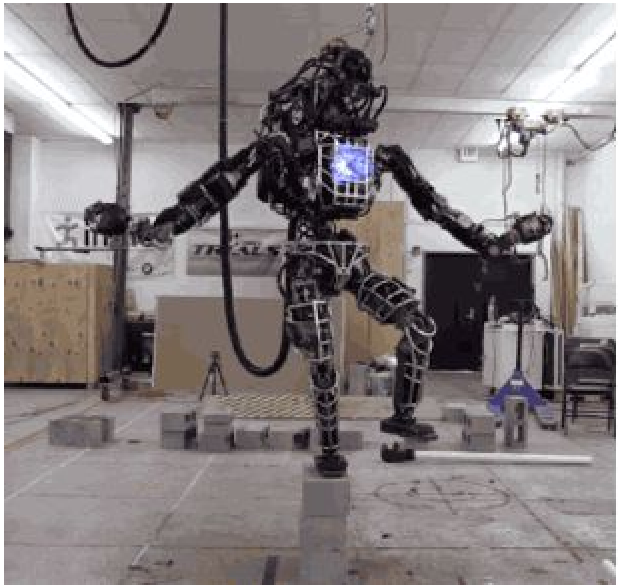
Real-time control in ROS and ROS 2.0

Jackie Kay

jackie@osrfoundation.org

Adolfo Rodriguez Tsouroukdissian

adolfo.rodriguez@pal-robotics.com

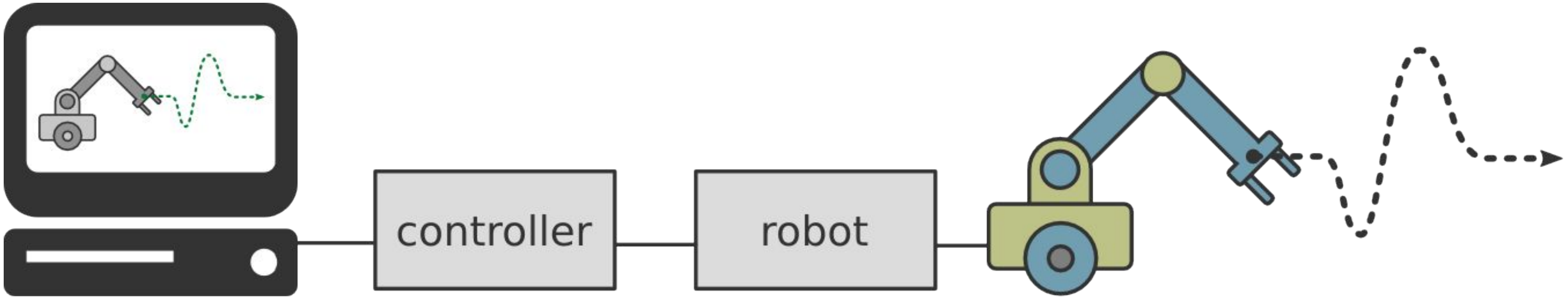


PAL ROBOTICS

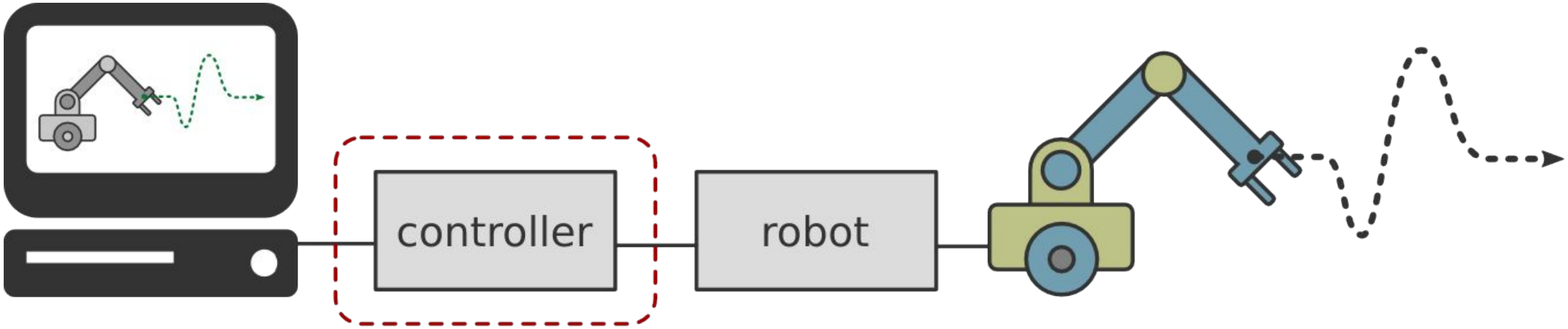
Open Source Robotics Foundation

Table of Contents

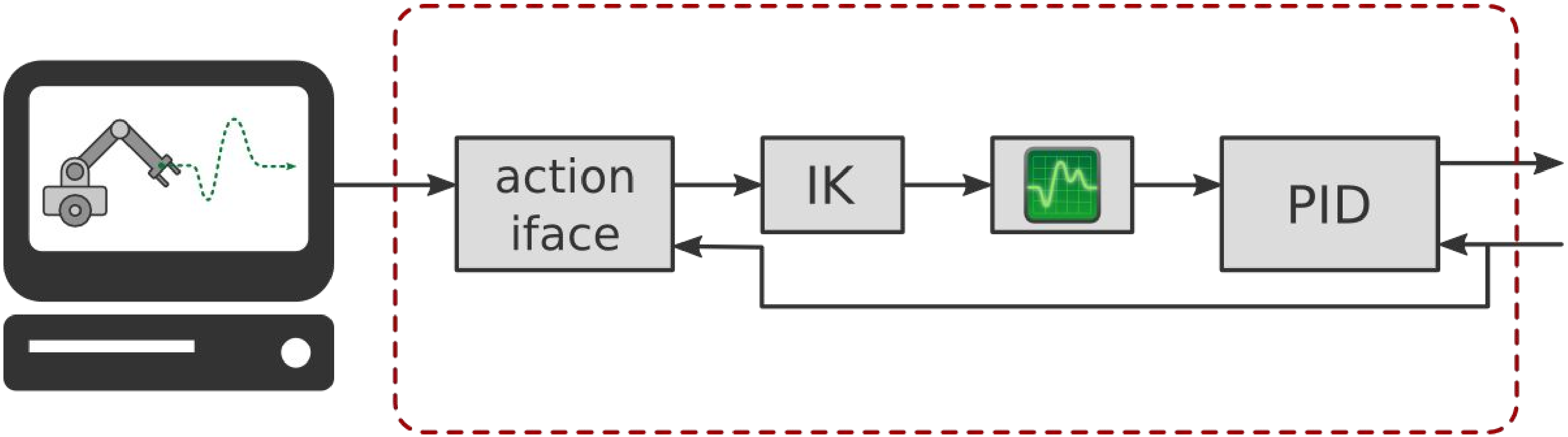
A motivating example



A motivating example

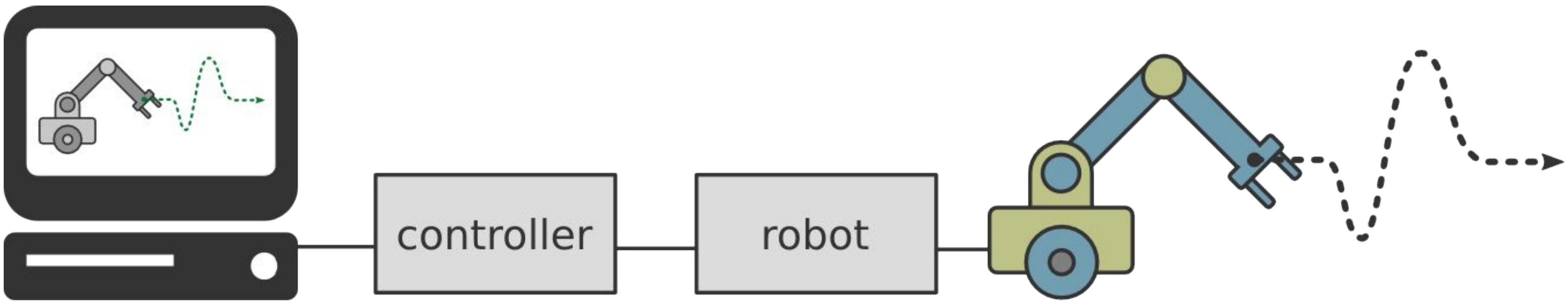


A motivating example



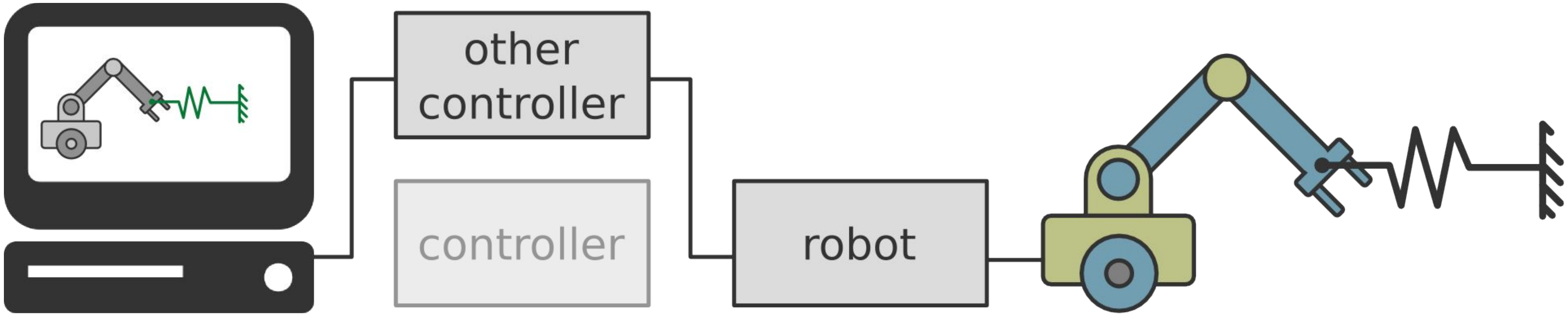
- Blocks can be **composed** by other blocks
- Some blocks are subject to **real-time constraints**

