

CS 5430

Information-Flow Policies

Elisavet Kozyri
Spring 2017

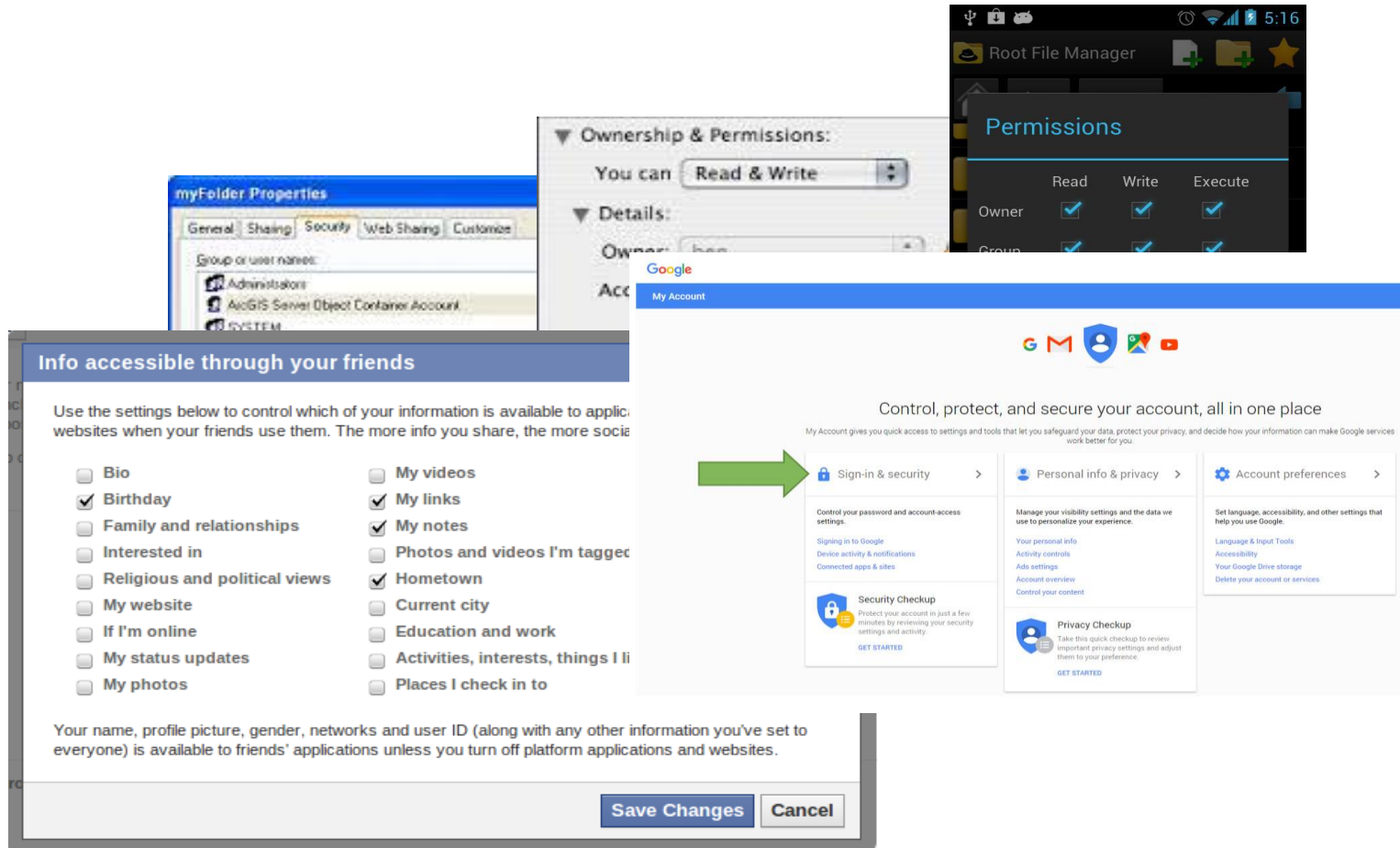
Restrictions on data



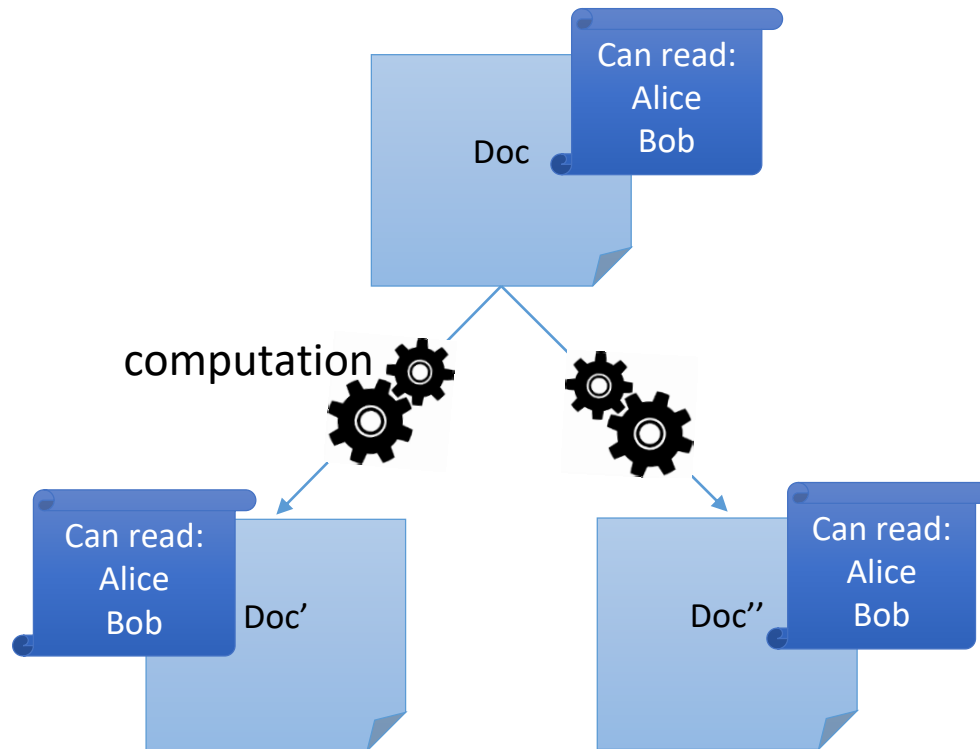
Restrictions on data

- Confidentiality
 - Who can read data.
- Integrity
 - How much trusted data is.
- Privacy
 - What operations can be applied on data.

