Avalon working group
Lyon, April 29th

A Reconfigurable Component Model for HPC

Vincent Lanore[1], Christian Pérez[2]

[1] : ENS de Lyon, [2] : Inria
LIP laboratory, Avalon Team
France

High-Performance Computing

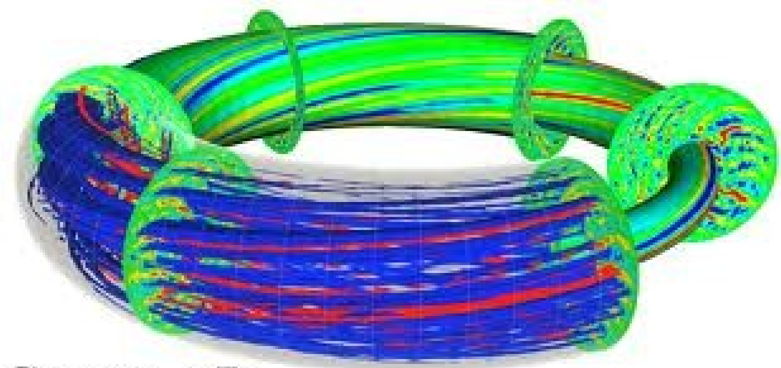
Goal: **run the biggest possible applications**

- *eg,* large simulations
- months/years of sequential computing time

using cutting edge hardware

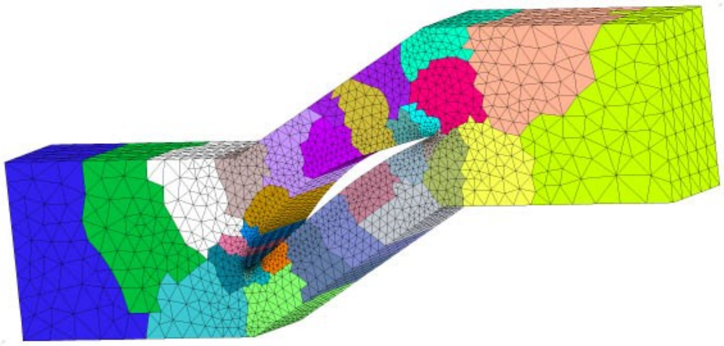
- very parallel

Challenge: scalability



GYSELA 5D
years of sequential computing time
source: master omp, Paris13

Tianhe-2
3,120,000 cores
source: top500.org



A scientific mesh-based simulation
source: NTUA,
school of mechanical engineering

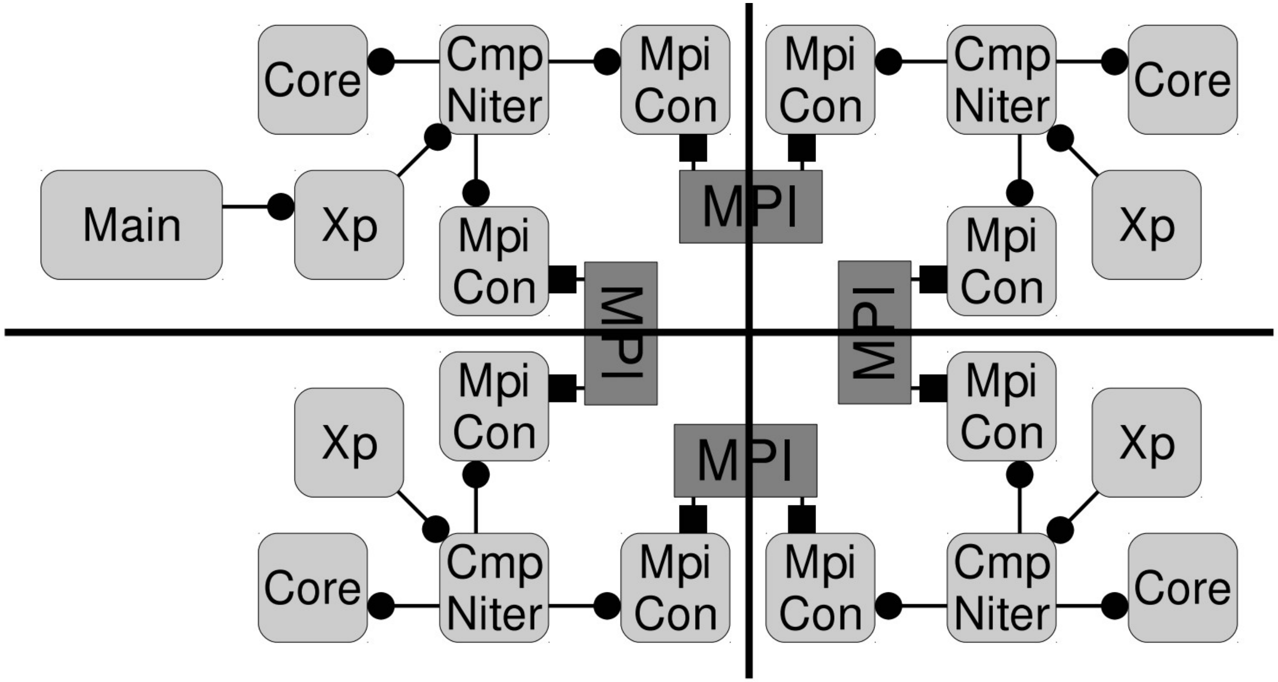
HPC Component Models

Examples: CCA, L2C

CCA Common Component Architecture

Typically:

- low-level
  - C++/FORTRAN-level abstractions
  - non-hierarchical
- distributed
  - *eg,* message passing, remote method call
  - process abstraction



A jacobi solver assembly on 4 processes

MPI = Message Passing Interface

Problem: Dynamic HPC Applications

Applications with...

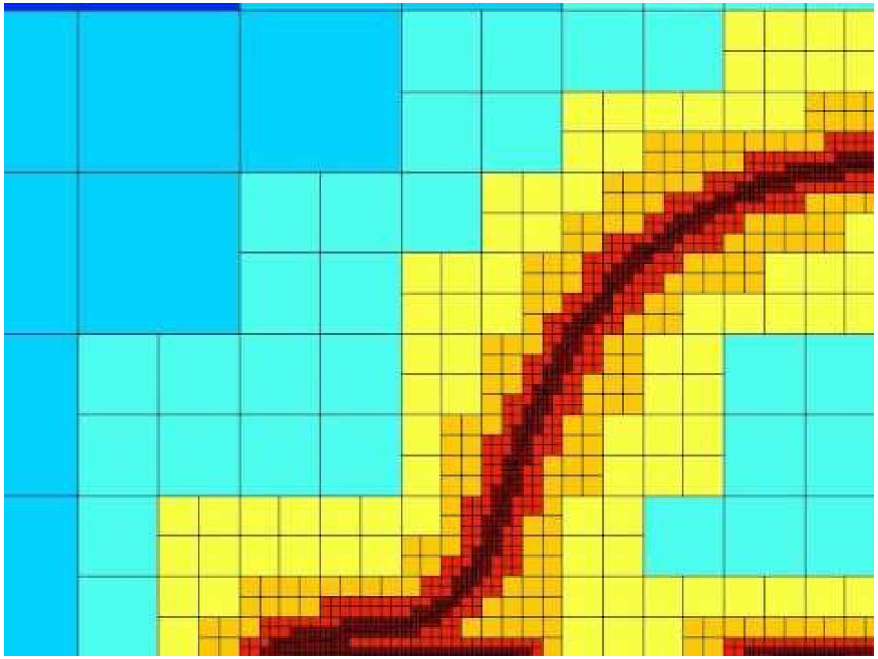
- dynamic communication topology
- dynamic data structure

Not supported by HPC component models

- reconfiguration needed

Examples

- Adaptive Mesh Refinement (AMR)
- dynamic load balancing



AMR mesh,
varying resolution

Goal of this talk:
HPC reconfigurable component model

Plan of the Talk

Context and related works

- Related work
- Our proposition

Presentation of the model, DirectMOD

- Assembly model
- Programming model

Implementation and evaluation

- Ring example
- DirectL2C
- Code and performance

Conclusions and perspectives

Reconfigurable Component Models

From the literature:

	Examples	Locking and representation	Scalable?	Reconf SE properties
No reconfiguration support	CCA, L2C	none	**up to the user**	**poor**
Global reconfiguration	global MAPE	global	**no**	**good**
Composite-level controllers	Fractal, SOFA	composite-level	**sometimes**	**sometimes**

Important parameters

- locking granularity
  - scalability
- representation granularity
  - ease of use

No model provides both

- scalable approach
- good SE properties for reconfigurable assemblies
  - reuse
  - separation of concerns

Our proposal: main ideas

Let users define locking units

- custom granularity / distribution → performance

	Locking and representation	Scalable?	Reconf SE properties
Domain-based	user-defined (domains)	yes	???

