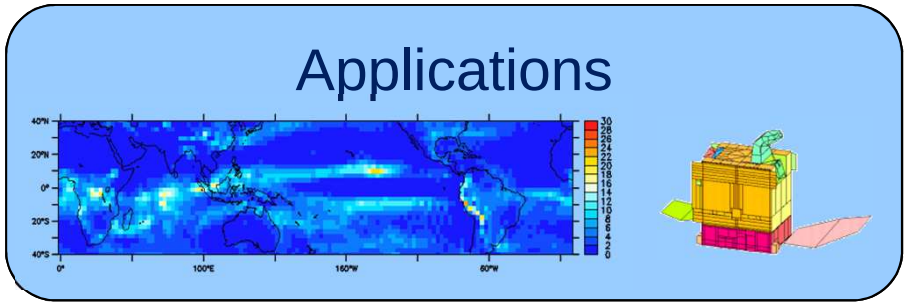
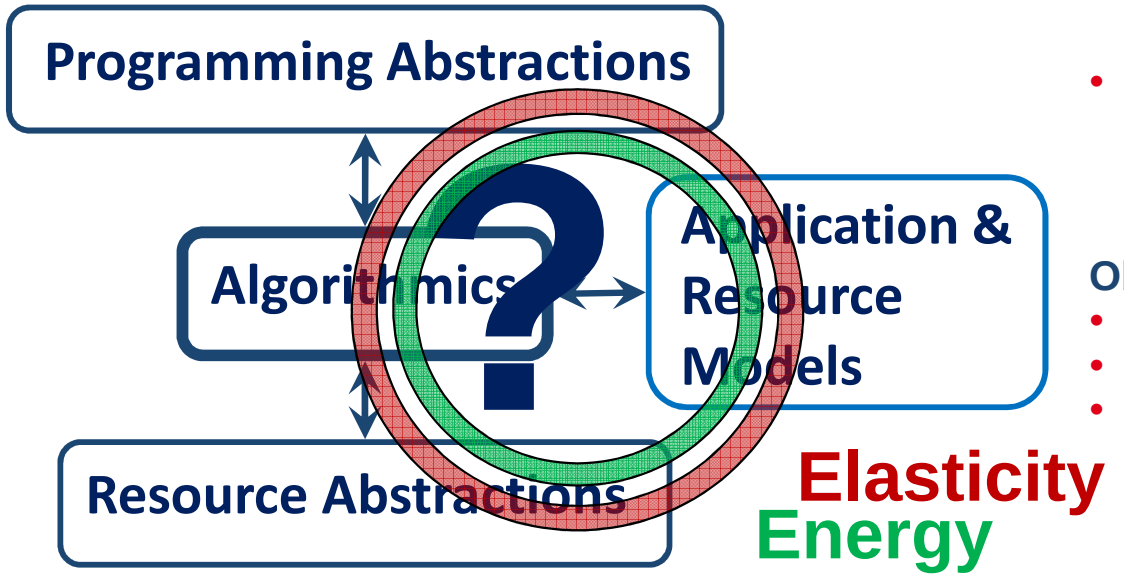
AVALON
Algorithms and Software Architectures for Distributed & High Performance Computing Platforms

Christian Perez

2013, May 17

Avalon: Research Activities





CPU/data-intensive Scientific Applications

- From "simple" to code coupling
  - Structure complexity
  - "New" forms of interactions (MR)

Computing platforms

- Different characteristics
  - Performance, energy, size, cost, reliability, QoS, etc.
- Hybridization
  - Sky computing, HPC@Cloud, Exascale, Spot instance

Objectives

- Expressiveness simplicity
- Application portability
- Resource specific optimizations
  - Elastic resource management
  - Energy consumption

Super-computers (Exascale)	Grids (EGI)	Desktop Grids	Clouds (IaaS, PaaS)
Large scale	Heterogeneity	Volatility	On demand

Component Models for HPC

J. Bigot, H. Bouziane, A. Ribes, C. Perez

LIP, ENS Lyon

Programming a Parallel Application

(High level) parallel languages

- HPF, PGAS, …

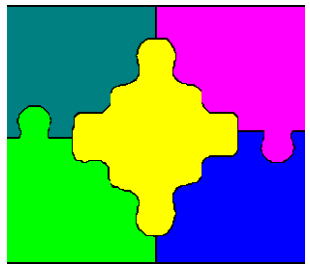
➢ Not yet mature

Platform oriented models

- Multi-core ⇔ Threads, OpenMP
- GPU ⇔ Cuda, OpenCL, OpenAPP
- Multi-node ⇔ MPI

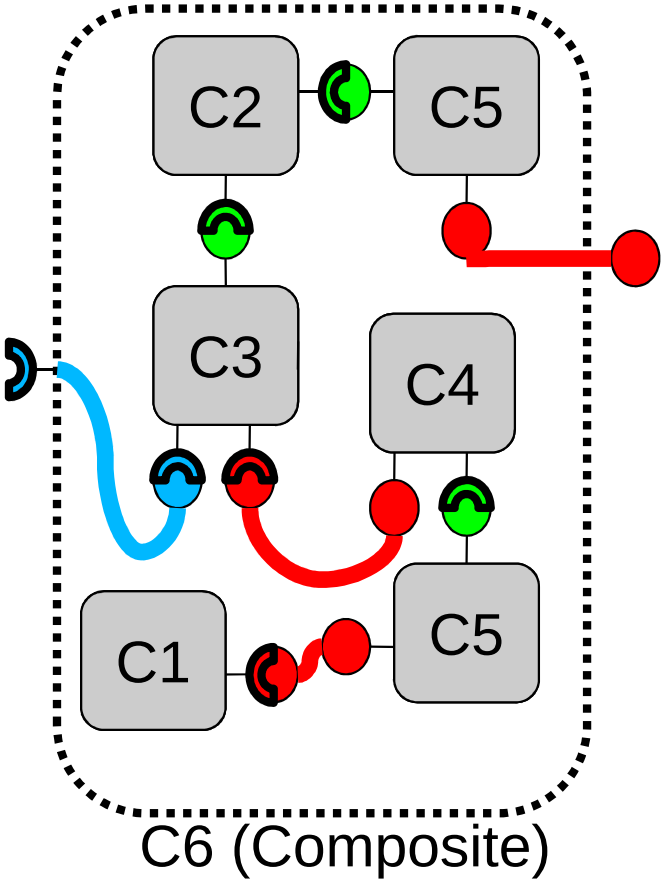
➢ Many versions of the same code
➢ Difficult to maintain all versions synchronized
➢ Difficult to keep specific machine optimizations
➢ Low code reuse

