Real-time Data-Flow Extensions for ROS2

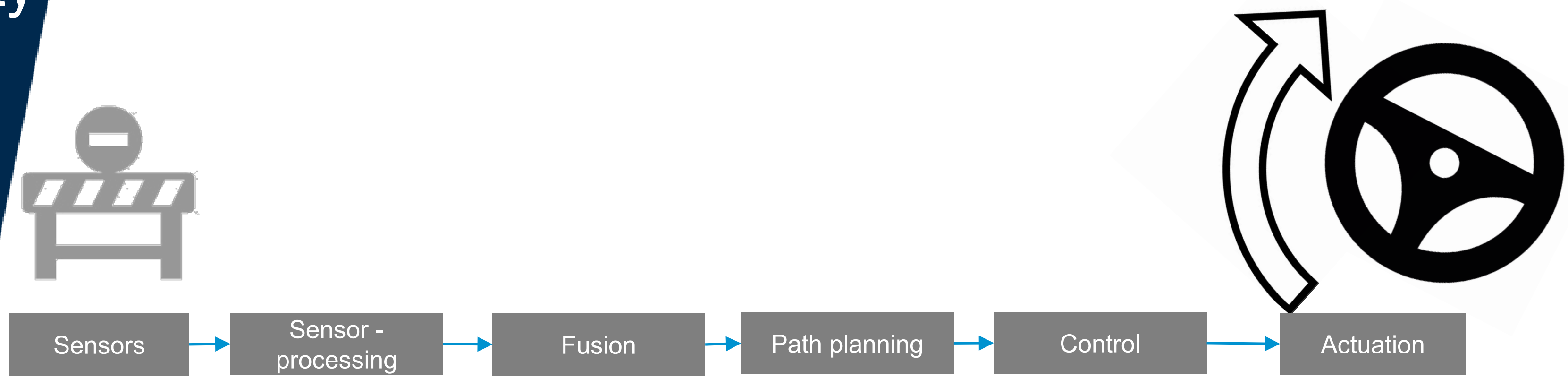
ROSCon 2023

Christopher Helpa

Autonomous Driving – Safety Critical Real-Time Systems

Requirements:

- Safety (SW & System)
- Strong separation of concern
- Many independent applications coexisting
- High level of offline analyzability
- Guaranteed deadlines for overall processing

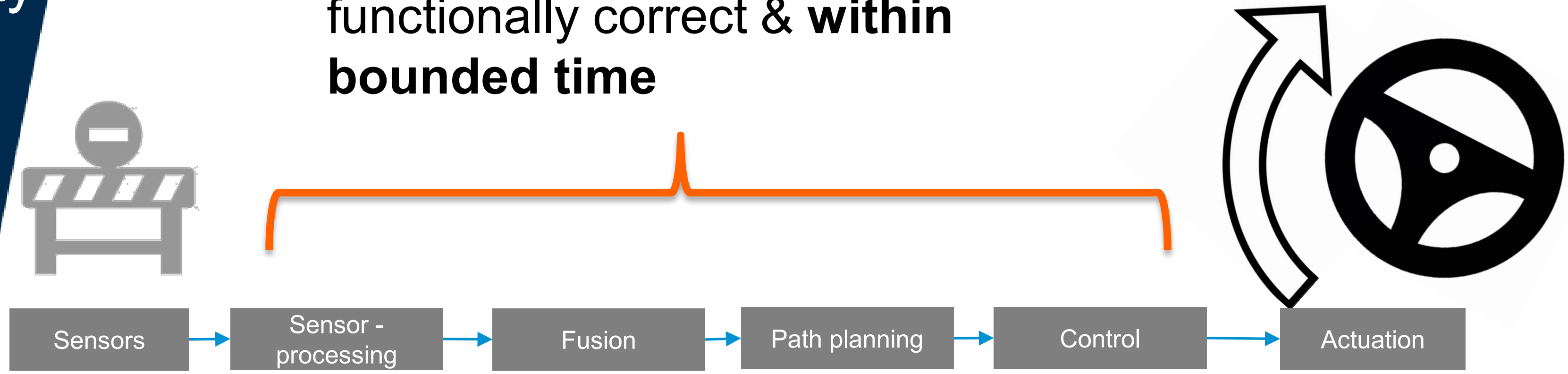


Autonomous Driving – Safety Critical Real-Time Systems

Requirements:

- Safety (SW & System)
- Strong separation of concern
- Many independent applications coexisting
- High level of offline analyzability
- Guaranteed deadlines for overall processing

