All Your IFCException Are Belong To Us

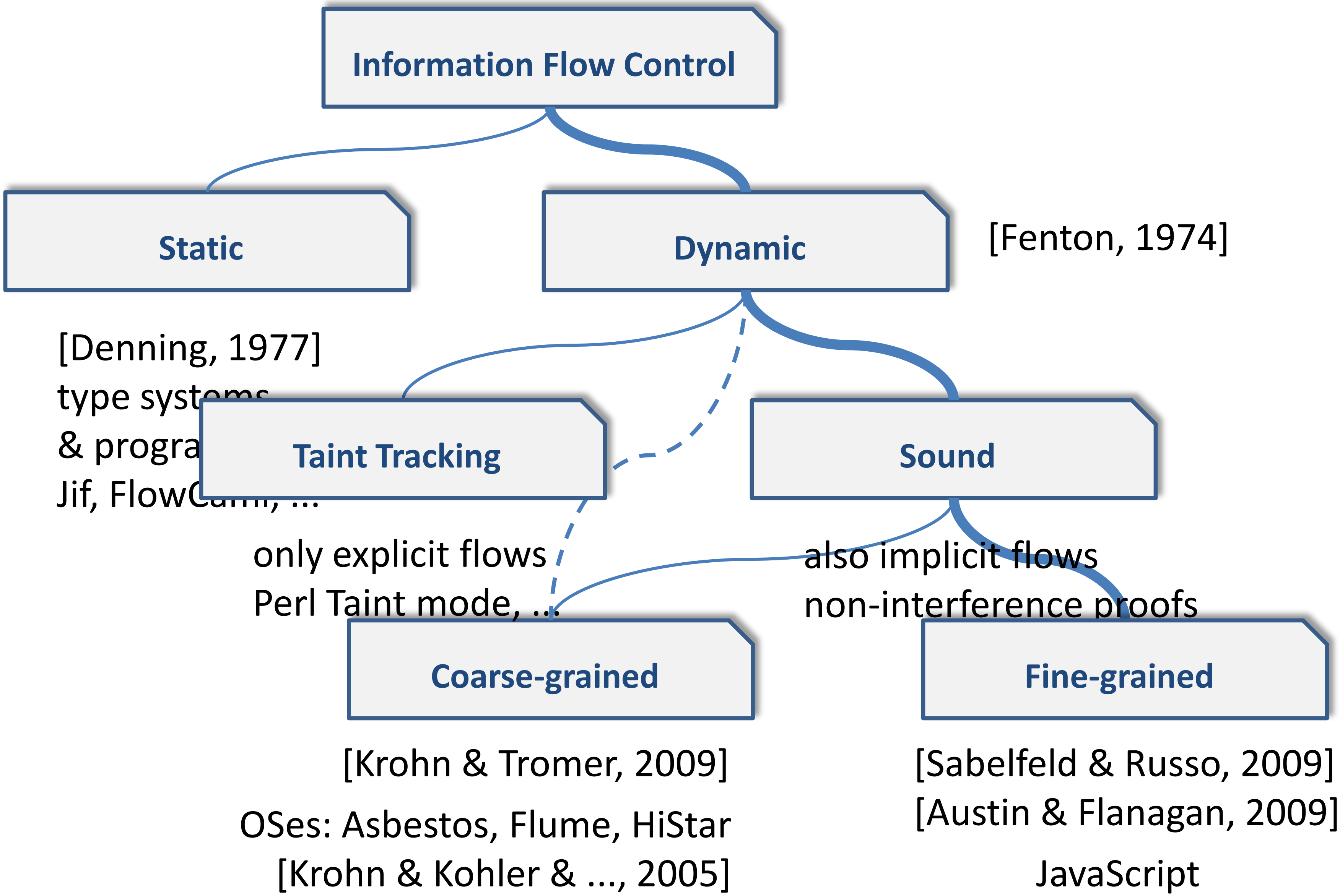
or

Exception Handling in Breeze

Cătălin Hrițcu

(joint work with Michael Greenberg, Ben Karel, Benjamin Pierce, Greg Morrisett, and more)

