Theoretical Corner: The *Non-Interference* Property

Marwan Burelle

marwan.burelle@lse.epita.fr
http://wiki-prog.infoprepa.epita.fr

LSE Security System
Laboratory of Epita

Outline

Introduction

Non-Interference ?

What the Hell is that ?

Non-Interference ?

It's a formal property about the link between input and output channels of an information system.

Non-Interference ?

- Used to verify security model in information system
- Used to analyse information flow in programs
- Used to verify that a parallel system is determinist

Non-Interference ?

- Used to verify security model in information system
- Used to analyse information flow in programs
- Used to verify that a parallel system is determinist

Non-Interference ?

- Used to verify security model in information system
- Used to analyse information flow in programs
- Used to verify that a parallel system is determinist

Theory And Security

Formal Security ?

- We need to define what we want:
- We need to define how to enforce it:
- We need to verify that it works:

Formal Security ?

- We need to define what we want: **Security Policy**
- We need to define how to enforce it:
- We need to verify that it works:

Formal Security ?

- We need to define what we want: **Security Policy**
- We need to define how to enforce it: **MAC, ACL, . . .**
- We need to verify that it works:

Formal Security ?

- We need to define what we want: **Security Policy**
- We need to define how to enforce it: **MAC, ACL, . . .**
- We need to verify that it works: ***Oh ! . . . shit . . .***

Models And Policies

Security Policy ?

- **Military:** like **Bell-LaPadula** model
- **Commercial:** like **Chinese Wall** model

Bell & LaPadula

- Latice of security labels
- Subjects (users or programs) have an upper bound security level called *security clearance*.
- Objects have a level that can only be raised
- Subject s can make a *read* access to object o, if and only if: $clearance(o) \leq clearance(s)$

  No read-up !
- s can make a *write* access to o, if and only if: $clearance(s) \leq clearance(o)$

  No write-down !

Bell & LaPadula

- Latice of security labels
- Subjects (users or programs) have an upper bound security level called *security clearance*.
- Objects have a level that can only be raised
- Subject s can make a *read* access to object o, if and only if: $clearance(o) \leq clearance(s)$

  No read-up !

- s can make a *write* access to o, if and only if: $clearance(s) \leq clearance(o)$

  No write-down !

Bell & LaPadula

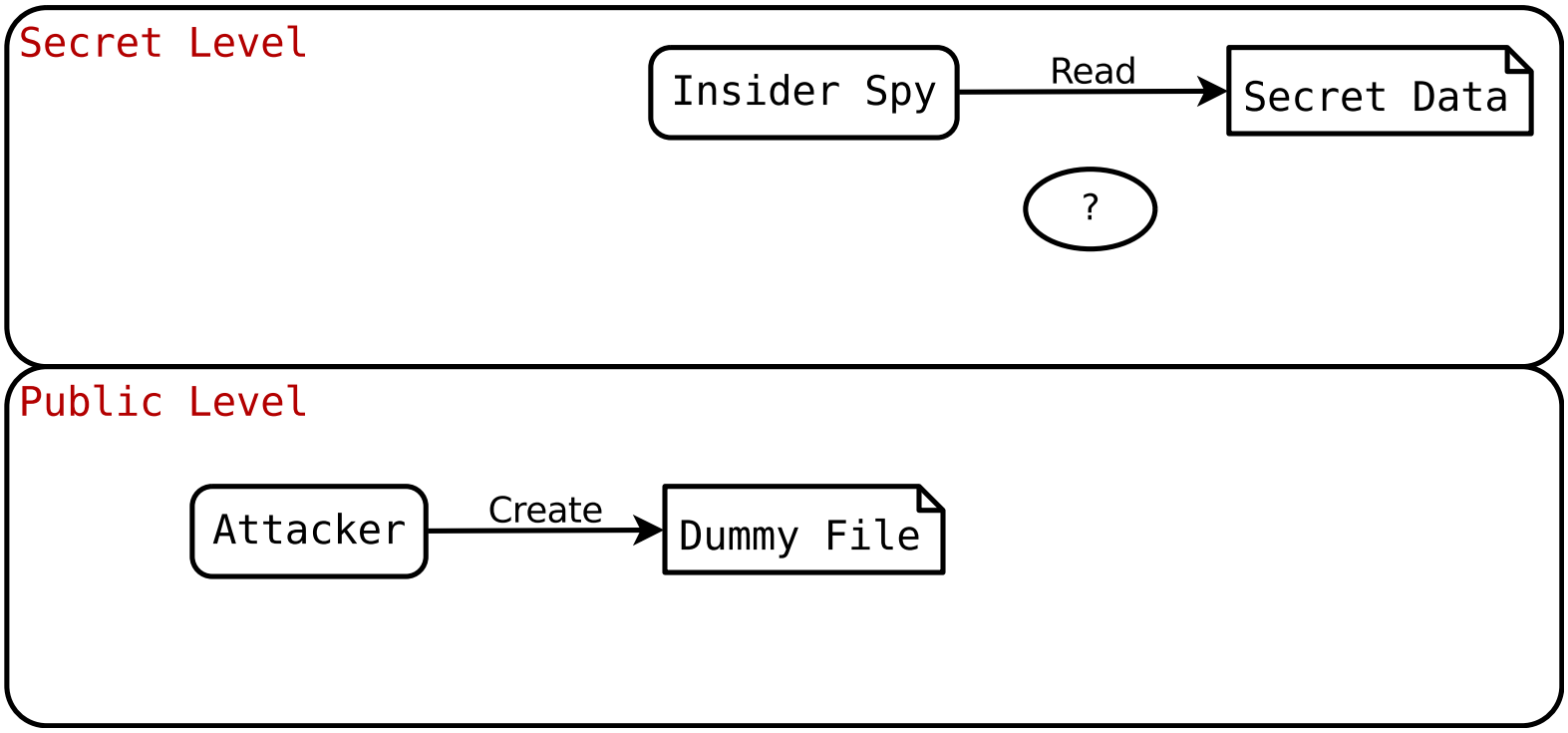
- Latice of security labels
- Subjects (users or programs) have an upper bound security level called *security clearance*.
- Objects have a level that can only be raised
- Subject s can make a *read* access to object o, if and only if:
  $clearance(o) \leq clearance(s)$
  No read-up !
- s can make a *write* access to o, if and only if:
  $clearance(s) \leq clearance(o)$
  No write-down !

