OPAL-RT TECHNOLOGIES
FROM IMAGINATION TO REAL-TIME

RT-LAB Solution for Real-Time Applications

OP101 : Getting started

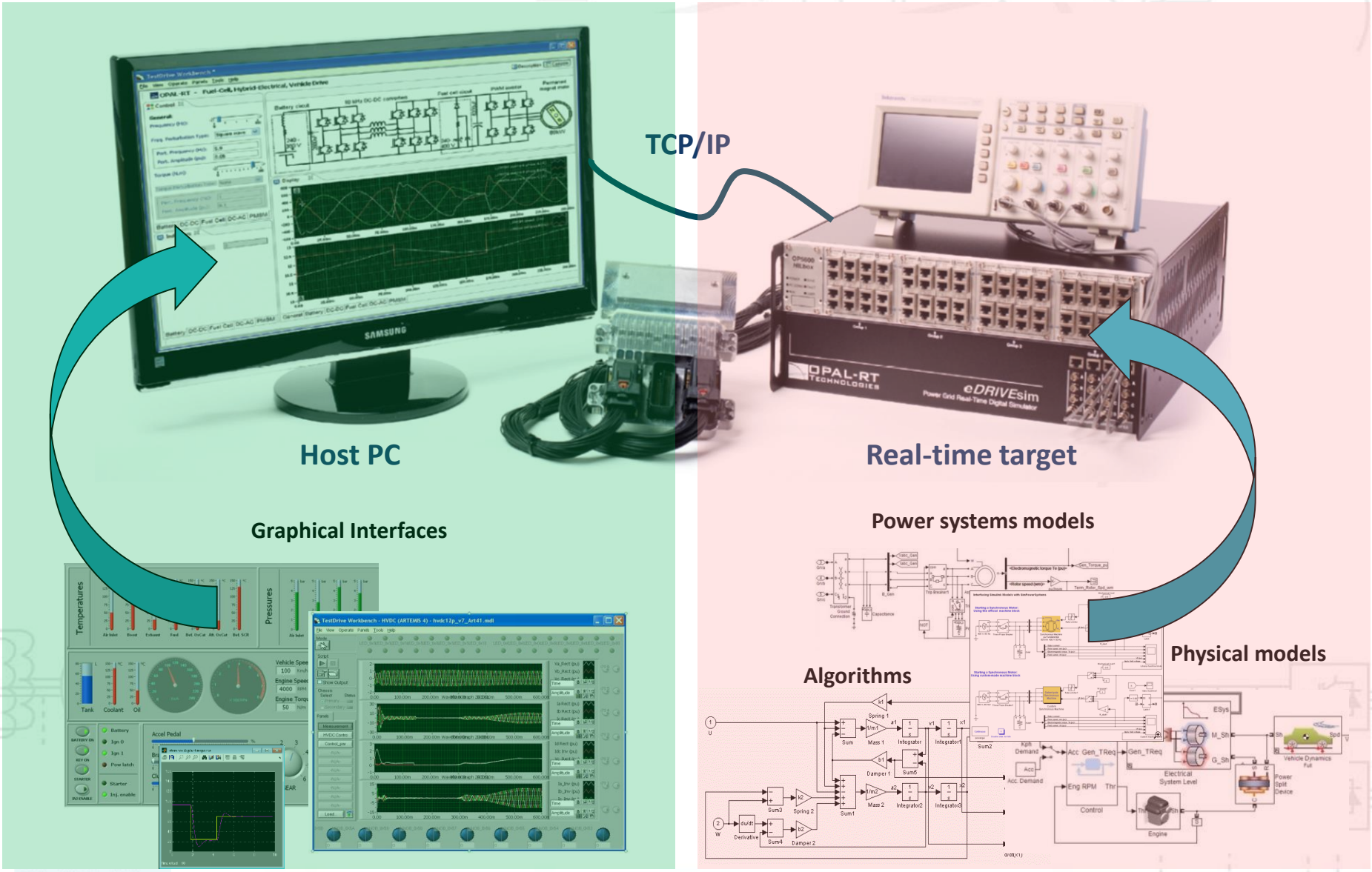
Modeling Concepts in Simulink®

Training Services

Outline

1. **General concepts**
2. Grouping into subsystems
3. Adding OpComm blocks
4. Maximizing parallel execution
5. Setting simulation parameters
6. Executing off-line

General concepts



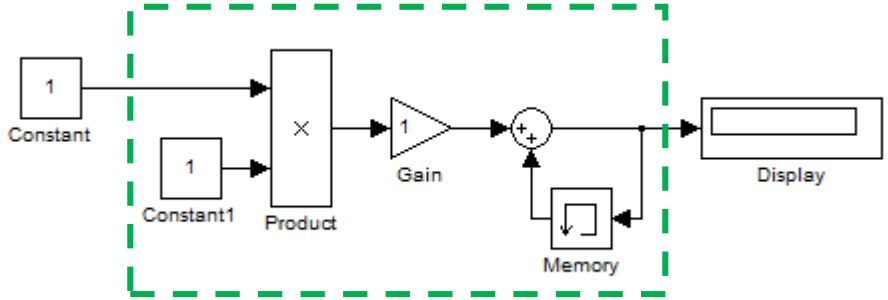
Outline

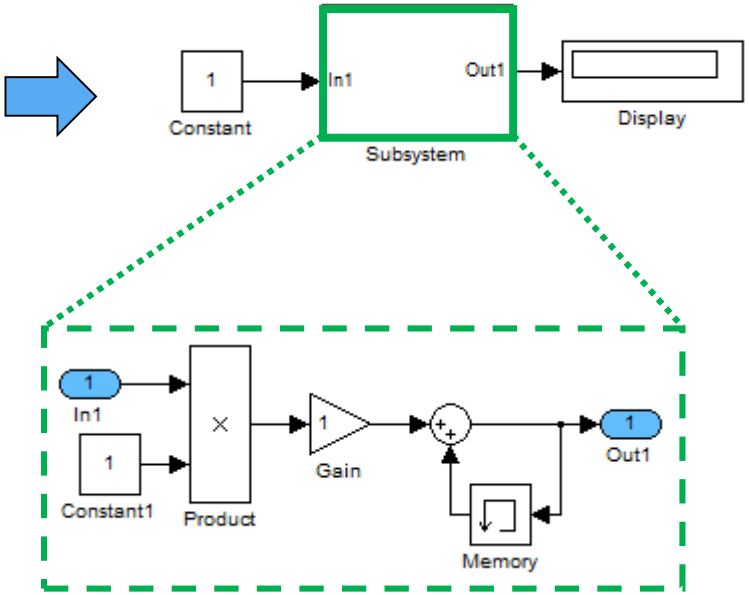
1. General concepts
2. **Grouping into subsystems**
3. Adding OpComm blocks
4. Maximizing parallel execution
5. Setting simulation parameters
6. Executing off-line

Grouping into subsystems

Concept

What is a ***subsystem*** in Simulink® ?





A set of blocks that are placed inside one single block called “Subsystem”

- Simplify the model by grouping blocks
- Establish hierarchical block diagram
- Keep functionally related blocks together

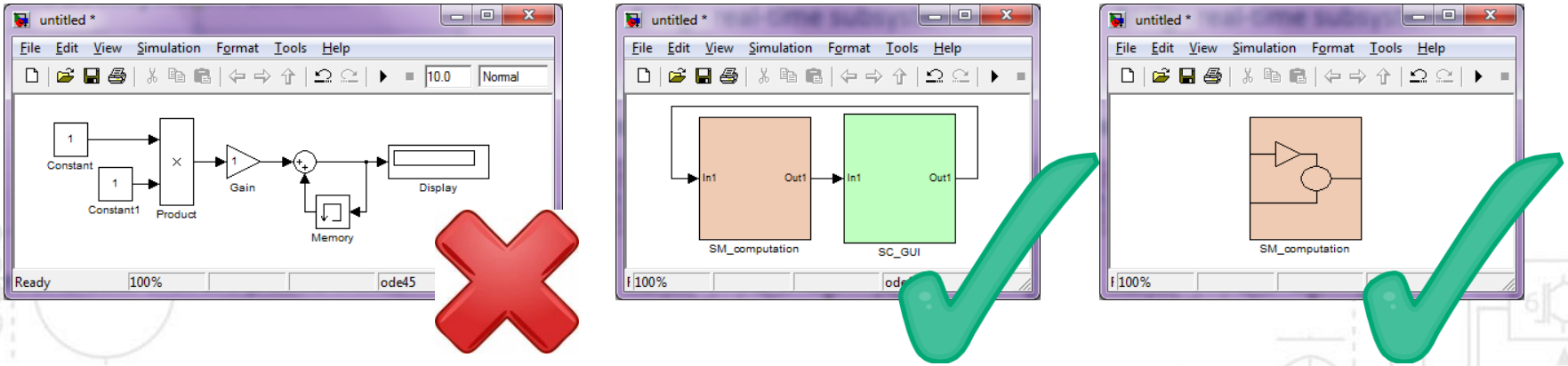
Grouping into subsystems

Concept

In RT-LAB platforms, ***subsystems*** have 2 objectives:

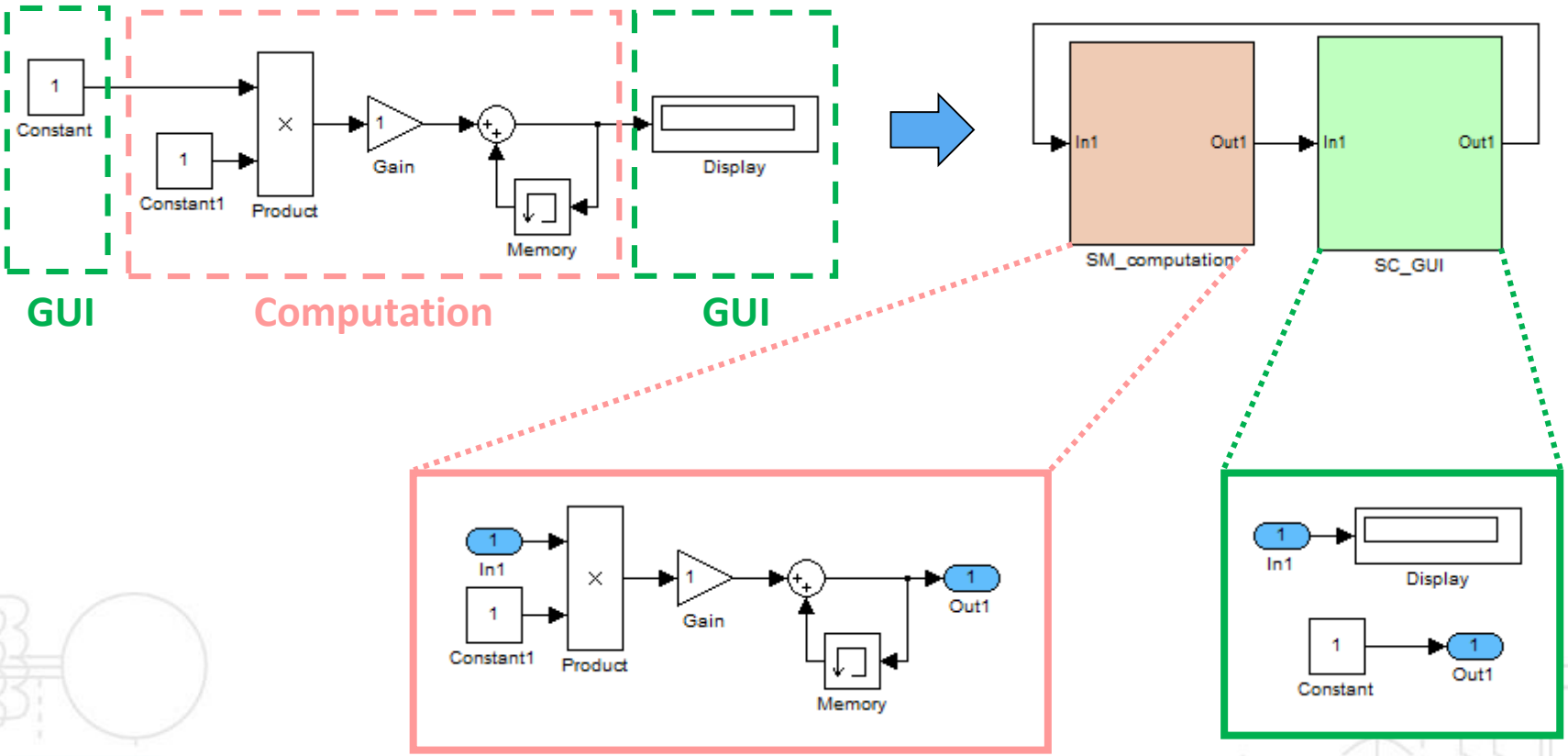
1. Distinguish computation subsystems and GUI subsystem
2. Assign computation subsystems to different CPU cores

Anyway, the top-level of a Simulink® model used with RT-LAB must only display ***subsystems***.



Grouping into subsystems

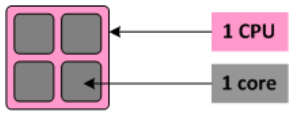
1. Distinguish real-time subsystems and GUI subsystem

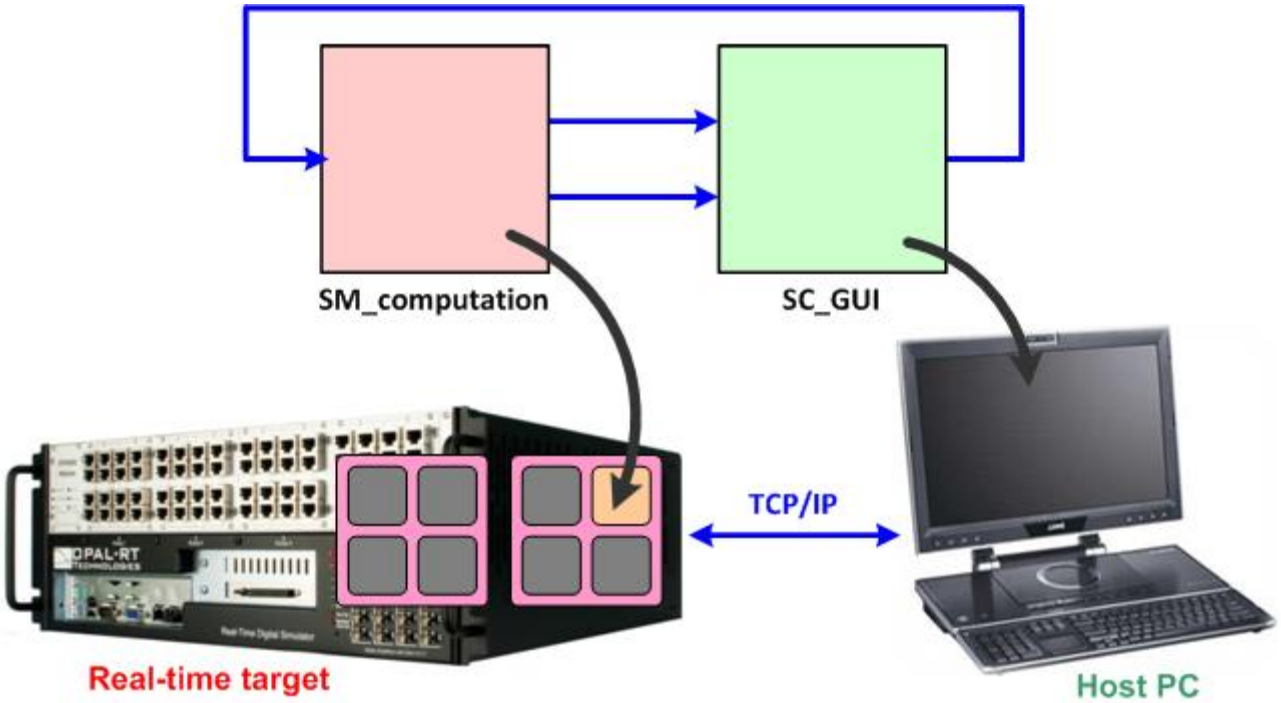


Grouping into subsystems

1. Distinguish real-time subsystems and GUI subsystem

- The computation subsystem will be executed in real-time (or accelerated simulation mode) on **one CPU core** of the real-time target
- The GUI subsystem will be displayed on the Host PC
- The data between computation subsystem and GUI subsystem is exchanged **asynchronously** through the **TCP/IP link**

