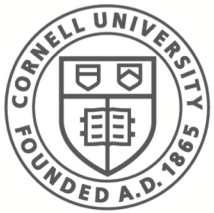
CS 5432:

Information Flow

Part III: Reactive Information Flow (RIF)

Fred B. Schneider
Samuel B Eckert Professor of Computer Science
(joint work with Elisavet Kozyri)

Department of Computer Science
Cornell University
Ithaca, New York 14853
U.S.A.



Cornell CIS
Computer Science

Label Creep with FBAC

Flow-Label Invariant (FLI):

$$v \rightarrow w \implies \Gamma(v) \sqsubseteq \Gamma(w)$$

Problematic examples:

wi := maj(v1, v2, …, vn)

v1 : voter1; v2 : voter2; … w : public

ctext := Enc(ptext, key)

ptext : secret; key : secret; ctext : public

Reactive Information Flow (RIF) Tags

RIF Tag: Seq of reclassifiers → restrictions

Defined in terms of (Λ, ℜ, ℑ).

- Λ is a set of labels
- ℜ is a restriction function ℜ: Λ → restrictions
  - Assume ordering relation ≤ on restrictions.
- ℑ is a transition function ℑ: Λ x F* → Λ

F* is a finite sequence of **reclassifiers**, which abstract the operations available for deriving values.

$(\Lambda, \mathcal{R}, \mathcal{T})$

- Values and variables are tagged with labels $\rho \in \Lambda$.
- $\mathcal{R}(\rho)$ are the restrictions on a value with label ρ.
- When value is transformed by operations, its label is transformed, too. Require that:
  - $\mathcal{T}(\rho , \epsilon) = \rho$
  - $\mathcal{T}(\mathcal{T}(\rho , F1) , F2)) = \mathcal{T}(\rho , F1\ F2)$

Specifying Reclassification

v : ρ

- v subject to restrictions: $\mathcal{R}(\rho)$
- $[\ v\ ]_F$ has tag $\mathcal{T}(\rho\ ,\ F)$
- $[\ v\ ]_F$ subject to restrictions: $\mathcal{R}(\mathcal{T}(\rho\ ,\ F))$

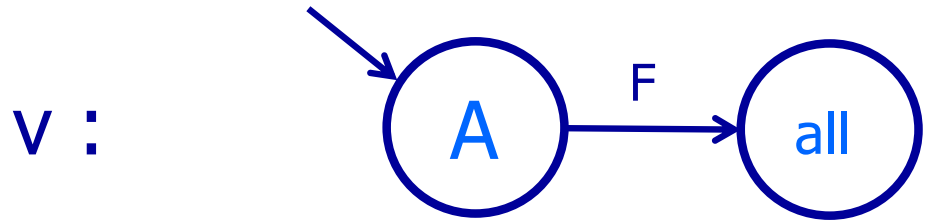
$\sqsubseteq$ for RIF Labels

$\rho \sqsubseteq \rho'$ defined to be:

$$(\forall F^*: \quad \mathcal{R}(\mathcal{T}(\rho\ ,\ F^*)) \leq \mathcal{R}(\mathcal{T}(\rho'\ ,\ F^*)))$$

- ρ' imposes at least the restrictions ρ does
- any value derived (via F*) from ρ' does, too

RIF Tag Example: RIF Automata

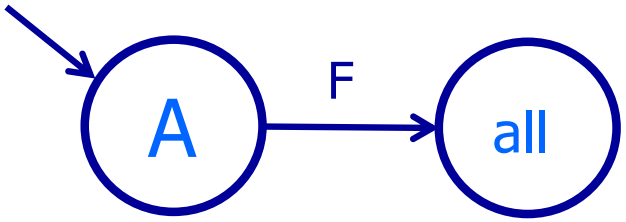


Automaton states give sets of restrictions.
Edge labels abstract classes of operations.
Implicit self-loop for unspecified operations

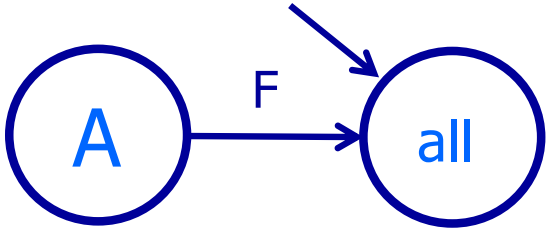
Transitions

$[\ expr\ ]_F$ Operation *expr* treated as an "F"

expr :



$[\ expr\ ]_F$:



Definitions for $\sqcup$ and $\sqsubseteq$

Every automaton M accepts a language L(M)

