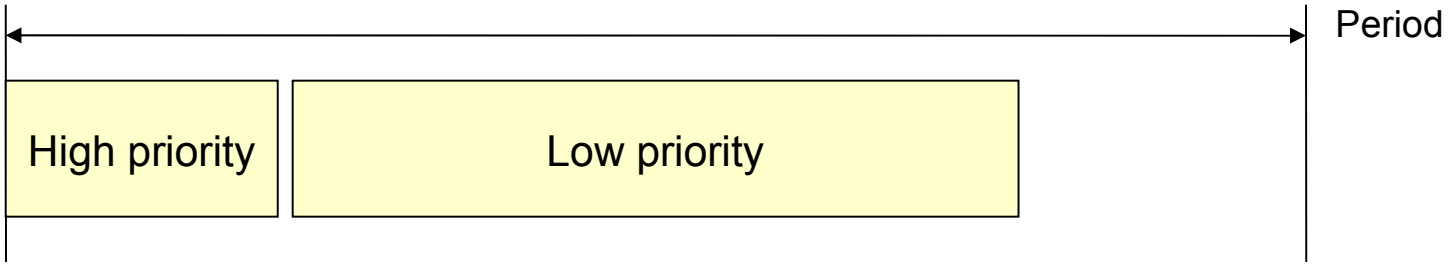
- Implementation of functional and stream interfaces
- Non-functional contracts
- Developed by component designer

- Component implementation installed into a container
- Possibly additional configurations (e.g., association of multiple implementations with one specification)
- Deployment performed by application designer

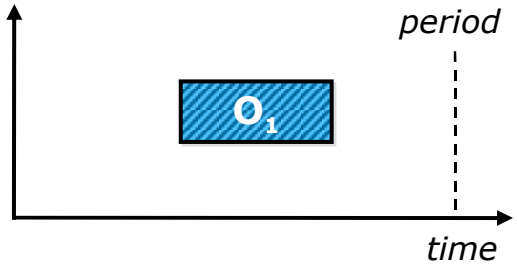
- Actual runtime instance of a component
- Includes allocation of resources at OS level
- Created by container based on user requests for specific functionality

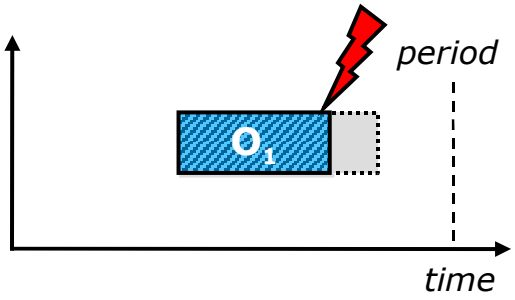

Legacy Applications
Editor, Compiler, …
Hybrid Applications
Multimedia, …
Real-Time Applications
Controller, …
Stubs
Real-Time File System
Network Protocol
• • •
Disk Driver
Network Driver
Window System
L4Linux
Non Real Time
Real Time
Resource Management
L4Env & Basic Resource Manager
L4/Fiasco Microkernel

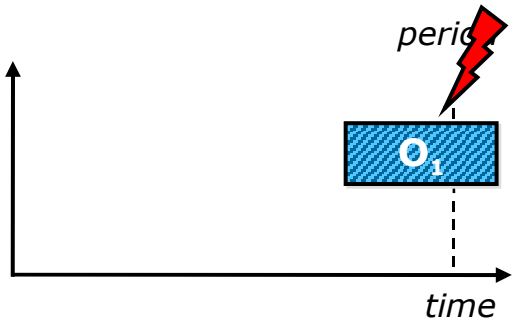
```
while (true) {
  next_period();
  // Mandatory part
  next_reservation();
  // Optional part
}
```

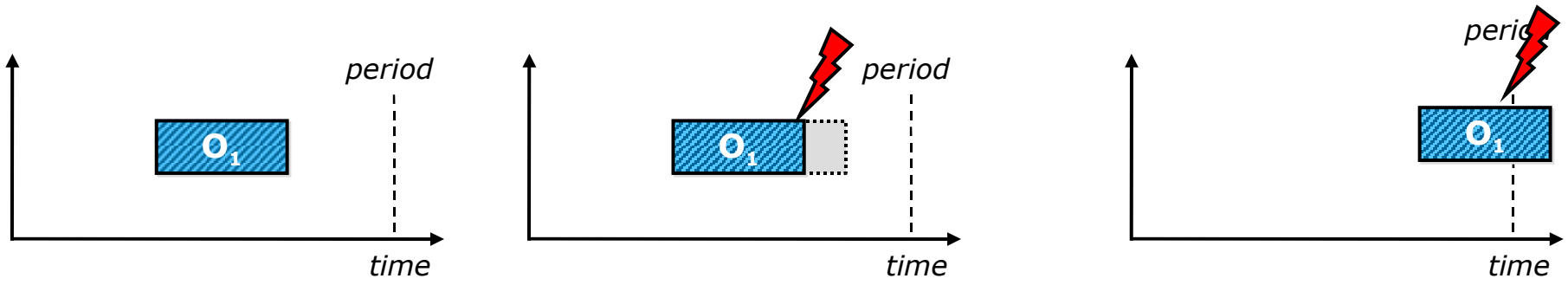


- **Periodic Processes**
  - Receive a number of quanta of a certain priority per period
- **End of quantum < end of period (missed deadline) or still work to do after end of quantum (exceeded quantum)**
  → Notification message

