Bytecode: Information flow

```
1  load x
2  store y
3  halt
```

explicit flow

x is loaded onto the stack, then it is stored into y, that is, y depends explicitly on x

```
1  load x
2  if 5
3  push 1
4  goto 6
5  push 0
6  store y
7  halt
```

implicit flow

variable x is loaded onto the stack. Depending on the value of x, either the constant 1 or the constant 0 is pushed onto the stack, and successively stored onto y

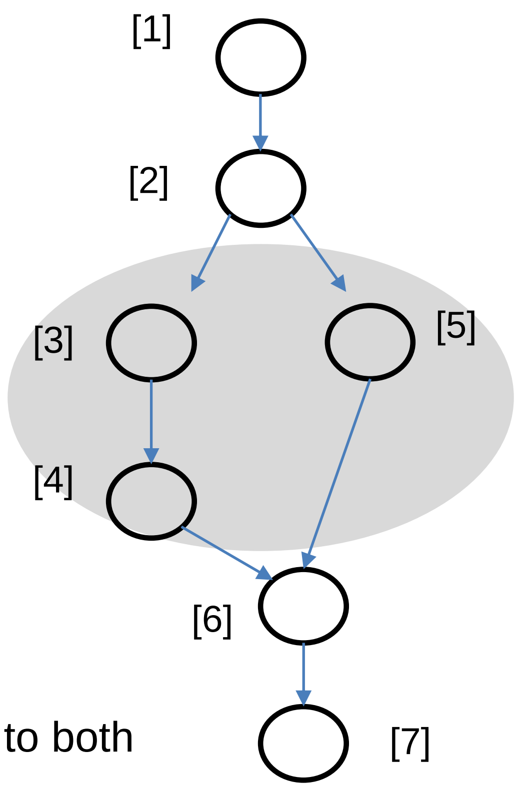
In both cases observing the final value of y reveals information on the value of x

Secure Information flow in Java bytecode

`op`	pop two operands off the stack, perform the operation, and push the result onto the stack
`pop`	discard the top value from the stack
`push k`	push the constant k onto the stack
`load x`	push the value of variable x onto the stack
`store x`	pop off the stack and store the value into x
`if j`	pop off the stack and jump to j if **non-zero**
`goto j`	jump to j
`halt`	stop

Implicit flow

```
1  load x
2  if 5
3  push 1
4  goto 6
5  push 0
6  store y
7  halt
```



Implicit flow starts at [2]

When implicit flow terminates?

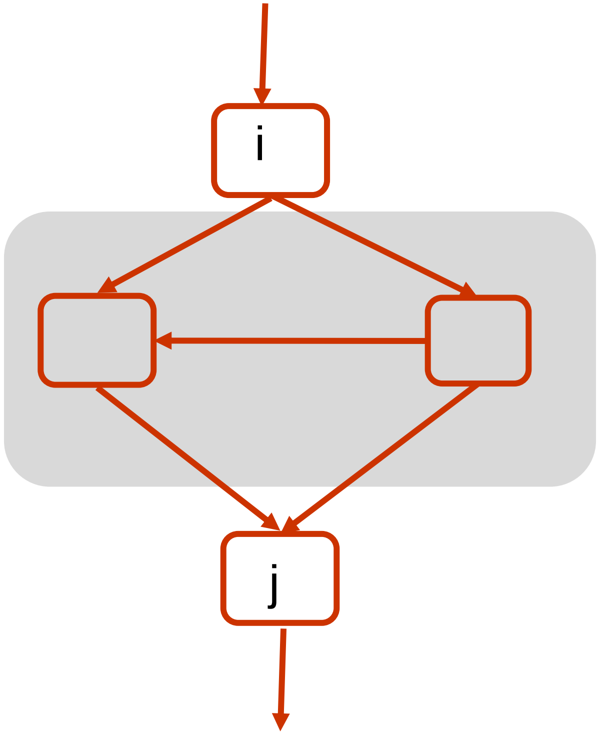
[6] is the first instruction that is common to both branches

The implicit flow terminates at [6]

[6] is the first instruction that is not under the implicit flow

Implicit flow

We use the concept of immediate postdominator on the CFG of the program to handle implicit flows



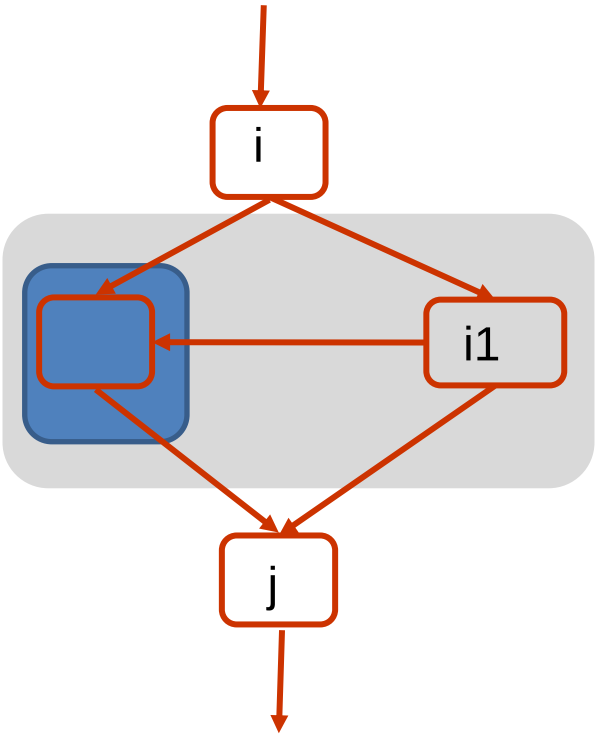
immediate postdominator of i: the first node belonging to all paths from i

ipd(i) = j

The implicit flow of an if instruction at address i terminates at the instruction with address ipd(i)

Implicit flow

What about nested control instructions?



