Real-Time aspects in CPS

Emmanuel GROLLEAU

Professor of CS, ISAE-ENSMA, Poitiers, France

grolleau@ensma.fr

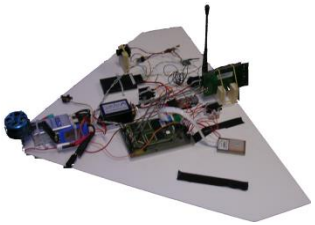
Course outline

1. Introduction
2. Functional specification & requirements
3. From sequential to parallel programming
4. Mapping software on hardware
5. Timing&Schedulability analysis

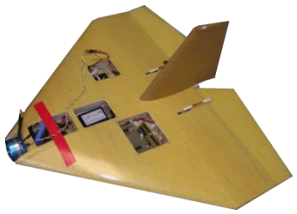
Example-based introduction

1. INTRODUCTION

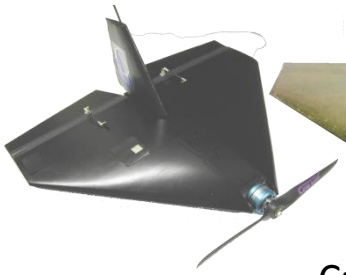
Mini-UAV example



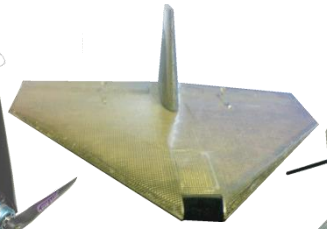
Components on a board



Kevlar version



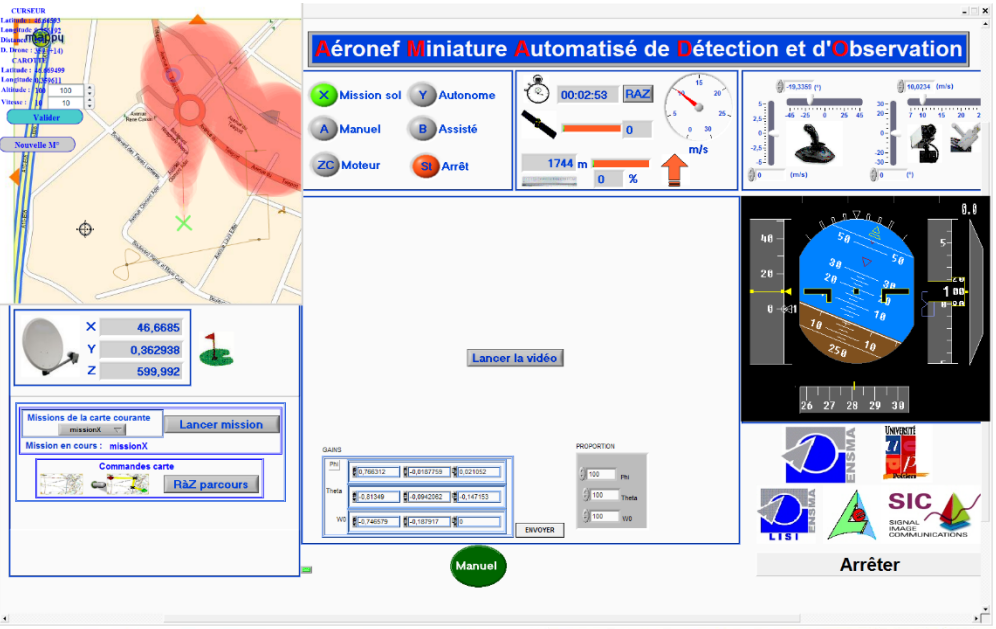
Carbon version



Carbon-kevlar version for wind tunnel

3D test version

Test trainer

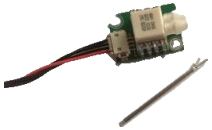


Ground station

Knowing and changing the state/1

- Sensors: knowing the system's state
  - Where (position, ground and/or air speed, altitude) ?
    - ✓ GPS receiver (, Pitot tube, barometer, ultrasonic sensor)

M8 GPS receiver


PX4 airspeed sensor


HP206C barometer


MB12XX sonar

  - How (attitude: roll, pitch (, yaw))
    - ✓ IMU (accelerometers, gyroscopes (, magnetometers))


MTx IMU

  - What (operator: setpoint, information display and storage)
    - ✓ Wireless communication (RC, RF Modem, Wi-Fi, GSM …), SD port


Xtend RF modem

Knowing and changing the state /2

- Actuators: changing the system's state
  - Servomotors of the elevons
  - ESC controlling the propeller

VHVBEC

- Mission specific hardware
  - Video, air quality, payload, etc.
- Computing and communication resources
  - Microcontroller(s + network(s)), FPGA,…


MPC555


ARM9 LPC

Main questions addressed

- Design a software architecture able to execute the set of missions
  - Based on a choice of hardware/influencing the choice of hardware
  - Hardware/software co-design
  - How to check if the hardware resources fullfill the need of the sofware?
- Side questions in this presentation
  - How much confidence (safety)?
  - How exposed/attack resilient should it be (security)?

2. FUNCTIONAL SPECIFICATION & REQUIREMENTS

Requirements

- Functional requirements
  - Regulatory requirements
    - ✓ The UAV autopilot should conform to the regulations
      - (in France, depending on category and scenario)
  - Authorization levels/authentification
    - ✓ The remote controller shall be authentified
    - ✓ A registered remote controller can set a flight mode (assisted/automatic)
  - Logging
    - ✓ The localization is stored on a drive every second, with a precision of …
  - Functions
    - ✓ In assisted mode, the controller provides a setpoint for a target roll angle, air speed, vertical speed that the UAV tends to conform

Non functional requirements

- Performance
  - The maximum delay between the attitude acquisition and the corresponding command should be less than 80 ms
  - The automatic flight control frequency should be at least 50 Hz
- Safety
  - Probability of total loss of control due to the system should be less than 10^{-4} per hour of flight

From requirements to system

- Model-Based Engineering
  - Combine several visual points of view
  - Focus on different aspects
    - ✓Software, hardware, structure, functional, behavioral, …
  - Check consistency
  - Facilitates bridges
    - ✓Autocoding
    - ✓Analysis tools

Functional decomposition

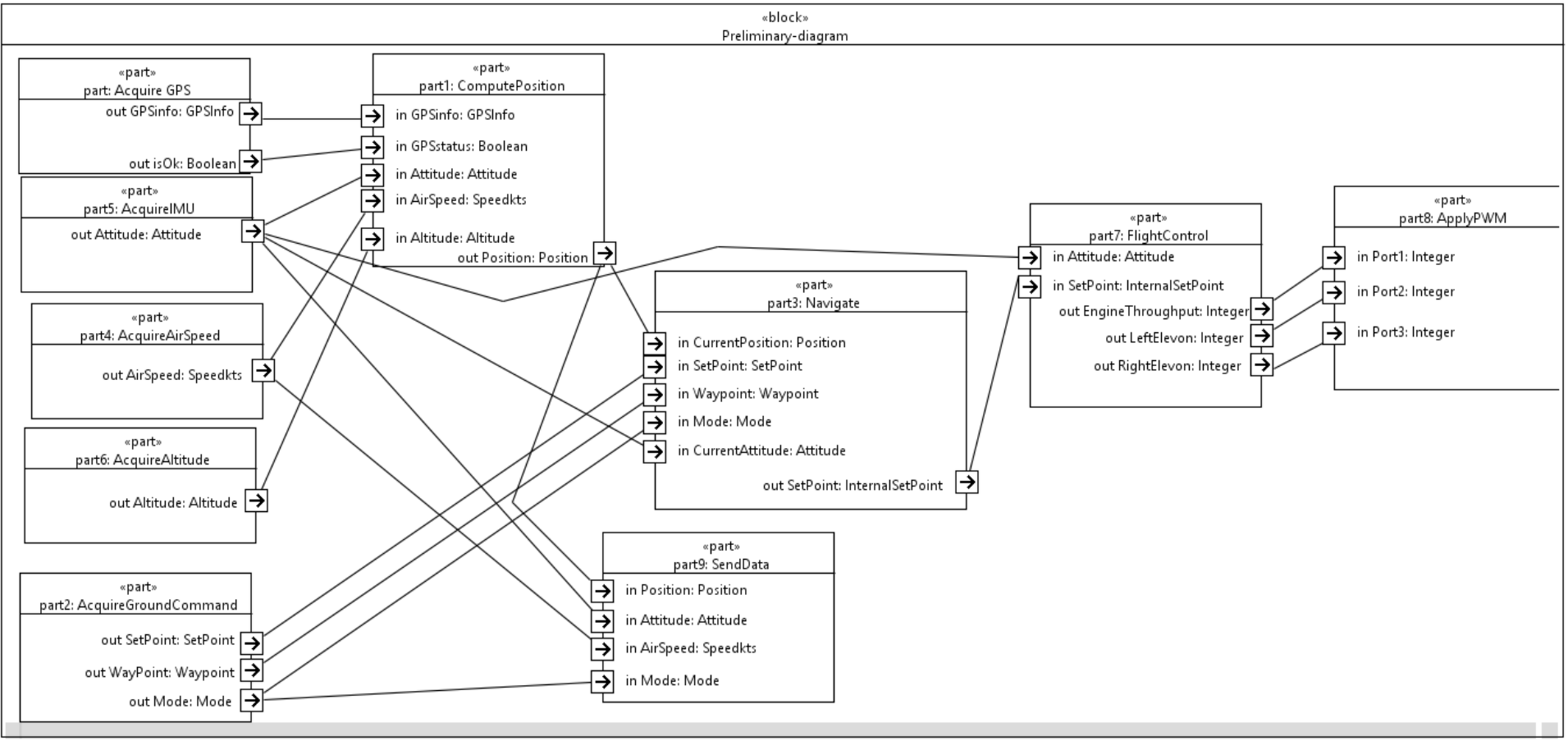
- Often expressed using dataflow-based languages
  - Hierarchical & Graphical representation
  - Easy to use and understand for every engineer
- Several languages
  - E.g. SA-RT (1986), SysML/UML (20xx)

Example: SysML

- OMG Standard
  - V1.0 2007
  - V1.3 2012
- Defined as an UML 2.0 profile
- 9 diagrams types
  - 7 from UML, 2 specific diagrams

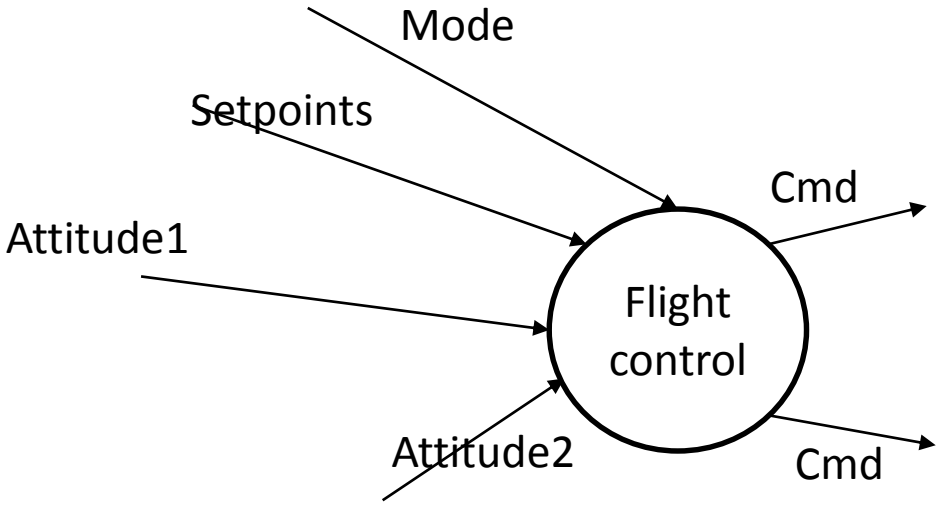
UAV example

High level Internal Block Diagram



To dataflow or not to dataflow?

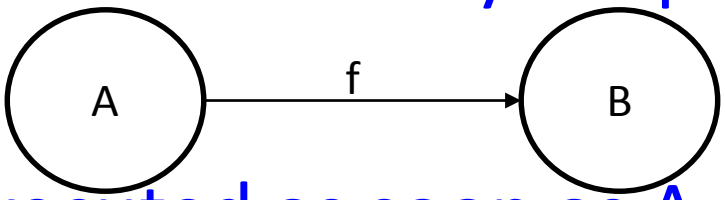
- Dataflow semantics
  - Ex: SA



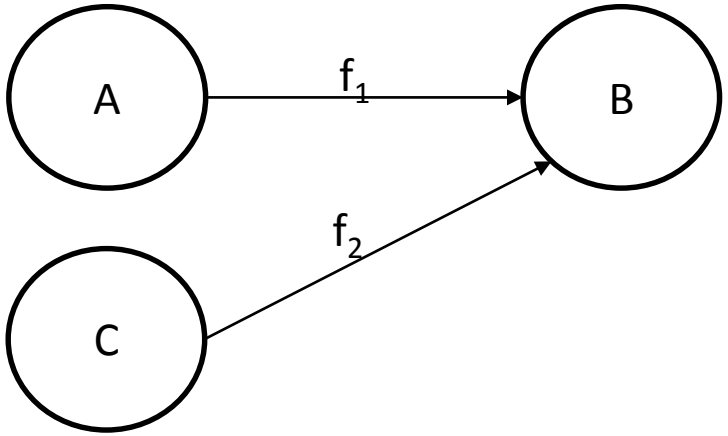
  - A dataflow process is executed as soon as every input data is available, and produces its outputs
  - Data is consumed when read

Typical specification ambiguity

- Dataflow = loosely coupled synchronisation?

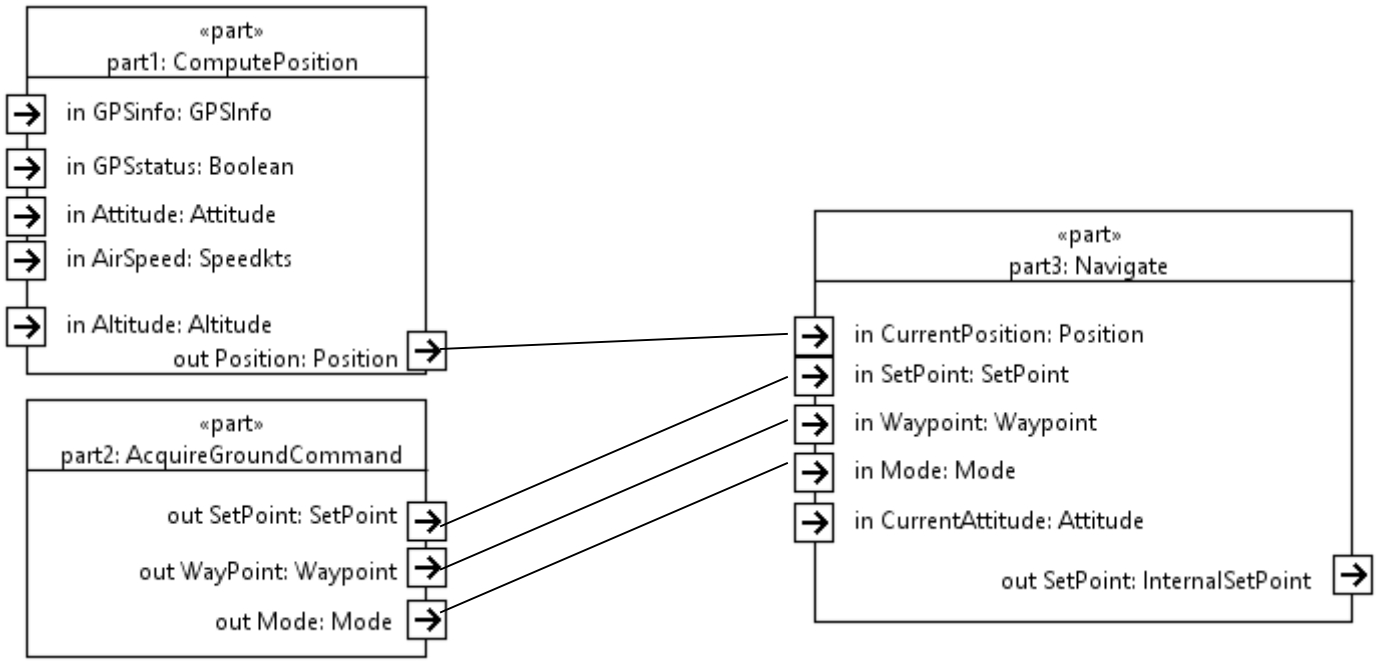


- B executed as soon as A and C produced f_1 and f_2
  - What if A faster than C? f_1 bufferized in a finite or infinite buffer?

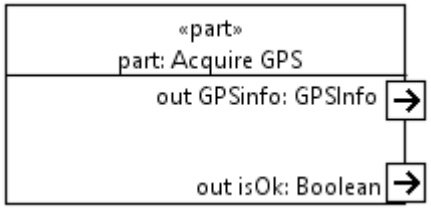


Rythms are unknown

- When will the ground commands be received?
  - Every 10ms? 100ms? Depends on the resources and bandwidth of the wireless communication
  - Never if up-datalink is loss?
- Navigate still has to be executed
- Or/And semantics + asynchronous/loosely synchronous semantics

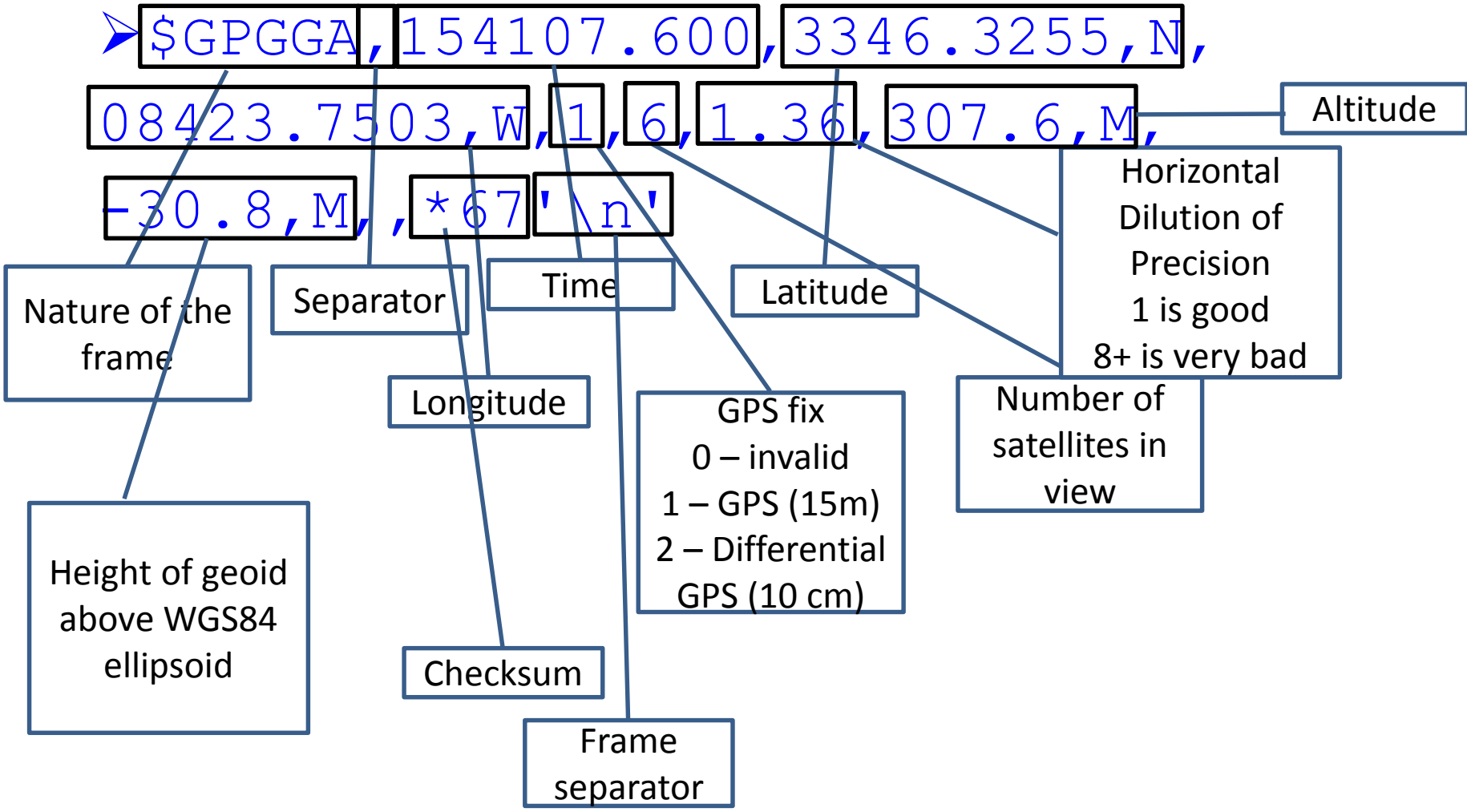


Detailing functions



- A GPS receiver typically sends a NMEA or proprietary frame at a pre-defined rate
  - `$GPGGA,154107.600,3346.3255,N,08423.7503,W,1,6,1.36,307.6,M,-30.8,M,,*67'\n'`
  - Ranging from 1 to 10 Hz typically
- The bus used is often serial
  - E.g. RS-232 serial, 57600 bauds

Entr'acte: NMEA Protocole



Entr'acte: NMEA Protocole

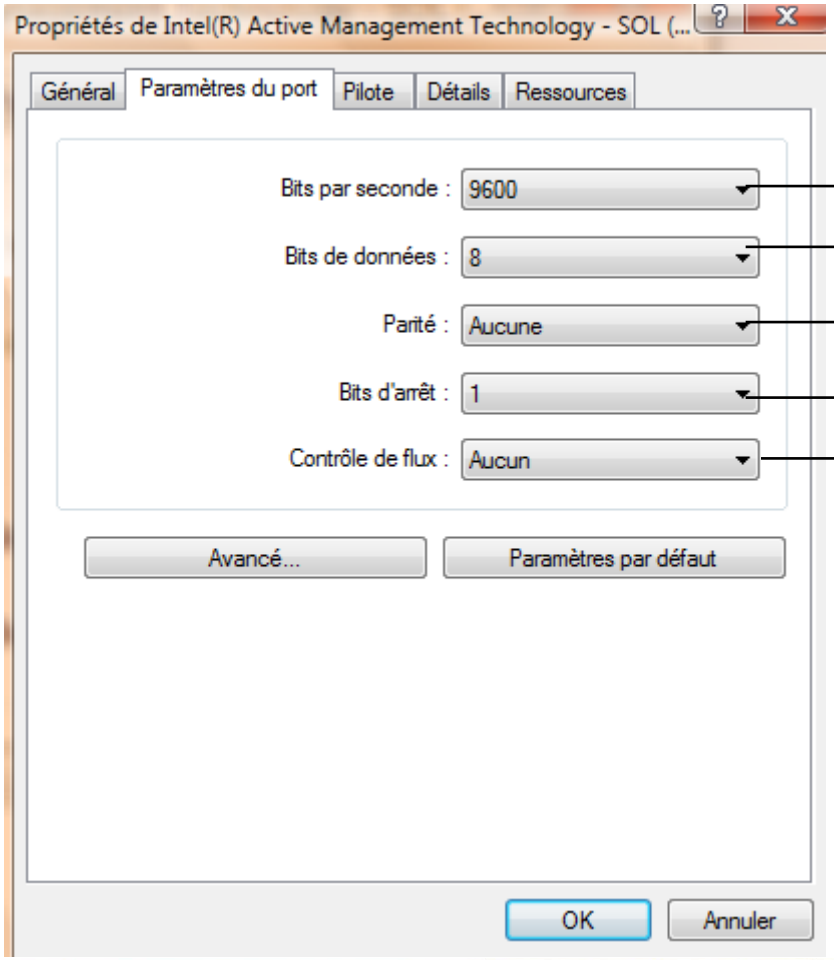
- Some GPxxx formats can also indicate
  - Ground speed
  - Vertical speed
- Or even follow a waypoint
  - Give the variation
  - Etc.

When is my position ‘correct’?

- Datawise
  - When I receive a frame, modulo the delays
  - And I recognize the header '$GPxxx'
  - And the checksum is correct
  - And the GPS fix is 1 or 2
  - And the HDoP is not too bad
- Behavior
  - And the position is consistent

How is a RS-232C link working?

