A Decentralized Model for Information Flow Control

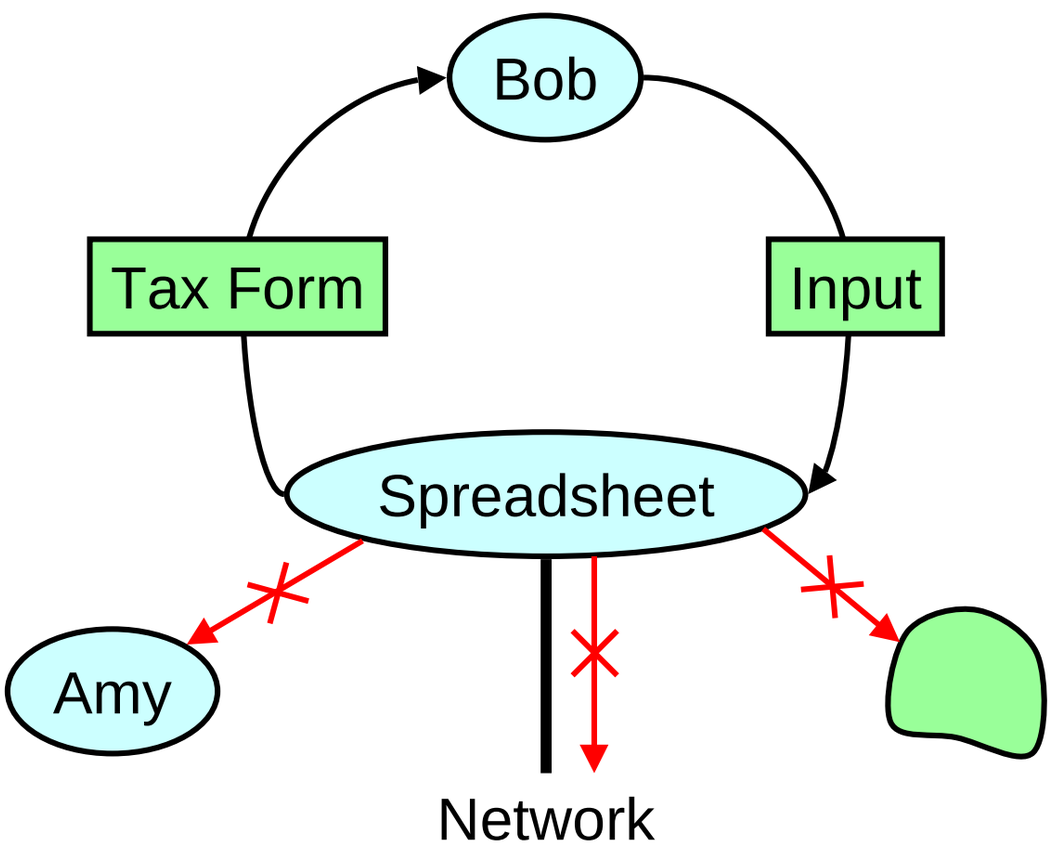
Andrew C. Myers
Barbara Liskov

MIT Laboratory for Computer Science
Cambridge, Massachusetts
http://www.pmg.lcs.mit.edu/

