A Language for Automatically Enforcing Privacy

Jean Yang

with Kuat Yessenov

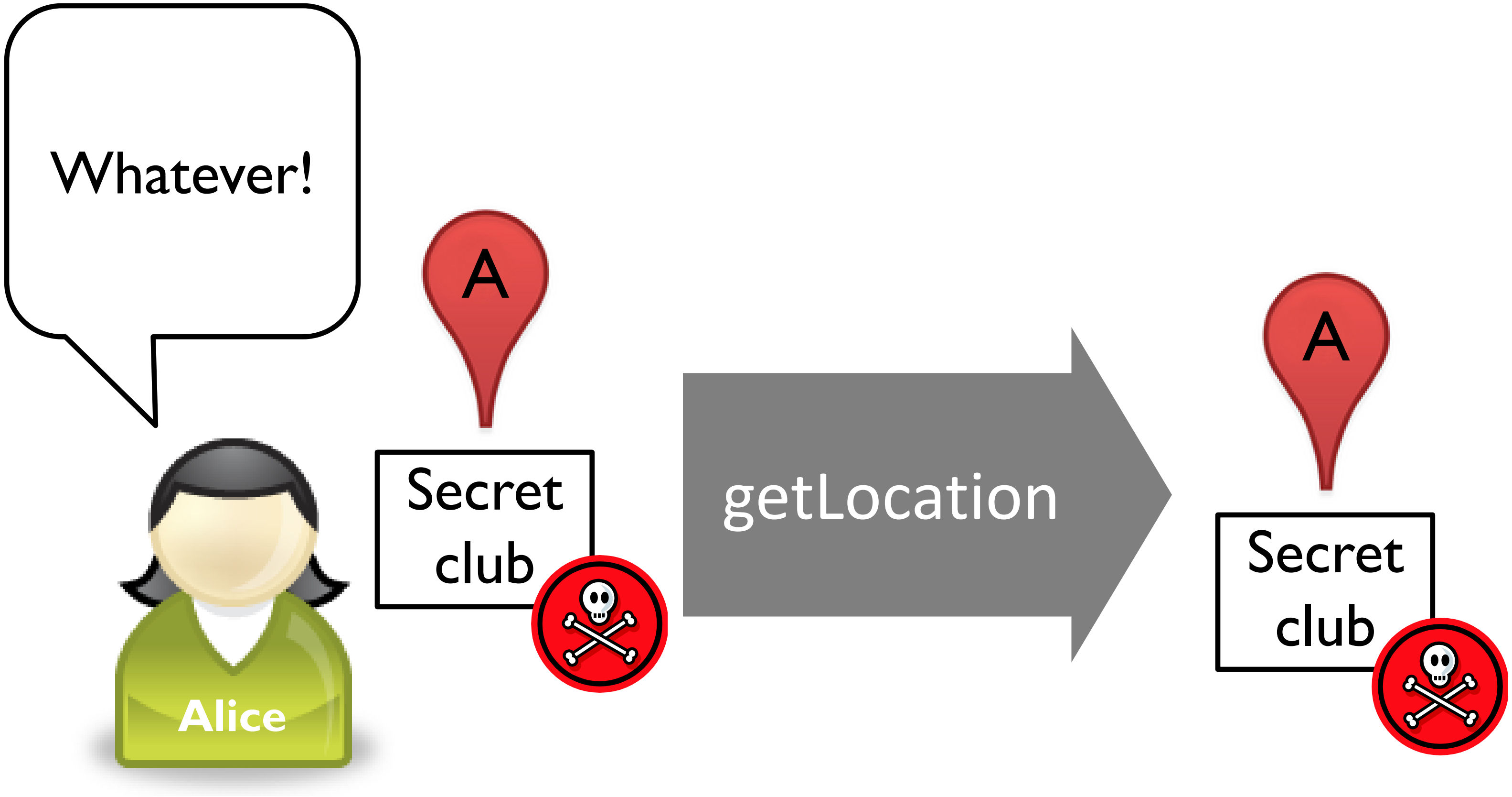
and Armando Solar-Lezama

Displaying User Locations to Other Users

getLocation

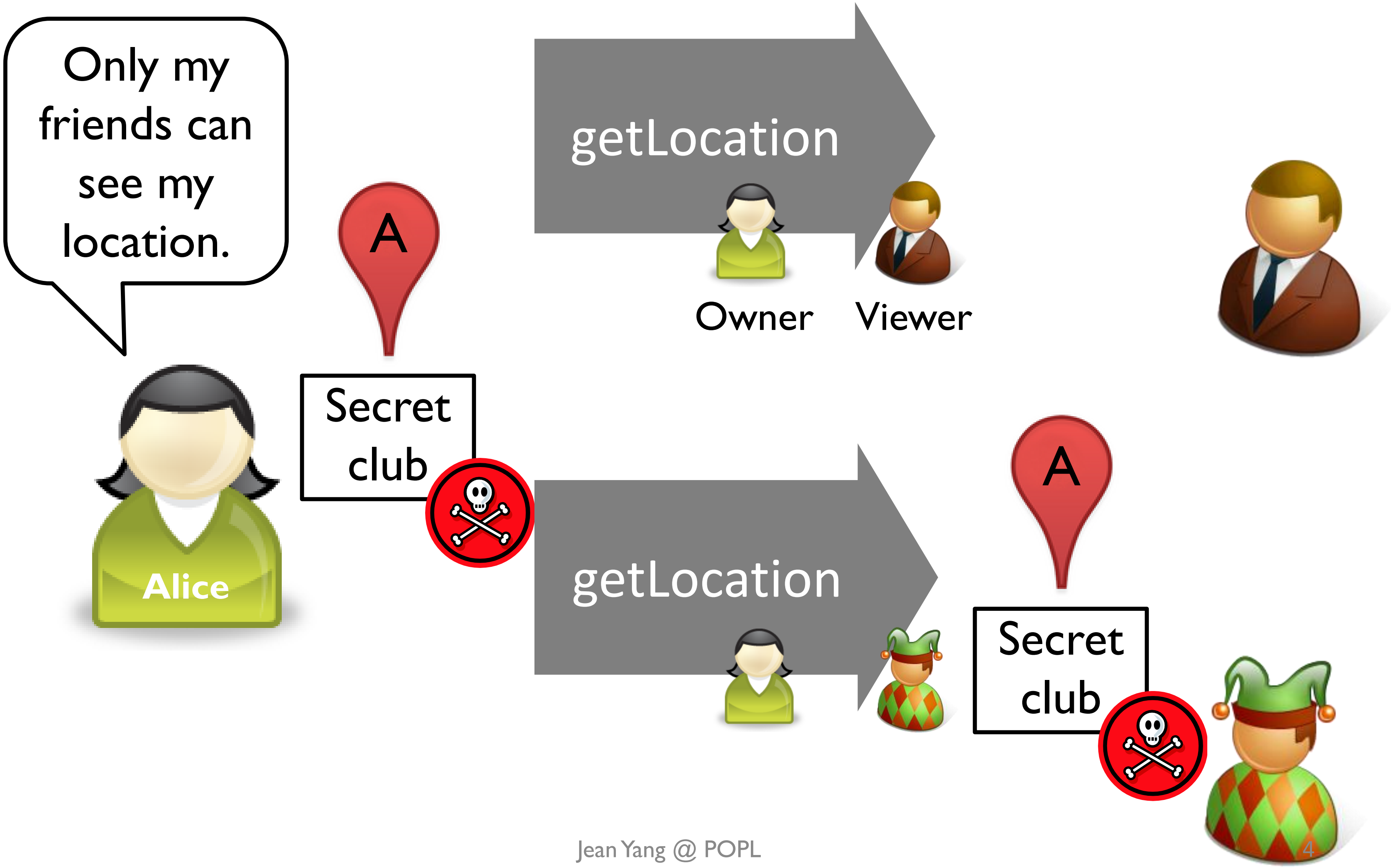
?

