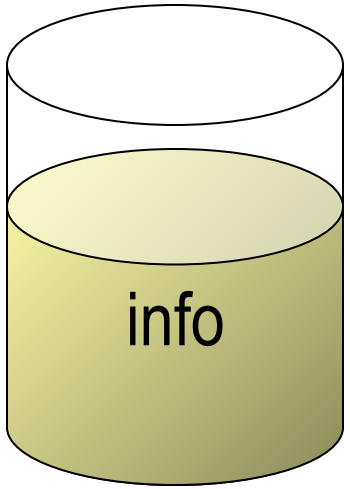
Course outline

1. Language-Based Security: motivation
2. Language-Based Information-Flow Security: the big picture
3. Dimensions and principles of declassification
4. Dynamic vs. static security enforcement
5. Tracking information flow in web applications
6. Information-flow challenge

Dimensions of Declassification in Theory and Practice

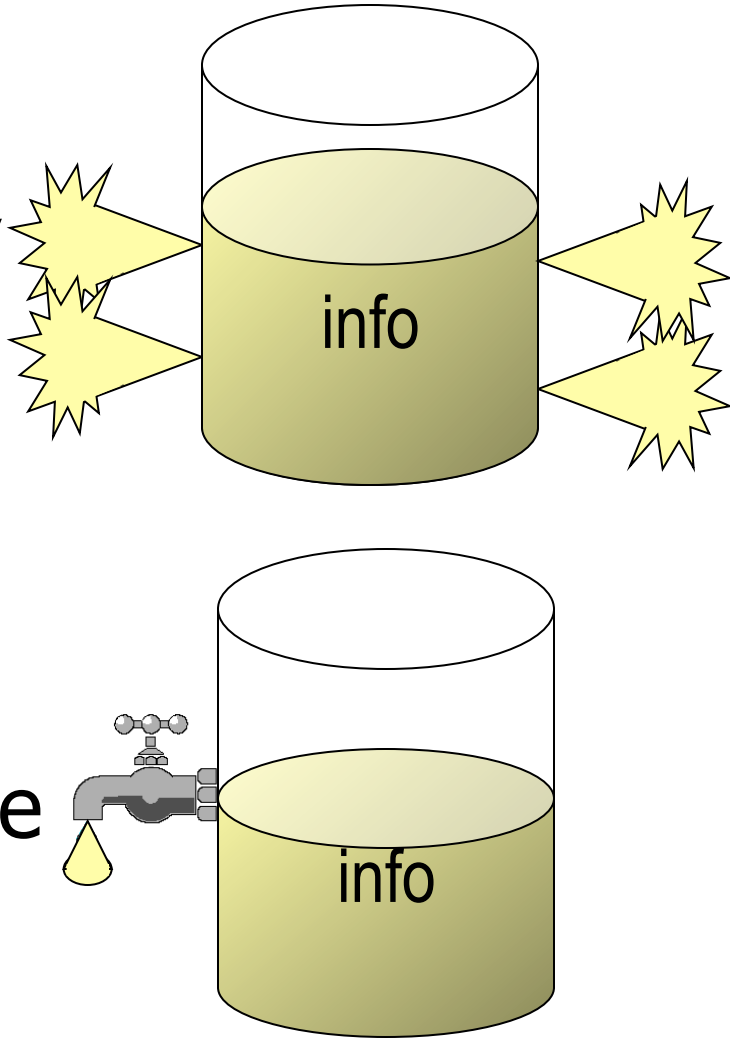
Confidentiality: preventing information leaks

- Untrusted/buggy code should not leak sensitive information
- But some applications depend on intended information leaks
  - password checking
  - information purchase
  - spreadsheet computation
  - …
- Some leaks must be allowed: need information release (or declassification)

