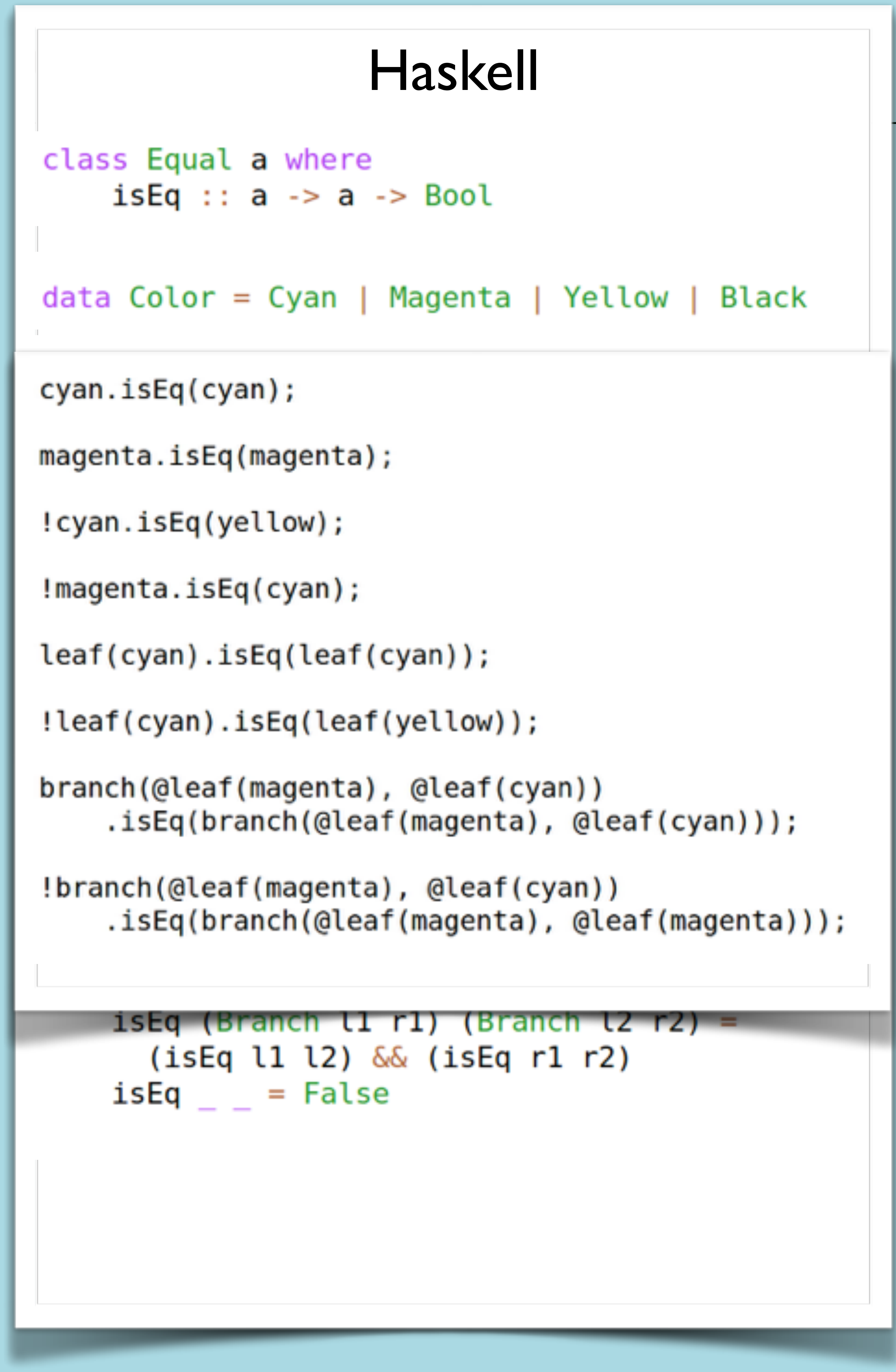
Rust typeclasses turn *trait*-er

Lindsey Kuper
Mozilla Research
August 9, 2012

What’s Rust?

