GridModelica: Modeling and Simulating on the Grid

Håkan Mattsson, Christoph W. Kessler,
Kaj Nyström, Peter Fritzson

Programming Environments Laboratory PELAB
Department of Computer and Information Science
Linköping University Sweden

{g-hakma,chrke,kajny,petfr}@ida.liu.se

Modeling on Linux Clusters

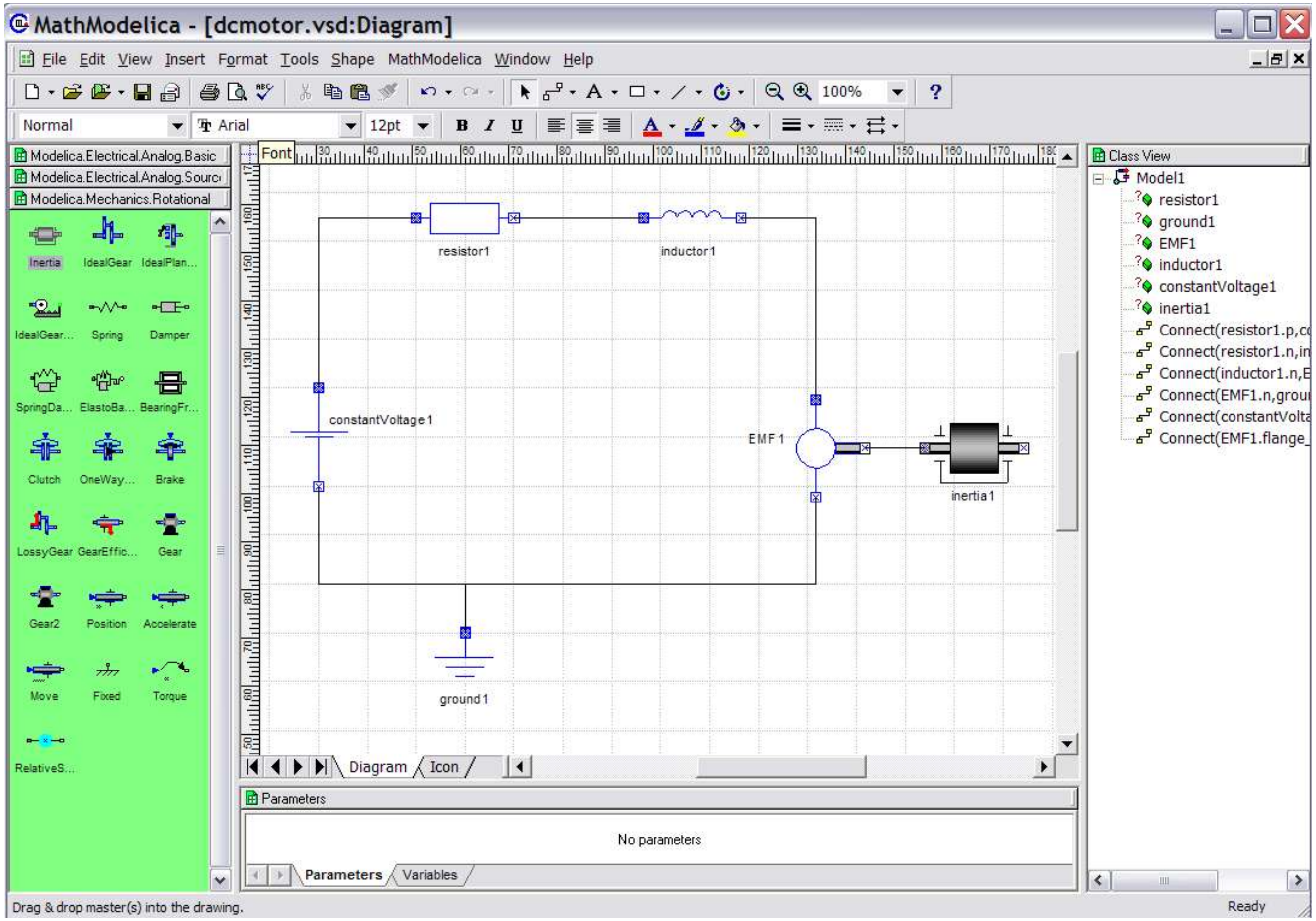
- Widely used for large models
- Requires expertise in parallel programming
- Excellent for run-many-times simulations, not so good for run-once simulations