2012-10-15 at Harvard


Information Flow Control
Static
Dynamic
[Fenton, 1974]
[Denning, 1977]
type syst
& progra
Jif, FlowC
Taint Tracking
Sound
only explicit flows
Perl Taint mode, ...
also implicit flows
non-interference proofs
Coarse-grained
Fine-grained
[Krohn & Tromer, 2009]
OSes: Asbestos, Flume, HiStar
[Krohn & Kohler & ..., 2005]
[Sabelfeld & Russo, 2009]
[Austin & Flanagan, 2009]
JavaScript

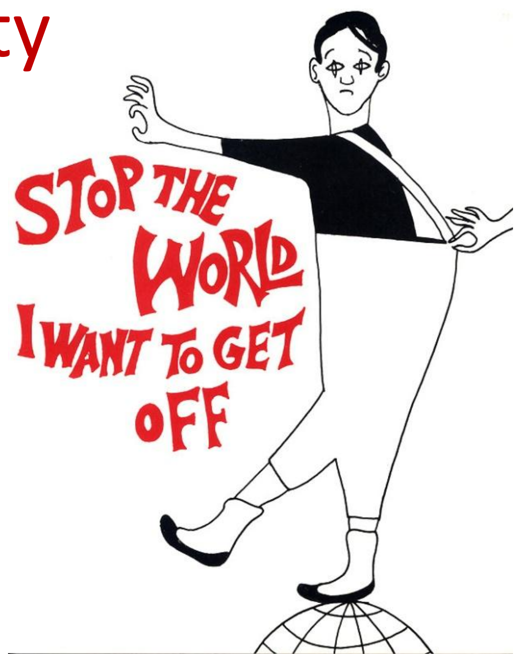
Breeze 2011

- **sound fine-grained dynamic IFC**
- **label-based discretionary access control**
  - clearance
- **functional core** (λ) + state(!) + concurrency (π)
  - from Pict/CML towards something more Erlang-ish
- **dynamically typed** (for now)
  - directly reflects capabilities of SAFE HW
  - dynamically-checked first-class **contracts**

Exception handling

- we wanted all Breeze errors to be **recoverable**
  - **including IFC violations!**
- however, existing work* assumes errors are **fatal**
  - makes some things easier ... at the expense of others

+secrecy +integrity –availability



*There are 2 recent (partial) exceptions:
[Stefan et al., 2012] and [Hedin & Sabelfeld, 2012]

Poison-pill attacks

```
let cin = chan low;
let cout = chan low;

fun process_max x y =                 3@low <= 2@high = false@high
  if x <= y then y else x                   pc=high

fun rec max_server_loop () =
  let (x,y) = recv cin in                   x=3@low  y=2@high
  let res = process_max x y in              res=3@high
  send cout res;  max_server gets killed because of IFC violation!?
  max_server_loop ()

let client = send cin (3, 5)@low; recv cout = 5
let attacker = send cin (3, 2@high)@low
```

