# SugarCubes

*Jean-Ferdinand Susini*
*Maître de Conférences, CNAM*
*Chaire systèmes enfouis et embarqués*

*Paris, le 9 janvier, 2009*

Cnam
CONSERVATOIRE NATIONAL
DES ARTS ET METIERS

Cédric

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

# Plan

- Les SugarCubes au dessus de J2ME
- Quelques résultats expérimentaux
- Les SugarCubes et le Multi-clocks
- Conclusion et perspectives

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

# Plan

- Les SugarCubes au dessus de J2ME
- Quelques résultats expérimentaux
- Les SugarCubes et le Multi-clocks
- Conclusion et perspectives

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

# L'approche réactive au dessus de J2ME



Les SugarCubes Lite : facilités pour exprimer (i) l'ordonnancement et le contrôle des tâches ; (ii) la communication entre les différentes tâches.

- *ordonnancement en ligne : interprétation de la politique d'ordonnancement ; gérer la création et la reconfiguration dynamique des composants concurrents*
- *efficacité : capable de gérer des milliers d'activités concurrentes ; communication au moyen de milliers d'événements différents*
- *faible emprunte mémoire à l'exécution (files d'attentes bornées, taille de l'interprète, taille des programmes, …)*
- *100% pure Java (basé sur la spec. 1.1 de Java) : portage J2ME facilité. environnement minimaliste*
- *conservation de la structure des SugarCubes:*
  - *instructions réactive (jeu d'instructions réduit)*
  - *machines d'exécution réactive (services en moins)*
  - *classes et interfaces prenant en charge les interactions entre les programmes réactifs et les objets java standard*

# SugarCubes Lite



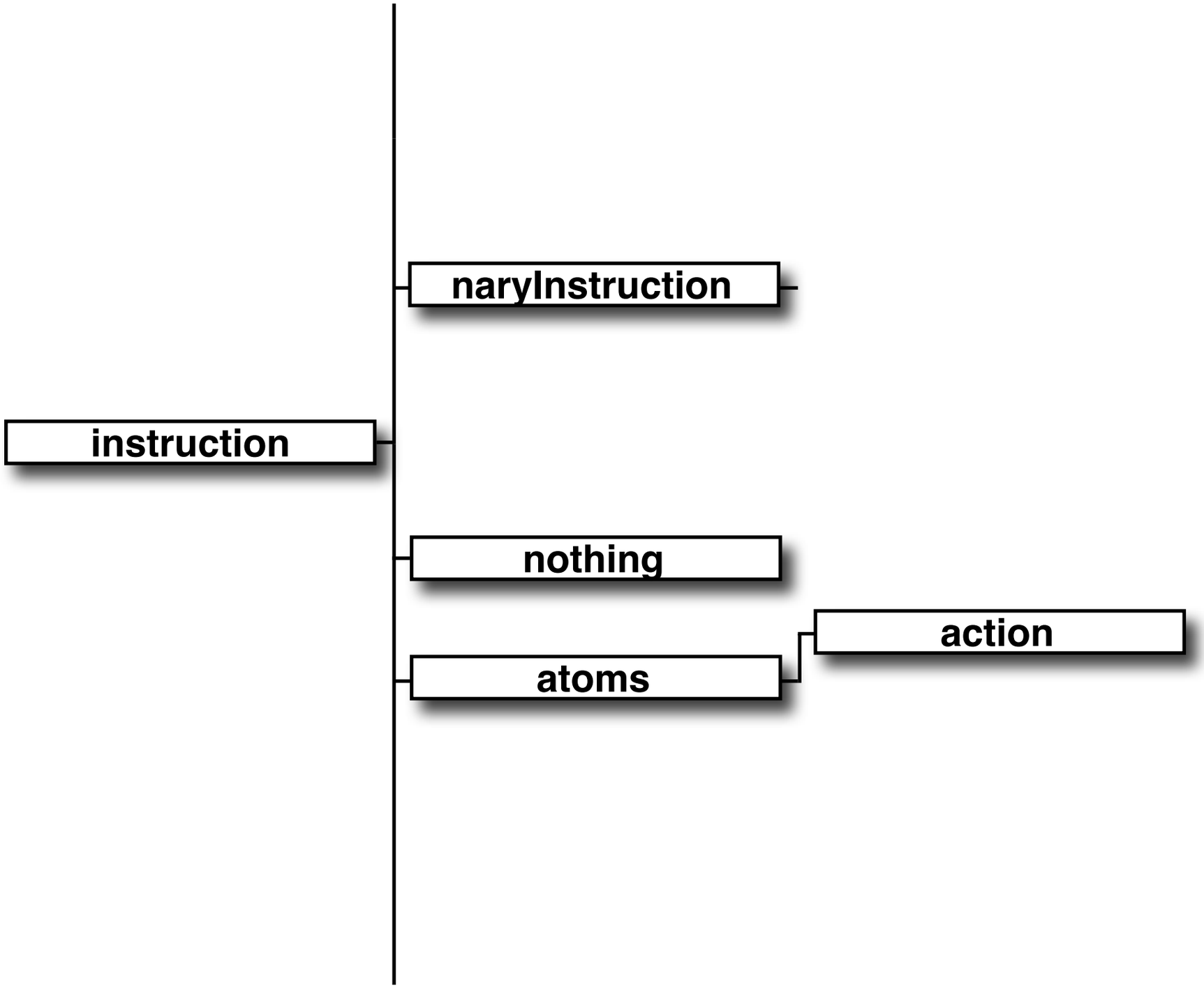
- Un framework réduit :
  - Seules les instructions primitives ou fréquemment utilisées. On élimine les instructions qui servent essentiellement de “sucre syntaxique”. (if, rif, les threads réactives, les actions de traces, les variantes trop nombreuses des différentes instructions, …)
  - Mécanismes abandonnés : migration de code, identificateurs d’événements et environnement simplifiés, pas de trace, …

➡ SCv4: 223KB, SCL: 96KB (jar zippé, pas d’obfuscateur de code)

- Nouvelles fonctionnalités introduites :
  - modèle de prog. multi-horloges (synchrone/asynchrone)
  - meilleure intégration avec les threads

