Climbing the Ladder of Abstraction

Towards the Cloud-to-Edge Continuum

Jonas Bonér
CEO & Founder Lightbend

jonasboner.com
@jboner

What we will discuss today

1. How the emerging Edge calls for a unified Cloud-to-Edge DX
2. How developer productivity is held back by overwhelming complexity, in Cloud and Edge
3. How we need to rethink and adapt to liberate developers and maximize business value

Edge Computing

“Edge computing is a distributed computing paradigm that brings computation and data storage closer to the sources of data.”

The Edge might look scary

But it
does not
have to be

The Edge is already here, and it's exciting

- **Telco**, **Cloud**, and **CDN** companies are building out the infrastructure
- **5G and Local-first software** is changing the game
- Customers demand **moving data and compute closer to end-user** for lower latency, better availability, and more advanced use-cases
- We are moving **towards stateful** and holistic view on **data and compute**
- **Huge market** on the rise: $4.6B in 2020 to $139.58B in 2030—Gartner

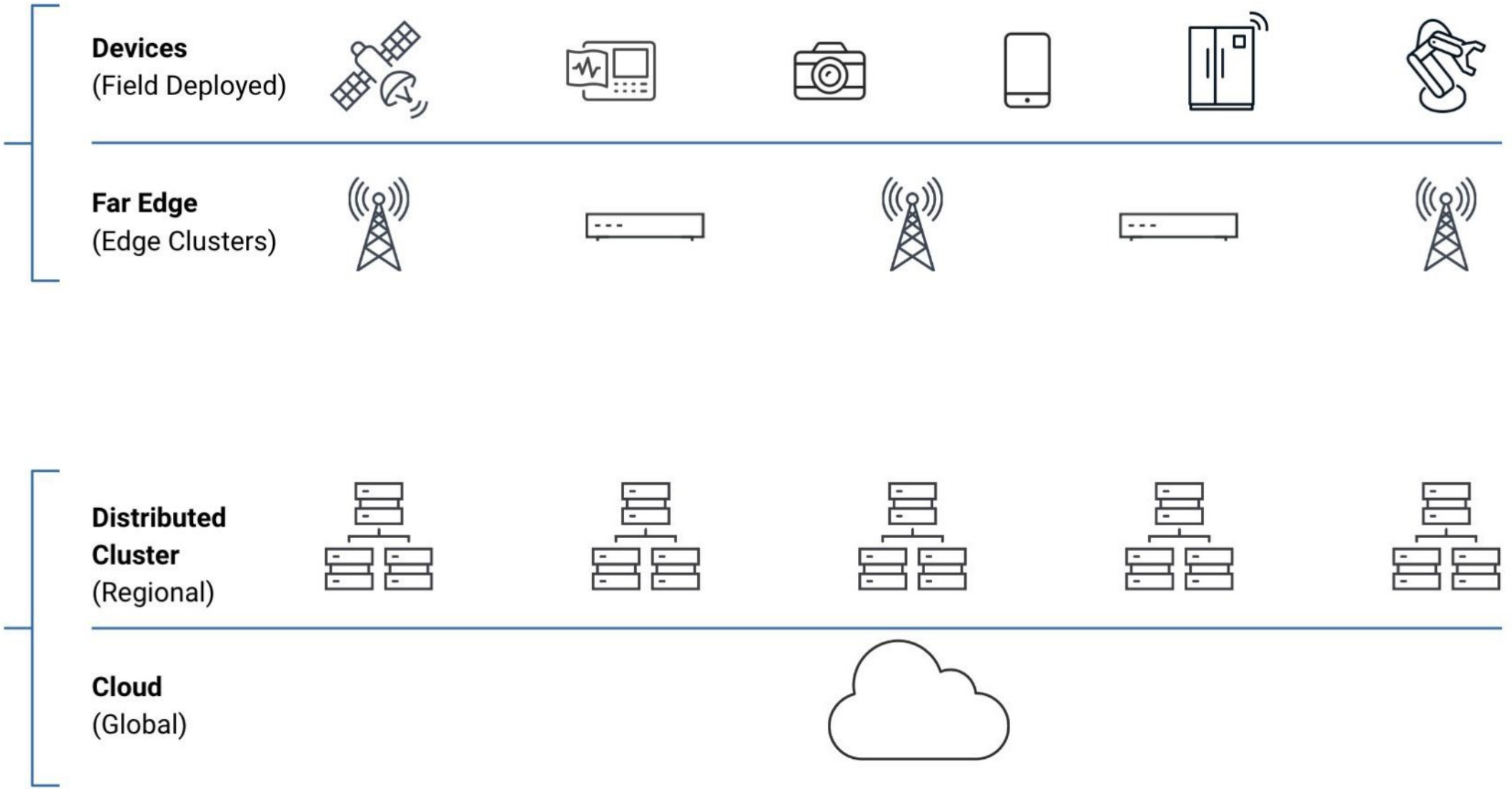
> Edge Computing is actually being implemented today in many of our clients' environments, [...] enabling entirely new applications and data models.
>
> Simply put, Edge has moved from concept and hype into successful vertical industry implementations, with general purpose platform status approaching rapidly.

Bob Gill
Gartner (Oct. 2021)

Tsunami Warning

- Gartner predicts that **by 2025**, **75%** of enterprise-generated data **will be created & processed on the Edge** (from 10% in 2018)
- **This requires:**
  - Keeping as much data on the Edge as possible
  - Move processing to the Edge
  - Physical Co-location of User, Data, Processing
  - Real-time Streaming of Data from Device-PoP / PoP-PoP / PoP-Cloud
- **Benefits:**
  - Real-time processing, lower latency, faster answers
  - Resilient, not relying on stable connection to cloud
  - More resource efficient, less data shipping

https://www.gartner.com/smarterwithgartner/what-edge-computing-means-for-infrastructure-and-operations-leaders

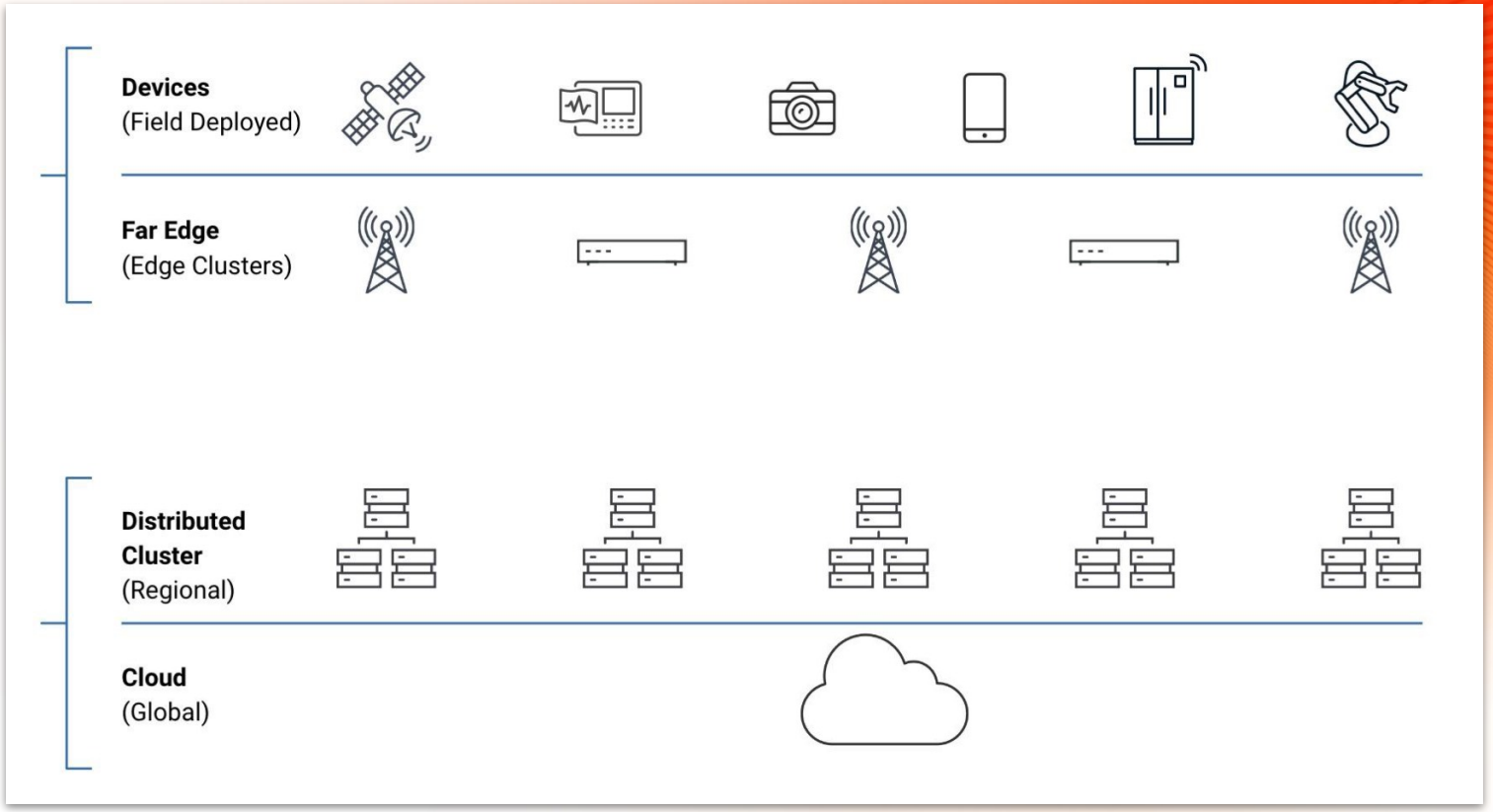

Devices
(Field Deployed)
Far Edge
(Edge Clusters)
Distributed
Cluster
(Regional)
Cloud
(Global)

The Problem:

Software designed to run at the Edge

…is not well-suited to run in the Cloud

And vice versa

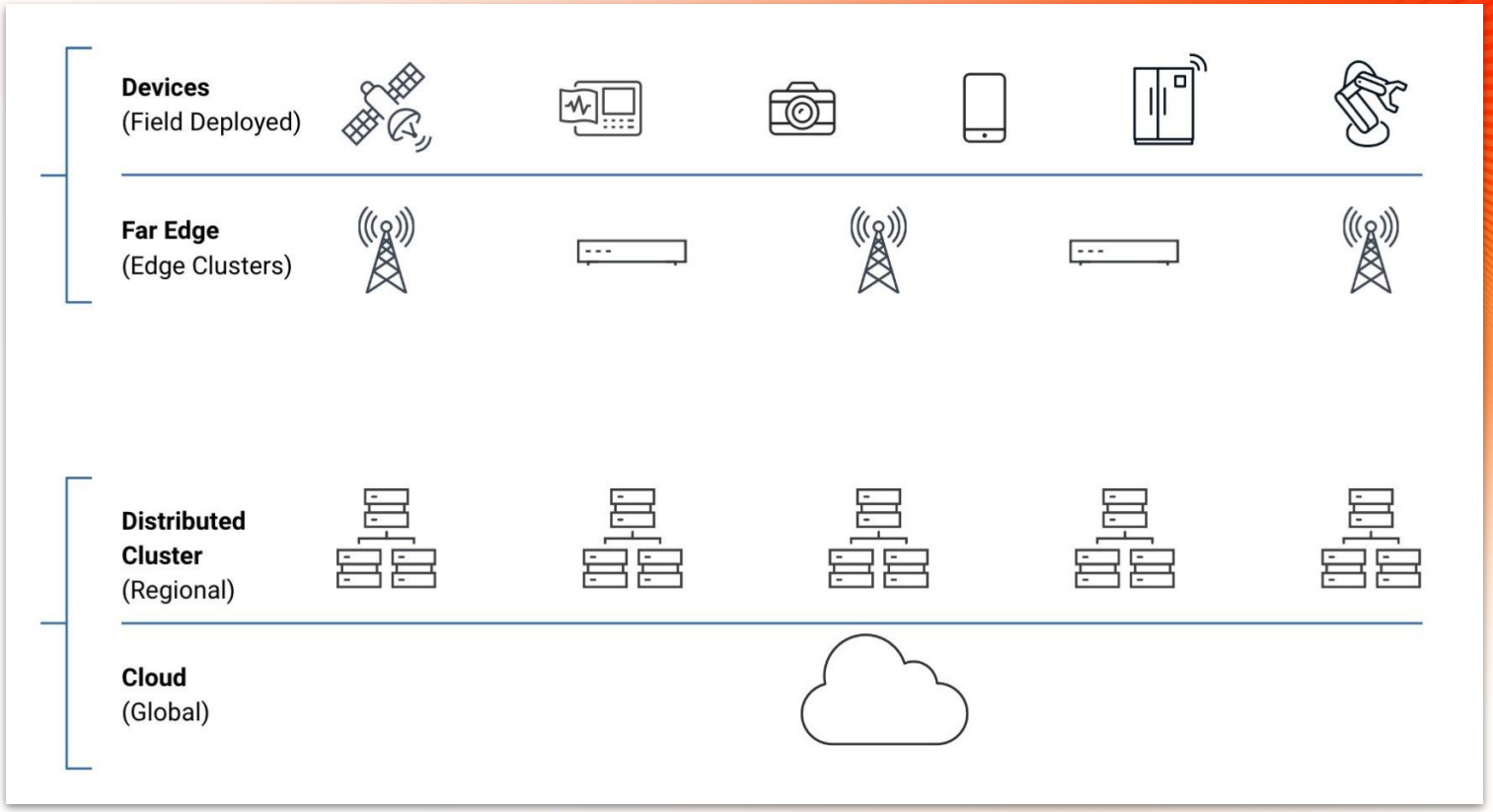


The Problem:

Software designed to run at the Edge

…is not well-suited to run in the Cloud

And vice versa



Further in (towards cloud):

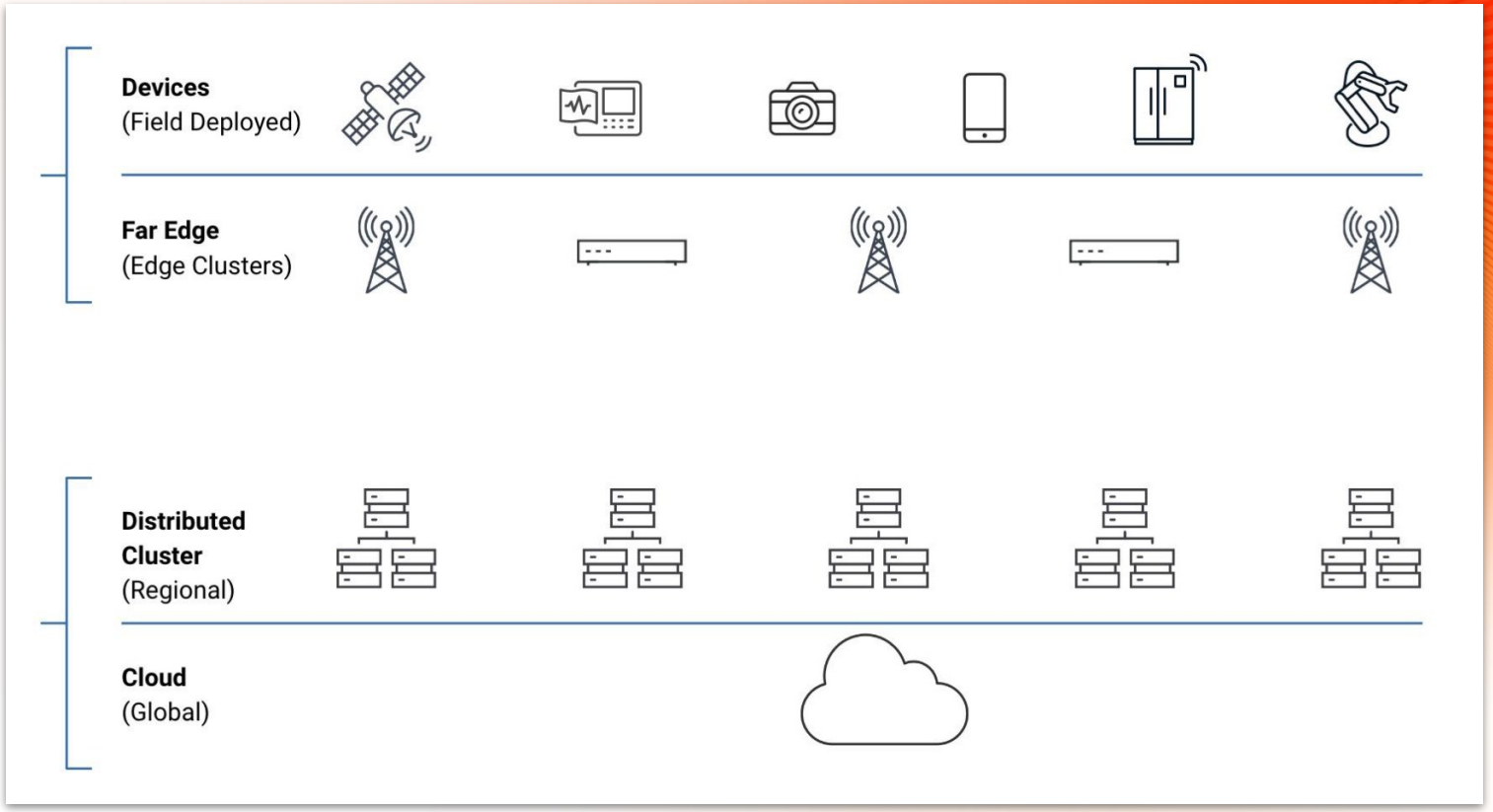
- 10s to 1000s of nodes to coordinate
- Reasonably reliable networks & hardware
- Vast resources & compute power
- Batch, high latency processing
- Allows stronger consistency/guarantees
- Global, slower, but “better” decisions
- Less resilient & available
- Coarse-grained data replication possible, data can be stationary

The Problem:

Software designed to run at the Edge

…is not well-suited to run in the Cloud

And vice versa



Further in (towards cloud):

- 10s to 1000s of nodes to coordinate
- Reasonably reliable networks & hardware
- Vast resources & compute power
- Batch, high latency processing
- Allows stronger consistency/guarantees
- Global, slower, but “better” decisions
- Less resilient & available
- Coarse-grained data replication possible, data can be stationary

Further out (towards devices):

- 10,000s to 100,000s of PoPs to coordinate
- Unreliable networks & hardware
- Limited resources & compute power
- Real-time, low latency processing
- Needs weaker consistency/guarantees
- Local, faster, less accurate decisions
- More resilient & available
- Requires fine-grained data replication and mobility

Why Cloud to Edge Matters for Businesses

“Data is the new oil”

– Clive Humby, OBE

Low Latency

Speed matters

- Making operational or business decisions as quickly as possible is essential

Resiliency

Availability matters

- Network/cloud/outages and device failures cannot be allowed to impact your business

Operational Efficiency

Cost matters

- Shipping data round the planet is expensive … as is the cost of the infrastructure to process it

Data Sovereignty

Compliance matters

- Increasingly important and impactful data privacy and data location laws need to be followed

Common Edge Use-Cases

- Autonomous vehicles
- Retail & E-commerce
- Trading systems
- Health care
- Emergency services
- Factories
- Stadium events (sport, concerts)
- Gaming
- Farming
- Financial services
- Smart homes
- And more…

My vision of a Cloud to Edge Data Plane

Cloud to Edge is a **Continuum**

- Powering location transparent **services that can run anywhere**, from public Cloud to 100,000 of PoPs at the Edge
- **Should not be** a **design**, **development**, or even **deployment** decision