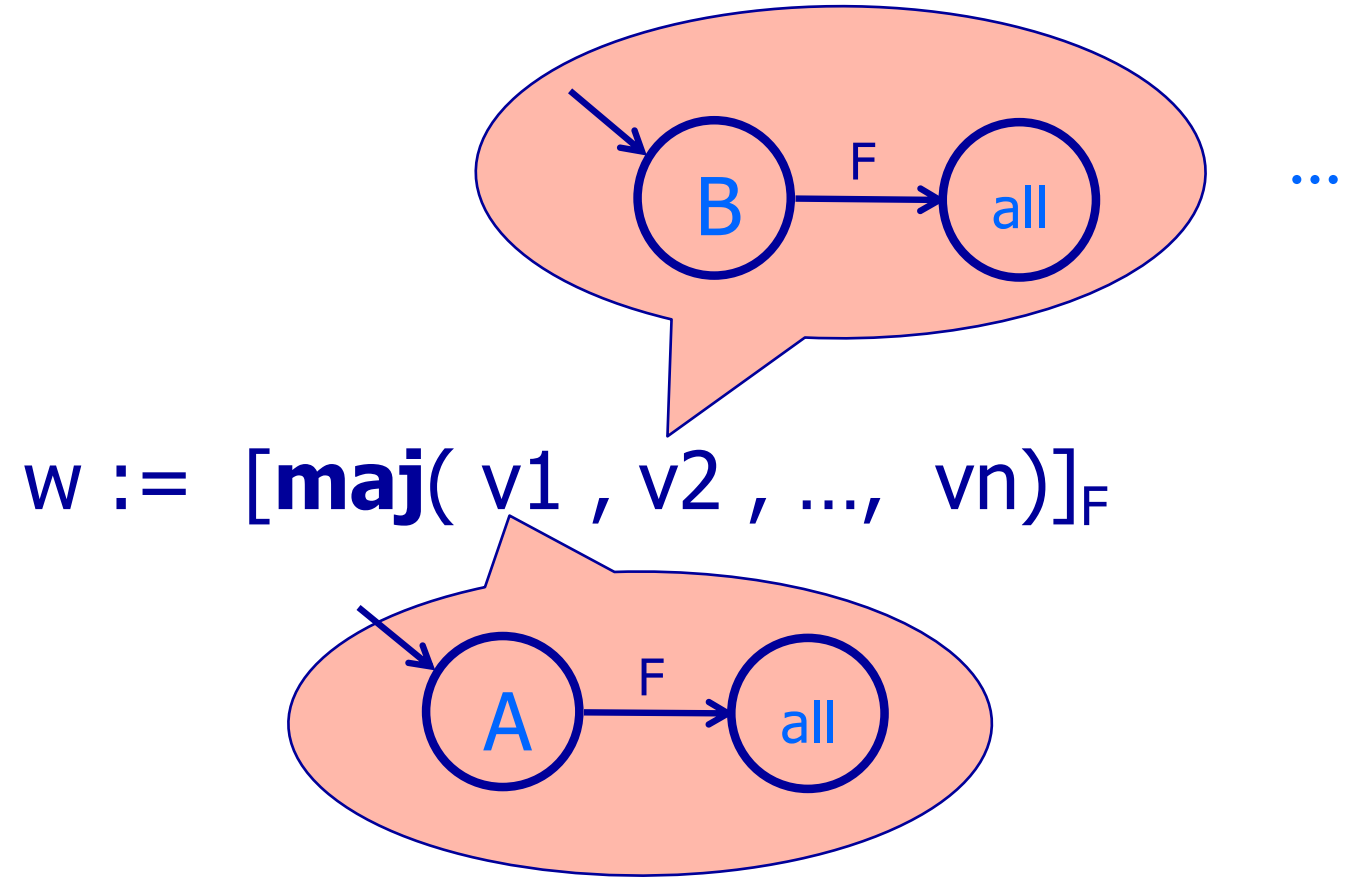
Each word in L(M)

- is a sequence F1 F2 … Fn of reclassifiers
- is associated with a set of restrictions

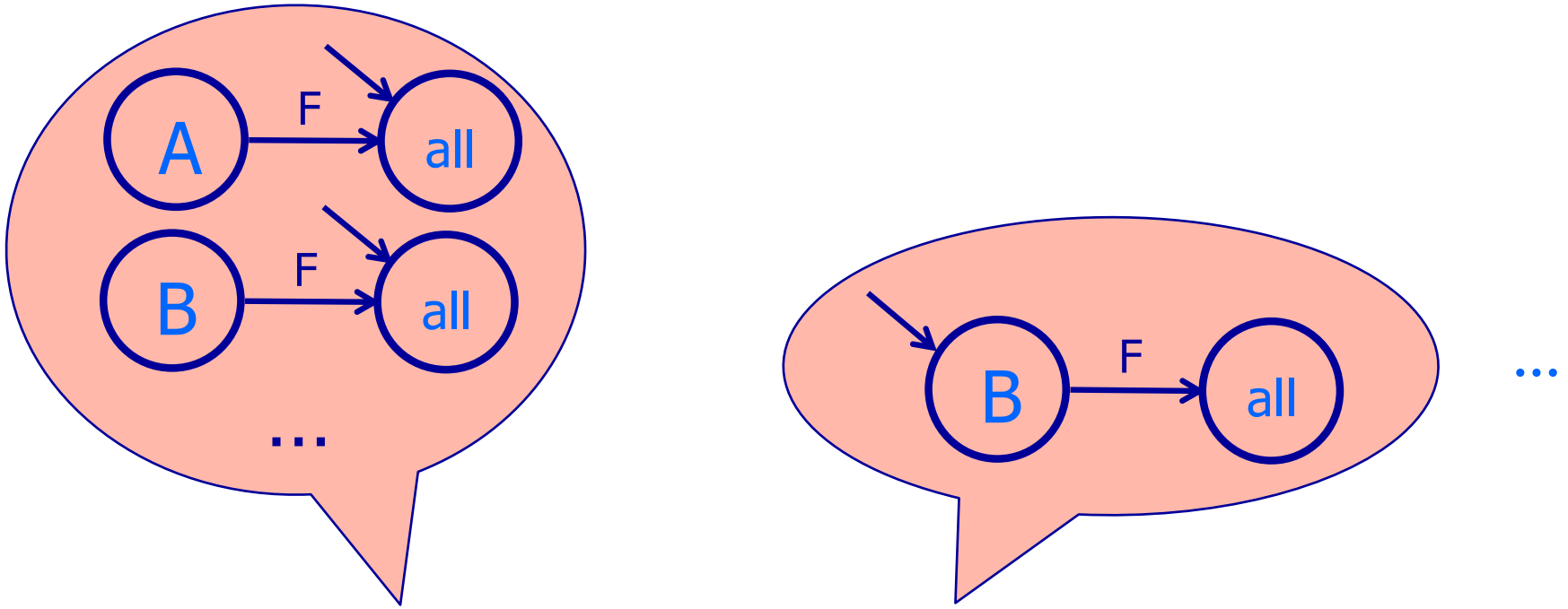
- M $\sqcup$ M′ is product automaton M $\times$ M′
- M $\sqsubseteq$ M′ is:

$$(\forall F^*: \quad \mathcal{R}(\mathcal{T}(M, F^*)) \leq \mathcal{R}(\mathcal{T}(M', F^*)))$$