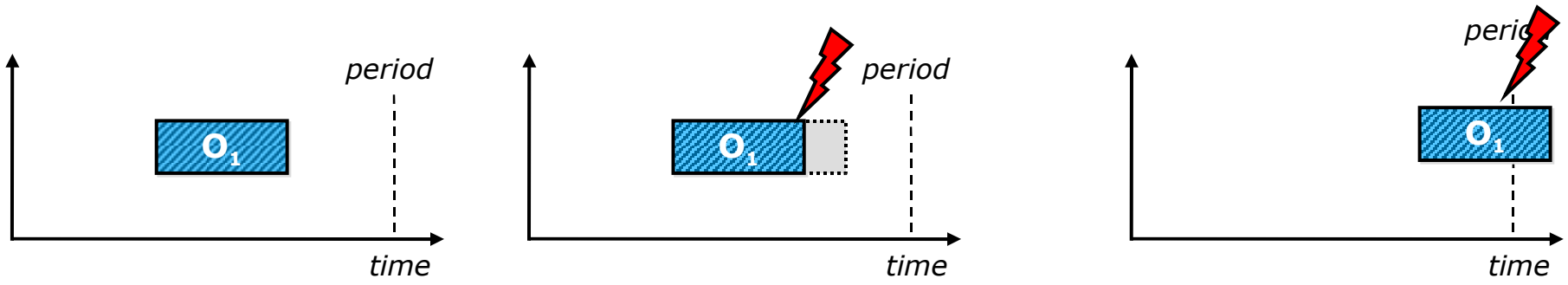

period
O1
time


period
O1
time


O1
time

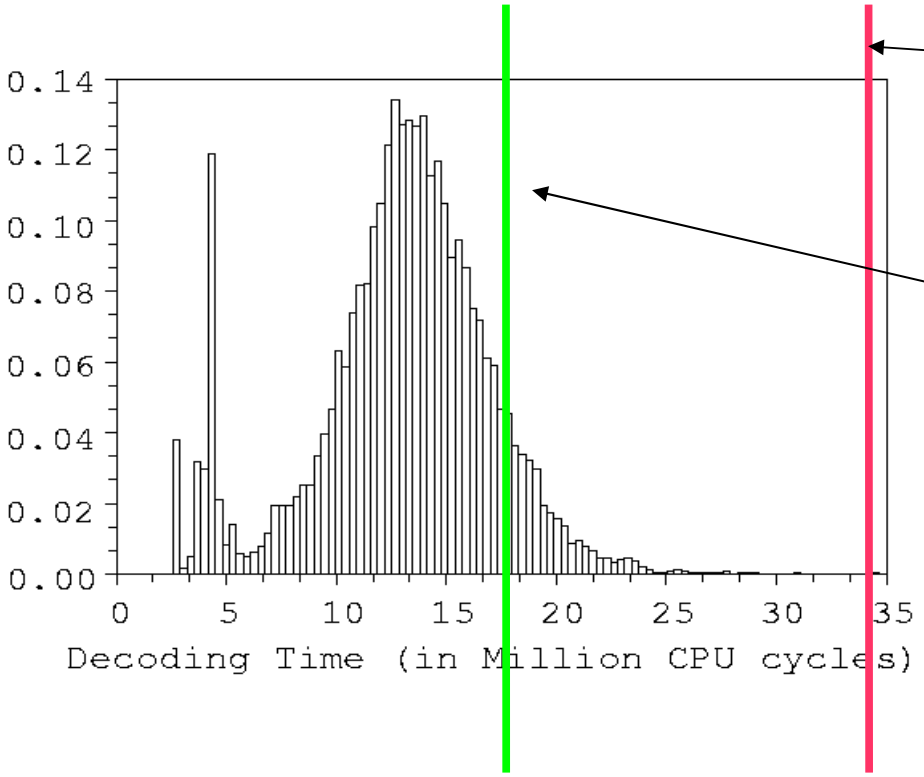



```
while (true) {
  message = receive_preemption_msg (theComponent, task_id);
  if ((message.type == MISSING_NEXT_RESERVATION) ||
      (message.type == MISSING_NEXT_PERIOD)) {
    run_optional = false;
  }
}
```




```
while (true) {
  message = receive_preemption_msg (theComponent, task_id);
  if ((message.type == MISSING_NEXT_RESERVATION) ||
      (message.type == MISSING_NEXT_PERIOD)) {
    run_optional = false;
  }
}
```

