Evaluation of an HPC Component Model on Jacobi and 3D FFT Kernels

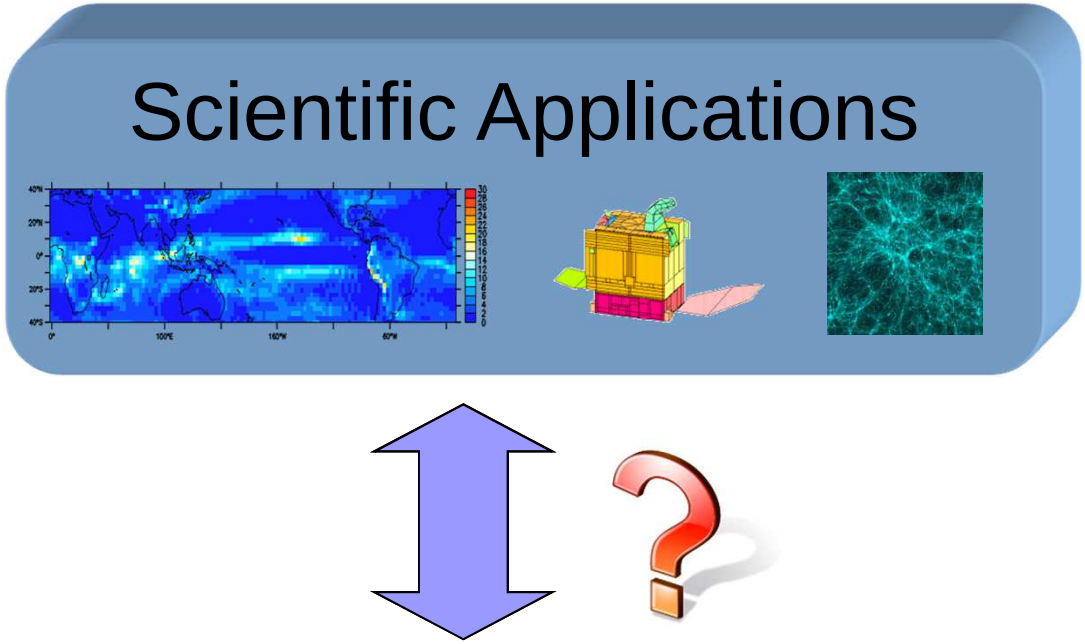
Christian Perez

Joint work with J. Bigot (CEA), V. Lanore (Inria), J. Richard (Orléans)

Avalon Research Team, LIP, Lyon, France

Clusters, Clouds, and Data for Scientific Computing (CCDSC'14)
La Maison des Contes, France, September 2nd-5th 2014

Context


Scientific Applications
Cluster
(GPU/MC/…)
Grids
(EGEE)
Super-
computer
(Exascale)
IaaS
(Cloud)

Parallel Programming

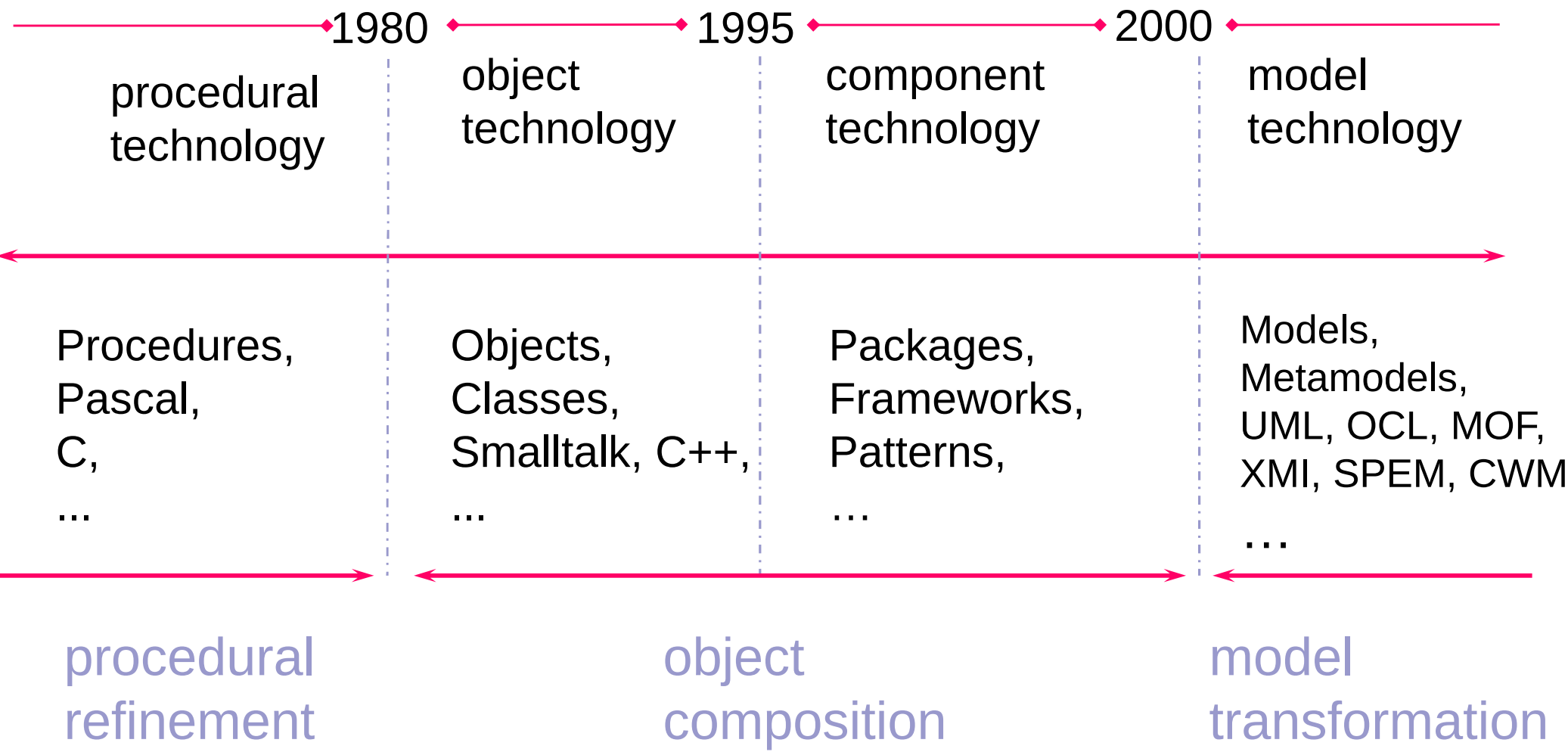
- (High level) parallel languages
  - PGAS, …
    - Not (yet) mature
- Platform oriented models
  - Multi-core ⇔ Threads, OpenMP
  - GPU ⇔ Cuda, OpenCL, OpenAPP
  - Multi-node ⇔ MPI
    - Many versions of the same code
    - Difficult to maintain all versions synchronized
    - Difficult to keep specific machine optimizations
    - Low code reuse

Proposed Approach Overview

- Separation of concerns
  - Machine specific code from re-usable code
    - Different algorithms!
- Make explicit points of configuration
  - Need a configurable representation of an application
- Generate machine specific version
  - Need a process

- Component model as an application description model to adapt to a particular machine

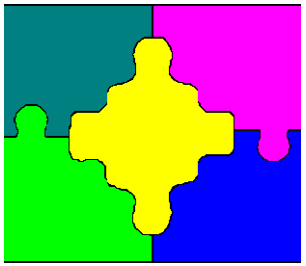
A global view of software engineering evolution



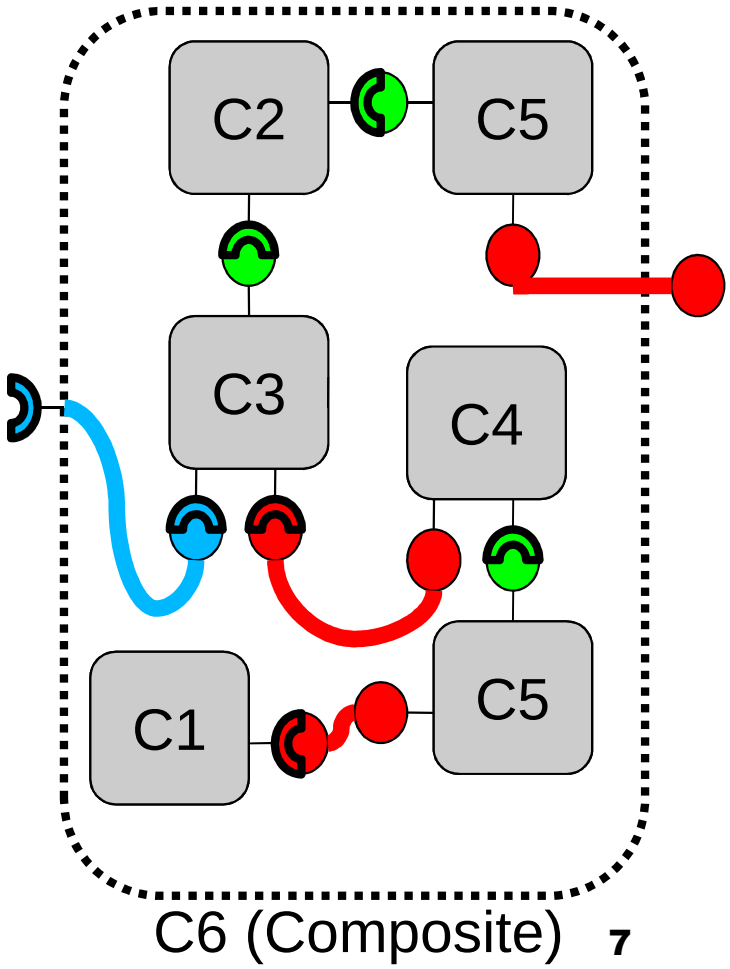
From Jean Bézivin presentation, ATLAS group (Inria & LINA), Nantes, France