To improve SE properties

- separate locking from representation and transformation

	Locking and representation	Scalable?	Reconf SE properties
Domain-based + separation lock/transfo	user-defined + separated	yes	good

A Formal Model

DirectMOD : formal model

- full syntax
- transformation semantics

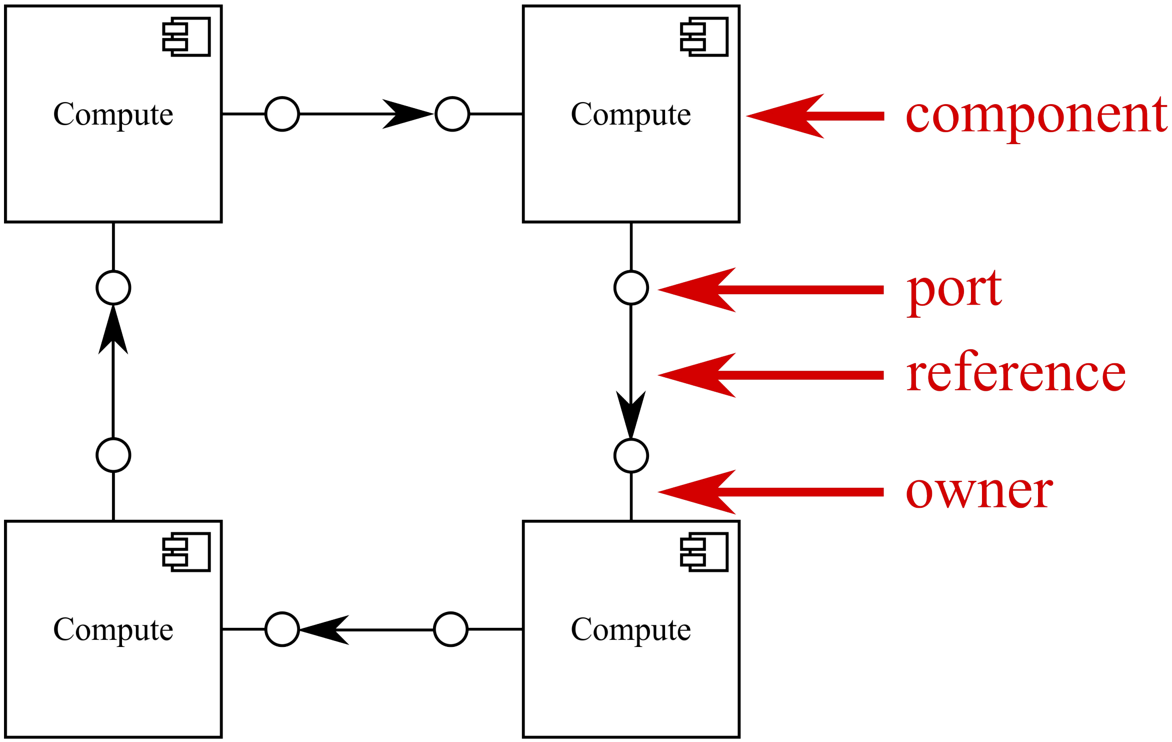
Benefits

- unambiguous specification
- tech-agnostic
- runtime representation

In addition :

- resource model (see paper)
- call stack operational semantics (see paper)

DirectMOD Assembly Model



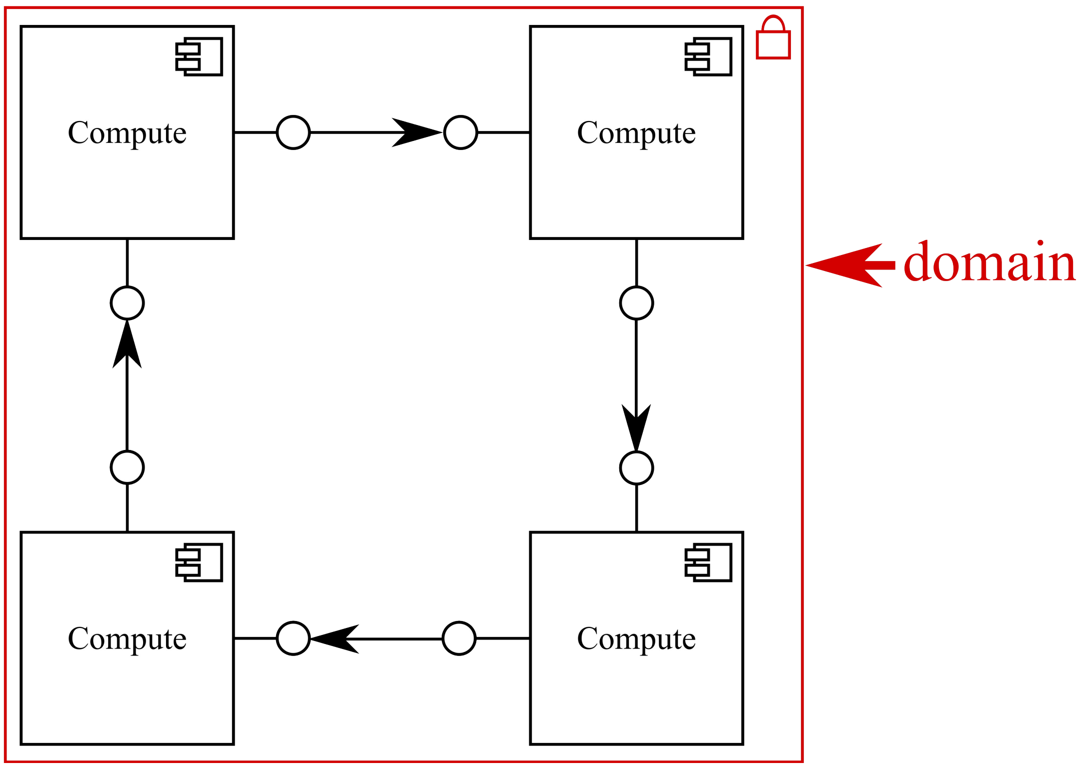
Elements

- components
- ports

Relations

- point-to-point references
- owner (component-port relation)

