# Oxide: The Essence of Rust

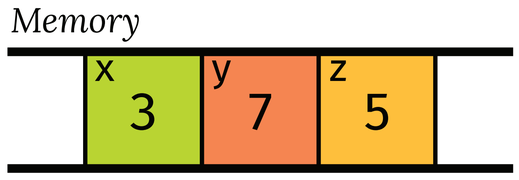
Aaron Weiss

## What is Rust?

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

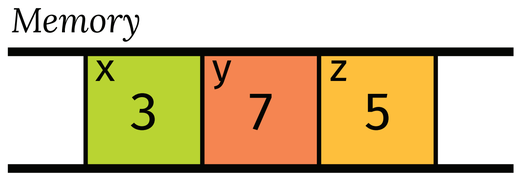
## What is the essence of Rust?

### ① Ownership




identifiers *own* the values they're bound to.
*e.g.* `x` *owns* the value `3`

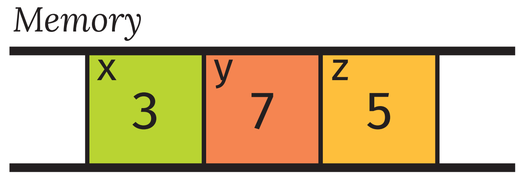
### ② Moves




```
let w = x;
```

using identifiers directly *moves* the values out of them

### ③ Borrows




```
let w = &x;
```

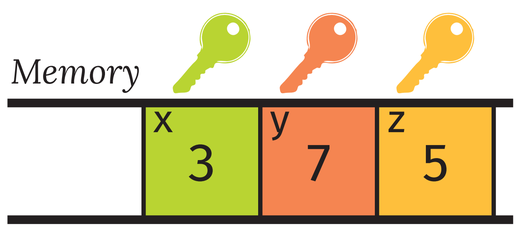
references *borrow* values from their owners

## What is Oxide?

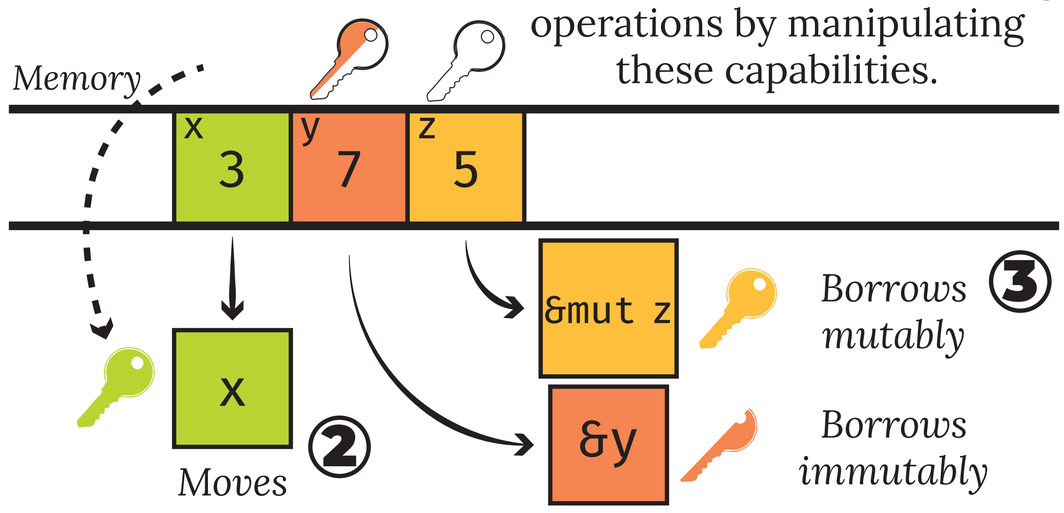
Oxide is a formal semantics of *the essence of Rust* which models a *core* language close to surface Rust, extended with *additional levels of key abstractions.*

## How does Oxide model it?

① To model *ownership*, we associate every identifier with a *fractional capability.*




We model ownership-affecting operations by manipulating these capabilities.




