# Static Analysis of Real-Time Embedded Systems with REK

Arie Gurfinkel[1]

joint work with Sagar Chaki[1],
Ofer Strichman[2], and Soonho Kong[1]

[1]Software Engineering Institute, CMU
[2]Technion, Israel Institute of Technology

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.

# Automated Software Analysis

Program

Property

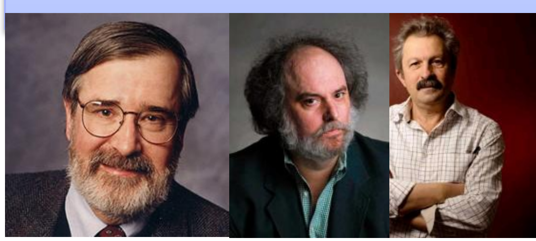
Automated Analysis

Correct + Proof

Incorrect + Counterexample

Software Model Checking with Predicate Abstraction

e.g., Microsoft's SDV



Abstract Interpretation with Numeric Abstraction

e.g., ASTREE, Polyspace

## Logic and Epistemology in Assurance Cases

- We have just two sources of doubt in an assurance case
- Logic doubt: the validity of the argument
    - Can be eliminated by formal verification
    - Subject to caveats discussed elsewhere
    - Automation allows what-if experimentation to bolster reviewer confidence
    - We can allow "because I say so" proof rules
- Epistemic doubt: the accuracy and completeness of our knowledge of the world in its interaction with the system
    - This is where we need to focus
- Same distinction underlies Verification and Validation (V&V)

# Motivation: Real-Time Embedded Systems

Avionics Mission System*

Rate Monotonic Scheduling (RMS)

| Task | Period |
|---|---|
| weapon release | 10ms |
| radar tracking | 40ms |
| target tracking | 40ms |
| aircraft flight data | 50ms |
| display | 50ms |
| steering | 80ms |

*Locke, Vogel, Lucas, and Goodenough. “Generic Avionics Software Specification”. SEI/CMU Technical Report CMU/SEI-90-TR-8-ESD-TR-90-209, December, 1990

# Periodic Program

An N-task periodic program PP is a set of tasks $\{\tau_1, \ldots, \tau_N\}$

A task $\tau$ is a tuple $\langle I, T, P, C, A \rangle$, where

- I is a task identifier
- T is a task body (i.e., code)
- P is a period
- C is the worst-case execution time
- A is the *release time:* the time at which task becomes first enabled

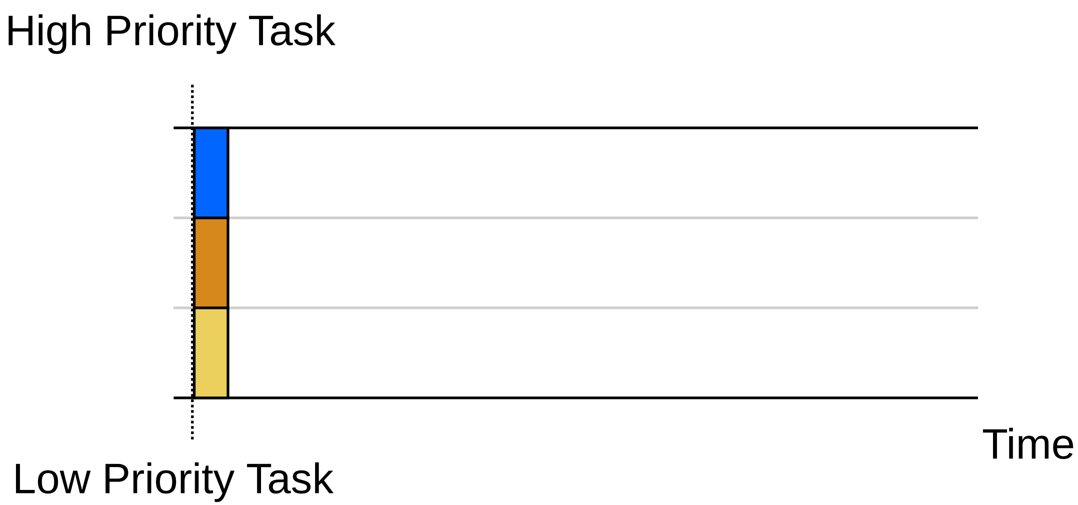
Semantics of PP is given by an asynchronous concurrent program:

parallel execution w/ priorities

```
|| k_i = 0;
   while (Wait(τ_i, k_i))
     T_i ();
     k_i = k_i + 1;
```

