What if Kubernetes was a compiler target?

David Morrison
Research Scientist
Applied Computing Research Labs

Tim Goodwin
PhD Student
UC Santa Cruz, CSE Dept.

14 November, 2024

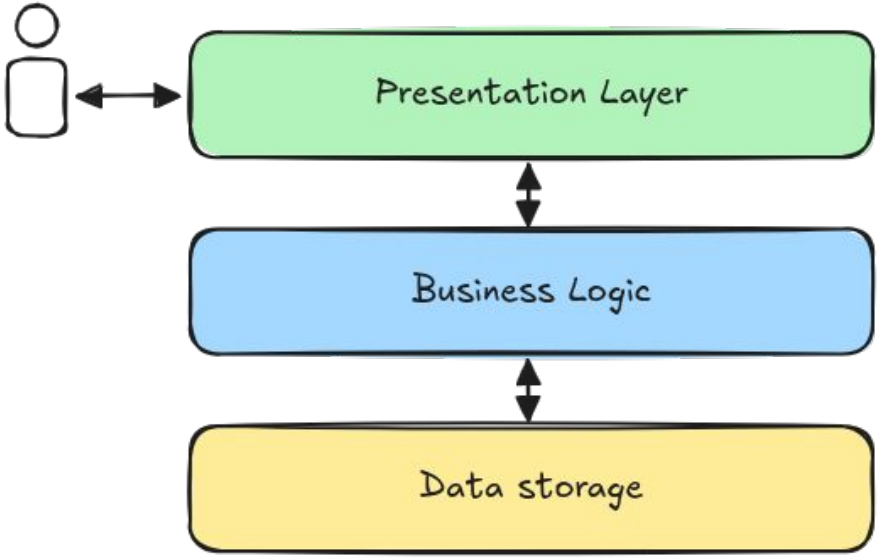
Who are we?

- David Morrison
- Founder/Research Scientist at ACRL (previously Yelp and Airbnb)
- Contributor to Cluster Autoscaler, Karpenter, SimKube
- drmorr on Slack/Hachyderm/Bluesky, drmorr0 on GH

- Tim Goodwin
- PhD student at UC Santa Cruz, previously @ Oscar Health
- Researching control plane debuggability
- tgoodwin on discuss.systems + GH

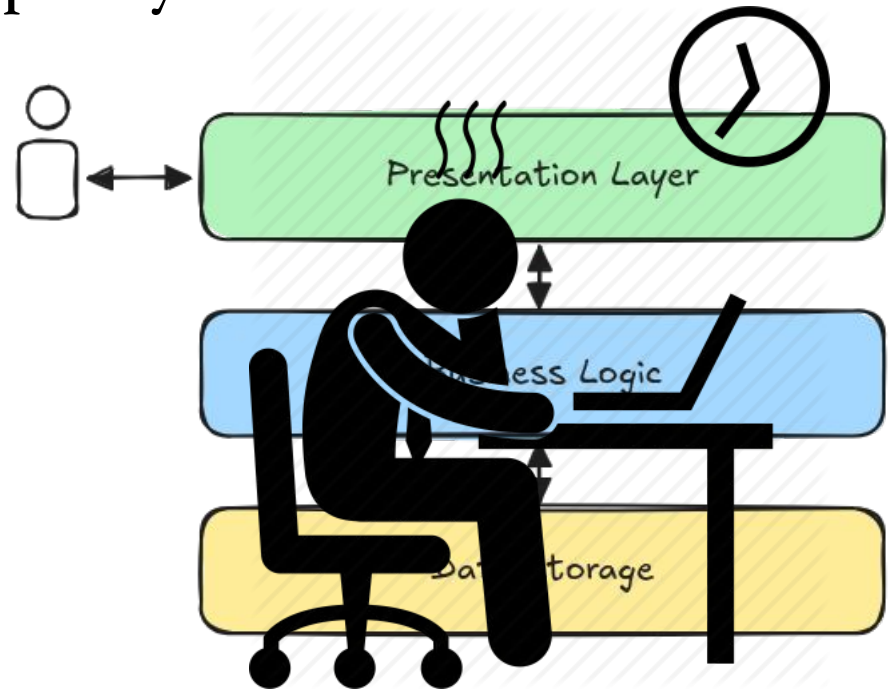
Tackling Distributed System Complexity

- Programming challenges in distributed systems
  - Spreading business logic across components
  - Handling communication, serialization, format conversion
  - Handling multiple languages and environments



