Lecture 19: Information Flow

CS 181S Spring 2024

Where we were…

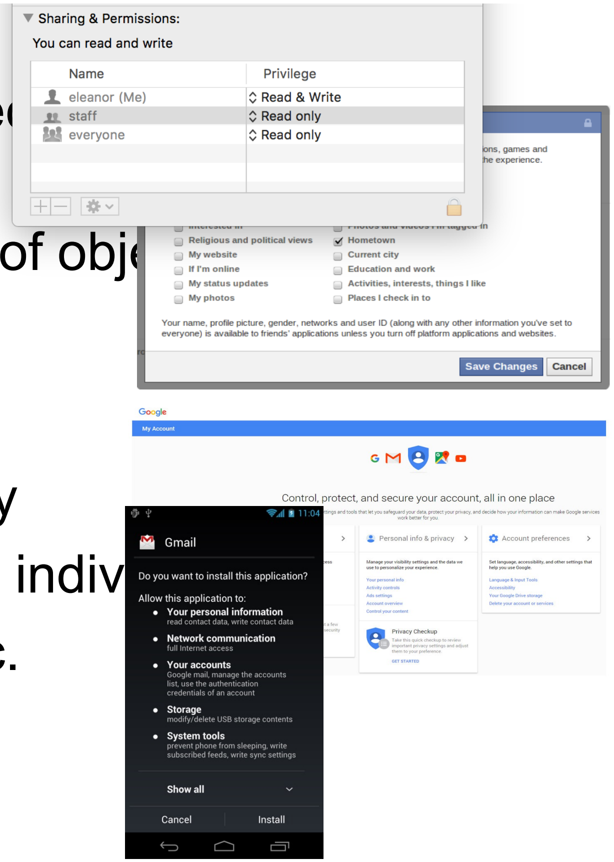
- **Authentication:** mechanisms that bind principals to actions

- **Authorization:** mechanisms that govern whether actions are permitted

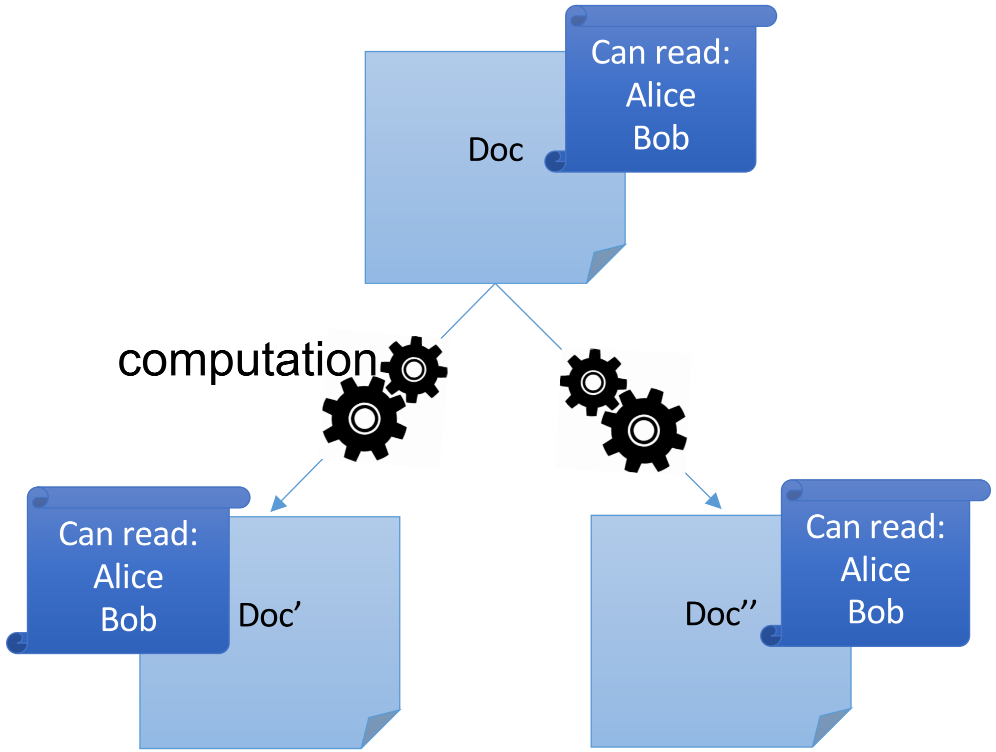
- **Audit:** mechanisms that record and review actions

Who defines Policies?

- Discretionary access control (DAC)
  - **Philosophy:** users have the *discretion* to spe
    themselves
  - Commonly, information belongs to the **owner** of obj
  - Access control lists, privilege lists, capabilities
- Mandatory access control (MAC)
  - **Philosophy:** central authority *mandates* policy
  - Information belongs to the authority, not to the indiv
  - MLS and BLP, Chinese wall, Clark-Wilson, etc.

