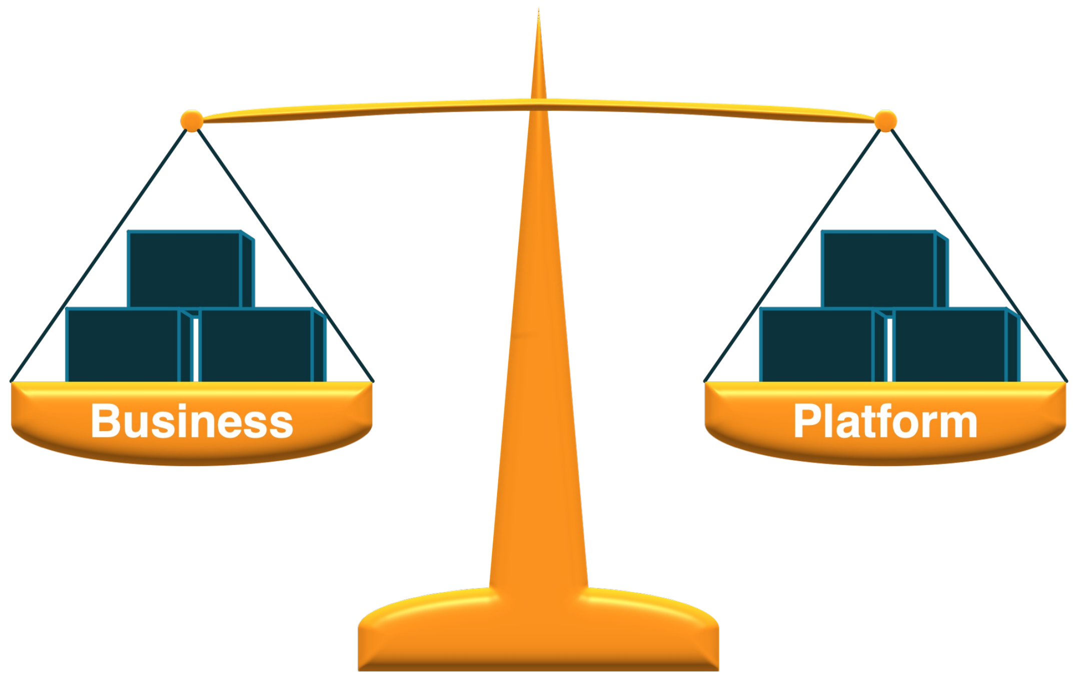
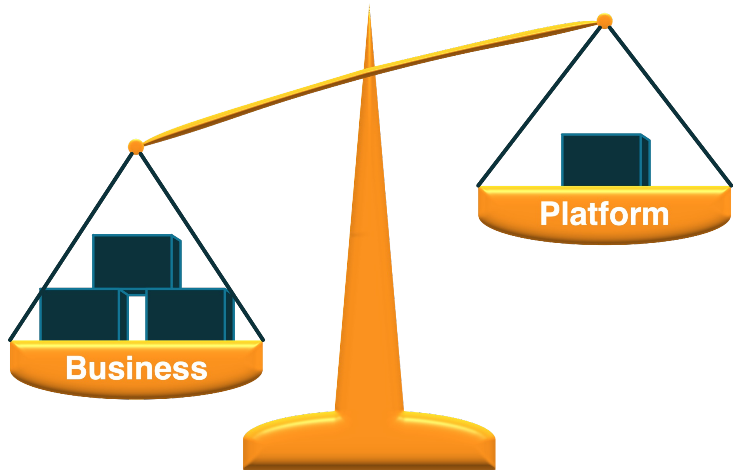
Serverless Applications

Business vs Platform



- Software development is a battle between solving *business* problems, and solving *platform* problems.
- Complex business applications require complex platforms.
- Complex platforms require resources, developers, and money.
- Building the platform distracts us from the *real* goal of solving business problems.