No Privacy Concerns

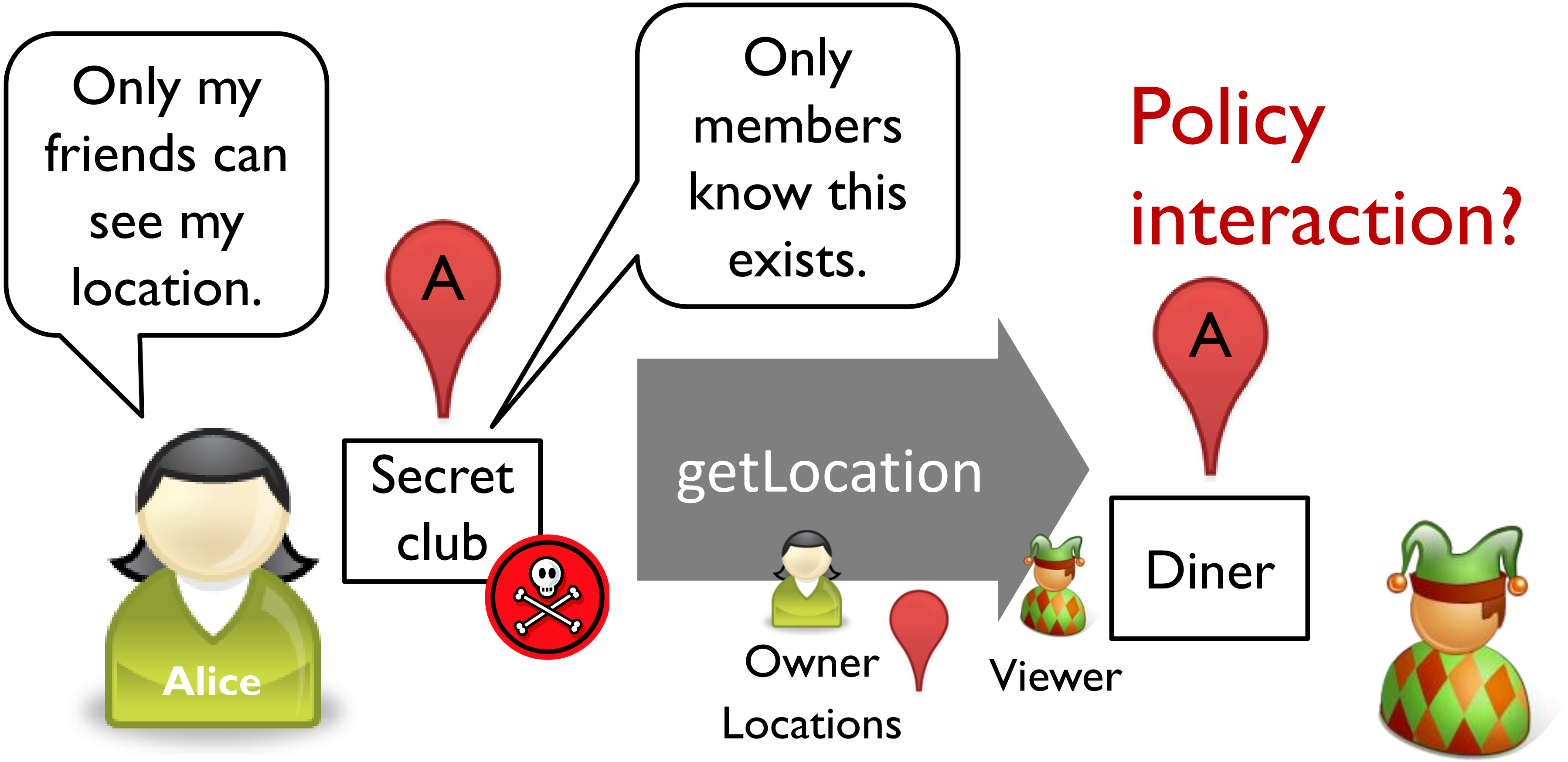


```
def getLocation (user: User): Location = user.location
```

Simple policy



Finer-Grained Policies



Only my friends can see my location.

Only members know this exists.

A

Secret club

getLocation

Owner

Locations

Viewer

A

Diner

Alice

Not a member!

Policy interaction?

Which policies apply where?

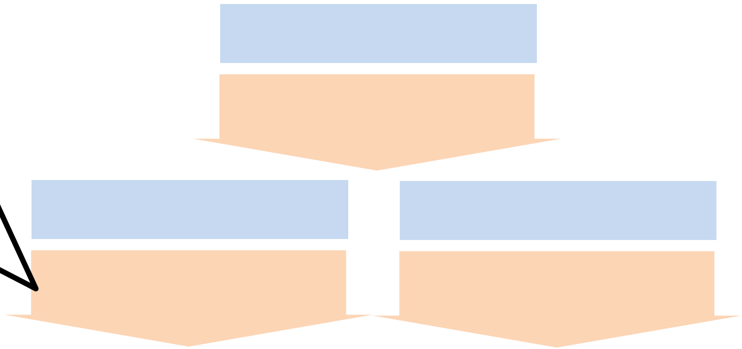
Programmer Burden

```
def getLocation (user: User) (viewer: User)
  : Location = {
  if (isFriends user viewer) {
      if (canSee user.location viewer) {
        user.location;
      } else { scrub(user.location, “Diner”); }
  } else { undisclosedLocation; }
}
```

