Lecture 17: Information Flow

CS 181S

November 12, 2018

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

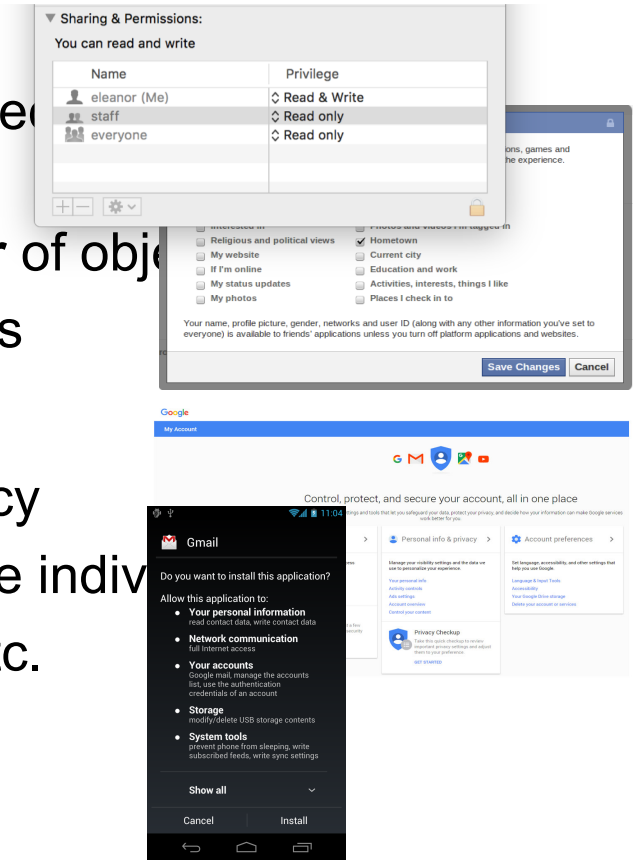
Access Control Policy

- An **access control policy** specifies which of the **operations** associated with any given **object** each **subject** is authorized to perform
- Expressed as a relation $Auth$:

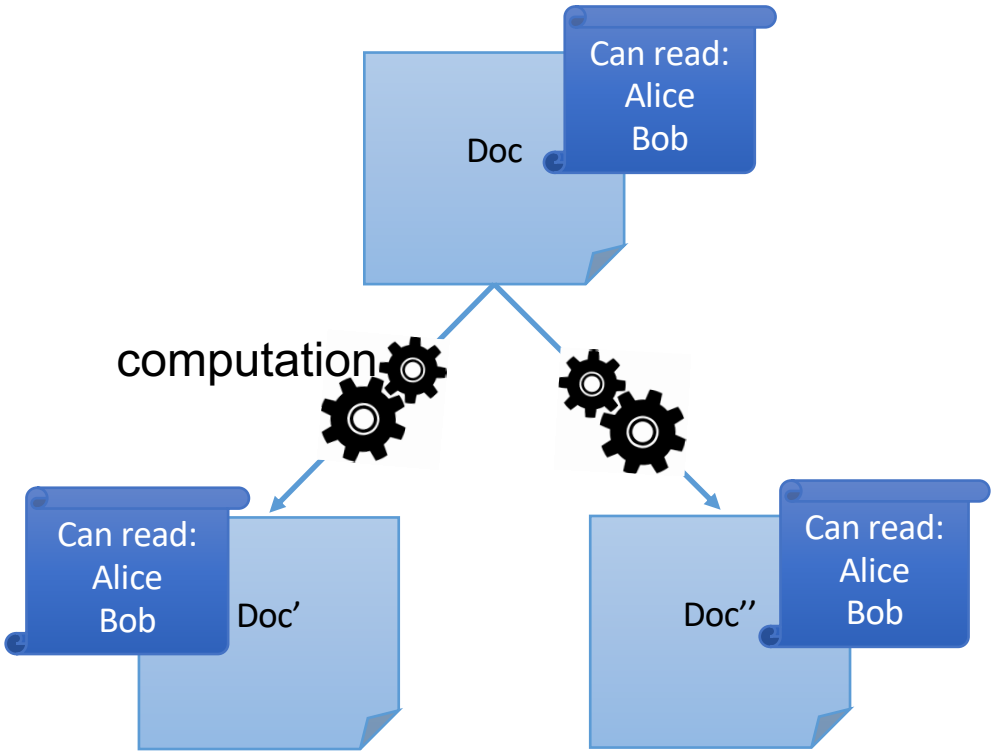
$Auth$		Objects	
		dac.tex	dac.pptx
subject	ebirrell	r,w	r,w
	clarkson	r	r
	student		r

Who defines Policies?

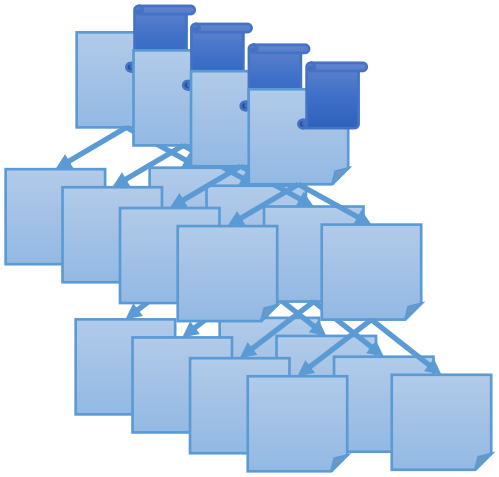
- Discretionary access control (DAC)
  - **Philosophy:** users have the *discretion* to spe themselves
  - Commonly, information belongs to the **owner** of obj
  - Access control lists, privilege lists, capabilities
- Mandatory access control (MAC)
  - **Philosophy:** central authority *mandates* policy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