Nested implicit flows

The innest implicit flow is the implicit flow that terminate first

immediate postdominator of i1: the first node belonging to all paths from i

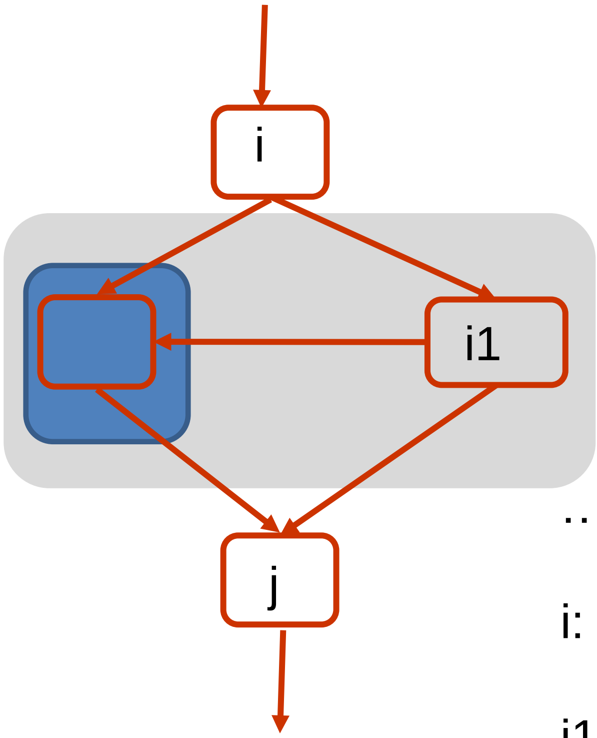
ipd(i1) = j

IPD stack:

when executing an instruction, the ipd stack mantains information on the open implict flows

IPD stack is updated any time a control instruction is enetered and any time a control instruction terminates

Implicit flow

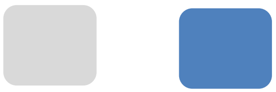


Execution of instructions

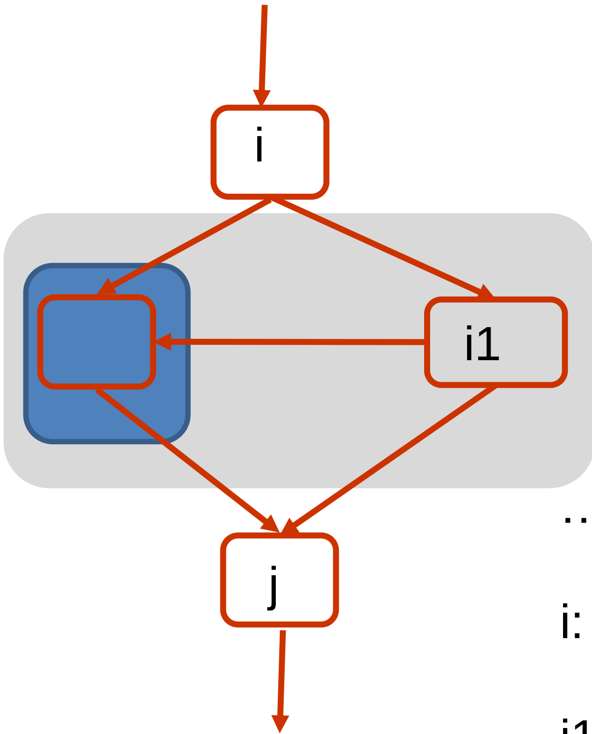
when an instruction j is executed: if the instruction j is the top of the ipd stack, the stack is updated by executing pop (j is removed from the stack)

…before i …….	Stack of ipd:	λ
i: control instruction	Stack of ipd:	ipd(i)
i1: control instruction	Stack of ipd:	ipd(i1) ipd(i)
j: top of the ipd stack	Stack of ipd:	ipd(i)
j: top of the ipd stack	Stack of ipd:	λ