Output Context

Policies

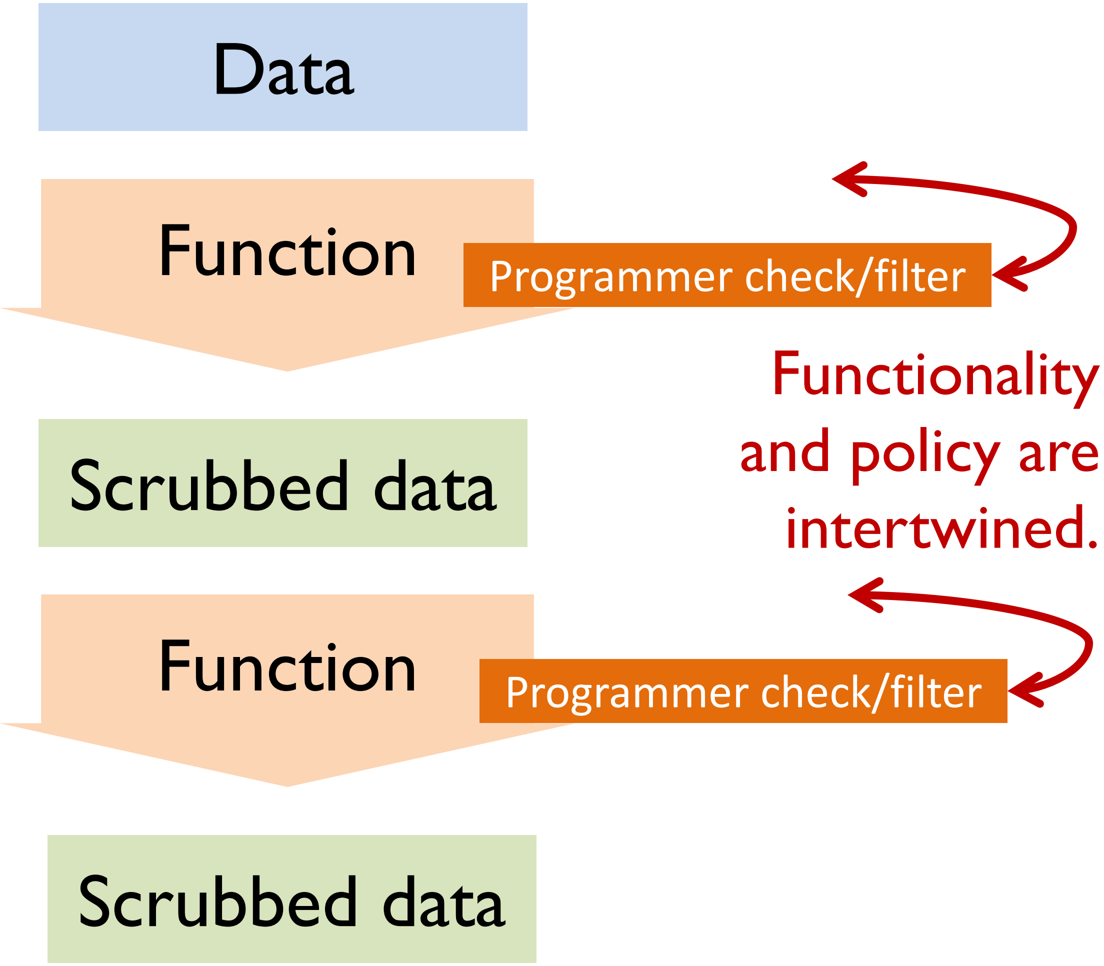
Views of sensitive values



Our Mission

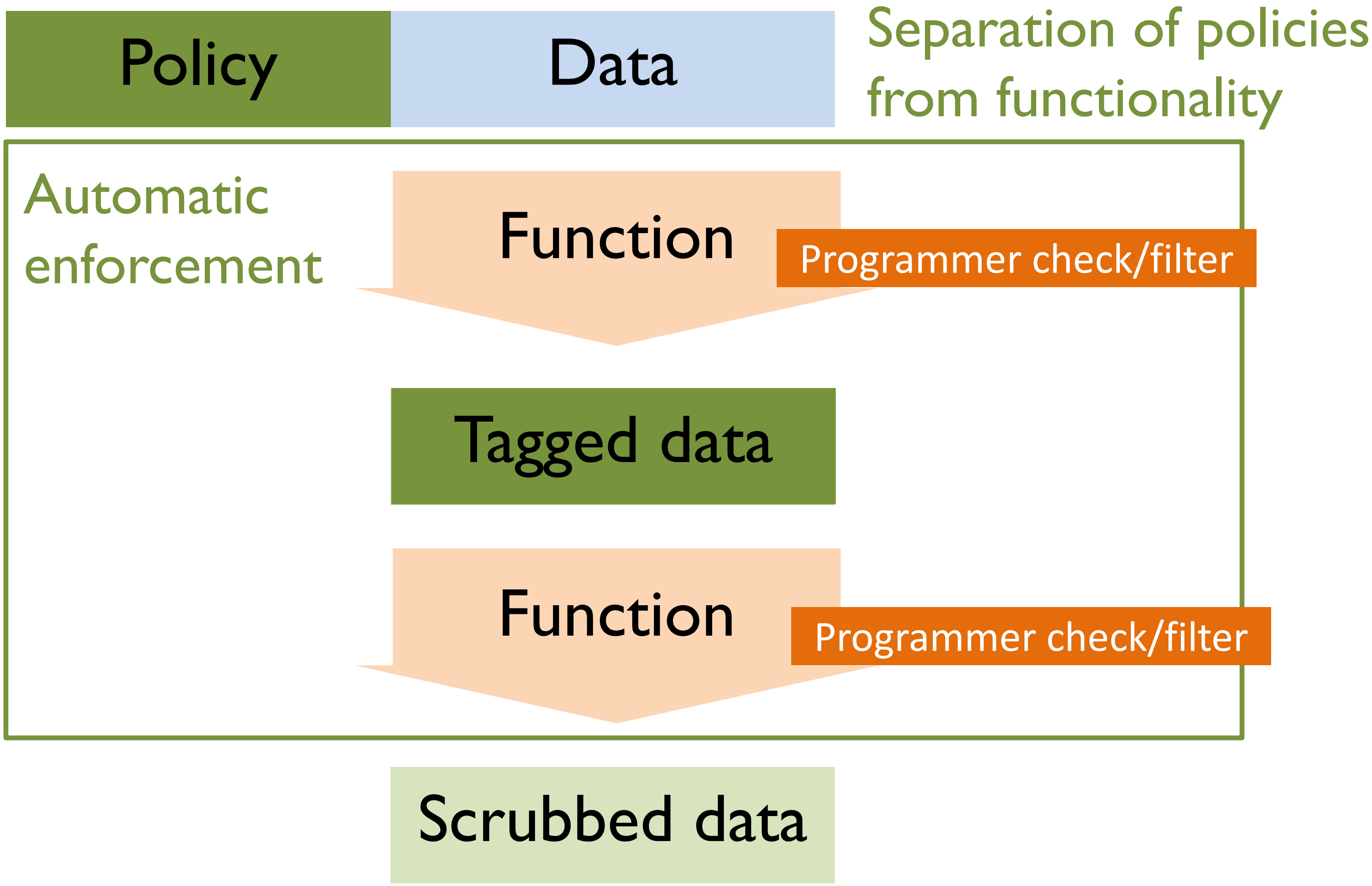
Make it easier for the **programmer** to preserve **confidentiality** of user data.

What's Hard?



Functionality and policy are intertwined.

Our Solution



Policy

Data

Separation of policies from functionality

Automatic enforcement

Function

Programmer check/filter

Tagged data

Function

Programmer check/filter

Scrubbed data

Jeeves Goal

State of the Art

```
def getLocation (user: User) (viewer: User)
  : Location = {
  if (isFriends user viewer) {
      if (canSee user.location viewer) {
        user.location;
      } else { scrub(user.location, “Work”); }
  } else { undisclosedLocation; }
}
```

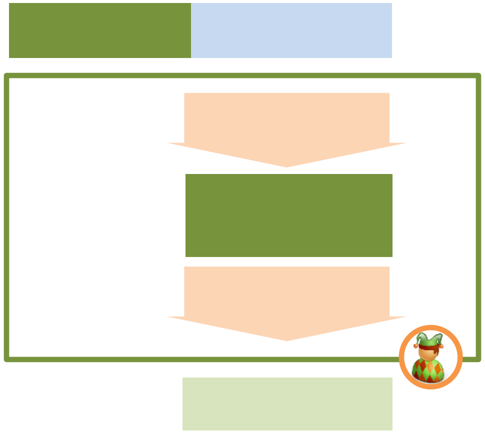
Policy + Functionality

Jeeves

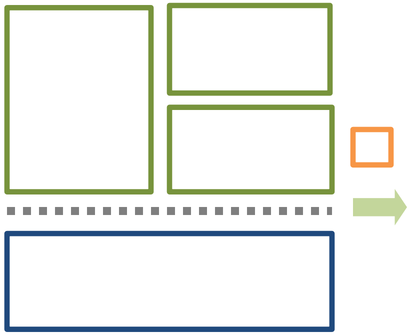
```
def getLocation (user: User): Location =
  user.location
```

