# Timing analysis of critical real-time systems

# Safety- and time-critical systems

## Safety

"*freedom from those conditions that can cause death, injury, illness, damage or loss of equipment or property, or environmental harm* »
Software Quality Assurance Handbook, 4th ed., 2008

- Safety is not reliability (probability that a system will perform its intended function satisfactorily)
- Safety is not security (protection or defense against attack, interference, or espionage)

# Safety- and time-critical systems

## Safety-critical software

"*software whose use in a system can result in unacceptable risk*"
IEEE Standard Glossary of Software Engineering Terminology, 1990

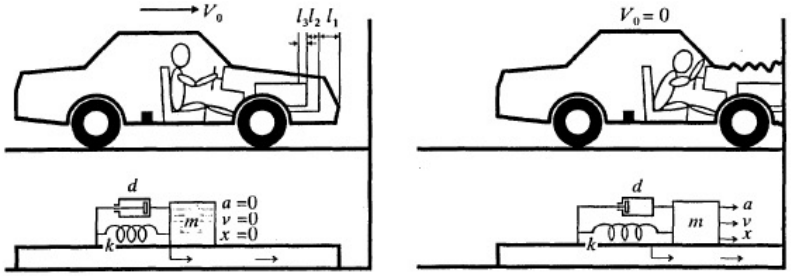
- requires failure mitigation
- possible failures:
  - hardware
  - software
    - functional
    - timing

## Real-time systems:

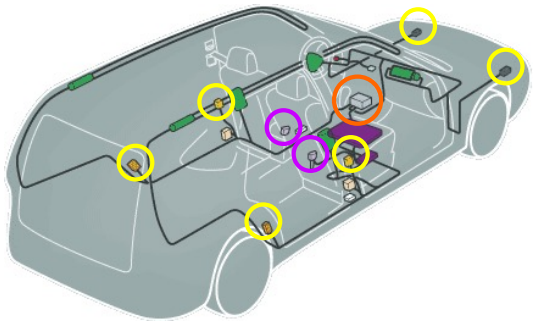
systems that are subject to timing deadlines