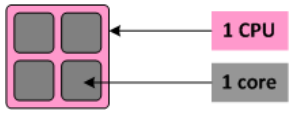


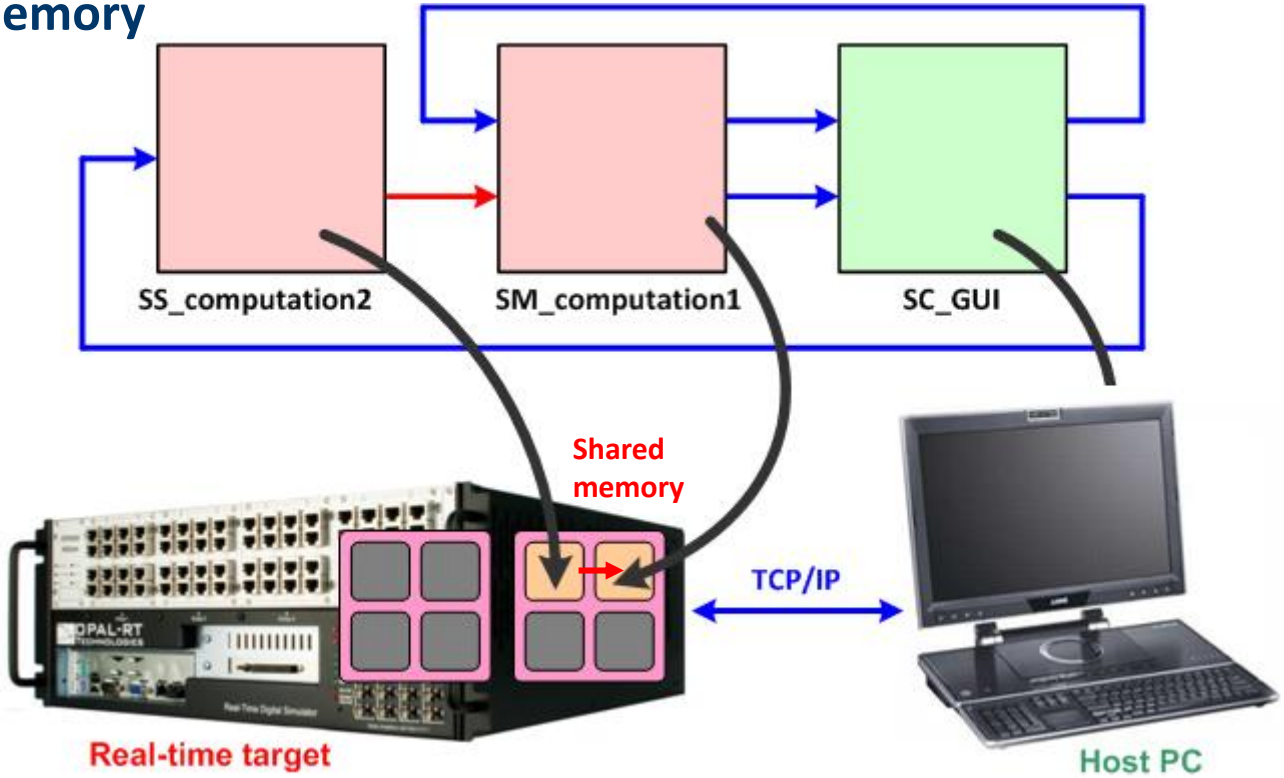


Grouping into subsystems

2. Assign real-time subsystems to different CPU cores

- The computation blocks can be split into different computation subsystems
- Each of the computation subsystems will be executed on one CPU core of the real-time target
- The data between 2 computation subsystems is exchanged **synchronously** through **shared memory**





Grouping into subsystems

Naming the subsystems

- **GUI subsystem** : **SC_anyName**
  - Allows interaction with the computation subsystems
  - Runs on host PC asynchronously from the computation subsystems
  - Not linked to a target CPU core
  - Contains user interface blocks (scopes, displays, switches, constants)
  - No signal generation/No mathematical operations/No physical model !

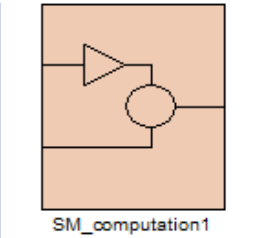
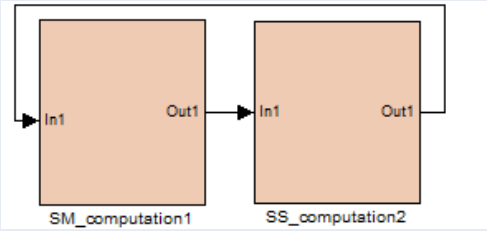
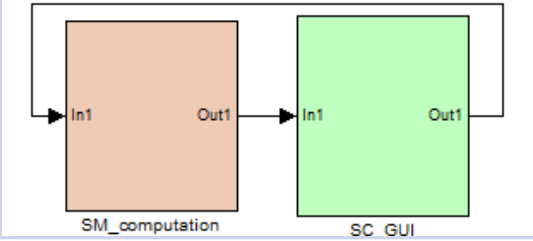
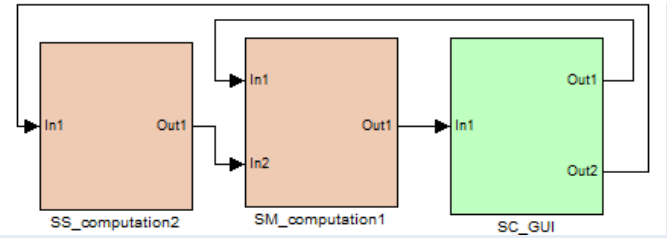
- **Computation subsystems**
  - All the computational elements of the model, mathematical operations, I/O blocks, signal generators, physical models, etc…

➔ If only one (main computation subsystem) : **SM_anyName**
  - Uses one CPU core

➔ Each additional computation subsystem : **SS_anyName**
  - Each additional SS subsystem uses an additional CPU core

Grouping into subsystems

Possible configurations

SM	SS	SC	Model
1			
1	n		
1		1	
1	n	1	

Grouping into subsystems

Conclusion

- Only ***subsystems*** are allowed at the top-level of the Simulink® model
- **SC_** subsystems are used as graphical interface
- **SM_** and **SS_** subsystems are used for computation
- One computation subsystem is executed on one CPU core
- Communication between computation subsystems is synchronous
- Communication between computation subsystems and GUI subsystem is asynchronous
- Each signal between subsystems("wire" in Simulink®) can be a **scalar** (single value) or a **vector** (multiple values), but it must be of type **double**

Outline

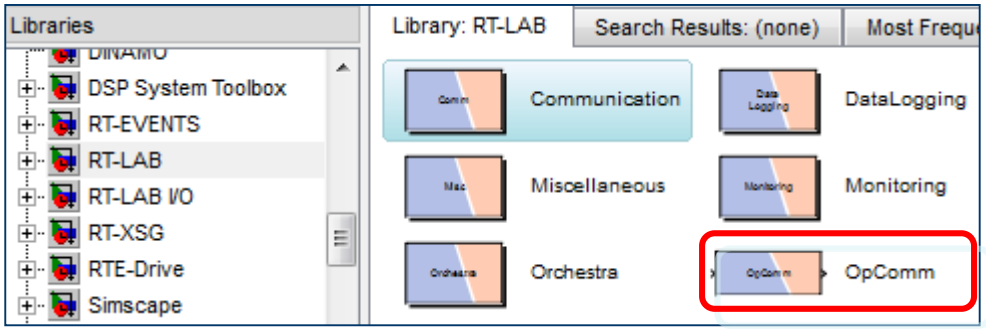
1. General concepts
2. Grouping into subsystems
3. **Adding OpComm blocks**
4. Maximizing parallel execution
5. Setting simulation parameters
6. Executing off-line

Adding OpComm blocks

What is an OpComm block ?

- Responsible for the communication
  - Between 2 computation subsystems
  - Between computation subsystems and GUI subsystem

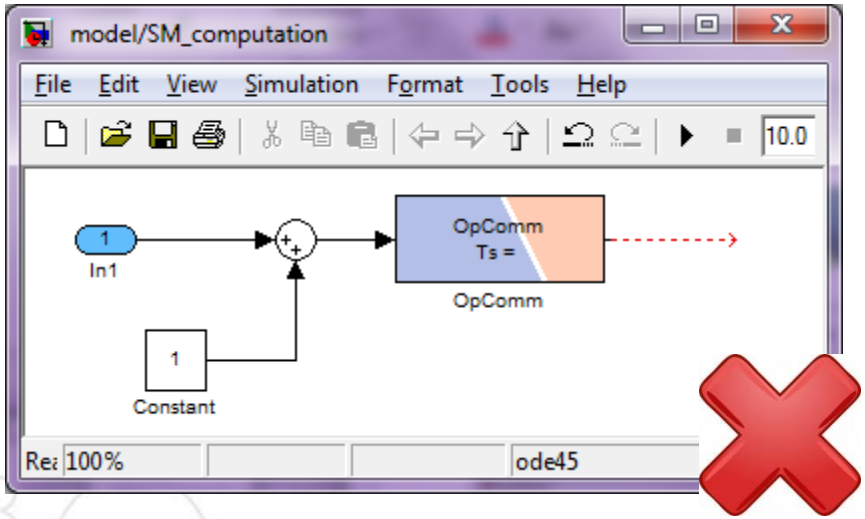
- **OpComm** block can be found in the RT-LAB library, in Simulink® library browser, once RT-LAB has been installed

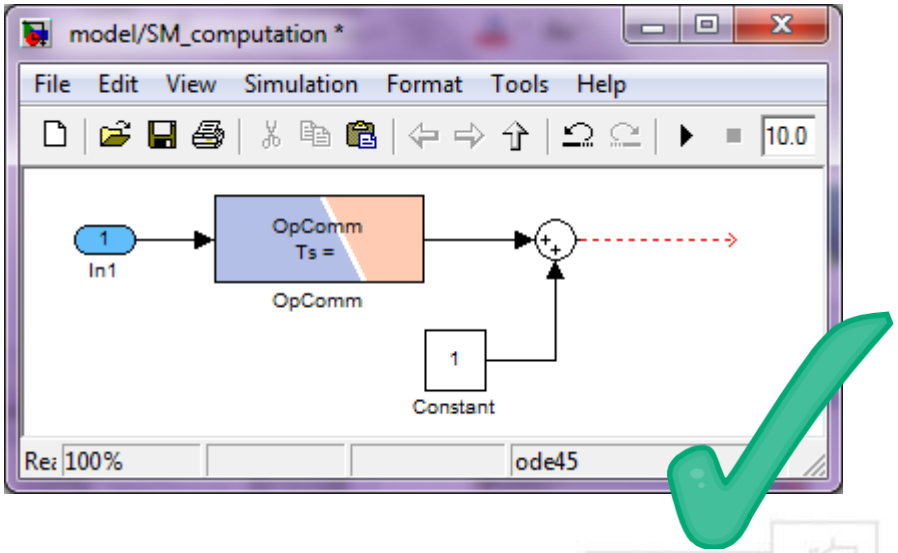


Adding OpComm blocks

How to place OpComm blocks in the model ?

