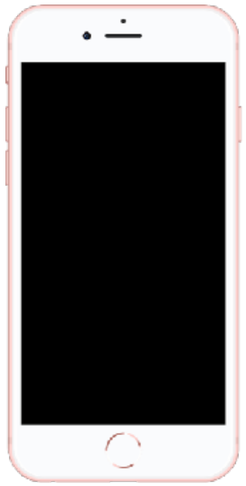
Information Flow Control

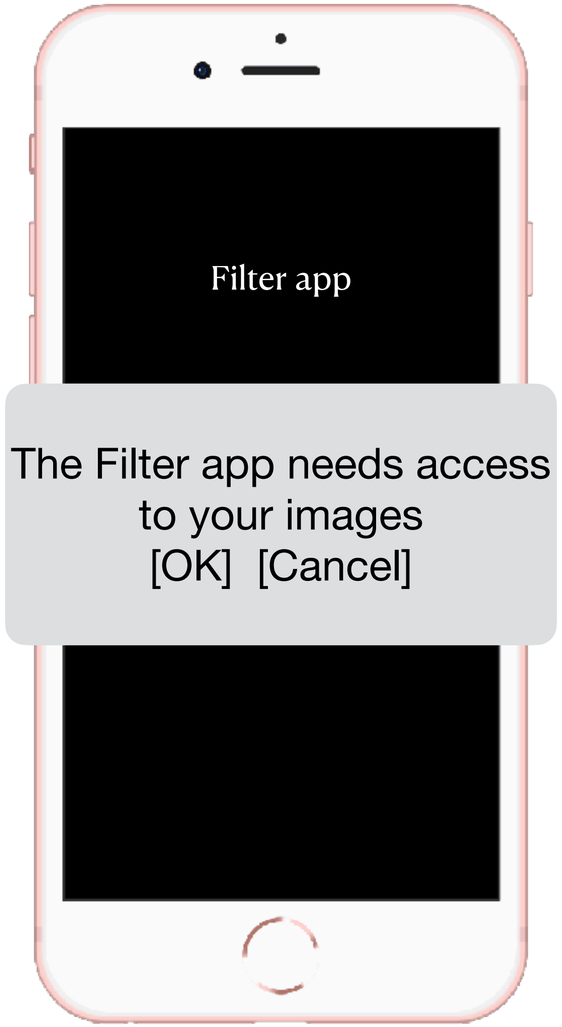
Part 1

Language-Based Security
aslan@cs.au.dk



Filter App

Imaginary filter app



Imaginary filter app

Filter app

Imaginary filter app



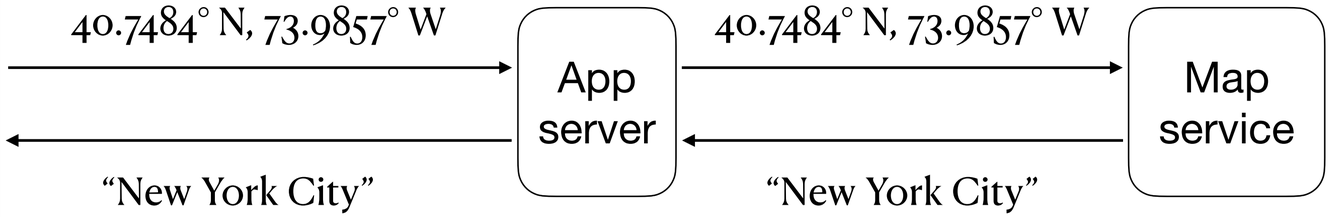






Imaginary filter app

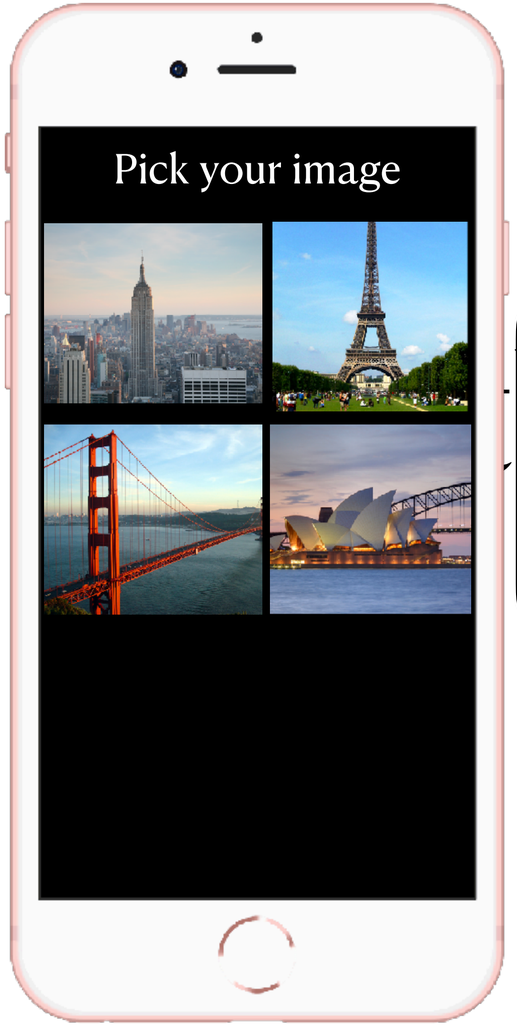
How does it know the GPS location of the photo? and the mapping from the coordinates to the name?



App server log:

```
...
{ID="user@au.dk", Coordinates="40.7484° N, 73.9857° W", …}
...
```

*Essentially a **perpetual** entry:* it will be saved in the log of the app server, replicated in the backups of the app's cloud provider, mined for ads by the services's *real* clients (i.e., not the user!), tapped into by government agencies (not necessarily your government), and resold to data brokers when the app is no longer maintained



App server log

```
...
{ID="user@au.dk", Coordinates="40.7484° N, 73.9857° W", Date = "2018-02-21", …},
{ID="user@au.dk", Coordinates="48.8584° N, 2.2945° E", Date = "2018-02-01", …},
{ID="user@au.dk", Coordinates="33.8568° S, 151.2153° E", Date = "2018-02-10", …},
{ID="user@au.dk", Coordinates="37.8199° N, 122.4783° W", Date = "2018-02-19", …},
...
```

Metadata *is* data

Metadata is data

- https://www.nytimes.com/interactive/2019/12/19/opinion/location-tracking-cell-phone.html

- https://www.nytimes.com/interactive/2019/12/20/opinion/location-data-national-security.html

What can we do?

- Access control? Prevent the app from reading the location or provide an empty location on read
  - OK, but then the app can’t save the new image with the location info, so we lose a feature
- Encrypt the location?
  - Better: the app can re-save the encrypted blob, but still cannot show it to the user in the GUI or search by a location, again we lose a feature…

Identifying whom to trust

A better design: resolve the coordinates on the device through a map service trusted by the user

updated filters

App server

Untrusted by the user

40.7484° N, 73.9857° W

"New York City"

Map service

Trusted by the user

What is confidential?

- “Metadata”:
  - Everything IoT
    - From health trackers to lightbulbs to toasters
  - Browsing history, TV watching history
  - “Nothing to hide argument”? doesn’t work
- Regular data:
  - Emails, bank accounts, identity information, etc...

What isn’t confidential?

Confidential data is processed by software

- There is generally little control over how software handles confidential data *once* it has access to it:
  - Apps that access contact lists
  - Spell-checker and other plugins in editors
  - JavaScript snippets on a page (recall example from Lecture 01)
  - Shell scripts with with root access
  - Financial software
- Isolation does not work because systems need to communicate with the rest of the world.
- Claiming that encryption is sufficient is often ignorant at the very least, dishonest at worst
  - we obviously need the strong guarantees of crypto protocols and primitives, but cryptography provides no guarantees for how software handles data after it is decrypted

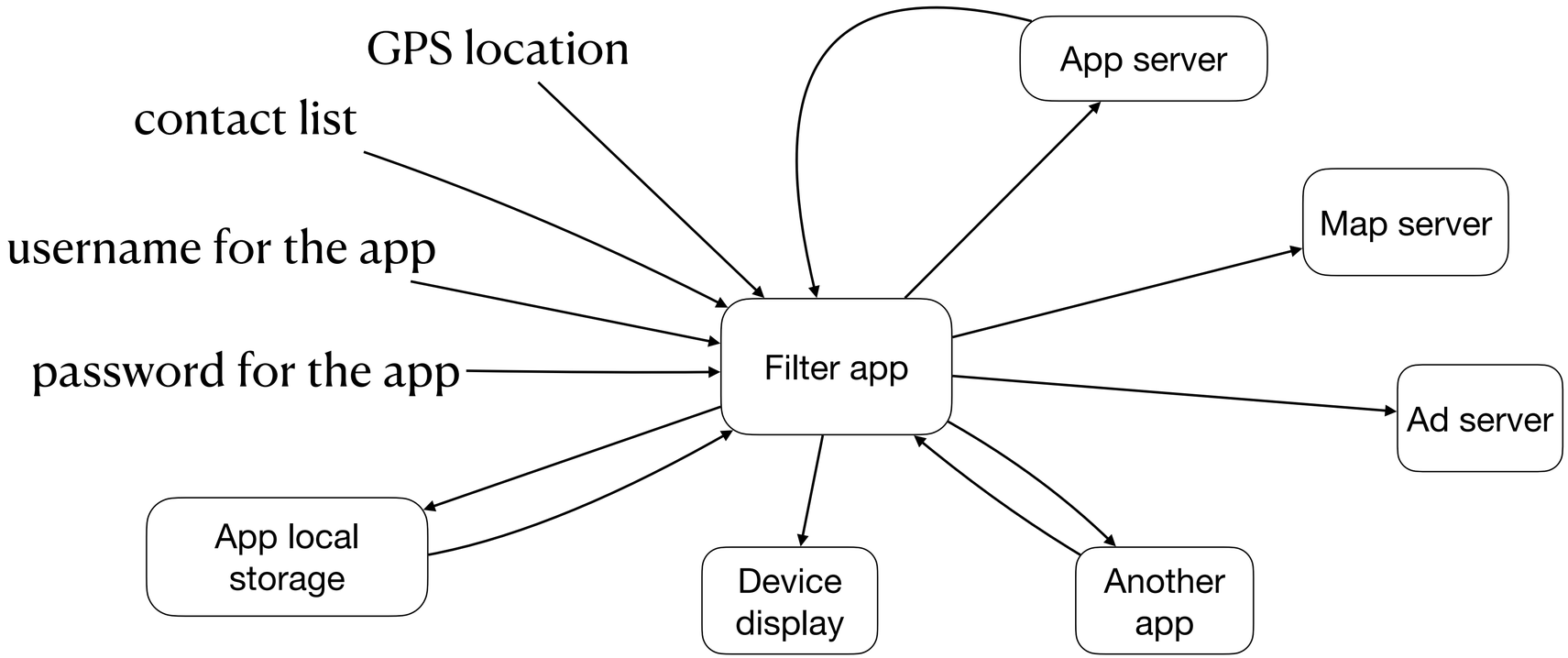
Malice vs Incompetence

- Our filter app developers may mean no harm and declare on their website that they "don't collect any sensitive data".
  - But how can they guarantee that?
  - An honest mistake may include a line of code such as

    ```
    saveToServerLog(obj.toString())
    ```

    - If the `obj` is the image, `toString` may contain the location
- Same for library code used with the application
- What about the code running on the app's backend servers?