A systems language
pursuing the trifecta:
fast, concurrent, safe

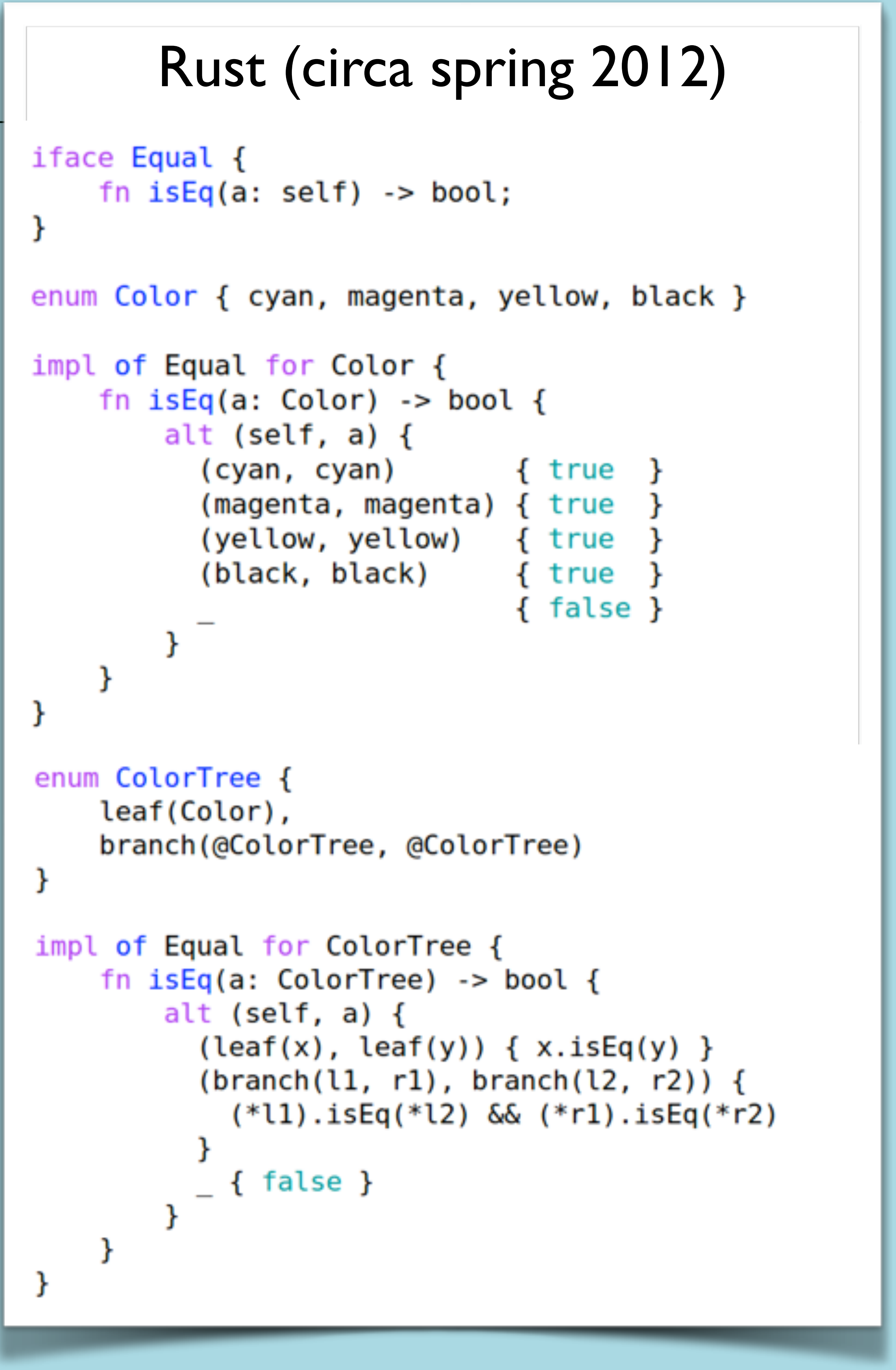


Haskell

```haskell
class Equal a where
    isEq :: a -> a -> Bool

data Color = Cyan | Magenta | Yellow | Black
```