- missing a deadline is a failure

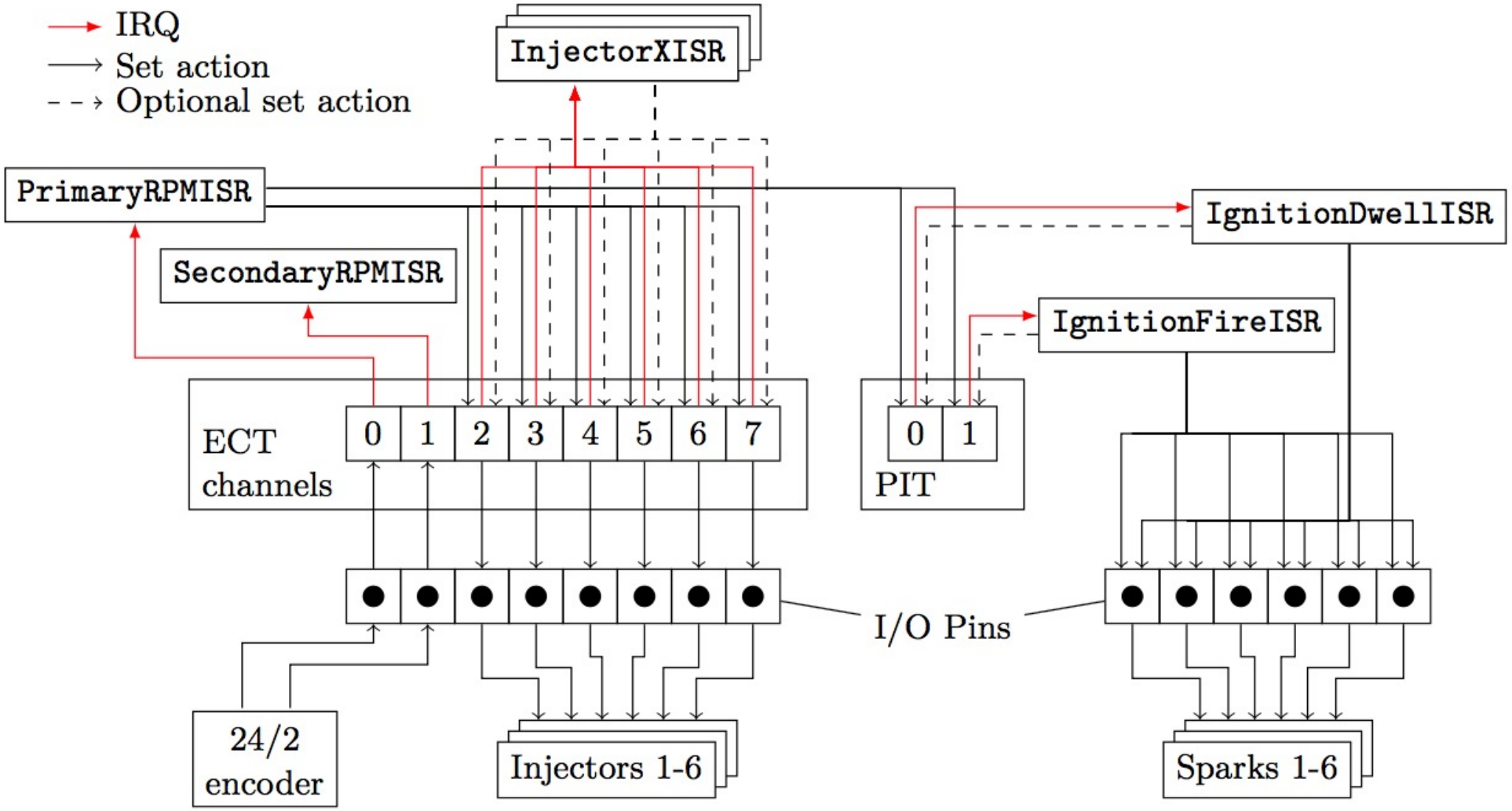
# Example 1: airbag controller



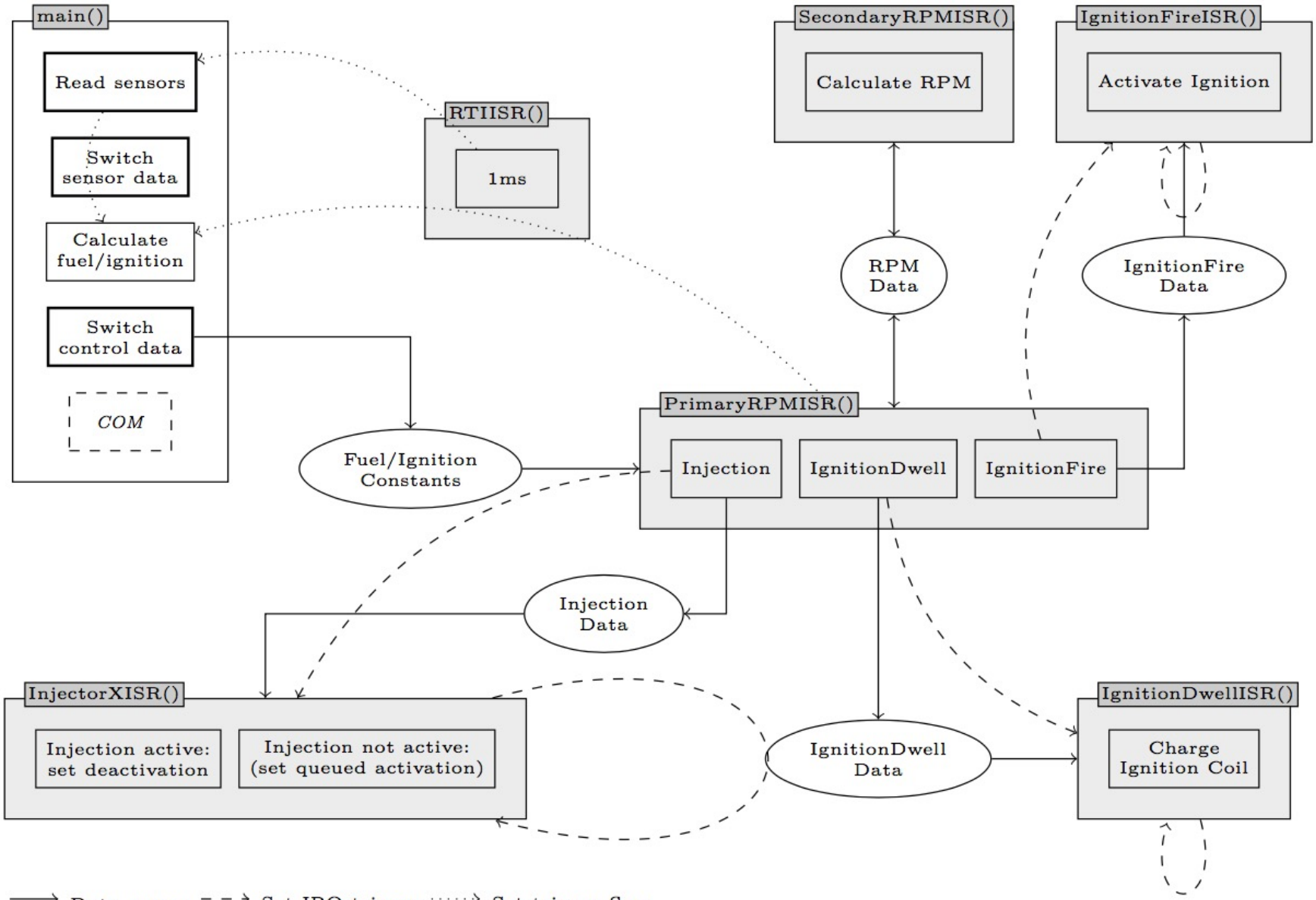




# Example 2: car engine controller

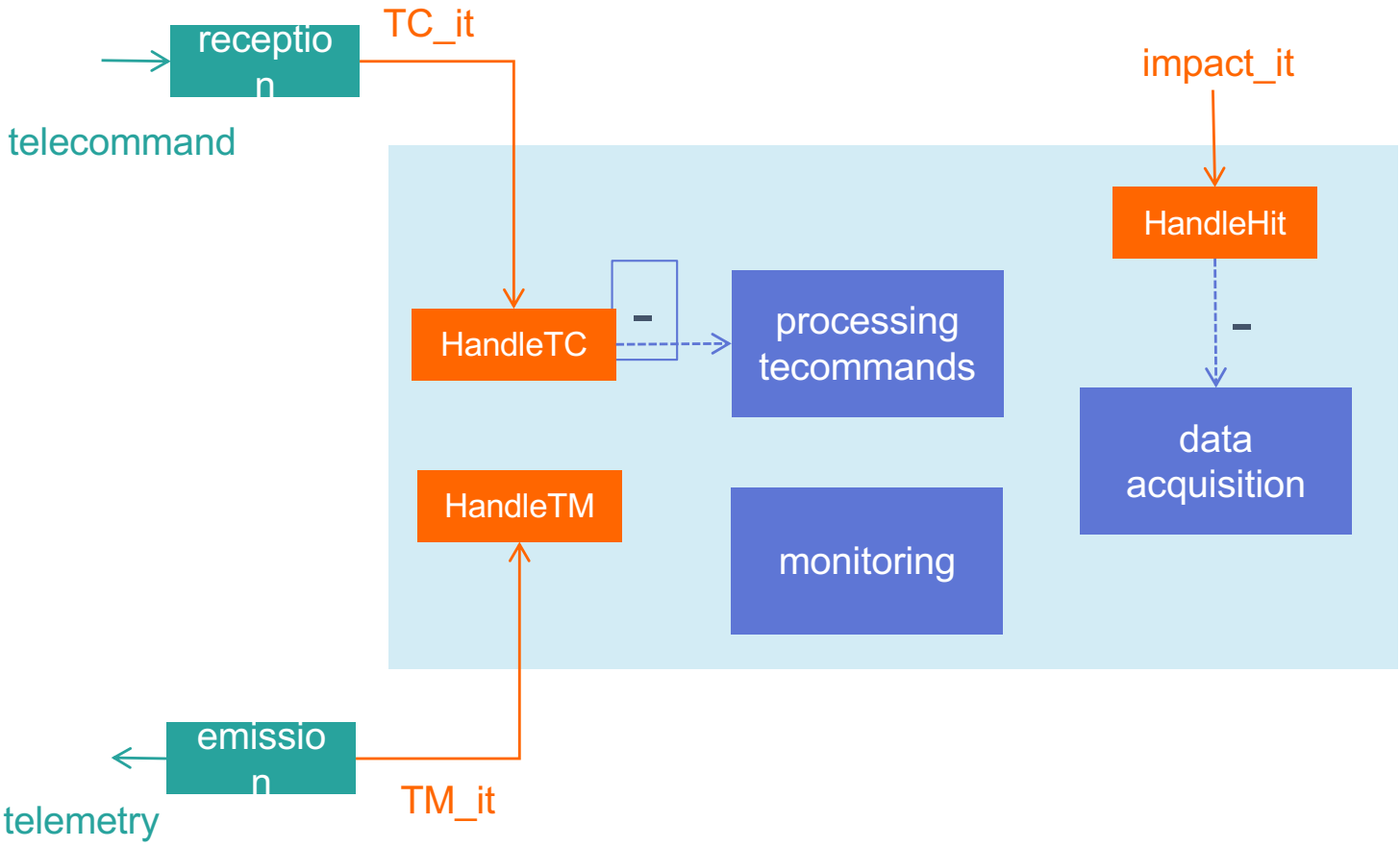


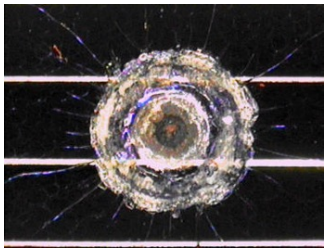
# Example 2: car engine controller

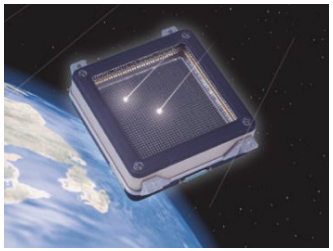


# Example 3: a device in aerospace

## DEBIE: DEBris In orbit Evaluator

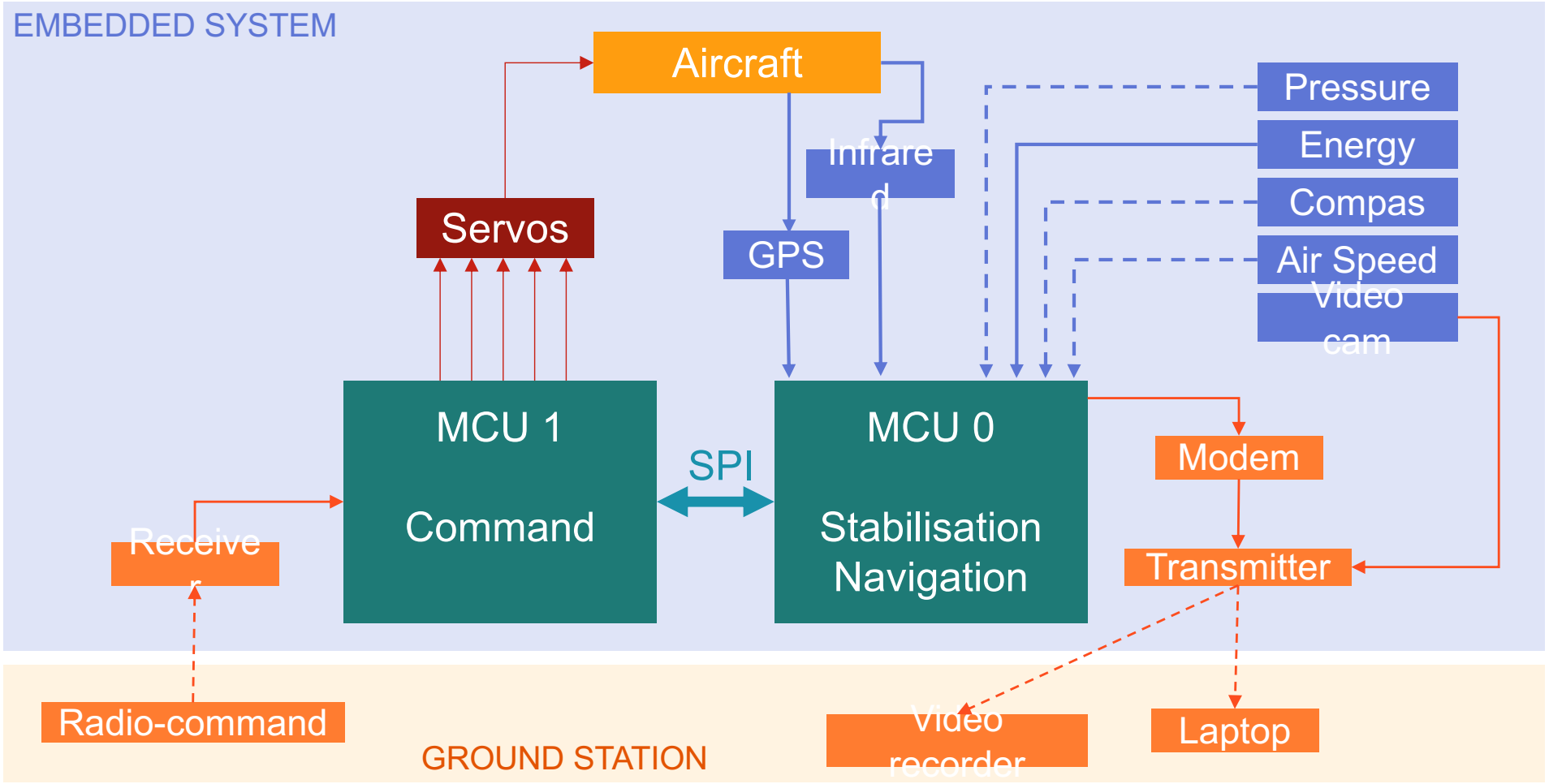




# Example 4: UAV software

## PapaBench (inspired from the Paparazzi project)



# Example 4: UAV software

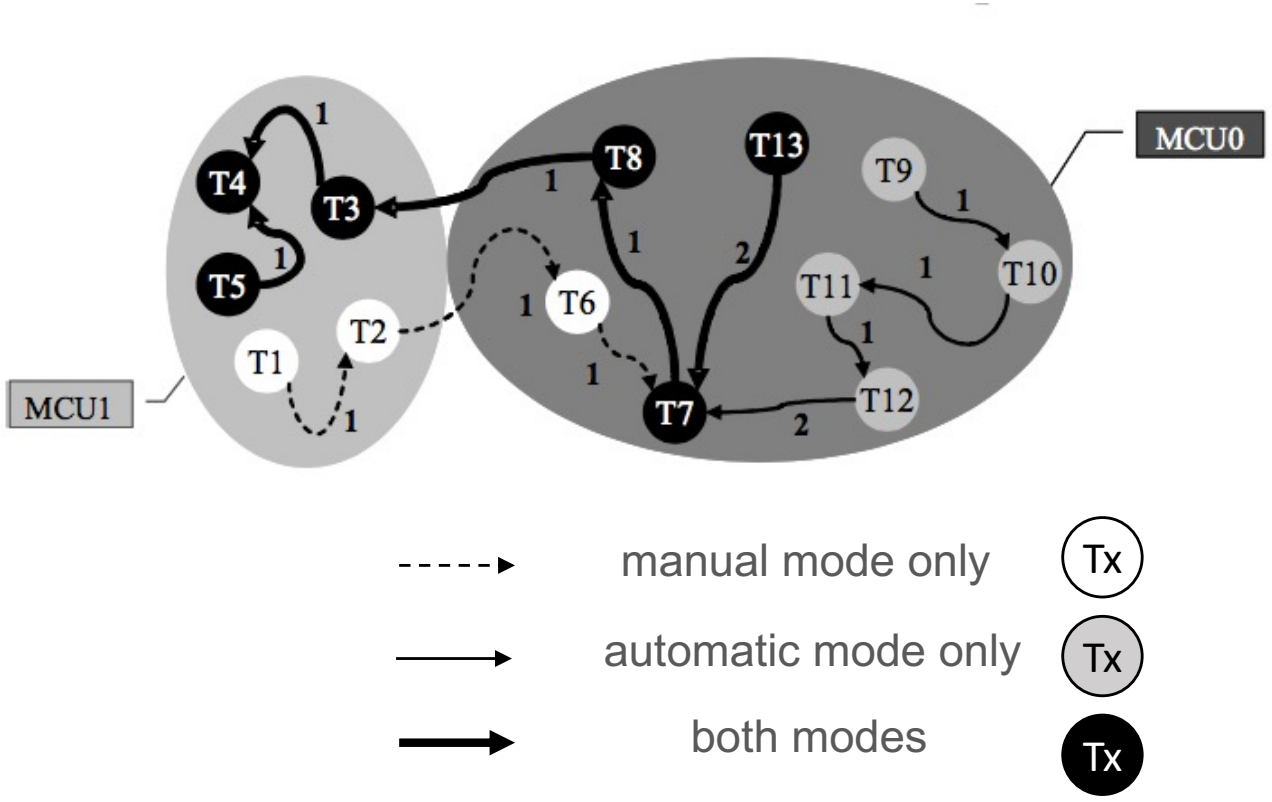
## PapaBench (inspired from the Paparazzi project)

| ID | Description | Frequency |
|---|---|---|
| *T1* | Receive Radio-Command orders | 40Hz |
| *T2* | Send Data to MCU0 | 40Hz |
| *T3* | Receive MCU0 values | 20Hz |
| *T4* | Transmit Servos | 20Hz |
| *T5* | Check Failsafe | 20Hz |
| *I1* | Transmission Servos interrupt | - |
| *I2* | SPI interrupt of MCU1 | - |
| *I3* | Radio interrupt | - |

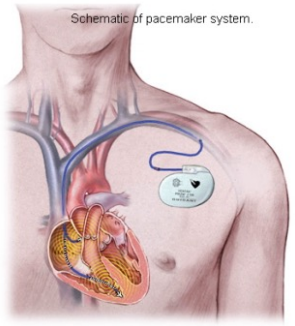
***Table 1:*** *MCU1 tasks and interrupts*

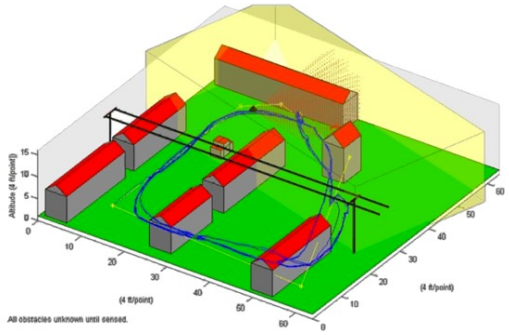
| ID | Description | Frequency |
