# *Real-Time Systems*

## *Lecture 01: Introduction*

*2012-04-24*

Dr. Bernd Westphal

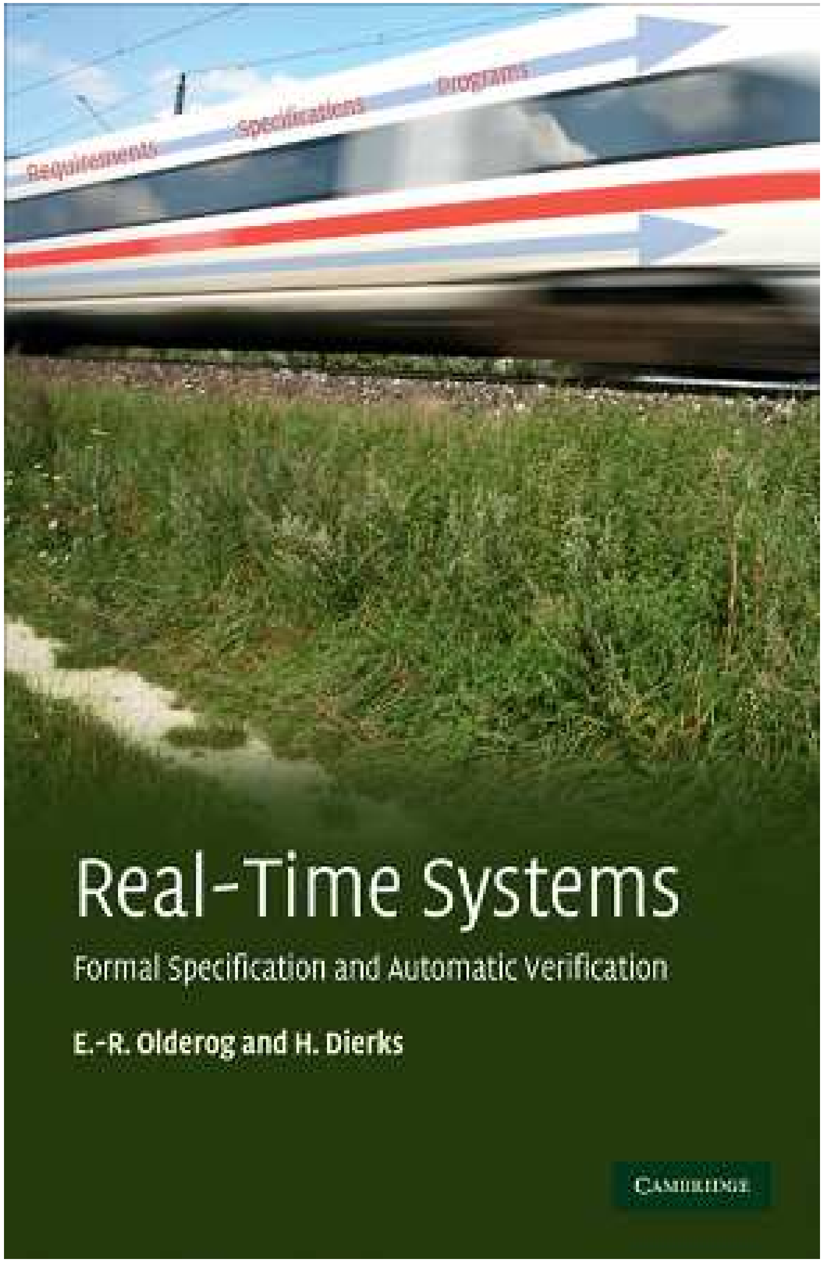
Albert-Ludwigs-Universität Freiburg, Germany

# *Today*

- **Introduction: Real-Time Systems**
- **Overview:** content (and non-content) of the lecture
- **Formalia:** dates/times, exercises, exam admission
- **Literature**

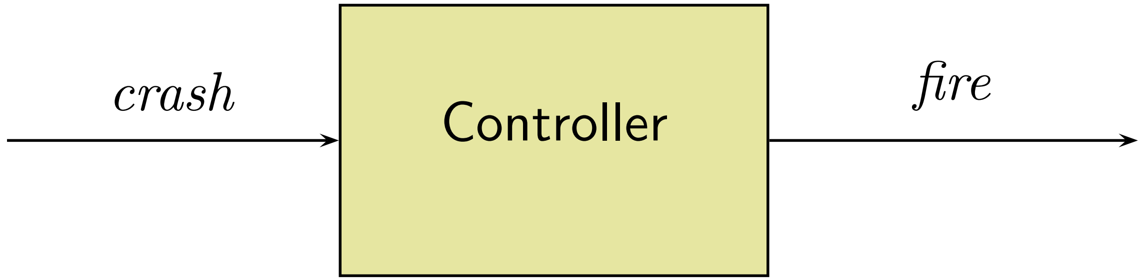
*Introduction*

# *Subject of the Lecture*



# *What is a Real-Time System?*

Classical example: **Airbag Controller**




**Requirement**: "When a crash is detected, fire the airbag."

- When firing **too early**: airbag ineffective.
- When firing **too late**: additional threat.