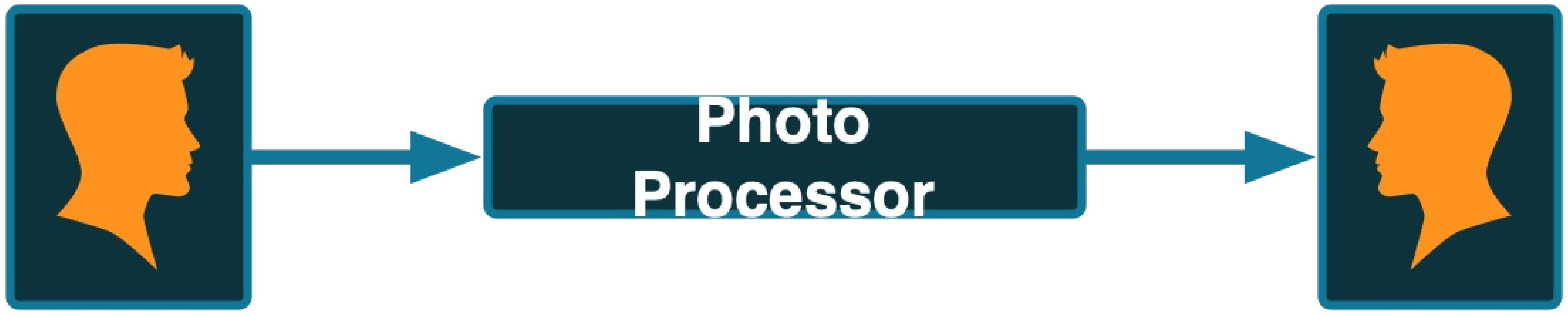
Enter Serverless Computing



- The dream of software development:
  - Focus on the *business* problems.
  - Leave the *platform* to someone else.
- Infrastructure such as deployment, scalability, and communication don't directly deliver business value.
- Traditional serverless platforms manage these concerns for you.
- Developers focus on the business problem, not the infrastructure needed to support it.
- However, these platforms often ignore *State*.