Information Flow Control



How does confidential information propagate within the system?
Not just the data itself, but also derived information

Ex: success of a login, whether one GUI element is larger than the other, length of a message sent to a servers, duration of computation

Recall *Functional Levels of Information Protection* from Saltzer & Schroeder [1975]

- *e) Putting strings on information:* [...] The fourth level of capability is to maintain some control over the user of the information even after it has been released. Such control is desired, for example, in releasing income information to a tax advisor; constraints should prevent him from passing the information on to a firm which prepares mailing lists. The printed labels on classified military information declaring a document to be "Top Secret" are another example of a constraint on information after its release to a person authorized to receive it. One may not (without risking severe penalties) release such information to others, and the label serves as a notice of the restriction. Computer systems that implement such strings on information are rare and the mechanisms are incomplete. For example, the ADEPT system [14] keeps track of the classification level of all input data used to create a file; all output data are automatically labeled with the highest classification encountered during execution.

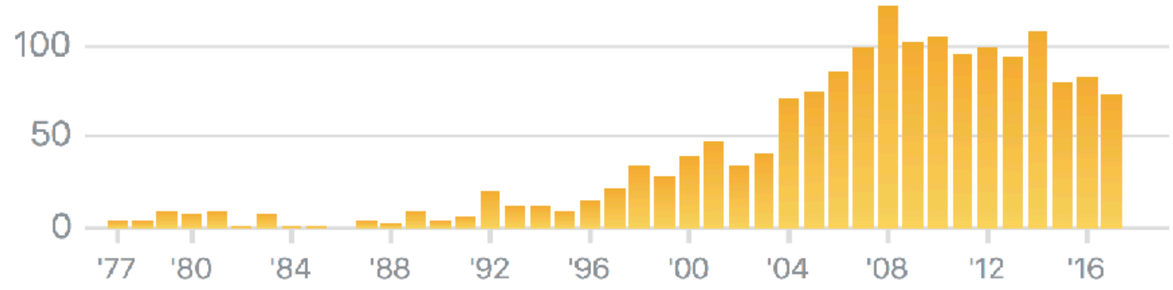
What we need in order to reason about information flow

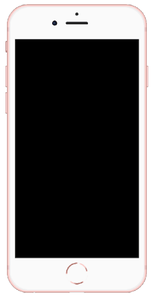
- Classification of who is allowed to know what (for confidentiality) and who is allowed to influence what (for integrity/availability)
  - Often specified using *security levels*
- Specification of what it means for a program to be secure given the security levels
  - This is our security policy
  - Requires also the spec of how the system works
- Tools that enforce these policies against both malicious parties and honest mistakes
- Assurance that these tools are sound

Information Flow Historically

- Lots of foundational work in late 1970s, early 1980s
- Very little interest throughout 1980s until late 1990s
- Explosion of interest from 1999 – present

Citation graph for the paper "A Lattice Model of Secure Information Flow" by Dorothy E. Denning, 1976; (source: semantic scholar)





Plan for today

- System model = simple imperative language
  - Single-threaded, deterministic
  - Batch-style execution: one input; one output
- Simple model of just 2 security levels: public and secret
- A noninterference security policy for confidentiality: public outputs should not reveal any information about secrets
- Brainstorming of how one would design a tool
- No proofs

Simple imperative language

The grammar

Commands: $c ::= \text{skip} \mid x := e \mid c; c \mid \text{if } e \text{ then } c \text{ else } c \mid \text{while } e \text{ do } c$

Expressions: $e ::= n \mid x \mid e \ op \ e$

Example program `while x>0 do x:=x-1`

Values are only natural numbers

Memory *m* - function from variable names to values

Example: starting from initial memory m such that m(x)=5, the evaluation of our example program results in final memory m', such that m'(x)=0

Formalizing the evaluation: expressions

Constant expression *n* evaluates to value *n* in any memory *m*

Variable expression *x* evaluates to value *v* in memory *m* when *m(x)=v*

$$\text{E-CONST}\quad \frac{}{\langle n, m\rangle \Downarrow n}$$

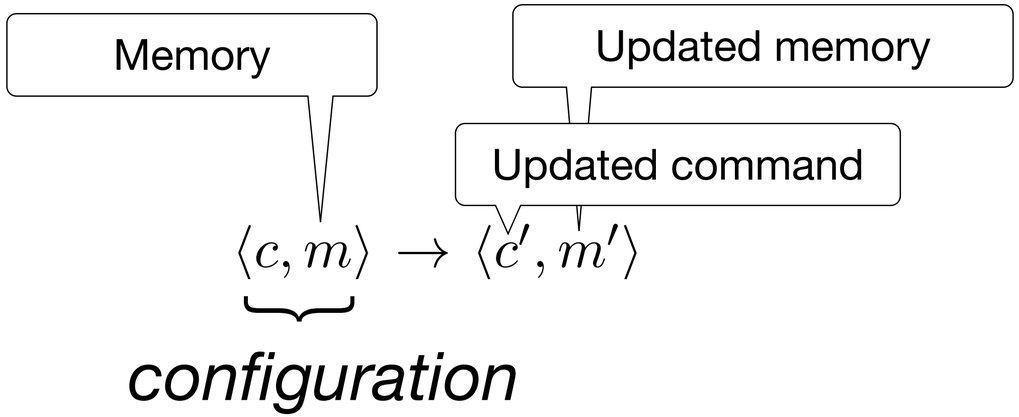
$$\text{E-VAR}\quad \frac{m(x) = v}{\langle x, m\rangle \Downarrow v}$$

$$\text{E-OP}\quad \frac{\langle e_i, m\rangle \Downarrow v_i, i = 1, 2 \qquad v = v_1 \ \mathbf{op}\ v_2}{\langle e_1\ op\ e_2, m\rangle \Downarrow v}$$

Expressions of the form e_1 *op* e_2, where op is a binary operator such as (+,-,<,==, etc) evaluates to value v if e_1 *evaluates to* v_1, *and* e_2 *evaluates to* v_2, *and v equals* v_1 *op* v_2

Q: what's the expression derivation tree for $\langle x + 1, m \rangle$ when $m(x)=42$

Formalizing the evaluation: step relation



Memory update $m[x \mapsto v] \triangleq \lambda y \ . \ \text{if } x = y \text{ then } v \text{ else } m(y)$

Reachability in zero or more steps $\langle c, m \rangle \to^* \langle c', m' \rangle$

does not appear in source syntax

Terminal configuration: $\langle \mathsf{stop}, m \rangle$

Step relation

S-SKIP

$$\frac{}{\langle \mathsf{skip}, m\rangle \to \langle \mathsf{stop}, m\rangle}$$

S-ASSIGN

$$\frac{\langle e, m\rangle \Downarrow v}{\langle x := e, m\rangle \to \langle \mathsf{stop}, m[x \mapsto v]\rangle}$$

S-SEQ1

$$\frac{\langle c_1, m\rangle \to \langle \mathsf{stop}, m'\rangle}{\langle c_1; c_2, m\rangle \to \langle c_2, m'\rangle}$$

S-SEQ2

$$\frac{\langle c_1, m\rangle \to \langle c_1', m'\rangle \qquad c_1' \neq \mathsf{stop}}{\langle c_1; c_2, m\rangle \to \langle c_1; c_2, m'\rangle}$$

S-IF1

$$\frac{\langle e, m\rangle \Downarrow v \qquad v \neq 0}{\langle \mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2, m\rangle \to \langle c_1, m\rangle}$$

S-IF2

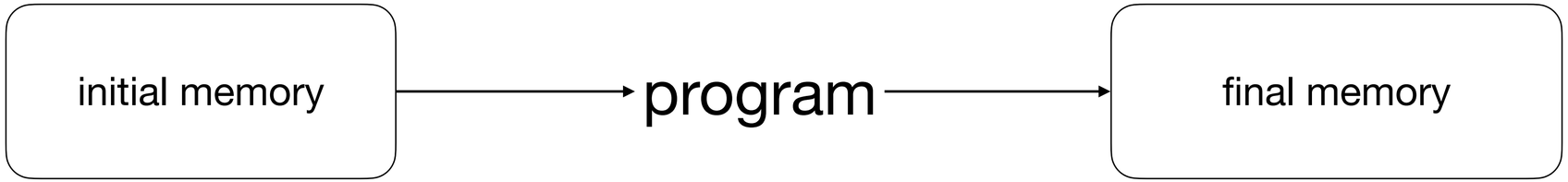
$$\frac{\langle e, m\rangle \Downarrow v \qquad v = 0}{\langle \mathsf{if}\ e\ \mathsf{then}\ c_1\ \mathsf{else}\ c_2, m\rangle \to \langle c_2, m\rangle}$$

S-WHILE

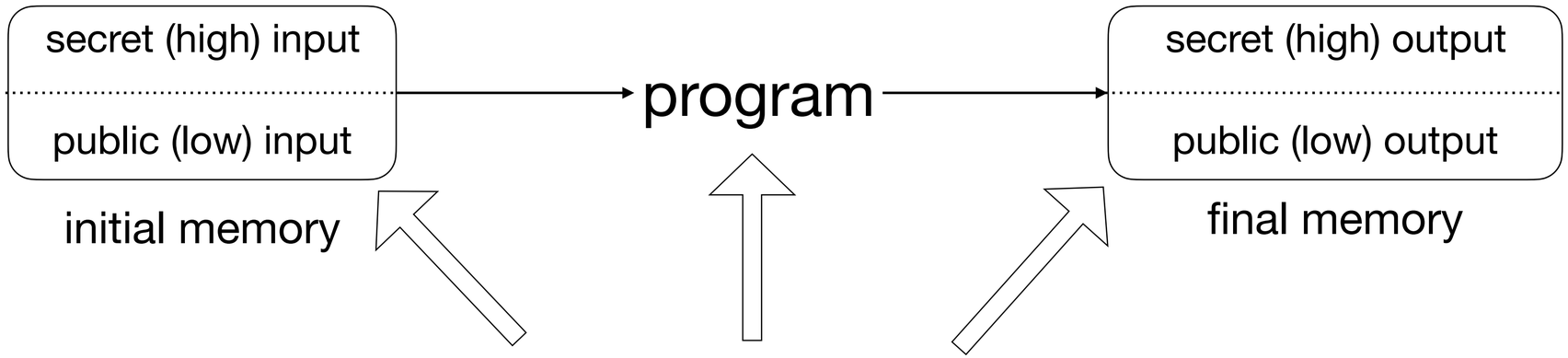
$$\frac{}{\langle \mathsf{while}\ e\ \mathsf{do}\ c, m\rangle \to \langle \mathsf{if}\ e\ \mathsf{then}\ c; \mathsf{while}\ e\ \mathsf{do}\ c\ \mathsf{else}\ \mathsf{skip}, m\rangle}$$

Q: how many steps does it take to evaluate the example program `while x>0 do x:=x-1` starting from memory *m* such that *m*(x)=5

System model



Security model



Assume attacker knows: program source, public input, and public output

Convention: variables with suffix _**p** stand for public variables, and with suffix _**s** for secret ones

x_p and z_p are public

y_s is secret

Building an intuition towards a security policy through examples

```
x_p := y_s
```

```
x_p := 42
```

```
y_s := 42; x_p := y_s
```

Building an intuition towards a security policy through examples

```
if y_s > 0 then x_p := 1 else x_p := 0
```

```
if y_s > 0 then x_p := 0 else x_p := 0
```

```
while y_s > 0 do skip
```

```
x_p:= 1; (while y_s > 0 do skip) x_p := 2
```

Security definition

- Noninterference: Goguen & Meseguer 1983
- Many variants in literature…

Initial memories agree on public variables

Definition 2 (Basic noninterference)**.** Program c is secure when for all pairs of memories m_1 and m_2 such that $m_1 \sim m_2$, and

$$\langle c, m_1 \rangle \rightarrow^* \langle \mathsf{stop}, m_1' \rangle$$

and

$$\langle c, m_2 \rangle \rightarrow^* \langle \mathsf{stop}, m_2' \rangle$$

it holds that $m_1' \sim m_2'$.

