

Soutenance de thèse

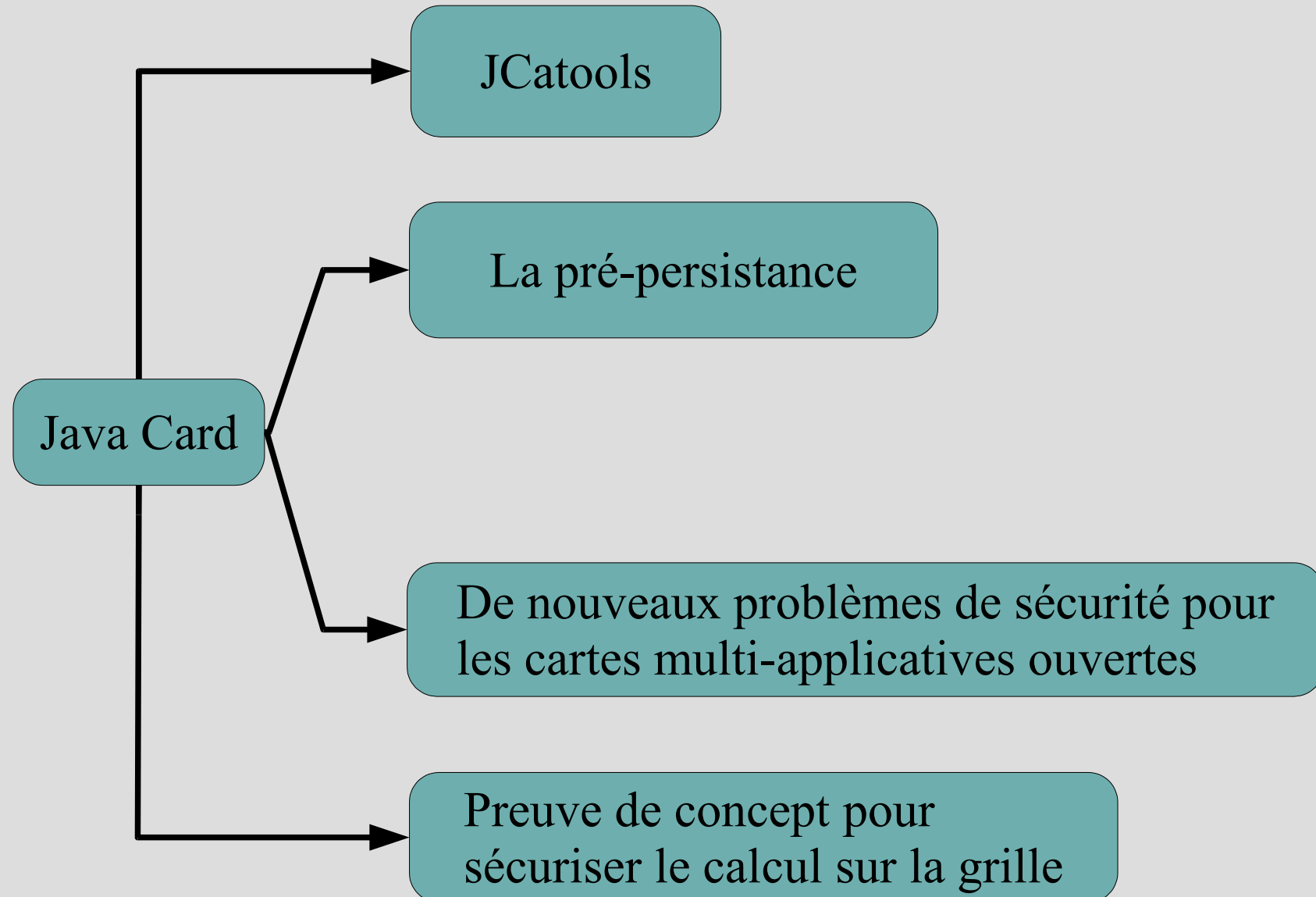
Étude et réalisation
d'un environnement
d'expérimentation et de modélisation
pour la technologie Java Card.
Application à la sécurité

Damien Sauveron

sauveron@labri.fr

Réalisée sous la direction de Serge Chaumette

Plan



Introduction

La carte à puce

La mono-application

La multi-application

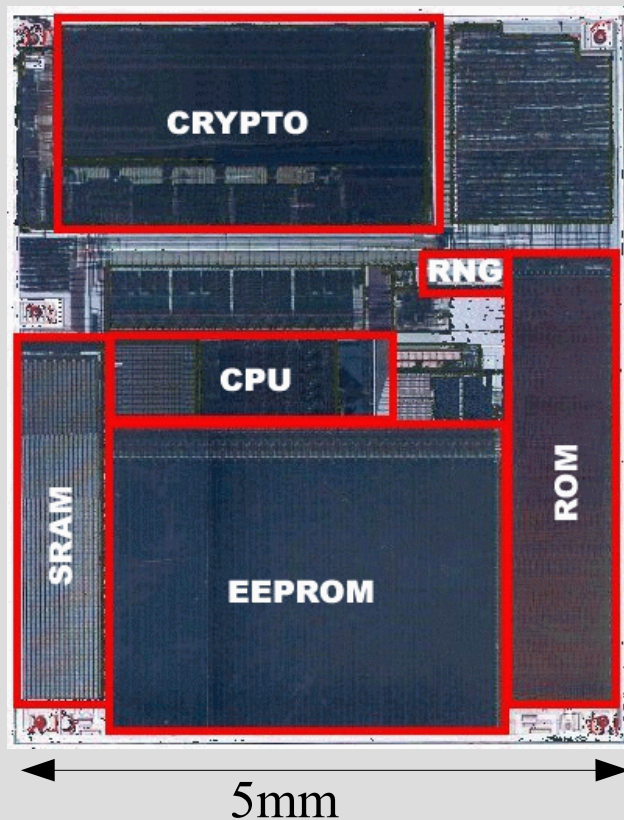
La technologie Java Card

Architecture

La persistance

Atomicité et transactions

La carte à puce : un périphérique de nature persistante



CPU :

CISC ou RISC
8, 16 ou 32 bits

Communications :

Contact
Contactless (RF)

Mémoires :

Permanente : ROM

- 32 à 64 Ko
- Card Operating System + données permanentes

Volatile : RAM

- 1 à 2 Ko
- très rapide
- *mémoire de travail* (pile d'appels, etc.)

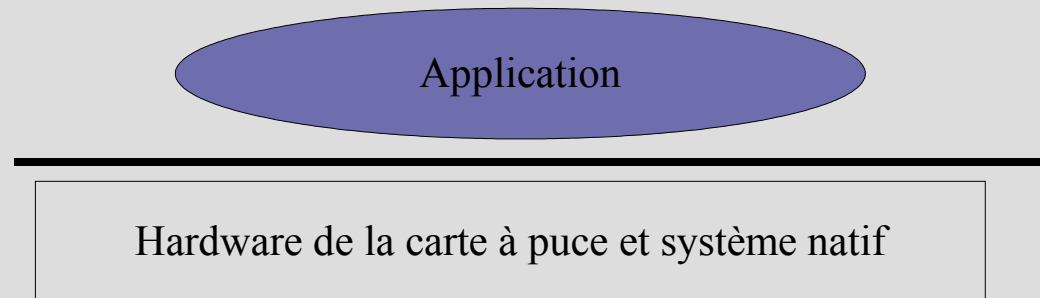
Persistante : EEPROM (ou FeRAM ou Flash)

- 24 à 64 Ko
- lente (200 fois plus que la RAM)
- *mémoire de stockage*
- durée de vie limitée à 100 000 cycles d'écriture

La mono-application

Historiquement :

- 1 application par carte
- Peu de problèmes de sécurité



Types d'applications :

- Bancaire (carte bleue, etc.)
- Santé (carte vitale, etc.)
- Télécommunication (télécarte, SIM, carte pré-payée)
- Transport (carte sans contact dans les transports en commun, etc.)
- Contrôle d'accès (carte sans contact pour l'accès aux bâtiments, etc.)
- Audiovisuel (télévision à péage, etc.)

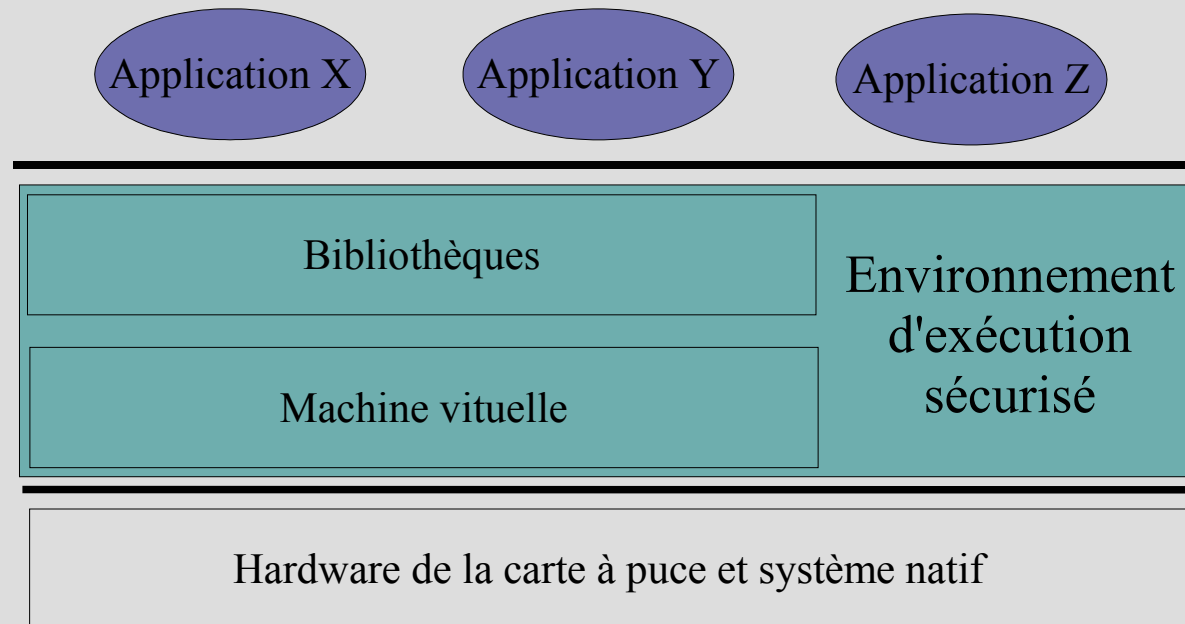
Nouveaux besoins => nouvelles applications :

- L'authentification (e.g. à des sites sur l'Internet)
- Les applications de fidélisation (e.g. l'accumulation de miles gratuits de voyages en avion)
- Les e-services (le e-commerce, la banque distante, le e-courrier, le télétravail, etc.)

La multi-application

Objectifs :

- Faire coexister plusieurs applications sur une même carte
- Coopération sécurisée
- Chargement dynamique après émission de la carte
- Réduction des coûts



Technologies multi-applicatives :

- MULTOS (1997)
- Windows for SmartCard (1998 - 2000)
- ZeitControl BasicCard (2004)
- Smartcard.NET (2003)
- Java Card (1996)

La technologie Java Card

Objectifs :

- Faire fonctionner des applications écrites en langage Java (les **applets**) sur des périphériques à mémoire limitée.
- Chargement dynamique.
- Garantir la sécurité de la plate-forme et des applications.

Cibles :

- Cartes à puce
- IButtons

Avantages :

- La facilité de développement des applications :
 - Orientée Objets
 - IDE
 - APIs ouvertes et de haut niveau d'abstraction

La sécurité :

- différents niveaux de contrôle d'accès
- un typage fort
- un pare-feu

La portabilité (*Write Once, Run Anywhere*).

La capacité de stockage et de gestion de plusieurs applications.

La compatibilité avec les standards existant pour les cartes à puce.

Architecture de Java Card

Difficultés pour adapter Java aux cartes :

- Faible capacité mémoire
- Périphérique de nature persistante

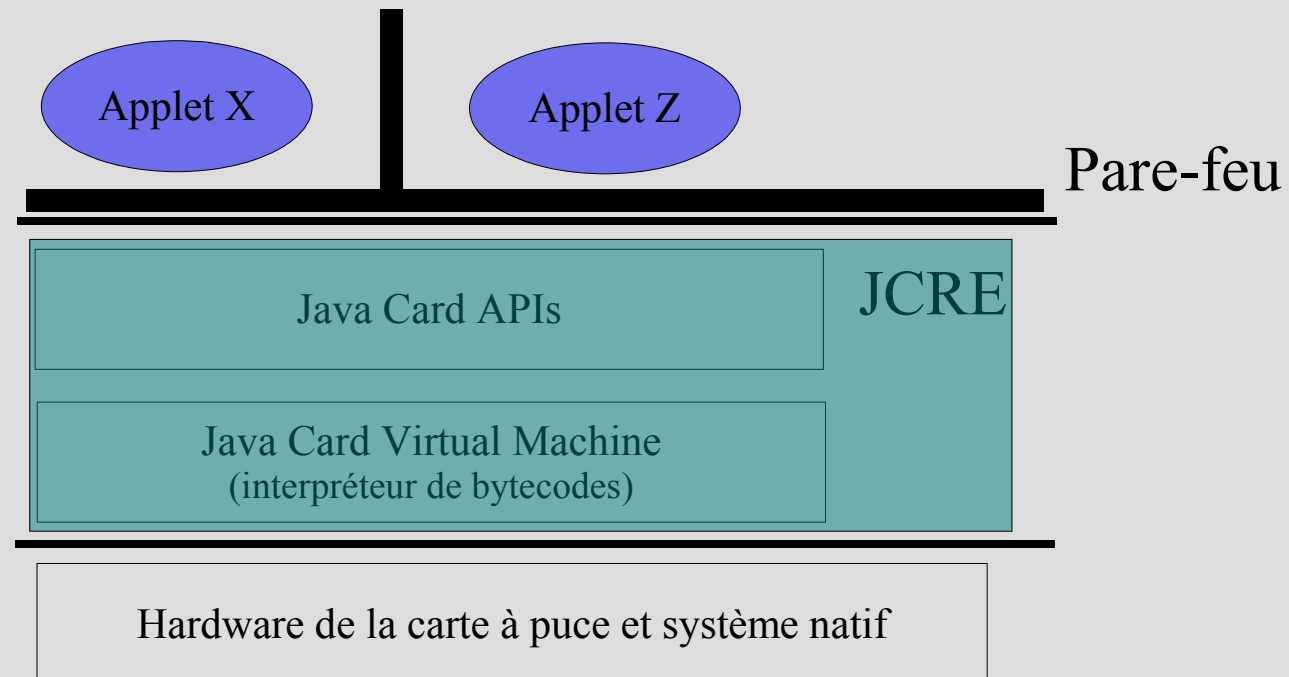
Solutions :

Sous-ensemble de Java :

- Limitation au strict nécessaire sur la carte (types simples, tableaux de types simples, ...)
- Pas de vérifieur embarqué => Ajout d'un pare-feu

Nouveau format pour l'interopérabilité binaire : le fichier CAP (*Converted APplet*)

Introduction du concept de persistance



Persistence

Un objet persistant est un objet stocké en mémoire persistante : entête + champs.

Quelques caractéristiques :

- il a été créé par le bytecode `new` avant de devenir persistant ;
- il conserve ses valeurs au travers des sessions avec le lecteur.

Atomicité et transactions

Objectif :

Garantir l'intégrité des données lors de la mise à jour des objets persistants.

Protections contre la perte d'alimentation, etc.

Atomicité :

Une opération est atomique si elle est soit totalement exécutée soit pas du tout.

JCVM : la mise à jour des champs **des objets persistants** est atomique

Transactions :

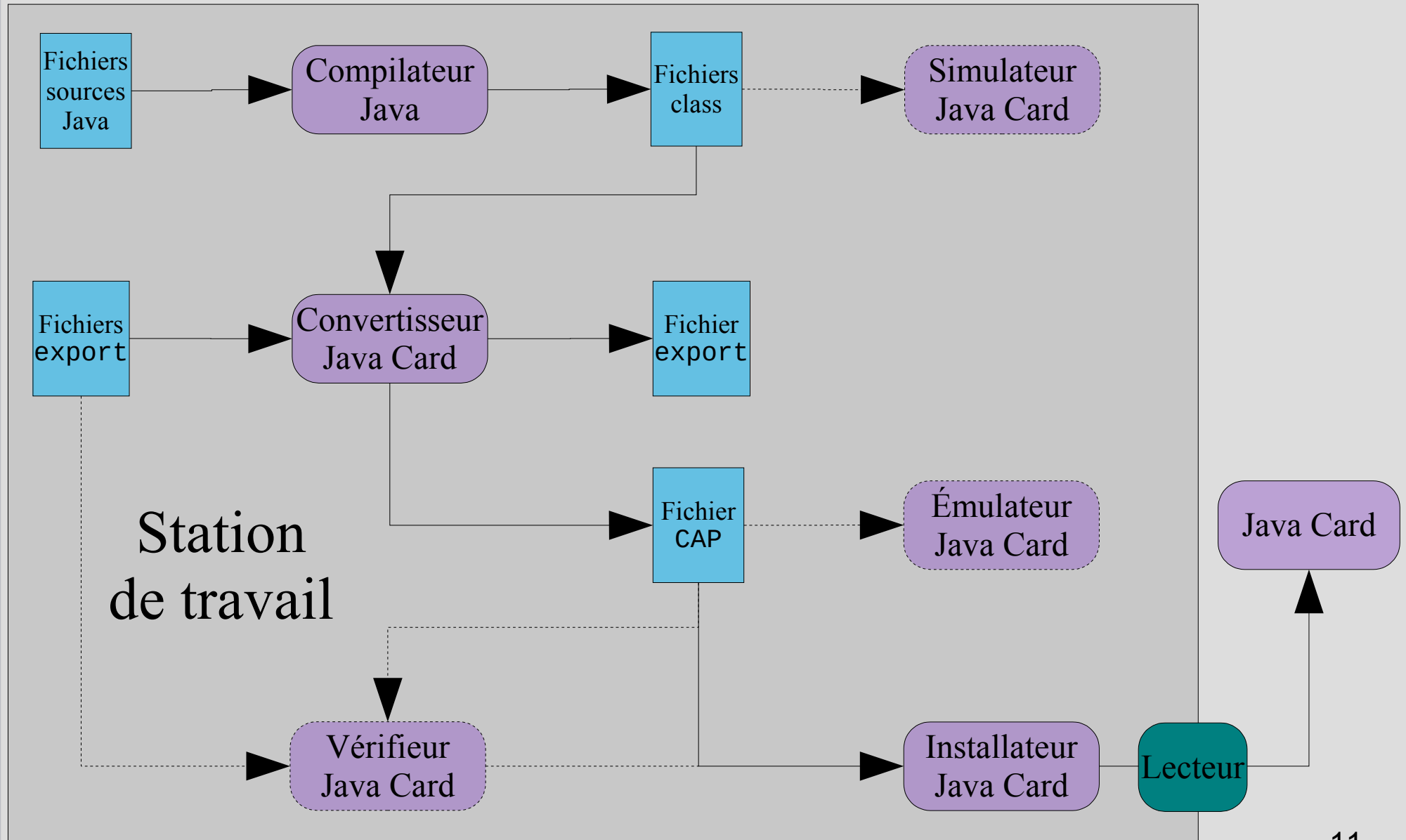
Une transaction représente un bloc d'opérations qui doit être effectué complètement ou pas du tout.