A motivating example



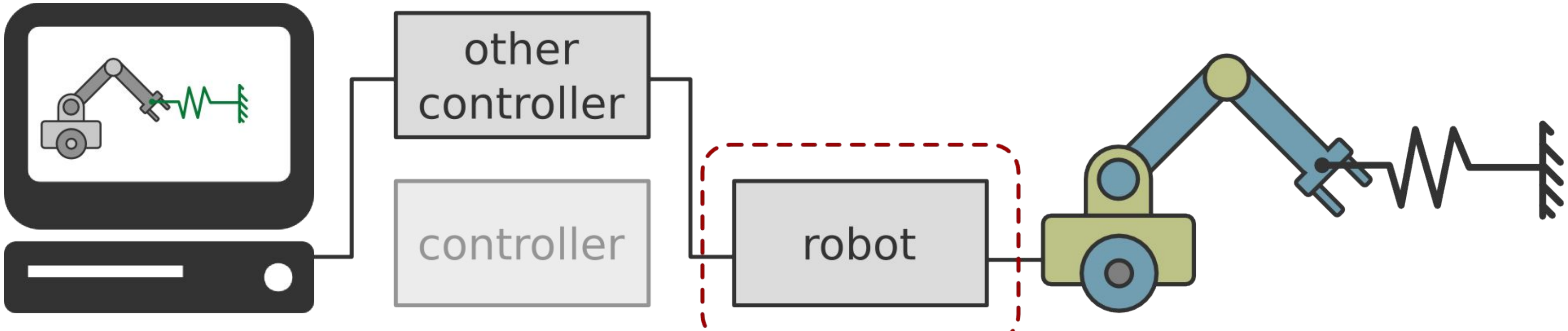
- Blocks can be **composed** by other blocks
- Some blocks are subject to **real-time constraints**

A motivating example



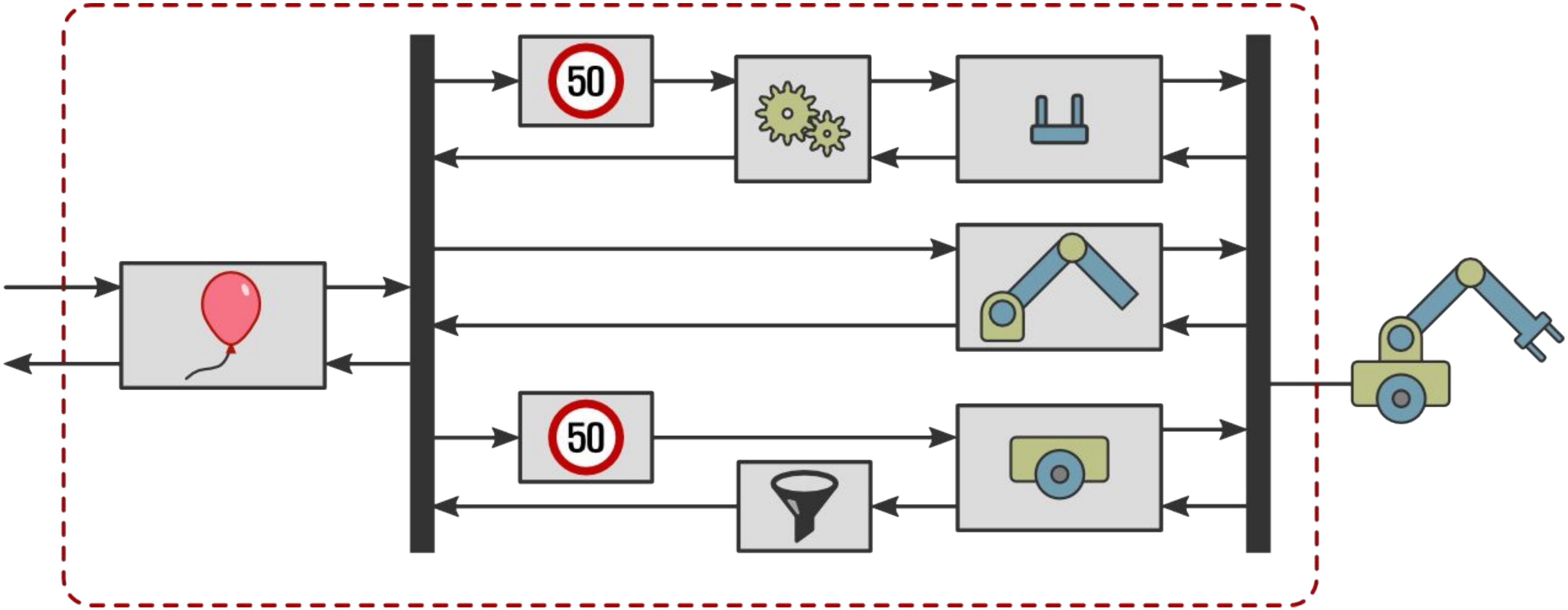
- Blocks can be **composed** by other blocks
- Some blocks are subject to **real-time constraints**
- System **topology** can **change at runtime**

A motivating example



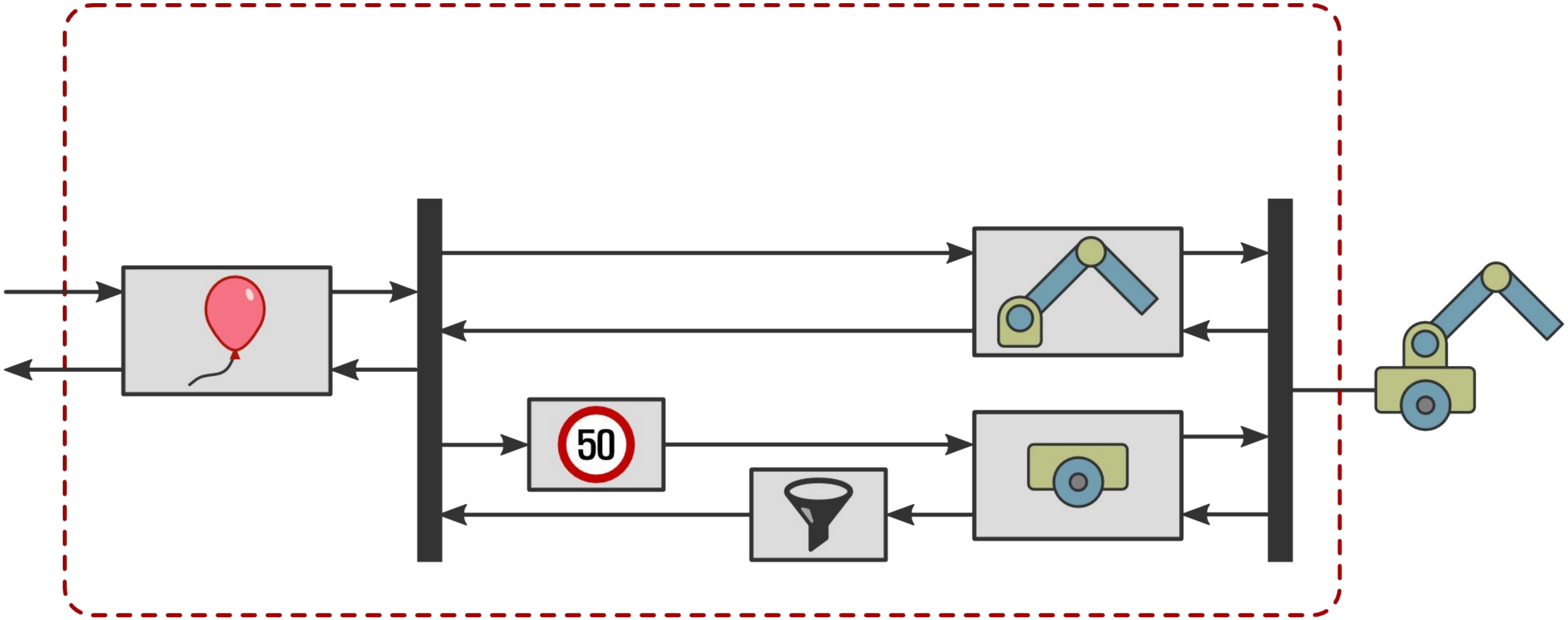
- Blocks can be **composed** by other blocks
- Some blocks are subject to **real-time constraints**
- System **topology** can **change at runtime**

A motivating example



- Blocks can be **composed** by other blocks
- Some blocks are subject to **real-time constraints**
- System **topology** can **change at runtime**

A motivating example

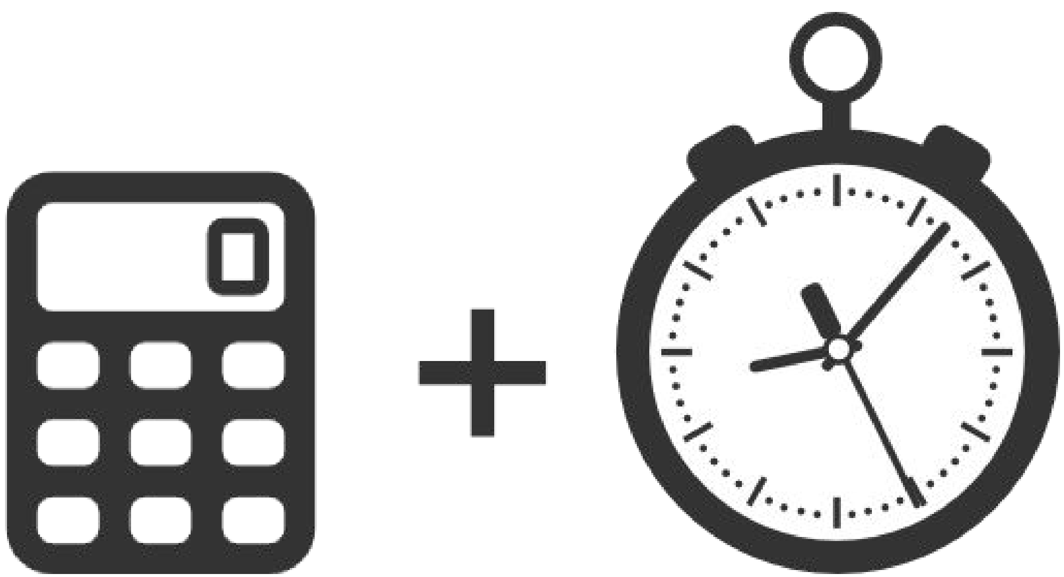


- Blocks can be **composed** by other blocks
- Some blocks are subject to **real-time constraints**
- System **topology** can **change at runtime**

