The Specification of Application Binary Interfaces

Andrew Wagner

Northeastern University

April 30, 2025 @ CS1710, Brown University

About Me

Then

- Brown Class of 2020 😷
- Advised by Tim and Shriram

About Me

Then

- Brown Class of 2020 😷
- Advised by Tim and Shriram
- TA/HTA’d LfS 3 times

2018

2019

2020

About Me

Then

- Brown Class of 2020 😷
- Advised by Tim and Shriram
- TA/HTA'd LfS 3 times
- Team Toad

2018

2019

2020

About Me

Then

- Brown Class of 2020 😷
- Advised by Tim and Shriram
- TA/HTA'd LfS 3 times
- Team Toad

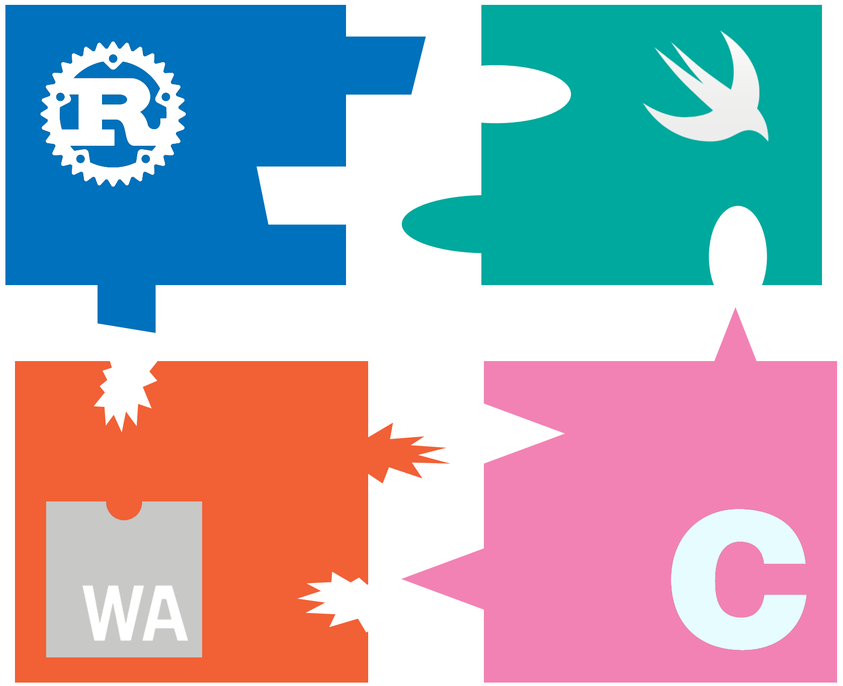
Now

- PhD student with Amal Ahmed at NEU
- Focus on the semantics of language interoperability

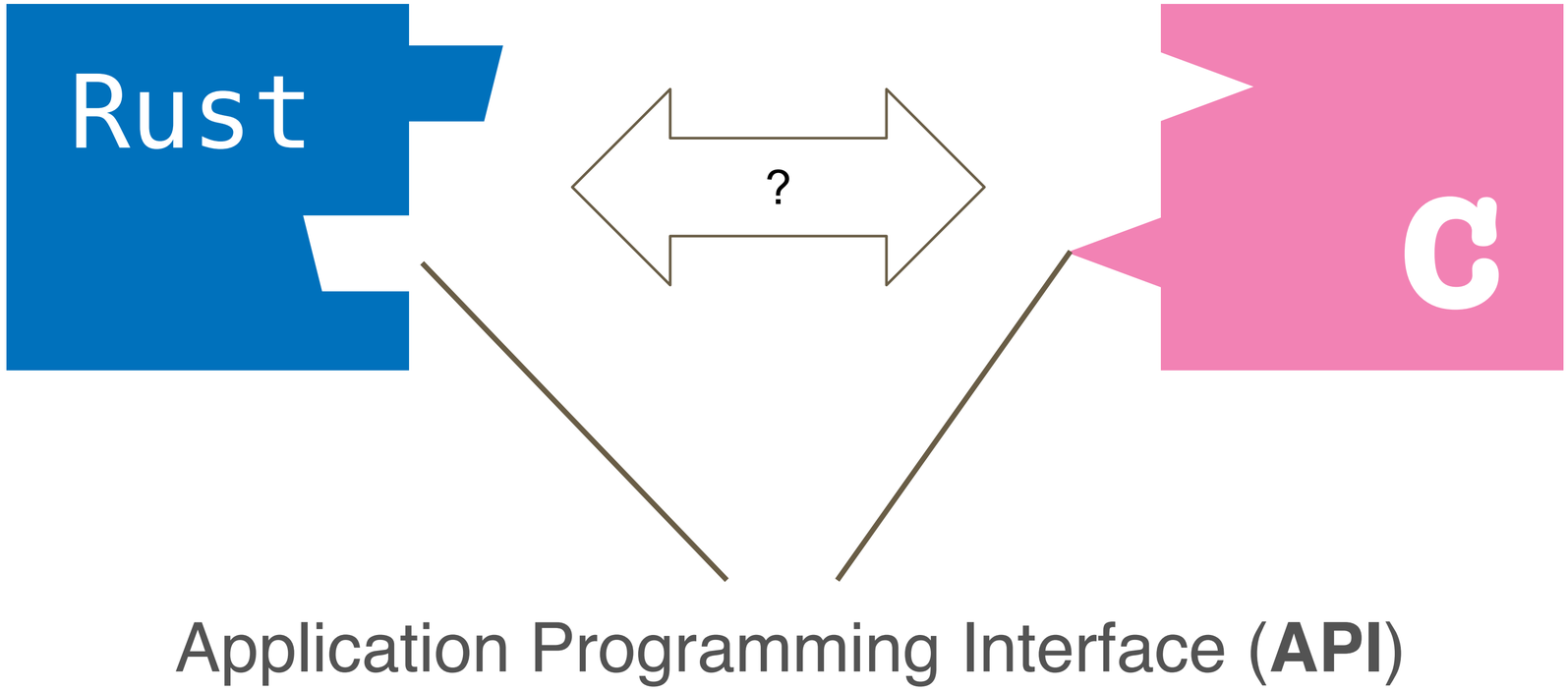
2018

2019

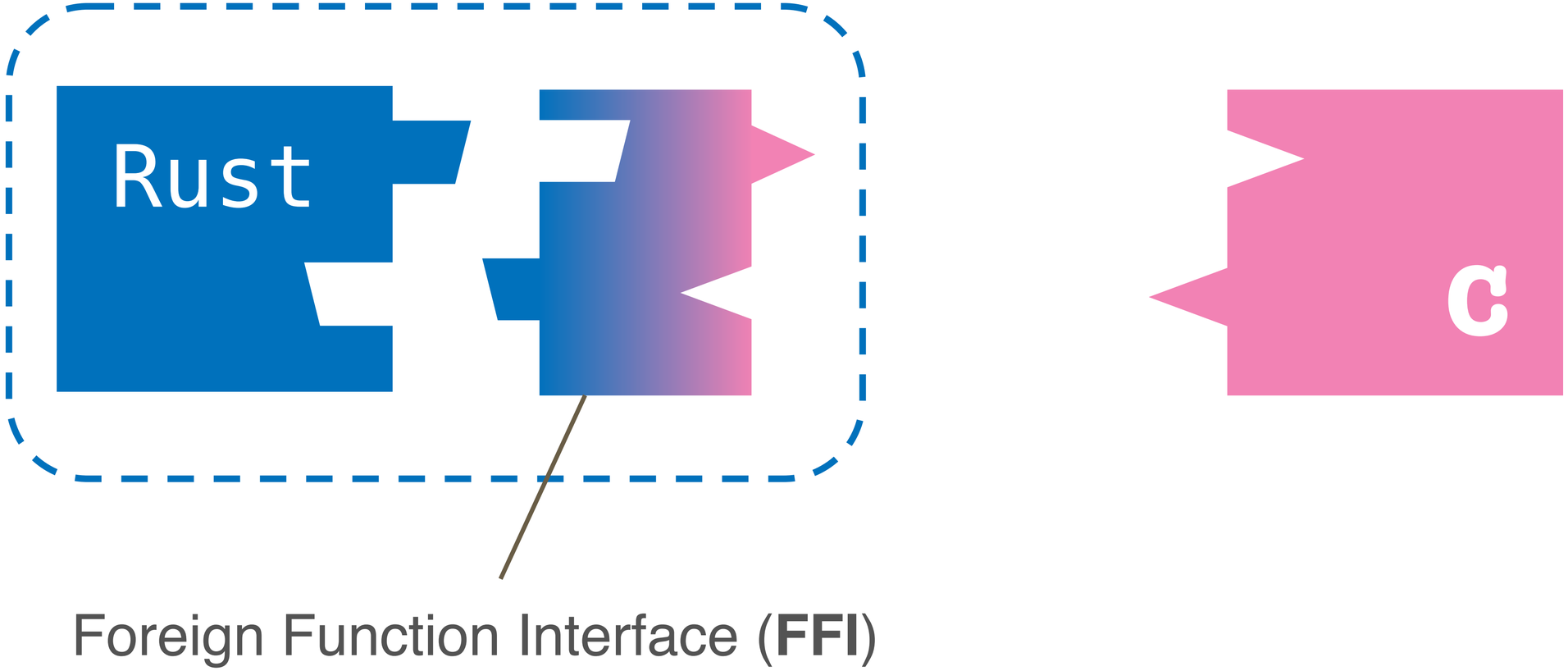
2020



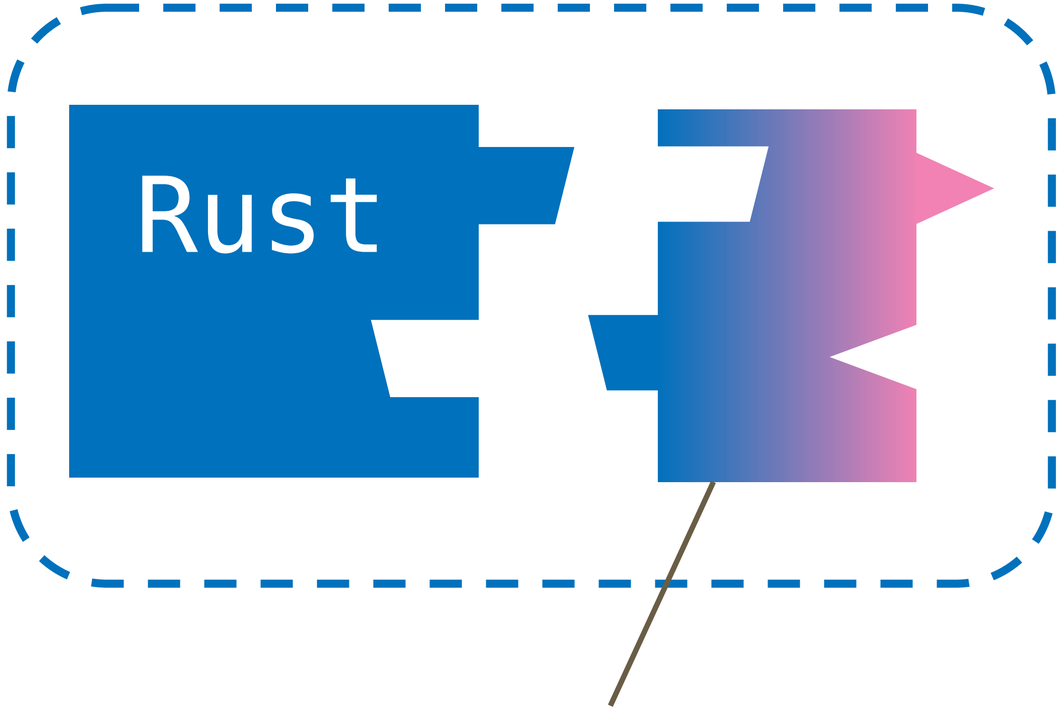
Piecing Languages Together

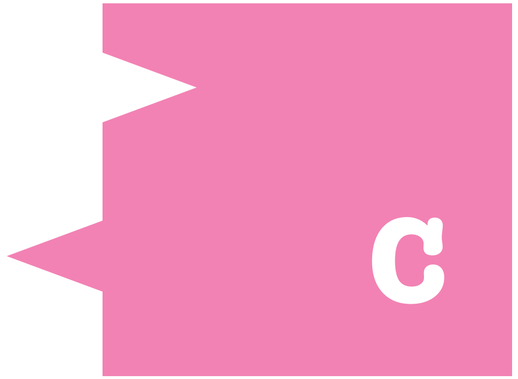


Piecing Languages Together



Piecing Languages Together





Foreign Function Interface (**FFI**)

TyDe23

Semantic Encapsulation using Linking Types

Daniel Patterson
dbp@dbpmail.net
Northeastern University
Boston, MA, USA

Andrew Wagner
Northeastern University
Boston, MA, USA
ahwagner@ccs.neu.edu

Amal Ahmed
amal@ccs.neu.edu
Northeastern University
Boston, MA, USA

Abstract

Interoperability pervades nearly all mainstream language implementations, as most systems leverage subcomponents written in different languages. And yet, such linking can expose a language to foreign behaviors that are internally inexpressible, which poses a serious threat to safety invariants and programmer reasoning. To preserve such invariants, a language may try to add features to limit the reliance on external libraries, but endless extensions can obscure the core abstractions the language was designed to provide.

