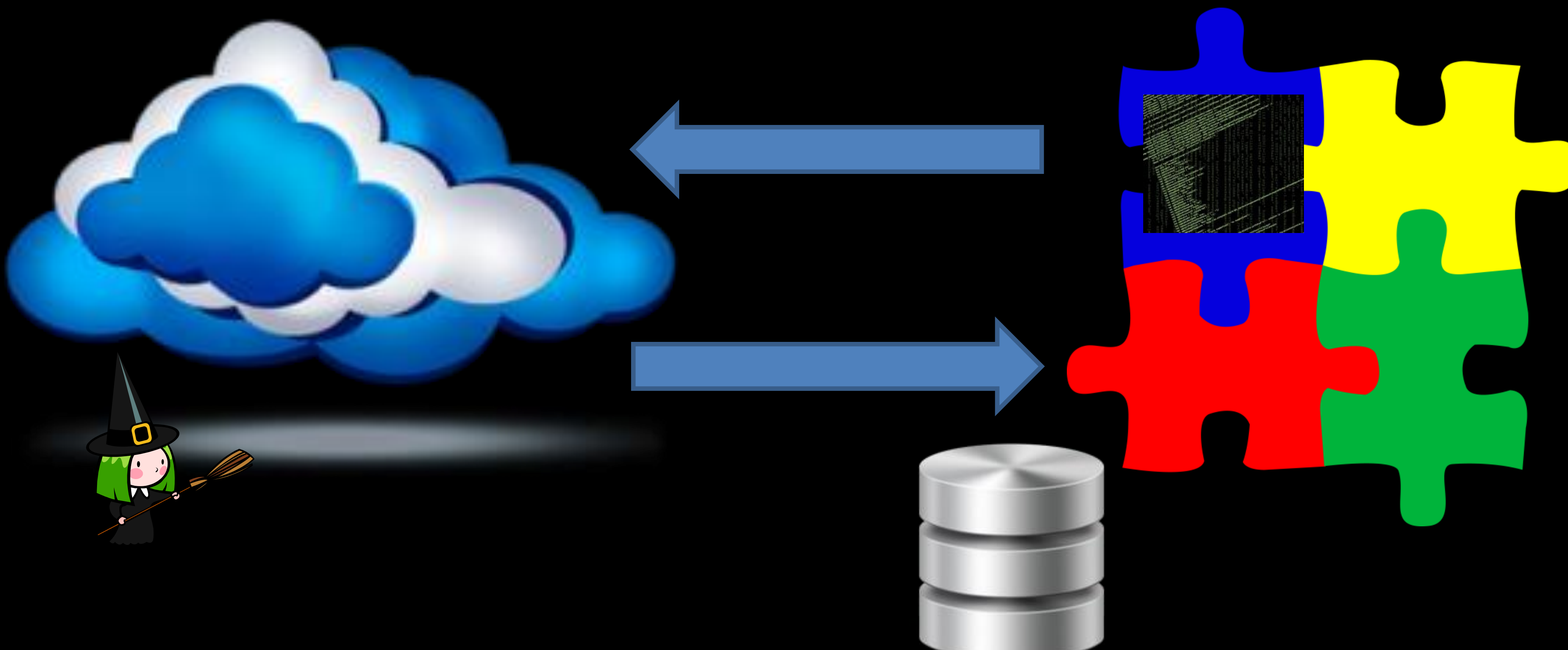
An information-flow calculus for the non-security expert

Alejandro Russo (russo@chalmers.se)
Visiting Associate Professor, Stanford, CA, U.S.A.
Chalmers, Göteborg, Sweden

Work-in-progress with Pablo Buiras (Chalmers), Deian Stefan (Stanford), and David Mazierès (Stanford)