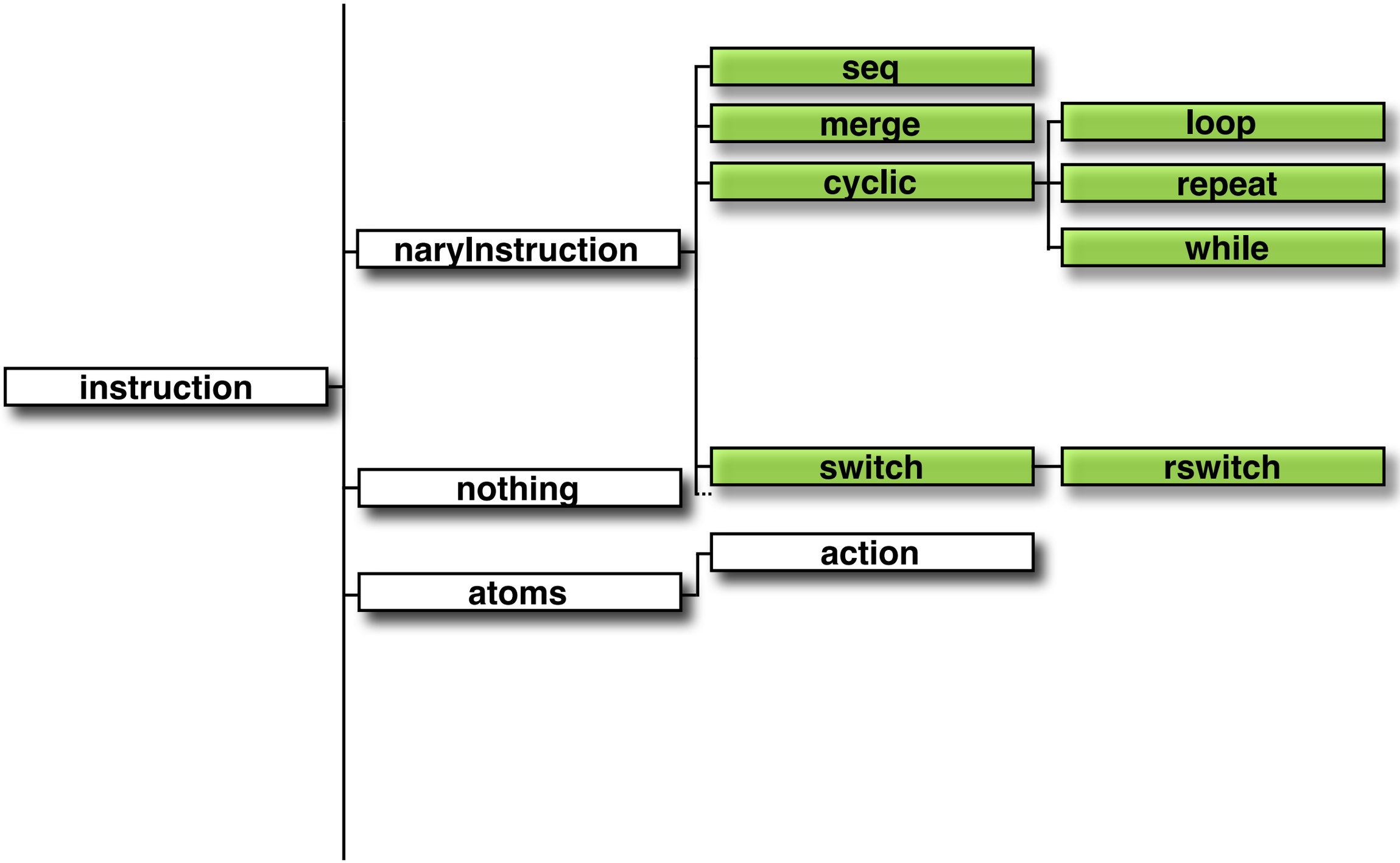
# Reactive Instructions





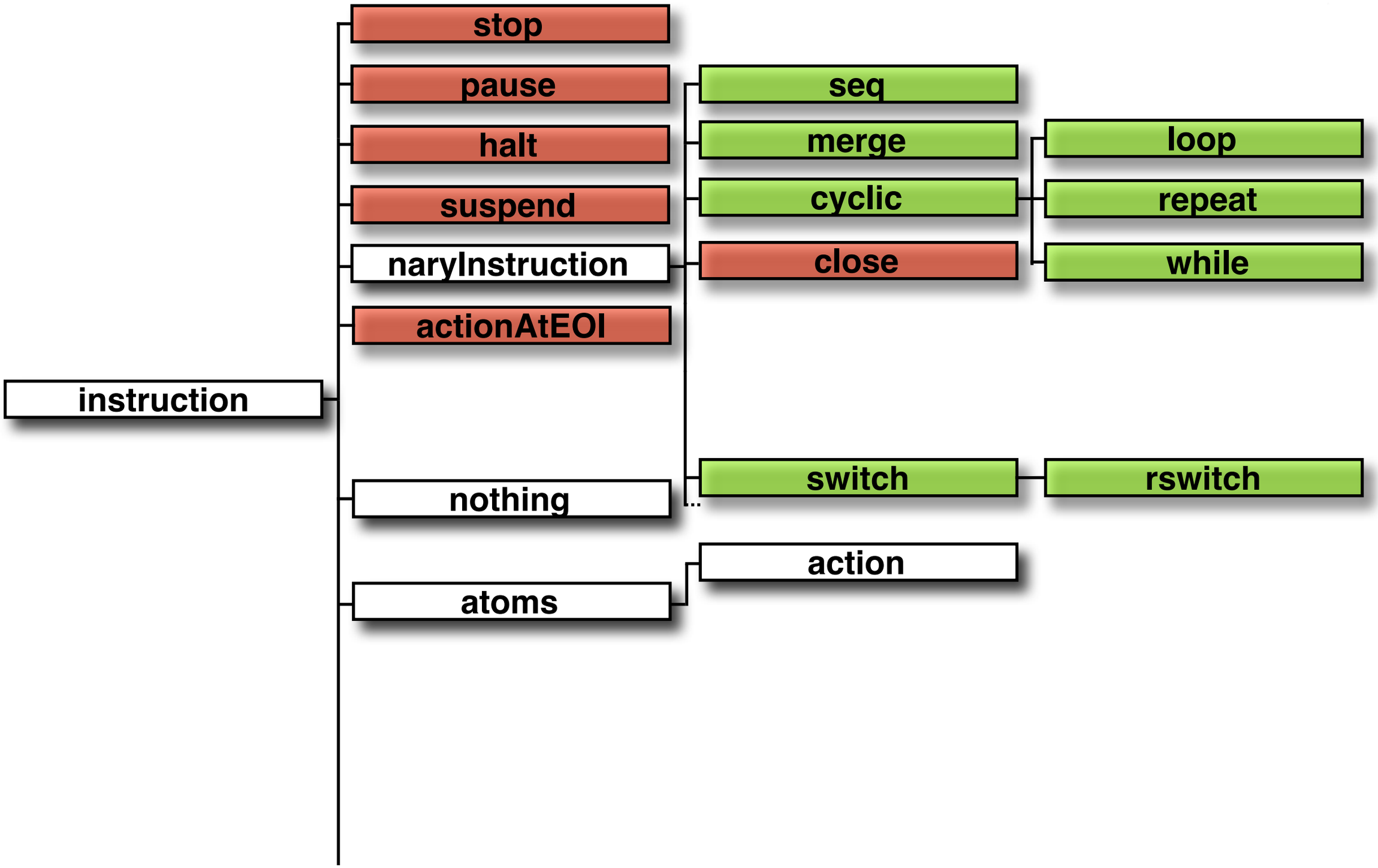
# Reactive Instructions





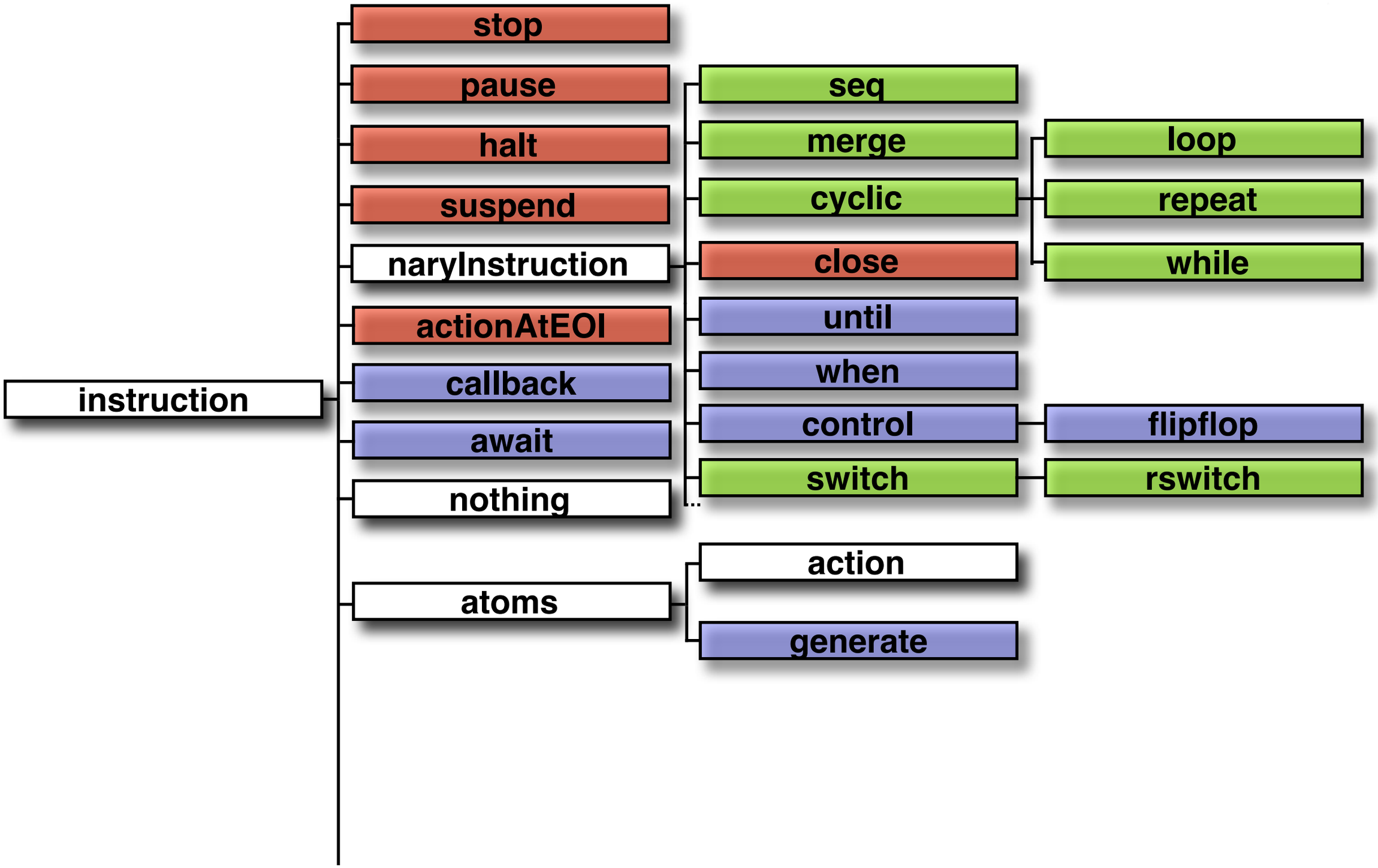
# Reactive Instructions





# Reactive Instructions

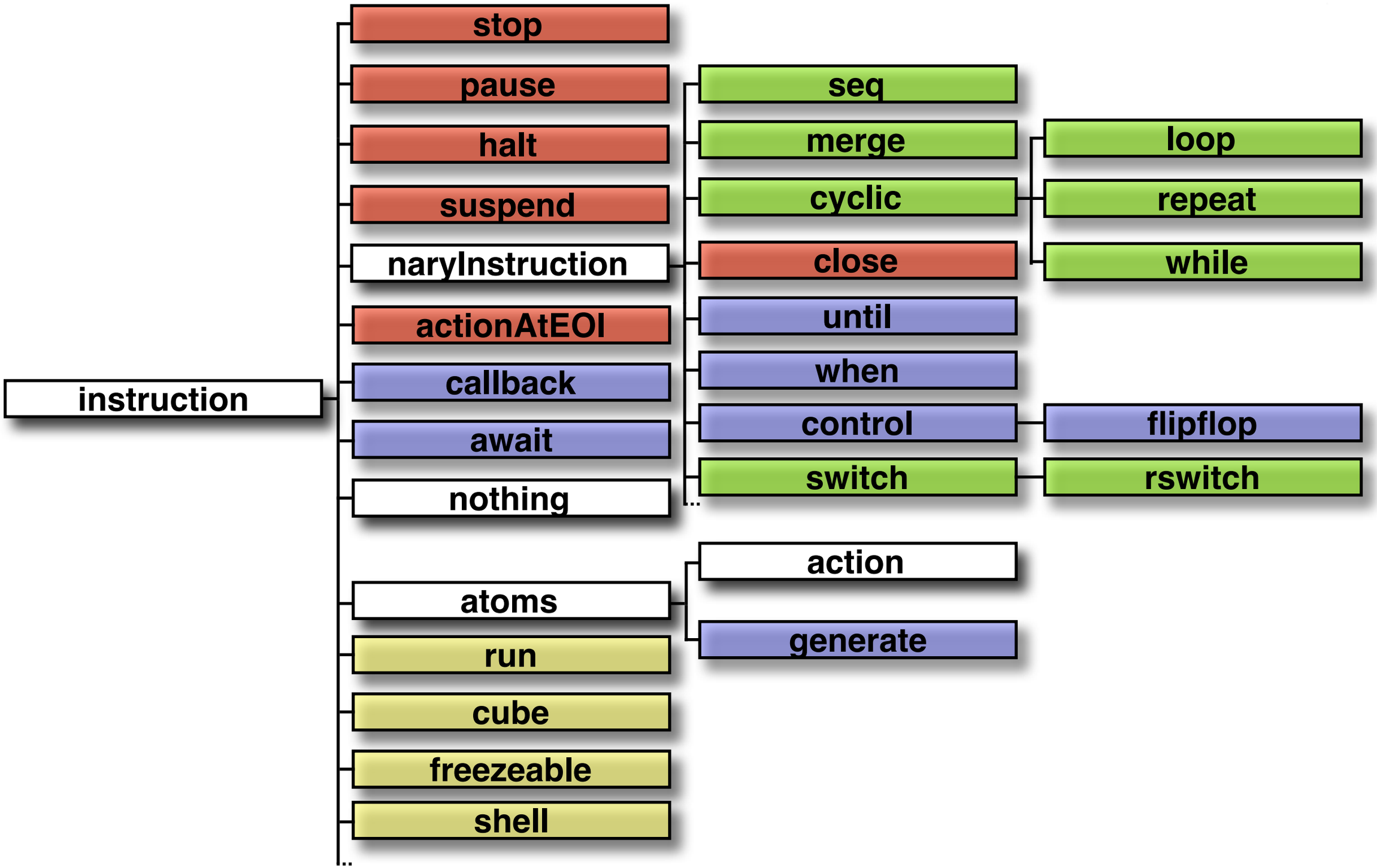




CONSERVATOIRE NATIONAL DES ARTS ET METIERS

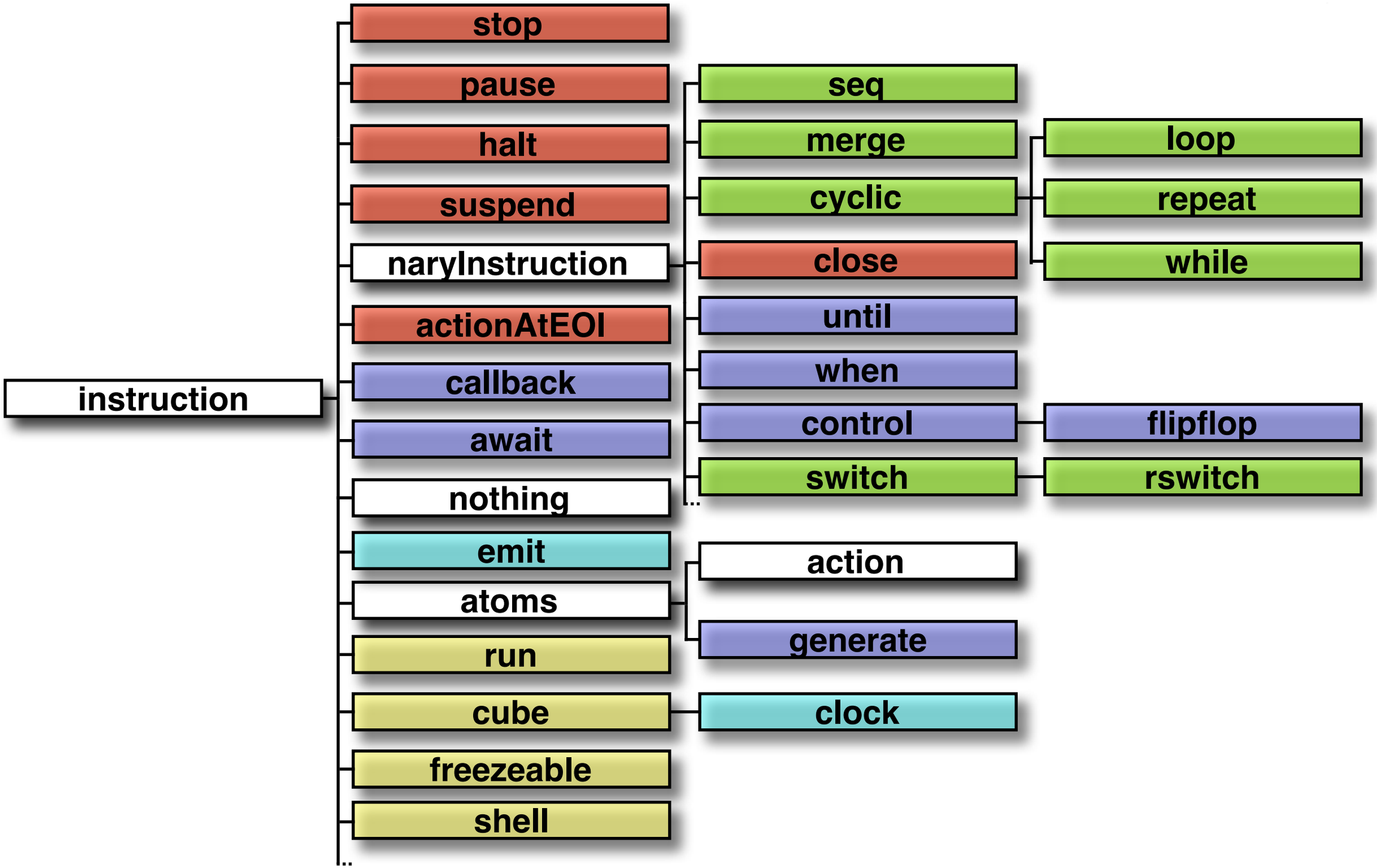
# Reactive Instructions





# Reactive Instructions





# Exemple de code J2ME

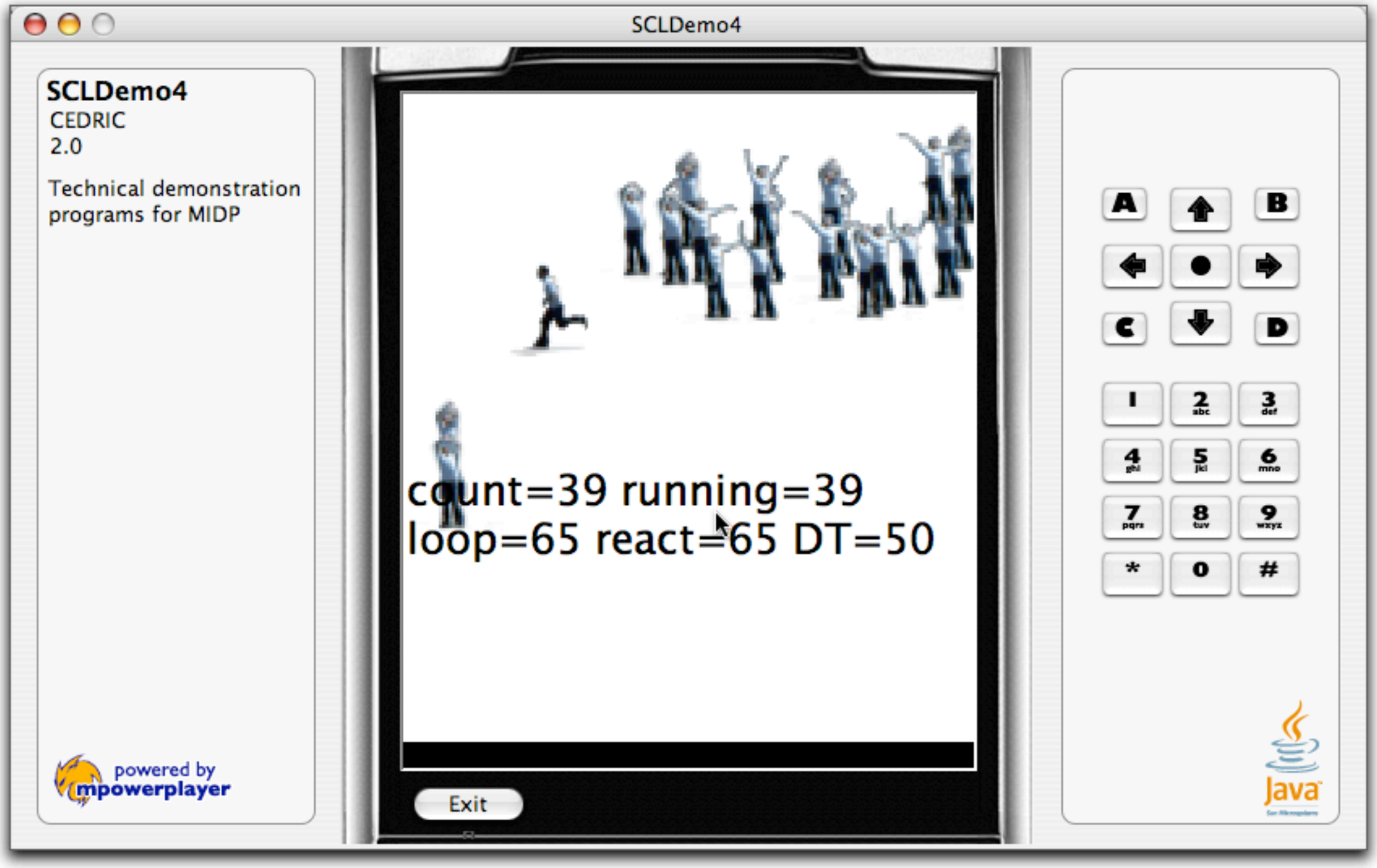


```
import cedric.scl.*;
public class SCLDemo extends MIDlet implements CommandListener, Runnable
{
   private Display display;
   private Machine machine = SC.emptyMachine();
    private Command exitCommand = new Command("Exit", Command.EXIT, 1);
    private Form view = new Form("SCLDemo");
    private boolean done = false;
    private final JavaAction PRINT_HELLO_ACT = new JavaAction(){
        public void execute(final Object self
                            , final ReactiveEngine engine){
            view.append("Hello World!\n");
        }
    };
  protected void startApp() throws MIDletStateChangeException {
     display = Display.getDisplay(this);
     Program p = SC.loop(SC.seq(SC.await("e"), SC.action(PRINT_HELLO_ACT)));
      machine.addProgram(p);
       machine.setMinimumTimeOfReactions(10);
        view.addCommand(exitCommand); view.setCommandListener(this);
        display.setCurrent(view); new Thread(this).start();
    }
    public void run(){
        do{
            view.append((i++)+"-: ");
        }
        while((!machine.react())&&!done);
    }
...}
```

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

# Exemple





CONSERVATOIRE NATIONAL DES ARTS ET METIERS

# Plan

- Les SugarCubes au dessus de J2ME
- Quelques résultats expérimentaux
- Les SugarCubes et le Multi-clocks
- Conclusion et perspectives