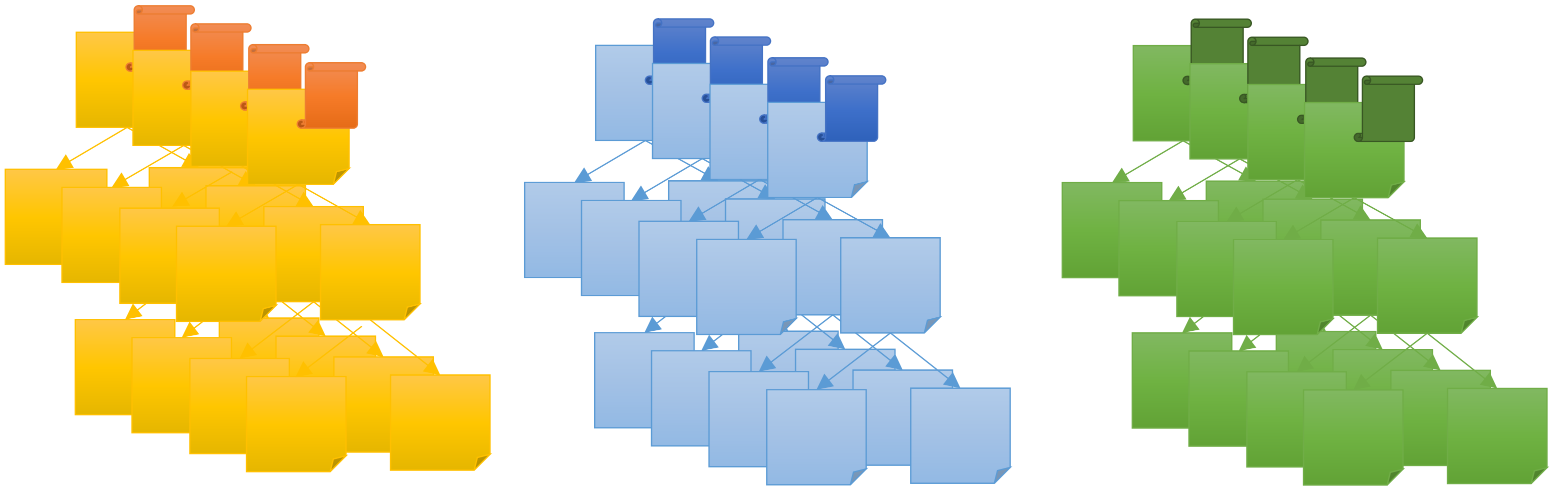


Access control for computed data

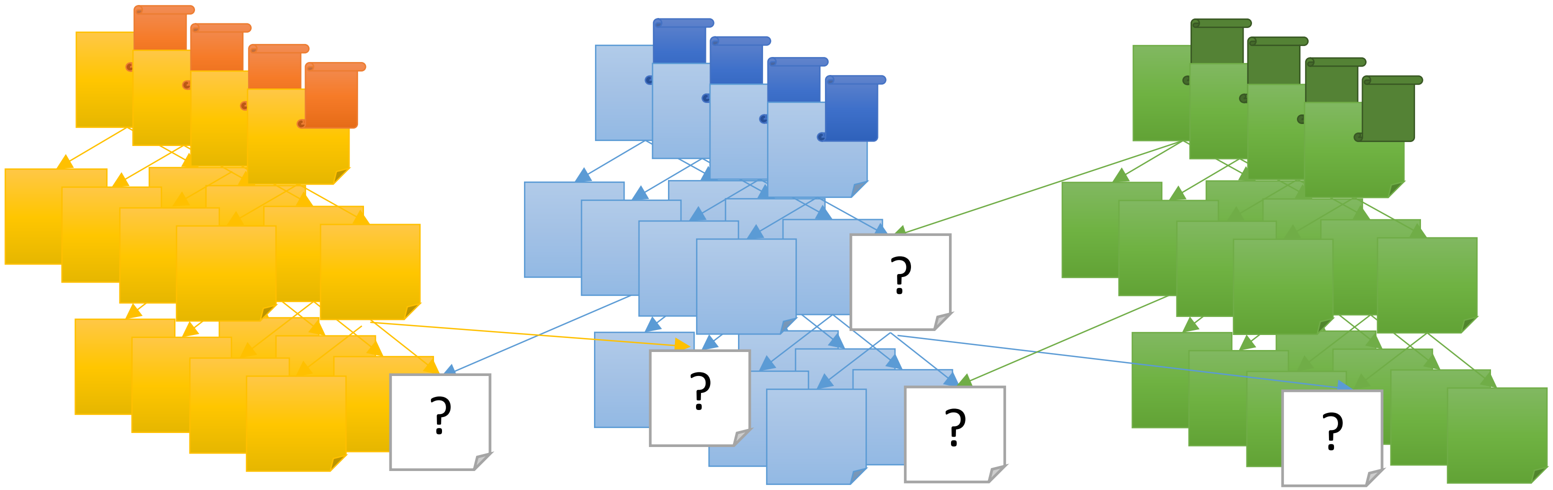


Scaling to many pieces of data…

Scaling to many users…

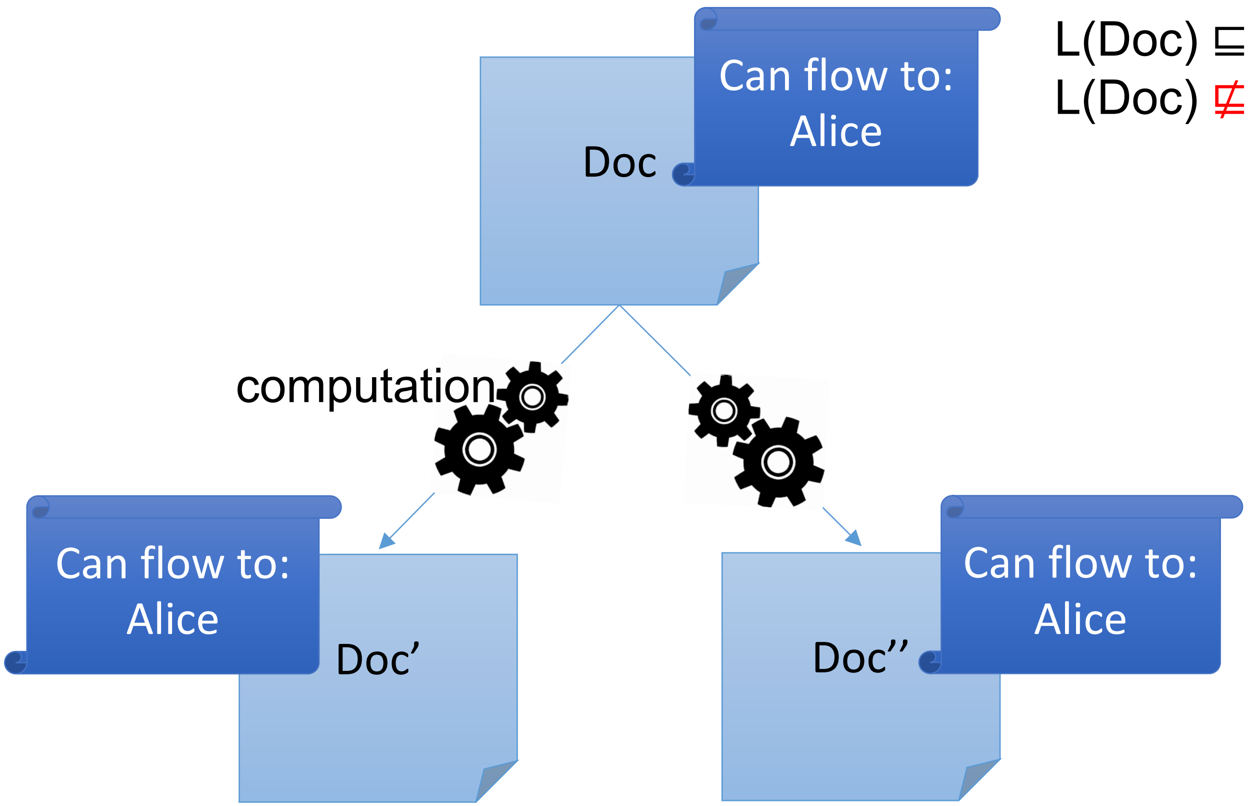


Scaling to many interactions…



