Clusters and Computational Grids for Scientific Computing

CCGSC 2008
From Clouds to blue sky research

Unconventional Grid Programming

Thierry Priol, INRIA

Joint work with Jean-Pierre Banâtre & Yann Radenac
From Clouds to blue sky research

Unconventional Grid Programming

Thierry Priol, INRIA

Joint work with Jean-Pierre Banâtre & Yann Radenac
Plan

• Why Unconventional Grid Programming?

• Chemical Programming
  • Principle and examples
  • High Order Chemical Language (HOCL)

• Desktop Grid Programming with HOCL
  • Chemical Desktop Grid
  • Example of a Grid Chemical program and its execution

• Conclusion & Perspectives
Why Unconventional Grid Programming Paradigms?
Why Unconventional Grid Programming Paradigms?

The New York Times  April 1st, 2010
Why Unconventional Grid Programming Paradigms?

The New York Times April 1st, 2010

EU Researchers in High-Energy Physics did not get the Nobel prize due to that prevented them to discover the Higgs Bozon despite billions of € spent to build the Large Hadron Collider and ~100 M€ for the computing infrastructure...
Why Unconventional Grid Programming Paradigms?

The New York Times  April 1st, 2010

EU Researchers in High-Energy Physics did not get the Nobel prize due to an Unreliable Grid Infrastructure that prevented them to discover the Higgs Bozon despite billions of € spent to build the Large Hadron Collider and ~100 M€ for the computing infrastructure.
Why Unconventional Grid Programming Paradigms?

EU Researchers in High-Energy Physics did not get the Nobel prize due to an Unreliable Grid Infrastructure that prevented them to discover the Higgs Boson despite billions of € spent to build the Large Hadron Collider and ~100 M€ for the computing infrastructure.

Early warning from well-known computer scientists:
Why Unconventional Grid Programming Paradigms?

The New York Times  April 1st, 2010

EU Researchers in High-Energy Physics did not get the Nobel prize due to an Unreliable Grid Infrastructure that prevented them to discover the Higgs Boson despite billions of € spent to build the Large Hadron Collider and ~100 M€ for the computing infrastructure.

Early warning from well-known computer scientists:

"You know that you are dealing with a distributed system when you are prevented from getting your work done because a node you never heard of has crashed."

Leslie Lamport
Why Unconventional Grid Programming Paradigms?

The New York Times April 1st, 2010

EU Researchers in High-Energy Physics did not get the Nobel prize due to an Unreliable Grid Infrastructure that prevented them to discover the Higgs Bozon despite billions of € spent to build the Large Hadron Collider and ~100 M€ for the computing infrastructure

Early warning from well-known computer scientists:

"You know that you are dealing with a distributed system when you are prevented from getting your work done because a node you never heard of has crashed."

Leslie Lamport

"Grid environments will require a rethinking of existing programming models and, most likely, new thinking about novel models more suitable for specific characteristics of grid applications and environments."

I. Foster & K. Kesselman
Was just a nightmare or a foreboding?

EGEE Grid infrastructure

- A 9-month study of the SEE-VO (Feb’06-Nov’06) showed that 52% of the jobs failed.
- Some people say it is now 5-10% some others mention 30%?

Grid infrastructures = uncertainty & complexity

- Lack of a “real” global state: should we pay the cost of knowing everything?
- Unprecedented level of complexity
- Hardware and software failures
How to deal with such complexity and uncertainty?

Adaptive and autonomic systems are the most promising approaches to cope with complexity and uncertainty

How to Deal with Complexity?

- Adaptivity is the key for applications to effectively use available resources whose complexity is exponentially increasing
- Goal:
  - Automatically bridge the gap between the application and computers that are rapidly changing and getting more and more complex

J. Dongarra

The Research Challenge: Addressing Uncertainty!

- Must be addressed at multiple levels
  - Algorithms
    - Asynchronous/chaotic, failure tolerant, ...
    - Uncertainty estimation/characterization (probabilistic collocation)
  - Programming systems
    - Adaptive, application/system aware, proactive, ...
  - Infrastructure
    - Decoupled, self-managing, resilient

M. Parashar
Current approaches to design adaptive and autonomic systems

Frameworks for adaptive / autonomic systems

- Many many specialized frameworks depending on the targeted systems (real-time, parallel, distributed) or applications (multimedia, HPC, ...)

But what about a programming model?

- Not only for applications but for Grid middleware as well
- Are there any available that would have autonomic/adaptive behaviors?
Unconventional Programming Paradigms

• Most of them are nature-inspired paradigms
  • Nature has proved to be successful to cope with scalability and faults
    • Scale: the average adult is made up of 100 trillion cells
    • Faults: 50,000,000 of the cells in my body will have died and been replaced with others, all while you have been reading this sentence ... and you did not notice this (hopefully...)

• Some examples
  • Amorphous (agent-based with local interaction)
  • Swarm (global behaviors emerging from local behaviors of swarm members)
  • Bio-inspired (genetic programming, evolutionary, neural, ...)
  • Chemical (analogy with chemical reactions)
Chemical Programming

- Initial work from Jean-Pierre Banâtre and Daniel Le Métayer (1986)
- Programming model using chemistry as a metaphor

<table>
<thead>
<tr>
<th>Programming Objects</th>
<th>Chemistry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data</td>
<td>Molecule</td>
</tr>
<tr>
<td>Multiset</td>
<td>Solution</td>
</tr>
<tr>
<td>Computation</td>
<td>Reaction</td>
</tr>
</tbody>
</table>

- Execution model using chemistry as a metaphor

<table>
<thead>
<tr>
<th>Properties</th>
<th>Chemistry</th>
</tr>
</thead>
<tbody>
<tr>
<td>Implicit parallelism</td>
<td>Brownian motion</td>
</tr>
<tr>
<td>Non determinism</td>
<td></td>
</tr>
</tbody>
</table>
Gamma: a chemical programming model

- Programming by a set of rewriting rules operating on a multiset

\[
\text{replace } x, y \text{ by } x \text{ if } x \div y
\]
Gamma: a chemical programming model

• Programming by a set of rewriting rules operating on a multiset

\[\text{replace } x, y \text{ by } x \text{ if } x \div y\]

\textit{instead of}
Gamma: a chemical programming model

• Programming by a set of rewriting rules operating on a multiset

\[ \text{replace } x, y \text{ by } x \text{ if } x \div y \]

\textit{instead of}

```java
public class Primes {
    static int MAXINT = 100;
    public static void main (String []args){
        int isprime = 1;
        for (int i = 1; i < MAXINT; i++){
            for(int j =1; j<i; j++){
                if (((i%j)==0) & (j==1)){
                    isprime =0;
                }
                else{
                    isprime =1;
                }
            }
            if (isprime==1){
                System.out.println(i);
            }
        }
    }
}
```
Gamma: a chemical programming model

- Programming by a set of rewriting rules operating on a multiset

```
replace x,y by x if x div y
```
Gamma: a chemical programming model

- Programming by a set of rewriting rules operating on a multiset
  \[
  \text{replace } x,y \text{ by } x \text{ if } x \text{ div } y
  \]
- Properties
Gamma: a chemical programming model

- Programming by a set of rewriting rules operating on a multiset
  
  \texttt{replace} \( x,y \) \texttt{by} \( x \) \texttt{if} \( x \div y \)

- Properties
  - Intuitive model
Gamma: a chemical programming model

• Programming by a set of rewriting rules operating on a multiset
  \[\text{replace } x,y \text{ by } x \text{ if } x \div y\]

• Properties
  • Intuitive model
  • Non-determinism
Gamma: a chemical programming model

- Programming by a set of rewriting rules operating on a multiset
  \[
  \text{replace } x, y \text{ by } x \text{ if } x \text{ div } y
  \]

- Properties
  - Intuitive model
  - Non-determinism
  - Mutual exclusion & atomic capture
Gamma: a chemical programming model

- Programming by a set of rewriting rules operating on a multiset
  
  \[\text{replace } x, y \text{ by } x \text{ if } x \text{ div } y\]

- Properties
  - Intuitive model
  - Non-determinism
  - Mutual exclusion & atomic capture
  - Proof of properties (termination)
Gamma: a chemical programming model

• Programming by a set of rewriting rules operating on a multiset
  \[ \text{replace } x, y \text{ by } x \text{ if } x \text{ div } y \]