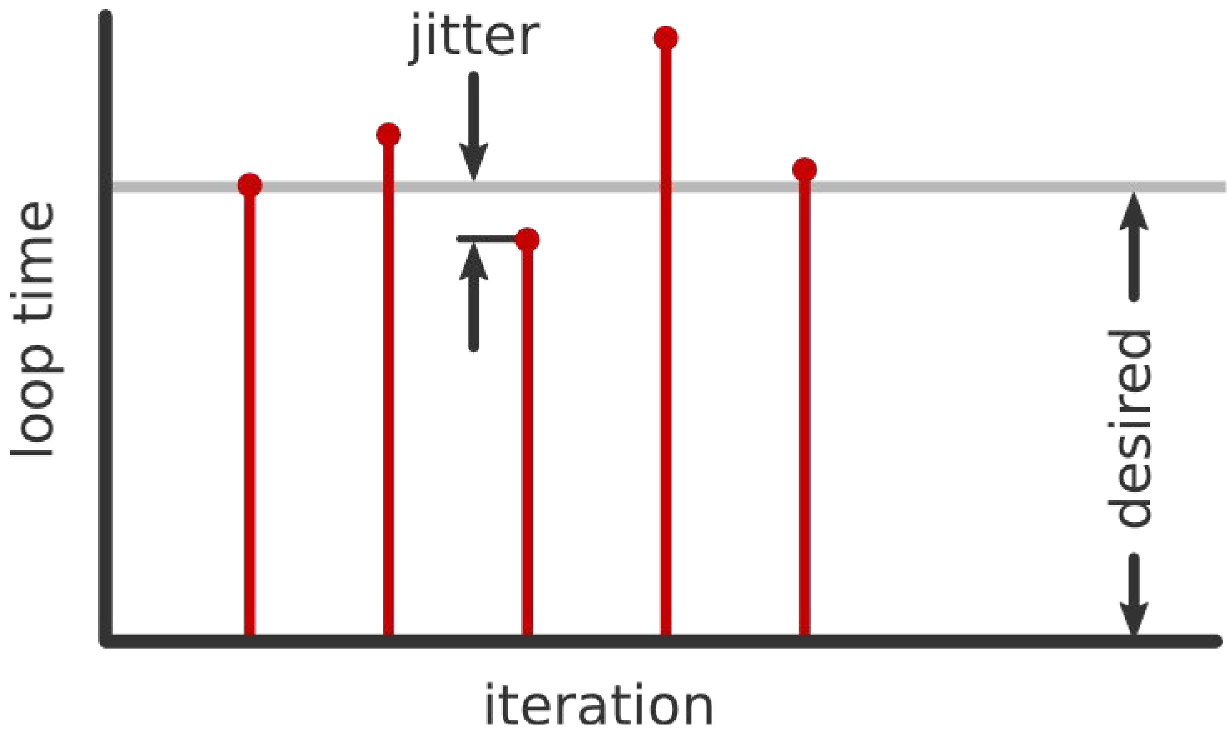
Table of Contents

Real-time computing

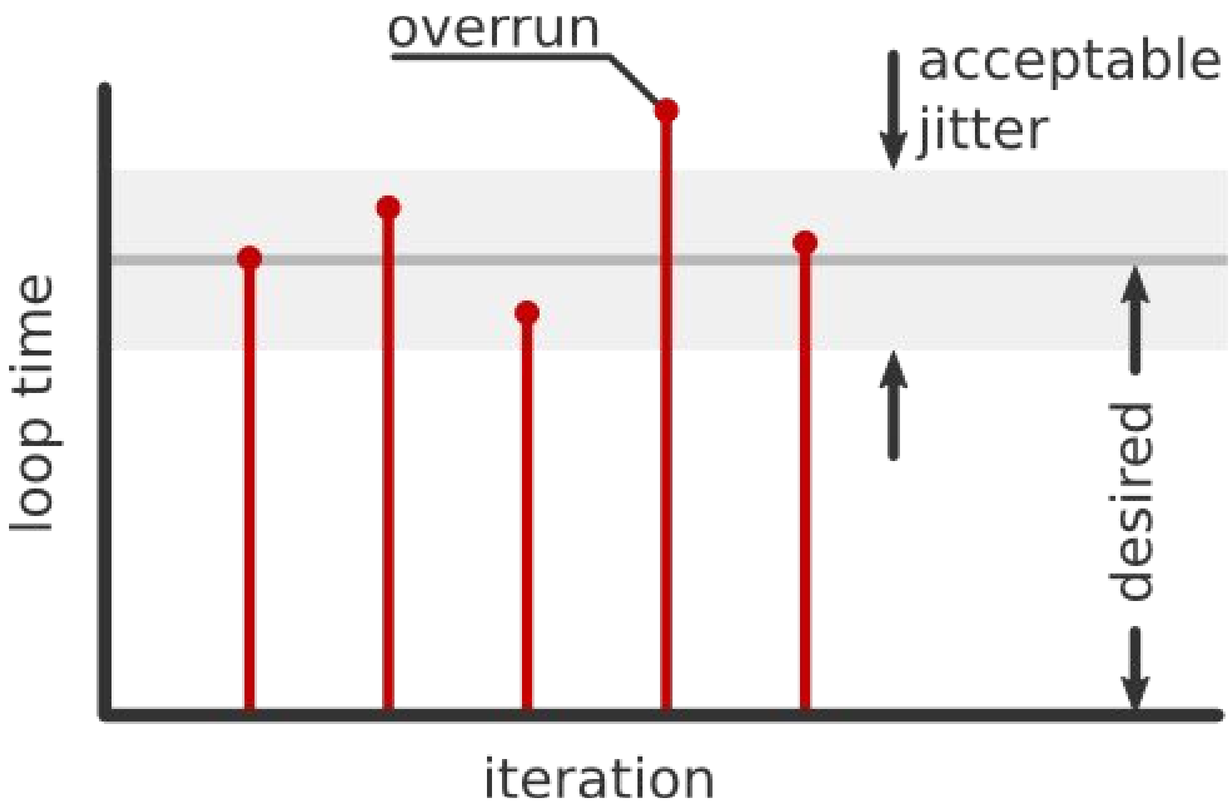
- It's about **determinism**, not **performance**
- **Correct computation** delivered at the **correct time**
- **Failure to respond** is as bad as a **wrong response**



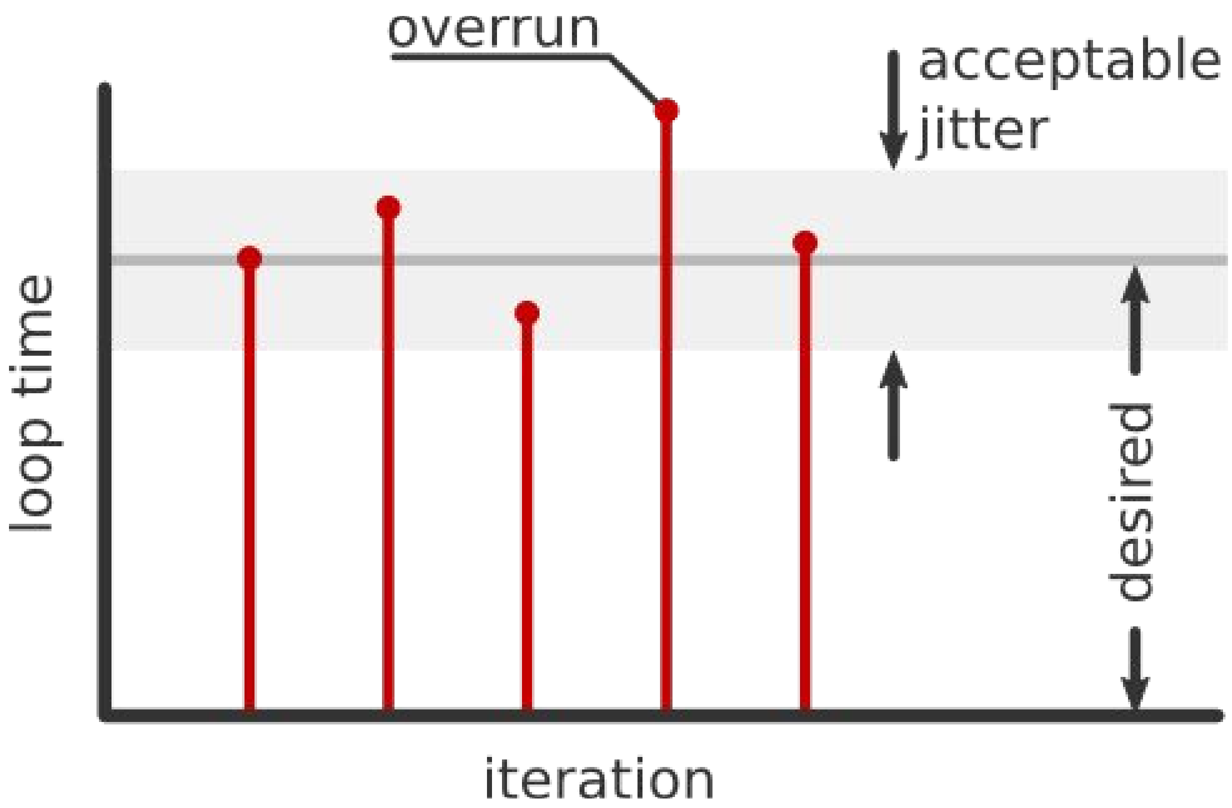
Real-time computing



Real-time computing



Real-time computing

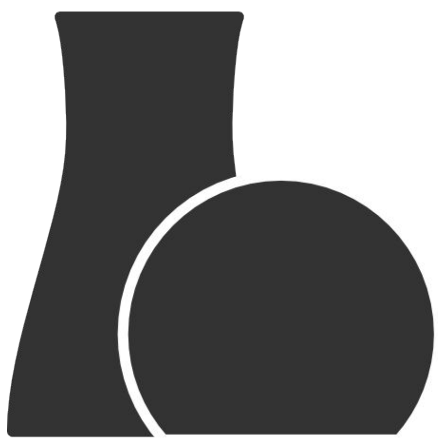


Usefulness of results after **missing a deadline?**

Real-time computing

Hard real-time systems

- Missing a deadline is considered a **system failure**
  Overruns may lead to loss of life or financial damage
- **Safety-** or **mission-critical** systems
  Examples: reactor, aircraft and spacecraft control