Access control for enforcing restrictions

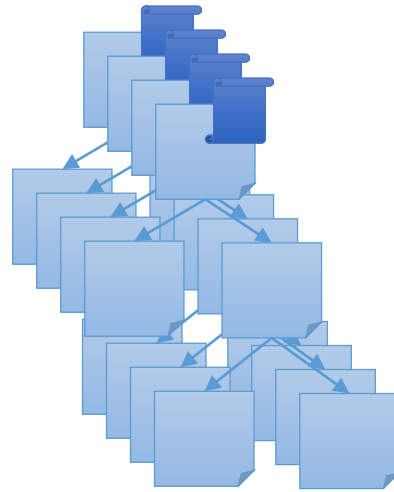


Access control for computed data

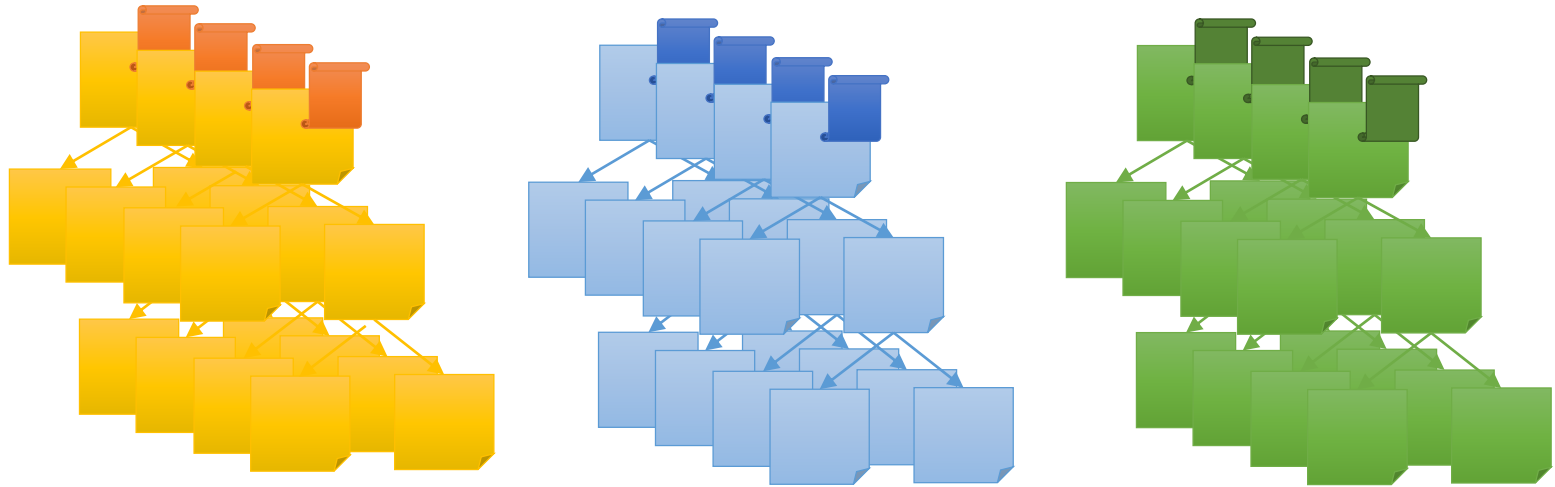


Manual assignment
of access control
policies to
computed data!

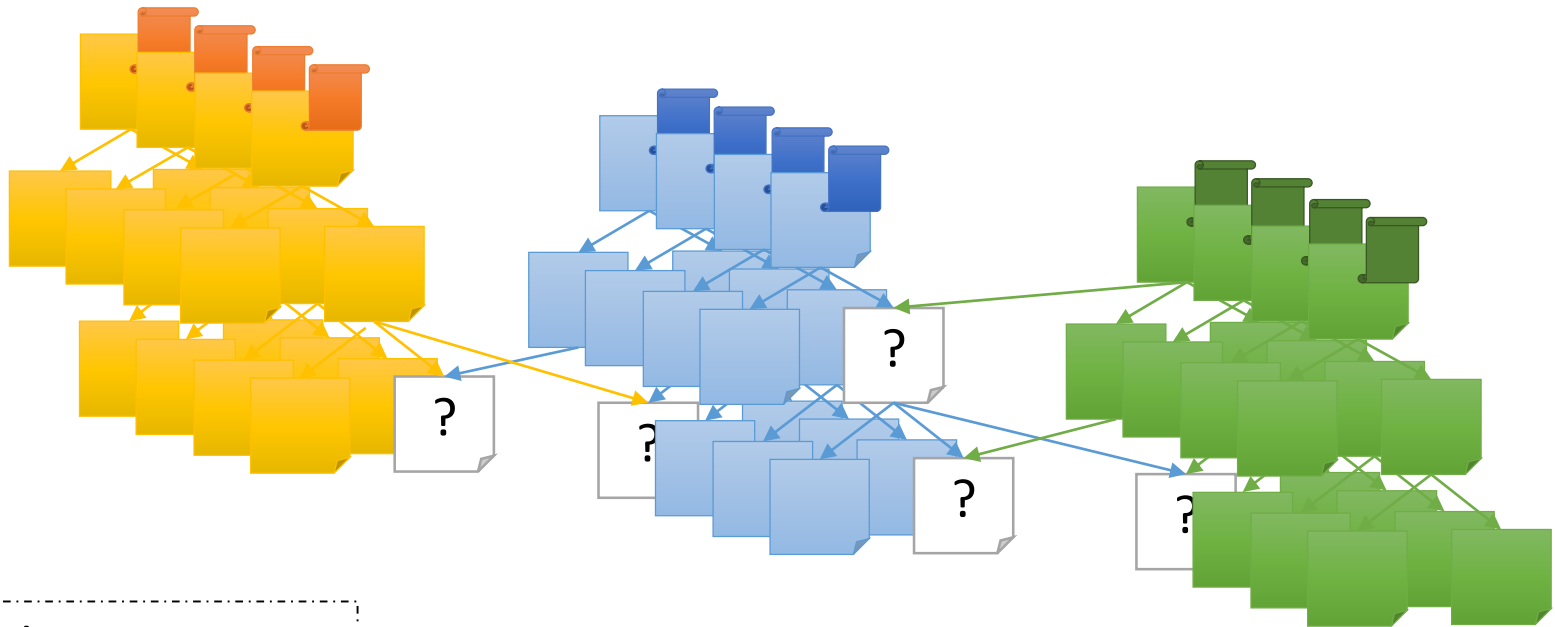
Scaling to many pieces of data...