- All subsystems **inputs** must first go through an OpComm block before any operations can be done on the signals they are associated with.





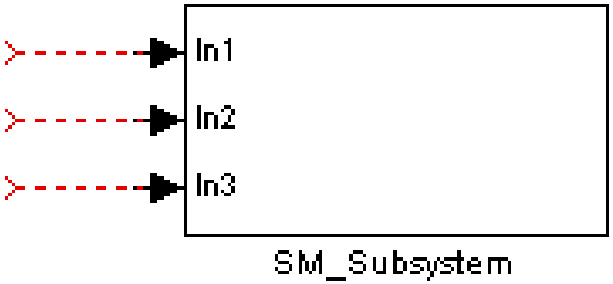
Adding OpComm blocks

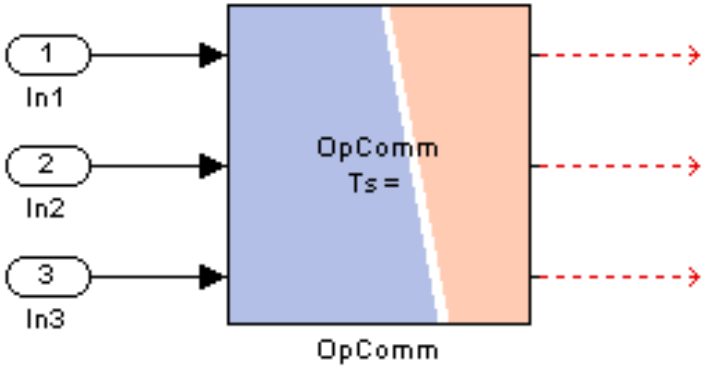
How to place OpComm blocks in the model ?

- The **OpComm** block must be inserted **after** the subsystems creation and renaming (SM, SS, SC)

- One **OpComm** block can accept multiple inputs in one subsystem. Double-click on the block to select the number of inputs required

- Each input signal can be a scalar or a vector





Adding OpComm blocks

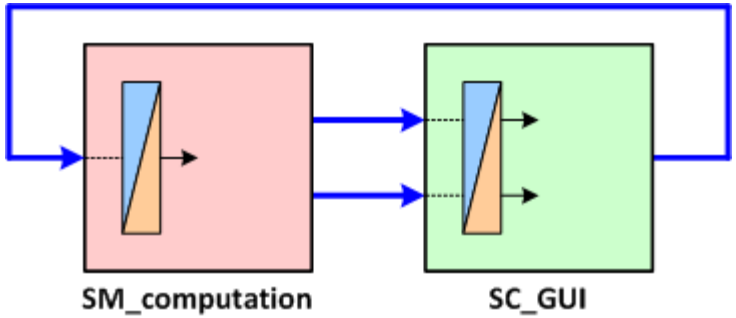
How to place OpComm blocks in the model ?

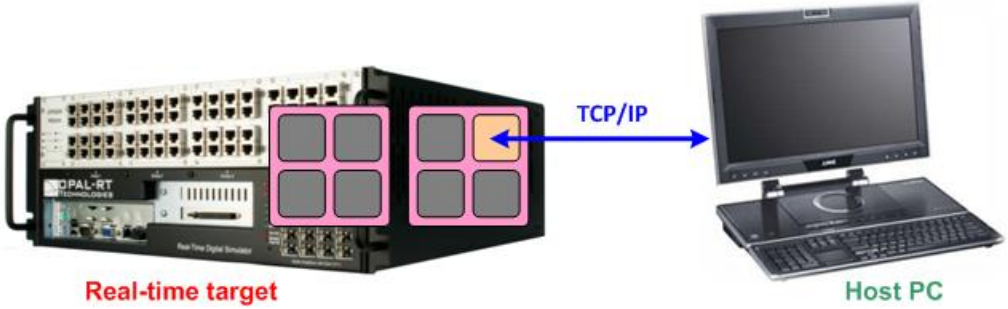
- **In the computation subsystems (SM or SS):**
  - One **OpComm** receives real-time-synchronized signals from other real-time subsystems
  - One **OpComm** receives asynchronous signals from the GUI subsystem
- **In the console subsystem (SC):**
  - One **OpComm** is enough in most cases
  - More **OpComm** blocks (up to 25) may be inserted to receive signals from the real-time subsystems. Multiple **OpComm** blocks define unique "acquisition groups" with their own data acquisition parameters (decimation, frame size,…)

Adding OpComm blocks

How to place OpComm blocks in the model ?

- ***SM_computation*** only receives **asynchronous** signals from SC_GUI
  ➔ Only 1 **OpComm** block in ***SM_computation***

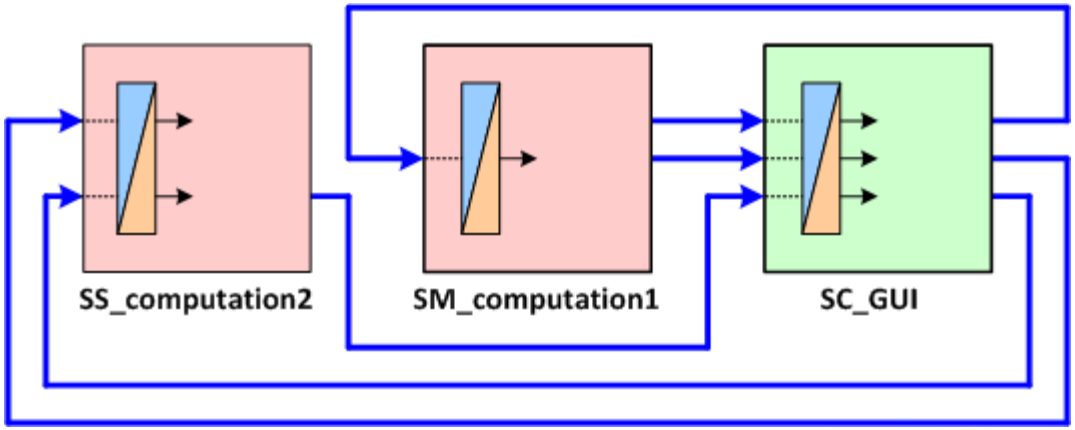


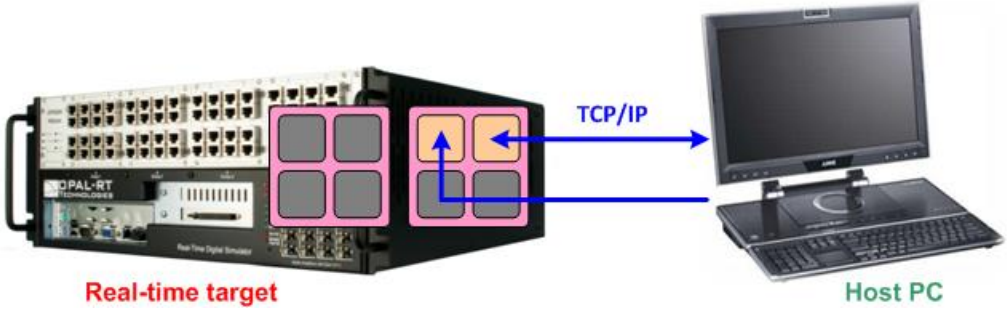


Adding OpComm blocks

How to place OpComm blocks in the model ?

- ***SM_computation1*** only receives **asynchronous** signals from ***SC_GUI***
  - ➔ Only 1 **OpComm** block in ***SM_computation1***
- ***SS_computation2*** only receives **asynchronous** signals from ***SC_GUI***
  - ➔ Only 1 **OpComm** block in ***SS_computation2***

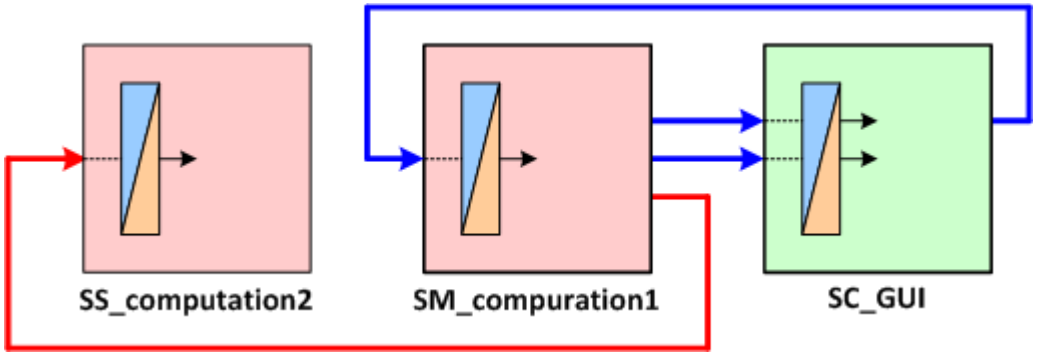


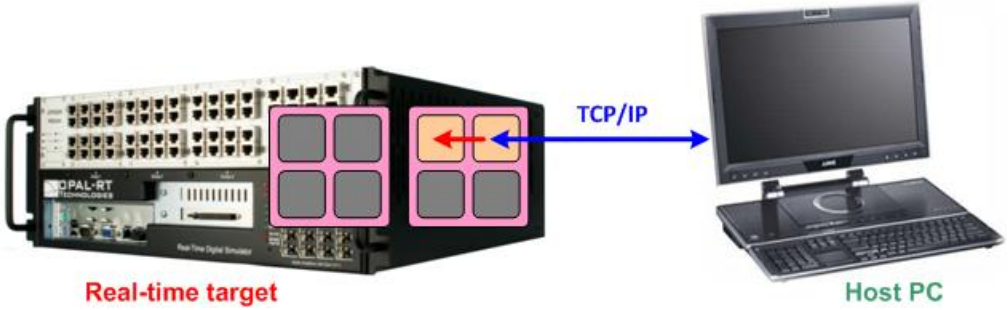


Adding OpComm blocks

How to place OpComm blocks in the model ?