My vision of a Cloud to Edge Data Plane

Cloud to Edge is a **Continuum**

- Powering location transparent **services that can run anywhere**, from public Cloud to 100,000 of PoPs at the Edge
- **Should not be** a **design**, **development**, or even **deployment** decision

Data always **exists wherever** and **whenever needed**, and only **for the time** it is needed

- **Data** and **compute moves adaptively, together** with the end-user
- **Data** is **always co-located** with processing and end-user, ensuring ultra low latency, high throughput and resilience
- **Data** is **injected into the services** on an as-needed basis, automatically, timely, efficiently, and intelligently

Technical Requirements

- **Autonomous self-organising** components
- **Physical co-location** of data, compute, end-user
- **Intelligent and adaptive placement** of data, compute, end-user

- **Local-first** cooperation (function just fine without backend cloud)
- **Fine-grained** and **adaptive replication** (row-level)

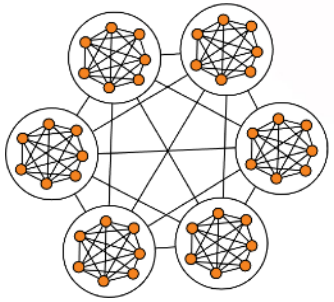
- **Consistency a la carte**—Eventual, Causal, Strong
- **Options for "state models" ("**shapes")—SQL, Event Sourcing, CRDTs, Views, Streaming

- **End-to-end guarantees** of SLAs, business rules, and data integrity

Reactive Event-driven Actor-based Platform for Distributed Systems

akka

Reactive Event-driven Actor-based Platform for Distributed Systems



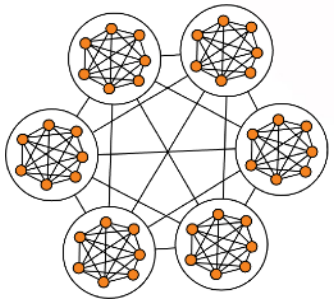
Low Latency

Fastest possible speed

Performance matters

Making operational or business decisions as quickly as possible is essential

Reactive Event-driven Actor-based Platform for Distributed Systems

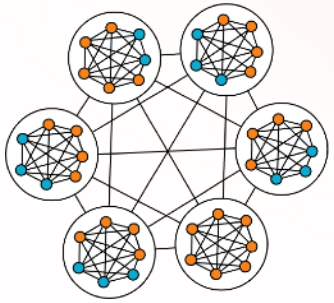


Low Latency

Fastest possible speed

Performance matters

Making operational or business decisions as quickly as possible is essential



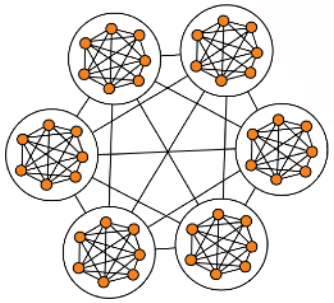
Scalable

Scales up and down

Efficiency matters

Cloud infrastructure is usually one of the largest IT expenses for an enterprise – only use what you need

Reactive Event-driven Actor-based Platform for Distributed Systems

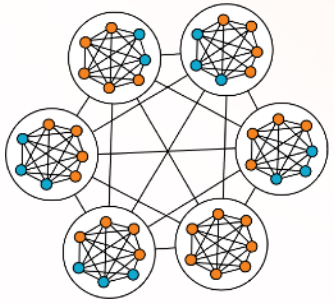


Low Latency

Fastest possible speed

Performance matters

Making operational or business decisions as quickly as possible is essential

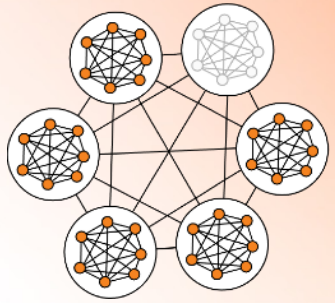


Scalable

Scales up and down

Efficiency matters

Cloud infrastructure is usually one of the largest IT expenses for an enterprise – only use what you need



Reliability

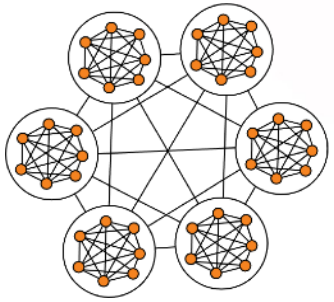
Systems are self-healing

Availability matters

Network / cloud outages and device failures cannot be allowed to impact your business

akka

Reactive Event-driven Actor-based Platform for Distributed Systems

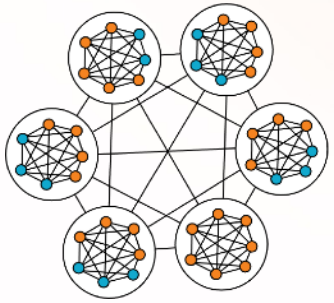


Low Latency

Fastest possible speed

Performance matters

Making operational or business decisions as quickly as possible is essential

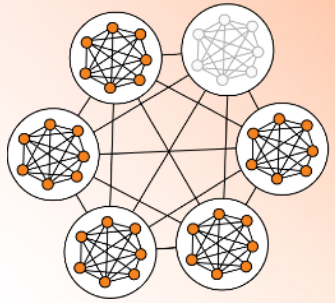


Scalable

Scales up and down

Efficiency matters

Cloud infrastructure is usually one of the largest IT expenses for an enterprise – only use what you need

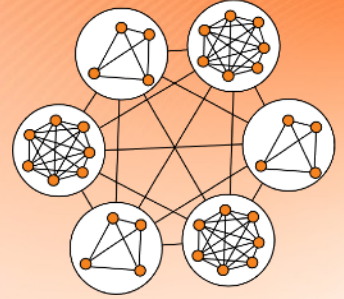


Reliability

Systems are self-healing

Availability matters

Network / cloud outages and device failures cannot be allowed to impact your business



Efficiency

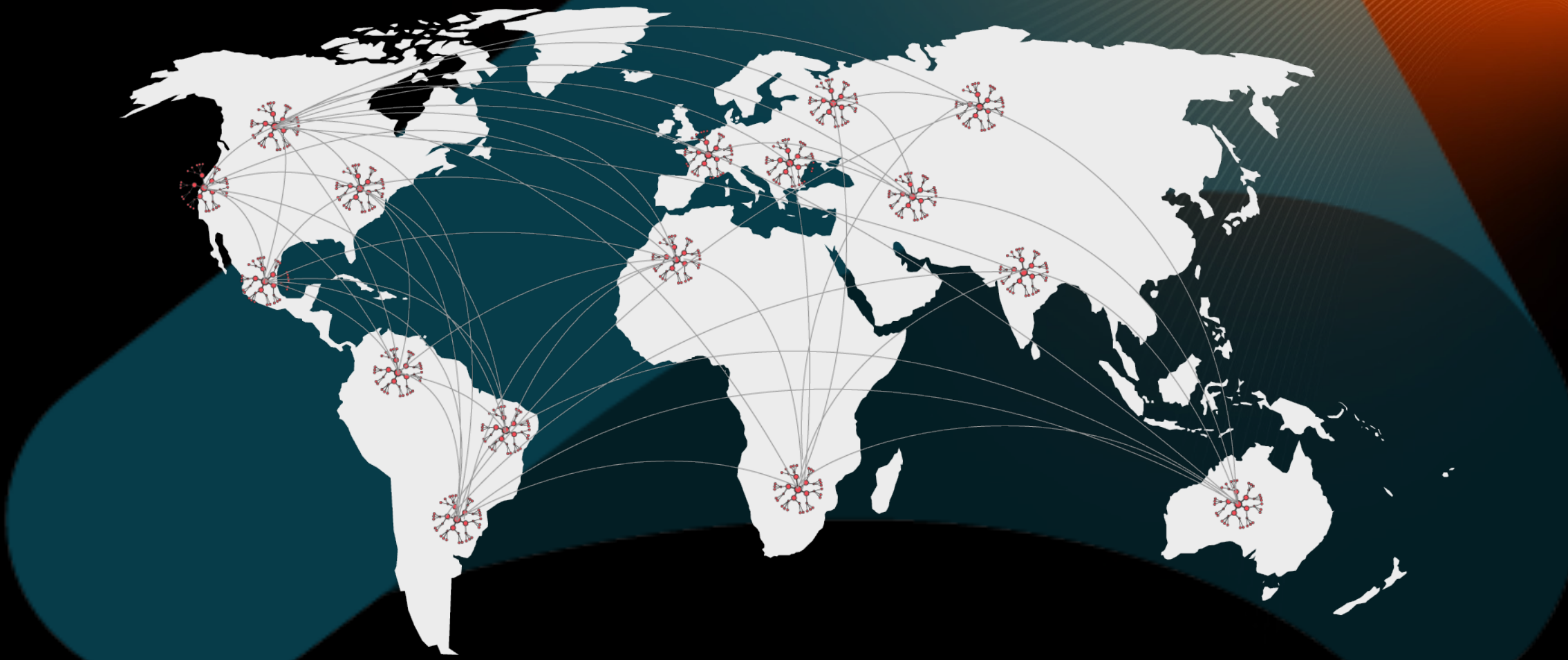
Uses less infrastructure

Cost matters

Shipping data round the planet is expensive … as is the cost of the infrastructure to process it

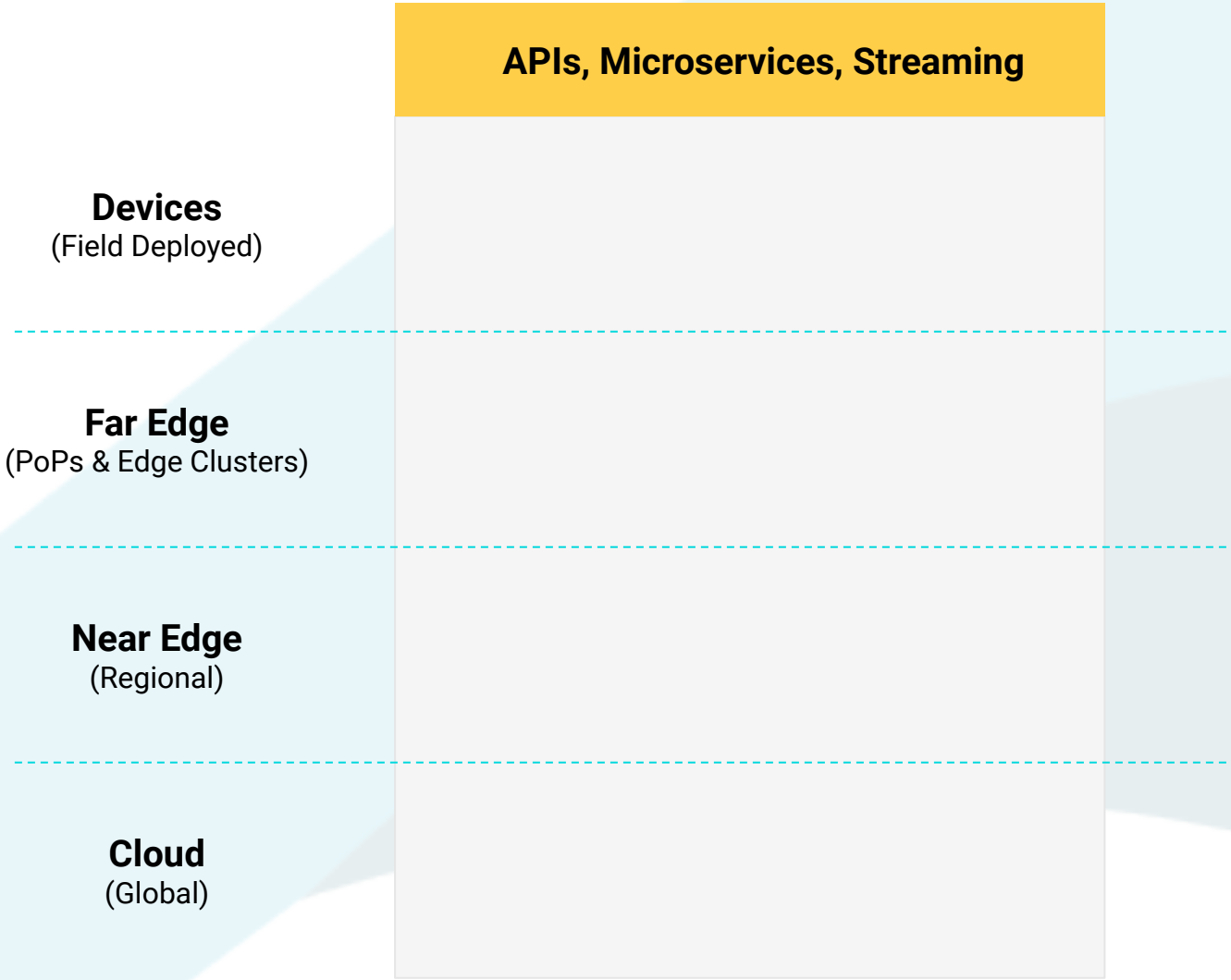
Akka Automatically Manages Distributed Data at Scale

Data is always available when needed
in the right place, at the right time, for the right duration



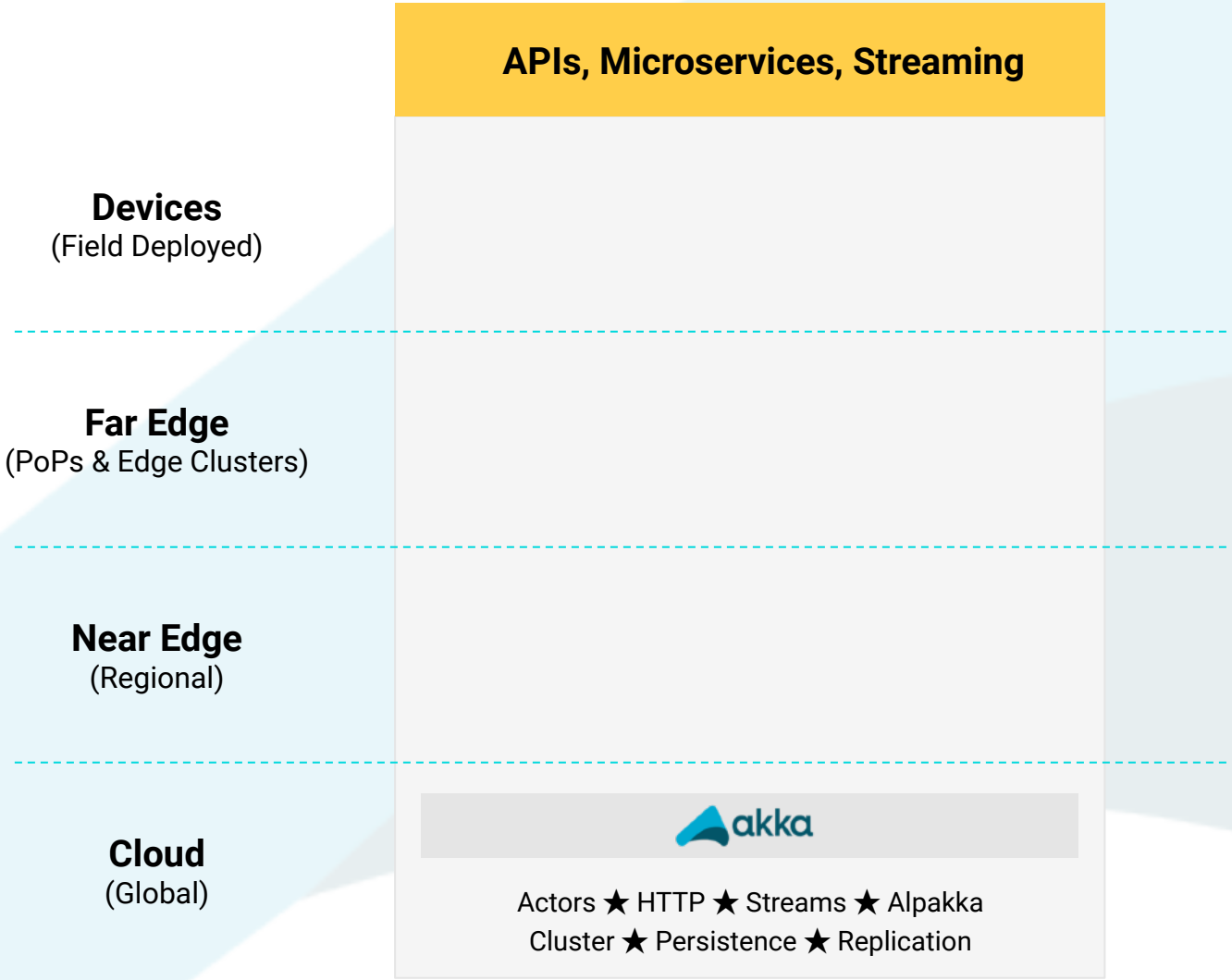
Akka Edge: Tackling the Cloud to Edge Continuum

Build once, using a single programming model. **Run anywhere**, on optimal infrastructure.



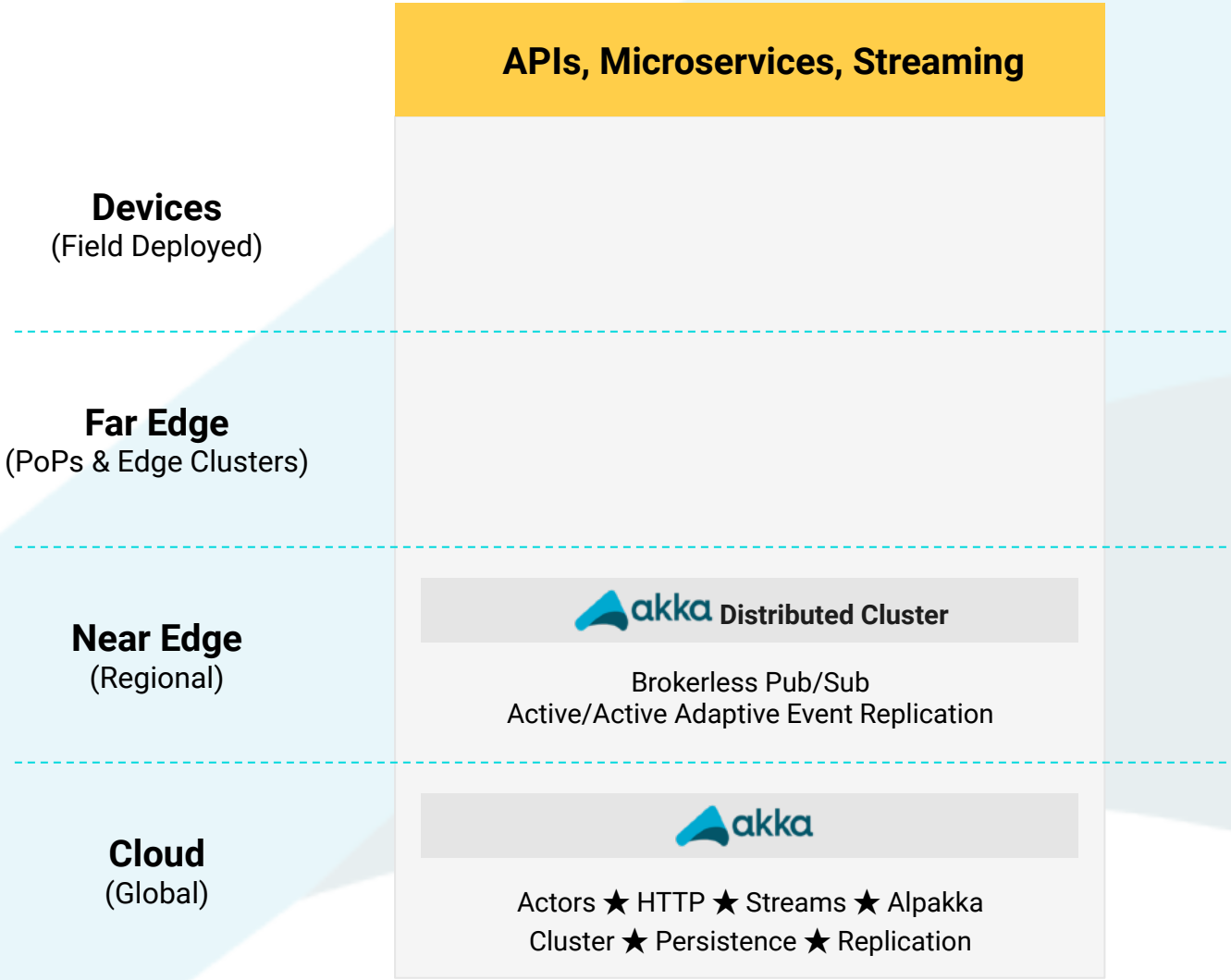
Akka Edge: Tackling the Cloud to Edge Continuum

Build once, using a single programming model. **Run anywhere**, on optimal infrastructure.



