Konfigurierbare Systemsoftware (KSS)

VL 7 – Summary and Discussion

Daniel Lohmann

Lehrstuhl für Informatik 4
Verteilte Systeme und Betriebssysteme

Friedrich-Alexander-Universität
Erlangen-Nürnberg

SS 16 – 2016-07-11

http://www4.informatik.uni-erlangen.de/Lehre/SS16/V_KSS

Agenda

The Operating System – A Swiss Army Knife?

The Operating System – A Swiss Army Knife?

Commodity operating systems provide a rich set of features to be prepared for all kinds of applications and contingencies:

- Malicious or erroneous applications
  - preemptive scheduling, address space separation, disk quotas
- Multi-user operation
  - authentication, access validation and auditing
- Multi-threaded and interacting applications
  - Threads, semaphores, pipes, sockets
- Many/large concurrently running applications
  - virtual memory, swapping, working sets

The Operating System – A Swiss Army Knife?

One size fits all? ↪ Variability

> *“Clearly, the operating system design must be strongly influenced by the type of use for which the machine is intended. Unfortunately it is often the case with ’general purpose machines’ that the type of use cannot be easily identified; a common criticism of many systems is that in attempting to be all things to all men they wind up being* ***totally satisfactory to no-one****.”*

Lister and Eager 1993: *Fundamentals of Operating Systems* [4]

The Operating System – A Swiss Army Knife?

One size fits all? ↪ Variability

> *“Clearly, the operating system design must be strongly influenced by the type of use for which the machine is intended. Unfortunately it is often the case with ’general purpose machines’ that the type of use cannot be easily identified; a common criticism of many systems is that in attempting to be all things to all men they wind up being* ***totally satisfactory to no-one****.”*

Lister and Eager 1993: *Fundamentals of Operating Systems* [4]

$\notin$

The Operating System – A Swiss Army Knife?

Big is beautiful? ↪ Granularity

> *“Some applications may require only a subset of services or features that other applications need. These ’less demanding’ applications should* ***not be forced to pay*** *for the resources consumed by unneeded features.”*

Parnas 1979: *“Designing Software for Ease of Extension and Contraction”* [5]

The Operating System – A Swiss Army Knife?

Big is beautiful? ↪ Granularity

> *“Some applications may require only a subset of services or features that other applications need. These ’less demanding’ applications should* ***not be forced to pay*** *for the resources consumed by unneeded features.”*

Parnas 1979: *“Designing Software for Ease of Extension and Contraction”* [5]

$\not\subseteq$

Between a Rock and a Hard Place...

functional and nonfunctional *requirements*

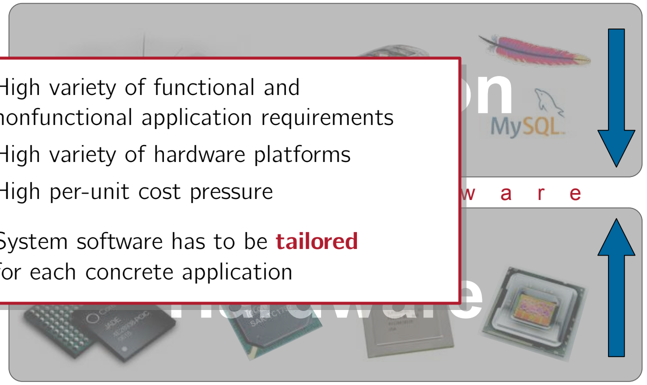
tasks
sockets
file system
...
event latency
safety
...

ISA
IRQ handling
MMU / MPU
...
cache size
coherence
IRQ latency
...

functional and nonfunctional *properties*

Between a Rock and a Hard Place...

functional and nonfunctional *requirements*



- High variety of functional and nonfunctional application requirements
- High variety of hardware platforms
- High per-unit cost pressure

