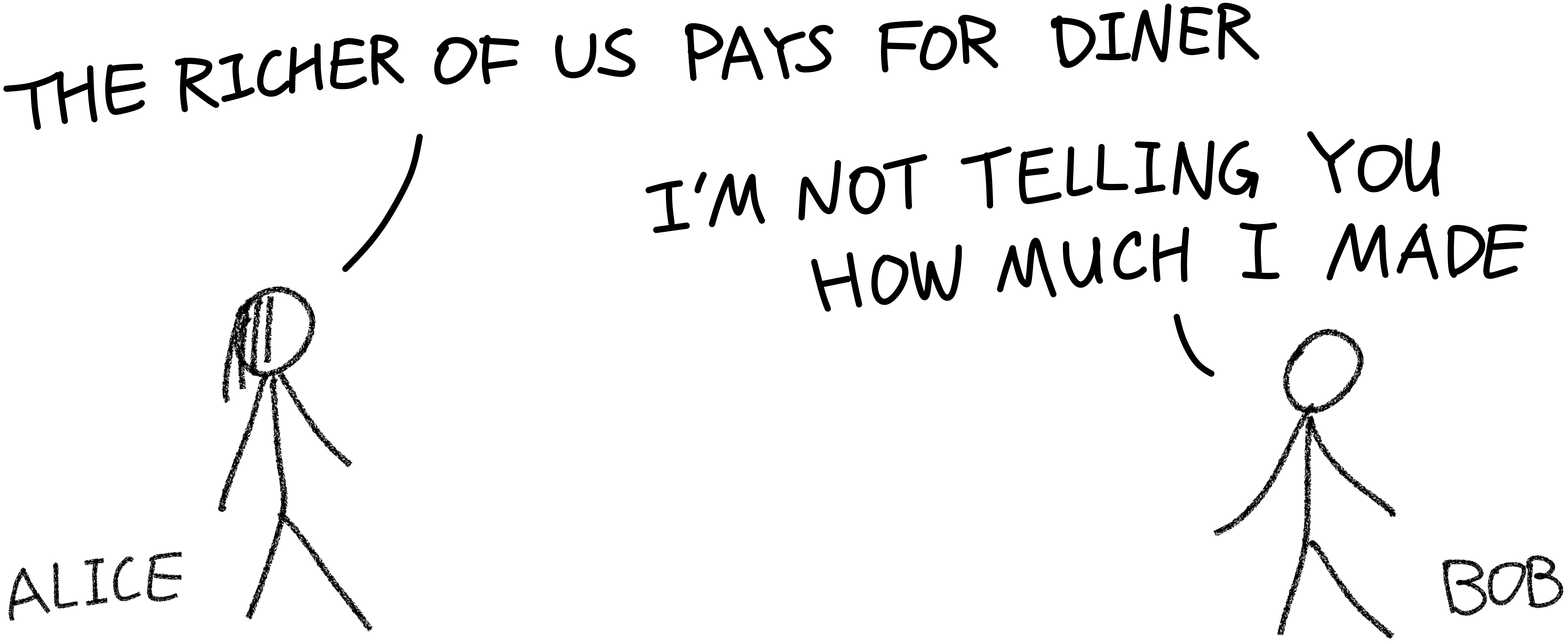
Policy-Agnostic Oblivious Computation

Qianchuan Ye

University at Buffalo
Department of Computer Science and Engineering
School of Engineering and Applied Sciences

A Mental Game



Andrew Yao. 1982. Protocols for Secure Computations

MPC To The Rescue

Secure multi-party computation (MPC) allows multiple parties to perform a joint computation while keeping their sensitive data secure

MPC To The Rescue

Secure multi-party computation (MPC) allows multiple parties to perform a joint computation while keeping their sensitive data secure

This can be achieved by cryptographic protocols, such as Yao's Garbled Circuits and other protocols based on secret-sharing schemes

Oblivious Computation

- Computation that does not leak private information, directly or indirectly

Oblivious Computation

- Computation that does not leak private information, directly or indirectly
- Secure multi-party computation, fully homomorphic encryption, virtualization, secure CPU, etc

MOZILLA

Privacy Preserving Attribution for Advertising

FEBRUARY 8, 2022 MARTIN THOMSON

Advertising provides critical support for the Web. We've been looking to apply privacy preserving advertising technology to the attribution problem, so that advertisers can get answers to important questions without harming privacy.

advertising campaigns are working. Attribution
understand how their advertising campaigns
niques also help publishers understand how
ution is crucial to advertising, current attribution

ing with a team from Meta (formerly Facebook)
ersion measurement – or attribution – for
ibution, or IPA.

lity to perform attribution while providing
privacy-preserving features. First, it uses
owing any single entity — websites, browser
r behavior. Mozilla has some experience with
ivacy-preserving telemetry. Second, it is an
oduces results that cannot be linked to
ean that IPA cannot be used to track or profile

for advertising businesses in terms of how they
owser attribution options in IPA enable new and
aintaining privacy. The IPA proposal aims to
ures with the *match key* concept, which allows
of entities to cross-device attribution.

e've recently proposed IPA to the Private
Advertising Technology Community Group, or PATCG. PATCG is a group in the W3C

Announcements · October 4, 2022

Our progress on developing and incorporating privacy-enhancing technologies

By Dennis Buchheim, Vice President, Science & Ecosystem

Last year, we shared our longer-term vision on privacy-enhancing technologies and how we believe they will become foundational to the future of personalized advertising experiences. Today, we want to share an update on the progress Meta and the industry have made towards these efforts and how advertisers can get involved.

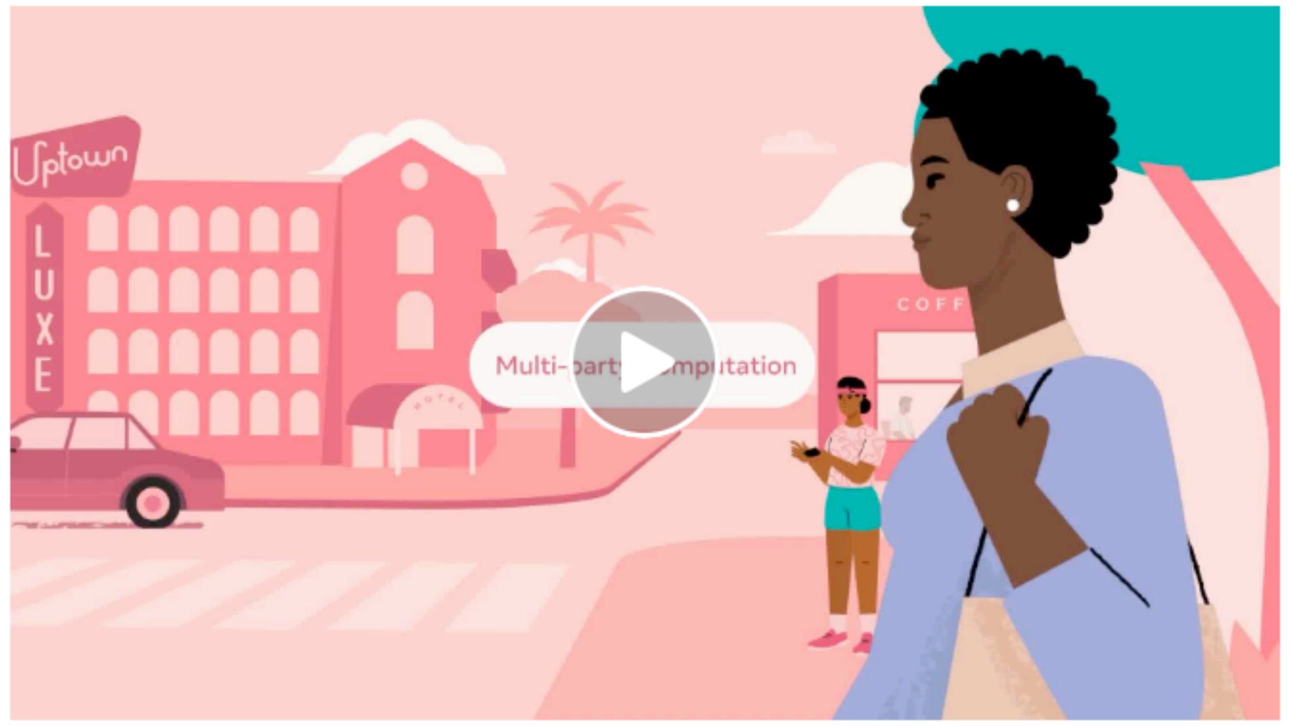
Industry momentum on privacy-enhancing technologies

Industry collaboration on privacy-enhancing technologies is essential for the development of interoperable solutions and a shared set of standards to support a free and open internet. This year, we saw collaboration turn into tangible progress.

First, our proposal with Mozilla on a new privacy-preserving standard for ad measurement, Interoperable Private Attribution (IPA), continues to advance within the World Wide Web Consortium's Private Advertising Technology Community Group (W3C PATCG). The goal of this proposal is to create a new standard for measurement

Incorporating PETs into Meta's portfolio of advertising solutions

Over the past year, we've also made progress developing our own portfolio of PET-based solutions, particularly our Private Lift Measurement product, which uses secure multi-party computation (MPC) to help advertisers understand how their campaigns are performing while limiting what the advertiser and Meta can learn about a person.



For more than a year, we've been testing this solution with advertisers from around the world, gathering feedback and improving the product's performance, and some of our largest clients are now using Private Lift.

For example, a global financial services advertiser, who was not previously using Lift measurement products, tried our Private Lift product to gain a more comprehensive view into the incremental conversions their ads were driving, while keeping their underlying data private. Specifically, the advertiser set up a test in which part of their target audience received an ad and the other part of the audience did not, and then compared conversions to understand what conversions were incremental. Their study found that the test group drove 55% more conversions than the control group, which was valuable insight into how many conversions would not have occurred without advertising on Meta.

Along with Private Lift, we also began testing a new Private Computation solution, known as Private Attribution, which keeps the advertiser's underlying data private by

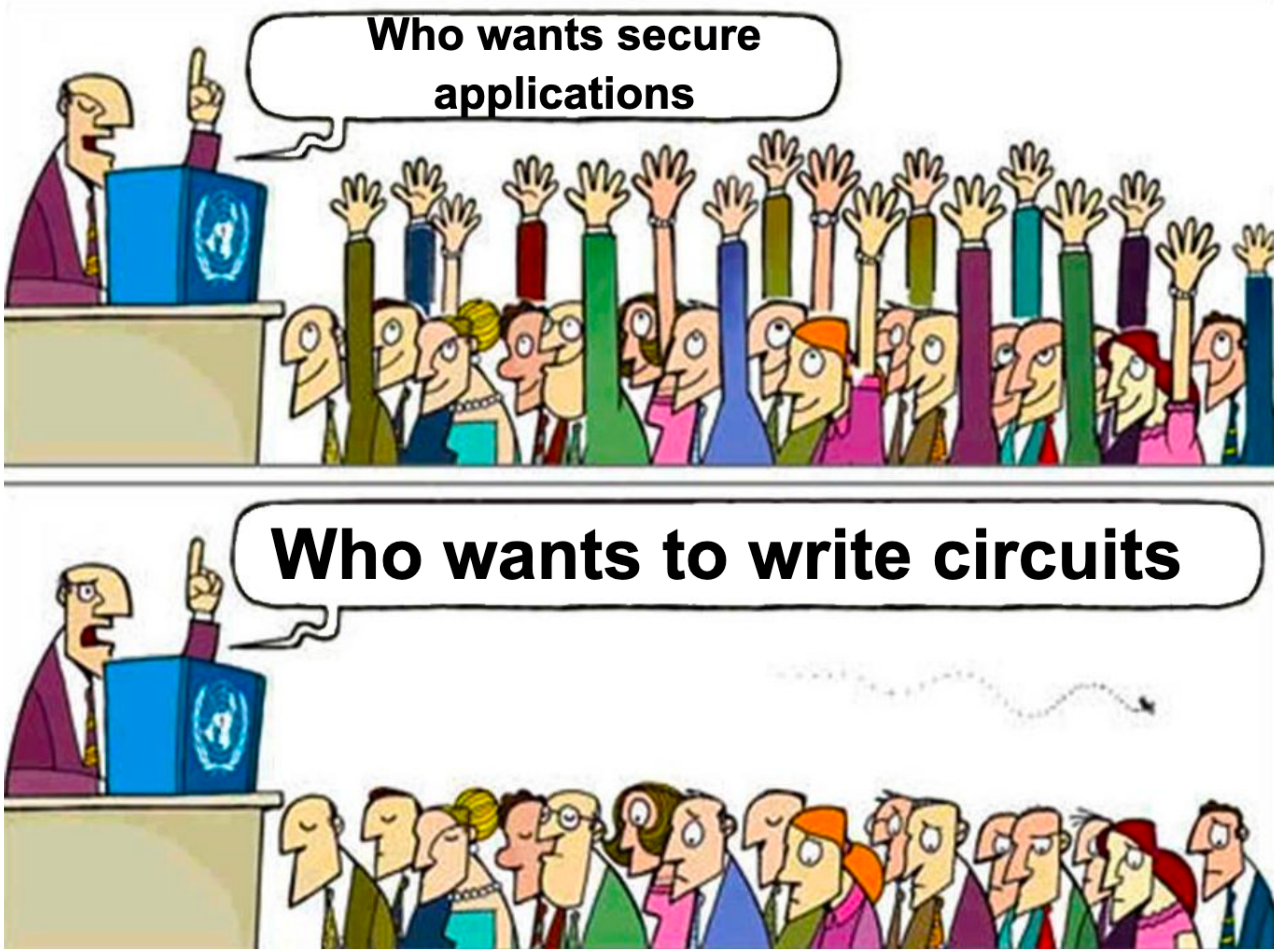
https://blog.mozilla.org/en/mozilla/privacy-preserving-attribution-for-advertising/
https://www.facebook.com/business/news/our-progress-on-developing-and-incorporating-privacy-enhancing-technologies/

Privacy-critical Applications

- Secure auction
- Voting
- Privacy-preserving machine learning
- Statistics about sensitive information

Writing Secure Applications

Writing Secure Applications



High-level Programming Languages for MPC

- Fairplay [Malkhi et al. 2004]
- PICCO [Zhang et al. 2013]
- Obliv-C [Zahur and Evans 2015]
- ObliVM [Liu et al. 2015]
- Wysteria/Wys* [Rastogi et al. 2014, 2019]
- λobliv [Darais et al. 2020]
- Viaduct [Acay et al. 2021]
- Symphony [Sweet et al. 2023]

A Secure Dating App

WHERE SOULMATE

WHERE SOULMATE

Input: Personal Profiles

age = ...
height = ...
weight = ...
salary = ...
job = ...

