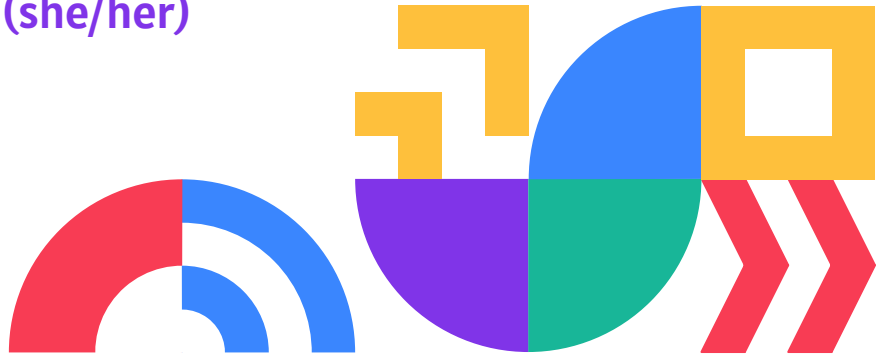
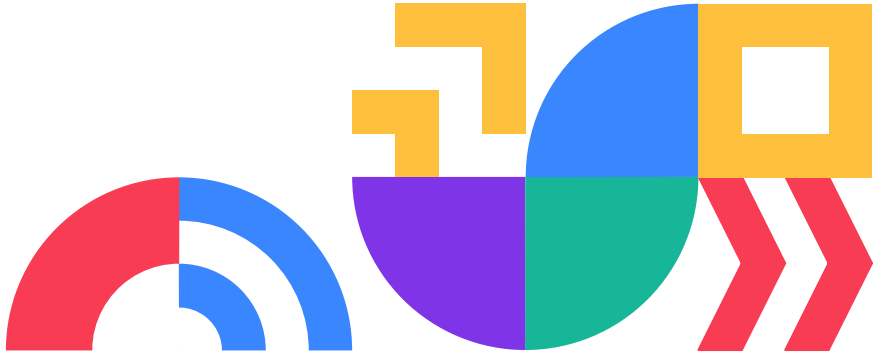
# CLAW AND COMPONENT-NATIVE LANGUAGES / TOOLCHAINS

Robin Brown (she/her)



# Outline

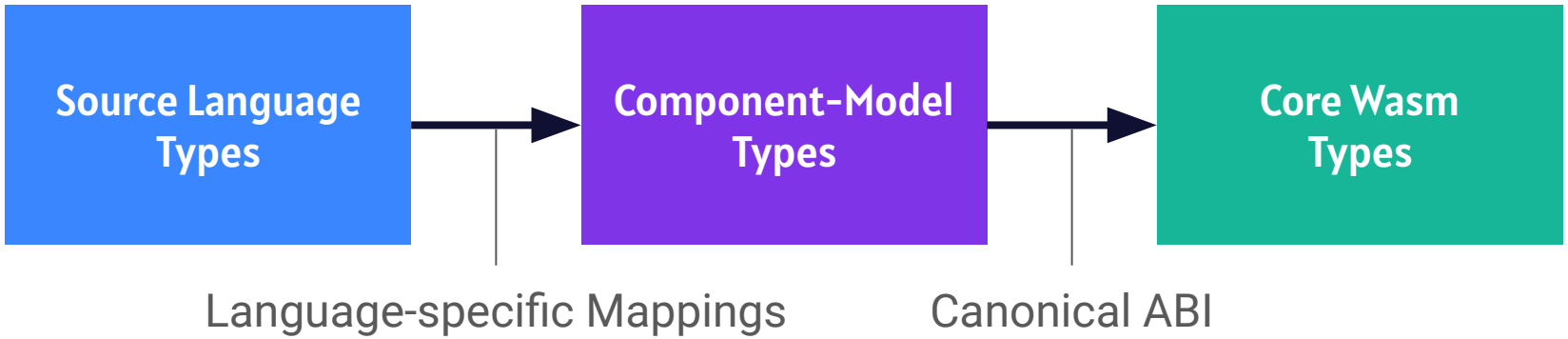
- Component-Native
  - What does that mean?
  - Benefits
- Claw
  - Overview
  - Demos
- Future work
  - Other tools
  - Other languages?