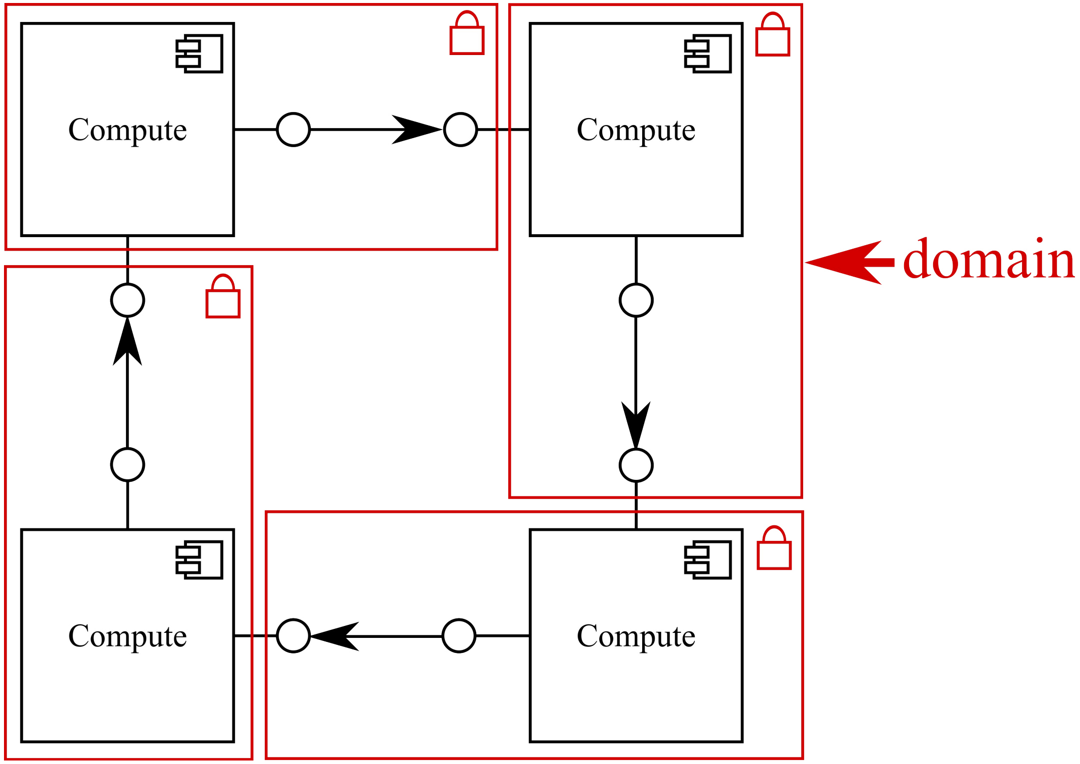
DirectMOD Domains



New element: domains

- manage a subassembly
- unit of **locking**
- unit of internal **representation**
- reconfigure their contents