|---|---|---|
| *T6* | Managing Radio orders | 40Hz |
| *T7* | Stabilization | 20Hz |
| *T8* | Send Data to MCU1 | 20Hz |
| *T9* | Receive GPS Data | 4Hz |
| *T10* | Navigation | 4Hz |
| *T11* | Altitude Control | 4Hz |
| *T12* | Climb Control | 4Hz |
| *T13* | Reporting Task | 10Hz |
| *I4* | SPI interrupt of MCU0 | - |
| *I5* | Modem interrupt | - |
| *I6* | GPS interrupt | - |

***Table 2:*** *MCU0 tasks and interrupts*



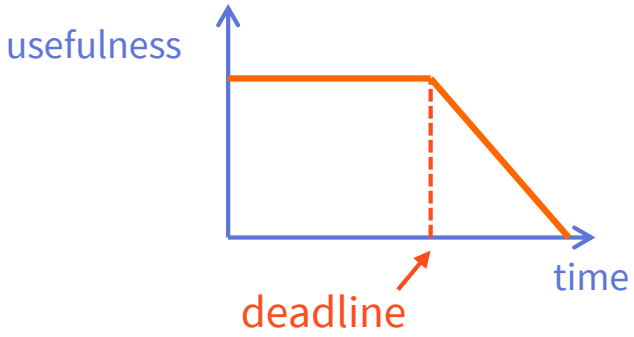
# Time-critical applications



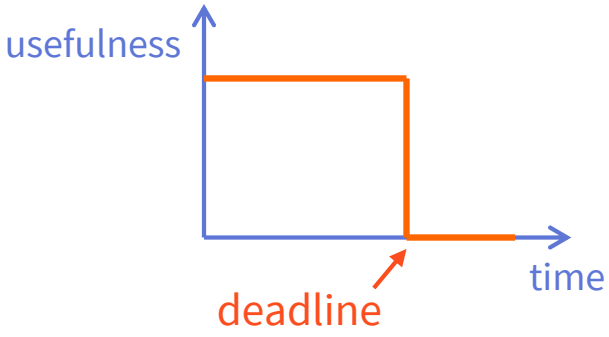



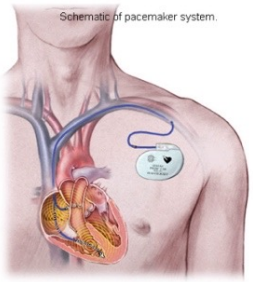
# Time criticity

soft real-time

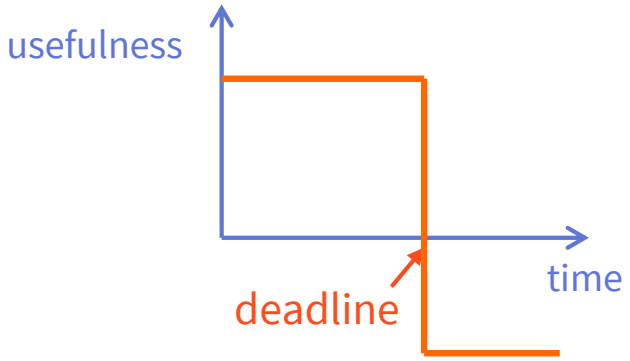


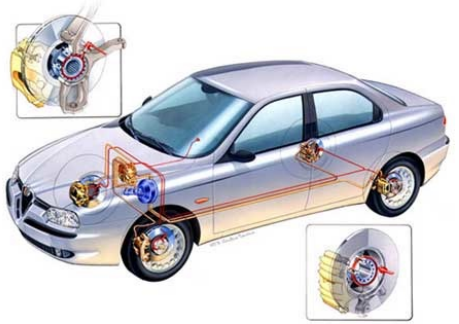





hard real-time

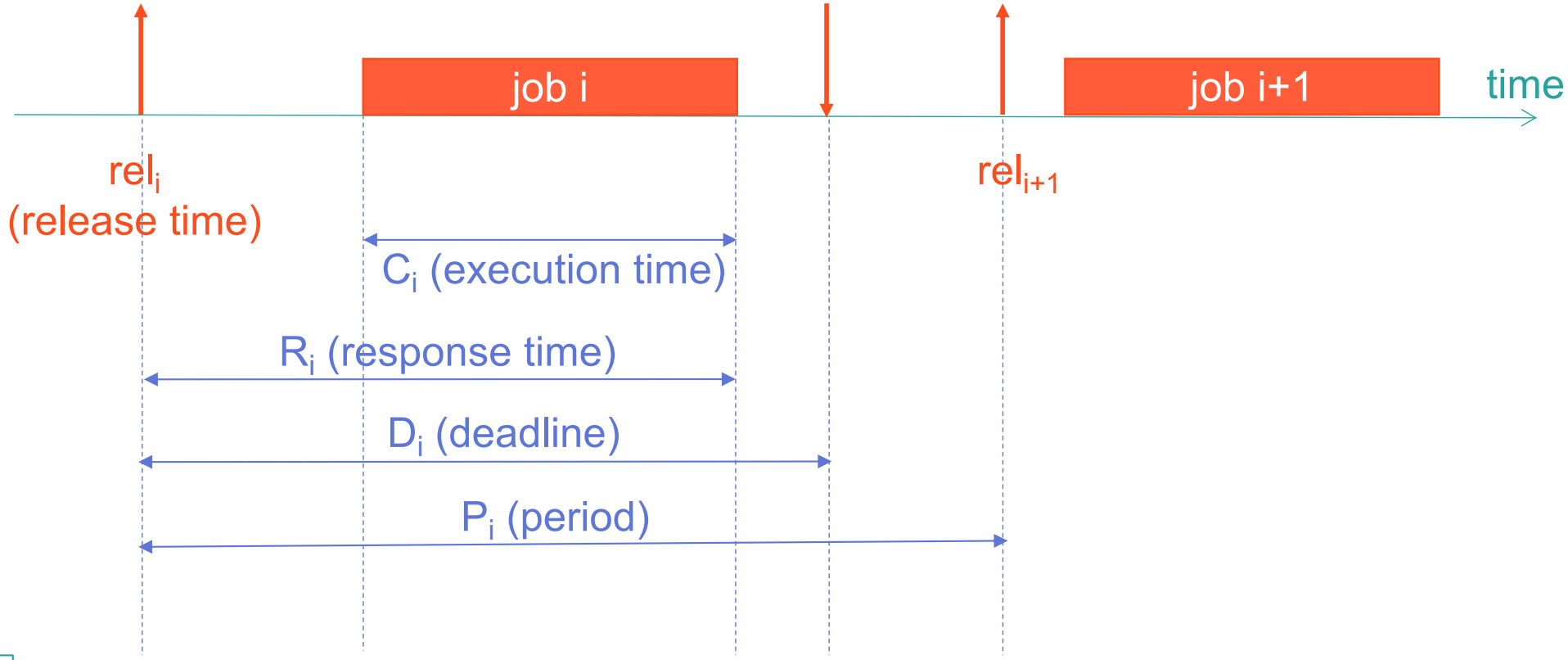


# Real-time scheduling

## Task model

- periodic, sporadic, aperiodic



# Real-time scheduling

## Objective

- allocating shared resources to tasks so that they all meet their deadlines

## Approaches

- off-line vs. online
- fixed vs. dynamic priorities
  - Rate Monotonic, Deadline Monotonic, Earliest Deadline First
- preemptive vs. non-preemptive
- on a multicore platform, global vs. partitioned