Software Component



Technology that advocates for composition

- Old idea (late 60's)
- *Assembling* rather than *developing*



Many types of composition operator

- Spatial, temporal, …. composition

Assembly of component

- Primitive & composite components

Many models

- Salome, CCA, CCM, Fractal, OGSi, SCA, …

Application in Hydrogeology: Saltwater Intrusion

Coupled physical models

One model = one software

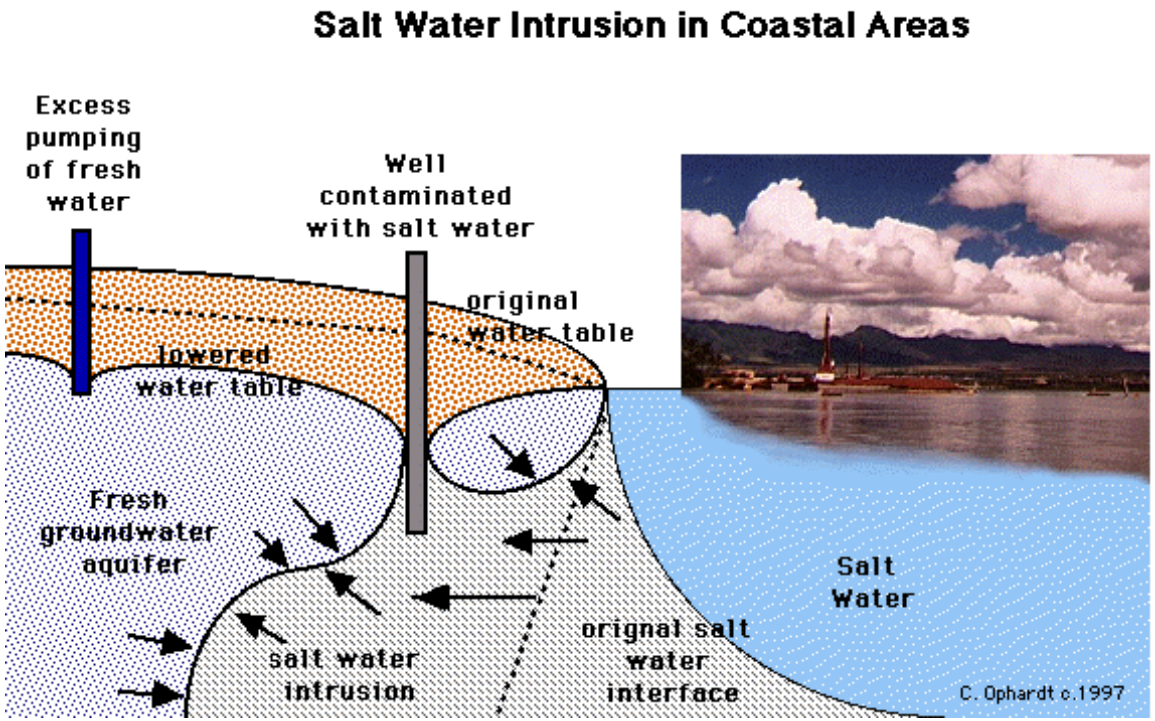
Saltwater intrusion

- Flow / transport

Reactive transport

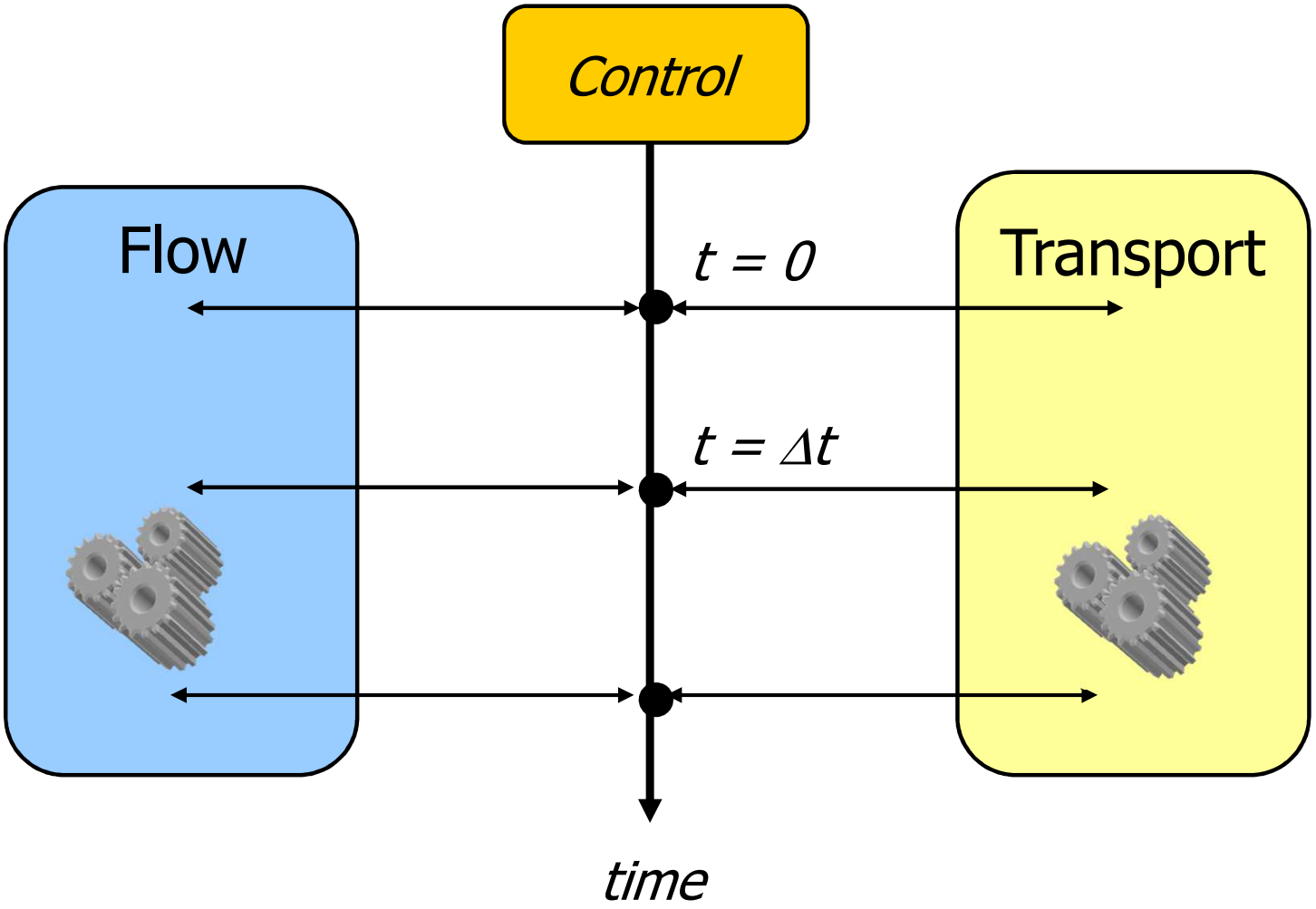
- Transport / chemistry

Hydrogrid project, supported by the French ACI-GRID



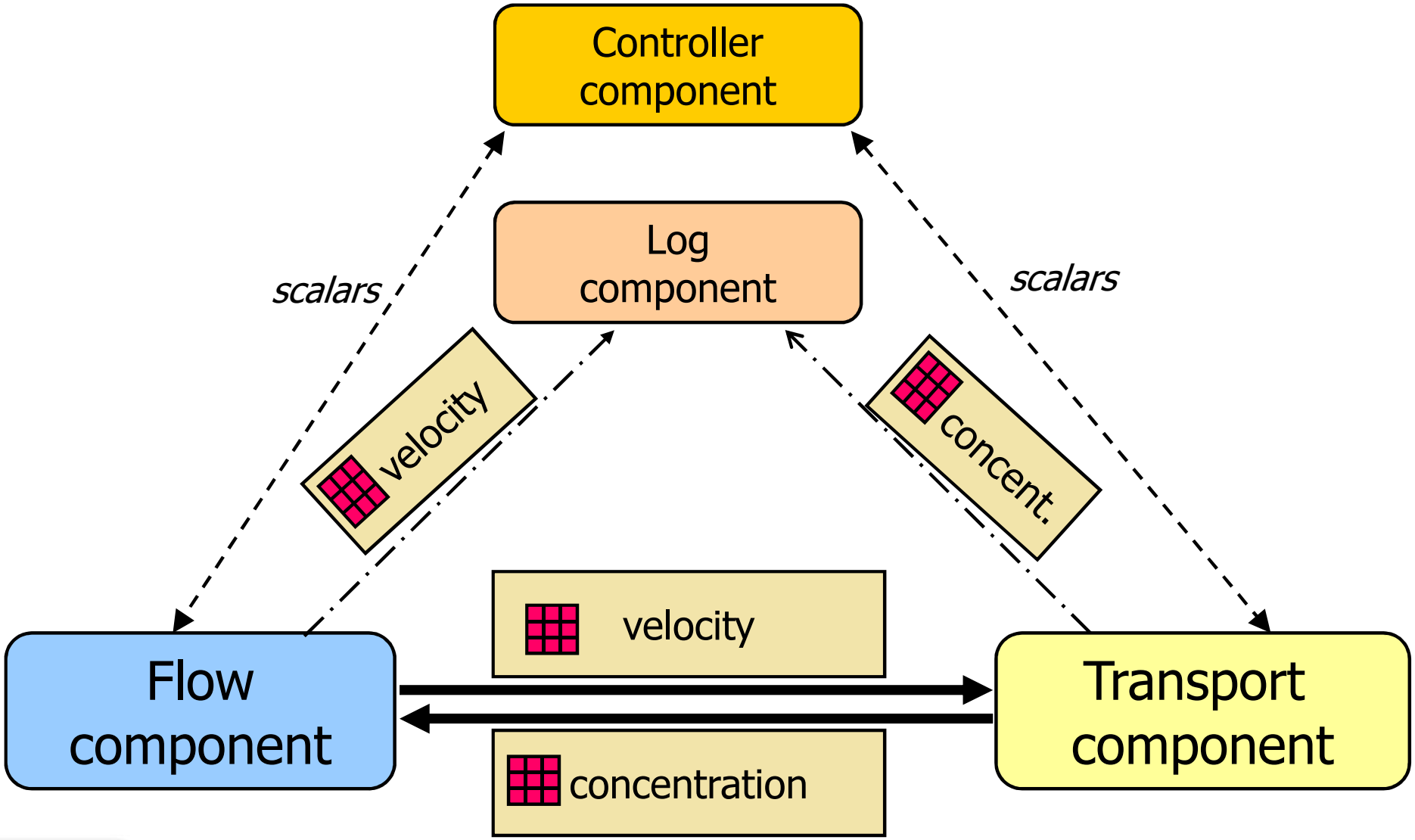
Flow: velocity and pressure function of the density
Density function of salt concentration
Salt transport: by convection (velocity) and diffusion