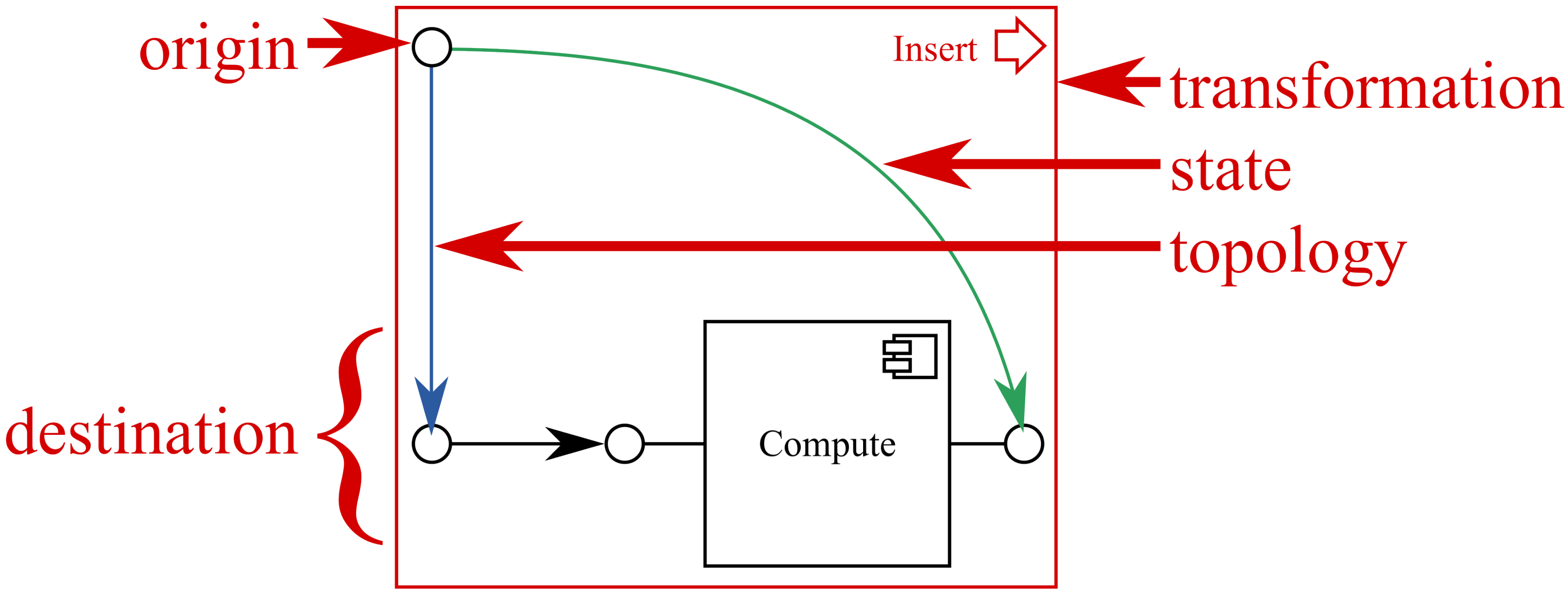
DirectMOD Domains

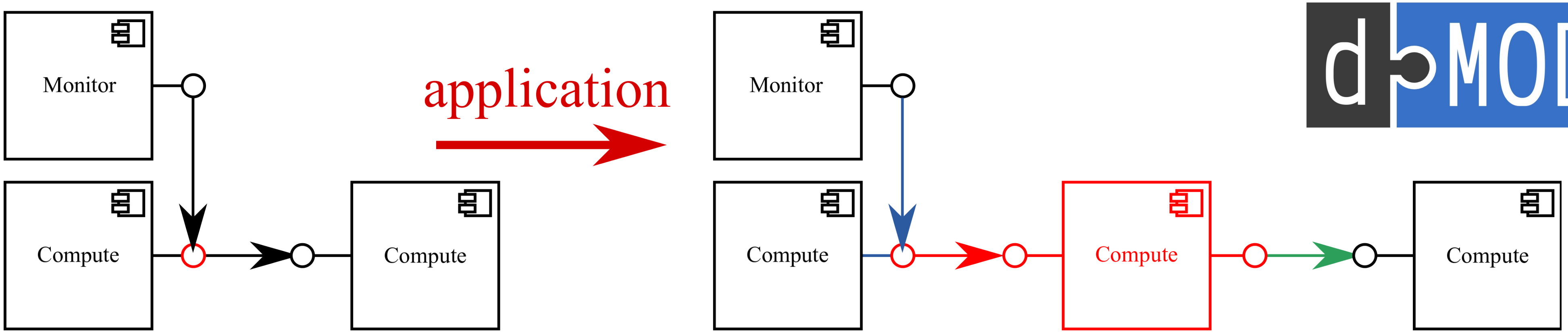


New element: domains