# Component-Native

Leaning into, and specializing for the features of Wasm and the Component-Model

# Map Types Directly to Wasm

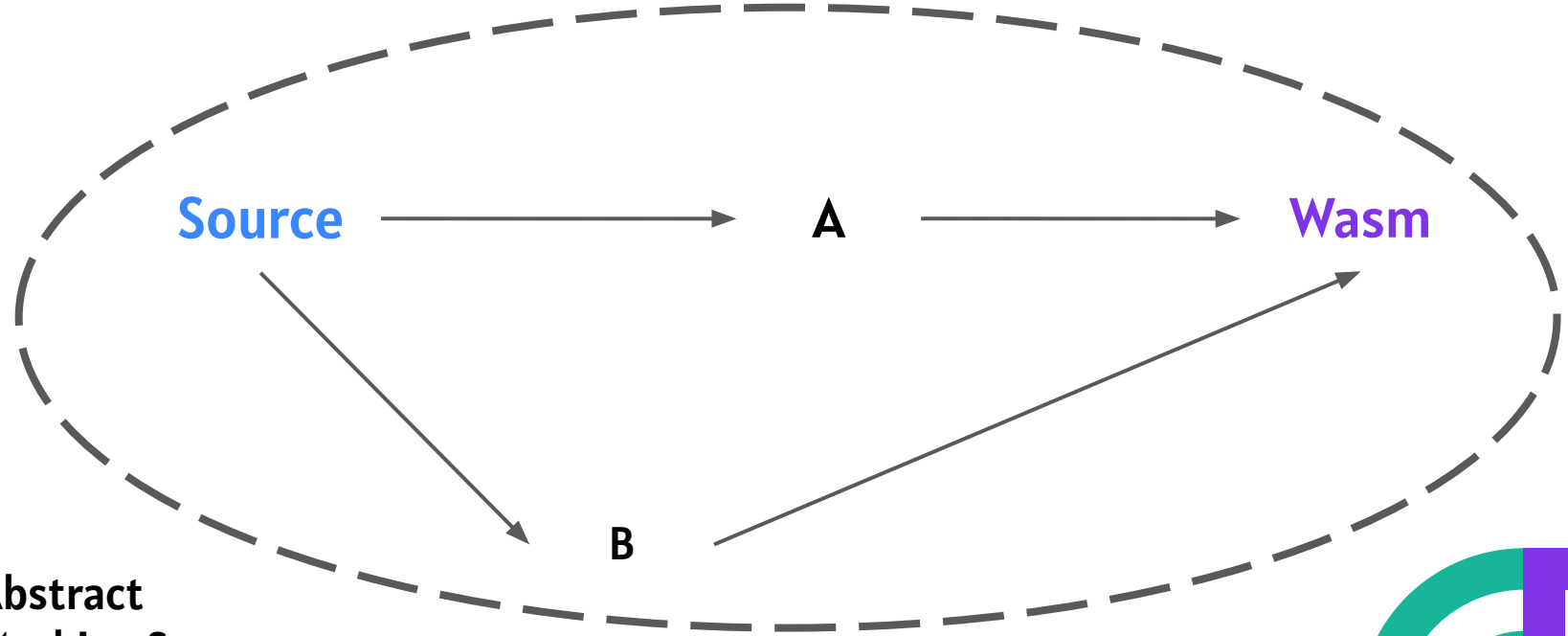



*Where we're going, we don't need bindgen*

# Intermediate Representations can still be “Direct”

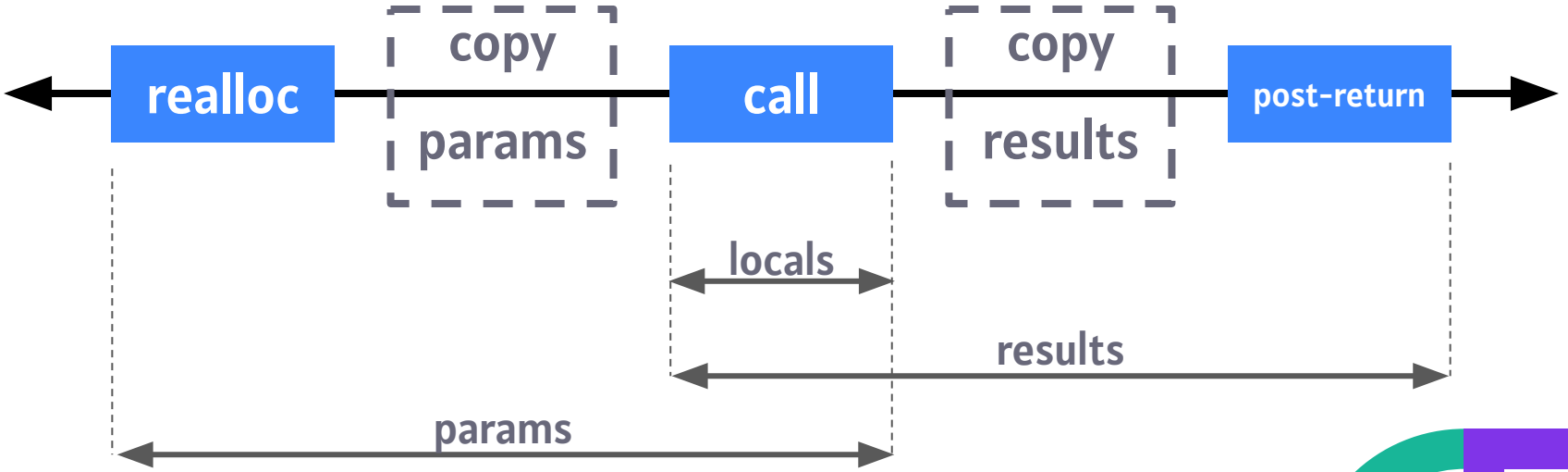
- Using **Intermediate Representations** (IRs) doesn’t make something less **Component-Native**
  - e.g. Rust’s High-level IR (HIR) and Mid-level IR (MIR)
- but “**Detouring**” through another Type System does
  - e.g. Representing them as C Header Types

# Direct vs. Detour



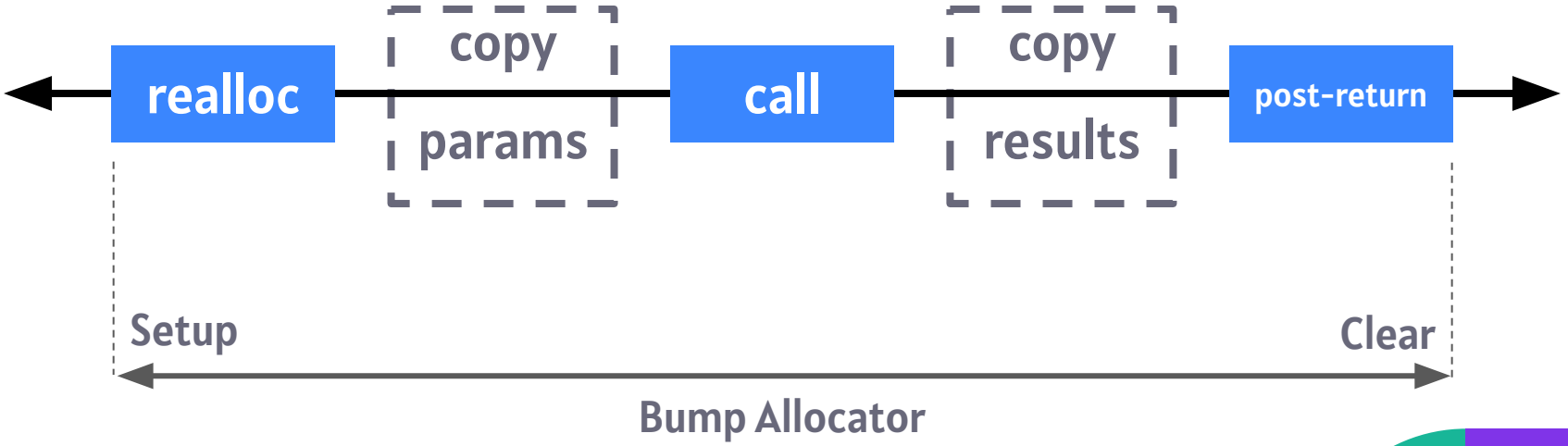
# Take Advantage of Component Data Lifetimes

# Take Advantage of Component Data Lifetimes



# Take Advantage of Component Data Lifetimes

# Take Advantage of Component Data Lifetimes



# Focus on Wasm Codegen and Optimization

- Wasm-aware code generation and optimization
- Don’t duplicate runtime or Wasm optimizations
- Focus on enabling runtime and Wasm optimizations

# Component-Native Benefits

# New Toolchains

- Development benefits
  - Simplicity
  - Maintainability
  - Extensibility
- Strong baseline performance
  - Compile speed
  - Binary size
  - Binary speed