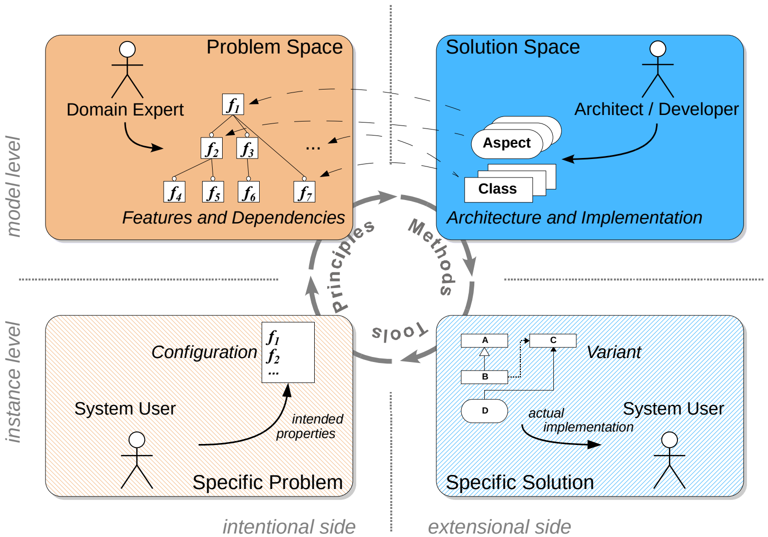
⇝ System software has to be **tailored** for each concrete application

tasks
sockets
file system
...
event latency
safety
...

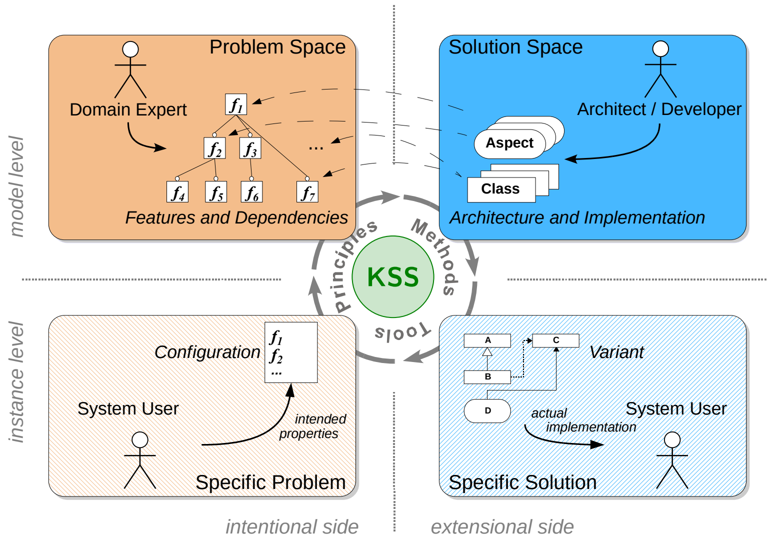
ISA
IRQ handling
MMU / MPU
...
cache size
coherence
IRQ latency
...

functional and nonfunctional *properties*

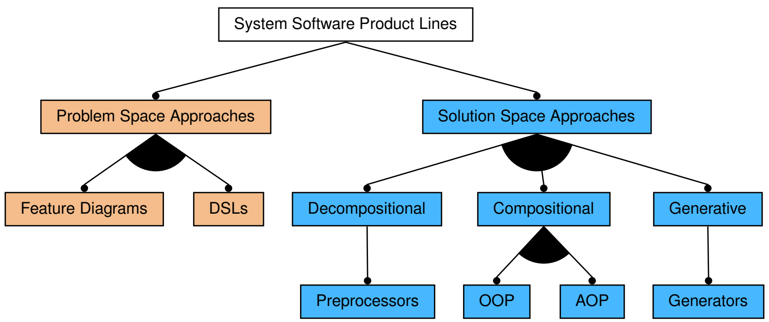
Configurable Software – Software Product Line



Configurable Software – Software Product Line



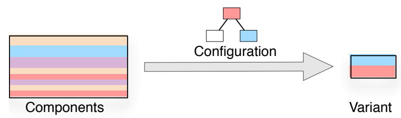
Software Product Line: Building Blocks



Focus: solution space techniques

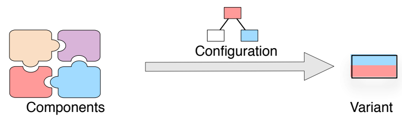
Implementation Techniques: Classification

- Decompositional Approaches



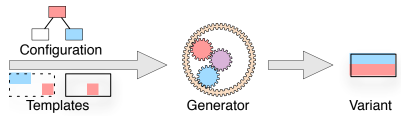
  - Text-based filtering (untyped)
  - Preprocessors