- `JCSystem.beginTransaction`
- `JCSystem.commitTransaction`
- `JCSystem.abortTransaction`

Restauration des anciennes données => la sauvegarde dans un *buffer* de transaction (dépendant de l'implantation)

Write once, run anywhere ?

Cycle de développement d'une applet

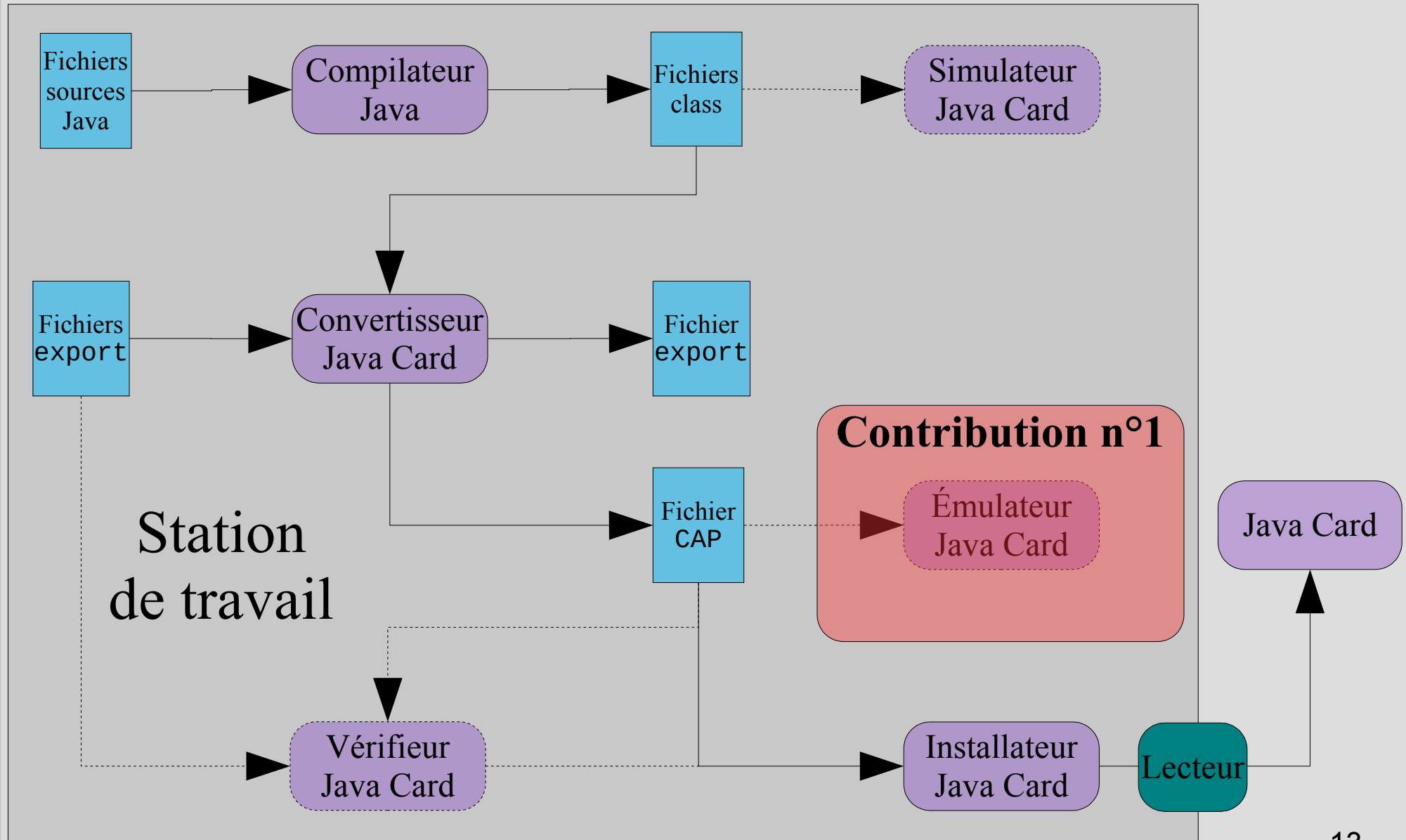


Les JCatools

JCat Emulator

JCat View

Contribution n°1 : l'émulateur Java Card



Les JCatools

Projet PROGSI « Sécurité Java Card » financé par le MINEFI entre le LaBRI et SERMA Technologies

Objectifs de SERMA Technologies

- Connaissance détaillée de la technologie Java Card
- Définition d'une méthodologie d'évaluation de plate-forme
- Création d'une batterie de tests d'attaques
- Environnement d'expérimentation de tests d'attaques logicielles & matérielles

Objectifs du LaBRI

- Se confronter à une vraie problématique industrielle
- Faire profiter de son savoir-faire Java
- Obtenir des outils modulaires et extensibles

JCatools : un environnement pour l'étude de plates-formes et d'applets Java Card

Ensemble outils :

- JCat Emulator
- JCat View,
- JCat Converter
- Utilitaires voués à simplifier les opérations de modification de fichiers CAP.

JCat Emulator

Implantation complète de Java Card 2.1.1 :

- VM
- JCRE
- APIs (sauf les APIs cryptographiques)

Il inclut :

- Le pare-feu
- L'atomicité et le mécanisme de transactions
- La lecture du format de fichiers CAP

Seul émulateur Java Card *open source*, mais il ne peut être distribué (discussions avec Sun microsystems) !

Fonctionnalités :

Débogage du *bytecode* : pas à pas + visualisation des états (mémoires, objets, pile d'appels, buffer de transaction)

Statistiques sur les bytecodes exécutés.

Simulation d'attaques physiques : arrachage de carte, attaque par rayonnement, etc.

Instanciation parallèle de Java Cards

JCAT

File Options Help

431 return

Java Card 0 Java Card 1 Java Card 2 Java Card 3

Loaded Package

0xa0	0x00	0x00	0x00	0x62	0x00	0x01
0x00	0x00	0x00	0x00	0x00	0x00	
0x00	0x00	0x00	0x00	0x00	0x00	0x01
0xa0	0x00	0x00	0x00	0x62	0x01	0x01
0x00	0x11	0x22	0x33	0x44		

Frames

Depth: 0

Owner: 0x0, null

PCReg: 0x682

Stack:

Local: 0x0 0x0 0x0

Transaction

In progress: ●

index: 0

0x0	0x1	0x0	0x1d	0x0	0x1	0x6
0xfd	0xfd					

eeeprom

- fields = 0x0
- fields = 0x0
- fields = 0x0 0x0
- fields = 0x0
- fields = 0x0
- fields = 0x0
- fields = 0xc
- ram ref: 0x1
- fields = 0x0 0x0 0x0 0x0
- fields = 0xf 0x0 0x0
- 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

ram

- ram
- 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00

reference 0xc

```

/* 416, 0x030 */ return , Operand Stack : [ ]
/* 417, 0x7e7 */ getstatic_a , Operand Stack : [ 0xb ]
/* 418, 0x7ea */ invokevirtual , 0x0, null, Operand Stack : [ ]
/* 419, 0x1bf */ getfield_a_this, Operand Stack : [ 0xc ]
/* 420, 0x1c1 */ areturn , Operand Stack : [ 0xc ]
/* 421, 0x7ed */ putstatic_a , static fields = 0x0 0x4 0x0 0x5 0x0 0x6 0x0 0x3 0x0 0xd 0x0 0
/* 422, 0x7f0 */ getstatic_a , Operand Stack : [ 0xc ]
/* 423, 0x7f3 */ invokestatic , 0x0, null, Operand Stack : [ ]
/* 424, 0x033 */ invokestatic , 0x0, null, Operand Stack : [ ]
/* 425, 0x0bc */ impdep1 , [0x18]
/* 426, 0x036 */ return , Operand Stack : [ ]
/* 427, 0x7f6 */ sconst_1 , Operand Stack : [ 0x1 ]
/* 428, 0x7f7 */ putstatic_b , static fields = 0x0 0x4 0x0 0x5 0x0 0x6 0x0 0x3 0x0 0xd 0x0 0
/* 429, 0x7fa */ return , Operand Stack : [ ]
/* 430, 0x682 */ invokestatic , 0x0, null, Operand Stack : [ ]
/* 431, 0x701 */ invokestatic
TEARING TEARING TEARING TEARING TEARING TEARING TEARING
, 0x0, null
/* 431, 0x67a */ getstatic_b , Operand Stack : [ 0x1 ]
/* 432, 0x67d */ ifne , Operand Stack : [ ]
/* 433, 0x682 */ invokestatic

```

L'appui sur le bouton provoque un arrachage (TEARING) et donc la carte redémarre du début

JCat Emulator : représentation des objets

Object view

Properties Values

Reference 0x1

Owner

PackageAID : null

Applet Owner : 0x0

Attributes

Special attributes : NONE

Transient type : NOT_A_TRANSIENT_OBJECT

Type

Static Fields: byte [30]

Object view

Properties Values

[0]	0x0
[1]	0x4
[2]	0x0
[3]	0x0A
[4]	0x0
[5]	0x6
[6]	0x0
[7]	0x3
[8]	0x0
[9]	0xd
[10]	0x0
[11]	0x9
[12]	0x0

Object view

Properties Values

Reference 0x1f

Owner

PackageAID : null

Applet Owner : 0x0

Attributes

Special attributes : PERMANENT_ENTRY_POINT

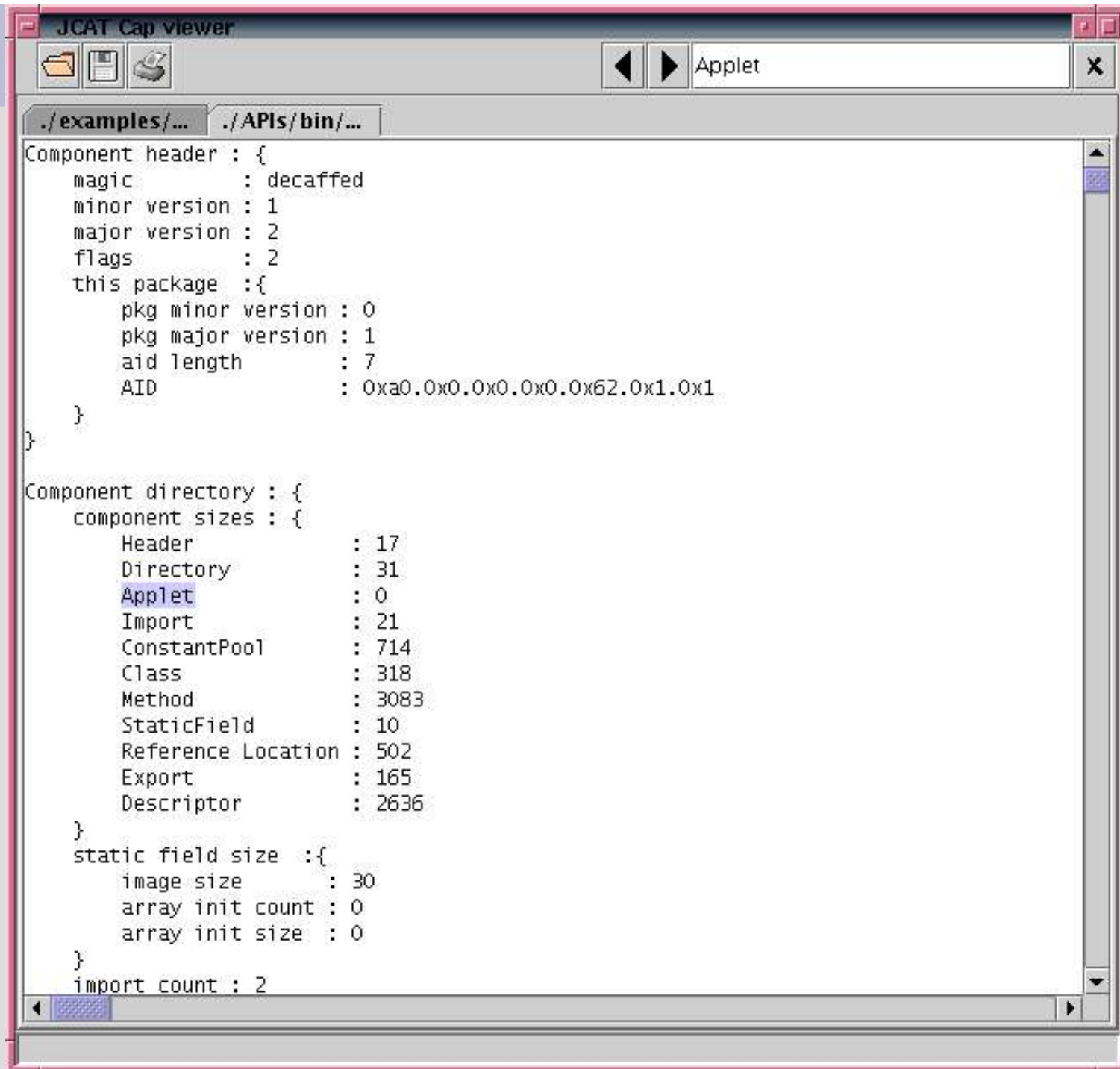
Transient type : NOT_A_TRANSIENT_OBJECT

Type

```
flags : 0
interface count : 0
super classRef : external class 0x0 of package 0x0
declared instance size : 1
first reference token : 0
reference count : 1
public method table base : 0
public method table count : 5
package method table base : 0
package method table count : 0
public virtual method table : { 0x129 0x193 0x159 0x116 0x176 }
package virtual method table : { }
implemented interfaces : { }
```

Inconvénient : Interface parfois un peu rebutante

JCat View



Introduction du concept de pré-persistance

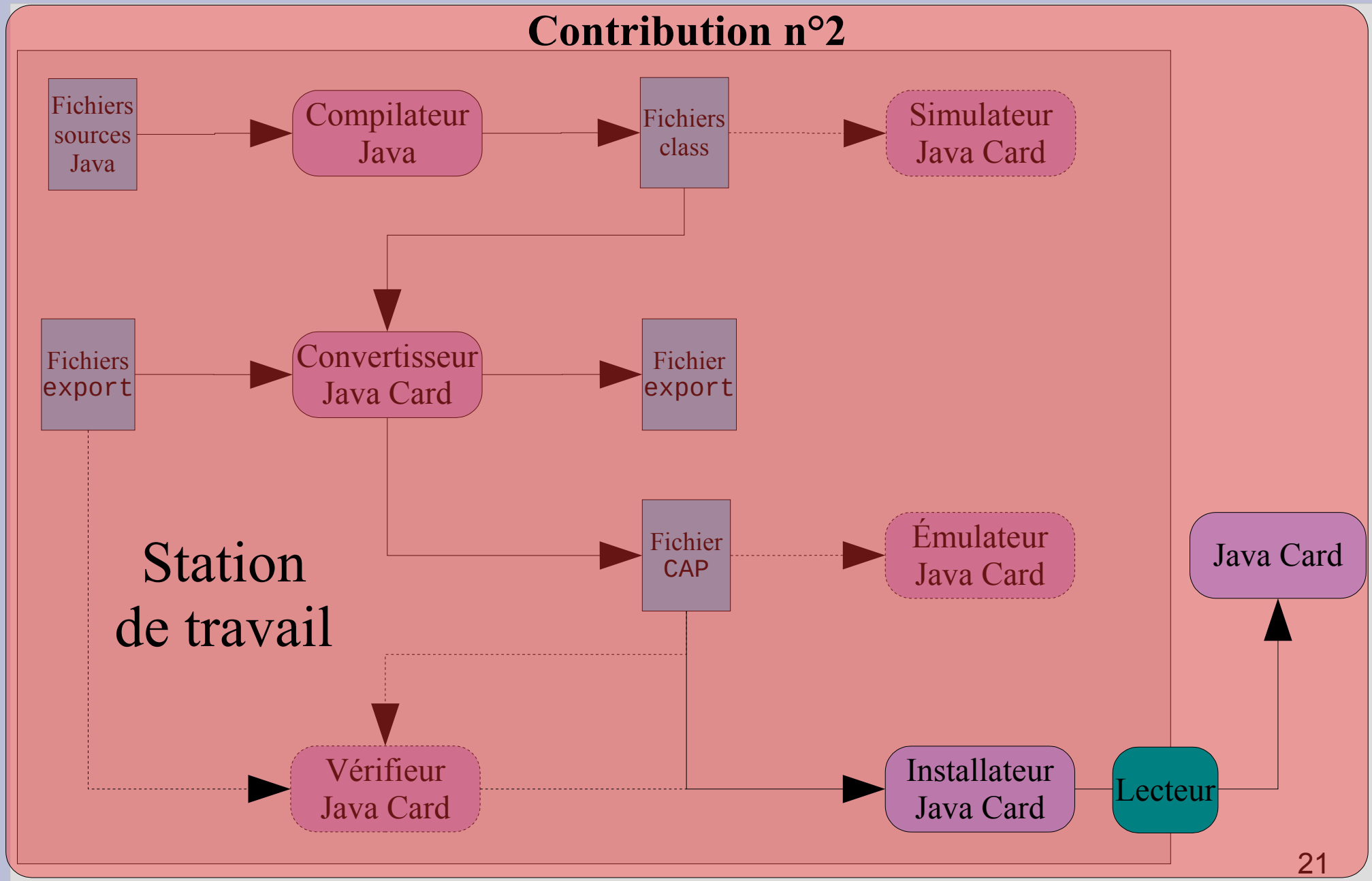
Les ambiguïtés des spécifications Java Card

Exemple de code en Java

Exemple de code en Java Card **sans** la pré-persistance

Exemple de code en Java Card **avec** la pré-persistance

Contribution n°2 : les spécifications Java Card



Introduction du concept de pré-persistence

D'après les spécifications :

- un objet est créé lors de l'exécution d'un bytecode `new`, `anewarray` et `newarray` ;

*- le développeur **rendra** les objets persistants lorsqu'une référence à l'objet est stockée dans un champ d'un objet persistant ou dans celui d'une classe.*

Ambiguïté : nature des objets **entre** le moment de **leur création** et celui de **leur affectation** à un champ d'un objet persistant ?

