aws SUMMIT

Build planetary scale applications with compartmentalization

Julien Lépine

Solutions Architect, Amazon Web Services

Software is taking over the world

SaaS impacts the entire value chain

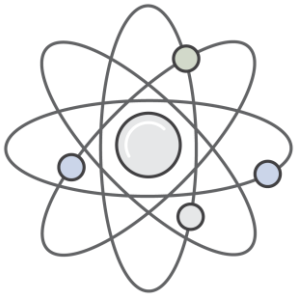
KASPERSKY lab okta AUTODESK. Adobe GE

Scale

Global Reach

Efficiency

From a single global platform



Global Platform

Centralized

Stable

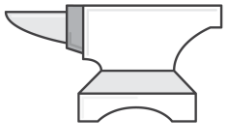
Highly available

Industrialized

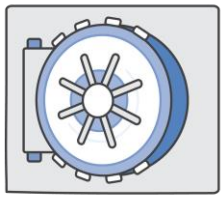
➢ **Focused on efficiency**

AWS Well Architected Framework

Performance



Reliability

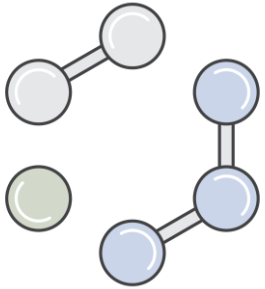


Security

Cost Efficiency

Operational Excellence

To a mesh of collaborating platforms



Global Platforms

Decentralized and replicated

Fully automated

Fault tolerant and isolated

Highly customizable

➢ **Focused on effectiveness**

AWS provides a wide set of services

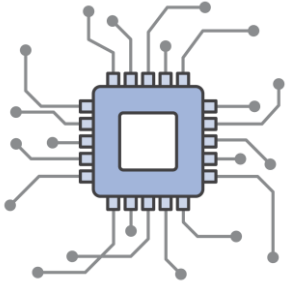
AWS
CloudFormation

Amazon
VPC

Amazon
DynamoDB

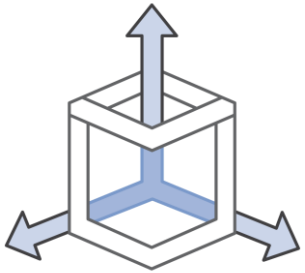
Amazon
CloudFront

SDK



Automation
Control

Data
Protection



Extensibility
Evolution

Intuit Cuts Costs to Run Key Consumer App by 6X Using AWS

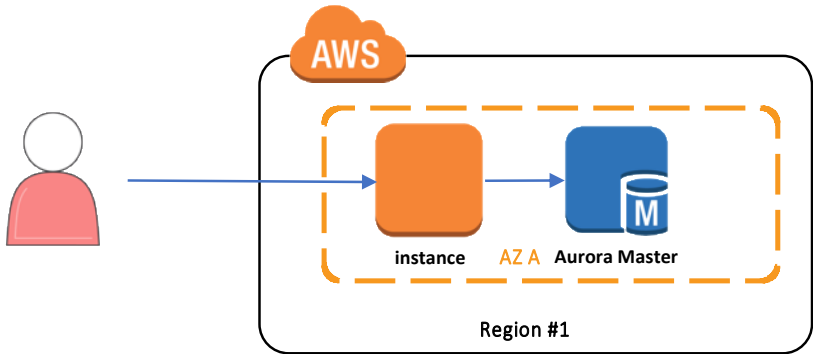
> AWS is strategic to our business and is playing a critical role in driving our business growth.
>
> Tayloe Stansbury
> SVP & CTO

Intuit is a leading provider of financial management software for consumers, small businesses, and accounting professionals. It is based in Mountain View, CA.