Tackling Distributed System Complexity

- Programming challenges in distributed systems
  - Spreading business logic across components
  - Handling communication, serialization, format conversion
  - Handling multiple languages and environments



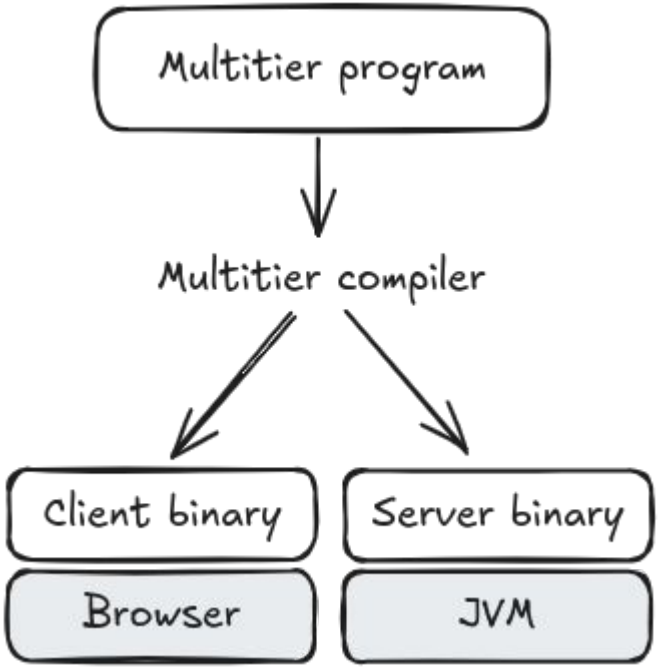
How can we abstract away some of this programming complexity?

How can we abstract away some of this programming complexity?

Write a distributed application as a single program!

Multitier Programming in a Nutshell

- Originates from PL community
- Singular program compiles into distributed system
- Logical “tiers” instead of individual components
- Compiler generates deployable units
  - Leverage user annotations + static analysis
  - Can target different environments



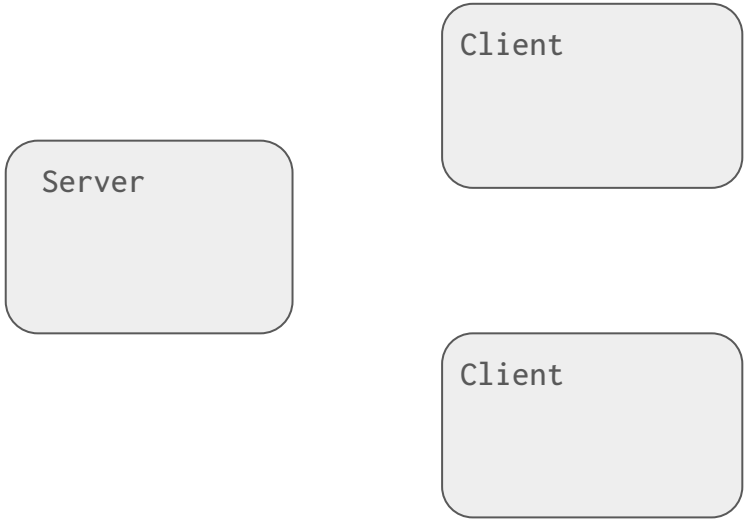
ScalaLoci Chat App - declaring a *Server*

```
@multitier object Chat {
  @peer type Server <: { type Tie <: Multiple[Client] }

}
```

Example adapted from scala-loci.github.io. Recreated by Tim Goodwin.

