Model-based development of deterministic, portable real-time software components

Prof. Dr. Wolfgang Pree

SoftwareResearch.net

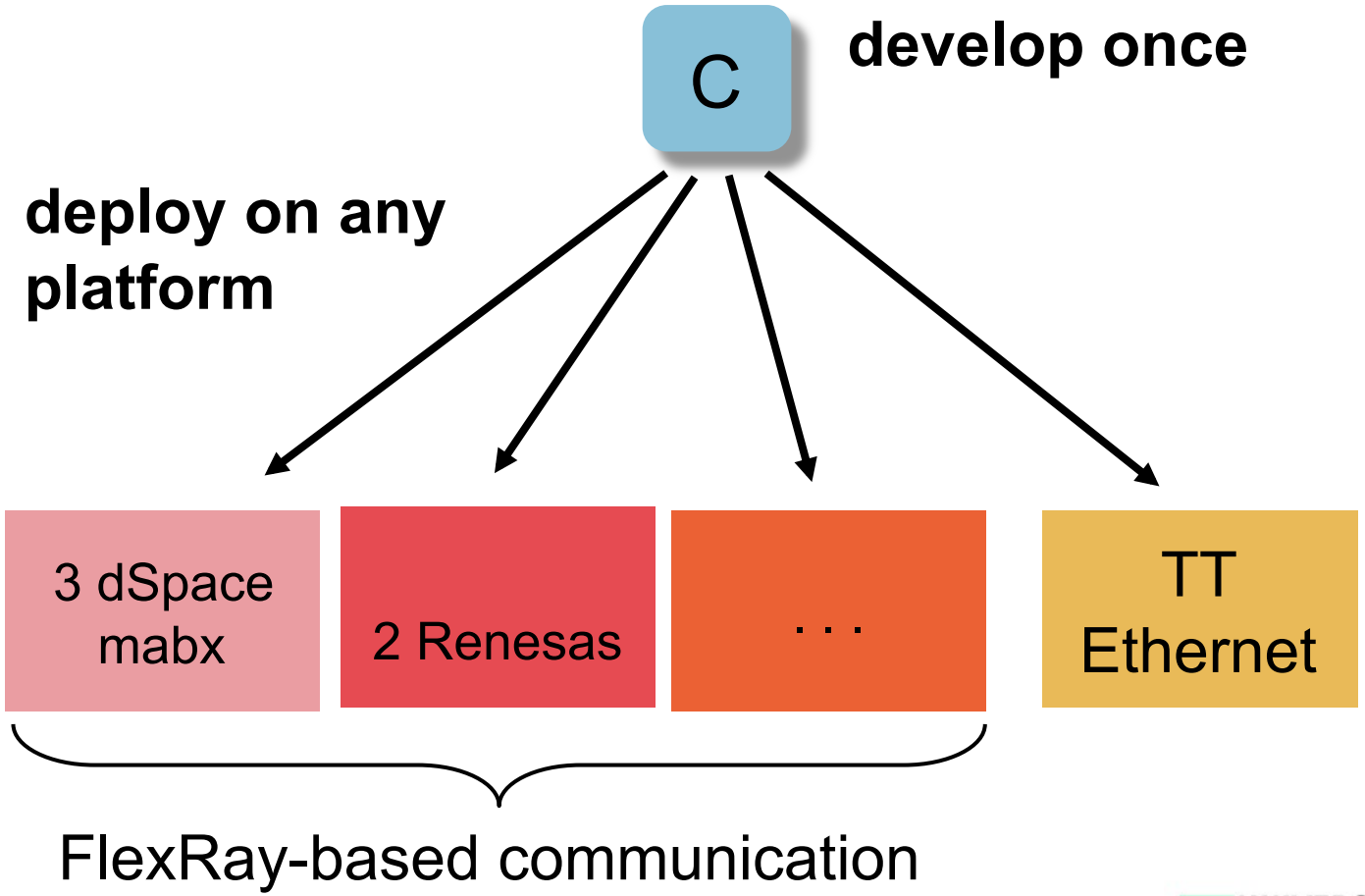
UNIVERSITÄT SALZBURG

Overview

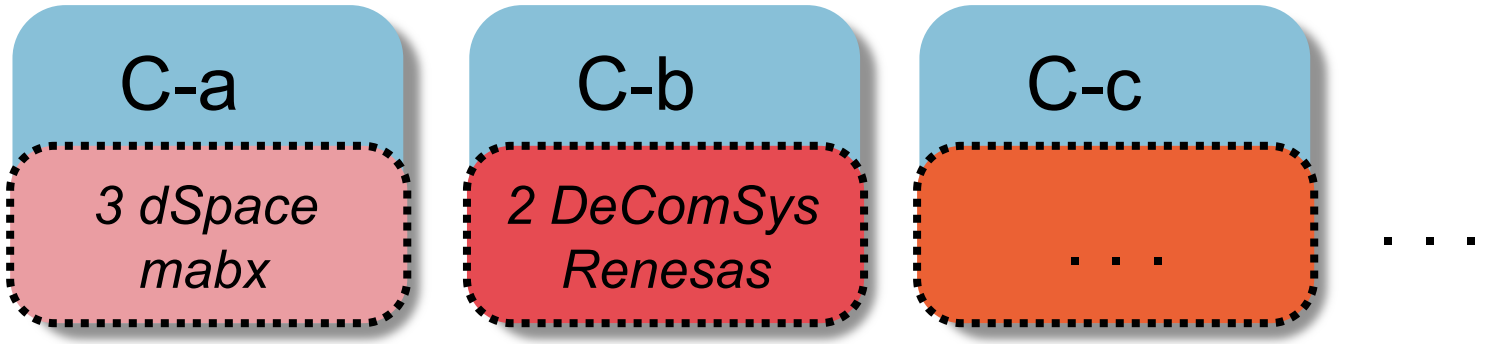
- **Motivation for a paradigm shift**
  - so far: platform first, software tailored to platform
  - future: software first, mapping to platforms later
  - requires appropriate platform abstractions
- The **Timing Definition Language (TDL)** in a nut shell
- **Transparent distribution of TDL components**
- **TDL development process**

Motivation

The TDL way:

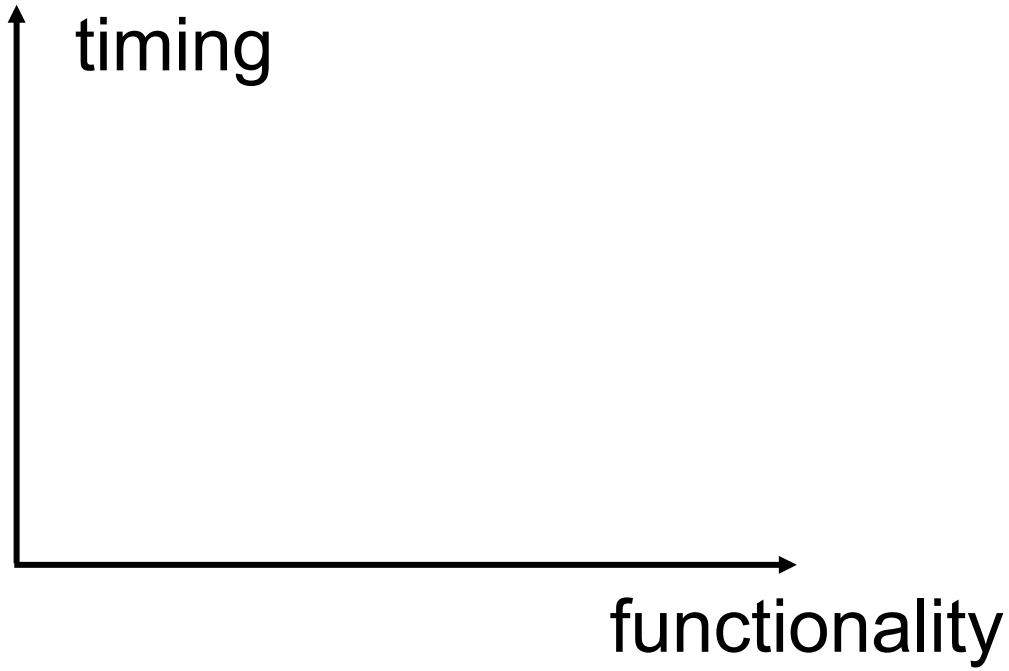


State-of-the-art:

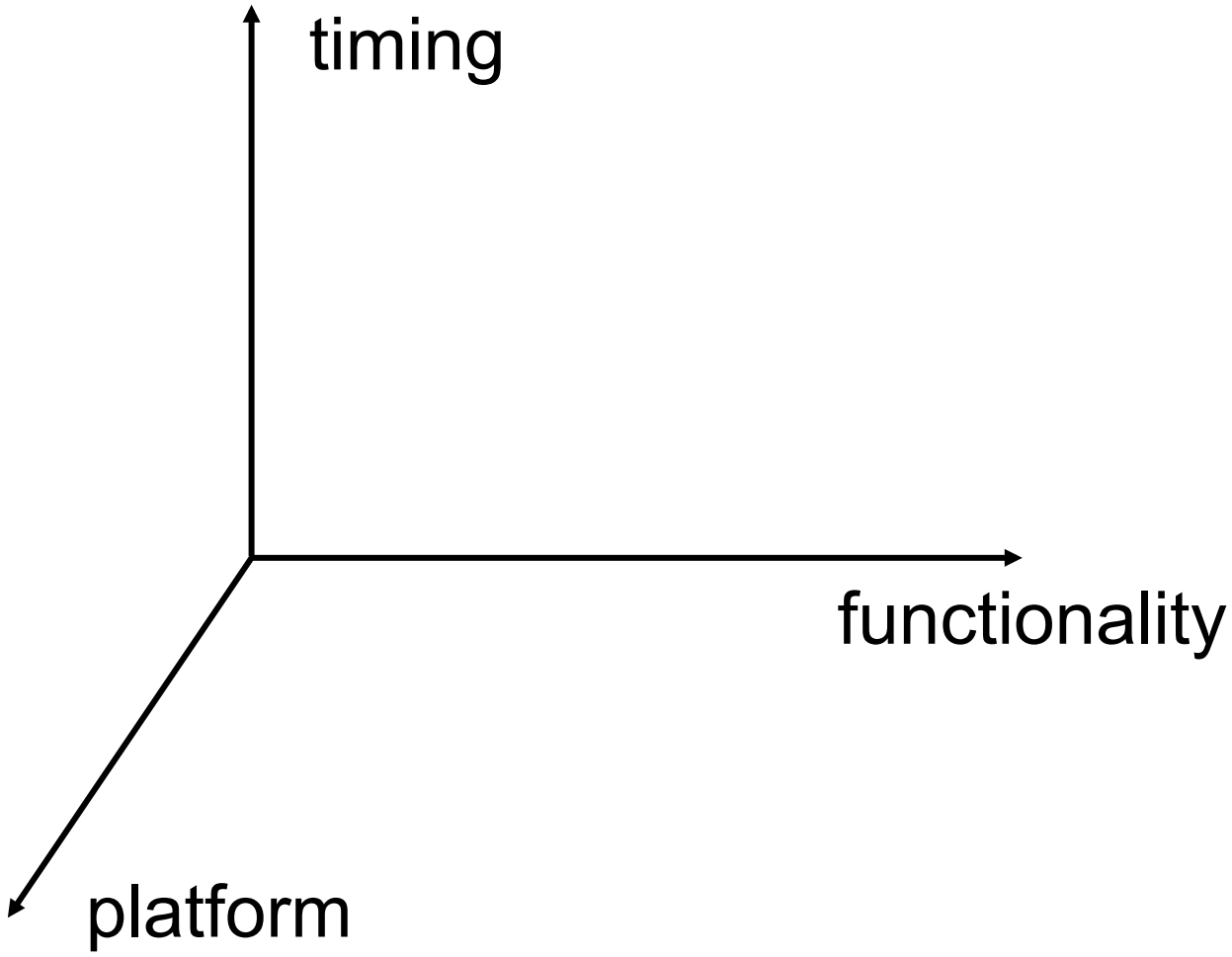


developers have to deal with 3 dimensions

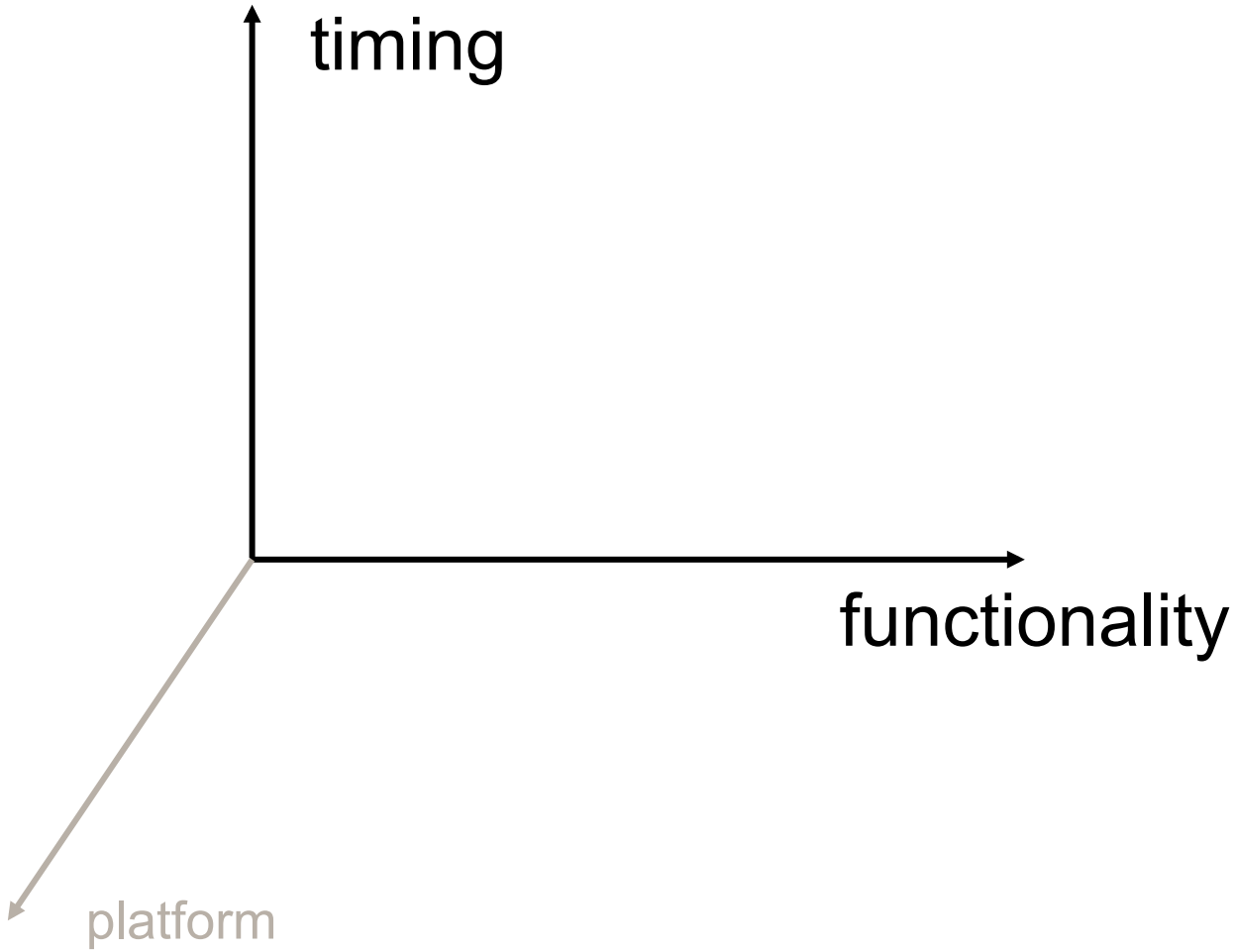
developers have to deal with 3 dimensions



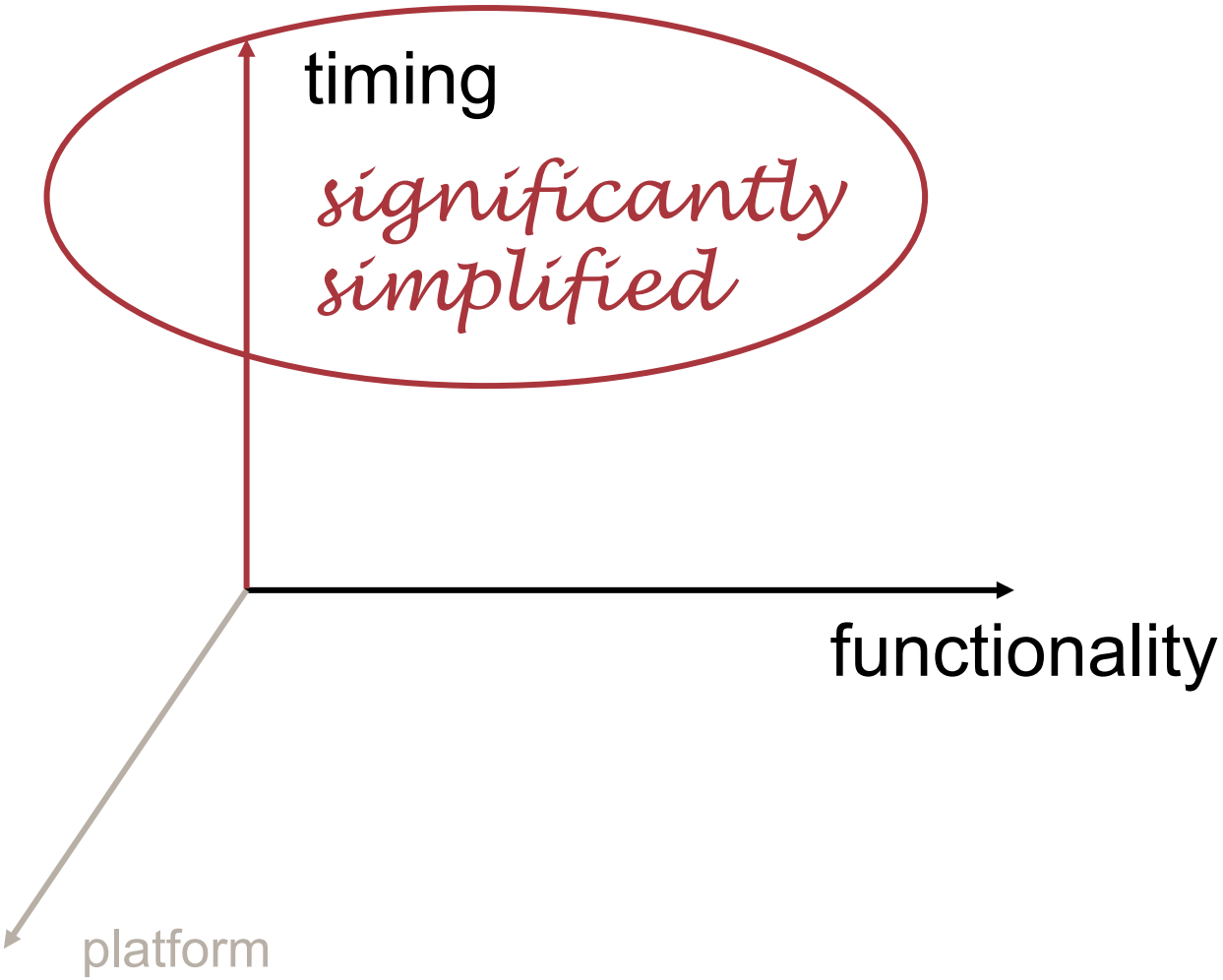
developers have to deal with 3 dimensions



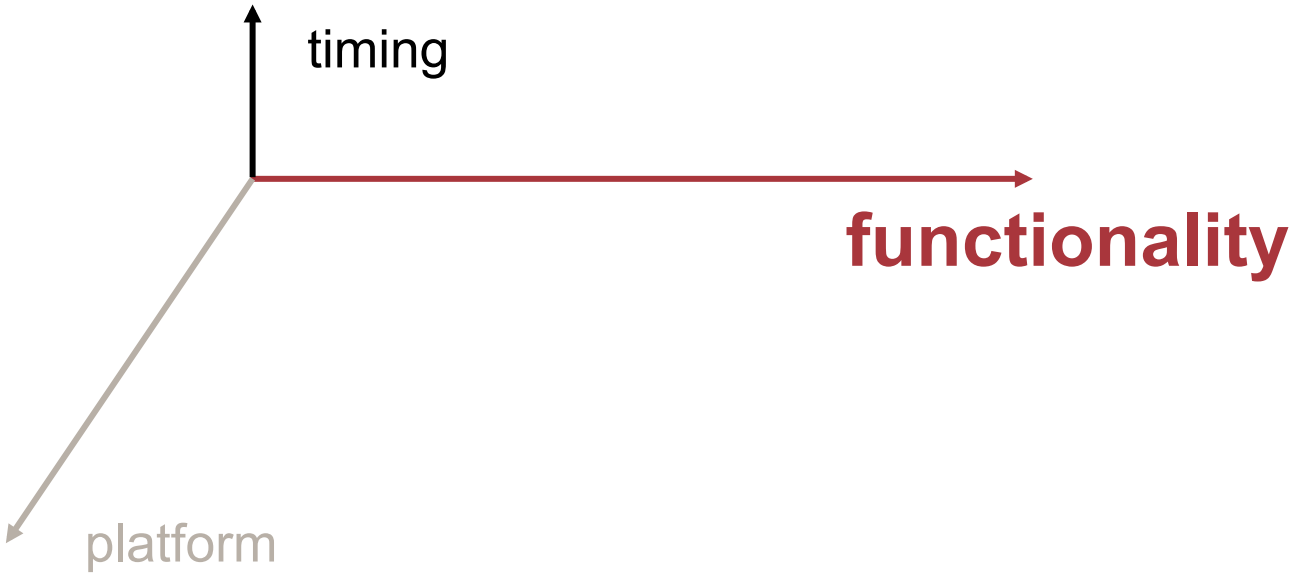
TDL reduces this to 2 dimensions



TDL reduces this to 2 dimensions



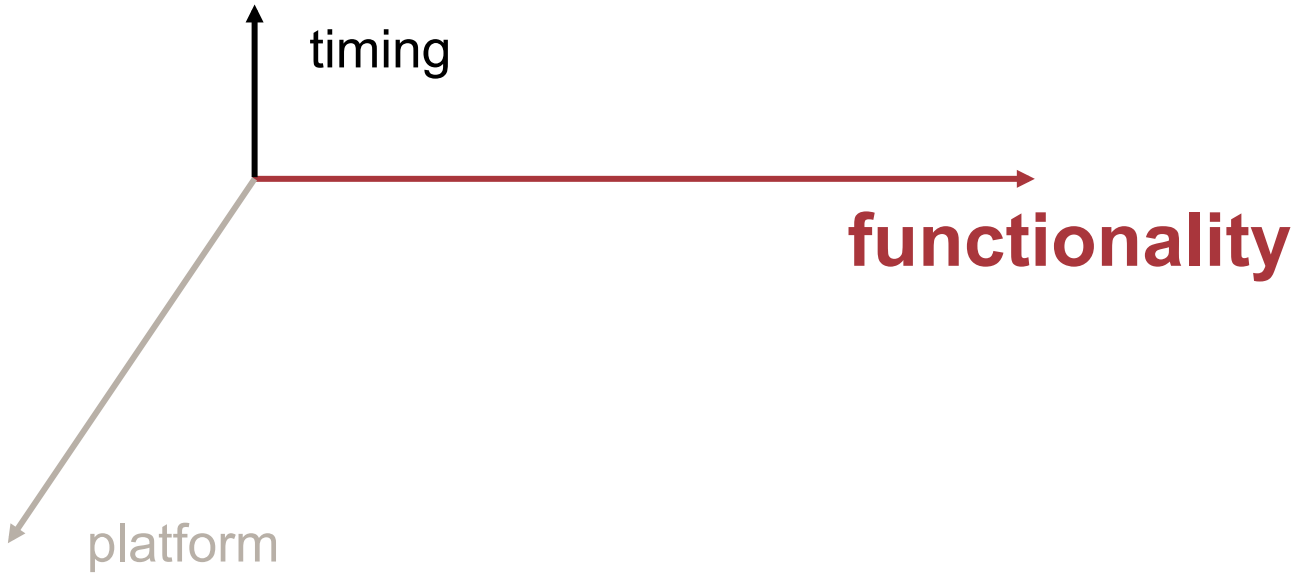
TDL allows your developers to focus on the functionality