# Real-time scheduling

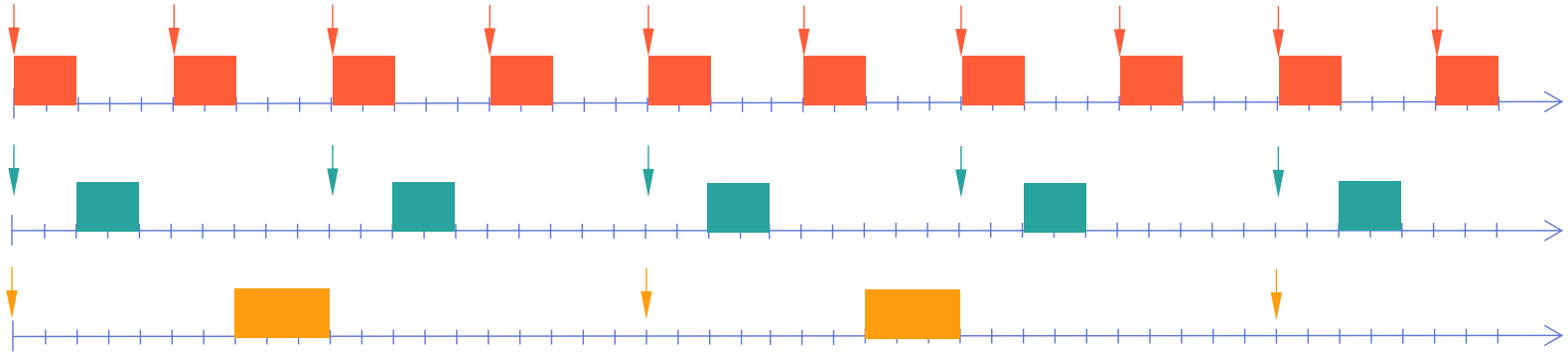
## Schedulability test

- example: non-preemptive EDF (Earliest Deadline First)

$$\sum_{i=1}^{n} \frac{C_i}{P_i} \leq 1$$

$$C_i + \sum_{j=1}^{i-1} \left\lfloor \frac{L-1}{P_j} \right\rfloor \times C_j \leq L \quad with \quad 1 < i \leq n \quad and \quad P_1 < L \leq P_i$$

| | $C_i$ | $P_i=D_i$ |
|---|---|---|
| $\tau_1$ | 2 | 5 |
| $\tau_2$ | 2 | 10 |
| $\tau_3$ | 3 | 20 |



# Variability of execution times

## Reasons for variability

- input data

```
index = 0;
while (data_in[index] != 0){
     /* processing */
}
```

how many iterations?

```
if (temperature > T_MAX){
     /* error processing */
}
else{
     /* normal processing */
}
```

which branch?

- initial hardware state

```
sum = 0;
for (i=0 ; i<100 ; i++)
     sum += a[i];
```

data cache contents?

- interferences with other tasks

on a miss, latency to the main memory?

# Execution time: example 1

## What is the execution time of this program?

source code

```
sum = 0;
num_even = 0;
for (i=0 ; i<100 ; i++)
{
   sum += i;
   if (i % 2 == 0)
      num_even++;
}
```

compiled code

```
        mov   r0,#0      @ r0:sum
        mov   r1,#0      @ r1:num_even
        mov   r2,#0      @ r2:i
for:    cmp   r2,#100
        bcs   end
        add   r0,r0,r2
if:     and   r3,r2,#1
        cmp   r3,#0
        bne   next
        add   r1,r1,#1
next:   add   r2,r2,#1
        b     for
end:
```

Assumptions:

- each instruction is executed in 1 cycle
  - 3 cycles if it follows a taken branch
- no pipeline

# Execution time: example 1

What is the execution time of this program?

```
        mov   r0,#0
        mov   r1,#0
        mov   r2,#0
for:    cmp   r2,#100
        bcs   fin
        add   r0,r0,r2
if:     and   r3,r2,#1
        cmp   r3,#0
        bne   suite
        add   r1,r1,#1
next:   add   r2,r2,#1
        b     for
end:
```

# Execution time: example 2

## What is the execution time of this program?

source code

```
sum = 0;
num_even = 0;
for (i=0 ; i<N ; i++)
{
   sum += a[i];
   if (a[i] % 2 == 0)
      num_even ++;
}
```

compiled code

```
        mov   r0,#0       @ r0:sum
        mov   r1,#0       @ r1:num_even
        mov   r2,#0       @ r2:i
        adr   r10,a
        adr   r11,N
        ldr   r11,[r11] @ r11:N
for:    cmp   r2,r11
        bcs   end
        ldr   r4,[r10,r2,LSL #2]
        add   r0,r0,r4
if:     and   r3,r4,#1
        cmp   r3,#0
        bne   next
        add   r1,r1,#1
next:   add   r2,r2,#1
        b     for
end:
```

Assumptions:

- each instruction is executed in 1 cycle
  - 3 cycles if it follows a taken branch
- no pipeline

# Execution time: example 2

What is the execution time of this program?

```
        mov   r0,#0
        mov   r1,#0
        mov   r2,#0
        adr   r10,a
        adr   r11,N
        ldr   r11,[r11]
for:    cmp   r2,r11
        bcs   end
        ldr   r4,[r10,r2,LSL #2]
        add   r0,r0,r4
if:     and   r3,r4,#1
        cmp   r3,#0
        bne   next
        add   r1,r1,#1
next:   add   r2,r2,#1
        b     for
end:
```

# Execution time: example 2

Can we estimate the maximum execution time of the program?

- we need to know the maximum value of N
- the iteration counts of loops must be upper-bounded

Assumptions:

- N ≤ 10
  - in the worst case, the loop has 10 iterations
- values of a[ ] ?
  - we will consider the most unfavourable situation (see code)

# Execution time: example 2

What is the (maximum) execution time of this program?

```
        mov   r0,#0
        mov   r1,#0
        mov   r2,#0
        adr   r10,tab
        adr   r11,N
        ldr   r11,[r11]
for:    cmp   r2,r11
        bcs   end
        ldr   r4,[r10,r2,LSL #2]
        add   r0,r0,r4
if:     and   r3,r4,#1
        cmp   r3,#0
        bne   next
        add   r1,r1,#1
next:   add   r2,r2,#1
        b     for
end:
```

# Execution time: example 2

## Possible values of the execution time?

- for a given value of N, and n even array elements ($0 \leq n \leq N$):