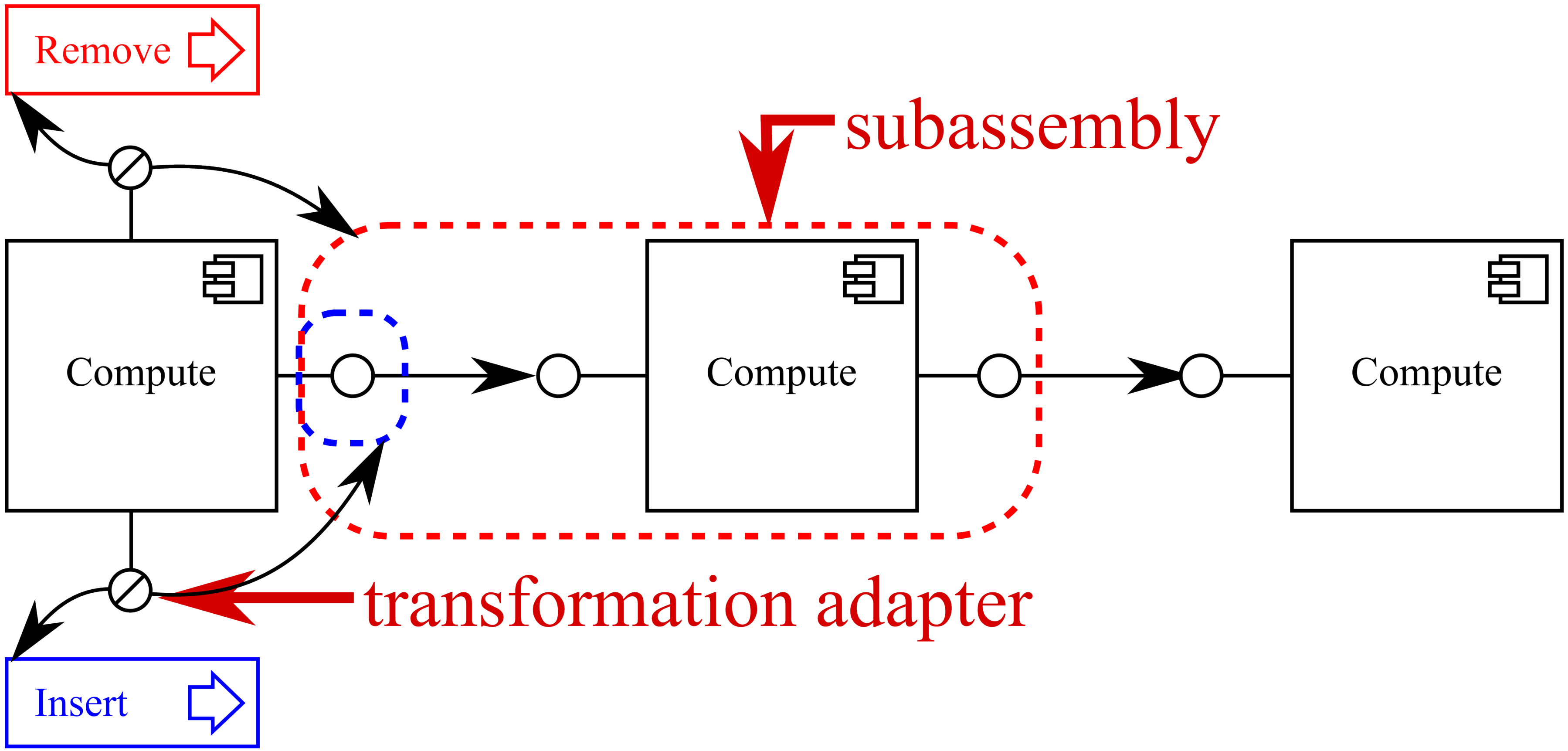
- manage a subassembly
- unit of **locking**
- unit of internal **representation**
- reconfigure their contents