Input: Preferences (as AST)

your.job ≠ "computer scientist" and
(your.height ≥ 7′ or
my.salary + your.salary > 1M)

true

Nontrivial Data

profile profile

preference preference

Nontrivial Data

$$\overbrace{\text{profile} \quad \text{profile}}^{\text{FLAT}}$$

preference preference

Nontrivial Data

FLAT

profile profile

preference preference

RECURSIVE

Nontrivial Data

profile profile

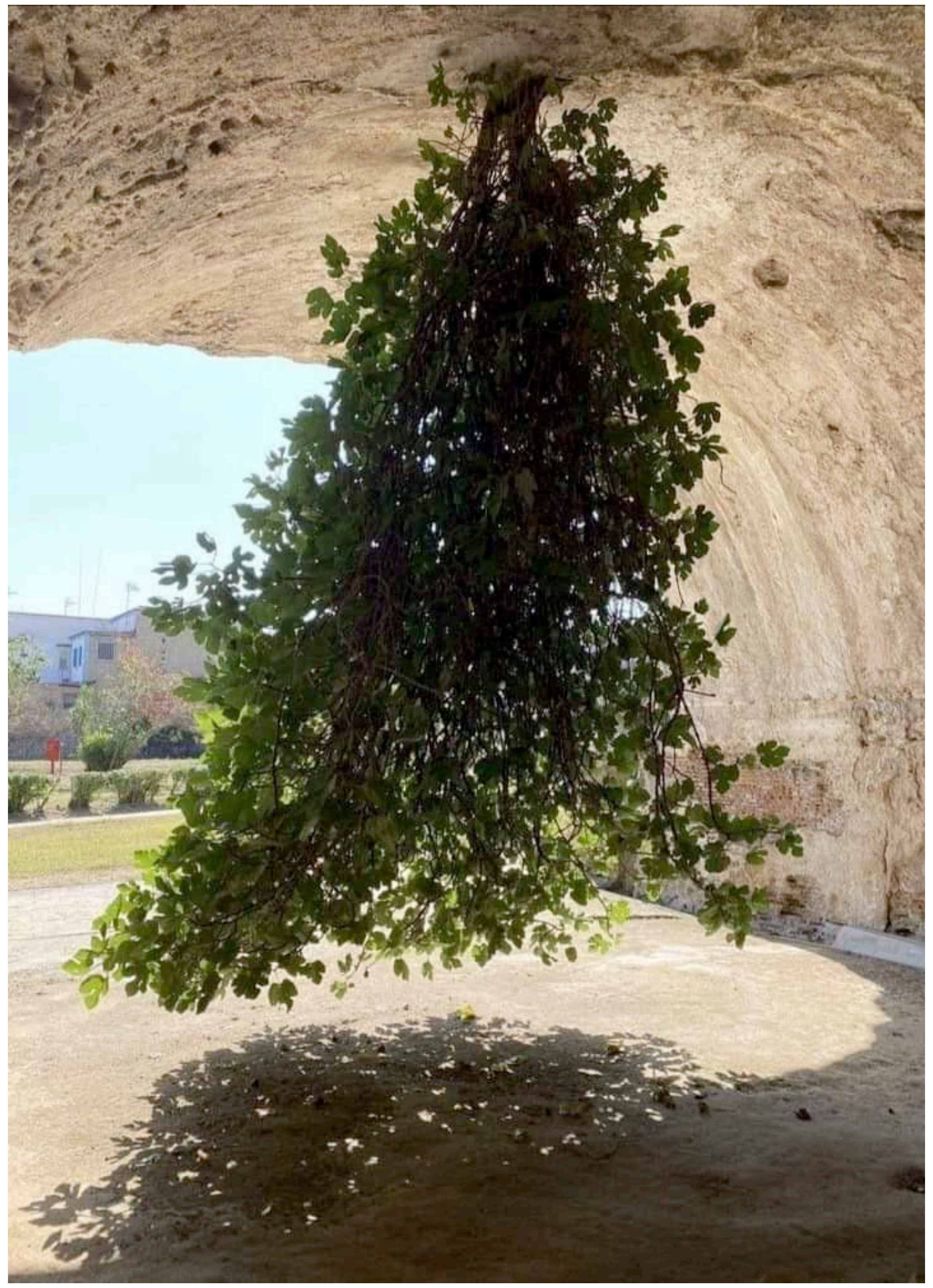
preference preference

MATCH

decision

Gap #1: Private Structured Data

Rich Recursive Data Structures like trees



Gap #2: Complex Policies

Gap #2: Complex Policies

- Go beyond “private or not”

Gap #2: Complex Policies

- Go beyond “private or not”
- Policies can be complex for structured data

Gap #2: Complex Policies

- Go beyond “private or not”
- Policies can be complex for structured data
- A data structure may have multiple policies

Gap #3: Modularity

- Don’t want to enforce policies manually within the application logic

Gap #3: Modularity

- Don’t want to enforce policies manually within the application logic
- Separating privacy policies from program logic

Gap #3: Modularity

- Don’t want to enforce policies manually within the application logic
- Separating privacy policies from program logic
- Allow for writing applications independently of the policies

Gap #3: Modularity

- Don't want to enforce policies manually within the application logic
- Separating privacy policies from program logic
- Allow for writing applications independently of the policies
- Allow for specifying and auditing policies independently of the functionality

Bridging The Gaps

Bridging The Gaps

- **Rich**: functional language with high-level abstractions, e.g., structured data, higher-order functions, and complex policies

Bridging The Gaps

- **Rich**: functional language with high-level abstractions, e.g., structured data, higher-order functions, and complex policies

- **Safe**: no private information is leaked throughout the execution

Bridging The Gaps

- **Rich**: functional language with high-level abstractions, e.g., structured data, higher-order functions, and complex policies
- **Safe**: no private information is leaked throughout the execution
- **Easy**: writing secure applications as easy as writing standard applications

The Ideal

The Ideal

FUNCTIONAL
PROGRAM

The Ideal

POLICY₁

FUNCTIONAL
PROGRAM

The Ideal

POLICY$_1$

FUNCTIONAL
PROGRAM

POLICY$_2$

The Ideal

. . .

POLICY$_1$

FUNCTIONAL
PROGRAM

POLICY$_2$

The Ideal

..

POLICY$_1$ + FUNCTIONAL PROGRAM

SECURE PROGRAM$_1$

POLICY$_2$

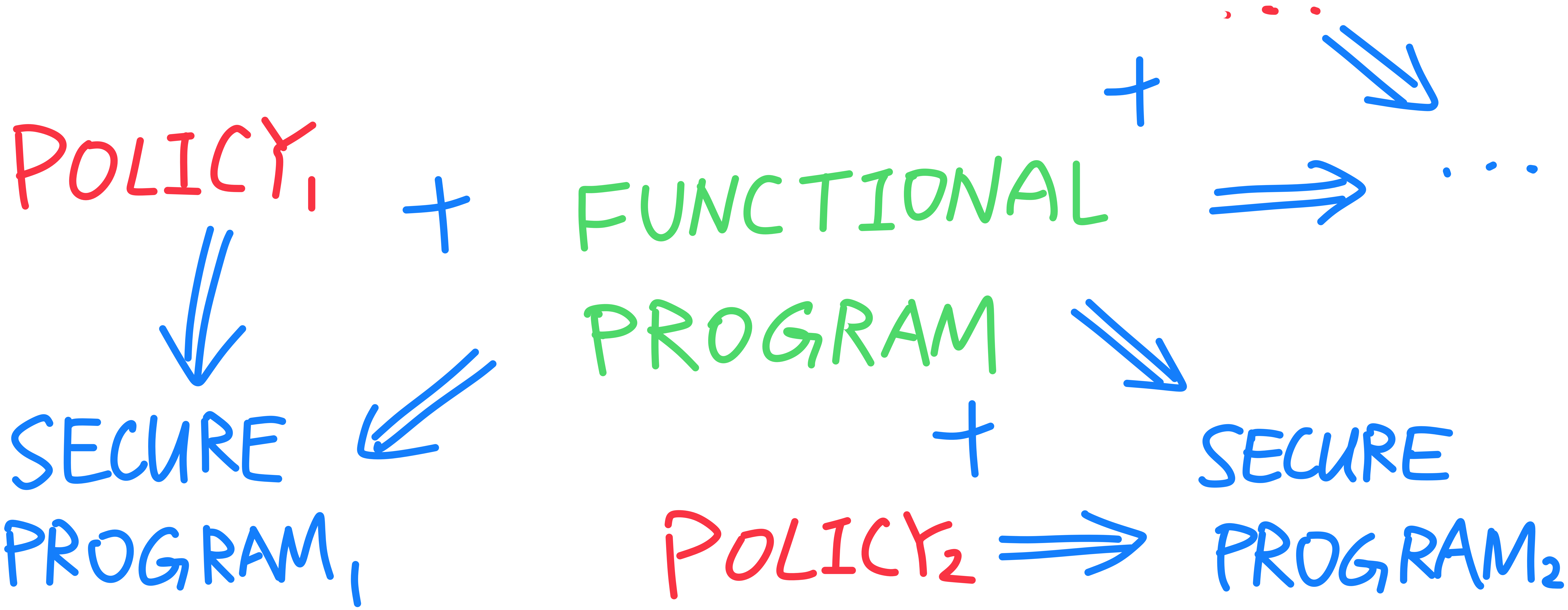
The Ideal

...

POLICY_1 + FUNCTIONAL PROGRAM $\Rightarrow$ SECURE PROGRAM_1

FUNCTIONAL PROGRAM + $\text{POLICY}_2 \Rightarrow$ SECURE PROGRAM_2

The Ideal



POLICY1
+
FUNCTIONAL PROGRAM
+
...
...
SECURE PROGRAM1
+
POLICY2
SECURE PROGRAM2

Overview

Overview

- What are complex privacy policies?

Overview

- What are complex privacy policies?
- How to encode private data and policies? [Rich]

Overview

- What are complex privacy policies?
- How to encode private data and policies? [Rich]
- How to enforce policies? [Safe]

Overview

- What are complex privacy policies?
- How to encode private data and policies? [Rich]
- How to enforce policies? [Safe]
- How to automatically enforce policies? [Easy]

Complex Policies

Policies for Flat Data

```
data patient =
  { id : int;
    age : int;
    height : int;
    weight : int; }
```

Simplest Policy

The whole record is private

```
data patient =
  { id : int;
    age : int;
    height : int;
    weight : int; }
```

Per-Field Policy

Height and weight are private

```
data patient =
  { id : int;
    age : int;
    height : int;
    weight : int; }
```

Either-Or Policy

Either ID or the data is private

```
data patient =
  { id : int;
    age : int;
    height : int;
    weight : int; }
```