Talk Outline

Jeeves language



How it works

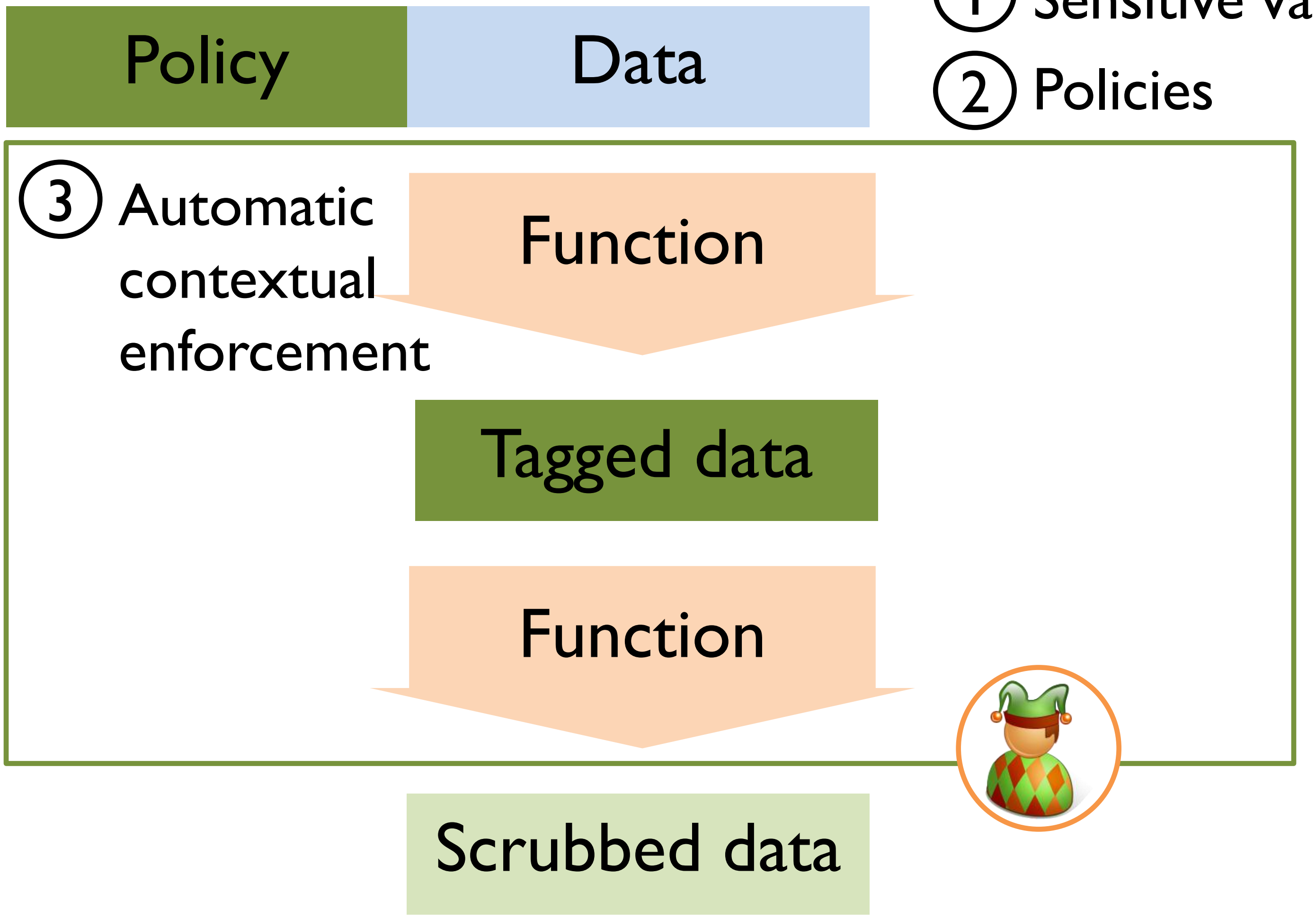


Coding in Jeeves

Jeeves Language

① Sensitive values

② Policies



Policy

Data

③ Automatic contextual enforcement

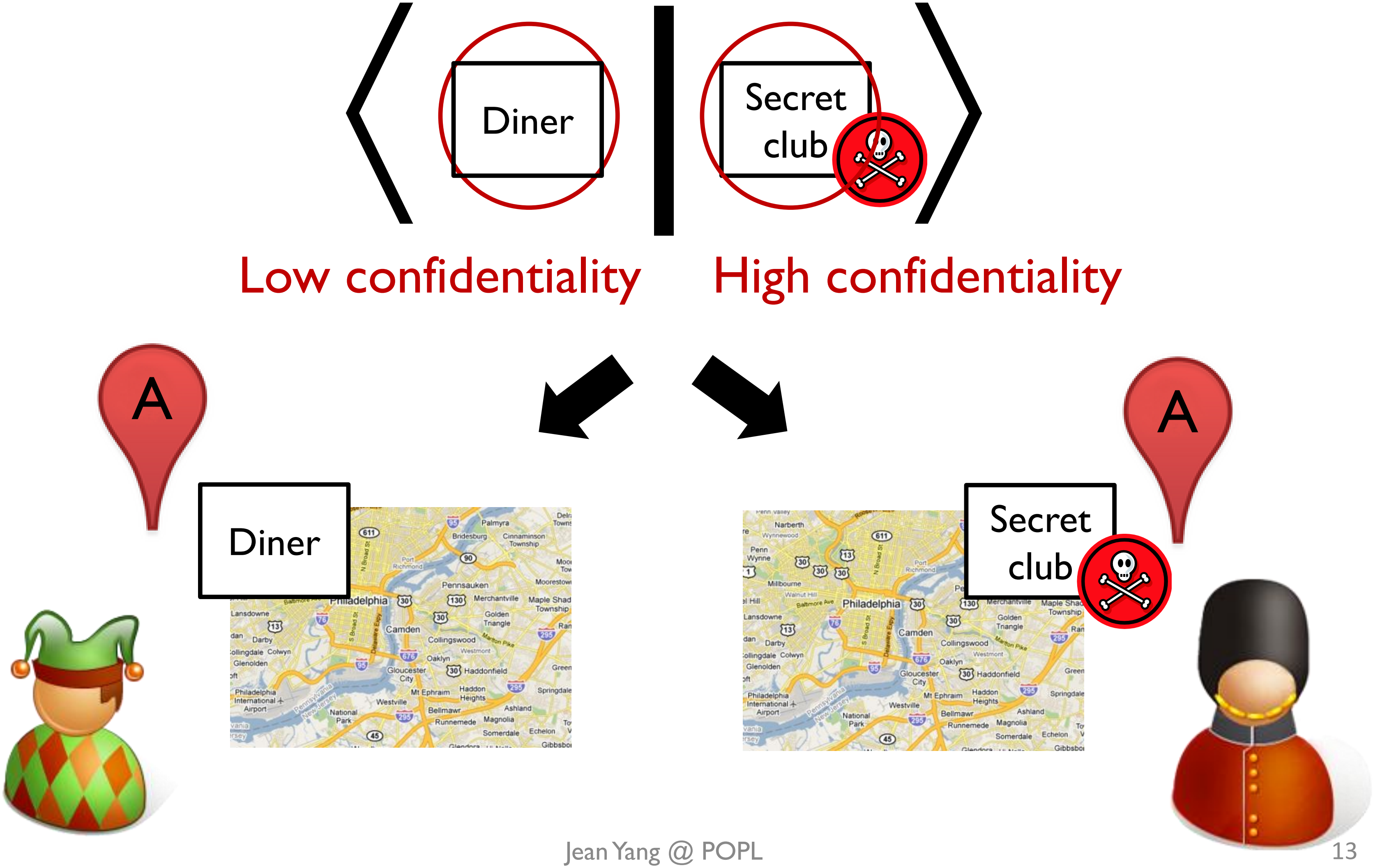
Function

Tagged data

Function

Scrubbed data

Jeeves for Locations



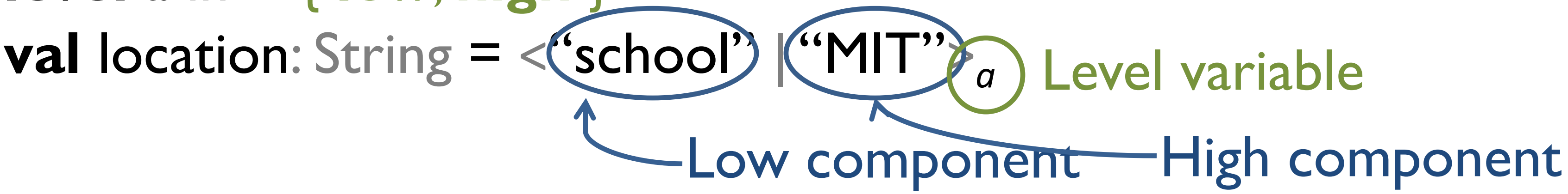
Low confidentiality

High confidentiality

Using Jeeves

Sensitive Values

level a in { **low, high** }



val location: String = <“school” | “MIT”>$_a$

Policies

policy a: **context** != alice ➔ **low**

Core Functionality

val msg: String = “Alice is at ” + location

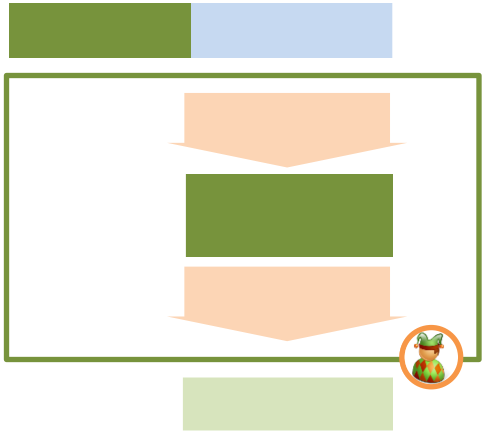
Contextual Enforcement

print {alice} msg /* *“Alice is at MIT”* */

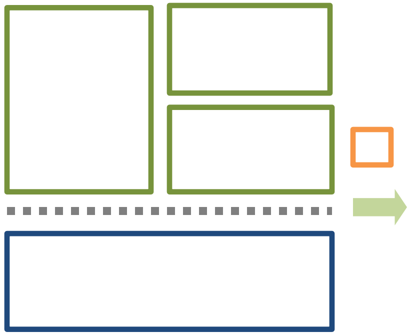
print {bob} msg /* *“Alice is at school”* */