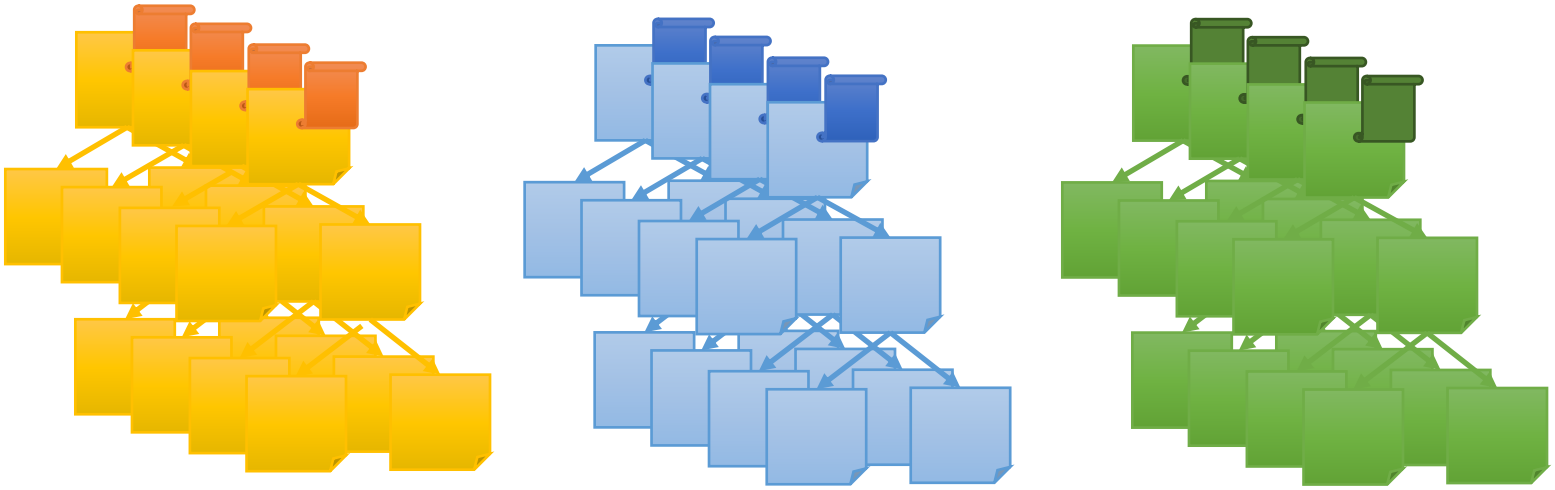
Access control for computed data



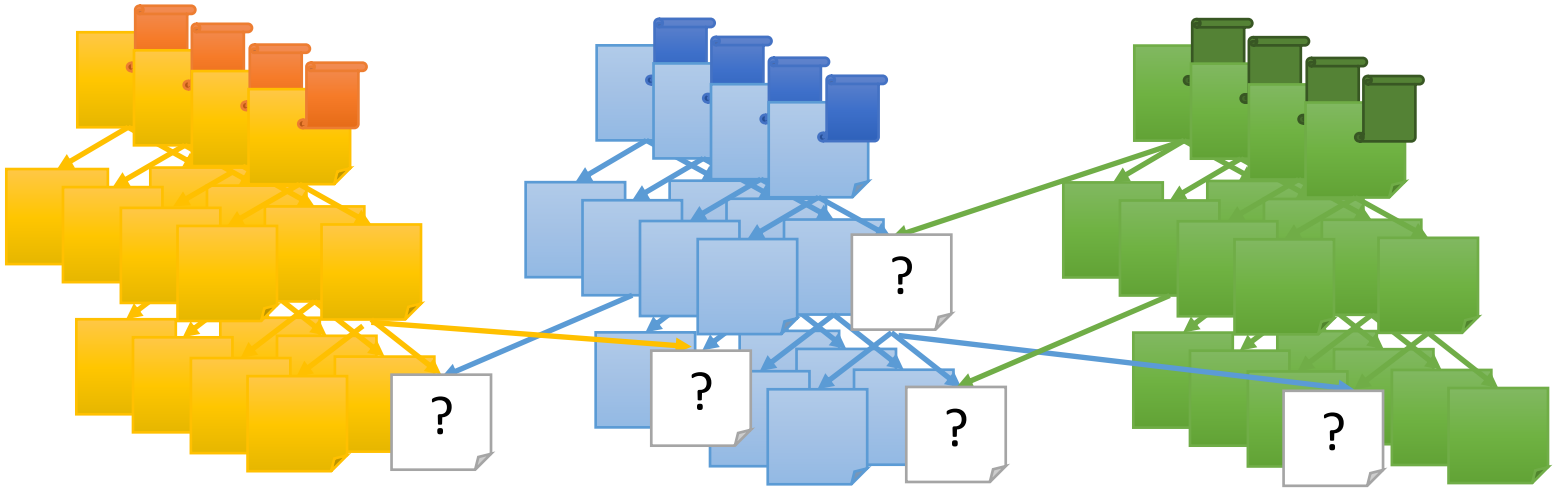
Scaling to many pieces of data…



Scaling to many users…

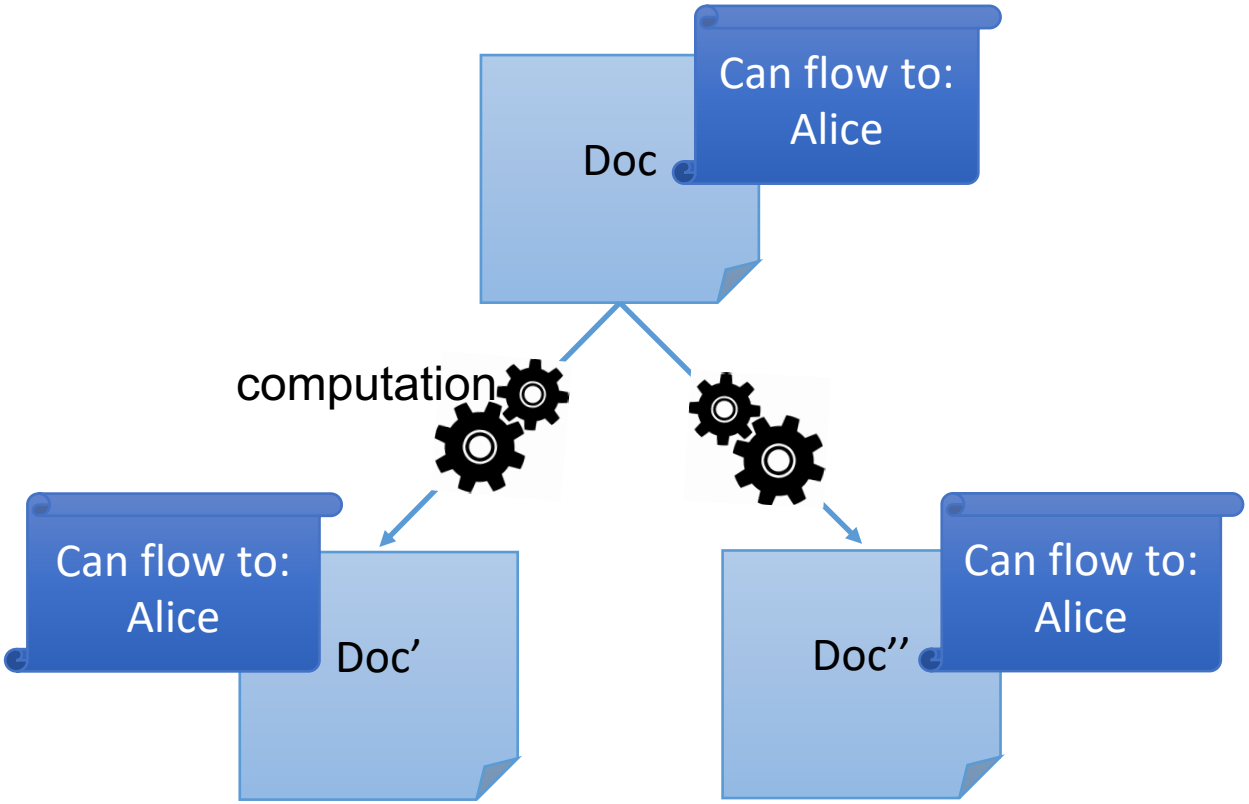


Scaling to many interactions…



Need to assign restrictions in an automatic way.

