M2 ATIAM – IM

Graphical reactive programming

Esterel, SCADE, Pure Data

Philippe Esling
esling@ircam.fr

First thing's first

Today we will see reactive programming languages

- Theory
- ESTEREL
- SCADE
- Pure Data

However our goal is to see how to apply this to **musical data** (audio) and **real-time constraints**

Hence all exercises are done with **Pure Data**

- Pure data is **open source, free** and **cross-platform**
- Works as **Max/Msp** (have almost all same features)
- Can easily be embedded on UDOO cards (special distribution)

So go now to **download it** and install it right away

https://puredata.info/downloads/pure-data

Reactive programming: Basic notions

Goals and aims

Here we aim towards creating **critical, reactive, embedded** software.

Reactive programming: Basic notions

Goals and aims

Here we aim towards creating **critical, reactive, embedded** software.

The mandatory notions that we should get first are

- **Embedded**
- **Critical**
- **Safety**
- **Reactive**
- **Determinism**

Reactive programming: Basic notions

Goals and aims

Here we aim towards creating **critical, reactive, embedded** software.

The mandatory notions that we should get first are

?
- **Embedded**
- **Critical**
- **Safety**
- **Reactive**
- **Determinism**

Reactive programming: Basic notions

Goals and aims

Here we aim towards creating **critical, reactive, embedded** software.

The mandatory notions that we should get first are

- **Embedded**
- **Critical**
- **Safety**
- **Reactive**
- **Determinism**

An embedded system is a computer system with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints.

Reactive programming: Basic notions

Goals and aims

Here we aim towards creating **critical, reactive, embedded** software.

The mandatory notions that we should get first are

- **Embedded**
- **Critical**
- **Safety**
- **Reactive**
- **Determinism**

An embedded system is a computer system with a dedicated function within a larger mechanical or electrical system, often with real-time computing constraints.

Examples

Reactive programming: Basic notions

Goals and aims

Here we aim towards creating **critical, reactive, embedded** software.

The mandatory notions that we should get first are

- **Embedded**
- **Critical**
- **Safety**
- **Reactive**
- **Determinism**

?

Reactive programming: Basic notions

Goals and aims

Here we aim towards creating **critical, reactive, embedded** software.

The mandatory notions that we should get first are

- **Embedded**
- **Critical**
- **Safety**
- **Reactive**
- **Determinism**

Intuitively, a **critical system** is a system in which **failure can have severe impacts**

- Nuclear
- Aeronautic
- Automotive
- Railway
- Space
- Medical

Reactive programming: Basic notions

Goals and aims

Here we aim towards creating **critical, reactive, embedded** software.

The mandatory notions that we should get first are

- **Embedded**
- **Critical**
- **Safety**
- **Reactive**
- **Determinism**

Intuitively, a **critical system** is a system in which **failure can have severe impacts**

Standards define **software criticality levels**

- DO-178C for airborne systems
- EN-50126/EN-50128, for railway applications
- IEC-61508, applied in the industry
- ISO-26262, for road vehicles

DO-178C Criticality Levels (airplanes)

Severity	Consequence
Catastrophic	Failure conditions which would prevent continued safe flight and landing
Hazardous / Severe-Major	Failure conditions which would reduce the capability of the aircraft or the ability of the crew to cope with adverse operating conditions to the extent that there would be: • a large reduction in safety margins or functional capabilities, • physical distress or higher workload such that the flight crew could not be relied on to perform their tasks accurately or completely, or • adverse effects on occupants including serious or potentially fatal injuries to a small number of those occupants
Major	Failure conditions which would reduce the capability of the aircraft or the ability of the crew to cope with adverse operating conditions to the extent that there would be, for example, a significant reduction in safety margins or functional capabilities, a significant increase in crew workload or in conditions impairing crew efficiency, or discomfort to occupants, possibly including injuries
Minor	Failure conditions which would not significantly reduce aircraft safety, and which would involve crew actions that are well within their capabilities. Minor failure conditions may include, for example, a slight reduction in safety margins or functional capabilities, a slight
No effect	Failure conditions which do not affect the operational capability of the aircraft or increase crew workload

Reactive programming: Basic notions

Goals and aims

Here we aim towards creating **critical, reactive, embedded** software.

The mandatory notions that we should get first are

- **Embedded**
- **Critical**
- ? **Safety**
- **Reactive**
- **Determinism**

Reactive programming: Basic notions

Goals and aims

Here we aim towards creating **critical, reactive, embedded** software.

The mandatory notions that we should get first are

- **Embedded**
- **Critical**
- **Safety**
- **Reactive**
- **Determinism**

(Usual definitions)

Safety

Safety is the state of being "safe", the condition of being protected against […] consequences of failure, damage, error, accidents, harm or any other event which could be considered non-desirable. It can include protection of people or possessions.

Security

Security is the degree of resistance to, or protection from, harm. It applies to any vulnerable and valuable asset […].

Reactive programming: Basic notions

Goals and aims

Here we aim towards creating **critical, reactive, embedded** software.

The mandatory notions that we should get first are

- **Embedded**
- **Critical**
- **Safety**
- **Reactive**
- **Determinism**

(Software engineering definitions)

Safety

The software must not harm the world.

Security

The world must not harm the software.

Reactive programming ?

Transformational systems

- Inputs available on execution start
- Outputs delivered on execution end

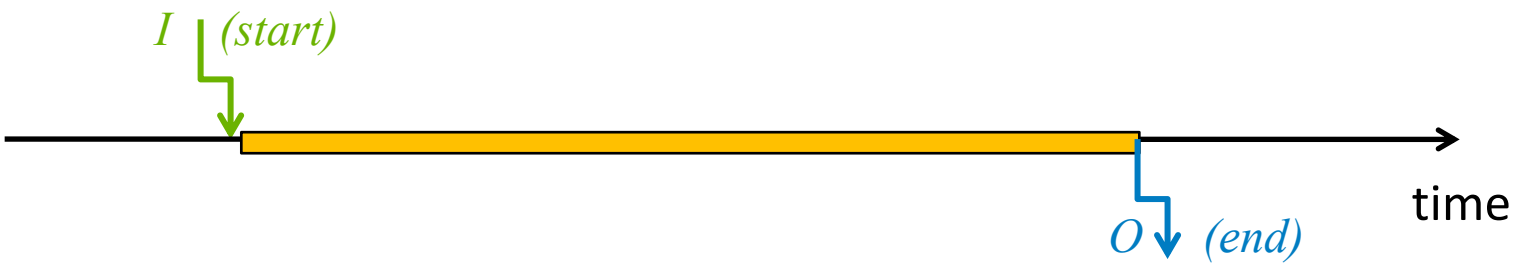
e.g. Mathematical computation

Reactive programming ?

Transformational systems

- Inputs available on execution start
- Outputs delivered on execution end

e.g. Mathematical computation



Reactive programming ?

Transformational systems

- Inputs available on execution start
- Outputs delivered on execution end

e.g. Mathematical computation

Interactive systems

- Interact with the environment
- Have subjective speed requirements

e.g. Unix, Powerpoint

Reactive programming ?

Transformational systems

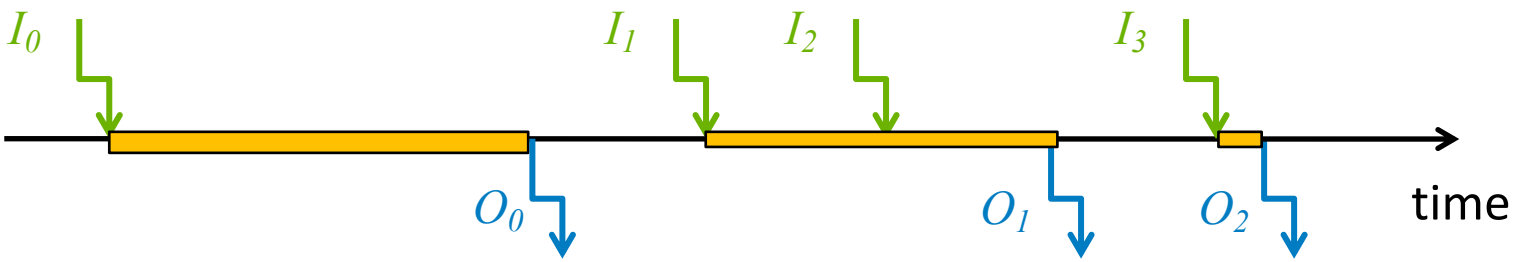
- Inputs available on execution start
- Outputs delivered on execution end

e.g. Mathematical computation

Interactive systems

- Interact with the environment
- Have subjective speed requirements

e.g. Unix, Powerpoint



Reactive programming ?

Transformational systems

- Inputs available on execution start
- Outputs delivered on execution end

e.g. Mathematical computation

Interactive systems

- Interact with the environment
- Have subjective speed requirements

e.g. Unix, Powerpoint

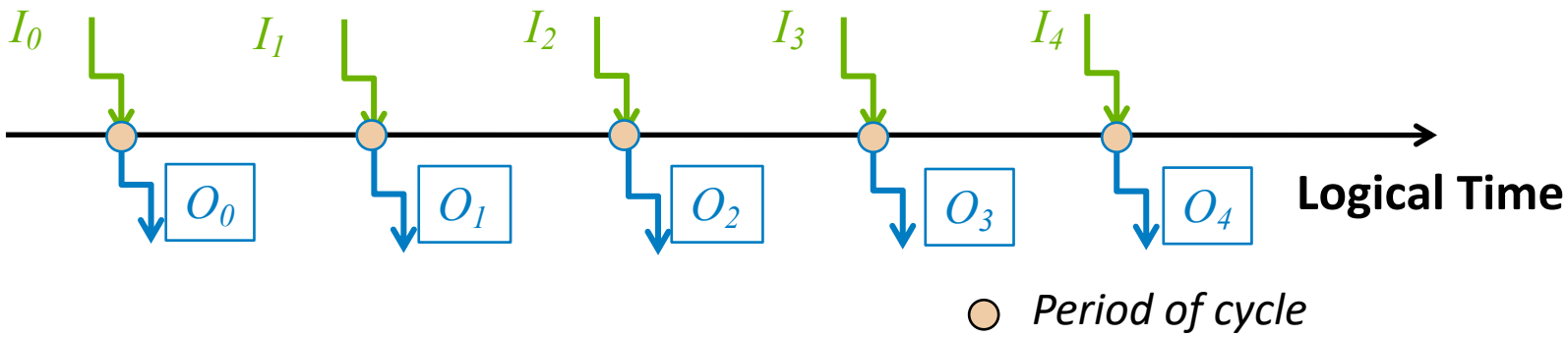
Reactive systems

- Interact with the environment
- Have objective speed requirements

e.g. Control /Command of a spacecraft

Reactive system time constraints

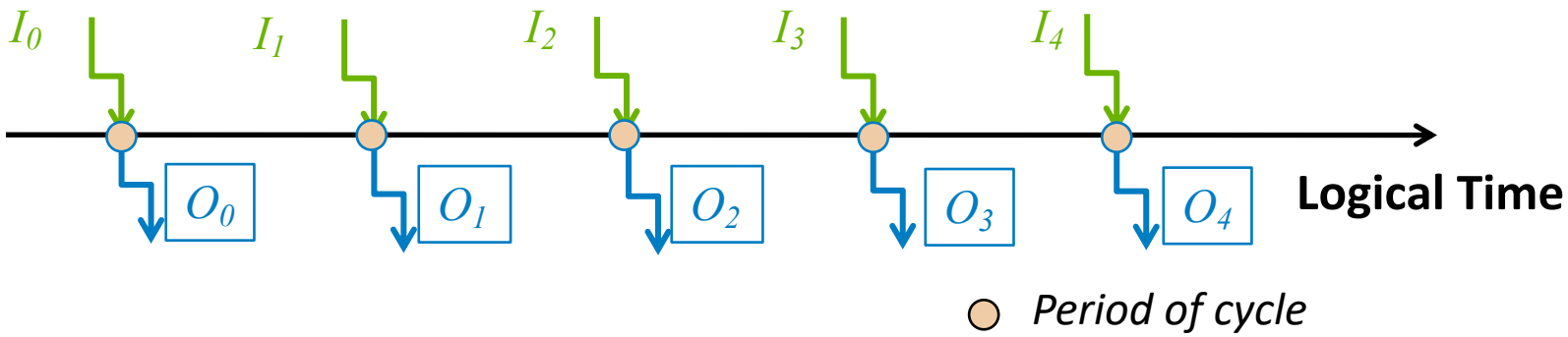
Time as a logical notion



Reactive system time constraints

Time as a logical notion

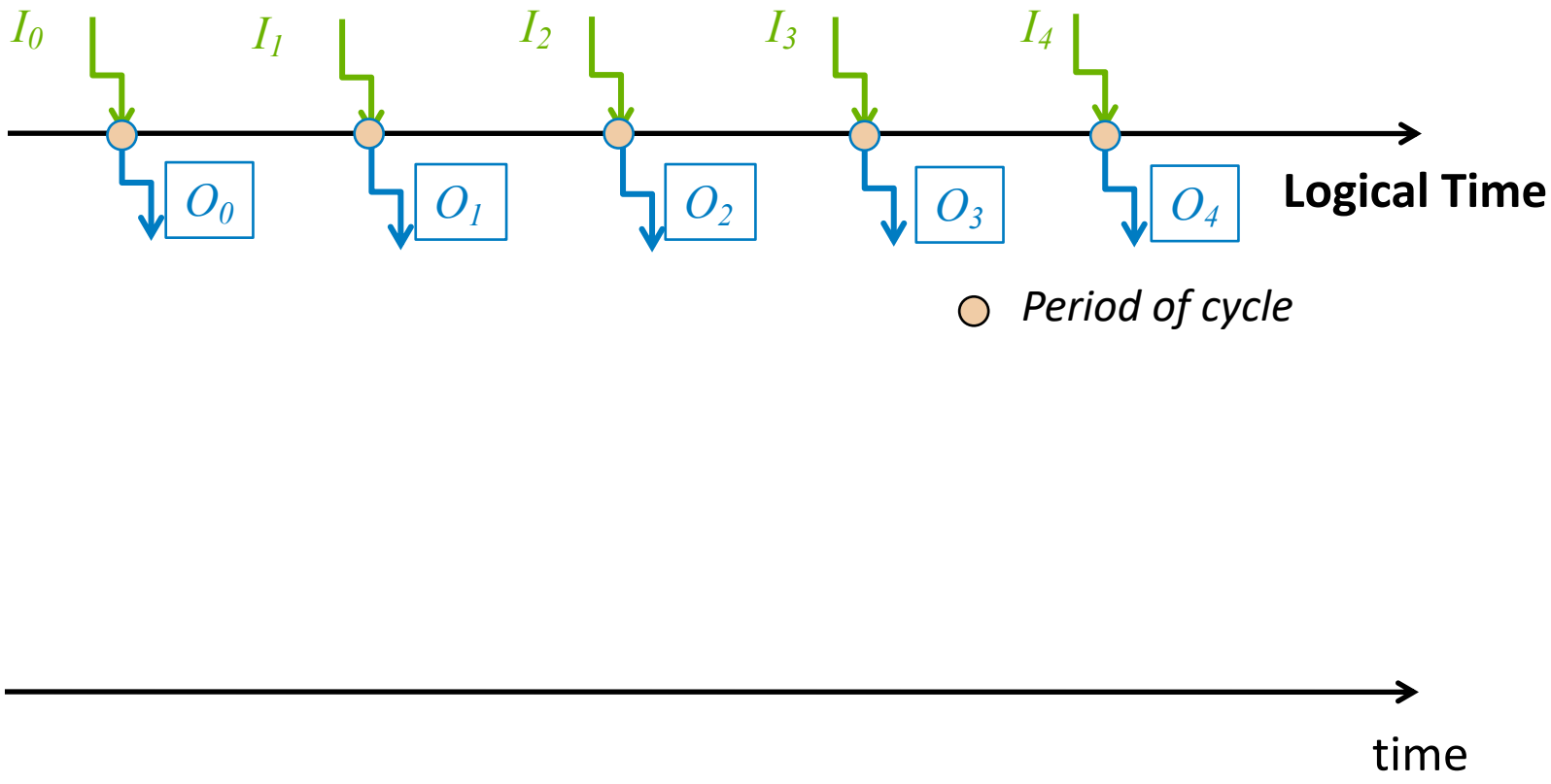
In theory, execution time is null but how to assume this hypothesis?



Reactive system time constraints

Time as a logical notion

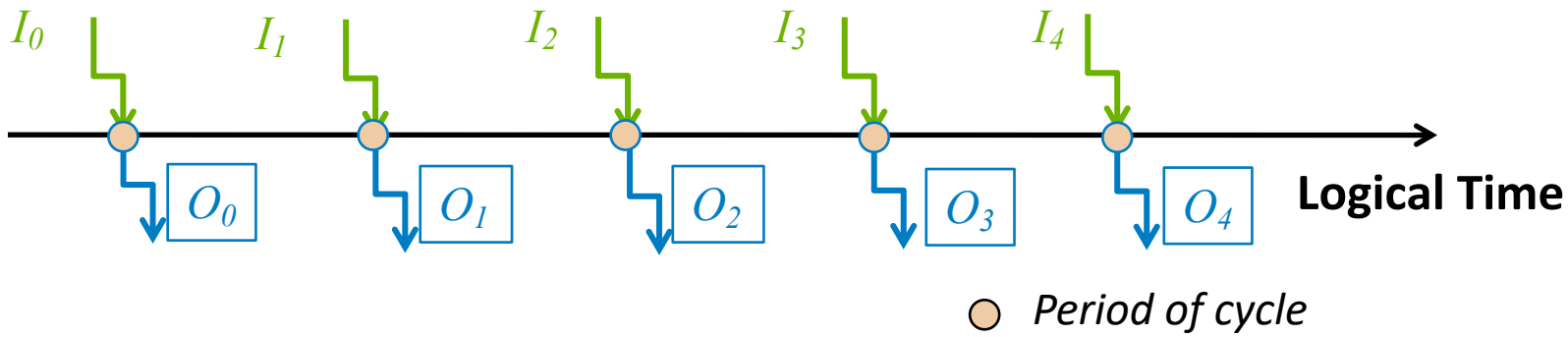
In theory, execution time is null but how to assume this hypothesis?



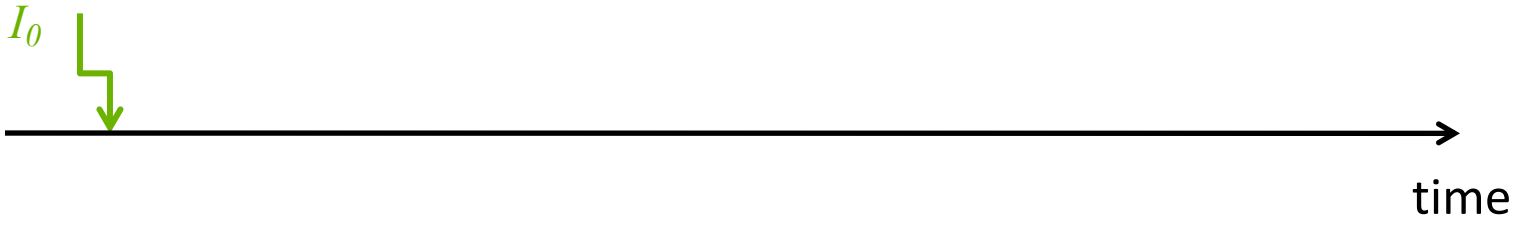
Reactive system time constraints

Time as a logical notion

In theory, execution time is null but how to assume this hypothesis?