Scaling to many users...



Scaling to many interactions...



Need to assign
restrictions in an
automatic way.

Limits of access control

- Not suitable for the Big Data era.
- [Lampson 1973] Malicious program could:
 - Leak information in **metadata** (billing reports, nonces chosen in protocols, ...)
 - Use **shared resources** and OS API to encode information (e.g., file locking, CPU cycles)

Limits of access control

- Not suitable for the Big Data era.
- [Lampson 1973] Malicious program could:
 - Leak information in **metadata** (billing reports, nonces chosen in protocols, ...)
 - Use **shared resources** and OS API to encode information (e.g., file locking, CPU cycles)

↓
Covert channels:

not intended for information
transfer yet exploitable for that purpose

Information Flow (IF) Policies

- 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 at most by Alice.

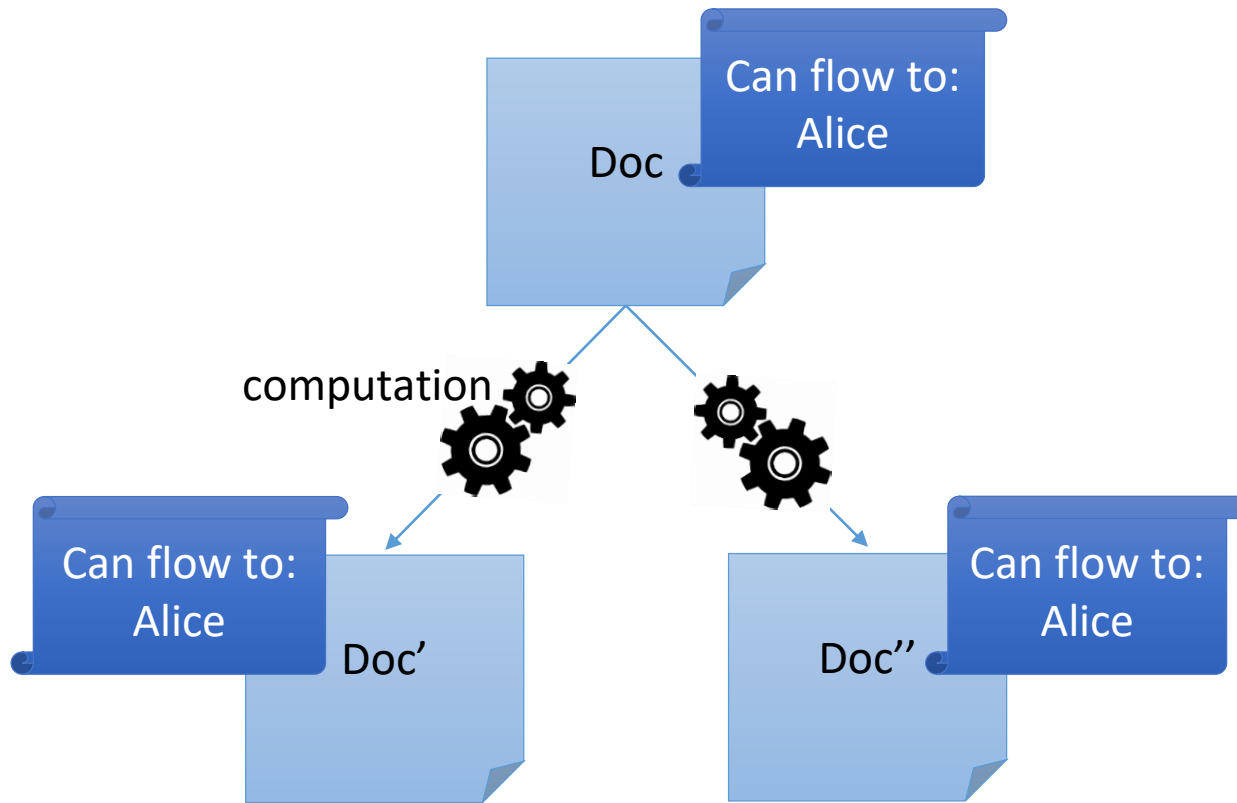


Different from the access control policy:
Value v is allowed to be read at most by Alice.

Information Flow (IF) Policies

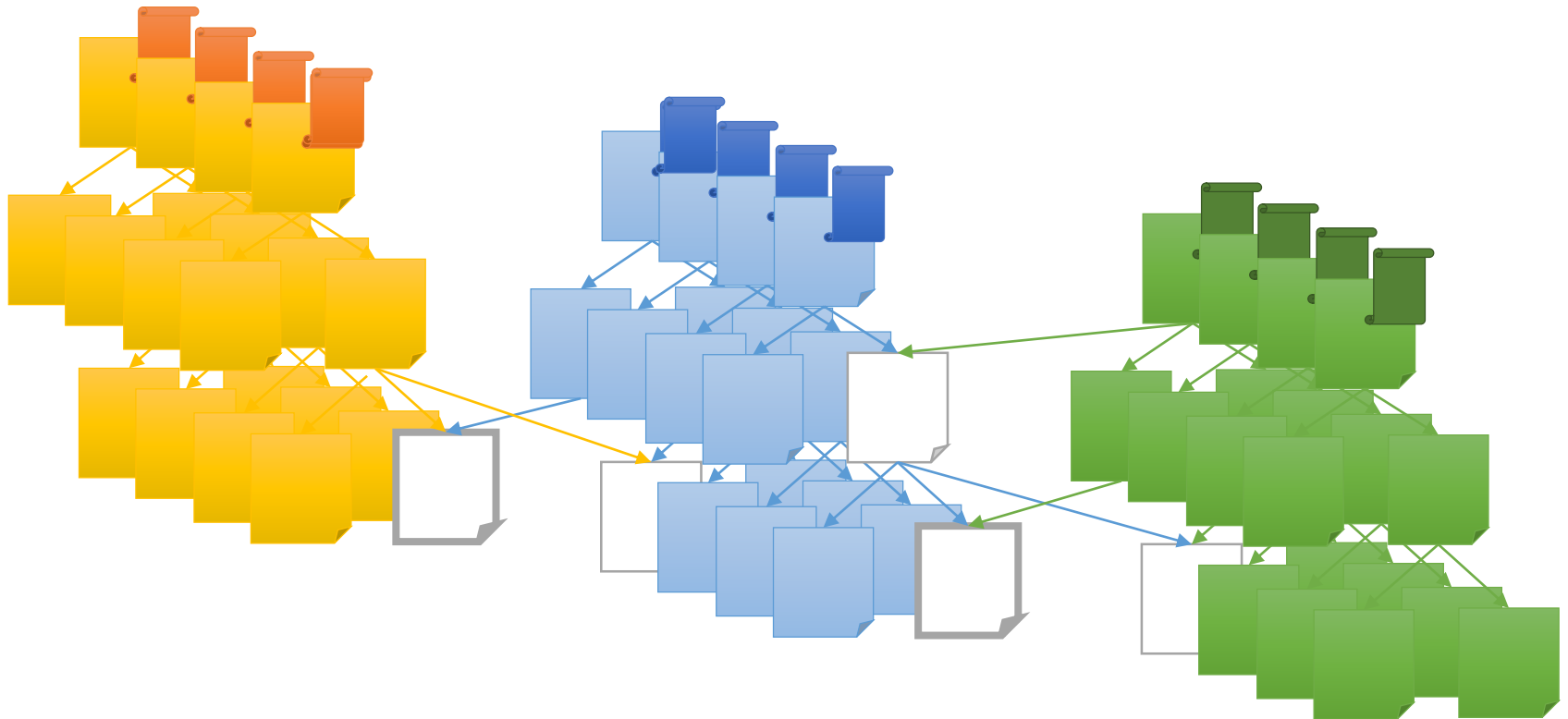
- An IF policy specifies **restrictions** on the associated data, and on all its derived data.
- IF policy for confidentiality:
 - Value v *and all its derived values* are allowed to be read at most by Alice.
 - Equivalently, v is allowed to **flow** only to Alice.
- The enforcement mechanism **automatically** deduces the restrictions for derived data.