*** Based on privacy rules from the Health Insurance Portability and Accountability Act (HIPAA)**

Policies for Recursive Data

```
data tree = Leaf | Node int tree tree
```

Policies for Recursive Data

```
data tree = Leaf | Node int tree tree
```

EMPTY

Policies for Recursive Data

```
data tree = Leaf | Node int tree tree
```

EMPTY

PAYLOAD

Policies for Recursive Data

LEFT SUBTREE

```
data tree = Leaf | Node int tree tree
```

EMPTY

PAYLOAD

Policies for Recursive Data

LEFT SUBTREE

```
data tree = Leaf | Node int tree tree
```

EMPTY

PAYLOAD

RIGHT SUBTREE

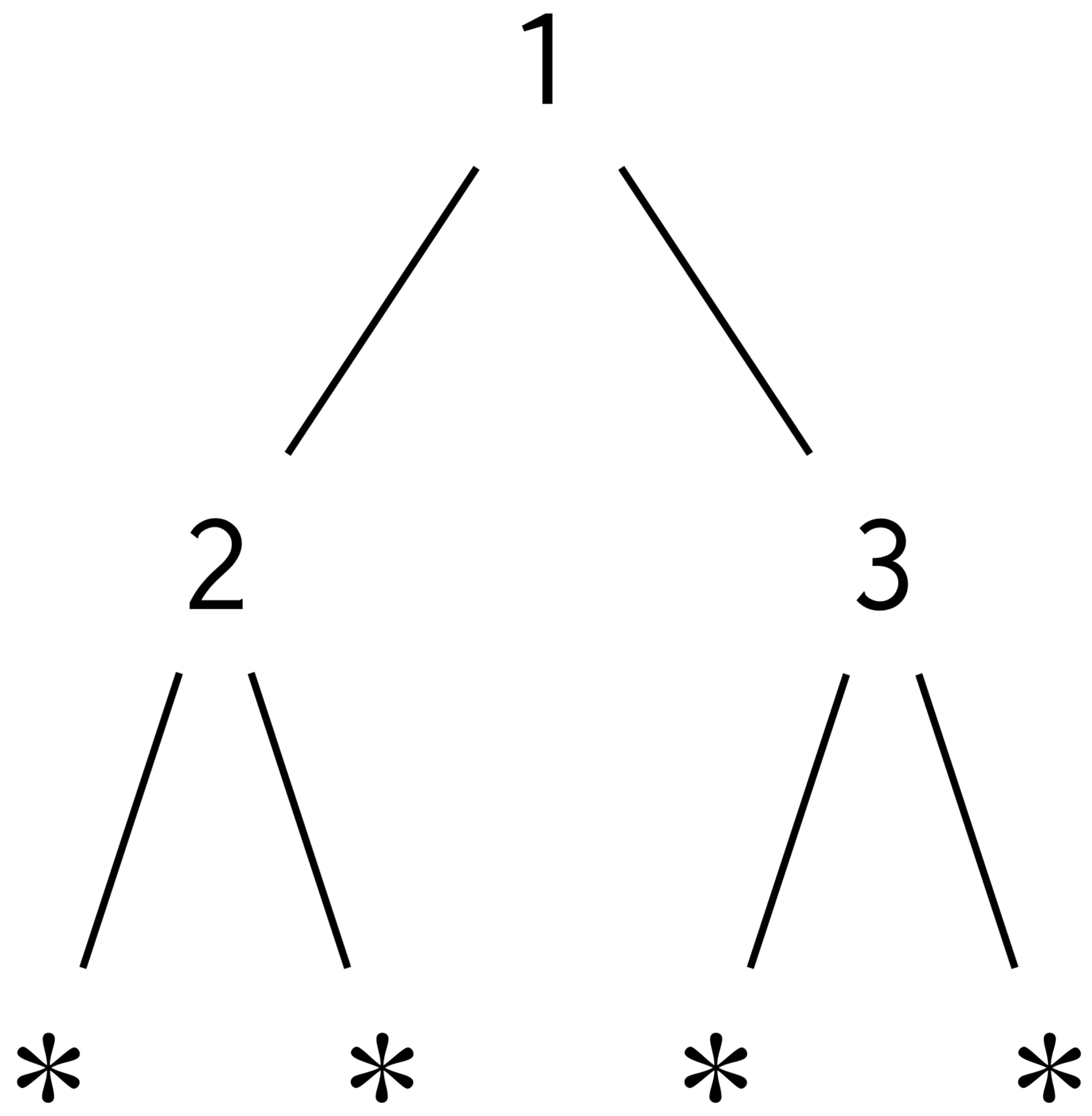
Policies for Recursive Data

`Leaf`

*

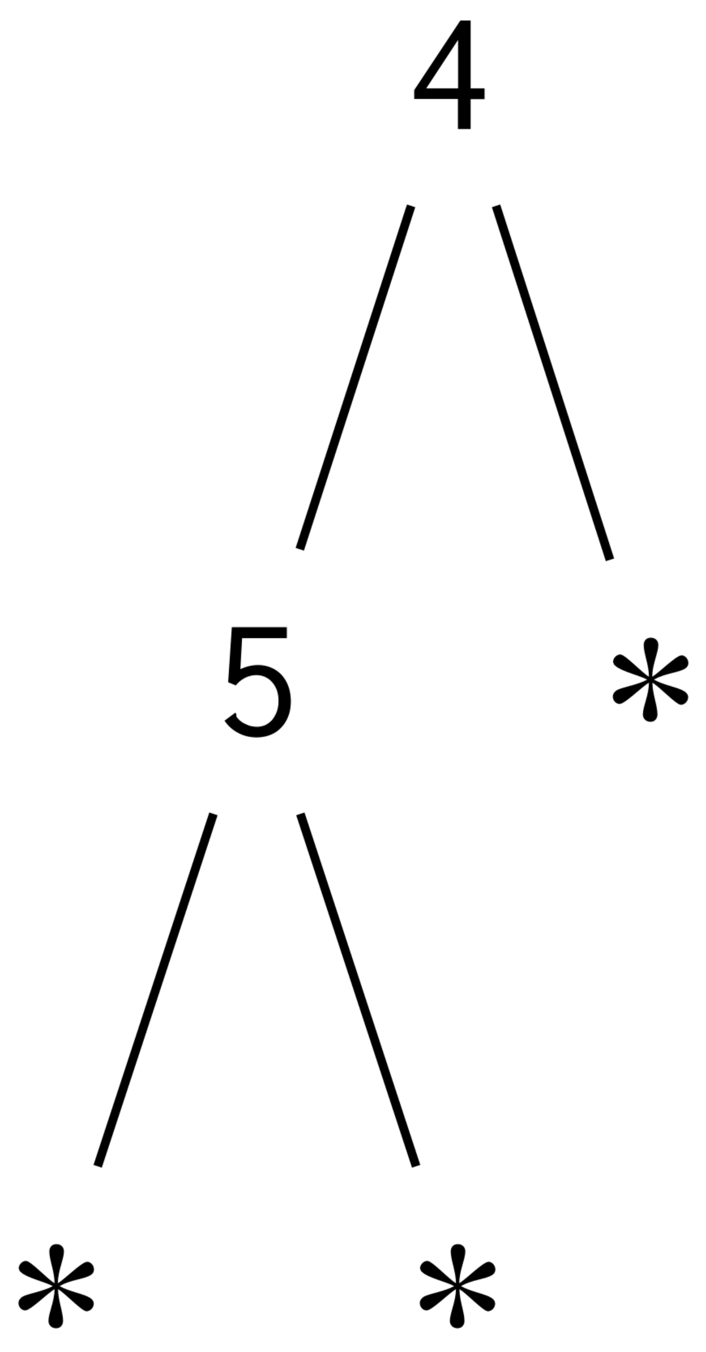
Policies for Recursive Data

`Node 1 (Node 2 Leaf Leaf) (Node 3 Leaf Leaf)`



Policies for Recursive Data

```
Node 4 (Node 5 Leaf Leaf) Leaf
```



Policies for Recursive Data

- Impossible to hide everything! Need to disclose some information for bounded representation and bounded computation.

Policies for Recursive Data

- Impossible to hide everything! Need to disclose some information for bounded representation and bounded computation.
- Many possible policies (for trees):

Policies for Recursive Data

- Impossible to hide everything! Need to disclose some information for bounded representation and bounded computation.
- Many possible policies (for trees):
    - Disclose maximum depth (hiding structural information, payload)

Policies for Recursive Data

- Impossible to hide everything! Need to disclose some information for bounded representation and bounded computation.
- Many possible policies (for trees):
    - Disclose maximum depth (hiding structural information, payload)
    - Disclose spine upper bound (hiding partial structural information, payload)

Policies for Recursive Data

- Impossible to hide everything! Need to disclose some information for bounded representation and bounded computation.
- Many possible policies (for trees):
  - Disclose maximum depth (hiding structural information, payload)
  - Disclose spine upper bound (hiding partial structural information, payload)
  - Disclose spine (hiding payload)

Policies for Recursive Data

- Impossible to hide everything! Need to disclose some information for bounded representation and bounded computation.
- Many possible policies (for trees):
  - Disclose maximum depth (hiding structural information, payload)
  - Disclose spine upper bound (hiding partial structural information, payload)
  - Disclose spine (hiding payload)
  - Disclose spine and some payload (hiding part of payload)

Policies for Recursive Data

- Impossible to hide everything! Need to disclose some information for bounded representation and bounded computation.
- Many possible policies (for trees):
    - Disclose maximum depth (hiding structural information, payload)
    - Disclose spine upper bound (hiding partial structural information, payload)
    - Disclose spine (hiding payload)
    - Disclose spine and some payload (hiding part of payload)
    - Disclose everything! (hiding nothing)

Policies for Recursive Data

- Impossible to hide everything! Need to disclose some information for bounded representation and bounded computation.
- Many possible policies (for trees): SLOW

MORE PRIVACY

  - Disclose maximum depth (hiding structural information, payload)
  - Disclose spine upper bound (hiding partial structural information, payload)
  - Disclose spine (hiding payload)
  - Disclose spine and some payload (hiding part of payload)
  - Disclose everything! (hiding nothing) FAST

LESS PRIVACY

A policy for a data specifies what information of this data can be publicly shared, which can be an arbitrary projection of the data, called

public view

Encoding private data and policies

Challenges

An attacker is one of the participants running the program, so they can:

Observe the data structures themselves

Challenges

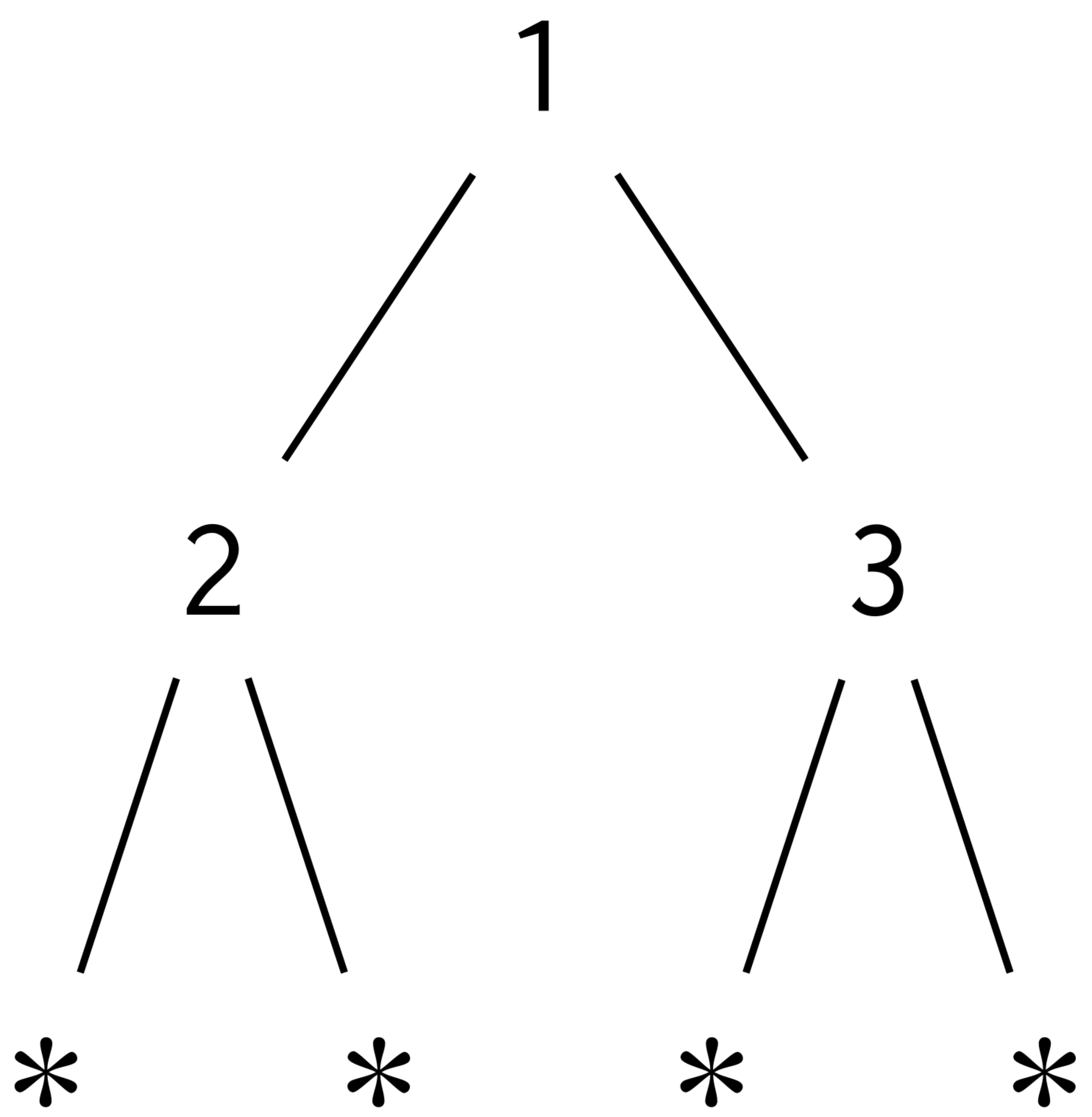
An attacker is one of the participants running the program, so they can:

Observe the data structures themselves: need to obscure the shape of the data

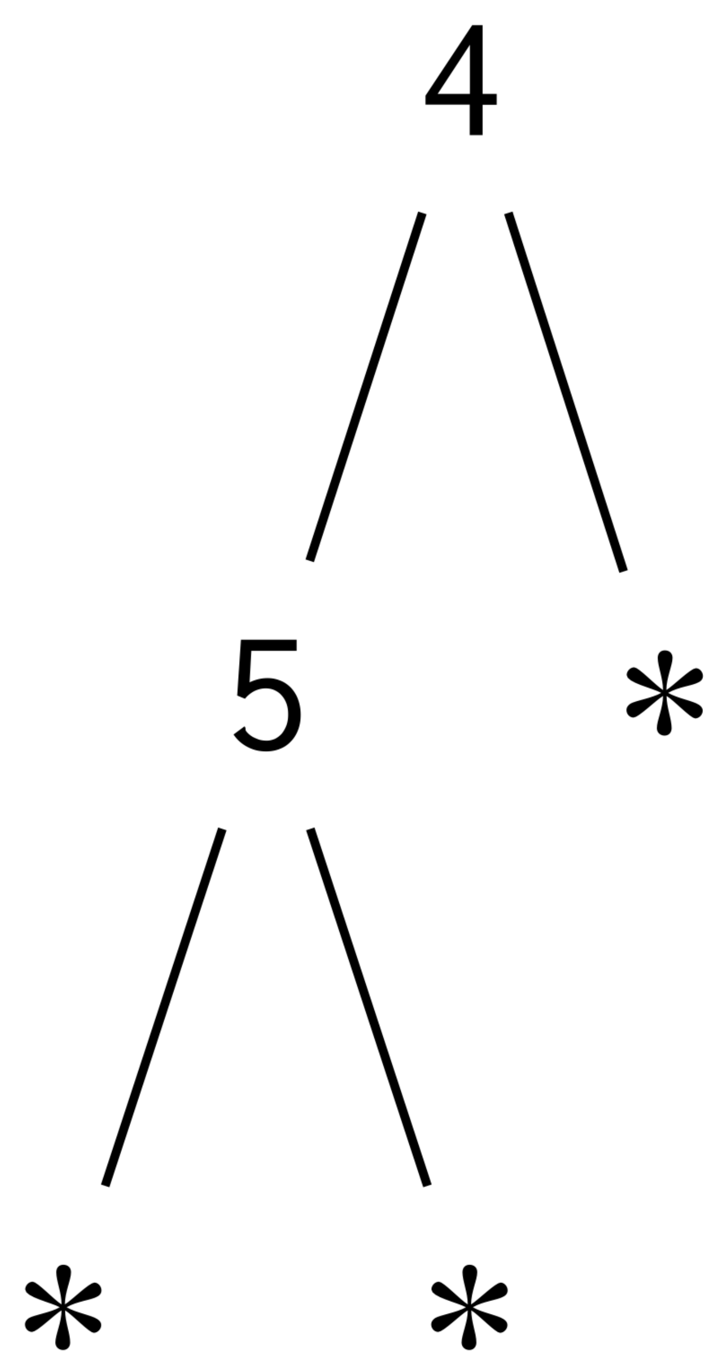
Obscure Data Representation

```
data tree = Leaf | Node int tree tree
```

