

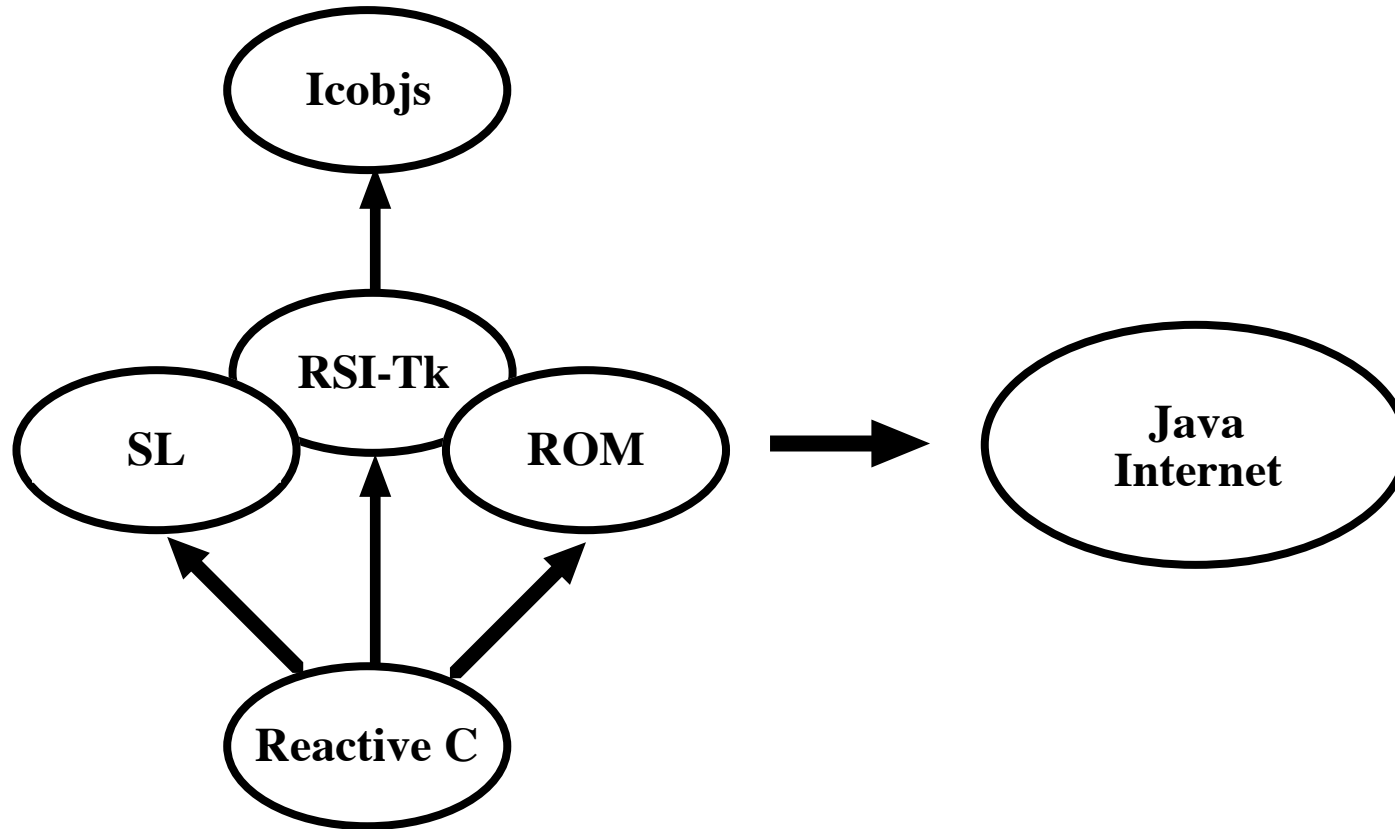
# Les SugarCubes

# Plan

---

- **Approche Réactive et SugarCubes**
- **Les Cubes**
- **Interaction avec Java**
- **Scripts Réactifs**