Numerical Coupling in Saltwater Intrusion

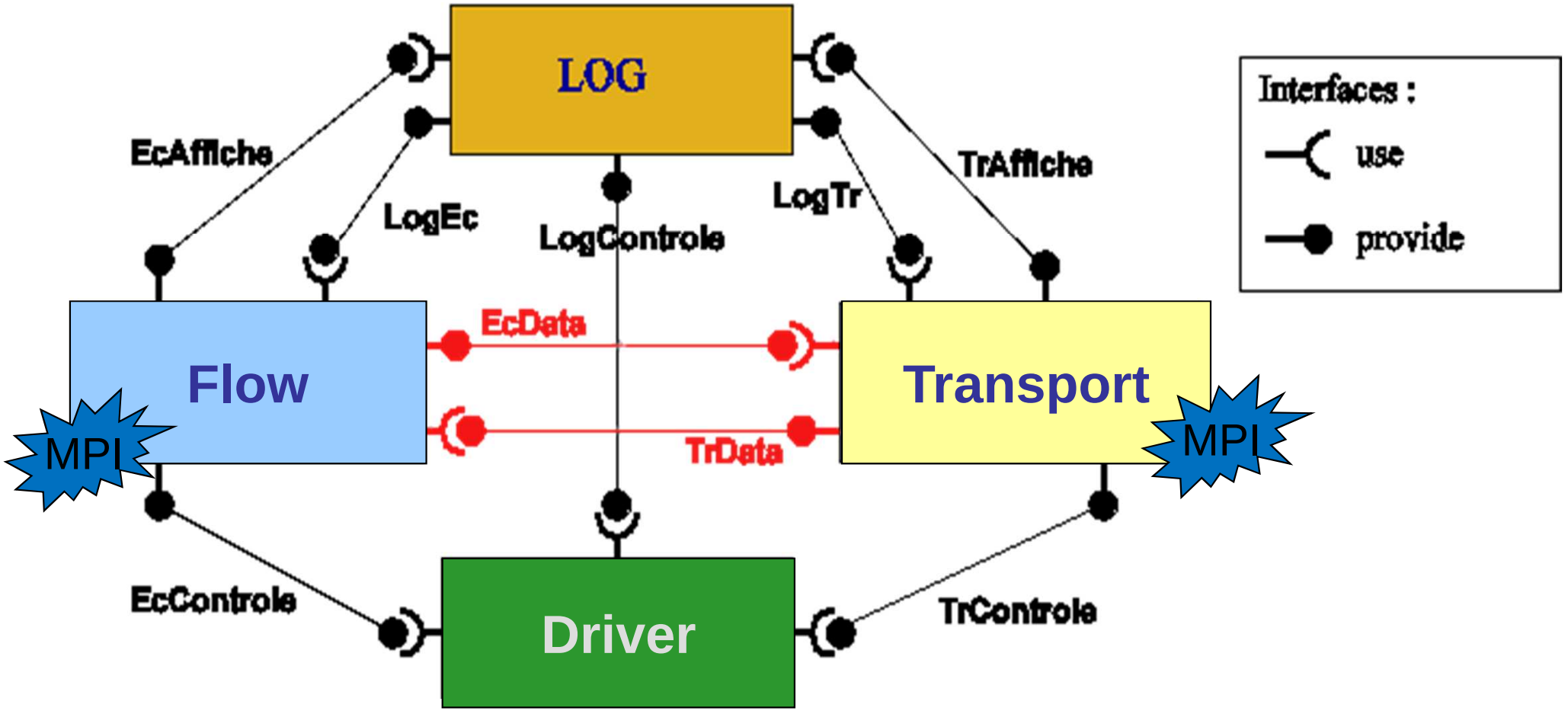


iterative scheme at each time step

Components and communications of PCSI



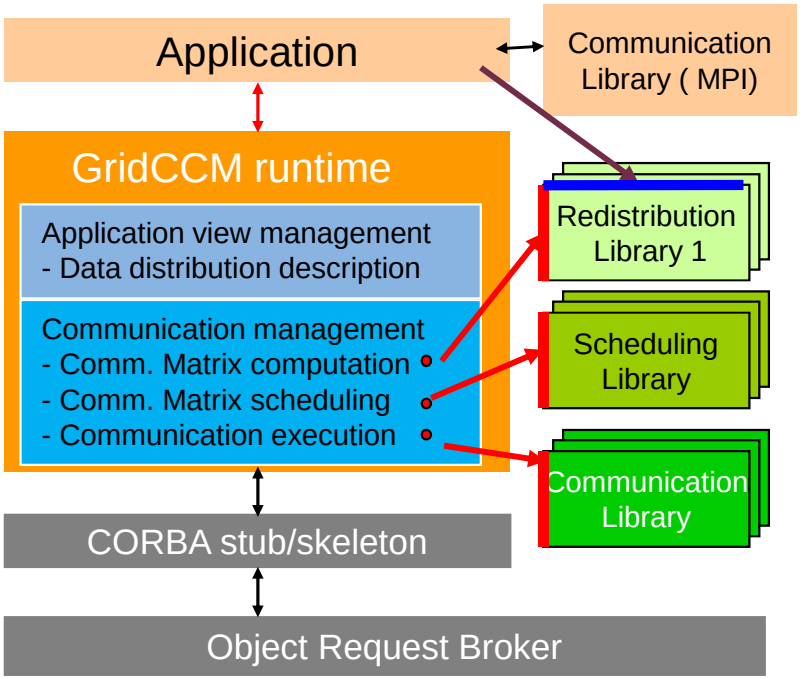
Components and interfaces of PCSI

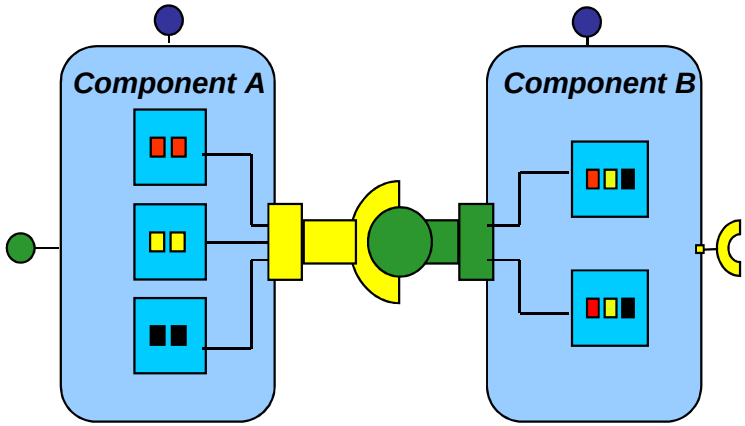


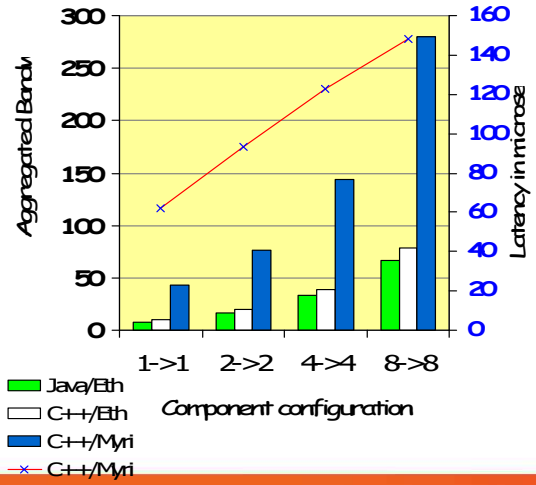
Parallel Components: SPMD Paradigm in GridCCM

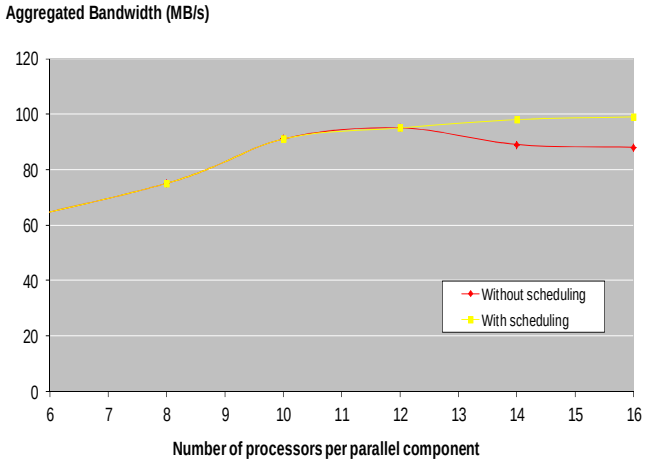
SPMD Component

- Parallelism is a non-functional property of a component
  - It is an implementation issue
- Collection of sequential components
  - SPMD execution model
  - **External com. mechanism (MPI)**
- Support of distributed arguments
  - API for data redistribution
  - API for communication scheduling
- Support of parallel exceptions