Final memories agree on public variables

OBS: two-trace property

Revisit the examples with the definition in mind

```
x_p := y_s
```

```
x_p := 42
```

```
y_s := 42; x_p := y_s
```

Definition 2 (Basic noninterference)**.** Program c is secure when for all pairs of memories m_1 and m_2 such that $m_1 \sim m_2$, and

$$\langle c, m_1 \rangle \rightarrow^* \langle \mathsf{stop}, m_1' \rangle$$

and

$$\langle c, m_2 \rangle \rightarrow^* \langle \mathsf{stop}, m_2' \rangle$$

it holds that $m_1' \sim m_2'$.

Revisit the examples with the definition in mind

```
if y_s > 0 then x_p := 1 else x_p := 0
```

```
if y_s > 0 then x_p := 0 else x_p := 0
```

```
while y_s > 0 do skip
```

Definition 2 (Basic noninterference)**.** Program c is secure when for all pairs of memories m_1 and m_2 such that $m_1 \sim m_2$, and

$$\langle c, m_1 \rangle \rightarrow^* \langle \mathsf{stop}, m_1' \rangle$$

and

$$\langle c, m_2 \rangle \rightarrow^* \langle \mathsf{stop}, m_2' \rangle$$

it holds that $m_1' \sim m_2'$.

Designing the mechanism

- Criteria:
  - Relative permissiveness (should be possible to write useful programs)
  - Soundness: flag all definite violations
    - Incomplete: there will be false positives
- Option A:
  - Type system
- Option B:
  - Dynamic mechanism: detect potential information flow violations at runtime
    - Fail-safe: crash *before* possible information
      - Reasoning: crashes can be observed (unlike leaks)

Enforcement via a type system

Type system

variable security type environment: mapping from variable names to levels

Typing for expressions: $\Gamma \vdash e : \ell.$

$$\text{T-INT}\quad \frac{}{\Gamma \vdash n : \ell}$$

$$\text{T-VAR}\quad \frac{\Gamma(x) = \ell}{\Gamma \vdash x : \ell}$$

$$\text{T-OP}\quad \frac{\Gamma \vdash e_i : \ell_i, (i = 1, 2)}{\Gamma \vdash e_1 \ op \ e_2 : \ell_1 \sqcup \ell_2}$$

$\ell_1 \sqcup \ell_2$ is public if both ℓ_1 and ℓ_2 are public; secret otherwise

Example: in environment Γ, where Γ(x) = public and Γ(y) = secret, we have:

Γ ⊢ 42 : public Γ ⊢ x : public Γ ⊢ x + 1 : public

Γ ⊢ y +1 : secret Γ ⊢x + y : secret Γ ⊢ y - y : secret

Type system

program counter label

Typing for expressions: $\Gamma, pc \vdash c$

This prevents updates to public variables in secret-dependent branches

$\ell_1 \sqsubseteq \ell_2$ when ℓ_1 is public OR ℓ_2 is secret

T-SKIP

$$\frac{}{\Gamma, pc \vdash \text{skip}}$$

T-ASSIGN

$$\frac{\Gamma \vdash e : \ell \qquad pc \sqcup \ell \sqsubseteq \Gamma(x)}{\Gamma, pc \vdash x := e}$$

T-SEQ

$$\frac{\Gamma, pc \vdash c_i, (i = 1, 2)}{\Gamma, pc \vdash c_1; c_2}$$

T-IF

$$\frac{\Gamma \vdash e : \ell \qquad \Gamma, pc \sqcup \ell \vdash c_i, (i = 1, 2)}{\Gamma, pc \vdash \text{if } e \text{ then } c_1 \text{ else } c_2}$$

raises the pc-label for the branches

T-WHILE

$$\frac{\Gamma \vdash e : \ell \qquad \Gamma, pc \sqcup \ell \vdash c}{\Gamma, pc \vdash \text{while } e \text{ do } c}$$

Example derivation tree

$$\dfrac{\dfrac{\Gamma \vdash 0 : \mathsf{Public} \qquad \mathsf{Public} \sqsubseteq \Gamma(\mathtt{x})}{\Gamma, \mathsf{Public} \vdash \mathtt{x\_p} := 0} \qquad \dfrac{\dfrac{\Gamma(\mathtt{y\_s}) = \mathsf{Secret}}{\Gamma \vdash \mathtt{y\_s} : \mathsf{Secret}} \qquad \dfrac{\dfrac{\Gamma(\mathtt{y\_s}) = \mathsf{Secret}}{\Gamma \vdash \mathtt{y\_s} : \mathsf{Secret}} \qquad \Gamma \vdash 1 : \mathsf{Public}}{\Gamma \vdash \mathtt{y\_s} - 1 : \mathsf{Secret}} \qquad \mathsf{Secret} \sqsubseteq \Gamma(\mathtt{y\_s})}{\Gamma, \mathsf{Secret} \vdash \mathtt{y\_s} := \mathtt{y\_s} - 1} \qquad \dfrac{\Gamma \vdash 1 : \mathsf{Public} \qquad \mathsf{Public} \sqsubseteq \Gamma(\mathtt{x})}{\Gamma, \mathsf{Public} \vdash \mathtt{x\_p} := 1}}{\dfrac{\Gamma, \mathsf{Public} \vdash \mathtt{while\ y\_s\ do\ y\_s} := \mathtt{y\_s} - 1}{\Gamma, \mathsf{Public} \vdash \mathtt{while\ y\_s\ do\ y\_s} := \mathtt{y\_s} - 1;\ \mathtt{x\_p} := 1}}}{\Gamma, \mathsf{Public} \vdash \mathtt{x\_p} := 0;\ \mathtt{while\ y\_s\ do\ y\_s} := \mathtt{y\_s} - 1;\ \mathtt{x\_p} := 1}$$

Why is this sound?

Theorem (VSI'96):

Given an environment Γ, a program c
such that Γ, pc ⊢ c then it must be that
for all memories m, s, such that m $\sim_\Gamma$ s
if $\langle c, m\rangle \rightarrow^* \langle stop, m'\rangle$
and $\langle c, s\rangle \rightarrow^* \langle stop, s'\rangle$
then m' $\sim_\Gamma$ s'

Will omit Γ in the future for clarity

blue part: definition of noninterference from the last lecture

There are different proof techniques that vary a bit in their details, here we only look at one. OBS: this presentation is slightly simplified from what's in the notes; for the most detailed exposition see an accompanying Coq proof at http://github.com/aslanix/SmallStepNI

Proof sketch

$\langle c, m\rangle \rightarrow^* \langle stop, m'\rangle$ means there

are intermediate configurations $\langle c_1, m_1\rangle$, $\langle c_2, m_2\rangle$ … such that

$\langle c, m\rangle \rightarrow \langle c_1, m_1\rangle \rightarrow \langle c_2, m_2\rangle \rightarrow \ldots \rightarrow \langle c_n, m_n\rangle \rightarrow \langle stop, m'\rangle$

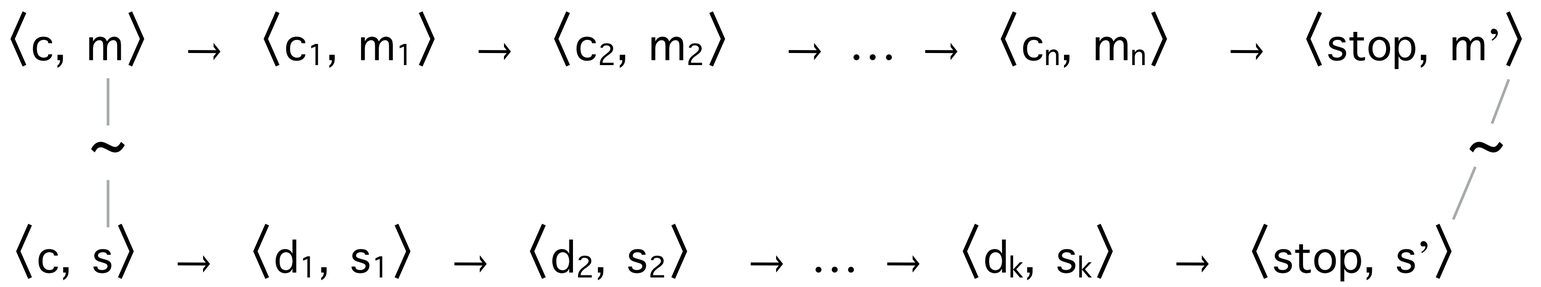
Similarly, for $\langle c, s\rangle \rightarrow^* \langle stop, s'\rangle$ we have

$\langle c, s\rangle \rightarrow \langle d_1, s_1\rangle \rightarrow \langle d_2, s_2\rangle \rightarrow \ldots \rightarrow \langle d_k, s_k\rangle \rightarrow \langle stop, s'\rangle$