# Historique



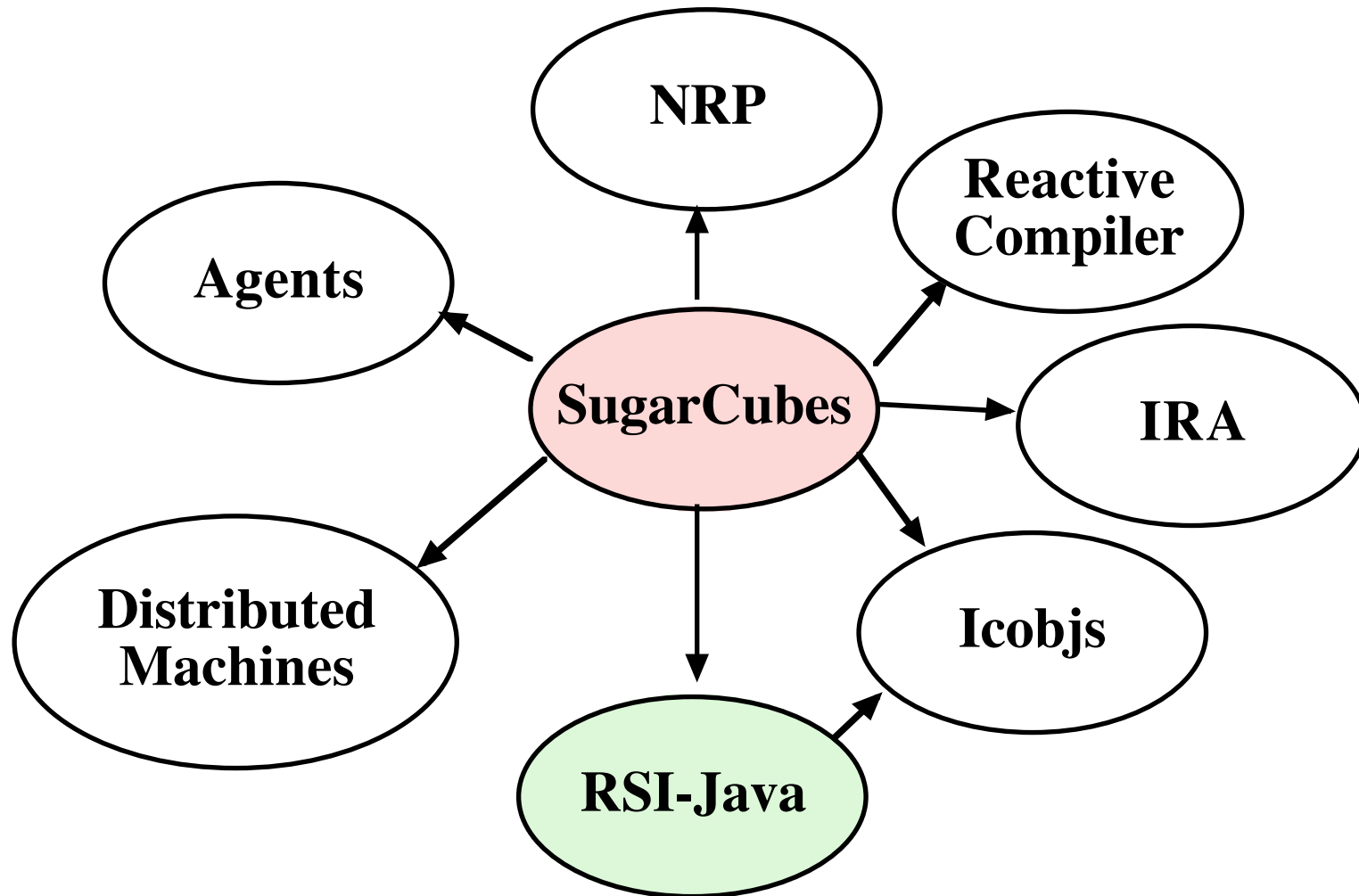
# Historique

---

## Pourquoi Java?

- simplicité (Orienté Objet)
- API très riche
- portabilité
- support réseau (Socket, RMI, ORBs...)

# Historique



# SugarCubes

Réactif + objets java =



**Ensemble de classes et d'interfaces Java**  
**Ancêtre de Junior**

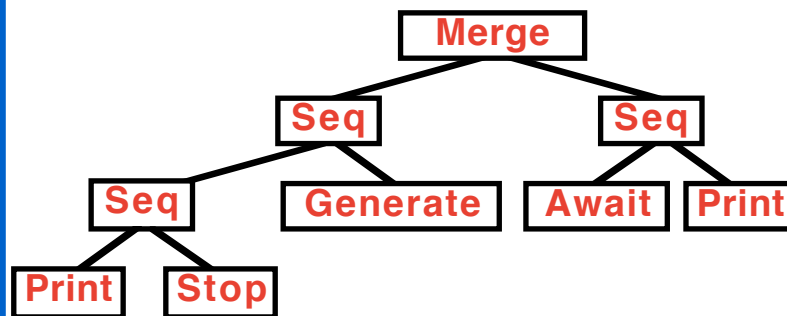
**2 points de vue possibles☐**

- **Framework avec une API (100% Java)**
- **Langage de description de comportements (expression du contrôle et de la concurrence)**