OVEREVIEW OF COMPONENT MODELS

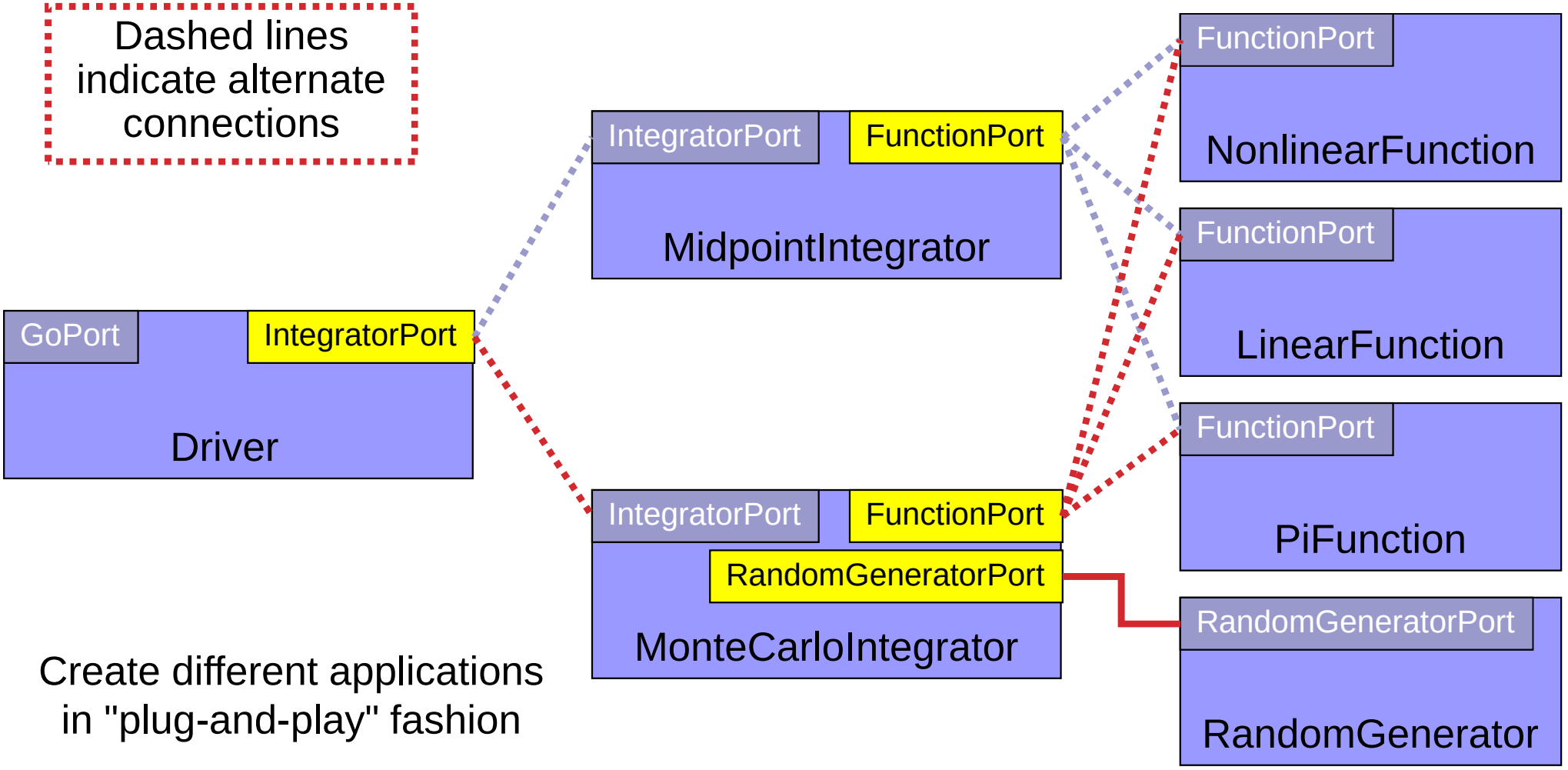
Software Component



- Technology that advocates for composition
  - Old idea (late 60's)
  - *Assembling* rather than *developing*
- Many types of composition operator
  - Spatial, temporal, ….
- Assembly of component
  - Primitive & composite components
- Many models (but in HPC)
  - CCA, Salome, CCM, Fractal, GCM, OGSi, SCA, …



Common Component Architecture (CCA) Example



Create different applications in "plug-and-play" fashion

From CCA Tutorial, http://www.cca-forum.org/tutorials/

Component in Parallel Computing

- Memory sharing between components
  - CCA & CCM Extensions
- Parallel components
  - CCA, SCIRun2, GridCCM
- Collective communications
  - CCM Extension
- Parallel method calls
  - SCIRun2, GridCCM
- Master / worker support
  - CCA & CCM Extensions
- Some algorithmic skeletons in assemblies
  - STKM

- Two type of features
  - Component **implementations**
    - ≈ skeletons
  - Component **interactions**

Limitation of Existing HPC Component Model

- Pre-defined set of interactions
  - Usually function/method invocation oriented
  - How to incorporate other interactions, eg MPI?
- Provide communication abstractions
  - Language interoperability (~IDL)
  - Network transparency
  - Potential overhead when not needed
  - Limited data types systems
    - Babel SIDL, OMG IDL, …
- Programming model vs execution model

Programming model vs Execution model

- L2C: Execution model
  - Performance oriented
  - Close to hardware
  - Not so easy to make use
- HLCM: « Programming » model
  - Assembly oriented
  - Abstract hardware
  - Shall be “easy” to make use

This talk

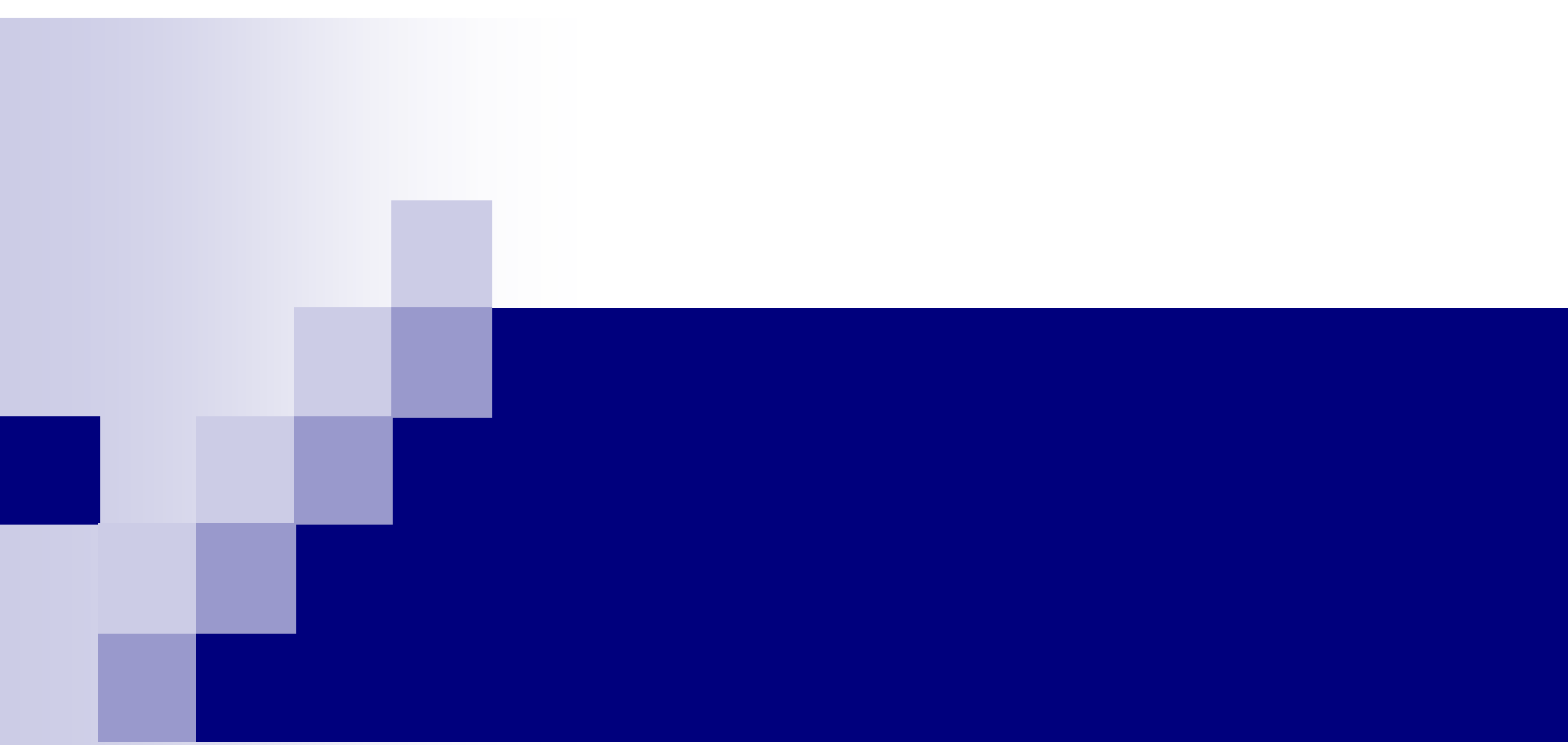
OVERVIEW OF L2C
LOW LEVEL COMPONENT

Low Level Component Model

- A minimalist component model for HPC
  - Component creation/deletion, configuration, and connection
  - An (optional) launcher
- No L2C code between components @ runtime
- Support native interactions
  - C++, MPI, CORBA, FORTRAN (2008)
- Extensible
- LGPL, available at *hlcm.gforge.inria.fr*