Real-time computing

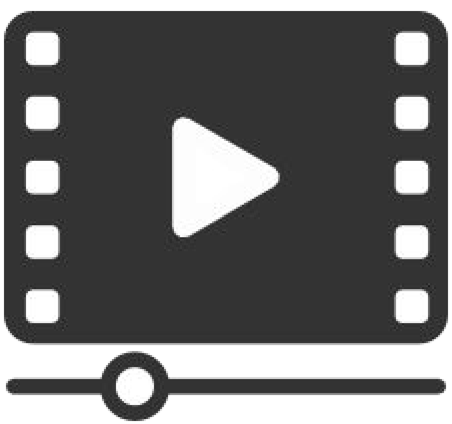
Soft real-time systems

- Missing a deadline has a cost, but is **not catastrophic**

  Result becomes **less useful** after deadline

- Often related to **Quality of Service**

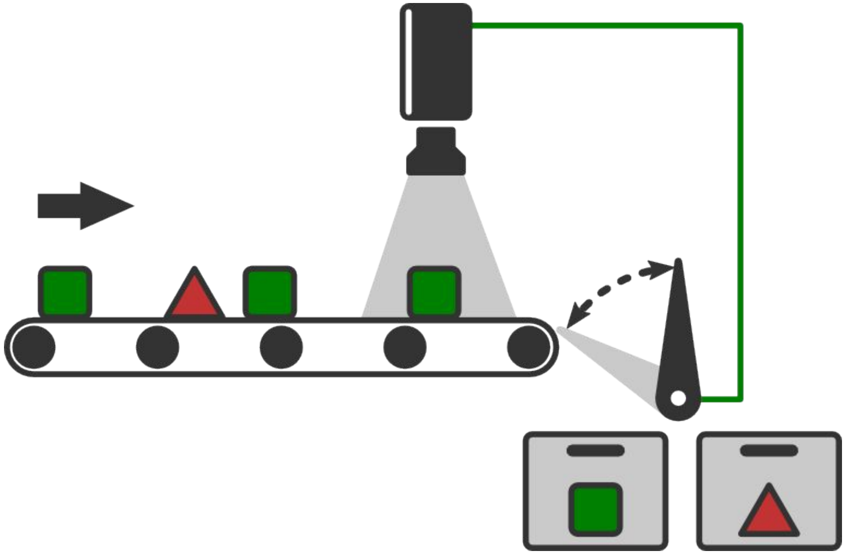
  Examples: audio / video streaming and playback



Real-time computing

Firm real-time systems

- Missing a deadline has a cost, but is **not catastrophic**
  Result becomes **useless** after deadline
- Cost might be interpreted as **loss of revenue**
  Examples: Financial forecasting, robot assembly lines



Real-time computing

Why do we care?

- **Event response**
  e.g. parts inspection
- **Closed-loop control**
  e.g. manipulator control
- **Added benefit:** Reliability, extended uptime
  Downtime is unacceptable or too expensive

The above is prevalent in **robotics software**

Goal of ROS 2

Real-time compatibility, from day one

Table of Contents

Requirements and best practices

Use an OS able to deliver the required determinism

- **Linux variants**

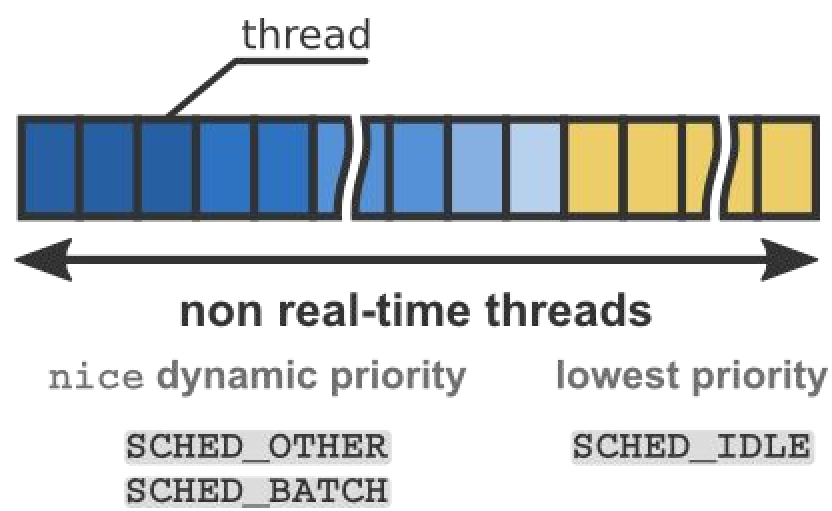
OS	real-time	max latency (µs)
Linux	no	10^4
RT PREEMPT	soft	10^1-10^2
Xenomai	hard	10^1

- **Proprietary:** e.g. QNX, VxWorks
  POSIX compliant, **certified** to IEC 61508 SIL3 et.al.

Requirements and best practices

Prioritize real-time threads

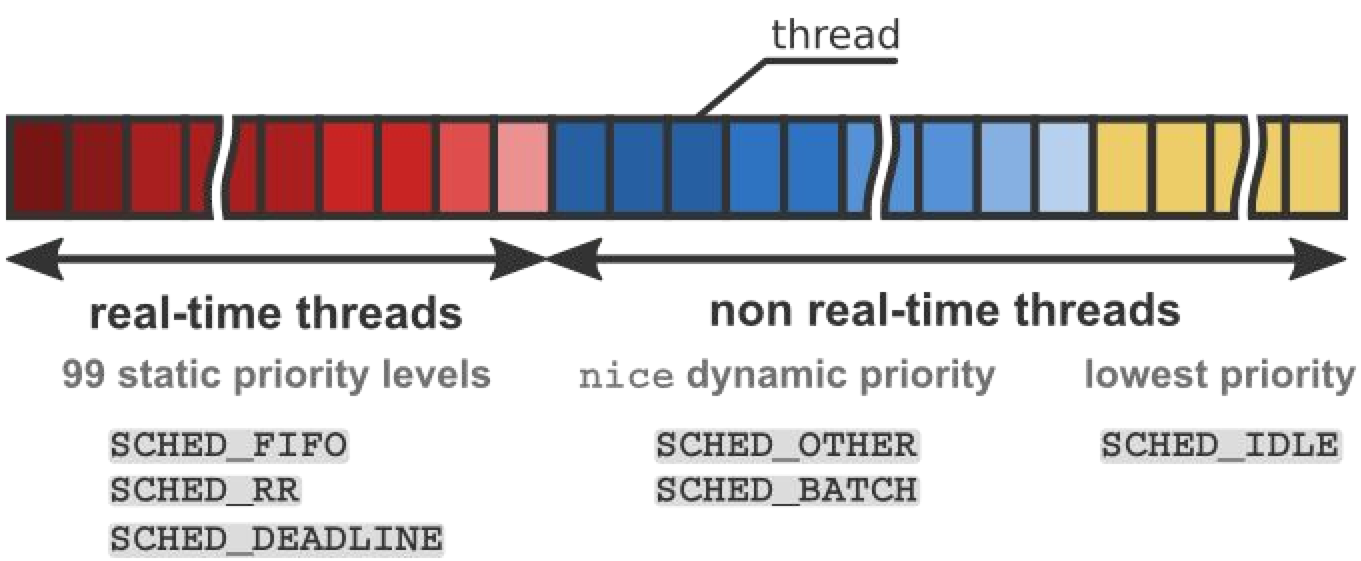
– Use a **real-time** scheduling policy



Requirements and best practices

Prioritize real-time threads

– Use a **real-time** scheduling policy



Requirements and best practices

Avoid sources of non-determinism in real-time code

- Memory allocation and management (`malloc`, `new`)
  Pre-allocate resources in the non real-time path
  Real-time safe *O(1)* allocators exist
- Blocking synchronization primitives (e.g. **mutex**)
  Real-time safe alternatives exist (e.g. lock-free)
- Printing, logging (`printf`, `cout`)
  Real-time safe alternatives exist

