AWS re:Invent

NOV. 28 – DEC. 2, 2022 | LAS VEGAS, NV

ARC305

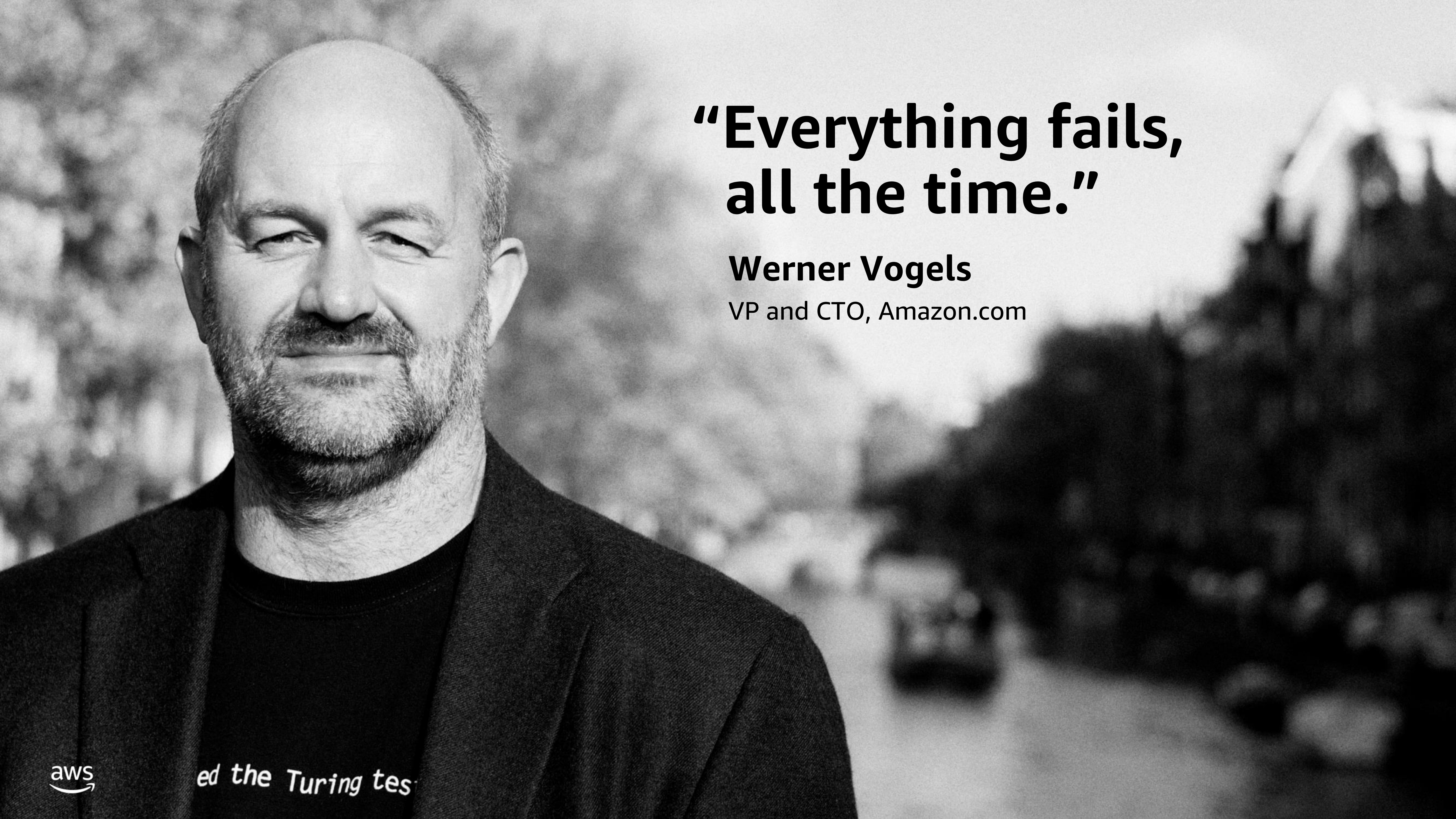
# Reducing your area of impact and surviving difficult days

Bruno Emer

Principal Solutions Architect
Critical Capabilities
AWS

Byron Arnao

Principal Solutions Architect
Critical Capabilities
AWS



"Everything fails, all the time."

**Werner Vogels**

VP and CTO, Amazon.com

# Categories of failure

Code deployments and configuration

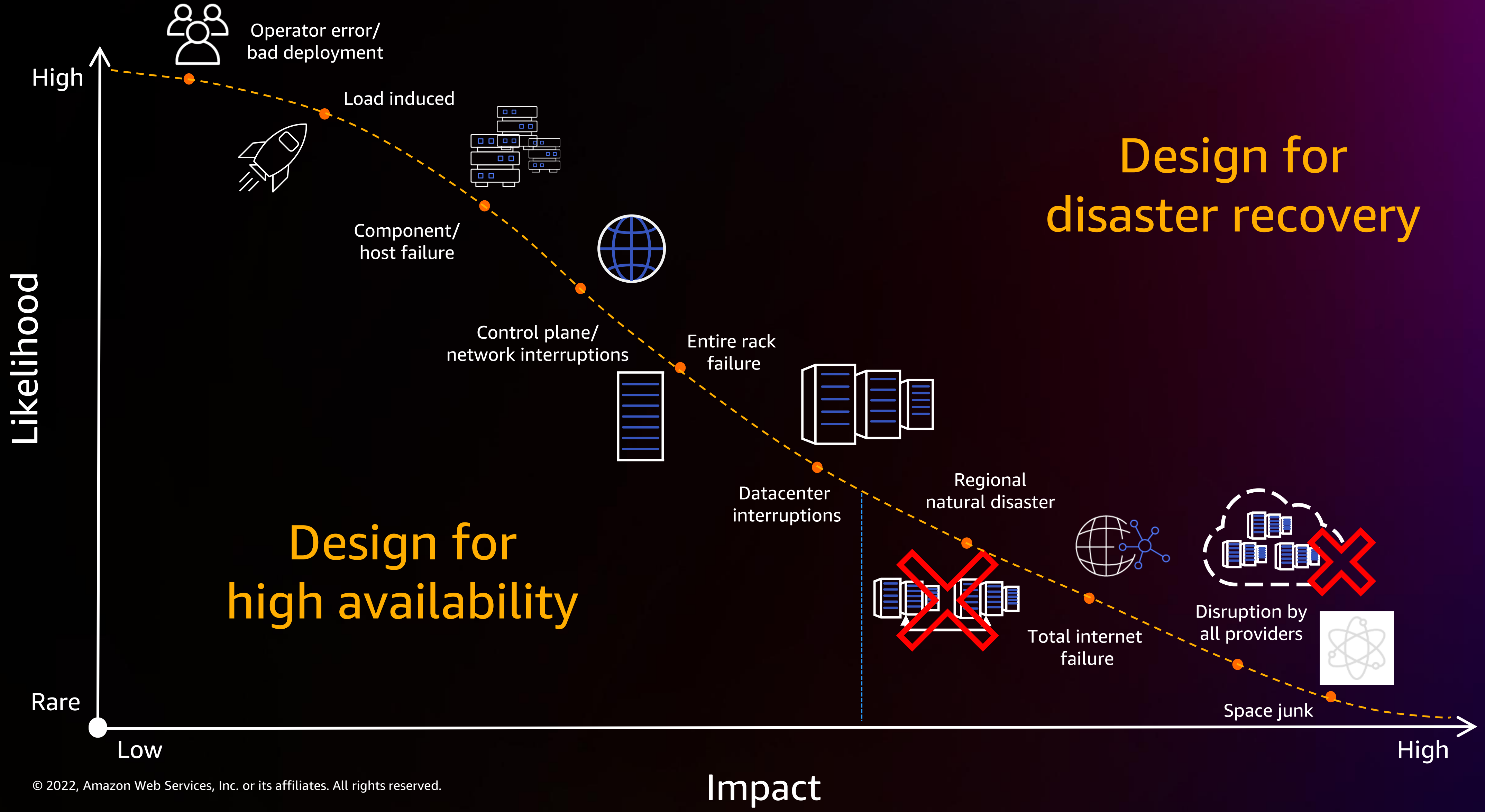
Data and state

Dependencies

Core infrastructure

Highly unlikely scenarios

# Various causes of failure



# How do we usually protect against failures?

| Failure mode | | Means of protection | |
|---|---|---|---|
| Rack down | ← | Autoscaling groups | ✓ |
| AZ impairment | ← | Multi-AZ deployments | ✓ |
| Regional impairment | ← | Region failover | ✓ |
| Bad deployment | ← | Canary deployments | ! |
| Human failure | ← | Processes, runbooks | ! |
| Poison pill | ← | Extensive testing | ! |

