

CS 162 Week 8

Kyle Dewey

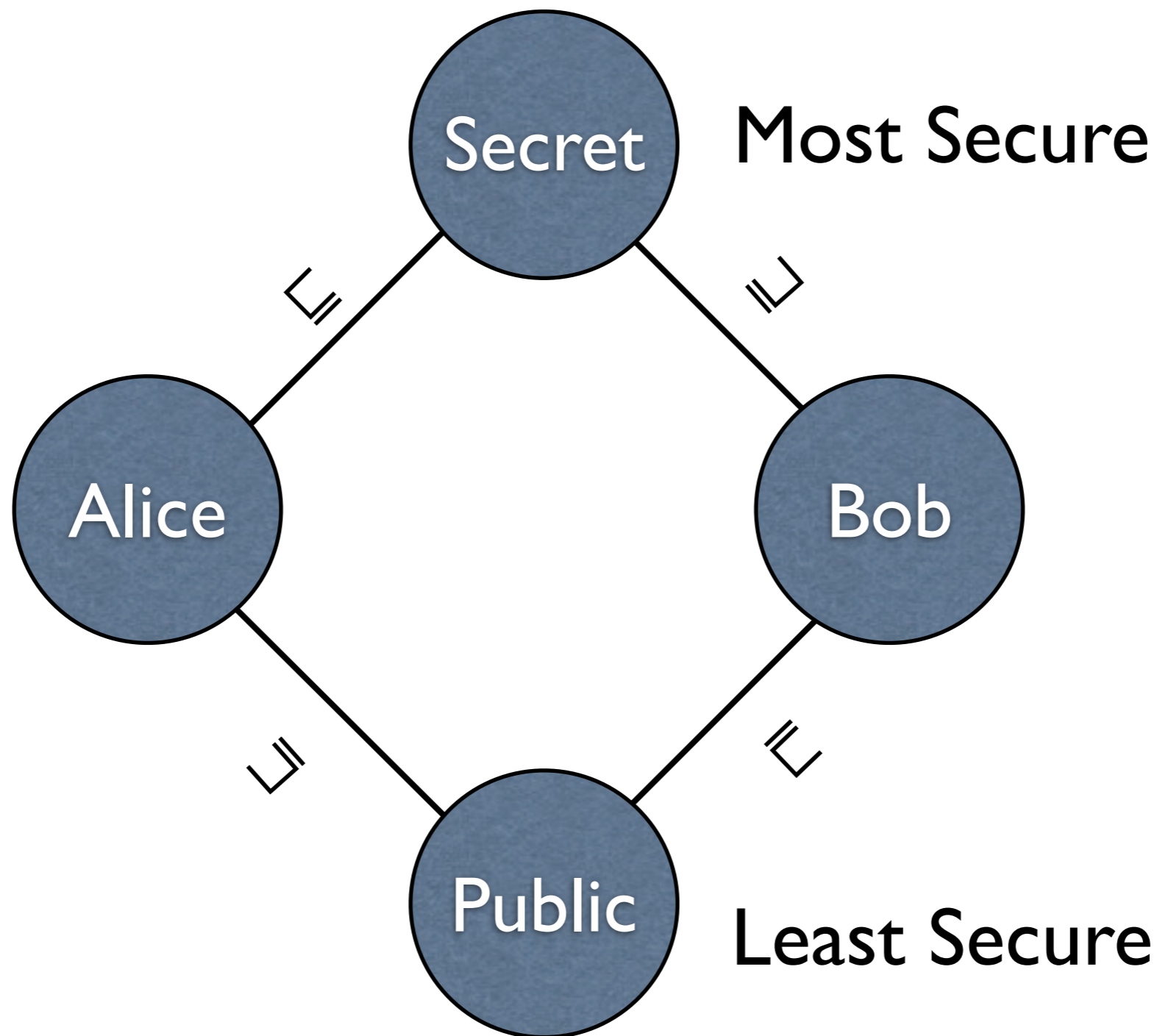
Overview

- Example online going over `fail03.not` (from the test suite) in depth
- A type system for secure information flow
- Implementing said type system

Flashback: Assignment 2

- Implemented dynamically enforced secure information flow via label tracking
- Each `Storable` is labeled with a tag describing its security level
- Security levels interact in well-defined ways

Security Levels



Basic Idea

- Specify which channel we output to (public, secure, etc.)
- Only output values of equal or lesser security than the specified value (i.e. do not output something secure on a public channel)
- When values interact to produce new values, the new values take the security level of the most secure thing they touched

Example

```
var x, y in
```

```
output secret "enter secret number: ";
```

```
x := input secret num;
```

```
y := x;
```

```
output public y
```

Issues

- This system works, but there are two major issues with it (well, 1.5 major issues)
- What's problematic?