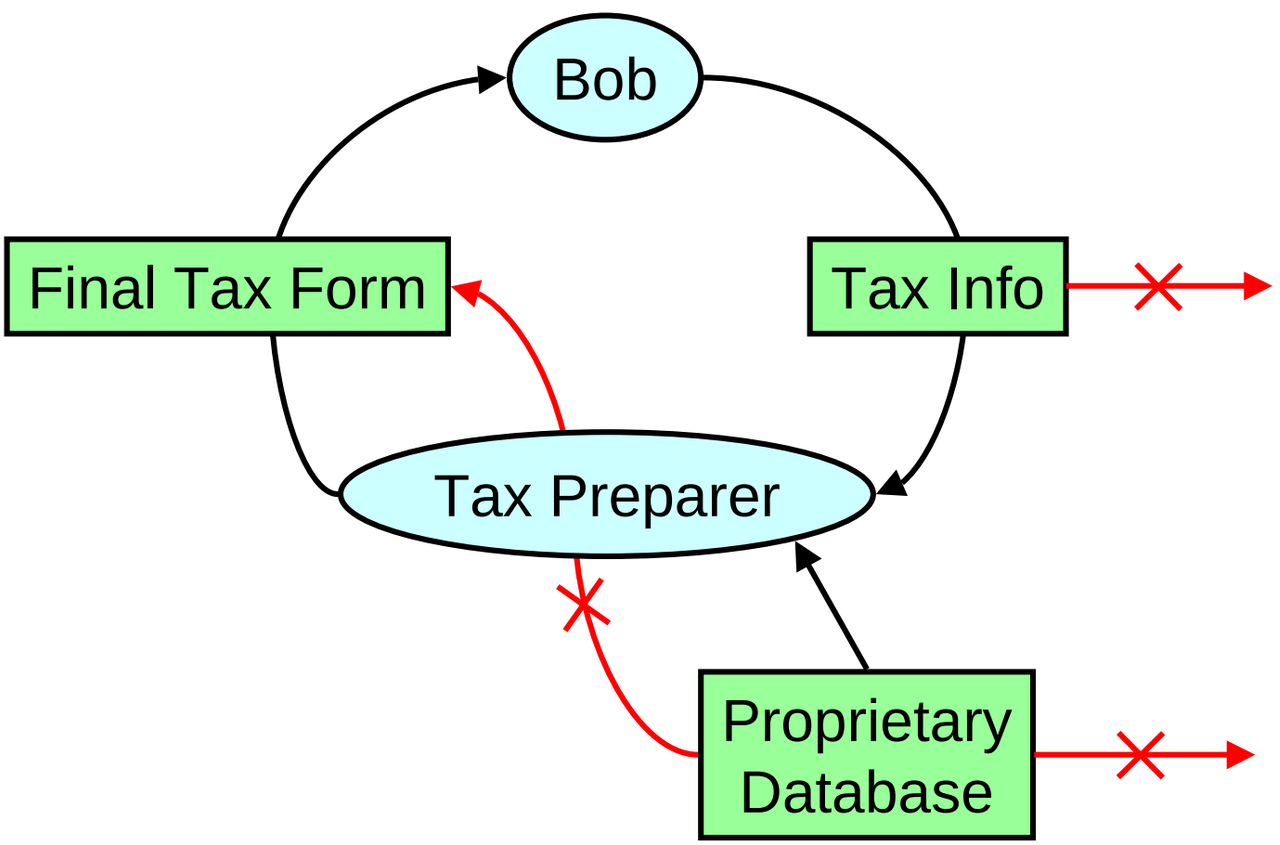
Problem: Information Flow

- Untrusted programs
- Fine-grained control
  - prevent information leaks
  - allow legitimate sharing
- Multiple principals : mutual distrust
- Mobile code : applets & servlets