ACM Reference Format:
Daniel Patterson, Andrew Wagner, and Amal Ahmed. 2023. Semantic Encapsulation using Linking Types. In *Proceedings of the 8th ACM SIGPLAN International Workshop on Type-Driven Development (TyDe '23), September 4, 2023, Seattle, WA, USA.* ACM, New York, NY, USA, 15 pages. https://doi.org/10.1145/3609027.3609405

1 Introduction

Languages cannot exist in isolation. Foreign function inter-

Crash Course: Memory Safety

Manually managed memory

- C does not guarantee safety

```
int *x = malloc(sizeof(int));
*x = 42;
free(x);
return *x;
// SIGSEGV: Seg. fault
```

Crash Course: Memory Safety

Manually managed memory

- C does not guarantee safety
- Rust guarantees safety using fancy types

```
int *x = malloc(sizeof(int));
*x = 42;
free(x);
return *x;
// SIGSEGV: Seg. fault
```

Crash Course: Memory Safety

Manually managed memory

- C does not guarantee safety
- Rust guarantees safety using fancy types

```
int *x = malloc(sizeof(int));
*x = 42;
free(x);
return *x;
// SIGSEGV: Seg. fault
```

Automatically managed memory

- Reference counting (e.g., Swift)
- Garbage collection (e.g., OCaml)

```
let x = ref 42 in !x
// no explicit free
```

Crash Course: Memory Safety

Manually managed memory

- C does not guarantee safety
- Rust guarantees safety using fancy types

```
int *x = malloc(sizeof(int));
*x = 42;
free(x);
return *x;
// SIGSEGV: Seg. fault
```

Automatically managed memory

- Reference counting (e.g., Swift)
- Garbage collection (e.g., OCaml)

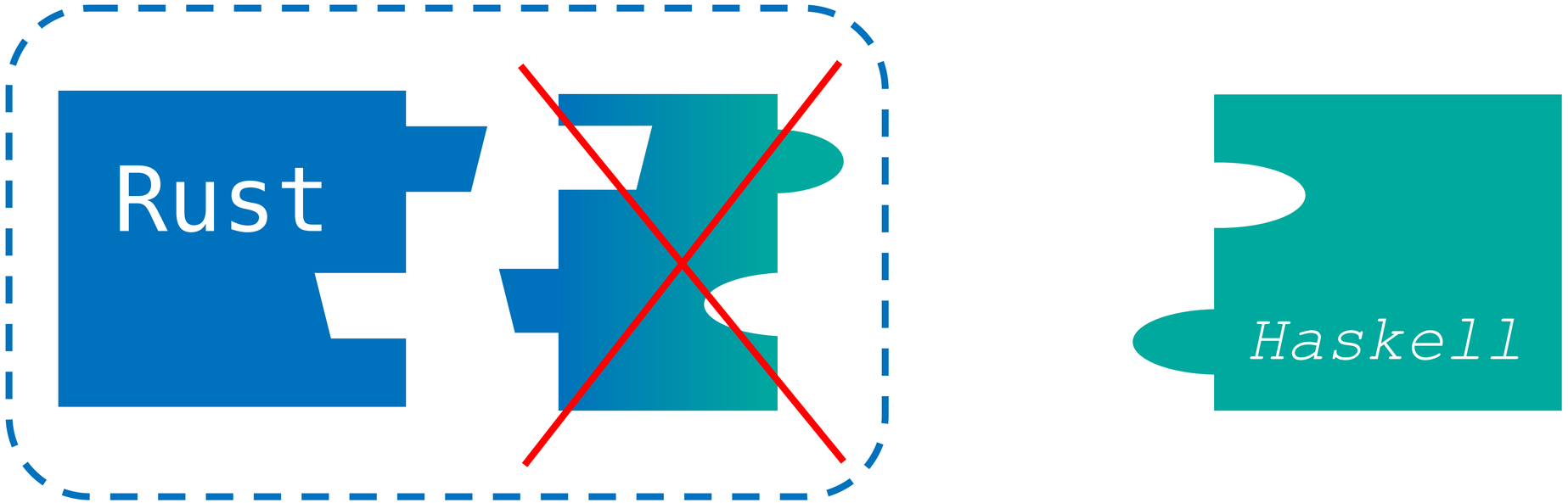
```
let x = ref 42 in !x
// no explicit free
```

No explicit memory (e.g., Haskell)

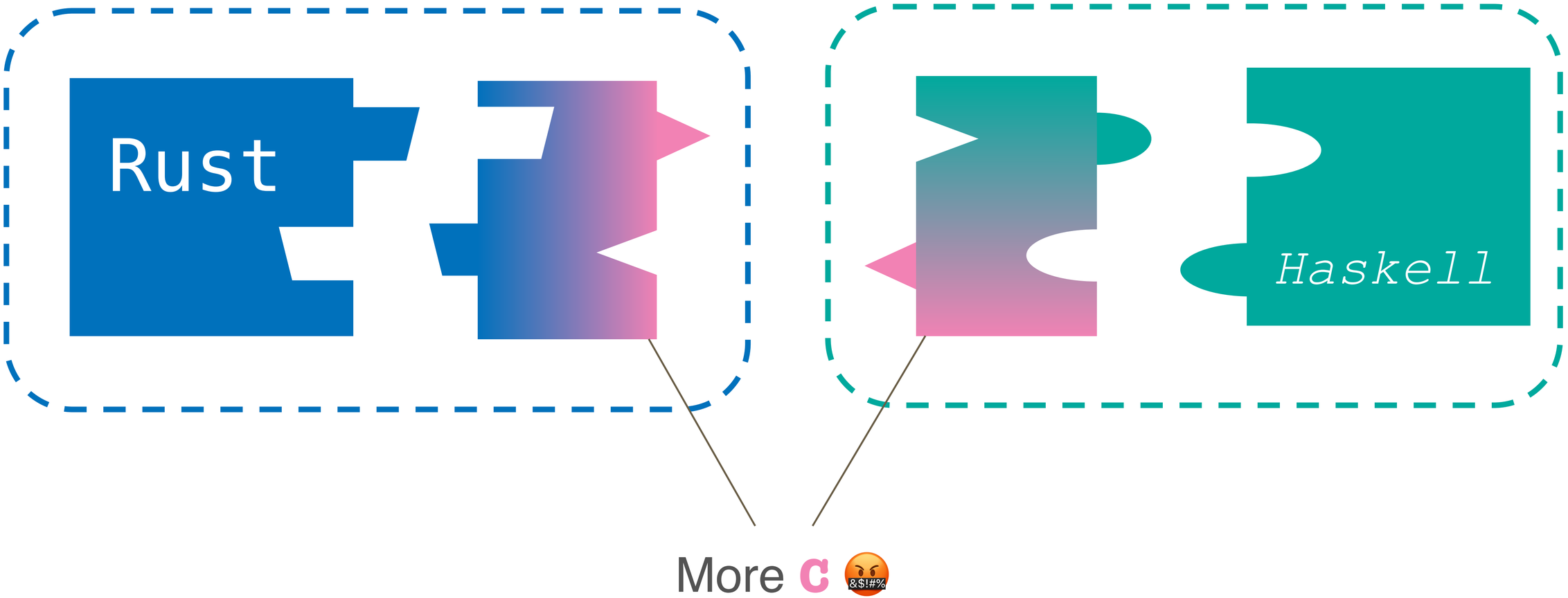
```
let x = 42 in x
```

Piecing Safe Languages Together

Piecing Safe Languages Together



Piecing Safe Languages Together in Practice



Why C?

Why C?

Because every language already “speaks C”

Why C?

Because every language already “speaks C”

But Why Does Every Language Speak C?

Why C?

Because every language already “speaks C”

But Why Does Every Language Speak C?

Because C is committed to *ABI stability*

> “The standard is haunted … by that **Three Letter Demon**. … a contract was **forged in blood**.”
> – JeanHeyd Meneide, WG14 (C/C++ Compatibility)

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?

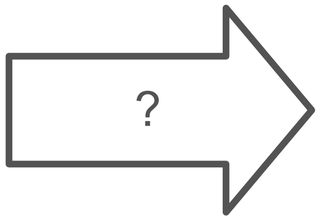
Application Binary Interface (ABI)

