Realistic Realizability:

Specifying ABIs You Can Count On

Andrew Wagner, Zachary Eisbach, Amal Ahmed

Northeastern University

October 29, 2024 @ POPV Seminar, Boston University

Realistic Realizability:

Specifying ABIs You Can Count On

Andrew Wagner, Zachary Eisbach, Amal Ahmed

Northeastern University

October 29, 2024 @ POPV Seminar, Boston University

> "The standard is haunted … by that **Three Letter Demon**.😈
> … a contract was forged in blood." 🩸
>
> *– JeanHeyd Meneide, WG14 C/C++ Compatibility Chair*

What is an ABI?

> **Application Binary Interface (ABI)**
>
> `The run-time contract for using` `a particular API` (or for an entire library), including things like symbol names, **calling conventions**, and type **layout** information.
>
> — Swift

What is an ABI?

Application Binary Interface (ABI)

`The run-time contract for using` `a particular API` (or for an entire library), including things like symbol names, $\underbrace{\textbf{calling conventions}}_{\textbf{Behavior}}$, and type **layout** information.

— Swift

What is an ABI?

> **Application Binary Interface (ABI)**
>
> The run-time contract for using `a particular API` (or for an entire library), including things like symbol names, **calling conventions**, and type **layout** information.
>
> **Behavior**
>
> — Swift

```
foo : (Int, Int) -> Int
```

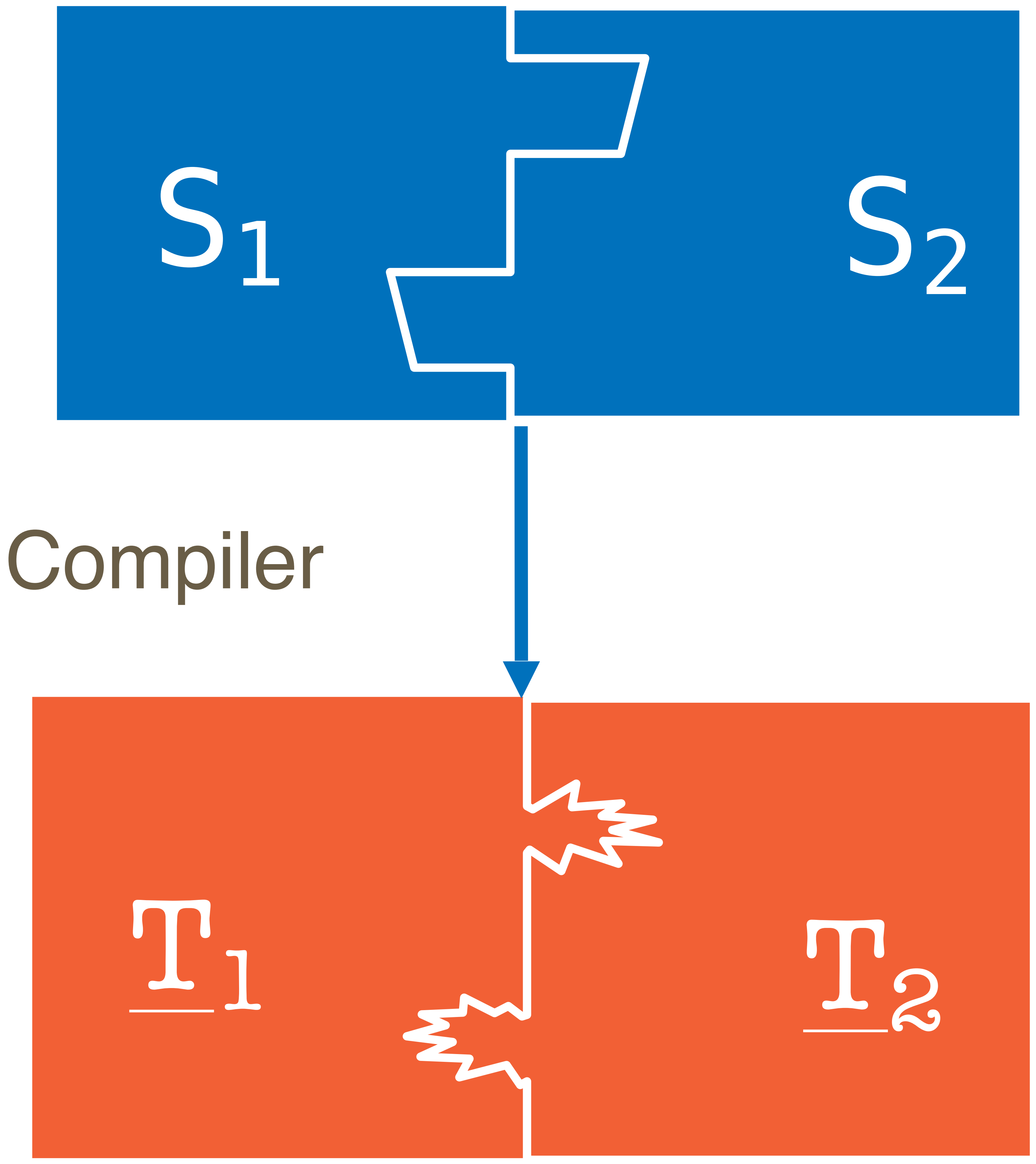
Compiler?

```
int foo(int fst, int snd)

int foo(int indir[])

void foo(int indir[], int *ret)
```

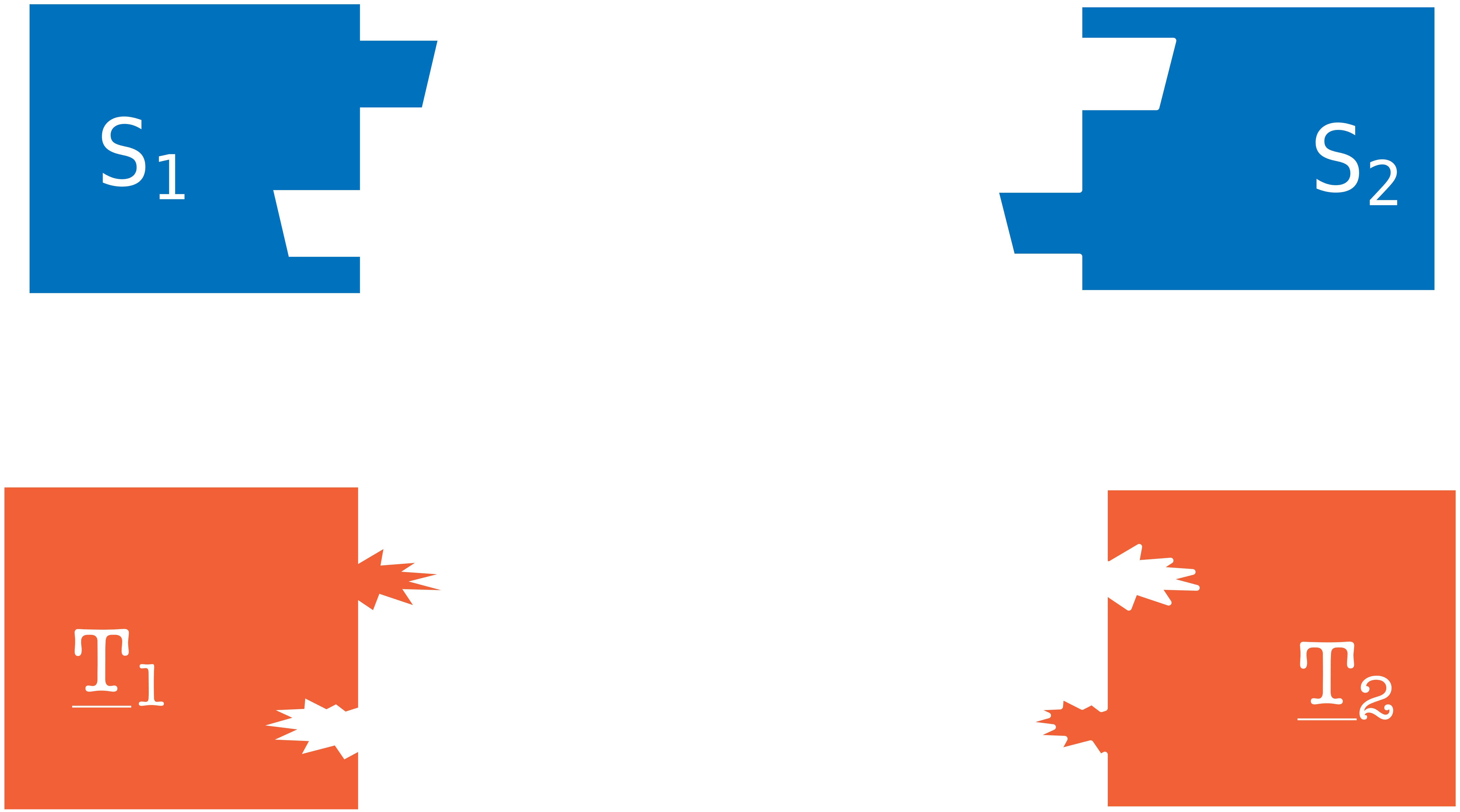
Why Use an ABI?

Why Use an ABI? Interoperability

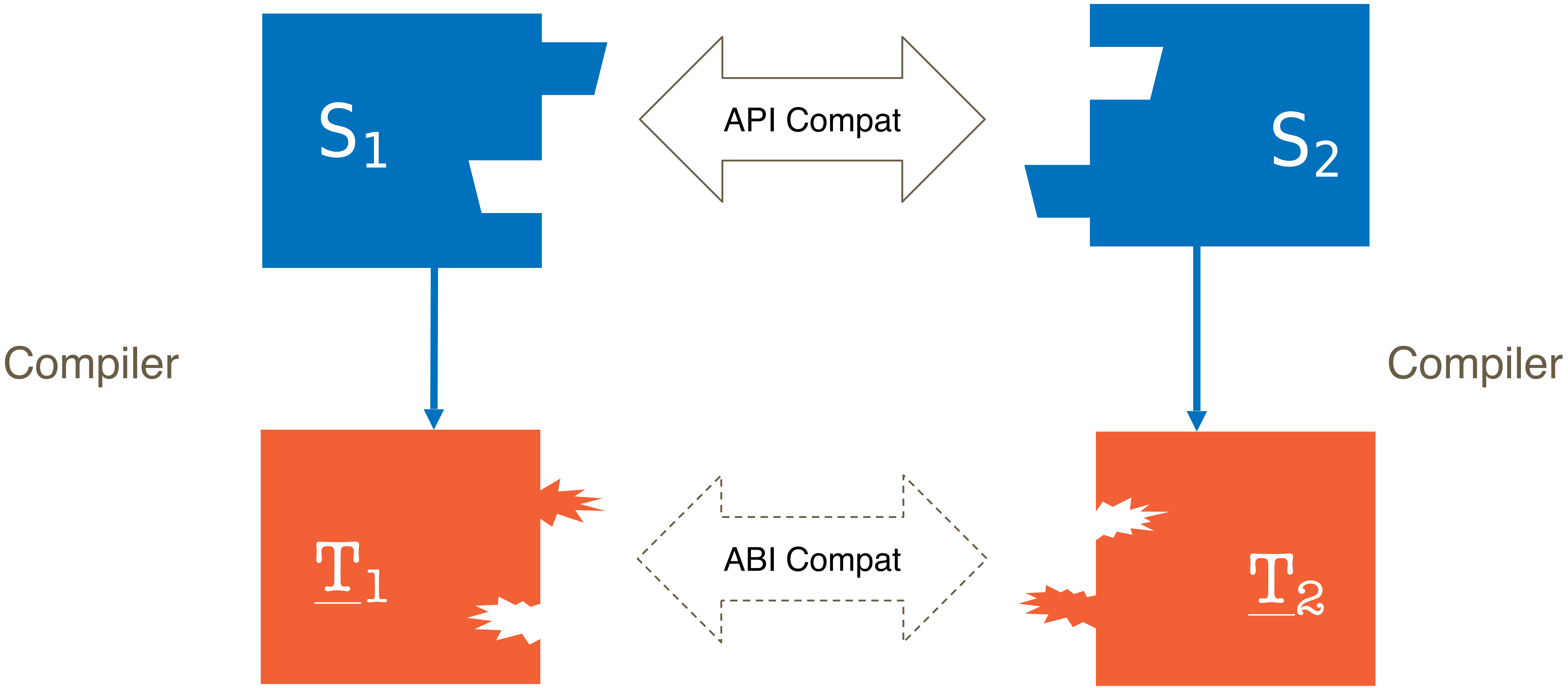
Why Use an ABI? Interoperability



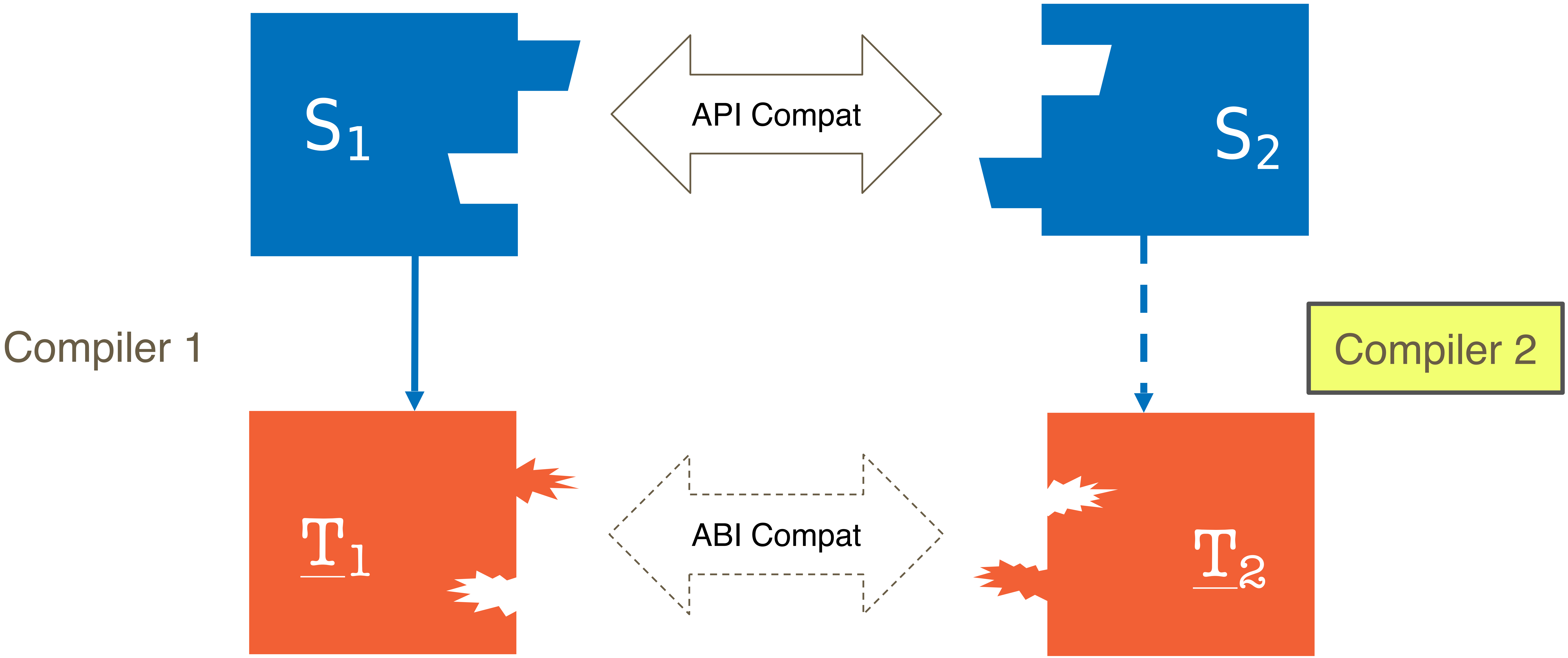
Why Use an ABI? Interoperability



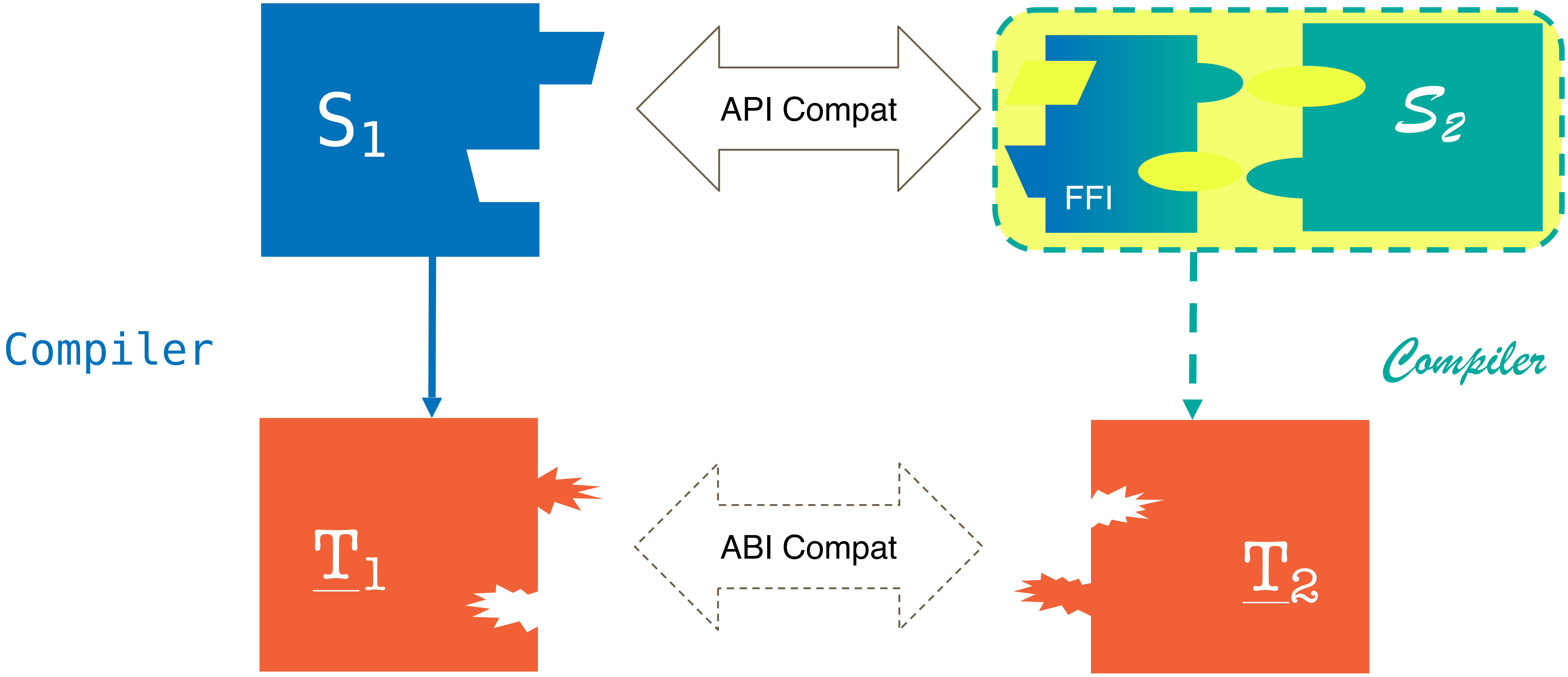
Why Use an ABI? Interoperability for Components



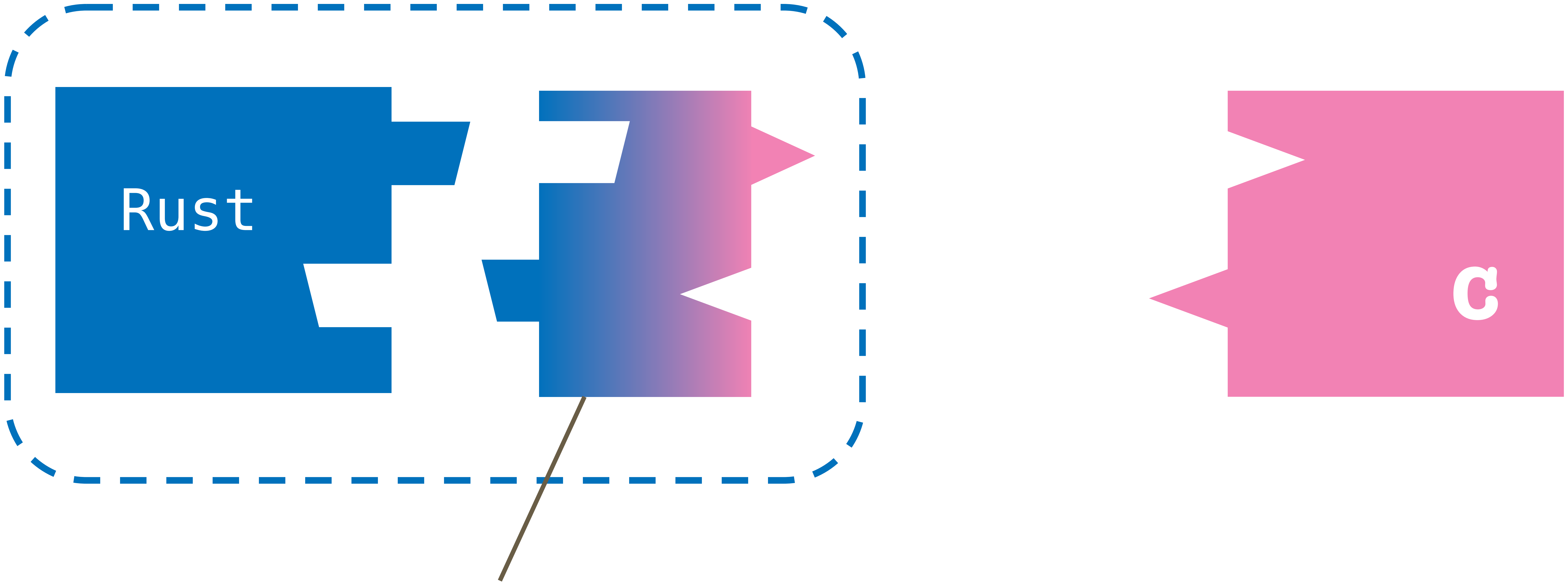
Why Use an ABI? Interoperability for Compilers



Why Use an ABI? Interoperability for Languages

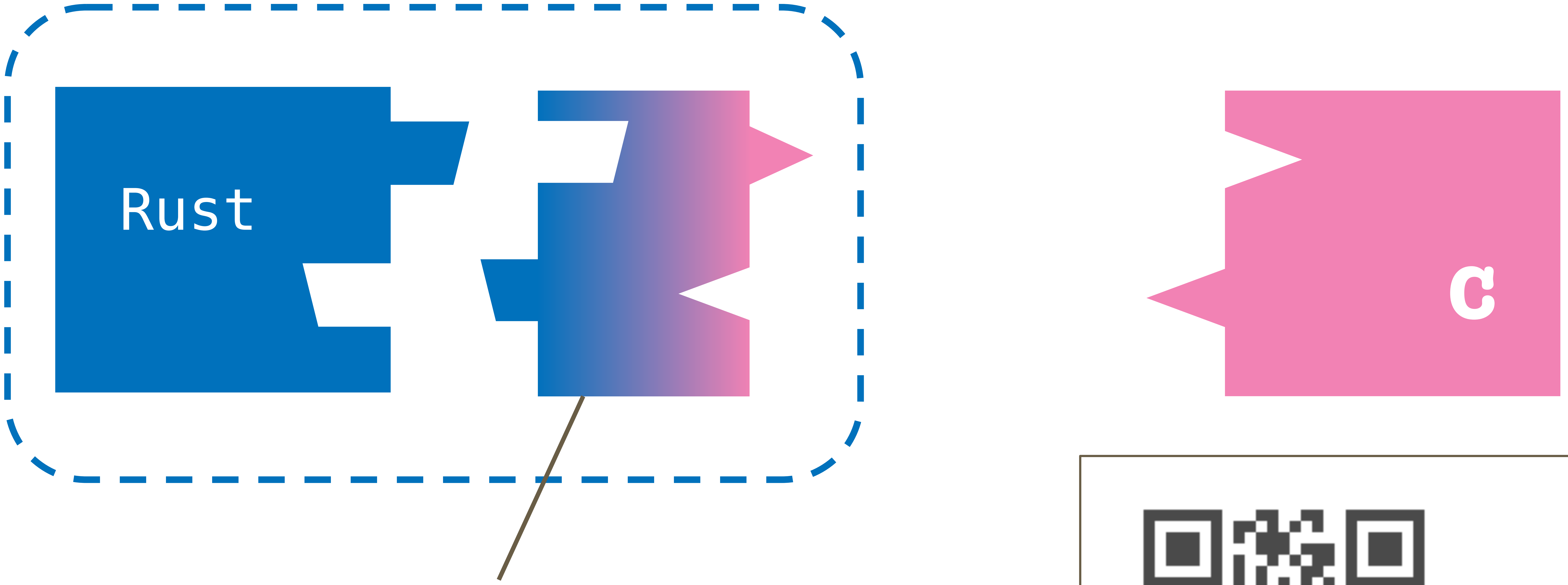


Foreign Function Interfaces



Foreign Function Interface (**FFI**)

