40 ans
la révolution de l'information

INSTITUT NATIONAL
DE RECHERCHE
EN INFORMATIQUE
ET EN AUTOMATIQUE

INRIA

September 14 - 17, 2008
Highland Lake Inn
Flat Rock, North
Carolina, USA

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

40 ans la révolution de l'information

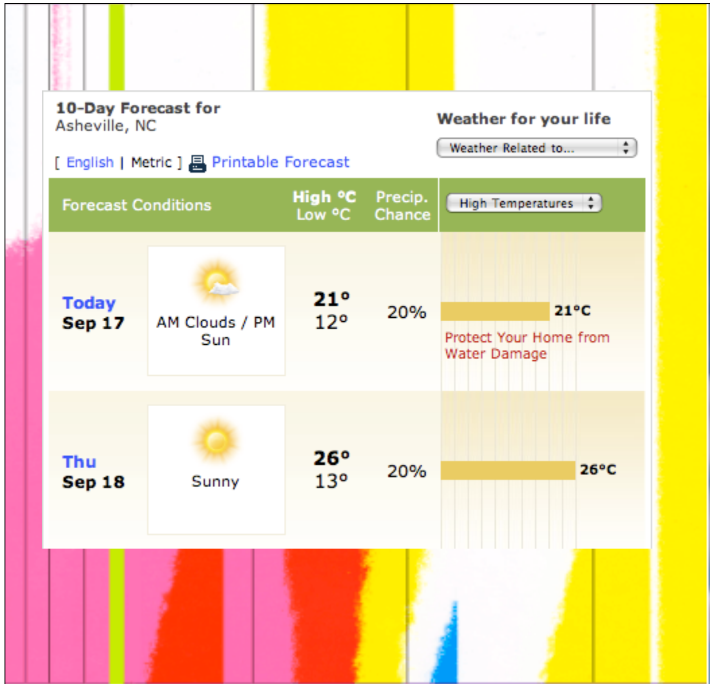
INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

INRIA

Clusters and Computational Grids for Scientific Computing

CCGSC 2008

September 14 - 17, 2008
Highland Lake Inn
Flat Rock, North
Carolina, USA



From Clouds to blue sky research

Unconventional Grid Programming

***Thierry Priol**, INRIA*

Joint work with Jean-Pierre Banâtre & Yann Radenac

40 ans la révolution de l'information

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

INRIA

Clusters and Computational Grids for Scientific Computing

CCGSC 2008

September 14 - 17, 2008
Highland Lake Inn
Flat Rock, North
Carolina, USA

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 Censored by the EU authorities *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 ?

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 :

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**

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

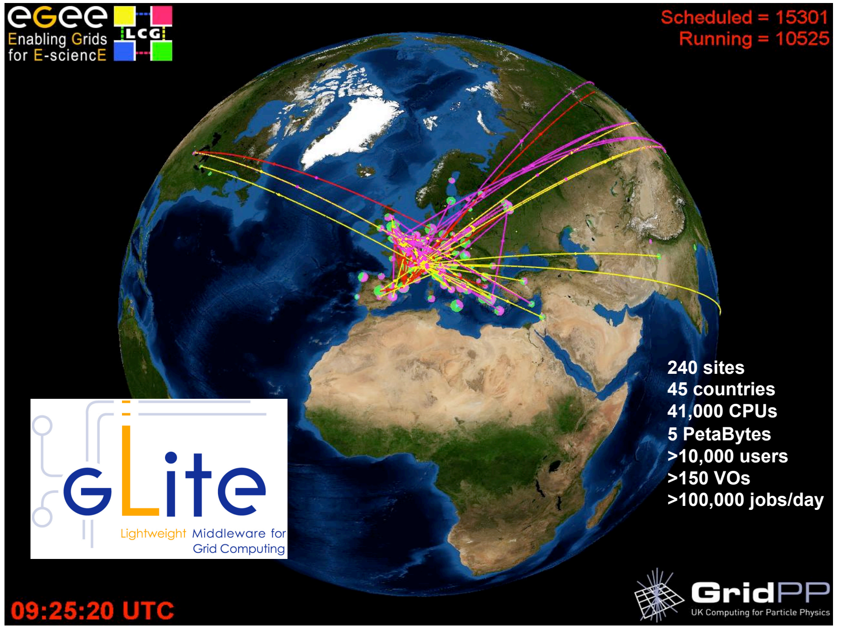
"Grid environments will require a rethinking of existing programming models and, most likely, new thinking about novel models more suitable for specific characteristics of grid applications and environments."

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% ?

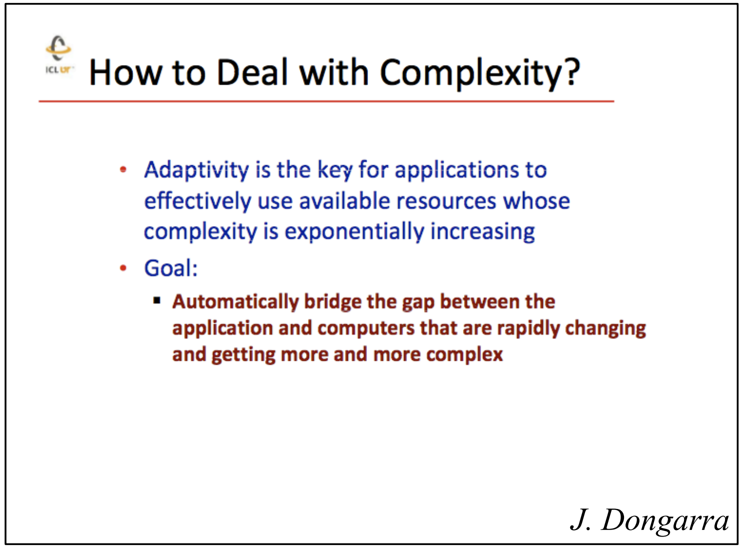


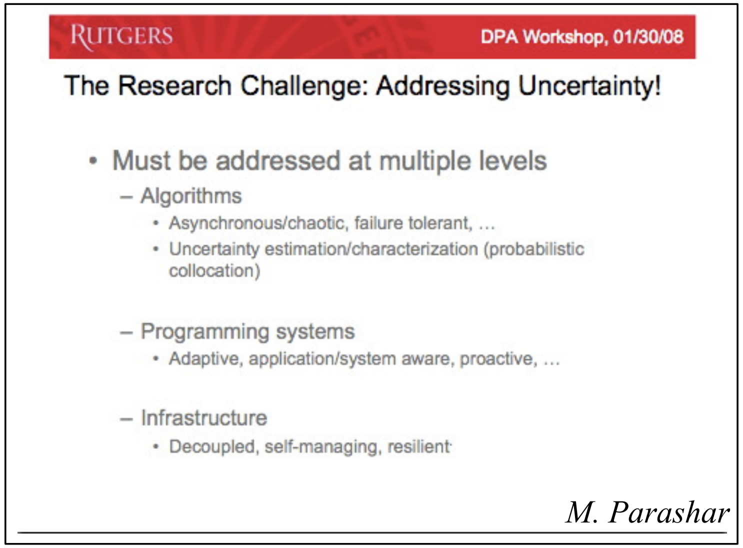
Grid infrastructures = uncertainty & complexity

- 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

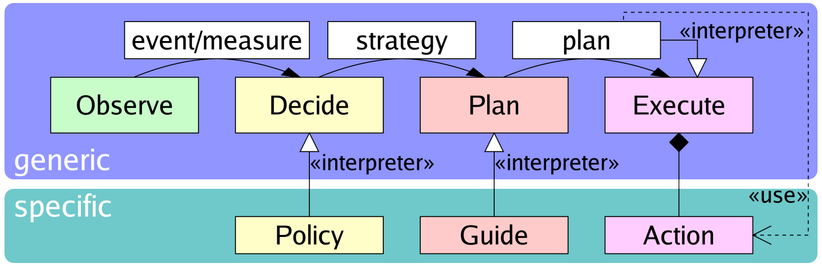


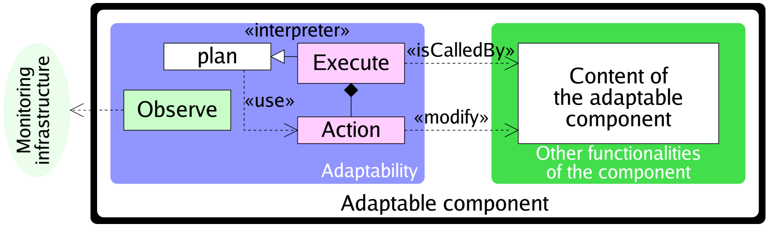


Current approaches to design adaptive and autonomic systems

Frameworks for adaptive / autonomic systems

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





Dynaco

But what about a programming model ?

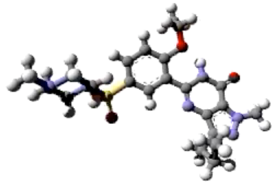
- Not only for applications but for Grid middleware as well
- Are there any available that would have autonomic/adaptive behaviors ?

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

Programming Objects	Chemistry
Data	Molecule
Multiset	Solution
Computation	Reaction

- Execution model using chemistry as a metaphor

Properties	Chemistry
Implicit parallelism Non determinism	Brownian motion

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

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

instead of

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

instead of

```
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;
                }
            }// inner for loop
            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

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

- Properties

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
```

- Properties
  - Intuitive model

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
```

- Properties
  - Intuitive model
  - Non-determinism

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
```

- 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

```
replace x,y by x if x 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