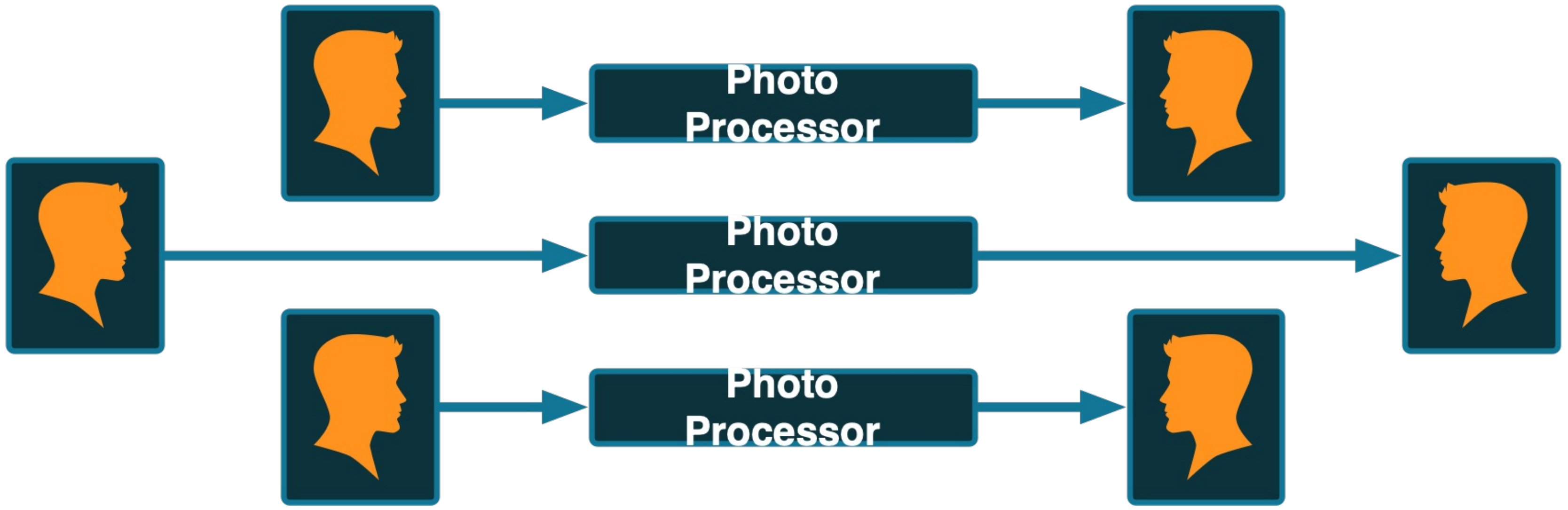
Stateless Applications

Stateless



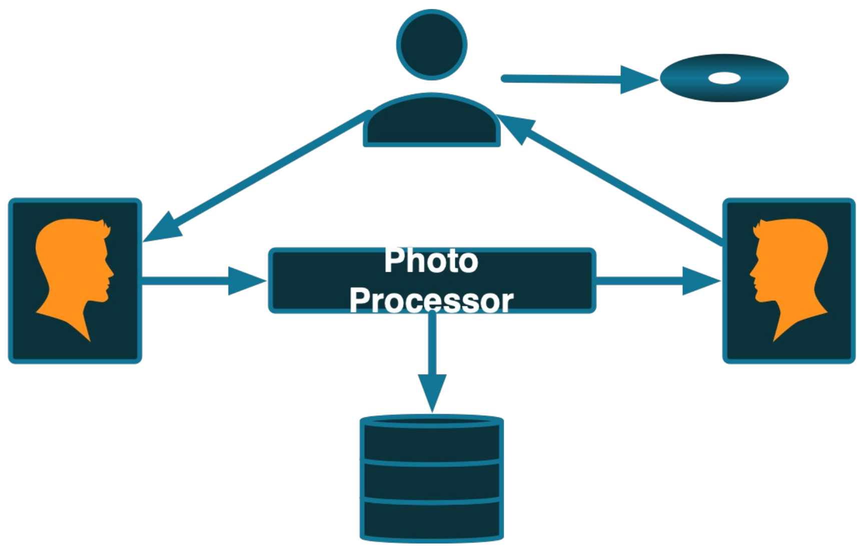
- If each request doesn't use *State* from previous requests, then the application is *Stateless*.
- Requests can be handled in-memory, with no data storage required.
- *Stateless* applications have few shared resources, and are highly scalable.
- If you need more power, you can simply deploy more instances.

Managing Stateless Applications



- Serverless works very well for *Stateless* applications.
- The platform monitors latency.
- As load increases, latency goes up.
- The platform can add new instances to compensate.
- More instances means more processing power.
- Latency decreases.

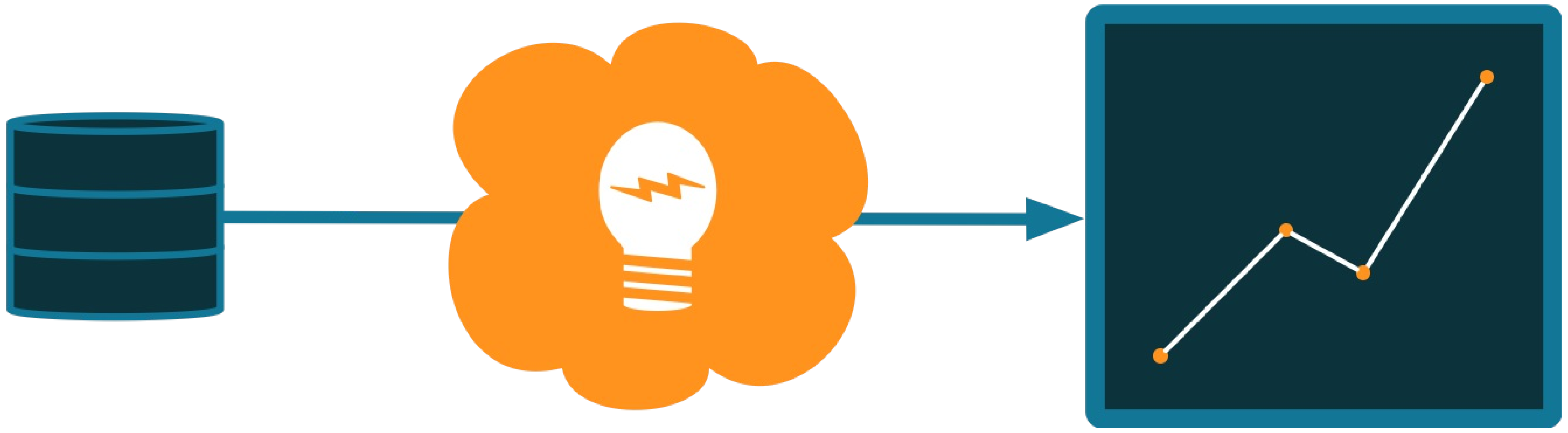
Where is the State?



- All applications have *State*. In a *Stateless* application, we either:
  - Require users to manage that *State*.
  - Delegate the *State* to an external component (eg. Database).
- *Stateless* applications don't eliminate *State* management, they just defer it.
- It's "somebody else's problem."