# SugarCubes

## 4 grands types d'objets ☐

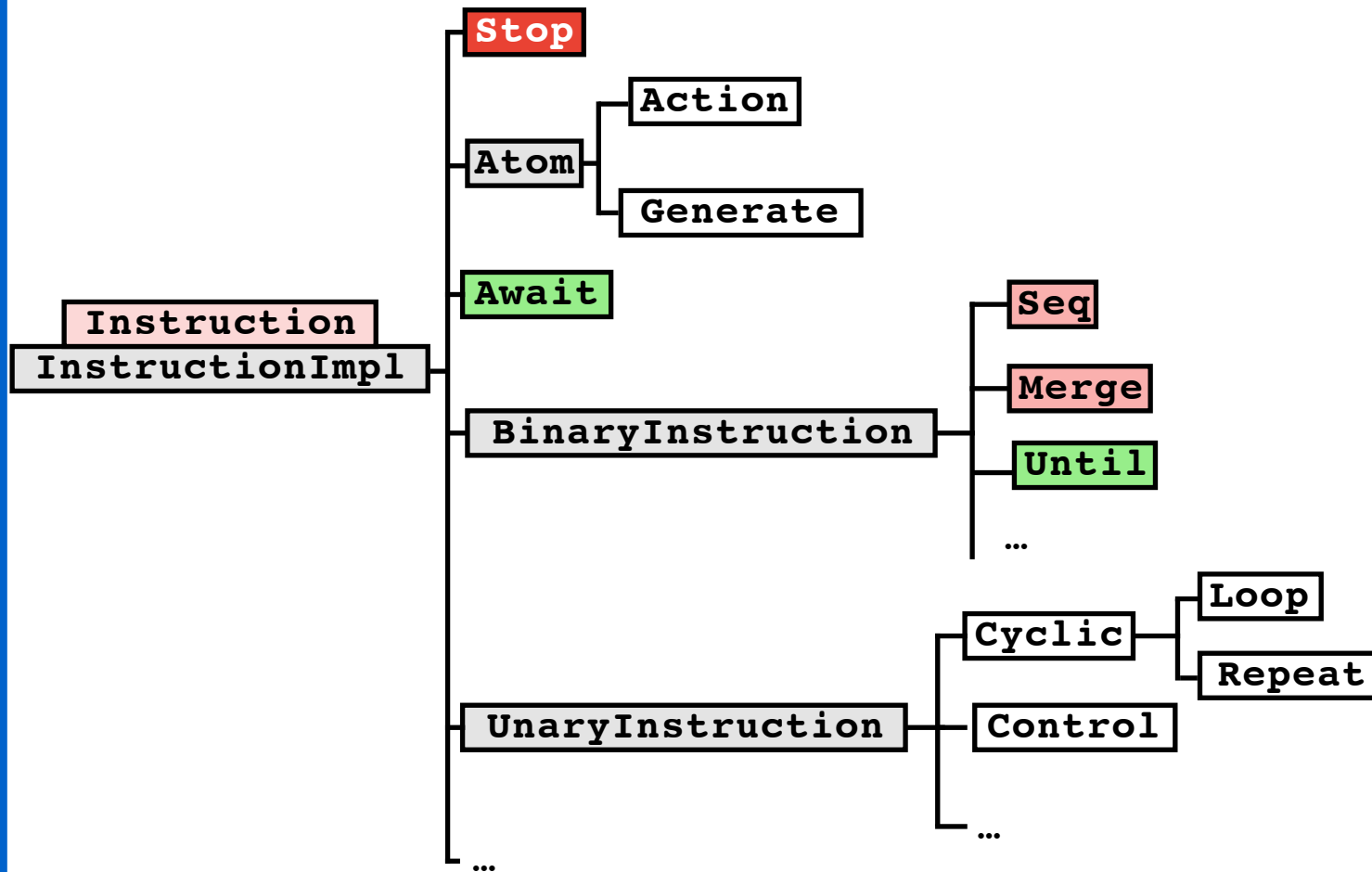
- Instructions réactives
- Classes événementielles
- Classes d'interfaçage avec Java
- Machines d'exécution réactives



```
Program p = SC.merge(  
  SC.seq(  
    SC.print("HelloWorld")  
    ,SC.stop()  
    ,SC.generate("e")  
  )  
  ,SC.seq(  
    SC.await("e")  
    ,SC.print("e!")  
  )  
);
```

# Instructions Réactives

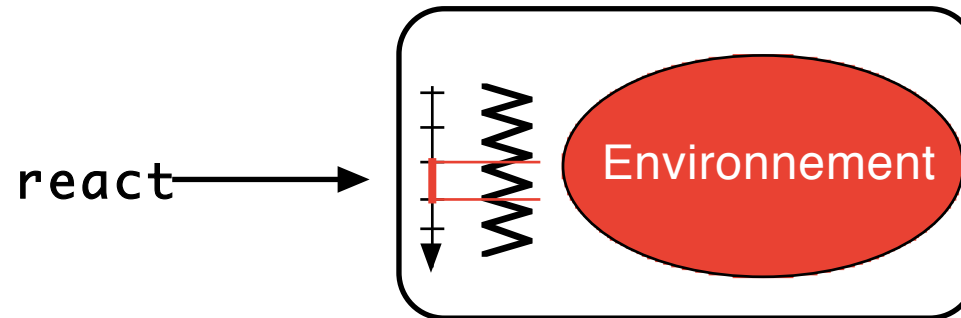
Les instructions réactives sont des objets java.



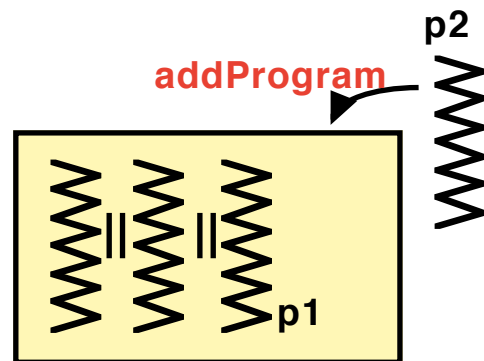


# Machine d'exécution réactives

Exécute un programme réactif, instant après instant (réactions) □



Ajouts dynamiques de nouveaux programmes en cours d'exécution, entre deux instants



```
Machine m = SC.machine();
```

```
Program p1 = .....
```

```
m.addProgram(p1);
```