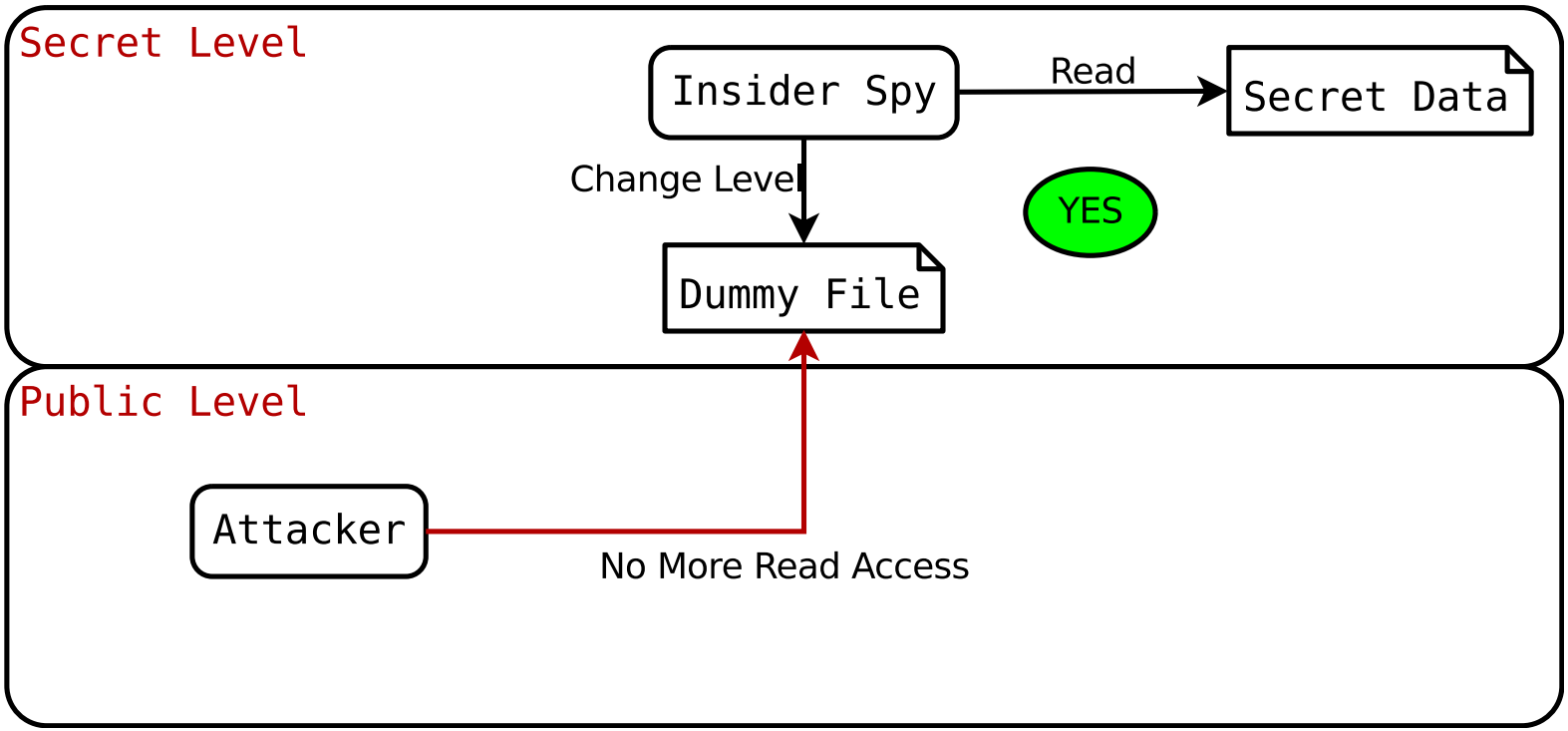
Chinese Wall Security

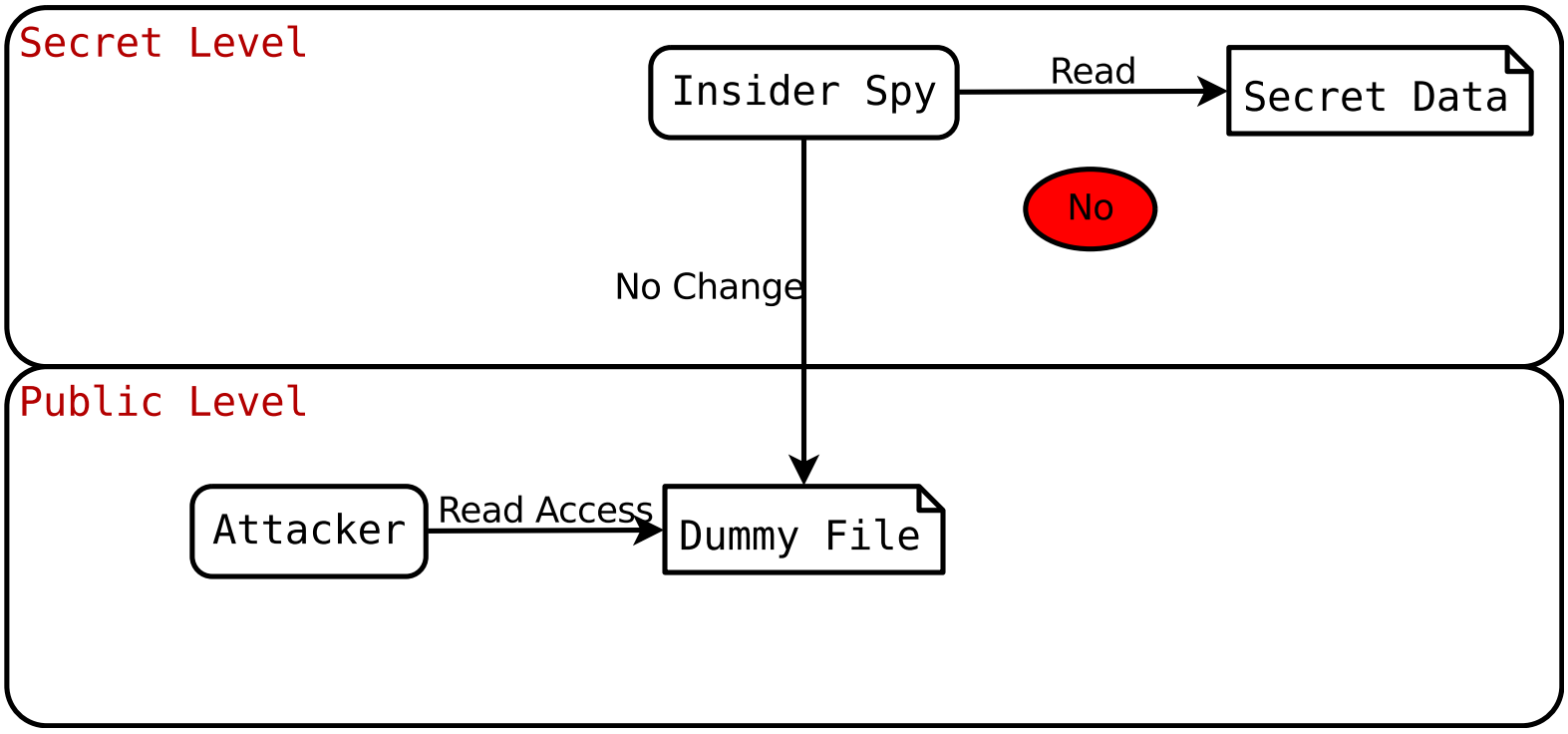
- Dynamic policy based on access history
- Try to prevent information leak between conflicting data sets
- A subject can't write to some data set if it has ever had access to another conflicting set.
- Conflicting state can be *inherited*: when a subject write to a set, it transmits conflicts to that set.

Indirect Information Flow (cover channels) ?

- Authorization can be an information channel
- There's a lot of indirect way to transmit information
- Bell&LaPadula are subject to a cover channel using access control
- Most models enforce their policy in a limited scope (direct information flow, over simplified operations descriptions . . .)


Secret Level
Insider Spy
Read
Secret Data
?
Public Level
Attacker
Create
Dummy File


Secret Level
Insider Spy
Read
Secret Data
Change Level
YES
Dummy File
Public Level
Attacker
No More Read Access


Secret Level
Insider Spy
Read
Secret Data
No
No Change
Public Level
Attacker
Read Access
Dummy File

Non-Interference

Security Policy ?