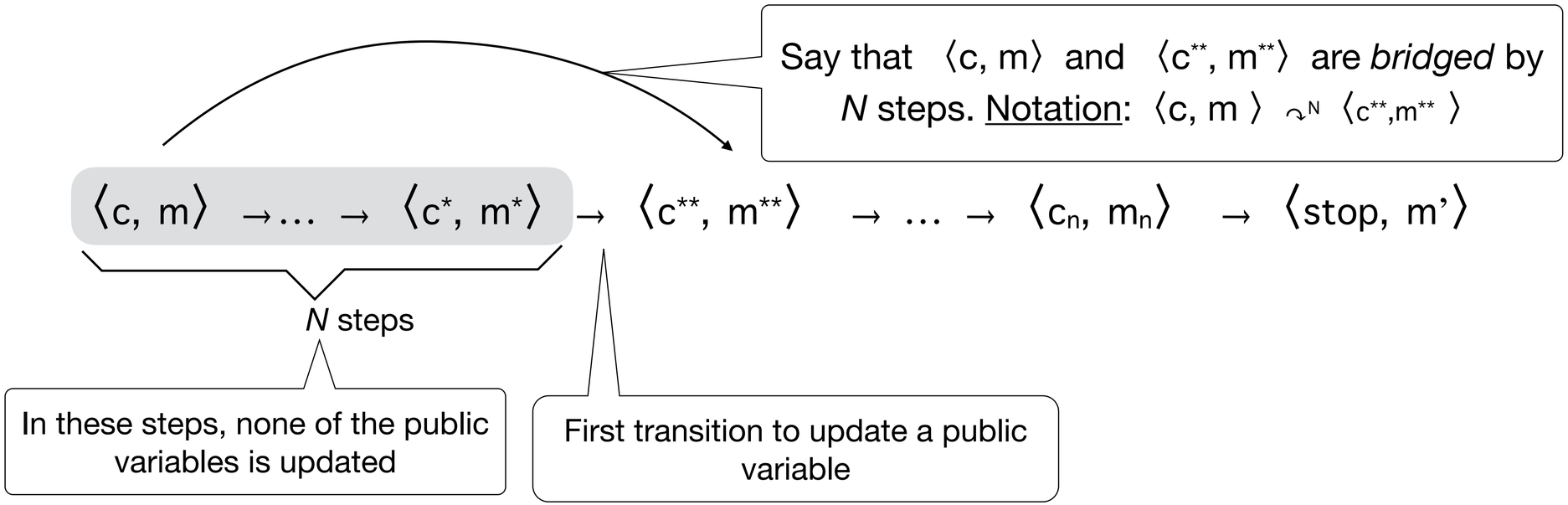
(note the potentially different number of steps and the different intermediate configurations)

What we need to prove

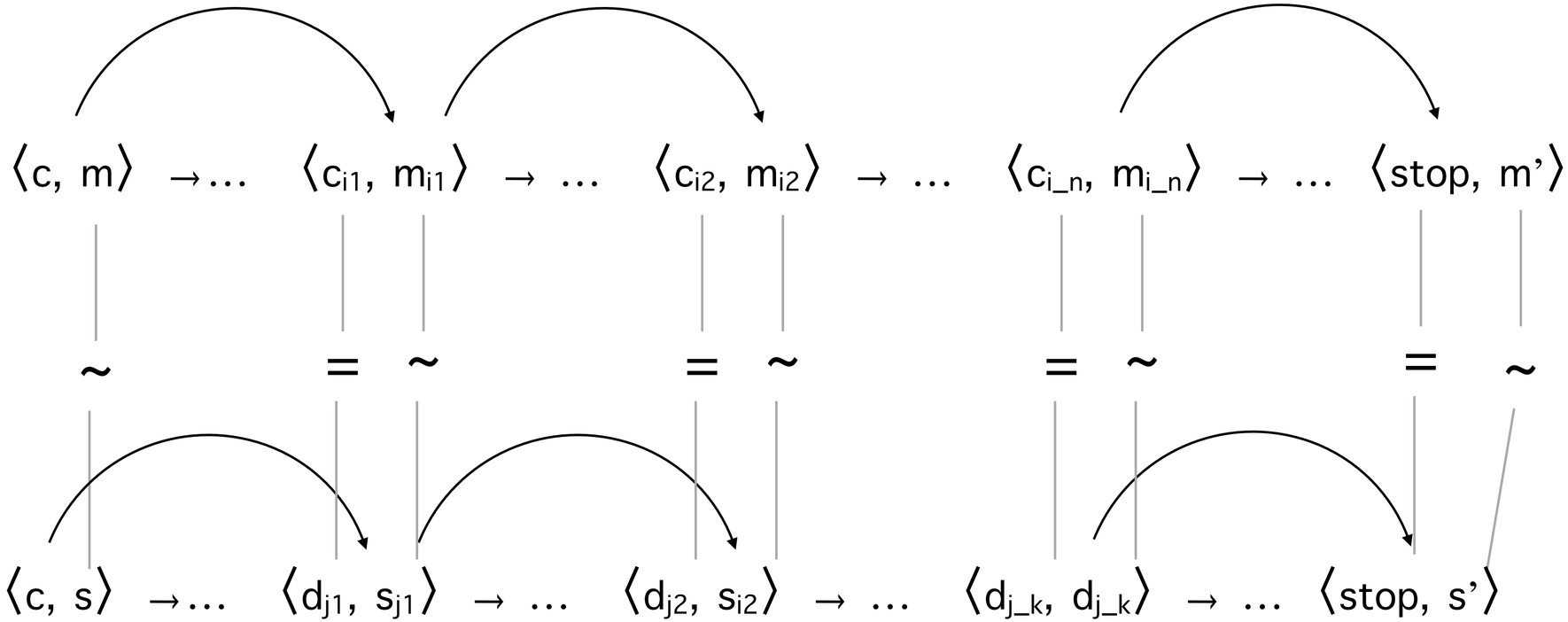
Assuming that Γ, pc $\vdash$ c, and m~s, it must be that m' ~ s'



Observation: if m and m' disagree on public variables, then m' must have been changed at some intermediate steps



Intuition for the proof

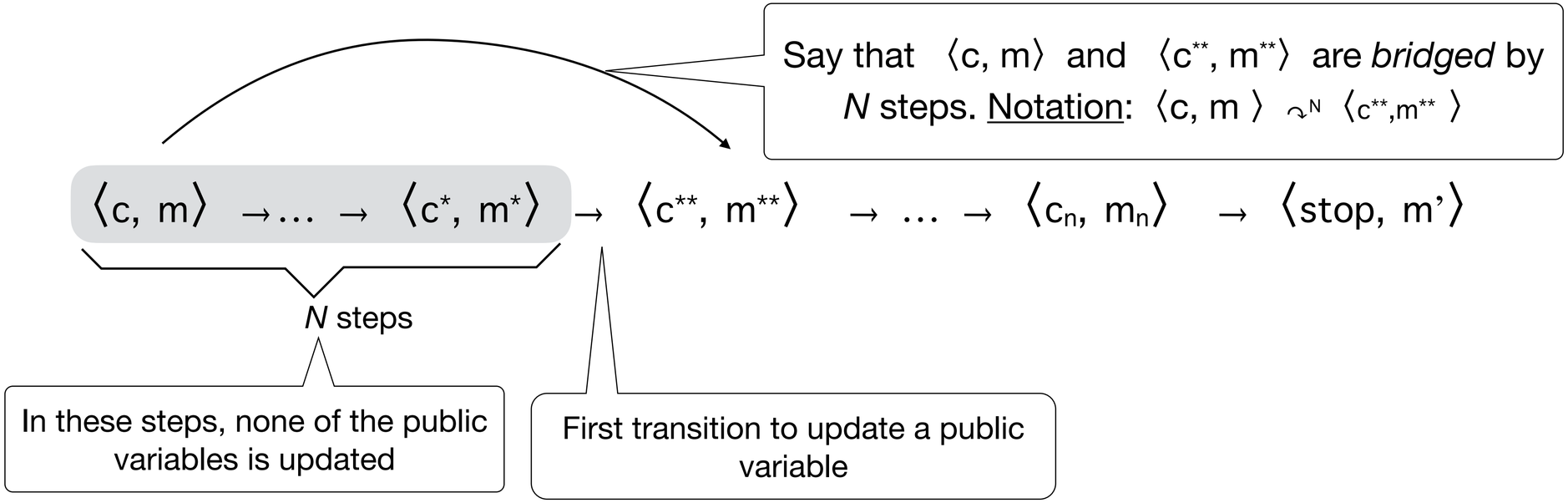


Proof: “triple-nested” induction

- by induction on the number of bridges in the first run
  - by induction on the number of intermediate steps within each bridge
    - by induction on the structure of the commands making the steps

need to formalize the bridge relation

Lemma: Bridge NI (coming later)



Bridge as an *inductive* relation

$$\frac{\langle c,\ m\rangle \ \rightarrow\ \langle \text{stop},m'\rangle \qquad m \sim m'}{\langle c,\ m\rangle \ \curvearrowright^{0}\ \langle \text{stop},m'\rangle}\ \text{Bridge-Stop}$$

some public variable is updated

$$\frac{\langle c,\ m\rangle \ \rightarrow\ \langle c',m'\rangle \qquad m \not\sim m'}{\langle c,\ m\rangle \ \curvearrowright^{0}\ \langle c',m'\rangle}\ \text{Bridge-Public}$$

none of public vars is updated

$$\frac{\langle c,\ m\rangle \ \rightarrow\ \langle c',m'\rangle \qquad m \sim m' \qquad c' \neq \text{stop} \qquad \langle c',\ m'\rangle \ \curvearrowright^{N}\ \langle c'',m''\rangle}{\langle c,\ m\rangle \ \curvearrowright^{N+1}\ \langle c'',m''\rangle}\ \text{Bridge-Multi}$$

In this definition, Bridge-Stop and Bridge-Public are the two base cases (N=0), and Bridge-Multi is the inductive clause

Main lemma: bridge noninterference

Given Γ, pc, c, m, s, N, N' such that

Γ, pc $\vdash$ c and

m $\sim$ s and

$\langle c, m \rangle \rightsquigarrow^{N} \langle c', m' \rangle$ and

$\langle c, s \rangle \rightsquigarrow^{N'} \langle d', s' \rangle$

Then: m' $\sim$ s' and c' = d'

Now that bridge is formally defined, this actually makes sense

Proof: "two-dimensional" induction. Outer layer: by induction on N; inner layer by induction on the structure of c

We also need a bunch of auxiliary lemmas

Auxiliary facts we need (1/3)

Lemma: noninterference for expressions:

Given Γ, e, m, s, ℓ such that m~s and $\Gamma \vdash$ e : ℓ and

$\langle$ e, m $\rangle \Downarrow$ v and $\langle$ e, s $\rangle \Downarrow$ u

Then: ℓ = Public $\Rightarrow$ v = u

Proof: by induction on the *structure* of e

Expression e can be one of n | x | e_1 op e_2. Consider them all

Base cases:

- e is n: We have $\Gamma \vdash$ n : Public. But also, both memories evaluate to the same value n, because $\langle$n, m $\rangle \Downarrow$ n and $\langle$n, s $\rangle \Downarrow$ n
- e is x: We have $\Gamma \vdash$ x : ℓ, where Γ(x) = ℓ. We have two cases
  - ℓ = Secret. In this case we're done
  - ℓ = Public. We have $\langle$x, m $\rangle \Downarrow$ v and $\langle$x, s $\rangle \Downarrow$ u, where v = m(x) and u = s(x). Because m ~ s, and Γ(x) = Public, we get that v = u

Inductive case: e is e_1 op e_2

Induction hypothesis: assume that the lemma holds for all e' that are structurally smaller than e (this includes e_1 and e_2)

IH: Given Γ, e' *that is structurally smaller than e*, m, s, ℓ such that m~s and Γ ⊢e' : ℓ and ⟨e', m ⟩ ⇓ v and ⟨e', s ⟩ ⇓ u. Then: ℓ = Public ⇒ v = u

Inductive case: e is e_1 op e_2

We have Γ ⊢ e_1 op e_2 : ℓ

If ℓ = Secret, we're done

If ℓ = Public then recall the typing rule for operations

$$\frac{\Gamma \vdash e_i : \ell_i, (i = 1, 2)}{\Gamma \vdash e_1 \ op \ e_2 : \ell_1 \sqcup \ell_2} \text{T-OP}$$

$\ell_1 \sqcup \ell_2$ is public if both ℓ_1 and ℓ_2 are public; secret otherwise

Since ℓ = Public, it means that both ℓ_1 and ℓ_2 must be public, too. Now, recall the rules for evaluating e_1 op e_2

$$\frac{\langle e_1, m \rangle \Downarrow v_1 \quad \langle e_2, m \rangle \Downarrow v_2 \quad v = v_1 \ \textbf{op} \ v_2}{\langle e_1 \ op \ e_2, m \rangle \Downarrow v}$$

$$\frac{\langle e_1, s \rangle \Downarrow u_1 \quad \langle e_2, s \rangle \Downarrow u_2 \quad u = u_1 \ \textbf{op} \ u_2}{\langle e_1 \ op \ e_2, s \rangle \Downarrow u}$$

We can apply the IH to e_1 because
- e_1 is structurally smaller than e
- m ~ s
- Γ ⊢ e1 : ℓ_1

From the IH and because ℓ_1 = Public, we get that $v_1 = u_1$.