• Properties
  • Intuitive model
  • Non-determinism
  • Mutual exclusion & atomic capture
  • Proof of properties (termination)
  • Parallel execution is implicit
Gamma: a chemical programming model

- Programming by a set of rewriting rules operating on a multiset
  \[ \text{replace } x,y \text{ by } x \text{ if } x \text{ div } y \]

- Properties
  - Intuitive model
  - Non-determinism
  - Mutual exclusion & atomic capture
  - Proof of properties (termination)
  - Parallel execution is implicit

- A well suited programming model when dealing with systems with unbounded “things”
Gamma: a chemical programming model

• Programming by a set of rewriting rules operating on a multiset
  \[ \text{replace } x, y \text{ by } x \text{ if } x \div y \]

• Properties
  • Intuitive model
  • Non-determinism
  • Mutual exclusion & atomic capture
  • Proof of properties (termination)
  • Parallel execution is implicit

• A well suited programming model when dealing with systems with unbounded “things”
  • No explicit iteration requiring to know the number of “things”
Chemical Programming Principle

• Example: computing prime numbers less than 10
Chemical Programming Principle

- Example: computing prime numbers less than 10

```
replace x, y by x if x div y
```
Chemical Programming Principle

- Example: computing prime numbers less than 10

replace $x,y$ by $x$ if $x \text{ div } y$
Chemical Programming Principle

• Example: computing prime numbers less than 10

\[\text{replace } x, y \text{ by } x \text{ if } x \div y\]
Chemical Programming Principle

• Example: computing prime numbers less than 10

\[
\text{replace } x,y \text{ by } x \text{ if } x \text{ div } y
\]
Chemical Programming Principle

- Example: computing prime numbers less than 10

```
replace x,y by x if x div y
```
Autonomic property
Autonomic property

\[ \langle \text{sieve, 2, 3, 4, 5, 6, 7, 8, 9, 10} \rangle \]

\[ \downarrow^* \]

\[ \langle \text{sieve, 2, 3, 5, 7} \rangle \]
Autonomic property

\[
\langle \text{sieve}, 2, 3, 4, 5, 6, 7, 8, 9, 10 \rangle
\]

Stabilizing

\[\downarrow^*\]

\[
\langle \text{sieve}, 2, 3, 5, 7 \rangle
\]

Perturbation

\[
\langle \text{sieve}, 2, 3, 5, 7, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20 \rangle
\]
**Autonomic property**

\[
\langle \text{sieve, 2, 3, 4, 5, 6, 7, 8, 9, 10} \rangle
\]

\[
\downarrow^* 
\]

**Stabilizing**

\[
\langle \text{sieve, 2, 3, 5, 7} \rangle
\]

**Perturbation**

\[
\langle \text{sieve, 2, 3, 5, 7, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20} \rangle
\]

\[
\downarrow^* 
\]

**Re-stabilizing**

\[
\langle \text{sieve, 2, 3, 5, 7, 11, 13, 17, 19} \rangle
\]
Autonomic property

\[ \langle \text{sieve, 2, 3, 4, 5, 6, 7, 8, 9, 10} \rangle \]

**Stabilizing**
\[ \downarrow^* \]
\[ \langle \text{sieve, 2, 3, 5, 7} \rangle \]

**Perturbation**
\[ \langle \text{sieve, 2, 3, 5, 7, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20} \rangle \]

**Re-stabilizing**
\[ \downarrow^* \]
\[ \langle \text{sieve, 2, 3, 5, 7, 11, 13, 17, 19} \rangle \]
Autonomic property

\[ \text{Autonomic property} \]

\[ \langle \text{sieve, 2, 3, 4, 5, 6, 7, 8, 9, 10} \rangle \]

\[ \downarrow^* \]

\[ \langle \text{sieve, 2, 3, 5, 7} \rangle \]

**Stabilizing**

\[ \langle \text{sieve, 2, 3, 5, 7, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20} \rangle \]

\[ \downarrow^* \]

\[ \langle \text{sieve, 2, 3, 5, 7, 11, 13, 17, 19} \rangle \]

**Perturbation**

**Re-stabilizing**

**Multiset external I/O**

- Collect
- Act
- Autonomic System
- Decide
- Analyze
Autonomic property

\[ \langle \text{sieve, 2, 3, 4, 5, 6, 7, 8, 9, 10} \rangle \]
\[ \downarrow^* \]
\[ \langle \text{sieve, 2, 3, 5, 7} \rangle \]

Stabilizing

\[ \langle \text{sieve, 2, 3, 5, 7, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20} \rangle \]
\[ \downarrow^* \]
\[ \langle \text{sieve, 2, 3, 5, 7, 11, 13, 17, 19} \rangle \]

Perturbation

Re-stabilizing

Multiset external I/O

Applying rules

Autonomic System

Collect

Analyze

Decide

Act
Higher Order Chemical Language (HOCL)

- Higher-order extension of the Gamma language
  - Reaction rules are molecules like data elements
  - Reaction rules can replace reactions rules in an inert solution only
- Based on the $\gamma$-calculus
- A HOCL program is a chemical solution of molecules $\langle M_1, \ldots, M_n \rangle$
  - A molecule can be an atom
  - Atoms $A_i$ may be:
    - Integers, strings, . . . any external object
    - Tuples $A_1: \cdots : A_k$
    - Sub-solutions
    - One-shot rules: one $P$ by $M$ if $C$
    - N-shot rules: replace $P$ by $M$ if $C$
- Multiplicity: $X^2, X^\infty, X^{-1}$
HOCL example: the high-order

The greatest prime number that is less than 10:

\[
\begin{align*}
\text{let } & \text{sieve } = \text{replace } x, y \text{ by } x \text{ if } x \text{ div } y \text{ in} \\
\text{let } & \text{max } = \text{replace } x, y \text{ by } x \text{ if } x \geq y \text{ in} \\
\langle \langle \text{sieve}, 2, 3, 4, 5, 6, 7, 8, 9, 10 \rangle, \text{one} \langle \text{sieve } = x, \omega \rangle \text{ by } \omega, \text{max} \rangle
\end{align*}
\]
HOCL example: the high-order

The greatest prime number that is less than 10:

\[
\text{let sieve} = \text{replace x, y by x if x div y in}
\]

\[
\text{let max} = \text{replace x, y by x if x \geq y in}
\]

\[
\langle\langle\text{sieve}, 2, 3, 4, 5, 6, 7, 8, 9, 10\rangle, \text{one}\langle\text{sieve} = x, \omega\rangle \text{by } \omega, \text{max}\rangle
\]
HOCL example: multiplicity

\[
\langle 2^2, 5, 3^{-2} \rangle, \langle 3, 5, 2^{-3} \rangle, \text{ one } \langle f, g \rangle \text{ by } \langle f, g \rangle
\]

\[
\downarrow^* \\
\langle 5^2, 3^{-1}, 2^{-1} \rangle
\]
Chemical Desktop Grid

- Grid viewed as a chemical solution
  - Resources = solutions/molecules
  - Coordination = chemical reactions

- Chemical program as
  - A specification of the application
    - Applications represented as a set of rules and data elements
    - Done by the application programmer
  - A specification of the coordination
    - Coordination represented a set of rules and data elements
    - Mapping of rules to solutions representing resources
    - Done by a Grid specialist
Chemical Desktop Grid

- Grid viewed as a chemical solution
  - Resources = solutions/molecules
  - Coordination = chemical reactions

