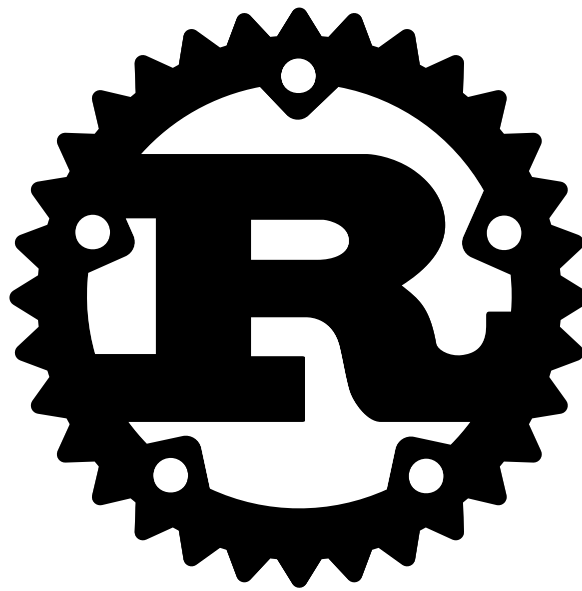
OXIDE:
THE ESSENCE OF RUST

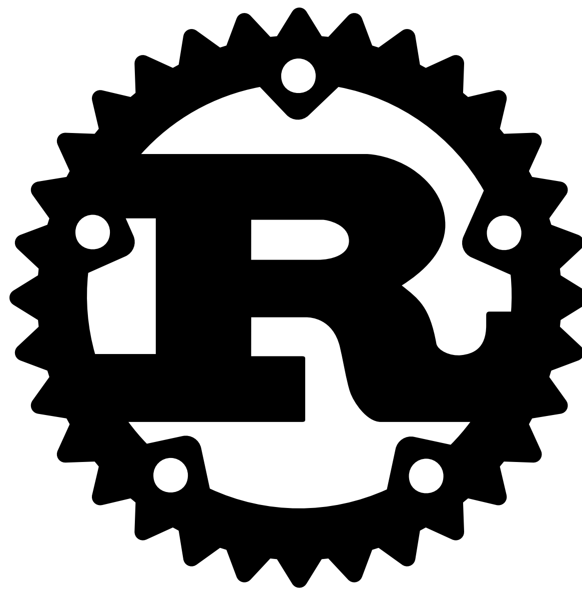
Aaron Weiss
Northeastern University



“

Rust is a systems programming language that runs blazingly fast, prevents segfaults, and guarantees thread safety.

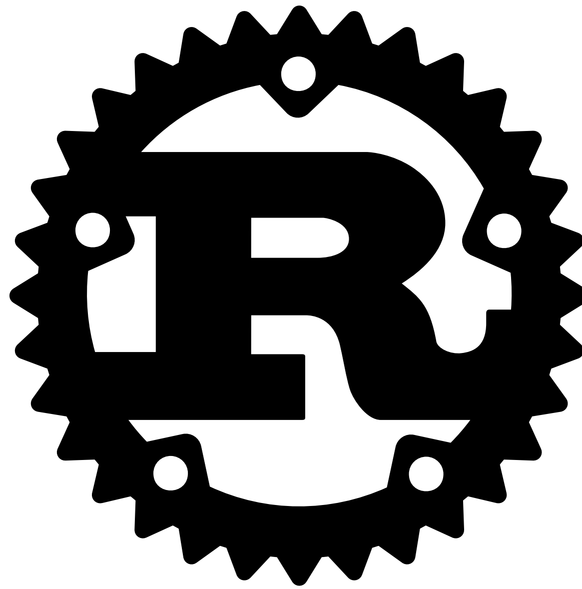
– the official Rust website



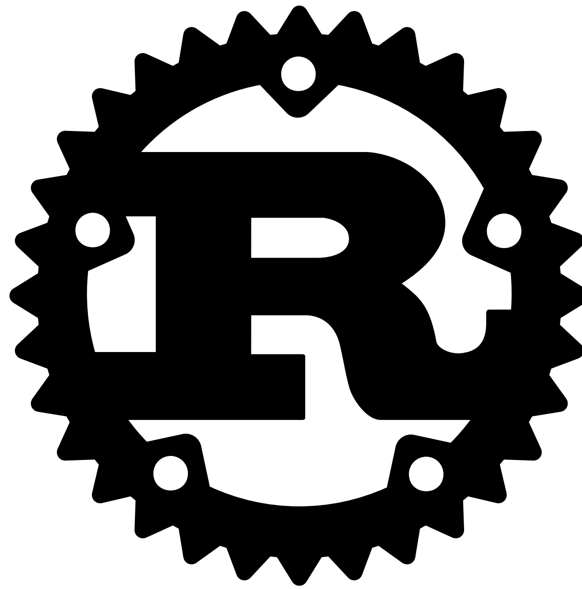
“

Rust is a *systems programming* language that runs blazingly fast, prevents segfaults, and guarantees thread safety.

– *the official Rust website*



> **Rust** is a systems programming language that runs *blazingly fast*, prevents segfaults, and guarantees thread safety.
>
> – *the official Rust website*



“

Rust is a systems programming language that runs blazingly fast, *prevents segfaults*, and *guarantees thread safety*.

– *the official Rust website*

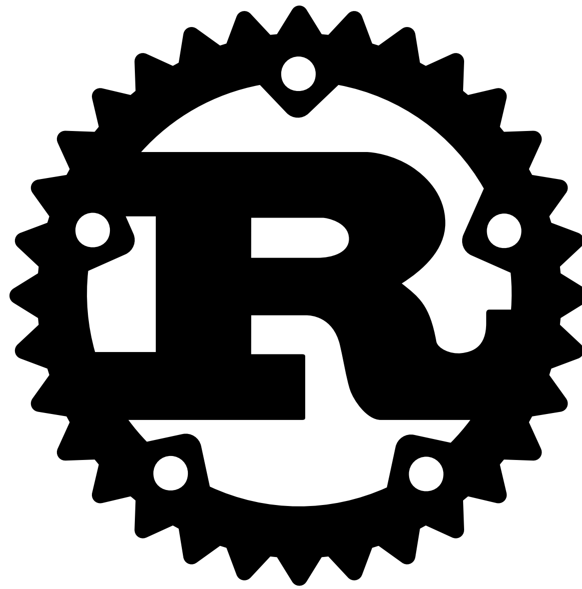
Memory safety without garbage collection

Abstraction without overhead

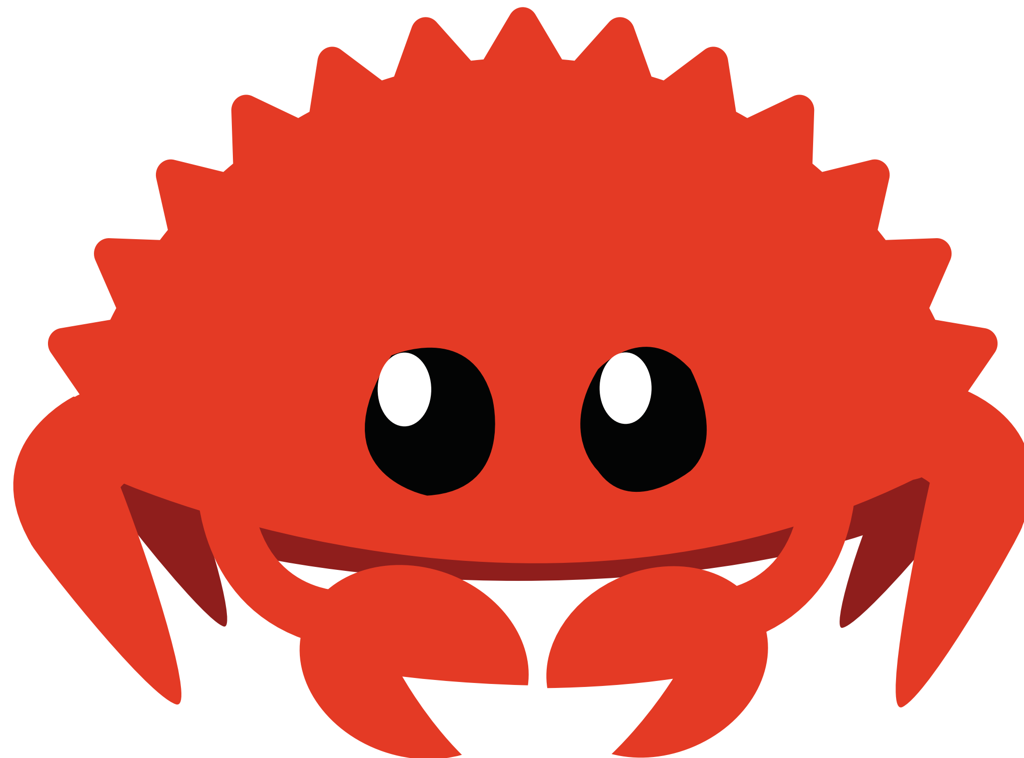
Concurrency without data races

Stability without stagnation

Hack without fear.



WE HAVE CUTE CRABS



... BUT HOW?

... BUT HOW?

Ownership

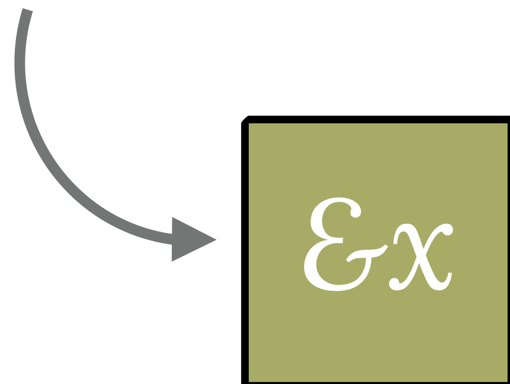
variables "*own*" the values they're bound to

... BUT HOW?