DB9 connector

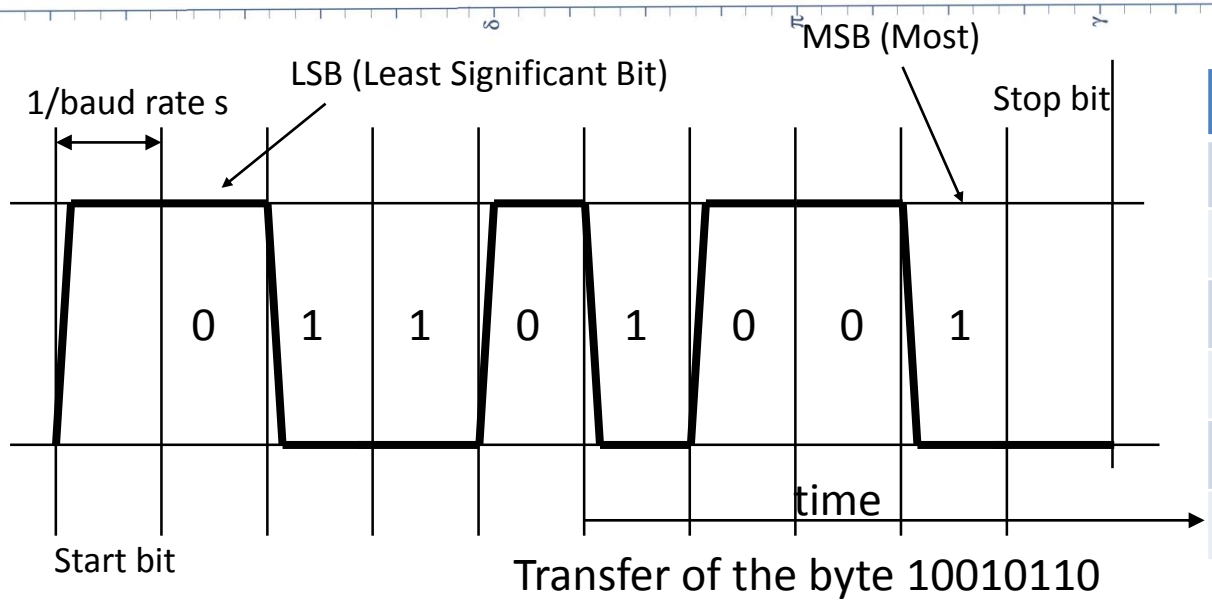


- Bauds rate: 75, …, **9600**, …, **57600**, 115200, 128000
- Data bits: 4, 5, 6, 7, **8**
- Parity check: Odd, Even, **None**, Mark, Space
- Stop bits: **1**, 1.5, 2
- Flow Control: **None**, Xon/Xoff, Hardware

RS-232C

- Without flow control (majority) asynchronous communication
- 1 wire used for each direction: Tx (Transmit) et Rx (Receive)
  - RS-232C is *full-duplex*
- Digital bus where 0 and 1 are sent (e.g. -6V for 1's..+6V for 0's)
- Each bit lasts 1/baudrate seconds on the bus (wire)
  - ~17.36µs at 56k (57600) bauds
- If no start/stop bit, potential desynchronization
  - Send 00000000 one thousand times => the clock on each system may drift
  - 8000 zeros are sent but 7999 or 8001 zeros are received
- Start bit 0 (high), stop bit 1 (low) insure clock resynchronization

Example



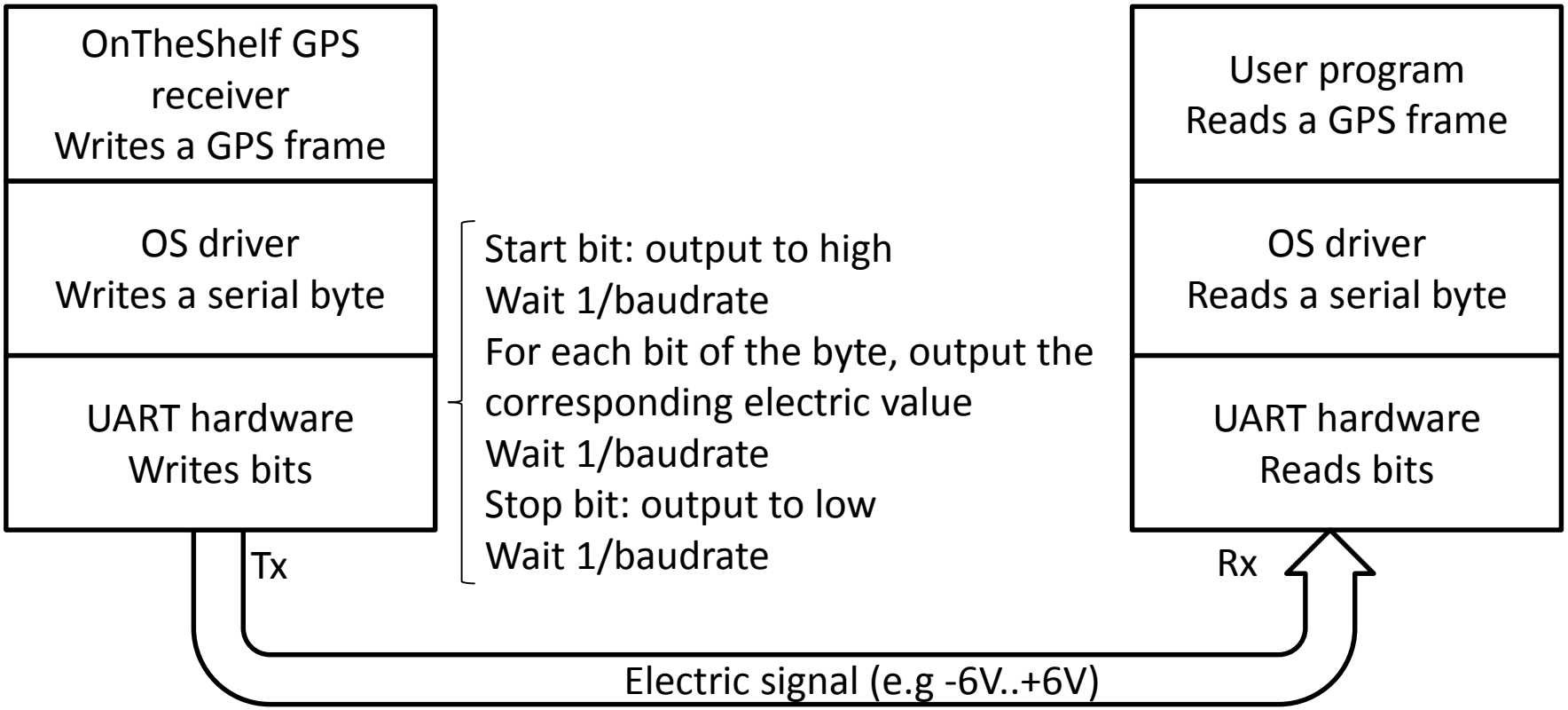
Transfer of the byte 10010110

ASCII table

char	code	binary
$	36	00100100
G	71	01000111
P	80	11010000
,	44	00101100
1	49	00110001
\n	13	00001101

- Throughput for 57600bps on 8 data bits and 1 stop bit: 5760 byte/s (2 bits over 10 used for synchronization)
- Duration of the NMEA frame of 73 characters (each one encoded by a byte):
  - 73 bytes/5760 bytes/s ~ 9.6 ms
- If GPS sends frames at 10 Hz, the bandwidth used is
  - 9.6.10-3 s/frame*10 frames/s=0.096=9.6% of the time

Communication protocole



- What if there is a transmission error?
  - Electromagnetic interference, synchronization problem, heavy ion, hardware bug…

Error control

- Each layer can have some error checking
  - UART level
    - ✓ the sender adds a bit such that parity is odd; the receiver checks the parity of the received data
  - Protocole level
    - ✓ For each byte $b_{i,i=1..n}$ of the frame, from '$' to '*', the sender first computes $b_1 \oplus b_2 \oplus \ldots \oplus b_n$=checksum
    - ✓ Checksum is happened in hexadecimal after the '*'
      - *65 means that the result of the checksum is 01100101
    - ✓ The receiver computes the checksum
      - If the received and computed checksum are ≠ then there is an error

What with the xor?

A	B	A⊕B
0	0	0
0	1	1
1	0	1
1	1	**0**

- Xor means
  - Exclusive or ('or else')
  - Different: (A⊕B) ⇔ (A≠B)
  - Binary sum without carry
  - **Odd parity**: (A⊕B) is true iff parity(AB)=odd
    - ✓Thus parity(AB(A⊕B))=even

Even

	0	1	0	0	0	1	1	1	G
⊕	1	1	0	1	0	0	0	0	P
⊕	0	1	0	0	0	1	1	1	G
	1	1	0	1	0	0	0	0	

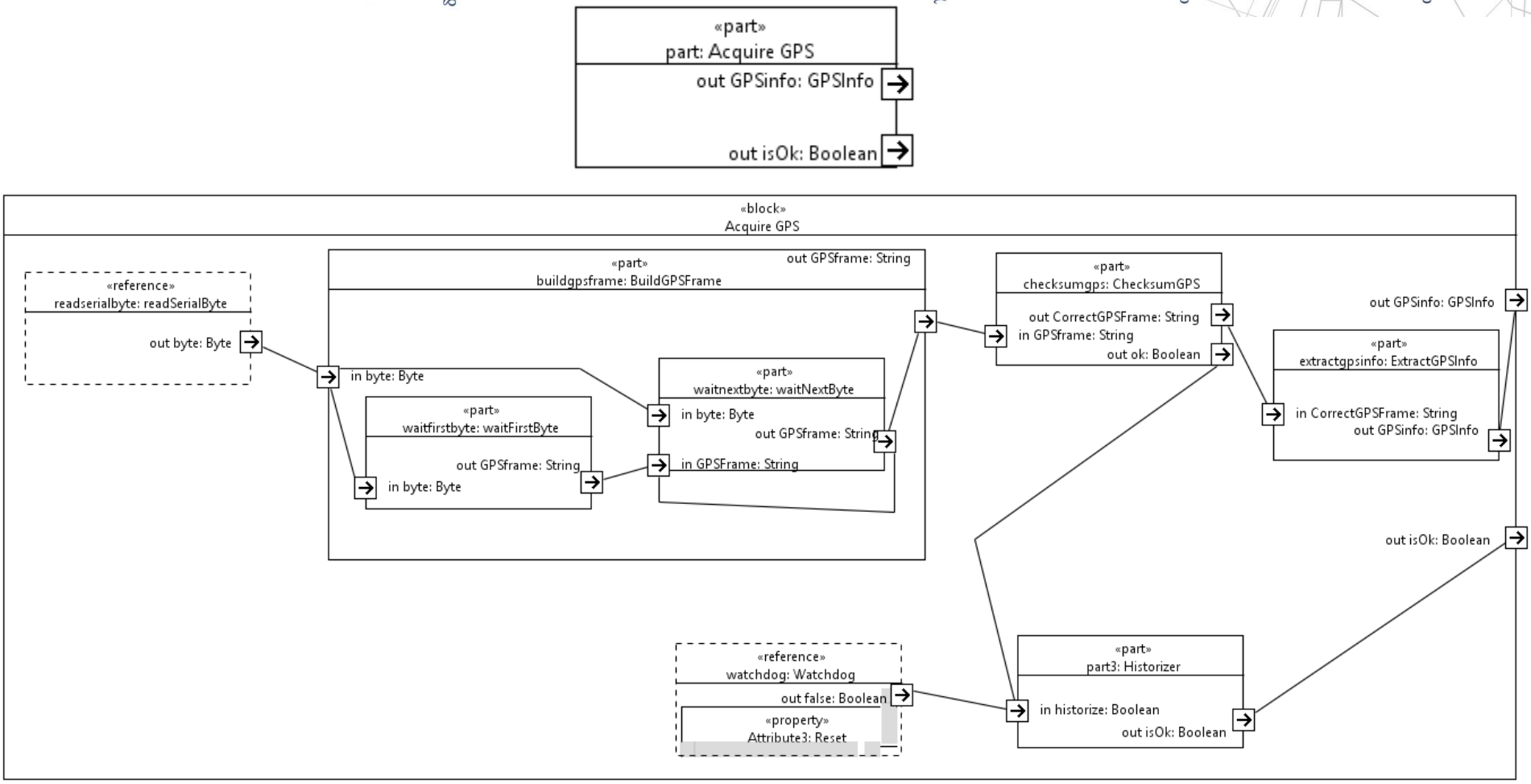
Column-wise parity

Checksum

- A checksum is a very efficient way of doing a column-wise parity check
- ~1 CPU cycle per byte

- Now let's see how to express how to read a GPS frame

AcquireGPS as an IBD



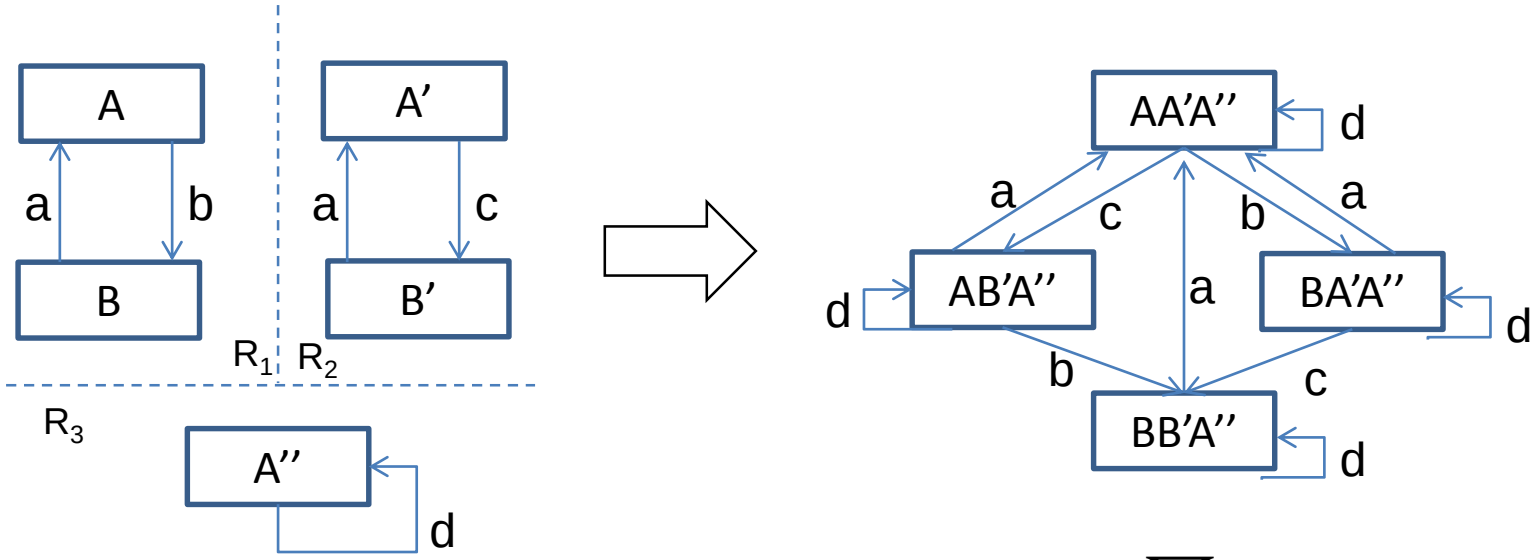
- Most visual languages are hierarchical
- Do not bother about parts and references

Formalizing the behavior

- Several diagrams express the behaviors
  - E.g. in SysML: behavioral diagram, State Machine diagram
- UML State Machine
  - Harel statechart
    - ✓Hierarchical nested states and orthogonal regions

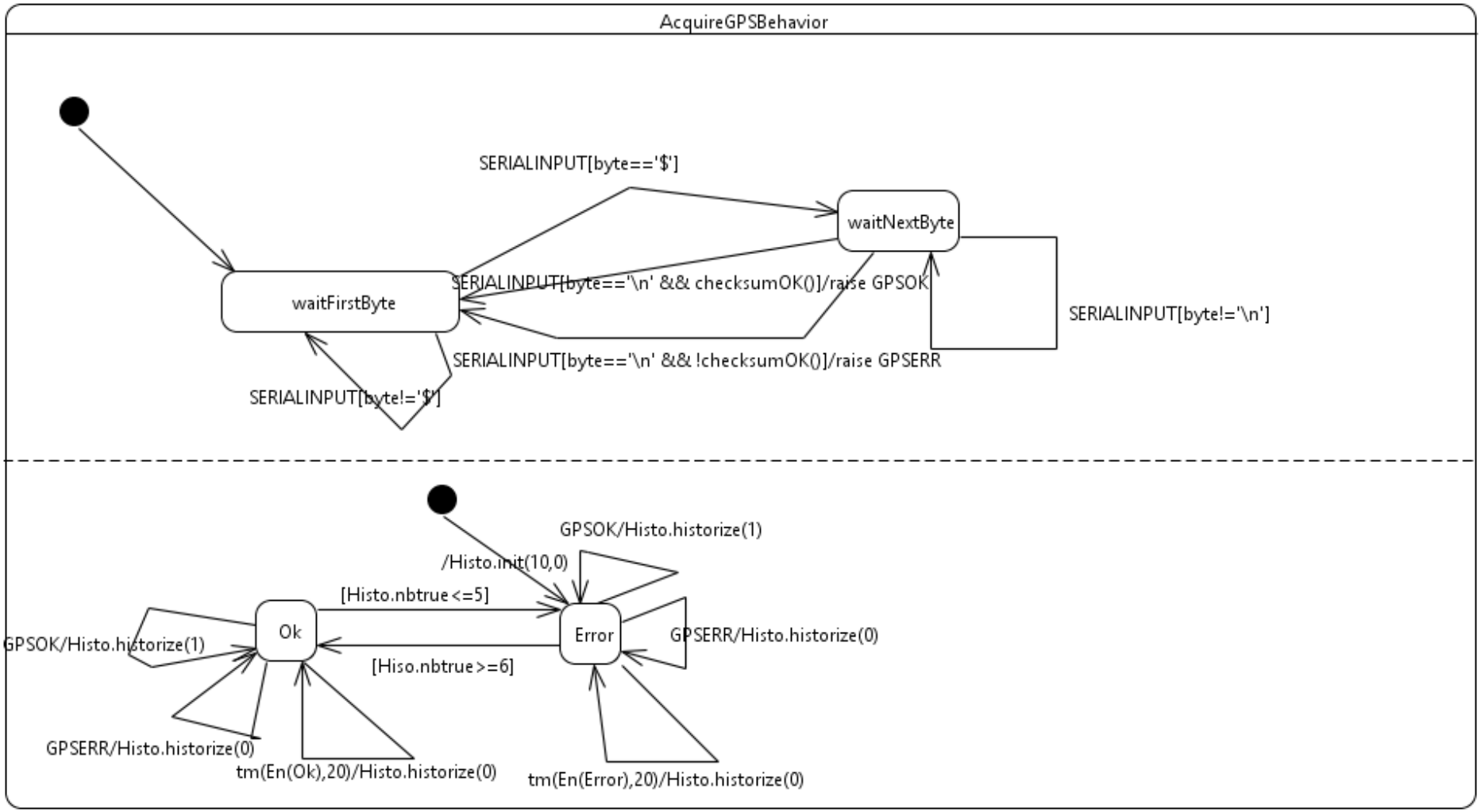
Orthogonal regions

- Cartesian synchronized product



$$number\ of\ states \leq \prod_{region\ R} number\ of\ states\ in\ R$$

GPS (simplified) example



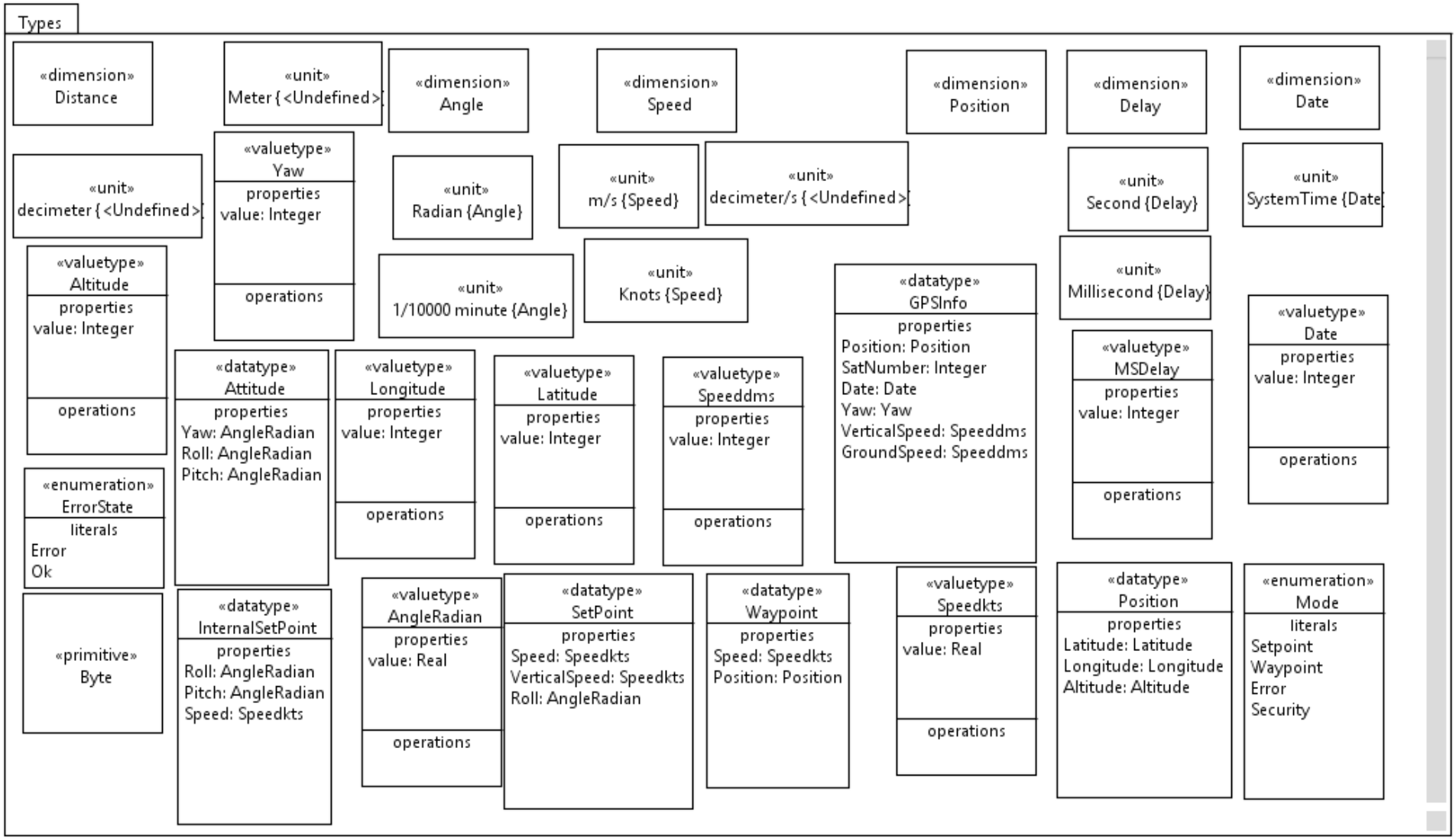
Typing and units

- Speed : Integer? Real?
  - Note that Real => Float
- Unit?
  - Km/h, m/s, knot/s, mph…

One of many unit error examples

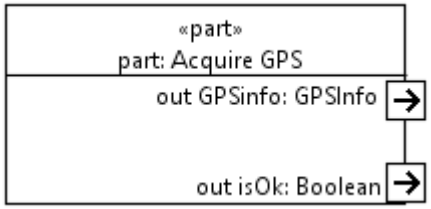
- Loss of the NASA Mars Climate Orbiter
  - September 1999
  - Orbital insersion maneuver software module
    - ✓Sending in pound/s
    - ✓Expected in Newton/s

Typing: a package in a BDD

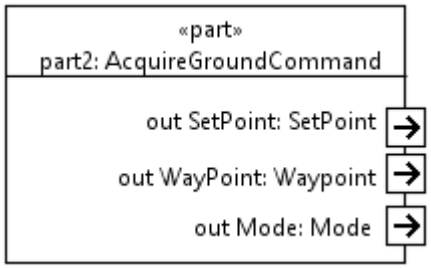


3. FROM SEQUENTIAL TO PARALLEL PROGRAMMING

Concurrent behaviors



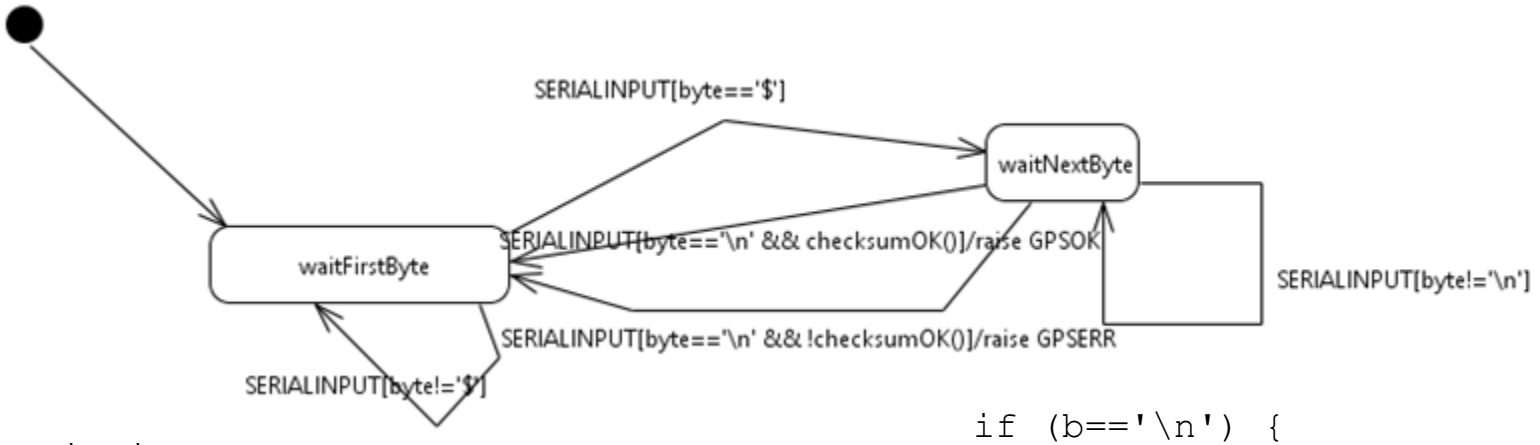
GPS frame “lasts” 9.6 ms, every byte is separated by 17.36 µs.