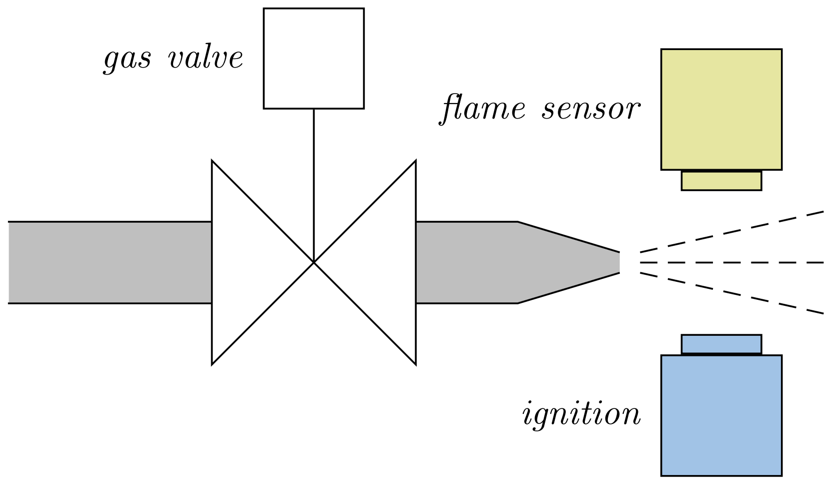
Say, 300ms (plus/minus small $\varepsilon$) after a crash is the right[TM] time to fire.

Then the **precise requirement** is

"When a crash is detected at time $t$, fire the airbag at $t + 300ms \pm \varepsilon$."

# *What is a Real-Time System?*

- Other example: **Gas Burner**




- **Leakage** is practically unavoidable:
  - for ignition, first open valve
  - then ignite the available gas
  - ignition may fail. . .

- **Leakage** is **safety critical**:
  Igniting large amounts of leaked gas may lead to a dangerous explosion.

# *No, Really, What is a Real-Time System?*

- The examples have in common that

  **it matters**, **when in time**

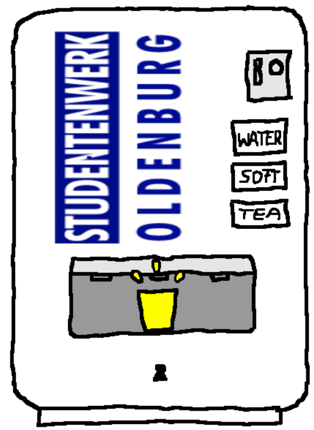
  the output for a given input (sequence) takes place.

  For instance,

  - "fire" 300ms after "crash",
  - within any interval of at least 60s, leakage (= have the gas valve open without a flame) amounts to at most 5% of the time.

  Note: **quantitative** (here) vs. **qualitative** notions of time (untimed).

- Often: There is a physical environment, which has a notion of time, and which evolves while our controller is computing.




- (Half-)**Contrast:** vending machine for soft-drinks:
  - If the customer is really thirsty, she'll wait.
  - Neither the usage of a really fast or a really slow contemporary controller causes a violation of (timing) requirements.
- (Real) **Contrast:** transformational systems, such as computing $\pi$.

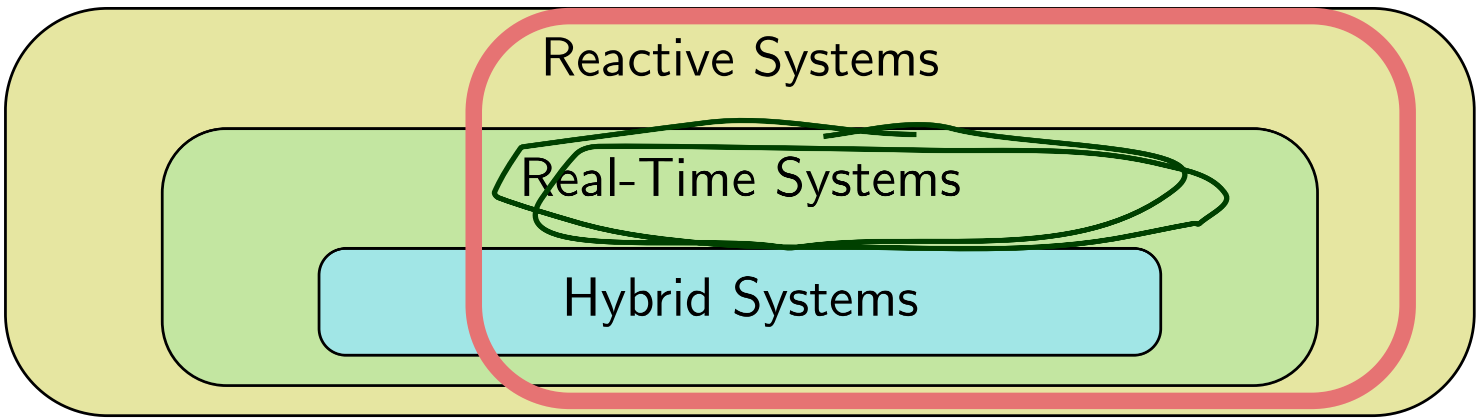
# Other Definitions [Douglass, 1999]

- "A **real-time** system is one that has **performance deadlines** on its computations and actions."
- Distinguish:
  - "**Hard** deadlines: performance requirements that **absolutely must** be met each and every event or time mark."
    "(Late data can be bad data.)"
  - "**Soft** deadlines: for instance about **average** response times."
    "(Late data is still good.)"
- Design Goal:
  A **timely system**, i.e. one meeting its performance requirements.
- Note: **performance** can in general be any unit of quantities:
  - (discrete) number of steps or processor instructions,
  - (discrete or continuous) number of seconds,
  - etc.