ScalaLoci Chat App - declaring a *Client*

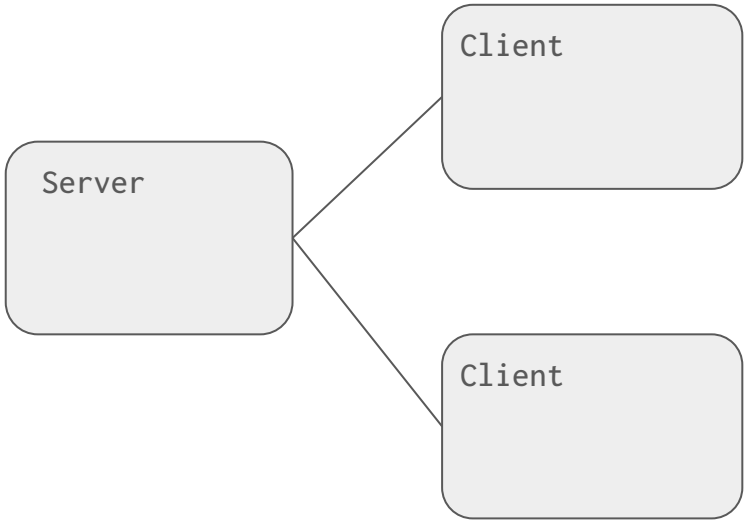
```
@multitier object Chat {
  @peer type Server <: { type Tie <: Multiple[Client] }
  @peer type Client <: { type Tie <: Single[Server] }

}
```



Example adapted from scala-loci.github.io. Recreated by Tim Goodwin.

ScalaLoci Chat App - declaring a *Client*

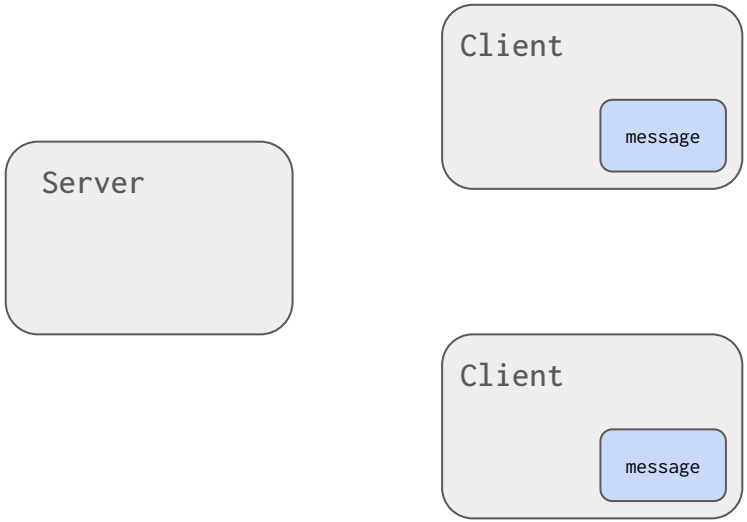
```
@multitier object Chat {
  @peer type Server <: { type Tie <: Multiple[Client] }
  @peer type Client <: { type Tie <: Single[Server] }

}
```



Example adapted from scala-loci.github.io. Recreated by Tim Goodwin.

ScalaLoci Chat App - submitting messages from the client

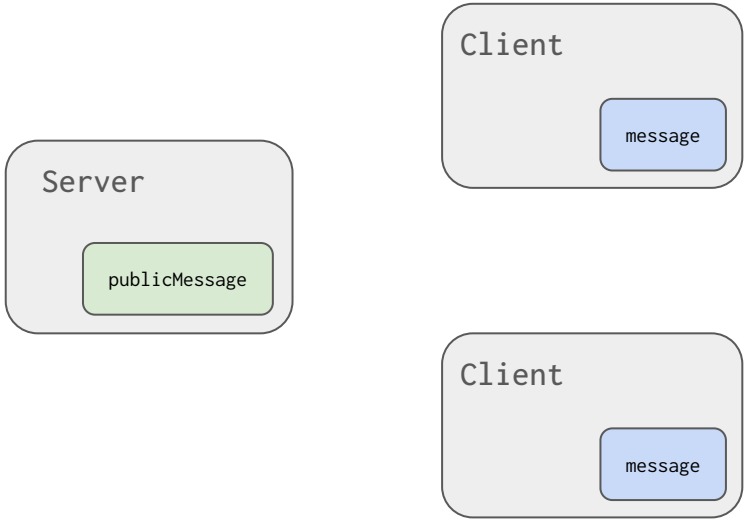
```
@multitier object Chat {
  @peer type Server <: { type Tie <: Multiple[Client] }
  @peer type Client <: { type Tie <: Single[Server] }
  val message = on[Client] { Evt[String]() }



  def main() = on[Client] {

    for (line <- io.Source.stdin.getLines)
      message.fire(line)
  }
}
```