DirectMOD Transformations



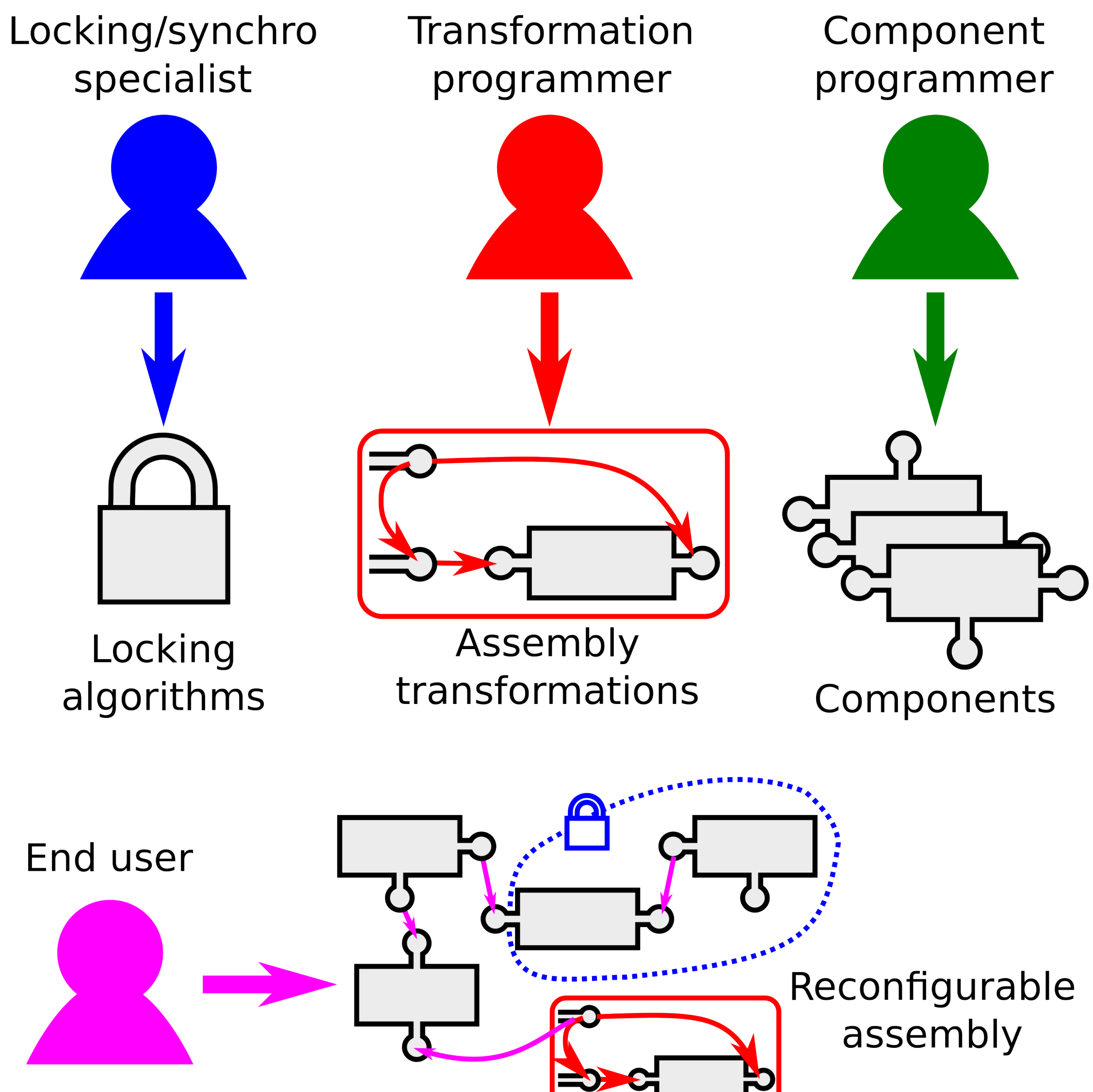


DirectMOD Transformation Adapters



- special kind of port
- reference to transformation and application subassembly

DirectMOD Programming Model



A C++/MPI Implementation

DirectL2C

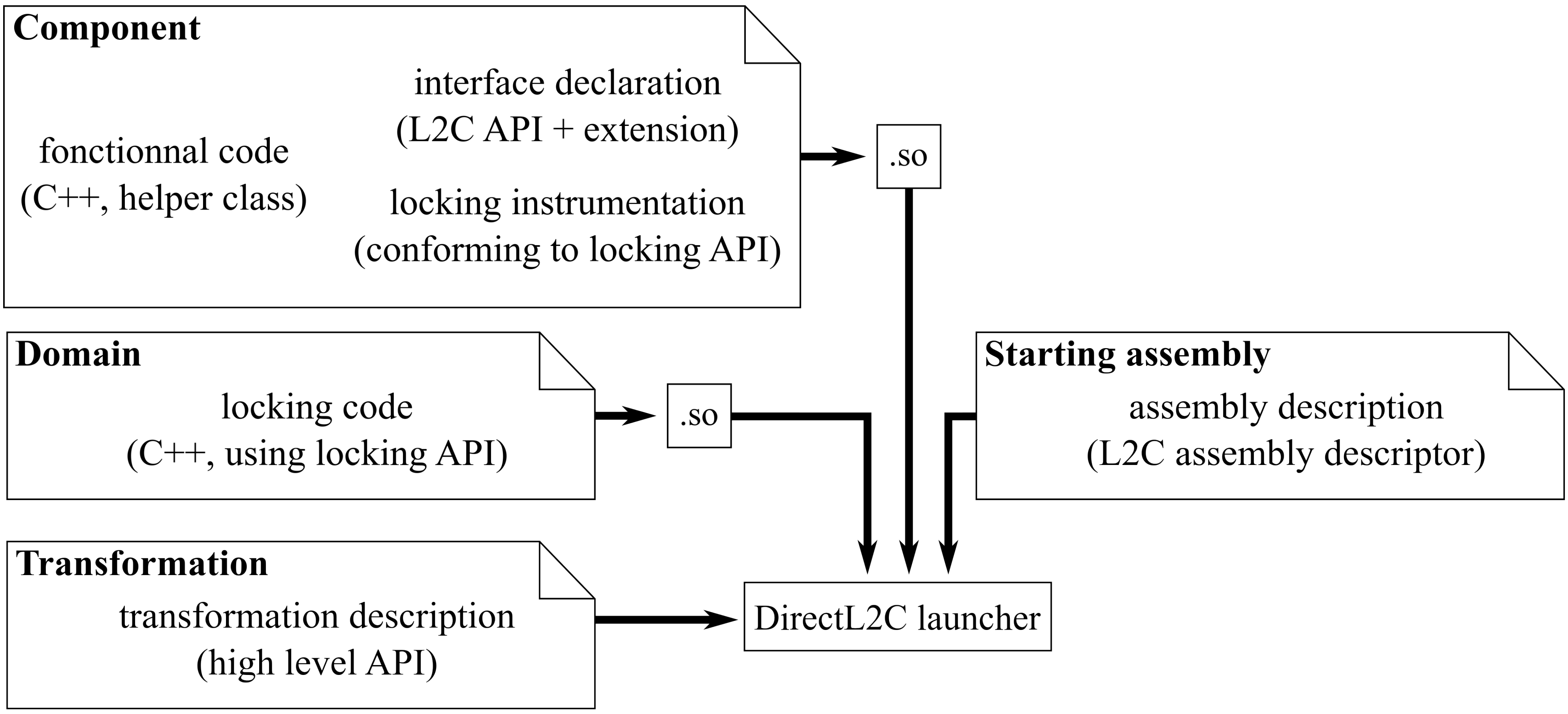
- DirectMOD implementation
- extension of L2C
- uses traditional HPC tech
  - C++
  - MPI (Message Passing Interface)
  - threads