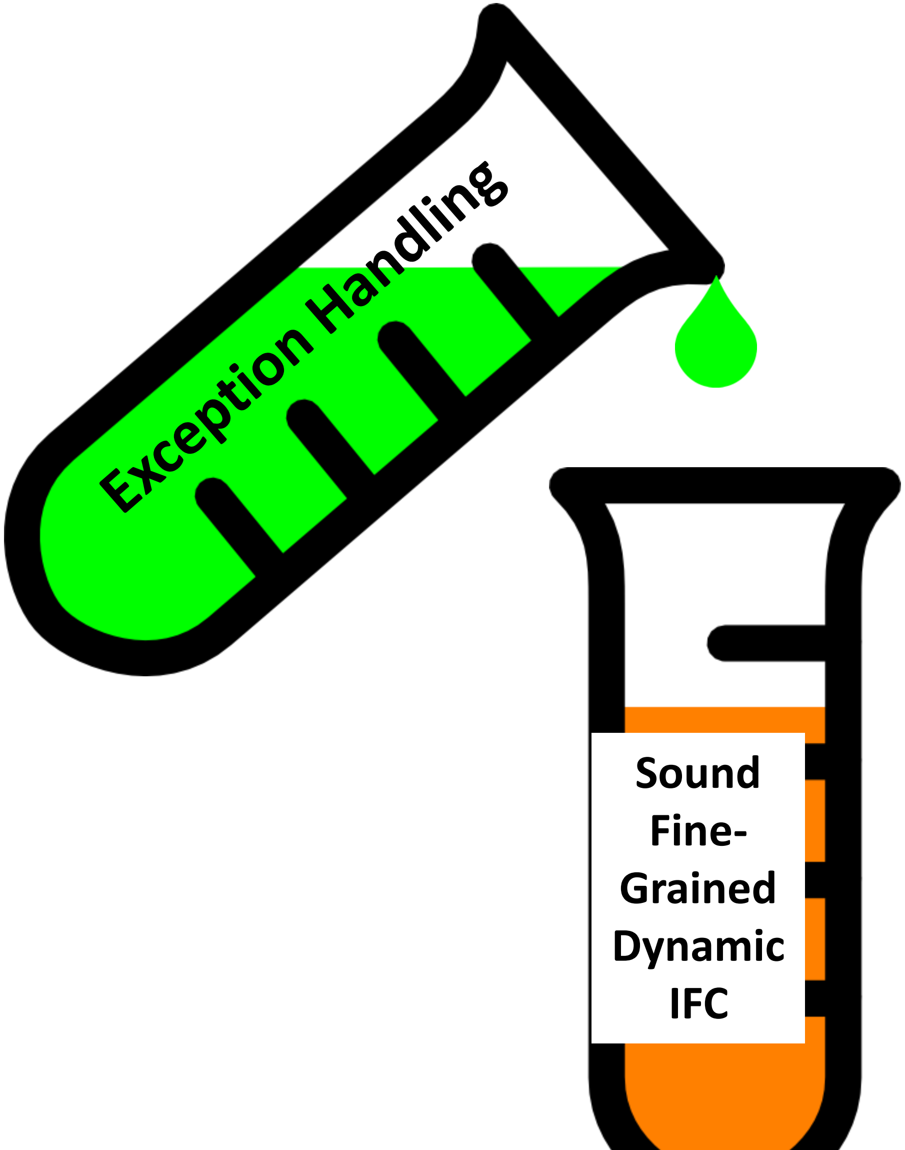
Wishful thinking

```
let cin = chan low;
let cout = chan low;

fun process_max (x,y) =
  if x <= y then y else x

fun rec max_server_loop' () =
  try
    send cout (process_max (recv cin))
  catch x => log x;
  max_server_loop' ()
```

But there is a problem

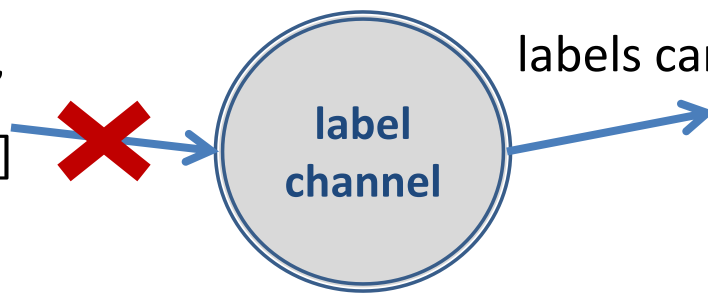


But there is a problem ... in fact two!

Labels are information channels

- well-known fact:
  - labels that change are themselves information channels
- more than one label channel:
  - labels on reference contents (strong updates)
  - vs. labels on values and components of values
- get soundness by preventing secrets from leaking either ***into*** or ***out of*** label channel

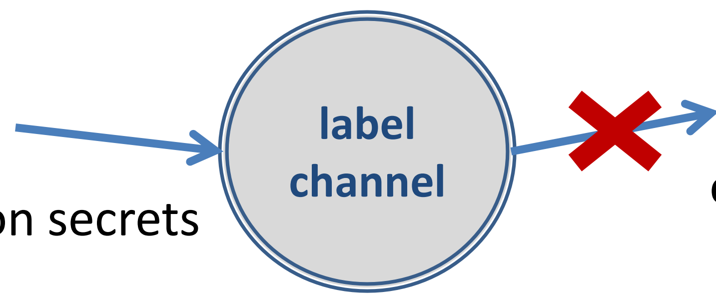
no strong updates or
“no-sensitive-upgrade”
[Austin & Flanagan, ’09]



labels can be observed

Labels are information channels

- well-known fact:
  - labels that change are themselves information channels
- more than one label channel:
  - labels on reference contents (strong updates)
  - vs. labels on values and components of values
- get soundness by preventing secrets from leaking either ***into*** or ***out of*** label channel