`Node 1 (Node 2 Leaf Leaf) (Node 3 Leaf Leaf)`



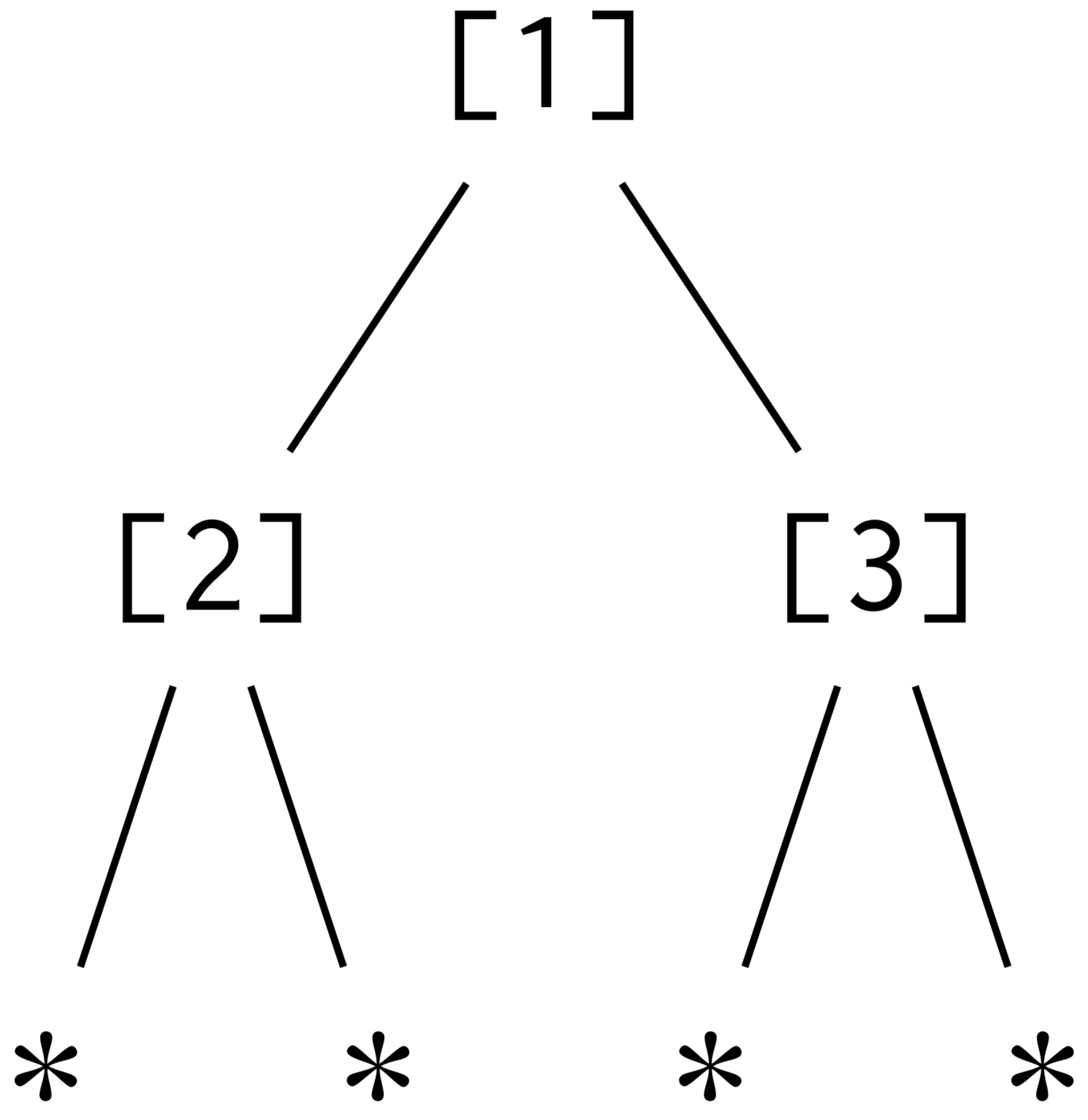
`Node 4 (Node 5 Leaf Leaf) Leaf`



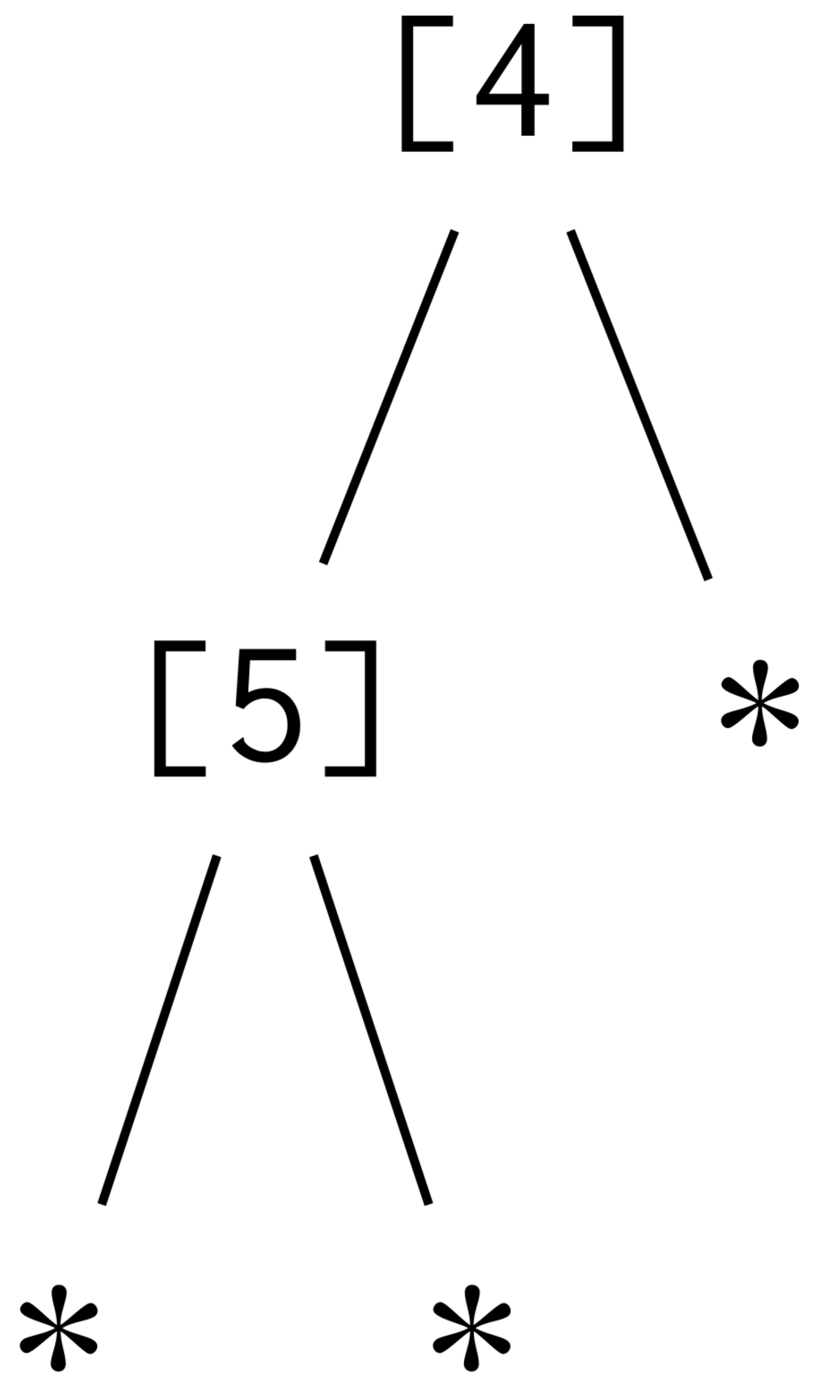
Public View: Maximum Depth = 2

```
data tree = Leaf | Node int tree tree
```

```
Node 1 (Node 2 Leaf Leaf) (Node 3 Leaf Leaf)
```




```
Node 4 (Node 5 Leaf Leaf) Leaf
```