Information flow policies

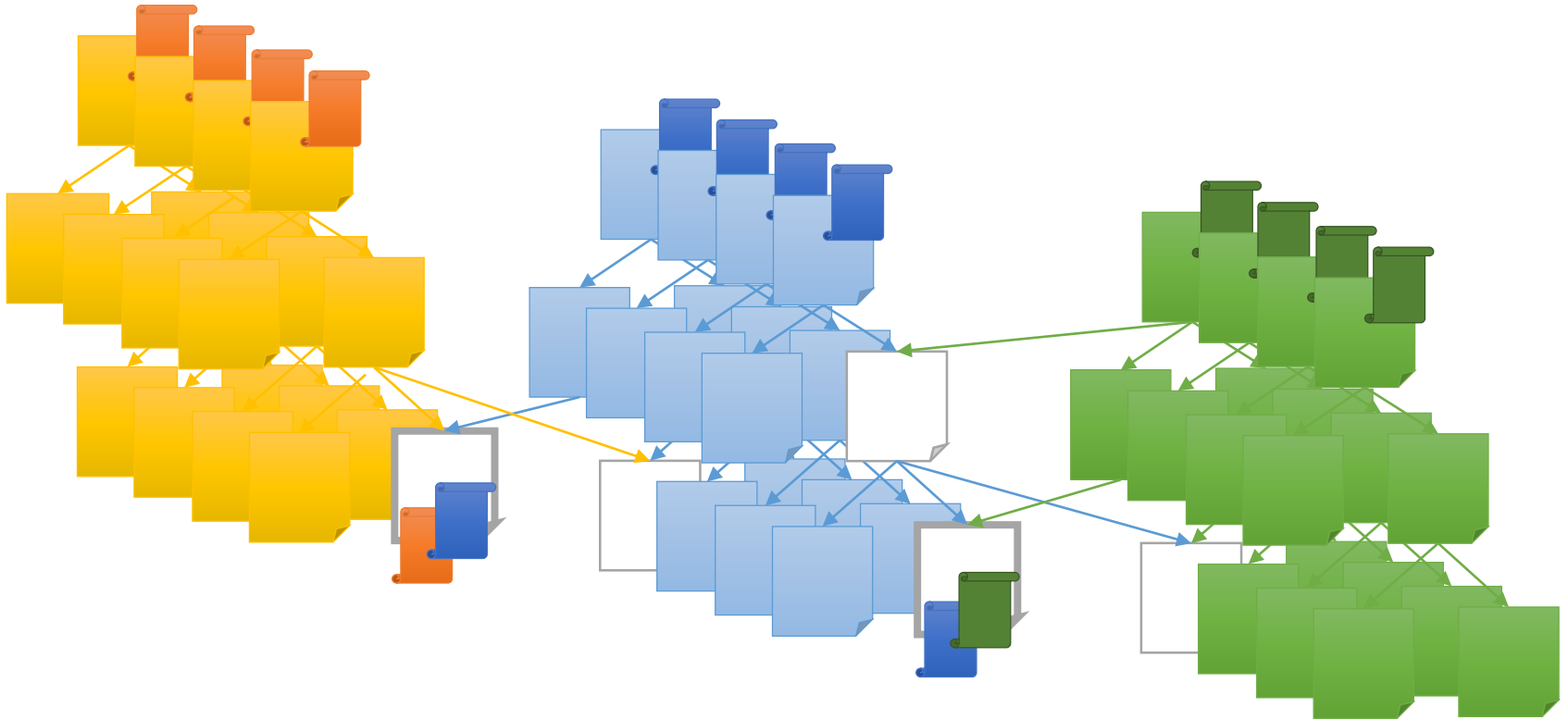


Automatic
deduction
of policies!

Scaling to many interactions...



Scaling to many interactions...