Spreadsheet Example

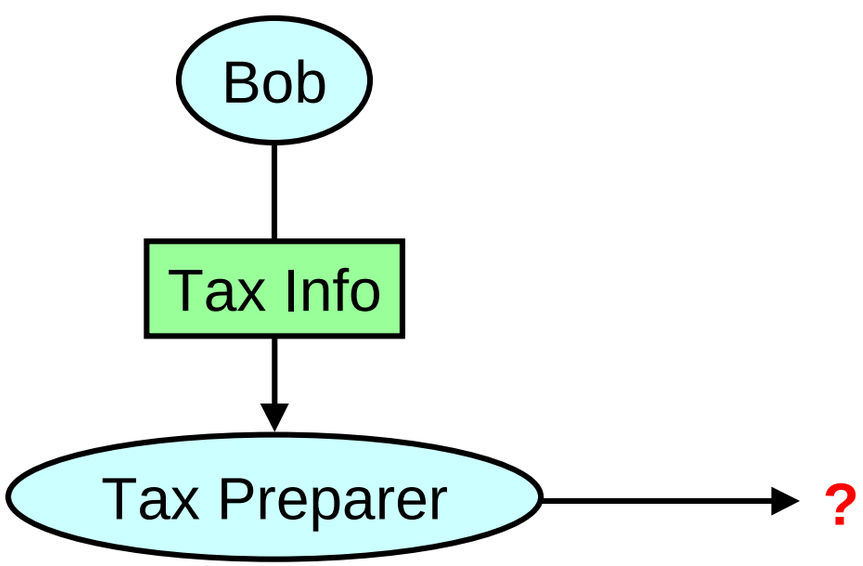


Tax Preparer Example



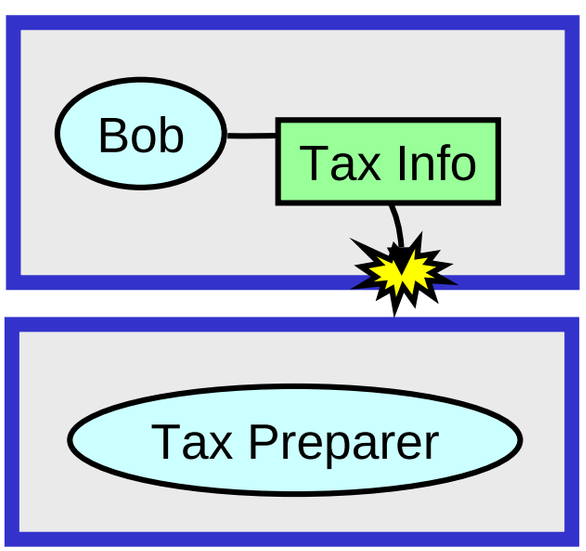
Current Approaches: Access Control

- Controls initial release of information
- Does not control propagation



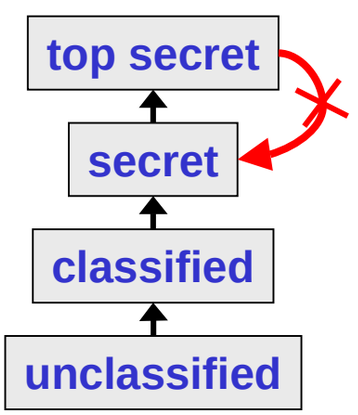
Current Approaches: Java Sandboxes

- Each program → different sandbox
- Prevents all sharing
- Different data have different requirements



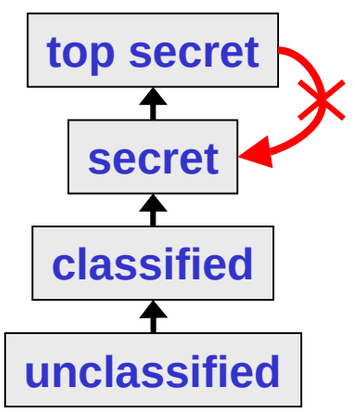
Related Work: Information Flow Control

Bell & LaPadula ’75
Denning ’76
Denning & Denning ’77
Andrews & Reitman ’80
Volpano, Smith, Irvine ’96

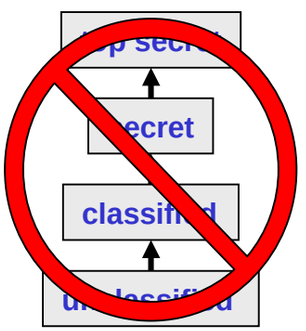


Current Approaches: Information Flow Control

- Each data item is labeled
- Labels can only become more restrictive
- Declassification/downgrading is outside the model
- Only protects one entity