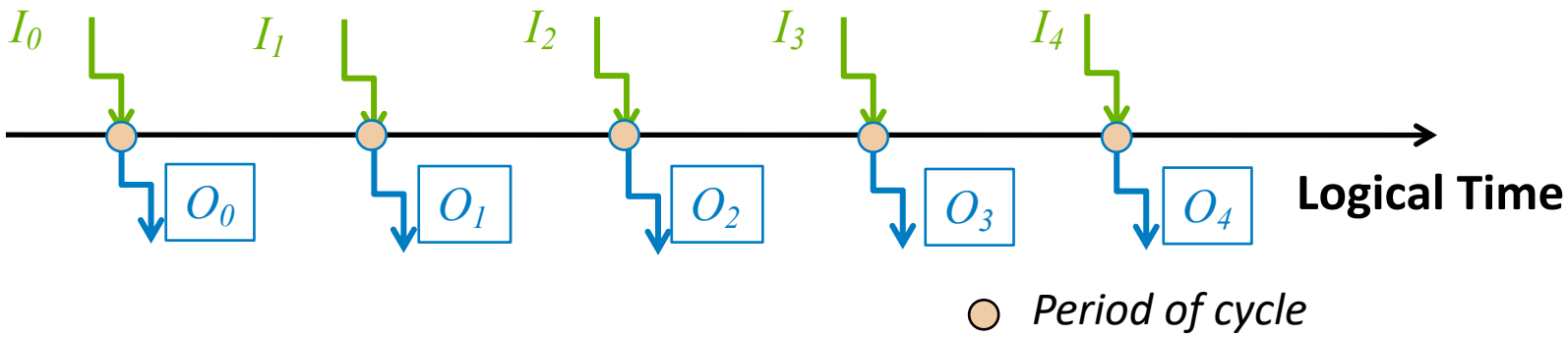
Receive inputs



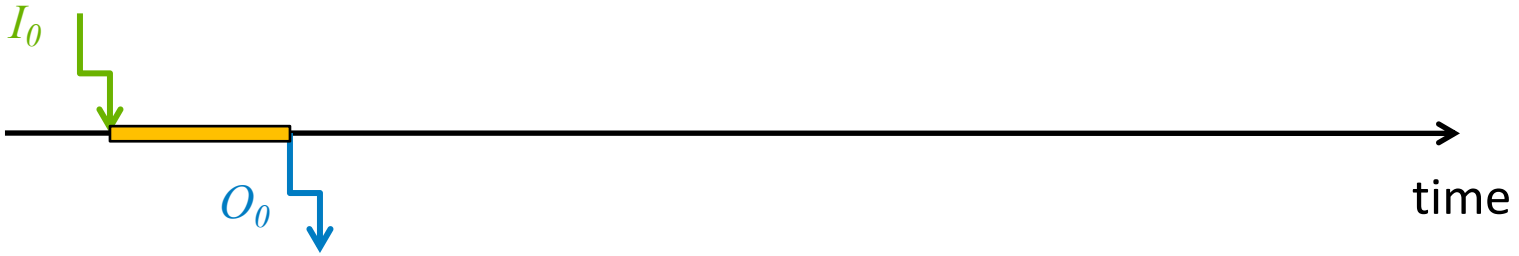
Reactive system time constraints

Time as a logical notion

In theory, execution time is null but how to assume this hypothesis?



Receive inputs



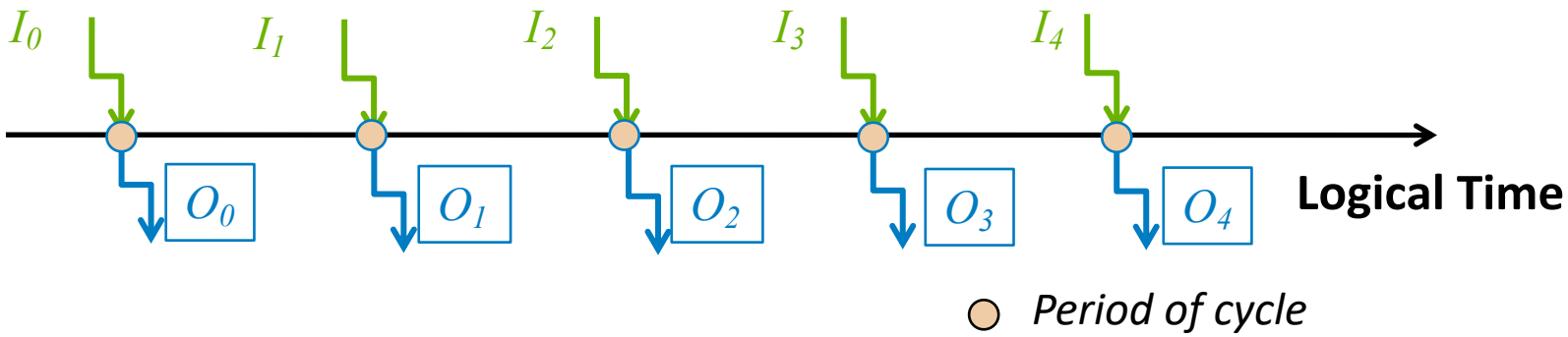
Provide outputs

Time to compute the output

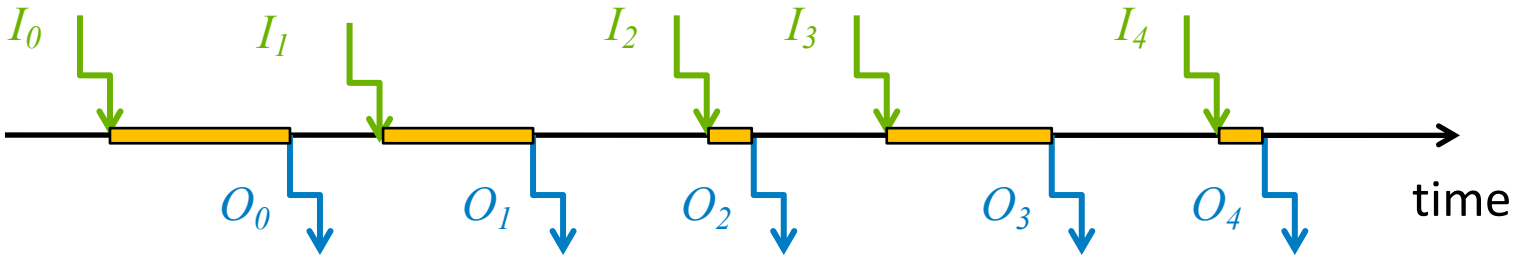
Reactive system time constraints

Time as a logical notion

In theory, execution time is null but how to assume this hypothesis?



Receive inputs



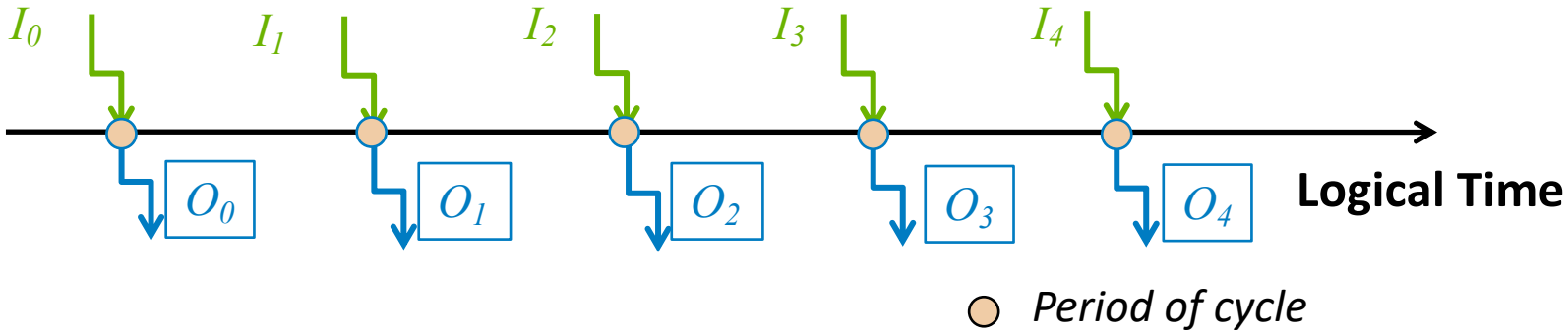
Provide outputs

Time to compute the output

Reactive system time constraints

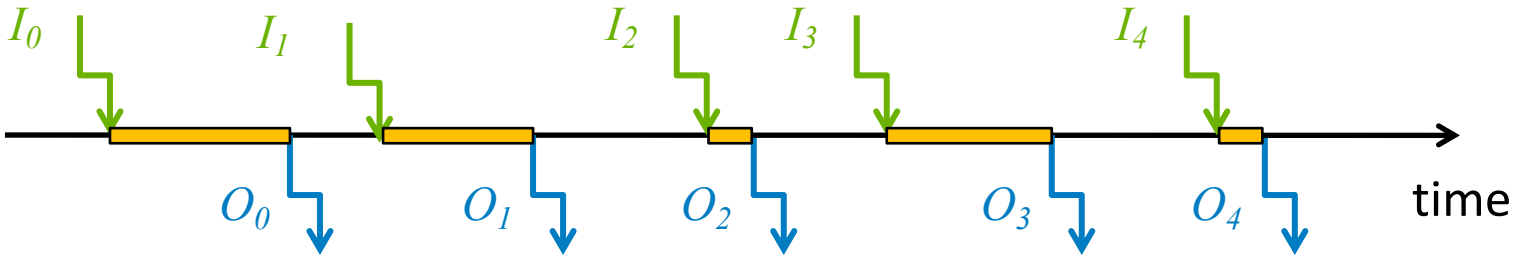
Time as a logical notion

In theory, execution time is null but how to assume this hypothesis?



Receive inputs

The outputs shall be produced before the reception of the next input



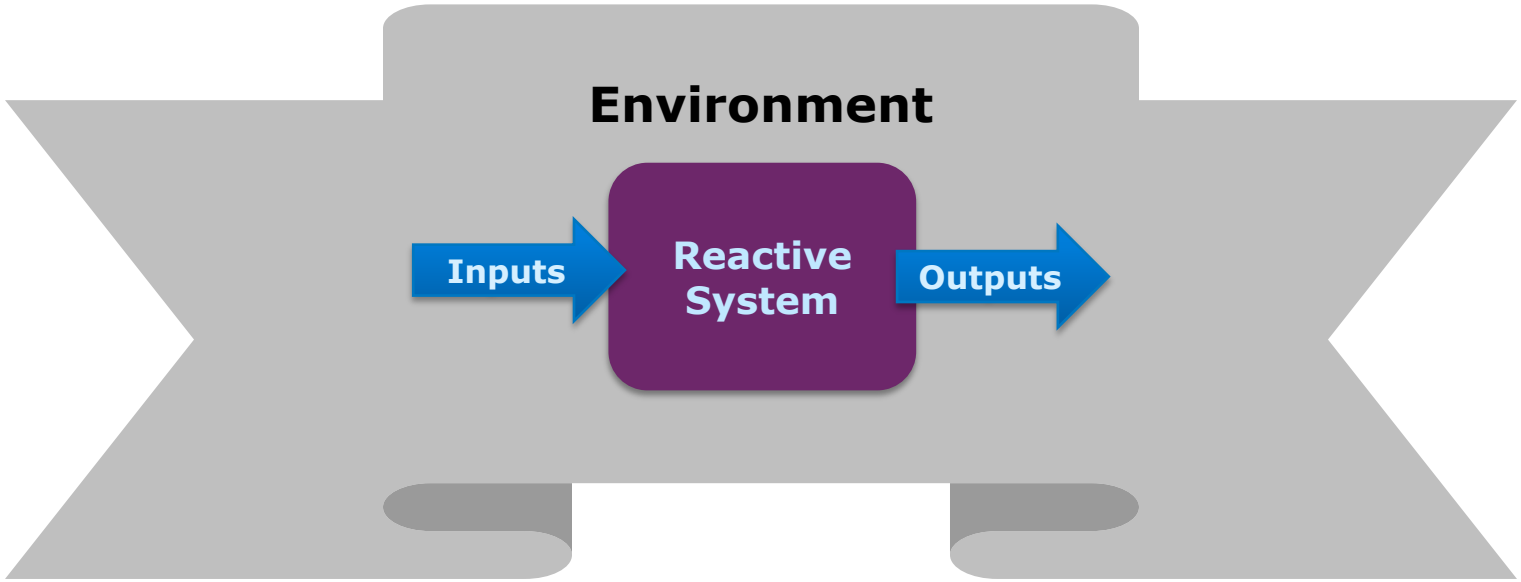
Provide outputs

Time to compute the output

Reactive systems

Interact with their environment

- Receive inputs
- Provide outputs



Determinism

The same cause implies the same effect.

Not a strong requirement for non-critical systems:

- Characteristic of interactive systems (e.g. OS, Internet)

Key requirements for safety-embedded systems:

- The same sequence of inputs always produces the same sequence of outputs
- This implies a reproducible behavior

Formal methods concept

Aim: Add mathematical reasoning in the system development flow.

Goal: Specify better, reduce verification costs, increase system reliability.

How: Provide concise formalism for specification and implementation.

Challenge: Enable smooth integration in engineering framework.

Formal methods concept

Aim: Add mathematical reasoning in the system development flow.

Goal: Specify better, reduce verification costs, increase system reliability.

How: Provide concise formalism for specification and implementation.

Challenge: Enable smooth integration in engineering framework.

Better specification reduces verification!

Formal methods concept

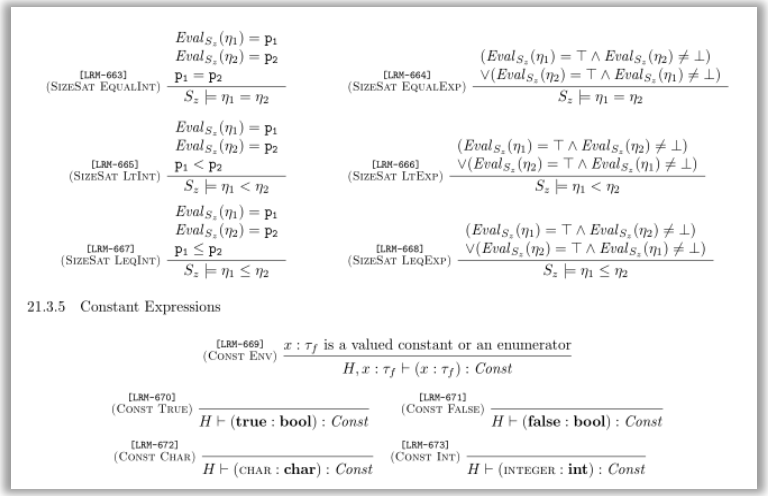
Aim: Add mathematical reasoning in the system development flow.

Goal: Specify better, reduce verification costs, increase system reliability.

How: Provide concise formalism for specification and implementation.

Challenge: Enable smooth integration in engineering framework.

Better specification reduces verification!



$$\text{[LRM-663] (SizeSat EqualInt)}\ \frac{Eval_{S_z}(\eta_1) = \mathrm{p}_1 \quad Eval_{S_z}(\eta_2) = \mathrm{p}_2 \quad \mathrm{p}_1 = \mathrm{p}_2}{S_z \models \eta_1 = \eta_2}$$

$$\text{[LRM-664] (SizeSat EqualExp)}\ \frac{(Eval_{S_z}(\eta_1) = \top \wedge Eval_{S_z}(\eta_2) \neq \bot) \vee (Eval_{S_z}(\eta_2) = \top \wedge Eval_{S_z}(\eta_1) \neq \bot)}{S_z \models \eta_1 = \eta_2}$$

$$\text{[LRM-665] (SizeSat LtInt)}\ \frac{Eval_{S_z}(\eta_1) = \mathrm{p}_1 \quad Eval_{S_z}(\eta_2) = \mathrm{p}_2 \quad \mathrm{p}_1 < \mathrm{p}_2}{S_z \models \eta_1 < \eta_2}$$

$$\text{[LRM-666] (SizeSat LtExp)}\ \frac{(Eval_{S_z}(\eta_1) = \top \wedge Eval_{S_z}(\eta_2) \neq \bot) \vee (Eval_{S_z}(\eta_2) = \top \wedge Eval_{S_z}(\eta_1) \neq \bot)}{S_z \models \eta_1 < \eta_2}$$

$$\text{[LRM-667] (SizeSat LeqInt)}\ \frac{Eval_{S_z}(\eta_1) = \mathrm{p}_1 \quad Eval_{S_z}(\eta_2) = \mathrm{p}_2 \quad \mathrm{p}_1 \leq \mathrm{p}_2}{S_z \models \eta_1 \leq \eta_2}$$

$$\text{[LRM-668] (SizeSat LeqExp)}\ \frac{(Eval_{S_z}(\eta_1) = \top \wedge Eval_{S_z}(\eta_2) \neq \bot) \vee (Eval_{S_z}(\eta_2) = \top \wedge Eval_{S_z}(\eta_1) \neq \bot)}{S_z \models \eta_1 \leq \eta_2}$$

21.3.5 Constant Expressions

$$\text{[LRM-669] (Const Env)}\ \frac{x : \tau_f \text{ is a valued constant or an enumerator}}{H, x : \tau_f \vdash (x : \tau_f) : \mathit{Const}}$$

$$\text{[LRM-670] (Const True)}\ \frac{}{H \vdash (\mathbf{true} : \mathbf{bool}) : \mathit{Const}} \qquad \text{[LRM-671] (Const False)}\ \frac{}{H \vdash (\mathbf{false} : \mathbf{bool}) : \mathit{Const}}$$

$$\text{[LRM-672] (Const Char)}\ \frac{}{H \vdash (\text{CHAR} : \mathbf{char}) : \mathit{Const}} \qquad \text{[LRM-673] (Const Int)}\ \frac{}{H \vdash (\text{INTEGER} : \mathbf{int}) : \mathit{Const}}$$

Mathematical formal definition

Formal methods concept

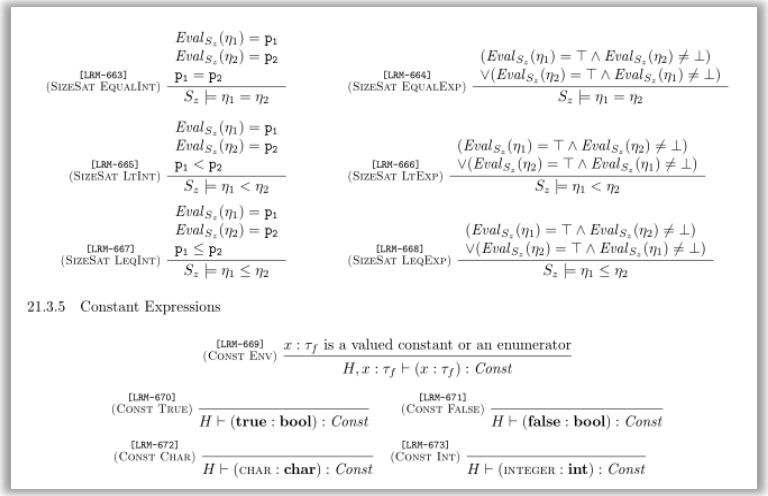
Aim: Add mathematical reasoning in the system development flow.

Goal: Specify better, reduce verification costs, increase system reliability.

How: Provide concise formalism for specification and implementation.

Challenge: Enable smooth integration in engineering framework.

Better specification reduces verification!



$$\frac{Eval_{S_z}(\eta_1) = p_1 \quad Eval_{S_z}(\eta_2) = p_2 \quad p_1 = p_2}{S_z \models \eta_1 = \eta_2} \text{ [LRM-663] (SizeSat EqualInt)}$$

$$\frac{(Eval_{S_z}(\eta_1) = \top \wedge Eval_{S_z}(\eta_2) \neq \bot) \vee (Eval_{S_z}(\eta_2) = \top \wedge Eval_{S_z}(\eta_1) \neq \bot)}{S_z \models \eta_1 = \eta_2} \text{ [LRM-664] (SizeSat EqualExp)}$$

$$\frac{Eval_{S_z}(\eta_1) = p_1 \quad Eval_{S_z}(\eta_2) = p_2 \quad p_1 < p_2}{S_z \models \eta_1 < \eta_2} \text{ [LRM-665] (SizeSat LtInt)}$$

$$\frac{(Eval_{S_z}(\eta_1) = \top \wedge Eval_{S_z}(\eta_2) \neq \bot) \vee (Eval_{S_z}(\eta_2) = \top \wedge Eval_{S_z}(\eta_1) \neq \bot)}{S_z \models \eta_1 < \eta_2} \text{ [LRM-666] (SizeSat LtExp)}$$

$$\frac{Eval_{S_z}(\eta_1) = p_1 \quad Eval_{S_z}(\eta_2) = p_2 \quad p_1 \leq p_2}{S_z \models \eta_1 \leq \eta_2} \text{ [LRM-667] (SizeSat LeqInt)}$$

$$\frac{(Eval_{S_z}(\eta_1) = \top \wedge Eval_{S_z}(\eta_2) \neq \bot) \vee (Eval_{S_z}(\eta_2) = \top \wedge Eval_{S_z}(\eta_1) \neq \bot)}{S_z \models \eta_1 \leq \eta_2} \text{ [LRM-668] (SizeSat LeqExp)}$$

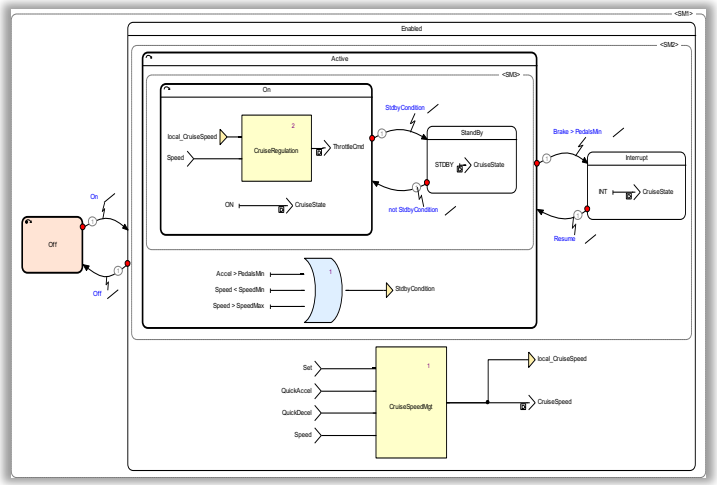
21.3.5 Constant Expressions

$$\frac{x : \tau_f \text{ is a valued constant or an enumerator}}{H, x : \tau_f \vdash (x : \tau_f) : Const} \text{ [LRM-669] (Const Env)}$$

$$\frac{}{H \vdash (\textbf{true} : \textbf{bool}) : Const} \text{ [LRM-670] (Const True)} \qquad \frac{}{H \vdash (\textbf{false} : \textbf{bool}) : Const} \text{ [LRM-671] (Const False)}$$

$$\frac{}{H \vdash (\text{CHAR} : \textbf{char}) : Const} \text{ [LRM-672] (Const Char)} \qquad \frac{}{H \vdash (\text{INTEGER} : \textbf{int}) : Const} \text{ [LRM-673] (Const Int)}$$