Need to assign restrictions in an automatic way.

Information flow policies

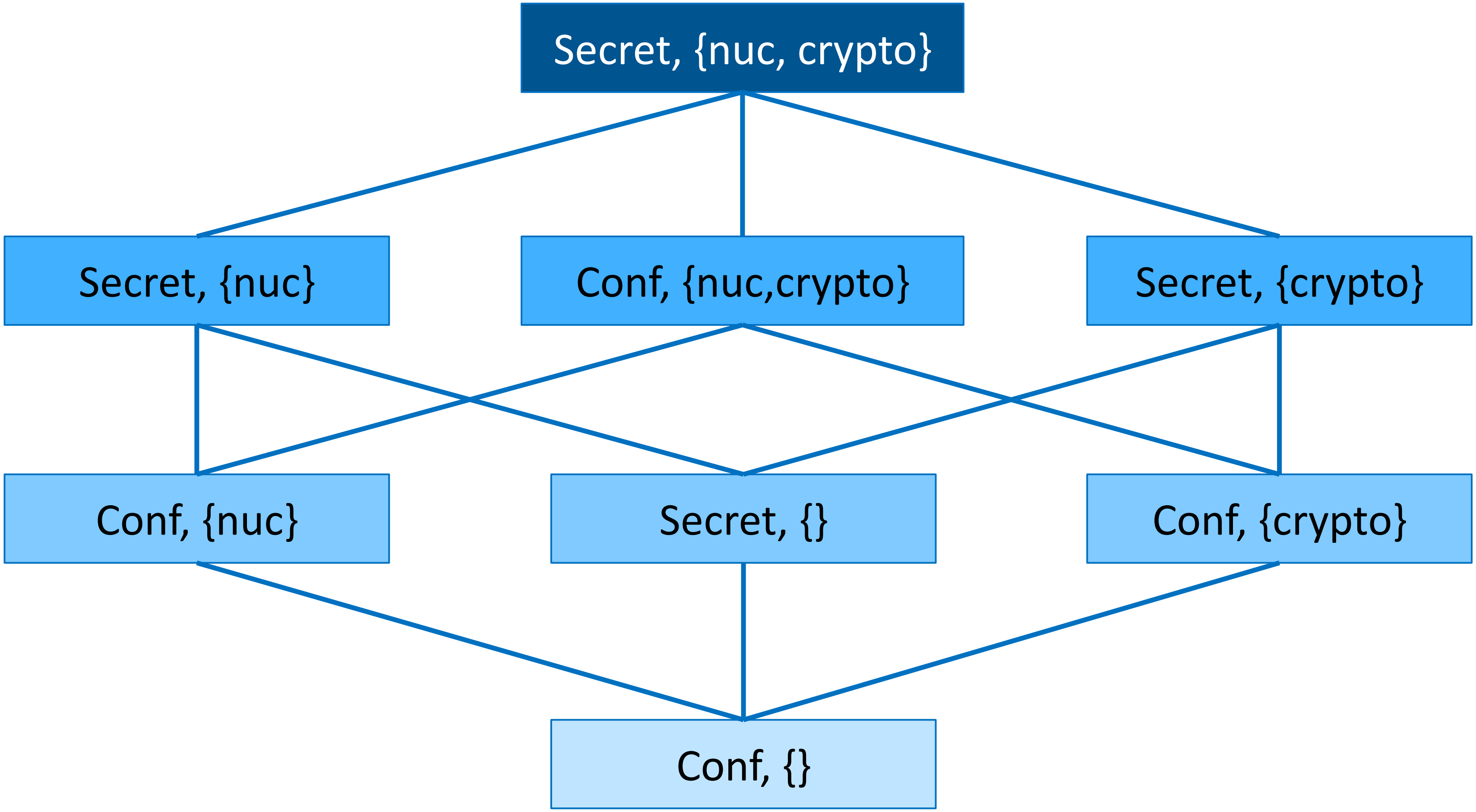


$L(Doc) \sqsubseteq L(Alice)$

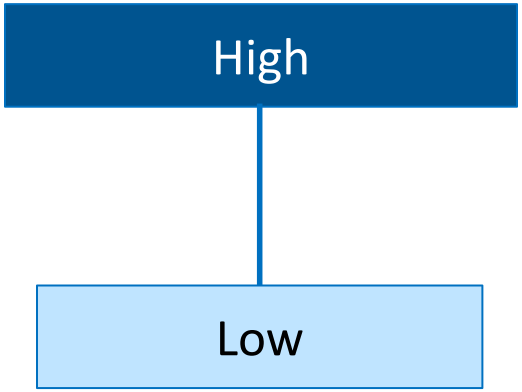
$L(Doc) \not\sqsubseteq L(Bob)$

Automatic deduction of policies!