CONTROL REGION of a branching instruction



Implicit flow

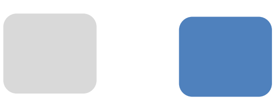


Execution of instructions

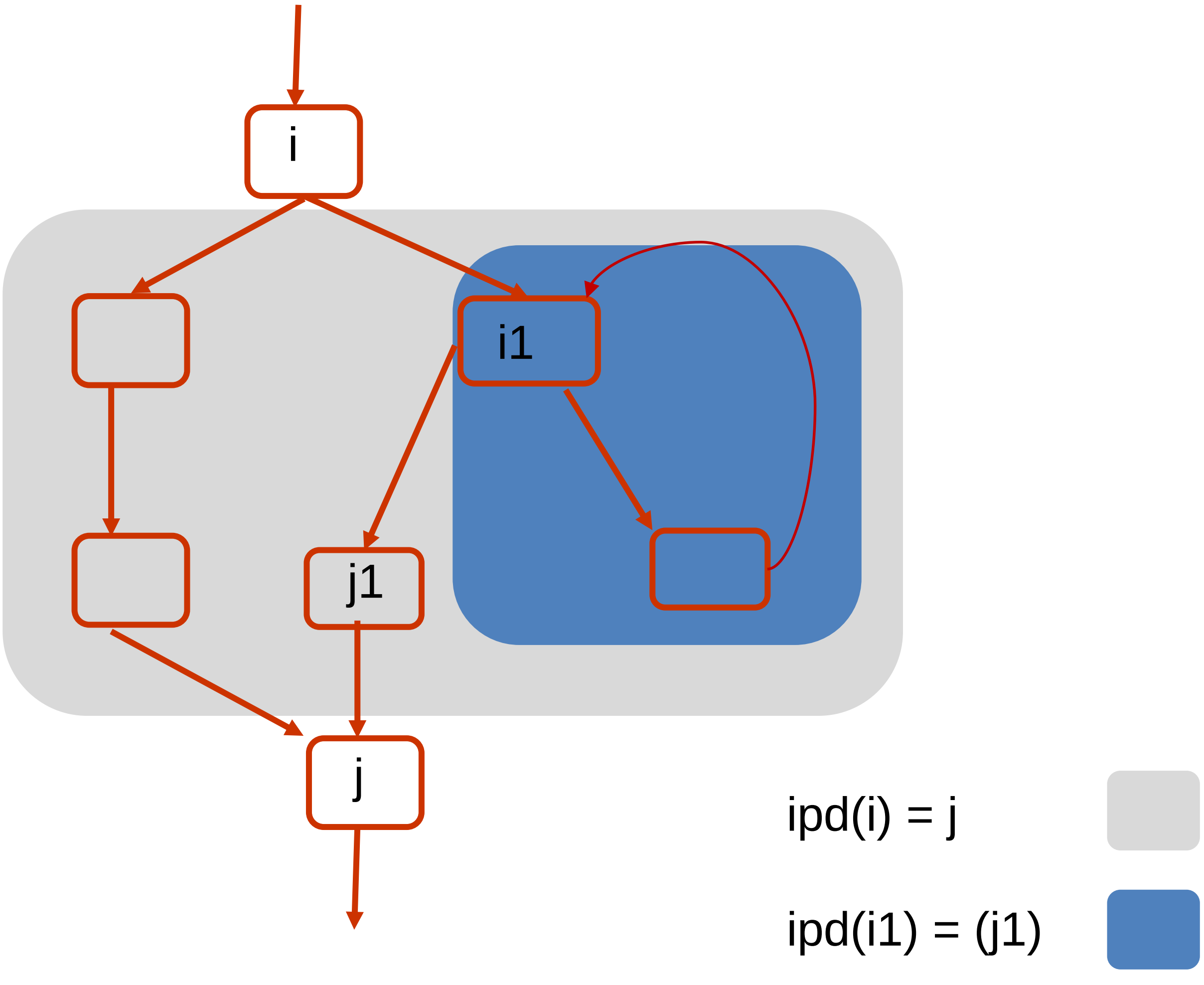
when an instruction j is executed: if the instruction j is the top of the ipd stack, the stack is updated by executing pop (j is removed from the stack)

…before i …….	Stack of ipd:	λ
i: control instruction	Stack of ipd:	ipd(i)
i1: control instruction	Stack of ipd:	ipd(i1) ipd(i)
j: top of the ipd stack	Stack of ipd:	ipd(i)
j: top of the ipd stack	Stack of ipd:	λ

CONTROL REGION of a branching instruction



Implicit flow



Basics of information flow

Influence of the implicit flow onto the operand stack

the stack may be manipulated in different ways by the branches of a branching instruction: they can perform a different number of pop and push operations, and with a different order.

The length and the content of the operand stack may be a means by which security leakages can occur

```
1  push 1
2  load x
3  if 5
4  pop
5  halt
```

The stack is empty or not, depending on the value of x

Basics of information flow

Secure Information flow

```
1  load x
2  if 5
3  push 1
4  goto 6
5  push 0
6  store y
7  halt
```

H={x}   L={y}

A program P= <c, H, L > satisfies secure information flow if the final value of each low variable does not depend on the initial value of the high variables.

Concrete Semantics

STATES $\mathcal{L} \times A \times \mathcal{M} \times \mathcal{S} \times \mathcal{A}^*$

$< \sigma, PC, M, S, \rho >$

$\boldsymbol{\sigma}$	security environment
PC	program counter
M	memory
S	operand stack (k,σ) …. (k',σ')
ρ	ipd stack (j, σ)…..(j', σ')

IPD Stack ρ

if $\rho = (j1, \boldsymbol{\sigma 1})$…..$(jn, \boldsymbol{\sigma}n)$
there are n open implicit flows
j1 holds the address where first implicit flow terminates
$\boldsymbol{\sigma 1}$ holds the level of the environment that must be restored

if $\rho = \lambda$
there are no open implicit flow

Transition relation rules

$$\texttt{load}\ \frac{\texttt{c[i]=}\ \textbf{\texttt{load x}}\ ,\quad M[x] = (k, \boldsymbol{\tau}),\ \ \text{not\_top}(i, \rho\ )}{<\boldsymbol{\sigma}\ , i, M, S, \rho > \ \rightarrow \ <\boldsymbol{\sigma}\ , i{+}1, M, (k, \boldsymbol{\sigma} \cup \boldsymbol{\tau}) \cdot S, \rho >}$$