- Compositional Approaches



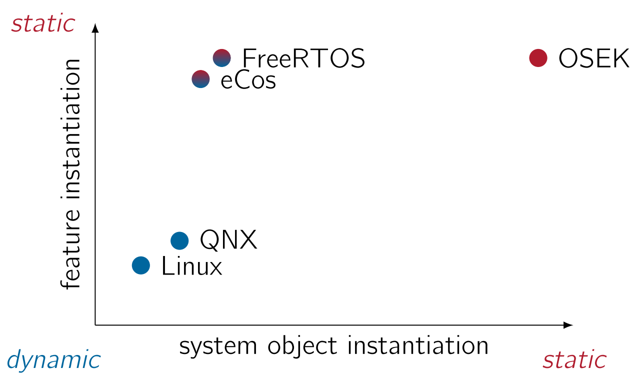
  - Language-based composition mechanisms (typed)
  - OOP, **AOP**, Templates

- Generative Approches

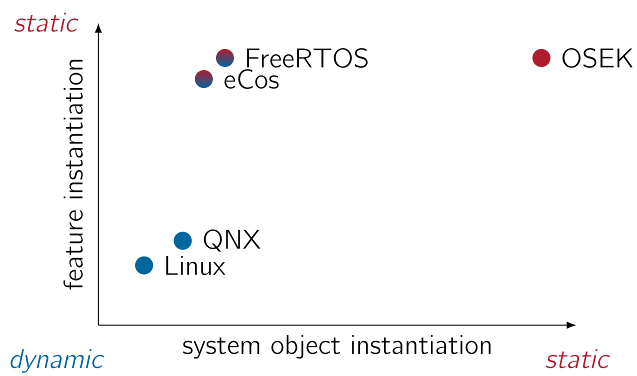


  - Metamodel-based generation of components (typed)
  - MDD, C++ TMP, generators

Feature vs. Instance-Based Configuration



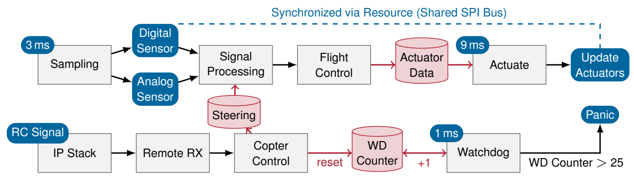
Feature vs. Instance-Based Configuration



Not only features, but also object instances are known at compile-time:

- Facilitates optimizations (static arrays instead of linked lists, . . .)
- Advantages wrt. footprint, latency, resilience, . . .

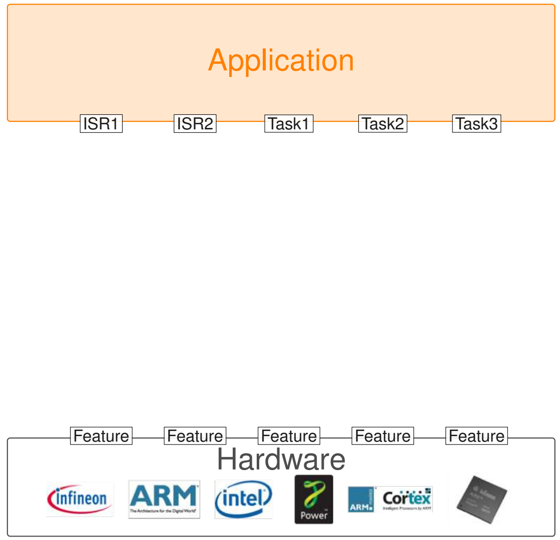
Feature vs. Instance-Based: Case Study [2]



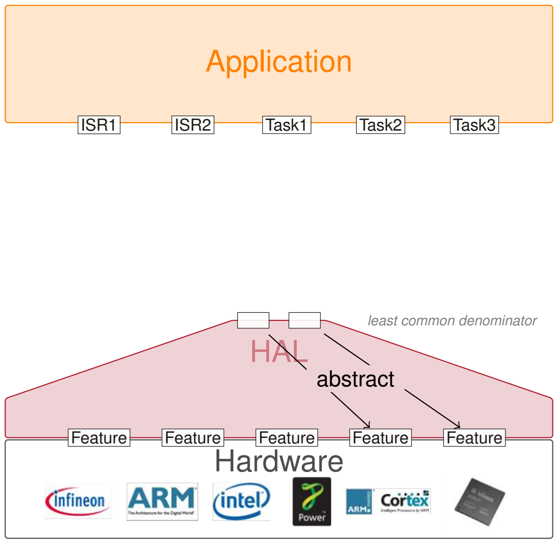
- Real-world flight-control application (11 tasks, 3 alarms, 1 ISR)
- Results with eCos and ERIKA Enterprise (open source OSEK)

	eCos	ERIKA	*factor*
kernel code (bytes)	14763	6765	2.2x
kernel time (instructions)	88465	46087	1.9x
robustness (10^9 SDCs)	148	18	8.2x