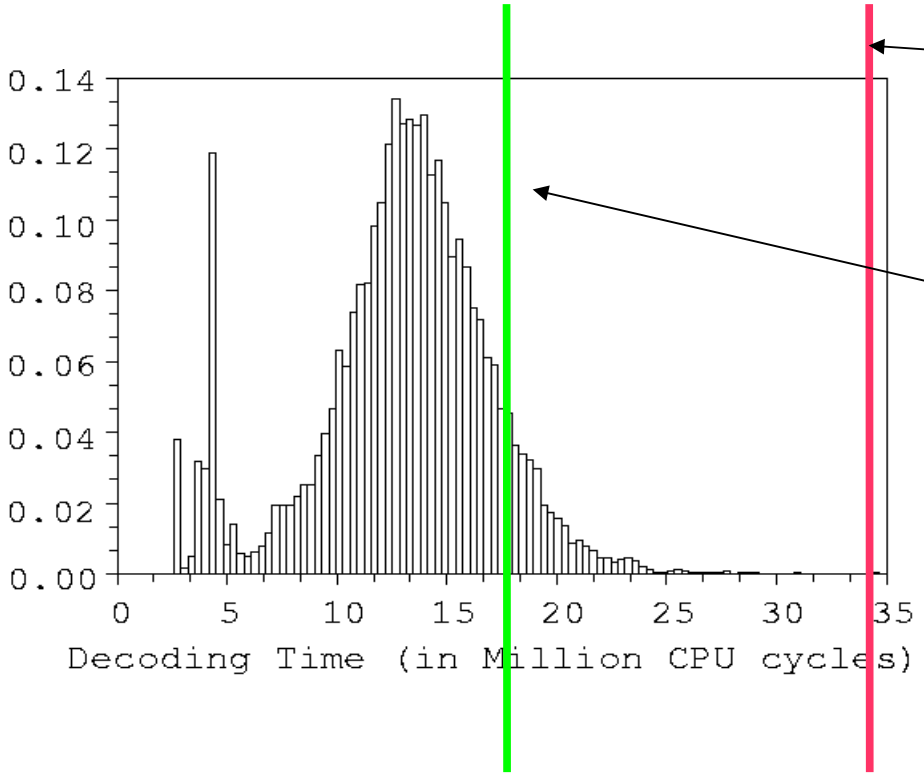
```
while (true) {
  next_period(); // Mandatory part
  next_reservation();
  if (run_optional){  // Optional part }
}
```