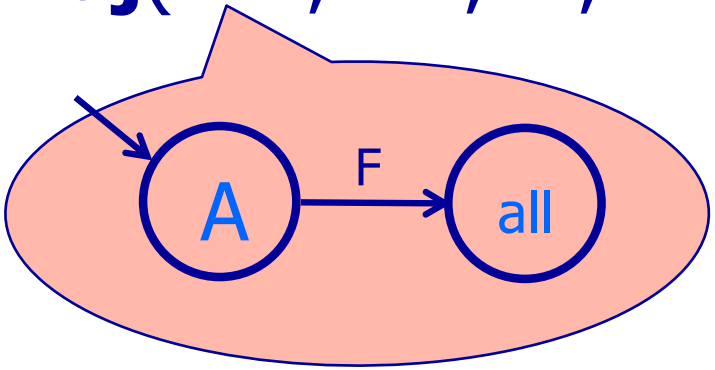
Example: Majority Voting



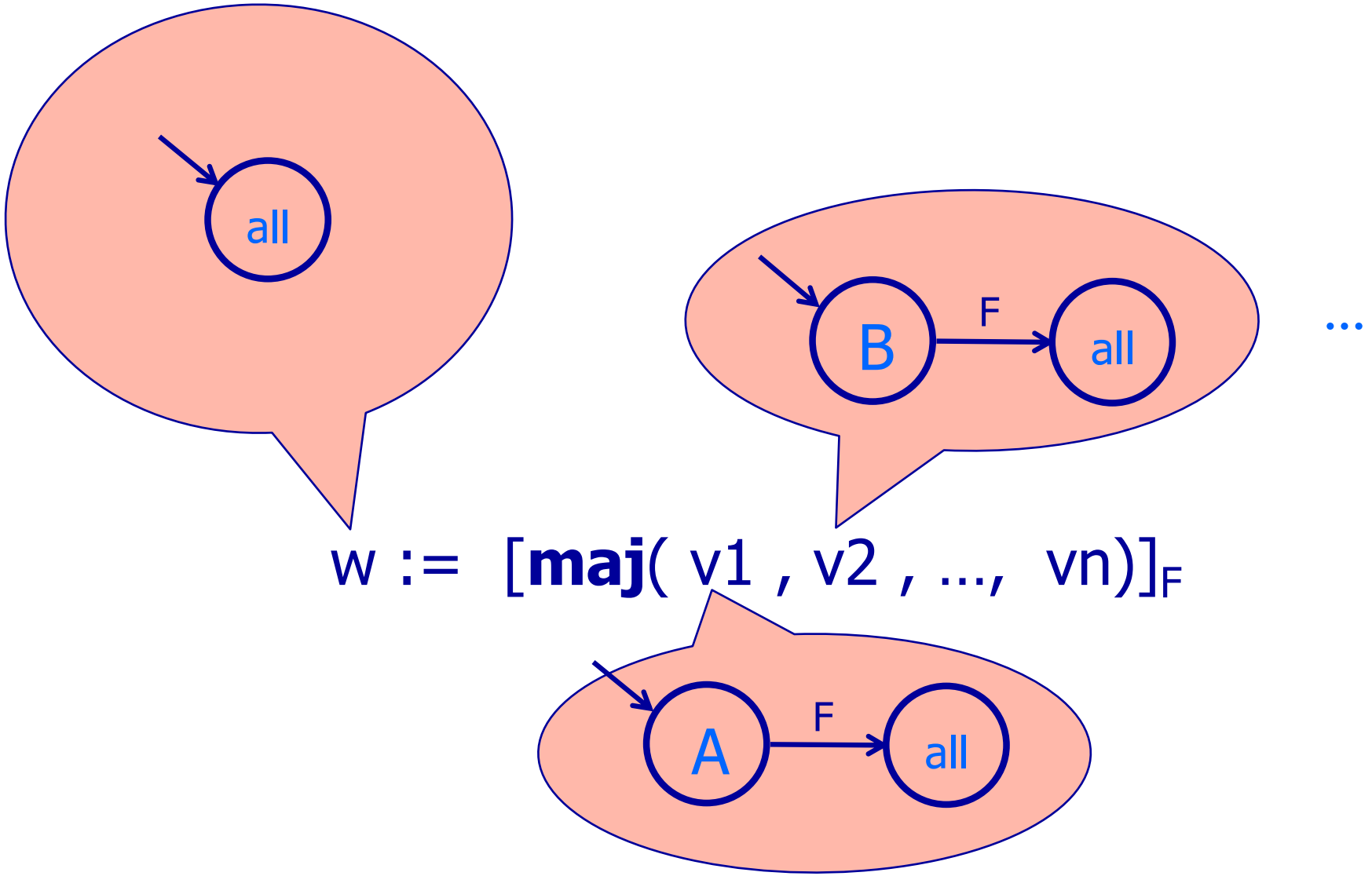
Example: Majority Voting



$w := [\mathbf{maj}(\ v1\ ,\ v2\ ,\ \ldots,\ \ vn)]_F$



Example: Majority Voting



RIF Tags for Crypto Operations

Concern: Confidentiality / symmetric crypto

m : secret k : secret

- c := Enc(m, k) c can be public
- d := Dec(c, k') d can be public if k ≠ k'
- e := Dec(c, k') e should be secret if k= k'

… finite state automata are not sufficiently expressive.