Foreign Function Interfaces



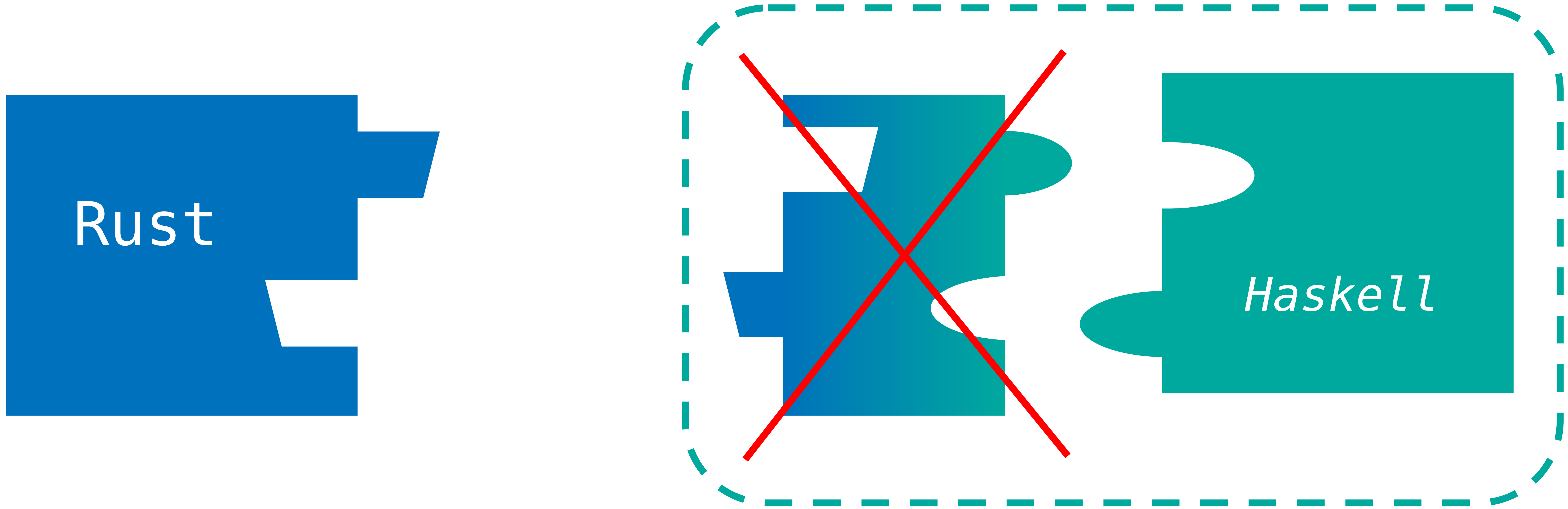
Foreign Function Interface (**FFI**)

Linking Types
Patterson, Wagner, Ahmed
TyDe '23

Safe Foreign Function Interfaces

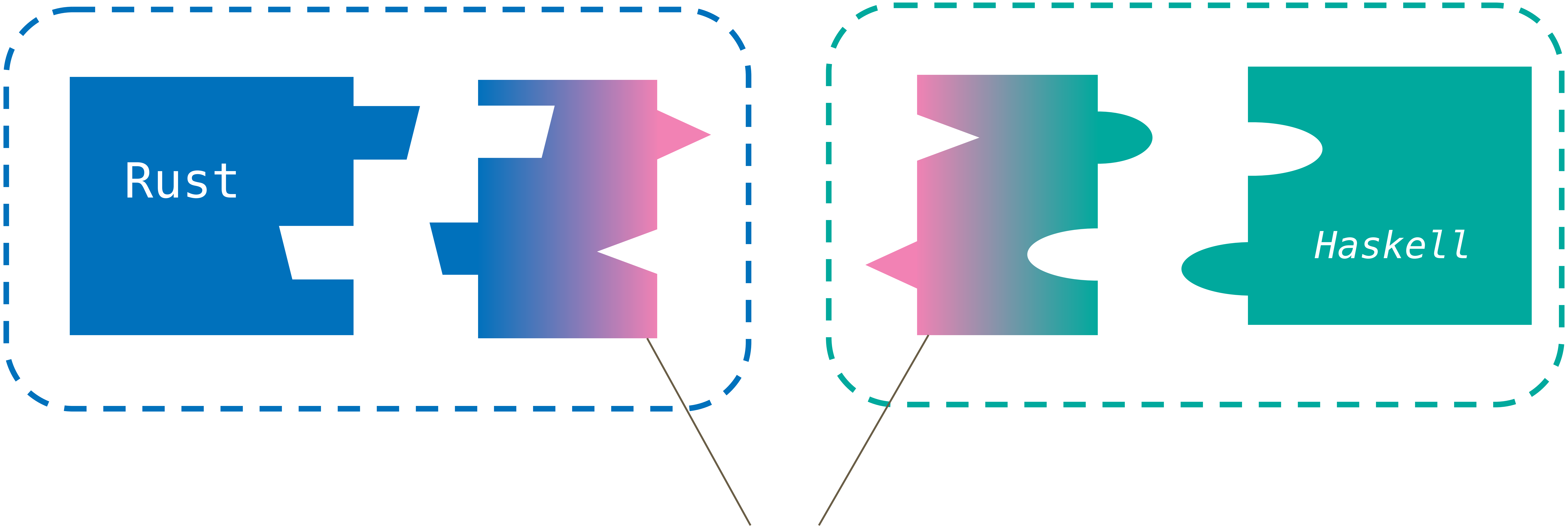
“Rewrite it in Rust!”

~~Safe~~ Foreign Function Interfaces



“Rewrite it in Rust!”

C, the Rosetta Stone?



More C Code 🤬

Why C?

Why C?

Shallow Answer: Because every language speaks C

Why C?

Shallow Answer: Because every language speaks C

But <u>Why</u> Does Every Language Speak C?

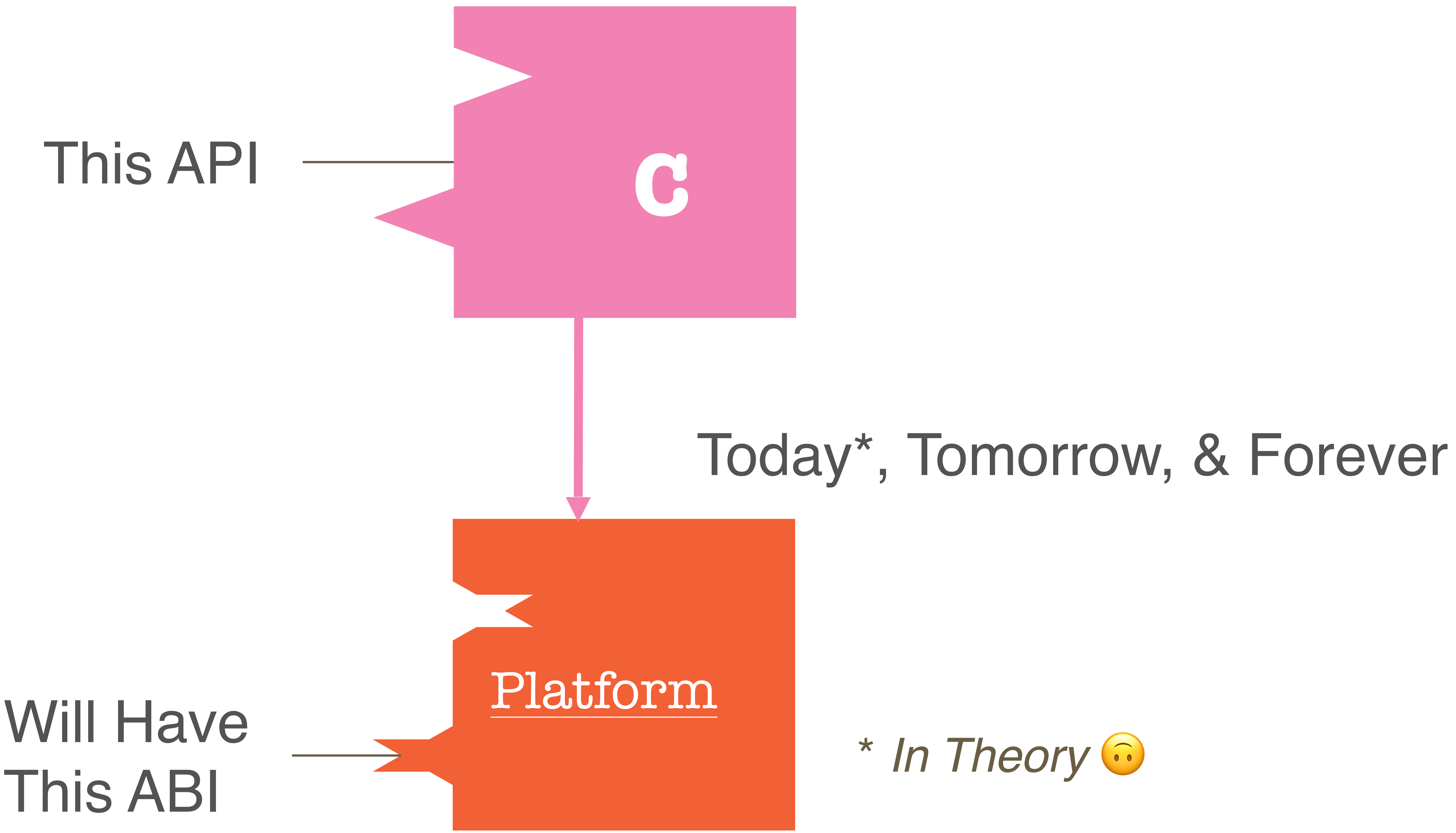
Why C?

Shallow Answer: Because every language speaks C

But Why Does Every Language Speak C?

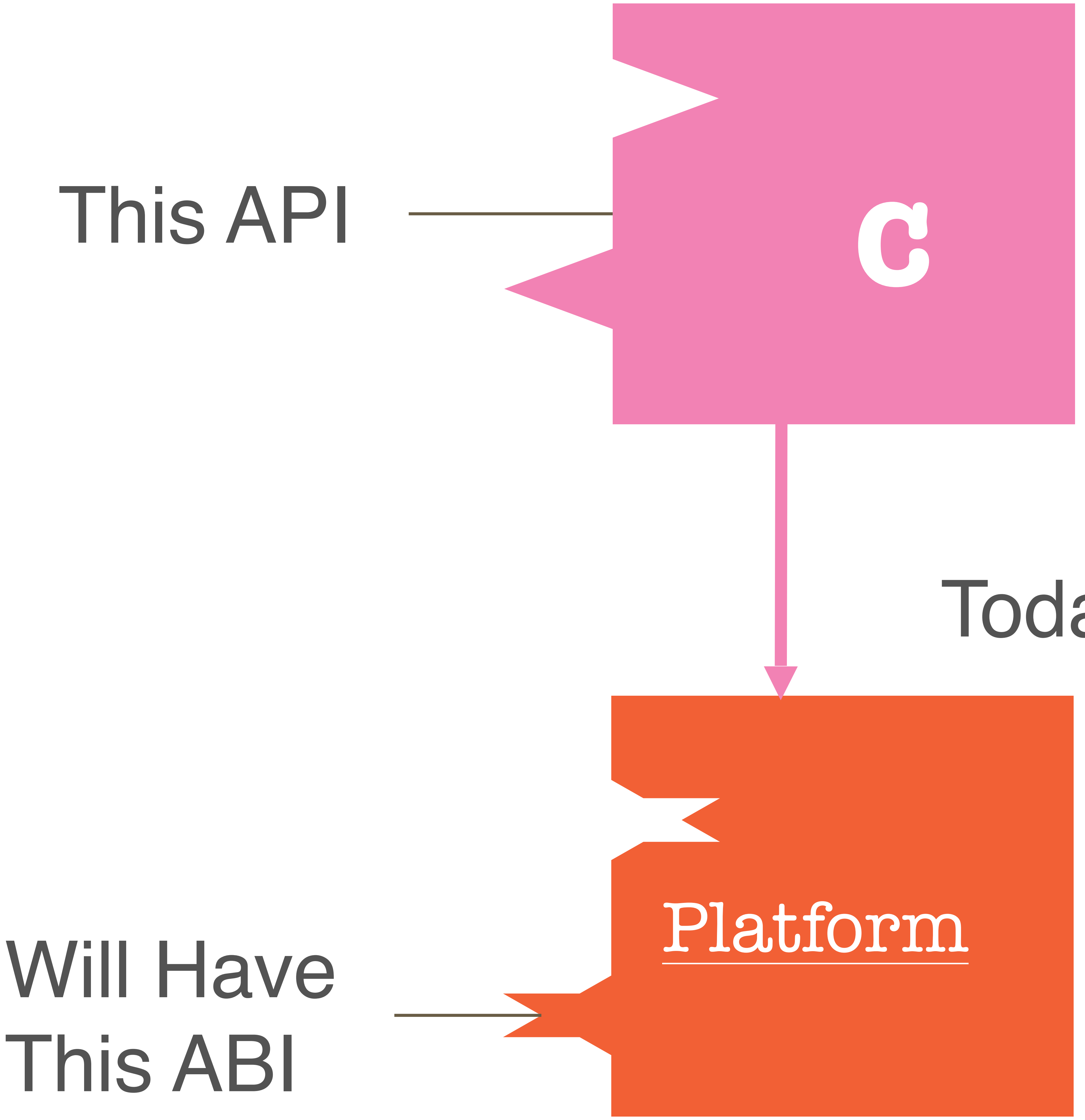
Deeper Answer: Because C is committed to ABI stability

ABI Stability



* *In Theory* 🙃

ABI Stability



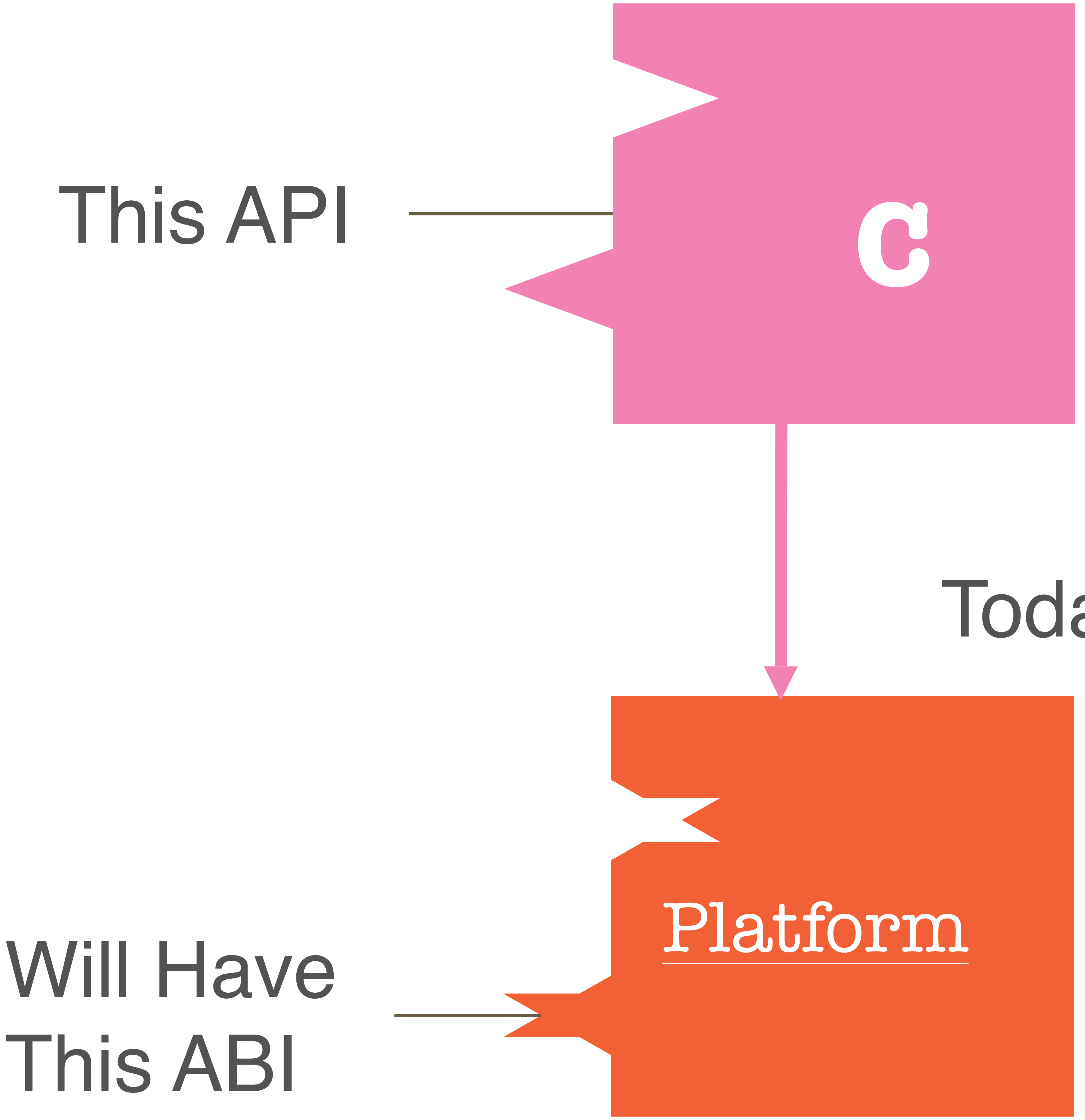
n2526: use const for data from the library that shall not be modified

Submitter: Philipp Klaus Krause
Submission Date: 2020-05-11

Today*, Tomorrow, & Forever

* *In Theory* 🙃

ABI Stability



Today*, Tomorrow, & Forever

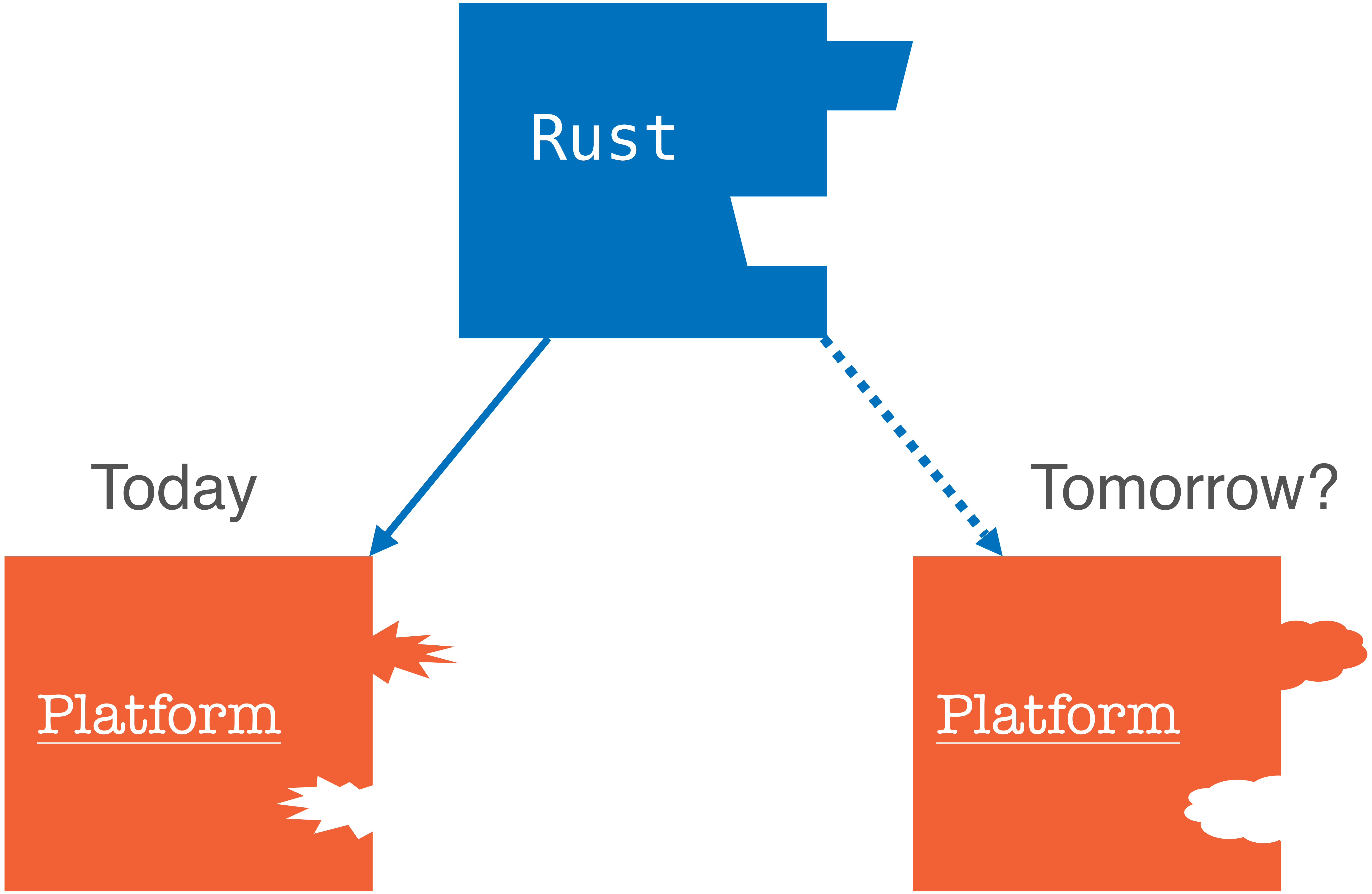
* *In Theory* 🙃

Rejected: New Warnings = Bad!

