STAARS Seminar
Grenoble, June 18th

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

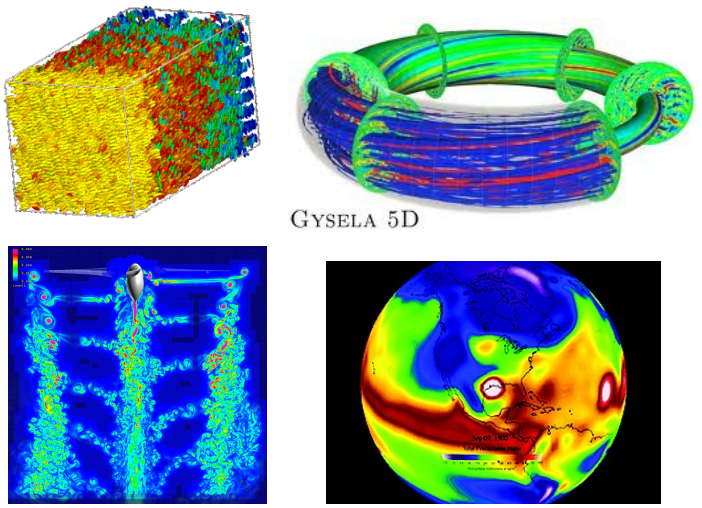
Goal: **run the biggest possible applications**

- from astrophysics, meteorology, industry...
- up to millions of years of sequential computing time

using cutting edge hardware

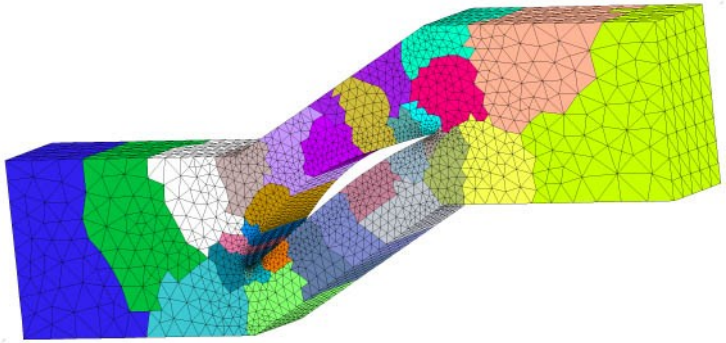
- very parallel

Challenge: scalability



Various application domains

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



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

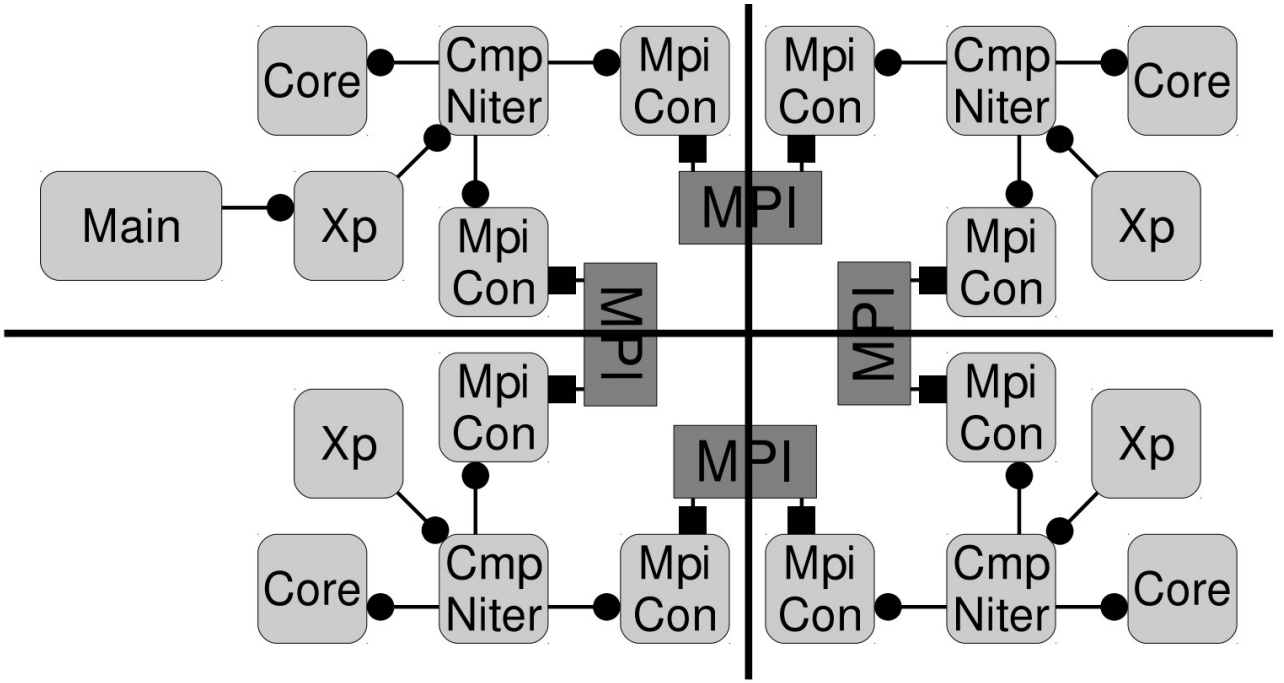
HPC Component Models

Examples:

- CCA
- L2C (Avalon team)

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...

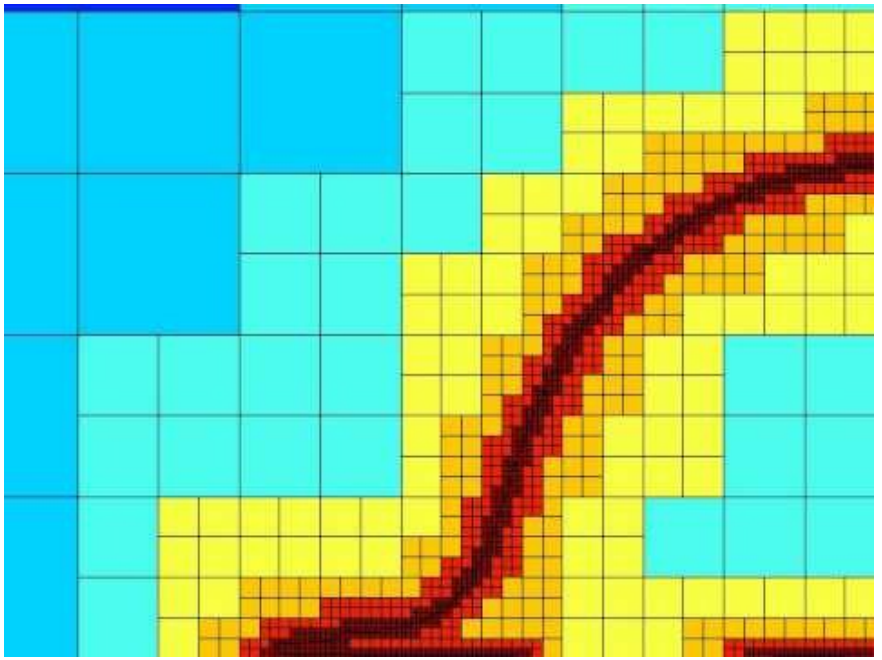
- dynamic communication topology
- dynamic data structure

Examples

- Adaptive Mesh Refinement (AMR)
- dynamic load balancing

Not supported by HPC component models

- reconfiguration needed



AMR mesh,
varying resolution

Goal of this talk:

HPC reconfigurable component model

reconfigurable
scalable (distributed, concurrent reconfiguration)
good SE properties (reuse, separation of concerns)

Plan of the Talk

Context and related works

- Related work
- Our proposition

Presentation of the model, DirectMOD

- Model elements
- Putting it all together

Implementation: DirectL2C

- Code
- Performance

Extensions of DirectMOD

- Efficient locking
- High-level language(s) for transformations

Conclusions and perspectives

Plan of the Talk

Context and related works

- Related work
- Our proposition

Presentation of the model, DirectMOD

- Model elements
- Putting it all together

Implementation: DirectL2C

- Code
- Performance

Extensions of DirectMOD

- Efficient locking
- High-level language(s) for transformations

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 (*domains*)

- custom granularity / distribution → performance

Separate locking units from transformations

- transformation on standalone representation
- assembly-level explicit mapping to domain contents (using *transformation adapters*)

	Locking units	Representation for transformations	Scalable?	Reconf SE properties
Domain-based	user-defined		**yes**	
Transformation adapters		standalone		**good**