Safety = guaranteed to be functionally correct & **within bounded time**

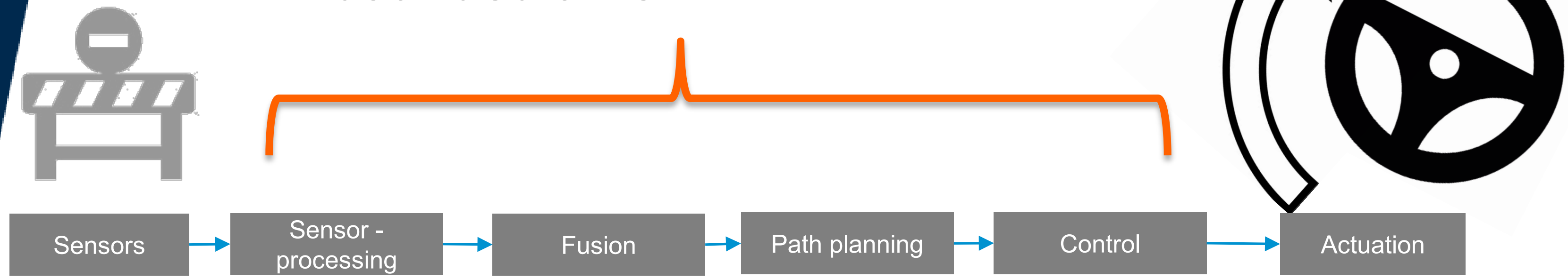


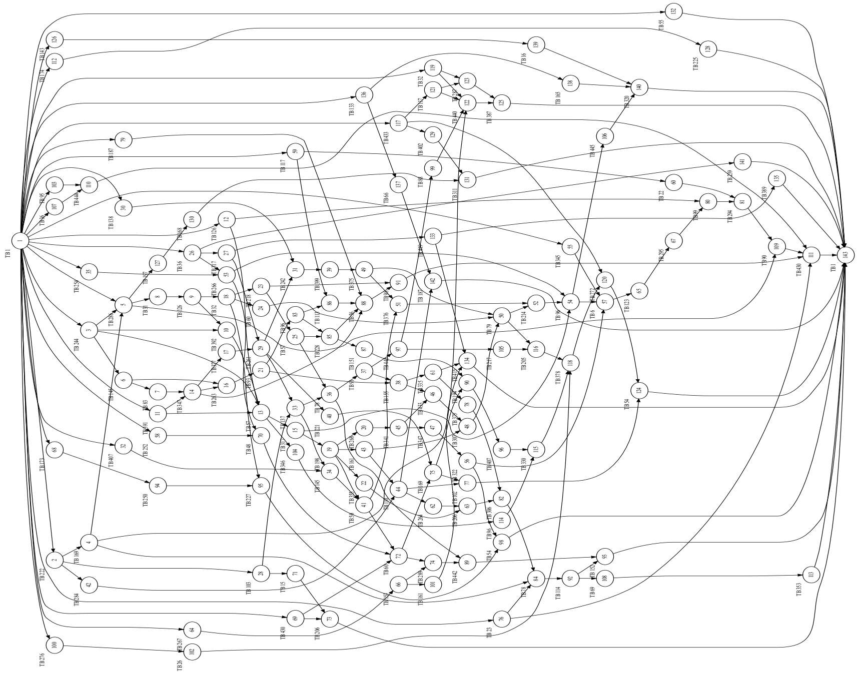
Autonomous Driving – Safety Critical Real-Time Systems

Requirements:

- Safety (SW & System)
- Strong separation of concern
- Many independent applications coexisting
- High level of offline analyzability
- Guaranteed deadlines for overall processing
- Timing requirements ~100ms
- Inherently periodic

Safety = guaranteed to be functionally correct & **within bounded time**





What we aim to achieve

- Defined and predictable execution behaviour
- More stable and predictable End-2-End Latency
- Data-flow deterministic execution across multiple processes
- Control when temporal behaviour does not match expectations

ROS2 weaknesses

Data driven leads to unnecessary activations

No n:1 communication

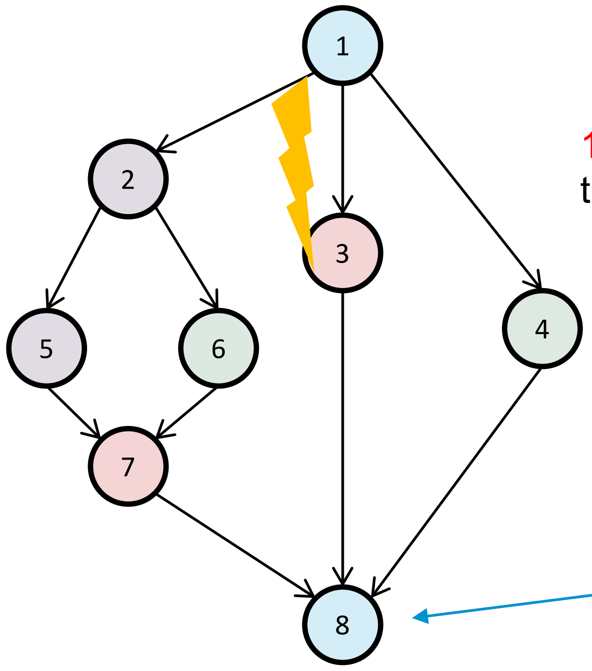
No notion of real-time progress

Workaround: Timers

Indeterministic system behaviour

Unsynchronized

	ROS2
Trigger Paradigm	Data-driven, time-driven
Communication Pattern	Bi-direction. Sync and async
Communication mechanism	Topics, services, actions
Participants	1:1, 1:n
Dispatching	**Implicit** when data is there