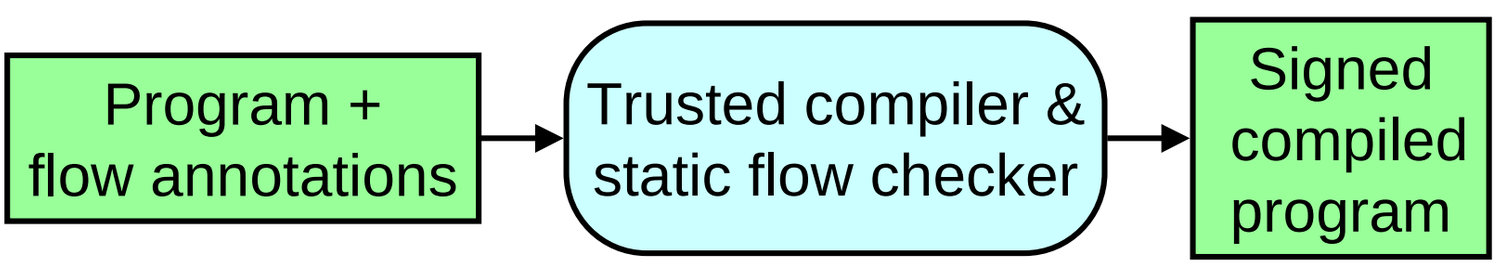
Our Approach



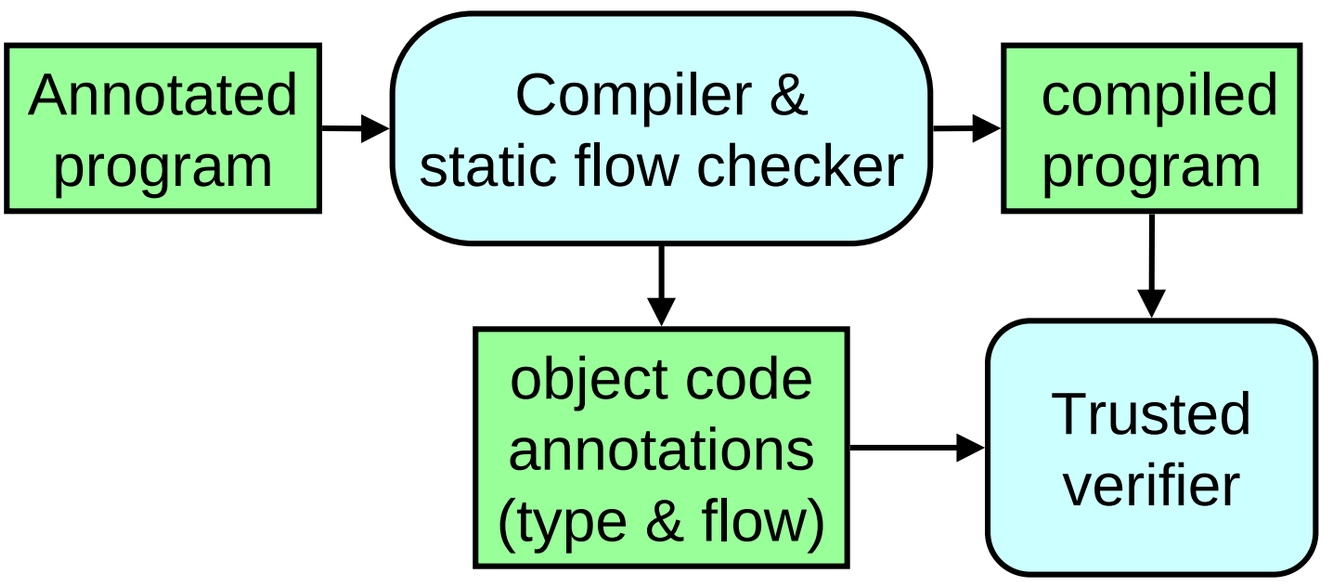
- New label model
  - User-defined flow policies
  - Safe declassification within model
- Improved static checking
  - Fine-grained control of data flow
  - More expressive
  - More usable: automatic inference of annotations