… stack automata are not decidable.

Crypto Tag Anatomy

c := encrypt(m,k) → c:= $[\text{encrypt}(m,k)]_{ENC(k)}$

v′ : { …., ⟨ F*, KN(v) ⟩ ….}

⟨ F*, KN(v) ⟩ defines restrictions on flow from v′

- KN(v) is set of principals allowed to read v
- F* is a canonical sequence of operations involving
  - ENC(k), DEC(k) for all keys k

Canonical sequence: Adjacent symbols aren't inverses.

… F ENC(k) DEC(k) F′ … → … F F′ …

Crypto Tags Restrict Reading

Assume set of keys known to P is given.

v′ : { … ⟨ F*, KN(v) ⟩ …}

⟨ F*, KN(v) ⟩ means v′ can be read by P if:

- P ∈ KN(v) -or-
- Keys known to P insufficient to reduce F* to empty sequence.

Crypto Tags Details

- $T \sqcup T'$ is union
- $T \sqsubseteq T'$ holds if readers under T' are also readers under T.
- Canonical sequence depends on what crypto operations are available.
  - Assume: equational theory that provides reductions.
  - Can handle shared key, public key, homomorphic, 1-time pad, …

What Policy Do RIF Tags Enforce?

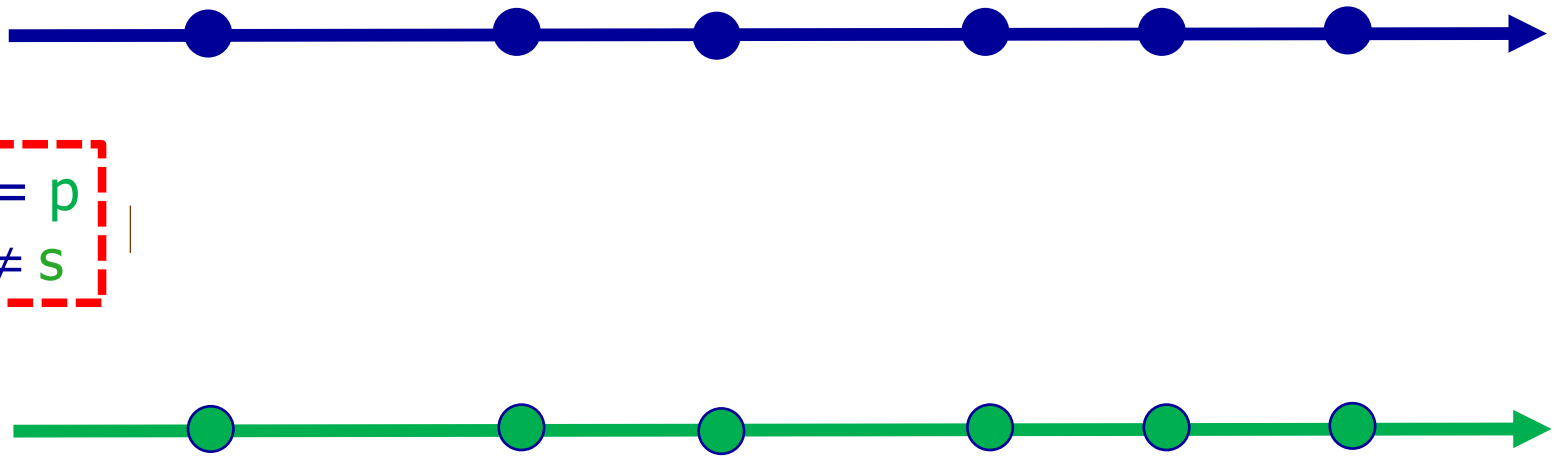
Threat model: Attacker with clearance L sees each update to a program variable v where:

$$Tv \sqsubseteq L$$

Goal: Enforce flow-based restrictions associated with all values.

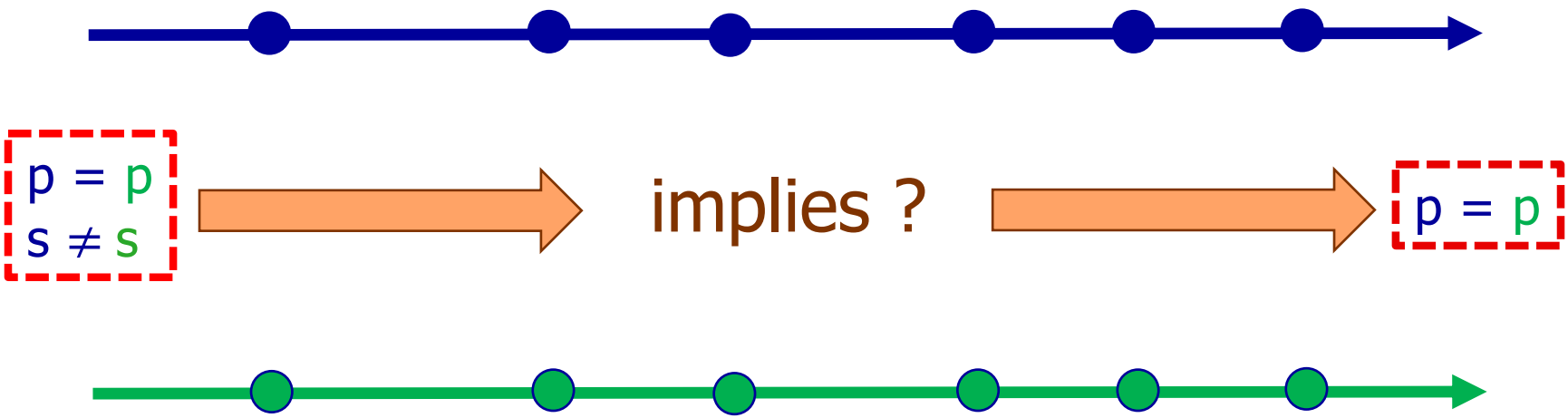
E.g., **if** s>0 **then** p:=1 **else** p:= 2 **fi**