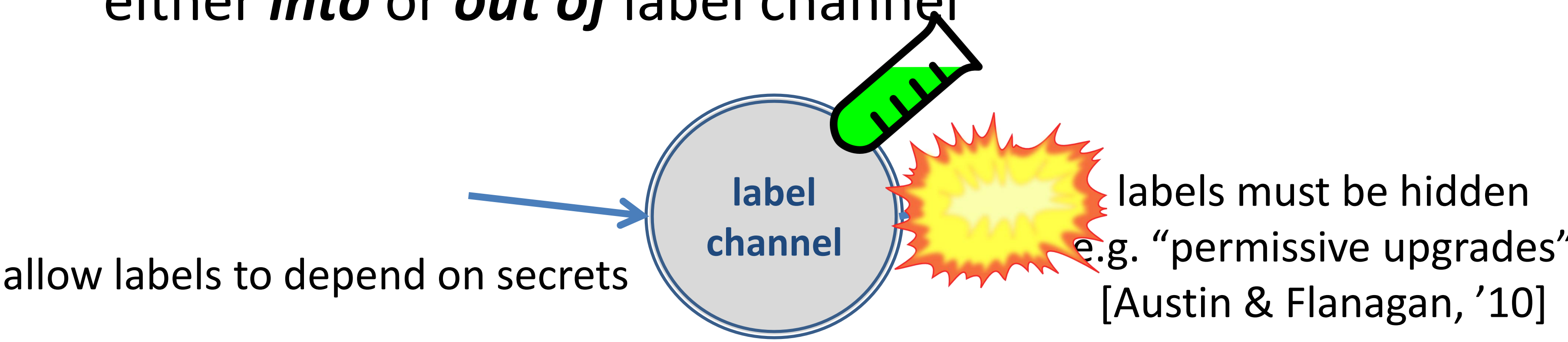


allow labels to depend on secrets

labels must be hidden
e.g. “permissive upgrades”
[Austin & Flanagan, ’10]

Labels are information channels

- well-known fact:
  - labels that change are themselves information channels
- more than one label channel:
  - labels on reference contents (strong updates)
  - vs. labels on values and components of values
- get soundness by preventing secrets from leaking either ***into*** or ***out of*** label channel



Problem #1: IFC exceptions make all label channels public

- we disallow strong updates
- still need to close label channel on values
- secret bit: h@high   `low <: high <: top`
- **let** href = ref high () **in**

```
.......
try
  href := (if h then ()@high
           else      ()@top );
  true
catch IFCException => false
```

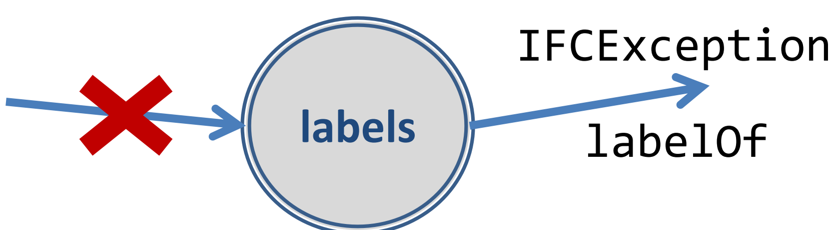
encode h into label

if branch – assignment works
else branch – IFCException

Automatic pc restoring just doesn't work!

Solution to problem #1: brackets

- no longer automatically restore pc
  - `pc=low` `if h then ()@high else ()@top` `pc=high`
- restore pc manually using **brackets**
  - choose label before branching on secrets
  - `pc=low` `top[if h then ()@high else ()@top]` `pc=low`
  - brackets are not declassification!
  - sound even when annotation is incorrect (more later)
- **labels are now public**
  - bracket annotations can be dynamically computed



Problem #2: exceptions destroy control flow join points

- ending brackets have to be control flow join points
  - ```
    try
      let _ = high[if h then throw Ex else ()] in
      false
    catch Ex => true
    ```
- failed brackets cannot raise exceptions
  - ```
    let lref = ref low false in
    try
      let _ = high[if h then ()@high else ()@top] in
      lref := true
    catch EBrk => ()
    ```
- brackets need to delay all exceptions!

