Realistic Realizability:

Specifying ABIs You Can Count On

Andrew Wagner, Zachary Eisbach, Amal Ahmed
Northeastern University

OOPSLA 2024, Pasadena, CA

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, **calling conventions**, and type **layout** information.
>
> **Behavior**
>
> — 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
```

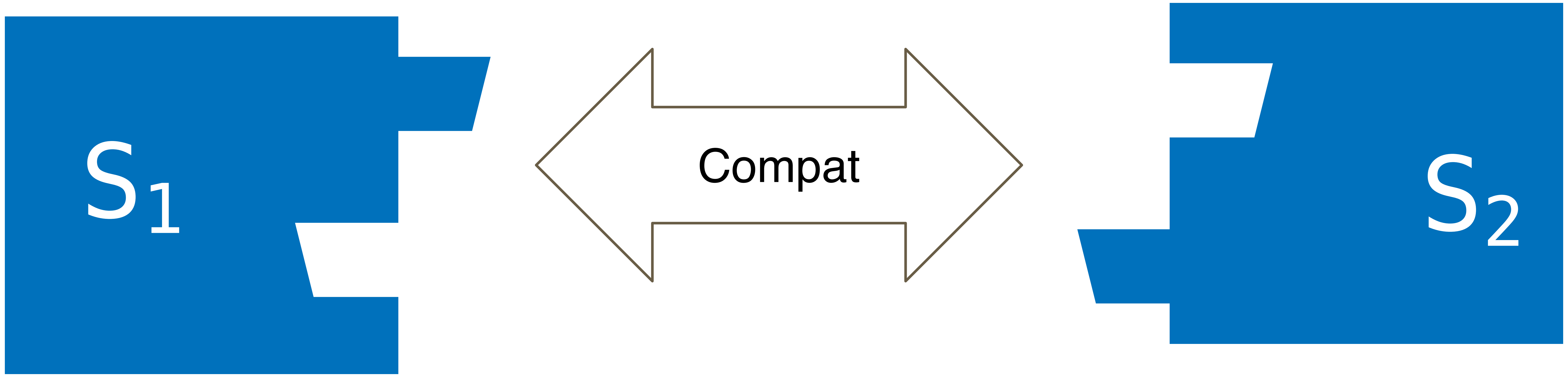
?

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

int foo(int indir[])

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

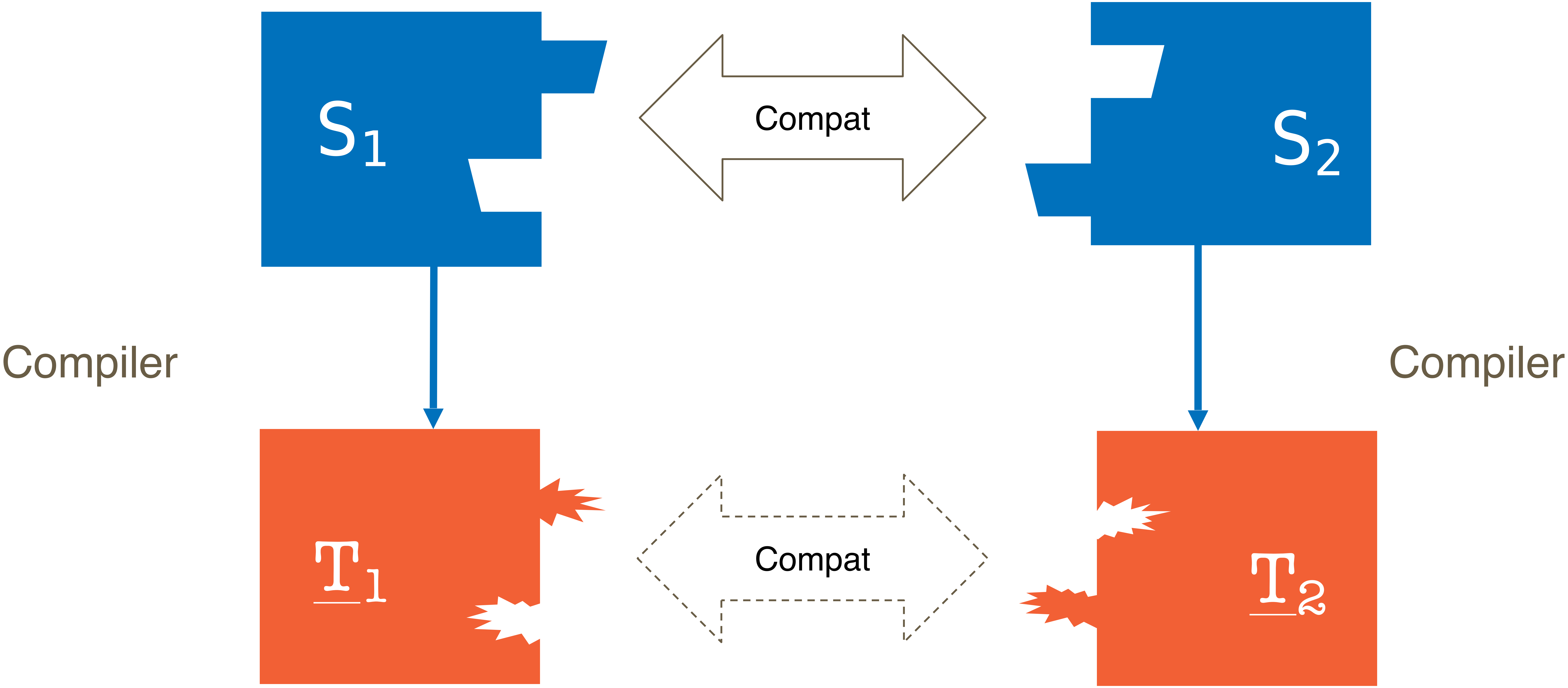
Why Use an ABI?

Why Use an ABI? Interoperability

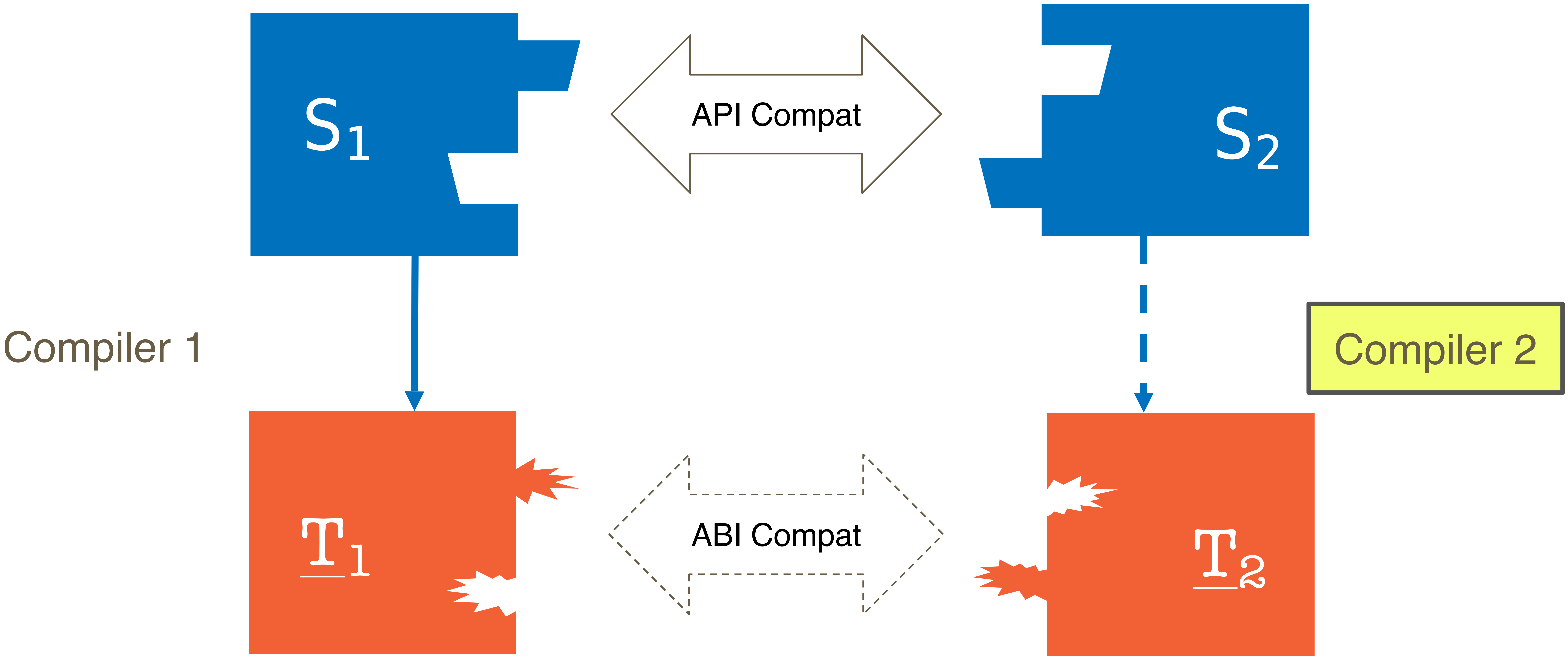
Why Use an ABI? Interoperability



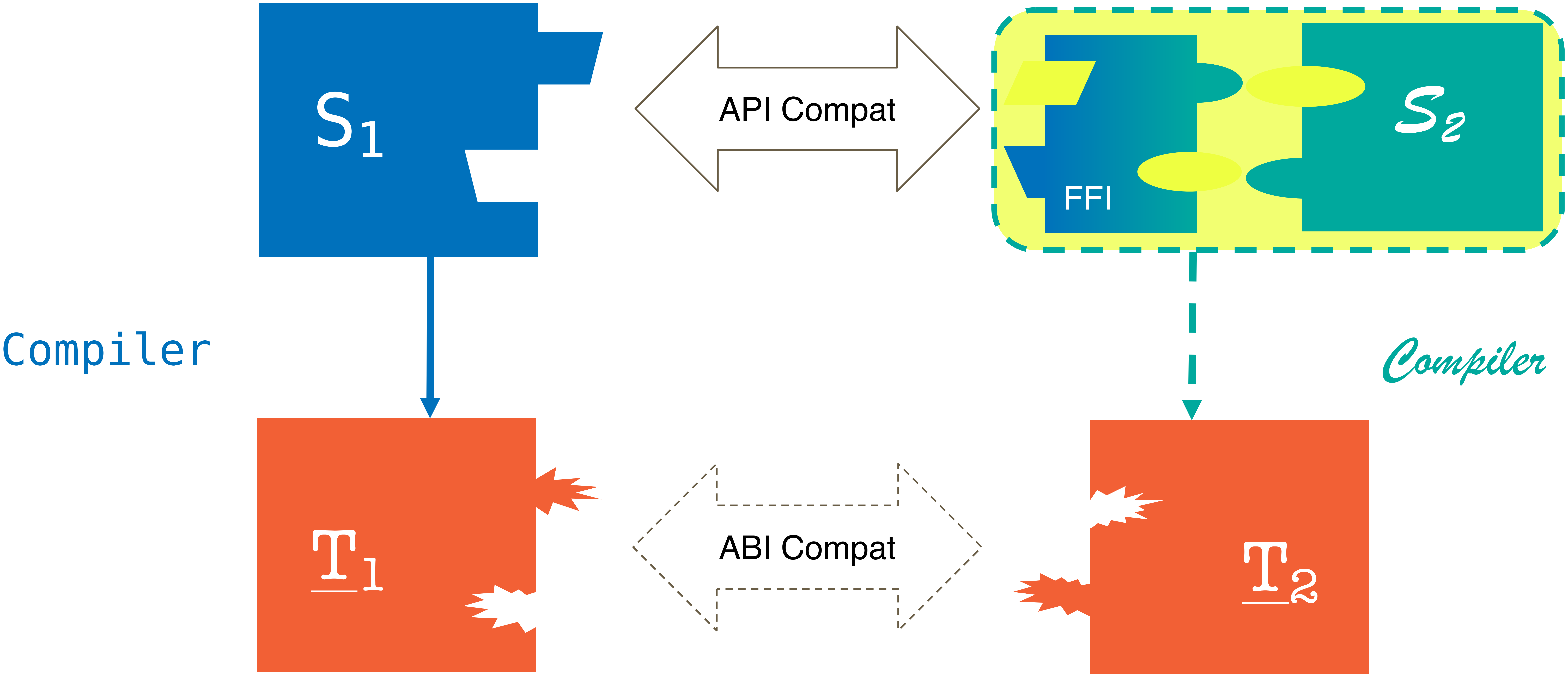
Why Use an ABI? Interoperability



Why Use an ABI? Interoperability for Compilers



Why Use an ABI? Interoperability for Languages



Who is Designing an ABI?

Swift

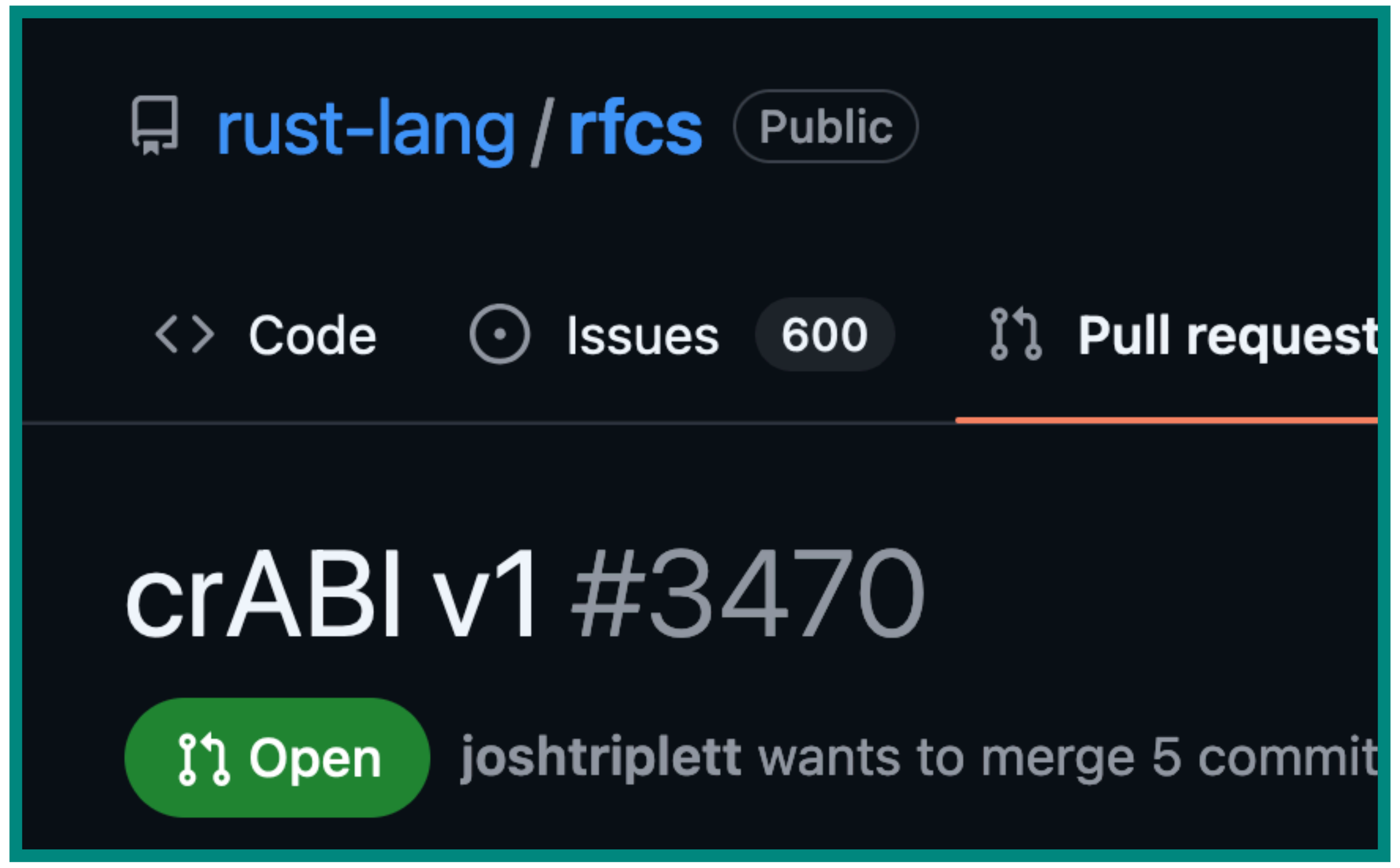
swift / docs / ABIStabilityManifesto.md

dev [docs] Update links to repositories moved to https://github.co

439 lines (230 loc) · 56.2 KB

Swift ABI Stability Manifesto

Rust

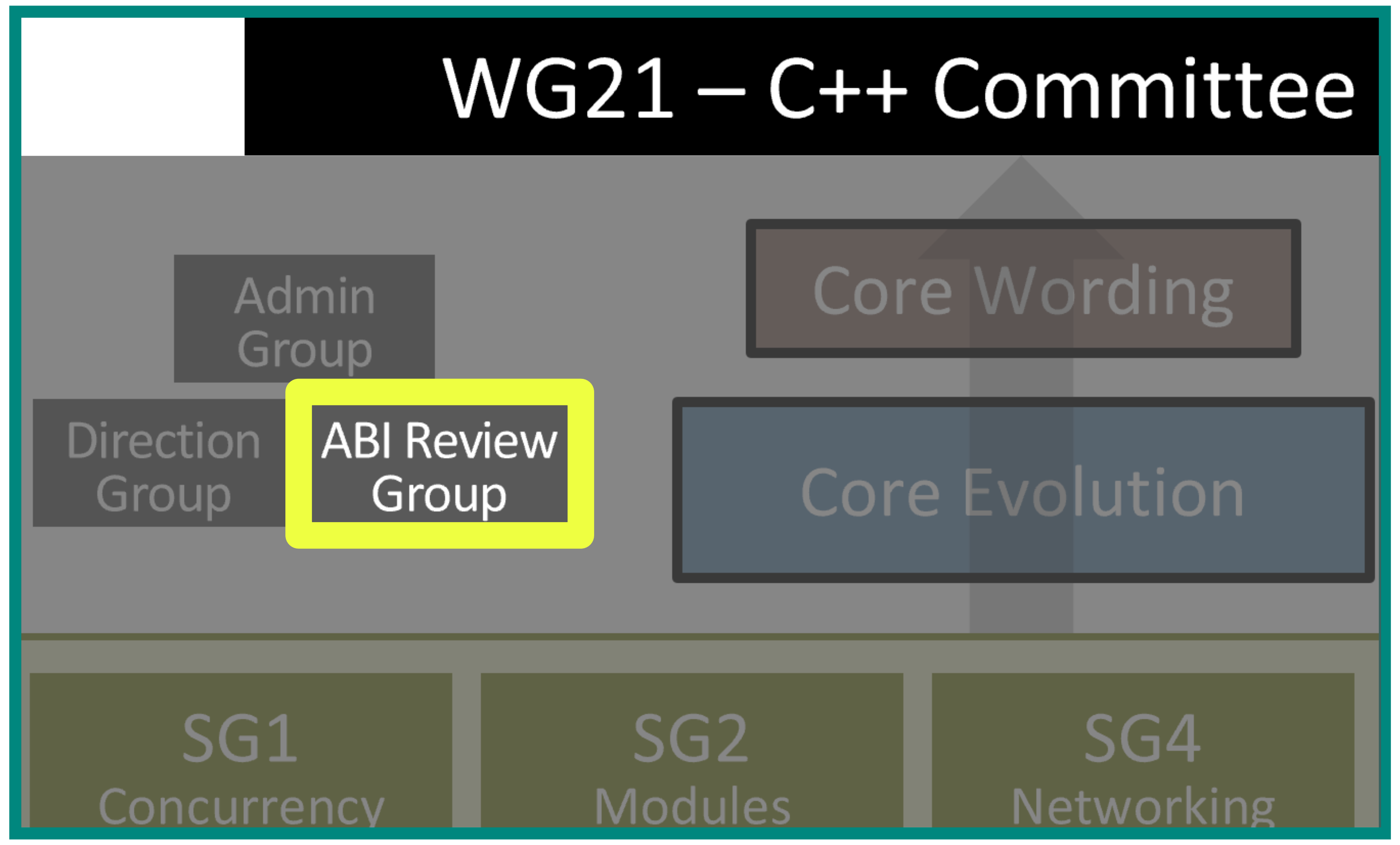


rust-lang / rfcs

crABI v1 #3470

joshtriplett wants to merge 5 commit

C++



WG21 – C++ Committee

Who is Designing an ABI?

