Systems Security

PROFESSIONAL EDUCATION

CSAIL
MIT COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE LABORATORY

The Role of Programming Languages in Security

Armando Solar-Lezama

Associate Professor without Tenure

Computer Science and Artificial Intelligence Laboratory (CSAIL)

Massachusetts Institute of Technology

Outline

Languages and low-level security properties

Type safety

Languages and high-level security properties

Do programming languages matter?

Not all languages are equal

Language choice matters for security

Language design decisions can influence bugs

- And the security implications of those bugs

The trend is towards automation

What is the meaning of a program?

```
#include <stdio.h>

int main(int argc, char** argv){
        printf("hello world")
}
```

Program above has a well defined meaning

The semantics of a language define how a program in that language is going to behave

Defining a language semantics

Implementation based

- The correct behavior of a program is whatever my compiler/interpreter does on that program
- By definition, the compiler/interpreter has no bugs (only surprising behaviors)

Standards documents

- Option of choice for mature languages
- Clearly define the behaviors that can be expected from different constructs

Formally defined in mathematical terms

- Several well established formalisms for doing this
- Essential if you want to prove programs correct

What will this program do?

```
#include <stdio.h>

int main(){
  char buf[100];
  fgets(buf, 1000, stdin);
  printf("You typed %s", buf);

}
```

```
void foo(){
  printf("You were hacked");

}
```

Dead Code

Semantics corner cases

int buf[10] ; buf[20] = 5;

“hello world” / 2.5

(new Button()) * 22

None of these program fragments makes sense

Someone may write them anyway, then what?

What to do?

Undefined

- Let the system do whatever it wants in these cases

Define a specific behavior for each case

Rule them out statically

What to do?

The C approach

Undefined

- Let the system do whatever it wants in these cases

Define a specific behavior for each case

Rule them out statically

What to do?

Undefined

- Let the system do whatever it wants in these cases

Define a specific behavior for each case

Rule them out statically

What to do?

Undefined

- Let the system do whatever it wants in these cases

Define a specific behavior for each case

Rule them out statically

Avoiding security problems with types

What is a type system

Narrow View

- It's a mechanism for ensuring that variables only take values from predefined sets
  - Ex. Integers, Strings, Characters
- A mechanism for avoiding unchecked errors
  - by ruling out programs with undefined behaviors
  - by specifying how a program should fail (eg. NullPointerException)

Expansive View

- It's a light-weight proof system and annotation mechanism for efficiently checking for a specific property of interest
- Address bugs that go beyond corner-cases in the semantics
  - Information flow violations
  - deadlocks
  - etc, etc, etc

Separating code from data

Users cannot add arbitrary code to application

&

All code has a well defined meaning

⇒

You can check if your application can do bad

(at least in theory)

Separating code from data

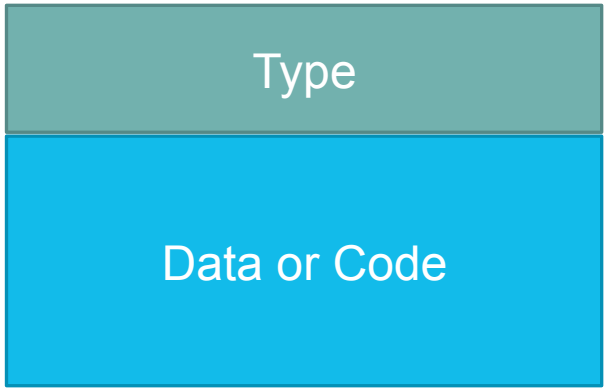
Most programs need to get data from user

If data can be confused for code you have a path for users to inject code

Dynamic approach

Tag everything

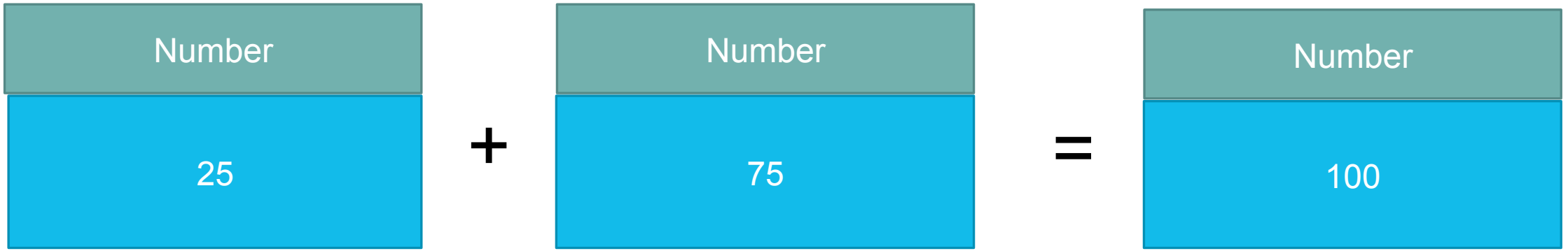
- References cannot point to the middle of a block
- Only the runtime system can write tags
- Check the tag before performing any operations