Evidence of a Leak [Goguen + Meseguer]



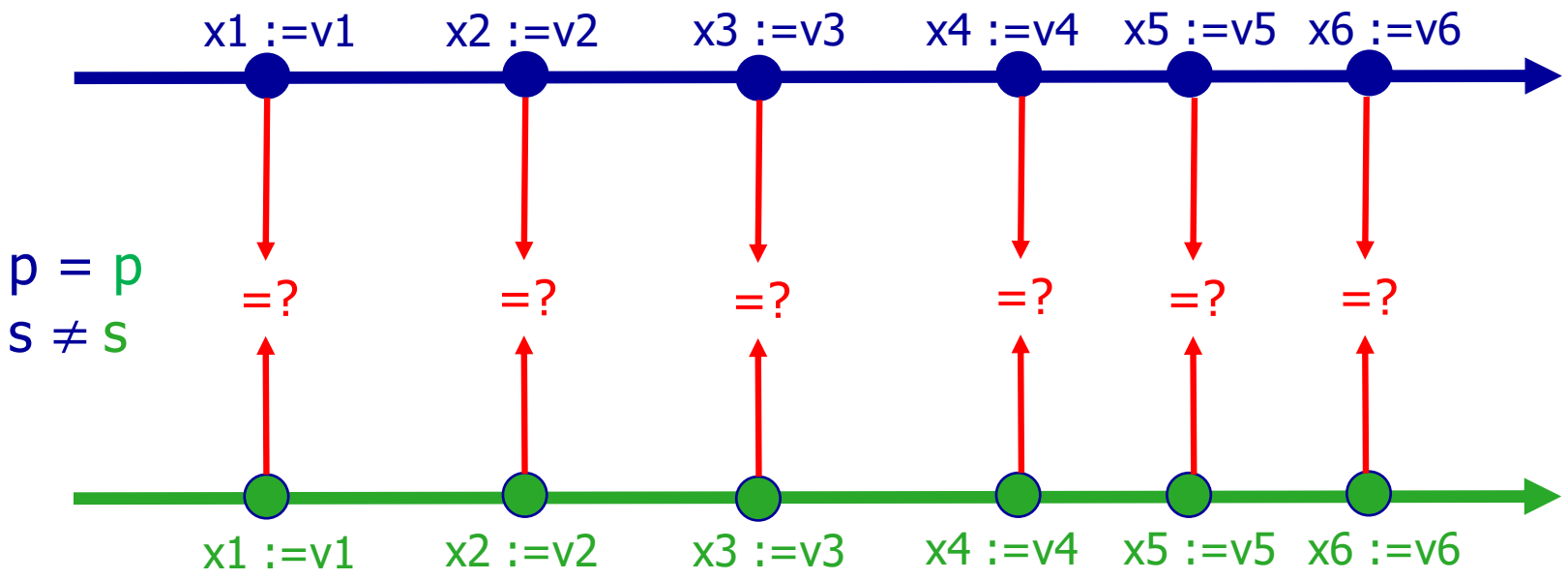
starting states agree on public values.

Evidence of a Leak [Goguen + Meseguer]



Final states do not agree on public values when starting states agree on public values.

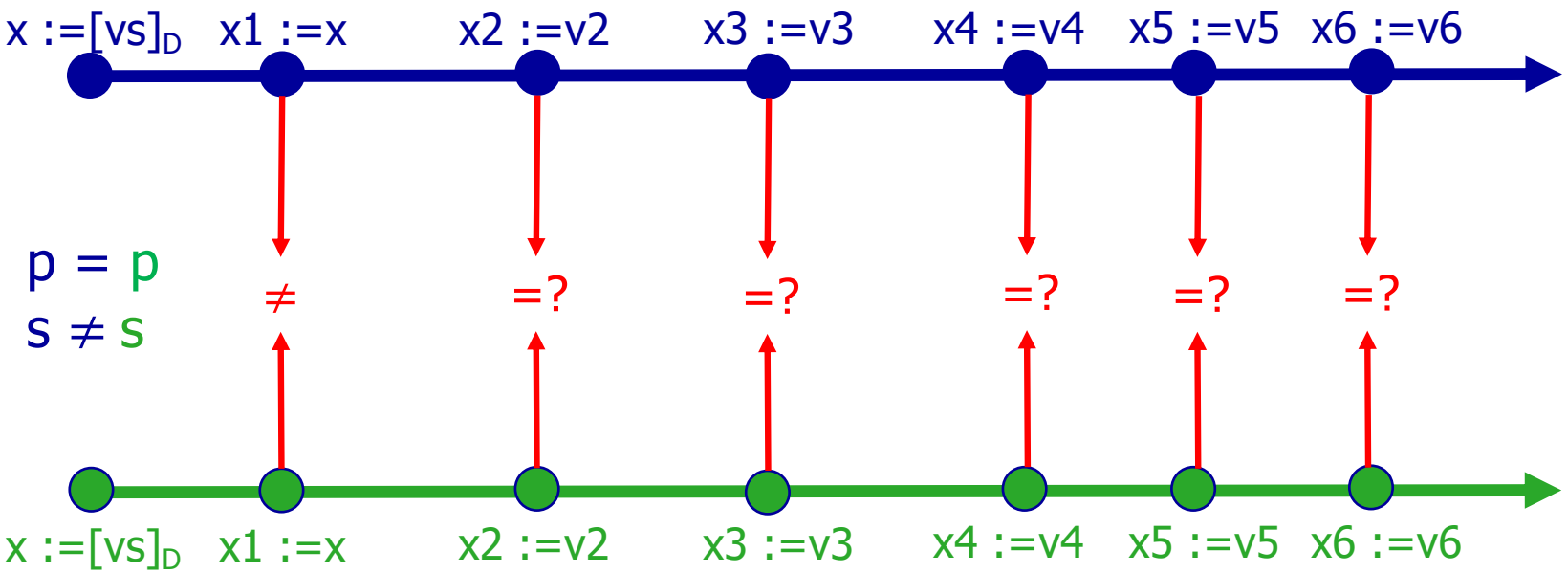
Evidence of a Leak (for updates)



Public updates do not agree on values when starting states agree on public values.