Application in Hydrogeology: Saltwater Intrusion

Coupled physical models

One model = one software

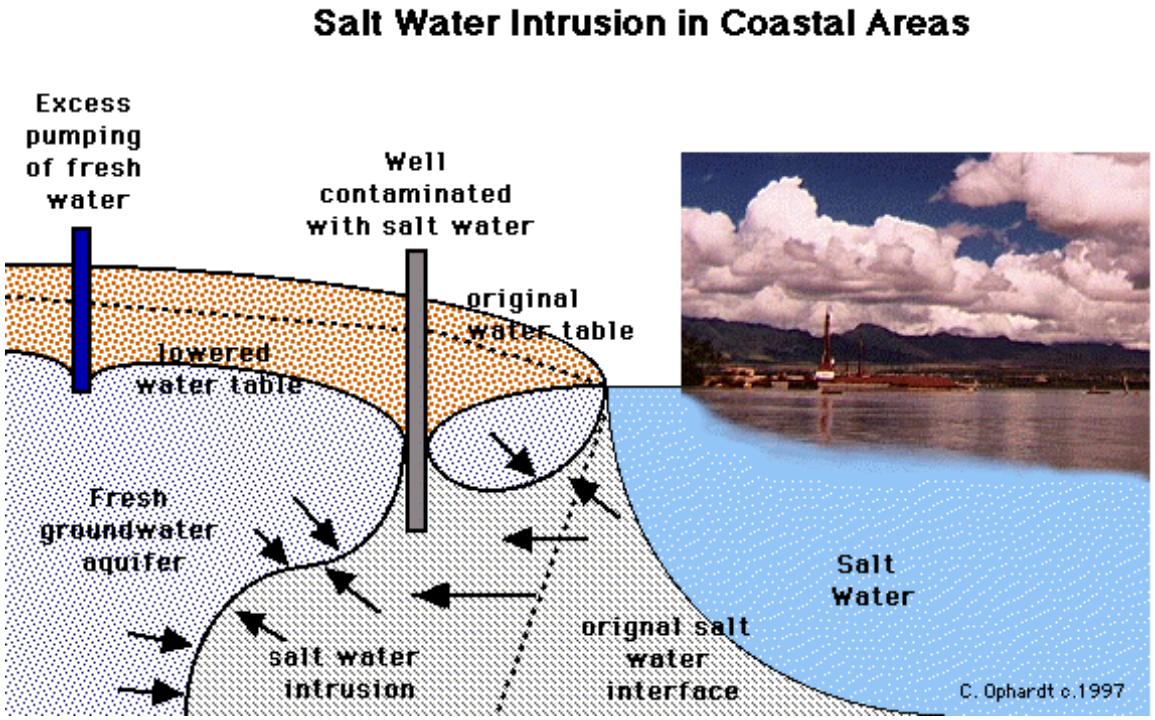
Saltwater intrusion

- Flow / transport

Reactive transport

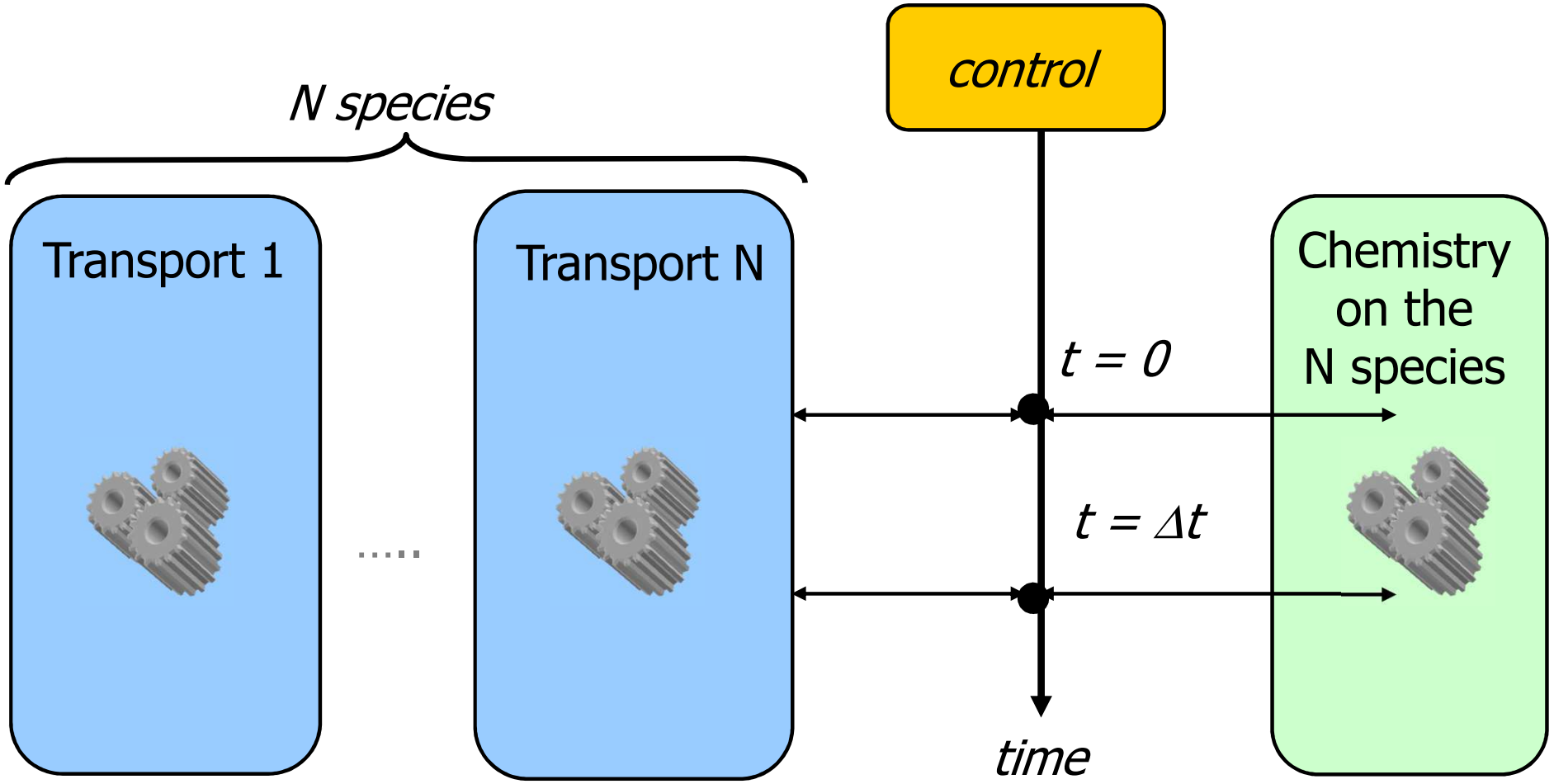
- Transport / chemistry

Hydrogrid project, supported by the French ACI-GRID



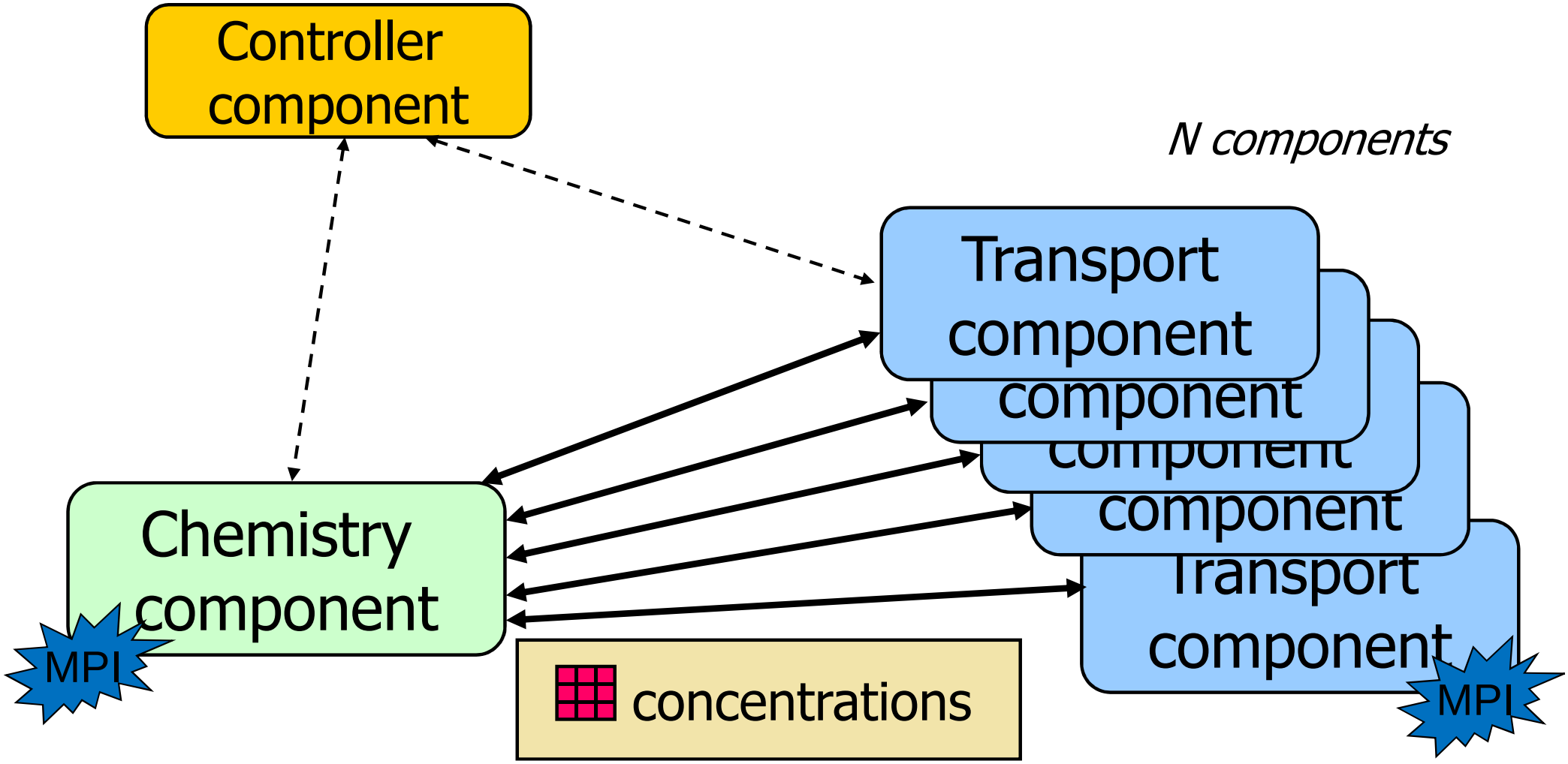
Flow: velocity and pressure function of the density
Density function of salt concentration
Salt transport: by convection (velocity) and diffusion