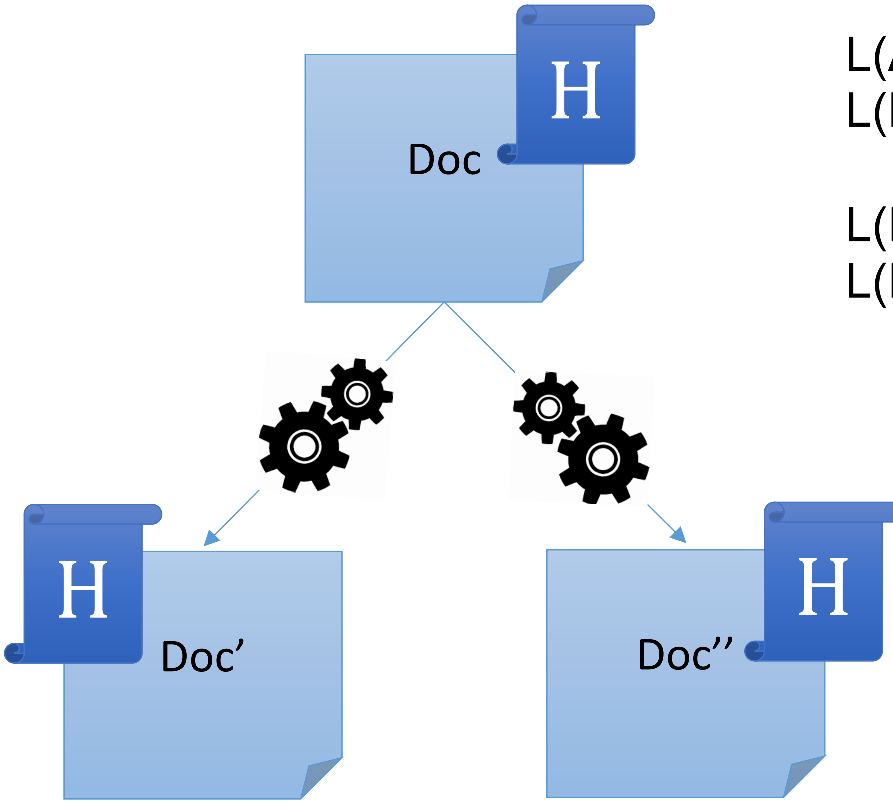
Labels represent policies



Labels represent policies



Labels represent policies



$L(\text{Alice}) = H$
$L(\text{Bob}) = L$

$L(\text{Doc}) \sqsubseteq L(\text{Alice})$
$L(\text{Doc}) \not\sqsubseteq L(\text{Bob})$

Information Flow (IF) Policies

- Focus on **information** not objects
- An IF policy specifies **restrictions** on some data, and on all its derived data.
- IF policy for confidentiality:
  - Value v *and all its derived values* are allowed to be read only by Alice
  - (Different from an access control policy, which would say something like Value v is allowed to be read onl y by Alice)
- The enforcement mechanism **automatically** deduces the restrictions for derived data.

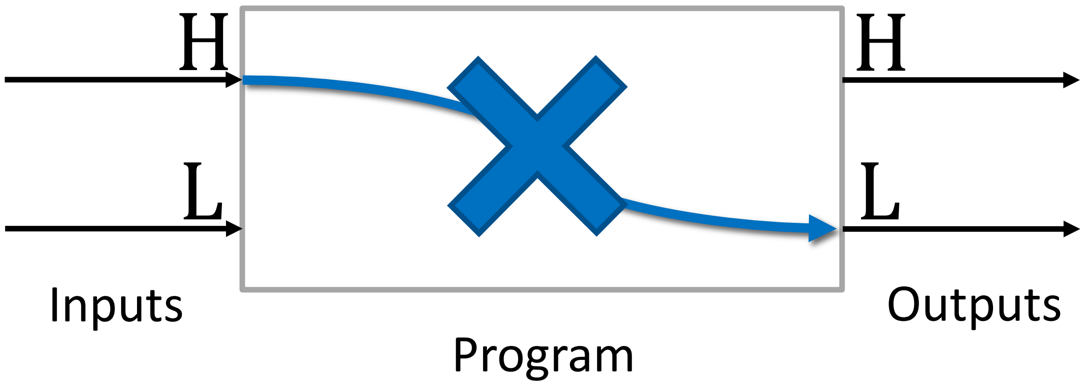
Policy Granularity

- Objects can be system principles (files, programs, sockets…)
- Objects can be program variables

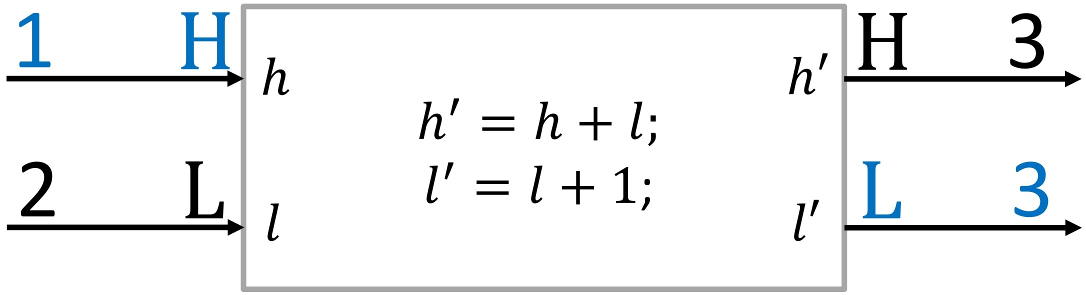
Noninterference [Goguen and Meseguer 1982]

