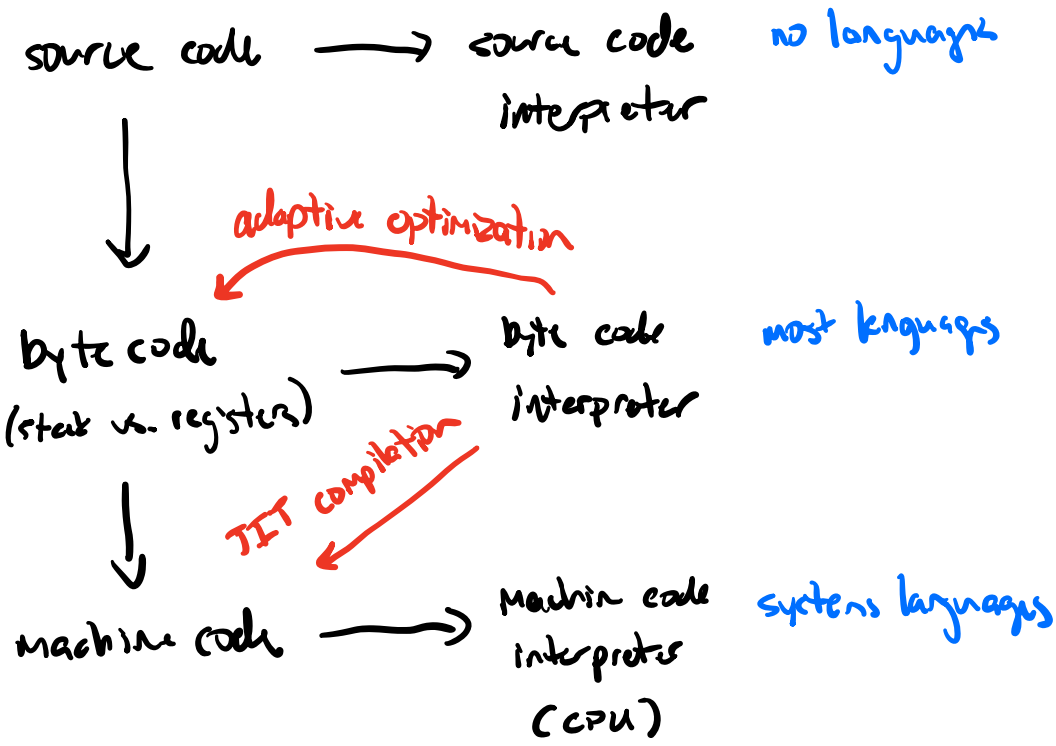
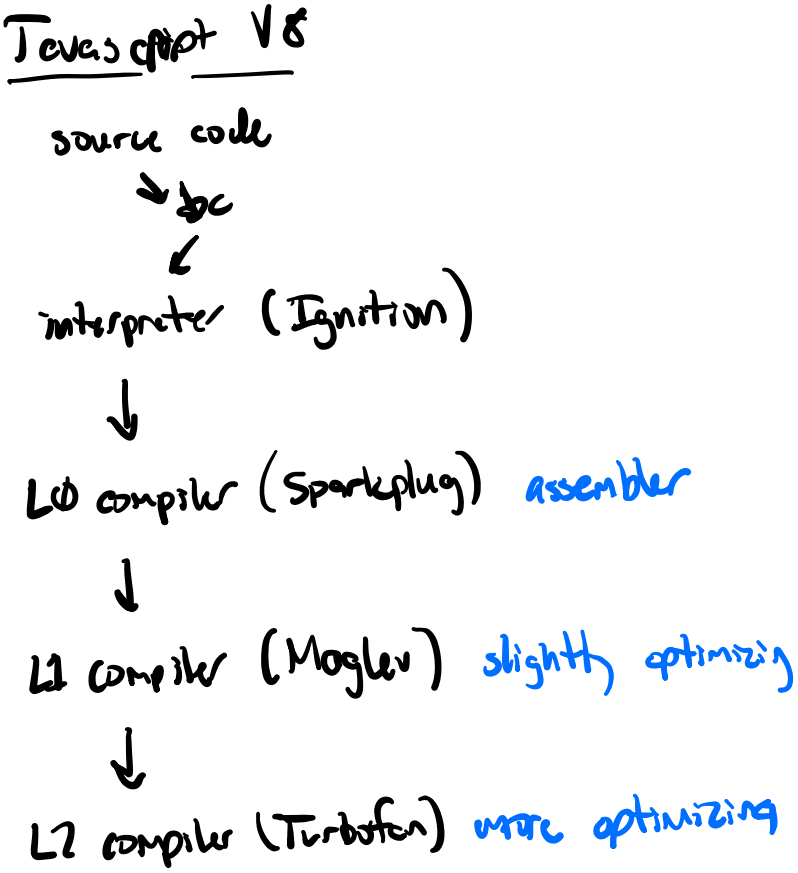
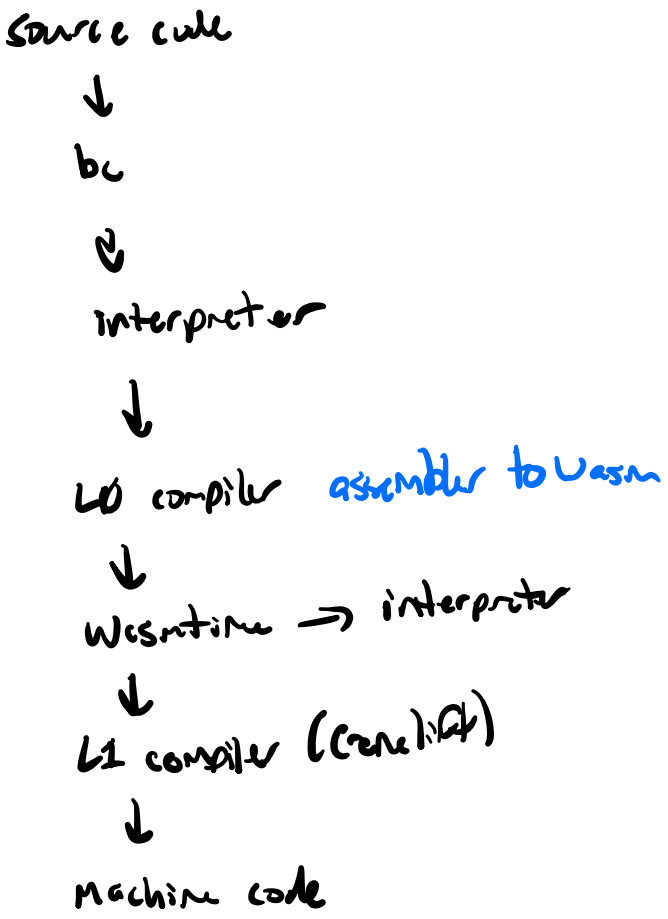

