Model Based Development of Embedded Control Software

Part 8: Transparent Distribution

Claudiu Farcas

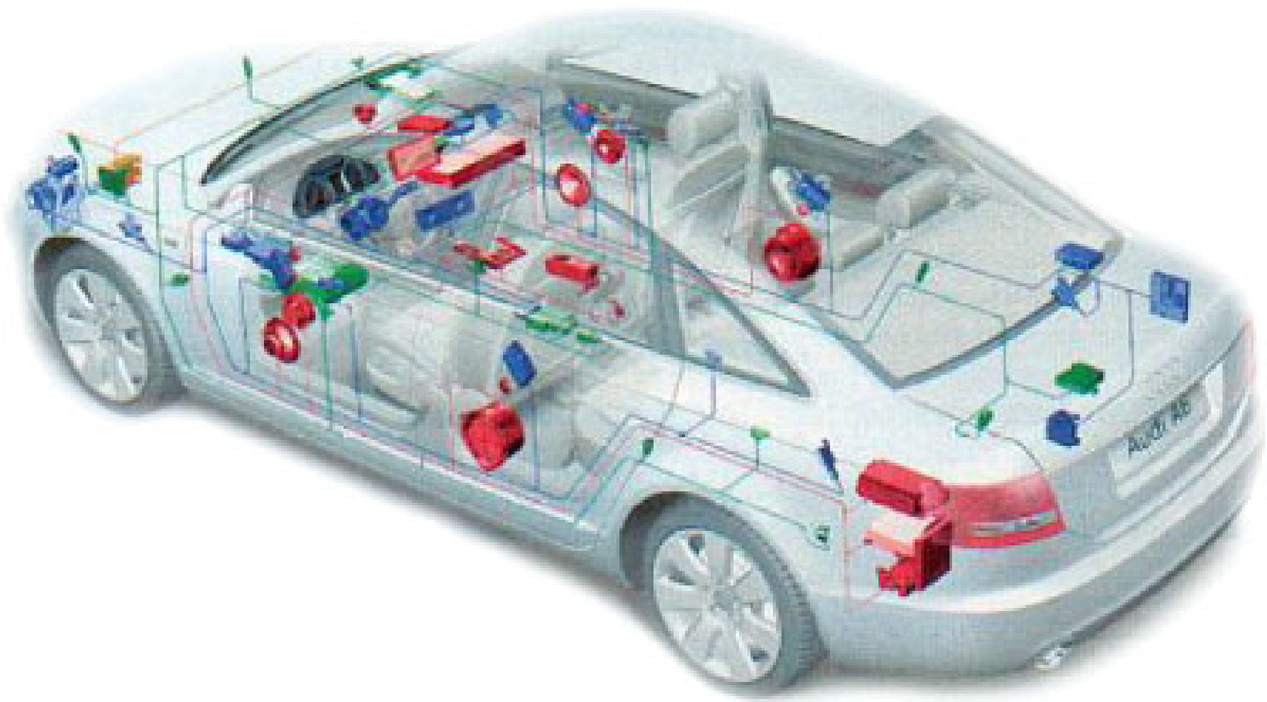
Credits: MoDECS Project Team, Giotto
Department of Computer Science
cs.uni-salzburg.at

UNIVERSITÄT SALZBURG

Overview

- Motivation
- Transparent Distribution
- Bus Schedule Generation Tool
- Module stubs and TDL Run-time Environment

Motivation



MOST-Bus
Multimedia subsystems

CAN-Bus
powertrain and body electronics
comfort / climatronic

Benefits from distribution:

- Scalability (CPU, IO)
- Low-cost components
- Fault Tolerance

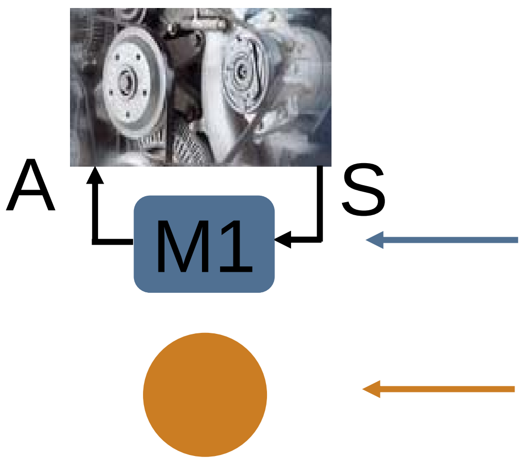
UNIVERSITÄT SALZBURG

The advantages of transparent distribution

- the **functional *and* temporal behavior of a system is the same no matter where a component is executed**
- developer's perspective:
  NO difference between local and distributed execution of components
- OEM-supplier perspective:
  the components can be developed independently

UNIVERSITÄT SALZBURG

Transparent distribution in a nutshell



plant

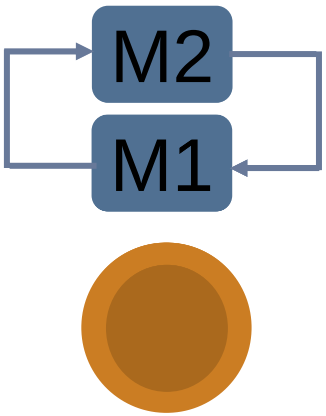
software component
(hard real-time control system)

computing platform,
for example, MPC555+OSEK

UNIVERSITÄT SALZBURG

Transparent distribution in a nutshell

component M2 added later,
if required even at run-time

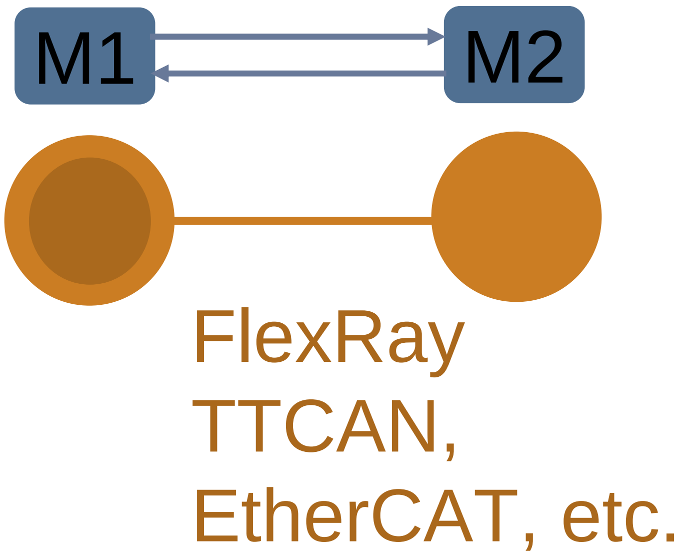


exactly defined
communication semantics
(TDL programming model)

UNIVERSITÄT SALZBURG

Transparent distribution in a nutshell

- deterministic timing and communication behavior
- independent of the computing and communication platform
  => portability through automatic code generation
  and run-time environment



What do we need to achieve transparent distribution?