GridModelica

- Structured modeling on clusters
- Does not require parallel programming expertise
- Domain agnostic (multidomain works too!)
- Graphical programming, close to physical prototyping
- The magic is done behind the scenes

High Level Modeling: Modelica

- Object oriented
- Graphical or textual
- Acausal
- General
- Fast
- Easy to use



Model1
- resistor1
- ground1
- EMF1
- inductor1
- constantVoltage1
- inertia1
- Connect(resistor1.p,co
- Connect(resistor1.n,in
- Connect(inductor1.n,E
- Connect(EMF1.n,groun
- Connect(constantVolta
- Connect(EMF1.flange_

Parameters

No parameters

Drag & drop master(s) into the drawing.

More on Modelica

- Graphical representation corresponds 1:1 to textual representation

```
model dcmotor
  Import Modelica.Electrical.Analog.Basic;
  Resistor r1(R=10);
  Inductor i1;
  EMF emf1;
  Modelica.Mechanics.Rotational.Inertia load;
  Ground g;
  Modelica.Electrical.Analog.Sources.ConstantVoltage v;
equation
  connect(v.p, r1.p);
  connect(v.n, g.p);
  connect(r1.n, i1.p);
  connect(i1.n, emf1.p);
  connect(emf1.n, g.p);
  connect(emf1.flange_b, load.flange_a);
end dcmotor;
```