- Chemical program as
  - A specification of the application
    - Applications represented as a set of rules and data elements
    - Done by the application programmer
  - A specification of the coordination
    - Coordination represented a set of rules and data elements
    - Mapping of rules to solutions representing resources
    - Done by a Grid specialist
Computing prime numbers for a Chemical Desktop Grid

```
let findLocalReactives = replace x,y by x:y if x div y in

let runSieve = replace r:(x:y, ω) by r:(x, ω) in

let split = replace r1:(x1:y1, x2:y2, ω1, ω2), r2:() by
  r1:(x1:y1, ω1),
  r2:(findLocalReactives, x2:y2, ω2) in

let findDistReactives = replace r1:(x, ω1), r2:(y, ω2) by
  r1:(x:y, ω1), r2:(ω2) if x div y in

let newRes = replace ω by Grid.getNewRes(): {}, ω if Grid.newResAvailable() in

let remRes = replace r:(), ω by ω if Grid.willBeRemoved(r) in

let migrate = replace r1:⟨ω⟩, r2:() by r1:⟨⟩, r2:⟨ω⟩ if
  Grid.willBeRemoved(r1) ∧
  not Grid.willBeRemoved(r2) in

⟨R1:(findLocalReactives, 2, 3, 4, . . . , 1000),
R2:(), . . . , Rn:(),
runSieve, split, findDistReactives,
newRes, remRes, migrate⟩
```
Computing prime numbers for a Chemical Desktop Grid

```
let findLocalReactives = replace x,y by x:y if x div y in

let runSieve = replace r:⟨x:y, ω⟩ by r:⟨x, ω⟩ in

let split = replace r1:⟨x1:y1, x2:y2, ω1, ω2⟩, r2:⟨⟩ by
            r1:⟨x1:y1, ω1⟩,
            r2:⟨findLocalReactives, x2:y2, ω2⟩ in

let findDistReactives = replace r1:⟨x, ω1⟩, r2:⟨y, ω2⟩ by
                        r1:⟨x:y, ω1⟩,
                        r2:⟨ω2⟩ if x div y in

let newRes = replace ω by Grid.getNewRes(): ⟨⟩, ω if Grid.newResAvailable() in
let remRes = replace r:⟨⟩, ω by ω if Grid.willBeRemoved(r) in

let migrate = replace r1:⟨ω⟩, r2:⟨⟩ by r1:⟨⟩, r2:⟨ω⟩ if Grid.willBeRemoved(r1) ^
                 not Grid.willBeRemoved(r2) in

⟨R1:⟨findLocalReactives, 2, 3, 4, . . . , 1000⟩,
  R2:⟨⟩, . . . , Rn:⟨⟩,
  runSieve, split, findDistReactives,
  newRes, remRes, migrate⟩
```
Computing prime numbers for a Chemical Desktop Grid

```plaintext
let findLocalReactives = replace x,y by x:y if x div y in

let runSieve = replace r:(x:y, ω) by r:(x, ω) in

let split = replace r1:(x1:y1, x2:y2, ω1 , ω2), r2:() by r1:(x1:y1, ω1),
            r2:(findLocalReactives, x2:y2, ω2) in

let findDistReactives = replace r1:(x, ω1), r2:(y, ω2) by r1:(x:y, ω1), r2:(ω2)
                      if x div y in

let newRes = replace ω by Grid.getNewRes(): (), ω if Grid.newResAvailable() in

let remRes = replace r:(), ω by ω if Grid.willBeRemoved(r) in

let migrate = replace r1 : ⟨ω⟩, r2 :() by r1 : ⟨⟩, r2 :⟨ω⟩
              if Grid.willBeRemoved(r1) ∧
              not Grid.willBeRemoved(r2) in

⟨R1:(findLocalReactives, 2, 3, 4, . . . , 1000),
R2:(), . . . , Rn:(),
runSieve, split, findDistReactives,
newRes, remRes, migrate⟩
```
Computing prime numbers for a Chemical Desktop Grid

```
let findLocalReactives = replace x,y by x:y if x div y in

let runSieve = replace r:(x:y, ω) by r:(x, ω) in

let split = replace r1:(x1:y1, x2:y2, ω1, ω2), r2:() by r1:(x1:y1, ω1),
              r2:⟨findLocalReactives, x2:y2, ω2⟩ in

let findDistReactives = replace r1:(x, ω1), r2:(y, ω2) by r1:(x:y, ω1),
                         r2:(ω2) if x div y in

let newRes = replace ω by Grid.getNewRes(): (), ω if Grid.newResAvailable() in

let remRes = replace r:(), ω by ω if Grid.willBeRemoved(r) in

let migrate = replace r1:(ω), r2:() by r1:(), r2:(ω) if Grid.willBeRemoved(r1) ∧
               not Grid.willBeRemoved(r2) in

⟨R1:⟨findLocalReactives, 2, 3, 4, . . . , 1000⟩, R2:(), . . . , Rn:(),
runSieve, split, findDistReactives,
newRes, remRes, migrate⟩
```
Computing prime numbers for a Chemical Desktop Grid

let findLocalReactives = replace x,y by x:y if x div y in

let runSieve = replace r:⟨x:y, ω⟩ by r:⟨x, ω⟩ in

let split = replace r1:⟨x1:y1, x2:y2, ω1, ω2⟩, r2:⟨⟩ by
                 r1:⟨x1:y1, ω1⟩,
                 r2:⟨findLocalReactives, x2:y2, ω2⟩ in

let findDistReactives = replace r1:⟨x, ω1⟩, r2:⟨y, ω2⟩ by
                          r1:⟨x:y, ω1⟩,
                          r2:⟨ω2⟩ if x div y in

let newRes = replace ω by Grid.getNewRes(): ⟨⟩, ω
              if Grid.newResAvailable() in

let remRes = replace r:⟨⟩, ω by ω
             if Grid.willBeRemoved(r) in

let migrate = replace r1:⟨ω⟩, r2:⟨⟩ by r1:⟨⟩, r2:⟨ω⟩
              if Grid.willBeRemoved(r1) ∧
              not Grid.willBeRemoved(r2) in

⟨R1:⟨findLocalReactives, 2, 3, 4, . . . , 1000⟩, R2:⟨⟩, . . . , Rn:⟨⟩, runSieve, split, findDistReactives, newRes, remRes, migrate⟩
Computing prime numbers for a Chemical Desktop Grid

let findLocalReactives = replace x,y by x:y if x div y in

let runSieve = replace r:(x:y, ω) by r:(x, ω) in

let split = replace r1:(x1:y1, x2:y2, ω1 , ω2), r2:() by
           r1:(x1:y1, ω1),
           r2:(findLocalReactives, x2:y2, ω2) in

let findDistReactives = replace r1:(x, ω1), r2:(y, ω2) by
                       r1:(x:y, ω1), r2:(ω2)
                       if x div y in

let newRes = replace ω by Grid.getNewRes(): {}, ω if Grid.newResAvailable() in

let remRes = replace r: {}, ω by ω if Grid.willBeRemoved(r) in

let migrate = replace r1 : ⟨ω⟩, r2 :⟨ω⟩ by r1 : ⟨⟩, r2 :⟨ω⟩
               if Grid.willBeRemoved(r1) \∧
               not Grid.willBeRemoved(r2) in

⟨R1:(findLocalReactives, 2, 3, 4, . . . , 1000),
 R2:, . . . , Rn:, runSieve, split, findDistReactives,
 newRes, remRes, migrate⟩
Computing prime numbers for a Chemical Desktop Grid

```ml
let findLocalReactives = replace x,y by x:y if x div y in

let runSieve = replace r:(x:y, ω) by r:(x, ω) in

let split = replace r1:(x1:y1, x2:y2, ω1, ω2), r2:() by r1:(x1:y1, ω1),
              r2:(findLocalReactives, x2:y2, ω2) in

let findDistReactives = replace r1:(x, ω1), r2:(y, ω2) by r1:(x:y, ω1), r2:(ω2)
                           if x div y in

let newRes = replace ω by Grid.getNewRes(): (), ω if Grid.newResAvailable() in

let remRes = replace r:(), ω by ω if Grid.willBeRemoved(r) in

let migrate = replace r1 : ⟨ω⟩, r2 :() by r1 : ⟨⟩, r2 :⟨ω⟩
             if Grid.willBeRemoved(r1) ^ not Grid.willBeRemoved(r2) in

⟨R1:(findLocalReactives, 2, 3, 4, ..., 1000),
 R2:(), ..., Rn:(),
 runSieve, split, findDistReactives,
 newRes, remRes, migrate⟩
```
Computing prime numbers for a Chemical Desktop Grid

```plaintext
let findLocalReactives = replace x,y by x:y if x div y in

let runSieve = replace r:(x:y, ω) by r:(x, ω) in

let split = replace r1:(x1:y1, x2:y2, ω1, ω2), r2:() by r1:(x1:y1, ω1), r2:(findLocalReactives, x2:y2, ω2) in

let findDistReactives = replace r1:(x, ω1), r2:(y, ω2) by r1:(x:y, ω1), r2:(ω2) if x div y in

let newRes = replace ω by Grid.getNewRes(): ω, ω if Grid.newResAvailable() in

let remRes = replace r:(), ω by ω if Grid.willBeRemoved(r) in

let migrate = replace r1 : ⟨ω⟩, r2 :⟨ω⟩ by r1 : ⟨⟩, r2 :⟨ω⟩ if Grid.willBeRemoved(r1) ∧ not Grid.willBeRemoved(r2) in

⟨R1:(findLocalReactives, 2, 3, 4, . . . , 1000), R2:(), . . . , Rn:(), runSieve, split, findDistReactives, newRes, remRes, migrate⟩
```