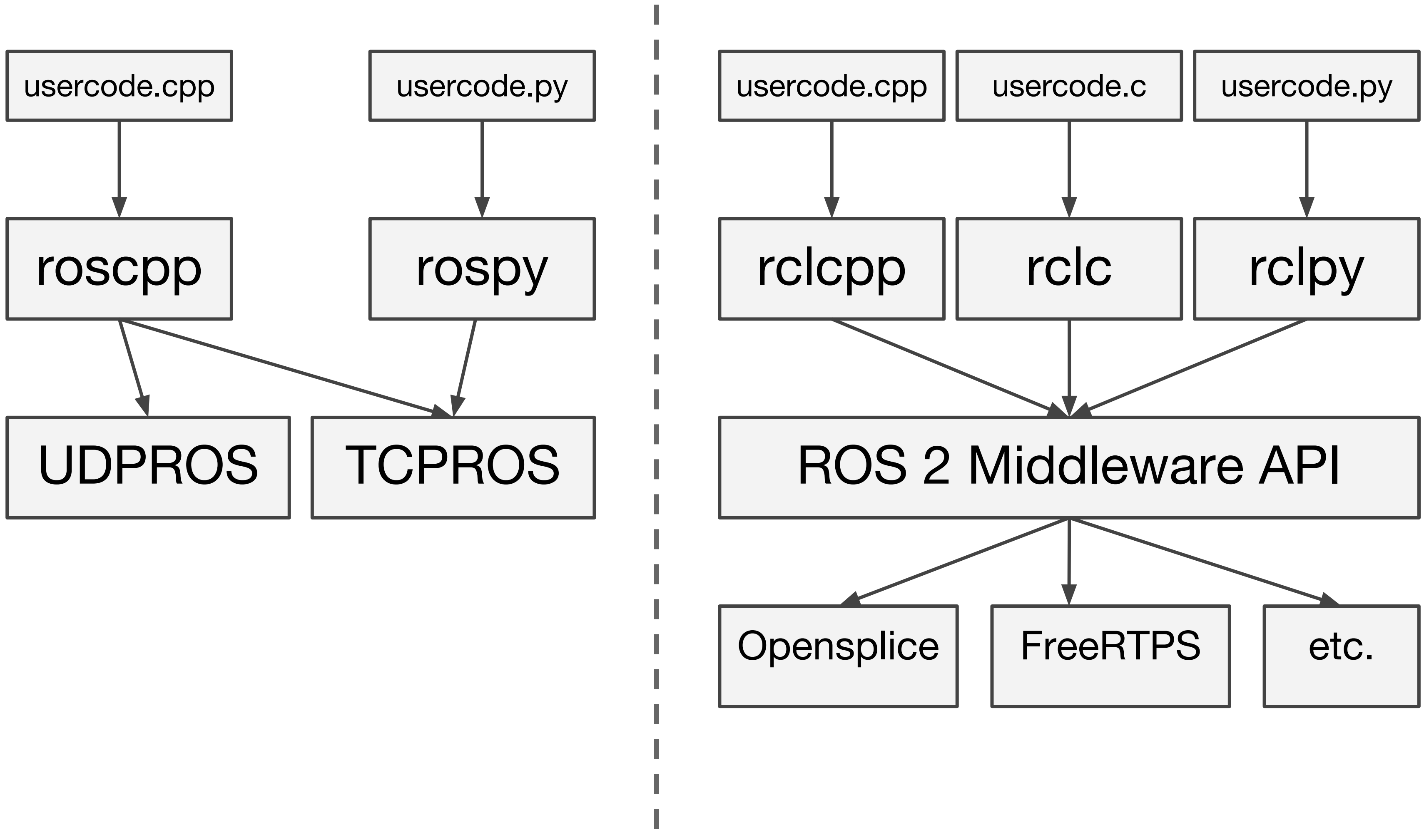
Requirements and best practices

Avoid sources of non-determinism in real-time code

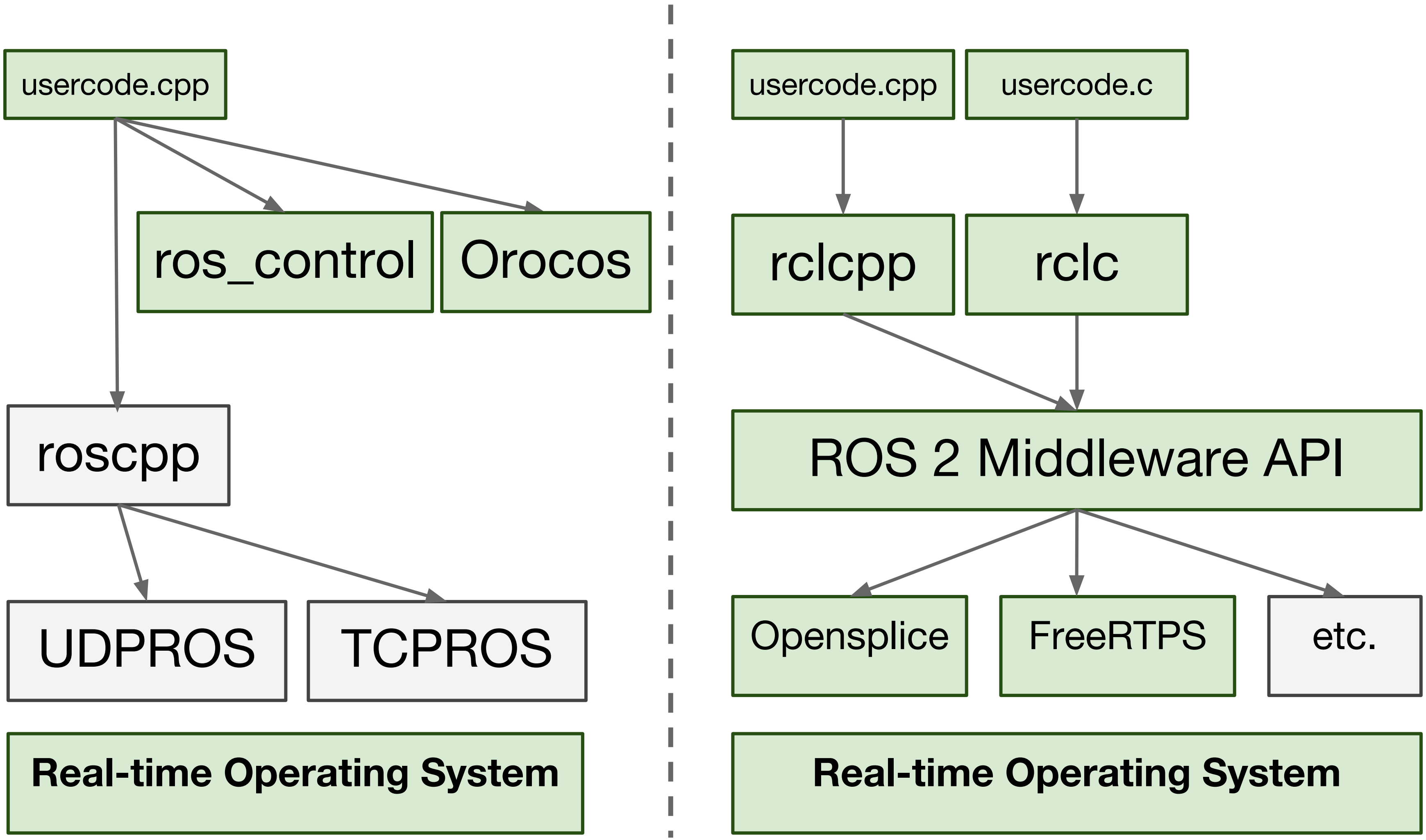
- Network access, especially TCP/IP
  RTnet stack, real-time friendly protocols like RTPS
- Non real-time device drivers
  Real-time drivers exist for some devices
- Accessing the hard disk
- Page faults
  Lock address space (`mlockall`), pre-fault stack

Table of Contents

ROS2 design - architecture comparison



ROS2 design - real-time architecture



ROS2 design – Modularity

- **ROS2 allows customization for real-time use-cases**
  - Memory management
  - Synchronization
  - Scheduling

  are **orthogonal** to each other, and to node topology

ROS 2 - current implementation

Executor

initialization — non real-time

preallocate memory

...

spin — real-time

```
rmw_wait(timeout)
{
    pass conditions to waitset
    wait (in DDS)
    wake-up if timed-out
}
do work if it came in
```

loop until interrupted

cleanup — non real-time

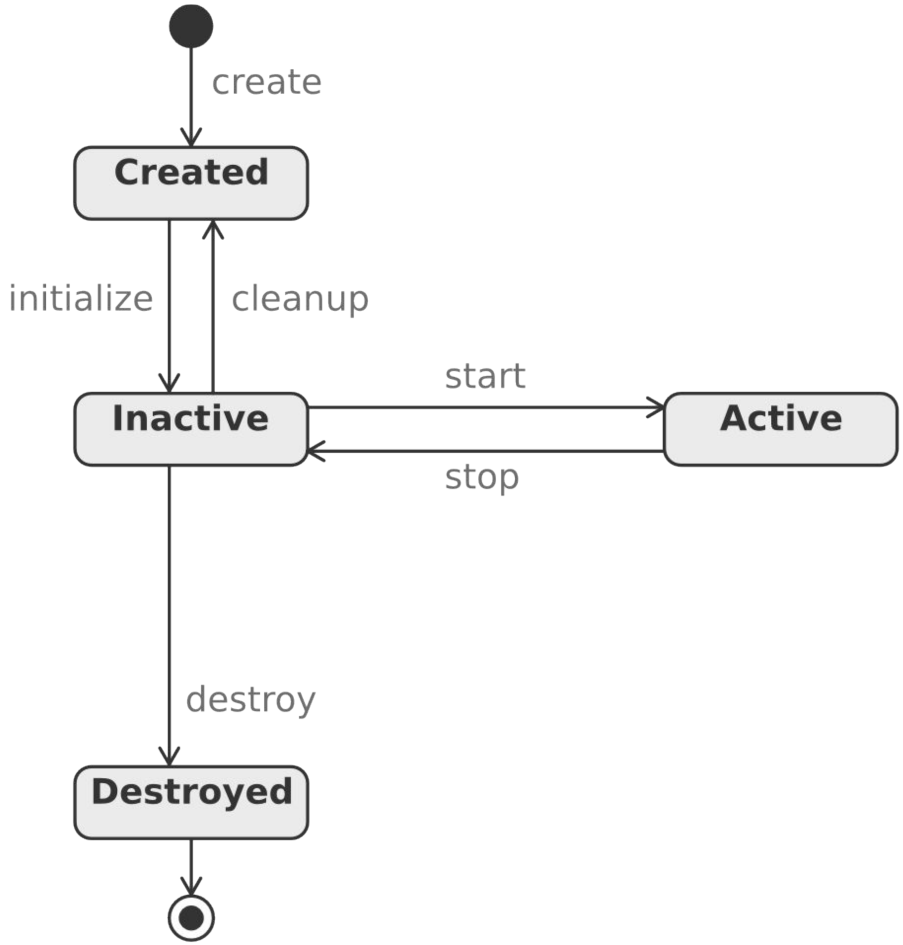
deallocate memory

...