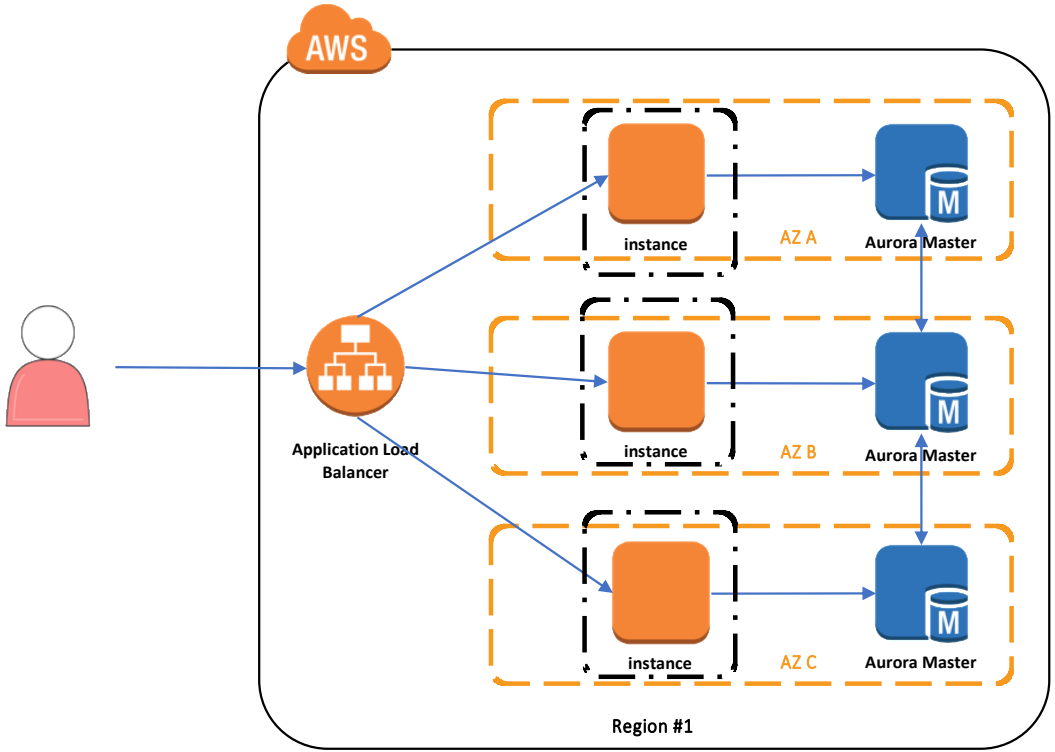
- Intuit is a leading provider of financial management software for consumers and small businesses
- Moved TurboTax AnswerXchange app to AWS when on-premises lease expired
- Pay-as-you-go AWS model cut costs to host and manage the app by factor of 6
- Set up new platform in 1/5th the time compared to old environment
- Positive experience led Intuit to move another 33 apps and 26 services to AWS