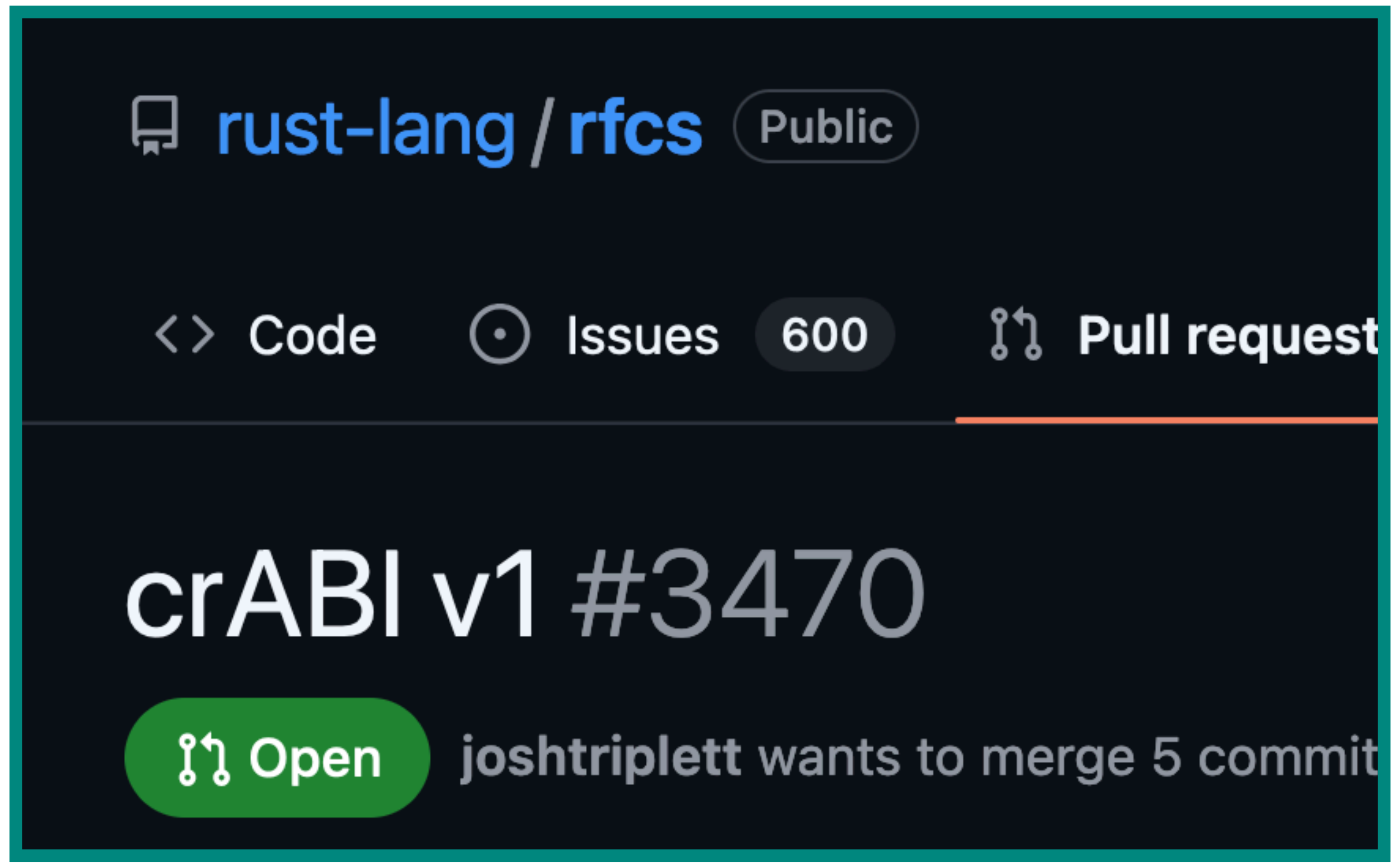
Swift

dev [docs] Update links to repositories moved to https://github.co

439 lines (230 loc) · 56.2 KB

Swift ABI Stability Manifesto

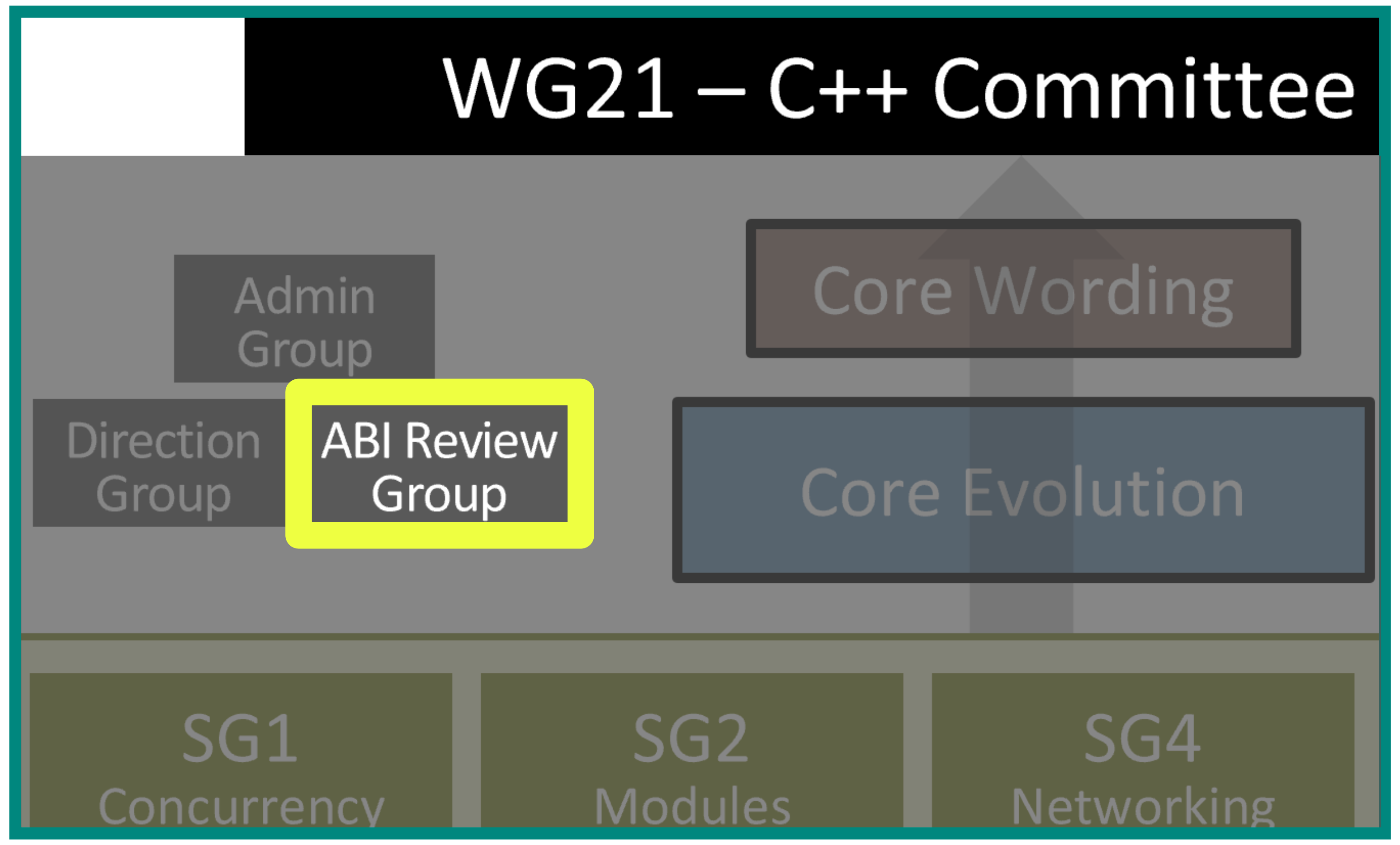
Rust



crABI v1 #3470

joshtriplett wants to merge 5 commit

C++



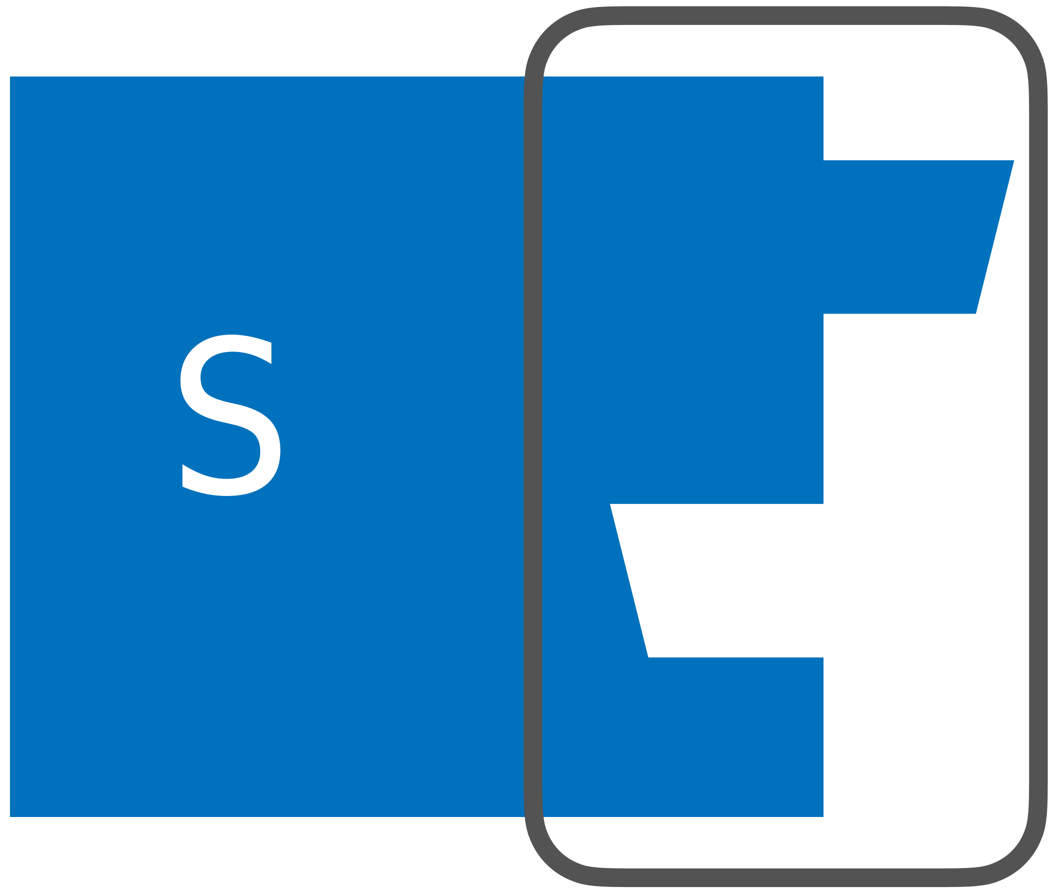
Richer Types → Richer ABIs?

How Should We Specify an ABI?

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

How Should We Specify an ABI?

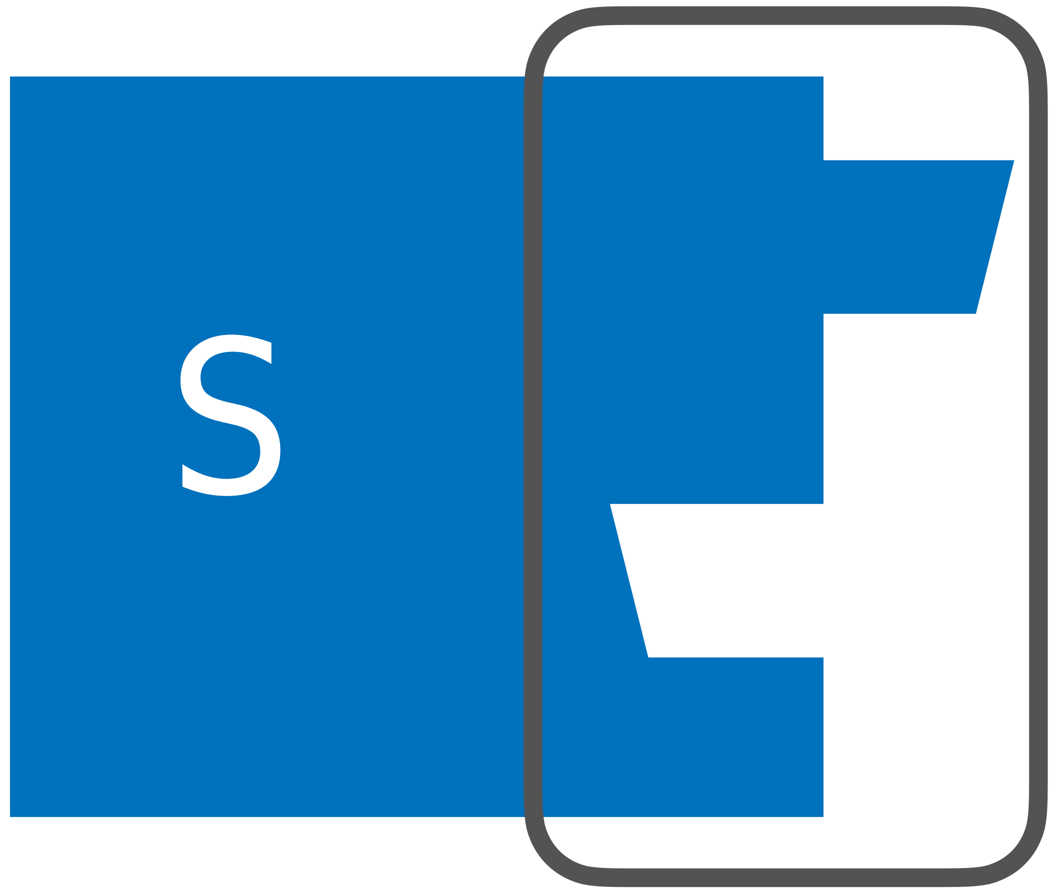
`The run-time contract for using a particular API`



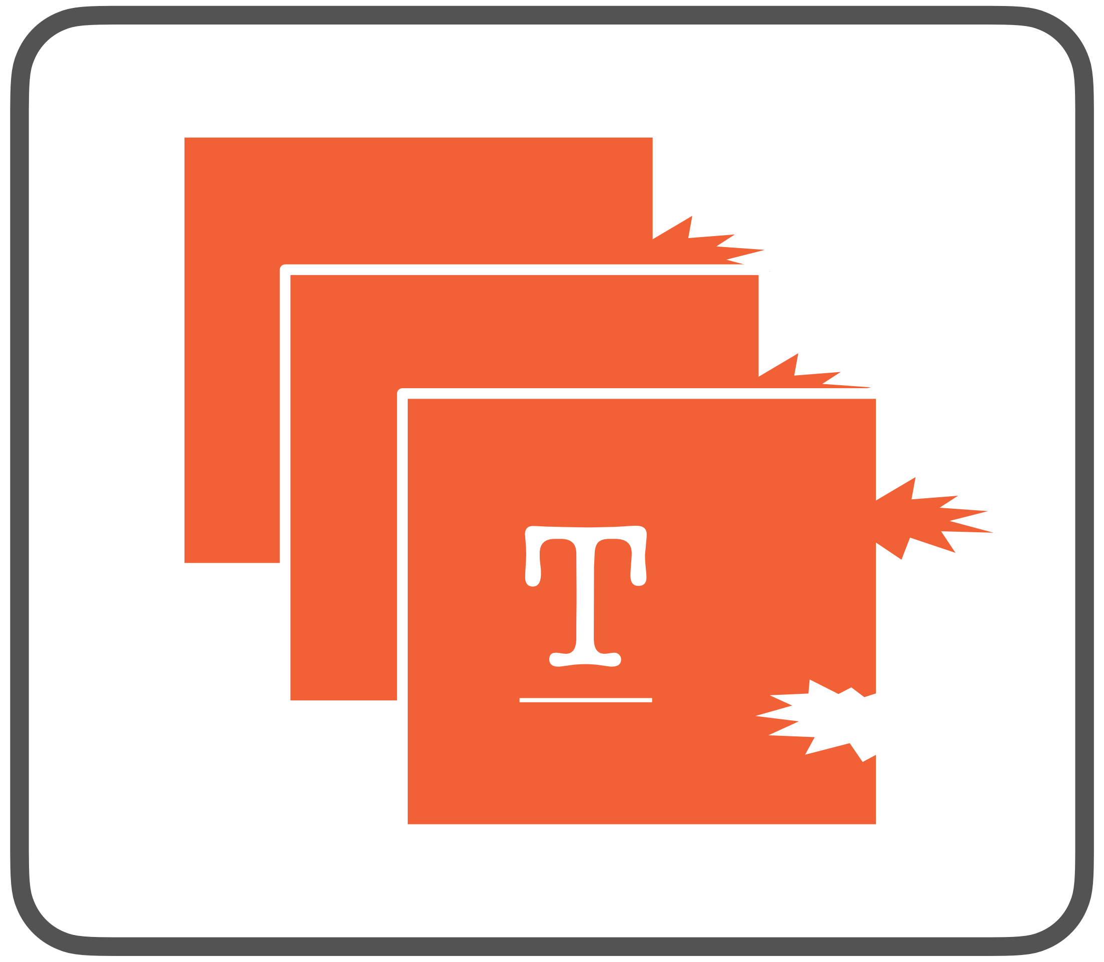
This Type `τ`

How Should We Specify an ABI?