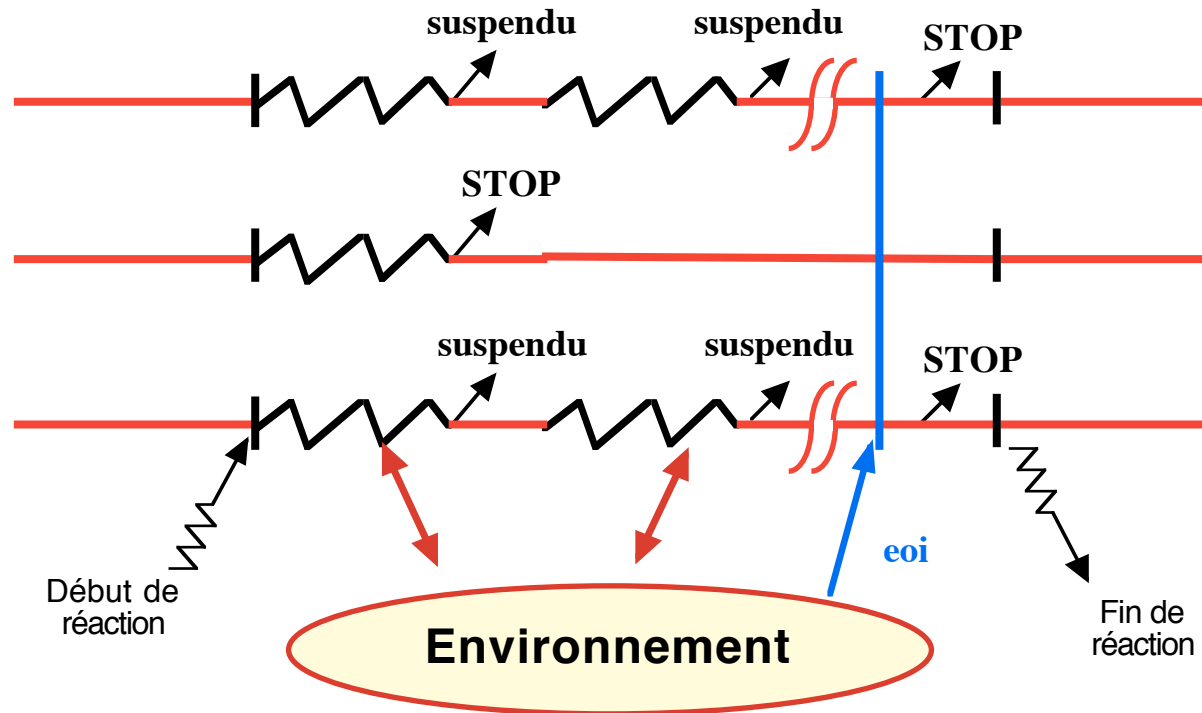
```
m.react();
```

```
Program p2 = .....
```

```
m.addProgram(p2);
```

```
m.react();
```