Let’s step back

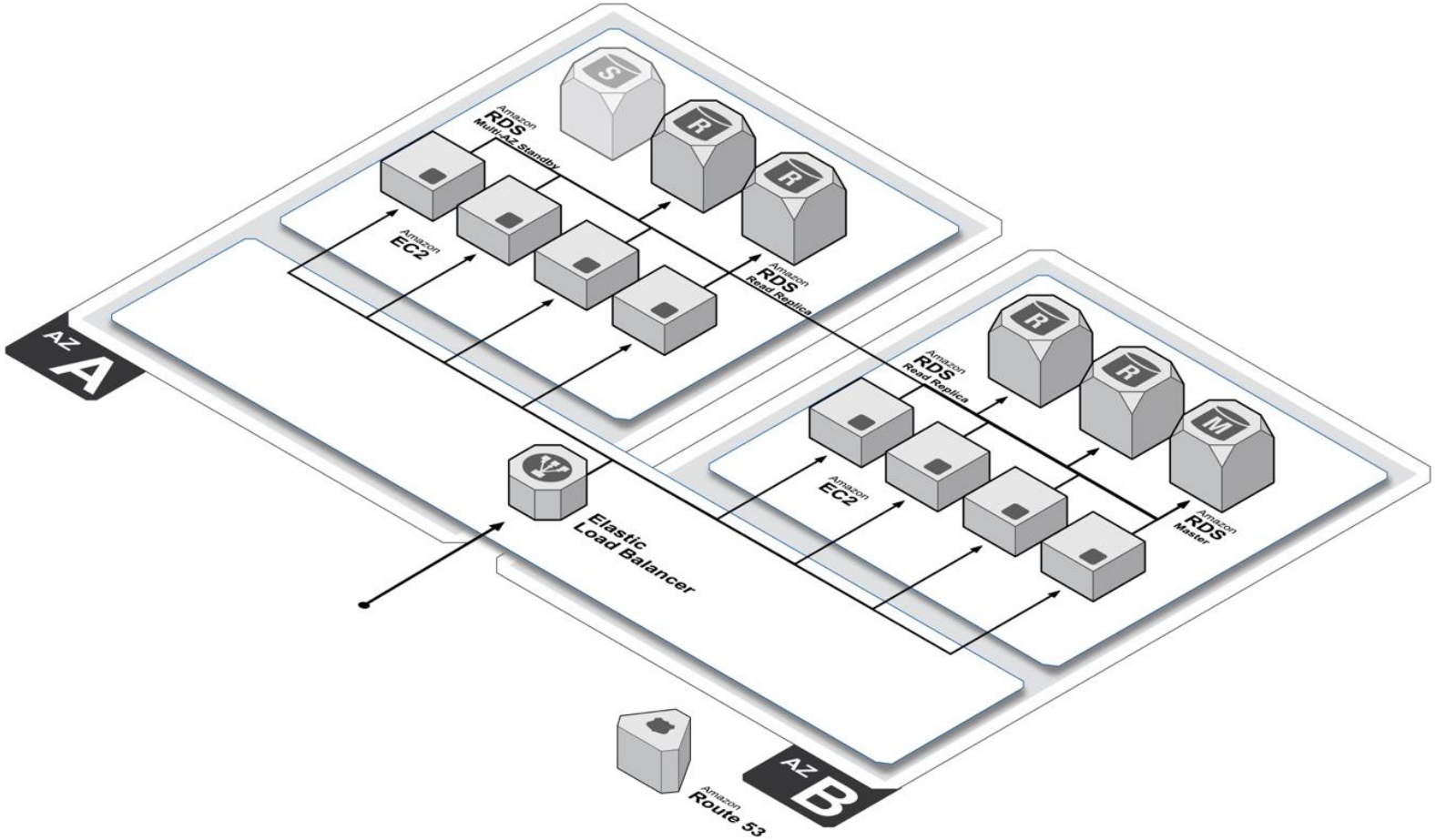
Start with a simple app



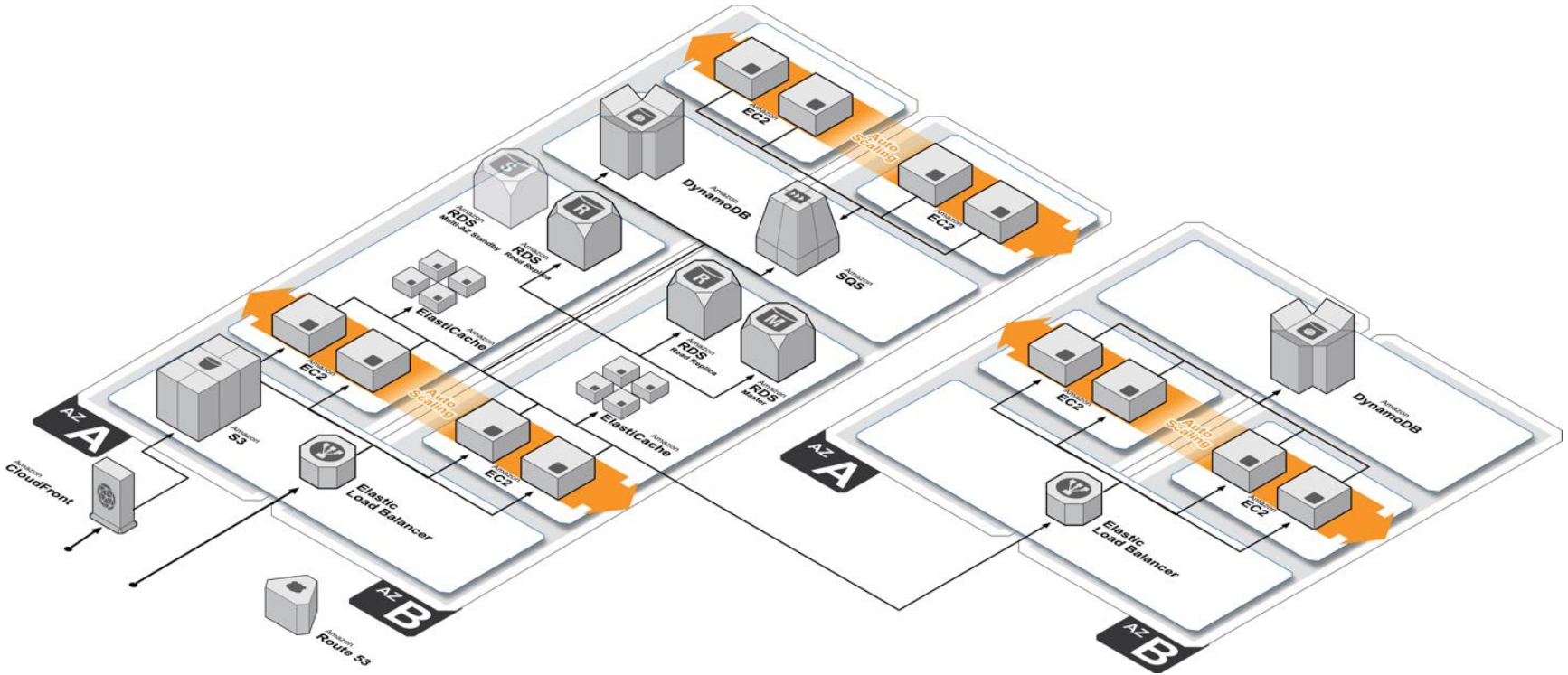
Make it a bit more reliable



We can also scale it further