Ownership

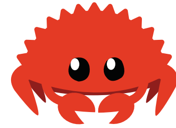
variables "*own*" the values they're bound to

Borrowing



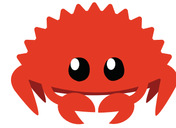
references "*borrow*" values from their owners

A PROGRAM IN RUST



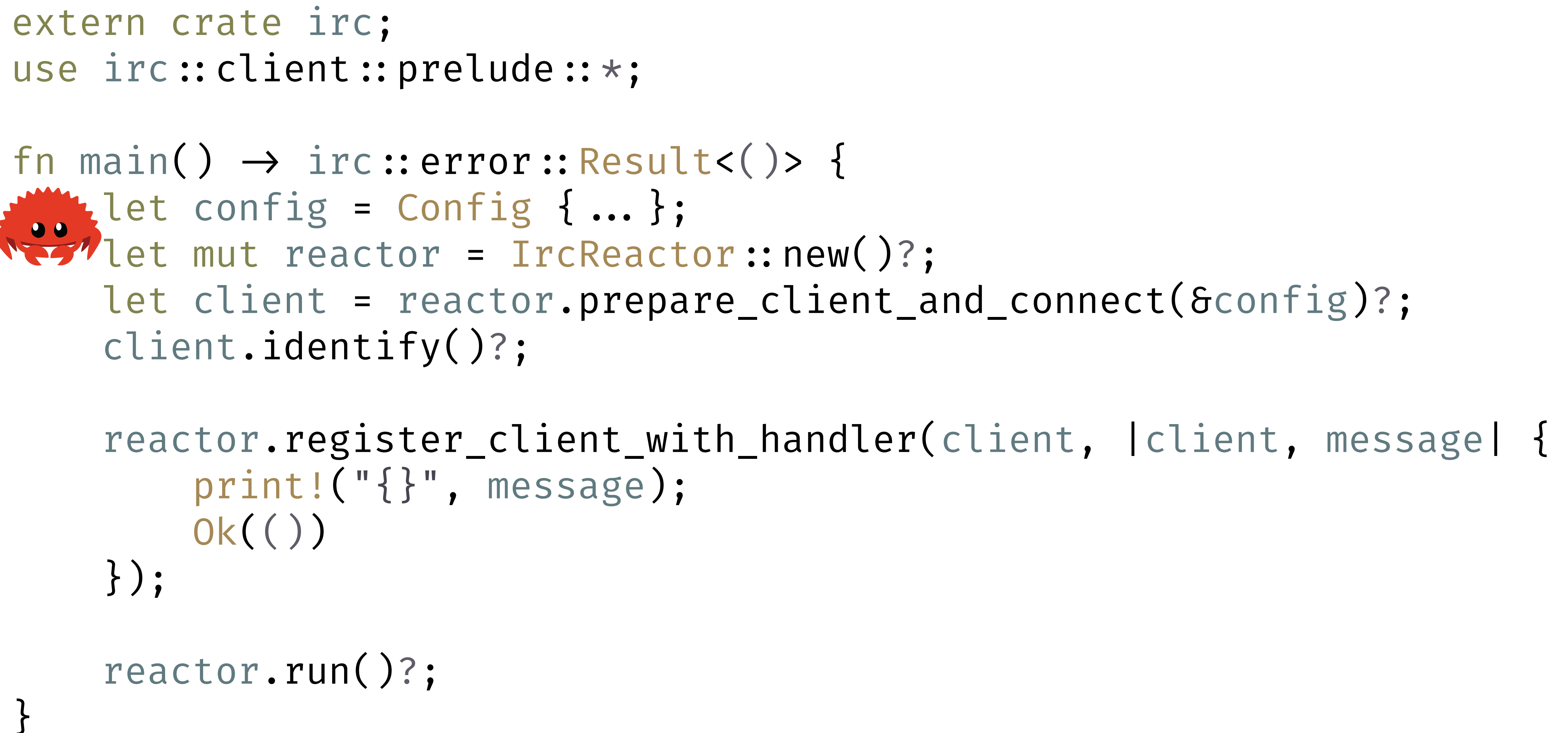
```rust
extern crate irc;
use irc::client::prelude::*;

fn main() -> irc::error::Result<()> {
    let config = Config { ... };
    let mut reactor = IrcReactor::new()?;
    let client = reactor.prepare_client_and_connect(&config)?;
    client.identify()?;

    reactor.register_client_with_handler(client, |client, message| {
        print!("{}", message);
        Ok(())
    });

    reactor.run()?;
}
```

A PROGRAM IN RUST



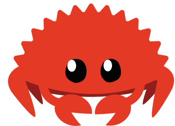
```rust
extern crate irc;
use irc::client::prelude::*;

fn main() -> irc::error::Result<()> {
    let config = Config { ... };
    let mut reactor = IrcReactor::new()?;
    let client = reactor.prepare_client_and_connect(&config)?;
    client.identify()?;

    reactor.register_client_with_handler(client, |client, message| {
        print!("{}", message);
        Ok(())
    });

    reactor.run()?;
}
```

A PROGRAM IN RUST



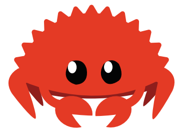
```rust
extern crate irc;
use irc::client::prelude::*;

fn main() -> irc::error::Result<()> {
    let config = Config { ... };
    let mut reactor = IrcReactor::new()?;
    let client = reactor.prepare_client_and_connect(&config)?;
    client.identify()?;

    reactor.register_client_with_handler(client, |client, message| {
        print!("{}", message);
        Ok(())
    });

    reactor.run()?;
}
```

A PROGRAM IN RUST



```
extern crate irc;
use irc::client::prelude::*;

fn main() -> irc::error::Result<()> {
    let config = Config { ... };
    let mut reactor = IrcReactor::new()?;
    let client = reactor.prepare_client_and_connect(&config)?;
    client.identify()?;

    reactor.register_client_with_handler(client, |client, message| {
        print!("{}", message);
        Ok(())
    });

    reactor.run()?;
}
```

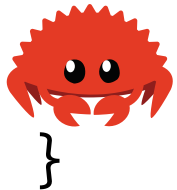
A PROGRAM IN RUST



```
extern crate irc;
use irc::client::prelude::*;

fn main() -> irc::error::Result<()> {
    let config = Config { ... };
    let mut reactor = IrcReactor::new()?;
    let client = reactor.prepare_client_and_connect(&config)?;
    client.identify()?;

    reactor.register_client_with_handler(client, |client, message| {
        print!("{}", message);
        Ok(())
    });

    reactor.run()?;
}
```

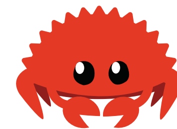
A PROGRAM IN RUST

```rust
extern crate irc;
use irc::client::prelude::*;

fn main() -> irc::error::Result<()> {
    let config = Config { ... };
    let mut reactor = IrcReactor::new()?;
    let client = reactor.prepare_client_and_connect(&config)?;
    client.identify()?;

    reactor.register_client_with_handler(client, |client, message| {
        print!("{}", message);
        Ok(())
    });

    reactor.run()?;
}
```

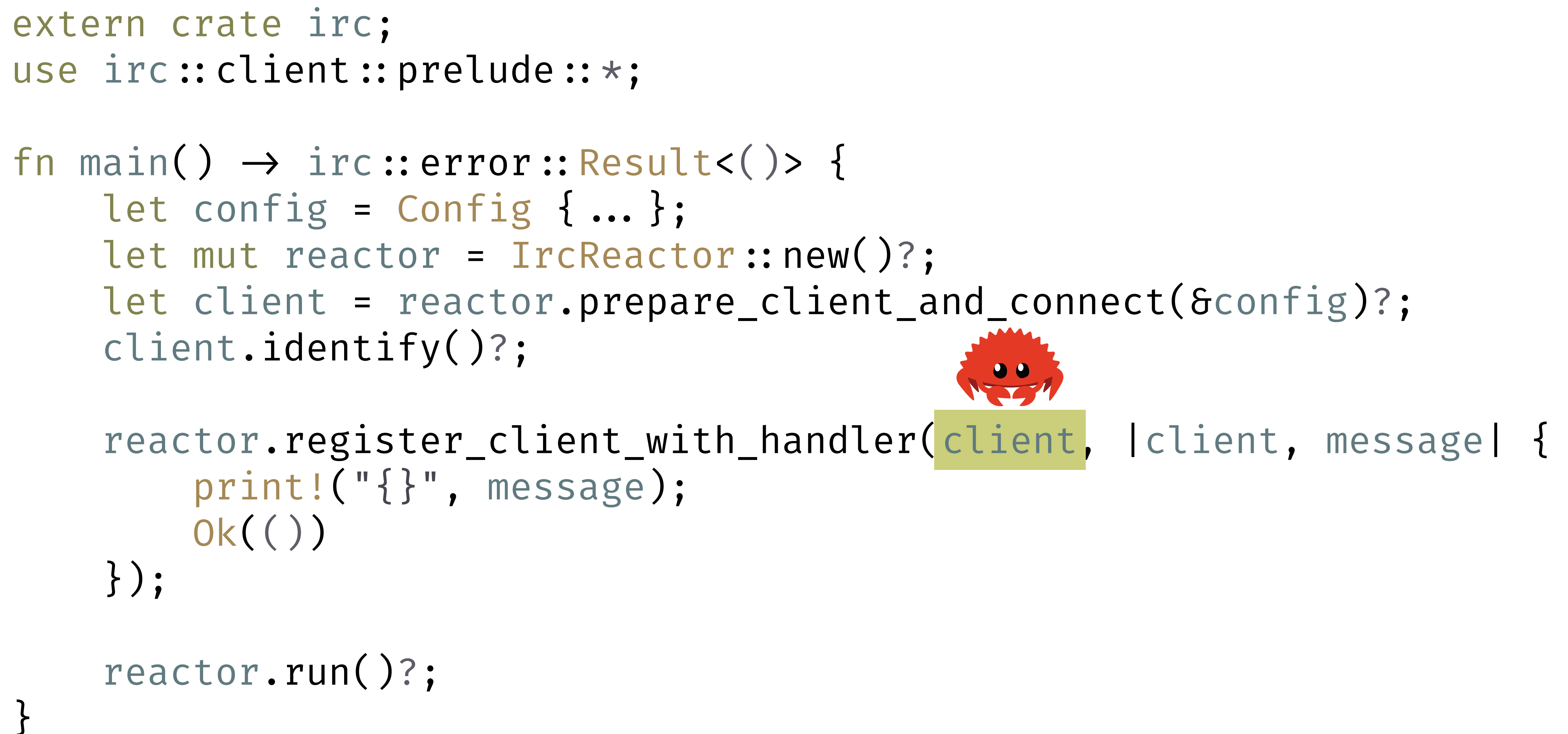


A PROGRAM IN RUST

```
extern crate irc;
use irc::client::prelude::*;

fn main() -> irc::error::Result<()> {
    let config = Config { ... };
    let mut reactor = IrcReactor::new()?;
    let client = reactor.prepare_client_and_connect(&config)?;
    client.identify()?;

    reactor.register_client_with_handler(client, |client, message| {
        print!("{}", message);
        Ok(())
    });

    reactor.run()?;
}
```



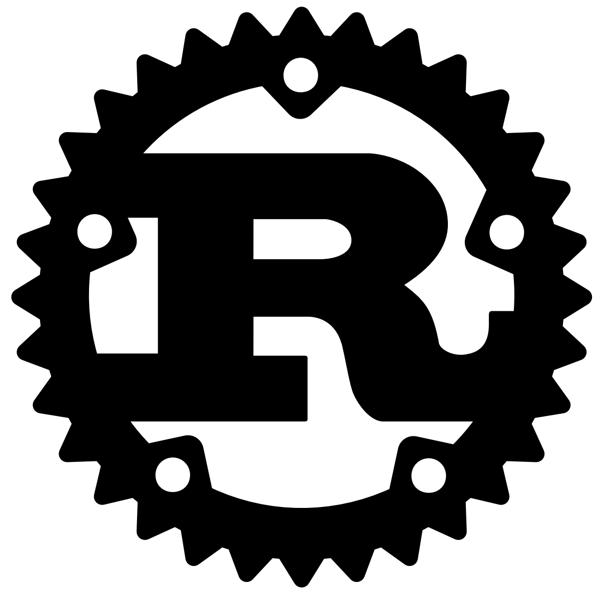
A PROGRAM IN RUST



```
extern crate irc;
use irc::client::prelude::*;

fn main() -> irc::error::Result<()> {
    let config = Config { ... };
    let mut reactor = IrcReactor::new()?;
    let client = reactor.prepare_client_and_connect(&config)?;
    client.identify()?;

    reactor.register_client_with_handler(client, |client, message| {
        print!("{}", message);
        Ok(())
    });

    reactor.run()?;
}
```