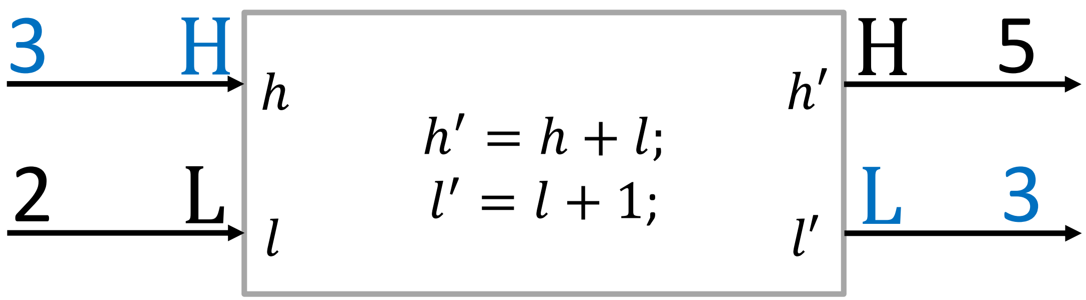
An interpretation of noninterference for a program:

- Changes on H inputs should not cause changes on L outputs.



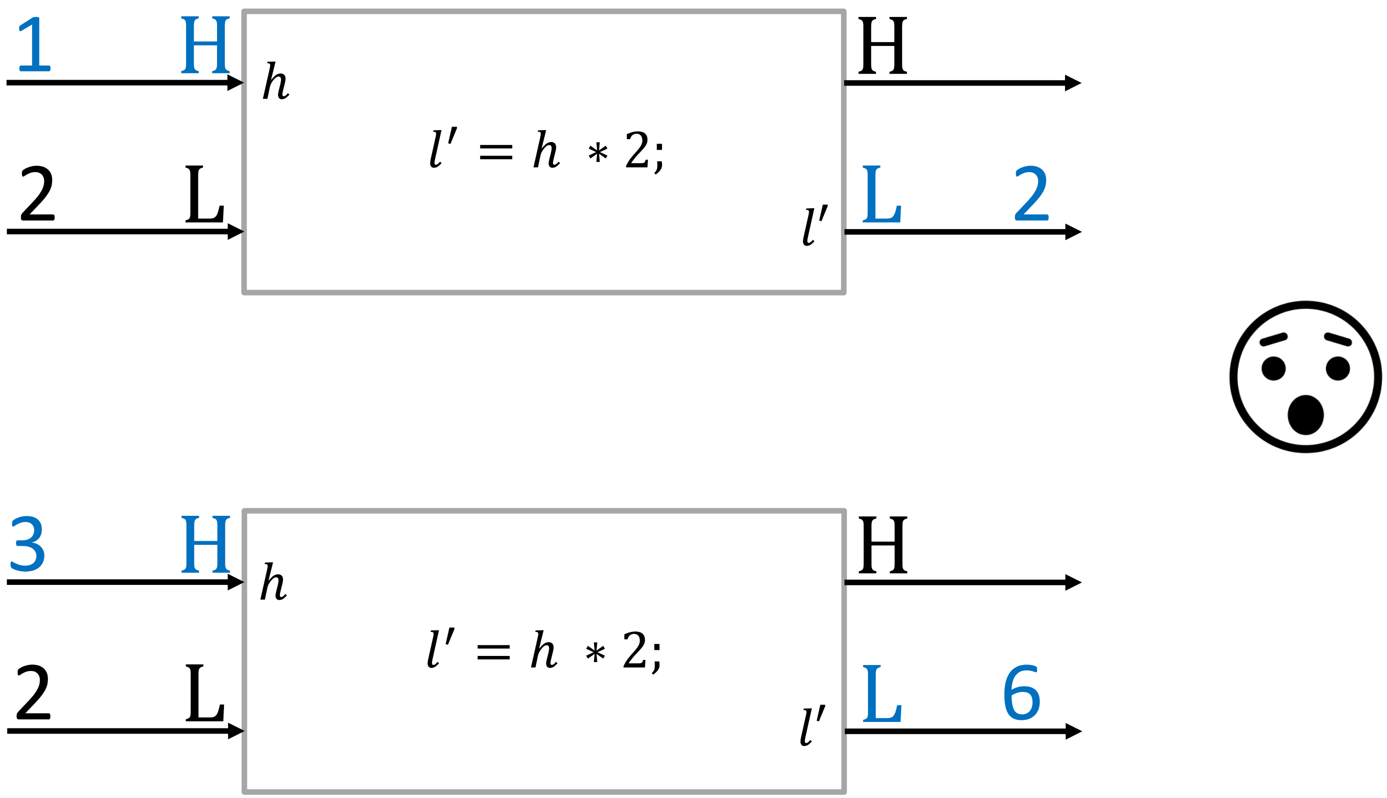
Noninterference: Example





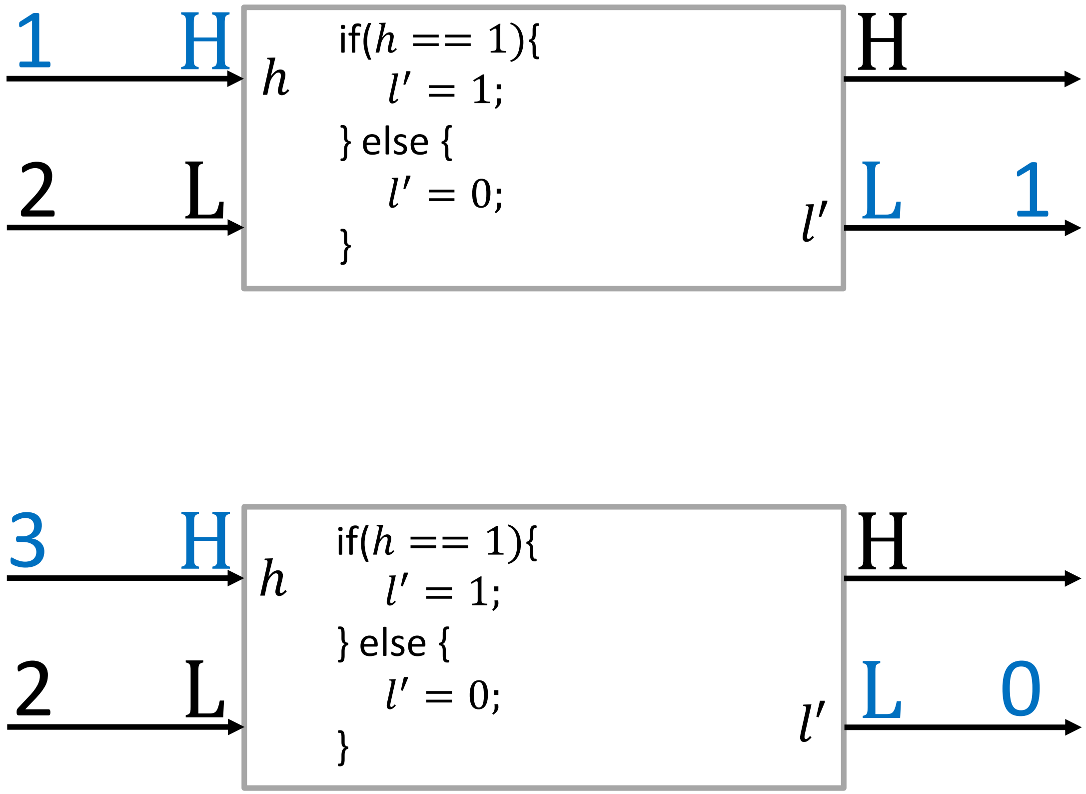
The program satisfies noninterference!

Noninterference: Example



The program does not satisfy noninterference!

Noninterference: Example



The program does not satisfy noninterference!

Noninterference

- Consider a program P.
- Consider two memories M_1 and M_2, such that they agree on values of variables tagged with L: $M_1 =_{\mathrm{L}} M_2$.

(M_1 and M_2 might not agree on values of variables tagged with H)

- $P(M_i)$ are the observations produced by executing C to termination on initial memory M_i:
  - final outputs, or
  - intermediate and final outputs.
- Then, observations tagged with L should be the same:
  - $P(M_1) =_{\mathrm{L}} P(M_2)$.

Noninterference: $\forall M_1, M_2$: if $M_1 =_{\mathrm{L}} M_2$, then $P(M_1) =_{\mathrm{L}} P(M_2)$.

Exercise 1: Noninterference

Assume P_1, P_2 each take two inputs: h_I (label H) and l_i (label L)

1. P_1 outputs (h_O, l_O) where $h_O = h_I || l_I$ and $l_O = l_I$
   - || denotes string concatenation.

2. P_2 outputs l_O where $l_o = \begin{cases} l_I & \text{if } h_I \text{ is even} \\ l_I || l_I & \text{if } h_I \text{ is odd} \end{cases}$

Enforcement Mechanisms

- Static Information Flow Control:
  - type checking
- Dynamic Information Flow Control:
  - taint-tracking
  - runtime monitoring

A simple programming language

```
e ::= n | x | e1+e2 | e1 < e2 | ...

p ::= x = e;
    | p1 p2
    | if(e) then { p1 } else { p2 }