Architecture

- Precise flow control
- Information flow tracked at fine grain
  - e.g., individual assignments
  - requires language-level integration



Mobile Architecture



- Check object code, like JVM or PCC

Labels

- Every data item has an attached label
- A label has a set of *owners* (principals)
- For each owner, a set of *readers* (principals)

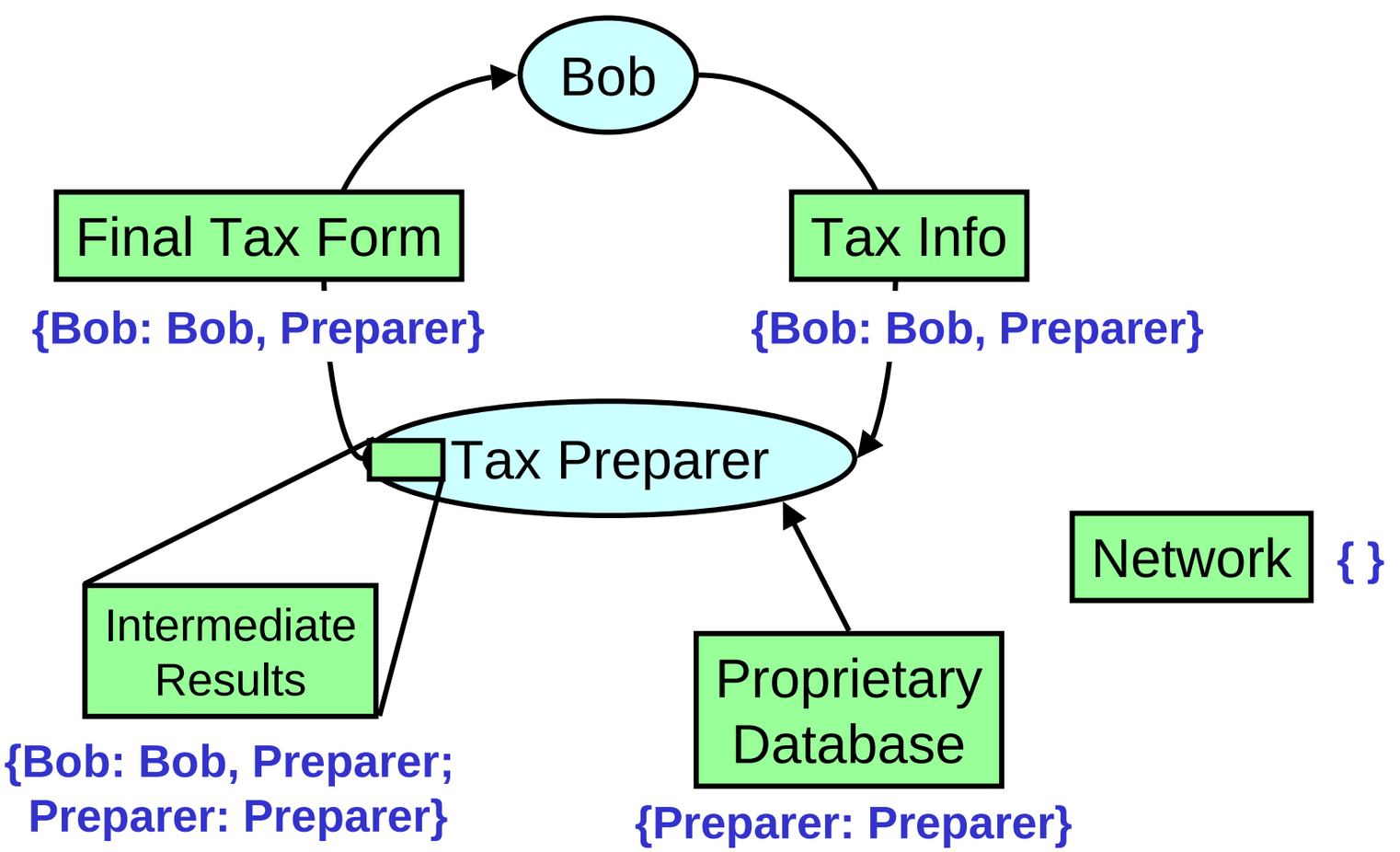
{owner **:** reader**, ...; ...}**

{ Bob: Bob, Preparer; Preparer: Preparer}

Each owner's policy is satisfied

– *effective reader set* is intersection of readers
– principals may change the part they own

Tax Preparer Example



Restriction

- Assignment *relabels* a value

  x := y

- Okay if $\underline{x}$ is at least as restrictive as $\underline{y}$

  (label of z is $\underline{z}$)

- $\underline{y} \sqsubseteq \underline{x}$ means:

  $owners(\underline{y}) \subseteq owners(\underline{x})$

  $readers(\underline{y}, O) \supseteq readers(\underline{x}, O)$