Similarly, by IH for e_2, we get $v_2 = u_2$

Assuming that **op** is deterministic, from $v_1 = u_1$ and $v_2 = u_2$ we get that v = u

Auxiliary facts we need (2/3)

Lemma: preservation of types by the step relation:

Given Γ, pc, c, m such that

$$\Gamma, \text{pc} \vdash c \text{ and } \langle c, m \rangle \rightarrow \langle c', m' \rangle$$

Then: Γ, pc $\vdash$ c' or c' = stop

This clause is needed because we have no typing rule for *stop*

Proof: by induction on the structure of c

(details omitted, try yourself at home)

Auxiliary facts we need (3/3)

Lemma: Secret PC

Given Γ, c, m such that Γ, Secret $\vdash$ c and $\langle c, m \rangle \rightarrow \langle c', m' \rangle$

Then: $m \sim m'$

Proof: by induction on the structure of c

(details omitted, try yourself at home)

Corollary: Secret PC*

Given Γ, c, m such that Γ, Secret $\vdash$ c and $\langle c, m \rangle \rightarrow^* \langle c', m' \rangle$

Then: $m \sim m'$

Proof: Using lemma above + preservation

Main lemma: bridge noninterference

Given Γ, pc, c, m, s, N, N' such that $\Gamma, pc \vdash c$ and $m \sim s$ and

$\langle c, m\rangle \curvearrowright^{N} \langle c',m'\rangle$ and $\langle c, s\rangle \curvearrowright^{N'} \langle d',s'\rangle$ Then: $m' \sim s'$ and $c' = d'$

Proof: "two-dimensional" induction. Outer layer: by induction on N; inner layer by induction on the structure of c

Outer base case N = 0.

Inner induction: on the structure of c.

$$c ::= \text{skip} \mid x := e \mid c; c \mid \sout{\text{if } e \text{ then } e \text{ else } e \mid \text{while } e \text{ do } e}$$

Impossible with N=0

Base cases

Inductive case

Case c is skip; consider the possibilities for the m-run

$$\frac{\langle \text{skip}, m\rangle \to \langle \text{stop}, m\rangle \quad m \sim m}{\langle \text{skip}, m\rangle \curvearrowright^{0} \langle \text{stop}, m\rangle}$$ Bridge-Stop

Not applicable

$$\frac{\langle \text{skip}, m\rangle \to \langle \text{stop}, m\rangle \quad m \not\sim m}{\langle \text{skip}, m\rangle \curvearrowright^{0} \langle \text{stop}, m\rangle}$$ Bridge-Public

For the s-run, because c=skip, the only possibility is that N'=0, also only via [Bridge-Stop], and then we can conclude that $\langle \text{skip}, s\rangle \curvearrowright^{0} \langle \text{stop}, s\rangle$

Main lemma: bridge noninterference

Given Γ, pc, c, m, s, N, N' such that $\Gamma, pc \vdash c$ and $m \sim s$ and

$\langle c, m\rangle \rightsquigarrow^{N} \langle c', m'\rangle$ and $\langle c, s\rangle \rightsquigarrow^{N'} \langle d', s'\rangle$ Then: $m' \sim s'$ and $c' = d'$

Outer base case N = 0. Inner induction: on the structure of c.

Case c is x := e. There are two possibilities

$$\frac{\langle x{:=}e,\ m\rangle \rightarrow \langle stop, m'\rangle \quad m \sim m'}{\langle x{:=}e,\ m\rangle \rightsquigarrow^{0} \langle stop, m'\rangle}\ \text{Bridge-Stop} \qquad \frac{\langle x{:=}e,\ m\rangle \rightarrow \langle stop, m'\rangle \quad m \nsim m'}{\langle x{:=}e,\ m\rangle \rightsquigarrow^{0} \langle stop, m'\rangle}\ \text{Bridge-Public}$$

In either case, it must be that $\langle x := e, m\rangle \rightarrow \langle stop, m[x \mapsto v]\rangle$ s.t. $\langle e, m\rangle \Downarrow v$

For the s-run, because c is x:=e, the only possibility is that N'=0, and it must also be that $\langle x{:=}e, s\rangle \rightarrow \langle stop, s'[x \mapsto u]\rangle$, s.t. $\langle e, s\rangle \Downarrow u$

1) Consider [Bridge-Public] first. From $m \nsim m'$ it must be that $\Gamma(x)$ = public.

Recall the typing rule for assignments: $$\frac{\Gamma \vdash e : \ell \qquad pc \sqcup \ell \sqsubseteq \Gamma(x)}{\Gamma, pc \vdash x := e}$$

$\ell_1 \sqsubseteq \ell_2$ when ℓ_1 is public OR ℓ_2 is secret

$\Gamma(x)$ = public means that pc = public AND ℓ is public

From NI for expressions we get that u = v; hence m' ~ s'

2) Consider [Bridge-Stop]. Two subcases:
   (a) $\Gamma(x)$ = Public, but it must be that m(x) = m'(x) . Proceed as in case 1 above
   (b) $\Gamma(x)$ = Secret. Immediate

Main lemma: bridge noninterference

Given Γ, pc, c, m, s, N, N' such that $\Gamma, pc \vdash c$ and $m \sim s$ and

$\langle c, m\rangle \curvearrowright^{N} \langle c',m'\rangle$ and $\langle c, s\rangle \curvearrowright^{N'} \langle d',s'\rangle$ Then: $m' \sim s'$ and $c' = d'$

Outer base case N = 0. Inner induction: on the structure of c.

Case c is $c_1;c_2$.

In the m-run, because N=0, the only possibility is

$$\frac{\langle c_1;c_2,\ m\rangle \rightarrow \langle c',m'\rangle \qquad m \nsim m'}{\langle c_1;c_2,\ m\rangle \curvearrowright^{0} \langle c',m'\rangle}\ \text{Bridge-Public}$$

Recall the semantics for sequential composition. Two possibilities are

$$\frac{\langle c_1,\ m\rangle \rightarrow \langle c'_1,m'\rangle \qquad c'_1 \neq \text{stop}}{\langle c_1;c_2,\ m\rangle \rightarrow \langle c'_1;c_2',m'\rangle}$$

$$\frac{\langle c_1,\ m\rangle \rightarrow \langle c'_1,r}{\langle c_1;c_2,\ m\rangle \rightarrow \langle c_2',m'\rangle}$$

Done by IH

In either case, it is c1 that performs the public update, so we write

$\langle c_1,\ m\rangle \rightarrow \langle c'_1,m'\rangle$, where c'_1 could be stop

In the s-run, we know less, so more is possible

$$\frac{\langle c_1;c_2,\ s\rangle \rightarrow \langle c',s'\rangle \qquad s \nsim s'}{\langle c_1;c_2,\ s\rangle \curvearrowright^{0} \langle d',s'\rangle}\ \text{Bridge-Public}$$

In this case, it is c1 that performs the public update

$\langle c_1,\ s\rangle \rightarrow \langle d'_1,s'\rangle$, where d'_1 could be stop

Alternatively:

$$\frac{\langle c_1;c_2,\ s\rangle \rightarrow \langle d^*,s^*\rangle \qquad s \sim s^* \qquad d^* \neq \text{stop} \qquad \langle d^*,\ s^*\rangle \curvearrowright^{N'-1} \langle d',s'\rangle}{\langle c_1;c_2,\ s\rangle \curvearrowright^{N'} \langle d',s'\rangle}\ \text{Bridge-Multi}$$

This means, that c_1 does not perform the public update

$\langle c_1,\ s\rangle \rightarrow \langle d^*_1,s^*\rangle$, where d'_1 could be stop

Impossible by IH, because $m' \sim s^* \sim s \sim m \nsim m'$ is a contradiction

Main lemma: bridge noninterference

Given Γ, pc, c, m, s, N, N' such that $\Gamma, pc \vdash c$ and $m \sim s$ and

$\langle c, m \rangle \curvearrowright^{N} \langle c', m' \rangle$ and $\langle c, s \rangle \curvearrowright^{N'} \langle d', s' \rangle$ Then: $m' \sim s'$ and $c' = d'$

Proof: "two-dimensional" induction. Outer layer: by induction on N; inner layer by induction on the structure of c

Outer inductive case N > 0.

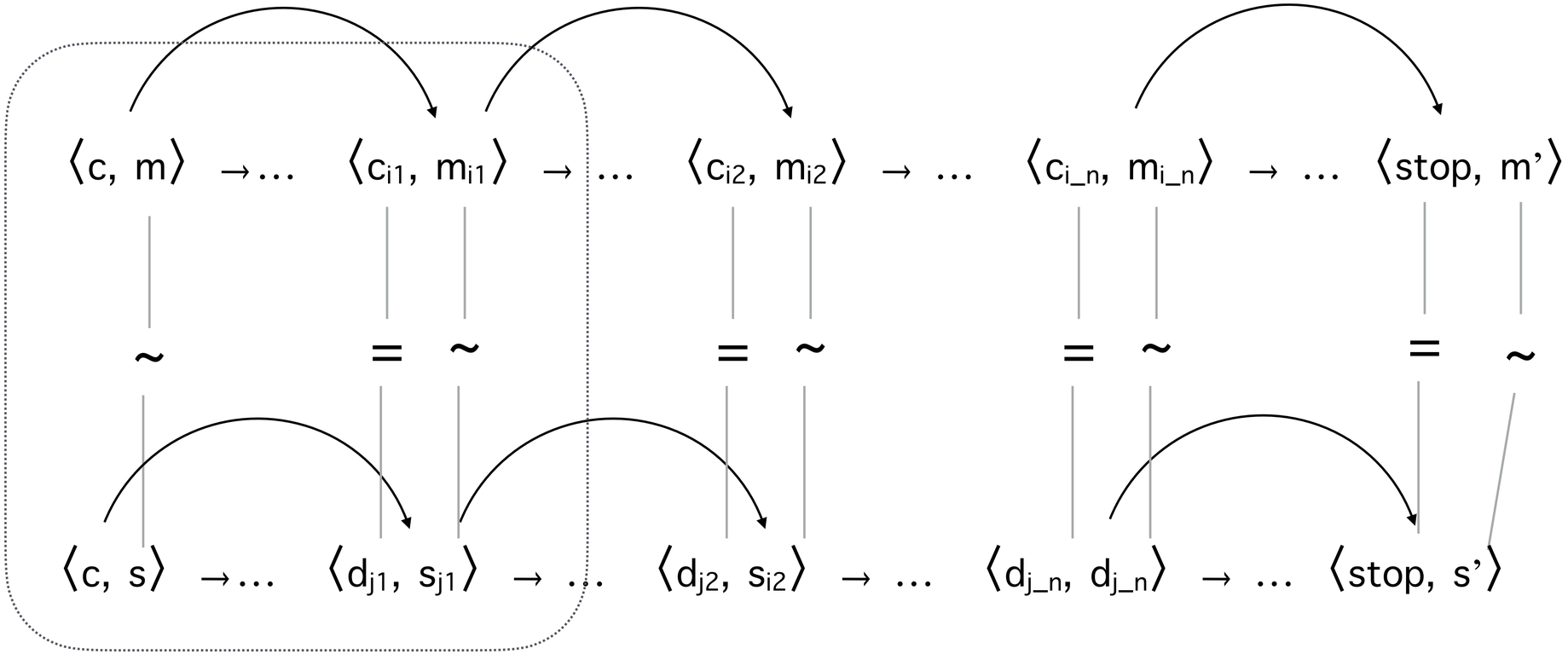
Inner induction: on the structure of c.