Talk Outline

Jeeves language

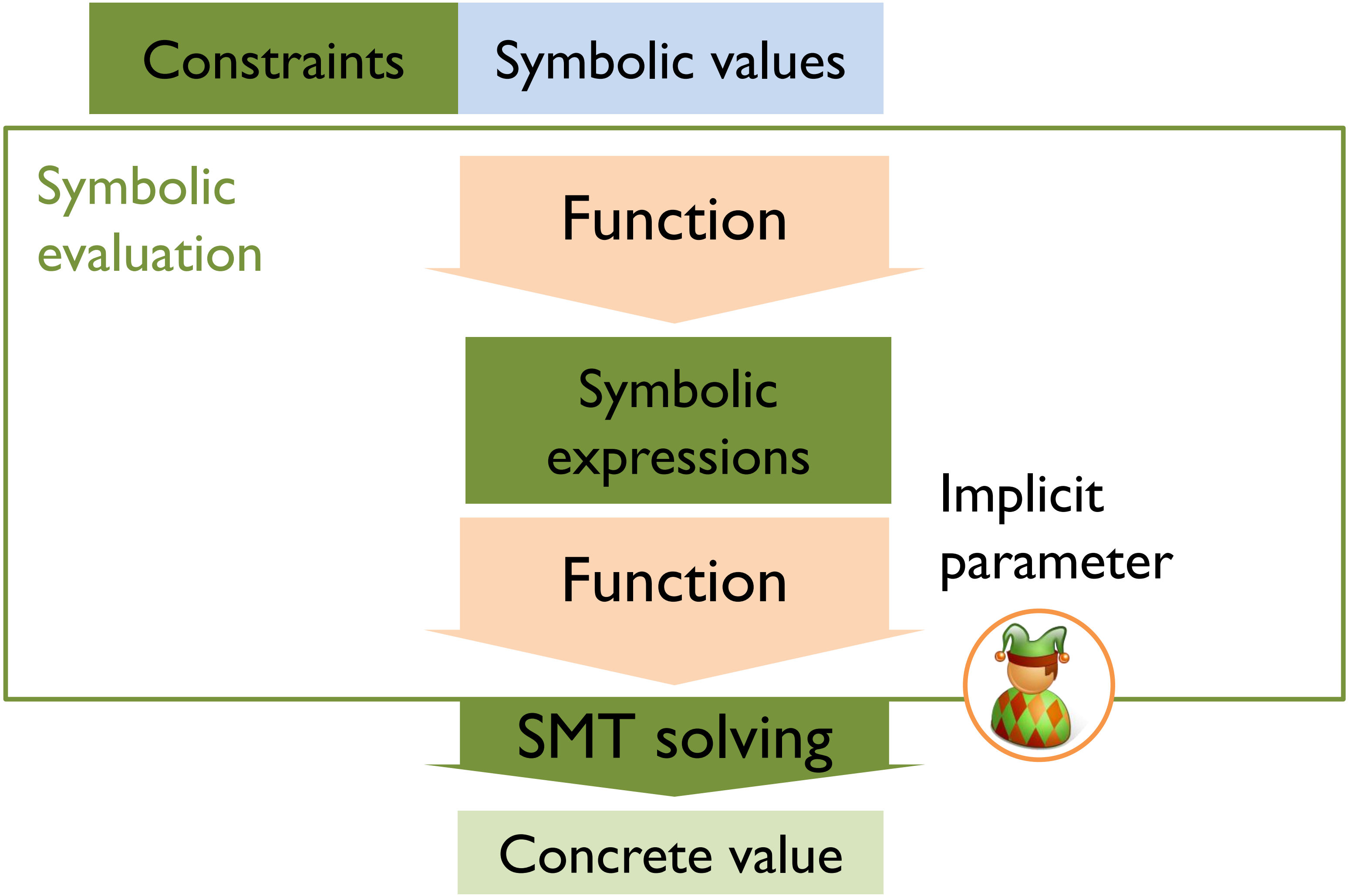


How it works



Coding in Jeeves

How Jeeves Works



Constraints

Symbolic values

Symbolic evaluation

Function

Symbolic expressions

Function

Implicit parameter

SMT solving

Concrete value

Representing Sensitive Values in Jeeves

Without Jeeves

Name	Location
Alice	MIT
Bob	POPL
Claire	POPL

Jeeves

Name	Location
Alice	$\langle\, ? \mid \mathrm{MIT} \rangle_a$ Policy
Bob	POPL
Claire	$\langle\, ? \mid \mathrm{POPL} \rangle_b$ Policy

Symbolic Evaluation for Information Flow

Name	Location
Alice	$\langle \; | \; \rangle_a$ 🟩
Bob	POPL
Claire	$\langle \; | \; \rangle_b$ 🟩

How many people are at POPL?

$$1 + ((x_1 = \mathrm{POPL})\ ?\ 1 : 0) + ((x_2 = \mathrm{POPL})\ ?\ 1 : 0)$$

Runtime Environment

context != alice → a = low

... → b = low

Outputs computed from sensitive values are **symbolic** & concretized under the policy environment.

Jeeves Non-Interference Guarantee

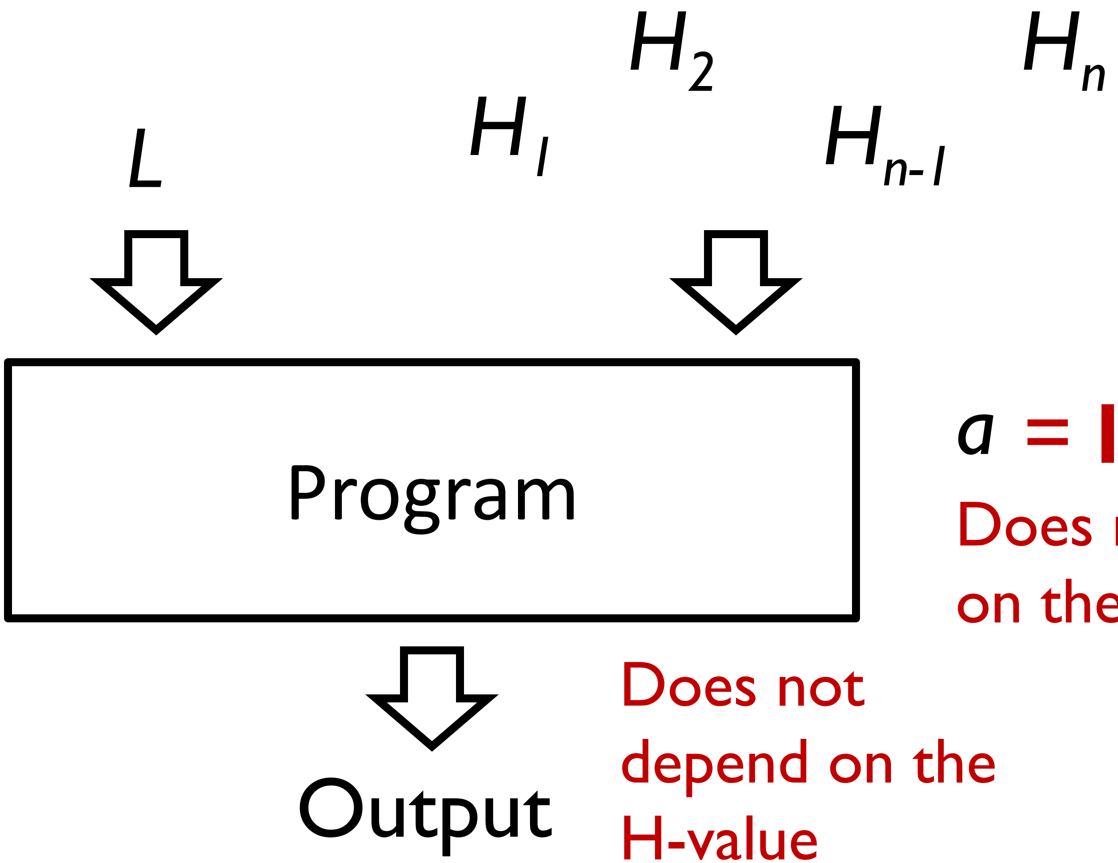
Consider the **sensitive value**

$$\langle \boxed{L} \mid \boxed{H} \rangle_{\boxed{a}}$$ Level variable

Low component High component

Given a fixed *L*, all executions where *a* must be **low** produce equivalent outputs no matter the value of *H*.

