embeddedworld
Exhibition&Conference
... it´s a smarter world

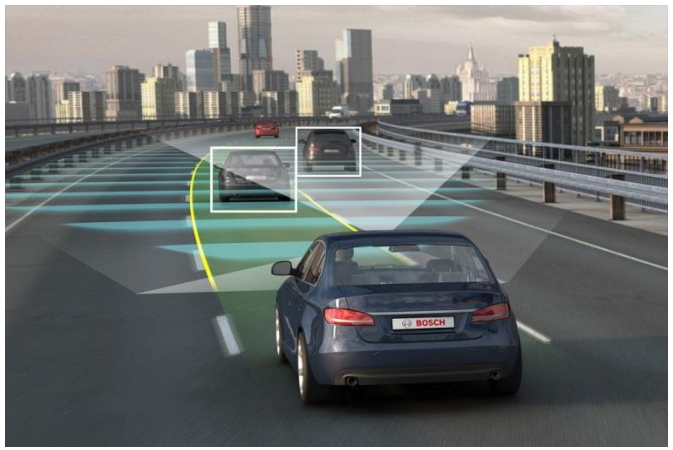
h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

Safety Architectures on Multicore Processors – Mastering the Time Domain

Thomas Barth (h-da)
Prof. Dr.-Ing. Peter Fromm (h-da)

Challenge

- Safety functions become more and more complex, e.g. autonomous driving
- Growing importance of x-by wire
- Former comfort functions become safety relevant, e.g. load control of a forklift truck
- Hardware systems become more complex, e.g. discrete redundancy is implemented on single multicore controllers



Content

- Problem Statement
- Timing Supervision
  - Alive Monitoring
  - Deadline Monitoring
  - Control Flow Monitoring
- Error Escalation
  - External Watchdog Devices
  - SmartPower – Last line of defense
- Conclusion / Summary / Discussion

Project: Future Technology Multicore

Supported by:

Federal Ministry
for Economic Affairs
and Energy

on the basis of a decision
by the German Bundestag

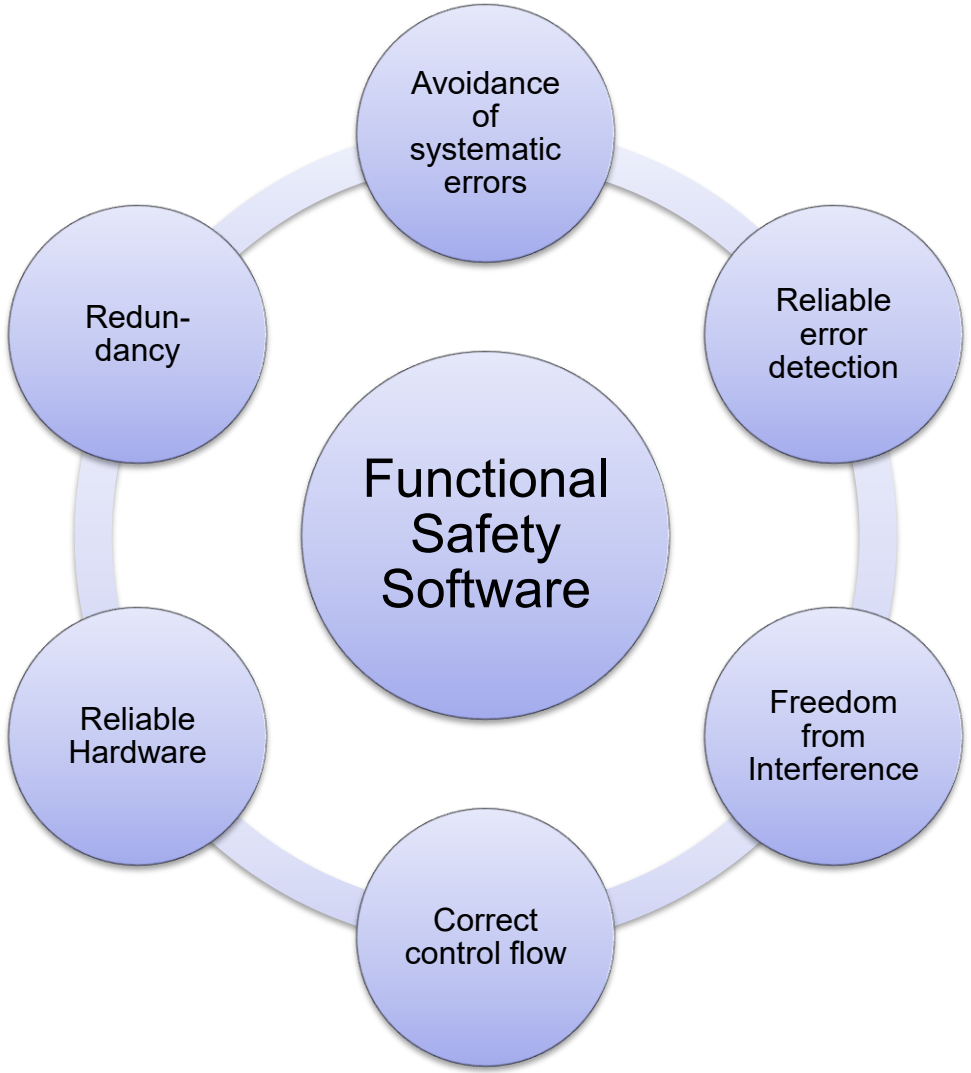
h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES

fbeit
FACHBEREICH ELEKTROTECHNIK
UND INFORMATIONSTECHNIK

FZI

HIGHTEC

Big Picture Embedded Safety

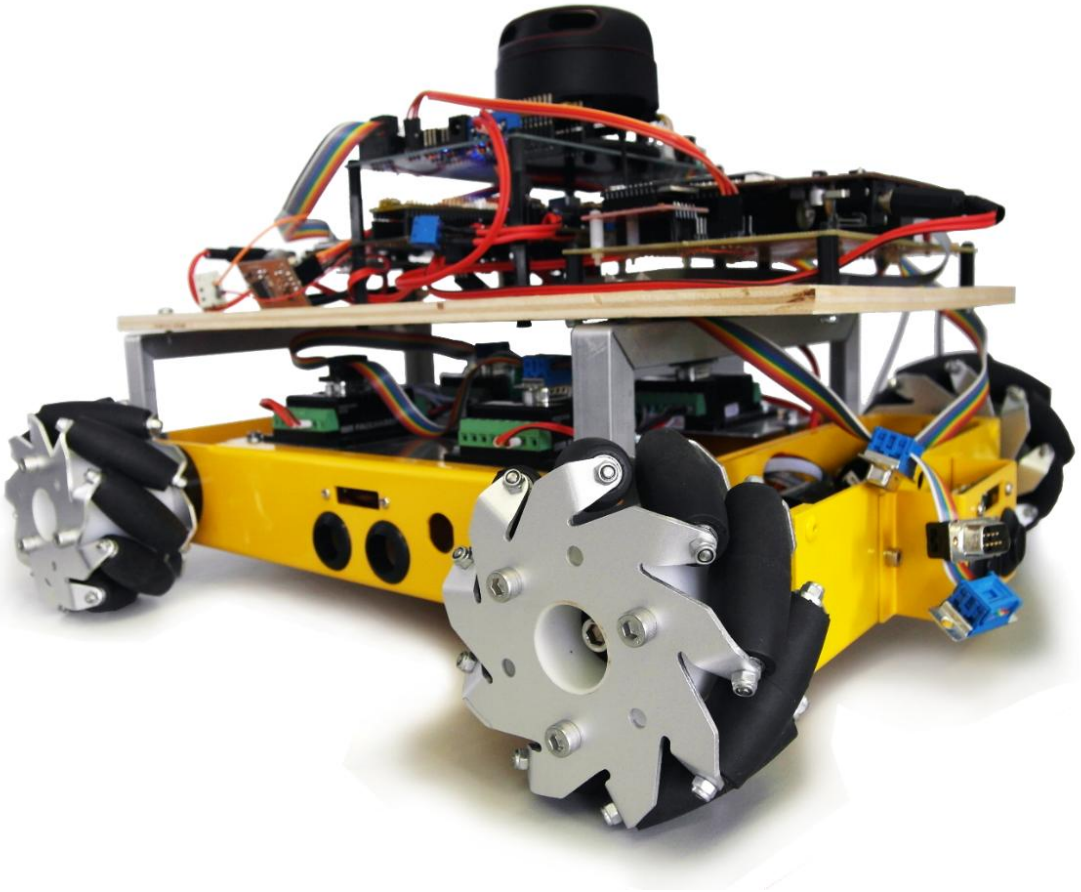


The safety function must be executed as specified under all condition.

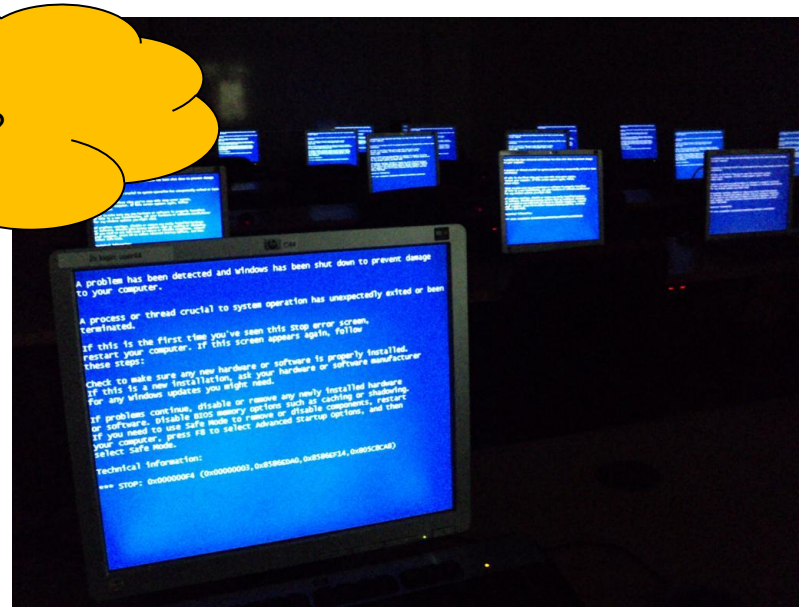
Fail safe: In case of sever error conditions, the system must come to a safe state.

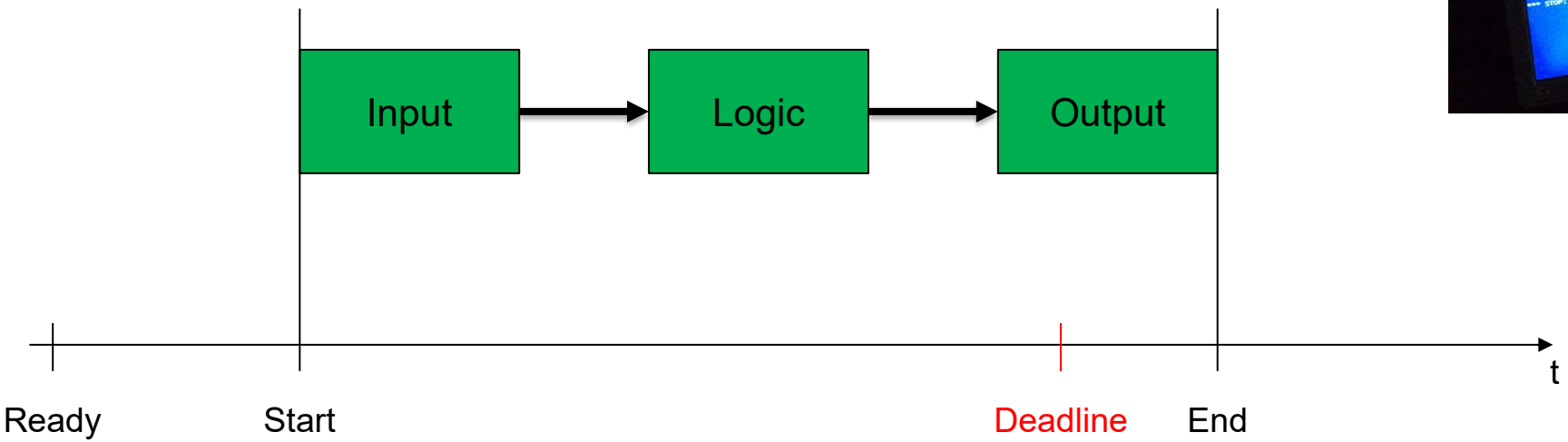
Fail operational: In case of sever error conditions, the system must continue to operate (out of scope)