React with two elements that belong to two distinct solutions
Computing prime numbers for a Chemical Desktop Grid

let findLocalReactives = replace x,y by x:y if x div y in

let runSieve = replace r:(x:y, ω) by r:(x, ω) in

let split = replace r1:(x1:y1, x2:y2, ω1 , ω2), r2:() by r1:(x1:y1, ω1),
             r2:(findLocalReactives, x2:y2, ω2) in

let findDistReactives = replace r1:(x, ω1), r2:(y, ω2) by r1:(x:y, ω1), r2:(ω2)
if x div y in

let newRes = replace ω by Grid.getNewRes(): (), ω if Grid.newResAvailable() in

let remRes = replace r:(), ω by ω if Grid.willBeRemoved(r) in

let migrate = replace r1 : ⟨ω⟩, r2 :⟨ω⟩ by r1 : ⟨⟩, r2 :⟨ω⟩
if Grid.willBeRemoved(r1) ∧ not Grid.willBeRemoved(r2) in

⟨R1:(findLocalReactives, 2, 3, 4, . . ., 1000),
 R2:⟨⟩, . . ., Rn:⟨⟩,
 runSieve, split, findDistReactives,
 newRes, remRes, migrate⟩
Computing prime numbers for a Chemical Desktop Grid

```
let findLocalReactives = replace x, y by x:y if x div y in

let runSieve = replace r:⟨x:y, ω⟩ by r:⟨x, ω⟩ in

let split = replace r1:⟨x1:y1, x2:y2, ω1, ω2⟩, r2:⟨⟩
  by r1:⟨x1:y1, ω1⟩,
  r2:⟨findLocalReactives, x2:y2, ω2⟩ in

let findDistReactives = replace r1:⟨x, ω1⟩, r2:⟨y, ω2⟩
  by r1:⟨x:y, ω1⟩, r2:⟨ω2⟩
  if x div y in

let newRes = replace ω by Grid.getNewRes(): ⟨⟩, ω
  if Grid.newResAvailable() in

let remRes = replace r:⟨⟩, ω by ω
  if Grid.willBeRemoved(r) in

let migrate = replace r1:⟨ω⟩, r2:⟨⟩ by r1:⟨⟩, r2:⟨ω⟩
  if Grid.willBeRemoved(r1) ∧
  not Grid.willBeRemoved(r2) in

⟨R1:⟨findLocalReactives, 2, 3, 4, . . . , 1000⟩, 
R2:⟨⟩, . . . , Rn:⟨⟩, 
runSieve, split, findDistReactives, 
newRes, remRes, migrate⟩
```
Computing prime numbers for a Chemical Desktop Grid

```haskell
let findLocalReactives = replace x,y by x:y if x div y in

let runSieve = replace r:(x:y, ω) by r:(x, ω) in

let split = replace r1:(x1:y1, x2:y2, ω1, ω2), r2:() by
    r1:(x1:y1, ω1), r2:(findLocalReactives, x2:y2, ω2) in

let findDistReactives = replace r1:(x, ω1), r2:(y, ω2) by
    r1:(x:y, ω1), r2:(ω2) if x div y in

let newRes = replace ω by Grid.getNewRes(): (), ω if Grid.newResAvailable() in
let remRes = replace r:(), ω by ω if Grid.willBeRemoved(r) in

let migrate = replace r1:(ω), r2:() by r1:(), r2:(ω) if Grid.willBeRemoved(r1) ∧
    not Grid.willBeRemoved(r2) in

⟨R1:(findLocalReactives, 2, 3, 4, . . . , 1000),
R2:(), . . . , Rn:(),
runSieve, split, findDistReactives,
newRes, remRes, migrate⟩
```
Migrate elements from one resource to another one
Computing prime numbers for a Chemical Desktop Grid

```plaintext
let findLocalReactives = replace x,y by x:y if x div y in

let runSieve = replace r:(x:y, ω) by r:(x, ω) in

let split = replace r1:(x1:y1, x2:y2, ω1 , ω2), r2:() by r1:(x1:y1, ω1),
            r2:(findLocalReactives, x2:y2, ω2) in

let findDistReactives = replace r1:(x, ω), r2:(y, ω2) by r1:(x:y, ω1), r2:(ω2) if x div y in

let newRes = replace ω by Grid.getNewRes(): ⟨⟩, ω if Grid.newResAvailable() in

let remRes = replace r:⟨⟩, ω by ω if Grid.willBeRemoved(r) in

let migrate = replace r1 : ⟨ω⟩, r2 :⟨ω⟩ by r1 : ⟨⟩, r2 :⟨ω⟩ if Grid.willBeRemoved(r1) ^
              not Grid.willBeRemoved(r2) in

⟨R1:(findLocalReactives, 2, 3, 4, . . . , 1000),
 R2:⟨⟩, . . . , Rn:⟨⟩,
 runSieve, split, findDistReactives,
 newRes, remRes, migrate⟩
```
Computing prime numbers for a Chemical Desktop Grid

```
let findLocalReactives = replace x,y by x:y if x div y in

let runSieve = replace r:(x:y, ω) by r:(x, ω) in

let split = replace r1:(x1:y1, x2:y2, ω1 , ω2), r2:() by
by r1:(x1:y1, ω1),
  r2:(findLocalReactives, x2:y2, ω2) in

let findDistReactives = replace r1:(x, ω1), r2:(y, ω2) by
by r1:(x:y, ω1), r2:(ω2) if x div y in

let newRes = replace ω by Grid.getNewRes(): {}, ω if Grid.newResAvailable() in

let remRes = replace r: {}, ω by ω if Grid.willBeRemoved(r) in

let migrate = replace r1:(), r2:() by r1:(), r2:() if Grid.willBeRemoved(r1) ∧
  not Grid.willBeRemoved(r2) in

(R1:(findLocalReactives, 2, 3, 4, . . . , 1000),
 R2:{}, . . . , Rn:{},
 runSieve, split, findDistReactives,
 newRes, remRes, migrate)
```
Computing prime numbers for a Chemical Desktop Grid

```
let findLocalReactives = replace x,y by x:y if x div y in

let runSieve = replace r:⟨x:y, ω⟩ by r:⟨x⟩ in

let split = replace r1:⟨x1:y1, x2:y2, ω1, ω2⟩, r2:⟨⟩
    by r1:⟨x1:y1, ω1⟩,
    r2:⟨findLocalReactives, x2:y2, ω2⟩ in

let findDistReactives = replace r1:⟨x, ω1⟩, r2:⟨y, ω2⟩
    by r1:⟨x:y, ω1⟩,
    r2:⟨ω2⟩ if x div y in

let newRes = replace ω by Grid.getNewRes(): ⟨⟩, ω
    if Grid.newResAvailable() in

let remRes = replace r:⟨⟩, ω by ω
    if Grid.willBeRemoved(r) in

let migrate = replace r1:⟨ω⟩, r2:⟨⟩ by r1:⟨⟩, r2:⟨ω⟩
    if Grid.willBeRemoved(r1) ∧
    not Grid.willBeRemoved(r2) in

⟨R1:⟨findLocalReactives, 2, 3, 4, . . . , 1000⟩,
R2:⟨⟩, . . . , Rn:⟨⟩,
runSieve, split, findDistReactives,
newRes, remRes, migrate⟩
```
A possible execution: local reactions within R1

let findLocalReactives = replace x,y by x:y if x div y in

let runSieve = replace r:(x:y, ω) by r:(x, ω) in

let split = replace r1:(x1:y1, x2:y2, ω1, ω2), r2:() by r1:(x1:y1, ω1), r2:(findLocalReactives, x2:y2, ω2) in

let findDistReactives = replace r1:(x, ω1), r2:(y, ω2) by r1:(x:y, ω1), r2:(ω2) if x div y in

let newRes = replace ω by Grid.getNewRes(): (), ω if Grid.newResAvailable() in

let remRes = replace r:(), ω by ω if Grid.willBeRemoved(r) in

let migrate = replace r1: ⟨ω⟩, r2: ⟨⟩ by r1: ⟨⟩, r2: ⟨ω⟩ if Grid.willBeRemoved(r1) ∧ not Grid.willBeRemoved(r2) in