Dynamic approach

Tag everything

- References cannot point to the middle of a block
- Only the runtime system can write tags
- Check the tag before performing any operations



Dynamic approach

Tag everything

- References cannot point to the middle of a block
- Only the runtime system can write tags
- Check the tag before performing any operations

Dynamic approach

Achieves separation of code & data

Useful in defining behavior of programs

Inefficient!

Ruling out bad programs with types

Key idea

Reason statically about the set of types any variable or field can point to

Restrict this set so each variable can only point to certain types

Then you don't need to carry tags dynamically!

Example

```
int foo(int x, int y){
    int z = x + y;
    return z;
}
```

- Someone else guarantees x and y are integers
- Z is integer because adding integers always produces integers
- Thus we can guarantee that we are returning an integer

Limitations of static reasoning

int* buf = new int[100]

Is buf[101] an integer?

- Dynamically it is easy to tell that an access to buf[101] should not be allowed.
- buf is of type int[100] and the operation [101] is not allowed for such an object
- Tracking this statically is very difficult

Type checking

Deductive approach to ensure a program is type safe

- Language has a well defined set of operations for each type of data
- A type safe program will never apply an operation to a type of data for which it is not defined

What do we mean by deductive

Defined by rules that determine establish that given some premises, some statement about the type of a value will be true

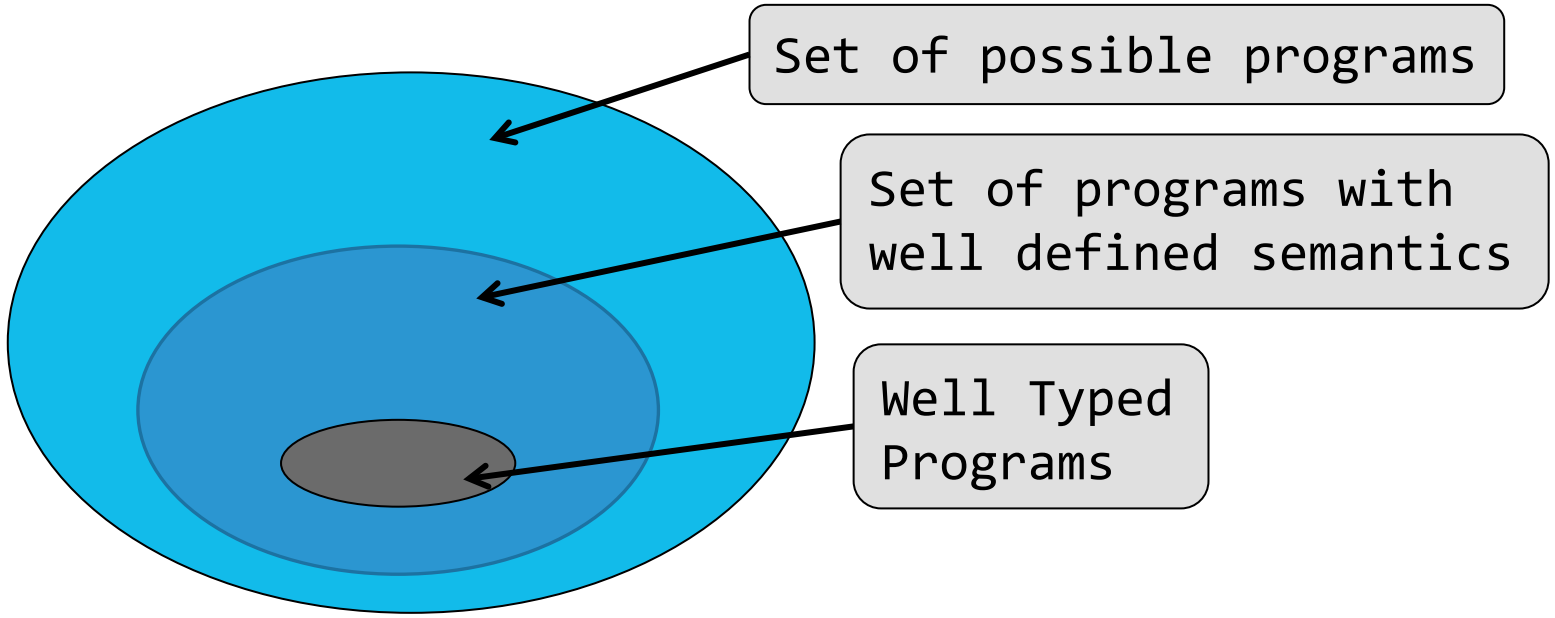
$$\frac{\Gamma \vdash e1 : int \qquad \Gamma \vdash e2 : int}{\Gamma \vdash e1 + e2 : int}$$