UNIVERSITÄT SALZBURG

TDL allows your developers to focus on the functionality

3D → 1,5D



UNIVERSITÄT SALZBURG

TDL leads to enormous gains in efficiency and quality

eg, FlexRay development reduced by a **factor of 20**

- 1 person year => 2 person weeks

deterministic system:

- simulation and executable on platform always exhibit equivalent (observable) behavior
- time and value determinism guaranteed

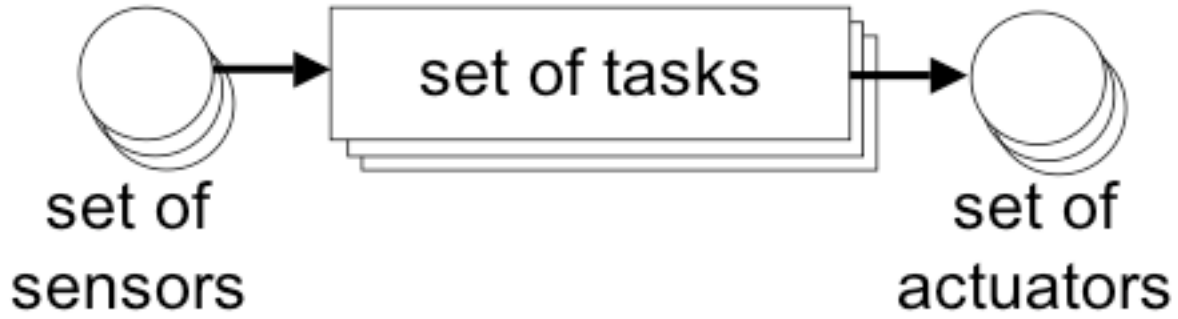
flexibility to change topology, even platform

- automatic code generators take care of the details

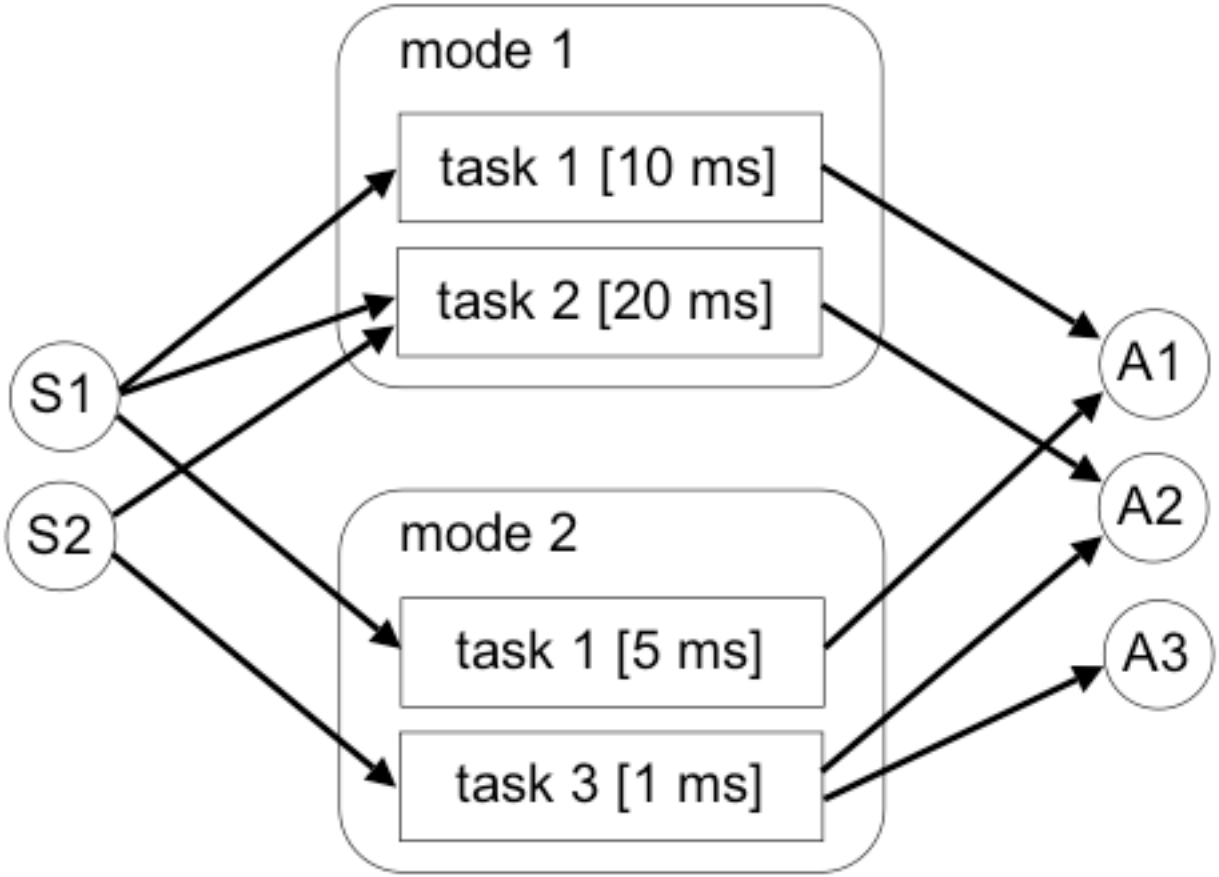
TDL in a nut shell

What is TDL?

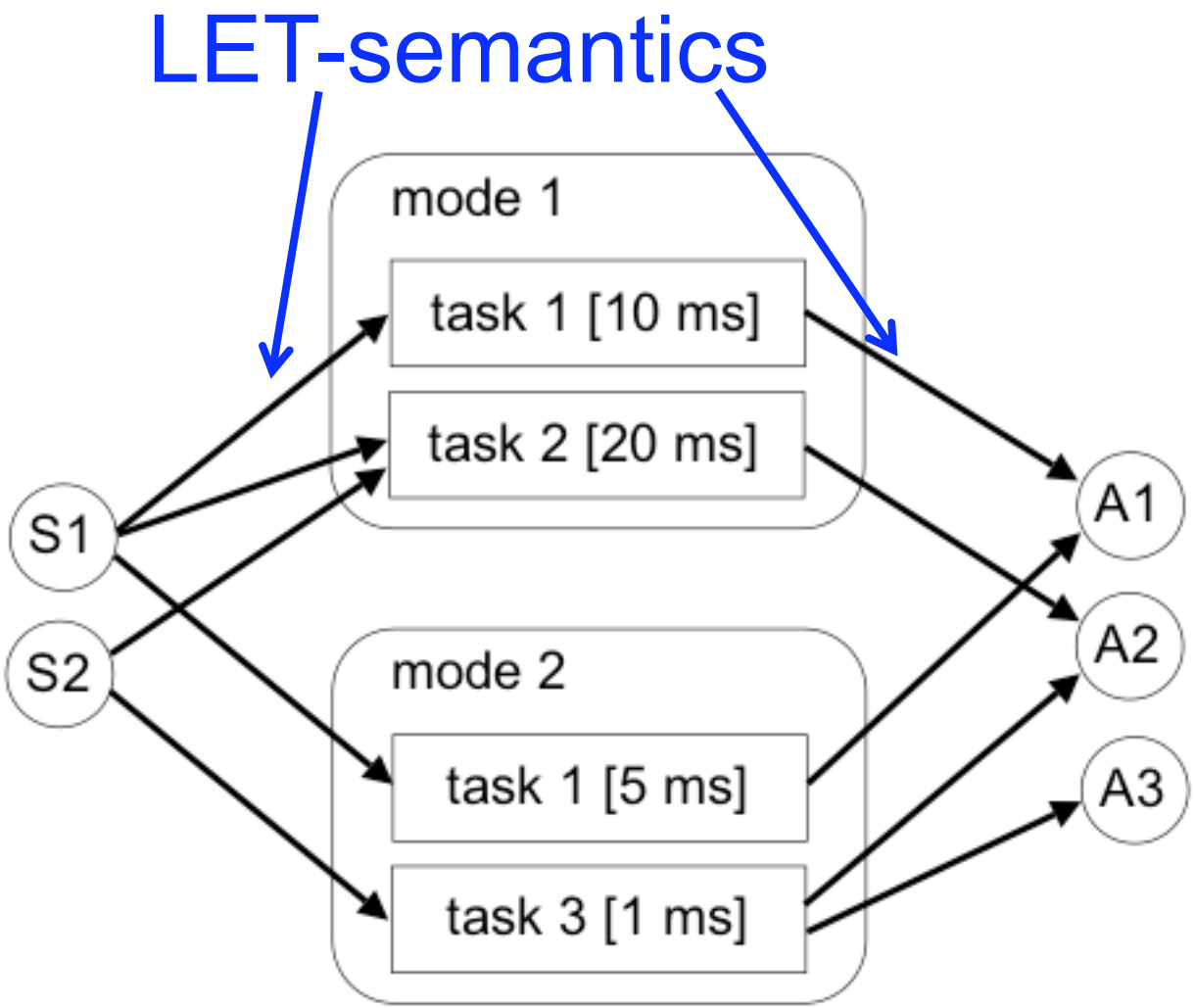
- A high-level textual notation for defining the timing behavior of a real-time application.



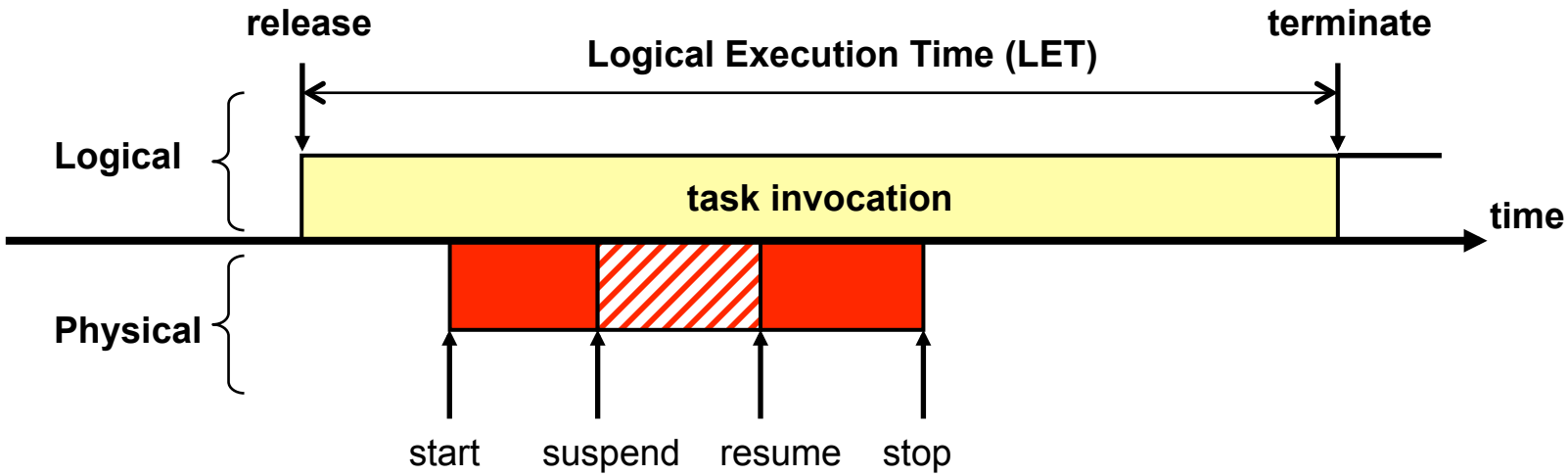
Multi-rate, multi-mode systems (I)