# *Definitions: Reactive vs. Real-Time vs. Hybrid Systems*

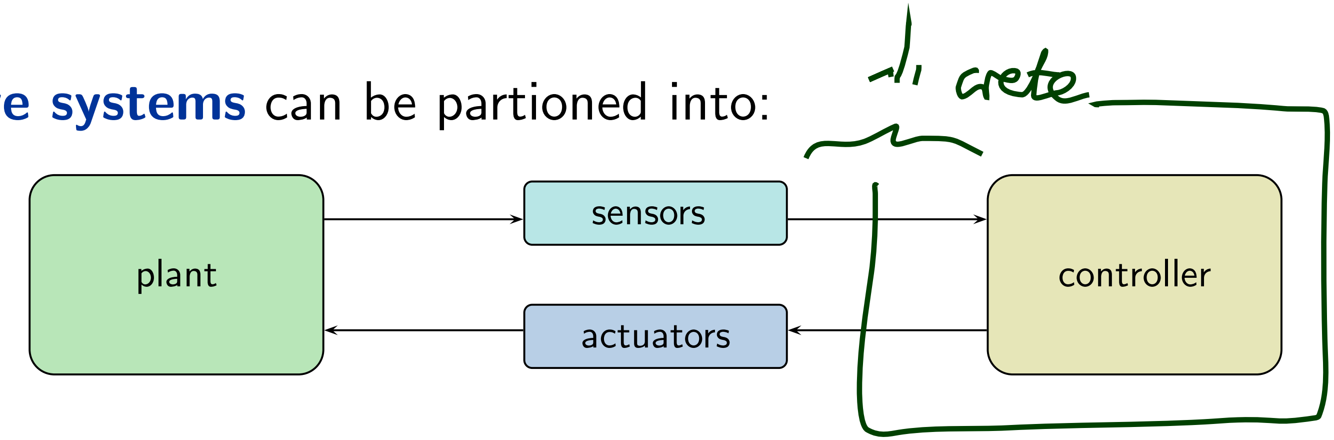
- **Reactive Systems** interact with their environment by reacting to inputs from the environment with certain outputs.
- A **Real-Time System** is a **reactive system** which, for certain inputs, has to compute the corresponding outputs within given time bounds.
- A **Hybrid System** is a **real-time system** consisting of continuous and discrete components. The continuous components are time-dependent (!) physical variables ranging over a continous value set.

- A system is called **Safety Critical** if and only if a malfunction can cause loss of goods, money, or even life.



# *The Problem: Constructing Safety-critical RT Systems*

- **Reactive systems** can be partioned into:



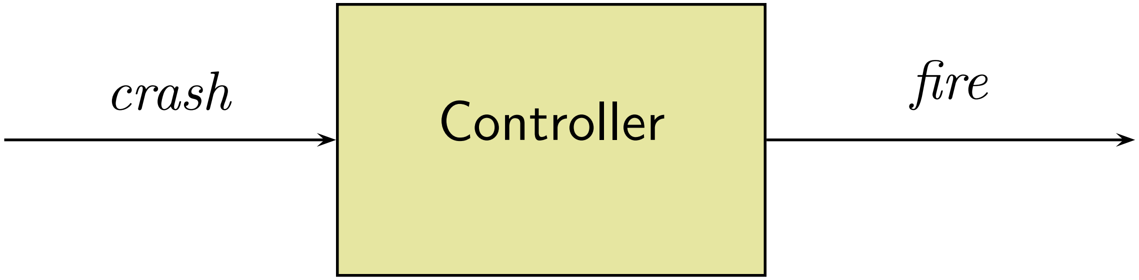

  - "In constructing a **real-time system** the aim is to control a physically existing environment, the **plant**, in such a way that the controlled plant satisfies all desired (timing) requirements."

- The design of **safety critical** (**reactive**) **systems** requires a high degree of precision:
  We want — at best — to be sure that a design meets its requirements.

- **Real-time systems** are often **safety-critical**.

- The lecture presents approaches for the precise development of **real-time systems** based on formal, mathematical methods.

# Constructing Safety-critical RT Systems: Examples




"When a crash is detected at time $t$, fire the airbag at $t + 300ms \pm \varepsilon$."

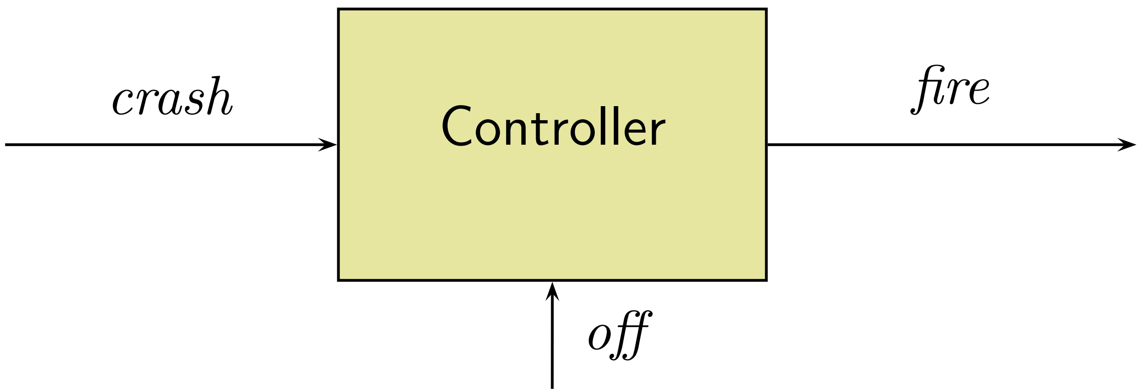
- A controller program is easy:

```
while (true) do
  poll_sensors();
  if (crash) tmr.start(300ms);
  if (tmr.elapsed()) fire := 1;
  update_actuators();
od
```