source code
source code interpreter
no languages
adaptive optimization
byte code
(stack vs. register)
byte code interpreter
most languages
JIT compilation
machine code
machine code interpreter (CPU)
systems languages

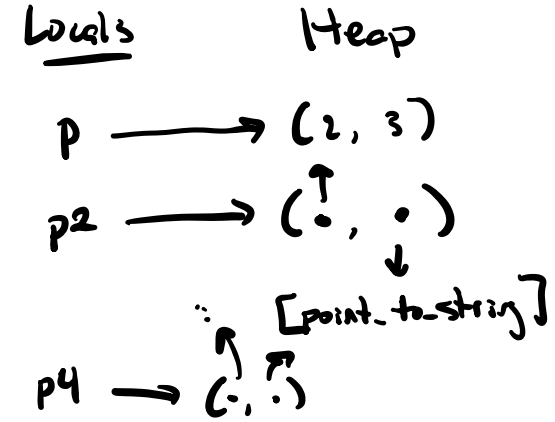

Javascript V8
source code
bc
interpreter (Ignition)
L0 compiler (Sparkplug)
assembler
L1 compiler (Maglev)
slightly optimizing
L2 compiler (Turbofan)
more optimizing


Rice
source code
bc
interpreter
L0 compiler
assembler to Wasm
Wasmtime → interpreter
L1 compiler (Cranelift)
machine code

```
struct Point(int, int);

interface ToString {
  fn to_string(self) -> string;
}

impl ToString for Point {
  fn to_string(self: Self) -> string {
    ...
  }
}
```




```
let p = new Point(2,3) in
let p2 = p as @ToString in
p2.to_string()
~> p2.1[0](p2.0)
```

```
let p3 =
  new Point(4,5)
let p4 = p3 as @ToString
```

```
interface Vec {
  fn get(self, int) -> float;
  fn len(self) -> int
}

fn mag(v: @Vec) -> float {
  let n = 0. in
  for i < v.len() {
    n := n + v.get(i) * v.get(i)
  };
  sqrt(n)
}
```

```
struct Vec2(float, float);

impl Vec for Vec2 { ... }
```

```
n = 0
i < v.len()
  ↳ v.1[i](v.0)
n := n + v.get(i)
  ↳ v.1[0](v.0, i)
```

```
let p = new Vec2(0, 1)
mag(p) ~> mag_vec2(p)
```

① type profiling

② type specialization

? ? ?

③ side exit

```
if p is Vec2 {
  mag_vec2(p)
} else {
  mag(p)
}
```

```
fn mag(v: @Vec) -> float {
  let n = 0. in
  for i < v.len() {
    n := n + v.get(i) * v.get(i)
  };
  sqrt(n)
}
```

```
fn mag(v: Vec2) -> float {
  let n = 0. in
  for i < vec2_len(v) {
    n := n + vec2_get(v, i) *
             vec2_get(v, i)
  };
  sqrt(n)
}
```

```
fn mag(v: Vec2) -> float {
  let n = 0. in
  for i < 2 {
    n := n + if i == 0 { v.0 }
             else { v.1 }
  };
  sqrt(n)
}
```

```
n := n + if 0 == 0 { v.0 }
         else { v.1 }
n := n + if 1 == 0 { v.0 }
         else { v.1 }
```

```
n := n + if true { v.0 }
...
n := n + if false { ... }
```

```
let n = 0. in
n := n + v.0
n := n + v.1
sqrt(n)
```