Multi-rate, multi-mode systems (II)



Logical Execution Time (LET) abstraction (II)

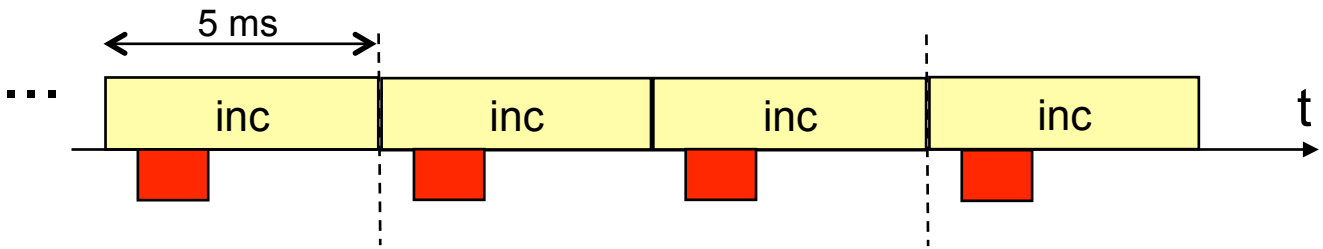


ET <= WCET <= LET

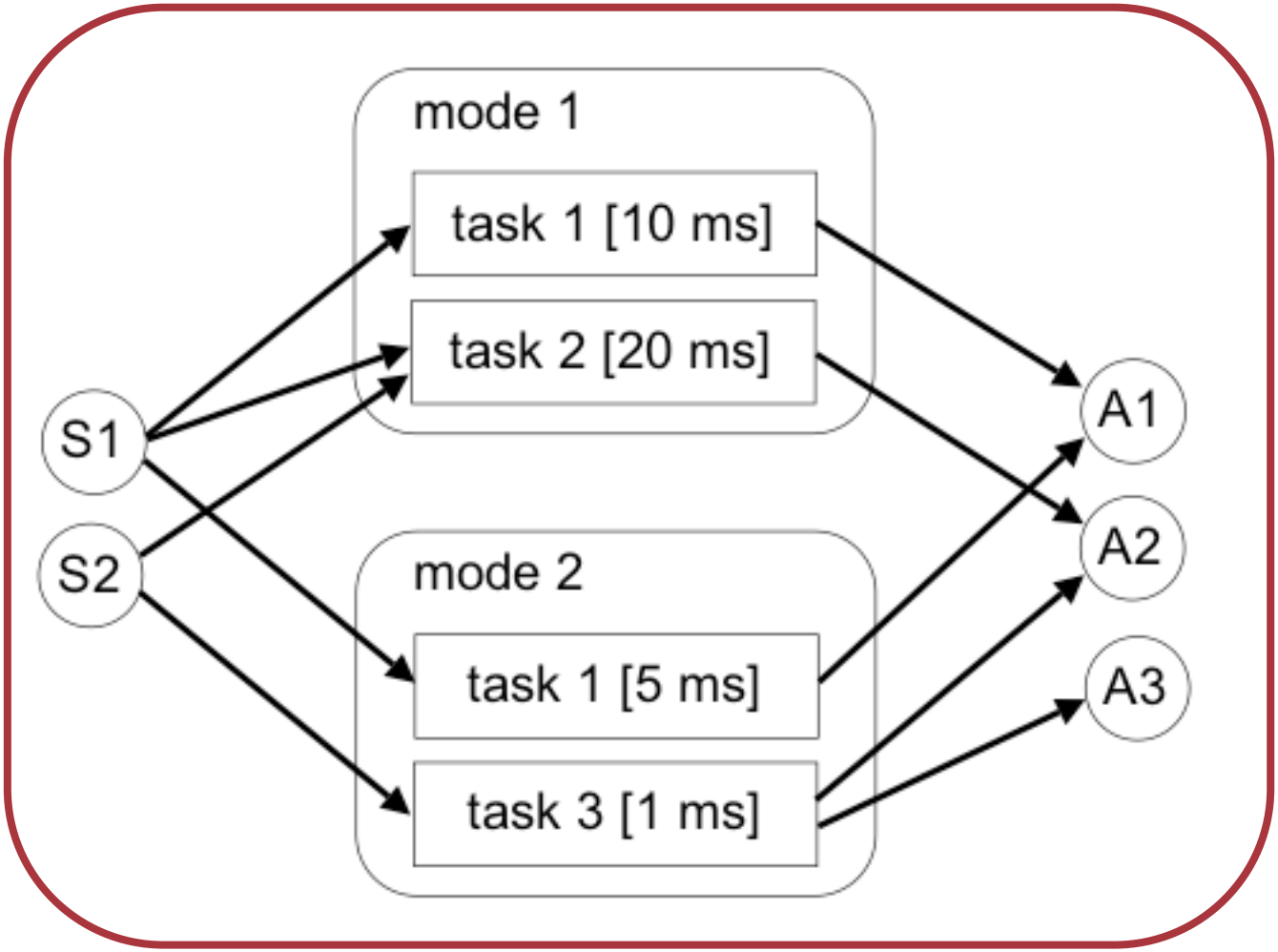
results are available at 'terminate'

for digital controllers: LET can also be zero => no delays

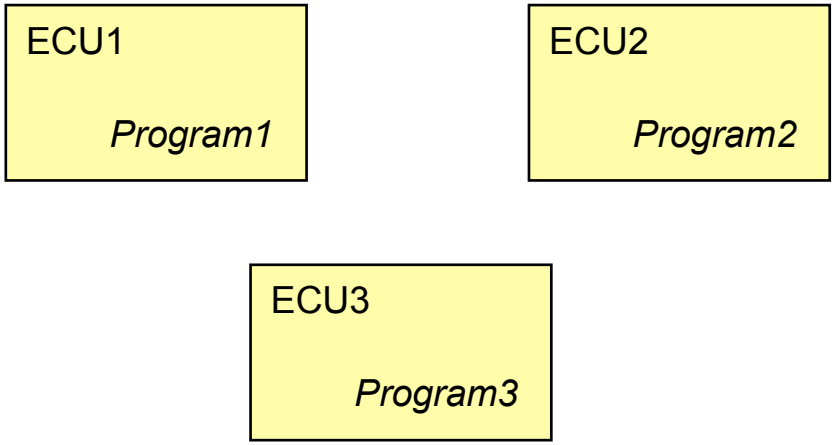
sample task with LET = 5ms



TDL module: modes, sensors and actuators form a unit

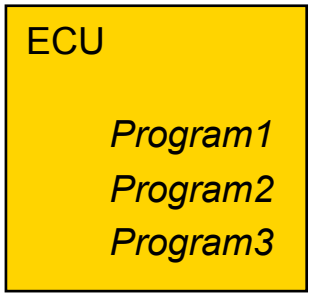


Motivation for TDL modules



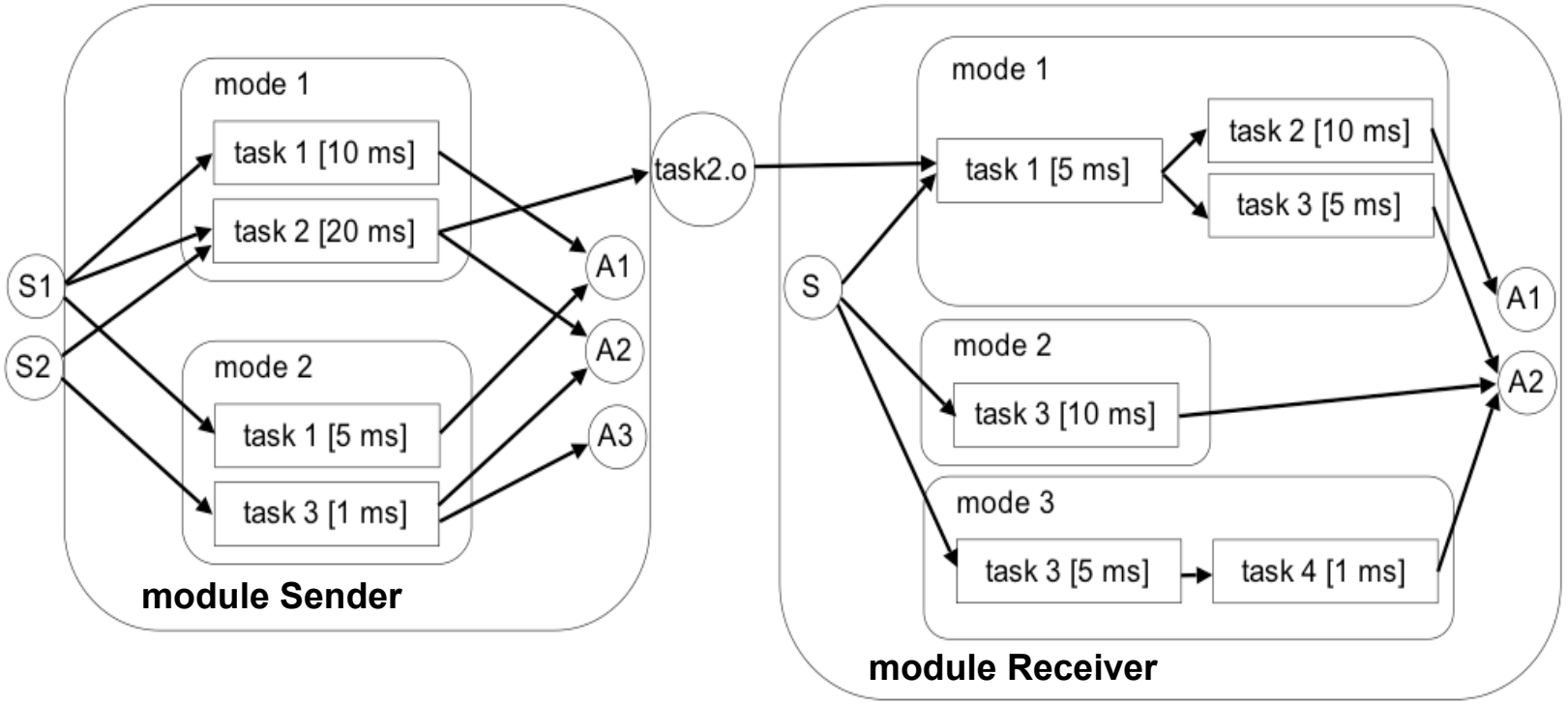
- e.g. modern cars have up to 80 control units (ECUs)
- ECU consolidation is a topic
- run multiple programs on one ECU
- leads to TDL modules

