Composing Real-Time Applications from Communicating Black-Box Components

Martin Becker[*], Alejandro Masrur[+], Samarjit Chakraborty[*]

[*]TU Munich, Institute for Real-Time Computer Systems (RCS)
[+]TU Chemnitz, Software Technology for Embedded Systems (STES)

Tokyo, 2014 January 22nd

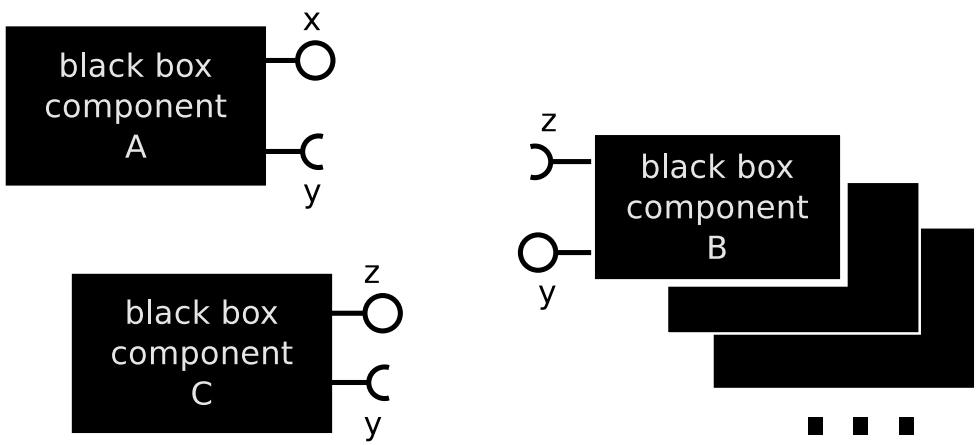
Lehrstuhl für
Realzeit-Computersysteme

Technische Universität München

Outline

Motivation

- real-time systems, consisting of individually developed components
  - common in industrial setups: different suppliers to be integrated, black boxes, not known to each other
  - different modeling paradigms and tools in each domain
  - *we are not trying to unify all of them*
- free developer from the burden of integration:
  - decide on execution schedule, manage communication (buffering, transport, different, interfaces, data consistency), ensure completeness and correctness (determinism, reactivity), ...