All these models cover different cases can't be expressed with each other.

We need a more powerfull property

Security Policy ?

All these models cover different cases can't be expressed with each other.

We need a more powerfull property

Security Policy and Security Models

One group of users, using a certain set of commands, is noninterfering with another group of users if what the first group does with those commands has no effect on what the second group of users can see.

– *Goguen and Meseguer (1982)*

Non-Interference

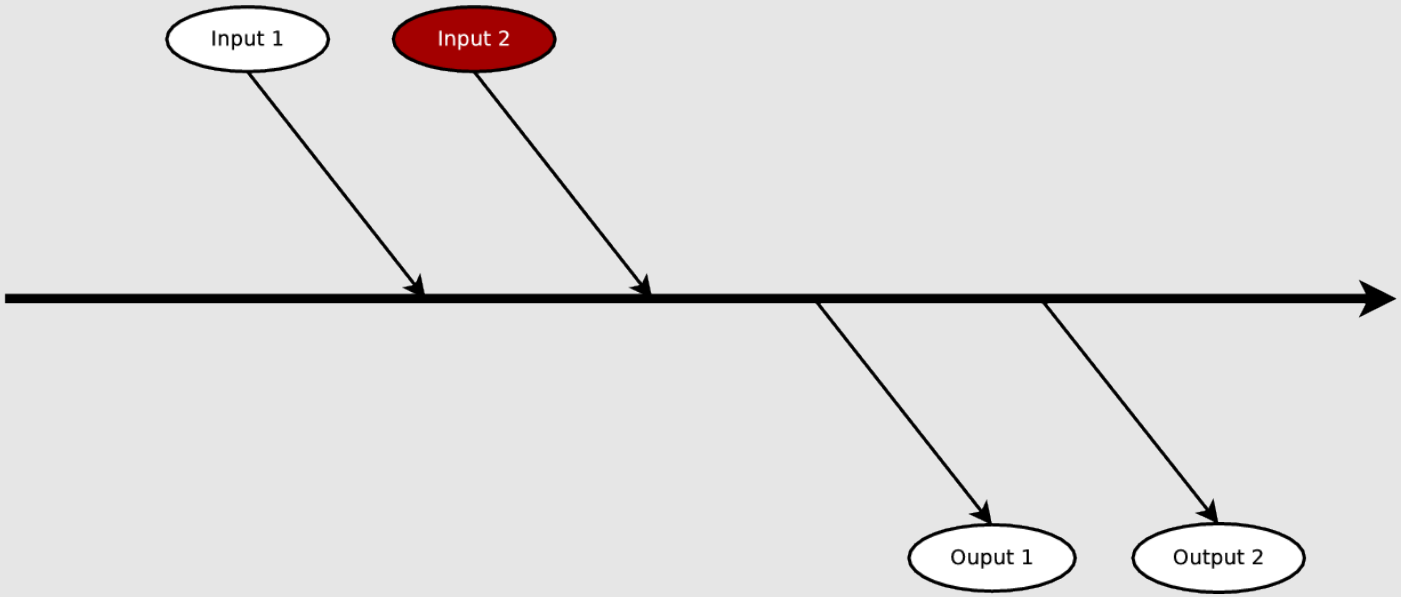
When observing a given *output channel*, if you can't see changes to another *input channel*, you can't gather information from it !

Non-Interference

- Given an input channel A and an output channel B, they are not interfering if for any possible input values on A (all others input channels being fixed) the output value on B won't change.
- Using trace theory: if we only observe B outputs, we can distinguish variations in A inputs.
- From a security point of view: public output doesn't depend on private input.