- ***SM_computation1*** only receives **asynchronous** signals from ***SC_GUI***
  ➔ Only 1 **OpComm** block in SM
- ***SS_computation2*** only receives **synchronous** signals from ***SM_computation1***
  ➔ Only 1 **OpComm** block in SS

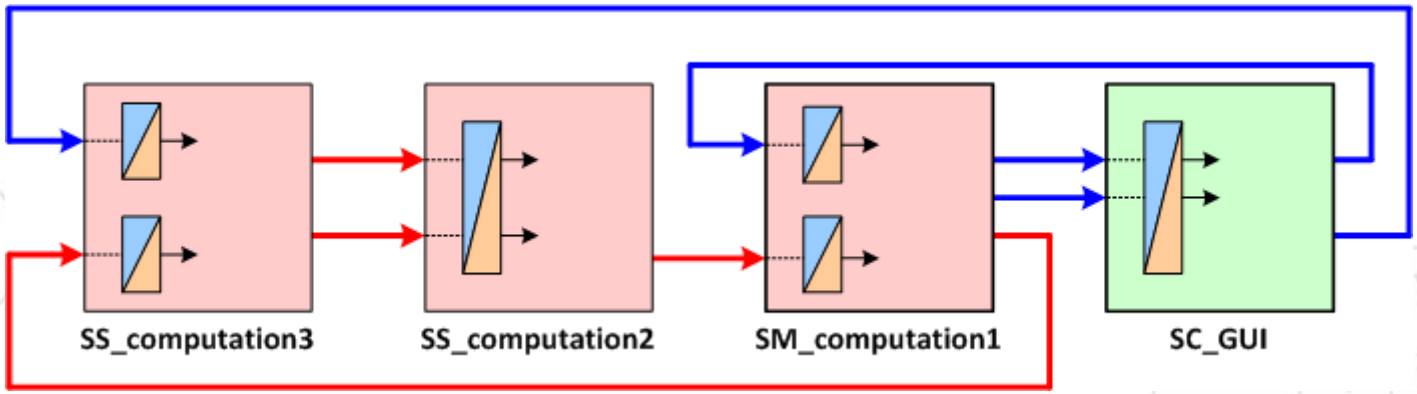




Adding OpComm blocks

How to place OpComm blocks in the model ?

- ***SM_computation1*** receives **asynchronous** signals from ***SC_GUI*** and **synchronous** signals from ***SS_computation2***
  - ➔ 2 **OpComm** blocks in ***SM_computation1***
- ***SS_computation2*** receives only **synchronous** signals from ***SS_computation3***
  - ➔ Only 1 **OpComm** block in ***SS_computation2***
- ***SS_computation3*** receives **asynchronous** signals from ***SC_GUI*** and **synchronous** signals from ***SM_computation1***
  - ➔ 2 **OpComm** blocks in ***SS_computation3***



Outline

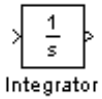
1. General concepts
2. Grouping into subsystems
3. Adding OpComm blocks
4. **Maximizing parallel execution**
5. Setting simulation parameters
6. Executing off-line

Maximizing parallel execution

- What is a state?

  ➔ A state can be defined as an output (signal) which is computed only from preceding inputs or outputs.

- Example of blocks which introduce a state are the “integrator” and the “memory” blocks.



$$y_z = y_{z-1} + x_{z-1}\Delta t$$

$$y_z = x_{z-1}$$

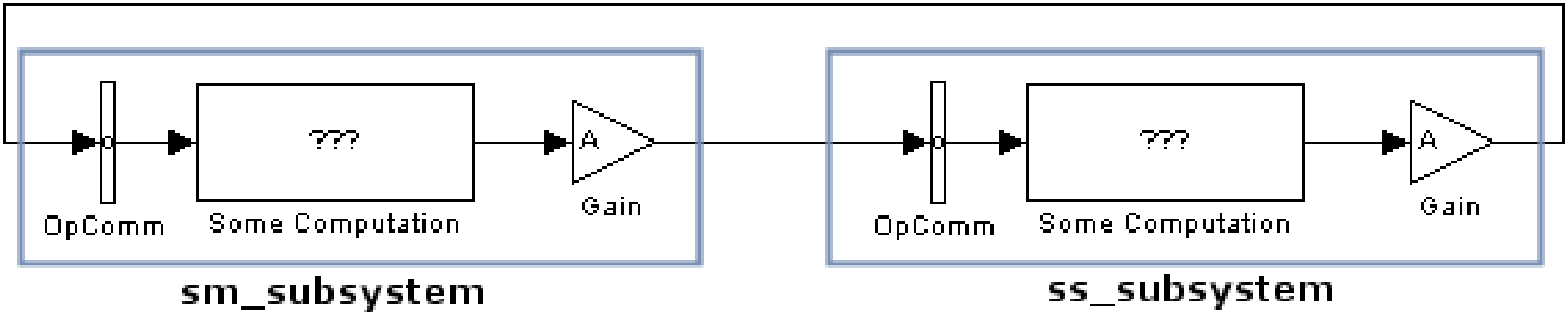
- A “gain” block does not produce a state because its output at step *z* depends on its input at the same step.

$$y_z = Ax_z$$

Maximizing parallel execution

Deadlock

- "Will not execute" case

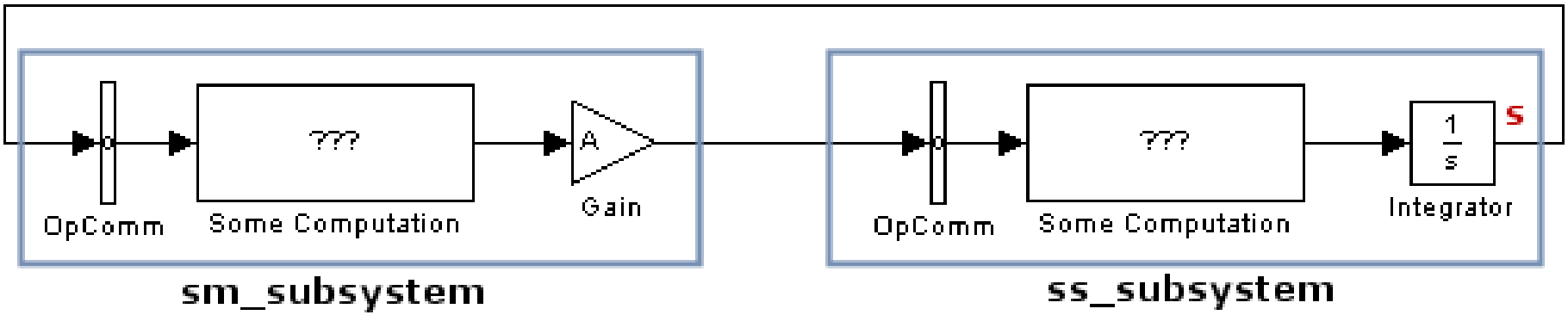


- RT-LAB is deadlocked!
  1. **sm_subsystem** waits for **ss_subsystem**
  2. **ss_subsystem** waits for **sm_subsystem**

Maximizing parallel execution

Serial execution

- Worst case

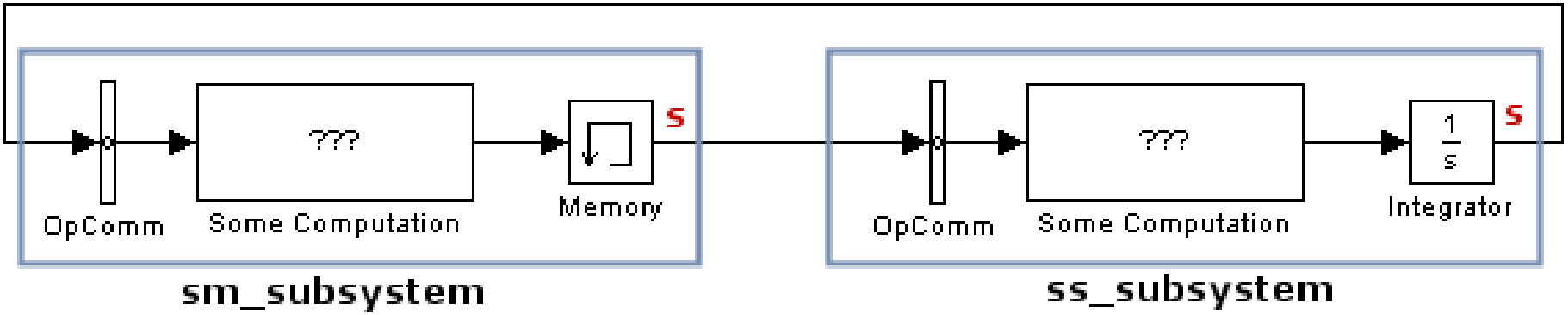


- At each step, RT-LAB does the following:
  1. **ss_subsystem** sends to **sm_subsystem**
  2. computation of **sm_subsystem**
  3. **sm_subsystem** sends to **ss_subsystem**
  4. computation of **ss_subsystem**