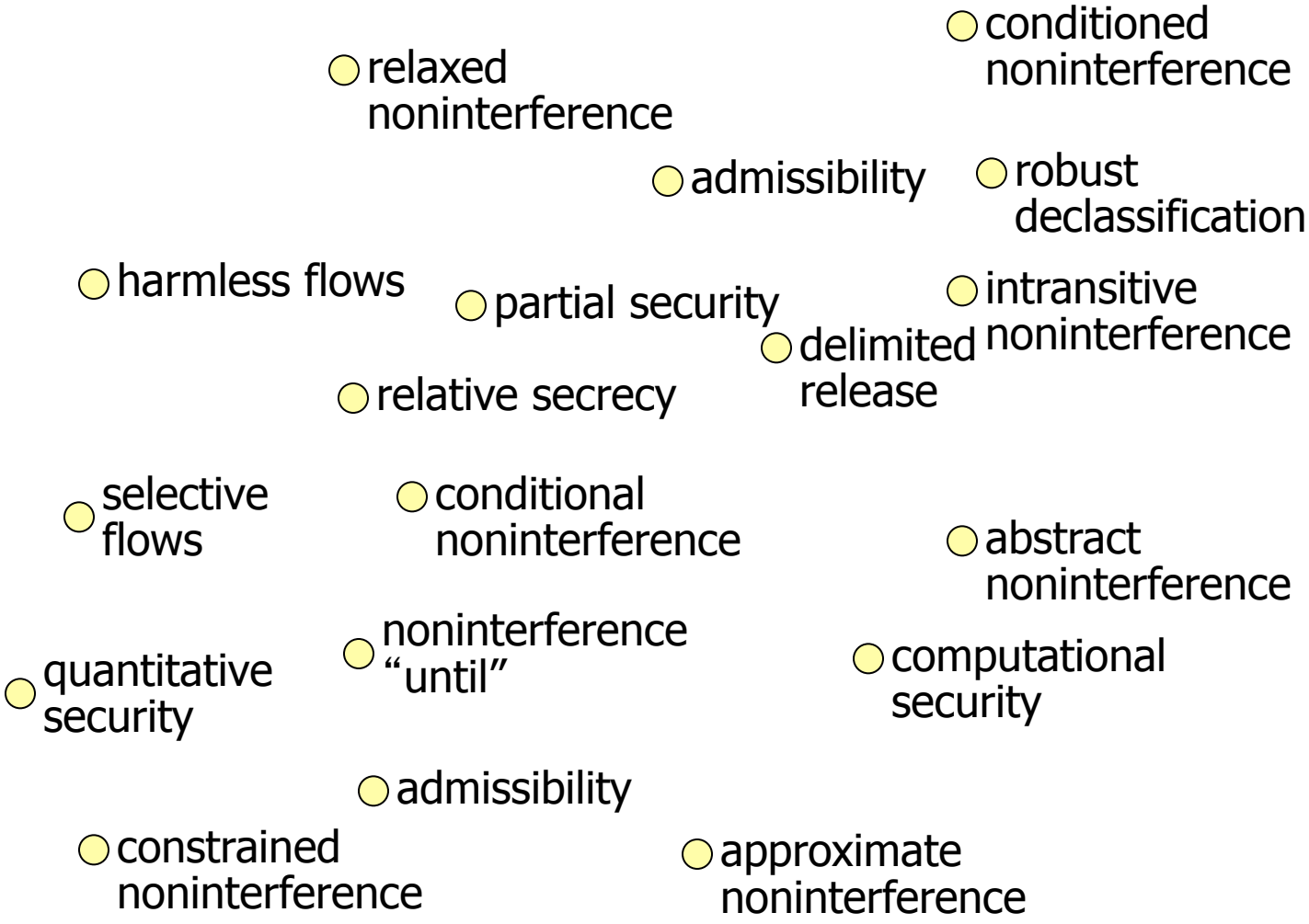


Confidentiality vs. intended leaks

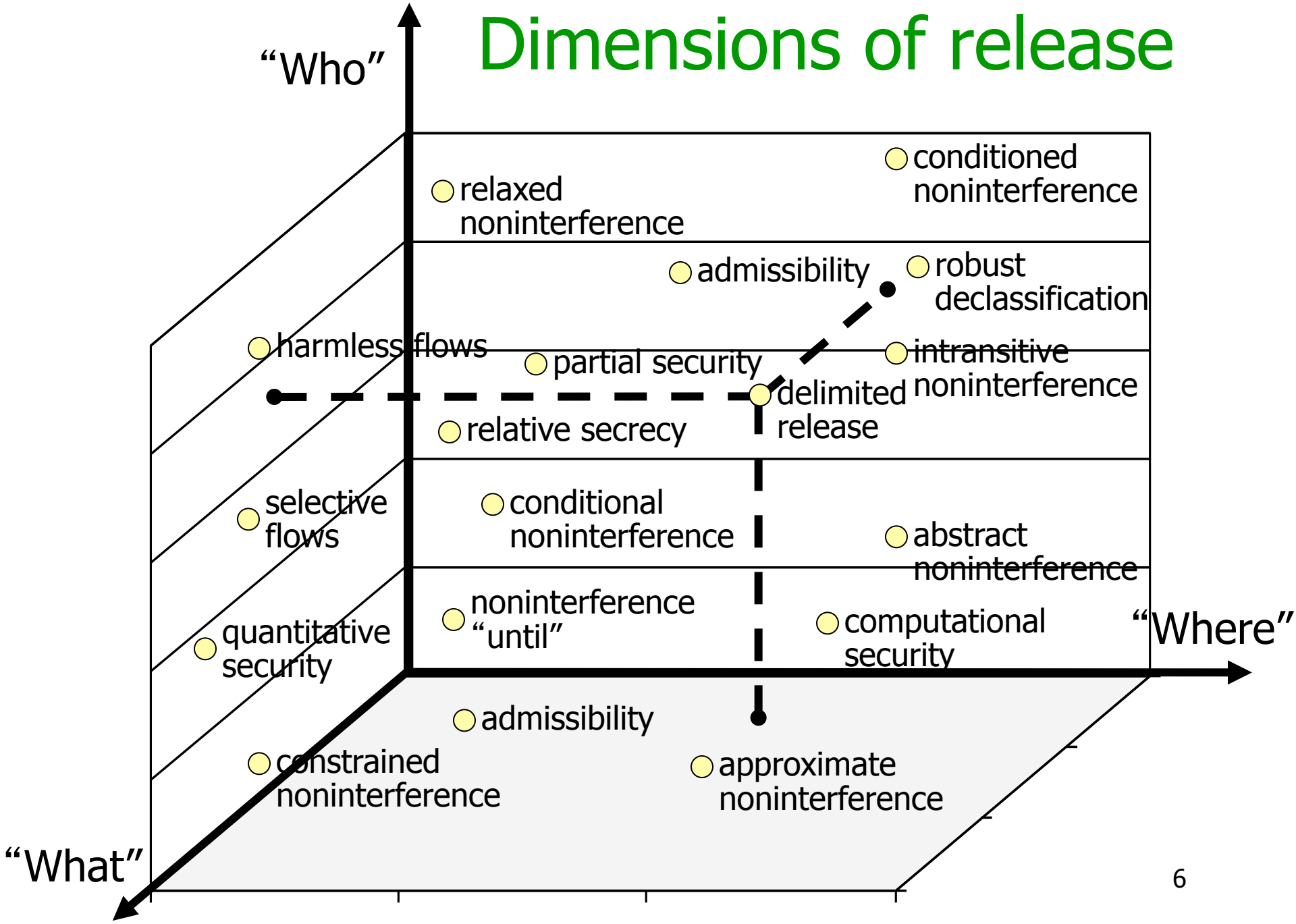
- Allowing leaks might compromise confidentiality
- Noninterference is violated
- How do we know secrets are not laundered via release mechanisms?
- Need for security assurance for programs with release