⟨R1:(findLocalReactives, 2, 3, 4, 5, 6, 7, 8, 9, 10), R2:⟩

runSieve, split, findDistReactives, newRes, remRes, migrate⟩
A possible execution: local reactions within R1

```plaintext
let findLocalReactives = replace x,y by x:y if x div y in

let runSieve = replace r:⟨x:y, ω⟩ by r:⟨x, ω⟩ in

let split = replace r1:⟨x1:y1, x2:y2, ω1, ω2⟩, r2:() by r1:⟨x1:y1, ω1⟩,
          r2:⟨findLocalReactives, x2:y2, ω2⟩ in

let findDistReactives = replace r1:⟨x, ω1⟩, r2:⟨y, ω2⟩
                       by r1:⟨x:y, ω1⟩, r2:⟨ω2⟩
                       if x div y in

let newRes = replace ω by Grid.getNewRes(): (), ω
             if Grid.newResAvailable() in

let remRes = replace r:⟨⟩, ω by ω
             if Grid.willBeRemoved(r) in

let migrate = replace r:⟨ω⟩, r2:⟨⟩ by r1:⟨⟩, r2:⟨ω⟩
              if Grid.willBeRemoved(r1) ^
              not Grid.willBeRemoved(r2) in

⟨R1:⟨findLocalReactives, 2, 3, 4, 5, 6, 7, 8, 9, 10⟩, R2:⟩
runSieve, split, findDistReactives,
newRes, remRes, migrate⟩
```
A possible execution: Split between R1 & R2

let findLocalReactives = replace x,y by x:y if x div y in

let runSieve = replace r:(x:y, ω) by r:(x, ω) in

let split = replace r1:(x1:y1, x2:y2, ω1 , ω2), r2:() by r1:(x1:y1, ω1),
     r2:(findLocalReactives, x2:y2, ω2) in

let findDistReactives = replace r1:(x, ω1), r2:(y, ω2) by r1:(x:y, ω1), r2:(ω2)
     if x div y in

let newRes = replace ω by Grid.getNewRes(): (), ω if Grid.newResAvailable() in
let remRes = replace r:(), ω by ω if Grid.willBeRemoved(r) in

let migrate = replace r1:⟨ω⟩, r2:⟨⟩ by r1:⟨⟩, r2:⟨ω⟩
     if Grid.willBeRemoved(r1) ^
     not Grid.willBeRemoved(r2) in

⟨R1:(findLocalReactives, 2, 3, 4, 5, 6, 7, 8, 9, 10), R2:⟩
runSieve, split, findDistReactives,
newRes, remRes, migrate⟩
A possible execution: Split between R1 & R2

```
let findLocalReactives = replace x,y by x:y if x div y in

let runSieve = replace r:⟨x:y, ω⟩ by r:⟨x, ω⟩ in

let split = replace r1:⟨x1:y1, x2:y2, ω1, ω2⟩, r2:⟨⟩ by r1:⟨x1:y1, ω1⟩,
            r2:⟨findLocalReactives, x2:y2, ω2⟩ in

let findDistReactives = replace r1:⟨x, ω1⟩, r2:⟨y, ω2⟩ by r1:⟨x:y, ω1⟩, r2:⟨ω2⟩
                        if x div y in

let newRes = replace ω by Grid.getNewRes(): ⟨⟩, ω
                         if Grid.newResAvailable() in

let remRes = replace r: ⟨⟩, ω by ω
               if Grid.willBeRemoved(r) in

let migrate = replace r1: ⟨ω⟩, r2:⟨⟩ by r1: ⟨⟩, r2:⟨ω⟩
                     if Grid.willBeRemoved(r1) ^
                       not Grid.willBeRemoved(r2) in

⟨R1:⟨findLocalReactives, 2, 3, 4, 5, 6, 7, 8, 9, 10⟩, R2:⟨⟩
  runSieve, split, findDistReactives,
  newRes, remRes, migrate⟩
```
A possible execution: Split between R1 & R2

```plaintext
let findLocalReactives = replace x,y by x:y if x div y in

let runSieve = replace r:(x:y, ω) by r:(x, ω) in

let split = replace r1:(x1:y1, x2:y2, ω1, ω2), r2:⟨⟩
by r1:(x1:y1, ω1),
   r2:(findLocalReactives, x2:y2, ω2) in

let findDistReactives = replace r1:(x, ω1), r2:(y, ω2)
by r1:(x:y, ω1), r2:(ω2)
if x div y in

let newRes = replace ω by Grid.getNewRes(): ⟨⟩, ω
if Grid.newResAvailable() in

let remRes = replace r:⟨⟩, ω by ω
if Grid.willBeRemoved(r) in

let migrate = replace r1 :(ω), r2 :⟨⟩ by r1 :⟨⟩, r2 :(ω)
if Grid.willBeRemoved(r1) ∧
   not Grid.willBeRemoved(r2) in

⟨R1:(findLocalReactives, 2, 3, 4, 5, 6, 7, 8, 9, 10),
   R2:⟨⟩
runSieve, split, findDistReactives,
   newRes, remRes, migrate⟩
```
A possible execution: R2 is inert

```
let findLocalReactives = replace x,y by x:y if x div y in

let runSieve = replace r:⟨x:y, ω⟩ by r:⟨x, ω⟩ in

let split = replace r1:⟨x1:y1, x2:y2, ω1 , ω2⟩, r2:⟨⟩ by r1:⟨x1:y1, ω1⟩,
               r2:⟨findLocalReactives, x2:y2, ω2⟩ in

let findDistReactives = replace r1:⟨x, ω1⟩, r2:⟨y, ω2⟩
                        by r1:⟨x:y, ω1 ⟩, r2:⟨ω2⟩
                        if x div y in

let newRes = replace ω by Grid.getNewRes(): ⟨⟩, ω
                        if Grid.newResAvailable() in

let remRes = replace r:⟨⟩, ω by ω
                        if Grid.willBeRemoved(r) in

let migrate = replace r1 : ⟨ω⟩, r2 : ⟨⟩ by r1 : ⟨⟩, r2 : ⟨ω⟩
                        if Grid.willBeRemoved(r1) \^ not Grid.willBeRemoved(r2) in

⟨R1: ⟨findLocalReactives, 2, 3, 4, 5, 6, 7, 8, 9, 10⟩, R2: ⟨⟩
runSieve, split, findDistReactives,
newRes, remRes, migrate⟩
```
A possible execution: R2 is inert

```plaintext
let findLocalReactives = replace x,y by x:y if x div y in

let runSieve = replace r:(x:y, ω) by r:(x, ω) in

let split = replace r1:(x1:y1, x2:y2, ω1 , ω2), r2:() by r1:(x1:y1, ω1),
   r2:(findLocalReactives, x2:y2, ω2) in

let findDistReactives = replace r1:(x, ω1), r2:(y, ω2) by r1:(x:y, ω1), r2:(ω2)
   if x div y in

let newRes = replace ω by Grid.getNewRes(): {}, ω if Grid.newResAvailable() in
let remRes = replace r:(), ω by ω if Grid.willBeRemoved(r) in

let migrate = replace r1:(ω), r2:() by r1:(), r2:(ω) if Grid.willBeRemoved(r1) ∧
   not Grid.willBeRemoved(r2) in

<R1:(findLocalReactives, 2, 3, 4, 5, 6, 7, 8, 9, 10), R2:()
runSieve, split, findDistReactives,
newRes, remRes, migrate>
A possible execution: R2 is inert