#1: Termination Leaks

- Whether or not a program halts can leak a bit to an attacker
- In a dynamic system, certain kinds of leaks are transformed into termination leaks
 - I.e. instead of outputting a secret value to public, throw an exception and terminate instead

#2: Dynamic

```
var x, y in
  x := input secret num;
  y := input public num;
  if (y = 42) {
    output public x
  }
```

A Solution

- A static type system and type checker, specifically for secure information flow
- If a program typechecks, then it is **guaranteed** that it is secure
- Type systems being type systems, if it does not typecheck, it still *might* be secure

Assignment 5

- Implement a type checker for secure information flow for miniJS
 - Slightly more complex lattice
 - Implicitly typed
- The type system, along with all the necessary rules, is provided
- Overall very similar to the type system coverage in the lecture

Major Difference from Lecture

- Lecture is using equivalence constraints
 - Solved using the union-find data structure
- This assignment will use subset constraints
 - These slides cover how to solve these

Three Core Challenges

1. Understanding the math
2. Implementing the math
3. Solving the constraints

#1 Understanding The Math

- Basics needed to understand it covered extensively in lecture
- Biggest difference from lecture: L is used as a type variable instead of T , since L is a security level

#2 Implementing Math

- General rule: implement as close to the math as possible
- The more it deviates, the more difficult it is to reason about whether or not they are equivalent
- ...and the more difficult it becomes to track down bugs

Implementing Math

$$\frac{\ell_w \sqsubseteq L \quad \ell \sqsubseteq L}{\rho \cdot \ell_w \vdash \mathbf{input} \ell \text{ typ} : L}$$

```
case In(typ, l) => {  
  val L = LevelVariable()  
  lw ∈ L  
  l ∈ L  
  L  
}
```


#3: Solving the Constraints

Constraint Generation

- Very similar to how constraints were generated in lecture
- Based on subsets instead of equality

$$\frac{\ell_w \sqsubseteq L \quad \ell \sqsubseteq L}{\rho \cdot \ell_w \vdash \mathbf{input} \ell \text{ typ} : L}$$

Constraint Generation

Once all the rules have completed, we end up with a bunch of constraints like these:

`Public ⊆ L1`

`Public ⊆ L2`

`Public ⊆ Secret`

`Public ⊆ L3`

`L3 ⊆ L1`

`L2 ⊆ L4`

`Secret ⊆ L4`

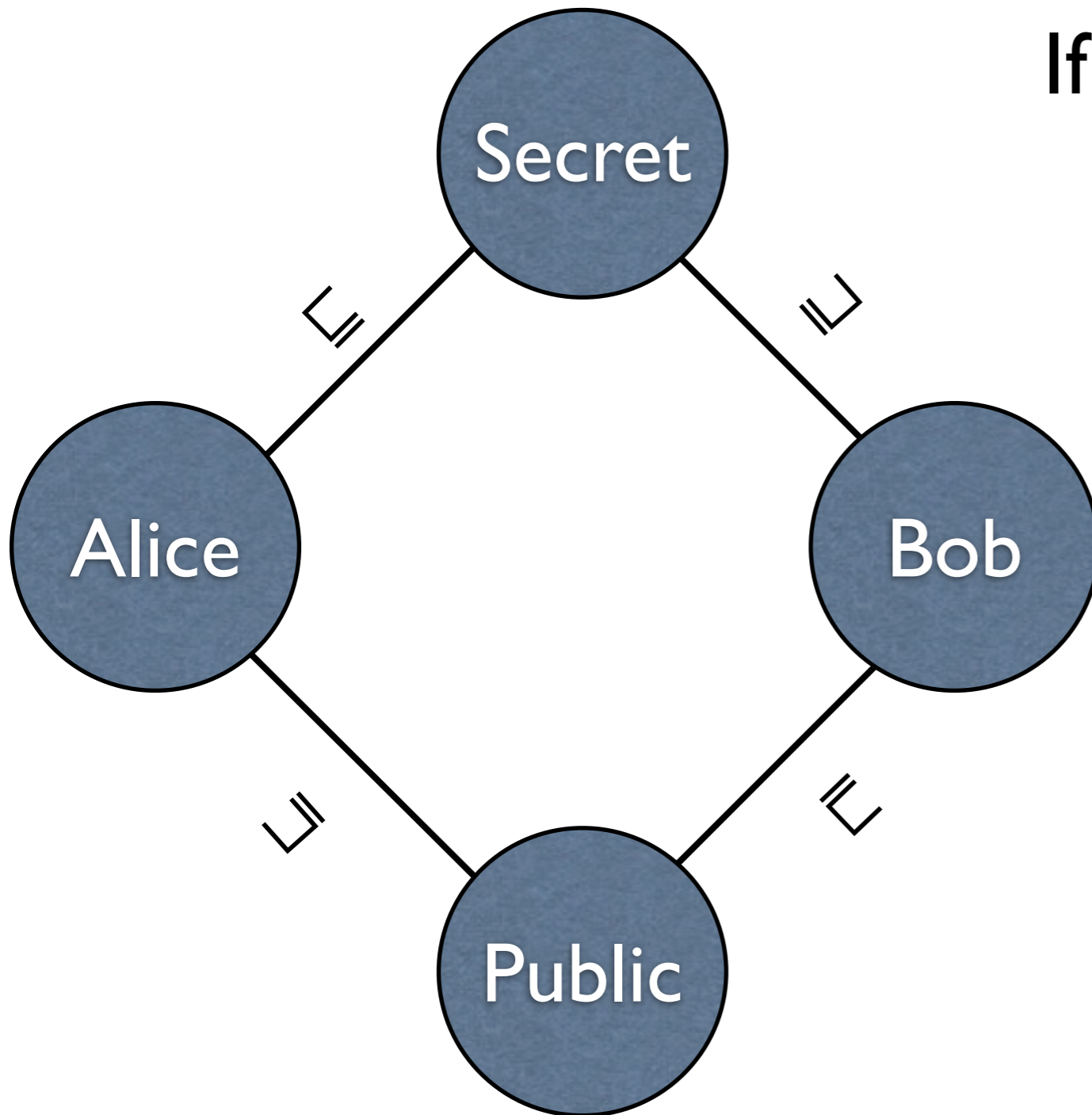
**...where `L1`, `L2`, and `L3`
are type variables**

Constraint Meaning

- These constraints show what the syntax of the program says about the security lattice
- We already know what the security lattice looks like
 - If the two are consistent, the program typechecks
 - If there are inconsistencies, the program fails to typecheck

Finding Inconsistencies

If any constraints violate this partial order, it means the program is not well-typed



Secret ~~≡~~ Public
Secret ~~≡~~ Bob
Secret ~~≡~~ Alice
Bob ~~≡~~ Public
Alice ~~≡~~ Public
Bob ~~≡~~ Alice
Alice ~~≡~~ Bob