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

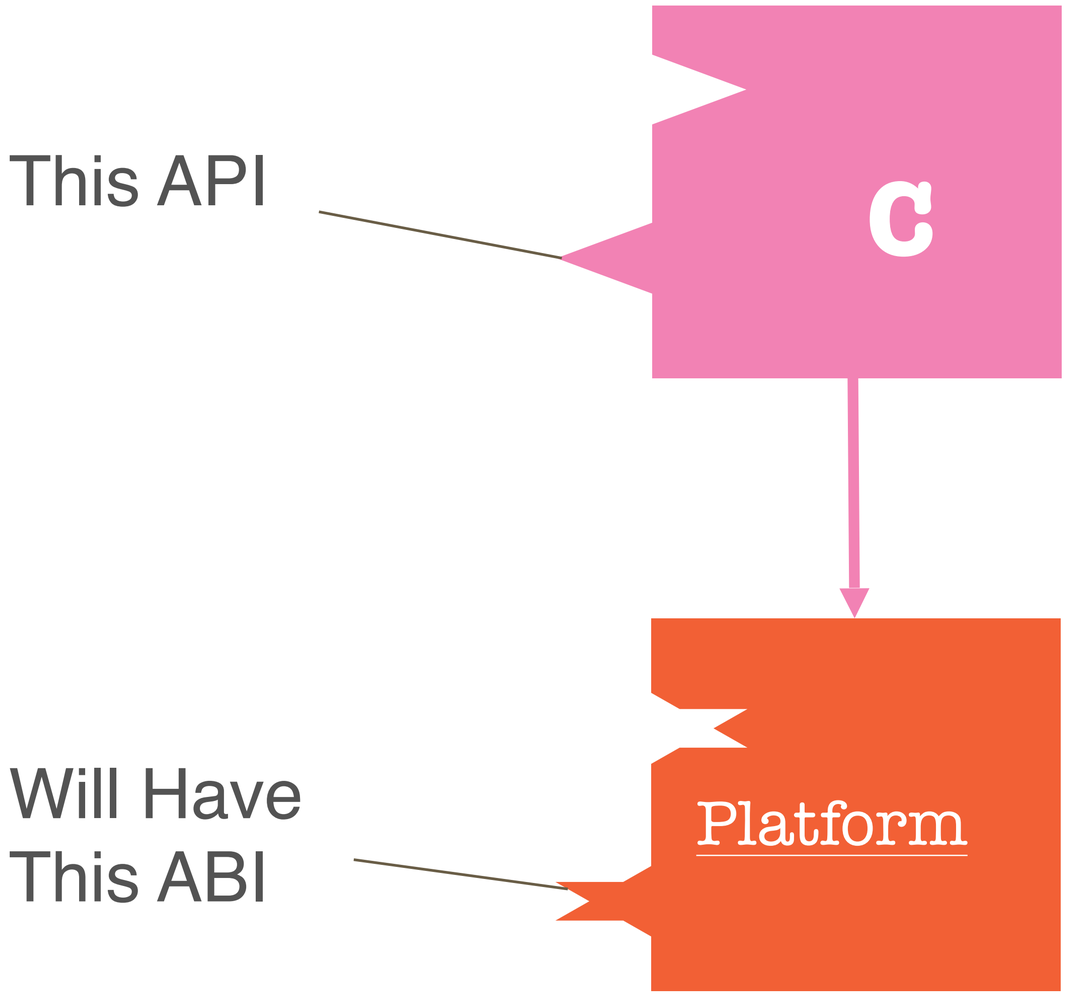
— Swift

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



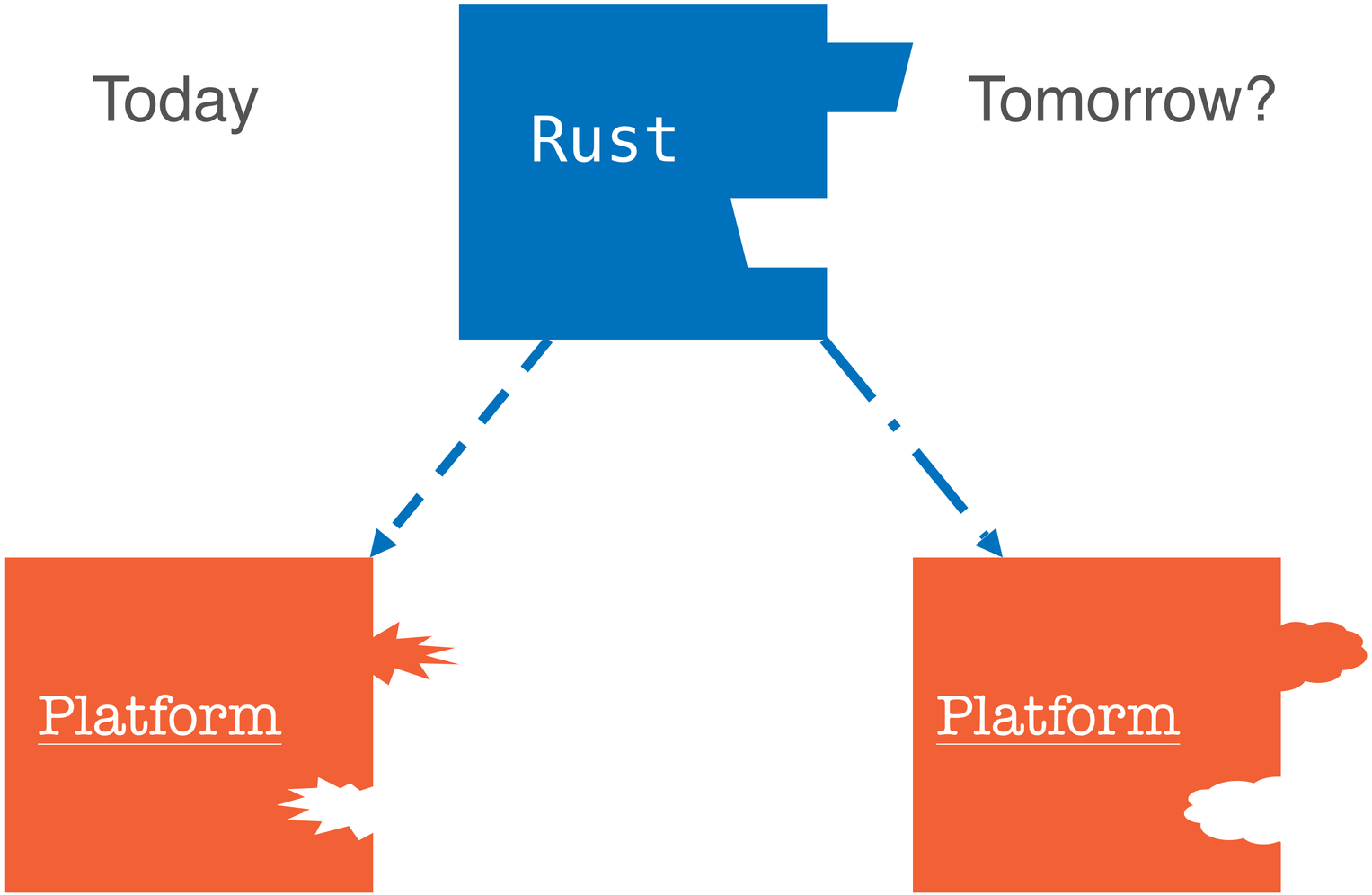

```
int foo(int fst, int snd)

int foo(int indir[])

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

ABI Stability



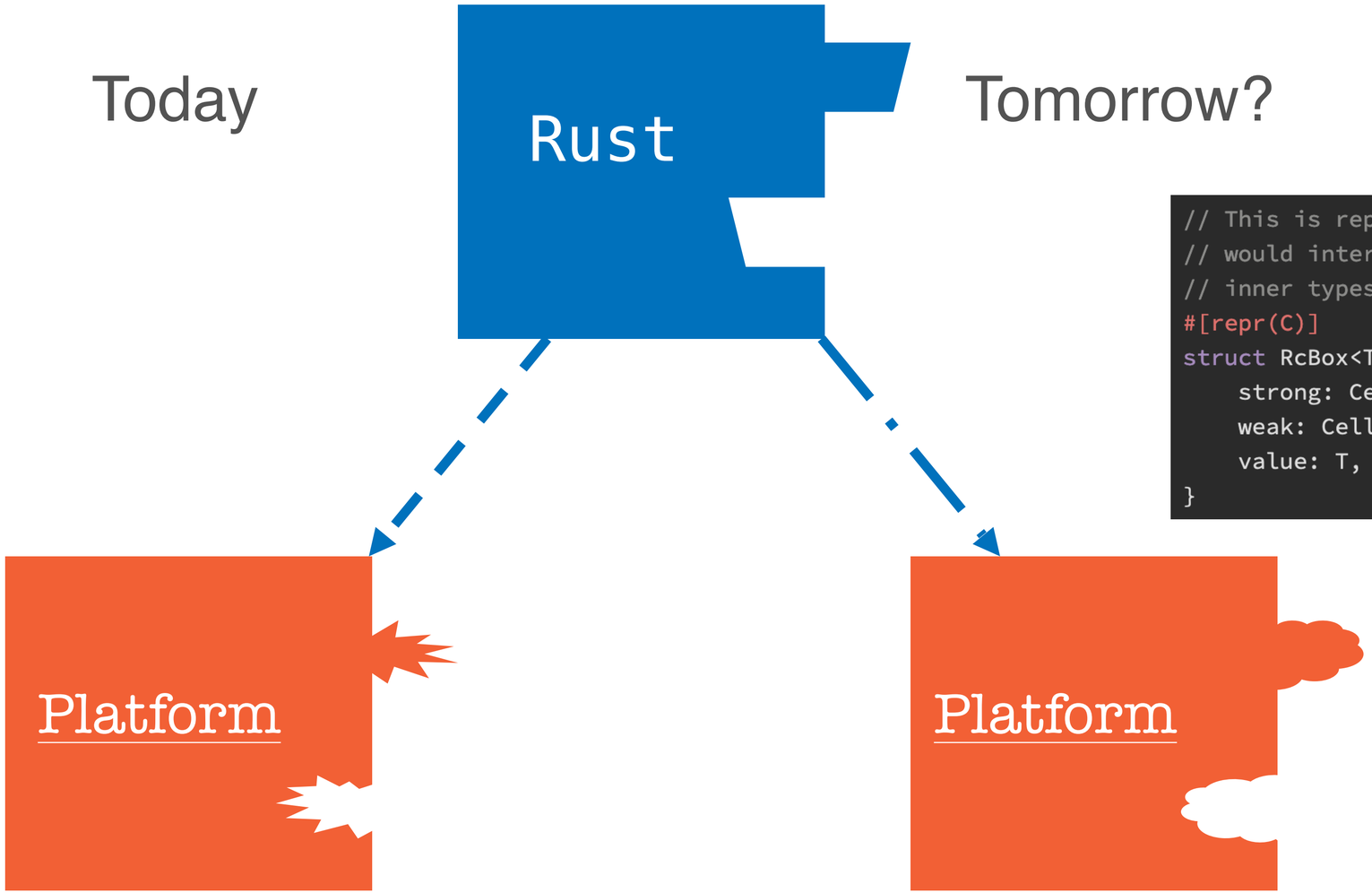
* *In Theory* 🙃

ABI Instability



ABI Instability

Today



Tomorrow?

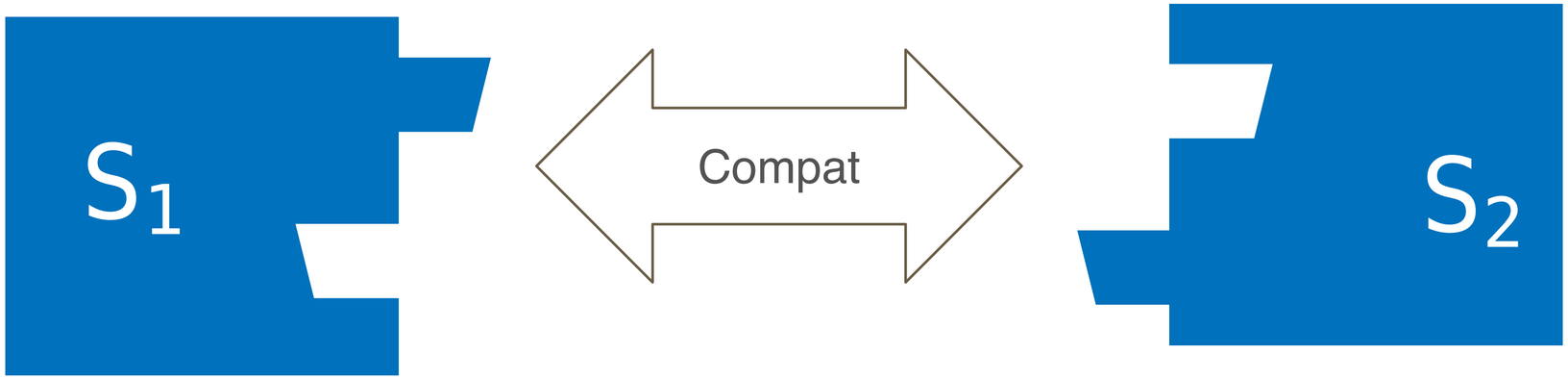
```rust
// This is repr(C) to future-proof against possible field-reordering,
// would interfere with otherwise safe [into|from]_raw() of transmuta
// inner types.
#[repr(C)]
struct RcBox<T: ?Sized> {
    strong: Cell<usize>,
    weak: Cell<usize>,
    value: T,
}
```

Rust Standard Library

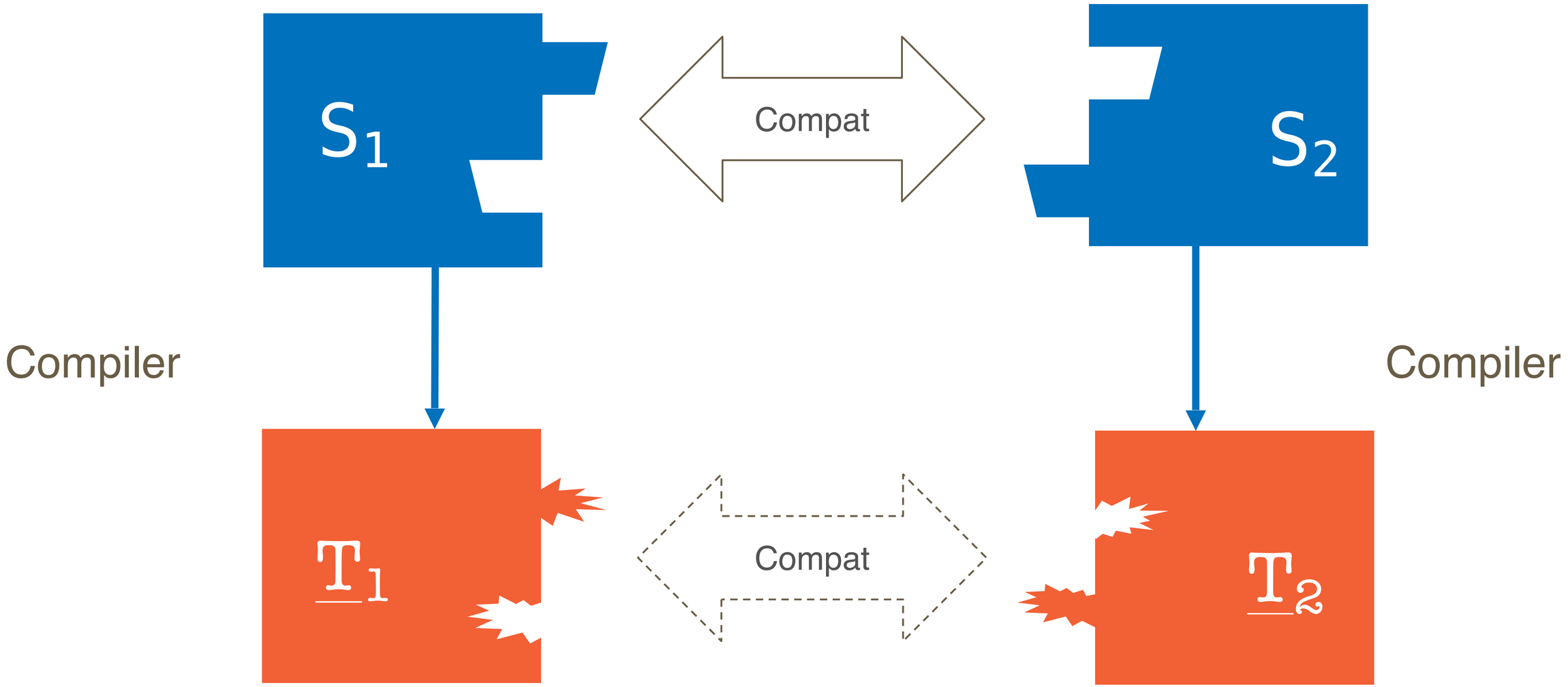
Why Use an ABI?

Why Use an ABI? Interoperability!

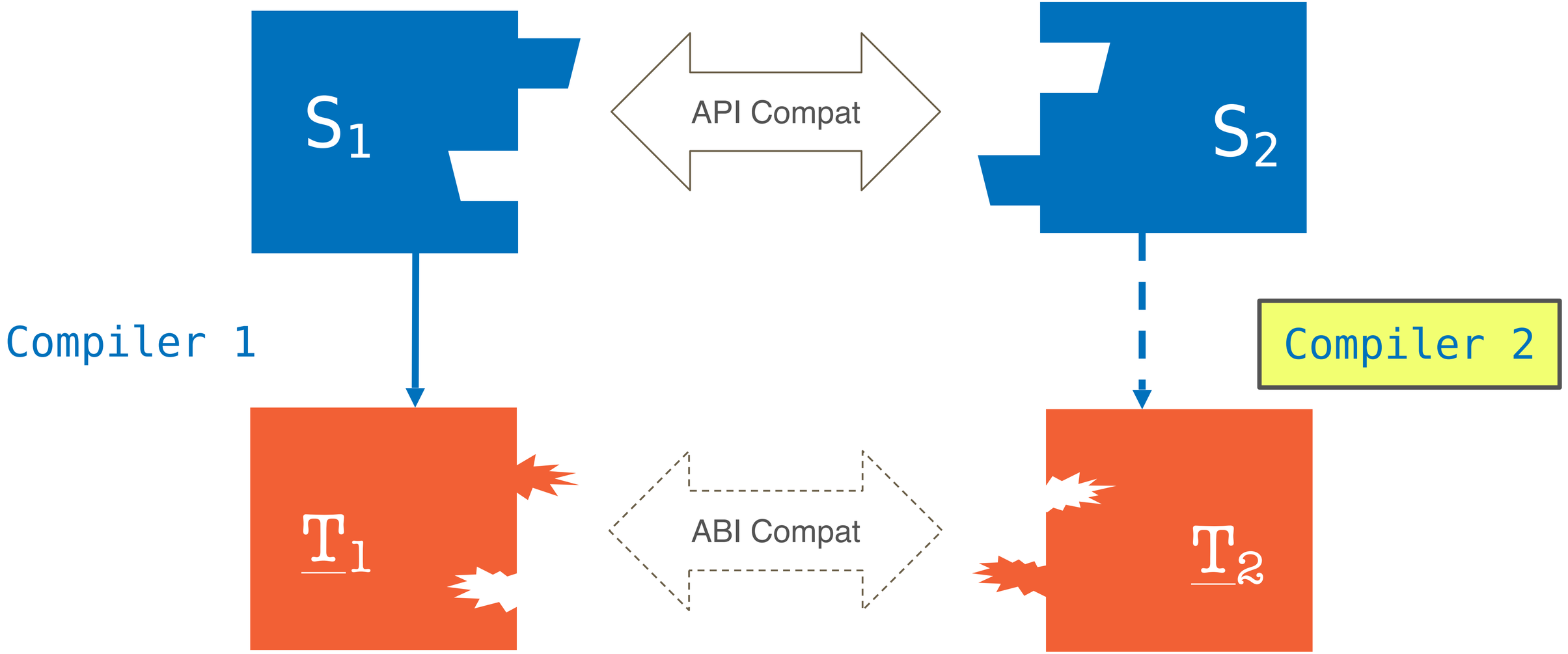
Why Use an ABI? Interoperability!



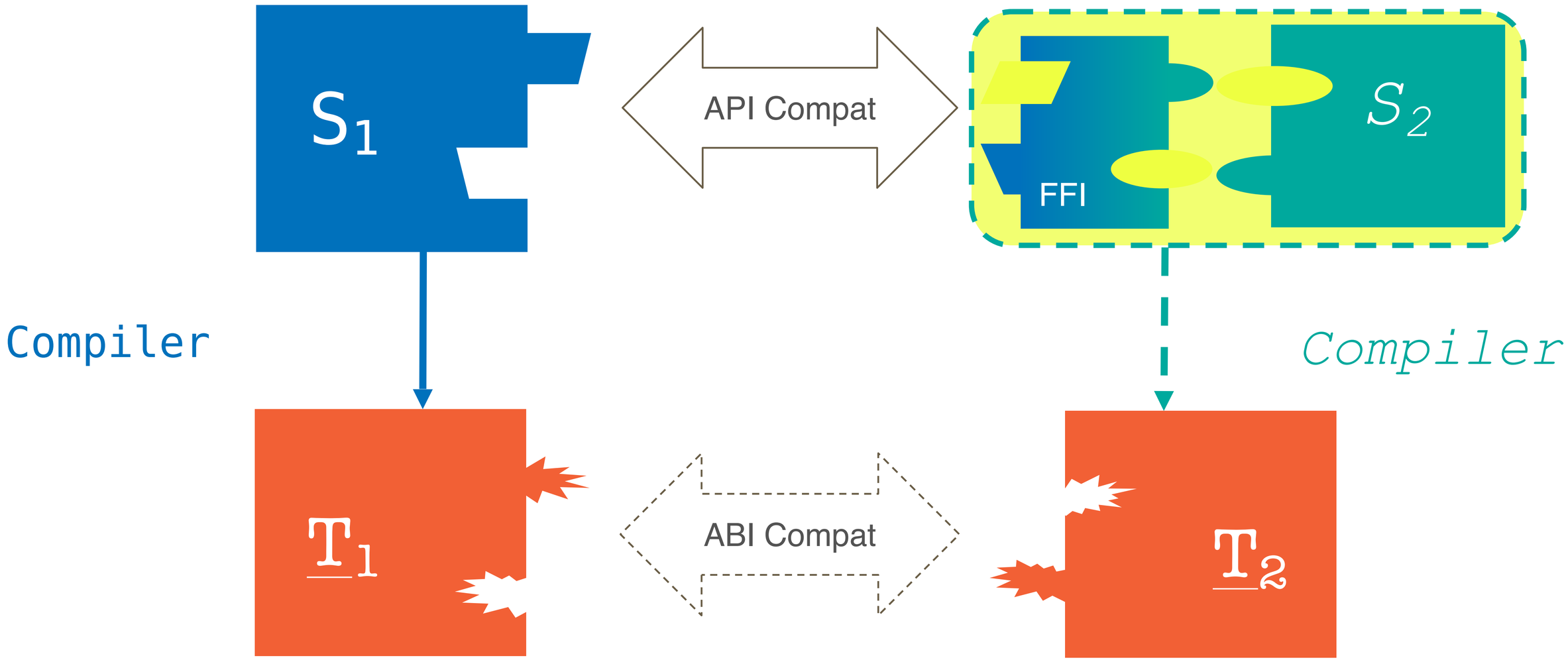
Why Use an ABI? Interoperability!



Why Use an ABI? Interoperability for Compilers



Why Use an ABI? Interoperability for **Languages**



So Why Doesn’t Every Language Stabilize an ABI?

So Why Doesn’t Every Language Stabilize an ABI?

