University of Ljubljana
Faculty of Computer and Information Science

BEYOND SEMICONDUCTOR

Information Flow Tracking

European Network for Cybersecurity
PhD Winter School 2024

Jurij Mihelič
Faculty of Computer and Information Science
University of Ljubljana
&
Beyond Semiconductor d.o.o.
Slovenia

Contents

- Cybersecurity
- Information flow
- Confidentiality and integrity
  - Bell-LaPadula and Biba models
- Information flow policy
  - Formalizing tag propagation
- Non-interference
- Language-based IFT
- Information downgrading
- Separation kernel formalization

Cybersecurity

InfoSec – information security

practice of protecting sensitive information and critical systems

CyberSec – cyber security

InfoSec related to computer systems and data

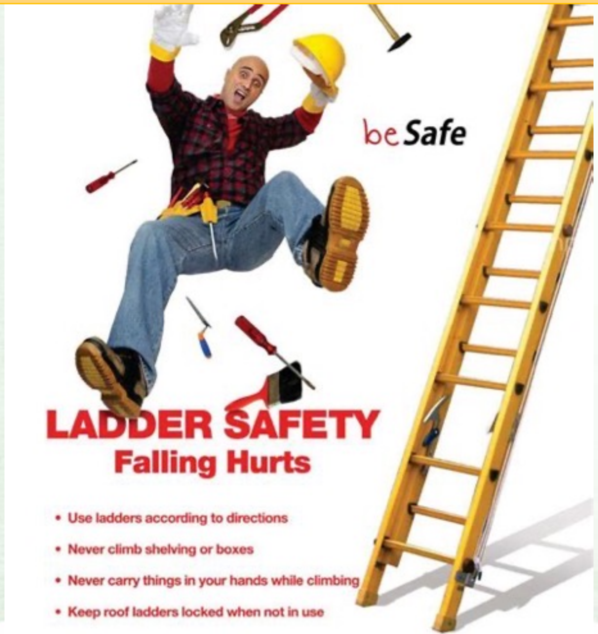
With the goal to prevent/reduce the likeliness of unauthorized/inappropriate access to data such as

unlawful use, disclosure, disruption
deletion, corruption, modification, inspection
recording, devaluation etc.

Cybersecurity

A **threat** is a **potential negative action or event** facilitated by a **vulnerability** that results in an **unwanted impact** on a computer system or application.

Accidental negative events
natural disasters, fires, tornados, radiation, malfunctioning



Intentional negative events
adversary attacks, criminal, hacking

Cybersecurity

- Certification: Common Criteria, CC
  - ISO/IEC 15408 standard
  - Common Criteria for Information Technology Security Evaluation
  - product evaluation criteria

EAL – Evaluation Assurance Levels

EAL1: Functionality Tested
EAL2: Structurally Tested
EAL3: Methodically Tested and Checked
EAL4: Methodically, Designed, Tested and Reviewed
EAL5: Semiformally Designed and Tested
EAL6: Semiformally Verified Design and Tested
EAL7: Formally Verified Designed and Tested

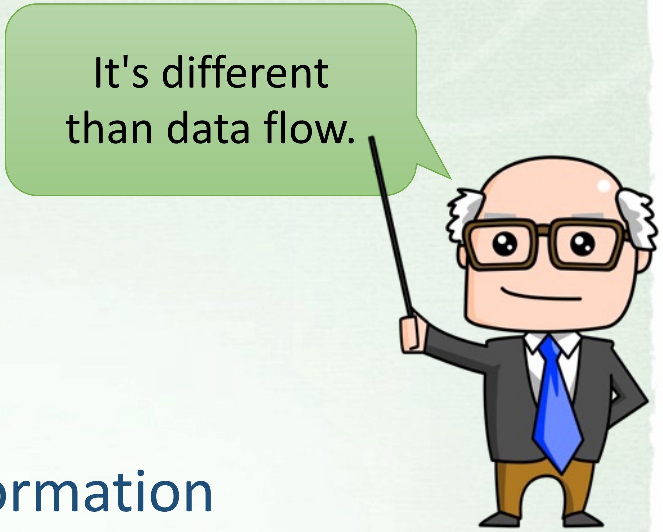
Cybersecurity

- Formal methods

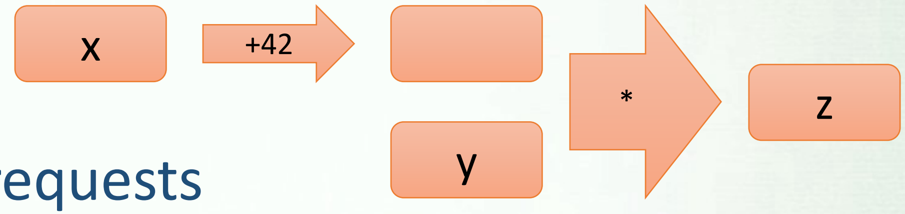
```
subsubsection ‹Interference relation›

abbreviation arc_in :: "'a policy ⇒ 'a ⇒ 'a ⇒ bool" ("(_: _ → _)" 70) where
  "arc_in p a b ≡ (a, b) ∈ arcs p"

hide_const (open) arc_in

fun flow_in :: "'a policy ⇒ 'a list ⇒ bool" where
  "flow_in _ [] = False" |
  "flow_in _ [_] = False" |
  "flow_in p (a#b#[]) = (p: a → b)" |
  "flow_in p (a#b#w) = ((p: a → b) ∧ flow_in p (b#w))"

definition flow_in' :: "'a policy ⇒ 'a list ⇒ bool" where
  "flow_in' p w ≡ length w ≥ 2 ∧ (∀ i < length w - 1 . (p: w!i → w!(i+1)))"

definition reachable_in :: "'a policy ⇒ 'a ⇒ 'a ⇒ bool" ("(_: _ ⇝ _)" 70) where
  "reachable_in p a b ≡ (∃ w . a = hd w ∧ b = last w ∧ flow_in p w)"
```

```
subsection ‹Non-exfiltration›

text ‹Non-reachable tags cannot be in outs of the last step›

definition non_exfiltration :: "'a policy ⇒ 'a step list ⇒ bool" where
  "non_exfiltration p w ≡  (w = [] ∨ (∀ a b . a ∈ ins (hd w) ∧ (¬ (p: a ⇝ b)) ⟶ b ∉ outs (las

lemma preservance_gives_non_exfiltration:
  shows "preservance p w ⟶ non_exfiltration p w"
  unfolding preservance_def non_exfiltration_def
  by blast

corollary non_exfiltration:
  assumes "valid_policy p"
    and "∀ u . restricted_step p u"
  shows "walk w ⟶ non_exfiltration p w"
  using assms
  using walks_are_restricted preservance_1 preservance_gives_non_exfiltration
  by blast
```

Information flow security



- Information flow
  - transfer of **information**
  - from a **source** to a **destination**
    - a passive entity that contains information
    - e.g., variable, record, object, file, memory or storage location
  - by a **subject**
    - an active entity that requests access to an object
    - e.g., user, process
  - during an information processing **activity**
    - ability of a subject to perform a task or interact with an object
    - e.g., operation, program statement, machine instruction



