Symbolic Faceted Execution

(possibly with…) Kris Micinski

What does the following function compute?

True

False

```
function(x) {
    y = true;
    z = true;
    if (x)
      y = false
    if (y)
      z = false
    return z;
}
```

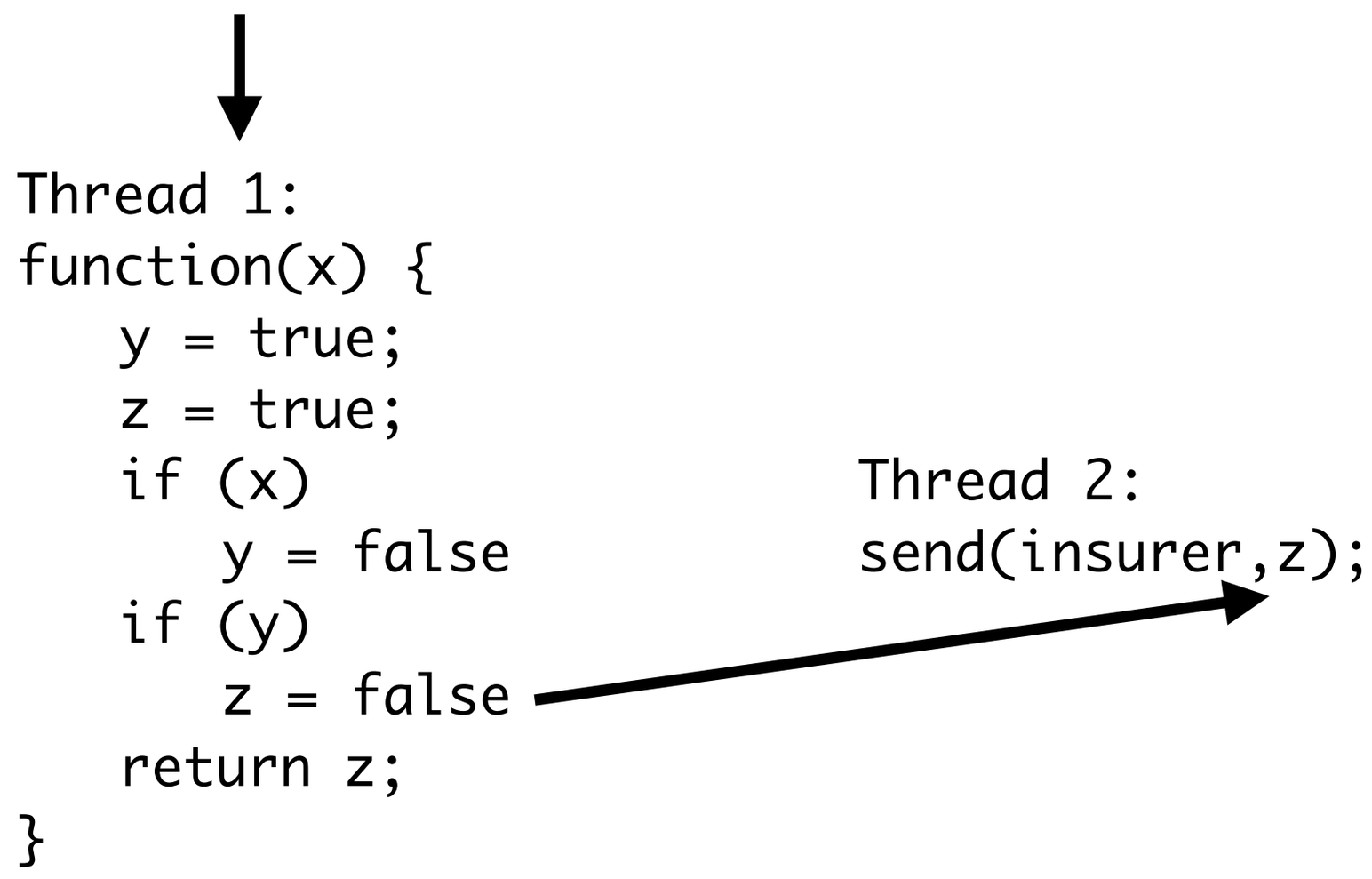
Let's pretend the observer can see z…

True

False

Should be identity, but apply it to a private variable…

Do you have prior medical conditions???



Ways to tame this

- For a “public” observer:
  - Pretend that the input x was NULL (⊥)
  - Run the program and propagate ⊥
- For a “private” observer, run program in separate addr space
- Run the program twice
  - Give public view to public (when sending network data)
  - Keep the private view inside

Running with ⊥

Treat ⊥ as false

I get `False` **no matter what**…

```
function(⊥) {
  y = true;
  z = true;
  if (⊥)
    y = false
  if (y)
    z = false ☑
  return z;
}
```