$$T = 6 + 2 + 4\times N + 5\times N + 3\times n + 4\times(N-n)$$

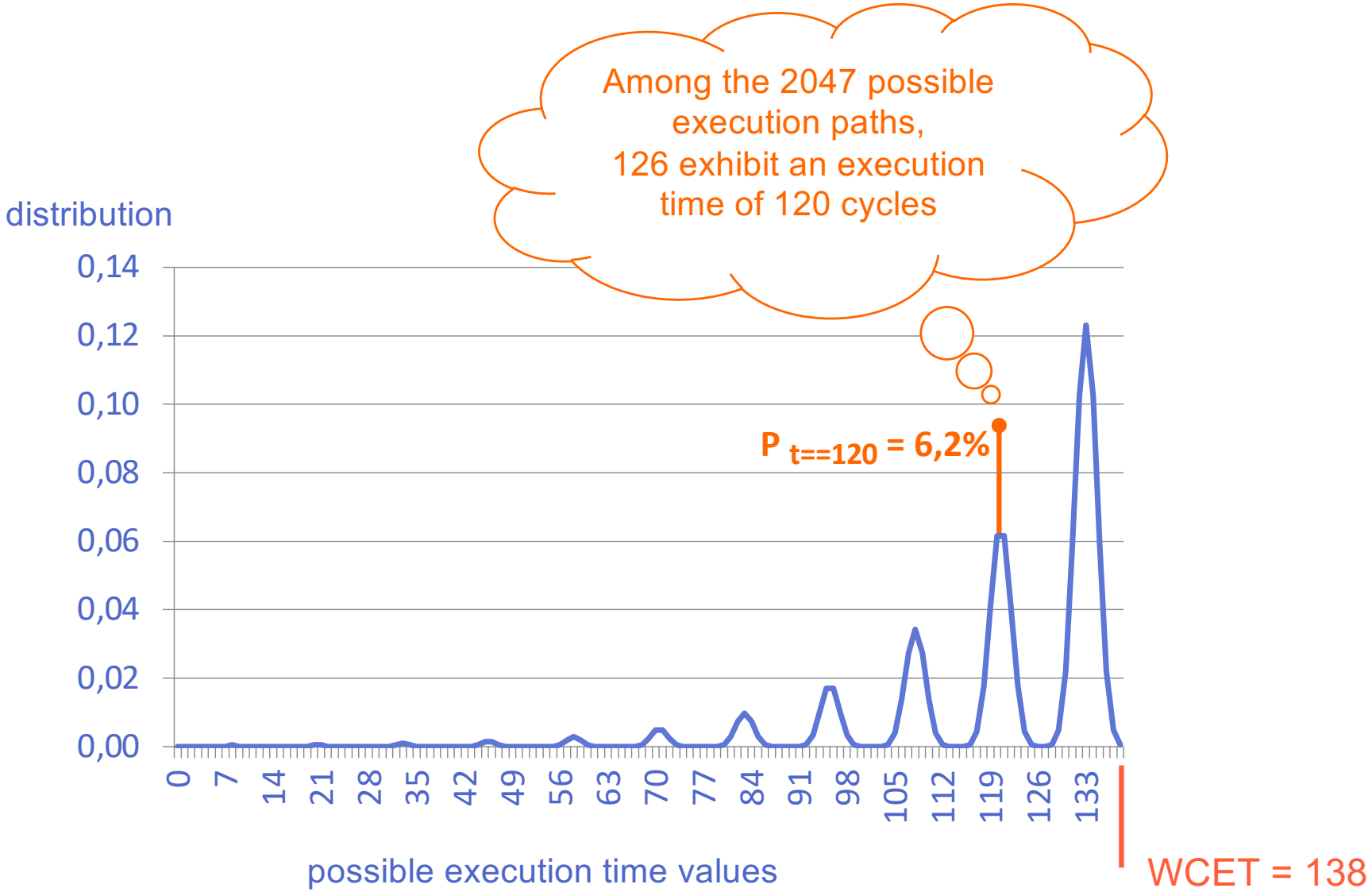
$$T = 8 + 13\times N - n$$

## Examples :

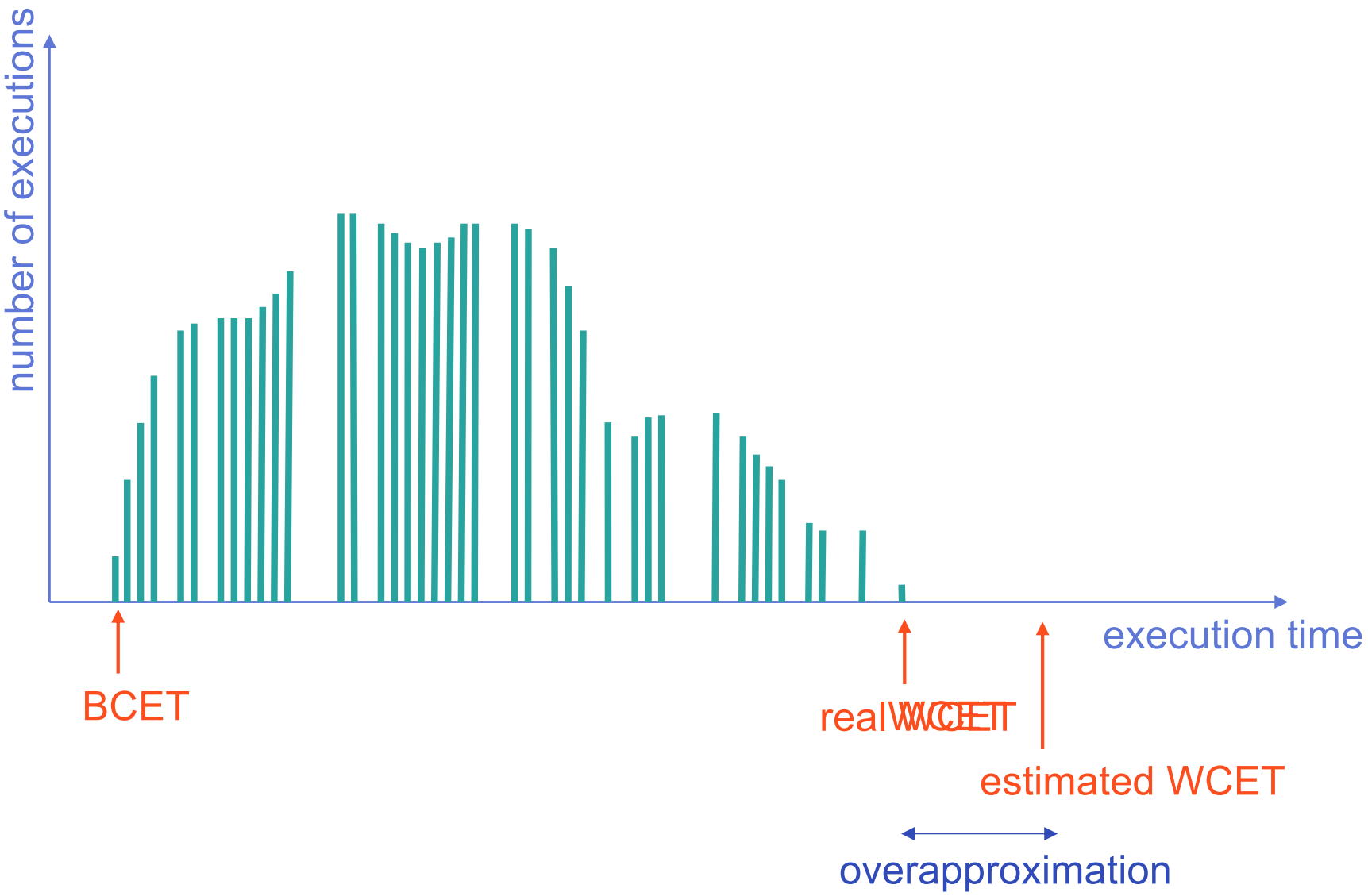
- N = 5 et n = 2 T = 71 10 possible execution paths
- N = 8 et n = 4 T = 108 70 possible paths
- N = 10 et n = 10 T = 128 a single path
- N = 10 et n = 0 T = 138 a single path

11000
10100
10010
10001
01100
01010
01001
00110
00101
00011

# Execution time: example 2



# Worst-Case Execution Time (WCET)



# Outline of the course

An overview of timing analysis

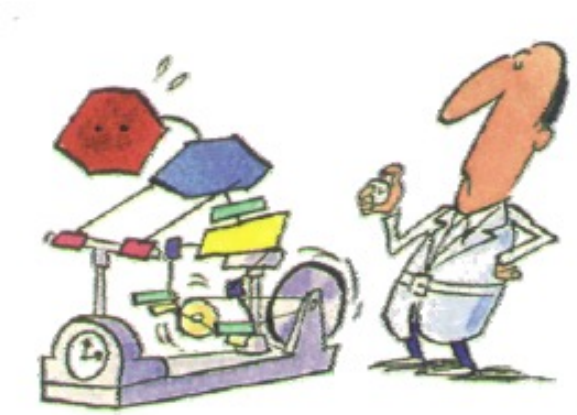
- techniques, tools

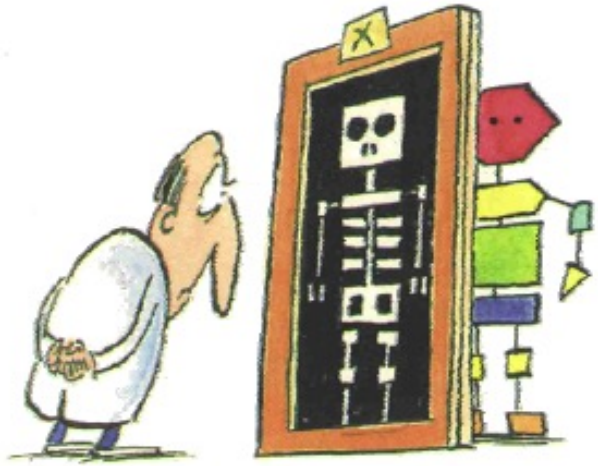
Static WCET analysis

- for a task running in isolation
- for a task that is subject to external interferences

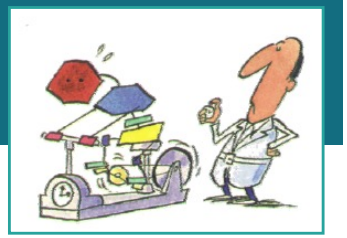
Approaches towards better timing predictability

# Approaches to WCET analysis





# Measurements



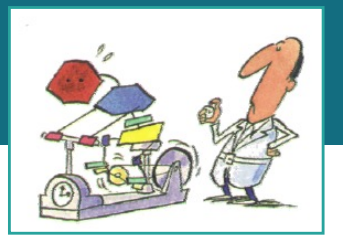
## On the target hardware

- requires precise hardware timing facilities

## Inputs?

- worst-case input vector?
  - generally unknown
- all possible input vectors?
  - too many combinations
- a set of input vectors that cover all possible execution paths?
  - not always computable, and may be too large
  - safe?
- a set of input vectors?
  - with adding a safety margin?
  - safe?

