a systems language
pursuing the trifecta
safe, concurrent, fast

mozilla

a systems language
pursuing the trifecta
safe, concurrent, fast

mozilla

a systems language
pursuing the trifecta
safe, concurrent, fast

Dropping the Drop Flag

Pop Quiz

playpen (http://is.gd/qC1f0X)

```
#[deriving(Show)] struct S1 { x: i32 }
#[deriving(Show)] struct S2 { y: i32 }

/*
 *
 * (various impls elided)
 *
 */

fn main() {
  use std::mem::size_of;
  println!("S1 {:u} bytes, S2 {:u} bytes",
           size_of::<S1>(), size_of::<S2>());
  assert_eq!(size_of::<S1>(), size_of::<S2>());
}
```

The Joke

playpen (http://is.gd/WmLy9g)

```
#[deriving(Show)] struct S1 { x: i32 }
#[deriving(Show)] struct S2 { y: i32 }

impl Drop for S2 {
  fn drop(&mut self) {
    println!("Hi for {}", self);
  }
}

fn main() {
  use std::mem::size_of;
  println!("S1 {:u} bytes, S2 {:u} bytes",
           size_of::<S1>(), size_of::<S2>());
  assert_eq!(size_of::<S1>(), size_of::<S2>());
}
```

Why

playpen (http://is.gd/ZfqgpI)

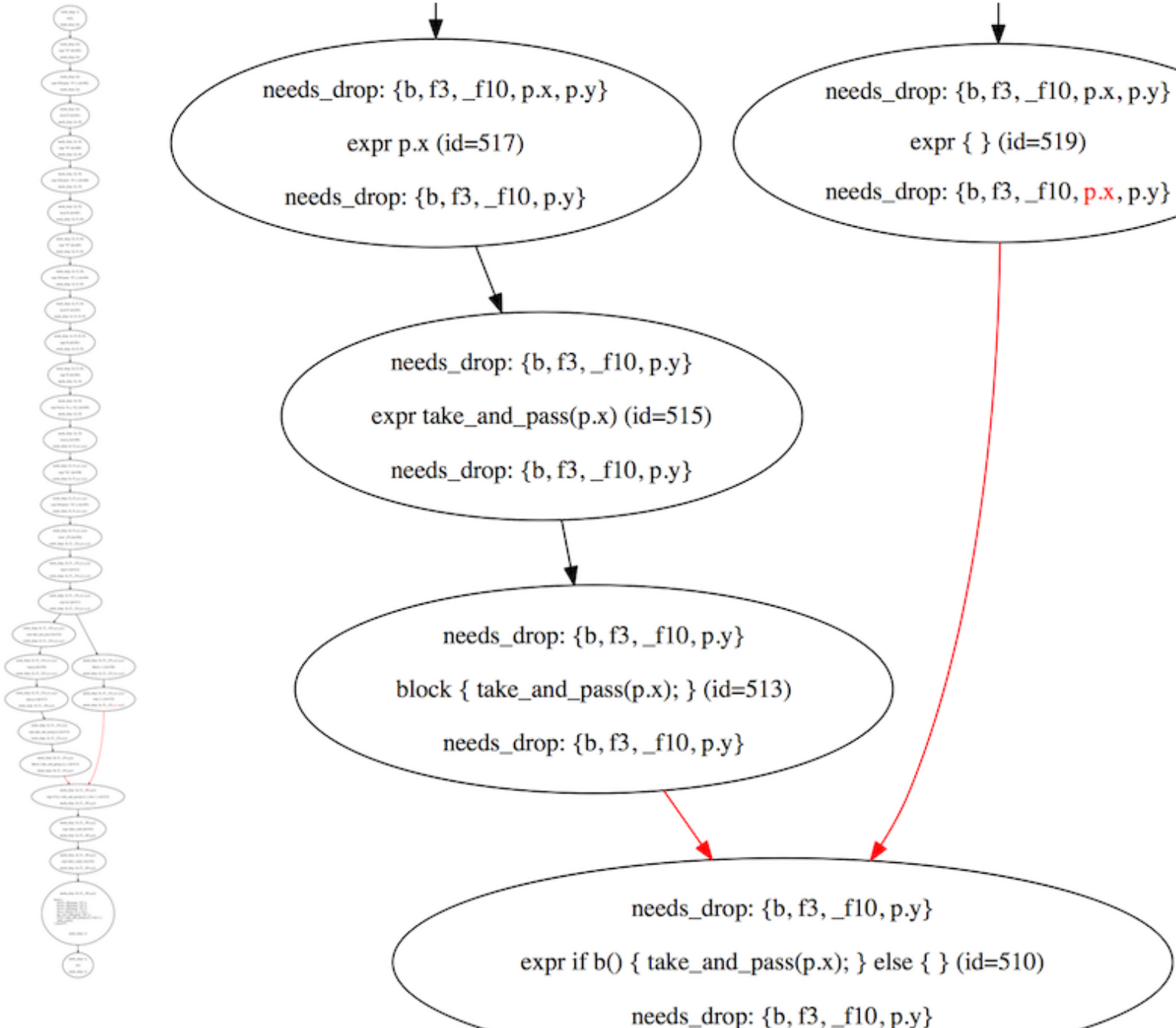
```
fn foo(b: || -> bool) {
    let f3 = Df  { name: "f3" };
    let f4 = Df  { name: "f4" };
    let f5 = Df  { name: "f5" };
    let p = Pair { x: f4, y: f5 };
    let _f10 = Df { name: "f10" };

    if b() {
        // `p.x` consumed by `take_and_pass`
        take_and_pass(p.x);
    }
    // drops here (f10,[ f4,] f5, f3)
}
```