Fear of Commitment 😳

So Why Doesn't Every Language Stabilize an ABI?

Fear of Commitment 😳

Example: What is the Layout of a `struct`?

Option 1: Rigid Layout *Like C ABI*

$[\![\text{struct Student } \{\text{reg} : \text{bool}, \text{id} : \text{int}\}]\!](\ell)$

+0	+1	+2	+3	+4	+5	+6	+7
TRUE	?	?	?	1710			

Option 1: Rigid Layout *Like C ABI*

$[\![\text{struct Student } \{\text{reg} : \text{bool}, \text{id} : \text{int}\}]\!](\ell)$

+0	+1	+2	+3	+4	+5	+6	+7
TRUE	?	?	?	1710			

No reordering 

$[\![\text{struct Student } \{\text{id} : \text{int}, \text{reg} : \text{bool}\}]\!](\ell)$

+0	+1	+2	+3	+4	+5	+6	+7
1710				TRUE	?	?	?

Option 1: Rigid Layout *Like C ABI*

$[\![\text{struct Student } \{\text{reg} : \text{bool}, \text{id} : \text{int}\}]\!](\ell)$

+0	+1	+2	+3	+4	+5	+6	+7
TRUE	?	?	?	1710			

No extensions 

$[\![\text{struct Student } \{\text{reg} : \text{bool}, \text{id} : \text{int}, \text{year} : \text{char}\}]\!](\ell)$

+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11
TRUE	?	?	?	1710				3	?	?	?

Option 1: Rigid Layout *Like C ABI*

$[\![\text{struct Student } \{\text{reg} : \text{bool}, \text{id} : \text{int}\}]\!](\ell)$

+0	+1	+2	+3	+4	+5	+6	+7
TRUE	?	?	?	1710			