Plan of the Talk

Context and related works

- Related work
- Our proposition

Presentation of the model, DirectMOD

- Model elements
- Putting it all together

Implementation: DirectL2C

- Code
- Performance

Extensions of DirectMOD

- Efficient locking
- High-level language(s) for transformations

Conclusions and perspectives

A Formal Model

DirectMOD : formal model

- full syntax
- transformation semantics

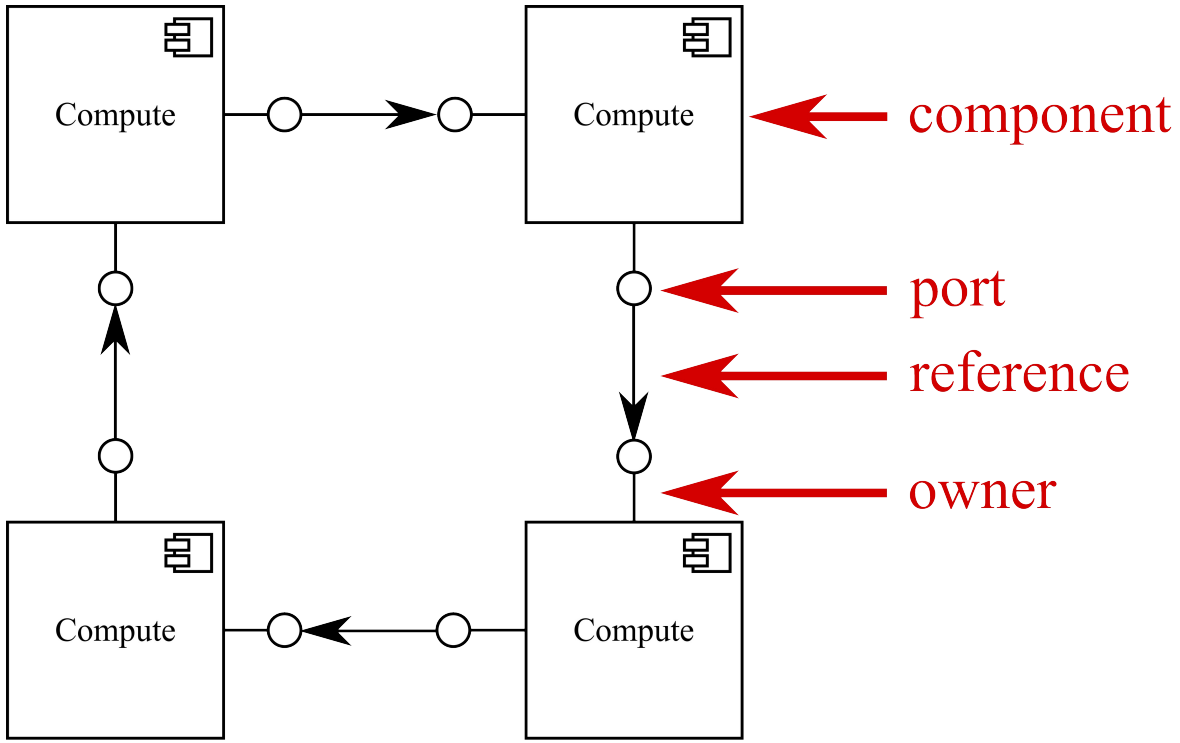
Benefits

- unambiguous specification
- tech-agnostic
- runtime representation

In addition :