Example adapted from scala-loci.github.io. Recreated by Tim Goodwin.

ScalaLoci Chat App - collecting messages at the server

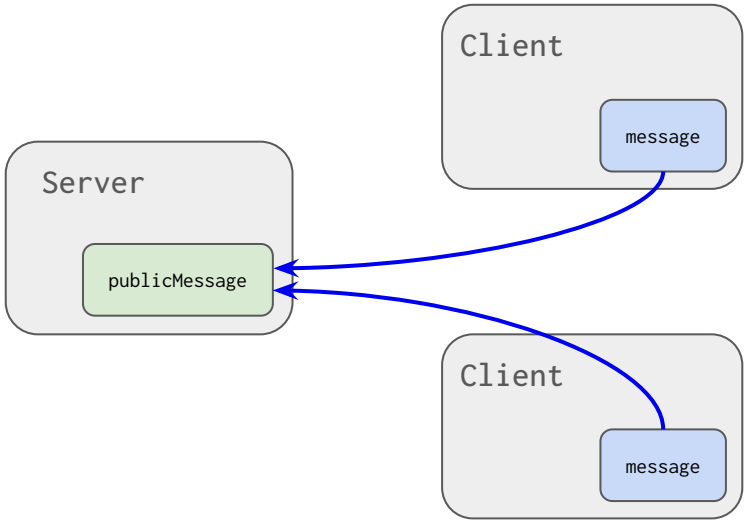
```
@multitier object Chat {
  @peer type Server <: { type Tie <: Multiple[Client] }
  @peer type Client <: { type Tie <: Single[Server] }
  val message = on[Client] { Evt[String]() }
  val publicMessage = on[Server] {
    message.asLocalFromAllSeq map { case (_, msg) => msg) }
  }
  def main() = on[Client] {

    for (line <- io.Source.stdin.getLines)
      message.fire(line)
  }
}
```



Example adapted from scala-loci.github.io. Recreated by Tim Goodwin.

ScalaLoci Chat App - collecting messages at the server

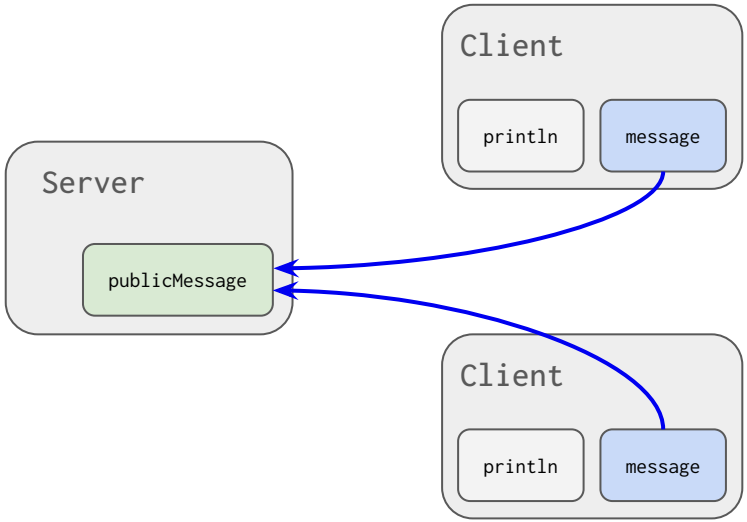
```
@multitier object Chat {
  @peer type Server <: { type Tie <: Multiple[Client] }
  @peer type Client <: { type Tie <: Single[Server] }
  val message = on[Client] { Evt[String]() }
  val publicMessage = on[Server] {
    message.asLocalFromAllSeq map { case (_, msg) => msg) }
  }
  def main() = on[Client] {

    for (line <- io.Source.stdin.getLines)
      message.fire(line)
  }
}
```



Example adapted from scala-loci.github.io. Recreated by Tim Goodwin.