Transition relation rules

$$\texttt{store}\ \frac{\texttt{c[i]=}\ \textbf{\texttt{store x}}\ ,\ \text{not\_top}(i, \rho)}{<\boldsymbol{\sigma}, i, M, (k, \tau)\cdot S, \rho> \rightarrow\ <\boldsymbol{\sigma}, i, M[(k, \boldsymbol{\sigma} \cup \tau)/\textbf{\texttt{x}}], S, \rho>}$$

Transition relation rules

$$\frac{\rho = (i, \tau) \,.\, \rho'}{\langle \sigma, i, M, S, (i, \tau) \,.\, \rho' \rangle \rightarrow \langle \tau, i, M, S, \rho' \rangle}\ \texttt{ipd}$$

i is the ipd of a control instruction

Transition relation rules

$$\texttt{goto}\ \frac{\texttt{c[i]=}\ \mathbf{goto\ j}\ ,\ \ \text{not\_top}(i, \rho)}{<\boldsymbol{\sigma}, i, M, S, \rho> \rightarrow <\boldsymbol{\sigma}, j, M, S, \rho>}$$

i is the ipd of a control instruction

Transition relation rules

$$\frac{\texttt{c[i]=}\ \mathbf{if\ j}\ ,\ \ \text{not\_top}(i, \rho)}{<\boldsymbol{\sigma}, i, M, (0, \tau)\cdot S, \rho> \ \longrightarrow\ <\boldsymbol{\sigma} \cup \tau, i+1, \ up(M), \ up(S), \ (ipd(i), \boldsymbol{\sigma})\ \rho>}\ \text{if-false}$$

An implicit flow begins, whose level is the least upper bound between the security environment (σ) and the security level of the condition of the `if` (τ). **The new security environment is** **($\sigma \cup \tau$)** **(ipd(pc), σ)** is pushed on the ipd stack ρ

up(M) upgrades the value of the variables assigned in the scope of the implicit flow beginning at PC

up(S) upgrades all elements in the stack

if : assume condition **non-zero**

Transition relation rules

$$\text{if-true}\ \frac{\texttt{c[i]= } \textbf{\texttt{if j , k!=0,}}\ \ \text{not\_top}(i, \rho)}{<\boldsymbol{\sigma}, i, M, (k, \boldsymbol{\tau}) \cdot S, \rho> \ \longrightarrow\ <\boldsymbol{\sigma} \cup \boldsymbol{\tau}, j,\ \ up(M),\ \ up(S),\ \ (ipd(i), \boldsymbol{\sigma}) . \rho>}$$

An implicit flow begins, whose level is the least upper bound between the security environment (σ) and the security level of the condition of the `if` (τ). **The new security environment is** **($\sigma \cup \tau$)**
(ipd(pc), σ) is pushed on the ipd stack ρ

up(M) upgrades the value of the variables assigned in the scope of the implicit flow beginning at i

up(S) upgrades all elements in the stack

Abstract operational semantics

the abstract semantics:

• abstracts concrete values into their security level:

$$\alpha\ (k,\sigma)=\sigma$$

• uses the same rules of the concrete semantics on the abstract domains

Both rules for if are always applied - if_{true} if_{false}

A(P) : abstract transition system for P

• finite
• multiple path
• each path of C(P) is correctly abstracted onto a path of A(P)

Results

Theorem 1

A program P satisfies SIF if for each state of A(P) such that c[i] = halt , then for each x in L it is:

M[x] = L (value)
or
M[x]=(i, L) for some i (address)

An example: concrete semantics

x:(0,H) y:(1,L)
ipd(2) = 5, ipd(6)=10

<ENV, PC, [M(x), M(y)], Stack, IPDstack>

```
1    load y
2    if 5
3    push 3
4    store x
5    load x
6    if 9
7    push 1
8    goto 10
9    push 0
10   store y
11   halt
```

$$\langle L, 1, [(0, H)(1, L)], \lambda, \lambda\rangle$$
$$\downarrow \textbf{load}$$
$$\langle L, 2, [(0, H)(1, L)], (1, L), \lambda\rangle$$
$$\downarrow \textbf{if}_{\textbf{true}}$$
$$\langle L, 5, [(0, H)(1, L)], \lambda, (5, L)\rangle$$
$$\downarrow \textbf{ipd}$$
$$\langle L, 5, [(0, H)(1, L)], \lambda, \lambda\rangle$$
$$\downarrow \textbf{load}$$
$$\langle L, 6, [(0, H)(1, L)], (0, H), \lambda\rangle$$
$$\downarrow \textbf{if}_{\textbf{false}}$$
$$\langle H, 7, [(0, H)(1, L)], \lambda, (10, L)\rangle$$
$$\downarrow \textbf{push}$$
$$\langle H, 8, [(0, H)(1, L)], (1, H), (10, L)\rangle$$
$$\downarrow \textbf{goto}$$
$$\langle H, 10, [(0, H)(1, L)], (1, H), (10, L)\rangle$$
$$\downarrow \textbf{ipd}$$
$$\langle L, 10, [(0, H)(1, L)], (1, H), \lambda\rangle$$
$$\downarrow \textbf{store}$$
$$\langle L, 11, [(0, H)(1, H)], \lambda, \lambda\rangle$$