- The $\sqsubseteq$ relation defines valid relabelings

  {Bob: Bob, Preparer} $\sqsubseteq$ {Bob: Bob; Preparer: Preparer}

Derived Labels

- Combining values → *Joining* labels

$$y + z \quad \rightarrow \quad \underline{y} \sqcup \underline{z}$$

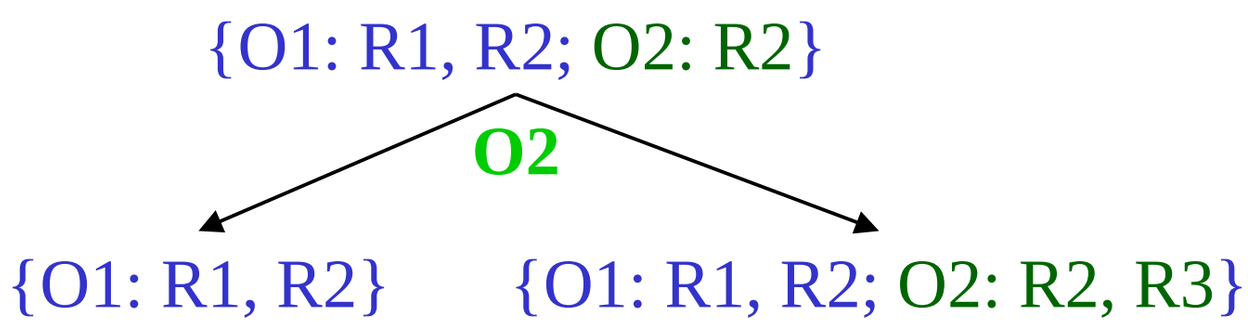
- The join of two labels ($\sqcup$) is the least restrictive label that is at least as restrictive as both:

$$owners(L_1 \sqcup L_2) = owners(L_1) \cup owners(L_2)$$

$$readers(L_1 \sqcup L_2, O) = readers(L_1, O) \cap readers(L_2, O)$$

Declassification

- Label tracks policy owners
- A principal can rewrite its part of the label



- Other owners' policies still respected
- Requires test of authority

Static Annotations

- Annotate a safe language with labels

$$\text{int}\{L_1\}\ x;$$

$$\text{int}\{L_2\}\ y;$$

- Treat label checking like type checking

$$x = y; \qquad \text{int} = \text{int} \ \&\ L_2 \sqsubseteq L_1$$

- Check program at procedure level (annotate arguments, return values)