100ms : Unexpected high execution time prevents fulfilling deadline

N:1 not properly supported in ROS2

Data-flow ROS2

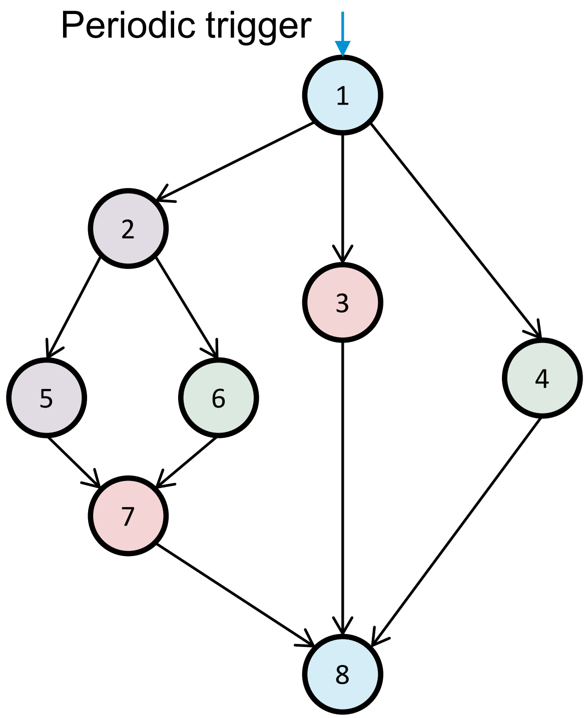
Treat system as directed acyclic graph (DAG)

Explicit dispatching decisions

Dispatch according to graph constraints

Inspired by Zettascale's Zenoh-Flow (*) and Berkeley's ERDOS (**)

	ROS2	Data-flow
Trigger Paradigm	Data-driven, time-driven	Data-flow driven
Communication Pattern	Bi-direction. Sync and async	Uni-directional, async
Communication mechanism	Topics, services, actions	topics
Participants	1:1, 1:n	1:1,1:n,n:1
Dispatching	**Implicit** when data is there	**Explicit** when predecessors finished



Trigger callbacks based on flow requirements

(*) https://github.com/eclipse-zenoh/zenoh-flow

(**) https://dl.acm.org/doi/10.1145/3492321.3519576

Explicit Data-flow Execution (1)

During development

Create mapping of callbacks and pub/sub topics

Init

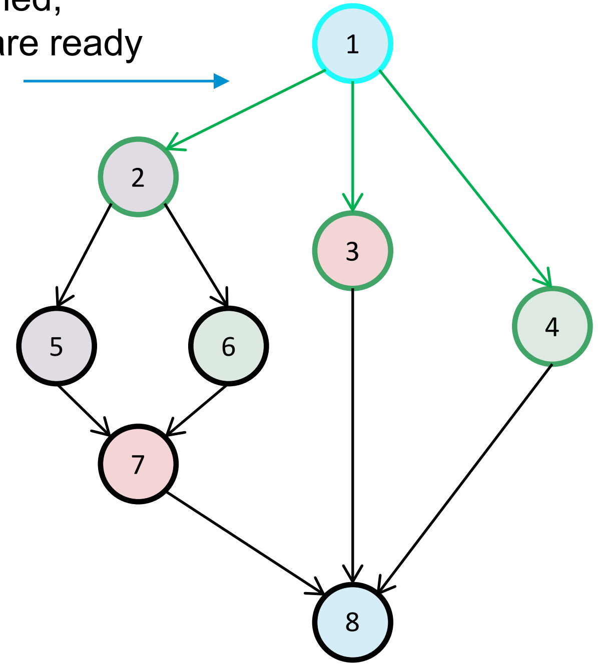
Build DAG of whole system

At runtime

Dispatch when **all** predecessors have finished

Fully data-flow deterministic

1 finished,
2,3,4 are ready



Explicit Data-flow Execution (2)

During development

Create mapping of callbacks and pub/sub topics

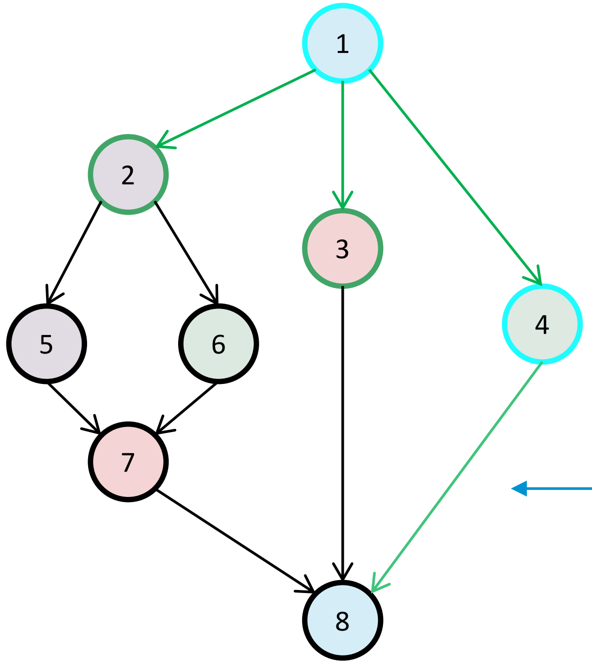
Init

Build DAG of whole system

At runtime

Dispatch when **all** predecessors have finished

Fully data-flow deterministic



4 finished,
Can not yet dispatch 8

Extension 1: Data-flow scheduling (1)