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



This Type τ

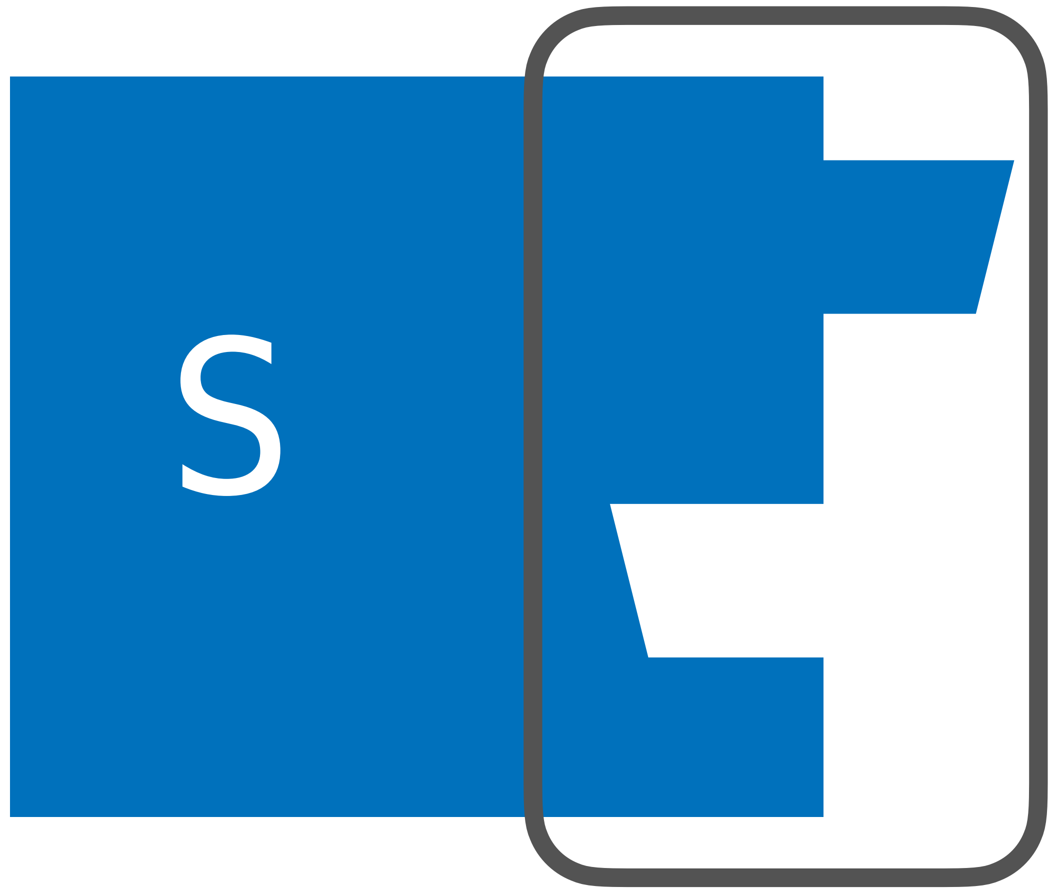


Is *Realized By* These Target Programs

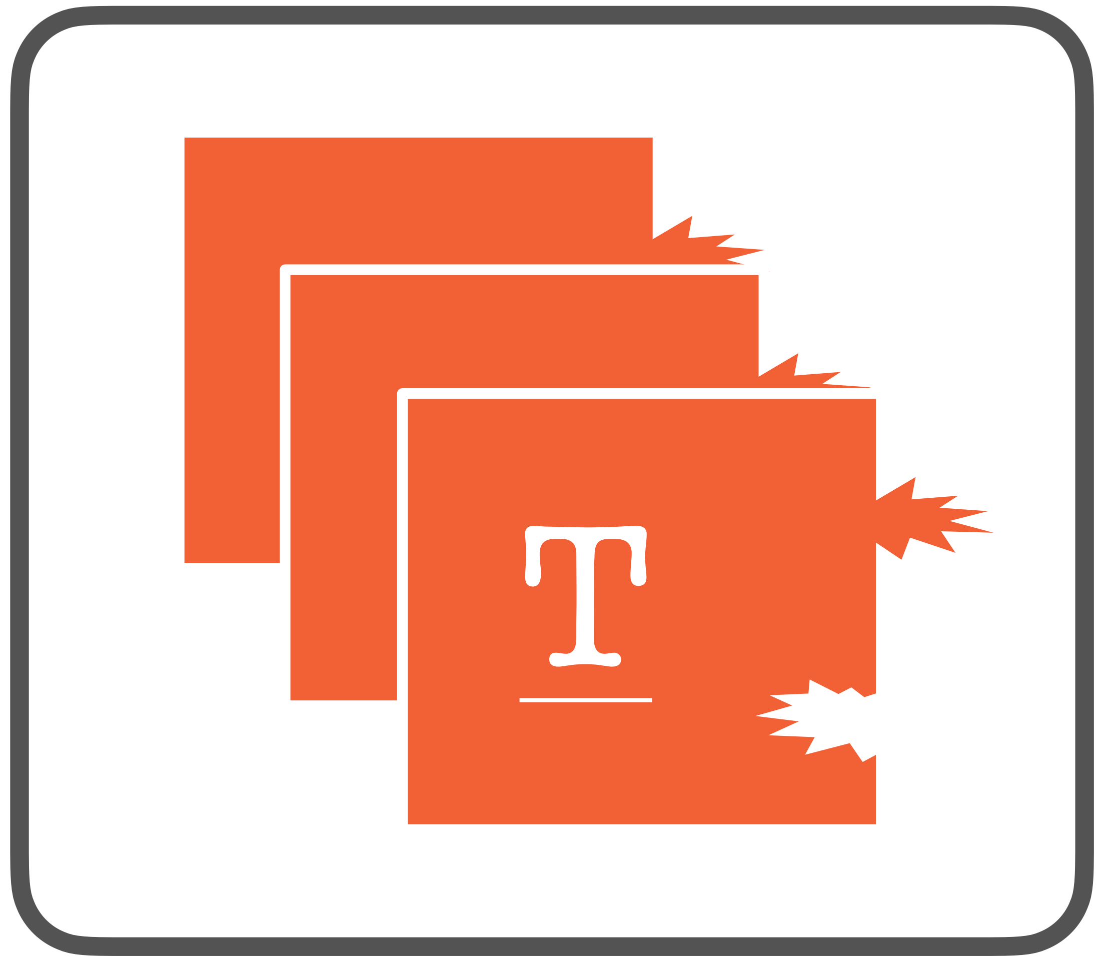
$[\![\tau]\!] = \{ \underline{e} \mid \dots \}$

How Should We Specify an ABI?

The run-time contract for using a particular API



This Type τ



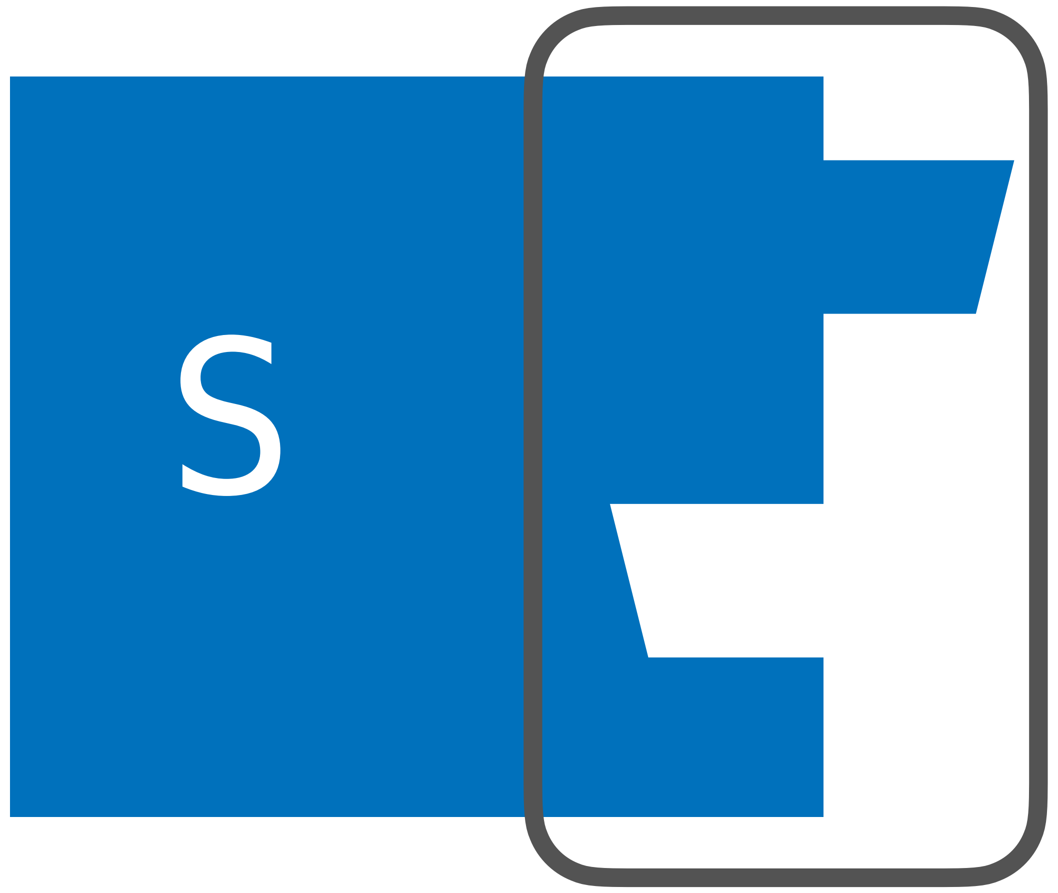
Is *Realized By* These Target Programs

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

Semantic Typing using *Realistic Realizability* [Benton06]

How Should We Specify an ABI?

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

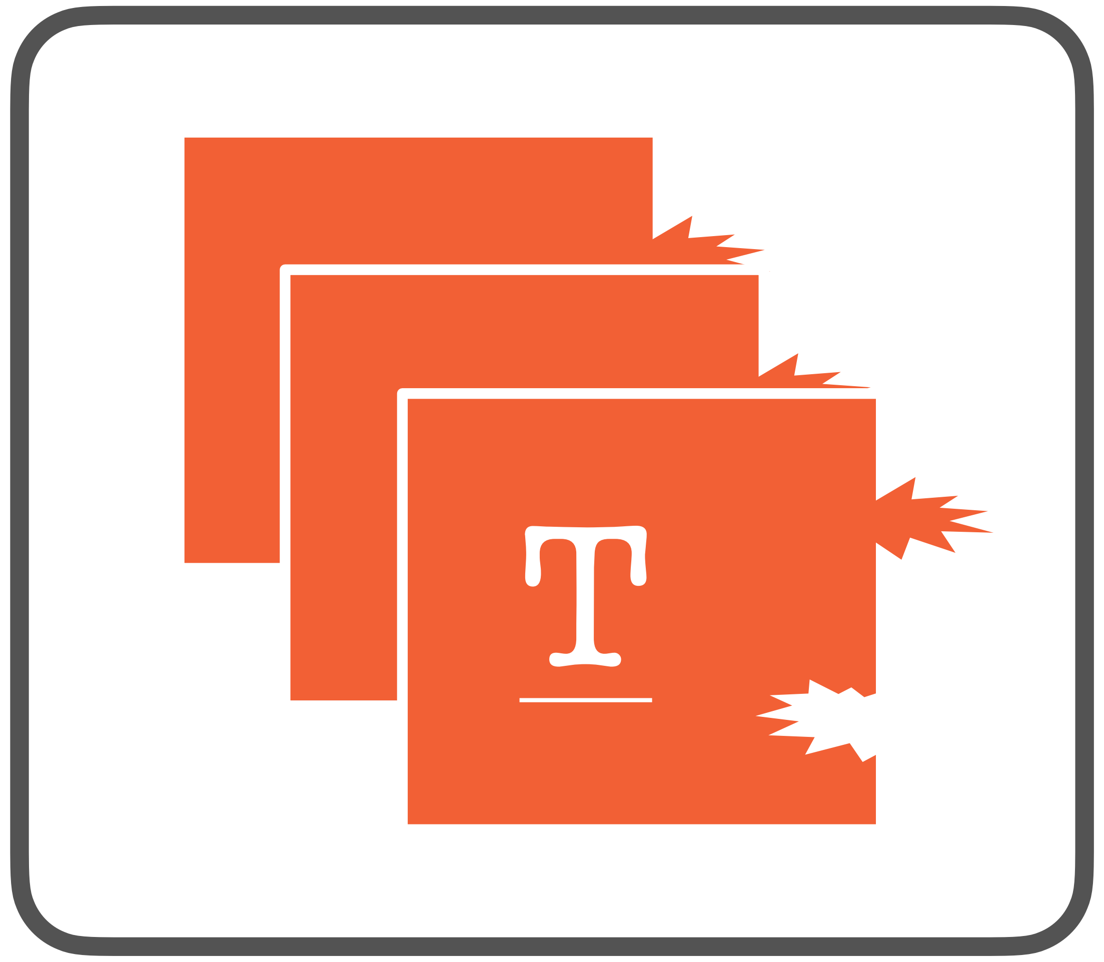


This Type τ

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]

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

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

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`

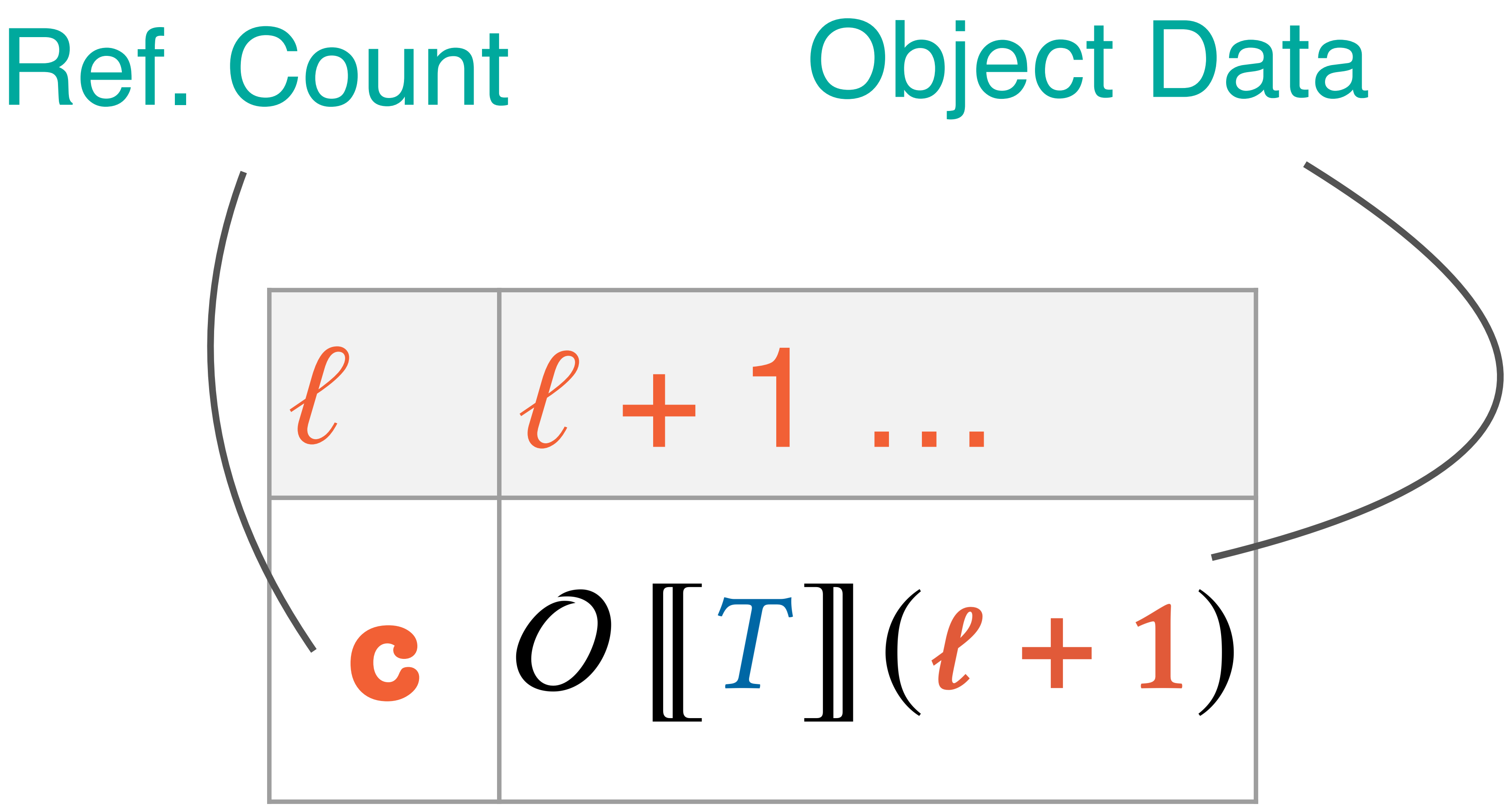
e is **ABI compliant** with τ if

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

References: Layout

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

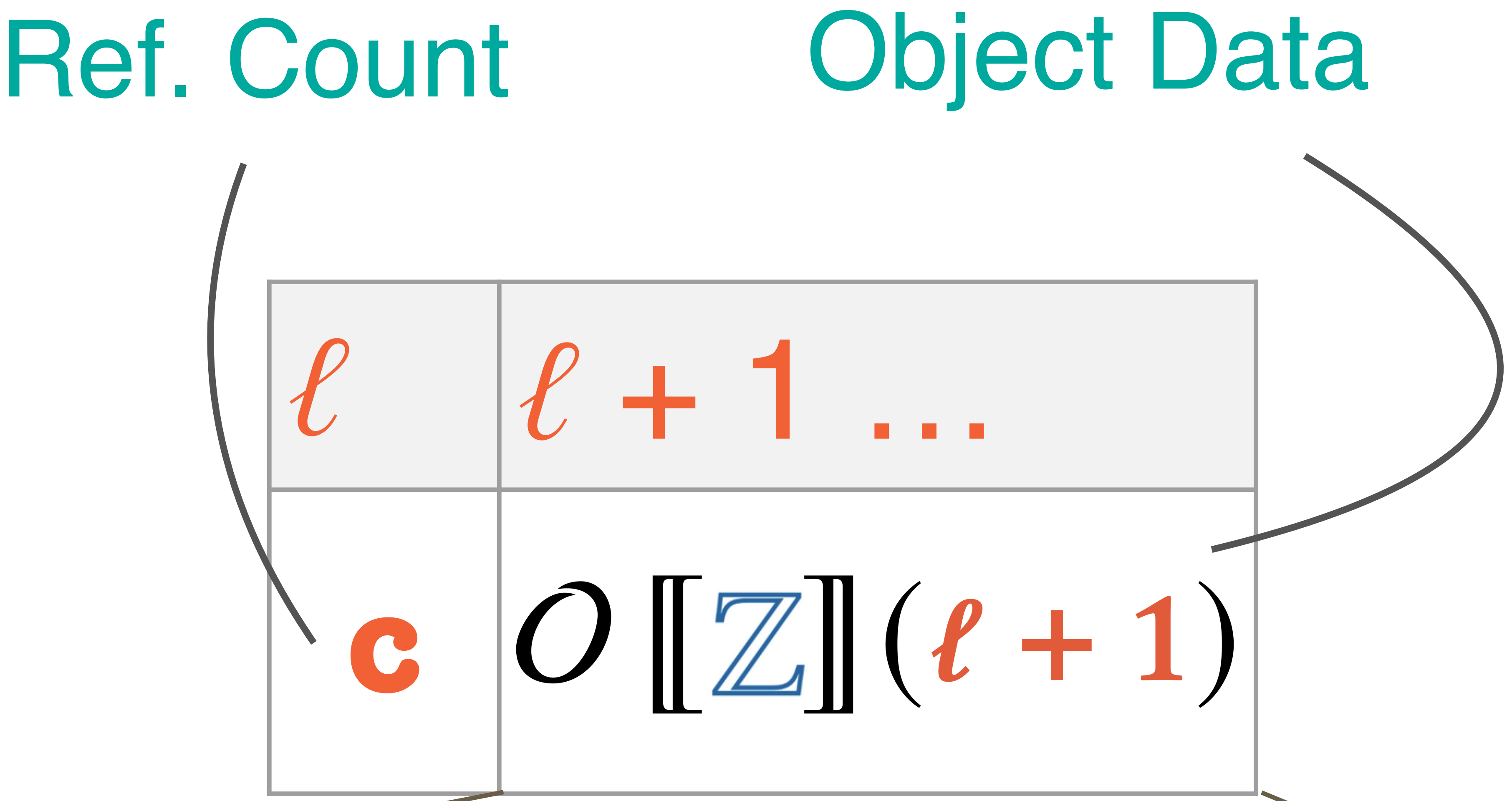
$$\mathcal{R}[\![T]\!](\ell) \approx \begin{array}{|c|l|}\hline \ell & \ell + 1 \ \dots \\ \hline \mathbf{c} & \mathcal{O}[\![T]\!](\ell + 1) \\ \hline \end{array}$$



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	$\ell + 1 \ldots$
c	$\mathcal{O}[\![\mathbb{Z}]\!](\ell + 1)$

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

More in Paper:
Unboxed types

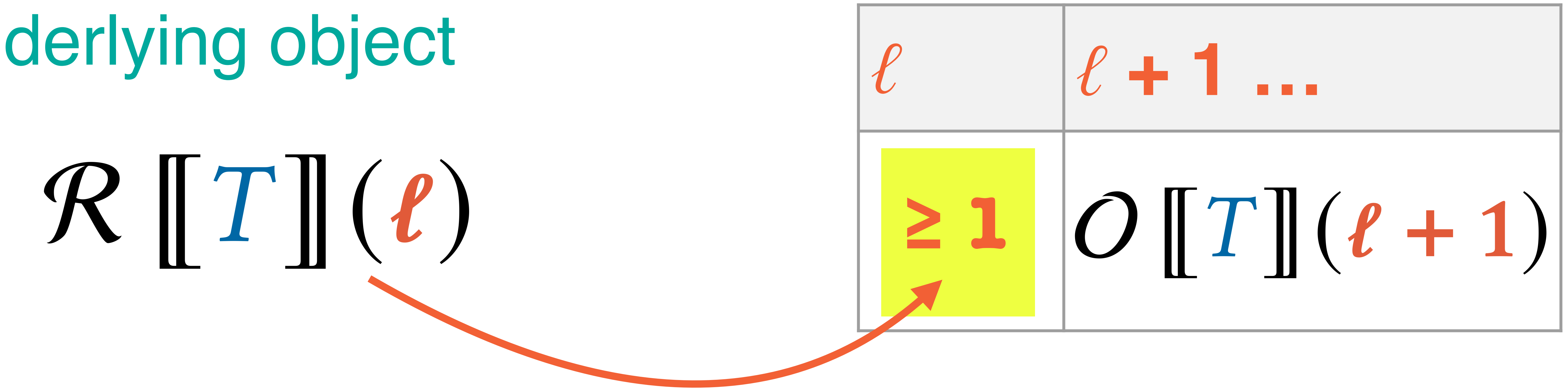