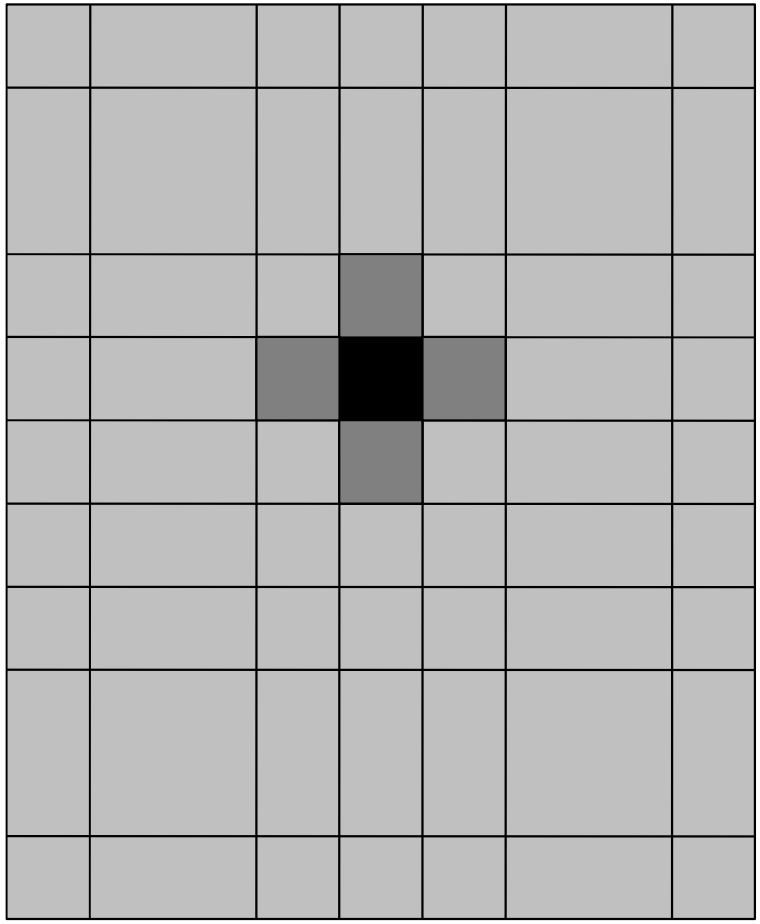
L2C: Connector Overview

- C++/FORTRAN Interactions
  - Use/Provide relationships
  - No language interoperability
    - Outside L2C goals
- MPI Interactions
  - Connector ~ communicator

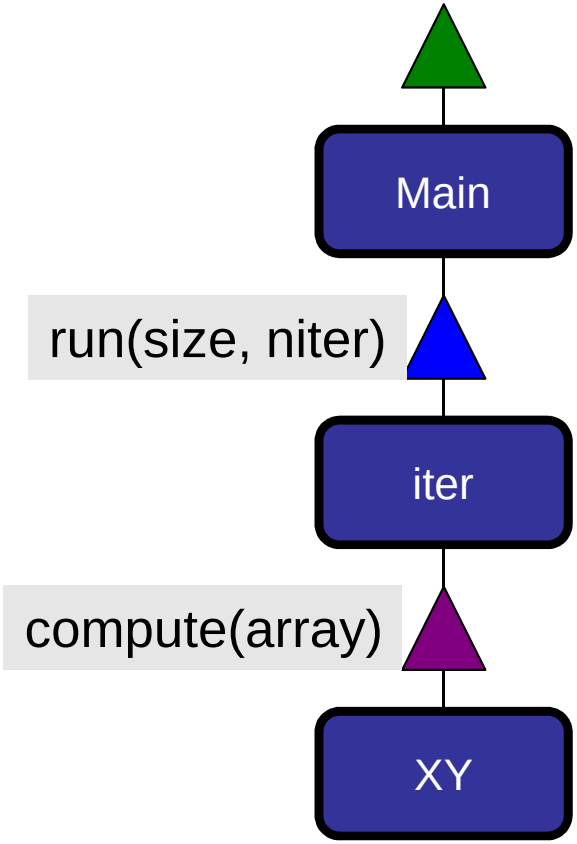


L2C AND JACOBI

Jacobi Sequential Computation



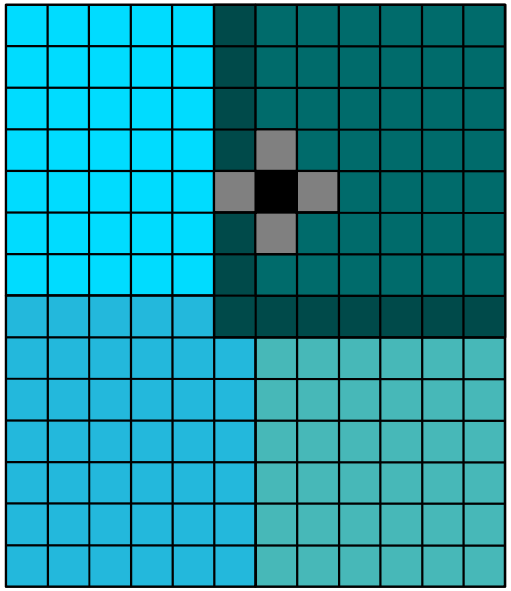
Iter = N



- For iter = 0 to Niter
  - For y = 0 to ymax
    - For x = 0 to xmax
      - tab[iter][x][y] = ...

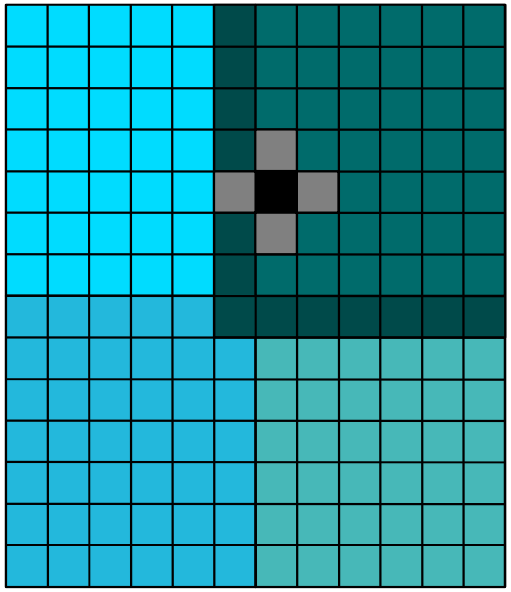
Thread Jacobi Parallelization

- 1 shared array
- Barrier after each iteration



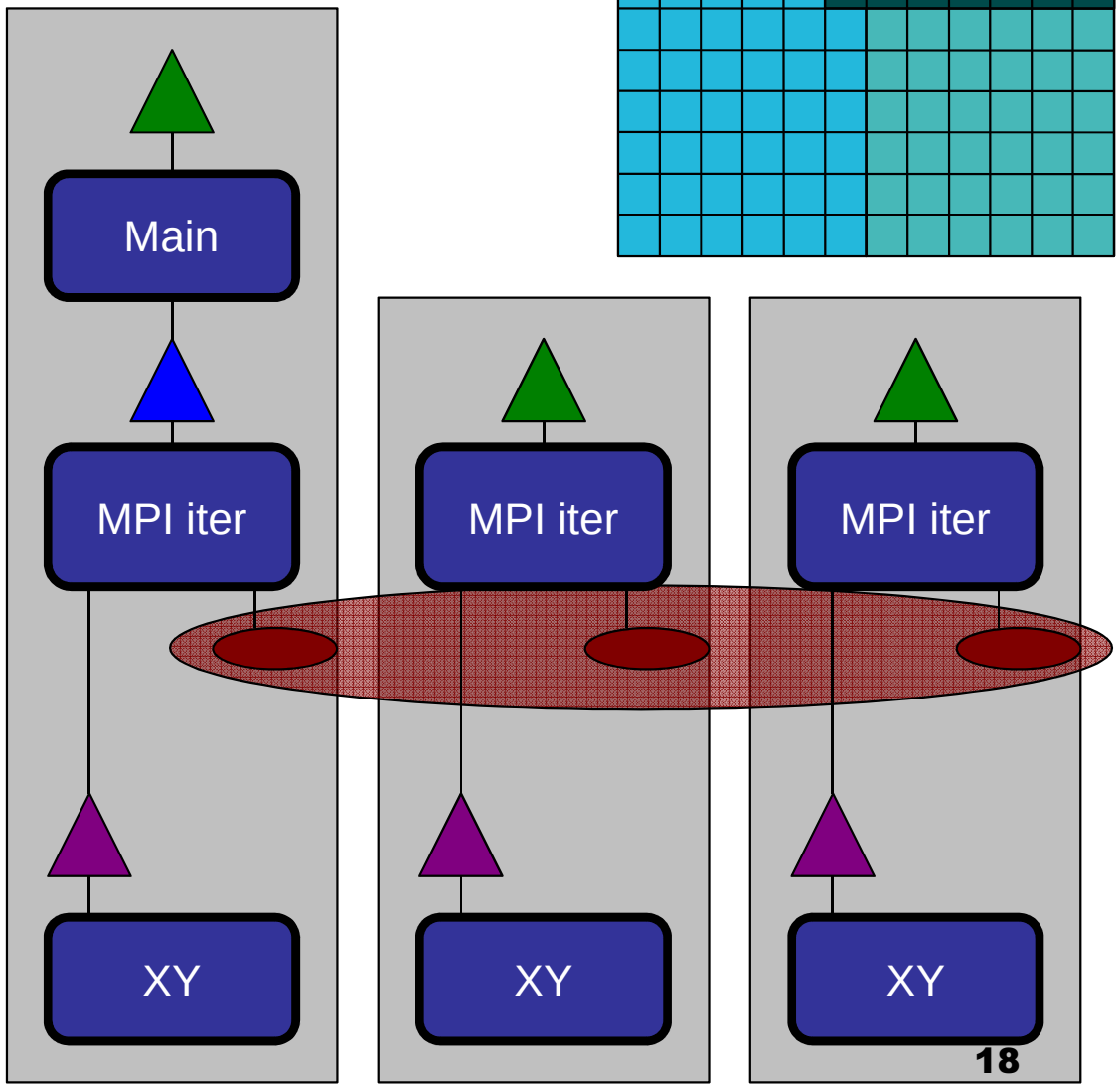
- For iter = 0 to Niter
  - For y = 0 to ymax
    - For x = 0 to xmax
      - tab[iter][x][y] = …
  - Barrier