Problem Statement – Safety Function Example: Drive by wire

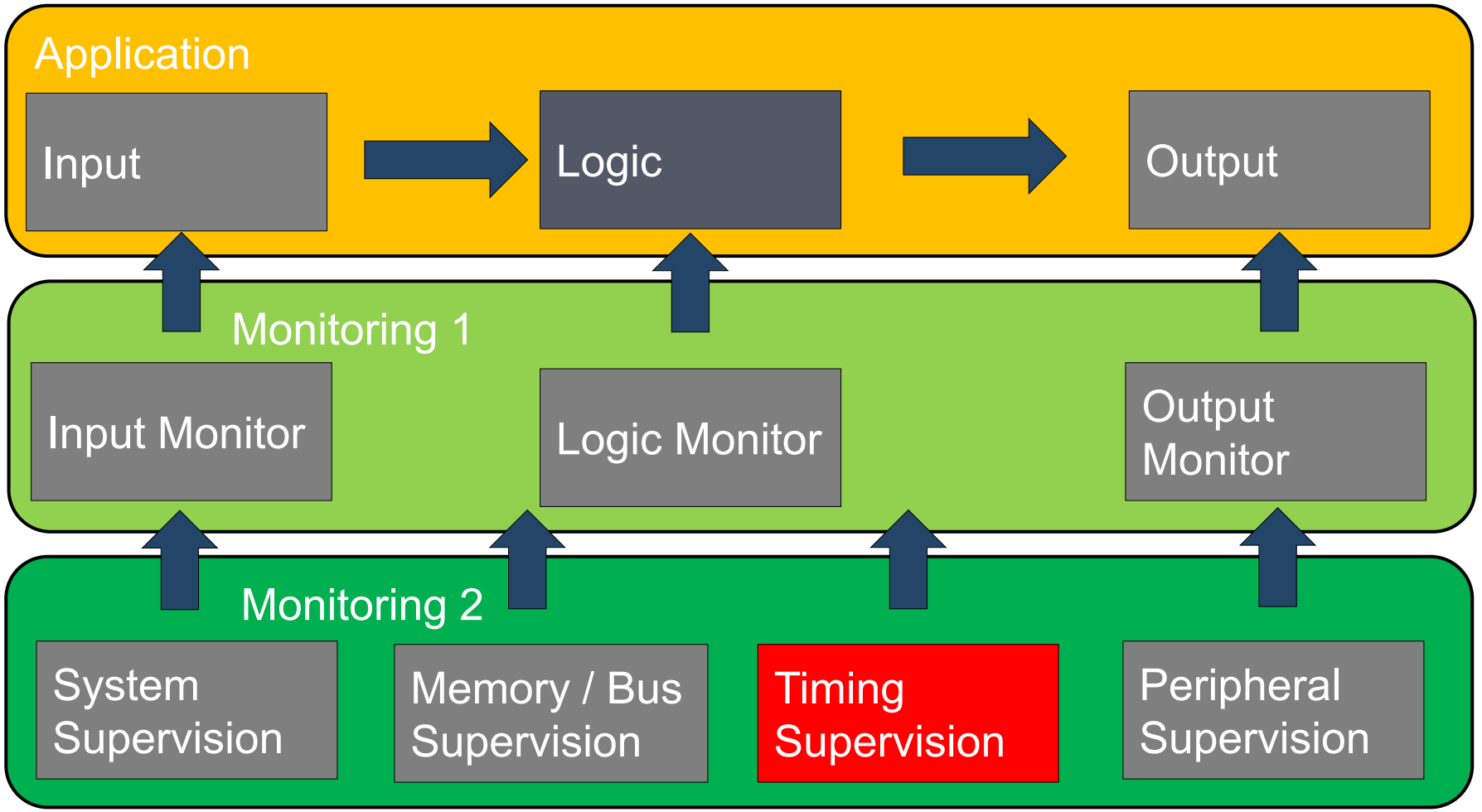


Problem Statement – Timing violations





Typical 3 Layer Architecture



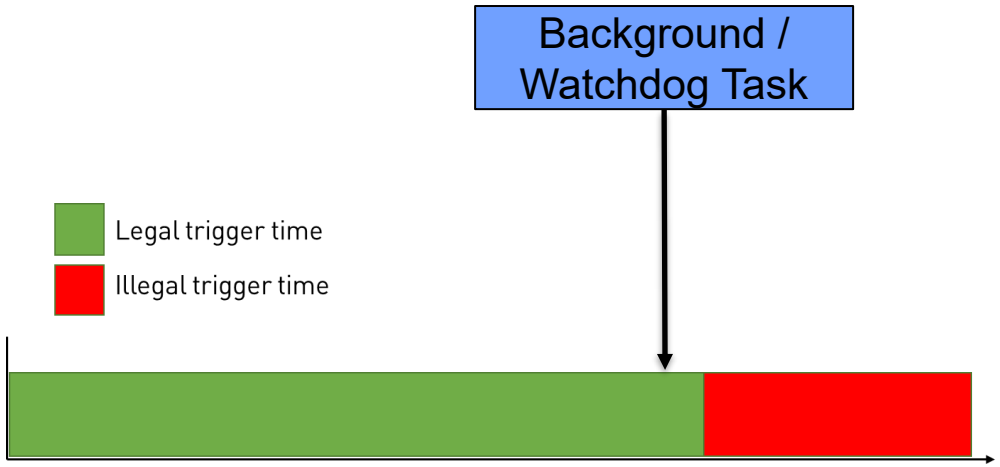
Timing Supervison

In most safety standards, the following concepts are mentioned

- Alive monitoring - Is the system up and running?
- Deadline monitoring - Are timing constraints like WCET met?
- Control flow monitoring - Is the process executed in a valid sequence?

Alive Monitoring – Watchdog Timer

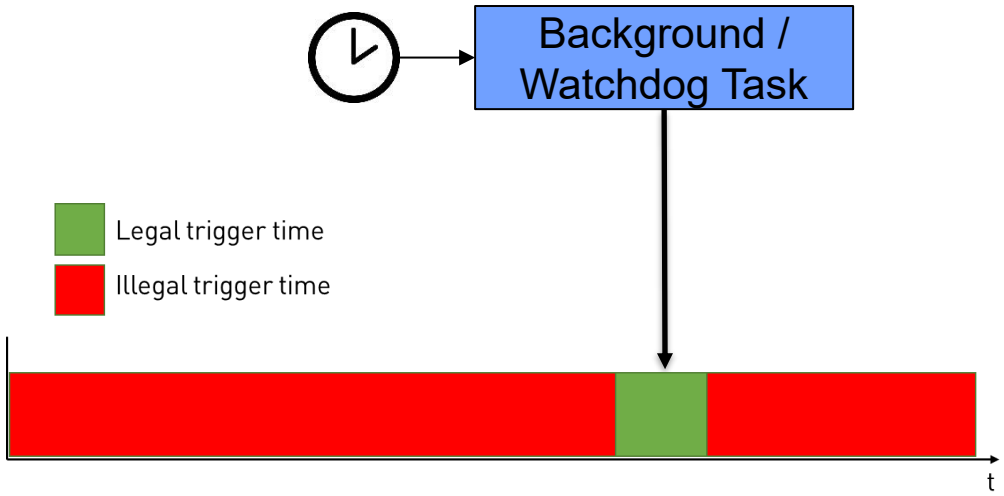
- Common approach: Watchdog timers



- Drawback: Only checks if the watchdog is triggered, not if the time base is correct.

Alive Monitoring – Window Watchdog

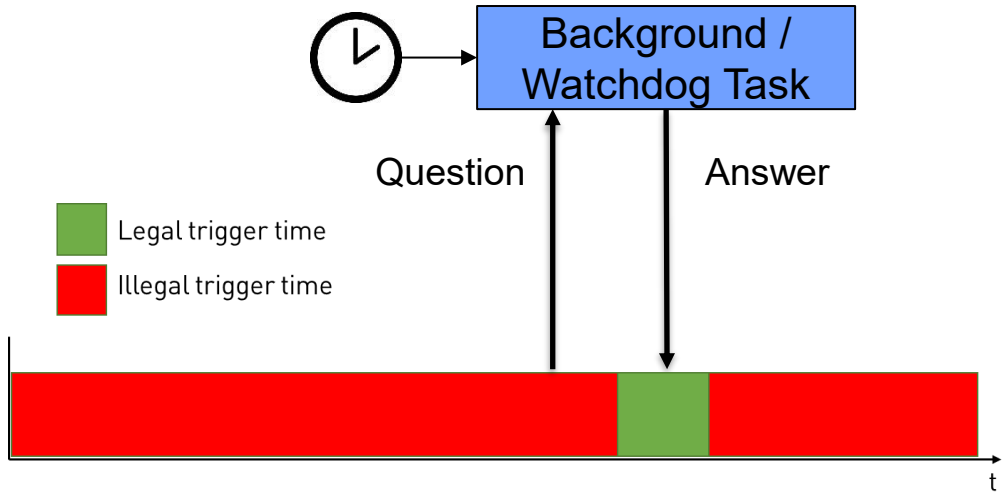
- Window watchdogs introduce a time window in which the watchdog has to be triggered



- With window watchdogs it becomes possible to validate that software is executed with the correct time base.

Alive Monitoring – Functional Watchdog

- Functional watchdogs add a certain “logical” complexity.
- A common functional watchdog is the question & answer watchdog.



