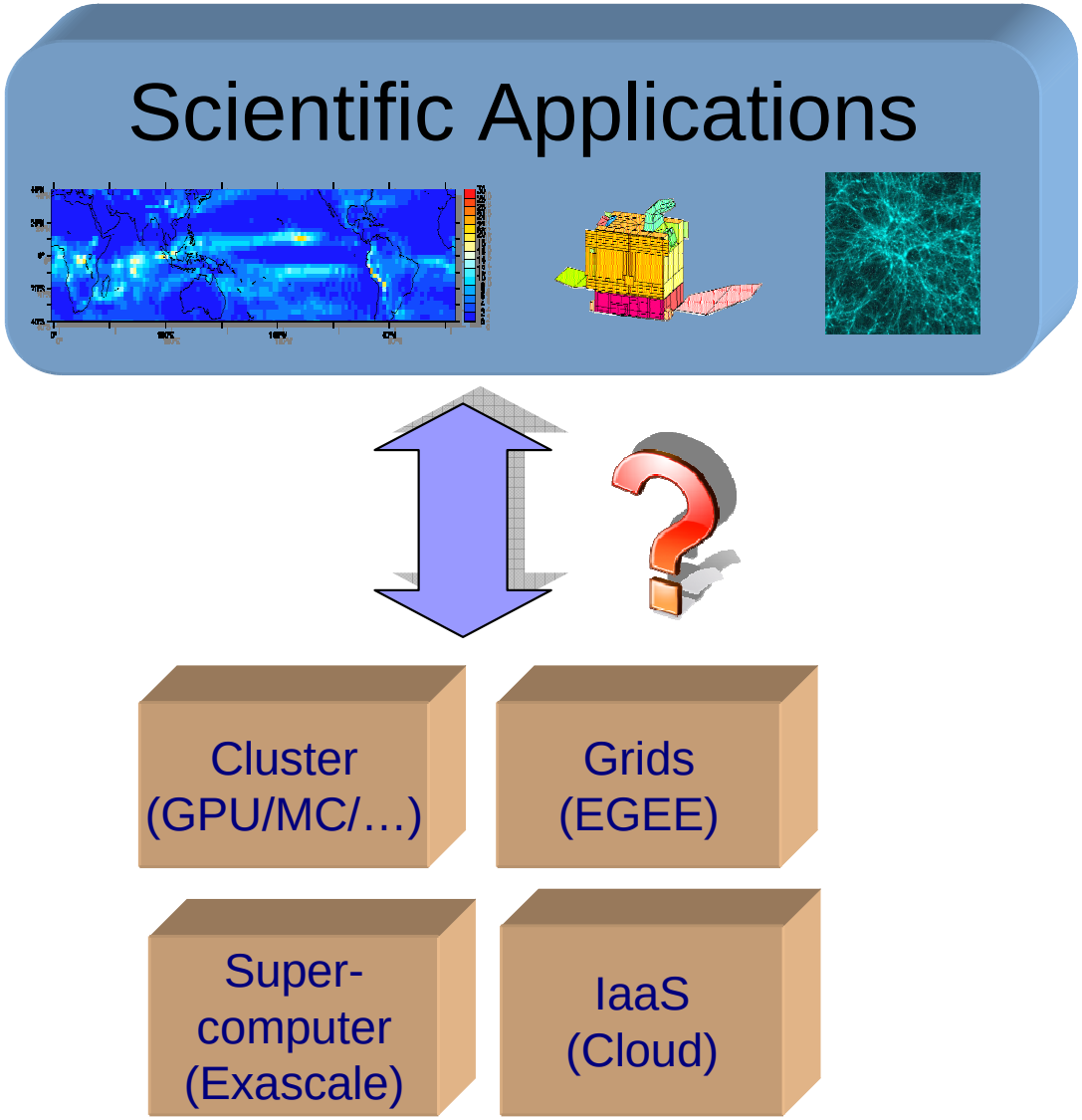
A Low Level Component Model enabling Performance Portability of HPC Applications

Julien Bigot (INRIA/CEA)
Christian Perez (INRIA)
Vincent Pichon (EDF R&D)
Zhengxhion Hou (INRIA)
Avalon Reseach Team, LIP, Lyon, France

5th International Workshop on Multi-Core Computing Systems (MuCoCoS)
November 16th 2012

Context


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

Programming a parallel applications

- (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

Proposed Approach Overview

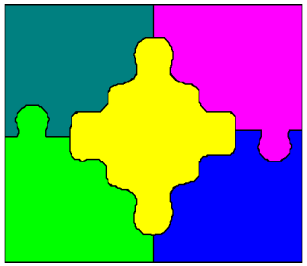
- Separation of concerns
  - Machine specific code from re-usable code
- Make explicit points of configuration
  - Need a configurable representation of an application
- Generate machine specific version
  - Need a process

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

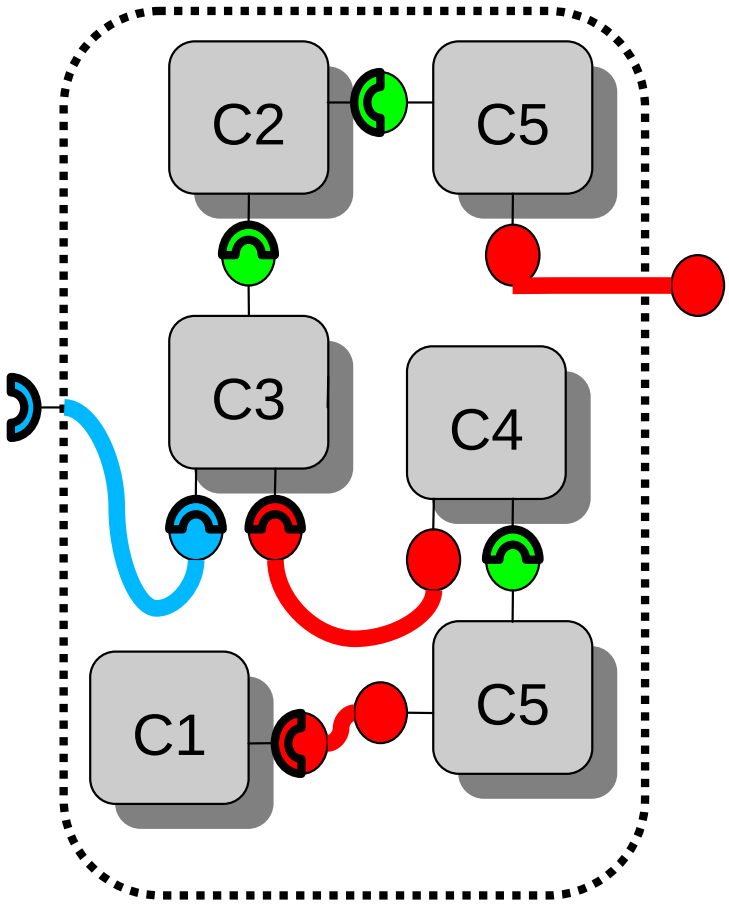
Content

- *Context*
- **Overview of component models**
- L2C: Low Level Component
- Jacobi & L2C
- Experimental evaluation
- Conclusion & Perspectives