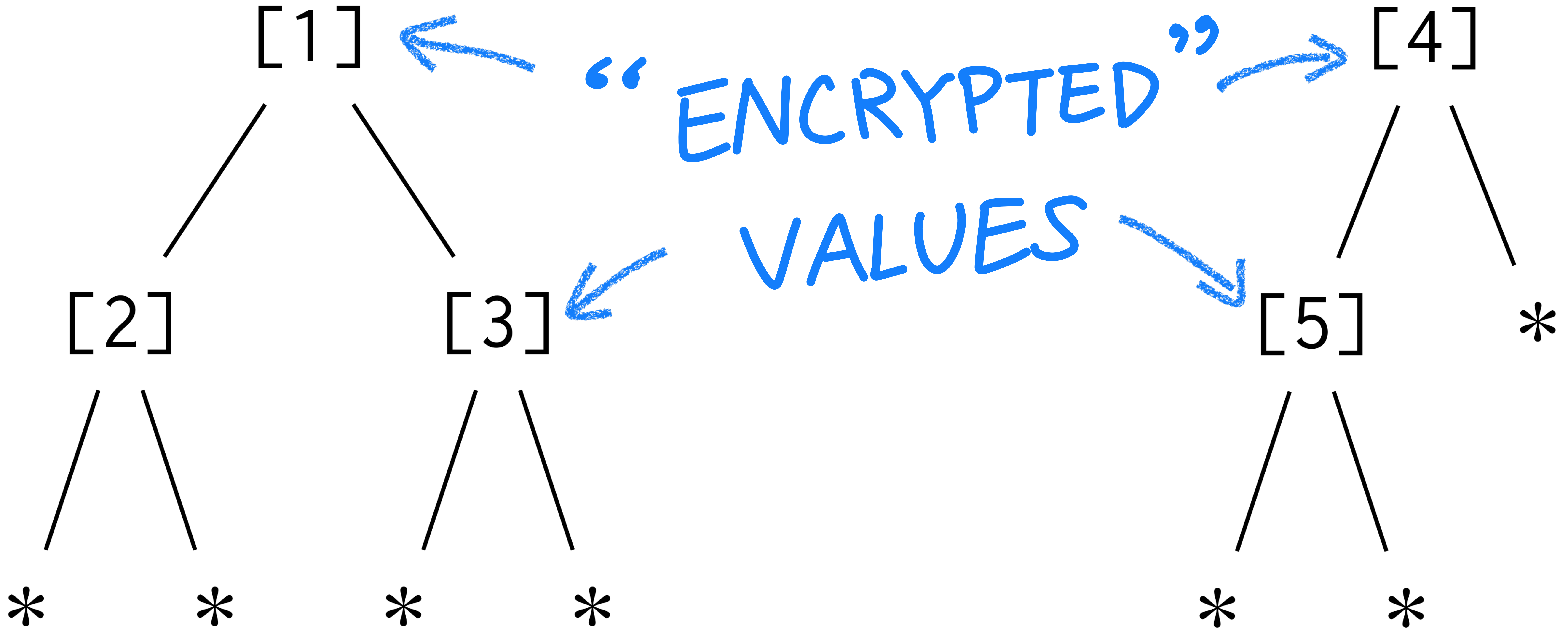


Public View: Maximum Depth = 2

```
data tree = Leaf | Node int tree tree
```

`Node 1 (Node 2 Leaf Leaf) (Node 3 Leaf Leaf)`

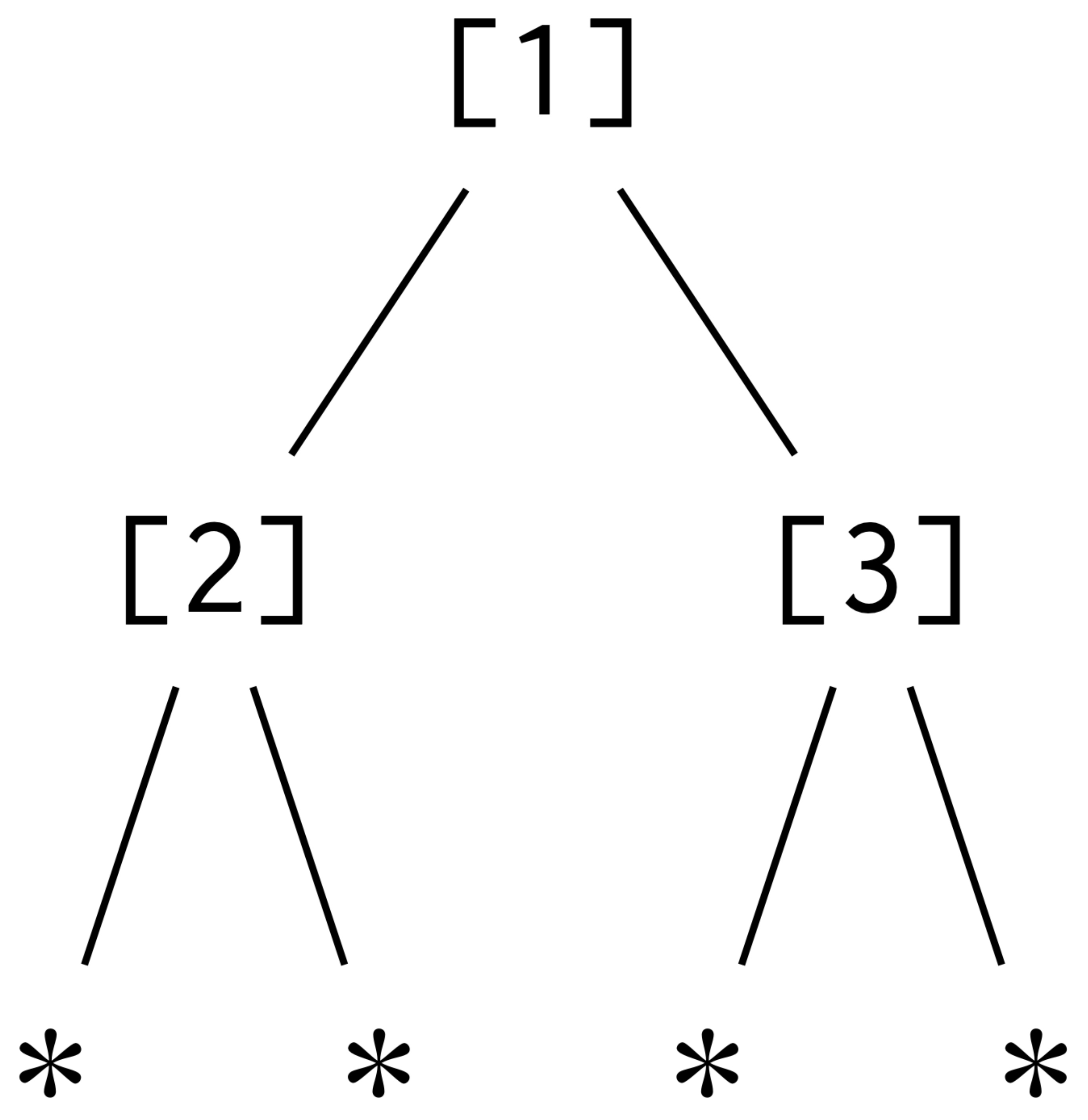
`Node 4 (Node 5 Leaf Leaf) Leaf`



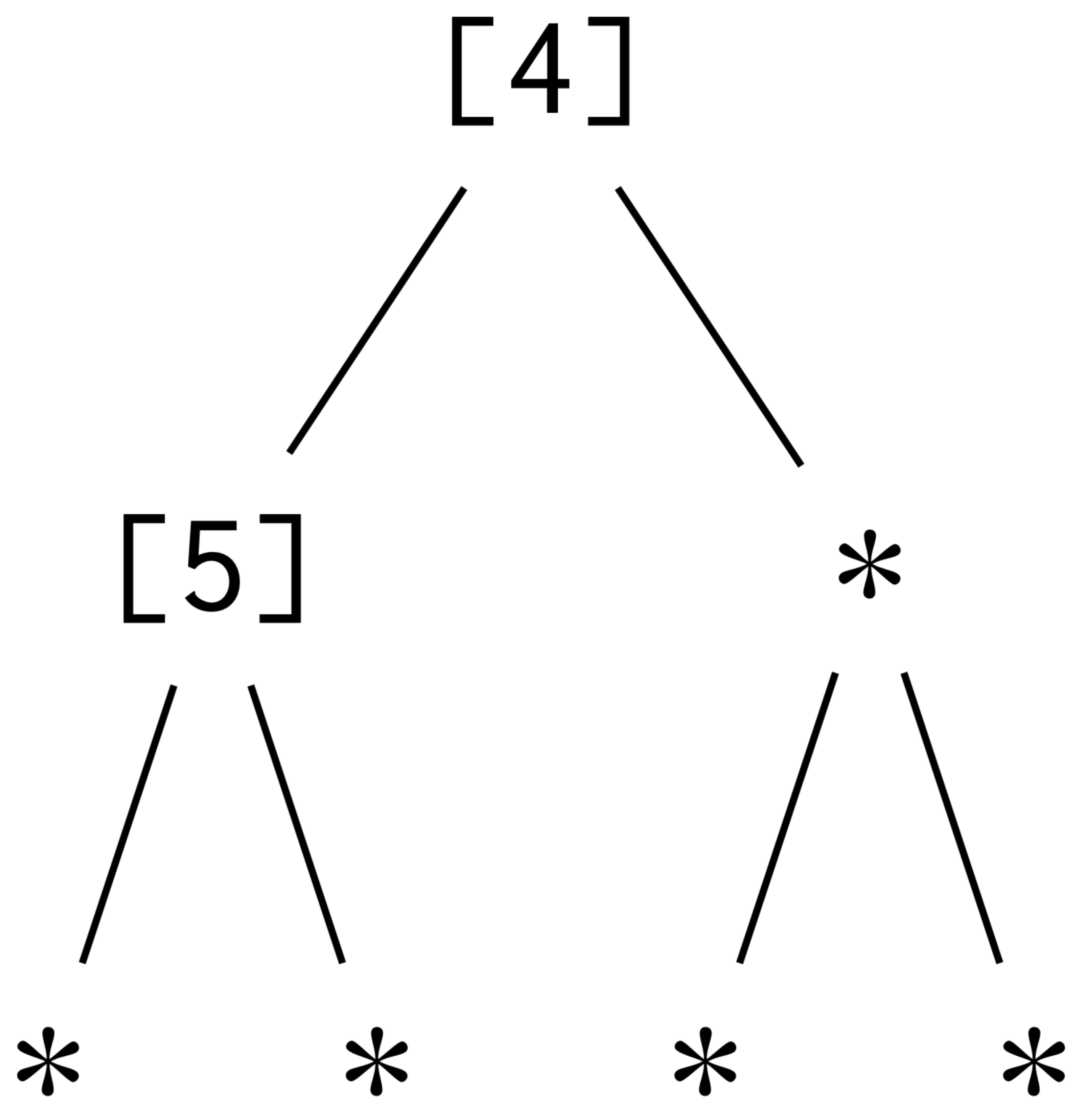
Public View: Maximum Depth = 2

```
data tree = Leaf | Node int tree tree
```

`Node 1 (Node 2 Leaf Leaf) (Node 3 Leaf Leaf)`



`Node 4 (Node 5 Leaf Leaf) Leaf`



Oblivious Algebraic Data Types (OADT)

$$
\begin{array}{l}
\texttt{obliv}\ \widehat{\texttt{tree}}\ (\texttt{k} : \texttt{nat}) = \\
\quad \texttt{if}\ \texttt{k} = \texttt{0} \\
\quad \texttt{then}\ \mathbb{1} \\
\quad \texttt{else}\ \mathbb{1}\ \widehat{+}\ \widehat{\texttt{int}} \times \widehat{\texttt{tree}}\ (\texttt{k}-1) \times \widehat{\texttt{tree}}\ (\texttt{k}-1)
\end{array}
$$

Oblivious Algebraic Data Types (OADT)

DEPENDENT TYPE

```
obliv tree (k : nat) =
  if k = 0
  then 𝟙
  else 𝟙 +̂ int × tree (k−1) × tree (k−1)
```

$$\texttt{obliv}\ \widehat{\texttt{tree}}\ (k : \texttt{nat}) = \texttt{if}\ k = 0\ \texttt{then}\ \mathbb{1}\ \texttt{else}\ \mathbb{1}\ \widehat{+}\ \widehat{\texttt{int}} \times \widehat{\texttt{tree}}\ (k-1) \times \widehat{\texttt{tree}}\ (k-1)$$

Oblivious Algebraic Data Types (OADT)

DEPENDENT TYPE

PUBLIC VIEW
(MAX DEPTH)

$$
\begin{aligned}
&\texttt{obliv}\ \widehat{\texttt{tree}}\ (\texttt{k} : \texttt{nat}) = \\
&\quad \texttt{if k = 0} \\
&\quad \texttt{then}\ \mathbb{1} \\
&\quad \texttt{else}\ \mathbb{1}\ \hat{+}\ \widehat{\texttt{int}} \times \widehat{\texttt{tree}}\ (\texttt{k}-1) \times \widehat{\texttt{tree}}\ (\texttt{k}-1)
\end{aligned}
$$

Oblivious Algebraic Data Types (OADT)

DEPENDENT TYPE

PUBLIC VIEW
(MAX DEPTH)

PRIVATE
REPRESENTATION

$$\texttt{obliv}\ \widehat{\texttt{tree}}\ (k : \texttt{nat}) =$$
$$\quad \texttt{if}\ k = 0$$
$$\quad \texttt{then}\ \mathbb{1}$$
$$\quad \texttt{else}\ \mathbb{1}\ \hat{+}\ \widehat{\texttt{int}} \times \widehat{\texttt{tree}}\ (k-1) \times \widehat{\texttt{tree}}\ (k-1)$$

Oblivious Algebraic Data Types (OADT)

$$
\begin{aligned}
&\textsf{obliv}\ \widehat{\textsf{tree}}\ (k : \textsf{nat}) = \\
&\quad \textsf{if}\ k = 0 \\
&\quad \textsf{then}\ \mathbb{1} \\
&\quad \textsf{else}\ \mathbb{1}\ \widehat{+}\ \widehat{\textsf{int}} \times \widehat{\textsf{tree}}\ (k-1) \times \widehat{\textsf{tree}}\ (k-1)
\end{aligned}
$$

UNIT (ISOMORPHIC TO LEAF)

Oblivious Algebraic Data Types (OADT)

$$\mathtt{obliv}\ \widehat{\mathtt{tree}}\ (\mathtt{k} : \mathtt{nat}) =$$
$$\quad \mathtt{if}\ \mathtt{k} = 0$$
$$\quad \mathtt{then}\ \mathbb{1}$$
$$\quad \mathtt{else}\ \mathbb{1}\ \widehat{+}\ \widehat{\mathtt{int}} \times \widehat{\mathtt{tree}}\ (\mathtt{k}-1) \times \widehat{\mathtt{tree}}\ (\mathtt{k}-1)$$

UNIT (ISOMORPHIC TO LEAF)

OBLIVIOUS SUM

Oblivious Algebraic Data Types (OADT)

UNIT (ISOMORPHIC TO LEAF)

$$\begin{array}{l} \mathsf{obliv}\ \widehat{\mathsf{tree}}\ (k : \mathsf{nat}) = \\ \quad \mathsf{if}\ k = 0 \\ \quad \mathsf{then}\ \mathbb{1} \\ \quad \mathsf{else}\ \mathbb{1}\ \widehat{+}\ \underbrace{\widehat{\mathsf{int}} \times \widehat{\mathsf{tree}}\ (k-1) \times \widehat{\mathsf{tree}}\ (k-1)}_{\text{ISOMORPHIC TO NODE}} \end{array}$$

OBLIVIOUS SUM

Oblivious Algebraic Data Types (OADT)

$$\begin{array}{l} \texttt{obliv}\ \widehat{\texttt{tree}}\ (k : \texttt{nat}) = \\ \quad \texttt{if}\ k = 0 \\ \quad \texttt{then}\ \mathbb{1} \\ \quad \texttt{else}\ \mathbb{1}\ \widehat{+}\ \underbrace{\widehat{\texttt{int}} \times \widehat{\texttt{tree}}\ (k-1) \times \widehat{\texttt{tree}}\ (k-1)}_{\text{ISOMORPHIC TO NODE}} \end{array}$$

OBLIVIOUS INTEGER

OBLIVIOUS SUM

Oblivious Algebraic Data Types (OADT)

$$
\begin{aligned}
&\texttt{obliv}\ \widehat{\texttt{tree}}\ (\texttt{k} : \texttt{nat}) = \\
&\quad \texttt{if}\ \texttt{k} = \texttt{0} \\
&\quad \texttt{then}\ \mathbb{1} \\
&\quad \texttt{else}\ \mathbb{1}\ \widehat{+}\ \widehat{\texttt{int}} \times \widehat{\texttt{tree}}\ (\texttt{k}-1) \times \widehat{\texttt{tree}}\ (\texttt{k}-1)
\end{aligned}
$$

OBLIVIOUS INTEGER

OBLIVIOUS SUM

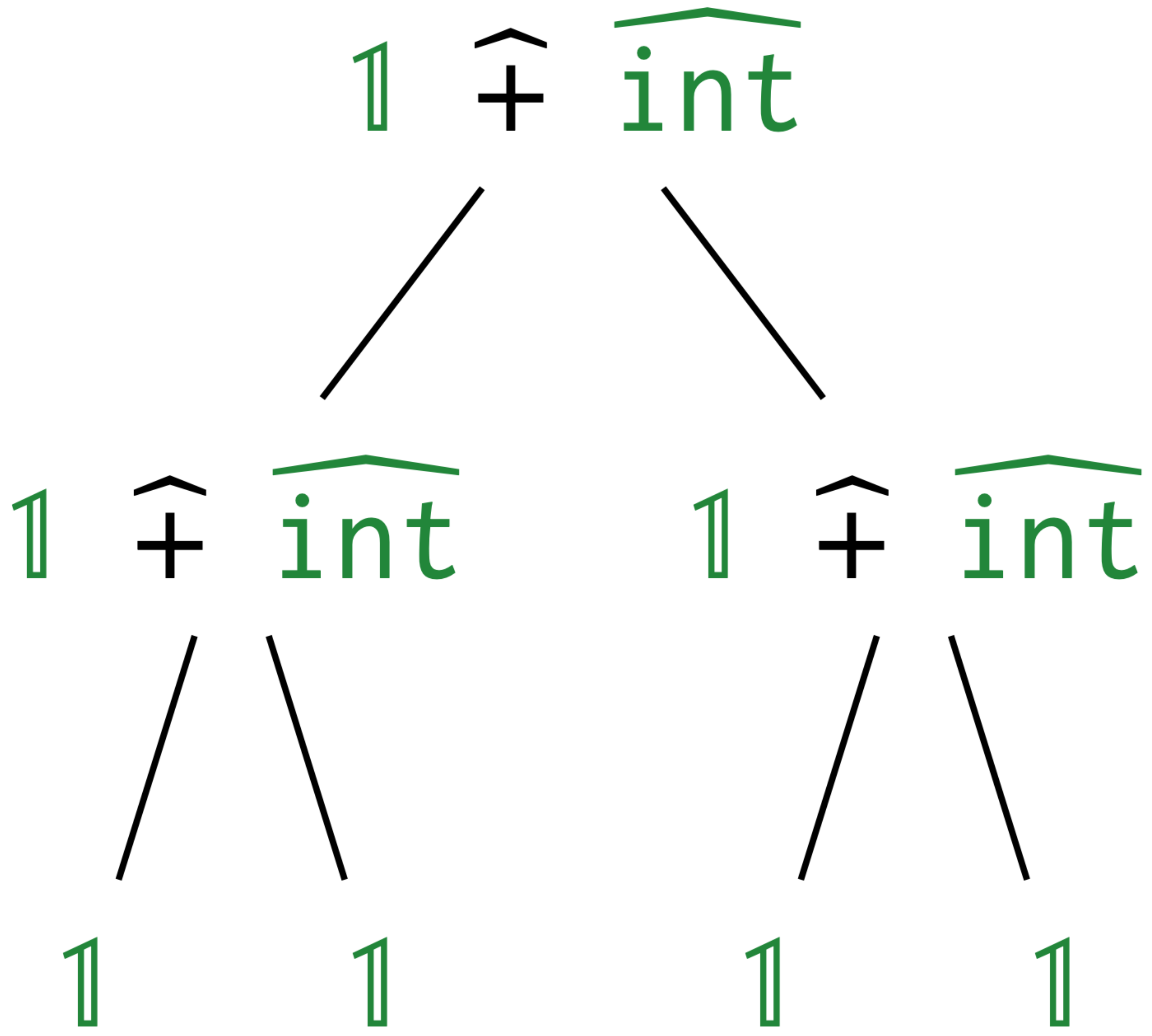
PRODUCT

An Example

$$\widehat{\texttt{tree}}\ 2 \equiv \mathbb{1}\ \hat{+}\ \widehat{\texttt{int}} \times (\mathbb{1}\ \hat{+}\ \widehat{\texttt{int}} \times \mathbb{1} \times \mathbb{1}) \times (\mathbb{1}\ \hat{+}\ \widehat{\texttt{int}} \times \mathbb{1} \times \mathbb{1})$$