Not that Easy

- What about type variables?
- Need a way to map these back to concrete types

Public \sqsubseteq L1

Public \sqsubseteq L2

Public \sqsubseteq Secret

Public \sqsubseteq L3

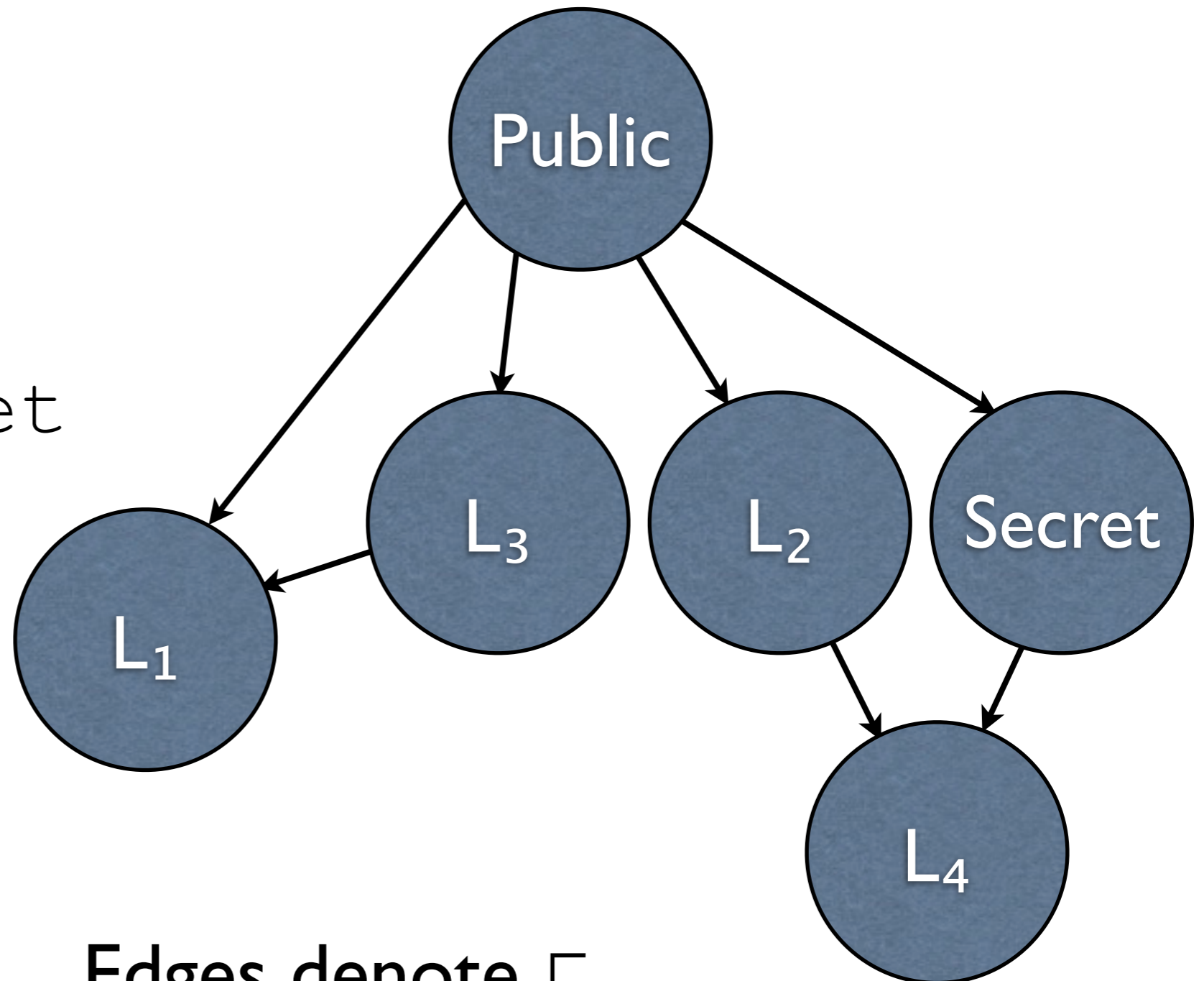
L3 \sqsubseteq L1