TDL modules

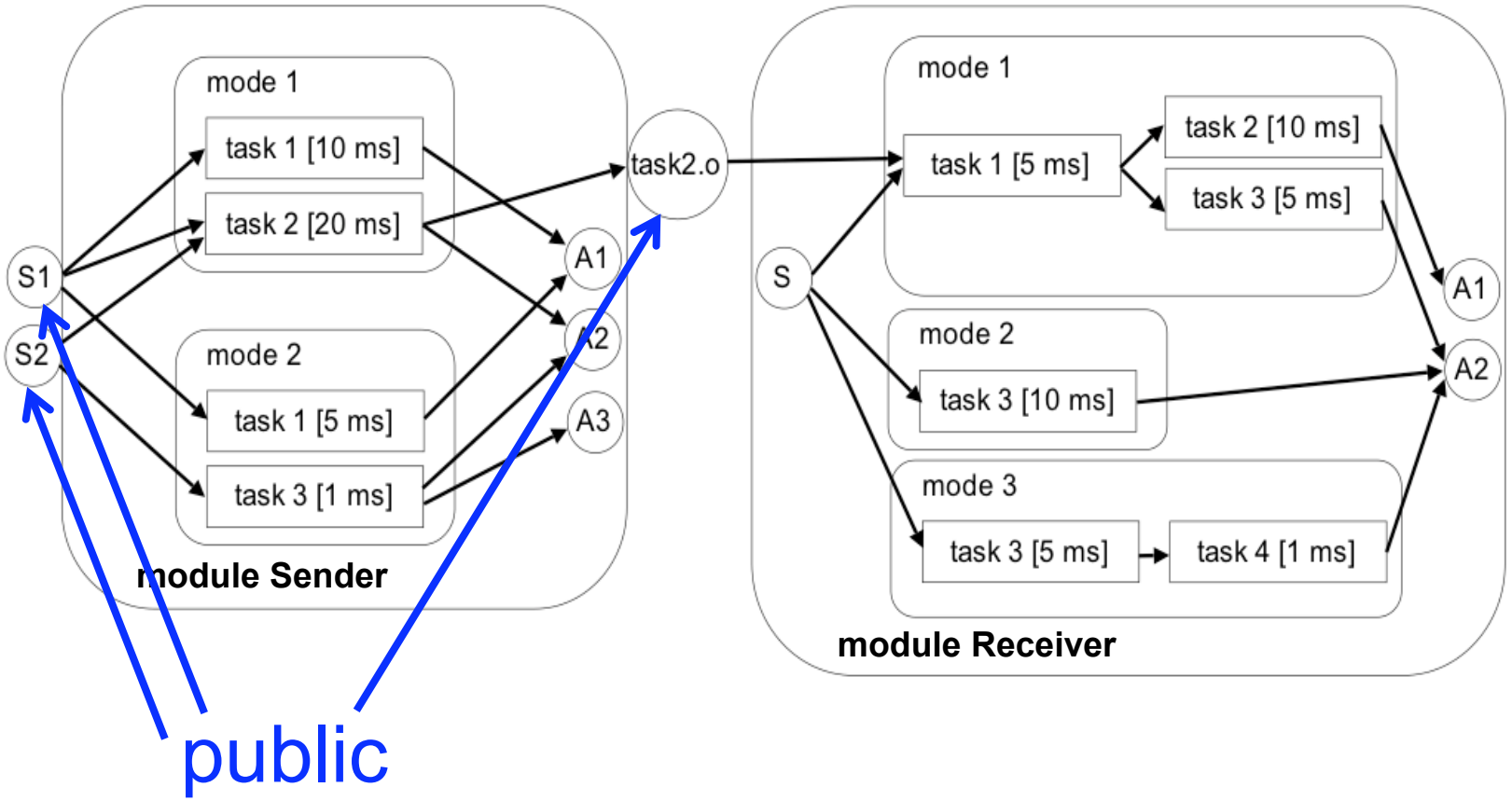


- ProgramX is called a *module*
- modules may be independent
- modules may also refer to each other
- modules can be used for multiple purposes

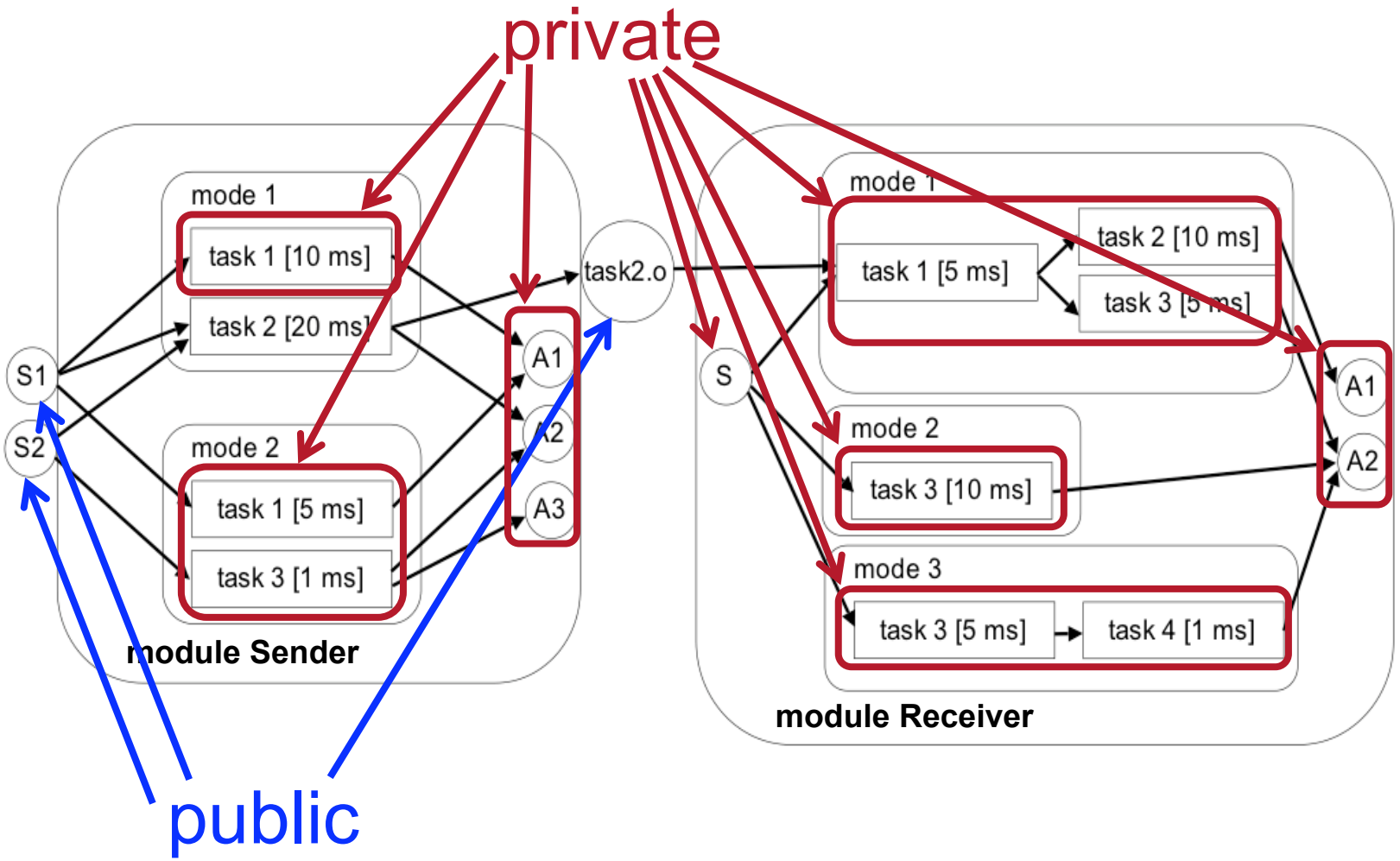
Example: Receiver imports from Sender module



Example: Receiver imports from Sender module

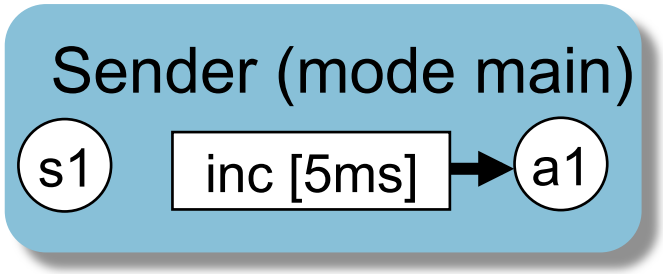


Example: Receiver imports from Sender module



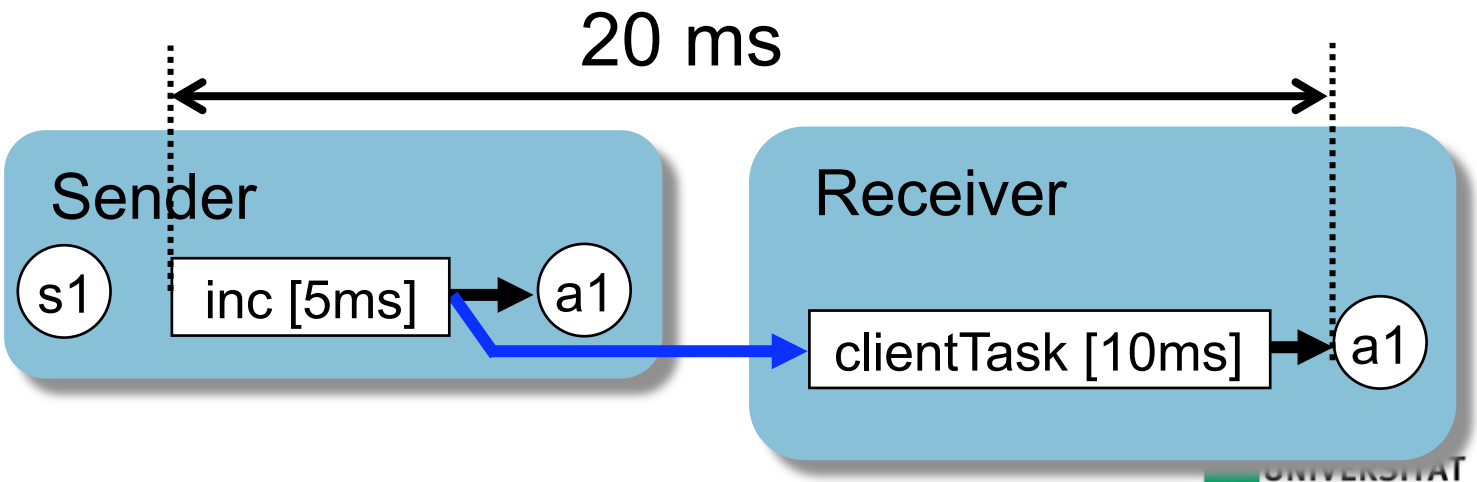
TDL syntax by example

```
module Sender {

  sensor boolean s1 uses getS1;
  actuator int a1 uses setA1;

  public task inc {
    output int o := 10;
    uses incImpl(o);
  }

  start mode main [period=5ms] {
    task
      [freq=1] inc();              // LET = 5ms / 1 = 5ms
    actuator
      [freq=1] a1 := inc.o;        // update every 5ms
    mode
      [freq=1] if exitMain(s1) then freeze;
  }

  mode freeze [period=1000ms] {}
}
```