WCET

90% quantile

one frame

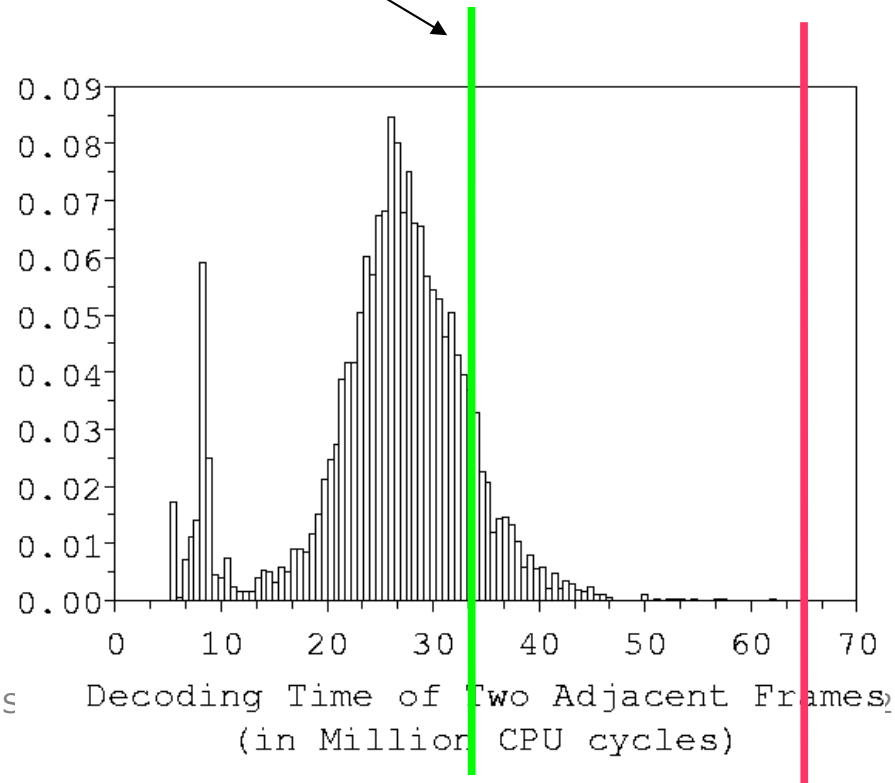


WCET

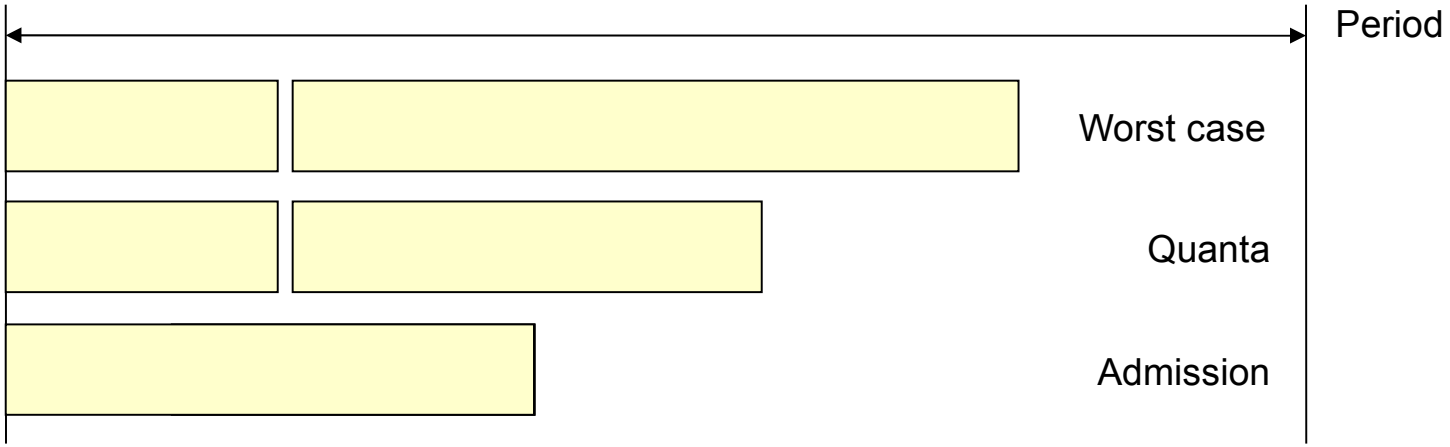
90% quantile

one frame

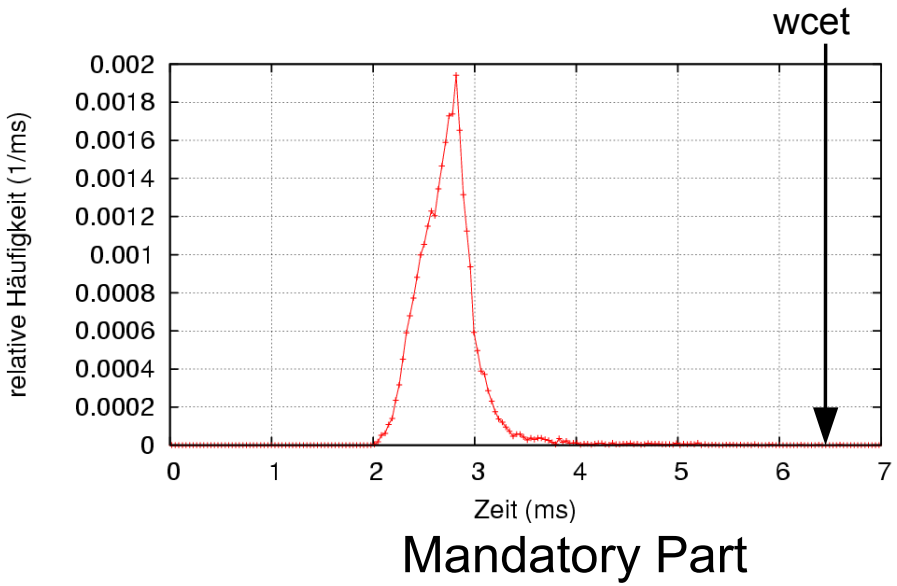
two frames



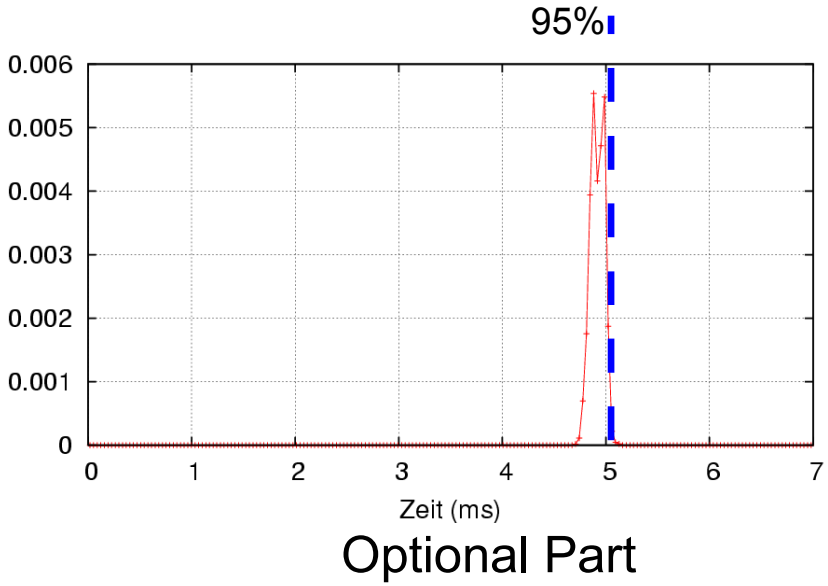