    | while(e){ p }
    | nop;
```

Exercise: A programming language

- Using our simple programming language, write a program that takes one input x_I and ends with an output x_O that is equal to the sum of the odd numbers between 0 and x_I (inclusive)

```
e ::= n | x | e1+e2 | e1 < e2 | …

p ::= x = e; | p1 p2
    | if(e) then { p1 } else { p2}
    | while(e){ p } | nop;
```

Type Systems

- A program is well-typed if all operands are the right type for the operator and all variables are the right type for the expression

```
int x;
string y;
```

```
x = 4 + 5;                   x = "hello" + 5;
y = "hello" + "world";       x = "hello" + "world";
```

- determining that a program is well-typed requires proving that all expressions and all assignments are the right type

Logical Inference

- Syntax for logical Inference: $\frac{premise(s)}{conclusion}$

- Examples:

$$\frac{x{=}4,\ y{=}5}{x{+}y{=}9} \qquad \frac{x{=}True,\ y{=}False}{x\ or\ y = True} \qquad \frac{}{security\ is\ fun!}$$

Type Inference (Expressions)

```
int x;
bool y;
```

$\Gamma(x) = \textbf{int};$

$\Gamma(y) = \textbf{bool};$

- Type environment Γ maps variables to type
- Goal: Judgement (aka proof that) $\Gamma \vdash$ **e** : t
  According to mapping Γ, expression **e** has type t
- Constants: $\dfrac{}{\Gamma \vdash n{::}int}$ $\dfrac{}{\Gamma \vdash True{::}bool}$ $\dfrac{}{\Gamma \vdash False{::}bool}$
- Variables: $\dfrac{\Gamma(\mathrm{x}) = t}{\Gamma \vdash x{::}\mathrm{t}}$
- Expressions: $\dfrac{\Gamma \vdash e1{::}int, \Gamma \vdash e2{::}int}{\Gamma \vdash e1 + e2{::}int}$ $\dfrac{\Gamma \vdash e1{::}int, \Gamma \vdash e2{::}int}{\Gamma \vdash e1 < e2{::}bool}$ …