$$c ::= \sout{\text{skip} \mid x := e} \mid c; c \mid \text{if } e \text{ then } c \text{ else } c \mid \text{while } e \text{ do } c$$

Impossible with N > 0

- Case $c_1;c_2$ – similar to the one we considered for N=0; but here we have 4 cases instead of 2 to apply the IH for c1. We also now have the possibility of when c1 does not update public var; there we also apply the IH to c2.
- Case if e then c_1 else c_2. Two cases depending on the level of pc $\sqcup \ell$, where $\Gamma \vdash e: \ell$
  - If pc $\sqcup \ell$ = public, then both runs take the same branch; Done by either of the IH
  - If pc $\sqcup \ell$ = secret, then by Lemma Secret PC*, public memories are not updated in either of the runs
- Case while e do c. Done by the outer IH

Intuition for the proof



Proof: “triple-nested” induction

- by induction on the number of bridges in the first run
  - by induction on the number of intermediate steps within each bridge
    - by induction on the structure of the commands making the steps

Lemma: Bridge NI

Soundness

Theorem (VSI'96):

Given an environment Γ, a program c such that $\Gamma, pc \vdash c$ then it must be that

for all memories m, s, such that $m \sim_\Gamma s$

if $\langle c, m\rangle \rightarrow^* \langle stop, m'\rangle$

and $\langle c, s\rangle \rightarrow^* \langle stop, s'\rangle$

then $m' \sim_\Gamma s'$

Enforcement via a monitor

Designing the mechanism

- Variant 1:
  - crash on all direct assignments
  - what we need: remember which values depend on secrets
    - need to update our $\Downarrow$ relation for expressions

$$\langle e, m \rangle \Downarrow \langle v, \ell \rangle \quad \text{where } \ell \text{ is the level}$$

    - and also update the rule for assignments: instead of one old rule, we have now 2 rules that discriminate based on the level of the expression

$$\frac{\langle e, m \rangle \Downarrow \langle v, \text{public} \rangle \quad \text{“x is public or secret”}}{\langle x := e, m \rangle \rightarrow \langle \text{stop}, m[x \mapsto v] \rangle}$$

$$\frac{\langle e, m \rangle \Downarrow \langle v, \text{secret} \rangle \qquad \text{“x is secret”}}{\langle x := e, m \rangle \rightarrow \langle \text{stop}, m[x \mapsto v] \rangle}$$

Q: do you see any issues with this?

Hint: construct a program that leaks but is still undetected

Designing the mechanism

- Variant 2:
  - introduce a program counter level
  - raise the level on branches
-

Q: do you see any issues with this?

Hint: think of good programs that will be rejected

Designing the mechanism

- Variant 2:
  - introduce a program counter level
  - raise the level on branches

Q: do you see any issues with this?

Hint: think of good programs that will be rejected

Designing the mechanism

- Variant 3
  - introduce a stack program counter levels
  - push a level on branches
  - pop on join points

Hint: we now need to distinguish join points…

Designing the mechanism

- Pros and cons of the dynamic mechanism?
  - Simple (+)
  - Sound** (+)
  - Not very precise (-)
  - Slows program execution (-)
  - Results in crashes (-)
- Option B:
  - we can do most of this *statically*

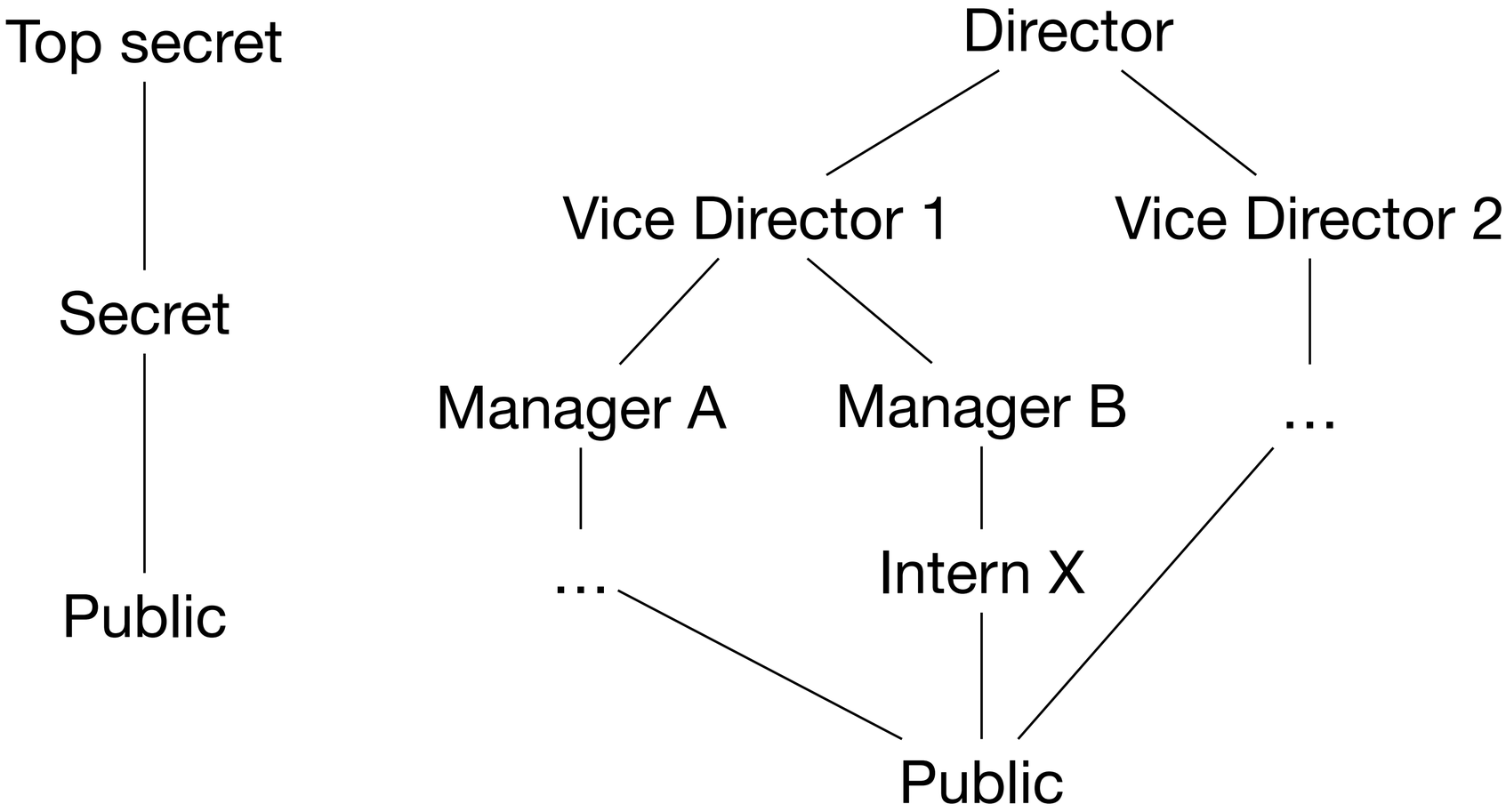
Limitations of our model

- The language is trivial
  - Functions/Exceptions/Objects/IO
    - What changes for the semantic policy?
    - What changes for the enforcement?
- Precision of the enforcement
  - Many secure programs are marked as false positives
-

Label models

Beyond public/secret

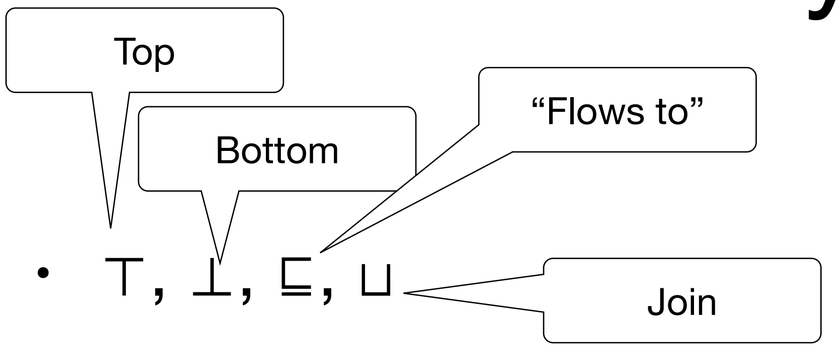
- In general we want to consider arbitrary security classes, not just public/secret



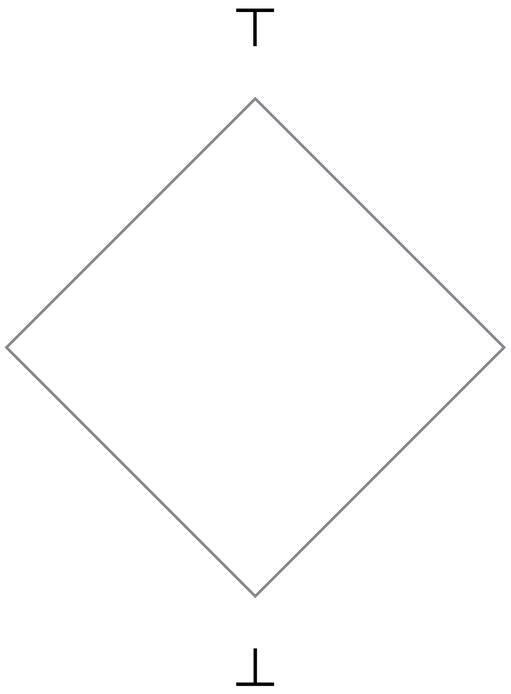
Desired properties of security classes

- Assume we have a set of security classes $\{\ell_1, \ell_2 \ldots \ell_n\}$
- When can information at one class propagate to another one?
  - Information at the security class of ViceDirector can propagate to the security class of Director, but not the other way around
  - This assumes a partial order between classes, denoted as $\ell_i \sqsubseteq \ell_j$
- If we combine information from two different security classes ℓ_1 and ℓ_2 , at what security class should that information be?
  - Can't always pick ℓ_1 or ℓ_2 because it may be that $\neg (\ell_2 \sqsubseteq \ell_1)$ and $\neg(\ell_1 \sqsubseteq \ell_2)$
  - We need to be able to pick ℓ_3 such that $\ell_1 \sqsubseteq \ell_3$ and $\ell_2 \sqsubseteq \ell_3$, we denote that as $\ell_3 = \ell_1 \sqcup \ell_2$.
  - Also: nice if ℓ_3 is the least upper bound of ℓ_1 and ℓ_2;
    - Consider some ℓ_4, such that $\neg (\ell_4 \sqsubseteq \ell_3)$ but for which it also holds that $\ell_4 = \ell_1 \sqcup \ell_2$. If we pick ℓ_4 as our level then the combined information can no longer flow to ℓ_3
- These properties lead to natural structure of security classes as *lattices.*