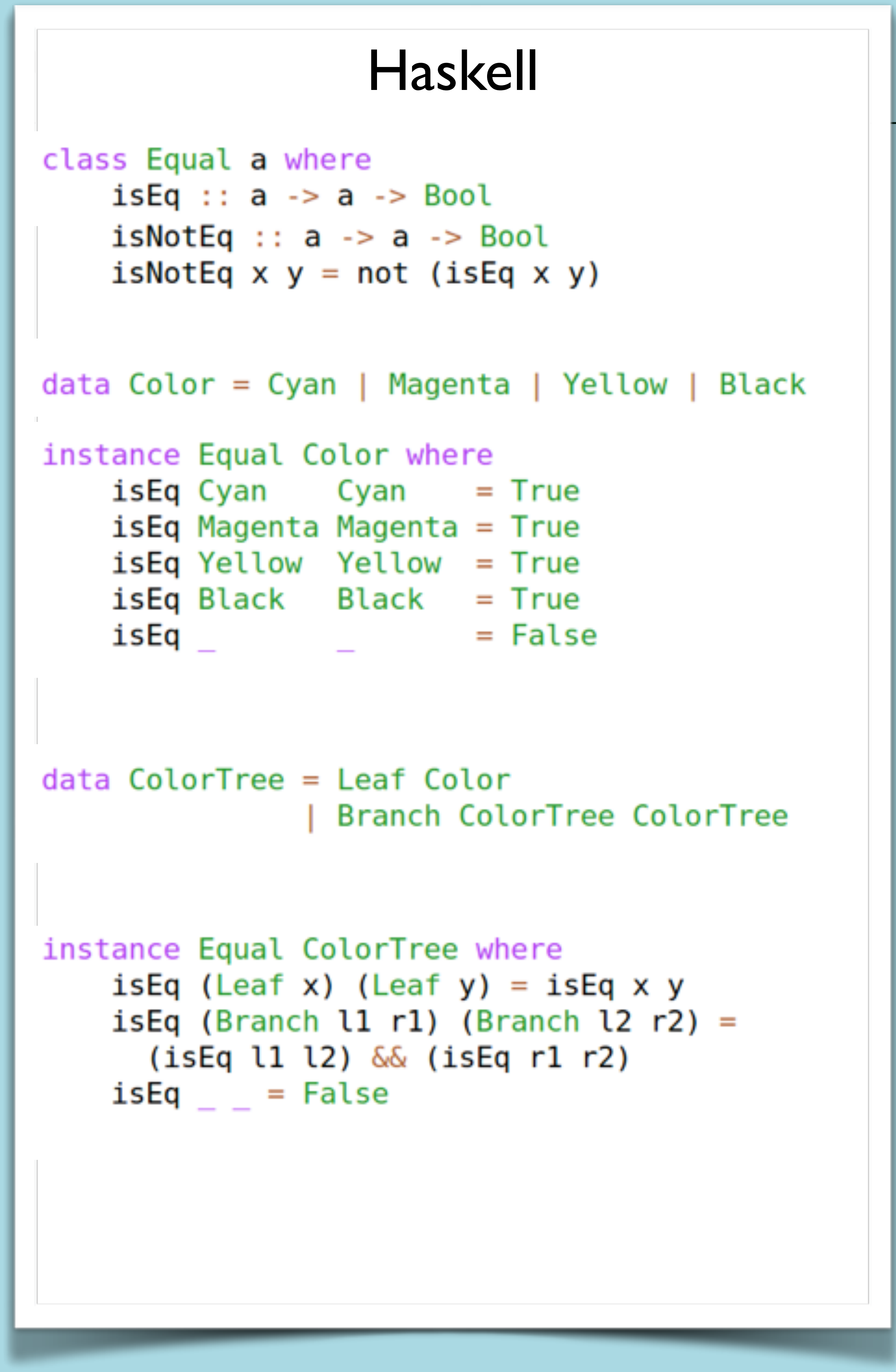
```rust
cyan.isEq(cyan);

magenta.isEq(magenta);

!cyan.isEq(yellow);

!magenta.isEq(cyan);

leaf(cyan).isEq(leaf(cyan));

!leaf(cyan).isEq(leaf(yellow));

branch(@leaf(magenta), @leaf(cyan))
    .isEq(branch(@leaf(magenta), @leaf(cyan)));

!branch(@leaf(magenta), @leaf(cyan))
    .isEq(branch(@leaf(magenta), @leaf(magenta)));
```

```haskell
    isEq (Branch l1 r1) (Branch l2 r2) =
      (isEq l1 l2) && (isEq r1 r2)
    isEq _ _ = False
```