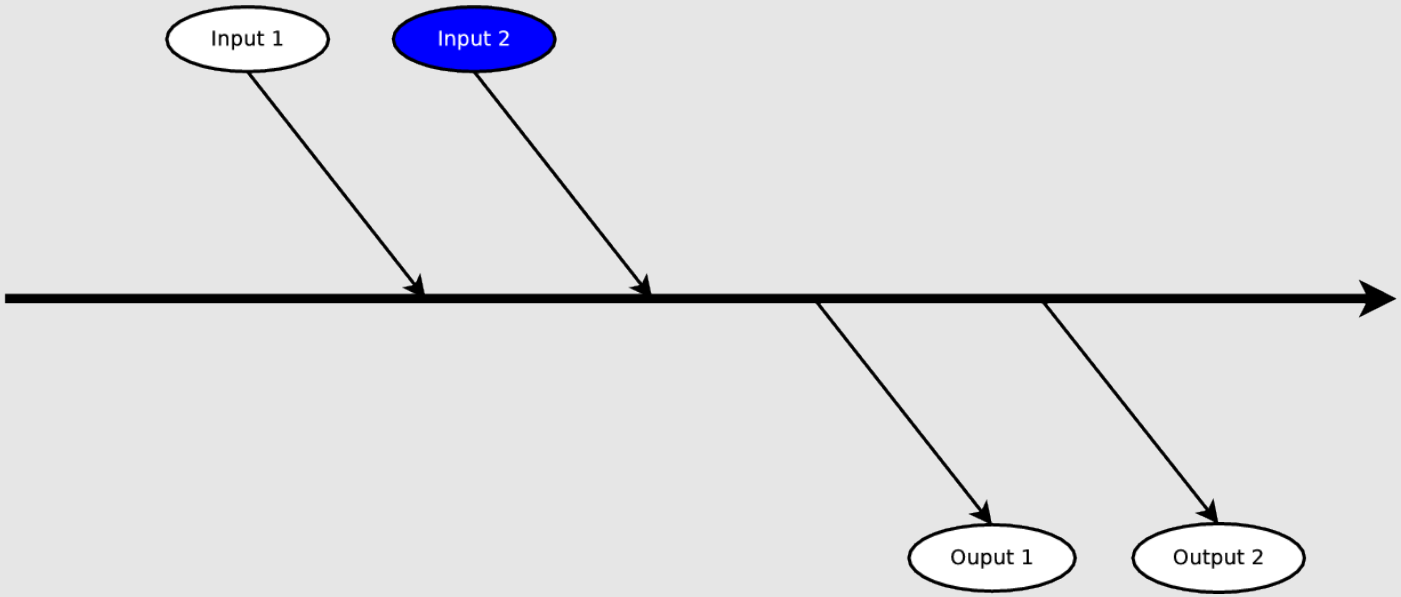
Non-Interference

Processus traces



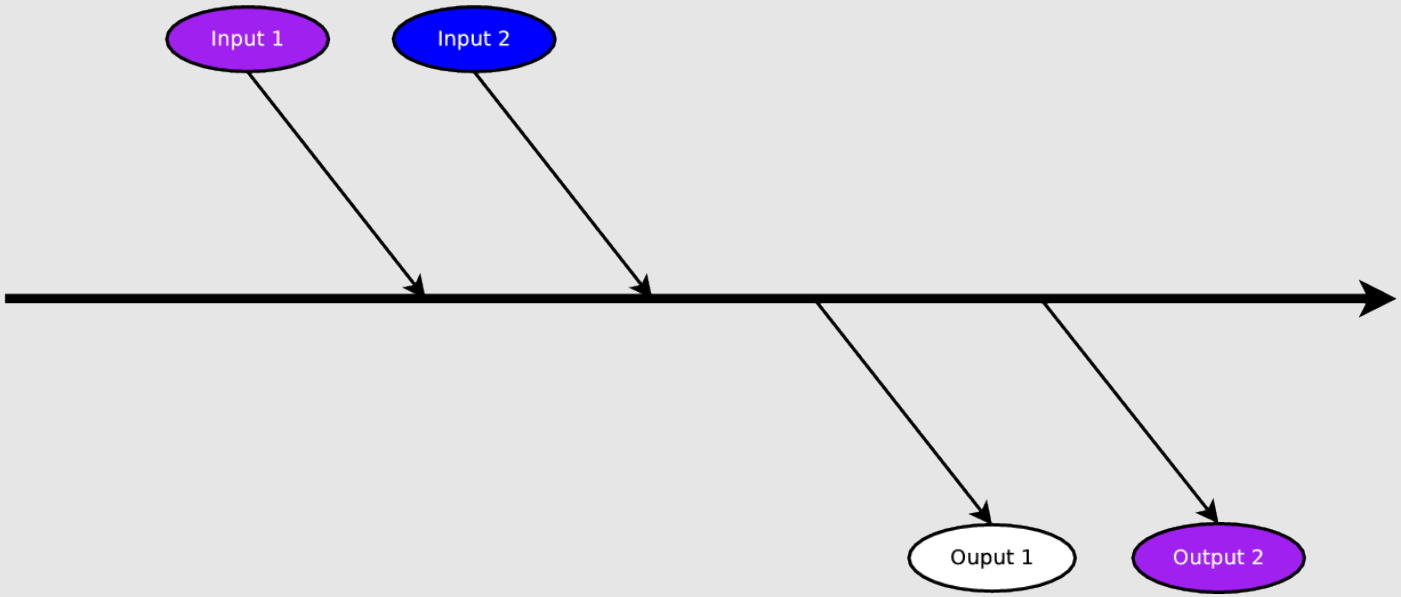
Non-Interference

Processus traces



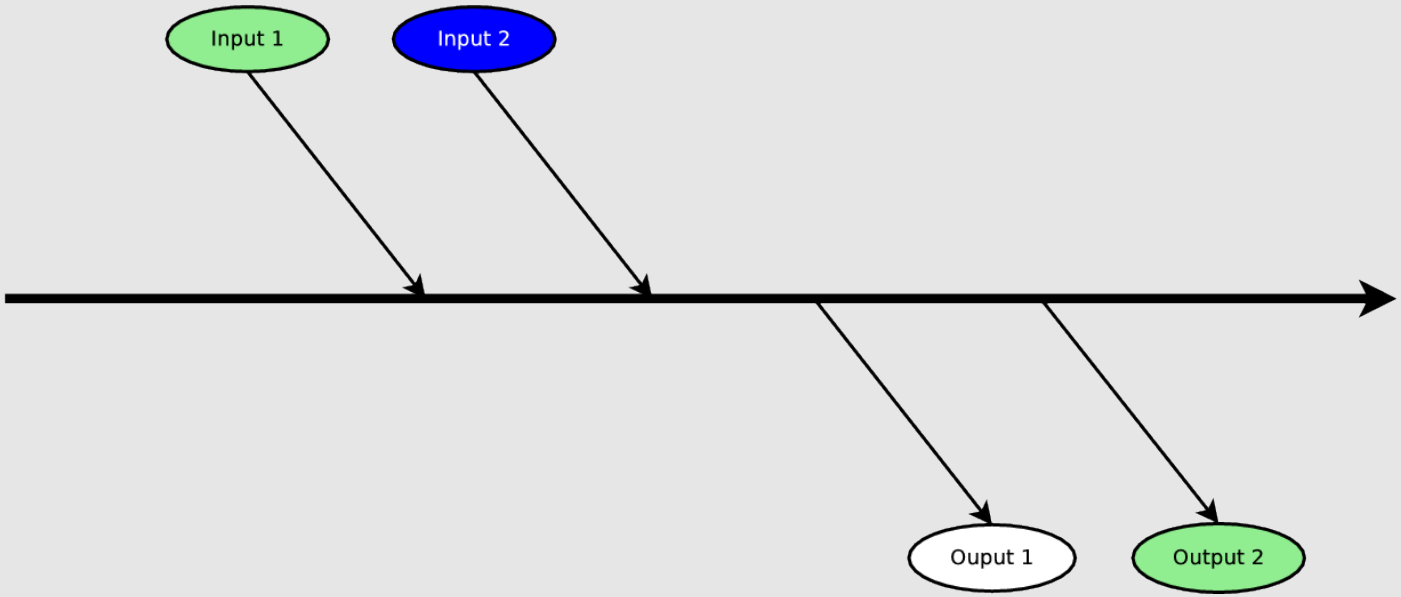
Non-Interference

Processus traces



Non-Interference

Processus traces



Flow Analysis

Non-Interference In A Functionnal World

Let t be a $\lambda-$term, δ an occurrence in t and t_0 the sub-term occurring at δ. We note $\mathscr{C}_t^\delta[]$ the context surrounding t_0 and $\mathscr{C}_t^\delta[t_1]$ is the term t where t_0 have been replaced by t_1.

t_0 is non-interfering in t, if:

$$\forall t_i, t \rightarrow^* v \Rightarrow \mathscr{C}_t^\delta[t_i] \rightarrow^* v$$

NI And Functionnal Language

Since data and code are one, checking for NI is equivalent to dead code detection.

Tracking flow with labels

Most flow analysis for languages derived from $\lambda-$calculus use *labels*: sub-terms are marked with labels which are propagated through the reduction process.

Using labels

$$\Big(\big(\lambda x.\lambda y.x\big)(\ell_1 : v_1)\Big)(\ell_2 : v_2)$$

using small step operational semantics