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

DirectMOD Assembly Model



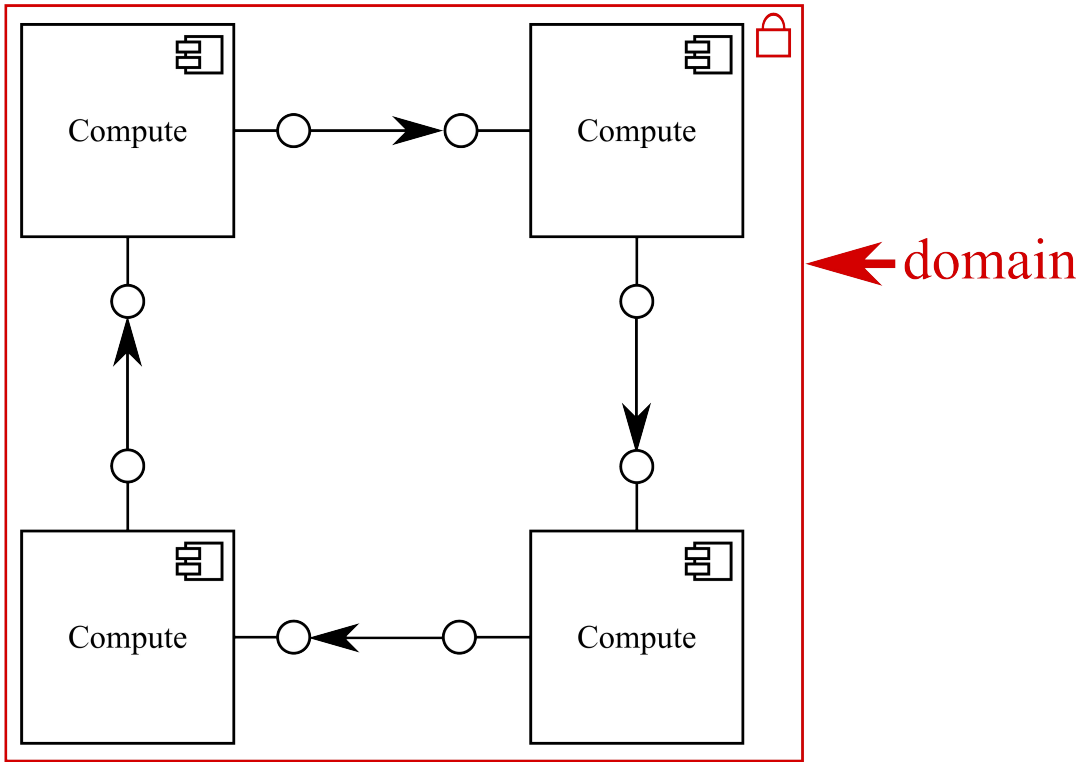
Elements

- components
- ports

Relations

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

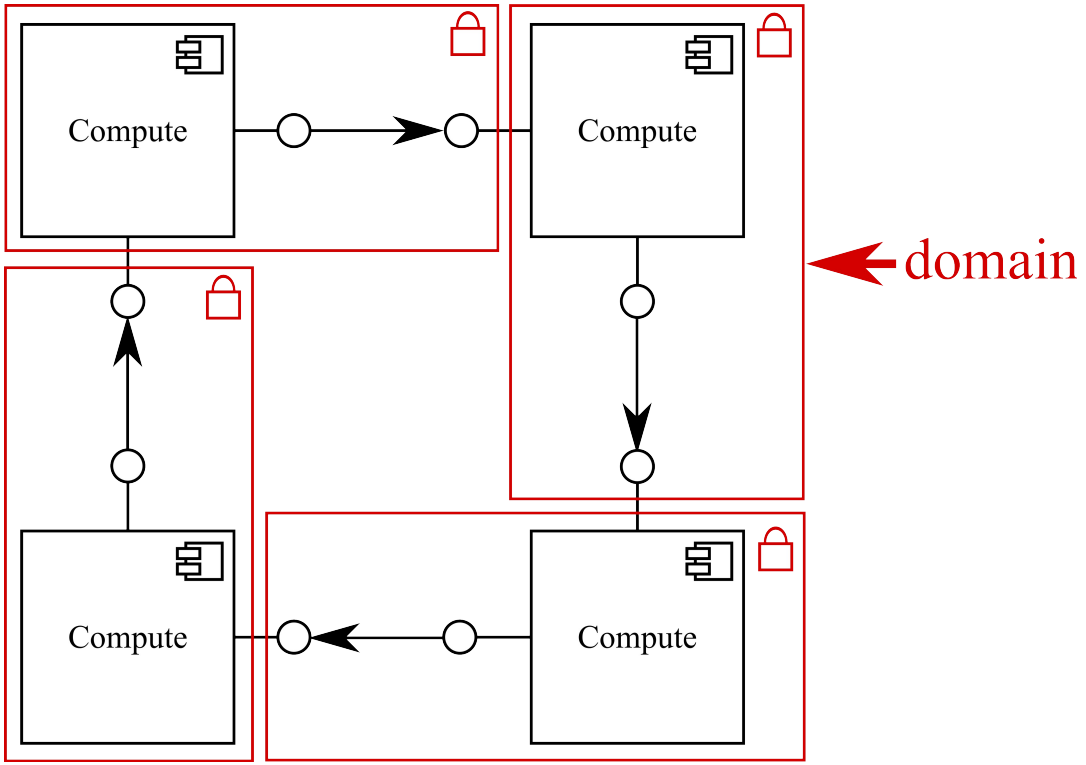
DirectMOD Domains



New element: domains

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

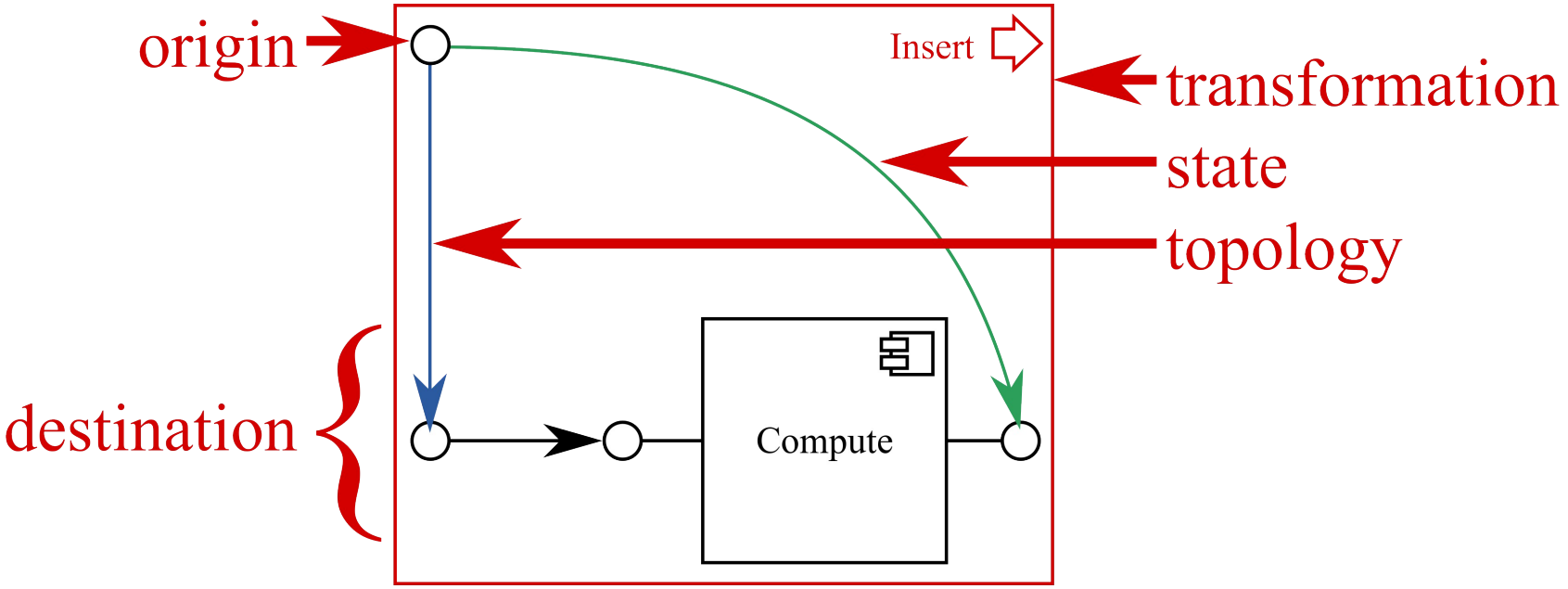
DirectMOD Domains



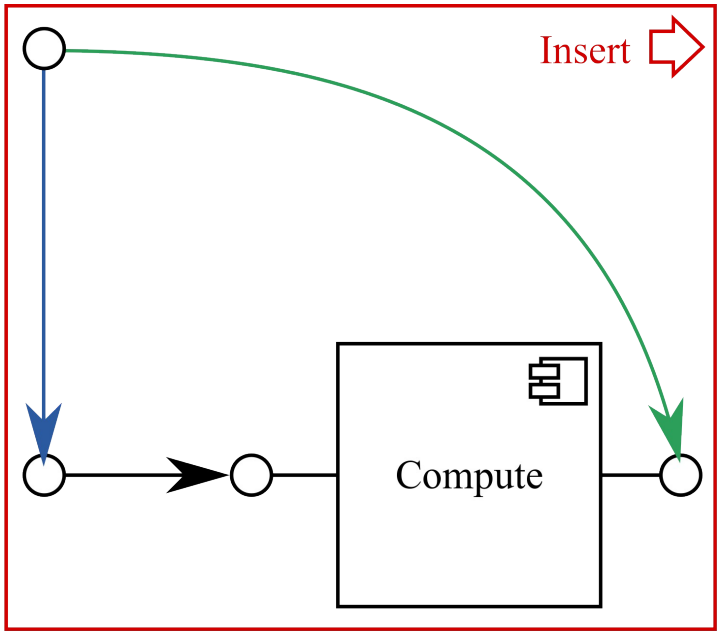
New element: domains

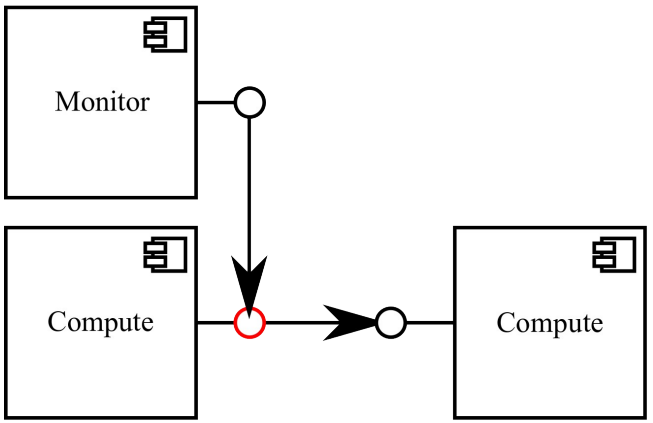
- manage a subassembly
- unit of **locking**
- unit of internal **representation**
- reconfigure their contents