Security lattices



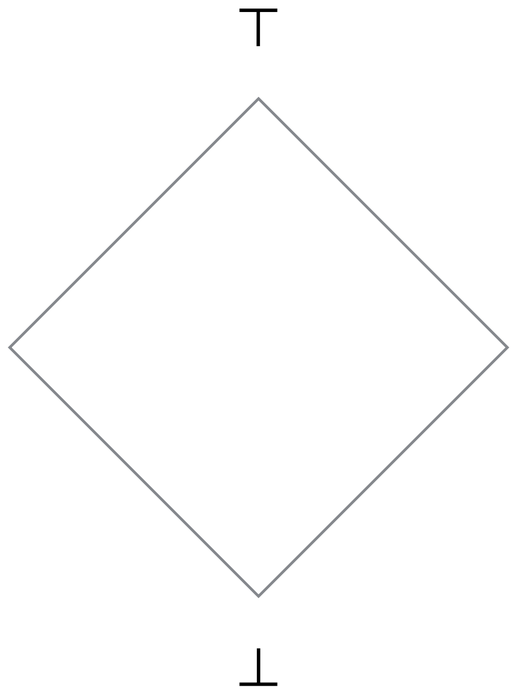
Higher levels on the lattice are more restrictive



Information flow: preventing flows from high points on the lattice to low ones

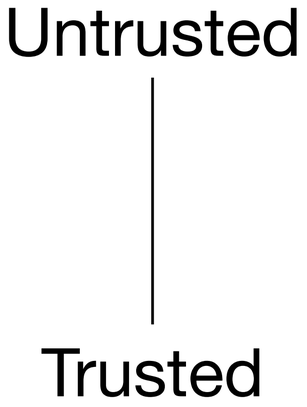
Declassification

- Allowing *some* information flows from secret to public
- Corresponds to intended information flows
  - *x = declassify (e)*
- Important to not misuse:
  - subject to code inspection
    - what is declassified/where/by who/ when



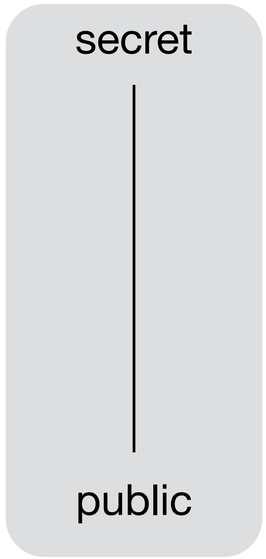
Integrity

- Trusted vs Untrusted
- Untrusted is higher on the lattice because we want to prevent trusted output being updated by untrusted input
- *Duality* of integrity and confidentiality
  - Noninterference for integrity can be formulated similar to noninterference for confidentiality

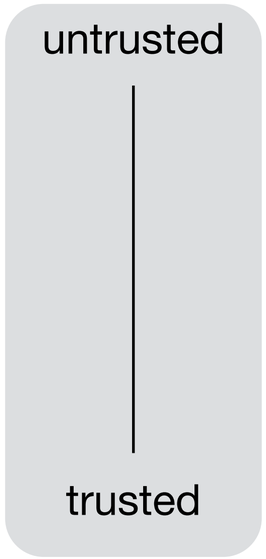


Combining integrity and confidentiality

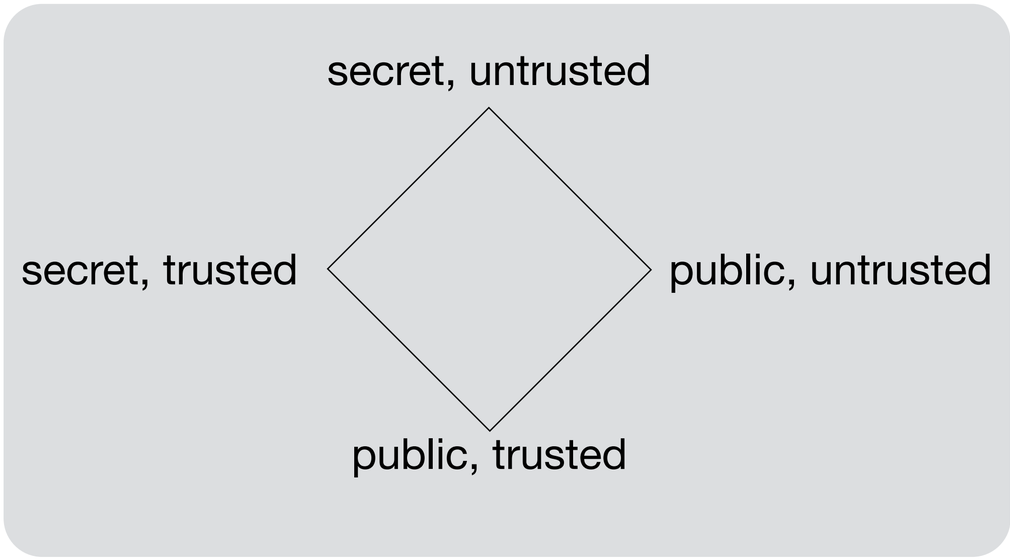
Intuition: the higher is the level the lattice, the more restrictive the level, i.e., information from that level must not flow to the lower lattice levels



Lattice for confidentiality



Lattice for integrity



Combined integrity and confidentiality lattice

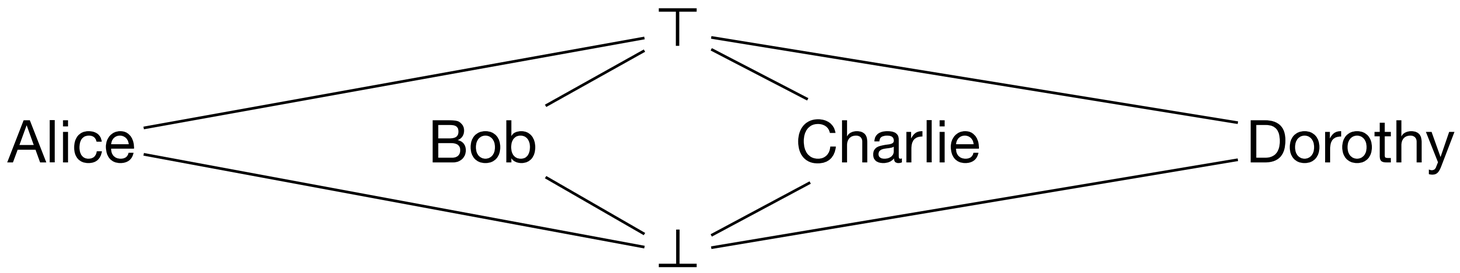
Decentralized Label Model

- Principals: `Alice, Bob, server`
- Principals form an *actsfor* hierarchy
  - Top principal ⊤; Bottom principal ⊥
- Owned policies
  - Confidentiality: `{Alice → Bob, Charlie}`
  - Integrity: `{Alice ← Bob}`
- Labels consist of a set of policies:
  `{Alice → Bob, Charlie; Alice → Bob; Alice ← Bob}`
- Best viewed as statements uttered by individual principals
  - Different from “access control” policies
  - Decentralized means that a policy can be ignored if the owner is not trusted
- “Joining” restrictions: intersection of readers, union of writers.

Alice says that Bob and Charlie may read

Alice says that Bob may write

Example principal hierarchy



Q: how should these labels be ordered?