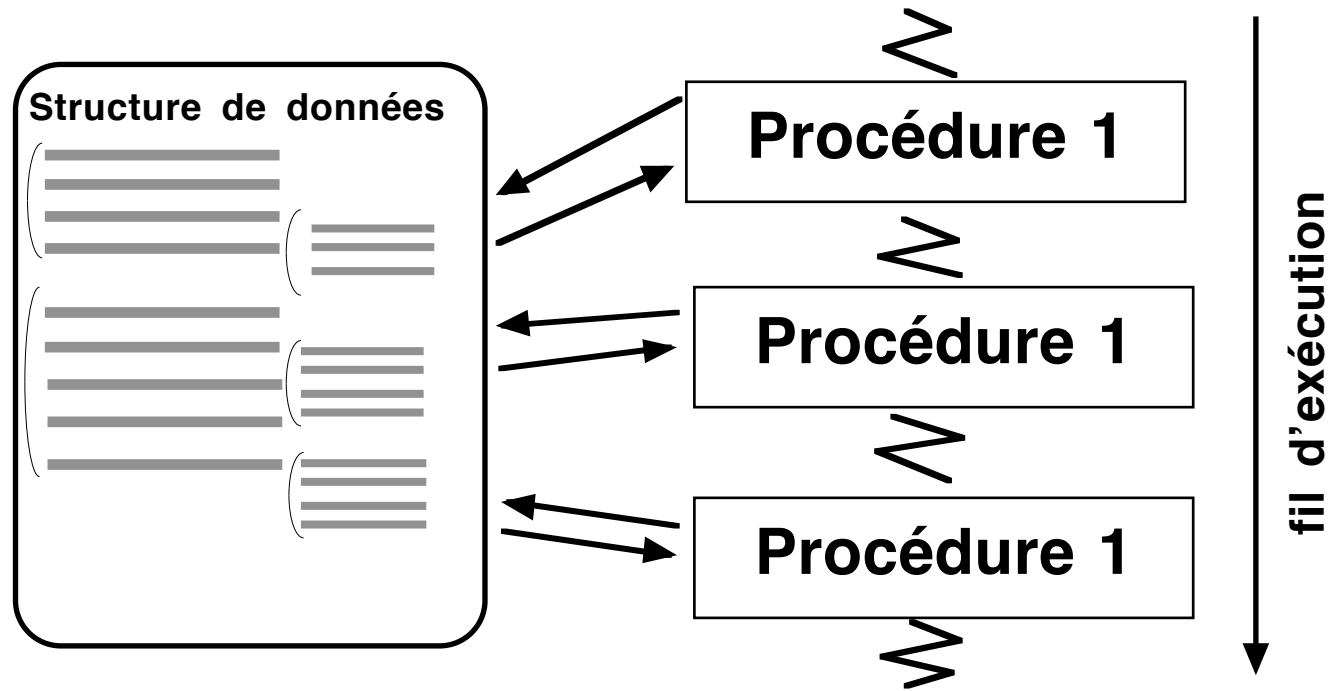
## Décision de la fin d'instant



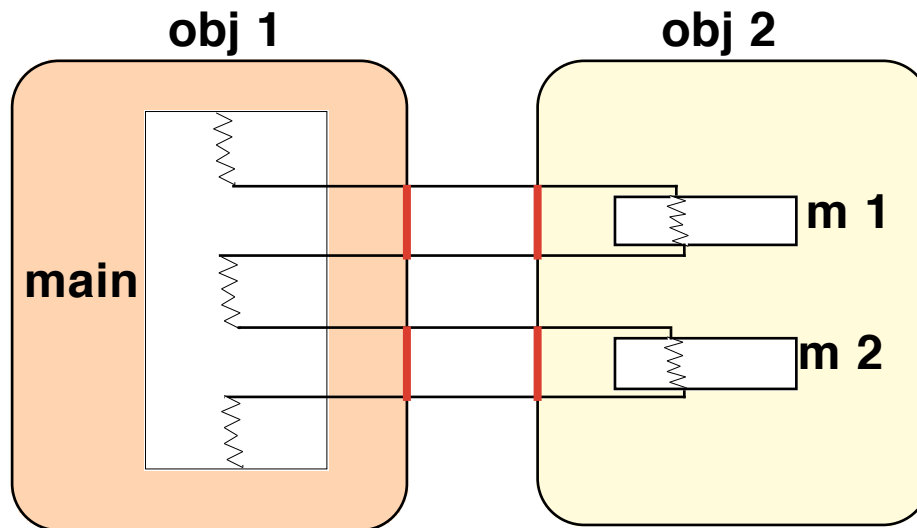
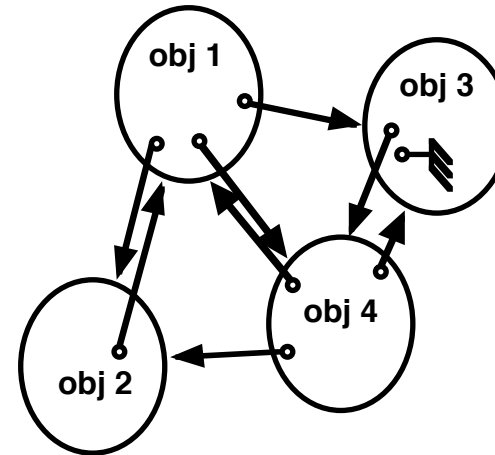
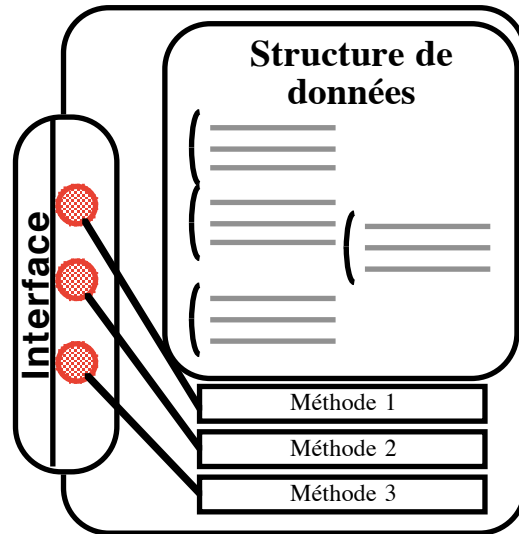
On déclare la fin de l'instant lorsqu'il ne reste plus que des composants stoppés ou suspendus et que l'environnement n'évolue plus

Algorithme de terminaison pouvant être appliqué à des systèmes distribués à travers le réseau