Typically only for compute

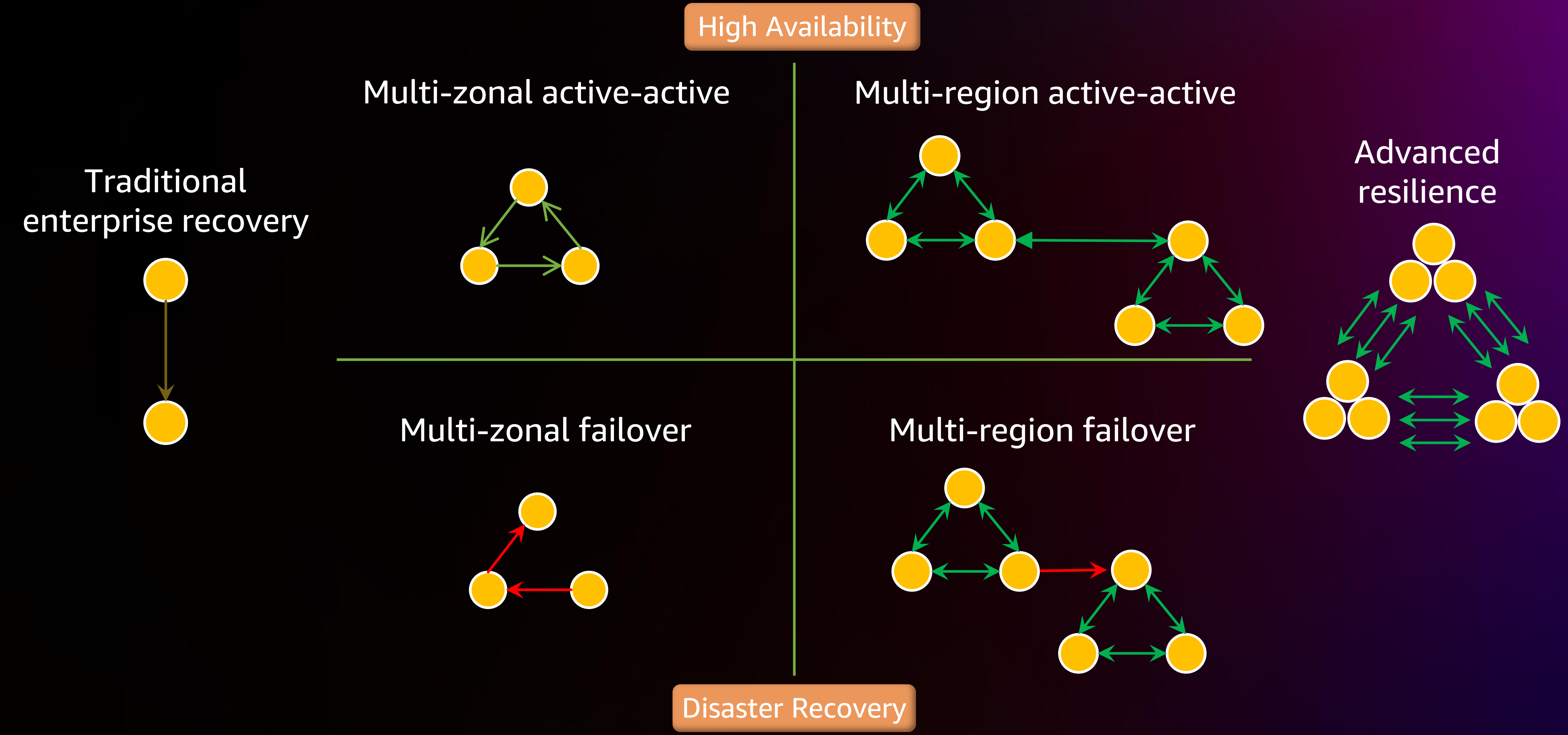
Can't be fully mitigated

## What happens when a workload becomes too big to fail?

## Let’s talk about a real-world scenario . . .

## How do we do this today?

# Designing resilient architectures



# How do we usually protect against failures?

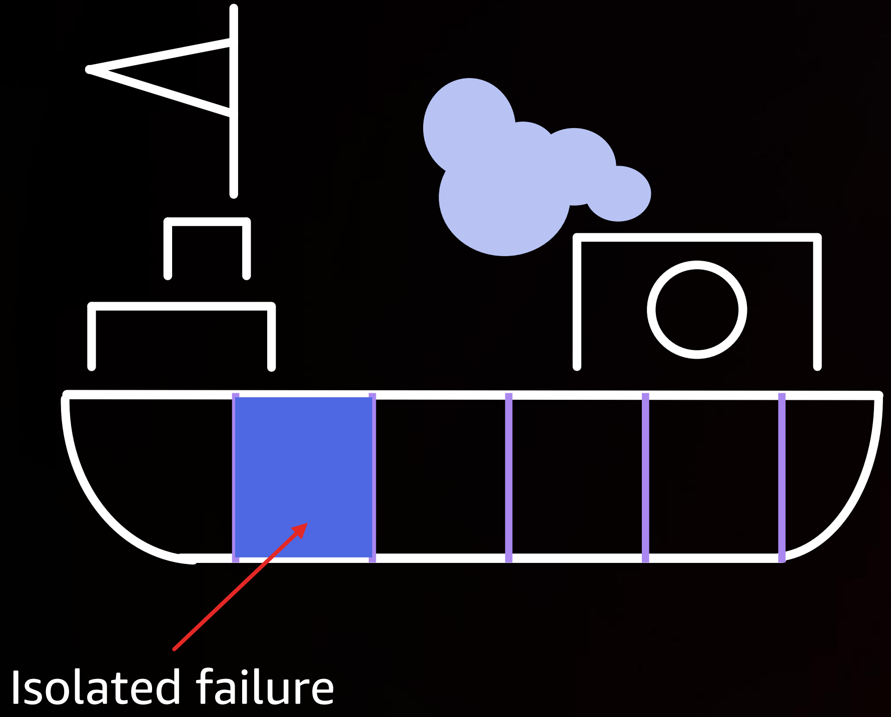
| Failure mode | | Means of protection | |
|---|---|---|---|
| Rack down | ← | Autoscaling Groups | ✓ |
| AZ impairment | ← | Multi-AZ Deployments | ✓ |
| Regional impairment | ← | Region Failover | ✓ |
| Bad deployment | ← | Canary deployments | ! |
| Human failure | ← | Processes, runbooks | ! |
| Poison pill | ← | Extensive testing | ! |