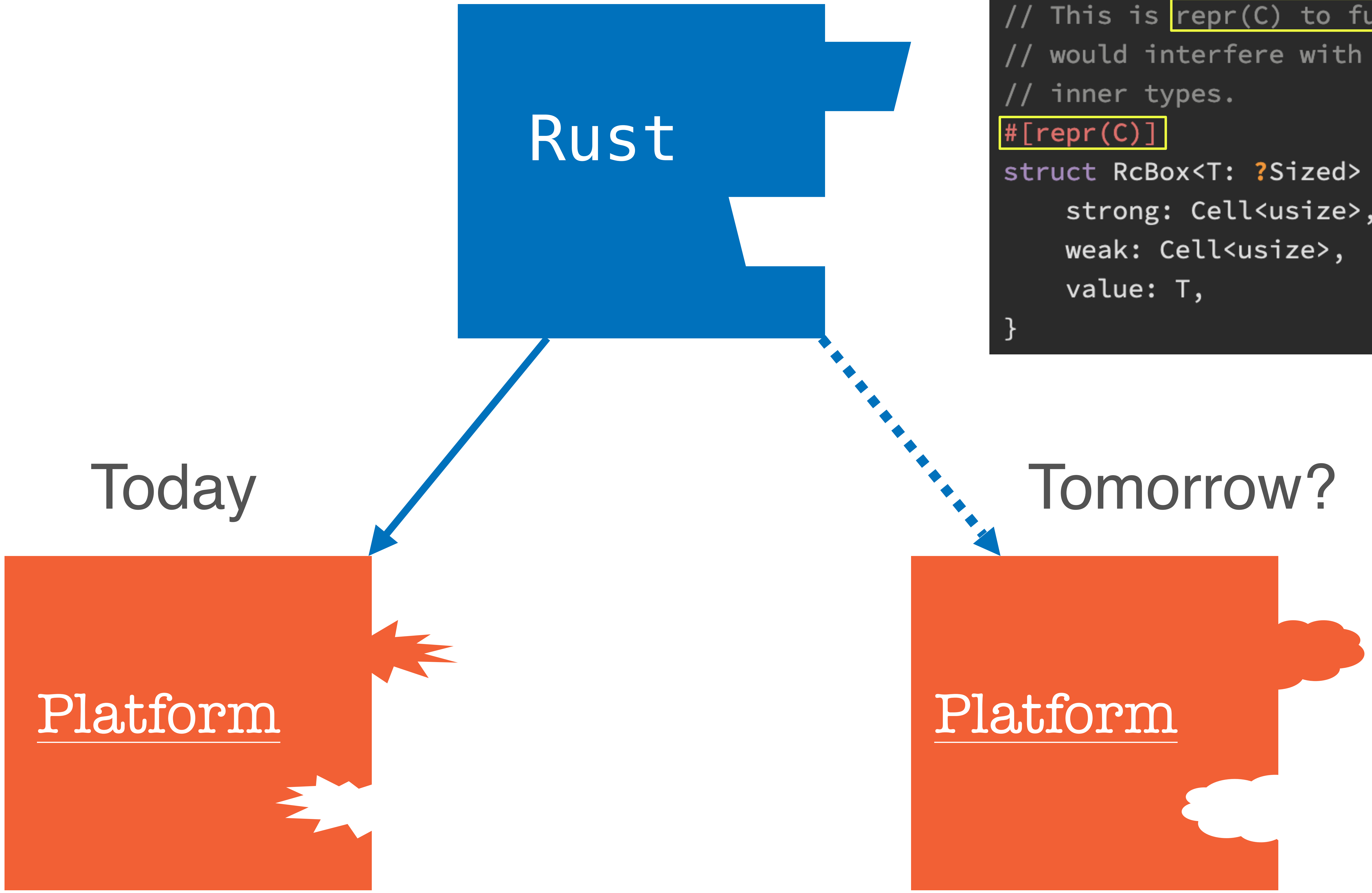
n2526: use const for data from the library that shall not be modified

Submitter: Philipp Klaus Krause
Submission Date:2020-05-11

ABI Instability



ABI Instability




```
// This is repr(C) to future-proof against possible field-reordering, whi
// would interfere with otherwise safe [into|from]_raw() of transmutable
// inner types.
#[repr(C)]
struct RcBox<T: ?Sized> {
    strong: Cell<usize>,
    weak: Cell<usize>,
    value: T,
}
```

To Stabilize or Not to Stabilize, That Is the Question

Pros

+ Precise control over interface to other languages
+ Proper support for shared libraries

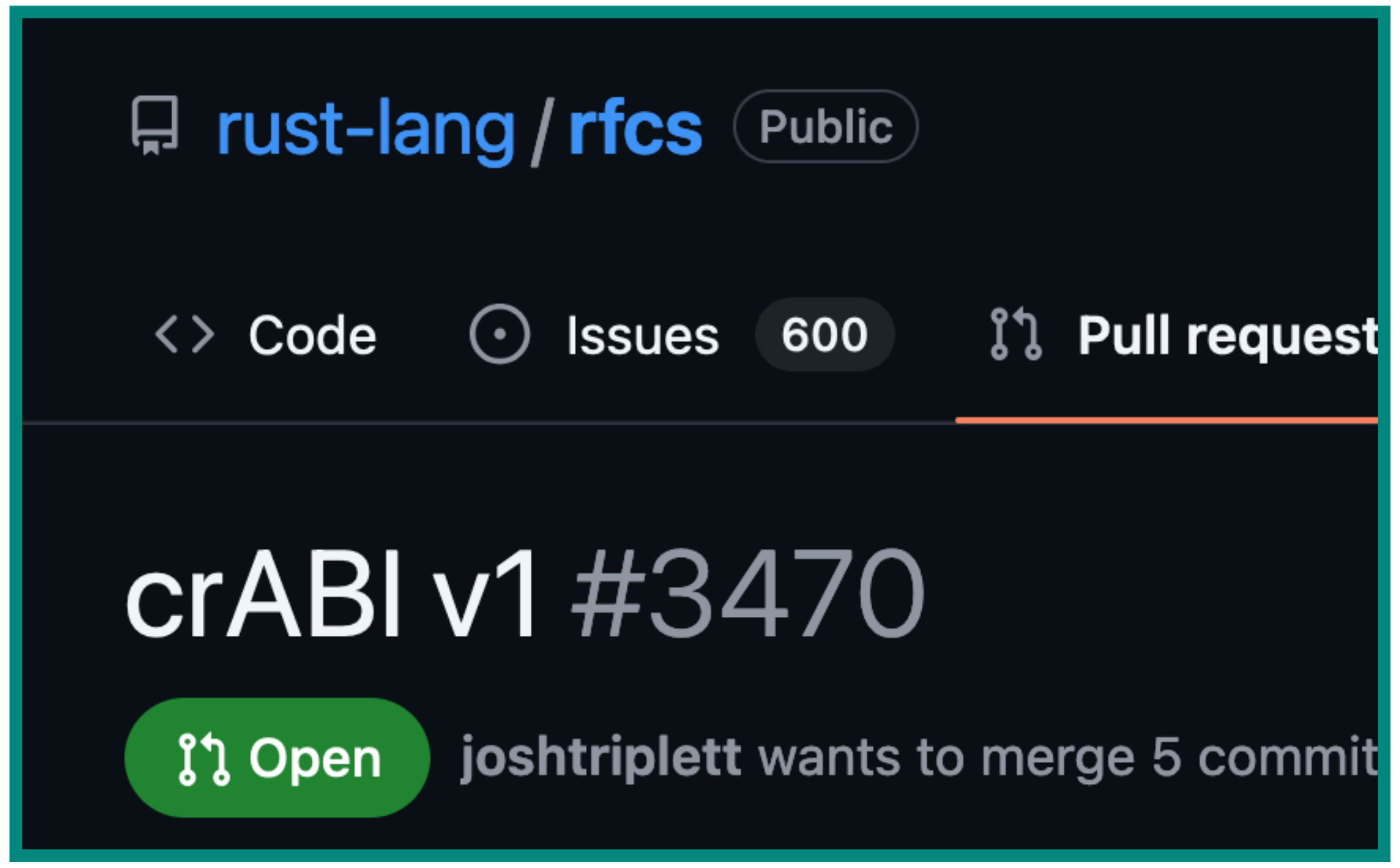
Cons

− Can stunt language growth
− Limits compiler optimizations
− Tension between flexibility and performance
− Pressure on library developers

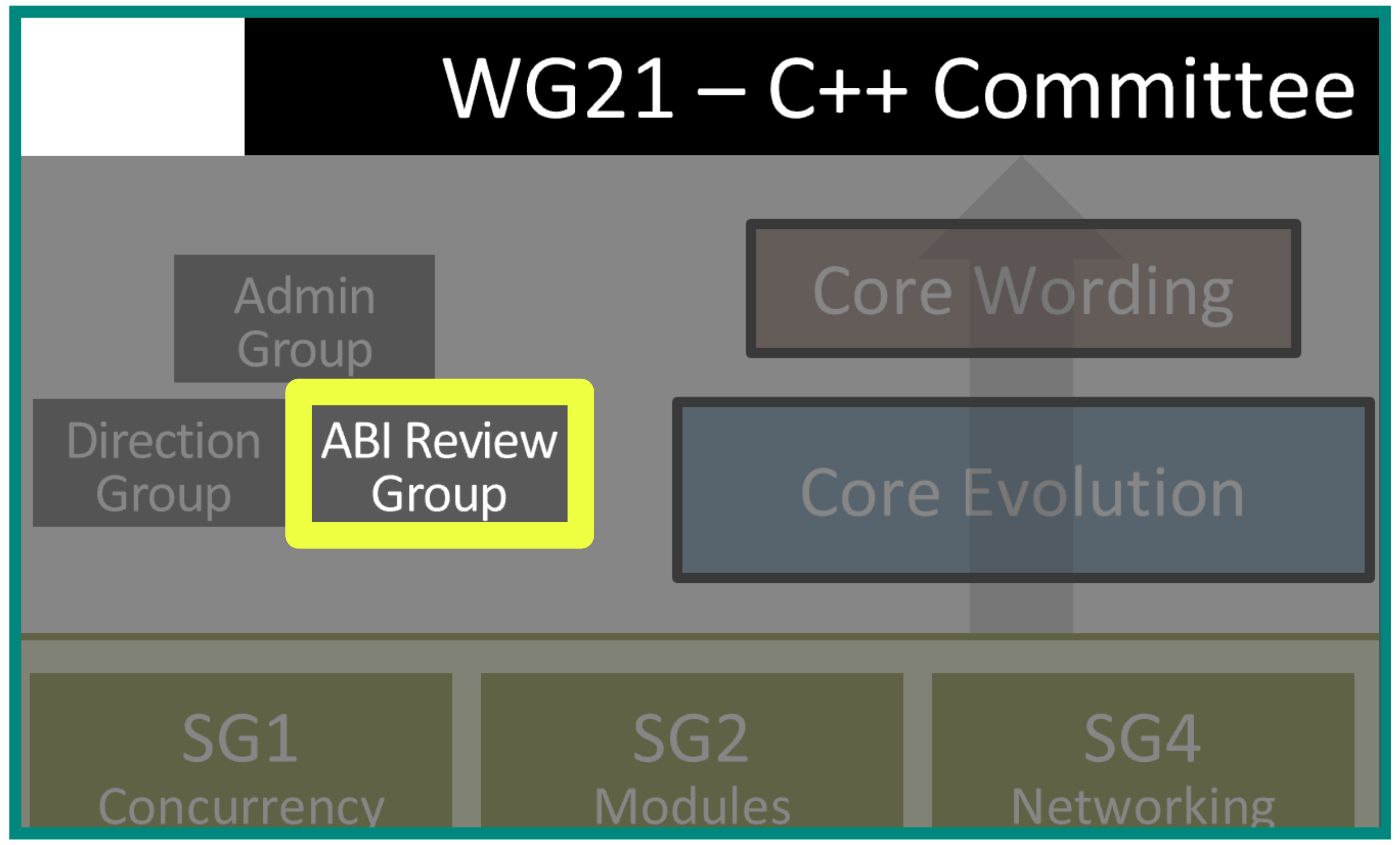
Who is Designing an ABI?

Swift

Rust



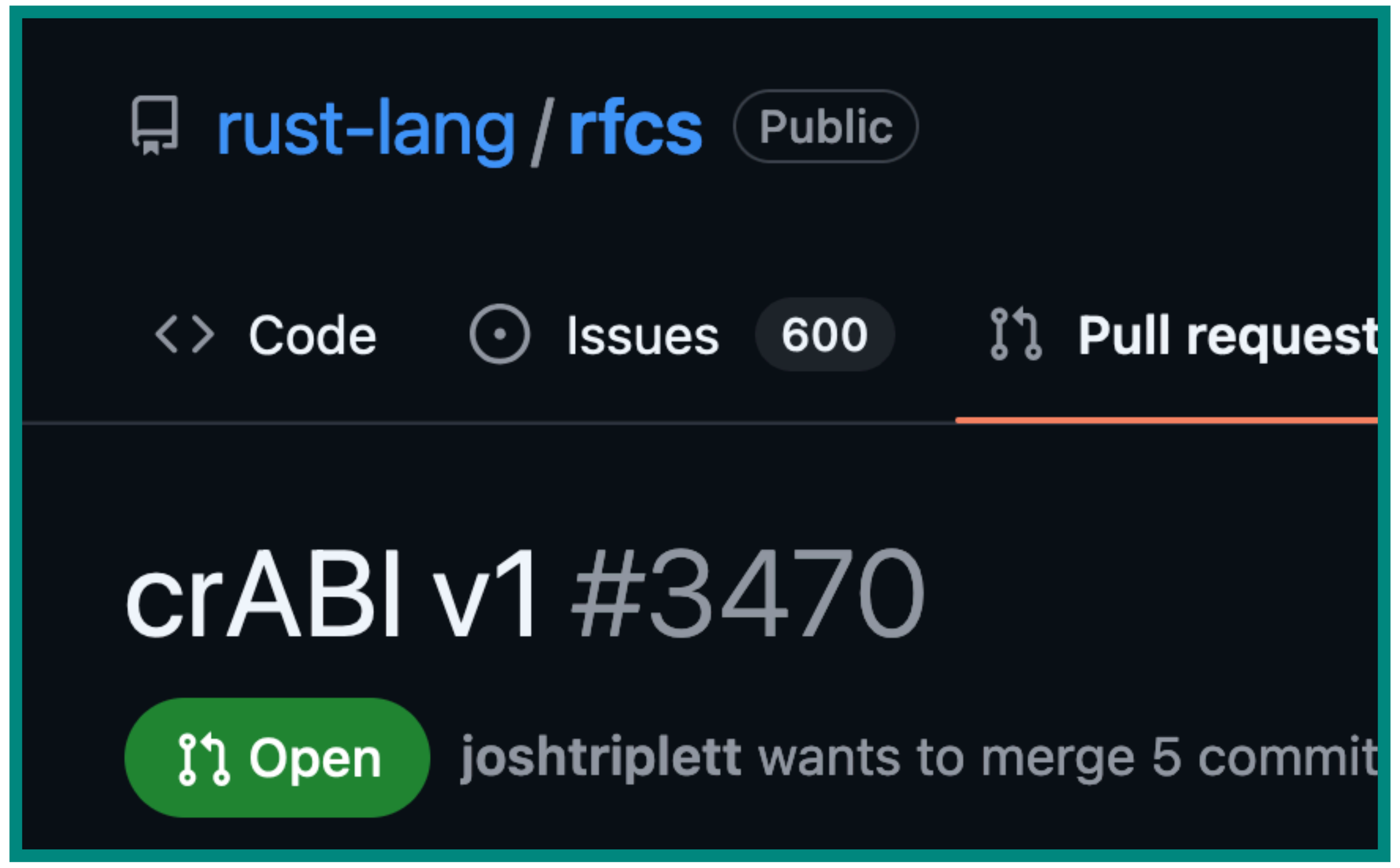
C++



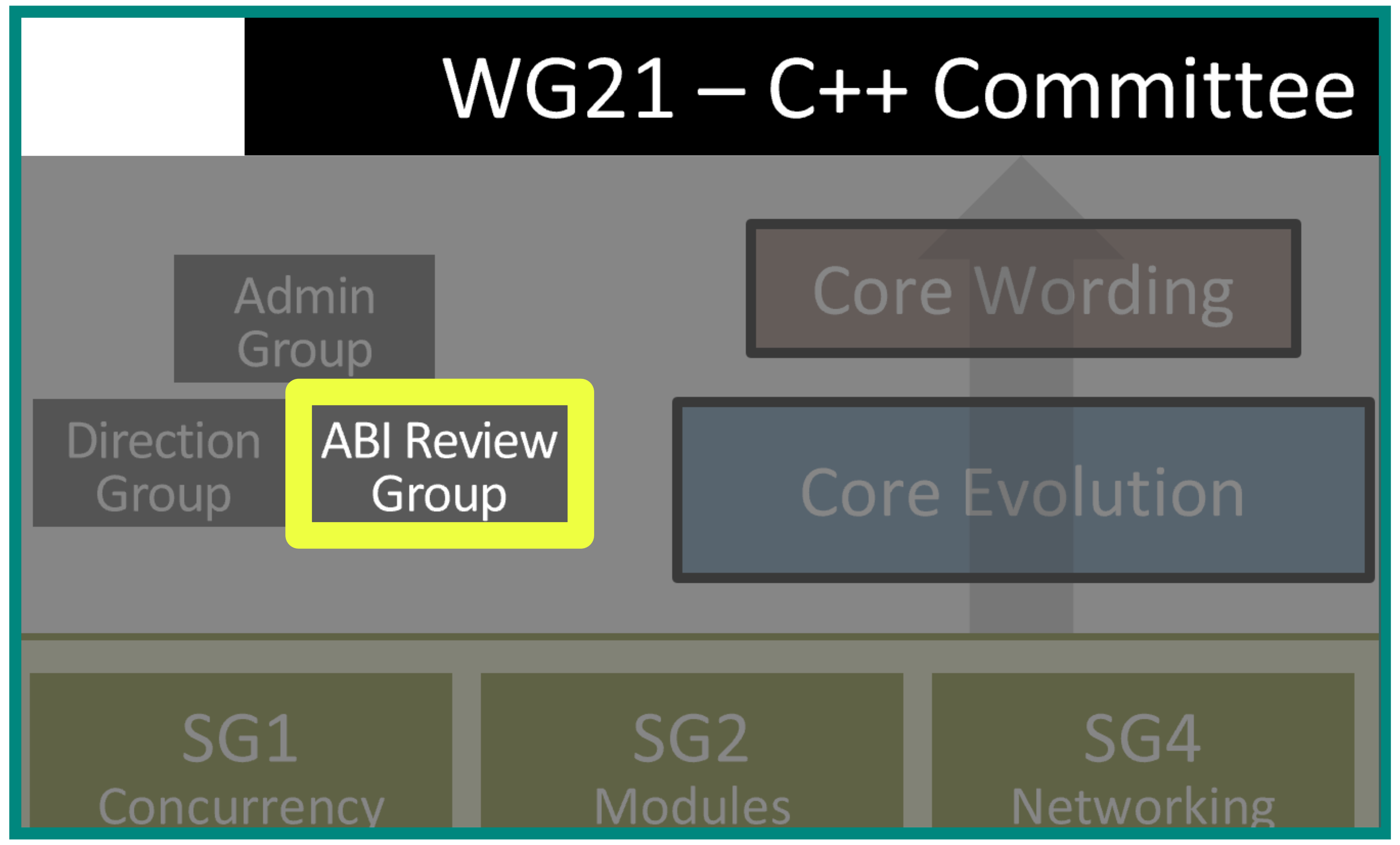
Who is Designing an ABI?

Swift

Rust



C++

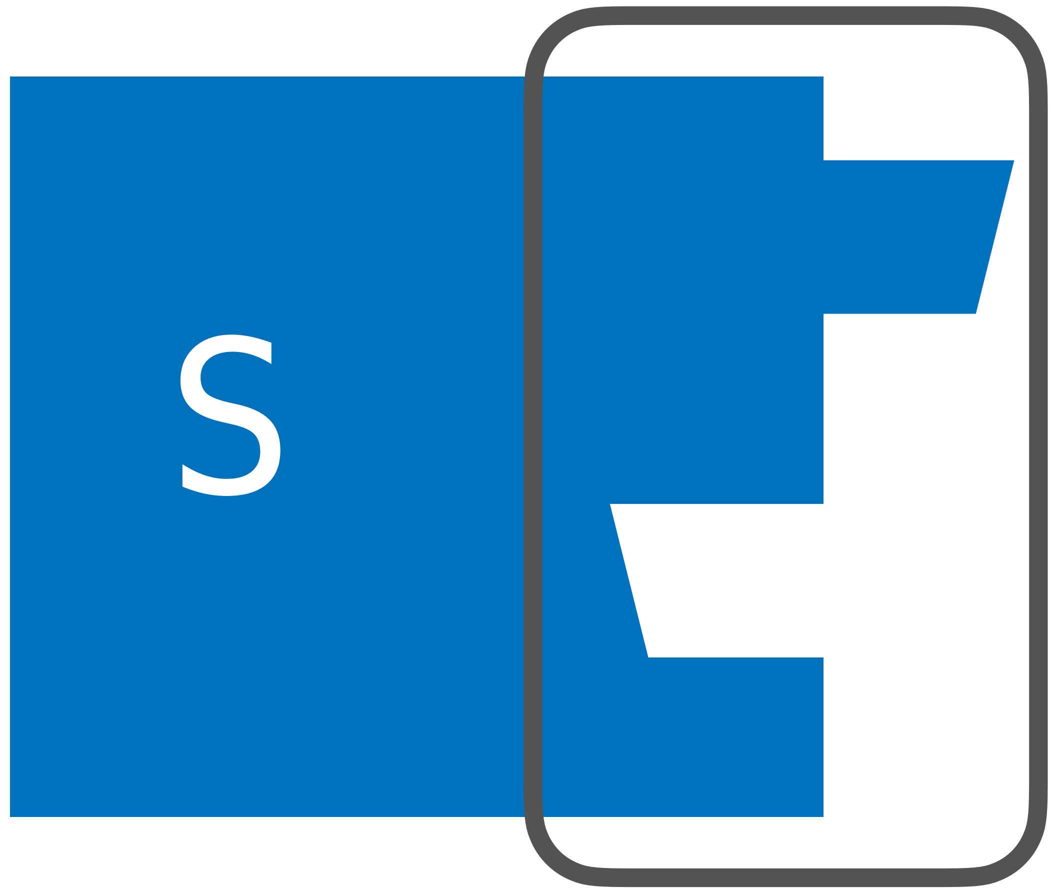


Richer Types ⇒ Richer ABIs?

How Should We Specify an ABI?

The run-time contract for using a particular API

How Should We Specify an ABI?