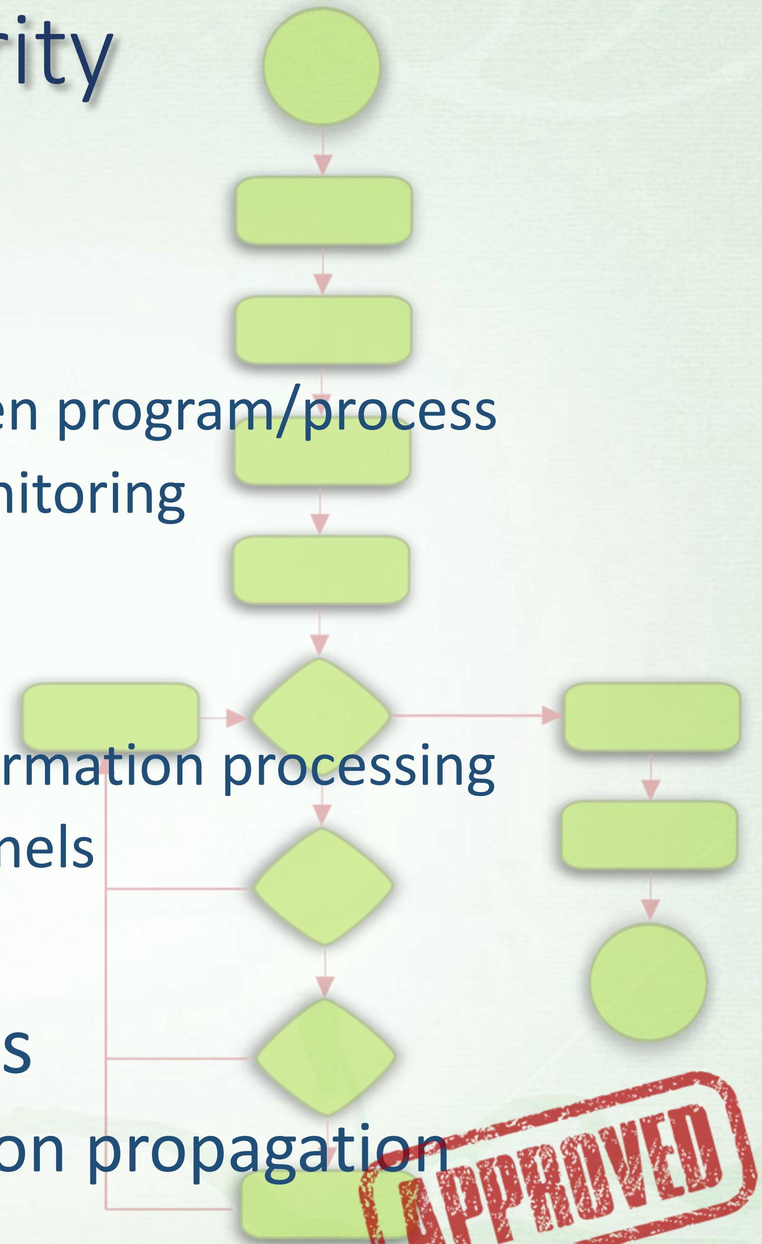
Information flow security

- **Desirable** vs. **undesirable** information flow
  - depends on the property/application

  - **confidentiality**
    - data can be **read by authorized** users and is not disclosed to unauthorized users
    - *secret data does not leak to a public place*
    - read protection
  - **integrity**
    - data can be **changed by authorized** users and cannot be altered by unauthorized users
    - *trusted data is not influenced by dubious data*
    - write protection

Information flow security

- Information flow **tracking**
  - **analysis and monitoring**
    - determine the flow in a given program/process
    - static analysis, dynamic monitoring
  - **control**
    - limiting the flow during information processing
    - firewalls, ACLs, secure channels
- Guarantees and assurances
  - properties about information propagation



Information flow security

- Perfect security is hard

Confidentiality

- Two-level confidentiality

low level: public data
- insensitive data
- may be publicly observed

high level: private data
- secret data
- may not be publicly observed

- Multiple levels
  - MLS – Mulitple Levels of Security
  - EU classified information

level	the unauthorised disclosure of this information could
EU Top secret	cause exceptionally grave prejudice to
EU Secret	seriously harm
EU Confidential	harm
EU Restricted	be disadvantageous to
	the essential interests of the EU or one or more of the member states

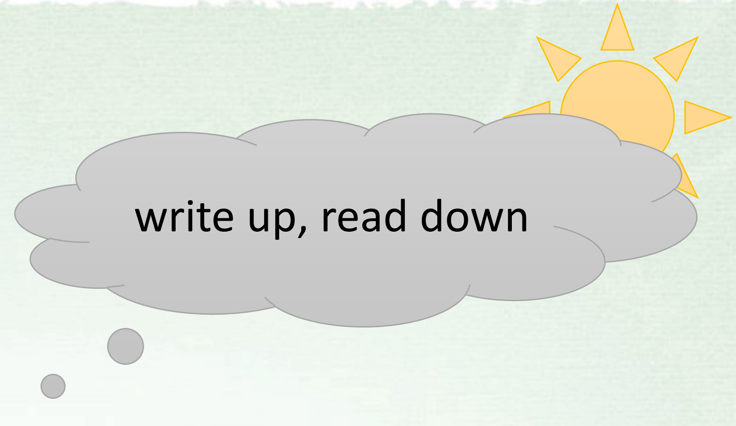
Confidentiality

- Bell-LaPadula model
  - defined by the US DoD to formalize a MLS policy
  - a state transition model of security policy
  - **security labels** on **objects**
  - **clearance levels** for **subjects**

Top secret
Secret
Confidential
Unclassified

  - subjects access objects
    - each state transition preserves a secure state
    - two MAC rules
    - one DAC rule (specified with an access matrix)