Alice → Bob

Bob → Alice

Alice → ⊥; Bob → Alice

Alice → Charlie, Dorothy

Alice → Dorothy

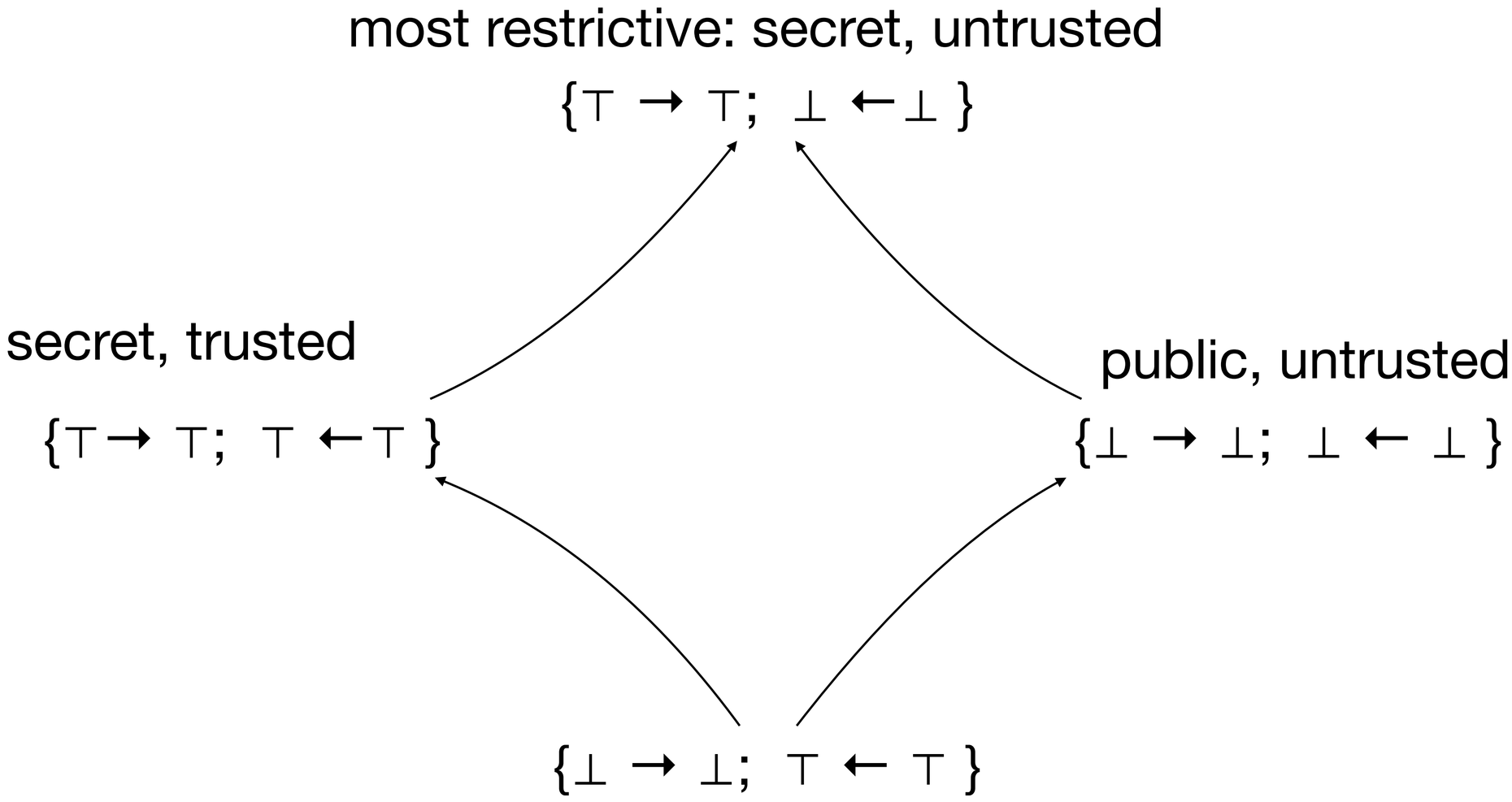
⊥ → ⊥; ⊤ ← ⊤

⊤ → ⊤; ⊤ ← ⊤

⊥ → ⊥; ⊥ ← ⊥

⊤ → ⊤; ⊥ ← ⊥

Label hierarchy



Nondeterminism

- Suppose we have some behavior in the language that is underspecified.
  - This is sometimes modeled with a nondeterministic assignment `x := nondet()`
  - Semantics $\langle \texttt{x := nondet(), m} \rangle \rightarrow \langle \texttt{stop, m [x} \mapsto \texttt{v]} \rangle$ where `v` could be any value
- This presents two problems:
  - Our earlier definition of security is too strict for this semantics
    - Q: how to weaken the definition?
  - This creates possibilities for *refinement* attacks if the adversary knows how the nondeterminism is resolved

Source of nondeterminism in language-specification

- Undefined behavior in C/C++
- Scheduling order in concurrent languages

Saltzer & Schroeder's design principles, 1975

Economy of mechanism	*simpler mechanisms are easier to verify*
Fail-safe defaults	*do not give access by default*
Complete mediation	*do not trade performance for security*
Open design	*no security through obscurity*
Separation of privilege	*minimize damage of a single accident, e.g, multi-factor auth*
Least privilege	*need to know, e.g., capability machines*
Least common mechanism	*sharing leads to security problems*
Psychological acceptability	*there are humans in the loop, e.g,. passwords not ideal*

Least Common Mechanism

Minimize the amount of mechanism common to more than one user and depended on by all users. Every shared mechanism (especially one involving shared variables) represents a potential information path between users and must be designed with great care to be sure it does not unintentionally compromise security.

Further, any mechanism serving all users must be certified to the satisfaction of every user, a job presumably harder than satisfying only one or a few users.

Structured Programming Language
Compiler
User-level Instruction Set Architecture
Operating system
Privileged Instruction Set Architecture
Processor
Hardware description language

Troupe program example

```
if secret                  ||    skip x 50                ||    send (public, 0)
then                             send (public, 1)
   skip x 100
else
   skip x 20
```

Ok under a round-robin scheduler

Thread 1 *Thread 2* *Thread 3*

Troupe scheduling bug

N_steps_per_thread = 60

```
if secret
then
   skip x 100
else
   skip x 20
```

||

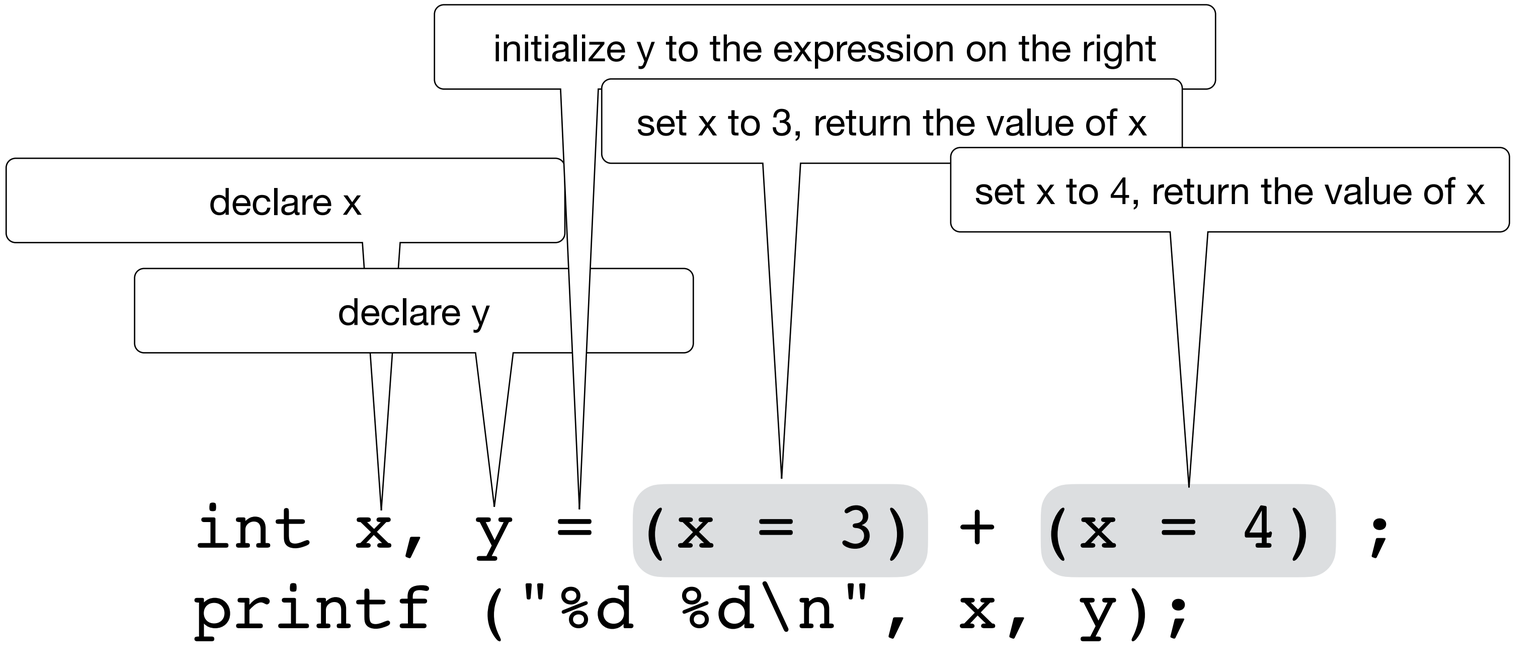
```
skip x 50
send (public, 1)
```

||

```
send (public, 0)
```

Thread 1 *Thread 2* *Thread 3*

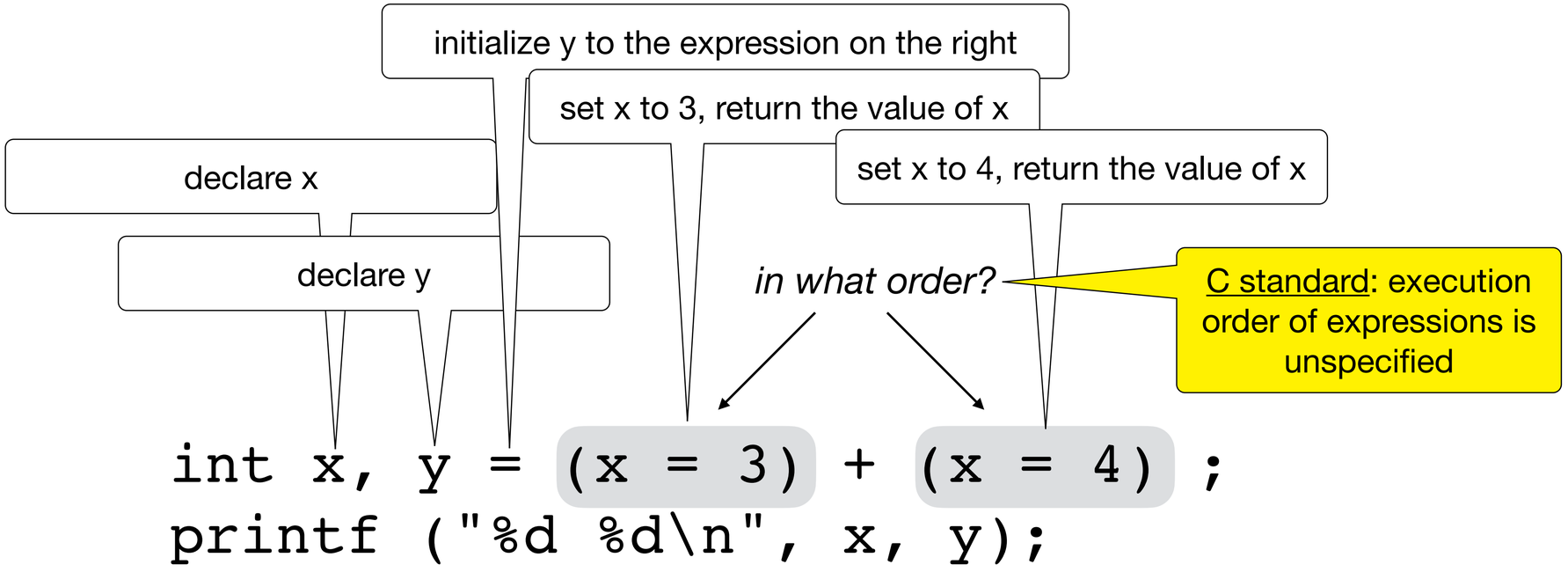
Undefined behavior in C



Q: what do you expect this program to print?

example from Robbert Krebbers' PhD thesis

Undefined behavior in C



C standard: an object can be modified at most ***once*** in an execution of an expression; because x is modified twice, this example exhibits undefined behavior…

C standard: undefined behavior ⇒ anything is allowed

example from Robbert Krebbers' PhD thesis

```
$ cat example.c
#include <stdio.h>

int main () {
    int x,y = (x = 3) + (x = 4) ;
    printf ("%d %d\n", x, y);
    return 0;
}


$ gcc -o2 example.c -o versionA

$ ./versionA
4 8

$ clang -o2 example.c -o versionB
example.c:4:18: warning: multiple unsequenced modifications to 'x' [-Wunsequenced]
    int x,y = (x = 3) + (x = 4) ;
                 ^         ~
1 warning generated.

$ ./versionB
4 7

$ gcc --version
gcc (Ubuntu 5.4.0-6ubuntu1~16.04.9) 5.4.0 20160609
Copyright (C) 2015 Free Software Foundation, Inc.
This is free software; see the source for copying conditions.  There is NO
warranty; not even for MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.


$ clang --version
clang version 3.8.0-2ubuntu4 (tags/RELEASE_380/final)
Target: x86_64-pc-linux-gnu
Thread model: posix
InstalledDir: /usr/bin

$
```