Maximizing parallel execution

Fully Parallel execution

- Best case

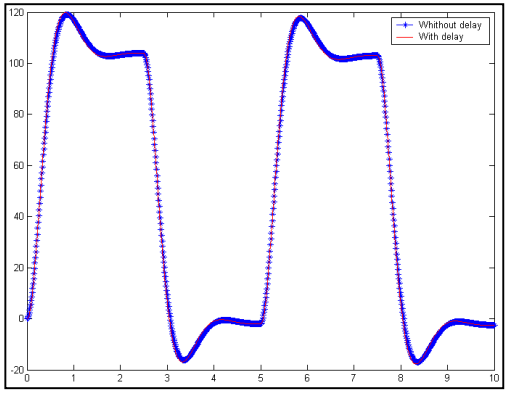


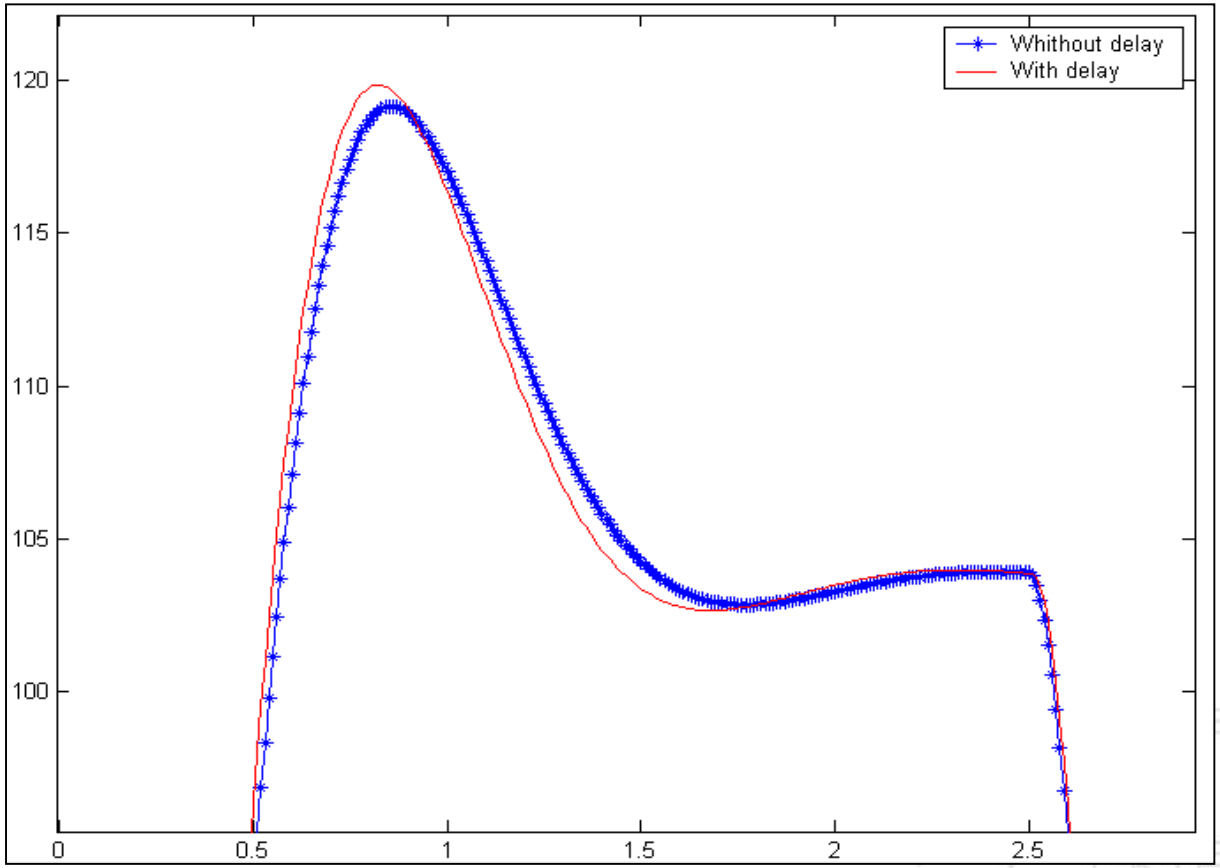
- At each step, RT-LAB does the following:
  1. **ss_subsystem** sends to **sm_subsystem**
  2. **sm_subsystem** sends to **ss_subsystem**
  3. computation of both subsystems at the same time

Maximizing parallel execution

Impact of the delay

- You must compare results before and after to make sure that the impact of the delay is acceptable.





- Is this acceptable? This is your call.

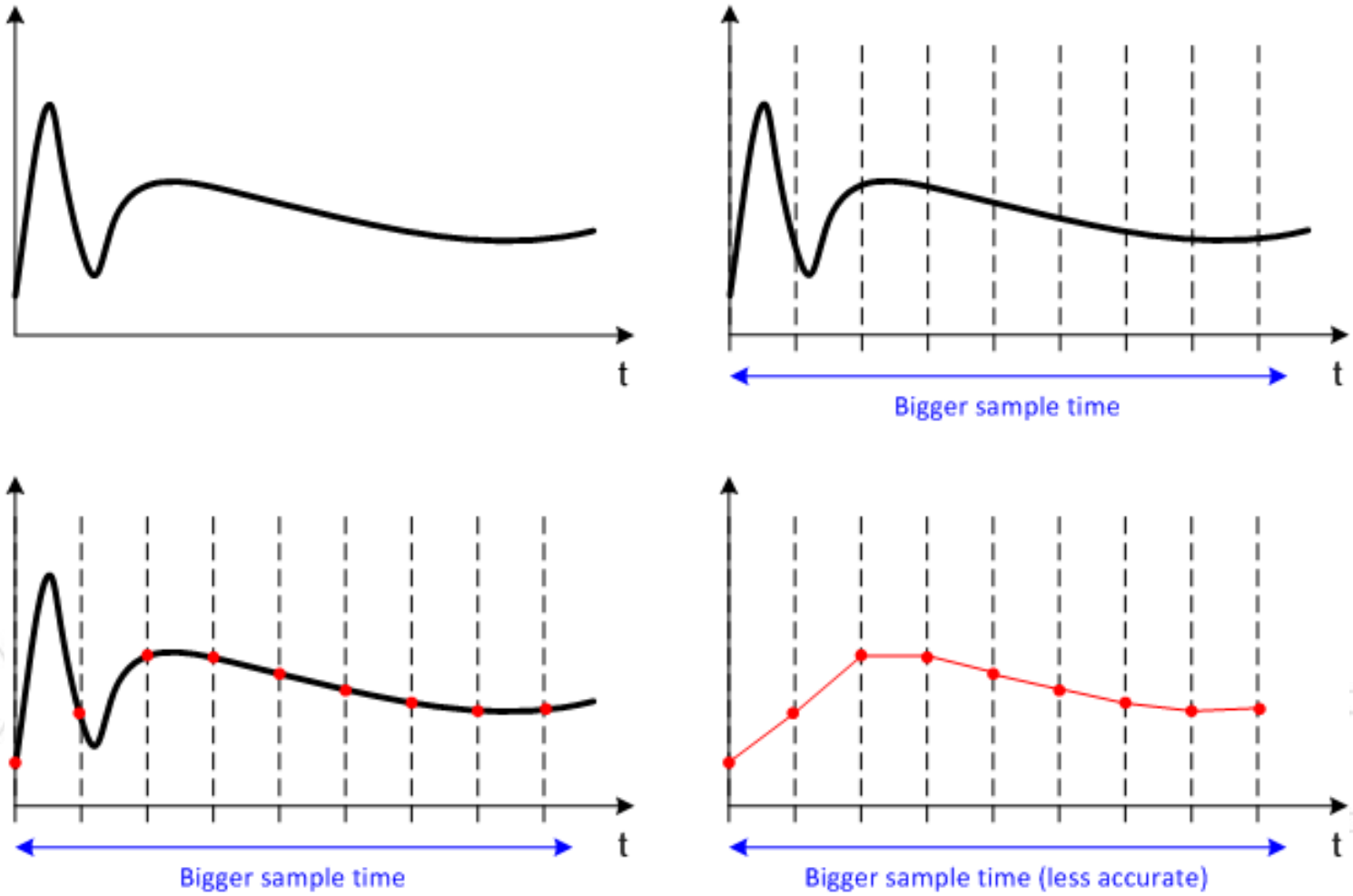
Outline

1. General concepts
2. Grouping into subsystems
3. Adding OpComm blocks
4. Maximizing parallel execution
5. **Setting simulation parameters**
6. Executing off-line