Nous désignerons par espace **pré-persistant** l'espace où sont placés les objets lors de leur création (*i.e.* `new`, `anewarray` et `newarray`).

Sa localisation reste à définir ; nous avons considéré plusieurs possibilités ; **dans la suite de cet exposé** nous ne considérerons **en RAM**

Attention : **pré-persistant** n'implique pas que l'objet deviendra nécessairement persistant.

Les fabricants de Java Cards ont choisi de rendre les objets persistants à **la création**.

Exemple de code en Java

```
public class A {  
  
    private short s1;  
    private Object o1;  
  
    public void foo() {  
        s1 = (short) 0;  
        o1 = new Object();  
    }  
  
    public void bar() {  
        short s2 = (short) 1;  
        Object o2 = new Object();  
    }  
}
```

Traduction en bytecodes des méthodes foo et bar

```
void foo() { ----- ICI
```

```
    aload_0  
    sconst_0  
    putfield_s s1  
    aload_0  
    new Object  
    putfield_a o1  
}
```

```
void bar() {  
    sconst_1  
    sstore s2  
    new Object  
    astore o2  
}
```

Instance de A
s1 =
o1 =

RAM

Pile de frames

var. locales

pile d'opérandes

Exemple de code en Java

```
public class A {  
  
    private short s1;  
    private Object o1;  
  
    public void foo() {  
        s1 = (short) 0;  
        o1 = new Object();  
    }  
  
    public void bar() {  
        short s2 = (short) 1;  
        Object o2 = new Object();  
    }  
}
```

Traduction en bytecodes des méthodes foo et bar

```
void foo() {  
    aload_0  
    sconst_0      <----- ICI  
    putfield_s s1  
    aload_0  
    new Object  
    putfield_a o1  
}  
  
void bar() {  
    sconst_1  
    sstore s2  
    new Object  
    astore o2  
}
```

Instance de A
s1 =
o1 =

RAM

Pile de frames

var. locales

pile d'opérandes

this
0x0000

Exemple de code en Java

```
public class A {  
  
    private short s1;  
    private Object o1;  
  
    public void foo() {  
        s1 = (short) 0;  
        o1 = new Object();  
    }  
  
    public void bar() {  
        short s2 = (short) 1;  
        Object o2 = new Object();  
    }  
}
```

Traduction en bytecodes des méthodes foo et bar

```
void foo() {  
    aload_0  
    sconst_0  
    putfield_s s1 <----- ICI  
    aload_0  
    new Object  
    putfield_a o1  
}  
  
void bar() {  
    sconst_1  
    sstore s2  
    new Object  
    astore o2  
}
```

Instance de A
s1 = 0x0000
o1 =

RAM

Pile de frames

var. locales

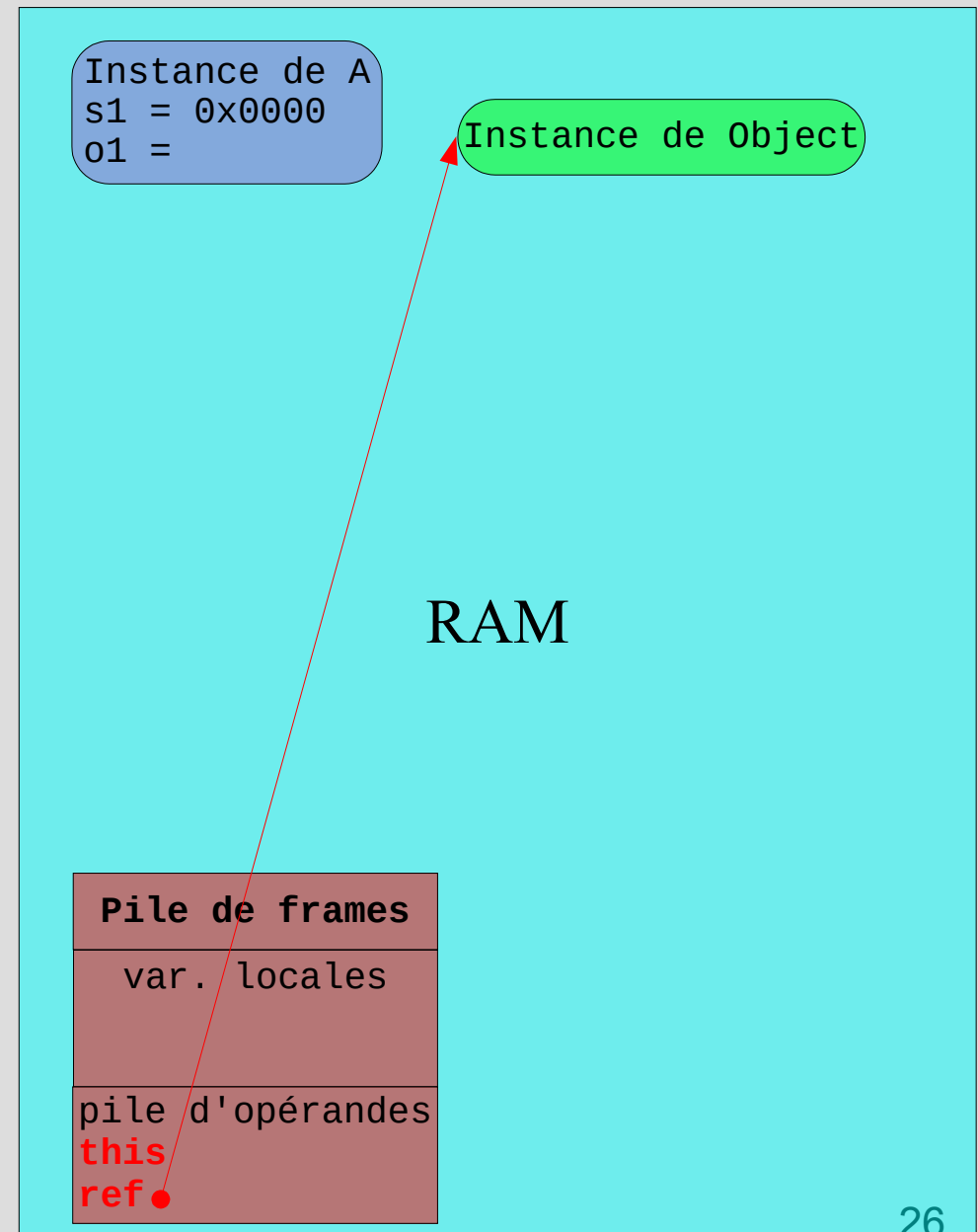
pile d'opérandes

Exemple de code en Java

```
public class A {  
  
    private short s1;  
    private Object o1;  
  
    public void foo() {  
        s1 = (short) 0;  
        o1 = new Object();  
    }  
  
    public void bar() {  
        short s2 = (short) 1;  
        Object o2 = new Object();  
    }  
}
```

Traduction en bytecodes des méthodes foo et bar

```
void foo() {  
    aload_0  
    sconst_0  
    putfield_s s1  
    aload_0  
    new Object    <----- ICI  
    putfield_a o1  
}  
  
void bar() {  
    sconst_1  
    sstore s2  
    new Object  
    astore o2  
}
```



Exemple de code en Java

```
public class A {  
  
    private short s1;  
    private Object o1;  
  
    public void foo() {  
        s1 = (short) 0;  
        o1 = new Object();  
    }  
  
    public void bar() {  
        short s2 = (short) 1;  
        Object o2 = new Object();  
    }  
}
```

Traduction en bytecodes des méthodes foo et bar

```
void foo() {  
    aload_0  
    sconst_0  
    putfield_s s1  
    aload_0  
    new Object  
    putfield_a o1 <----- ICI  
}  
  
void bar() {  
    sconst_1  
    sstore s2  
    new Object  
    astore o2  
}
```

Instance de A
s1 = 0x0000
o1 = ●

Instance de Object

RAM

Pile de frames

var. locales

pile d'opérandes

Exemple de code en Java

```
public class A {  
  
    private short s1;  
    private Object o1;  
  
    public void foo() {  
        s1 = (short) 0;  
        o1 = new Object();  
    }  
  
    public void bar() {  
        short s2 = (short) 1;  
        Object o2 = new Object();  
    }  
}
```

Traduction en bytecodes des méthodes foo et bar

```
void foo() {  
    aload_0  
    sconst_0  
    putfield_s s1  
    aload_0  
    new Object  
    putfield_a o1  
}
```

```
void bar() {  
    sconst_1  
    sstore s2  
    new Object  
    astore o2  
}
```

<----- ICI

Instance de A
s1 = 0x0000
o1 = ●

Instance de Object

RAM

Pile de frames

var. locales

s2 =

o2 =

pile d'opérandes

Exemple de code en Java

```
public class A {  
  
    private short s1;  
    private Object o1;  
  
    public void foo() {  
        s1 = (short) 0;  
        o1 = new Object();  
    }  
  
    public void bar() {  
        short s2 = (short) 1;  
        Object o2 = new Object();  
    }  
}
```

Traduction en bytecodes des méthodes foo et bar

```
void foo() {  
    aload_0  
    sconst_0  
    putfield_s s1  
    aload_0  
    new Object  
    putfield_a o1  
}
```

```
void bar() {  
    sconst_1  
    sstore s2  
    new Object  
    astore o2  
}
```

←----- ICI

Instance de A
s1 = 0x0000
o1 = ●

Instance de Object

RAM

Pile de frames

var. locales
s2 =
o2 =

pile d'opérandes
0x0001

Exemple de code en Java

```
public class A {  
  
    private short s1;  
    private Object o1;  
  
    public void foo() {  
        s1 = (short) 0;  
        o1 = new Object();  
    }  
  
    public void bar() {  
        short s2 = (short) 1;  
        Object o2 = new Object();  
    }  
}
```

Traduction en bytecodes des méthodes foo et bar

```
void foo() {  
    aload_0  
    sconst_0  
    putfield_s s1  
    aload_0  
    new Object  
    putfield_a o1  
}
```

```
void bar() {  
    sconst_1  
    sstore s2  
    new Object  
    astore o2  
}
```

----- ICI

Instance de A
s1 = 0x0000
o1 = ●

Instance de Object

RAM

Pile de frames

var. locales
s2 = 0x0001
o2 =

pile d'opérandes

Exemple de code en Java

```
public class A {  
  
    private short s1;  
    private Object o1;  
  
    public void foo() {  
        s1 = (short) 0;  
        o1 = new Object();  
    }  
  
    public void bar() {  
        short s2 = (short) 1;  
        Object o2 = new Object();  
    }  
}
```

Traduction en bytecodes des méthodes foo et bar

```
void foo() {  
    aload_0  
    sconst_0  
    putfield_s s1  
    aload_0  
    new Object  
    putfield_a o1  
}
```

```
void bar() {  
    sconst_1  
    sstore s2  
    new Object  
    astore o2  
}
```

<----- ICI

Instance de A
s1 = 0x0000
o1 = ●

Instance de Object

RAM

Pile de frames

var. locales
s2 = 0x0001
o2 =

pile d'opérandes
ref ●

Instance de Object

Exemple de code en Java

```
public class A {  
  
    private short s1;  
    private Object o1;  
  
    public void foo() {  
        s1 = (short) 0;  
        o1 = new Object();  
    }  
  
    public void bar() {  
        short s2 = (short) 1;  
        Object o2 = new Object();  
    }  
}
```

Traduction en bytecodes des méthodes foo et bar

```
void foo() {  
    aload_0  
    sconst_0  
    putfield_s s1  
    aload_0  
    new Object  
    putfield_a o1  
}
```

```
void bar() {  
    sconst_1  
    sstore s2  
    new Object  
    astore o2  
}
```

<----- ICI

Instance de A
s1 = 0x0000
o1 = ●

Instance de Object

RAM

Pile de frames

var. locales
s2 = 0x0001
o2 = ●

Instance de Object

pile d'opérandes

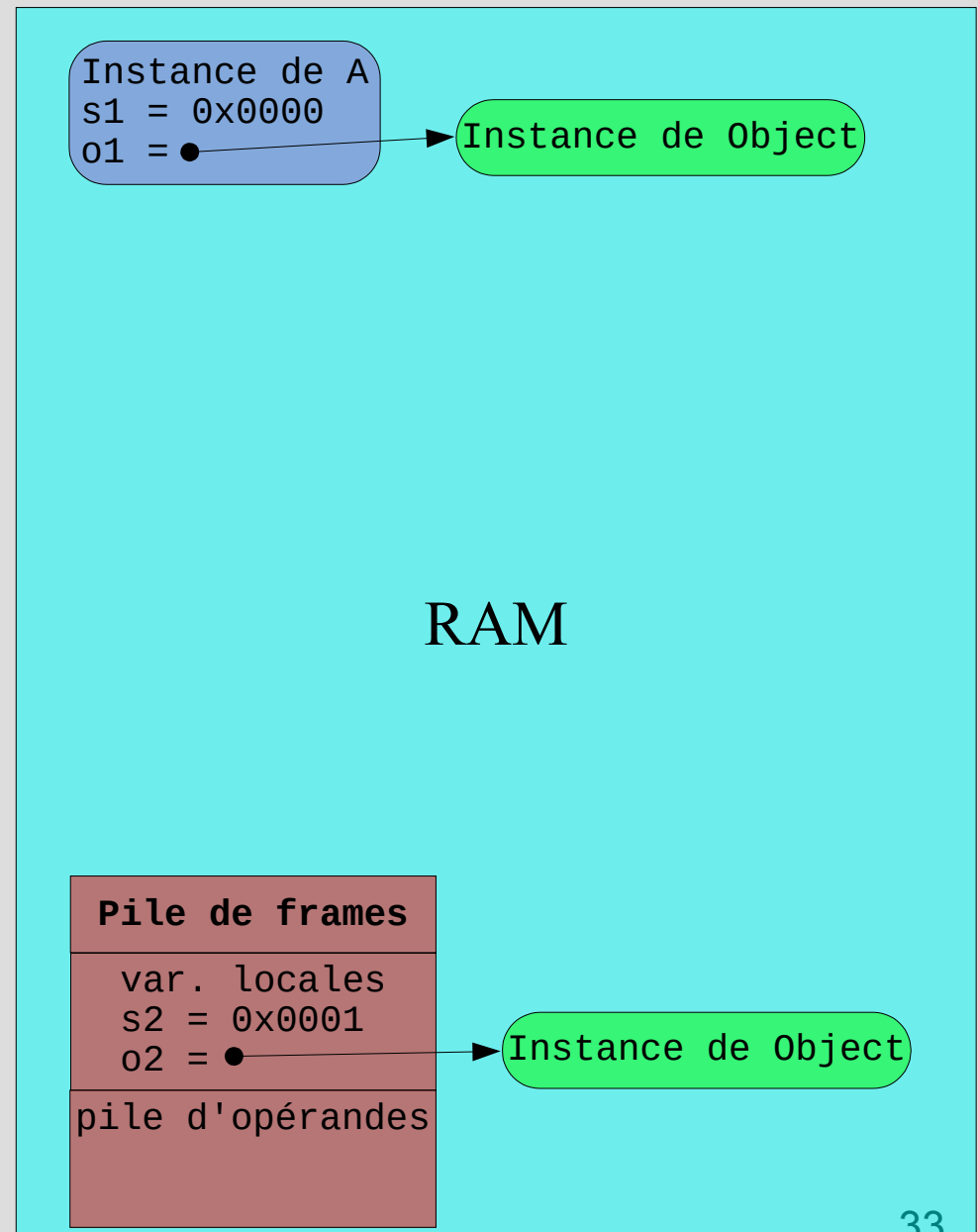
Exemple de code en Java

Remarques :

- sémantiques **identiques et homogènes** pour les variables locales aux méthodes et les variables d'instances (contenu et référence)
- Tous les objets sont dans une même mémoire **RAM**.