Abstract semantics

$M_0^{\natural}(x) = H$ and $M_0^{\natural}(y) = L$.

$$\langle L, 1, [H\ L], \lambda, \lambda\rangle \xrightarrow{\text{load}} \langle L, 2, [H\ L], L, \lambda\rangle$$

From $\langle L, 2, [H\ L], L, \lambda\rangle$:

$$\xrightarrow{\text{if}_{\text{false}}} \langle L, 3, [H\ L], \lambda, (5, L)\rangle \xrightarrow{\text{push}} \langle L, 4, [H\ L], L, (5, L)\rangle \xrightarrow{\text{store}} \langle L, 5, [L\ L], \lambda, (5, L)\rangle \xrightarrow{\text{ipd}} \langle L, 5, [L\ L], \lambda, \lambda\rangle \xrightarrow{\text{load}} \langle L, 6, [L\ L], L, \lambda\rangle$$

From $\langle L, 6, [L\ L], L, \lambda\rangle$:

$$\xrightarrow{\text{if}_{\text{false}}} \langle L, 7, [L\ L], \lambda, (10, L)\rangle \xrightarrow{\text{push}} \langle L, 8, [L\ L], L, (10, L)\rangle \xrightarrow{\text{goto}} \langle L, 10, [L\ L], L, (10, L)\rangle$$

$$\xrightarrow{\text{if}_{\text{true}}} \langle L, 9, [L\ L], \lambda, (10, L)\rangle \xrightarrow{\text{push}} \langle L, 10, [L\ L], L, (10, L)\rangle$$

$$\langle L, 10, [L\ L], L, (10, L)\rangle \xrightarrow{\text{ipd}} \langle L, 10, [L\ L], L, \lambda\rangle \xrightarrow{\text{store}} \langle L, 11, [L\ L], \lambda, \lambda\rangle$$

$$\xrightarrow{\text{if}_{\text{true}}} \langle L, 5, [H\ L], \lambda, (5, L)\rangle \xrightarrow{\text{ipd}} \langle L, 5, [H\ L], \lambda, \lambda\rangle \xrightarrow{\text{load}} \langle L, 6, [H\ L], H, \lambda\rangle$$

From $\langle L, 6, [H\ L], H, \lambda\rangle$:

$$\xrightarrow{\text{if}_{\text{false}}} \langle H, 7, [H\ L], \lambda, (10, L)\rangle \xrightarrow{\text{push}} \langle H, 8, [H\ L], H, (10, L)\rangle \xrightarrow{\text{goto}} \langle H, 10, [H\ L], H, (10, L)\rangle$$

$$\xrightarrow{\text{if}_{\text{true}}} \langle H, 9, [H\ L], \lambda, (10, L)\rangle \xrightarrow{\text{push}} \langle H, 10, [H\ L], H, (10, L)\rangle$$

$$\langle H, 10, [H\ L], H, (10, L)\rangle \xrightarrow{\text{ipd}} \langle L, 10, [H\ L], H, \lambda\rangle \xrightarrow{\text{store}} \langle L, 11, [H\ H], \lambda, \lambda\rangle$$

Colluding apps

Java cards:

Secure interactions in Java cards

Java cards

- Smart cards: embedded systems that allow to store and process information

- Typical Aplications: Credit cards, Electronic cash, Loalty systems, Helthcare, Government identification ....