Setting simulation parameters

Fixed-step vs variable-step solvers

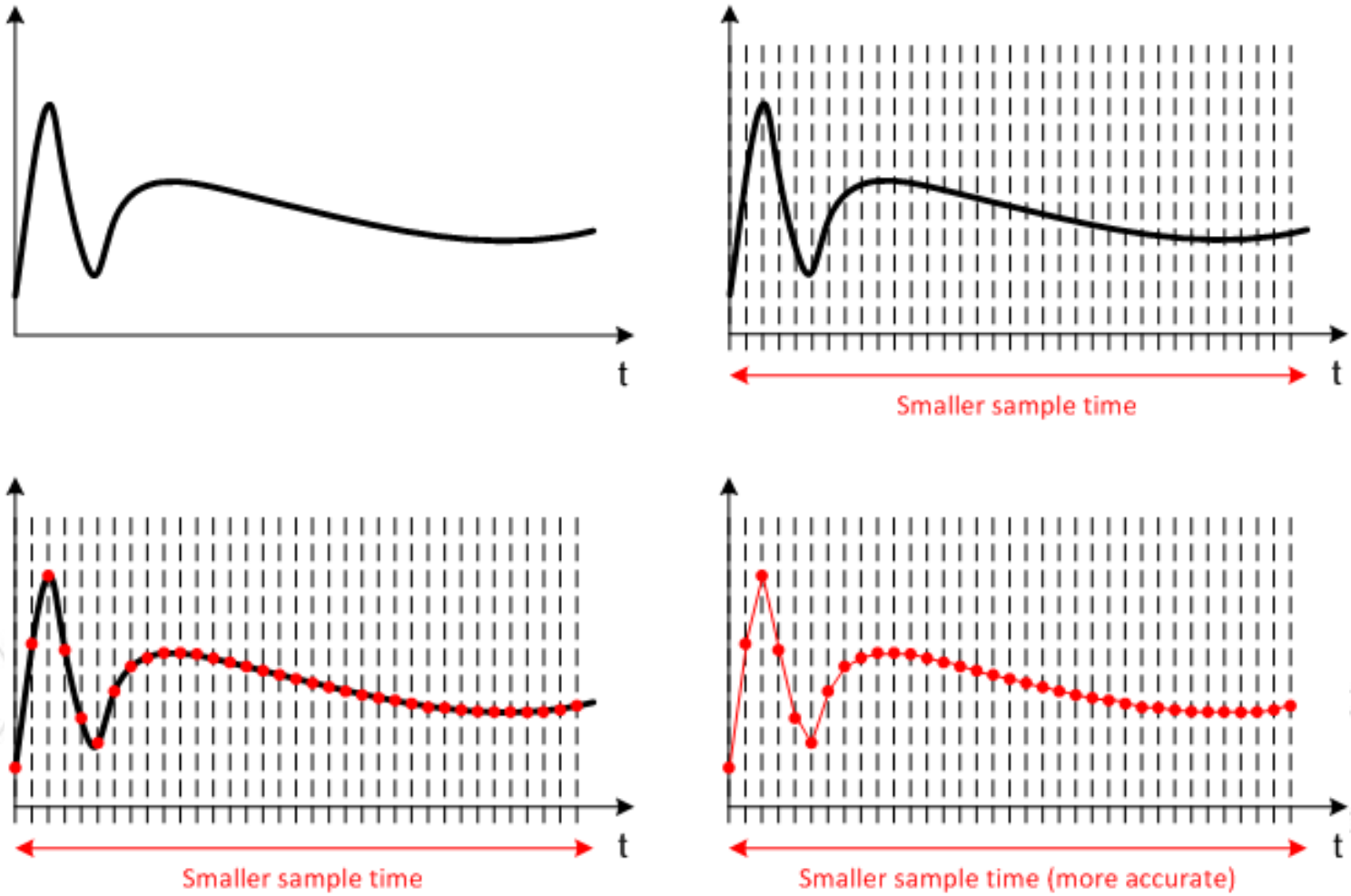
- Fixed-step solver



Setting simulation parameters

Fixed-step vs variable-step solvers

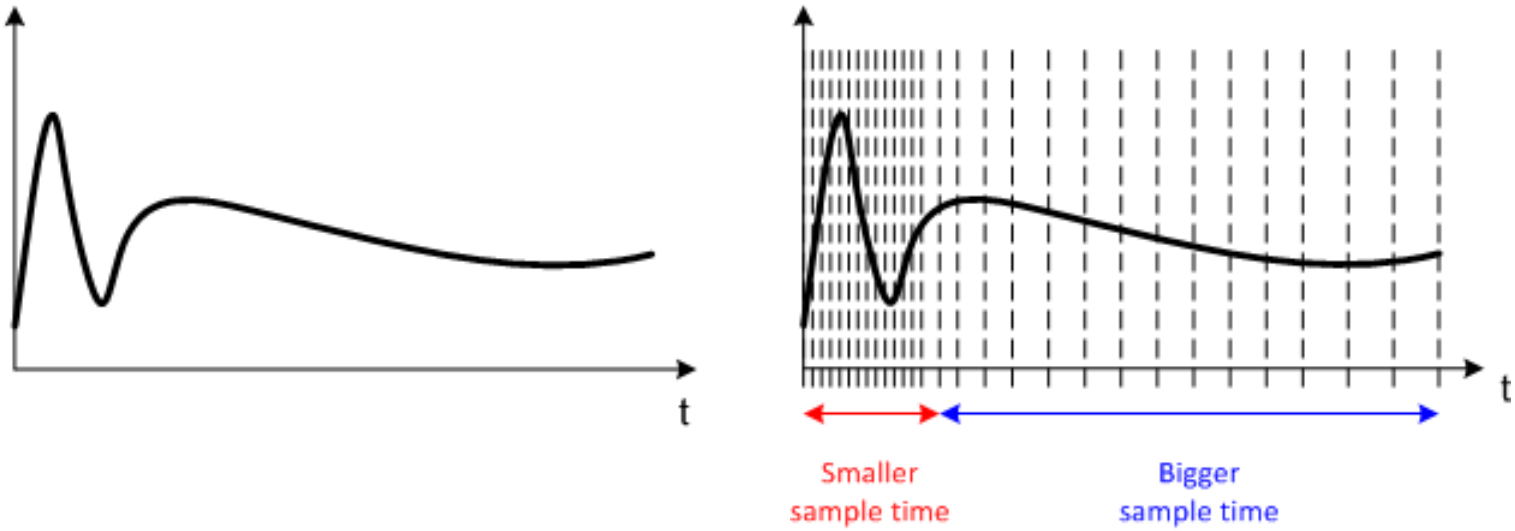
- Fixed-step solver



Setting simulation parameters

Fixed-step vs variable-step solvers

- Variable-step solver



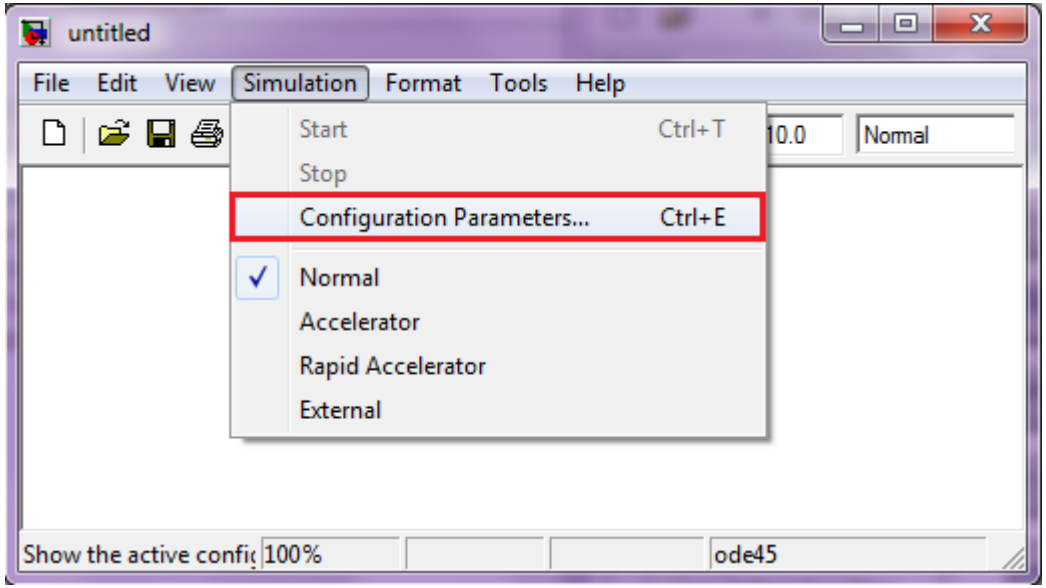
- Variable-step solvers are easier to use (the sample time is automatically determined)
- BUT they do not allow ***DETERMINISM*** (which is mandatory for real-time applications) : we do not know *a priori* how long the next step will last