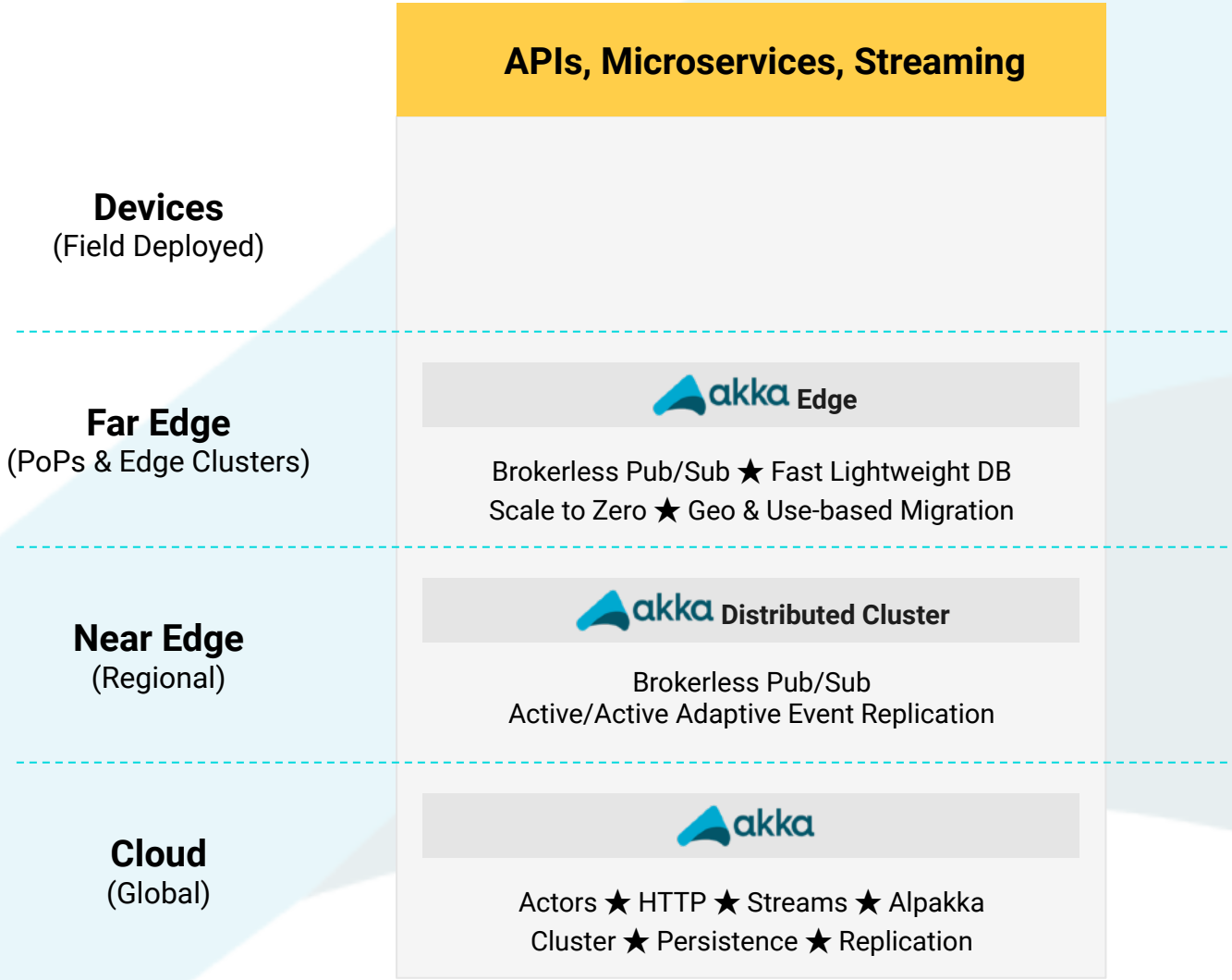
Akka Edge: Tackling the Cloud to Edge Continuum

Build once, using a single programming model. **Run anywhere**, on optimal infrastructure.



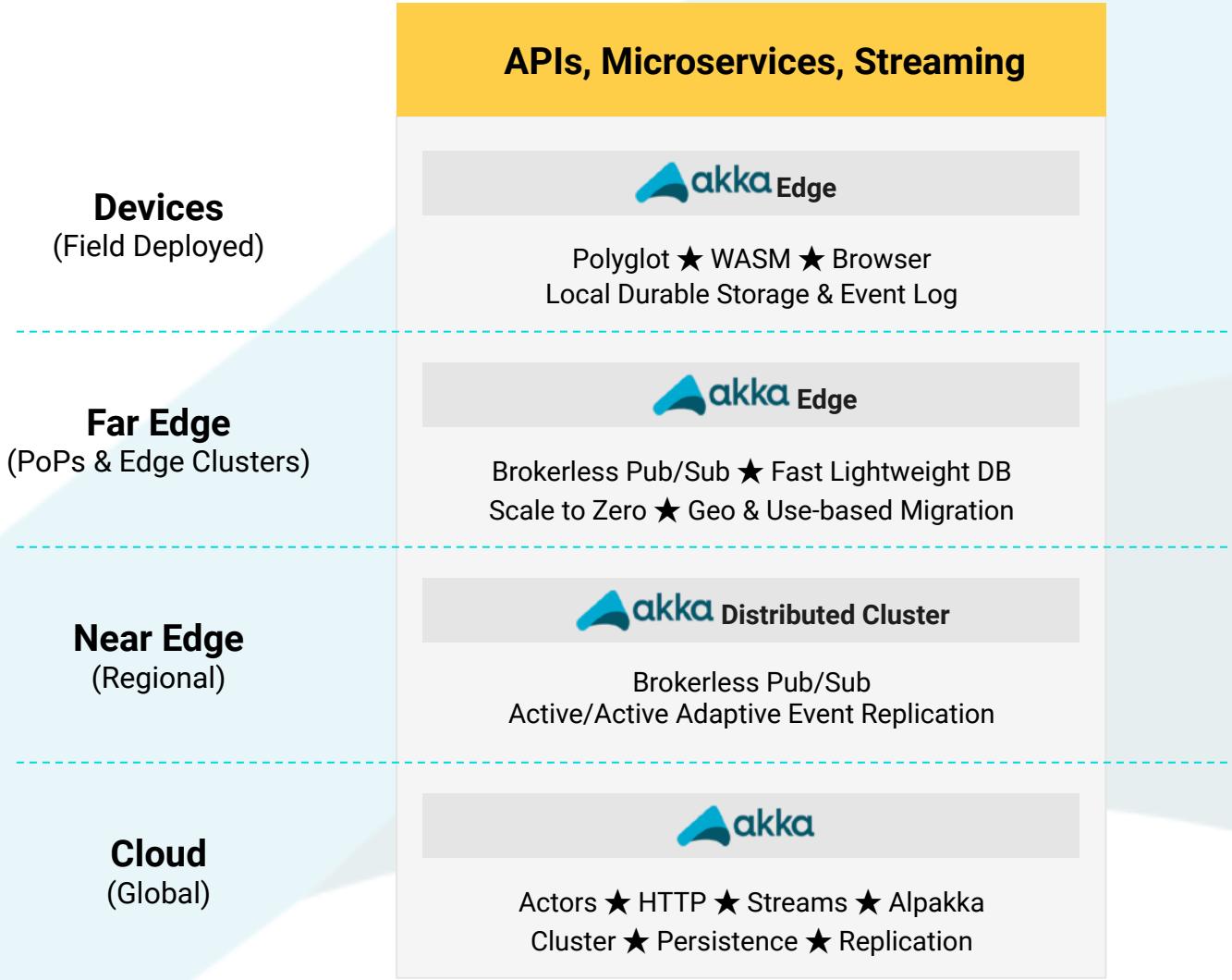
Akka Edge: Tackling the Cloud to Edge Continuum

Build once, using a single programming model. **Run anywhere**, on optimal infrastructure.



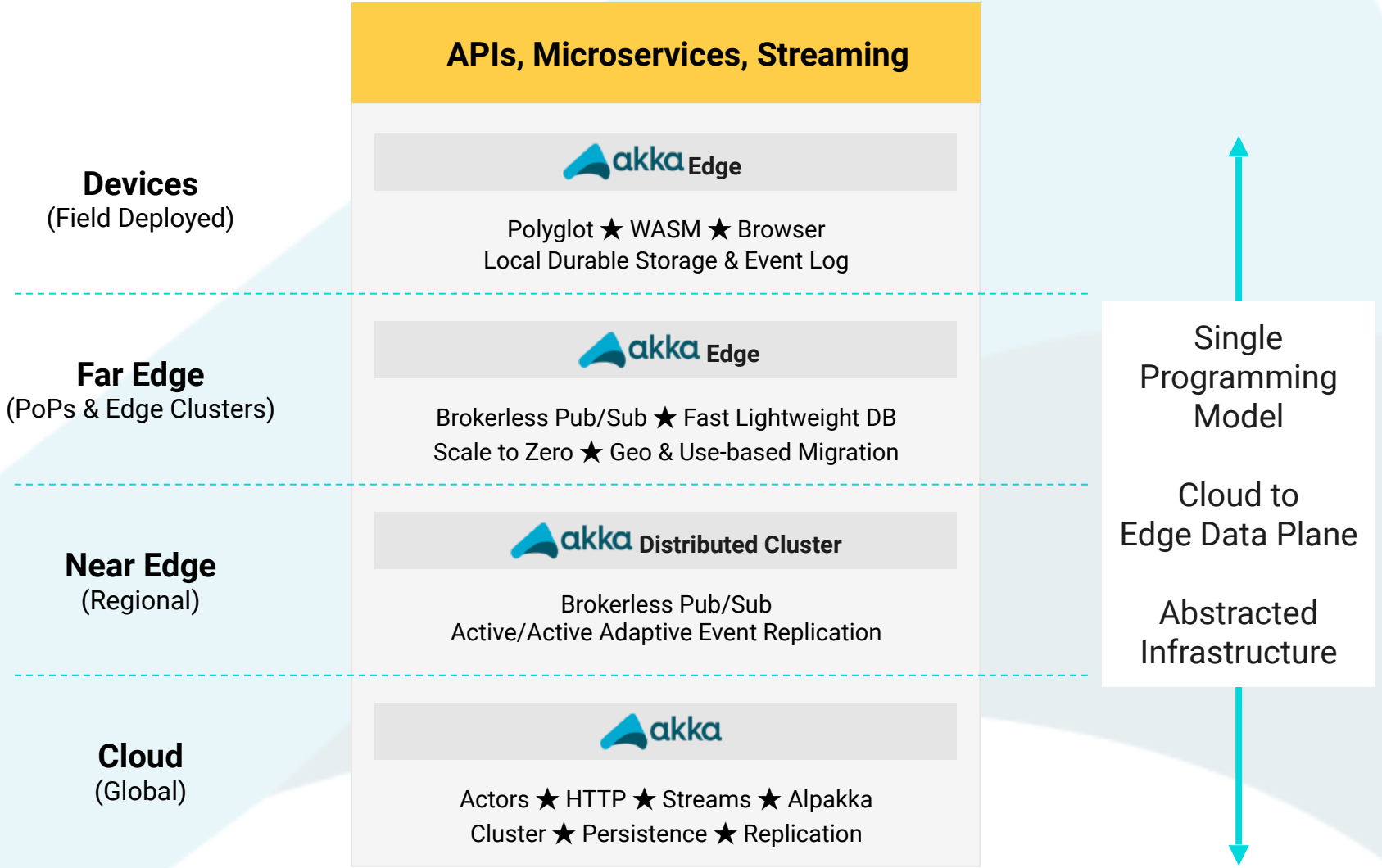
Akka Edge: Tackling the Cloud to Edge Continuum

Build once, using a single programming model. **Run anywhere**, on optimal infrastructure.



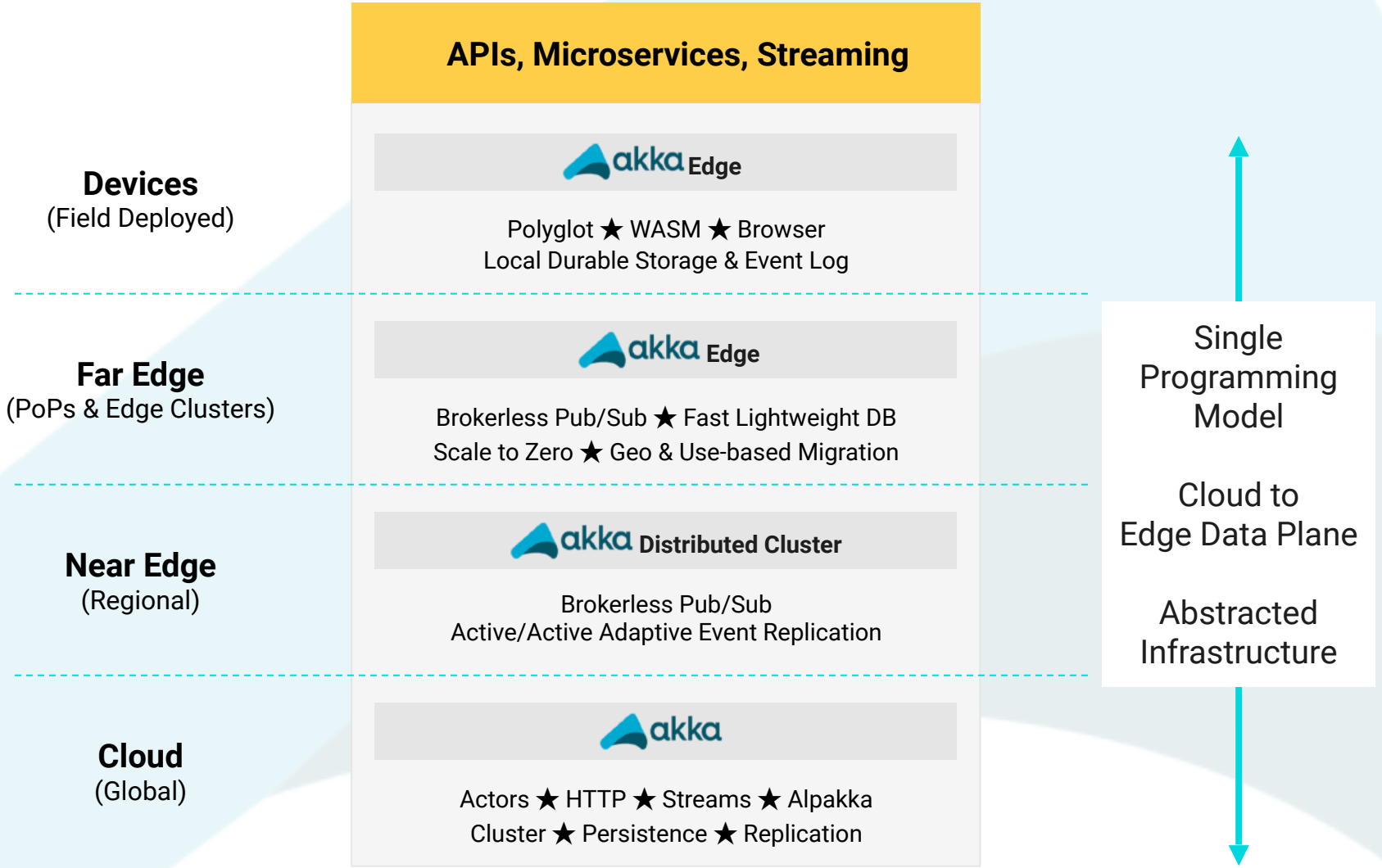
Akka Edge: Tackling the Cloud to Edge Continuum

Build once, using a single programming model. **Run anywhere**, on optimal infrastructure.



Akka Edge: Tackling the Cloud to Edge Continuum

Build once, using a single programming model. **Run anywhere**, on optimal infrastructure.