- Java cards:
  Java Virtual machine / applications (applets) are portable

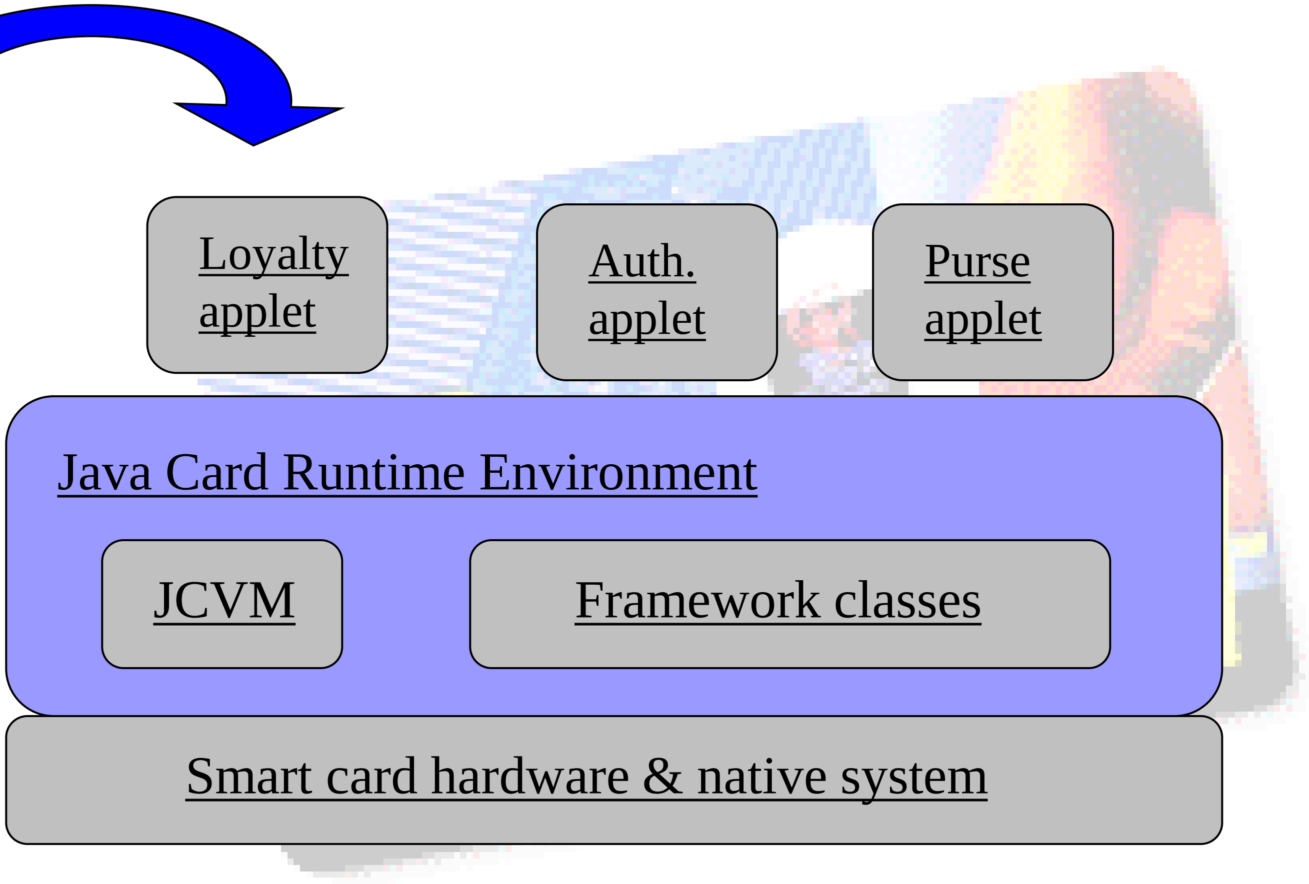
  - Multiapplicative Java cards: applets can be downloaded and installed on card after the card issuance

  - Applet’s sensitive data must be protected against anouthorised accesses

Java cards

Card reader

Multiapplicative Java cards



Java cards security

- Security in Java cards is a combination of the security mechanisms in Java and additional security procedures imposed by the card platform