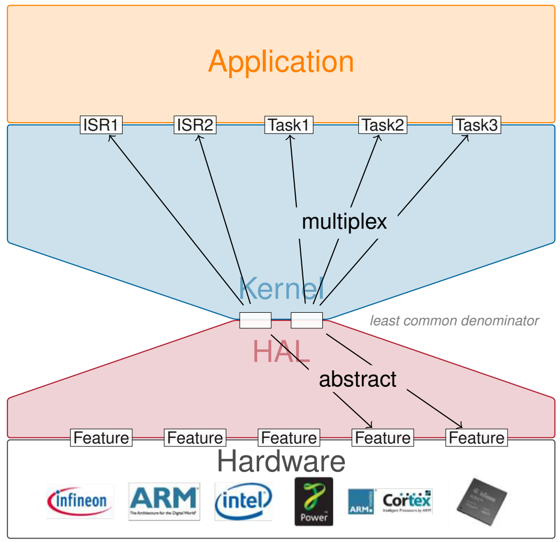
Traditional Operating-System Design



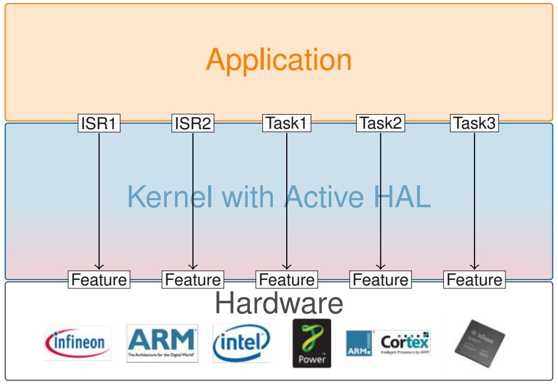
Traditional Operating-System Design



Traditional Operating-System Design



Hardware-Centric Operating-System Design



Agenda

*d*OSEK: Dependability-Oriented Static Embedded Kernel

An extremely fault-tolerant OSEK implementation

RTAS '15 [3]

- Dependability by constructive measures
  - Employ standard hardware memory protection
  - Agressive avoidance of indirections ⇝ lots of inlining
  - Arithmetic encoding of the kernel path (scheduler)

*d*OSEK: Dependability-Oriented Static Embedded Kernel

An extremely fault-tolerant OSEK implementation

RTAS '15 [3]

- Dependability by constructive measures
  - Employ standard hardware memory protection
  - Agressive avoidance of indirections ⇝ lots of inlining
  - Arithmetic encoding of the kernel path (scheduler)
- Scenario: quadrotor flight-control application
  - 11 tasks, 3 alarms, 1 ISR
  - 53 syscall invocations

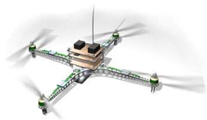
- Results (compared to ERIKA enterprise)
  - SDC reduction by **5 orders of magnitude**: $10^9 \longrightarrow 10^4$ SDCs
  - Code size increases by **factor 25**: $8 \longrightarrow 200$ KiB
  - Syscall latency increases by **factor 4**: $100 \longrightarrow 400$ cycles

*d*OSEK: Dependability-Oriented Static Embedded Kernel

An extremely fault-tolerant OSEK implementation

RTAS '15 [3]

- Dependability by constructive measures
  - Employ standard hardware memory protection
  - Agressive avoidance of indirections ⇝ lots of inlining
  - Arithmetic encoding of the kernel path (scheduler)
- Scenario: quadrotor flight-control application
  - 11 tasks, 3 alarms, 1 ISR
  - 53 syscall invocations



- Results (compared to ERIKA enterprise)
  - SDC reduction by **5 orders of magnitude**: $10^9 \longrightarrow 10^4$ SDCs
  - Code size increases by **factor 25**: $8 \longrightarrow 200$ KiB
  - Syscall latency increases by **factor 4**: $100 \longrightarrow 400$ cycles

Culprit: arithmetically encoded scheduler ⇝ **avoid scheduling!**

An OSEK System

Task 1; Priority 4

```
TASK(Task1) {
    int data = read_data();
    if (data == '\0') {
      ActivateTask(Task3);
    } else {
      bb_put(data);
    }
    ChainTask(Task2);
}
```