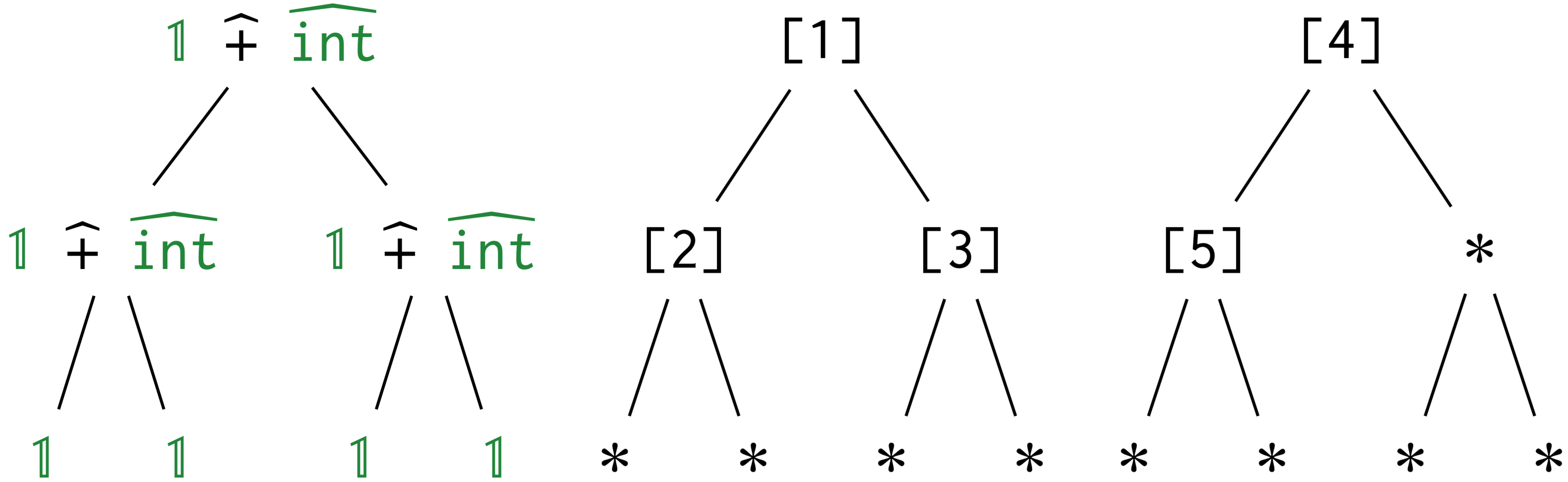
An Example

$$\widehat{\mathtt{tree}}\ 2 \equiv \mathbb{1} \mathbin{\widehat{+}} \widehat{\mathtt{int}} \times (\mathbb{1} \mathbin{\widehat{+}} \widehat{\mathtt{int}} \times \mathbb{1} \times \mathbb{1}) \times (\mathbb{1} \mathbin{\widehat{+}} \widehat{\mathtt{int}} \times \mathbb{1} \times \mathbb{1})$$



An Example

$$\widehat{\mathtt{tree}}\ 2 \equiv \mathbb{1} \mathbin{\hat{+}} \widehat{\mathtt{int}} \times (\mathbb{1} \mathbin{\hat{+}} \widehat{\mathtt{int}} \times \mathbb{1} \times \mathbb{1}) \times (\mathbb{1} \mathbin{\hat{+}} \widehat{\mathtt{int}} \times \mathbb{1} \times \mathbb{1})$$



OADTs Generalize Secure Integer

`int`

OADTs Generalize Secure Integer

`int`

$\widehat{\texttt{int}}$

OADTs Generalize Secure Integer

$$\begin{tikzcd} \texttt{int} \arrow[d, bend left, "\widehat{\texttt{int}}\#s"] \\ \widehat{\texttt{int}} \arrow[u, bend left, "\widehat{\texttt{int}}\#r"] \end{tikzcd}$$

OADTs Generalize Secure Integer

$$\begin{tikzcd} \mathtt{int} \arrow[d, bend left, "\widehat{\mathtt{int}}\#s"] \\ \widehat{\mathtt{int}} \arrow[u, bend left, "\widehat{\mathtt{int}}\#r"] \end{tikzcd}$$

SECTION ("ENCRYPTION")

OADTs Generalize Secure Integer

RETRACTION ("DECRYPTION")

SECTION ("ENCRYPTION")

$$
\begin{tikzcd}
\mathtt{int} \arrow[d, bend left, "\widehat{\mathtt{int}}\#\mathtt{s}"] \\
\widehat{\mathtt{int}} \arrow[u, bend left, "\widehat{\mathtt{int}}\#\mathtt{r}"]
\end{tikzcd}
$$

Secure Integer Has Public View!

$$
\begin{tikzcd}
\{\, \mathtt{n} : \mathbb{Z} \mid \mathtt{width\ n} = 32 \,\} \arrow[d, bend left, "\widehat{\mathtt{int}}\mathtt{\#s}"] \\
\widehat{\mathtt{int}}\ \mathtt{32} \arrow[u, bend left, "\widehat{\mathtt{int}}\mathtt{\#r}"]
\end{tikzcd}
$$

OADTs Are “Encryption Spaces” Indexed By Public Views

$$\begin{tikzcd} \{\, t : \mathtt{tree} \mid \mathtt{depth}\ t \le k \,\} \arrow[d, bend left, "\widehat{\mathtt{tree}}\#s"] \\ \widehat{\mathtt{tree}}\ k \arrow[u, bend left, "\widehat{\mathtt{tree}}\#r"] \end{tikzcd}$$

Enforcing Privacy Policies

Challenges

An attacker is one of the participants running the program, so they can:

Observe how the data structures are used

Challenges

An attacker is one of the participants running the program, so they can:

Observe how the data structures are used: need to prevent leakage through timing channel and control flow channel

A Simple Example

```
fn sum (t : tree) : int =
  match t with
  | Leaf ⇒ 0
  | Node x l r ⇒ x + sum l + sum r
```

Control Flow Channel

```
match t with
| Leaf ⇒ 0
| Node x l r ⇒ x + sum l + sum r
```

Control Flow Channel

```
match t with
| Leaf ⇒ 0
| Node x l r ⇒ x + sum l + sum r
```

↓

```
match t with
| Leaf ⇒ 0
| Node x l r ⇒ x + sum l + sum r
```

Oblivious Operations

$$\texttt{mux}\ ([3]\ \widehat{\le}\ [4])\ ([5]\ \widehat{+}\ [1])\ ([6]\ \widehat{+}\ [1])$$

Oblivious Operations

$$\texttt{mux}\ \underbrace{([3]\ \widehat{\leq}\ [4])}_{\text{PRIVATE CONDITION}}\ ([5]\ \widehat{+}\ [1])\ ([6]\ \widehat{+}\ [1])$$

Oblivious Operations

$$\texttt{mux}\ \underbrace{([3]\ \widehat{\le}\ [4])}_{\text{PRIVATE CONDITION}}\ \underbrace{([5]\ \widehat{+}\ [1])\ ([6]\ \widehat{+}\ [1])}_{\text{PRIVATE BRANCHES}}$$

Oblivious Operations

$$\texttt{mux}\ ([3]\ \widehat{\leq}\ [4])\ ([5]\ \widehat{+}\ [1])\ ([6]\ \widehat{+}\ [1])$$

$$\downarrow$$

$$\texttt{mux}\ [\texttt{true}]\ ([5]\ \widehat{+}\ [1])\ ([6]\ \widehat{+}\ [1])$$

Oblivious Operations

$$\texttt{mux}\ ([3]\ \widehat{\leq}\ [4])\ ([5]\ \widehat{+}\ [1])\ ([6]\ \widehat{+}\ [1])$$

$$\downarrow$$

$$\texttt{mux}\ [\texttt{true}]\ ([5]\ \widehat{+}\ [1])\ ([6]\ \widehat{+}\ [1])$$

$$\downarrow$$

$$\texttt{mux}\ [\texttt{true}]\ [6]\ ([6]\ \widehat{+}\ [1])$$

Oblivious Operations

$$\texttt{mux}\ ([3]\ \widehat{\leq}\ [4])\ ([5]\ \widehat{+}\ [1])\ ([6]\ \widehat{+}\ [1])$$

$$\downarrow$$

$$\texttt{mux}\ [\texttt{true}]\ ([5]\ \widehat{+}\ [1])\ ([6]\ \widehat{+}\ [1])$$

$$\downarrow$$

$$\texttt{mux}\ [\texttt{true}]\ [6]\ ([6]\ \widehat{+}\ [1])$$

$$\downarrow$$

$$\texttt{mux}\ [\texttt{true}]\ [6]\ [7]$$

Oblivious Operations

```
mux ([3] ≤̂ [4]) ([5] +̂ [1]) ([6] +̂ [1])
↓
mux [true] ([5] +̂ [1]) ([6] +̂ [1])
↓
mux [true] [6] ([6] +̂ [1])
↓
mux [true] [6] [7]
↓
[6]
```

Oblivious Operations

$$\texttt{mux}\ ([5]\ \widehat{\leq}\ [4])\ ([5]\ \widehat{+}\ [1])\ ([6]\ \widehat{+}\ [1])$$

$$\downarrow$$

$$\texttt{mux}\ [\texttt{false}]\ ([5]\ \widehat{+}\ [1])\ ([6]\ \widehat{+}\ [1])$$

$$\downarrow$$

$$\texttt{mux}\ [\texttt{false}]\ [6]\ ([6]\ \widehat{+}\ [1])$$

$$\downarrow$$

$$\texttt{mux}\ [\texttt{false}]\ [6]\ [7]$$

$$\downarrow$$

$$[7]$$

The same idea is generalized to other oblivious operations for manipulating OADTs, and the security-type system ensures these operations are used securely

Type System and Formal Guarantees

Type System and Formal Guarantees

- Incorporates ideas from dependently typed languages and security-typed languages

Type System and Formal Guarantees

- Incorporates ideas from dependently typed languages and security-typed languages
- Formalized the core calculus (i.e. formal model of the language)

Type System and Formal Guarantees

- Incorporates ideas from dependently typed languages and security-typed languages

- Formalized the core calculus (i.e. formal model of the language)

- Proved type system is **sound** and ensures an **obliviousness** property: no private information can be inferred by observing the execution traces

Type System and Formal Guarantees

- Incorporates ideas from dependently typed languages and security typed languages

Theorem 4.4 (Obliviousness). *If* $e_1 \approx e_2$ *and* $\cdot \vdash e_1 :_{l_1} \tau_1$ *and* $\cdot \vdash e_2 :_{l_2} \tau_2$, *then*
(1) $e_1 \longrightarrow^n e_1'$ *if and only if* $e_2 \longrightarrow^n e_2'$ *for some* e_2'.
(2) *if* $e_1 \longrightarrow^n e_1'$ *and* $e_2 \longrightarrow^n e_2'$, *then* $e_1' \approx e_2'$.

- Proved type system is **sound** and ensures an **obliviousness** property: no private information can be inferred by observing the execution traces

Type System and Formal Guarantees

- Incorporates ideas from dependently typed languages and security-typed languages

Theorem 4.4 (Obliviousness). *If* $\mathsf{e}_1 \approx \mathsf{e}_2$ *and* $\cdot \vdash \mathsf{e}_1 :_{l_1} \tau_1$ *and* $\cdot \vdash \mathsf{e}_2 :_{l_2} \tau_2$, *then*
(1) $\mathsf{e}_1 \longrightarrow^n \mathsf{e}_1'$ *if and only if* $\mathsf{e}_2 \longrightarrow^n \mathsf{e}_2'$ *for some* e_2'.
(2) *if* e_1

Certified By The Coq Proof Assistant

- Pr **obliviousness** property: no private information can be inferred by observing the execution traces

So far, we are able to encode complex policies for structured data and implement private computation painstakingly

Recall

```
fn sum (t : tree) : int =
  match t with
  | Leaf ⇒ 0
  | Node x l r ⇒ x + sum l + sum r
```

You Don't Want To Write This

$$
\begin{array}{l}
\texttt{fn}\ \widehat{\texttt{sum}}\ (\texttt{k} : \texttt{nat}) : \widehat{\texttt{tree}}\ \texttt{k} \longrightarrow \widehat{\texttt{int}} = \\
\quad \texttt{if}\ \texttt{k} = \texttt{0} \\
\quad \texttt{then}\ \lambda\texttt{\_} \Rightarrow \widehat{\texttt{int}}\texttt{\#s}\ \texttt{0} \\
\quad \texttt{else}\ \lambda\texttt{t} \Rightarrow \widehat{\texttt{match}}\ \texttt{t}\ \texttt{with} \\
\qquad\qquad\qquad\qquad \mid \widehat{\texttt{inl}}\ \texttt{\_} \Rightarrow \widehat{\texttt{int}}\texttt{\#s}\ \texttt{0} \\
\qquad\qquad\qquad\qquad \mid \widehat{\texttt{inr}}\ (\texttt{x},\ \texttt{l},\ \texttt{r}) \Rightarrow \texttt{x}\ \widehat{+}\ \widehat{\texttt{sum}}\ (\texttt{k}-\texttt{1})\ \texttt{l}\ \widehat{+}\ \widehat{\texttt{sum}}\ (\texttt{k}-\texttt{1})\ \texttt{r}
\end{array}
$$