Using labels

$$\Big(\big(\lambda x.\lambda y.x\big)\,(\ell_1 : v_1)\Big)\,(\ell_2 : v_2)$$

$$\rightarrow \Big(\lambda y.\,(\ell_1 : v_1)\Big)\,(\ell_2 : v_2)$$

Using labels

$$\Big(\big(\lambda x.\lambda y.x\big)(\ell_1 : v_1)\Big)(\ell_2 : v_2)$$

$$\rightarrow (\ell_1 : v_1)$$

Using labels

$$\Big(\big(\lambda x.\lambda y.x\big)(\ell_1 : v_1)\Big)(\ell_2 : v_2)$$

Obviously, v_2 is non-interfering, while v_1 is.

Catching *Code Flow*

$$\Big(\ell : (\lambda x.e_0)\Big)\, e_1$$

Catching *Code Flow*

$$\Big(\ell : (\lambda x.e_0)\Big)\, e_1$$

we want to remember the fact that the function $(\lambda x.e_0)$ *was apply to* e_1

Catching *Code Flow*

$$\Big(\ell : (\lambda x.e_0)\Big)\, e_1$$

$$\rightarrow \ell : \Big((\lambda x.e_0)\, e_1\Big)$$

Catching *Code Flow*

$$\Big(\ell : (\lambda x.e_0)\Big)\, e_1$$

$$\rightarrow^* \ell : v$$

$$with \Big((\lambda x.e_0)\, e_1\Big) \rightarrow^* v$$

Theorem (Non-Interference in labeled calculus)

Let t be a term and t_0 a sub-term of t of the form $\left(\ell : t'_0\right)$, if $t \rightarrow^* v$ and ℓ does not appear in v, then t_0 is non-interfering in t.