⇒ **Tout est temporaire !**

Normal, Java est un environnement de programmation temporaire.



Exemple de code en Java Card **sans** pré-persistance

```
public class A {  
  
    private short s1;  
    private Object o1;  
  
    public void foo() {  
        s1 = (short) 0;  
        o1 = new Object();  
    }  
  
    public void bar() {  
        short s2 = (short) 1;  
        Object o2 = new Object();  
    }  
}
```

Traduction en bytecodes des méthodes foo et bar

```
void foo() { <----- ICI
```

```
    aload_0  
    sconst_0  
    putfield_s s1  
    aload_0  
    new Object  
    putfield_a o1  
}
```

```
void bar() {  
    sconst_1  
    sstore s2  
    new Object  
    astore o2  
}
```

Instance de A
s1 =
o1 =

EEPROM

RAM

Pile de frames

var. locales

pile d'opérandes

Exemple de code en Java Card **sans** pré-persistence

```
public class A {  
  
    private short s1;  
    private Object o1;  
  
    public void foo() {  
        s1 = (short) 0;  
        o1 = new Object();  
    }  
  
    public void bar() {  
        short s2 = (short) 1;  
        Object o2 = new Object();  
    }  
}
```

Traduction en bytecodes des méthodes foo et bar

```
void foo() {  
    aload_0  
    sconst_0          <----- ICI  
    putfield_s s1  
    aload_0  
    new Object  
    putfield_a o1  
}  
  
void bar() {  
    sconst_1  
    sstore s2  
    new Object  
    astore o2  
}
```

Instance de A
s1 =
o1 =

EEPROM

RAM

Pile de frames

var. locales

pile d'opérandes

this
0x0000

Exemple de code en Java Card **sans** pré-persistence

```
public class A {  
  
    private short s1;  
    private Object o1;  
  
    public void foo() {  
        s1 = (short) 0;  
        o1 = new Object();  
    }  
  
    public void bar() {  
        short s2 = (short) 1;  
        Object o2 = new Object();  
    }  
}
```

Traduction en bytecodes des méthodes foo et bar

```
void foo() {  
    aload_0  
    sconst_0  
    putfield_s s1 <----- ICI  
    aload_0  
    new Object  
    putfield_a o1  
}  
  
void bar() {  
    sconst_1  
    sstore s2  
    new Object  
    astore o2  
}
```

Instance de A
s1 = 0x0000
o1 =

EEPROM

RAM

Pile de frames

var. locales

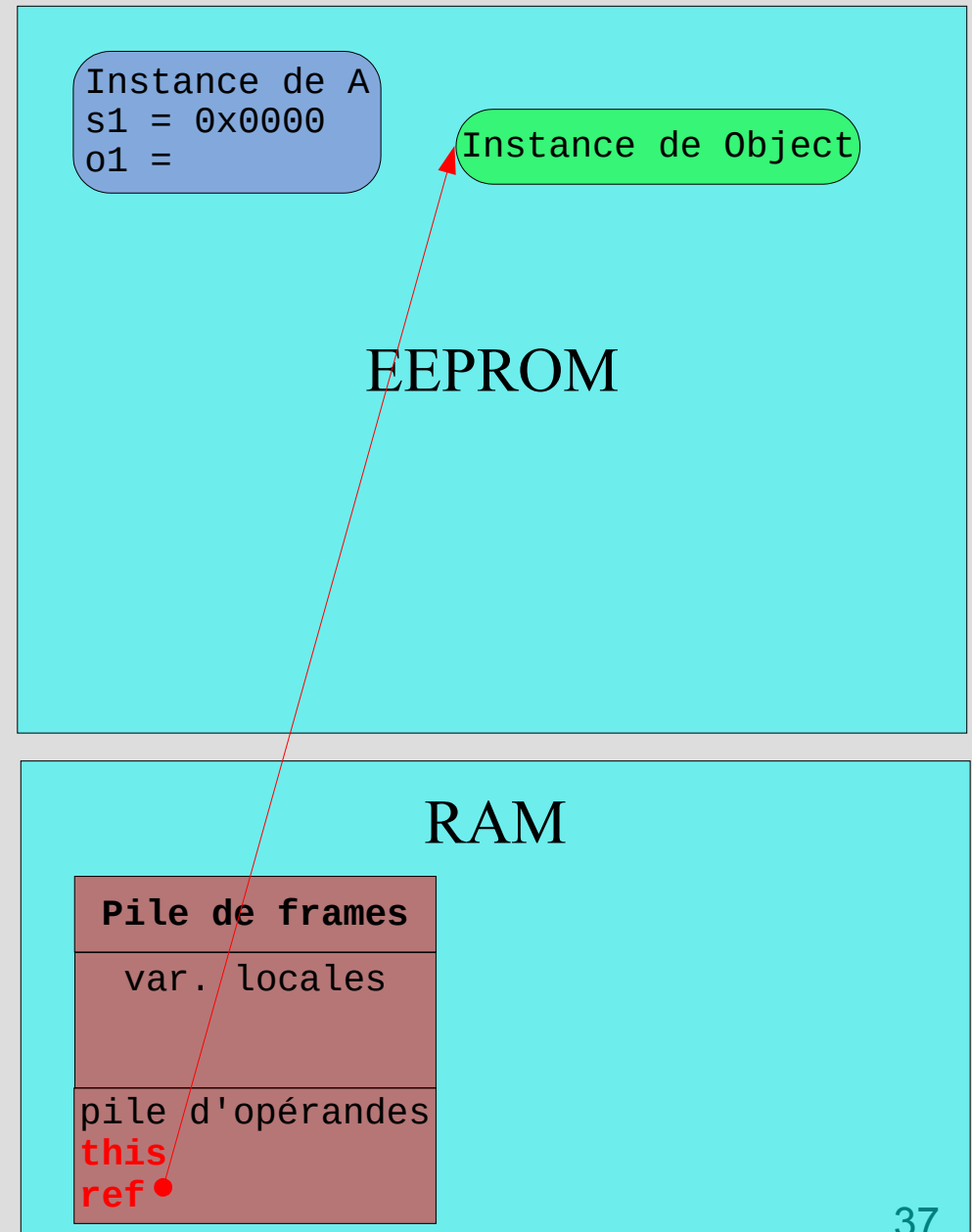
pile d'opérandes

Exemple de code en Java Card **sans** pré-persistence

```
public class A {  
  
    private short s1;  
    private Object o1;  
  
    public void foo() {  
        s1 = (short) 0;  
        o1 = new Object();  
    }  
  
    public void bar() {  
        short s2 = (short) 1;  
        Object o2 = new Object();  
    }  
}
```

Traduction en bytecodes des méthodes foo et bar

```
void foo() {  
    aload_0  
    sconst_0  
    putfield_s s1  
    aload_0  
    new Object    <----- ICI  
    putfield_a o1  
}  
  
void bar() {  
    sconst_1  
    sstore s2  
    new Object  
    astore o2  
}
```



Exemple de code en Java Card **sans** pré-persistence

```
public class A {  
  
    private short s1;  
    private Object o1;  
  
    public void foo() {  
        s1 = (short) 0;  
        o1 = new Object();  
    }  
  
    public void bar() {  
        short s2 = (short) 1;  
        Object o2 = new Object();  
    }  
}
```

Traduction en bytecodes des méthodes foo et bar

```
void foo() {  
    aload_0  
    sconst_0  
    putfield_s s1  
    aload_0  
    new Object  
    putfield_a o1 <----- ICI  
}  
  
void bar() {  
    sconst_1  
    sstore s2  
    new Object  
    astore o2  
}
```

Instance de A
s1 = 0x0000
o1 = ●

Instance de Object

EEPROM

RAM

Pile de frames

var. locales

pile d'opérandes

Exemple de code en Java Card **sans** pré-persistence

```
public class A {  
  
    private short s1;  
    private Object o1;  
  
    public void foo() {  
        s1 = (short) 0;  
        o1 = new Object();  
    }  
  
    public void bar() {  
        short s2 = (short) 1;  
        Object o2 = new Object();  
    }  
}
```

Traduction en bytecodes des méthodes foo et bar

```
void foo() {  
    aload_0  
    sconst_0  
    putfield_s s1  
    aload_0  
    new Object  
    putfield_a o1  
}
```

```
void bar() { <----- ICI  
    sconst_1  
    sstore s2  
    new Object  
    astore o2  
}
```

Instance de A
s1 = 0x0000
o1 = ●

Instance de Object

EEPROM

RAM

Pile de frames

var. locales

s2 =

o2 =

pile d'opérandes

Exemple de code en Java Card **sans** pré-persistence

```
public class A {  
  
    private short s1;  
    private Object o1;  
  
    public void foo() {  
        s1 = (short) 0;  
        o1 = new Object();  
    }  
  
    public void bar() {  
        short s2 = (short) 1;  
        Object o2 = new Object();  
    }  
}
```

Traduction en bytecodes des méthodes foo et bar

```
void foo() {  
    aload_0  
    sconst_0  
    putfield_s s1  
    aload_0  
    new Object  
    putfield_a o1  
}
```

```
void bar() {  
    sconst_1  
    sstore s2  
    new Object  
    astore o2  
}
```

←----- ICI

Instance de A
s1 = 0x0000
o1 = ●

Instance de Object

EEPROM

RAM

Pile de frames

var. locales

s2 =

o2 =

pile d'opérandes

0x0001

Exemple de code en Java Card **sans** pré-persistence

```
public class A {  
  
    private short s1;  
    private Object o1;  
  
    public void foo() {  
        s1 = (short) 0;  
        o1 = new Object();  
    }  
  
    public void bar() {  
        short s2 = (short) 1;  
        Object o2 = new Object();  
    }  
}
```

Traduction en bytecodes des méthodes foo et bar

```
void foo() {  
    aload_0  
    sconst_0  
    putfield_s s1  
    aload_0  
    new Object  
    putfield_a o1  
}
```

```
void bar() {  
    sconst_1  
    sstore s2  
    new Object  
    astore o2  
}
```

----- ICI

Instance de A
s1 = 0x0000
o1 = ●

Instance de Object

EEPROM

RAM

Pile de frames

var. locales

s2 = 0x0001

o2 =

pile d'opérandes

Exemple de code en Java Card **sans** pré-persistance

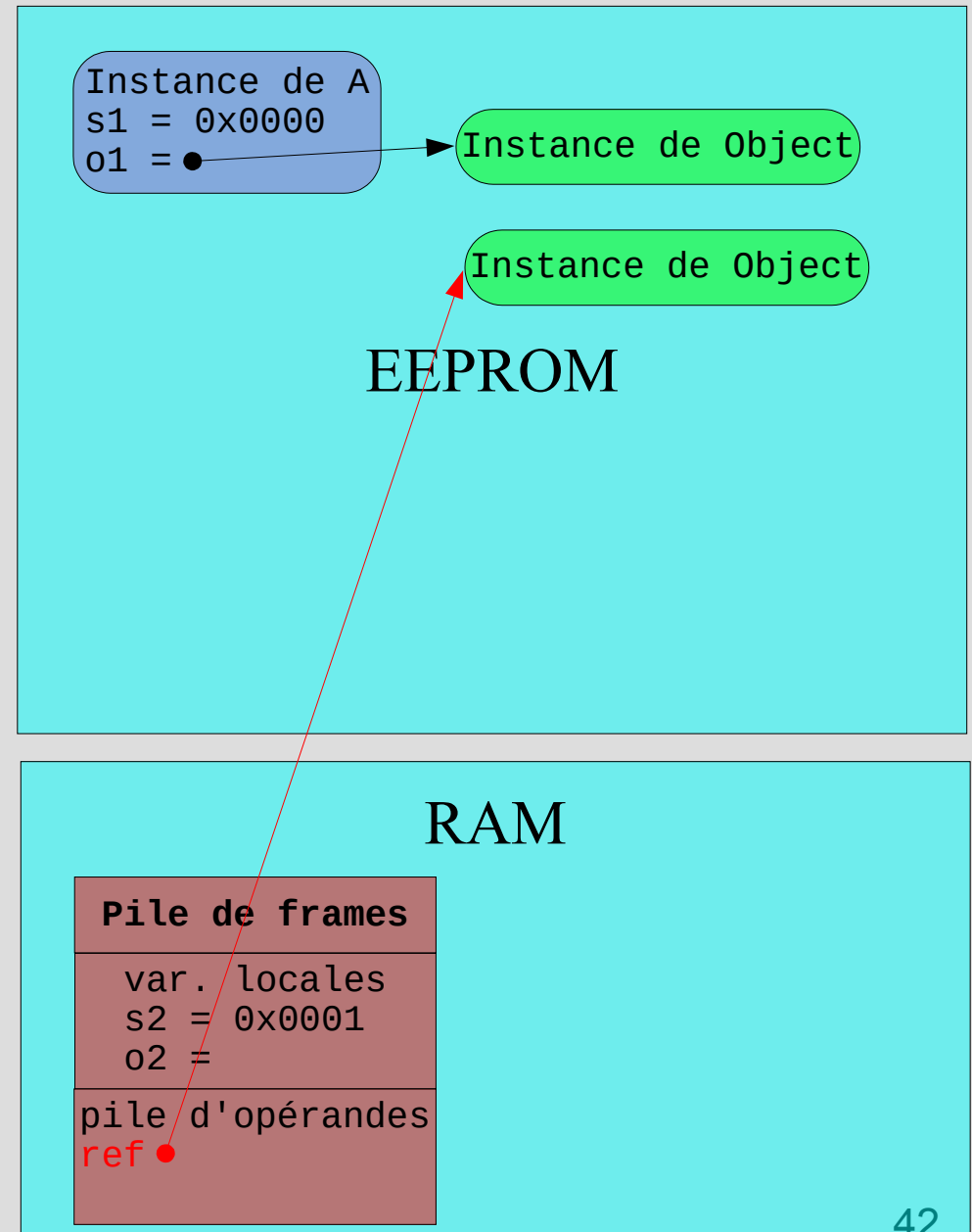
```
public class A {  
  
    private short s1;  
    private Object o1;  
  
    public void foo() {  
        s1 = (short) 0;  
        o1 = new Object();  
    }  
  
    public void bar() {  
        short s2 = (short) 1;  
        Object o2 = new Object();  
    }  
}
```

Traduction en bytecodes des méthodes foo et bar

```
void foo() {  
    aload_0  
    sconst_0  
    putfield_s s1  
    aload_0  
    new Object  
    putfield_a o1  
}
```

```
void bar() {  
    sconst_1  
    sstore s2  
    new Object  
    astore o2  
}
```

<----- ICI



Exemple de code en Java Card **sans** pré-persistence

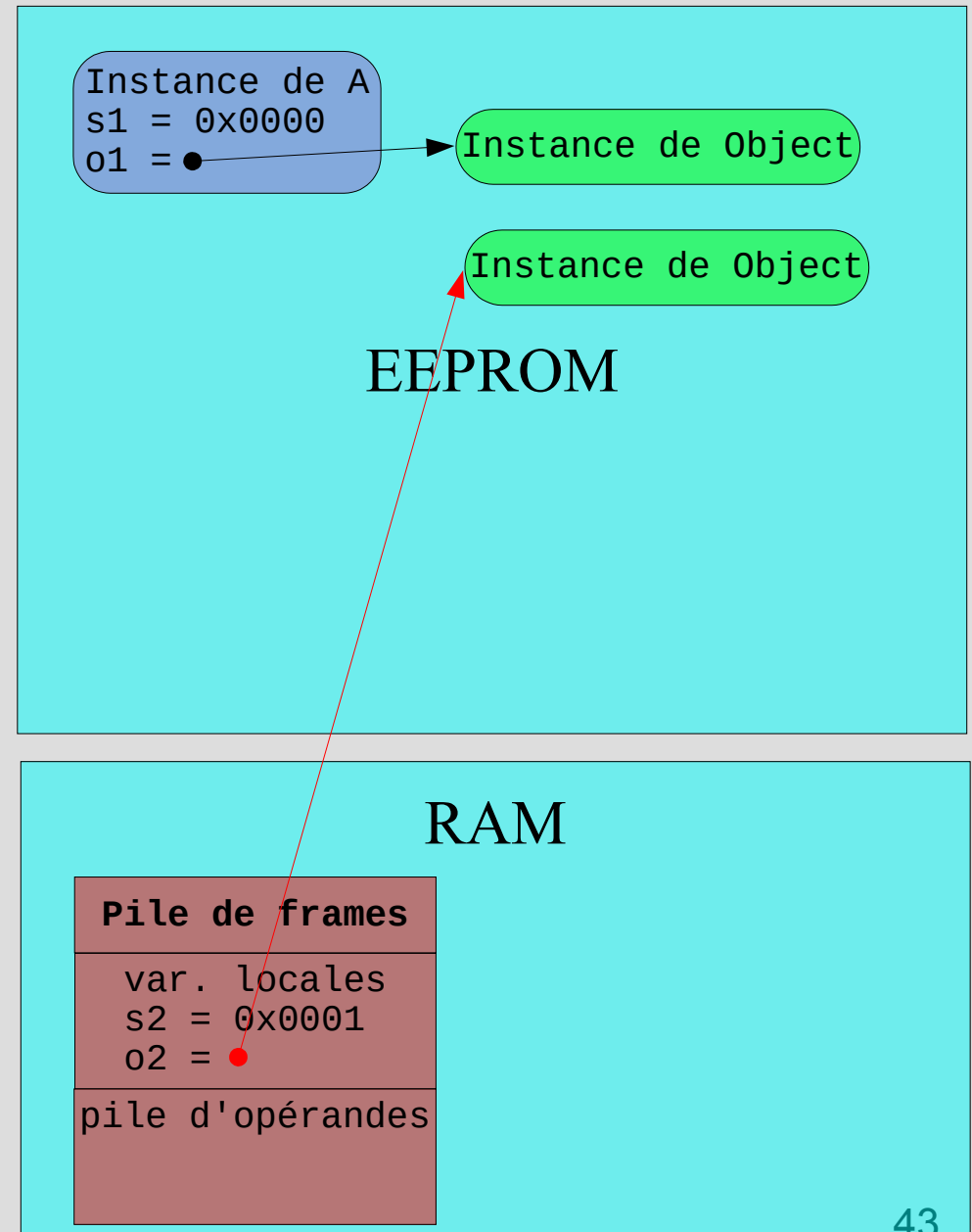
```
public class A {  
  
    private short s1;  
    private Object o1;  
  
    public void foo() {  
        s1 = (short) 0;  
        o1 = new Object();  
    }  
  
    public void bar() {  
        short s2 = (short) 1;  
        Object o2 = new Object();  
    }  
}
```