State-of-the-art

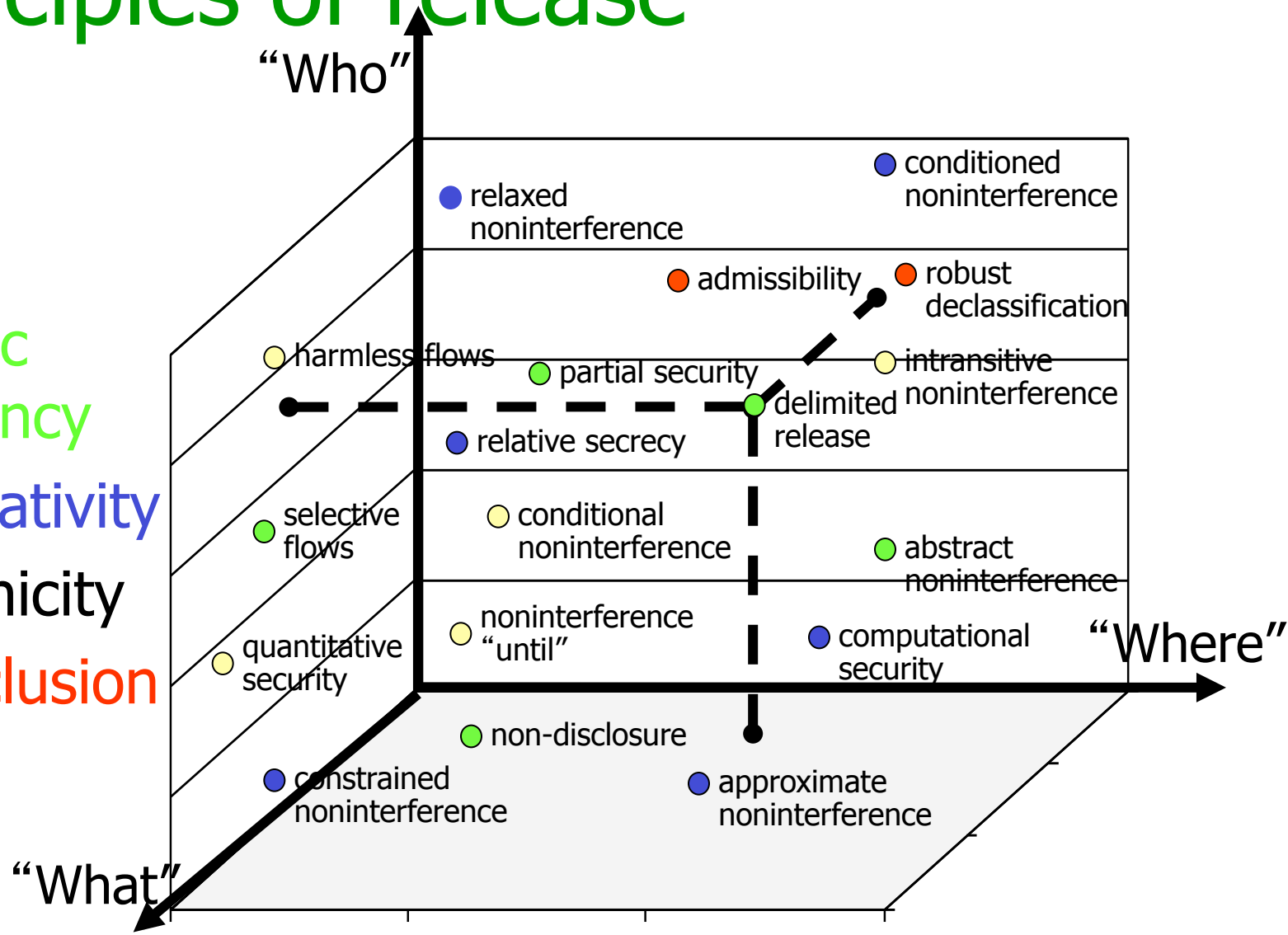


Dimensions of release



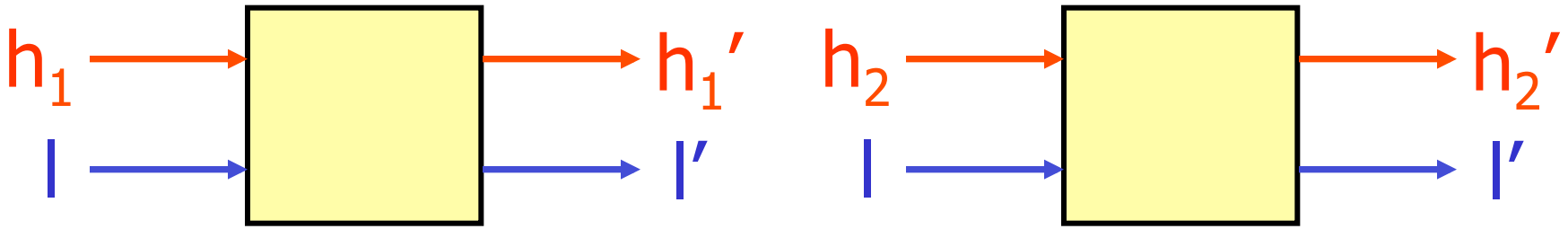
Principles of release

- Semantic consistency
- Conservativity
- Monotonicity
- Non-occlusion



What

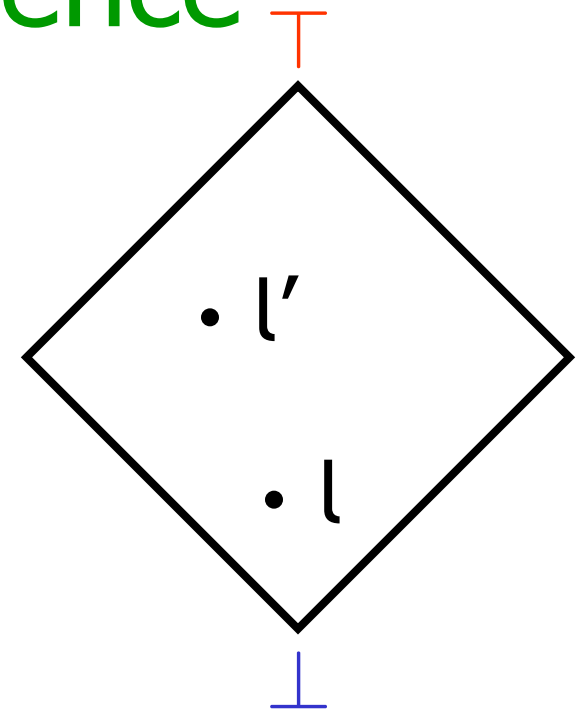
- Noninterference [Goguen & Meseguer]: as high input varied, low-level outputs unchanged