Information flow policies



Automatic deduction of policies!

Information Flows between Principals

- Channel: means to communicate information
- Storage channel: written by one program and read by another
- Legitimate channel: intended for communication between programs
- Covert channel: not intended for information transfer yet exploitable for that purpose

Information Flow (IF) Policies

- Focus on **information** not objects
- An IF policy specifies **restrictions** on the associated 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

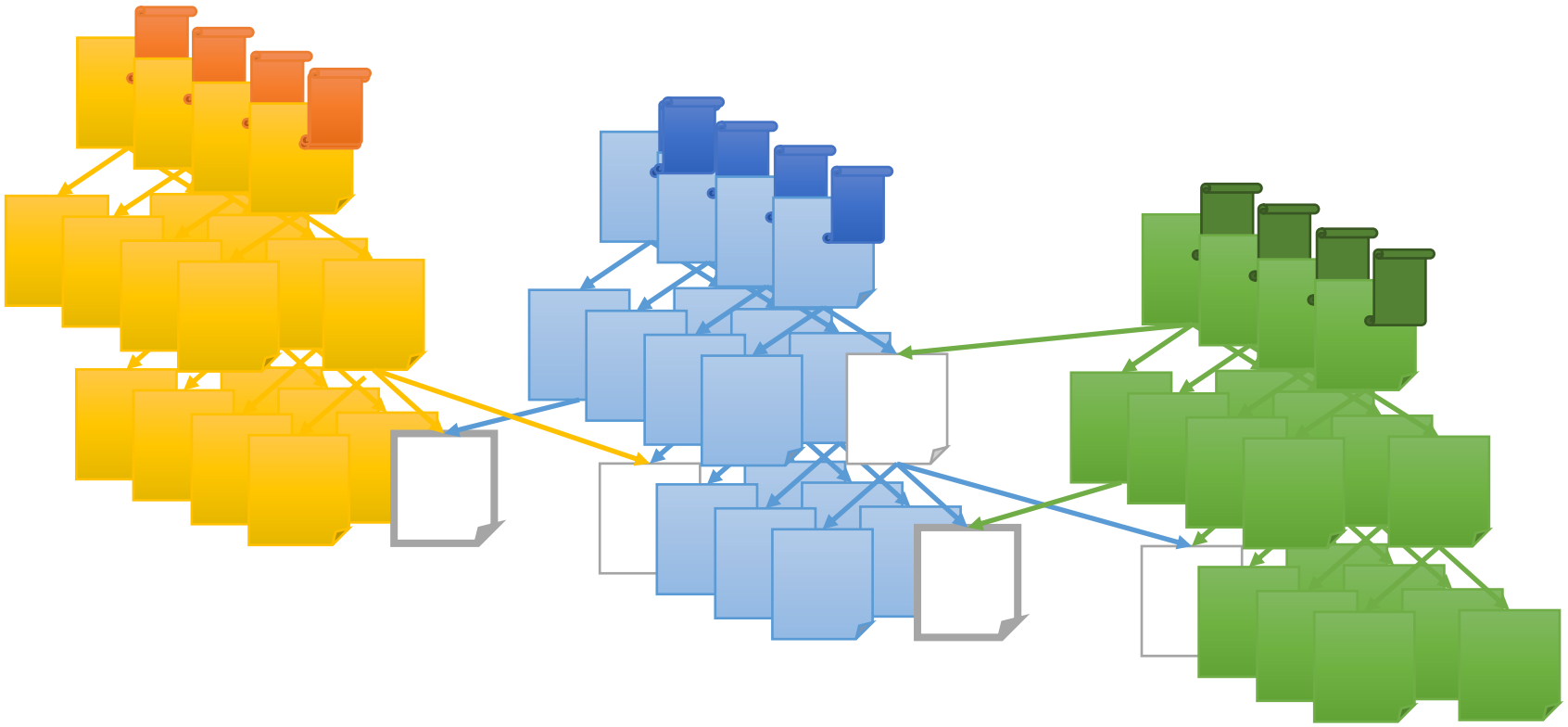
Different from the access control policy:
Value v is allowed to be read at most by Alice.

- The enforcement mechanism **automatically** deduces the restrictions for derived data.