Modularity

Practical type systems are also modular

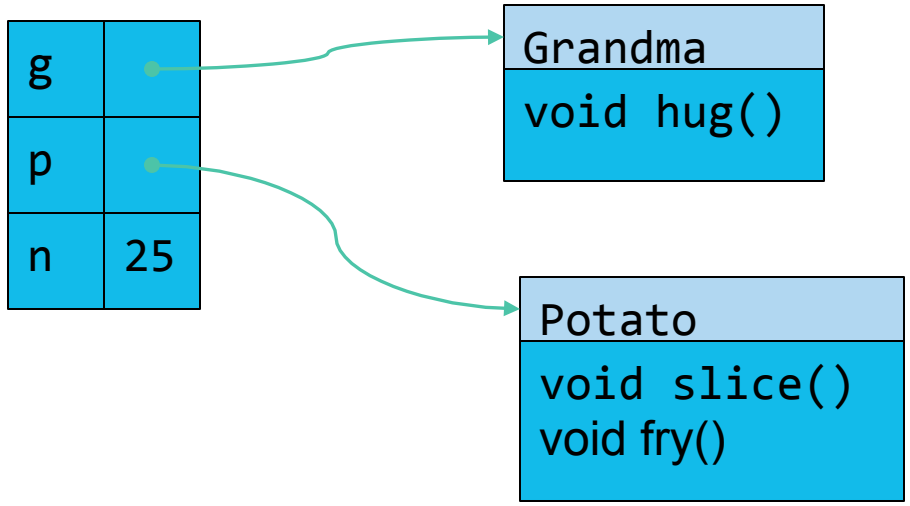
- You can reason about different parts of the program independently
- You can pinpoint problems to their root cause

Recap



Beyond separating code and data

Recap



~~p.hug()~~

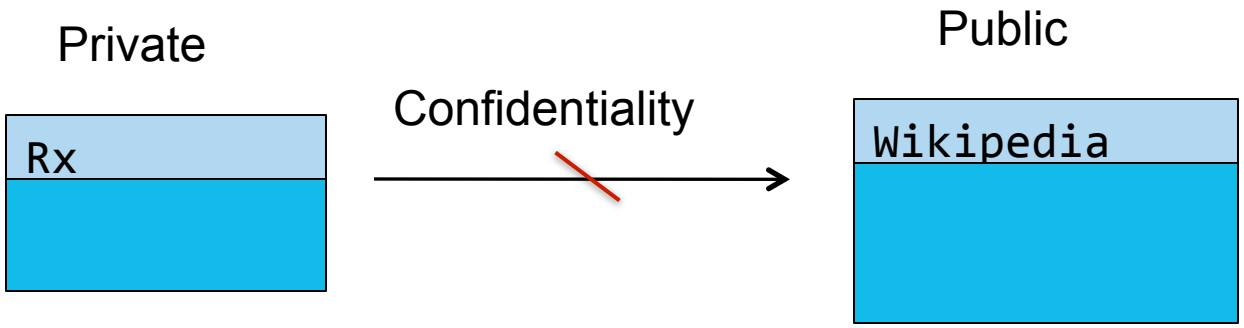
~~g.fry()~~

~~n.hug()~~

types place restrictions on the store

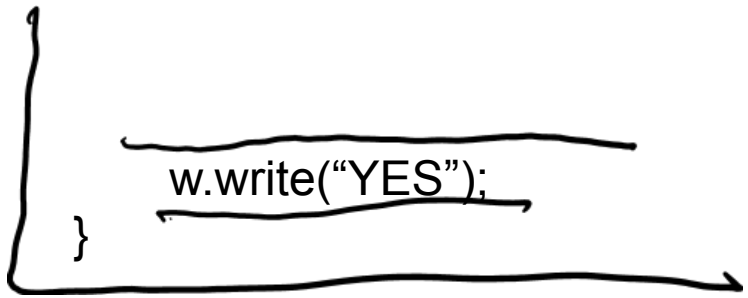
- this allows static reasoning about legal operations on the objects in the store