FIREWALL

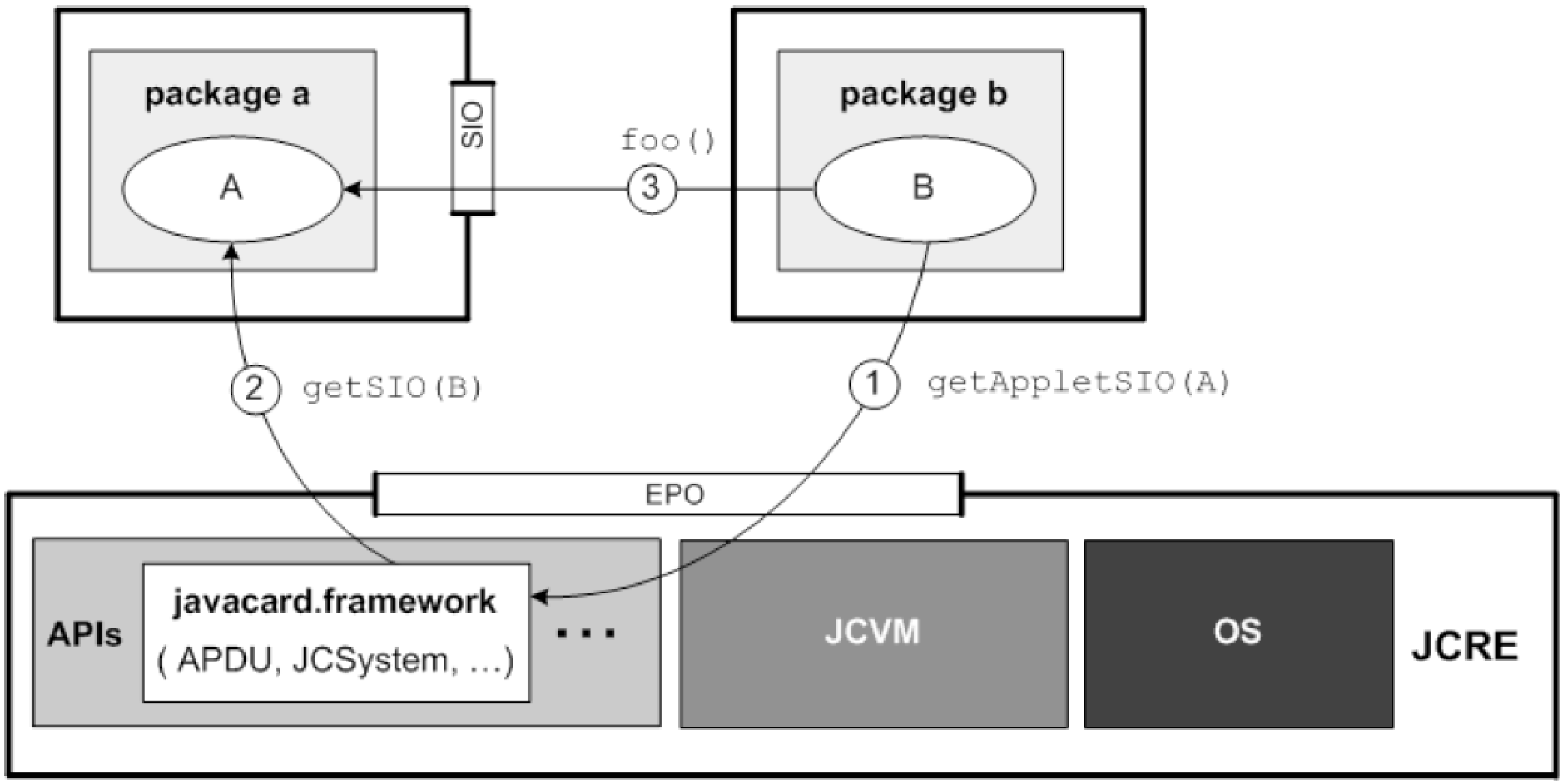
ATOMICITY and TRANSACTIONS

PERSISTENT and TRANSIENT objects

JAVA security mechanisms

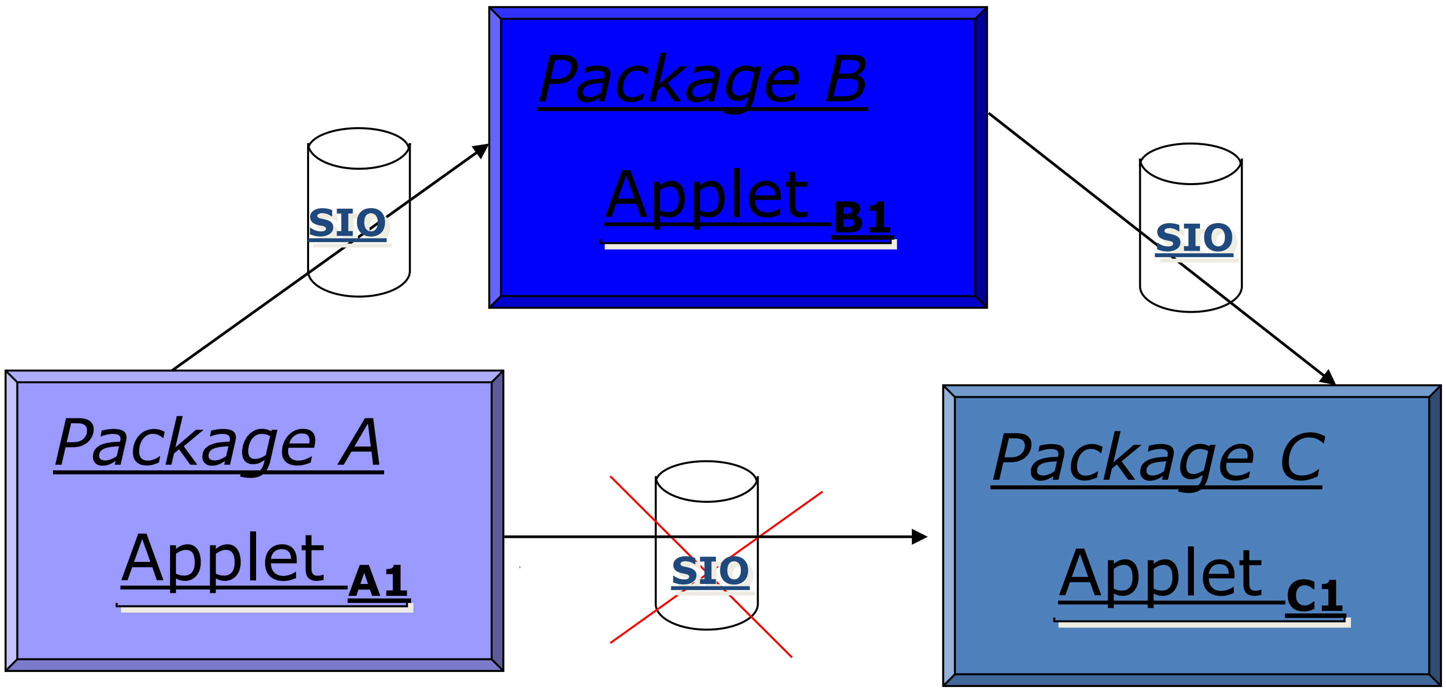
- The Firewall forces the isolation between objects of applets belonging to different packages