```
replace x,y by x if x 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

```
replace x,y by x 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”

Gamma: a chemical programming model

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

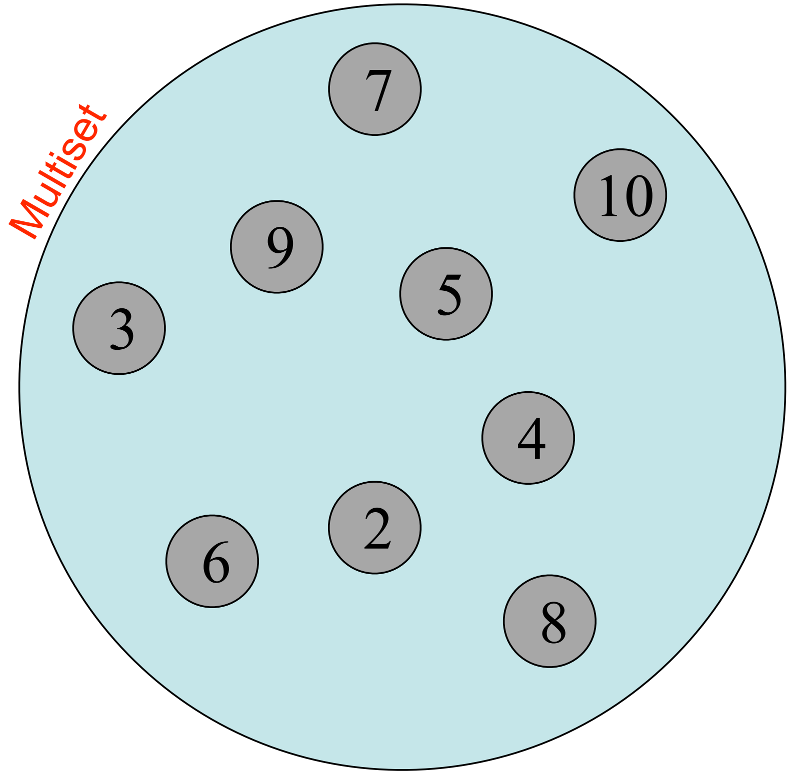
```
replace x,y by x if x div y
```

- 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”

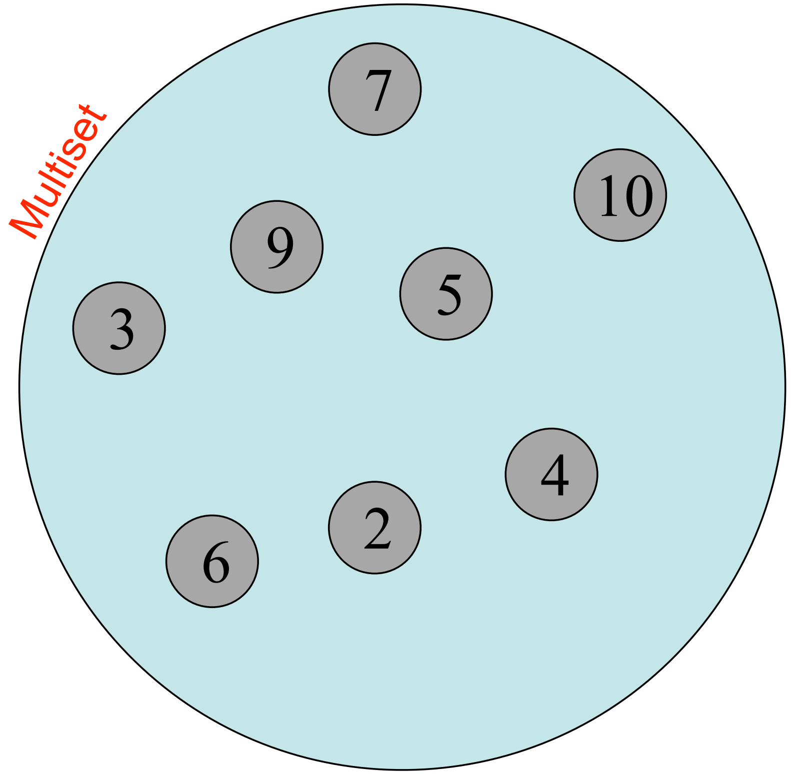
Chemical Programming Principle

- Example: computing prime numbers less than 10



Chemical Programming Principle

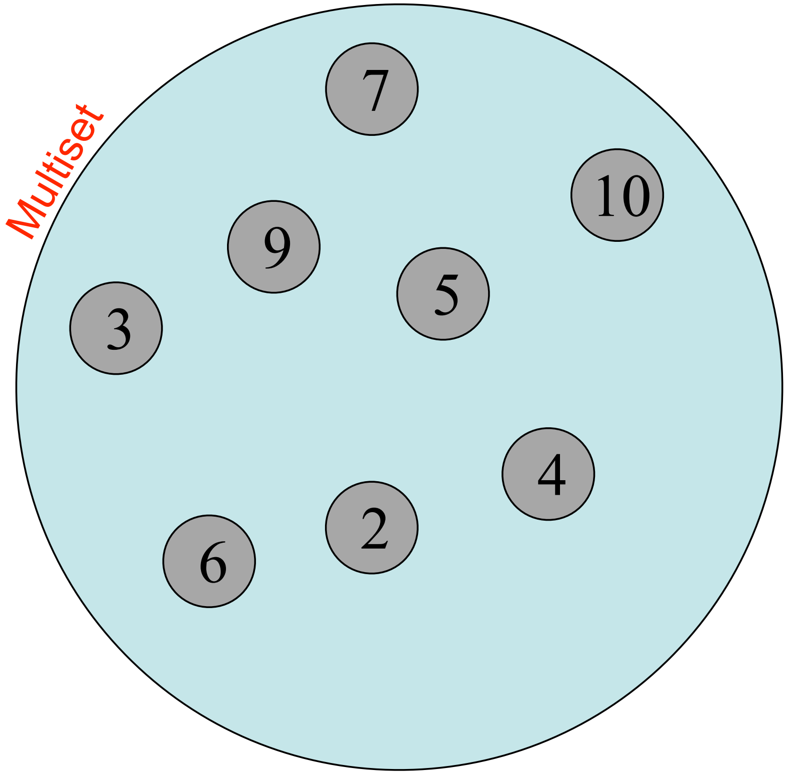
- Example: computing prime numbers less than 10



replace x,y **by** x **if** x div y

Chemical Programming Principle

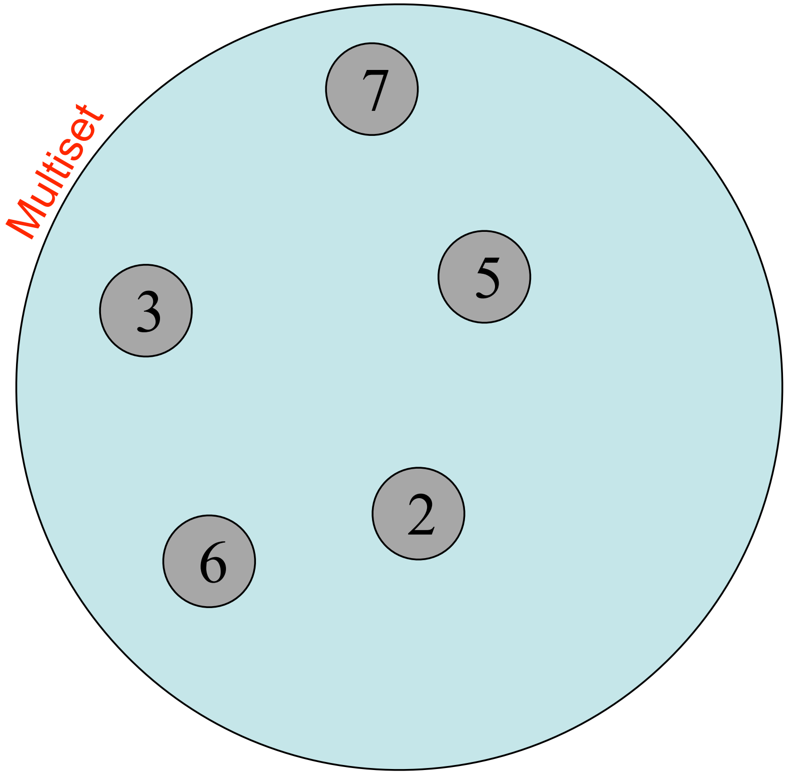
- Example: computing prime numbers less than 10




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

Chemical Programming Principle

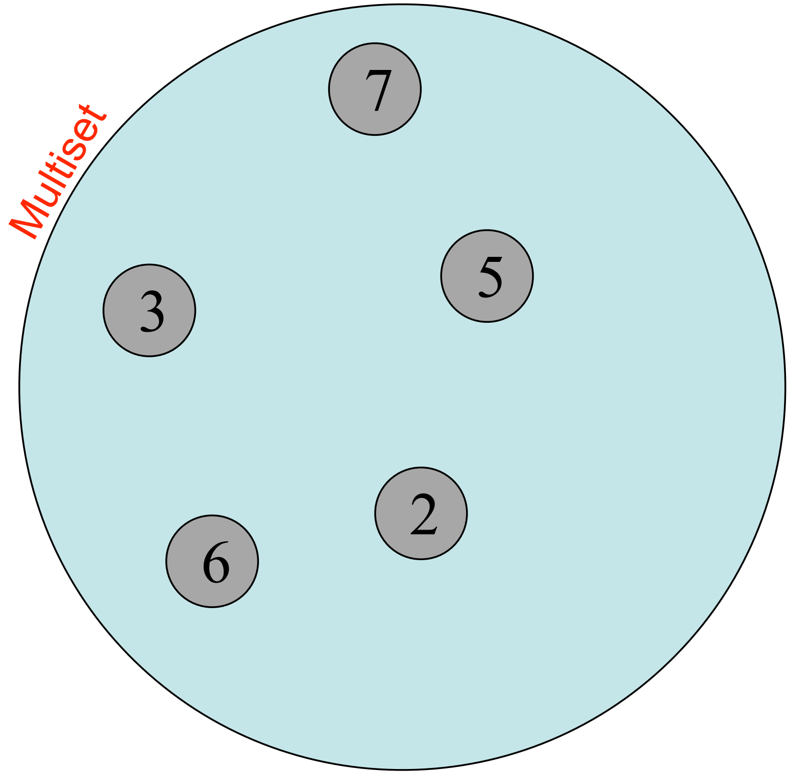
- Example: computing prime numbers less than 10




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

Chemical Programming Principle

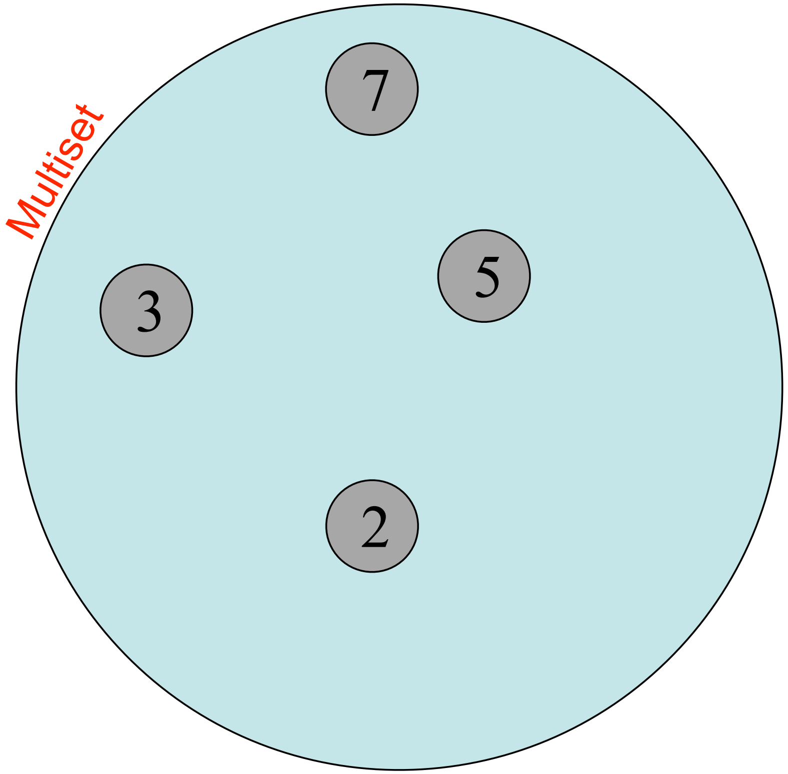
- Example: computing prime numbers less than 10



replace x,y **by** x **if** x 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

Stabilizing

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

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

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

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

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

Stabilizing

$$\downarrow^*$$

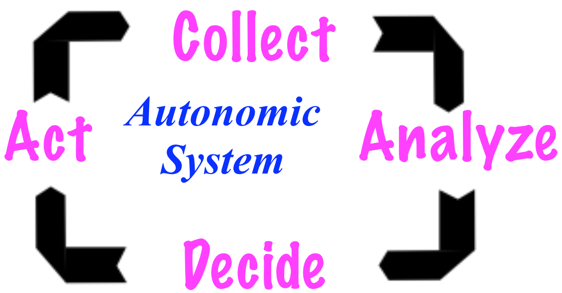
$$\langle \text{sieve}, 2, 3, 5, 7 \rangle$$

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

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

Stabilizing $\downarrow^*$

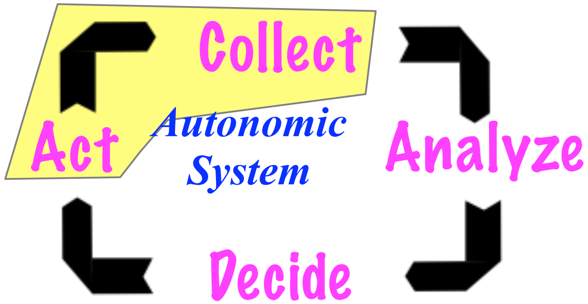
$$\langle \text{sieve}, 2, 3, 5, 7 \rangle$$

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

Re-stabilizing $\downarrow^*$