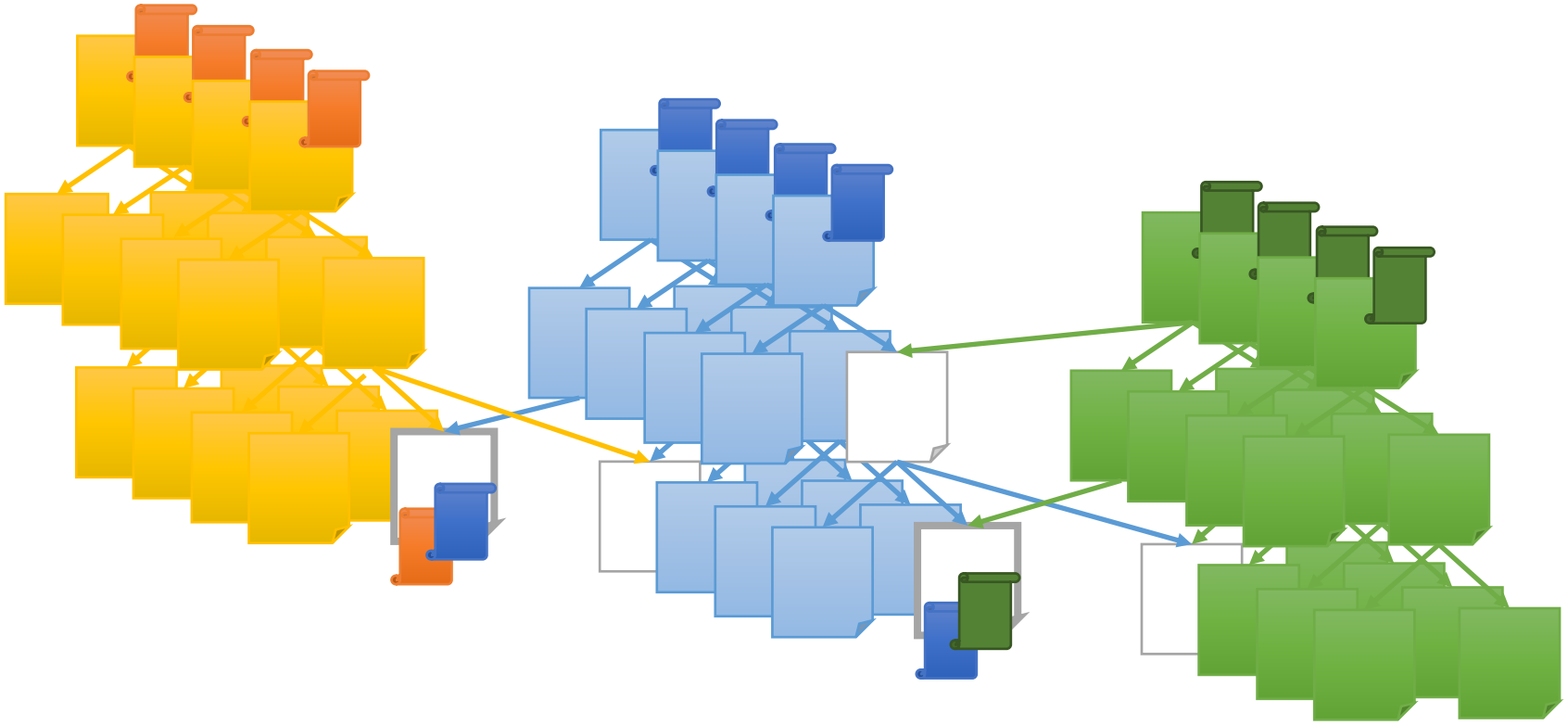
Policy Granularity

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

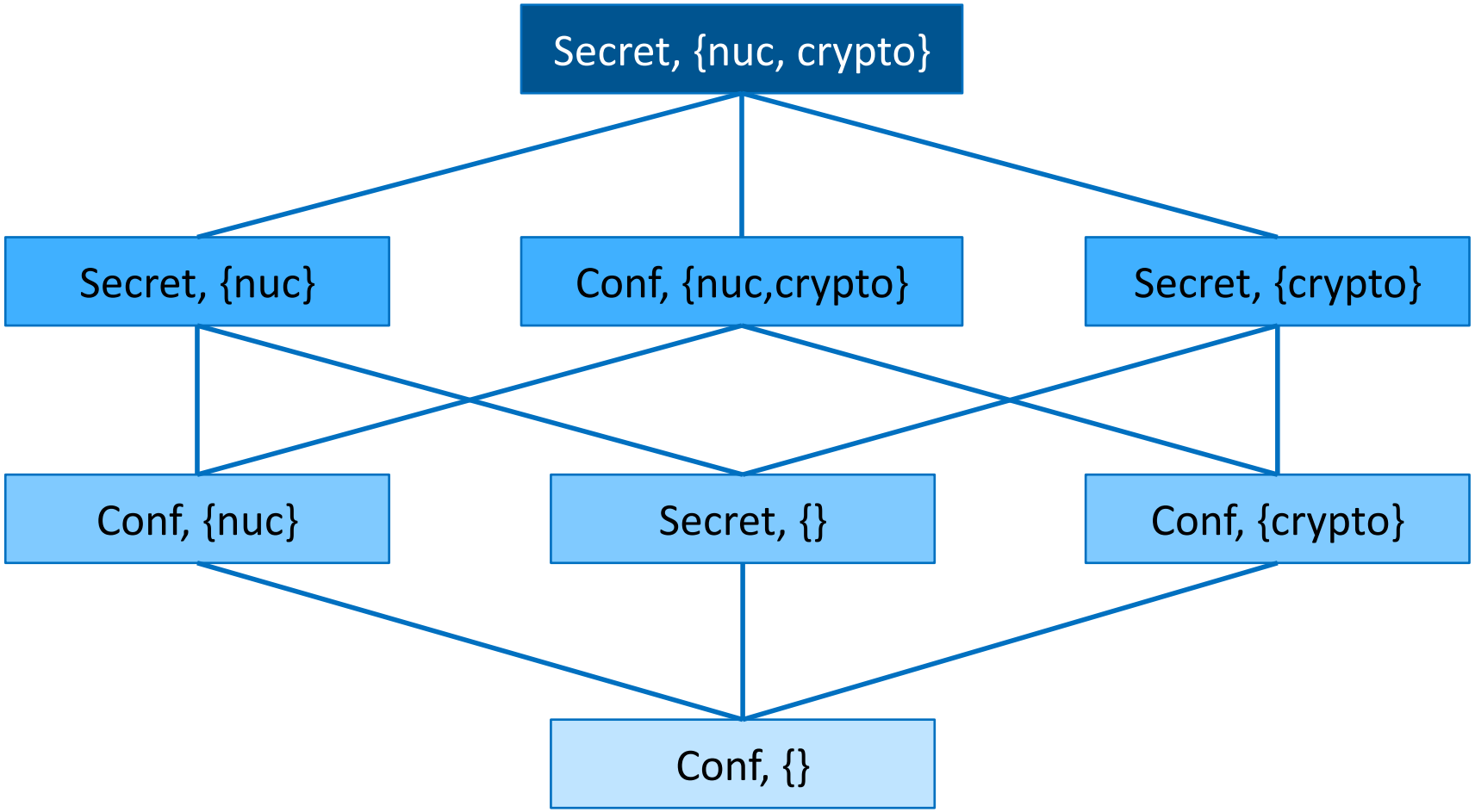
Scaling to many interactions…



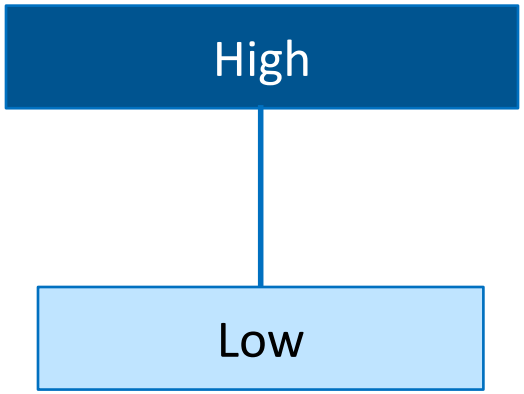
Scaling to many interactions…



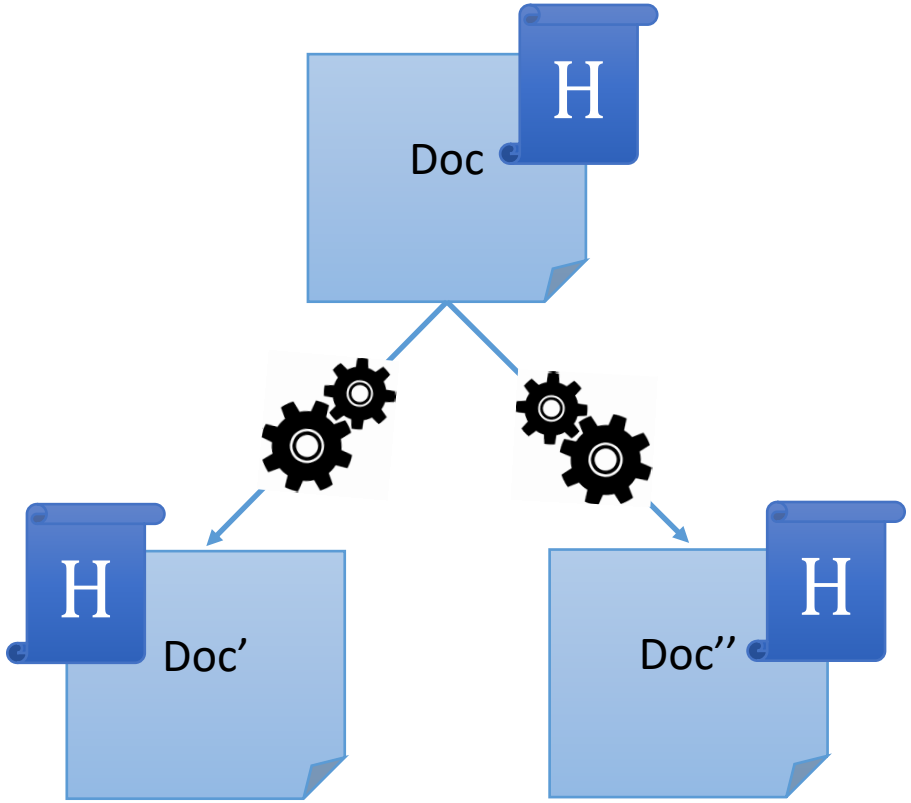