Problems

1. Partition the model

3. Structured communication
   (Håkan Mattsson)

Partitioning a model

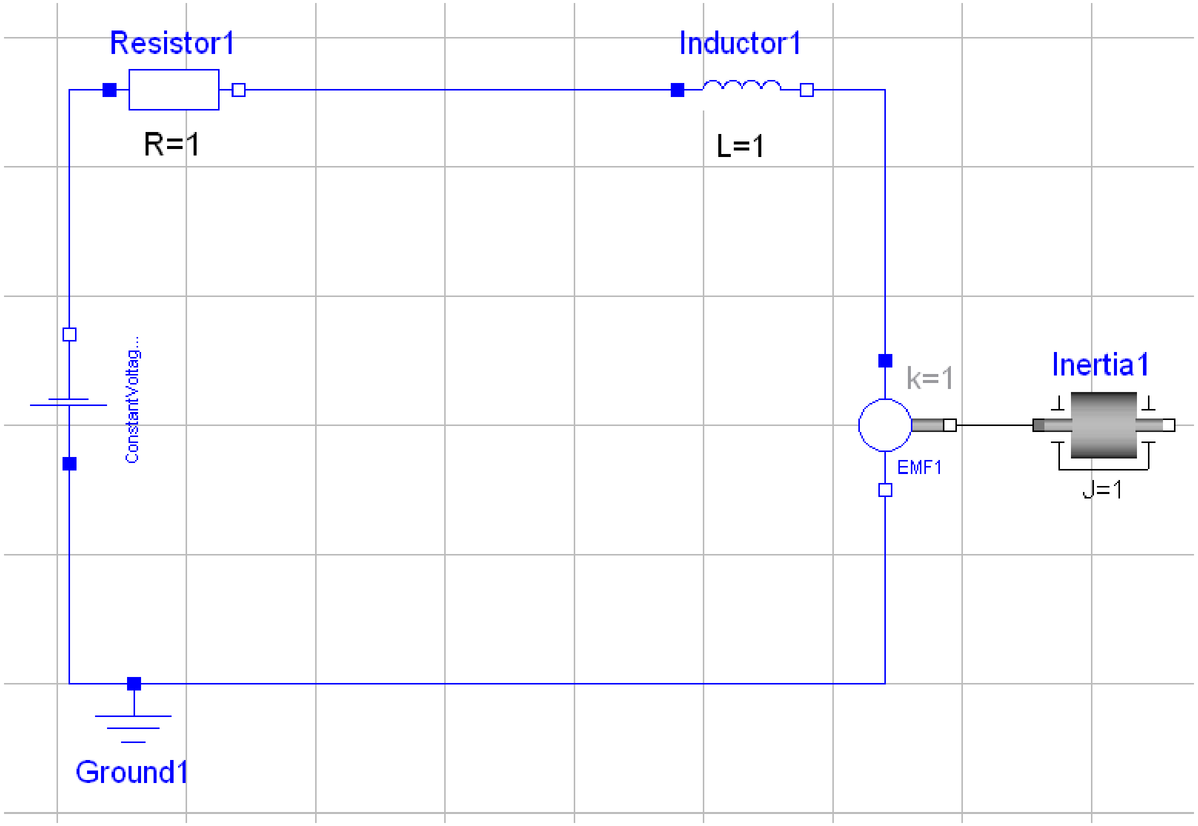
Some observations

- It is all about solving large systems of equations
- Parallel solvers exist but can not always be applied (stability issues) and do not always improve speed.

Transmission Line Modeling [1]

All propagation in a model (waves, force, current etc)
is done with a certain *delay*.

Use this delay to send data less frequently.



1. [Johns 1972]