## By Example

**Rust**

```
fn option_as_mut<'a>(
  x: &'a mut Option<u32>
) -> Option<&'a mut u32> {
  match *x {
    None => None,
    Some(ref mut t) =>
      Some(t)
  }
}
```

**RustBelt**

```
funrec option_as_mut(x)
ret ret :=
 let r = new(2) in
 letcont k() :=
   delete(1, x);
   jump ret(r) in
 let y = *x in case *y of
 - r :=^0 (); jump k()
 - r :=^1 y.1; jump k()
```

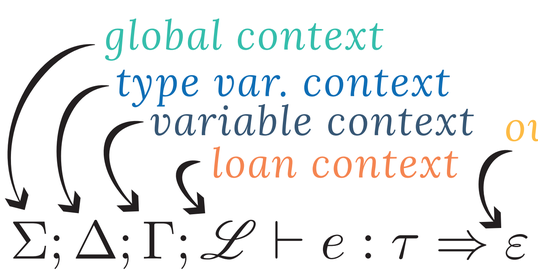
**Oxide**

```
fn option_as_mut<'a>(
  x: &'a mut Option<u32>
) -> Option<&'a mut u32> {
  match *x {
    None => None,
    Some(ref<'t> mut t) =>
      Some(t)
  }
}
```

## Oxide, formally...

Oxide uses a novel *type-and-effect* system to automatically track *fractional capabilities* representing *ownership.*

### Typing in Oxide




our *loan context* tracks which *loans* are *currently alive*

$\Gamma =$ `x` $:_1$ `&'a mut Option<u32>`,
`*x` $:_1$ `Option<u32>`,
`*x.0` $:_1$ `u32`
$\mathscr{L} =$ `'a` $\mapsto_1 \nabla$

$\varepsilon ::= \text{borrow } \mu\ \pi \text{ as } \ell \mid \text{drop } \pi \mid \varepsilon_1, \varepsilon_2 \mid \diamond$

*ownership effects* record how our *loan context* changes

```
fn option_as_mut<'a>(
  x: &'a mut Option<u32>
) -> Option<&'a mut u32> {
[…]
  match *x {
[Γ ∪ *x :_0 Option<u32>]
    None => None,
    Some(ref<'t> mut t) =>
[Γ ∪ t :_1 &{'t} mut u32, 𝓛 ∪ 't ↦_1 *x]
      Some(t)
[Γ ∪ t :_0 &{'t} mut u32]
  } }
```

## Selected Typing Rules

T-Move
$$\frac{\Gamma \vdash \pi :_1 \tau}{\Sigma; \Delta; \Gamma; \mathscr{L} \vdash \pi : \tau \Rightarrow \text{drop } \pi}$$

T-Copy
$$\frac{\Gamma \vdash \pi :_f \tau \qquad \text{copyable } \tau}{\Sigma; \Delta; \Gamma; \mathscr{L} \vdash \pi : \tau \Rightarrow \diamond}$$

T-BorrowImm
$$\frac{\ell \notin \mathscr{L} \qquad \Gamma \vdash \pi :_f \tau \qquad f \neq 0}{\Sigma; \Delta; \Gamma; \mathscr{L} \vdash \&\ell \text{ imm } \pi : \&\{\ell\} \text{ imm } \tau \Rightarrow \text{borrow imm } \pi \text{ as } \ell}$$

T-BorrowMut
$$\frac{\ell \notin \mathscr{L} \qquad \Gamma \vdash \pi :_1 \tau}{\Sigma; \Delta; \Gamma; \mathscr{L} \vdash \&\ell \text{ mut } \pi : \&\{\ell\} \text{ mut } \tau \Rightarrow \text{borrow mut } \pi \text{ as } \ell}$$

T-Seq
$$\frac{\Sigma; \Delta; \Gamma; \mathscr{L} \vdash e_1 : \tau_1 \Rightarrow \varepsilon_1 \qquad \Sigma; \Delta; \varepsilon_1(\Gamma; \mathscr{L}) \vdash e_2 : \tau_2 \Rightarrow \varepsilon_2}{\Sigma; \Delta; \Gamma; \mathscr{L} \vdash e_1; e_2 : \tau_2 \Rightarrow \varepsilon_1, \varepsilon_2}$$

T-Branch
$$\frac{\Sigma; \Delta; \Gamma; \mathscr{L} \vdash e_1 : \texttt{bool} \Rightarrow \varepsilon_1 \qquad \Sigma; \Delta; \varepsilon_1(\Gamma; \mathscr{L}) \vdash e_2 : \tau_2 \Rightarrow \varepsilon_2 \qquad \Sigma; \Delta; \varepsilon_1(\Gamma; \mathscr{L}) \vdash e_3 : \tau_3 \Rightarrow \varepsilon_3 \qquad \tau_2 \sim \tau_3 \Rightarrow \tau}{\Sigma; \Delta; \Gamma; \mathscr{L} \vdash \text{if } e_1\{e_2\} \text{ else } \{e_3\} : \tau \Rightarrow \varepsilon_1, \varepsilon_2, \varepsilon_3}$$

@aatxe

https://aaronweiss.us