- **abstractions for embedded software** that
  - ignore the platform details, but
  - capture the essence of embedded hard-real-time systems

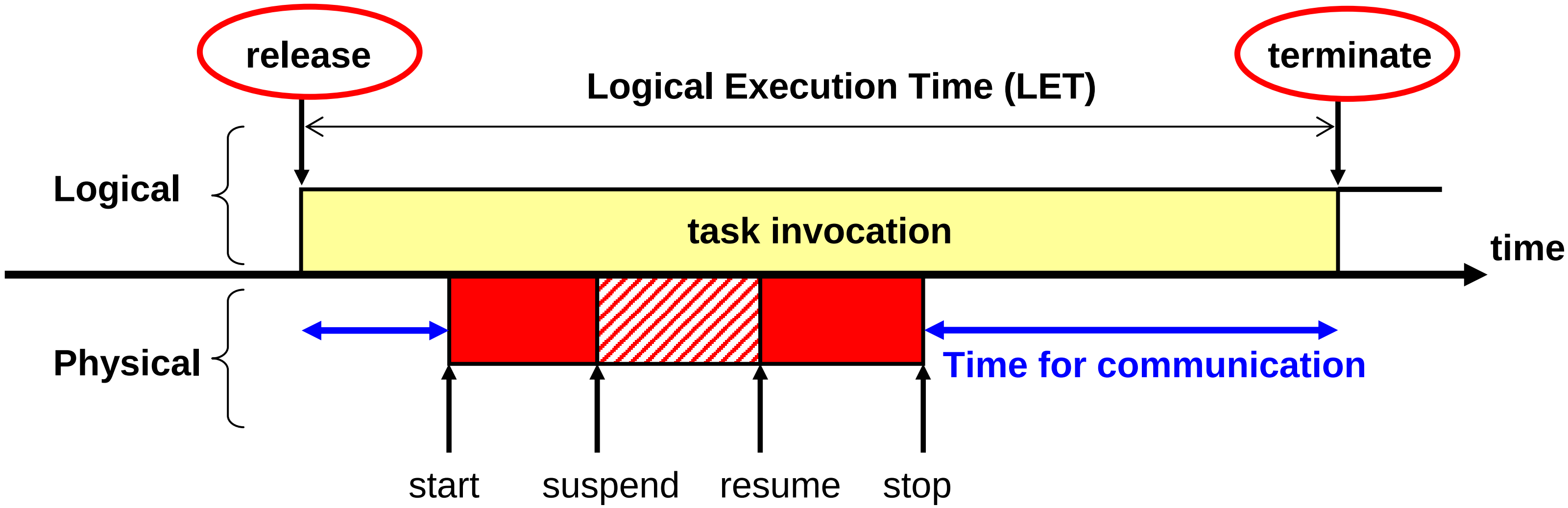
=> Timing Definition Language (TDL)

- run-time environment that
  - efficiently executes programs
  - is flexible enough to allow dynamic changes (adding/replacing/moving of components)

=> TDL run-time environment

UNIVERSITÄT SALZBURG

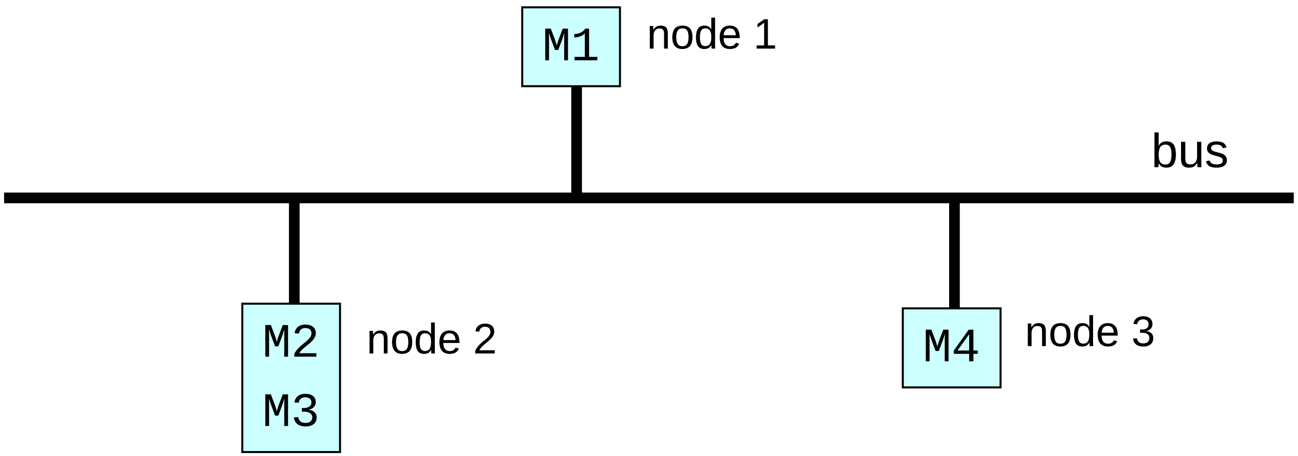
The Giotto/TDL core abstraction: **LET**



- LET means that the observable temporal behavior of a task is independent from its physical execution.
- we gain crucial software properties: determinism, portability, composability

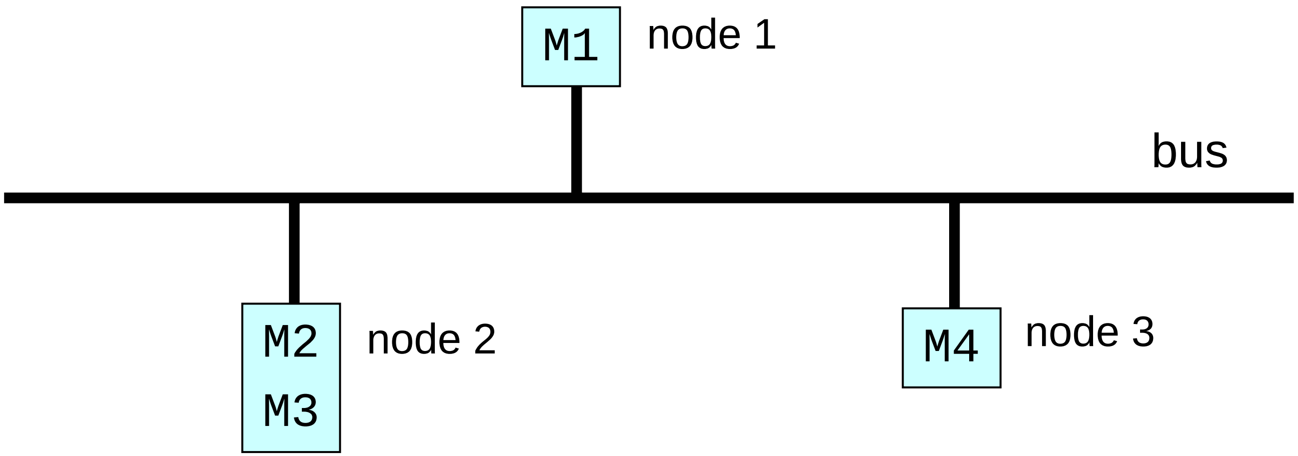
UNIVERSITÄT SALZBURG

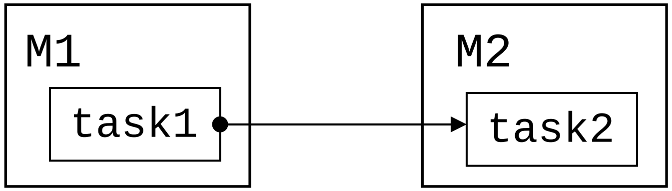
Sample distribution of TDL components M1, M2, M3, M4