Static Analysis ?

Labeled calculus provides a dynamic technique but can also be used to build a static types system.

Static Analysis ?

Volpano&Smith introduced a simple types system for a *while language* that support side effects.

Static Analysis ?

$$\frac{\Gamma \vdash x : \ell' \text{ var} \quad \Gamma \vdash e : \ell \quad \ell \leq \ell'}{\Gamma \vdash x \leftarrow e : \ell \text{ cmd}}$$

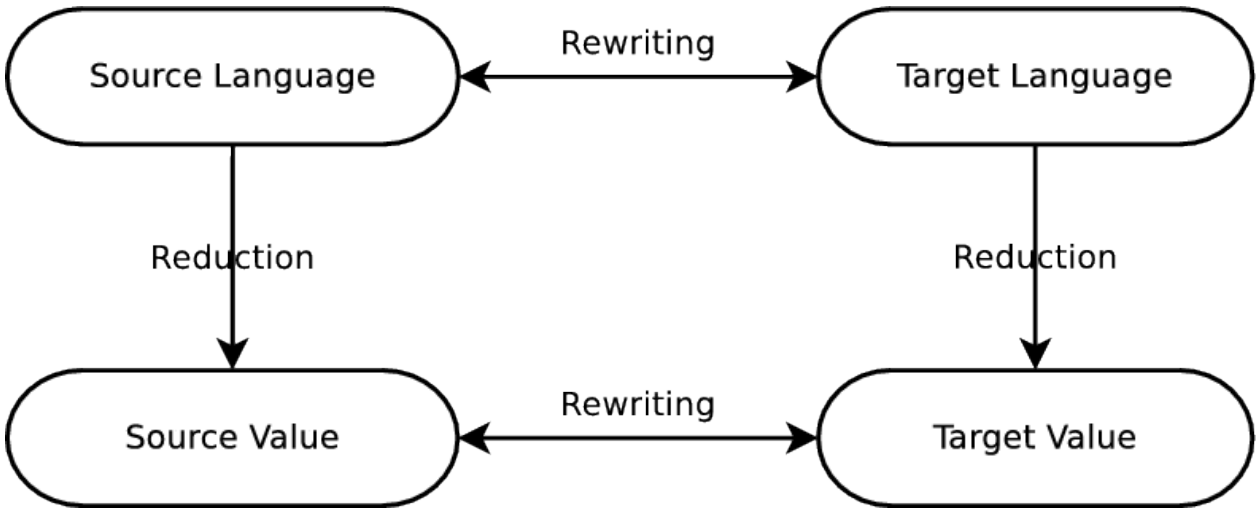
$$\frac{\Gamma \vdash e : \ell' \quad \Gamma \vdash s_0 : \ell' \text{ cmd} \quad \Gamma \vdash s_1 : \ell' \text{ cmd} \quad \ell \leq \ell'}{\Gamma \vdash \texttt{if } e \texttt{ then } s \texttt{ else } s' : \ell \text{ cmd}}$$

Static Analysis ?

The full types system is sound and one can implement an inference mechanism over it.

Static Analysis ?

Pottier&Conchon designed a system based on *rewriting* to gain *Information Flow Inference For Free*.



A Complete System: Flow Caml

A Flow Caml Example

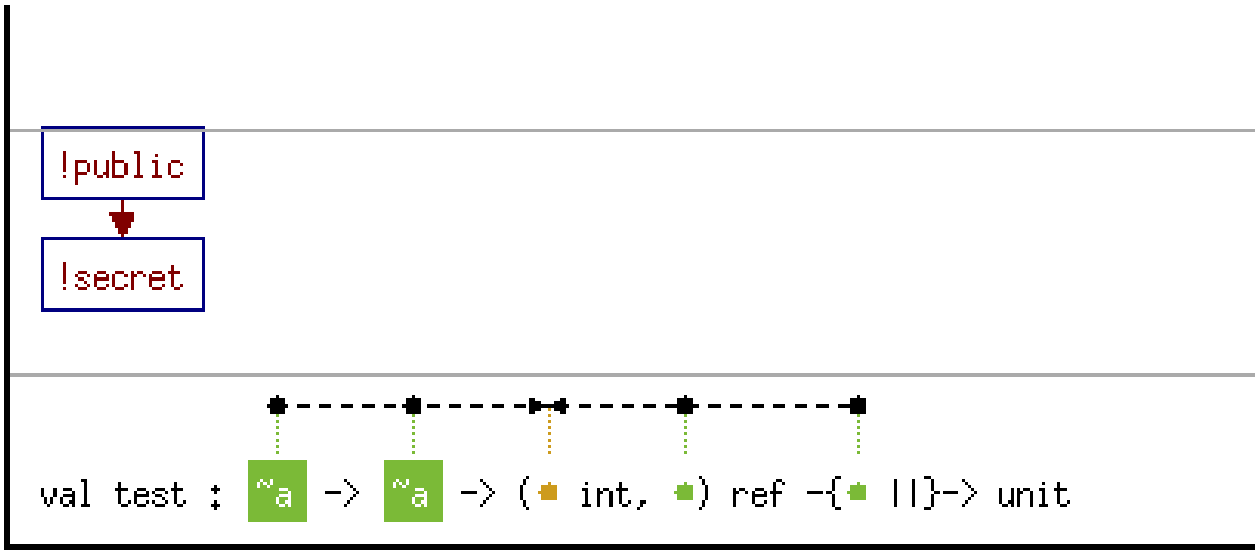
```
flow !public < !secret;;

let test a b r =
  if a = b then r := 1
  else r := 2;;

let a : !secret int = 42;;
let b : !public int = 42;;
let r : (!public int, 'a) ref = ref 0;;

test a b r;;
```