DirectMOD Transformations

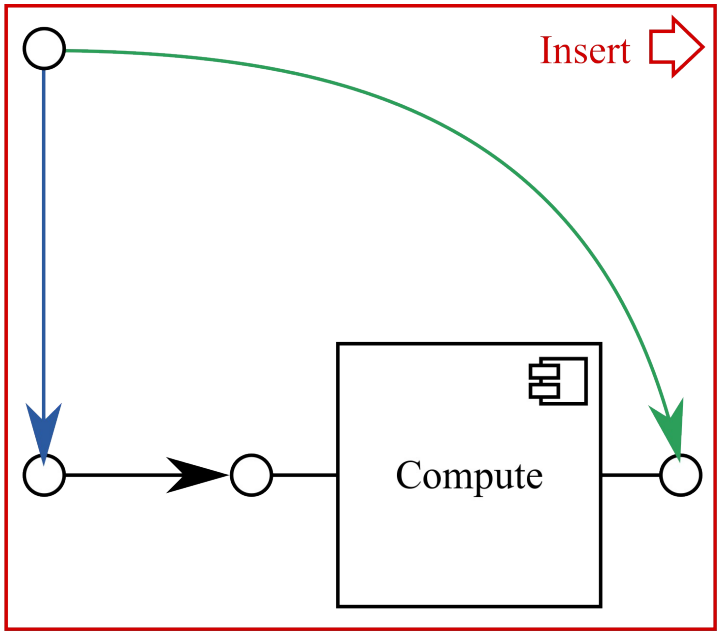


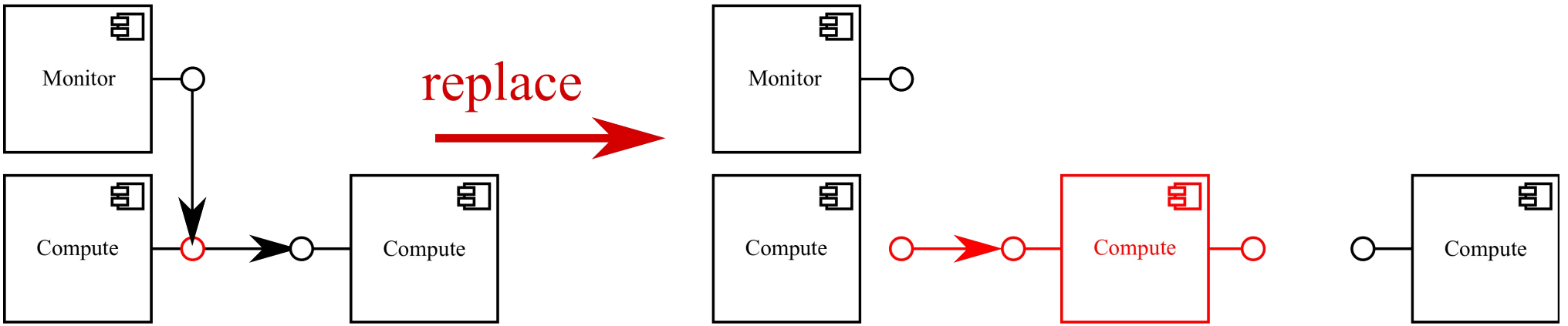
DirectMOD Transformations



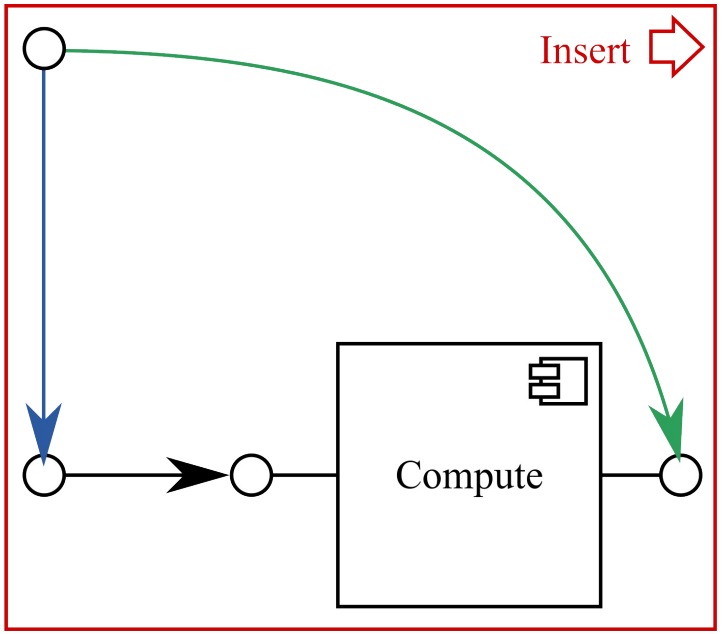


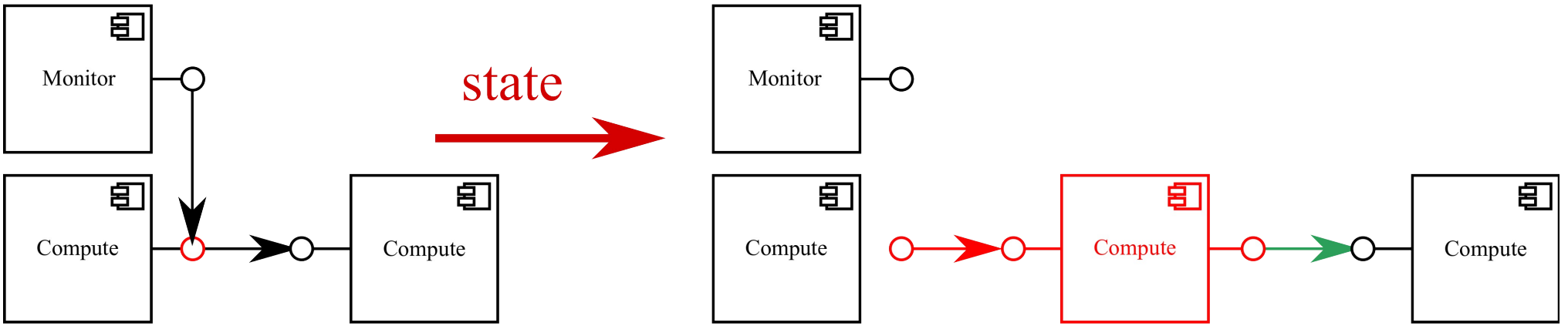
DirectMOD Transformations



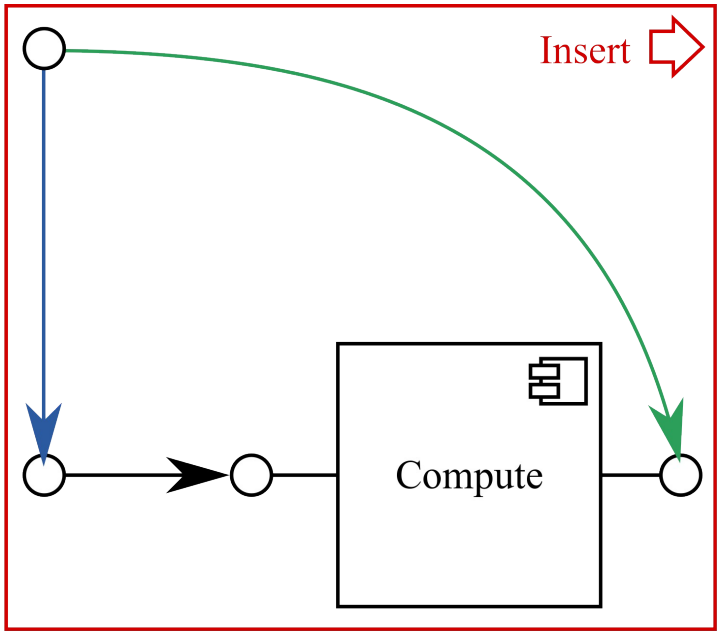


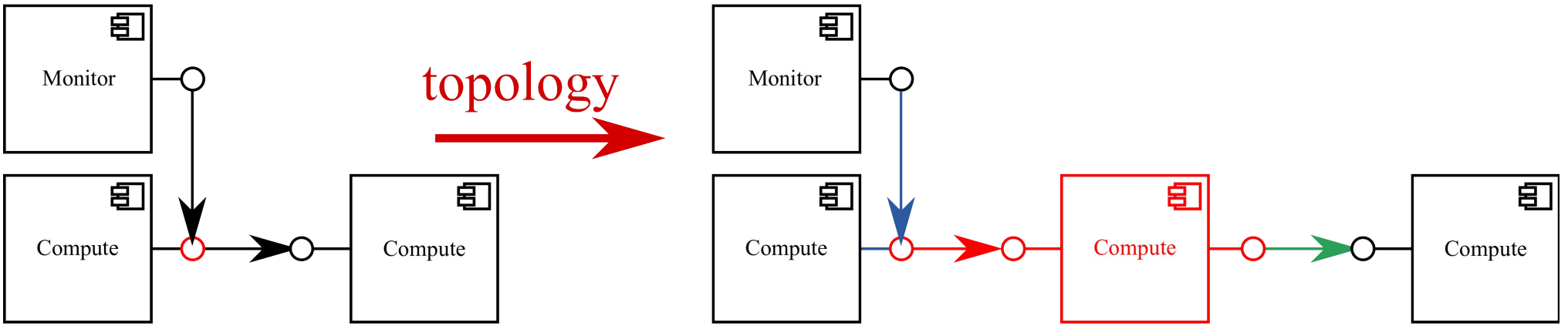
DirectMOD Transformations



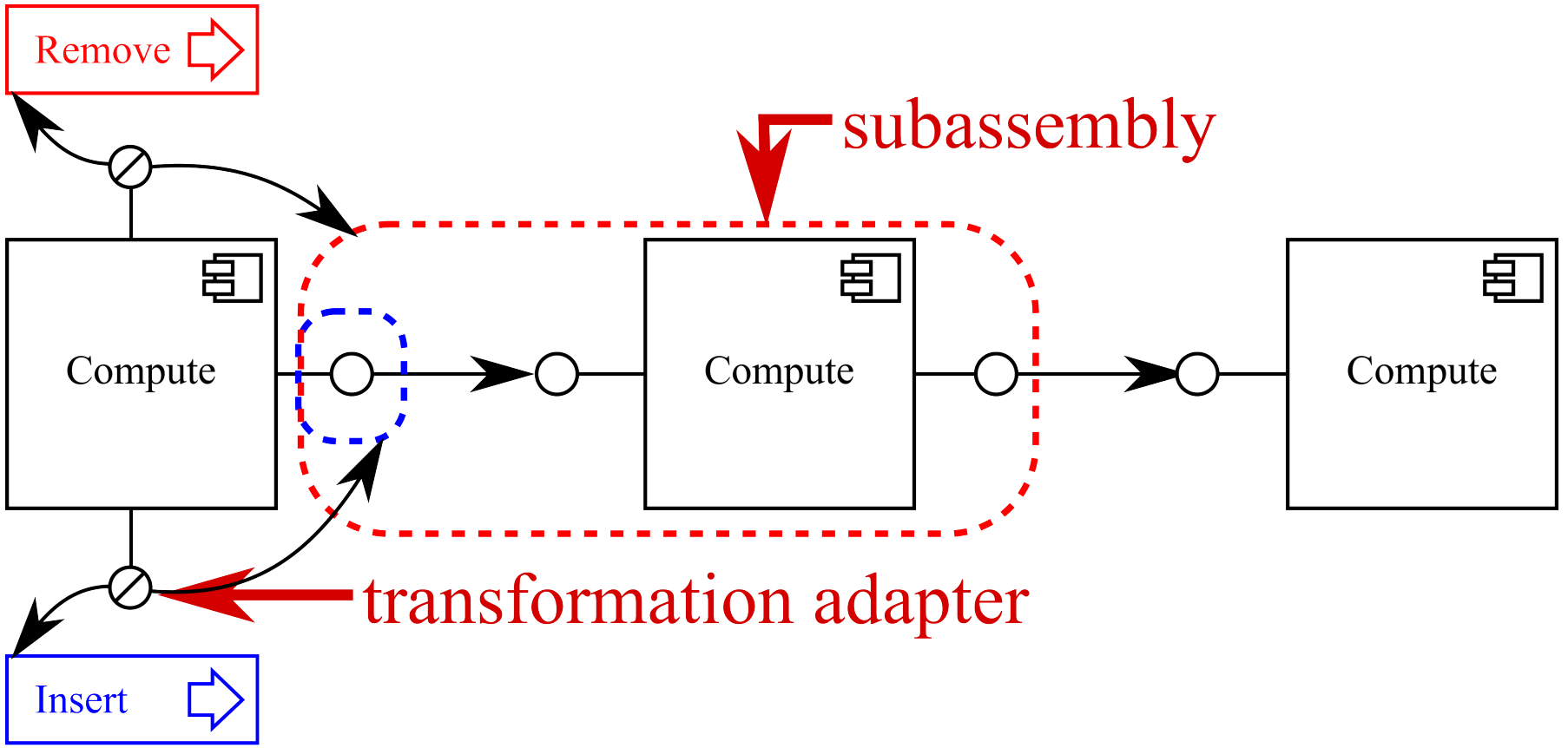


DirectMOD Transformations



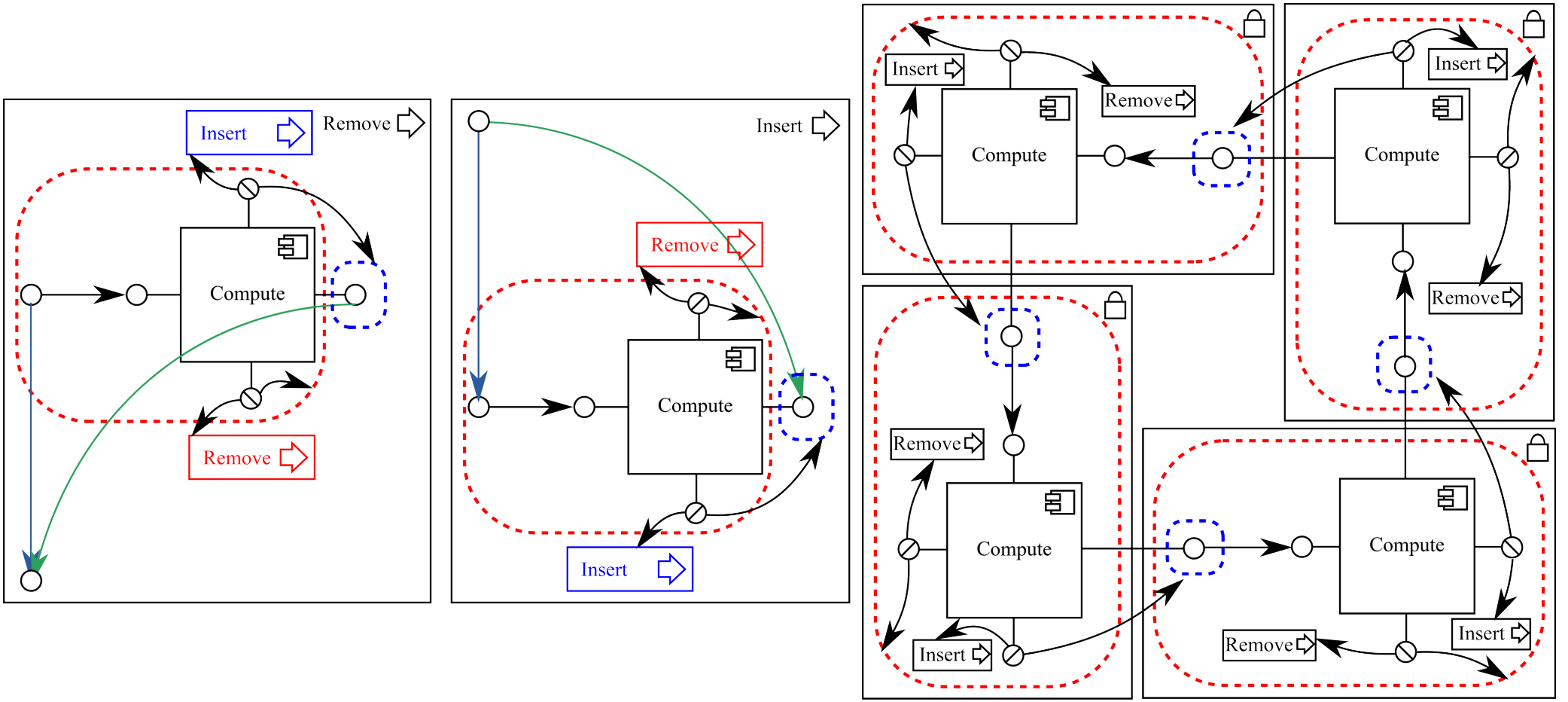


DirectMOD Transformation Adapters

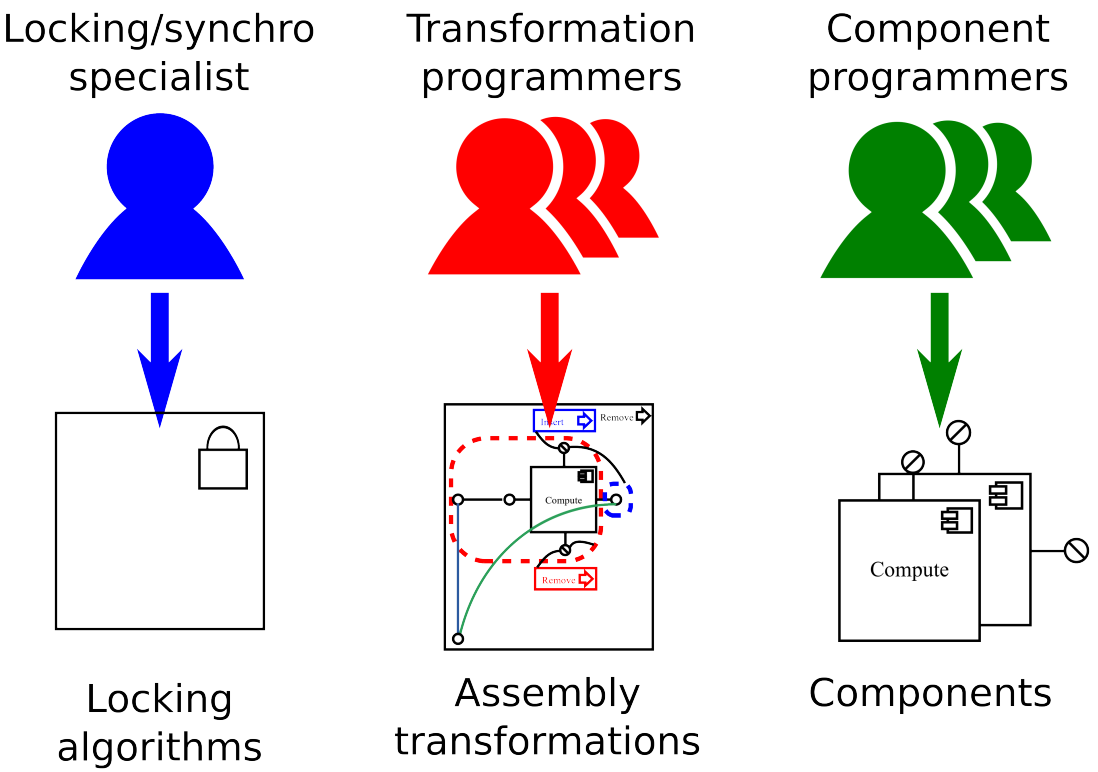


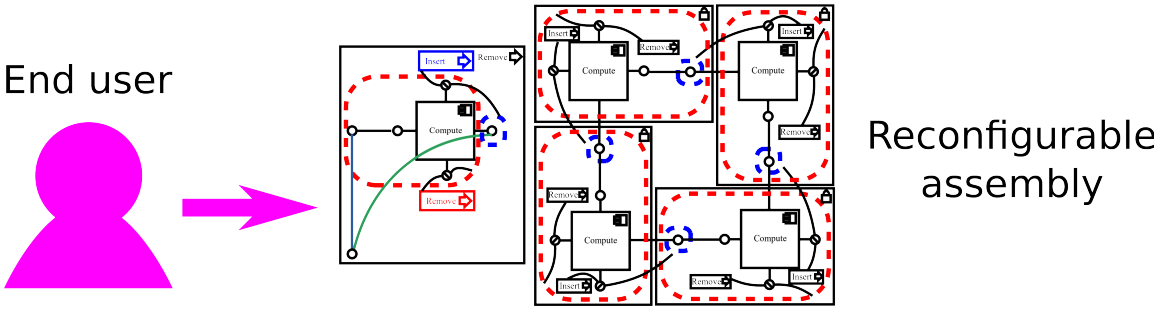
- special kind of port
- reference to transformation and application subassembly

Putting Everything Together



DirectMOD Programming Model





Plan of the Talk

Context and related works

- Related work
- Our proposition

Presentation of the model, DirectMOD

- Model elements
- Putting it all together

Implementation: DirectL2C

- Code
- Performance

Extensions of DirectMOD

- Efficient locking
- High-level language(s) for transformations

Conclusions and perspectives

A C++/MPI Implementation

DirectL2C

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

	Components	Files	mLOC
L2C	37	79	4570
DirectL2C	3	10	1118

L2C provides

- C++ components with zero overhead
- basic component operations
- assembly deployment
  - distributed

DirectL2C provides

- transformation parsing and execution
- interface and locking APIs

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
- complex synchronization code