Example: Type Inferences

- Let $\Gamma(\mathbf{x}) = \text{int}$ and $\Gamma(\mathbf{y}) = \text{int}$.
- What is the type of **x+y+1**?
- *Proof tree:*

$$\dfrac{\dfrac{\dfrac{\Gamma(\mathtt{x}) = \mathit{int}}{\Gamma \vdash \mathtt{x} : \mathit{int}} \quad \dfrac{\Gamma(\mathtt{y}) = \mathit{int}}{\Gamma \vdash \mathtt{y} : \mathit{int}}}{\Gamma \vdash \mathtt{x} + \mathtt{y} : \mathit{int}} \quad \dfrac{}{\Gamma \vdash \mathtt{1} : \mathit{int}}}{\Gamma \vdash \mathtt{x} + \mathtt{y} + \mathtt{1} : \mathit{int}}$$

Exercise: Type Inference

- Let $\Gamma(\mathbf{x}) = \text{int}$ and $\Gamma(\mathbf{y}) = \text{int}$.
- What is the type of **`y>x+5`**?
- *Proof tree:*

$$\dfrac{\dfrac{\Gamma(\mathbf{y}) = \textit{int}}{\Gamma \vdash \mathbf{y} : \textit{int}} \qquad \dfrac{\dfrac{\Gamma(\mathbf{x}) = \textit{int}}{\Gamma \vdash \mathbf{x} : \textit{int}} \qquad \dfrac{}{\Gamma \vdash 5 : \textit{int}}}{\Gamma \vdash \mathbf{x} + 5 : \textit{int}}}{\Gamma \vdash \mathbf{y} > \mathbf{x} + 5 : \textit{bool}}$$

Label Inference (Expressions)

- Type environment Γ maps variables to ~~type~~ label
- Goal: Judgement (aka proof that) $\Gamma \vdash \mathbf{e} : \ell$

  According to mapping Γ, expression **e** has label ℓ

$\Gamma(x) = \boldsymbol{L};$

$\Gamma(y) = \mathbf{H};$

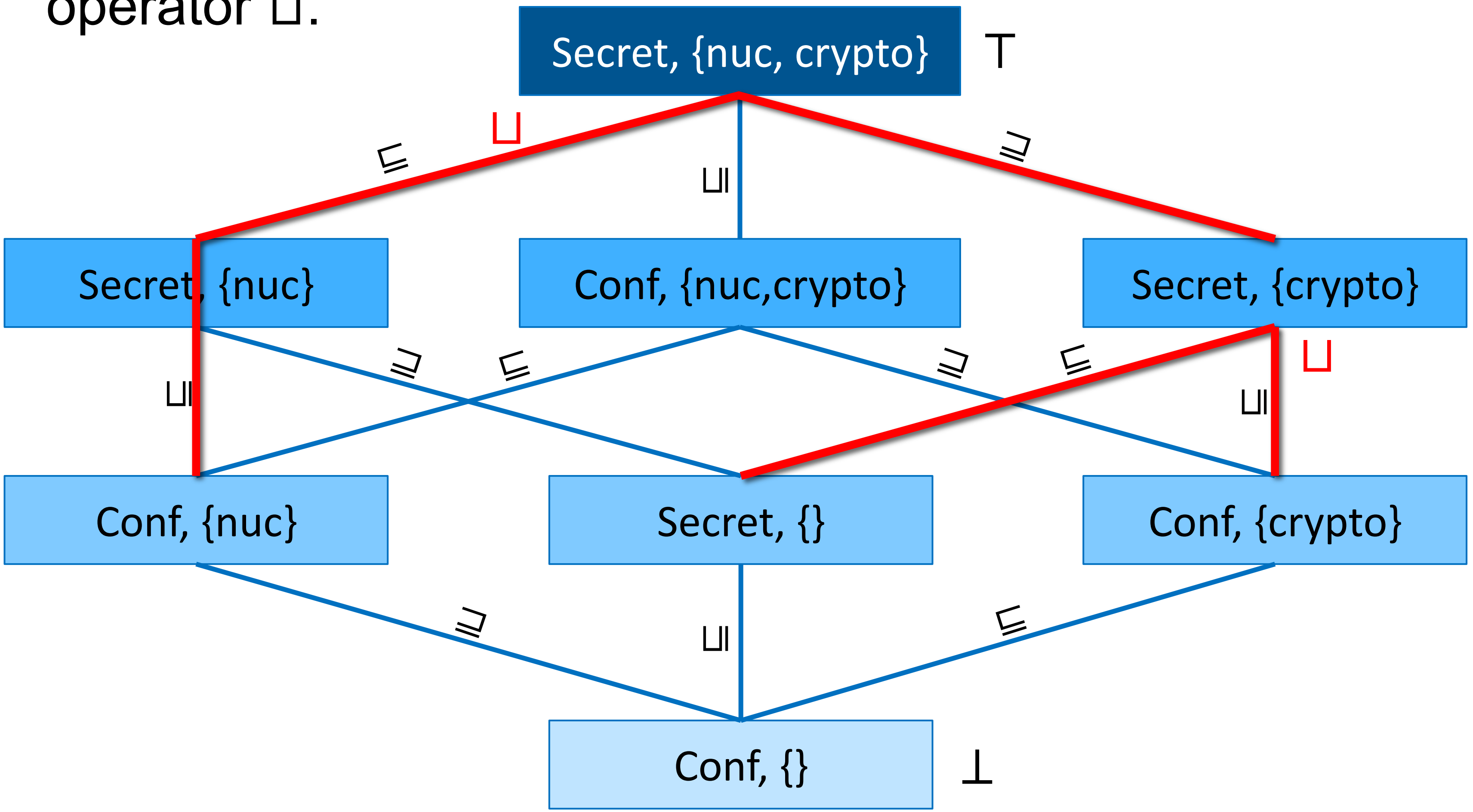
- Constants: $\dfrac{}{\Gamma \vdash n{::}L}$
- Variables: $\dfrac{\Gamma(\mathrm{x})=\ell}{\Gamma \vdash x{::}\ell}$
- Expressions:

Join Operator for combining labels

- For each $\ell 1$ and $\ell 2$, there exists a label $\ell 3$, such that:
  - $\ell 1 \sqsubseteq \ell 3$
  - $\ell 2 \sqsubseteq \ell 3$
  - for all $\ell 4$ such that $\ell \sqsubseteq \ell 4$ and $\ell 2 \sqsubseteq \ell 4$, then $\ell 3 \sqsubseteq \ell 4$.
- $\ell 3$ is called the **join** of ℓ and $\ell 2$ and denoted $\ell 1 \sqcup \ell 2$
- Operator $\sqcup$ is associative and commutative.