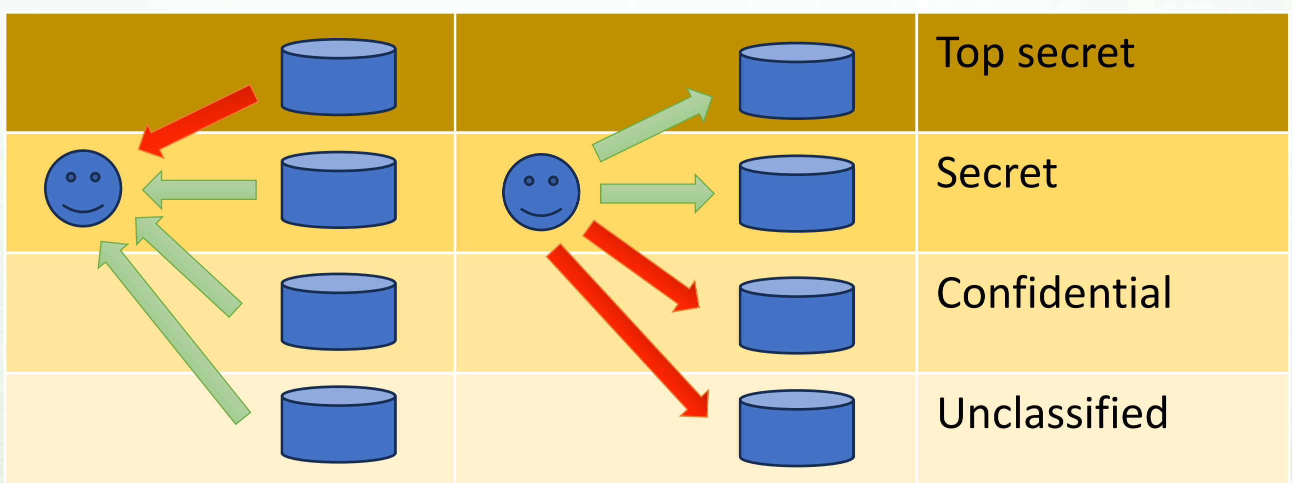
Confidentiality



- Bell-LaPadula model
  - two MAC rules

Simple Security Property
read down / no read up

Star Property
write up / no write-down



Confidentiality

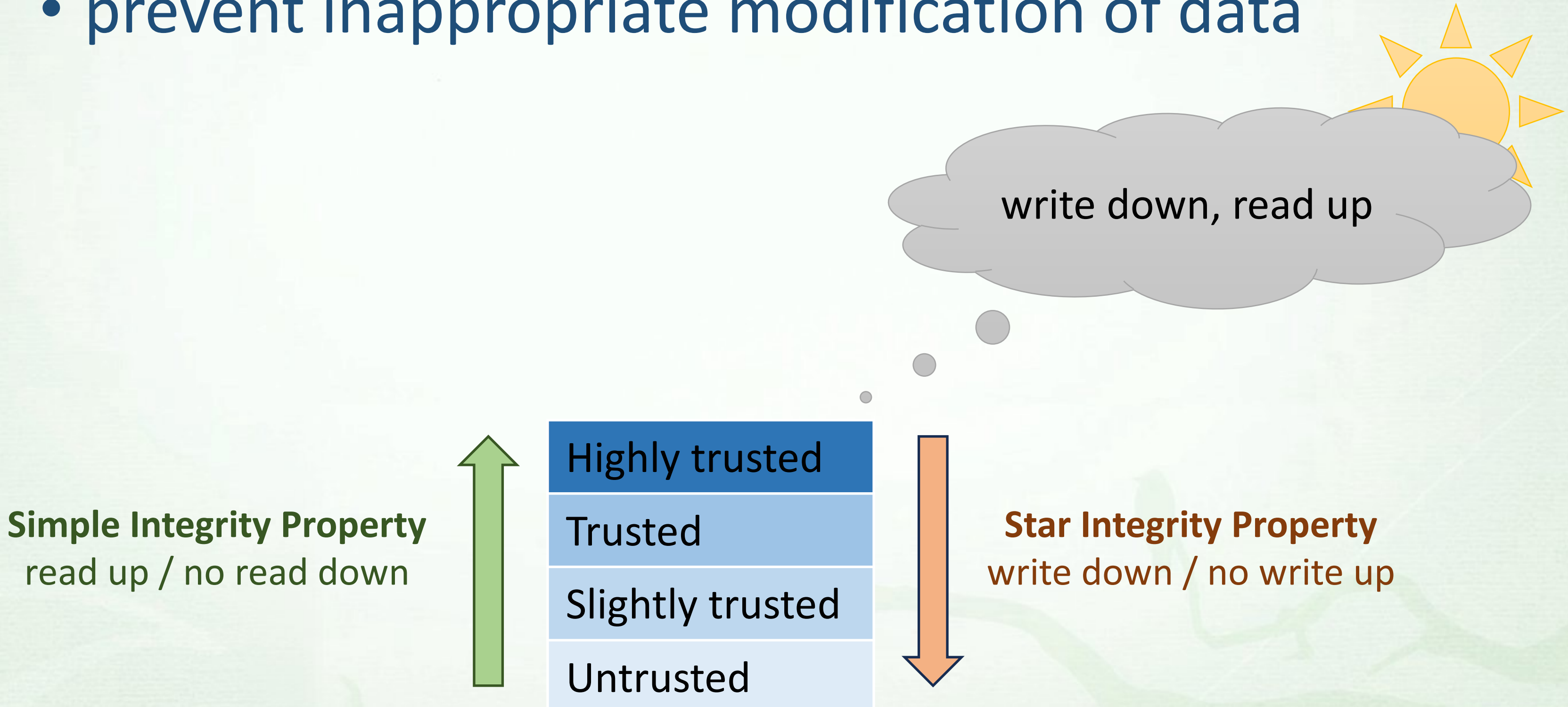
- Bell-LaPadula model
  - **Strong Star Property**
    - subject can write objects only to the same level
    - motivated by the **integrity** concerns
  - **Trusted Subjects**
    - can **downgrade** the information: high to low transfer
    - are not restricted to the Star Property
  - **Principle of Tranquility**
    - the security level of an object or subject may never change while it is being referenced

Integrity

- Two-level integrity
  - high level: trusted data
  - low level: dubious data
  - information flow policy
    - low to low, high to high, **high to low**
    - but **low to high** is prohibited

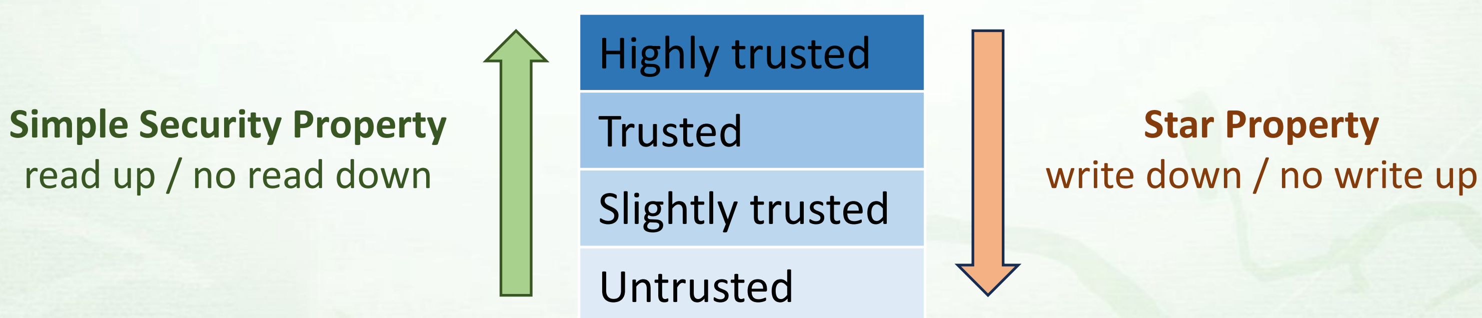
Integrity

- Biba model
  - objects and subjects are classified by **integrity** levels
  - prevent inappropriate modification of data



Integrity

- Bell-LaPadula and Biba models duality



Information flow policy

Information flow policy

A set of rules specifying directions between entities in which the information may flow or must not flow.

- entities
  - subjects: process, person
  - objects: file, memory page, variable
  - tags, labels: data classifications
  - actions: read, write, computation

Information flow policy

- Definition

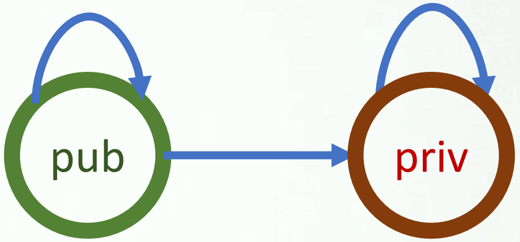
$\mathcal{P} = (T, \rightsquigarrow)$

- a set T of entities (labels, tags)
  - specifying security classes
- a binary relation $\rightsquigarrow$ over T
  - a set of ordered pairs: $\rightsquigarrow \subseteq T \times T$
  - specifying allowed flow between entities
- a negation of $\rightsquigarrow$
  - $x \not\rightsquigarrow y \equiv \neg(x \rightsquigarrow y)$

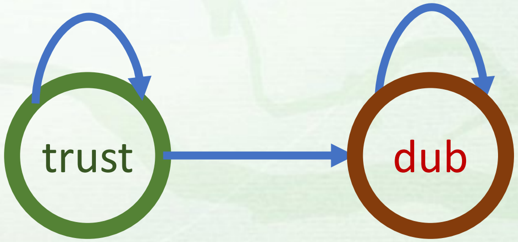
Information flow policy

- Confidentiality
  - $T = \{ \text{pub}, \text{priv} \}$
  - $\leadsto = \{ \text{pub} \leadsto \text{pub}, \text{priv} \leadsto \text{priv}, \text{pub} \leadsto \text{priv} \}$