Transformation Pseudocode

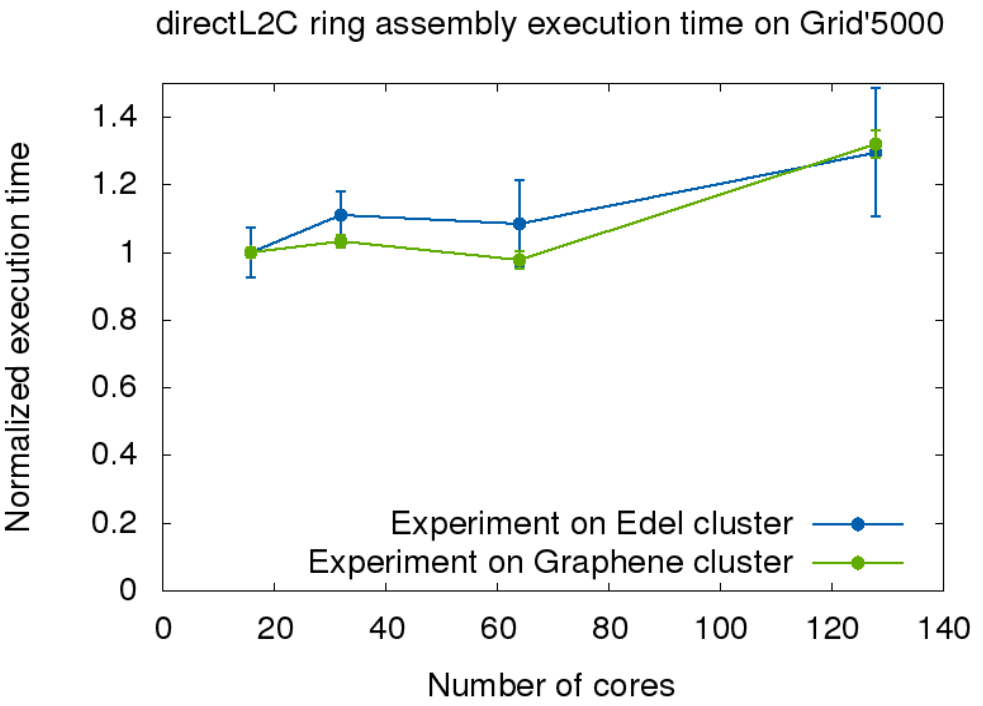
```
Inputdata: portName, reconfPortName, resourceName

Direct::transformation insert();
insert.create("Cp", "newCp", resourceName);
//ports of created component have implicit names
insert.connect(portName, "newCpLeft");
insert.statePort(portName, "newCpRight");
insert.topoPort(reconfPortName, "newCpReconf");
```

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 clusters
- acceptable scalability up to 128 cores

Plan of the Talk

Context and related works

- Related work
- Our proposition

Presentation of the model, DirectMOD

- Model elements
- Putting it all together

Implementation: DirectL2C

- Code
- Performance

Extensions of DirectMOD

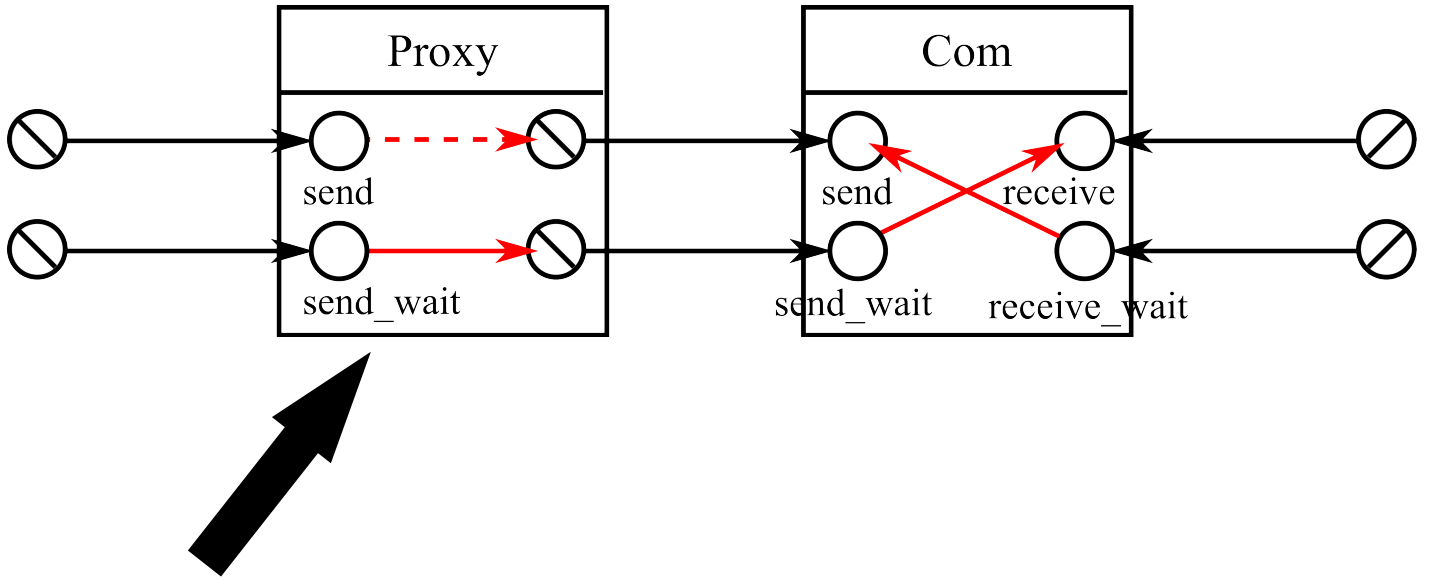
- Efficient locking
- High-level language(s) for transformations

Conclusions and perspectives

Some questions left unanswered

- **what about locking domains?**
  - looks difficult
  - poor reuse
- **what about transformations?**
  - how to write them? language?
  - what about genericity?
  - … hierarchy?

Stopping Components

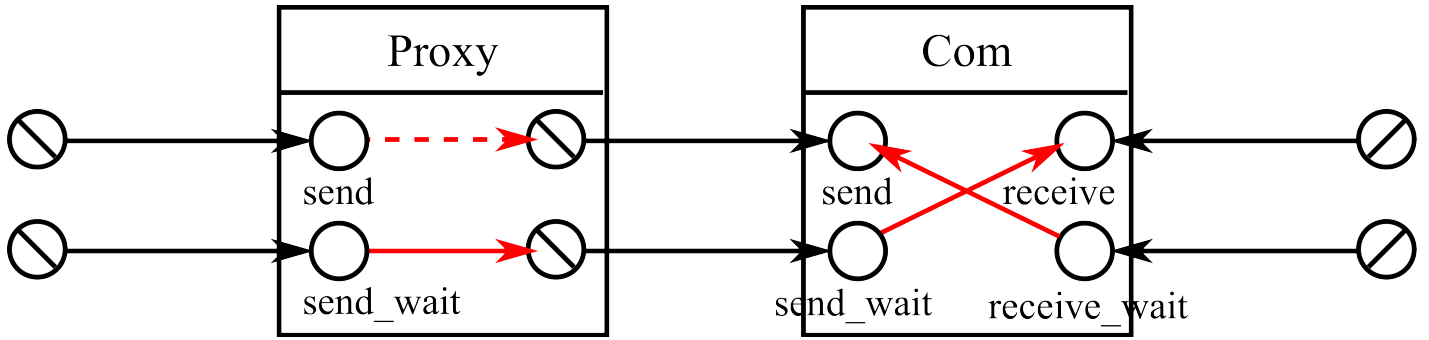


This assembly is running...

... how to stop it?

- no control info
- performance?
- deadlocks?
- call stacks

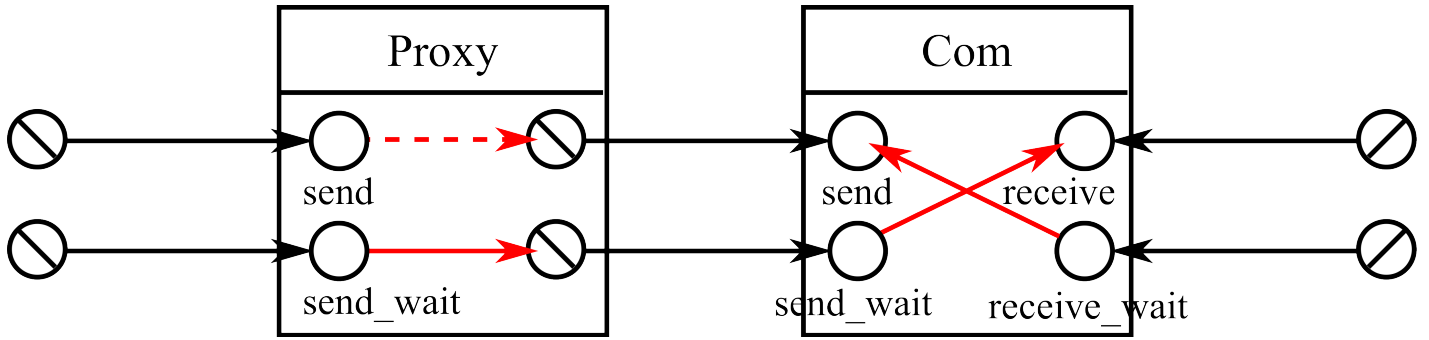
Locking use ports



Approach: locking use ports

- no more calls once locked
- mutex-like behaviour
- in what order?
- deadlocks?