Mathematical formal definition



Graphical programming

Another Simple Example

Ada

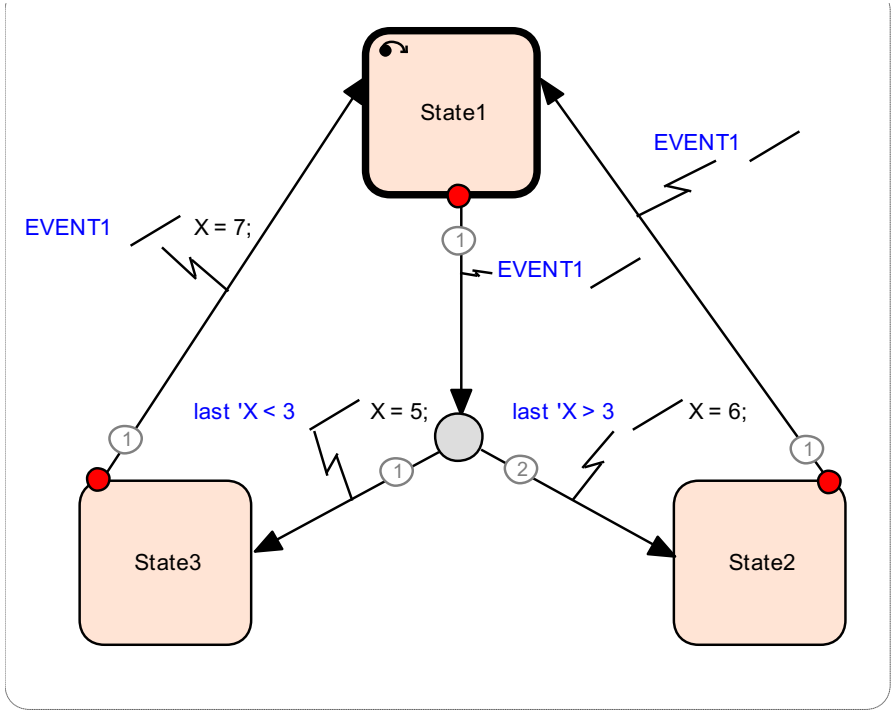
```
case State is
  when State1 => Guard1 := X < 3; Guard2 := X > 3;
   if (EVENT1 and (Guard1 or Guard2)) then
        if (Guard1) then
            X := 5;
            State := State2;
        else
            if (Guard2) then
               X := 6;
            end if;
            State := State3;
        end if;
   end if;
  when State2 =>
    if (EVENT1) then
       X := 7;
       State := State1;
    end if;
 when State3 =>
     if (EVENT1 and EVENT1) then
        X := 8;
        State := State1;
    end if;
End case
```

Simple? Yes…

But what does this piece of code do?

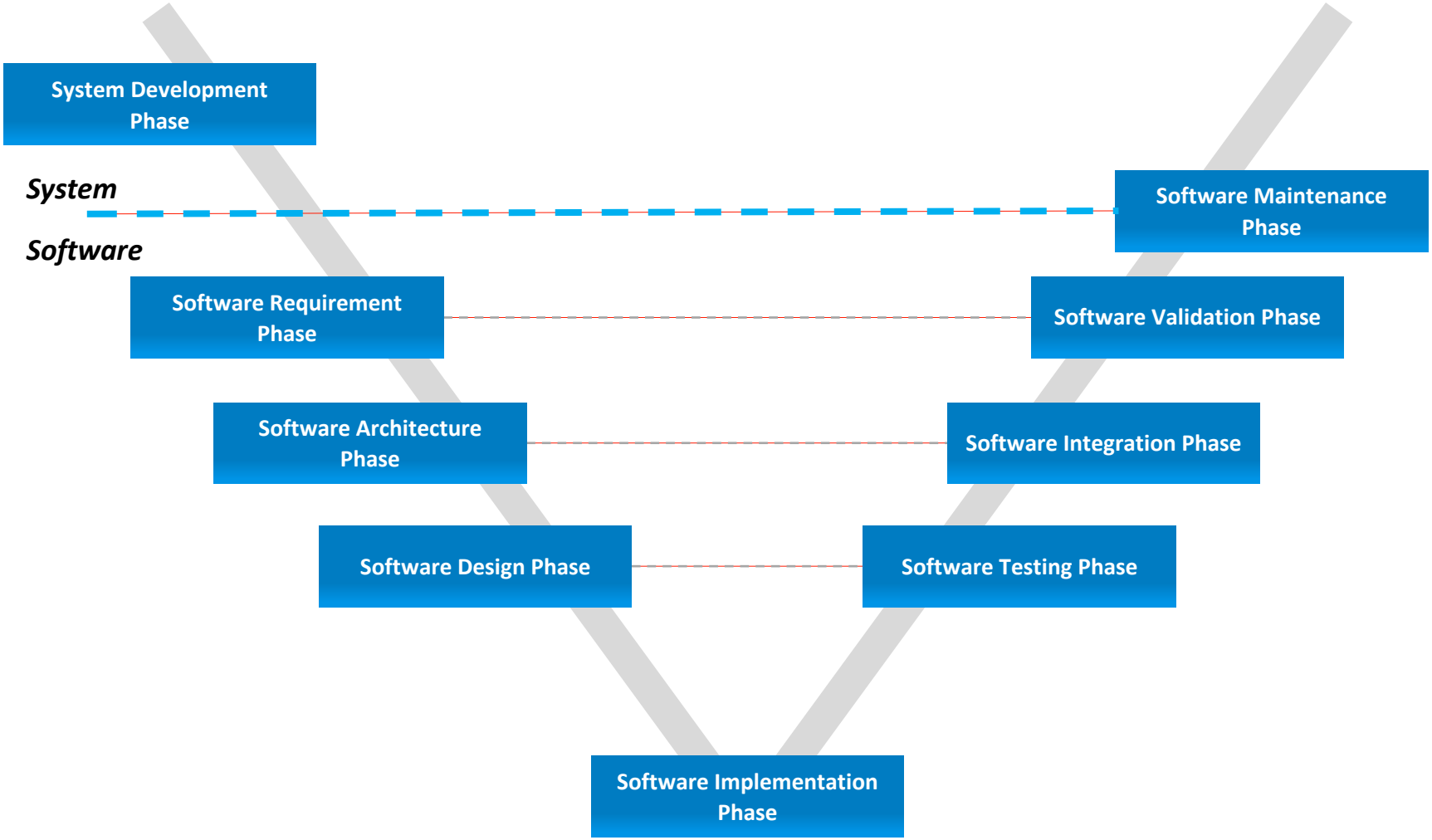
Code alone is not sufficient for communication.

The Same Example

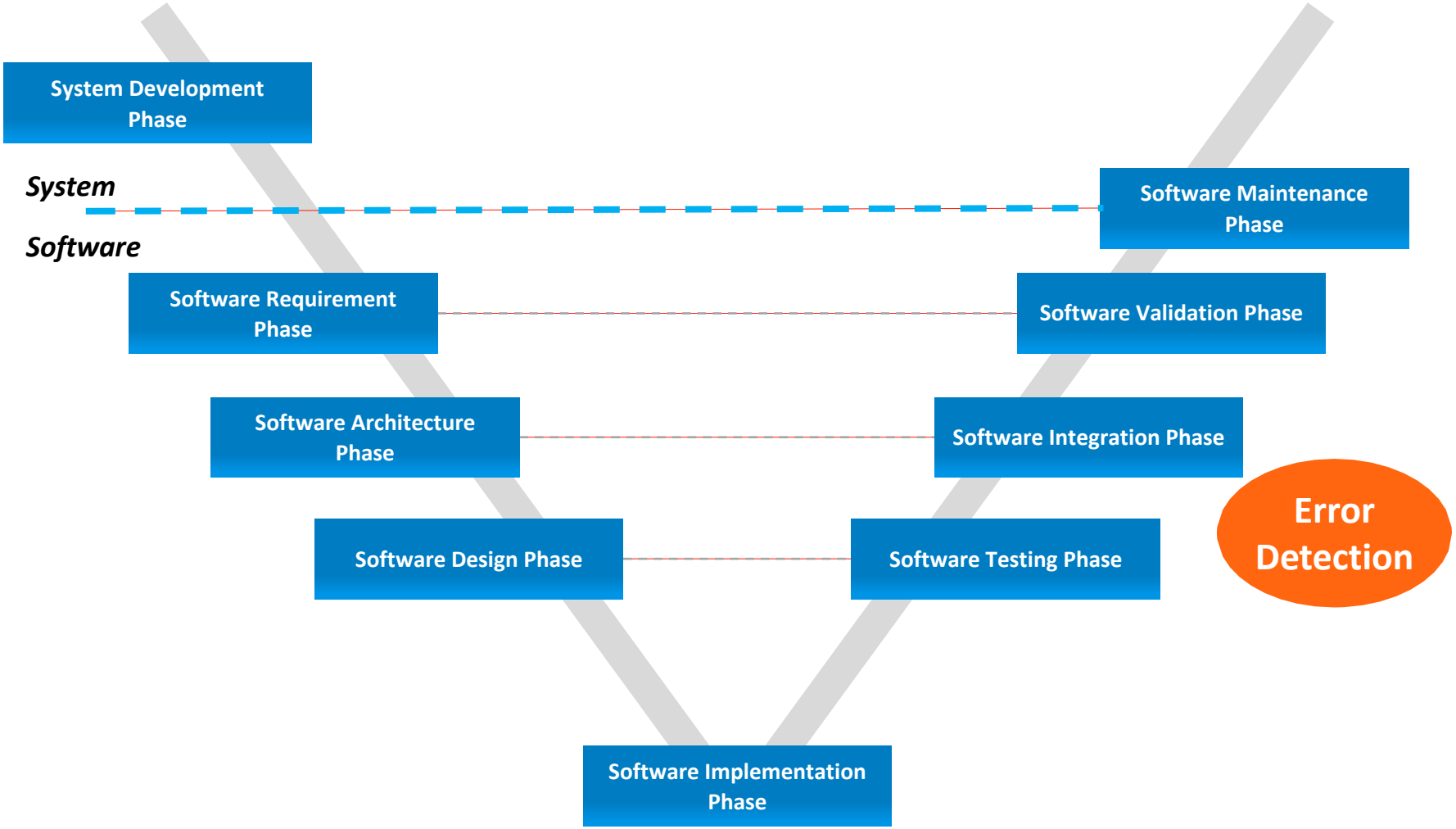


A graphical language with a high level of abstraction facilitates the communication.