# Programme



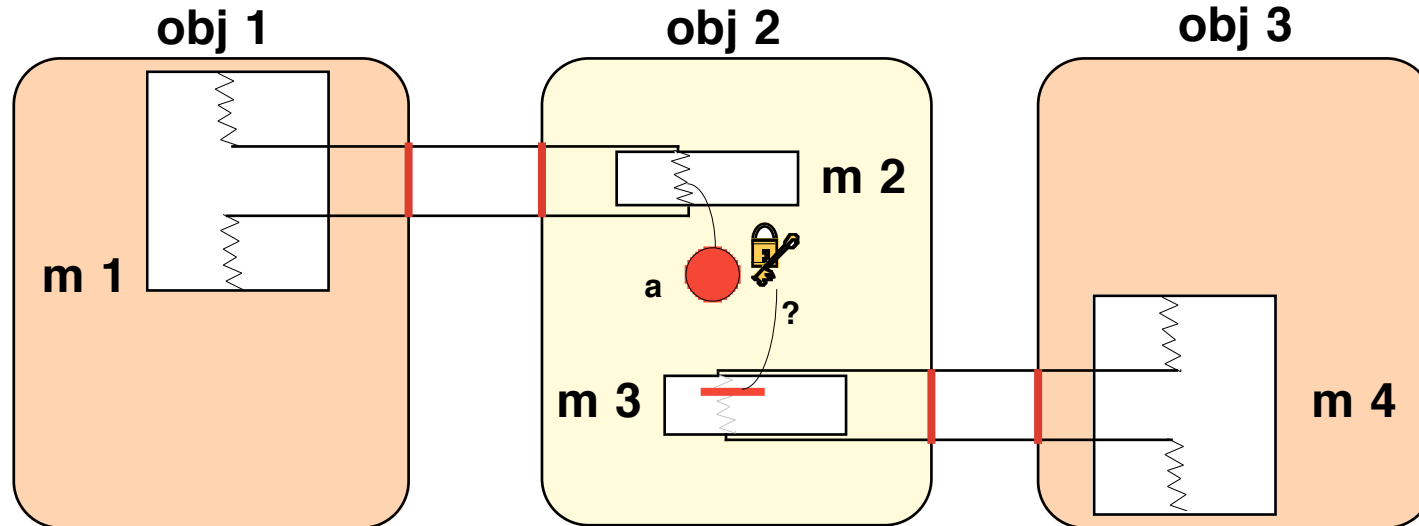
# Les objets



**Le fil d'exécution  
passe le contrôle  
d'un objet à un  
autre au gré des  
appels de  
méthodes**

# Objets et concurrence

## Contrôle d'exécution en multi-threads ☐

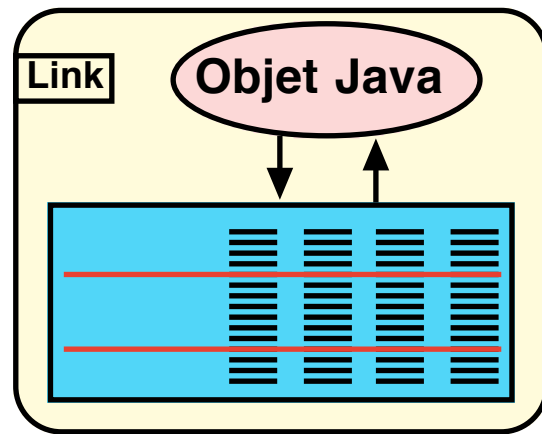


Lorsque plusieurs fils de contrôle d'exécution (Thread) aboutissent simultanément sur le même objet, on est obligé d'en tenir compte dans l'implémentation de cet objet en verrouillant l'accès à certains champs

Cette situation peut conduire à des **interblocages**

# Les Cubes

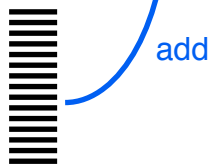
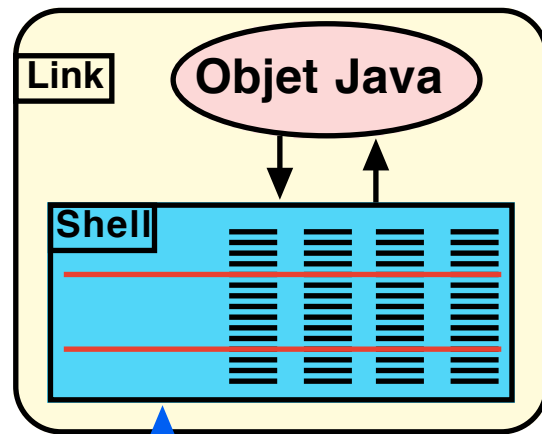
Un modèle dérivé de la notion d'**acteurs**



**Link = objet java + comportement réactif**

# Les Cubes

Un modèle dérivé de la notion d'**acteurs**



**Link = objet java + comportement réactif**

**Shell = ajout de composants en parallèle**

# Les Cubes

Un modèle dérivé de la notion d'~~acteurs~~



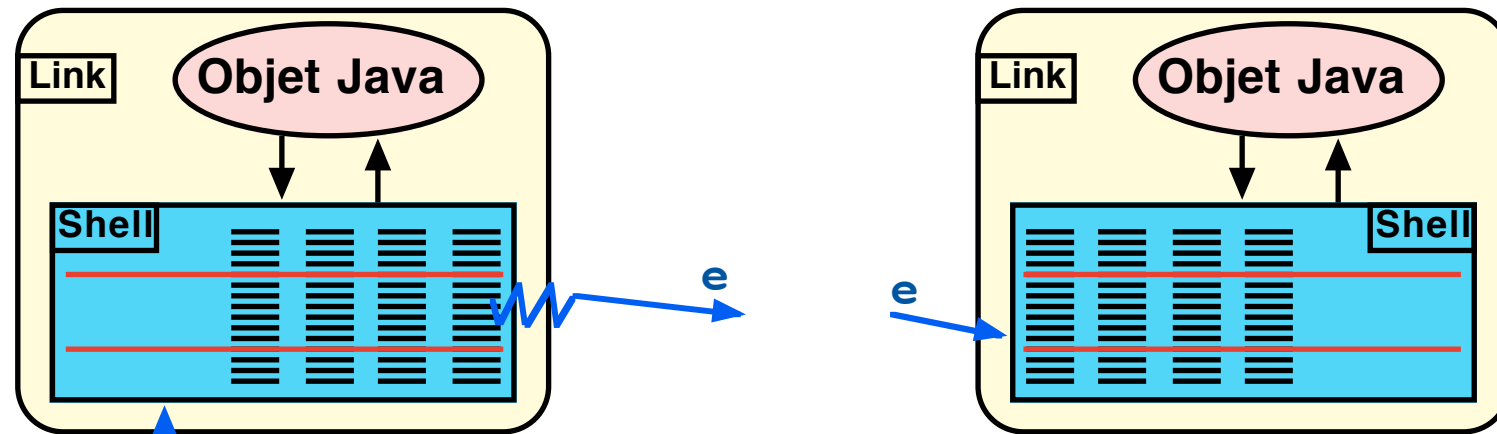
**Link = objet java + comportement réactif**

**Shell = ajout de composants en parallèle**



# Les Cubes

Un modèle dérivé de la notion d'**acteurs**



Link = objet java + comportement réactif

Shell = ajout de composants en parallèle

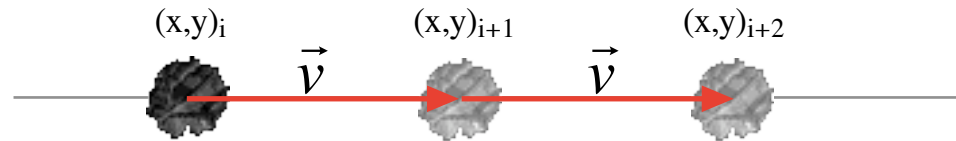
Les Cubes communiquent entre eux par **événements valués**

Chaque **occurrence** au cours d'un instant implique une réaction → exécution d'une **opération atomique particulière**