➔ The use of **fixed-step** solvers is mandatory

Setting simulation parameters

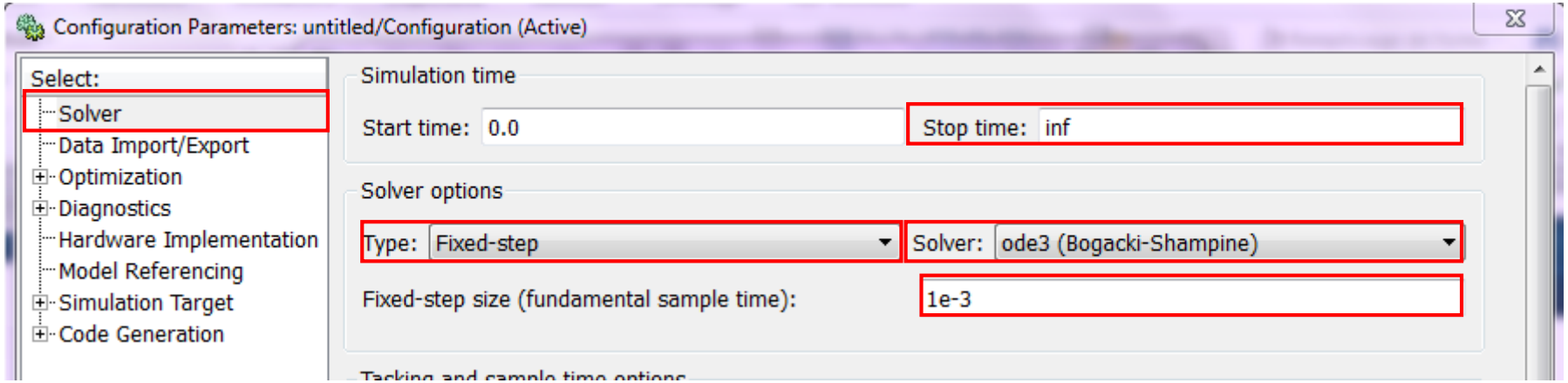
Configuration parameters

- Some simulation options need to be set before running the simulation
- In the Simulink® model, menu **Simulation → Configuration Parameters…**



Setting simulation parameters

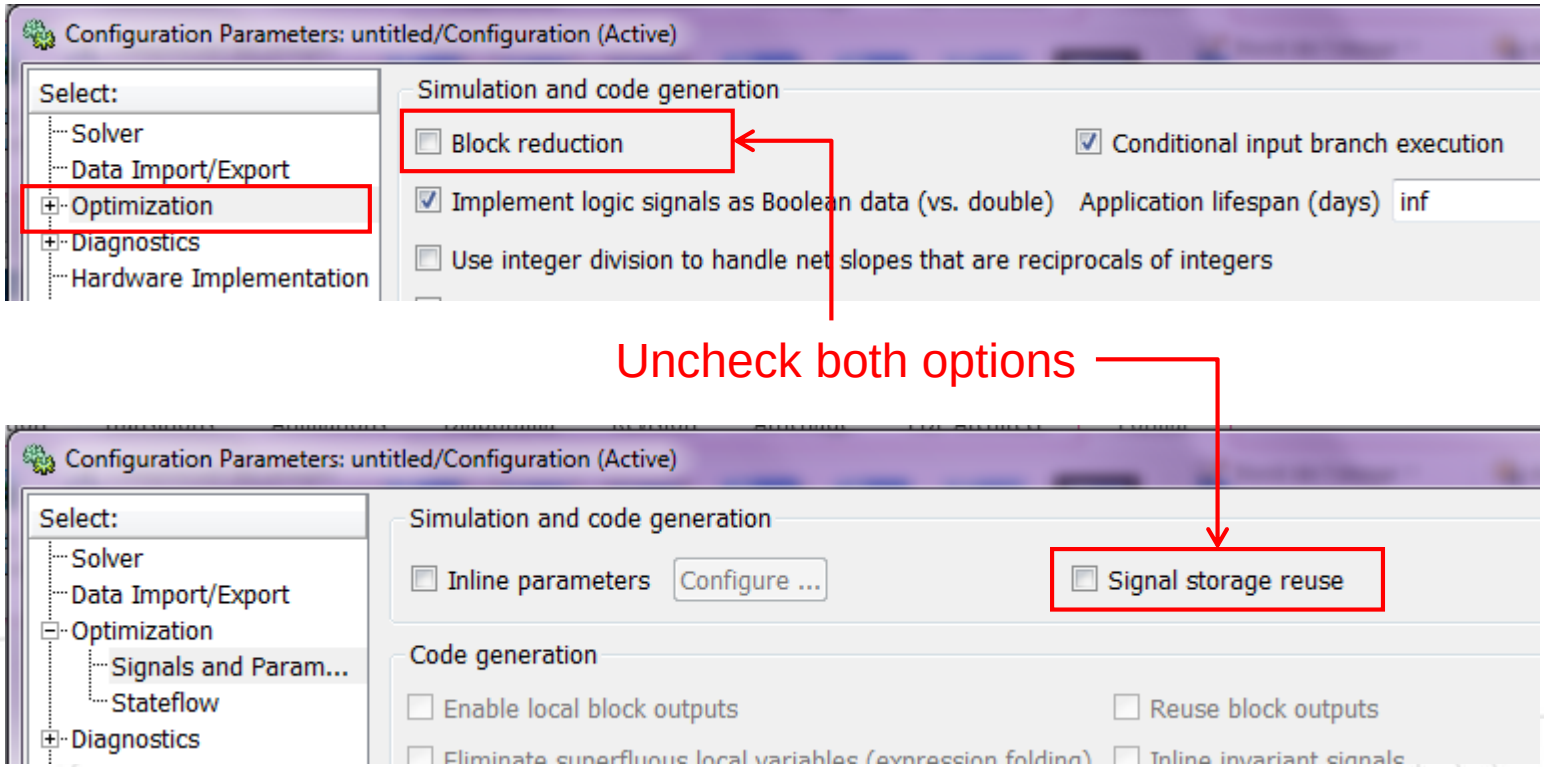
Configuration parameters



- Set **Stop time** to ***inf*** : the simulation will run until user decides to stop it
- Set **Type** to ***Fixed-step*** : see previous slides
- Select any fixed step **Solver**
- Set the **Fixed-step size** : value in seconds

Setting simulation parameters

Configuration parameters

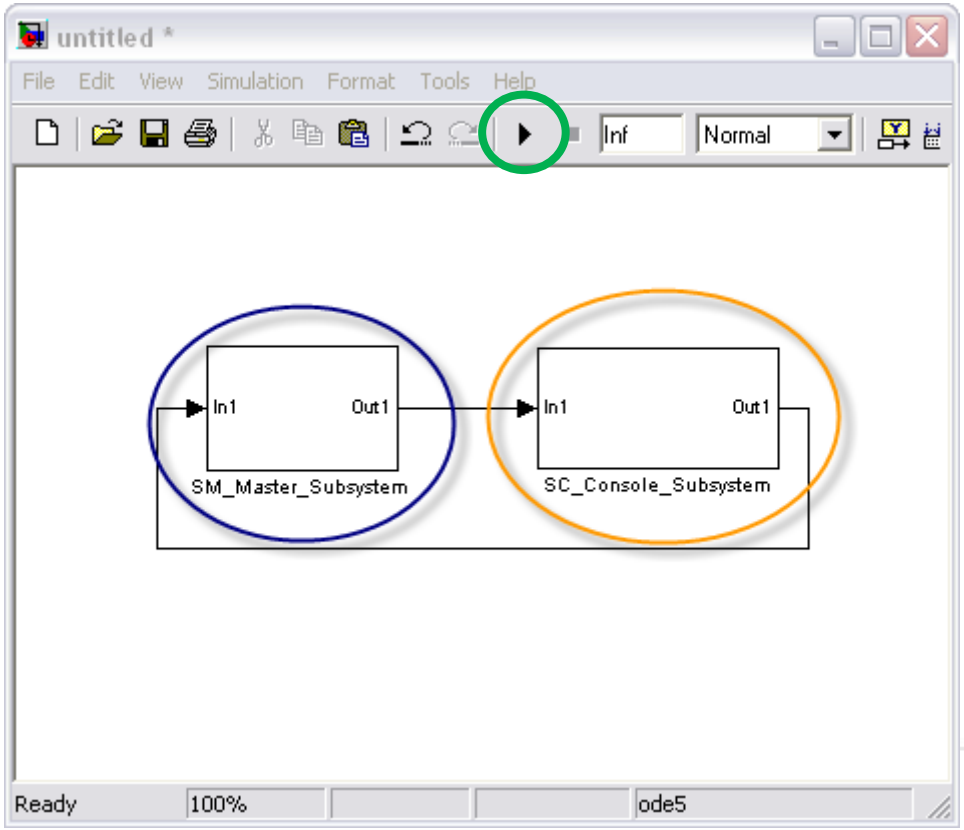


Outline

1. General concepts
2. Grouping into subsystems
3. Adding OpComm blocks
4. Maximizing parallel execution
5. Setting simulation parameters
6. **Executing off-line**

Executing off-line

- Run the model off-line and make sure no error is raised
- If the model does not run under Simulink®, it will not work in real-time
- Once the model runs off-line, we can try to build it with RT-LAB



Questions ?