```
/*
torvalds/linux/include/uapi/linux/stat.h
*/
struct statx {
  ...
  __u64 __spare3[9];
  /* Spare space for future expansion */
};
```

No extensions 

$[\![\text{struct Student } \{\text{reg} : \text{bool}, \text{id} : \text{int}, \text{year} : \text{char}\}]\!](\ell)$

+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11
TRUE	?	?	?	1710				3	?	?	?

Option 1: Rigid Layout *Like C ABI*

$[\![\text{struct Student } \{\text{reg} : \text{bool}, \text{id} : \text{int}, \text{year} : \text{char}\}]\!](\ell)$

+0	+1	+2	+3	+4	+5	+6	+7	+8	+9	+10	+11
TRUE	?	?	?	1710				3	?	?	?

No optimizations?! 

$[\![\text{struct Student } \{\text{reg} : \text{bool}, \text{id} : \text{int}, \text{year} : \text{char}\}]\!](\ell)$

+0	+1	+2	+3	+4	+5	+6	+7
TRUE	3	?	?	1710			

Option 2: Resilient Layout *Like Swift ABI*

Option 2: Resilient Layout *Like Swift ABI*

Client Using `Student`

Offset Table

...	reg	...	id	...
...	o_{reg}	...	o_{id}	...

...	o_{reg}	...	o_{id}	+1	+2	+3	...
...	TRUE	...	1710				...

Option 2: Resilient Layout

Like Swift ABI

Client Using `Student`

Offset Table

...	reg	...	id	...
...	O_{reg}	...	O_{id}	...

...	O_{reg}	...	O_{id}	+1	+2	+3	...
...	TRUE	...	1710				...

Library Providing `Student`

Offset Table

reg	id	year
5	0	4

+0	+1	+2	+3	+4	+5	+6	+7
1710				3	TRUE	?	?

Option 2: Resilient Layout

Like Swift ABI

Client Using `Student`

Offset Table

...	reg	...	id	...
...	o_{reg}	...	o_{id}	...

...	o_{reg}	...	o_{id}	+1	+2	+3	...
...	TRUE	...	1710				...

Library Providing `Student`

Offset Table

reg	id	year
5	0	4

+0	+1	+2	+3	+4	+5	+6	+7
1710				3	TRUE	?	?

Many valid options ➔ Flexibility 📈

Option 2: Resilient Layout

Like Swift ABI

Client Using `Student`

Offset Table

...	reg	...	id	...
...	o_{reg}	...	o_{id}	...

...	o_{reg}	...	o_{id}	+1	+2	+3	...
...	TRUE	...	1710				...

Indirect access ➔ Performance 📉

Library Providing `Student`

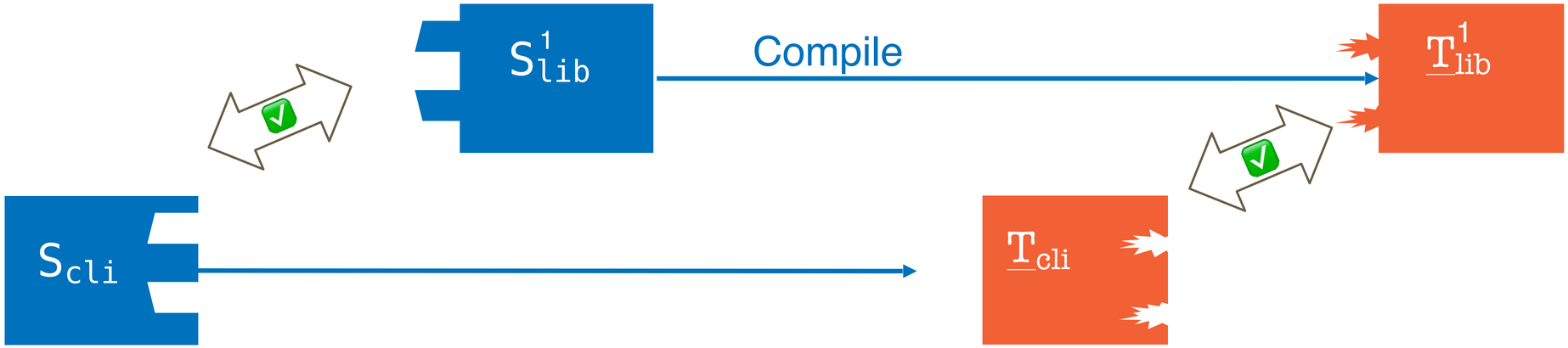
Offset Table

reg	id	year
5	0	4

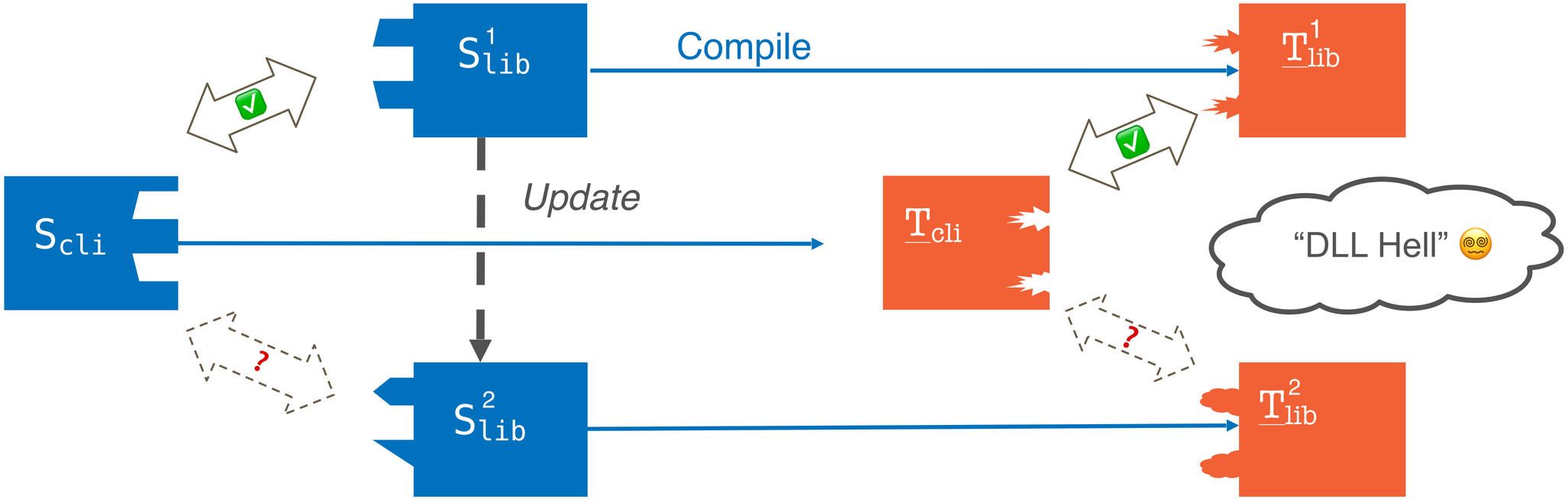
+0	+1	+2	+3	+4	+5	+6	+7
1710				3	TRUE	?	?

Many valid options ➔ Flexibility 📈

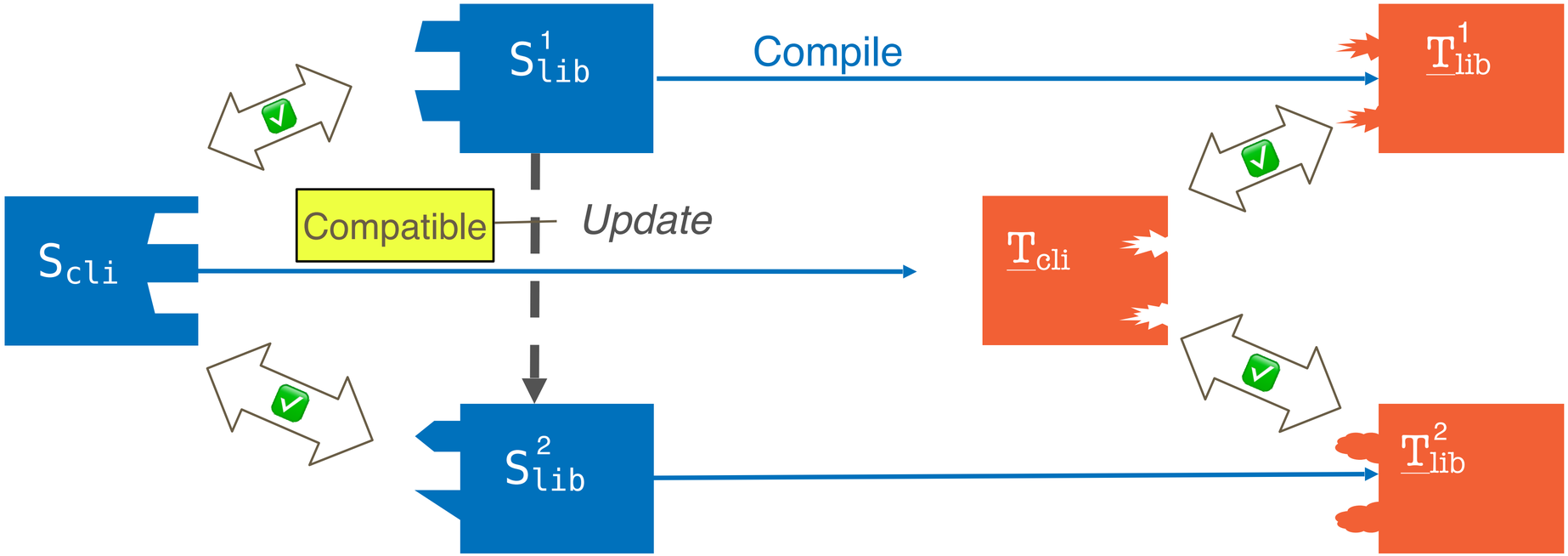
Consequences: Library Evolution



Consequences: Library Evolution



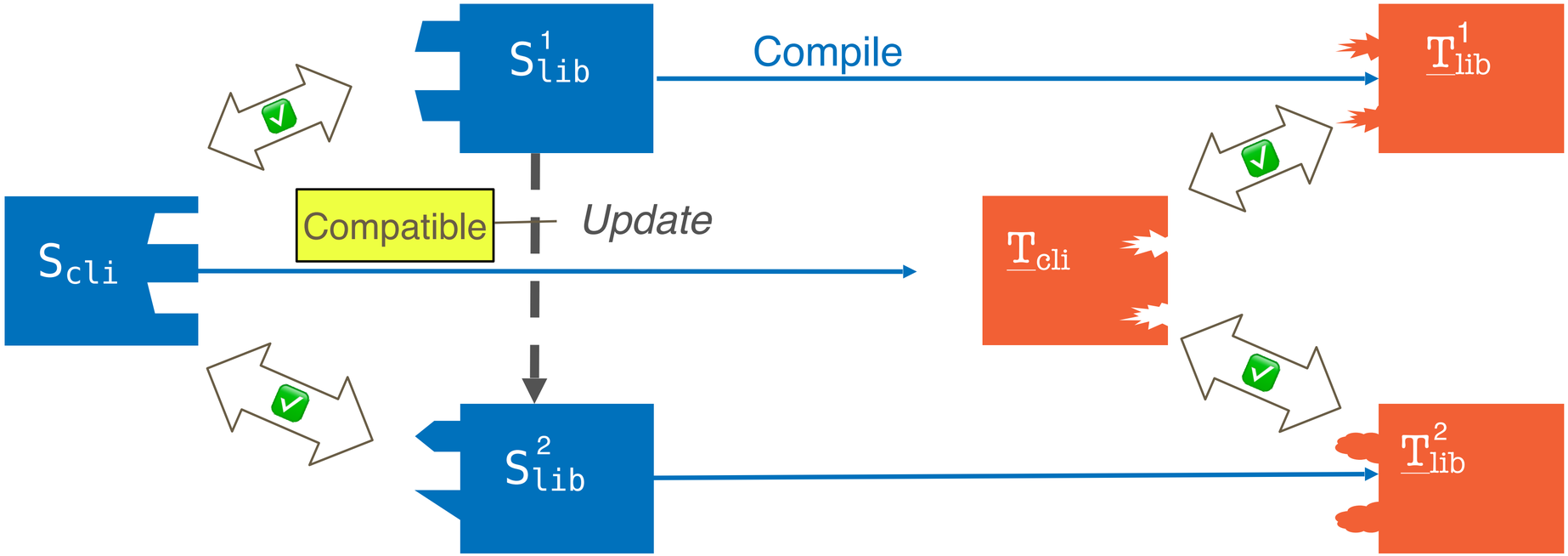
Consequences: Library Evolution



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

$[\![\tau_2]\!]$ refines $[\![\tau_1]\!]$

Consequences: Library Evolution



Swift ? C

Flexible Rigid

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

$[\![\tau_2]\!]$ refines $[\![\tau_1]\!]$

To Stabilize or Not to Stabilize?

Pros

+ Precise control of interface to other languages
+ First-class support for shared libraries

Cons

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

To Stabilize or Not to Stabilize?

Pros

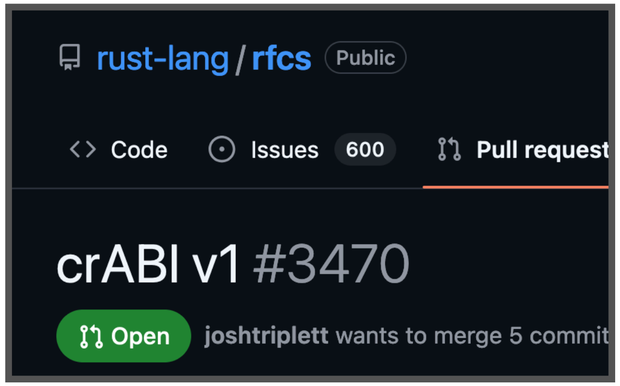
+ Precise control of interface to other languages
+ First-class support for shared libraries

Cons

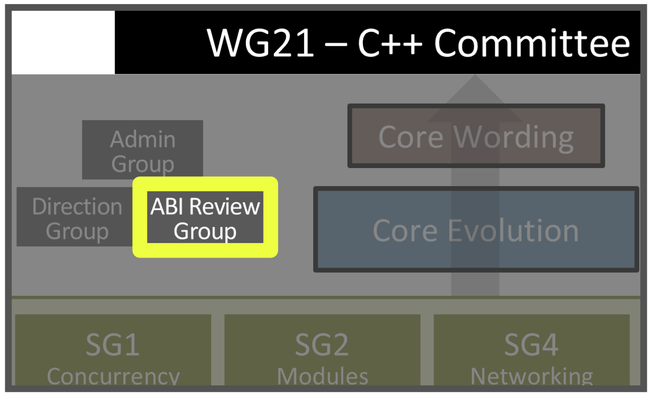
− Can stunt language growth
− Limits compiler optimizations
− Tension between flexibility and performance
− Pressure on library developers

Swift

Rust



C++



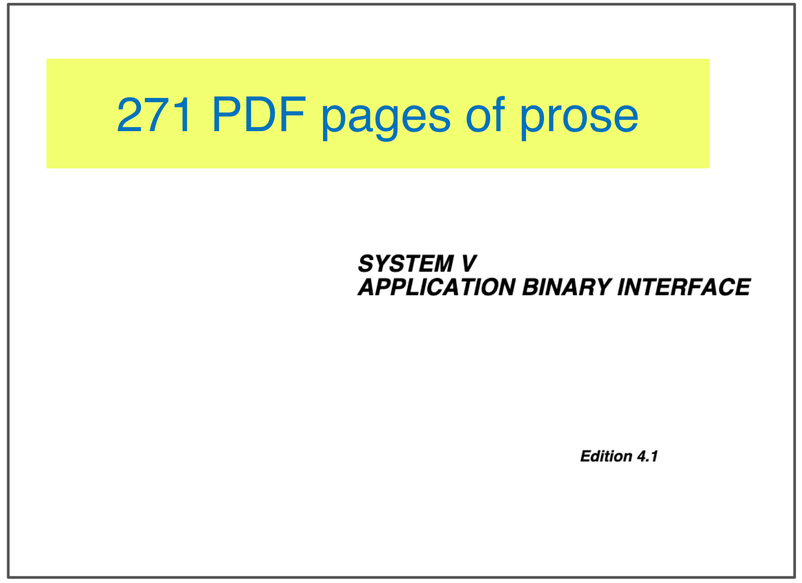
How Are ABIs Specified?

C ABI

Swift ABI

How Are ABIs Specified?