```
while (true) {
  next_period();
  decode


}
```



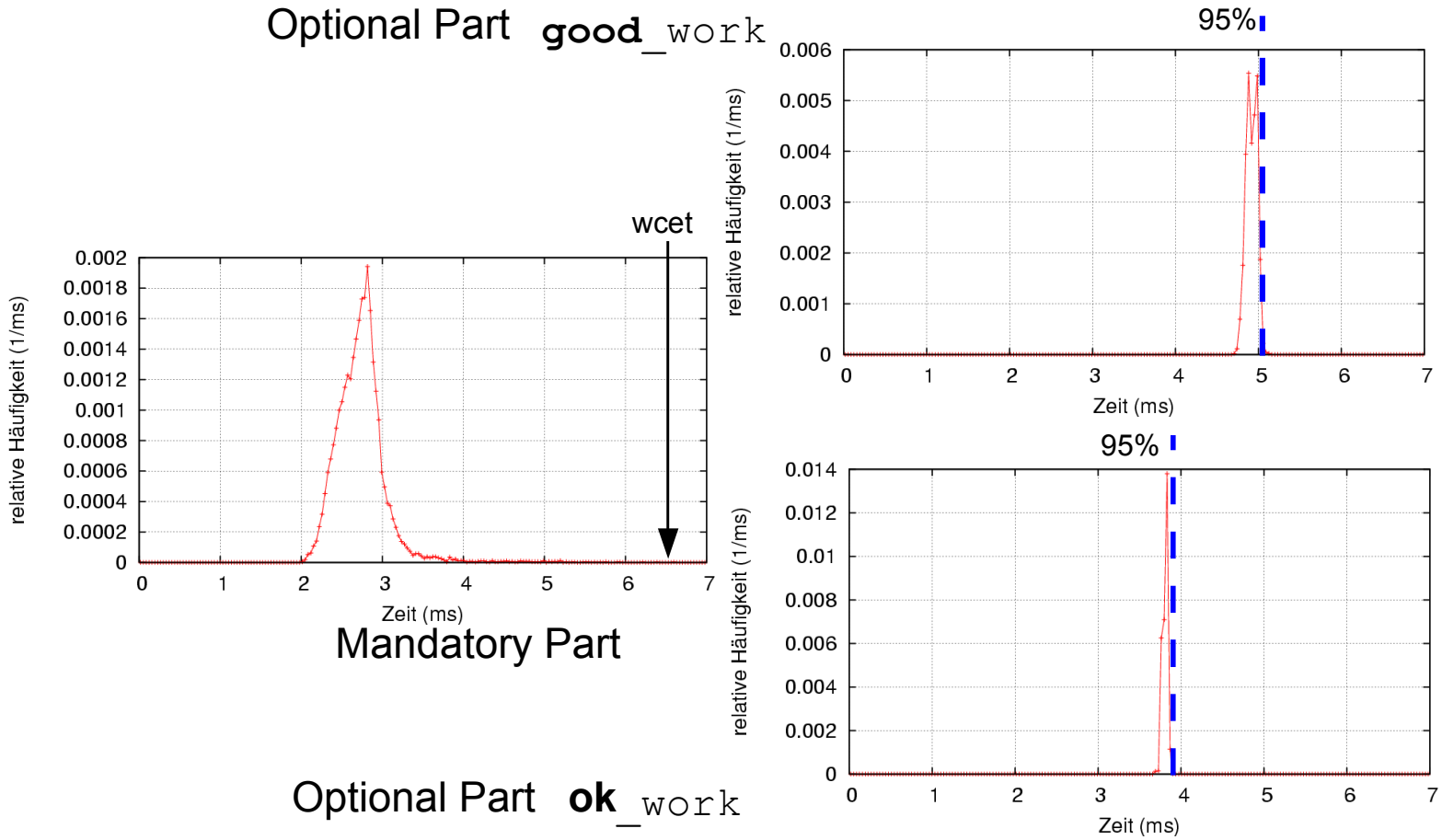
- **On average, only partial quanta are used**
  - → Controlled overbooking by the resource manager
  - → Requires execution time distributions