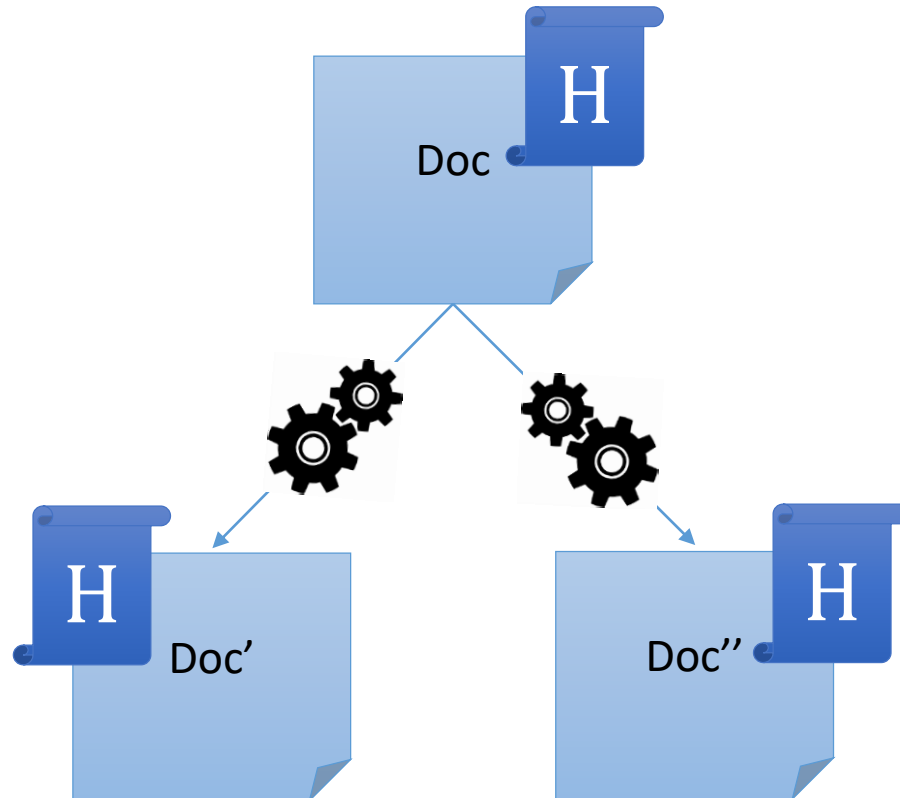


Labels to represent policies

Examples for confidentiality:





- Classifications
 - Unclassified (**U**), Confidential (**C**), Secret (**S**), Top Secret (**TS**)
 - Low confidentiality (**L**), High confidentiality (**H**)
- Sets of principals:
 - {Alice, Bob}, {Alice}, {Bob}, {}

Labels to represent policies



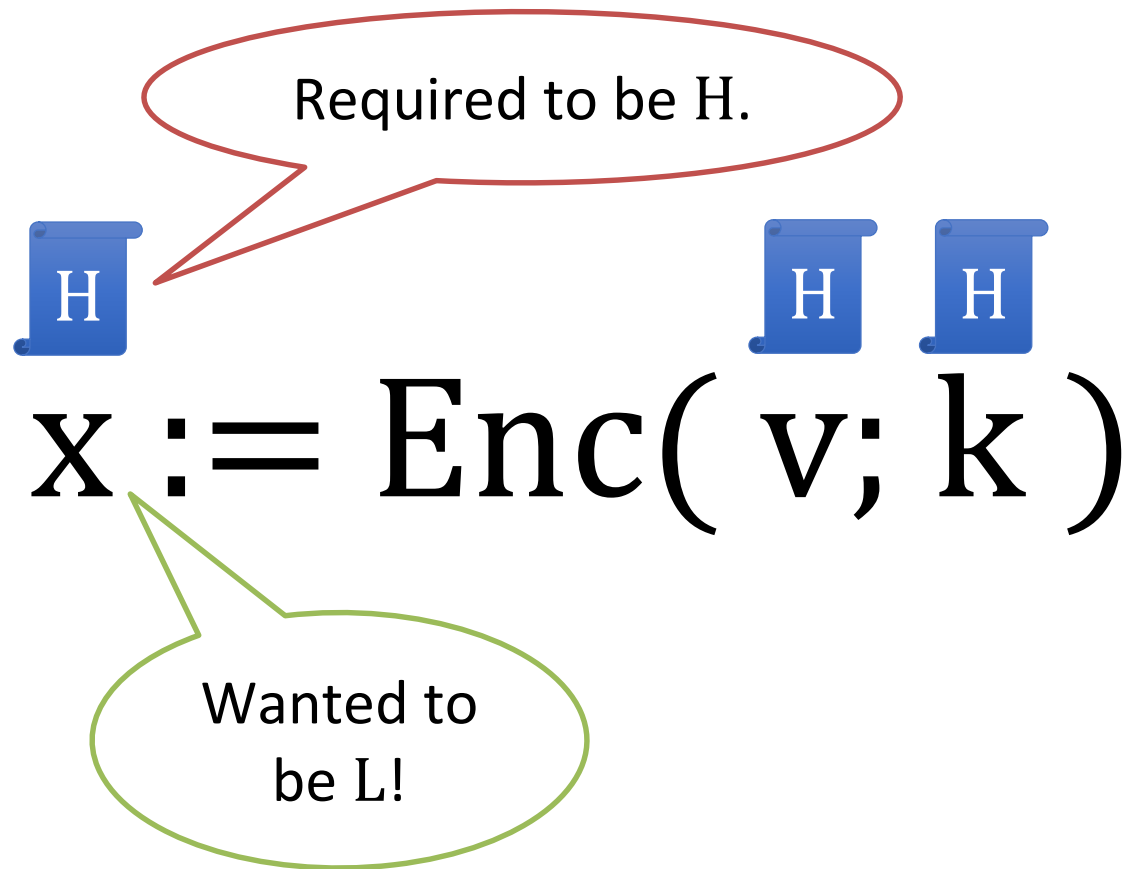
More restrictive than necessary...

Required to be H.