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

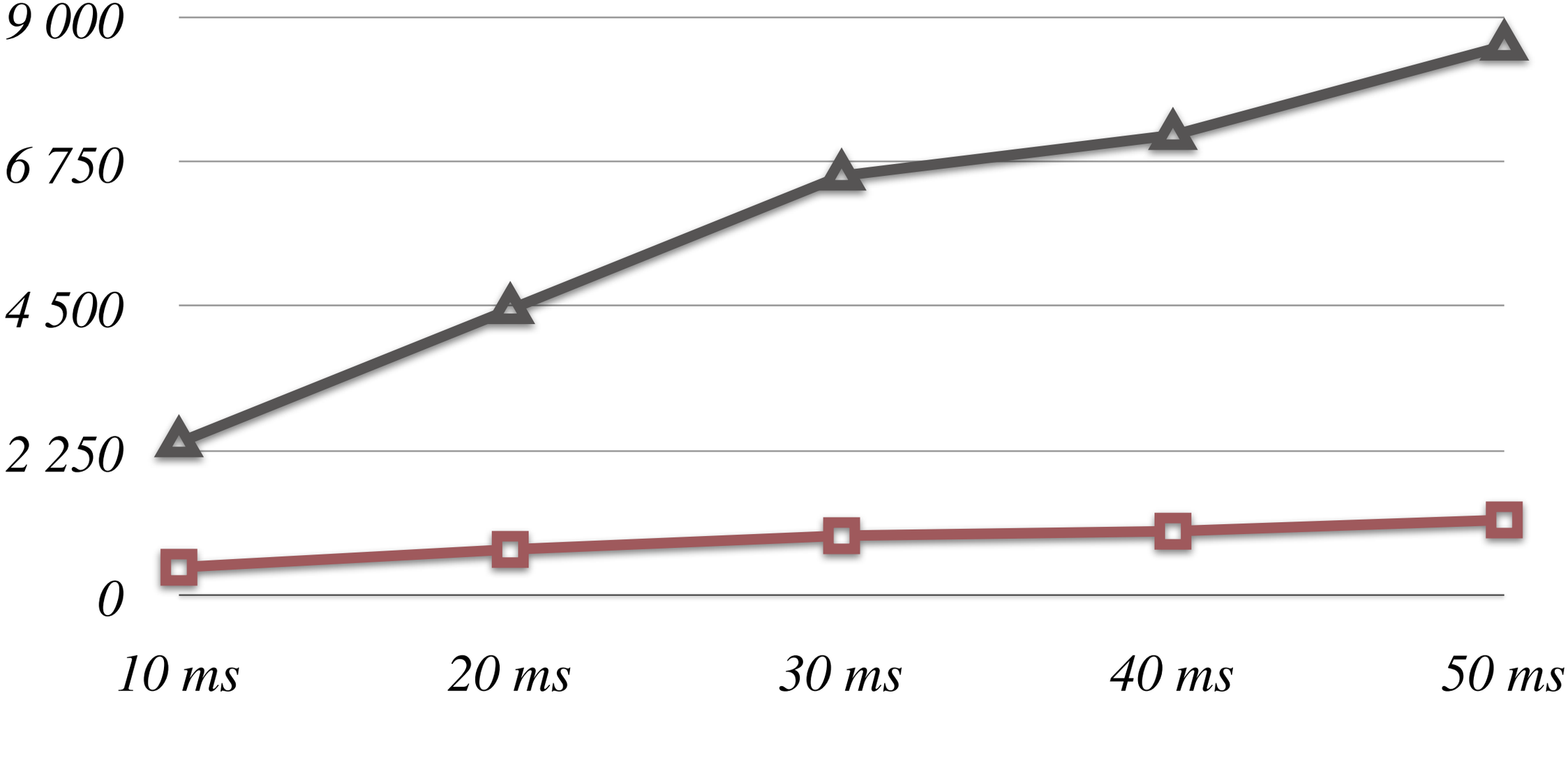
# Le protocole expérimental



- Utilisation d'un émulateur générique J2ME sur une station de travail
- Objectif des premières mesures : valider le passage à l'échelle :
  - réactivité
  - consommation mémoire
- Développement d'une MIDlet qui génère dynamiquement des activités concurrentes fortement synchronisés (période de synchronisation variable
  - Objectif : "respect des échéances"
- Implantation réactive et basée sur des threads

# Performance and Scalability (1)

Tests réalisés sur un émulateur tournant sur une station de travail (HotSpot VM)



CONSERVATOIRE NATIONAL DES ARTS ET METIERS

# Analyse



- Résultats très encourageants mais ...

  un air de “déjà vu” !

  - Performances peu réalistes (10 000 composants pour 50ms )
  - Consommation mémoire étonnante (plus de 1000 threads allouées)

# Analyse



- Résultats très encourageants mais ...

  un air de “déjà vu” !

  - Performances peu réalistes (10 000 composants pour 50ms )
  - Consommation mémoire étonnante (plus de 1000 threads allouées)

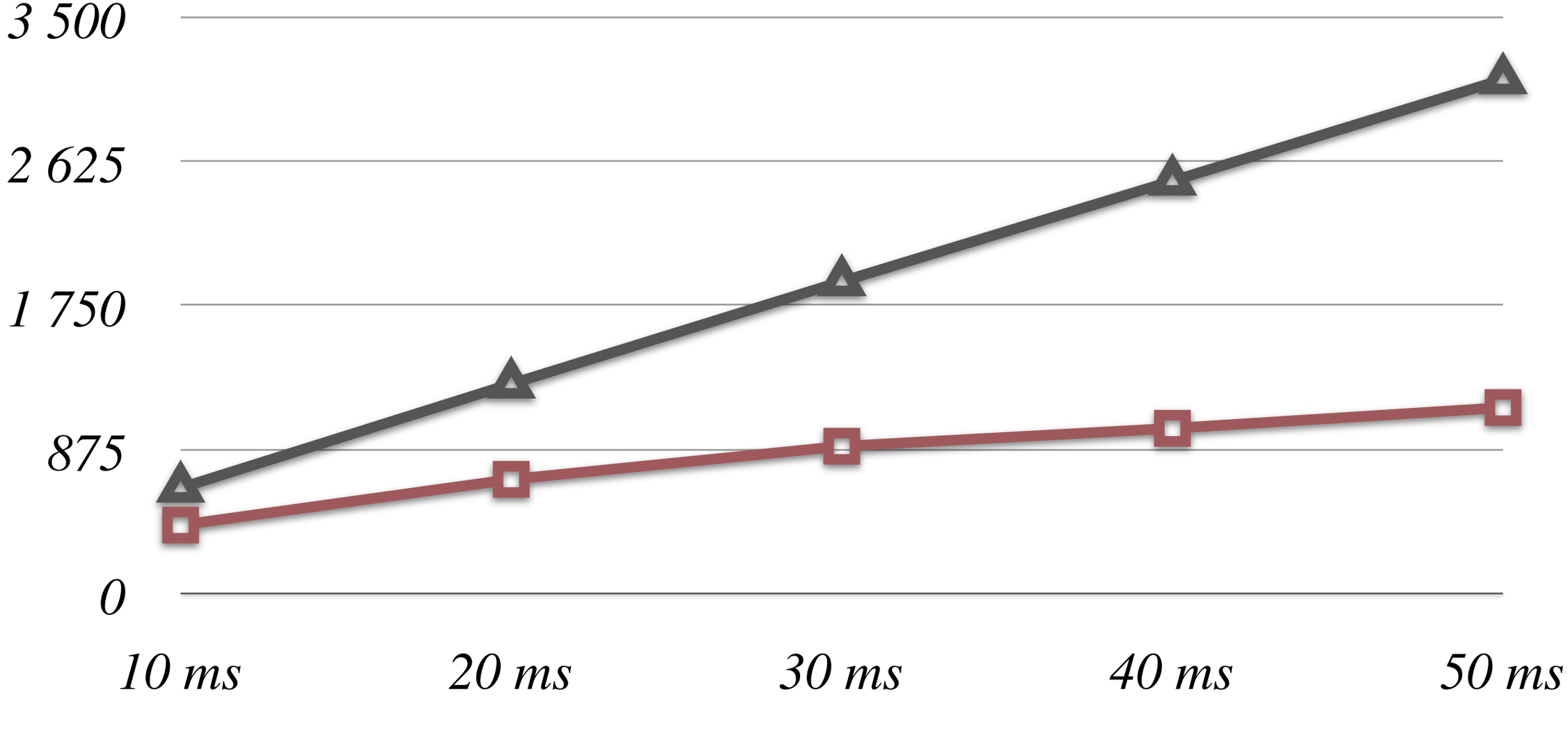
→ **Supprimer l’utilisation du JIT et de Hotspot**

# Performance and Scalability (2)

Tests réalisés sur un émulateur tournant sur une station de travail (mode interprété)