L2 \sqsubseteq L4

Secret \sqsubseteq L4

Coalescing Constraints

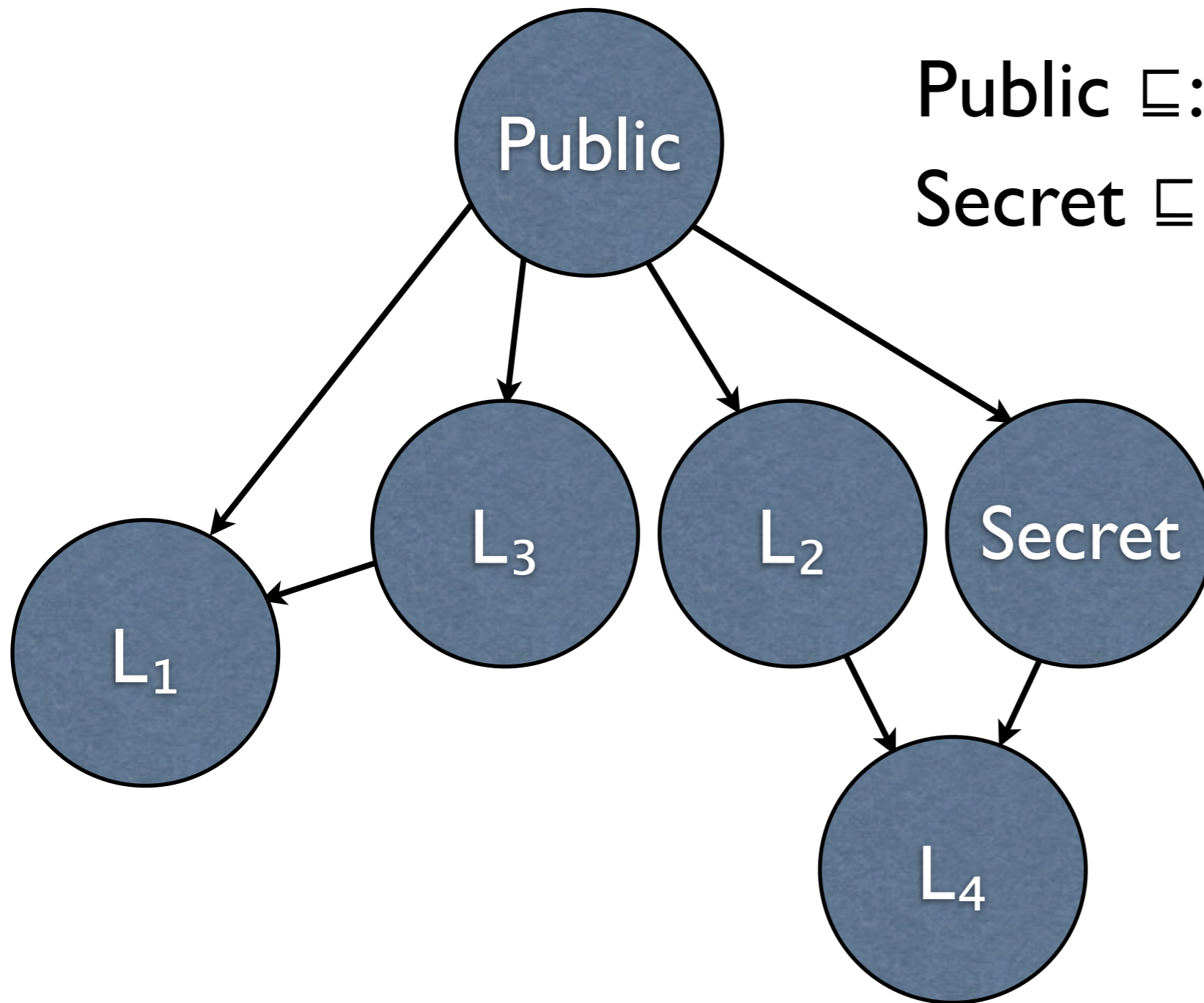
Public \sqsubseteq L1
Public \sqsubseteq L2
Public \sqsubseteq Secret
Public \sqsubseteq L3
L3 \sqsubseteq L1
L2 \sqsubseteq L4
Secret \sqsubseteq L4