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

Multiset external I/O



Autonomic property

$\langle$sieve, 2, 3, 4, 5, 6, 7, 8, 9, 10$\rangle$

Stabilizing $\downarrow^*$

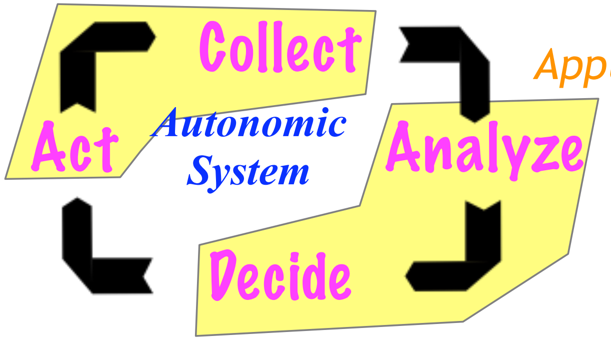
$\langle$sieve, 2, 3, 5, 7$\rangle$

Perturbation $\langle$sieve, 2, 3, 5, 7, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20$\rangle$

Re-stabilizing $\downarrow^*$

$\langle$sieve, 2, 3, 5, 7, 11, 13, 17, 19$\rangle$

Multiset external I/O



Applying rules

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 γ-calculus
- A HOCL program is a chemical solution of molecules $\langle M_1 , . . . , M_n \rangle$
  - 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^{∞}, X^{-1}

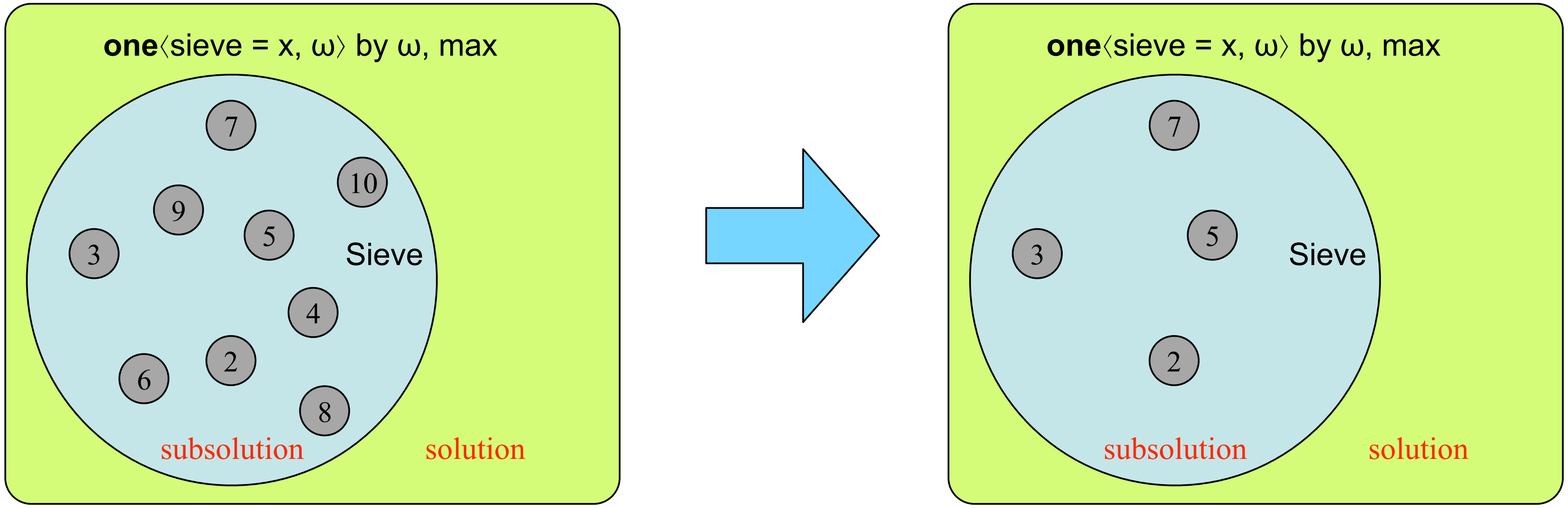
HOCL example : the high-order

The greatest prime number that is less than 10:

let sieve = **replace** x, y **by** x **if** x div y **in**

let max = **replace** x, y **by** x if x ≥ y **in**

⟨⟨sieve, 2, 3, 4, 5, 6, 7, 8, 9, 10⟩, **one**⟨sieve = x, ω⟩ by ω, max⟩



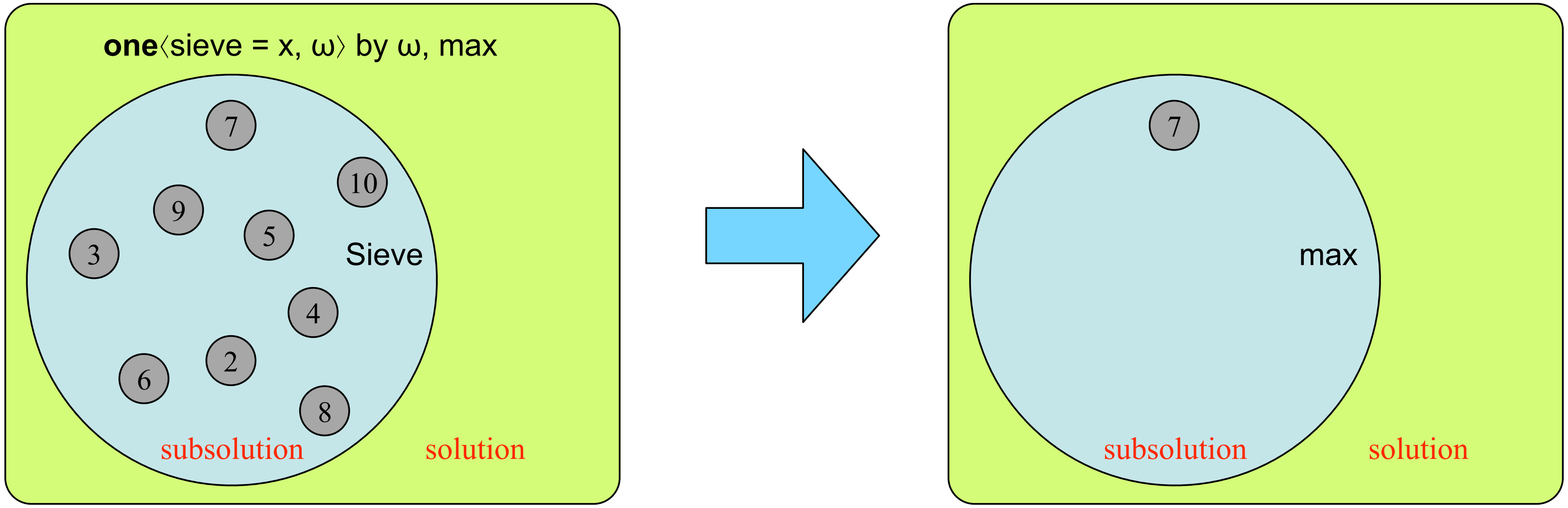
HOCL example : the high-order

The greatest prime number that is less than 10:

let sieve = **replace** x, y **by** x **if** x div y **in**

let max = **replace** x, y **by** x if x ≥ y **in**

⟨⟨sieve, 2, 3, 4, 5, 6, 7, 8, 9, 10⟩, **one**⟨sieve = x, ω⟩ by ω, max⟩



HOCL example : multiplicity

$$\frac{20}{9} * \frac{15}{8}$$

$\langle 2^2 , 5, 3^{-2} \rangle, \langle 3, 5, 2^{-3} \rangle$, one $\langle f \rangle, \langle g \rangle$ by $\langle f,g \rangle$

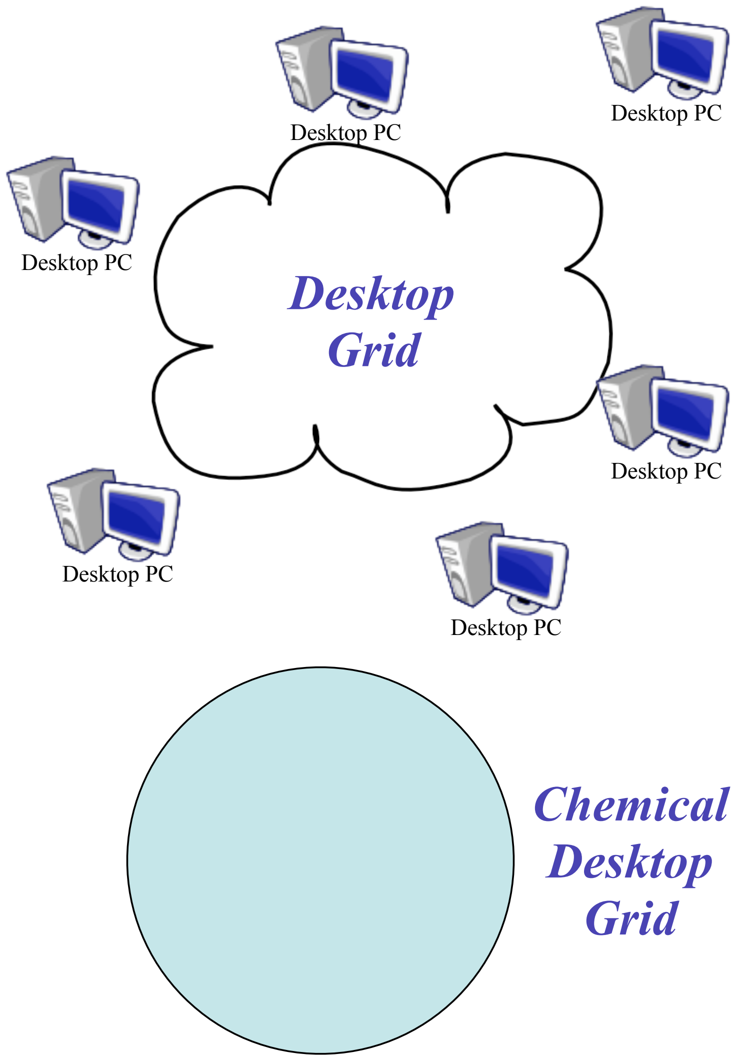
$$\downarrow^*$$

$\langle 5^2 , 3^{-1} , 2^{-1} \rangle$

Chemical Desktop Grid

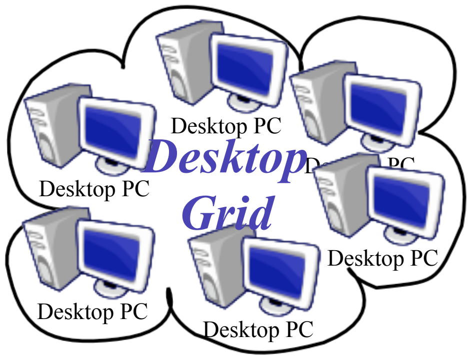
- Grid viewed as a chemical solution
  - Resources = solutions/molecules
  - Coordination = chemical reactions

- 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

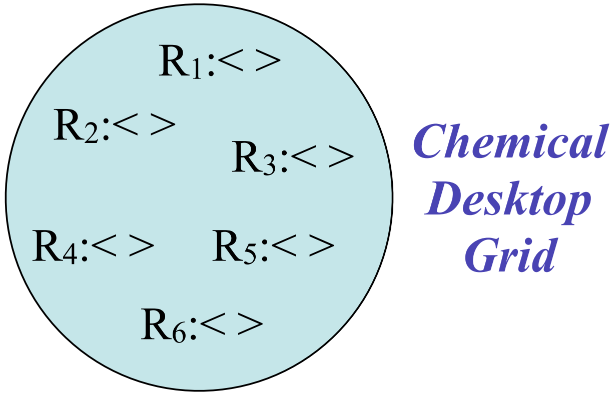


Chemical Desktop Grid

- 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⟩
```