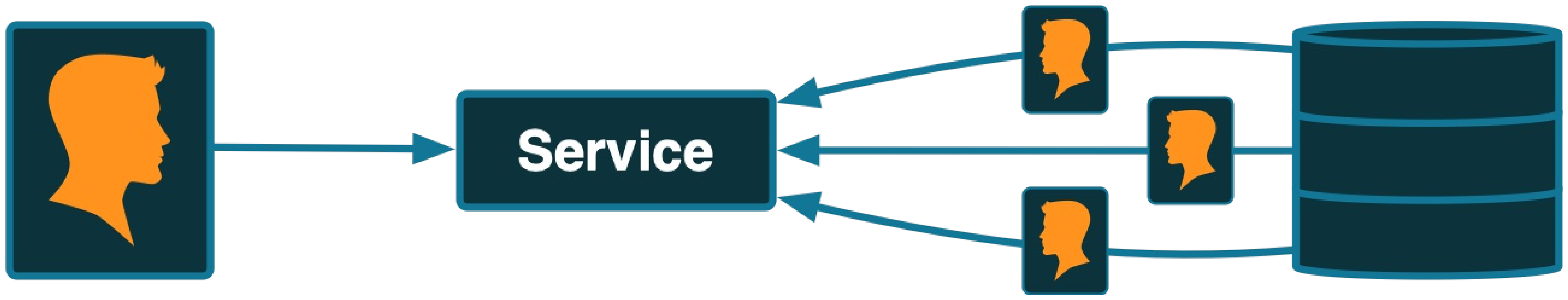
Managing State in a Serverless Application

Can a Business be Stateless?



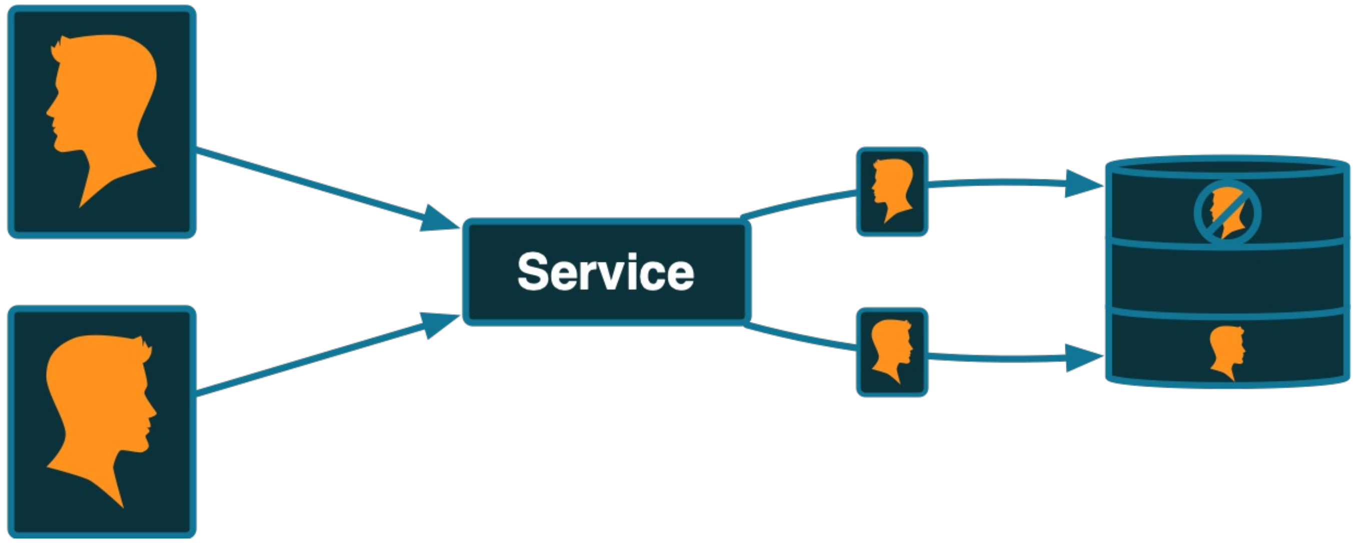
- Managing *State*, manipulating it, and deriving insight from it, is what drives businesses.
- Regardless of the business, you need to maintain some *State*.
- You can't push it back to your users. You have to handle it internally.
- The common solution is a database.