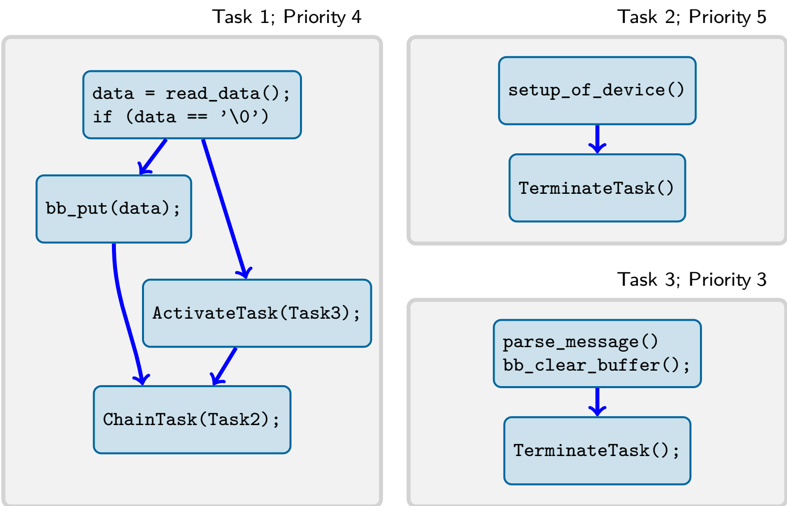
Task 2; Priority 5

```
TASK(Task2) {

  setup_of_device();

  TerminateTask();
}
```

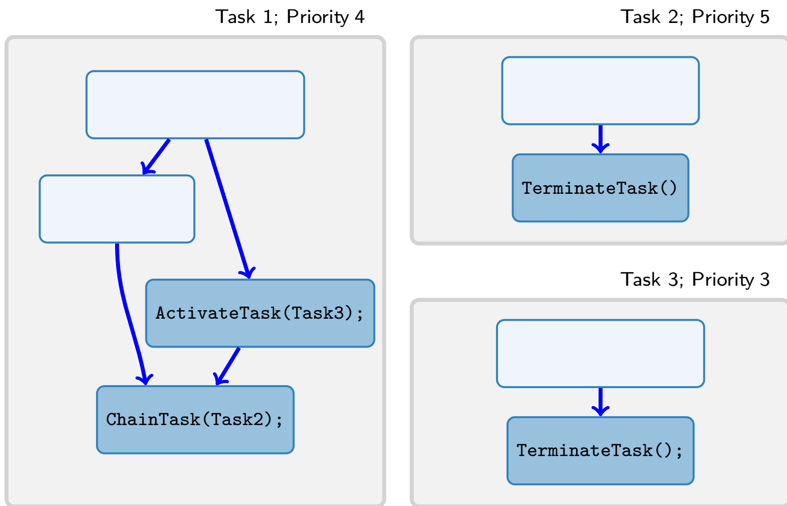
Task 3; Priority 3

```
TASK(Task3) {
  parse_message();
  bb_clear_buffer();

  TerminateTask();
}
```