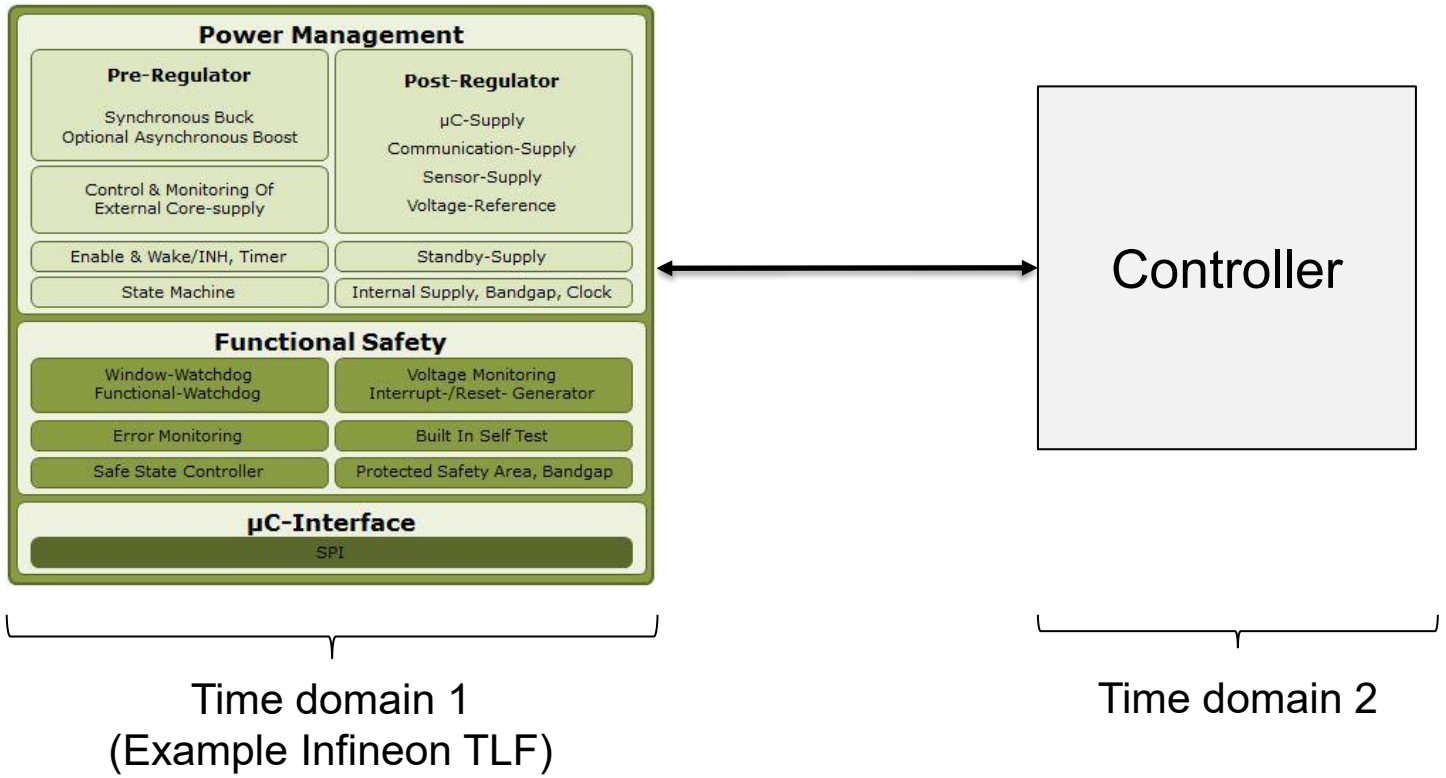
Question	Answer
0x01	0xff
0x10	0xb0
0x40	0xe9
0xf0	0xa6

- The question can be routed trough vital components of the system to verify that they are operational

Alive Monitoring - Structure

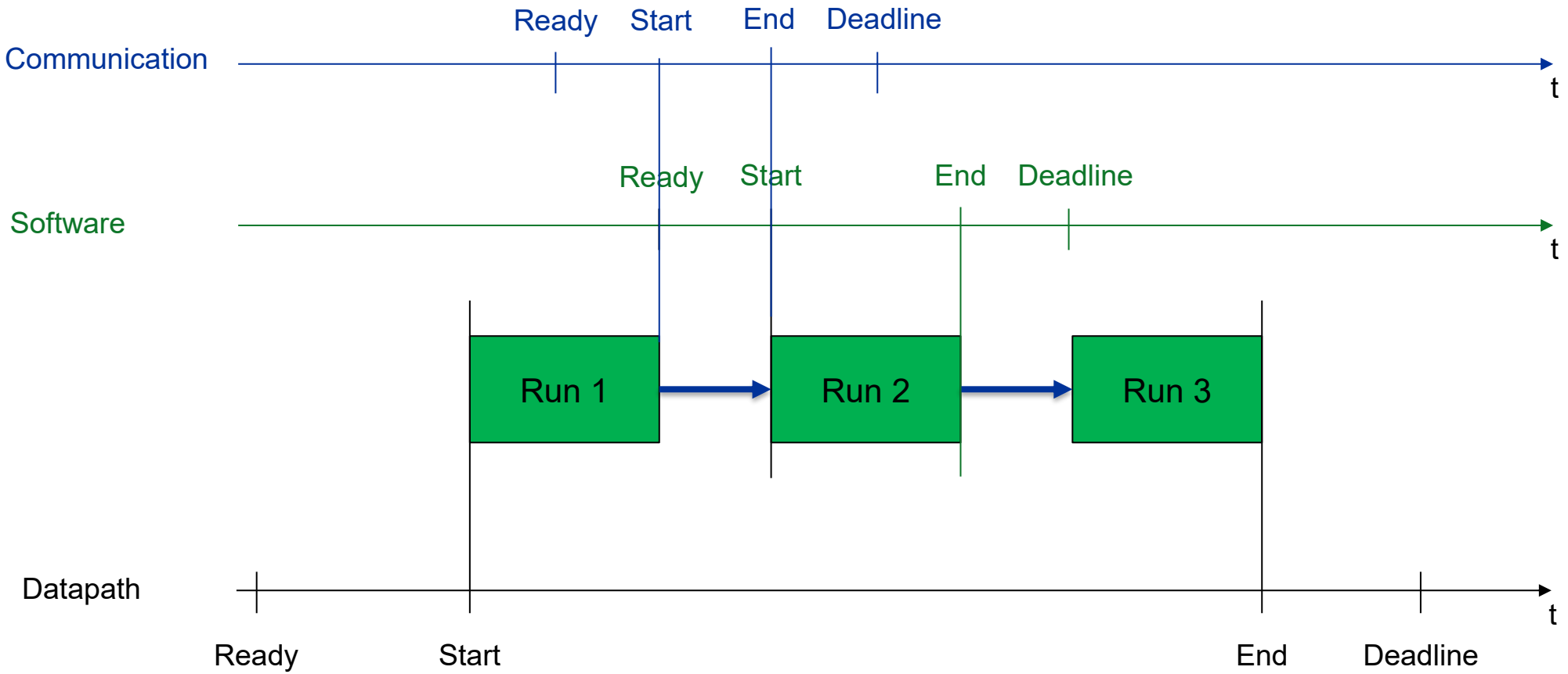
- Internal watchdogs are only reliable if the chips vital components such as power supply or clock synthesis are operational.
- Hence, common cause failures could occur.