THE CURRENT STATE OF AFFAIRS

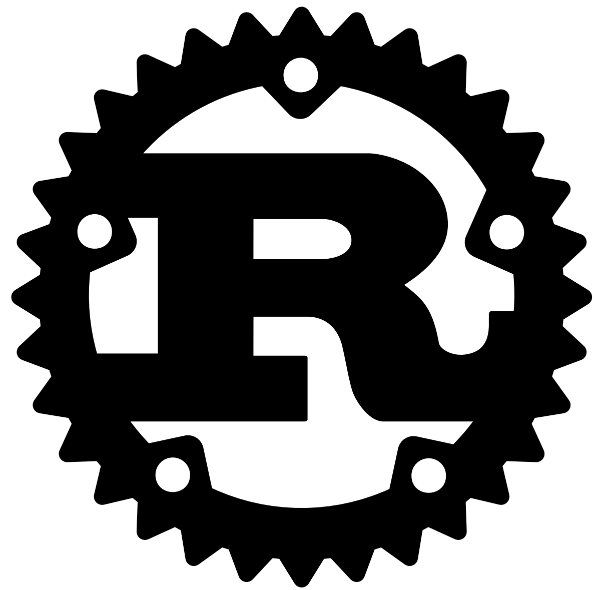
THE CURRENT STATE OF AFFAIRS



RUST

interprocedural static analysis
with *ad-hoc* constraint solving

THE CURRENT STATE OF AFFAIRS



RUST

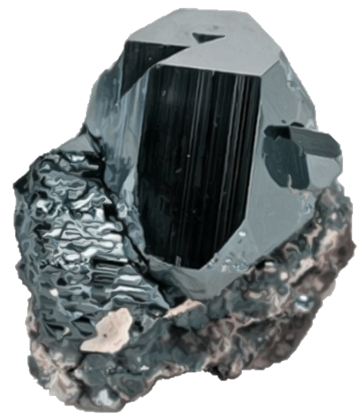
interprocedural static analysis
with *ad-hoc* constraint solving

RUSTBELT (JUNG, JOURDAN, KREBBERS, AND DREYER, POPL '18)

formal language specified in *Iris*
but low-level, in a *CPS-style*.

BUT WE WANT TO GO HIGHER

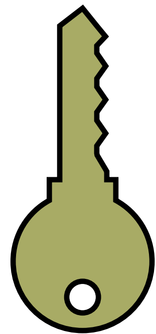
BUT WE WANT TO GO HIGHER

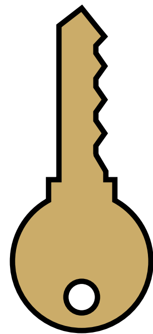


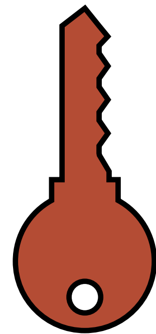
Oxide

CAPABILITIES FOR OWNERSHIP

CAPABILITIES FOR OWNERSHIP

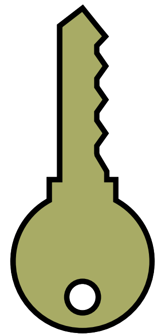


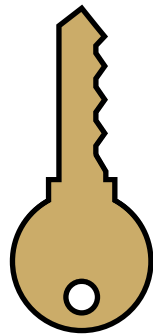


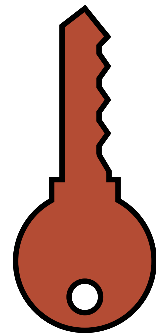


capabilities guard the use of identifiers

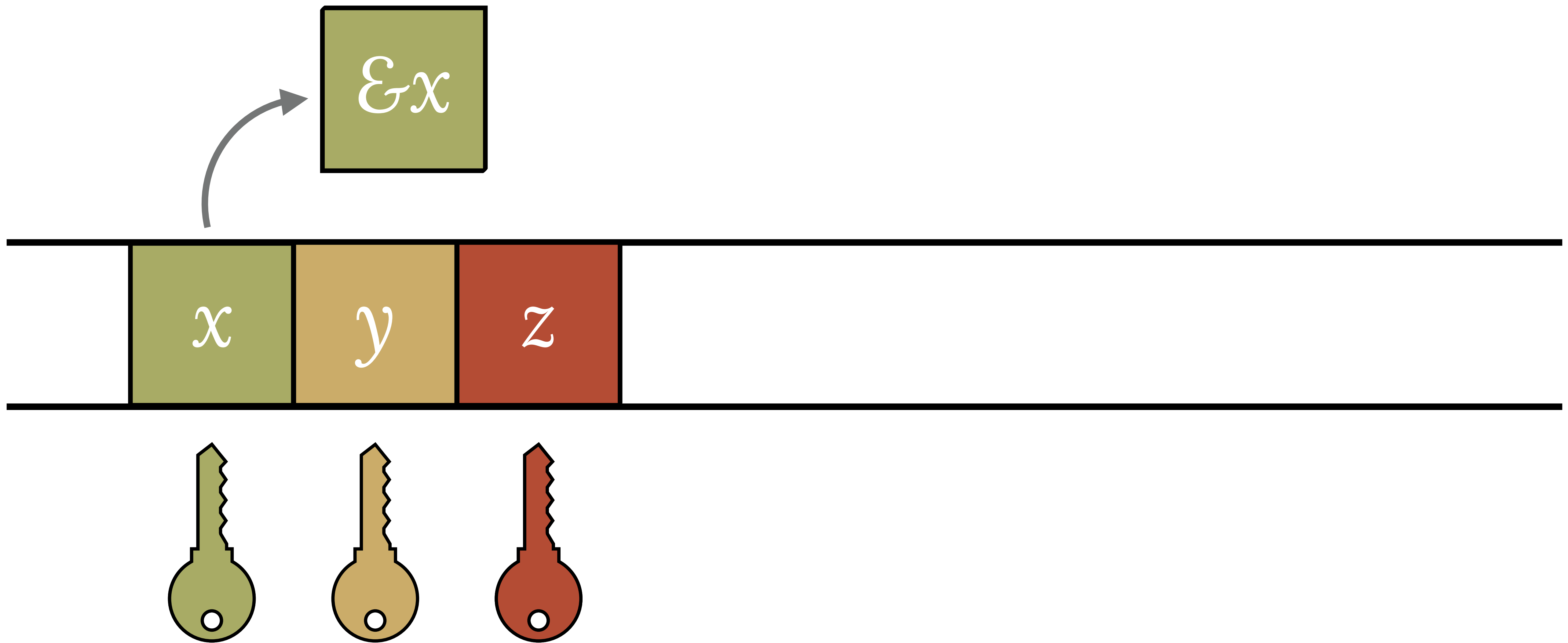
BORROWS BREAK CAPABILITIES INTO FRACTIONS