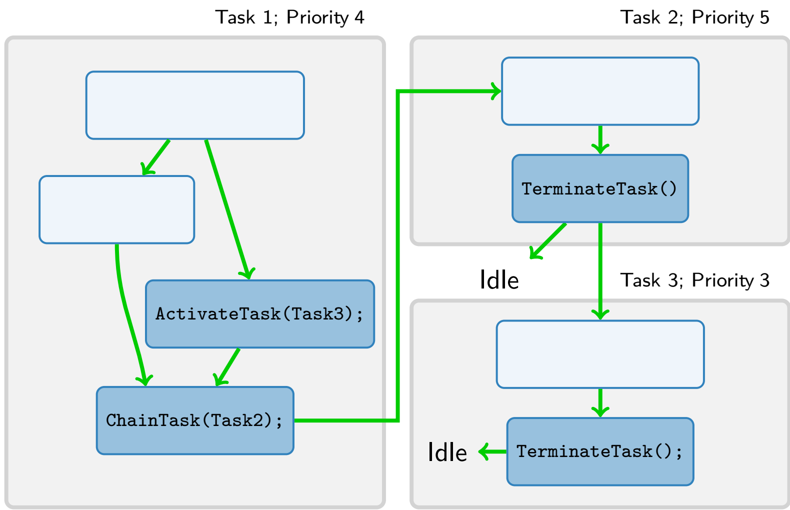

An OSEK System: Control-Flow Graphs



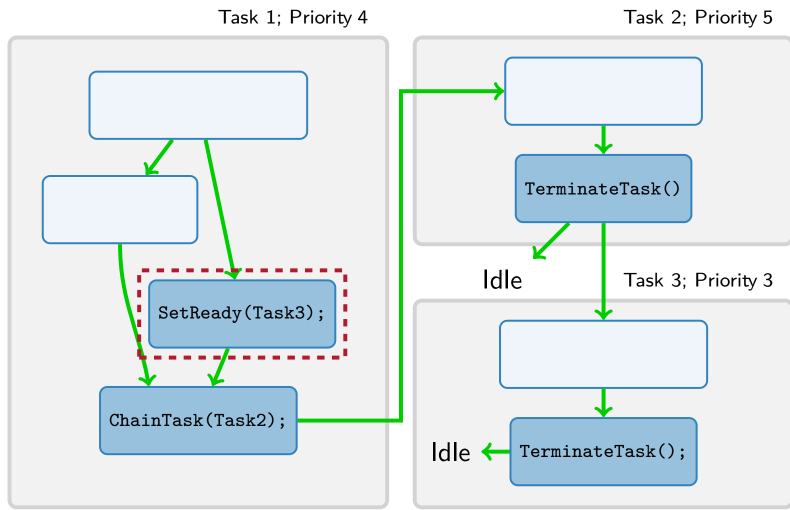
An OSEK System: Control-Flow Graphs



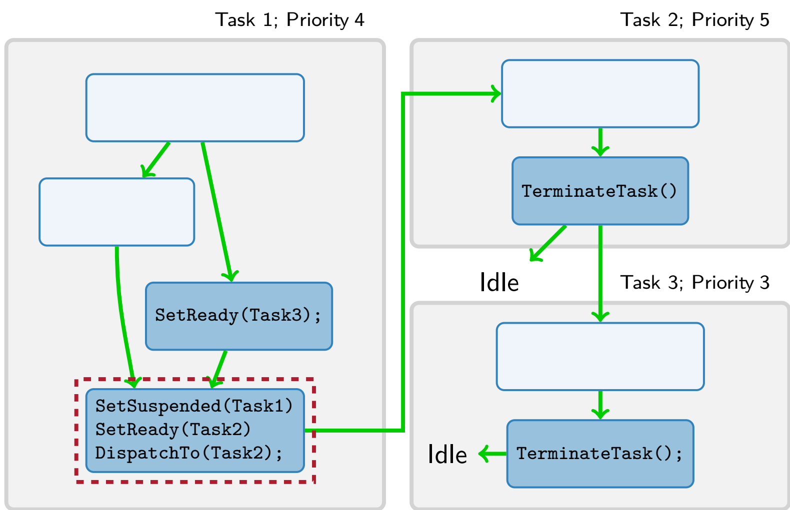
An OSEK System: Global Control-Flow Graph



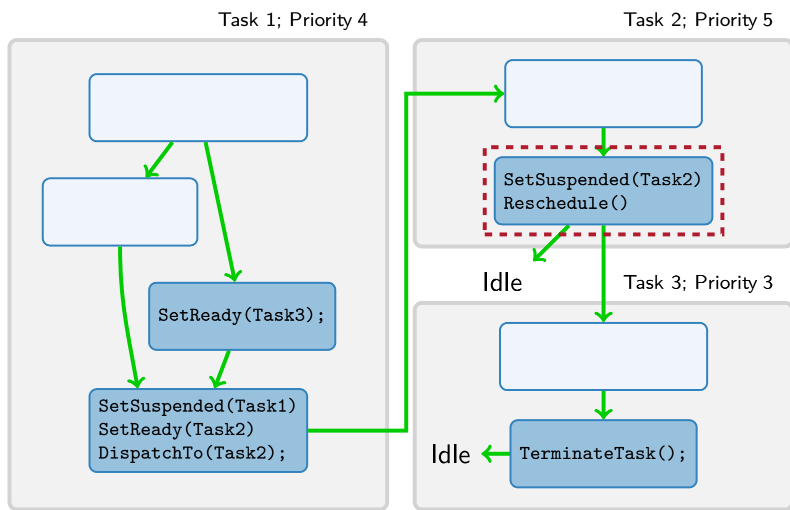
(Partial) Specialization of System Calls [1]