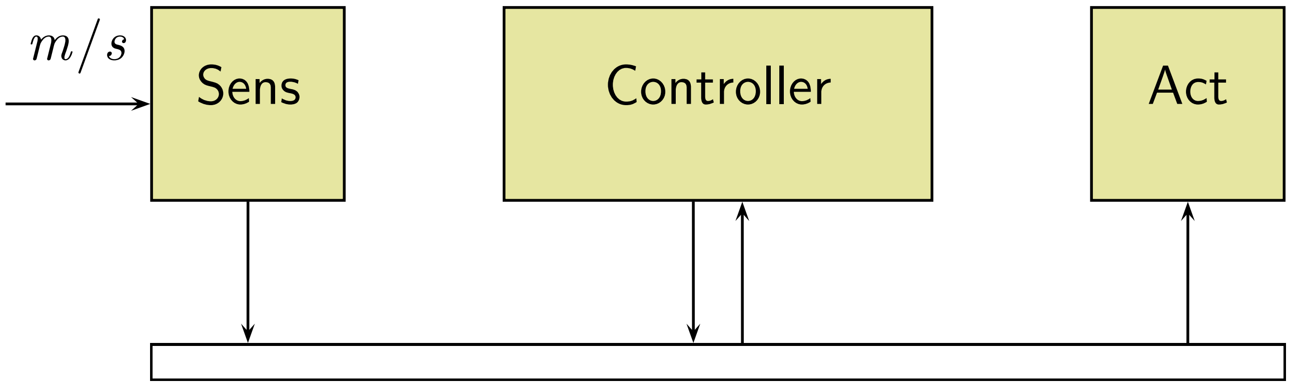
- And likely to be believed to be correct.

# *Constructing Safety-critical RT Systems: Examples*

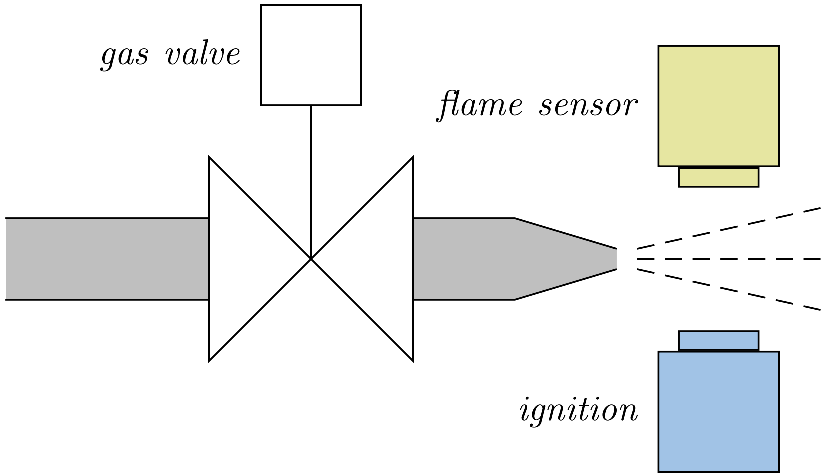
- More complicated: **additional features**.




- More complicated: **distributed implementation**.



# Constructing Safety-critical RT Systems: Examples



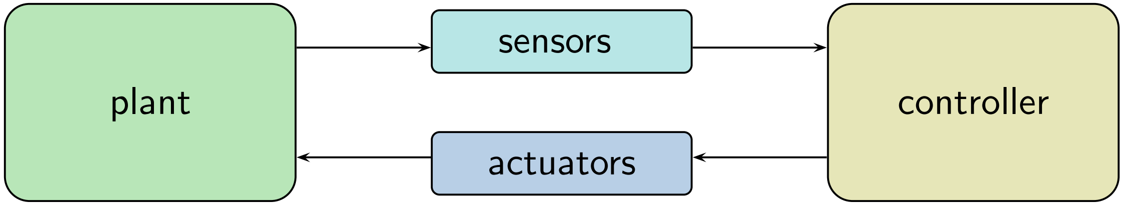

- Leakage is **safety critical**:
  Igniting large amounts of leaked gas may lead to a dangerous explosion.

- Controller program for ignition is easy:

```
while (!flame) do
  open_valve();
  wait(t);
  ignite();
od
```

- Is it **correct**? (Here: Is it avoiding dangerous explosions?)