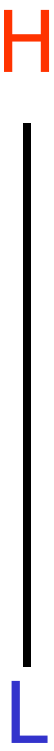
- Selective (partial) flow
  - Noninterference within high sub-domains [Cohen' 78, Joshi & Leino' 00]
  - Equivalence-relations view [Sabelfeld & Sands' 01]
  - Abstract noninterference [Giacobazzi & Mastroeni' 04,' 05]
  - Delimited release [Sabelfeld & Myers' 04]
- Quantitative information flow [Denning' 82, Clark et al.' 02, Lowe' 02]

Security lattice and noninterference

Security lattice:



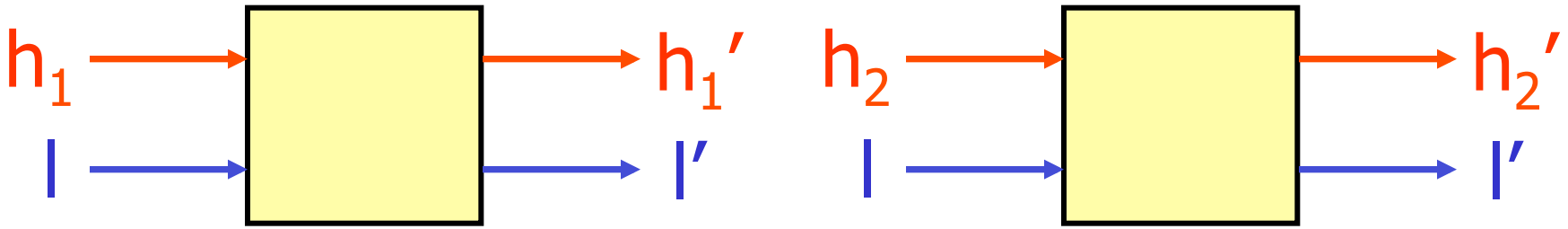
e.g.:



Noninterference: flow from l to l′ allowed when $l \sqsubseteq l'$

Noninterference

- Noninterference [Goguen & Meseguer]: as high input varied, low-level outputs unchanged



- Language-based noninterference for c:

$$M_1 =_L M_2 \;\&\; \langle M_1, c\rangle \Downarrow M'_1 \;\&\; \langle M_2, c\rangle \Downarrow M'_2 \Rightarrow M'_1 =_L M'_2$$

Low-memory equality:
$M_1 =_L M_2$ iff $M_1|_L = M_2|_L$

Configuration with M_2 and c

Average salary

- Intention: release average

$$avg := declassify((h_1 + \ldots + h_n)/n, low);$$

- Flatly rejected by noninterference
- If accepting, how do we know declassify does not release more than intended?
- Essence of the problem: what is released?
- "Only declassified data and no further information"
- Expressions under declassify: "escape hatches"

Delimited release

[Sabelfeld & Myers, ISSS'03]

- Command c has expressions declassify(e_i,L); c is secure if:

$$M_1 =_L M_2 \;\&\; \langle M_1, c\rangle \Downarrow M'_1 \;\&\; \langle M_2, c\rangle \Downarrow M'_2 \;\&\; \forall i\ .\mathrm{eval}(M_1, e_i) = \mathrm{eval}(M_2, e_i) \Rightarrow M'_1 =_L M'_2$$

if M_1 and M_2 are indistinguishable through all e_i...

...then the entire program may not distinguish M_1 and M_2

$\Rightarrow$ security

- For programs with no declassification: Security $\Rightarrow$ noninterference

Average salary revisited

- Accepted by delimited release:

$$\text{avg}:=\text{declassify}((h_1+\ldots+h_n)/n,\text{low});$$

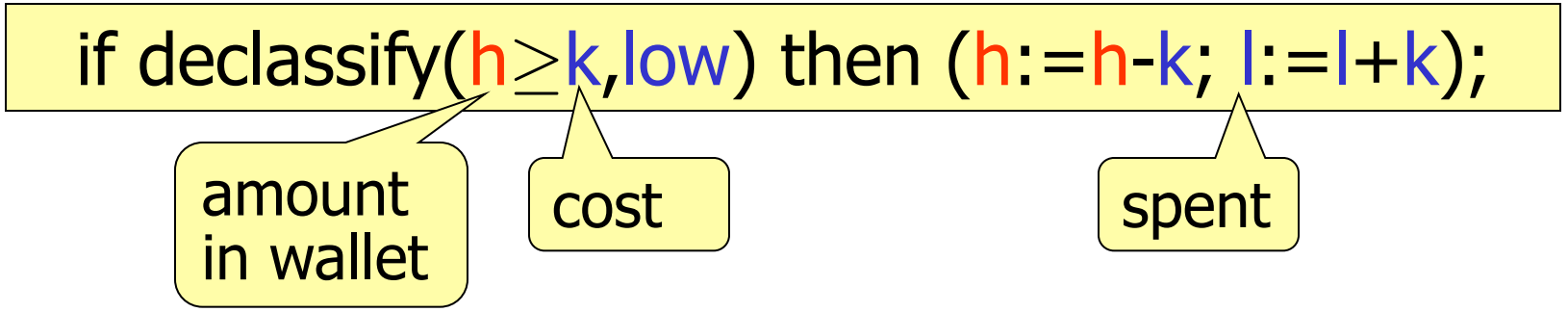
$$\text{temp}:=h_1;\ h_1:=h_2;\ h_2:=\text{temp};\ \text{avg}:=\text{declassify}((h_1+\ldots+h_n)/n,\text{low});$$

- Laundering attack rejected:

$$h_2:=h_1;\ldots;\ h_n:=h_1;\ \text{avg}:=\text{declassify}((h_1+\ldots+h_n)/n,\text{low}); \sim \text{avg}:=h_1$$

Electronic wallet

- If enough money then purchase



- Accepted by delimited release

Electronic wallet attack

- Laundering bit-by-bit attack (h is an n-bit integer)

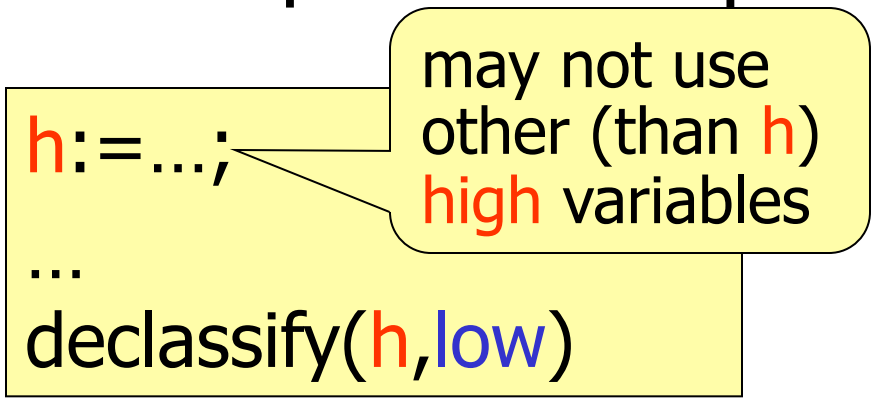
```
l:=0;
while(n>0) do
   k:=2^(n-1);
   if declassify(h≥k,low)
      then (h:=h-k; l:=l+k);
   n:=n-1;
```

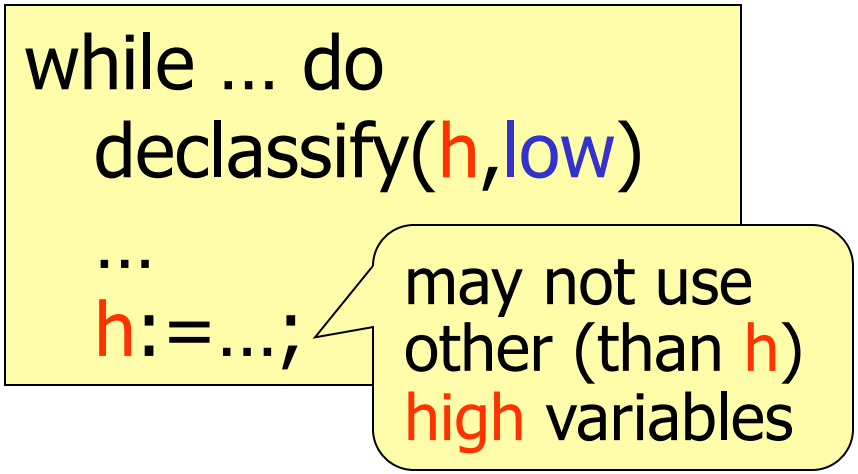
$\sim$ `l:=h`

- Rejected by delimited release

Security type system

- Basic idea: prevent new information from flowing into variables used in escape hatch expressions