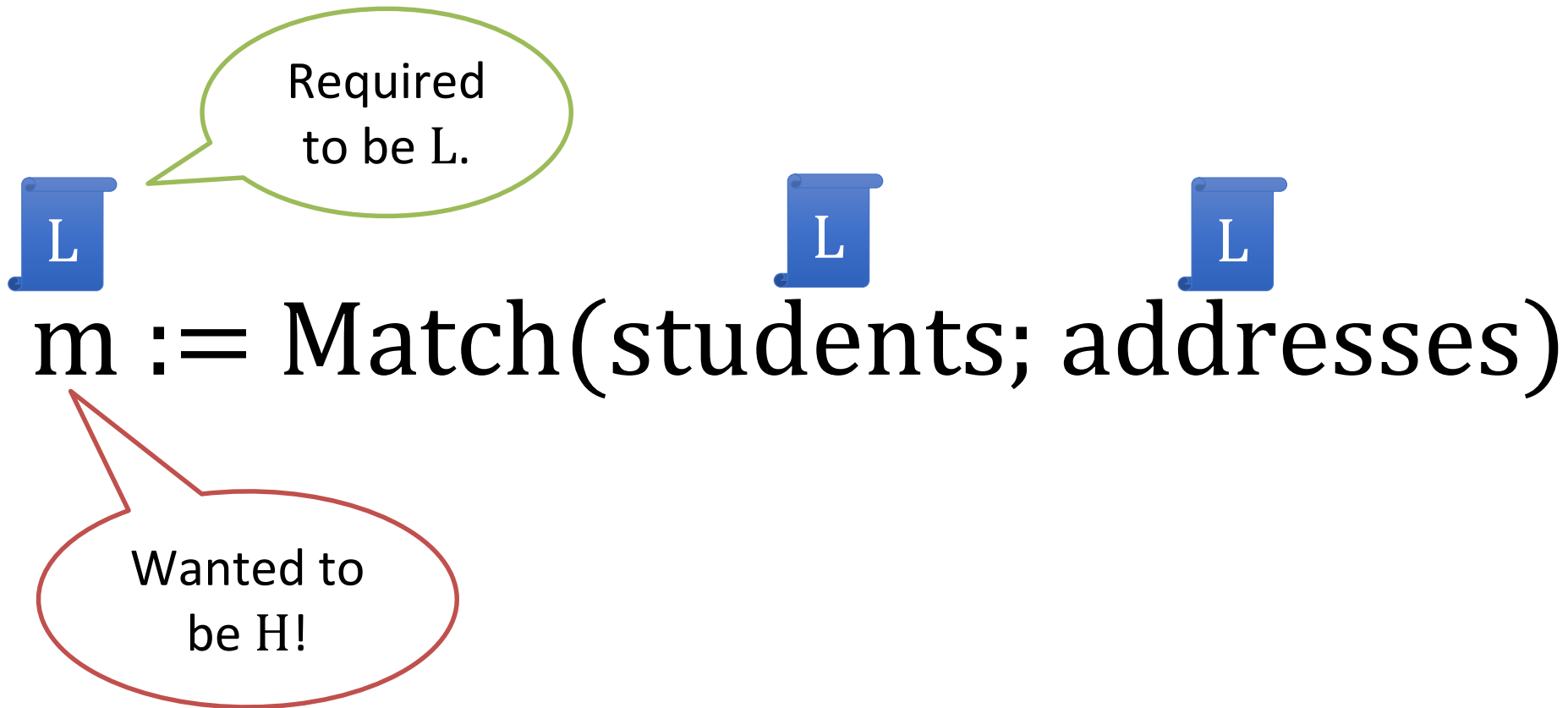
 $x := \text{maj}(\text{  v_1, \text{  v_2, \dots, \text{  } v_n)$

Wanted to
be L!

More restrictive than necessary...



Less restrictive than necessary...

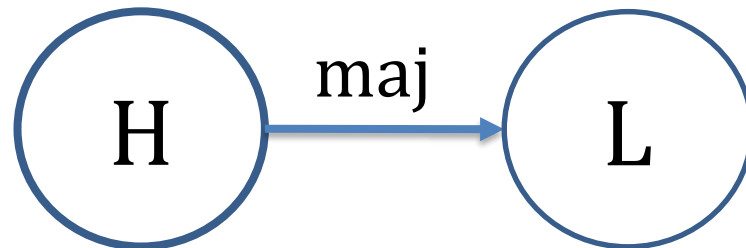


More expressive IF labels

Need to specify changes of restrictions based on:

- applied operations, or
- conditions on execution state, or
- ownership of values, or ...

For example, a vote v_i can be tagged with label:



Satisfaction of IF policies

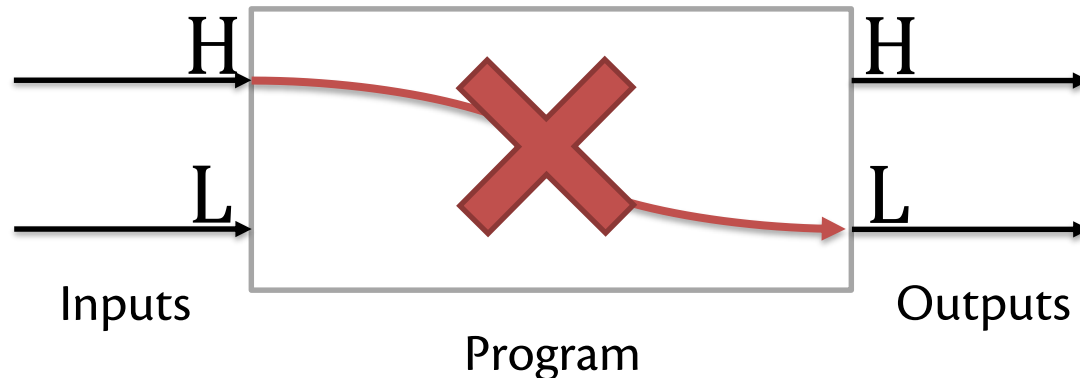
- Consider inputs and outputs of a program being tagged with label H or L.
- Inputs tagged with H are allowed to flow only to outputs tagged with H.
- Inputs tagged with H are *not allowed to flow* to outputs tagged with L.
- Changing input values tagged with H, should not cause changes on outputs tagged with L.
- This requirement is an instantiation of **noninterference**.
 - Inputs tagged with H should not interfere with outputs tagged with L.
- Noninterference is a semantic guarantee that should be offered by the enforcement mechanism of IF policies.
- Access control does not offer a similar semantic guarantee.

Noninterference

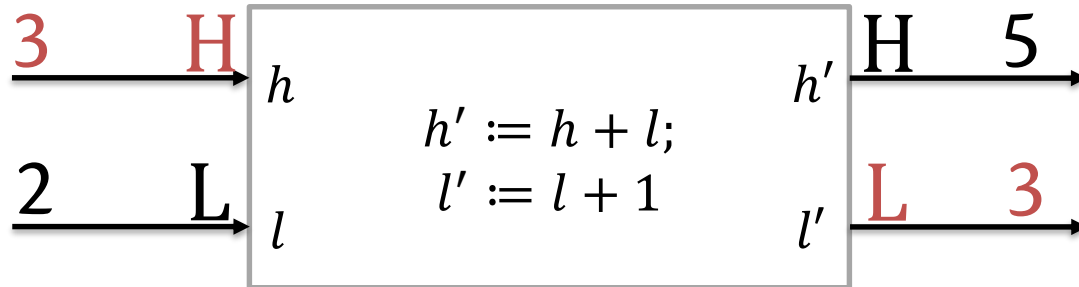
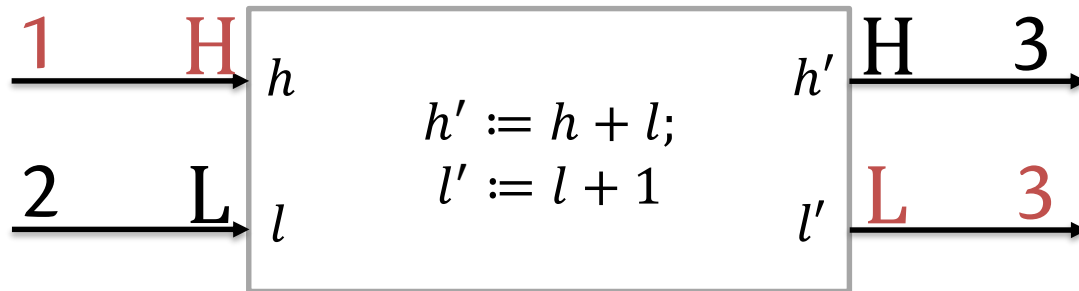
[Goguen and Meseguer 1982]