$x \leadsto y$	pub	priv
pub	1	1
priv	0	1

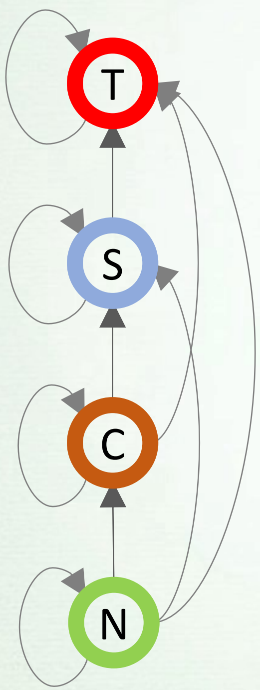


- Integrity
  - $t \leadsto t, d \leadsto d, t \leadsto d$



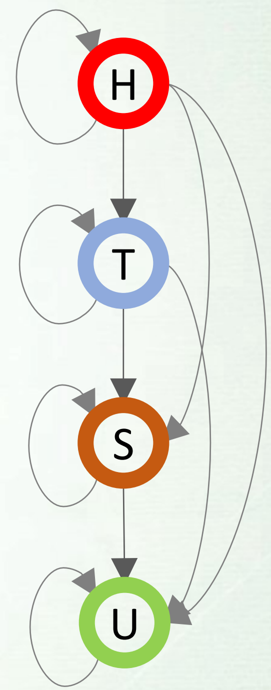
Information flow policy

- Confidentiality and integrity



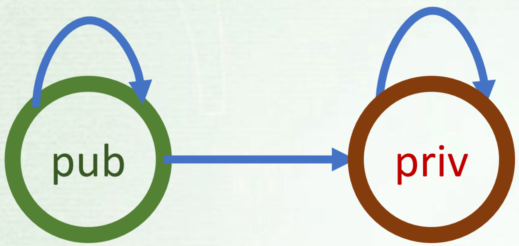
Top secret
Secret
Confidential
Unclassified

Highly trusted
Trusted
Slightly trusted
Untrusted



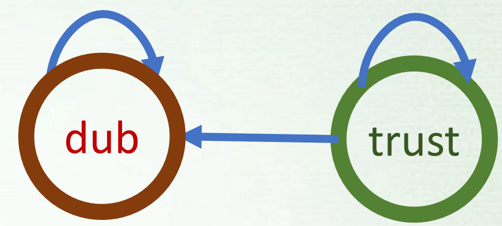
Information flow policy

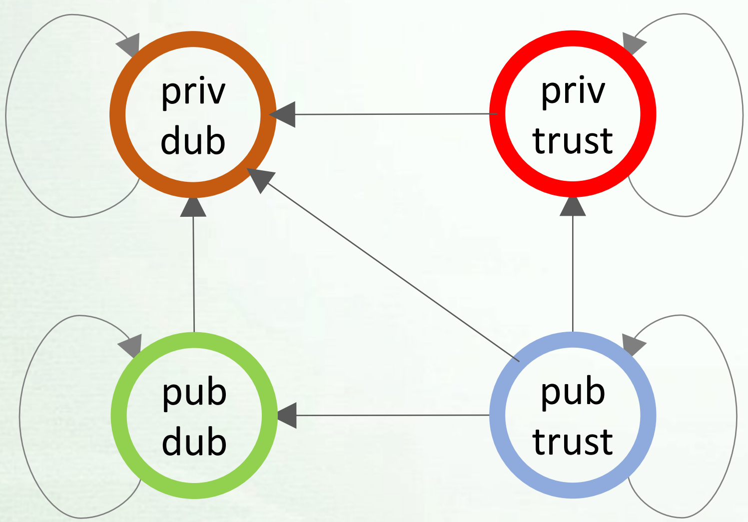
- Confidentiality and integrity combined



Private
Public

Trusted
Dubious

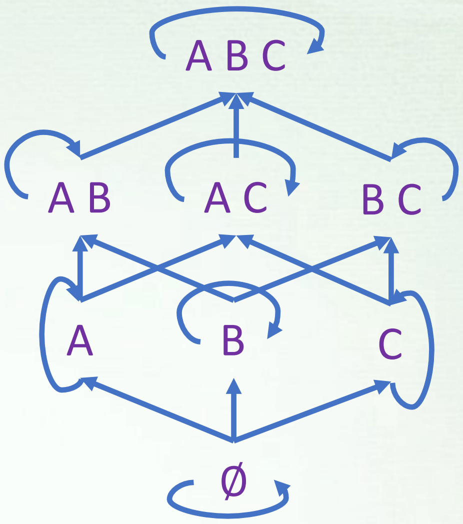




	Dubious	**Trusted**
Private	priv dub	priv trust
Public	pub dub	pub trust

Information flow policy

- Non-linear policies
    - Cartesian product
    - subset of permissions



- Timing
    - constant/variable time operations
- Tracking different sources
    - keyboard, mouse, GPS, camera

Information flow policy

- Properties of relations

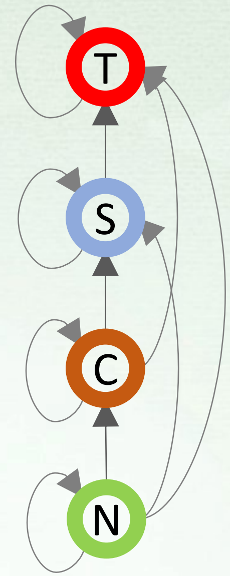
x to x, $\forall x$:
- reflexive: $x \rightsquigarrow x$
- irreflexive: $\neg(x \rightsquigarrow x)$

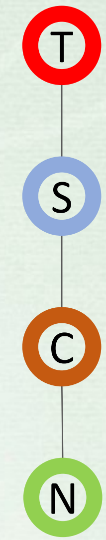
x to y, $\forall x, y$:
- connected: $x \neq y \Longrightarrow x \rightsquigarrow y \vee y \rightsquigarrow x$
- strongly connected: connected + reflexive

x to y vs y to x, $\forall x, y$:
- symmetric: $x \rightsquigarrow y \Longrightarrow y \rightsquigarrow x$
- asymmetric: $x \rightsquigarrow y \Longrightarrow \neg(y \rightsquigarrow x)$
- antisymmetric: $x \rightsquigarrow y \wedge y \rightsquigarrow x \Longrightarrow x = y$

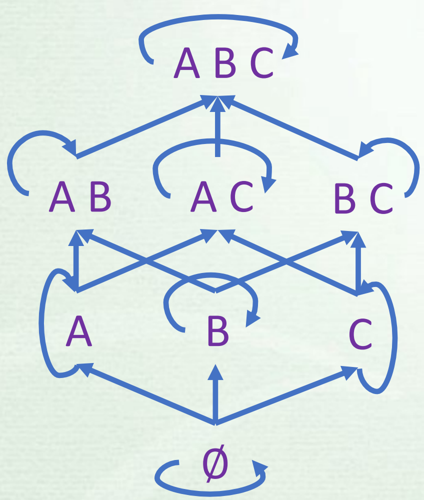
x, y, and z, , $\forall x, y, z$:
- transitive: $x \rightsquigarrow y \wedge y \rightsquigarrow z \Longrightarrow x \rightsquigarrow z$

Information flow policy

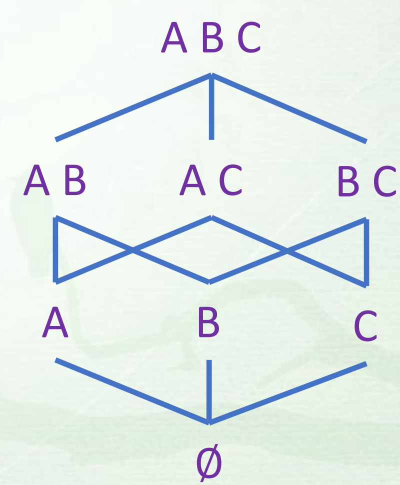




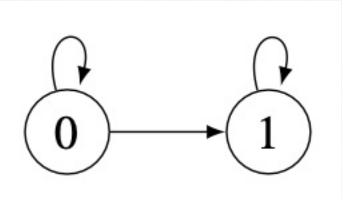
- Properties of relations
  - **partially ordered set** (POS)
    - **reflexive**, **transitive**, antisymmetric
  - **universally bounded lattice** $(S, \rightsquigarrow, \bot, T, \oplus, \otimes)$
    - POS + supremum/join and infimum/meet