MPI Jacobi Parallelization



- 1 local array per thread
- Send/receive at each iter

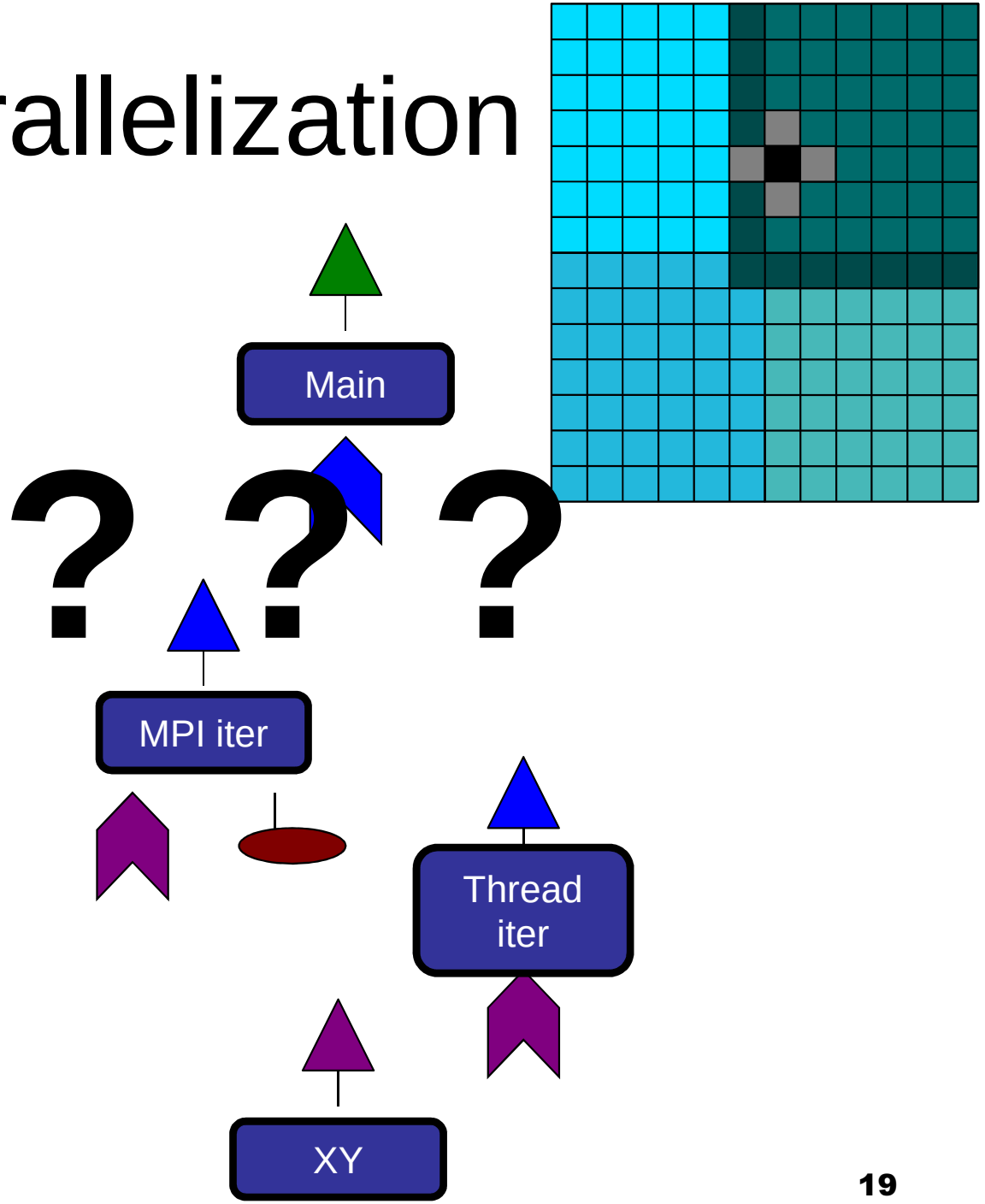
- For iter = 0 to Niter
  - For y = 0 to ymax
    - For x = 0 to xmax
      - tab[iter][x][y] = …
  - SendReceive



Hierarchic Parallelization

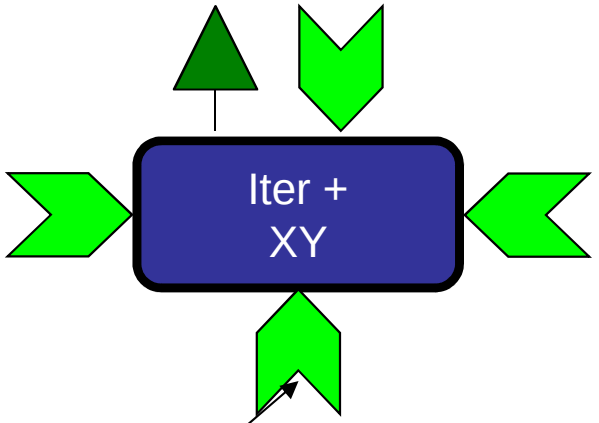
- Multi nodes
  - MPI
- Multi core
  - Threads

- For iter = 0 to Niter
  - For y = 0 to ymax
    - For x = 0 to xmax
      - tab[iter][x][y] = …
  - Local Barrier
  - SendReceive



The 4 connector way

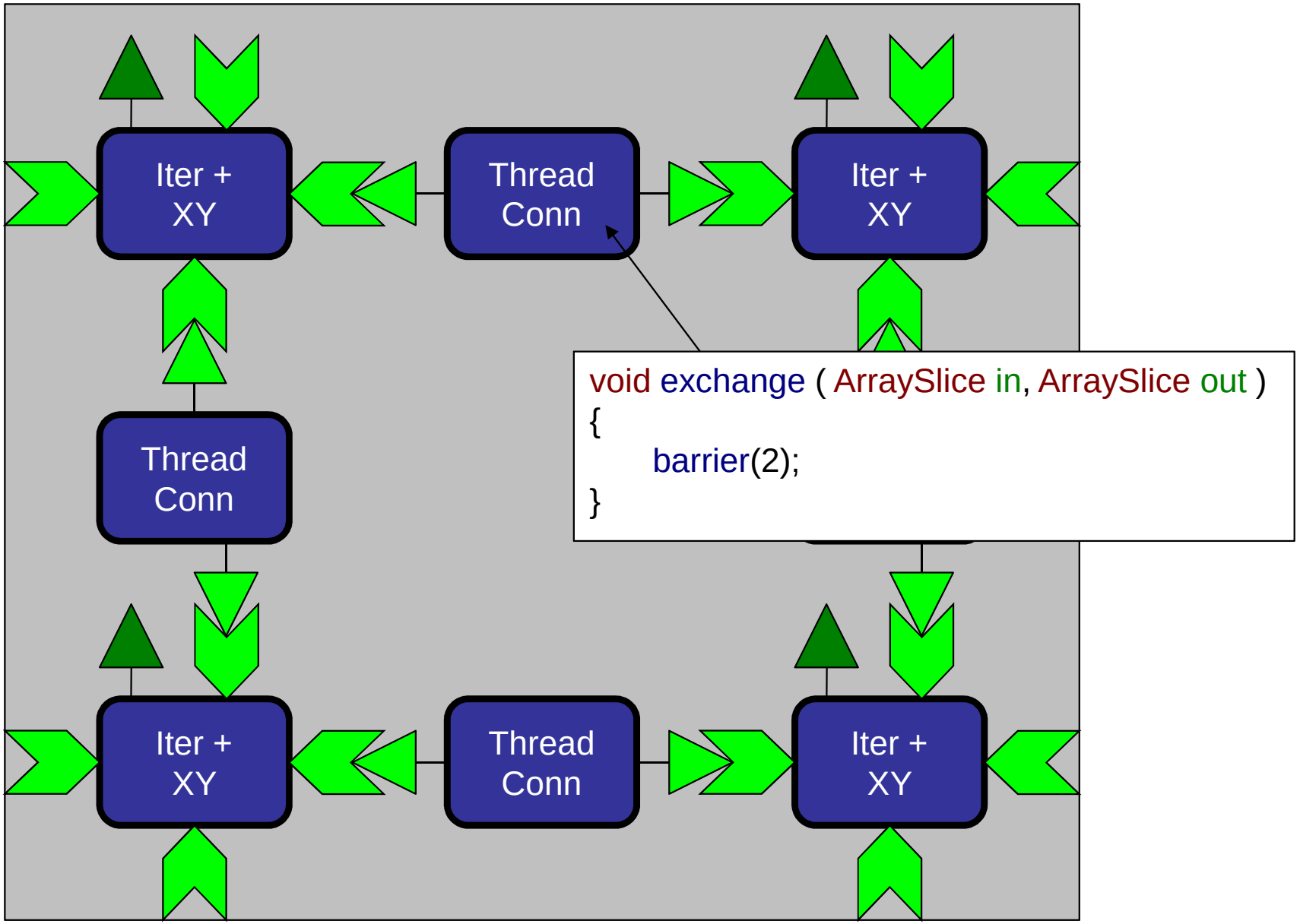
- 1 connector instance
  - 1 domain
- 1 DataExchange/side
  - Implementation agnostic interface