A Complete System: Flow Caml



A Complete System: Flow Caml

```
$ flowcamlc example.fml
File "example.fml", line 11, characters 0-10:
This expression generates the following information flow:
  !secret < !public
which is not legal.
```

What About R-Types ?

- Flow analysis can be extended to support R-Types (like in CDuce or XDuce.)
- Since types are leading execution, they must be integrate in the Non-Interference property.

Non-Interference With R-Types

Let e be an expression, e_0 a sub-expression occurring at δ and t a type such that $e_0 : t$. e_0 is non-interfering *w.r.t.* t in e if (and only if):

$$\forall e_i : t, e \rightarrow^* v \Rightarrow \mathscr{C}_e^{\delta}[e_i] \rightarrow^* v$$

What About R-Types ?

- Languages with R-Types, semantic subtyping and type based pattern matching can also be extend in a labeled form.
- The label mechanism is conservative (reduction with and without labels yield the same result.)
- One can build a type system and an inference algorithm to perform a static flow analysis.
- Issues:
  - Since language like CDuce and XDuce provides overloading, we need an inference for overloaded functions an open (almost) issue.
  - Classical systems (like **ML** or **HM(X)**) are too restrictive.

Issue With Constraint Based Inference

The term:

$$\lambda f\ x\ y.(f\ x, f\ y)$$

Has type:

$$\forall[\alpha_0 \leq \alpha_2, \alpha_1 \leq \alpha_2].\alpha_0 \to \alpha_1 \to (\alpha_2 \to \alpha_3) \to \alpha_3 \times \alpha_3$$

The forced unification of the second and third parameters is too restrictive for a flow analysis.

Application To Parallelism

Parallelism ?

NI identify links and flow. And parallelism issues are all matters of links and flow.

Task Oriented Programming

Example:

```
task[res] fib(x) {
 if (x < 2) {
  res = x;
  return;
 }
 var r1=0, r2=0;
 f1<-invoke[r1]::fib(x-1);
 f2<-invoke[r2]::fib(x-2);
 wait(f1);
 wait(f2);
 res = r1 + r2;
 return;
}
```

Prototype Language
Basic integer arithmetic
Spawn'n'wait task
Explicit shared vars

Tasks

- Tasks have a set of input variables (IN) and a set of output variables (OUT)
- Two task t_1 and t_2 are non-interfereing if:
  $\texttt{IN}(t_1) \cap \texttt{OUT}(t_2) = \texttt{IN}(t_2) \cap \texttt{OUT}(t_1) = \texttt{OUT}(t_1) \cap \texttt{OUT}(t_2) = \emptyset$
- We can safely execute concurrently non-interfering tasks.

Proof Of Concept

- I use a type inference algorithm to build input and output set for all tasks based on derived form of the Hindley/Milner unification algorithm.
- Finally, using infered types, I'm able to verify that:
  - no two interfering activities occurs;
  - no task has depencies on variables that may get out-of-scope before the end of the task;
- The available information can also help for placing barriers, re-ordering operations or moving operations on *local storage* (no shared locations such as variables or registers.)

Example:

```
task[a] f() {
  a = 42;
}

task[] main() {
  var y = 0, x = 0;
  r <- invoke[x]::f();
  y = x; // R/W conflict with task f
  wait(r);
  y = x; // no conflict here
}
```

Example:

```
task[a] f0() {
  a = 42;
}
task[] f1(x) {
  if (x > 0) {
    var y = x;
    r <- invoke[y]::f0();
  } else {
    var y = -x;
    r <- invoke[y]::f0();
  }
  wait(r); // Scope Conflict: y has been dropped
}
```

Going Further ?

- A more realistic language with real features
- **Pointers :** a lot of trouble with aliasing
- Code generation: basic approach using thread spawning and joining is unrealistic, we need a real task scheduling mechanism.
- Apply this to a real language: we need to define the task mechanism, circumvent usage of aliasing and define whether protection mechanism (like mutex) are interference free or not.