Frame of 30 bytes at 9600 bauds, lasts ~31 ms, every byte is separated by ~1 ms

- Suppose functions
  - readGPSFrame => lasts >= 9.6 ms
  - readGroundCommad => lasts >=31 ms
- We do not know when the frames will arrive

Triggering a function



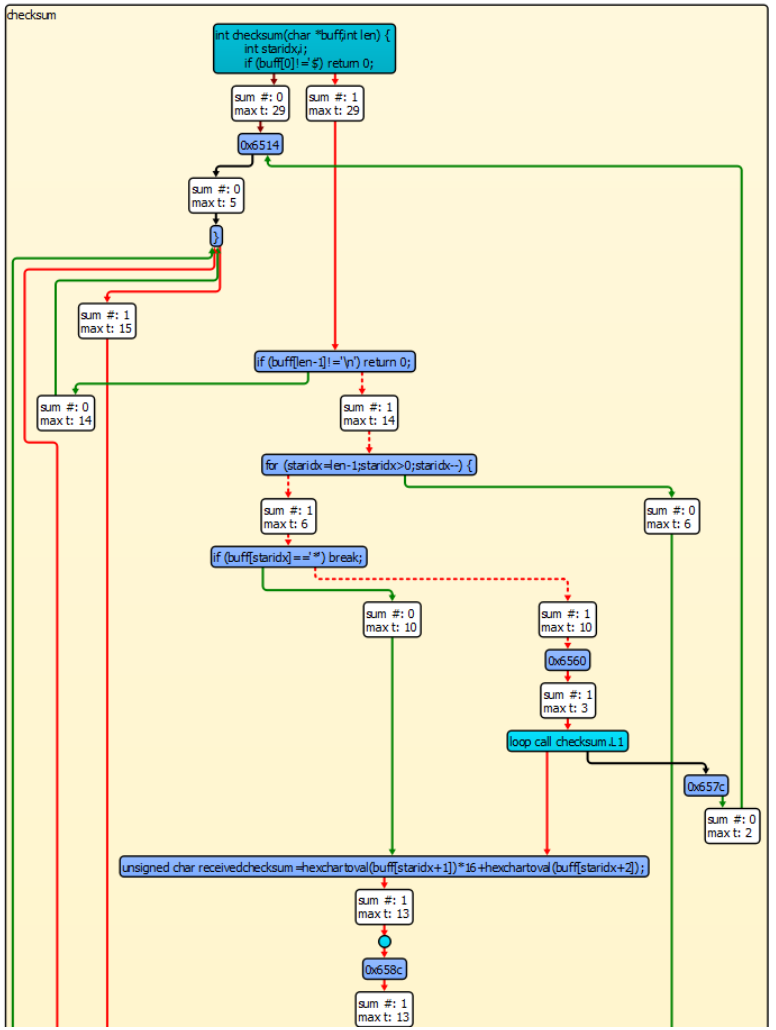

```
State= waitFirstByte
i=0;
∀ Incoming serial byte b
switch (State) {
    waitFirstByte:
    if (b=='$') {
        State=waitNextByte;
        buffer[i]=b;
        i++;
    }
    break;

    waitNextByte:
    buffer[i]=b;
    i++;
    if (b=='\n') {
        ok=checksum(buffer,i);
        if (ok) {
            resetWatchdog();
        }
        State=waitFirstByte;
        i=0;
    }
}
```

- Note: buffer overflow vulnerability

Timing analysis of a function

```
unsigned char hexchartoval(char c) {
    if (c>='0' && c<='9') return c-'0';
    if (c>='A' && c<='F') return c-'A'+10;
    if (c>='a' && c<='f') return c-'z'+10;
    return 0;
}
int checksum(char *buff,int len) {
    int staridx,i;
    if (buff[0]!='$') return 0;
    if (buff[len-1]!='\n') return 0;
    for (staridx=len-1;staridx>0;staridx--) {
        if (buff[staridx]=='*') break;
    }
    if (staridx==0) return 0;
    unsigned char receivedchecksum=hexchartoval(buff[staridx+1])*1(
    unsigned char mychecksum=0;
    for (i=1;i<staridx;i++) {
        mychecksum=mychecksum^buff[i];
    }
    return mychecksum==receivedchecksum;
}
```

AbsInt aiT



Can use up to 80µs of CPU for a 120 bytes long frame on a MPC555 at 40 MHz

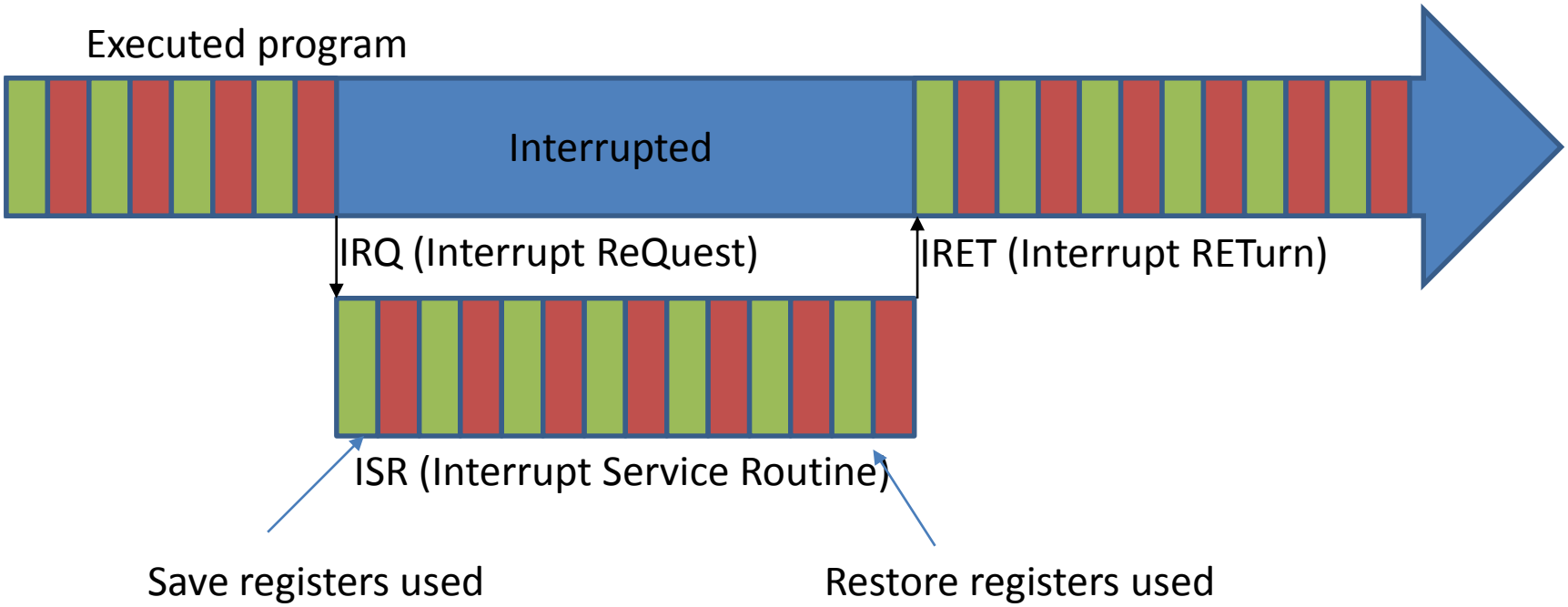
Interrupt

- What if…
  - Some byte arrives while the CPU is busy computing the checksum?
    - ✓ If it is not stored, it will be lost when the next byte arrives
    - ✓ Checksum takes 80 µs, interbyte separation is 17µs at 56k (and it is a very slow bus!!!)
- Good news
  - Interrupts exist for this purpose since the 1950's

Interrupt/contd

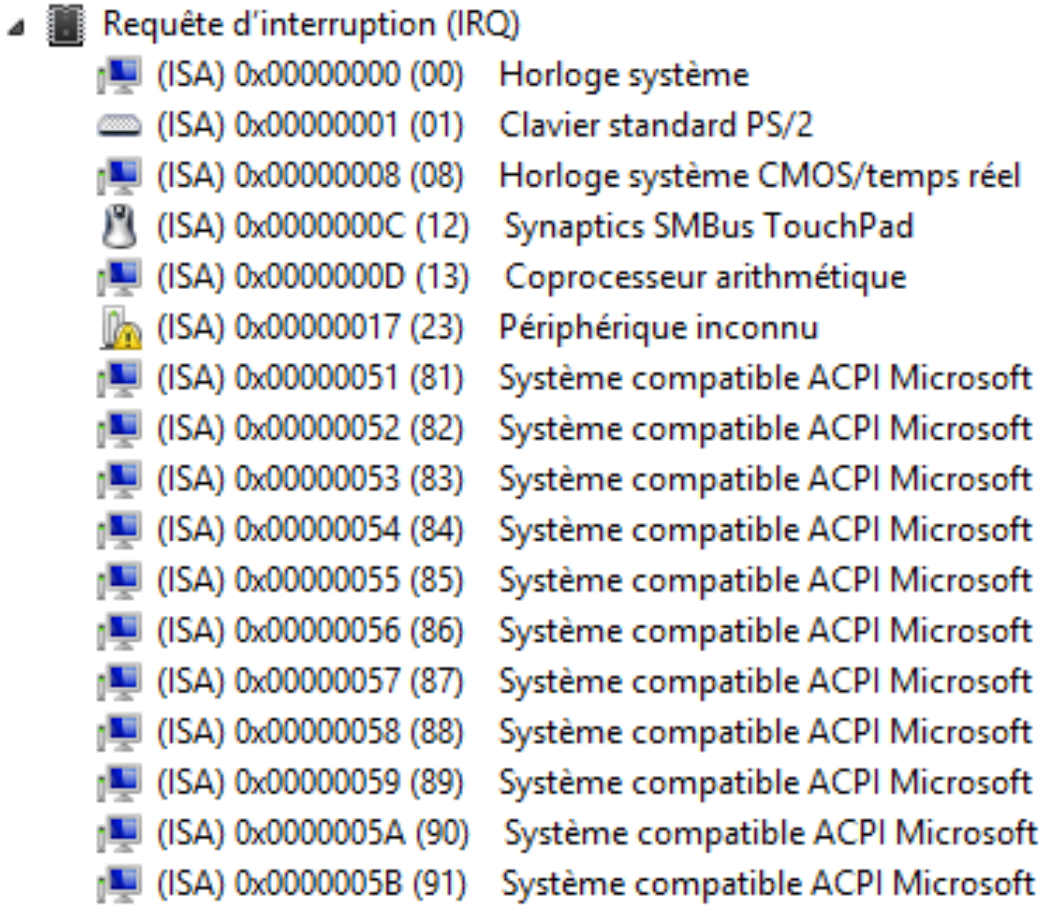
- Highly dependent of the hardware
  - pre-fetching/pipeline/…

Interrupt /contd



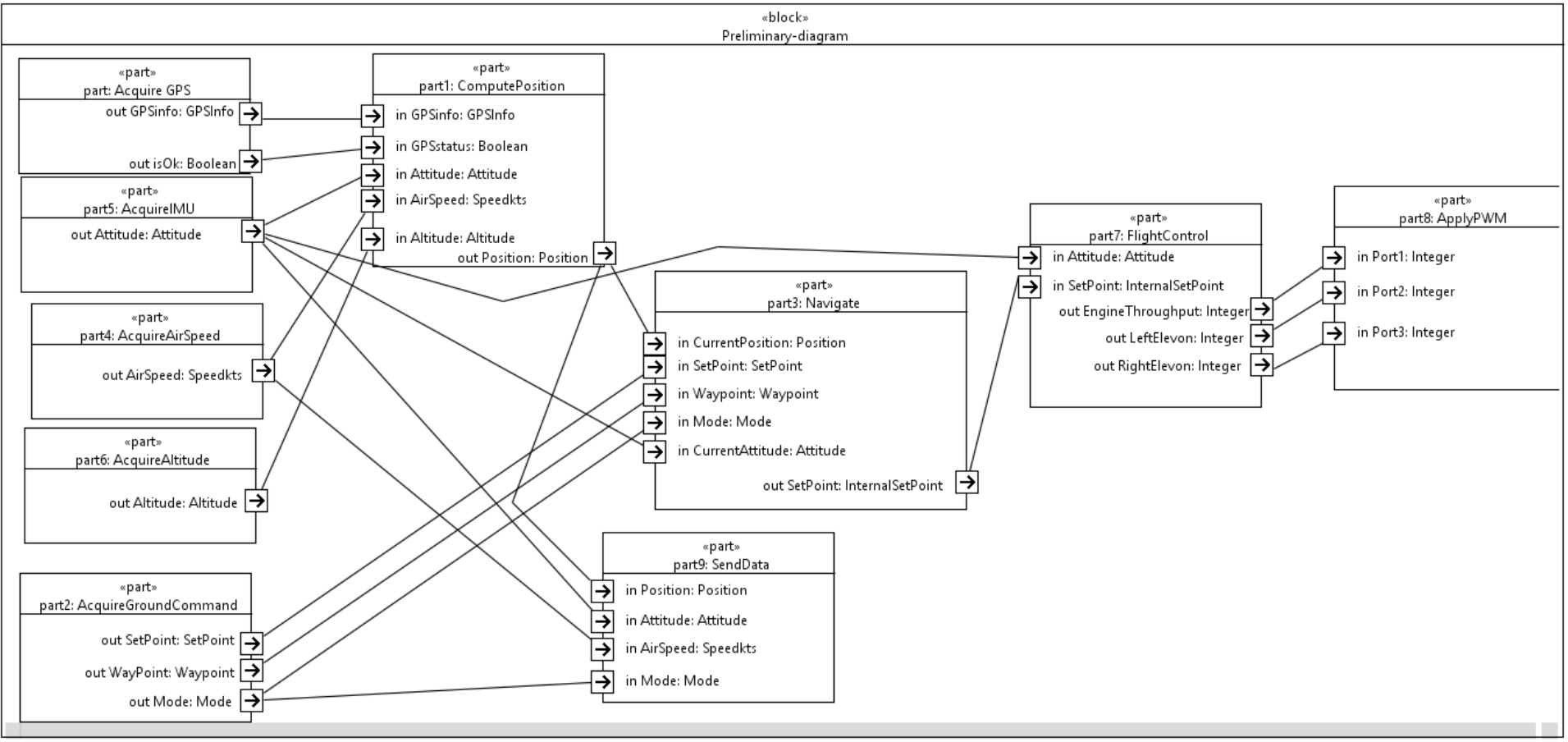
- Most I/O handling works this way

Interrupts on my PC

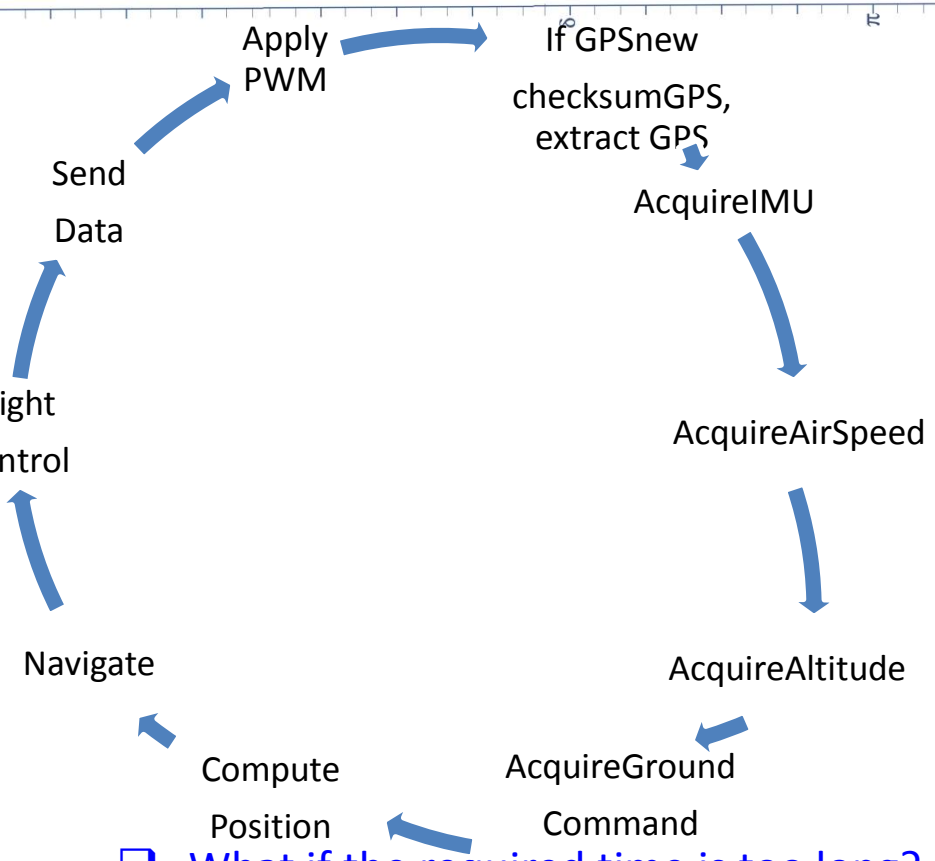


...

Functional specification



Time driven mono task programming



Interrupts are used to insure I/O

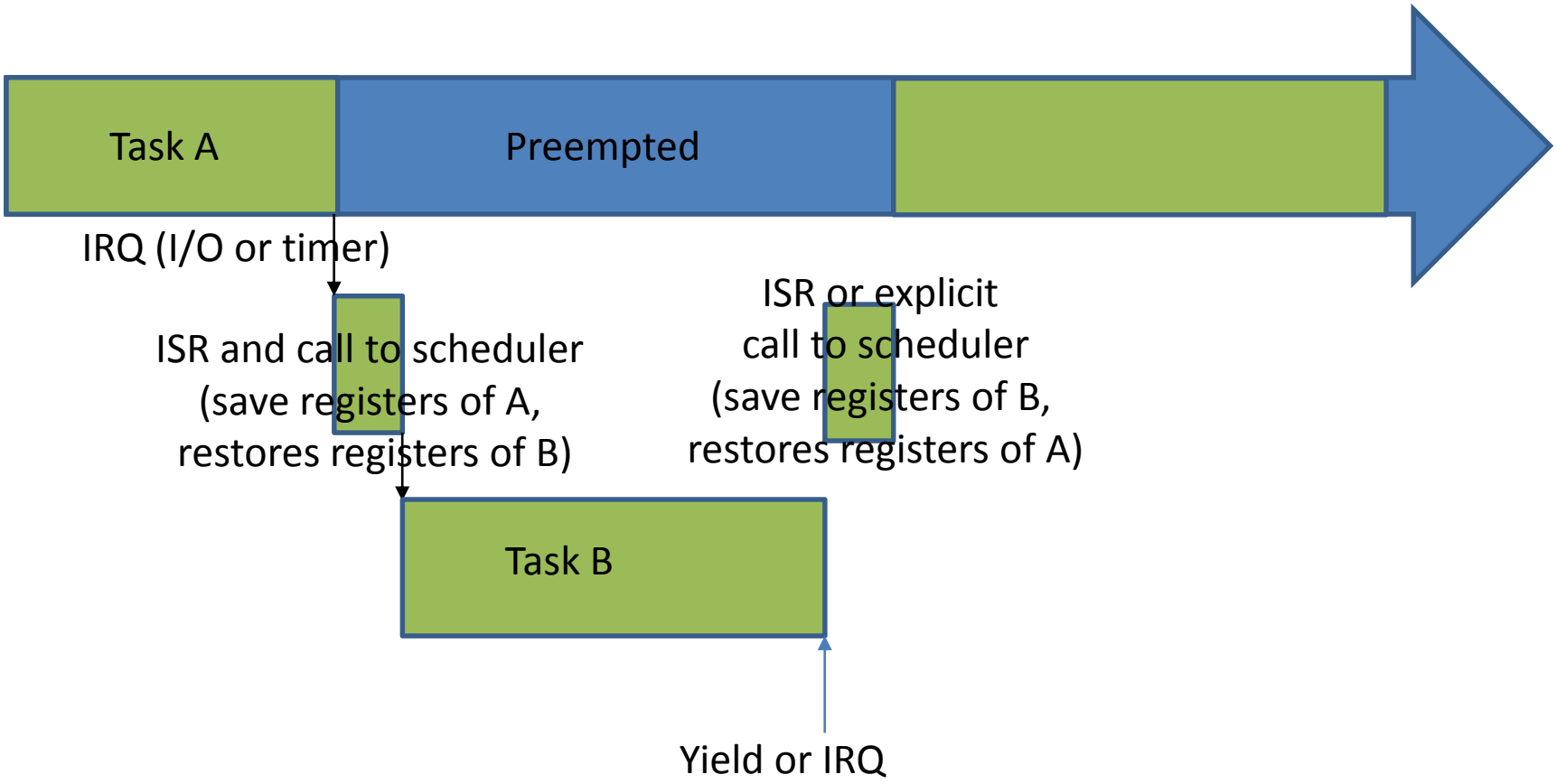
Example:

```
char GPSbuff[120];
int GPSidx=0;
int GPSnew=0;
int GPSstate=waitFirstByte;
```

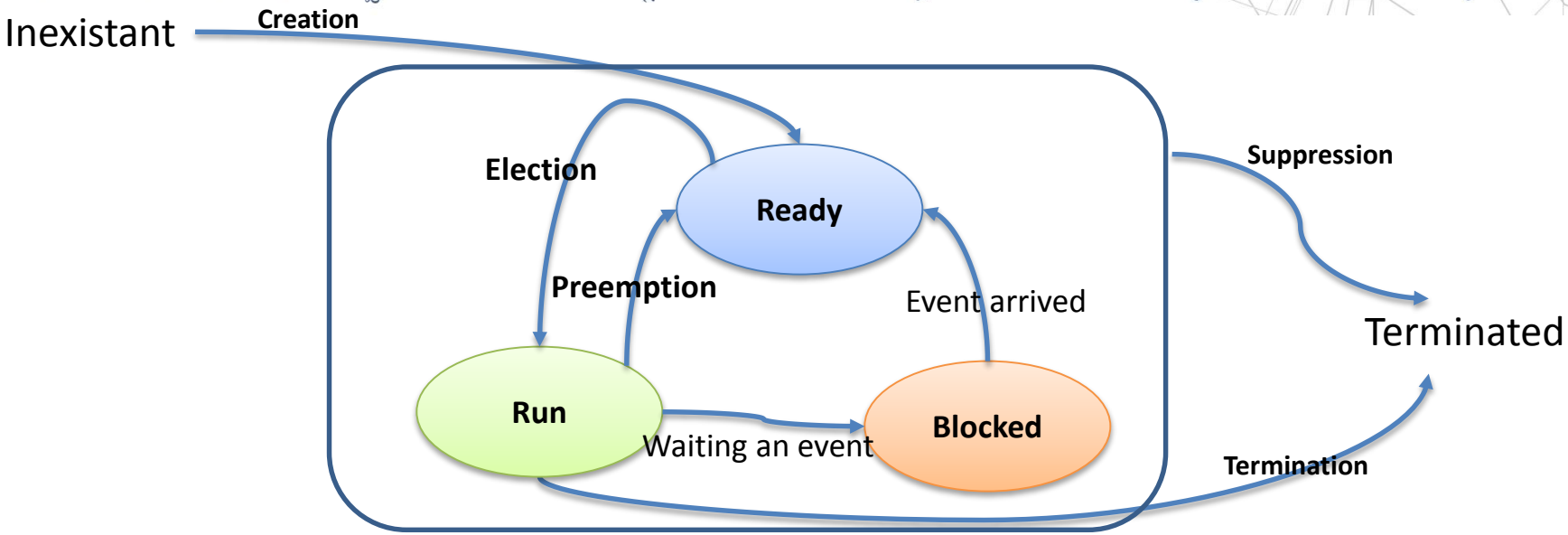
GPS serial ISR (∀**incoming serial byte**):
Same as before except when a frame is complete, we just set GPSnew to 1

- What if the required time is too long?
  - It will be the case unless the CPU is overpowered
- AcquireGPS and Navigation related 4 Hz, Attitude and flight control related 50 Hz, Altitude related 10 Hz, etc.
  - Different rythms for different dynamics

From interruption to preemption



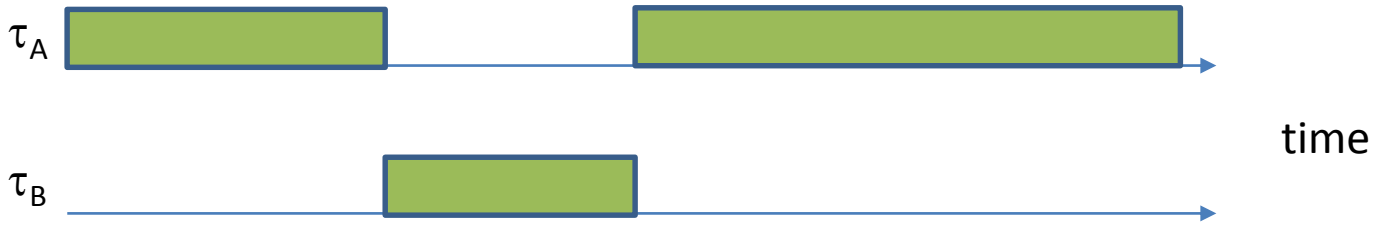
Task states



- The scheduler considers only ready tasks for election
- Blocked tasks do not use **any** processor
  - Waiting for some event or data => send task in blocked state

Gantt diagram

- Neglecting preemption overhead



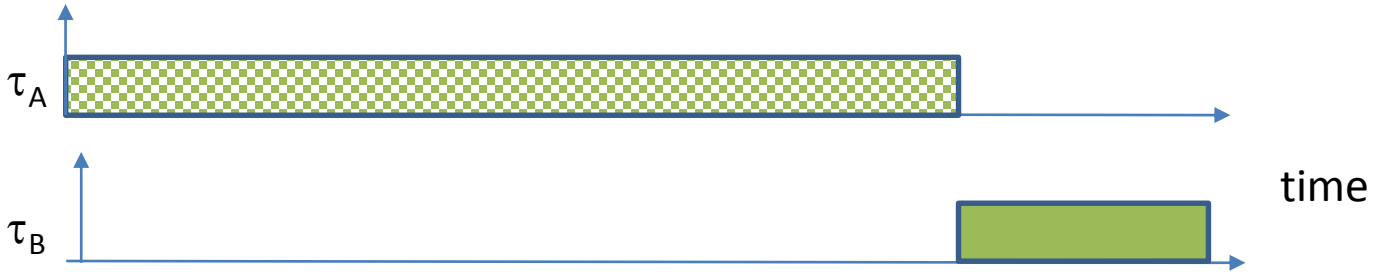
We will talk later about scheduling strategies