You Don’t Want To Write This

```
fn sum (k : nat) : tree k ⟶ int =
  if k = 0
  then λ_ ⟹ int#s 0
  else λt ⟹ match t with
             | inl _ ⟹ int#s 0
             | inr (x, l, r) ⟹ x + sum (k−1) l + sum (k−1) r
```

Entanglement of Program Logic And Policies

Entanglement of Program Logic And Policies

- Need to manually restructure the programs to capture the policies and make sure the control flow only depends on the public information

Entanglement of Program Logic And Policies

- Need to manually restructure the programs to capture the policies and make sure the control flow only depends on the public information
- Programs are harder to read, write and reason about

Entanglement of Program Logic And Policies

- Need to manually restructure the programs to capture the policies and make sure the control flow only depends on the public information
- Programs are harder to read, write and reason about
- Policies are harder to update and audit

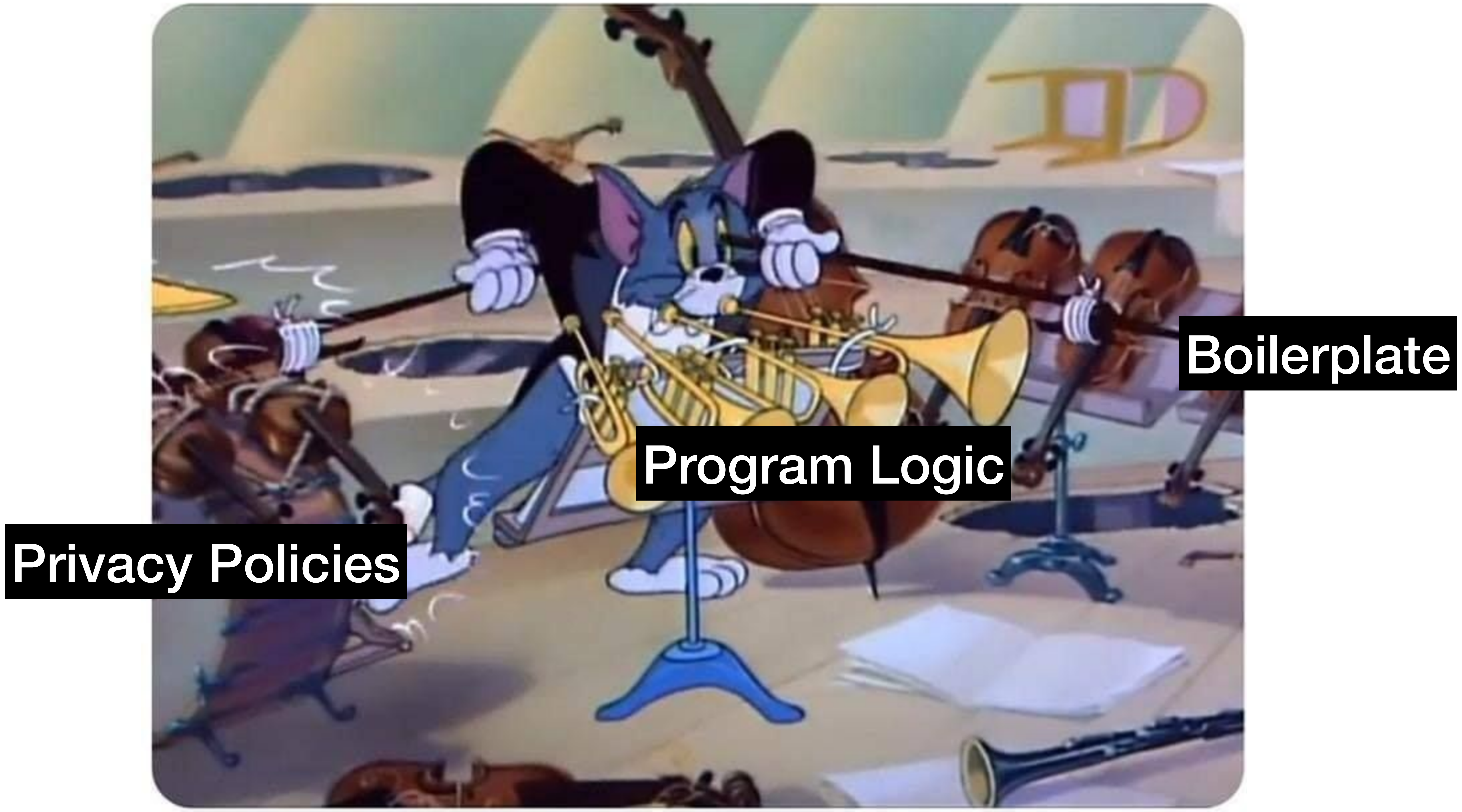
Entanglement of Program Logic And Policies

- Need to manually restructure the programs to capture the policies and make sure the control flow only depends on the public information
- Programs are harder to read, write and reason about
- Policies are harder to update and audit
- We may want to support multiple policies at the same time

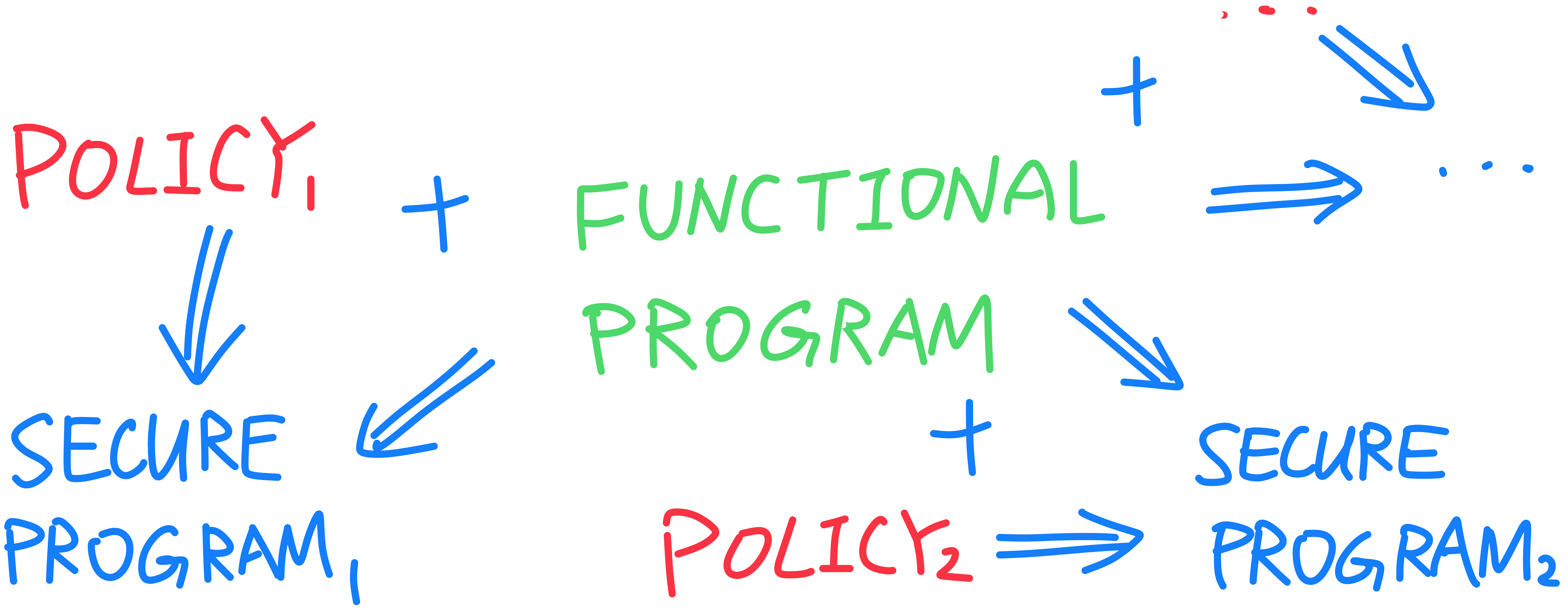
Entanglement of Program Logic And Policies

- Need to manually restructure the programs to capture the policies and make sure the control flow only depends on the public information
- Programs are harder to read, write and reason about
- Policies are harder to update and audit
- We may want to support multiple policies at the same time
- We may want to trade off between privacy and performance

Entanglement of Program Logic And Policies



Modularity / Policy-agnosticism



Automatically Enforcing Privacy Policies

You Want To Write This

$$\texttt{fn}\ \widehat{\texttt{sum}}\ (\texttt{k} : \texttt{nat})\ (\widehat{\texttt{t}} : \widehat{\texttt{tree}}\ \texttt{k}) : \widehat{\texttt{int}} = \ldots$$

You Want To Write This

$$\texttt{fn } \widehat{\texttt{sum}}\ \texttt{(k : nat) (}\hat{\texttt{t}}\texttt{ : }\widehat{\texttt{tree}}\texttt{ k) : }\widehat{\texttt{int}}\texttt{ = ...}$$

sum

You Want To Write This

$\texttt{fn}\ \widehat{\texttt{sum}}\ (\texttt{k} : \texttt{nat})\ (\hat{\texttt{t}} : \widehat{\texttt{tree}}\ \texttt{k}) : \widehat{\texttt{int}} = \ldots$

MAGIC(sum)

A Good First Step

$$\widehat{t}$$

A Good First Step

PRIVATE INPUT $\longrightarrow$ $\widehat{t}$

A Good First Step

$$
\begin{array}{c}
t \\
\Big\uparrow {\scriptstyle \widehat{\texttt{tree}}\#\texttt{r}} \\
\widehat{t}
\end{array}
$$

A Good First Step

RETRACTION ("DECRYPTION")

$$\begin{tikzcd} t \\ \hat{t} \arrow[u, "\widehat{\mathtt{tree}}\#r"] \end{tikzcd}$$

A Good First Step

RETRACTION
("DECRYPTION")

$$
\begin{tikzcd}
t \arrow[r, "\mathsf{sum}"] & o \\
\widehat{t} \arrow[u, "\overline{\mathsf{tree}}\#r"] &
\end{tikzcd}
$$

A Good First Step

RETRACTION
("DECRYPTION")

$$
\begin{tikzcd}
t \arrow[rr, "\mathtt{sum}"] & & o \arrow[d, "\widehat{\mathtt{int}}\#s"] \\
\hat{t} \arrow[u, "\widehat{\mathtt{tree}}\#r"] & & \hat{o}
\end{tikzcd}
$$

A Good First Step

RETRACTION
("DECRYPTION")

SECTION
("ENCRYPTION")

$$\begin{tikzcd} t \arrow[r, "\mathtt{sum}"] & o \arrow[d, "\widehat{\mathtt{int}}\#\mathtt{s}"] \\ \hat{t} \arrow[u, "\widehat{\mathtt{tree}}\#\mathtt{r}"] & \hat{o} \end{tikzcd}$$

A Good First Step

RETRACTION
("DECRYPTION")

SECTION
("ENCRYPTION")

$$\begin{tikzcd} t \arrow[r, "\mathtt{sum}"] & o \arrow[d, "\widehat{\mathtt{int}}\#s"] \\ \hat{t} \arrow[u, "\widehat{\mathtt{tree}}\#r"] \arrow[r, "\widehat{\mathtt{sum}}"] & \hat{o} \end{tikzcd}$$

A Good First Step

$$
\begin{tikzcd}
t \arrow[r, "\texttt{sum}"] & o \arrow[d] \\
\hat{t} \arrow[u] \arrow[r, "\widehat{\texttt{sum}}"] & \hat{o}
\end{tikzcd}
$$

$$\widehat{\texttt{int}}\#\texttt{s}\ (\texttt{sum}\ (\widehat{\texttt{tree}}\#\texttt{r}\ \texttt{k}\ \hat{\texttt{t}}))$$

Tape Semantics: dynamically repairs unsafe computation

An “Unsafe” Operation

$\widehat{\texttt{if}}$ `[true] then true else false`

An “Unsafe” Operation

$$\widehat{\mathtt{if}}\ \mathtt{[true]\ then\ true\ else\ false}$$

PRIVATE

An “Unsafe” Operation

$\widehat{\texttt{if}}\ \texttt{[true] then true else false}$

PRIVATE

PUBLIC

An “Unsafe” Operation

$\widehat{\texttt{if}}$ `[true] then true else false`

PRIVATE

PUBLIC

$\downarrow$

`true`

An “Unsafe” Operation

$\widehat{\texttt{if}}\ \texttt{[true] then true else false}$

PRIVATE

PUBLIC

$\cancel{\downarrow}$

true

An Example

```
tape (int#s (if [true] then 3 else 4))
```

An Example

$$\texttt{tape}\ (\underbrace{\widehat{\texttt{int}}\texttt{\#s}\ (\widehat{\texttt{if}}\ [\texttt{true}]\ \texttt{then}\ 3\ \texttt{else}\ 4))}_{\text{EVENTUALLY OBLIVIOUS}}$$

Delay Unsafe Computation

```
tape (int#s (if [true] then 3 else 4))
```

Can’t Leak If You Don’t Run The Program

Propagate Surrounding Computation

```
tape (int#s (if [true] then 3 else 4))
↓
tape (if [true] then int#s 3 else int#s 4)
```

Propagate Surrounding Computation

```
tape (int#s (if [true] then 3 else 4))
```

↓

```
tape (if [true] then int#s 3 else int#s 4)
```

Propagate Surrounding Computation

```
tape (int#s (if [true] then 3 else 4))
```

$\downarrow$

```
tape (if [true] then int#s 3 else int#s 4)
```

Propagate Surrounding Computation

```
tape (int#s (if [true] then 3 else 4))
↓
tape (if [true] then int#s 3 else int#s 4)
↓
tape (if [true] then [3] else [4])
```

Propagate Surrounding Computation

```
tape (int#s (if [true] then 3 else 4))
↓
tape (if [true] then int#s 3 else int#s 4)
↓
tape (if [true] then [3] else [4])
```

PRIVATE NOW!