Numerical coupling in reactive transport



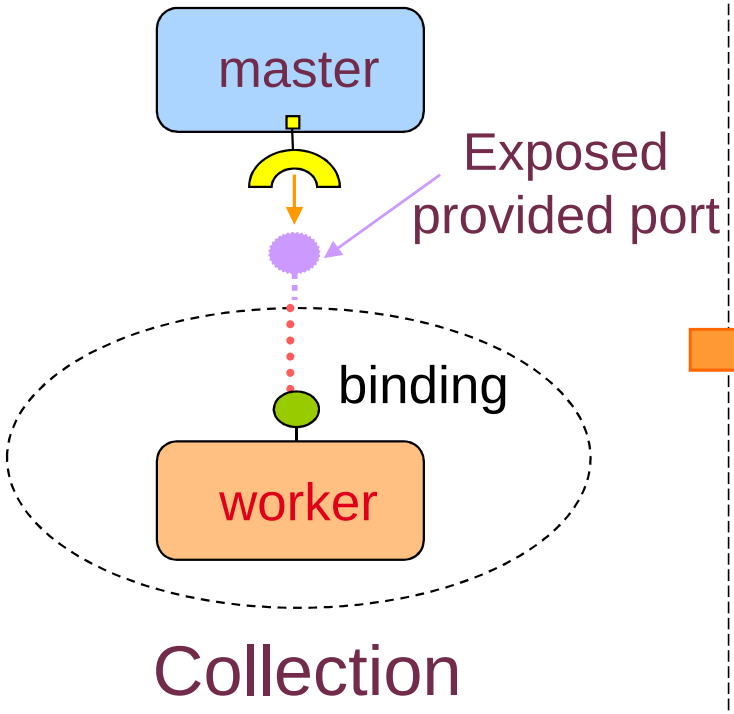
Iterative scheme at each time step

Component Model for Reactive Transport

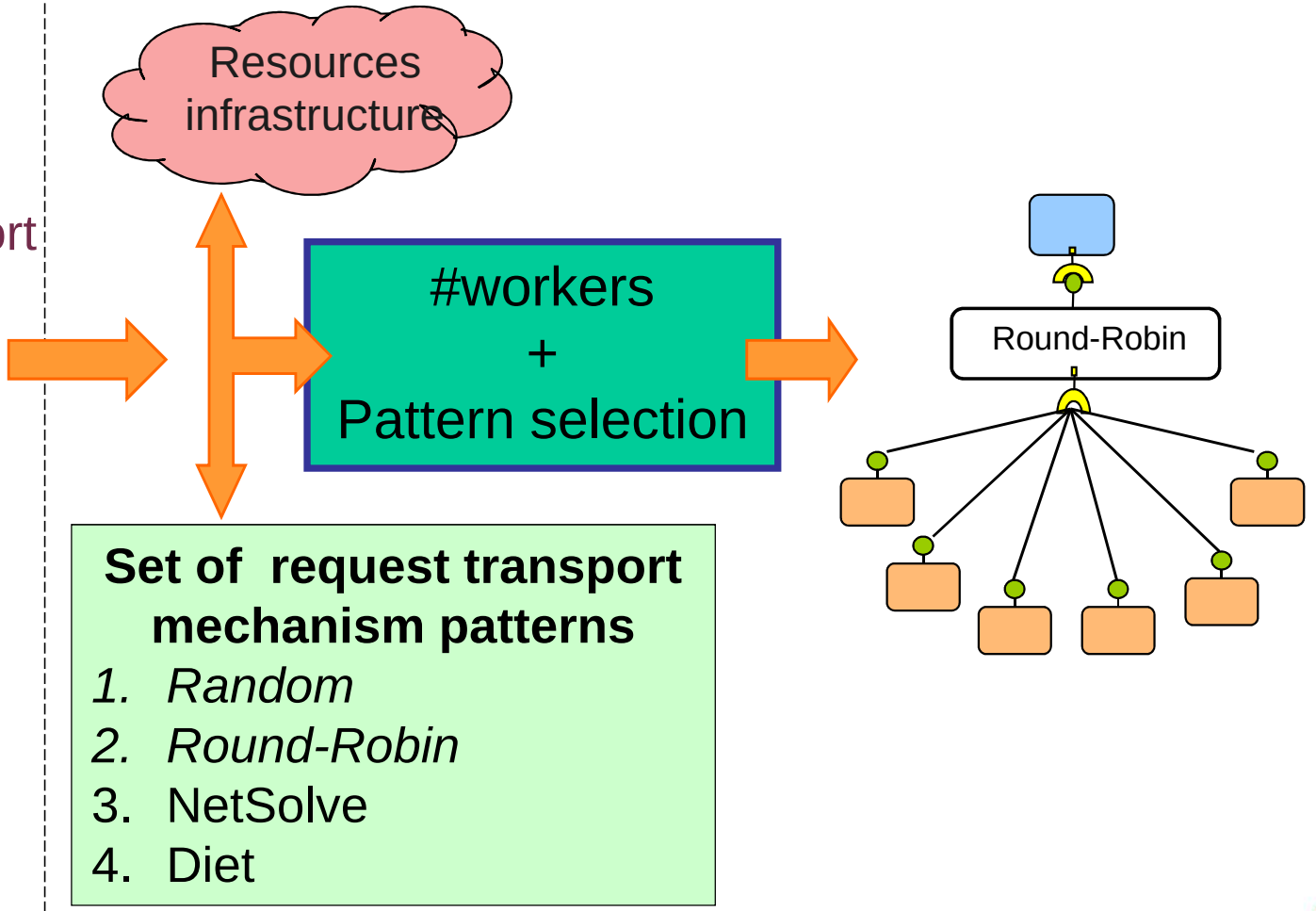


Master-Worker oriented Component Collection (aka Farm Skeleton)

Programmer/designer view



Framework implementation view



Set of request transport mechanism patterns

1. *Random*
2. *Round-Robin*
3. NetSolve
4. Diet

Limitations of Existing HPC Component Models

Pre-defined set of interactions

- Usually function/method invocation oriented

➢ How to incorporate other interactions, e.g. MPI?

Provide communication abstraction

- Language interoperability (~IDL)
- Network transparency

➢ Potential overhead when not needed

➢ Limited data types systems

  - Babel SIDL, OMG IDL, …

➢ ***Programming model vs execution model***

Objectives