Main concurrency problems and paradigms

- Mutual exclusion
- Producer/Consumer
- Main tools offered by OS
  - Non-preemptive code (any CPU/OS),
  - Semaphore (every OS),
  - Mailbox/message queue (VxWorks, POSIX, RTEMS, …),
  - Hoare Monitor (POSIX, Ada)
    - ✓Most powerfull concept => can replace all

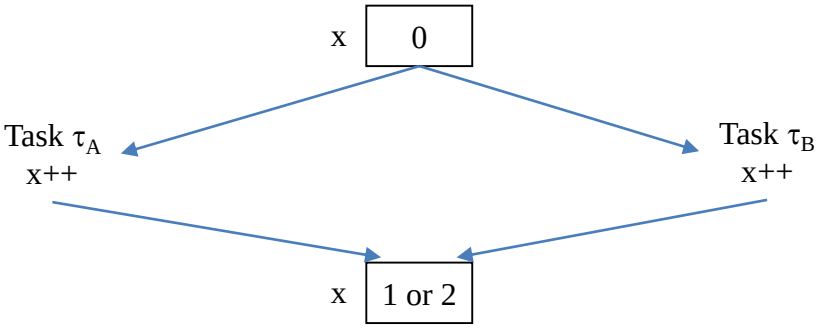
Non-preemptive code

- Easy but
  - Unsafe: what if the non-preemptive code countains an unbounded loop?
  - Worst-case response-time for other tasks or ISR

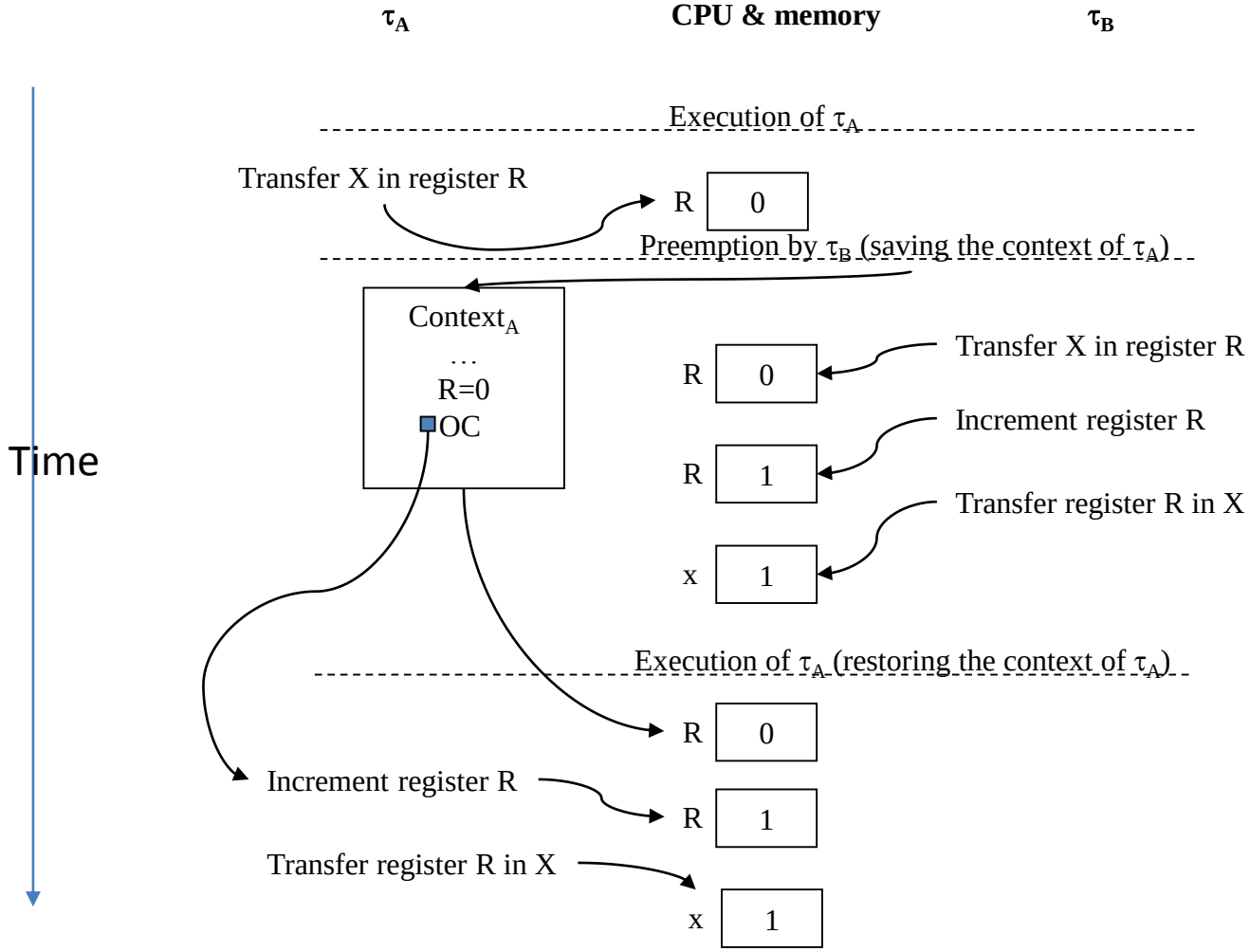


  - Can be used for very short and safe sections

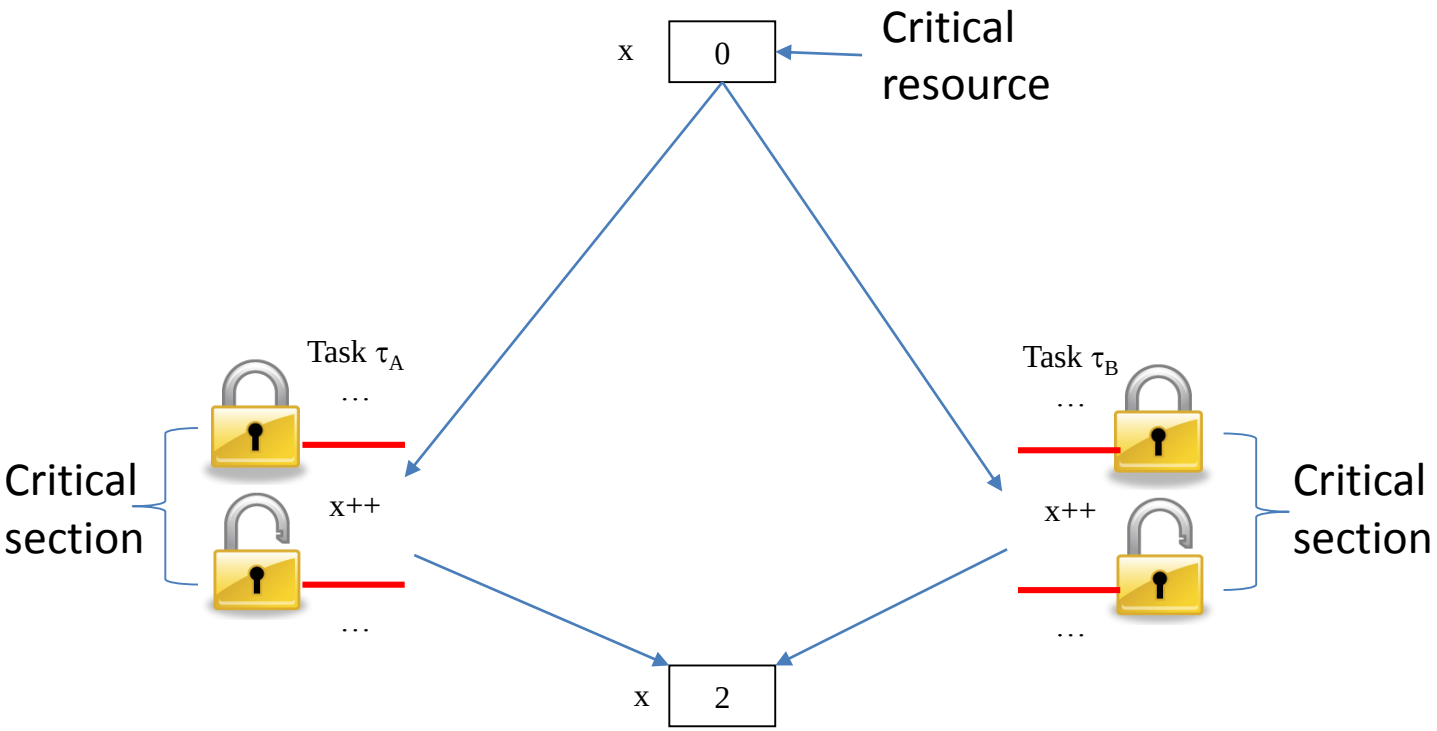
Mutual exclusion problem



Mutual exclusion problem



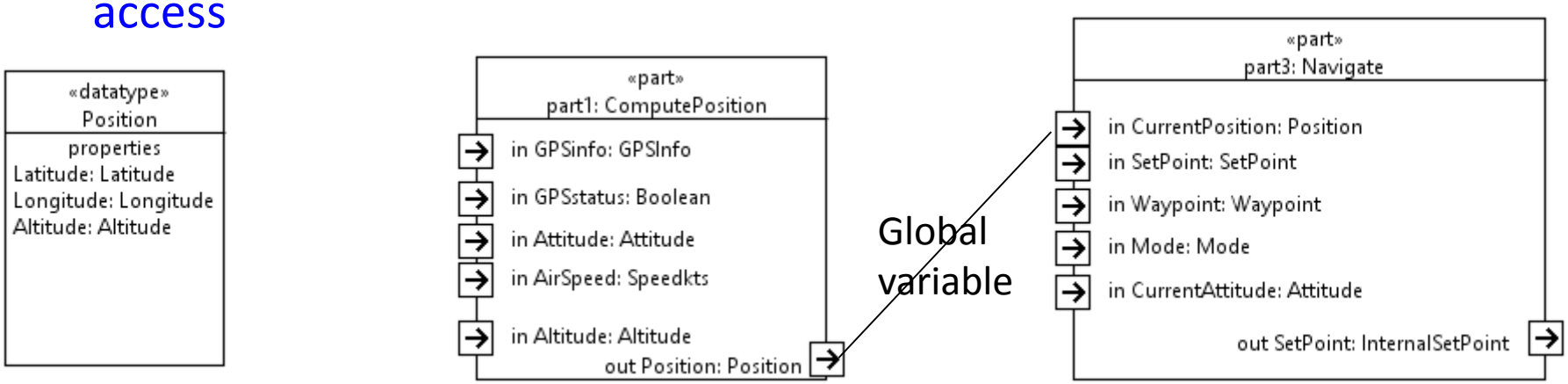
Mutual exclusion mechanisms



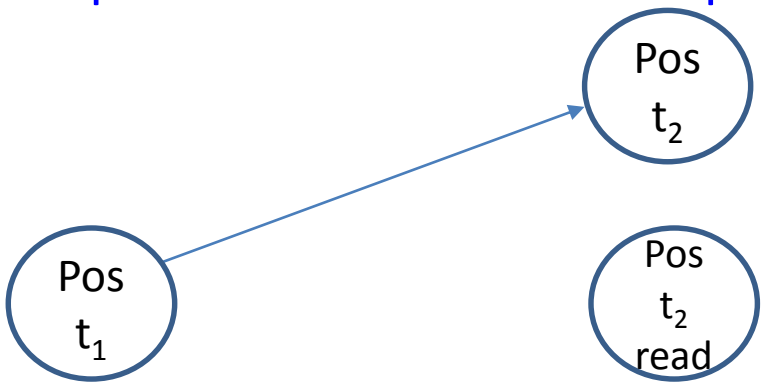
- Critical resource: shared data, shared hardware, non thread-safe function
- Semaphore or monitor

Shared data

- A classic example of mutual exclusion is protecting shared data access

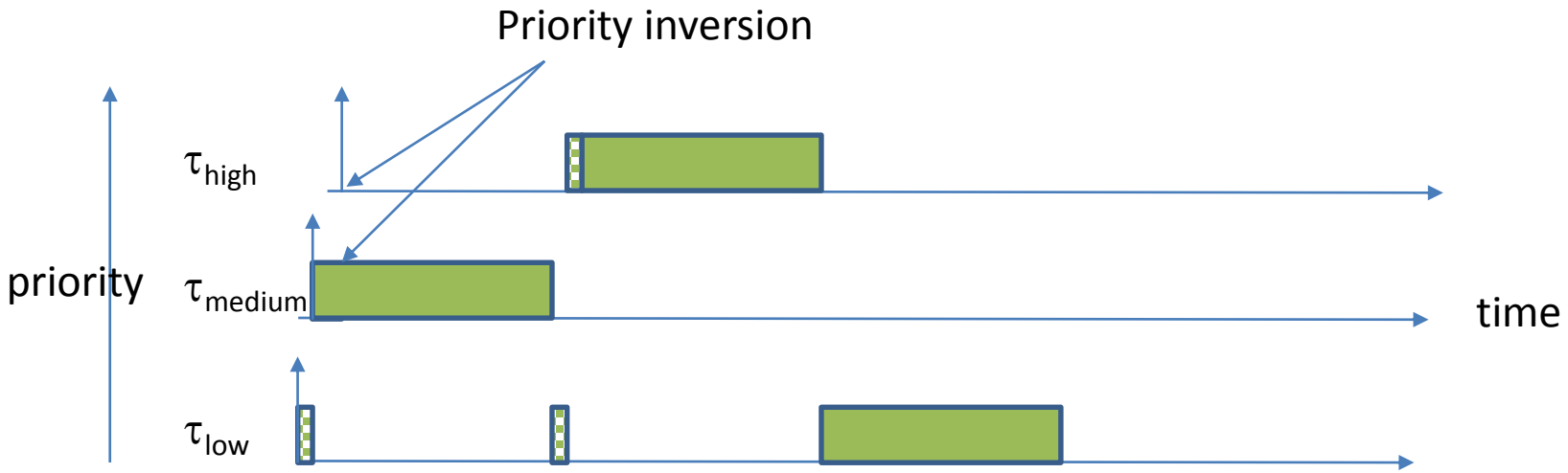


- A classic example of mutual exclusion is protecting shared data access



Mutual exclusion/end

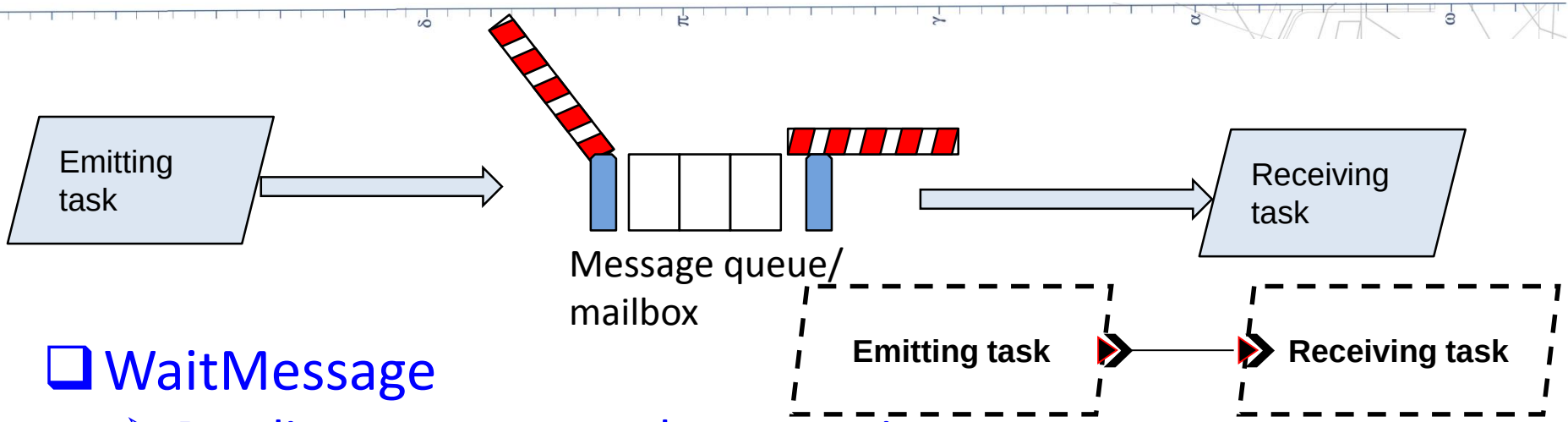
- Any critical resource (shared data, shared hardware, non thread-safe shared function) must be protected via a mutual exclusion mechanism



Resource management protocol

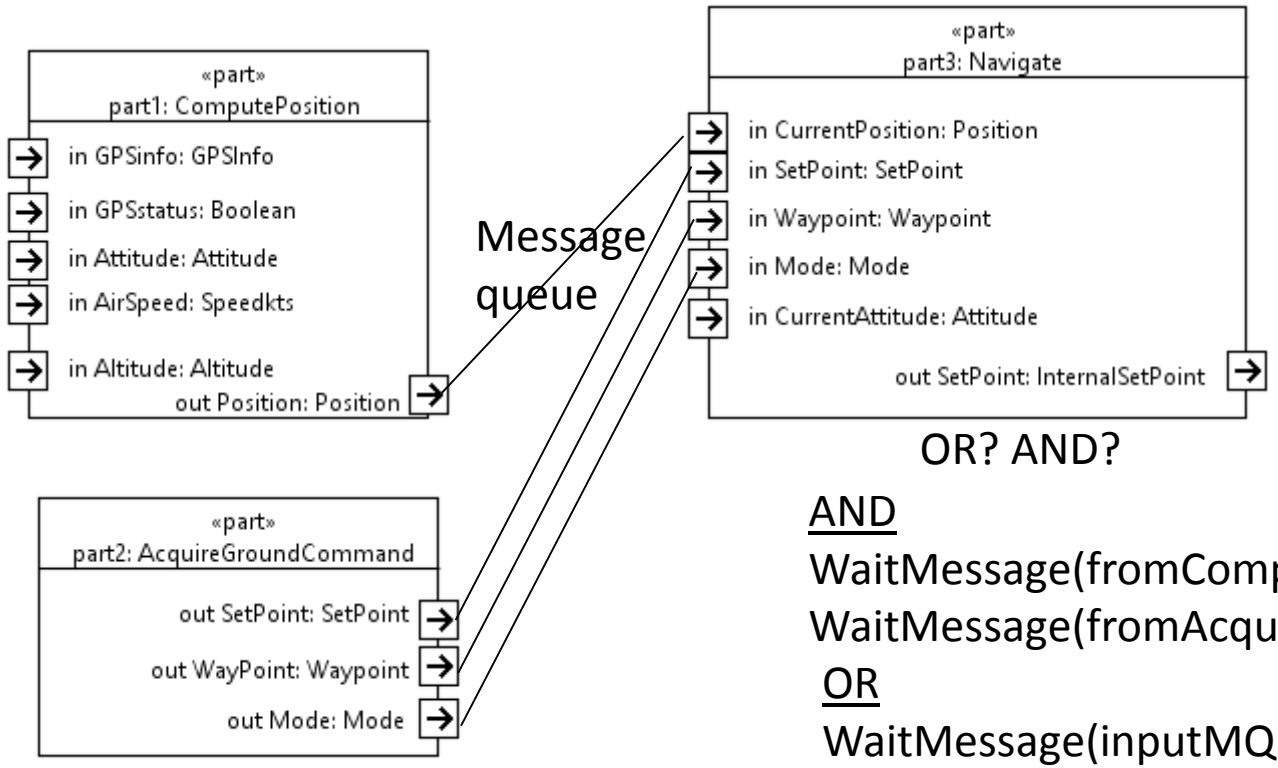
- Avoid priority inversion and bound maximum blocking time
  - Priority Inheritance Protocol (PIP)
  - Priority Ceiling Protocol (PCP)
  - Immediate Priority Ceiling Protocol (IPCP)
- Most implemented: IPCP
  - Critical resource ceiling= max prio tasks using it
  - When in a critical section inherit the ceiling prio

Producer/consumer



- WaitMessage
  - Reading a message « destroys » it
  - Task is blocked if queue is empty
  - Ready when a message arrives
- SendMessage
  - With overriding: if message queue is full, oldest message overridden
  - Without overriding: if message queue is full, task is blocked until a message is read

Loosely synchronous communication



OR? AND?

AND

WaitMessage(fromComputeMQ…);

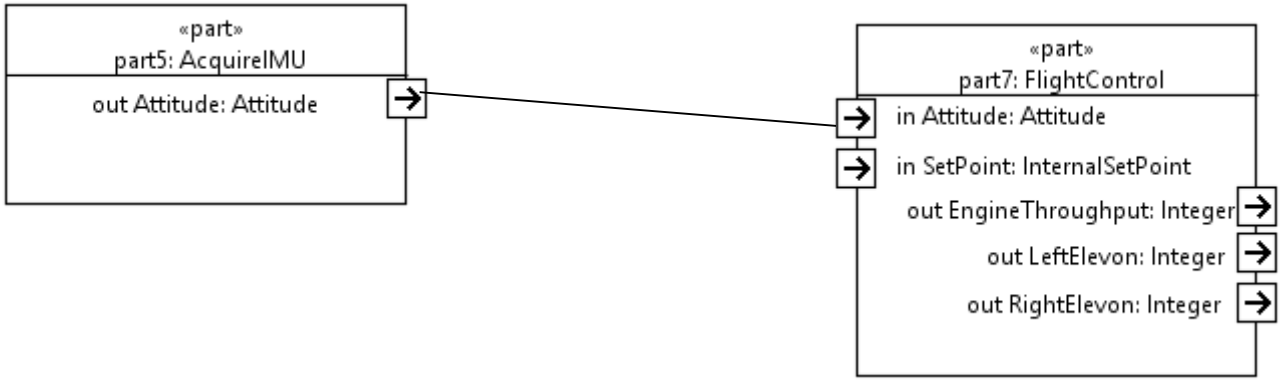
WaitMessage(fromAcquireMQ…);

OR

WaitMessage(inputMQ…);

- Message queue: loosely synchronous (non-ambiguous dataflow)
- Shared data: asynchronous

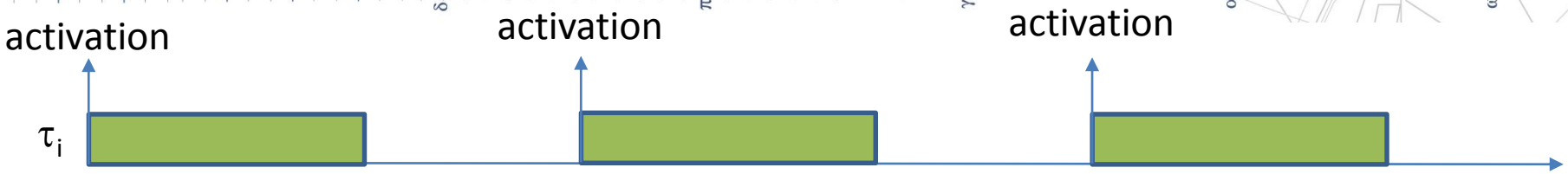
When to trigger functions?



- AcquireIMU: as soon as a frame arrives (I/O interrupt)? Periodically (timer interrupt)? Continuously (requires specific hardware)?
- Should FlightControl be triggered as soon as AcquireIMU computed an attitude? Periodically? Continuously?

4. MAPPING SOFTWARE ON HARDWARE

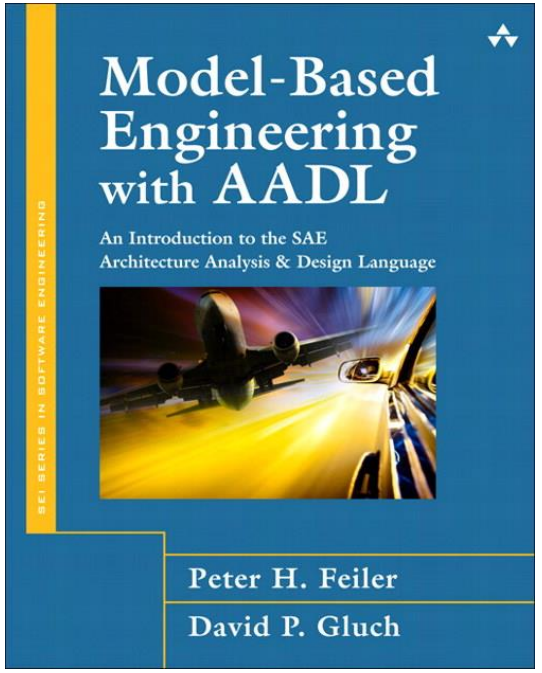
Rythm vs. Resource usage



- Processor utilization
  - $U_i=C_i/T_i$
    - ✓ C_i worst-case execution time
    - ✓ T_i most frequent activation period
  - Example
    - ✓ Serial byte acquisition at 57600 bauds
    - ✓ T_i=17µs
    - ✓ C_i=1 µs
    - ✓ U_i~6%

AADL

AADL SAE INTERNATIONAL



- 2004 v1, 2009 v2
- Domain Specific Modeling Language (DSML) for high integrity systems
- SAE (Society of Automotive Engineers) standard
- Committee leads
  - B. Lewis (US Army), P. Feiler (Soft. Eng. Instit.), O. Sokolski (U. Penn), J. Hugues (ISAE)
- Participants
  - Airbus, Boeing, Dassault, Lokheed Martin, Rockwell, ESA, Toyota, US Army, US Navy, etc.