Threads

SCL Activities



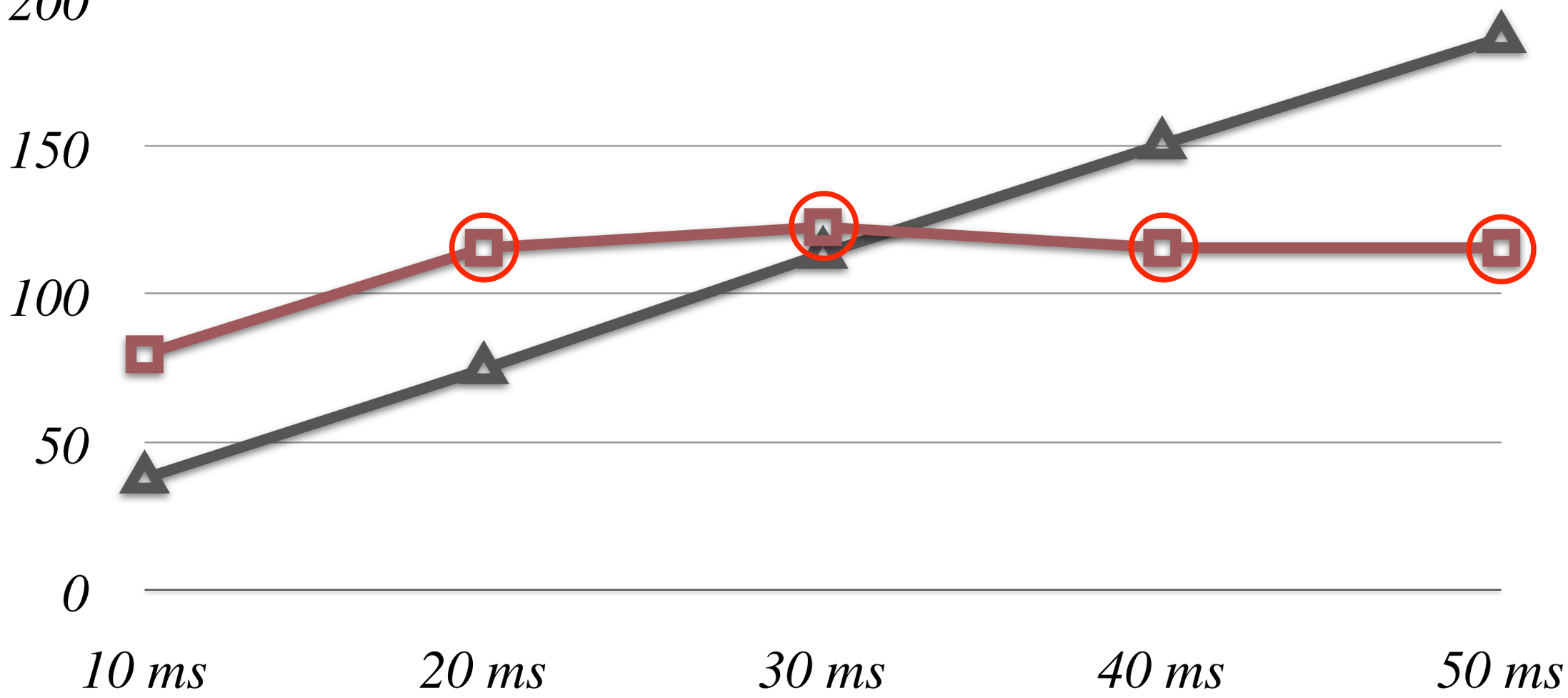
CONSERVATOIRE NATIONAL DES ARTS ET METIERS

# Performance and Scalability (3)



Tests exécutés sur un Palm Treo 600 avec une JVM : J9-571 (CLDC/MIDP-2)

Threads  SCL Activities  memory full



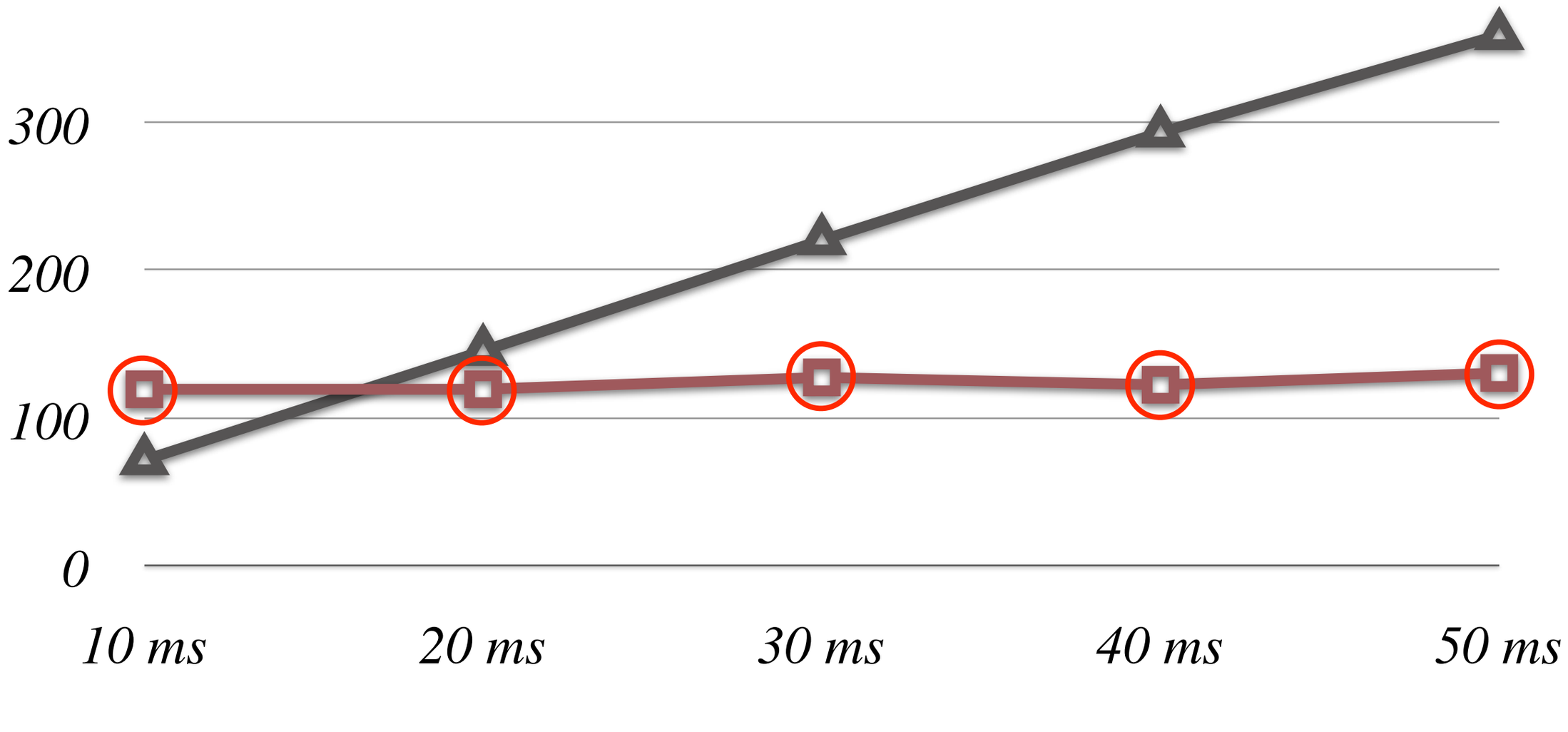
CONSERVATOIRE NATIONAL DES ARTS ET METIERS

# Performance and Scalability (4)



Tests exécutés sur un Palm Tungsten E2 avec une JVM : J9-571 (CLDC/MIDP-2)