Standard Non-Interference



H_2 H_n

L H_1 H_{n-1}

Program

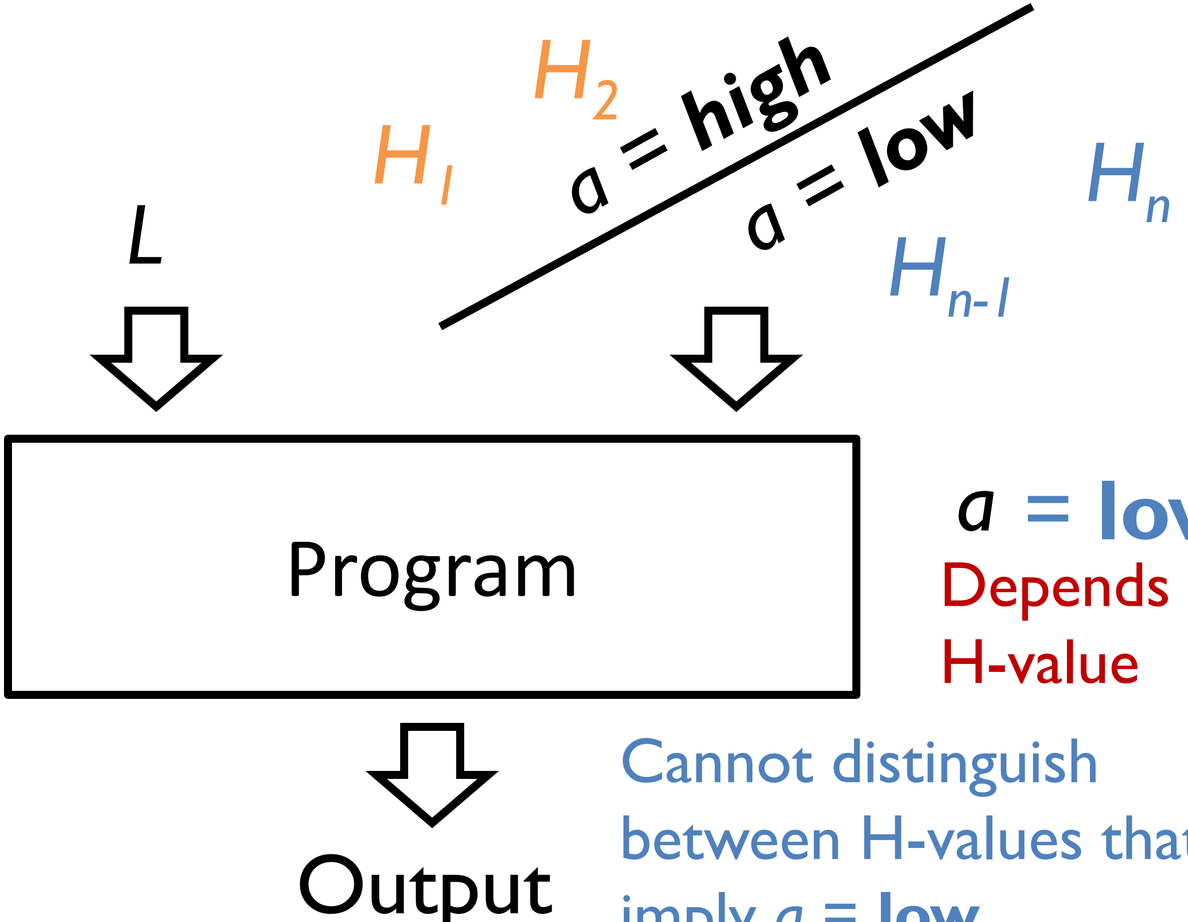
Output

a = **low**

Does not depend on the H-value

Does not depend on the H-value

Jeeves Non-Interference

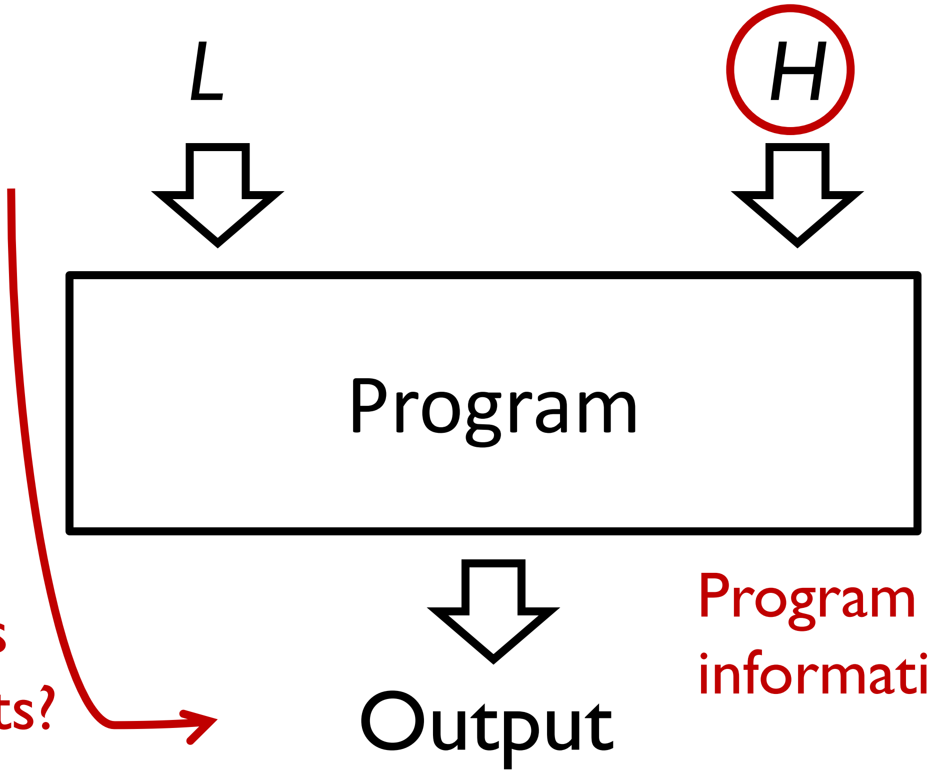


a = **low**

Depends on the H-value

Cannot distinguish between H-values that imply a = **low**

Jeeves Non-Interference

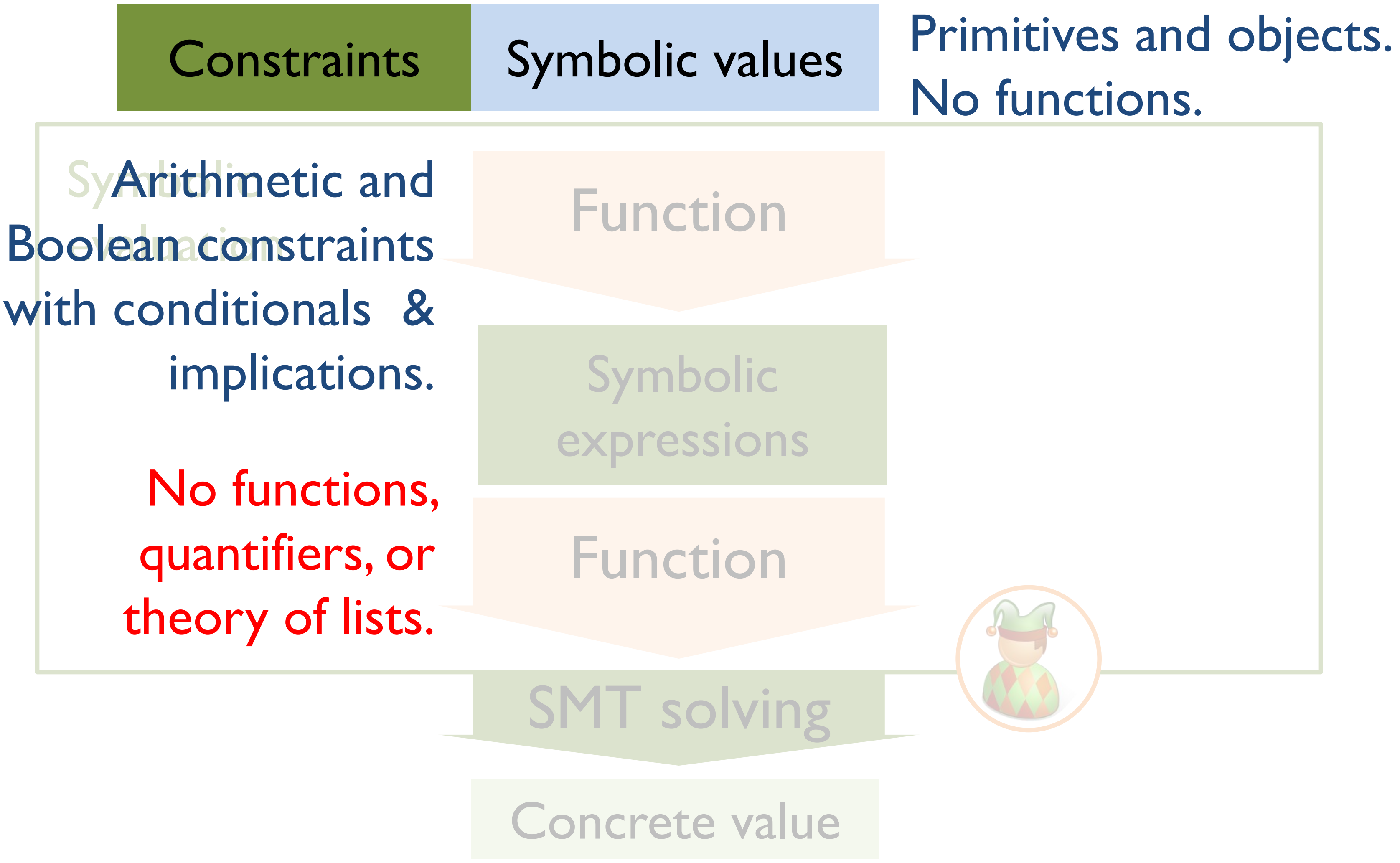


a = **low**

Program does not leak information about *H*.

Programs to outputs?