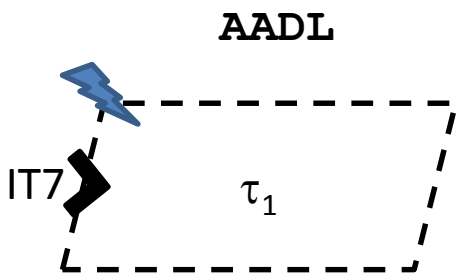
Triggering tasks

- A task is a (sequential) function
  - with a stack except in some very specific cases
  - Can be triggered
    - ✓By an interrupt (I/O, timer, etc.)
    - ✓By another task (producer/consumer)
  - Can yield
    - ✓Wait for an interrupt (I/O, timer, etc.)
    - ✓Wait for another task to trigger it (wait semaphore, wait message,…)

Event driven task

Event driven ≈ interrupt driven

Task τ_1
Initialize();
Loop
    Wait event
    Iterate();
End loop



With the help of the API

ReadSerial(buff,10);

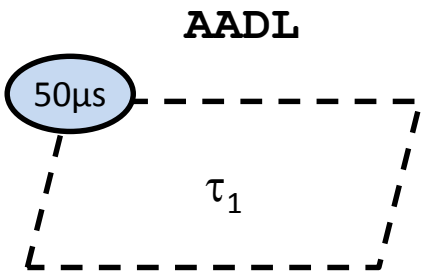
Using an ISR/DSR (e.g. in VxWorks)

```
SEM_ID trigger;
triggerT1 = semCCreate(SEM_Q_PRIORITY,0);
intConnect(INUM_TO_IVEC(7), myISR,0);
void myISR(int dummy) {
   semGive(triggerT1);
}
void T1 () {
   for (;;) {
      semTake(triggerT1,WAIT_FOREVER);
      …
   }
}
```

Time driven task

Time driven

Task τ_1
Initialize();
Loop
    Iterate();
    Wait next (periodic) release
End loop



POSIX pthread (1003.1)

```
struct timespec {
  time_t   tv_sec;    /* Seconds */
  long     tv_nsec;   /* Nanoseconds */
};

void add_us(struct timespec *time,
                                    long us) {
   (*time).tv_nsec+=(us%1000000)*1000;
   (*time).tv_sec+=(us/1000000)+
            ((*time).tv_nsec/1000000000);
   (*time).tv_nsec%=1000000000;
}

Initialize()
clock_gettime(CLOCK_MONOTONIC,&releasetime);
For (;;) {
   Iterate();
   add_us(&releasetime,50);
   clock_nanosleep(CLOCK_MONOTONIC,
      TIMER_ABSTIME,&releasetime,0);
}
```

Time driven task

Osek

OIL

```
TASK T1 {
   AUTOSTART = FALSE;
   …
};
ALARM cyclic_alarm1{
   …
   ACTION = ACTIVATETASK {
      TASK = T1;
   };
   …
};
```

C

```
TASK T1() {
   Iterate();
}
```

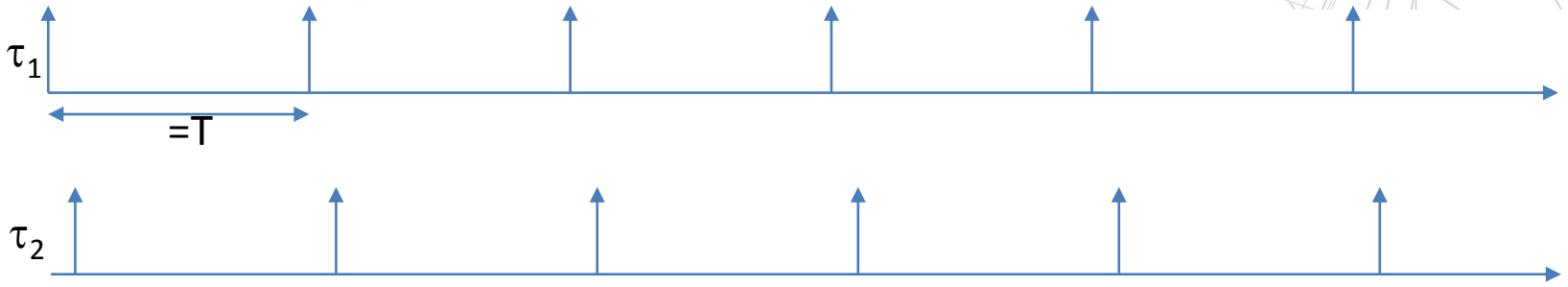
Note: Initialize() is executed at OS initialization

VxWorks

```
Initialize();
For (;;) {
   Iterate();
   delay(1);
   /* Assuming a 50µs quantum */
}
```

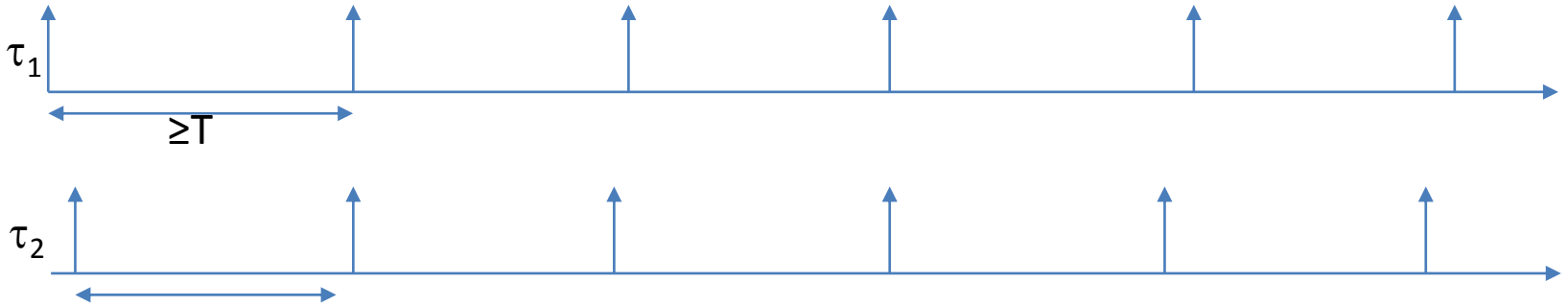
Clock drift and simultaneous first release

POSIX
Osek



No defined « zero » but the delay is kept between T_1 and T_2

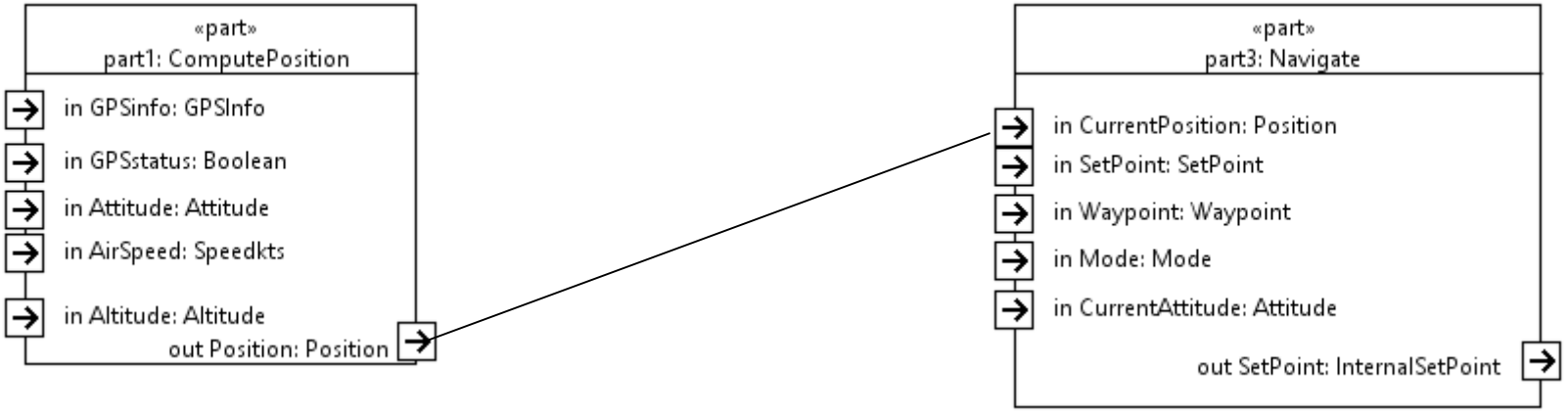
VxWorks



No defined « zero » and release dates may drift

*If T multiple of the hardware clock

Software triggering

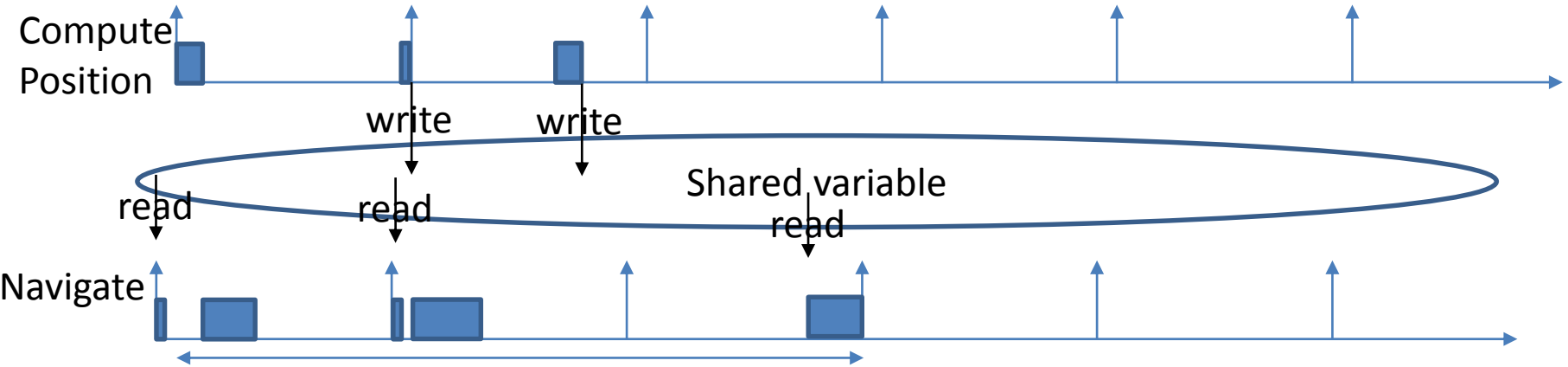


$U_{CompPos}=C_{CompPos}/T$

- How should Navigate be triggered?

Asynchronous communication

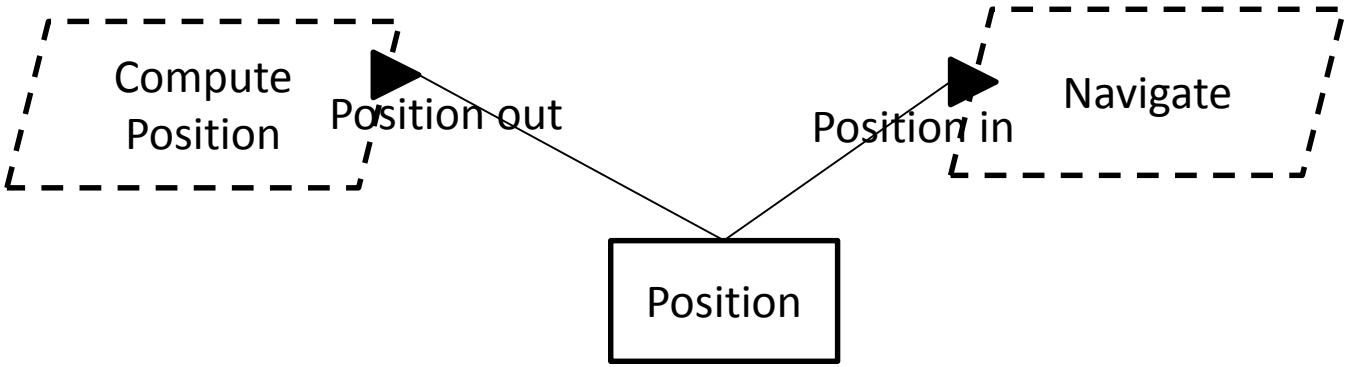
- Case 1: navigate has its own rythm
  - Assume it to be periodic of period T, independently released compared to ComputePosition
    - ✓ Assumption: each occurrence executed within the period



    - ✓ Delay up to 3T
    - ✓ Some values are never read except if $T_{reader} \leq T_{writer}/2$
      - But then U_{reader} is twice larger

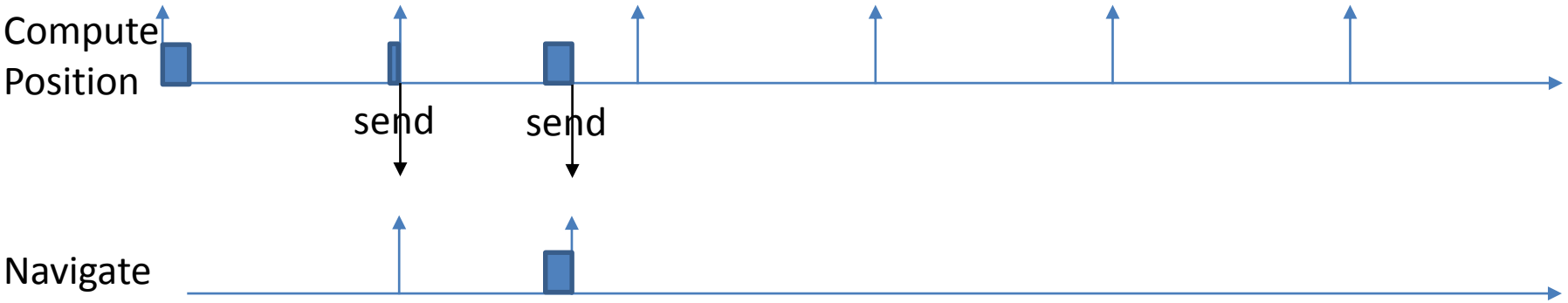
Asynchronous communication in AADL

- Data port
- Or equivalently by data storage

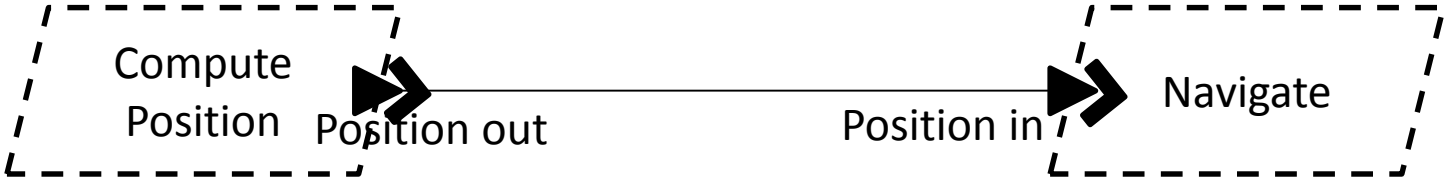


Loosely synchronous communication

Case 2: Producer/Consumer paradigm



- Delay up to 2T, for same U



- Properties are used to tune the behavior of the event-data port (queue_size, overrinding, etc.)

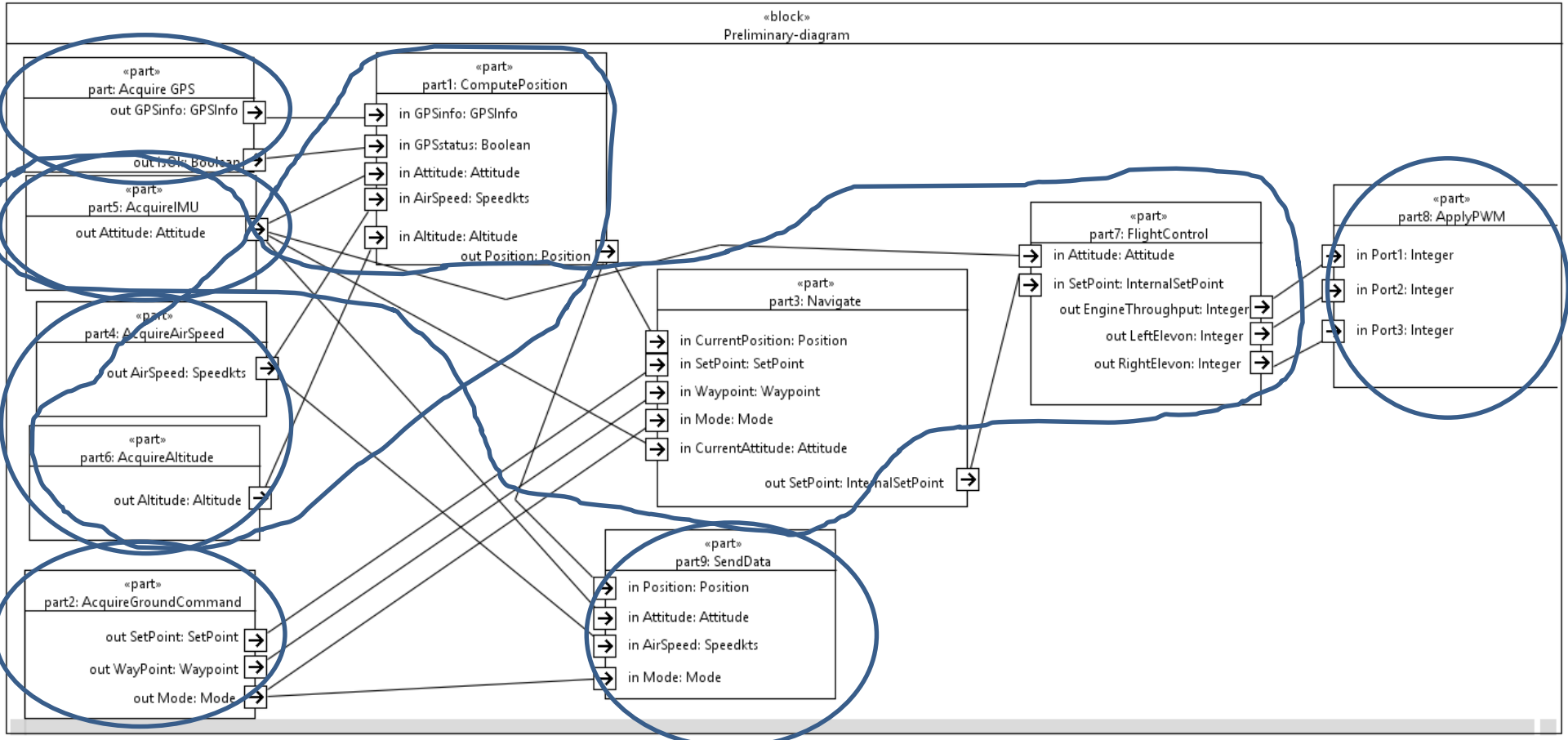
Good old sequence

- Delay up to T for the same U

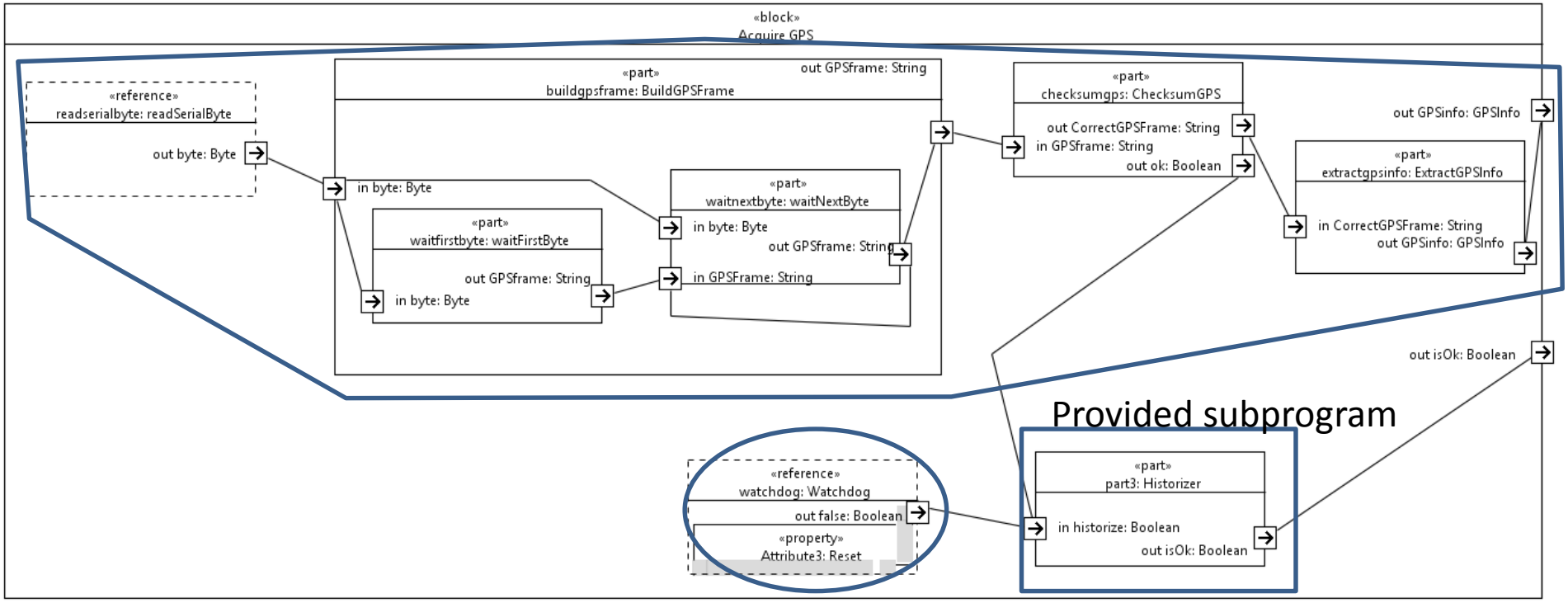
Summarizing

- On a single processor, the best reactivity for the same processor utilization is
  - If possible, sequence
  - Else if possible, loosely synchronous
  - Else asynchronous
- On several processors
  - NP-hard problem

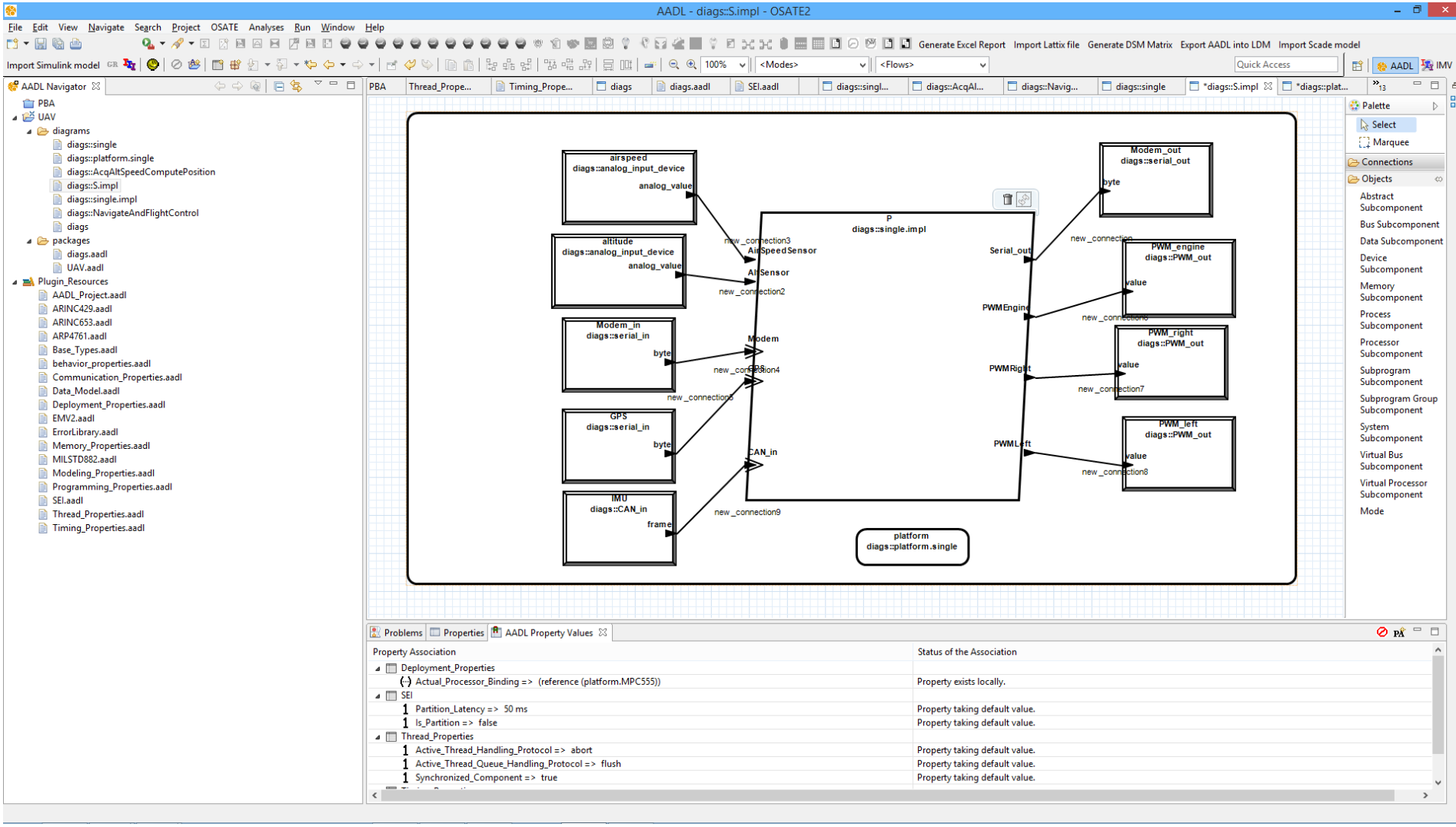
UAS example


«block»
Preliminary-diagram
«part»
part: Acquire GPS
out GPSinfo: GPSInfo
out FSI: Boolean
«part»
part5: AcquireIMU
out Attitude: Attitude
«part»
part4: AcquireAirSpeed
out AirSpeed: Speedkts
«part»
part6: AcquireAltitude
out Altitude: Altitude
«part»
part2: AcquireGroundCommand
out SetPoint: SetPoint
out WayPoint: Waypoint
out Mode: Mode
«part»
part1: ComputePosition
in GPSinfo: GPSInfo
in GPSstatus: Boolean
in Attitude: Attitude
in AirSpeed: Speedkts
in Altitude: Altitude
out Position: Position
«part»
part3: Navigate
in CurrentPosition: Position
in SetPoint: SetPoint
in Waypoint: Waypoint
in Mode: Mode
in CurrentAttitude: Attitude
out SetPoint: InternalSetPoint
«part»
part9: SendData
in Position: Position
in Attitude: Attitude
in AirSpeed: Speedkts
in Mode: Mode
«part»
part7: FlightControl
in Attitude: Attitude
in SetPoint: InternalSetPoint
out EngineThroughput: Integer
out LeftElevon: Integer
out RightElevon: Integer
«part»
part8: ApplyPWM
in Port1: Integer
in Port2: Integer
in Port3: Integer