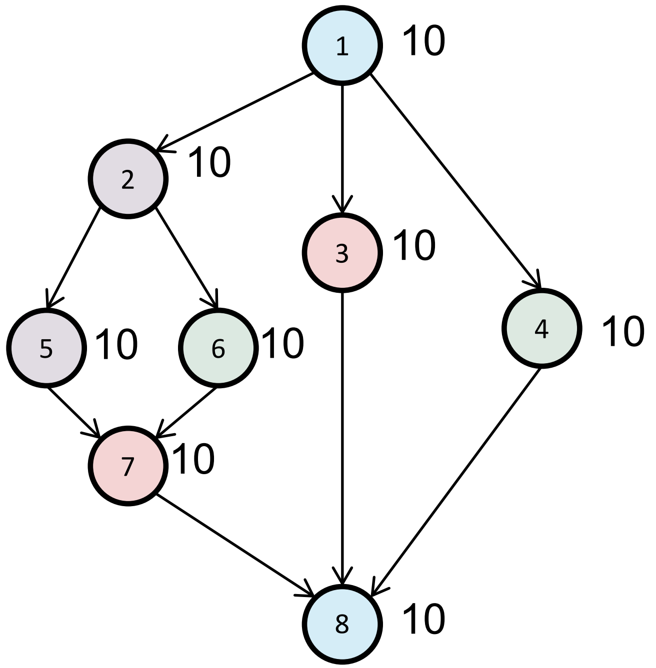
Explicit control allows extensions

Real-time requirement

- Stable predictable timing
- Tolerate timing variability
- Optimized E2E latency

Solution

- Timing annotations



Extension 1: Data-flow scheduling (2)

Solution

Timing annotations

Graph allows taking “future” into account

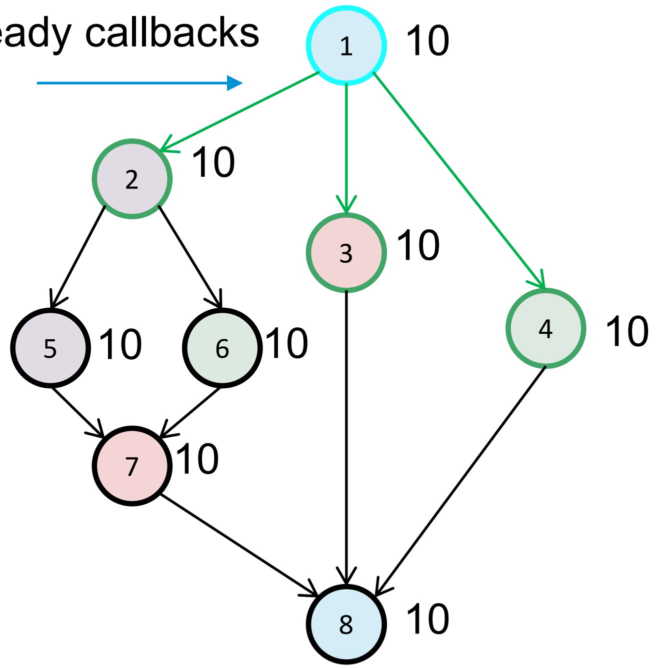
Heuristics to optimize system

Scheduling NP-complete

Derive priority from timing annotations

Dispatch according to priorities

2 cores
3 ready callbacks



Ready callbacks

2	3	4

CPU Cores

??	??

Extension 1: Data-flow scheduling (3)

Graph allows taking “future” into account

Heuristics to optimize system

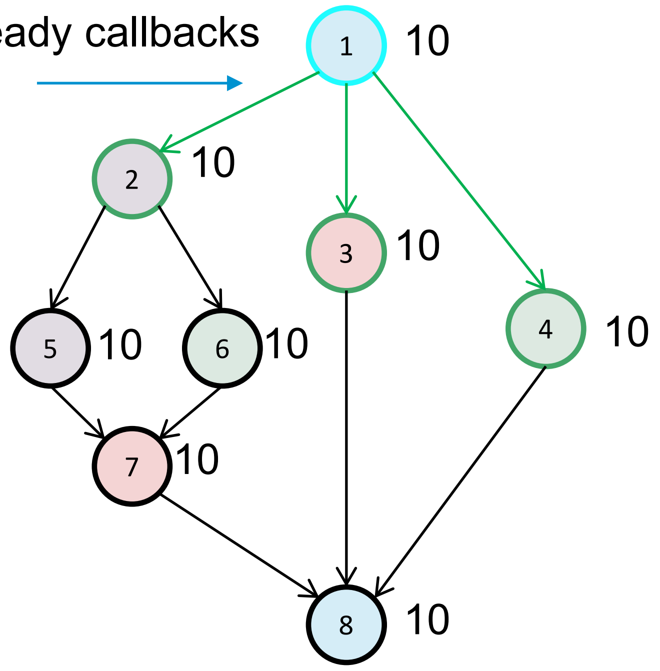
Scheduling NP-complete

Derive priority from timing annotations

Dispatch according to priorities

	Good dispatching decision:	Bad dispatching decision:
Step 1:	2,3,4	2,3,4
Step 2:	4,5,6	2
Step 3:	4,7	5,6
Step 4:	8	7
Step 5:	-	8
Overall Worst-Case Time	40	50

2 cores
3 ready callbacks



Ready callbacks

2	3	4

CPU Cores

??	??