Transmission Line Modeling

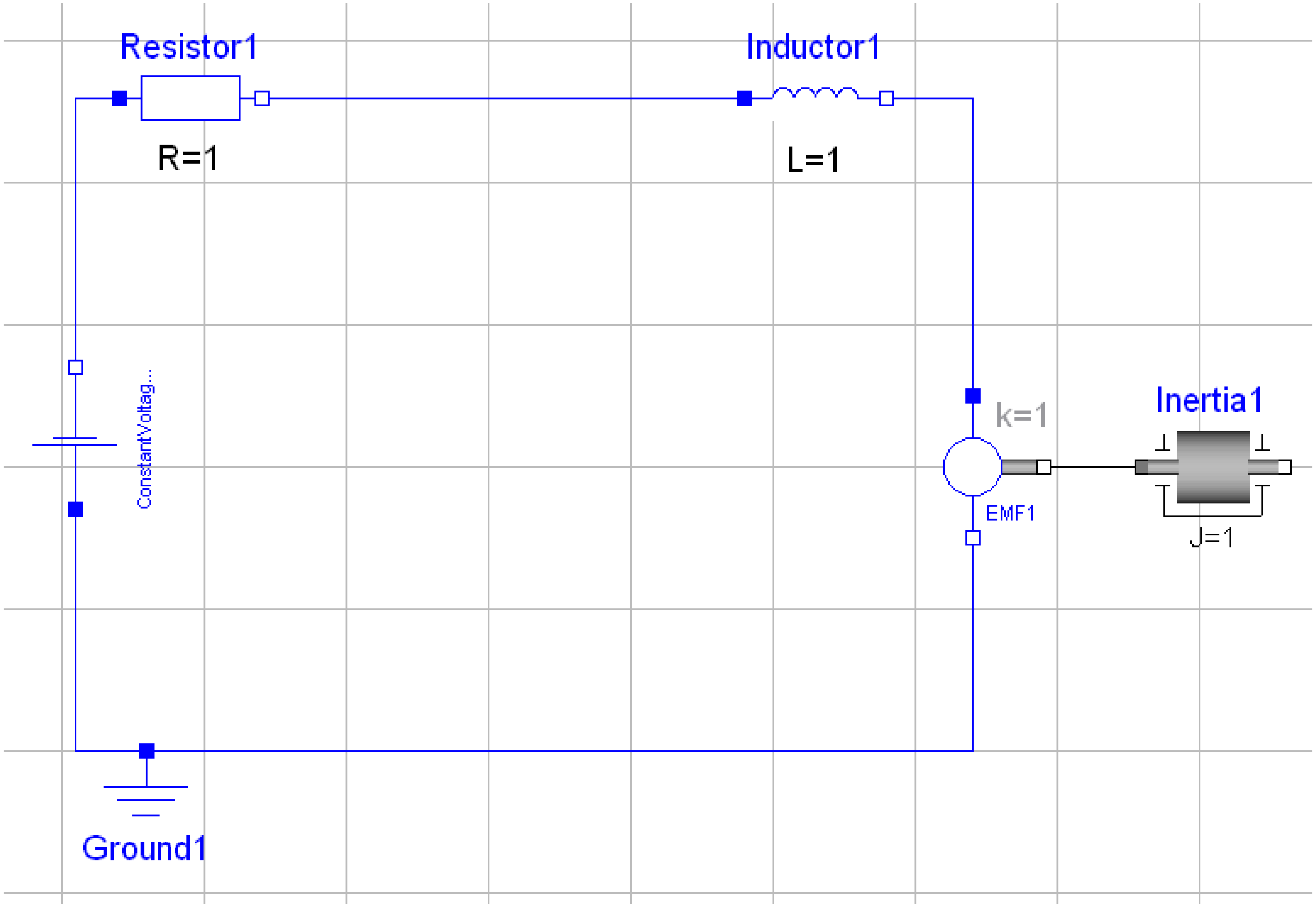
- Reuse values

- Different solvers (and settings) for different parts of a system

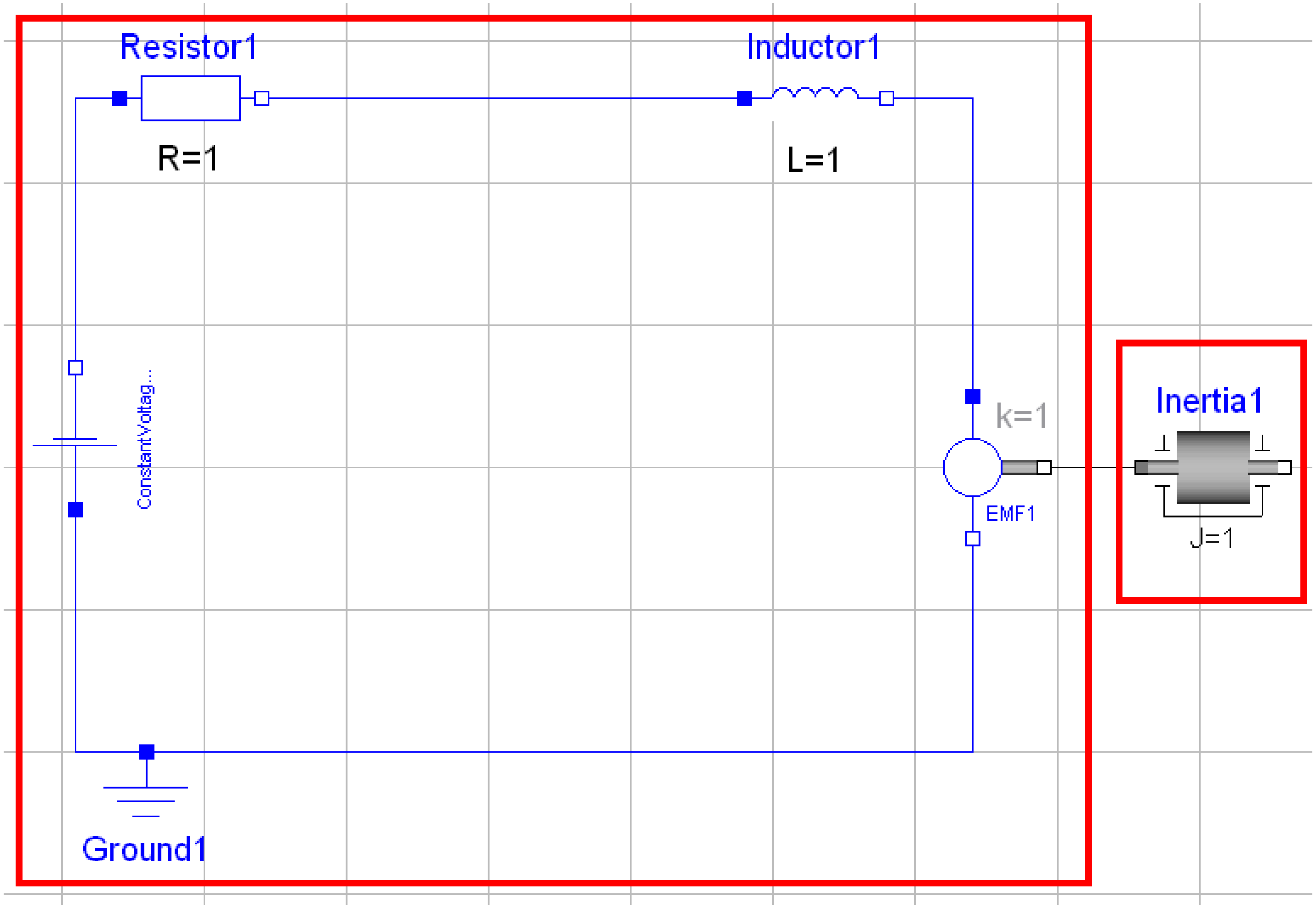
- Communication in bulk

- The error introduced is well defined and generally very small.

Transmission Line Modeling



Transmission Line Modeling



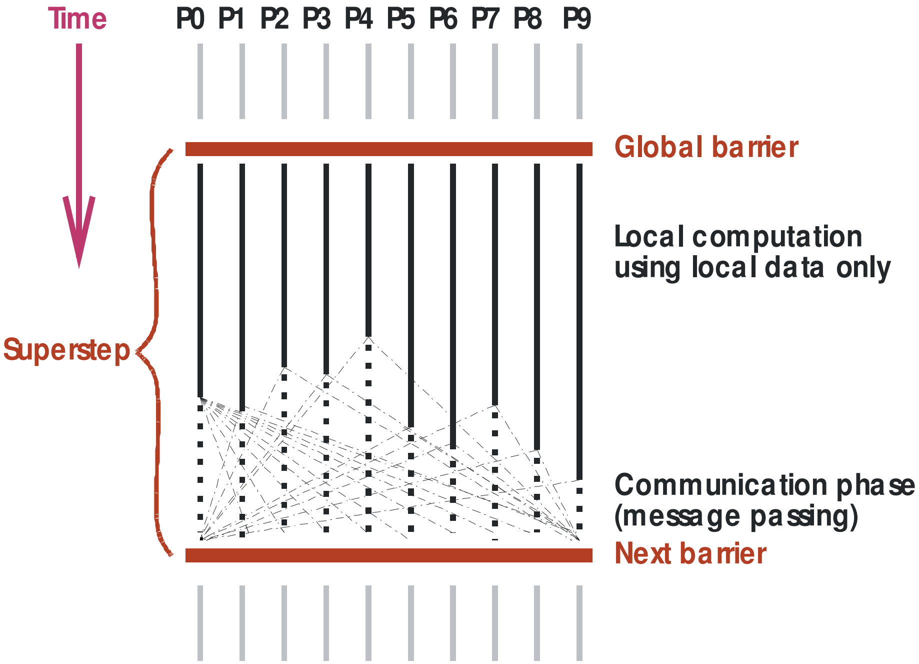
GridNestStep

- For grid applications with a non-trivial structure of parallelism, generation of efficient, scalable code is an unsolved problem

- **Goal** – to provide an ”easy-to-use” programming environment by introducing a programming language, *GridNestStep*, that supports
  - development of applications exploiting less trivial kinds of parallelism
  - a virtual shared memory view of a grid system