ScalaLoci Chat App - serve public messages to the client

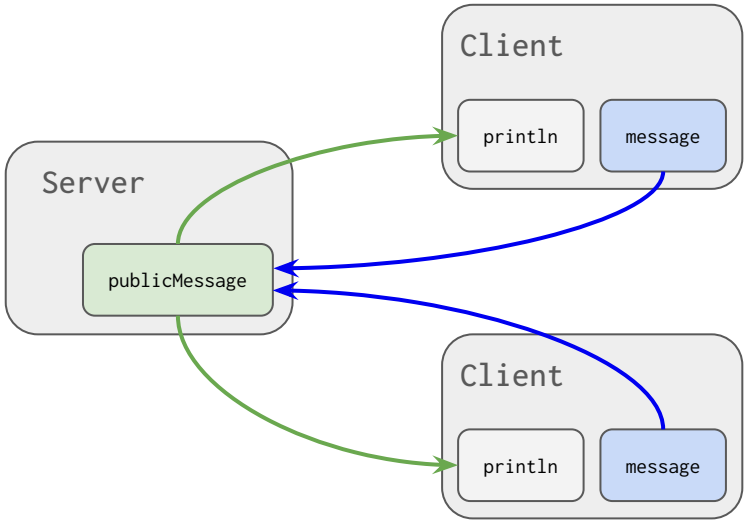
```
@multitier object Chat {
  @peer type Server <: { type Tie <: Multiple[Client] }
  @peer type Client <: { type Tie <: Single[Server] }
  val message = on[Client] { Evt[String]() }
  val publicMessage = on[Server] {
    message.asLocalFromAllSeq map { case (_, msg) => msg) }
  }
  def main() = on[Client] {
    publicMessage.asLocal observe println
    for (line <- io.Source.stdin.getLines)
      message.fire(line)
  }
}
```



Example adapted from scala-loci.github.io. Recreated by Tim Goodwin.

ScalaLoci Chat App - serve public messages to the client

```
@multitier object Chat {
  @peer type Server <: { type Tie <: Multiple[Client] }
  @peer type Client <: { type Tie <: Single[Server] }
  val message = on[Client] { Evt[String]() }
  val publicMessage = on[Server] {
    message.asLocalFromAllSeq map { case (_, msg) => msg) }
  }
  def main() = on[Client] {
    publicMessage.asLocal observe println
    for (line <- io.Source.stdin.getLines)
      message.fire(line)
  }
}
```



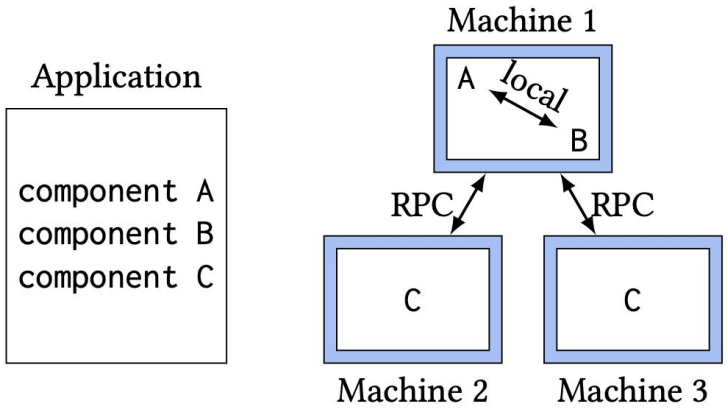
Example adapted from scala-loci.github.io. Recreated by Tim Goodwin.