Labels represent policies



Labels represent policies



Labels represent policies

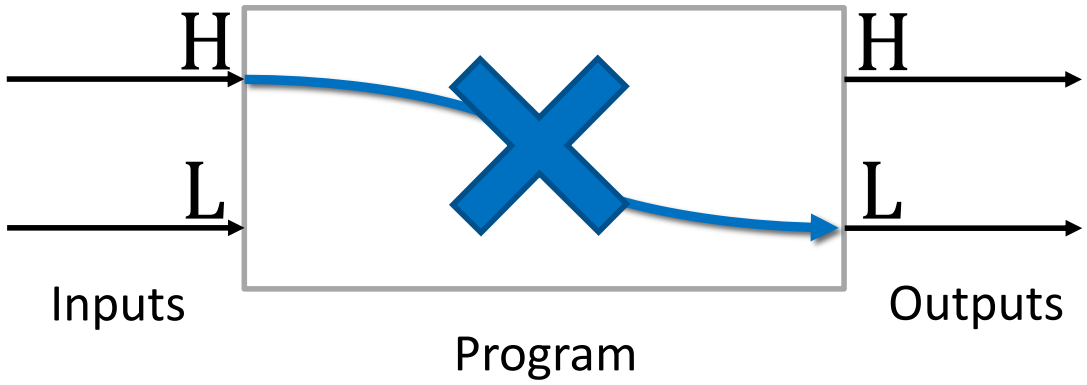


Noninterference

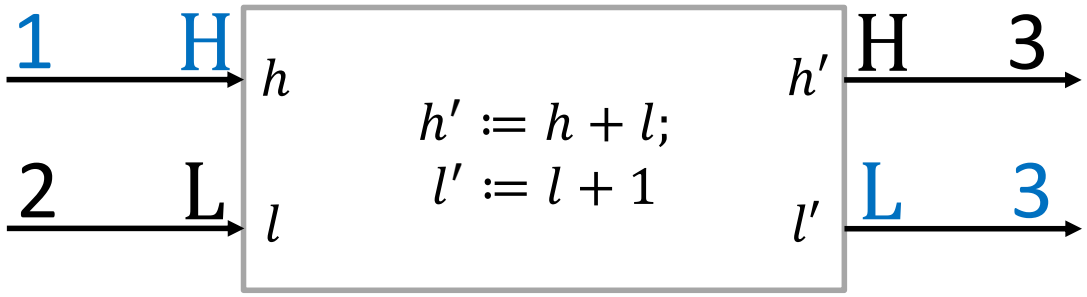
[Goguen and Meseguer 1982]

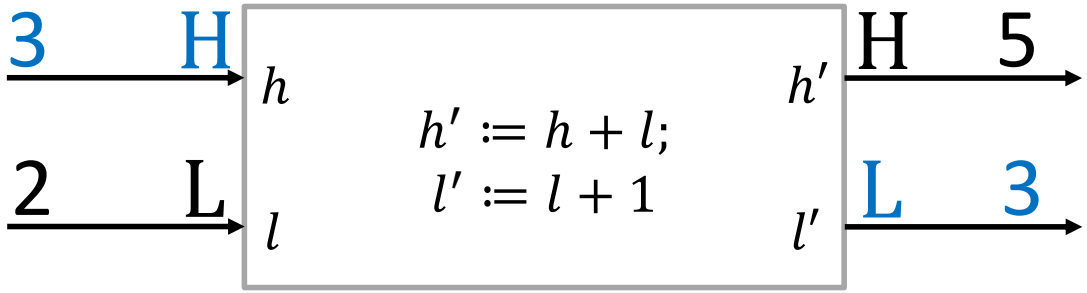
An interpretation of noninterference for a program:

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



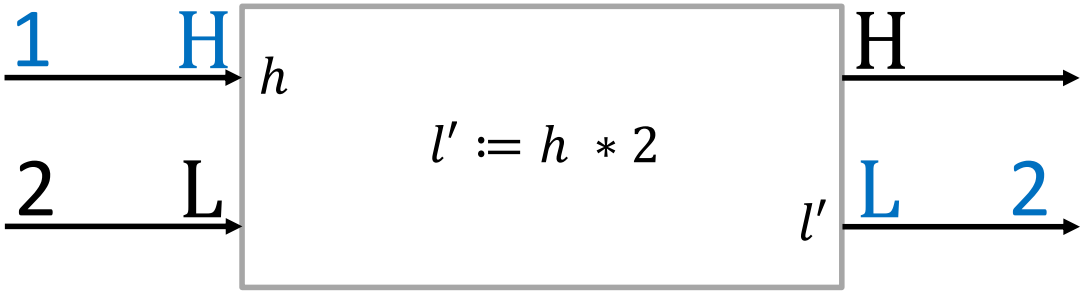
Noninterference: Example

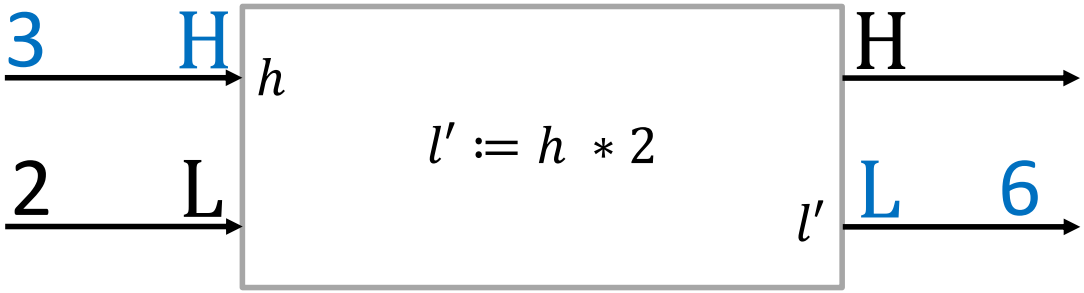




The program satisfies noninterference!

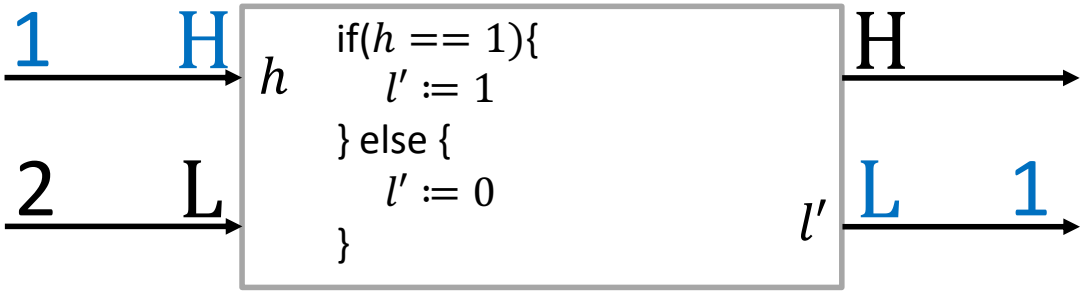
Noninterference: Example

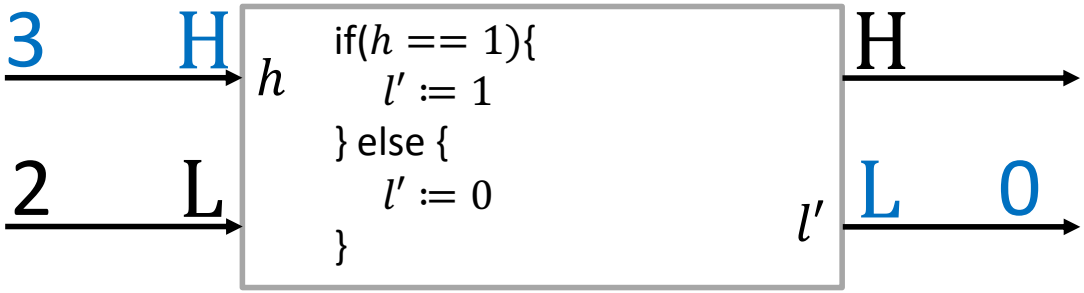




The program does not satisfy noninterference!

Noninterference: Example





The program does not satisfy noninterference!

Noninterference

- Consider a program C.
- 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.

- $C(M_i)$ 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:
  - $C(M_1) =_{\mathrm{L}} C(M_2)$.

Noninterference

For a program C and a mapping from variables to labels in $\{\mathrm{L}, \mathrm{H}\}$:

$$\forall M_1, M_2\colon \quad \text{if } M_1 =_{\mathrm{L}} M_2, \text{ then } C(M_1) =_{\mathrm{L}} C(M_2).$$

Examples

- P outputs (H_O, L_O) where $H_O = H_I || L_I$ and $L_O = L_I$
  - || denotes string concatenation.

- P outputs L_O where $L_o = \begin{cases} L_I & \text{if } H_I \text{ is even} \\ L_I || L_I & \text{if } H_I \text{ is odd} \end{cases}$

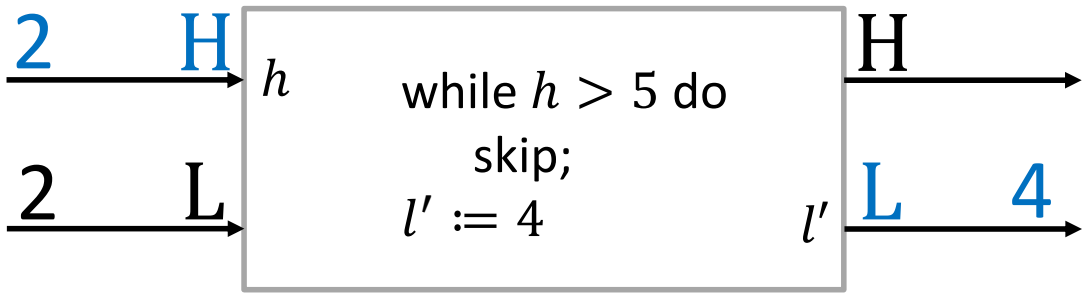
Examples

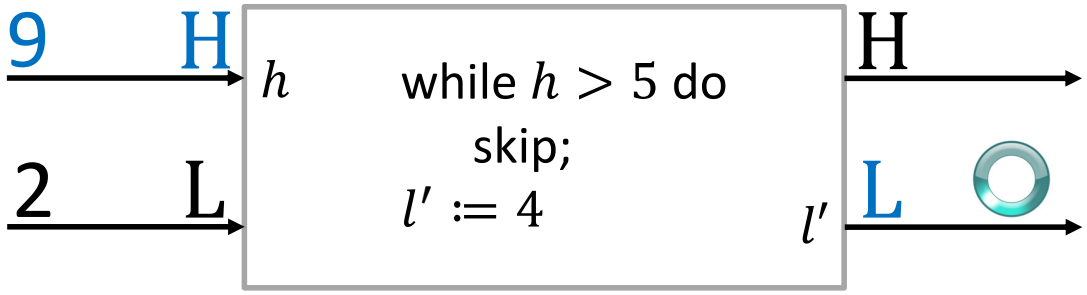
- P := while $H_I > 5$ do skip; $L_O := 4$

- P outputs $L_O = H_I \oplus k$ where k is a freshly generated, uniformly random number 32-bit binary string
  - Assume H_I is always a 32-bit binary string.