- Theorem:
  c is typable $\Rightarrow$ c is secure

Who

- Robust declassification in a language setting [Myers, Sabelfeld & Zdancewic'04/06]
- Command c[•] has robustness if

$$\forall M_1, M_2, a, a'.\ \langle M_1, c[a]\rangle \approx_L \langle M_2, c[a]\rangle \Rightarrow \langle M_1, c[a']\rangle \approx_L \langle M_2, c[a']\rangle$$

attacks

- If a cannot distinguish between M_1 and M_2 through c then no other a' can distinguish between M_1 and M_2

Robust declassification: examples

- Flatly rejected by noninterference, but secure programs satisfy robustness:

$[\bullet]; x_{LH} := \text{declassify}(y_{HH}, LH)$

$[\bullet]; \text{if } x_{LH} \text{ then } y_{LH} := \text{declassify}(z_{HH}, LH)$

- Insecure program:

$[\bullet]; \text{if } x_{LL} \text{ then } y_{LL} := \text{declassify}(z_{HH}, LH)$

is rejected by robustness

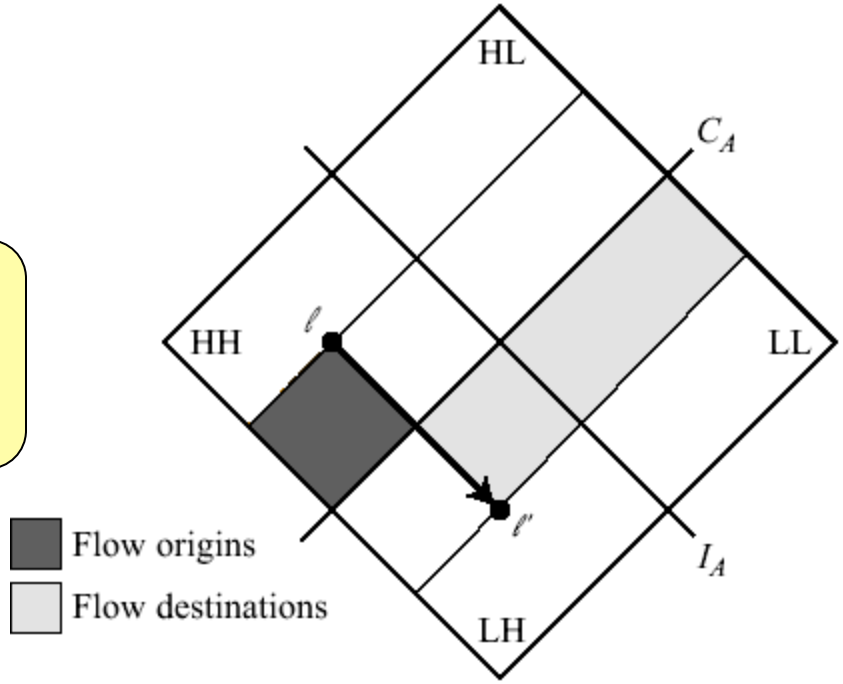
Enforcing robustness

- Security typing for declassification:

context must be high-integrity

data must be high-integrity

$$\frac{LH \vdash e : HH}{LH \vdash \text{declassify}(e,l') : LH}$$



Where

- Intransitive (non)interference
  - assurance for intransitive flow [Rushby' 92, Pinsky' 95, Roscoe & Goldsmith' 99]
  - nondeterministic systems [Mantel' 01]
  - concurrent systems [Mantel & Sands' 04]
  - to be declassified data must pass a downgrader [Ryan & Schneider' 99, Mullins' 00, Dam & Giambiagi' 00, Bossi et al.' 04, Echahed & Prost' 05, Almeida Matos & Boudol' 05]

When

- Time-complexity based attacker
  - password matching [Volpano & Smith' 00] and one-way functions [Volpano' 00]
  - poly-time process calculi [Lincoln et al.' 98, Mitchell' 01]
  - impact on encryption [Laud' 01,' 03]
- Probabilistic attacker [DiPierro et al.' 02, Backes & Pfitzmann' 03]
- Relative: specification-bound attacker [Dam & Giambiagi' 00,' 03]
- Non-interference "until" [Chong & Myers' 04]

Principle I

Semantic consistency

The (in)security of a program is invariant under semantics-preserving transformations of declassification-free subprograms

- Aid in modular design
- "What" definitions generally semantically consistent
- Uncovers semantic anomalies

Principle II

Conservativity

Security for programs with no declassification is equivalent to noninterference

- Straightforward to enforce (by definition); nevertheless:
- Noninterference "until" rejects

if h>h then l:=0

Principle III

Monotonicity of release

Adding further declassifications to a secure program cannot render it insecure

- Or, equivalently, an insecure program cannot be made secure by *removing* declassification annotations
- "Where": intransitive noninterference (a la M&S) fails it; declassification actions are observable

```
if h then declassify(l=l) else l=l
```