Enforcing Security Properties

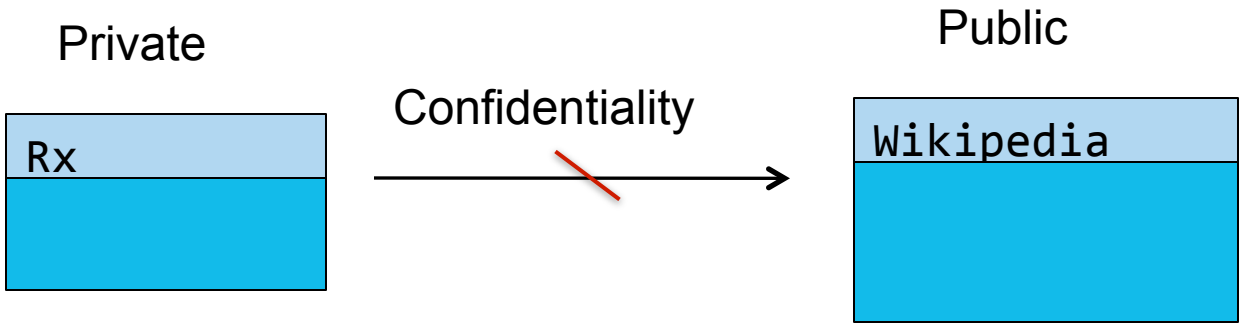



```
Rx myrx = getMyRx();
Wikipedia w = getWPEntry("Armando");

w.addEntry(myrx.toString());
```



Enforcing Security Properties



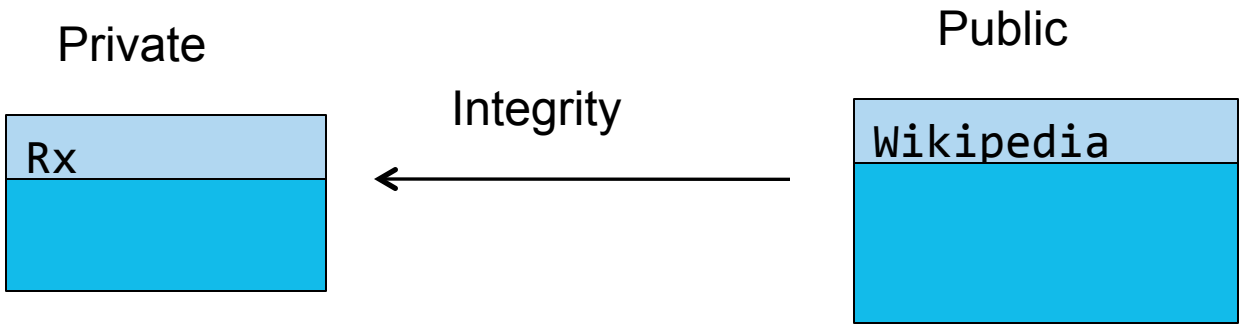

```
Rx myrx = getMyRx();
Wikipedia w = getWPEntry("Armando");

w.write("Has flu:");
p.val = myrx.contains("Tamiflu");
if(q.val){
      w.write("YES");
}
```

Even if p!=q, information can still leak if p!=q was caused by some information about myrx.