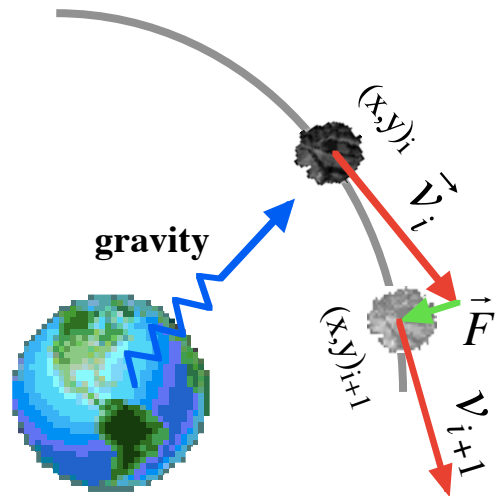
# Plan

## Météore ☐

Objet réactif avec un comportement physique d'inertie



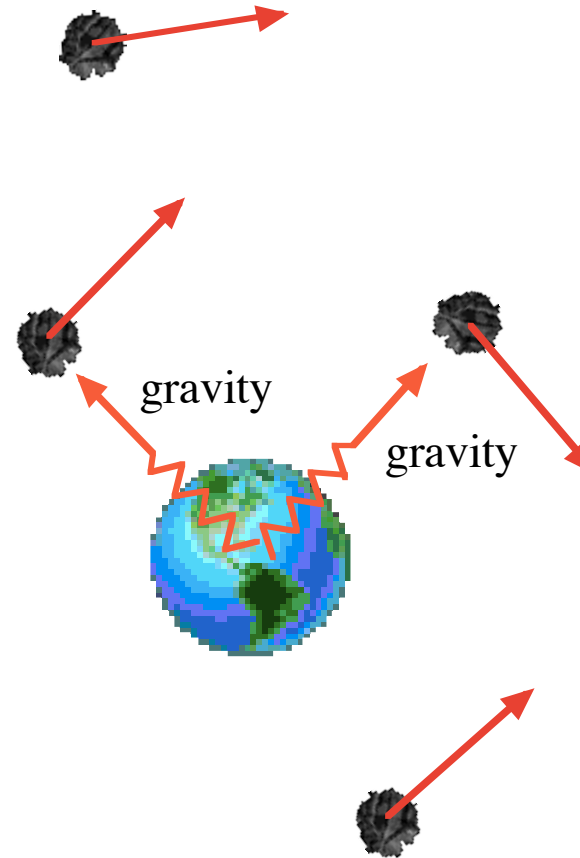
On ajoute en parallèle un objet réactif planète



# Événements diffusés

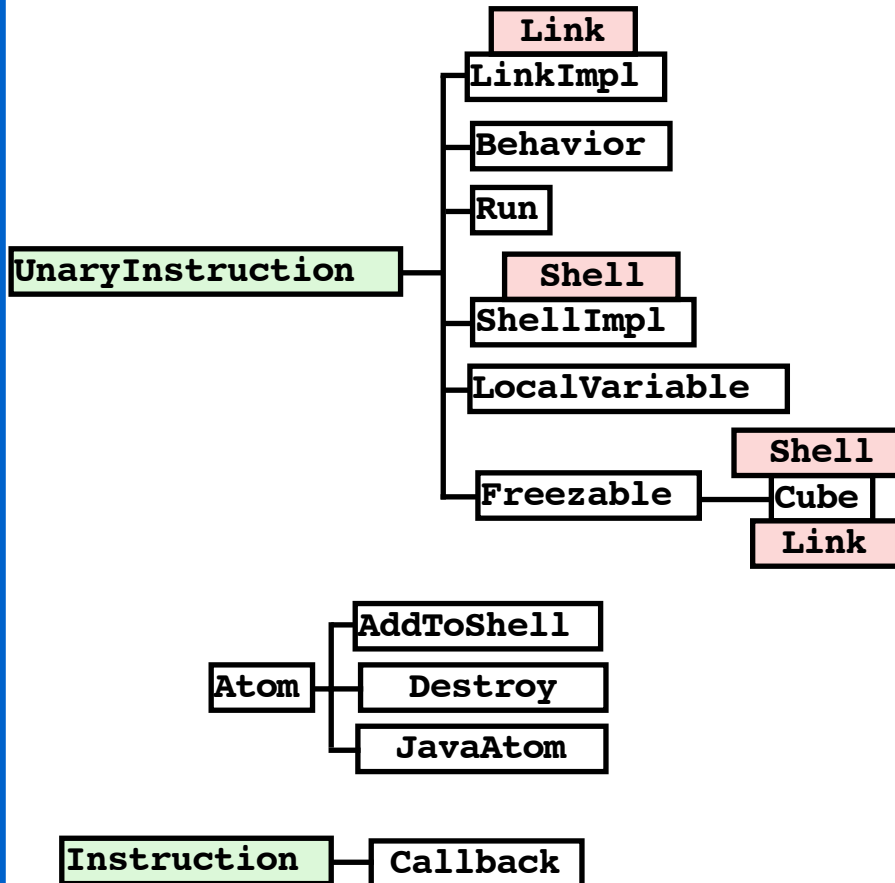
La diffusion instantanée des événements assure la cohérence de la simulation

Plusieurs planètes peuvent générer l'événement "gravity", l'ensemble des occurrences sera reçu par tous les objets à l'écoute

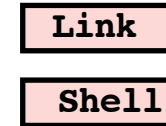


# Intéraction SugarCubes-Java

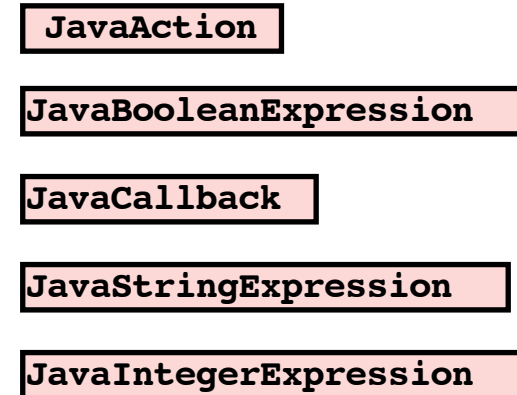
## Instructions



## Interfaces



à implémenter par l'utilisateur



...