Principle IV

Occlusion

The presence of a declassification operation cannot mask other covert declassifications

Checking the principles

What

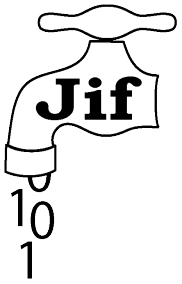
Property	Semantic consistency	Conservativity	Monotonicity of release	Non-occlusion
Partial release [Coh78, JL00, SS01, GM04, GM05]	✓	✓	N/A	✓
Delimited release [SM04]	✓	✓	✓	✓
Relaxed noninterference [LZ05a]	×	✓	✓	✓
Naive release	✓	✓	✓	×
Who				
Robust declassification [MSZ04]	✓*	✓	✓	✓
Qualified robust declassification [MSZ04]	✓*	✓	✓	×
Where				
Intransitive noninterference [MS04]	✓*	✓	×	✓
When				
Admissibility [DG00, GD03]	×	✓	×	✓
Noninterference "until" [CM04]	×	×	✓	✓
Typeless noninterference "until"	✓*	✓	×	×

* Semantic anomalies

Declassification in practice: A case study

[Askarov & Sabelfeld, ESORICS'05]

- Use of security-typed languages for implementation of crypto protocols
- Mental Poker protocol by [Roca et.al, 2003]
  - Environment of mutual distrust
  - Efficient
- Jif language [Myers et al., 1999-2005]
  - Java extension with security types
  - Decentralized Label Model
  - Support for declassification
- Largest code written in security-typed language up to publ date [~4500 LOC]



Security assurance/Declassification

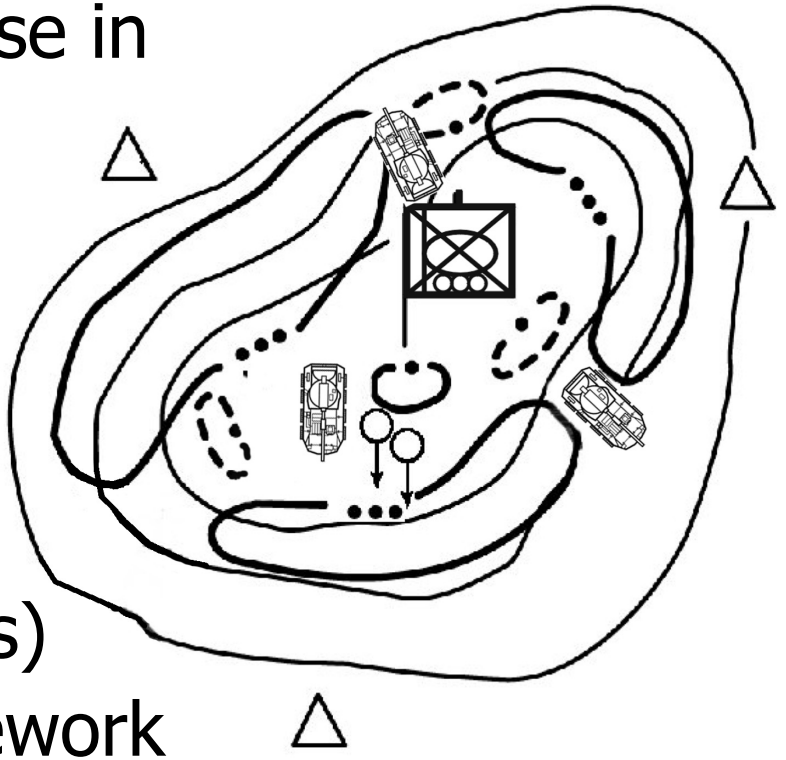
Group	Pt.	What	Who	Where
I	1 2	Public key for signature Public security parameter	Anyone Player	Initialization Initialization
II	3 4-7 8-10	Message signature Protocol initialization data Encrypted permuted card	Player Player Player	Sending msg Initialization Card drawing
III	11	Decryption flag	Player	Card drawing
IV	12-13 14	Player's secret encryption key Player's secret permutation	Player Player	Verification Verification

Group I – naturally public data Group II – required by crypto protocol

Group III – success flag pattern Group IV – revealing keys for verification

Dimensions: Conclusion

- Road map of information release in programs
- Step towards policy perimeter defense: to protect along each dimension
- Prudent principles of declassification (uncovering previously unnoticed anomalies)
- Need for declassification framework for relation and combination along the dimensions



References

- Declassification: Dimensions and Principles
  [Sabelfeld & Sands, JCS]