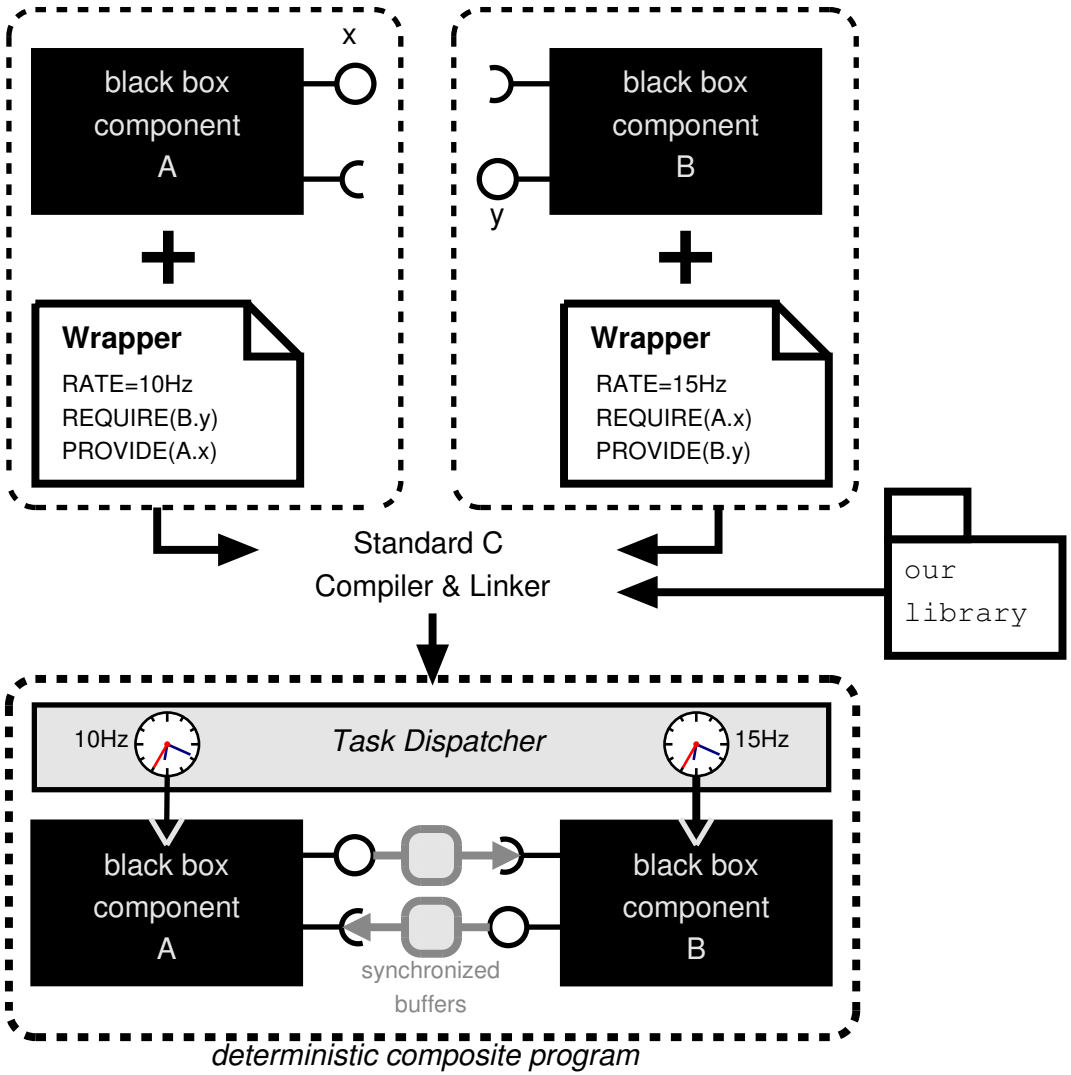


Motivation (2)

How could this work be interesting for you?

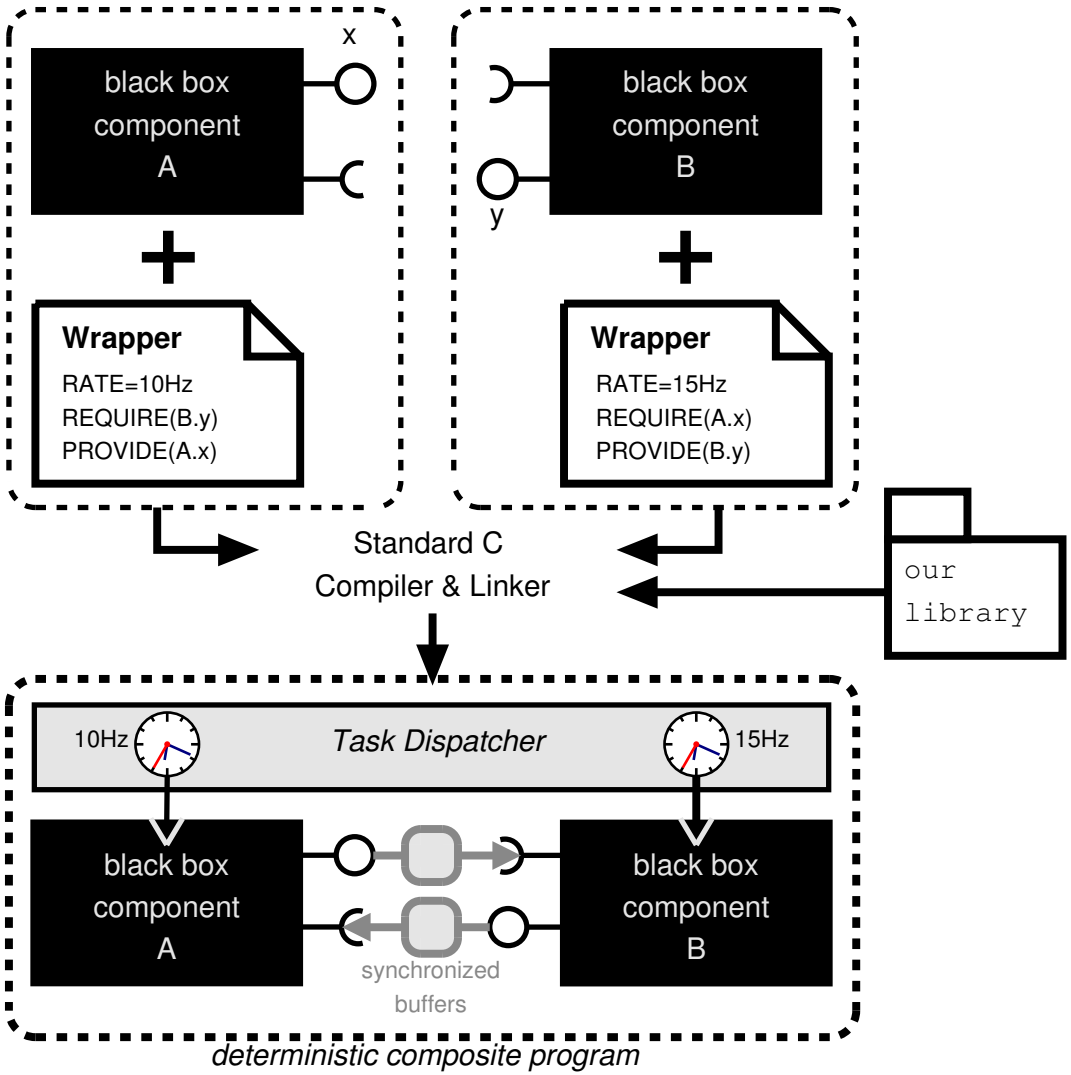
- **theory:** proposing and analyzing a simple, deterministic multi-rate model of communication
- **software development:** offering shortcut to synchronizing components running at different rates
- **hardware development:** same here