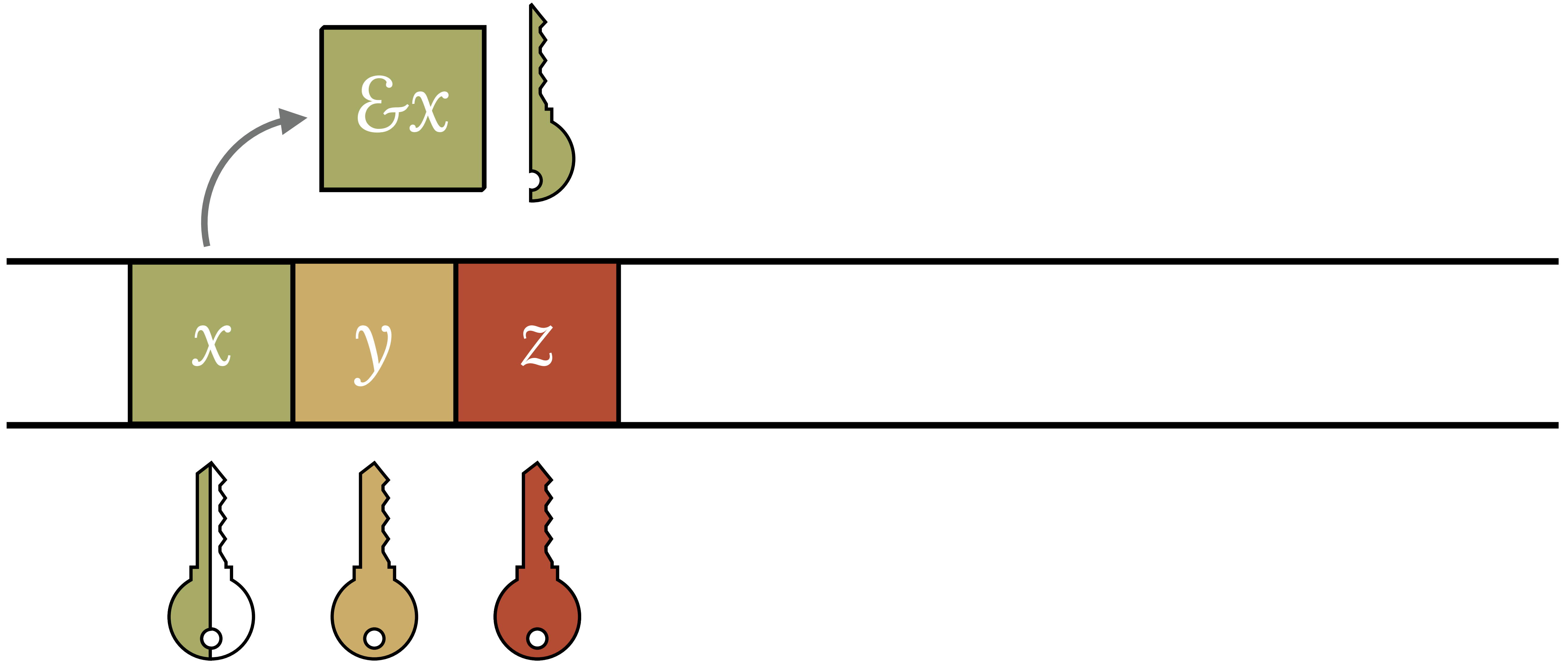




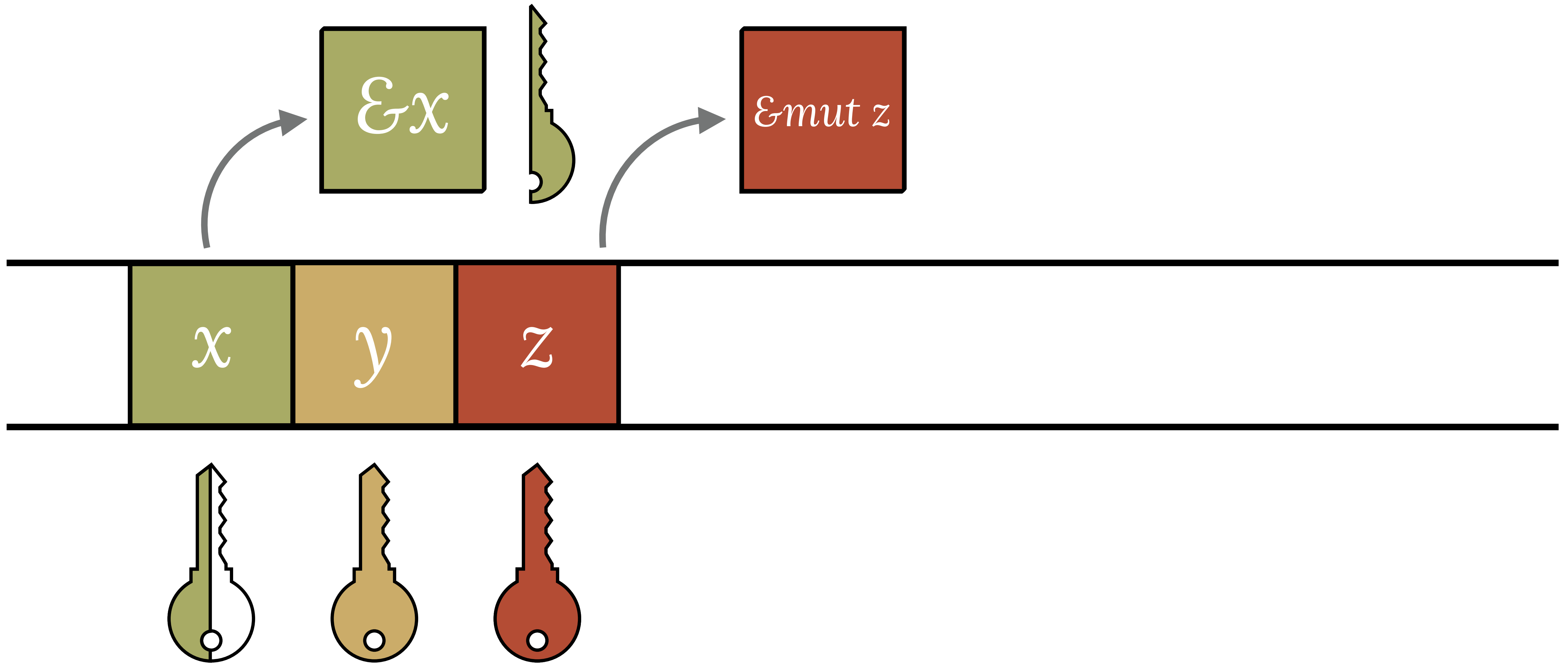
BORROWS BREAK CAPABILITIES INTO FRACTIONS



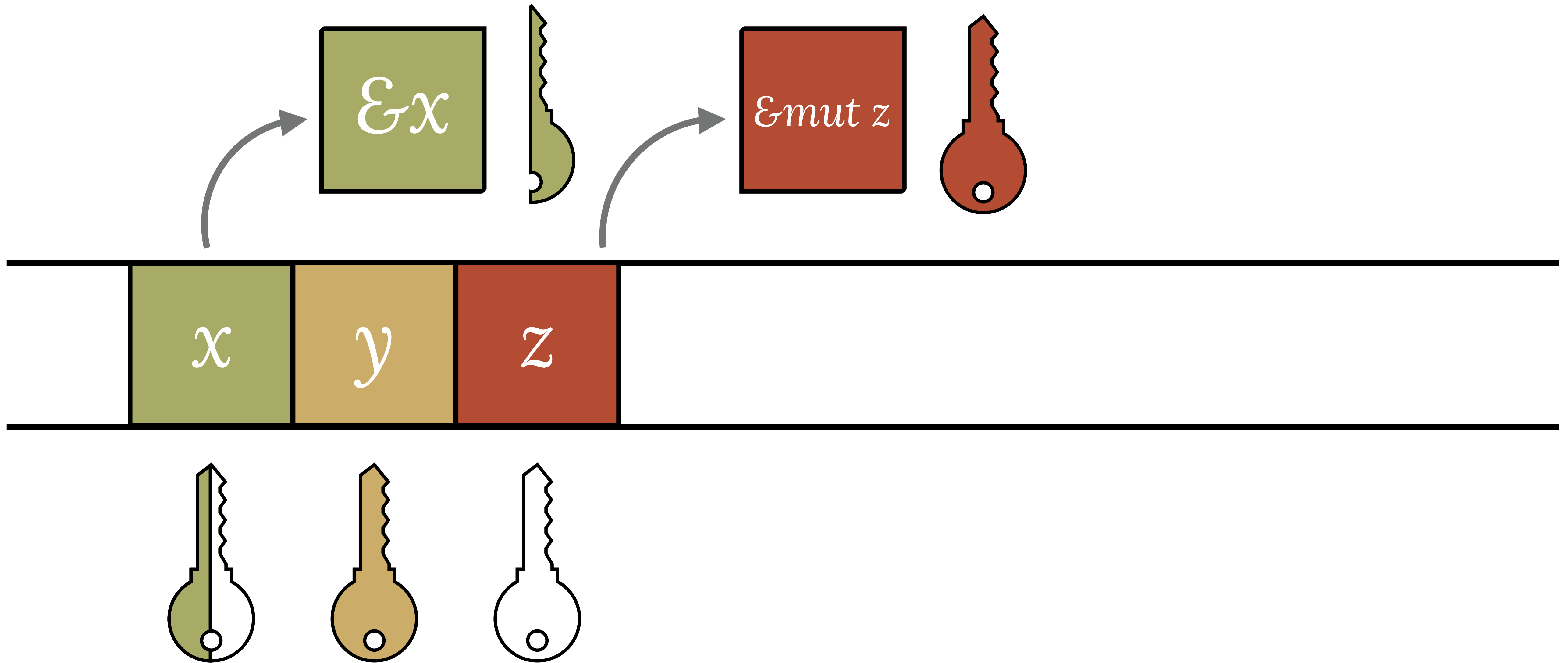
BORROWS BREAK CAPABILITIES INTO FRACTIONS



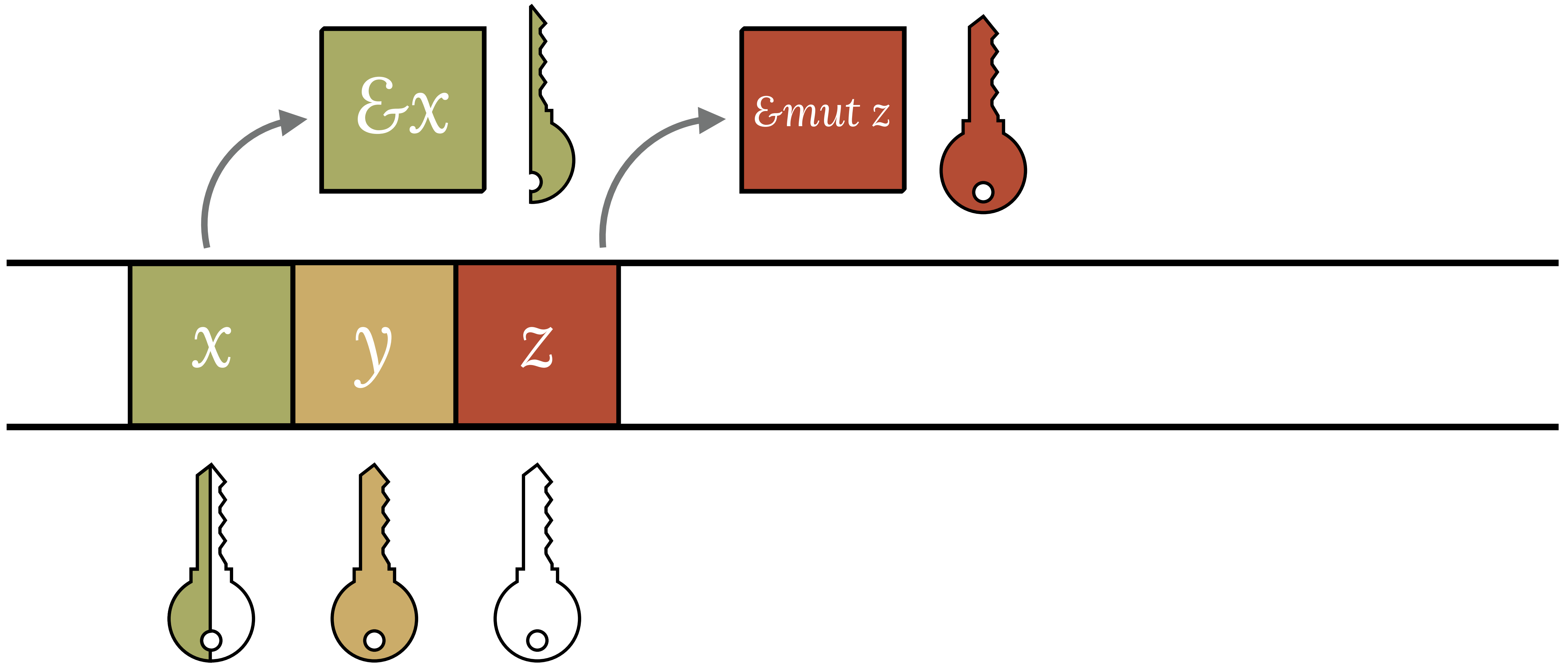
BORROWS BREAK CAPABILITIES INTO FRACTIONS



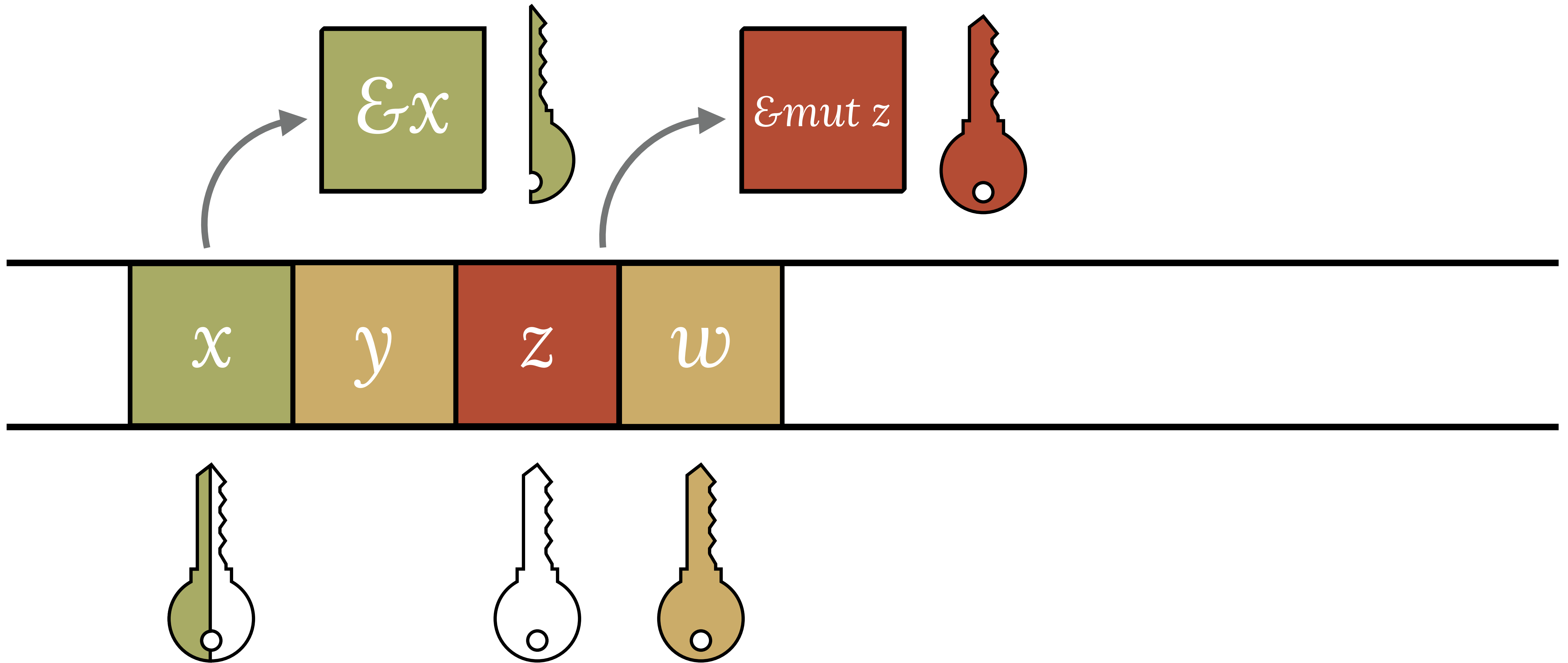
BORROWS BREAK CAPABILITIES INTO FRACTIONS