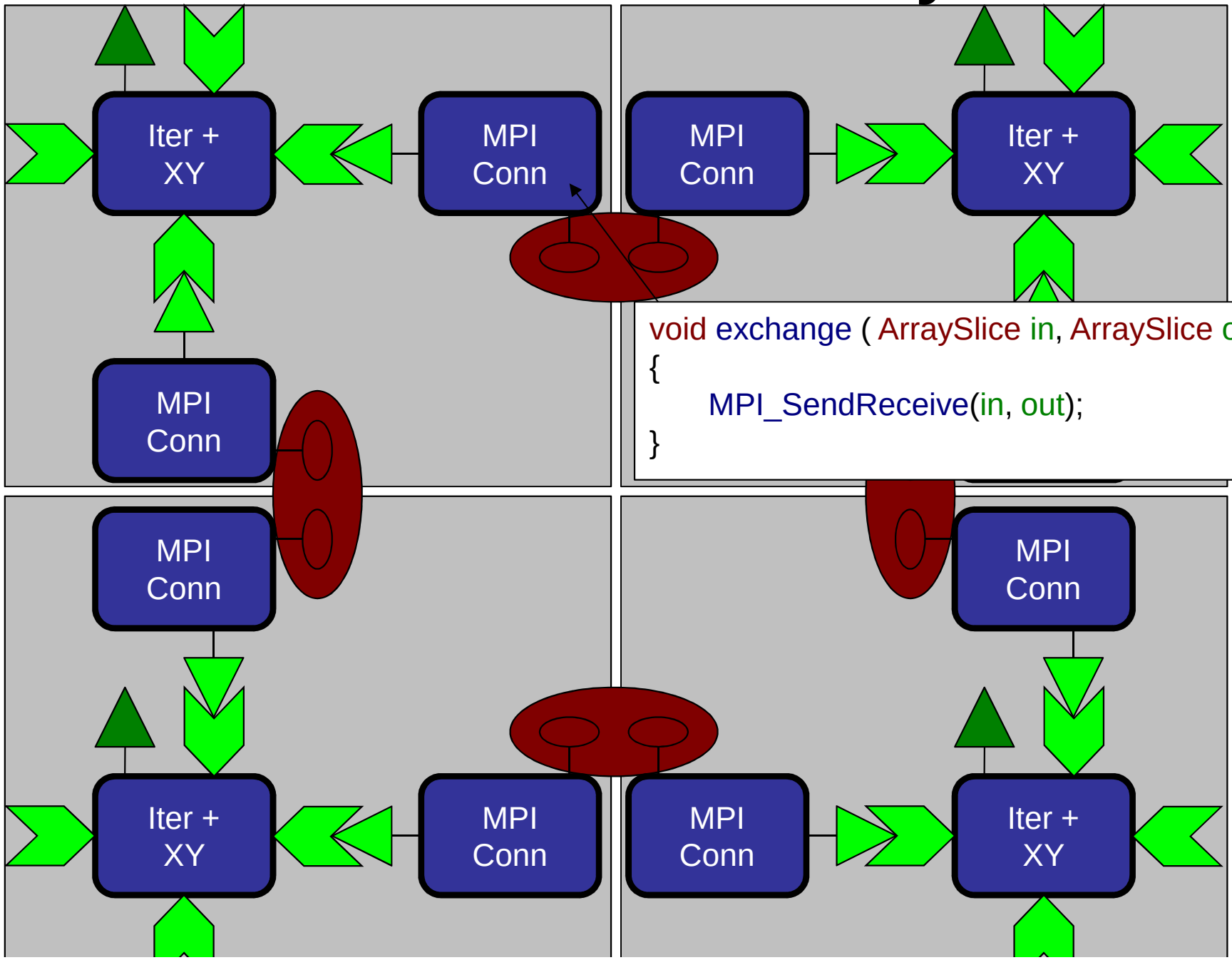
- For iter = 0 to Niter
  - Wait for frontier
  - For y = 0 to ymax
    - For x = 0 to xmax
      - tab[iter][x][y] = …
  - Data update T/B/L/R

```
class DataUpdate
{
public:
      virtual void exchange (
                  ArraySlice in,
                  ArraySlice out
      ) = 0;
};
```

The 4 Connector Way: Threads

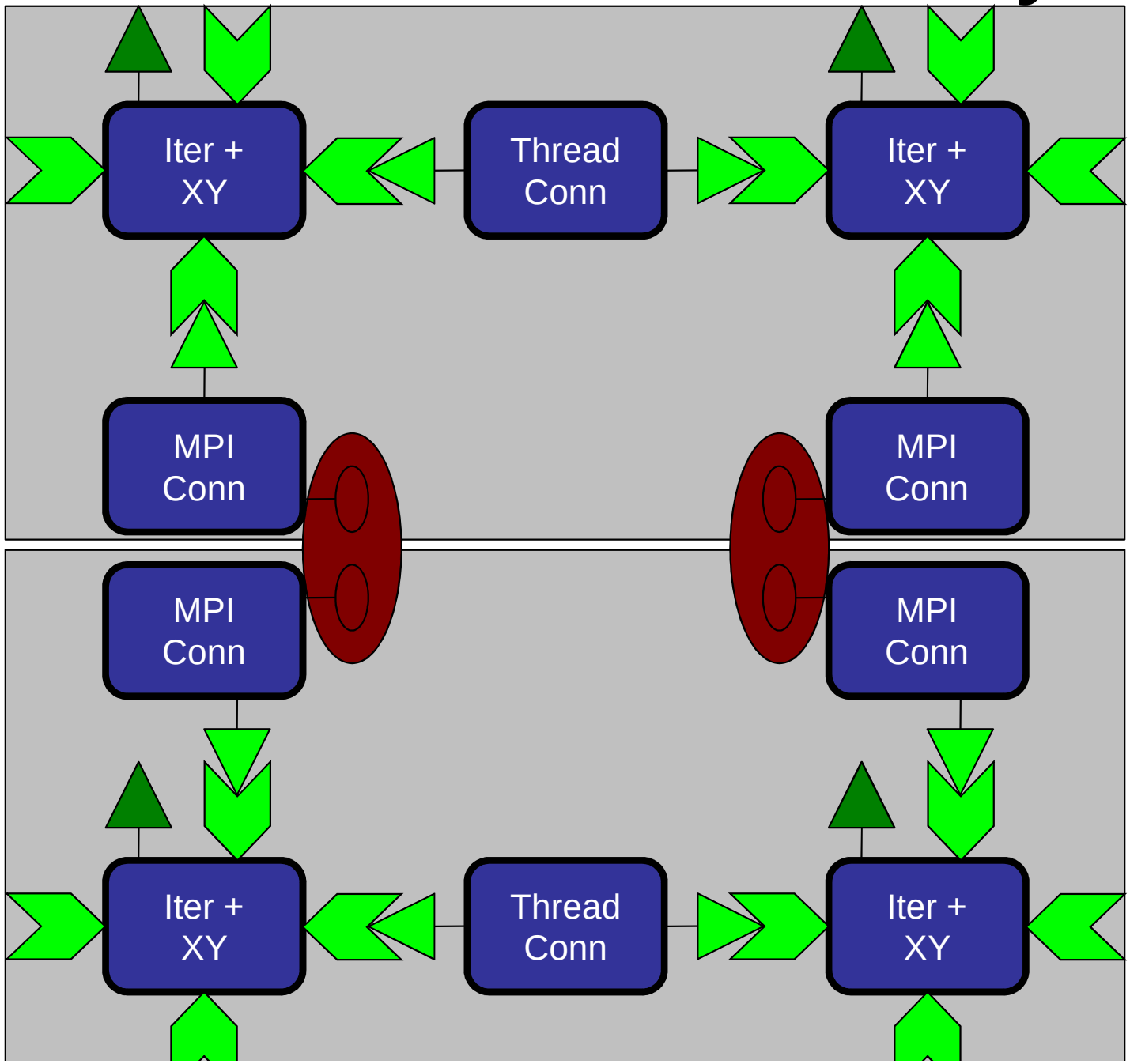


The 4 Connector Way: MPI



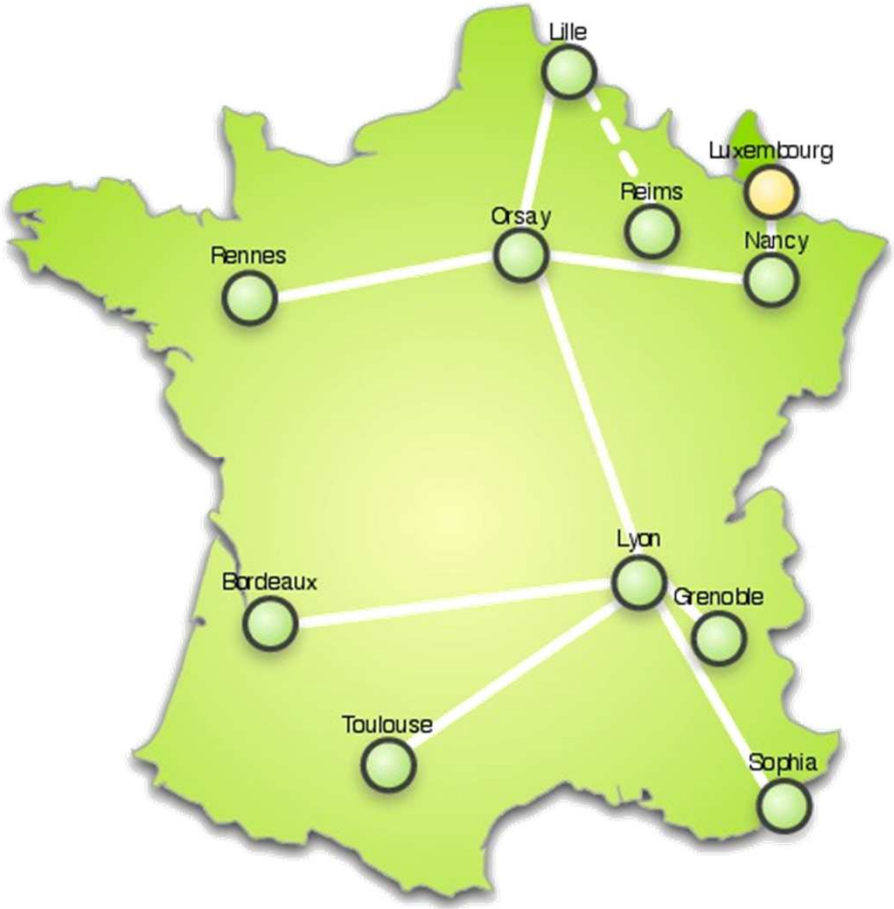

```
void exchange ( ArraySlice in, ArraySlice out )
{
      MPI_SendReceive(in, out);
}
```

The 4 Connector Way: Hierarchy

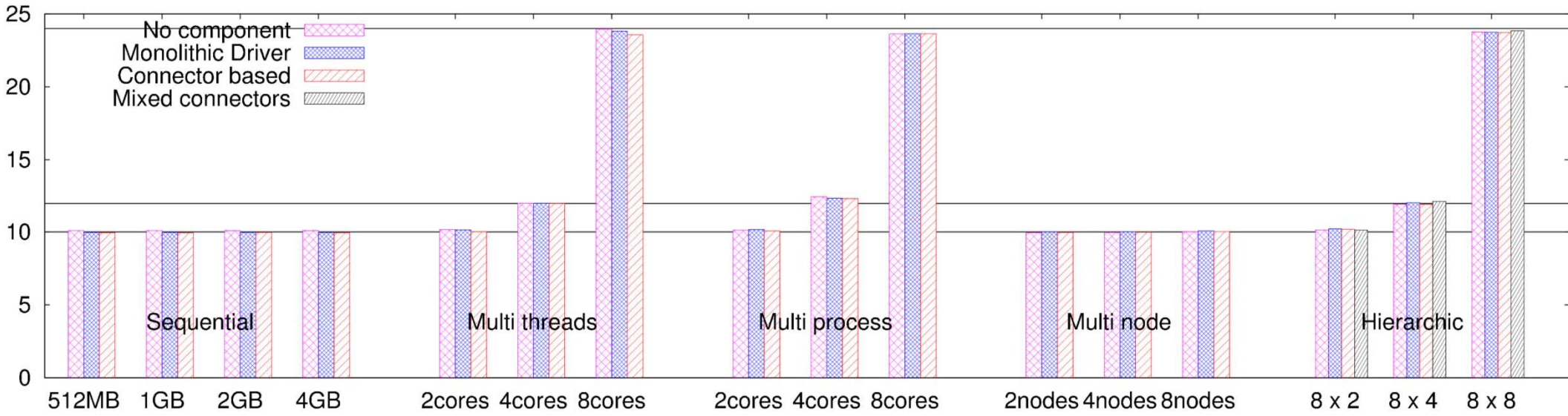


Experimental Platform: Grid'5000

- Griffon cluster
  - Intel Xeon L5420 2.5 GHz
    - 4 cores per CPU
    - 2 CPU per node
  - 92 nodes
  - 16 GB RAM
  - Infiniband-20G network



Iteration Time



Overhead coming from using too much threads on this machine!