Generate a pair of elements (x:y) that satisfies the condition x div y

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⟩
```

Replace a pair by its first element

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⟩
```

Split the computation when a new resource joins the Grid

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⟩
```

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 :⟨⟩ 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⟩
```

Adding and removing of resources

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⟩
```

Migrate elements from one resource to another one

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⟩
```

Initial solution

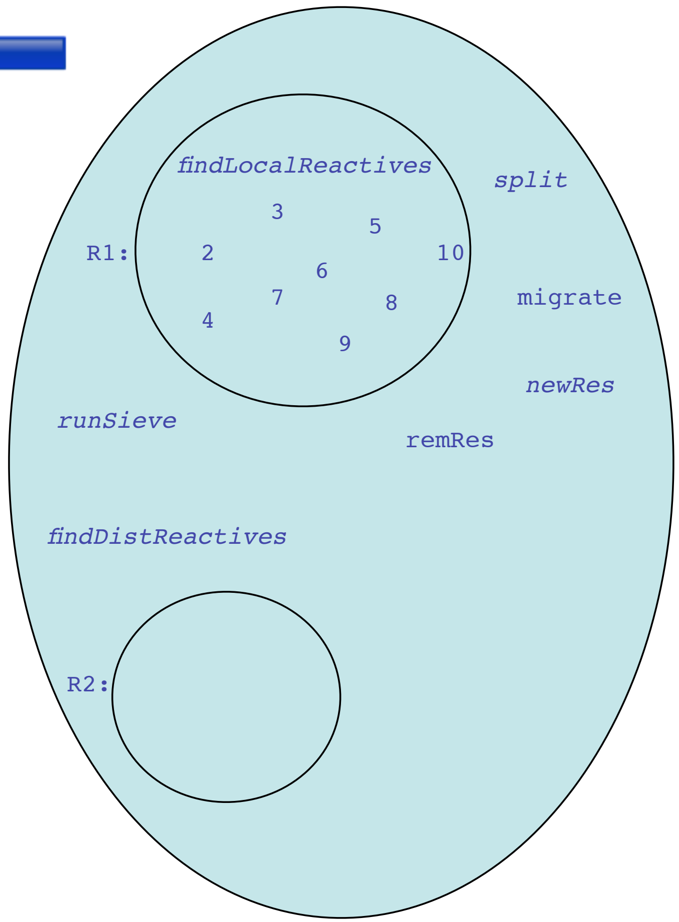
Computing prime numbers for a Chemical Desktop Grid

```
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⟩
```

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⟩
```

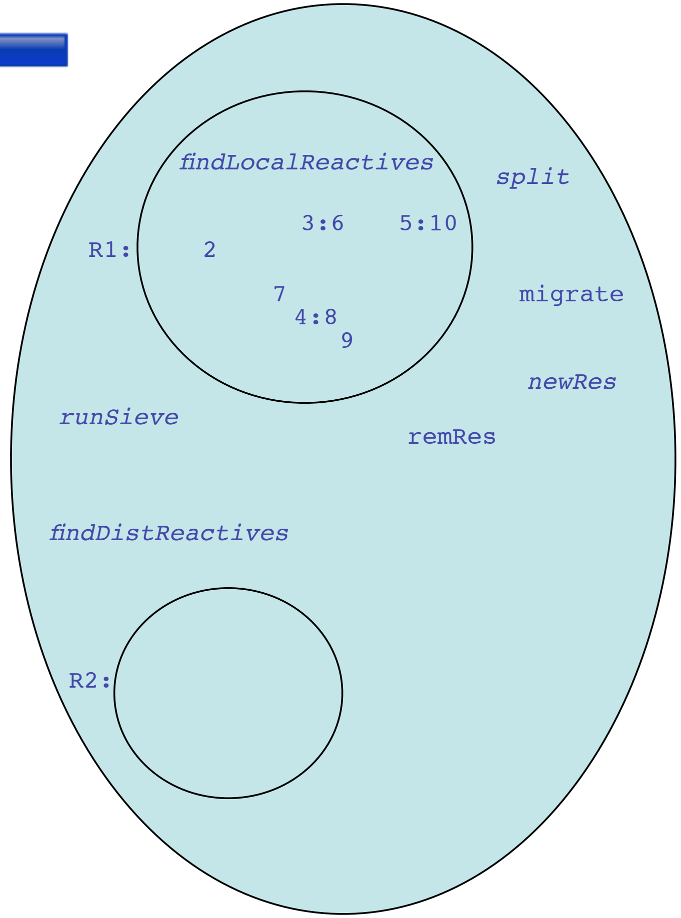
Solution



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⟩
```

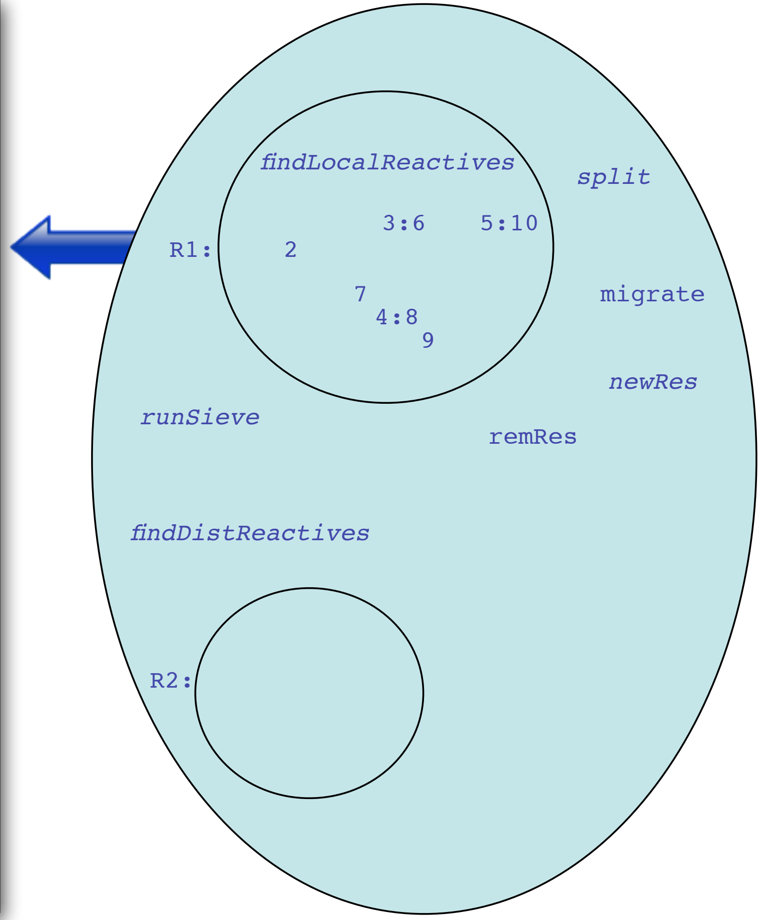
Solution



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⟩
```

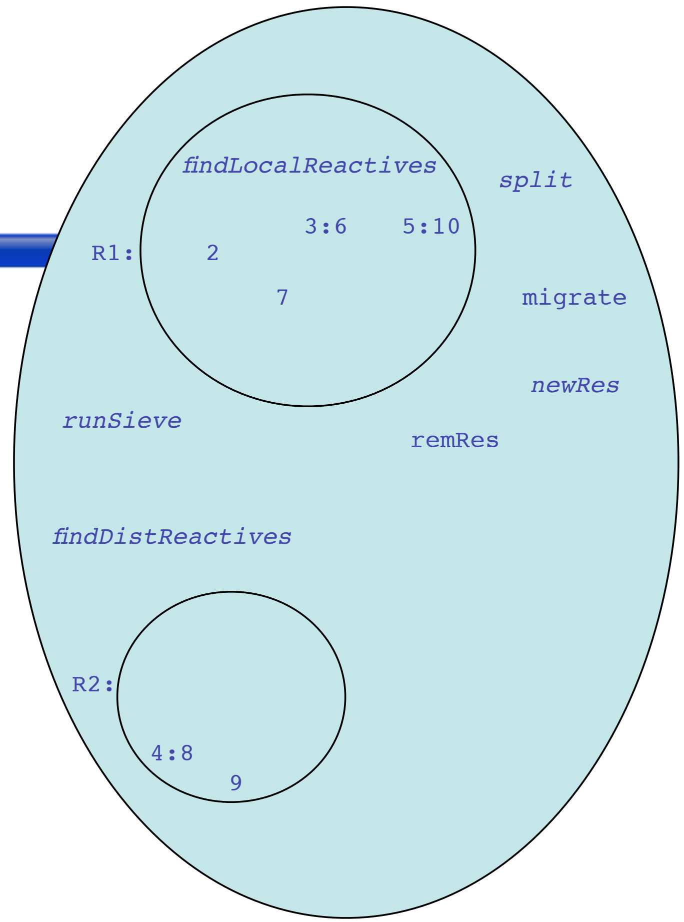
Solution



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⟩
```

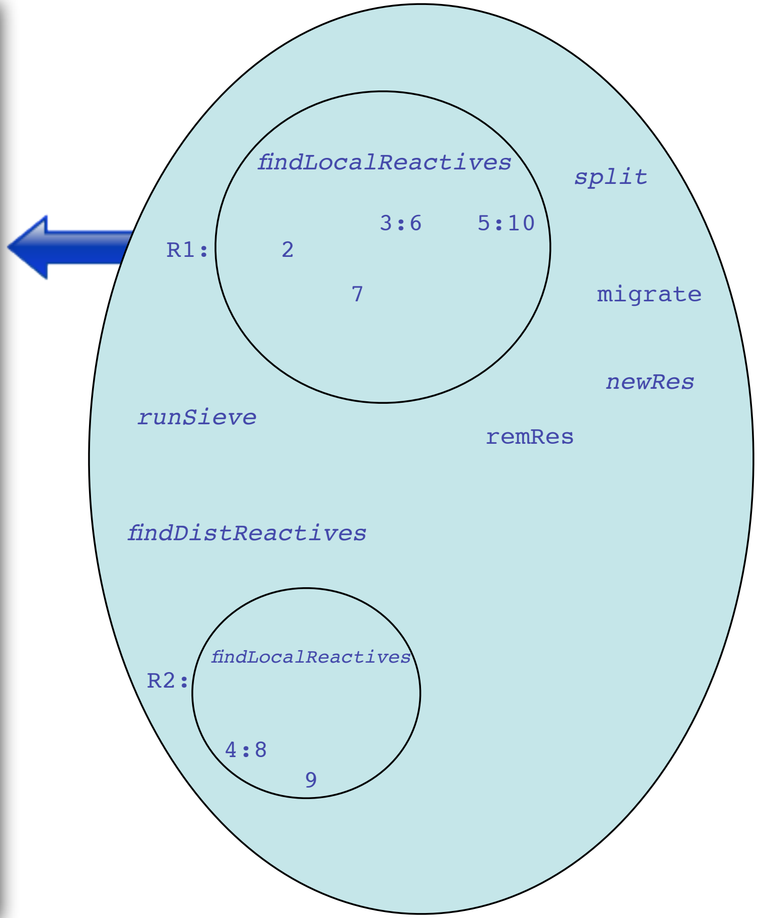
Solution



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⟩
```

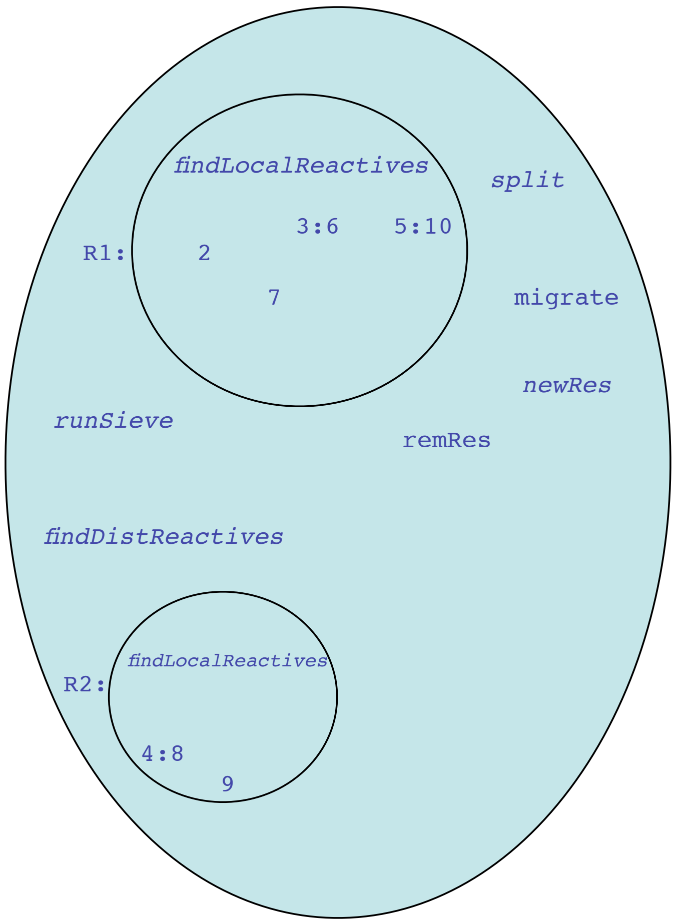
Solution



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⟩
```