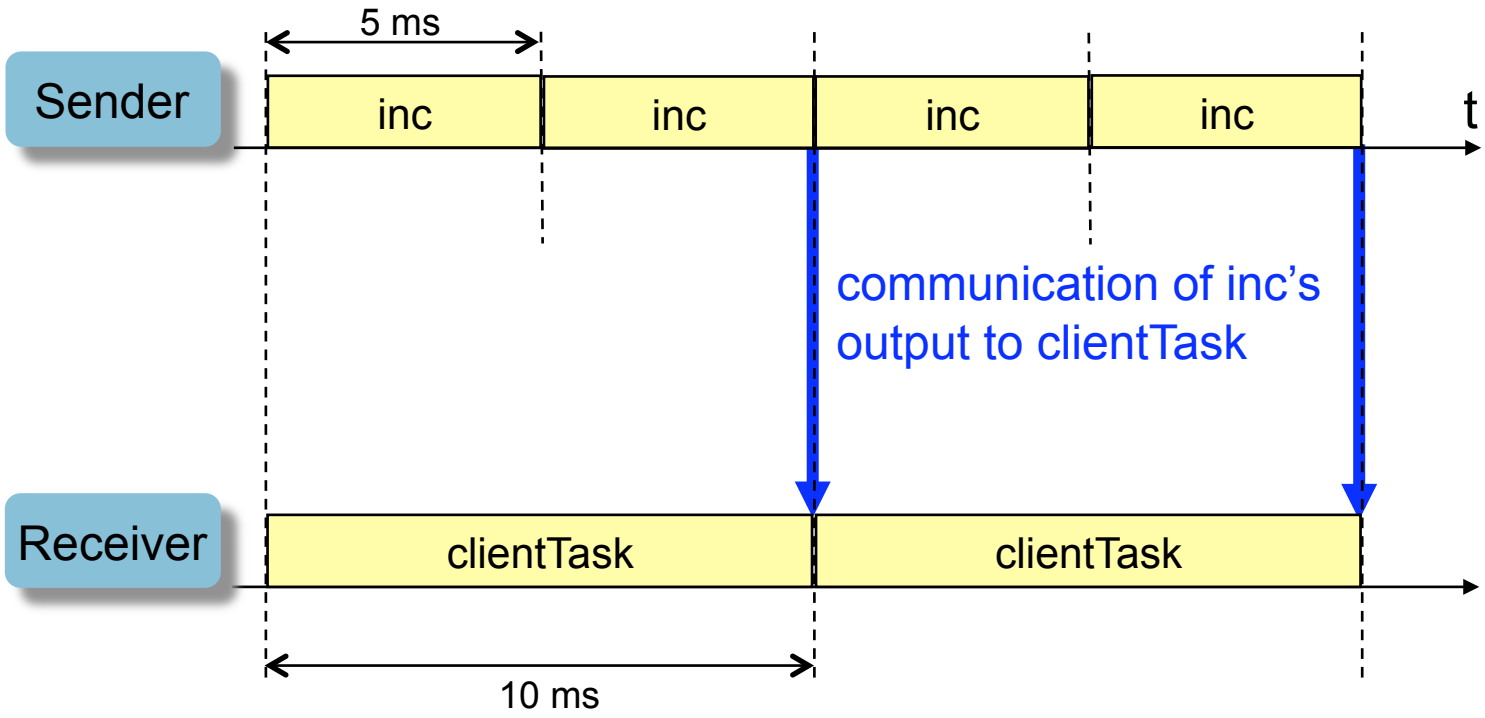


Module import

```
module Receiver {

  import Sender;
  …
  task clientTask {
    input int i1;
    …
  }
  mode main [period=10ms] {
    task [freq=1] clientTask(Sender.inc.o); // LET = 10ms / 1 = 10ms
    …
  }
}
```

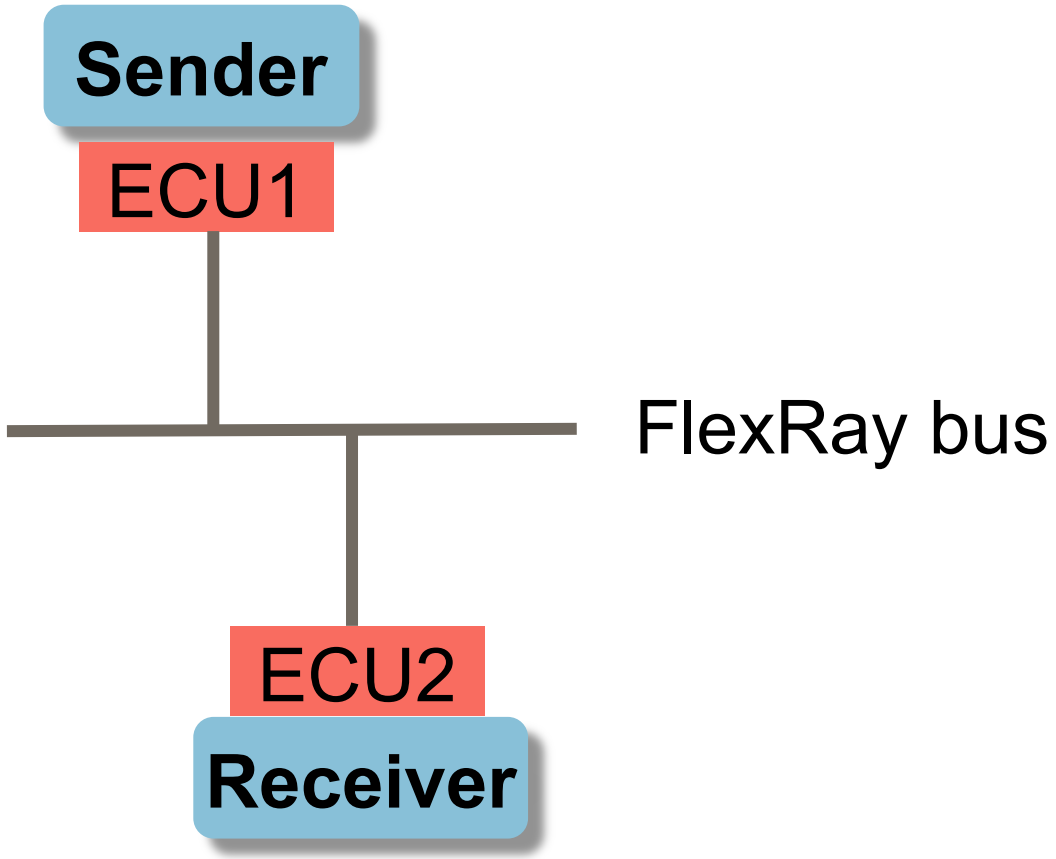


LET-behavior (independent of component deployment)



Transparent distribution

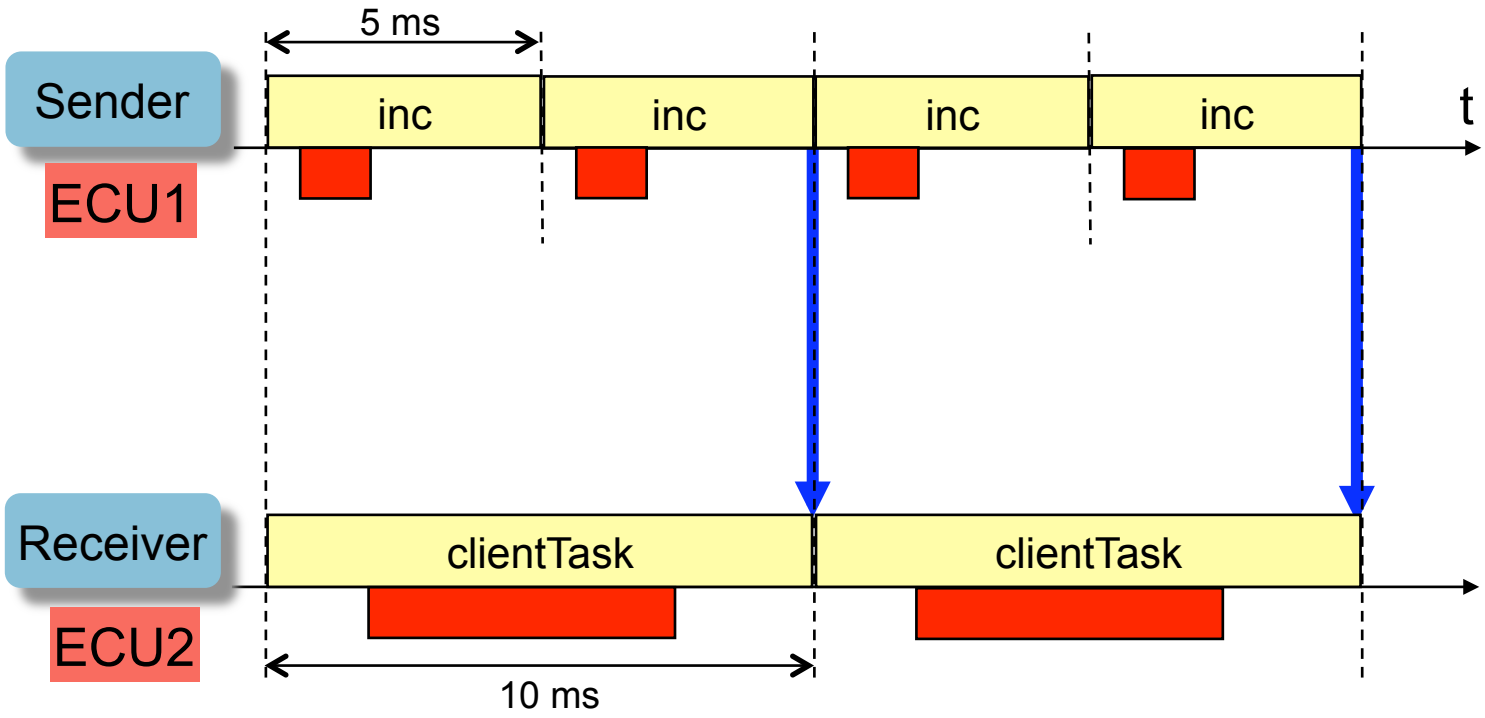
TDL module-to-node-assignment



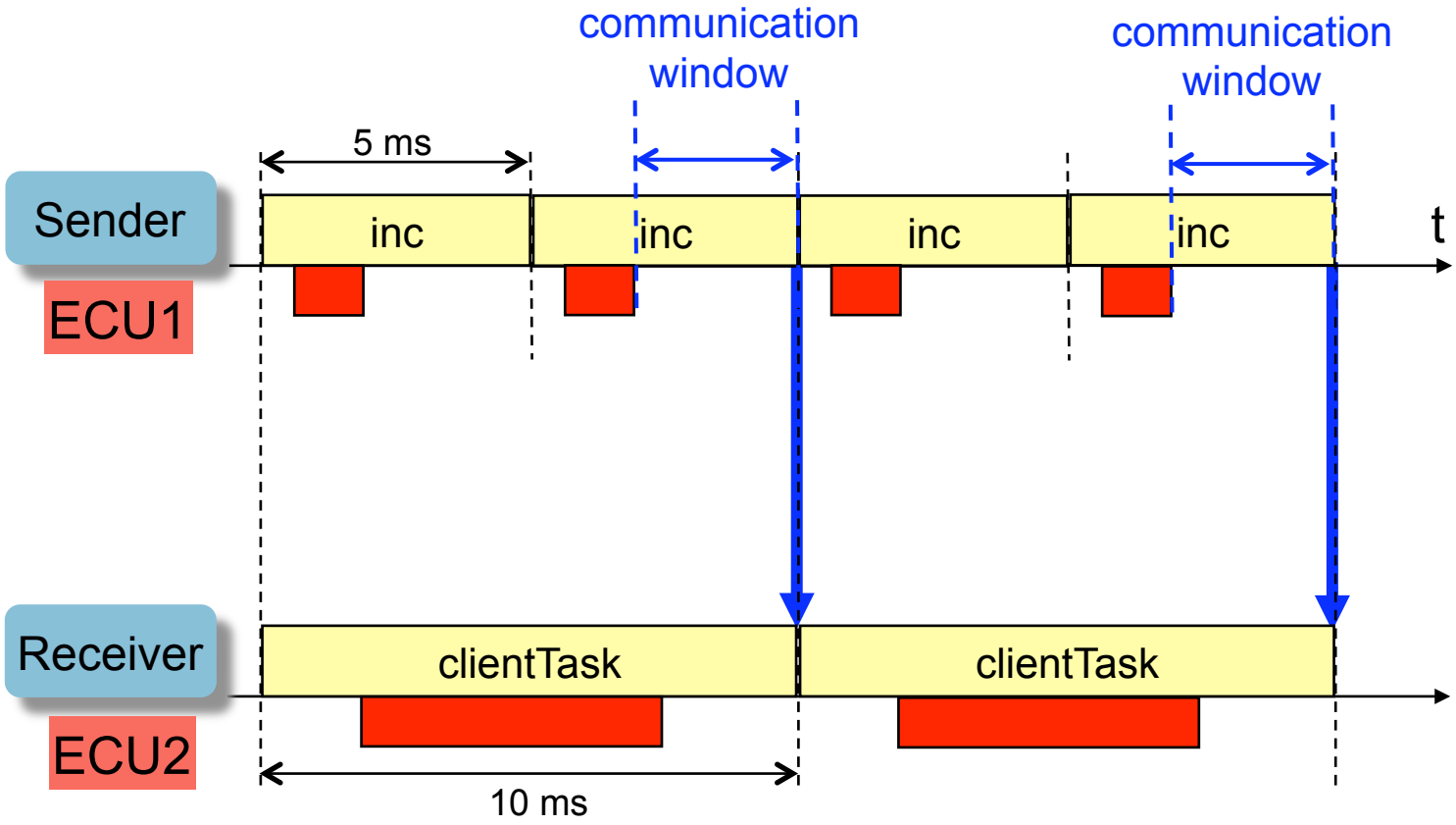
Transparent distribution of TDL components:

- Firstly, at runtime a set of **TDL components behaves exactly the same**, no matter if all components are **executed on a single node or** if they are **distributed across multiple nodes**.
  The logical timing is always preserved, only the physical timing, which is not observable from the outside, may be changed.
- Secondly, **for the developer of a TDL component, it does not matter where the component itself and any imported component are executed**.