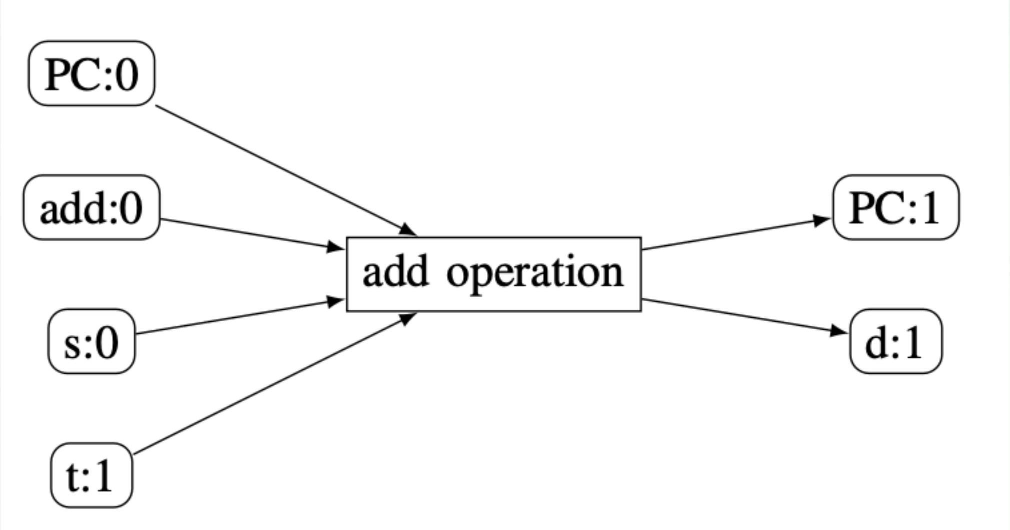


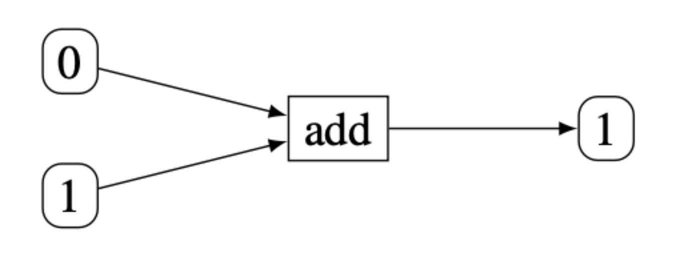
- $S = \{ABC, AB, AC, BC, A, B, C, \emptyset\}$
- $\rightsquigarrow$ = see the figure
- $\bot = \emptyset$
- T = ABC
- $\oplus = \cup$
- $\otimes = \cap$