References: Ownership + Sharing

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

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

References: Ownership + Sharing

Single reference represents one share of underlying object

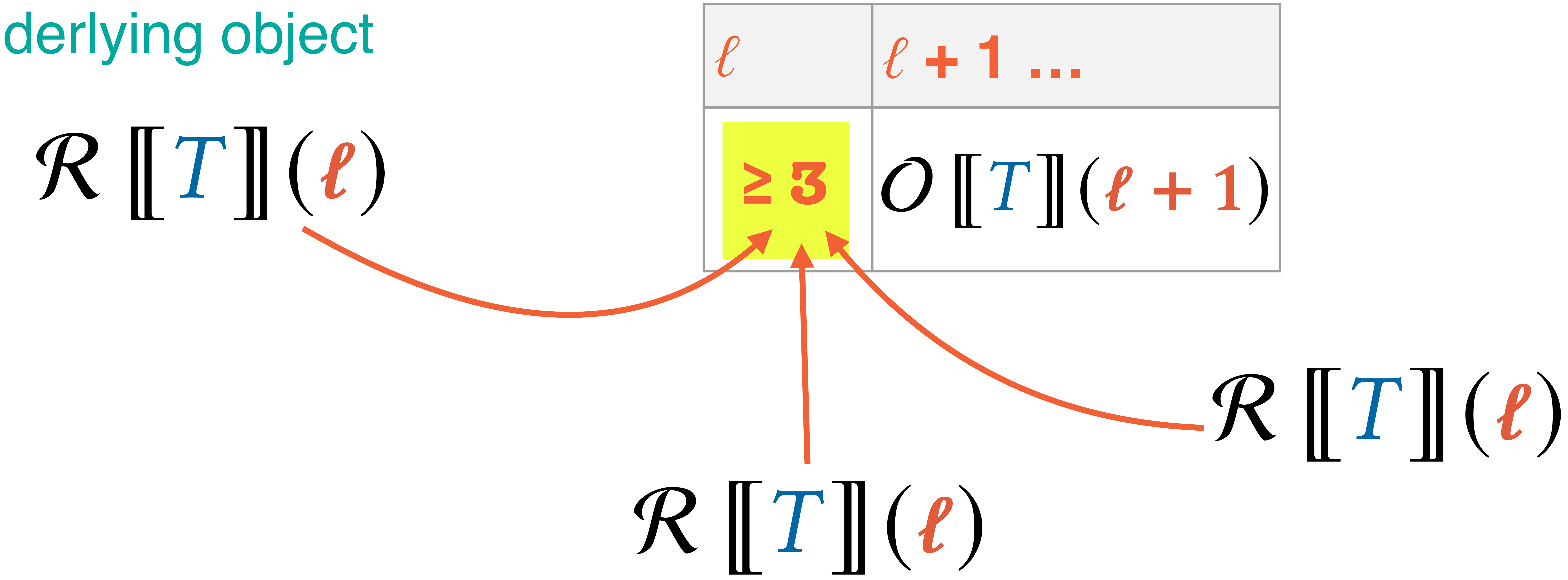


$$\mathcal{R}\,[\![T]\!](\boldsymbol{\ell})$$

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

References: Ownership + Sharing

Single reference represents one share of underlying object



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

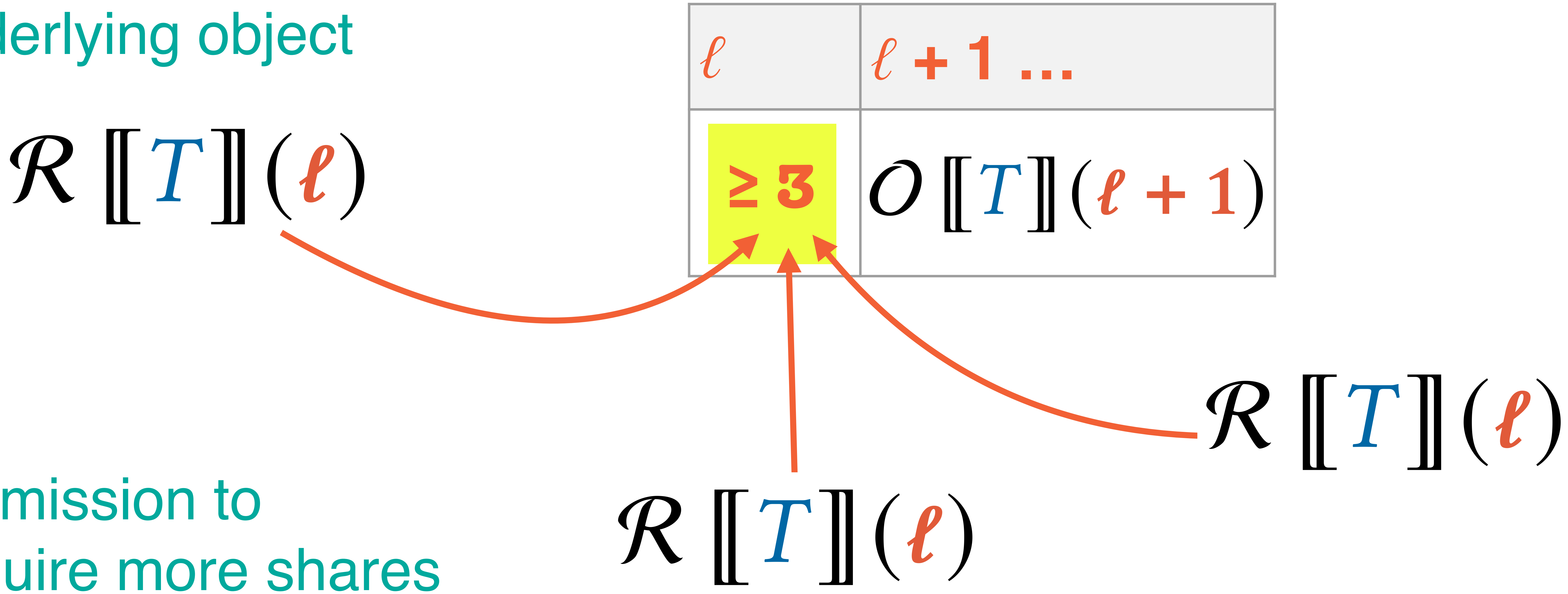
ℓ	$\ell + 1$ ...
≥ 3	$\mathcal{O}[\![T]\!](\ell + 1)$

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

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

References: Ownership + Sharing

Single reference represents one share of underlying object



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

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

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

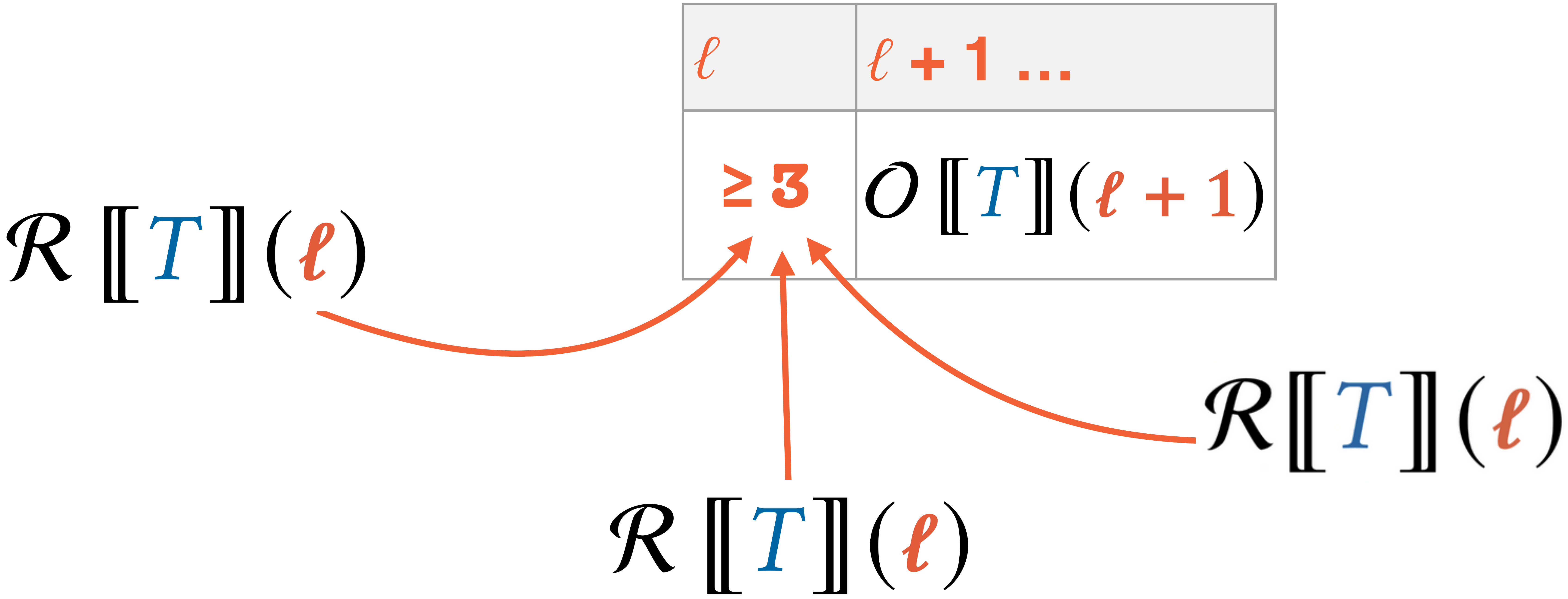
$$\mathcal{R}[\![T]\!](\ell)$$

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



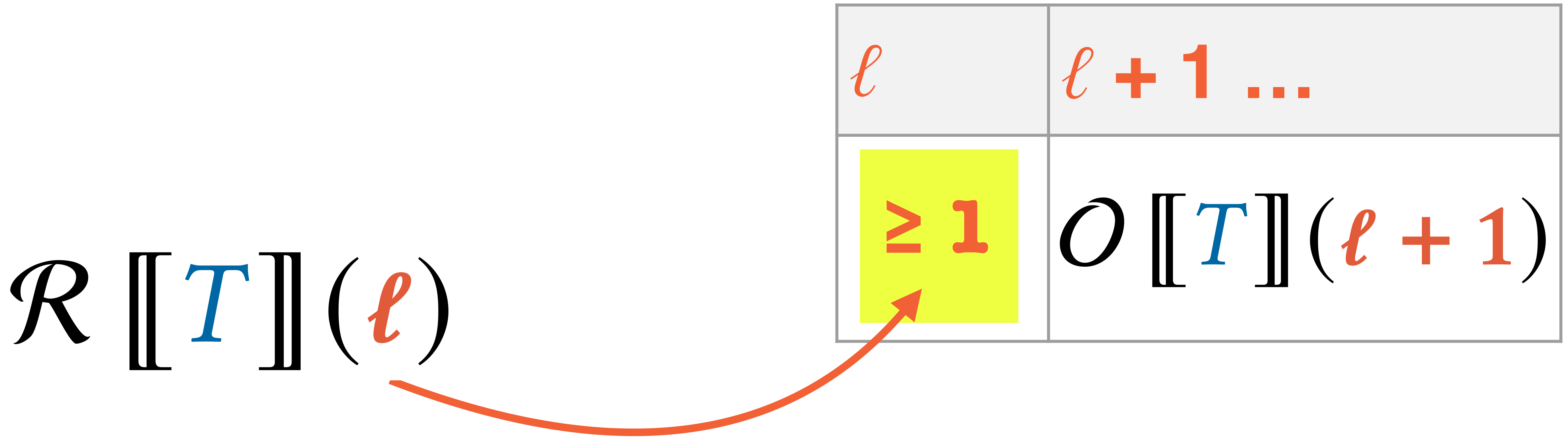
ℓ	$\ell + 1$ ...
≥ 3	$\mathcal{O}[\![T]\!](\ell + 1)$

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

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

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

References: Ownership + Sharing



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

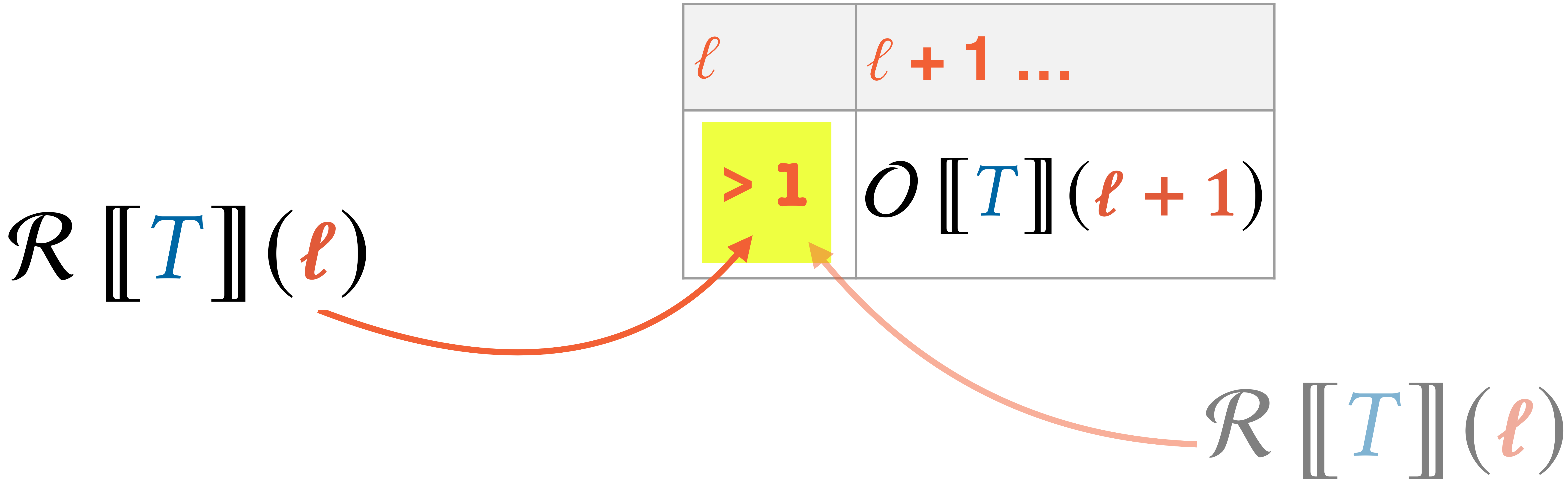
ℓ	ℓ + 1 ...