Solution #2: Delayed exceptions

- **delayed exceptions unavoidable**
  – still have a choice how to propagate them
- we study **two alternatives** for error handling:
  1. **mix active and delayed exceptions** ($\lambda^{[\,]}_{throw}$)
  2. **only delayed exceptions** ($\lambda^{[\,]}_{NaV}$)
     - delayed exception = not-a-value (NaV)
     - NaVs are first-class replacement for values
     - NaVs propagated solely via data flow
     - NaVs are labeled and pervasive
     - more radical solution; implemented by Breeze

NaV-lax vs. NaV-strict behavior

- all non-parametric operations are NaV-strict
  - `NaV@low + 42@high => NaV@high`
- for parametric operations we can chose:

```
                          NaV-lax          or   NaV-strict
– (fun x => 42) NaV => 42                  or   => NaV
– Cons NaV Nil      => Cons NaV Nil        or   => NaV
– (r := NaV,r=7)    => ((),r=NaV)          or   => (NaV,r=7)
```

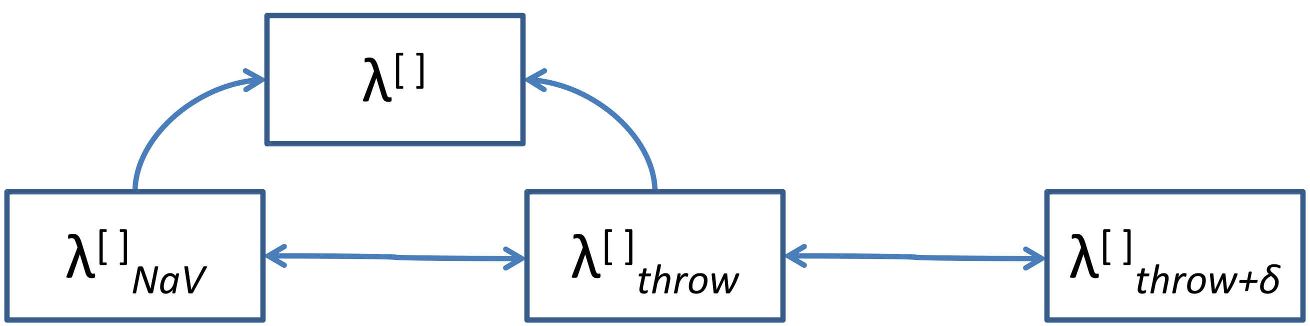
- NaV-strict behavior reveals errors earlier
  - but it also introduces additional IFC constraints
- in Breeze the programmer can choose
  - in formal development NaV-lax everywhere

What’s in a NaV?

- error message
  - \`EDivisionByZero (“can’t divide %1 by %2”, 42@high, 0@low)
  - high clearance code can obtain:
    “EDivisionByZero: can’t divide 42@high by 0@low”@high
  - all code can obtain:
    “EDivisionByZero: can’t divide <hidden>@high by 0@low”@low
- stack trace
  - pinpoints error **origin** (not the billion-dollar mistake)
- propagation trace
  - how did the error make it here?

Formal results

- proved **error-sensitive non-interference** in Coq for $\lambda^{[\,]}$, $\lambda^{[\,]}_{NaV}$, and $\lambda^{[\,]}_{throw}$ (termination-insensitive)
  - for $\lambda^{[\,]}_{NaV}$ even with all debugging aids
- **conjecture**: NaVs and catchable exceptions have equivalent expressive power
  - translations validated by quick-checking code extracted from Coq (working on Coq proofs)



Conclusion

- reliable error handling *possible* even for sound fine-grained dynamic IFC systems
- we study two mechanisms ($\lambda^{[\,]}_{NaV}$ and $\lambda^{[\,]}_{throw}$)
  - **all errors recoverable**, even IFC violations
  - necessary ingredients:
    public labels (via brackets) + **delayed exceptions**
  - quite radical design (not backwards compatible!)
- practical experience with NaVs
  - issues are surmountable
  - writing good error recovery code is still hard