Software Component

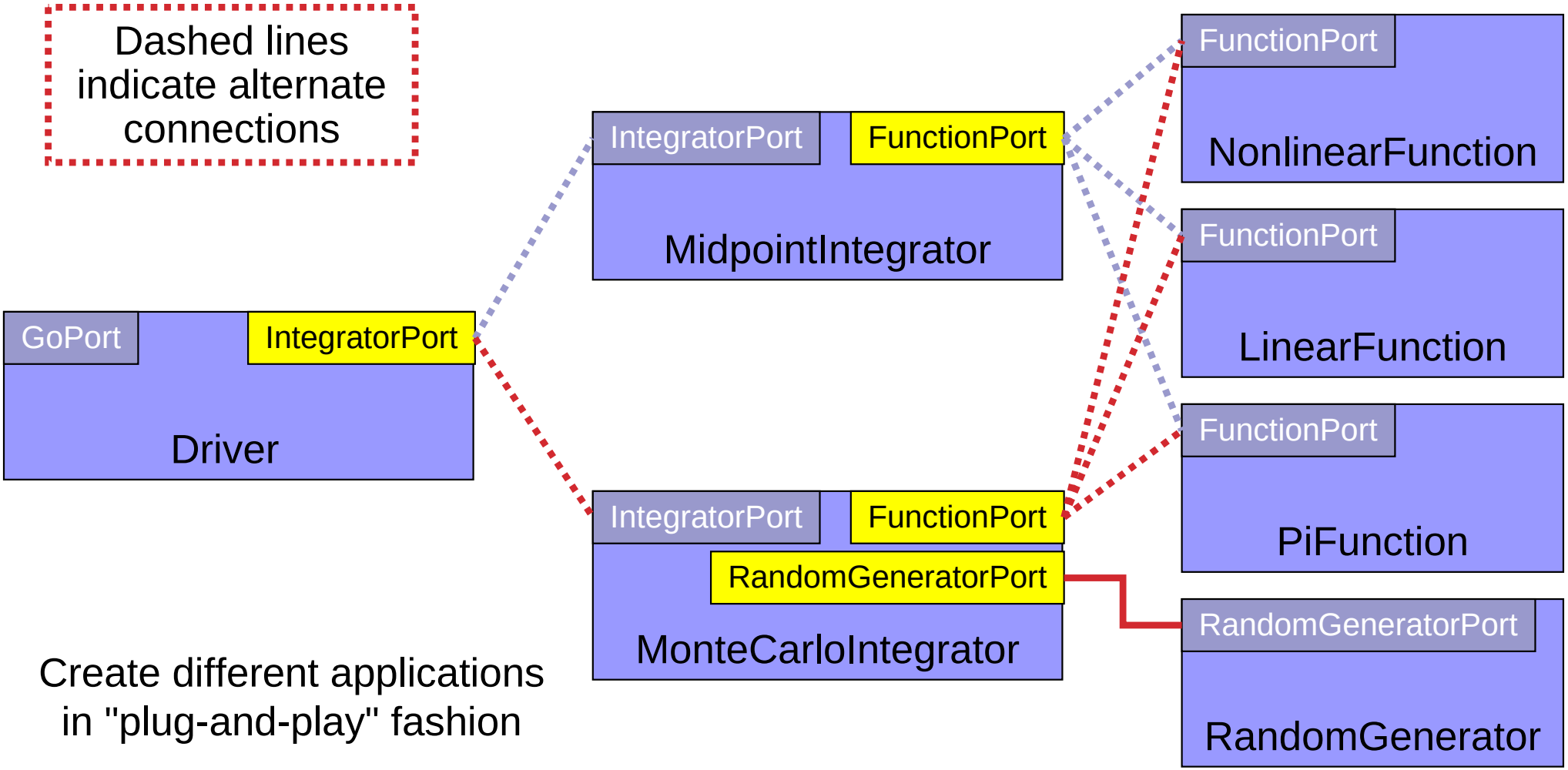


- Technology that advocates for composition
  - Old idea (late 60's)
  - *Assembling* rather than *developing*
- Many types of composition operator
  - Spatial, temporal, ….
- Assembly of component
  - Primitive & composite components
- Many models
  - CCA, Salome, CCM, Fractal, GCM, OGSi, SCA, …



C6 (Composite)

CCA Example



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

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

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 abstraction
  - Language interoperability (~IDL)
  - Network transparency
  - Potential overhead when not needed
  - Limited data types systems
    - Babel SIDL, OMG IDL, …
- Programming model vs execution model

Content

- *Context*
- *Overview of component models*
- **L2C: Low Level Component**
- Jacobi & L2C
- Experimental evaluation
- Conclusion & Perspectives

Low Level Component Model

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

A Minimal L2C Component

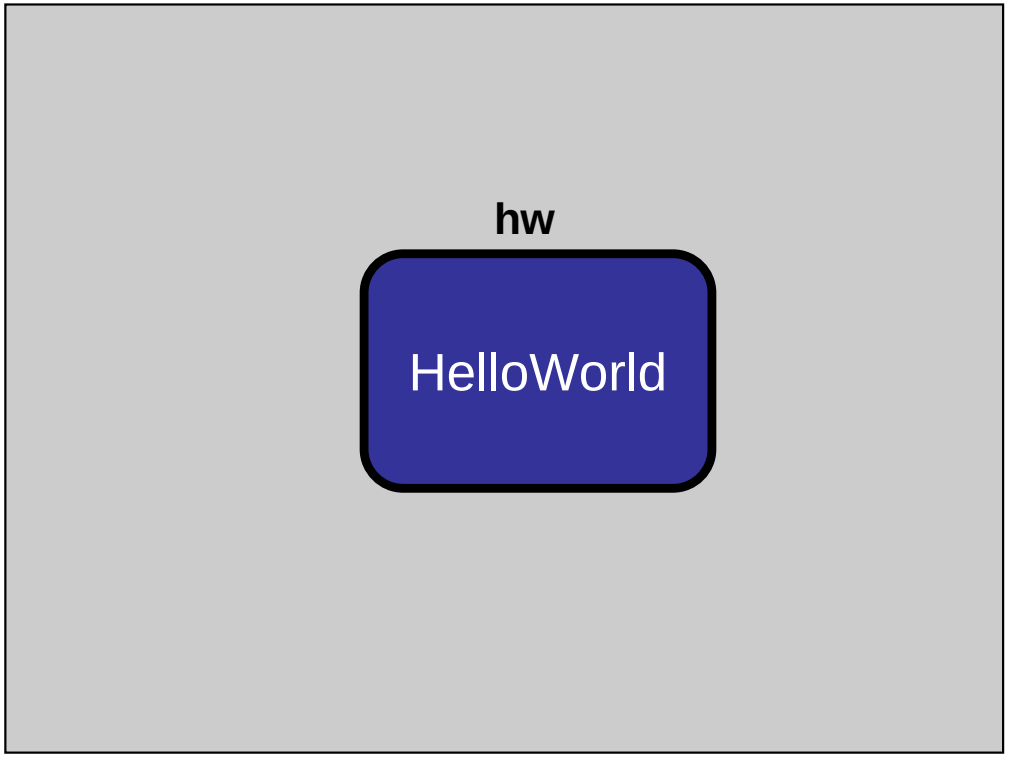
- A component is a class
  - HelloWorld
- A few macros make it a component
  - LCMP … LEND
- No library dependency (only header)

```
class HelloWorld
{
public:
	HelloWorld ()
	{
		cerr << "HelloWorld created"<<endl;
	}

	~HelloWorld ()
	{
		cerr <<"HelloWorld destroyed"<<endl;
	}
};

LCMP(HelloWorld)
LEND
```

A Minimal L2C Assembly

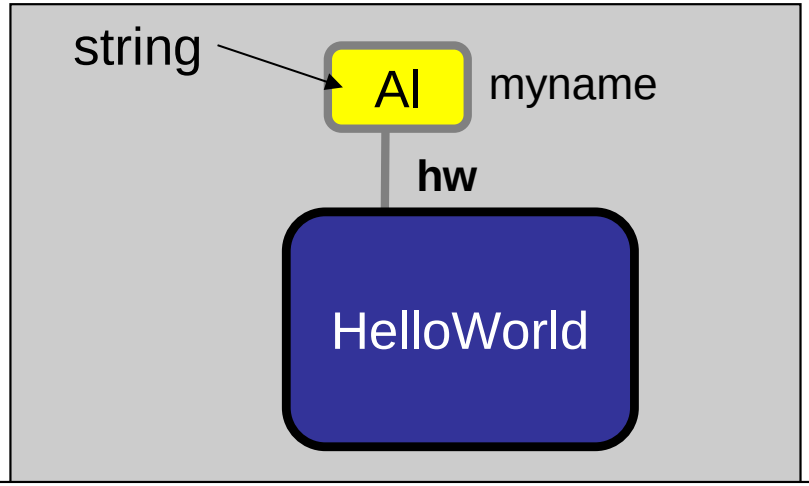
- Assembly
  - Simple XML file
  - Set of instance
- Instance
  - Name (id)
  - Type (class name)

```
<lad>
    <instance id="hw" type="HelloWorld"/>
</lad>
```



Configuration properties

- A type
  - string, int, complex, double, ...
- A name
- Implementation
  - Public member variable
  - L_PROPERTY macro
- Value set in Assembly

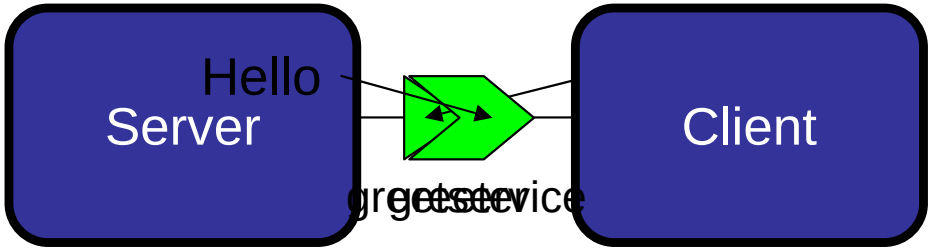



```
class HelloWorld
{
public:
    string myname;

<las>
    <instance id="hw" type="HelloWorld">
        <property id="myname">
            <value type="string">Al</value>
        </property>
    </instance>
</lad>

    L_PROPERTY(string, myname);
LEND
```

C++ Interactions between components

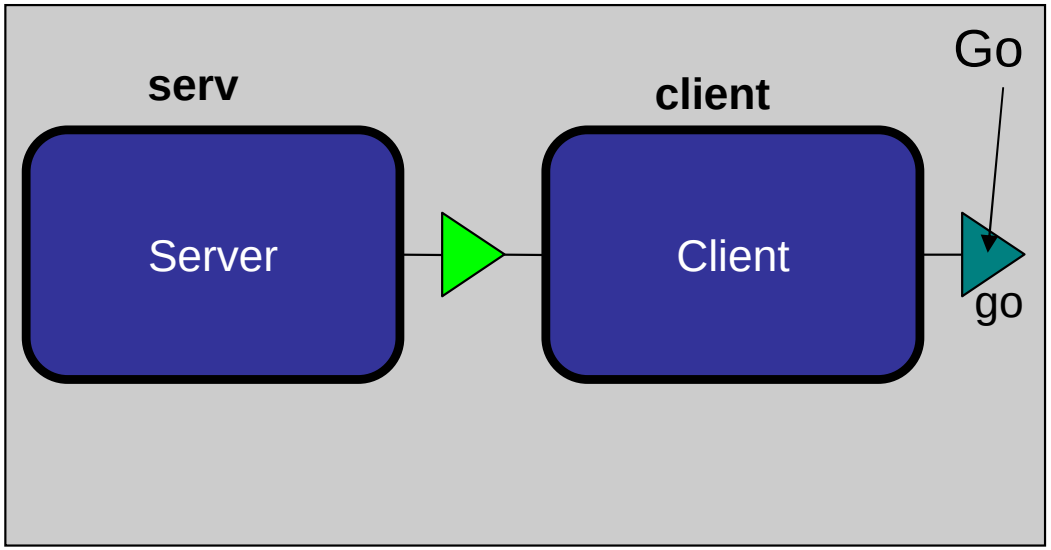
- Method call interactions
  - Typed by an “interface”
- Provide the service
  - Inheritance
  - L_CPP_PROVIDE macro
- Use the service
  - Pointer import
  - L_CPP_USE macro

```
class Client
{
public:
        Hello* greetservice;

};

LCMP(Client)
        L_CPP_USE(Hello, greetservice);
LEND
```