- Limited memory bandwidth

Software Complexity

- Number of Lines

Jacobi Version	Native	Driver	Connector
Sequential	**161**	239	**388**
Multithreaded	**338**	386	**643**
MPI	**261**	285	**446**

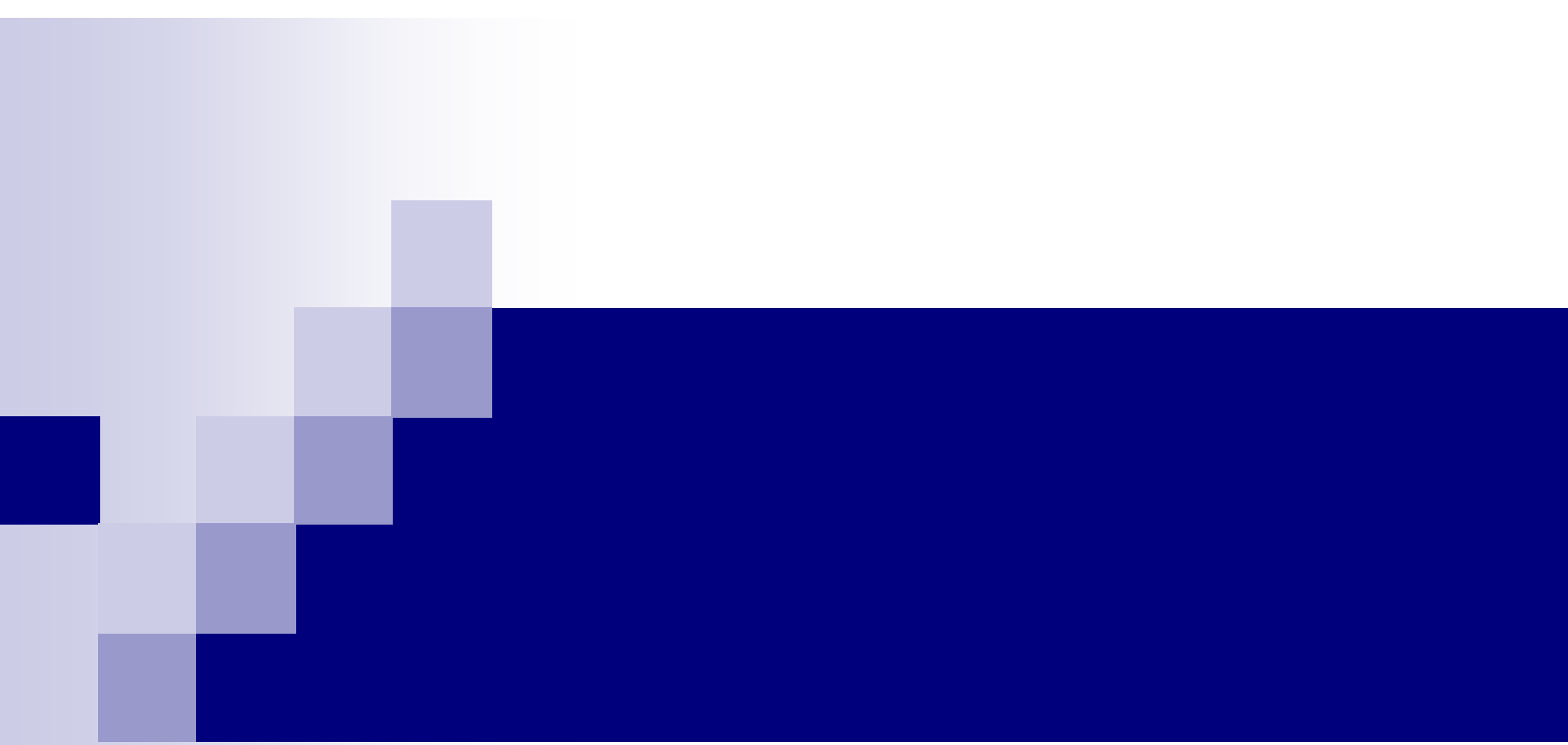
- Code Reuse

Code Reuse vs Seq (%)	**Driver**	**Connector**
Thread	26%	**31%**
MPI	32%	**87%**
MPI+Thread	-	**100%**

Software Complexity

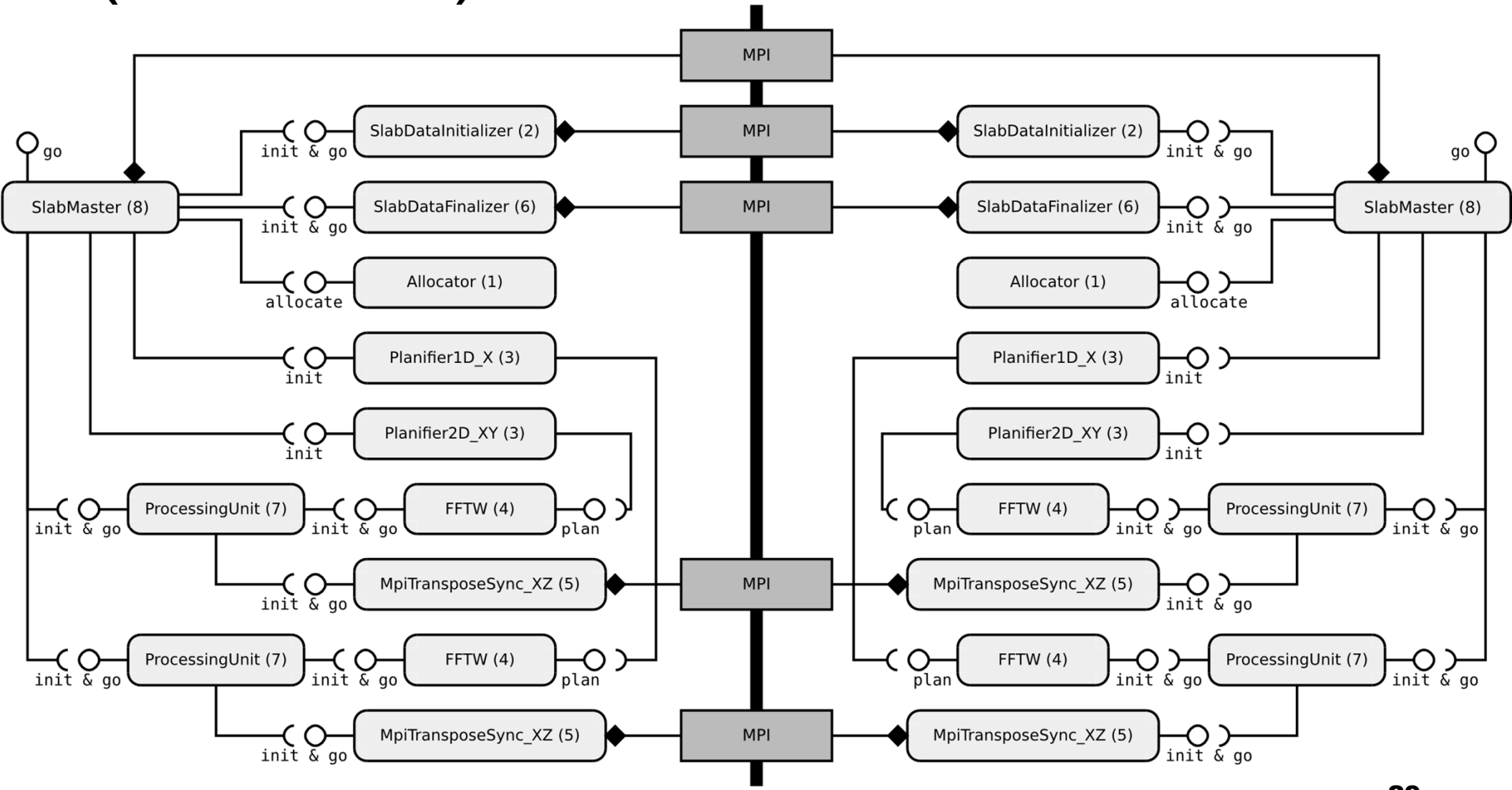
- Cyclomatic complexity
  - *It directly measures the number of linearly independent paths through a program's source code. Wikipedia*

Jacobi Version	Native	Driver	Connector
Sequential	**28**	**32**	**8**
Multithreaded	**76**	41	**26**
MPI	**55**	22	**13**