Edges denote \sqsubseteq

Getting the Full \sqsubseteq

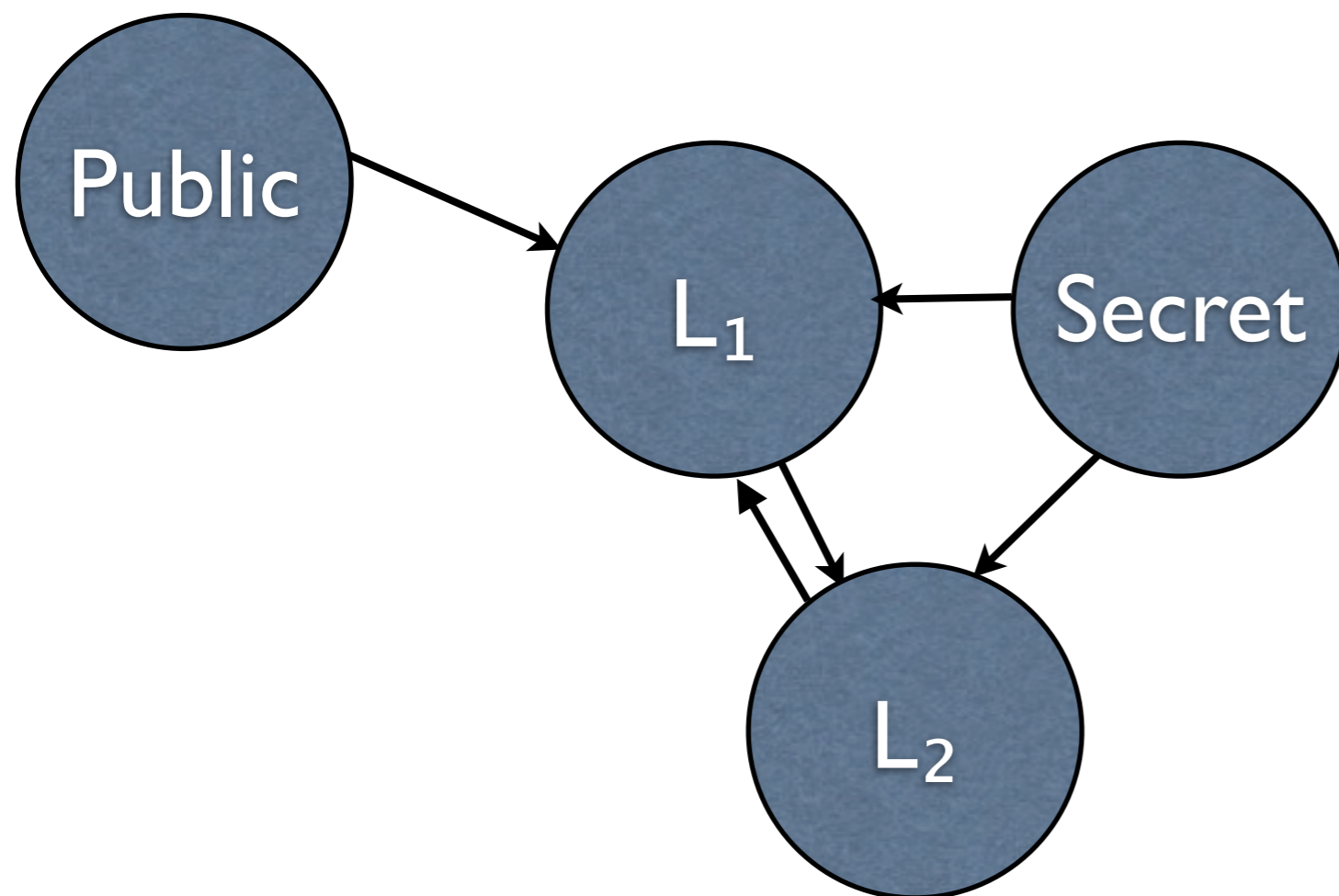
If node n_1 can reach node n_2 , then $n_1 \sqsubseteq n_2$



Public \sqsubseteq : $\{L_1, L_2, L_3, L_4, \text{Secret}\}$
Secret \sqsubseteq : $\{L_4\}$

Cycles

- The graph can contain cycles
- This should not break anything



Public \sqsubseteq : $\{L_1, L_2\}$

Secret \sqsubseteq : $\{L_1, L_2\}$

Final Step

- Verify the full sets are consistent
- Only need to consider concrete security levels (public, secure, etc.)

Secret ~~⊆~~ Public

Secret cannot reach Public

Secret ~~⊆~~ Bob

Secret cannot reach Bob

Secret ~~⊆~~ Alice

Secret cannot reach Alice

Bob ~~⊆~~ Public

Bob cannot reach Public

Alice ~~⊆~~ Public

Alice cannot reach Public

Bob ~~⊆~~ Alice

Bob cannot reach Alice

Alice ~~⊆~~ Bob

Alice cannot reach Bob

Constraint Solver Implementation

- Free to use mutability
- It is not necessary to make an explicit graph, but you are free to do so if you wish
 - It is likely easier to avoid it if possible
- For each constraint, add an edge and possibly nodes to this graph
- A global counter will be needed for generating unique type variables