THE END

INTEGRITY

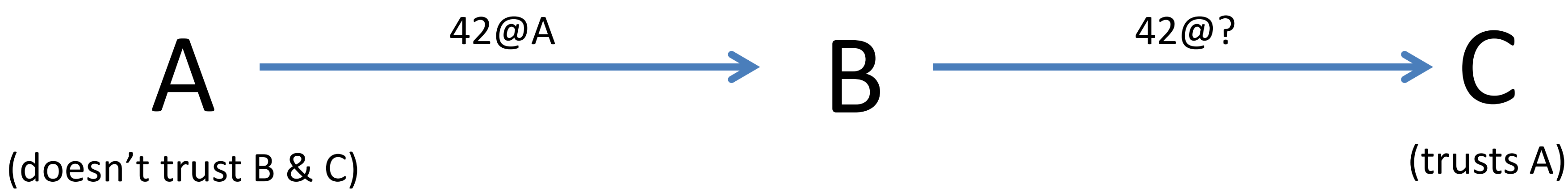
- **Integrity**
  - **Sound taint tracking**

    default label = bottom (trusted)
    (same as for secrecy)
    you know “who” is untrusted
    and trust everyone else
  - **Endorsement**

    default label
    = top (untrusted)
    - **IFC-endorsement**

      pc starts at integrity top
      stronger guarantees
      (by “being paranoid”)
      coarser-grained
    - **Signatures**

      not tracking implicit/explicit flows
      simple and natural model
      not IFC!
      finer-grained



Q: Should A's endorsement be preserved?

A1: No! (IFC-endorsement)

A2: Yes! (Signatures)

Signature labels

- Very much like digital signatures
  - P’s signing authority – P’s signing key
  - P’s name – P’s public verification key
- Lattice structure useful
  - conjunctive labels [[P],[Q]] – multi-signatures
  - disjunctive labels [[P,Q]] – group signatures
- Unforgeable
  - New atoms start out “unsigned” (integrity top)
  - Just passing around atoms preserves signatures

Data abstraction by signing

secrecy part:
only Bool can
access this data
(enforced by
clearance, not IFC)

integrity part:
only Bool can
create this
signature

Bool module
- private principal B
- true
- false
- not

1@([[B]]; [[B]])

Data abstraction by signing

More flexible than dynamic sealing:

- no extra boxing;
- secrecy separate from integrity;
- multiple signers and “decrypters”

Bool module

- private principal B
- true
- false
- not

1@([[B]]; [[B]])

0@([[B]]; [[B]])