Extension 2: Timeout handling for real-time progress

What if callbacks exceed timing budget?

Supervise exec time

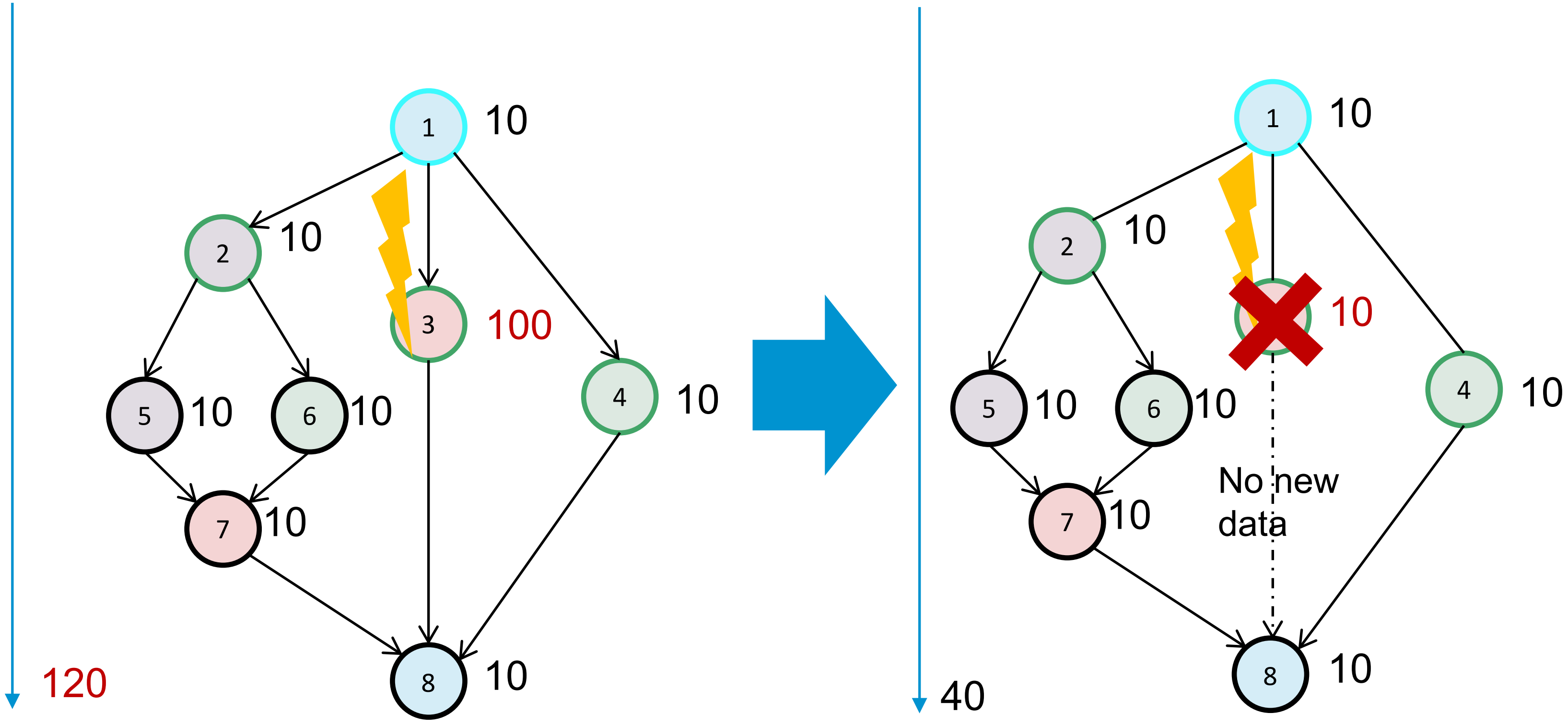
Trigger successors on timing violation

Trade Off: Determinism vs Real-time progress

Callback can choose how to handle

Callbacks know what happened

Decide: Abort, shutdown. use old data etc.