ROS2 design – Node lifecycle

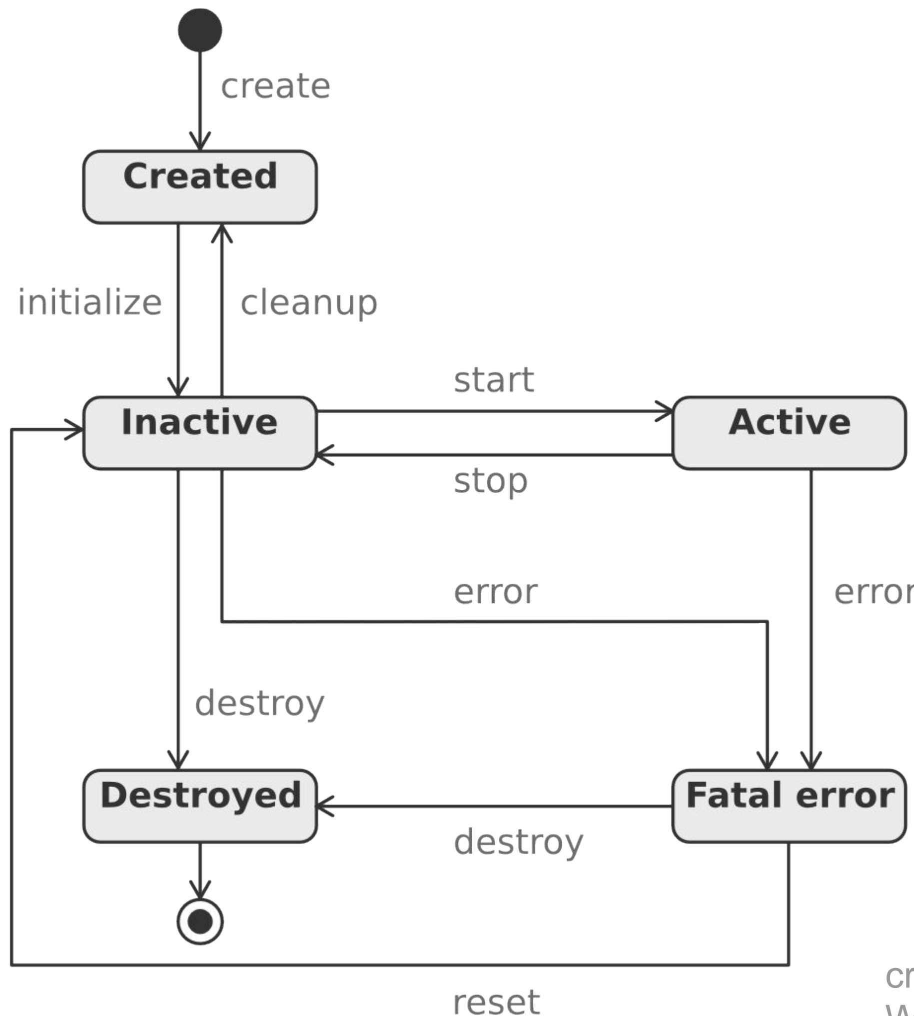
- **Standard node lifecycle state machine**
  - Opt-in feature
  - Node lifecycle can be managed without knowledge of internals (black box)
- **Best practice from existing frameworks**
  - microblx
  - OpenRTM
  - Orocos RTT
  - ros_control

ROS2 design – Node lifecycle



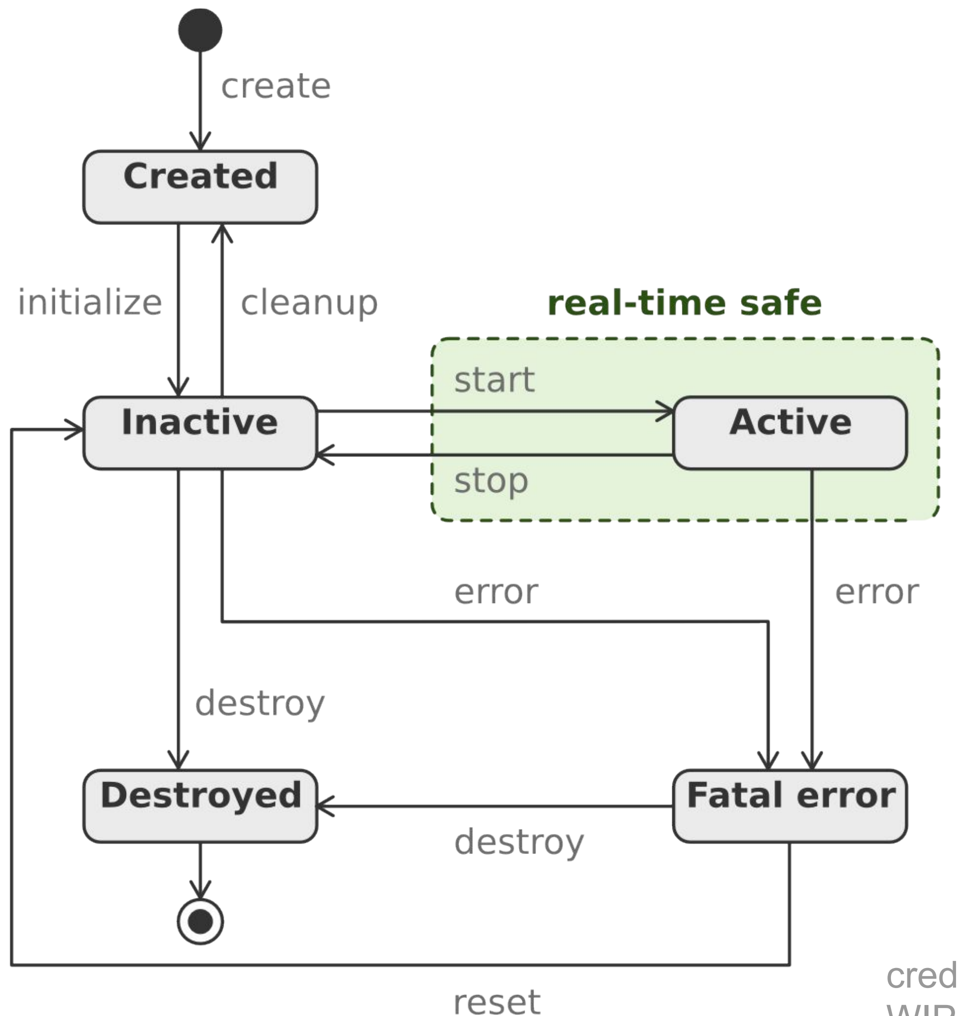
credit: Geoffrey Biggs et.al.
WIP, design subject to change

ROS2 design – Node lifecycle



credit: Geoffrey Biggs et.al.
WIP, design subject to change

ROS2 design – Node lifecycle



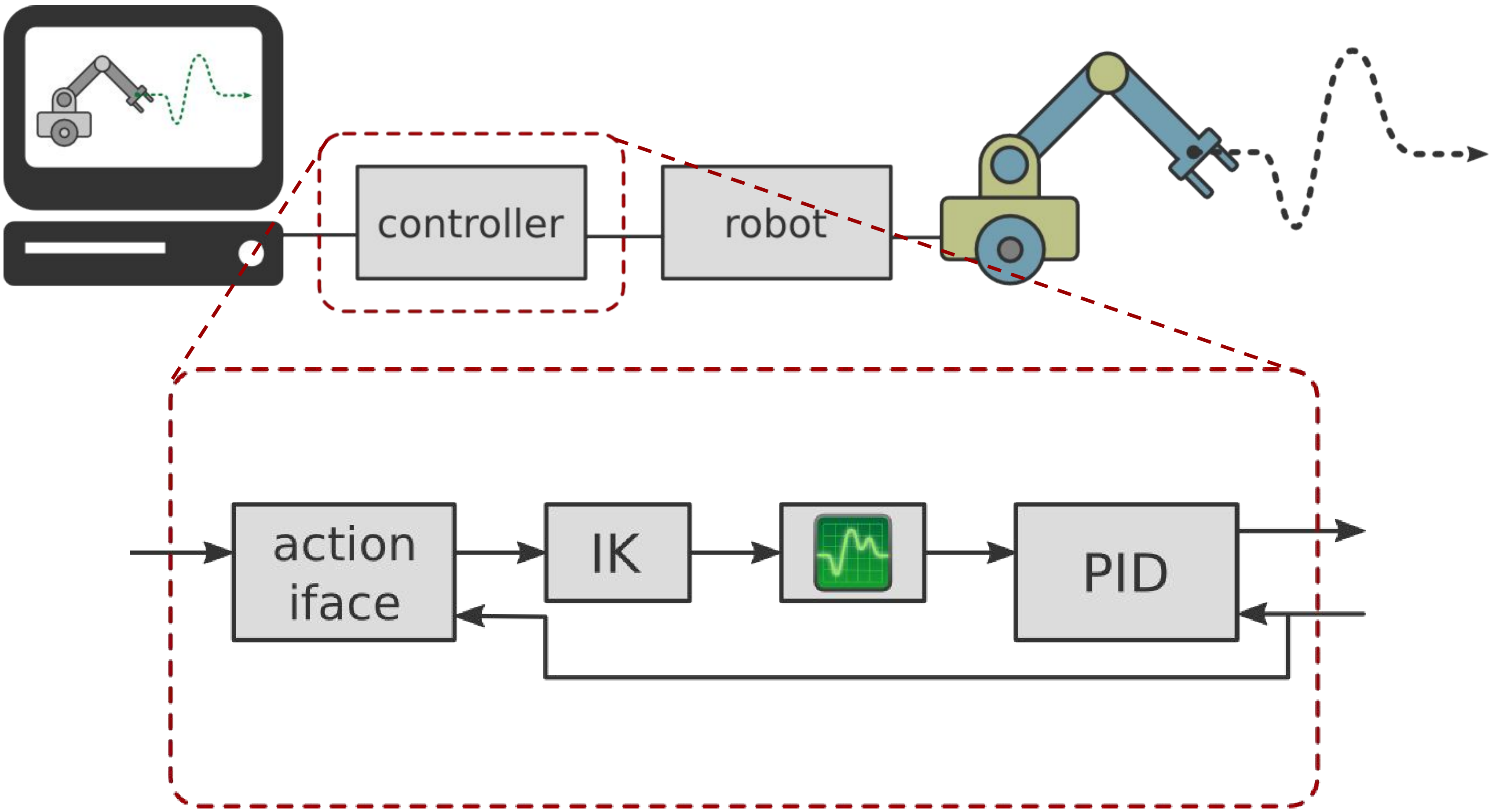
credit: Geoffrey Biggs et.al.
WIP, design subject to change