Traduction en bytecodes des méthodes foo et bar

```
void foo() {  
    aload_0  
    sconst_0  
    putfield_s s1  
    aload_0  
    new Object  
    putfield_a o1  
}
```

```
void bar() {  
    sconst_1  
    sstore s2  
    new Object  
    astore o2  
}
```

<----- ICI



Exemple de code en Java Card **sans** pré-persistence

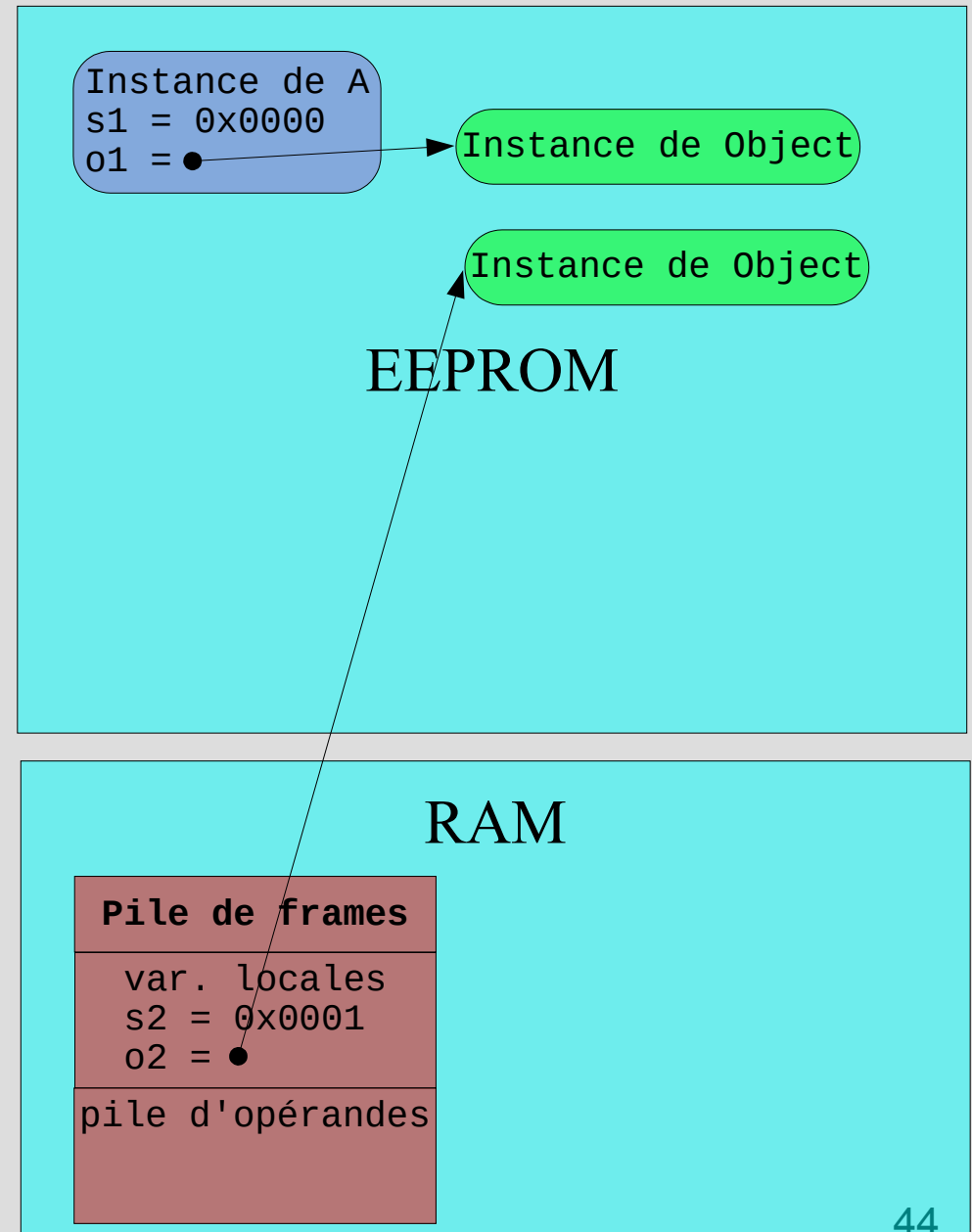
Remarques :

- sémantiques **identiques** pour les variables d'instances (contenu et référence)
- sémantiques **différentes** pour les variables locales (contenu et référence)
- Tous les objets sont dans l'**EEPROM**.

Problèmes :

Perte d'alimentation

=> **objet inaccessible et non ramassé**



Exemple de code en Java Card **avec** pré-persistance

```
public class A {  
  
    private short s1;  
    private Object o1;  
  
    public void foo() {  
        s1 = (short) 0;  
        o1 = new Object();  
    }  
  
    public void bar() {  
        short s2 = (short) 1;  
        Object o2 = new Object();  
    }  
}
```

Traduction en bytecodes des méthodes foo et bar

```
void foo() { <----- ICI
```

```
    aload_0  
    sconst_0  
    putfield_s s1  
    aload_0  
    new Object  
    putfield_a o1  
}
```

```
void bar() {  
    sconst_1  
    sstore s2  
    new Object  
    astore o2  
}
```

Instance de A
s1 =
o1 =

EEPROM

RAM

Pile de frames

var. locales

pile d'opérandes

Exemple de code en Java Card **avec** pré-persistance

```
public class A {  
  
    private short s1;  
    private Object o1;  
  
    public void foo() {  
        s1 = (short) 0;  
        o1 = new Object();  
    }  
  
    public void bar() {  
        short s2 = (short) 1;  
        Object o2 = new Object();  
    }  
}
```

Traduction en bytecodes des méthodes foo et bar

```
void foo() {  
    aload_0  
    sconst_0          <----- ICI  
    putfield_s s1  
    aload_0  
    new Object  
    putfield_a o1  
}  
  
void bar() {  
    sconst_1  
    sstore s2  
    new Object  
    astore o2  
}
```

Instance de A
s1 =
o1 =

EEPROM

RAM

Pile de frames

var. locales

pile d'opérandes

this
0x0000

Exemple de code en Java Card **avec** pré-persistance

```
public class A {  
  
    private short s1;  
    private Object o1;  
  
    public void foo() {  
        s1 = (short) 0;  
        o1 = new Object();  
    }  
  
    public void bar() {  
        short s2 = (short) 1;  
        Object o2 = new Object();  
    }  
}
```

Traduction en bytecodes des méthodes foo et bar

```
void foo() {  
    aload_0  
    sconst_0  
    putfield_s s1 <----- ICI  
    aload_0  
    new Object  
    putfield_a o1  
}  
  
void bar() {  
    sconst_1  
    sstore s2  
    new Object  
    astore o2  
}
```

Instance de A
s1 = **0x0000**
o1 =

EEPROM

RAM

Pile de frames

var. locales

pile d'opérandes

Exemple de code en Java Card **avec** pré-persistance

```
public class A {  
  
    private short s1;  
    private Object o1;  
  
    public void foo() {  
        s1 = (short) 0;  
        o1 = new Object();  
    }  
  
    public void bar() {  
        short s2 = (short) 1;  
        Object o2 = new Object();  
    }  
}
```

Traduction en bytecodes des méthodes foo et bar

```
void foo() {  
    aload_0  
    sconst_0  
    putfield_s s1  
    aload_0  
    new Object    <----- ICI  
    putfield_a o1  
}  
  
void bar() {  
    sconst_1  
    sstore s2  
    new Object  
    astore o2  
}
```

Instance de A
s1 = 0x0000
o1 =

EEPROM

RAM

Pile de frames

var. locales

pile d'opérandes

this
ref ●

Instance de Object

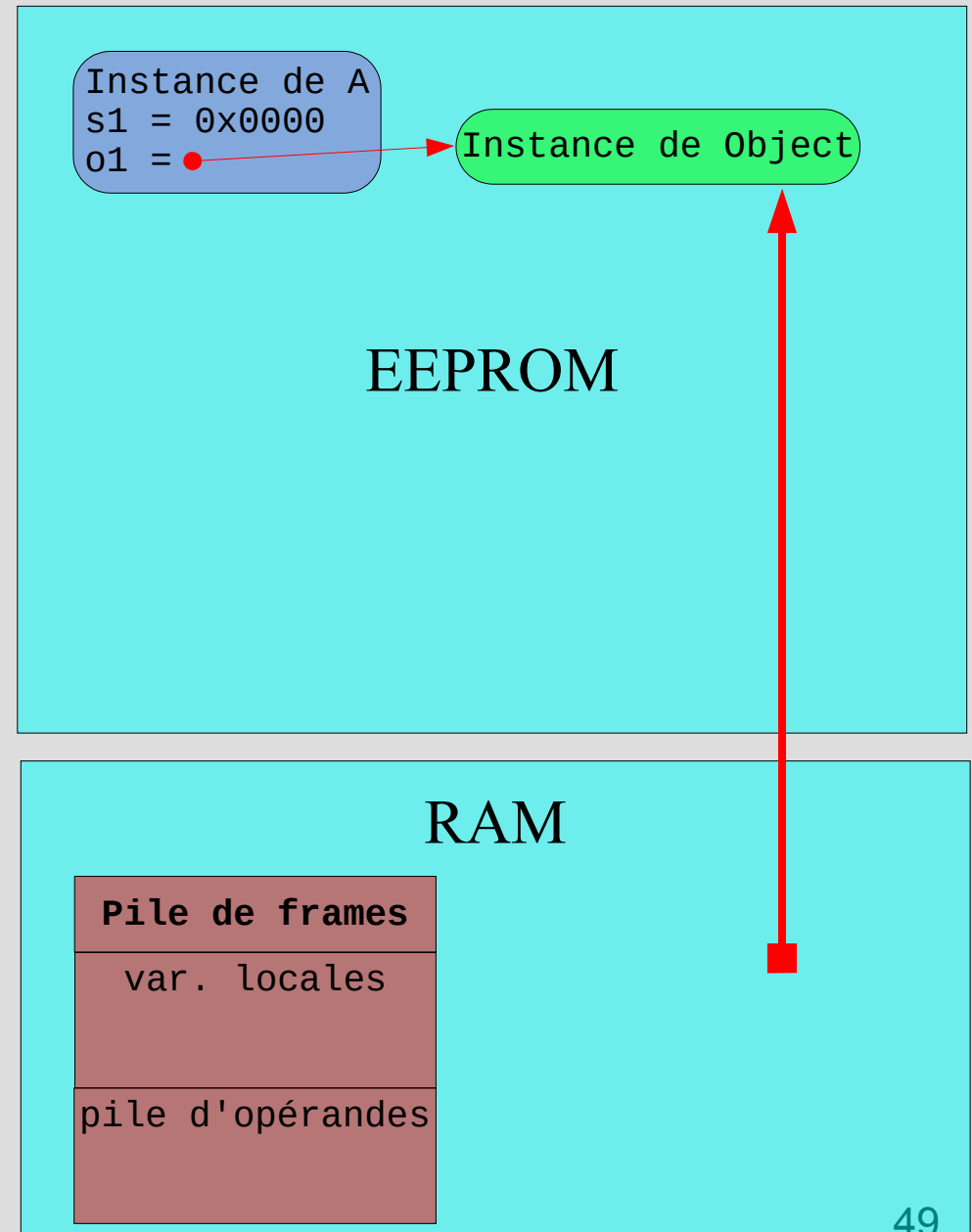
Exemple de code en Java Card **avec** pré-persistance

```
public class A {  
  
    private short s1;  
    private Object o1;  
  
    public void foo() {  
        s1 = (short) 0;  
        o1 = new Object();  
    }  
  
    public void bar() {  
        short s2 = (short) 1;  
        Object o2 = new Object();  
    }  
}
```

Traduction en bytecodes des méthodes foo et bar

```
void foo() {  
    aload_0  
    sconst_0  
    putfield_s s1  
    aload_0  
    new Object  
    putfield_a o1 <----- ICI  
}
```

```
void bar() {  
    sconst_1  
    sstore s2  
    new Object  
    astore o2  
}
```



Exemple de code en Java Card **avec** pré-persistance

```
public class A {  
  
    private short s1;  
    private Object o1;  
  
    public void foo() {  
        s1 = (short) 0;  
        o1 = new Object();  
    }  
  
    public void bar() {  
        short s2 = (short) 1;  
        Object o2 = new Object();  
    }  
}
```

Traduction en bytecodes des méthodes foo et bar

```
void foo() {  
    aload_0  
    sconst_0  
    putfield_s s1  
    aload_0  
    new Object  
    putfield_a o1  
}
```

```
void bar() {  
    sconst_1  
    sstore s2  
    new Object  
    astore o2  
}
```

<----- ICI

Instance de A
s1 = 0x0000
o1 = ●

Instance de Object

EEPROM

RAM

Pile de frames

var. locales

s2 =

o2 =

pile d'opérandes

Exemple de code en Java Card **avec** pré-persistance

```
public class A {  
  
    private short s1;  
    private Object o1;  
  
    public void foo() {  
        s1 = (short) 0;  
        o1 = new Object();  
    }  
  
    public void bar() {  
        short s2 = (short) 1;  
        Object o2 = new Object();  
    }  
}
```

Traduction en bytecodes des méthodes foo et bar

```
void foo() {  
    aload_0  
    sconst_0  
    putfield_s s1  
    aload_0  
    new Object  
    putfield_a o1  
}
```

```
void bar() {  
    sconst_1  
    sstore s2  
    new Object  
    astore o2  
}
```

<----- ICI

Instance de A
s1 = 0x0000
o1 = ●

Instance de Object

EEPROM

RAM

Pile de frames

var. locales

s2 =

o2 =

pile d'opérandes

0x0001

Exemple de code en Java Card **avec** pré-persistance

```
public class A {  
  
    private short s1;  
    private Object o1;  
  
    public void foo() {  
        s1 = (short) 0;  
        o1 = new Object();  
    }  
  
    public void bar() {  
        short s2 = (short) 1;  
        Object o2 = new Object();  
    }  
}
```

Traduction en bytecodes des méthodes foo et bar

```
void foo() {  
    aload_0  
    sconst_0  
    putfield_s s1  
    aload_0  
    new Object  
    putfield_a o1  
}
```

```
void bar() {  
    sconst_1  
    sstore s2  
    new Object  
    astore o2  
}
```

<----- ICI

Instance de A
s1 = 0x0000
o1 = ●

Instance de Object

EEPROM

RAM

Pile de frames

var. locales

s2 = 0x0001

o2 =

pile d'opérandes

Exemple de code en Java Card **avec** pré-persistance

```
public class A {  
  
    private short s1;  
    private Object o1;  
  
    public void foo() {  
        s1 = (short) 0;  
        o1 = new Object();  
    }  
  
    public void bar() {  
        short s2 = (short) 1;  
        Object o2 = new Object();  
    }  
}
```

Traduction en bytecodes des méthodes foo et bar

```
void foo() {  
    aload_0  
    sconst_0  
    putfield_s s1  
    aload_0  
    new Object  
    putfield_a o1  
}
```

```
void bar() {  
    sconst_1  
    sstore s2  
    new Object  
    astore o2  
}
```