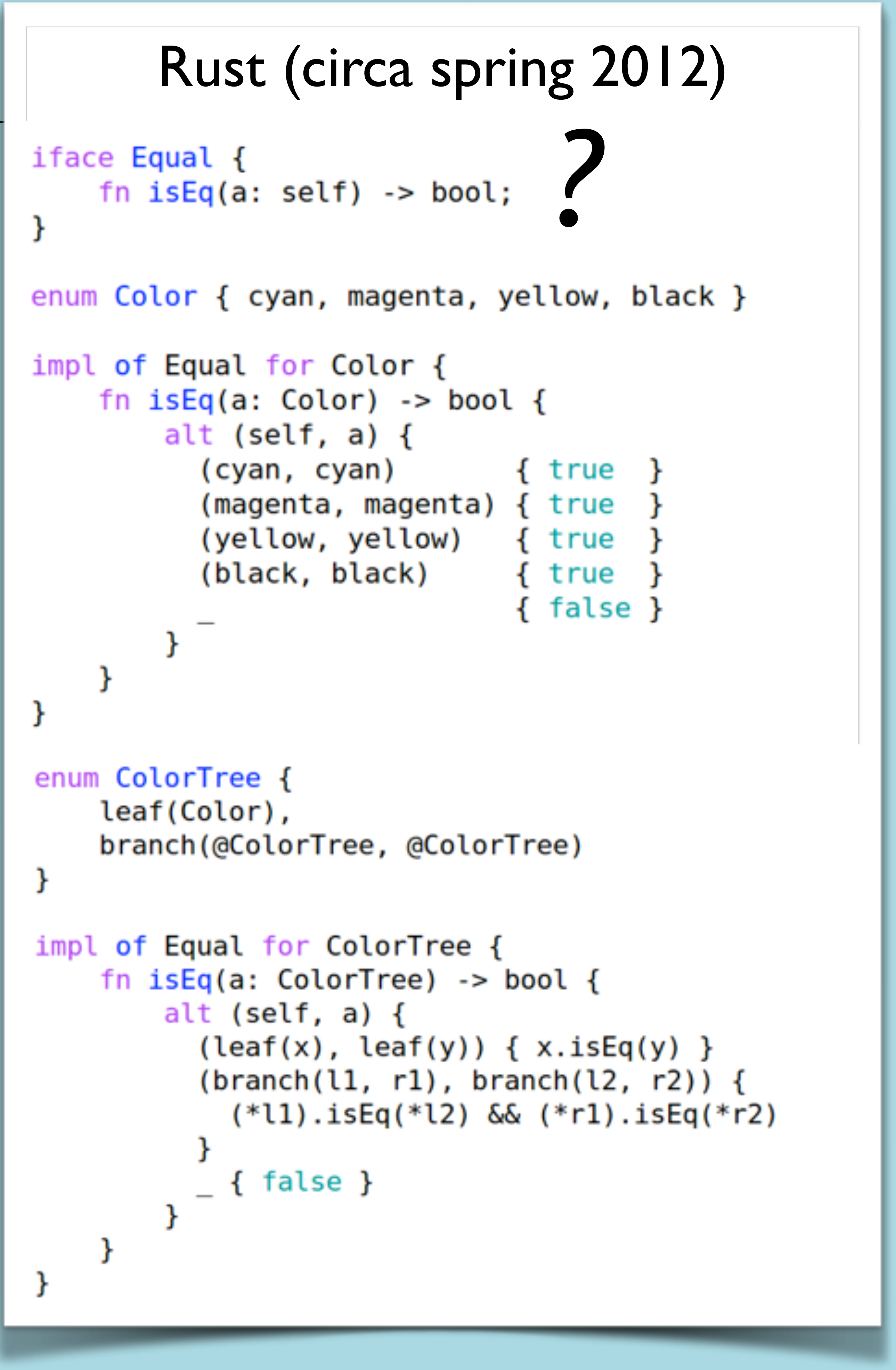
Rust (circa spring 2012)

```rust
iface Equal {
    fn isEq(a: self) -> bool;
}

enum Color { cyan, magenta, yellow, black }

impl of Equal for Color {
    fn isEq(a: Color) -> bool {
        alt (self, a) {
          (cyan, cyan)       { true  }
          (magenta, magenta) { true  }
          (yellow, yellow)   { true  }
          (black, black)     { true  }
          _                  { false }
        }
    }
}

enum ColorTree {
    leaf(Color),
    branch(@ColorTree, @ColorTree)
}

impl of Equal for ColorTree {
    fn isEq(a: ColorTree) -> bool {
        alt (self, a) {
          (leaf(x), leaf(y)) { x.isEq(y) }
          (branch(l1, r1), branch(l2, r2)) {
            (*l1).isEq(*l2) && (*r1).isEq(*r2)
          }
          _ { false }
        }
    }
}
```