Reading Data



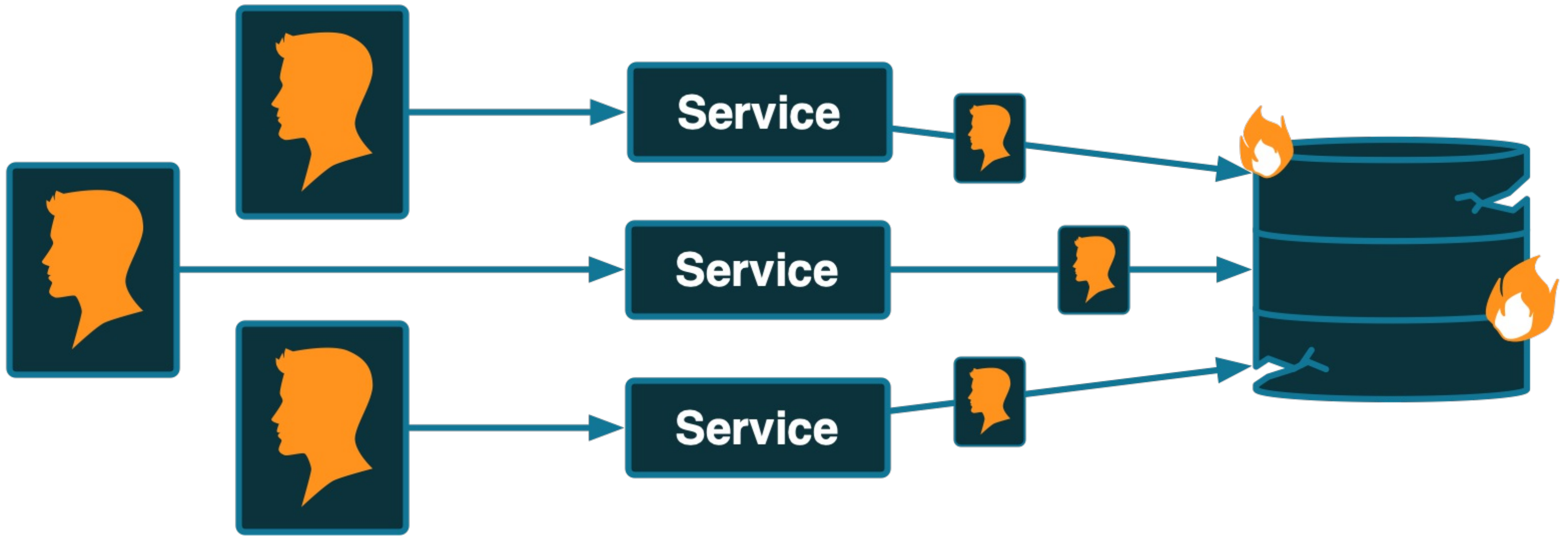
- Your serverless application needs access to the *State* in the database.
- For every request, one or more reads will be required.
- Many applications require multiple reads from different locations.
- Depending on the size and complexity this can take a lot of time.
- Data access is typically the slowest part of an application.
- We are left solving read time issues (a platform problem).

Writing Data



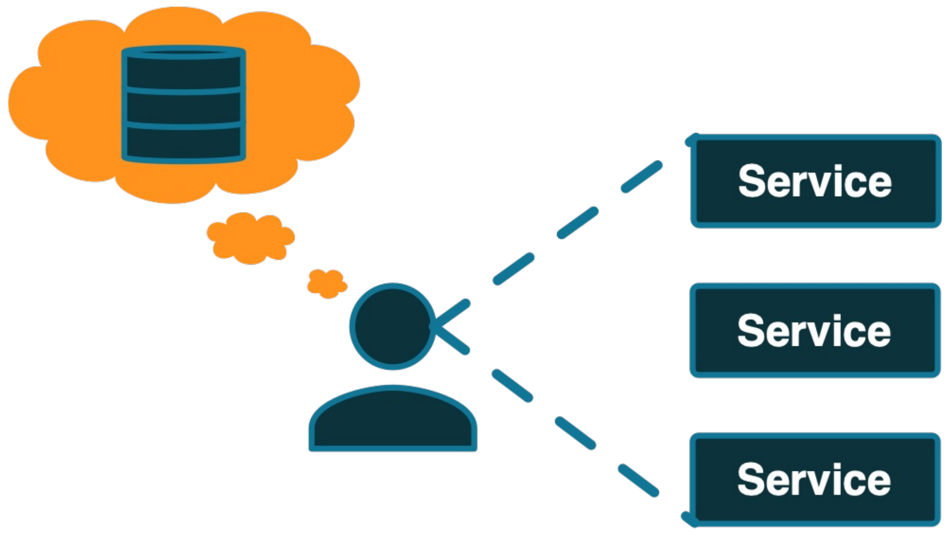
- For some requests, reading *State* is sufficient (read-only requests).
- However, many requests will alter the *State*.
- Typically, this is done by taking the old *State*, and overwriting it with the new.
- This destroys the old *State*.
- That old *State* may contain valuable insights.
- We are stuck thinking about database tables and queries (more platform problems).