Multitier Programming Benefits

- Higher abstraction level
  - Avoid repetitive plumbing code, RPC boilerplate
  - Focus on business logic
- Improved software design
  - Better modularity (avoid spreading common features across components)
  - Less repetition
- Code maintenance
  - Less change amplification
  - Easy to refactor logic between tiers

Multi-tier ⇝ “Distributed Component Applications”

- Developers focus on encapsulating business logic into framework components
- Runtime handles how components are deployed
  - Microsoft DCOM (90s)
  - Enterprise Java Beans (90s/2000s)
  - Google’s ServiceWeaver (2023)



From Ghemawat et al. “Towards Modern Development of Cloud Applications”

Where are we today?

- Microservices are ubiquitous
- Kubernetes has become the industry standard for *managing* distributed applications
  - Greatly simplifies scaling, orchestration, fault tolerance
- Hasn't really changed the way we *write* applications
  - Other than encouraging the proliferation of the microservice pattern
- Where do we go from here?
  - How might we leverage Kubernetes to simplify application *development?*
  - Support generic applications written in general purpose languages

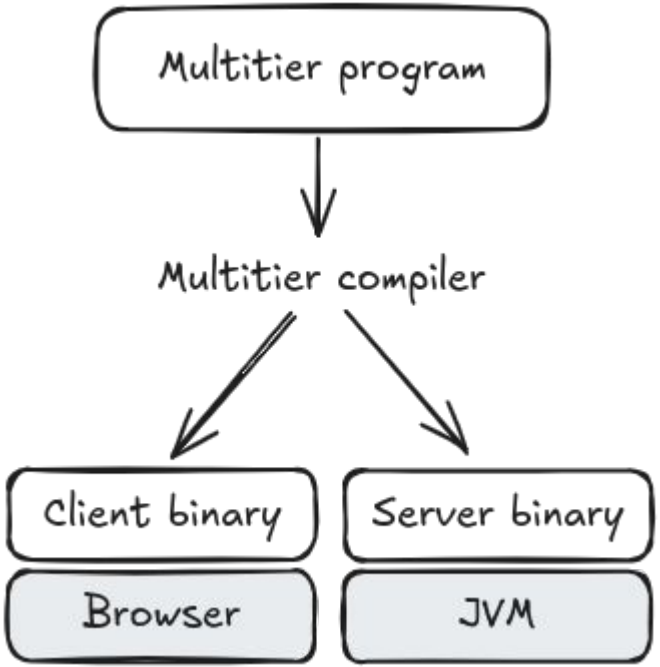
What if we could apply these same concepts to developing Kubernetes apps?

What if we didn’t have to worry about microservices?

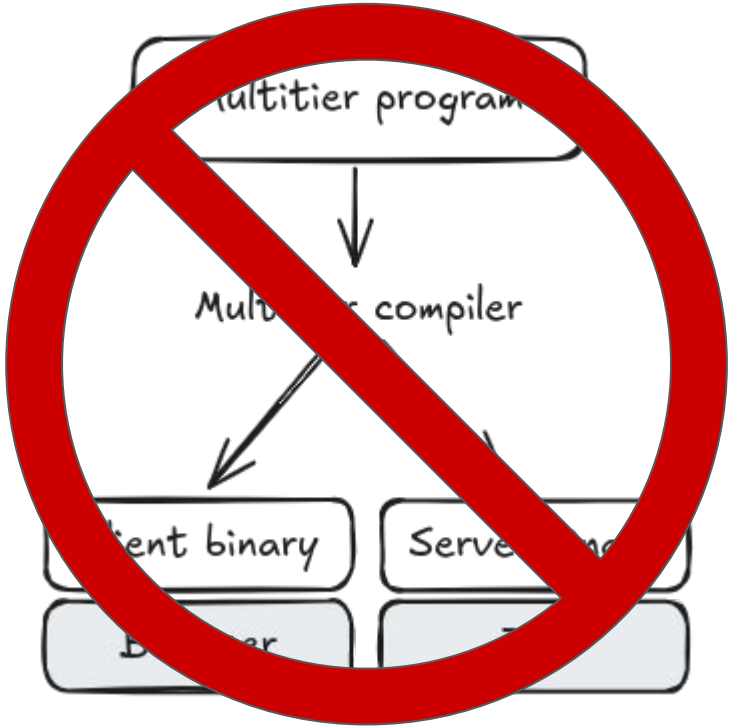
What if we could just go back to the monolith?