```
let findLocalReactives = replace x,y by x:y if x div y in

let runSieve = replace r:(x:y, ω) by r:(x, ω) in

let split = replace r1:(x1:y1, x2:y2, ω1, ω2), r2:() by r1:(x1:y1, ω1),
r2:(findLocalReactives, x2:y2, ω2) in

let findDistReactives = replace r1:(x, ω1), r2:(y, ω2) by r1:(x:y, ω1), r2:(ω2)
if x div y in

let newRes = replace ω by Grid.getNewRes(): {}, ω
if Grid.newResAvailable() in

let remRes = replace r:(), ω by ω
if Grid.willBeRemoved(r) in

let migrate = replace r1:⟨ω⟩, r2:⟨⟩ by r1:⟨⟩, r2:⟨ω⟩
if Grid.willBeRemoved(r1) ^
not Grid.willBeRemoved(r2) in

⟨R1:(findLocalReactives, 2, 3, 4, 5, 6, 7, 8, 9, 10), R2:⟩
runSieve, split, findDistReactives, newRes, remRes, migrate⟩
```
A possible execution: R3 is added

```
let findLocalReactives = replace x,y by x:y if x div y in

let runSieve = replace r:(x:y, ω) by r:(x, ω) in

let split = replace r1:(x1:y1, x2:y2, ω1 , ω2), r2:() by r1:(x1:y1, ω1),
           r2:(findLocalReactives, x2:y2, ω2) in

let findDistReactives = replace r1:(x, ω1), r2:(y, ω2)
                        by r1:(x:y, ω1), r2:(ω2)
                        if x div y in

let newRes = replace ω by Grid.getNewRes(): {}, ω
              if Grid.newResAvailable() in

let remRes = replace r: {}, ω by ω
            if Grid.willBeRemoved(r) in

let migrate = replace r1 : ⟨ω⟩, r2 : ⟨⟩ by r1 : ⟨⟩, r2 : ⟨ω⟩
              if Grid.willBeRemoved(r1) \n             not Grid.willBeRemoved(r2) in

⟨R1:(findLocalReactives, 2, 3, 4, 5, 6, 7, 8, 9, 10), R2:⟩
runSieve, split, findDistReactives, 
newRes, remRes, migrate⟩
```
A possible execution: R3 is added

```
let findLocalReactives = replace x,y by x:y if x div y in

let runSieve = replace r:⟨x:y, ω⟩ by r:⟨x, ω⟩ in

let split = replace r1:⟨x1:y1, x2:y2, ω1, ω2⟩, r2:⟨⟩ by
  r1:⟨x1:y1, ω1⟩,
  r2:⟨findLocalReactives, x2:y2, ω2⟩ in

let findDistReactives = replace r1:⟨x, ω1⟩, r2:⟨y, ω2⟩ by
  r1:⟨x:y, ω1⟩,
  r2:⟨ω2⟩ if x div y in

let newRes = replace ω by Grid.getNewRes(): ⟨⟩, ω if Grid.newResAvailable() in
let remRes = replace r:⟨⟩, ω by ω if Grid.willBeRemoved(r) in

let migrate = replace r1:⟨ω⟩, r2:⟨⟩ by
  r1:⟨⟩,
  r2:⟨ω⟩ if Grid.willBeRemoved(r1) ∧
  not Grid.willBeRemoved(r2) in

⟨R1:⟨findLocalReactives, 2, 3, 4, 5, 6, 7, 8, 9, 10⟩, R2:⟨⟩ runSieve, split, findDistReactives, newRes, remRes, migrate⟩
```
A possible execution: Split is activated

```
let findLocalReactives = replace x,y by x:y if x div y in

let runSieve = replace r:(x:y, ω) by r:(x, ω) in

let split = replace r1:(x1:y1, x2:y2, ω1, ω2), r2:() by r1:(x1:y1, ω1),
            r2:(findLocalReactives, x2:y2, ω2) in

let findDistReactives = replace r1:(x, ω1), r2:(y, ω2) by r1:(x:y, ω1), r2:(ω2)
                      if x div y in

let newRes = replace ω by Grid.getNewRes(): {}, ω
             if Grid.newResAvailable() in

let remRes = replace r:(), ω by ω
             if Grid.willBeRemoved(r) in

let migrate = replace r1:(ω), r2:() by r1:(), r2:(ω)
               if Grid.willBeRemoved(r1) ^
                  not Grid.willBeRemoved(r2) in

(R1:(findLocalReactives, 2, 3, 4, 5, 6, 7, 8, 9, 10), R2:()
runSieve, split, findDistReactives,
newRes, remRes, migrate)
```
A possible execution: Split is activated

```
let findLocalReactives = replace x, y by x:y if x div y in

let runSieve = replace r:⟨x:y, ω⟩ by r:⟨x, ω⟩ in

let split = replace r1:⟨x1:y1, x2:y2, ω1, ω2⟩, r2:⟨⟩
    by r1:⟨x1:y1, ω1⟩,
    r2:⟨findLocalReactives, x2:y2, ω2⟩ in

let findDistReactives = replace r1:⟨x, ω1⟩, r2:⟨y, ω2⟩
    by r1:⟨x:y, ω1⟩, r2:⟨ω2⟩
    if x div y in

let newRes = replace ω by Grid.getNewRes(): ⟨⟩, ω
    if Grid.newResAvailable() in

let remRes = replace r: ⟨⟩, ω by ω
    if Grid.willBeRemoved(r) in

let migrate = replace r1: ⟨ω⟩, r2: ⟨⟩ by r1: ⟨⟩, r2: ⟨ω⟩
    if Grid.willBeRemoved(r1) ^
    not Grid.willBeRemoved(r2) in

⟨R1:⟨findLocalReactives, 2, 3, 4, 5, 6, 7, 8, 9, 10⟩, R2:⟨⟩ runSieve, split, findDistReactives, newRes, remRes, migrate⟩
```
A possible execution: Split is activated

```plaintext
let findLocalReactives = replace x,y by x:y if x div y in

let runSieve = replace r:⟨x:y, ω⟩ by r:⟨x, ω⟩ in 

let split = replace r1:⟨x1:y1, x2:y2, ω1 , ω2⟩, r2:⟨⟩ by 
   r1:⟨x1:y1, ω1⟩, 
   r2:⟨findLocalReactives, x2:y2, ω2⟩ in

let findDistReactives = replace r1:⟨x, ω1⟩, r2:⟨y, ω2⟩ by 
   r1:⟨x:y, ω1⟩, r2:⟨ω2⟩ 
if x div y in

let newRes = replace ω by Grid.getNewRes(): ⟨⟩, ω 
  if Grid.newResAvailable() in

let remRes = replace r: ⟨⟩, ω by ω 
  if Grid.willBeRemoved(r) in

let migrate = replace r1 : ⟨ω⟩, r2 : ⟨⟩ by r1 : ⟨⟩, r2 :⟨ω⟩ 
  if Grid.willBeRemoved(r1) ∧ 
     not Grid.willBeRemoved(r2) in

⟨R1:⟨findLocalReactives, 2, 3, 4, 5, 6, 7, 8, 9, 10⟩, R2:⟨⟩ runSieve, split, findDistReactives, 
newRes, remRes, migrate⟩
```
A possible execution: R1 and R3 are inert

let findLocalReactives = replace x,y by x:y if x div y in

let runSieve = replace r:(x:y, ω) by r:(x, ω) in

let split = replace r1:(x1:y1, x2:y2, ω1, ω2), r2:() by r1:(x1:y1, ω1),
            r2:(findLocalReactives, x2:y2, ω2) in

let findDistReactives = replace r1:(x, ω1), r2:(y, ω2) by r1:(x:y, ω1), r2:(ω2)
                           if x div y in

let newRes = replace ω by Grid.getNewRes():(), ω if Grid.newResAvailable() in

let remRes = replace r:(), ω by ω if Grid.willBeRemoved(r) in

let migrate = replace r1:⟨ω⟩, r2:() by r1:⟨ω⟩, r2:⟨ω⟩ if Grid.willBeRemoved(r1) ^
               not Grid.willBeRemoved(r2) in

⟨R1:(findLocalReactives, 2, 3, 4, 5, 6, 7, 8, 9, 10), R2:⟩
runSieve, split, findDistReactives,
newRes, remRes, migrate⟩
A possible execution: R1 and R3 are inert

\[
\begin{aligned}
&\text{let } \text{findLocalReactives} = \text{replace } x,y \text{ by } x:y \text{ if } x \text{ div } y \text{ in} \\
&\text{let } \text{runSieve} = \text{replace } r:(x:y, \omega) \text{ by } r:(x, \omega) \text{ in} \\
&\text{let } \text{split} = \text{replace } r1:(x1:y1, x2:y2, \omega1, \omega2), r2:\emptyset \\
&\quad \text{by } r1:(x1:y1, \omega1), \\
&\quad r2:(\text{findLocalReactives}, x2:y2, \omega2) \text{ in} \\
&\text{let } \text{findDistReactives} = \text{replace } r1:(x, \omega1), r2:(y, \omega2) \\
&\quad \text{by } r1:(x:y, \omega1), r2:(\omega2) \\
&\quad \text{if } x \text{ div } y \text{ in} \\
&\text{let } \text{newRes} = \text{replace } \omega \text{ by } \text{Grid.getNewRes}(): \emptyset, \omega \\
&\quad \text{if } \text{Grid.newResAvailable}() \text{ in} \\
&\text{let } \text{remRes} = \text{replace } r:\emptyset, \omega \text{ by } \omega \\
&\quad \text{if } \text{Grid.willBeRemoved}(r) \text{ in} \\
&\text{let } \text{migrate} = \text{replace } r1:(), r2:() \text{ by } r1:(), r2:() \\
&\quad \text{if } \text{Grid.willBeRemoved}(r1) \land \\
&\quad \text{not } \text{Grid.willBeRemoved}(r2) \text{ in} \\
\langle \text{R1:} \langle \text{findLocalReactives}, 2, 3, 4, 5, 6, 7, 8, 9, 10 \rangle, \text{R2:} \emptyset, \text{runSieve}, \text{split}, \text{findDistReactives}, \\
&\text{newRes}, \text{remRes}, \text{migrate} \rangle
\end{aligned}
\]
A possible execution: R1, R2 & R3 inert