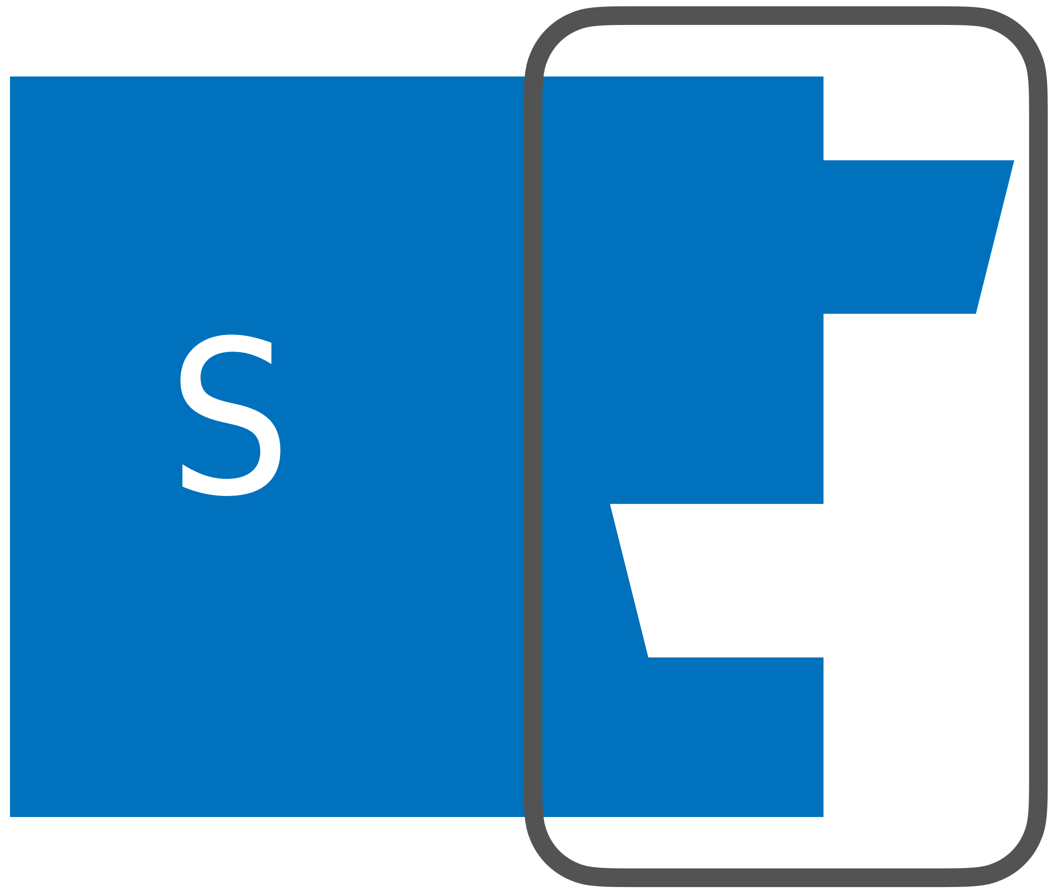


The run-time contract for using a particular API

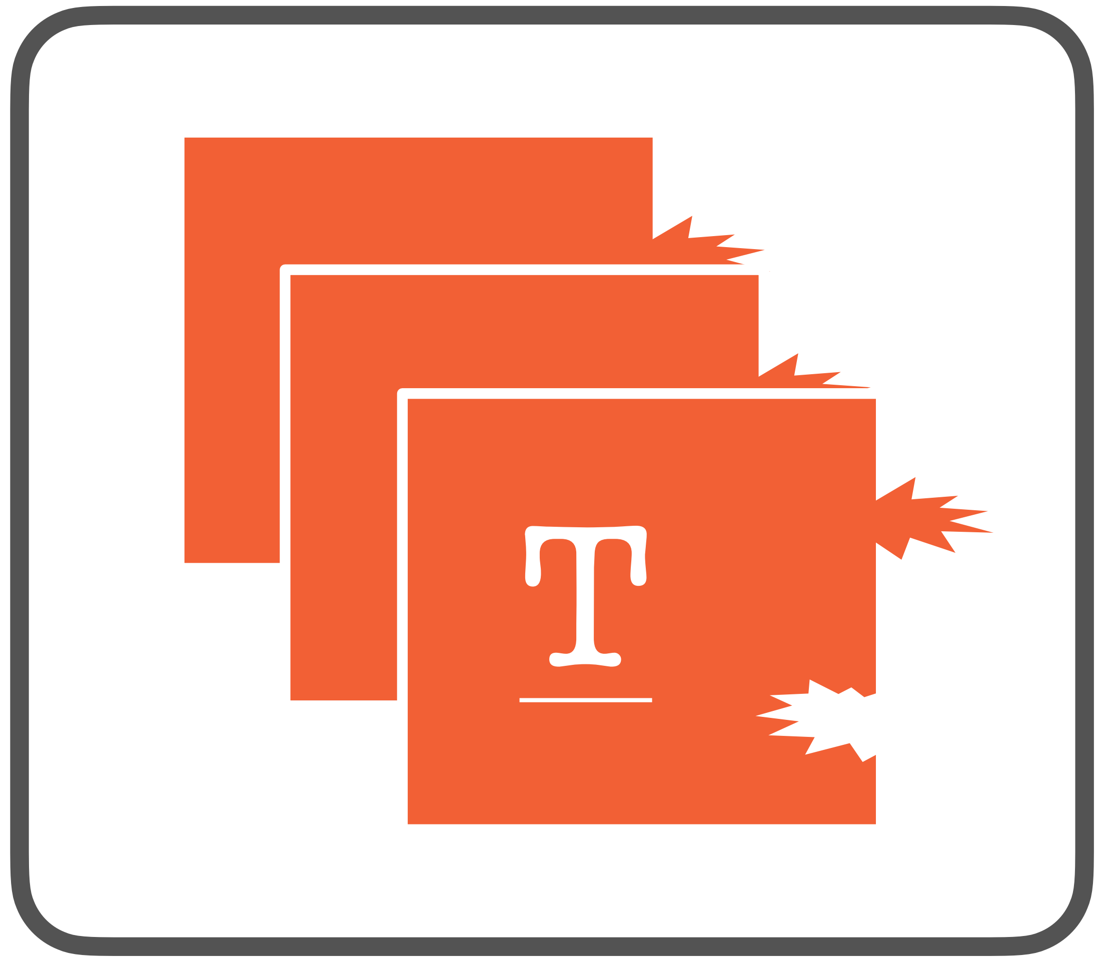
This Type τ

How Should We Specify an ABI?

The run-time contract for using a particular API



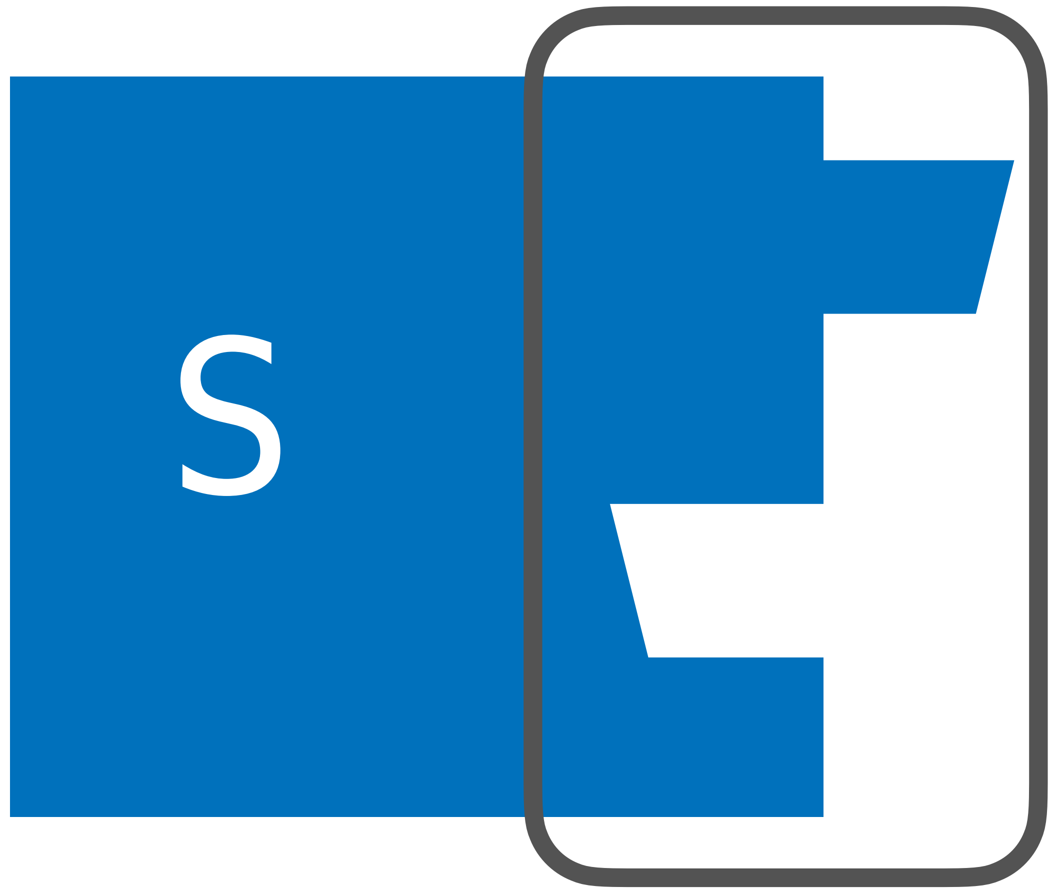
This Type τ



Is *Realized By* These Target Programs

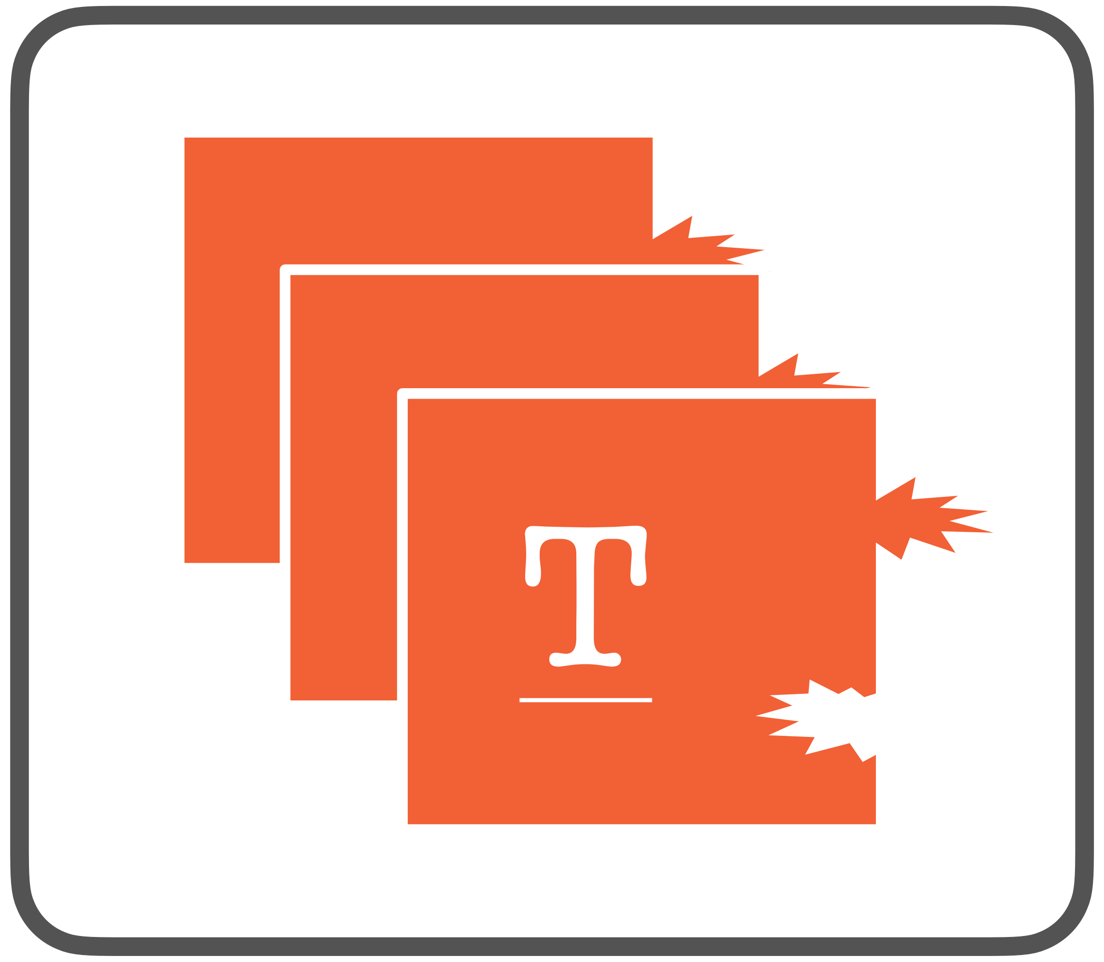
$[\![\tau]\!] = \{ \underline{e} \mid \dots \}$

How Should We Specify an ABI?



The run-time contract for using `a particular API`

This Type τ



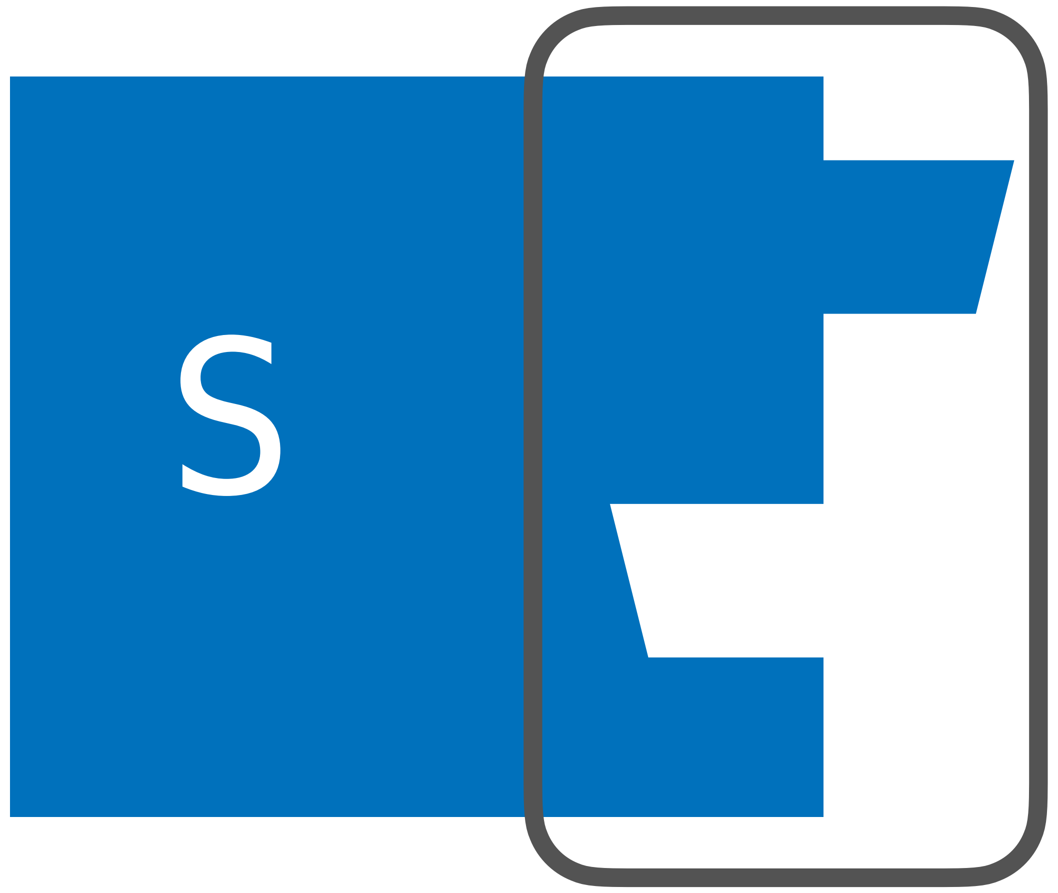
Is *Realized By* These Target Programs

$[\![\tau]\!] = \{\ \underline{e} \mid \ldots\ \}$

Semantic Typing using *Realistic Realizability* [Benton06]

How Should We Specify an ABI?

`The run-time contract for using a particular API`

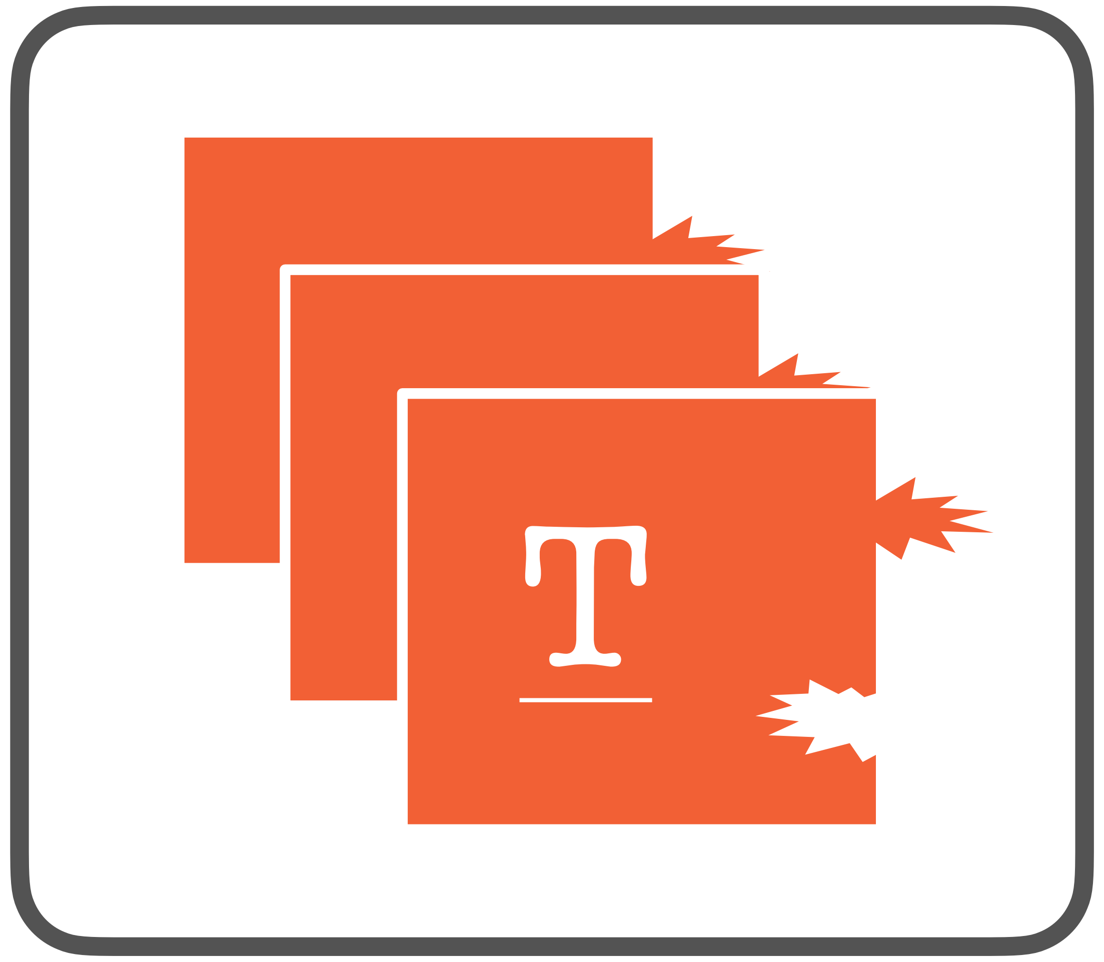


This Type τ

Our Proposal

$\underline{e}$ is **ABI compliant** with τ if

$$\underline{e} \in [\![\tau]\!]$$



Is *Realized By* These Target Programs

$[\![\tau]\!] = \{ \underline{e} \mid \ldots \}$

Semantic Typing using *Realistic Realizability* [Benton06]

Interlude: Semantic Typing via Realizability

$$\Gamma \vdash \quad e \quad : T$$ *Syntactic typing*

Interlude: Semantic Typing via Realizability

$\Gamma \vdash e : T$ *Syntactic typing*

$\Gamma \vDash e : T$ *Semantic typing / Logical relation*

Interlude: Semantic Typing via Realizability

$\Gamma \vdash e : T$ *Syntactic typing*

$\Gamma \vDash e : T$ *Semantic typing / Logical relation*

$$\left\{ \text{“}\mathit{Prestate\ like}\ \Gamma\text{”} \right\} \quad e \quad \left\{ v.\ \text{“}v\ \mathit{like}\ T\text{”} \right\}$$

Interlude: Semantic Typing via Realizability

$$\Gamma \vdash \quad e \quad : T$$

Syntactic typing

$$\Gamma \vDash \quad e \quad : T$$

Semantic typing / Logical relation

$$\left\{ \text{"Prestate like } \Gamma \text{"} \right\} \quad e \quad \left\{ v.\ \text{"} v \text{ like } T \text{"} \right\}$$

$$(\Gamma = \overline{x : T_x})$$

$$\left\{ \bigstar\, \overline{\mathcal{V}[\![T_x]\!](x)} \right\} \quad e \quad \left\{ v.\ \mathcal{V}[\![T]\!](v) \right\}$$

Case Study

- `Functional Source Language`
  - Recursive records and variants, higher-order recursive functions
- C-like Target
  - Block-based memory, pointer arithmetic
- Automatic Reference Counting (ARC) Implementation
  - Values are boxed and reference-counted
  - Separation logic abstractions for reasoning about RC

Case Study

- `Functional Source Language`
    - Recursive records and variants, higher-order recursive functions
- C-like Target
    - Block-based memory, pointer arithmetic
- Automatic Reference Counting (ARC) Implementation
    - Values are boxed and reference-counted
    - Separation logic abstractions for reasoning about RC

Layout + Behavior

The run-time contract for using `a particular type`

e is **ABI compliant** with τ if

$e \in [\![\tau]\!]$

Automatic Reference Counting Compiler

$\boxed{\Gamma \vdash e : T}$

Type system is internally linear

$$\frac{\Gamma_1 \vdash e_1 : T_1 \qquad \Gamma_2, x : T_1 \vdash e_2 : T_2 \qquad \Gamma_2 \not\ni x}{\Gamma_1, \Gamma_2 \vdash \mathsf{let}\, x = e_1; e_2 : T_2}\ (\mathsf{let}^+) \qquad \qquad x : T \vdash x : T \quad (\mathsf{var}^+)$$

Inspired by *Perceus* [Reinking et. al. 2021]

Automatic Reference Counting Compiler

$\boxed{\Gamma \vdash e : T}$

Type system is internally linear

$$\frac{\Gamma_1 \vdash e_1 : T_1 \qquad \Gamma_2, x : T_1 \vdash e_2 : T_2 \qquad \Gamma_2 \not\ni x}{\Gamma_1, \Gamma_2 \vdash \mathsf{let}\, x = e_1; e_2 : T_2} \ (\mathsf{let}^+) \qquad \overline{x : T \vdash x : T} \ (\mathsf{var}^+)$$

Explicit occurrences of weakening and contraction

$$\frac{\Gamma \ni x : T' \qquad \Gamma, x : T' \vdash e : T}{\Gamma \vdash e : T} \ (\mathsf{dup}^+) \qquad \frac{\Gamma \vdash e : T}{\Gamma, x : T' \vdash e : T} \ (\mathsf{drop}^+)$$

Inspired by *Perceus* [Reinking et. al. 2021]

Automatic Reference Counting Compiler

$$\boxed{\Gamma \vdash e : T \rightsquigarrow e}$$

Type system is internally linear

$$\frac{\Gamma_1 \vdash e_1 : T_1 \rightsquigarrow e_1 \quad \Gamma_2, x : T_1 \vdash e_2 : T_2 \rightsquigarrow e_2 \quad \Gamma_2 \not\ni x}{\Gamma_1, \Gamma_2 \vdash \mathsf{let}\, x = e_1; e_2 : T_2 \rightsquigarrow \mathtt{const}\ x = e_1;\ e_2}\ (\mathsf{let}^+)$$

$$x : T \vdash x : T \rightsquigarrow x\ (\mathsf{var}^+)$$

Explicit occurrences of weakening and contraction

$$\frac{\Gamma \ni x : T' \quad \Gamma, x : T' \vdash e : T \rightsquigarrow e}{\Gamma \vdash e : T \rightsquigarrow \underline{\mathtt{dup}}_{T'}\ (x)\ ; e}\ (\mathsf{dup}^+)$$

$$\frac{\Gamma \vdash e : T \rightsquigarrow e}{\Gamma, x : T' \vdash e : T \rightsquigarrow \underline{\mathtt{drop}}_{T'}\ (x)\ ; e}\ (\mathsf{drop}^+)$$

Inspired by *Perceus* [Reinking et. al. 2021]