- External alive monitoring with independent time domain becomes necessary



Deadline Monitoring

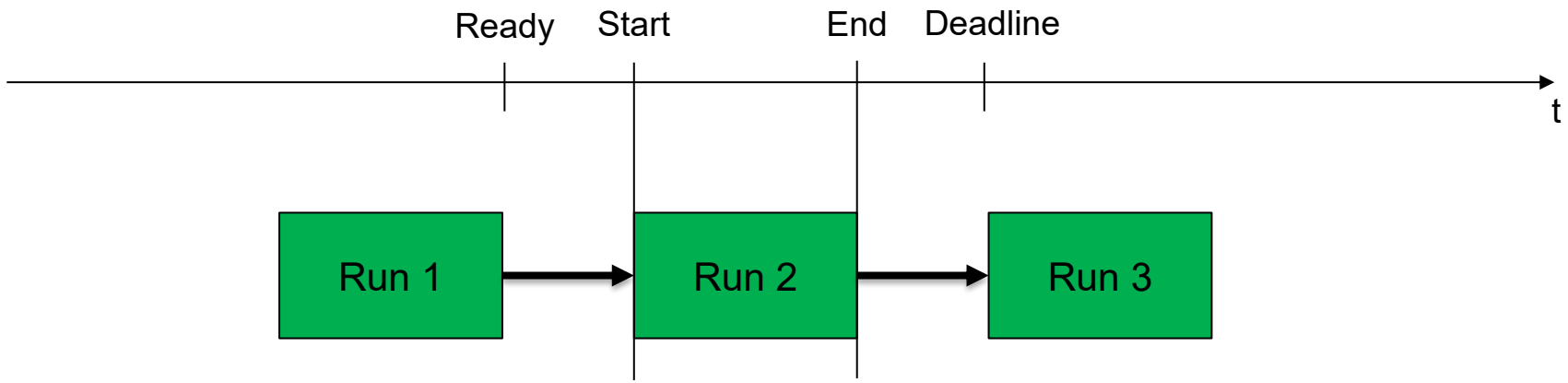
- Deadline monitoring can be applied on multiple levels
- As software functions typically execute within a time range which is way beyond the resolution of software timers, functions to be monitored might be grouped.



Deadline Monitoring – Abort Event

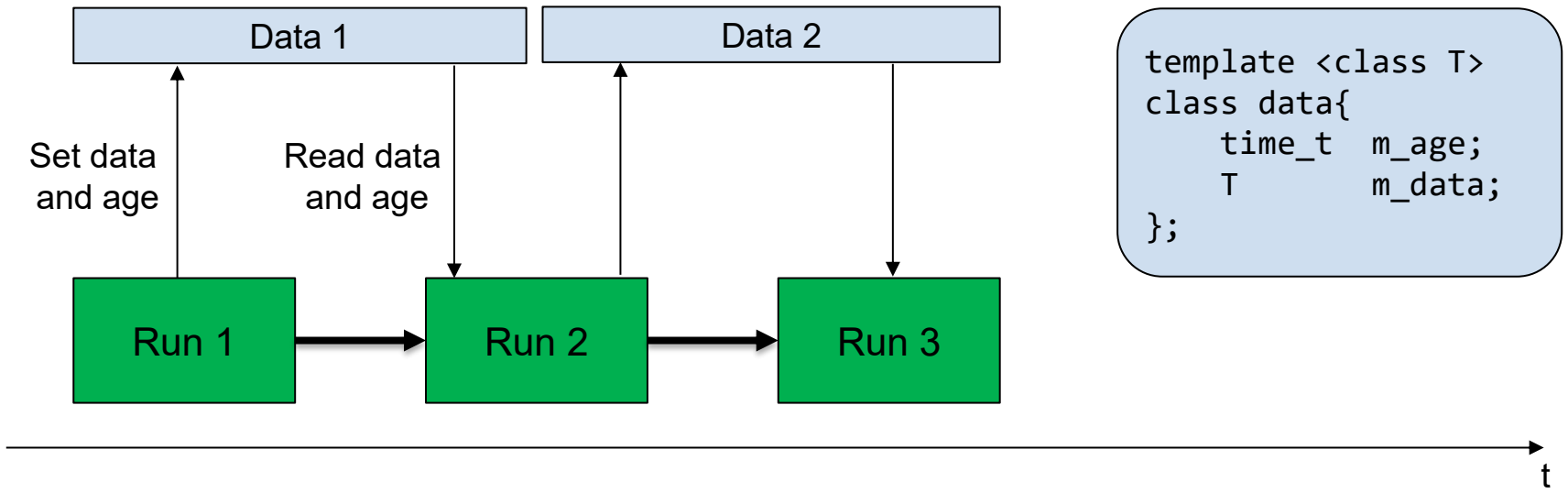
- Safety RTOS like PxROS support timeout mechanisms.
- Software monitoring is mainly suitable on software module level.
- Additional mechanisms are required to cover communication between software modules.

```
abortEventMask = PxExpectAbort(ev_timeout, runnable2);
if (abortEventMask.events != 0){
    //error handling
}
```



Deadline Monitoring – Signal Age Metadata

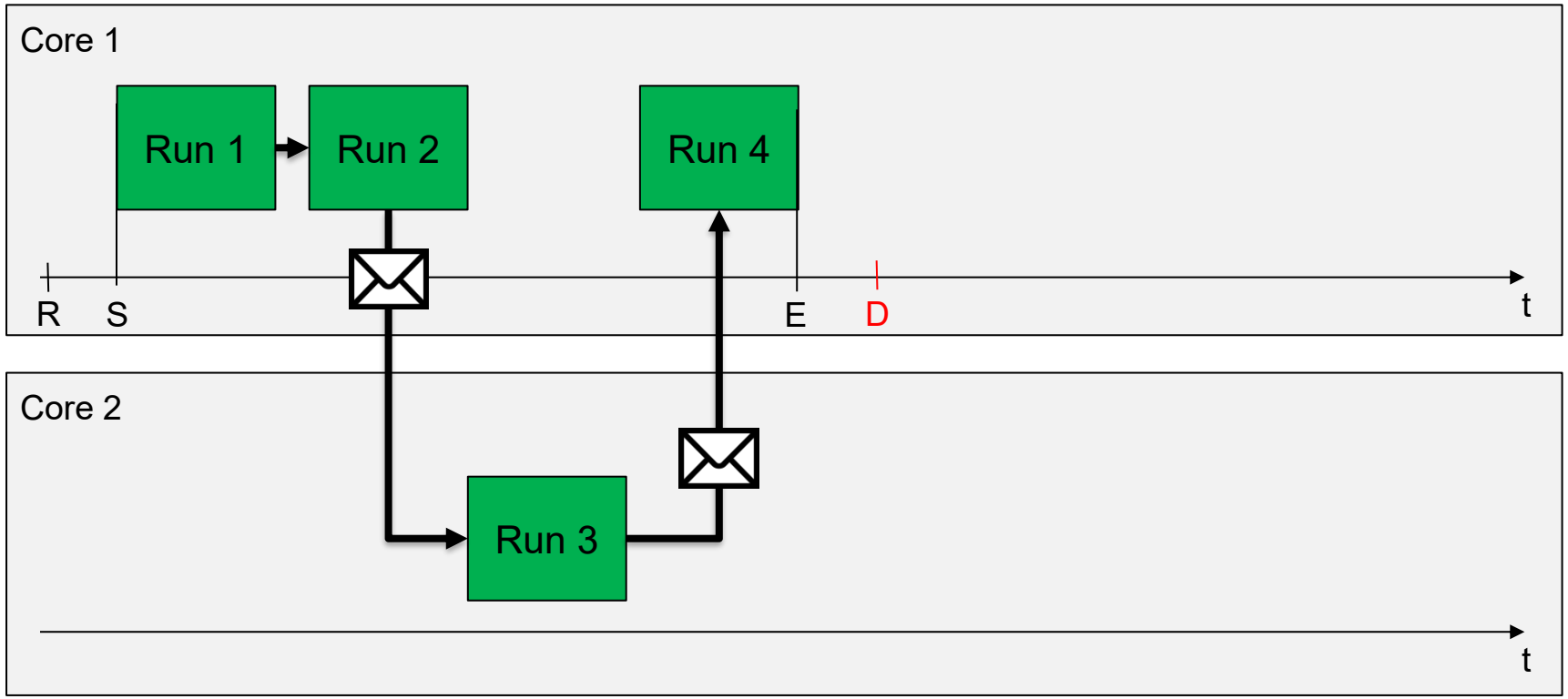
- A more data centric approach is to store the age of data together with the data payload.




```
template <class T>
class data{
    time_t  m_age;
    T       m_data;
};
```