```plaintext
let findLocalReactives = replace x,y by x:y if x div y in 

let runSieve = replace r:(x:y, ω) by r:(x, ω) in 

let split = replace r1:(x1:y1, x2:y2, ω1 , ω2), r2:() 
by r1:(x1:y1, ω1), 
   r2:findLocalReactives, x2:y2, ω2) in 

let findDistReactives = replace r1:(x, ω1), r2:(y, ω2) 
by r1:(x:y, ω1 ), r2:(ω2) 
if x div y in 

let newRes = replace ω by Grid.getNewRes(): {}, ω 
   if Grid.newResAvailable() in 

let remRes = replace r: {}, ω by ω 
   if Grid.willBeRemoved(r) in 

let migrate = replace r1 : ⟨ω⟩, r2 :⟨ω⟩ by r1 : ⟨⟩, r2 :⟨ω⟩ 
   if Grid.willBeRemoved(r1) ^ 
      not Grid.willBeRemoved(r2) in 

⟨R1:⟨findLocalReactives, 2, 3, 4, 5, 6, 7, 8, 9, 10⟩, R2:⟩
runSieve, split, findDistReactives, 
newRes, remRes, migrate⟩
```
A possible execution: R1,R2 & R3 inert

let findLocalReactives = replace x,y by x:y if x div y in

let runSieve = replace r:(x:y, ω) by r:(x, ω) in

let split = replace r1:(x1:y1, x2:y2, ω1, ω2), r2:() by r1:(x1:y1, ω1),
            r2:(findLocalReactives, x2:y2, ω2) in

let findDistReactives = replace r1:(x, ω1), r2:(y, ω2) by r1:(x:y, ω1), r2:(ω2)
                        if x div y in

let newRes = replace ω by Grid.getNewRes():(), ω
            if Grid.newResAvailable() in

let remRes = replace r:(), ω by ω
            if Grid.willBeRemoved(r) in

let migrate = replace r1:(ω), r2:() by r1:(), r2:(ω)
                if Grid.willBeRemoved(r1) ^
                   not Grid.willBeRemoved(r2) in

⟨R1:(findLocalReactives, 2, 3, 4, 5, 6, 7, 8, 9, 10), R2:()
runSieve, split, findDistReactives,
newRes, remRes, migrate⟩
A possible execution: R1, R2 & R3 inert

\[
\begin{align*}
\text{let } & \text{findLocalReactives} = \text{replace } x, y \text{ by } x:y \text{ if } x \text{ div } y \text{ in} \\
\text{let } & \text{runSieve} = \text{replace } r:⟨x:y, ω⟩ \text{ by } r:⟨x, ω⟩ \text{ in} \\
\text{let } & \text{split} = \text{replace } r1:⟨x1:y1, x2:y2, ω1, ω2⟩, r2:⟨⟩ \text{ by } r1:⟨x1:y1, ω1⟩, \\
& \quad r2:⟨\text{findLocalReactives}, x2:y2, ω2⟩ \text{ in} \\
\text{let } & \text{findDistReactives} = \text{replace } r1:⟨x, ω1⟩, r2:⟨y, ω2⟩ \text{ by } r1:⟨x:y, ω1⟩, r2:⟨ω2⟩ \text{ if } x \text{ div } y \text{ in} \\
\text{let } & \text{newRes} = \text{replace } ω \text{ by } \text{Grid.getNewRes}(): ⟨⟩, ω \text{ if } \text{Grid.newResAvailable()} \text{ in} \\
\text{let } & \text{remRes} = \text{replace } r:⟨⟩, ω \text{ by } ω \text{ if } \text{Grid.willBeRemoved}(r) \text{ in} \\
\text{let } & \text{migrate} = \text{replace } r1:⟨⟨ω⟩⟩, r2:⟨⟩ \text{ by } r1:⟨⟩, r2:⟨ω⟩ \text{ if } \text{Grid.willBeRemoved}(r1) \wedge \\
& \quad \text{not } \text{Grid.willBeRemoved}(r2) \text{ in} \\
\langle R1:⟨\text{findLocalReactives}, 2, 3, 4, 5, 6, 7, 8, 9, 10⟩, R2:⟨⟩ \rangle \\
\text{runSieve, split, findDistReactives, newRes, remRes, migrate} \\
\end{align*}
\]
A possible execution: R2 inert, R1 & R3 active

```plaintext
let findLocalReactives = replace x,y by x:y if x div y in

let runSieve = replace r:⟨x:y, ω⟩ by r:⟨x, ω⟩ in

let split = replace r1:⟨x1:y1, x2:y2, ω1, ω2⟩, r2:⟨⟩
  by r1:⟨x1:y1, ω1⟩,
  r2:⟨findLocalReactives, x2:y2, ω2⟩ in

let findDistReactives = replace r1:⟨x, ω1⟩, r2:⟨y, ω2⟩
  by r1:⟨x:y, ω1⟩, r2:⟨ω2⟩
  if x div y in

let newRes = replace ω by Grid.getNewRes(): ⟨⟩, ω
  if Grid.newResAvailable() in

let remRes = replace r:⟨⟩, ω by ω
  if Grid.willBeRemoved(r) in

let migrate = replace r1:⟨ω⟩, r2:⟨⟩
  by r1:⟨⟩, r2:⟨ω⟩
  if Grid.willBeRemoved(r1) ^
  not Grid.willBeRemoved(r2) in

⟨R1:⟨findLocalReactives, 2, 3, 4, 5, 6, 7, 8, 9, 10⟩, R2:⟨⟩
runSieve, split, findDistReactives,
newRes, remRes, migrate⟩
```
A possible execution: R2 inert, R1 & R3 active

```plaintext
let findLocalReactives = replace x, y by x:y if x div y in

let runSieve = replace r:(x:y, ω) by r:(x, ω) in

let split = replace r1:(x1:y1, x2:y2, ω1, ω2), r2:() by r1:(x1:y1, ω1),
             r2:(findLocalReactives, x2:y2, ω2) in

let findDistReactives = replace r1:(x, ω1), r2:(y, ω2)
                           by r1:(x:y, ω1), r2:(ω2)
                           if x div y in

let newRes = replace ω by Grid.getNewRes(): (), ω
              if Grid.newResAvailable() in

let remRes = replace r:(), ω by ω
             if Grid.willBeRemoved(r) in

let migrate = replace r1:(ω), r2:() by r1:(), r2:⟨ω⟩
               if Grid.willBeRemoved(r1) ^
                  not Grid.willBeRemoved(r2) in

⟨R1:(findLocalReactives, 2, 3, 4, 5, 6, 7, 8, 9, 10), R2:⟩
runSieve, split, findDistReactives,
newRes, remRes, migrate⟩
```
A more complex example...