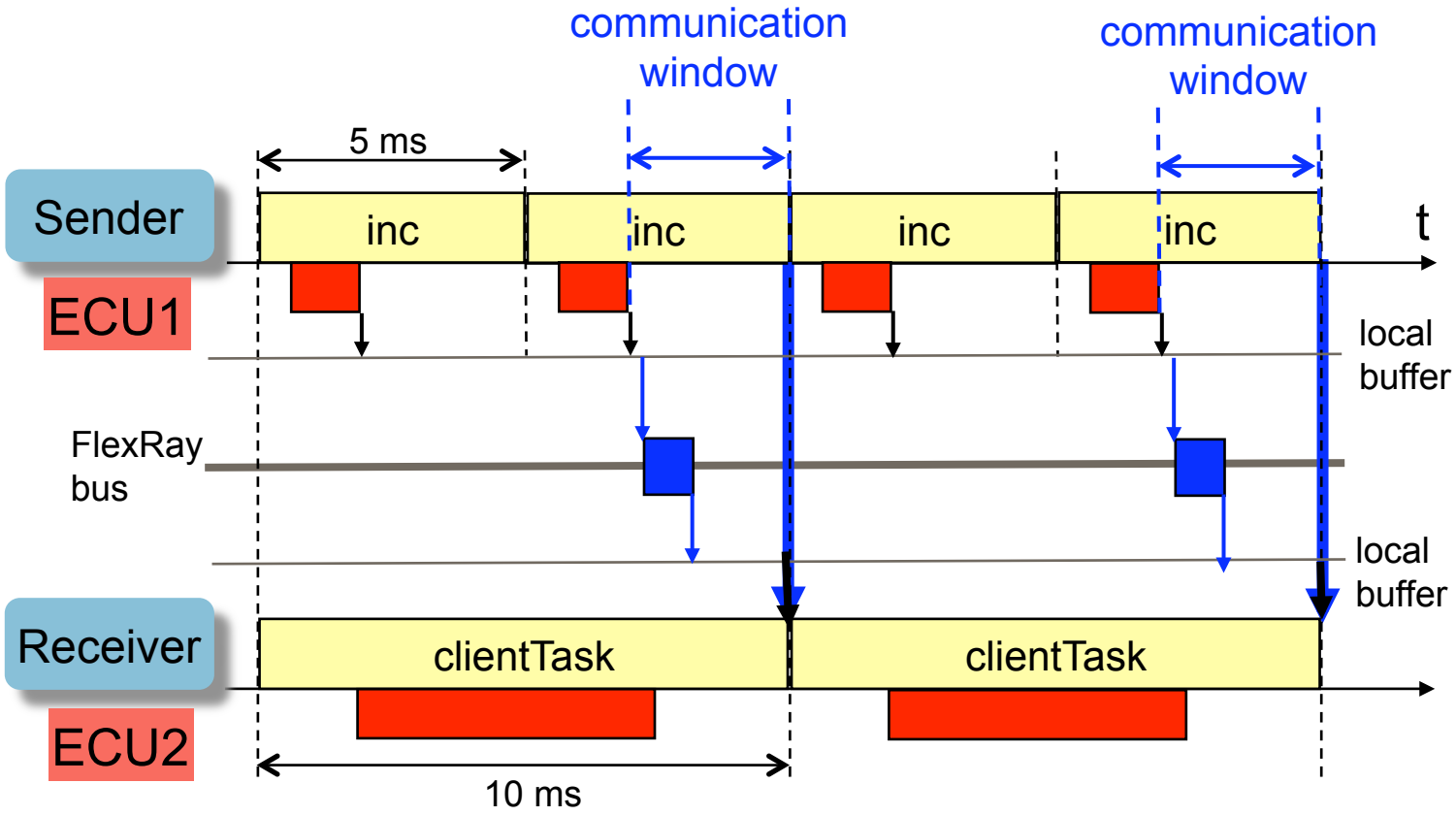
sample physical execution times on ECU1/ECU2



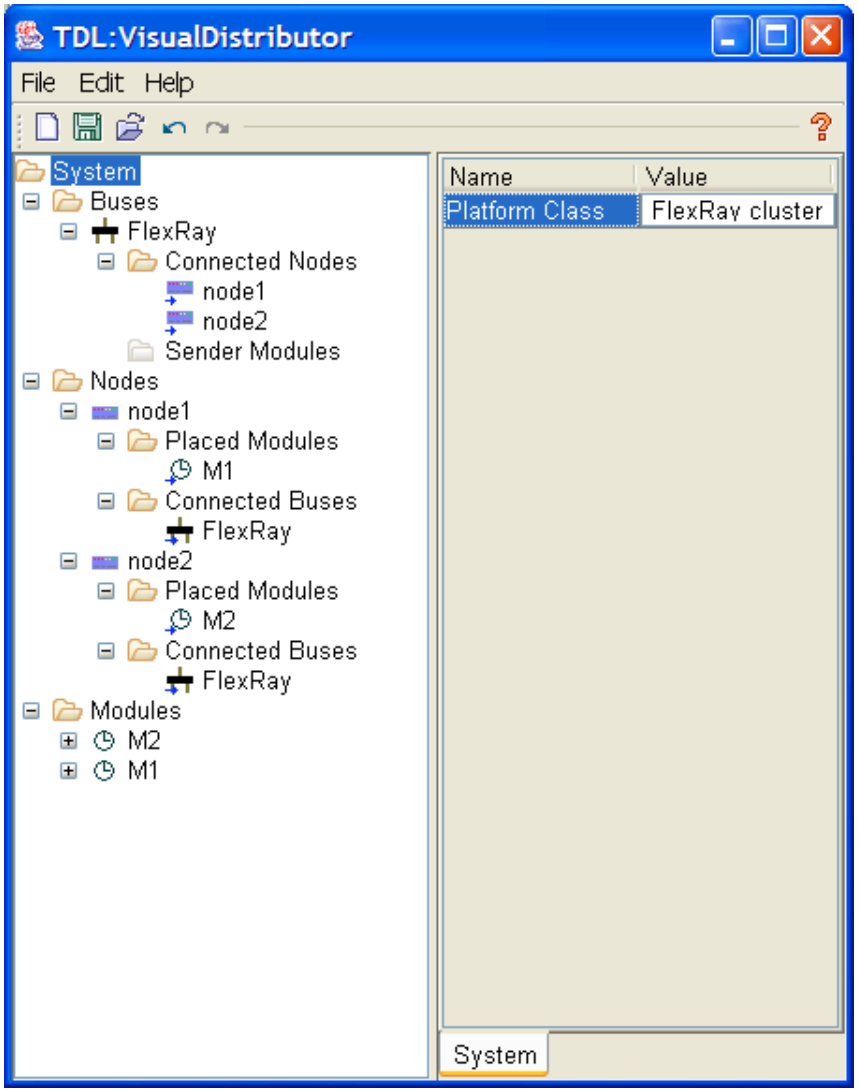
Constraints for automatic schedule generation