Deadline monitoring – Limitations

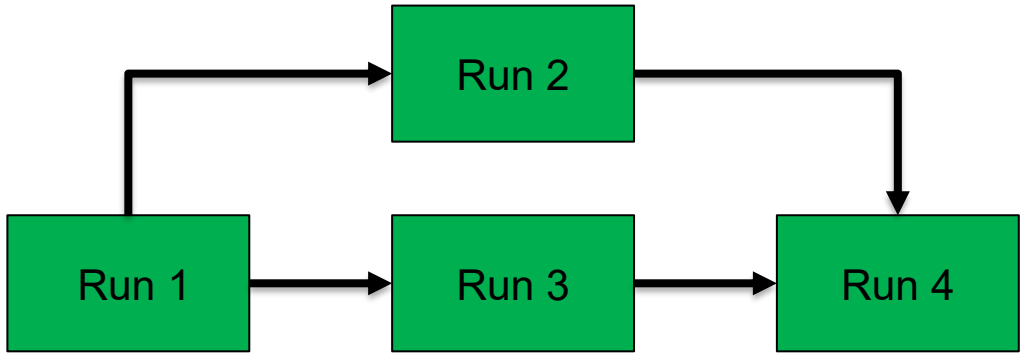
- Supervising the age of data covers communication but might come to limitations if independent sets of data containers are used within a single data path
- In a multicore environment introducing asynchronous, non-blocking messaging mechanisms between the cores, deadline monitoring alone comes to a limit



Control flow monitoring

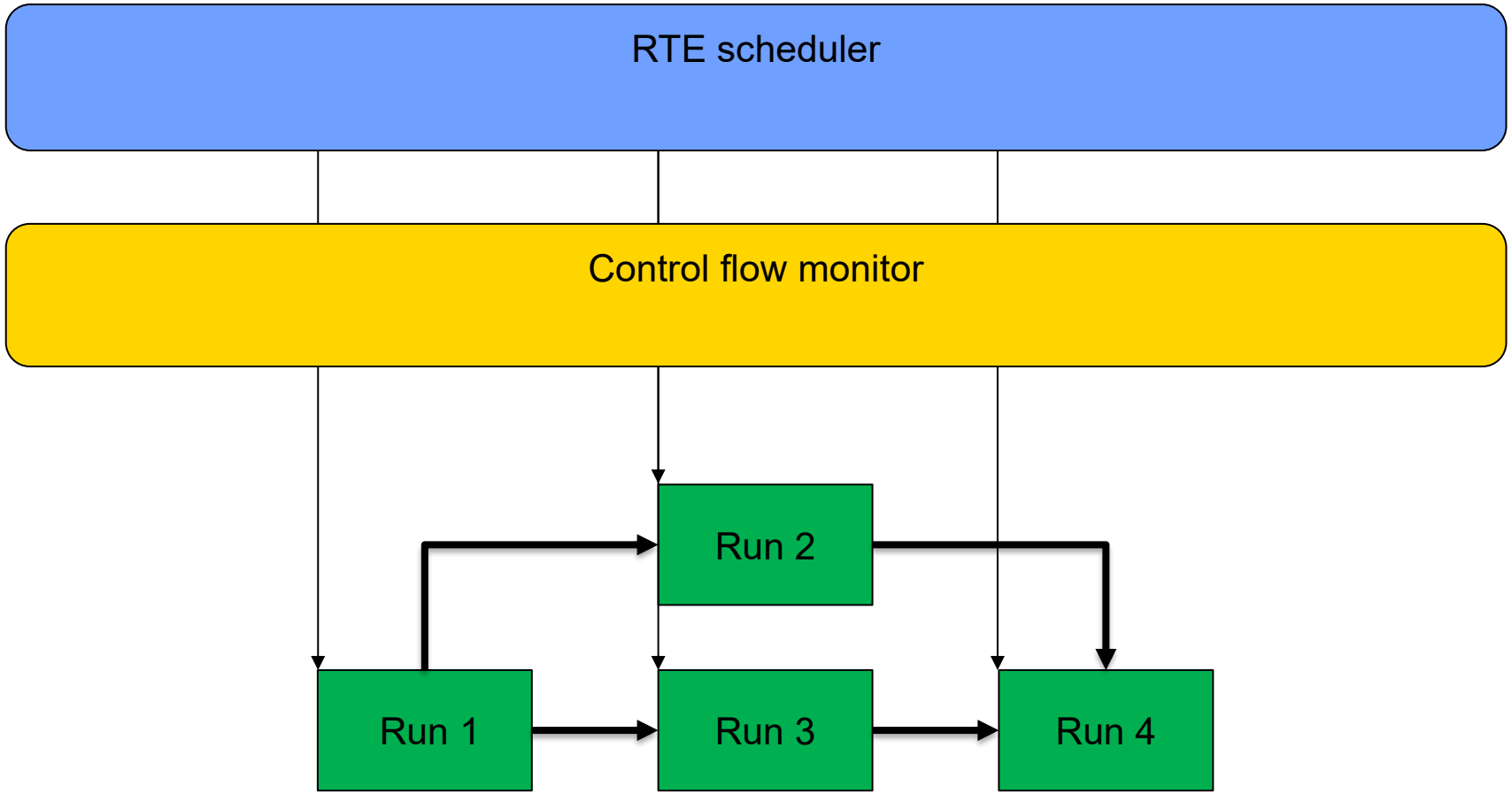
- Control flow monitoring ensures that software entities are executed in a valid sequence. There might be multiple valid sequences for a data path.

- In the example below, valid sequences are 1-2-4 and 1-3-4.
- Any other sequence indicates a control flow violation



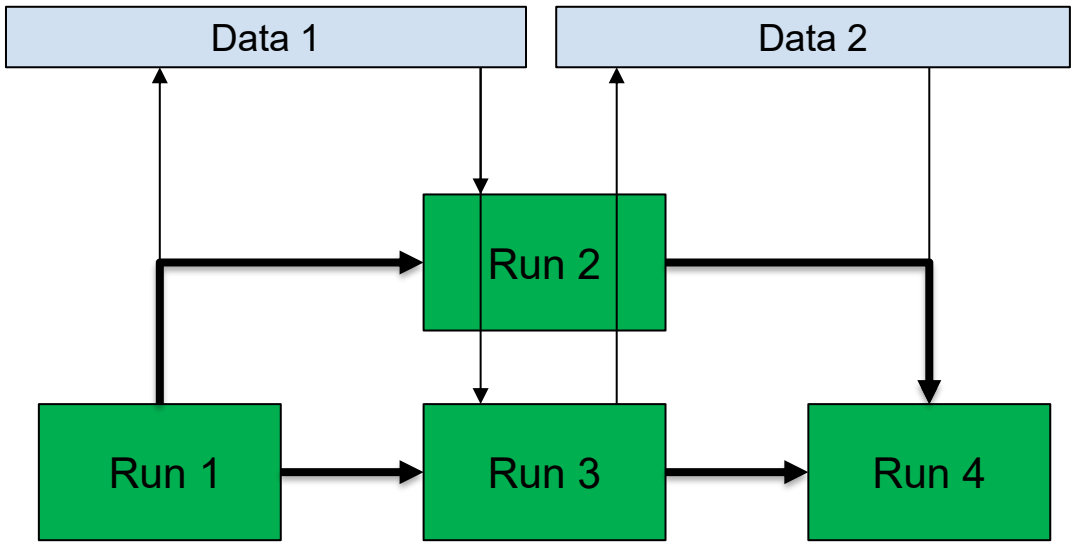
Control flow monitoring – RTE Integration

- If the software modules are called from a runtime environment, a dedicated monitor can be added for control flow supervision.