Solution: Timed Containers



- components are wrapped into *containers*

Solution: Timed Containers



- components are wrapped into *containers*
- containers automatically solve communication and execution schedule
- user gets guarantees
- math. proof of correctness

Solution: User's View

1. declare each black-box (BB) component
   1. identifier
   2. interface (inputs, outputs)
   3. execution rate
   4. (delay constraints)
2. instantiate BB (and connect to I/O ports)
3. compile+link as usual $\Rightarrow$ executable

Result: well-timed system of semantically correct communicating black box components

✓Composing RT system on a high-level view. No need to care for scheduling or communication mechanisms.

What is a black box (BB) exactly?

Here:

- known interfaces
  - I/O direction, types
- bounded worst-case execution time
  - i.e., no indefinitely blocking calls, no infinite loops
- time-discrete, periodic computation
  - in time steps T_i
- fixed binding of logical to physical time (...algorithms...)
  - each BB invocation: $time = time + T_i$

Nothing else.

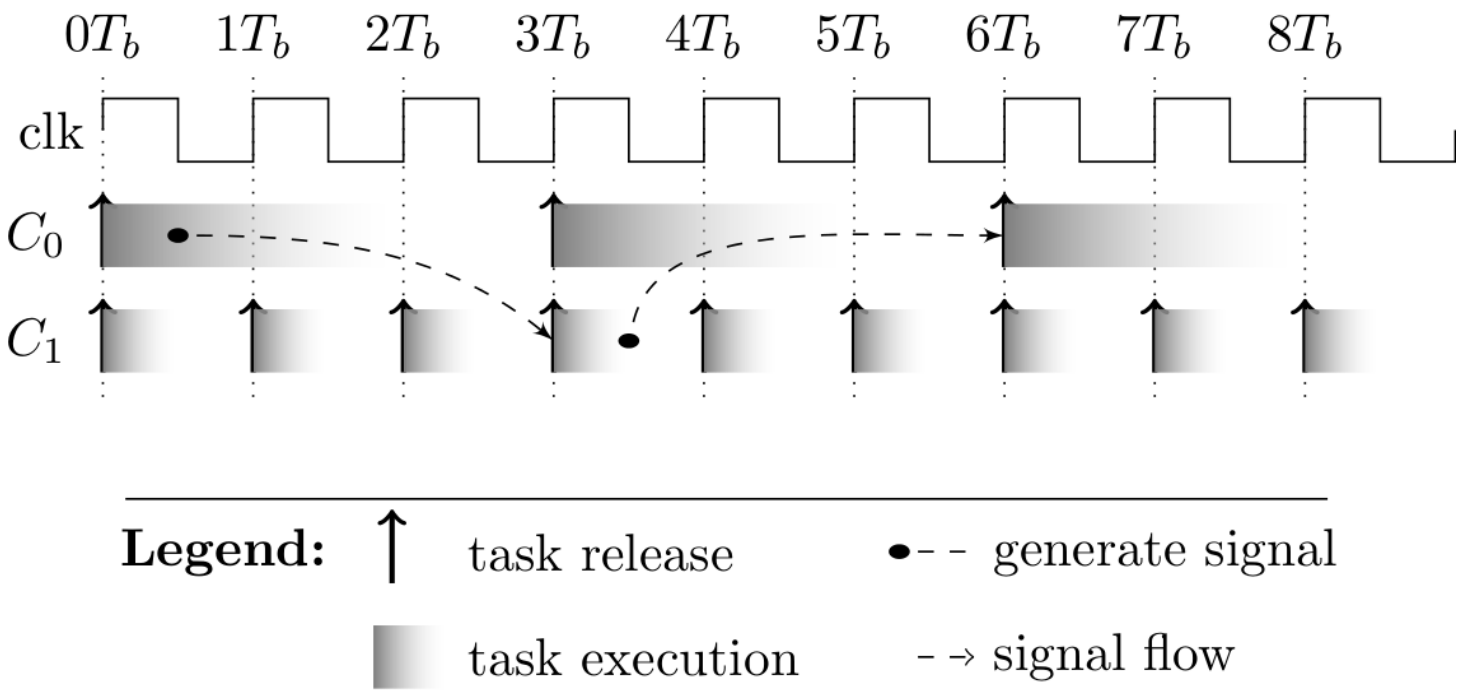
Execution Semantics

- **similarly to physical components:** components run logically in parallel to each other
- **typical for RT systems:** components run periodically
- underlying execution platform is abstracted out
  - components could run on a multi-core platform with OS
  - or on a bare-metal single core controller

In any case: periodic and parallel.

Communication Semantics

- components have *signals* and *events*
- ‘‘freshest value’’ semantics
- logically non-zero communication delay between components
  - guaranteed bounds (min,max)
- transport mechanism, synchronization etc. abstracted out



Verification of Semantics

If verification is successful, then the *implementation* (hardware+software) behaves the same as the abstract model.

- **things to prove:** parallel, periodic, freshest value, bounded delays, deterministic
- we need platform information only now (processor, caches, OS, ...)
- obtain (wort-case) execution time of each component
- show that platform's scheduling leads to (worst-case) response times short enough to preserve semantics:
  - components produce signals/events faster than *min*-bound
  - components finish execution before next execution starts

Reference Implementation for C

C library, works with any std. C89 compiler

- *Component integration* takes place during compilation and linking process.
- **Correctness and Completness:** Any non-deterministic use is prevented (no/multiple drivers, non-matching data type or unsynchronized signals/events).
- Further:
  - small memory footprint,
  - no dynamic memory allocation,
  - support for multi-threading,
  - provision of signal/event trace logging.

Intuitive usage for any C developer.

Example: Loopback Application

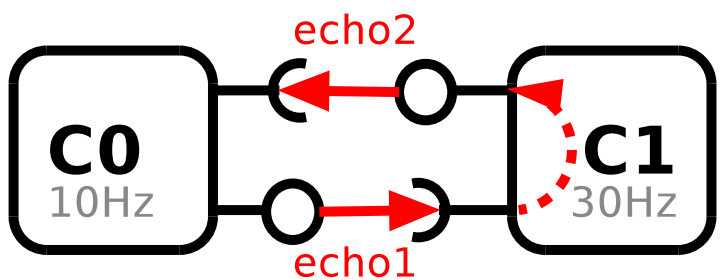
```
// file c0.c
#define CONT_NAME C0
CONT_PRESCALER(3);
ANNOUNCE_EVENT(echo_1);
REQUIRE_EVENT(echo_2);

void CONT_READIN_ME() {
  UPDATE_EVENT(echo_2);
}
void CONT_PROCESS_ME() {
  SET_EVENT(echo_1);
}
//
```

```
// file c1.c
#define CONT_NAME C1
CONT_PRESCALER(1);
ANNOUNCE_EVENT(echo_2);
REQUIRE_EVENT(echo_1);

void CONT_READIN_ME() {
  UPDATE_EVENT(echo_1);
}
void CONT_PROCESS_ME() {
  if (GET_EVENT(echo_1))
    SET_EVENT(echo_2);
}
```

with 30 Hz timer:



Example: Loopback Application

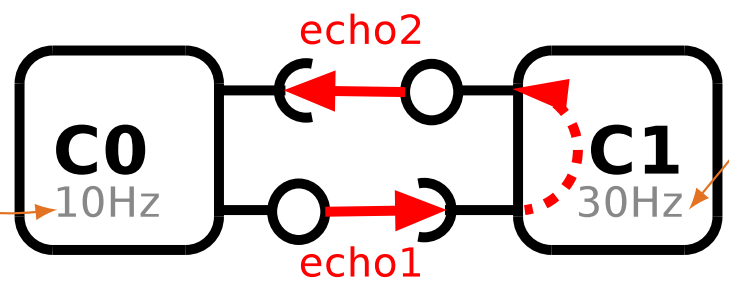
```
// file c0.c
#define CONT_NAME C0
CONT_PRESCALER(3);
ANNOUNCE_EVENT(echo_1);
REQUIRE_EVENT(echo_2);

void CONT_READIN_ME() {
  UPDATE_EVENT(echo_2);
}
void CONT_PROCESS_ME() {
  SET_EVENT(echo_1);
}
//
```

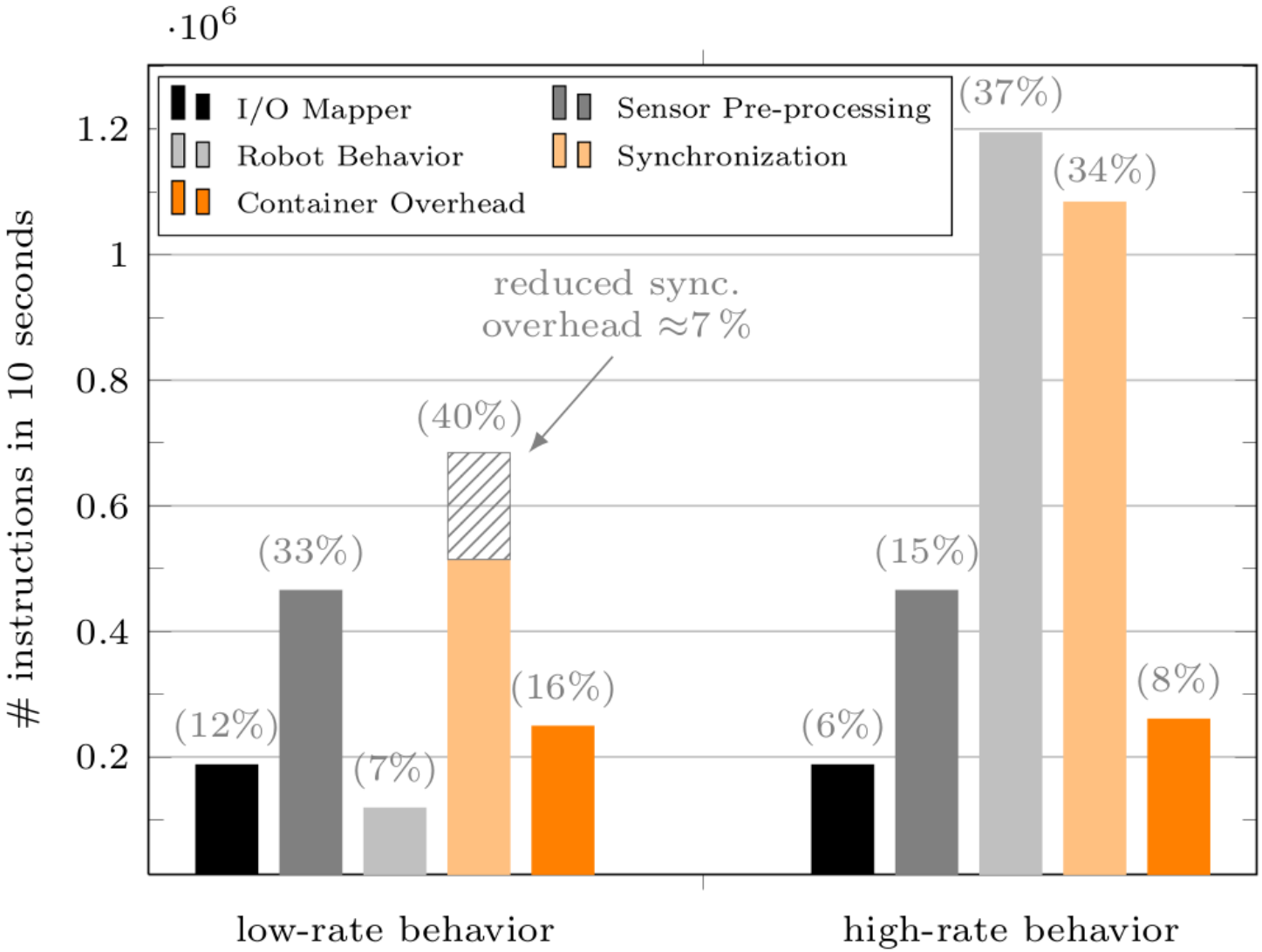
```
// file c1.c
#define CONT_NAME C1
CONT_PRESCALER(1);
ANNOUNCE_EVENT(echo_2);
REQUIRE_EVENT(echo_1);

void CONT_READIN_ME() {
  UPDATE_EVENT(echo_1);
}
void CONT_PROCESS_ME() {
  if (GET_EVENT(echo_1))
    SET_EVENT(echo_2);
}
```

with 30 Hz timer:



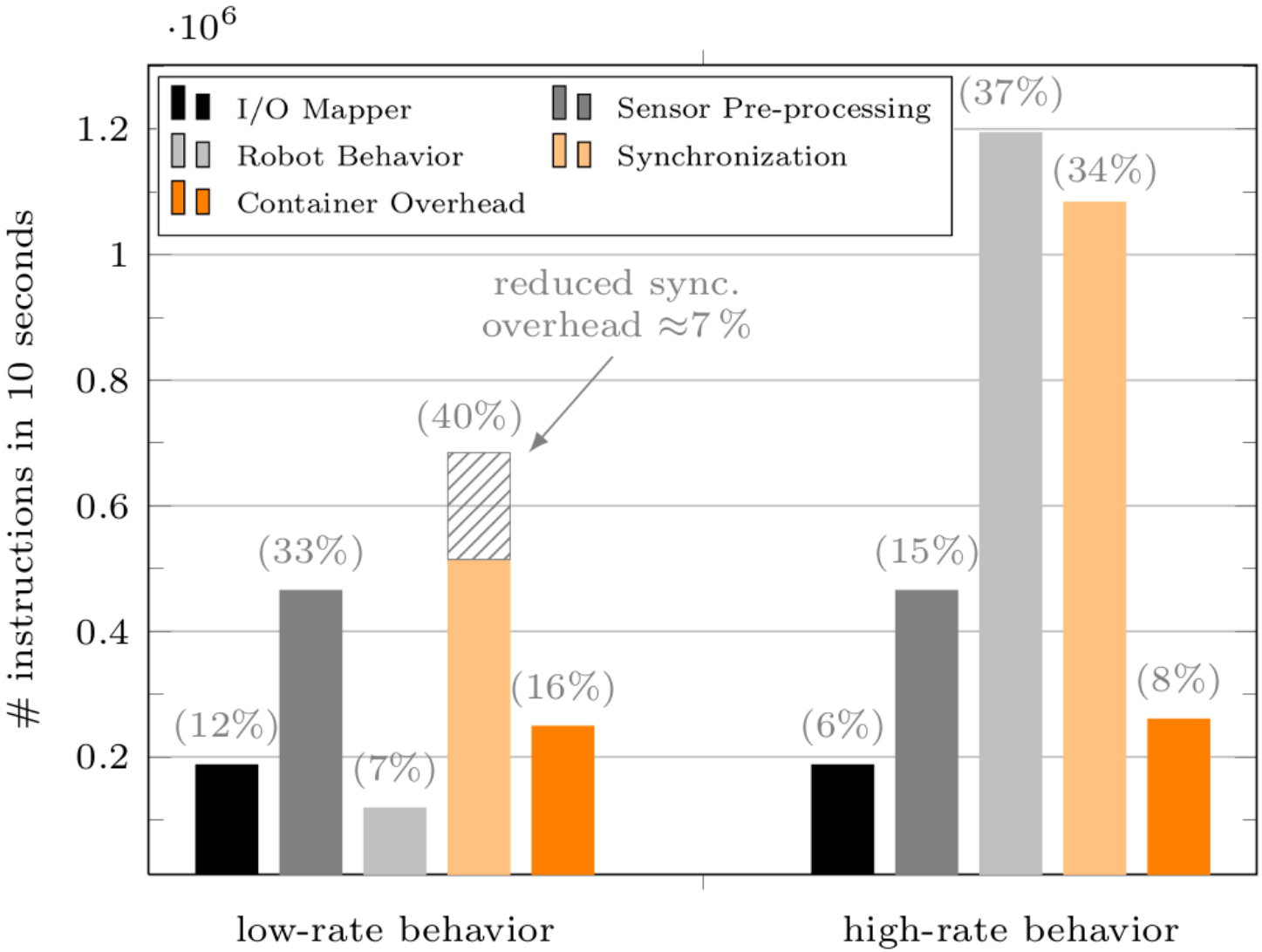
Performance Measurements (1)



10x rate change

...