Adding metadata

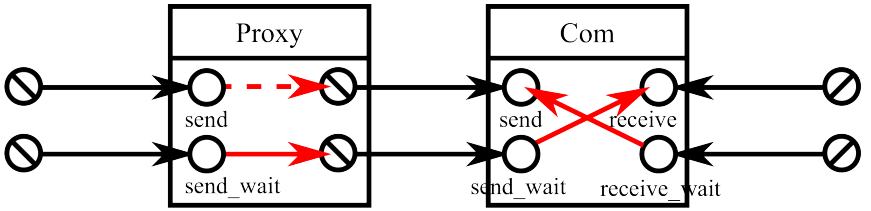


First step: add metadata

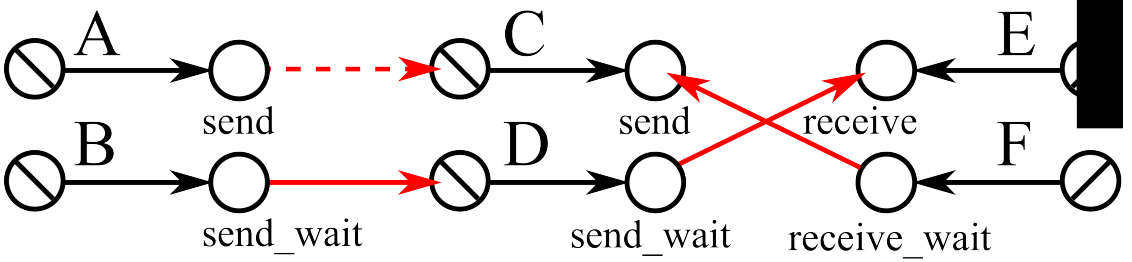
- minimal
- call dependencies
- termination dependencies

Second Step: Locking Order Algorithm

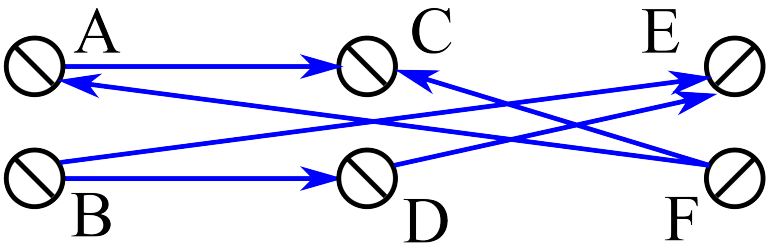
Input:



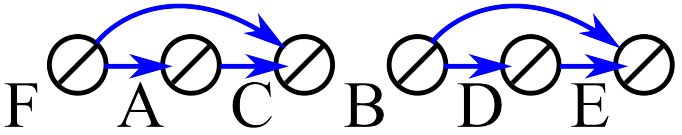
a)



b)



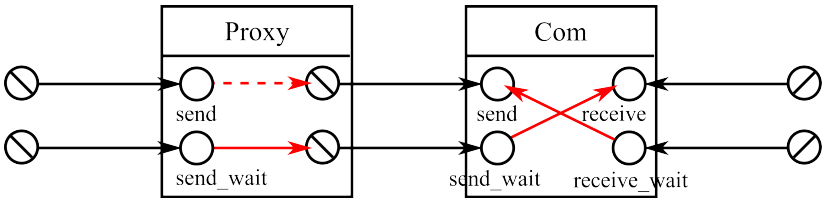
c)



Output: F, A, C, B, D, E

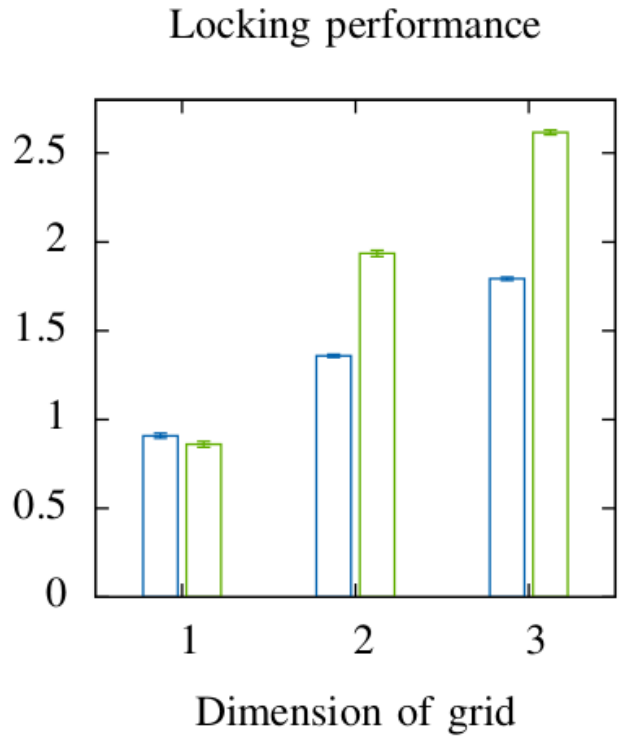
It Works for Stencil Applications

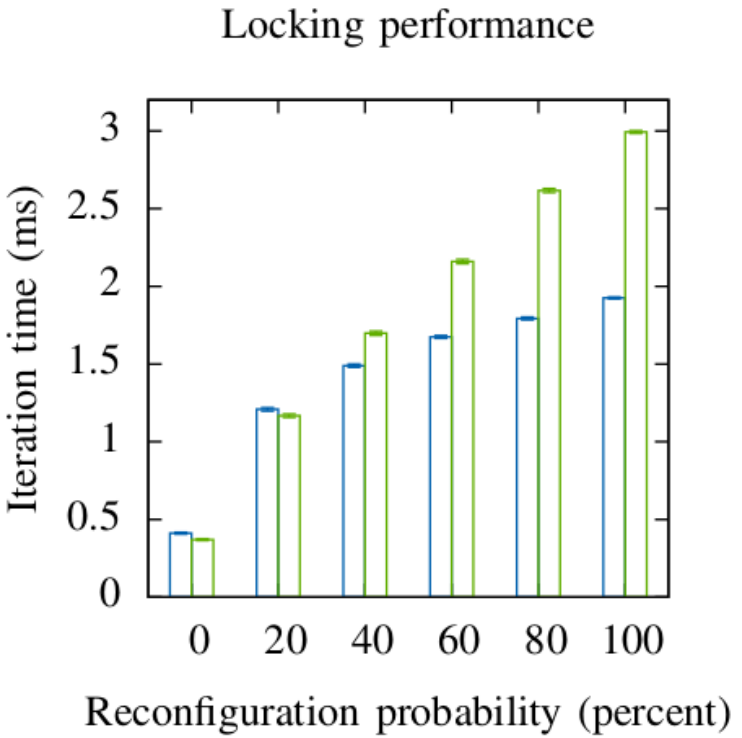
- **stencil applications**
  - simple control
  - concurrent locking of connections

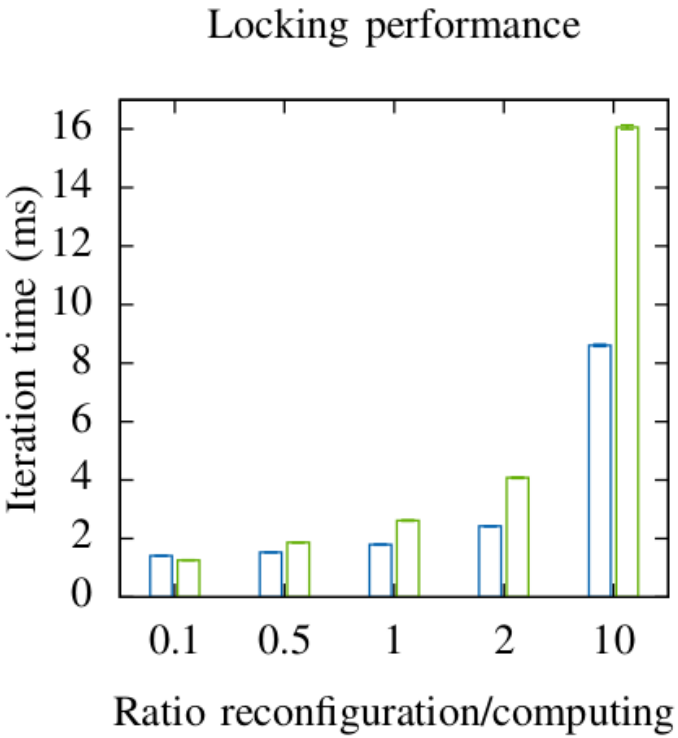


- **useful automation: connection variants**
  - local, distributed
  - proxies
  - 1-to-n connections

Locking Performance







- **1/2/3D grid benchmark w/ locking only**
- **versus component-wise approach**

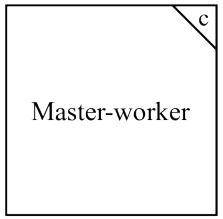
High-level transformation language(s)

- **High-level languages**
  - hierarchy
  - genericity
  - decent compact language for humans (ie, not XML)

- **Transformation to low-level**
  - off-line process
  - optimization during transformation

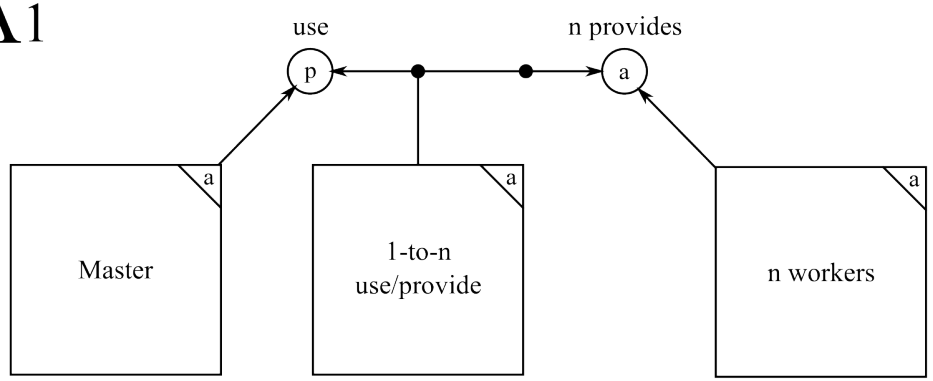
High-level transformation language(s)