`False`

Private

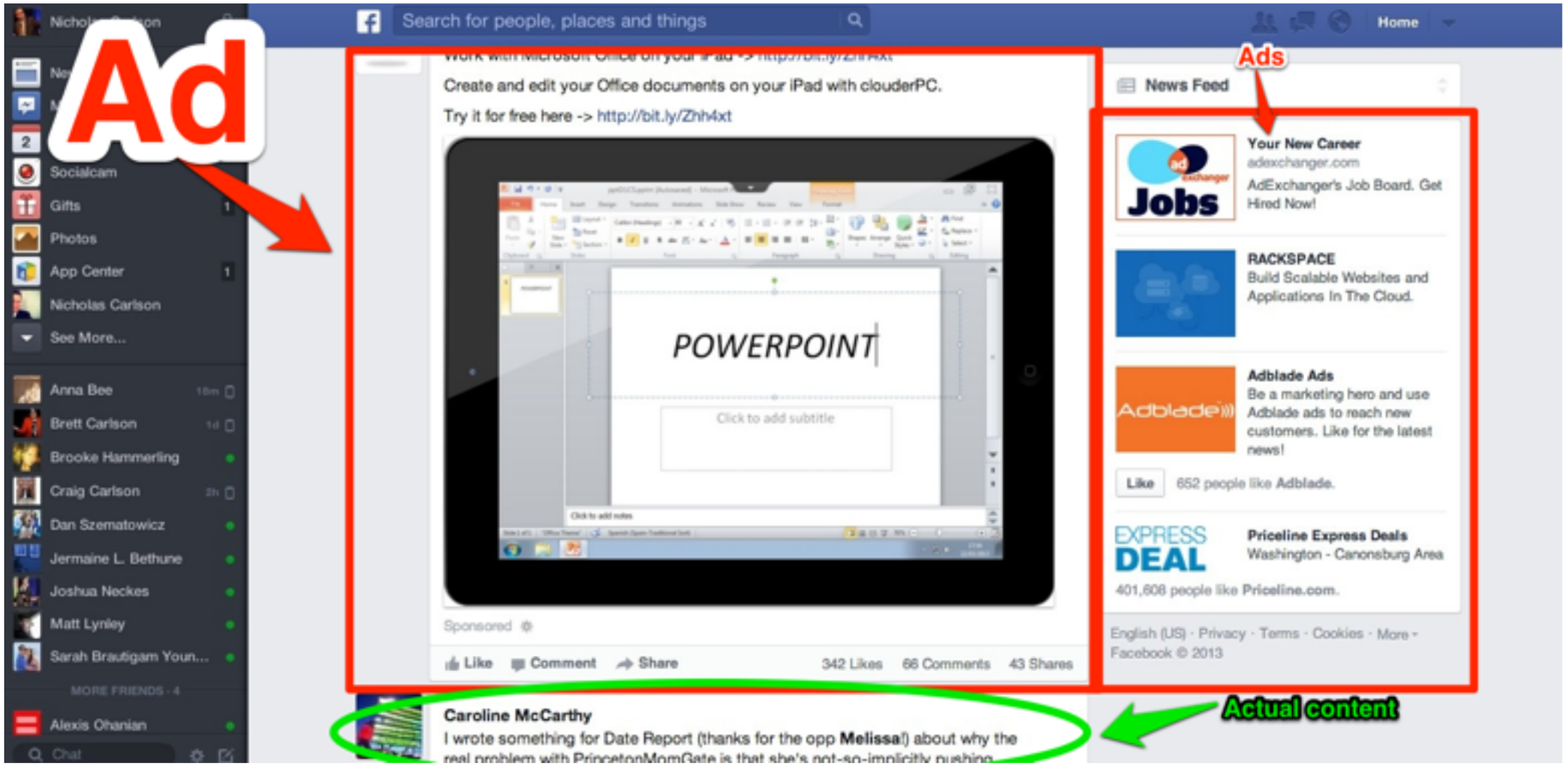
Unclassified

Public

Public sees computation as if it had no secret input

Multiple principles

Code from lots of different people running in same browser (mashups)



Execute multiple copies

- Need to execute multiple copies of program
  - So z stays distinct for the rest of execution
- Now consider we have more than just one observer
  - Facebook, Yahoo Ads, and random.com ads
  - Now need to execute many copies of program
- Doesn't scale very well! (In general 2^n)

Faceted execution

- Key concept: proxy data with tags
  - (principal, private view, low view)
  - **If I'm principal k, I see the result V_h otherwise I see V_l**

$$\{ k \mid V_h , V_l \}$$

Principal (e.g., **FB**, foo.com)

The way **k** sees the result of the computation

The way **the rest of the world** sees the computation

```
function(x = <FB | true , ⊥>) {
   y = true;
   z = true;
   if (x)
      y = false
   if (y)
      z = false
   return z;
}
```

```
function(x = <FB | true , ⊥>) {
  y = true;
  z = true;
  if (x)
    y = false
  if (y)
    z = false
  return z;
}
```

What happens **here**

Execute **twice**:

- Once for private facet
- Once for public facet
- join back together

Private

- x is true
- y is false

Public

- x is false (⊥)
- y is true

Join them! y becomes <FB | false , true>

```
function(x = <FB | true , ⊥>) {
   y = true;
   z = true;
   if (x)
      y = false
   if (<FB | false, true>)
      z = false
   return z;
}
```

Apply trick again…

after if `z = <FB | true, false>`

Using faceted values

- Imagine some malicious code tries to send out **z**
- Should see computation as if private inputs were ⊥
- Use **projection**:

```
send(foo.com, z = <FB | true, false>)
```

Check to see if foo.com is FB

NO!
`false`

- This is a simple trick that solves a big problem
- Some technical details I can help you figure out
  - But works pretty well!
- Also supports **declassification**
- Potential research:
  - might propagate facet to place where it **isn't needed**
  - Can eliminate facet that is of form $\langle\ k\ \mid\ v_1\ ,\ v_1\ \rangle$
  - Also places where facet is never projected?

Silly example of when facets are unnecessary…

```
function(x = <FB | v , ⊥>) {
  y = true;
  if (x+x-2*x = 0)
    y = true
  else
    y = false
  return <FB | false , false>;
}
```

Probably gets more interesting with more interesting domains / higher order control flow…

Your project

- Read and figure out the faceted execution paper
- Implement faceted execution in Redex
- Research: eliminate unnecessary facets
- **Use symbolic execution to prove agreement**
- **Show on various examples**
- Prove implementation is correct
- Extend actual implementation

Thanks!

- I look forward to talking to you about this!

- Please talk to me if you're interested