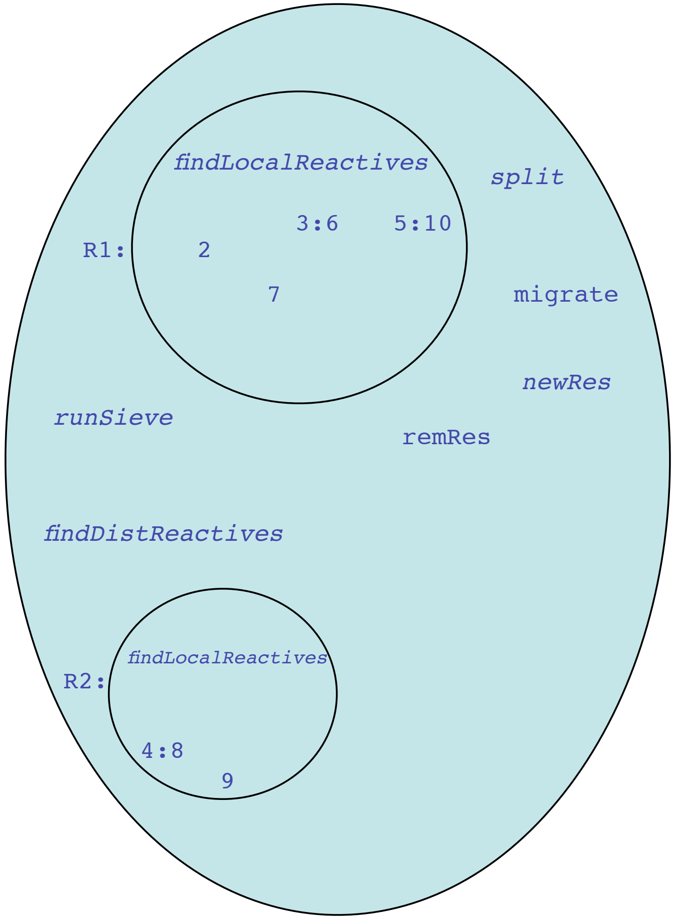
Solution



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⟩
```

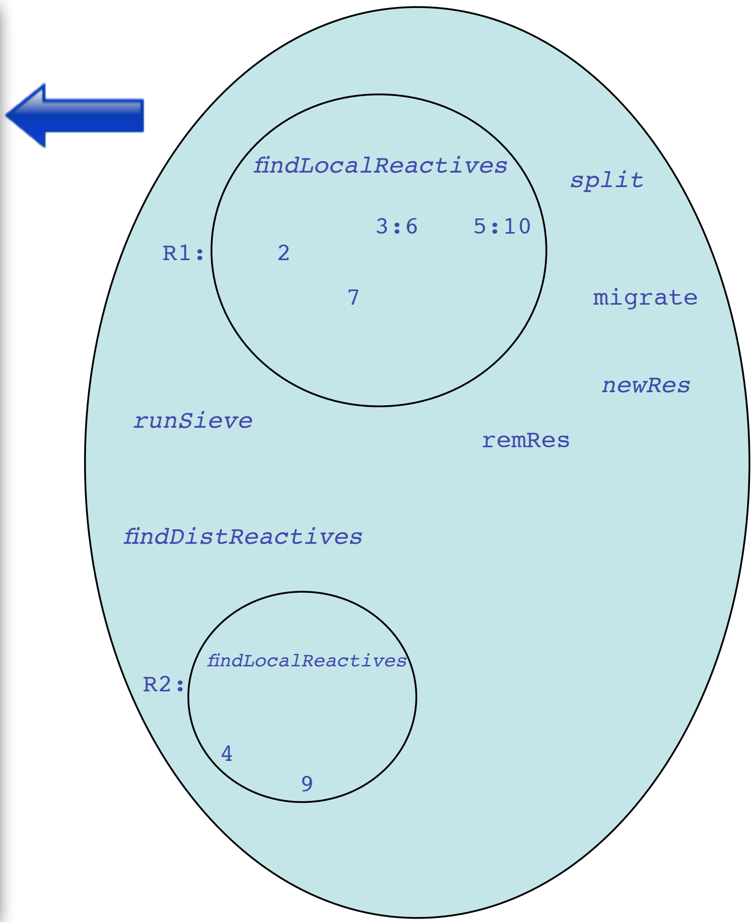
Solution



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⟩
```

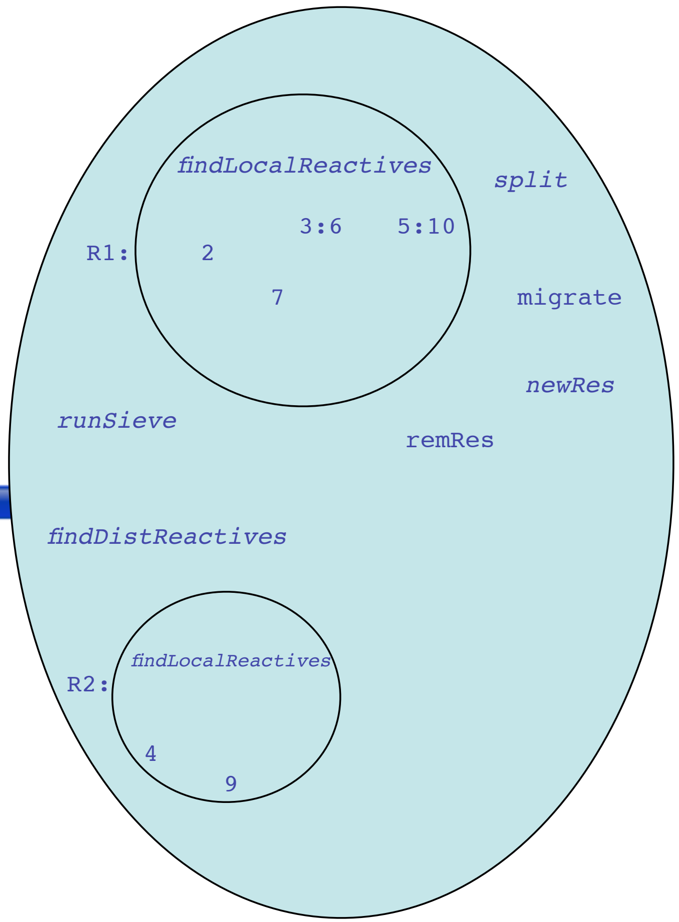
Solution



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⟩
```

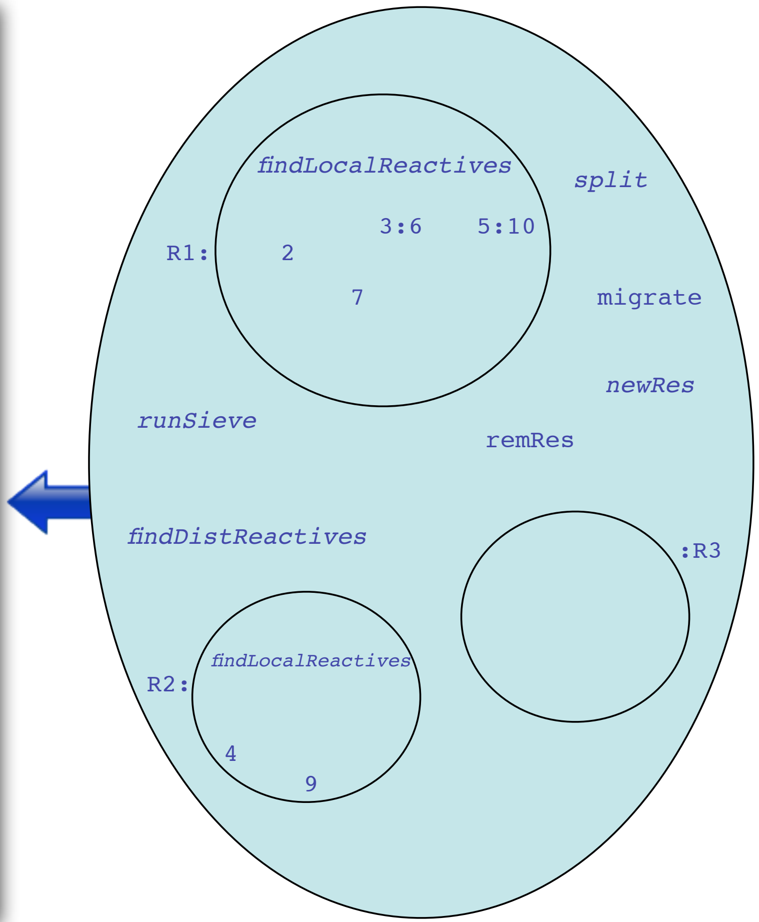
Solution



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⟩
```