Haskell

```haskell
class Equal a where
    isEq :: a -> a -> Bool
    isNotEq :: a -> a -> Bool
    isNotEq x y = not (isEq x y)

data Color = Cyan | Magenta | Yellow | Black

instance Equal Color where
    isEq Cyan    Cyan    = True
    isEq Magenta Magenta = True
    isEq Yellow  Yellow  = True
    isEq Black   Black   = True
    isEq _       _       = False


data ColorTree = Leaf Color
               | Branch ColorTree ColorTree


instance Equal ColorTree where
    isEq (Leaf x) (Leaf y) = isEq x y
    isEq (Branch l1 r1) (Branch l2 r2) =
      (isEq l1 l2) && (isEq r1 r2)
    isEq _ _ = False
```

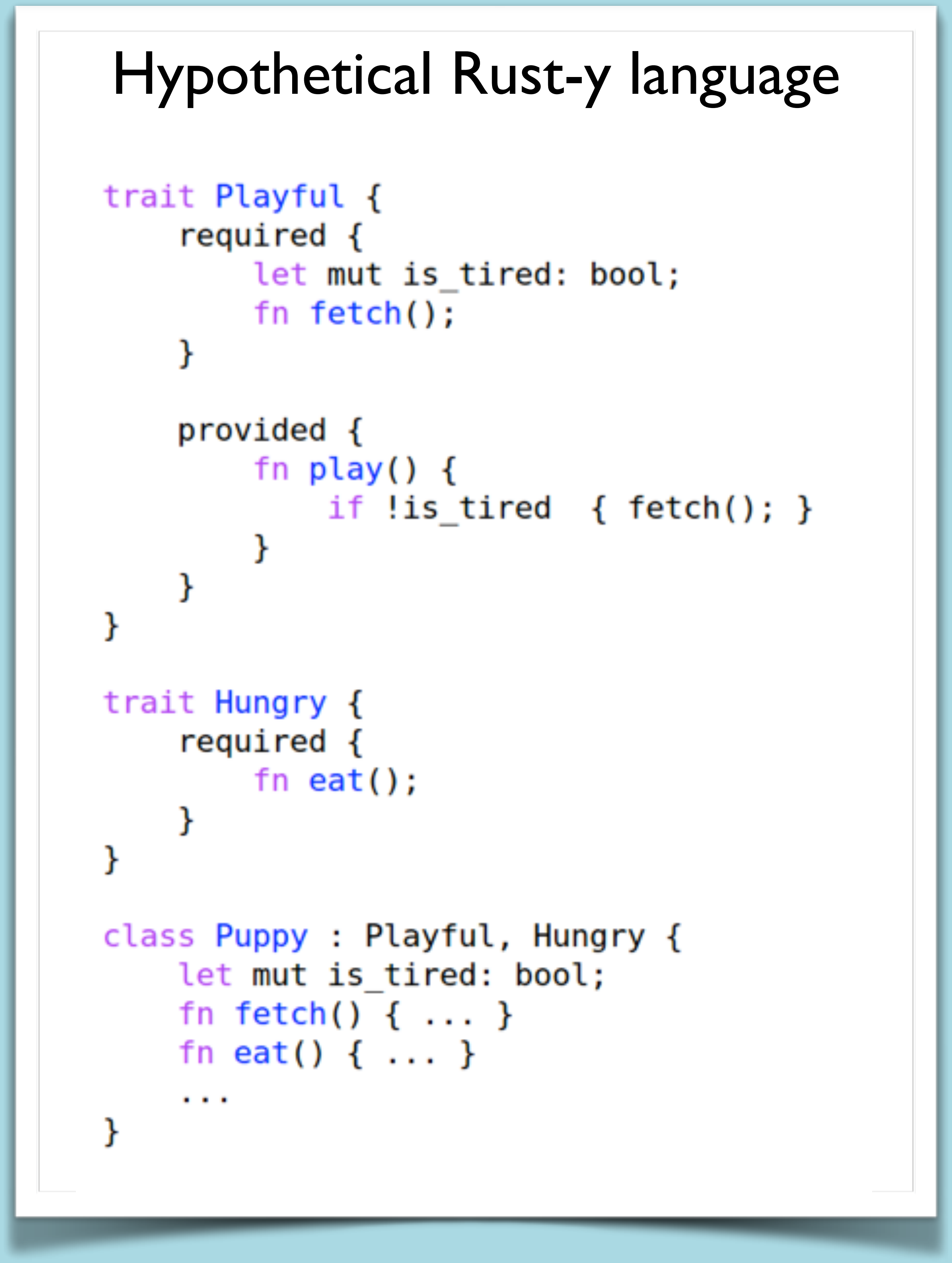


Rust (circa spring 2012)

?

```rust
iface Equal {
    fn isEq(a: self) -> bool;
}

enum Color { cyan, magenta, yellow, black }

impl of Equal for Color {
    fn isEq(a: Color) -> bool {
        alt (self, a) {
          (cyan, cyan)        { true  }
          (magenta, magenta)  { true  }
          (yellow, yellow)    { true  }
          (black, black)      { true  }
          _                   { false }
        }
    }
}

enum ColorTree {
    leaf(Color),
    branch(@ColorTree, @ColorTree)
}

impl of Equal for ColorTree {
    fn isEq(a: ColorTree) -> bool {
        alt (self, a) {
          (leaf(x), leaf(y)) { x.isEq(y) }
          (branch(l1, r1), branch(l2, r2)) {
            (*l1).isEq(*l2) && (*r1).isEq(*r2)
          }