**Let's solve for these**

Typically only for compute

Can't be fully mitigated

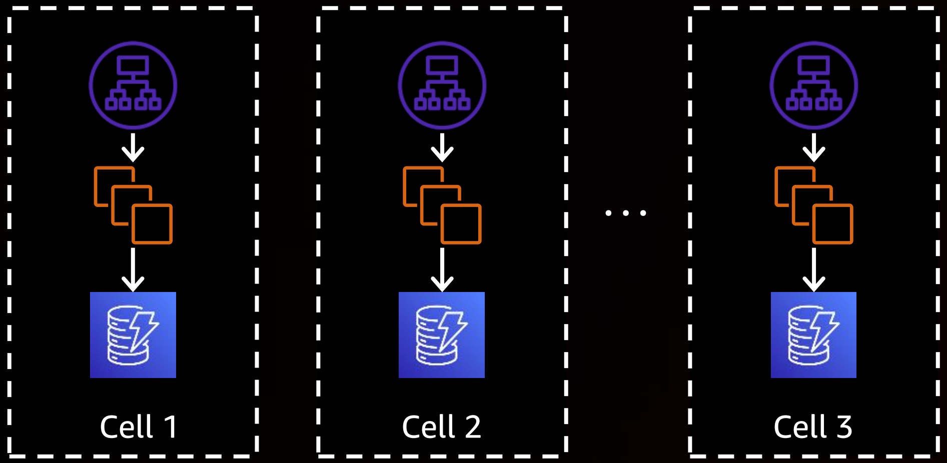
## Limiting the impact of failures



- Bulkhead pattern
- Concept from naval engineering
- Internal chambers isolate the hull
- This prevents water from filling the entire ship

## How do we build a bulkhead with software?

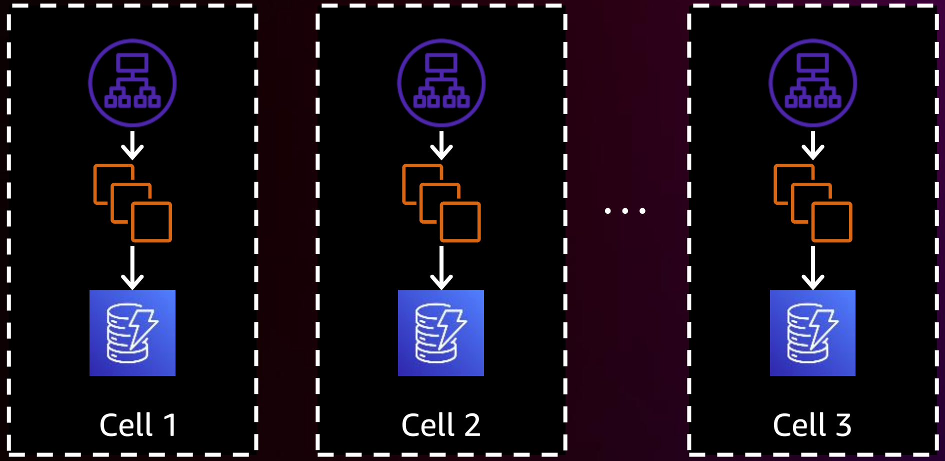
# Cell-based architectures



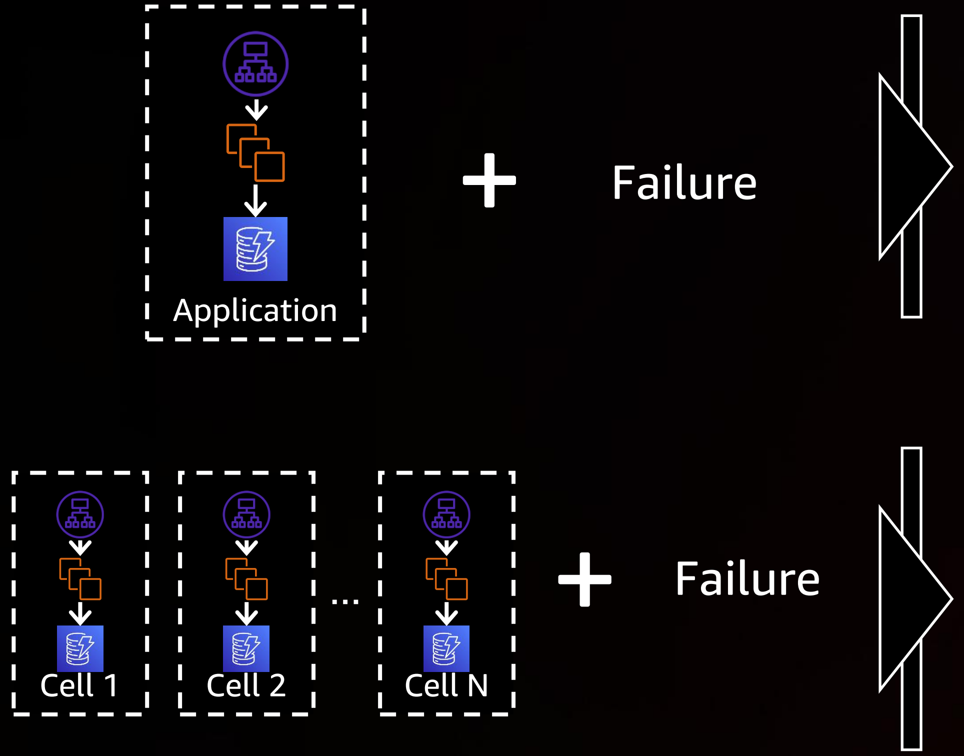
- Multiple copies of the **entire application** deployed in each cell
- Data is **partitioned** – there is no replication between cells
- **Complete isolation** between cells limits and contains failure
- Cells provide a **predictable scale** unit

# Cell-based architectures properties

- Workload isolation
- Failure containment
- Scale-out vs. scale-up
- Cells have maximum size
- Testability
- Manageability



# Cells reduce the area of impact for failures

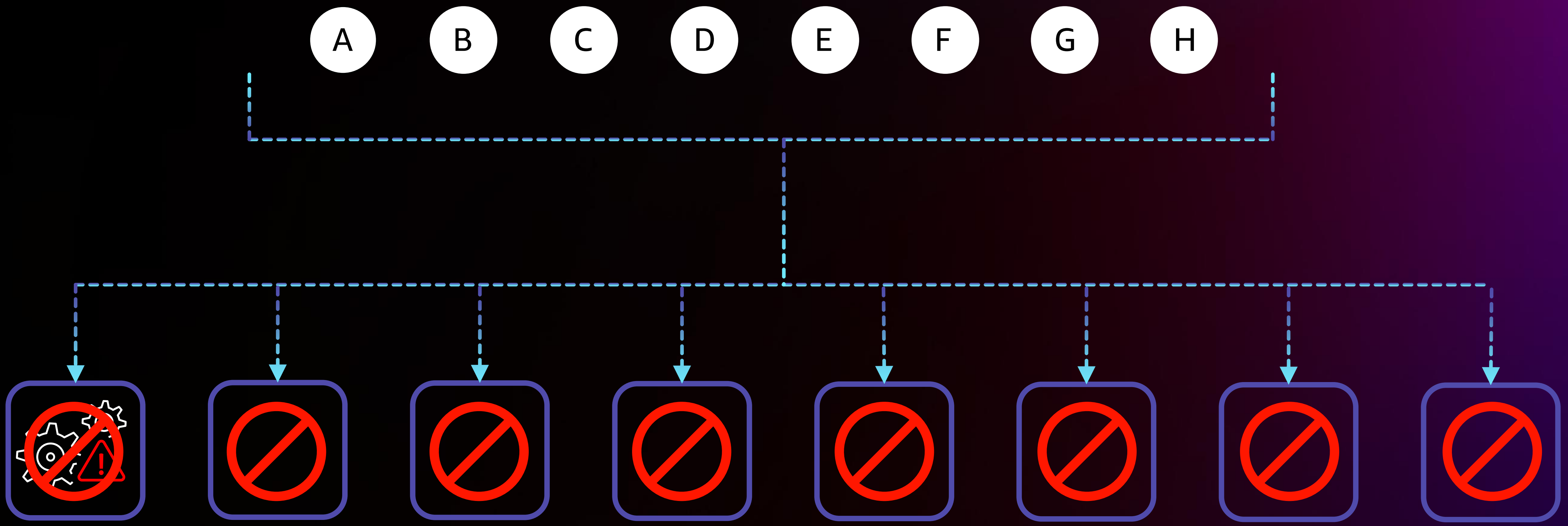


"The application was down for everyone!"