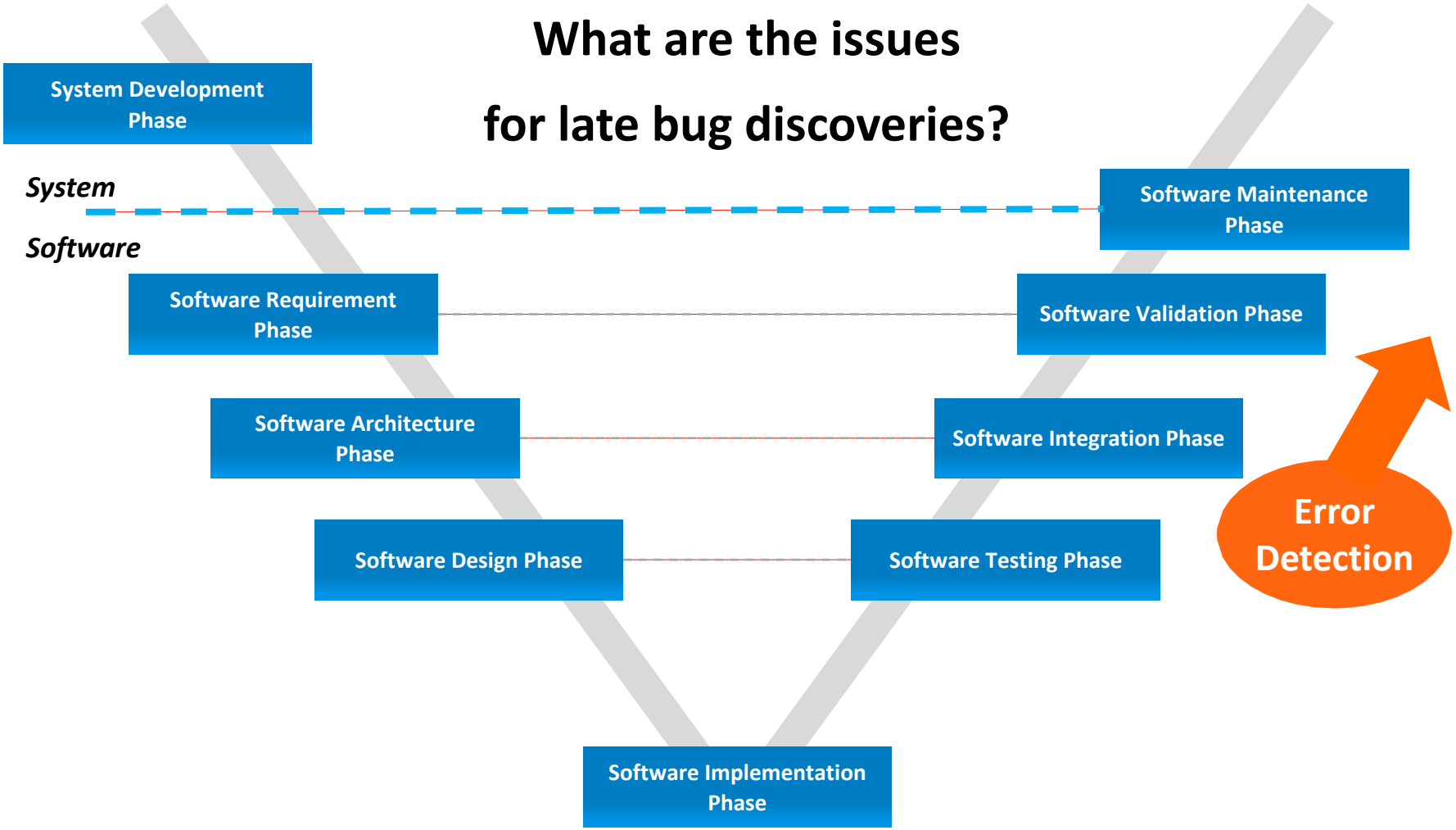
Software lifecycle



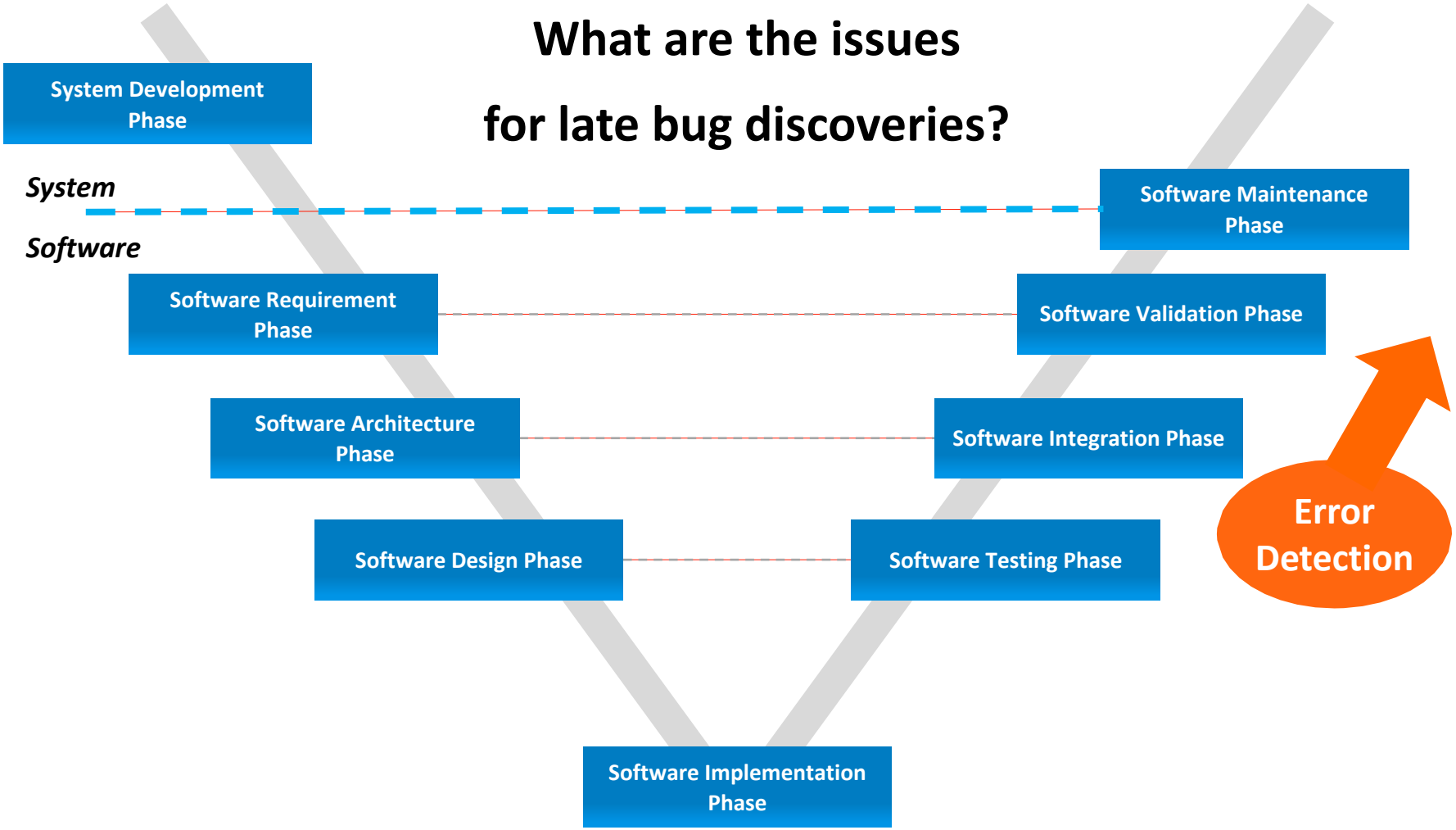
Software lifecycle



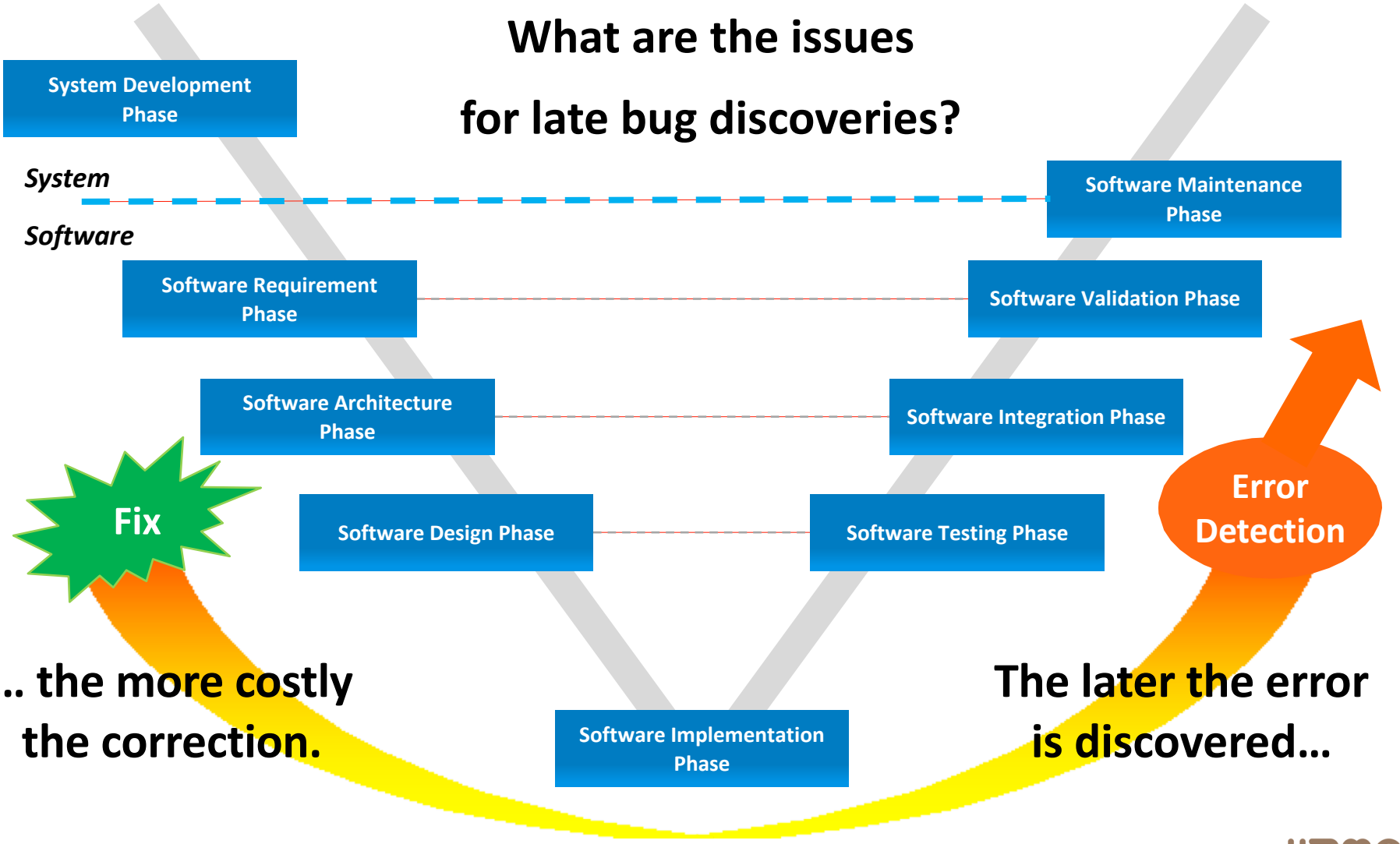
Software lifecycle



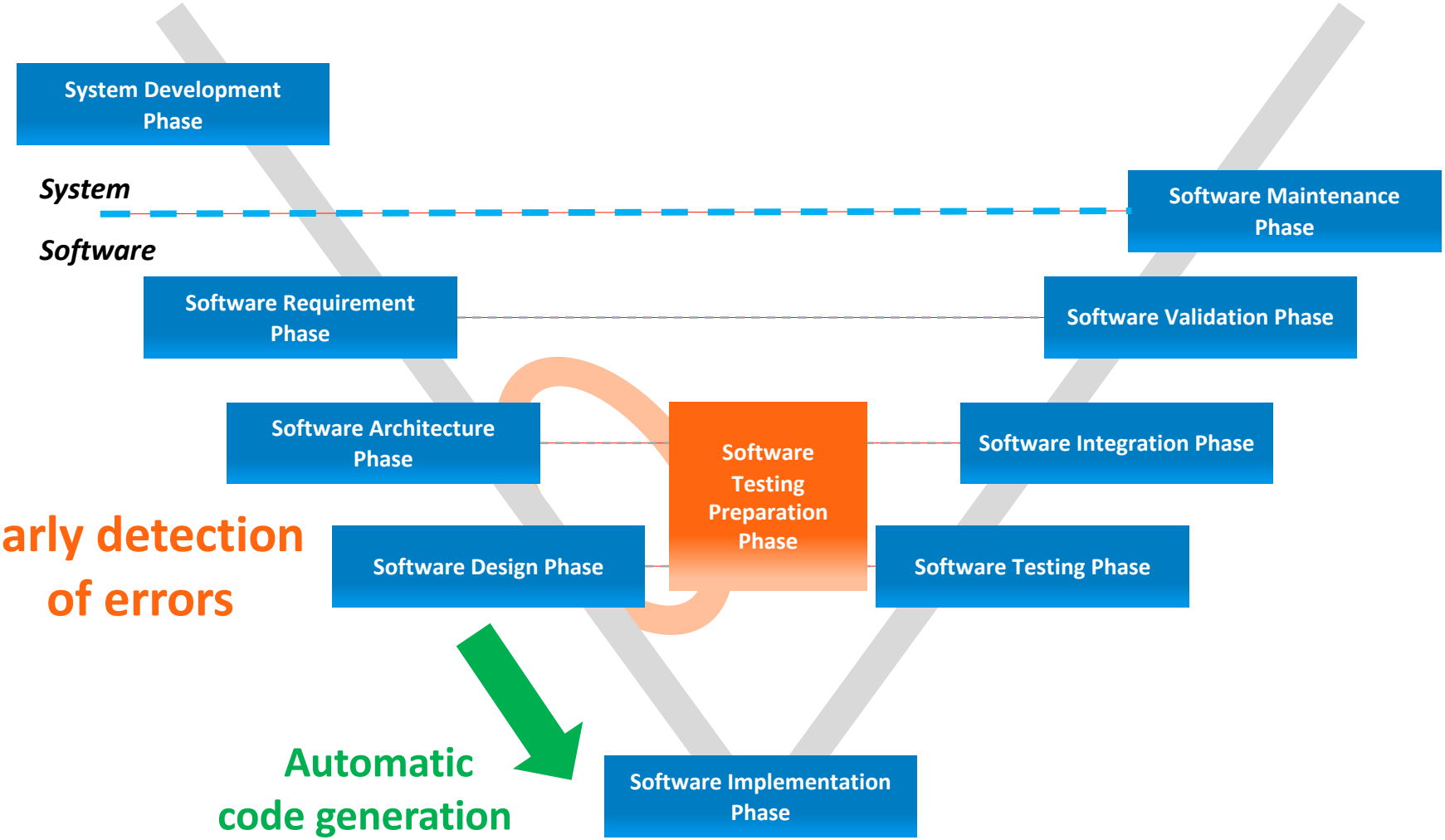
Cost of errors detection



Cost of errors detection



SCADE Software lifecycle



Safety critical and synchronous semantics

Errors lead to dramatic consequences involving human lives and huge costs.

Synchronous approach helps safety-critical application

- Providing rigorous design methods
- Providing formally defined languages and analyses

Existence of a discrete clock:

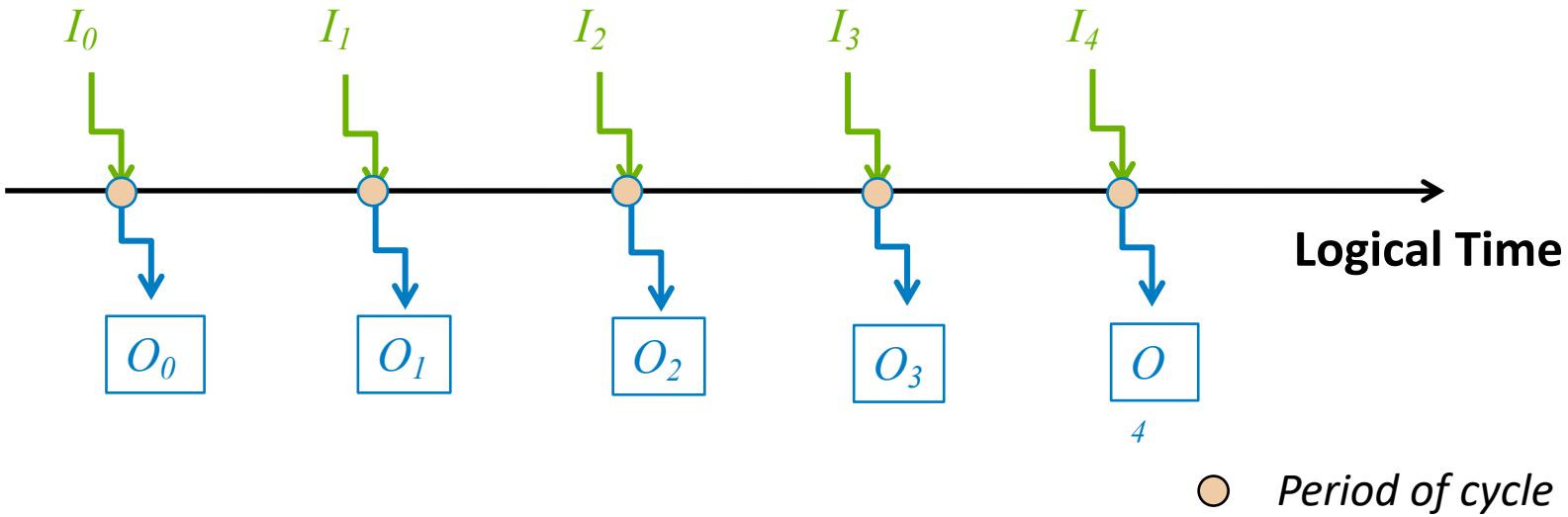
- Software **cyclically** activated,
- Inputs read at the cycle beginning,
- Outputs delivered at cycle end: read / write forbidden during the cycle.

The cycle execution duration is null:

- **No cycle overflow**

The Synchronous Time Model

Time is seen as a logical notion



In theory, execution time is null

but how to assume this hypothesis?

Synchronous language implementation

No interaction between program and world during a computation cycle

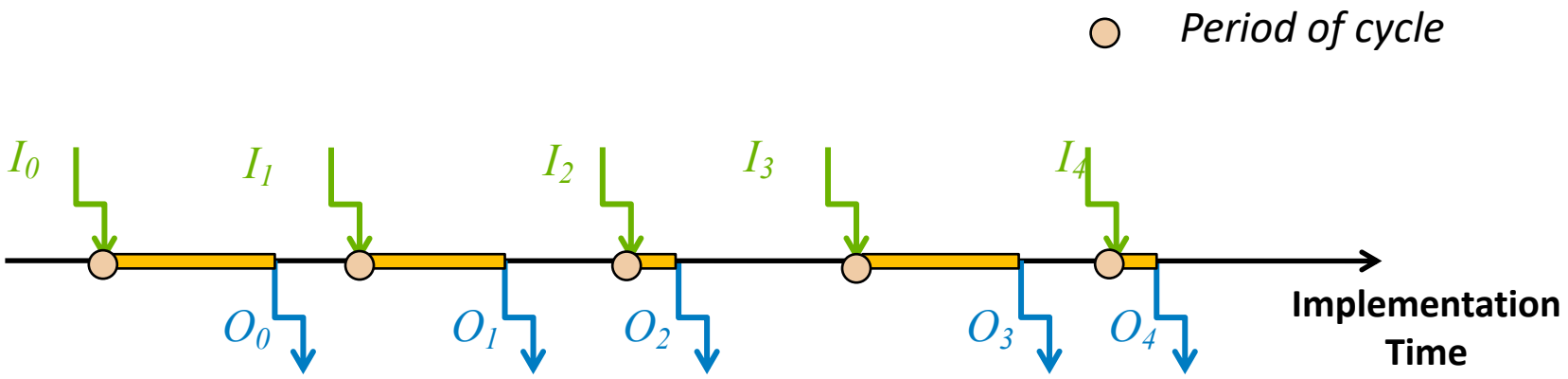
➔ **strong theoretical and practical properties:**

- **Inputs must not change during the execution cycle: using a double buffer or a write protection or any other architecture.**
- **Internal variables and outputs are frozen as soon as they are computed during the execution cycle:**
  - **Once computed, they are never modified later during the cycle,**
  - **They only have one value at a given cycle.**

➔ **Fully deterministic behavior, finite computation time.**

➔ **Much easier to verify than asynchronous programs.**

Synchronous Time Implementation



Finite computation time

Path-based Worst Case Execution Time (WCET)

Guarantee of no-overlap

History of synchronous languages

Created in the early 80's from a joint research effort by control engineering and computer sciences researchers.

Lustre

P. Caspi – N. Halbwachs
(VERIMAG)

Signal

A. Benveniste – P. Le Guernic
(IRISA)

Esterel

JP Rigault, JP Marmorat – G. Berry
(INRIA/Ecole des Mines)

History of synchronous languages

Created in the early 80's from a joint research effort by control engineering and computer sciences researchers.

Lustre

P. Caspi – N. Halbwachs
(VERIMAG)

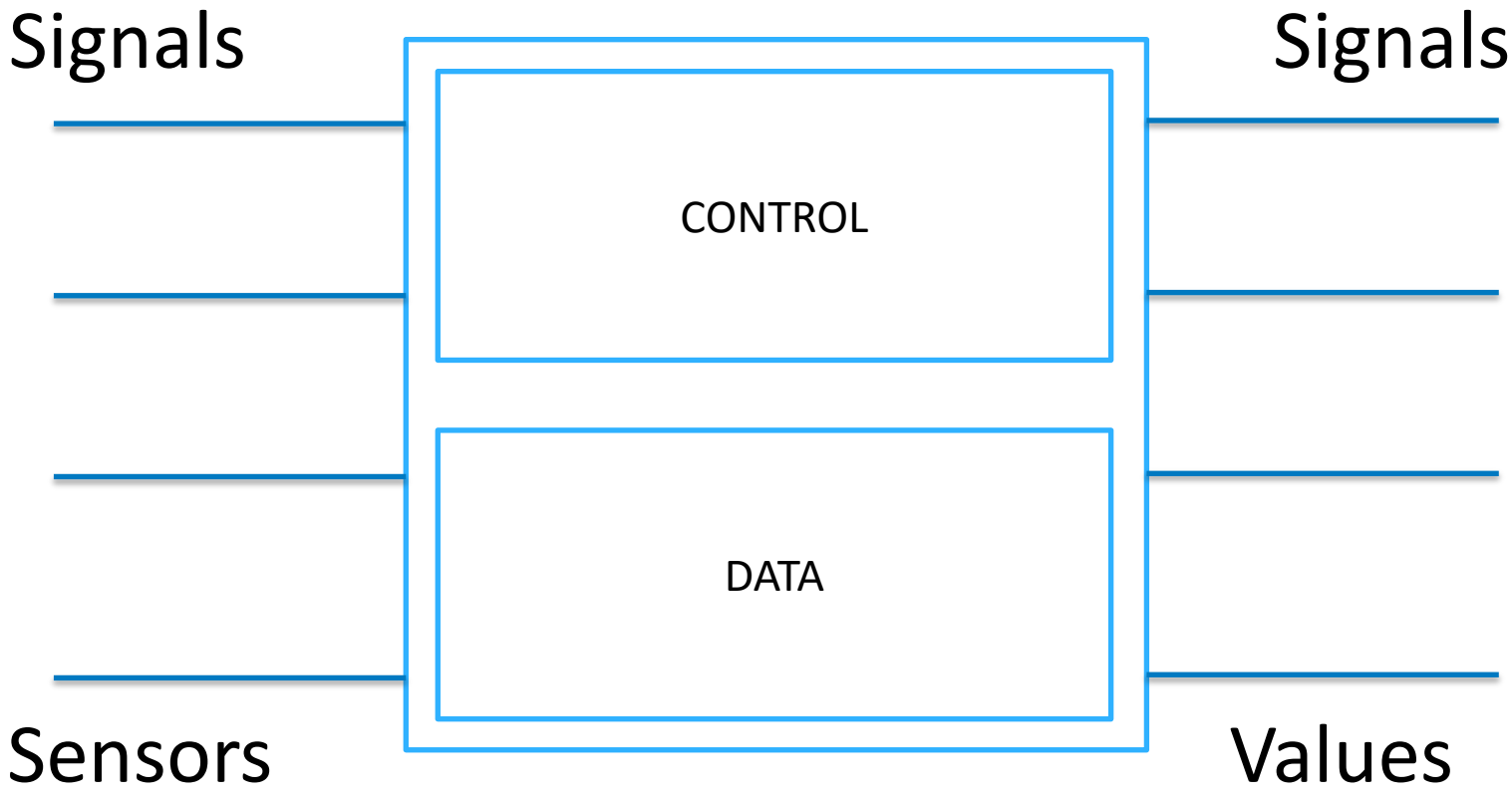
Signal

A. Benveniste – P. Le Guernic
(IRISA)

Esterel

JP Rigault, JP Marmorat – G. Berry
(INRIA/Ecole des Mines)

ESTEREL – Logic



Links with automatics / circuits
(joysticks, plane controls)

ESTEREL – The race track example

```
Module Training :
        input Second, Hour, Morning;
        relation Hour => Second, Morning => Second;
        input Meter, Lap, Step, HeartBeat;
        relation Lap => Meter;
        … // behavioral code
End module
```

Execution of a sub-module

run WalkSlowly

Pre-emption !

abort run WalkSlowly when 100 Meter ;

```
abort   run WalkSlowly when 100 Meter ;
abort
        every Step do
        run Jump || run Breathe
        end every
when 15 second ;
```

Sequence

Parallelism

```
abort
loop
        abort   run WalkSlowly when 100 Meter ;
        abort
                every Step do
                run Jump || run Breathe
                end every
        when 15 second ;
        run RunFullSpeed
each Lap
when 4 Lap
```

```
every Morning do
        abort
        loop
                abort   run WalkSlowly when 100 Meter ;
                abort
                        every Step do
                        run Jump || run Breathe
                        end every
                when 15 second ;
                run RunFullSpeed
        each Lap
        when 4 Lap
end every
```

```
trap HeartAttack in
every Morning do
        abort
        loop
                abort   run WalkSlowly when 100 Meter ;
                abort
                        every Step do
                        run Jump || run Breathe || <CheckHeart>
                        end every
                when 15 second ;
                run RunFullSpeed
        each Lap
        when 4 Lap
end every
handle HeartAttack
        run RushToHospital
end trap
```

Code of <CheckHeart>

```
loop
        await 3 Second ;
        exit HeartAttack
each Heartbeat
```

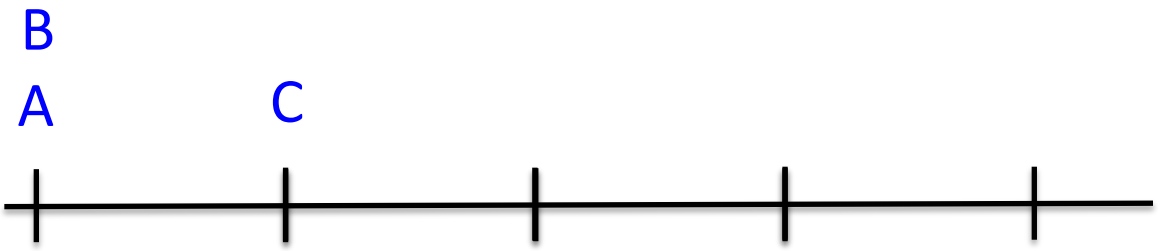
Every HeartBeat will cancel a waiting for 3 Seconds

ESTEREL – Foundation ideas

- Perfect synchronism
  - Administration / communication in 0 time
  - Deterministic and understandable parallelism
  - Behavioral sharing
- Mathematical semantics
- Professional implementation and use
  - Formal checking
- Professional use for critical software