C++ Interactions in the Assembly

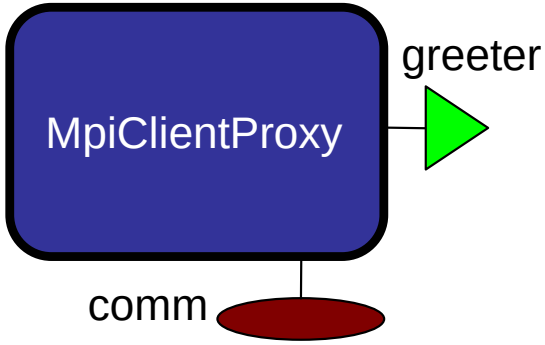
- Connection
  - Set USE property
  - Reference PROVIDE property



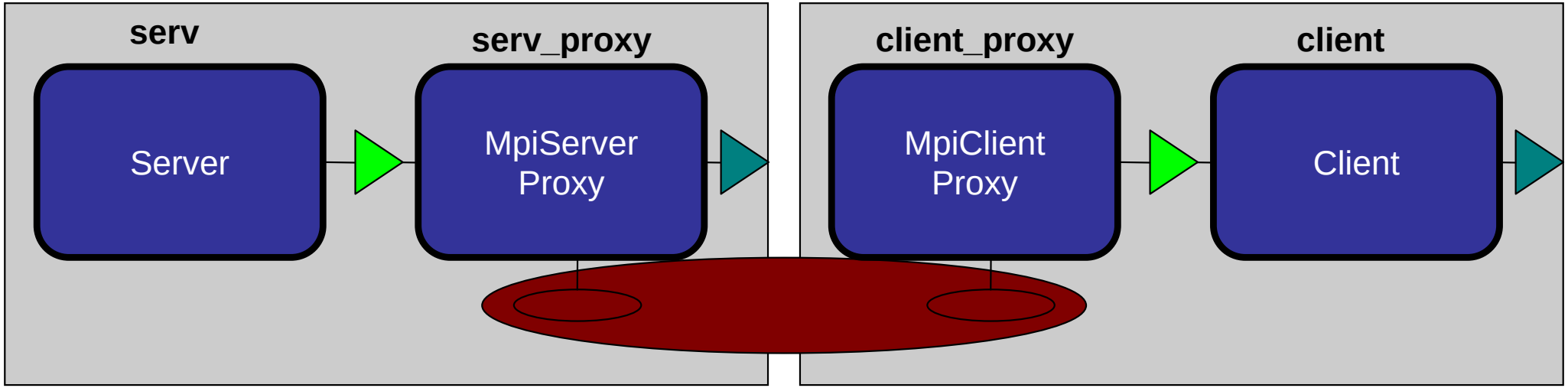

```
<lad>

    <instance id="serv" type="Server" />

    <instance id="client" type="Client" >
        <property id="greetservice"><propref instance="serv" property="greeter"/></property>
        <start property="go"/>
    </instance>

</lad>
```

MPI Interactions

```
class MpiClientProxy :
        virtual public Hello
{
public:
    MPI_Comm comm;

    virtual void greet (string name)
    {
        unsigned long size = name.size()+1;
        MPI_Send(&size, 1, MPI_UNSIGNED_LONG, ...);
        MPI_Send(name.c_str(), size, MPI_CHAR, ...);
    }
};

LCMP(MpiClientProxy)
    L_CPP_PROVIDE(Hello, greeter);
    L_MPI_COMM(comm);
LEND
```



MPI Assembly

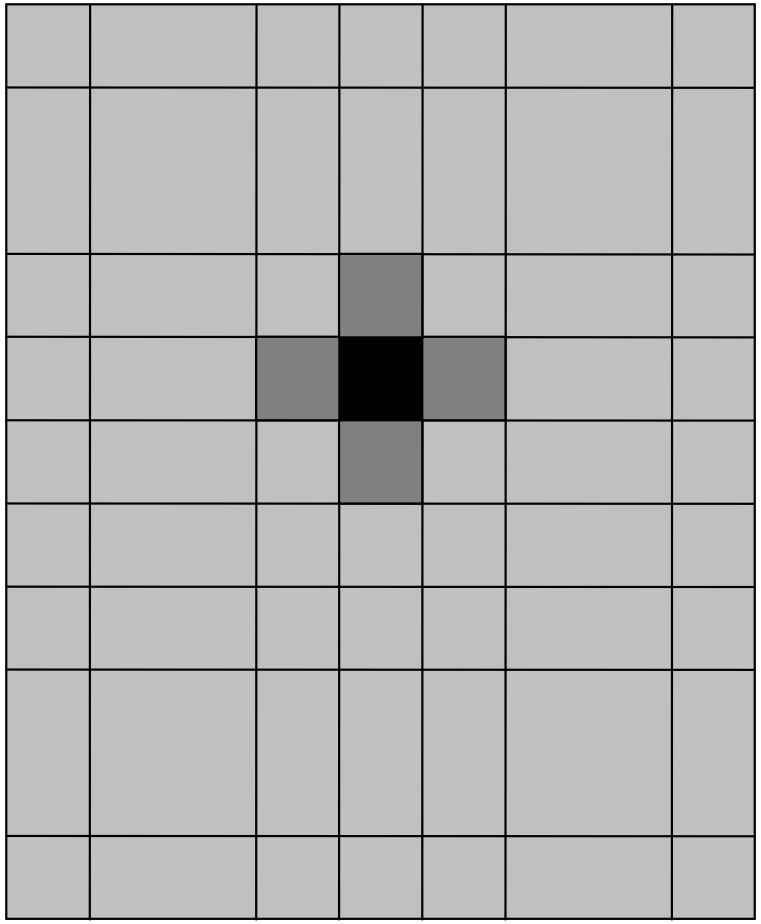



```
<lad><mpi>
        <process>
            <instance id="serv" type="Server"/>
            <instance id="serv_proxy" type="MpiServerProxy">...</instance>
        </process>
        <process>
            <instance id="client_proxy" type="MpiClientProxy"/>
            <instance id="client" type="Client">...</instance>
        </process>
        <communicator>
            <peer instance="client_proxy" property="comm"/>
            <peer instance="serv_proxy" property="comm"/>
        </communicator>
</mpi></lad>
```

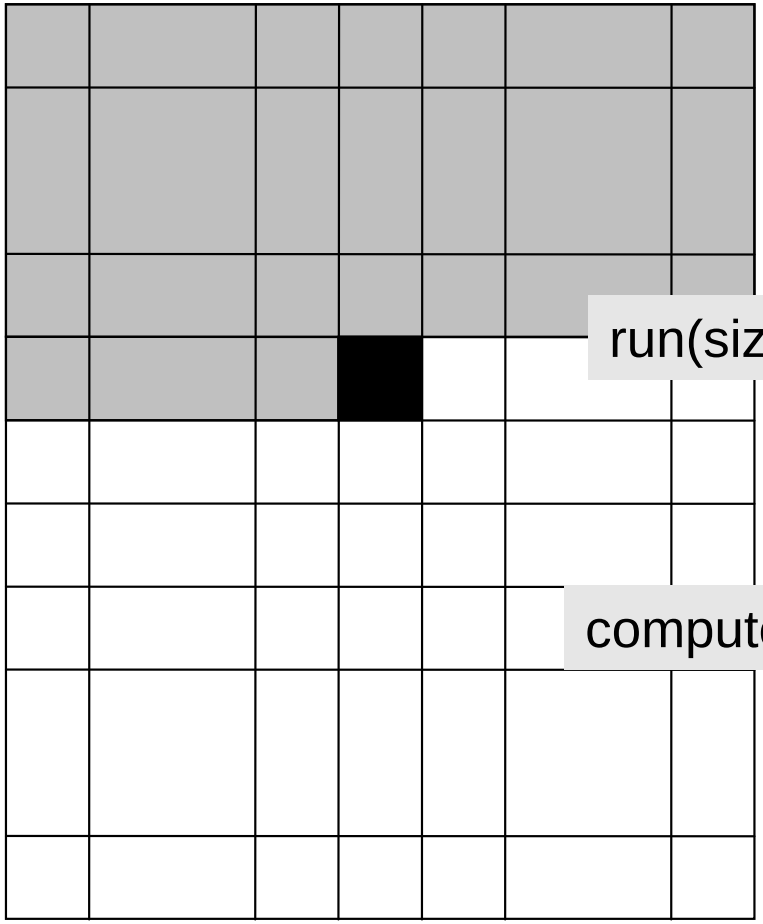
Content

- *Context*
- *Overview of component models*
- *L2C: Low Level Component*
- **Jacobi & L2C**
- Experimental evaluation
- Conclusion & Perspectives