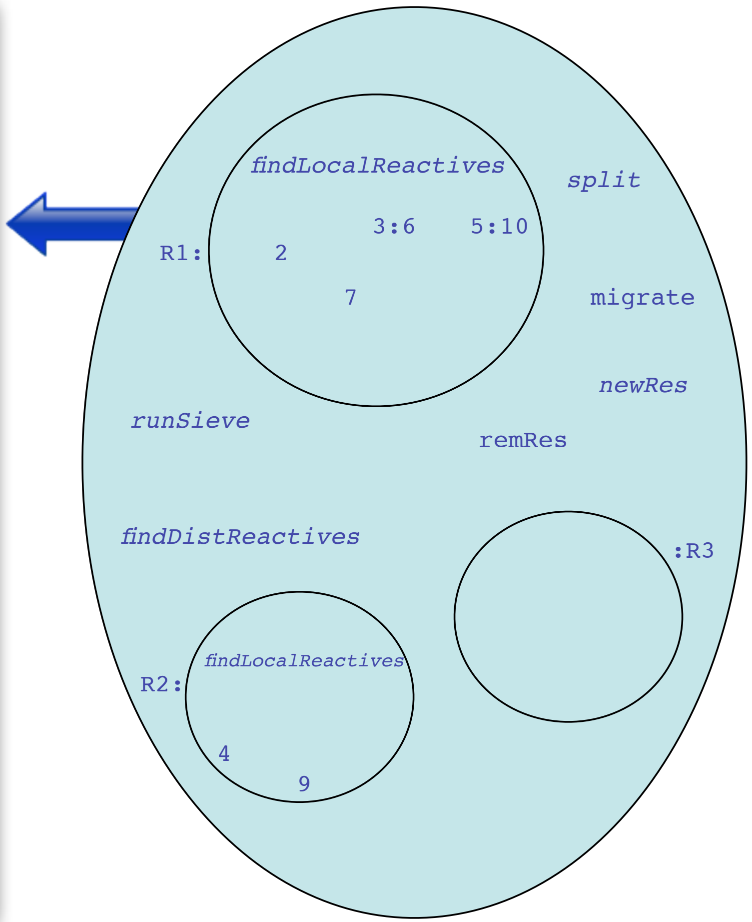
Solution



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⟩
```

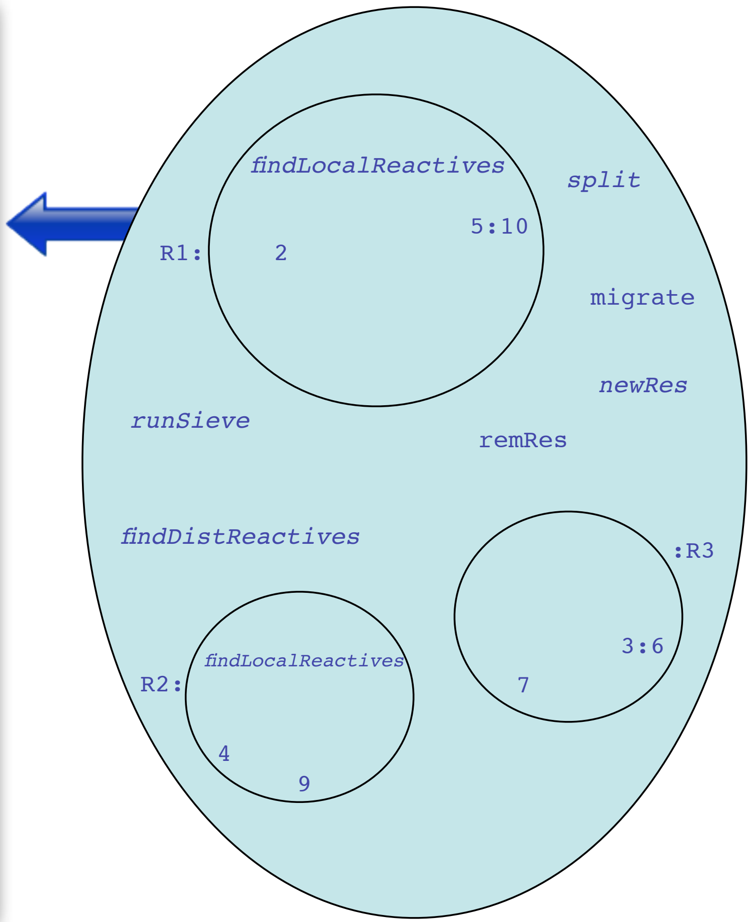
Solution



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⟩
```

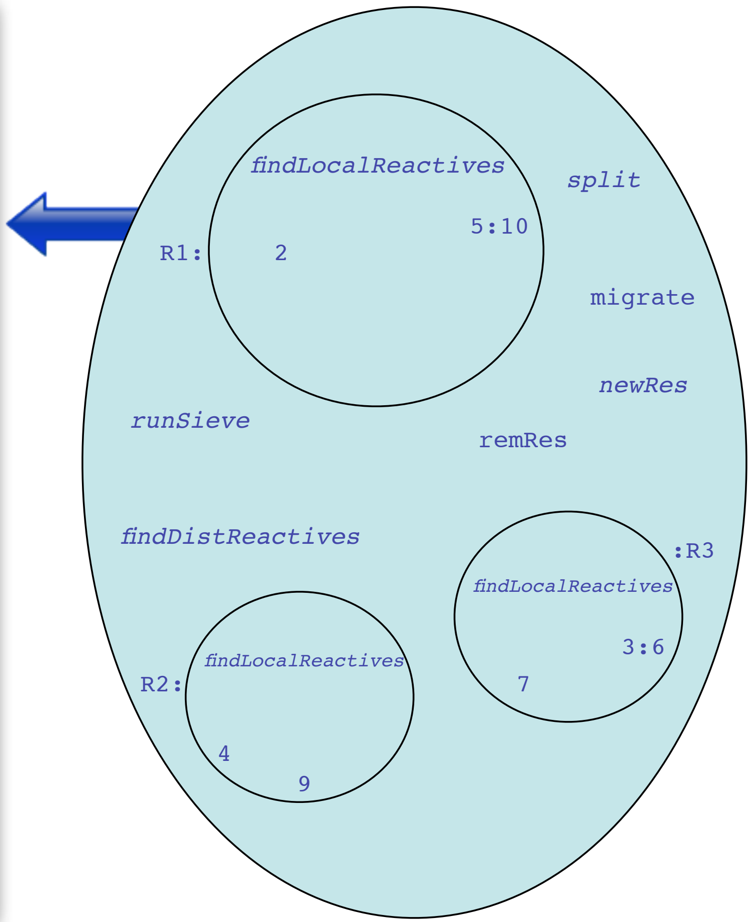
Solution



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⟩
```

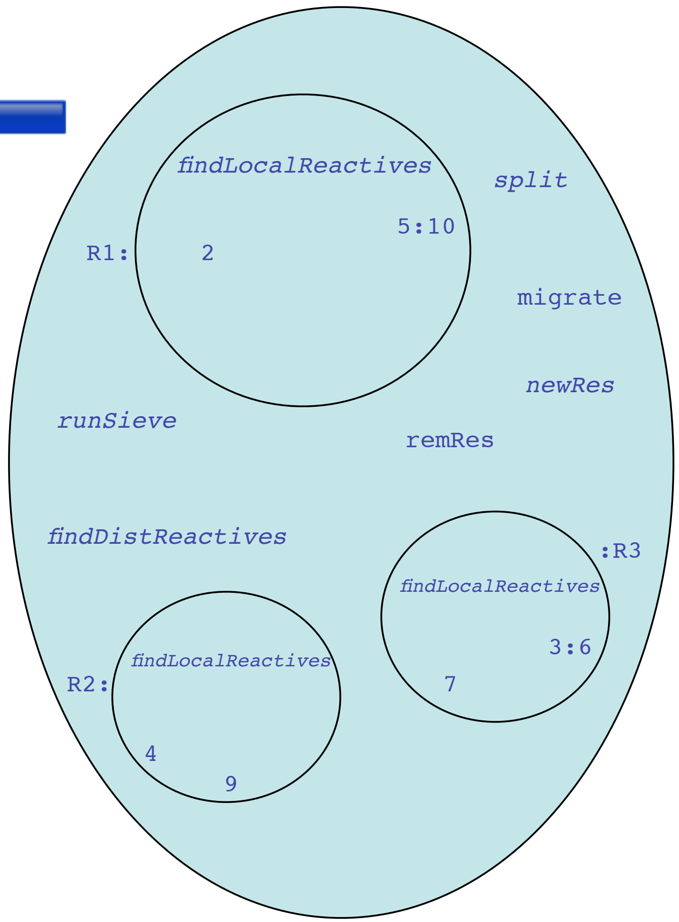
Solution



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 : ⟨⟩, 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⟩
```

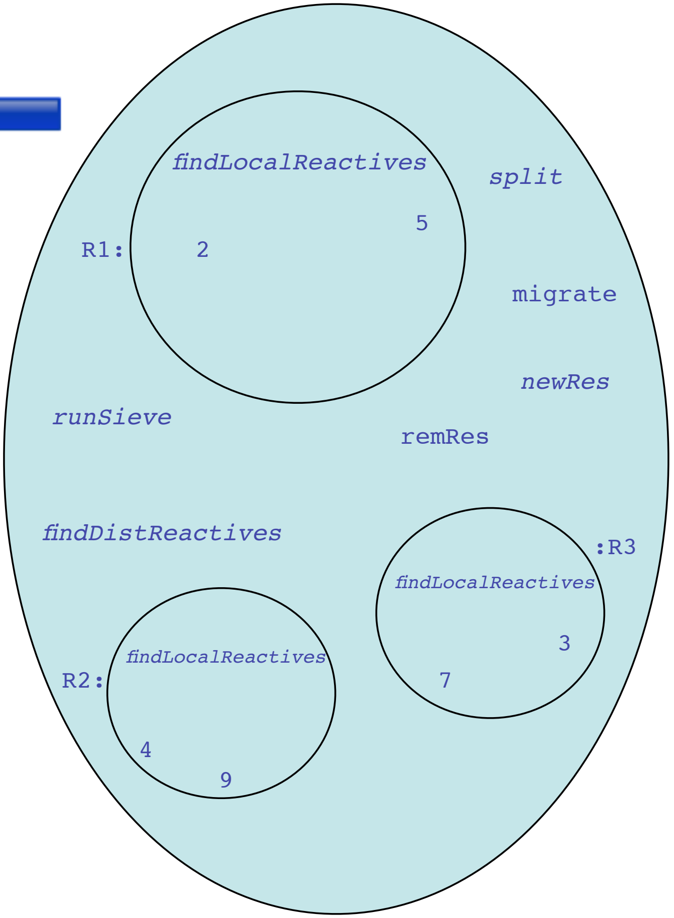
Solution



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 : ⟨⟩, 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⟩
```

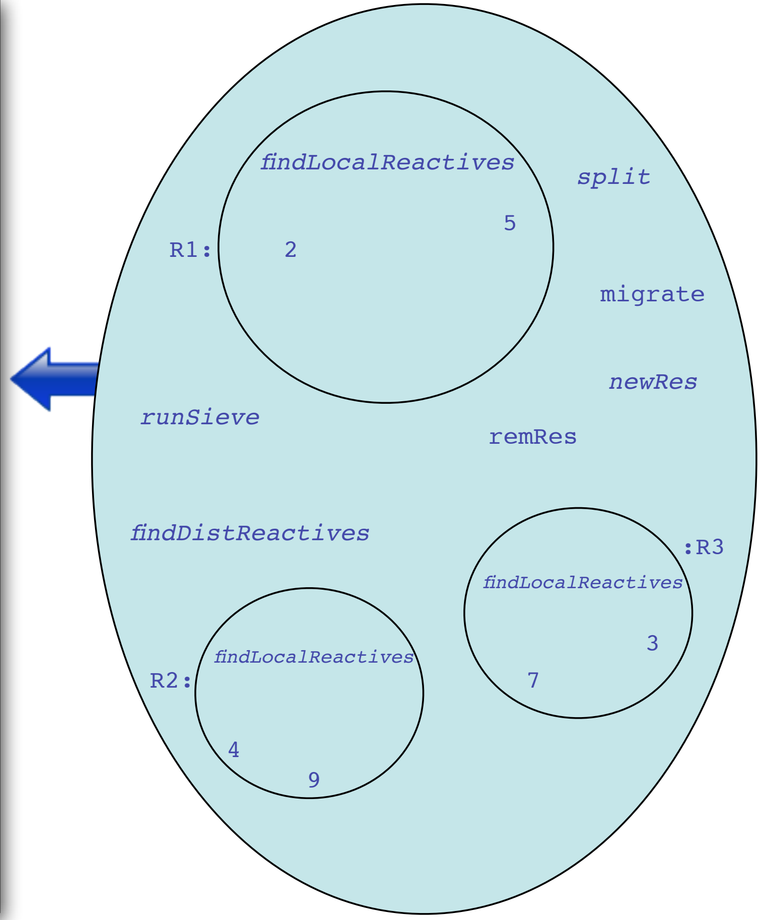
Solution



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⟩
```