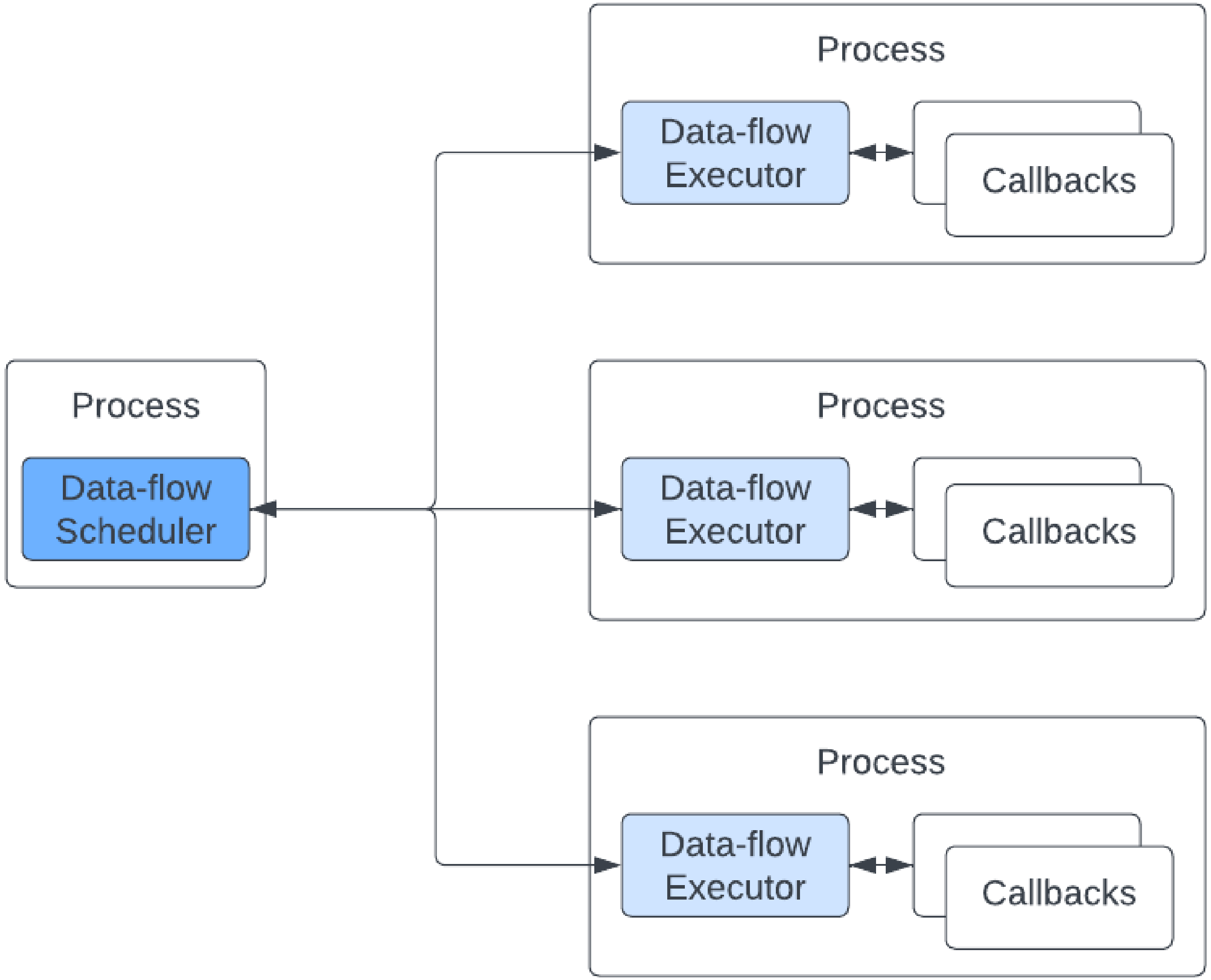
Not time deterministic but data & data-flow deterministic

Data-flow & time deterministic but not data deterministic

TTTechAuto

PoC Architecture

- Single Host
- Multiple Processes
- Process-local executors
- Central Data-flow Scheduler
  - Explicit dispatching decisions
  - Additional logic