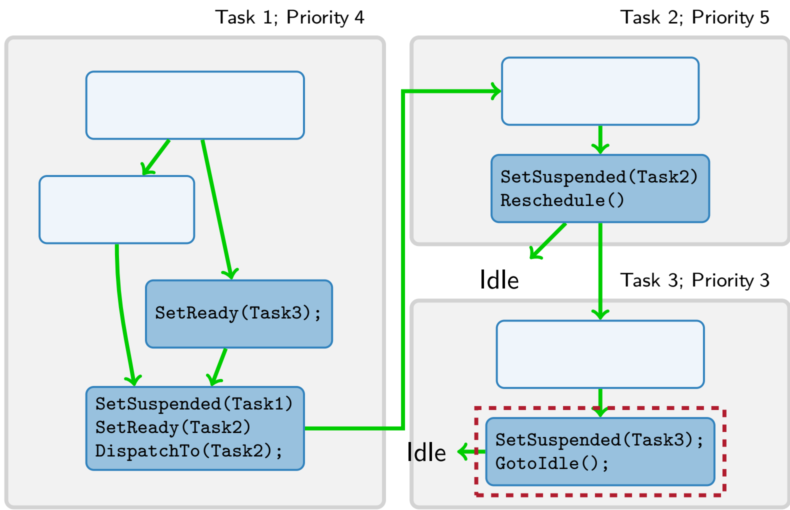
(Partial) Specialization of System Calls [1]



(Partial) Specialization of System Calls [1]



(Partial) Specialization of System Calls [1]



*d*OSEK: Dependability-Oriented Static Embedded Kernel

An extremely fault-tolerant OSEK implementation

RTAS '15 [3]

- Dependability by constructive measures
  - Employ standard hardware memory protection
  - Agressive avoidance of indirections ⇝ lots of inlining
  - Arithmetic encoding of the kernel path (scheduler)
- Scenario: quadrotor flight-control application
  - 11 tasks, 3 alarms, 1 ISR
  - 53 syscall invocations

- Results (compared to ERIKA enterprise)
  - SDC reduction by **5 orders of magnitude**: $10^9 \longrightarrow 10^4$ SDCs
  - Code size increases by **factor 25**: $8 \longrightarrow 200$ KiB
  - Syscall latency increases by **factor 4**: $100 \longrightarrow 400$ cycles

*d*OSEK: Dependability-Oriented Static Embedded Kernel

An extremely fault-tolerant OSEK implementation

RTAS '15 [3]

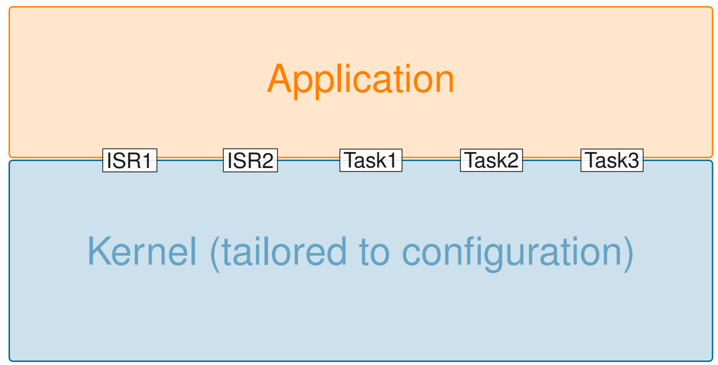
- Dependability by constructive measures
  - Employ standard hardware memory protection
  - Agressive avoidance of indirections ⇝ lots of inlining
  - Arithmetic encoding of the kernel path (scheduler)
- Scenario: quadrotor flight-control application
  - 11 tasks, 3 alarms, 1 ISR
  - 53 syscall invocations

  } 243 GCFG edges

- Results with **call-site specialization** LCTES '15 [1]
  - SDC reduction by **5 orders of magnitude**: $10^9 \longrightarrow 10^4$ SDCs
  - Code size increases by **factor 10.5**: $8 \longrightarrow 85$ KiB
  - Syscall latency increases by **factor 1.5**: $100 \longrightarrow 150$ cycles