Evidence of a leak:

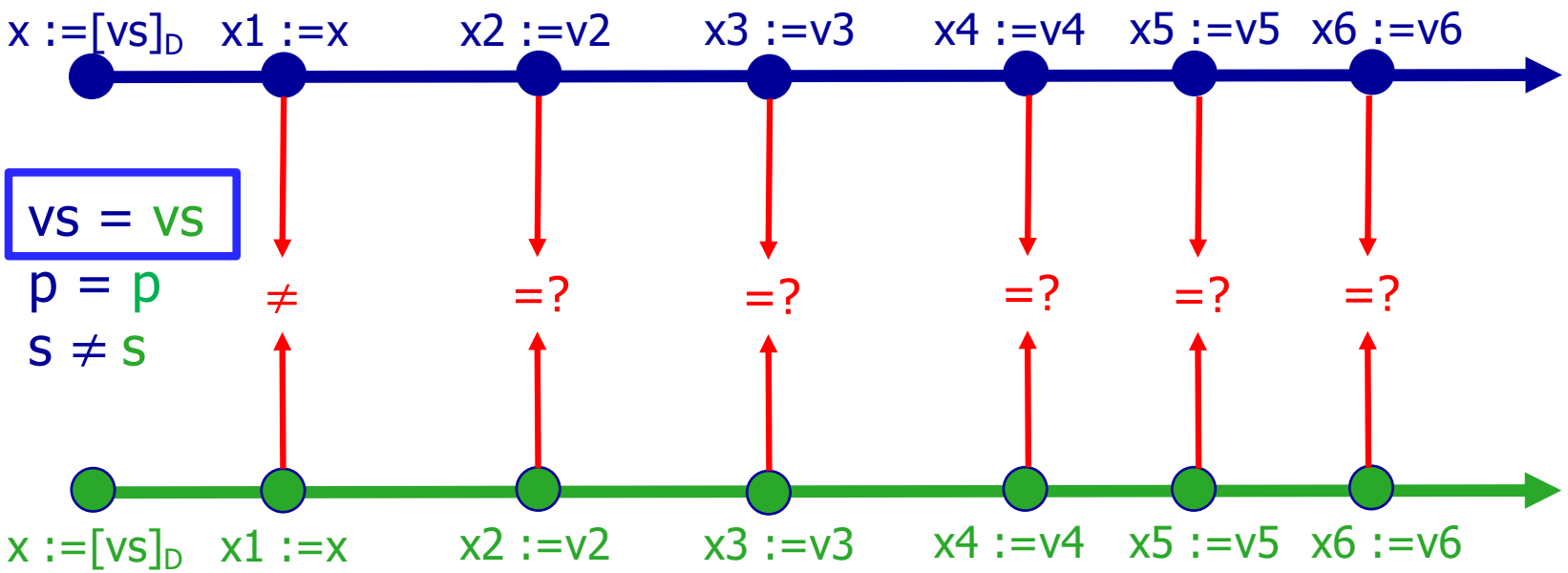
Handling Initial Declassification



Public updates do not agree on values when starting states agree on public values.

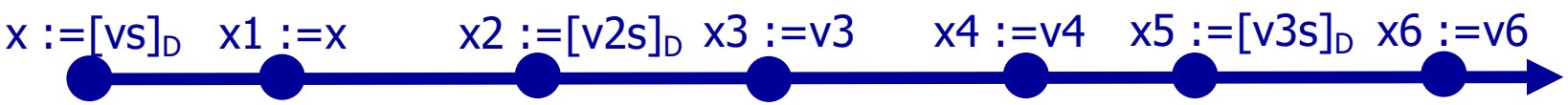
Evidence of a leak:

Handling Initial Declassifcation

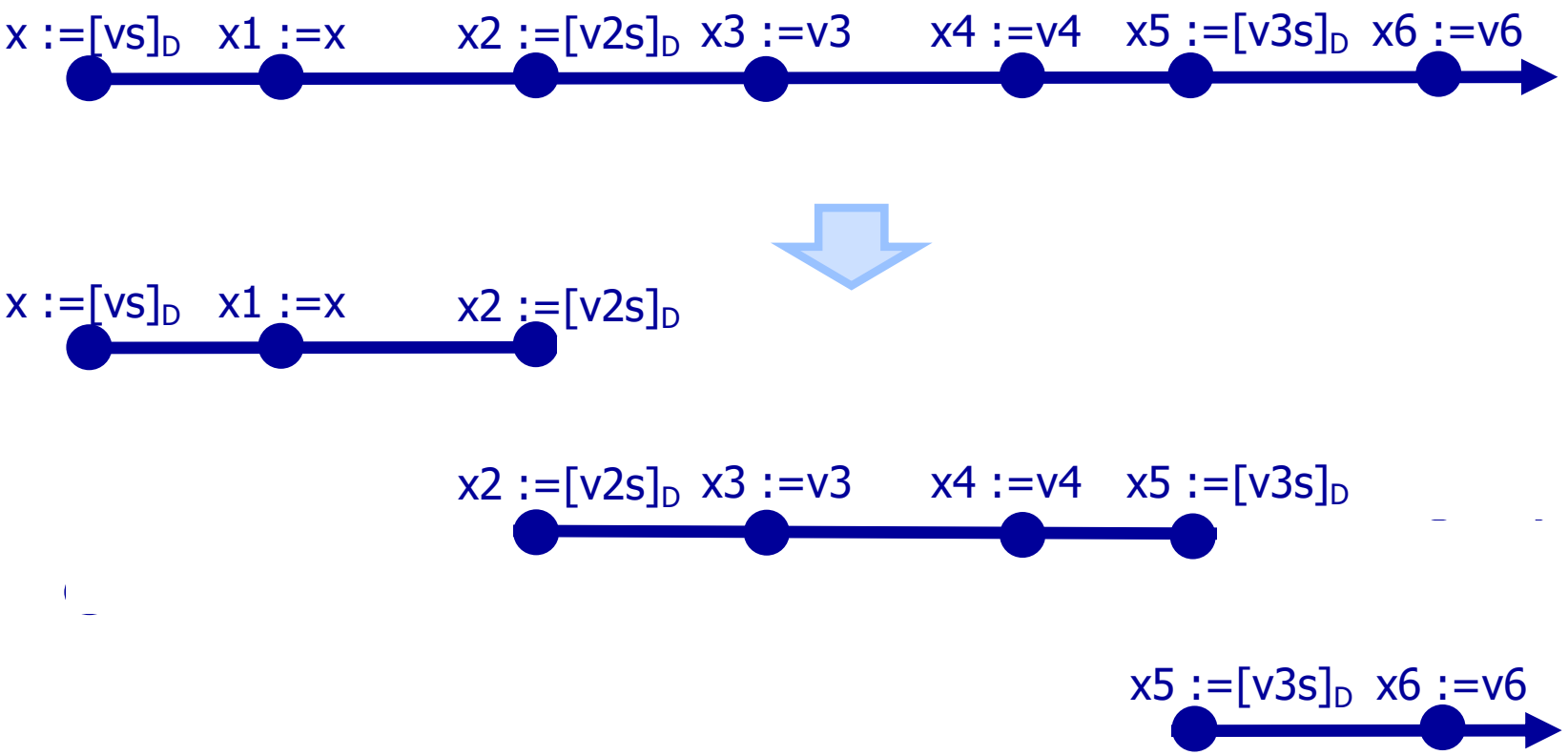


Public updates do not agree on values when starting states agree on public values **and on values being declassified.**