- P outputs $L_O = \mathrm{Enc}(H_I; L_I)$
  - Assume L_I is an RSA public key

Less restrictive than necessary…





Termination sensitive noninterference

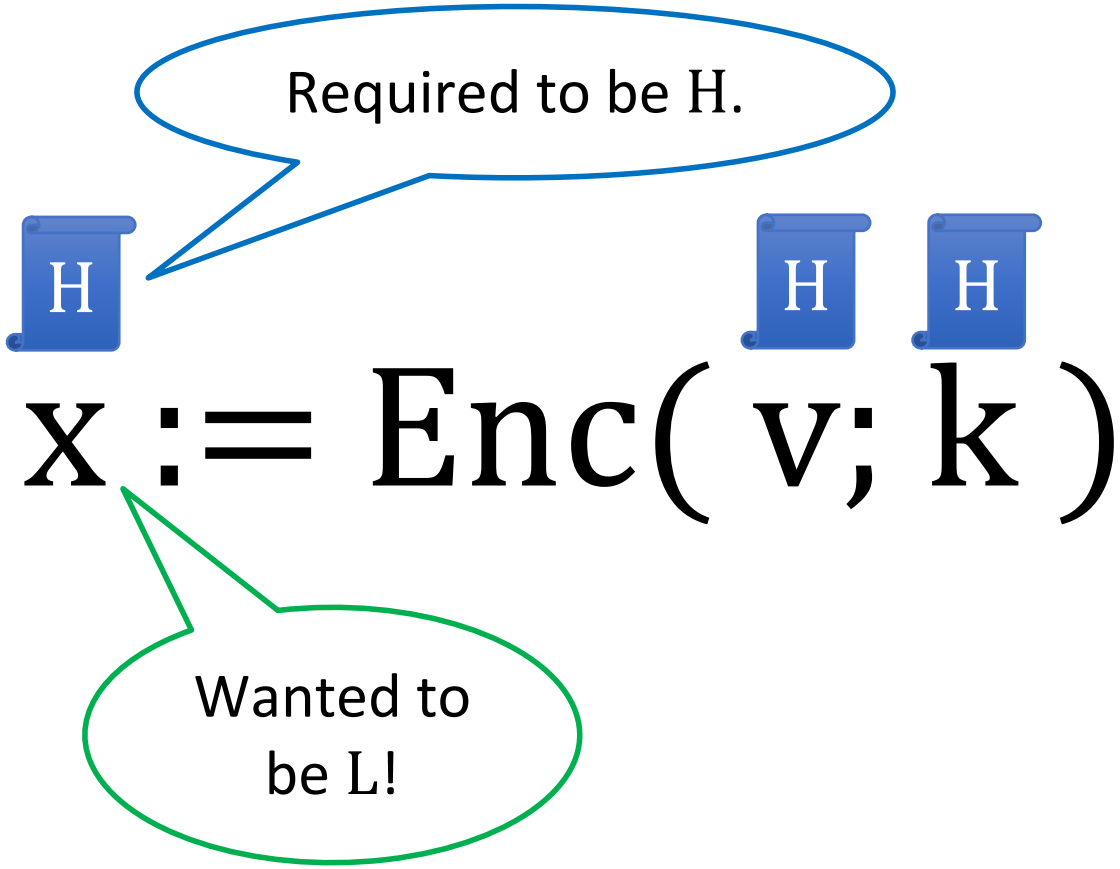
- If
  - $M_1 =_{\mathrm{L}} M_2$,
- then
  - **C terminates on M_1 iff C terminates on M_2, and**
  - $C(M_1) =_{\mathrm{L}} C(M_2)$.

Probabilistic Randomness

- **Probabilistic Noninterference:** For a program C and a mapping from variables to labels in $\{\mathrm{L}, \mathrm{H}\}$, the output distribution $\mathrm{H_I} = C(H_I, L_I)$ is independent of H_I