Bus schedule generation

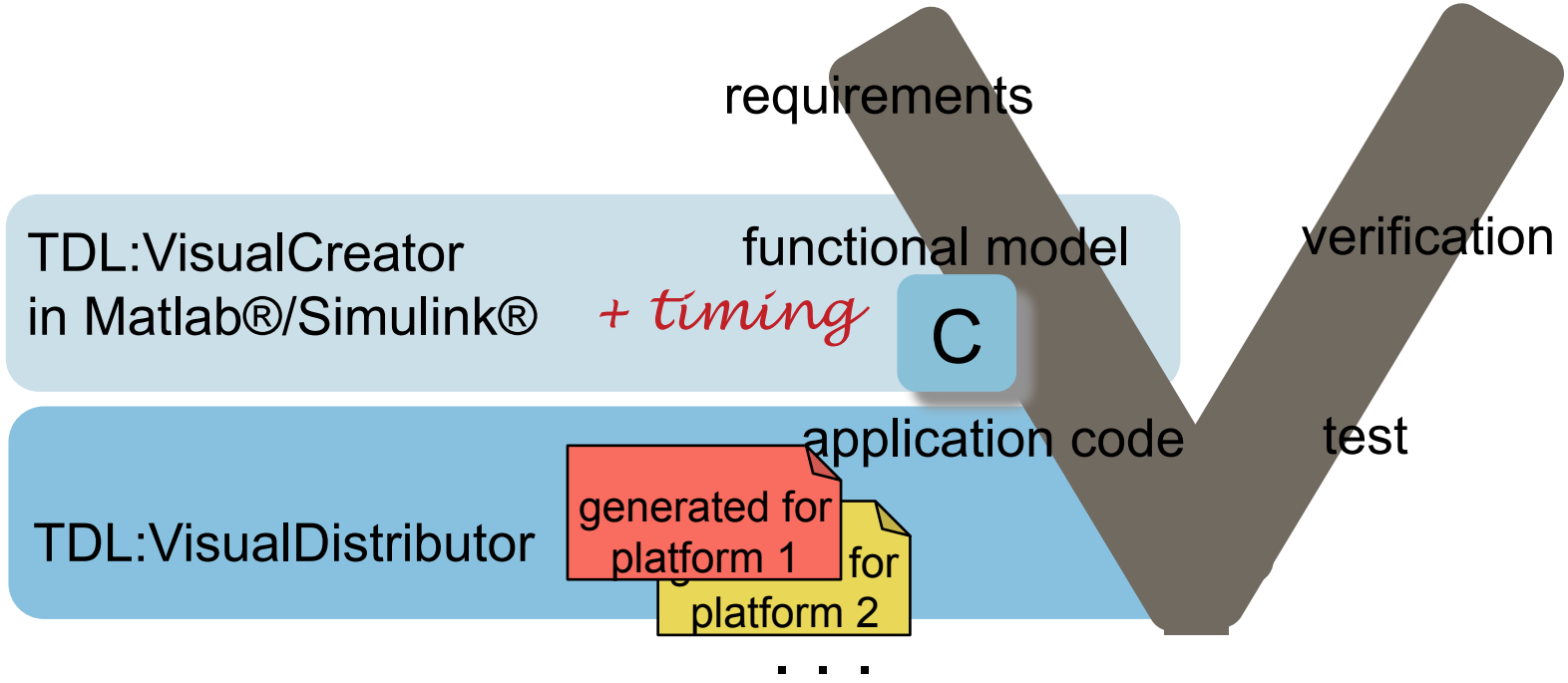


TDL:VisualDistributor maps TDL modules to nodes



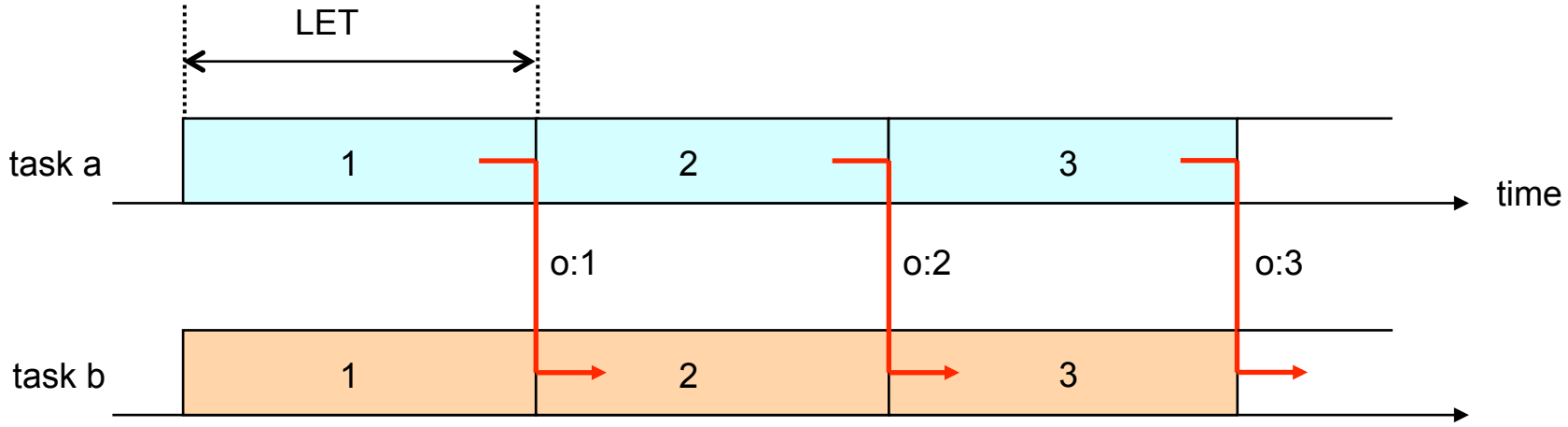
TDL-based development process

preeTEC tools in the V model

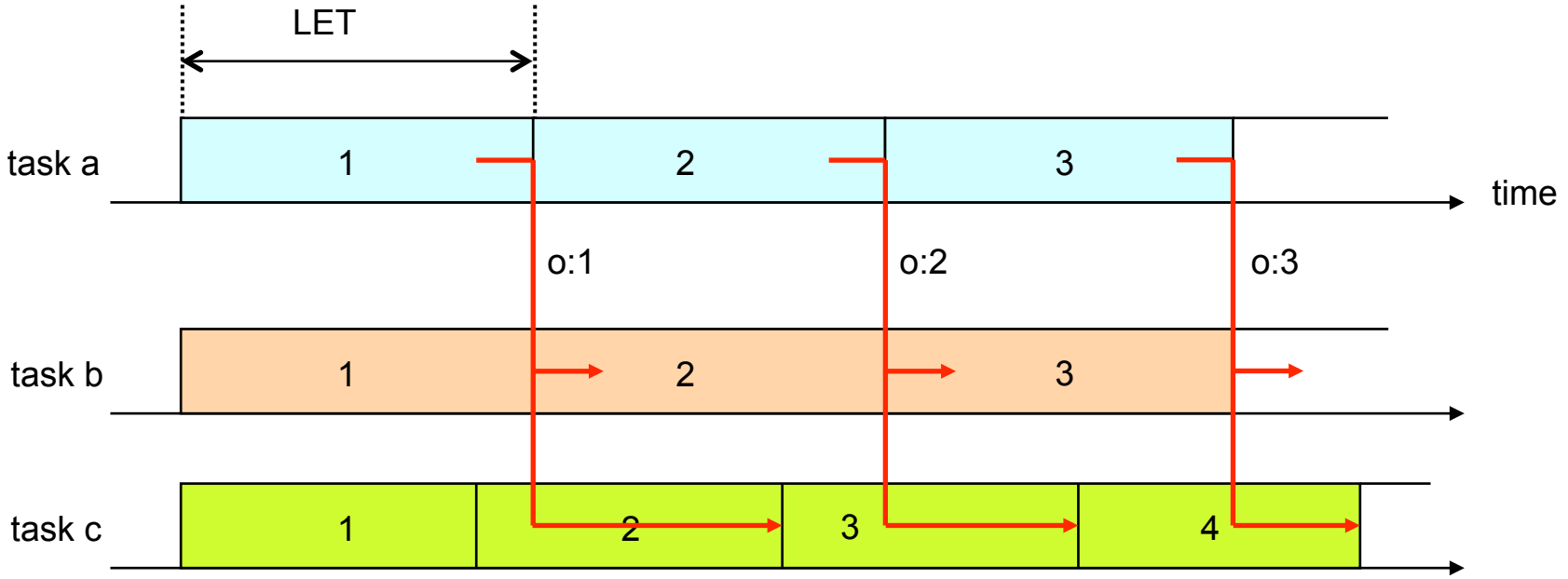


TDL extensions

Control engineering view: LET implies unit delays



Control engineering view: LET implies unit delays

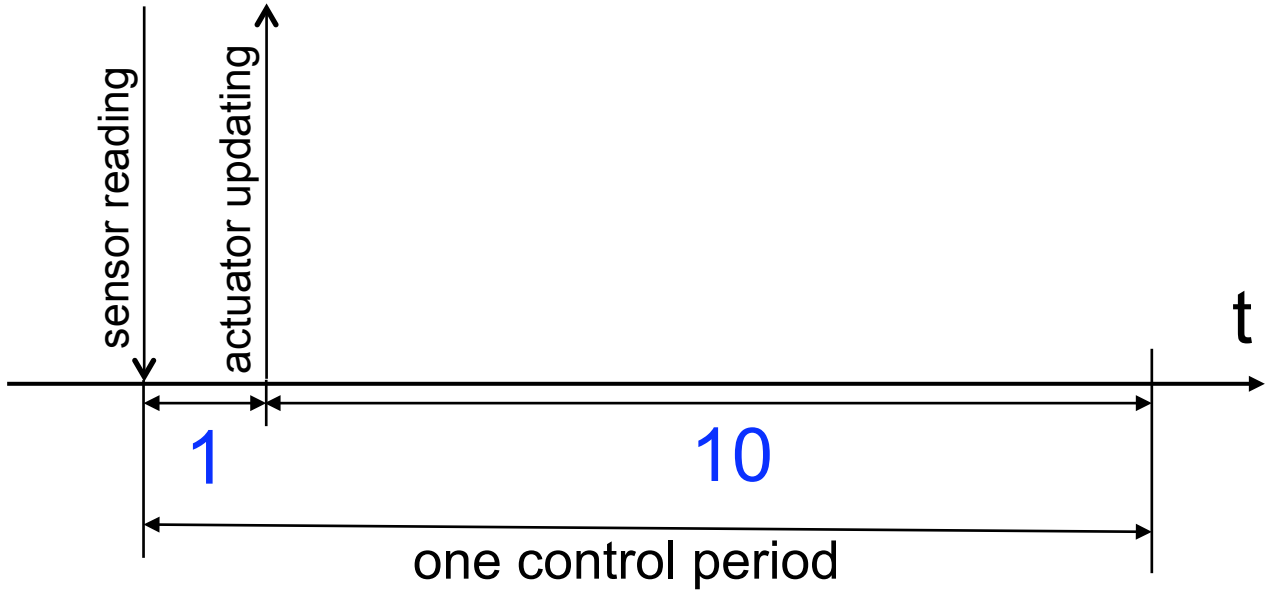


... a waste of time?

\+ determinism, composition, transparent distribution

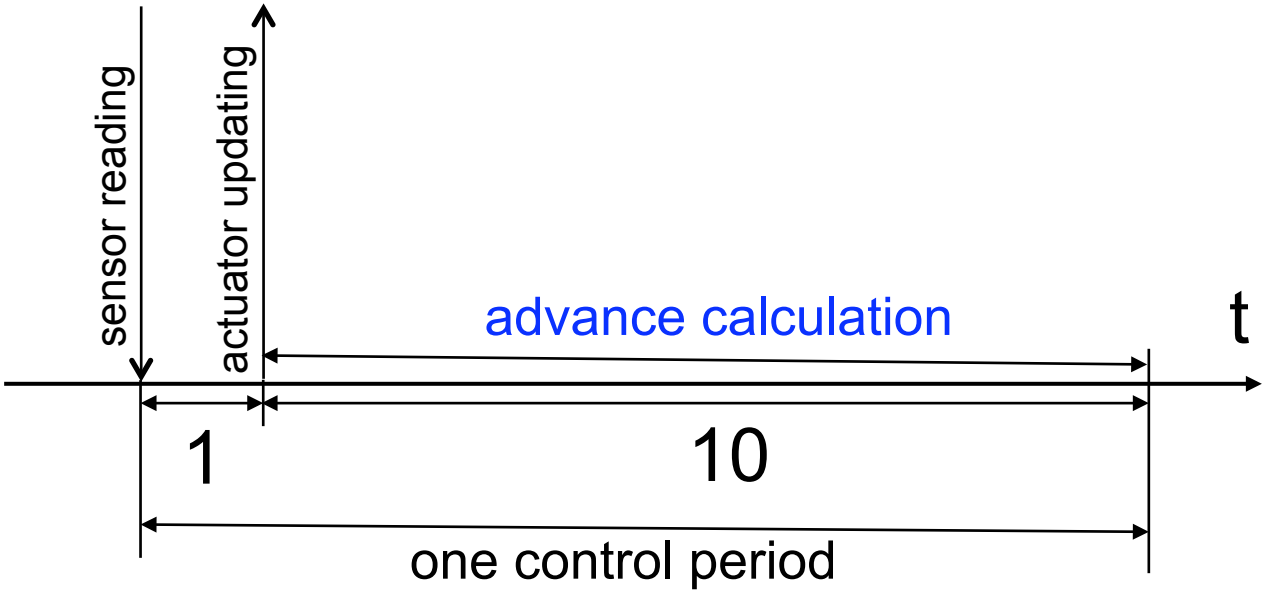
– contradicts conventional wisdom of control engineering

10:1 rule and advance calculation



- actuating as fast as possible after sensor reading
- the control period should be at least 10 times as large as the delay between reading the sensor and setting the actuator in order to get stable controller

10:1 rule and advance calculation



- the period after actuating can be used for advance calculations (eg, computing a polynomial) which might be necessary on slow CPUs

TDL support for 10:1 rule and advance calculation

- split a task execution in two parts

  (1) a fast step and

  (2) a slow step.
- Core idea: The fast step is considered to be executed in logical zero time. In other words, the fast step is executed synchronously by the E-Machine at the start of the LET of a task.
- The slow step is executed later but must be finished before the end of a task's LET.

TDL syntax for 10:1 rule and advance calculation

```
module M1 {

  sensor int s uses getS;
  actuator int a := 0 uses setA;

  task t {
   input int i;
   output int o;
   state M1State s;
   uses [release] fastStep(i, s, o); slowStep(i, o, s);
  }

  start mode main [period = 10ms] {
    task
      [freq=1] { t(s); a := t.o; }
  }
}
```

Status quo

- ready
  - TDL:VisualCreator (stand-alone or in Matlab®/Simulink®)
  - TDL:VisualDistributor (extensible via plugins; currently a plugin for FlexRay is available as product, together with plug-ins for various cluster nodes such as the MicroAutoBox, and Renesas–AES)
    The TDL:VisualDistributor is available as stand-alone tool or in Matlab®/Simulink® and provides the following features:
    - Communication Schedule Generator
    - TDL:CommViewer
    - automatic generation of all node-, OS- and cluster-specific files
  - TDL:Compiler
  - TDL:Machine for Simulink, mabx, AES, INtime, OSEK
  - multiple slot selection (decoupling of LET and period; eg, for event modeling)
  - harnessing existing FlexRay communication schedules (via FIBEX) for their incremental extension
  - TDL:VisualAnalyzer (beta; recording and debugging tool)
- work in progress
  - seamless integration of asynchronous events with TDL
  - 'intelligent' FlexRay parameter configuration editor
  - TDL:Machine for further platforms (AutosarOS, etc.)

Thank you for your attention!