"Some users report problems accessing the application"

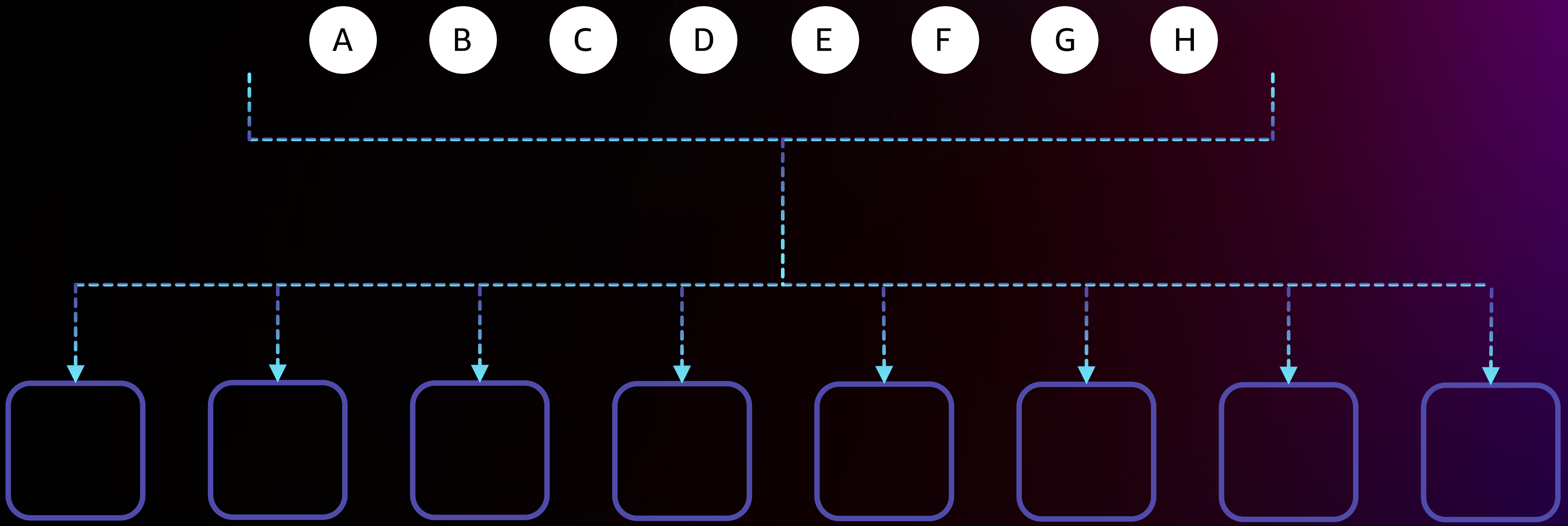
## How to achieve the benefits of isolation with access patterns?

# Traditional architecture

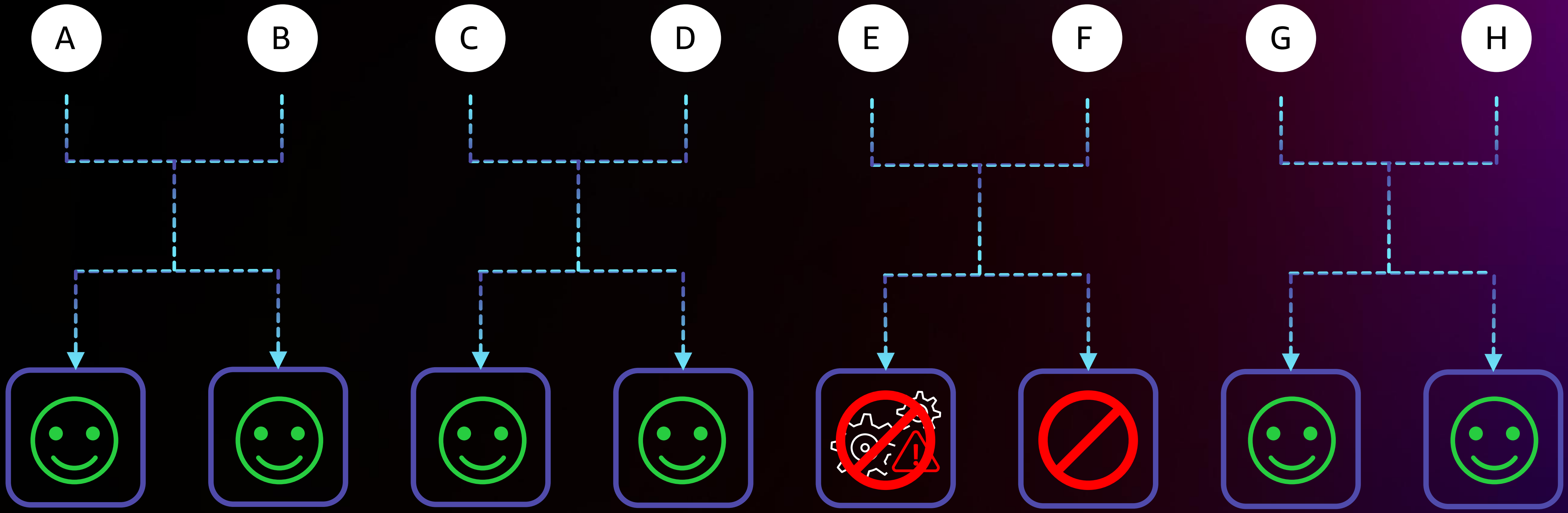


Area of impact = All clients

# Sharding



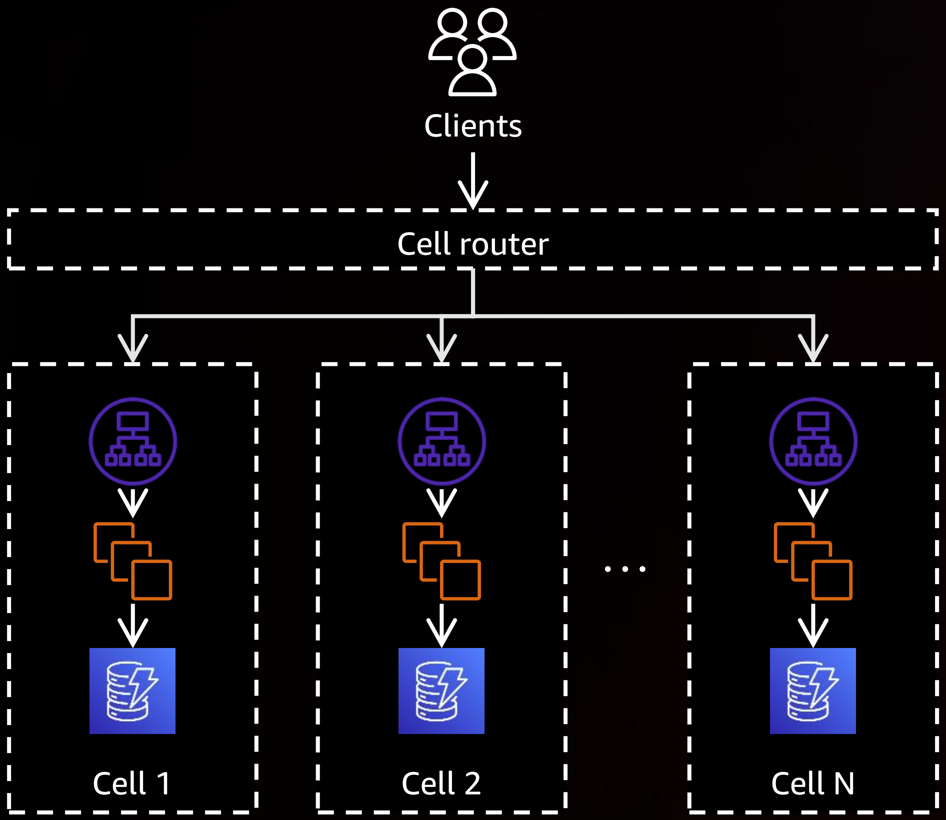
## Sharding



$$\text{Area of impact} = \frac{\text{Clients}}{\text{Shards}}$$

## How will we route the requests to the right cells?

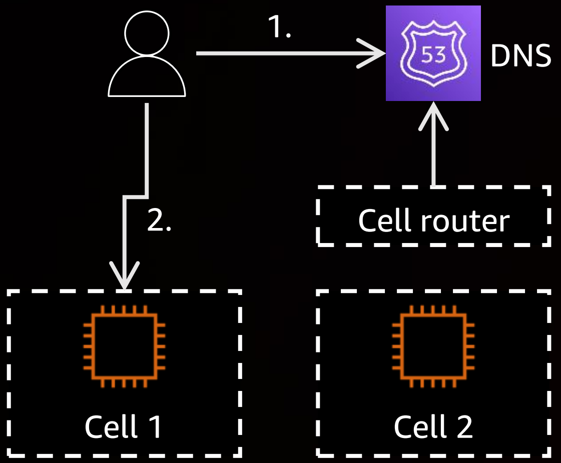
# Routing mechanisms



- Add a thin routing layer
- Needs to be resilient
- Keep it simple
- Statically stable
- Ephemeral