- **Computational Probabilistic Noninterference**

Computational Probabilistic Noninterference

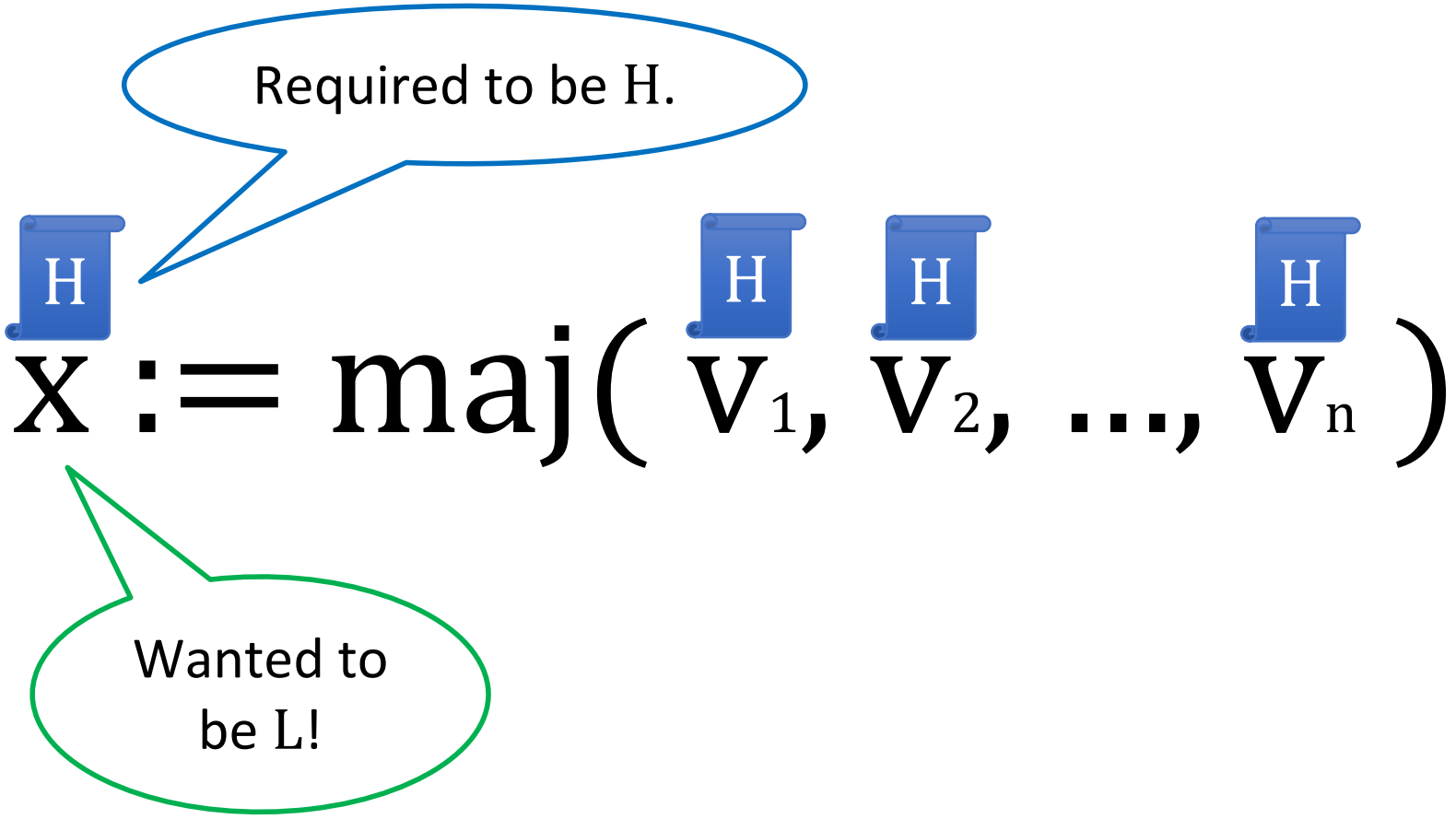


Examples

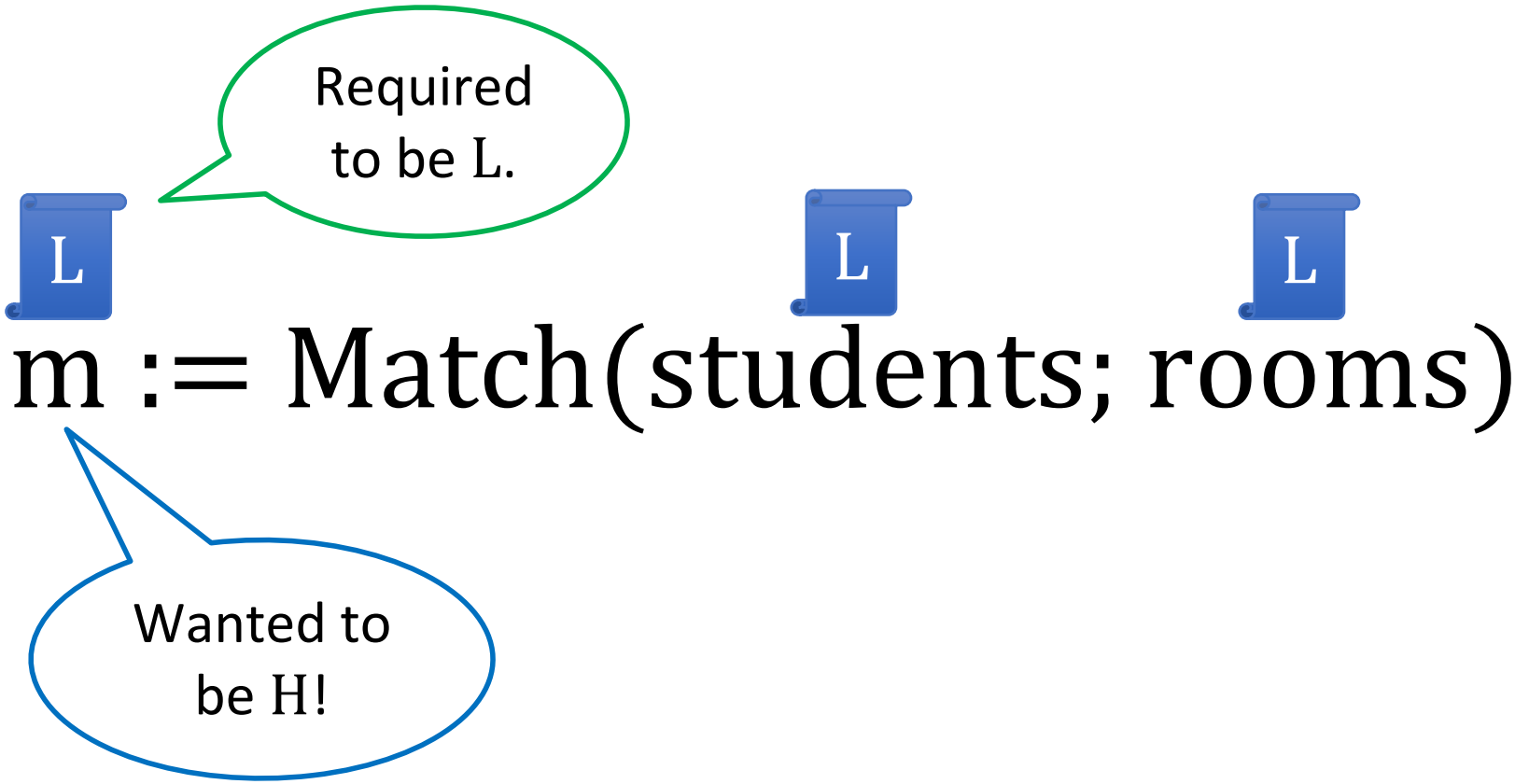
- P takes a list of ballots is H_I and returns L_O, the results of the election (which candidate receives a plurality of the vote)

- P takes a list of students at Pomona $L_{I,1}$ and a list of dorm rooms $L_{I,2}$ and returns a L_O, a list of room assignments

More restrictive than necessary…



Less restrictive than necessary…



Declassification

- **What**: specify what information may be declassified
  - e.g., LastFourDigits(credit card number) should be low
  - Partial Equivalence Relation (PER) Model, Reactive NI
- **Who**: specify who may declassify information
  - e.g., high object owner can write to low objects
  - Decentralized Label Model, robust declassification
- **Where**: specify which pieces of code may declassify
  - e.g., encryption function can write to low objects
  - Intransitive Noninterference, Constrained Noninterference
- **When**: specify when information may be declassified
  - e.g., software key may be shared after payment has been received
  - Temporal, Relative, Probabilistic

Enforcement Mechanisms

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