References: Layout

Location ℓ is a reference to an object that behaves like type T

Ref. Count

Object Data

$$\mathcal{R}[\![T]\!](\ell) \approx$$

ℓ	$\ell + 1$ ...
c	$\mathcal{O}[\![T]\!](\ell + 1)$

References: Layout

Location ℓ is a reference to an object that behaves like type T

$$\mathcal{R}[\![\mathbb{Z}]\!](\ell) \approx$$

Ref. Count	Object Data
ℓ	$\ell + 1$ ...
c	$\mathcal{O}[\![\mathbb{Z}]\!](\ell + 1)$

$$\mathcal{O}[\![\mathbb{Z}]\!](\ell + 1) = \exists\, n.\ \ell + 1 \mapsto n$$

Later:
Unboxed types

References: Ownership + Sharing

$$\mathcal{R}\,[\![\,T\,]\!](\boldsymbol{\ell})$$

ℓ	ℓ **+ 1 ...**
c	$\mathcal{O}\,[\![\,T\,]\!](\boldsymbol{\ell}+1)$

References: Ownership + Sharing

Single reference represents one
share of underlying object

$$\mathcal{R}\,[\![\,T\,]\!](\boldsymbol{\ell})$$

ℓ	$\ell + 1$ ...
≥ 1	$\mathcal{O}\,[\![\,T\,]\!](\boldsymbol{\ell} + 1)$

References: Ownership + Sharing

Single reference represents one share of underlying object

$$\mathcal{R}[\![T]\!](\ell)$$

ℓ	ℓ + 1 ...
≥ 3	$\mathcal{O}[\![T]\!](\ell + 1)$

$$\mathcal{R}[\![T]\!](\ell)$$

$$\mathcal{R}[\![T]\!](\ell)$$

References: Ownership + Sharing

Single reference represents one share of underlying object

$\mathcal{R}[\![T]\!](\ell)$

ℓ	ℓ + 1 ...
≥ 3	$\mathcal{O}[\![T]\!](\ell + 1)$

$\mathcal{R}[\![T]\!](\ell)$

$\mathcal{R}[\![T]\!](\ell)$

Reference confers permission to increment count & acquire more shares

RC-INCR

$$\left\{\mathcal{R}[\![T]\!](\ell)\right\} \ {++}\ell\ \left\{n.\ \ulcorner n > 1 \urcorner \star \mathcal{R}[\![T]\!](\ell) \star \mathcal{R}[\![T]\!](\ell)\right\}$$

References: Ownership + Sharing

ℓ	ℓ + 1 ...
≥ 3	$\mathcal{O}[\![T]\!](\ell + 1)$

$\mathcal{R}[\![T]\!](\ell)$

$\mathcal{R}[\![T]\!](\ell)$

$\mathcal{R}[\![T]\!](\ell)$

References: Ownership + Sharing

$$\mathcal{R}\,[\![T]\!](\ell)$$

ℓ	$\ell + 1 \ldots$
≥ 1	$\mathcal{O}\,[\![T]\!](\ell + 1)$

Reference confers permission to decrement count & release shares

RC-DECR

$$\left\{\mathcal{R}[\![T]\!](\ell)\right\}\ \texttt{--}\ell\ \left\{n.\ \right\}$$

References: Ownership + Sharing

ℓ	$\ell + 1 \ldots$
> 1	$\mathcal{O}[\![T]\!](\ell + 1)$

$$\mathcal{R}[\![T]\!](\ell)$$

$$\mathcal{R}[\![T]\!](\ell)$$

Reference confers permission to
decrement count & release shares

RC-DECR

$$\left\{\mathcal{R}[\![T]\!](\ell)\right\} \ \texttt{--}\ell\ \left\{n.\left(\ulcorner n > 0 \urcorner \wedge \mathsf{emp}\right)\right\}$$

References: Ownership + Sharing

ℓ	$\ell + 1 \ldots$
1	$\mathcal{O}[\![T]\!](\ell+1)$

$$\mathcal{R}[\![T]\!](\ell)$$

Reference confers permission to decrement count & release shares

RC-DECR

$$\left\{\mathcal{R}[\![T]\!](\ell)\right\} \;--\ell\; \left\{n.\left(\ulcorner n > 0 \urcorner \wedge \mathsf{emp}\right)\right\}$$

References: Ownership + Sharing

ℓ	ℓ + 1 ...
0	$\mathcal{O}\,[\![T]\!](\ell + 1)$

Reference confers permission to
decrement count & release shares

RC-DECR

$$\left\{\mathcal{R}[\![T]\!](\ell)\right\} \text{--}\text{--}\ell \left\{n. \left(\ulcorner n > 0 \urcorner \wedge \mathsf{emp}\right) \vee \left(\ulcorner n = 0 \urcorner \star \ell \mapsto 0 \star \mathcal{O}\,[\![T]\!](\ell + 1)\right)\right\}$$

References: Ownership + Sharing

ℓ	ℓ + 1 ...
0	$\mathcal{O}[\![T]\!](\ell + 1)$

References: Ownership + Sharing

ℓ	$\ell + 1$ ...
1	$\mathcal{O}[\![T]\!](\ell + 1)$

$$\left\{\ell \mapsto 1 \star \mathcal{O}[\![T]\!](\ell + 1)\right\}\ e\ \left\{Q\right\}$$

References: Ownership + Sharing

$$\mathcal{R}\,[\![T]\!](\ell)$$

ℓ	$\ell + 1 \ldots$
1	$\mathcal{O}\,[\![T]\!](\ell + 1)$

RC-NEW

$$\frac{\left\{\mathcal{R}[\![T]\!](\ell)\right\}\ e\ \left\{Q\right\}}{\left\{\ell \mapsto 1 \star \mathcal{O}[\![T]\!](\ell + 1)\right\}\ e\ \left\{Q\right\}}$$

Calling Conventions: Simple Functions

$$\mathcal{O}\,[\![T_1 \to T_2]\!](\ell) \triangleq \exists\, f.\ \ell \mapsto f \star$$

Pointer to function

Calling Conventions: Simple Functions

$$\mathcal{O}\,[\![T_1 \longrightarrow T_2]\!](\ell) \triangleq \exists\, f.\ \ell \mapsto f \star$$
$$\forall\, \ell_1.\ \left\{\mathcal{R}\,[\![T_1]\!](\ell_1)\right\}\ f(\ell_1)\ \left\{\ell_2.\ \mathcal{R}\,[\![T_2]\!](\ell_2)\right\}$$

Pointer to function

Calling convention:
Caller increment

Calling Conventions: Simple Functions

$$\mathcal{O}\,[\![T_1 \to T_2]\!](\ell) \triangleq \exists f.\ \ell \mapsto f \star \\ \forall \ell_1.\ \Big\{\mathcal{R}\,[\![T_1]\!](\ell_1)\Big\}\ f(\ell_1)\ \Big\{\ell_2.\ \mathcal{R}\,[\![T_2]\!](\ell_2)\Big\}$$

Pointer to function

Calling convention:
Caller increment

vs.

$$\forall \ell_1.\ \Big\{\mathcal{R}\,[\![T_1]\!](\ell_1)\Big\}\ f(\ell_1)\ \Big\{\ell_2.\ \mathcal{R}\,[\![T_2]\!](\ell_2) \star \underbrace{\mathcal{R}\,[\![T_1]\!](\ell_1)}_{\text{Callee increment}}\Big\}$$

Calling Conventions: Simple Functions

$$\mathcal{O}\,[\![T_1 \to T_2]\!](\ell) \triangleq \exists\, f.\; \ell \mapsto f \star \forall\, \ell_1.\; \left\{\mathcal{R}\,[\![T_1]\!](\ell_1)\right\}\; f(\ell_1)\; \left\{\ell_2.\; \mathcal{R}\,[\![T_2]\!](\ell_2)\right\}$$

Pointer to function

Calling convention: Caller increment

vs.

$$\forall\, \ell_1.\; \left\{\mathcal{R}\,[\![T_1]\!](\ell_1)\right\}\; f(\ell_1)\; \left\{\ell_2.\; \mathcal{R}\,[\![T_2]\!](\ell_2) \star \underbrace{\mathcal{R}\,[\![T_1]\!](\ell_1)}_{\text{Callee increment}}\right\}$$

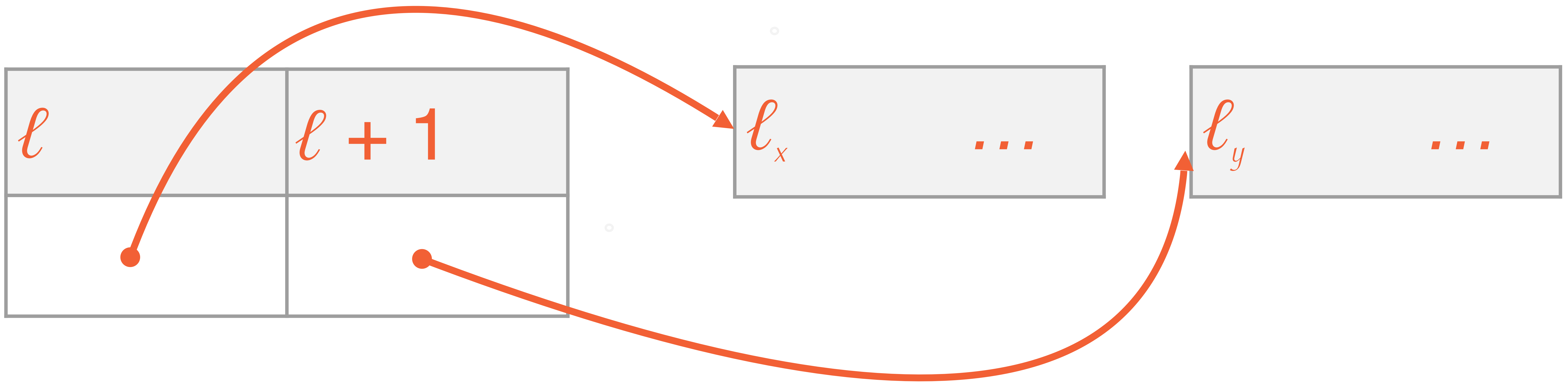
Later:
Recursive closures

Aggregate Layout

$$\mathcal{O}\,[\![\mathsf{struct\ Point}\ \{x : \mathbb{Z}, y : \mathbb{Z}\}]\!](\boldsymbol{\ell})$$

Aggregate Layout

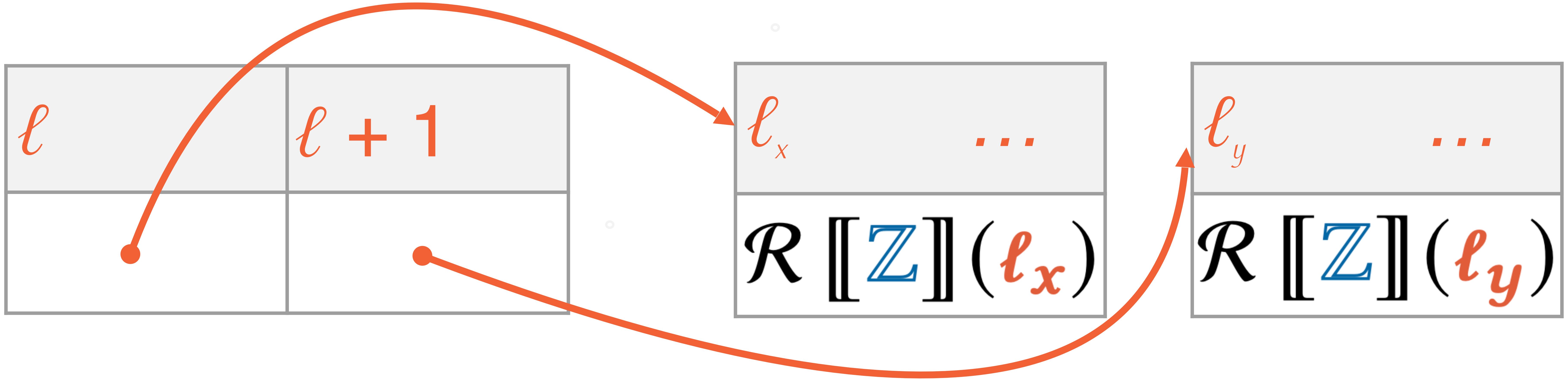
$$\mathcal{O}\,[\![\text{struct Point}\,\{x : \mathbb{Z}, y : \mathbb{Z}\}]\!](\ell)$$



$$\exists\, \ell_x, \ell_y.\, \underbrace{\ell \mapsto \ell_x \star \ell + 1 \mapsto \ell_y}_{\text{Physical footprint}}$$

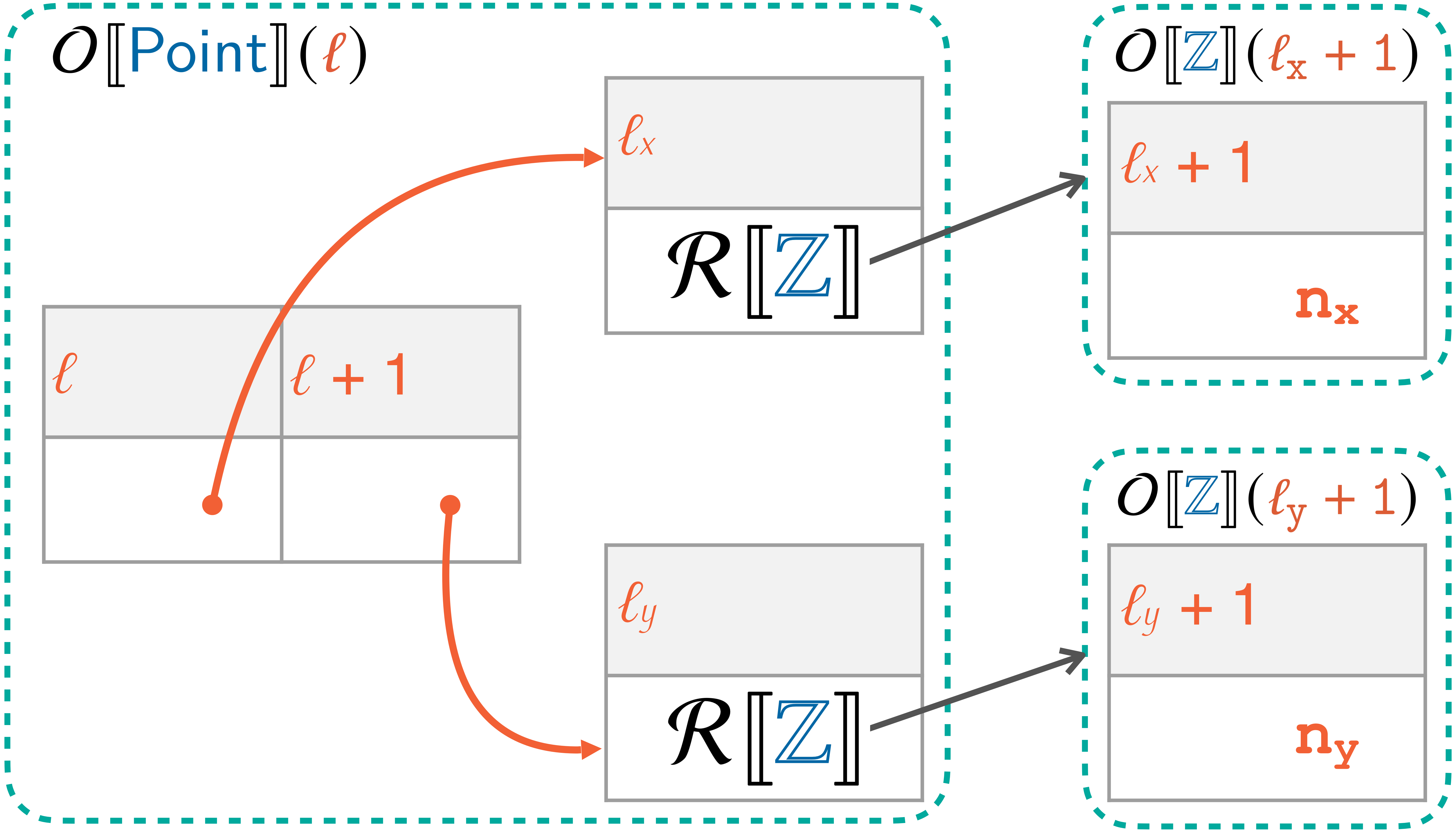
Aggregate Layout

$$\mathcal{O}[\![\text{struct Point}\ \{x : \mathbb{Z}, y : \mathbb{Z}\}]\!](\ell)$$

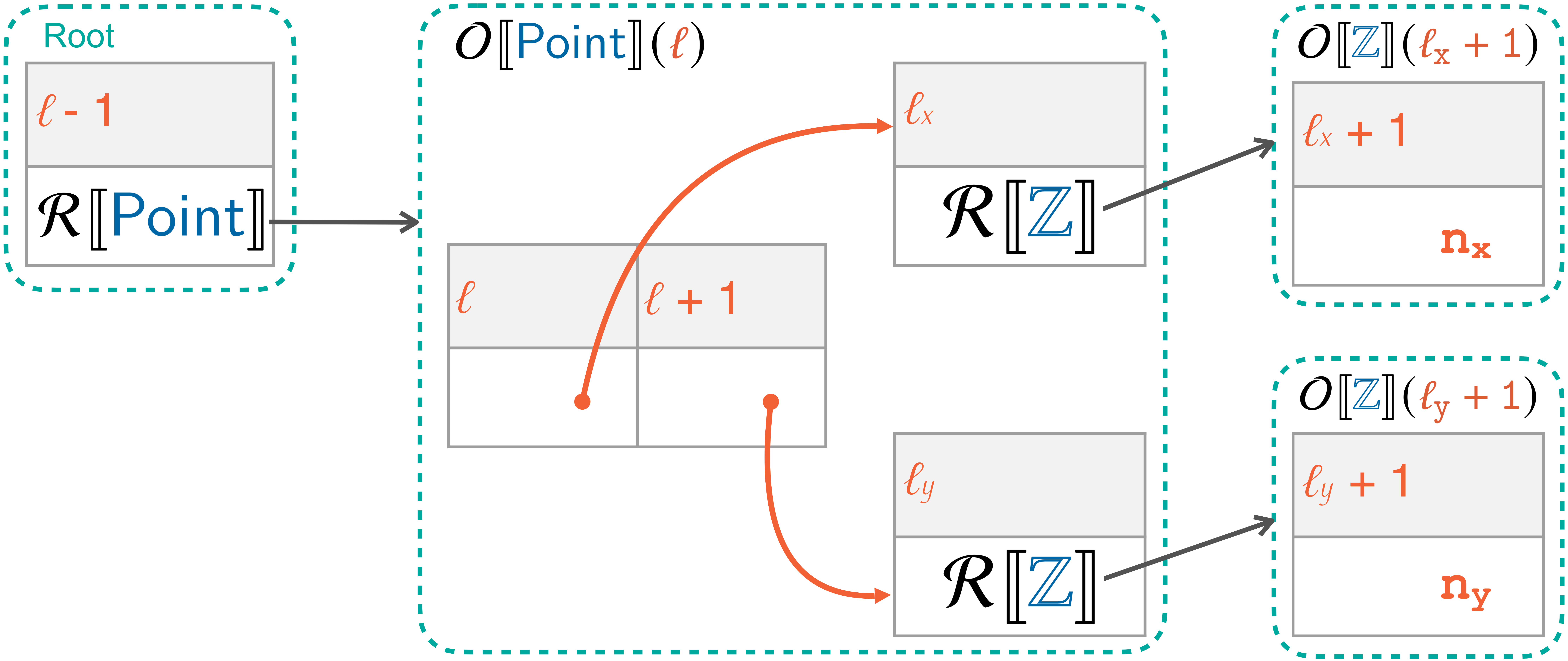


$$\boxed{\exists\, \ell_x, \ell_y.\ \underbrace{\ell \mapsto \ell_x \star \ell + 1 \mapsto \ell_y}_{\text{Physical footprint}} \star \underbrace{\mathcal{R}[\![\mathbb{Z}]\!](\ell_x) \star \mathcal{R}[\![\mathbb{Z}]\!](\ell_y)}_{\text{Logical footprint includes permission to access fields}}}$$