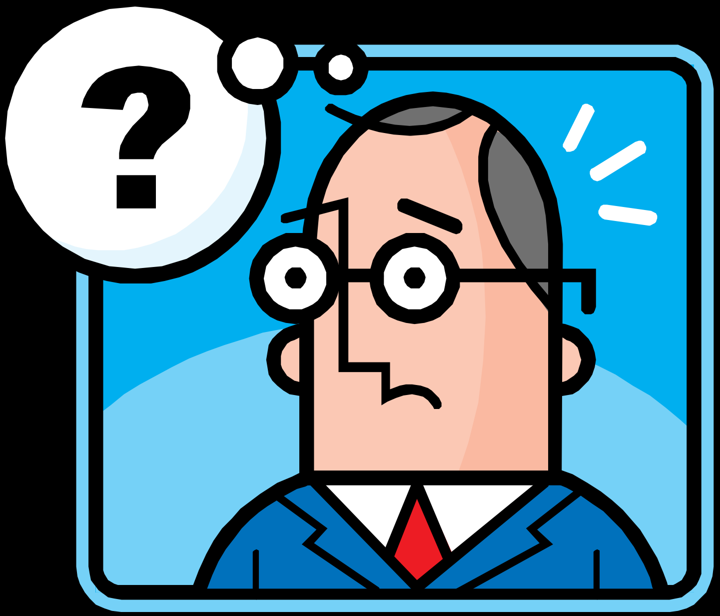
Information-flow Scenario

Preserve confidentiality even in the presence of malicious code



Motivation

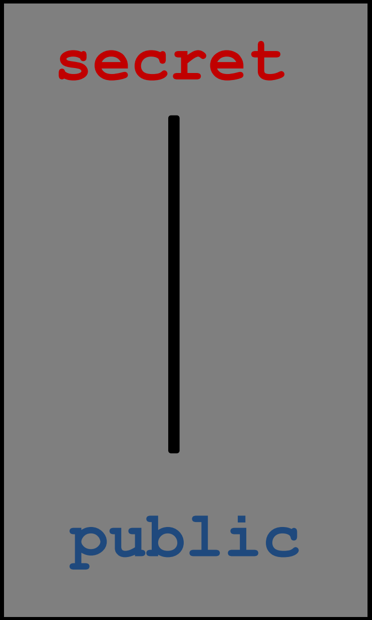
Security measures

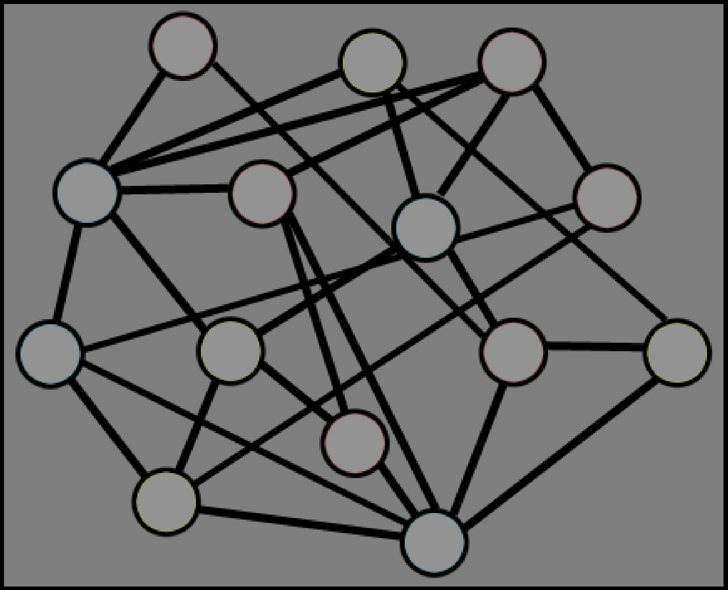


- Access control
  - State-of-the-art

Security lattice

- It specifies the allowed flows of information