# Measurements



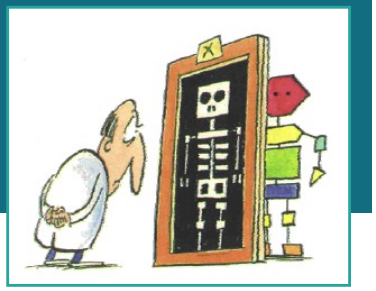

number of executions

execution time

real WCET

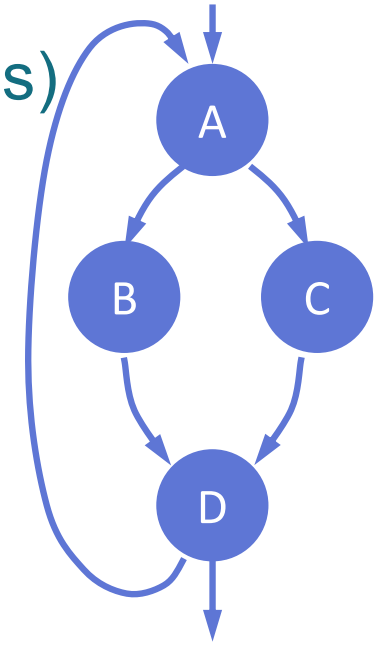
observed WCET

# Static analysis



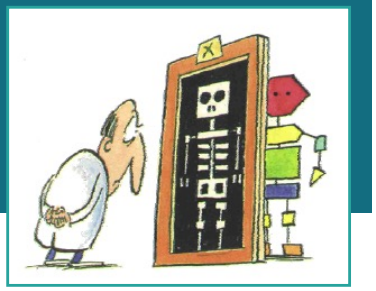
## Representation of the code as a Control Flow Graph (CFG)

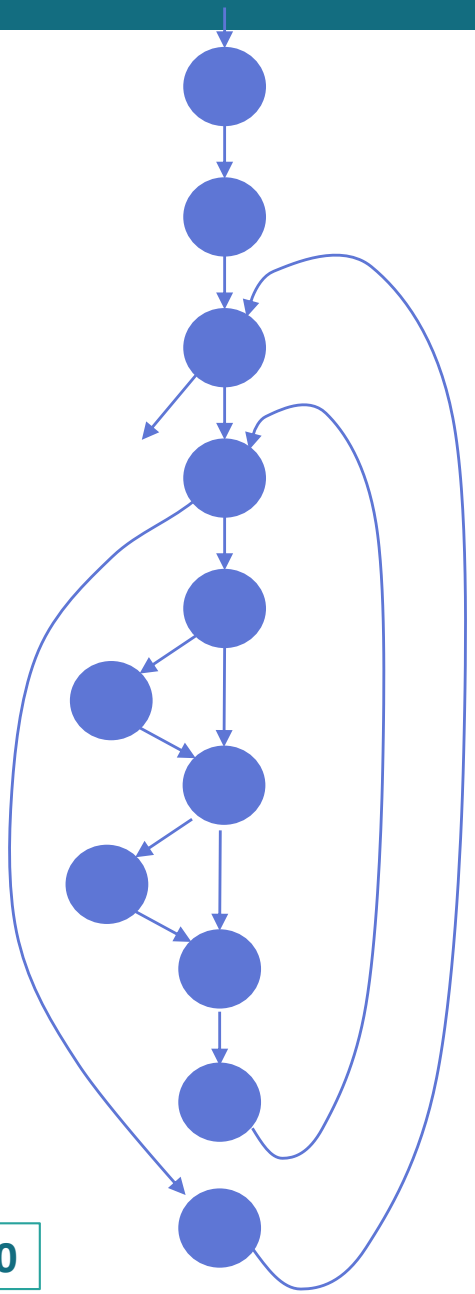
- The binary code is split into basic blocks (nodes)
  - a sequence of instructions
  - a single entry point
    - first instruction in the basic block
  - a single exit point
    - last instruction in the basic block




- Directed edges
  - express that two basic blocks can be executed in sequence

# Three steps





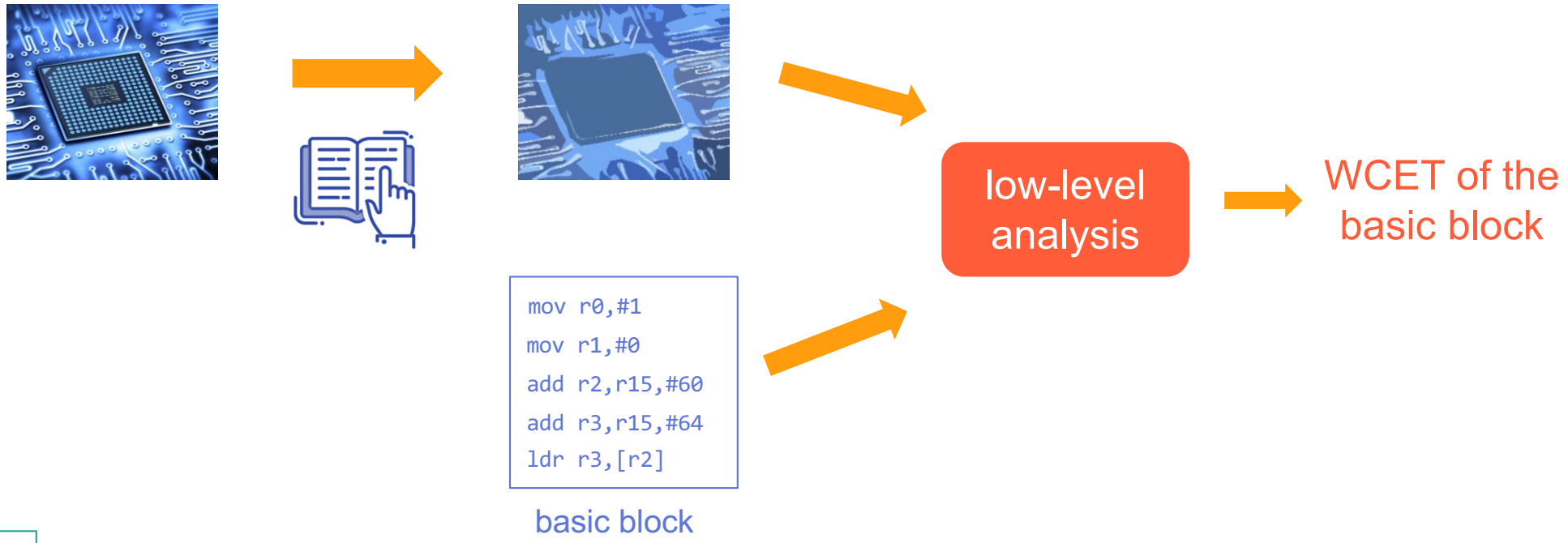
```
for (int i=0 ; i<64 ; i++)
    for (int j=0 ; j<i ; j++){
        if (a[i][j] > MAX)
            a[i][j] = MAX;
        if (a[i][j] < MIN)
            a[i][j] = MIN;
    }
```

1. Flow analysis
   - loop bounds, infeasible paths
2. Low-level/timing/microarchitectural analysis
   - WCET of basic blocks
3. WCET computation

# Timing/low-level analysis

## The main focus of this part of the course

- the objective is to determine the (worst-case) execution times/costs of basic blocks
- based on a model of the target hardware architecture



# Determining the longest path (IPET)

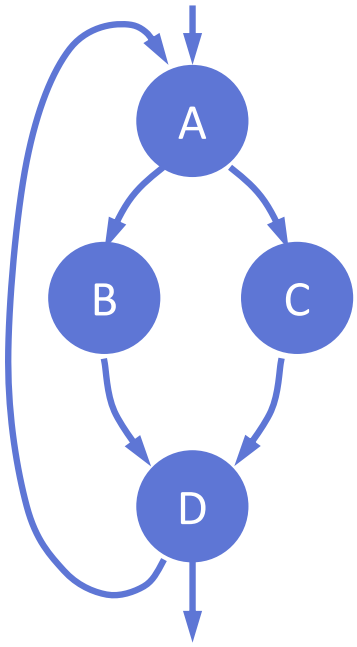
## Integer linear program

- global execution time
  - $T = \Sigma\ x_i\ .\ t_i$

execution time on a given path

execution count of block i on that path (unknown)

execution time of block i (known)




- maximize T to determine the WCET
  - compute the $x_i$ values that maximize T
    - requires specifying constraints on $x_i$

# Determining the longest path (IPET)

## Constraints on the $x_i$

- structural constraints:

  execution count of a basic block

  = sum of execution counts on input edges

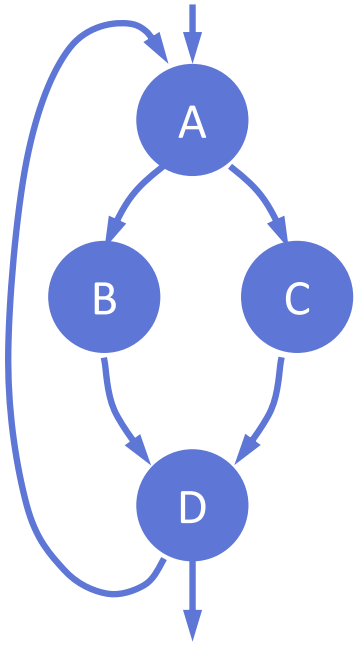
  = sum of execution counts on output edges