<----- ICI

Instance de A
s1 = 0x0000
o1 = ●

Instance de Object

EEPROM

RAM

Pile de frames

var. locales
s2 = 0x0001
o2 =

pile d'opérandes
ref ●

Instance de Object

Exemple de code en Java Card **avec** pré-persistance

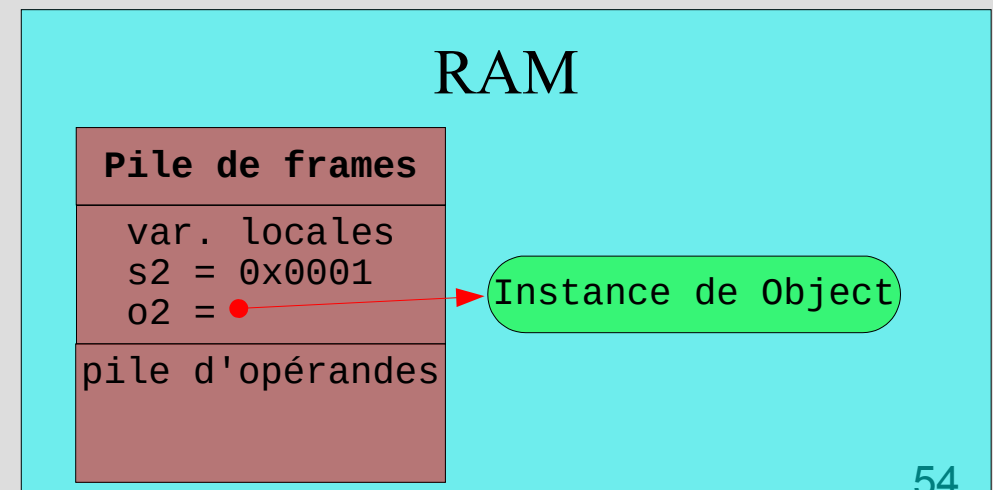
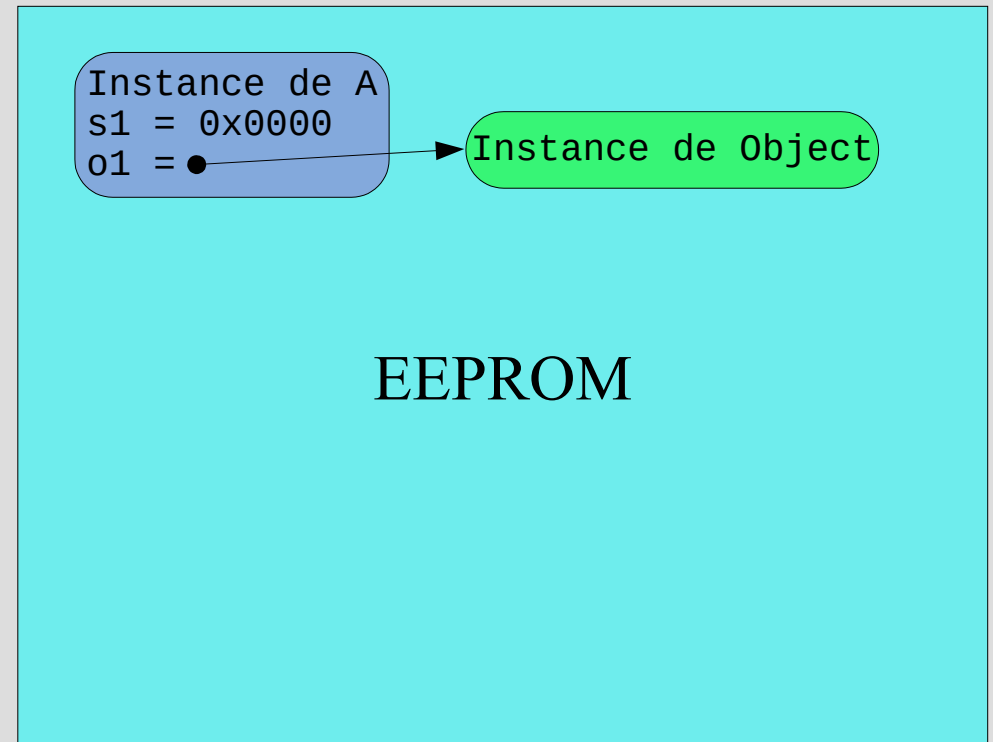
```
public class A {  
  
    private short s1;  
    private Object o1;  
  
    public void foo() {  
        s1 = (short) 0;  
        o1 = new Object();  
    }  
  
    public void bar() {  
        short s2 = (short) 1;  
        Object o2 = new Object();  
    }  
}
```

Traduction en bytecodes des méthodes foo et bar

```
void foo() {  
    aload_0  
    sconst_0  
    putfield_s s1  
    aload_0  
    new Object  
    putfield_a o1  
}
```

```
void bar() {  
    sconst_1  
    sstore s2  
    new Object  
    astore o2  
}
```

<----- ICI



Exemple de code en Java Card **avec** pré-persistence

Remarques :

- sémantiques **identiques** les variables d'instances (contenu et référence)
- sémantiques **identiques** pour les variables locales (contenu et référence)
- Les objets sont dans deux mémoires

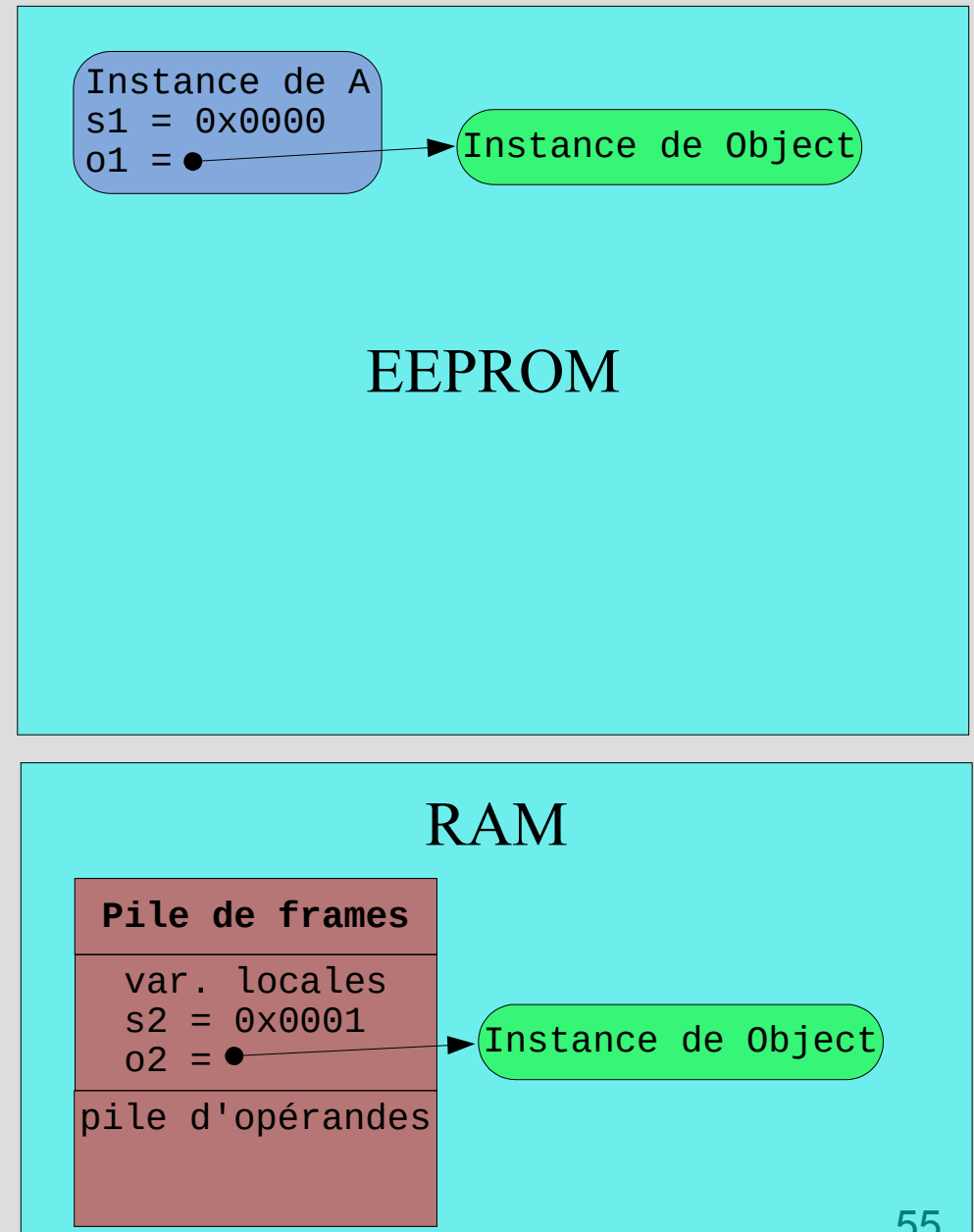
Avantages :

- sémantiques **homogènes comme en Java**
 - sur la perte d'alimentation : **ramasse-miette naturel de la RAM**
- => pas d'objet inaccessible non ramassé

Inconvénients :

- Mécanisme de recopie des objets de la RAM vers l'EEPROM

Implantations Java Cards différentes mais conforme aux spécifications => failles de sécurité. (DCSSI)



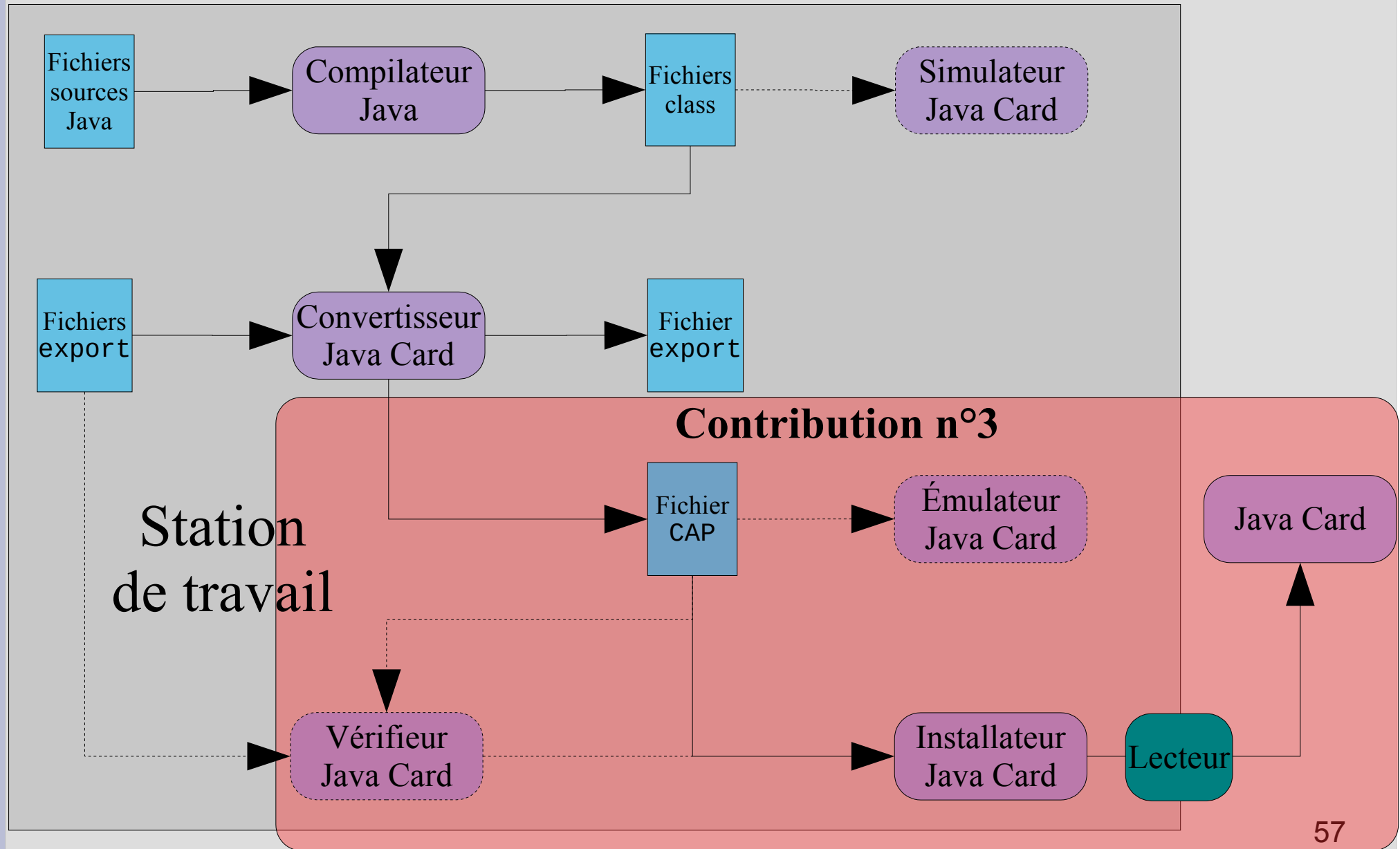
De nouveaux problèmes de sécurité pour les cartes multi-applicatives ouvertes

Quelques attaques classiques

Les attaques internes

Les nouvelles vulnérabilités et leur exploitation

Contribution n°3 : les problèmes de sécurité des cartes multi-applicatives ouvertes



Quelques vulnérabilités **classiques** des cartes

Les attaques non invasives :

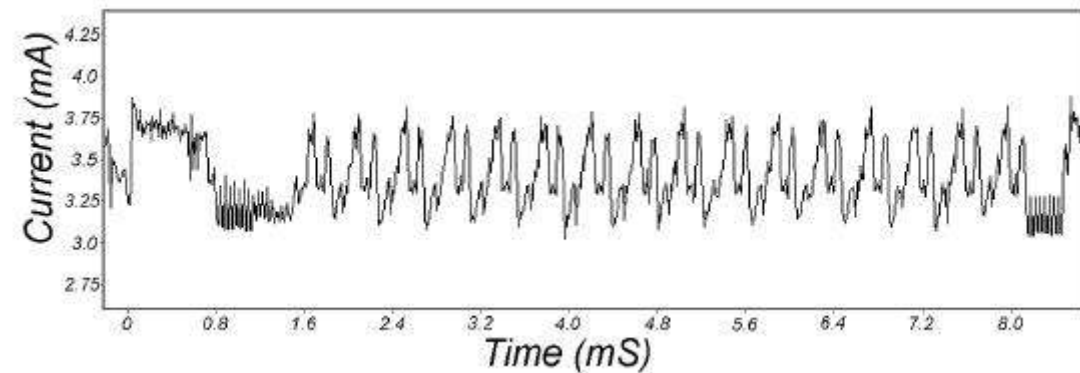
Fonctionnement hors conditions normal (T° , tension, lumière, etc.)

(Images SERMA Technologies)

Par injection de fautes

Par canaux cachés

- Temps d'exécution
- **Consommation en courant** →
- Émission électromagnétique



SPA sur le DES

Contre-mesure : Pompe de charge

Les attaques invasives :

Microprobing

Modification de circuit

Les attaques internes sur les cartes multi-applicatives ouvertes

Attaques sur le typage ou la visibilité :

Bloquées par le vérifieur

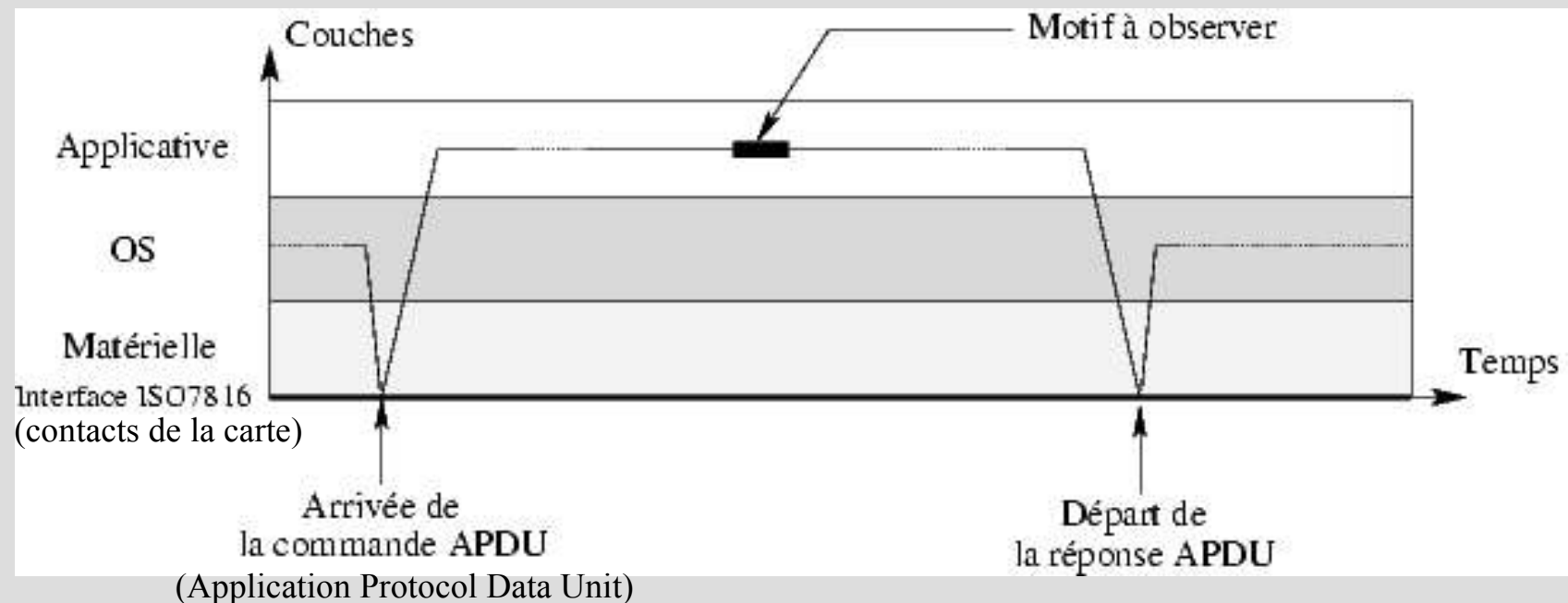
Attaques sur le pare-feu :

Possible seulement si une applet sur la carte utilise mal les services de partage d'objets offerts par la plate-forme
=> kit de développement sécurisé

Attaques de collecte d'information :

Instrumentation d'une application

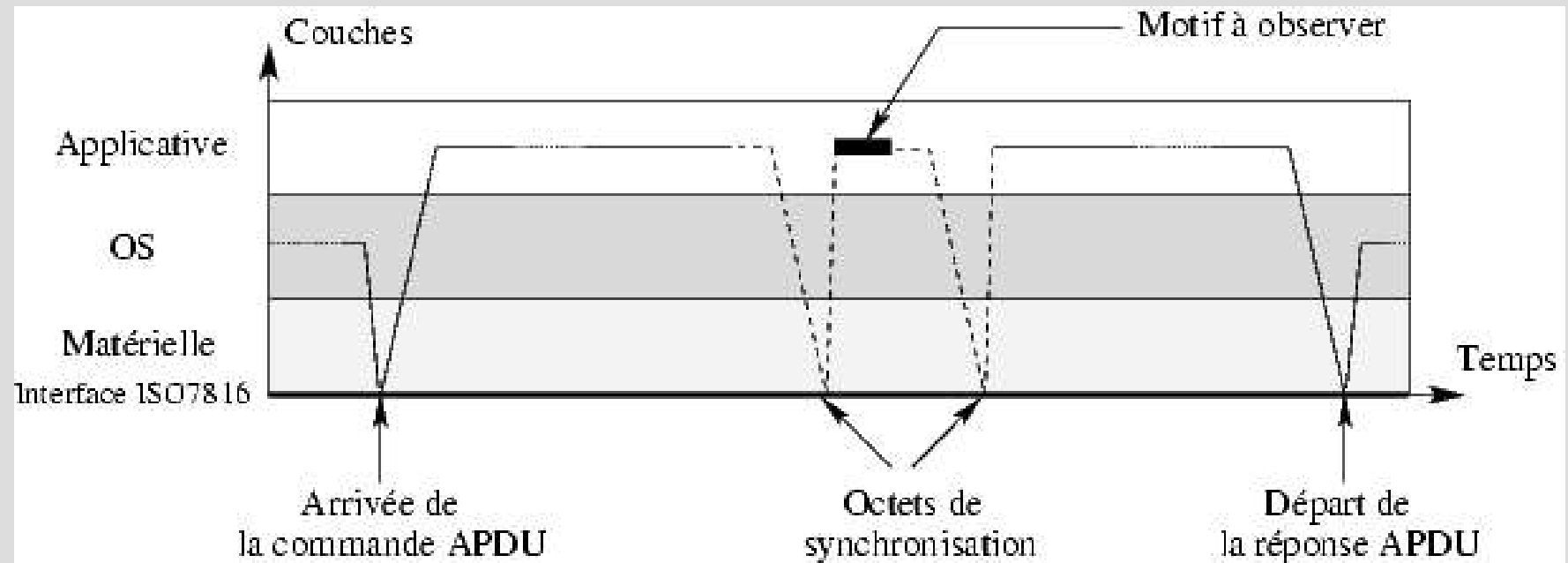
Pas de parade ! Mais l'intérêt est très limité tel quel.