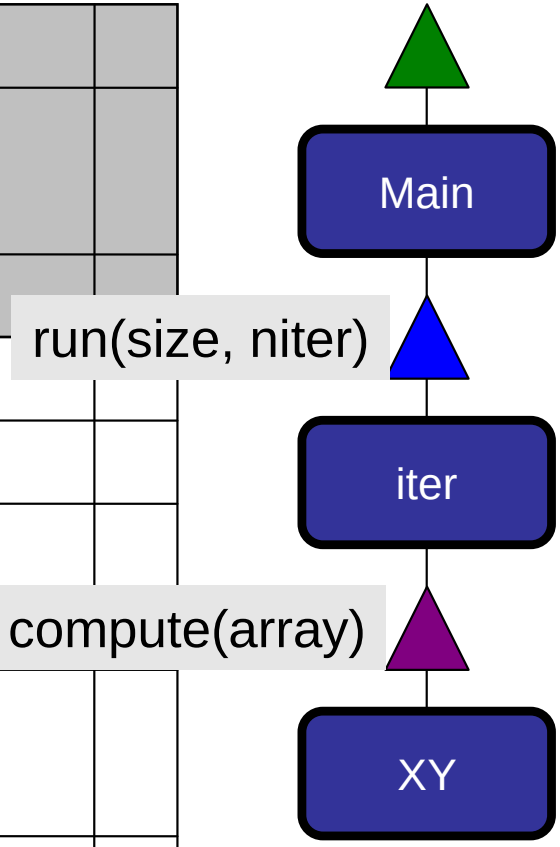
Jacobi computation



Iter = N



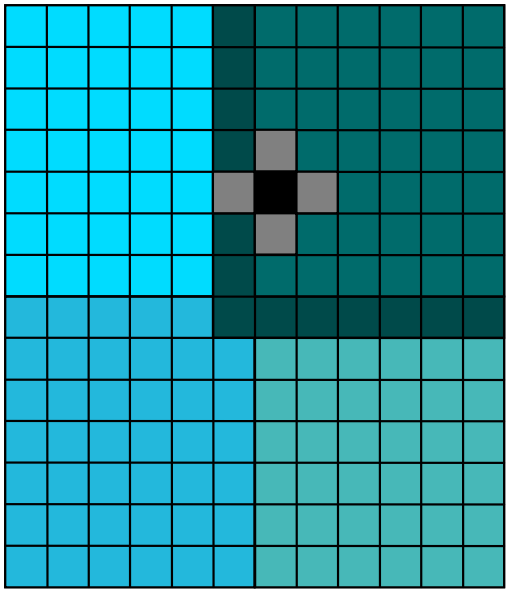
Iter = N+1



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

Jacobi Parallel Computation

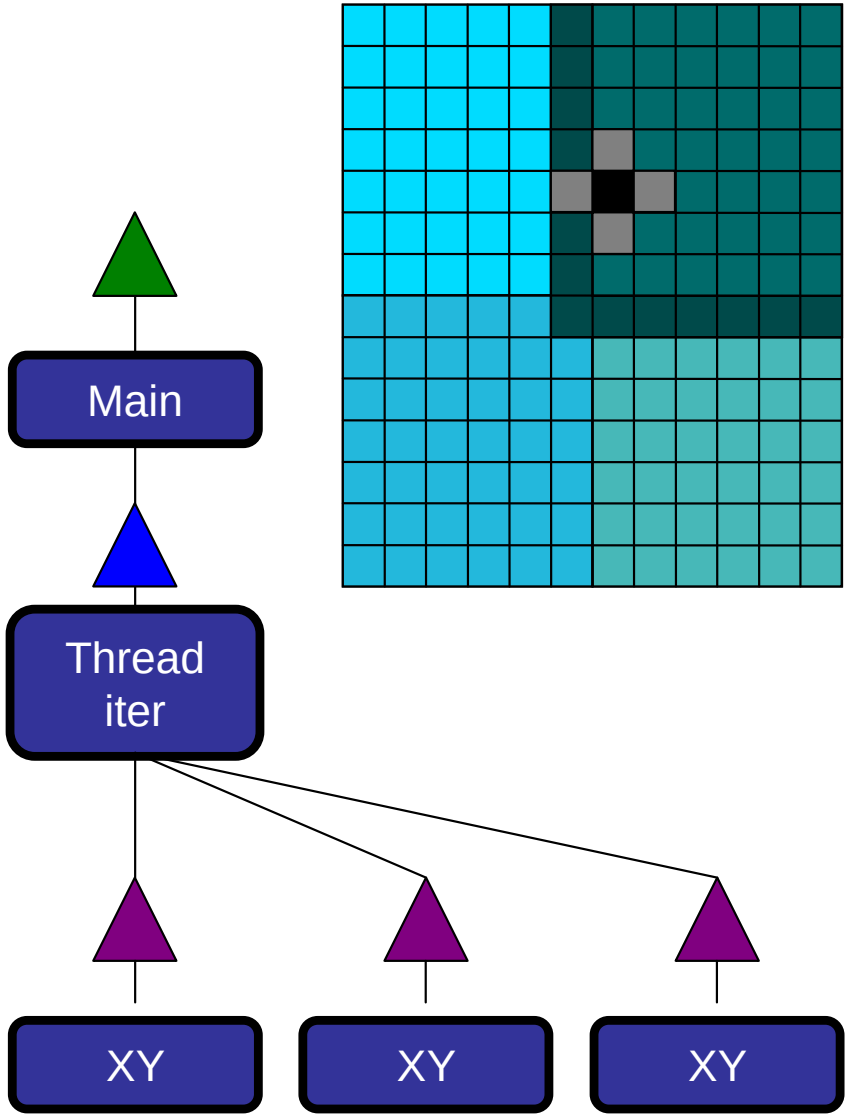
- Domain decomposition
  - 1 thread, 1 subdomain
- Frontier depends on neighbor value
  - Data overlap
- Overlap Consistency
  - Shared memory + synchronization
    - (Not in this talk: distinct memory regions)
  - Distributed memory + message passing



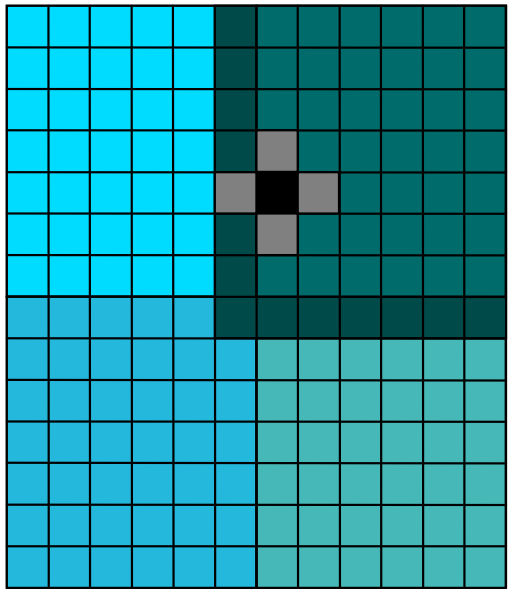
Thread Jacobi Parallelization

- 1 shared array
- Barrier after each iter

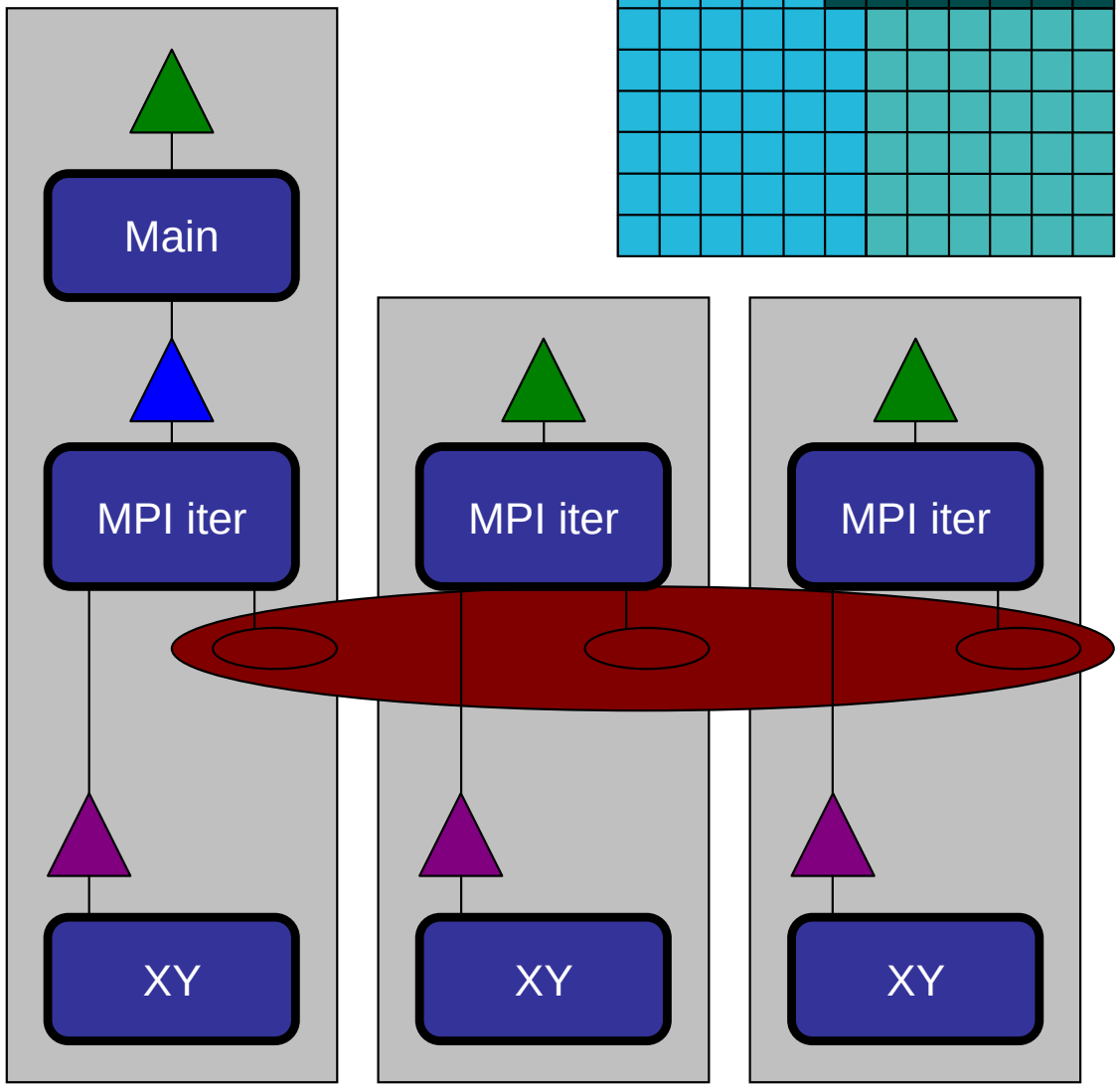
- For iter = 0 to Niter
  - For y = 0 to ymax
    - For x = 0 to xmax
      - tab[iter][x][y] = …
  - Barrier