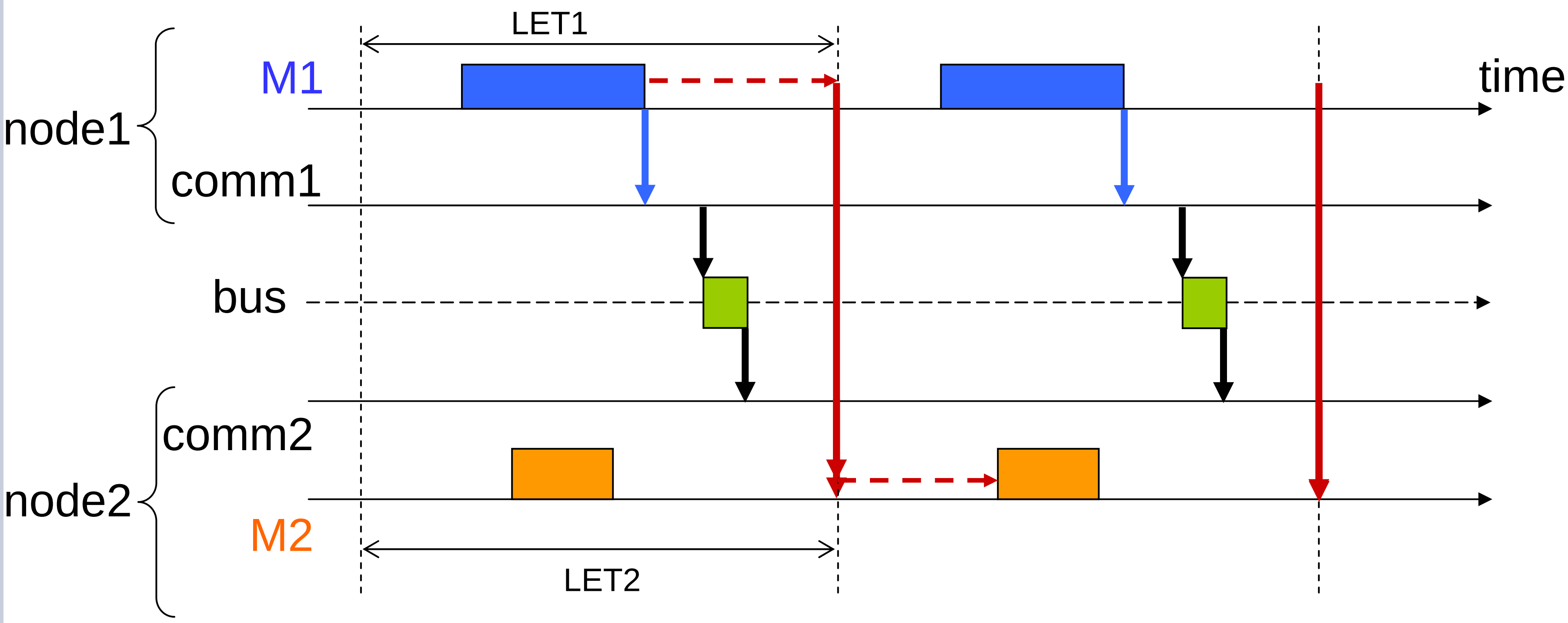
Unit of distribution:	TDL module
Behavior:	as if executed locally
Communication:	via broadcast (bus)
Medium access control:	TDMA (time-slotting)
Cooperation model:	Producer-Consumer (Push)