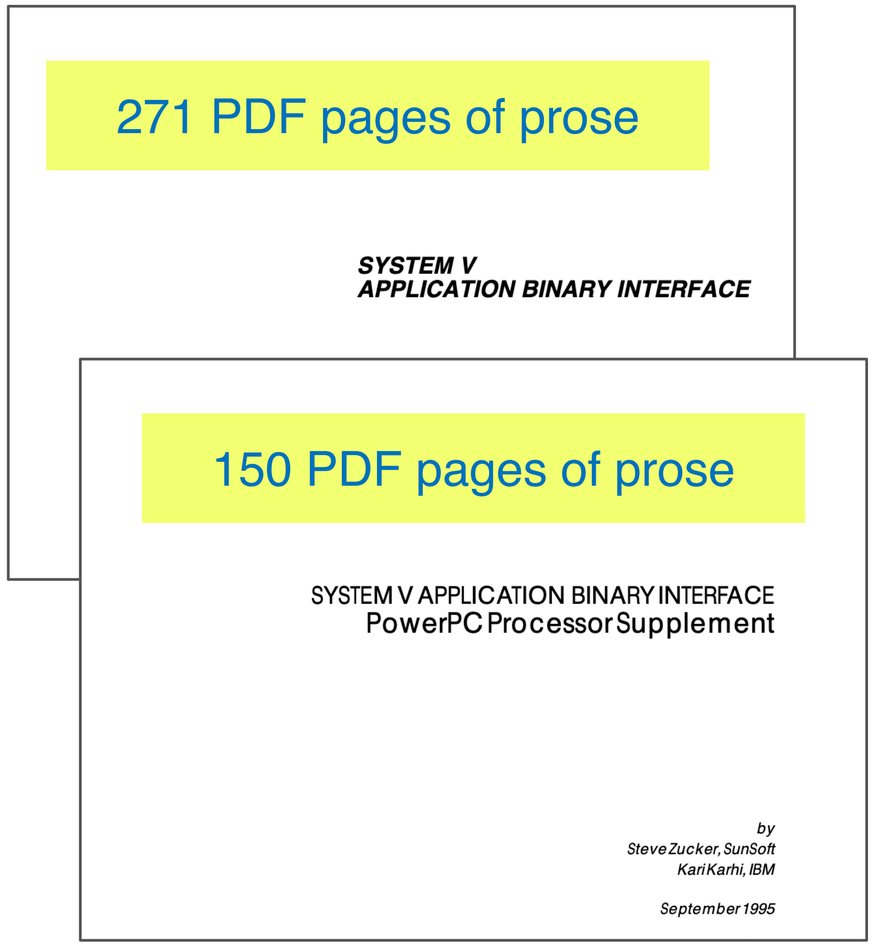
C ABI



Swift ABI

How Are ABIs Specified?

C ABI



Swift ABI

How Are ABIs Specified?

C ABI

271 PDF pages of prose

SYSTEM V
APPLICATION BINARY INTERFACE

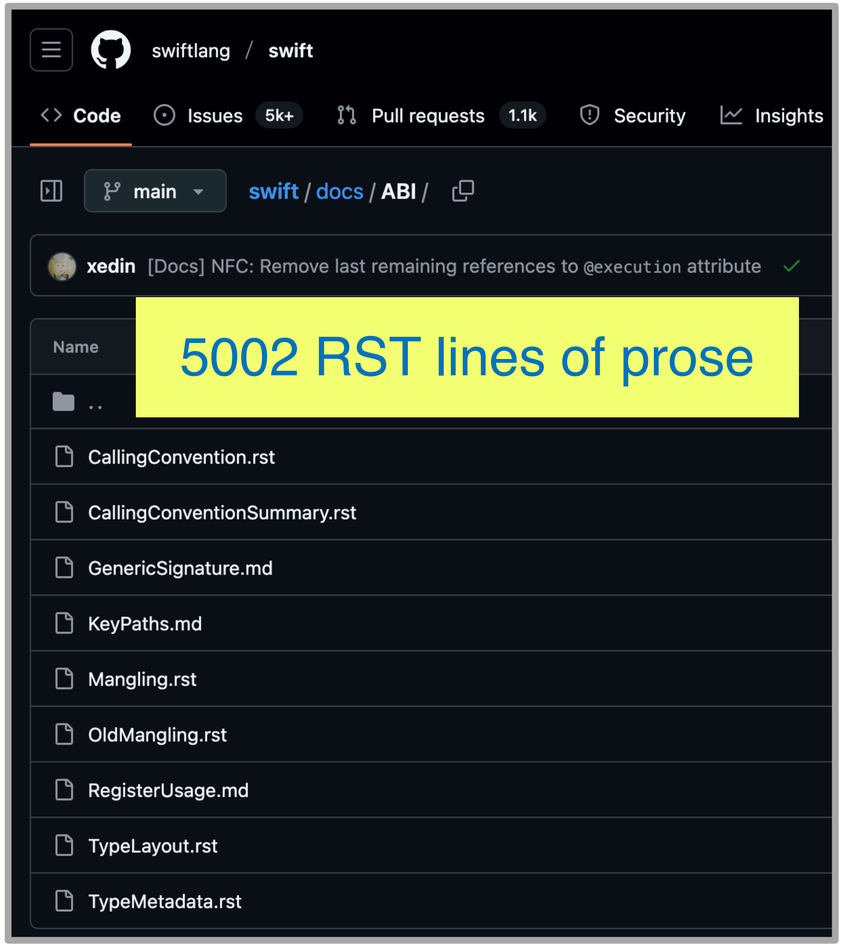
150 PDF pages of prose

SYSTEM V APPLICATION BINARY INTERFACE
PowerPC Processor Supplement

by
Steve Zucker, SunSoft
Kari Karhi, IBM

September 1995

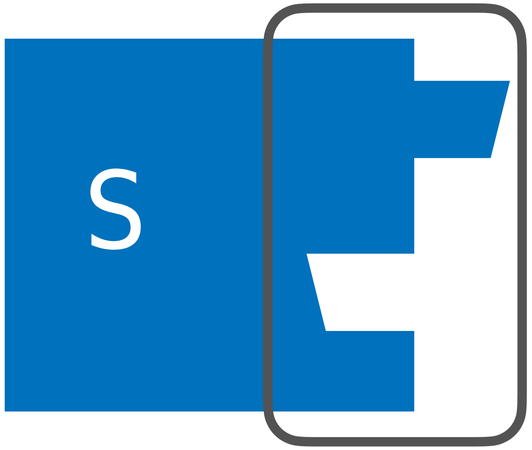
Swift ABI



How Can ABIs Be Specified Formally?

The run-time contract for using a particular API

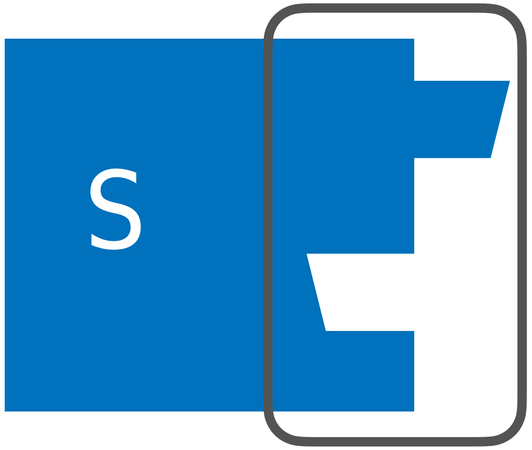
How Can ABIs Be Specified Formally?



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

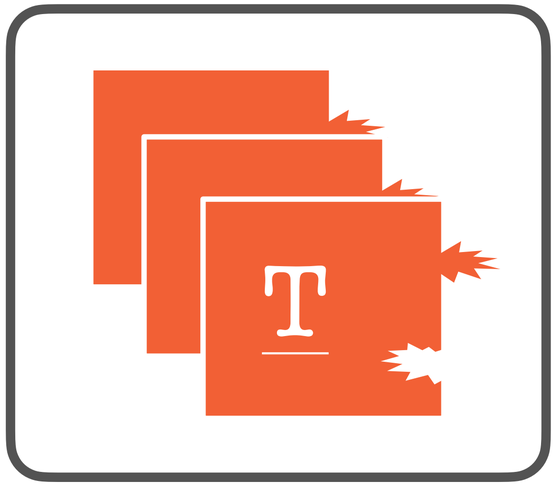
This Type `τ`

How Can ABIs Be Specified Formally?



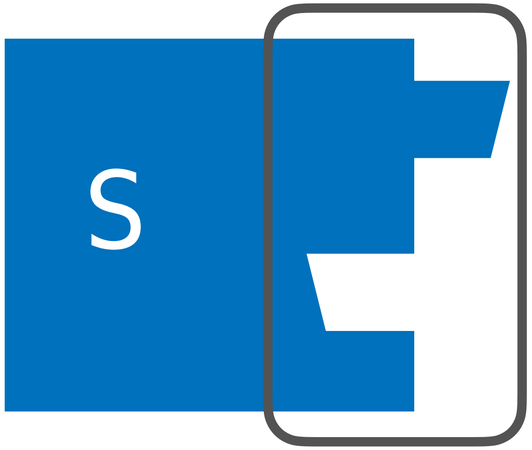
The run-time contract for using a particular API

This Type τ



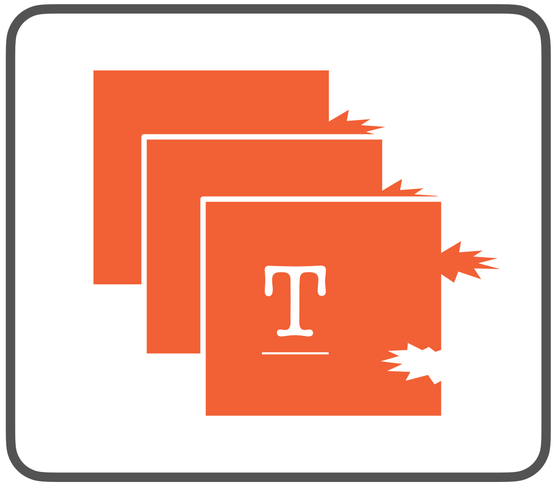
Is *Realistically Realized [Benton06]*
By These Target Programs
⟦τ⟧ = { e | … }

How Can ABIs Be Specified Formally?



The run-time contract for using a particular API

This Type τ



Is *Realistically Realized [Benton06]*
By These Target Programs
⟦τ⟧ = { e | … }

Our Approach

e is **ABI compliant** with τ if

e ∈ ⟦τ⟧

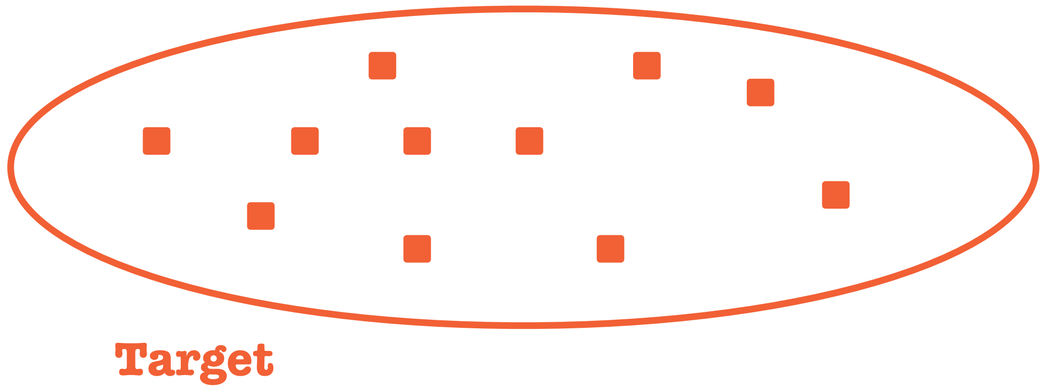
OOPSLA24

Realistic Realizability: Specifying ABIs You Can Count On

ANDREW WAGNER, Northeastern University, USA
ZACHARY EISBACH, Northeastern University, USA
AMAL AHMED, Northeastern University, USA

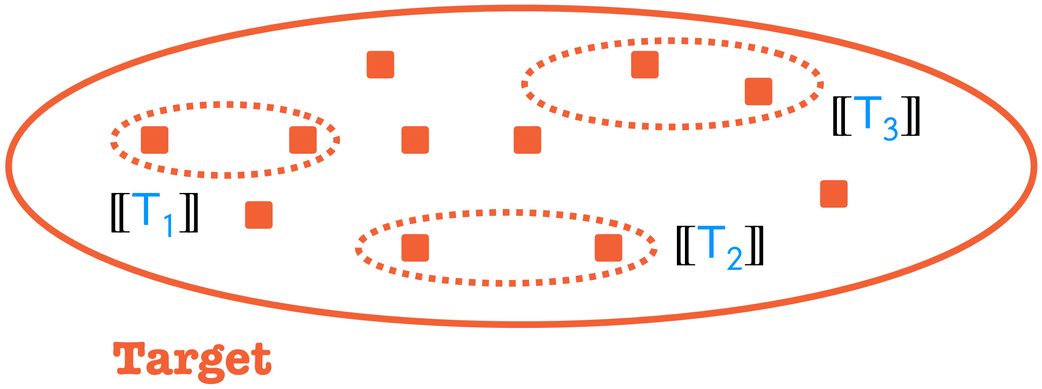
The Application Binary Interface (ABI) for a language defines the interoperability rules for its target platforms, including data layout and calling conventions, such that compliance with the rules ensures "safe" execution and perhaps certain resource usage guarantees. These rules are relied upon by compilers, libraries, and foreign-function interfaces. Unfortunately, ABIs are typically specified in prose, and while type systems for source

The Recipe



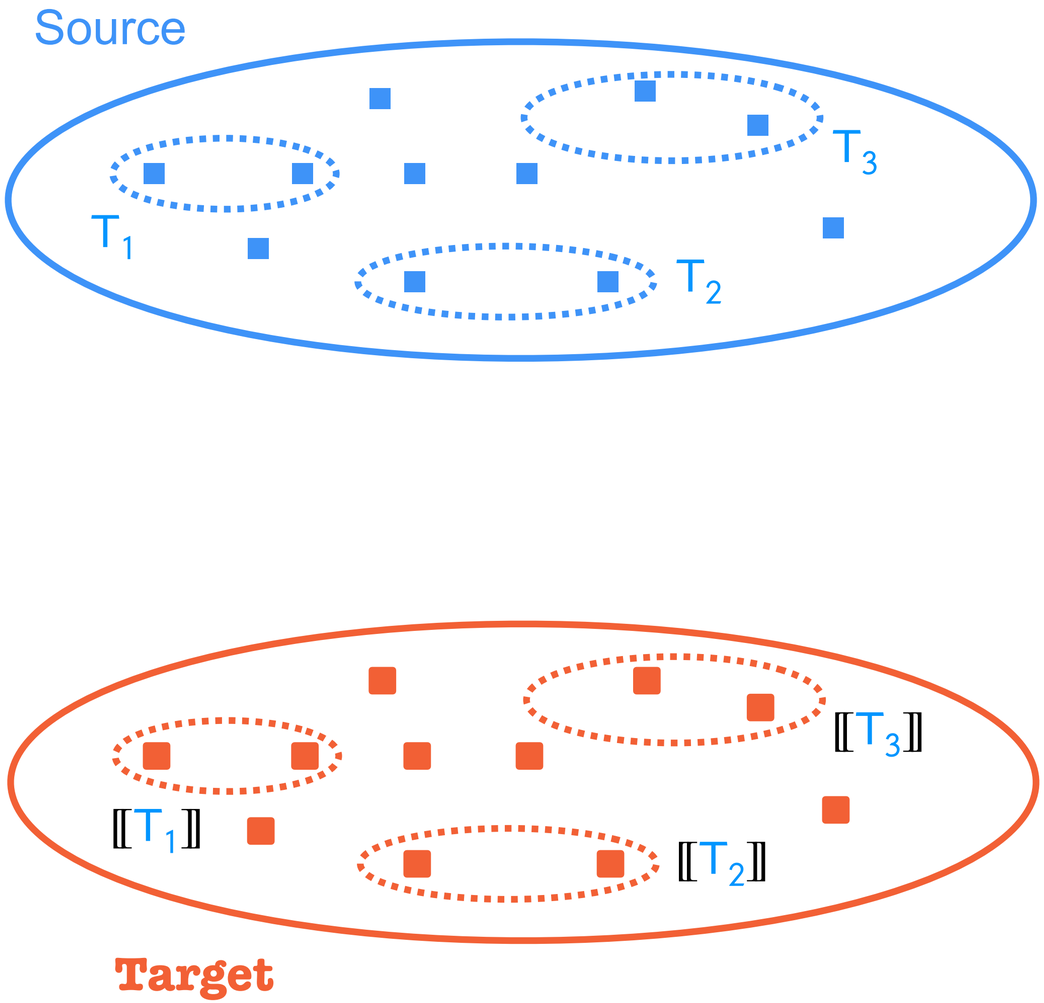
The Recipe

- **Define the ABI** as a mapping $[\![-]\!]$ from source types T to *separation logic* predicates over target terms



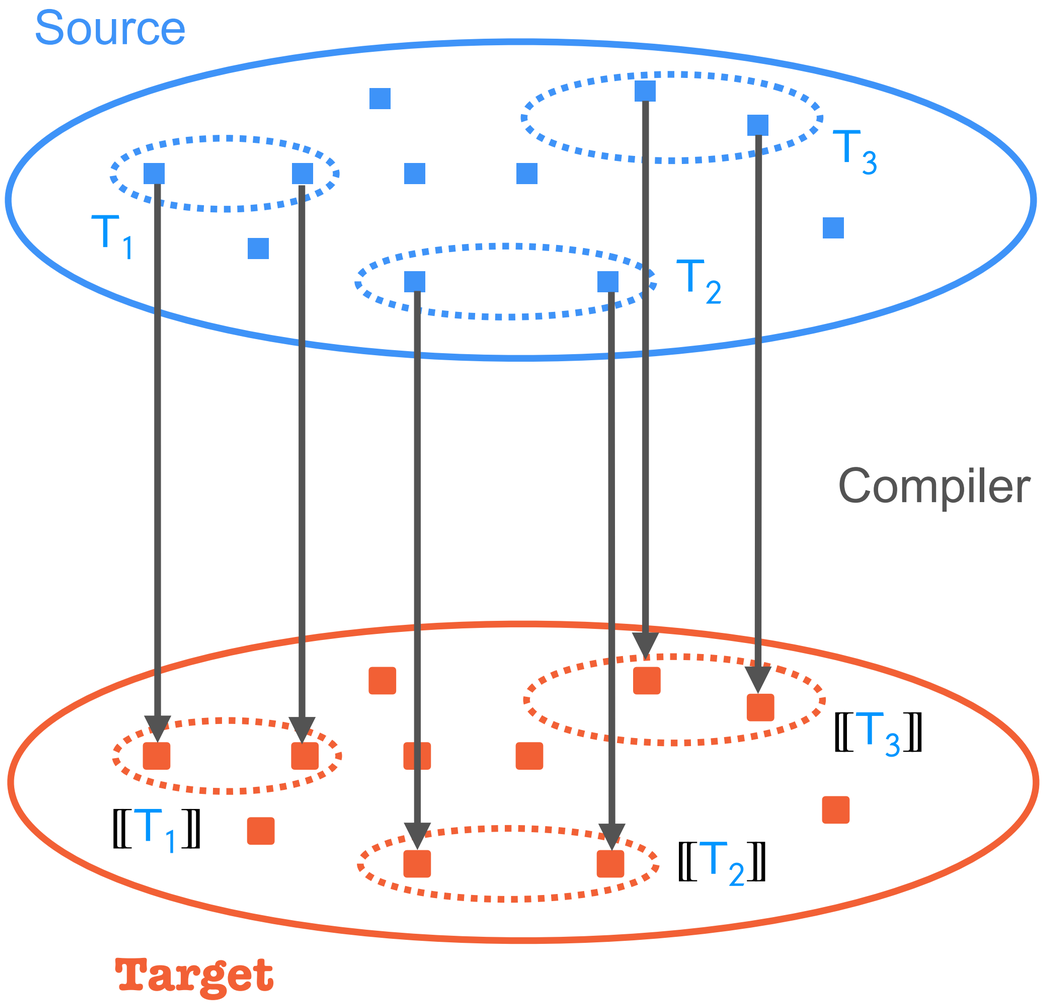
The Recipe

- **Define the ABI** as a mapping $[\![-]\!]$ from source types T to *separation logic* predicates over target terms



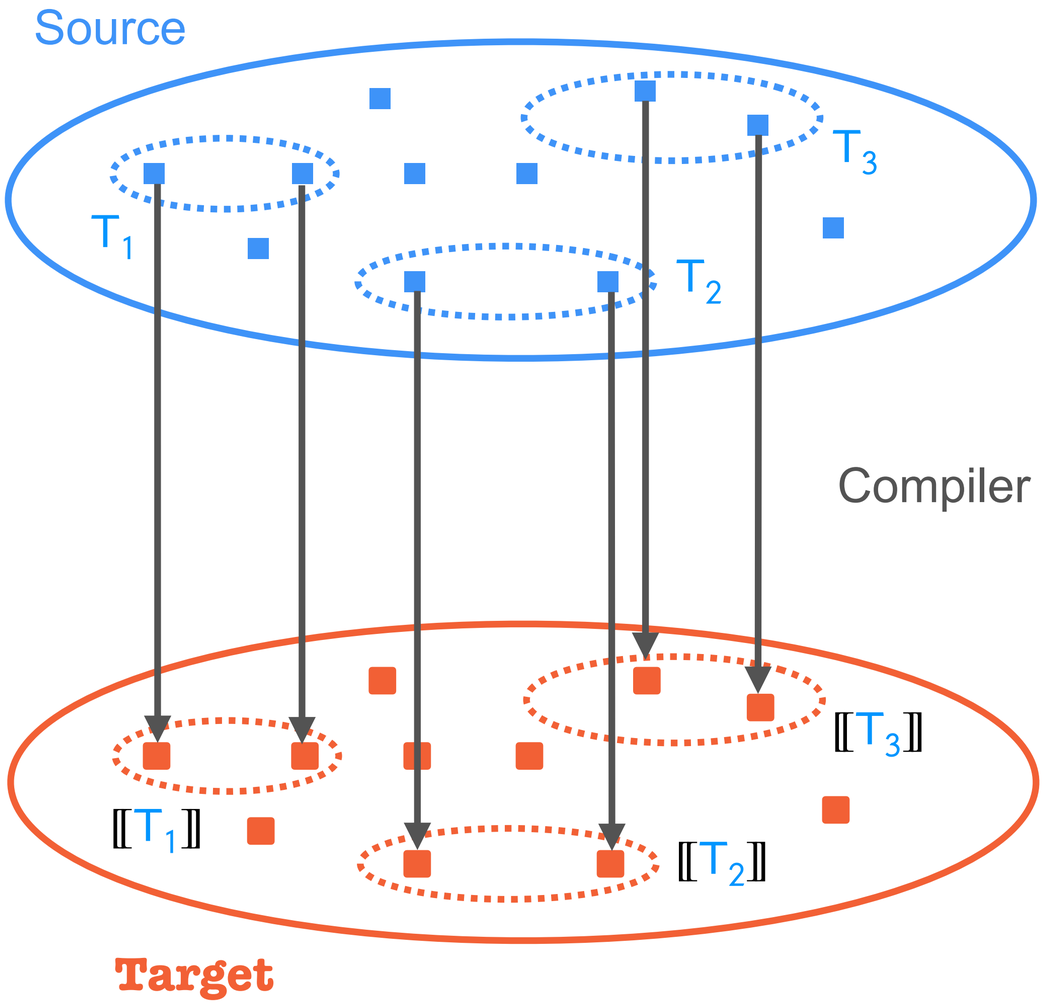
The Recipe

- **Define the ABI** as a mapping $[\![-]\!]$ from source types T to *separation logic* predicates over target terms
- **Prove compiler compliance** by showing that $\mathsf{e} \in [\![\mathsf{T}]\!]$ whenever a source term e of type T compiles to target term e



The Recipe

- **Define the ABI** as a mapping $[\![-]\!]$ from source types T to *separation logic* predicates over target terms
- **Prove compiler compliance** by showing that $e \in [\![\mathsf{T}]\!]$ whenever a source term e of type T compiles to target term e



Crash Course: Hoare Logic

$$\{P\}\,\mathtt{e}\,\{\mathtt{v}\,.\,Q\}$$

"In any state satisfying the *precondition* P,
expression $\mathtt{e}$ will run to a value $\mathtt{v}$
and a state satisfying *postcondition* Q"

Crash Course: Hoare Logic

$$\{P\}\,\mathtt{e}\,\{\mathtt{v}\,.\,Q\}$$

"In any state satisfying the *precondition* P,