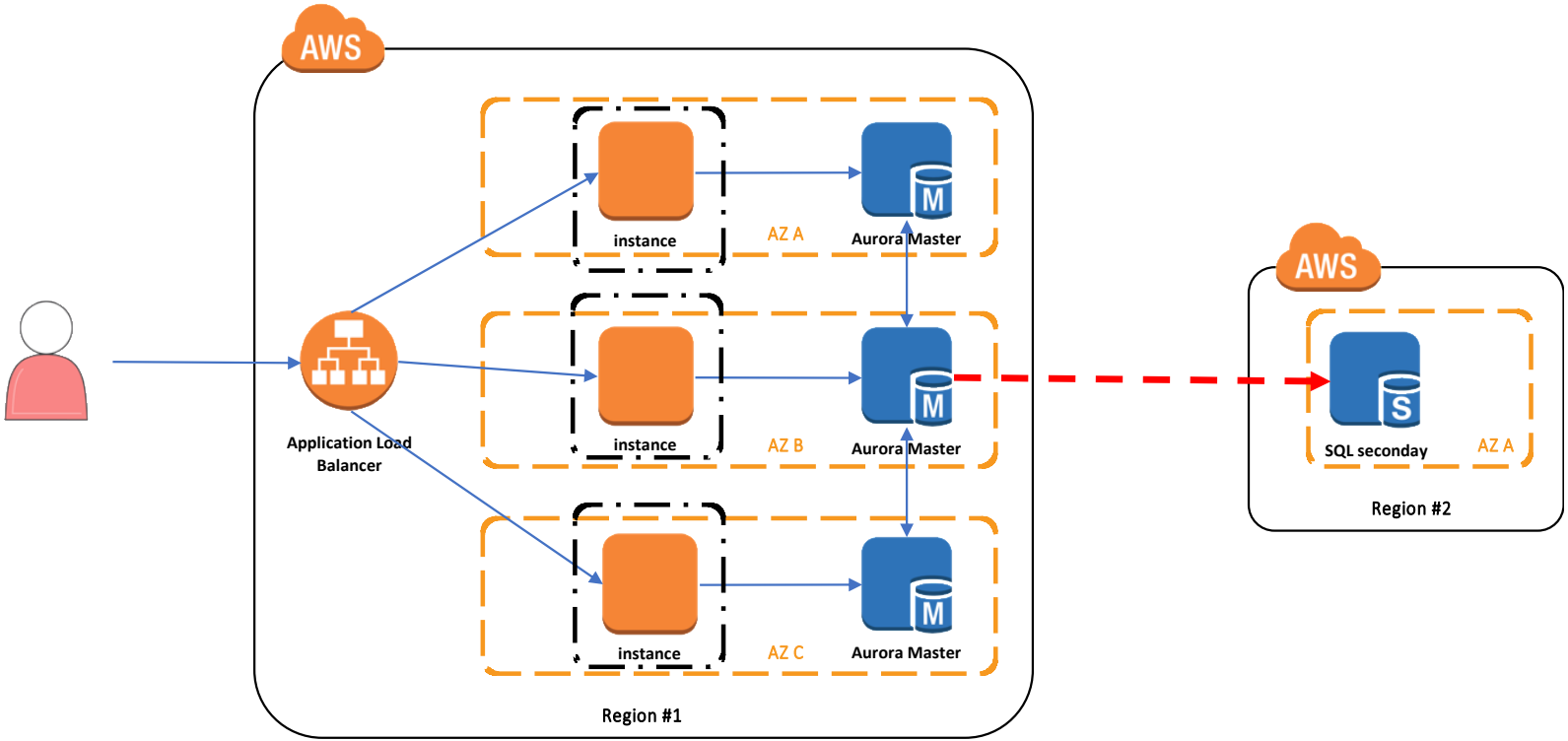


To millions of users

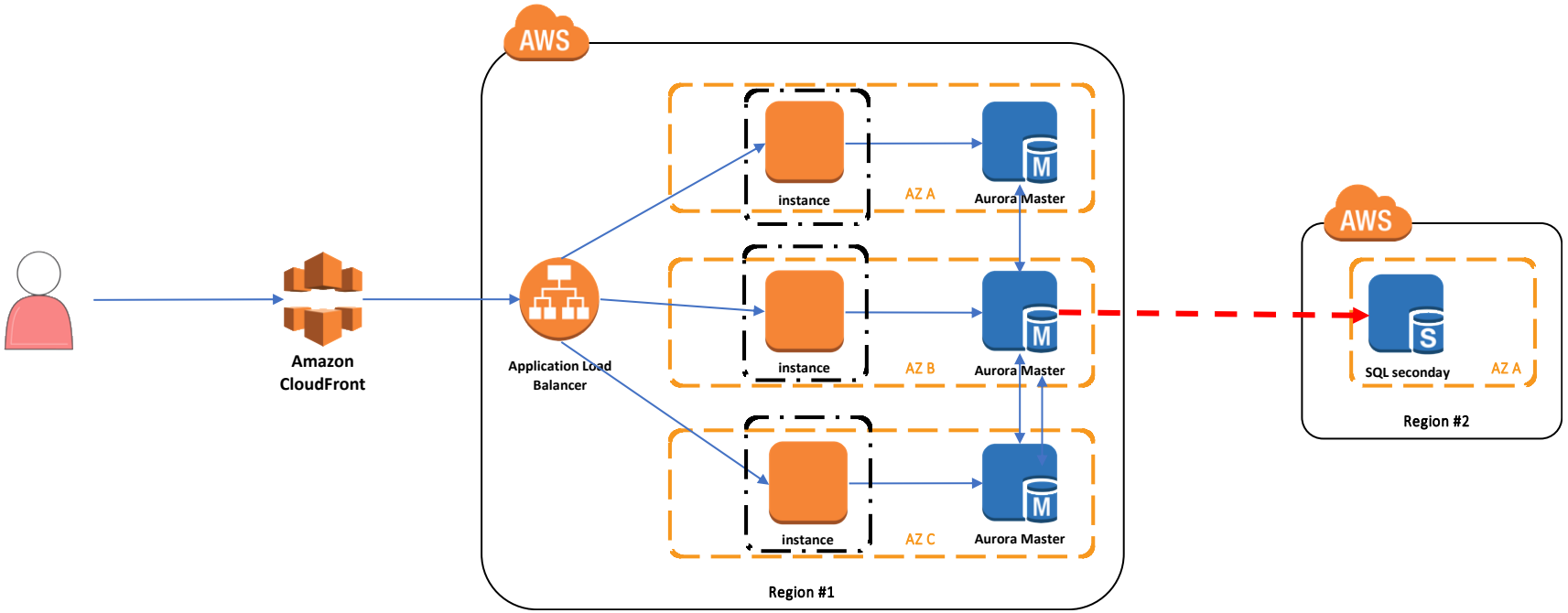


… but …

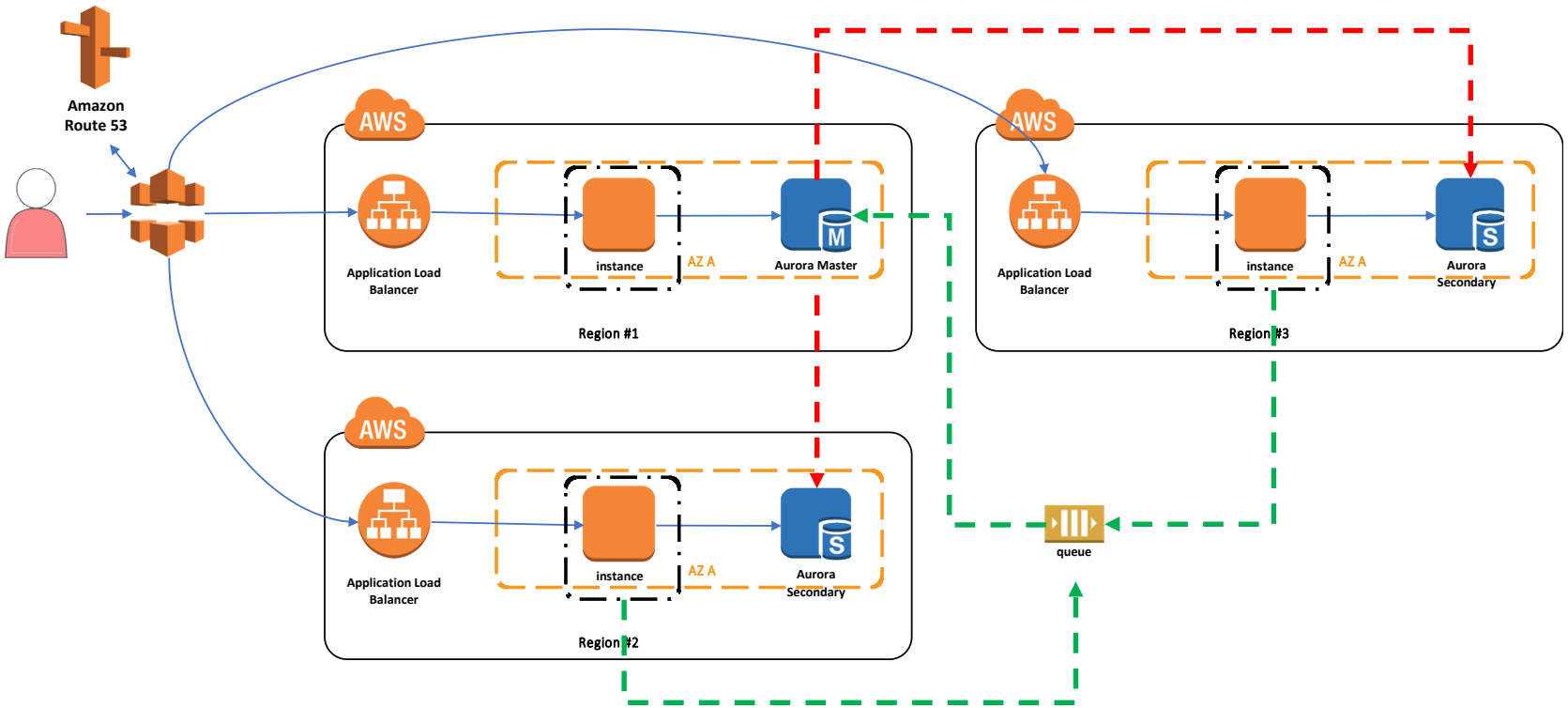
Customers will want Disaster Recovery