What if Kubernetes was a compiler target?

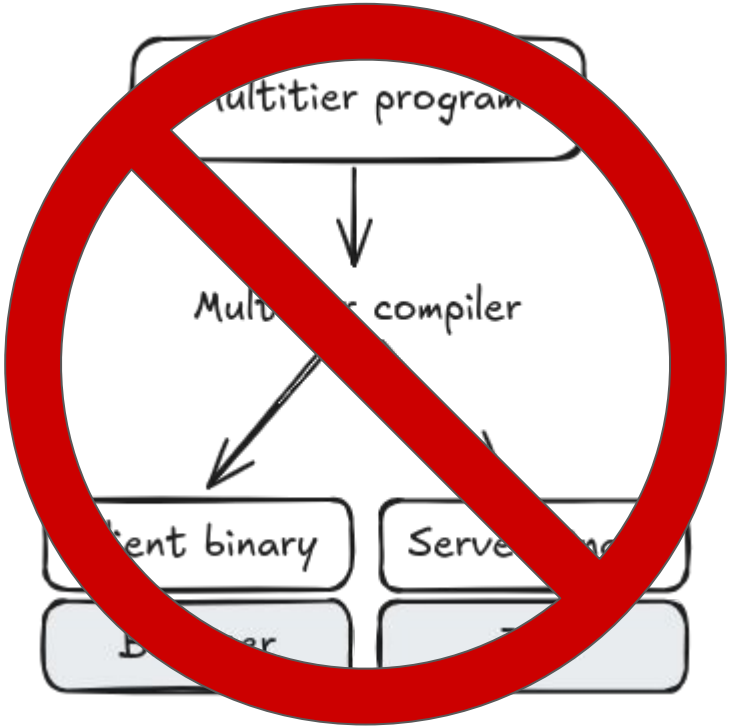
What if Kubernetes was a compiler target?

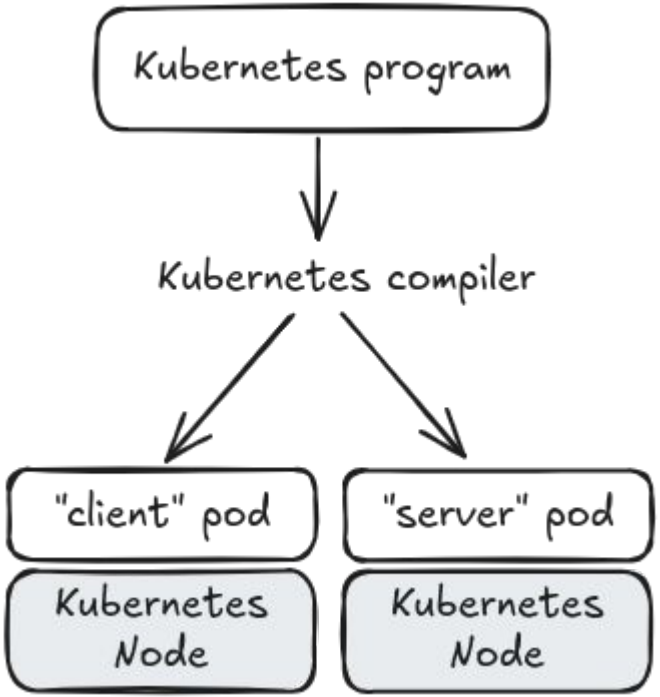


What if Kubernetes was a compiler target?



What if Kubernetes was a compiler target?