# Intéraction SugarCubes-Java

```
SC.cube("exemple",new Frame("exemple"),SC.seq(
    SC.action(new InitFrame())
    ,SC.loop(
        SC.seq(
            SC.generate("DisplayHere",new GraphicsArg())
            ,SC.stop()
        )
    )
),
new DisposeFrame()
,new DisposeFrame()
,new WarmupFrame()
);
```

création d'un Cube

objet Java associé au comportement

génération d'un événement valué

actions atomiques à exécuter à des moments critiques de la vie d'un cube:

- terminaison
- gel
- réveil

# Interface Java

## Action atomique

```
public interface JavaAction extends java.io.Serializable
{
    public void execute(Object self,LocalVariables vars,ReactiveEngine engine);
}
```

objet Java associé

accès au contexte d'exécution

variables locales

## Évaluation d'expression atomique

```
public interface JavaIntegerExpression extends JavaExpression
{
    int evaluate(Object self,LocalVariables vars,ReactiveEngine engine);
}
```

l'évaluation atomique de cette méthode retourne un entier

## Constantes

```
public final class JavaIntegerValue implements JavaIntegerExpression
{
    protected final int i;
    public JavaIntegerValue(int i){ this.i = i; }
    public final int evaluate(Object self,LocalVariables vars,ReactiveEngine engine){
        return i;
    }
}
```

# Interface Java

---

## Variables Locales

```
SC.localVariable("btn",theClass, ...);
```

## Interface

```
public interface LocalVariables{  
    public Object getVariableValue(String varName,Class expectedType)  
        throws NoSuchLocalVariable;  
    public Object getVariableValueNoCheck(String varName)  
        throws NoSuchLocalVariable;  
    public Object setVariableValue(String varName,Object value)  
        throws NoSuchLocalVariable;  
}
```

# Interface Java

## Reactive Engine

```
public final class ReactiveEngine{  
    ...  
    public final void addProgram(Program aComponent){...}  
    public final void generateEvent(Identifier eventName,Argument[] args){...}  
    public final void addTask(Task t, boolean beforeReaction){...}  
    public final void removeTask(Task t){...}  
    public final Program getFrozenProgram(String instructionName){...}  
    public final String curentCubeName() {...}  
}
```

## Tâches inter instant

```
public interface Task  
{  
    public static final boolean AFTER_REACTION = false;  
    public static final boolean BEFORE_REACTION = true;  
    public void doIt();  
}
```

 Méthode appelée inter instant par la machine d'exécution



# Callback réactive

## Évaluation d'une callback réactive

```
public interface JavaCallback extends java.io.Serializable
{
    void execute(Object self,LocalVariables vars,ReactiveEngine engine
                ,Argument args[]);
}
```

Valeurs véhiculées par l'occurrence 

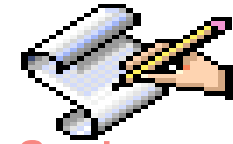
## Utilisation d'une Callback réactive

```
SC.cube("exemple",new DemoObject()
    ,SC.merge(
        SC.loop(
            SC.action(new Move())
            ,SC.stop()
        )
        SC.loop(
            SC.callback("DisplayHere",new DisplayReaction())
        )
    )
);
```

on réagit à chaque instant aux éventuelles occurrences de l'événement DisplayHere 

# Scripts réactifs

Réactif + interprétation =



Scripts  
Réactifs

```
|| (await e; {System.out.println("e!")})  
generate e
```

Traduction en SugarCubes :

```
Program program =  
  SC.merge(  
    SC.seq(  
      SC.await("e")  
      , SC.action(new JavaPrintAction("e!"))  
    )  
    , SC.generate("e")  
  );
```

# Scripts Réactifs

## Les Cubes en RSI-Java

```
object nom with {objet}  
  corps  
  methods  
    m1  
    m2  
  on terminate  
    {final_action}  
  on freeze  
    {freeze_action}  
  on warmup  
    {warmup_action}  
end
```

opérations sur les objets `freeze`, `destroy`,  
`send`

## Exemple

associe un objet de type Frame au script réactif



```
object myWindow with {new java.awt.Frame()};  
  await show;  
  {this.setVisible(true)};  
  halt  
  
  on terminate {dispose()}  
end.
```



action atomique exécutée lorsque l'objet réactif termine

# Scripts Réactifs

---

## Implémentation des appels externes☐

- **invocation de méthodes publiques** sur☐ (i)l'objet java associé au cube, (ii)l'un de ses champs publiques ou (iii)une classe publique
- **invocation de constructeurs publiques** (opérateur new)
- **accès à la valeur d'un champ public** d'un objet ou d'une classe publique
- **affectation d'une valeur à un champ public** de l'objet ou d'une classe publique à l'aide de l'opérateur = .
- **accès aux variables locales** (lecture, affectation)
- **gestion des types de base** : byte, short, int, long, float, double, boolean, char, String, null.
- **objet java associé référencé par this** (pour lever les ambiguïtés).