Provides

- remote basic reconfiguration operations
- transformation parsing and execution
- helper classes for locking
- interface and locking APIs

	Components	Files	mLOC
L2C	37	79	4570
DirectL2C	3	10	1118

DirectL2C in Practice



Evaluation: Easy to Write?

Implemented: ring assembly

- insert new component
- remove component

Preliminary implementation

- not yet fully optimized
- shortest possible code

Ring assembly LOC

Function	C++ LOC
Transformation	8
Non-functional sync	20
Code instrumentation	13
L2C overhead	7
DirectL2C overhead	6
Functional code	31
Other	3
TOTAL	**88**

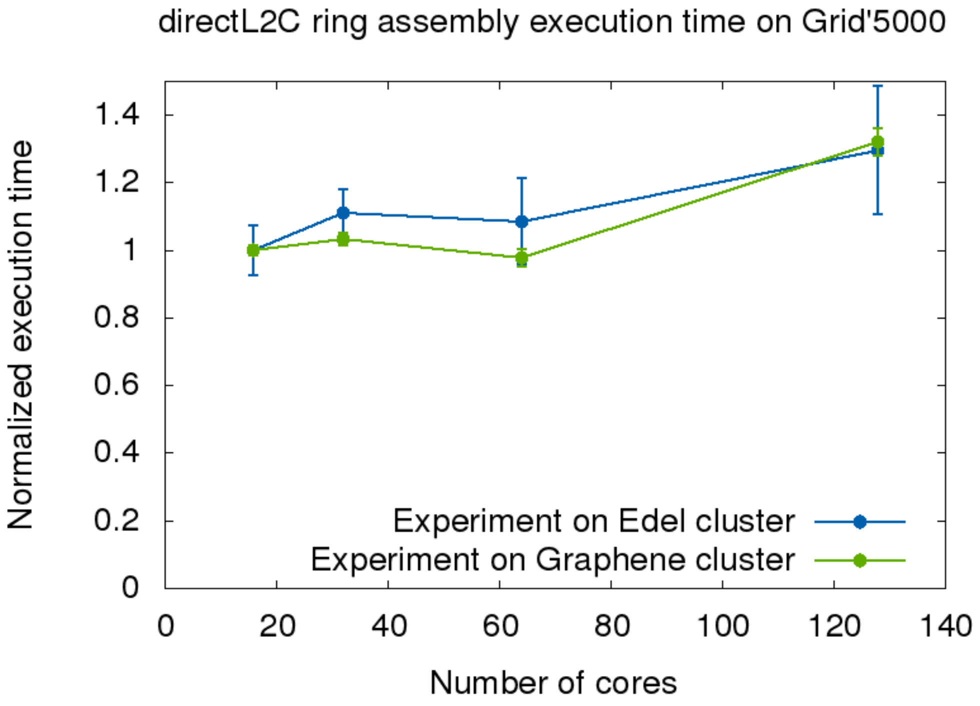
- short and easy transformation code
- large and difficult synchronization code

Evaluation: scalable?

Testing our ring assembly

- one component per core at startup
- fixed number of *insert* and *remove* transformations per starting component
- one domain per component (fully distributed)

Preliminary experiments on Grid'5000



- acceptable scalability up to 128 cores

Conclusion and Perspectives

Presented DirectMOD

- a reconfigurable component model
- formal
- +implementation (DirectL2C)
- preliminary evaluation
  - short and easy to write code
  - scalability up to 128 cores

Perspectives

- improve evaluation
  - ongoing work on a complex benchmark
  - experiments on large platform (*eg*, Curie)
- ease domain development
  - ongoing work on automating certain locking patterns
- transformation specification
  - genericity
  - compact language