# *Prerequisites*




To

**design a controller that meets its requirements**

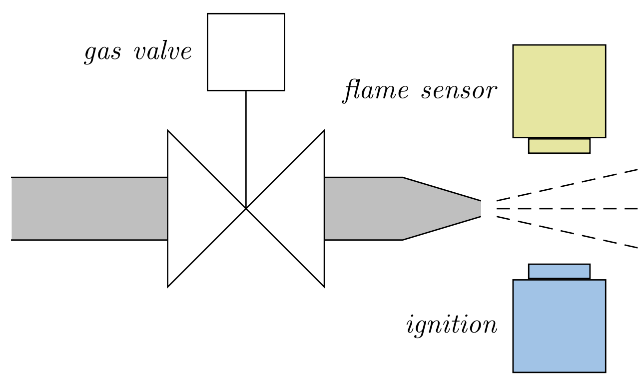
we need

- a formal model of behaviour in (quantitative) time,
- a language to concisely, conveniently specifiy requirements on behaviour,
- a language to specify behaviour of controllers,
- a notion of "meet" and a methodology to verify "meeting".

Then we can devise **a methodology** to get **from requirements** to a **(correct) implementation** — here: following [Olderog and Dierks, 2008].

# Sketch of the Methodology: Gas Burner Example

- **Requirements**
  - At most 5% of any at least 60s long interval amounts to leakage.
- **Reflective Design**
  - Time intervals with leakage last at most 1s.
  - After each leak, wait 30s before opening valve again.
- **Constructive Design**
  - PLC Automaton
    (open valve for 0.5s;
    ignite;
    if no flame after 0.1s close valve)
- **Implementation**
  - IEC 61131-3 program



# *Content Overview*

# *Content*

**Introduction**

- **First-order Logic**
- **Duration Calculus** (DC)
- Semantical Correctness Proofs with DC
- DC Decidability
- DC Implementables
- **PLC-Automata**

$obs : \mathsf{Time} \to \mathscr{D}(obs)$

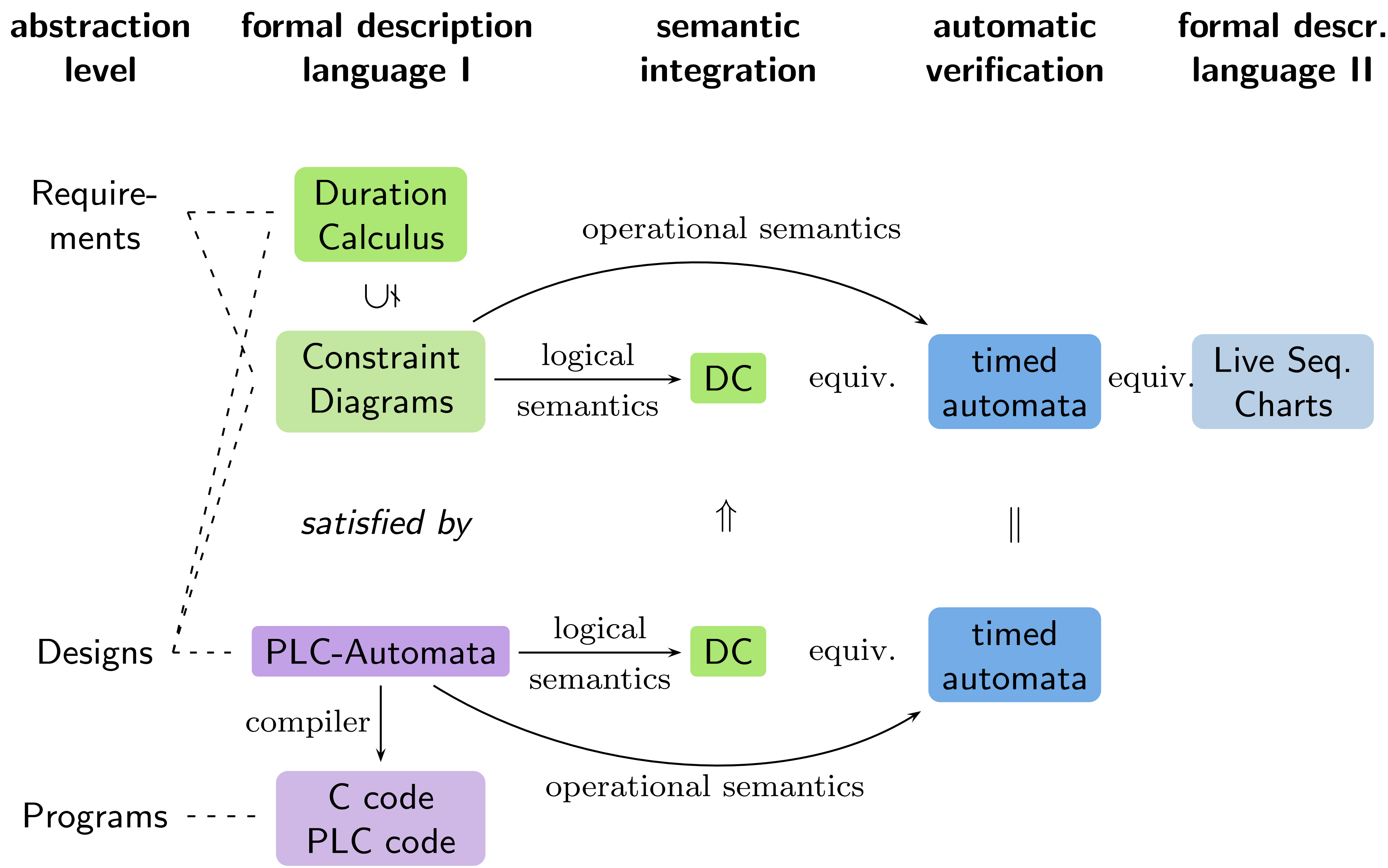
- **Timed Automata** (TA), Uppaal
- Networks of Timed Automata
- Region/Zone-Abstraction
- Extended Timed Automata
- Undecidability Results