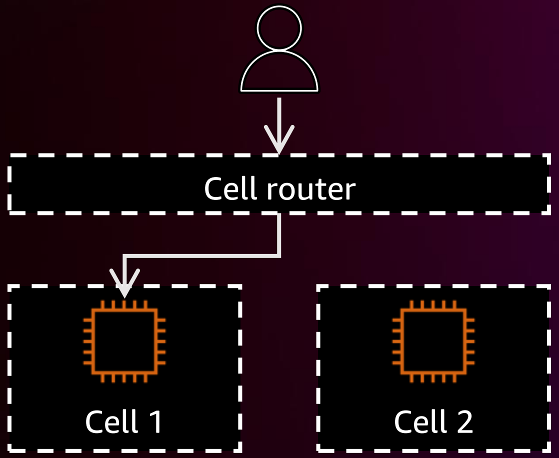
# Routing mechanisms

## Routing through DNS



- ✓ Can leverage Amazon Route 53 HA
- Clients need to map users to DNS names

## Cell router as load balancer

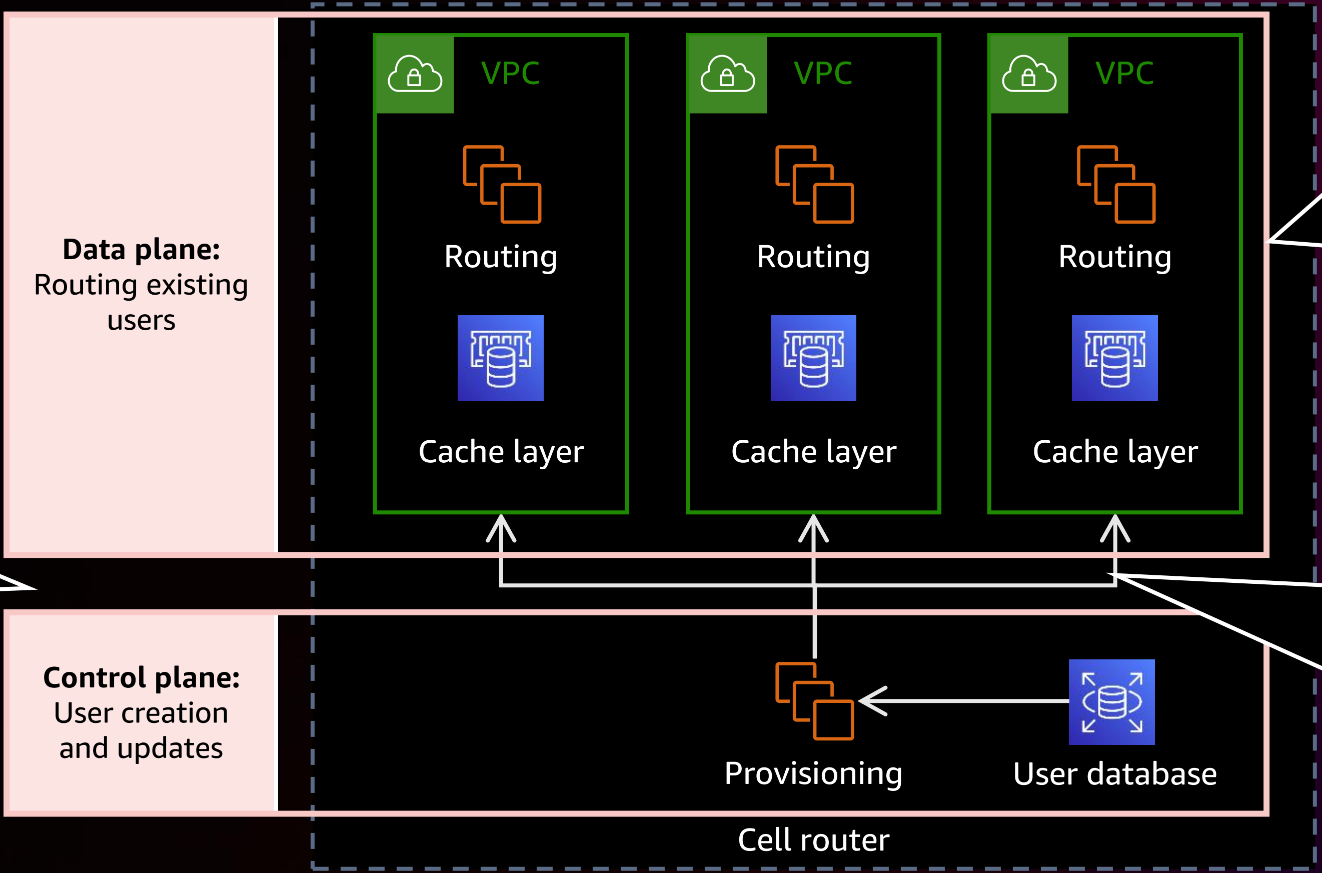


- ✓ Transparent to clients
- Cell router is critical to ongoing transactions

# Hardening the routing layer

- **The routing layer is on the critical path for all cells**
- **Apply advanced resilience patterns**

**Separate data plane** (routing existing users) and **control plane** (user creation and updates)



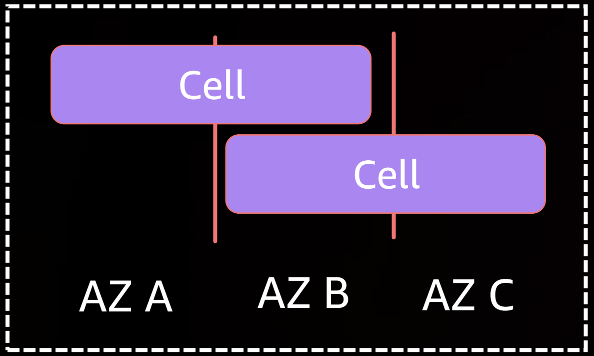
Data plane with built-in **redundancy**

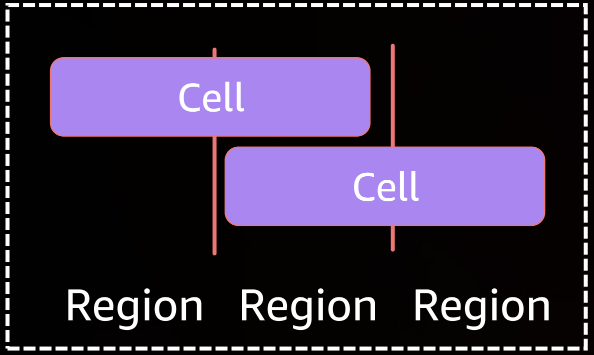
**Push** updates from the control plane to the data plane (For increased resilience use **the constant work pattern**)