Mandatory Part



Optional Part


Optional Part good_work
95%
relative Häufigkeit (1/ms)
0.006
0.005
0.004
0.003
0.002
0.001
0
0 1 2 3 4 5 6 7
Zeit (ms)
wcet
relative Häufigkeit (1/ms)
0.002
0.0018
0.0016
0.0014
0.0012
0.001
0.0008
0.0006
0.0004
0.0002
0
0 1 2 3 4 5 6 7
Zeit (ms)
Mandatory Part
95%
relative Häufigkeit (1/ms)
0.014
0.012
0.01
0.008
0.006
0.004
0.002
0
0 1 2 3 4 5 6 7
Zeit (ms)
Optional Part ok_work

```
profile decodingBehavior for VernerVideoDecoder {
  profile good_work {
    provides decoder_quality_q3;
    uses demuxer_speed_25fps;
    resources decoder_good_cpu and decoder_good_memory;
  }
  profile ok_work {
    provides decoder_quality_q1;
    uses demuxer_speed_25fps;
    resources decoder_ok_cpu and decoder_good_memory;
  }

  transition good_work -> ok_work {
    setPostProcessQuality (Quality.q1);
  }

  …
}
```

```
resource memory {
  quality_characteristic size {
    domain: numeric kilobyte;
  }
}

quality decoder_good_memory {
  memory.size = 7168; -- 7 MB
}
```

```
XML:
<resource>
  <name>memory</name>
  <properties>
    <property>
      <name>size</name>
      <value>7168</value>
    </property>
  </properties>
</resource>
```

```
XML simplified:

{ "memory",
  { "size", "7168" }
}
```

Resource Specification for CPU (1)

```
resource cpu {
  quality_characteristic task_quality {
    domain: numeric [0..100] percent;
  }
  quality_characteristic execution_time {
    domain: distribution;
  }
  quality_characteristic task {
    domain: tupel {
      task_quality,
      execution_time
    };
  }
  quality_characteristic mandatory : task {
    domain: tupel {
      task_quality,
      execution_time
    };
    invariant: task_quality = 100;
  }
  quality_characteristic optional : task {}
  …
```

```
...
  quality_characteristic period {
    domain: numeric microseconds;
  }
  quality_characteristic demand {
    domain: tupel {
      period,
      mandatory,
      optional
    };
  }
}
```

Resource Specification for CPU (3)

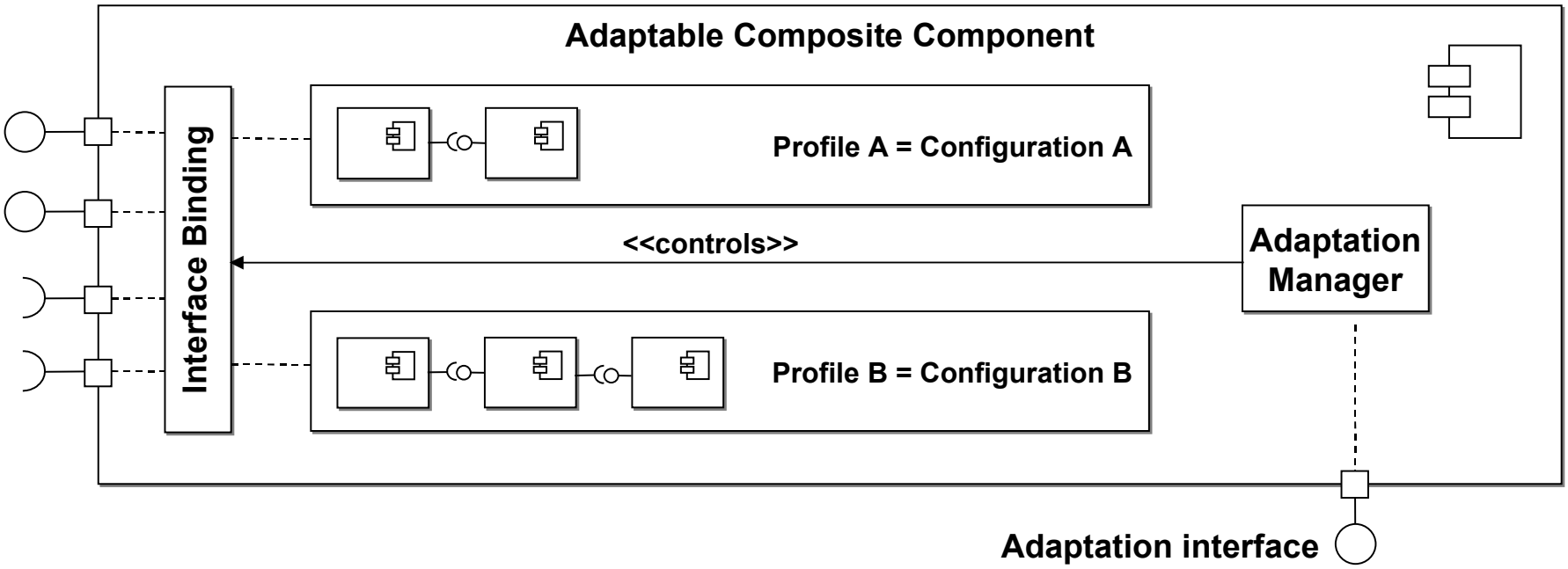
```
quality decoder_good_cpu {
  cpu.period = 40000;
  cpu.mandatory.task_quality = 100;
  cpu.mandatory.execution_time = "vdecoder_mand.dist";
  cpu.optional.task_quality >= 95;
  cpu.optional.execution_time = "vdecoder_opt_q3.dist";
}
```