blocks task *i* until next arrival time

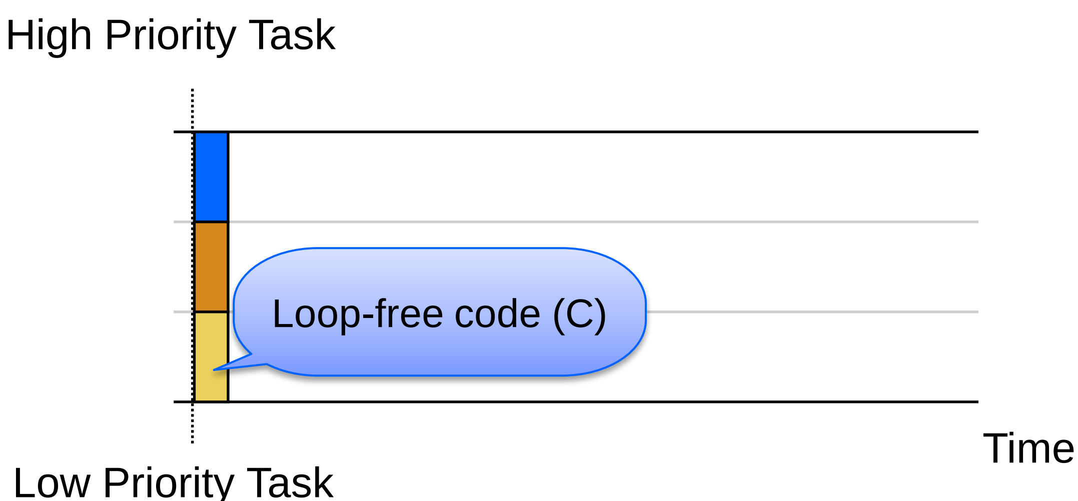
# Periodic Programs




Priority

Task $\tau = (I, T, P, C, A)$

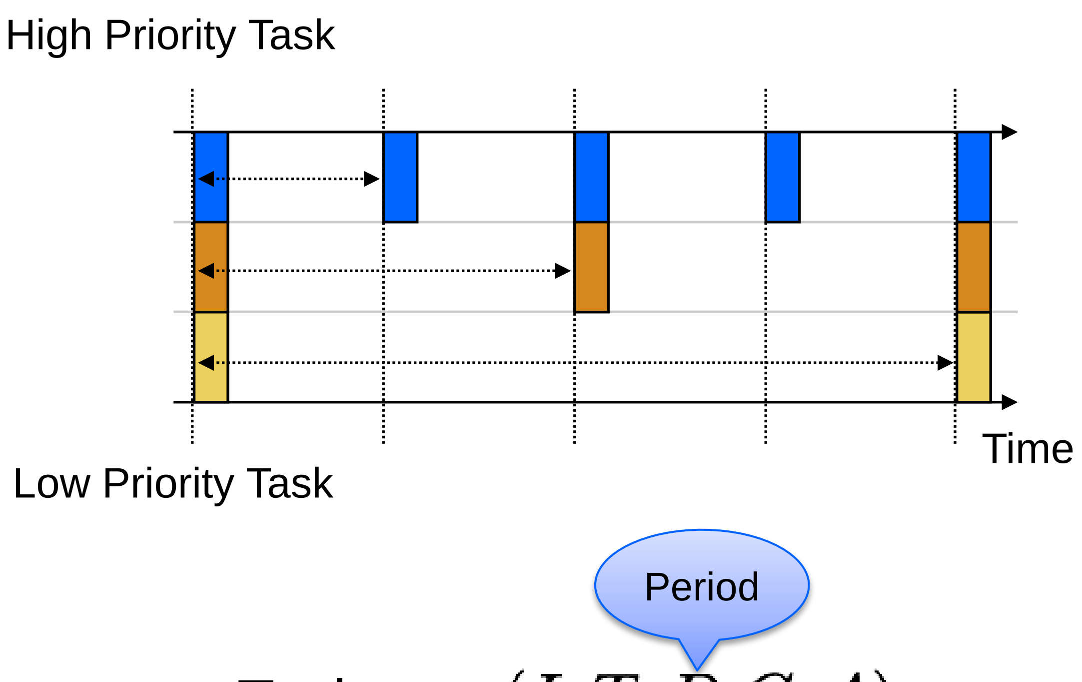
# Periodic Programs




Task body

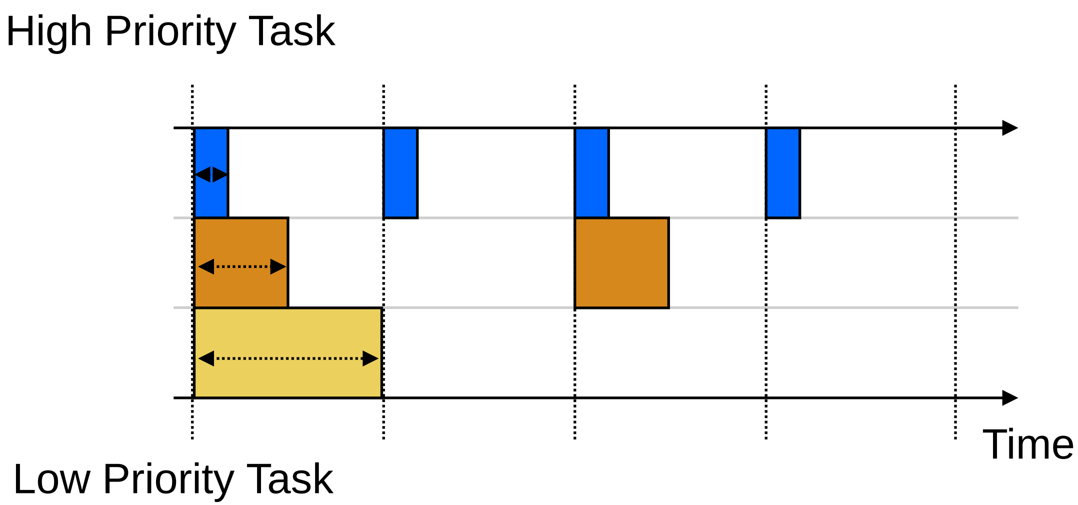
$$\text{Task } \tau = (I, T, P, C, A)$$

# Periodic Programs




Task $\tau = (I, T, P, C, A)$

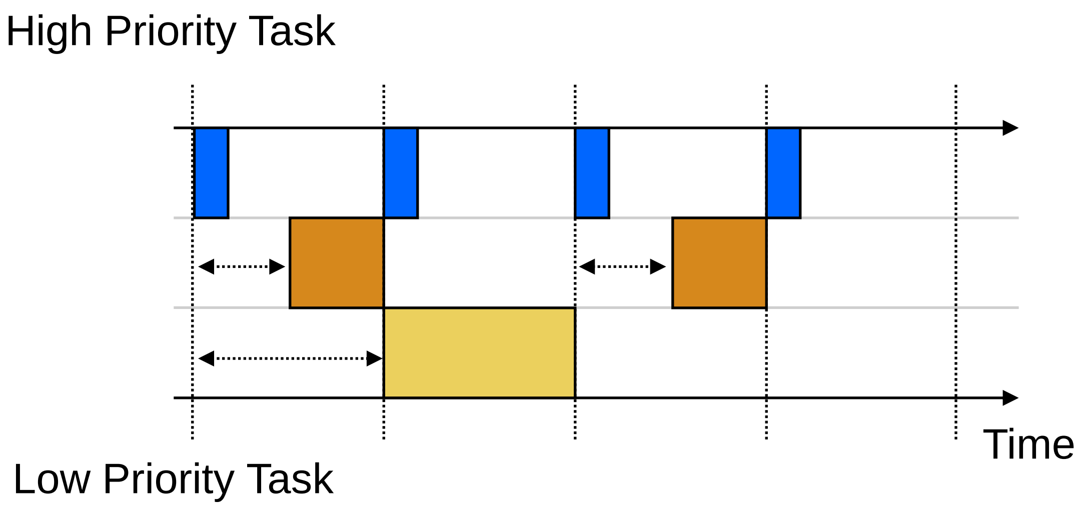
# Periodic Programs




WCET