Example of Rules

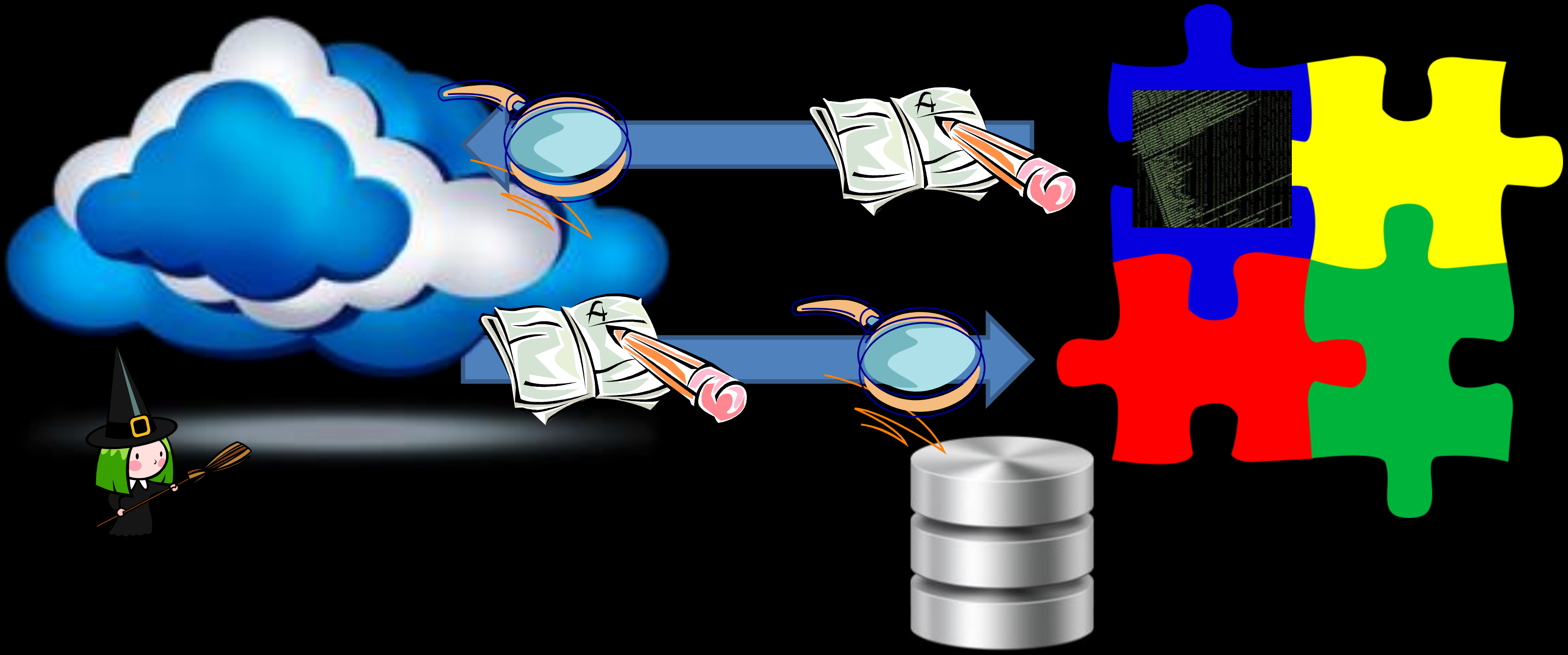
Arrows for Secure Information Flow

[Li, Zdancewic 10]

$$\Phi \vdash c : l \to l'$$

$$\frac{\Phi_1 \vdash c_1 : l_1 \to l_2 \qquad \Phi_2 \vdash c_2 : l_3 \to l_4}{\Phi_1 \cup \Phi_2 \cup \{l_2 \sqsubseteq l_3\} \vdash c_1 \ggg c_2 : l_1 \to l_4}$$