MOVES TAKE THE CAPABILITY AND THE HOLE



MOVES TAKE THE CAPABILITY AND THE HOLE



WE CALL REFERENCE SITES LOANS

```
extern crate irc;
use irc::client::prelude::*;

fn main() -> irc::error::Result<()> {
    let config = Config { ... };
    let mut reactor = IrcReactor::new()?;
    let client = reactor.prepare_client_and_connect(&config)?;
    client.identify()?;

    reactor.register_client_with_handler(client, |client, message| {
        print!("{}", message);
        Ok(())
    });

    reactor.run()?;
}
```

a loan

WHAT ABOUT LIFETIMES?

WHAT ABOUT LIFETIMES?

```
x : u32
```

WHAT ABOUT LIFETIMES?

```
x : u32

&x : &'x u32
```

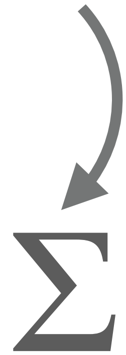
WHAT ABOUT LIFETIMES?

```
x : u32

&x : &'x u32
```

regions represent sets of loans

TYPE CHECKING

TYPE CHECKING



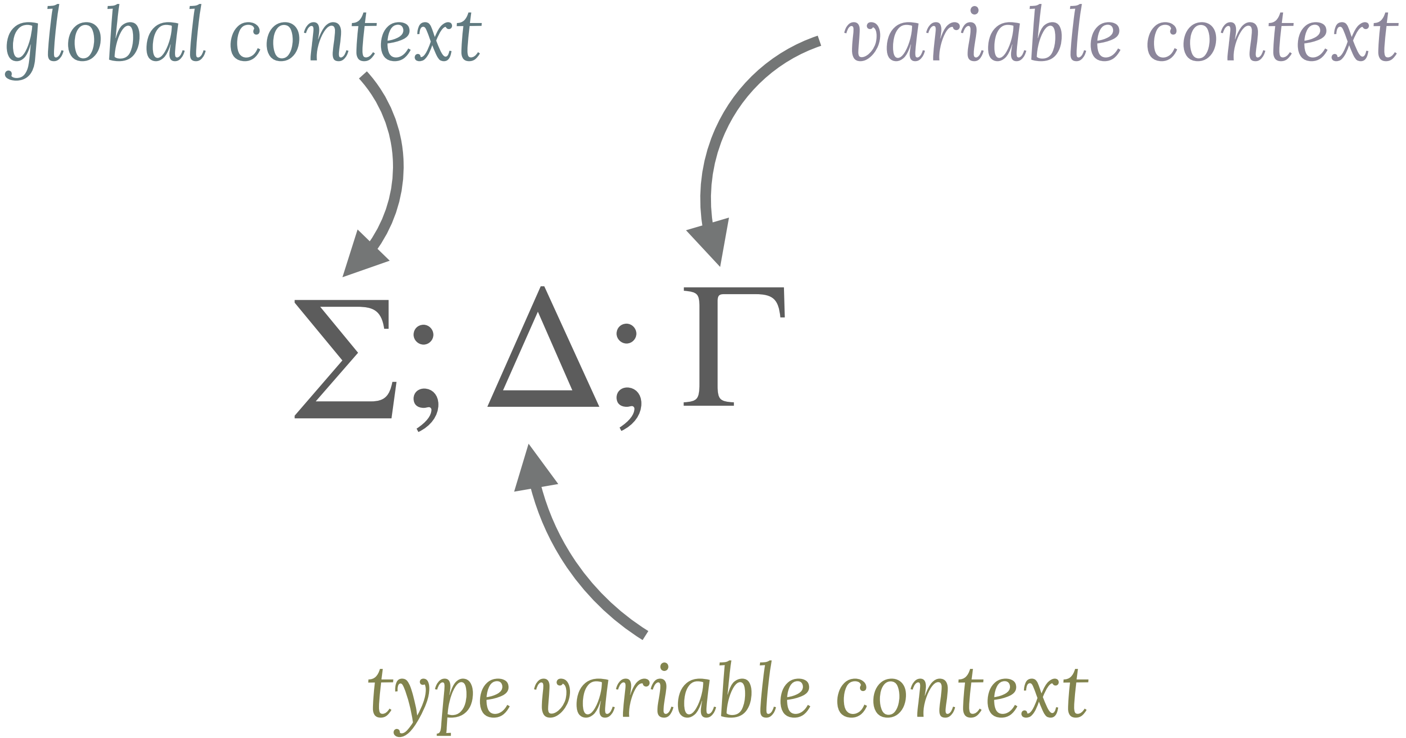
TYPE CHECKING

global context

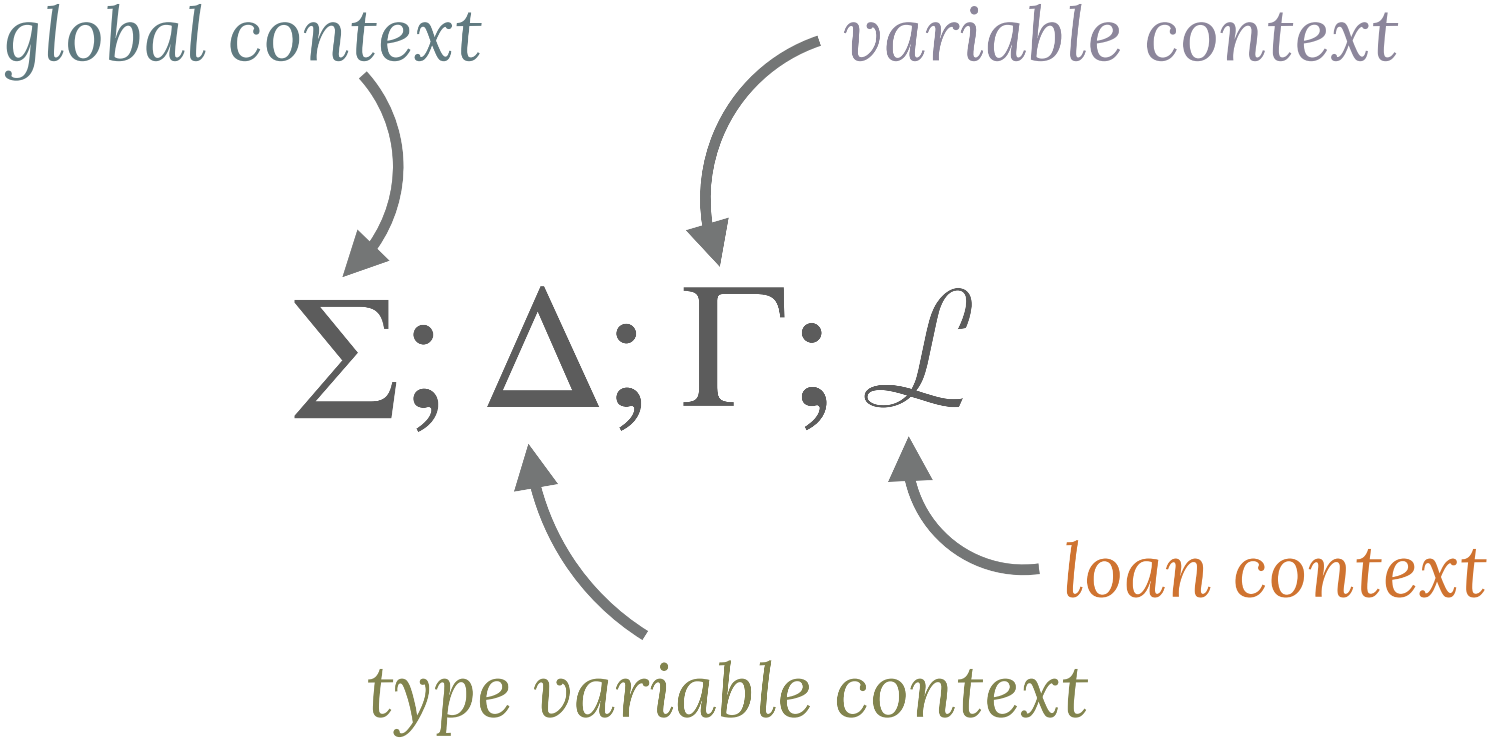
$$\Sigma; \Delta$$

type variable context

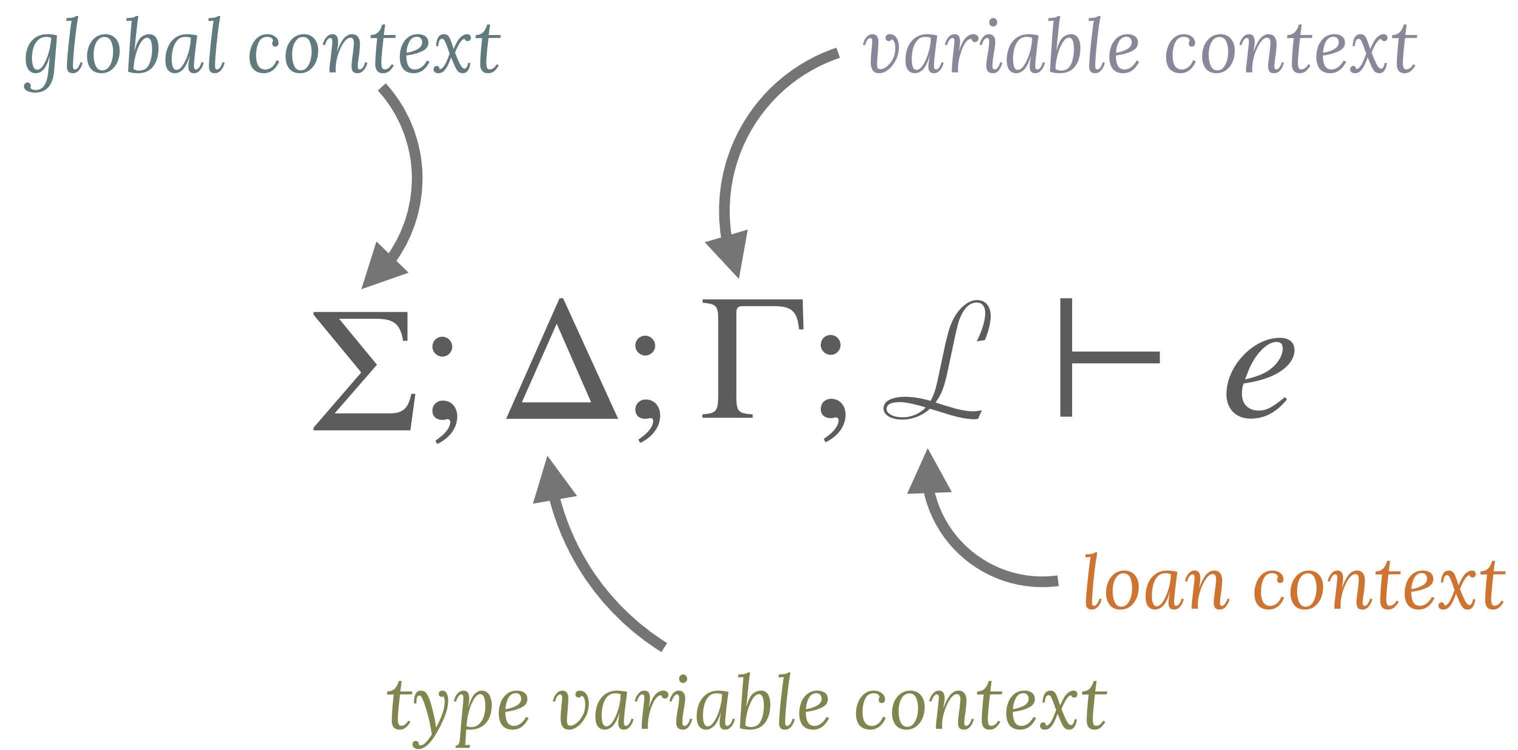
TYPE CHECKING



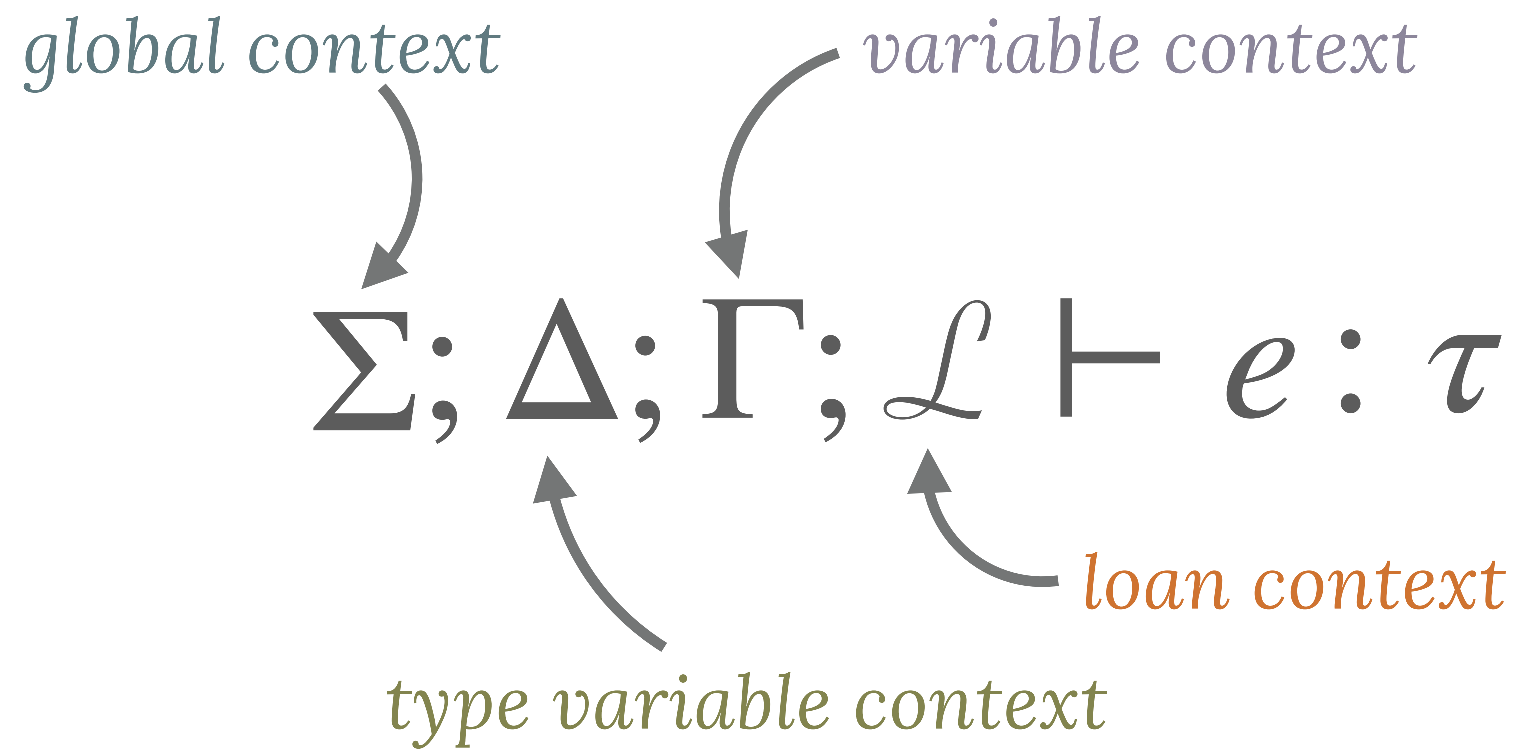
TYPE CHECKING



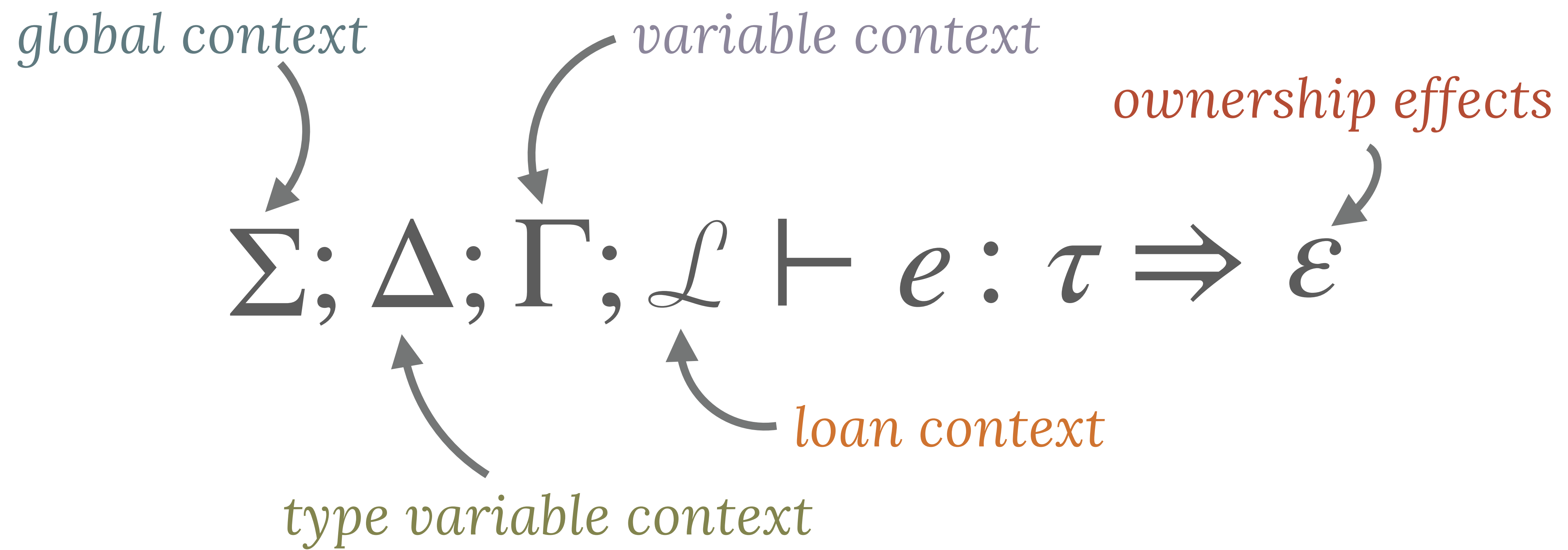
TYPE CHECKING



TYPE CHECKING



TYPE CHECKING



TYPING BORROWS

$$\frac{}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \texttt{\&'a x} :}$$

TYPING BORROWS

$$\frac{'a \notin \mathcal{L}}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \&'a\ x :}$$

TYPING BORROWS