Lattice of labels

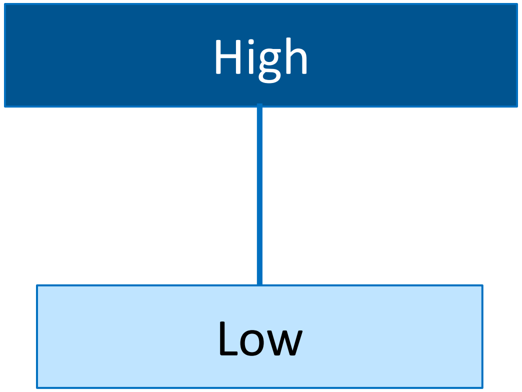
- The set of labels and relation ⊑ define a lattice, with join operator ⊔.



Exercise: Join

- What are the following labels (H or L)?

1. $H \sqcup H$
2. $H \sqcup L$
3. $L \sqcup H$
4. $L \sqcup L$



Label Inference (Expressions)

- Type environment Γ maps variables to ~~type~~ label
- Goal: Judgement (aka proof that) $\Gamma \vdash$ **e** : ℓ
  According to mapping Γ, expression **e** has label ℓ

$\Gamma(x) = \boldsymbol{L}\,;$
$\Gamma(y) = \mathbf{H}\,;$

- Constants: $\dfrac{}{\Gamma \vdash n{::}L}$
- Variables: $\dfrac{\Gamma(\mathrm{x}) = \ell}{\Gamma \vdash x{::}\ell}$
- Expressions: $\dfrac{\Gamma \vdash e1{::}\ell 1,\ \Gamma \vdash e2{::}\ell 2}{\Gamma \vdash e1 + e2{::}\ell 1 \sqcup \ell 2}$ $\quad \dfrac{\Gamma \vdash e1{::}\ell 1,\ \Gamma \vdash e2{::}\ell 2}{\Gamma \vdash e1 < e2{::}\ell 1 \sqcup \ell 2}$ $\quad \ldots$

Example

- Let $\Gamma(\mathbf{x}) = \mathrm{L}$ and $\Gamma(\mathbf{y}) = \mathrm{H}$.
- What is the type of **`x+y+1`**?
- *Proof tree:*

$$\dfrac{\dfrac{\Gamma(\mathbf{x}) = \mathrm{L}}{\Gamma \vdash \mathbf{x} : \mathrm{L}} \qquad \dfrac{\Gamma(\mathbf{y}) = \mathrm{H}}{\Gamma \vdash \mathbf{y} : \mathrm{H}} \qquad \dfrac{}{\Gamma \vdash \mathbf{1} : \mathrm{L}}}{\Gamma \vdash \mathbf{x + y + 1} : \mathrm{H}}$$

Exercise

- Let $\Gamma(\mathbf{x}) = \mathrm{L}$ and $\Gamma(\mathbf{y}) = \mathrm{H}$.
- What is the type of **y>x+5**?
- *Proof tree:*

$$
\dfrac{\dfrac{\Gamma(\mathbf{y}) = \mathrm{H}}{\Gamma \vdash \mathbf{y} : \mathrm{H}} \qquad \dfrac{\dfrac{\Gamma(\mathbf{x}) = \mathrm{L}}{\Gamma \vdash \mathbf{x} : \mathrm{L}} \qquad \dfrac{}{\Gamma \vdash \mathbf{5} : \mathrm{L}}}{\Gamma \vdash \mathbf{x + 5} : \mathrm{L}}}{\Gamma \vdash \mathbf{y > x + 5} : \mathrm{H}}
$$

Type Checking (Programs)

- When is a one-line program `x = e;` well-typed?

Static type system

Assignment-Rule: $$\frac{\Gamma \vdash \mathbf{e} : \mathrm{t} \qquad \mathrm{t} \sqsubseteq \Gamma(\mathbf{x})}{\Gamma \vdash \mathbf{x{=}e;}}$$

Sequence-Rule: $$\frac{\Gamma \vdash \mathbf{p1} \qquad \Gamma \vdash \mathbf{p2}}{\Gamma, ctx \vdash \mathbf{p1\ p2}}$$

If-Rule: $$\frac{\Gamma \vdash \mathbf{e} : \mathrm{bool} \qquad \Gamma \vdash \mathbf{p1} \qquad \Gamma \vdash \mathbf{p2}}{\Gamma \vdash \mathbf{if(e)\ then\{\ p1\ \}\ else\{\ p2\ \}}}$$

While-Rule: $$\frac{\Gamma \vdash \mathbf{e} : \mathrm{bool} \qquad \Gamma \vdash \mathbf{p}}{\Gamma \vdash \mathbf{while(e)\{\ p\ \}}}$$

Skip-Rule: $$\frac{}{\Gamma \vdash \mathbf{nop;}}$$

Enforcing Information Flow

- Goal: Design a type system such that

$\Gamma \vdash \mathbf{p} \;\; \Rightarrow \;\; \mathbf{p}$ satisfies NonInterference