Information-flow Scenario



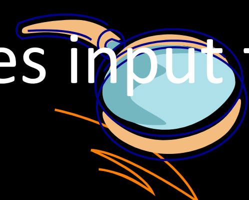
Towards a Monadic Calculus

Reader Monad

- The attacker might observe the systems data

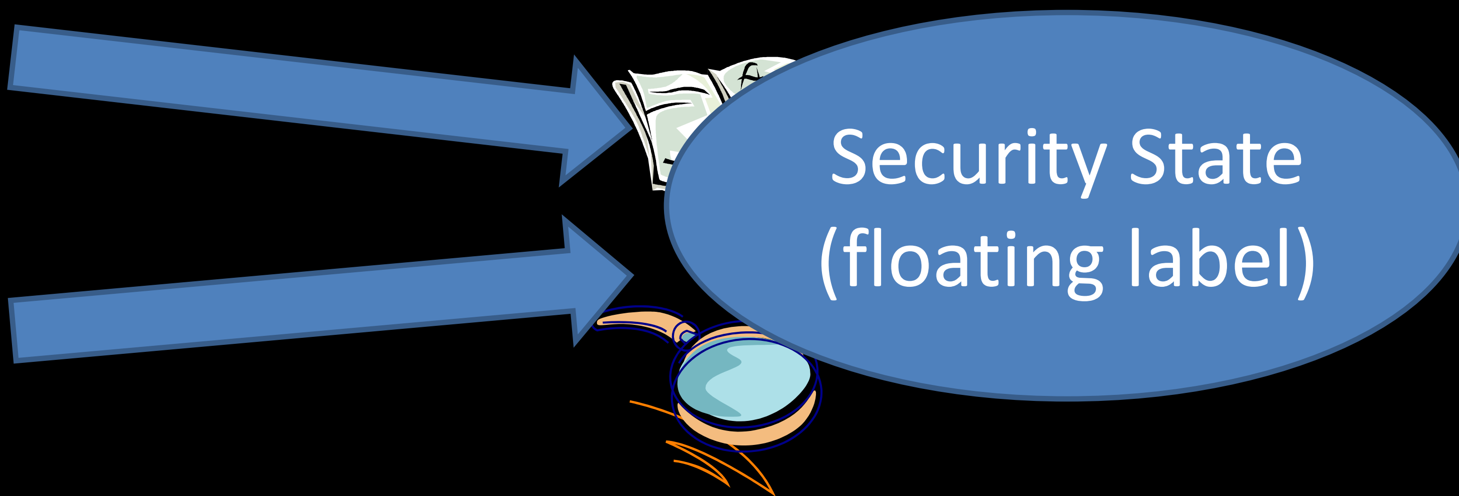
Writer Monad

- The attacker writes input to the system



Informatoin-flow control is <u>almost</u> just about controling reading and writing side-effects