Communication between packages



Limits of the firewall

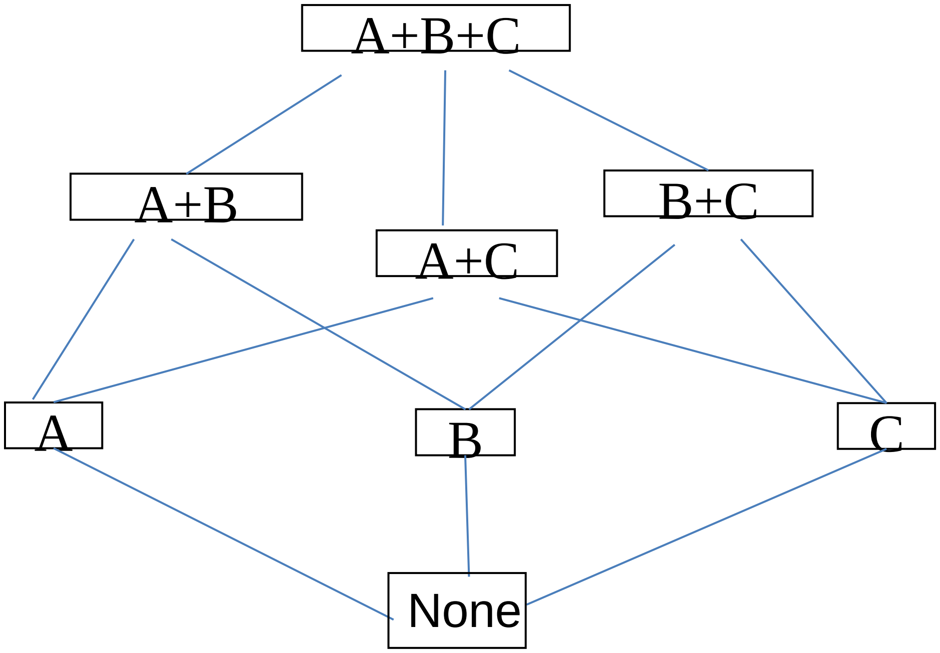
- Based on access control checks
- Place restrictions on the applets that can access to methods of applets belonging to other packages
- Does not control the propagation of the information from an applet of a package towards applets of other packages



Secure Information Flow

Security levels assigned to packages

Lattice of security levels



Secure Information Flow: check that information exchanged between

- A and B has a security level equal to or lower than A+B
- A and C has a security level equal to or lower than A+C
- B and C has a security level equal to or lower than B+C

Java Card Information Flow Verifier

JCIFV performs the analysis according to the following main steps

1. Unique security levels are automatically assigned to packages and shareable interface objects. An initial security level is assigned to the other methods and object fields

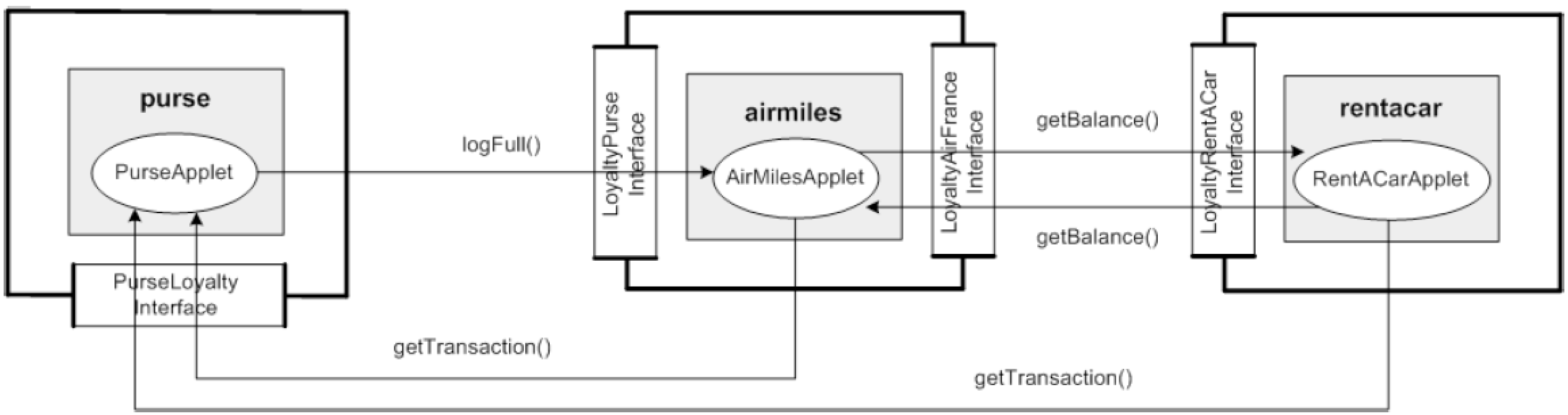
2. CAP file (native code of an applet) is decoded and saved as a bytecode

3. Abstract interpretation of the bytecode is performed

4. The analysis stops when the state of the abstract interpreter does not longer change and all methods have been analyzed

5. Secure information flow is checked

Electronic Purse



Purse: log-full service (logFull()), which notifies registered applets that the transaction log is going to be over-written

.

Airmail: registered for the log-full service

RentACar: not registered for the log-full service

Electronic Purse

Assume that AirFrance requests RentACar the amount of miles (getBalance()) every time Purse notifies AirFrance that the transaction log is full.

logFull() method implemented by AirFrance contains an invocation of method getTransaction() of Purse followed by an invocation of method getBalance() of RentACar.

Applet RentaACar, whenever observes an invocation of getBalance(), can infer that Purse is going to over-write the transaction log.

Thus, even without subscribing to the log-full service, RentACar is able to benefit from such a service.

Purse is not able to detect such information flow.

illicit information flow from Purse to RentACar caused by a method invocation (no parameters) from AirMiles and RentAcar