The Fix: Static Drop Semantics

```
fn foo(b: || -> bool) {// DROP OBLIGATIONS
  let f3 = Df {...};    // { f3 }
  let f4 = Df {...};    // { f3,f4 }
  let f5 = Df {...};    // { f3,f4,f5 }
  let p = Pair{
     x: f4, y: f5 };    // { f3,      , p }
  let _f10 = Df {...}; // { f3,      , p,   _f10 }
  if b() {
                         // { f3,      , p,   _f10 }
    take_and_pass(p.x);
                         // { f3,      , p.y, _f10 }
  } else {
                         // { f3,      , p,   _f10 }
  }
                    // drop mismatch (p versus p.y)
}
```

```
rustc ex.rs --pretty flowgraph=foo \
            -Z flowgraph-print-needs-drop
```



Common Case: Auto-Inserted Drop

```
fn foo(b: || -> bool) {  // DROP OBLIGATIONS
  let f3 = Df { ... };   // { f3 }
  let f4 = Df { ... };   // { f3,f4 }
  let f5 = Df { ... };   // { f3,f4,f5 }
  let p = Pair{ x: f4,
                y: f5 }; // { f3,      , p }
  let _f10 = Df { ... }; // { f3,      , p,   _f10 }
  if b() {
                         // { f3,      , p,   _f10 }
    take_and_pass(p.x);
                         // { f3,      , p.y, _f10 }
  } else {
                         // { f3,      , p,   _f10 }
    rustc_inserted_drop(p.x);
                         // { f3,      , p.y, _f10 }
  }
}
```

Gotcha

- Drops with side-effects?

Gotcha

- Drops with side-effects?
- RAII patterns

Gotcha

- Drops with side-effects?
- RAII patterns
  - locks
  - flushing buffers
  - etc (?)

Lint to the Rescue

```
  let p = Pair{ x: f4,
                y: f5 }; // { f3,      , p }
  let _f10 = Df { ... }; // { f3,      , p,   _f10 }
  if b() {
                         // { f3,      , p,   _f10 }
    take_and_pass(p.x);
                         // { f3,      , p.y, _f10 }
  } else {
                         // { f3,      , p,   _f10 }




  }
  other_code();
}
```

Lint to the Rescue

```
  let p = Pair{ x: f4,
                y: f5 }; // { f3,      , p }
  let _f10 = Df { ... }; // { f3,      , p,   _f10 }
  if b() {
                         // { f3,      , p,   _f10 }
    take_and_pass(p.x);
                         // { f3,      , p.y, _f10 }
  } else {
                         // { f3,      , p,   _f10 }
    warning: `p.x` initialized here, but
    not on other paths. (Use Option, call
    `drop()`, or reinitialize `p.x` elsewhere.)
    #[warn(unmarked_early_drop)] on by default
  }
  other_code();
}
```

Tiers of linting

```
#[lang="noisy_drop"] trait NoisyDrop { }
#[lang="quiet_drop"] trait QuietDrop { }

                     // Drops default to quiet...
impl Drop for Df { ... }
                     // ...until marked noisy.
impl NoisyDrop for Df {}

                     // Noisiness bubbles out...
struct S { d: Df }

                     // ... until marked quiet again.
struct Q { s: S }
impl QuietDrop for Q {}

#[warn(early_loud_drop)]
#[allow(early_quiet_drop)]
```

Thanks for listening