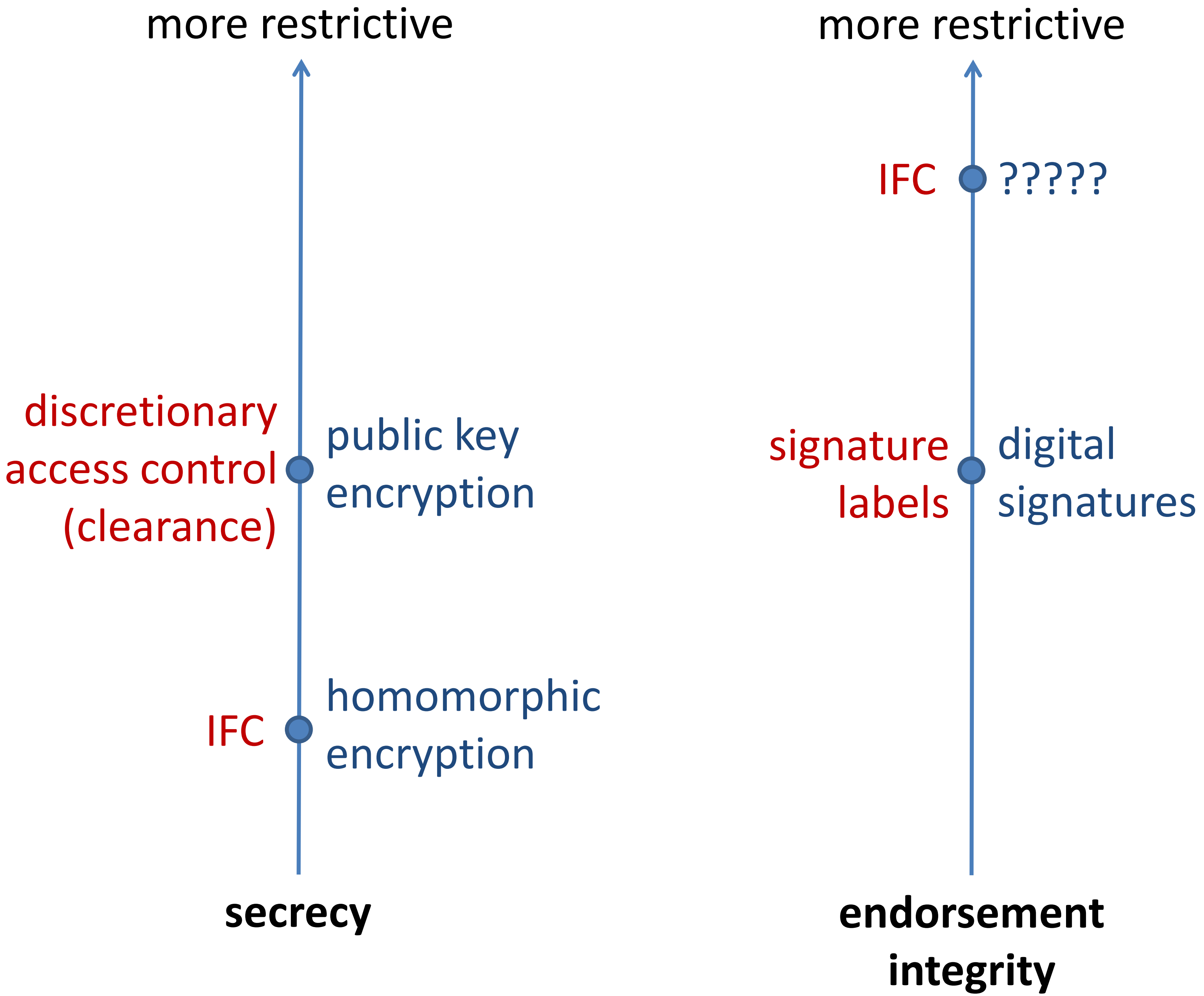
A

0@([[B],[C]]; [[A],[B]])

0@([[B]]; [[A],[B]])

1@([[B],[C]]; [[B]])

C (trusts B to access his data, but not to declassify it)


more restrictive
discretionary access control (clearance)
public key encryption
IFC
homomorphic encryption
secrecy
more restrictive
IFC
?????
signature labels
digital signatures
endorsement integrity

Signature labels are no silver bullet

$$A \xrightarrow{42^{o}@A \quad 45^{o}@A \quad 38^{o}@A} B \xrightarrow{45^{o}@A \quad 42^{o}@A \quad 42^{o}@A} C$$

Signature labels are no silver bullet

$$A \xrightarrow{(42^{o},\ 10\text{am},\ 2012\text{-}10\text{-}15)@A} B \xrightarrow{(42^{o},\ 10\text{am},\ 2012\text{-}10\text{-}15)@A} C$$

Signature labels are no silver bullet

$A \xrightarrow{(42^{o},\ 10am,\ 2012\text{-}10\text{-}15)@A} B \xrightarrow{(42^{o},\ 10am,\ 2012\text{-}10\text{-}15)@A} C$