MPI Jacobi Parallelization



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

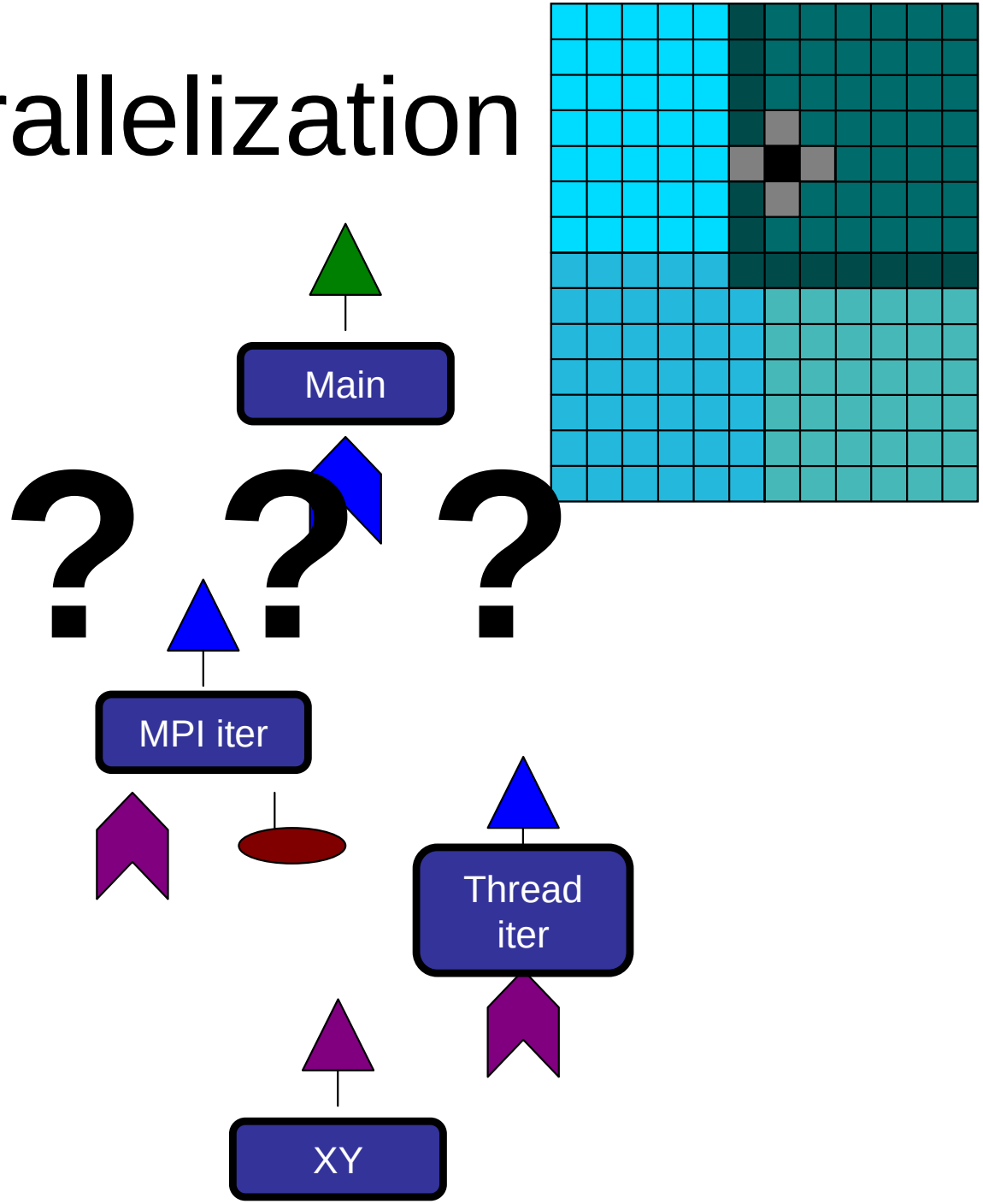


- For iter = 0 to Niter
  - For y = 0 to ymax
    - For x = 0 to xmax
      - tab[iter][x][y] = …
  - SendReceive

Hierarchic Parallelization

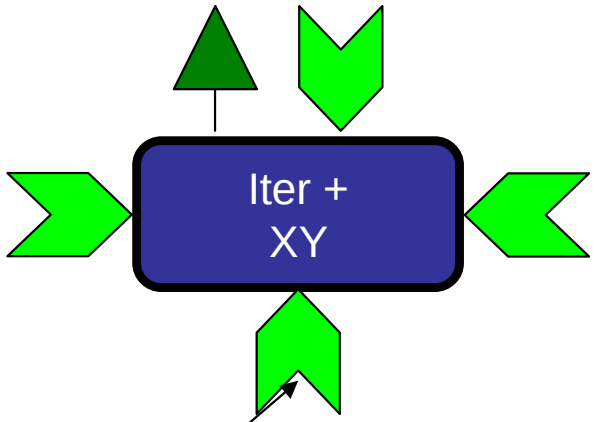
- Multi nodes
  - MPI
- Multi core
  - Threads

- 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

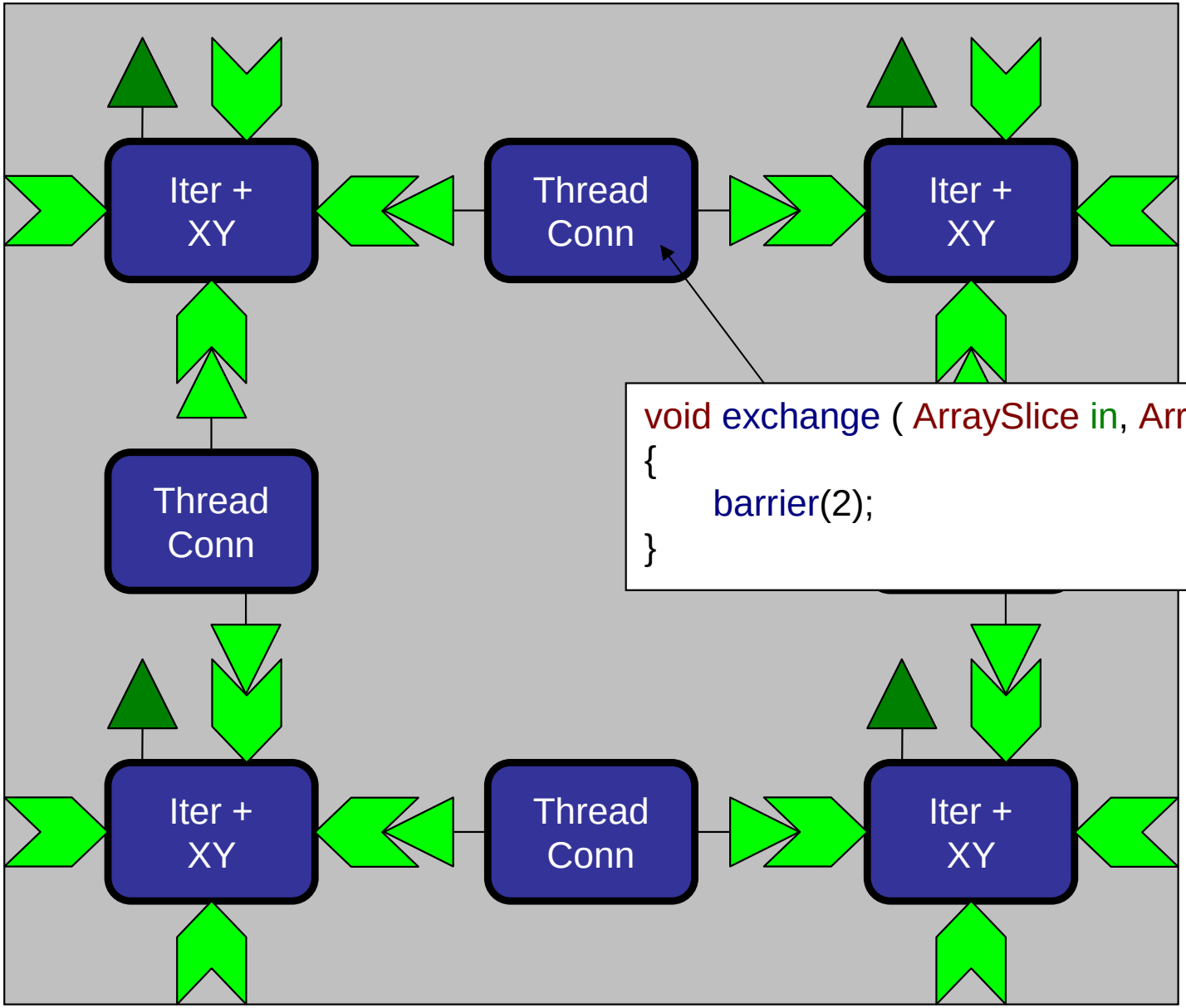
- 1 connector instance
  - 1 domain
- 1 DataExchange/side
  - Implementation agnostic interface