If p==q information clearly leaks

Enforcing Security Properties

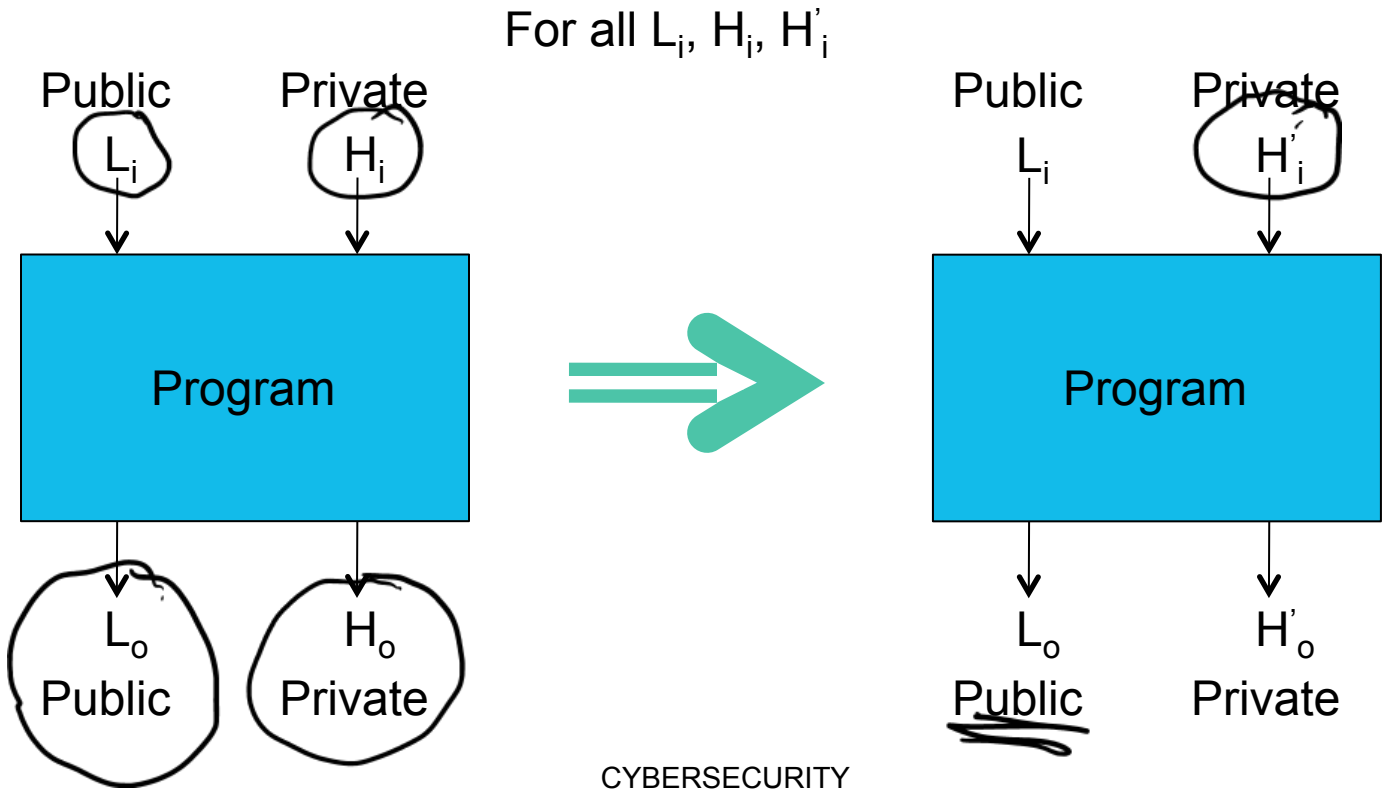



```
class Doctor{
    Rx cureFlu(){
        Rx myrx = new Rx();
        Wikipedia w = getWPEntry("Flu");
        myrx.set(w.getSubEntry("Treatment"));
        return myrx;
    }
}
```

What is information flow?

If there is no information flow from private to public, then a change in a private input can not affect a public output

- you can't determine this from a single execution



Information flow with types

Based on the paper "**JFlow: practical mostly-static information flow control" by Andrew C Myers**

Solution Strategy

- Define a dynamic labeling scheme so that at any given time, the labels in a piece of data tell us whether it's OK to leak it or not.
  - Labels turn a global property about all executions into a local property in a conservative way
  - This will be the dynamic semantics against which we can prove type safety.
- Define a type system that allows us to approximate the set of labels that the data pointed at by a variable can have.
  - If an action is ok according to the conservative approximation, we know it would be ok according to the dynamic scheme.

Labeling Data With Security Policies

Policies for information flow

Owner: reader1, reader2, reader3

- "according to owner, this data can only be read by reader1, reader2, or reader3"

Label

{ policy1, policy2, policy3 }

- If an owner is not mentioned, it is assumed she has no privacy concerns

Why do we need an owner?

Revocation

Labels have a partial order

L1 <= L2

L1 can be relabeled to L2

- means that L2 is more restrictive (fewer readers)
- Warning: this is counterintuitive
  - L2 actually has fewer readers.

If a variable is certified to handle data with L2 labels correctly, we can trust that variable to hold a value with label L1

- Just like subtyping!