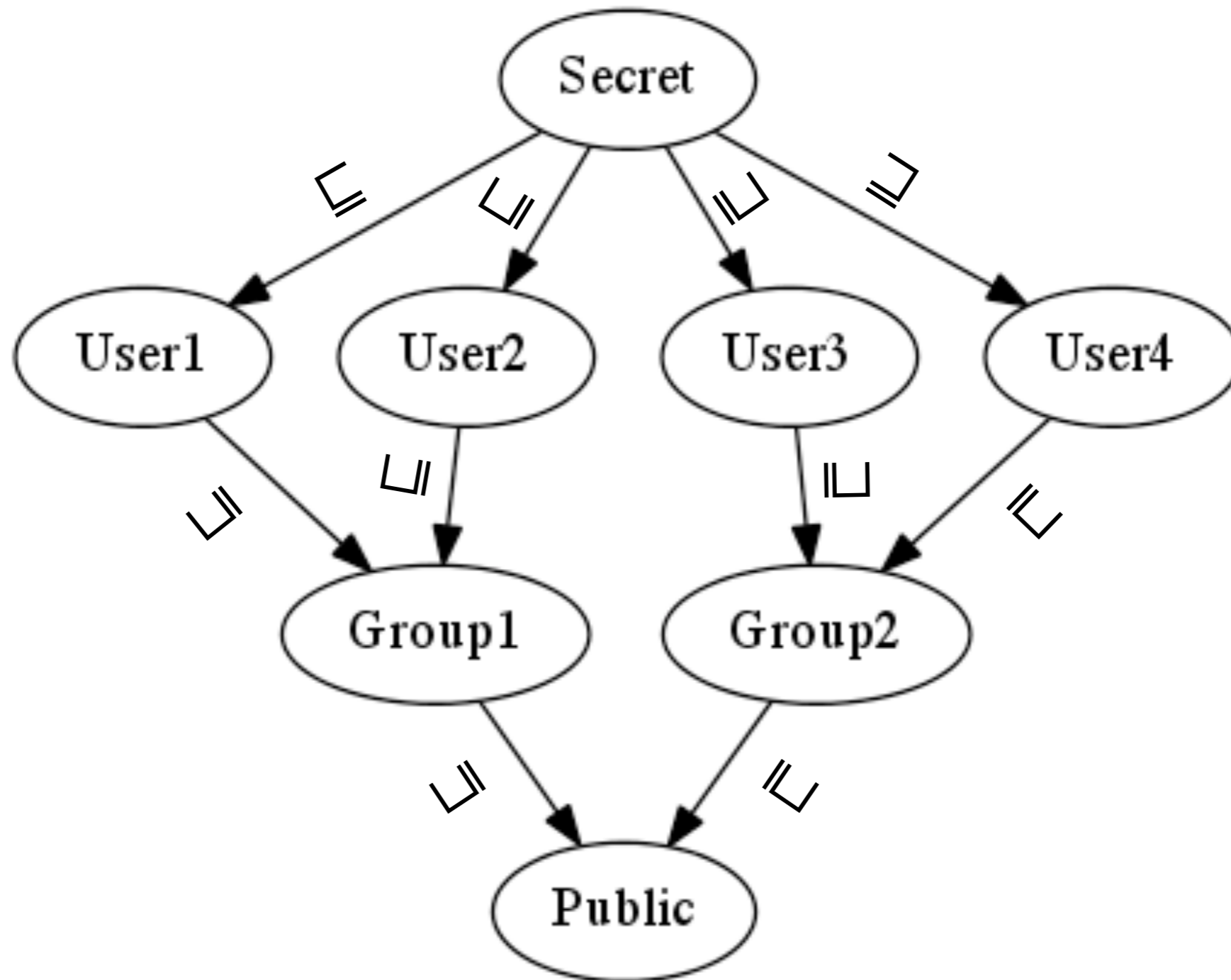
Removed for Simplicity

- Each node should have an edge pointing to itself
- Since the definition of \sqsubseteq includes equality, for all levels l , $l \sqsubseteq l$
- Hint: this can be exploited in implementations that do not have an explicit graph

Security Lattice

- The security lattice we use for this assignment is more complex than the `Public`, `Secret`, `Alice`, and `Bob` lattice used in Assignment 2

Security Lattice



Things to Avoid

- For full credit, you **MUST** implement the subset constraint solving outlined here
- Maximum score for any other method: C+
- Do not try to hack assignment 2 until it is a type checker
- Like hacking on a compiler until it's a text editor
- Historically, starting from scratch is simpler

The Math in Depth

fail03.not **Example**