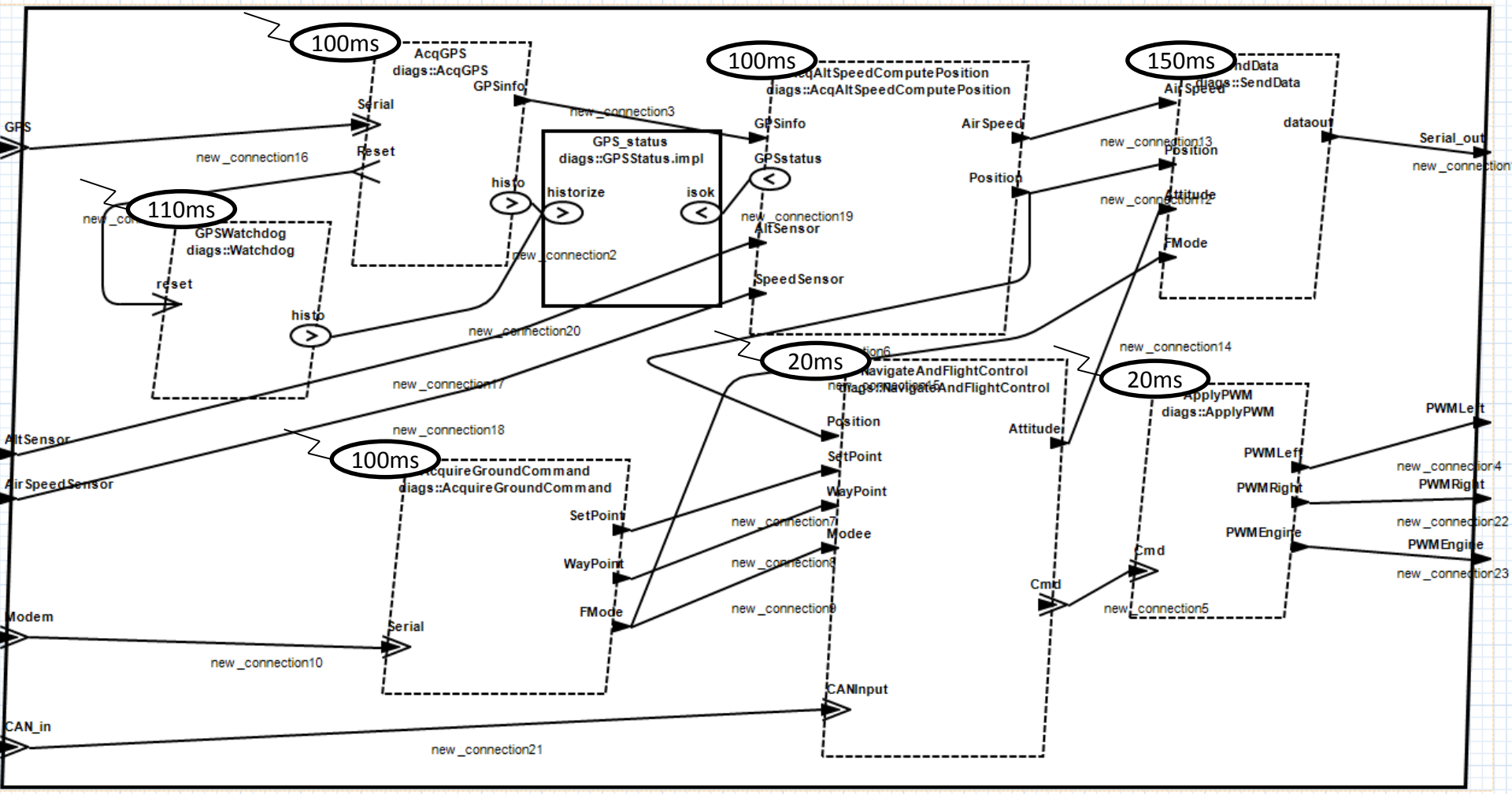
UAS example/2



Software AADL design



Software AADL design

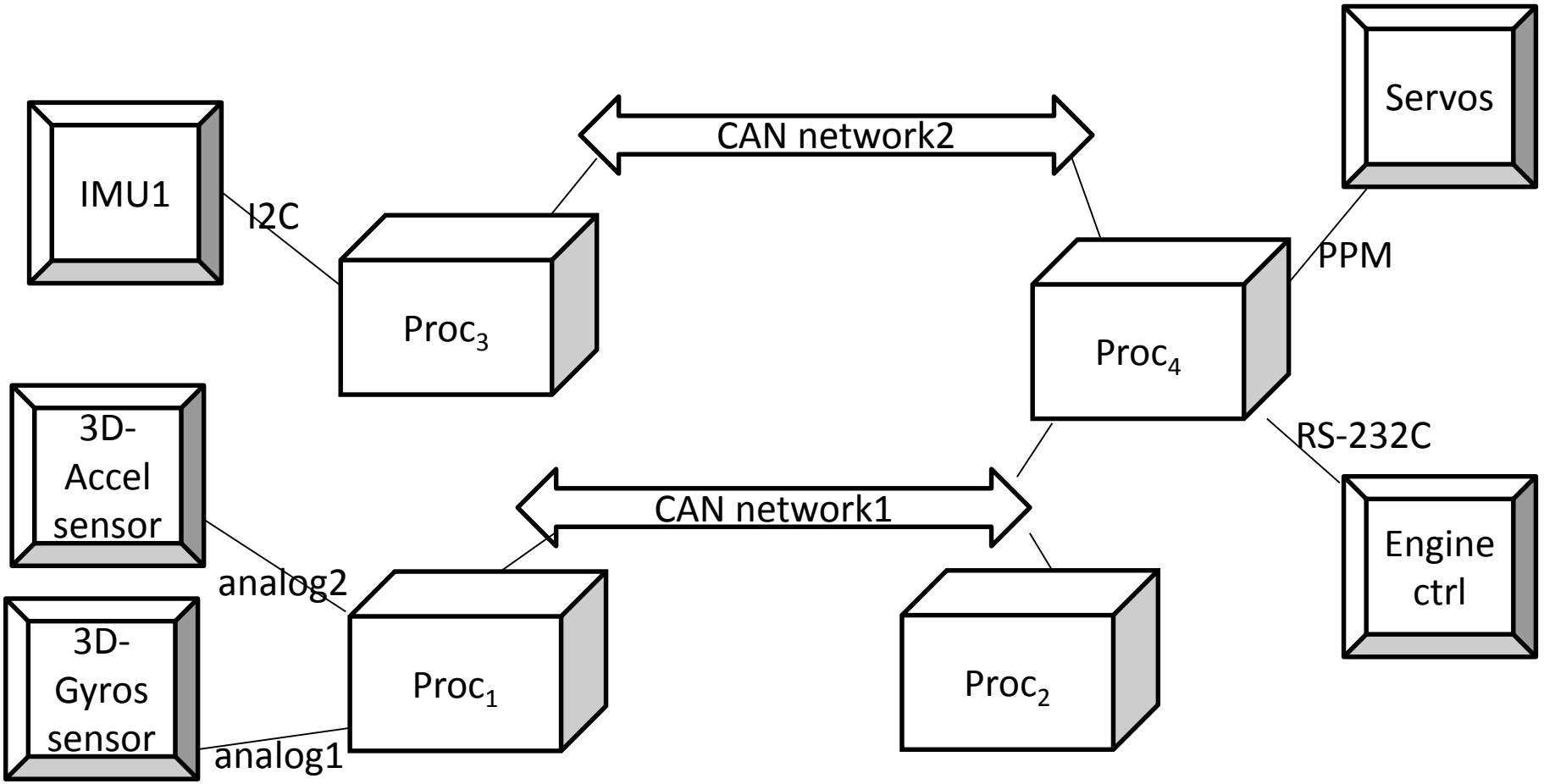


AADL representations

```
properties
Dispatch_Trigger => (reference (Serial));
Dispatch_Protocol => Sporadic;
Period => 100 ms;
end AcquireGroundCommand;
thread SendData
features
dataout : out data port;
Position : in data port;
Attitude : in data port;
AirSpeed : in data port;
FMode : in data port;
end SendData;
thread Watchdog
features
reset : in event port;
histo : requires subprogram access historize;
end Watchdog;
system S

end S;
Processor MPC555
end MPC555;
system platform
end platform;
system implementation platform.single
subcomponents
MPC555: processor MPC555;
end platform.single;
system implementation S.impl
subcomponents
platform: system platform.single;
P : process single.impl;
airspeed : device analog_input_device;
altitude : device analog_input_device;
GPS : device serial_in;
Modem_in : device serial_in;
Modem_out : device serial_out;
PWM_right : device PWM_out;
PWM_left : device PWM_out;
PWM_engine : device PWM_out;
IMU: device CAN_in;
connections
new_connection : port P.Serial_out -> Modem_out.byte;
new_connection2 : port altitude.analog_value -> P.AltSensor;
new_connection3 : port airspeed.analog_value -> P.AirSpeedSensor;
new_connection4 : port Modem_in.byte -> P.Modem;
new_connection5 : port GPS.byte -> P.GPS;
new_connection6 : port P.PWMEngine -> PWM_engine.value;
new_connection7 : port P.PWMRight -> PWM_right.value;
new_connection8 : port P.PWMLeft -> PWM_left.value;
new_connection9 : port IMU.frame -> P.CAN_in;
properties
Actual_Processor_Binding => (reference (platform.MPC555));
end S.impl;
end diags;
```

+XML/XMI

Hardware design



Technology behind ADLs

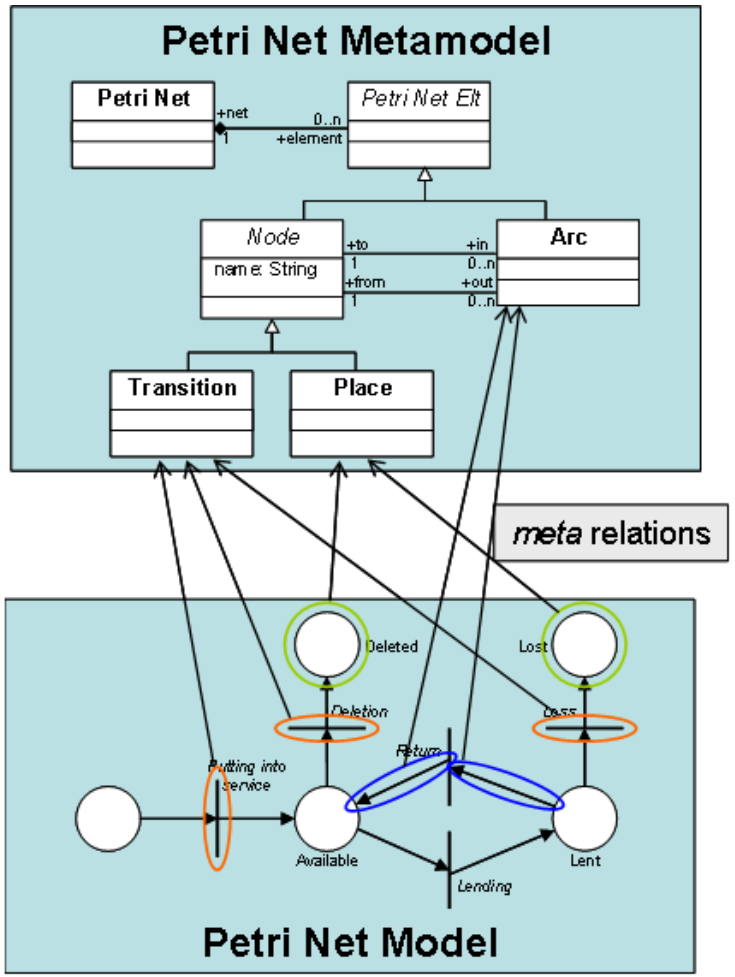
- Common meta-meta-models
  - Application architecture is an instance
  - Elements rely on a user-defined model
  - Relying on AADL meta-model
  - Relying on the EMF (Eclipse Modeling Framework) meta-meta-model

Meta-meta-meta-meta*

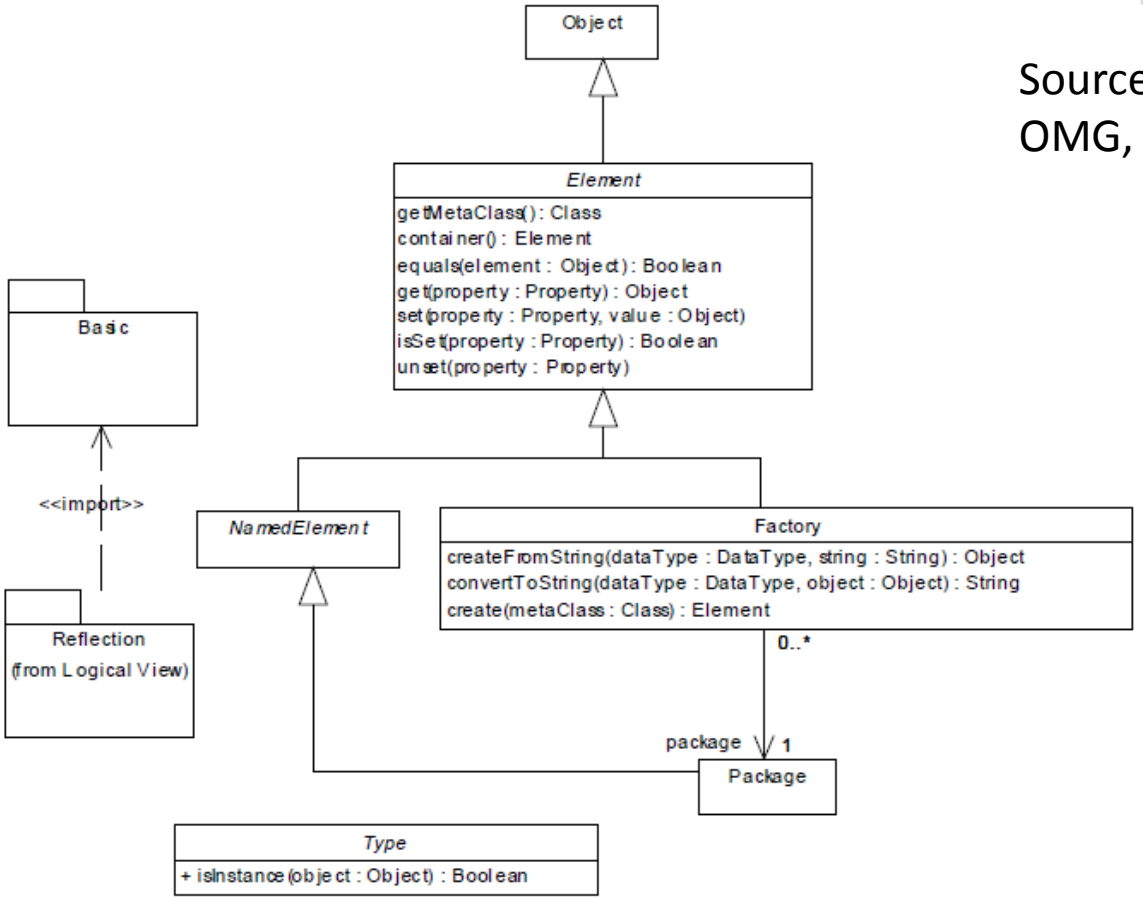
M_3 Meta Meta Model	Examples : MOF, BNF, Ecore
M_2 Meta Model	Examples: XMI, DSL, UML, UML profile
M_1 Model	Example: class diagram
M_0 Objetcs/Instances	Example: instance of a model

Example

- Source wiki.eclipse.org



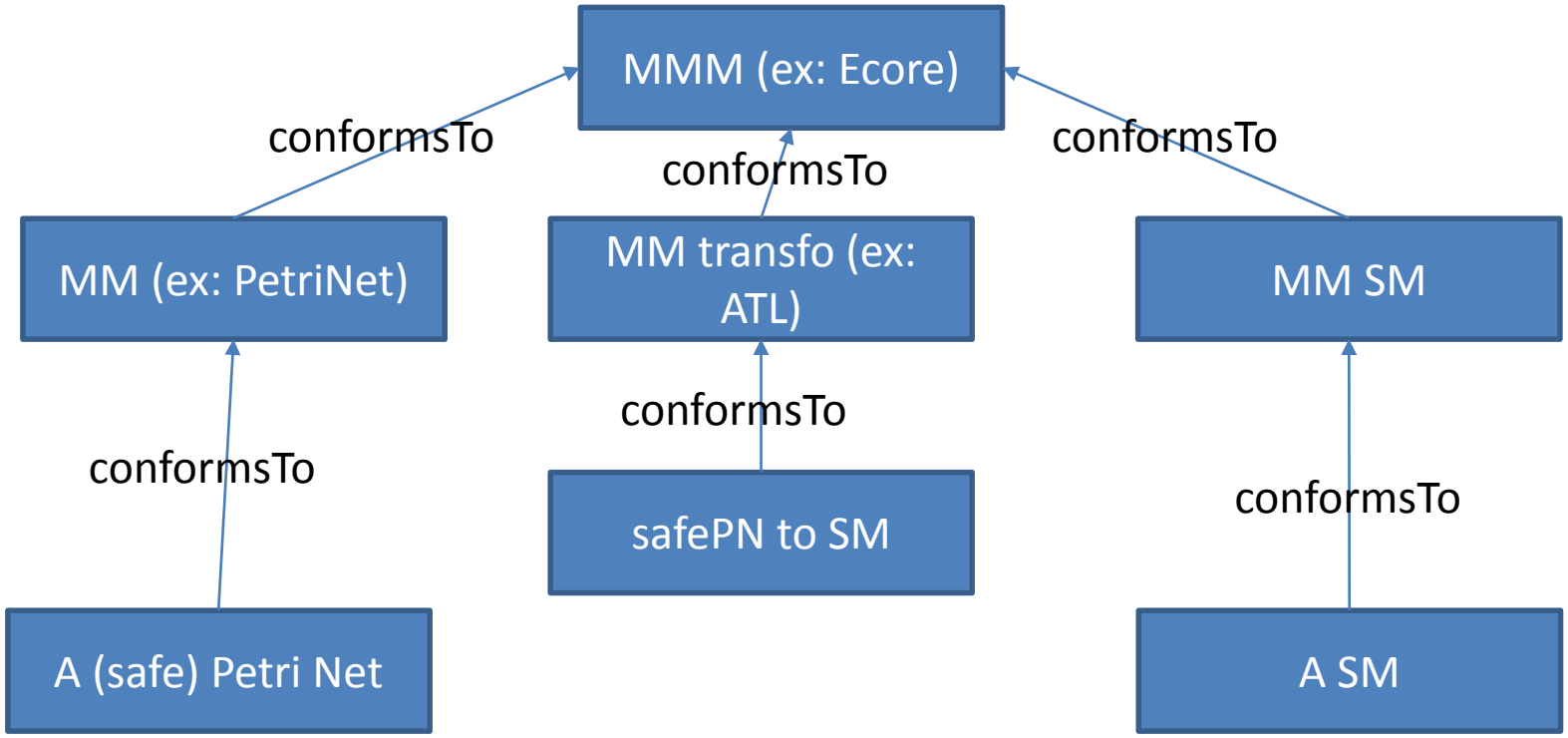
MOF: example



Source:
OMG, UML 2 standard

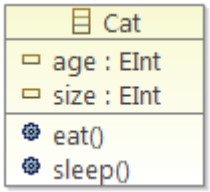
- Reflexivity, introspection, serialization in MOF2.0

Model transformation



Code generators

- Example: Acceleo (Obeo) based on OMG Model-to-text specification




```
[comment encoding = UTF-8 /]
[module generate('http://www.eclipse.org/emf/2002/Ecore')/]

[template public generate(e : EClass)]
[comment @main /]
[file (e.name + '.java', false, 'UTF-8')]
public class [e.name.toUpperFirst()/] {
    [e.generateAttributes()/]
    [e.generateMethods()/]
}
[/file]
[/template]

[template public generateAttributes (eClass : EClass) ]
[for (attribute : EAttribute | eClass.eAllAttributes)]
/**
 * The documentation of [attribute.name/]
 */
private [attribute.eType.instanceClassName/] [attribute.name/];

[/for]
[/template]

[template public generateMethods (eClass : EClass) ]
[for (operation : EOperation | eClass.eAllOperations)]
/**
 * The documentation of [operation.name/]
 */
public void [operation.name/]() {

}

[/for]
[/template]
```

```
public class Cat {
    /**
     * The documentation of age
     */
    private int age;

    /**
     * The documentation of size
     */
    private int size;


    /**
     * The documentation of eat
     */
    public void eat() {

    }

    /**
     * The documentation of sleep
     */
    public void sleep() {

    }


}
```

Image source: wikipedia

Creating a language

- Extend UML or another UML profile
  - UML profile
    - ✓Stereotypes and tagged values
    - ✓Constraints, expressed, for example in OCL
  - Domain Specific Language (DSL)
    - ✓Based on a meta meta model (Ecore)
- Easy integration in UML based frameworks
- Example : labs of 4 hrs in ISAE-ENSMA
  - Create a simple robot manipulation language with code generator for NXT robots

Resources and consumers

Need resources

Processes
Need memory

Threads (tasks)
need core
cycles&cache&
memory access

T1 on
$Proc_1$

message

T2 on
$Proc_2$

Needs bandwidth

Resources

Cores (schedulers)
L1, L2 Cache (caching algorithms)
Memory bus (arbitration)

Computing resources

Memory (allocation & partitioning)

Placement resources

Bus/bandwidth (arbitration)

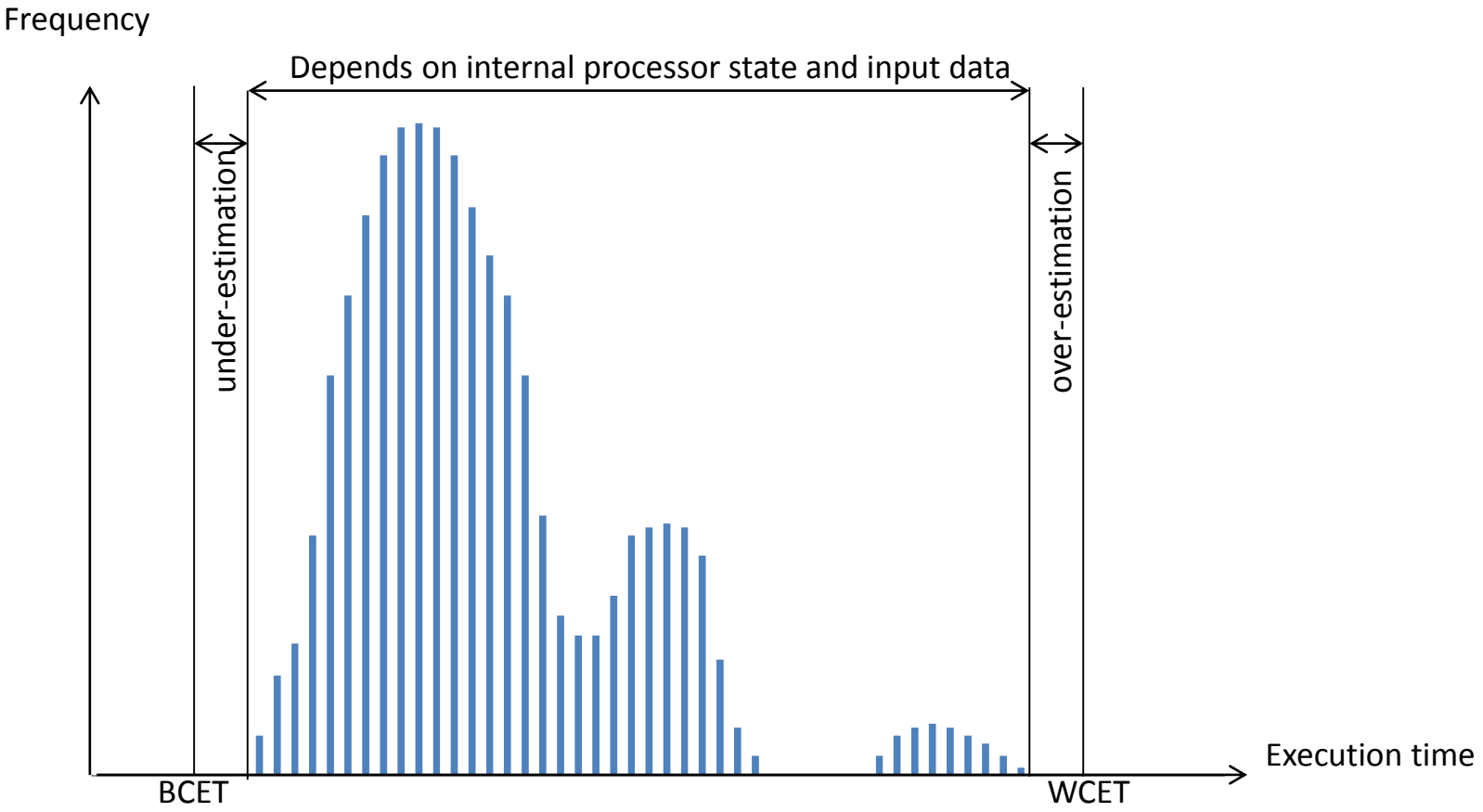
Communication resources

5. TIMING&SCHEDULABILITY ANALYSIS

Timing analysis

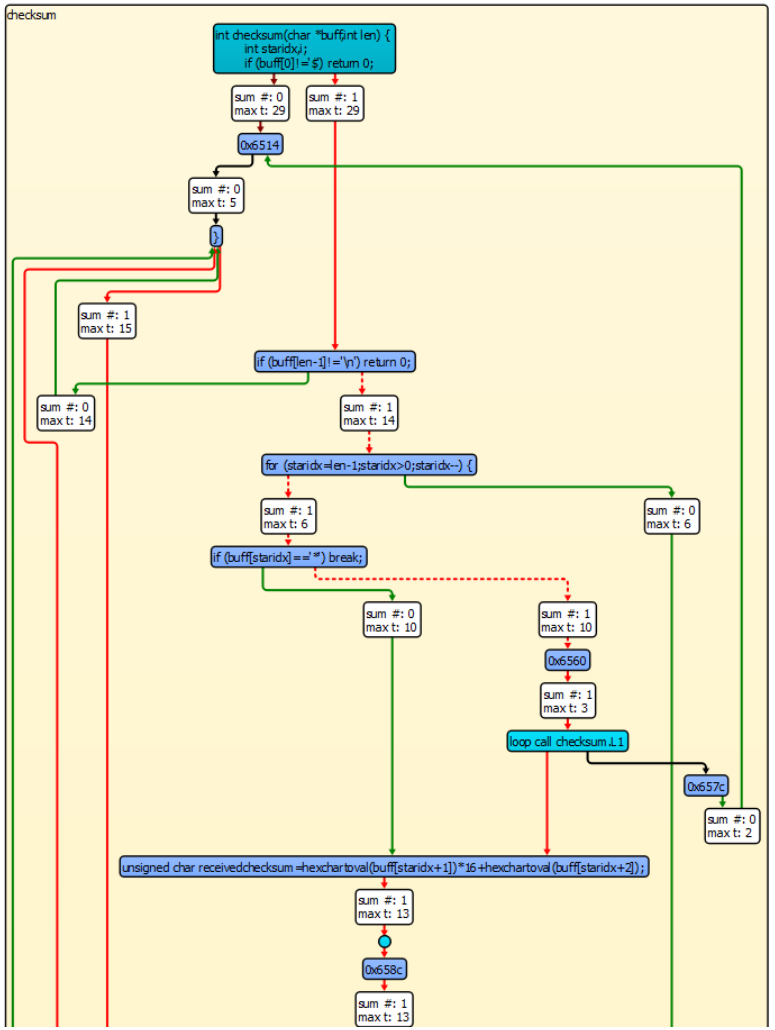
- Compute the Worst (Best, Average)-Case Execution Time
  - WCET, BCET, ACET
- Static analysis or dynamic analysis
  - Dynamic : testing testing testing, add a safety margin
    - ✓Easy but not safe
  - Static : analyze the CFG (Control Flow Graph)
    - ✓Count the CPU cycles
    - ✓With cache memories => NP-hard

Dynamimc analysis example



Static Analysis example

- CFG of a function
- Critical path in red
- Every loop must be bounded
- Every I/O must be bounded



Accounting for preemptions

- Just a sequential program
- Interrupts and preemptions
- In the over-estimation
  - How to be safe (conservative) without having a prohibitive over-estimation?