Assignment

x{L2} := v{L1};

L1 <= L2

Can only assign to a variable to a more restrictive label

Binary Operations

a{L1} + b{L2};

Trick question:

- What should be the label for a+b?

```
int{Joe:everyone} a, b, c;
...
int{Joe:Joe} p;
c = 0;
if(p){
    c = a + b;
}
```

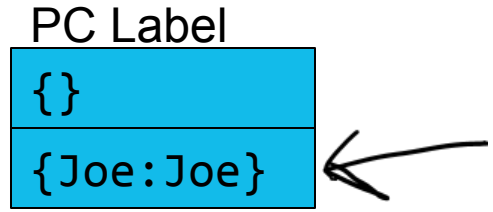
- What information would be leaked if this code were to execute?

Information flow through control

Information flow through the PC

- We need to keep track of the information that is leaked just from knowing that the computation reached a particular point.

```
int{Joe:everyone} a, b, c;
...
int{Joe:Joe} p;
c = 0;
if(p){
    c = a + b;
}
```



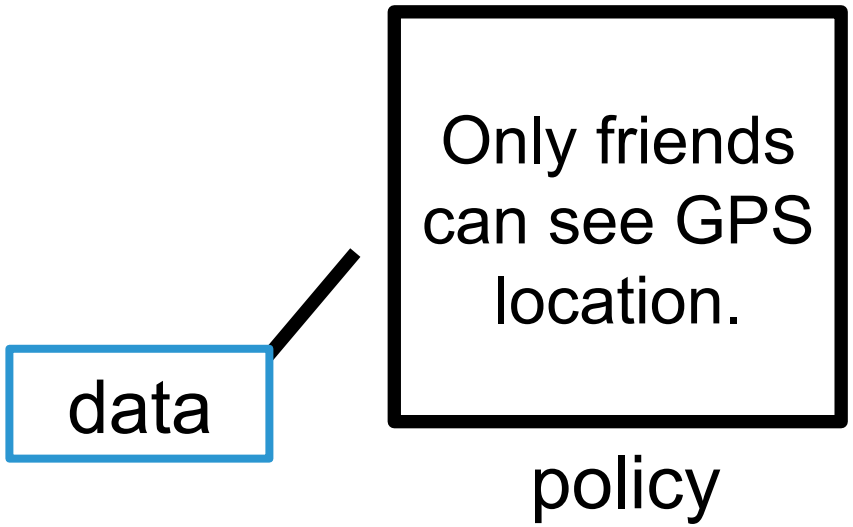
Simple scheme except for non-structured control