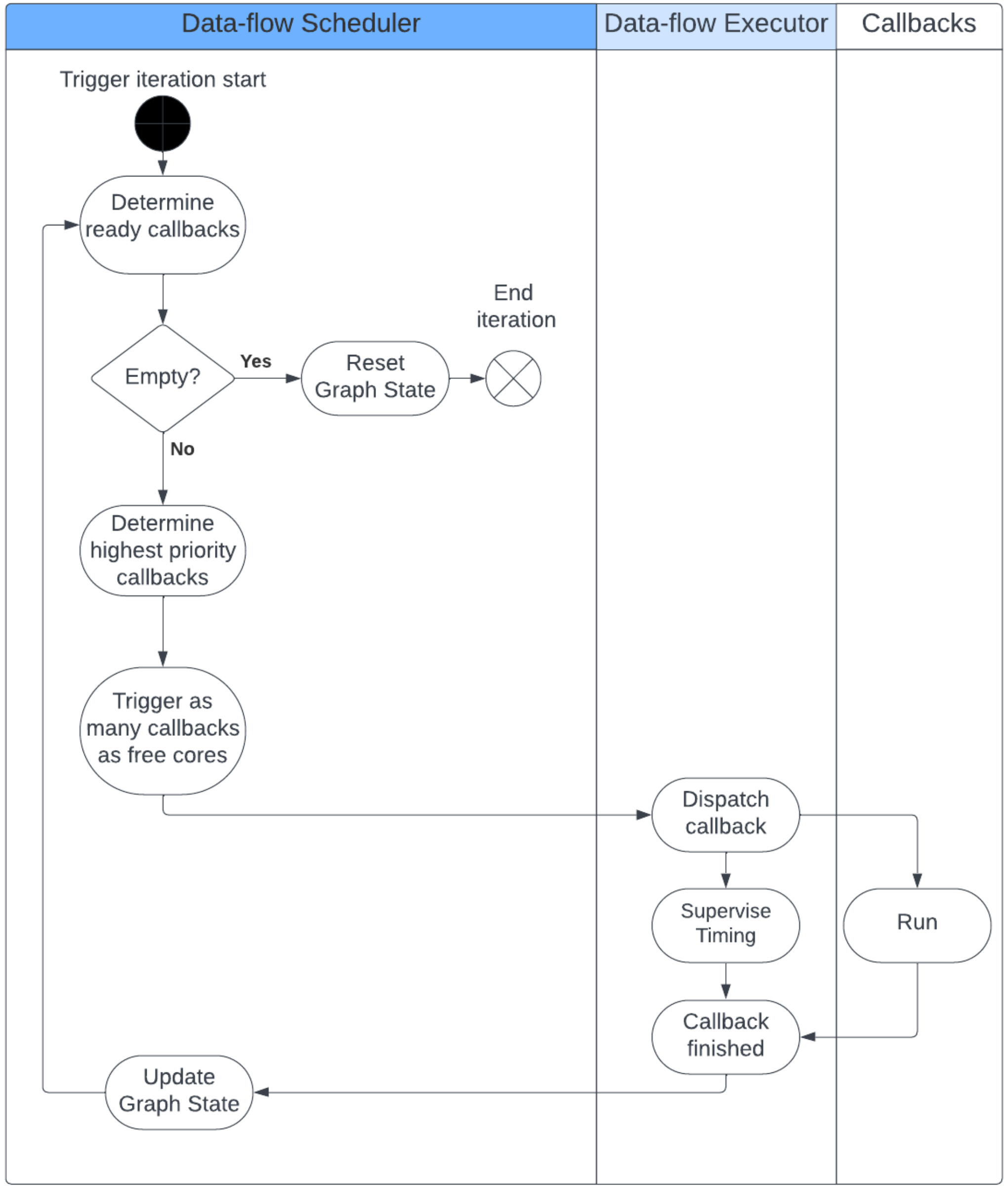
How the scheduler works

Maintain graph state

Dispatch according to schedule priorities

Coordinate local data-flow executors

Timing supervision



How to use – POC example

Multi-rate external inputs?