But we have a
big problem
to solve

We All Want To Do

More With Less

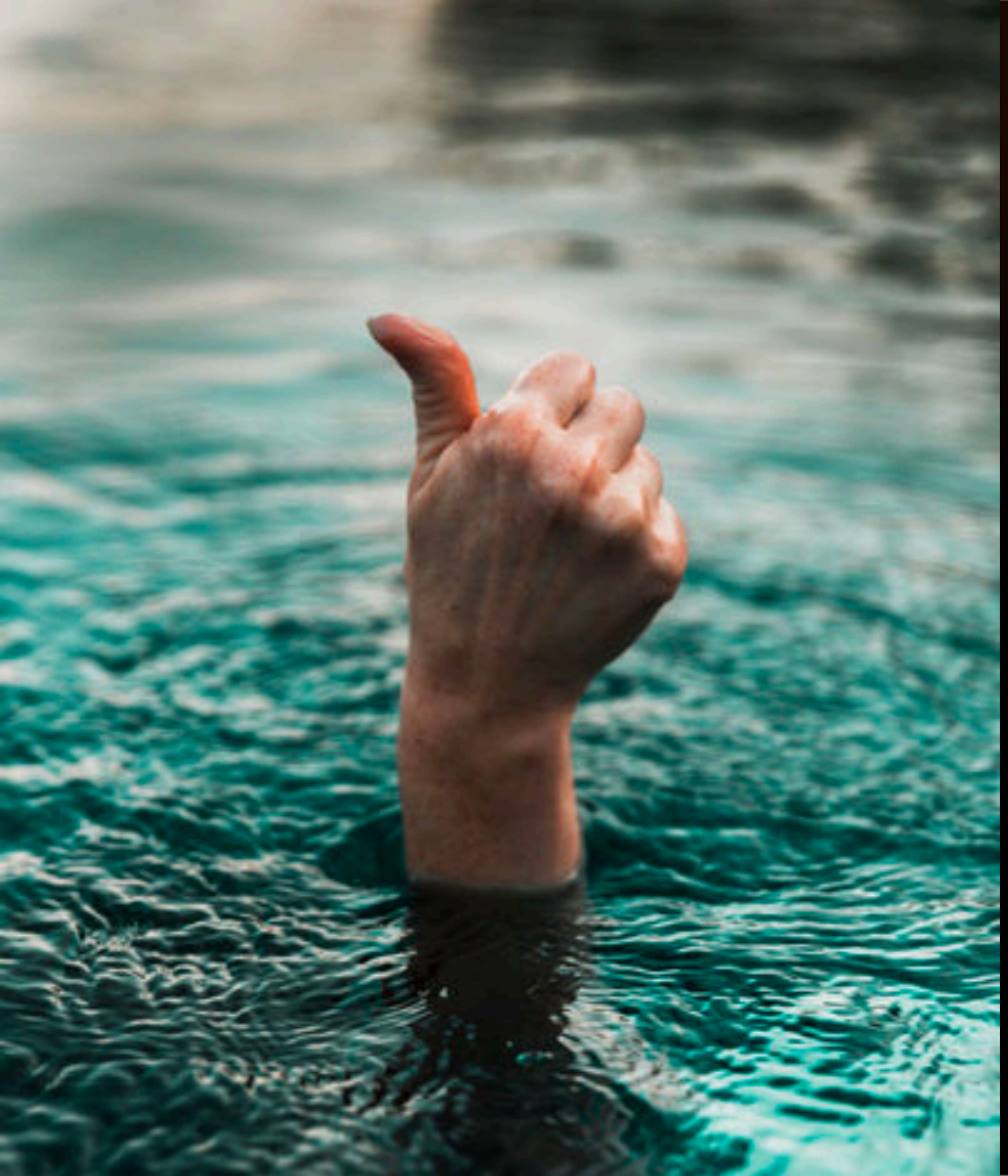
- Faster Time To Market
- Cost Efficiency
- Energy Efficiency

With:

- Predictability
- Repeatability
- Reusability

Today's Cloud & Edge infrastructure is

Great



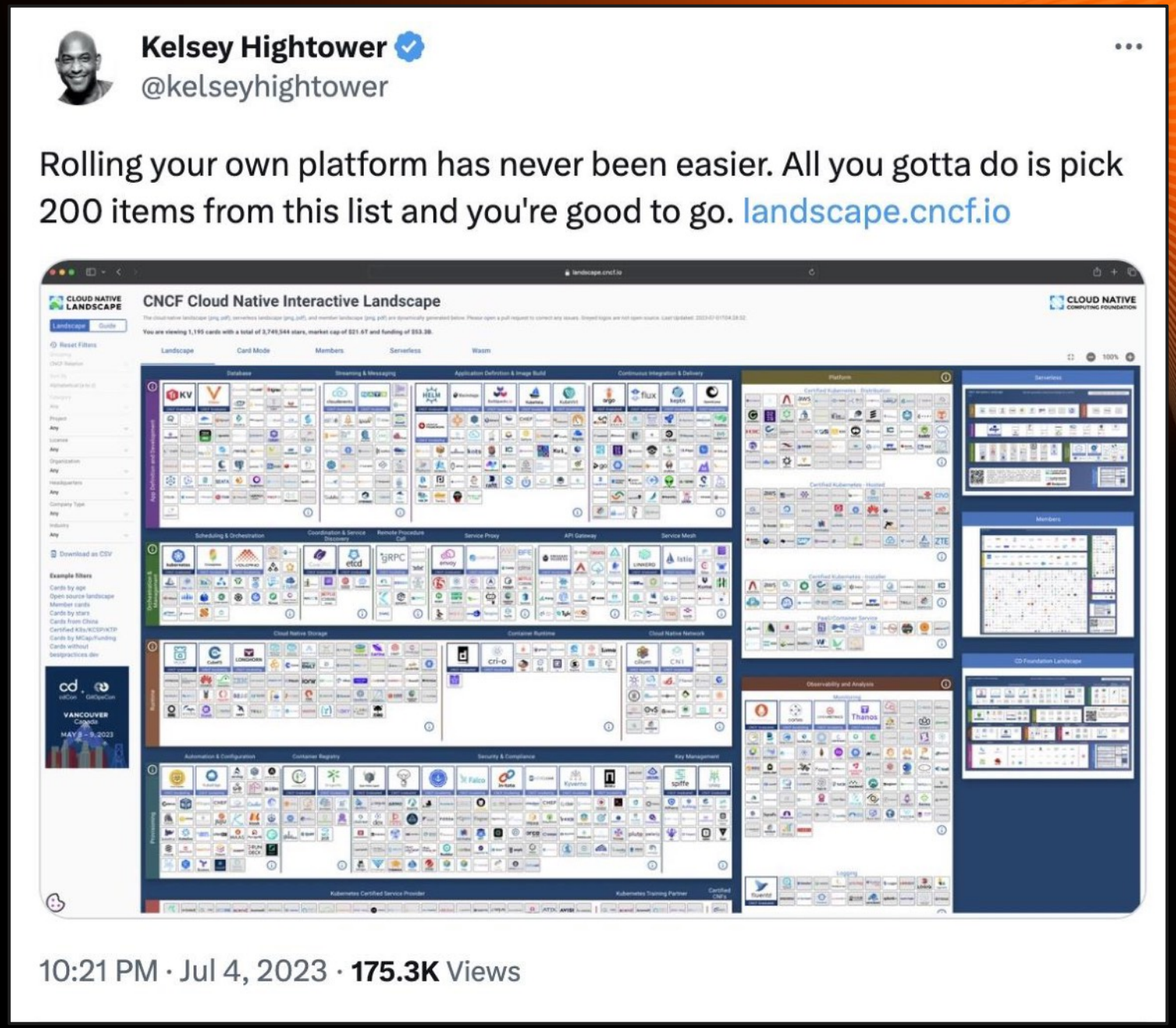
But we are

drowning in complexity

The options are
overwhelming
Too many
Decisions

The options are overwhelming

Too many Decisions



> There are already too many primitives for engineers to deeply understand and manage them all, and more arrive by the day.
>
> And even if that were not the case, there is too little upside for the overwhelming majority of organizations to select, implement, integrate, operate and secure every last component or service.
>
> Time spent managing enterprise infrastructure minutiae is time not spent building out its own business.

Stephen O'Grady, RedMonk
Vertical Integration: The Collision of App Platforms and Database


The Cloud is the New Computer

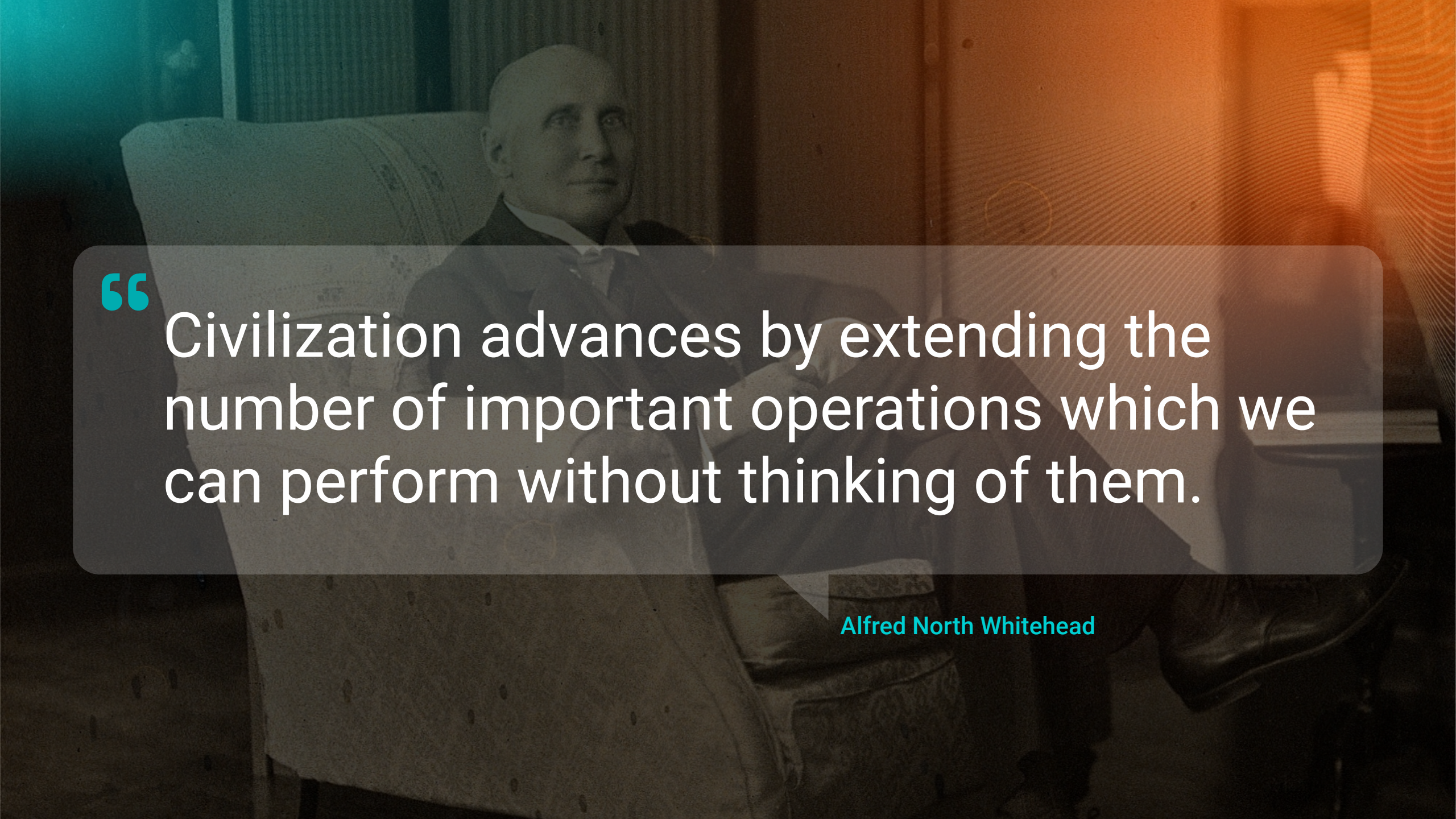

The Cloud is the New Computer


The Cloud is the New Computer
The Problem:


The Cloud is the New Computer
The Problem:
To use it we are expected to
“hack the kernel” or write
“device drivers”


The Cloud is the New Computer
The Problem:
To use it we are expected to
“hack the kernel” or write
“device drivers”


The Cloud is the New Computer
The Problem:
To use it we are expected to
“hack the kernel” or write
“device drivers”
This is not sustainable!!!

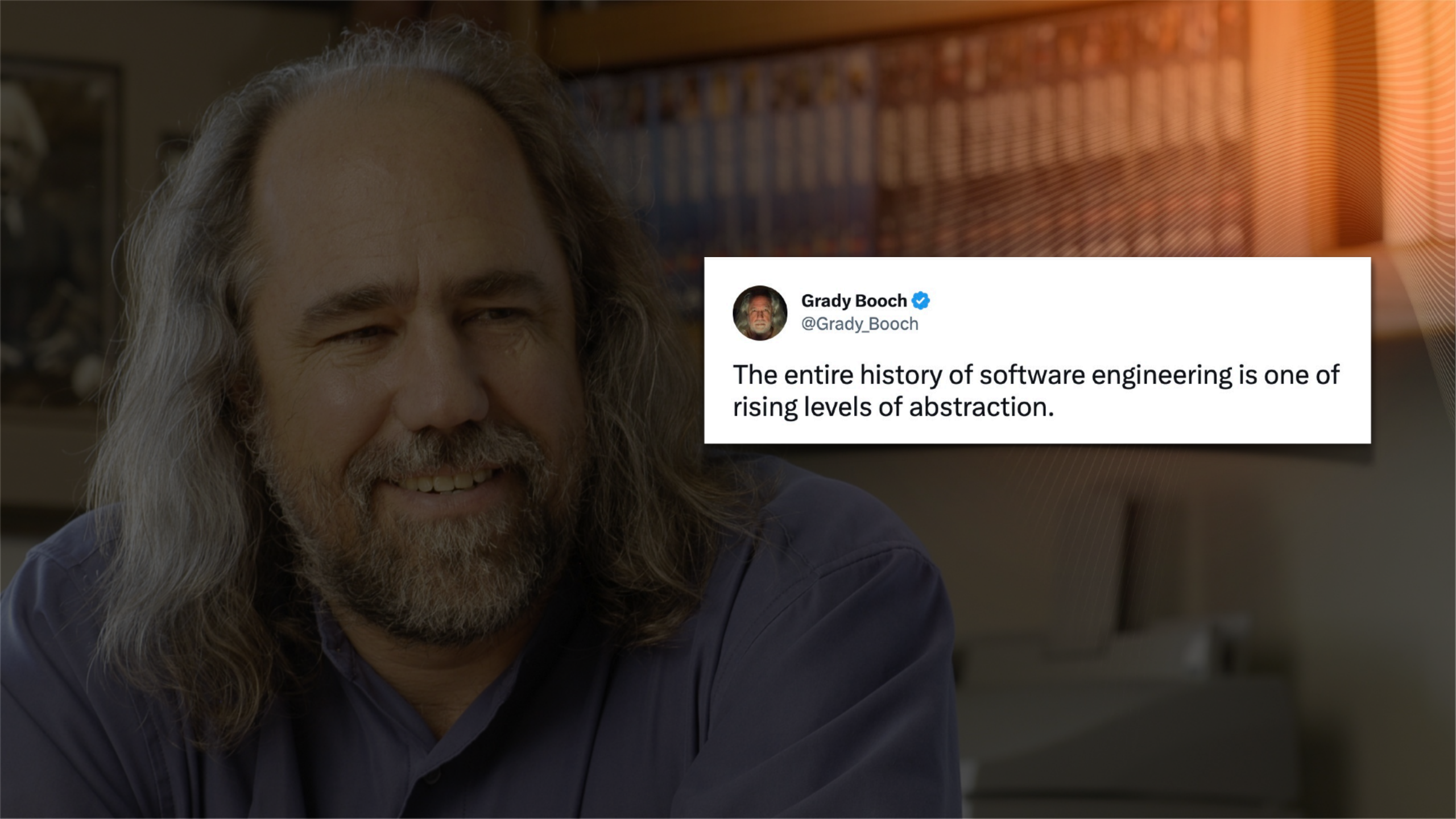


> Civilization advances by extending the number of important operations which we can perform without thinking of them.

Alfred North Whitehead

We Need To Continue to Climb

Ladder of Abstraction


Grady Booch
@Grady_Booch
The entire history of software engineering is one of rising levels of abstraction.