Scheduling algorithms

- Hundreds of scheduling algorithms
  - But only a few are really used
- Most embedded RTOS
  - Fixed-task priority scheduling (FTP)
- FTP: 1 task=1 priority
- Schedulability
  - Check if the (potentially infinite) set of jobs (job = instance of a task) will always be executed within its deadline

Response-Time Analysis

- A classic analysis for FTP scheduling
- RT = worst-case delay between release of a task and its termination
- Relative deadline = feasible window size from release to deadline
- For implicit deadline tasks, relative deadline = period
- But relative deadline can be lower or greater than period in general
- The system is schedulable iff WCRT of every task is ≤ relative deadline

Tractability

- For FTP, independent tasks, can be released simultaneously
  - Computing the WCRT is Weakly NP-hard (pseudo-polynomial algorithms)
  - Note: independent systems do not exist in real life
- Any other case
  - Strongly NP-hard
- Exact answer is intractable
- Conservative (sufficient) conditions used
  - Polynomial & pseudo-polynomial

Optimality

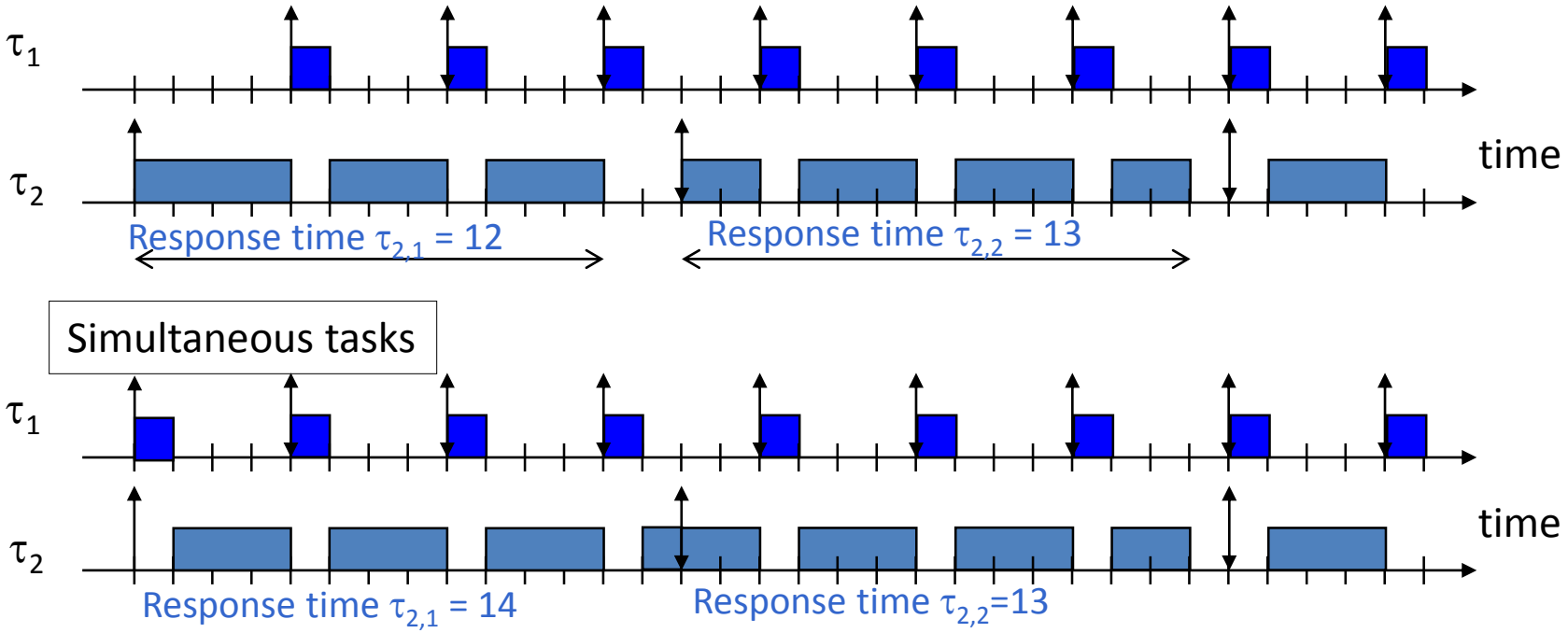
- Optimal algorithm A
  - Either A does it, or there is no algorithm doing it
- Optimality in algorithm classes
  - Either A does it, or no algorithm of this class does it
- RM is optimal in the class of FTP for independent implicit deadline tasks, if they can be released simultaneously

Sustainability

- Execution time varies from BCET to WCET
- If tasks are sporadic, period varies
- A test is sustainable regarding a parameter if « improving » this parameter cannot increase a response time

FTP schedulability analysis

Critical instant theorem

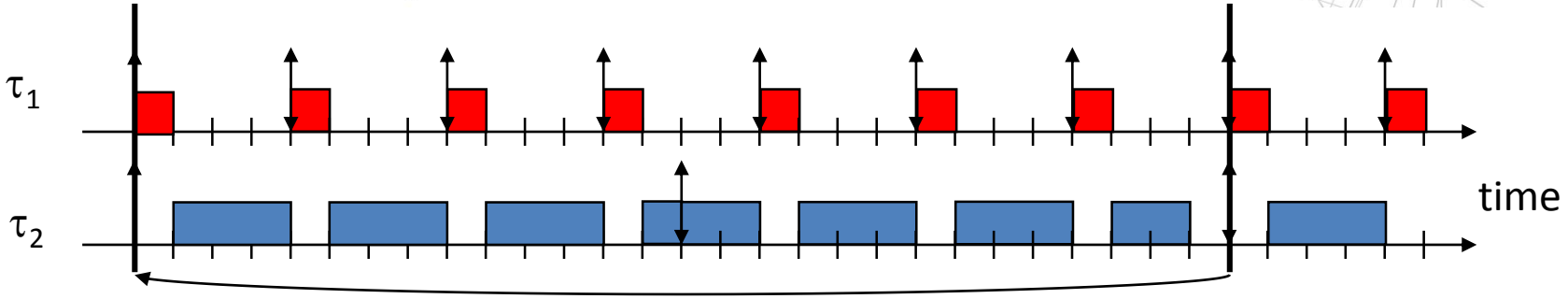


- The WCRT of a task of priority *i* scheduled by FTP occurs in the longest *i*-level busy period

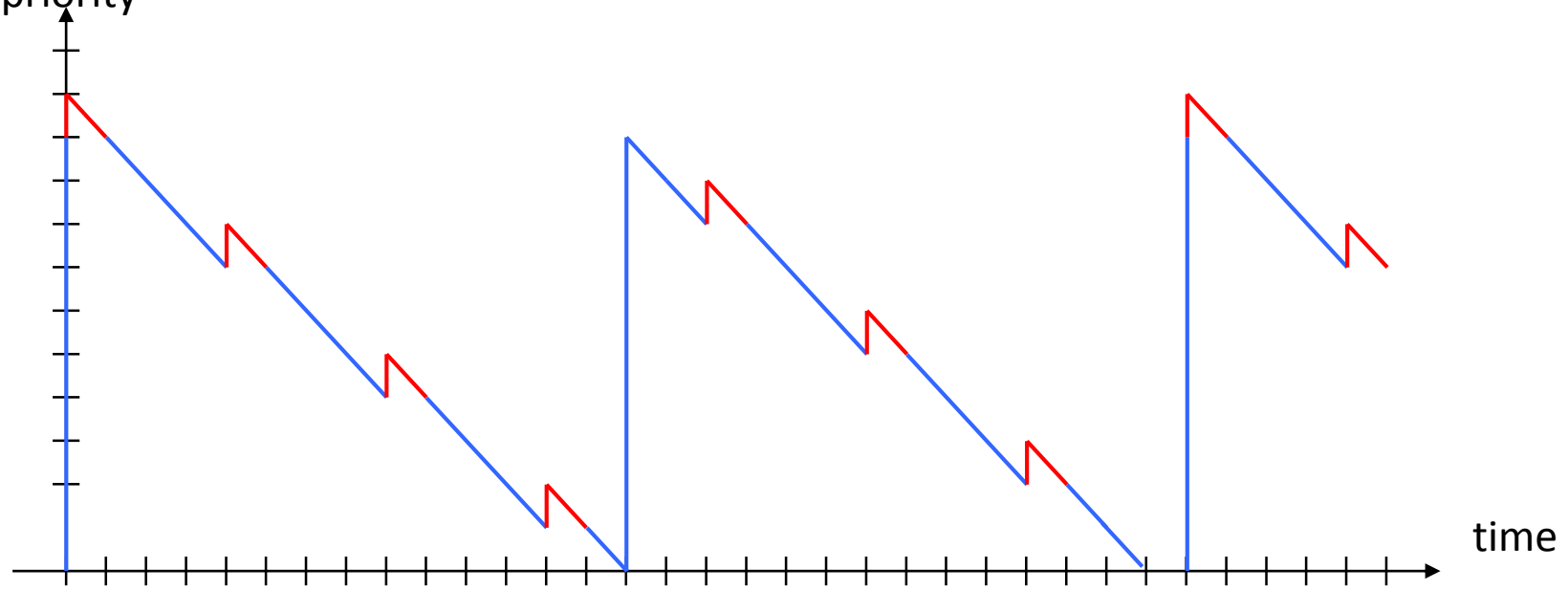
Critical instant

- The longest busy period starts at the critical instant
- The critical instant is initiated by the simultaneous release of every task of priority $\geq i$

Level-*i* Busy period

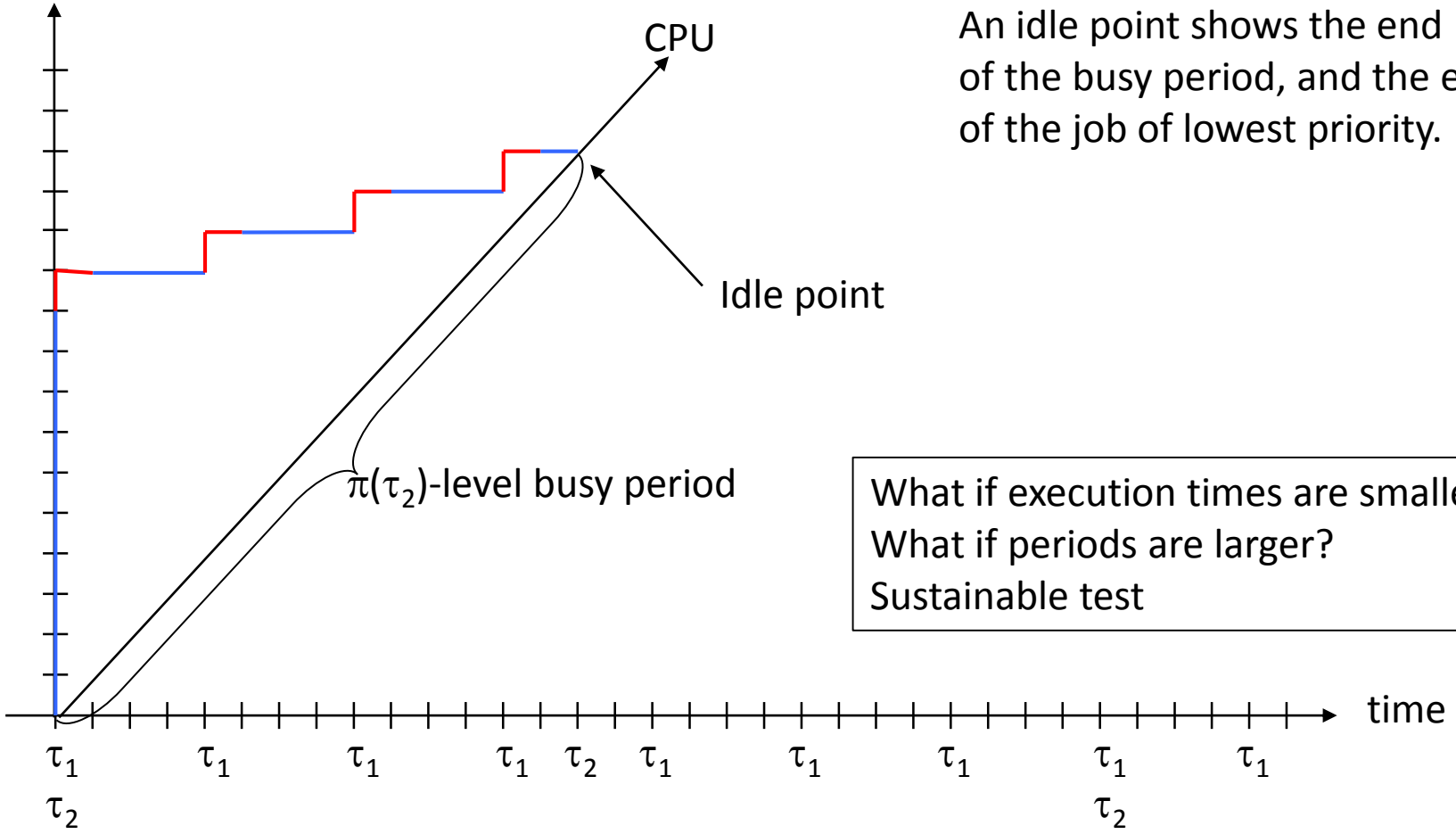


Remaining work of priority $\geq \pi(\tau_2)$



Classic view: RBF

Request bound function (RBF)

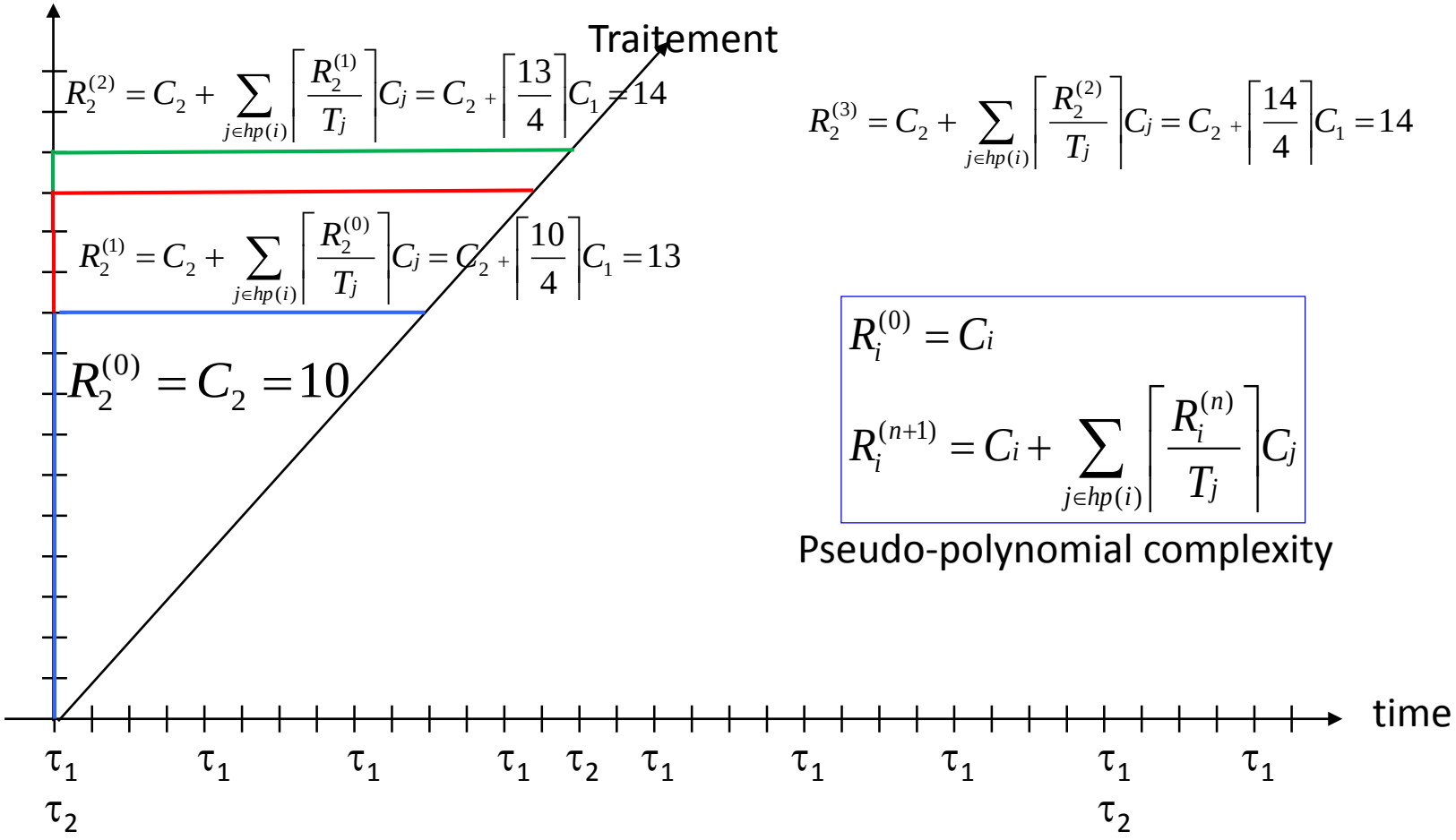


An idle point shows the end of the busy period, and the end of the job of lowest priority.

What if execution times are smaller?
What if periods are larger?
Sustainable test

Efficient computation

Request Bound Function (RBF)



$$R_2^{(3)} = C_2 + \sum_{j \in hp(i)} \left\lceil \frac{R_2^{(2)}}{T_j} \right\rceil C_j = C_2 + \left\lceil \frac{14}{4} \right\rceil C_1 = 14$$

$$R_i^{(0)} = C_i$$

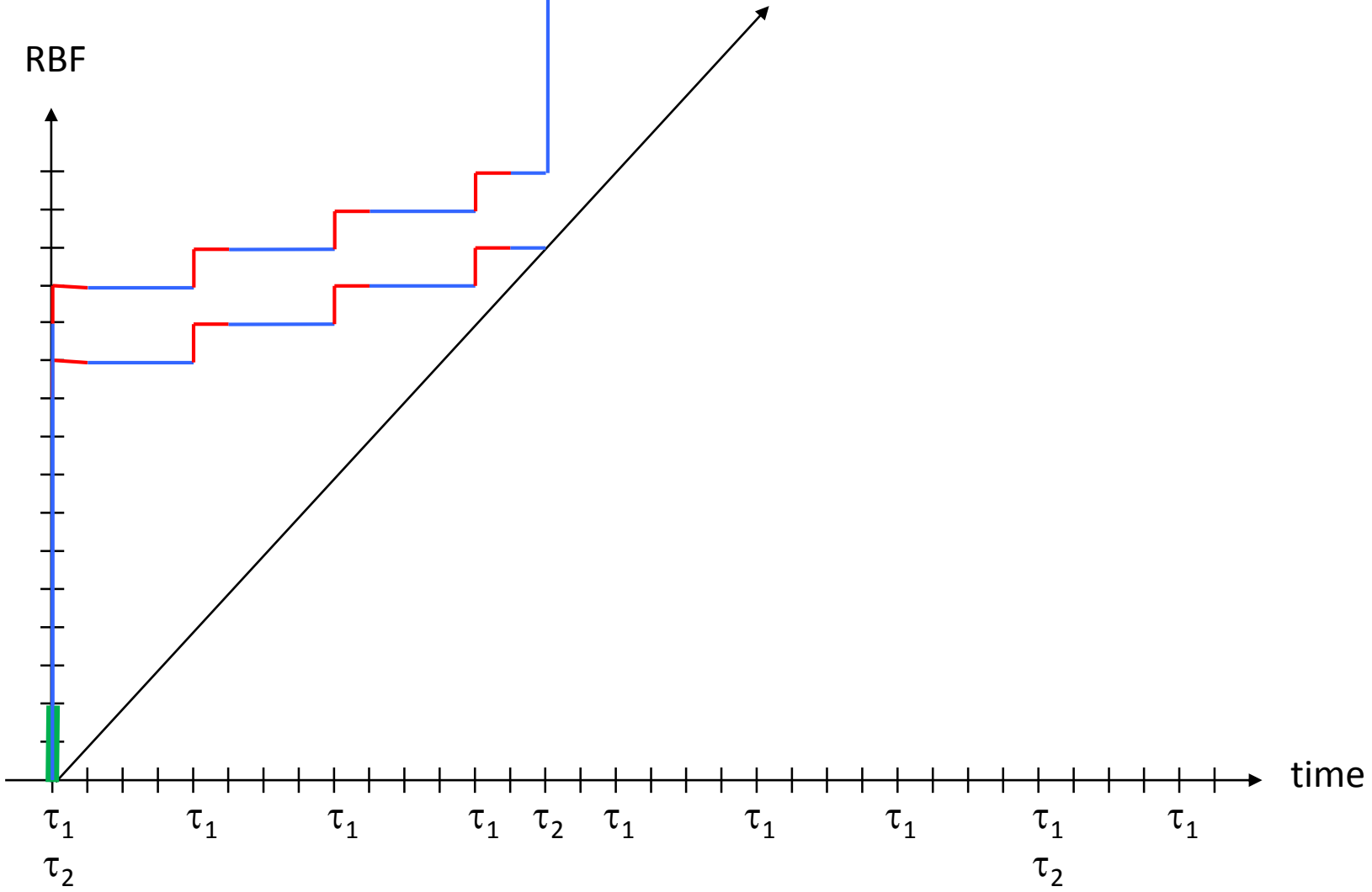
$$R_i^{(n+1)} = C_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i^{(n)}}{T_j} \right\rceil C_j$$

Pseudo-polynomial complexity

Variations for critical sections

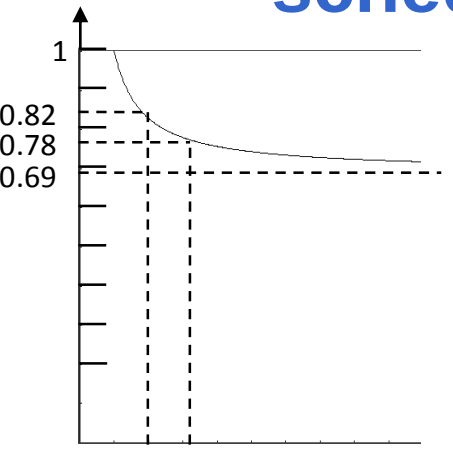
- Critical sections
  - IPCP: lower priority tasks can block a higher priority task only once in a busy period, and only for one critical section