Nouvelles vulnérabilités des cartes multi-applicatives ouvertes

Idée de base : Comment rendre un motif localisable pour un observateur extérieur ?

Solution n°1 : envoi d'octets de synchronisation.



Méthodes :

`waitExtension` : désactivé et géré par la plate-forme

On a proposé une technique de détournement de la communication : impossible de la désactiver

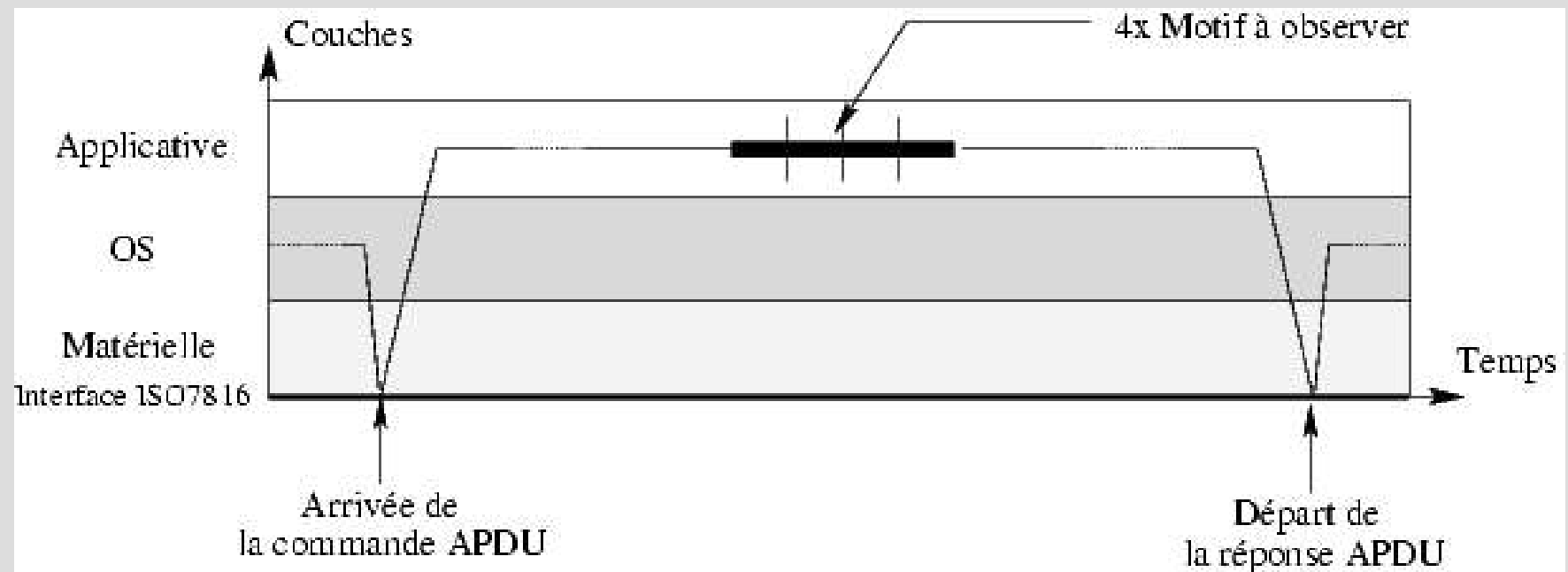
Amélioration de la précision par des modifications au niveau bytecode (**JCatools**)

Nouvelles vulnérabilités des cartes multi-applicatives ouvertes

Idée de base : Comment rendre un motif localisable pour un observateur extérieur ?

Solution n°2 : répéter plusieurs fois le motif à observer

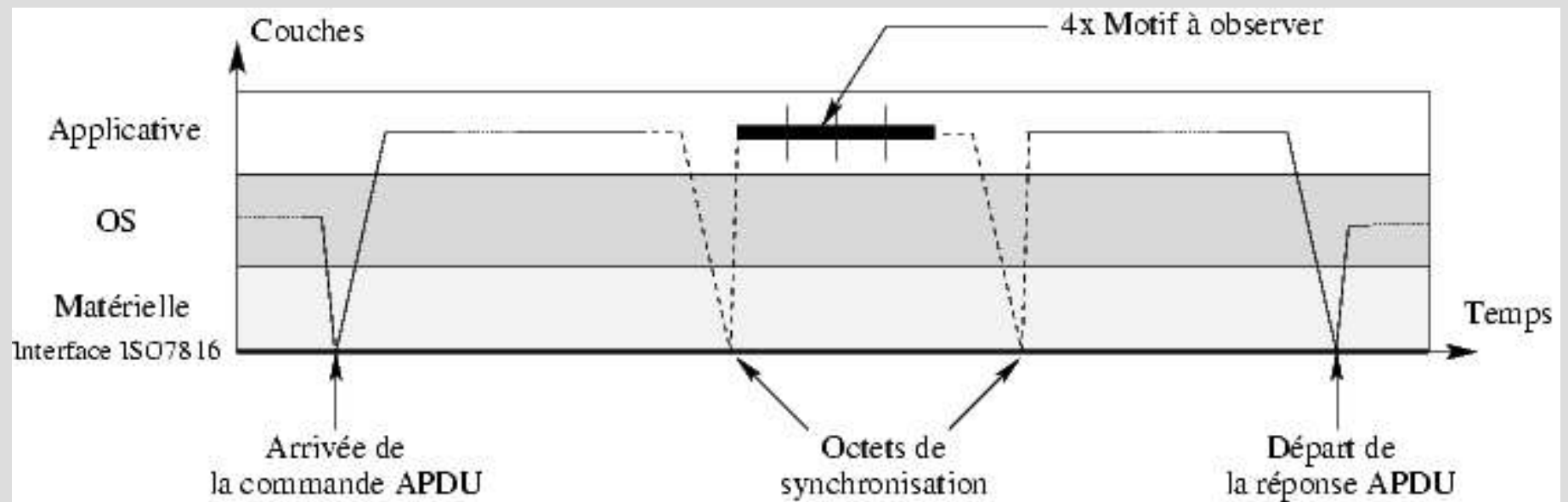
Nécessite parfois des astuces (exemple : pop => séquence dup-pop)



Nouvelles vulnérabilités des cartes multi-applicatives ouvertes

Idée de base : Comment rendre un motif localisable pour un observateur extérieur ?

Solution n°3 : mélanger les 2 solutions précédentes



Exploitation des vulnérabilités

Attaques par couplage

Construction d'un dictionnaire des émanations physiques pour :

- Les bytecodes
- Les services

Comparaison par rapport à la trace d'une applet officielle présente sur la carte

- Rétro-conception à partir de signaux physiques

Attaque contre la confidentialité du code de l'applet officielle

Attaques physiques

Déterminer le motif à attaquer (service) d'une applet officielle

Implanter sa propre applet utilisant ce motif facilement localisable et l'attaquer (injection de fautes, etc.)

=> Maximise l'efficacité de l'attaquant : il est dans les meilleures conditions.

En cas de succès, attaquer le même motif dans l'applet officielle.

Résultats

Techniques de localisation pour tester les attaques physiques contre les plates-formes :

Utilisées par SERMA Technologies

Succès !

Constitution d'un dictionnaire de bytecodes :

Non abouti !

Parce que :

- Java Card déjà certifiée EAL5+
- peu de moyens déployés.

Preuve de concept pour sécuriser le calcul sur la grille

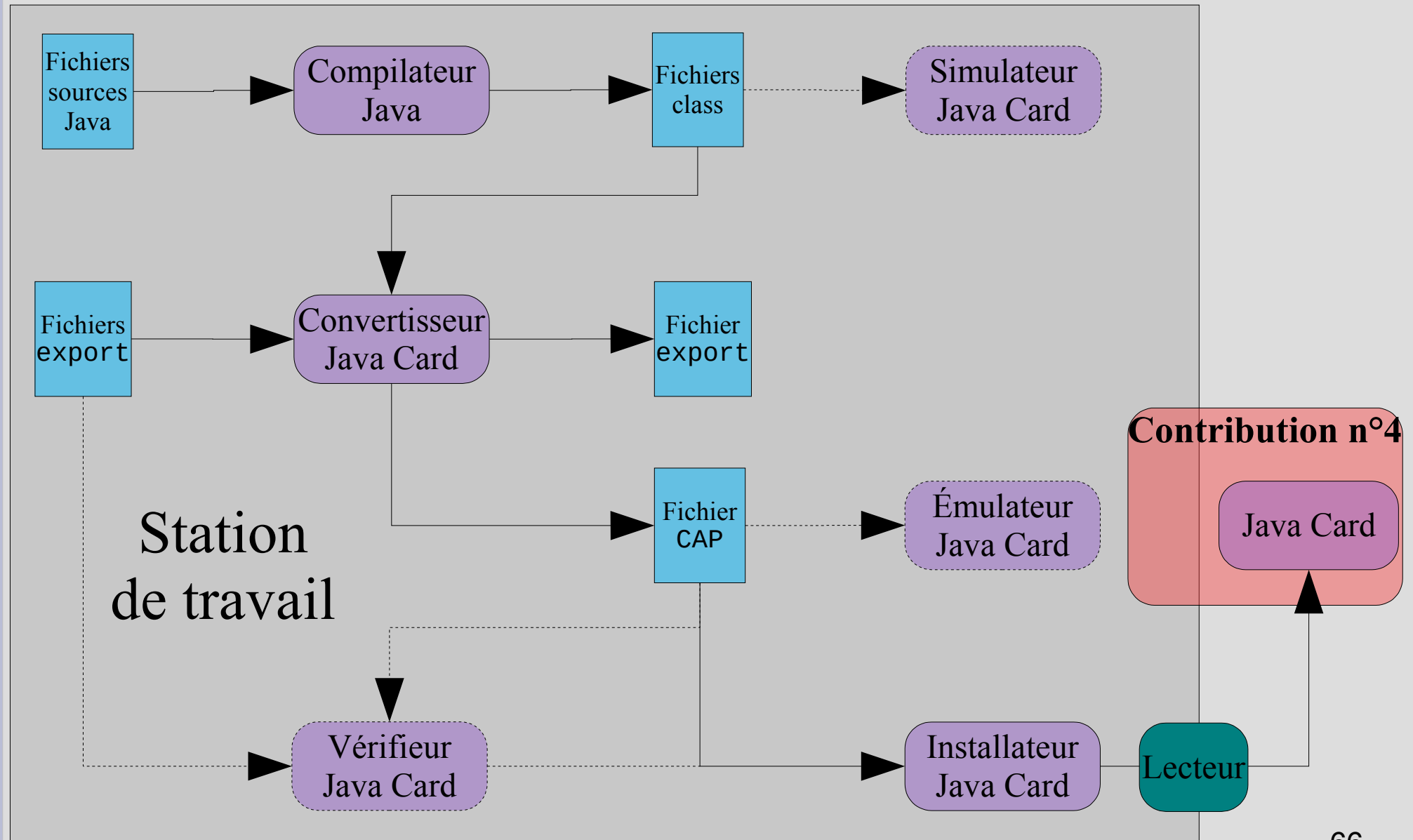
Quelques problèmes de sécurité sur les grilles

La preuve de concept

Quelques défis

Une application de démonstration

Contribution n°4 : l'utilisation des Java Cards pour sécuriser le calcul sur la grille



Quelques problèmes des grilles de calculs

Pour le propriétaire des ressources de calculs :

Sécurité du système et des autres applications

- Solutions classiques :

Sandbox

Analyse du code (*open source* ?)

Signature de l'application

- **Notre solution** : une architecture plus sécurisée, la carte à puce (évaluée, certifiée, OS et puce sécurisés)

Pour le distributeur d'application :

Confidentialité de son code

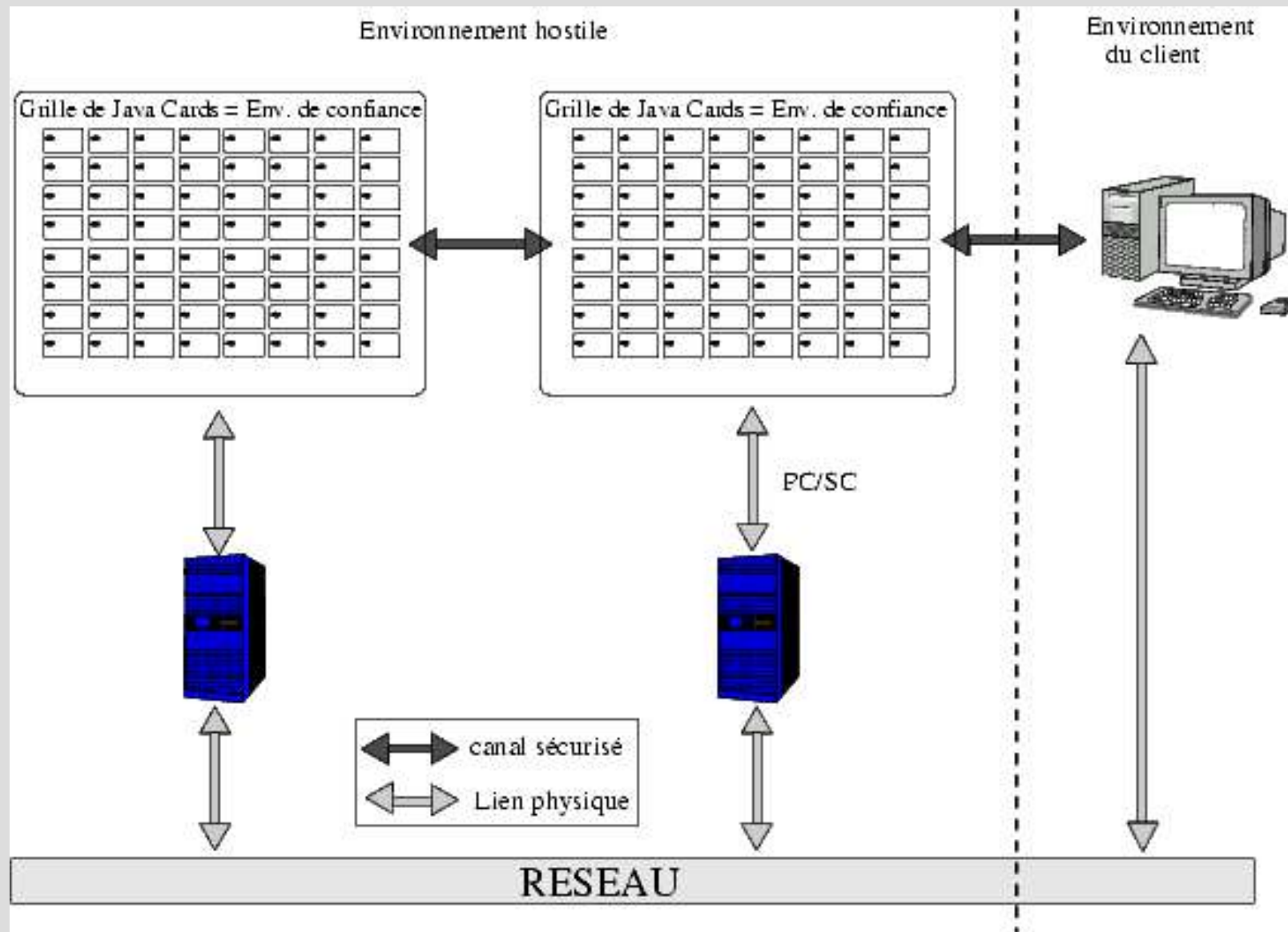
- Solutions classiques :

Distribution sous forme binaire (problème de rétro-conception par analyse logicielle ou matérielle)

Hypothèse de confiance dans le fournisseur des ressources de calculs

- **Notre solution** : la Java Card grâce à ses protections physiques (puce) et logicielles (OS et pare-feu)

Preuve de concept : la grille de Java Cards (JCGrid)