$\langle obs_0, \nu_0 \rangle, t_0 \xrightarrow{\lambda_0} \langle obs_1, \nu_1 \rangle, t_1 \ldots$

- **Automatic Verification**...
- ...whether TA satisfies DC formula, observer-based

**Recap**

# Tying It All Together



# Maybe-Content

- **Worst Case Execution Time**
  - Recall over-simplified airbag controller:

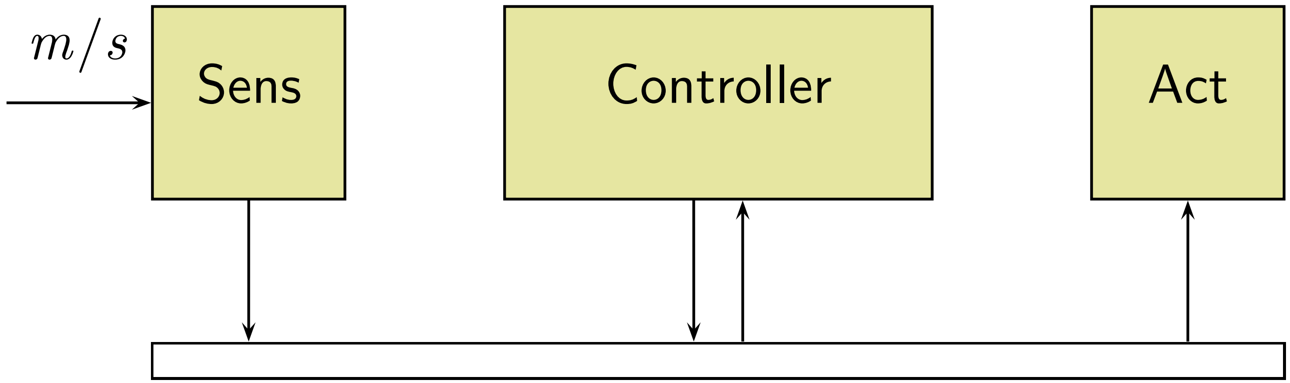
```
while (true) do
  poll_sensors();
  if (crash) tmr.start(300ms);
  if (tmr.elapsed()) fire := 1;
  update_actuators();
od
```

  - The execution of `poll_sensors()` and `update_actuators()` also takes time! (And we have to consider it!)

  - **Maybe in lecture**:
    How to determine the WCET of, for instance, C code.
    (A science of its own.)

# *Non-Content*

- **Scheduling**
  - Recall over-simplified airbag controller:




  - **Not in lecture**: Specialised methods to determine...
    - ...whether the bus provides sufficient bandwidth.
    - ...whether the Real-Time OS controlling CPU 'Controller' schedules the airbag control code in time.
    - ...how to distribute tasks over multiple CPUs.
    - etc.

  (Also a science of its own.)

# *Formalia*

# *Formalia: Event*

- **Lecturer:** Dr. Bernd Westphal
- **Support:** Siyar Andisha
- **Homepage:**

  `http://swt.informatik.uni-freiburg.de/teaching/SS2012/rtsys`

- **Questions:**
  - **"online"**:
    (i) ask immediately or in the break
  - **"offline"**:
    (i) try to solve yourself
    (ii) discuss with colleagues
    (iii) contact lecturer by mail (cf. homepage) or just drop by:
    Building 52, Room 00-020

# *Formalia: Dates/Times, Break*

- **Schedule:**

| | | | |
|---|---|---|---|
| Thursday, week $N$: | 10–12 **lecture** | | (exercises $M$ **online**) |
| Tuesday, week $N+1$: | 10–12 **lecture** | | |
| Thursday, week $N+1$: | 10–12 **lecture** | | |
| Monday, week $N+2$: | 9:00 | | (exercises $M$ **early turn-in**) |
| Tuesday, week $N+2$: | 10–12 **tutorial** | | (exercises $M$ **late turn-in**) |
| Thursday, week $N+2$: | 10–12 **lecture** | | (exercises $M+1$ **online**) |

  With a prefix of lectures, with public holidays; see homepage for details.

- **Location:**
  - Tuesday, Thursday: here

- **Break:**
  - Unless a majority objects **now**,
    we'll have a **10 min. break** in the middle of each event from now on.

# *Formalia: Lectures*

- **Course language: English**
  (slides/writing, presentation, questions/discussions)

- **Presentation:**
  half slides/half on-screen **hand-writing** — for reasons

- **Script/Media:**
  - slides without annotations on **homepage**,
    **trying** to put them there **before** the lecture
  - slides with annotations on **homepage**, 2-up for printing,
    typically soon **after** the lecture
  - recording on eLectures portal with (max. 1 week delay)
    (link on **homepage** – eLectures is updated first, look there!)