## Module-Native Toolchains

(e.g. Grain, Guile)

Join the ~~dark side~~ [component ecosystem],

we have ~~cookies~~ [guest interop, WASI, registries]

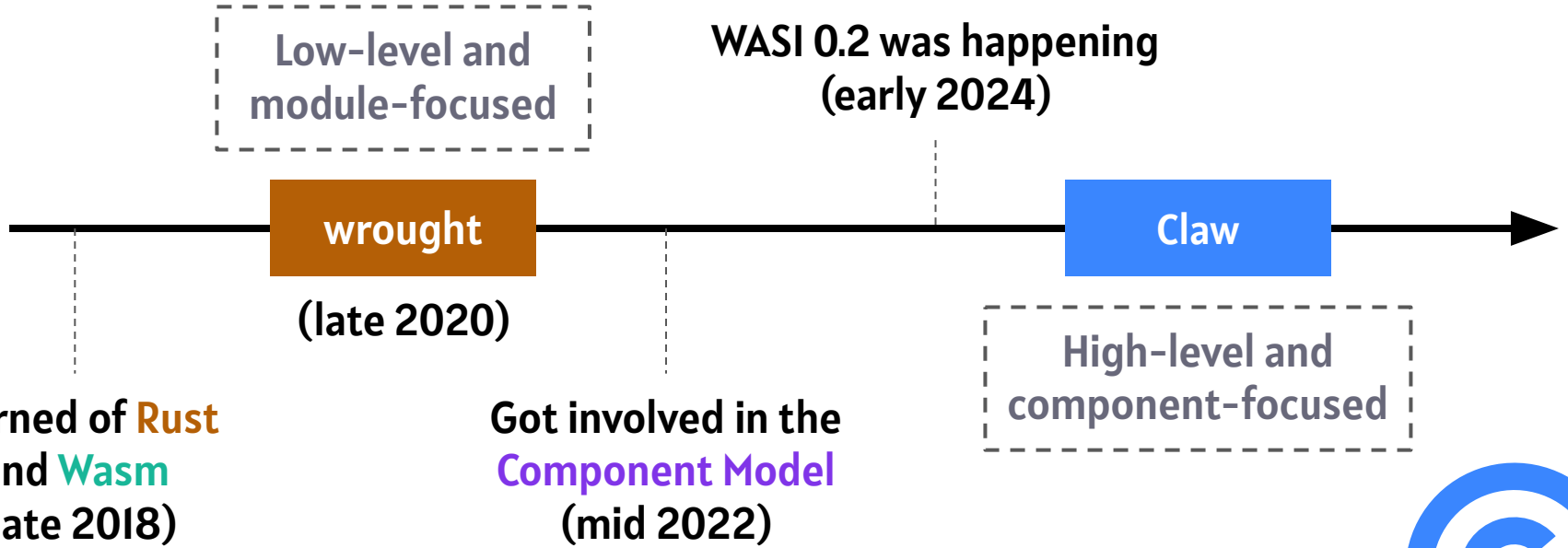
# Established Toolchains

- May have UX and complexity benefits over bindgen
- May improve binary size or startup speed