Cancel Unsafe Computation

```
tape (int#s (if [true] then 3 else 4))
                  ↓
tape (if [true] then int#s 3 else int#s 4)
                  ↓
tape (if [true] then [3] else [4])
                  ↓
mux [true] [3] [4]
```

Cancel Unsafe Computation

```
tape (int#s (if [true] then 3 else 4))
↓
tape (if [true] then int#s 3 else int#s 4)
↓
tape (if [true] then [3] else [4])
↓
mux [true] [3] [4]
↓
[3]
```

Similar ideas for other “unsafe” operators

The language and type system extended with this dynamic policy enforcement are still sound and secure!

Modular Implementation

$$
\begin{array}{l}
\texttt{obliv}\ \widehat{\texttt{tree}}\ \texttt{(k : nat) = ...} \\
\texttt{fn}\ \widehat{\texttt{tree}}\texttt{\#s (k : nat) (t : tree) : }\widehat{\texttt{tree}}\texttt{ k = ...} \\
\texttt{fn}\ \widehat{\texttt{tree}}\texttt{\#r (k : nat) (}\hat{\texttt{t}}\texttt{ : }\widehat{\texttt{tree}}\texttt{ k) : tree = ...} \\
\\
\texttt{fn}\ \widehat{\texttt{sum}}\ \texttt{(k : nat) (}\hat{\texttt{t}}\texttt{ : }\widehat{\texttt{tree}}\texttt{ k) : }\widehat{\texttt{int}}\texttt{ =} \\
\quad \texttt{tape (}\widehat{\texttt{int}}\texttt{\#s (sum (}\widehat{\texttt{tree}}\texttt{\#r k }\hat{\texttt{t}}\texttt{)))}
\end{array}
$$

Modular Implementation

ONCE & FOR ALL
& REUSABLE

```
obliv tree (k : nat) = ...
fn tree#s (k : nat) (t : tree) : tree k = ...
fn tree#r (k : nat) (t : tree k) : tree = ...

fn sum (k : nat) (t : tree k) : int =
   tape (int#s (sum (tree#r k t)))
```

Modular Implementation

```
obliv tree′ (s : spine) = ...
fn tree′#s (s : spine) (t : tree) : tree′ s = ...
fn tree′#r (s : spine) (t : tree′ s) : tree = ...

fn sum (s : spine) (t : tree′ s) : int =
  tape (int#s (sum (tree′#r s t)))
```

We Did It!

We Did It!

- **Rich**: the language (Taype) is a high-level functional language with supports for structured data and complex policies

We Did It!

- **Rich**: the language (Taype) is a high-level functional language with supports for structured data and complex policies
- **Safe**: secure by construction by obliviousness theorem

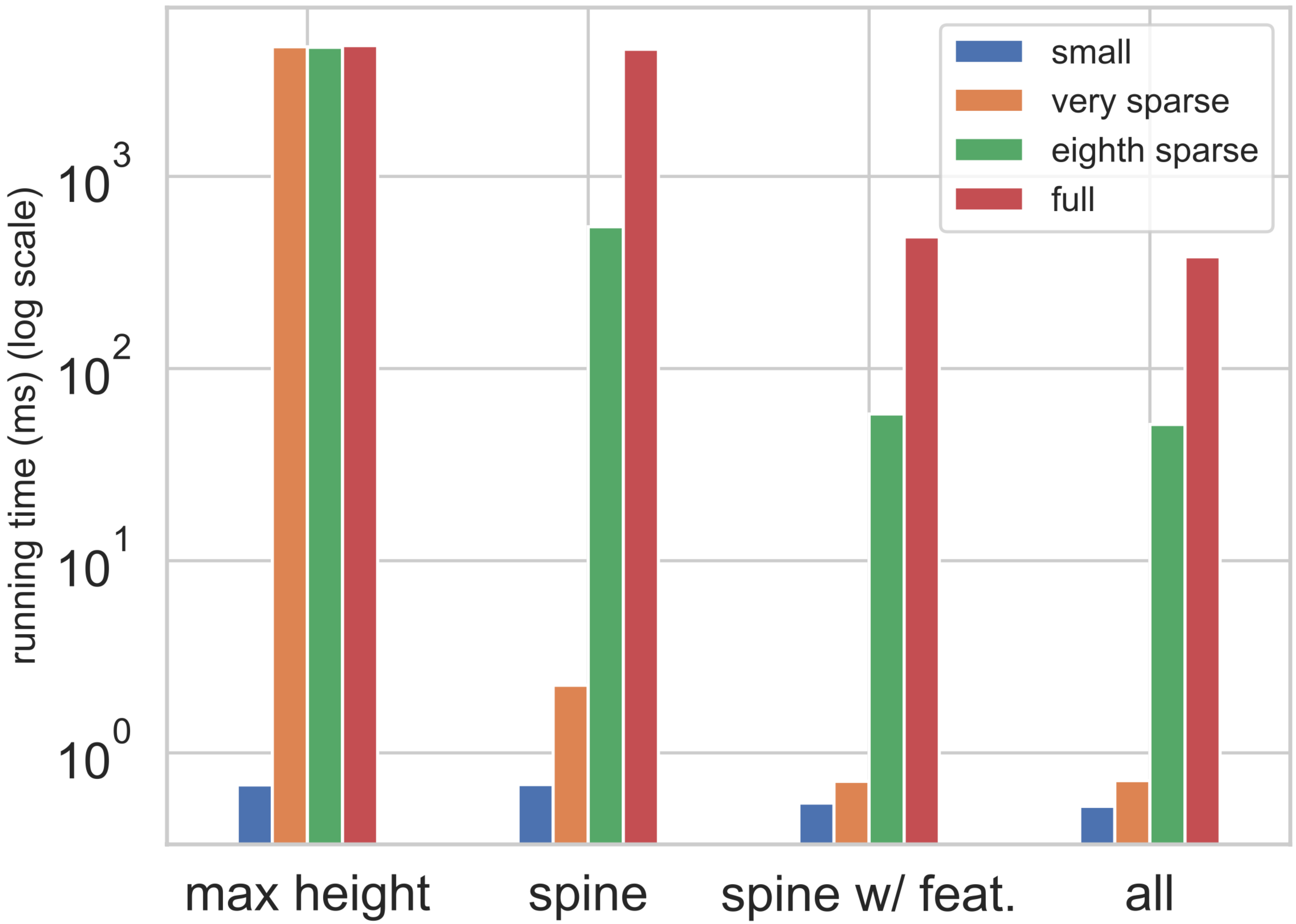
We Did It!

- **Rich**: the language (Taype) is a high-level functional language with supports for structured data and complex policies
- **Safe**: secure by construction by obliviousness theorem
- **Easy**: writing application logic for the secure computation is as easy as writing normal programs

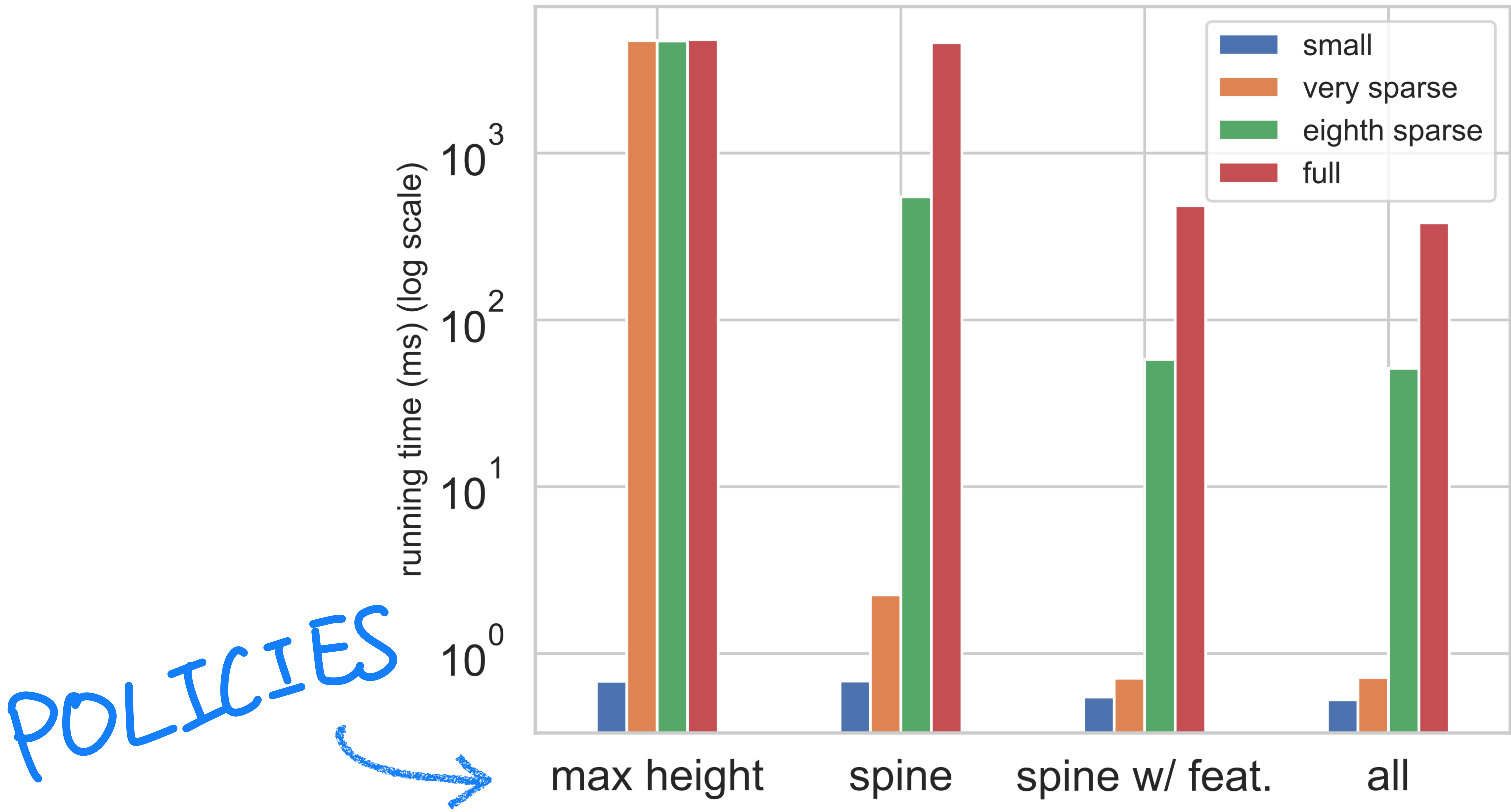
Case Studies

- Medical records
- Dating application
- Secure calculator
- K-means
- Private decision trees

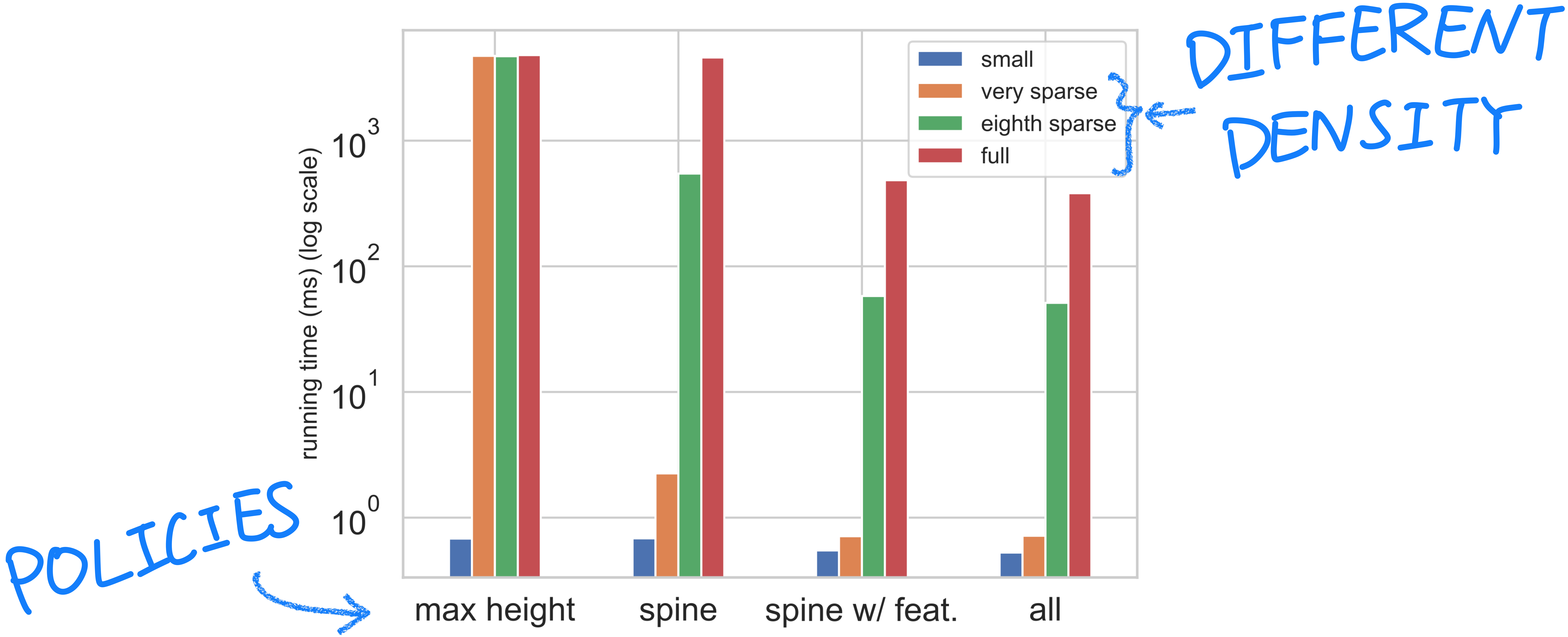
Private Decision Tree Classification



Private Decision Tree Classification



Private Decision Tree Classification



Takeaway

Takeaway

- By designing good abstractions, building high-assurance systems can be made accessible with provable correctness and security guarantees

Takeaway

- By designing good abstractions, building high-assurance systems can be made accessible with provable correctness and security guarantees
- **Theory**: sound and secure language design

Takeaway

- By designing good abstractions, building high-assurance systems can be made accessible with provable correctness and security guarantees
- **Theory**: sound and secure language design
- **Implementation**: type checker and end-to-end compiler