https://aws.amazon.com/builders-library/reliability-and-constant-work/

## Cells are not Availability Zones or Regions

# Cells are a logical isolation mechanism



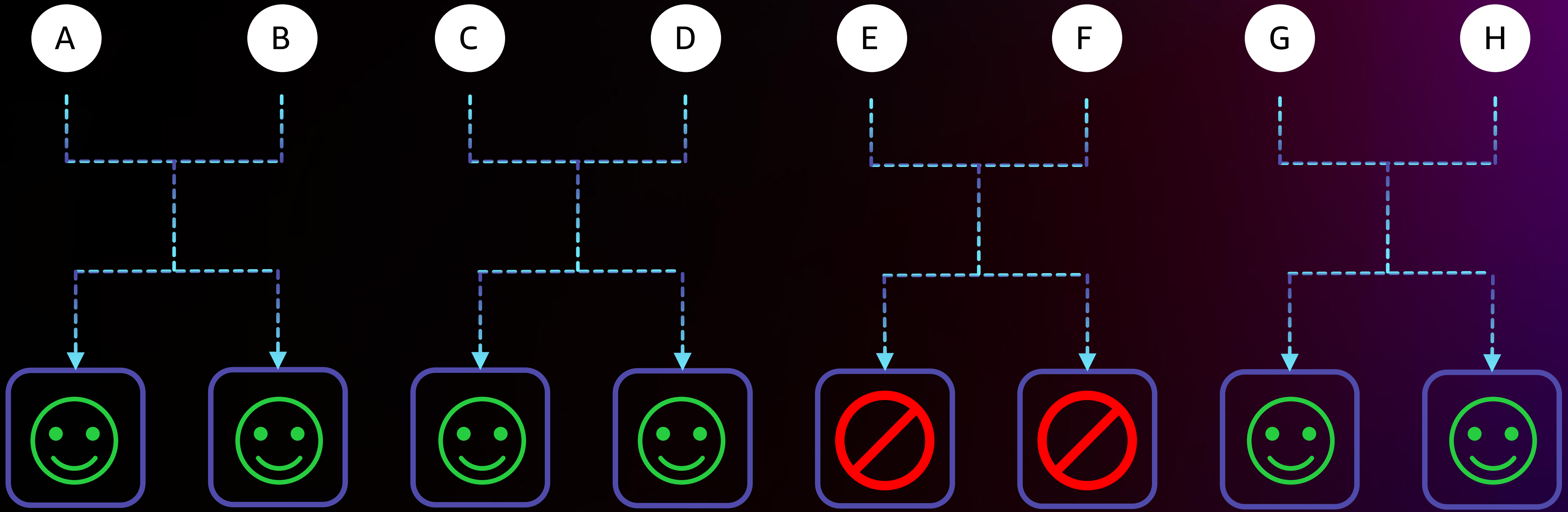


- Availability Zones and Regions provide redundancy and physical isolation
- Cells provide compartmentalization
- Design for the combination; cells span AZs or Regions (if necessary)