≥ 1	$\mathcal{O}\,[\![T]\!](\ell + 1)$

Reference confers permission to decrement count & release shares

RC-DECR

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

References: Ownership + Sharing



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

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

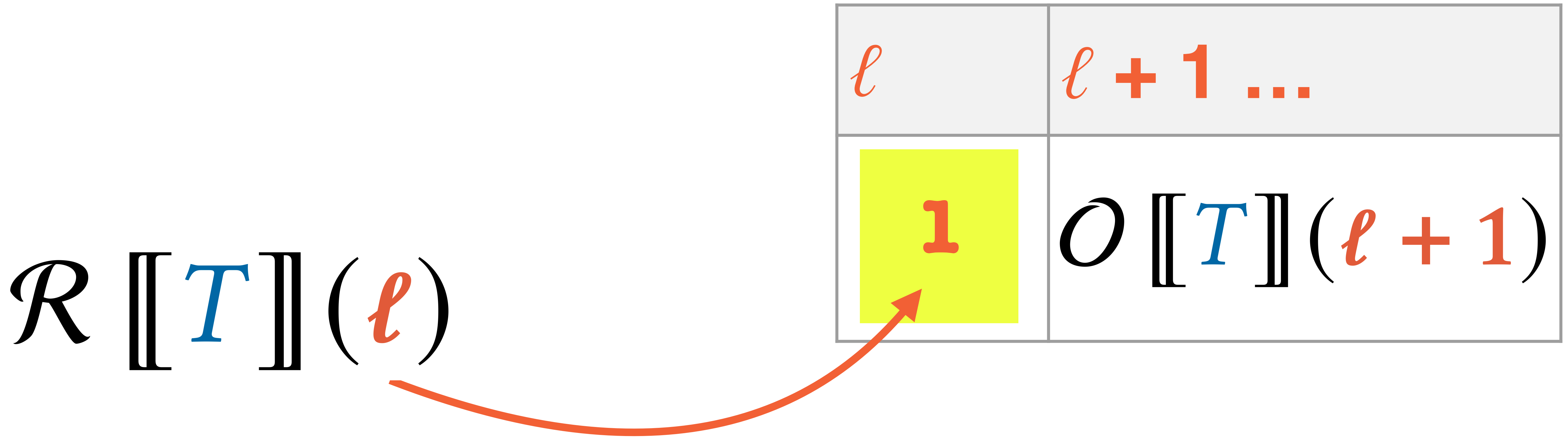
$\mathcal{R}[\![T]\!](\ell)$

Reference confers permission to decrement count & release shares

RC-DECR

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

References: Ownership + Sharing



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

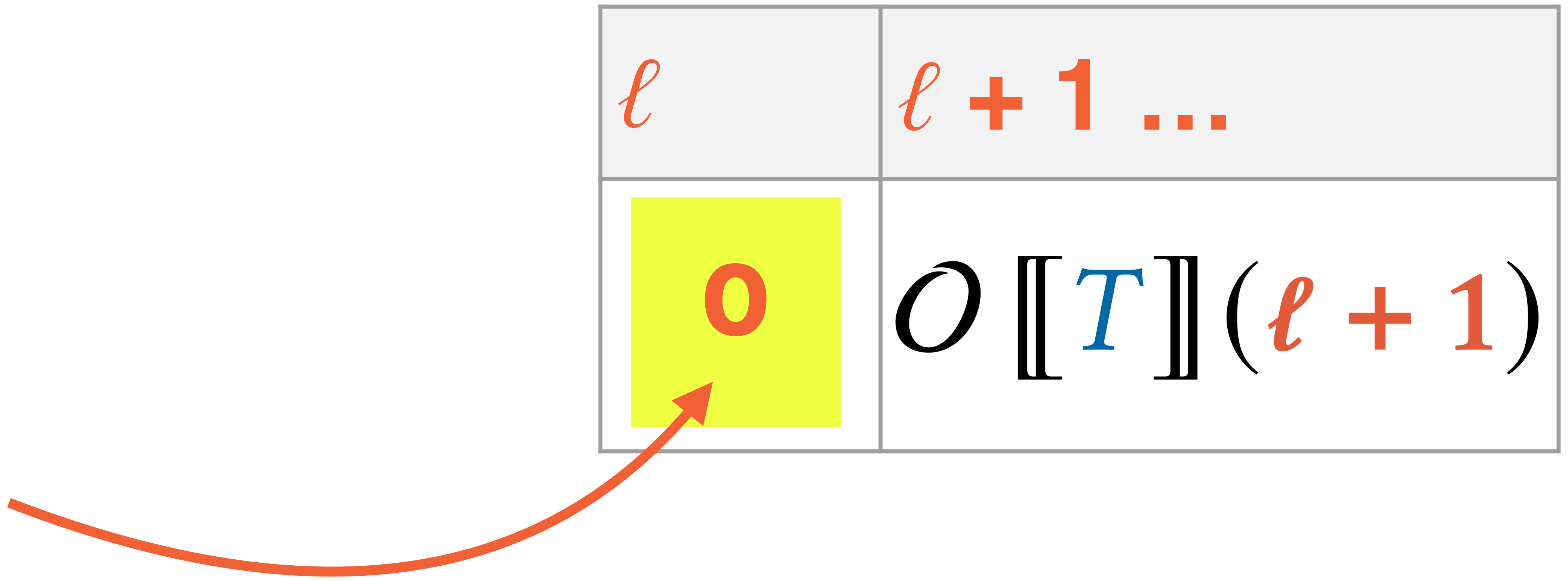
ℓ	$\ell + 1 \ldots$
1	$\mathcal{O}[\![T]\!](\ell + 1)$

Reference confers permission to decrement count & release shares

RC-DECR

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

References: Ownership + Sharing



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

Reference confers permission to
decrement count & release shares

RC-DECR

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

Calling Conventions

$$\mathcal{O}\llbracket T_1 \rightarrow T_2 \rrbracket(\ell) \triangleq \exists f.\, \ell \mapsto f \star$$

Pointer to function

Calling Conventions

$$\begin{aligned} &\mathcal{O}[\![T_1 \longrightarrow T_2]\!](\ell) \overset{\triangle}{\approx} \exists f.\ \ell \mapsto f \star \\ &\qquad \forall \ell_1.\ \left\{\mathcal{R}[\![T_1]\!](\ell_1)\right\}\ f(\ell_1)\ \left\{\ell_2.\ \mathcal{R}[\![T_2]\!](\ell_2)\right\} \end{aligned}$$

Pointer to function

Calling convention:
Caller increment

Calling Conventions

$$\boxed{\begin{aligned} &\mathcal{O}[\![T_1 \longrightarrow T_2]\!](\ell) \overset{\triangle}{\approx} \exists\, f.\ \ell \mapsto f \star \\ &\qquad \forall\, \ell_1.\ \left\{\mathcal{R}[\![T_1]\!](\ell_1)\right\}\ f(\ell_1)\ \left\{\ell_2.\ \mathcal{R}[\![T_2]\!](\ell_2)\right\} \end{aligned}}$$

Pointer to function

Calling convention:
Caller increment

vs.

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

Calling Conventions

$$\boxed{\begin{aligned}&\mathcal{O}[\![T_1 \rightarrow T_2]\!](\ell) \overset{\triangle}{\approx} \exists\, f.\, \ell \mapsto f \star \\ &\qquad \forall\, \ell_1.\, \left\{\mathcal{R}[\![T_1]\!](\ell_1)\right\}\, f(\ell_1)\, \left\{\ell_2.\, \mathcal{R}[\![T_2]\!](\ell_2)\right\}\end{aligned}}$$

Pointer to function

Calling convention:
Caller increment

vs.

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

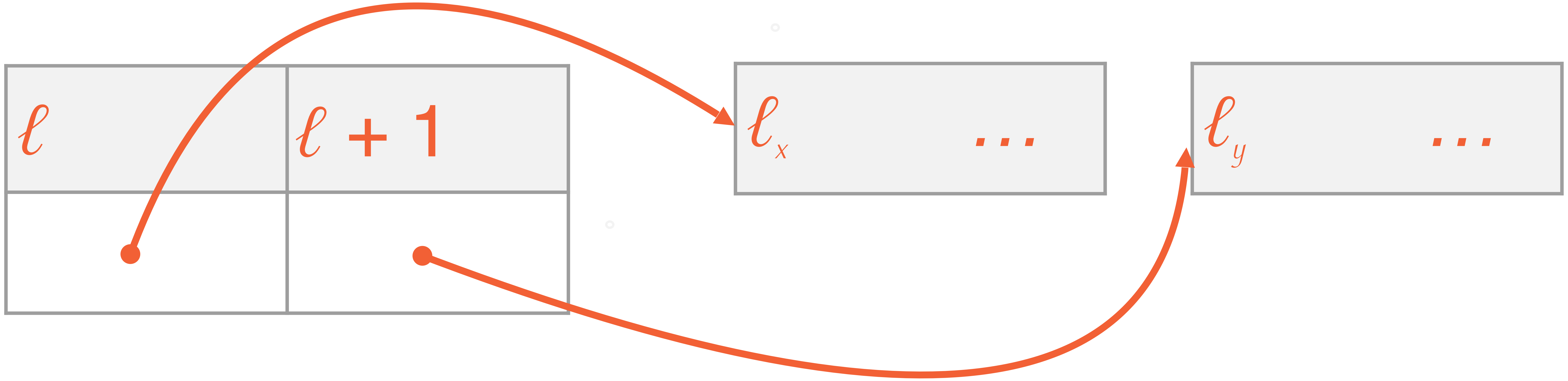
More in Paper:
Recursive closures

Aggregate Layout

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

Aggregate Layout

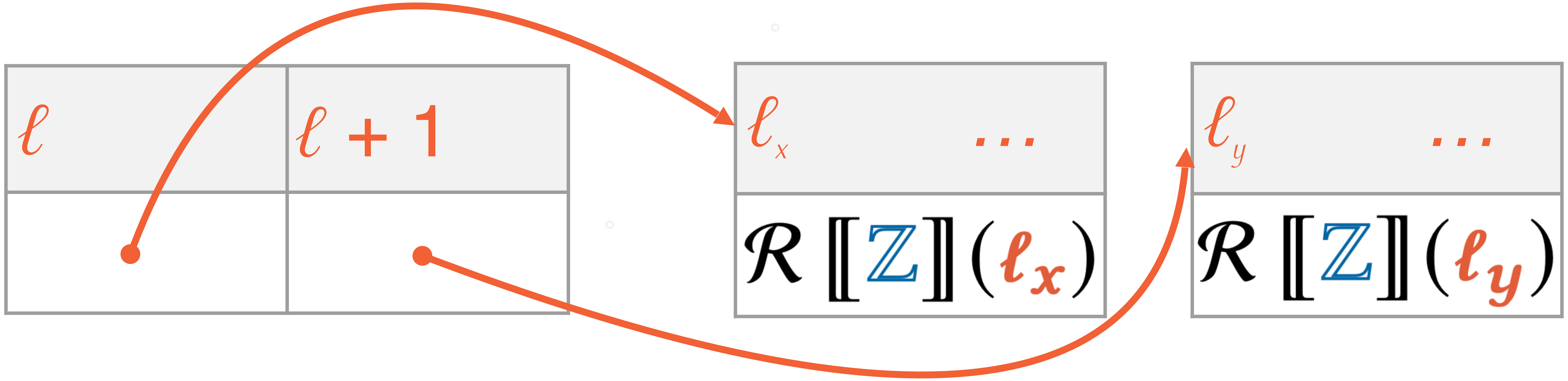
$$\mathcal{O}\,[\![\text{struct Point}\ \{x : \mathbb{Z}, y : \mathbb{Z}\}]\!](\ell)$$



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

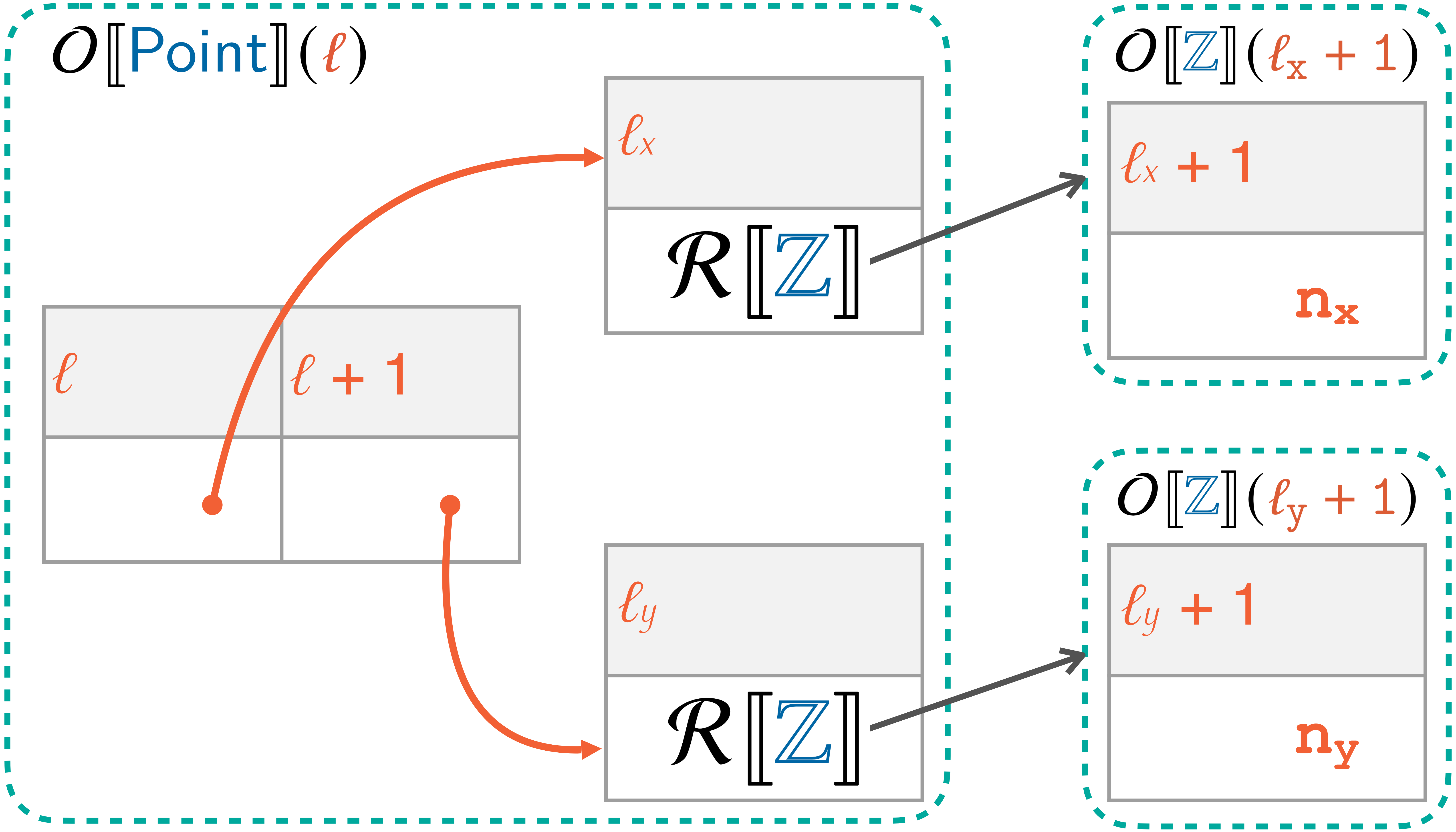
Aggregate Layout

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

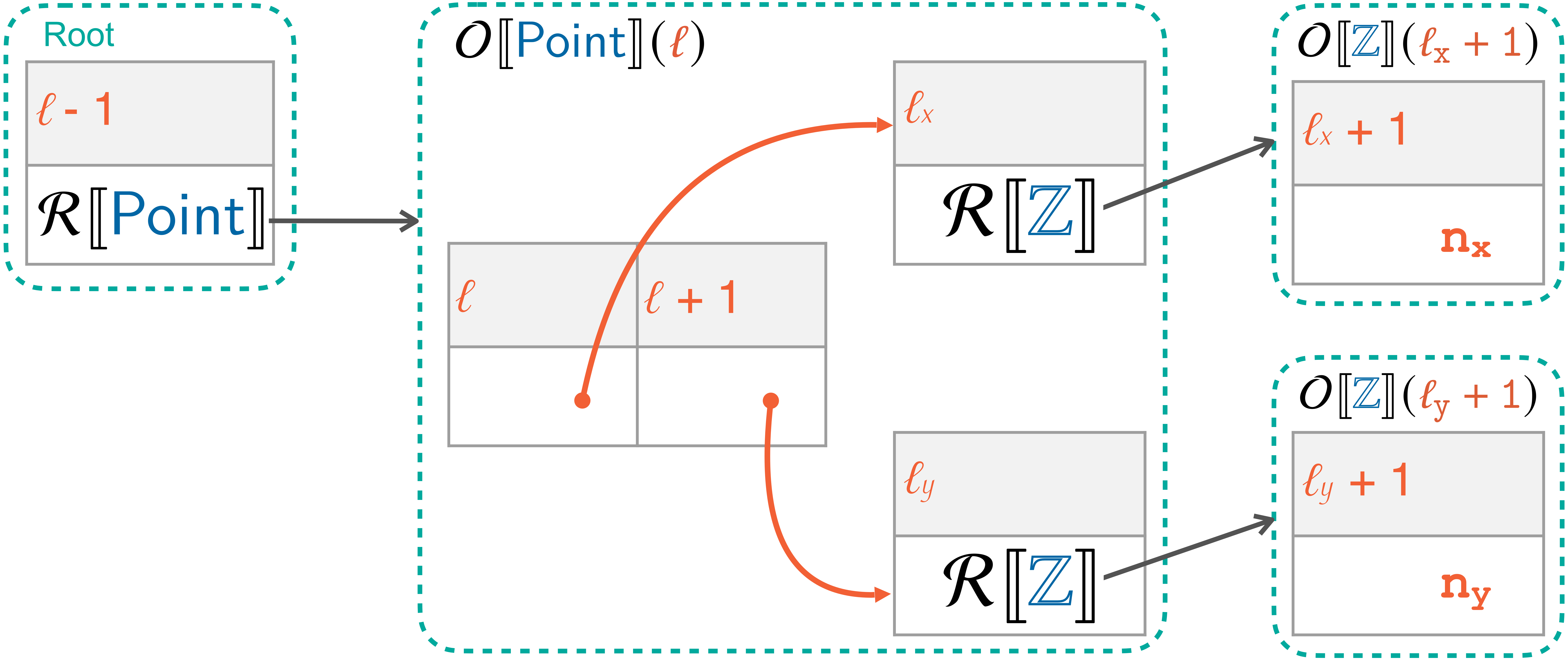


$$\exists\, \ell_x, \ell_y.\ \underbrace{\ell \mapsto \ell_x \star \ell + 1 \mapsto \ell_y}_{\text{Physical footprint}} \star \underbrace{\mathcal{R}[\![\mathbb{Z}]\!](\ell_x) \star \mathcal{R}[\![\mathbb{Z}]\!](\ell_y)}_{\text{Logical footprint includes permission to access fields}}$$

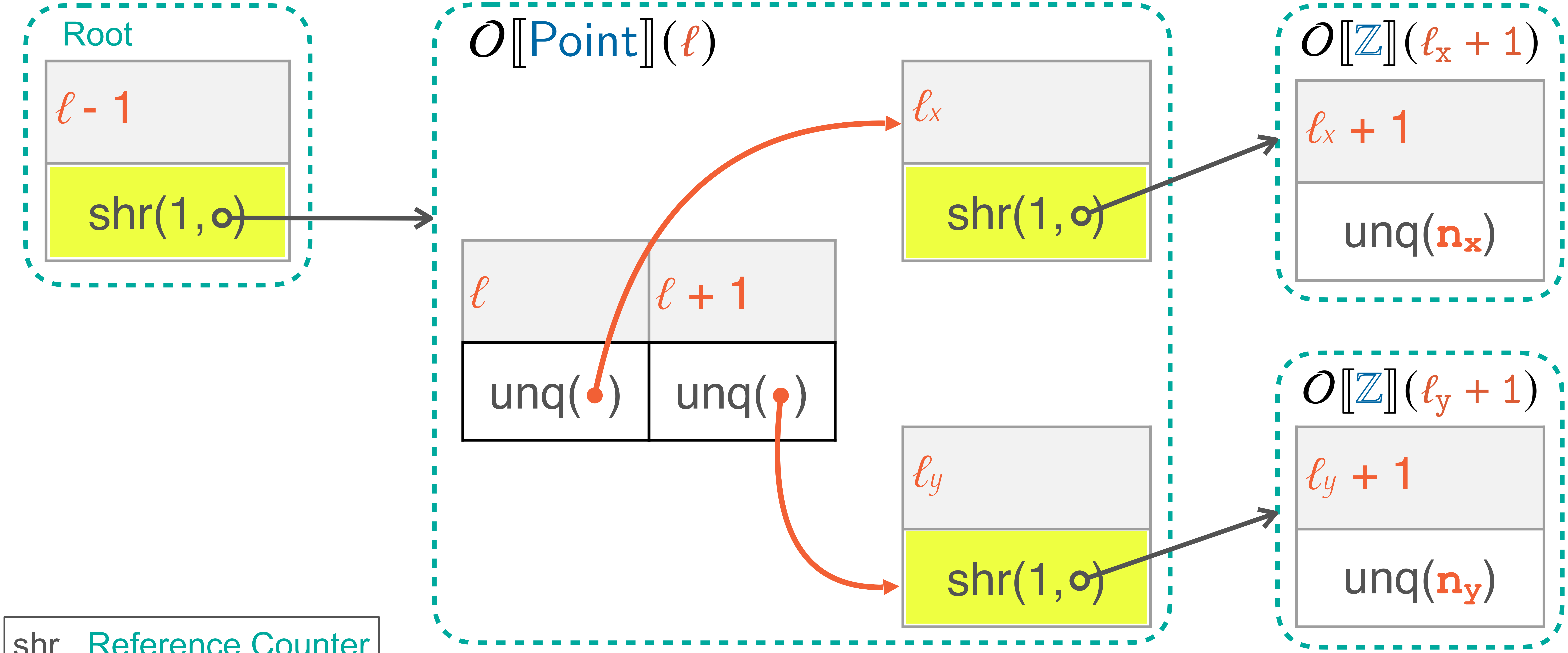
Resource Graphs



Resource Graphs

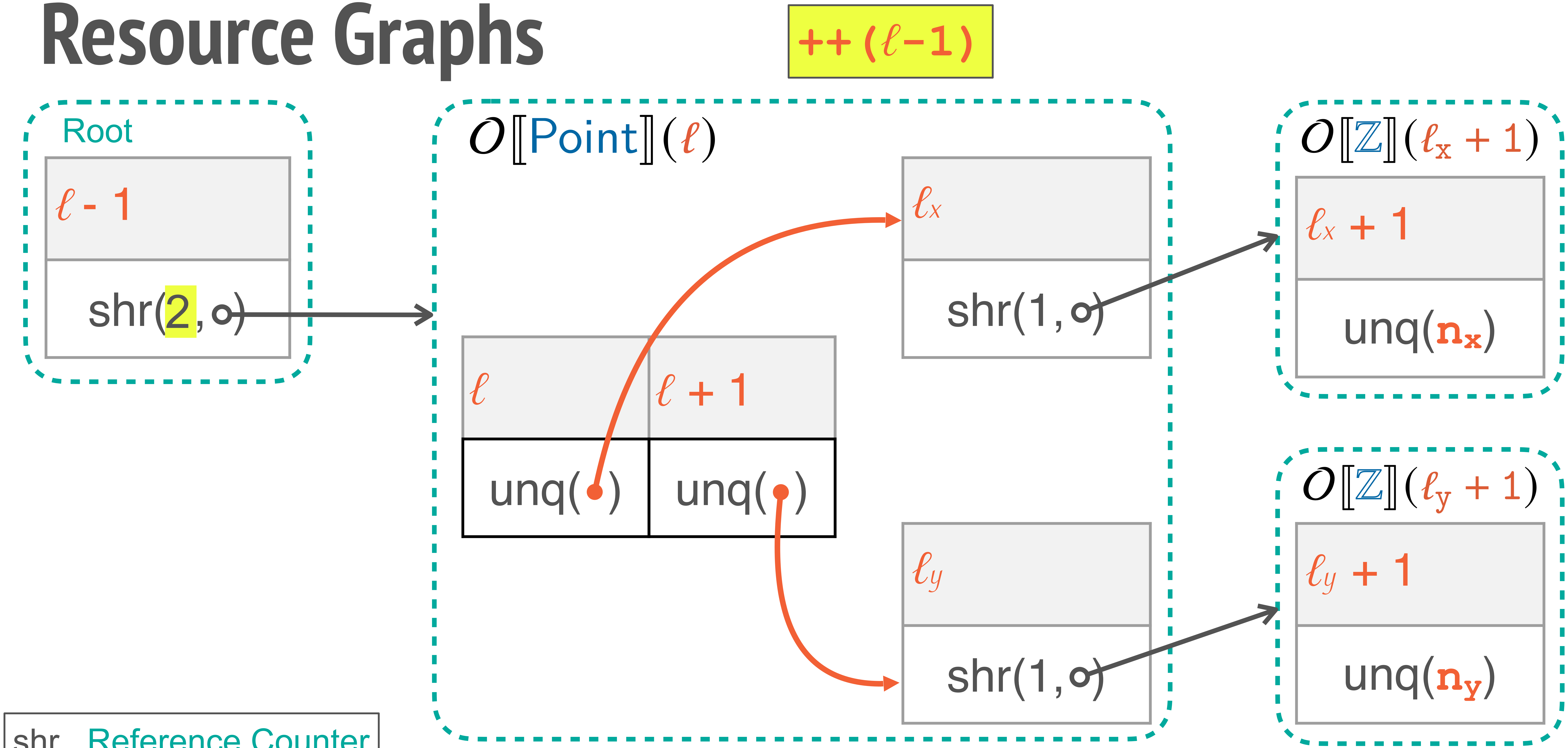


Resource Graphs



shr	Reference Counter
unq	Plain Old Data

Resource Graphs



`++(ℓ-1)`

Root

ℓ - 1

shr(2, ∘)

$\mathcal{O}[\![\text{Point}]\!](\ell)$

ℓ

unq(•)

$\ell + 1$

unq(•)

ℓ_x

shr(1, ∘)

ℓ_y

shr(1, ∘)

$\mathcal{O}[\![\mathbb{Z}]\!](\ell_x + 1)$

$\ell_x + 1$

unq($\mathtt{n_x}$)

$\mathcal{O}[\![\mathbb{Z}]\!](\ell_y + 1)$

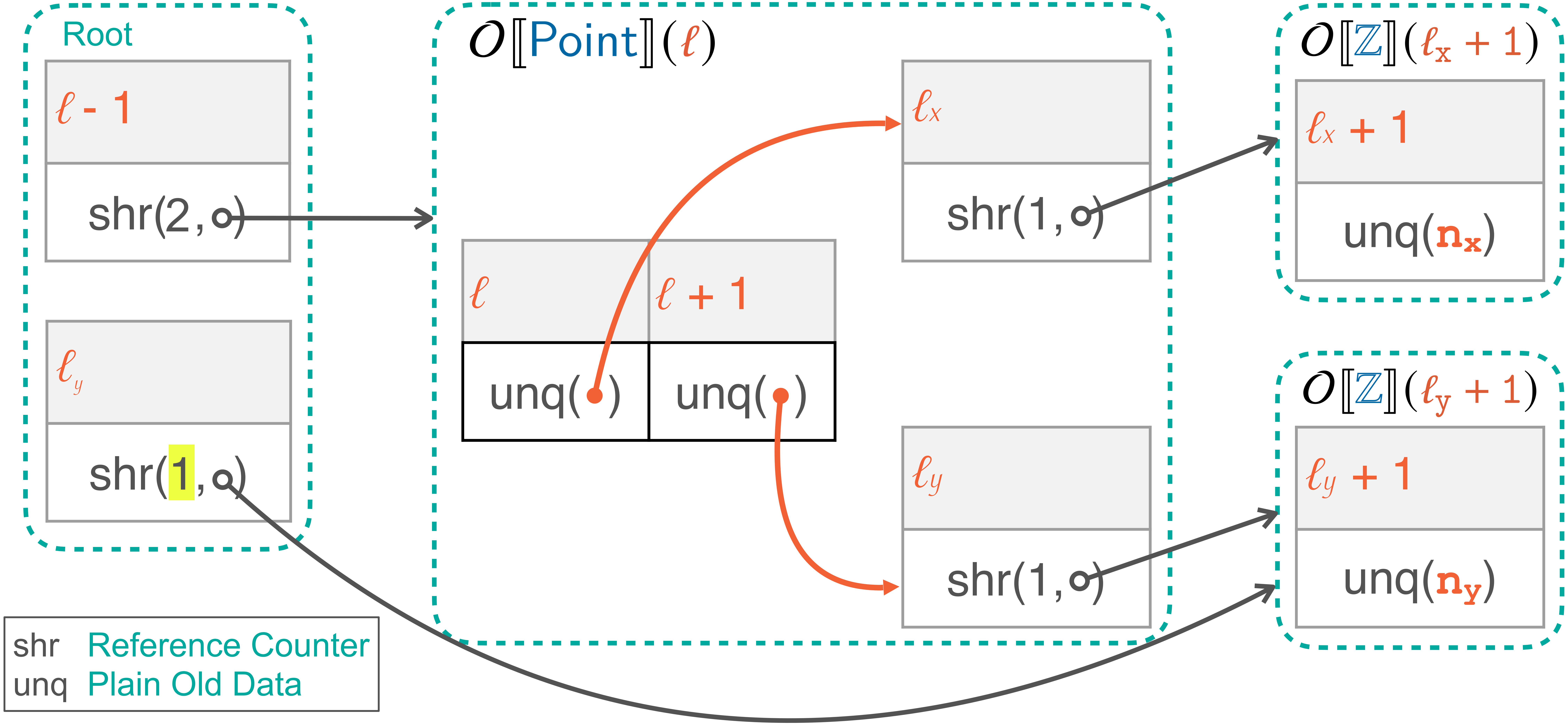
$\ell_y + 1$

unq($\mathtt{n_y}$)

shr	Reference Counter
unq	Plain Old Data

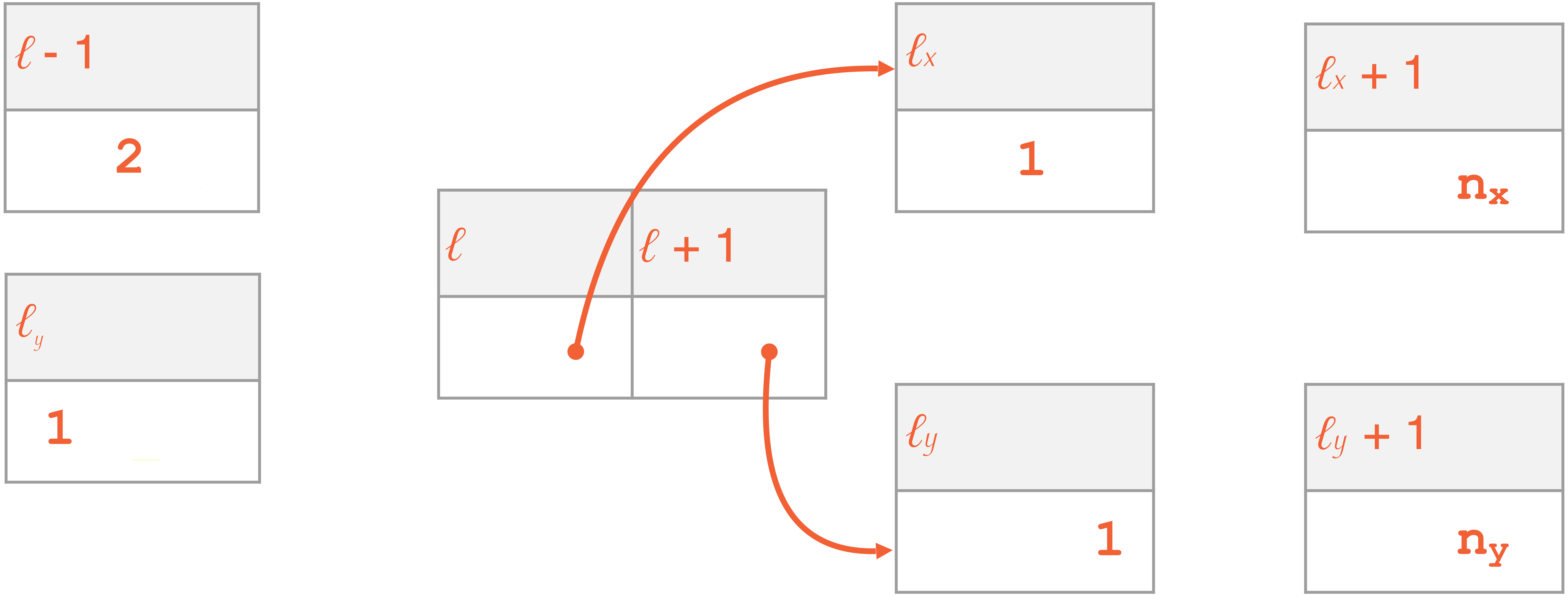