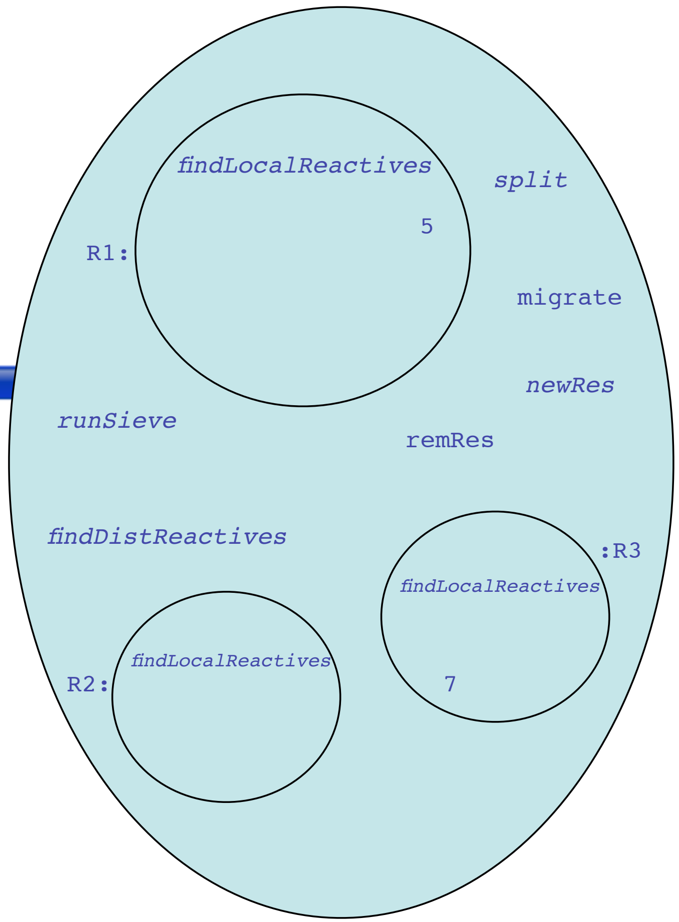
Solution



A possible execution: R1,R2 & R3 inert

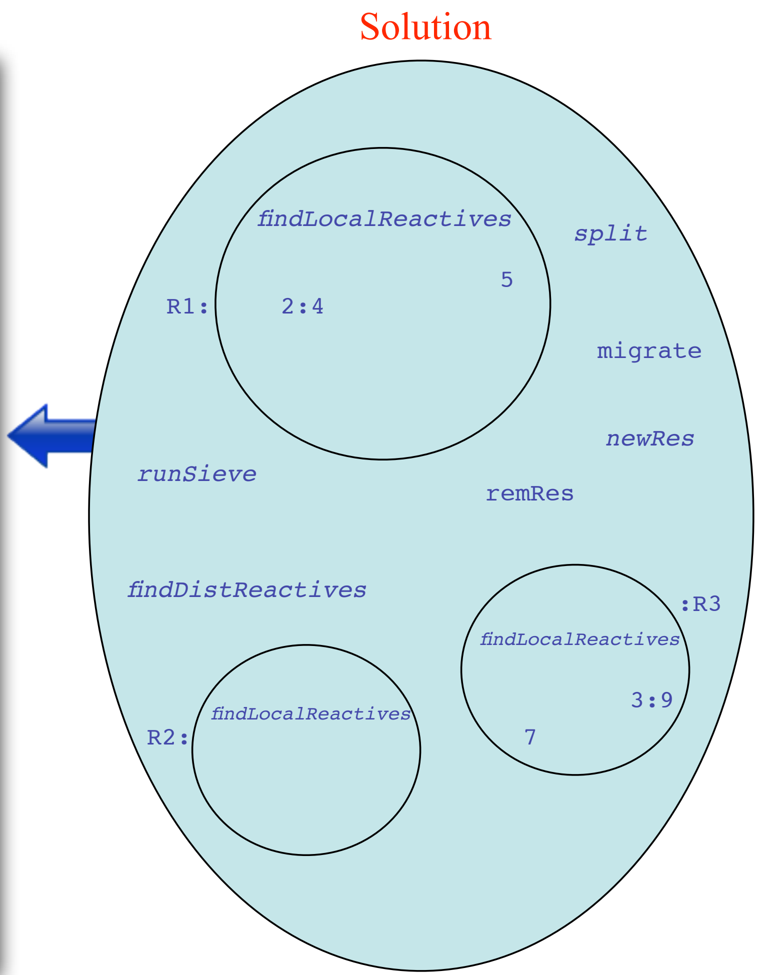
```
let findLocalReactives = replace x,y by x:y if x div y in

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⟩
```

Solution



A possible execution: R1,R2 & R3 inert

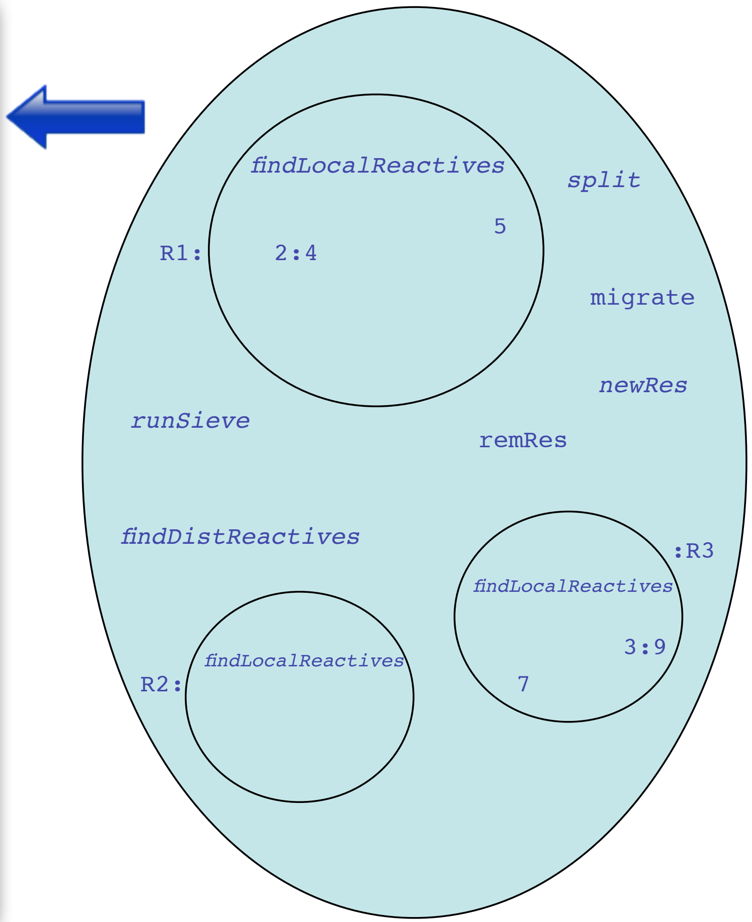
```
let findLocalReactives = replace x,y by x:y if x div y in

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: R2 inert, R1 & R3 active

```
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⟩
```

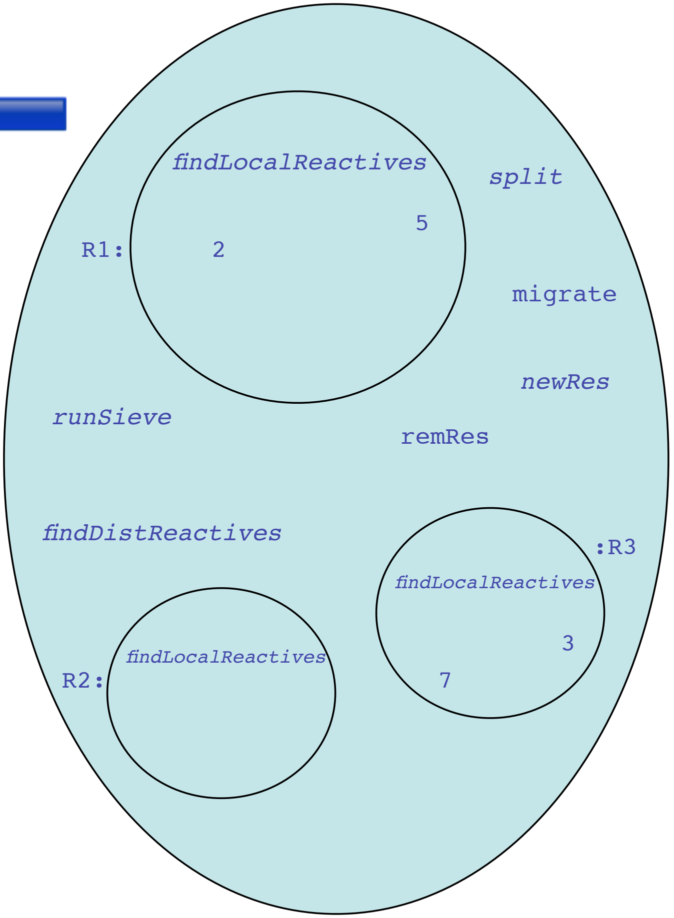
Solution



A possible execution: R2 inert, R1 & R3 active