Task $\tau = (I, T, P, C, A)$

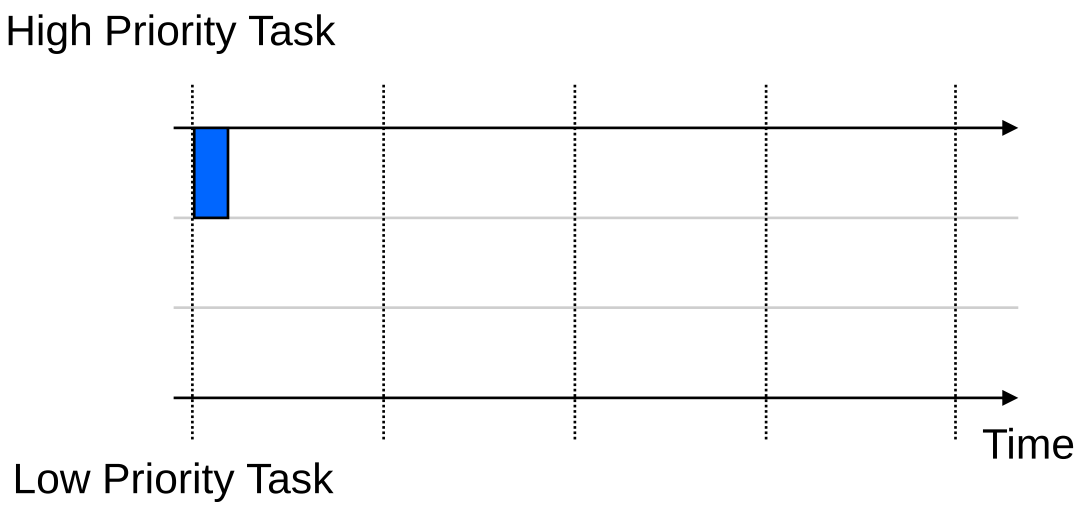
# Periodic Programs



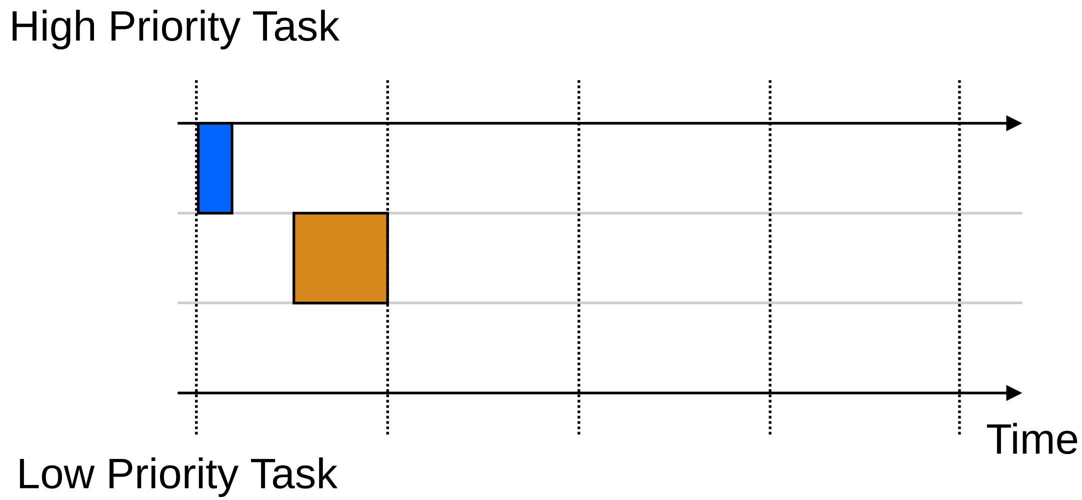

Arrival Time

Task $\tau = (I, T, P, C, A)$

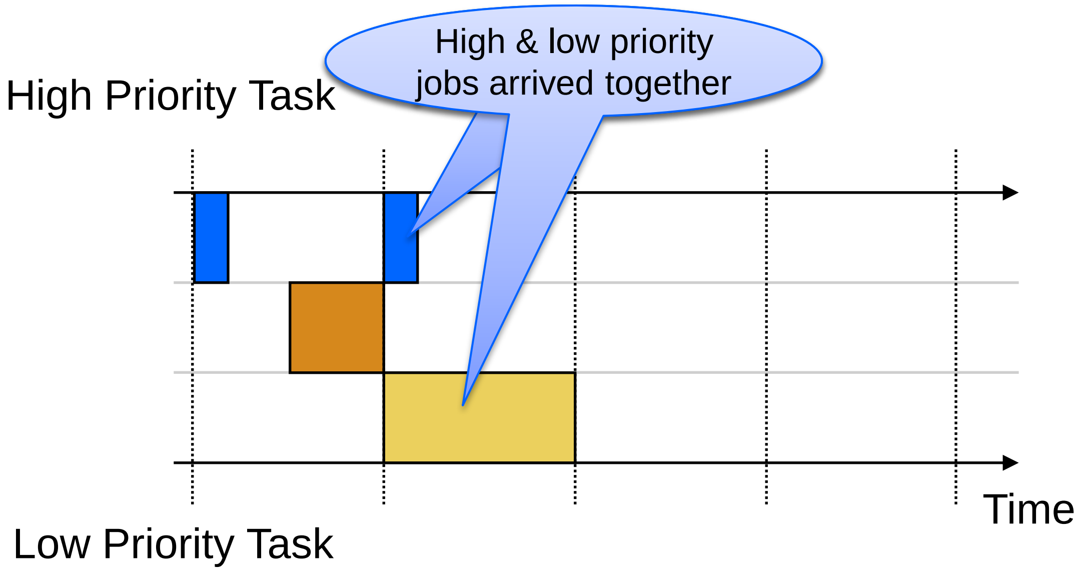
# Preemptive Fixed Priority Scheduling



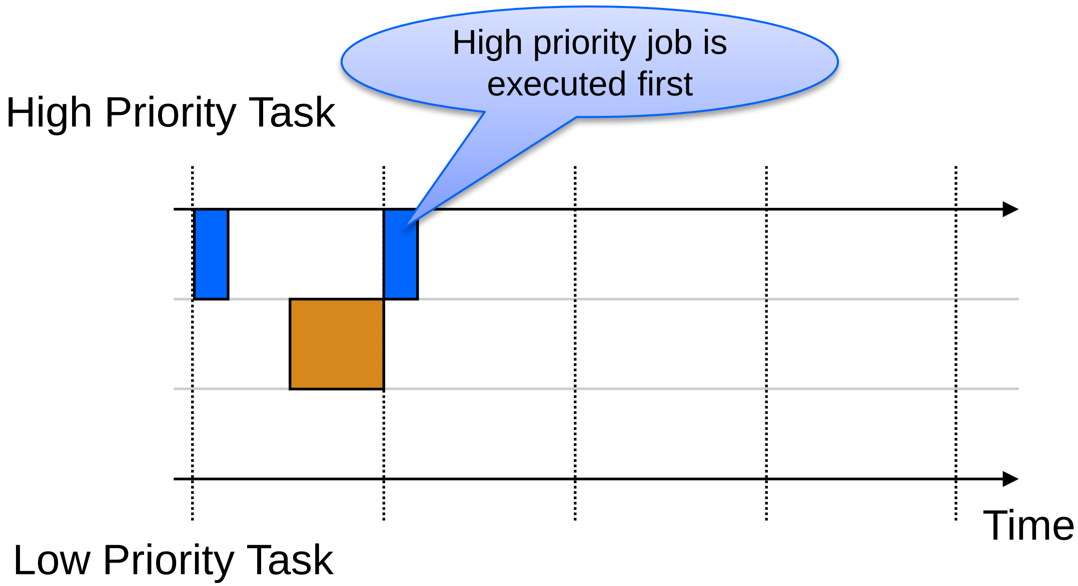
# Preemptive Fixed Priority Scheduling



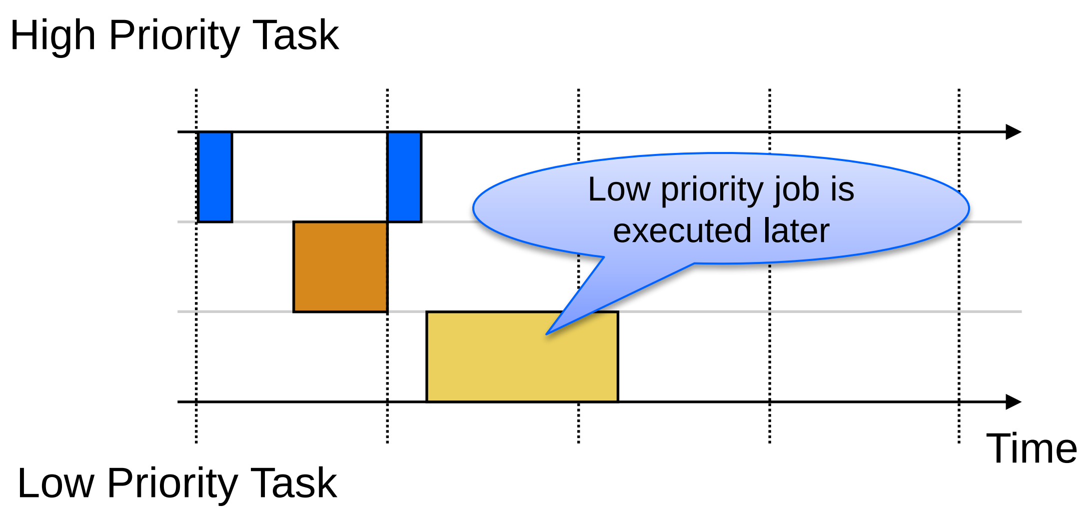
# Preemptive Fixed Priority Scheduling



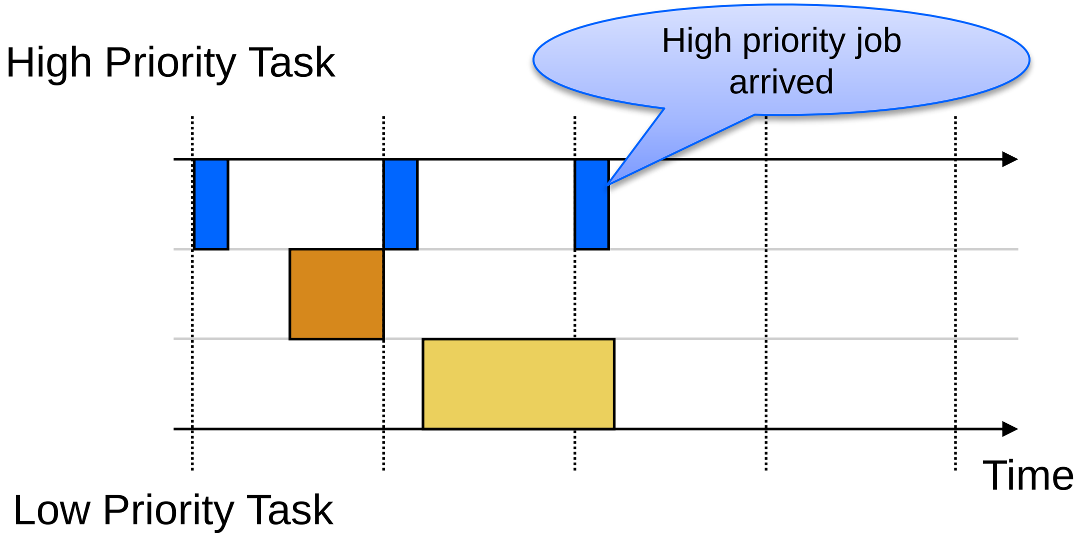