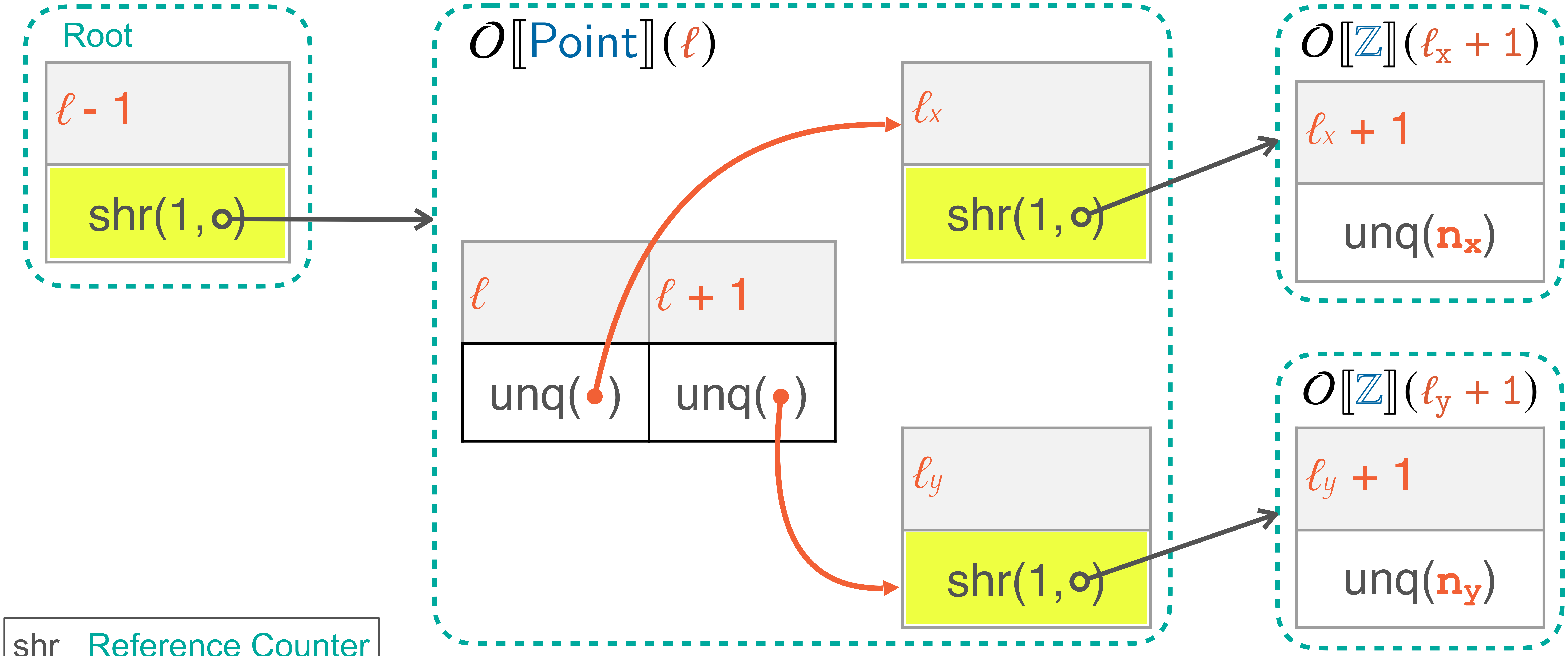
Resource Graphs



Resource Graphs

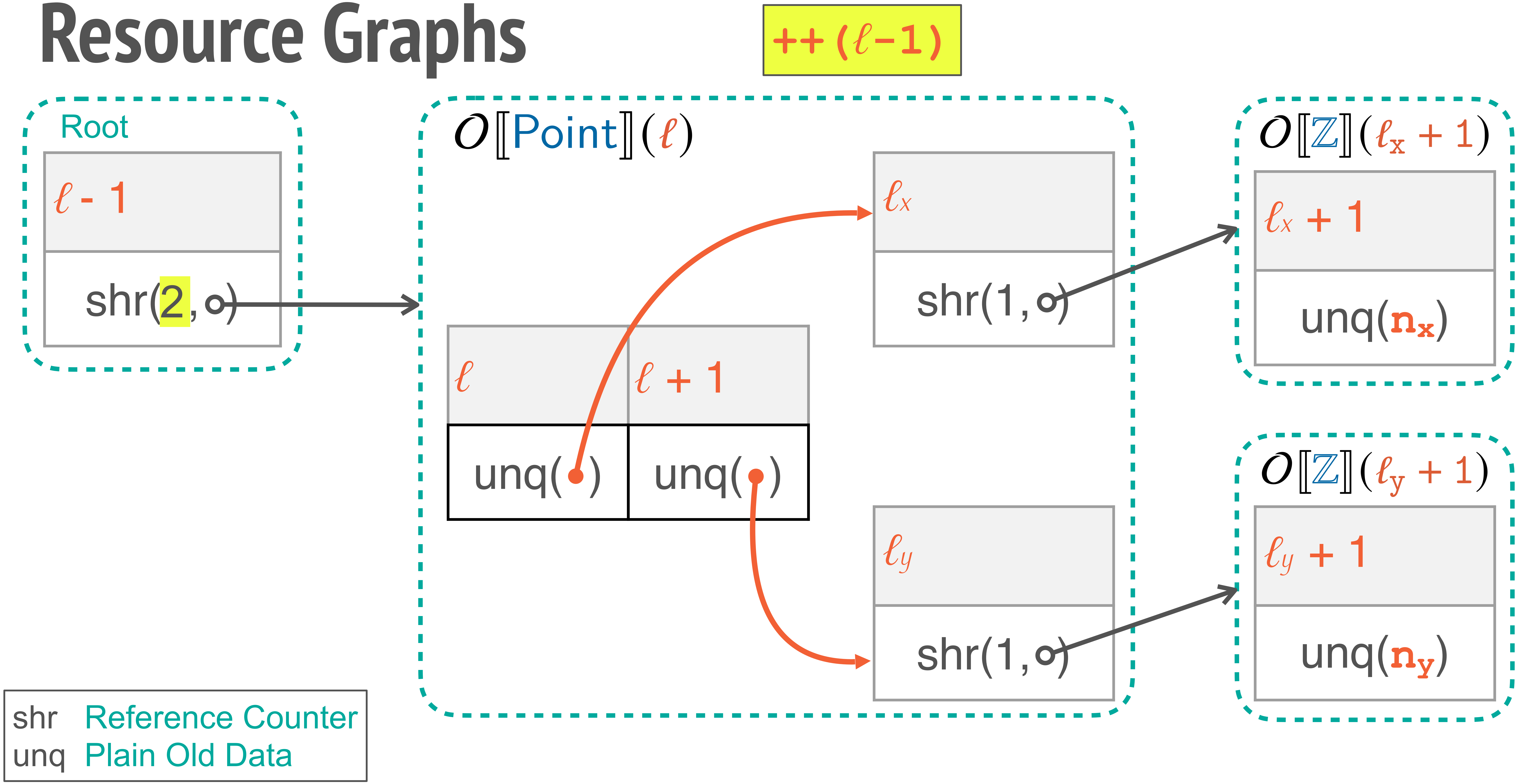


Resource Graphs

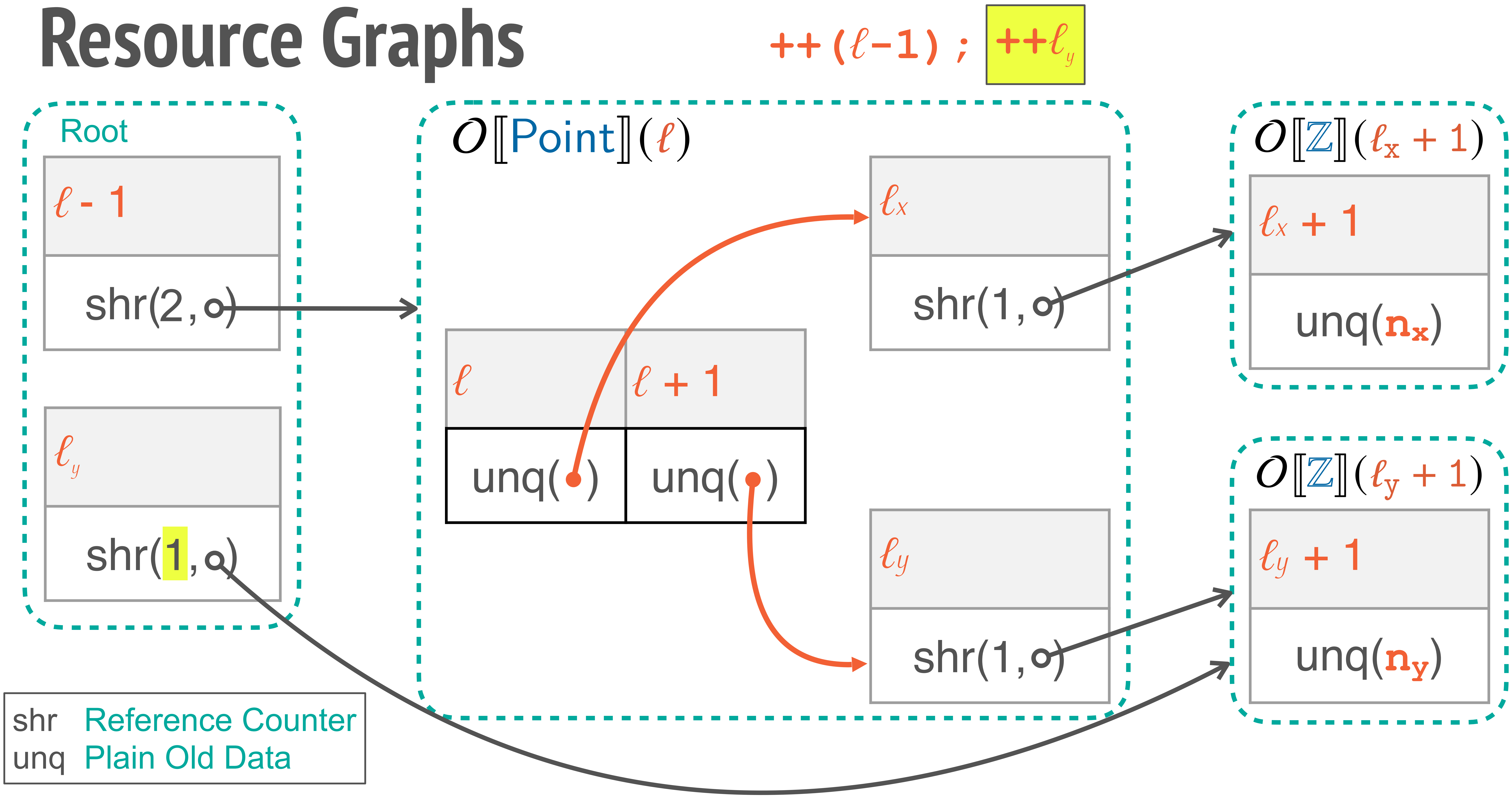


shr	Reference Counter
unq	Plain Old Data

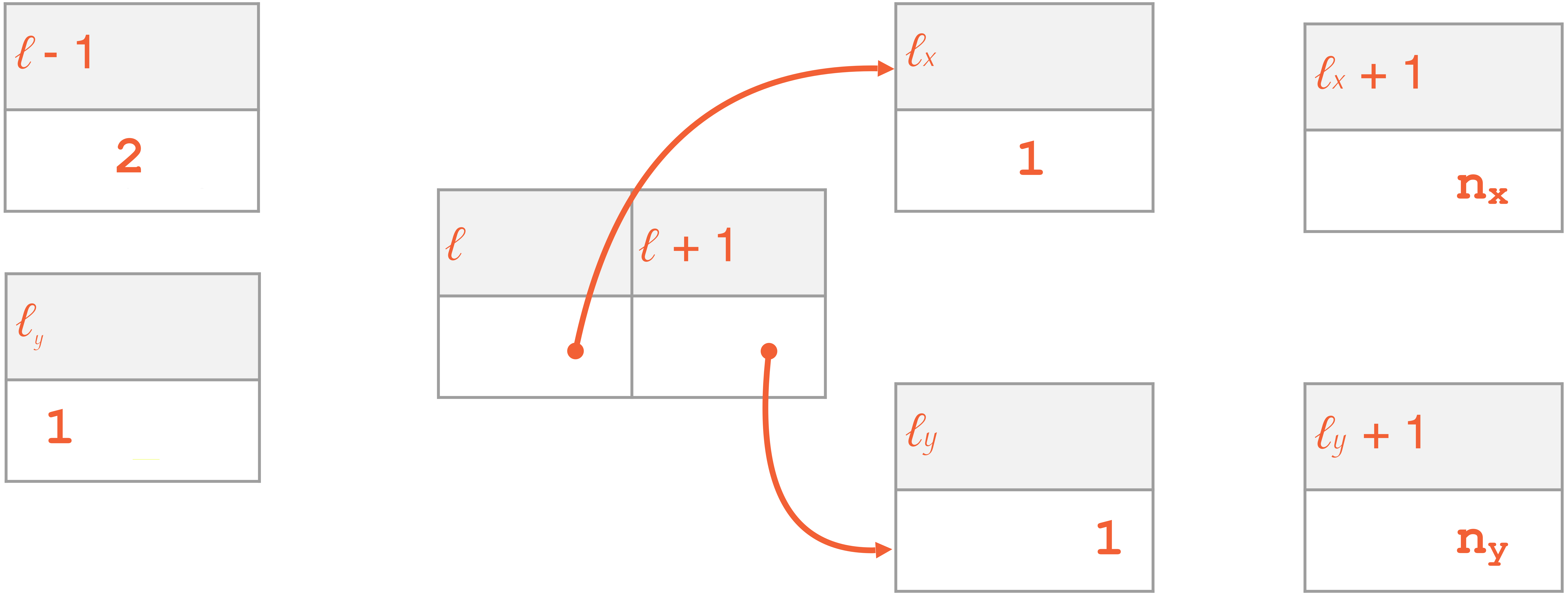
Resource Graphs



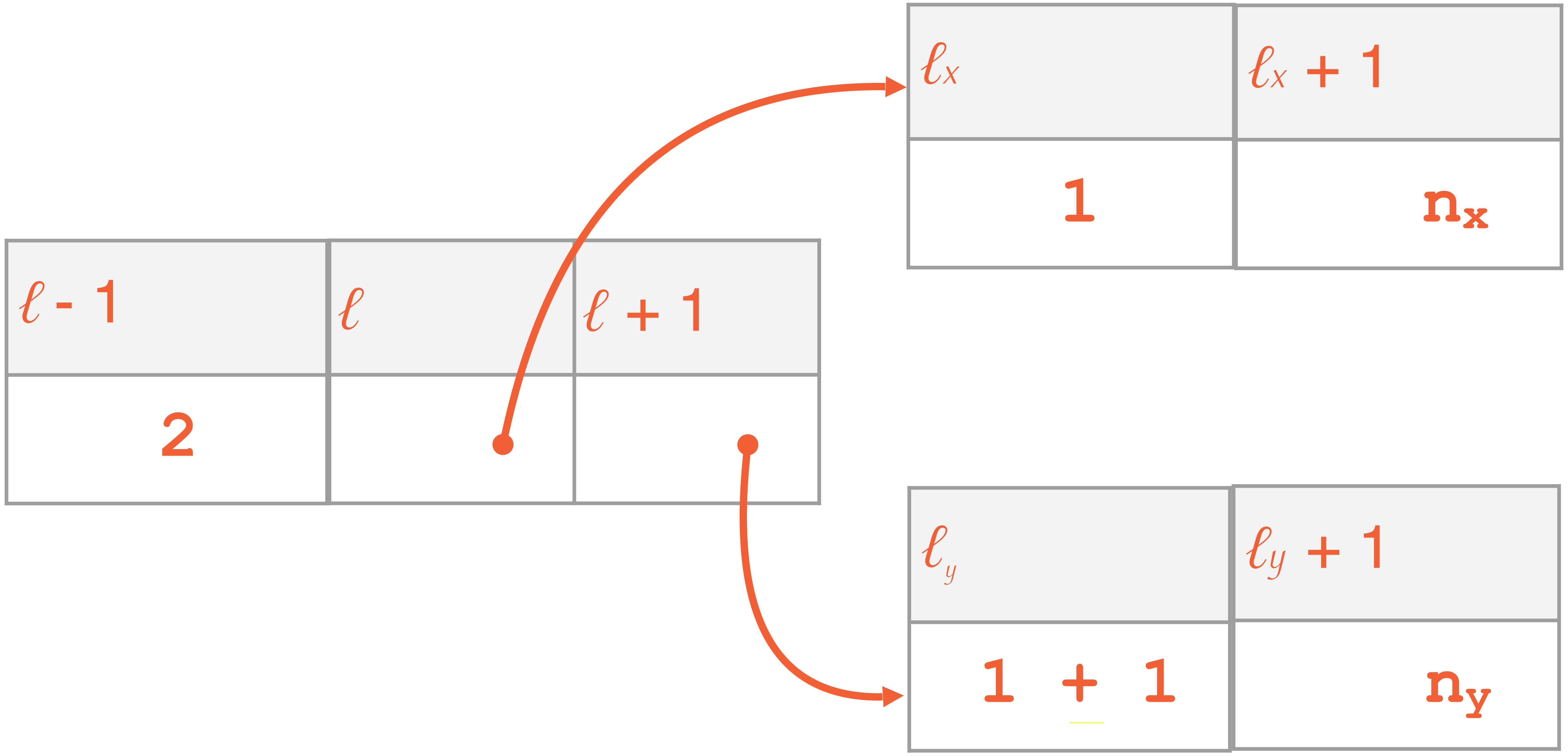
Resource Graphs



Resource Graphs



Resource Graphs



Resources

$$\rho \in \textsc{Res} \triangleq \textsc{Loc} \rightharpoonup \mathsf{own}(\textsc{Val}) + \mathsf{shr}(\mathbb{N}_{>0} \times \textsc{Res})$$

Resources

$$\rho \in \textsc{Res} \triangleq \textsc{Loc} \rightharpoonup \mathsf{own}(\textsc{Val}) + \mathsf{shr}(\mathbb{N}_{>0} \times \textsc{Res})$$

$$\rho_1 \bullet \rho_2 = \rho_1 \uplus \rho_2 \quad (\text{if } \mathrm{dom}(\rho_1) \cap \mathrm{dom}(\rho_2) = \varnothing)$$

Resources

$$\rho \in \textsc{Res} \triangleq \textsc{Loc} \rightharpoonup \mathsf{own}(\textsc{Val}) + \mathsf{shr}(\mathbb{N}_{>0} \times \textsc{Res})$$

$$\rho_1 \bullet \rho_2 = \rho_1 \uplus \rho_2 \qquad (\text{if } \mathrm{dom}(\rho_1) \cap \mathrm{dom}(\rho_2) = \emptyset)$$

$$\ell \mapsto \mathsf{shr}(n_1, \rho) \bullet \ell \mapsto \mathsf{shr}(n_2, \rho) = \ell \mapsto \mathsf{shr}(\overbrace{n_1 + n_2}^{\text{Siblings split counter}}, \rho)$$

Agree on resource

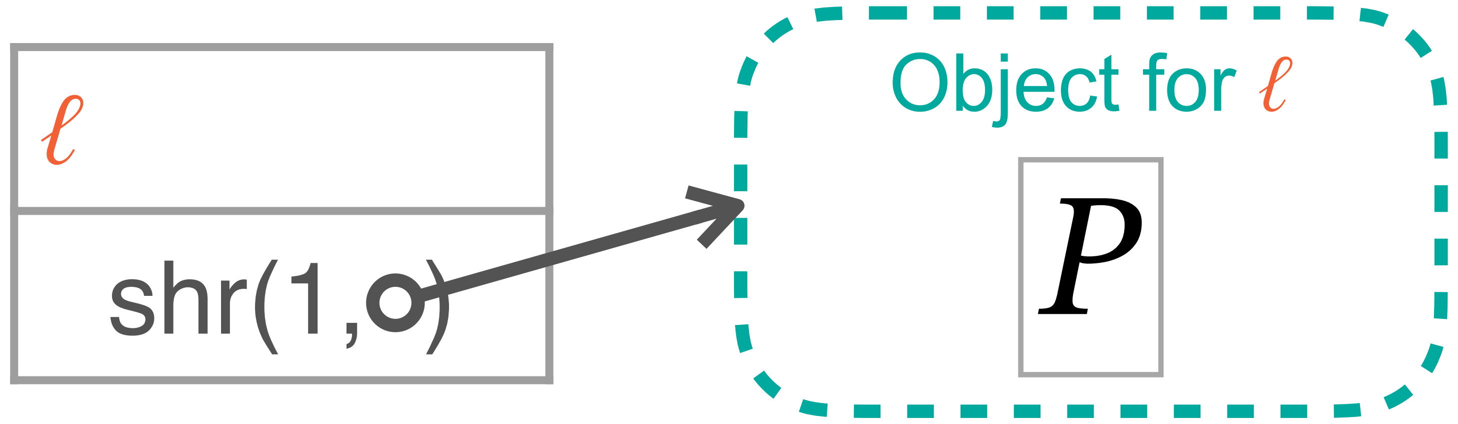
But share resource

Modalities

Modalities

Jump Modality: It is possible to "jump" from ℓ to an object that satisfies P

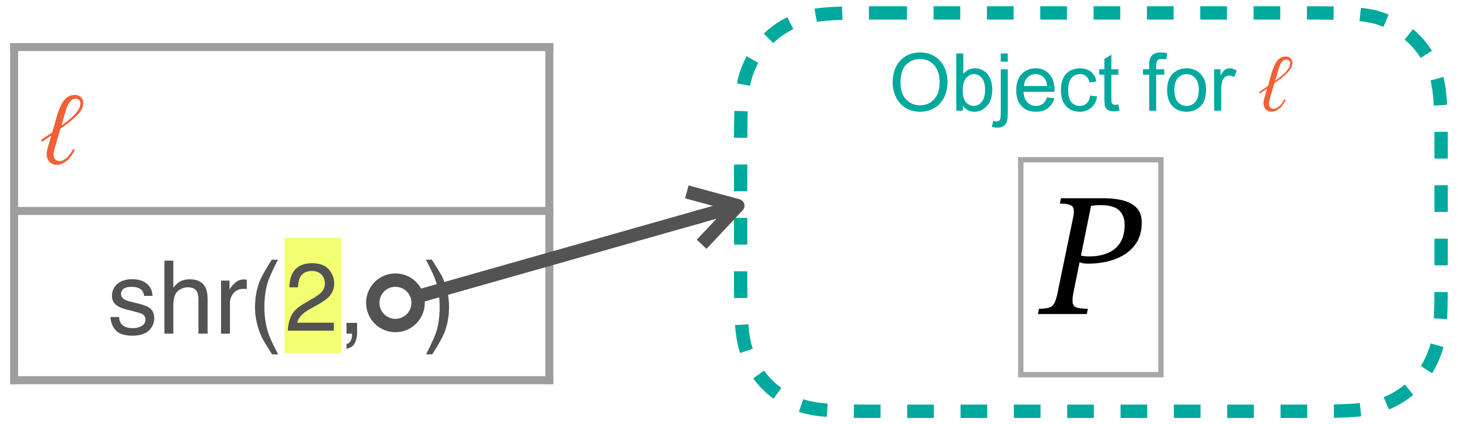
$$@_{\ell}\, P \;\approx$$



Modalities

Jump Modality: It is possible to "jump" from ℓ to an object that satisfies P

$$@_\ell P \star @_\ell P \approx$$

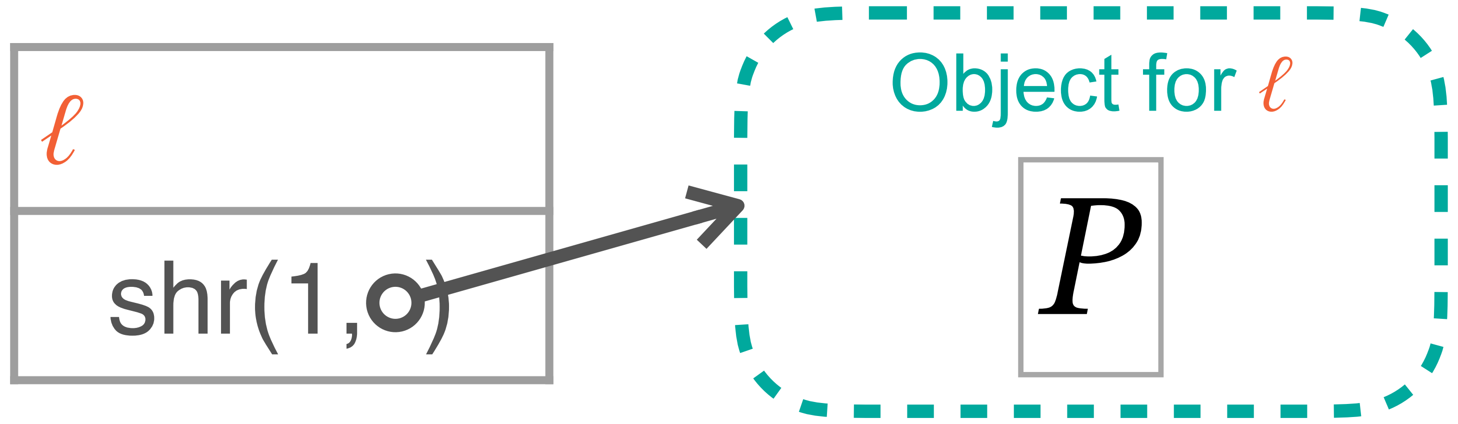


Composition sums counters at the root, not in objects

Modalities

Jump Modality: It is possible to “jump” from ℓ to an object that satisfies P

$$@_{\ell}\, P \;\approx$$

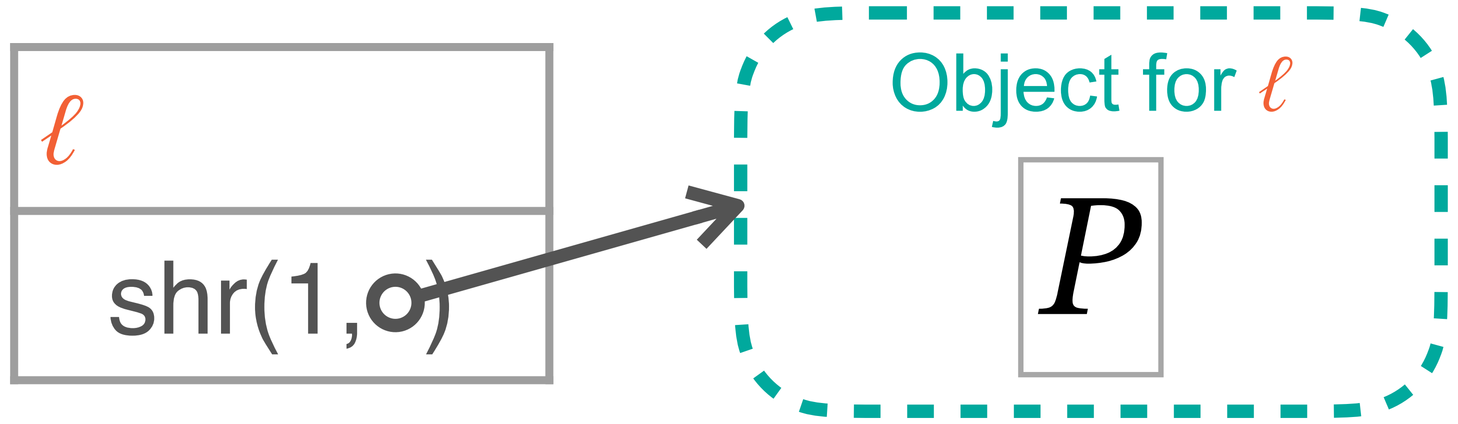


$$\boxed{\mathcal{R}\,[\![T]\!](\ell) \triangleq @_{\ell}\, \mathcal{O}\,[\![T]\!](\ell + 1)}$$

Modalities

Jump Modality: It is possible to "jump" from ℓ to an object that satisfies P

$$@_{\ell}\, P \;\approx$$



$$\boxed{\mathcal{R}\,[\![T]\!](\ell) \triangleq @_{\ell}\, \mathcal{O}\,[\![T]\!](\ell + 1)}$$