Checking Flows

```
x = 0;
if (b)
    x = y;
```

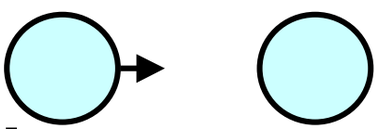
- Final value of x may reveal values of y, b
- Label of x must be at least as restrictive as labels of y, b:

$$\underline{y} \sqsubseteq \underline{x} \;\&\; \underline{b} \sqsubseteq \underline{x}$$

Run time vs. Static Checking

```
x = 0
if (b)
    x = y;
```

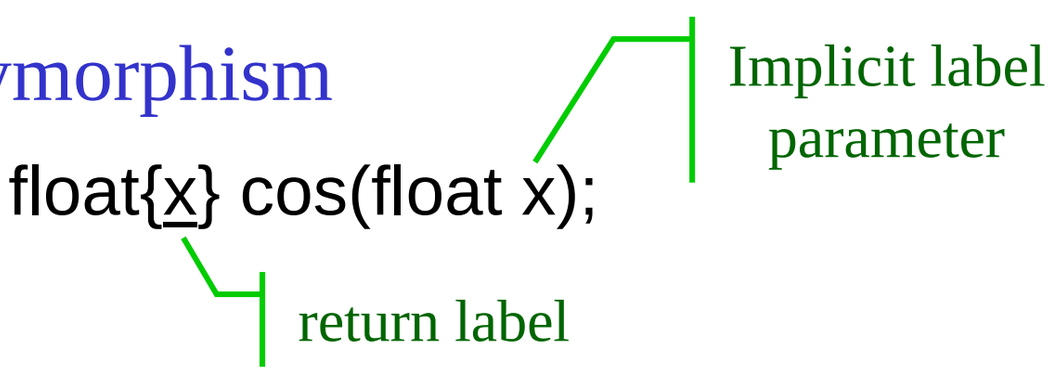
$\underline{y} \sqsubseteq \underline{x}$ & $\underline{b} \sqsubseteq \underline{x}$



- Run-time check:
  - implicit flow leaks b when b = false
  - b is also leaked if check fails!
- Static check:
  - failure gives no information about x, y, b
- Static checking is not just an optimization!

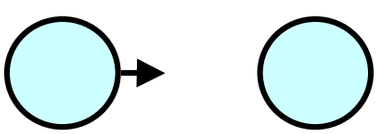
Label Inference

- Many labels need not be written
- Local variable labels can be inferred
  - like type inference
  - constraint system ($a \sqsubseteq b \sqcup c$)
- Label polymorphism



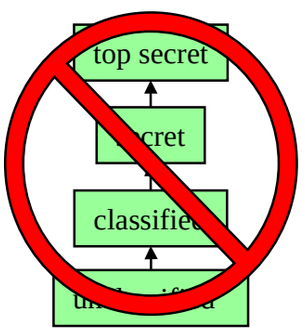
Other Extensions

- Explicit declassify operator
- Principal hierarchy (*acts-for*)
- Authority annotations
  - run-time tests
  - delegation
- Run-time label checks
  - statically checked for leaks

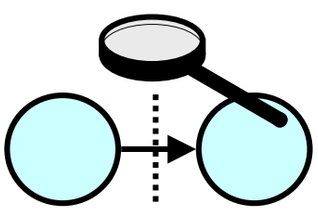


- Input/Output channels

Summary



- New Label Model
  - Keeps track of owners/sources
  - Allows safe declassification
  - More flexible than existing models
- Static Analysis of Annotated Programs
  - More expressive
    - Label polymorphism
    - Safe run-time checks
    - Explicit declassification
  - More usable: Label inference



Extensions

- In progress:
  - Implement compiler for annotated Java
  - Extend label model
- Future work:
  - untrusted components
  - integrity
  - static analysis for timing channels

A Decentralized Model for Information Flow Control

Andrew C. Myers

Barbara Liskov

MIT Laboratory for Computer Science
Cambridge, Massachusetts
http://www.pmg.lcs.mit.edu/
{andru,liskov}@lcs.mit.edu