$x_A = 1 + x_{DA}$ $\quad$ $x_C = x_{AC}$

$x_A = x_{AB} + x_{AC}$ $\quad$ $x_C = x_{CD}$

$x_B = x_{AB}$ $\quad$ $x_D = x_{BD} + x_{CD}$

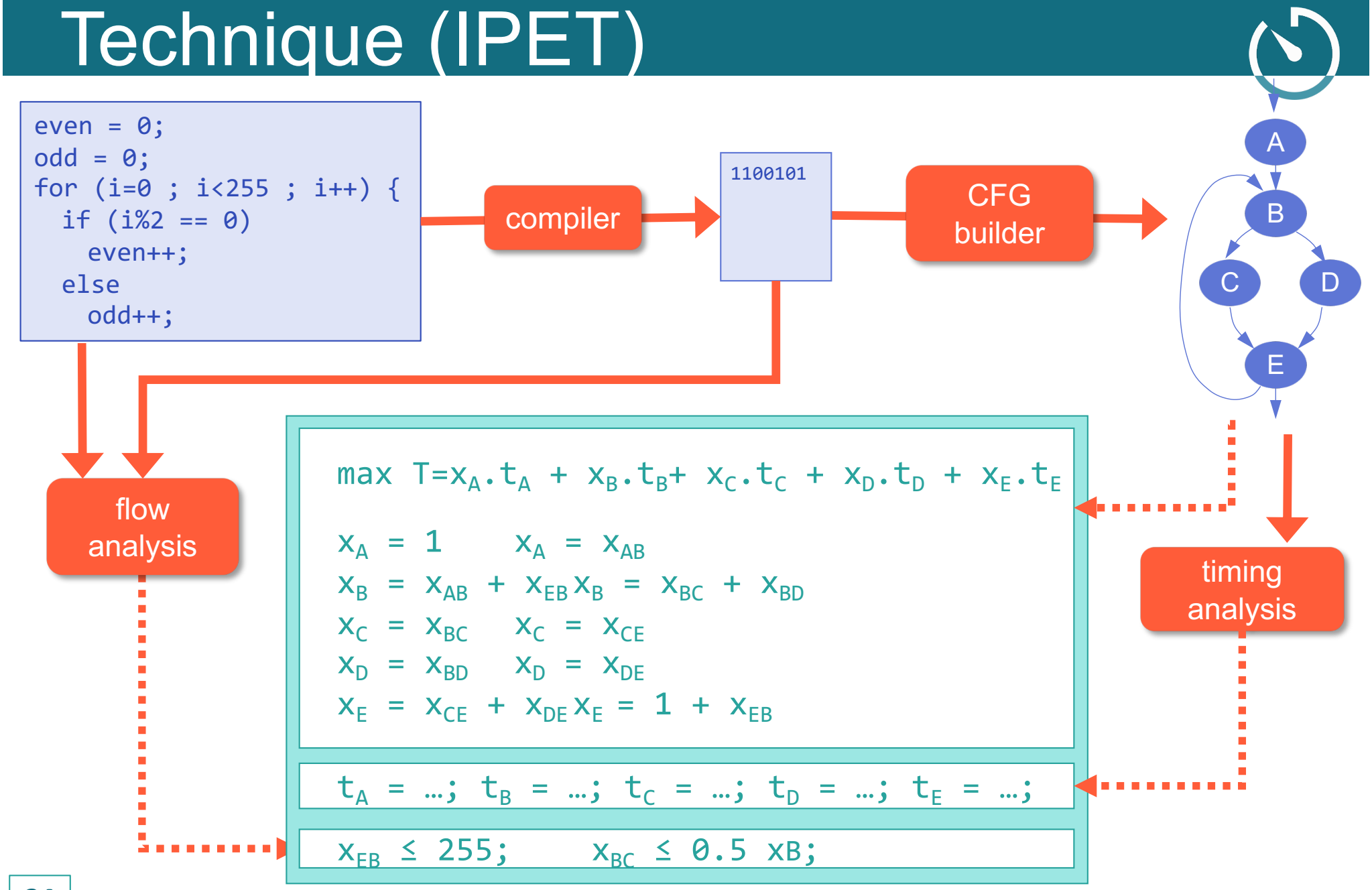
$x_B = x_{BD}$ $\quad$ $x_D = 1 + x_{DA}$




- flow constraints:

$$x_{DA} \leq N$$

# The Implicit Path Enumeration Technique (IPET)



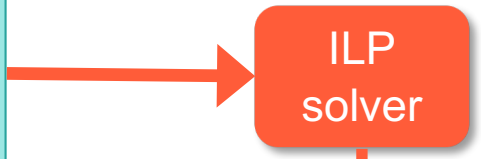
# The Implicit Path Enumeration Technique (IPET)

$$\max T = x_A.t_A + x_B.t_B + x_C.t_C + x_D.t_D + x_E.t_E$$

$$x_A = 1 \qquad x_A = x_{AB}$$
$$x_B = x_{AB} + x_{EB} \qquad x_B = x_{BC} + x_{BD}$$
$$x_C = x_{BC} \qquad x_C = x_{CE}$$
$$x_D = x_{BD} \qquad x_D = x_{DE}$$
$$x_E = x_{CE} + x_{DE} \qquad x_E = 1 + x_{EB}$$

$$t_A = \ldots;\ t_B = \ldots;\ t_C = \ldots;\ t_D = \ldots;\ t_E = \ldots;$$

$$x_{EB} \leq 255; \qquad x_{BC} \leq 0.5\ x_B;$$




Tmax = … (WCET)
*with:*
$x_A$ = …
$x_B$ = …
$x_C$ = …
$x_D$ = …
$x_E$ = …

# Hybrid WCET analysis

## Combines measurements and static analysis

- instrumentation code (ipoints) →timestamped traces

```
(address1, timestamp1)
(address2, timestamp2)
…
```

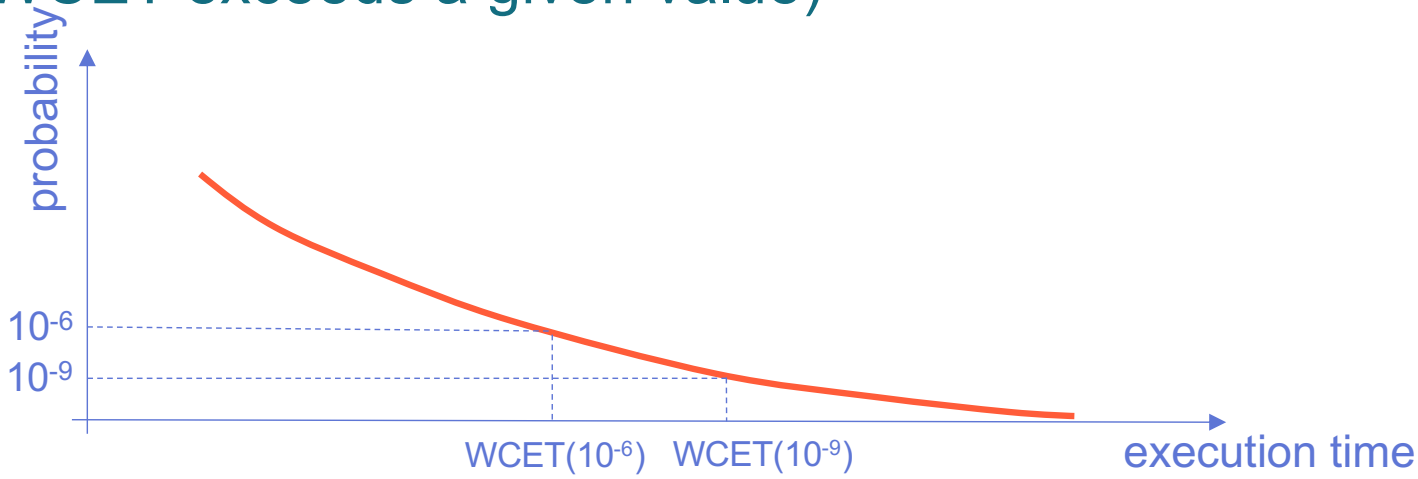
  - used to estimate the local WCETs of basic blocks

- combines the advantages and drawbacks of measurements and static analysis
- instrumentation compatible with certification?

# Probabilistic approaches

## Basic idea

- produce several WCET estimates with associated probabilities
- compute the probability of a timing failure (the WCET exceeds a given value)