Sample distribution of TDL components M1, M2, M3, M4



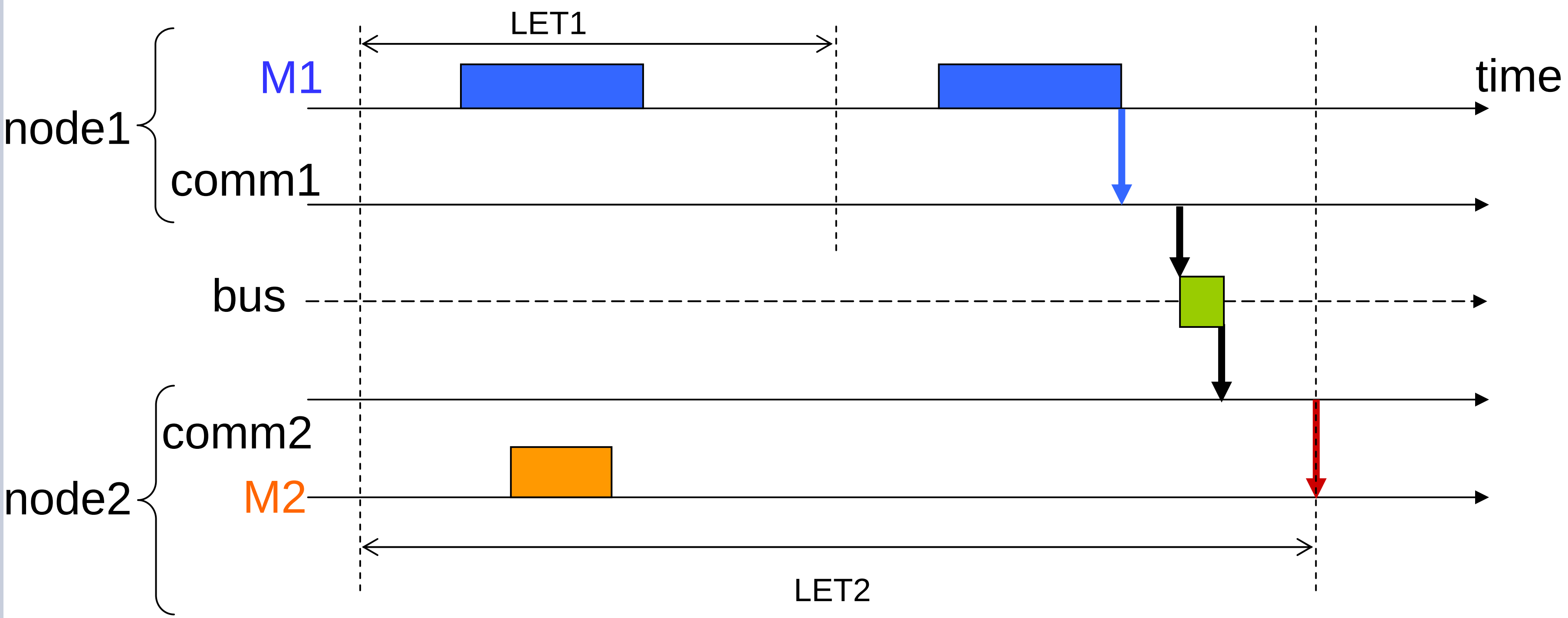


The purpose of the TDL-Comm layer



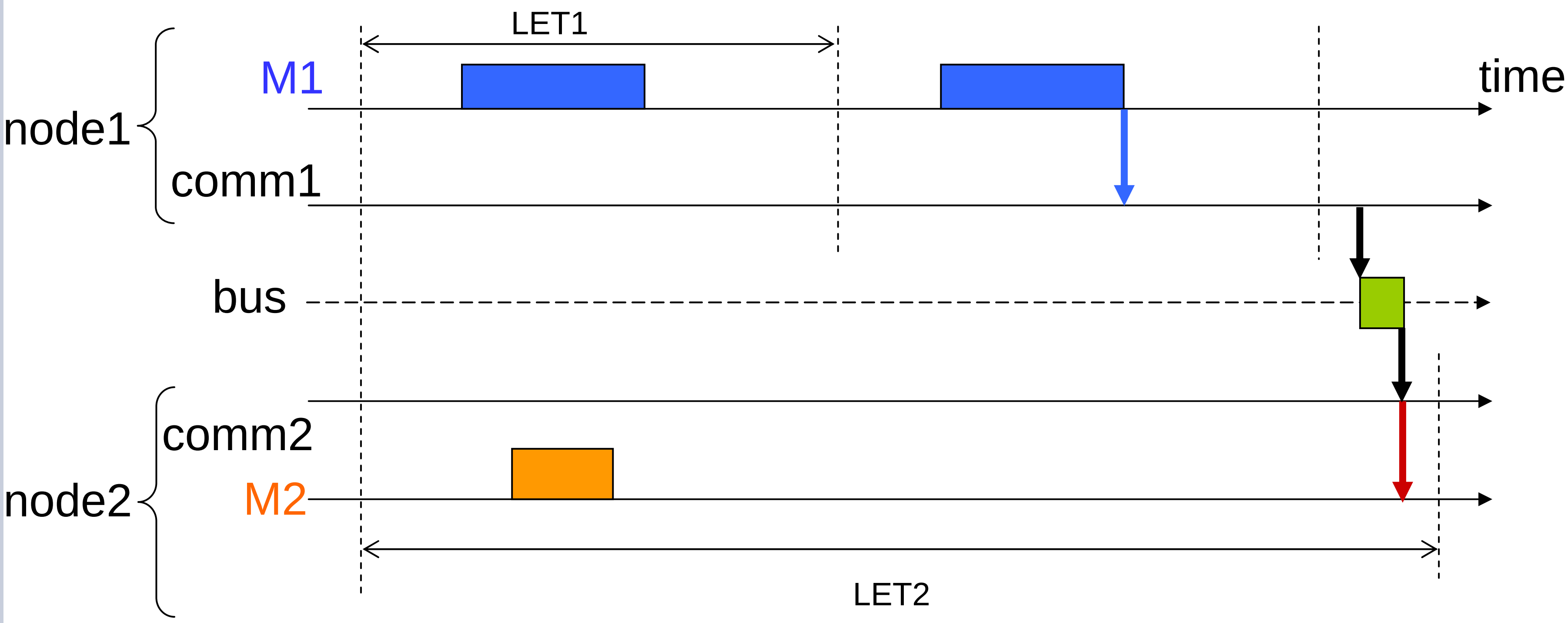
- messages are sent according to a bus schedule (TDMA)

Optimization I



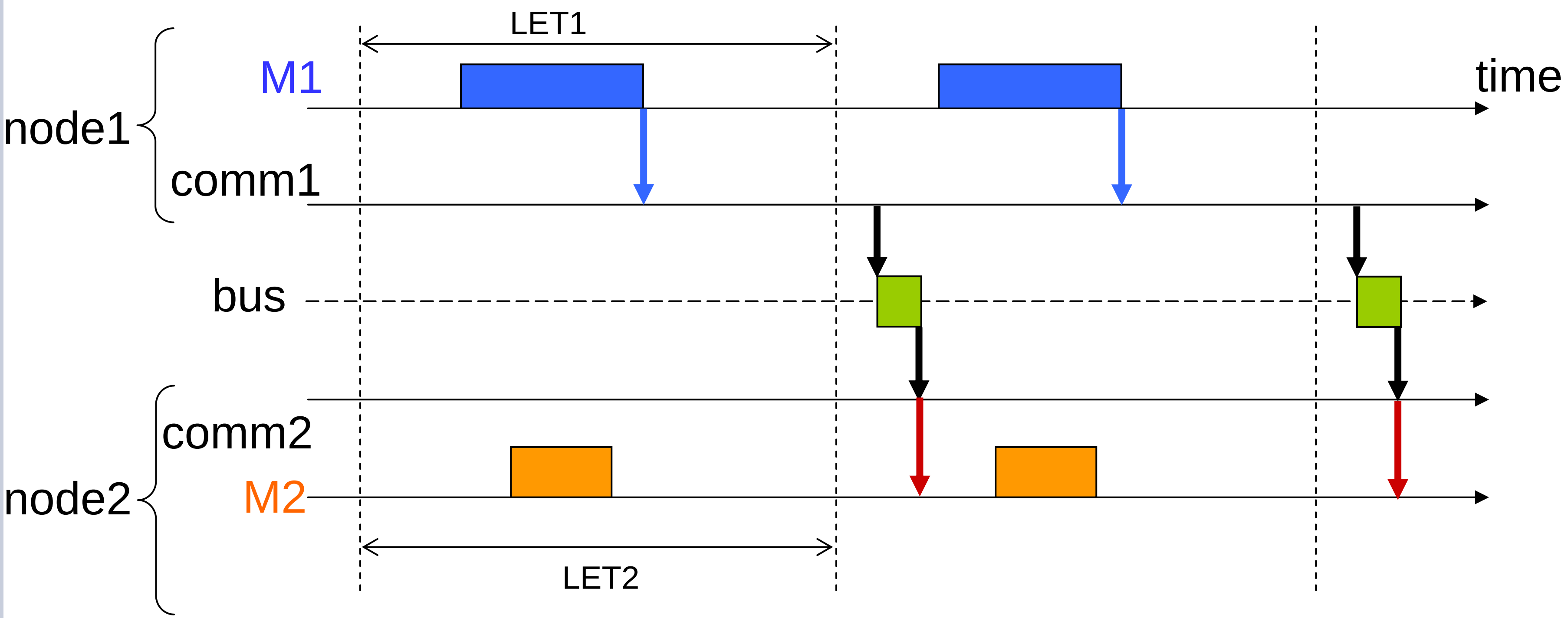
- if consumer runs slower e.g. by a factor of 2
- redundant message are avoided
- saves bandwidth

Optimization II



- if the consumer needs variable later than the producer's LET
- can lead to better bus utilization