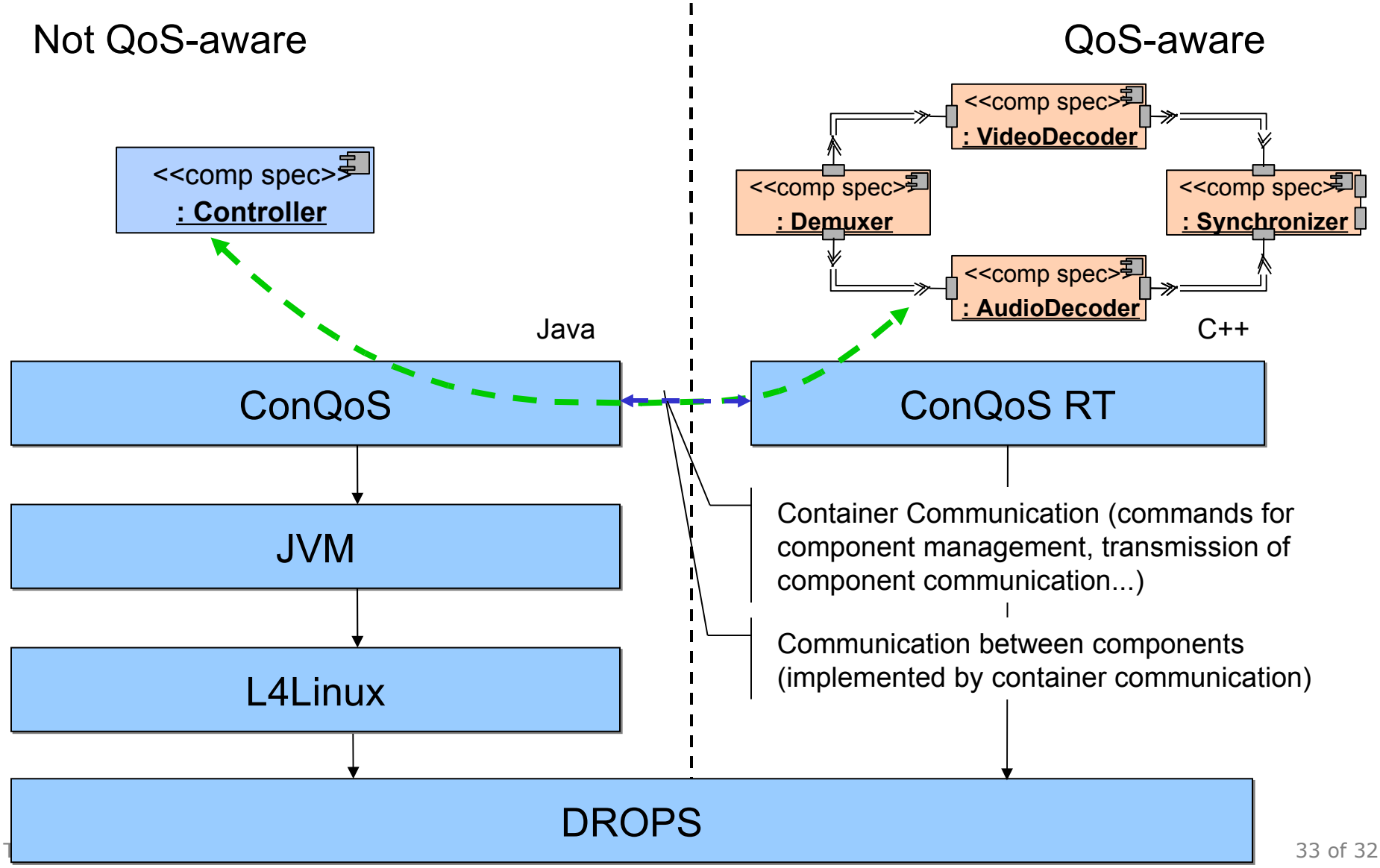
```
XML simplified:
{ "cpu",
  { "period", "40000" },
  { "mandatory.task_quality", "100" },
  { "mandatory.execution_time", "vdecoder_mand.dist" },
  { "optional.task_quality", "95" },
  { "optional.execution_time", "vdecoder_opt_q3.dist" }
}
```