A_0



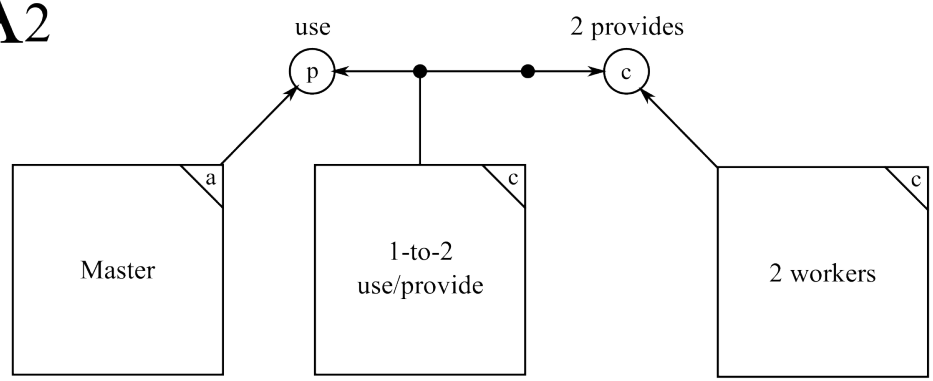
$V(A_0,\Sigma)$ = {
im_c(Master-Worker)
}

A_1



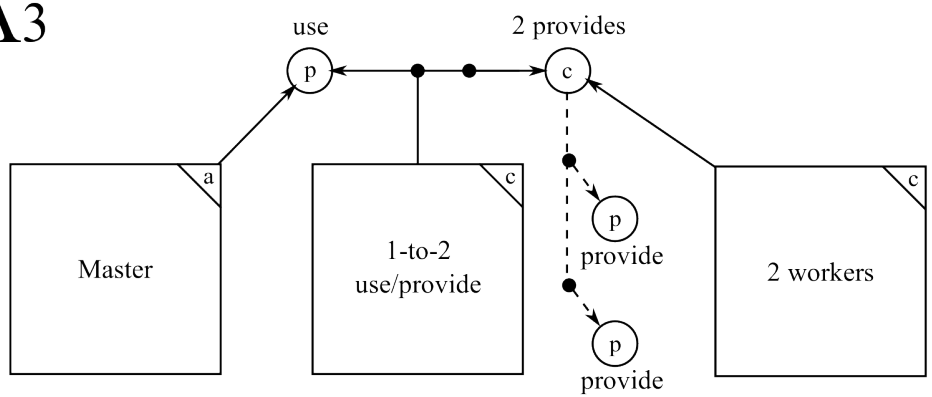
$V(A_1,\Sigma)$ = {
sp_c(Master, Master A)
sp_c(Master, Master B)
sp_c(n workers, 2 workers)
sp_c(n workers, 1 worker)
sp_c(n workers, 2 identical workers)
sp_c(n provides, 2 provides)
sp_c(n provides, 1 provide)
sp_c(1-to-n use/provide, 1-to-2 use/provide)
sp_c(1-to-n use/provide, use/provide)
}

A_2



$V(A_2,\Sigma)$ = {
sp_c(Master, Master A)
sp_c(Master, Master B)
im_c(2 provides)
}

A_3

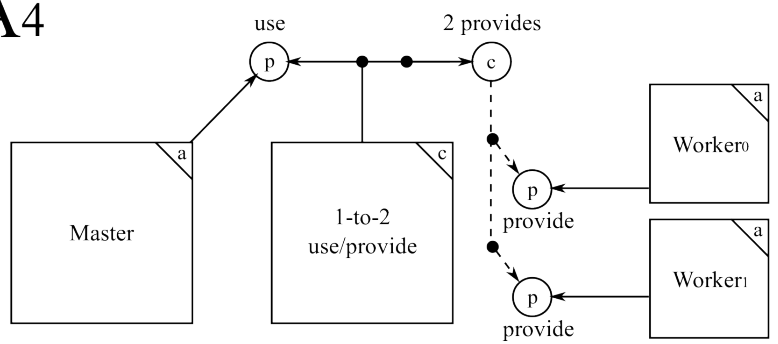


$V(A_3,\Sigma)$ = {
sp_c(Master, Master A)
sp_c(Master, Master B)
im_c(2 workers)
im_c(1-to-2 use/provide)
}

$V(A_5,\Sigma)$ = {
sp_c(Master, Master A)
sp_c(Master, Master B)
spc(Worker$_0$, Worker A)
spc(Worker$_1$, Worker A)
spc(Worker$_0$, Worker B)
spc(Worker$_1$, Worker B)

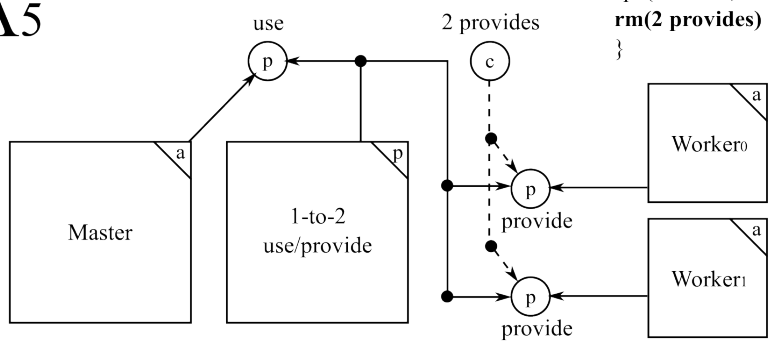
High-level transformation language(s)

A4



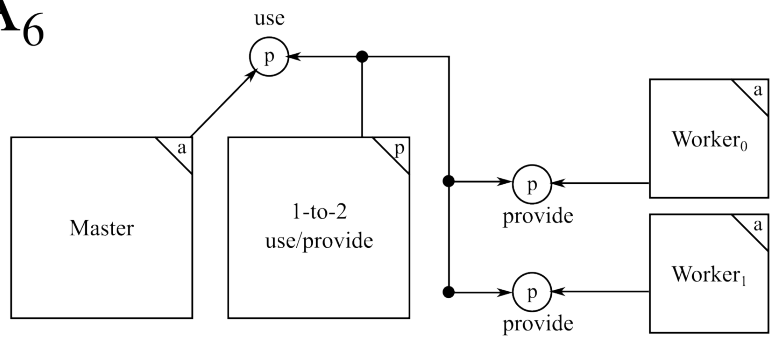
$V(A_4,\Sigma)$ = {
sp_c(Master, Master A)
sp_c(Master, Master B)
spc(Worker$_0$, Worker A)
spc(Worker$_1$, Worker A)
spc(Worker$_0$, Worker B)
spc(Worker$_1$, Worker B)
im$_c$(1-to-2 use/provide)
}

A5



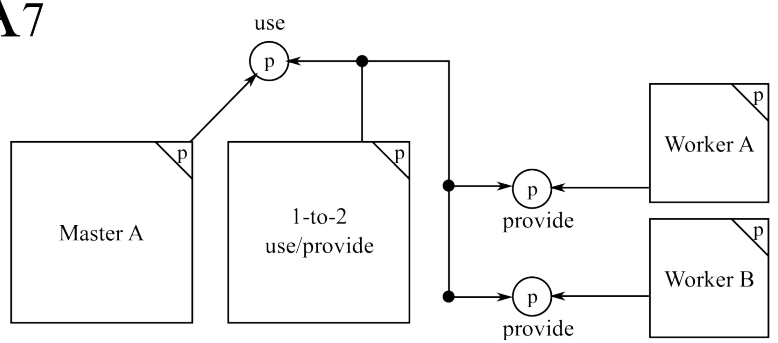
spc(Worker$_0$, Worker B)
spc(Worker$_1$, Worker B)
rm(2 provides)
}

A6



$V(A_6,\Sigma)$ = {
sp_c(Master, Master A)
sp_c(Master, Master B)
spc(Worker$_0$, Worker A)
spc(Worker$_1$, Worker A)
spc(Worker$_0$, Worker B)
spc(Worker$_1$, Worker B)
}

A7



$V(A_7,\Sigma)$ = {
}

Plan of the Talk

Context and related works

- Related work
- Our proposition

Presentation of the model, DirectMOD

- Model elements
- Putting it all together

Implementation: DirectL2C

- Code
- Performance

Extensions of DirectMOD

- Efficient locking
- High-level language(s) for transformations

Conclusions and perspectives

Conclusion and Perspectives

Presented DirectMOD

- a reconfigurable component model
- formal
- distributed
- + implementation (DirectL2C)
- + efficient locking for stencil applications

Perspectives

- improve evaluation & implementation
  - ongoing work on a complex benchmark
  - experiments on a large platform (DARI allowance)
  - + work on directL2C
- transformation specification
  - genericity
  - compact language