- 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

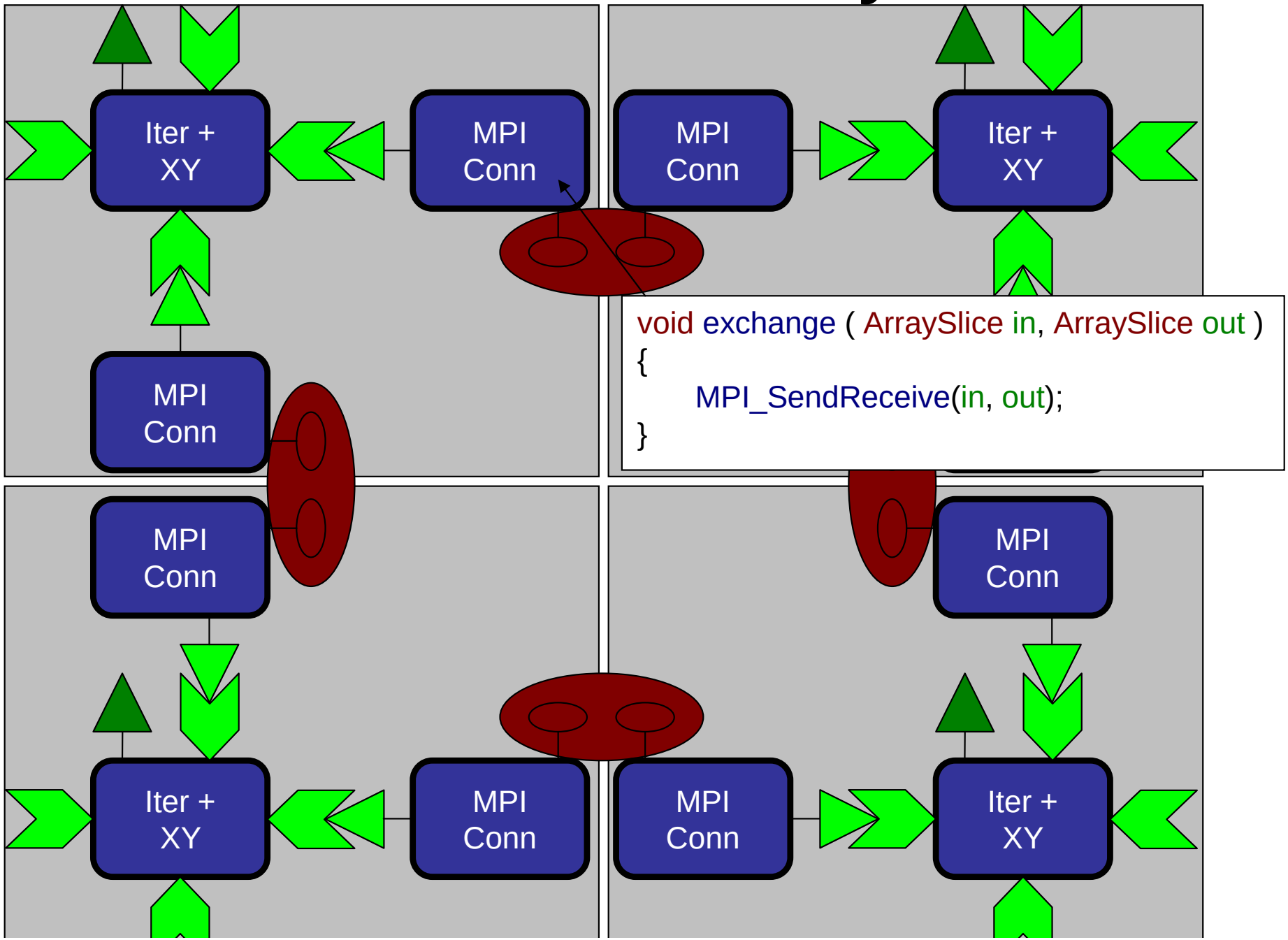
```
class DataUpdate
{
public:
    virtual void exchange (
            ArraySlice in,
            ArraySlice out
    ) = 0;
};
```

The 4 connector way: Threads



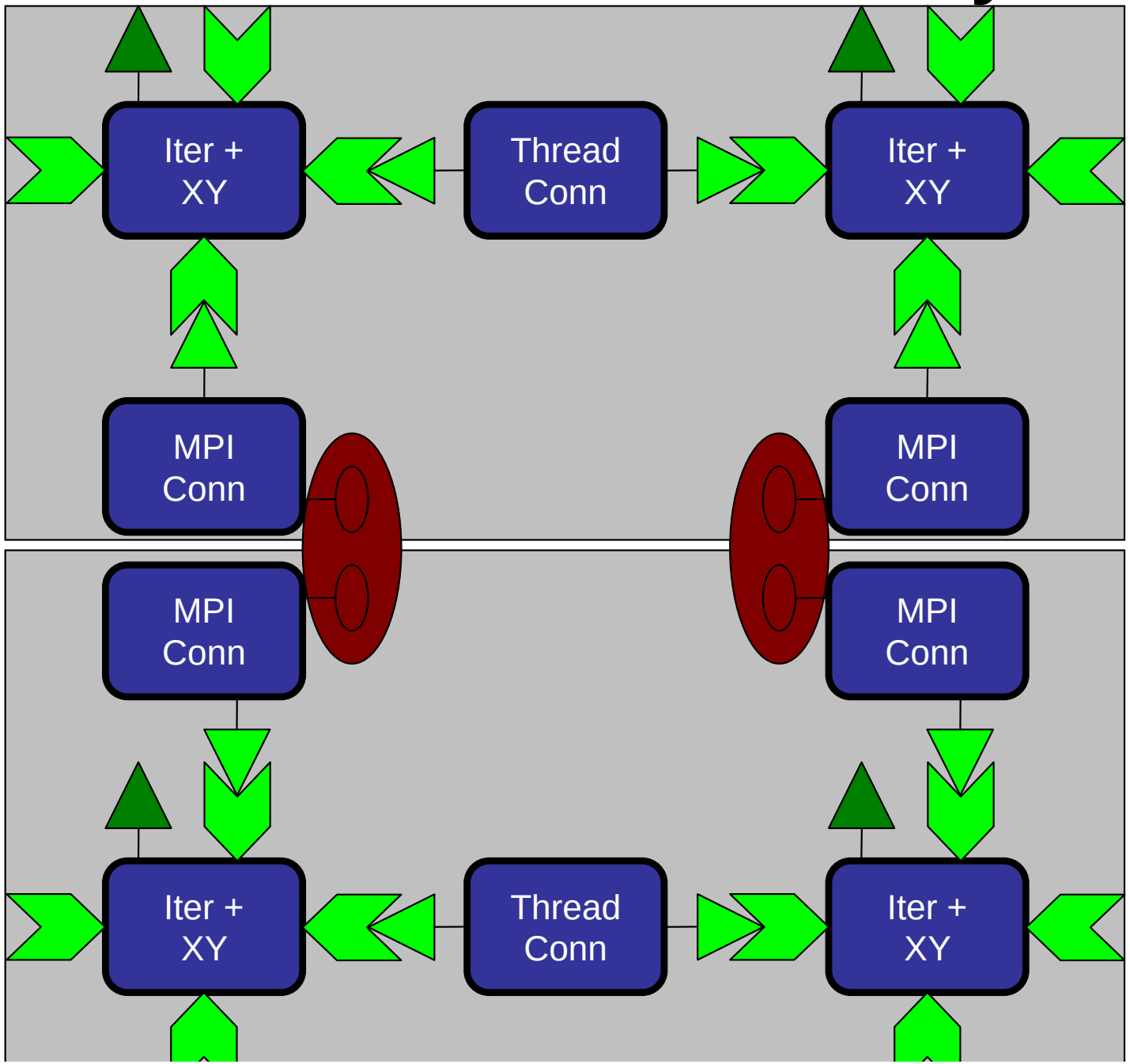

```
void exchange ( ArraySlice in, ArraySlice out )
{
    barrier(2);
}
```