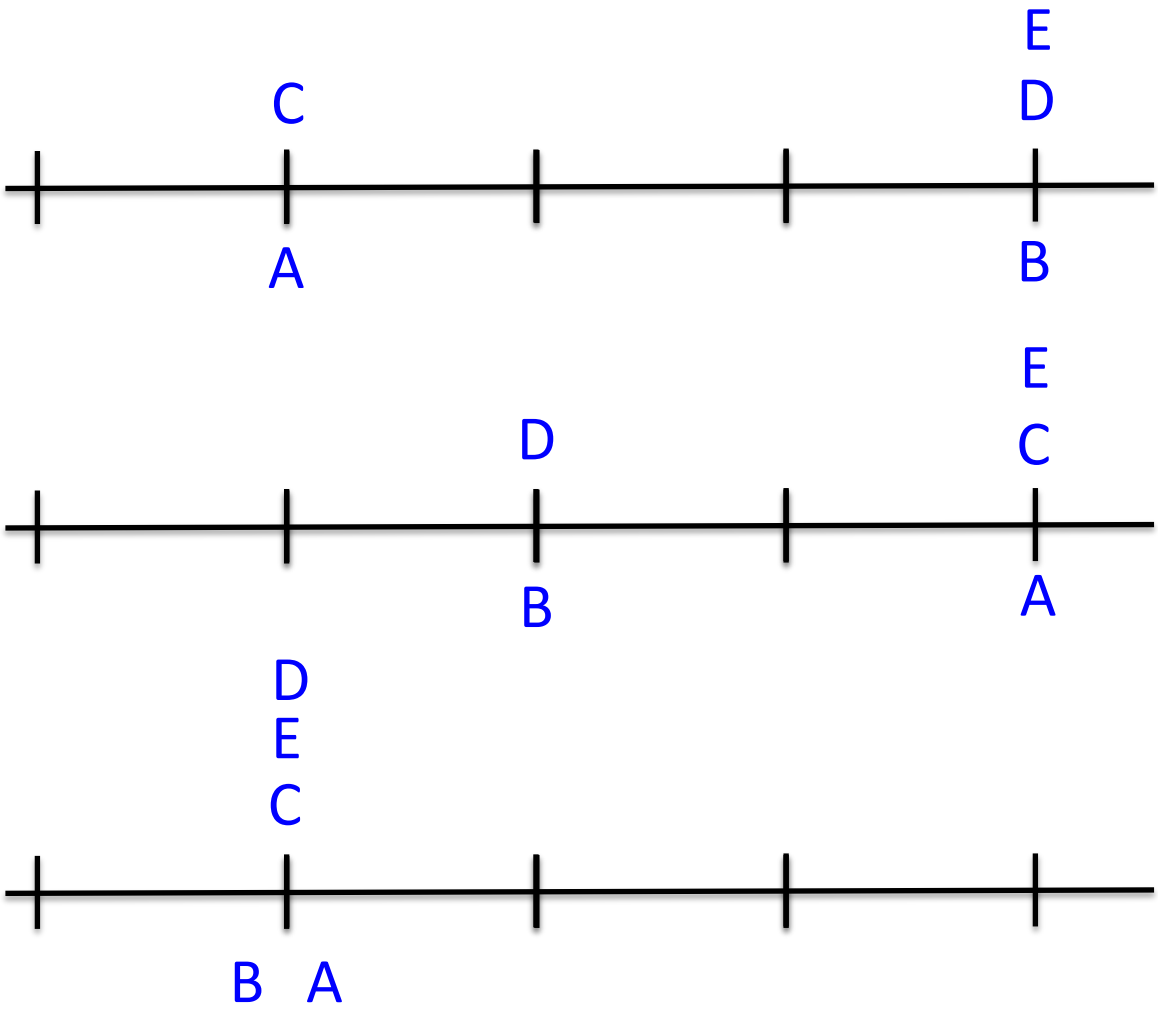
ESTEREL – Sequence

emit A ; emit B ; pause ; emit C



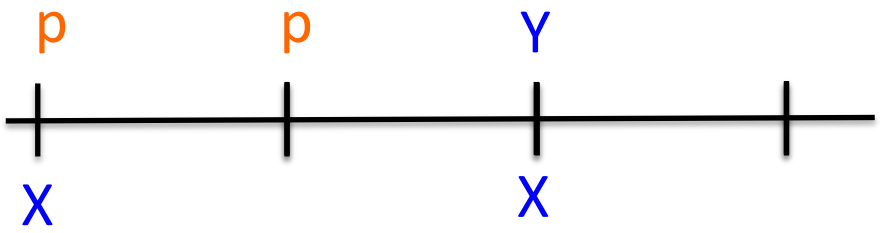
ESTEREL – Parallelism

[await A ; emit C || await B ; emit D] ; emit E

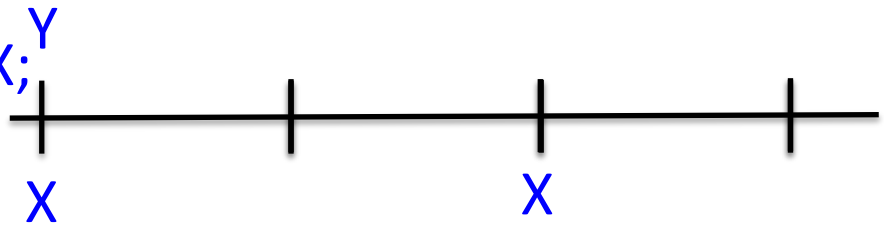


Strong preemption

abort p when X;
emit Y

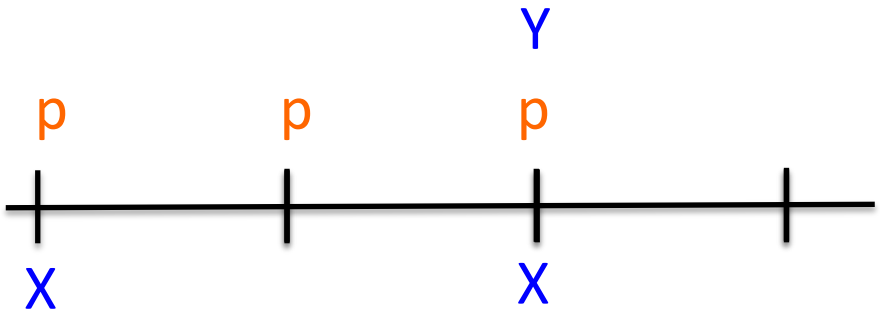


abort p when immediate X;
emit Y

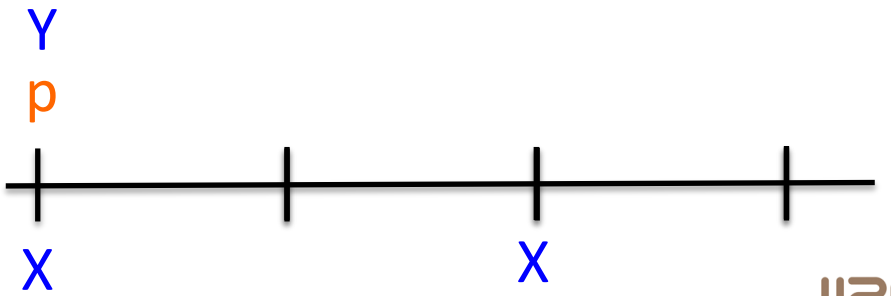


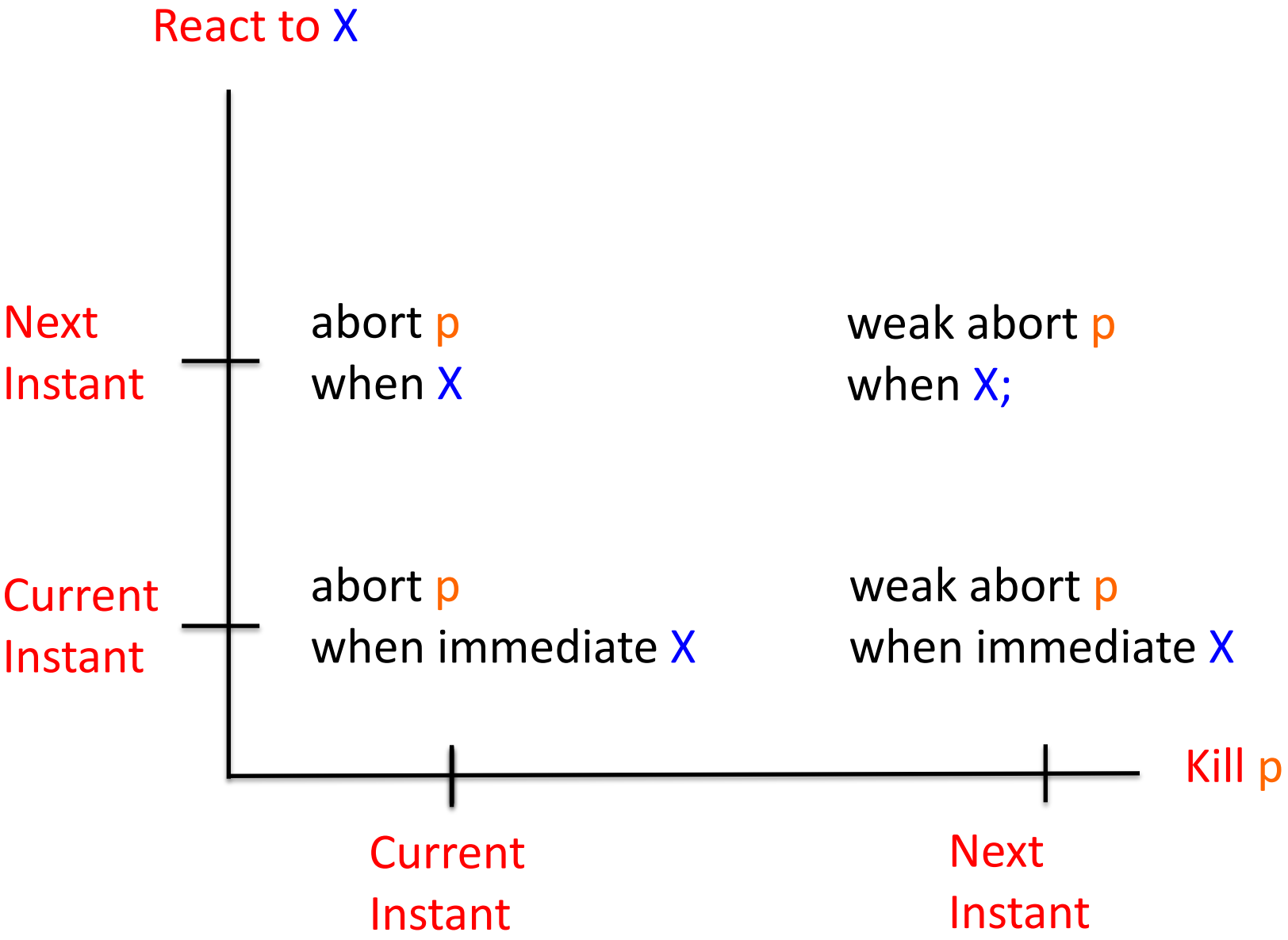
Weak preemption

weak abort p when X;
emit Y



weak abort p when immediate X;
emit Y




React to X
Next Instant
abort p when X
weak abort p when X;
Current Instant
abort p when immediate X
weak abort p when immediate X
Kill p
Current Instant
Next Instant

ABRO: ESTEREL's Fibonnaci

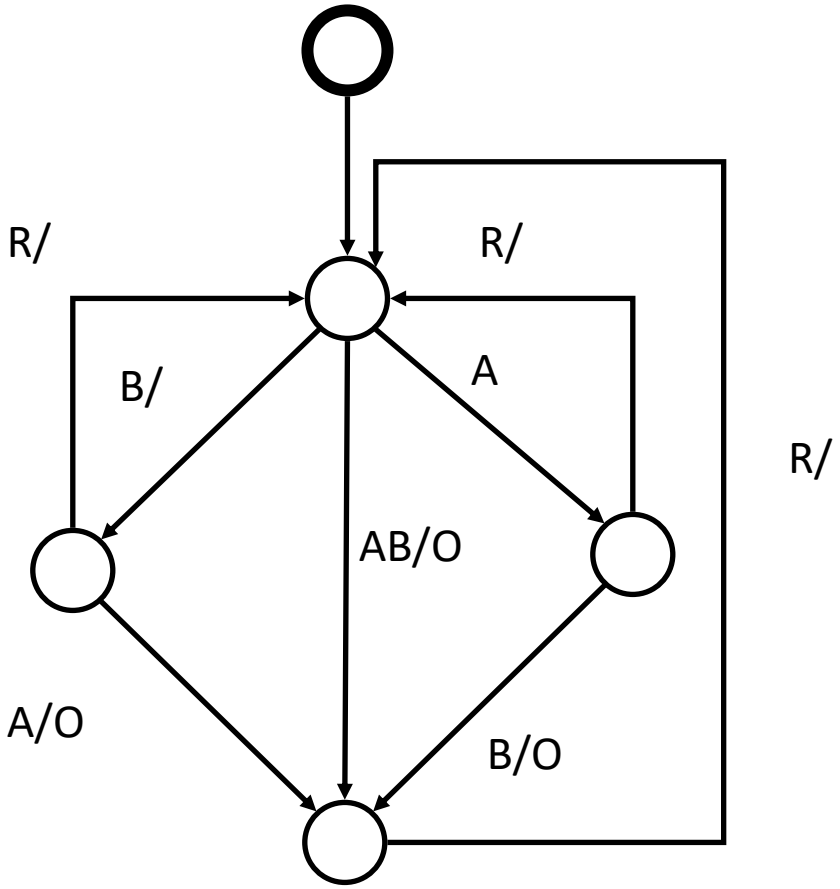
Emit O as soon as A and B are arrived

Re-init the behavior at each R

ABRO: ESTEREL's Fibonnaci

Emit O as soon as A and B are arrived

Re-init the behavior at each R




```
module ABRO
input A, B, R;
output O;

loop
  [ await A || await B ];
  emit O
each R

end module
```

SCADE – LUSTRE successor

Scade language is formally defined with key safety objectives:

- The language is simple and stable, forbidding dangerous constructs (e.g. unbounded loops, goto, dynamic memory allocation,…)
- Interpretation of a model does not depend on the readers or their environment.

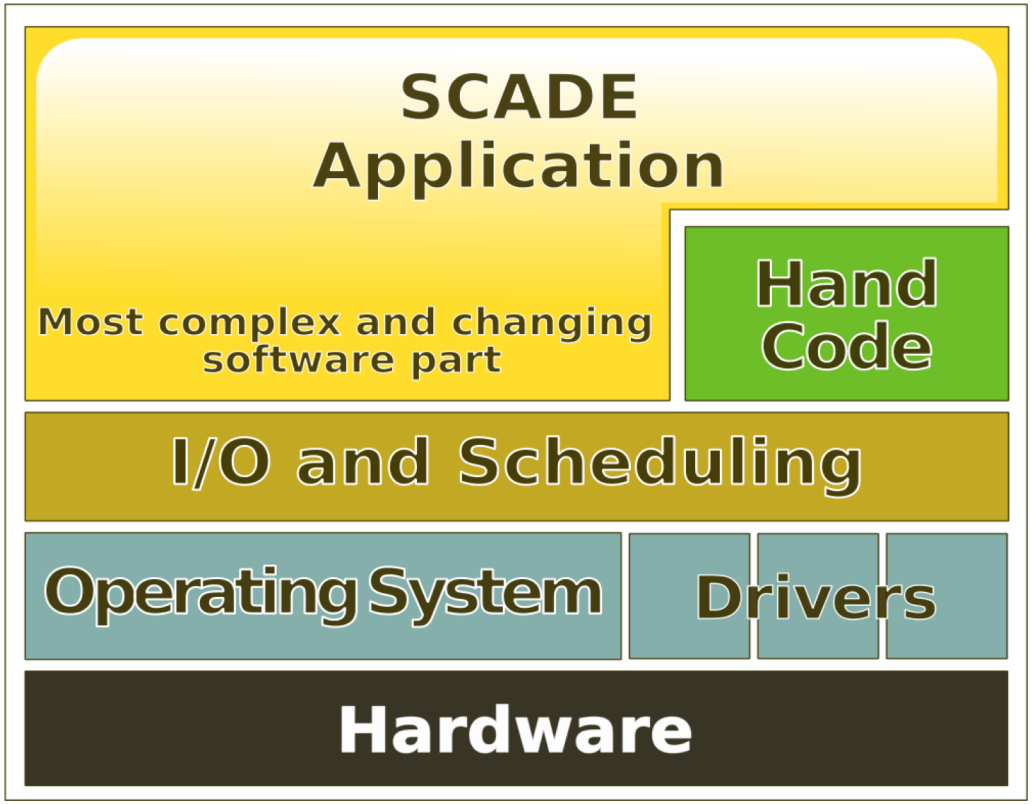
Very active research work for more than 20 years:

- Worldwide visibility in the safety critical embedded software field.

Designed with close connections with certification authorities in the aeronautics & nuclear energy domains:

- SCADE Suite KCG is a C code generator developed with stringent qualification objectives (DO-178B/C DAL A, IEC 62508 SIL3, EN 50128 SIL3/4, ISO 26262 ASIL D)

SCADE Suite - Application part



The application part is the largest and most frequently/lately changing part

SCADE Fields of application

Continuous control:

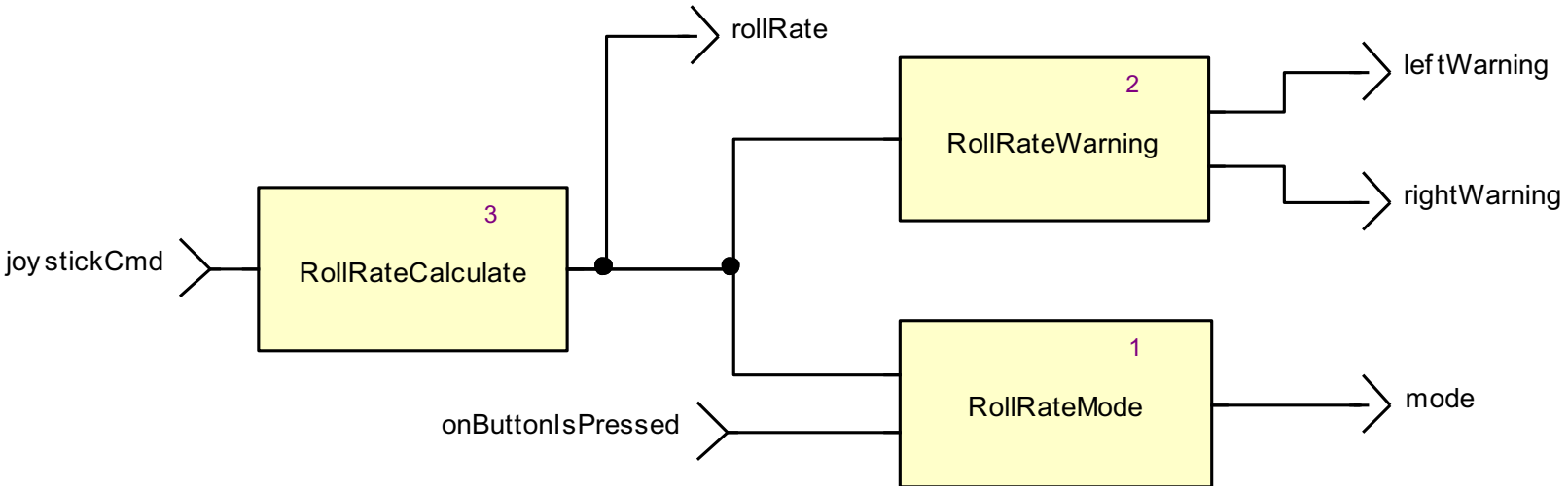
- Performs data processing tasks using complex mathematics
- Updates actuator's outputs

Discrete control:

- Reacts from external or internal events: 0/1 sensors, alarms, threshold detection
- Performs combinational logic operations to compute the new system's state
- Updates actuator's outputs

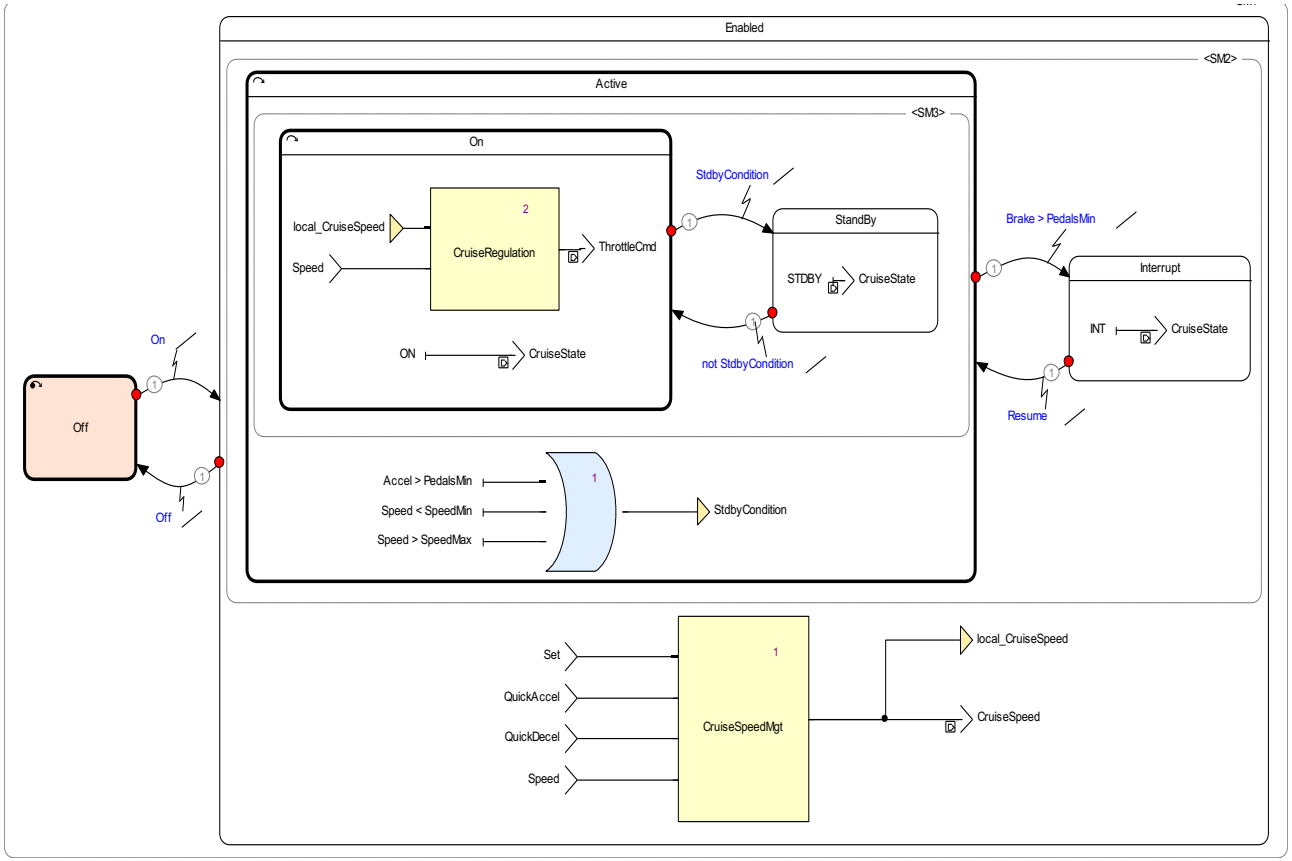
SCADE Language

A simple network of operators (functions)



SCADE Language – State machines

Hierarchical state machines, that can be mixed with data-flow



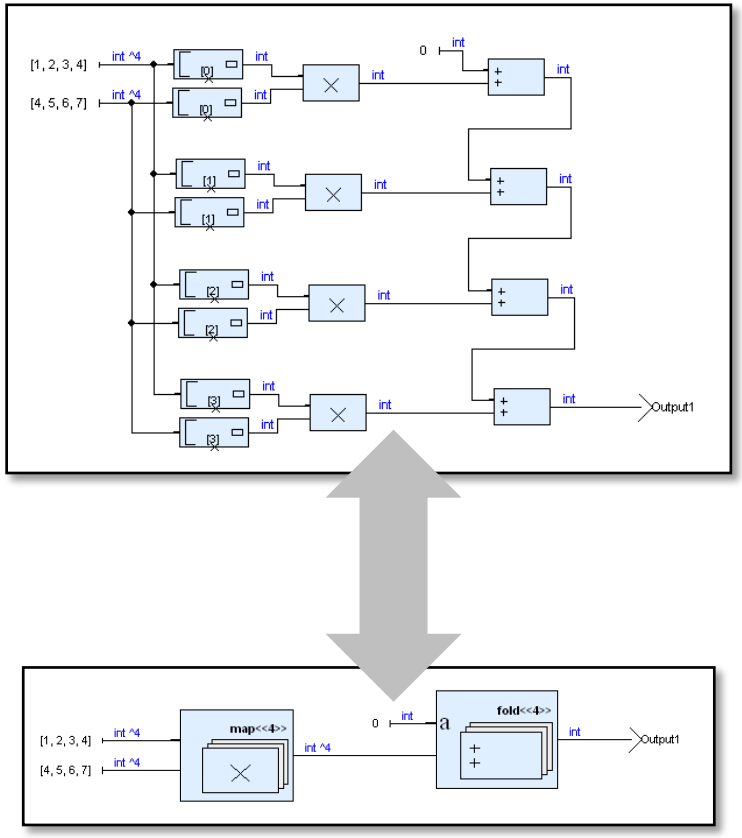
SCADE Language

Arrays & Iterators

Scade provides **array data types** and **iterators**.