# Preemptive Fixed Priority Scheduling



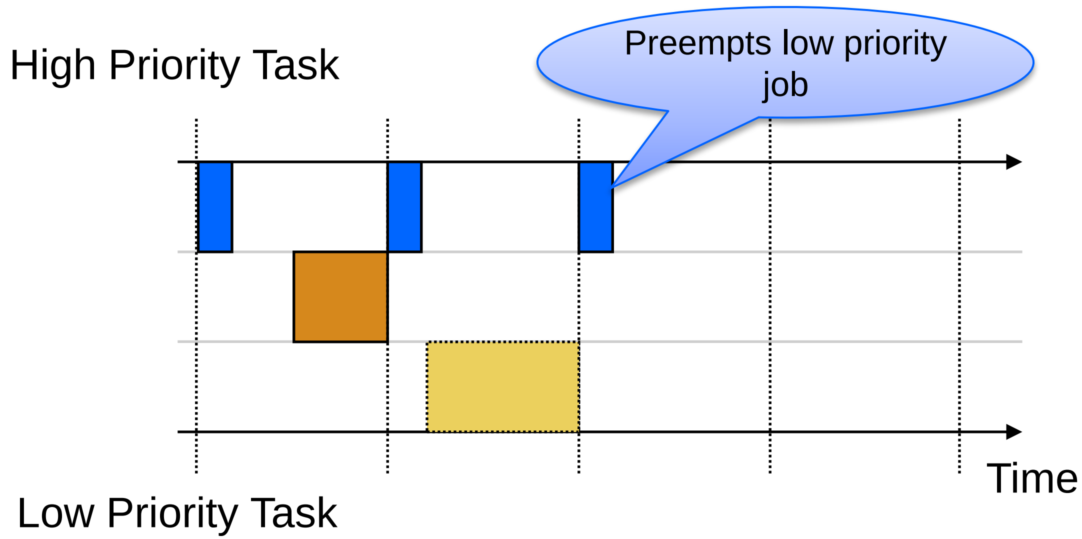
# Preemptive Fixed Priority Scheduling



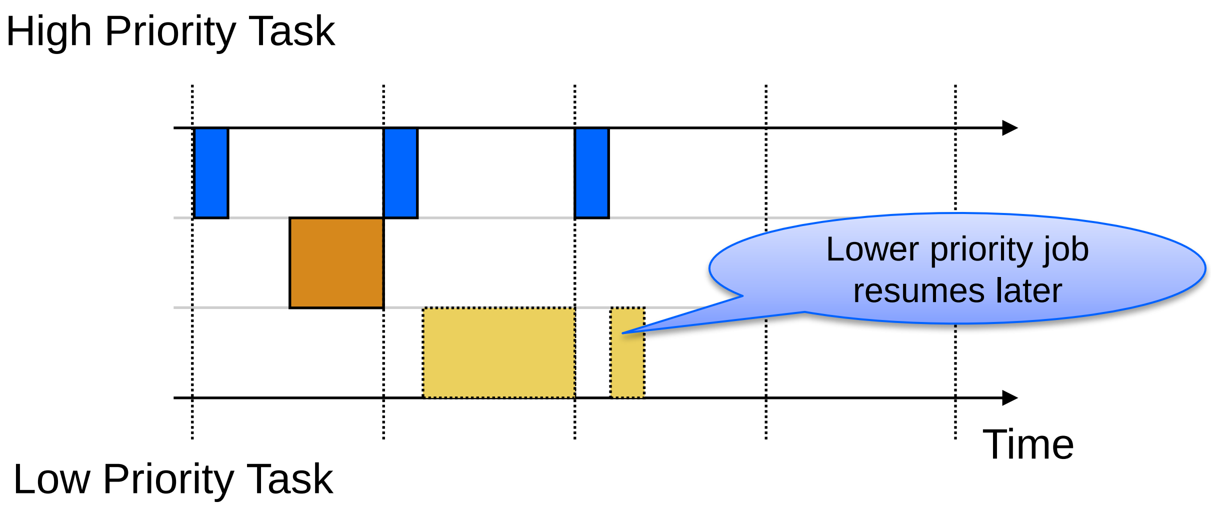
# Preemptive Fixed Priority Scheduling



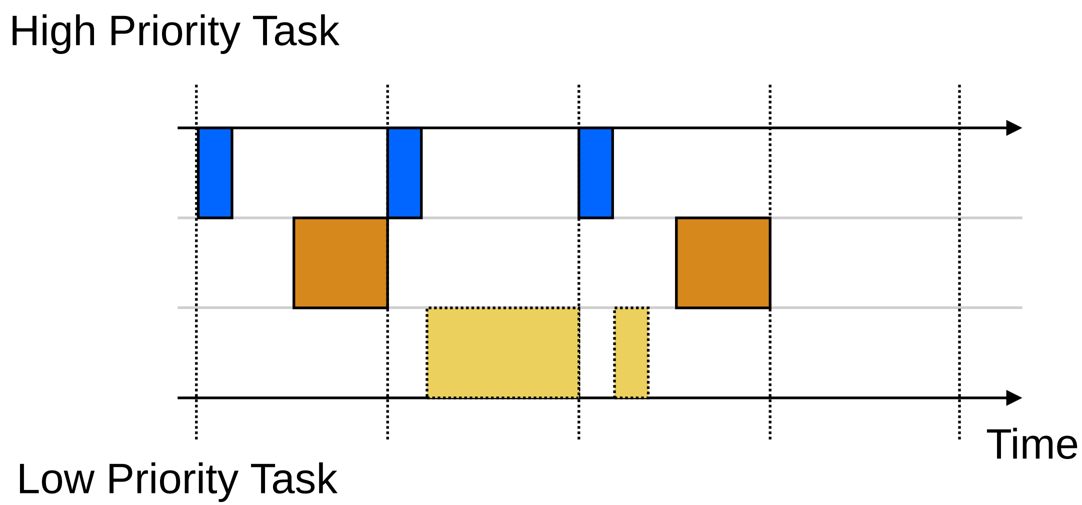
# Preemptive Fixed Priority Scheduling



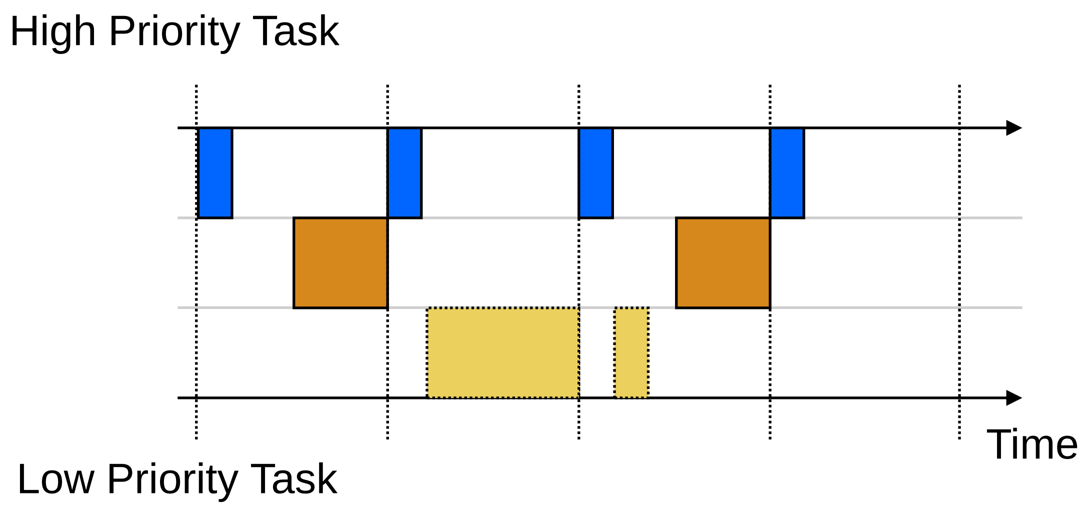
# Preemptive Fixed Priority Scheduling



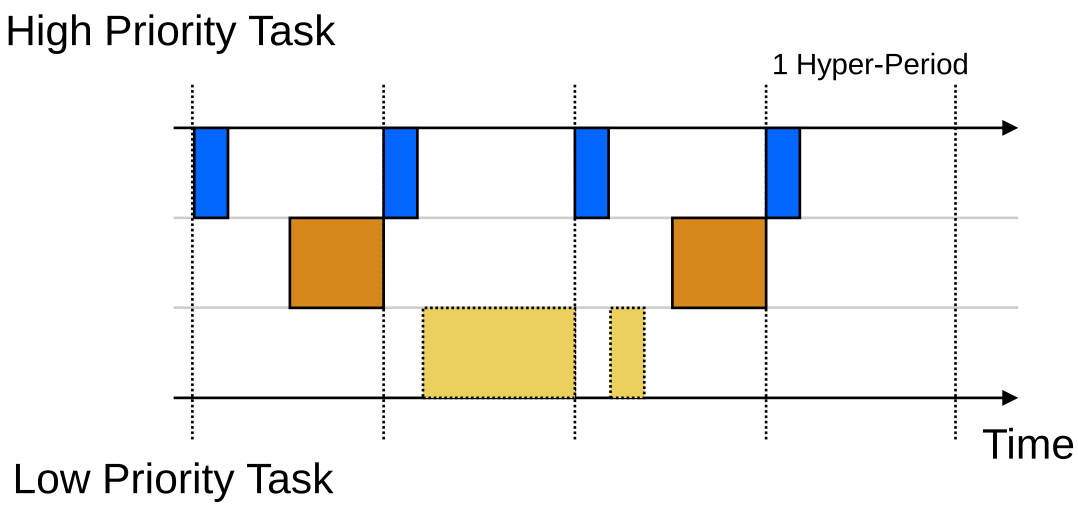