Scaling Up



- Scaling is easy when there is no *State*.
- Introducing a database for *State* creates new problems.
- Scaling your application increases its processing power.
- But it also increases the load and contention on the database.
- We can try to scale the database, but this doesn't happen automatically (yet another platform problem).

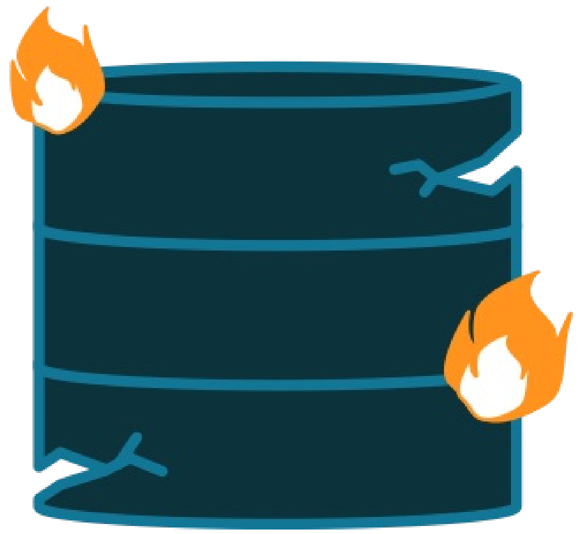
State as an Afterthought



- We originally went to serverless to avoid dealing with platform problems.
- Introducing *State* into our application created new platform problems.
- This happens because *State* management is not part of the platform.
- It's an afterthought.
- As a result, we've lost much of the benefit of serverless.