ROS2 design – Node lifecycle

Benefits of managed lifecycle

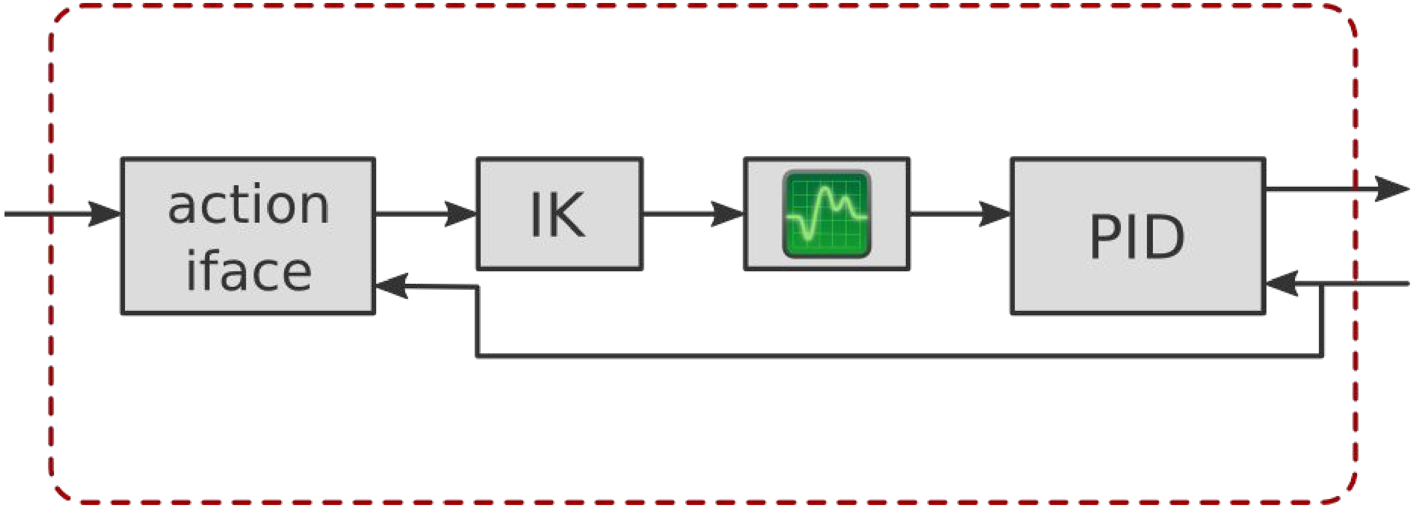
- **Clear separation of real-time code path**
- **Greater control of ROS network**
  - Help ensure correct launch sequence
  - Online node restart / replace
- **Better monitoring and supervision**
  - Standard lifecycle → standard tooling

ROS2 design – Node composition



ROS2 design – Node composition

- Composite node is a **black box** with well-defined API
- Lifecycle can be **stepped in sync** for all internal nodes
- **Resources** can be **shared** for internal nodes



ROS2 design – Communications

- **Inter-process**
  DDS can deliver soft real-time comms
  Customizable QoS, can be tuned for real-time use-case

- **Intra-process**
  Efficient (zero-copy) shared pointer transport

- **Same-thread**
  No need for synchronization primitives. Simple, fast

ROS 2 – alpha release

- Real-time safety is **configurable**
- Can configure custom allocation policy that **preallocates resources**
- Requires **hard limit** on number of pubs, subs, services
- Requires messages to be **statically sized**

ROS2 – progress overview

In progress

- Component **lifecycle**
- **Composable** components
- Complete **intra-process pipeline**

Future work

- Pre-allocate **dynamic messages**
- CI for verifying real-time **constraints**
- **Lock-free** multi-threaded executor

Table of Contents

Comparison with ROS1 + ros_control

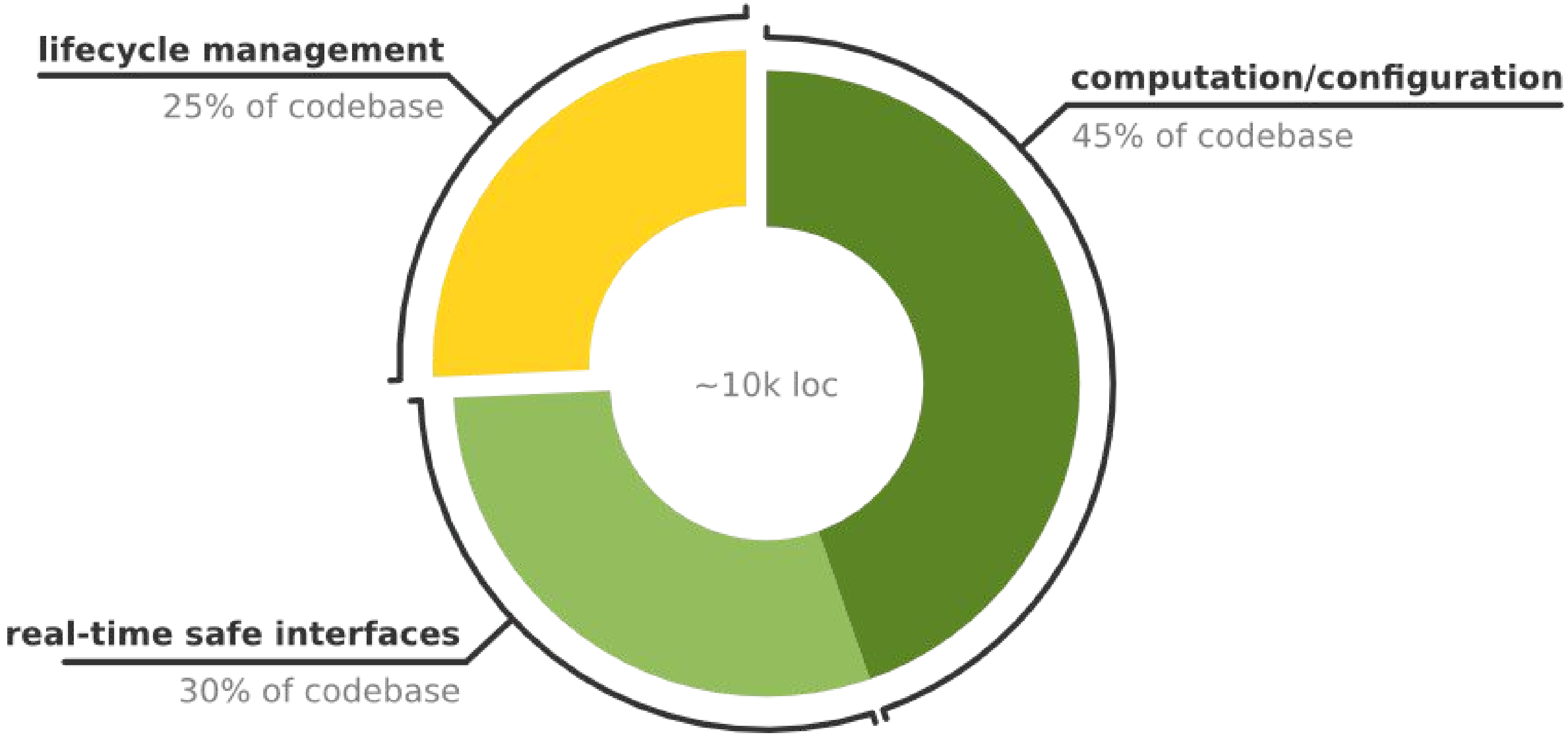
- Real-time safe communications
- Lifecycle management
- Composability

Comparison with ROS1 + ros_control

- **Real-time safe communications**
- **Lifecycle management**
- Composability

Comparison with ROS1 + ros_control

ROS1 + ros_control:

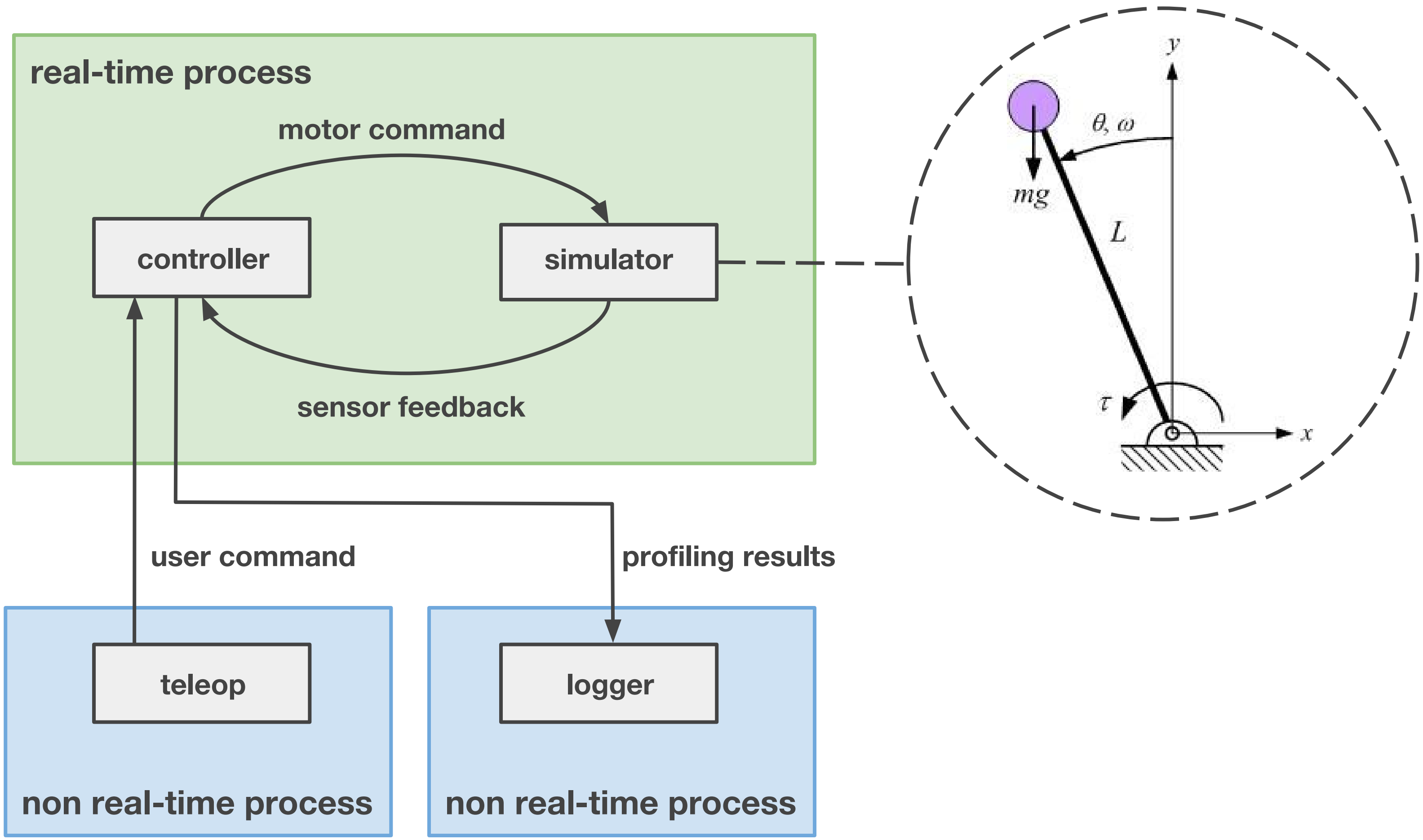


ROS2 equivalent:

- drop non-standard lifecycle / interfaces → gentler learning curve
- smaller codebase → easier to maintain

Table of Contents

ROS 2 Real-time Benchmarking: Setup



ROS 2 Real-time Benchmarking: Setup

Configuration

- **`RT_PREEMPT`** kernel
- Round robin scheduler (**`SCHED_RR`**), thread priority: 98
- **`malloc_hook`**: control malloc calls
- **`getrusage`**: count pagefaults

Goal

- **1 kHz** update loop (1 ms period)
- Less than **3%** jitter (30 μs)

Code

- **ros2/demos** - pendulum_control

ROS 2 Real-time Benchmarking: Memory

Zero runtime allocations

```
static void * testing_malloc(size_t size, const void * caller) {
  if (running) {
    throw std::runtime_error("Called malloc from real-time context!" );
  }
  // ... allocate and return pointer...
}
```

Zero major pagefaults during runtime

- Some minor pagefaults on the first iteration of the loop, none after
- Conclusion: all required pages allocated before execution starts

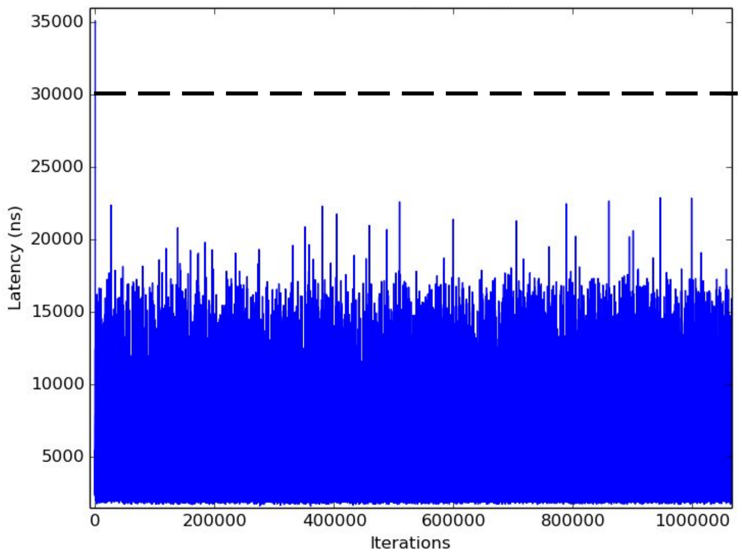
ROS 2 Real-time Benchmarking: Results

No stress

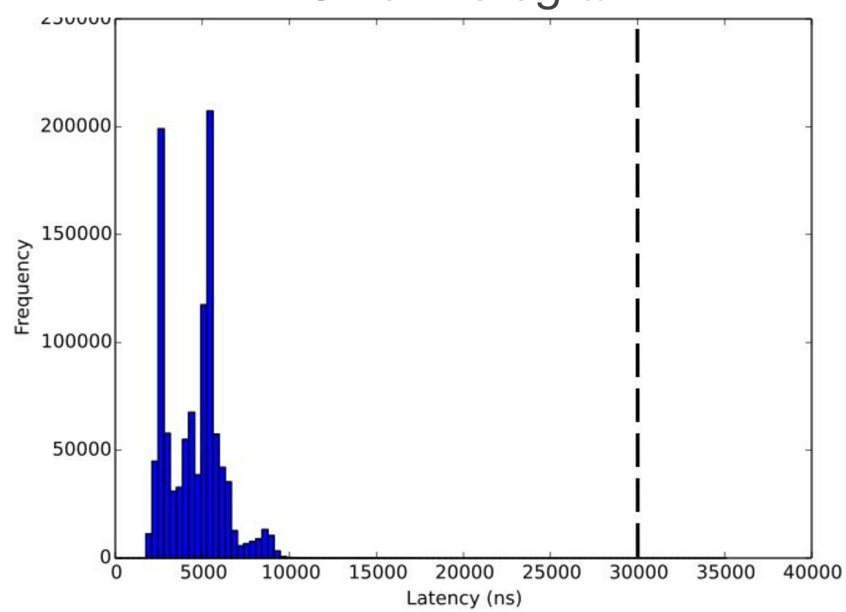
1,070,650 cycles observed

	Latency (ns)	% of update rate
Min	1620	0.16%
Max	35094	3.51%
Mean	4567	0.46%

Timeseries



Jitter histogram



ROS 2 Real-time Benchmarking: Results

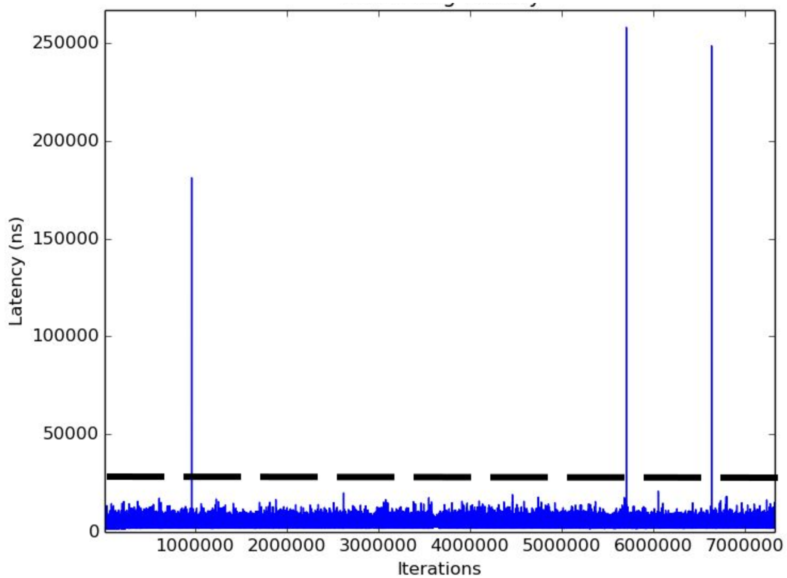
Stress applied:

`stress --cpu 2 --io 2`

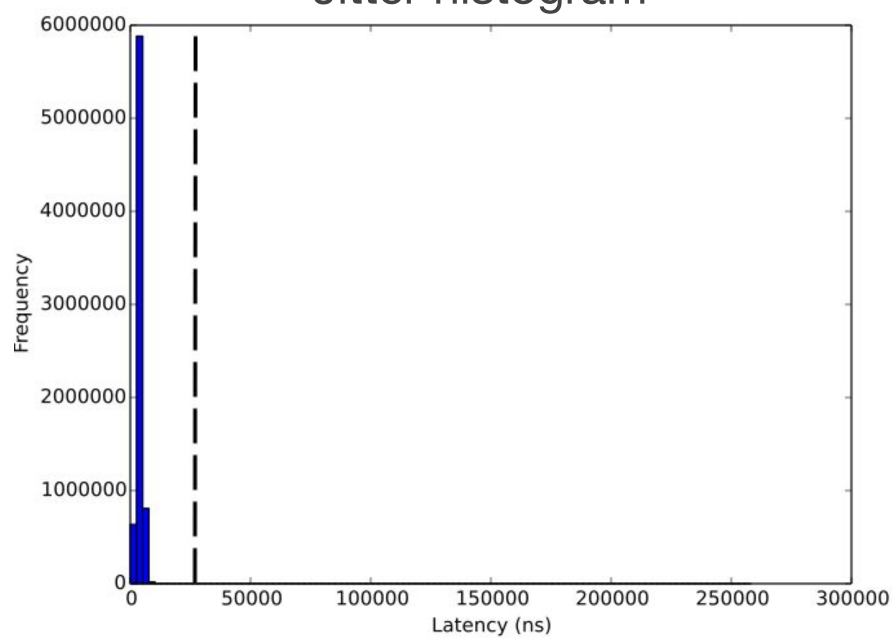
7,345,125 cycles observed

	Latency (ns)	% of update rate
Min	1398	0.14%
Max	258064	25.8%
Mean	3729.11	0.38%

3 instances of overrun observed



Jitter histogram



Closing remarks

- Systems subject to real-time constraints are very relevant in robotics
- ROS2 will allow user to implement such systems
  - with a proper RTOS, and carefully written user code
- Initial results based on ROS2 alpha are encouraging
  - inverted pendulum demo
- Design discussions and development are ongoing!
  - ROS SIG Next-Generation ROS
  - ros2 Github organization

Selected references

- **[Biggs, G.]** ROS2 design article on node lifecycle (under review)
- **[Bruyninckx, H.]** Real Time and Embedded Guide
- **[Kay, J.]** ROS2 design article on Real-time programming
- **[National Instruments]** What is a Real-Time Operating System (RTOS)?
- **[OMG]** OMG RTC Specification
- **[ROS Control]** ROS Control, an Overview
- **[RTT]** Orocos RTT component builder's manual
- **[RT PREEMPT]** Real-Time Linux Wiki
- **[Xenomai]** Xenomai knowledge base