RTA variation

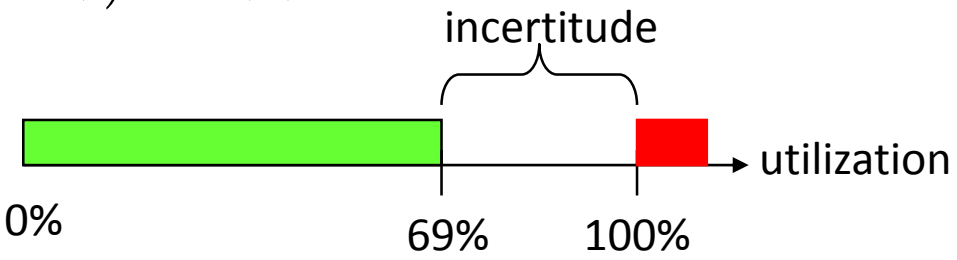


Specific polynomial tests

- Classic Liu&Layland schedulability test
  - Independent, implicit deadline tasks
  - FTP scheduling such that the smaller period, the higher priority (Rate Monotonic)
  - **$U \leq n\,(2^{1/n} - 1)$ is a conservative schedulability test**



$$\lim_{n\to\infty}\left(n\times(2^{1/n}-1)\right)=\ln(2)\approx 0.69$$



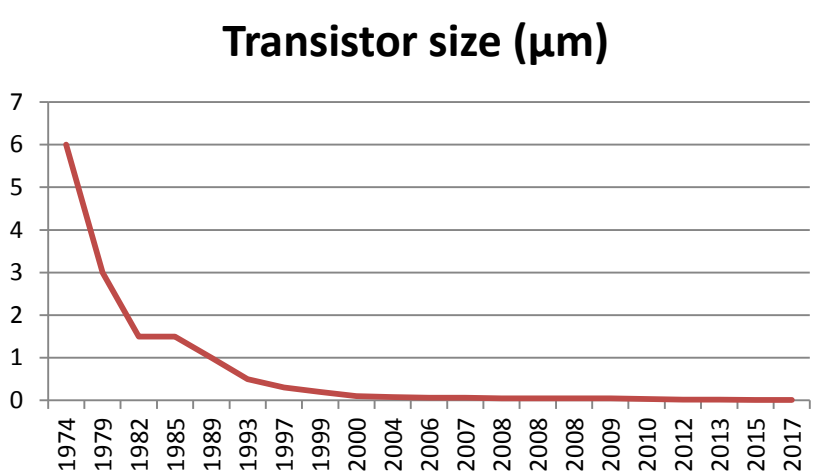
Hyperbolic test

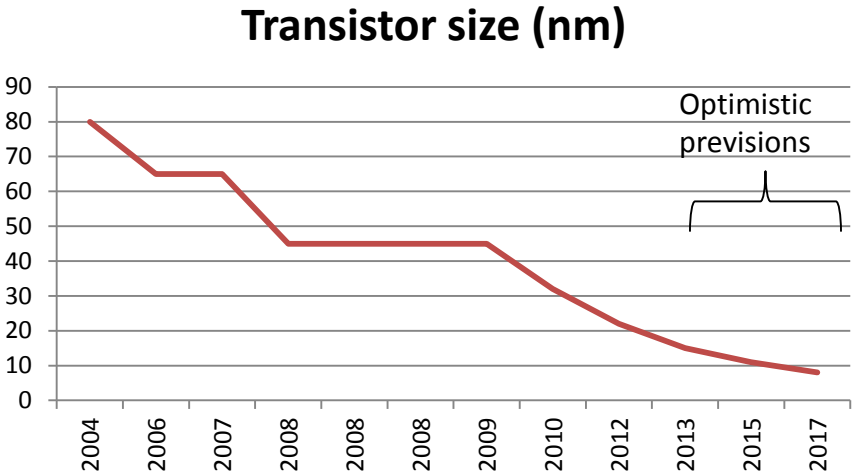
- L&L test 1973
- Hyperbolic test 2003

$U_i = C_i / T_i$ $\qquad$ $\prod_{i \in 1..n} (U_i + 1) \leq 2$

- Strictly better than L&L test
- Same restrictive context

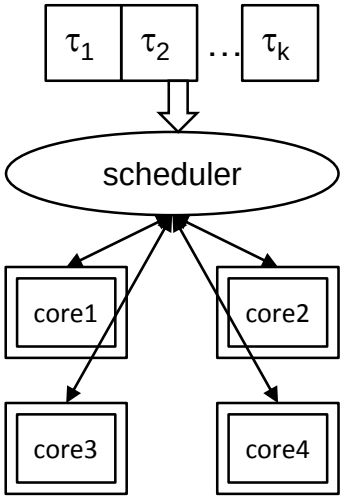
Multi/Many core



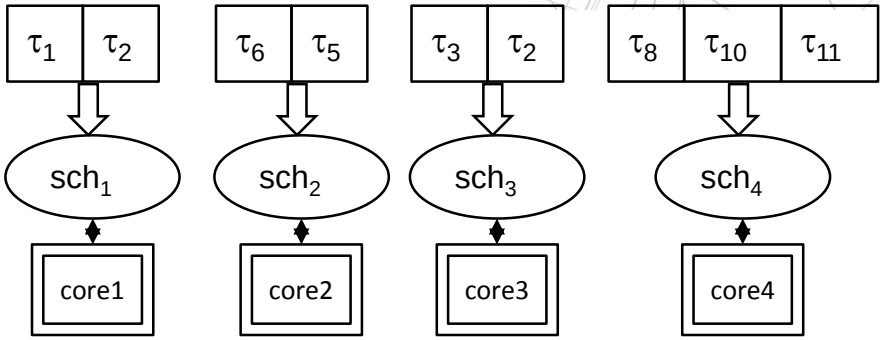


- Increasing frequency
  - Reduce transistor size
  - Or for same transistor size => consumed power square function of the gain in fequency
  - 1 linear nm ≈ 5 silicium atoms
    - ✓Quantic mechanic effects, leakage currents

Global vs. partitioned



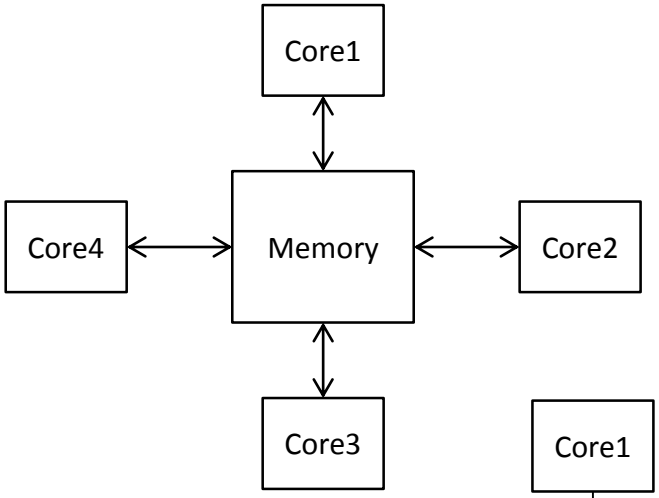
Global



Partitioned

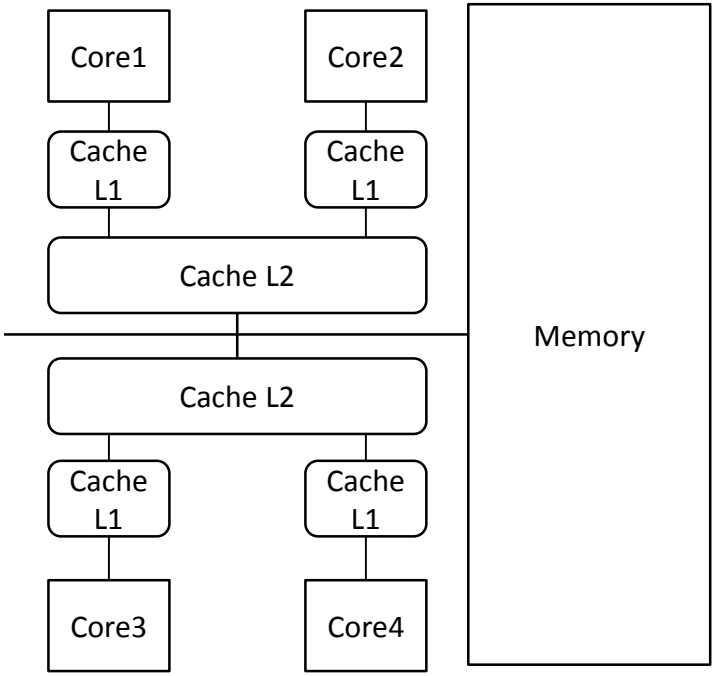
- Semi-partitioned
  - Some tasks can migrate
- Clustered
  - Several global schedulers

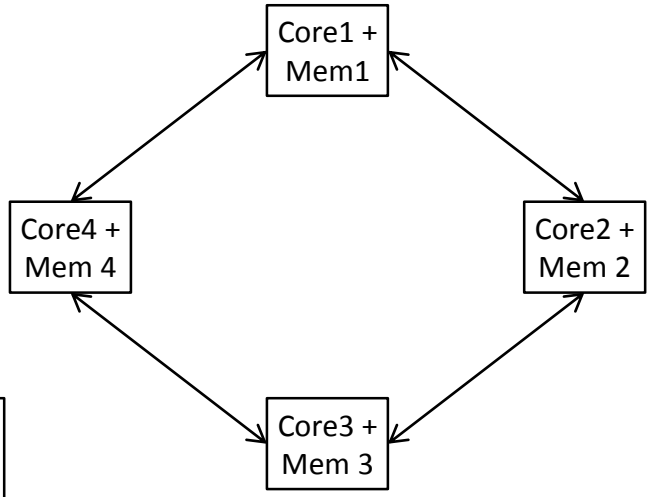
Simplified hypothesis



Scheduling

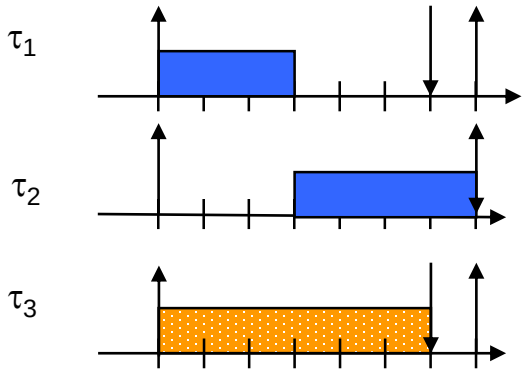
Timing analysis

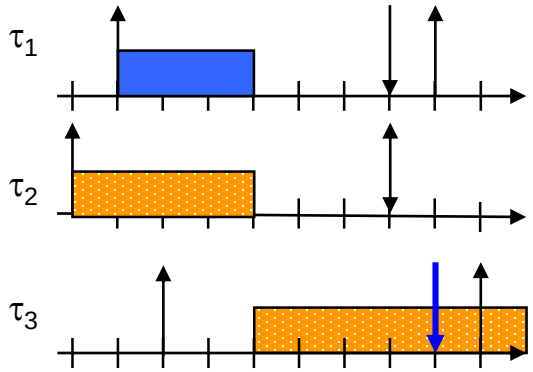




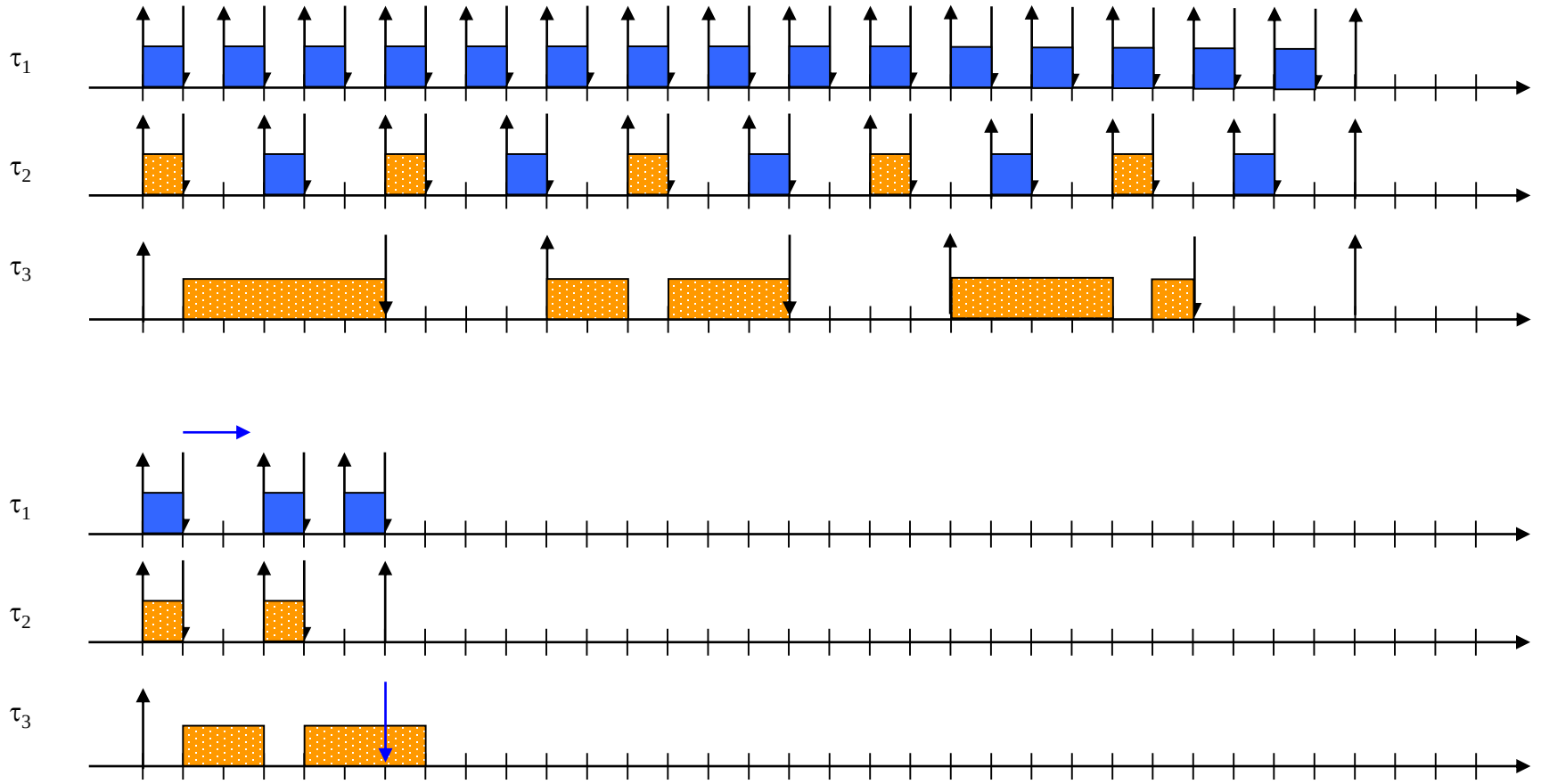
NoC

Critical instant does not apply



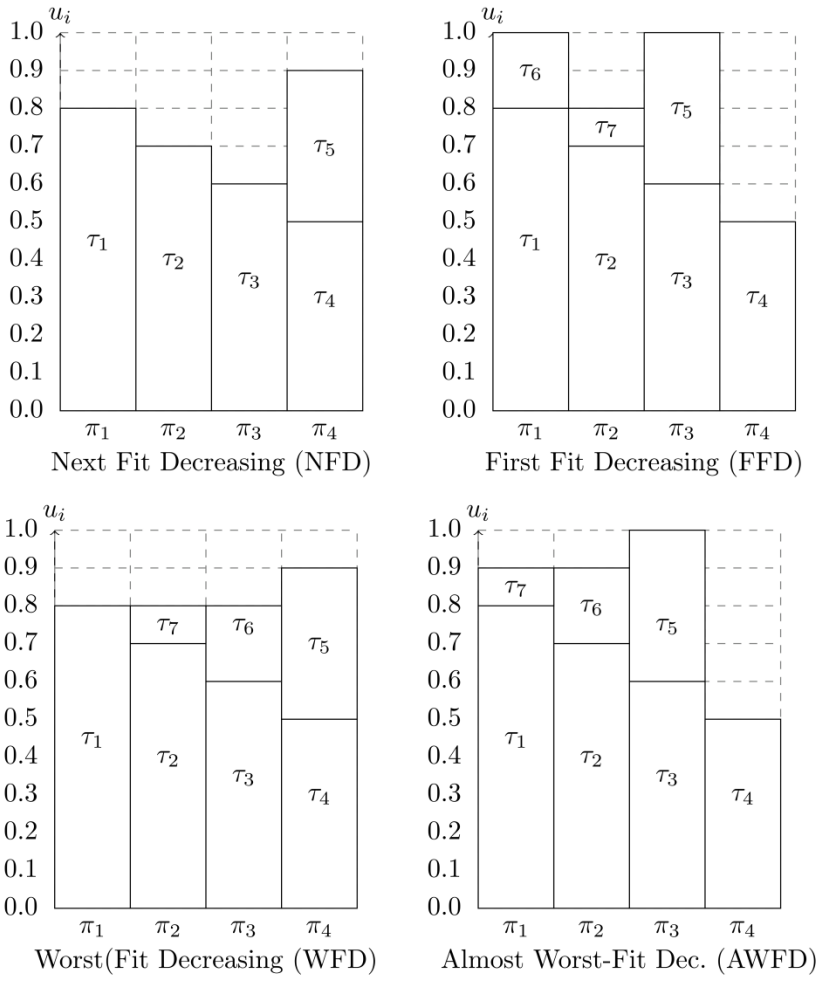


T-Sustainability does not apply



Partitioned scheduling

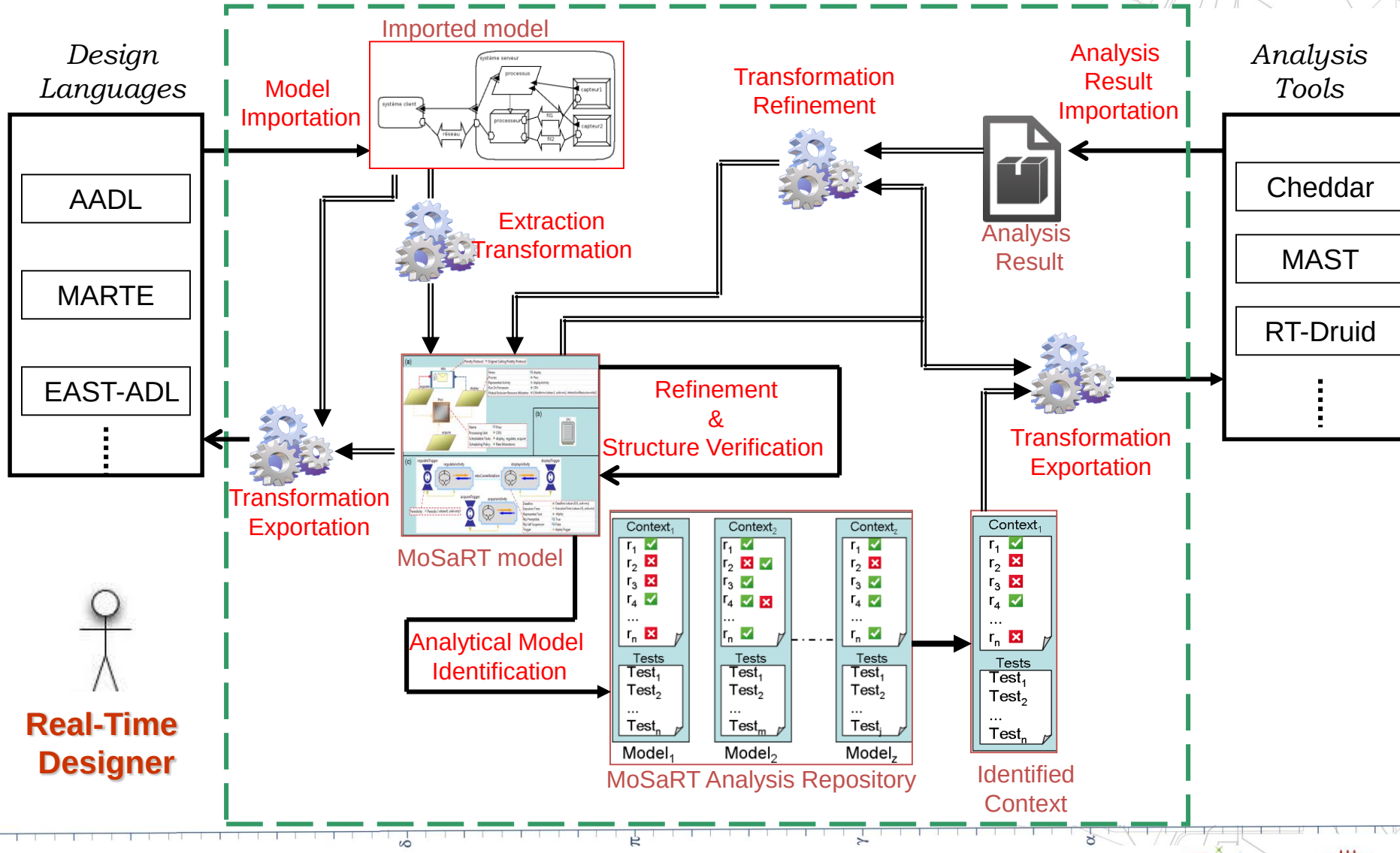
- Knapsack problem
  - NP-complete



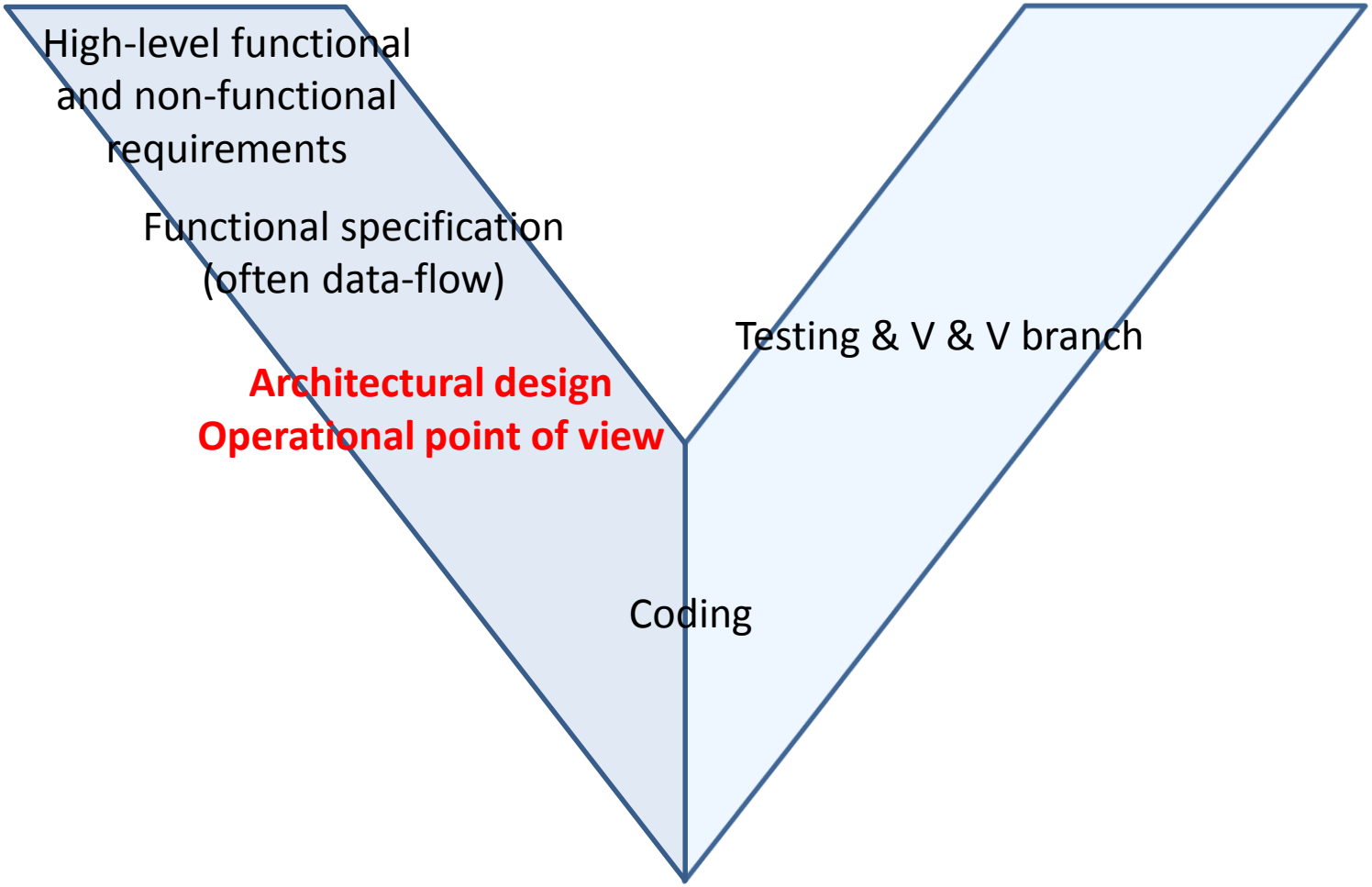
Global scheduling

- Fair algorithms based on fluid scheduling are optimal
  - Generate a lot of preemptions
  - Generate a lot of migrations
- Not so interesting in practice
- Classic scheduling algorithms can behave awfully
  - Dhall effect: global-EDF can be as bad as not be able to schedule systems with $U=1+\varepsilon$ on an infinite number of processors

Design & sched. analysis



Life-cycle Holy Grail



Questions?