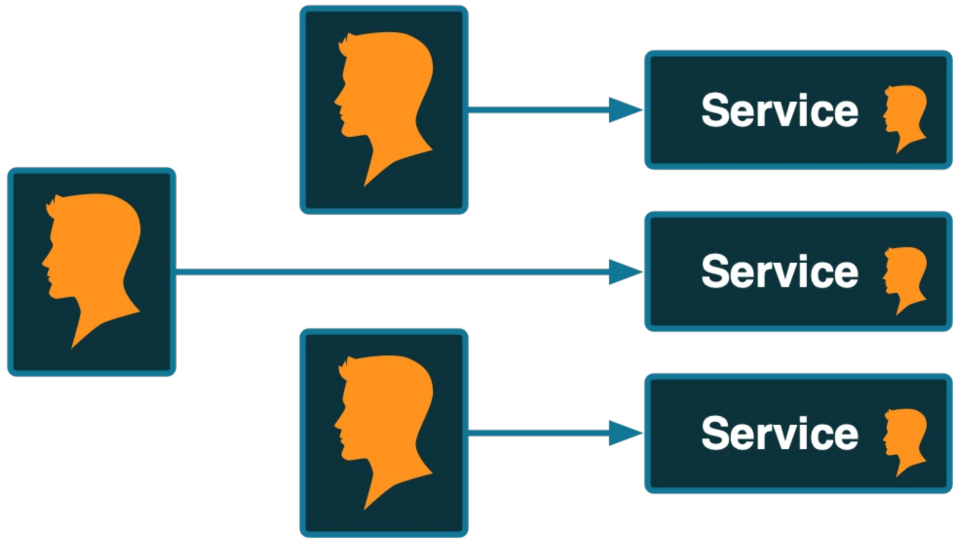
Stateful Serverless Applications

Problems with State and Serverless



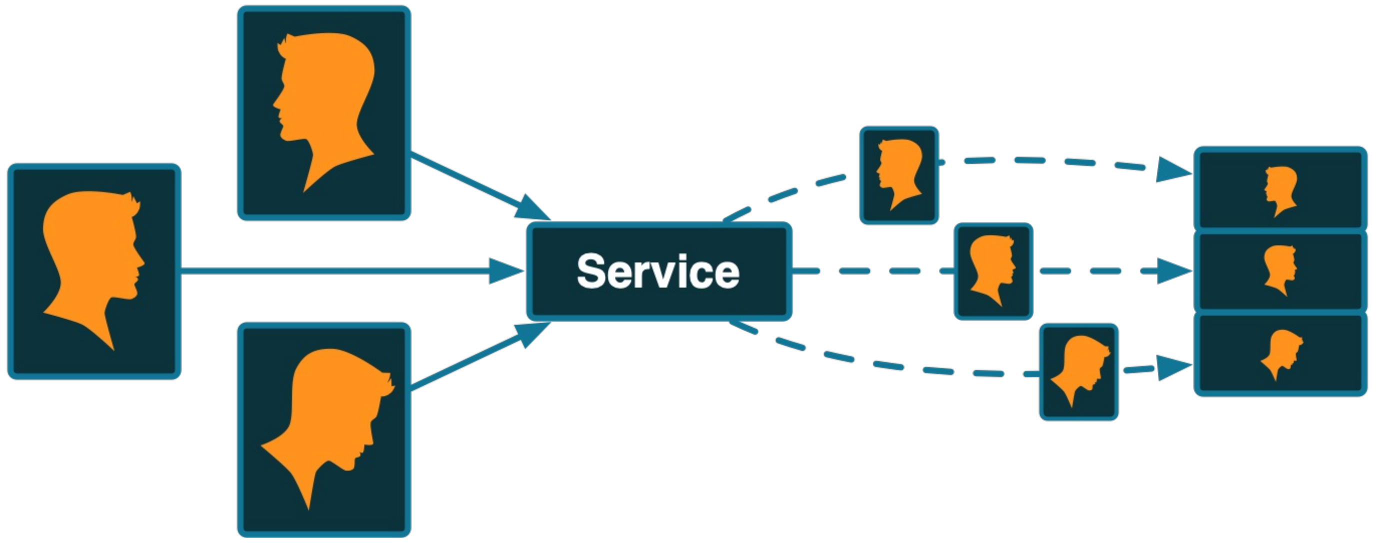
- We've highlighted some issues with how *State* is typically handled in a serverless platform.
  - Too many reads may slow down your application.
  - Valuable data is lost every time you update the *State*.
  - Scaling up your application has unintended consequences.
- These problems occur because *State* isn't part of the platform.
- Akka Serverless helps to alleviate this by moving *State* into the platform.

State Management



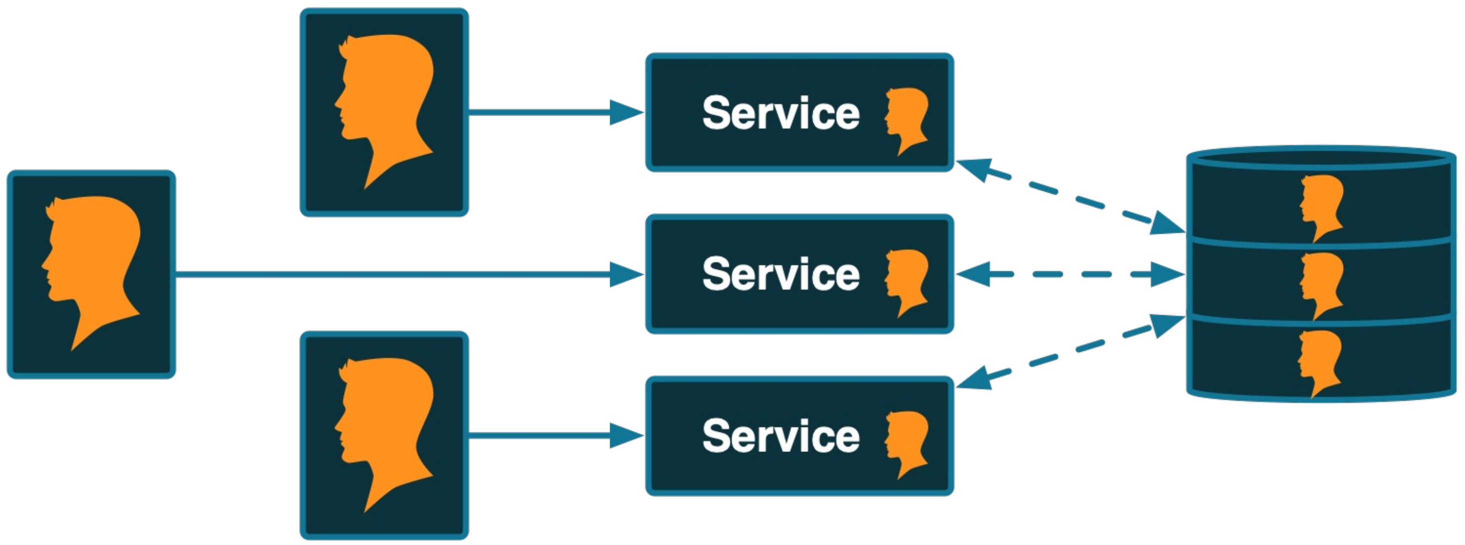
- Akka Serverless takes full responsibility for *State* management.
- Each instance of a *Service* manages a subset of the *State*
- Active *State* is kept in-memory.
- Accessing it doesn't require a database read.
- This helps reduce latency.