Let Go of Control
You Don't Need
Every Knob

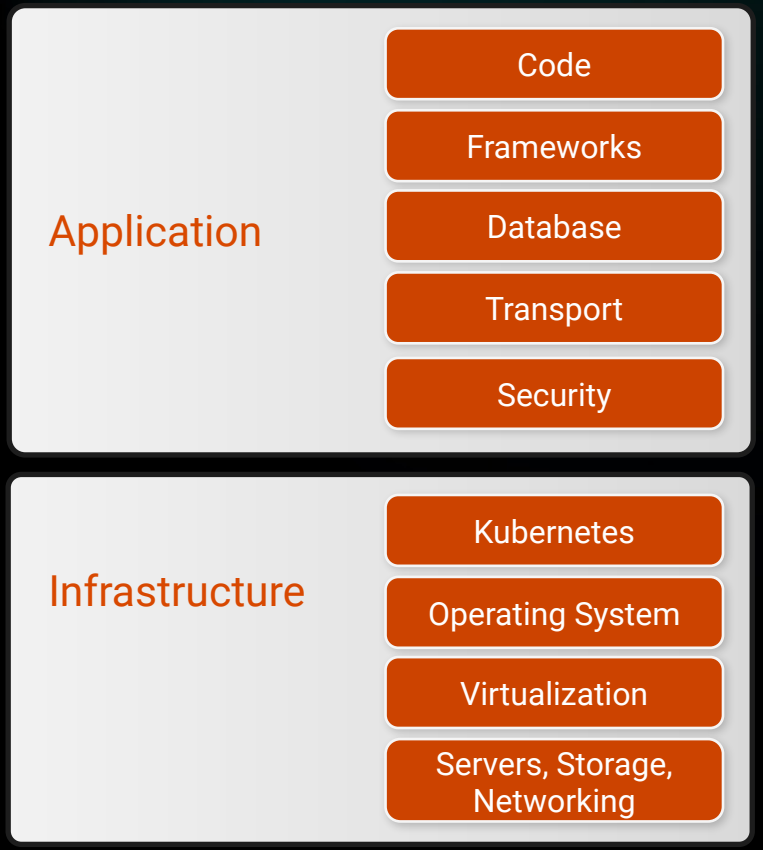
Infrastructure Inferred From Code

Customer Managed

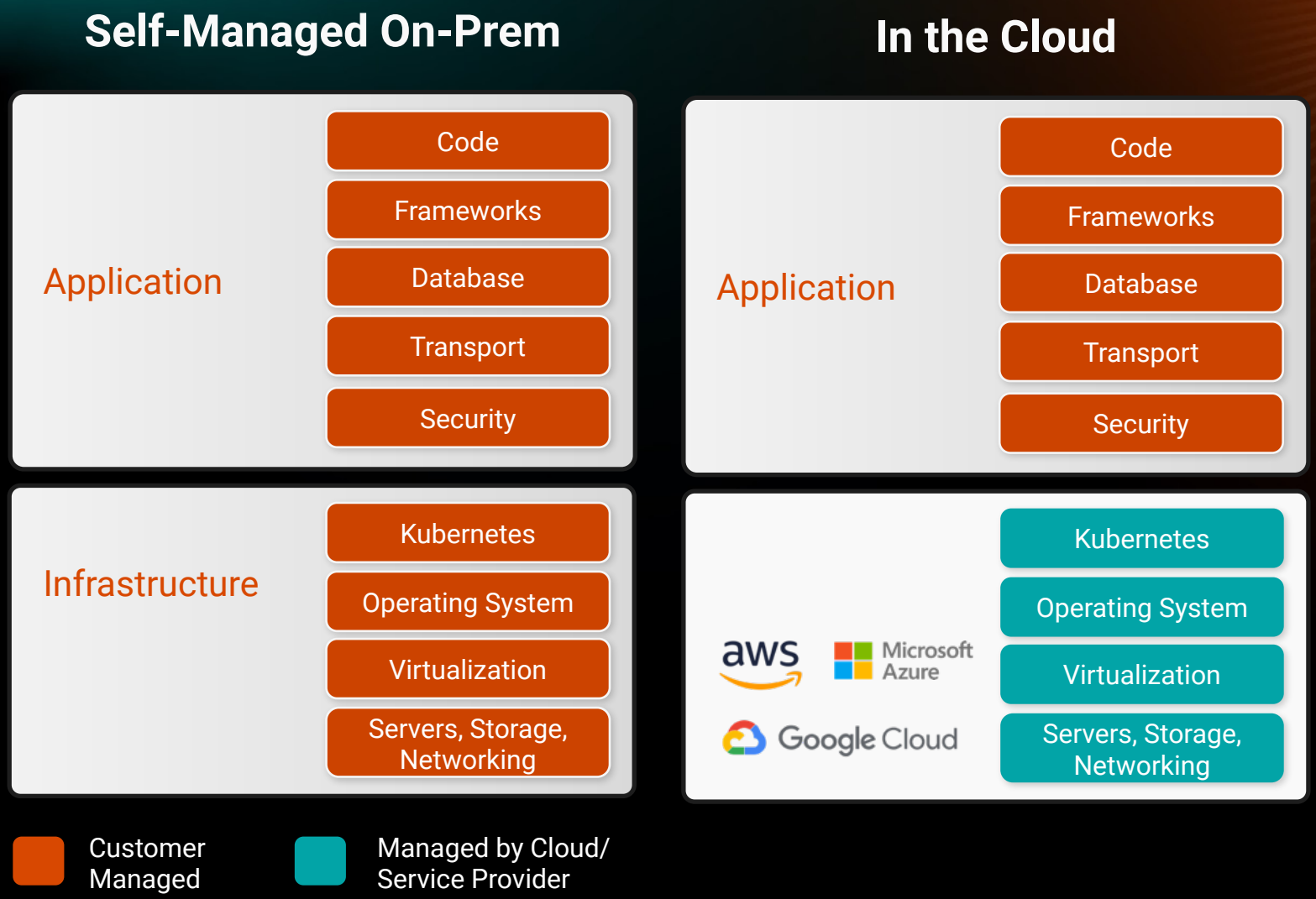
Managed by Cloud/ Service Provider

Infrastructure Inferred From Code

Self-Managed On-Prem



Infrastructure Inferred From Code



Infrastructure Inferred From Code

Self-Managed On-Prem

Application

- Code
- Frameworks
- Database
- Transport
- Security

Infrastructure

- Kubernetes
- Operating System
- Virtualization
- Servers, Storage, Networking

In the Cloud

Application

- Code
- Frameworks
- Database
- Transport
- Security

Infrastructure

aws · Microsoft Azure · Google Cloud

- Kubernetes
- Operating System
- Virtualization
- Servers, Storage, Networking

What We Need

Application

- Code
- Frameworks
- Database
- Transport
- Security

Infrastructure

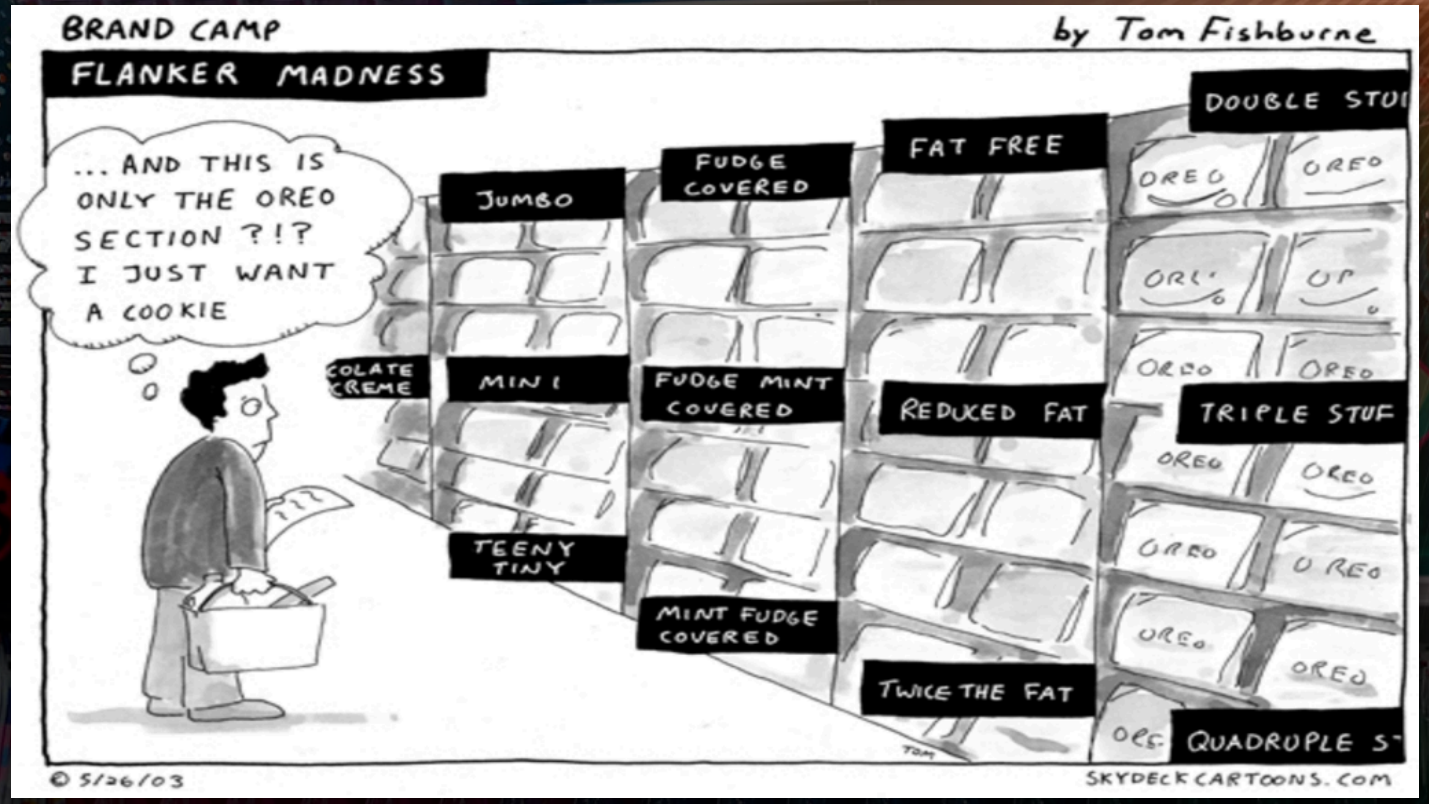
aws · Microsoft Azure · Google Cloud

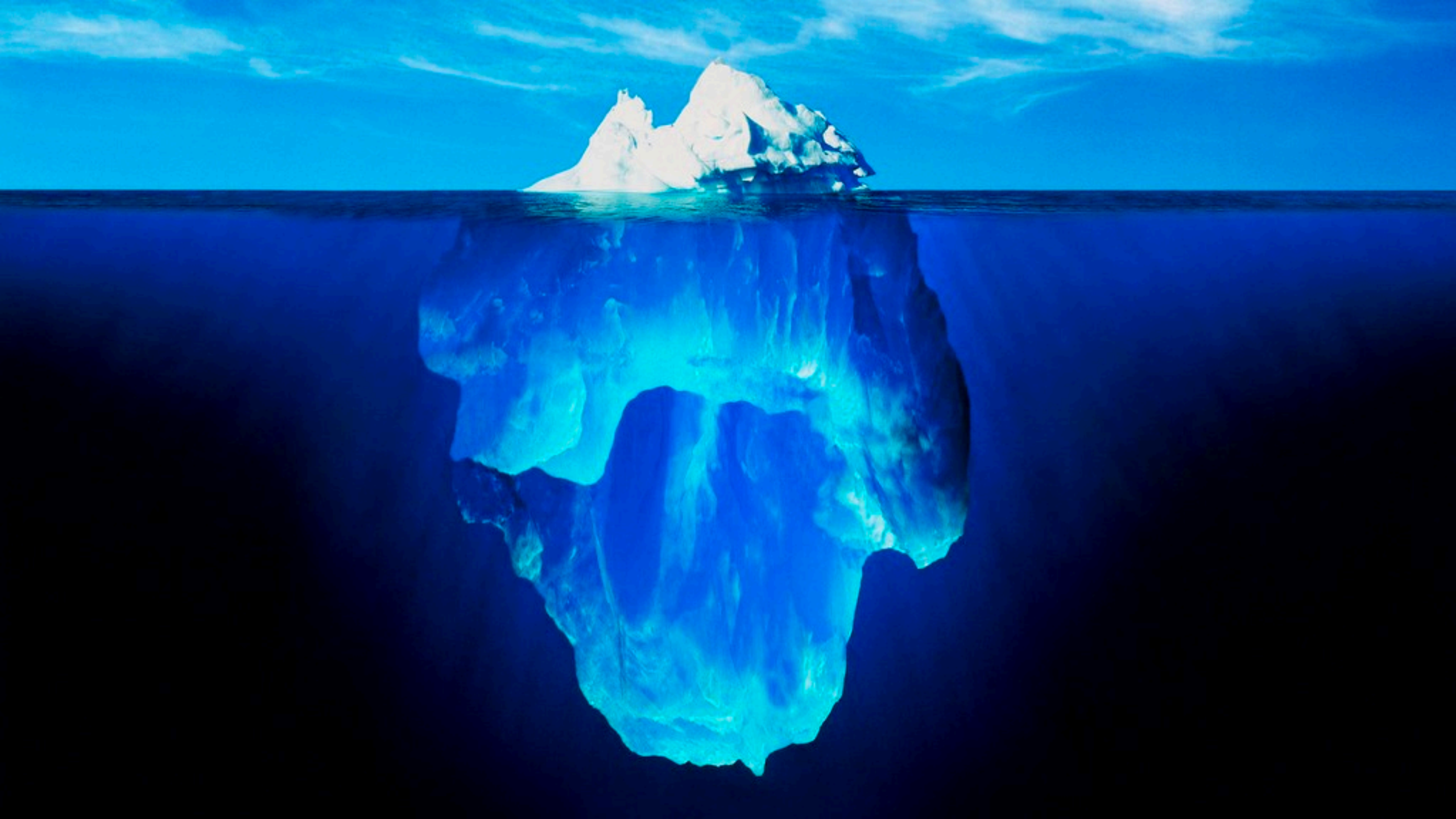
- Kubernetes
- Operating System
- Virtualization
- Servers, Storage, Networking