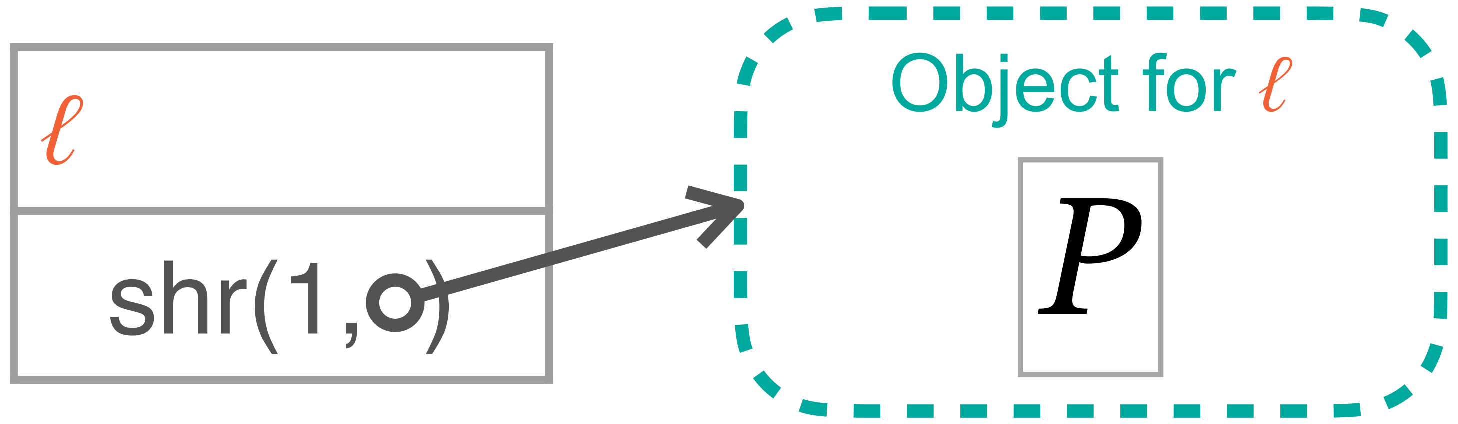
Reachability Modality $\diamond P$: It is possible to reach P via some sequence of jumps

Allows reading and incrementing from deeply nested objects

Modalities

Jump Modality: It is possible to "jump" from ℓ to an object that satisfies P

$$@_{\ell}\, P \;\approx$$



$$\boxed{\mathcal{R}\,[\![T]\!](\ell) \triangleq @_{\ell}\, \mathcal{O}\,[\![T]\!](\ell + 1)}$$

Reachability Modality $\diamond\, P$ **:** It is possible to reach P via some sequence of jumps

Allows reading and incrementing from deeply nested objects

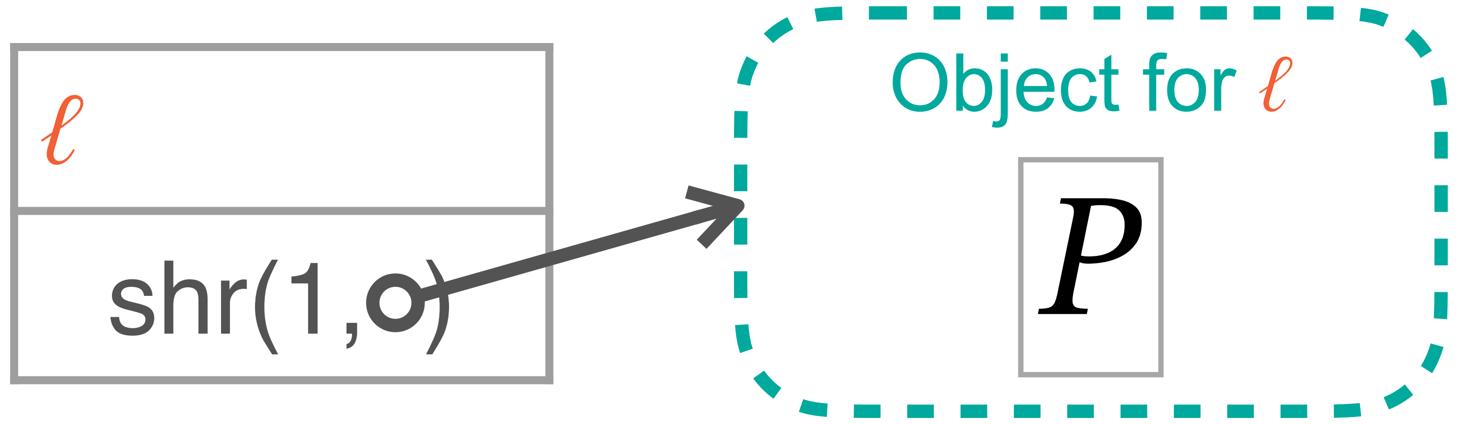
@-INCR

$$\frac{}{\{\, @_{\ell}\, P \,\}\ {+}{+}\ell\ \{\, n.\ \ulcorner n > 1 \urcorner \star @_{\ell}\, P \star @_{\ell}\, P \,\}}$$

Modalities

Jump Modality: It is possible to “jump” from ℓ to an object that satisfies P

$$@_{\ell}\, P \;\approx$$



$$\boxed{\mathcal{R}[\![T]\!](\ell) \triangleq @_{\ell}\, \mathcal{O}[\![T]\!](\ell + 1)}$$

Reachability Modality $\diamond P$ **:** It is possible to reach P via some sequence of jumps

Allows reading and incrementing from deeply nested objects

$$\textsc{@-Incr-}\diamond \quad \frac{Q \vdash \diamond @_{\ell}\, P}{\{\ Q\ \}\ {+}{+}\ell\ \{ n.\ \ulcorner n > 1 \urcorner \star\ Q\ \star @_{\ell}\, P \}}$$

Calling Conventions: Recursive Closures

$$\mathcal{O}[\![T_1 \rightarrow T_2]\!](\ell) \triangleq \exists\, \textbf{call}$$

$$\ell \mapsto \textbf{call}$$

Calling Conventions: Recursive Closures

$\mathcal{O}\,[\![T_1 \to T_2]\!](\ell) \triangleq \exists\, \textbf{call}, \textbf{destroy}, Q_{\mathsf{env}}.$ Environment

$$\ell \mapsto \textbf{call} \star \ell + 1 \mapsto \textbf{destroy} \star Q_{\mathsf{env}}$$

Calling Conventions: Recursive Closures

$$\mathcal{O}\,[\![T_1 \to T_2]\!](\ell) \triangleq \exists\, \mathbf{call}, \mathbf{destroy}, Q_{\mathsf{env}}.$$

Environment

$$\ell \mapsto \mathbf{call} \star \ell + 1 \mapsto \mathbf{destroy} \star Q_{\mathsf{env}}$$

$$\star \quad \forall\, \ell_1.\ \left\{@_{\ell-1}\,(\ell \mapsto \mathbf{call} \star \ell + 1 \mapsto \mathbf{destroy} \star Q_{\mathsf{env}}) \star \mathcal{R}\,[\![T_1]\!](\ell_1)\right\}$$

"self" argument

$$\mathbf{call}(\ell - 1, \ell_1)$$

$$\left\{\ell_2.\ \mathcal{R}\,[\![T_2]\!](\ell_2)\right\}$$

Calling Conventions: Recursive Closures

$$\mathcal{O}[\![T_1 \to T_2]\!](\ell) \triangleq \exists\, \mathbf{call}, \mathbf{destroy}, Q_{\mathsf{env}}.$$

Environment

$$\ell \mapsto \mathbf{call} \star \ell + 1 \mapsto \mathbf{destroy} \star Q_{\mathsf{env}}$$

$$\star \quad \forall\, \ell_1.\ \left\{@_{\ell-1}\,(\ell \mapsto \mathbf{call} \star \ell + 1 \mapsto \mathbf{destroy} \star Q_{\mathsf{env}}) \star \mathcal{R}[\![T_1]\!](\ell_1)\right\}$$

"self" argument

$$\mathbf{call}(\ell - 1, \ell_1)$$

$$\left\{\ell_2.\ \mathcal{R}[\![T_2]\!](\ell_2)\right\}$$

$$\star \quad \left\{\underbrace{\ell - 1 \mapsto 0 \star \ell \mapsto \mathbf{call} \star \ell + 1 \mapsto \mathbf{destroy} \star Q_{\mathsf{env}}}_{\text{After ref. counter hits zero}}\right\}\ \mathbf{destroy}(\ell - 1)\ \left\{\mathsf{emp}\right\}$$

Rigid Layout

$$\mathcal{O}\,[\![\mathsf{struct\ Point}\ \{x : \mathbb{Z}, y : \mathbb{Z}\}]\!](\ell)$$
$$= \exists\, \ell_x, \ell_y.\ \ell \mapsto \ell_x \star \ell + 1 \mapsto \ell_y \star \mathcal{R}\,[\![\mathbb{Z}]\!](\ell_x) \star \mathcal{R}\,[\![\mathbb{Z}]\!](\ell_y)$$

Like C ABI

Rigid Layout

$$\begin{aligned}&\mathcal{O}\,[\![\textsf{struct Point}\ \{x:\mathbb{Z}, y:\mathbb{Z}\}]\!](\boldsymbol{\ell})\\ &\quad= \exists\, \boldsymbol{\ell_x}, \boldsymbol{\ell_y}.\ \boldsymbol{\ell} \mapsto \boldsymbol{\ell_x} \star \boldsymbol{\ell} + \mathbf{1} \mapsto \boldsymbol{\ell_y} \star \mathcal{R}\,[\![\mathbb{Z}]\!](\boldsymbol{\ell_x}) \star \mathcal{R}\,[\![\mathbb{Z}]\!](\boldsymbol{\ell_y})\end{aligned}$$

Like C ABI

No reordering

$$\textsf{struct Point}\ \{x:\mathbb{Z}, y:\mathbb{Z}\} \overset{\textsf{upd}}{\not\Rightarrow} \textsf{struct Point}\ \{y:\mathbb{Z}, x:\mathbb{Z}\}$$

Rigid Layout

$$\mathcal{O}\,[\![\mathsf{struct\ Point}\ \{x : \mathbb{Z}, y : \mathbb{Z}\}]\!](\ell)$$
$$= \exists\, \ell_x, \ell_y.\ \ell \mapsto \ell_x \star \ell + 1 \mapsto \ell_y \star \mathcal{R}\,[\![\mathbb{Z}]\!](\ell_x) \star \mathcal{R}\,[\![\mathbb{Z}]\!](\ell_y)$$

Like C ABI

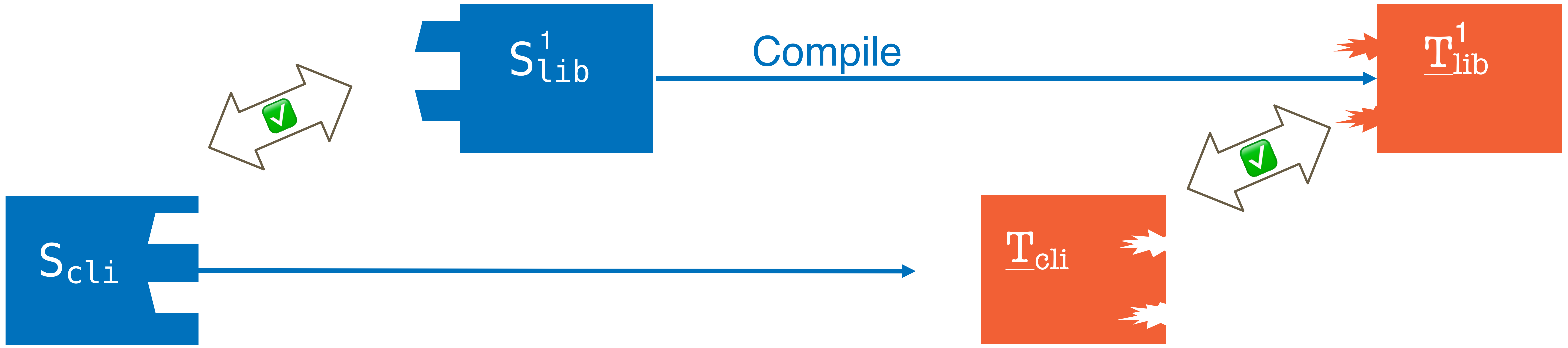
No reordering

$$\mathsf{struct\ Point}\ \{x : \mathbb{Z}, y : \mathbb{Z}\} \overset{\mathsf{upd}}{\not\Rightarrow} \mathsf{struct\ Point}\ \{y : \mathbb{Z}, x : \mathbb{Z}\}$$

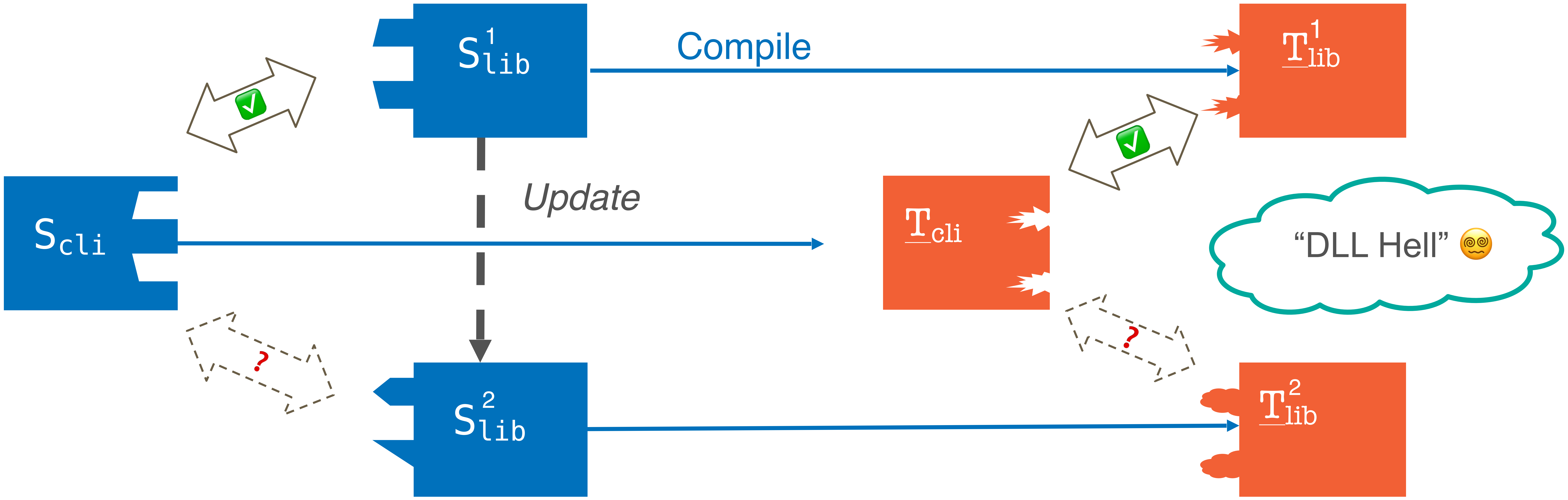
No extensibility

$$\mathsf{struct\ Point}\ \{x : \mathbb{Z}, y : \mathbb{Z}\} \overset{\mathsf{upd}}{\not\Rightarrow} \mathsf{struct\ Point}\ \{x : \mathbb{Z}, y : \mathbb{Z}, z : \mathbb{Z}\}$$

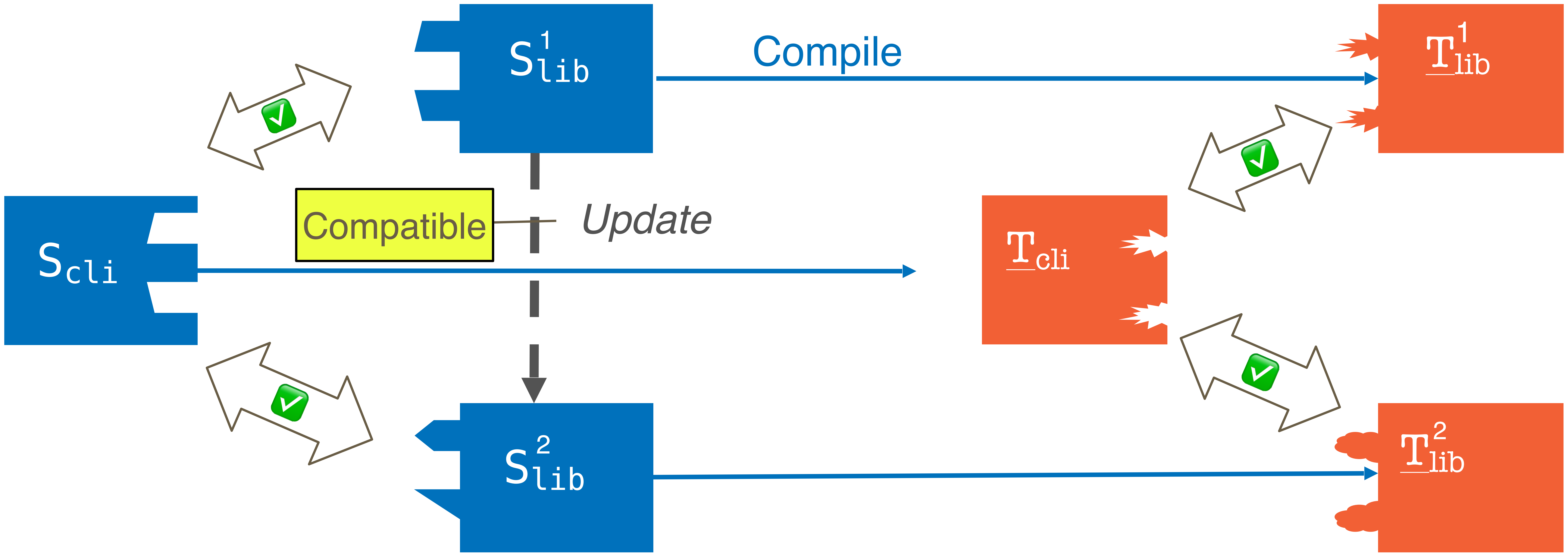
Library Evolution



Library Evolution



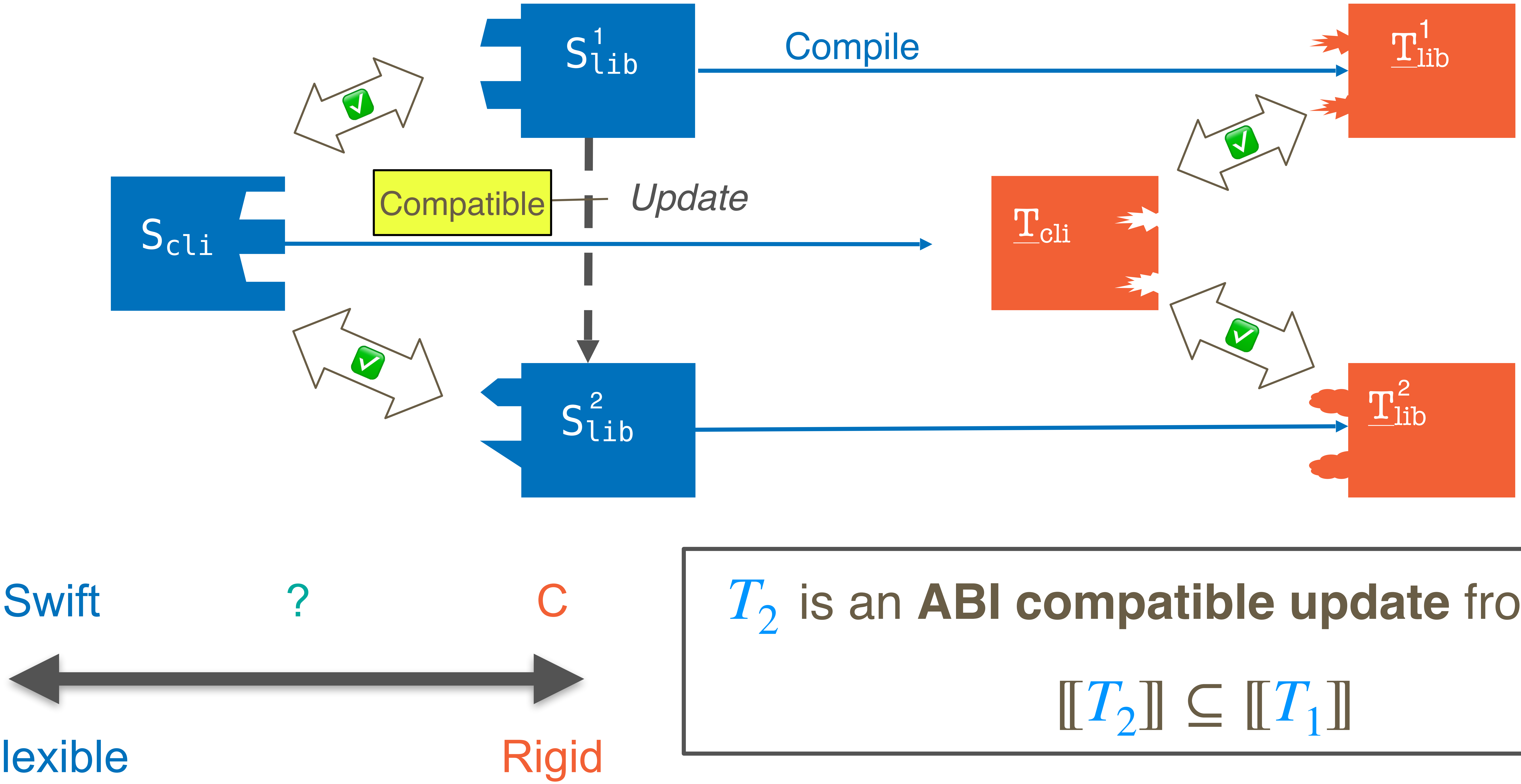
Library Evolution



T_2 is an **ABI compatible update** from T_1 if

$$[\![T_2]\!] \subseteq [\![T_1]\!]$$

Library Evolution



T_2 is an **ABI compatible update** from T_1 if

$$[\![T_2]\!] \subseteq [\![T_1]\!]$$

Resilient Layout

Like Swift ABI

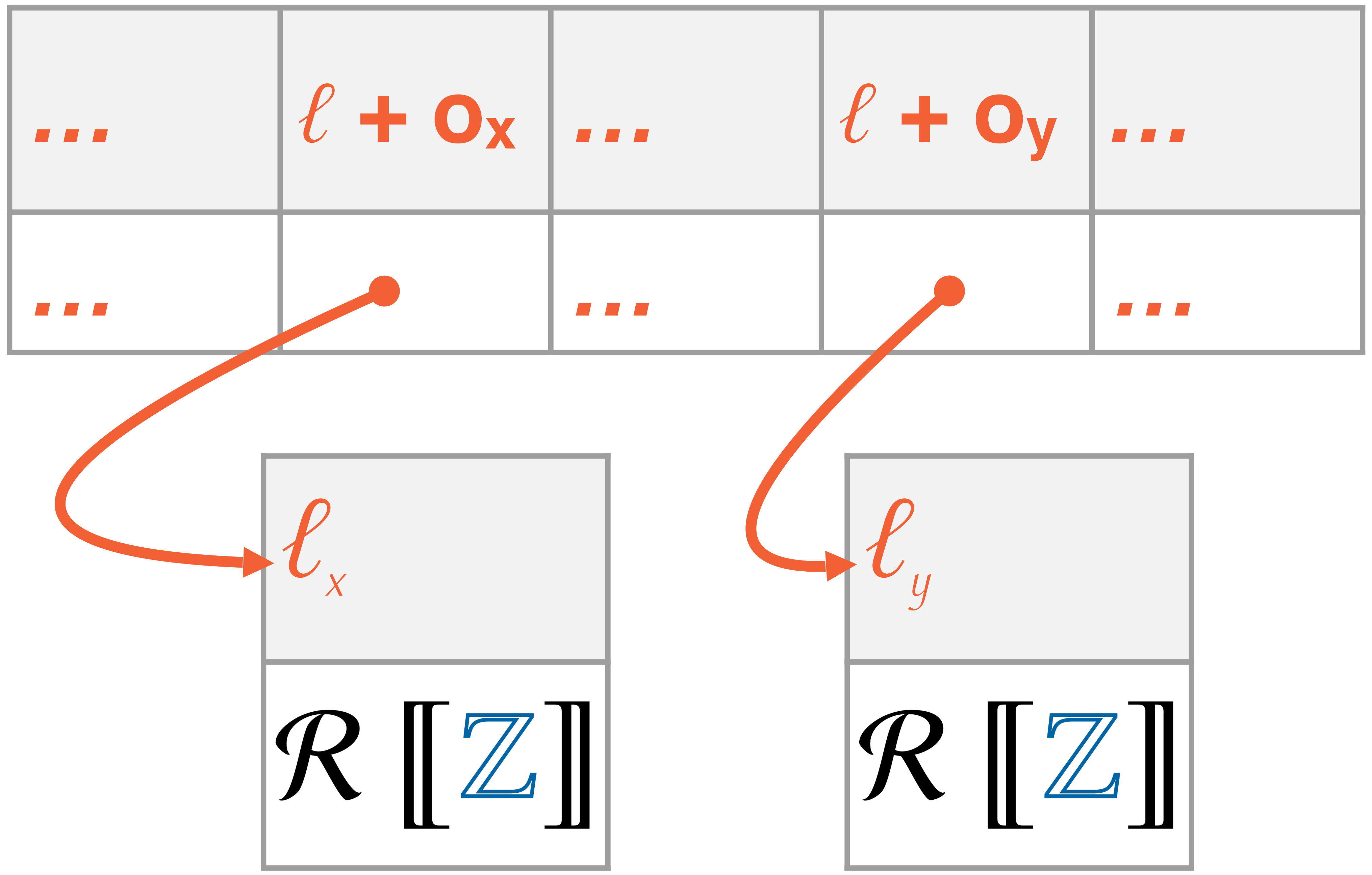
Resilient Layout

Like Swift ABI

Client Using `Point`

…	o_x	…	o_y	…
	?		?	