```
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⟩
```

Solution



A more complex example...

import Pixel, Ray, LightRay, Contrib, Scene.

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

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

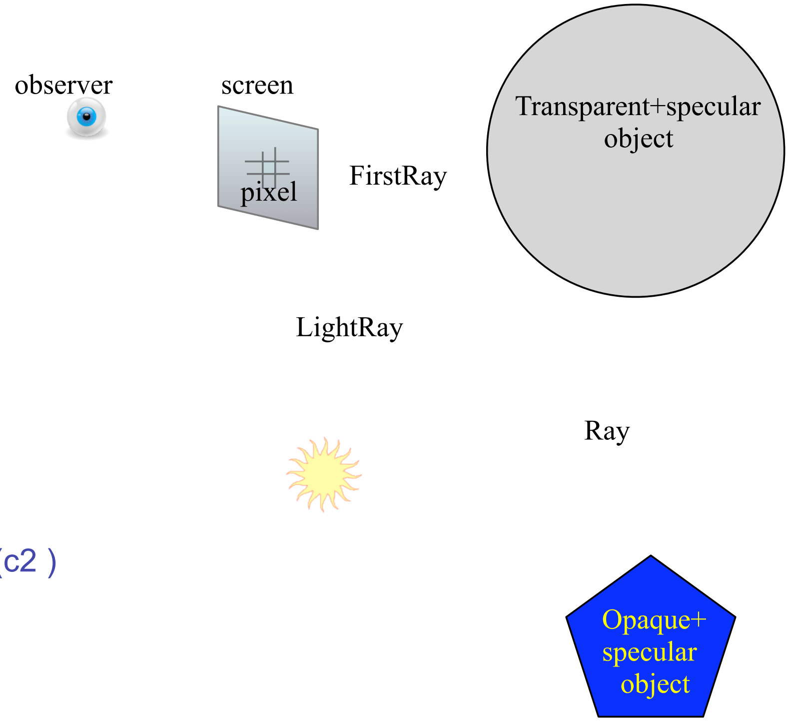
renderRay = **replace** r::Ray
by Scene.intersectRays (r)
if r.contribution () > ϵ

deleteRay = **replace** r::Ray, ω by ω **if** r.contribution () $\leq \epsilon$

enlighten = **replace** l::LightRay

by l.computeContrib ()

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



A more complex example...

import Pixel, Ray, LightRay, Contrib, Scene.

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

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

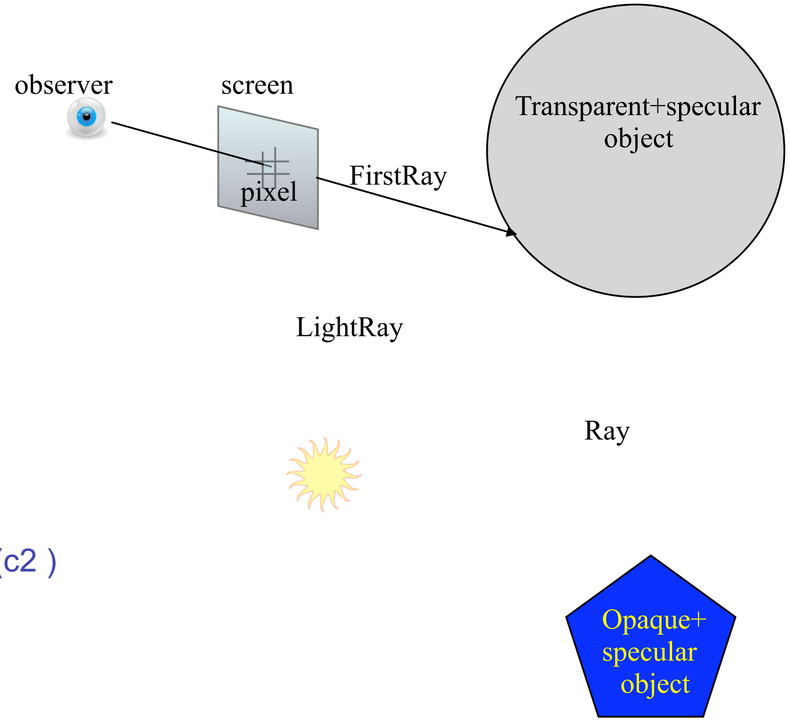
renderRay = **replace** r::Ray
by Scene.intersectRays (r)
if r.contribution () > ϵ

deleteRay = **replace** r::Ray, ω by ω **if** r.contribution () $\leq \epsilon$

enlighten = **replace** l::LightRay

by l.computeContrib ()

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



A more complex example...

import Pixel, Ray, LightRay, Contrib, Scene.

⟨p_1 , . . . , p_n , renderPixel⟩

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

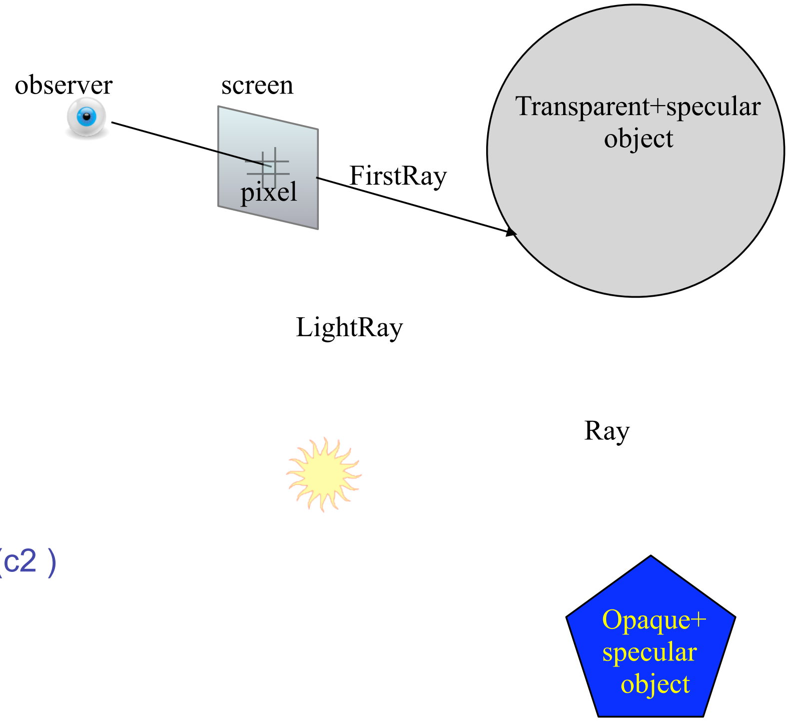
renderRay = **replace** r::Ray
by Scene.intersectRays (r)
if r.contribution () > ϵ

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

enlighten = **replace** l::LightRay

by l.computeContrib ()

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



A more complex example...

import Pixel, Ray, LightRay, Contrib, Scene.

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

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

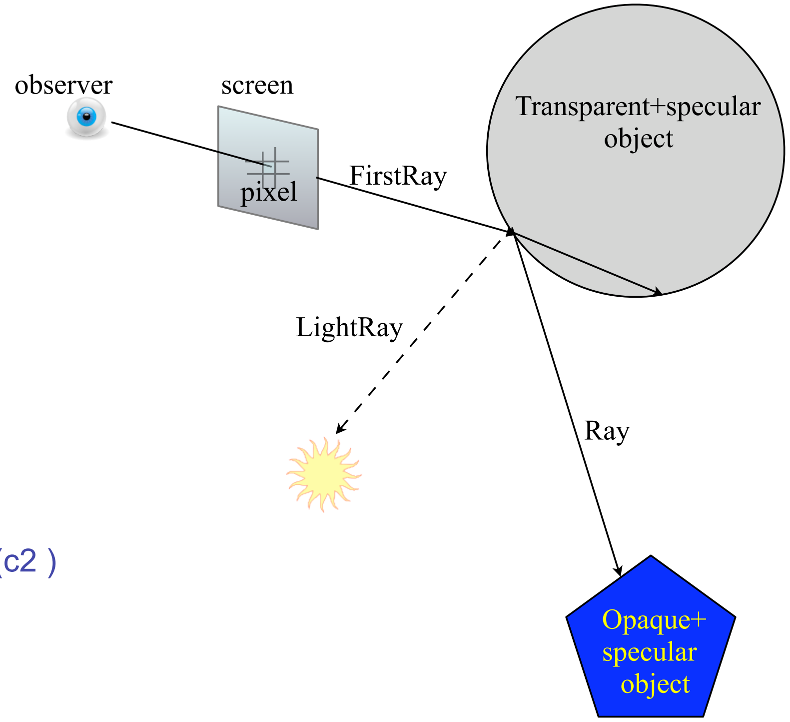
renderRay = **replace** r::Ray
by Scene.intersectRays (r)
if r.contribution () > ϵ

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

enlighten = **replace** l::LightRay

by l.computeContrib ()

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



A more complex example...

import Pixel, Ray, LightRay, Contrib, Scene.

$\langle p_1 , \ldots , p_n$, renderPixel$\rangle$

renderPixel = **replace** p::Pixel
by $\langle$firstRay (p), renderRay, deleteRay, enlighten, sumContrib$\rangle$

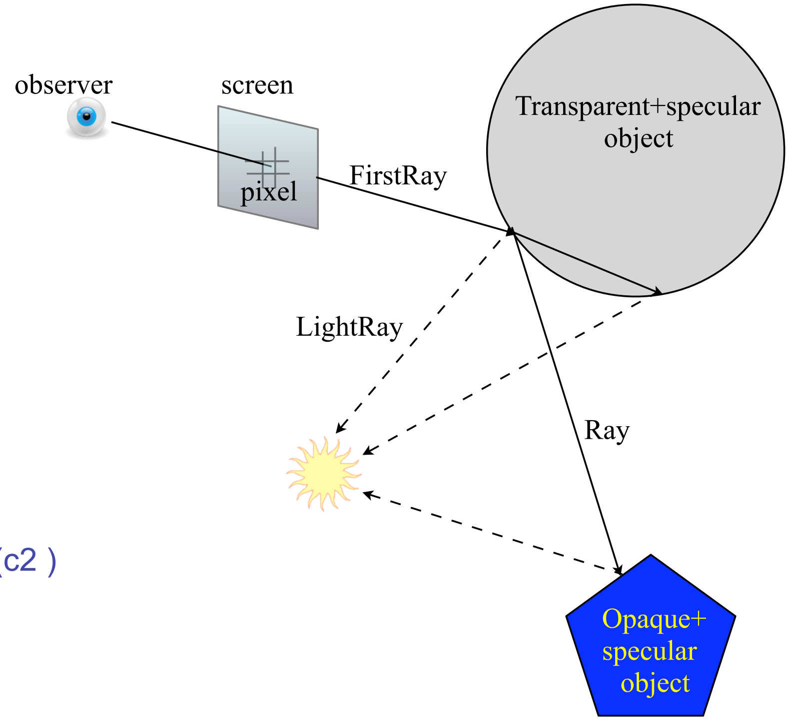
renderRay = **replace** r::Ray
by Scene.intersectRays (r)
if r.contribution () > ϵ

deleteRay = **replace** r::Ray, ω by ω **if** r.contribution () $\leq \epsilon$

enlighten = **replace** l::LightRay

by l.computeContrib ()

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



A more complex example...

import Pixel, Ray, LightRay, Contrib, Scene.

$\langle p_1, \ldots, p_n$, renderPixel$\rangle$

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

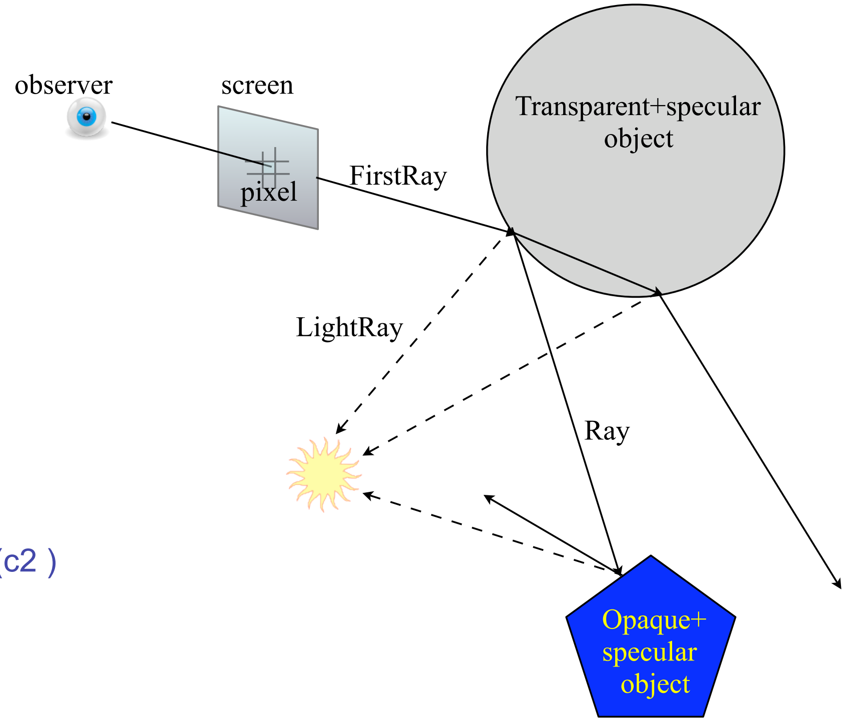
renderRay = **replace** r::Ray
by Scene.intersectRays (r)
if r.contribution () > ϵ

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

enlighten = **replace** l::LightRay

by l.computeContrib ()

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



A more complex example...

import Pixel, Ray, LightRay, Contrib, Scene.

⟨p_1 , . . . , p_n , renderPixel⟩

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

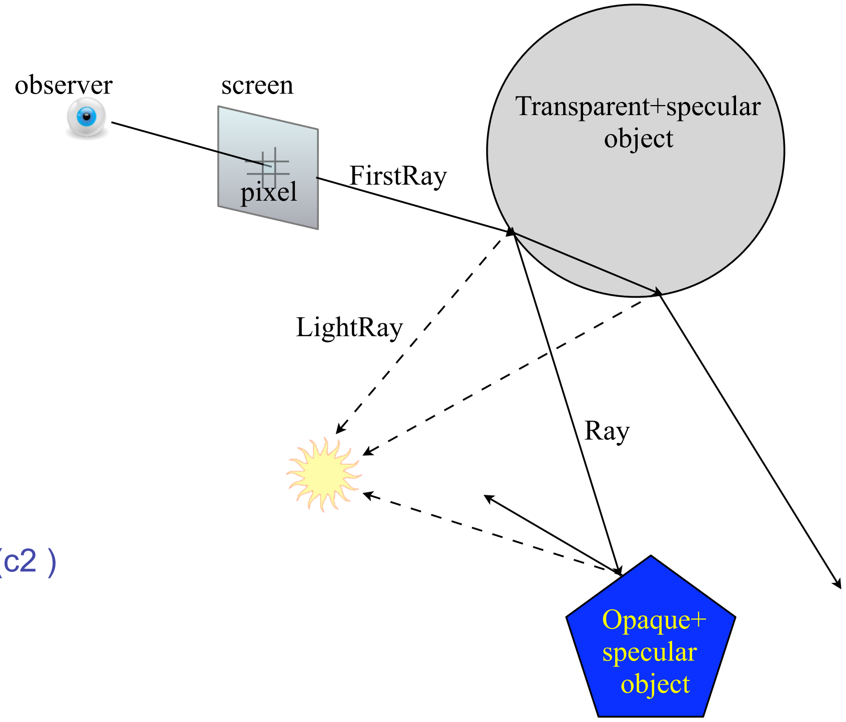
renderRay = **replace** r::Ray
by Scene.intersectRays (r)
if r.contribution () > ϵ

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

enlighten = **replace** l::LightRay

by l.computeContrib ()

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



Coordination rule

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.wil lBeRemoved (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)