Control flow monitoring – Using signal age information

- The physical age of the signal can be compared with the expected age.
- This allows the detection of missing signal updates.

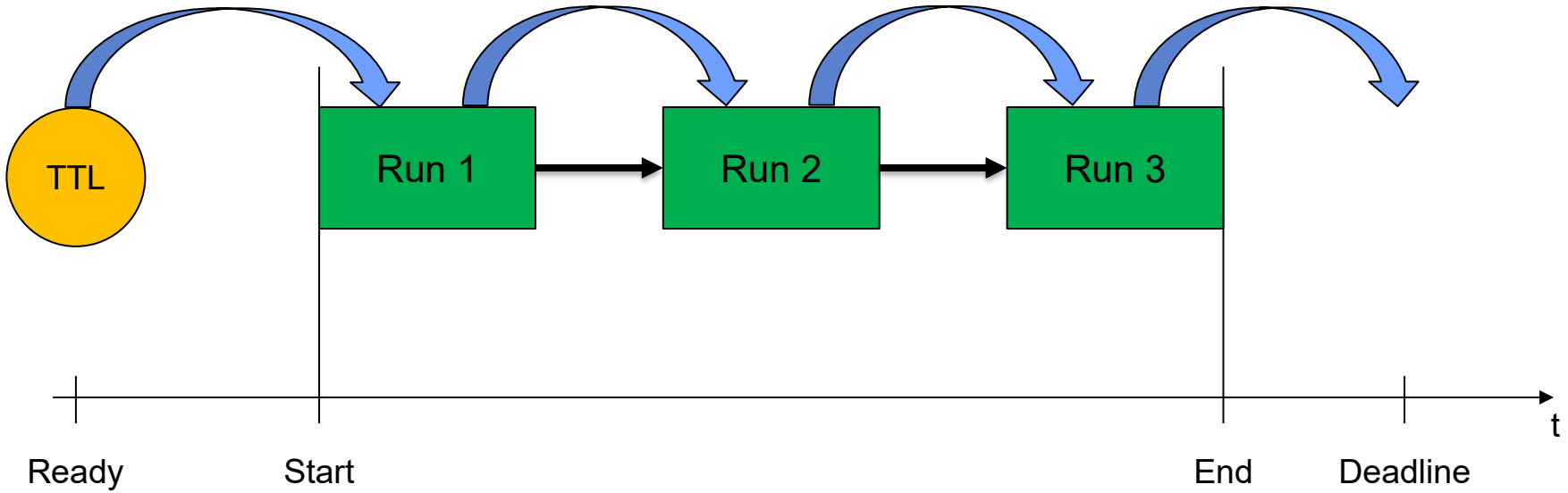



```
template <class T>
class data{
    time_t  m_age;
    T       m_data;
};
```

Control flow monitoring

Currently, the utilization of tokens is under investigation.

A token with a limited lifetime, which is passed trough the data path not only can be used for deadline monitoring but also might be suitable for control flow monitoring.

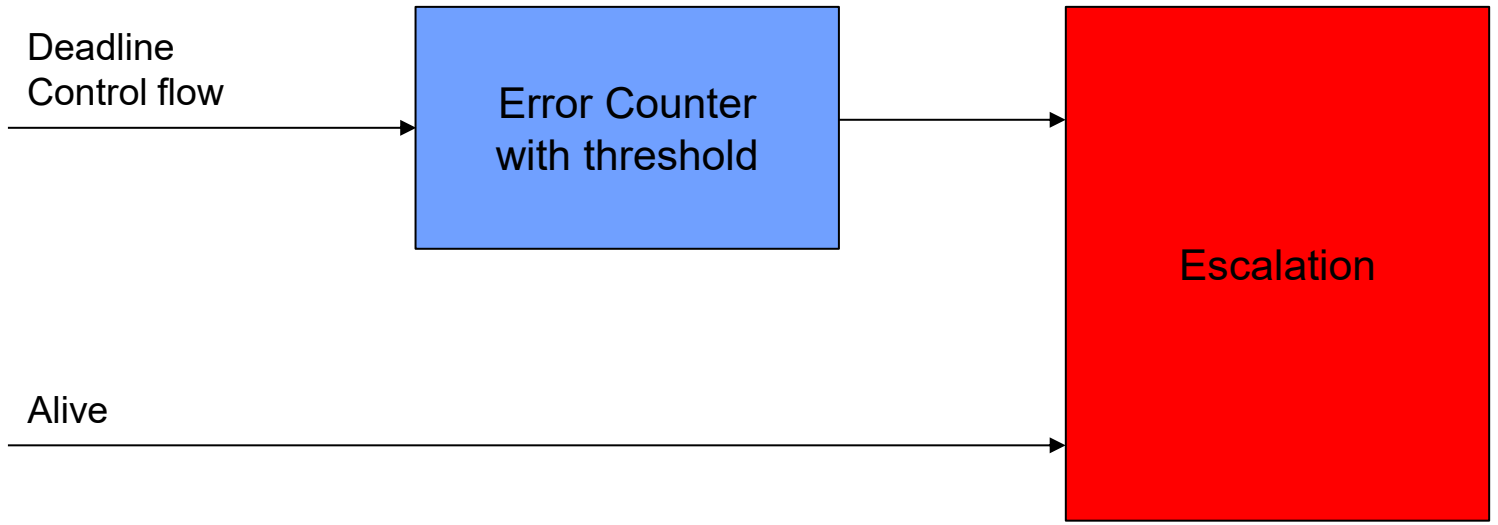


How to escalate errors?