Language Restrictions

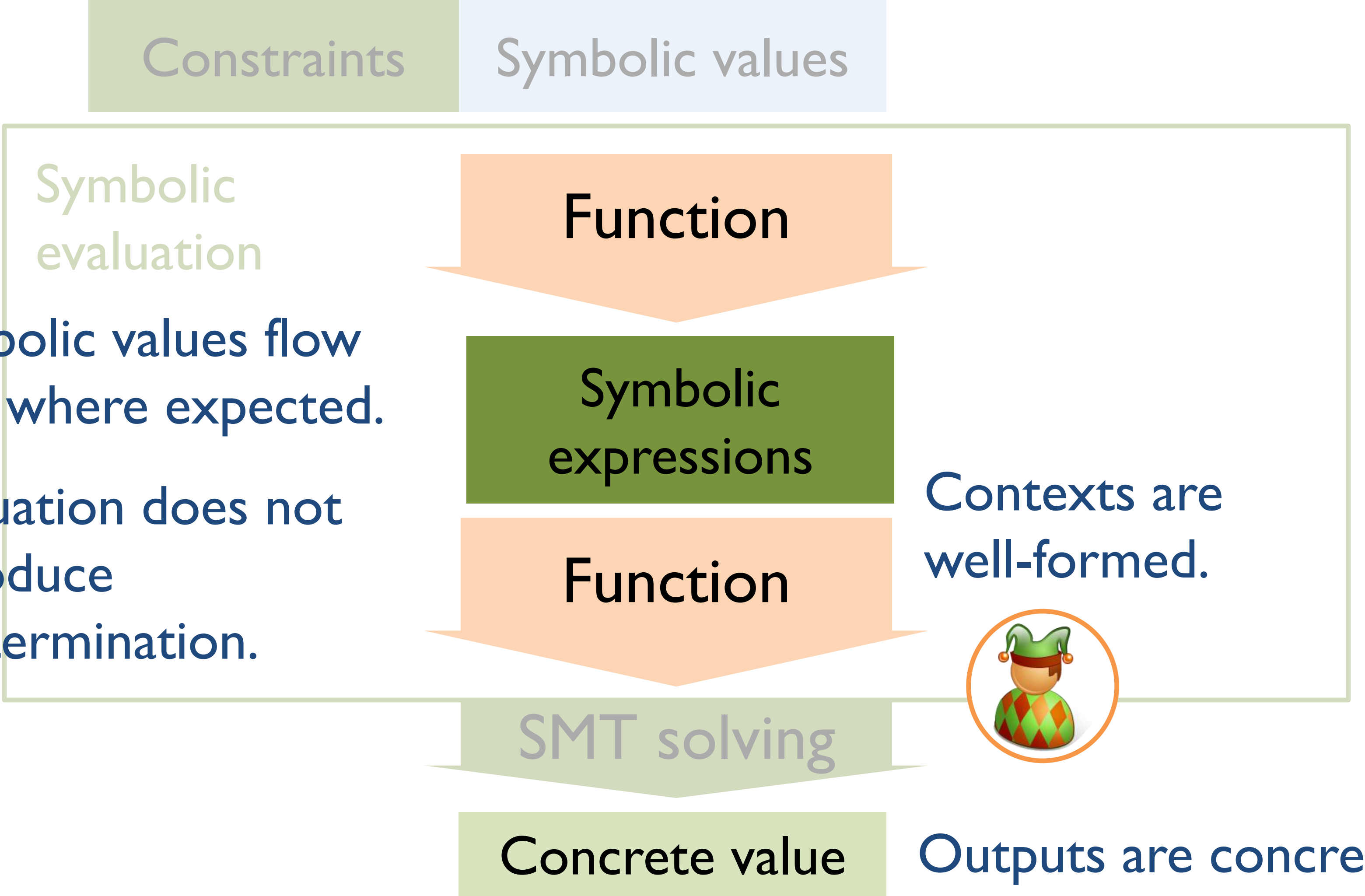


Primitives and objects.
No functions.

Arithmetic and Boolean constraints with conditionals & implications.

No functions, quantifiers, or theory of lists.

Static Checks



Constraints

Symbolic values

Symbolic evaluation

Function

Symbolic expressions

Function

SMT solving

Concrete value

Symbolic values flow only where expected.

Evaluation does not introduce nontermination.

Contexts are well-formed.

Outputs are concrete.

Stateful Policies

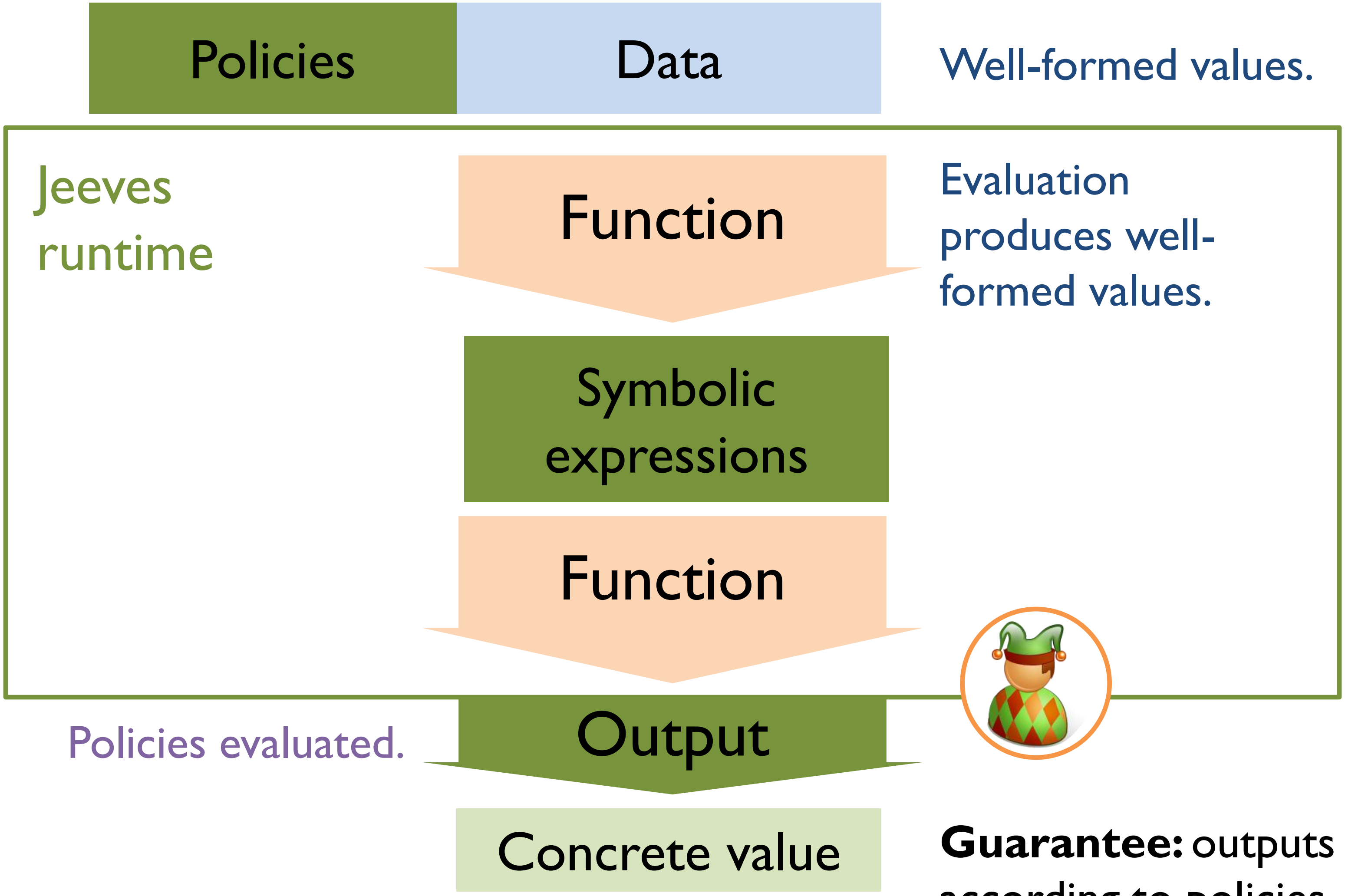
Only people **near me** can see my location.
A
Secret club
Alice

```
policy a:
  (distance context alice
    ≥ radius ) → low
```

But Alice’s location is changing…

Jeeves: Delay policy evaluation until **output**.

Jeeves System



Policies

Data

Well-formed values.

Jeeves runtime

Function

Evaluation produces well-formed values.

Symbolic expressions

Function

Output

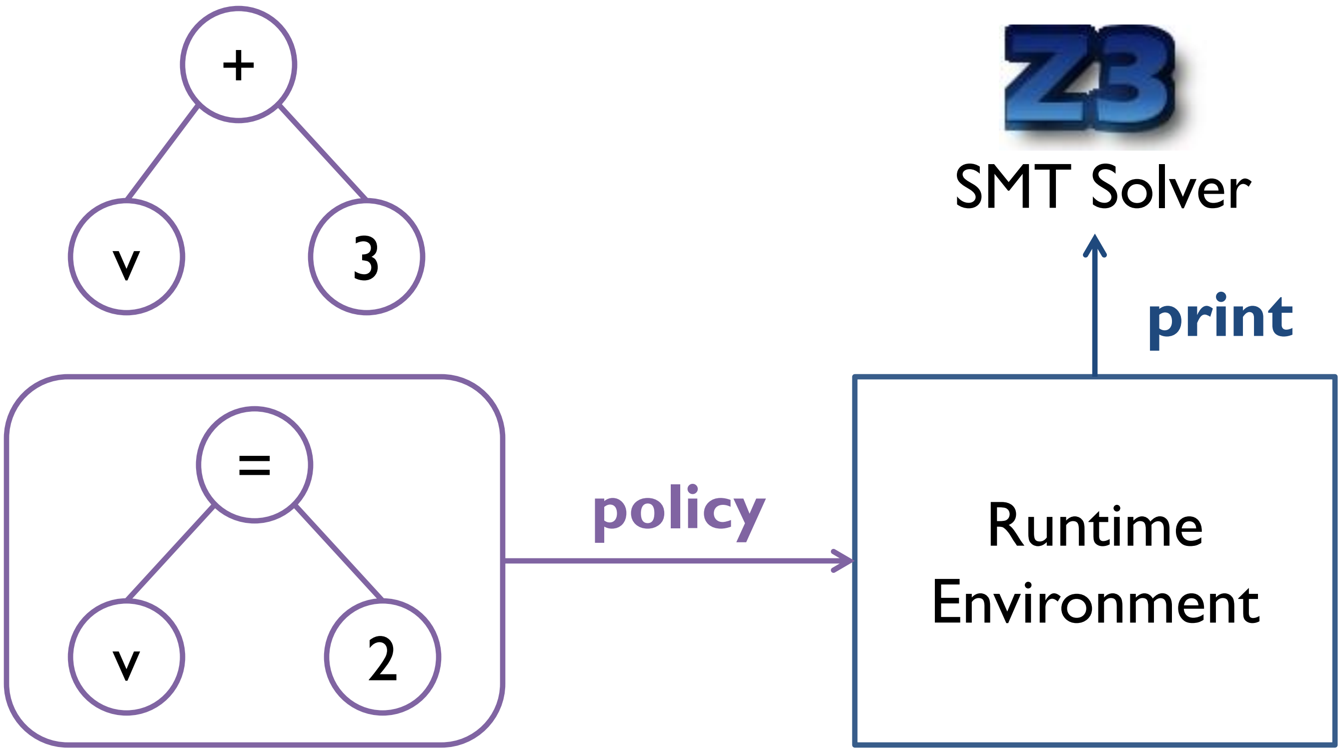
Policies evaluated.