- **Interaction:**
  absence often moaned but **it takes two**,
  so please ask/comment immediately

# *Formalia: Exercises and Tutorials*

- **Schedule/Submission:**
  - Recall: exercises **online** on Thursday before lecture, regular **turn in** on corresponding tutorial day until **10:00 local time**
  - should work in groups of **max. 3**, clearly give **names** on submission
  - please submit **electronically** by Mail to **me** (cf. homepage), some LaTeX styles on homepage; paper submissions are tolerated

- **Didactical aim:**
  - deal more extensively with notions from lecture (easy)
  - explore corner cases or alternatives (medium)
  - evaluate/appreciate approaches (difficult)
  - additional **difficulty**: imprecise/unclear tasks — by intention

- **True aim:** **most complicated** rating system **ever**, namely two ratings
  - Good-will ("reasonable solution with knowledge **before** tutorial")
  - Evil/Exam ("reasonable solution with knowledge **after** tutorial")

10% **bonus** for **early** submission.

# *Formalia: Exam*

- **Exam Admission:**

  50% of the maximum possible non-bonus **good-will points** in total are **sufficient** for admission to exam

- **Exam Form:** (oral or written) not yet decided

# *Formalia: Evaluation*

Speaking of **grading and examination**...

- **Mid-term Evaluation:**
  We will have a **mid-term evaluation**[a], but we're **always** interested in comments/hints/proposals concerning form or content.

[a] that is, students are asked to evaluate lecture, lecturer, and tutor...

# *Formalia: Questions?*

*Real-Time Behaviour, More Formally...*

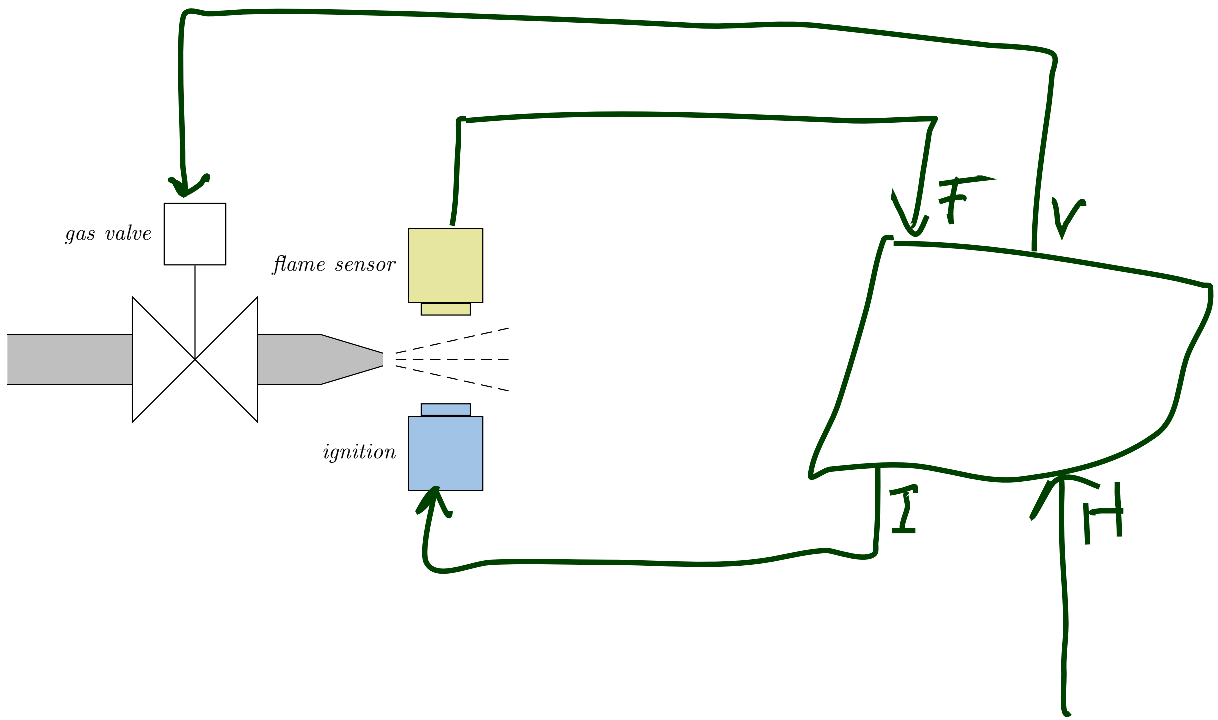
# *State Variables (or Observables)*

- We assume that the real-time systems we consider is characterised by a finite set of **state variables** (or **observables**)

$$obs_1, \dots, obs_n$$

each equipped with a **domain** $\mathcal{D}(obs_i)$, $1 \leq i \leq n$.

- **Example**: gas burner




- $V$, $\mathcal{D}(V) = \{0, 1\}$
- $F$, $\mathcal{D}(F) = \{0, 1\}$
- $I$, $\mathcal{D}(I) = \{0, 1\}$
- $H$, $\mathcal{D}(H) = \{0, 1\}$

# System Evolution over Time

- **One** possible evolution (or **behaviour**) of the considered system over time is represented as a function

$$\pi : \mathsf{Time} \to \mathcal{D}(obs_1) \times \cdots \times \mathcal{D}(obs_n).$$

- If (and only if) observable $obs_i$ has value $d_i \in \mathcal{D}(obs_i)$ at time $t \in \mathsf{Time}$, $1 \leq i \leq n$, we set

$$\pi(t) = (d_1, \ldots, d_n).$$

- For convenience, we use

$$obs_i : \mathsf{Time} \to \mathcal{D}(obs_i)$$

to denote the projection of $\pi$ onto the $i$-th component.