An iterator applies an operator over arrays.

This optimizes the design while preserving the safety (bounded size).



SCADE Language

Properties

- Parallelism is naturally expressed.
- The **computation order is based on dependencies** (Not on the diagrams' layout or on hidden properties):
  - Each data element is computed before it is used, once and only once.
- The **generated code is efficient and deterministic** (no tasking overhead, no deadlock, no race condition).
- **Execution time is finite**.

SCADE Suite KCG

A qualifiable/certified code generator

- Ensures that the input model complies with language syntax and semantics.
- Generates C or Ada code fitting safety-critical constraints.
- KCG is deterministic:
  - A given input model and a given set of options always produce the same generated code
- Qualified/certified for:
  - DO-178B/DO-178C level A,
  - EN 50128 up to SIL 3/4,
  - IEC 61508 up to SIL 3,
  - ISO 26262 up to ASIL D.

Puredata

Puredata is a graphical programming environement for handling audio, video and graphics

Puredata

Puredata is a graphical programming environement for handling audio, video and graphics

Compared to what we have seen up to now PureData is still

- A **graphical**, **reactive** (potentially **embeded**) programming language
- However, a **different notion of criticality** is at play

Puredata

Puredata is a graphical programming environement for handling audio, video and graphics

Compared to what we have seen up to now PureData is still

- A **graphical**, **reactive** (potentially **embeded**) programming language
- However, a **different notion of criticality** is at play

We are not flying planes around but yet …

- The human ear has a latency threshold around 15 ms.
- Typical FFT window size is 512 samples (61.5 ms)
- System designs usually runs between 1 to 30 ms delays

Puredata

Puredata is a graphical programming environement for handling audio, video and graphics

Compared to what we have seen up to now PureData is still

- A **graphical**, **reactive** (potentially **embeded**) programming language
- However, a **different notion of criticality** is at play

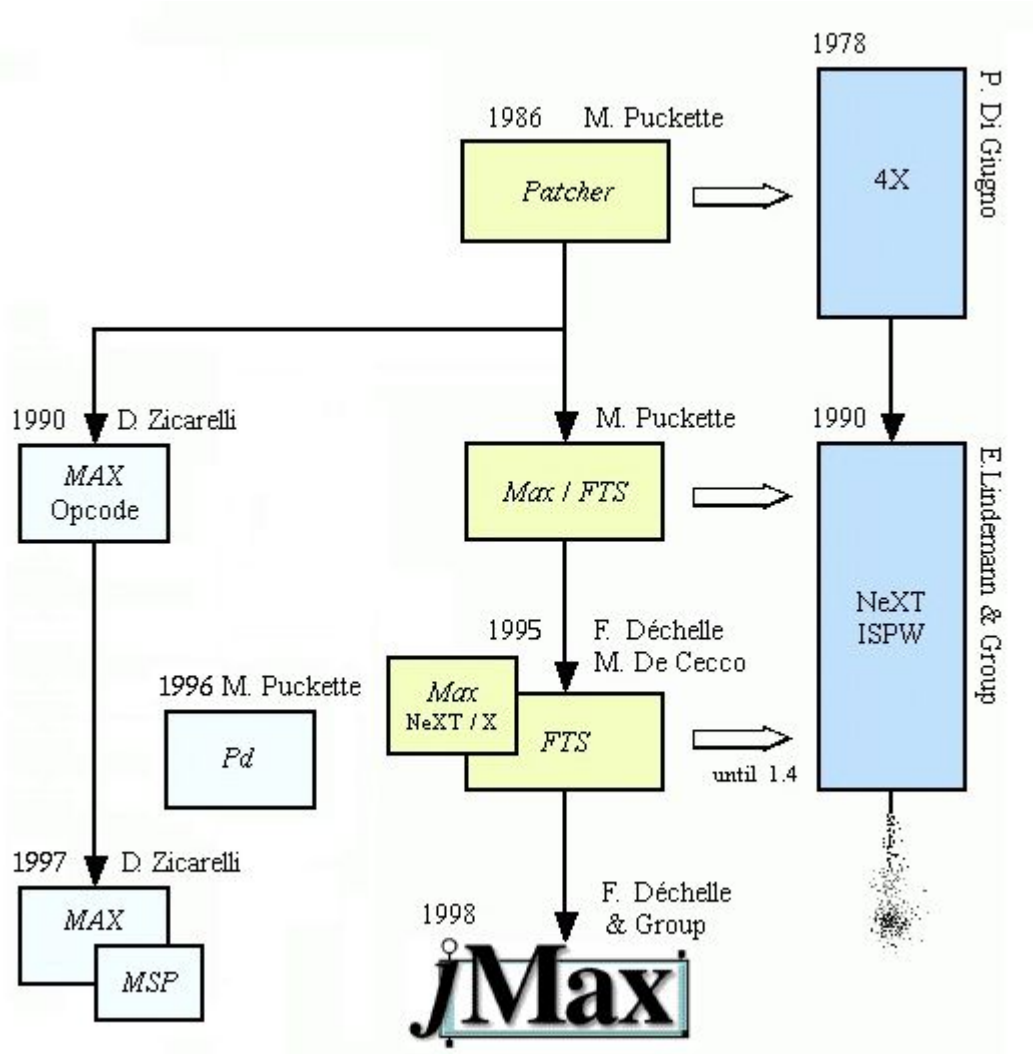
We are not flying planes around but yet …

- The human ear has a latency threshold around 15 ms.
- Typical FFT window size is 512 samples (61.5 ms)
- System designs usually runs between 1 to 30 ms delays

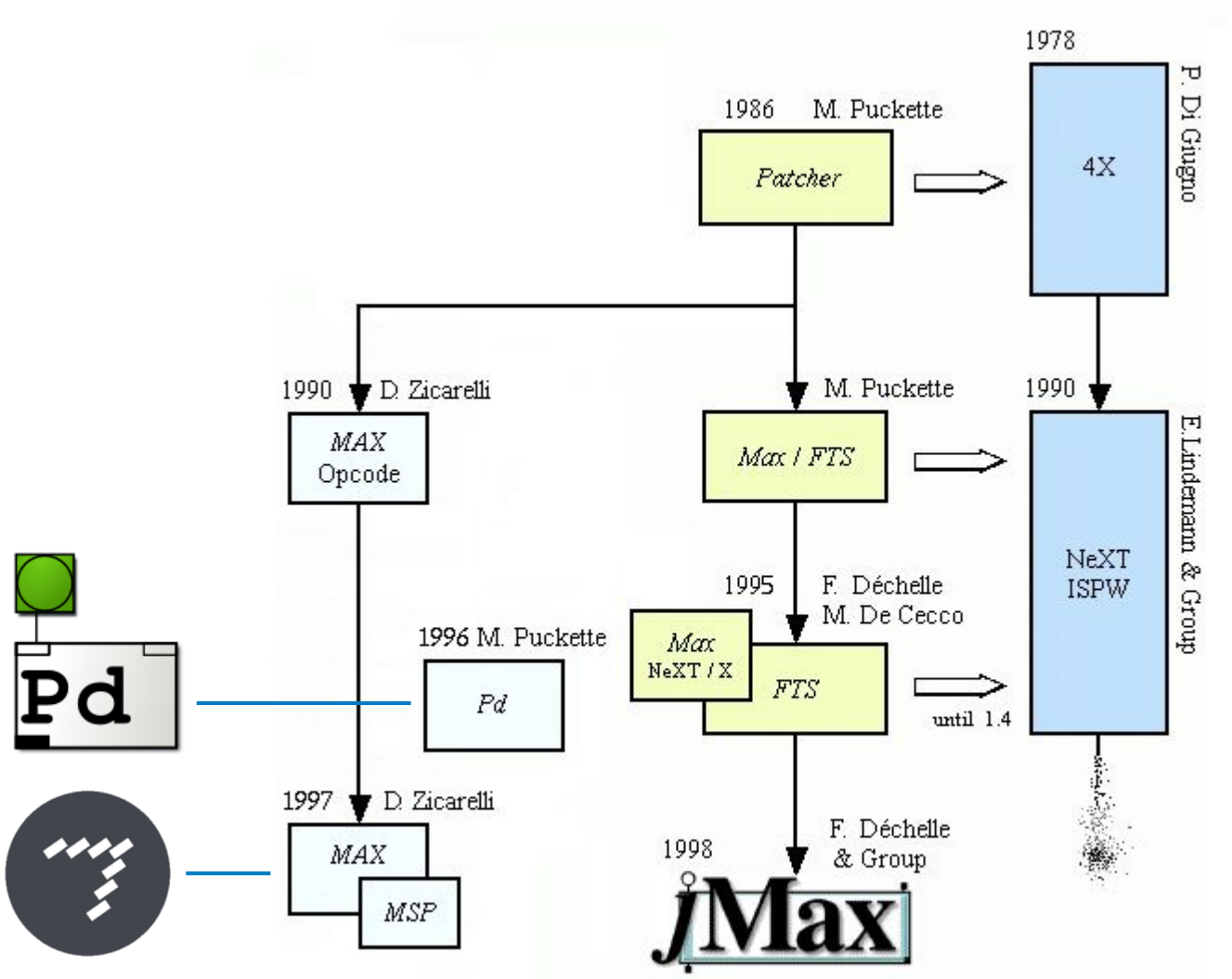
Hence, we have very hard **real-time constraints** in programming

- You might hear systems being « 300 times real-time »
- Means you can process the length of 300 buffers (of your specs)
- Will be a **huge problem** when coding **externals**

Patcher/Max/Pd Family Tree



Patcher/Max/Pd Family Tree



Puredata

Developed by Miller Puckette.
Pure Data has two major components

- Pure Data Vanilla
- Pure Data Extended

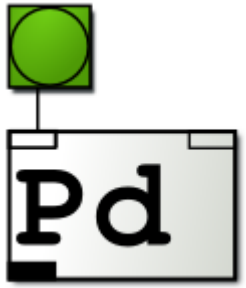
Pd-vanilla
Allows manipulating audio and MIDI.
Is the basic version of Pure Data.

Puredata

Developed by Miller Puckette.
Pure Data has two major components

- Pure Data Vanilla
- Pure Data Extended

Pd-extended
Allows to add video processing
Communication with USB and Firewire
Single-shot complex operations handling



Puredata

INSTALATION

The official website www.puredata.info, provides the pre-compiled version for the most known OS GNU/Linux, Mac OS X et Microsoft Windows).

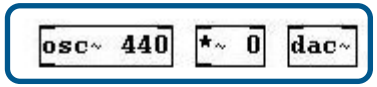
QUICK START TUTORIALS

http://fr.flossmanuals.net/puredata/

Puredata

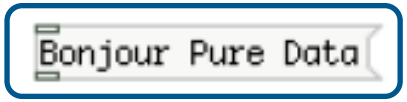
Puredata is a programming environement where the commands and operations are defined through a set of *boxes*.

Three major types of boxes are defined



Objects

These boxes contain a function (operation) and arguments



Messages

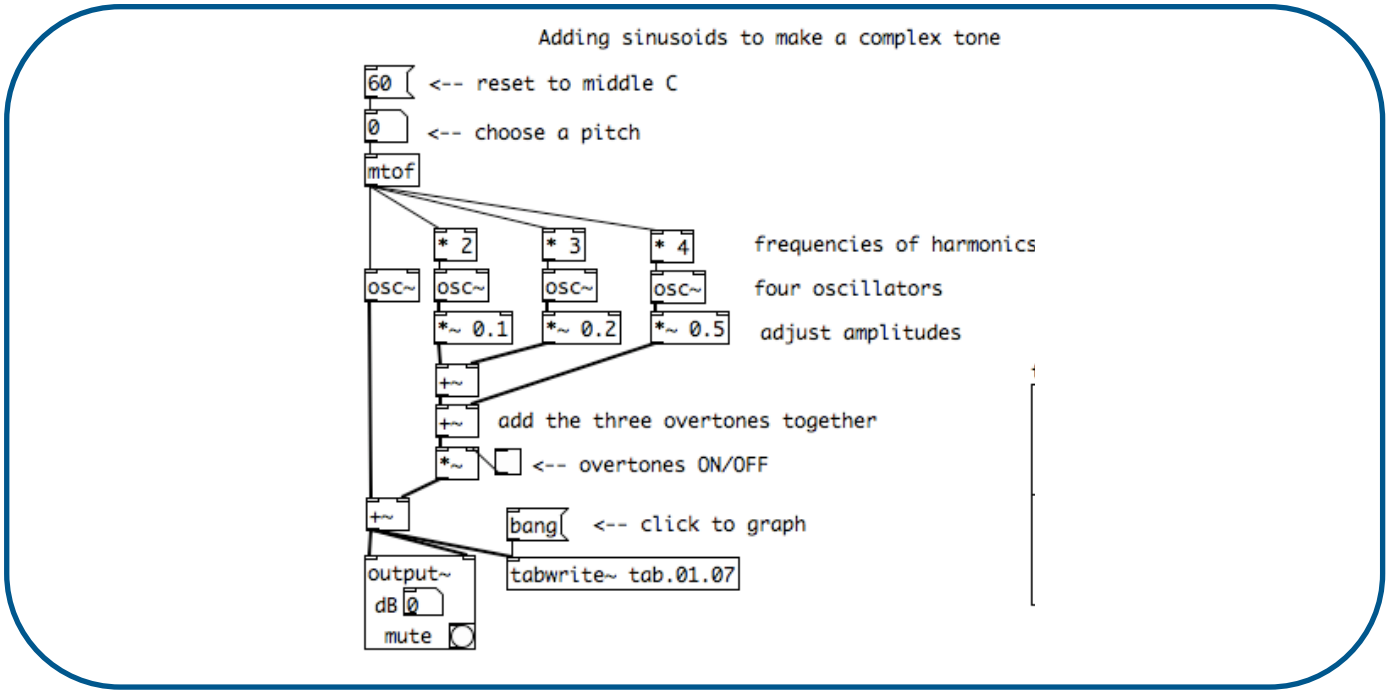
Boxes used to store messages (we will see their purposes later)

Nombres

Boxes to output or display numbers

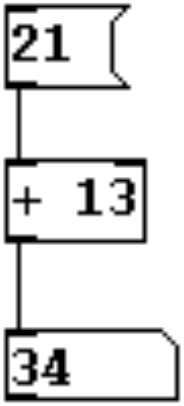
Puredata

This is a PureData algorithm, called a **patch**



My first PD patches

Patches are created by associating different objects (boxes) through *links,* which carry the information between all different functions.



Message – send messages

Object – do things

GUI – number / slider / toggles

First exercise

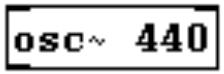
- Reproduce this patch
  - Create new patch
  - Ensure you are in edit mode
- Try to fiddle with the parameters and objects
- Use the [print] operation to observe values
  - Send simple messages or numbers to this object

More PD elements

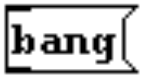
Objects - Perform operations

Messages – contains information (any type)

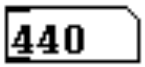
Numbers – numeric information



Objeto

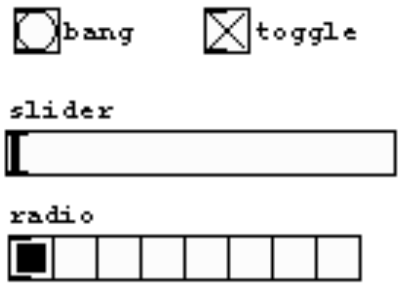


Mensaje

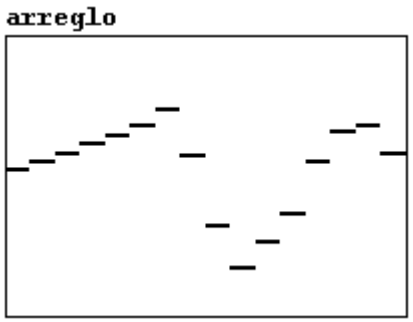


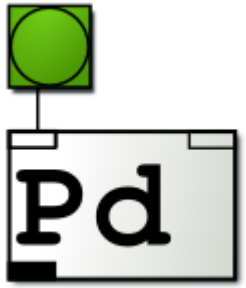
Número

More variety of GUI objects



Displays can be used to plot an array of numbers

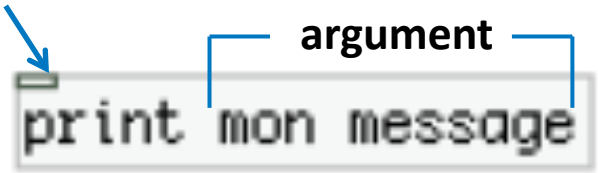


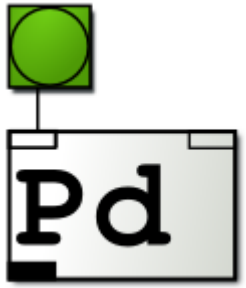


Puredata

The object boxes

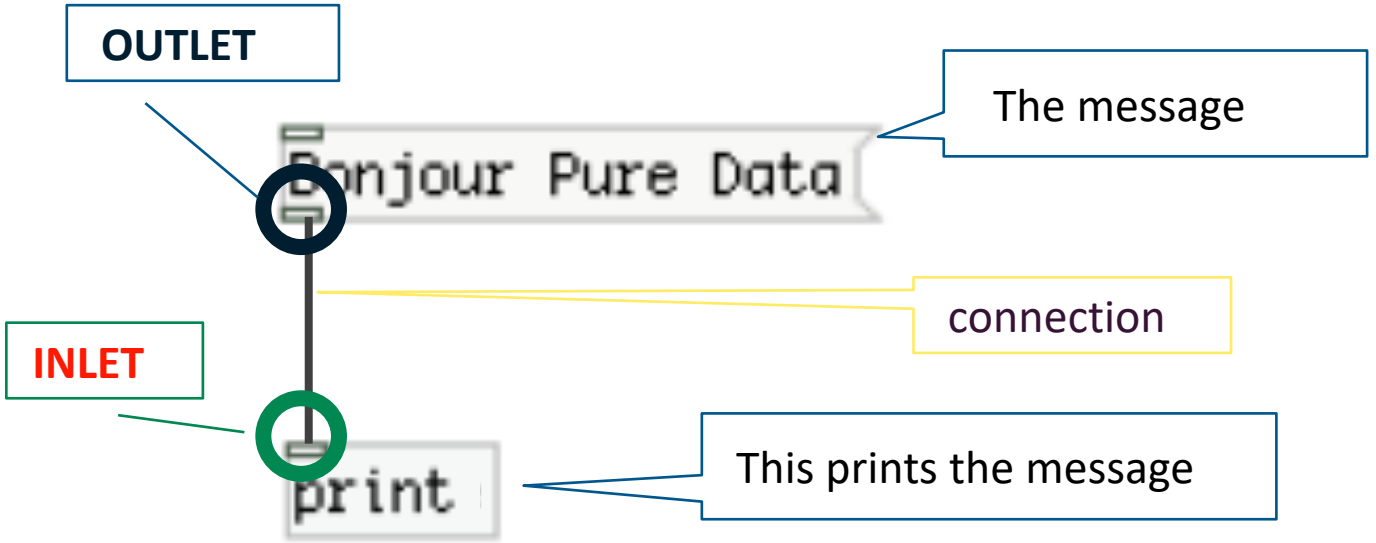
Function **print (**display)





Puredata

We have to link boxes together in order to code an algorithm



The « message » box can send a command to an object (see later)

Different types of information

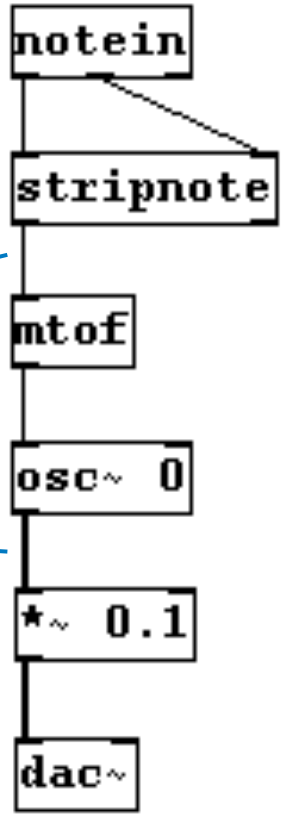
Inlets on top

Outlets on bottom

Control – (messages) thin lines

Signal – (audio) thick lines

Signal processing objects often have “~”

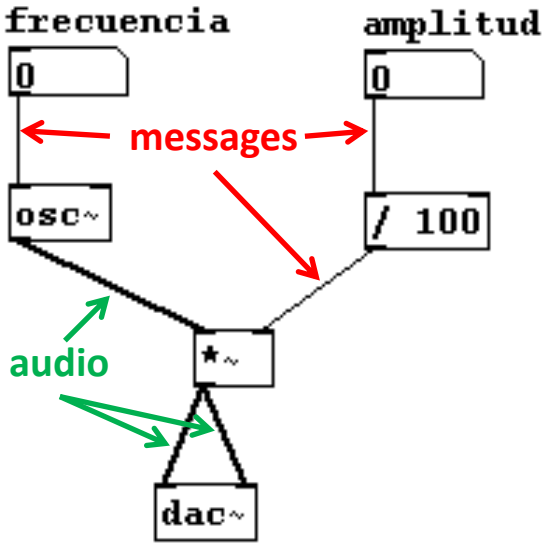


Different types of information

We can mix the different types of information in the same patch

Messages can be used as a way to control audio objects

Audio is actually a continuous stream of numbers so you need particular objects to be able to display it (try number ?)