Enable code-reuse

Let expert develop a piece of code

- Software Component
  - Primitive component for re-using implementation code
  - Composite component for re-using assemblies of components

Enable *adaptation* when re-using code

Let re-use code with parameterization options

- Genericity

Enable any kind of composition operators

Do not impose any communication models

- Connectors

Enable efficient implementation of composition operators

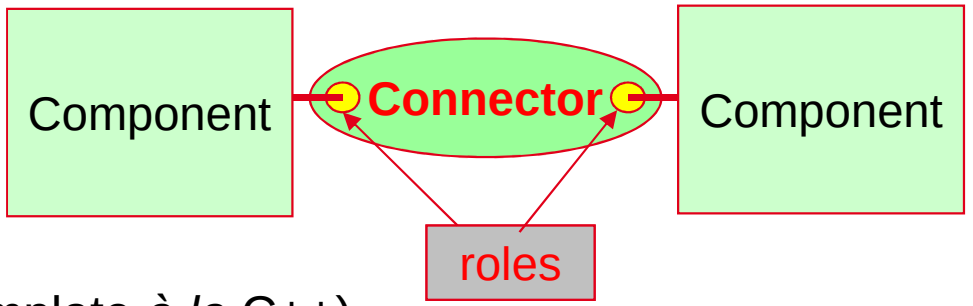
Let have (resource) specific implementations

- Open connection

HLCM: High Level Component Model

Major concepts

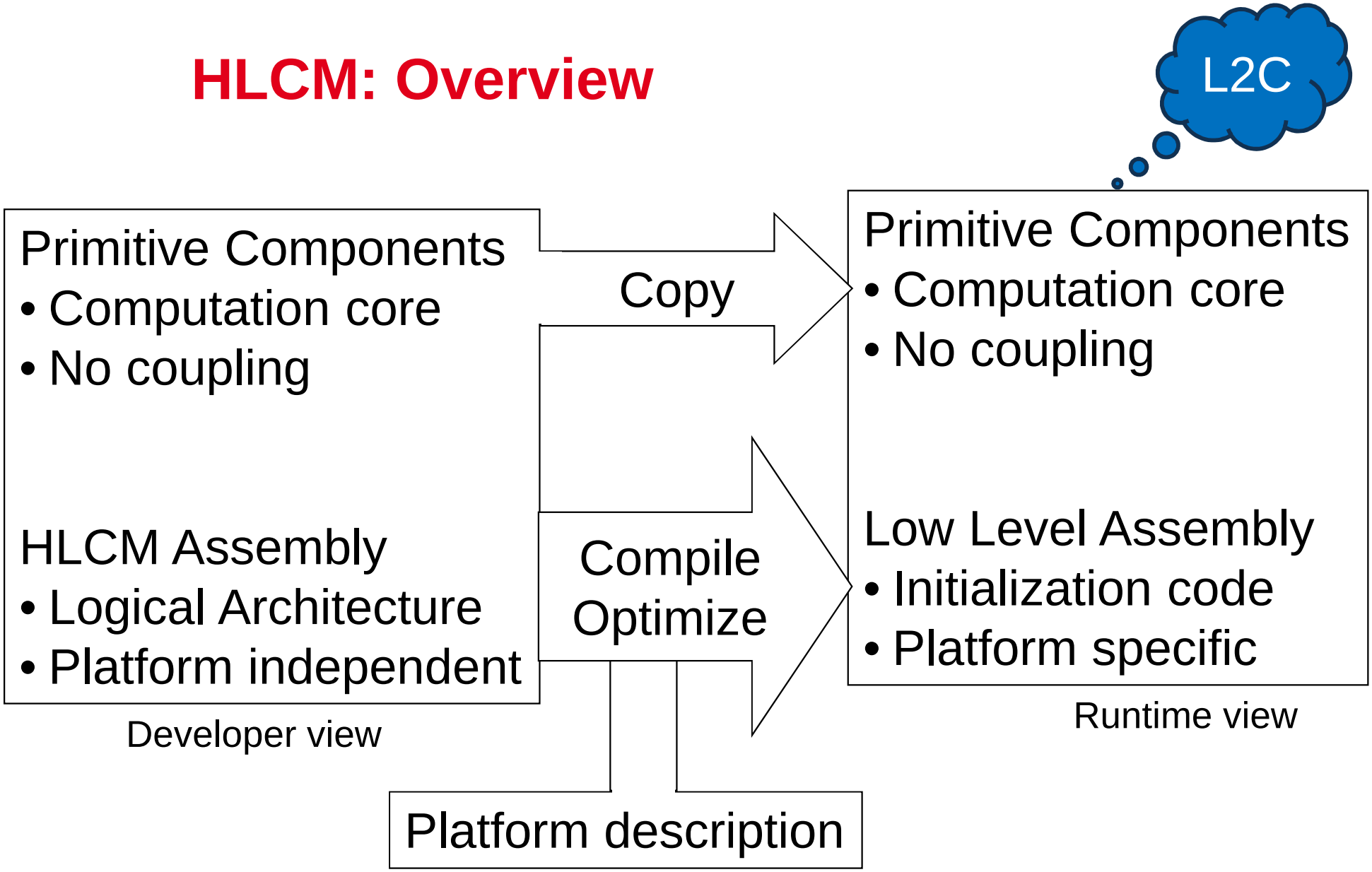
- Component model (hierarchical)
  - Primitive and composite
- Connector based
  - Primitive and composite
- Generic model
  - Support meta-programming (template *à la* C++)
- Currently static