- return, continue, throw, break

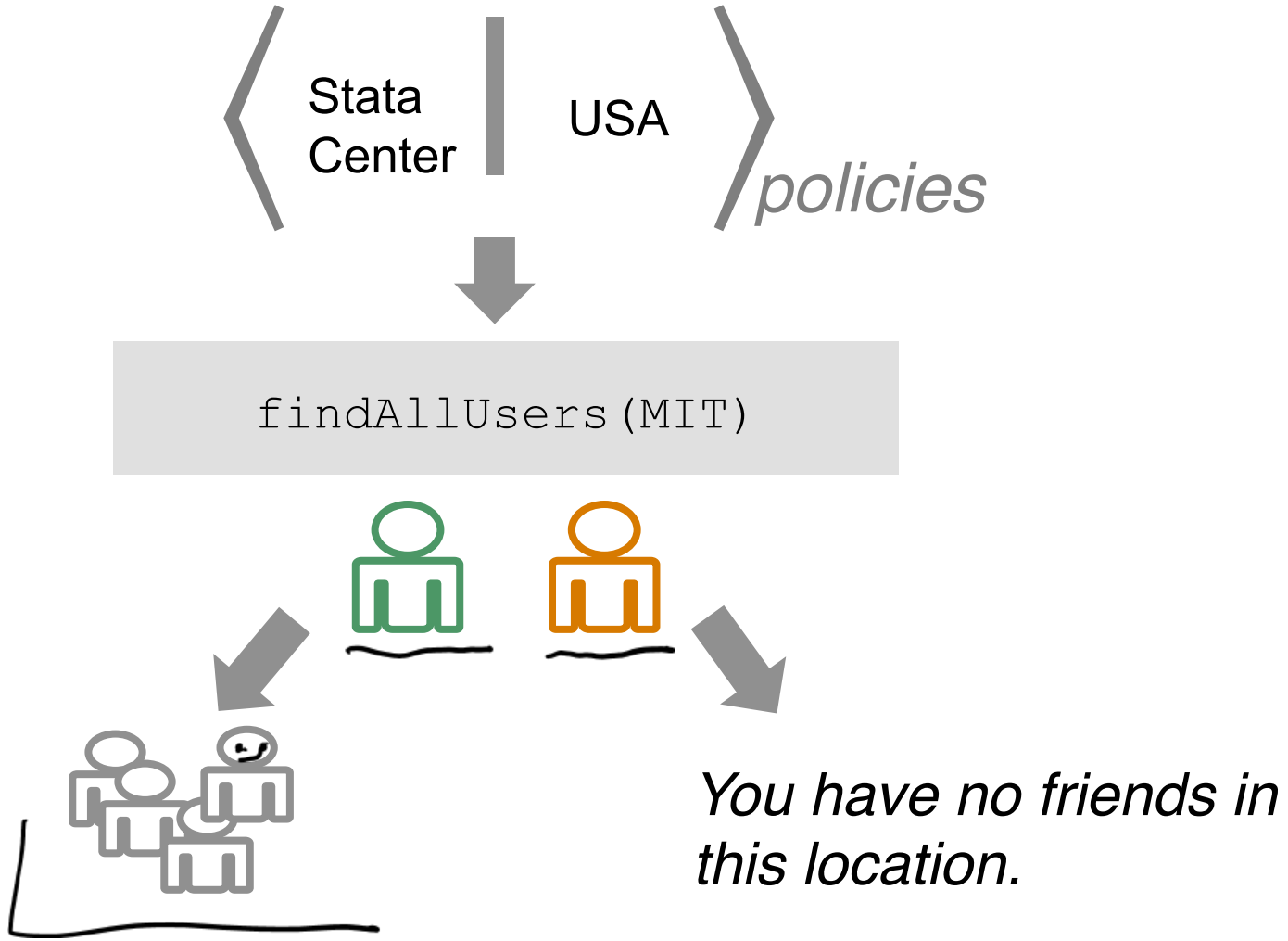
Beyond checking

Based on the paper "**A language for automatically enforcing privacy policies.**" **by Jean Yang, Kuat Yessenov, Armando Solar-Lezama**

In Jeeves, associate policies with data for automatic enforcement.



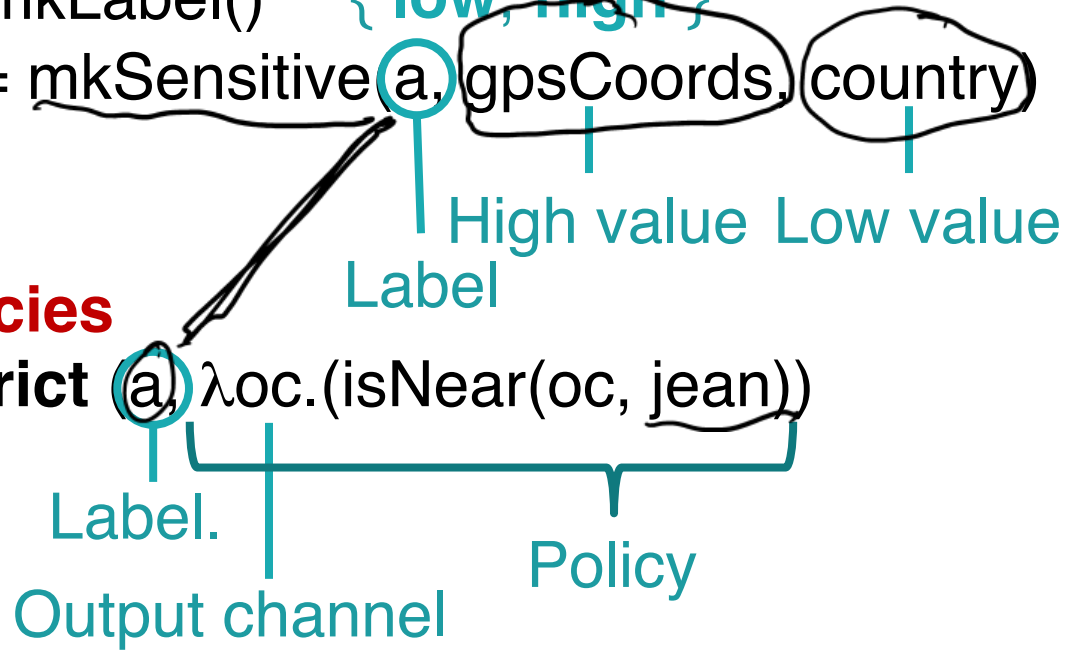
The Jeeves Language



Sensitive Values

a = mkLabel() { low, high }

loc = mkSensitive(a, gpsCoords, country)