Globally available applications

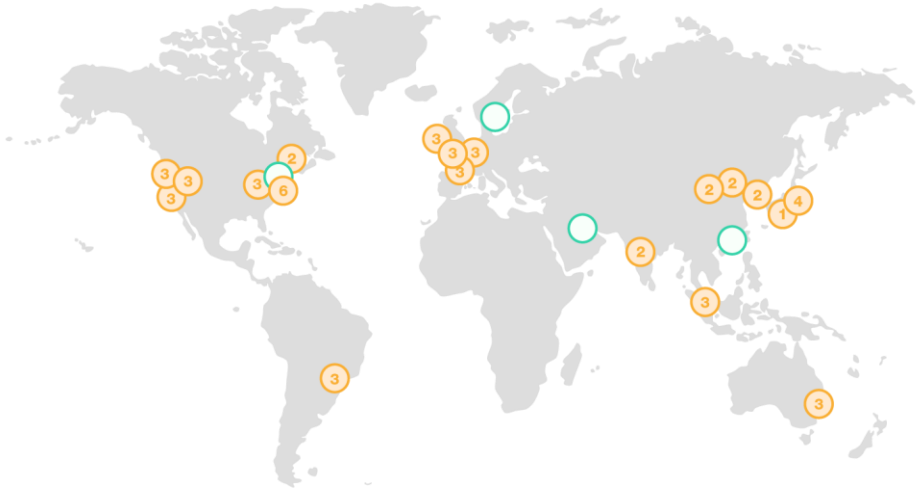


Or truly global applications



They will want more!

- Different regions
- Dedicated hardware
- Regulatory constraints
- More customers mean more ideas!



Challenges

- How to update the application, the database?
- How to handle failure?
- How to test new features, elements?

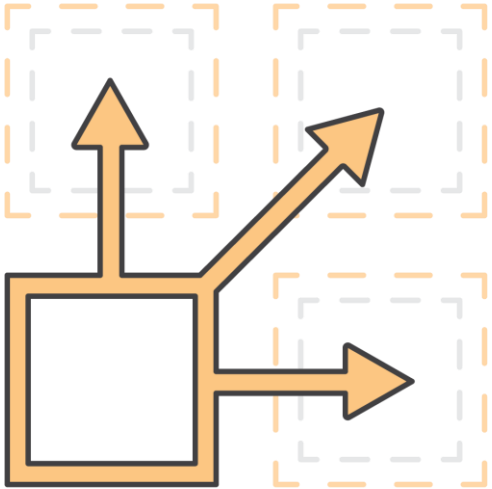
➔ At scale, this becomes an operational burden

Ok, what can we do?

What do we want?

- Adapt quickly to specific needs
- Control our blast radius
- Benefit from industry best practices