Resource Graphs

$++(\ell-1)\,;\ \boxed{++\ell_y}$



shr	Reference Counter
unq	Plain Old Data

Resource Graphs



ℓ - 1

2

ℓ_y

1

ℓ

$\ell + 1$

ℓ_x

1

ℓ_y

1

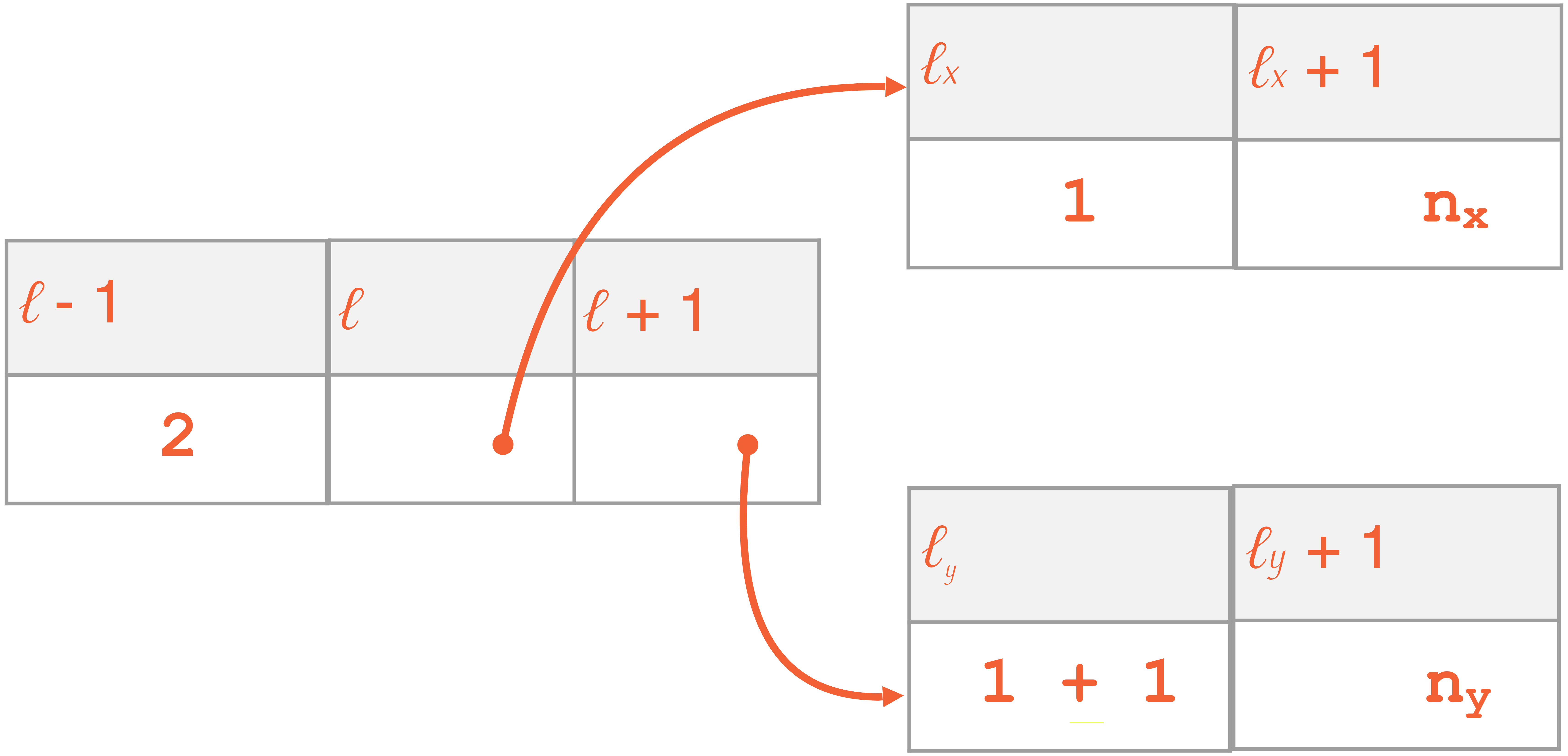
$\ell_x + 1$

$\mathbf{n_x}$

$\ell_y + 1$

$\mathbf{n_y}$

Resource Graphs



$\ell - 1$

2

ℓ

$\ell + 1$

ℓ_x

1

$\ell_x + 1$

$\mathbf{n_x}$

ℓ_y

$1 + 1$

$\ell_y + 1$

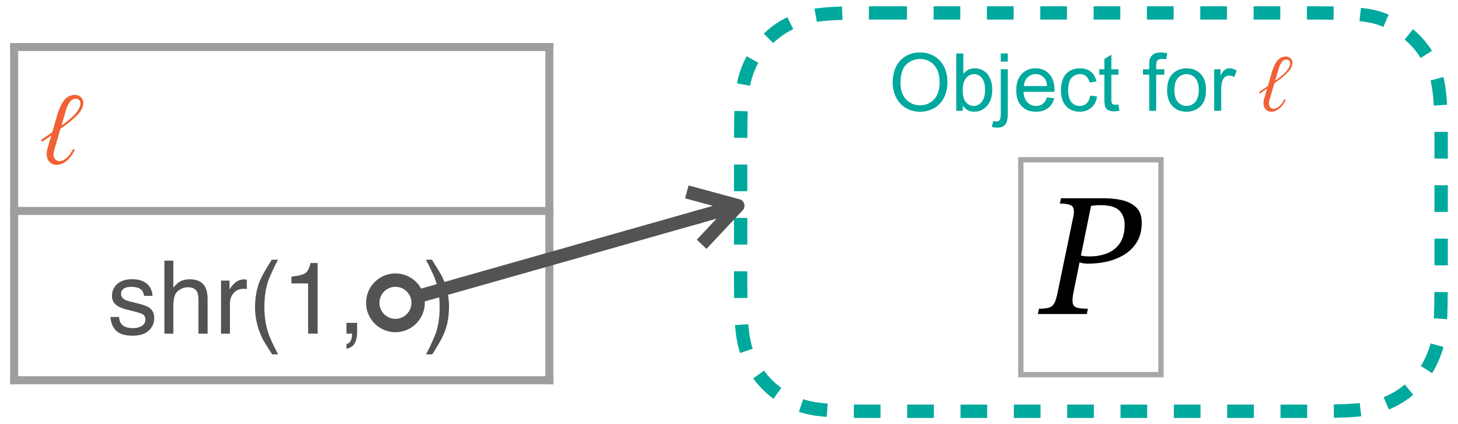
$\mathbf{n_y}$

Modalities

Modalities

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

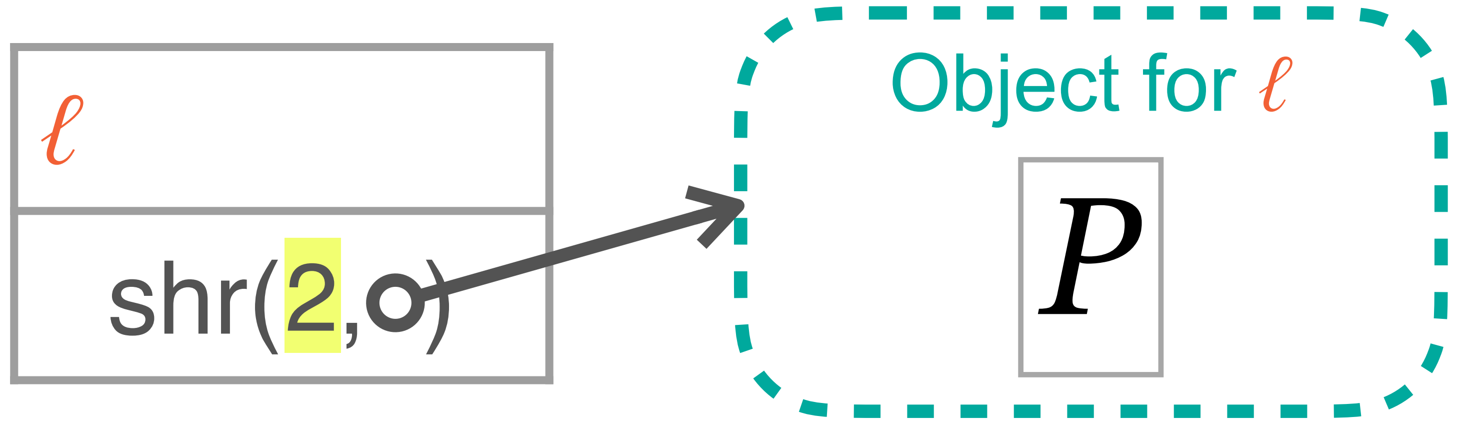
$$@_{\ell}\, P \approx$$



Modalities

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

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

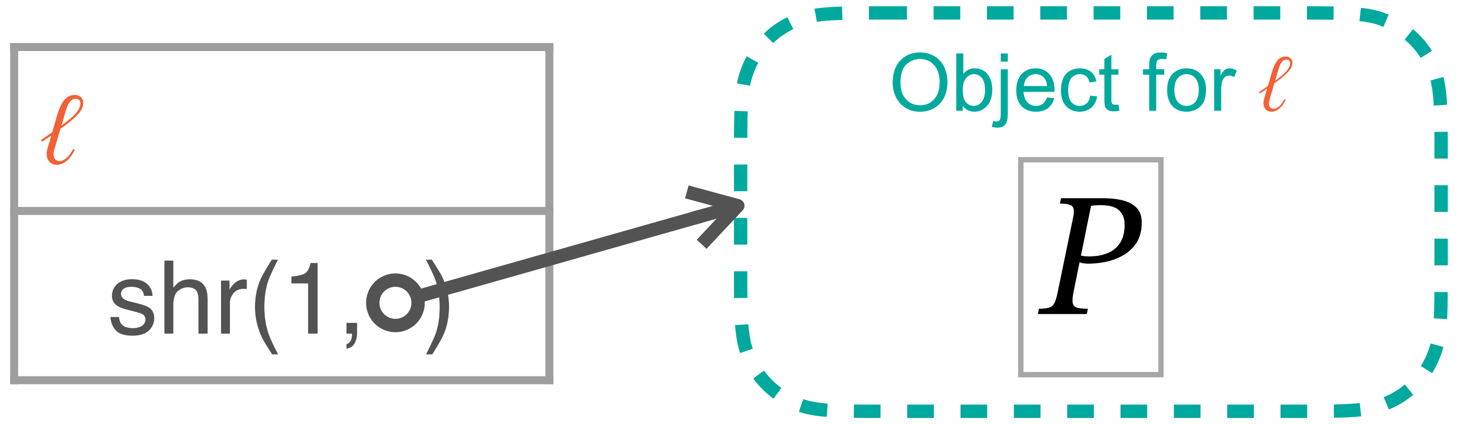


Composition sums counters at the root, not in objects

Modalities

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

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

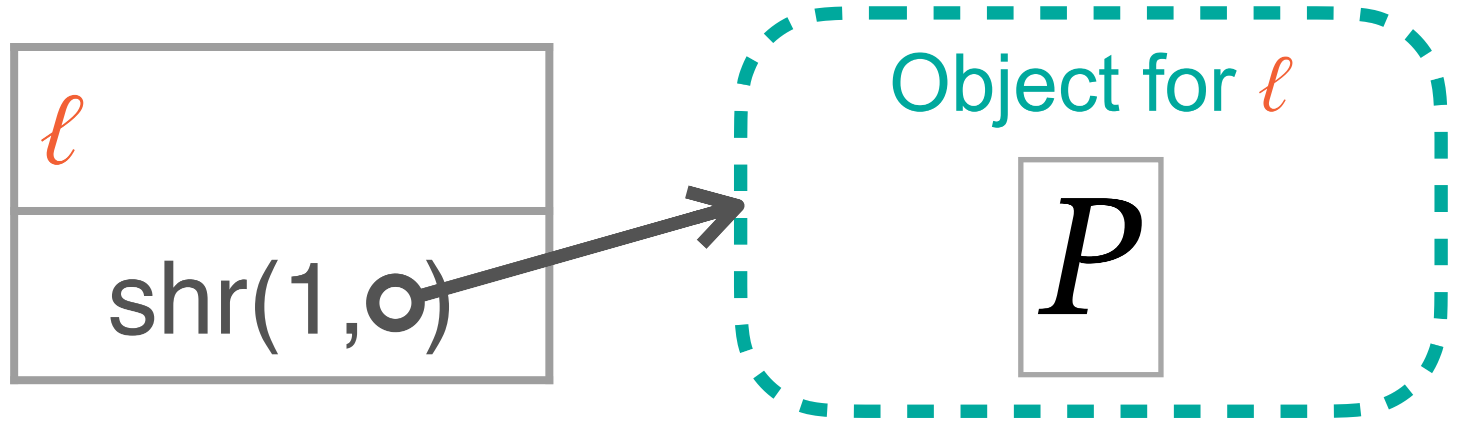


$$\boxed{\mathcal{R}[\![T]\!](\ell) \triangleq @_{\ell}\, \mathcal{O}[\![T]\!](\ell + 1)}$$

Modalities

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

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



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

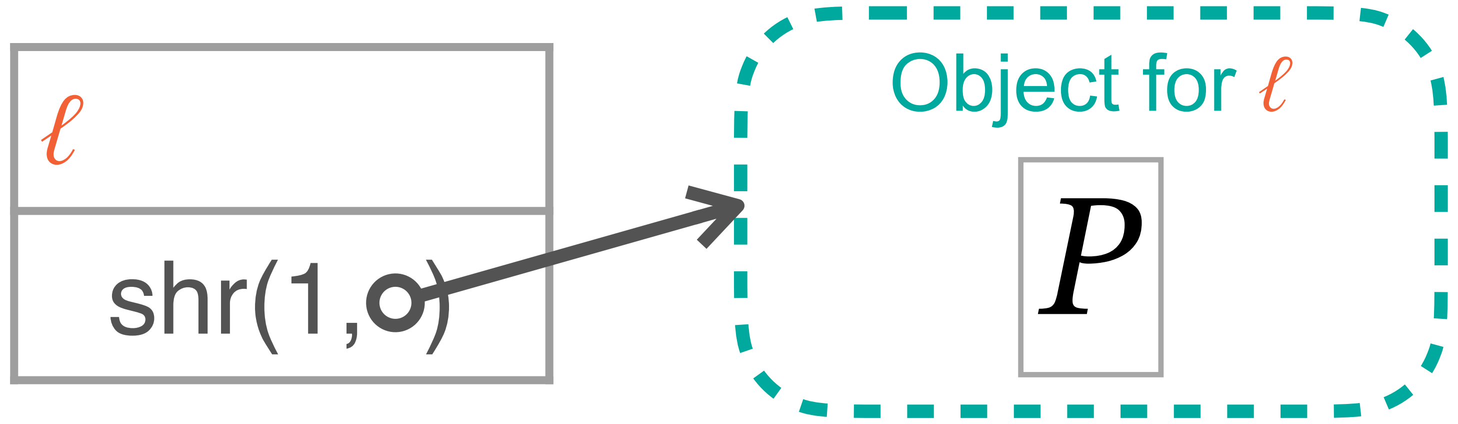
Reachability Modality $\diamond P$: It is possible to reach P via some set of jumps

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 set of jumps

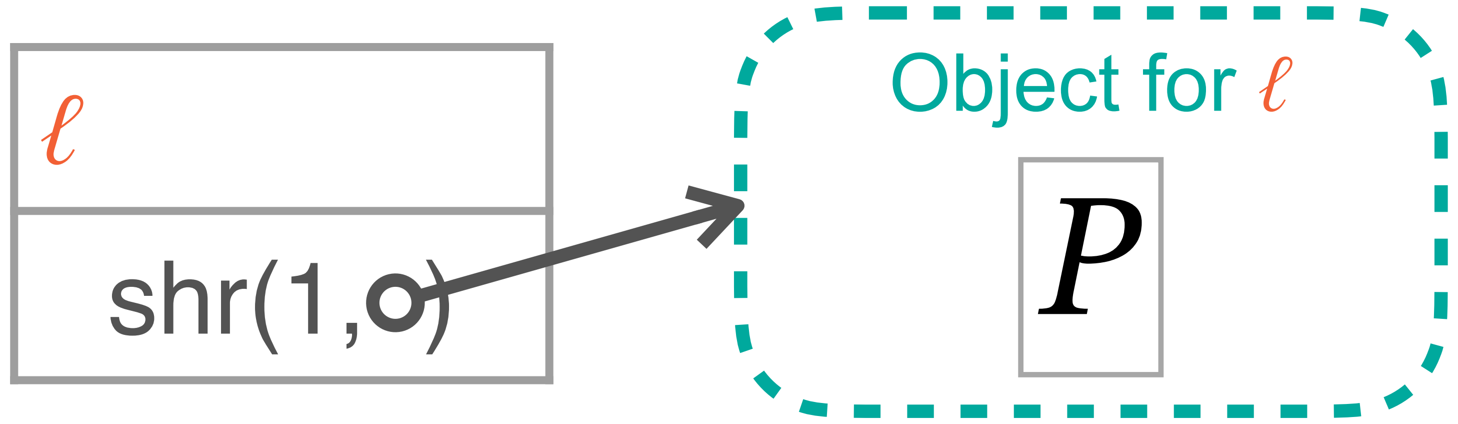
Allows reading and incrementing from deeply nested objects

$$\boxed{\begin{array}{l}\text{@-INCR}\\ \dfrac{}{\{ @_\ell P \} \mathbin{{+}{+}}\ell \{ n. \ulcorner n > 1 \urcorner \star @_\ell P \star @_\ell P \}}\end{array}}$$

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 set of jumps

Allows reading and incrementing from deeply nested objects

@-INCR-◇

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

Rigid Layout