Offset Table



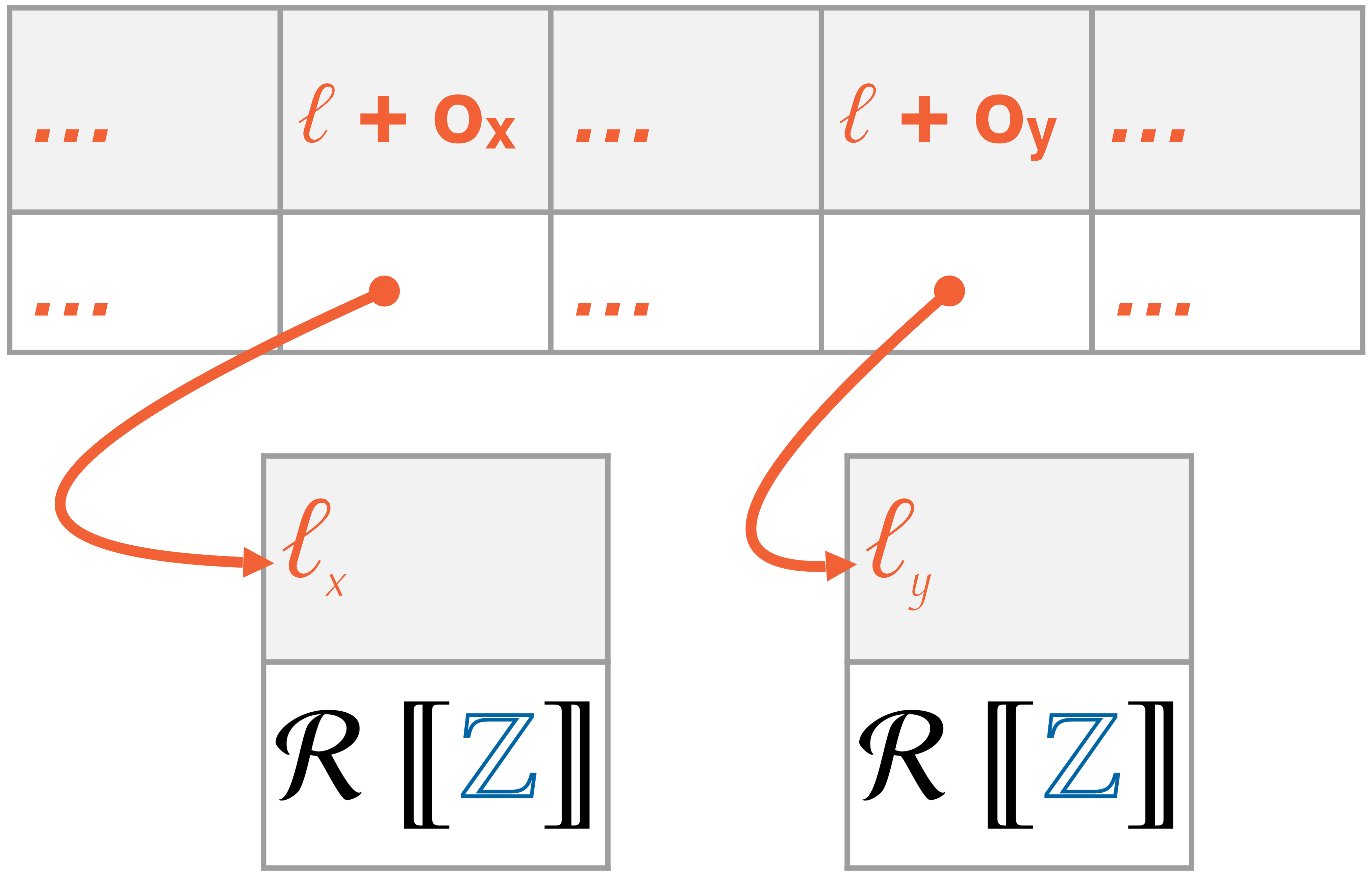
Resilient Layout

Like Swift ABI

Client Using `Point`

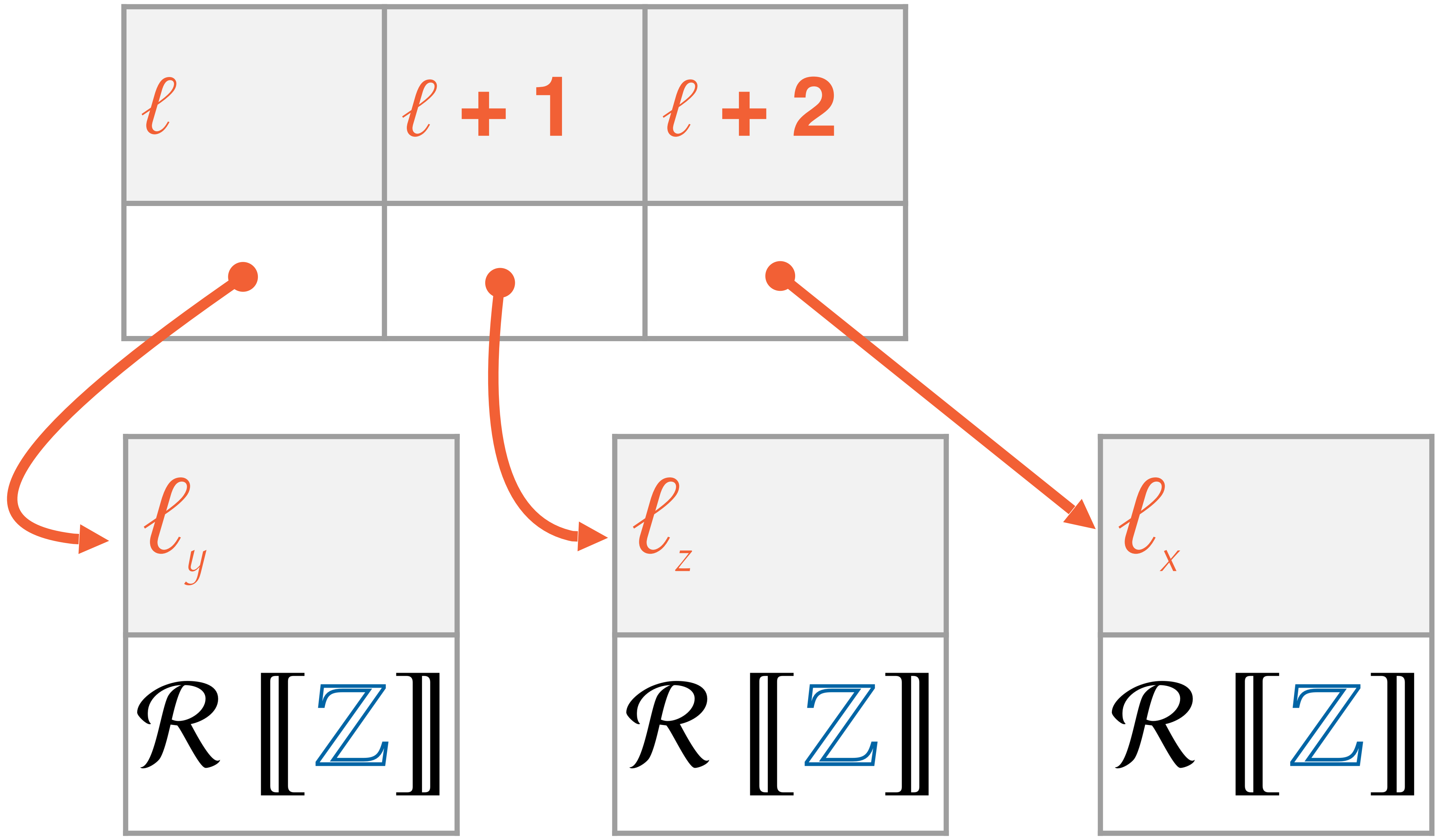
…	O_x	…	O_y	…
	?		?	

Offset Table



Library Providing `Point`

O_x	O_y	O_z
2	0	1



Interlude: Type Polymorphism

$$
\begin{array}{lcl}
\mathcal{V}[\![\alpha]\!]_\delta(v) & \triangleq & \delta(\alpha)(v) \\
\mathcal{V}[\![\forall\, \alpha.\, T]\!]_\delta(v) & \triangleq & \forall\, P \in \mathsf{SemTy}.\, \mathcal{E}[\![T]\!]_{\delta[\alpha \mapsto P]}(v[\,]) \\
 & & \\
\mathcal{D}[\![\Delta]\!](\delta) & \triangleq & \delta \in \mathrm{dom}(\Delta) \to \mathsf{SemTy}
\end{array}
$$

Resilient Layout: Signature Predicate

Resilient Layout: Signature Predicate

$$\mathcal{O}[\![\mathsf{Point}]\!]_{\varsigma}(\ell) \stackrel{\triangle}{\approx} \varsigma(\mathsf{Point})[\ell]$$

Resilient Layout: Signature Predicate

$$\mathcal{O}[\![\mathsf{Point}]\!]_{\varsigma}(\ell) \stackrel{\triangle}{\approx} \varsigma(\mathsf{Point})[\ell]$$

$\mathcal{S}[\![\Sigma]\!](\varsigma)$ ensures that

Resilient Layout: Signature Predicate

$$\mathcal{O}[\![\mathsf{Point}]\!]_{\varsigma}(\ell) \stackrel{\triangle}{\approx} \varsigma(\mathsf{Point})[\ell]$$

$\mathcal{S}[\![\Sigma]\!](\varsigma)$ ensures that

1. For every *known* field, an **accessor** is defined that is consistent with $\varsigma(\mathsf{Point})[\ell]$

Resilient Layout: Signature Predicate

$$\mathcal{O}[\![\mathsf{Point}]\!]_{\varsigma}(\ell) \stackrel{\triangle}{\approx} \varsigma(\mathsf{Point})[\ell]$$

$\mathcal{S}[\![\Sigma]\!](\varsigma)$ ensures that

1. For every *known* field, an **accessor** is defined that is consistent with $\varsigma(\mathsf{Point})[\ell]$
2. A **destructor** is defined that can clean up a $\varsigma(\mathsf{Point})[\ell]$

Resilient Layout: Signature Predicate

$$\mathcal{O}[\![\mathsf{Point}]\!]_{\varsigma}(\ell) \stackrel{\triangle}{\approx} \varsigma(\mathsf{Point})[\ell]$$

$\mathcal{S}[\![\Sigma]\!](\varsigma)$ ensures that

1. For every *known* field, an **accessor** is defined that is consistent with $\varsigma(\mathsf{Point})[\ell]$
2. A **destructor** is defined that can clean up a $\varsigma(\mathsf{Point})[\ell]$
3. If Point is rigid, then the accessors exactly match the **declaration order** of the fields

Evolvable Types

$\texttt{struct}$ or $\texttt{enum}$

$$\mathsf{Mode} \ni \mathsf{m} ::= \mathsf{flex} \mid \mathsf{rigid} \qquad\qquad \mathsf{Sig} \ni \Sigma ::= \varnothing \mid \Sigma, \mathsf{m}\,\mathsf{k}\,\mathsf{X}\,\{\overline{\mathsf{s} : \mathsf{T}}\}$$

Evolvable Types

Like `non_exhaustive` in Rust

Like `@frozen` in Swift

`struct` or `enum`

$$\mathsf{Mode} \ni \mathsf{m} ::= \mathsf{flex} \mid \mathsf{rigid} \qquad \mathsf{Sig} \ni \Sigma ::= \varnothing \mid \Sigma, \mathsf{m}\,\mathsf{k}\,\mathsf{X}\,\{\overline{\mathsf{s} : \mathsf{T}}\}$$

Evolvable Types

Like `non_exhaustive` in Rust — Like `@frozen` in Swift — `struct` or `enum`

$$\mathsf{Mode} \ni m ::= \mathsf{flex} \mid \mathsf{rigid} \qquad \mathsf{Sig} \ni \Sigma ::= \varnothing \mid \Sigma, m\; k\; X\; \{\overline{s : T}\}$$

$$\boxed{(\text{rigid struct-I})}\quad \frac{\Sigma \ni \mathsf{rigid\ struct}\; X\; \{\overline{s : T}\} \qquad \overline{\Sigma; \Gamma \vdash e : T}}{\Sigma; \overline{\Gamma} \vdash \{\overline{s : e}\} : X}$$

$$\boxed{(m\ \text{struct-E})}\quad \frac{\Sigma \ni m\; \mathsf{struct}\; X\; \{\overline{s_i : T_i}^{\,i<n}\} \qquad \Sigma; \Gamma \vdash e : X \qquad j < n}{\Sigma; \Gamma \vdash e.s_j : T_j}$$

Evolvable Types

Like `non_exhaustive` in Rust

Like `@frozen` in Swift

`struct` or `enum`

$$\mathsf{Mode} \ni m ::= \mathsf{flex} \mid \mathsf{rigid} \qquad \mathsf{Sig} \ni \Sigma ::= \varnothing \mid \Sigma, m\ k\ X\ \{\overline{s : T}\}$$

$$\text{(rigid struct-I)}\quad \frac{\Sigma \ni \mathsf{rigid\ struct}\ X\ \{\overline{s : T}\} \qquad \overline{\Sigma;\Gamma \vdash e : T}}{\Sigma;\overline{\Gamma} \vdash \{\overline{s : e}\} : X}$$

$$\text{(m struct-E)}\quad \frac{\Sigma \ni m\ \mathsf{struct}\ X\ \{\overline{s_i : T_i}^{\,i<n}\} \qquad \Sigma;\Gamma \vdash e : X \qquad j < n}{\Sigma;\Gamma \vdash e.s_j : T_j}$$

$$\frac{\begin{array}{c}\Sigma \ni m\ \mathsf{enum}\ X\ \{\overline{s_i : T_i}^{\,i<n}\} \\ \Sigma;\Gamma \vdash e : T_j \qquad j < n\end{array}}{\Sigma;\Gamma \vdash s_j\ e : X}\quad \text{(m enum-I)}$$

$$\frac{\begin{array}{c}\Sigma \ni \mathsf{rigid\ enum}\ X\ \{\overline{s : T_1}\} \qquad \Sigma;\Gamma_1 \vdash e_1 : X \\ \overline{\Sigma;\Gamma_2, x : T_1 \vdash e_2 : T_2} \qquad \Gamma_2 \not\ni \overline{x}\end{array}}{\Sigma;\Gamma_1,\Gamma_2 \vdash \mathsf{case}\ e_1\ \{\overline{s\ x \Rightarrow e_2}\} : T_2}\quad \text{(rigid enum-E)}$$

Unboxed Data

$$\mathcal{V}[\![T]\!](\ell) \triangleq \exists\ (b, i) = \ell.\ \left\{ \begin{array}{l} \\ \\ \\ \\ \end{array} \right.$$

Always boxed

$(T = \mathsf{struct}\ -)$

Never boxed

$(T = \mathbb{Z})$

Sometimes boxed

$(T = - \rightarrow -)$

Unboxed Data

$$\mathcal{V}[\![T]\!](\ell) \triangleq \exists\, (b, i) = \ell.$$

$$\begin{cases} \ulcorner i = 1 \urcorner \star \mathcal{R}\,[\![T]\!](\ell - 1) \\ \\ \\ \\ \end{cases}$$

Always boxed

$(T = \mathsf{struct}\ -)$

Never boxed

$(T = \mathbb{Z})$

Sometimes boxed

$(T = - \to -)$

Unboxed Data

$$\mathcal{V}[\![T]\!](\ell) \triangleq \exists\ (b, i) = \ell.\ \begin{cases} \ulcorner i = 1 \urcorner \star \mathcal{R}[\![T]\!](\ell - 1) \\ \\ \ulcorner i = 0 \urcorner \star \mathcal{U}[\![T]\!](b) \\ \\ \end{cases}$$

Always boxed

$(T = \mathsf{struct}\ -)$

Never boxed

$(T = \mathbb{Z})$

Sometimes boxed

$(T = - \to -)$

Unboxed Data

$$\mathcal{V}[\![T]\!](\ell) \triangleq \exists\, (b, i) = \ell.\ \begin{cases} \ulcorner i = 1 \urcorner \star \mathcal{R}[\![T]\!](\ell - 1) & (T = \mathsf{struct}\ -) \\ \ulcorner i = 0 \urcorner \star \mathcal{U}[\![T]\!](b) & (T = \mathbb{Z}) \\ \underbrace{\left(\ulcorner i = 0 \urcorner \star \mathcal{U}[\![T]\!](b)\right)}_{\text{Top-level functions}} \vee \underbrace{\left(\ulcorner i = 1 \urcorner \star \mathcal{R}[\![T]\!](\ell - 1)\right)}_{\text{Closures}} & (T = - \to -) \end{cases}$$

Always boxed

Never boxed

Sometimes boxed

More in the Paper

- Variations: Unboxed types, calling conventions, layout optimizations
- Theorems: Safety & memory reclamation, compiler compliance, type evolution

Next Steps

- Ongoing: `Rust`-like ABI over `Wasm` with ownership and borrowing
- Application: Verified FFI

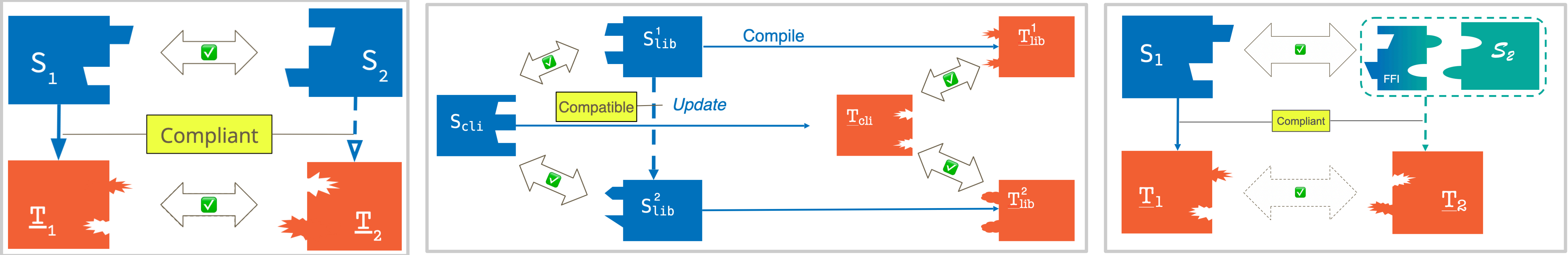
Takeaways

The Methodology

ABI Spec with Realistic Realizability

$e \in [\![\tau]\!]$

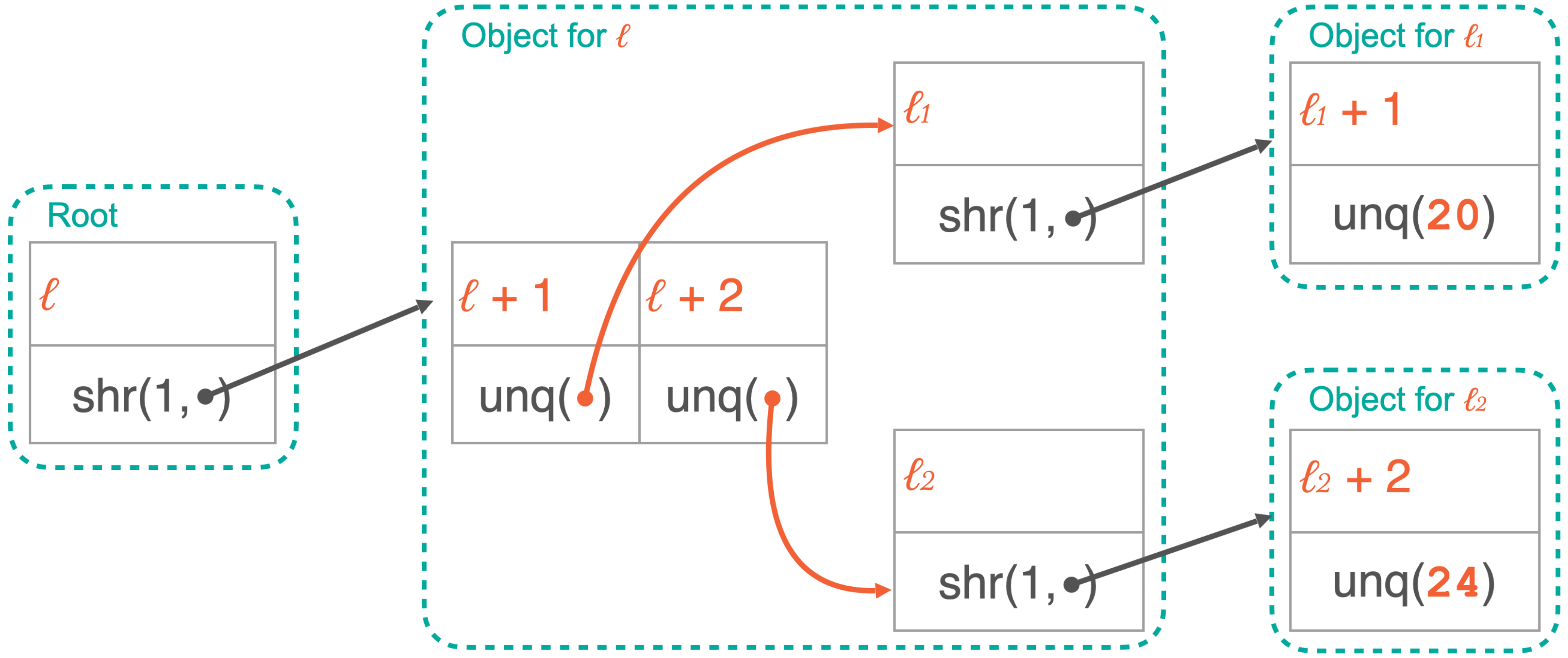
Compiler Compliance, Library Evolution, FFI Safety*



The Case Study

Graph-Based Resources for RC

$@_\ell P \quad \diamond P$



Paper
Slides
Contact