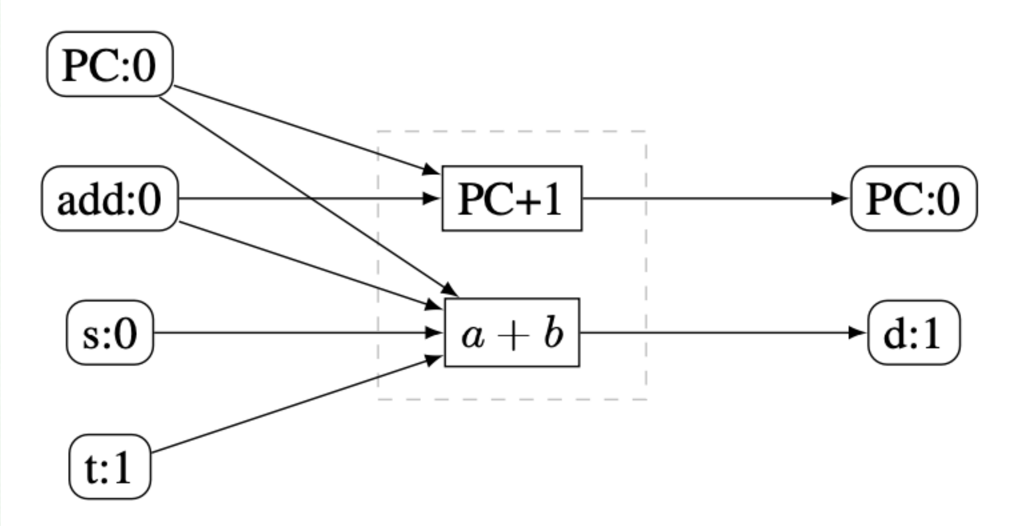


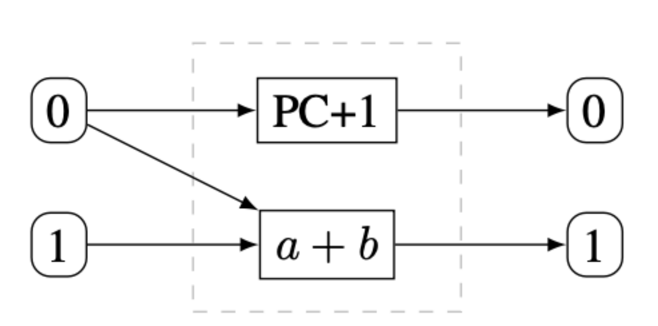
Information flow policy



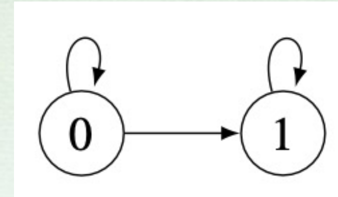


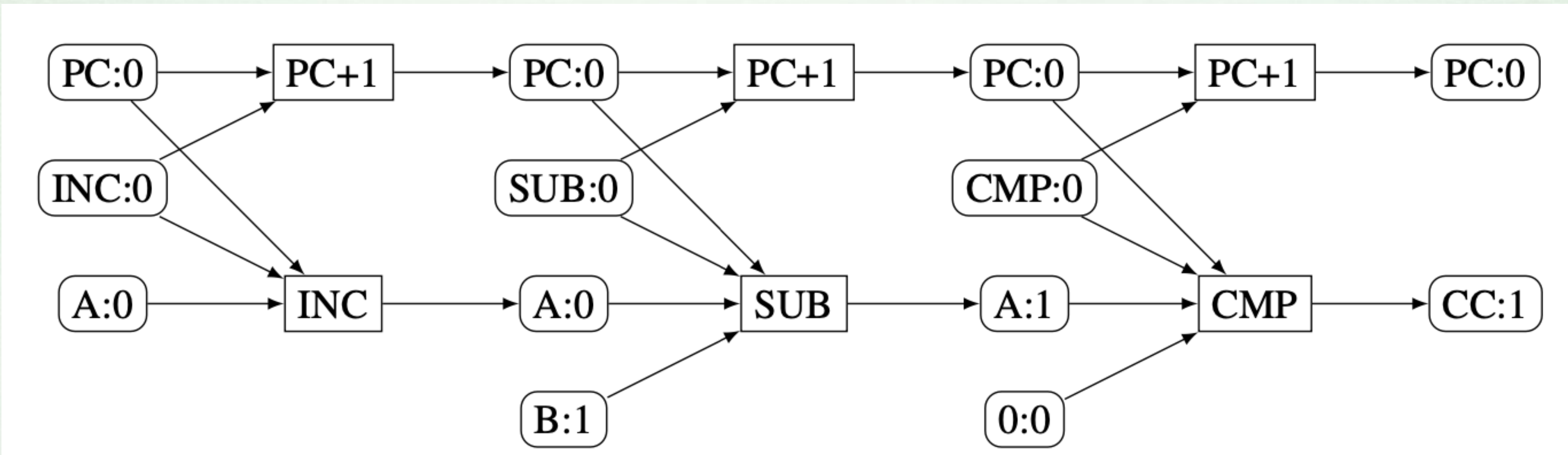


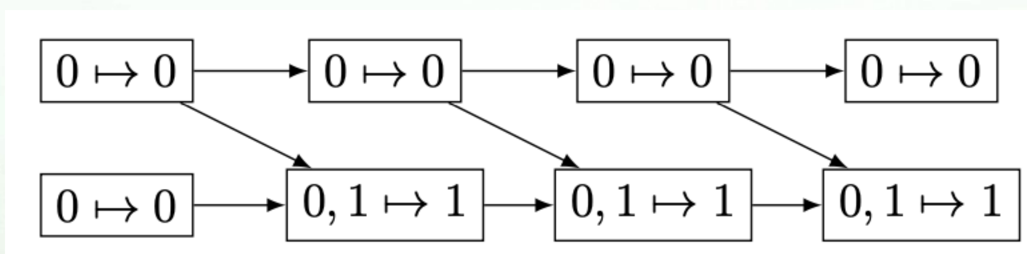




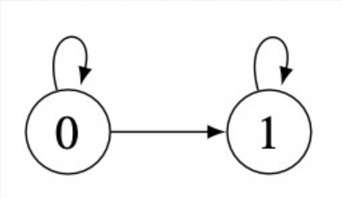
Information flow policy

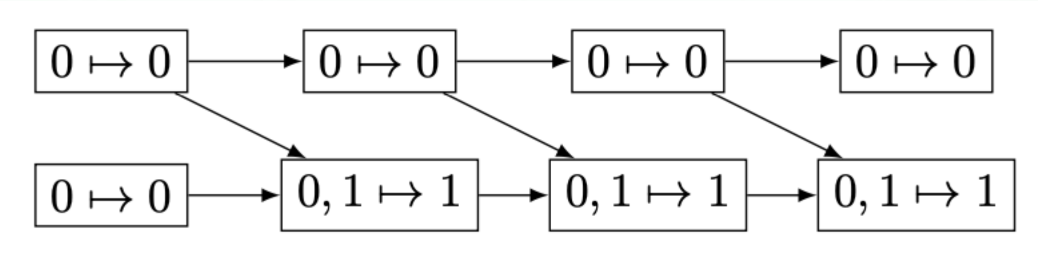


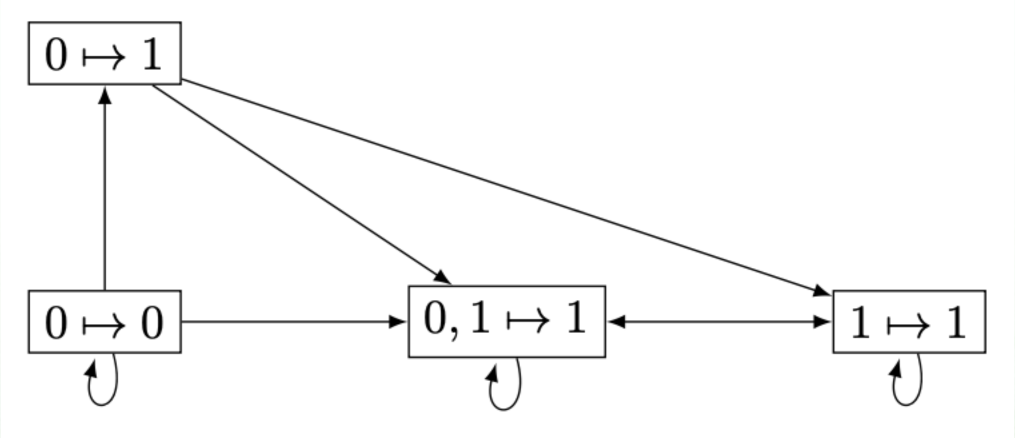




Information flow policy







Information flow policy

- Secure propagation

Theorem 2. *Given a security policy $\mathcal{P} = (T, \preceq)$ and a walk $(u_1, u_2, \ldots, u_l)$ in a $\mathcal{P}$-restricted step graph we have that the tag **out** u_l is reachable from any tag $s \in$ **ins** u_1.*

- Non-exfiltration

Corollary 4 (Non-exfiltration)**.** *Given a security policy $\mathcal{P} = (T, \preceq)$ and a walk $(u_1, u_2, \ldots, u_l)$ in a $\mathcal{P}$-restricted step graph it holds for all $t \in T$ that are not reachable from $s \in$ **ins** u_1 then $t \neq$ **out** u_l.*

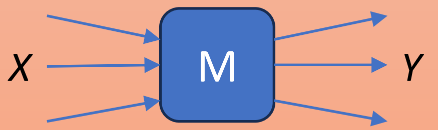
- Non-infiltration

Corollary 5 (Non-infiltration)**.** *Given a security policy $\mathcal{P} = (T, \preceq)$ and a walk $w = (u_1, u_2, \ldots, u_l)$ in a $\mathcal{P}$-restricted step graph it holds for all $s \in T$ from which we cannot reach **out** u_l then $s \notin$ **ins** u_1.*

Noninterference

- Noninterference
  - introduced by Goguen and Meseguer, 1982
  - a property that **restricts** the information flow through a system

X is **noninterfering** with *Y* across a system *M* if *X*'s input to *M* does not affect *M*'s output to *Y*.



Noninterference

- Noninterference implies **confidentiality**

X is **noninterfering** with *Y* across a system *M* if *X*'s input to *M* does not affect *M*'s output to *Y*.

Observations of *Y* are entirely **independent** of the actions of *X*.

Expresses *X*'s **confidentiality** guarantee: *X* cannot reveal any secrets to *Y* via *M*.

Noninterference

- Noninterference implies **integrity**

X is **noninterfering** with *Y* across a system *M* if *X*'s input to *M* does not affect *M*'s output to *Y*.

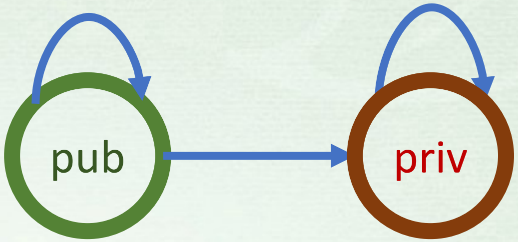
↓

No information flows from *X* to *Y* through *M.*

↓

Expresses *Y*'s **integrity** guarantee:
Y cannot be corrupted by *X* via *M.*

Noninterference



- Interference
  - $pub \rightsquigarrow pub, priv \rightsquigarrow priv, pub \rightsquigarrow \text{priv}$
- Noninterference
  - $priv \not\rightsquigarrow pub$
  - private data does **not** interfere with public data
    - any variation of private data does not cause a variation of public data
  - adversary
    - has access to the public data
    - cannot cannot observe any difference between two executions that differ only in their private data

Language-based IFT

Program analysis
a process of automatic analysis of
the behavior of computer programs

Check correctness

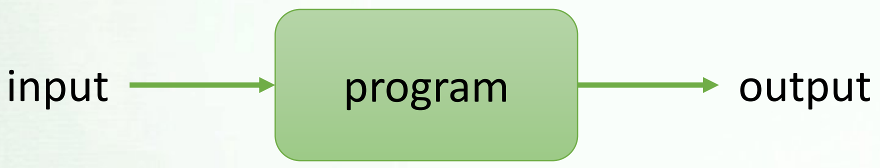
- find programming errors (bugs)
- reveal safety errors
- reveal security vulnerabilities

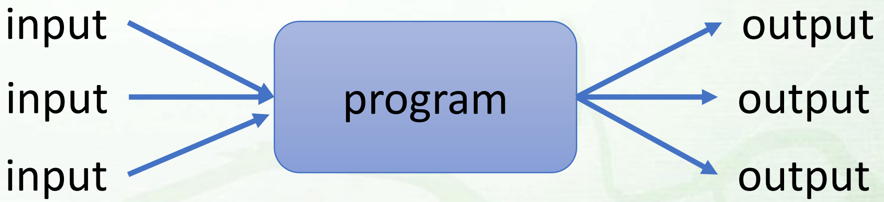
Optimize performance

- improve program performance
- reduce resource usage

Language-based IFT

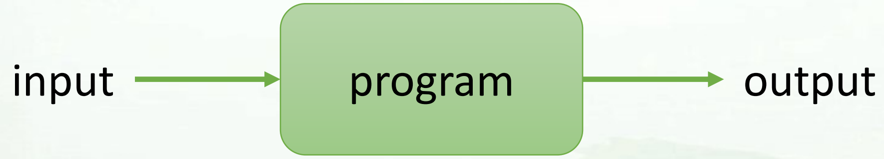
- Language-based IFT
  - to secure data manipulated by a **program**
  - enforce a given information flow policy
  - track possible transfers of information occurring throughout program execution