Threads  SCL Activities  memory full



CONSERVATOIRE NATIONAL DES ARTS ET METIERS

# Analyse



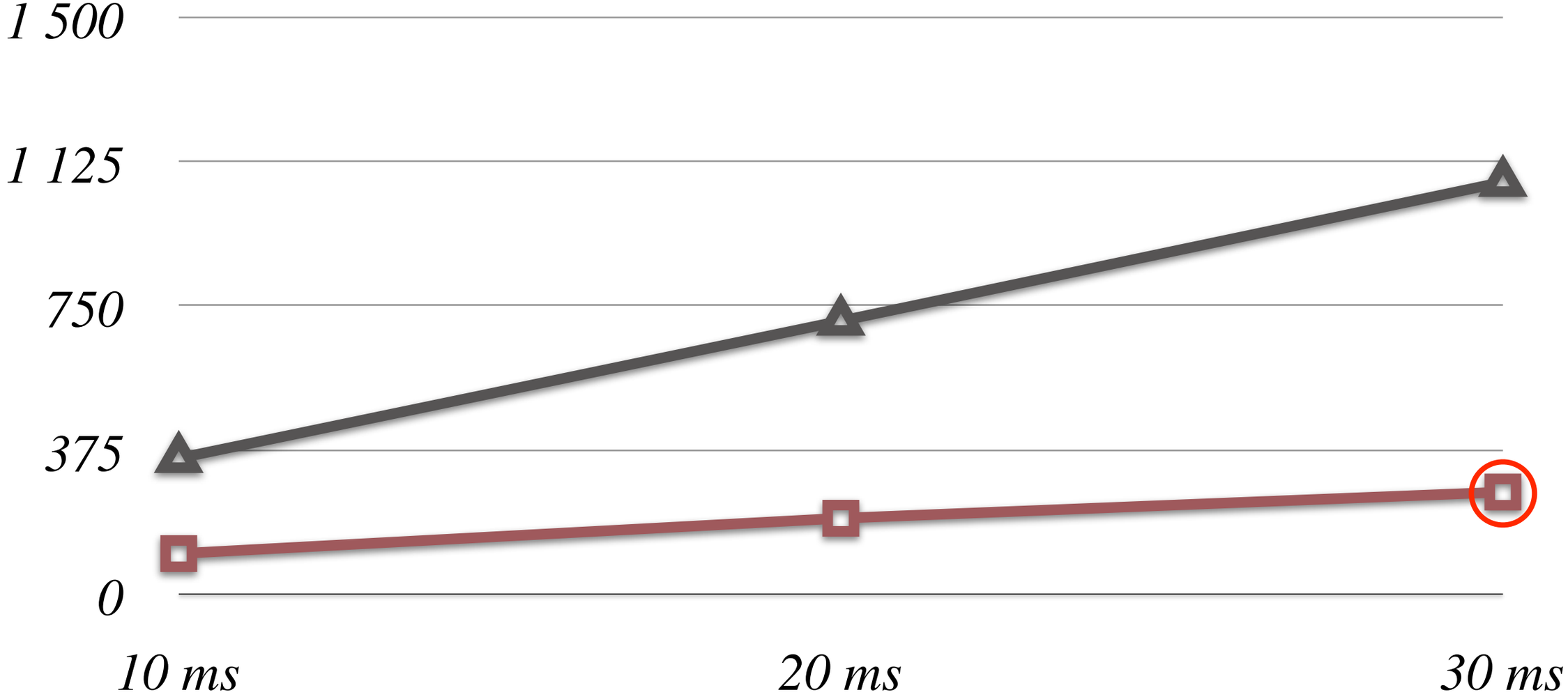
- Implantation sur IBM J9 :
  - Les threads sont traités en natif
  - Il n'y a pas de JIT et SCL (dérivant de SC) fait des choix d'implémentation favorisant l'utilisation d'un JIT
  - résultats encourageants pour ce qui est de l'occupation mémoire

# Performance and Scalability (5)



Tests exécutés sur un Axim 50v avec une JVM : J9-571 (CLDC/MIDP-2)

Threads  SCL Activities  memory full



CONSERVATOIRE NATIONAL DES ARTS ET METIERS

# Analyse



- Implantation sur IBM J9 sur un PDA haut de gamme :
  - Les threads sont toujours traitées en natif
  - Il n'y a toujours pas de JIT
  - résultats similaires à ceux sur station de travail

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

# Analyse



- Implantation sur IBM J9 sur un PDA haut de gamme :
  - Les threads sont toujours traitées en natif
  - Il n'y a toujours pas de JIT
  - résultats similaires à ceux sur station de travail

→ **La même VM supporte le profil CDC**

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

# Plan

- Les SugarCubes au dessus de J2ME
- Quelques résultats expérimentaux
- Les SugarCubes et le Multi-clocks
- Conclusion et perspectives

# Application au jeux



- Pas de mise en œuvre des threads dans le moteur du jeu (modèle des threads inapproprié) ; existent malgré tout (graphisme, GC, ...)
- Utilisation dans le cas de tâches asynchrones longues : chargement et écriture de données (à travers le réseau, sur un “FS”), sons, gestion des entrées/sorties...
- Une thread animation :

```
while(running){
    gameUpdate();
    gameRender();
    paintScreen();
    ...
}
```

# Le modèle multi-horloges



- Fournir un moyen d'intégrer les mondes synchrones et asynchrones :
  - faciliter la description d'activités rythmées par des horloges différentes et indépendantes (en particulier adresser les difficultés une horloge de base commune ≠ du cas synchrone)
  - fournir un mécanisme pour gérer l'asynchronie naturelle du système, par exemple pour les systèmes distribués
  - fournir quelques propriétés synchrones au dessus d'une communication asynchrone (donner un sens précis à la simultanéité ou à l'absence)
  - fournir un éventail de mécanismes de communication entre asynchrone et synchrone. Objectif : éviter que le composant le ou les plus lents ne pénalisent complètement le système

# Le modèle multi-horloges

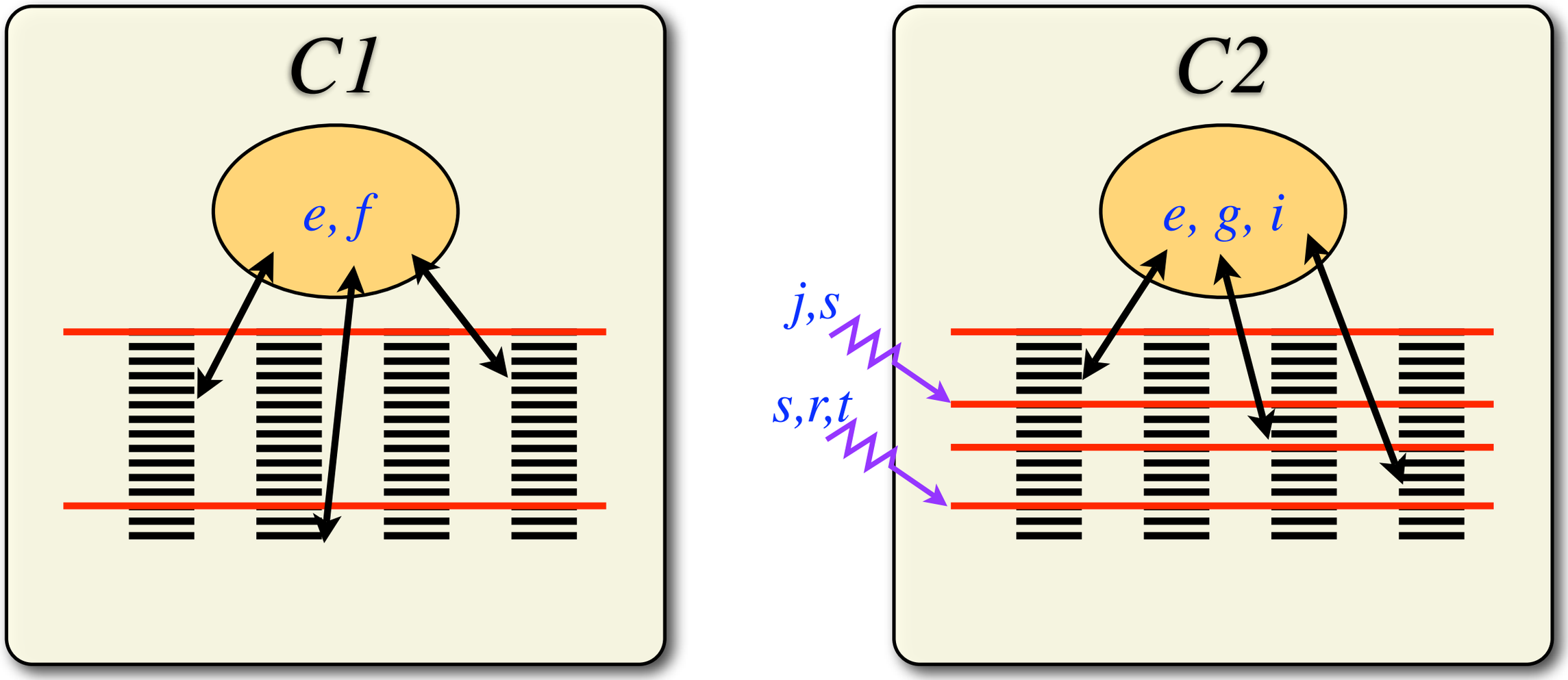


- L'instruction clock :
  - définit une séquence d'instants. Elle contient un programme qui est exécuté au rythme de cette horloge
  - Une horloge fournit un environnement d'événements où l'absence et la présence des événements ne peuvent être définies que par référence à cette horloge