Performance Measurements (1)

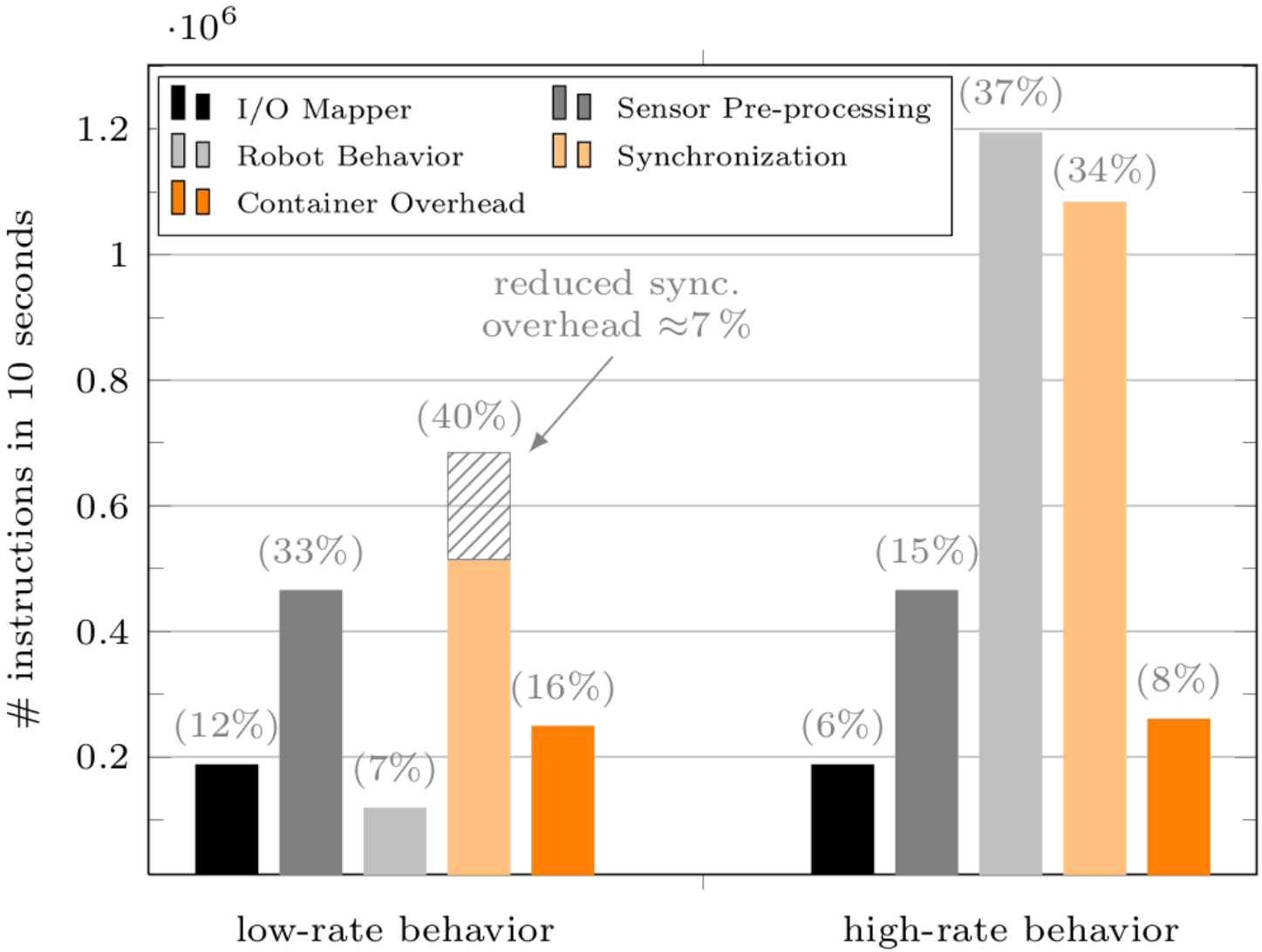


10x rate change

→ 10x processing load

...