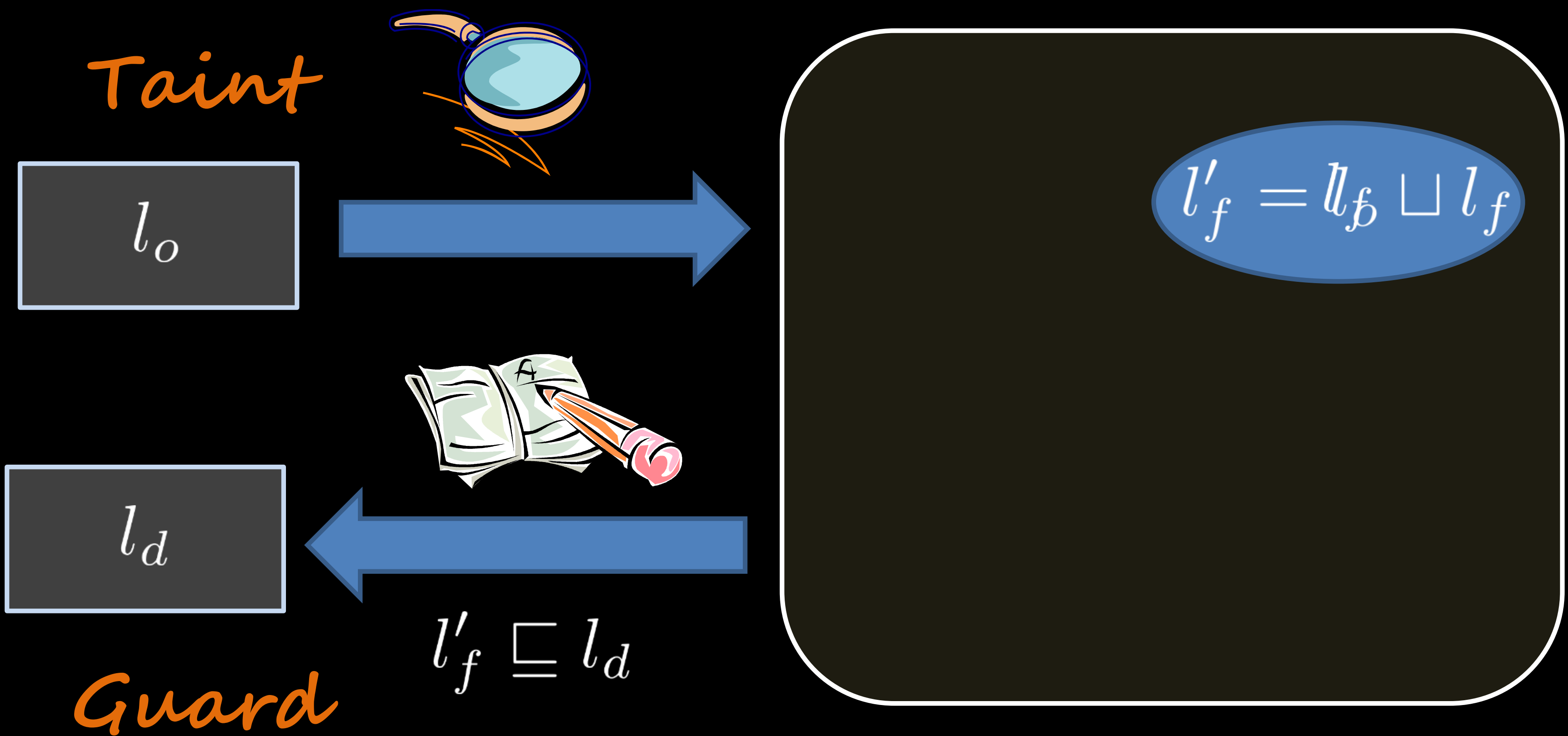
Towards a Monadic Calculus



Restricted interface for the State monad!

A Floating Label System

[Stefan et al. 11]



Example of Rules

Arrows for Secure Information Flow
[Li, Zdancewic 10]

$$\Phi \vdash c : l \rightarrow l'$$

$$\frac{\Phi_1 \vdash c_1 : l_1 \rightarrow l_2 \qquad \Phi_2 \vdash c_2 : l_3 \rightarrow l_4}{\Phi_1 \cup \Phi_2 \cup \{l_2 \sqsubseteq l_3\} \vdash c_1 \ggg c_2 : l_1 \rightarrow l_4}$$

Guard

Example of Rules

Information-Flow Security for a Core of JavaScript
[Hedin, Sabelfeld 12]

$$\frac{o = E[r] \quad o = \{s^{\underline{l_3}} \mapsto v^{\underline{l}}, \ldots\}}{GetOwnProperty(r^{\underline{l_1}}, s^{\underline{l_2}}) = v^{\underline{l \sqcup l_1 \sqcup l_2 \sqcup l_3}}}$$

Taint

Designing IFC Systems

	Read effect	Write effect
`newIORef`		✓
`readIORef`	✓	
`writeIORef`		✓
`modifyIORef`	✓	✓

	Read effect	Write effect
`newLIORef`		*Guard*
`readLIORef`	*Taint*	
`writeLIORef`		*Guard*
`modifyLIORef`	*Taint*	*Guard*

The IFC Monad

[Swierstra 08]