An interpretation of noninterference for a program:

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

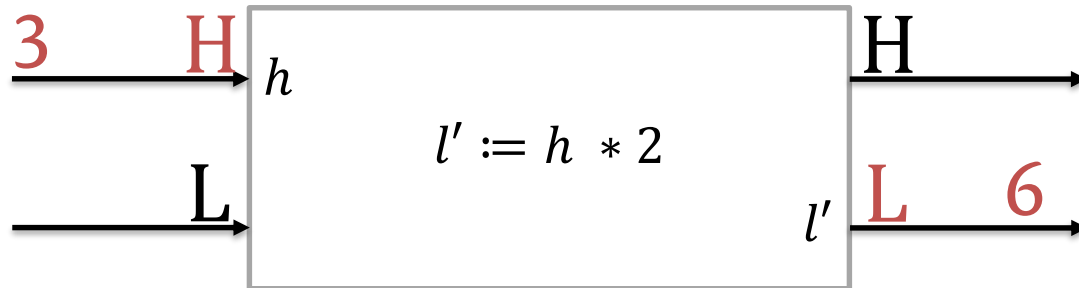
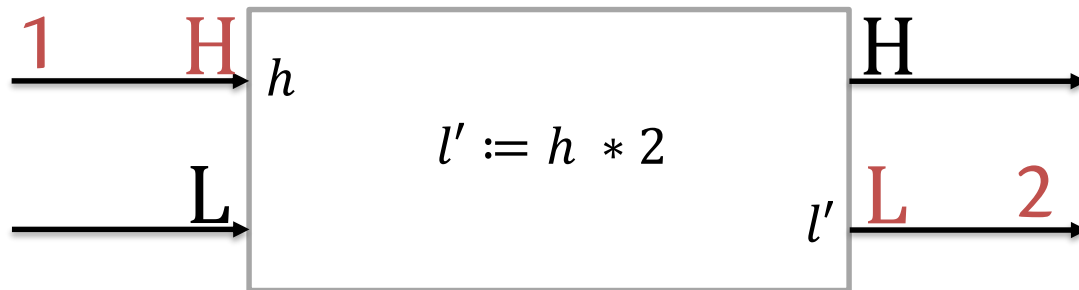


Noninterference: Example



The program satisfies noninterference!

Noninterference: Example



The program does not satisfy noninterference!

Noninterference

- Consider a program C .
- Consider two memories M_1 and M_2 , such that
 - they agree on values of variables tagged with L:
 - $M_1 =_L M_2$.



M_1 and M_2 may not agree on values of variables tagged with H.

Noninterference

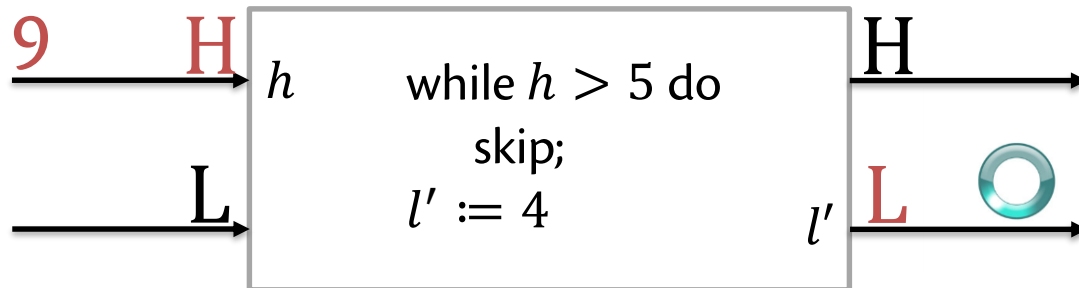
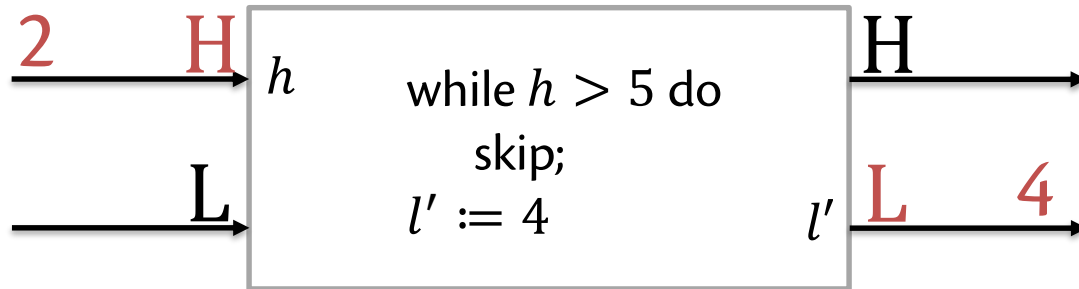
- Consider a program C .
- Consider two memories M_1 and M_2 , such that
 - they agree on values of variables tagged with L:
 - $M_1 =_L M_2$.
- $C(M_i)$ are the observations produced by executing C to termination on initial memory M_i :
 - final outputs, or
 - intermediate and final outputs.
- Then, observations tagged with L should be the same:
 - $C(M_1) =_L C(M_2)$.

Noninterference

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

$\forall M_1, M_2$: if $M_1 =_L M_2$, then $C(M_1) =_L C(M_2)$.