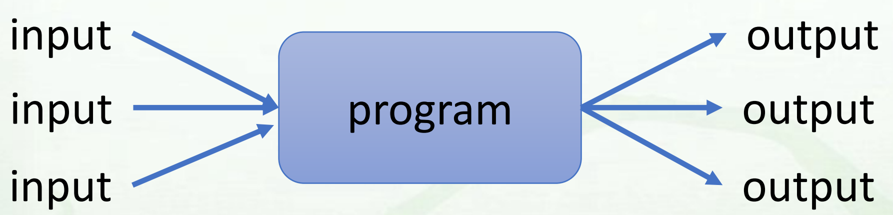
Language-based IFT

- Dynamic IFT
  - analysis during execution (runtime)
    - data from untrusted sources is labeled (tainted)
    - each data (memory location) has a label
    - label propagation at runtime
    - can cause overhead on execution
  - examines only one possibility
    - the actual input
    - may underapproximate possible behavior



Language-based IFT

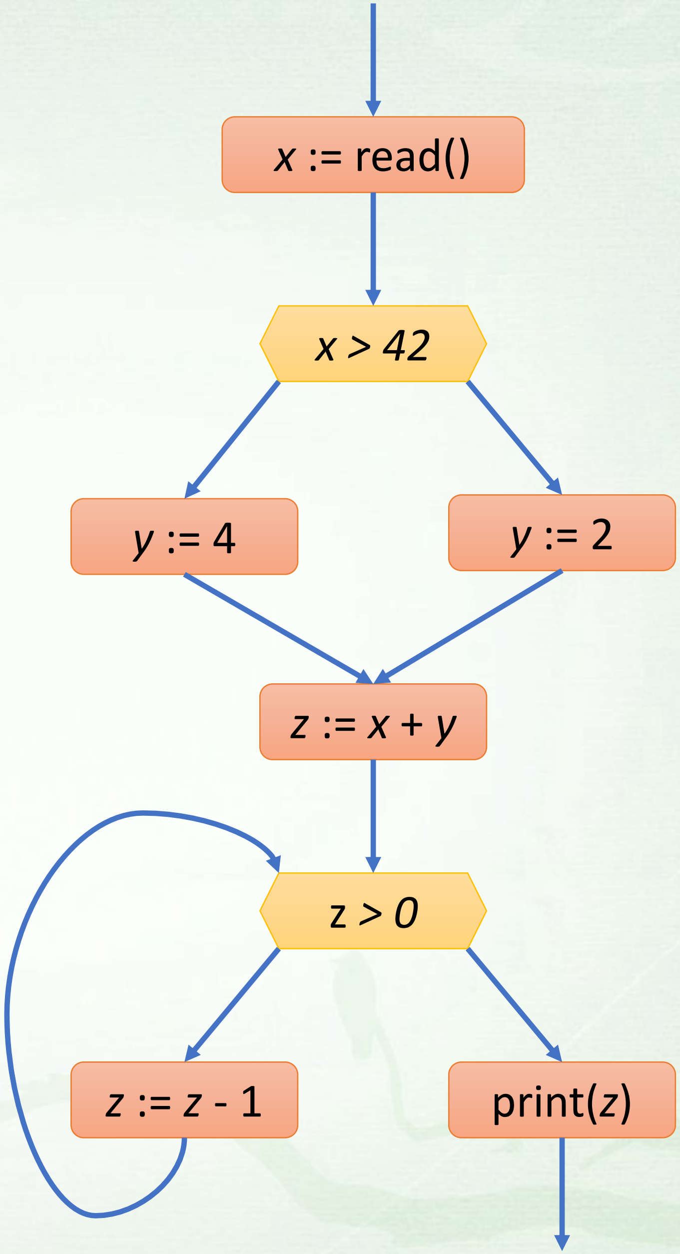
- Static IFT
  - analysis without executing the program/code
    - performed before execution (on compilation)
    - major overhead of analysis
  - examines all possibilities
    - considers all inputs and all execution paths
    - can reveal errors that may not manifest themselves for a long time
    - can overapproximate possible behavior



Language-based IFT

- Control flow graph
  - nodes: operations
  - edges: transfer of control

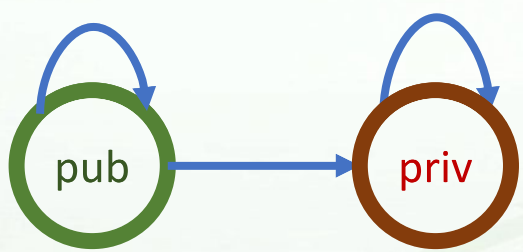
```
x := read()
if x > 42
      then y := 4
      else y := 2
z := x + y
while z > 0 do
      z := z - 1
print(z)
```



Language-based IFT

- Variables and security labels
  - the policy specifies security classes
  - but the program uses variables

- Flow relation on variables
  - $x \rightsquigarrow y \equiv \mathrm{tag}(x) \rightsquigarrow \mathrm{tag}(y)$



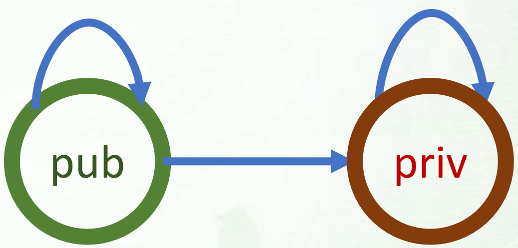

```
x := read()
if x > 42
        then y := 4
        else y := 2
z := x + y
while z > 0 do
        z := z - 1
print(z)
```

Language-based IFT

caused by a
data flow dependency

- Explicit flow
  - from inputs of an operation to its outputs
  - tag propagation rule
    - $\mathrm{tag}(result) = \mathrm{tag}(arg1) \oplus \mathrm{tag}(arg2)$ ...

```
int a: public
int b: private

int x, y, z

// private or public?
x := a + a
y := b + b
z := a + b
```



Language-based IFT

caused by a **control** flow dependency

- Implicit flow
  - in conditionally executed code
  - from the condition to the code

```
bool a: public
bool b: private

bool x, y, z, w

// private or public?
if a then x := true else x := false
if b then y := true else y := false

z := w := false
if a then z := true
if b then w := true
```

```
bool a: trusted
bool b: dubious

string x, y, z, w
string s = user_input()

// trusted or dubious?
if a then x := "Some string"
if a then y := s
if b then z := "Some string"
if b then w := s
```

Language-based IFT

- Hidden implicit flow
  - if a branch is not executed
  - How to handle such flows?
    - Add spurious definitions into branches

```
x := false
if cond then x := true
```

```
x := false
if cond then x := true else x := x
```

```
x := y := 0
if cond then
        x := 42
else
        y := 3.14
```

```
x := y := 0
if cond then
        x := 42
        y := y
else
        y := 3.14
        x := x
```