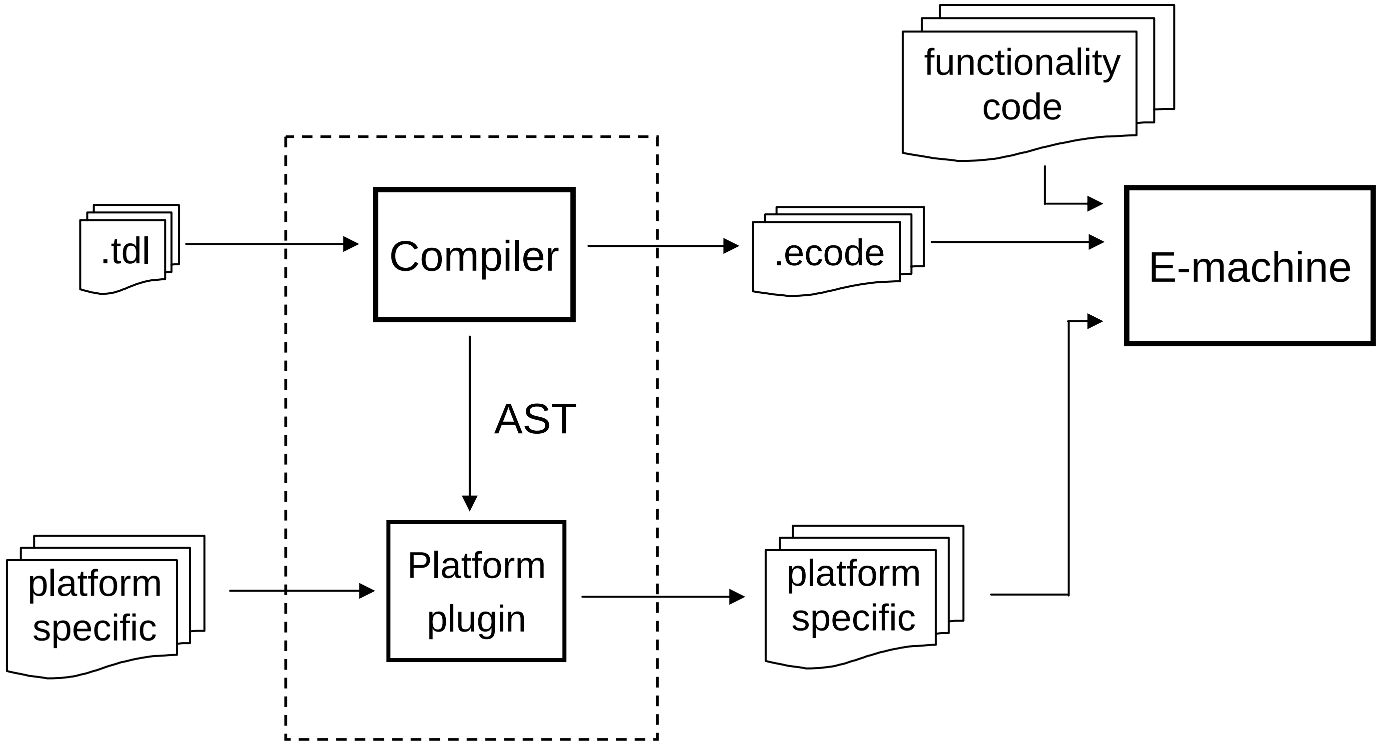
Optimization III



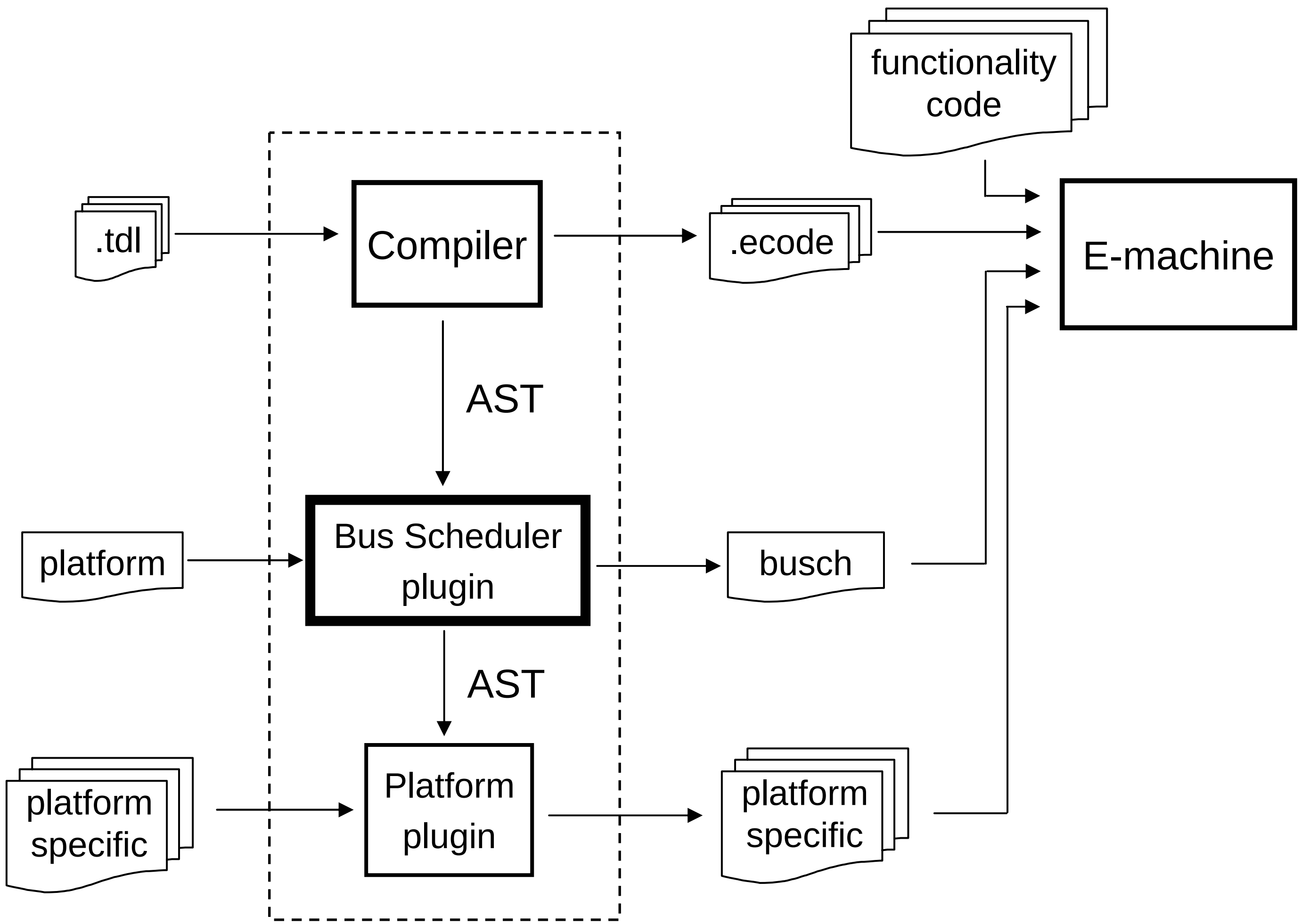
- the release of the receiver can be delayed until the message with the input variable is received

Bus Schedule Generation Tool

Tool chain – Single Node setup



Tool chain – Distributed system



What Does the Tool Do?

It generates a global bus schedule file, which contains the following information:

- Which node has to send a packet and when.
- Which nodes have to receive a packet and when.
- The content for bus packets (a corresponding datagram, which has one or more items/variables).

What Does the Tool Need?

- TDL modules
- Platform description file
  - module to node assignment
  - physical bus properties (e.g., bus frequency, protocol overhead, inter frame gaps, min/max payload)

The tool automatically detects:

- Who has to communicate with whom.
- Which messages are needed in a communication cycle (bus period).