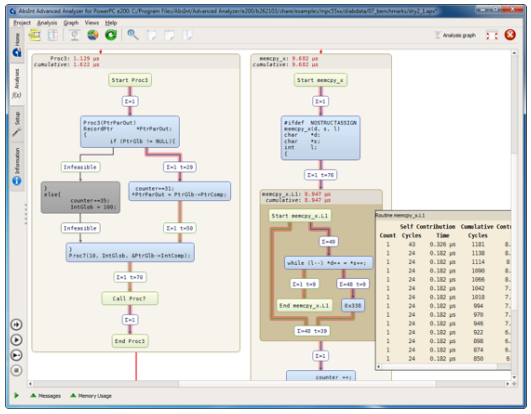

## Techniques

- randomized hardware
- static or measurement-based

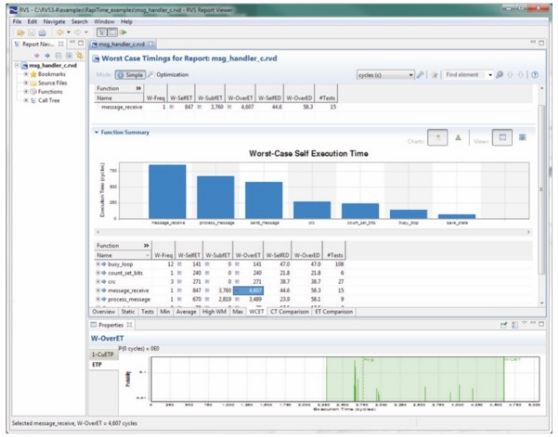
# WCET Analysis Tools

## Commercial tools:

- aiT AbsInt

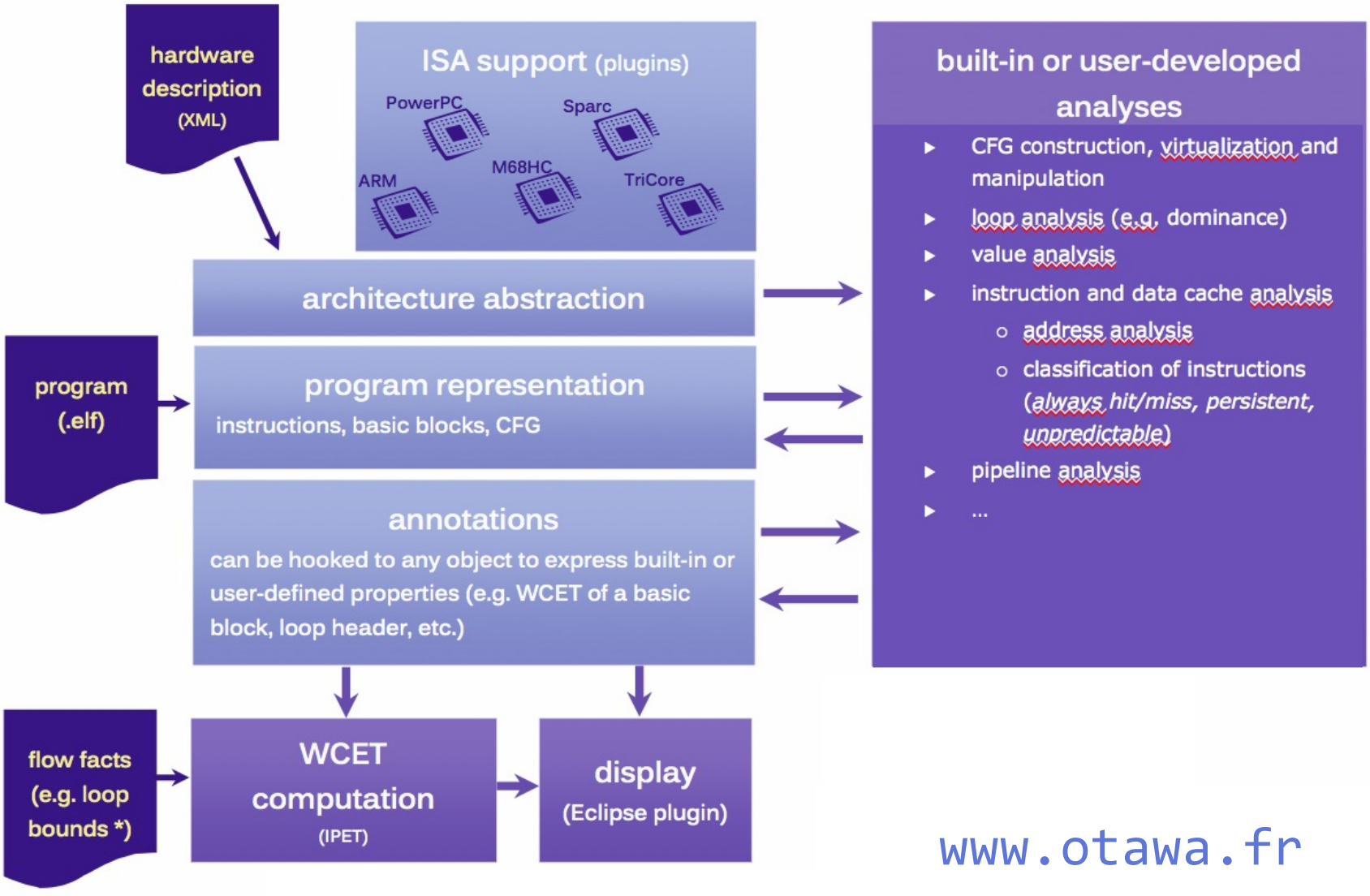


- RapiTime RAPITA Systems Ltd

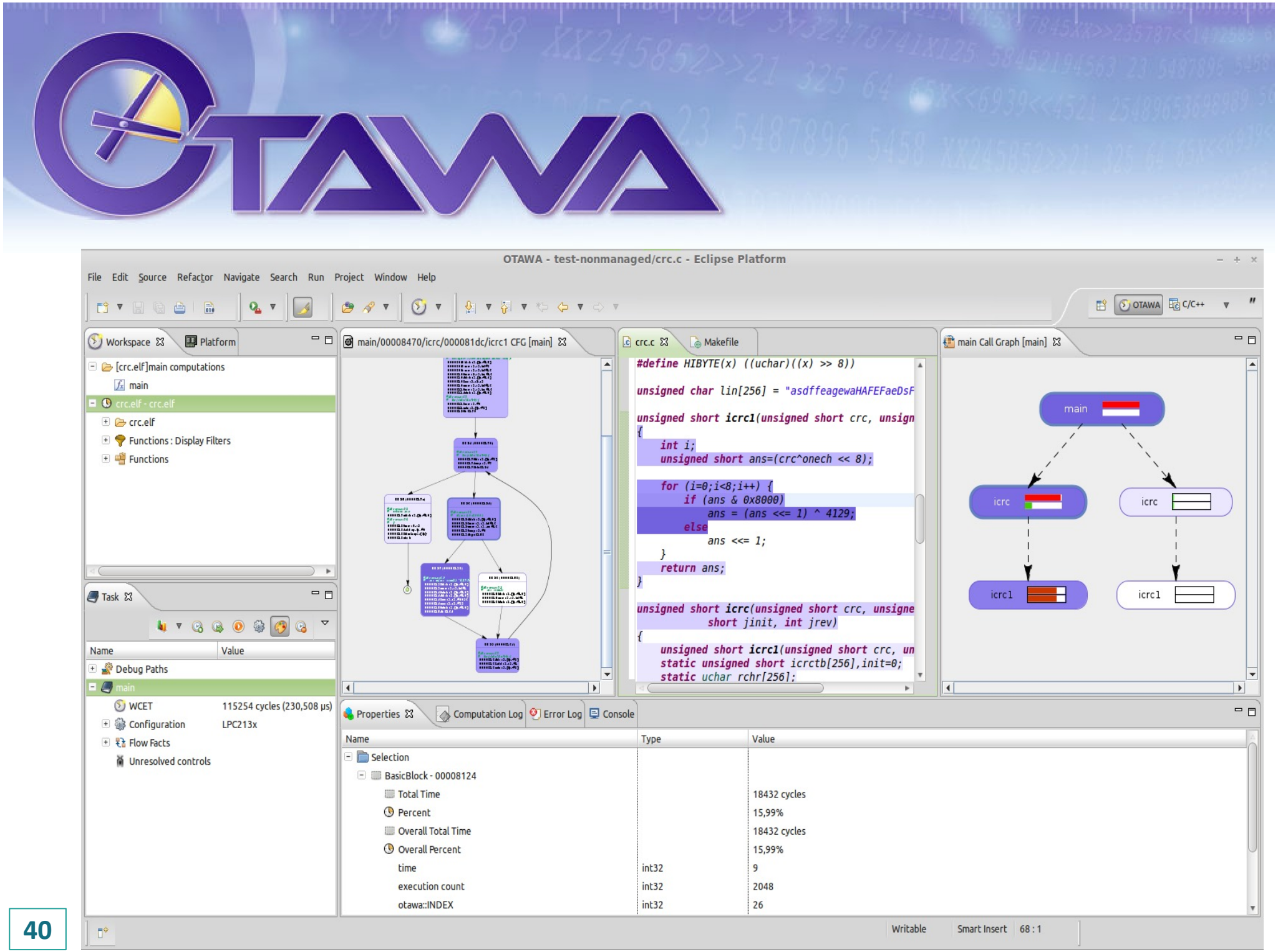



## Academic tools:

- Chronos (Singapour)
- Heptane (Rennes)
- KTA (Stockholm)
- OTAWA (Toulouse)
- SWEET (Mälardalen)


hardware description (XML)
ISA support (plugins)
PowerPC
Sparc
ARM
M68HC
TriCore
architecture abstraction
program (.elf)
program representation
instructions, basic blocks, CFG
annotations
can be hooked to any object to express built-in or user-defined properties (e.g. WCET of a basic block, loop header, etc.)
flow facts (e.g. loop bounds *)
WCET computation
(IPET)
display
(Eclipse plugin)
built-in or user-developed analyses
CFG construction, virtualization and manipulation
loop analysis (e.g. dominance)
value analysis
instruction and data cache analysis
address analysis
classification of instructions (always hit/miss, persistent, unpredictable)
pipeline analysis
...

www.otawa.fr


OTAWA - test-nonmanaged/crc.c - Eclipse Platform
File Edit Source Refactor Navigate Search Run Project Window Help
Workspace
Platform
[crc.elf]main computations
main
crc.elf - crc.elf
crc.elf
Functions : Display Filters
Functions
main/00008470/icrc/000081dc/icrc1 CFG [main]
crc.c
Makefile
#define HIBYTE(x) ((uchar)((x) >> 8))
unsigned char lin[256] = "asdffeagewaHAFEFaeDsF
unsigned short icrc1(unsigned short crc, unsign
{
int i;
unsigned short ans=(crc^onech << 8);
for (i=0;i<8;i++) {
if (ans & 0x8000)
ans = (ans <<= 1) ^ 4129;
else
ans <<= 1;
}
return ans;
}
unsigned short icrc(unsigned short crc, unsigne
short jinit, int jrev)
{
unsigned short icrc1(unsigned short crc, un
static unsigned short icrctb[256],init=0;
static uchar rchr[256];
main Call Graph [main]
main
icrc
icrc
icrc1
icrc1
Task
Name
Value
Debug Paths
main
WCET
115254 cycles (230,508 µs)
Configuration
LPC213x
Flow Facts
Unresolved controls
Properties
Computation Log
Error Log
Console
Name
Type
Value
Selection
BasicBlock - 00008124
Total Time
18432 cycles
Percent
15,99%
Overall Total Time
18432 cycles
Overall Percent
15,99%
time
int32
9
execution count
int32
2048
otawa::INDEX
int32
26
Writable
Smart Insert
68 : 1