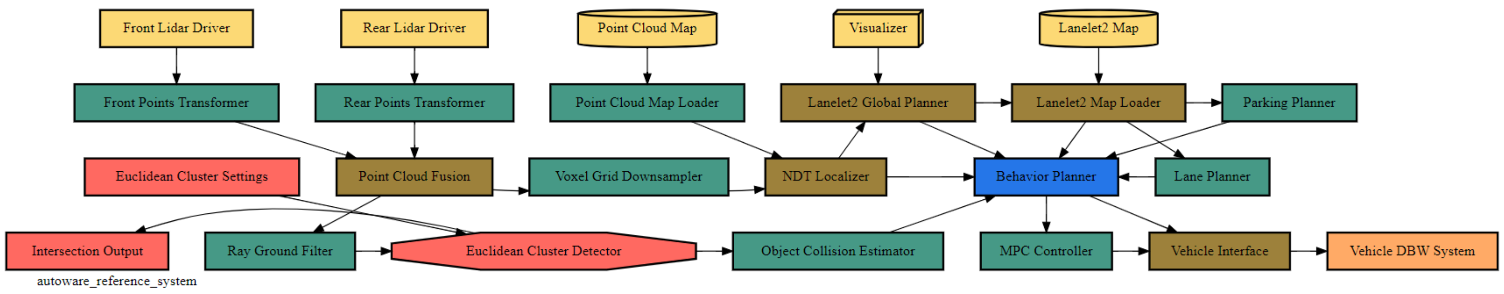
DDS history

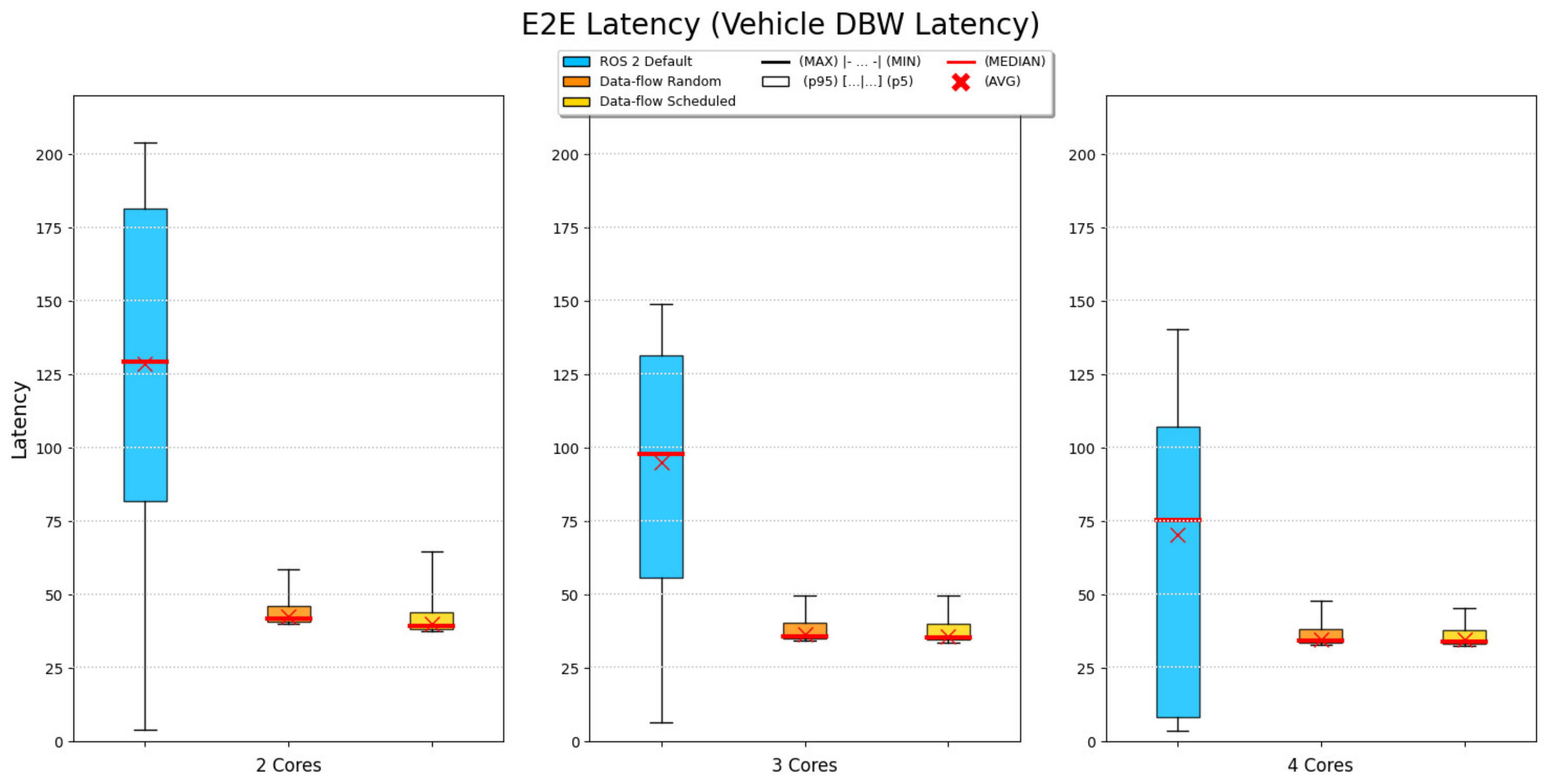
Read all samples

```
class MultiSub : public rclcpp::Node {
…
public:
   void callback() {
             // Do the subscription
               for (auto sub_ptr : subscriptions) {
                if (sub_ptr->take(msg, msg_info)) {
                   std::shared_ptr<void> type_erased_msg =
                       std::make_shared<std_msgs::msg::String>(msg);
                   std::cout << " Value: " << msg.data.c_str() << "\n";
                } else {
                   RCLCPP_WARN(this->get_logger(), "   |->No message available");
                  }
                }
            // Add the business logic

               // Do the publishing
               for (auto pub_ptr : publishers) {
                           auto message = std_msgs::msg::String();
                           message.data = "my data";
                           pub_ptr->publish(message);
                     }
   }
…
};

int main(int argc, char *argv[]) {
  rclcpp::init(argc, argv);
…
  std::vector<std::string> publish_topics   {"topic3", "topic4"};
  std::vector<std::string> subscribe_topics {"topic1", "topic2"};
  auto node = std::make_shared<MyNode>("Node3", publish_topics, subscribe_topics);
 //  for each callback in node
  DFSched::CallbackInfoVector cinfo(1);
  //  here we define the callback that is going to be called when all predecessors are done
  cinfo[0].callback_ptr = [&node]() { node->callback(); };
  cinfo[0].subs = subscribe_topics;
  cinfo[0].pubs = publish_topics;
  //  Time supervision based on thread CPU usage, realtime or no supervision at all
  cinfo[0].supervision_kind = DFSched::TimeSupervision::ThreadCPUTime;
  cinfo[0].runtime = 1000000; //in microseconds
  cinfo[0].id = 0;

  DFSched::DFSExecutor executor(std::string("Node3"), cinfo);
  executor.spin();
…
}
```

Evaluation & Results of PoC

- Raspberry PI 4
- Autoware reference system
  - 1 Node = 1 Process
  - Different crunch values for callbacks
- Scheduling
  - Still performing all work
- Without timeout handling





Summary

TTTechAuto

Main results

- Data-flow approach allows:
  - More deterministic and predictable runtime behaviour
  - Fully data-flow & data deterministic when timing is not relevant (during functional testing)
  - Guaranteed forward progress (for real-time requirements in the field)
  - Better control over temporal behaviour
- Explicit control of dispatching enables adding custom logics
  - E.g. Scheduling/Timeout handling
  - But more possible

Lessons learned

- Applications needs to be adapted to make proper use of approach
- Handling timeouted callbacks is delicate
- Async data handling in communication stack is problematic for polling access to topic – Improvements to comm stack would be beneficial

Next steps

- Potentially fix some limitations/add new features:
  - Distinguish between critical and non-critical callbacks
  - More graceful handling of timeouted threads
  - Trigger callbacks every n'th cycle
- Performance optimizations
- Discuss general usefulness for the ROS2 community and how to contribute

TTTechAuto

Thank you

Any questions?

christopher.helpa@tttech-auto.com

Let's talk

Follow us on LinkedIn
www.linkedin.com/company/tttech-auto

www.tttech-auto.com