# Preemptive Fixed Priority Scheduling



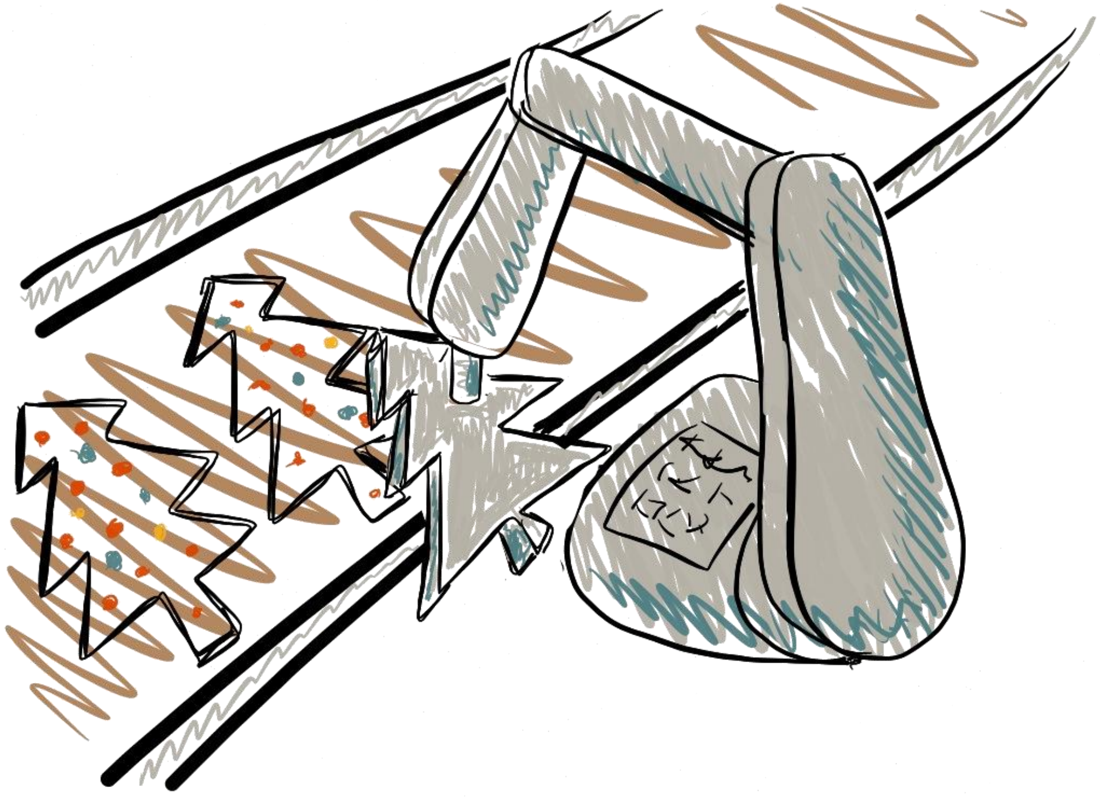
# Preemptive Fixed Priority Scheduling



# Preemptive Fixed Priority Scheduling



# Case Study: A Metal Stamping Robot

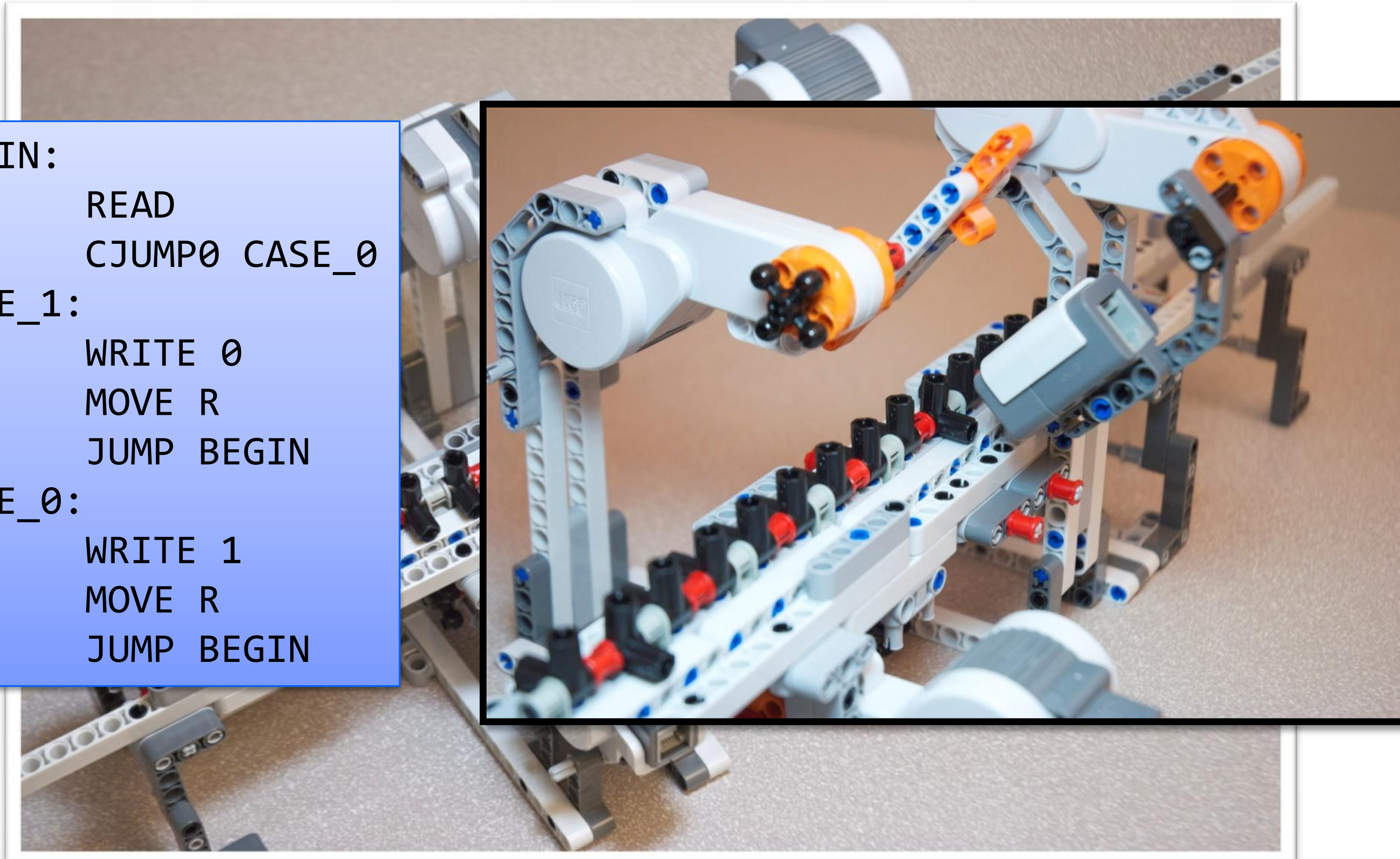


## a.k.a. LEGO Turing Machine

Image courtesy of Taras Kowaliw

# LEGO Turing Machine

```
BEGIN:
      READ
      CJUMP0 CASE_0
CASE_1:
      WRITE 0
      MOVE R