GridNestStep

- GridNestStep
  - follows the *Bulk Synchronous Parallel* (BSP) model of computation
  - will be based on *NestStep*


Time
P0
P1
P2
P3
P4
P5
P6
P7
P8
P9
Global barrier
Local computation using local data only
Superstep
Communication phase (message passing)
Next barrier

- BSP
  - cost model for parallel programs
  - Single Program, Multiple Data execution style, (SPMD)
  - organizes program in *supersteps* consisting of
    1 – computation
    2 – communication

NestStep

- NestStep [Kessler, 2000]
  - parallel programming language for the BSP model
  - language extensions for Java / C / C++
- Extends BSP by
  - static and dynamic *nesting* of supersteps
  - synchronization of processor *subsets* (groups)
  - software emulation of *virtual shared memory*

-
```
step {                           neststep(2, @=expr) {
  statements                      statements
}                                 } // @ = group id
```

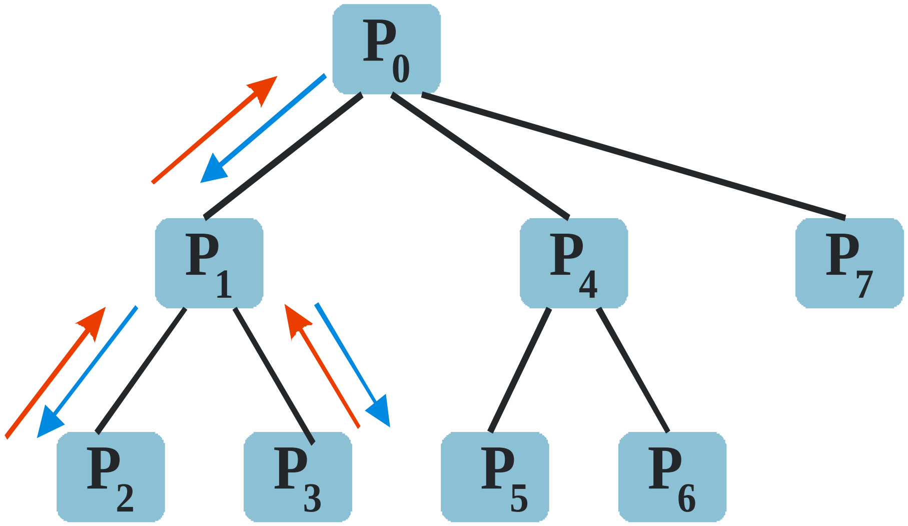
NestStep

- Variables, arrays and objects are
  - *private* to a processor or
  - *shared* between a group of processors
- Modes of sharing:
  - *replicated,* local copy on *each* processor in a group
  - *distributed*, an array partitioned between processors in a group
- NestStep superstep invariants:
  - *superstep synchronicity,* all processors of the same group work on same superstep
  - *superstep consistency*: entry to a `(nest)step` statement ⇒ equal values for local copies of shared variables