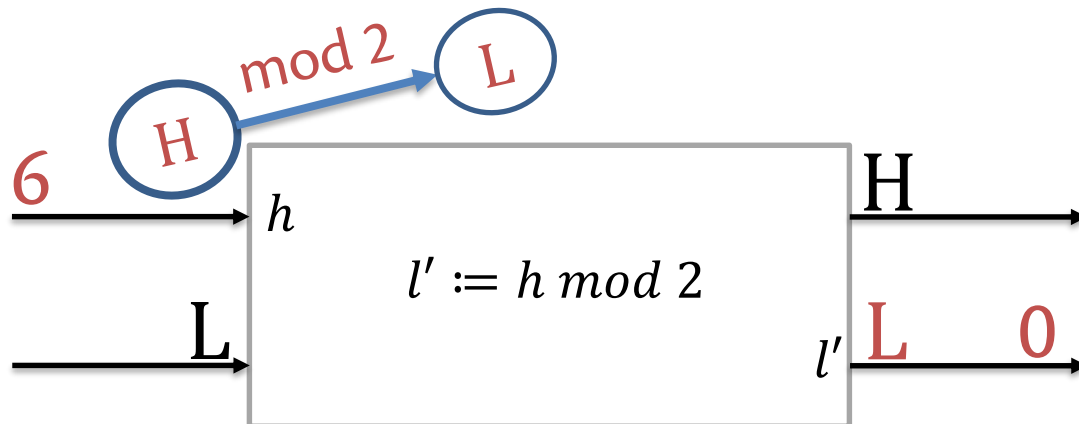
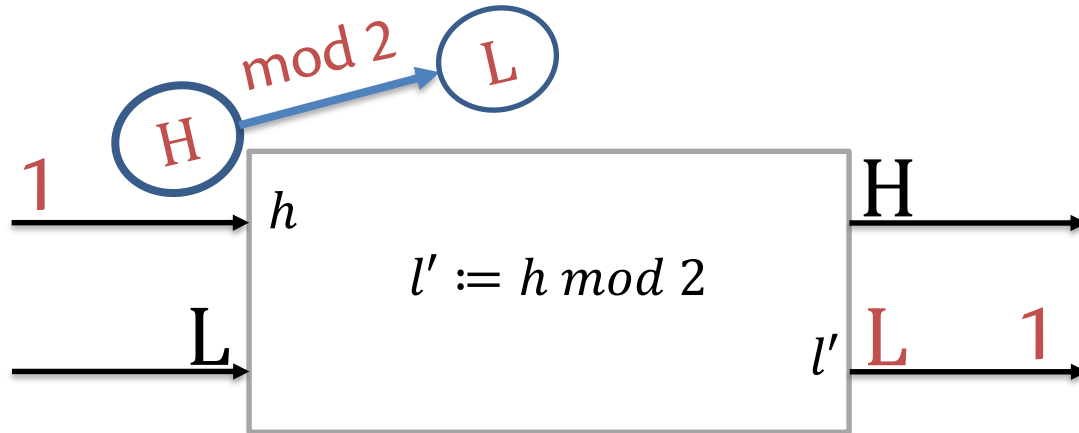
Termination sensitive noninterference



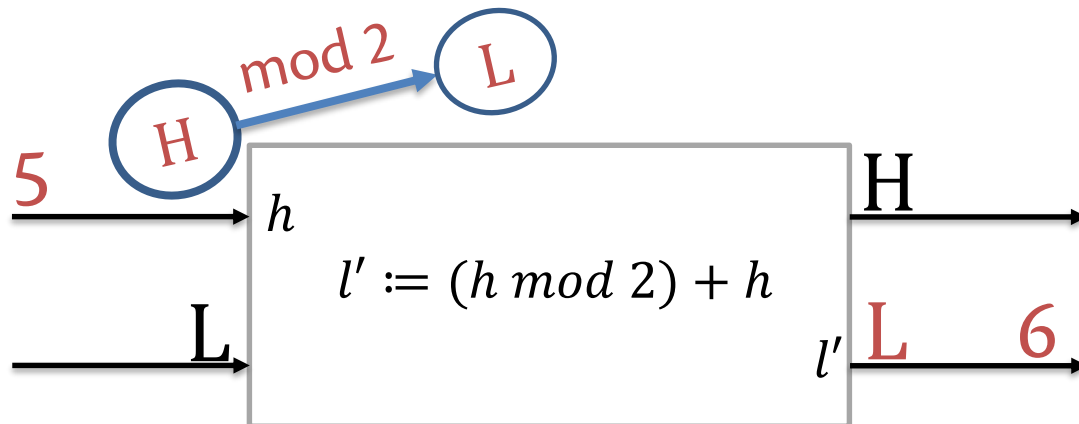
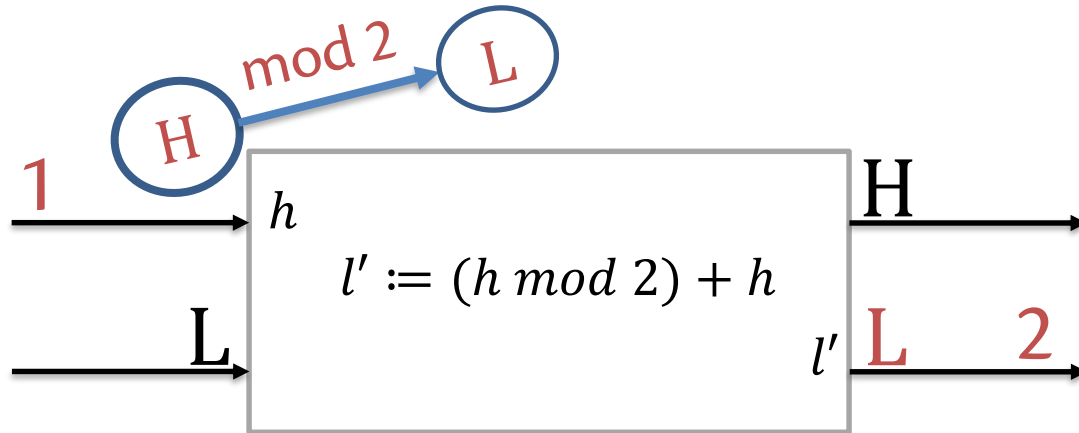
Termination sensitive noninterference

- If
 - $M_1 =_L M_2$,
- then
 - C terminates on M_1 iff C terminates on M_2 , and
 - $C(M_1) =_L C(M_2)$.

Noninterference (variation)



Noninterference (variation)



Noninterference for previous example

- If
 - $M_1 =_L M_2$, and
 - $M_1(h) \bmod 2 = M_2(h) \bmod 2$
- Then,
 - $C(M_1) =_L C(M_2)$.

More variants of noninterference

Prof. Clarkson is guilty too.

- [O'Neill, Clarkson, Chong 2006]: a variant of probabilistic noninterference
- [Micinski, Fetter-Degges, Jeon, Foster, Clarkson 2015]: noninterference for Android apps

Noninterference

- The more expressive the IF policies, the less appropriate noninterference becomes.
- Active research:
 - New semantic guarantees for expressive IF policies.

Upcoming events

- [Final exam] Please, read post on Piazza (@105) for important information.

*Don't let school interfere with your education.
– Mark Twain*