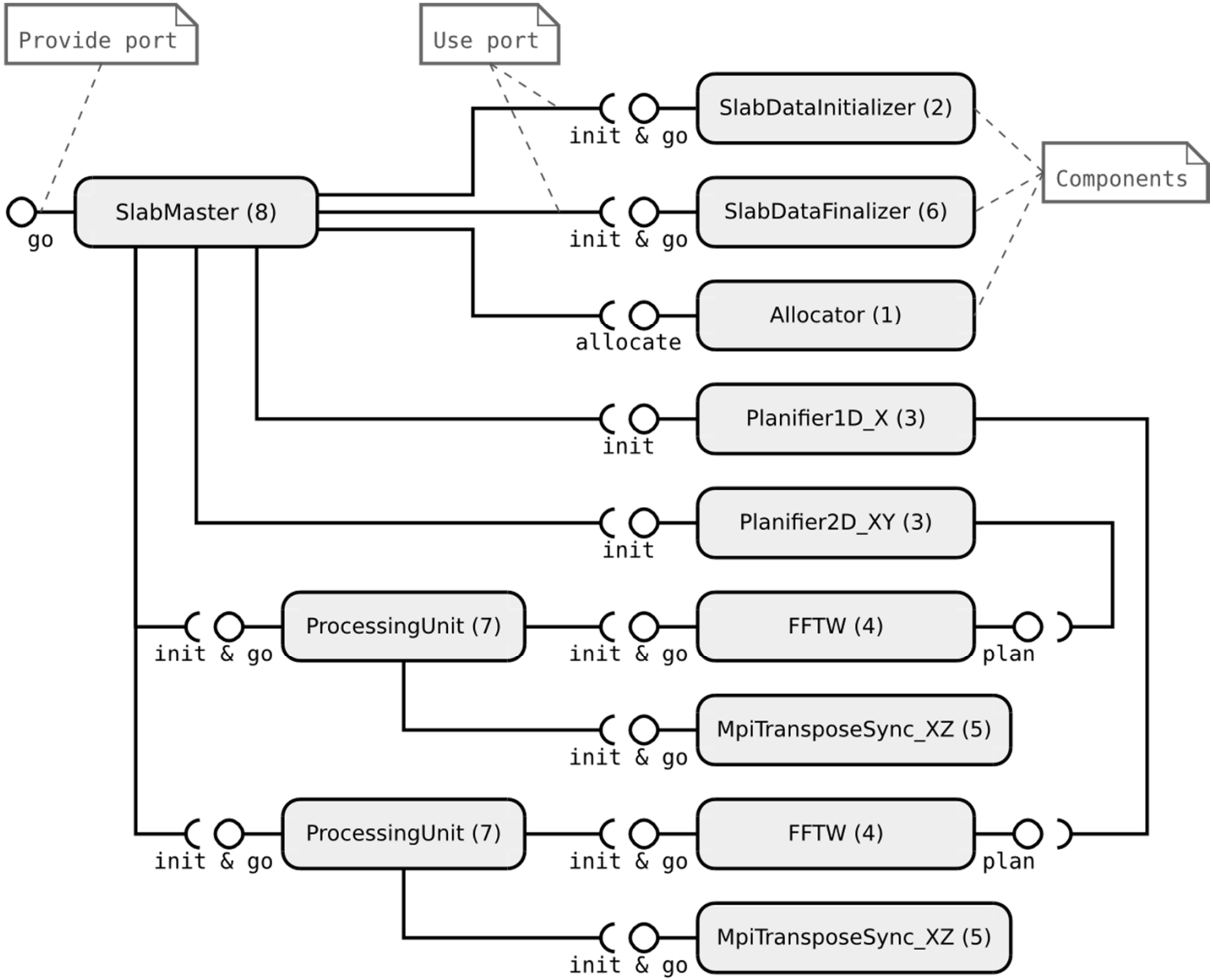


L2C AND 3D FFT

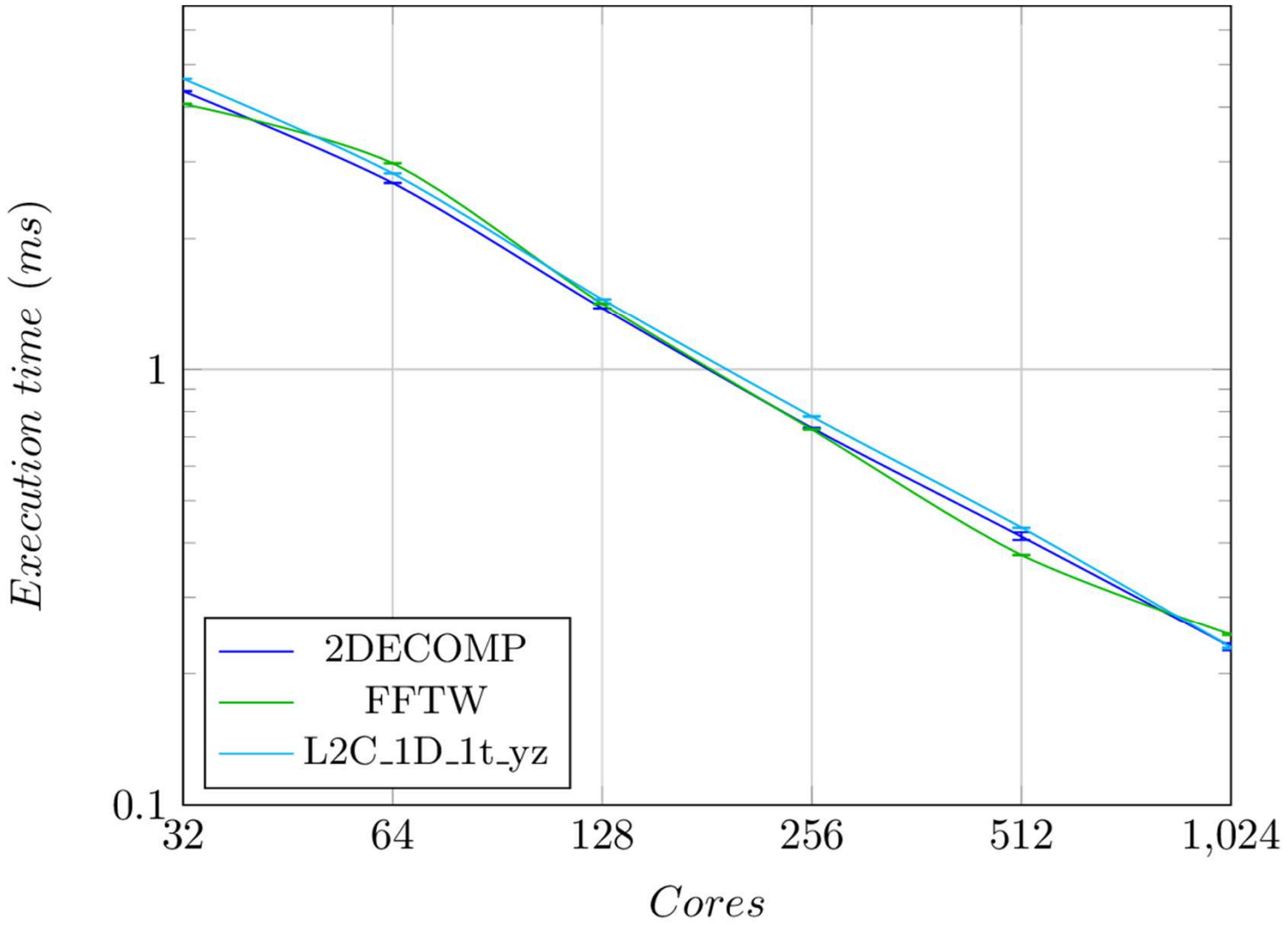
1D MPI 3D FFT Assembly (2 nodes)