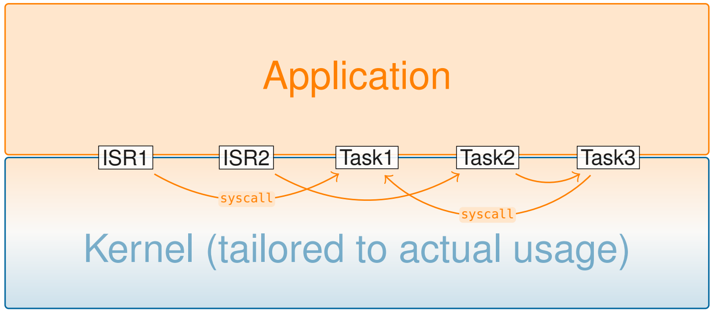
⇝ **Further application-specific tailoring pays off!**

Instance-Based Tailoring (e.g., based on OIL file)



- Kernel constrained to specified **features** and **system objects**.

Interaction-Based Tailoring (e.g., based on GCFG analysis)



- Kernel constrained to specified **features** and **system objects**.
- Further constrained to **actually possible** *app* → *kernel* **interactions**.

Agenda

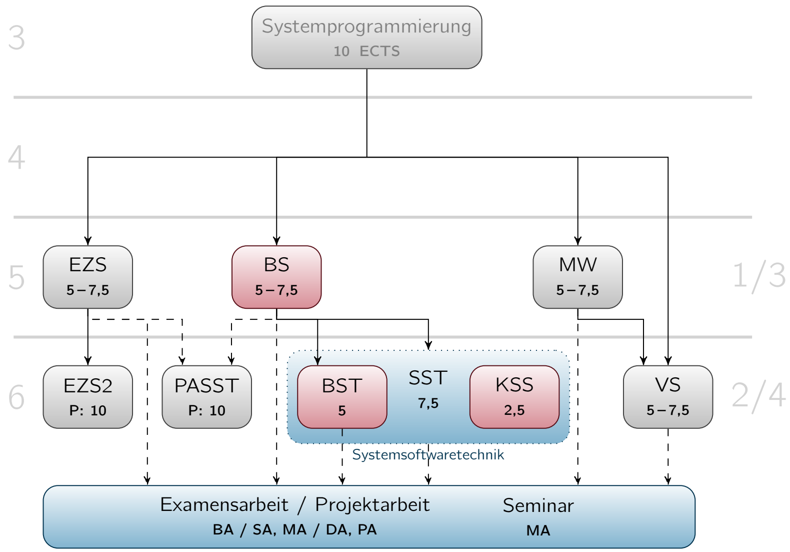
Evaluation

...

Diskussion

- Am coolsten finde / fand ich...
- Ich habe vermisst...
- Bei einer Erweiterung auf 5 ECTS...

Wie gehts weiter? (Bachelor/Master)



Referenzen

[1] Christian Dietrich, Martin Hoffmann, and Daniel Lohmann. “Cross-Kernel Control-Flow-Graph Analysis for Event-Driven Real-Time Systems”. In: *Proceedings of the 2015 ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems (LCTES ’15)*. (Portland, Oregon, USA). New York, NY, USA: ACM Press, June 2015. isbn: 978-1-4503-3257-6. doi: `10.1145/2670529.2754963`.

[2] Martin Hoffmann, Christoph Borchert, Christian Dietrich, Horst Schirmeier, Rüdiger Kapitza, Olaf Spinczyk, and Daniel Lohmann. “Effectiveness of Fault Detection Mechanisms in Static and Dynamic Operating System Designs”. In: *Proceedings of the 17th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC ’14)*. (Reno, Nevada, USA). IEEE Computer Society Press, 2014, pp. 230–237. doi: `10.1109/ISORC.2014.26`.

[3] Martin Hoffmann, Florian Lukas, Christian Dietrich, and Daniel Lohmann. “dOSEK: The Design and Implementation of a Dependability-Oriented Static Embedded Kernel”. In: *Proceedings of the 21st IEEE International Symposium on Real-Time and Embedded Technology and Applications (RTAS ’15)*. Washington, DC, USA: IEEE Computer Society Press, 2015, pp. 259 –270. doi: `10.1109/RTAS.2015.7108449`.

[4] A.M. Lister and R.D. Eager. *Fundamentals of Operating Systems*. 5th. Macmillian, 1993. isbn: 0-333-46986-0.

Referenzen (Cont’d)

[5] David Lorge Parnas. “Designing Software for Ease of Extension and Contraction”. In: *IEEE Transactions on Software Engineering* SE-5.2 (1979), pp. 128–138.