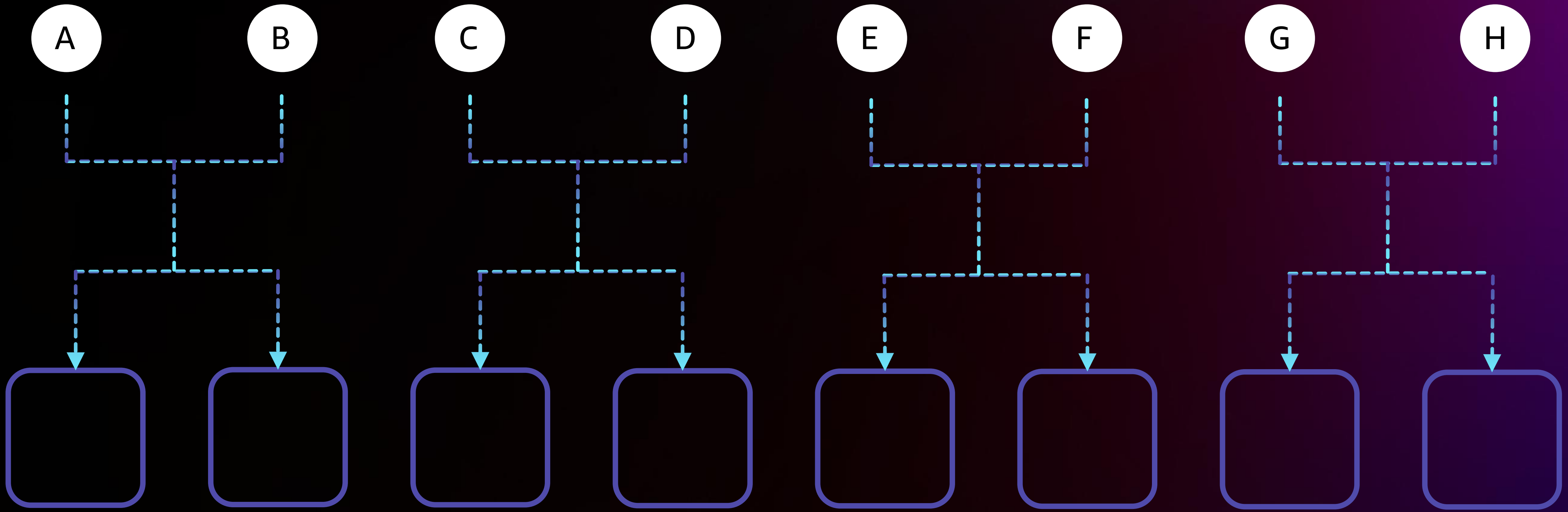
## Cells are not a scaling mechanism

## Can we make it even more resilient?

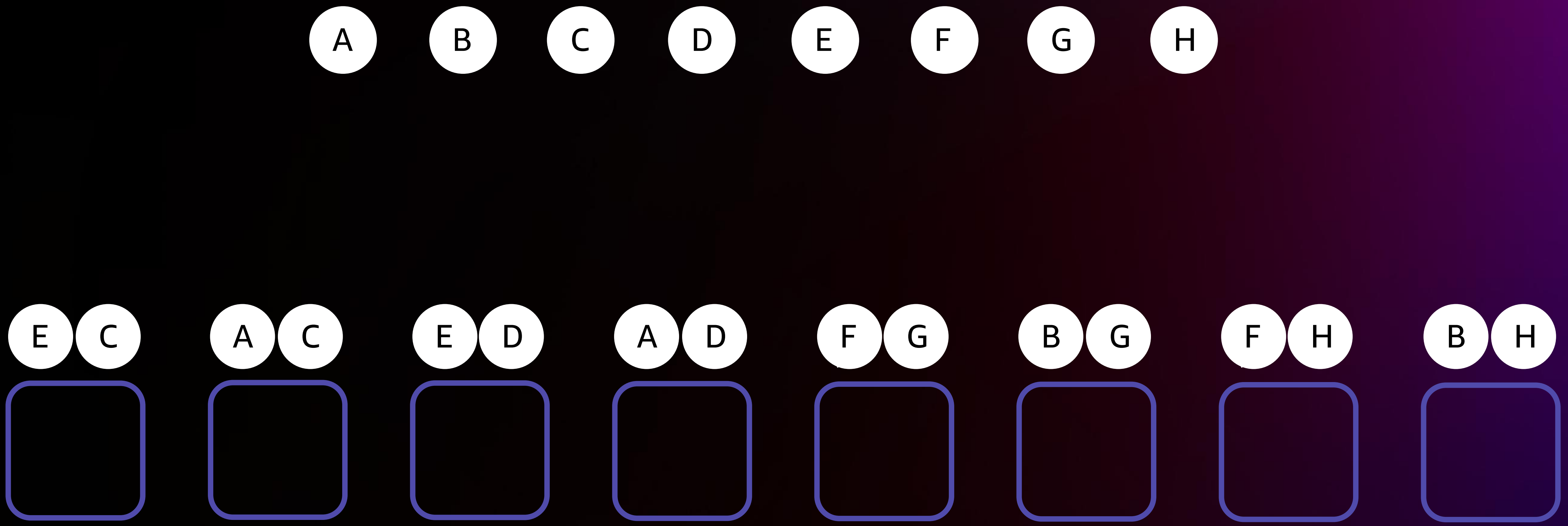
# Sharding



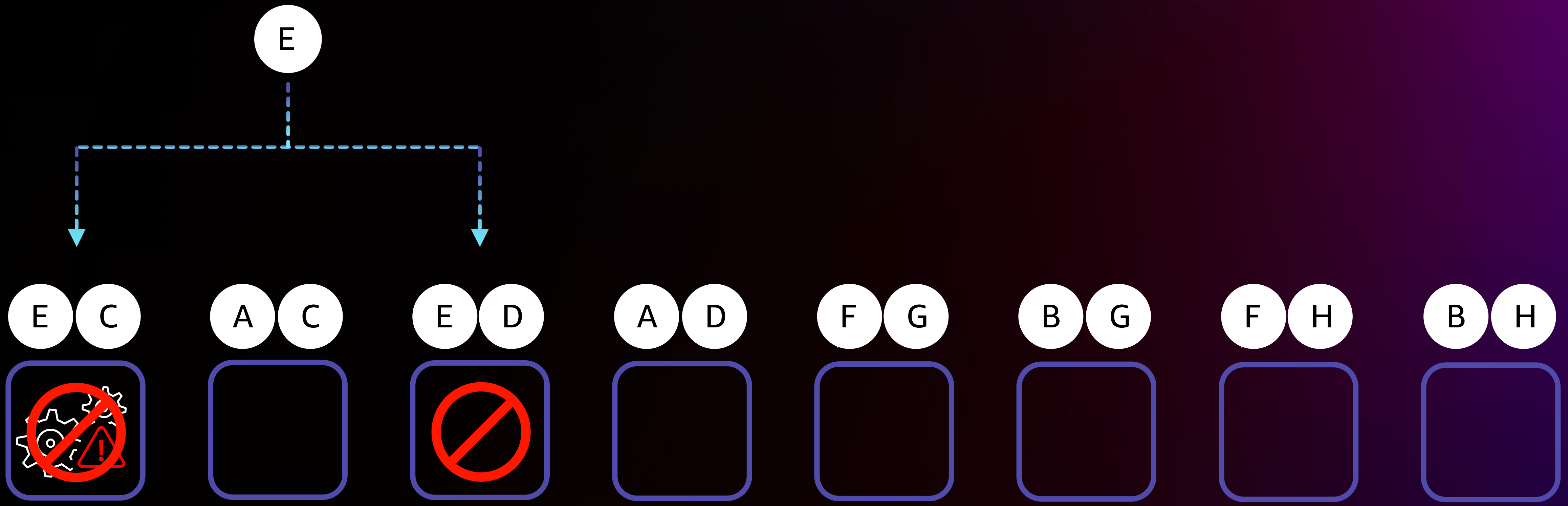
# Shuffle sharding



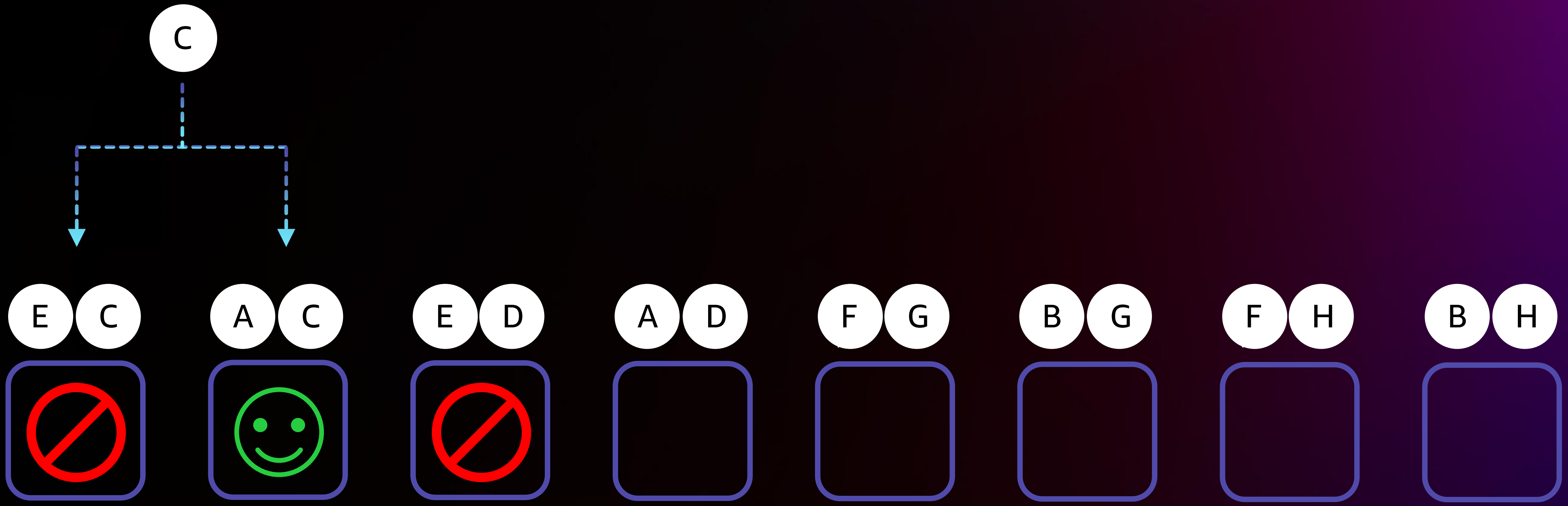
# Shuffle sharding



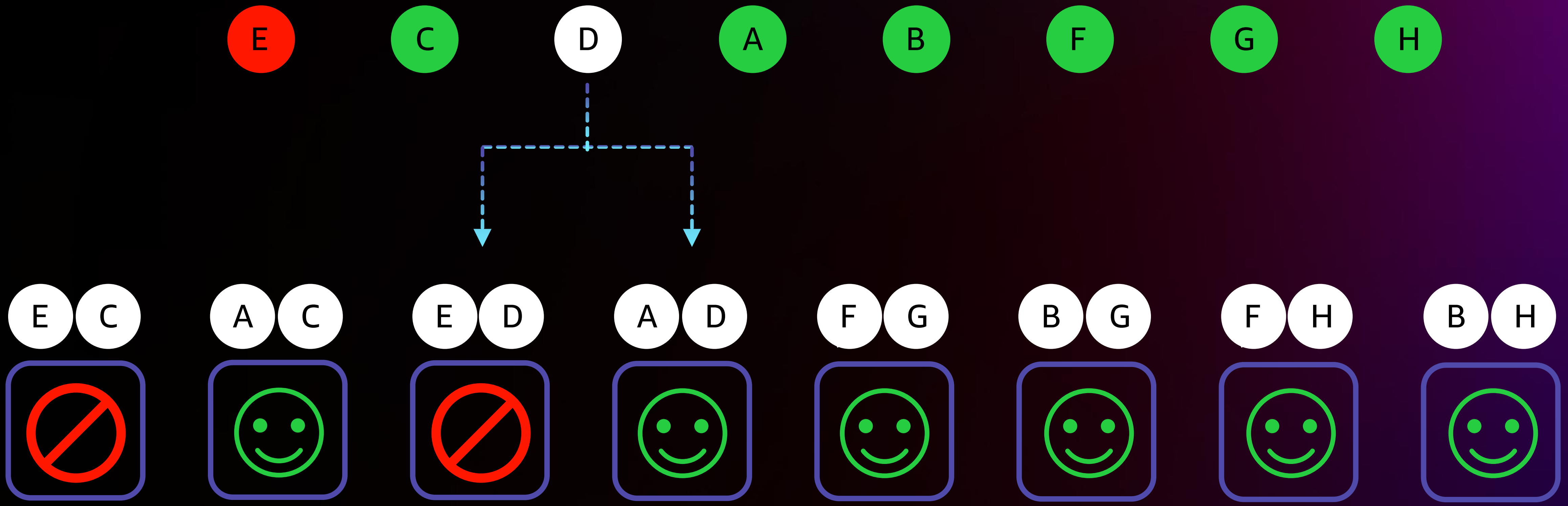
# Shuffle sharding



# Shuffle sharding



# Shuffle sharding



$$\text{Area of impact} = \frac{\text{Clients}}{\text{Combinations}}$$

## Observations

- Traditional architecture – complete outage
- Sharding
  - Impact localized to customers on the same shard
  - 25% of customers affected
- Shuffle sharding
  - Impact localized to customers having the same combination of nodes
  - 12.5% of customers affected