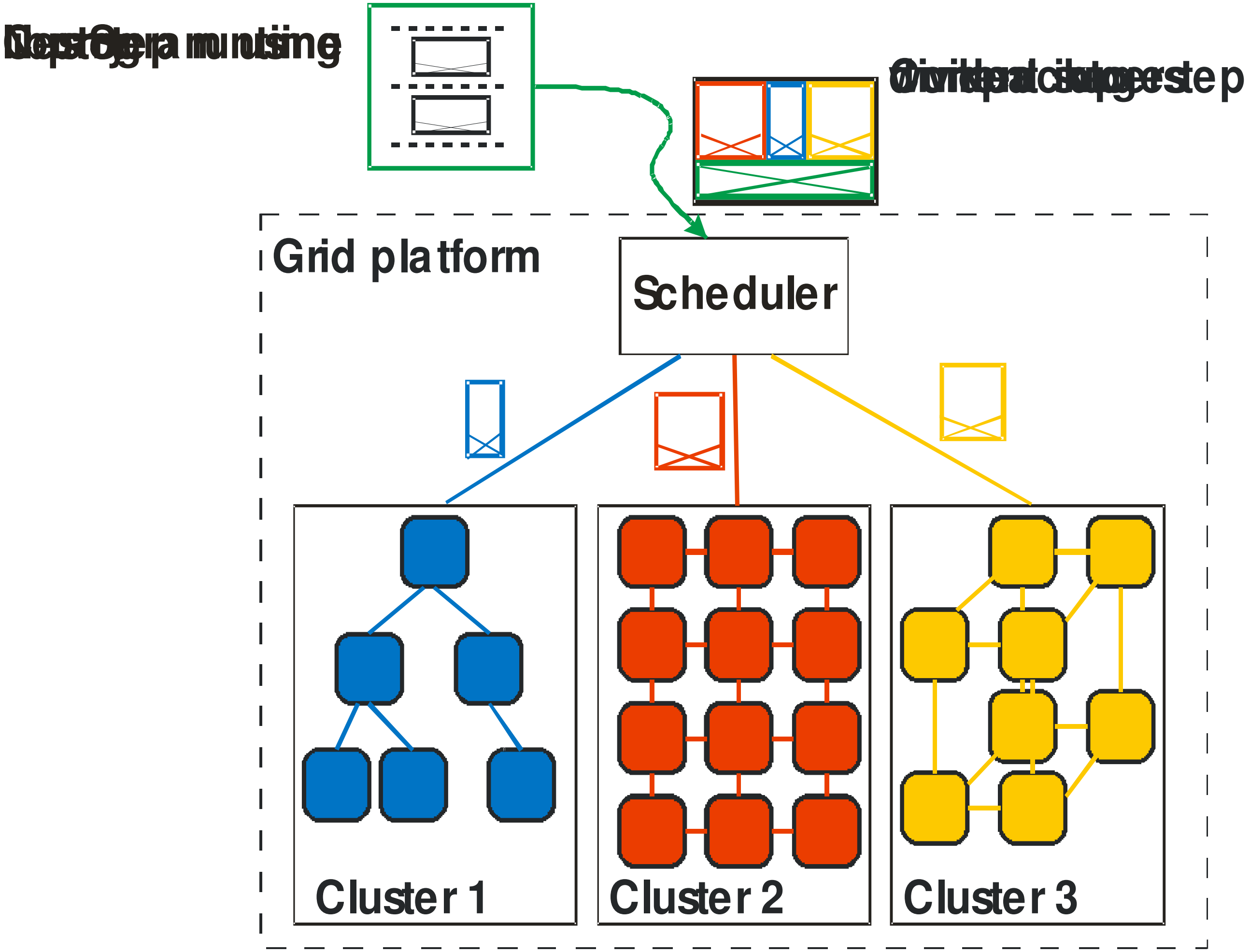
NestStep

- Communication in processor groups organized as trees

- Superstep consistency maintained by a combine phase at the end of each superstep
  - upwards combine
  - downwards commit



GridNestStep



GridNestStep

- Some (known) problems to be solved:
  - superstep analysis and partitioning into workpackages:
    - how to monitor load and
    - perform load balancing accordingly
  - latency
  - failing grid nodes
  - code distribution

Current status

- Parameter sweep tool for Modelica works fine (Modelica runs on the grid!)
- Partial test implementation for TLM in Modelica exists
- Only very simple examples works for now
- Partitioning only by hand and only in textual model (no drag'n drop tool support yet)
- NestStep runs on a single cluster

Future Work

- Generalize the partitioning method to all physical domains
- Automatic partitioning at domain boundaries and natural subsystem borders
- Automatic solver and step size selection
- Better scheduling
- Co-simulation integration (with SKF)
- Continue with multi-cluster support and transition to SweGrid
- NestStep front end

Questions?