expression $\mathtt{e}$ will run to a value $\mathtt{v}$
and a state satisfying *postcondition* Q"

$$\{\underbrace{\ell \mapsto 3}\}\,\mathtt{load}\,\ell\,\{\mathtt{v}\,.\,\mathtt{v} = 3 \land \ell \mapsto 3\}$$

"Location ℓ
maps to the value 3
in memory"

Crash Course: Hoare Logic

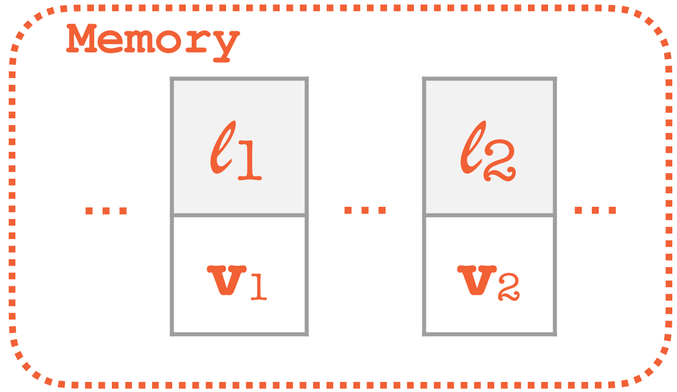
$$\{\ell_1 \mapsto v_1 \land \ell_2 \mapsto v_2\}$$

$$\texttt{free}\ \ell_1;$$

$$\{\ell_2 \mapsto v_2\}$$

$$\texttt{load}\ \ell_2$$

$$\{v.\, v = v_2 \land \ell_2 \mapsto v_2\}$$



Crash Course: Hoare Logic

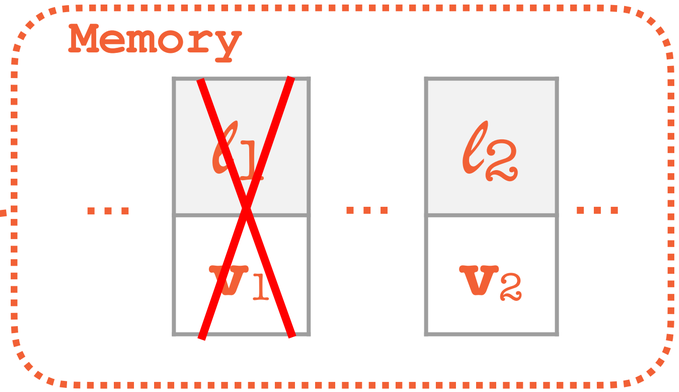
$$\{\ell_1 \mapsto v_1 \land \ell_2 \mapsto v_2\}$$

$$\texttt{free}\ \ell_1;$$

$$\{\ell_2 \mapsto v_2\}$$

$$\texttt{load}\ \ell_2$$

$$\{v.\, v = v_2 \land \ell_2 \mapsto v_2\}$$



Crash Course: Hoare Logic

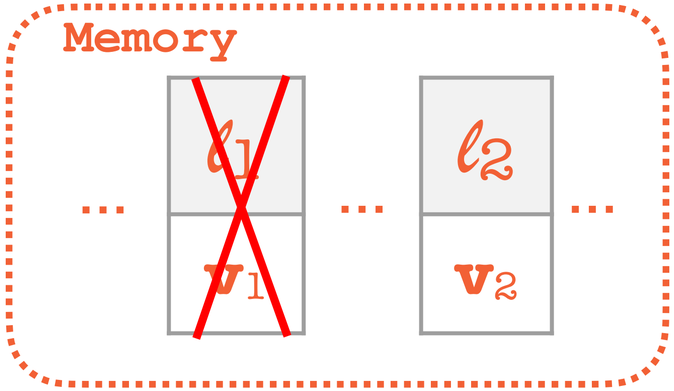
$$\{\ell_1 \mapsto v_1 \wedge \ell_2 \mapsto v_2\}$$

$$\texttt{free}\ \ell_1;$$

$$\{\ell_2 \mapsto v_2\}$$

$$\texttt{load}\ \ell_2$$

$$\{v.\, v = v_2 \wedge \ell_2 \mapsto v_2\}$$



Crash Course: Hoare Logic

$$\{\ell_1 \mapsto v_1 \land \ell_2 \mapsto v_2\}$$

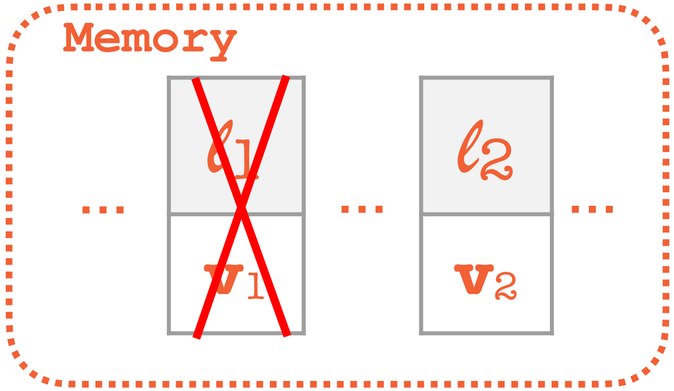
$$\texttt{free}\ \ell_1;$$

$$\{\ell_2 \mapsto v_2\}$$

$$\texttt{load}\ \ell_2$$

$$\{v.\, v = v_2 \land \ell_2 \mapsto v_2\}$$

But what if $\ell_1 = \ell_2$??



Crash Course: Hoare Logic

$$\{\ell_1 \mapsto v_1 \land \ell_2 \mapsto v_2\}$$

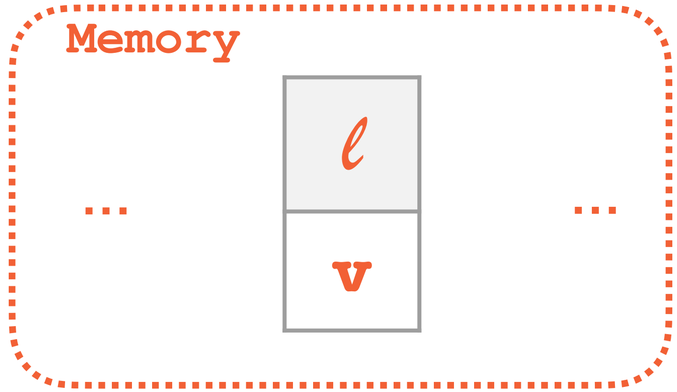
$$\texttt{free}\ \ell_1;$$

$$\{\ell_2 \mapsto v_2\}$$

$$\texttt{load}\ \ell_2$$

$$\{v.\, v = v_2 \land \ell_2 \mapsto v_2\}$$

But what if $\ell_1 = \ell_2 = \ell$??



Crash Course: Hoare Logic

$$\{\ell_1 \mapsto v_1 \wedge \ell_2 \mapsto v_2\}$$

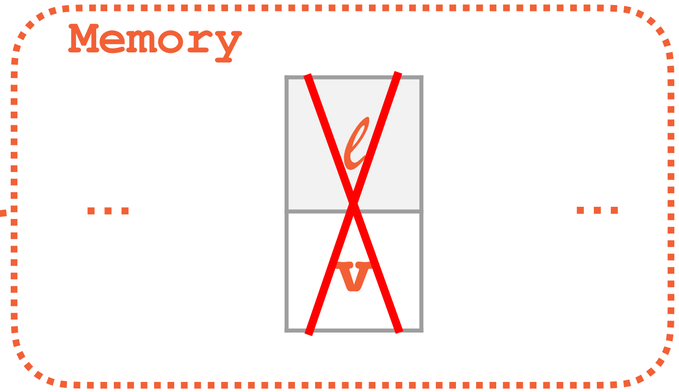
$$\texttt{free}\ \ell_1;$$

$$\{\cancel{\ell_2 \mapsto v_2}\}$$

$$\texttt{load}\ \ell_2$$

$$\{v.\, v = v_2 \wedge \ell_2 \mapsto v_2\}$$

But what if $\ell_1 = \ell_2 = \ell$??



Crash Course: Hoare Logic

$\{\ell_1 \mapsto v_1 \land \ell_2 \mapsto v_2\}$

`free` ℓ_1;

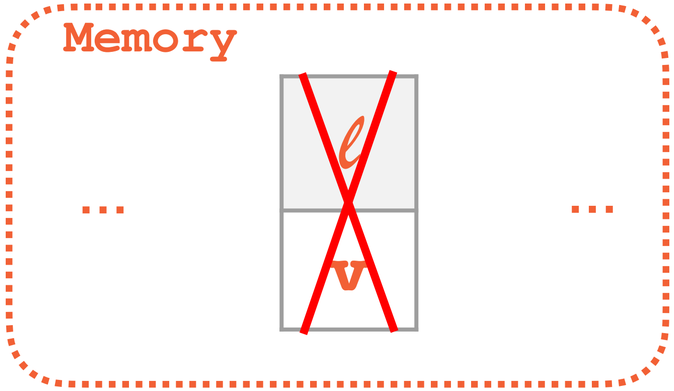
$\{\ell_2 \mapsto v_2\}$

`load` ℓ_2

$\{v.\, v = v_2 \land \ell_2 \mapsto v_2\}$

error: use after free!

But what if $\ell_1 = \ell_2 = \ell$??



Crash Course: Separation Logic

$$\frac{\{P\}\,\mathtt{e}\,\{\mathtt{v}\,.\,Q\}}{\{P \star P_f\}\,\mathtt{e}\,\{\mathtt{v}\,.\,Q \star P_f\}}\text{(Frame)}$$

"Any valid triple is still valid if extended with a *separate frame* (P_f)"

Crash Course: Separation Logic

Separating Conjunction (“sep”)

$$\frac{\{P\}\,\mathtt{e}\,\{\mathtt{v}\,.\,Q\}}{\{P \star P_f\}\,\mathtt{e}\,\{\mathtt{v}\,.\,Q \star P_f\}}\text{(Frame)}$$

“Any valid triple is still valid if extended with a *separate frame* (P_f)”

Crash Course: Separation Logic

Separating Conjunction (“sep”)

$$\frac{\{P\}\,\mathtt{e}\,\{\mathtt{v}\,.\,Q\}}{\{P \star P_f\}\,\mathtt{e}\,\{\mathtt{v}\,.\,Q \star P_f\}}\text{(Frame)}$$

“Any valid triple is still valid if extended with a *separate frame* (P_f)”

$$\ell \mapsto \mathtt{v}_1 \star \ell \mapsto \mathtt{v}_2 \vdash \text{False}$$

Crash Course: Separation Logic

No aliasing ➔ No use-after-free

$$\{\ell_1 \mapsto v_1 \star \ell_2 \mapsto v_2\}$$

$$\texttt{free}\,\ell_1;$$

$$\{\ell_2 \mapsto v_2\}$$