The 4 connector way: MPI




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

The 4 connector way: Hierarchy

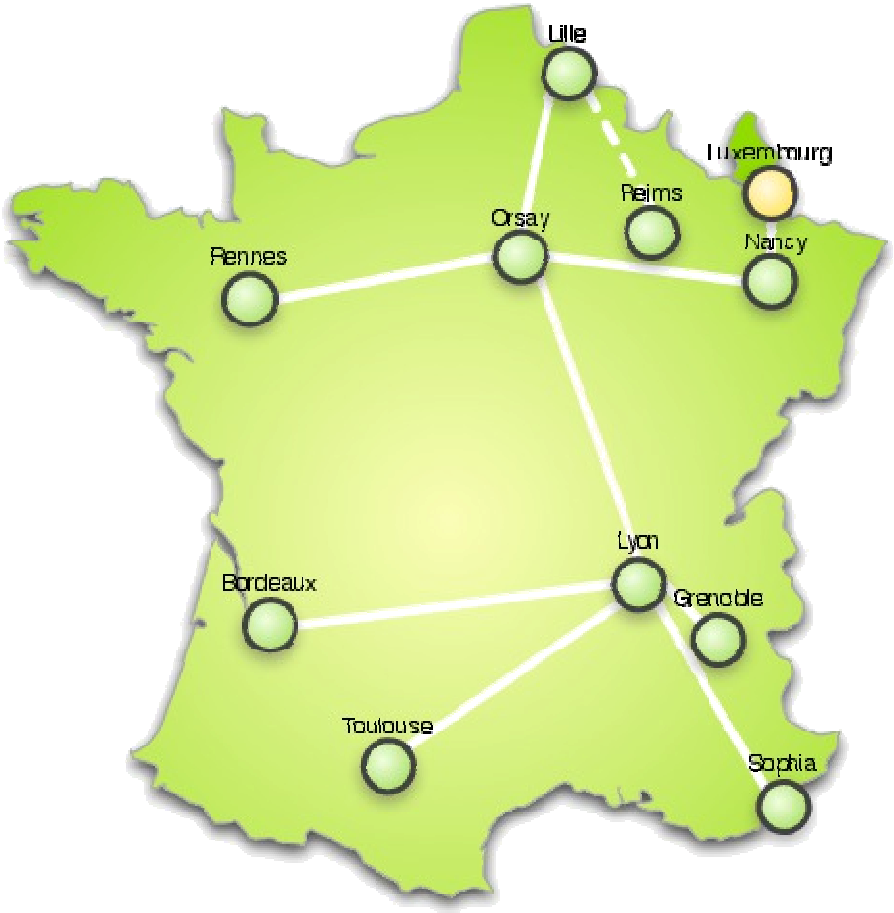


Content

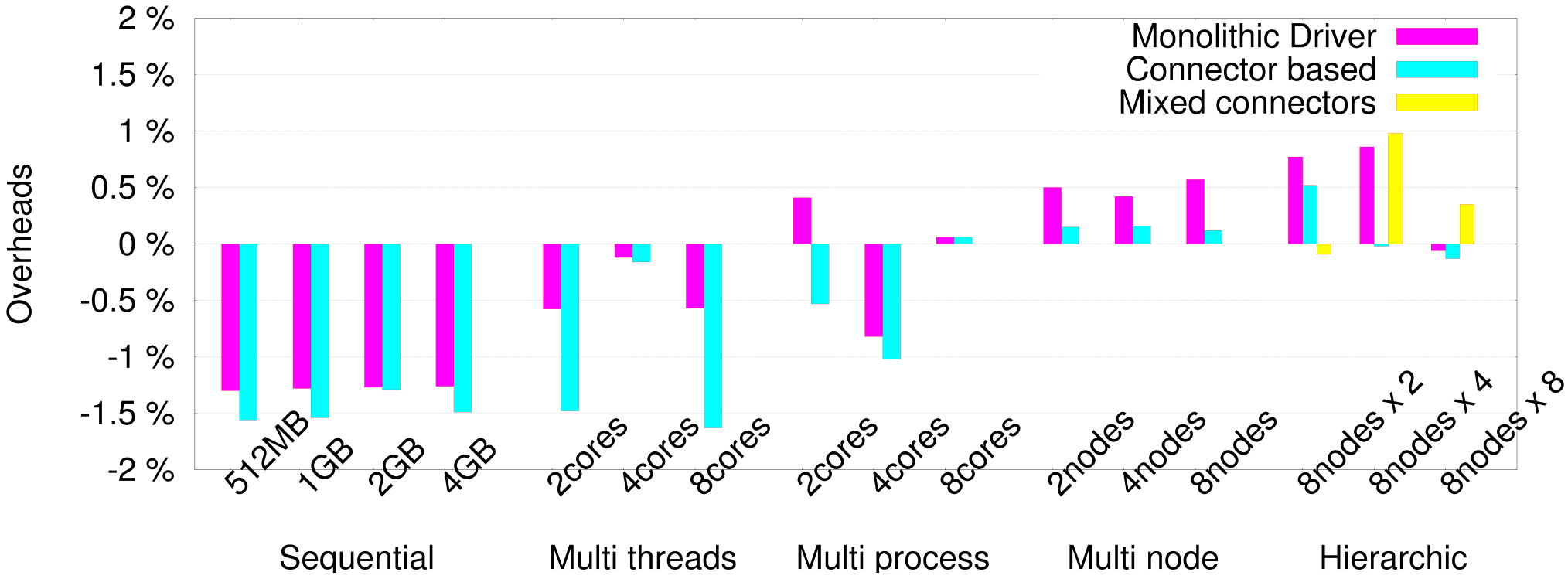
- *Context*
- *Overview of component models*
- *L2C: Low Level Component*
- *Jacobi & L2C*
- **Experimental evaluation**
- Conclusion & Perspectives

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

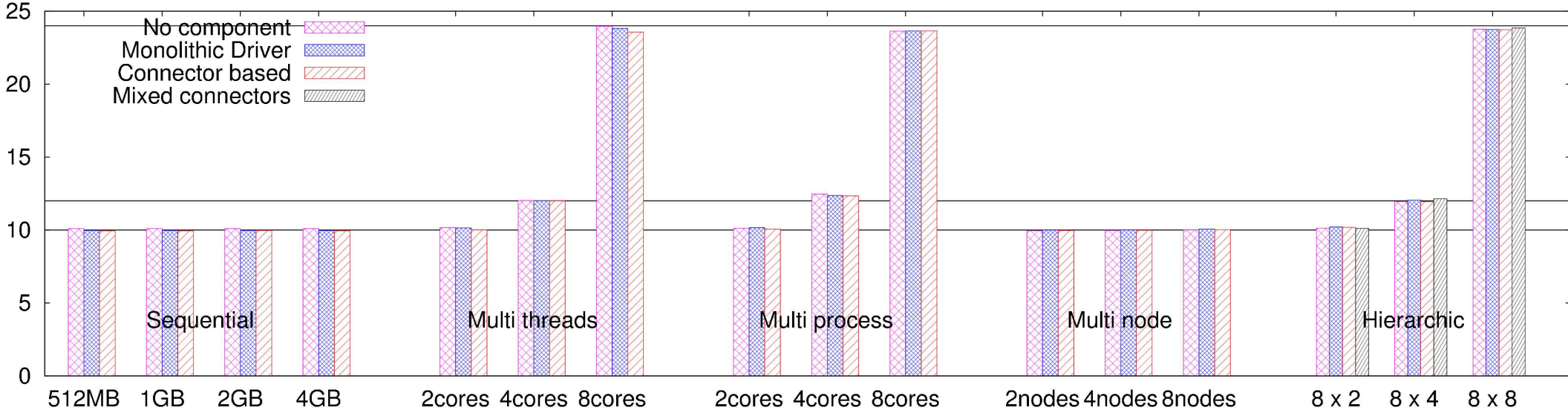


Performance vs Native Codes



- No overhead of using component in Jacobi !