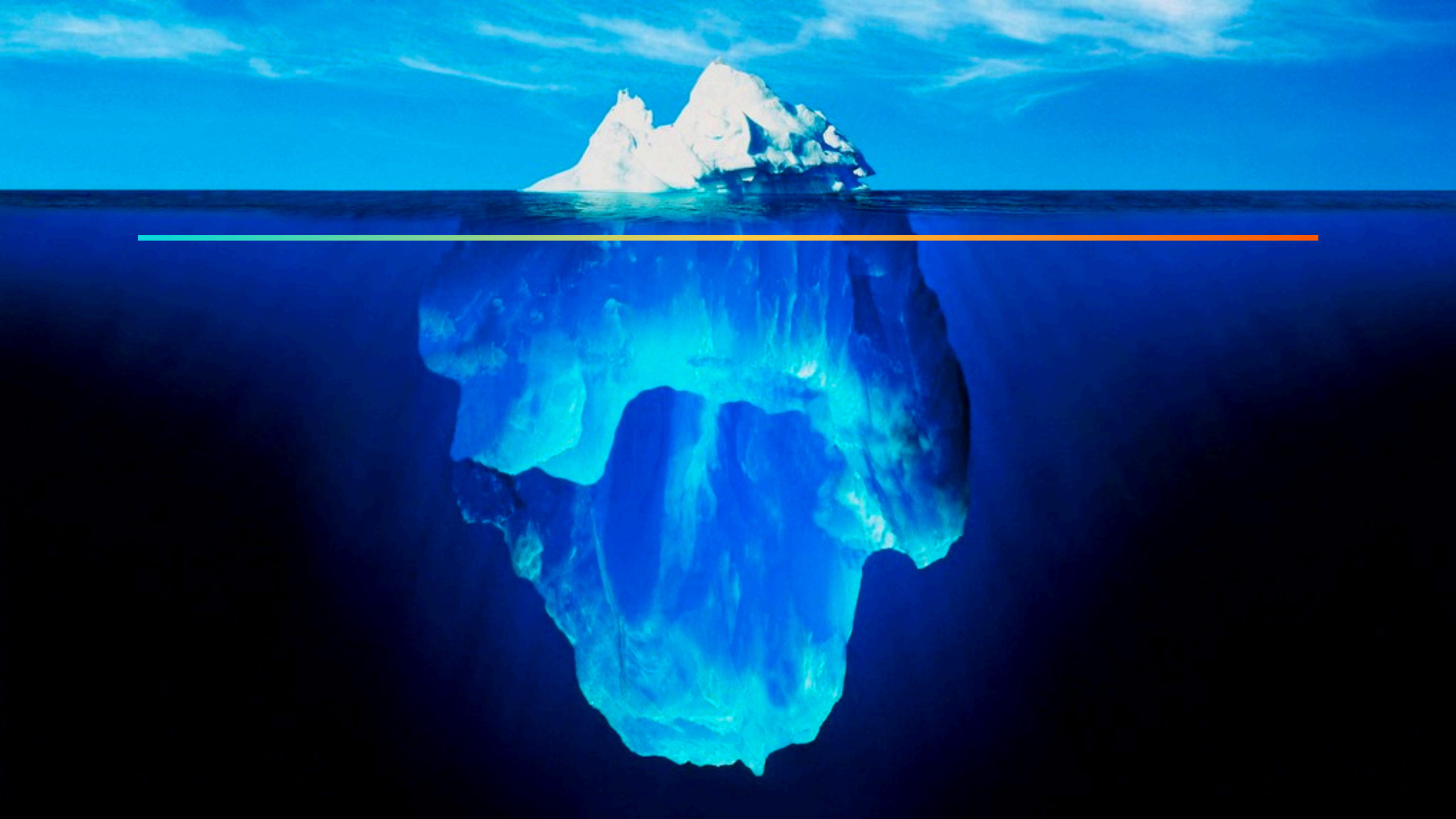
Customer Managed

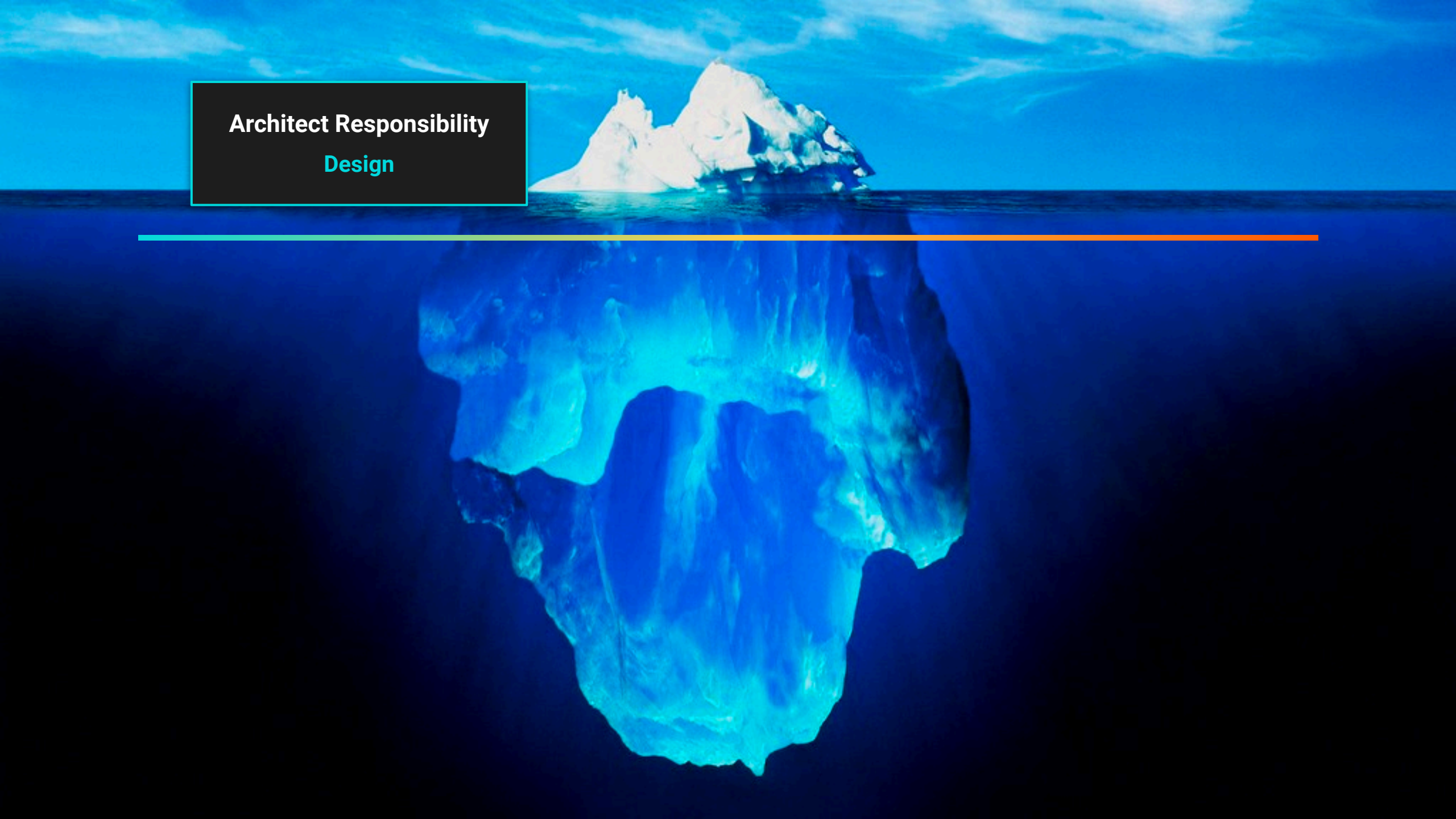
Managed by Cloud/ Service Provider

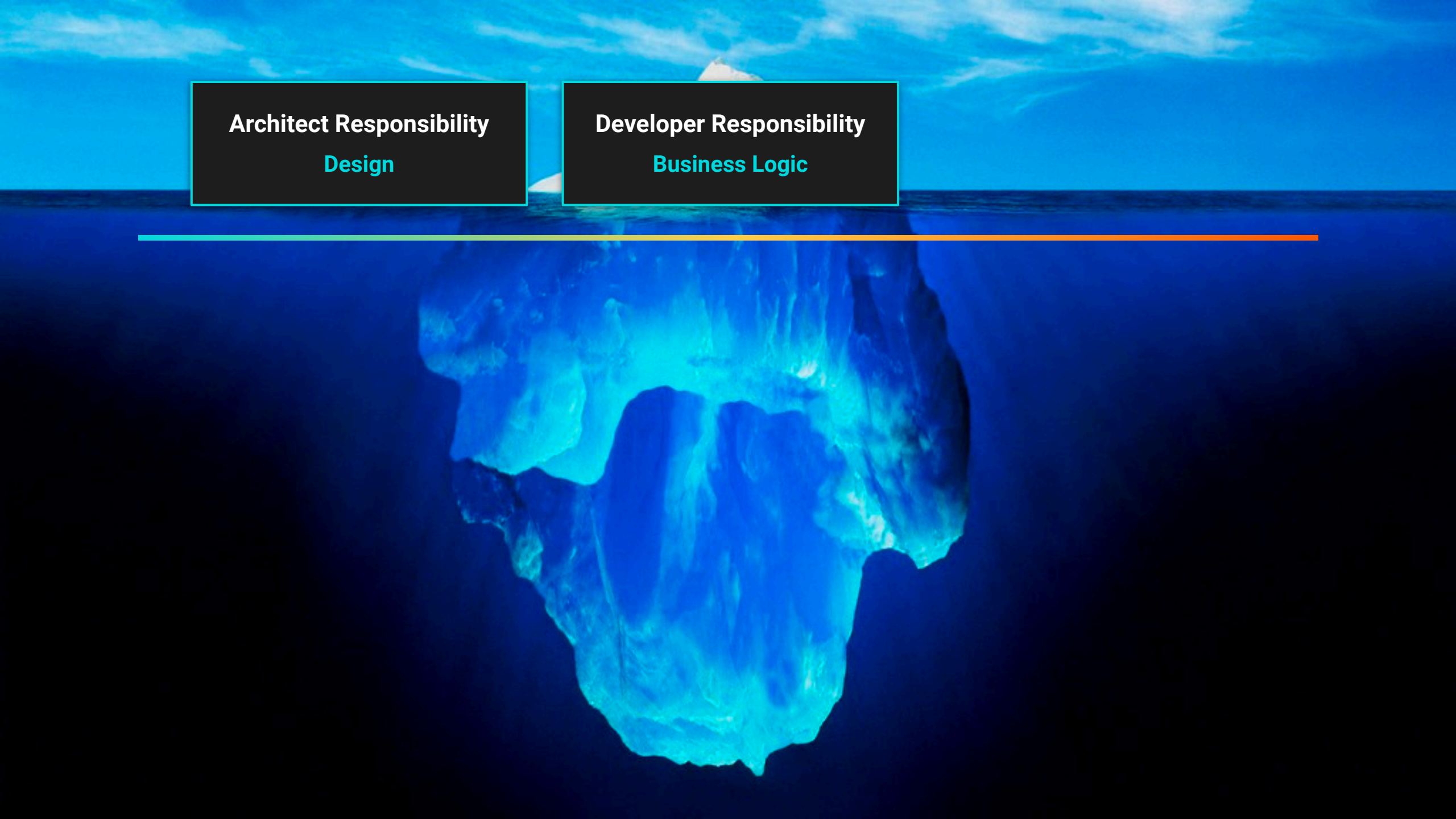
Too many options can be paralyzing

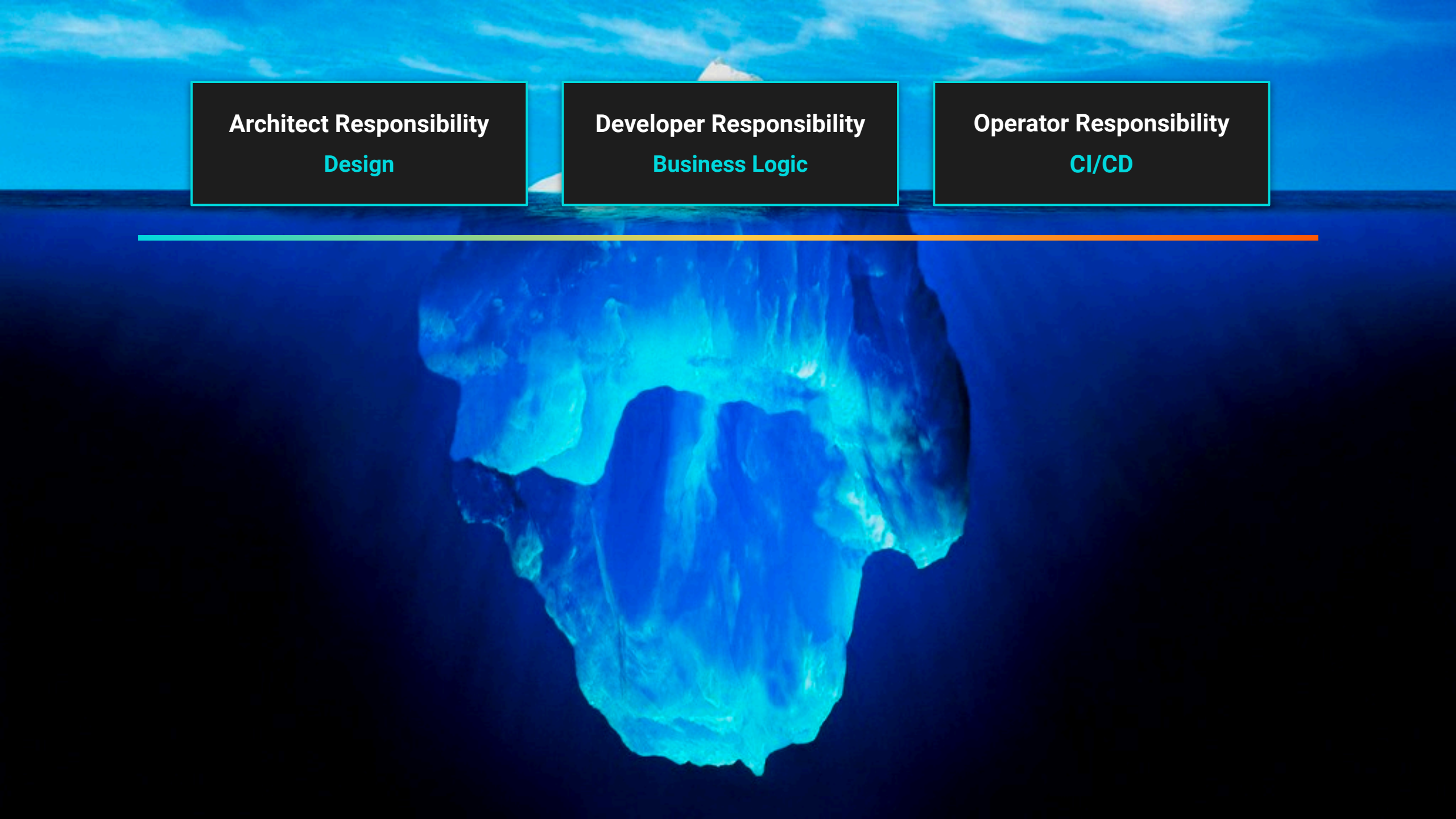


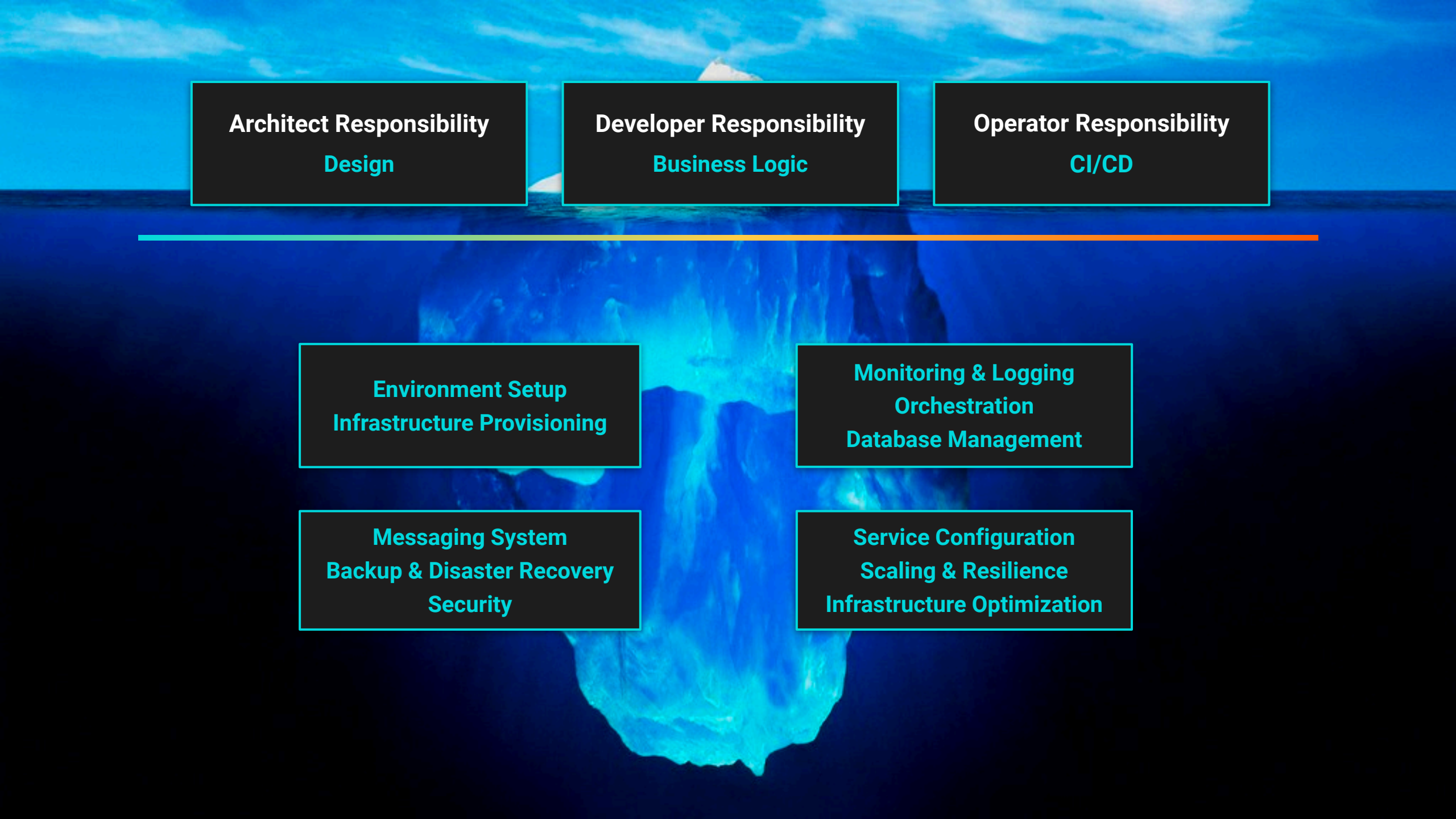


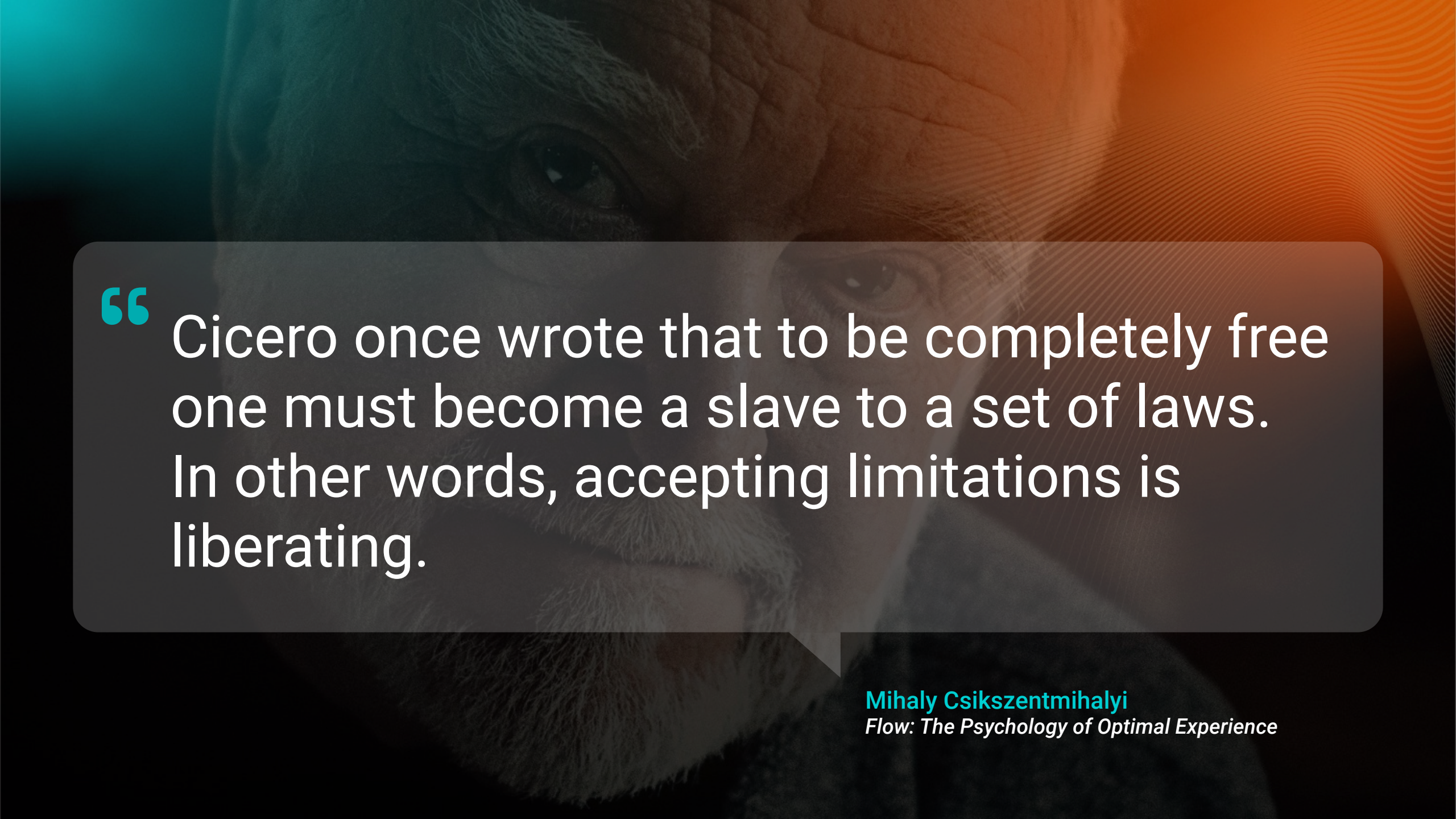


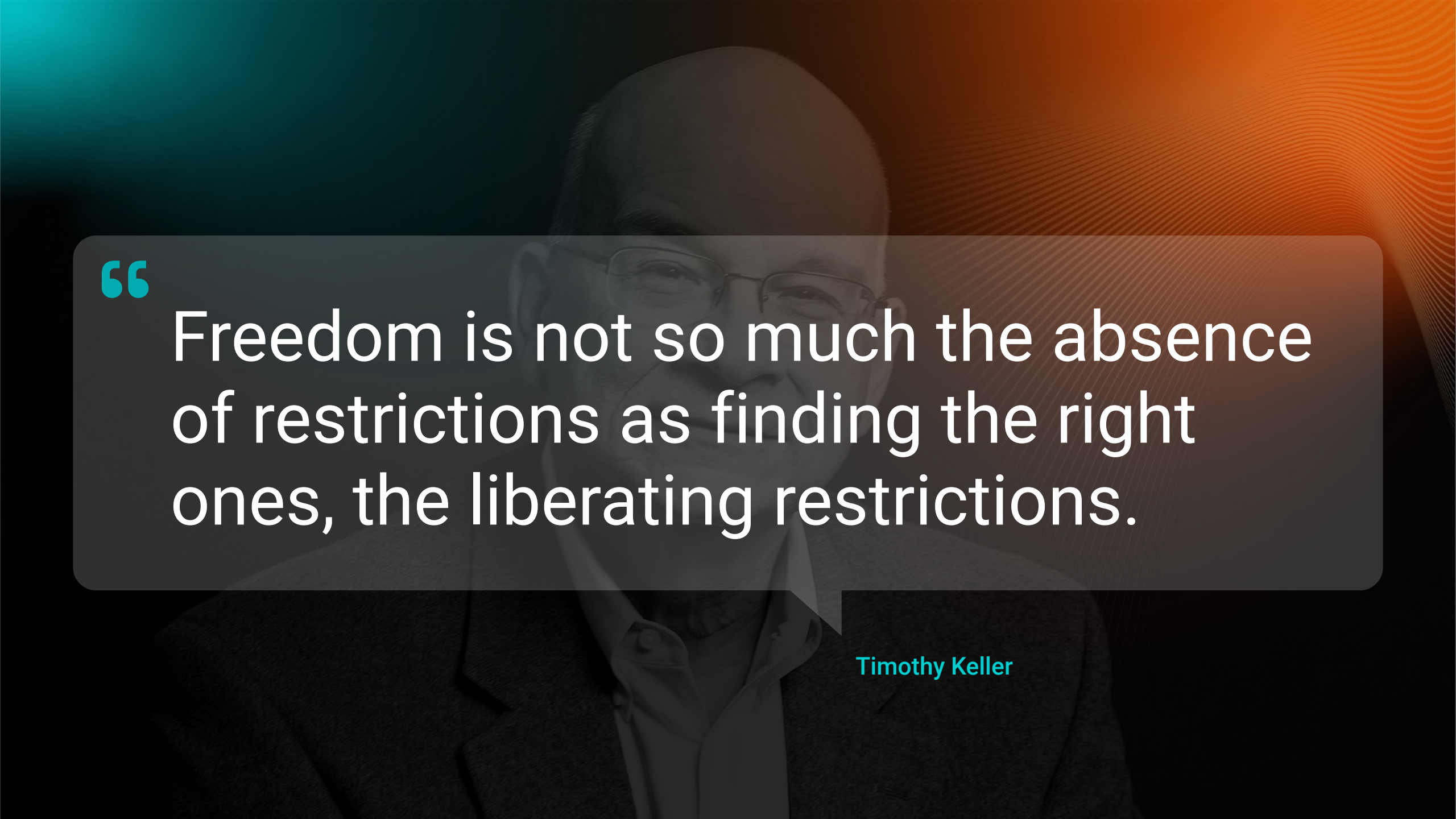

Architect Responsibility
Design


Architect Responsibility
Design
Developer Responsibility
Business Logic


Architect Responsibility
Design
Developer Responsibility
Business Logic
Operator Responsibility
CI/CD


Architect Responsibility
Design
Developer Responsibility
Business Logic
Operator Responsibility
CI/CD
Environment Setup
Infrastructure Provisioning
Monitoring & Logging
Orchestration
Database Management
Messaging System
Backup & Disaster Recovery
Security
Service Configuration
Scaling & Resilience
Infrastructure Optimization


“
Cicero once wrote that to be completely free one must become a slave to a set of laws. In other words, accepting limitations is liberating.
Mihaly Csikszentmihalyi
Flow: The Psychology of Optimal Experience


“
Freedom is not so much the absence of restrictions as finding the right ones, the liberating restrictions.
Timothy Keller


Constraints Liberate
Liberties Constrain
Watch this great talk by Runar Bjarnason: https://youtu.be/GqmsQeSzMdw

Distilling the Ultimate

Cloud to Edge Programming Model

Distilling the Ultimate

Cloud to Edge Programming Model

3 things we as developers can **never delegate**:

Distilling the Ultimate

Cloud to Edge Programming Model

3 things we as developers can **never delegate**:

1. **Data model**

Distilling the Ultimate

Cloud to Edge Programming Model

3 things we as developers can **never delegate**:

1. **Data model**
   - How to model the business data

Distilling the Ultimate

Cloud to Edge Programming Model

3 things we as developers can **never delegate**:

1. **Data model**
   - How to model the business data
   - Its structure, constraints, guarantees, and query model

Distilling the Ultimate

Cloud to Edge Programming Model

3 things we as developers can **never delegate**:

1. **Data model**
   - How to model the business data
   - Its structure, constraints, guarantees, and query model
2. **API**

Distilling the Ultimate

Cloud to Edge Programming Model

3 things we as developers can **never delegate**:

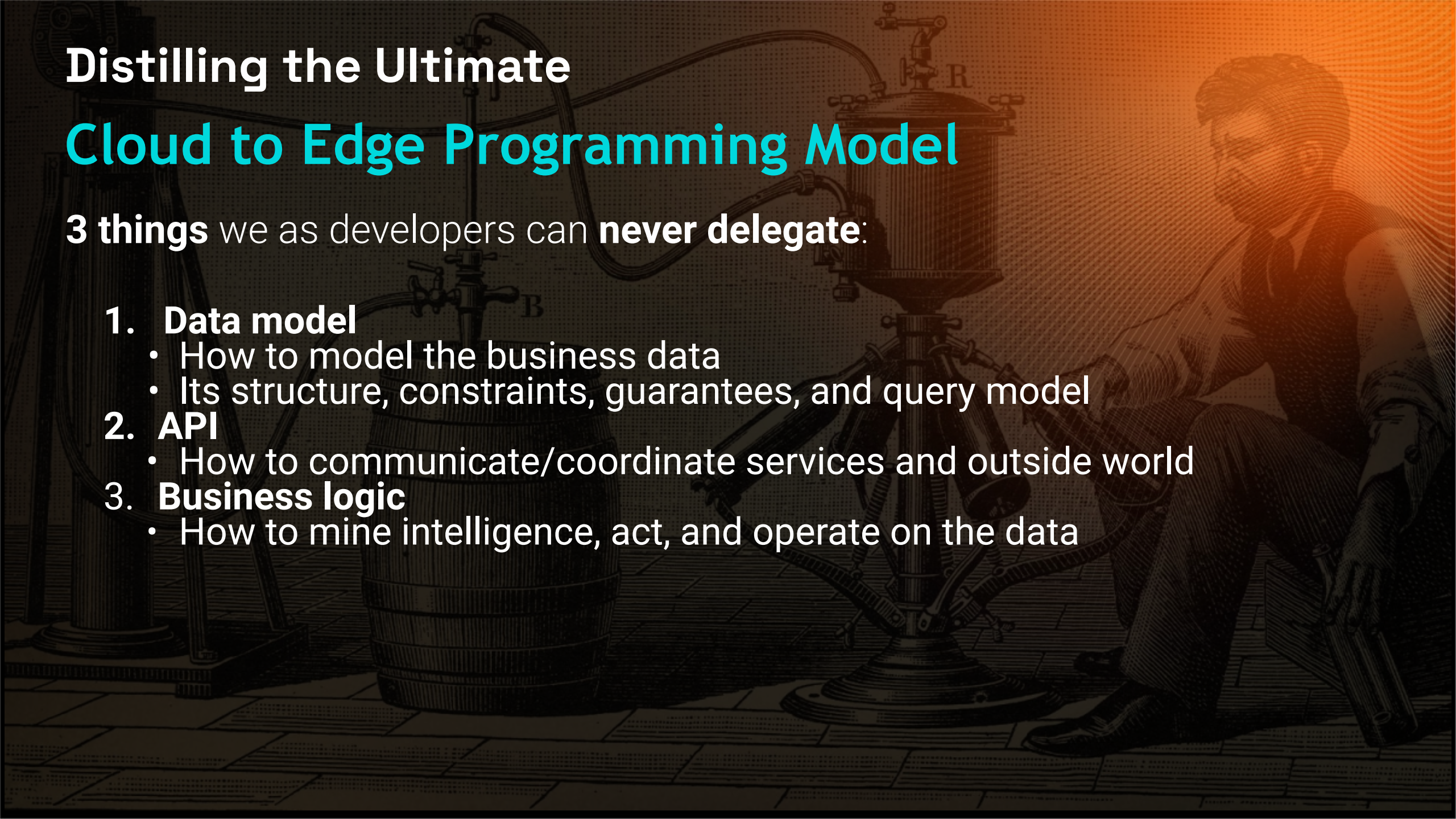
1. **Data model**
   - How to model the business data
   - Its structure, constraints, guarantees, and query model
2. **API**
   - How to communicate/coordinate services and outside world

Distilling the Ultimate

Cloud to Edge Programming Model

3 things we as developers can **never delegate**:

1. **Data model**
   - How to model the business data
   - Its structure, constraints, guarantees, and query model
2. **API**
   - How to communicate/coordinate services and outside world
3. **Business logic**

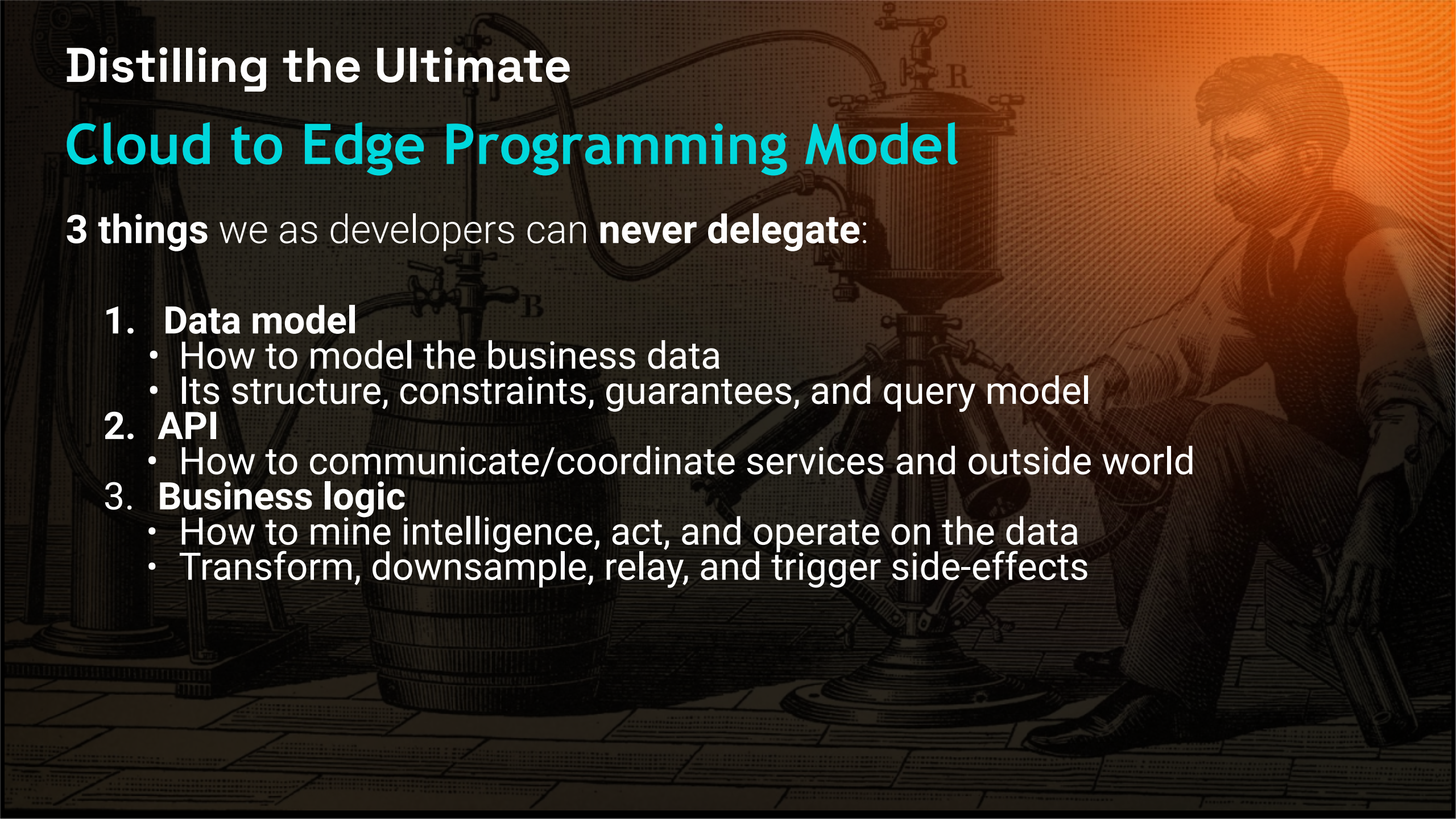


Distilling the Ultimate

Cloud to Edge Programming Model

3 things we as developers can **never delegate**:

1. **Data model**
   - How to model the business data
   - Its structure, constraints, guarantees, and query model
2. **API**
   - How to communicate/coordinate services and outside world
3. **Business logic**
   - How to mine intelligence, act, and operate on the data

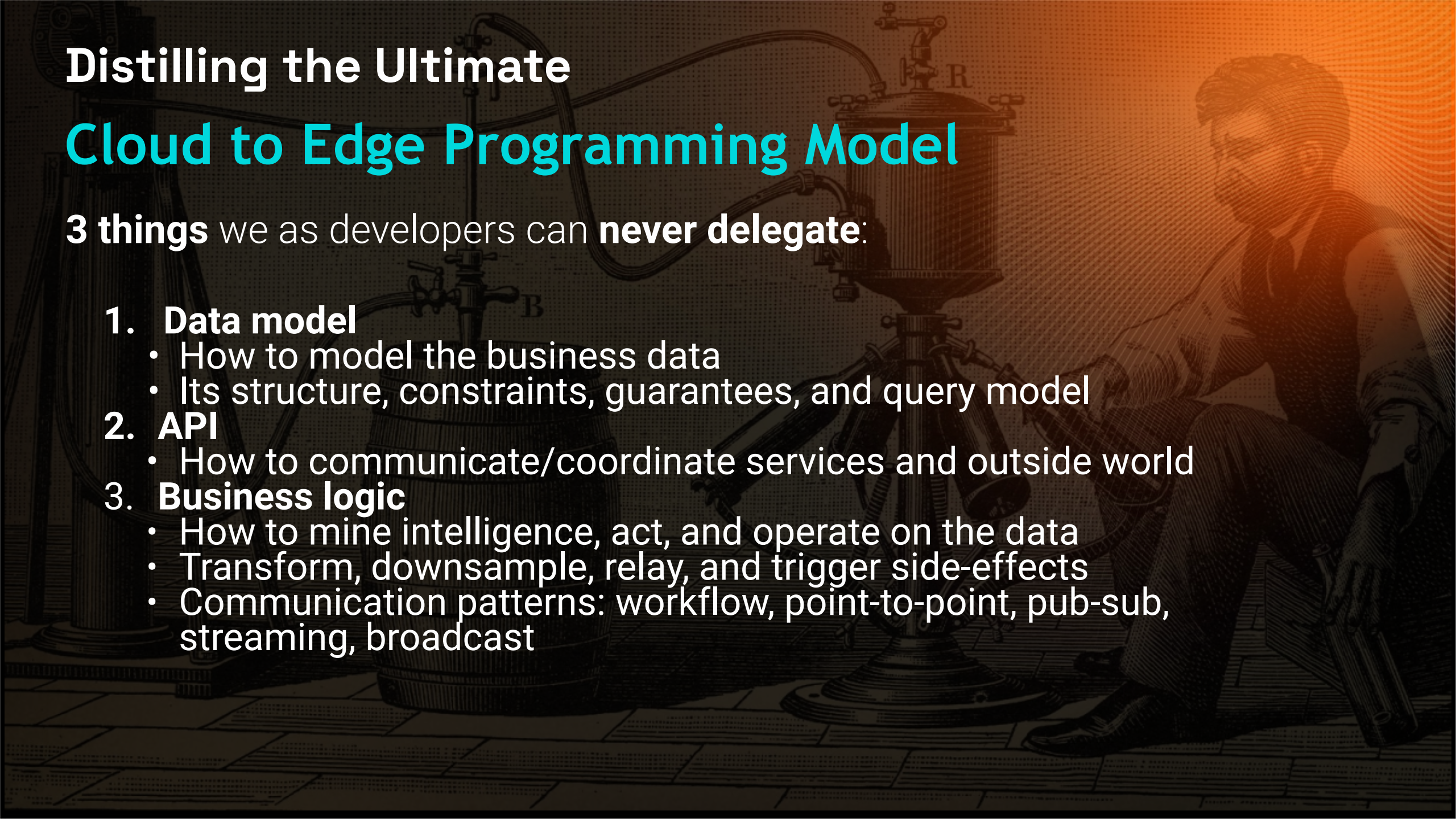


Distilling the Ultimate

Cloud to Edge Programming Model

3 things we as developers can **never delegate**:

1. **Data model**
   - How to model the business data
   - Its structure, constraints, guarantees, and query model
2. **API**
   - How to communicate/coordinate services and outside world
3. **Business logic**
   - How to mine intelligence, act, and operate on the data
   - Transform, downsample, relay, and trigger side-effects



Distilling the Ultimate

Cloud to Edge Programming Model

3 things we as developers can **never delegate**:

1. **Data model**
   - How to model the business data
   - Its structure, constraints, guarantees, and query model
2. **API**
   - How to communicate/coordinate services and outside world
3. **Business logic**
   - How to mine intelligence, act, and operate on the data
   - Transform, downsample, relay, and trigger side-effects
   - Communication patterns: workflow, point-to-point, pub-sub, streaming, broadcast

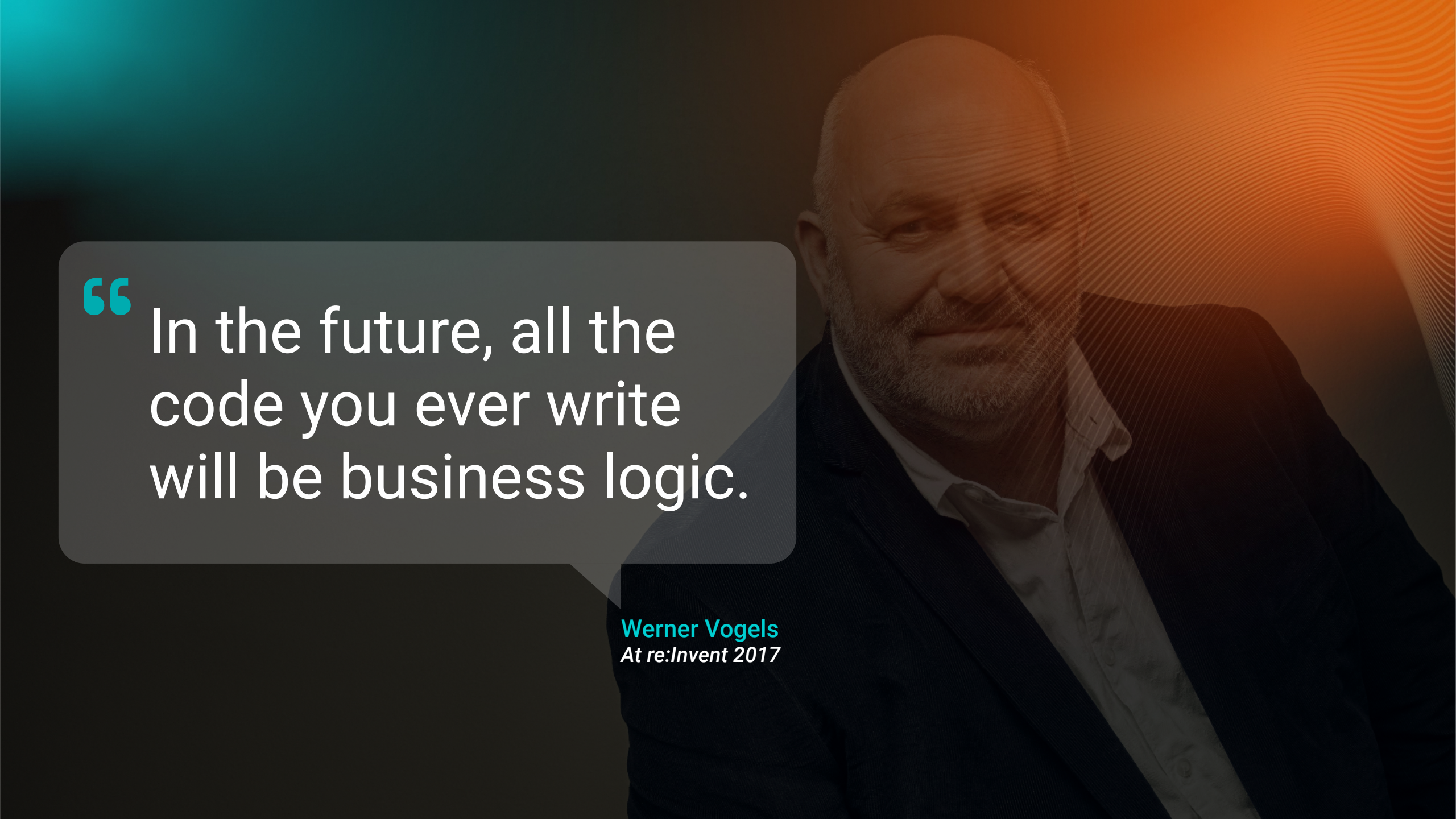
Distilling the Ultimate

Cloud to Edge Programming Model

3 things we as developers can **never delegate**:

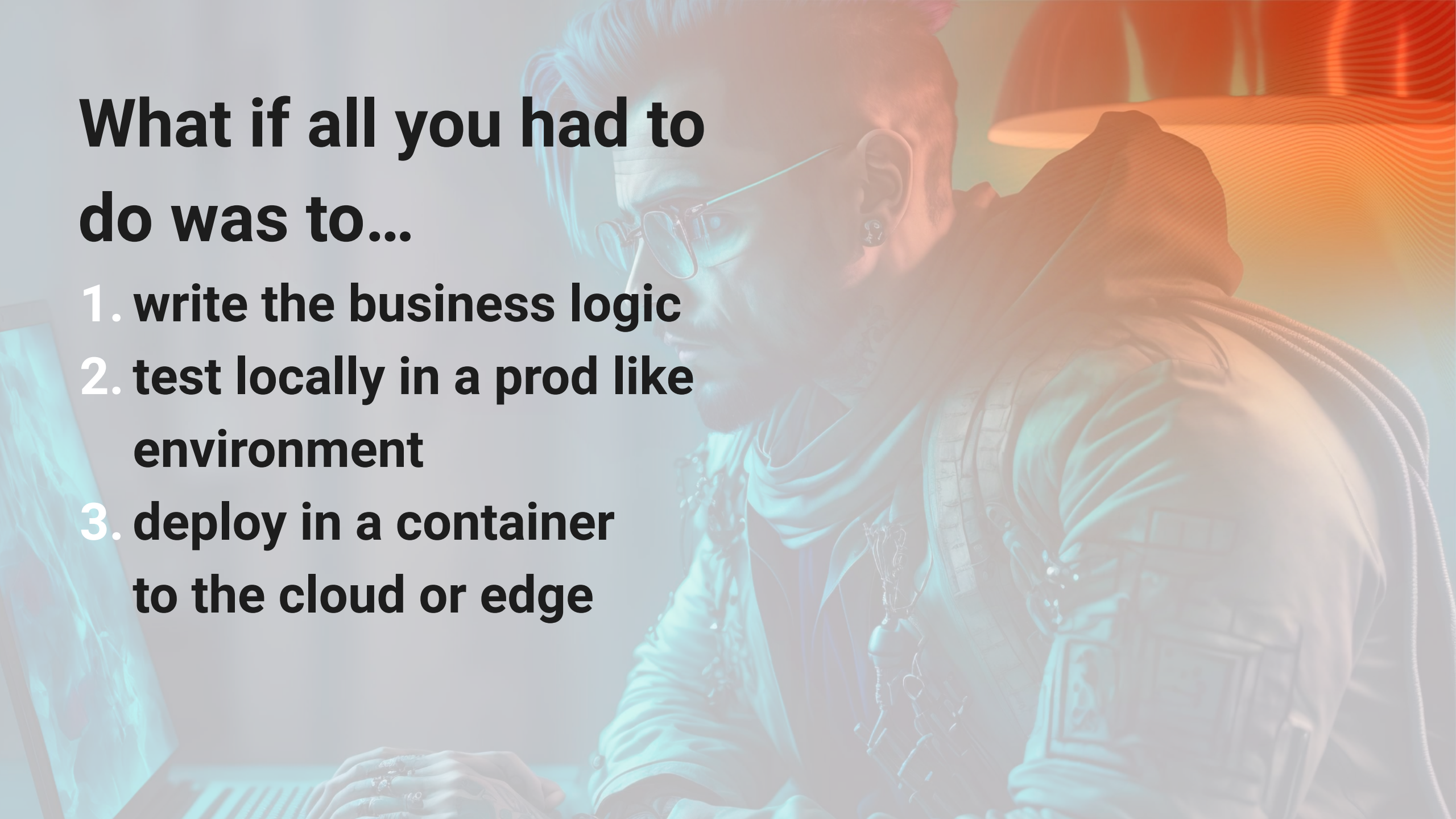
1. **Data model**
   - How to model the business data
   - Its structure, constraints, guarantees, and query model
2. **API**
   - How to communicate/coordinate services and outside world
3. **Business logic**
   - How to mine intelligence, act, and operate on the data
   - Transform, downsample, relay, and trigger side-effects
   - Communication patterns: workflow, point-to-point, pub-sub, streaming, broadcast