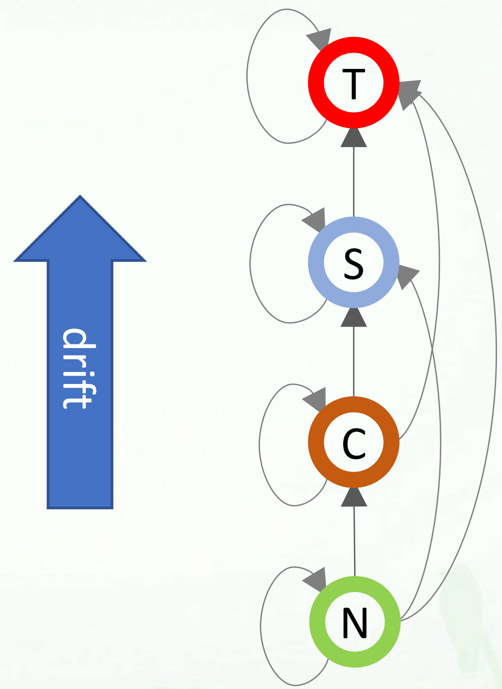
Language-based IFT

- Tag propagation for implicit flow
  - stack *S* of tags
    - contains tags of values
      that influence the current flow of control
  - rules
    - when an operation is executed,
      consider also all tags on *S* for tag propagation
    - when a value *x* influences a branch decision
      push tag(*x*) on the stack *S*
    - when end-of-branch is reached
      pop label(x) from the stack S

Downgrading

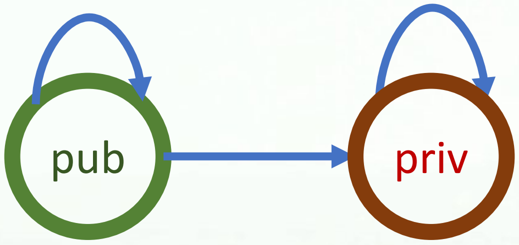
- Challenge: Information upwards drift
  - also called label-creep phenomenon

Top secret
Secret
Confidential
Unclassified



Downgrading

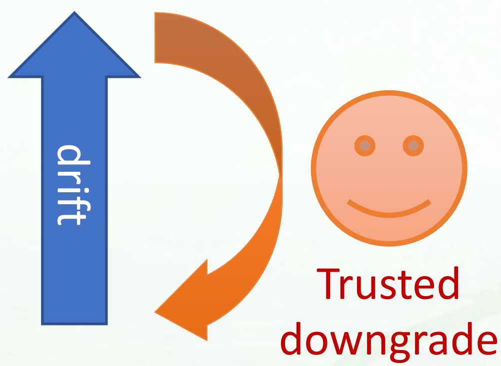
- Challenge: Noninterference is not practical
  - noninterference is too strict
    for use in most real-world applications
    - e.g., prevents all information flows
      from private to public
  - for most applications, the appropriate policy
    should permit controlled downward flows

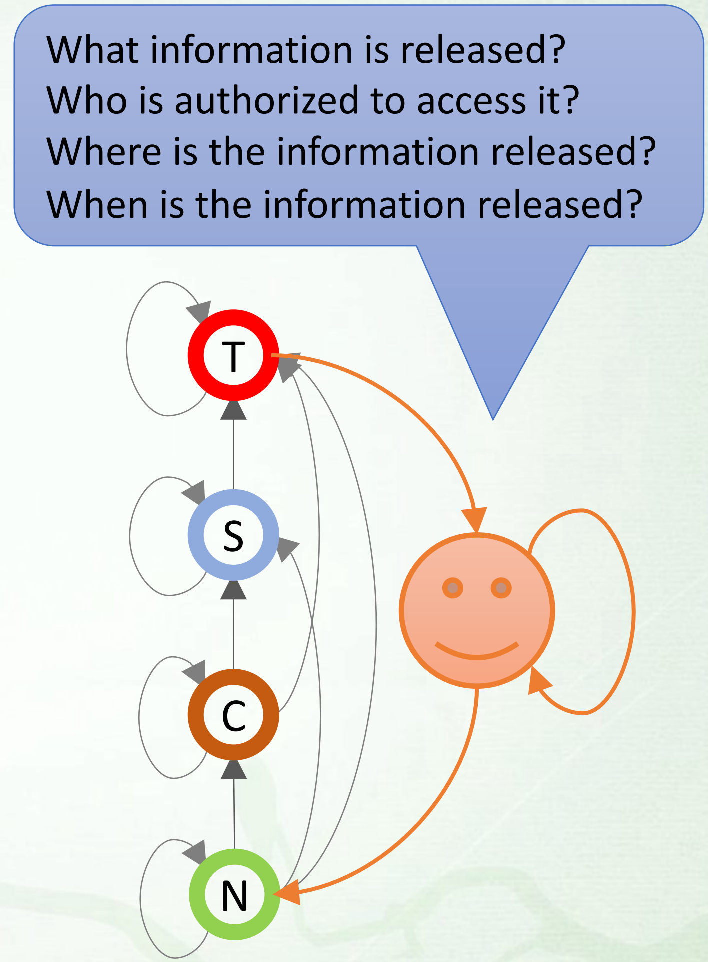


Downgrading

- Trusted user/process
  - may perform downgrading
  - **declassification**
    - for confidentiality policies
  - **endorsement**
    - integrity policies

Top secret
Secret
Confidential
Unclassified





Downgrading

- Examples
  - encryption

```
pt := "42 is the answer"
ct := encrypt(pt)
```

  - hashing

```
m := "A private message"
h := hash_sha256(m)
```

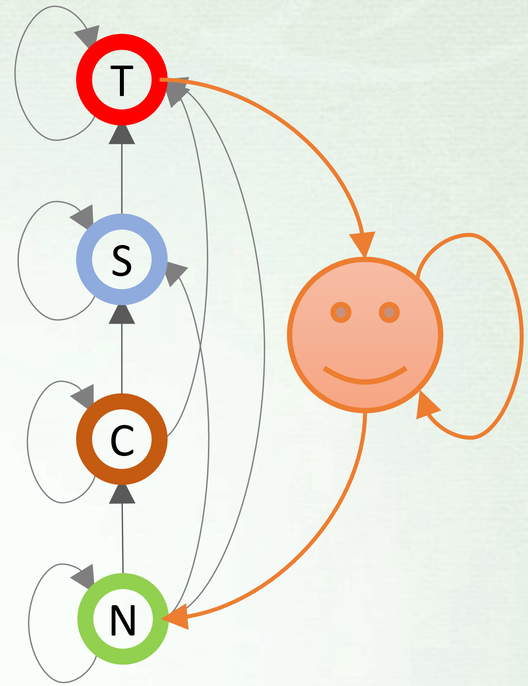
  - password check

```
pw := read_input()
ok := pw.length() >= 10
```

  - html escaping

```
x := read_input()
y := html_escape(x)
```

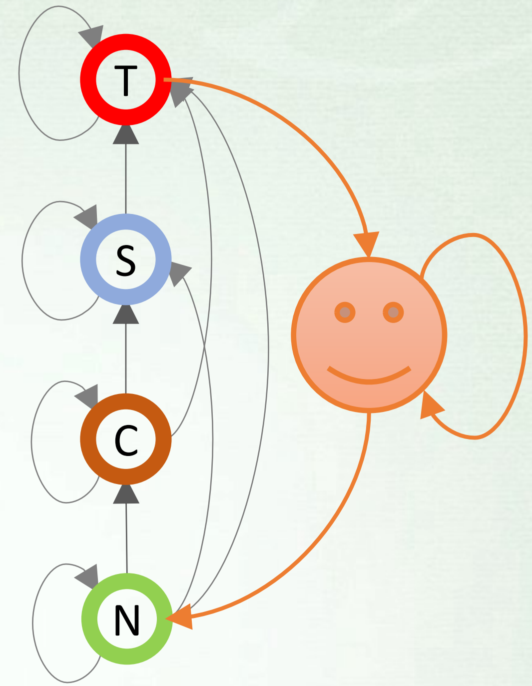
Downgrading



- Intransitive security policy
  - ensures that downward information flow passes through trusted user
  - cycles in the IF policy
- Intransitive non-interference
  - not accurate description
    - actually, interference relation is not transitive
  - noninterference under an intransitive security policy

Downgrading

- Separating the relation
  - security-oblivious operations
  - security-aware operations




```
pw := read_input()
ok := pw.length() >= 10
ok := downgrade(ok)
print(ok)
```

Thank you