HLCMi: an implementation of HLCM

- Model-transformation based (EMF)
- Connectors
  - Use/Provide
  - Shared Data
  - Collective Communications
  - MxN
  - Some skeletons
    - Replication, Simple Domain Decomposition, MapReduce

HLCM: Overview



HLCM compiler (hla to l2c) GPL

L2C: Low Level Component Model

A minimalist component model for HPC

- Component creation/deletion & connection
- An (optional) launcher

No L2C code between components @ runtime

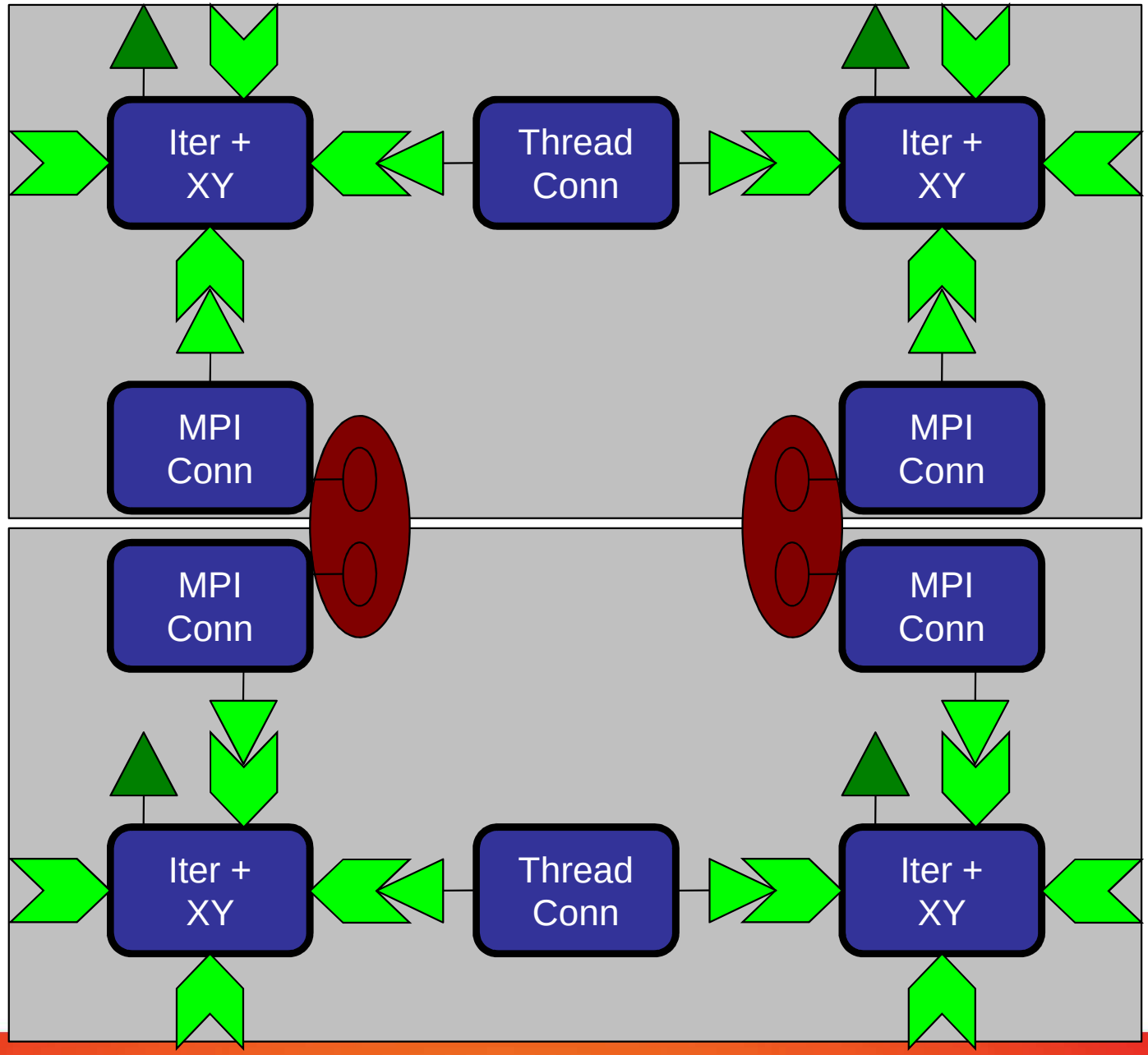
Support native interactions

- FORTRAN procedure (2003+), C++ interface, MPI, CORBA

Extensible

Runtime, LGPL, available at *hlcm.gforge.inria.fr*

Jacobi & The 4-Connector: Hierarchy



Conclusion

Software component is a promising technology for handling code & resource complexity

Component model as a *programming* model

- Many composition operators has been already defined & prototyped
- Re-use existing specialized middleware
- Good feedback from users
- HLCM as a general purpose component model
- L2C as primitive component model

Future work

- Other operators? Finer grain?
  - Domain decomposition, AMR, MapReduce, …
- HLCM expressiveness
  - GPU? PGAS?
  - Temporal composition seems possible
- Automatic component/connector selection & configuration
  - Need to interact with resources