Event Sourcing



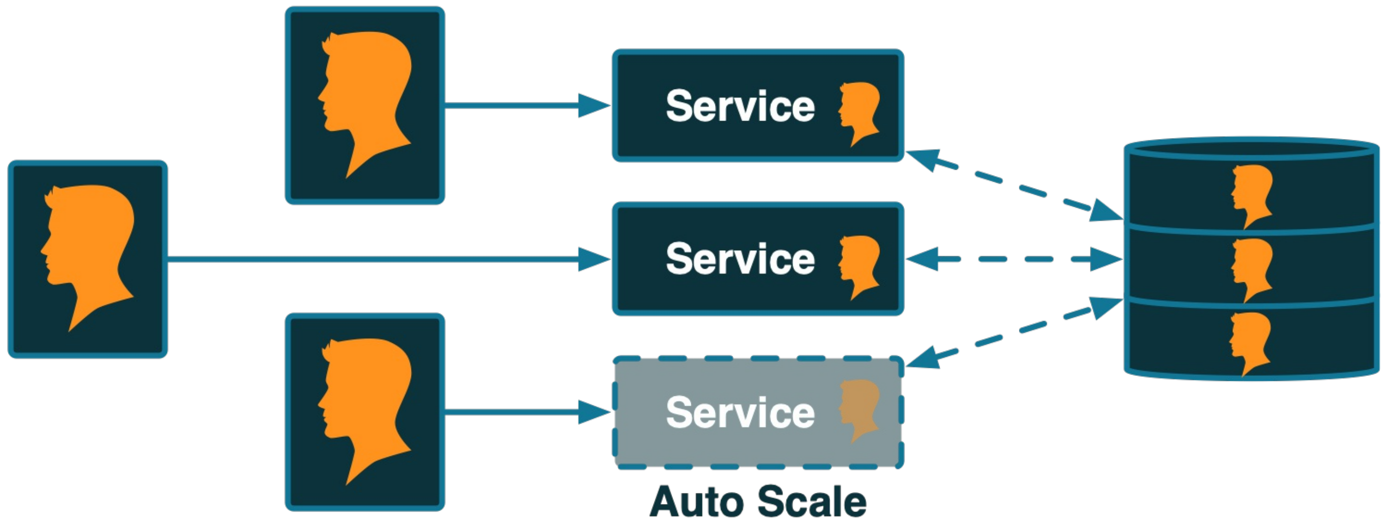
- To ensure *State* is durable, Akka Serverless uses *Event Sourcing*.
- An append-only journal of *Events* persists all changes.
- No data is ever lost.
- The journal can be used to recover *State*.
- But it we can also use it to retroactively derive business insights.

Consistent Latency



- Built-in *State* management has additional benefits at scale.
- When we scale an Akka Serverless application, we aren't just scaling its processing power.
- We also scale the *State* management capability.
- Reads come from memory, not from the database.
- Writes are small and efficient.
- This results in consistent latency, even at scale.

Intelligent Autoscaling



- Knowledge of your *State* allows for more intelligent auto-scaling.
- The platform has more information to figure out what to scale, and when.
- This ensures that you are only using (and paying for) the resources you need.

- *Akka Serverless* provides a *fully-managed stateful serverless* platform.
- You handle your business problems, *Akka Serverless* handles your platform problems.
- And all of this power is available in the language of your choice!