$$\frac{\text{'a} \notin \mathcal{L} \qquad \Gamma \vdash \text{x} :_f \tau}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \&\text{'a x} :}$$

TYPING BORROWS

$$\frac{\texttt{'a} \notin \mathcal{L} \quad \Gamma \vdash \texttt{x} :_f \tau \quad f \neq 0}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \texttt{\&'a x} :}$$

TYPING BORROWS

$$\frac{\text{'a} \notin \mathcal{L} \qquad \Gamma \vdash \texttt{x} :_f \tau \qquad f \neq 0}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \texttt{\&'a x} : \texttt{\&\{'a\}}\ \tau}$$

TYPING BORROWS

$$\frac{\texttt{'a} \notin \mathcal{L} \qquad \Gamma \vdash \texttt{x} :_f \tau \qquad f \neq 0}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \texttt{\&'a x} : \texttt{\&\{'a\}}\ \tau \Rightarrow \texttt{borrow imm x as 'a}}$$

TYPING BRANCHING

$$\frac{}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \texttt{if } e_1 \{e_2\} \texttt{ else } \{e_3\} :}$$

TYPING BRANCHING

$$\Sigma; \Delta; \Gamma; \mathcal{L} \vdash e_1 : \texttt{bool} \Rightarrow \varepsilon_1$$

$$\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \texttt{if}\ e_1\ \{e_2\}\ \texttt{else}\ \{e_3\} :$$

TYPING BRANCHING

$$\Sigma; \Delta; \Gamma; \mathcal{L} \vdash e_1 : \texttt{bool} \Rightarrow \varepsilon_1$$

$$\Sigma; \Delta; \varepsilon_1(\Gamma; \mathcal{L}) \vdash e_2 : \tau_2 \Rightarrow \varepsilon_2$$

$$\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \texttt{if}\ e_1 \{e_2\}\ \texttt{else}\ \{e_3\} :$$