Only sign “self-contained” (+immutable) messages

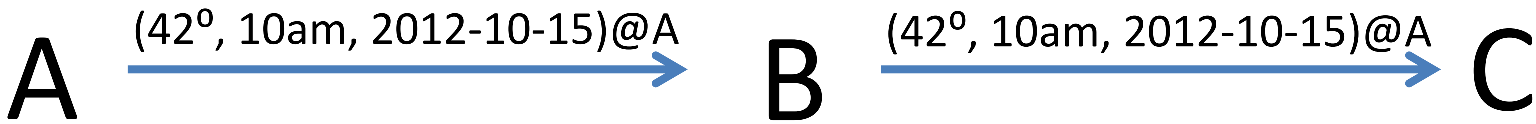
“Give B 42$”@A

“Give B 42$”@A

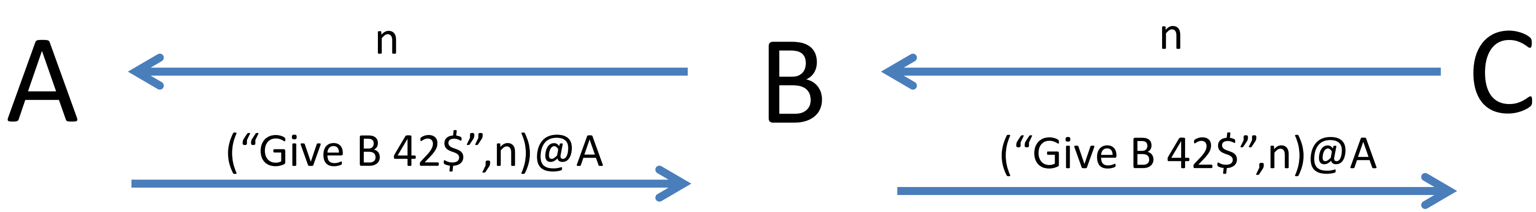
“Give B 42$”@A

$A \xrightarrow{\text{“Give B 42\$”@A}} B \longrightarrow C$

Signature labels are no silver bullet



Only sign “self-contained” (+immutable) messages



Signature labels are no silver bullet

A $\xrightarrow{\text{(42°, 10am, 2012-10-15)@A}}$ B $\xrightarrow{\text{(42°, 10am, 2012-10-15)@A}}$ C

Only sign “self-contained” (+immutable) messages

Signatures alone don’t guarantee freshness

- Linear/unique signatures? (could work in a closed system)

~~“Give B 42$”@A~~

~~“Give B 42$”@A~~

A $\xrightarrow{\text{“Give B 42\$”@A}}$ B $\xrightarrow{\text{“Give B 42\$”@A}}$ C

BACKUP SLIDES

Rules ($\lambda^{[\,]}{}_{NaV}$)

Boxes and atoms

$$
\begin{array}{lcl}
b & ::= & v \mid \delta\ excp \\
a & ::= & b@L
\end{array}
$$

$$\frac{\rho(x) = a}{\rho \vdash x, pc \Downarrow a, pc}$$

$$\frac{}{\rho \vdash (\lambda x.t), pc \Downarrow \langle \rho,\ \lambda x.\ t\rangle @\bot, pc}$$

$$\frac{\rho(x_1) = \langle \rho',\ \lambda x.\ t\rangle @L \quad \rho(x_2) = a \qquad (\rho', x \mapsto a) \vdash t, (pc \vee L) \Downarrow a', pc'}{\rho \vdash (x_1\ x_2), pc \Downarrow a', pc'}$$

$$\frac{\rho(x) = v@L}{\rho \vdash \mathsf{labelOf}\ x, pc \Downarrow L@\bot, pc}$$

Rules ($\lambda^{[\,]}{}_{NaV}$)

$$\frac{\rho(x) = b@\cancel{L'} \quad \mathit{tagOf}\ b \neq \mathsf{TLab}}{\rho \vdash \underline{x}[t], pc \Downarrow (\delta\ (\mathit{prEx}\ b))@\bot, (pc \vee \cancel{L'})}$$

prEx (δ excp) = δ excp
prEx _ = EType

$$\frac{\begin{array}{l}\rho(x) = L@\cancel{L'} \quad \rho \vdash t, (pc \vee \cancel{L'}) \Downarrow b@L'', pc' \\ L'' \vee pc' \sqsubseteq L \vee (pc \vee \cancel{L'})\end{array}}{\rho \vdash \underline{x}[t], pc \Downarrow b@L, (pc \vee \cancel{L'})}$$

$$\frac{\begin{array}{l}\rho(x) = L@\cancel{L'} \quad \rho \vdash t, (pc \vee \cancel{L'}) \Downarrow b@L'', pc' \\ L'' \vee pc' \not\sqsubseteq L \vee (pc \vee \cancel{L'})\end{array}}{\rho \vdash \underline{x}[t], pc \Downarrow (\delta\ \mathsf{EBrk})@L, (pc \vee \cancel{L'})}$$