      JUMP BEGIN
CASE_0:
      WRITE 1
      MOVE R
      JUMP BEGIN
```



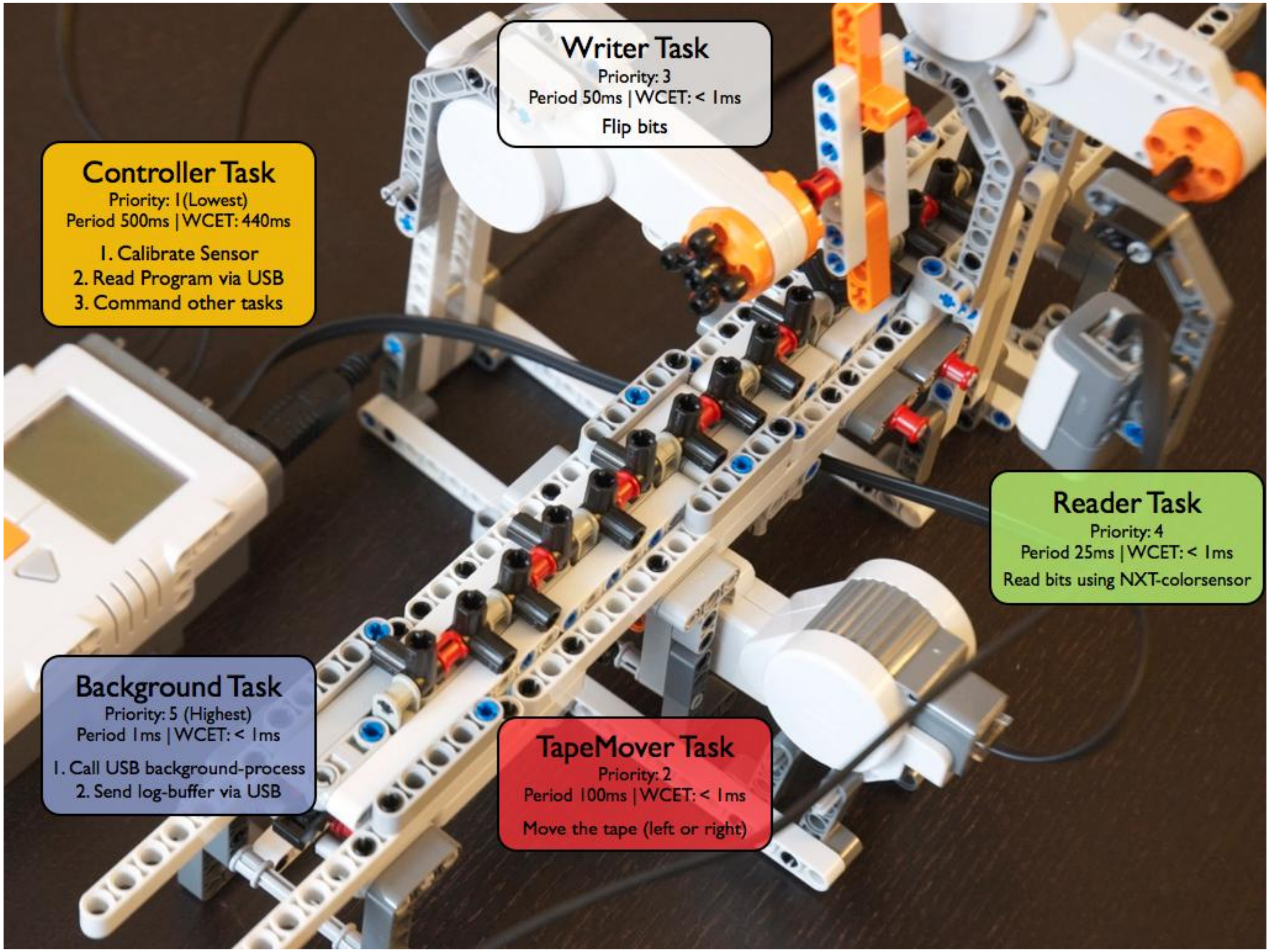
by Soonho Kong. See http://www.cs.cmu.edu/~soonhok for building instructions.

# OSEK/VDX RTOS

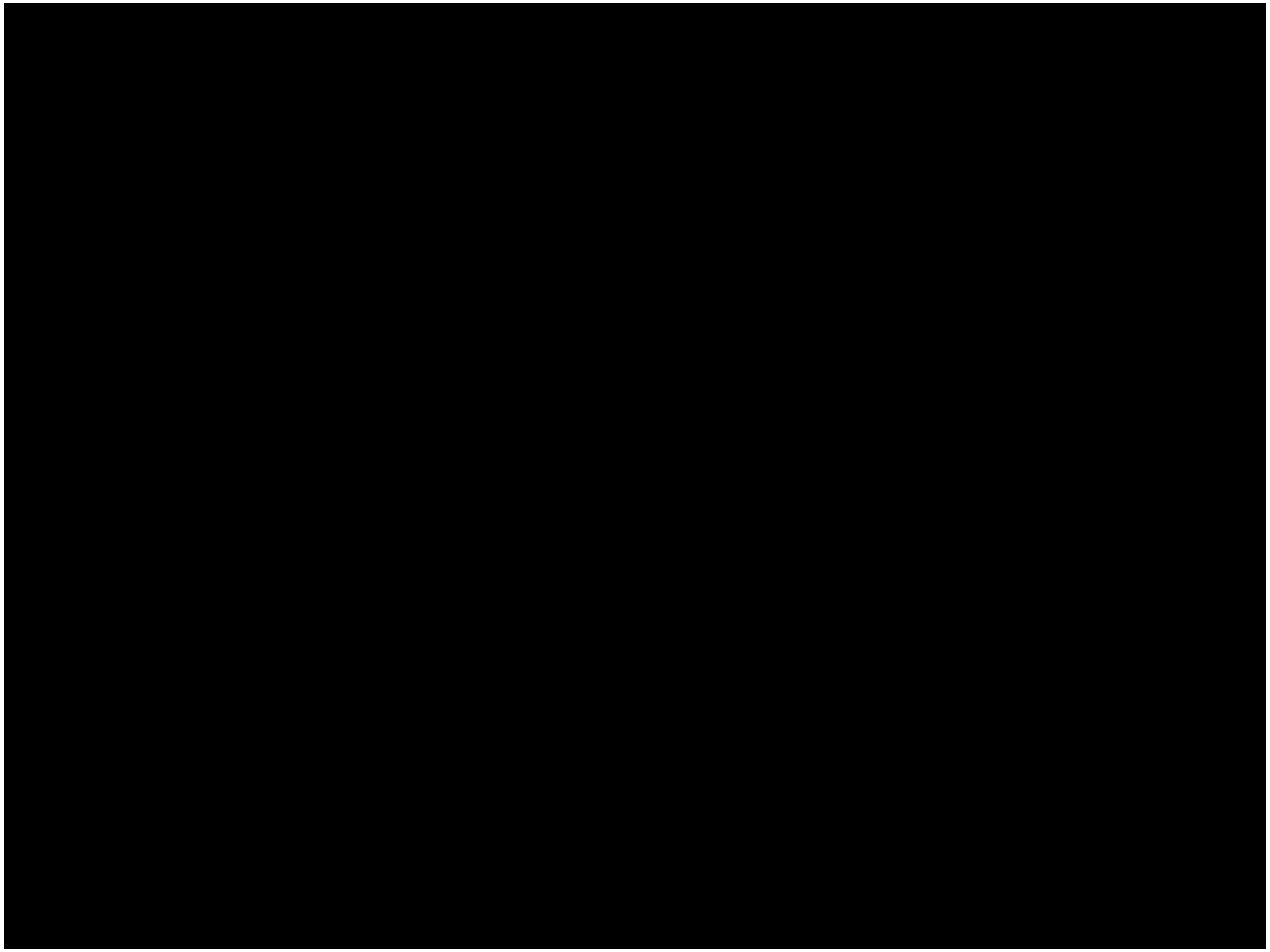
**O**ffene Systeme und deren **S**chnittstellen für die **E**lektronik in **K**raftfahrzeugen;
("Open Systems and their Interfaces for the Electronics in Motor Vehicles“)