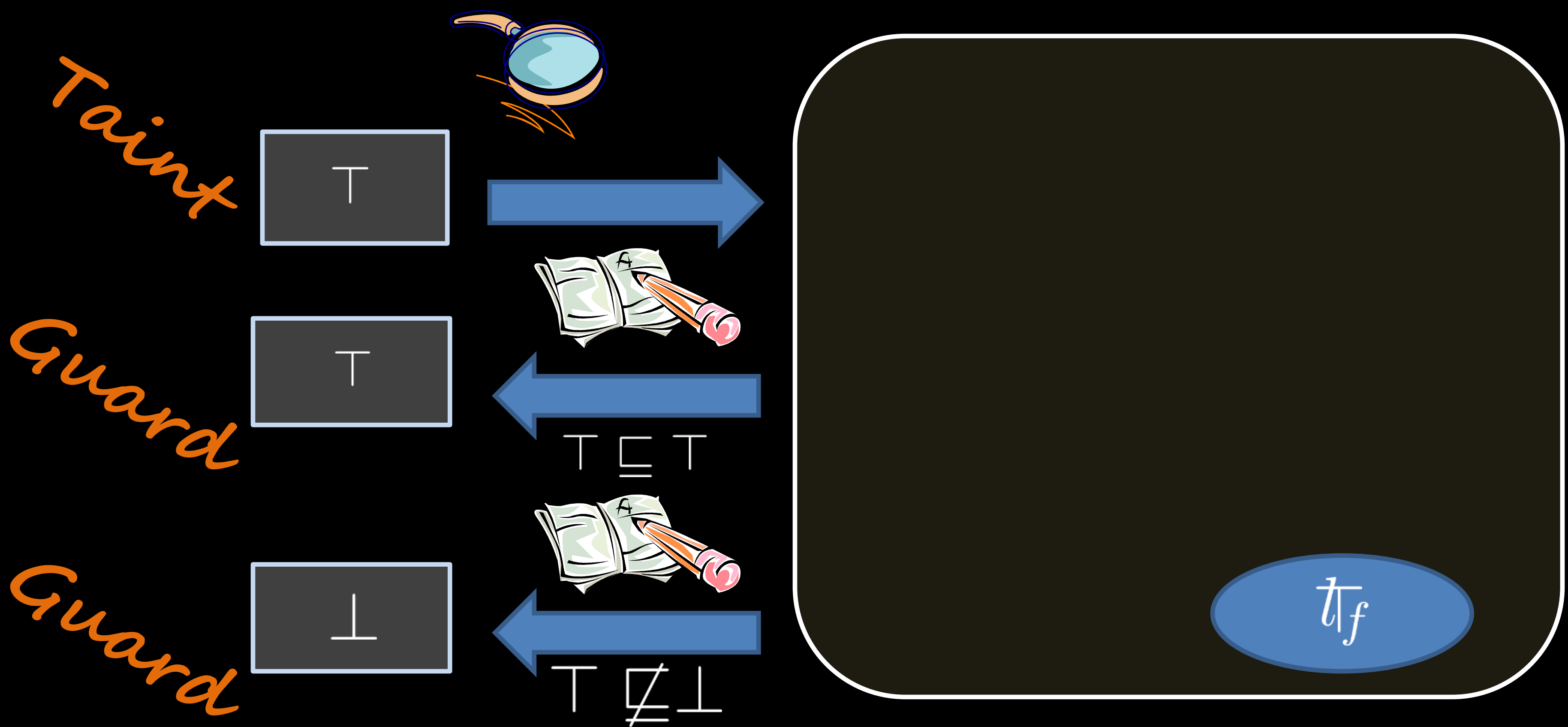
```
data ReadEffect l a where
     Taint :: l -> a -> ReadEffect l a

data WriteEffect l a where
     Guard :: l -> a -> WriteEffect l a

Free (ReadEffect l) a
Free (WriteEffect l) a
IFC l a = Free (WriteEffect l :+: ReadEffect l) a
```

Types reflects the behavior w.r.t taint and guard!