TYPING BRANCHING

$$\Sigma; \Delta; \Gamma; \mathcal{L} \vdash e_1 : \texttt{bool} \Rightarrow \varepsilon_1$$

$$\Sigma; \Delta; \varepsilon_1(\Gamma; \mathcal{L}) \vdash e_2 : \tau_2 \Rightarrow \varepsilon_2$$

$$\Sigma; \Delta; \varepsilon_1(\Gamma; \mathcal{L}) \vdash e_3 : \tau_3 \Rightarrow \varepsilon_3$$

$$\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \texttt{if}\ e_1 \{e_2\}\ \texttt{else}\ \{e_3\} :$$

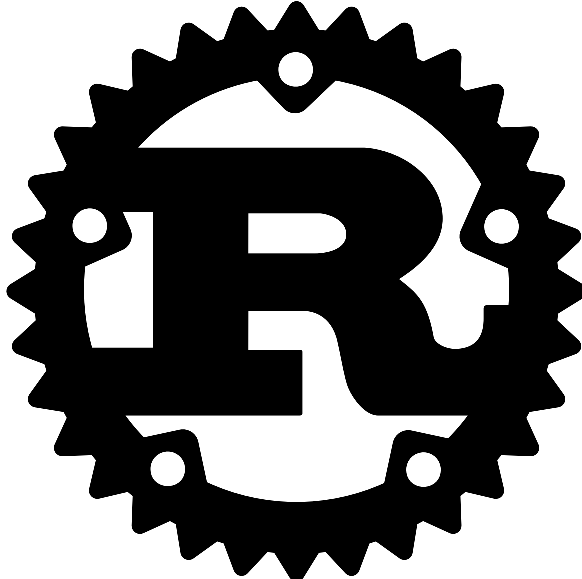
TYPING BRANCHING

$$\frac{\begin{array}{c}\Sigma; \Delta; \Gamma; \mathcal{L} \vdash e_1 : \texttt{bool} \Rightarrow \varepsilon_1 \\ \Sigma; \Delta; \varepsilon_1(\Gamma; \mathcal{L}) \vdash e_2 : \tau_2 \Rightarrow \varepsilon_2 \\ \Sigma; \Delta; \varepsilon_1(\Gamma; \mathcal{L}) \vdash e_3 : \tau_3 \Rightarrow \varepsilon_3 \\ \tau_2 \sim \tau_3 \Rightarrow \tau\end{array}}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \texttt{if}\ e_1\, \{e_2\}\ \texttt{else}\ \{e_3\} :}$$

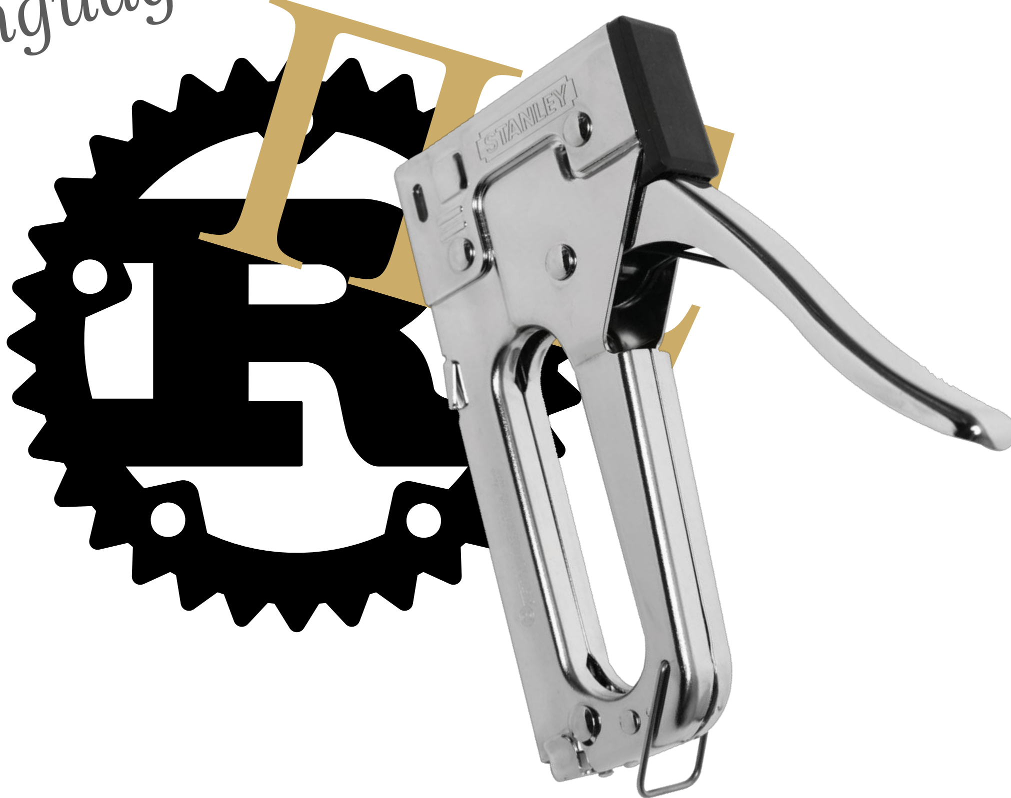
TYPING BRANCHING

$$\frac{\begin{array}{c}\Sigma; \Delta; \Gamma; \mathcal{L} \vdash e_1 : \texttt{bool} \Rightarrow \varepsilon_1 \\ \Sigma; \Delta; \varepsilon_1(\Gamma; \mathcal{L}) \vdash e_2 : \tau_2 \Rightarrow \varepsilon_2 \\ \Sigma; \Delta; \varepsilon_1(\Gamma; \mathcal{L}) \vdash e_3 : \tau_3 \Rightarrow \varepsilon_3 \\ \tau_2 \sim \tau_3 \Rightarrow \tau\end{array}}{\Sigma; \Delta; \Gamma; \mathcal{L} \vdash \texttt{if}\ e_1 \{e_2\}\ \texttt{else}\ \{e_3\} : \tau \Rightarrow \varepsilon_1, \varepsilon_2, \varepsilon_3}$$

A RUSTY FUTURE

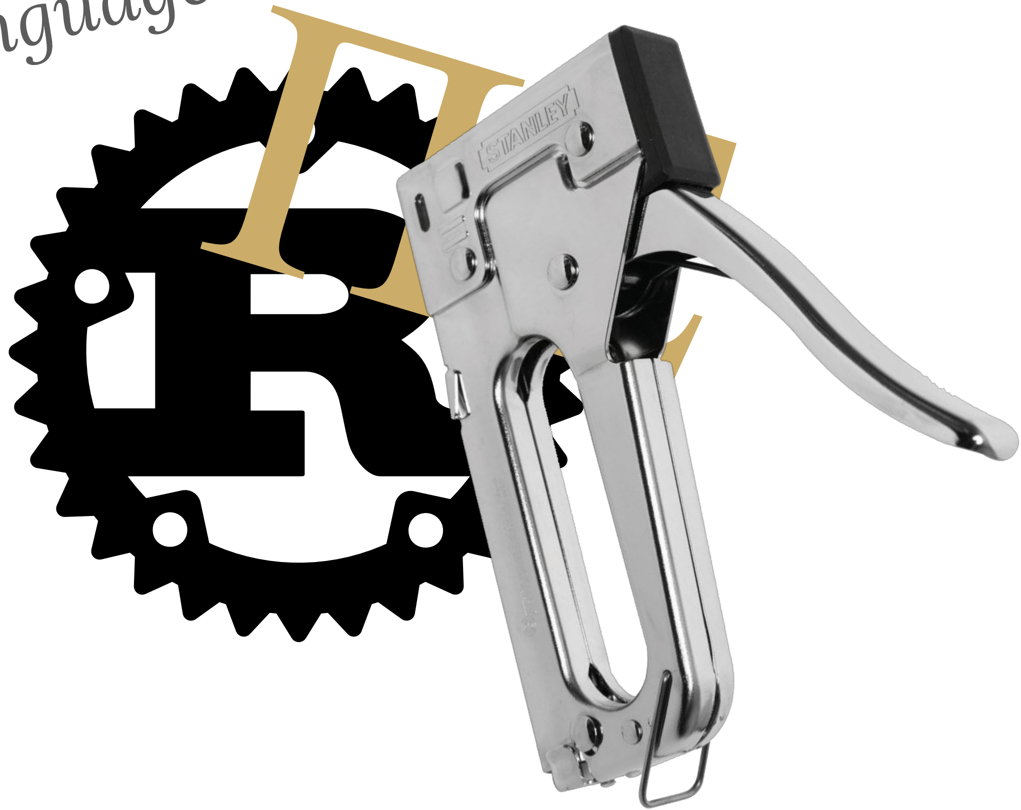


A RUSTY FUTURE



A RUSTY FUTURE

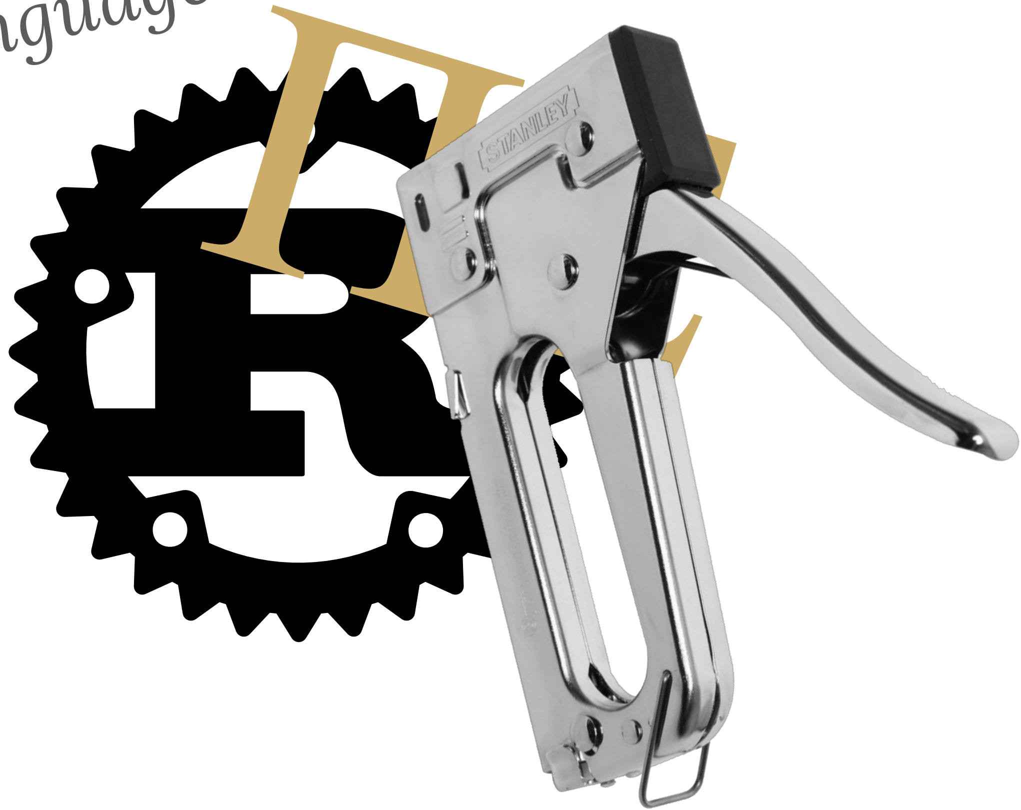
Language Extensions



Safe Interoperability

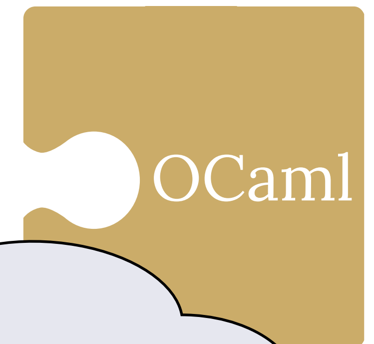
A RUSTY FUTURE

Language Extensions



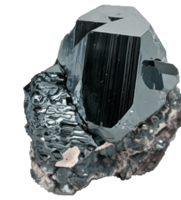
Unsafe Code Guidelines

Safe Interoperability

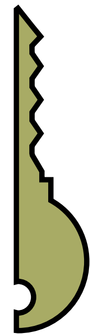


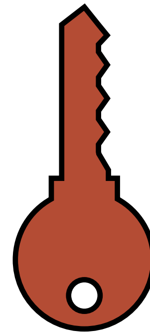
What unsafe code is safe to write?