          _ { false }
        }
    }
}
```

What are traits?



Hypothetical Rust-y language

```
trait Playful {
    required {
        let mut is_tired: bool;
        fn fetch();
    }

    provided {
        fn play() {
            if !is_tired  { fetch(); }
        }
    }
}

trait Hungry {
    required {
        fn eat();
    }
}

class Puppy : Playful, Hungry {
    let mut is_tired: bool;
    fn fetch() { ... }
    fn eat() { ... }
    ...
}
```

Unifying traits and typeclasses in Rust

- Hey, wait a second:
  - `provided` methods are analogous to *default methods* in typeclasses!
  - `required` methods are analogous to the method signatures that we have in `iface`s already!
- Add default methods, rename `iface` to `trait`
  - One fewer concept for our users to have to learn
  - Also: trait composition, implementation coherence
  - ETA: next week sometime

Where to find out more

- Typeclasses
  - The Haskell tutorial, section on “classes” (haskell.org)
  - *Real World Haskell*, chapter 6 (book.realworldhaskell.org)
- Traits
  - Schärli *et al*., 2003: “Traits: Composable units of behaviour”
- What’s to come in Rust
  - Dev roadmap (on the Rust wiki)
  - “Proposal for unifying traits and interfaces” (on the Rust wiki)
  - “A Gentle Introduction to Traits in Rust” (pcwalton’s blog)

Go try it out!

rust-lang.org

Thanks!

Photo by mag3737 on Flickr. Thanks!