```python
import Pixel, Ray, LightRay, Contrib, Scene.
    \langle p_1, \ldots, p_n, \text{renderPixel} \rangle

renderPixel = \textbf{replace} \ p::\text{Pixel}
    \text{by} \ \langle \text{firstRay} (p), \text{renderRay}, \text{deleteRay},
    \text{enlighten}, \text{sumContrib} \rangle

renderRay = \textbf{replace} \ r::\text{Ray}
    \text{by} \ \text{Scene.intersectRays} (r)
    \text{if} \ r\text{.contribution} () > \epsilon

delteRay = \textbf{replace} \ r::\text{Ray}, \omega \text{by} \ \omega
    \text{if} \ r\text{.contribution} () \leq \epsilon

enlighten = \textbf{replace} \ l::\text{LightRay}
    \text{by} \ l\text{.computeContribution} ()

sumContrib = \textbf{replace} \ c1::\text{Contrib}, \ c2::\text{Contrib} \text{by} \ c1\text{.add} (c2)
```
A more complex example...

```python
import Pixel, Ray, LightRay, Contrib, Scene.

\langle p_1, \ldots, p_n, \text{renderPixel} \rangle

\text{renderPixel} = \text{replace } p::\text{Pixel}
\hspace{1cm} \text{by } \langle \text{firstRay} (p), \text{renderRay}, \text{deleteRay},
\hspace{1cm} \text{enlighten, } \text{sumContrib} \rangle

\text{renderRay} = \text{replace } r::\text{Ray}
\hspace{1cm} \text{by } \text{Scene.intersectRays} (r)
\hspace{1cm} \text{if } r.\text{contribution} () > \epsilon

\text{deleteRay} = \text{replace } r::\text{Ray}, \omega \text{ by } \omega \text{ if } r.\text{contribution} () \leq \epsilon

\text{enlighten} = \text{replace } l::\text{LightRay}
\hspace{1cm} \text{by } l.\text{computeContrib} ()

\text{sumContrib} = \text{replace } c1::\text{Contrib}, c2::\text{Contrib} \text{ by } c1.\text{add} (c2)
```
A more complex example...

```
import Pixel, Ray, LightRay, Contrib, Scene.

\langle p_1, \ldots, p_n, \text{renderPixel}\rangle

\text{renderPixel} = \textbf{replace} \ p::\text{Pixel}
  \quad \text{by} \ \langle \text{firstRay} \ (p), \text{renderRay}, \text{deleteRay},
  \quad \text{enlighten}, \text{sumContrib} \rangle

\text{renderRay} = \textbf{replace} \ r::\text{Ray}
  \quad \text{by} \ \text{Scene.intersectRays} \ (r)
  \quad \text{if} \ r.\text{contribution} () > \epsilon

\text{deleteRay} = \textbf{replace} \ r::\text{Ray}, \omega \ \text{by} \ \omega
  \quad \text{if} \ r.\text{contribution} () \leq \epsilon

\text{enlighten} = \textbf{replace} \ l::\text{LightRay}
  \quad \text{by} \ l.\text{computeContrib} ()

\text{sumContrib} = \textbf{replace} \ c1::\text{Contrib}, c2::\text{Contrib} \ \text{by} \ c1.\text{add} \ (c2)
```
A more complex example...

```python
import Pixel, Ray, LightRay, Contrib, Scene.

⟨p₁, . . . , pₙ, renderPixel⟩

renderPixel = replace p::Pixel
   by ⟨firstRay (p), renderRay, deleteRay, enlighten, sumContrib⟩

renderRay = replace r::Ray
   by Scene.intersectRays (r)
   if r.contribution () > ϵ

deleteray = replace r::Ray, ω by ω if r.contribution () ≤ ϵ

enlighten = replace l::LightRay
   by l.computeContribution ()

sumContrib = replace c1::Contrib, c2::Contrib by c1.add (c2 )
```
A more complex example...

```python
import Pixel, Ray, LightRay, Contrib, Scene.

\[
\langle p_1, \ldots, p_n, \text{renderPixel} \rangle
\]

\[
\text{renderPixel} = \text{replace } p::\text{Pixel}
\]
\[
\quad \text{by } \langle \text{firstRay } (p), \text{renderRay}, \text{deleteRay}, \text{enlighten}, \text{sumContrib} \rangle
\]

\[
\text{renderRay} = \text{replace } r::\text{Ray}
\]
\[
\quad \text{by } \text{Scene.intersectRays } (r)
\quad \text{if } r.\text{contribution } () > \epsilon
\]

\[
\text{deleteRay} = \text{replace } r::\text{Ray}, \omega \text{ by } \omega \text{ if } r.\text{contribution } () \leq \epsilon
\]

\[
\text{enlighten} = \text{replace } l::\text{LightRay}
\]
\[
\quad \text{by } l.\text{computeContrib } ()
\]

\[
\text{sumContrib} = \text{replace } c1::\text{Contrib}, c2::\text{Contrib by } c1.\text{add } (c2)
\]
```
A more complex example...

```plaintext
import Pixel, Ray, LightRay, Contrib, Scene.

\langle p_1, \ldots, p_n, \text{renderPixel} \rangle

\text{renderPixel} = \text{replace } p::\text{Pixel} \\
\phantom{\text{renderPixel} = } \text{by } \langle \text{firstRay} (p), \text{renderRay}, \text{deleteRay}, \\
\phantom{\text{renderPixel} = } \text{enlighten, sumContrib} \rangle

\text{renderRay} = \text{replace } r::\text{Ray} \\
\phantom{\text{renderRay} = } \text{by } \text{Scene.intersectRays}(r) \\
\phantom{\text{renderRay} = } \text{if } r.\text{contribution}() > \epsilon

\text{deleteRay} = \text{replace } r::\text{Ray}, \omega \text{ by } \omega \text{ if } r.\text{contribution}() \leq \epsilon

\text{enlighten} = \text{replace } l::\text{LightRay} \\
\phantom{\text{enlighten} = } \text{by } l.\text{computeContrib}()

\text{sumContrib} = \text{replace } c_1::\text{Contrib}, c_2::\text{Contrib} \text{ by } c_1.\text{add}(c_2)
```
A more complex example...

```python
import Pixel, Ray, LightRay, Contrib, Scene.
⟨p₁, . . . , pₙ, renderPixel⟩

renderPixel = replace p::Pixel
  by ⟨firstRay (p), renderRay, deleteRay, enlighten, sumContrib⟩

renderRay = replace r::Ray
  by Scene.intersectRays (r)
  if r.contribution () > ε

delteRay = replace r::Ray, ω by ω
  if r.contribution () ≤ ε

enlighten = replace l::LightRay
  by l.computeContrib()

sumContrib = replace c1::Contrib, c2::Contrib by c1.add (c2)
```

**Coordination rule**

```python
let split = replace res1:⟨⟨r::Ray, ωp⟩, ω1⟩, res2:⟨ω2⟩
  by res1:⟨ω1⟩, res2:⟨r, ωp, ω2⟩
  if Grid.load (res1) >> Grid.load(res2) ∧
  not Grid.willBeRemoved (res2)
```
Conclusion

- Chemical programming paradigm is well suited to design self-* systems
  - Implicit parallelism
  - Autonomic behavior
  - High-level abstraction
  - Self-modifying programs thanks to the high-order with a well defined semantics

- Several applications of the chemical programming paradigm
  - Workflow enactment (joint work with SZTAKI within CoreGRID)
  - Secure Grid systems using HOCL (joint work with STFC within CoreGRID)
  - Formal Semantics of GSML (joint work with ICT within EchoGRID)
  - Expressing Web Service coordination using HOCL (INRIA)

- Current state of the project
  - HOCL implementation is done (compiler/interpreter in Java)
  - Distributed implementation of the multiset is on-going
Perspectives

- This research generates several issues (challenges ???)
  - Distributed implementation of the multiset
    - P2P architecture + Distributed Shared Memory + Fault tolerance
  - Performance ?
    - All molecules can potentially react with all others !
    - “Simplicity cost performance”
    - “On ne peut pas avoir le beurre et l’argent du beurre”
  - Add topology inside the multiset
  - Expressing distribution thanks to a generic framework (map-reduce)
  - Change dynamically the rule syntax (runtime aspects)