standard software architecture for the electronic control units (ECUs) in a car

# Turing Machine: Task Structure



# Turing Machine (Video)



http://www.youtube.com/watch?v=teDyd0d5M4o

# Turing Machine: Properties

Tape does not move when a bit is read or written

Read sensor and Write arm can move concurrently but must not interfere with one another

Read sensor's light is off when not in use

Read task WCET is less than 25ms

- reduced to checking API usage rules

No log messages are lost during USB communication

- each message is delivered to the server before a new one is produced

# An Example Property

When writer flips a bit, the tape motor and read motor should **stop**.

Controller Task

```
case C_WRITE:

  /* Check if we need to chagne the bit */
  if(R(input) != R(output)) {
    /* Check the header and move it back if necessary */
    if(nxt_motor_get_count(READ_MOTOR) > 0 && R(R_state) == READ_IDLE) {
      W(R_state, READ_MOVE_HEADER_BACKWARD);
    }

    /* Check the header and flip the bit if it is safe to do */
    if(nxt_motor_get_count(READ_MOTOR) <= 0 && R(W_state) == WRITE_IDLE) {
      W(W_state, WRITE_FLIP);
    }
  } else {
    /* Nothing to change for writer */
    W(W_state, WRITE_IDLE);
    C_state = C_MOVE;
  }
  break;
```

Writer Task

```
  case WRITE_FLIP:
#ifdef VERIFICATION
    /* Property 3: When writer flips a bit, the tape motor and read
       motor should be stopped. */

    /* FAILED!! with BOUND 120 */
    assert(R(T_speed) == 0 && R(R_speed) == 0);
#endif
```

# Time-Bounded Verification of Periodic Programs

## Time-Bounded Verification

- Is an assertion A violated within X milliseconds of a system's execution from initial state I
  - A, X , I are user specified

## Periodic Program

- Collection of periodic tasks
  - Execute concurrently with fixed-priority scheduling
  - Priorities respect RMS
  - Communicate through shared memory
  - Synchronize through preemption and priority ceiling locks

## Assumptions

- System is schedulable
- WCET of each task is given

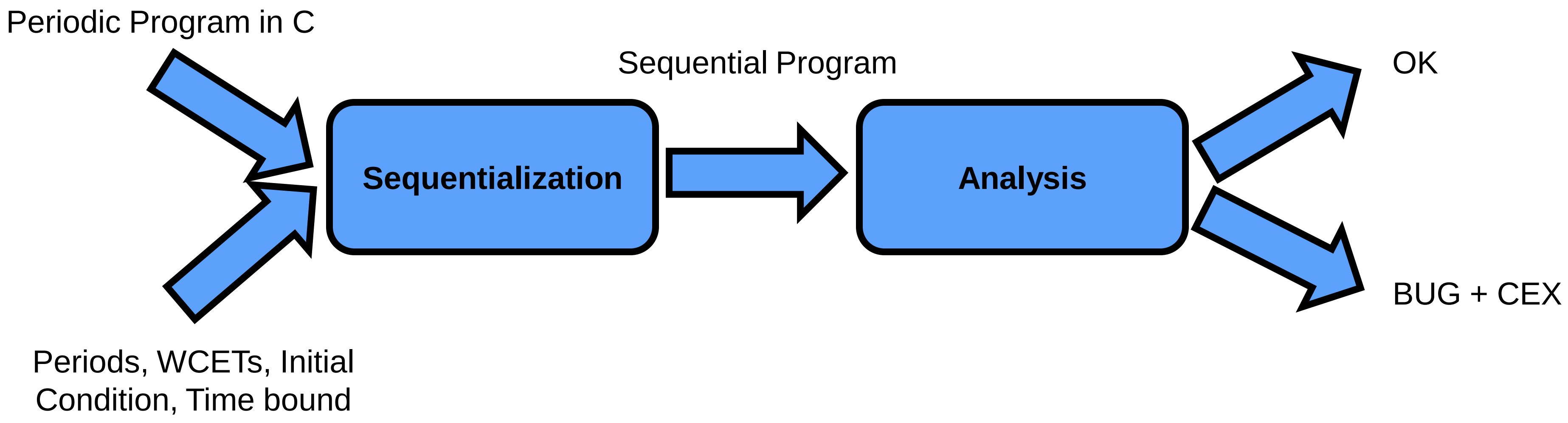
Time-Bounded Analysis of Real-Time Systems, Proc. of FMCAD 2011

# Overall Approach

Supports C programs w/ tasks, priorities, priority ceiling protocol, shared variables

Works in two stages:

1. *Sequentialization* – reduction to sequential program w/ *prophecy* variables
2. *Bounded program analysis*: bounded C model checker (CBMC, HAVOC, …)



# START Family

http://www.andrew.cmu.edu/~arieg/Rek

REK (2011)

- OSEK-based programs with Priority Ceiling locks
- small robotics case study

REK-H (2012)

- compositional sequentialization for harmonic tasks
- Turing Machine case study

REK-PIP (2013)

- Working on this now
- Support for Priority-Inheritance-Locks (difficult ☹)

REK-INF (Future)

- Extend to complete verification by finding inductive invariants

# Intellectually Defensible Base for Qualification

How should automatic verifiers be qualified for certification?

What is the base for automatic program analysis (or other automatic formal methods) to replace testing?