## *What's the time?*

- There are two main choices for the time domain Time:

  - **discrete time**: Time $= \mathbb{N}_0$, the set of natural numbers.
  - **continuous or dense time**: Time $= \mathbb{R}_0^+$, the set of non-negative real numbers.

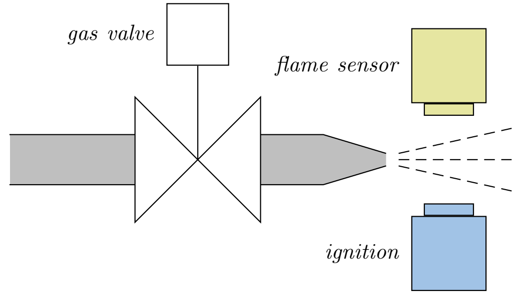
- Throughout the lecture we shall use the **continuous** time model and consider **discrete** time as a special case.

  Because

  - plant models usually live in **continuous** time,
  - we avoid too early introduction introduction of hardware considerations,

- Interesting view: continous-time is a well-suited **abstraction** from the discrete-time realms induced by clock-cycles etc.

# Example: Gas Burner



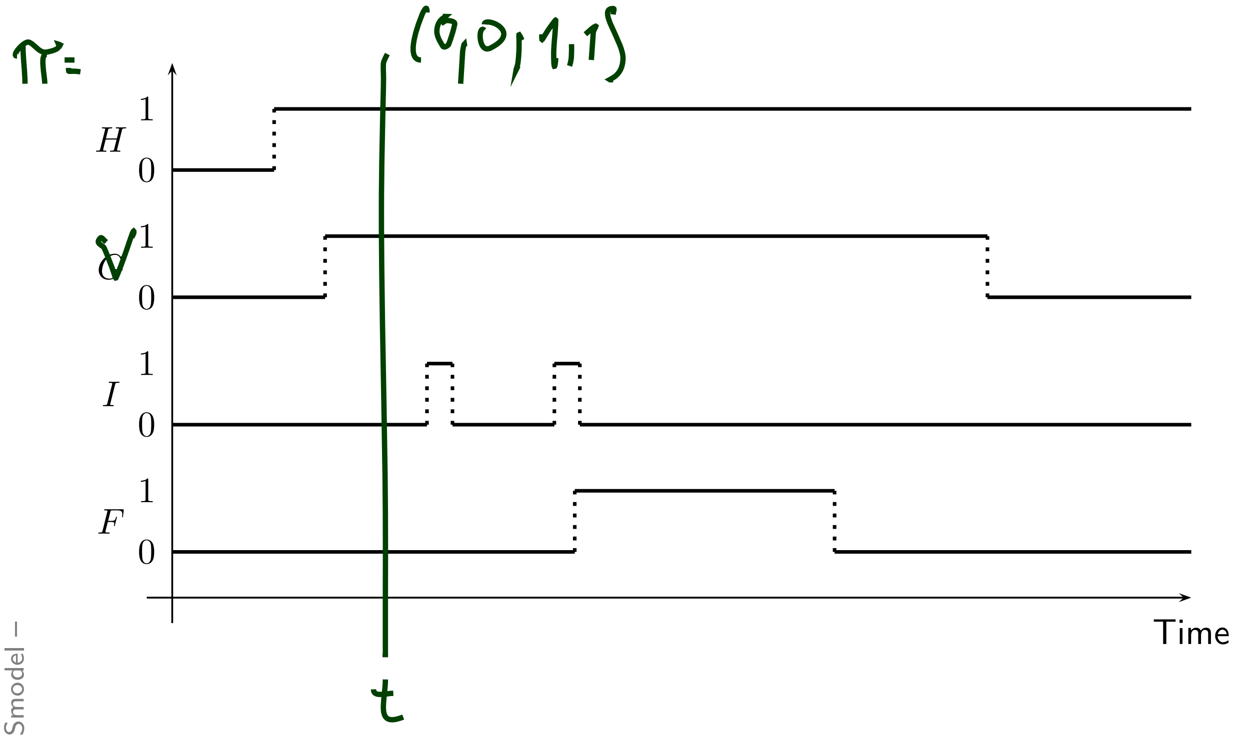

One possible evolution of considered system over time is represented as function

$$\pi : \text{Time} \to \mathcal{D}(obs_1) \times \cdots \times \mathcal{D}(obs_n).$$

If (and only if) observable $obs_i$ has value $d_i \in \mathcal{D}(obs_i)$ at time $t \in \text{Time}$, set:

$$\pi(t) = (d_1, \ldots, d_n).$$

For convenience: use $obs_i : \text{Time} \to \mathcal{D}(obs_i)$.



# *References*

# References

[Douglass, 1999] Douglass, B. P. (1999). *Doing Hard Time*. Addison-Wesley.

[Olderog and Dierks, 2008] Olderog, E.-R. and Dierks, H. (2008). *Real-Time Systems - Formal Specification and Automatic Verification*. Cambridge University Press.