**Recommendation:** Keep these techniques in mind even if it isn't worth it yet

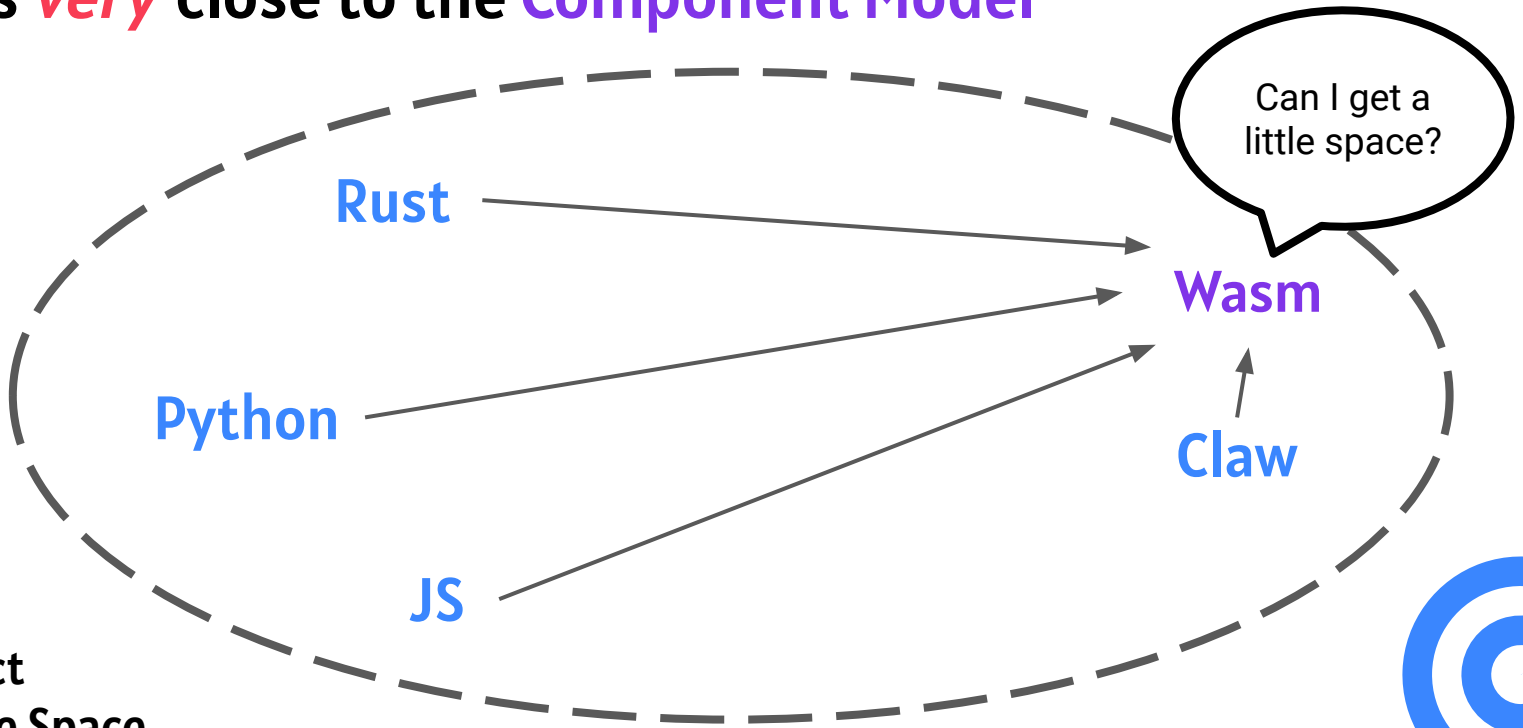
# Claw

# History


Low-level and module-focused
WASI 0.2 was happening (early 2024)
wrought
Claw
(late 2020)
High-level and component-focused
Learned of Rust and Wasm (late 2018)
Got involved in the Component Model (mid 2022)

# Claw leans *all* the way in to the Component Model

# So it's *very* close to the Component Model



# It matches other Component Model tools as much as possible

# Simple WIT

```
import trim: func(s: string) -> string;

export name: func(p: person) -> string;
```

# Simple Claw

```
import trim: func(s: string) -> string;

export func name(p: person) -> string {
    return trim(p.first) + " " + trim(p.last);
}
```

# Simple Claw

```
import trim: func(s: string) -> string;

export func name(p: person) -> string {
    return trim(p.first) + " " + trim(p.last);
}
```

# Using External WIT Types

```
import { level, log } from wasi:logging/logging;

export func run() {
    log(level::debug, "demo", "run() started");
    ...
    log(level::debug, "demo", "run() finished");
}
```

# Use Case: Component Testing

```
import add: func(lhs: s32, rhs: s32) -> s32;

export interface tests {
  func test() -> result<(), string> {
    check!(add(1, 1) == 2)?;
    ...
    return ok(());
  }

  ...
}
```

# Claw Demo!!!

# Future Work

# Claw's Future

- Finish implementing Component-Model types
  - e.g. results
- Add missing language features
  - e.g. break, continue
- Implement –no-alloc mode (Claw for Embedded?)
  - Error if a feature is used that requires an allocator
  - Don't code generate an allocator in the resulting component

# Come contribute!!

https://github.com/esoterra/claw-lang

# Acknowledgements


- Dan Gohman
- Daniel Macovei
- Timmy Silesmo

# #opentowork

github.com/**esoterra**

linkedin.com/**esoterra**

@**esoterra**@hachyderm.io

**esoterra**.dev

# Q & A