Error escalation

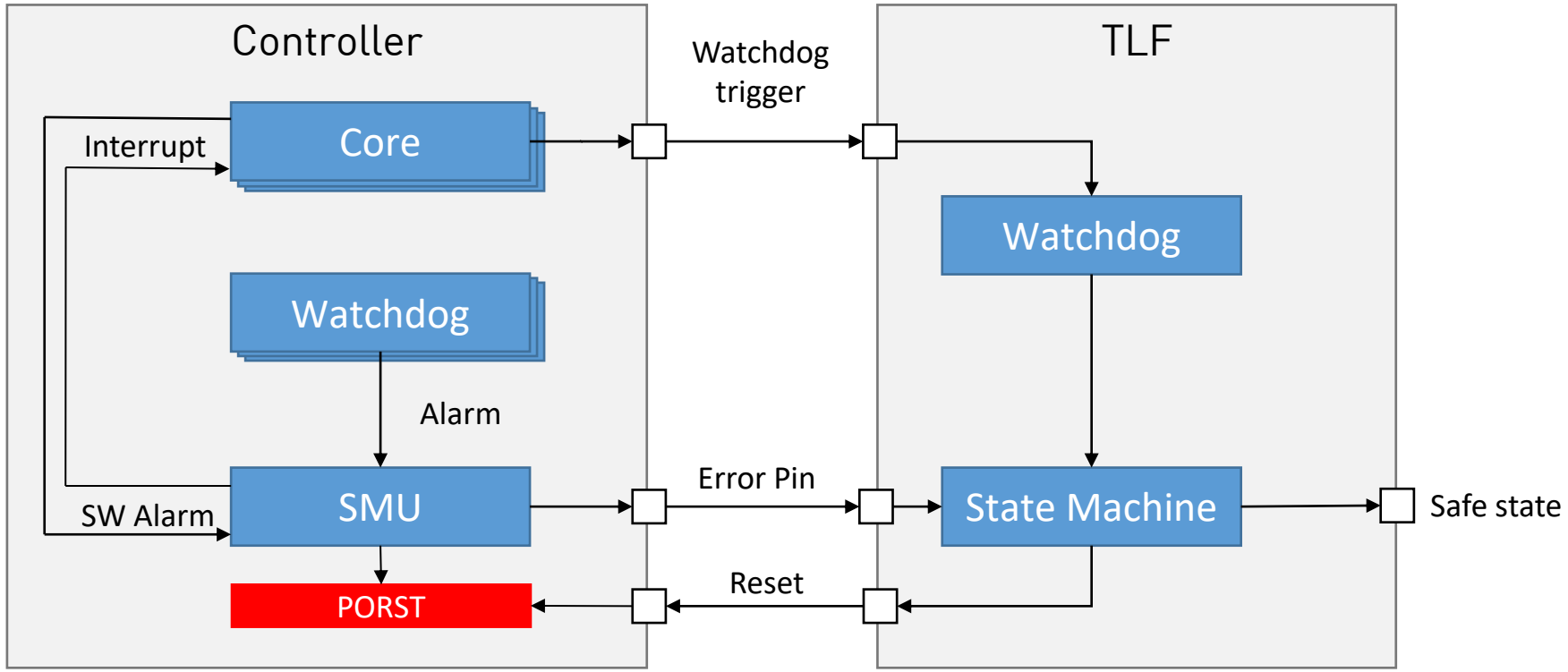
- Runnable level (deadline/control flow)
  - Might be a sporadic error
  - Is the datapath vital for machine safety?
  - Is there a backup mechanism?

- System level (alive)
  - All we know is that the controller is no longer in a consistent state



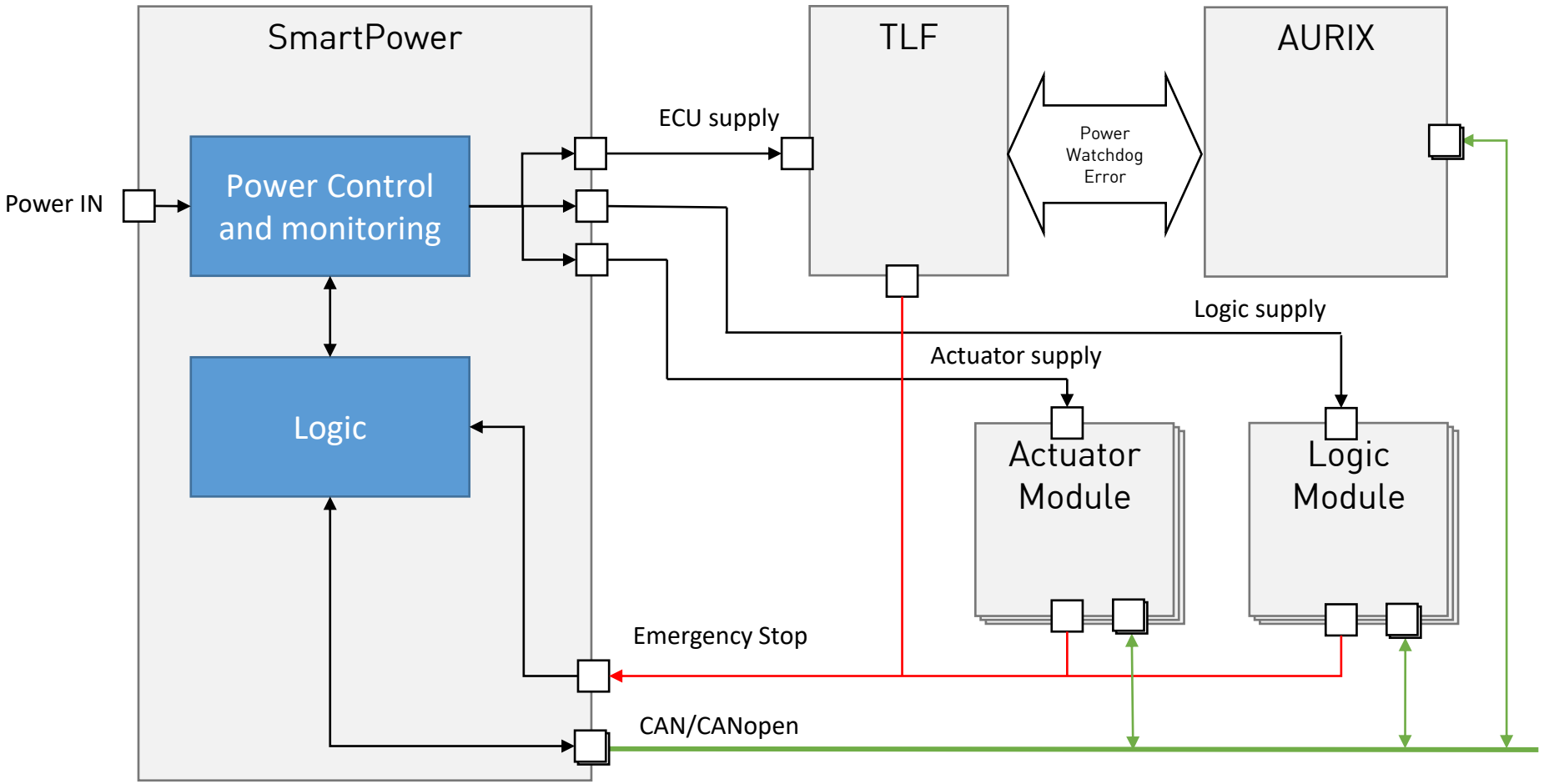
Error escalation

- The escalation of critical timing violations needs to be handled in hardware, as it cannot be guaranteed that software is executed properly anymore.
- Example: Infineon AURIX 1G



Error escalation – Last line of defense

- The safe state of the controller also needs to be escalated to the overall system



Conclusion

Multicore challenge

- The truly parallel execution of code data paths across cores require advanced monitoring concepts such as control flow supervision.
- Inter core error signaling is challenging.
- The cores need a common time base for synchronization.
- Monitoring concepts have a significant impact on the software architecture and need to be designed in an early design phase
- Robust error escalation strategies need to be considered also on system level where external monitors are required

Contact

M.Sc. Thomas Barth
Prof. Dr.-Ing. Peter Fromm


Hochschule Darmstadt - University of Applied Sciences
FB Elektrotechnik und Informationstechnik (EIT)
Birkenweg 8
64295 Darmstadt

Email: thomas.barth@h-da.de; peter.fromm@h-da.de
Web: www.eit.h-da.de