Policies

restrict (a, λoc.(isNear(oc, jean))

Core Functionality

msg = “Jean’s location is ” + asStr(loc)

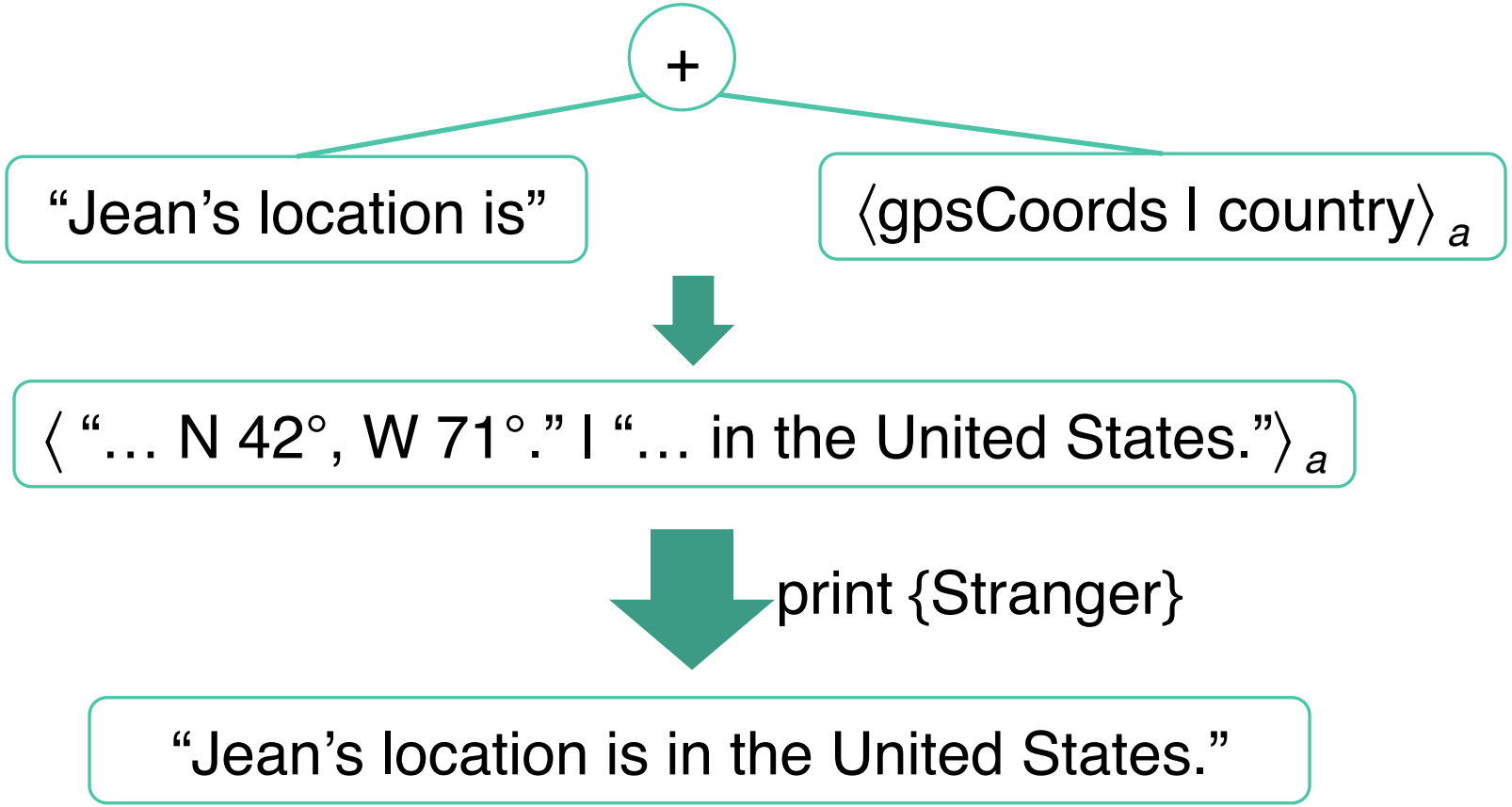
Contextual Enforcement

print {owen} msg *“Jean’s location is N 42°, W 71°.”*

print {rishabh} msg *“Jean’s location is in the United States.”*

Jean Yang / Jeeves

Faceted Execution



Takeaway points

Language can help enforce complex policies

Tradeoffs between dynamic enforcement and performance

THANK YOU

Armando Solar-Lezama

Associate Professor Without Tenure