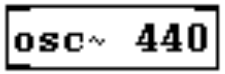


More PD elements

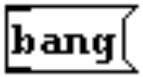
Objects - Perform operations

Messages – contains information (any type)

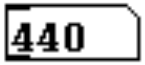
Numbers – numeric information



Objeto

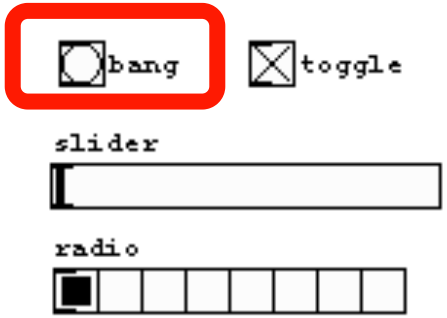


Mensaje

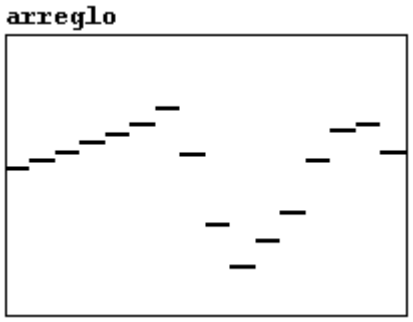


Número

More variety of GUI objects



Displays can be used to plot an array of numbers

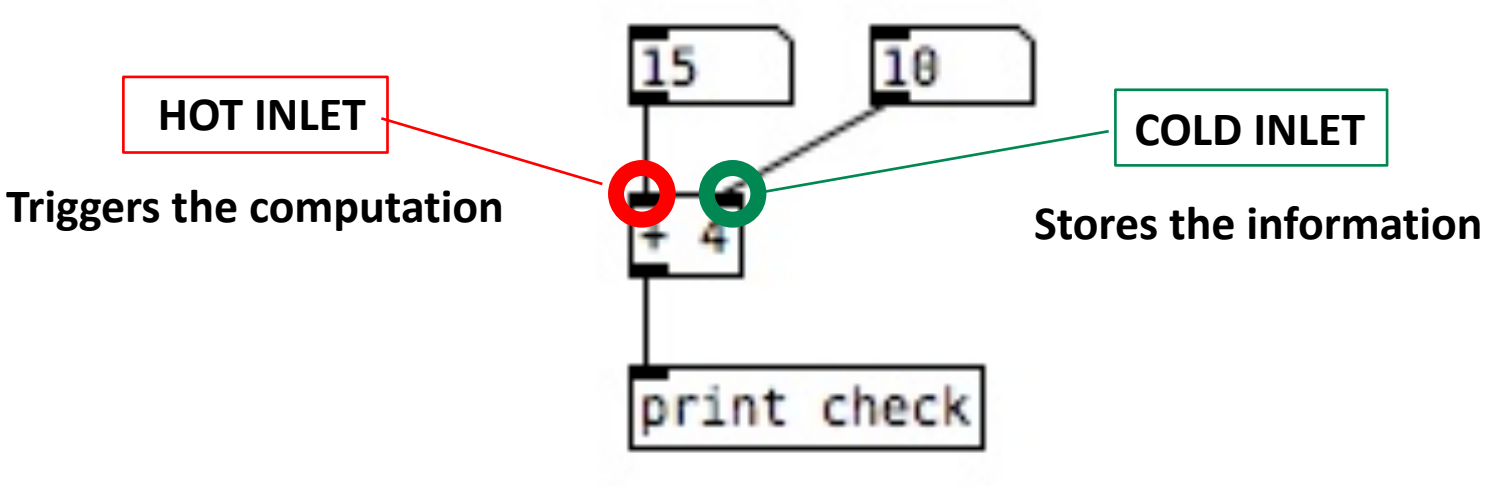


Understanding the bang message

Bang sends a trigger message & makes stuff happen

Cold vs. hot inlets

Different types of inlets exist in Pure Data

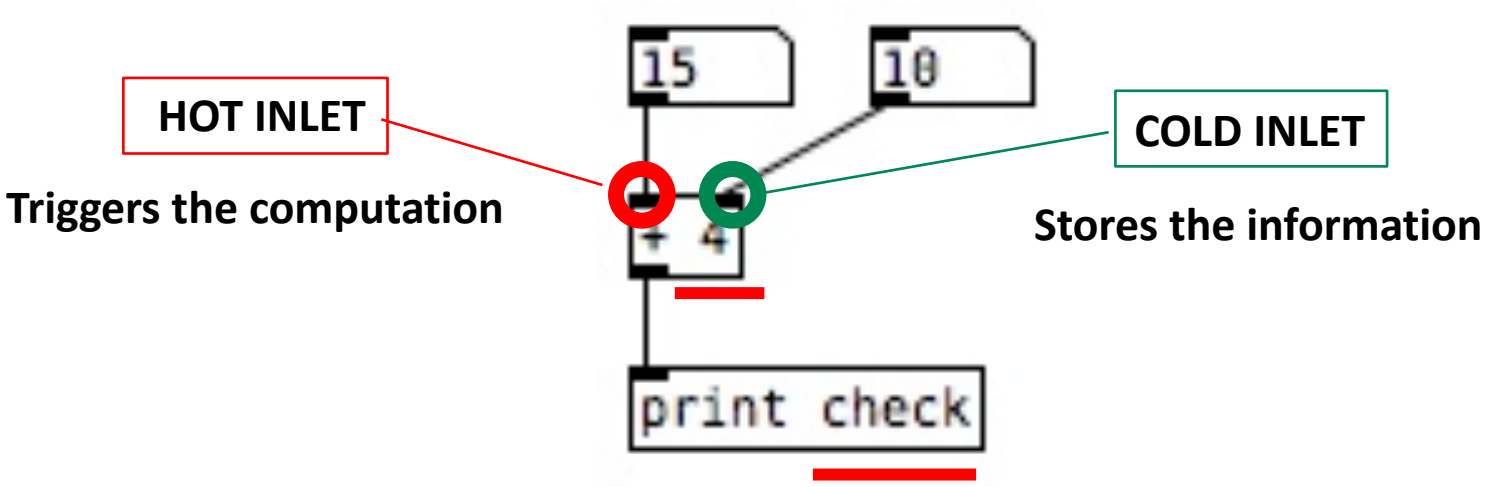


Hot inlet is always the left-most one

A bang message can allow to trigger computation (without change)

Cold vs. hot inlets

Different types of inlets exist in Pure Data



Hot inlet is always the left-most one

A bang message can allow to trigger computation (without change)

NB : Also note the difference in argument change or use

Fibonnaci in PD

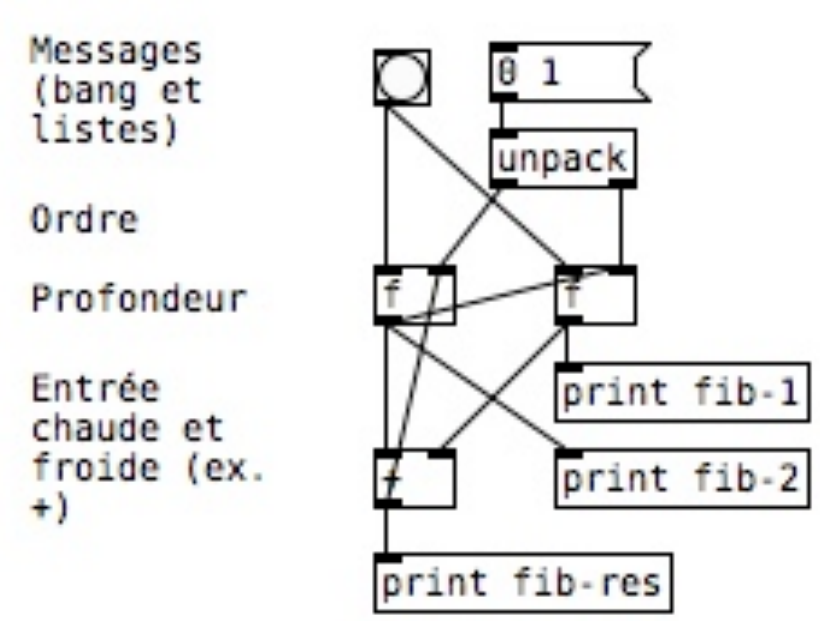
Exercice : Code the Fibonnaci sequence in PD

- At every bang, your patch prints the next number in the Fibonnaci sequence
- *Tip : the [f] object stores a number*
- *Tip : lists and [unpack] objects ?*

Fibonnaci in PD

Exercice : Code the Fibonnaci sequence in PD

- At every bang, your patch prints the next number in the Fibonnaci sequence
- *Tip : the [f] object stores a number*
- *Tip : lists and [unpack] objects ?*

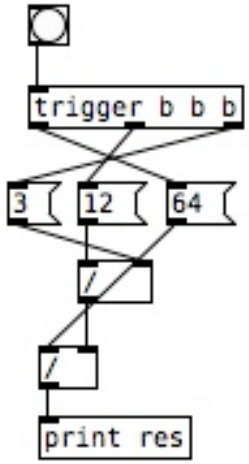


Ordering problem ?

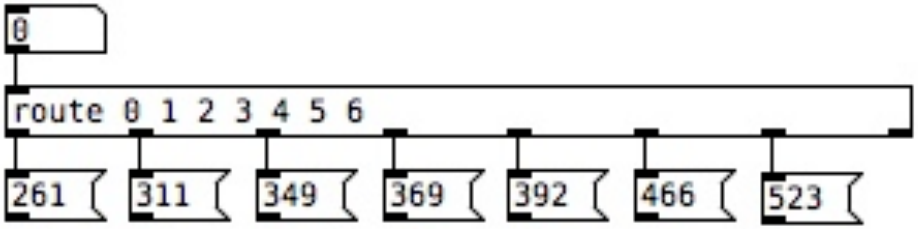
ALWAYS REMEMBER : RIGHT TO LEFT and DEPTH-WISE

Controlling the order (to avoid cold vs. hot inlet problem)

[trigger] object sends bangs from right to left



Use the **[route]** object to send bang depending on input number



Objects in PD (signal)

[osc~] – Sinusoidal oscillator

[dac~] – Audio out (stereo)

[line~] – Ramp generator

[lop~], [hip~] – Low-pass and high-pass filters

[bp~], [vcf~] – Band-pass filters

[noise~] – White noise generator

[phasor~] – Sawtooth generator

[send~], [receive~] – Signal distribution

Audio synthesis in PD exercise

First patches in PD to perform simple audio synthesis

1. First perform a simple pitch-controled sinusoid
   1. Create a simple oscillator
   2. Control its amplitude (volume)
   3. Control its frequency

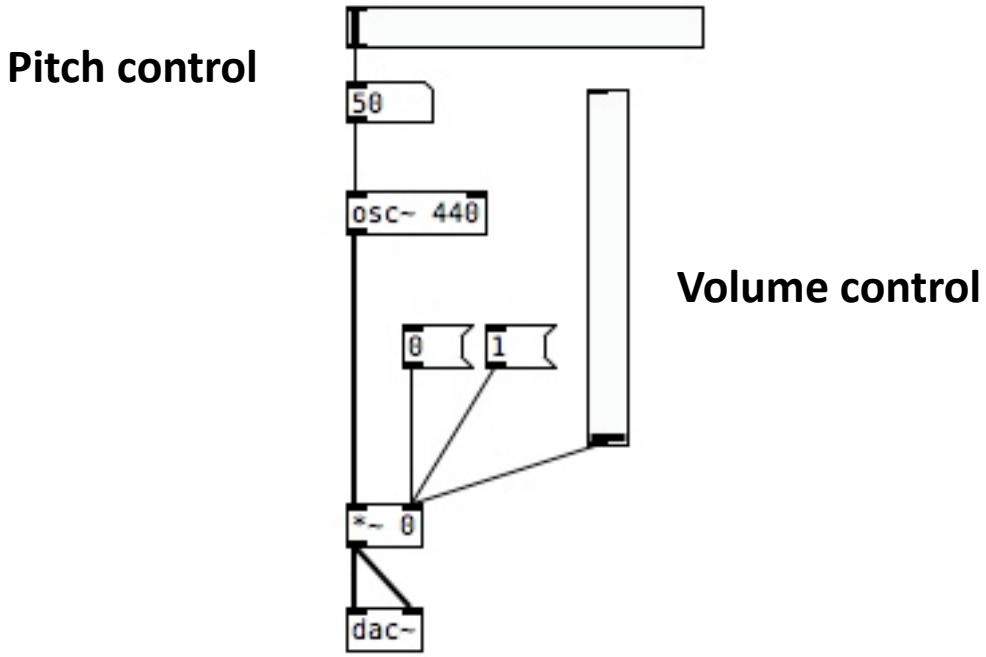
   Useful objects : **[osc~] [*~] [dac~] [hslider] [vslider]**
2. Then complexify with FM synthesis and display
   1. Create a FM synthesizer (carrier / modulator)
   2. Control the amount of modulation
   3. Store values inside a table
   4. Display the table on screen
   5. (Optionnal) Add AM to the mix

   More useful objects (with previous) : **[metro] [table] [tabwrite~]**

Audio synthesis in PD exercise

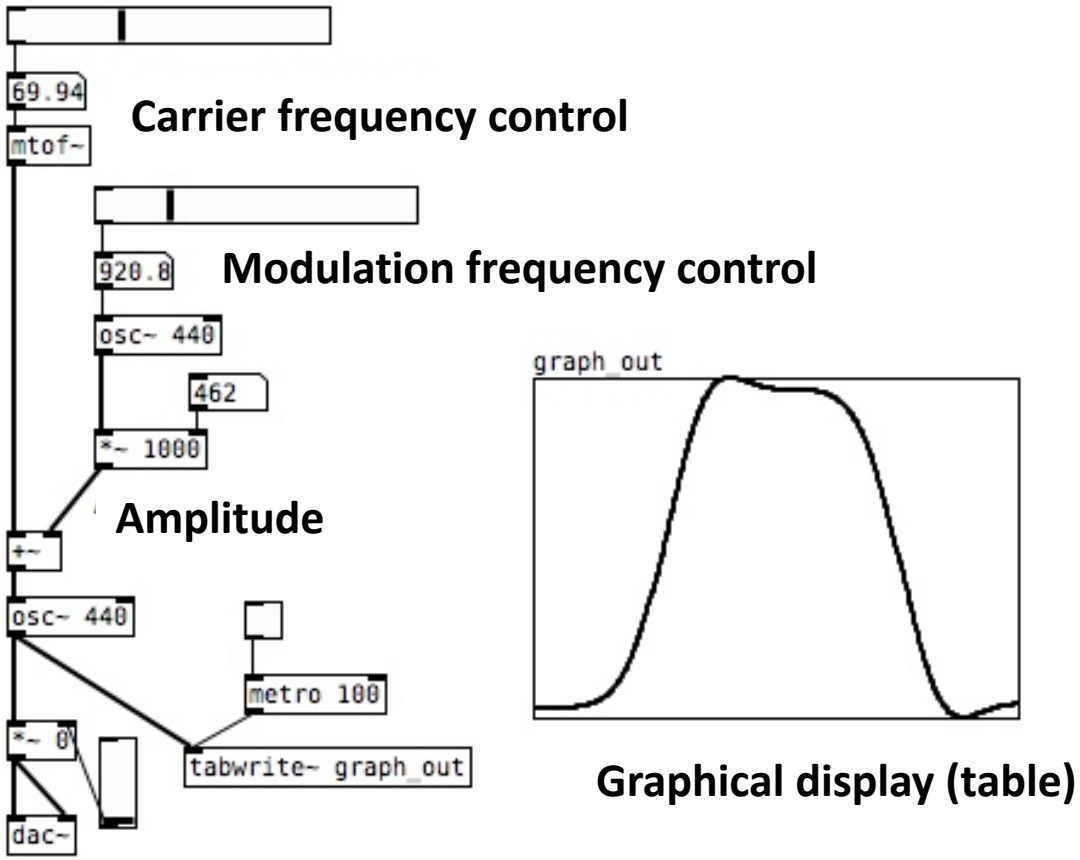
First patch in PD to perform audio synthesis

- First perform a simple pitch-controled sinusoid



Audio synthesis in PD exercise

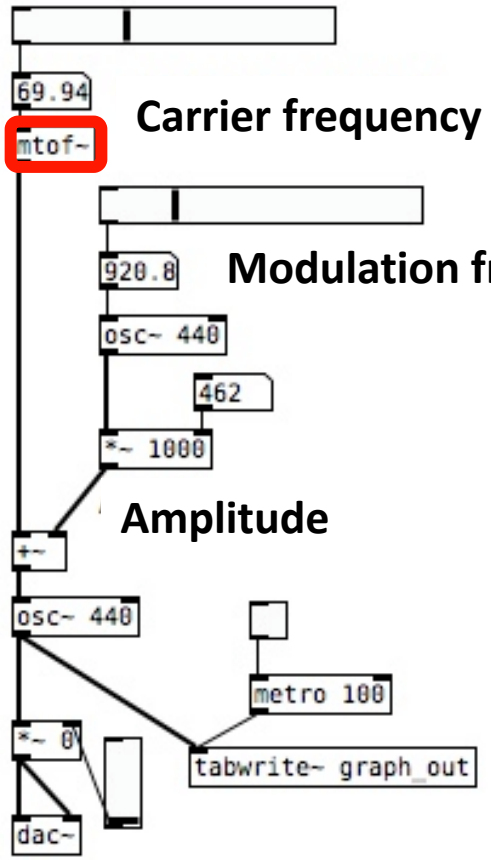
Then perform a FM synth and display waveform

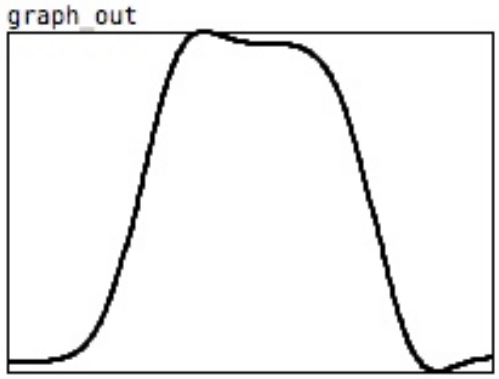


Audio synthesis in PD exercise

Then perform a FM synth and display waveform

We need to turn a number into a stream (+ MIDI to Hz)



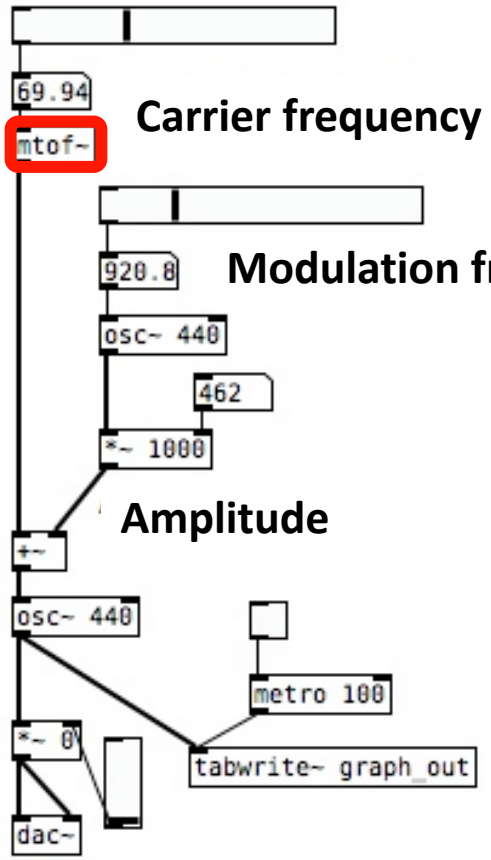


Graphical display (table)