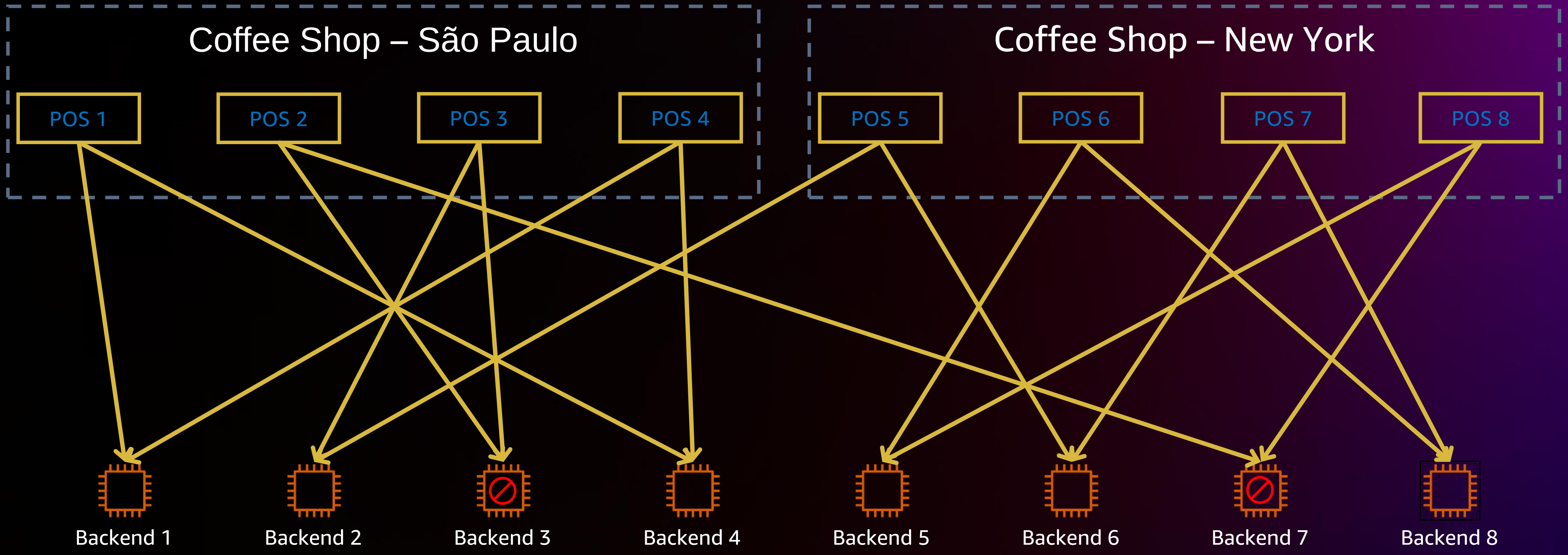
## Observations

### 100 nodes, 5 nodes per shard

| Overlap | % customers |
|---|---|
| 0 | 77% |
| 1 | 21% |
| 2 | 1.8% |
| 3 | 0.06% |
| 4 | 0.0006% |
| 5 | 0.0000013% |

Amazon Route 53

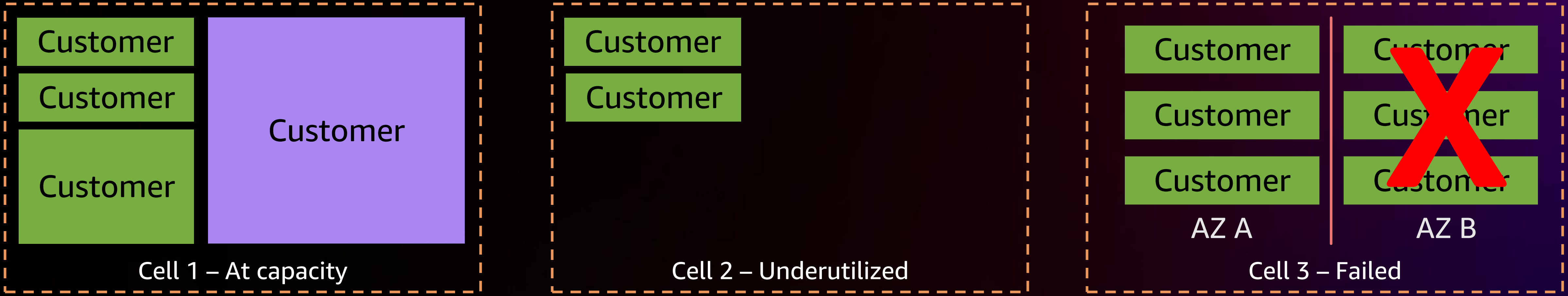
# Architecture – Shuffle sharding



## How do we operate these cells?

# Cell management

- Always build a rebalancing/migration tool
- Think about load distribution issues
- Cells can fail at any time
- Tech stack will influence management



*Sharding based on customers

# Monitoring in cell-based architectures

Monitor each cell individually

Aggregate metrics into
healthy/unhealthy cells per workload

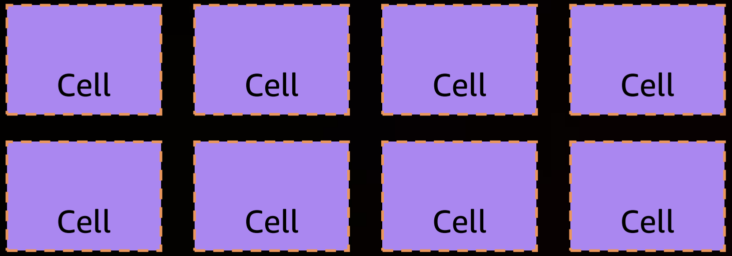
Make sure messages and errors can
easily be correlated with cells

## Is it better to have small or big cells?

# Small versus large cells

**Smaller cells**

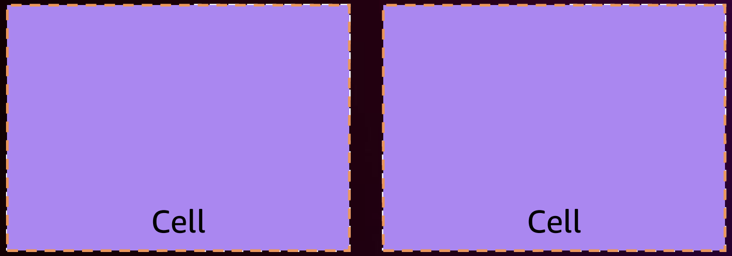
- Reduced area of impact
- Easier to test
- Cells easier to operate



VS.

**Larger cells**

- Cost efficiency
- Reduced splits
- System easier to operate



It depends!

# Other topics to consider

**Authentication**?
Where are credentials stored?

How do we **network?**

Does each cell have its own
**SSL certificate?**

What do we have on a cell basis?
VPCs? **AWS accounts?**

Where are **team boundaries?**
Is each type of cell maintained by one or multiple teams?

How do we handle
**infrastructure failures?**
How do we do **disaster recovery?**

## So, this solution will solve . . .

# Recap . . .

- Cells provide logical isolation
- There is a need to focus on data modeling
- Build a thin, statically stable routing layer

- Monitoring changes at a cell and service level
- Think about deployments
- Cell orchestration needs to be implemented
- Extreme resilience can be achieved with shuffle sharding

- Cells are not AZs
- Cells are not Regions
- Cells are not a scaling mechanism
- Data is not shared between cells
- Cells have no inter-dependent logic

# Thank you!

Please complete the session survey in the **mobile app**