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

Like C ABI

Rigid Layout

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

Like C ABI

No reordering

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

Rigid Layout

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

Like C ABI

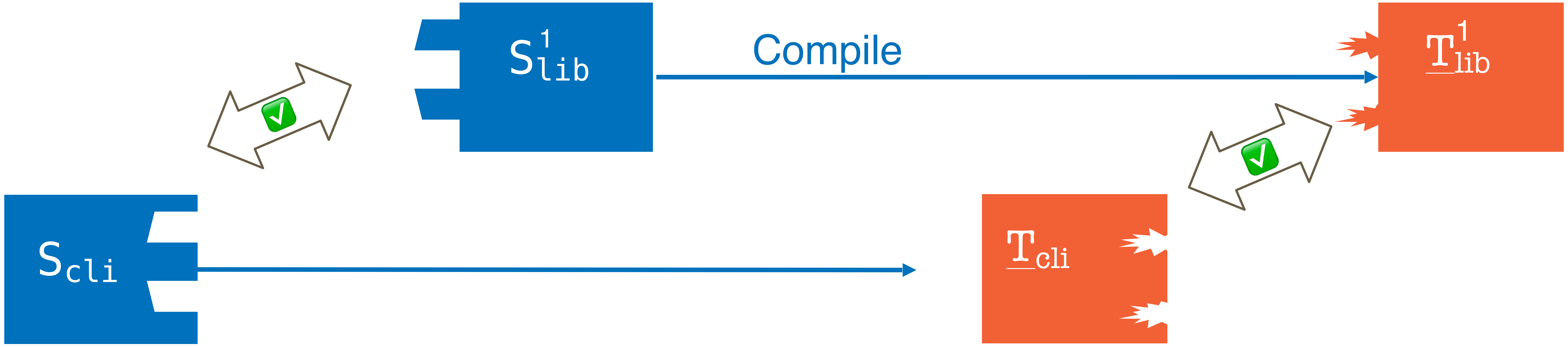
No reordering

$$\text{struct Point}\ \{x:\mathbb{Z}, y:\mathbb{Z}\} \overset{\text{upd}}{\nRightarrow} \text{struct Point}\ \{y:\mathbb{Z}, x:\mathbb{Z}\}$$

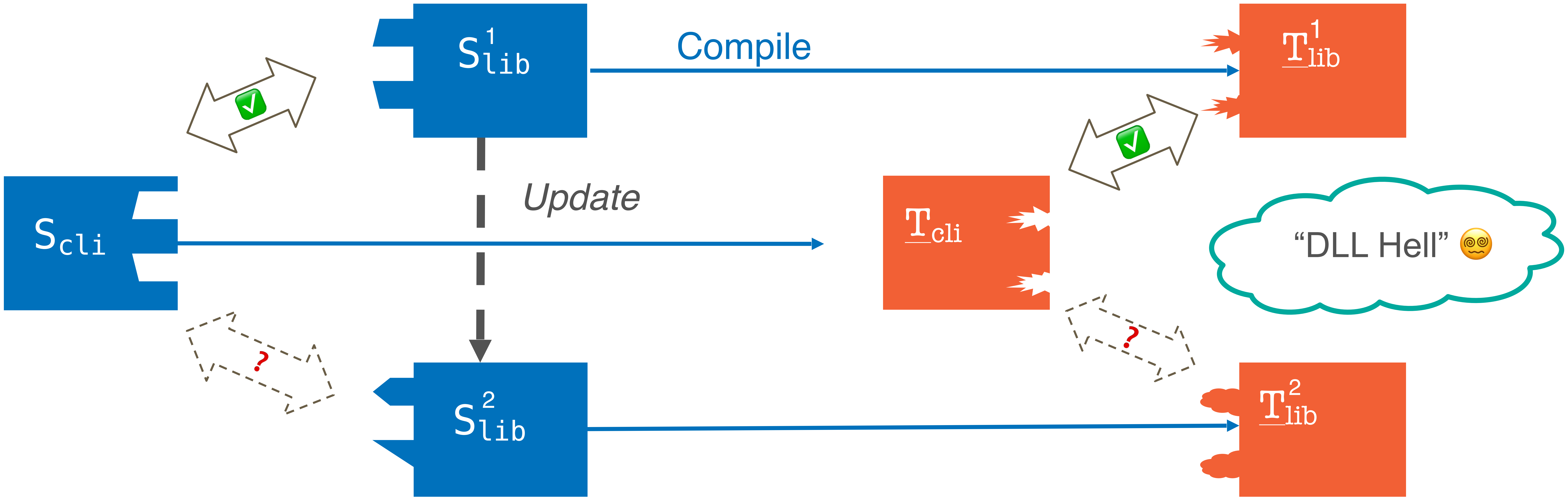
No extensibility

$$\text{struct Point}\ \{x:\mathbb{Z}, y:\mathbb{Z}\} \overset{\text{upd}}{\nRightarrow} \text{struct Point}\ \{x:\mathbb{Z}, y:\mathbb{Z}, z:\mathbb{Z}\}$$

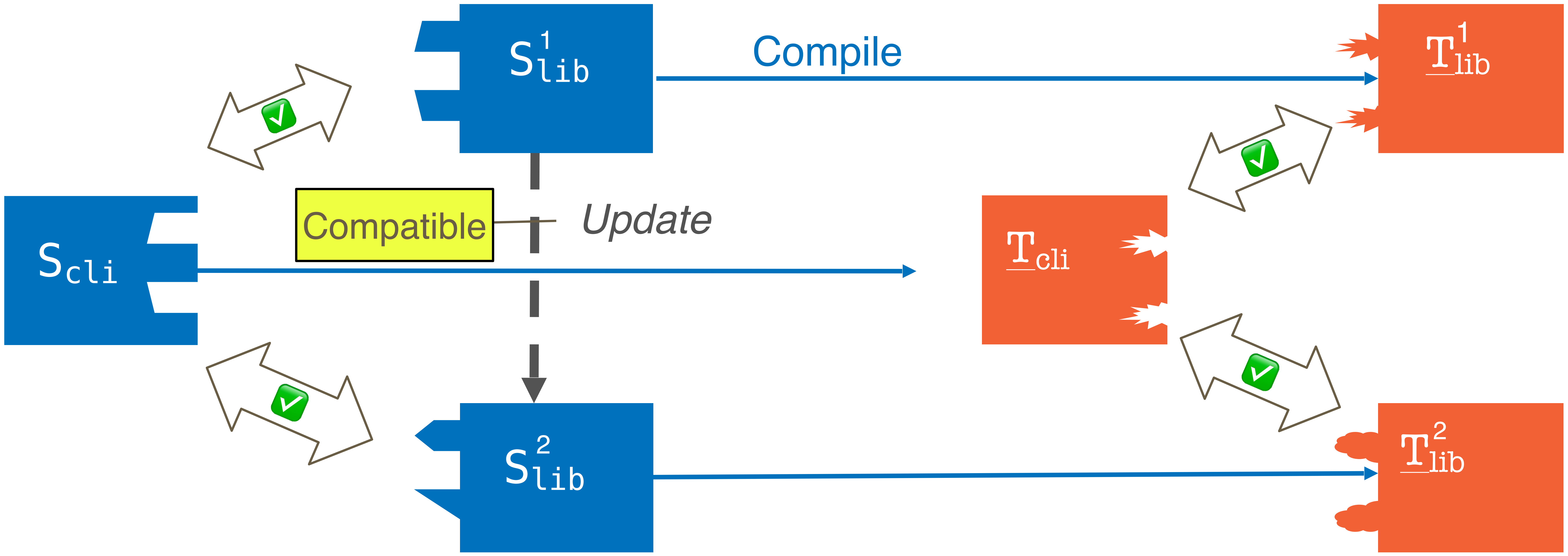
Library Evolution



Library Evolution



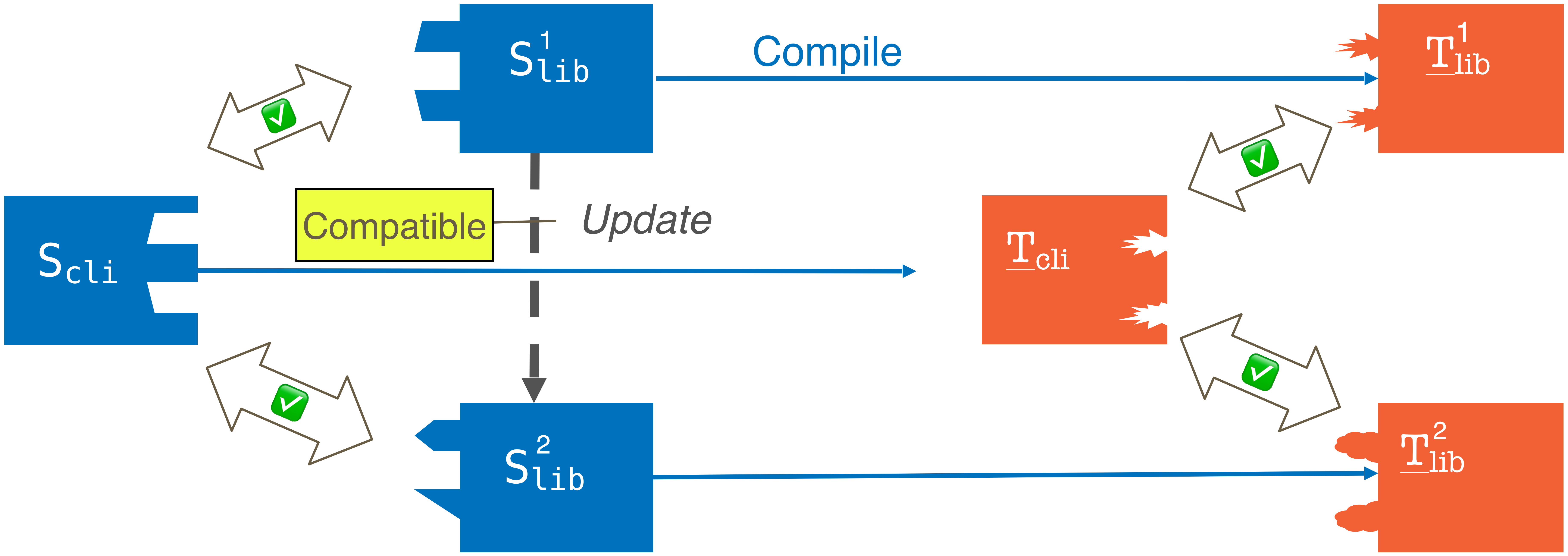
Library Evolution

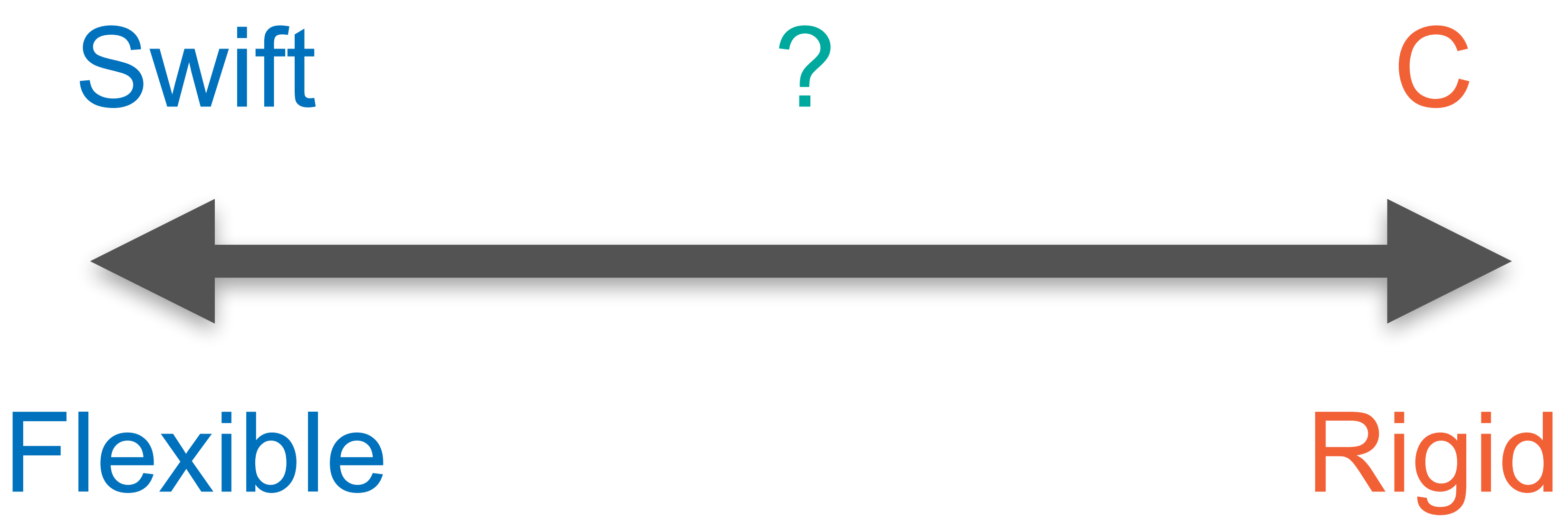


τ_2 is an **ABI compatible update** from τ_1 if

$$\llbracket \tau_2 \rrbracket \subseteq \llbracket \tau_1 \rrbracket$$

Library Evolution





τ_2 is an **ABI compatible update** from τ_1 if

$$[\![\tau_2]\!] \subseteq [\![\tau_1]\!]$$

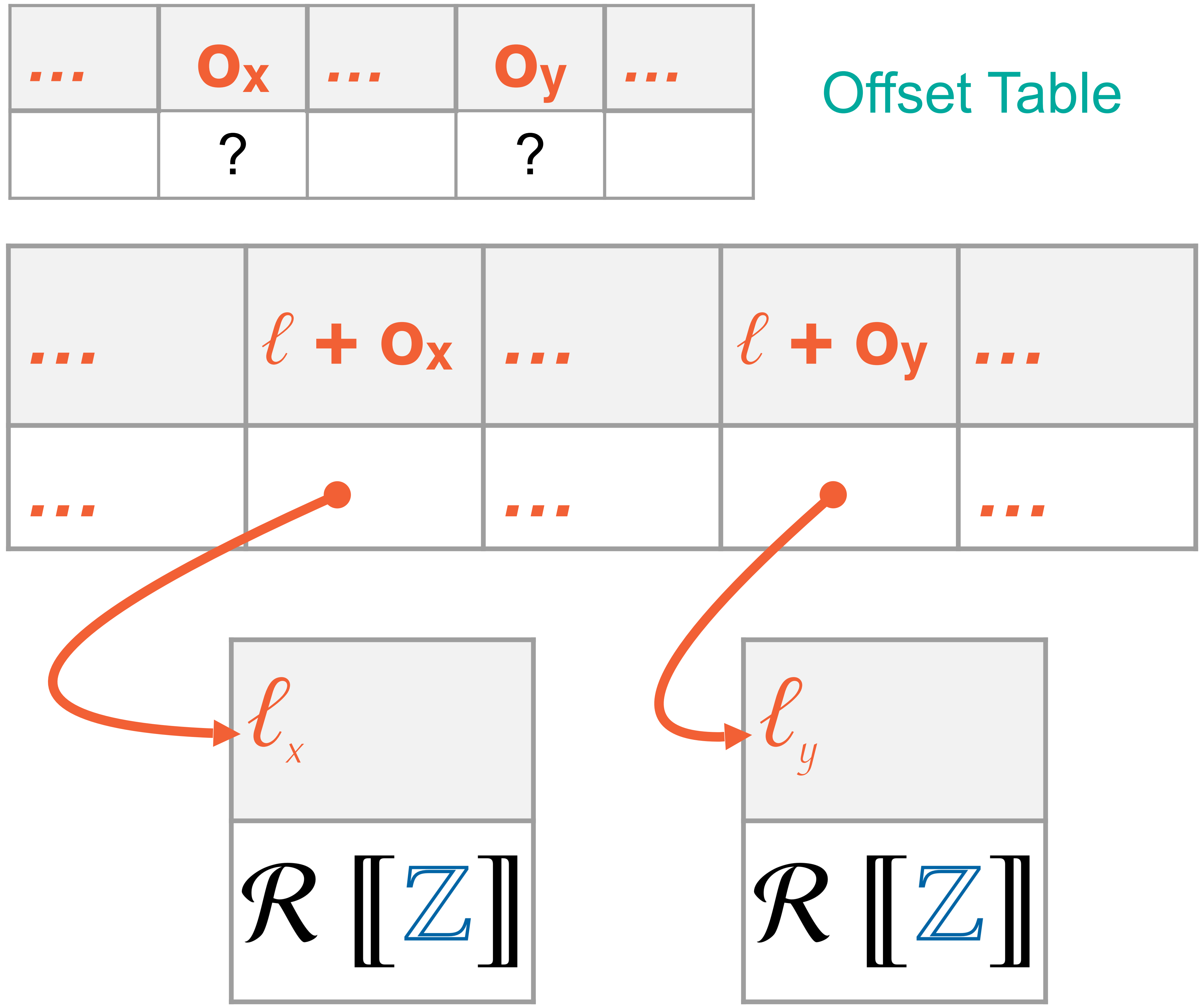
Resilient Layout

Like Swift ABI

Resilient Layout

Like Swift ABI

Client Using `Point`



Offset Table

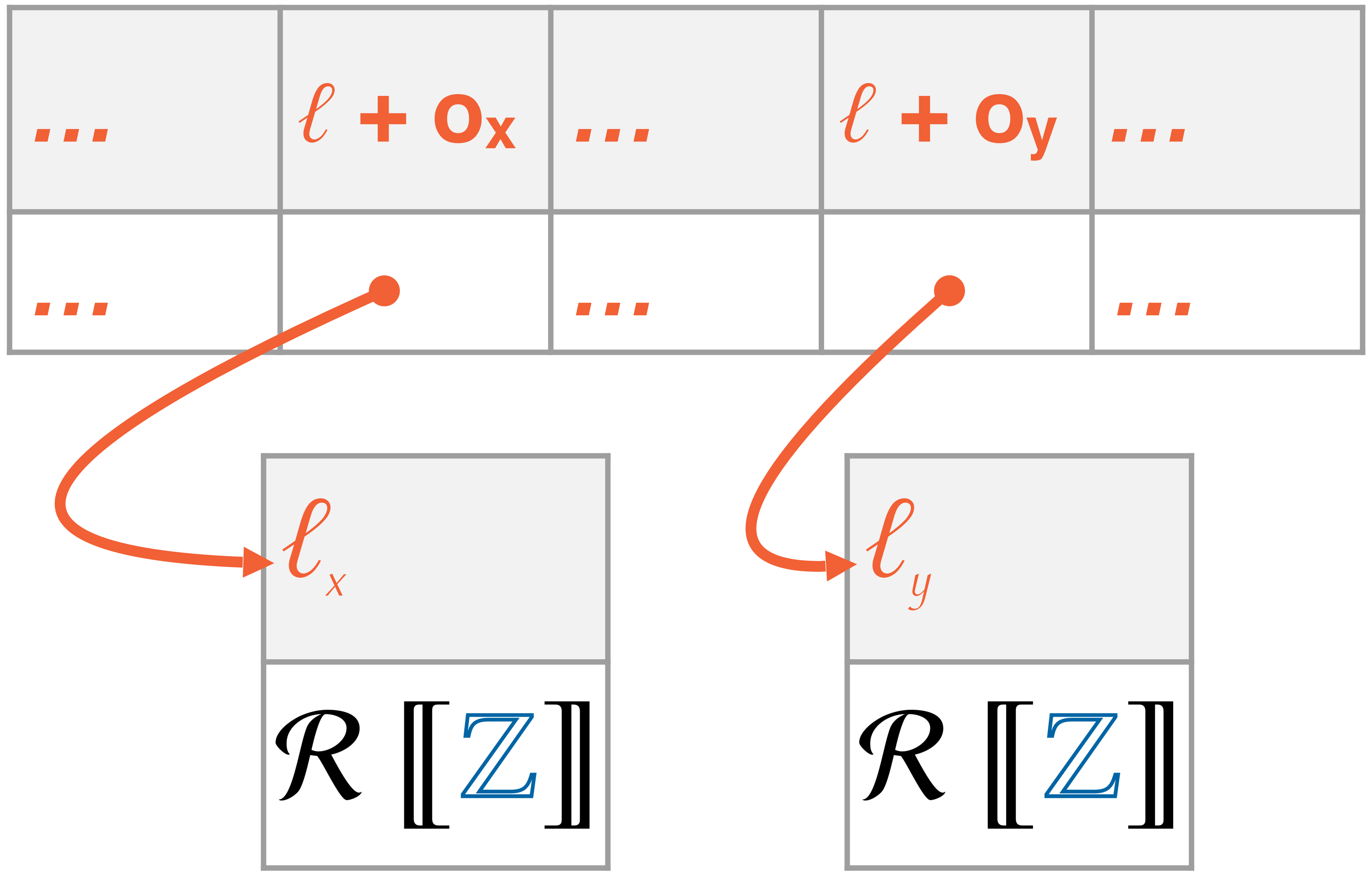
Resilient Layout

Like Swift ABI

Client Using `Point`

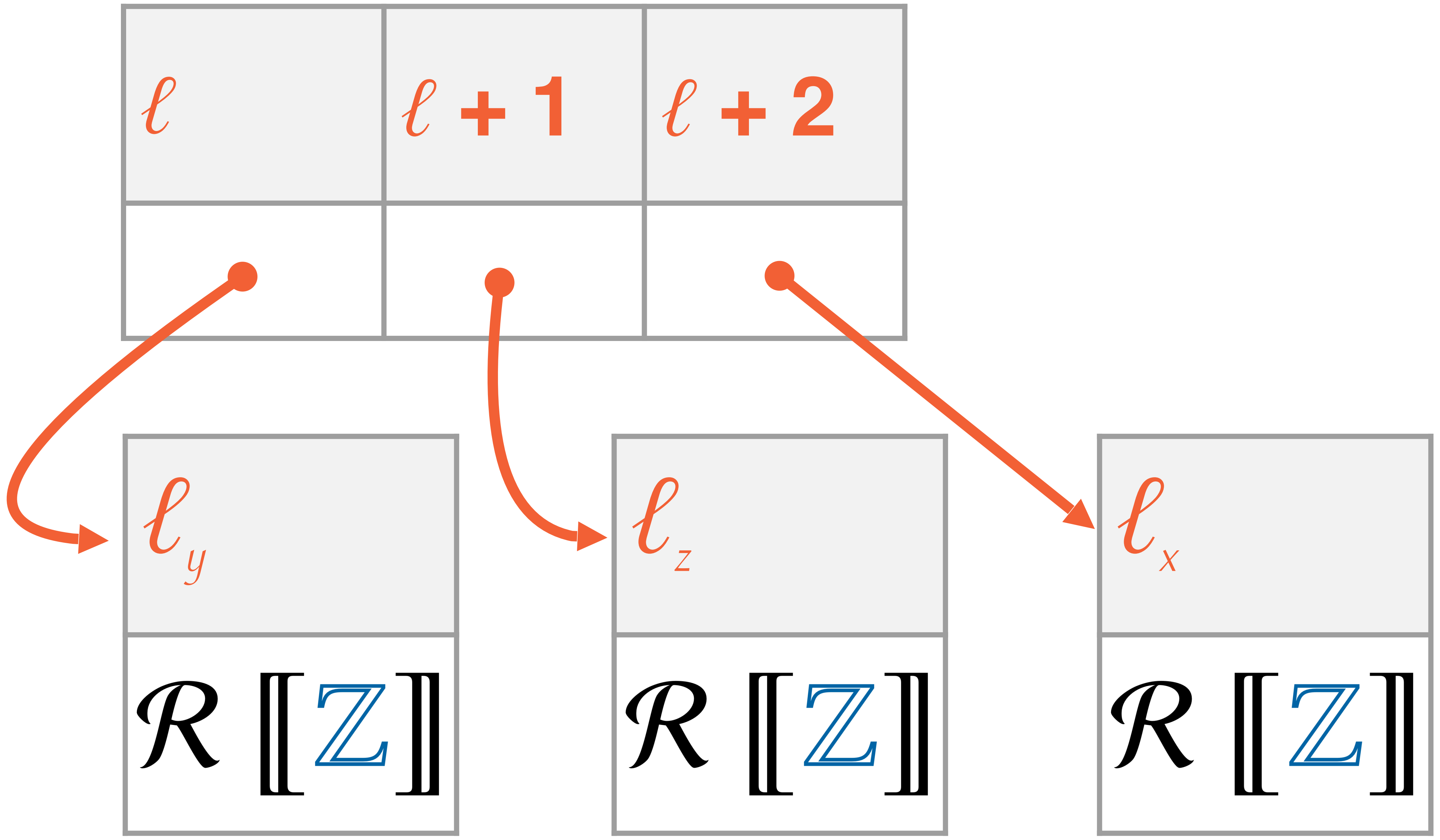
…	O_x	…	O_y	…
	?		?	

Offset Table



Library Providing `Point`

O_x	O_y	O_z
2	0	1



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

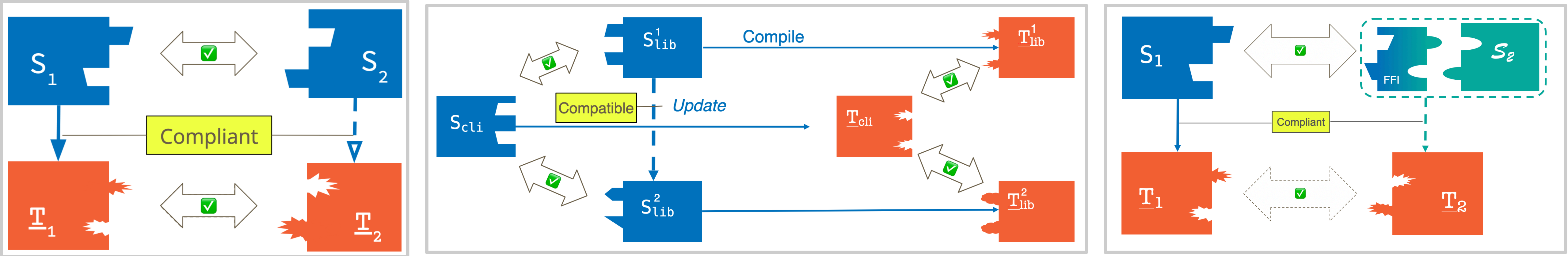
Takeaways

The Methodology

ABI Spec with Realistic Realizability

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

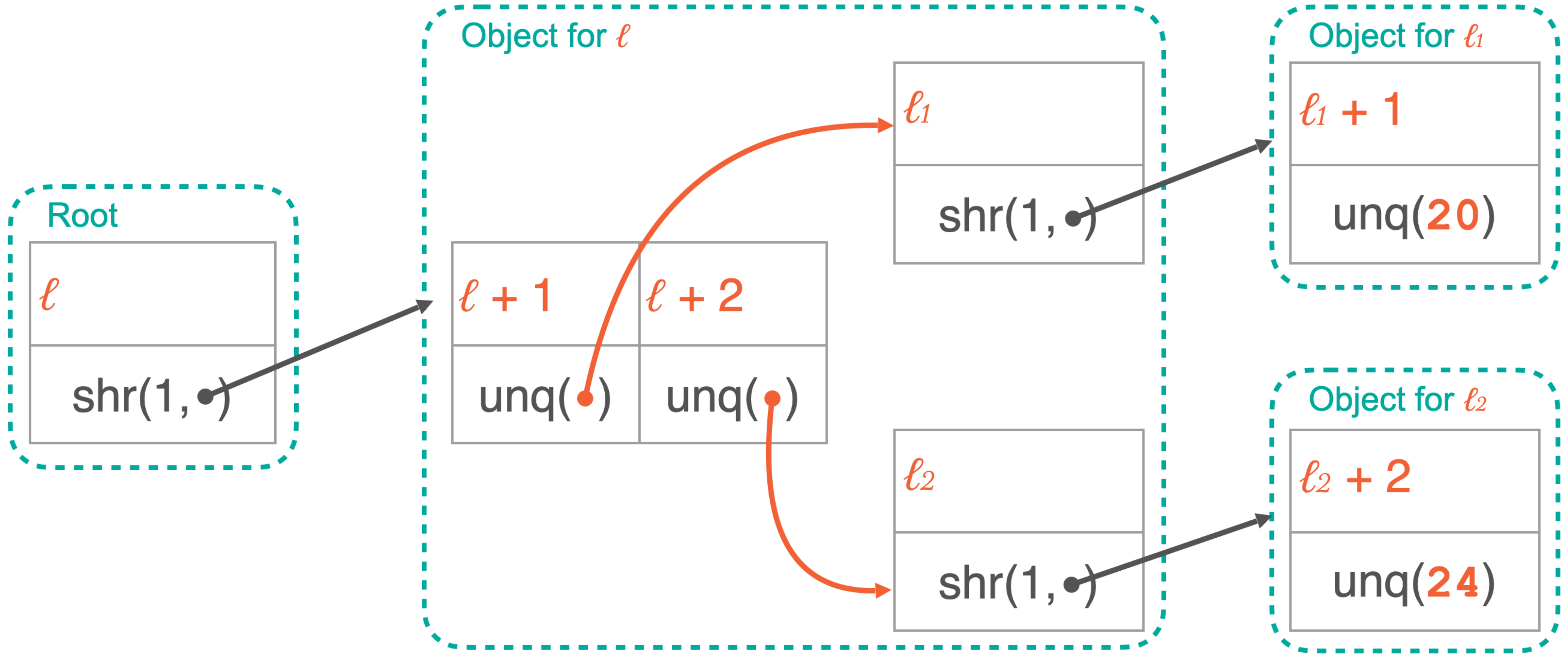
Compiler Compliance, Library Evolution, FFI Safety*



The Case Study

Graph-Based Resources for RC

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



Paper
Slides
Contact