Who has to Communicate with Whom

Results a set of messages.

- A message has: a Sender port, one or more Receiver ports, size.
- A Sender or Receiver port has: unique qualified identifier, period, and WCET.

- Senders: sensors, task output ports.
- Receivers: actuators, task input ports, guard arguments.

Messages Needed in a Bus Period

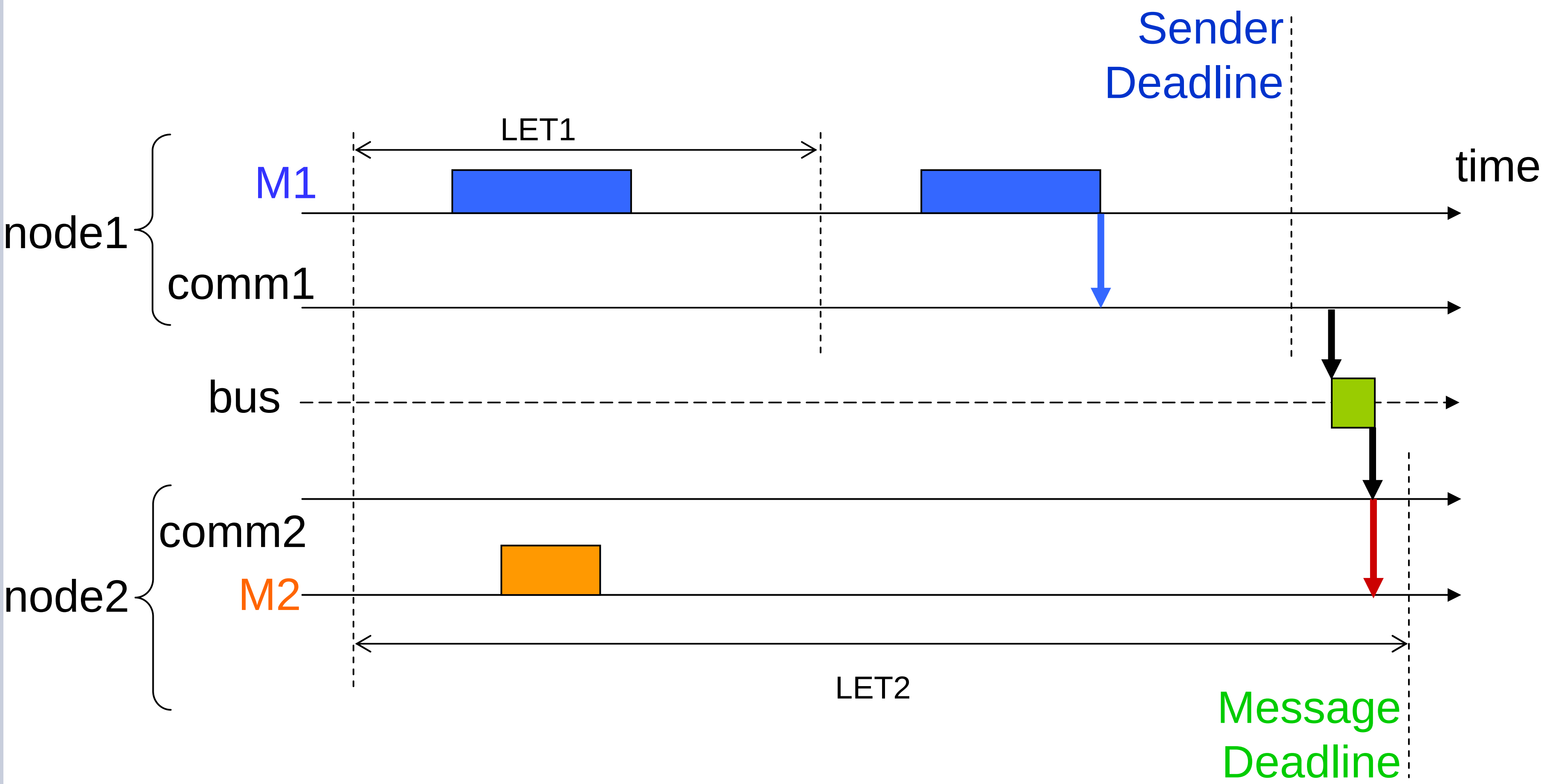
Results a set of message instances, with individual timing constraints:

- Release Offset
- Deadline

- Basic Producer-Consumer:
  - Send messages with the frequency of the Sender:
  - Message deadline = sender deadline.
  - BusPeriod = LCM(Sender.period)

- Optimized Producer-Consumer:
  - Send messages only when they are needed by the Receivers.
  - Message deadline depends on the optimization (e.g., = receiver release time).
  - BusPeriod = LCM(Sender.period, Receiver.period)

Message Deadline in Optimization II



UNIVERSITÄT SALZBURG

Message Scheduling

Current approach:

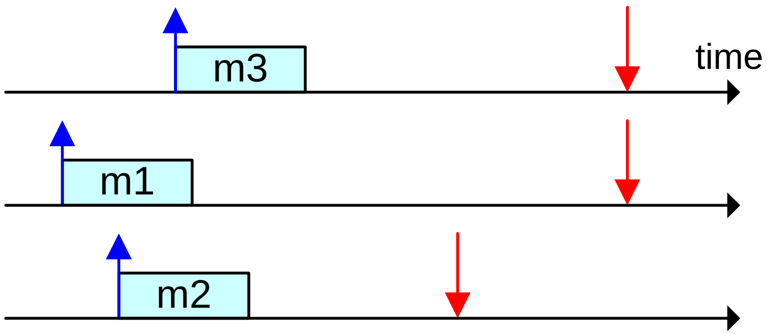
- 2 Steps scheduling:
  - schedule first the messages.
  - schedule then the tasks with deadlines constraints from messages.

- Optimizations:
  - We build bus schedulers which allow more flexibility for the task scheduler.
  - We try several bus schedulers and get feedback from the TSC for tasks.
  - Schedule individual messages or merge messages sent from the same node

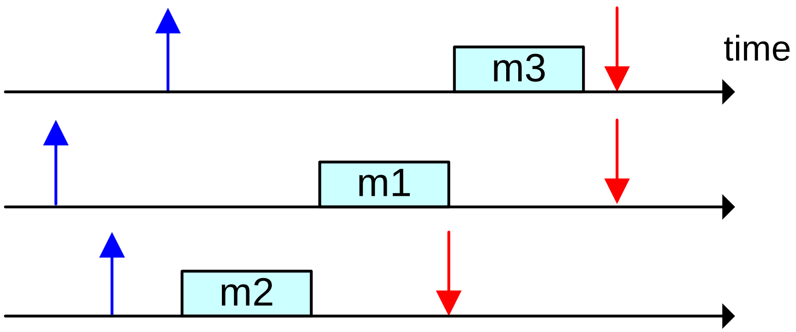
Scheduling Algorithms

- Heuristic schedule - Latest Deadline Last (variant of Reversed EDF)
  - Schedule messages as late as possible
  - May fail even when a schedule exists
- Optimal schedule
  - Branch and bound search
  - Exponential complexity in the worst case.