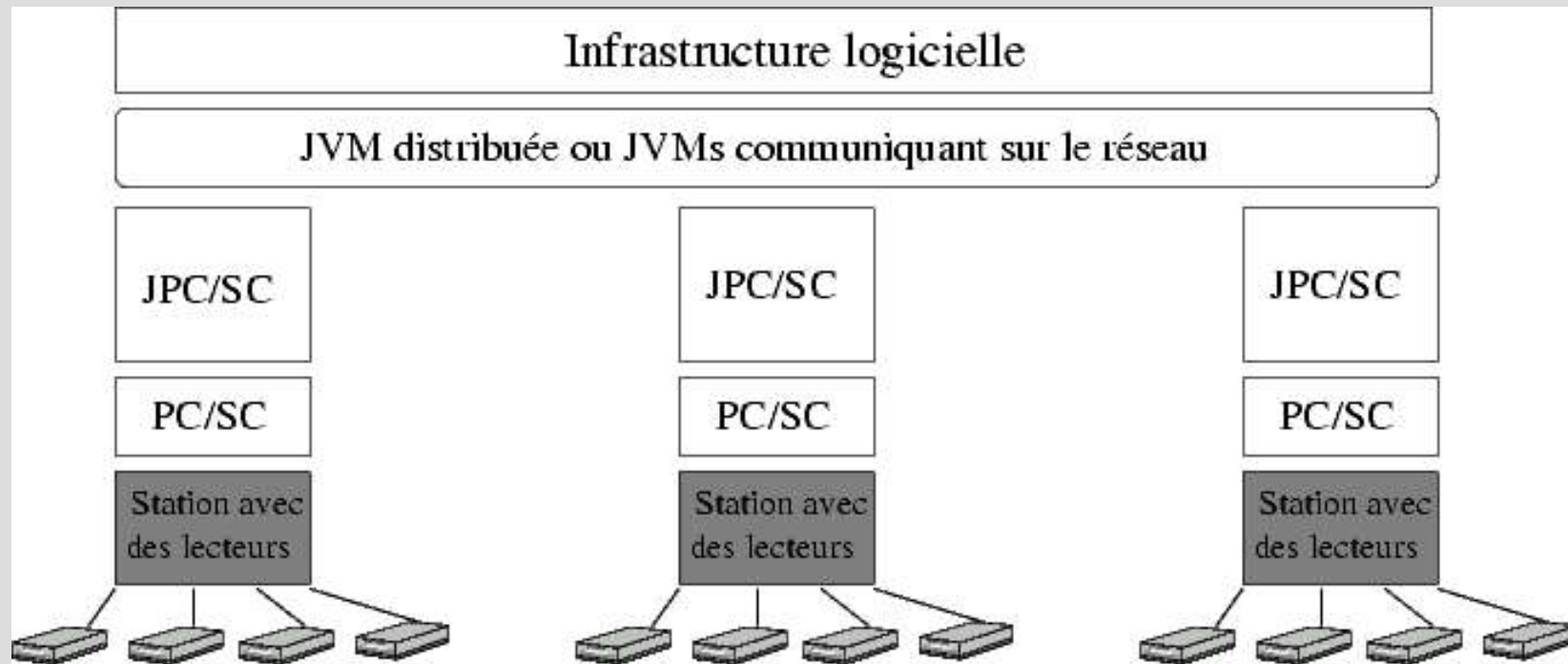


Attention : on ne prétend pas faire du calcul efficace mais seulement du calcul sécurisé 68

L'infrastructure matérielle



L'infrastructure logicielle



PC/SC est un standard qui fournit une API de haut niveau pour dialoguer avec les lecteurs et donc les cartes à puce.

Quelques défis

Faire du calcul scientifique sur une Java Card

problème de type, de réutilisation d'objet

Gestions de plusieurs lecteurs en parallèle

pcsc-lite souffrait de quelques inconvénients

Modèle d'exécution

Asynchronisme

Carte pro-active

La sécurité des communications

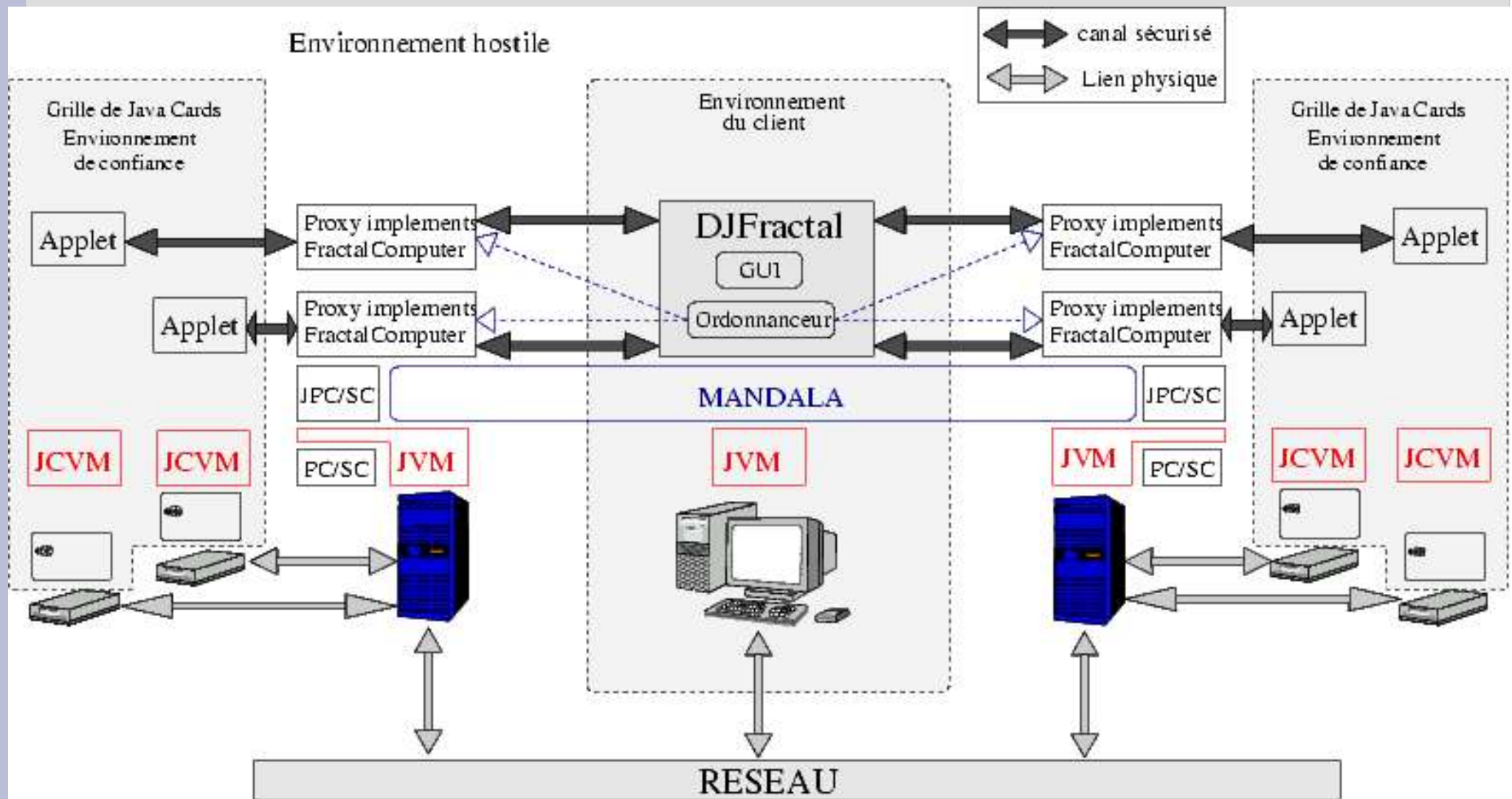
Entre l'utilisateur et la grille

Entres les cartes au sein de la grille

La dynamicité

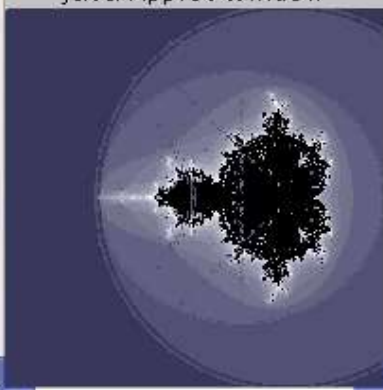
...

Le calcul **sécurisé** de la fractale de Mandelbrot



Résultats (1/2)

JFractal
Java Applet Window



JFractal Controls
Java Applet Window

Start Stop Reset

	X	Y
Left/Bottom	-3	-2
Right/Top	1	2
Delta	0.02010	0.02010
Mouse	-1.59300	-1.93960
Image size	200	200

Parameters
Iterations 20
Precision 5
Subdivisions 100
Steps: <= 6
Points: <= 5460
Points/Size: <= 14
Areas ratio
0 25 50 75 100

Monitoring

Progress

Progress 100%

Elapsed time 04:10:53.932

Remaining time 00:00:00.000

Estimated time 04:10:53.932

Iterations stats

Min 1

Max 20

Average 4

Total 173547

Time stats

Computed time min INFINITY

Computed time max -INFINITY

Computed time average 00:00:02.723

Computed time total 1 day.

Engine

FCS size 9 Manage...

Schedulers djfractal.SmartScheduler

☐ details

Log

Starting...
00:00:00.527 [Thread(Thread-13,6,main)]: 733 kbytes have been freed invoking System.gc()
00:00:03.675 [Thread(Thread-13,6,main)]: Starting the computation of FractalData[geometry: 200x200+0
00:00:03.841 [Thread(Thread-13,6,main)]: compute() areasNb: 0

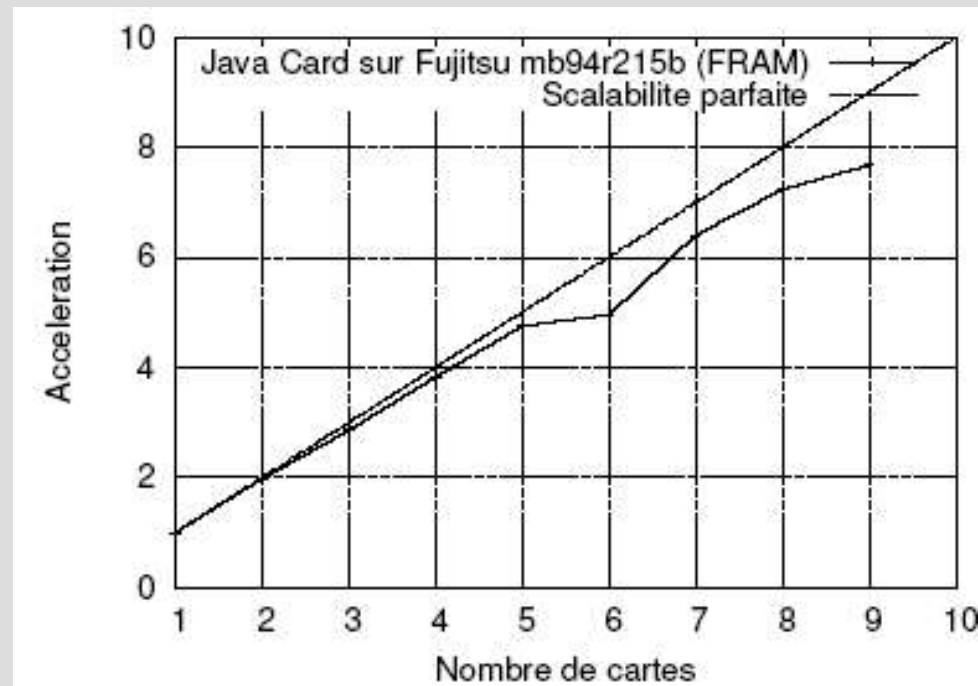
☒ Verbose mode ☒ Debug mode ☒ Progress mode

Benchmarks

Modèle de Java Card	Temps d'exécution (min)
9 Java Card sur un composant Fujitsu mb94r215b (FRAM)	122
9 GemXpresso Pro R3 E32PK	202
9 JCOP31bio	253
9 GemXpresso Pro R3 E64PK	376
9 SmartC@fé Expert	389
Le mélange	344

=> utilisation
de FRAM

Scalabilité



Conclusion et perspectives

Autour de Java Card et de la sécurité :

JCatools :

- permet le test d'applets et de plates-formes (batteries de tests d'attaques)
- Perspective : poursuite du développement (plug'ins, etc.) et libération du produit

Pré-persistance :

- contribution pour clarifier les spécifications => meilleure sécurité
- présentation à la DCSSI et au Java Card Forum
- Perspective : établir une formalisation

Nouvelles vulnérabilités :

- batterie de tests d'attaques
- Perspective : Approfondir la construction d'un dictionnaire au sein de XLIM (Université de Limoges)

JCGrid :

- utilisation de la Java Card pour sécuriser un système plus large
- Perspective : passage à l'échelle et utilisation dans le cadre de la mobilité (LaBRI – XLIM)



Pare-feu

Objectif :

Isoler les applets et les différents objets créés au sein d'espaces appelé contextes.

Le contexte du JCRE possède des privilèges spéciaux.

Espace du système

Contexte du JCRE

Espace des applets

Contexte 1

Applet A

Applet B

Paquetage X

Contexte 2

Applet C

Applet D

Paquetage Y

Objets transients

Un objet transient est un objet dont le **contenu est temporaire :**

- l'entête persiste et l'espace alloué au contenu également.



Un objet transient est associé à un événement dont l'occurrence déclenche la réinitialisation de ses champs.

Les seuls objets transients sont les tableaux de types simples.

Quelques caractéristiques :

- il a été créé par `makeTransientBooleanArray` (et consœurs de la classe `JCSysTem`) ;
- il est réinitialisé lors de l'occurrence d'un événement.

Comparatif des 2 solutions par rapport à Java

Propriété	Java	Java Card sans PP	Java Card avec PP
Sémantique pour les variables d'instance	cohérent avec lui même	cohérent mais persistant	cohérent mais persistant
Sémantique pour les variables locales	cohérent avec lui même	incohérent avec Java	cohérent avec Java
Inaccessibilité des objets	aucune grâce au ramasse-miette	oui	non sauf si le programmeur le désire ! Il y a un mécanisme de ramasse-miette naturelle de la RAM

Pré-persistance

Notre contribution :

- L'introduction du concept de pré-persistance :
- obtention de la sémantique originelle de Java

Une clarification des spécifications concernant le TAS (définition, contenu, localisation).

⇒ 4 modèles mémoires aux propriétés très différentes : fonctionnalités, sémantique et **sécurité** !

Perspectives :

Formalisation à l'aide d'une sémantique opérationnelle des interactions entre objets :

- persistants
- transients
- pré-persistants

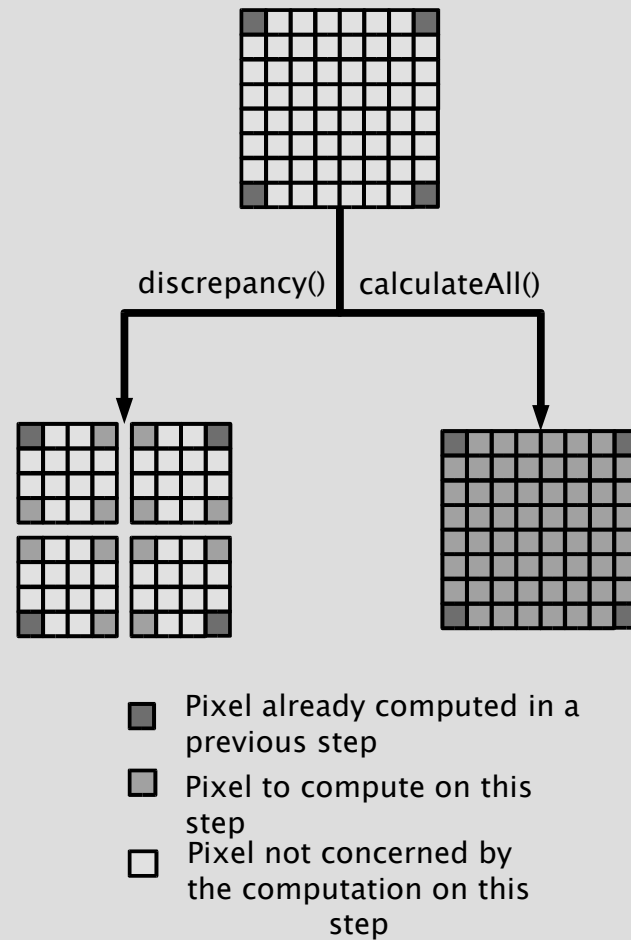
⇒ visualisation de la contamination lors de l'affectation d'un objet transient ou pré-persistant à un champ d'un objet persistant !

But final : Proposer au Java Card Forum une modification des spécifications.

Présentation à la DCSSI.

Le principe du découpage

Inspiré de l'algorithme de Peter Alfeld



Quelques problèmes des grilles de calculs

Pour le distributeur d'application :

Intégrité de l'exécution de l'application

- Solutions classiques :
 - Recalcul des résultats (faux positifs et faux négatifs)
 - Hypothèse de confiance
- **Notre solution** : la carte à puce car elle est prévue pour résister à l'injection de fautes

Sécurité des communications

- Solution classique :
 - PKI (problème : les clés privées sont rarement stockées de façon sécurisée)
- **Notre solution** : la carte à puce, le coffre-fort logique le plus sûr existant

FBI - Air France

Situation internationale :

Suite aux attentats du 11 septembre 2001, le FBI veut accéder aux fichiers des compagnies aériennes.
En Europe, des directives européennes renforcent la confidentialité des informations sur les passagers.

=> **Problème !**

Application de *data-mining*

Jeu de données à analyser

Les contraintes :

Les données ou une partie d'entre elles doivent rester confidentielles (données d'Air France)

Le code manipulant les données doit rester confidentiel (code du FBI)

Partenaires

Sponsors



Supports



Contacts