PWNI for Internal Declassifications

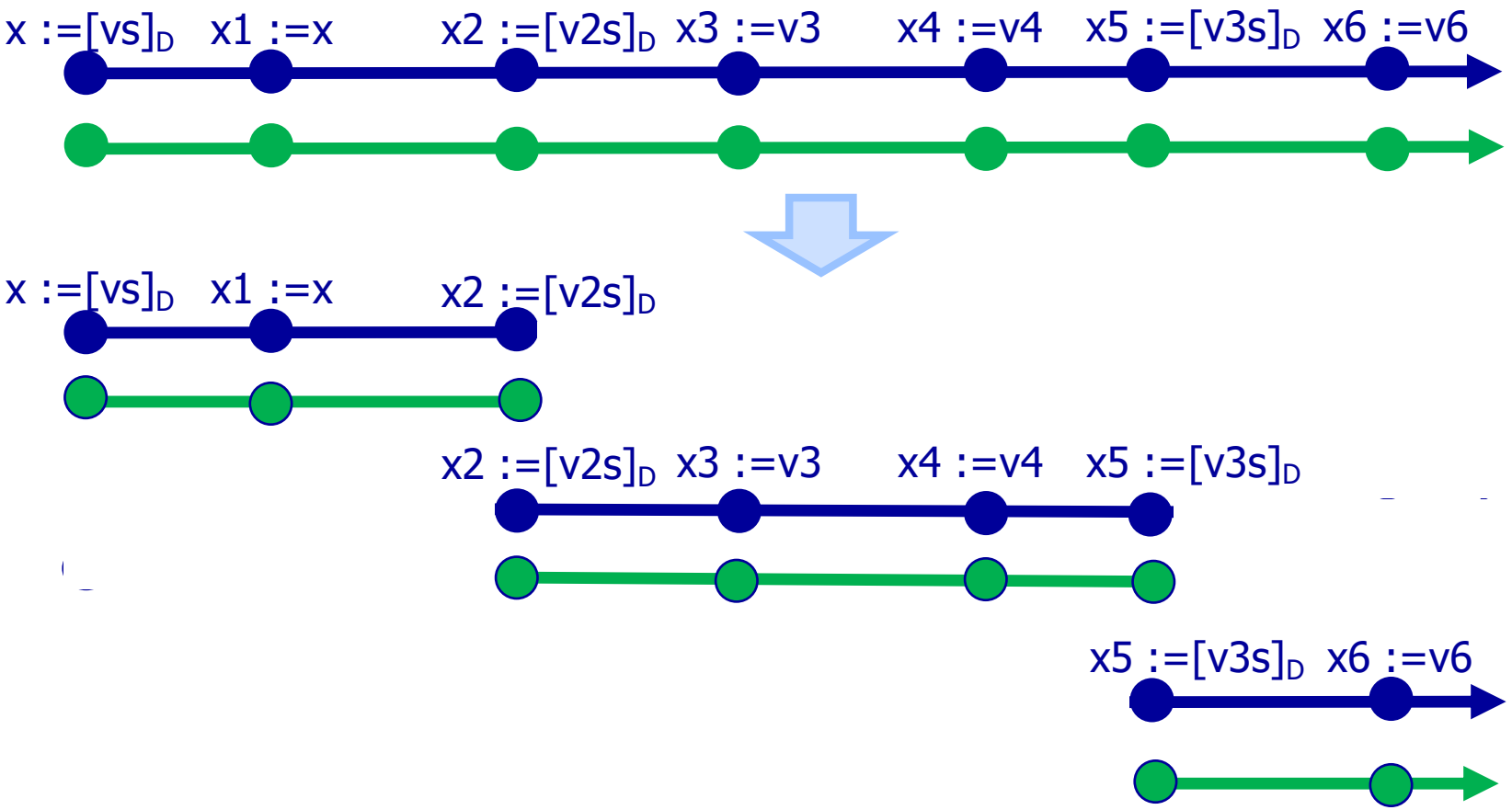


PWNI for Internal Declassifications



Evidence of a leak:

PWNI for Internal Declassifications



For each piece: Public updates do not agree on values if starting states agree on public values and on values being declassified.

PWNI Check: Details

No leak exists provided:

For initial pieces p, p′ in every pair of executions "blue" / "green": Check pwni(p,p′), comprising

- **if** p and p′ agree on initial command, public values, and declassified values
- **then**
  - p and p′ agree on final command, updates to public values, and
  - pieces p″ and p″ that are successors to p and p′ satisfy pwni(p″, p″)

Declassification Example

$low := [high]_D$

$low' := [high']_D$

$low := high$

Declassification Example

low := $[high]_D$

low′ := $[high']_D$

low := high

Declassification Example

{ high=high high′≠high′ }

✓ low := $[high]_D$

low′ := $[high']_D$

low := high

Declassification Example

{ high=high high′≠high′ }

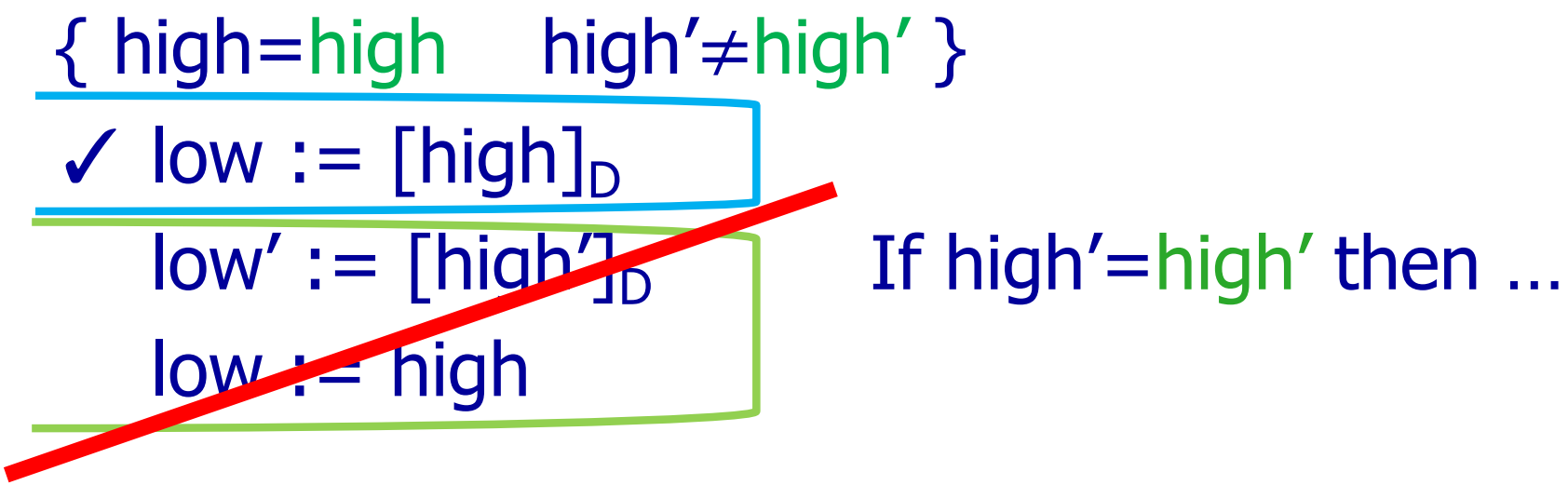
✓ low := $[high]_D$

low′ := $[high']_D$

low := high

If high′ ≠ high′ then …

Declassification Example



Declassification Example

{ high=high high′=high′ }

✓ low := $[high]_D$

low′ := $[high']_D$

low := high

Declassification Example

{ high=high high′=high′ }

✓ low := $[high]_D$

✓ low′ := $[high']_D$

✓ low := high

If high′=high′ then …

Declassification Example (2)

low := $[high]_D$

high := high’

low := high

Declassification Example

low := $[high]_D$

high := high’

low := high

Declassification Example

{ high=high high′≠high′ }

✓ low := $[high]_D$

high := high′

low := high

Declassification Example

{ high=high high′≠high′ }

✓ low := $[high]_D$

✓ high := high′

low := high

Declassification Example

{ high=high high′≠high′ }

✓ low := $[high]_D$

✓ high := high′

✗ low := high

Handling Upgrades

low := $[low']_U$

- satisfies PWNI
- differences in H values lead to differences in L values!

Conclude: PWNI does not handle upgrades.

$$[low']_U \not\sqsubseteq low'$$

Solution: Replace each upgraded expression (e.g. $[low']_U$) with reference to fresh sequence of values to provide values of each high variable.

Summary

- RIF labels specify
  - restrictions on values
  - RIF labels for derived values

… suffices for reclassification

- PWNI (piecewise non-interference)
  - candidate security condition for RIF labels