Latest Deadline Last - Example



Released messages {m1, m2, m3}



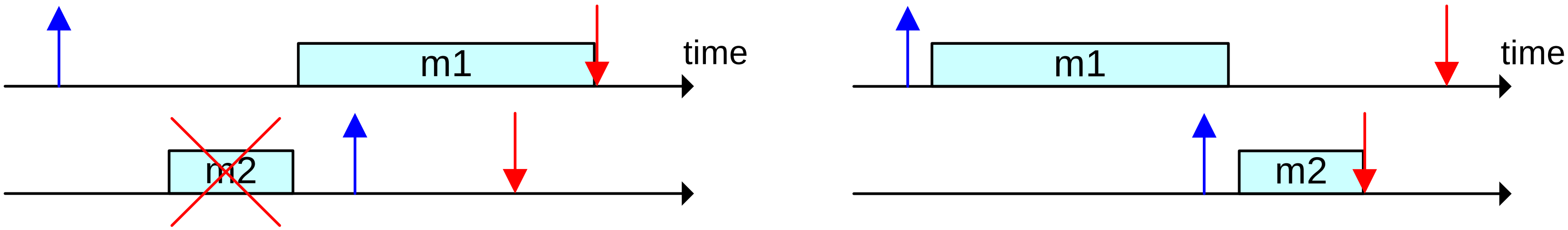
LDL scheduling {m2, m1, m3}

UNIVERSITÄT SALZBURG

Latest Deadline Last

- Sorts the list of messages by:
  - Key1 = message deadline
  - Key2 = message release time
  - Key3 = sender deadline
- Bus Scheduler is non-preemptive and just schedules the messages in the resulted order.
  - Starts from the end of the Bus Period
  - Merges messages if they have to be sent by the same node, and are adjacent.

Search Scheduler - Example



LDL scheduling failure {m2, m1}

Search scheduler {m1, m2}

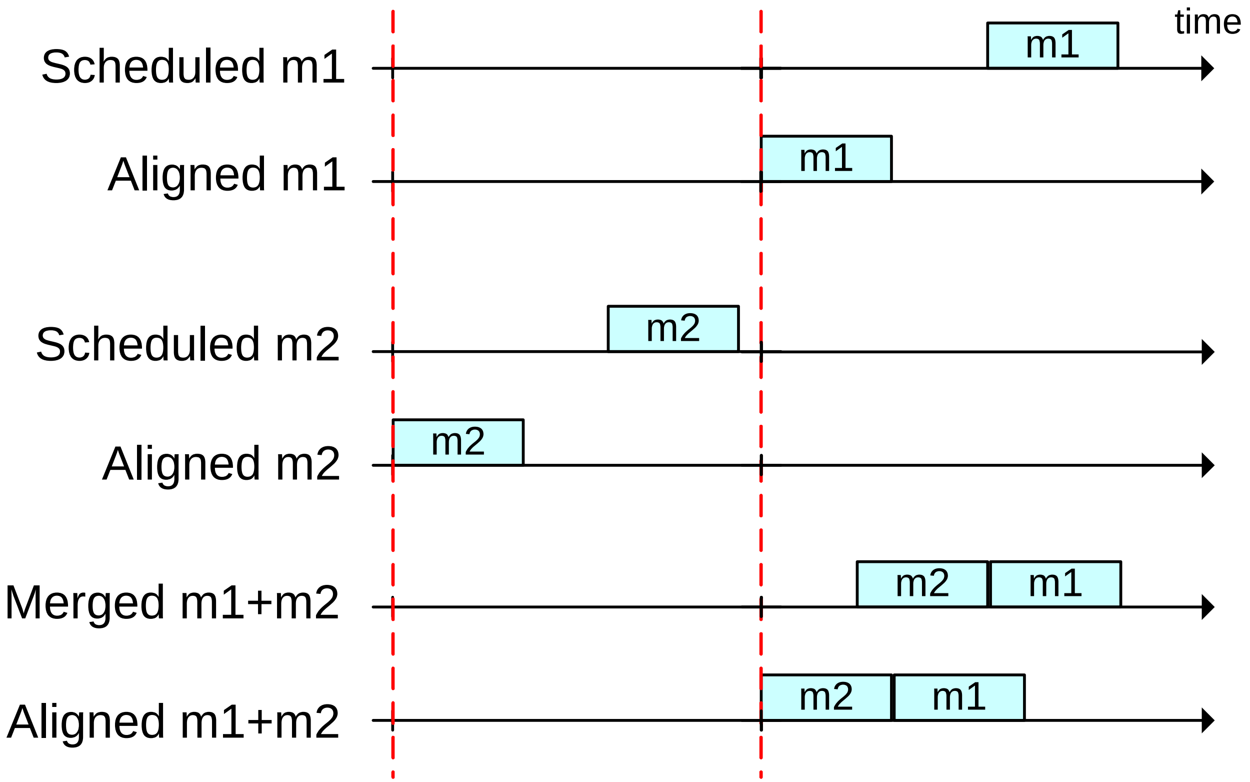
UNIVERSITÄT SALZBURG

Bus Properties as Constraints

- Relevant for:
  - Merging messages (min/max payload)
  - WCCT (Bps, protocol overhead)
  - Time alignment (inter frame gaps, clock resolution)
  - Control packets (time synchronization)

- Clock Resolution:
  - TDL time unit is microsecond (us).
  - Different platforms have a given clock resolution (e.g., 1ms or 100us).
  - Bus communication is computed in microseconds or even nanoseconds.

Merging Messages and Clock Resolution



UNIVERSITÄT SALZBURG

Measurements

Metrics relevant for efficient bus utilization:

- Throughput
- Bus utilization
- Average data efficiency
- Maximum and average sending rates
- Maximum and average receiving rates

Metrics relevant for flexibility in task scheduling:

- Minimum and average release-send intervals
- Minimum and average relative release-send intervals

Module stubs and the TDL E-Machine

TDL run-time system: E-Machine

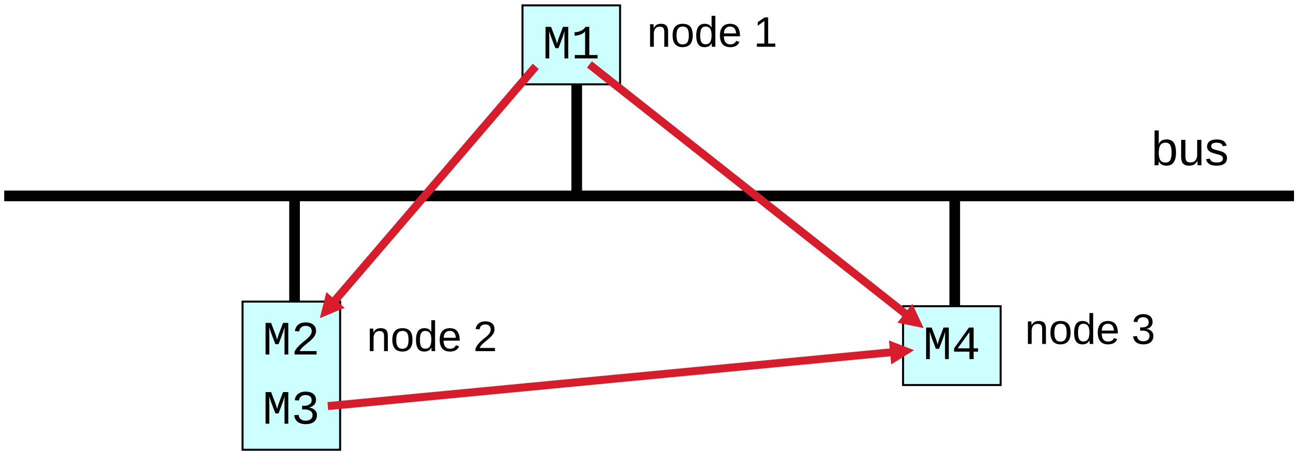
- runs on each computing node
- executes E-code instructions at logical time instances
- implementation is platform dependent (OSEK, InTime, RTLinux, Java)
- it is **fast and lightweight (e.g. 13 KB** for the OSEK E-machine).