1D MPI 3D FFT Assembly

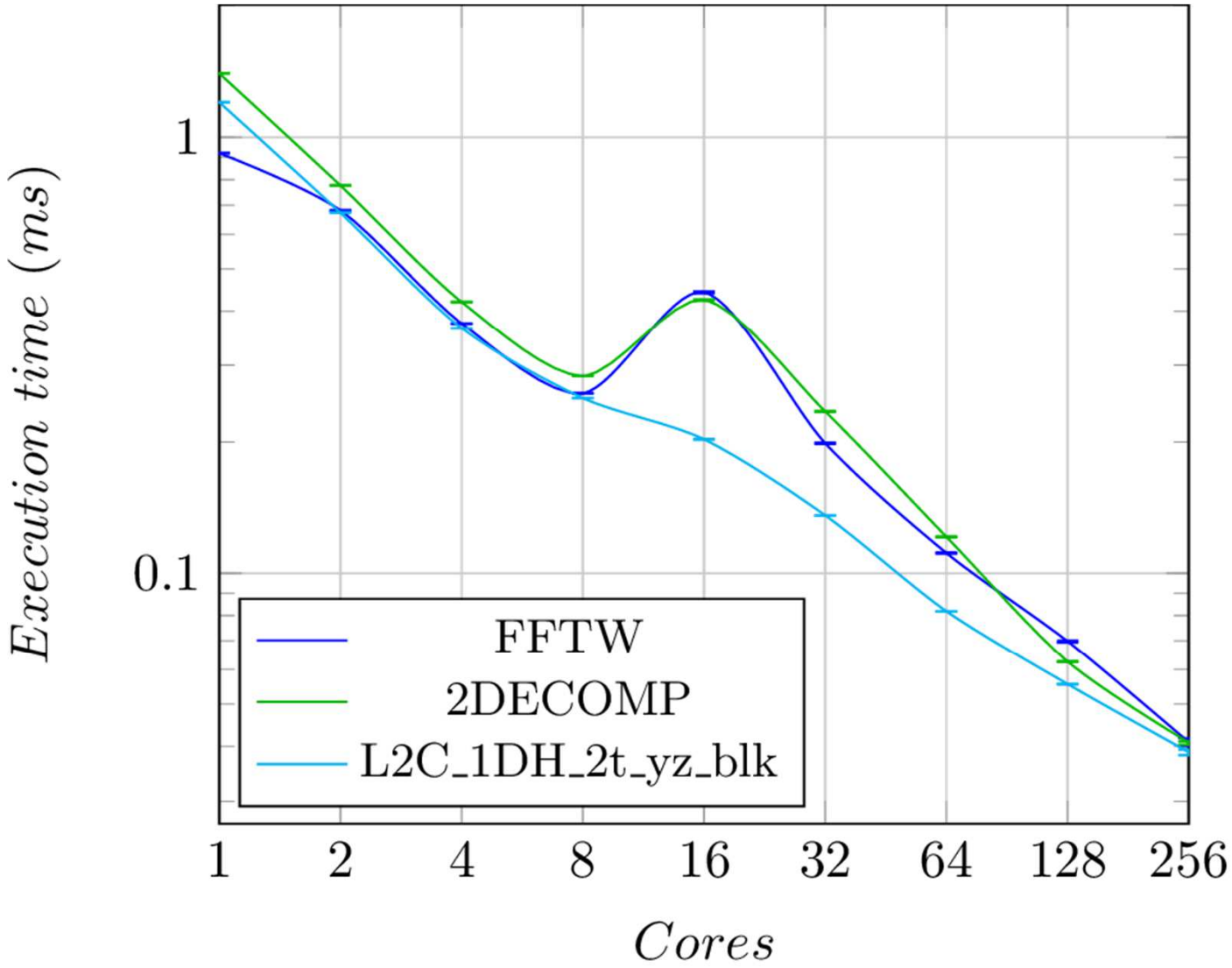


Homogeneous Experiments



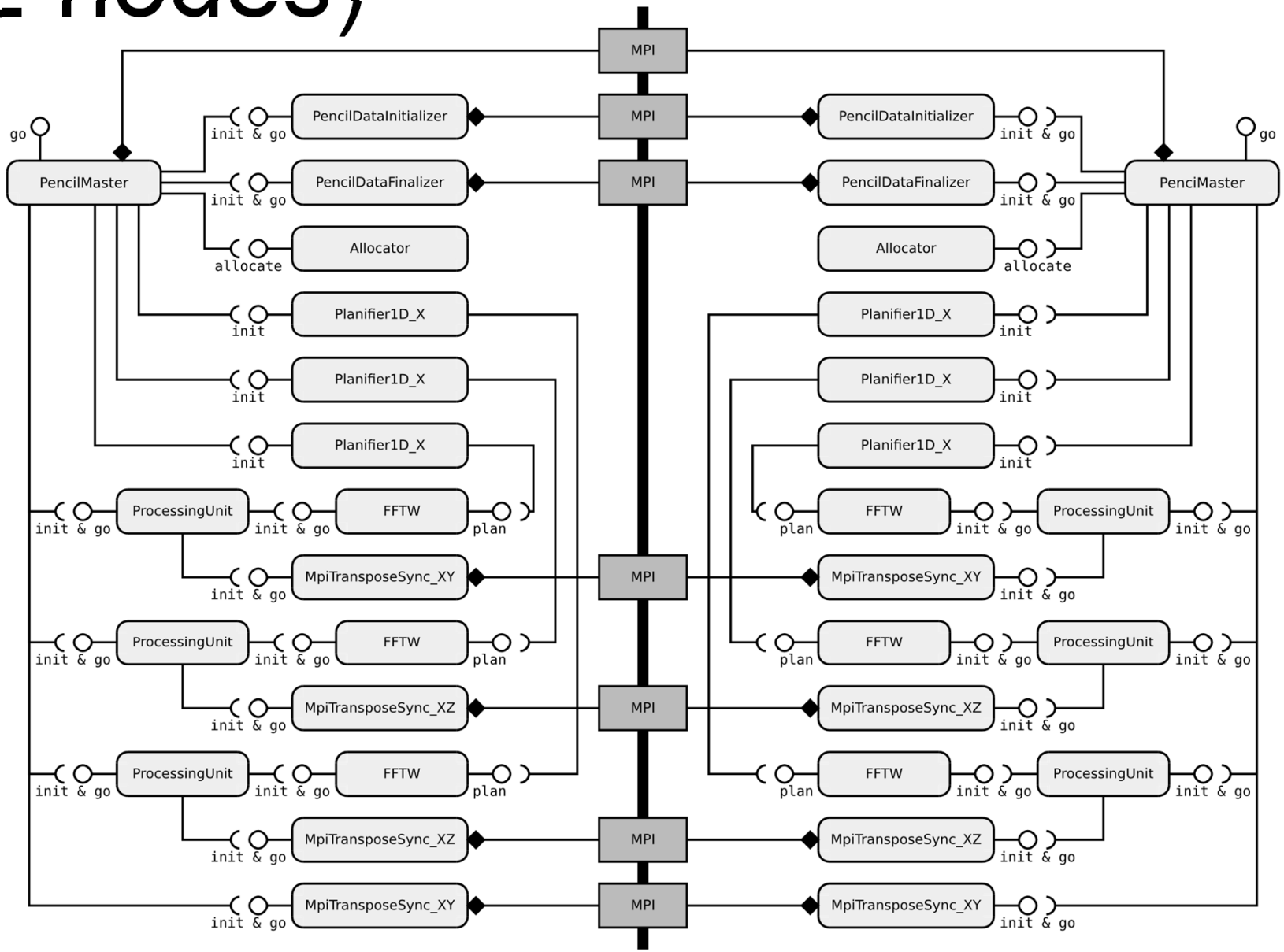
1024^3 3D FFT – 1D Decomposition – Curie (Thin node)

Heterogeneous Experiments

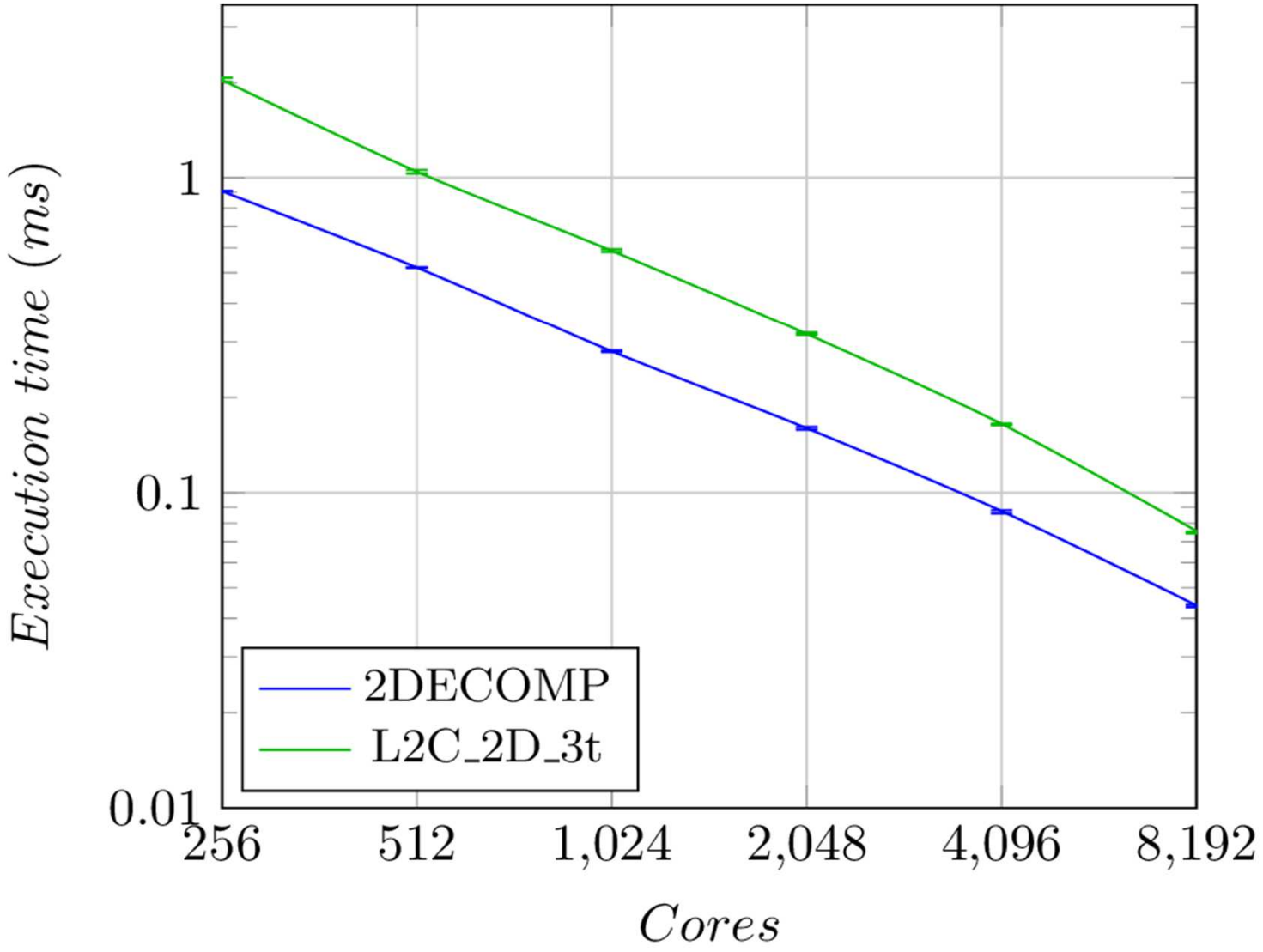


256^3 3D FFT – 1D Decomposition : Edel+Genepi Cluster (Grid'5000)

2D MPI 3D FFT Assembly (2 nodes)



Homogeneous Experiments



1024^3 3D FFT – 2D Decomposition – Curie (Thin node)

Number of Lines & Reusability

Version	C++ Lines of Code	% Reused Code
L2C 1D 2t xz	927	-
L2C 1D 1t xz	929	77%
L2C 1D 2t yz	929	100%
L2C 1D 2t yz blk	1035	69%
L2C 1DH 1t yz	983	80%
L2C 1DH 2t yz blk	1097	72%
L2C 2D 3t	1067	87%
L2C 2DH 3t	1146	69%

Conclusion & Perspectives

- Component model as a way to handle versions
  - Application adaptation => assembly modification
- L2C
  - A simple, efficient, and extensible model
- Towards component + task graph (L2C+StarPU)
- Efficient reconfiguration of component (on going)
- L2C assembly complex to write
  - Shall be generated by a higher model
  - HLCM: A high level component model
- Transformation algorithms from “HLCM” to “L2C”

Component and Task Graph

- Component
  - Good for describing application structure
- Task graph
  - Efficient to handle task dependencies
- Towards a Component+Task graph model
  - Runtime/Avalon/CEA PhD starting Nov 1.
  - Superseding L2C+StarPU
  - Gysela5D as a target application