Label Creep

[Stefan et al. 11][Breeze 13]

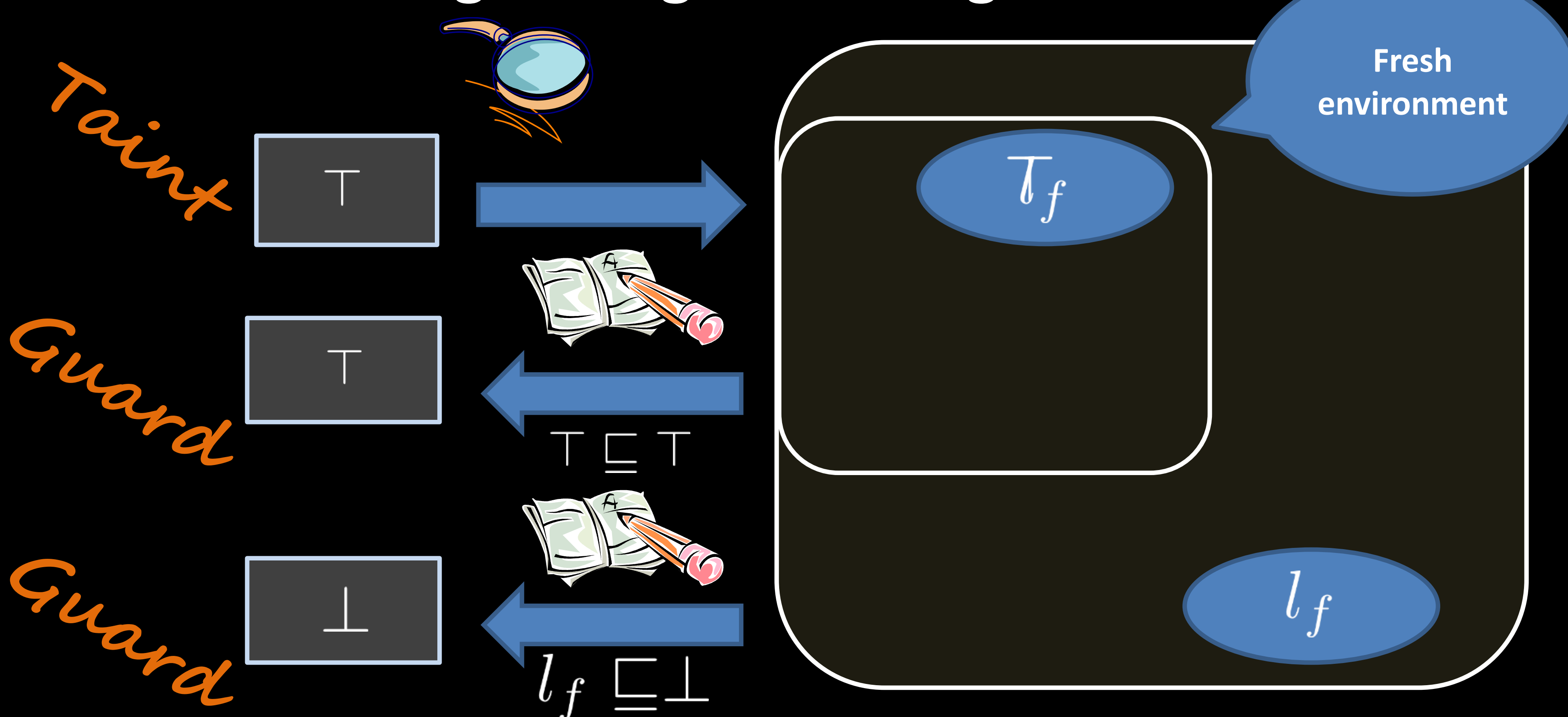
- The floating label gets too high too soon!



Label Creep

[Stefan et al. 11][Breeze 13]

- The floating label gets too high too soon!