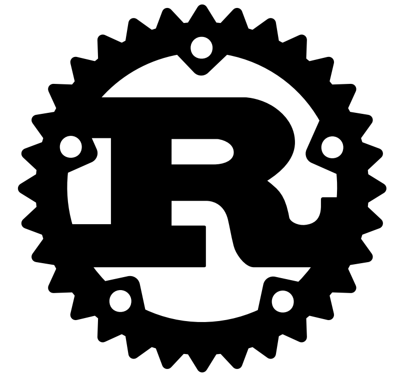
TAKEAWAYS

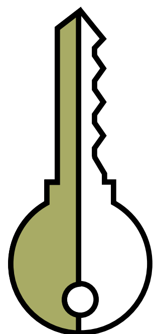


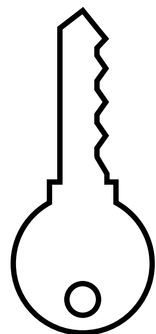
Oxide

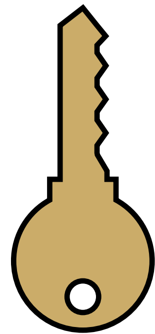


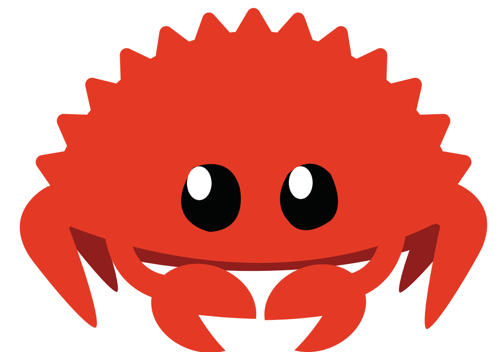




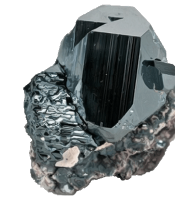






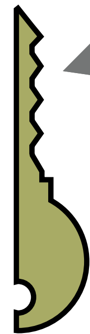


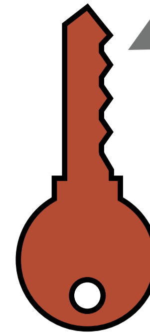
TAKEAWAYS

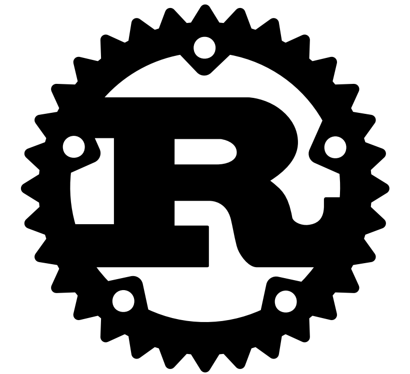


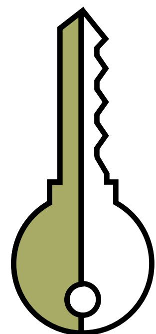
Oxide

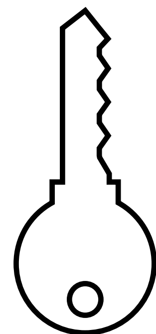
Ownership with fractional capabilities

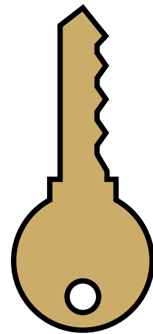


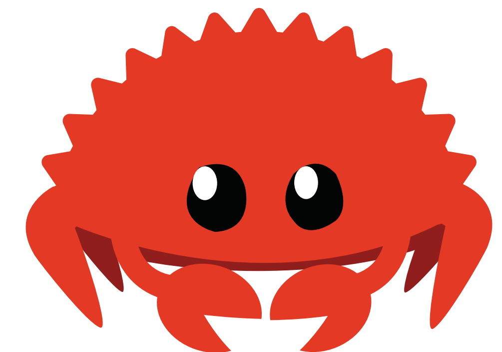




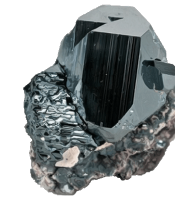






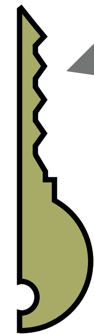


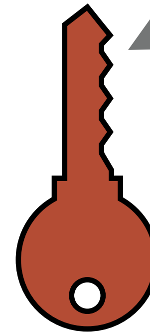
TAKEAWAYS

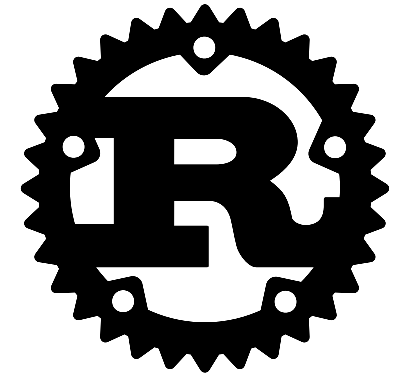


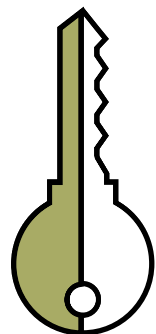
Oxide

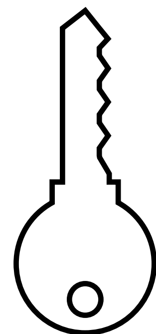
Ownership with *fractional* capabilities

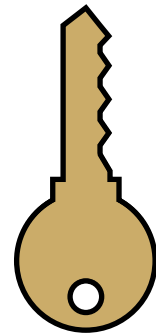


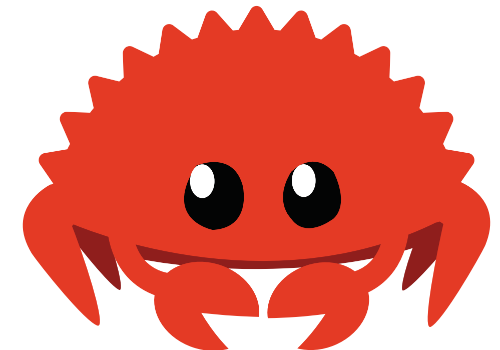






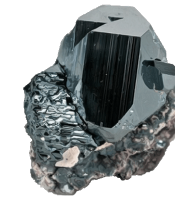






Moves *never* return their capability

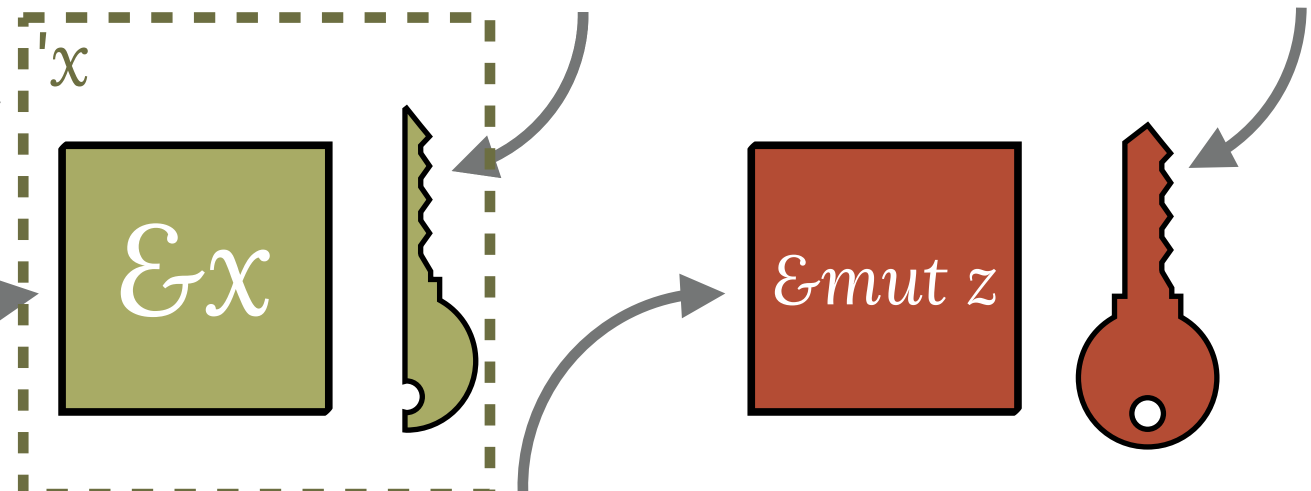
TAKEAWAYS

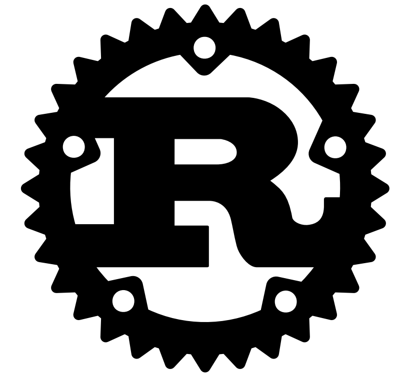


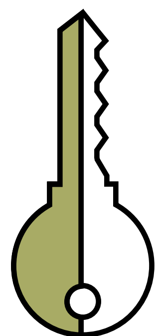
Oxide

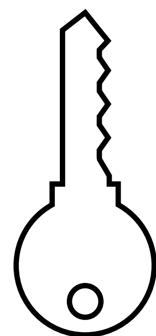
Regions are sets of loans

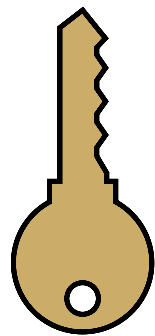
Ownership with fractional capabilities

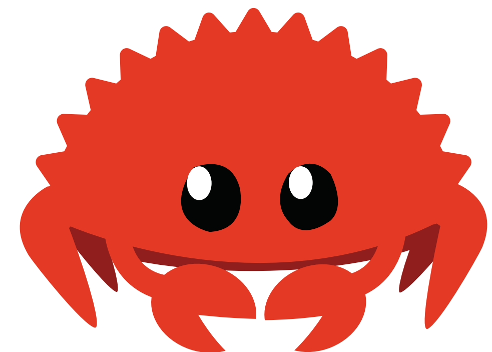












Moves never return their capability