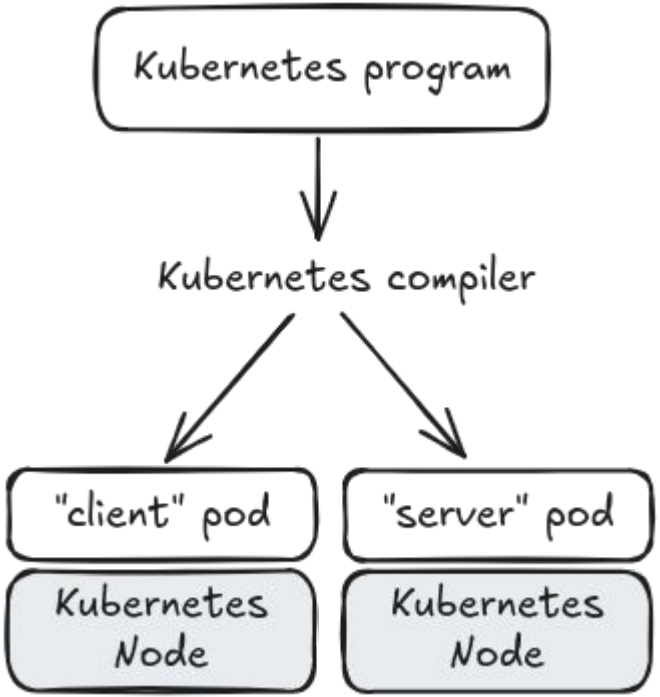
Key Idea #1:
Goroutines ≈ Kubernetes Pods

Key Idea #2:
Channels ≈ Network Requests

Kompile demo

What did we learn?

- Multi-tier programming: writing one program for multiple deployment “tiers”
- Often require specialized programming languages (or code annotations)



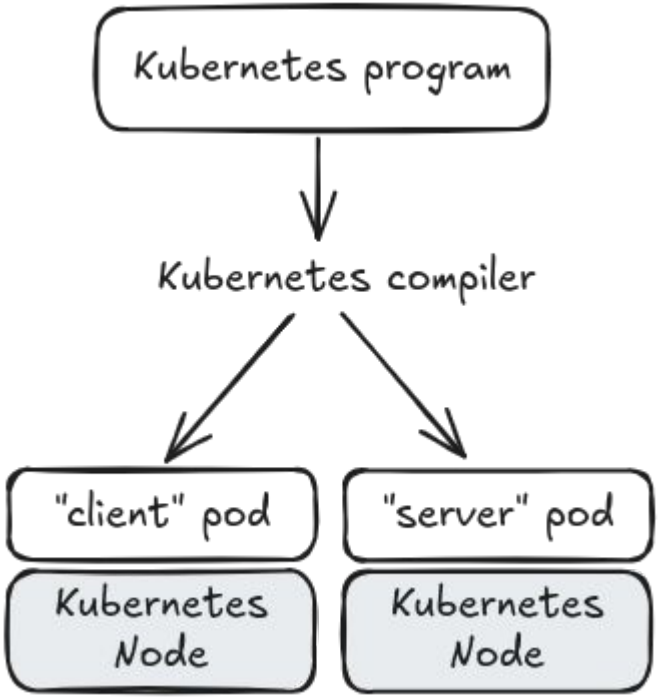
Service Weaver 

🚨 Important Announcement

> **Service Weaver** began as an exploratory initiative to understand the challenges of developing, deploying, and maintaining distributed applications. We were excited by the strong interest from the developer community, which led us to open-source the project.
>
> We greatly appreciate the continued advocacy and support of the Service Weaver community. However, we realized that it was hard for users to adopt Service Weaver directly since it required rewriting large parts of existing applications. Therefore, Service Weaver did not see much direct use, and **effective December 5,**

What did we learn?

- Multi-tier programming: writing one program for multiple deployment “tiers”
- Often require specialized programming languages (or code annotations)
- Kompile shows: we *could* do a “multi-tier-like” system for a generic PL




THERE'LL BE NO YAML THEN.
I'LL TELL THE CHILDREN.

Related Reading and Links

- A Survey of Multitier Programming (Weisenberger, Wirth, Salvaneschi)
- Distributed System Development with ScalaLoci (Weisenberger, Köhler, Salvaneschi)
  - https://scala-loci.github.io
- Towards Modern Development of Cloud Applications (Ghemawat et al)
  - https://serviceweaver.dev
- Kompile