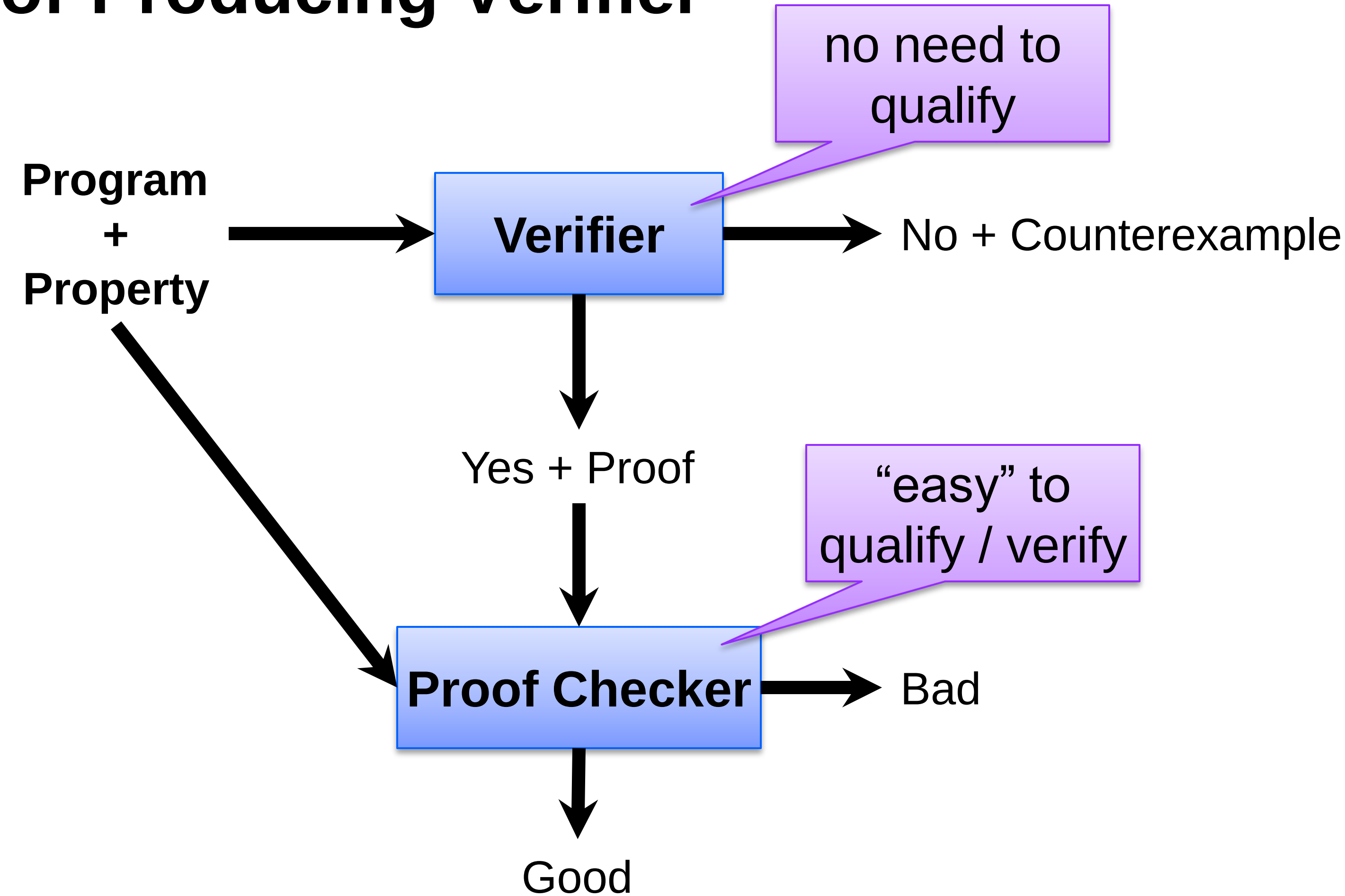
Verify the verifier

- (too) expensive
- verifiers are often very complex tools
- difficult to continuously adapt tools to project-specific needs

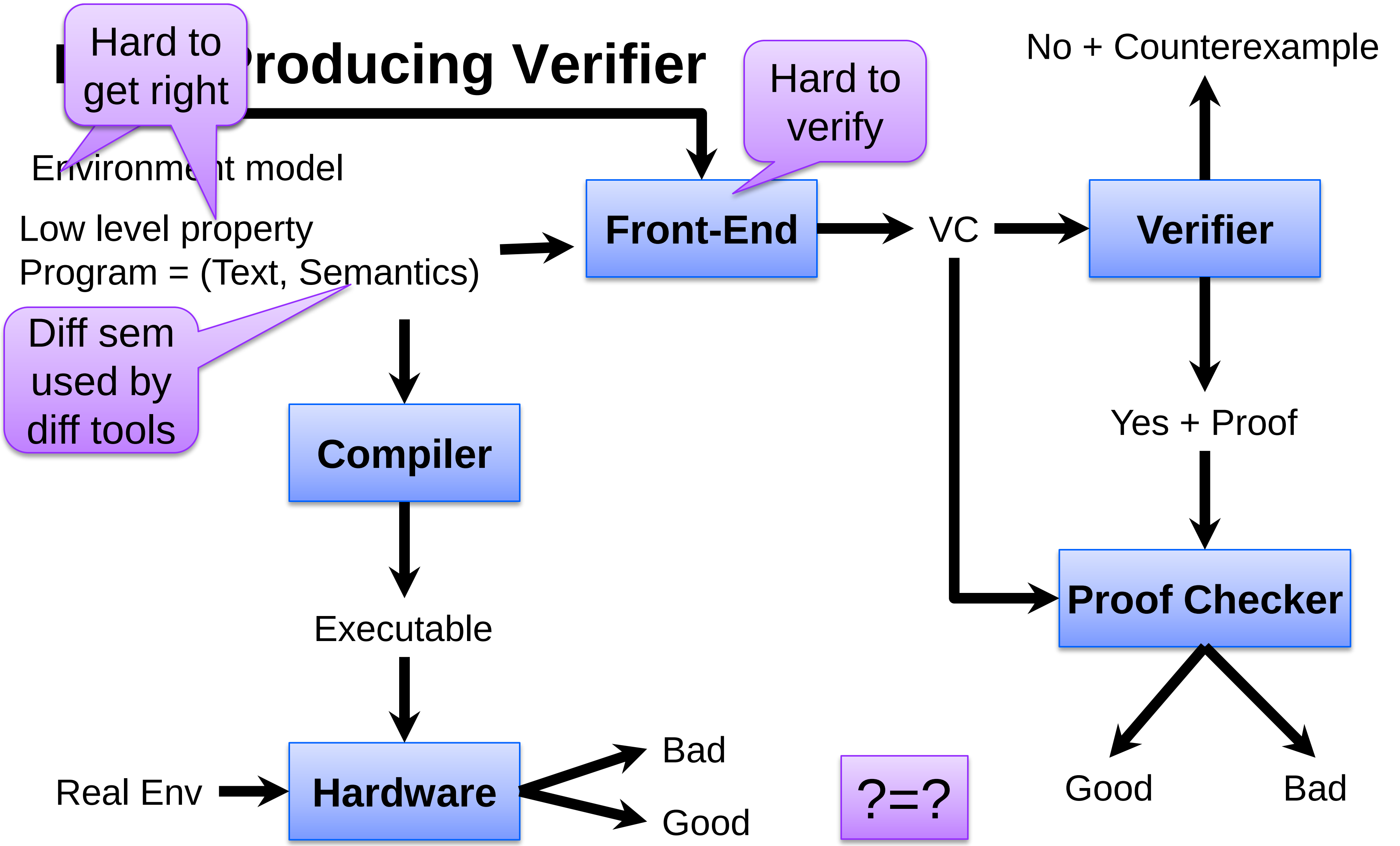
Proof-producing (or certifying) verifier

- Only the proof is important – not the tool that produced it
- Only the proof-checker needs to be verified/qualified
- Single proof-checker can be re-used in many projects

# Proof-Producing Verifier




But things are not that simple in practice !!!


Hard to get right
roducing Verifier
Hard to verify
No + Counterexample
Environment model
Low level property
Program = (Text, Semantics)
Front-End
VC
Verifier
Diff sem used by diff tools
Compiler
Yes + Proof
Proof Checker
Executable
Bad
Good
?=?
Good
Bad
Real Env
Hardware

# Contact Information

**Presenter**

Arie Gurfinkel

RTSS

Telephone:  +1 412-268-5800

Email:  arie@cmu.edu

**U.S. mail:**

Software Engineering Institute

Customer Relations

4500 Fifth Avenue

Pittsburgh, PA 15213-2612

USA

**Web:**

www.sei.cmu.edu

http://www.sei.cmu.edu/contact.cfm

**Customer Relations**

Email: info@sei.cmu.edu

Telephone: +1 412-268-5800

SEI Phone: +1 412-268-5800

SEI Fax: +1 412-268-6257