Performance Measurements (1)



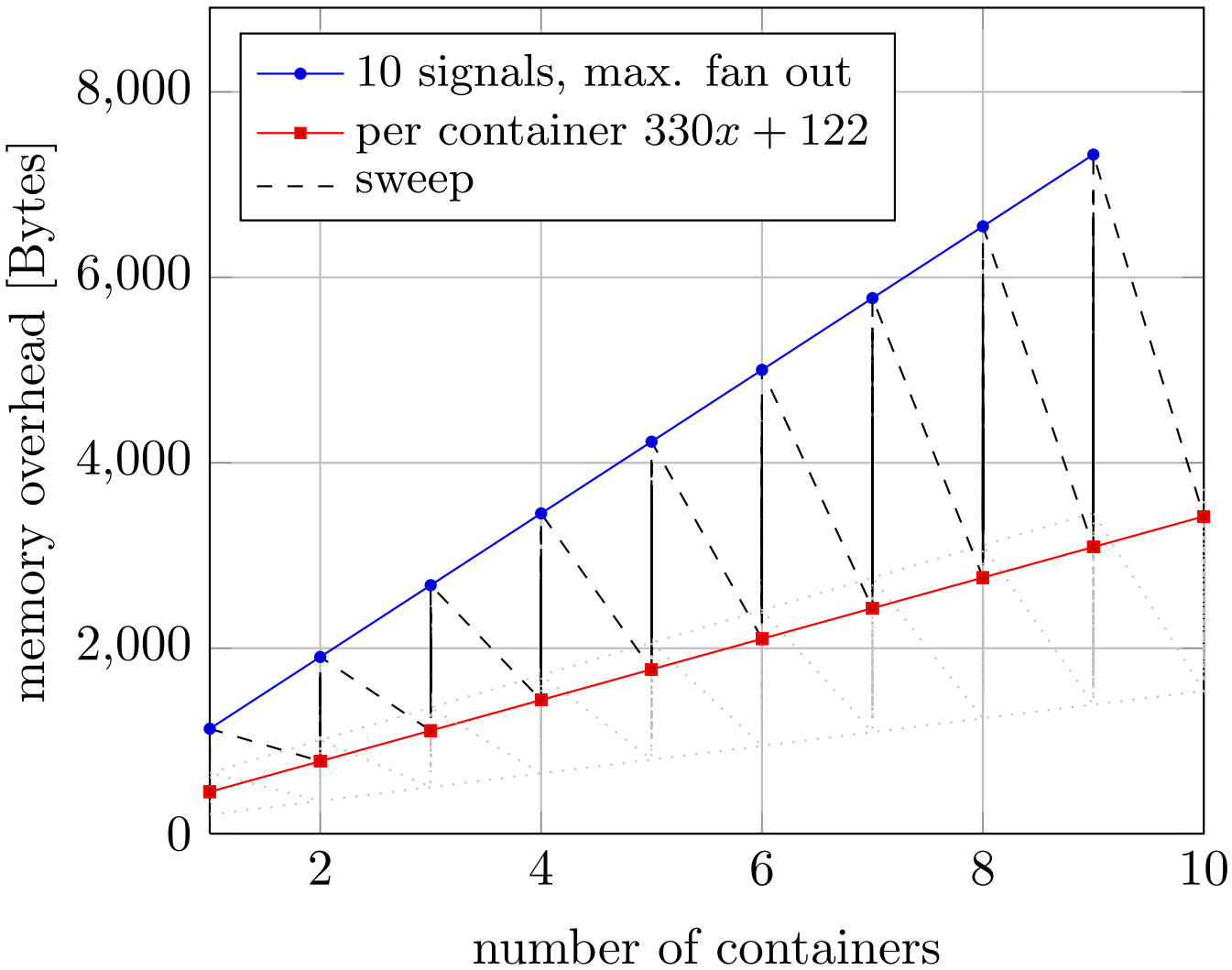
10x rate change

→ 10x processing load

→ same timing overhead

(relative overhead changes, of course)

Performance Measurements (2)



memory demand: linear with num. containers and num. signals

(All plots show *unoptimized* code)

Conclusions

- enable component-based systems with focus on components development instead of integration
- works with black boxes
- execution platform abstracted away
- formal verification of implementation semantics $\Rightarrow$ amenable for safety-critical systems
- developed a C library as reference implementation
  - is easy to use
  - does all the component integration for you
  - ensures consistency and correctness
  - can be ported to other languages, e.g., VHDL

Future Work

- Support multi-threading/multi-core ✓
- Reduce minimum delay ...ongoing
- Imposing interface *requirements* (e.g., max. delay) ...ongoing
- Synthesis to Hardware/VHDL...planned

The End

Thank you

Source code available at
`https://github.com/mbeckersys/libcontainers`