- supports three kinds of module executions:
  - local,
  - push, and
  - stub.

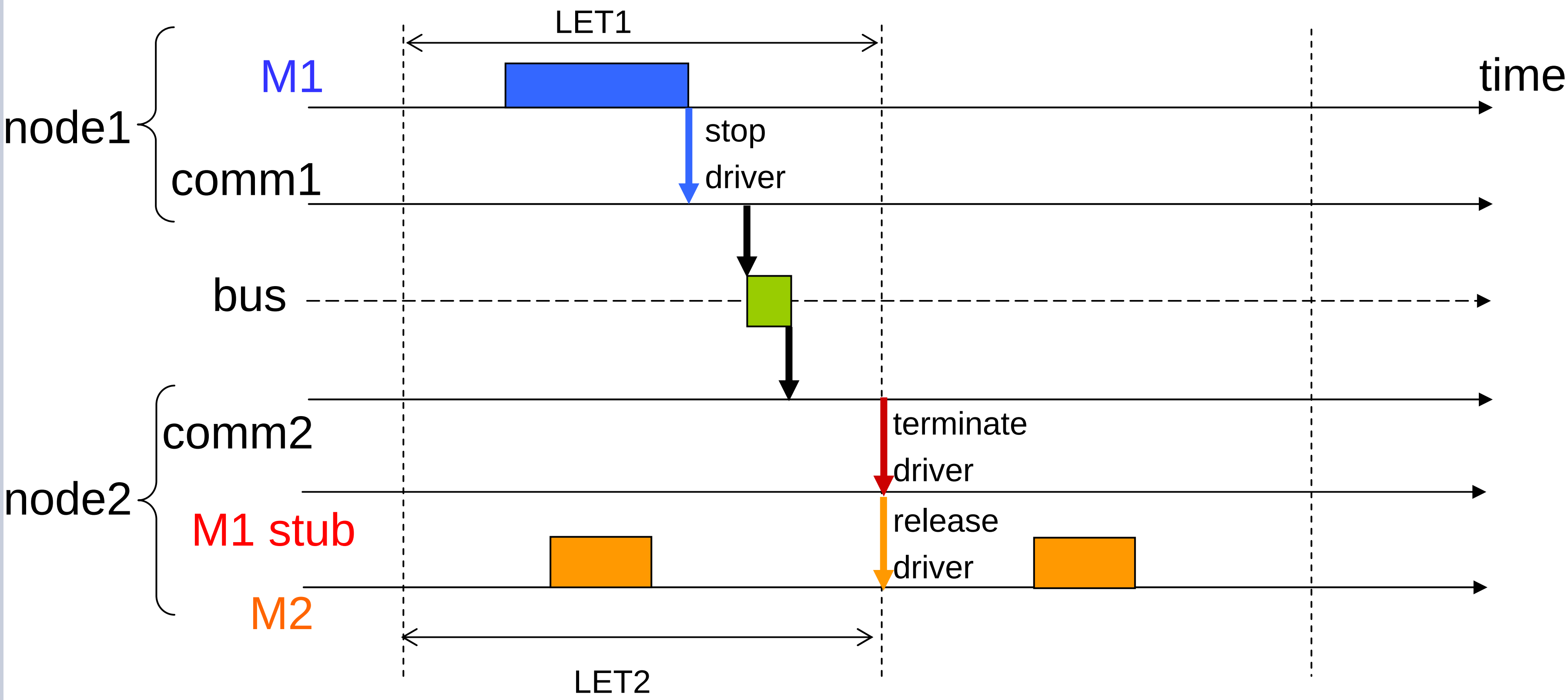
Module import relationships

M2 imports M1

M4 imports M1, M3



E-Machine actions

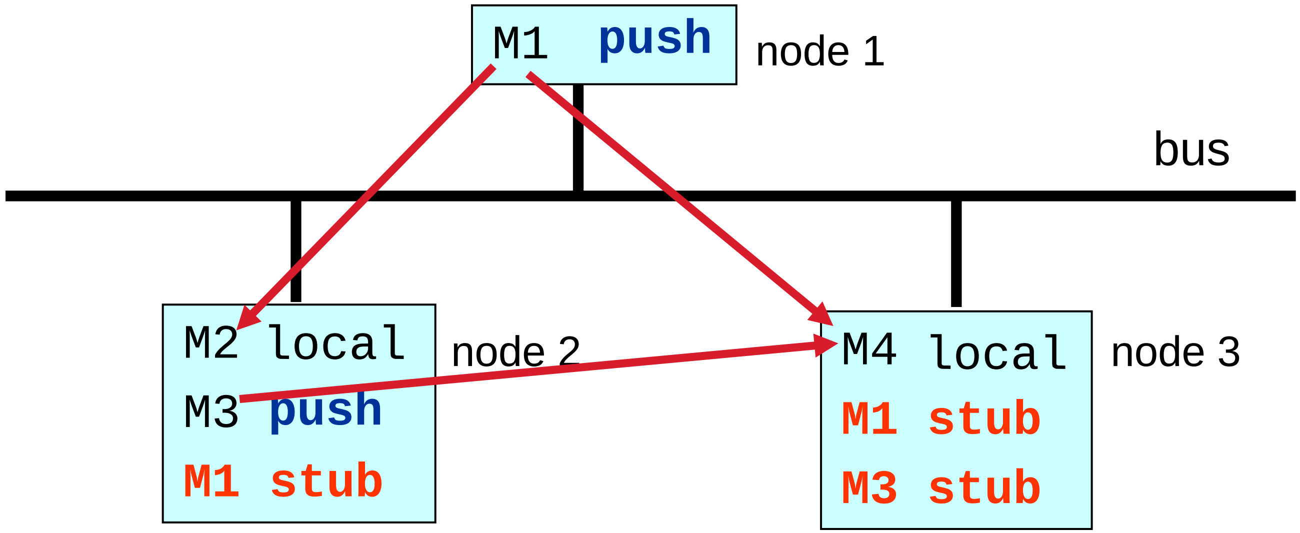


UNIVERSITÄT SALZBURG

Module execution attributes

M2 imports M1

M4 imports M1, M3



UNIVERSITÄT SALZBURG