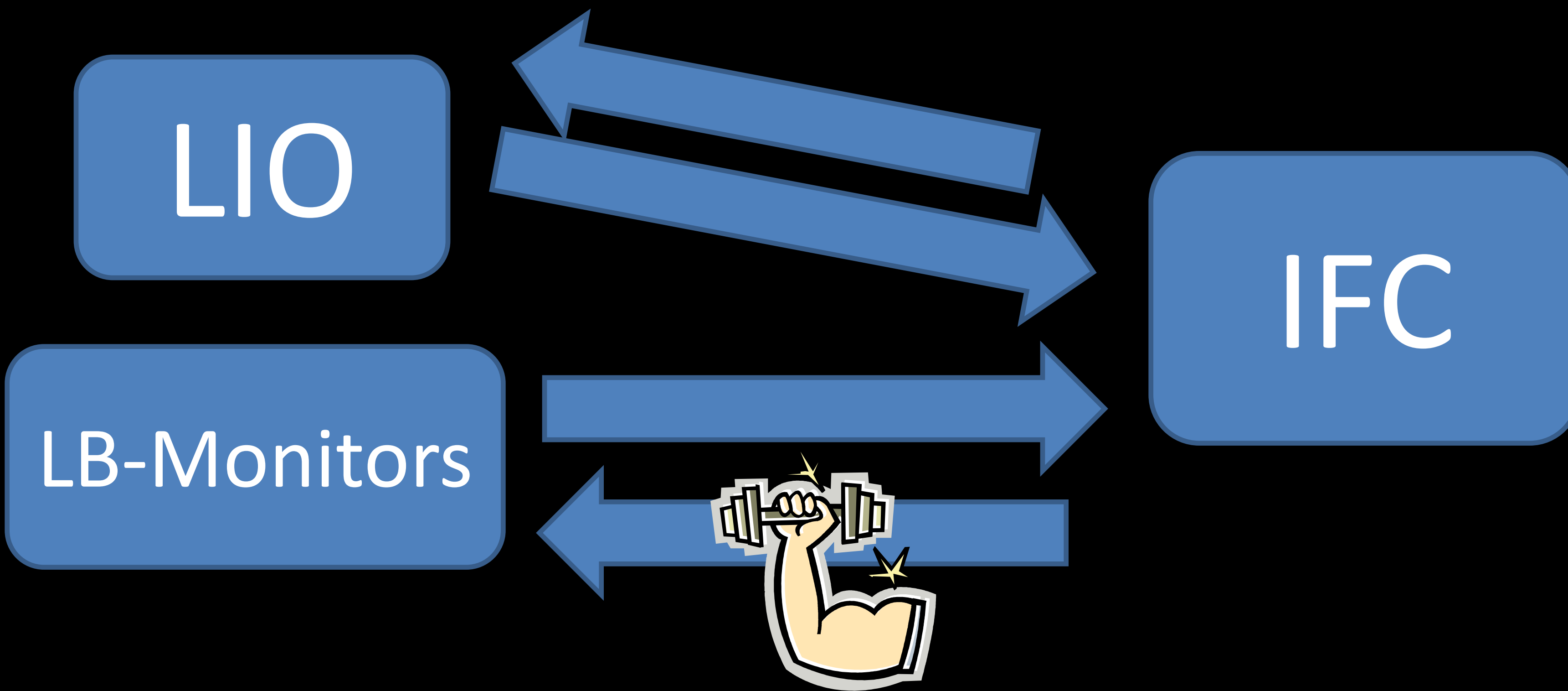


A specific Local (Reader Monad)

```
data Env l a where
    Ask :: (l -> a) -> Env l a
    Put :: l -> Env l a

 IFC l a = Free (WriteEffect l :+:
                 ReadEffect l  :+:
                 Env l) a

local :: forall l. Label l => IFC l () -> IFC l ()
local m = do (s :: l) <- IFC.ask
             m
             IFC.put s
             return ()
```

Extended

Is it General Enough?



Final Remarks

- IFC = controlling reading and writing side-effects
  + a notion of scope (local)

```
type IFC l a = Free (ReadEffect l  :+:
                     WriteEffect l :+:
                     Env l) a
```

- A non-security expert can have a good impression of the security checks (taint/guard)

- Floating label systems seems to be more convenient than traditional LB-monitors

Interested in Details?

https://github.com/alejandrorusso/ifc-wg2.8.git