    ↳ Le statut et les valeurs d'un événement ne peuvent être lus que par un programme qui appartient à cette horloge
- Les événements peuvent être générés en cours d'instant par le programme de l'horloge ou traité comme des entrées (prises en compte au début de l'instant) générés par l'environnement ou encore de façon asynchrone par des programmes réactifs exécutés sur d'autres horloges

# Le modèle multi-horloges





# Le modèle multi-horloges

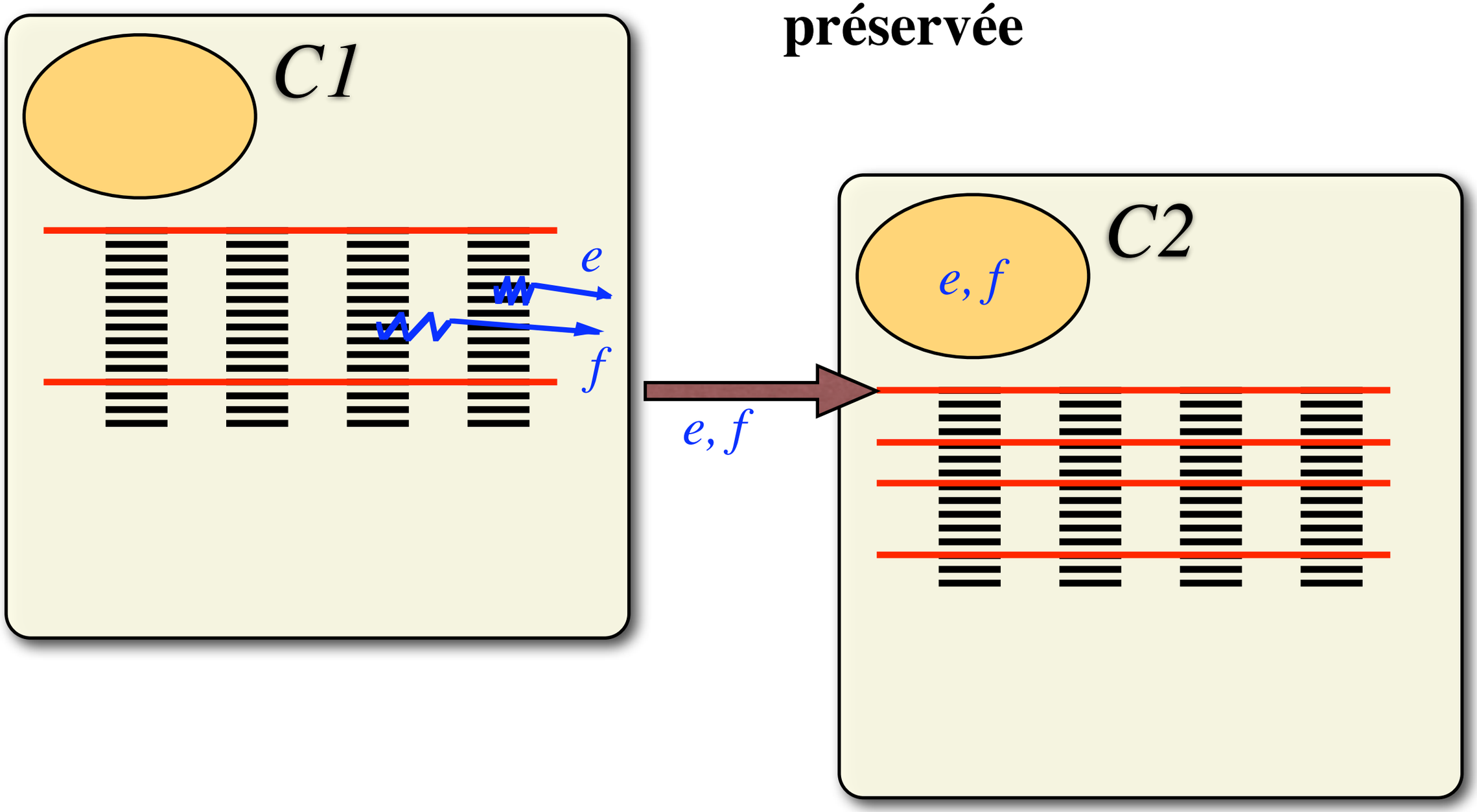


- L'instruction emit :
  - permet à un composant réactif exécuté sur une horloge c1 d'émettre un événement asynchrone ciblant une horloge c2 (les émissions sont effectuées à la fin de l'instant courant de c1)
  - Les événements émis sur une horloge c2 sont traités comme des entrées du prochain instant à exécuter de l'horloge c2 (les composants réactifs de l'horloge c2 peuvent alors réagir à cet événement)
  - Le composant émetteur voit son exécution suspendue jusqu'à ce que l'événement émis soit pris en compte par l'horloge c2
  - **Propriété importante** : si 2 événements sont émis simultanément à partir de l'horloge c1 ils seront vus comme étant simultanés par les composants réactifs de l'horloge c2

# Le modèle multi-horloges



**Simultanéité préservée**



# Le modèle multi-horloges



- Applications envisagées :
  - simulations distribués et jeux multi-joueurs : implantation sous forme de zones localement synchrone distribuées sur les différents clients ; besoin de synchronisation relativement fort (synchroniser l'ensemble du système n'est pas envisageable/recommandé)

    Exemple : faciliter la re-synchronisation entre des entités répliquées d'une application distribuée

# Les timers



- Permettent d'exécuter de tâches de manière cyclique à échéance d'un compte à rebours
- `schedule()` pour une tâche déclenché à un moment donné
- `scheduleAtFixedRate()` pour une tâche déclenchée périodiquement possibilité de vérifier les échéances : `scheduledExecutionTime()`
- Tâches définies par la classe `TimerTask`
- Pas suffisamment précis en pratique (gigue), pas de contrôle sur la période

CONSERVATOIRE NATIONAL DES ARTS ET METIERS

# Plan

- Les SugarCubes au dessus de J2ME
- Quelques résultats expérimentaux
- Les SugarCubes et le Multi-clocks
- Conclusion et perspectives

# Conclusion



- SugarCubes Lite est une première implantation de l'approche réactive au dessus de J2ME
- SCL offre de bonnes performances (mais nécessite encore des travaux importants de reengineering pour tourner efficacement sur une KVM)
- Le modèle multi-horloges est une piste pour réconcilier le monde synchrone et asynchrone permettant ainsi de prendre en compte des aspects différents d'une application interactive dans une approche unifiée

# Travaux futurs



- Diminuer l'emprunte mémoire du framework (et dans une moindre mesure sa consommation mémoire à l'exécution)
- Adapter l'implantation des SugarCubes Lite au mode interprété de la KVM pour améliorer les performances
- Étudier d'autres moyens de communication  entre les systèmes : "Refresh Events" afin d'éliminer silencieusement des informations devenues obsolètes au cours d'un échange entre 2 horloges différentes
- Étudier la notion d'horloge distribuées permettant à des composants réactifs distribués de partager une forte synchronisation bien qu'étant exécutés sur des horloges de base différentes

# Travaux futurs



- Intégration sur une plate-forme temps réel Java afin de bénéficier de services temporels avancés pour implanter des systèmes temps réels
- Décliner différentes solutions de tolérance aux pannes