Audio synthesis in PD exercise

Then perform a FM synth and display waveform

We need to turn a number into a stream (+ MIDI to Hz)



Name of the table

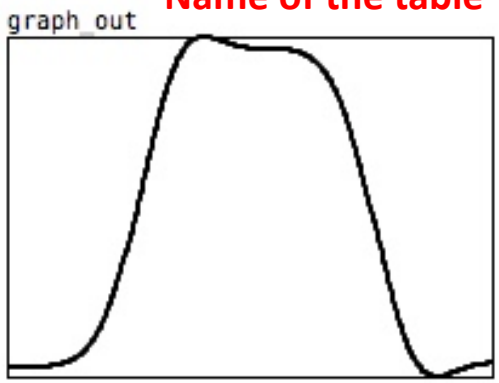


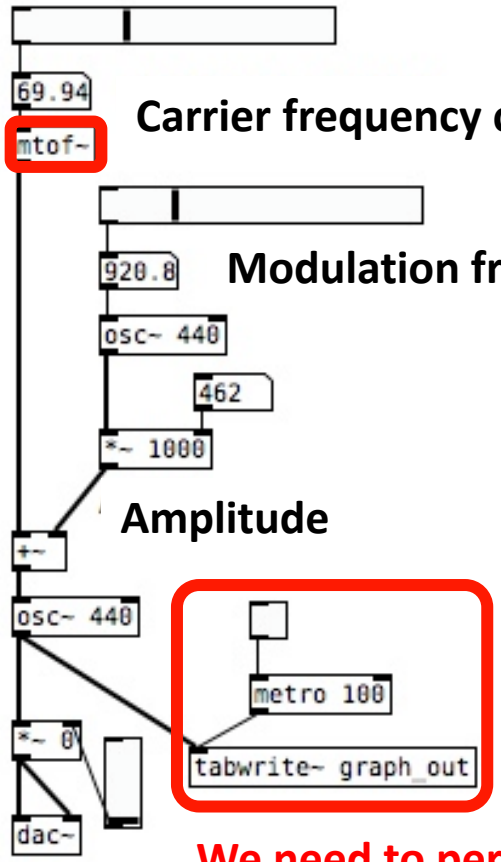
Table objects allow to store and display

Graphical display (table)

Audio synthesis in PD exercise

Then perform a FM synth and display waveform

We need to turn a number into a stream (+ MIDI to Hz)



Name of the table

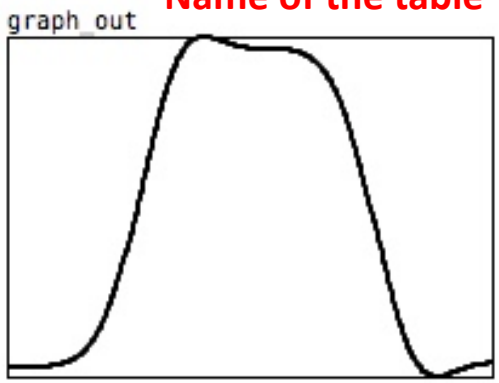
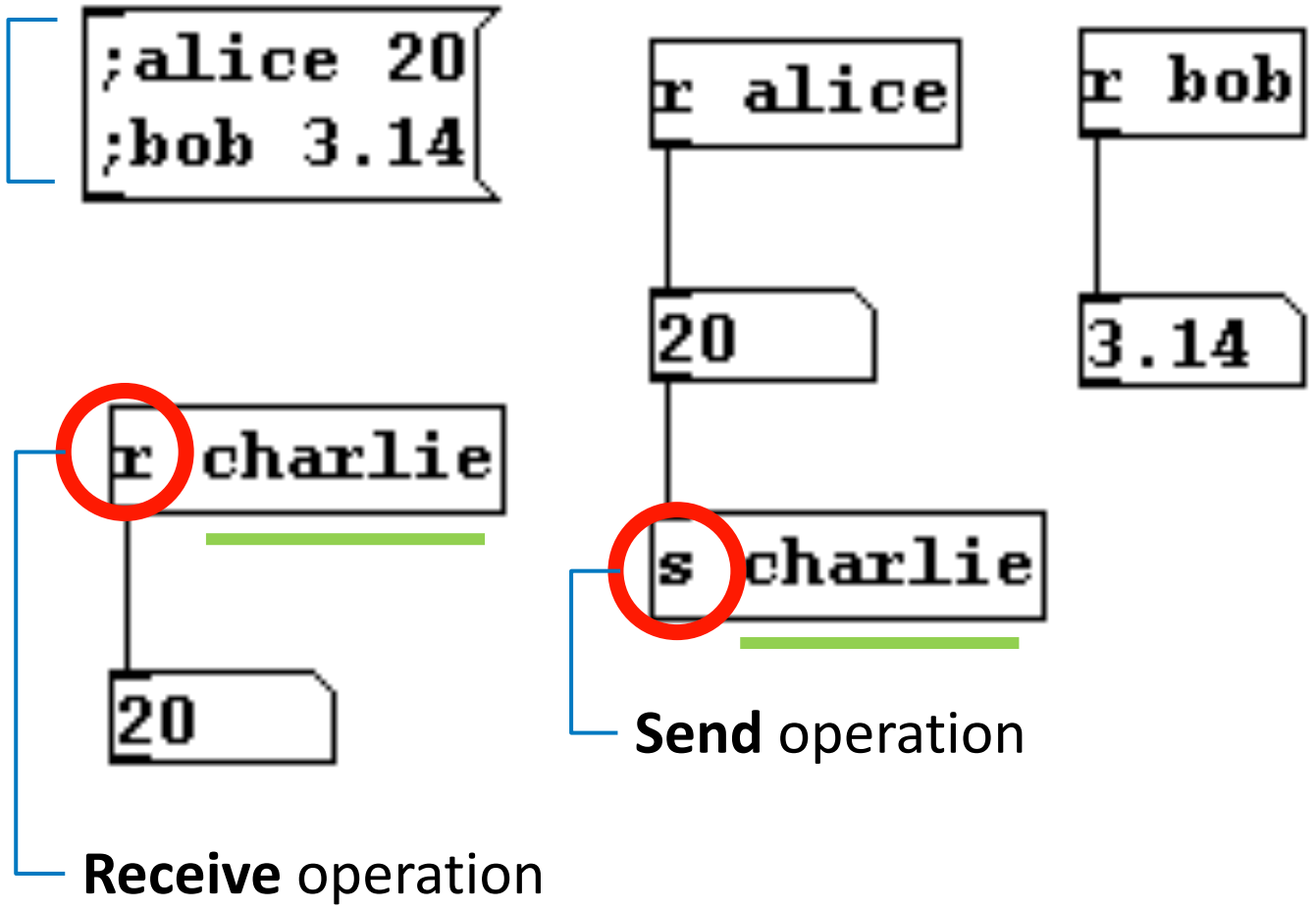


Table objects allow to store and display

Graphical display (table)

We need to periodically store information to the table
[metro 100] sends a bang every 100ms (if activated)

Sending messages (without cables)

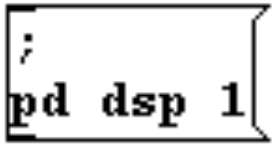


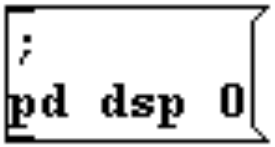
Important things

Right click on object to get Help

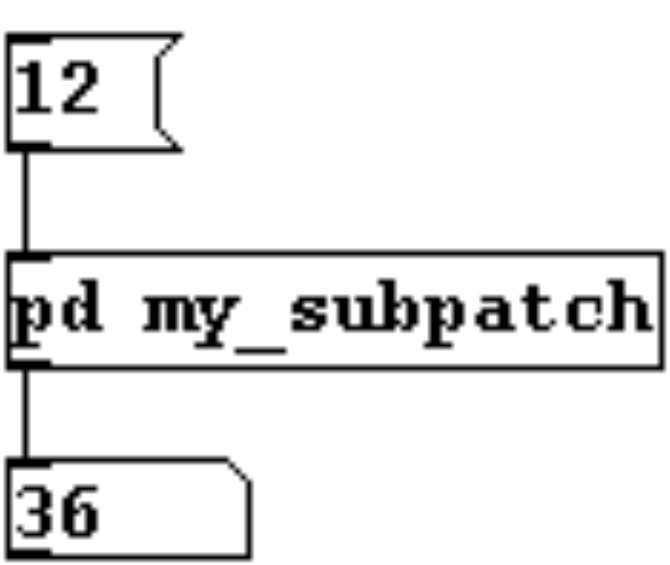
Ctrl-E to move in and out of “Edit Mode”

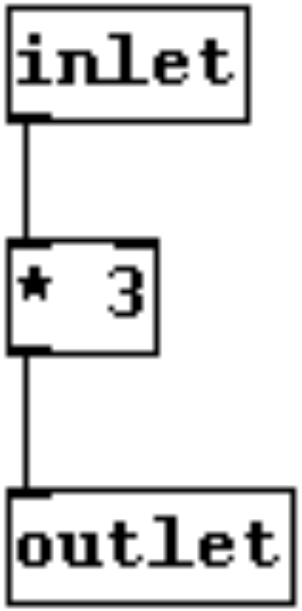
Turn audio on and off in a patch with these messages:





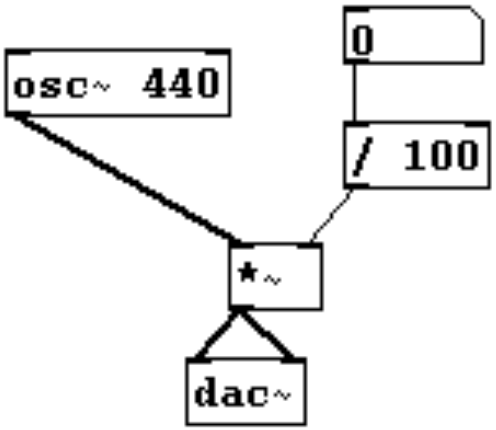
Subpatches in PD



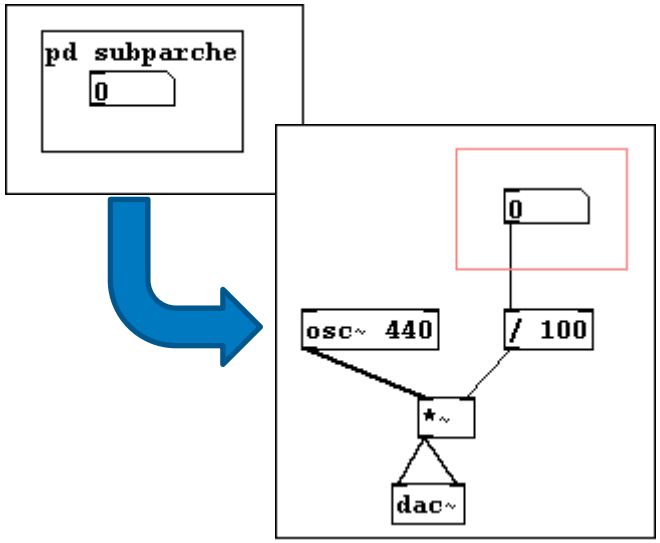


Subpatches in PD

Patches contain elements or objects in PD that are interconnected. This might lead to very heavy and messy schemes.



Sub-patches can embed patches inside other patches. (using [inlet] and [outlet] helps)



Abstraction are independent patches that need to be used inside patches

Objects in PD (control / time)

[metro] – Metronome information (bang)

[trigger] – Sending message in given order

[delay] – Delaying messages by a given time

[random] – Random number generator

[select] – Selector of input

[mtof] – Convert MIDI to Hz

[loadbang] – Send a bang message when loading

[send], [receive] – Message distribution

Advanced audio synthesis in PD

Create a patch for random melody generation

1. Perform additive synthesis (multiple of F0)
2. Randomly selects a pitch
3. Randomly selects a time duration
4. Pitch and time are changed in “rhythm”

Exercise: German techno generator.

Augment the previously defined patch so that

1. The pitch played are selected into a defined scale (array)
2. The duration of notes are selected inside a set of multiples (quarter, half)
3. A bassdrum (kick) is added at every beat
    1. No samples should be used
    2. Purely synthesized kick

Useful objects: **metro, random**, **route, trigger, line~**

Objects in PD (calculus)

[+], [-], [*], [/], [%] – Messages arithmetics
[sin], [cos], [tan], [atan], [atan2] - Trigonometric
[sqrt], [log], [exp], [abs], [clip] – Other functions
[expr] – Arbitrary expressions evaluation

Audio versions
[+~], [-~], [*~], [/~] – Signal arithmetics
[cos~] – Cosinusoidal function
[sqrt~], [rsqrt~] – Square root and reciprocal
[max~], [min~] – Maximum and minimum
[clip~] – Clipping operation
[expr~], [fexpr~] – Arbitrary expressions

Objects in PD (arrays)

[table] – Define a subpatch with an array

[tabread] – Reading an array

[tabwrite] – Writing an array

[tabread4~] – Continuous read and interpolation

[tabwrite~] – Continuous writing of an array

[tabosc4~] – Cyclical reading of an array

Objects in PD (subpatches)

[pd] – Create a sub-patch inside a patch

[inlet] – Adding a control inlet into a subpatch

[inlet~] – Create an audio inlet into a subpatch

[outlet] – Create a control outlet from a subpatch

[outlet~] – Agrega una salida de audio

Advanced audio sampler and effects

Home exercise: Loop sampler and audio effects

1. Create a system to load a sample (.wav, .aiff) file
2. Store it inside an array
3. Display the content of the file
4. Allow to play it as a loop
5. Control start and end points of loop
6. Add effects to the system
    1. Delay line (through buffer)
        1. Ms. Control
        2. % Control
    2. Clipping distortion
        1. Clip amount
        2. Global volume
    3. Whatever kicks your bucket

A quick intro to externals

- So what is going on inside a given box ?
- In fact we can even go deeper in Pure Data objects

A quick intro to externals

- So what is going on inside a given box ?
- In fact we can even go deeper in Pure Data objects
- Possibility to *define your own* boxes ☺
- The overall system defines **PD *externals***
- Here we will code in C but still talk about *objects*

A quick intro to externals

- So what is going on inside a given box ?
- In fact we can even go deeper in Pure Data objects
- Possibility to *define your own* boxes ☺
- The overall system defines **PD *externals***
- Here we will code in C but still talk about *objects*
- PD provides a set of *includes* and *specs*
  - Simple SDK with a clear notation
  - Entirely dynamic linking
  - Runtime class loading

A quick intro to externals

- So what is going on inside a given box ?
- In fact we can even go deeper in Pure Data objects
- Possibility to *define your own* boxes ☺
- The overall system defines **PD *externals***
- Here we will code in C but still talk about *objects*
- PD provides a set of *includes* and *specs*
  - Simple SDK with a clear notation
  - Entirely dynamic linking
  - Runtime class loading
- Everything defined as a C struct
- Then simply a set of functions

Defining your own external

- First we define the header **horloge.h**

```
#ifndef _HORLOGE_H_
# define _HORLOGE_H_

# include "m_pd.h"
```

We need to **include the** PD header definitions

Defining your own external

- First we define the header **horloge.h**

```
#ifndef _HORLOGE_H_
# define _HORLOGE_H_

# include "m_pd.h"

static t_class      *horloge_class;
typedef struct      _horloge
{
    t_object        x_obj;
    t_outlet        *h_out;
}
t_horloge;
```

We need to **include the** PD header definitions

Then we define the **class of our object**

This object is **mandatory in all objects** (cf. later)

We need to **manually define inlets and outlets**

Defining your own external

- First we define the header **horloge.h**

```
#ifndef _HORLOGE_H_
# define _HORLOGE_H_

# include "m_pd.h"

static t_class      *horloge_class;
typedef struct      _horloge
{
    t_object        x_obj;
    t_outlet        *h_out;
}
t_horloge;
```

We need to **include the** PD header definitions

Then we define the **class of our object**

This object is **mandatory in all objects** (cf. later)

We need to **manually define inlets and outlets**

All objects have a default left-most hot inlet

Defining your own external

- First we define the header **horloge.h**

```
#ifndef _HORLOGE_H_
# define _HORLOGE_H_

# include "m_pd.h"

static t_class      *horloge_class;
typedef struct      _horloge
{
    t_object        x_obj;
    t_outlet        *h_out;
}
t_horloge;
```

We need to **include the** PD header definitions

Then we define the **class of our object**

This object is **mandatory in all objects** (cf. later)

We need to **manually define inlets and outlets**

All objects have a default left-most hot inlet

```
/*
 * Q.2 - Chargement en mémoire des objets de type horloge
 */
void                horloge_setup(void);

#endif
```

1. Define what happens at **runtime load (once)**

Defining your own external

- First we define the header **horloge.h**

```
#ifndef _HORLOGE_H_
# define _HORLOGE_H_

# include "m_pd.h"

static t_class      *horloge_class;
typedef struct      _horloge
{
    t_object        x_obj;
    t_outlet        *h_out;
}
t_horloge;
```

We need to **include the** PD header definitions

Then we define the **class of our object**

This object is **mandatory in all objects** (cf. later)

We need to **manually define inlets and outlets**

All objects have a default left-most hot inlet

```
/*
 * Q.3 - Création d'un nouvel objet horloge
 */
void                *horloge_new(void);

/*
 * Q.2 - Chargement en mémoire des objets de type horloge
 */
void                horloge_setup(void);

#endif
```

2. Defines what happens at **each object creation**

1. Define what happens at **runtime load (once)**

Defining your own external

- First we define the header **horloge.h**

```
#ifndef _HORLOGE_H_
# define _HORLOGE_H_

# include "m_pd.h"

static t_class      *horloge_class;
typedef struct      _horloge
{
    t_object        x_obj;
    t_outlet        *h_out;
}
t_horloge;

/*
 * Q.4 - Comportement de l'objet en cas de message bang
 */
void                horloge_bang(t_horloge *x);

/*
 * Q.3 - Création d'un nouvel objet horloge
 */
void                *horloge_new(void);

/*
 * Q.2 - Chargement en mémoire des objets de type horloge
 */
void                horloge_setup(void);

#endif
```

We need to **include the** PD header definitions

Then we define the **class of our object**

This object is **mandatory in all objects** (cf. later)

We need to **manually define inlets and outlets**

All objects have a default left-most hot inlet

3. Define **one method per message received**
Here we code what happens when bang received

2. Defines what happens at **each object creation**

1. Define what happens at **runtime load (once)**

Implement your external (horloge.c)

1. Define what happens at **runtime load (once)**

```
void            horloge_setup(void)
{
    horloge_class = class_new(gensym("horloge"),
                              (t_newmethod)horloge_new,
                              0, sizeof(t_horloge),
                              CLASS_DEFAULT, 0);
    class_addbang(horloge_class, horloge_bang);
}
```

Class creation method class_new

Name of the object
Method to call for each new object
Size / malloc options

Add the behavior for bang with class_addbang
Later we will also use class_admethod (messages)

Reminder of the data structure

```
static t_class      *horloge_class;
typedef struct      _horloge
{
    t_object        x_obj;
    t_outlet        *h_out;
}
t_horloge;
```

Implement your external (horloge.c)

2. Defines what happens at **each object creation**

```
void            *horloge_new(void)
{
    t_horloge   *h;

    h = (t_horloge *)pd_new(horloge_class);
    h->h_out = outlet_new(&h->x_obj, &s_symbol);
    return (void *)h;
}
```

Instanciation method pd_new

Returned object (void *)

Create the (symbol) outlet and store in x_obj !

Return the created object

Reminder of the data structure

```
static t_class      *horloge_class;
typedef struct      _horloge
{
    t_object        x_obj;
    t_outlet        *h_out;
}
t_horloge;
```

Implement your external (horloge.c)

3. Define **the method for bang behavior**

```
void            horloge_bang(t_horloge *x)
{
    time_t      rawtime;
    struct tm   *timeinfo;

    time(&rawtime);
    timeinfo = localtime(&rawtime);
    outlet_symbol(x->h_out, gensym(asctime(timeinfo)));
}
```

Specific object instance

(Here just a bunch of C code to get the time)

Need to write symbols to a given symbol table !

Write information to a specific outlet

Reminder of the data structure

```
static t_class      *horloge_class;
typedef struct      _horloge
{
    t_object        x_obj;
    t_outlet        *h_out;
}
t_horloge;
```

External exercises

- Check the online subject
- We will code 3 mandatory externals
  - Horloge
  - Multipouet
  - Duck~
- Then one more advanced external
  - My_fft~
- Then optional externals for the geekiest
  - Reactive cross-synthesis instruments
  - Spectral cross-synthesis (cross~)
  - Convolutive cross-synthesis (xcross~)
- **Questions ????**