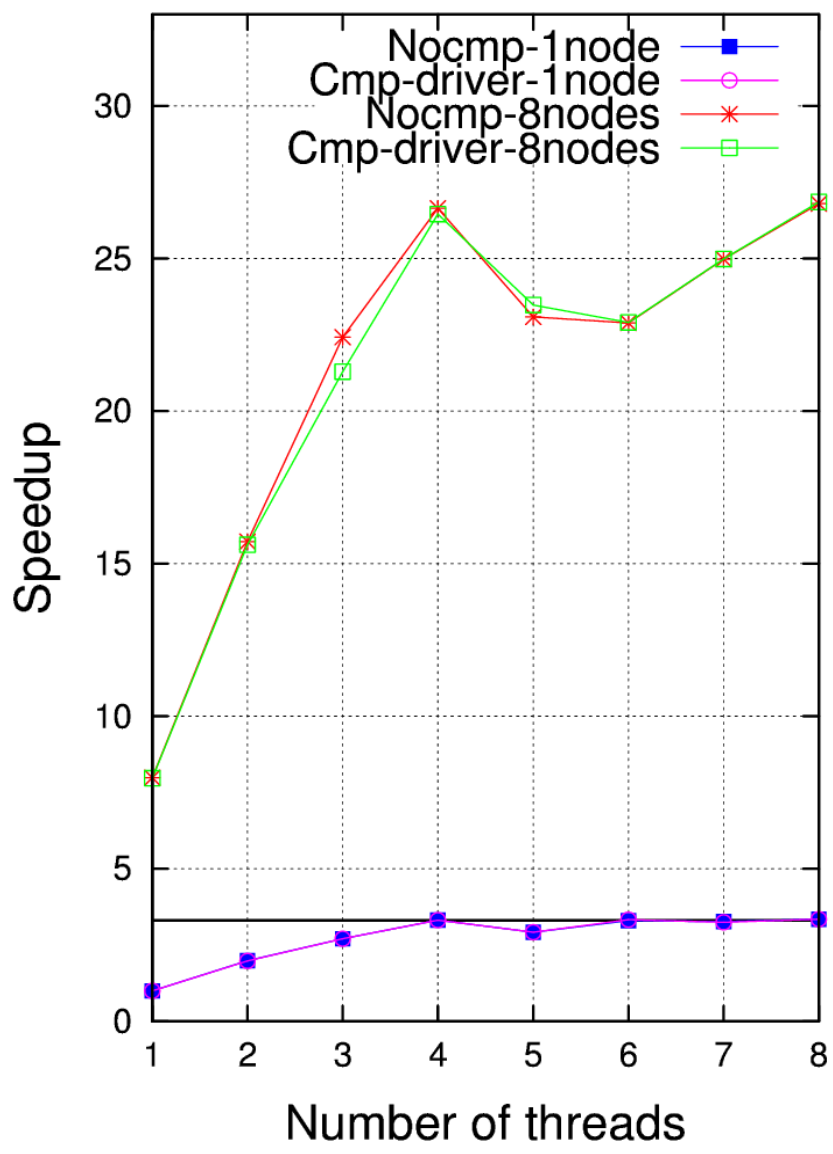
Iteration Time

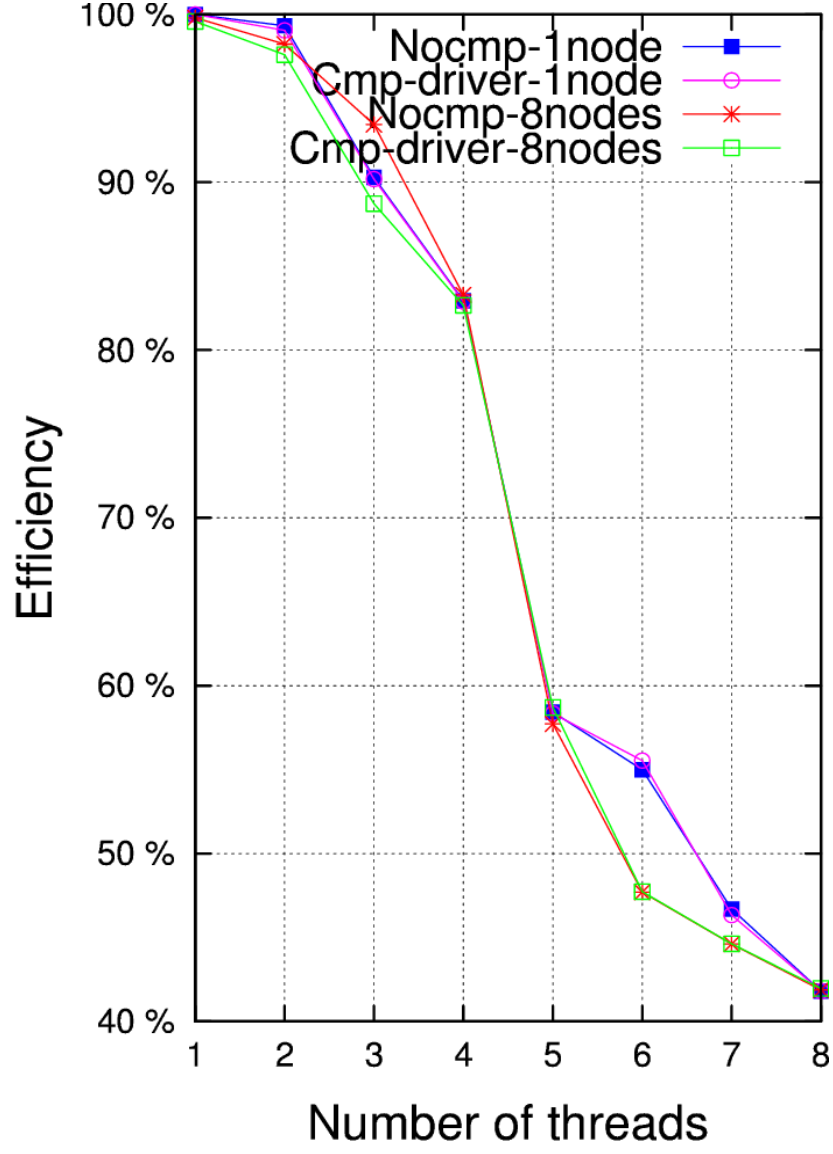


Overhead coming from using too much threads on this machine!

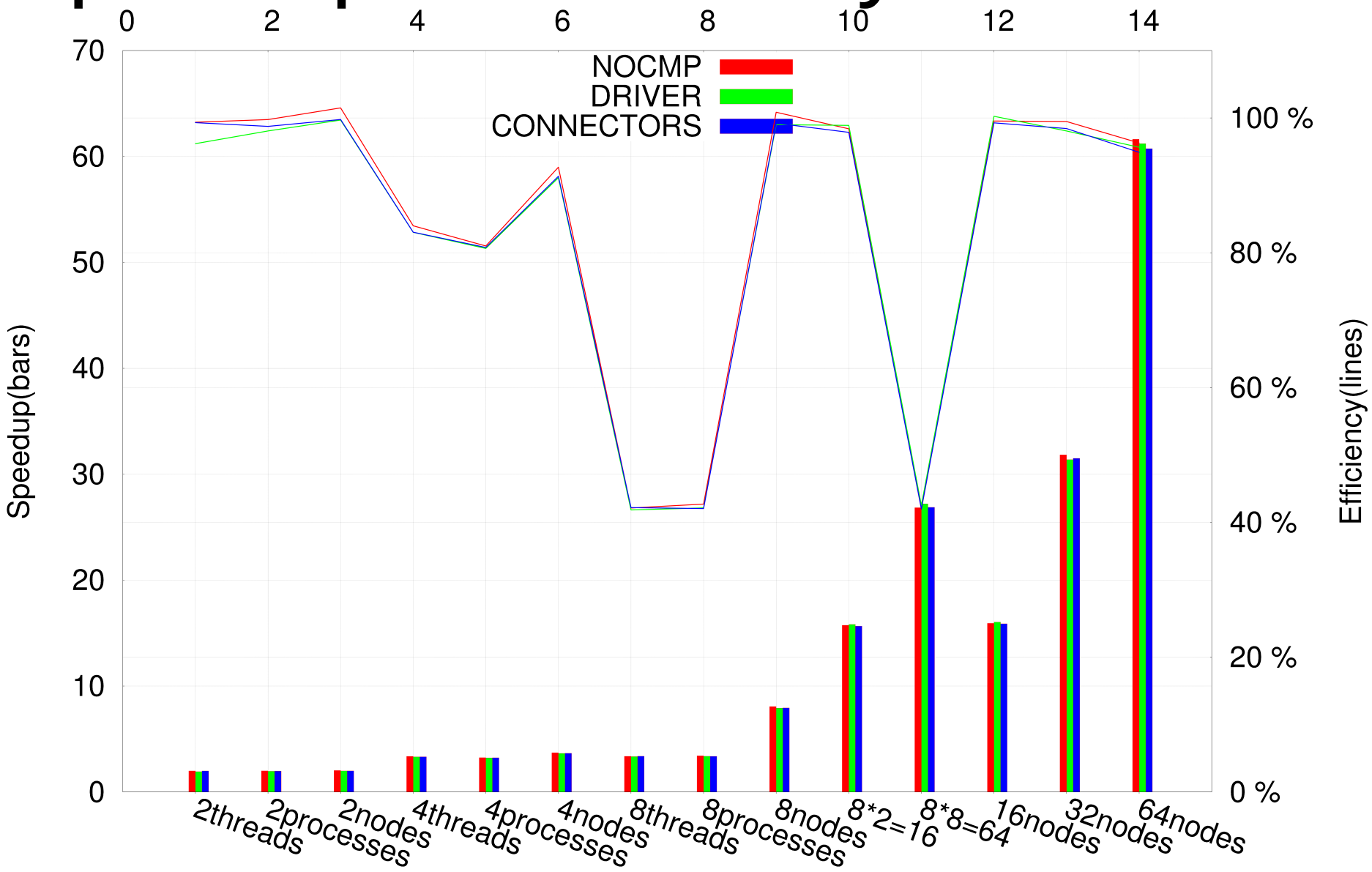
- Limited memory bandwidth

Impact of #Threads per Node





Speedup & Efficiency



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%**

Detailed Component SLOC

Assembly Version	Cpt Name	SLOC	Seq. Reused
Driver & Connector	JacobiCore	**25**	Yes
Driver & Connector	DataInitializer	**68**	Yes
Driver & Connector	Main	**105**	Yes
Driver	SeqDriver	**109**	
Driver	MPIDriver	**155**	
Driver	ThreadDriver	**256**	
Connector	XP	**71**	Yes
Connector	JacobiNiter	**187**	Yes
Connector	ThreadXP	**186**	
Connector	ThreadConnector	**140**	
Connector	MPIConnector	**58s**	

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**

Conclusion & Perspectives

- Component model as a way to master versions
  - Application adaptation => assembly modification
- L2C
  - A simple, efficient, and extensible model
    - C++, MPI, CORBA, (FORTRAN 2008 soon)
- L2C assembly complex to write (XML)
  - Shall be generated by a higher model
  - HLCM: A high level component model
- Extensive memory bandwidth more machines
  - Curie (BULL) & Juquene (BG/Q)
- Design configuration algorithms