Concrete value

Guarantee: outputs shown according to policies. ✓

Scala Implementation

Overload operators to
create symbolic expressions.



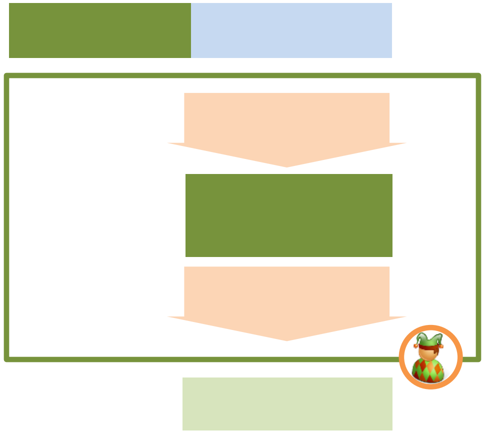
Use an SMT
solver as a
model finder.

Delay
evaluation of
policies until
output.

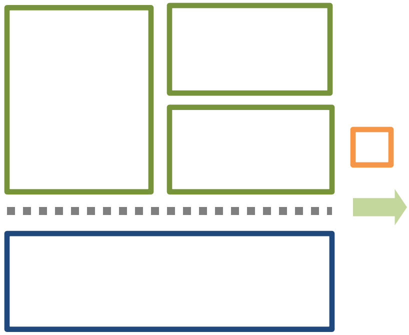
Propagate policies.

Talk Outline

Jeeves language

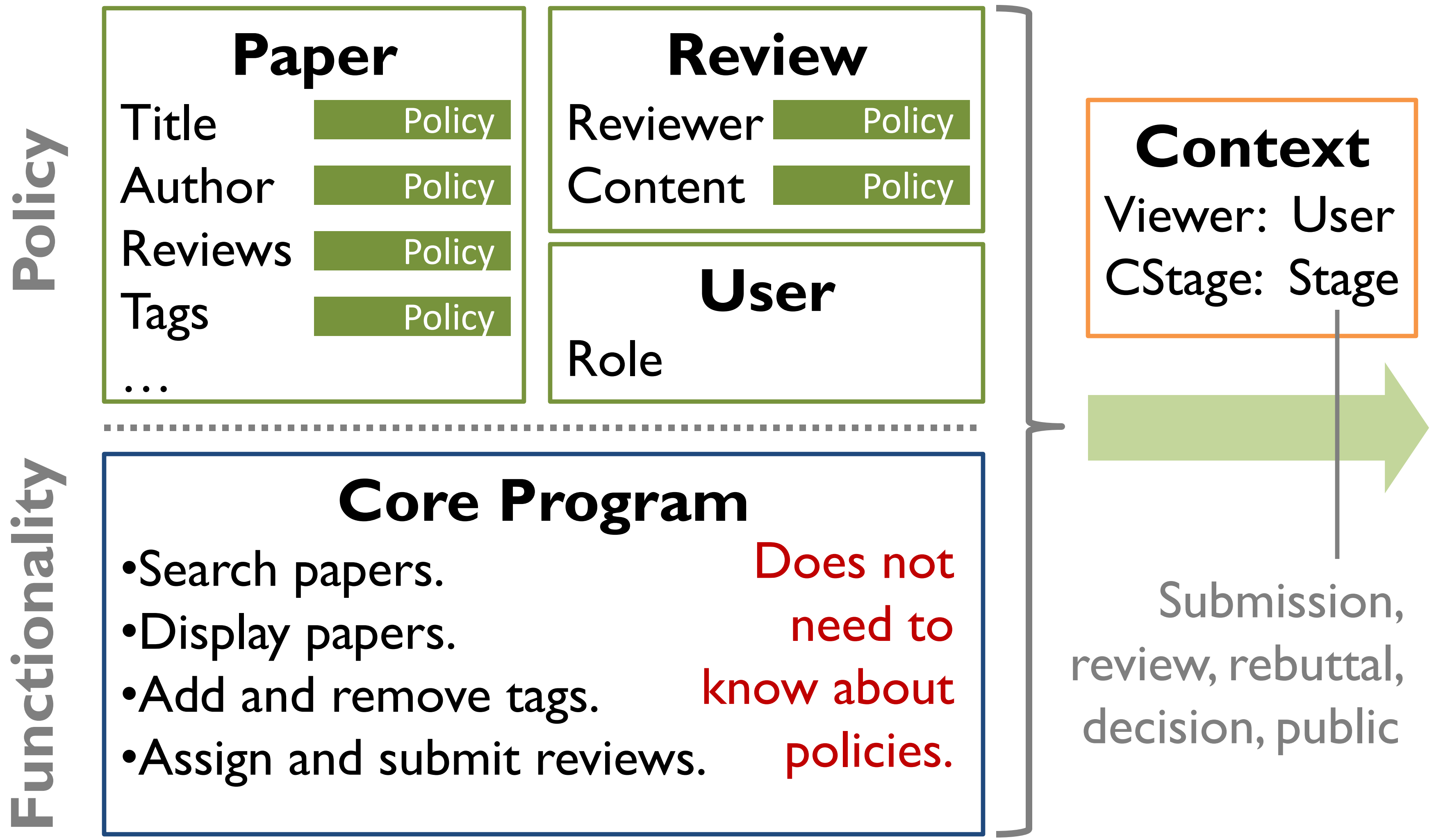


How it works



Coding in Jeeves

JConf Architecture



Policy

Paper

Title	Policy
Author	Policy
Reviews	Policy
Tags	Policy
…	

Review

Reviewer	Policy
Content	Policy

User

Role

Functionality

Core Program

- Search papers.
- Display papers.
- Add and remove tags.
- Assign and submit reviews.

Does not need to know about policies.

Context

Viewer: User

CStage: Stage

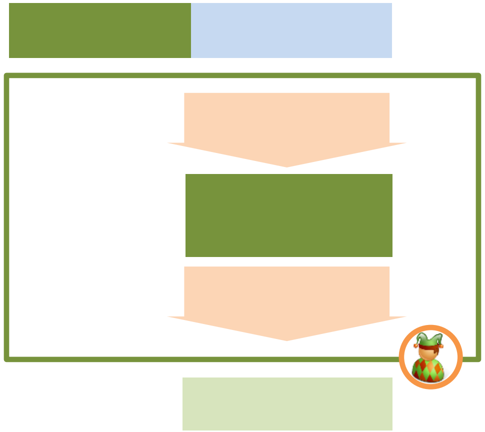
Submission, review, rebuttal, decision, public

Functionality vs. Policy

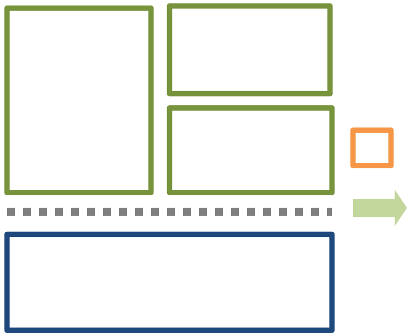
File	Total LOC	Policy LOC
ConfUser.scala	59	17
PaperRecord.scala	103	48
PaperReview.scala	21	11
ConfContext.scala	6	0
Backend	123	0
Frontend (Scalatra)	161	0
Total	**473**	**76**

Conclusions

The Jeeves language: pushing responsibility of privacy to the runtime.



How we designed a language with constraints using symbolic evaluation to provide execution guarantees.



Evaluation of Jeeves in practice: conference management example.

Website: sites.google.com/site/jeevesprogramming
Google Code: code.google.com/p/scalasmt
Contact: jeanyang@mit.edu