$$\texttt{load}\,\ell_2$$

$$\{v.\, v = v_2 \wedge \ell_2 \mapsto v_2\}$$

Specifying Layout

$[\![\text{struct Student } \{\text{reg} : \text{bool}, \text{id} : \text{int}\}]\!](\ell)$

+0	+1	+2	+3	+4	+5	+6	+7
TRUE	?	?	?	1710			

$$\ell \mapsto \texttt{TRUE} \star \left(\bigstar_{i=1}^{3} \ell + \texttt{i} \mapsto ? \right) \star \left(\bigstar_{i=4}^{7} \ell + \texttt{i} \mapsto \text{byte}_{i-4}(1710) \right)$$

Calling Conventions

$$[\![\overrightarrow{\mathtt{T}_1} \to \mathtt{T}_2]\!](\mathtt{f})$$

Calling Conventions

$$[\![\overrightarrow{\mathtt{T}_1} \to \mathtt{T}_2]\!](\mathtt{f})$$

$$\{ \overrightarrow{\star\ [\![\mathtt{T}_1]\!](\mathtt{v}_1)} \}\mathtt{f}(\overrightarrow{\mathtt{v}_1})\{\mathtt{v}_2 . [\![\mathtt{T}_2]\!](\mathtt{v}_2)\}$$

Calling Conventions

$$[\![\overrightarrow{T_1} \to T_2]\!](f)$$

Argument Order

$$\{ \overrightarrow{\star\ [\![T_1]\!](v_1)} \}f(\overrightarrow{v_1})\{v_2 . [\![T_2]\!](v_2)\} \quad \text{vs.} \quad \{ \overrightarrow{\star\ [\![T_1]\!](v_1)} \}f(\overleftarrow{v_1})\{v_2 . [\![T_2]\!](v_2)\}$$

Left-to-Right

Right-to-Left

Calling Conventions

$$[\![\overrightarrow{T_1} \to T_2]\!](f)$$

Argument Order

$$\{ \overrightarrow{\star\, [\![T_1]\!](v_1)} \} f(\overrightarrow{v_1}) \{ v_2 . [\![T_2]\!](v_2) \} \quad \text{vs.} \quad \{ \overrightarrow{\star\, [\![T_1]\!](v_1)} \} f(\overleftarrow{v_1}) \{ v_2 . [\![T_2]\!](v_2) \}$$

Left-to-Right | Right-to-Left

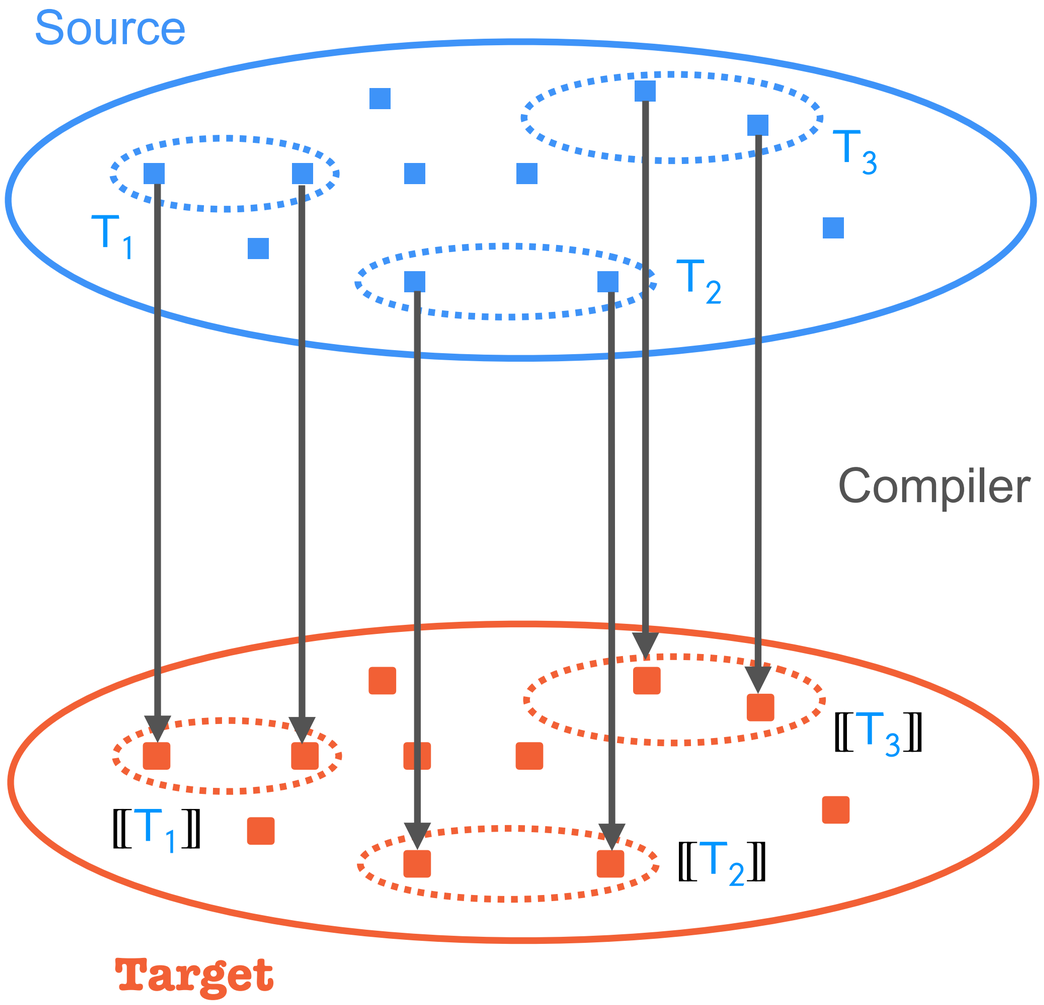
Ownership

$$\{ \overrightarrow{\star\, [\![T_1]\!](v_1)} \} f(\overrightarrow{v_1}) \{ v_2 . [\![T_2]\!](v_2) \} \quad \text{vs.} \quad \{ \overrightarrow{\star\, [\![T_1]\!](v_1)} \} f(\overrightarrow{v_1}) \{ v_2 . \overrightarrow{\star\, [\![T_1]\!](v_1)} \star [\![T_2]\!](v_2) \}$$

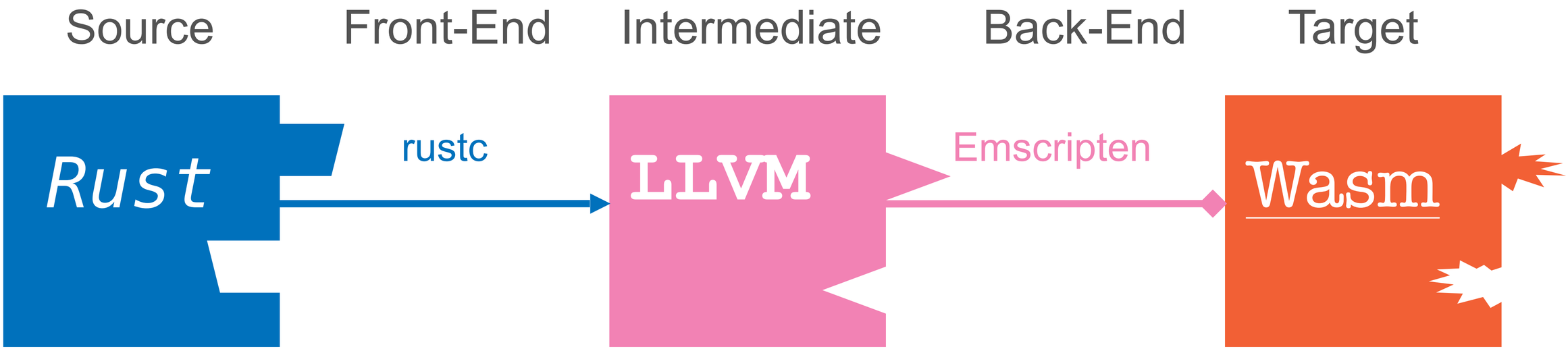
Caller Save | Callee Save

The Recipe

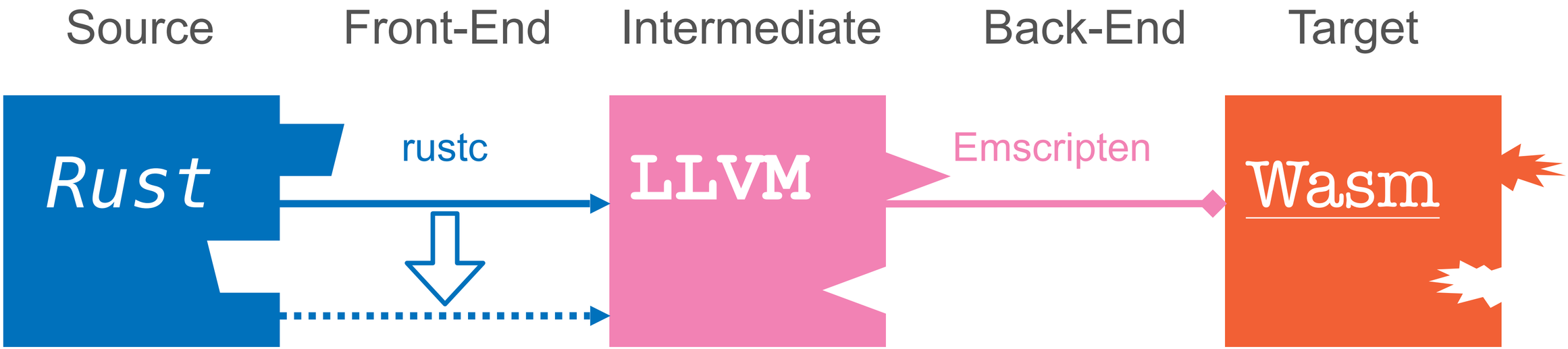
- **Define the ABI** as a mapping $[\![-]\!]$ from source types T to *separation logic* predicates over target terms
- **Prove compiler compliance** by showing that $\mathsf{e} \in [\![\mathsf{T}]\!]$ whenever a source term e of type T compiles to target term e



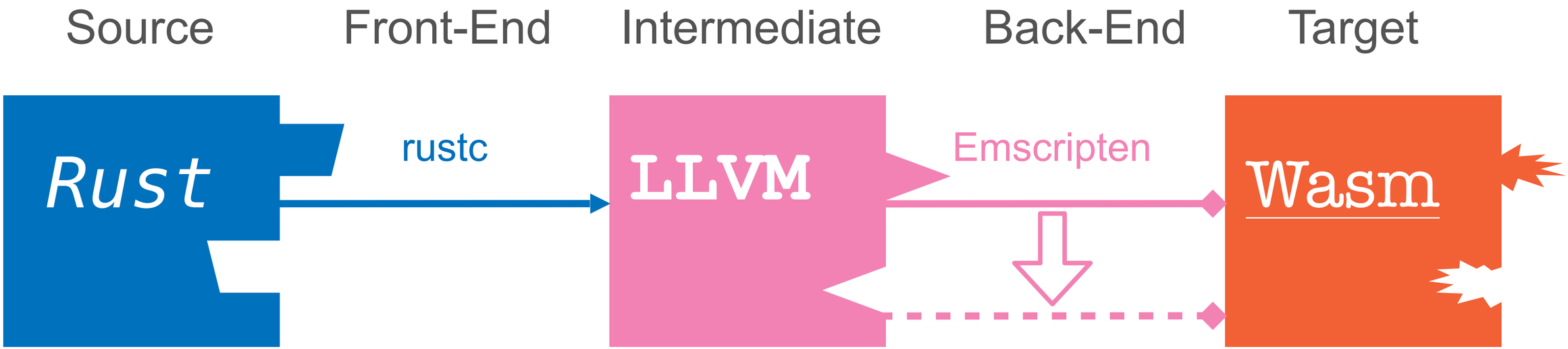
Multi-Pass Compilation (Ongoing)



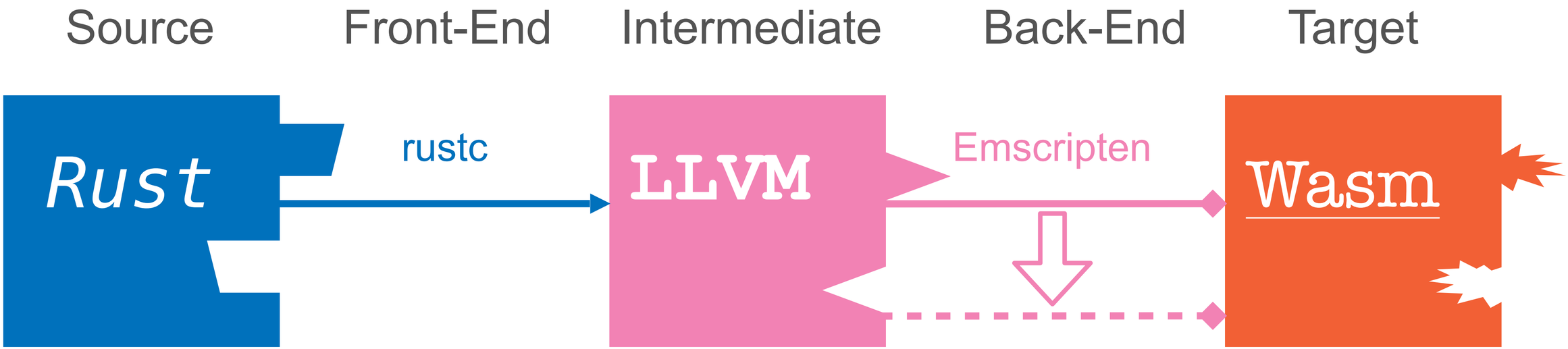
Multi-Pass Compilation (Ongoing)



Multi-Pass Compilation (Ongoing)



Multi-Pass Compilation (Ongoing)



How can we allow *independent updates* to the front-end and back-end?

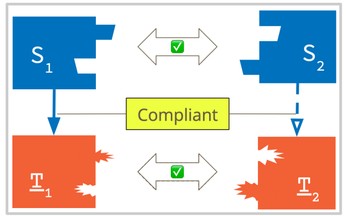
Takeaways

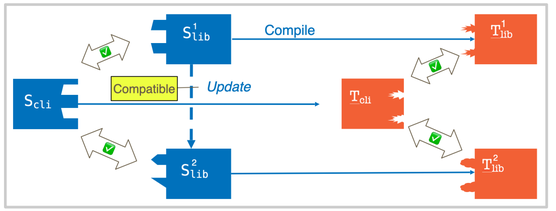
The Methodology

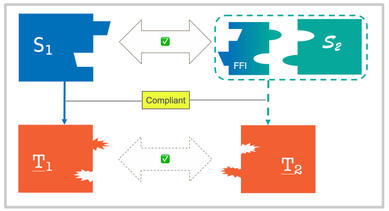
ABI Spec with Realistic Realizability

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

Compiler Compliance, Library Evolution, FFI Safety*







The Design Decisions

Performance vs. Flexibility

$[\![\text{struct Student } \{\text{reg} : \text{bool}, \text{id} : \text{int}\}]\!](\ell)$

Rigid

+0	+1	+2	+3	+4	+5	+6	+7
TRUE	?	?	?	1951			

Resilient

Client Using `Student`

Offset Table

...	reg	...	id	...
...	o_{reg}	...	o_{id}	...

...	o_{reg}	...	o_{id}	+1	+2	+3	...
...	TRUE	...	1710				...

Library Providing `Student`

Offset Table

reg	id	year
5	0	4

+0	+1	+2	+3	+4	+5	+6	+7
1710			3	TRUE	?	?	

Paper
Slides
Contact