Resource Specification for CPU (3)

```
quality decoder_ok_cpu {
  cpu.period = 40000;
  cpu.mandatory.task_quality = 100;
  cpu.mandatory.execution_time = "vdecoder_mand.dist";
  cpu.optional.task_quality >= 95;
  cpu.optional.execution_time = "vdecoder_opt_q1.dist";
}
```

```
XML simplified:
{ "cpu",
  { "period", "40000" },
  { "mandatory.task_quality", "100" },
  { "mandatory.execution_time", "vdecoder_mand.dist" },
  { "optional.task_quality", "95" },
  { "optional.execution_time", "vdecoder_opt_q1.dist" }
}
```


Adaptable Composite Component
Interface Binding
Profile A = Configuration A
<<controls>>
Adaptation Manager
Profile B = Configuration B
Adaptation interface


Not QoS-aware
QoS-aware
<<comp spec>>
: Controller
<<comp spec>>
: VideoDecoder
<<comp spec>>
: Demuxer
<<comp spec>>
: Synchronizer
<<comp spec>>
: AudioDecoder
Java
C++
ConQoS
ConQoS RT
JVM
L4Linux
DROPS
Container Communication (commands for component management, transmission of component communication...)
Communication between components (implemented by container communication)

- **Prediction-Enabled Component Technology (PECT)**
  - Software Engineering Institute at Carnegie Mellon University, USA
  - Components are transformed into task structure for execution
  - Components not visible at run time
  - No adaptation support
- **Safe-CCM**
  - PROGRESS institute at Mälardalen University, Sweden
  - Component model for embedded systems / automotive
  - Components are transformed into run-time structures → No components at run time
- **Real-Time CORBA (RT-CORBA)**
  - Object Management Group
  - Extension at the container level only
  - Real-time support must be imperatively implemented for each application
- **Framework for real-time embedded systems based on contracts (Frescor)**
  - Ongoing European project
  - Some parallels to presented approach, much more ambitious

- Ronald Aigner
- Steffen Göbel
- Christoph Pohl
- Martin Pohlack
- Simone Röttger
- Claude-Joachim Hamann
- Heinrich Hußmann

Operating Systems Group
http://os.inf.tu-dresden.de/

Software Engineering Group
http://st.inf.tu-dresden.de/

Computer Networks Group
http://rn.inf.tu-dresden.de/

Hermann Härtig, Steffen Zschaler, Martin Pohlack, Ronald Aigner, Steffen Göbel, Christoph Pohl, and Simone Röttger: *Enforceable Component-Based Realtime Contracts – Supporting Realtime Properties from Software Development to Execution.*
In **Springer Real-Time Systems Journal**, 35(1)2007, Springer, January, 2007.

DOI: 10.1007/s11241-006-9002-1

Claude-Joachim Hamann and Steffen Zschaler: *Scheduling Real-Time Components Using Jitter-Constrained Streams.* Journal of Object Technology, Special Issue on Quality of Service Management. 6(11)2007, http://www.jot.fm/issues/issue_2007_12/

Simone Röttger and Steffen Zschaler: *Tool Support for Refinement of Non-functional Specifications.* Journal on Software and Systems Modeling (SoSyM) 6(2)2007, Springer.

Steffen Göbel, Christoph Pohl, Ronald Aigner, Martin Pohlack, Simone Röttger, Steffen Zschaler: *The COMQUAD component container architecture.* J. Magee, C. Szyperski, and J. Bosch eds., 4th Working IEEE/IFIP Conference on Software Architecture (WICSA), pp. 315-318, Oslo, Norway. IEEE, 2004.

Steffen Göbel, Christoph Pohl, Simone Röttger, and Steffen Zschaler: *The COMQUAD Component Model – Enabling Dynamic Selection of Implementations by Weaving Non-functional Aspects.* Int'l Conference on Aspect-Oriented Software Development (AOSD'04), Lancaster. ACM, 2004.

Simone Röttger and Steffen Zschaler: *CQML +: Enhancements to CQML.* J.-M. Bruel, ed., Proc. 1st Int'l Workshop on Quality of Service in Component-Based Software Engineering, Toulouse, France, pages 43-56. Cépaduès- Éditions, 2003.

Jan Øyvind Aagedal. *Quality of Service Support in Development of Distributed Systems.* PhD thesis, University of Oslo, 2001.

- No fundamental difficulties encountered
- Communication between RTOS and SWE worlds can be difficult