The rest can (and should) be fully managed and automated by the underlying platform


“
In the future, all the code you ever write will be business logic.
Werner Vogels
At re:Invent 2017

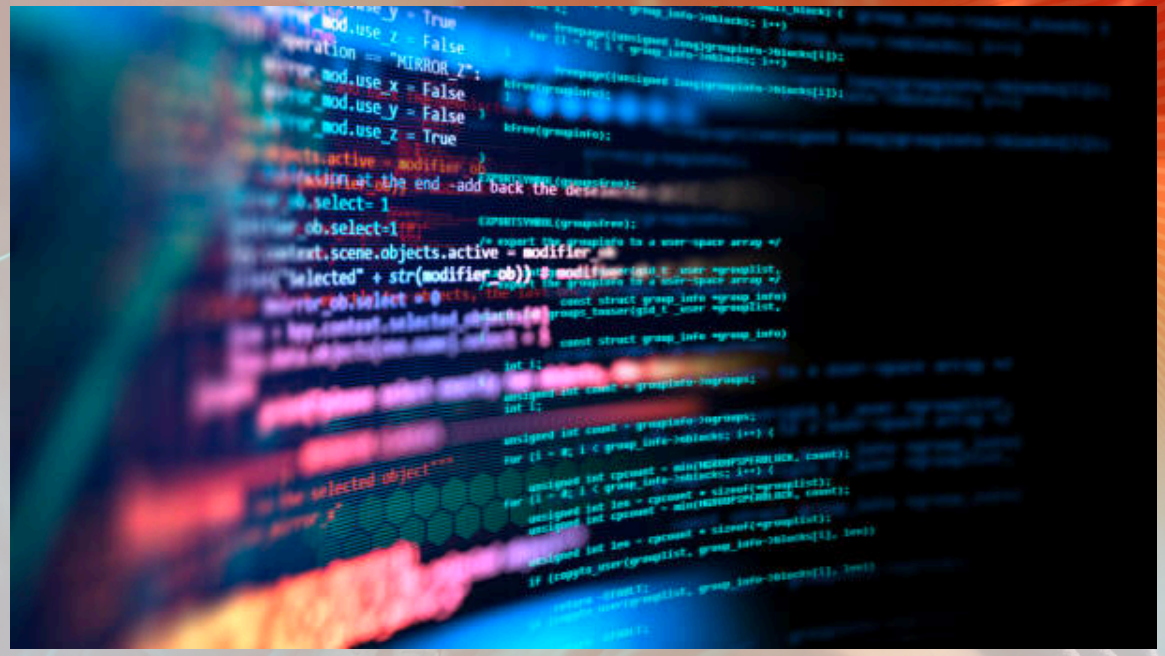
What if all you had to do was to…

1. write the business logic
2. test locally in a prod like environment
3. deploy in a container to the cloud or edge



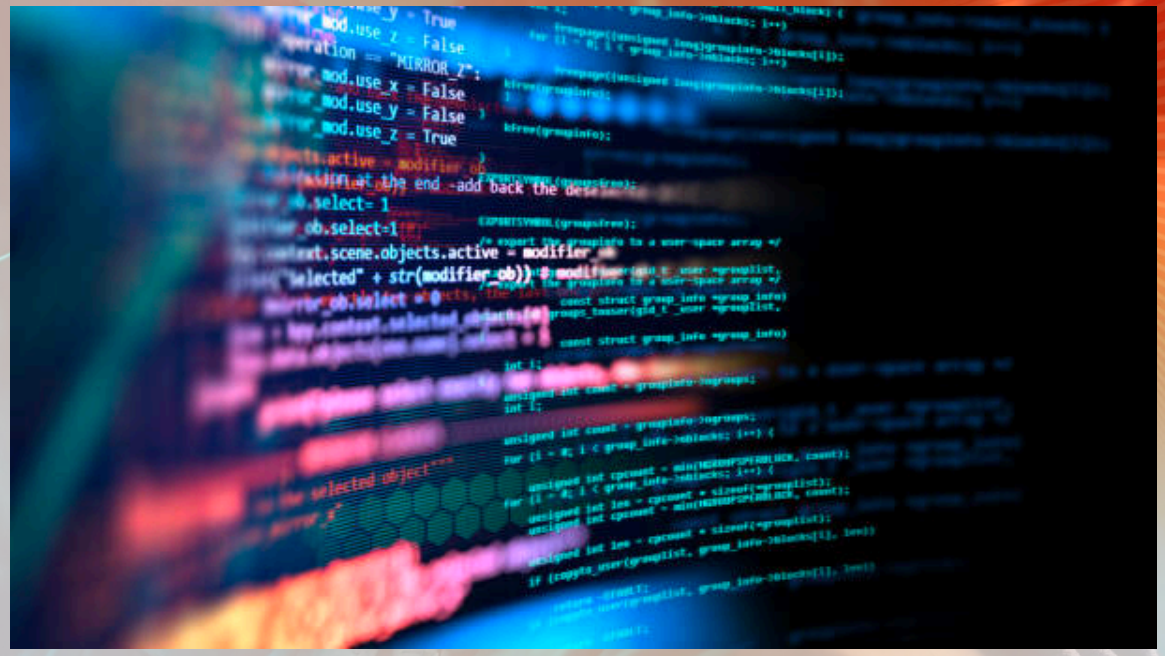
What if all you had to do was to…

1. write the business logic
2. test locally in a prod like environment
3. deploy in a container to the cloud or edge



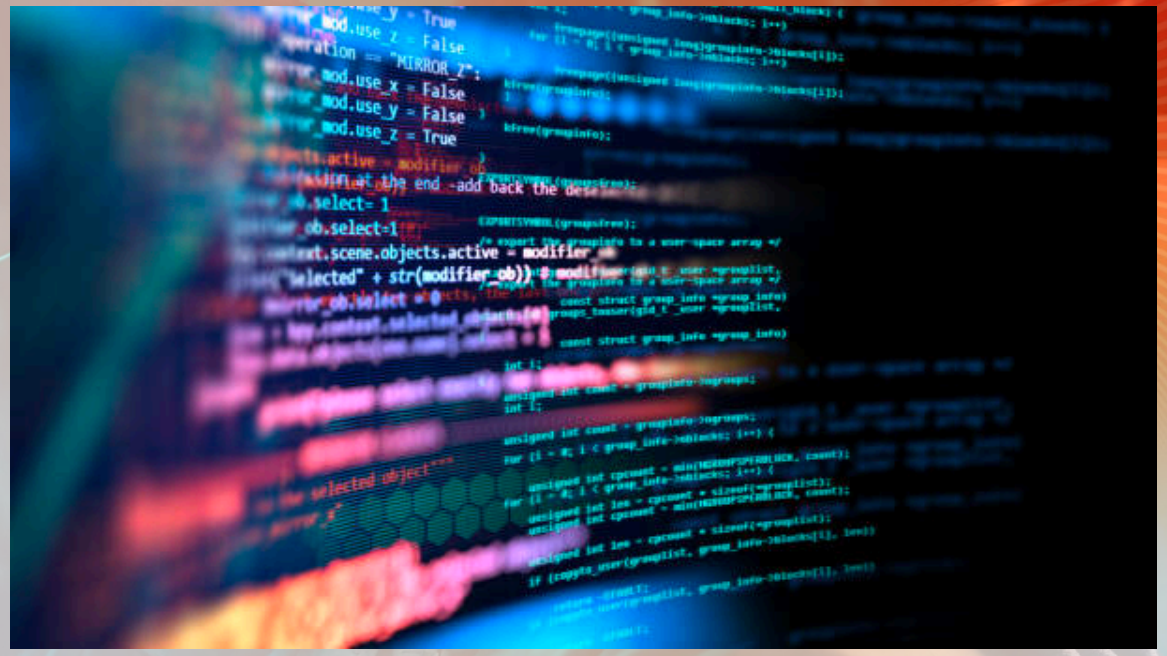
What if all you had to do was to…

1. write the business logic
2. test locally in a prod like environment
3. deploy in a container to the cloud or edge



What if all you had to do was to…

1. write the business logic
2. test locally in a prod like environment
3. deploy in a container to the cloud or edge



The future is here

KALIX

KALIX

1. **Developer PaaS**—for building **event-driven backend services and APIs**

KALIX

1. **Developer PaaS**—for building **event-driven backend services and APIs**
2. **Guard-railed hyper-productive DX**—using a simple **declarative programming model**

KALIX

1. **Developer PaaS**—for building **event-driven backend services and APIs**
2. **Guard-railed hyper-productive DX**—using a simple **declarative programming model**
3. **Infrastructure inferred from code**—**Akka**, **Kubernetes**, Databases, Message Brokers, etc.

KALIX

1. **Developer PaaS**—for building **event-driven backend services and APIs**
2. **Guard-railed hyper-productive DX**—using a simple **declarative programming model**
3. **Infrastructure inferred from code**—**Akka**, **Kubernetes**, Databases, Message Brokers, etc.
4. **Fully managed**—NoOps, Serverless Cloud Native stack

KALIX

1. **Developer PaaS**—for building **event-driven backend services and APIs**
2. **Guard-railed hyper-productive DX**—using a simple **declarative programming model**
3. **Infrastructure inferred from code**—**Akka**, **Kubernetes**, Databases, Message Brokers, etc.
4. **Fully managed**—NoOps, Serverless Cloud Native stack
5. **Polyglot**—choose your language: Java, Javascript, Typescript, Scala, etc.

KALIX

1. **Developer PaaS**—for building **event-driven backend services and APIs**
2. **Guard-railed hyper-productive DX**—using a simple **declarative programming model**
3. **Infrastructure inferred from code**—**Akka**, **Kubernetes**, Databases, Message Brokers, etc.
4. **Fully managed**—NoOps, Serverless Cloud Native stack
5. **Polyglot**—choose your language: Java, Javascript, Typescript, Scala, etc.
6. **Reactive at its core**—low latency, high throughput, always available, adaptive scaling

Developer's Responsibility

Kalix's Responsibility

Developer's Responsibility

Kalix's Responsibility

Entity (Stateful Service)

API endpoint with:

- Domain Data
- Business Logic
- HTTP definition

```java
@EntityType("customer")
public class CustomerEntity extends ValueEntity<CustomerEntity.Customer> {

  record Customer(String name, String email) {}
  record CreateCustomer(String name, String email) {}

  @EntityKey("customer_id")
  @PostMapping("/customers/{customer_id}")
  public Effect<String> createCustomer(@RequestBody CreateCustomer create) {
    return effects()
      .updateState(new Customer(create.name, create.email))
      .thenReply("done");
  }
}
```

Developer's Responsibility

Kalix's Responsibility

Entity (Stateful Service)

API endpoint with:

- Domain Data
- Business Logic
- HTTP definition

```
@EntityType("customer")
public class CustomerEntity extends ValueEntity<CustomerEntity.Customer> {

  record Customer(String name, String email) {}
  record CreateCustomer(String name, String email) {}

  @EntityKey("customer_id")
  @PostMapping("/customers/{customer_id}")
  public Effect<String> createCustomer(@RequestBody CreateCustomer create) {
    return effects()
      .updateState(new Customer(create.name, create.email))
      .thenReply("done");
  }
}
```

View (Materialized Query)

API endpoint:

- Querying domain data
- Return streamed data set
- HTTP definition

```
@Table("customers")
@Subscribe.ValueEntity(CustomerEntity.class)
public class CustomerByNameView extends View<CustomerEntity.Customer>

  @GetMapping("/customers/by-name/{name}")
  @Query("SELECT * FROM customers WHERE name = :name")
  public Flux<CustomerEntity.Customer> findCustomers(String name) {
      return null;
  }
}
```

Developer's Responsibility

Kalix's Responsibility

Entity (Stateful Service)

API endpoint with:

- Domain Data
- Business Logic
- HTTP definition

```
@EntityType("customer")
public class CustomerEntity extends ValueEntity<CustomerEntity.Customer> {

  record Customer(String name, String email) {}
  record CreateCustomer(String name, String email) {}

  @EntityKey("customer_id")
  @PostMapping("/customers/{customer_id}")
  public Effect<String> createCustomer(@RequestBody CreateCustomer create) {
    return effects()
      .updateState(new Customer(create.name, create.email))
      .thenReply("done");
  }
}
```

View (Materialized Query)

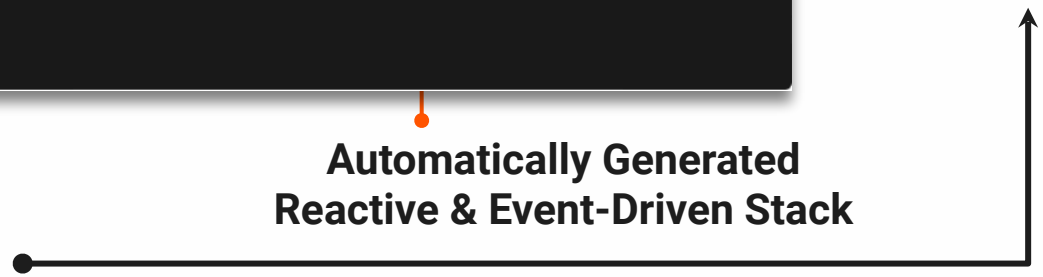
API endpoint:

- Querying domain data
- Return streamed data set
- HTTP definition

```
@Table("customers")
@Subscribe.ValueEntity(CustomerEntity.class)
public class CustomerByNameView extends View<CustomerEntity.Customer>

  @GetMapping("/customers/by-name/{name}")
  @Query("SELECT * FROM customers WHERE name = :name")
  public Flux<CustomerEntity.Customer> findCustomers(String name) {
      return null;
  }
}
```

Automatically Generated
Reactive & Event-Driven Stack



Developer's Responsibility

Kalix's Responsibility

Entity (Stateful Service)

API endpoint with:

- Domain Data
- Business Logic
- HTTP definition

```
@EntityType("customer")
public class CustomerEntity extends ValueEntity<CustomerEntity.Customer> {

  record Customer(String name, String email) {}
  record CreateCustomer(String name,

  @EntityKey("customer_id")
  @PostMapping("/customers/{customer_
  public Effect<String> createCustome
    return effects()
      .updateState(new Customer(creat
      .thenReply("done");
  }
}
```

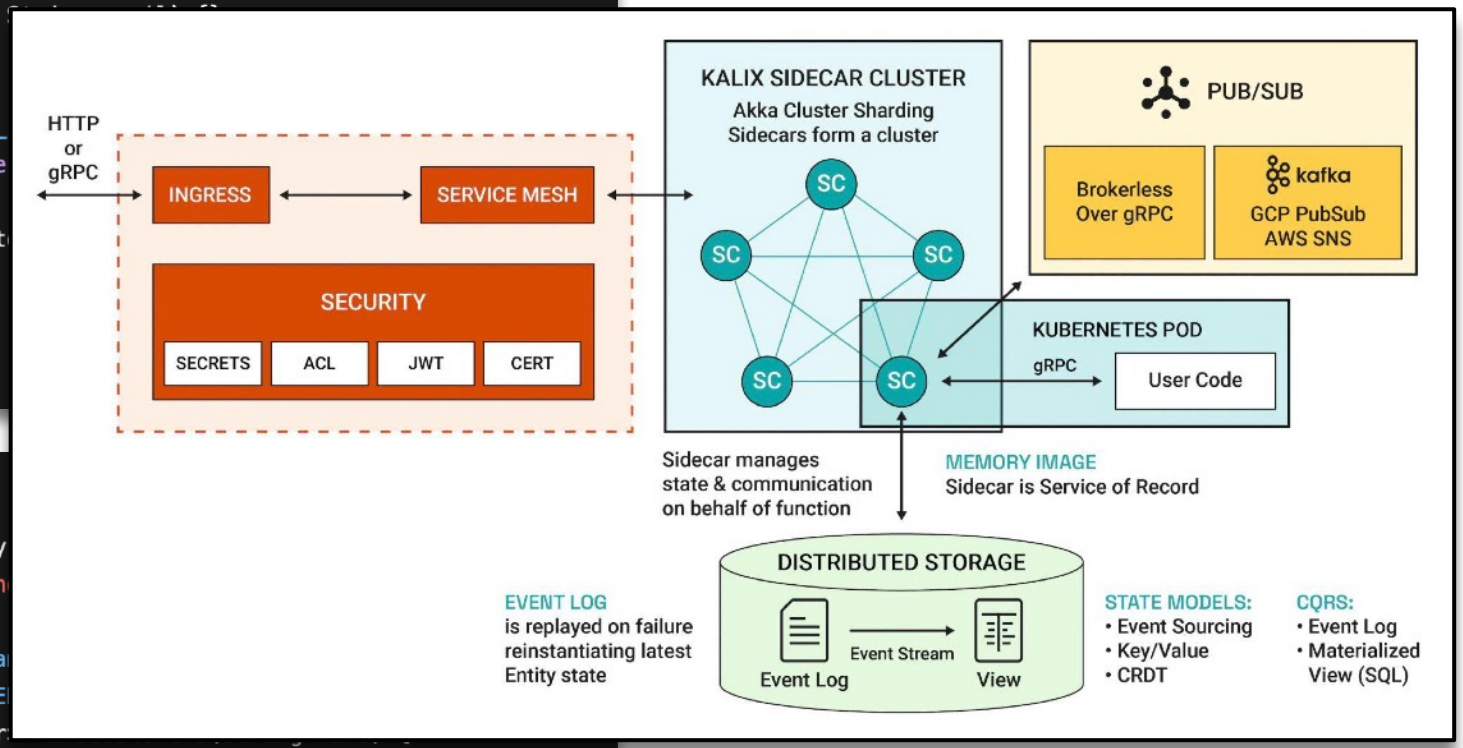
View (Materialized Query)

API endpoint:

- Querying domain data
- Return streamed data set
- HTTP definition

```
@Table("customers")
@Subscribe.ValueEntity(CustomerEntity
public class CustomerByNameView exten

  @GetMapping("/customers/by-name/{na
  @Query("SELECT * FROM customers WHE
  public Flux<CustomerEntity.Customer
      return null;
  }
}
```



Automatically Generated
Reactive & Event-Driven Stack

To sum things up...

Cloud Native is too **complex**

- **Serverless** is promising, but currently **falls short**
- We need to continue to **climb the ladder of abstractions**
- We need to **liberate developers** to focus on the essence: business logic
- We need **infrastructure inferred from code,** not an integration project

Edge Computing is **already here**

- Opens up a **new world of possibilities** and challenges
- We need to see **Cloud and Edge as a continuum**
- Apps soon need to be both **Cloud Native** and **Edge Native**
- We need to a new **Programming Model and DX** designed to serve this Cloud to Edge continuum

We are **working on it**

- **Kalix**: Polyglot, real-time, event-driven, reactive, zero-ops **developer PaaS**
- **Akka Edge**: Adaptive, resilient, and scalable **Cloud-to-Edge Data Plane**
- **Deploy to Edge today**, **or** build Cloud apps that are **Edge Future Proof**

goto;

Don't forget to
rate this session
in the **GOTO Guide app**

Thanks

Questions?

Learn more:

kalix.io

akka.io

goto;

GOTO
Copenhagen 2023

#GOTOcph