➔ Think 100x



We overall want to avoid this

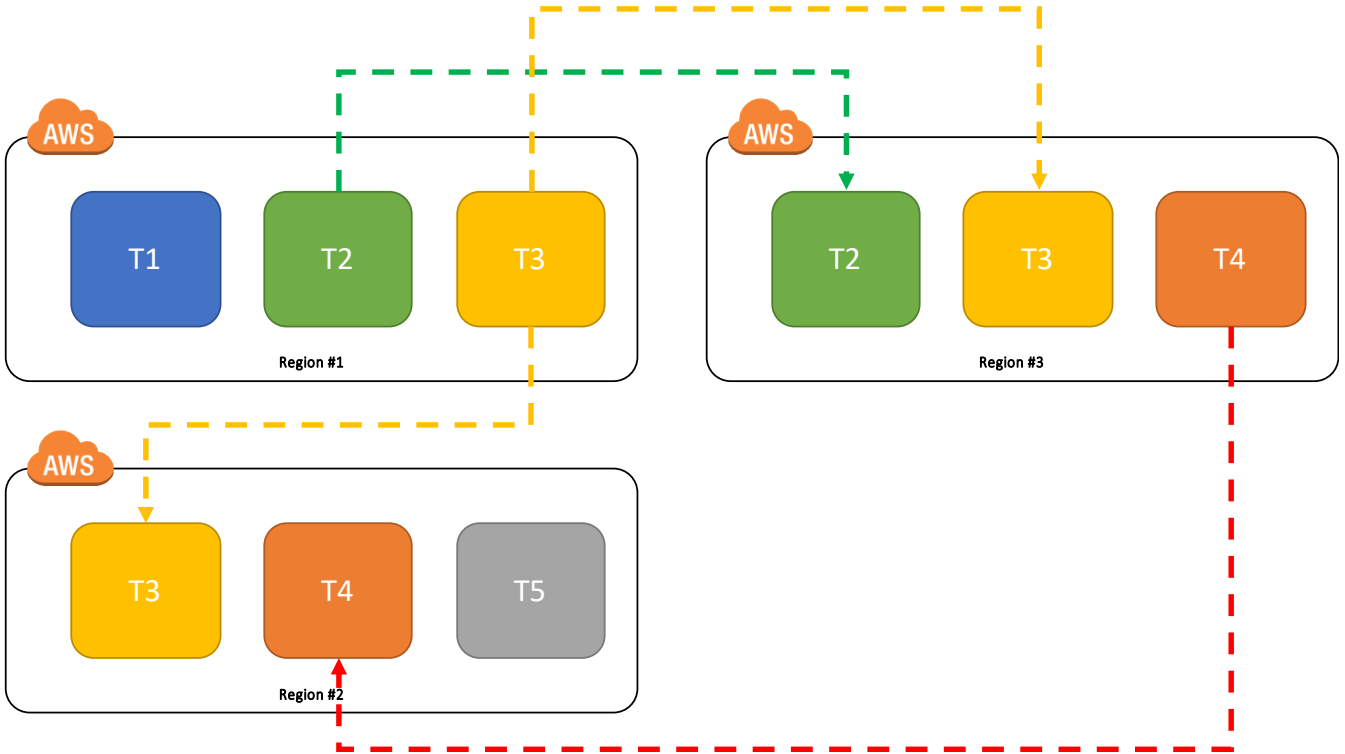
Scheduled Maintenance Notification

Maintenance planned: This entire weekend, the platform will be entirely unavailable due to ongoing maintenance on our application. Enjoy the weather and see you Monday!

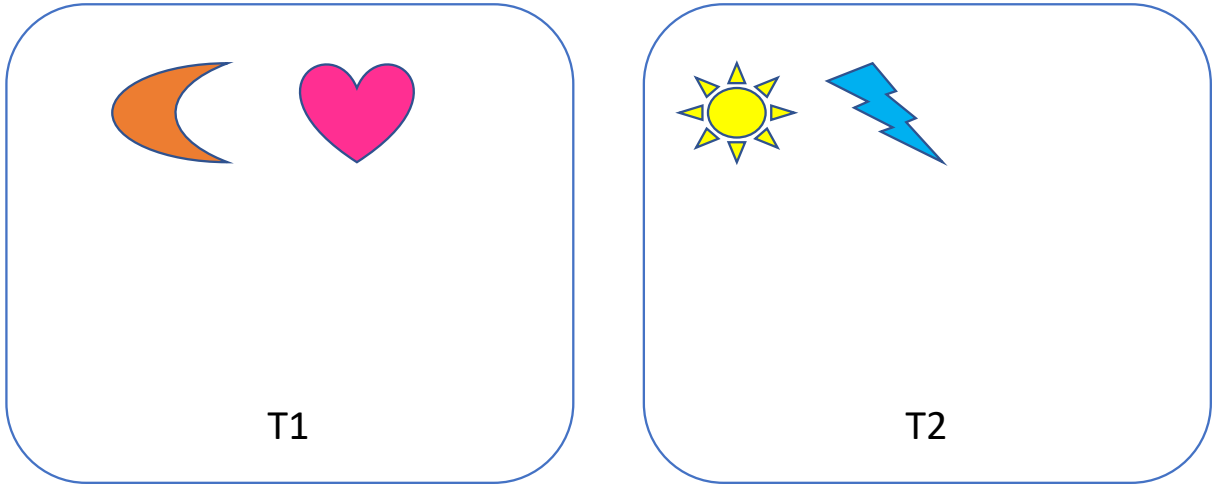
Oops, we're over capacity

We have a catastrophic success, and our application can't handle the load. Please come back later as we are working to fix the platform.

Each region will contain compartments

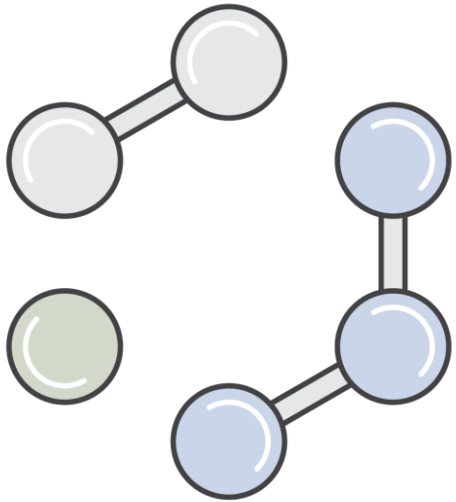


Each compartment contains customer data



What's a cell

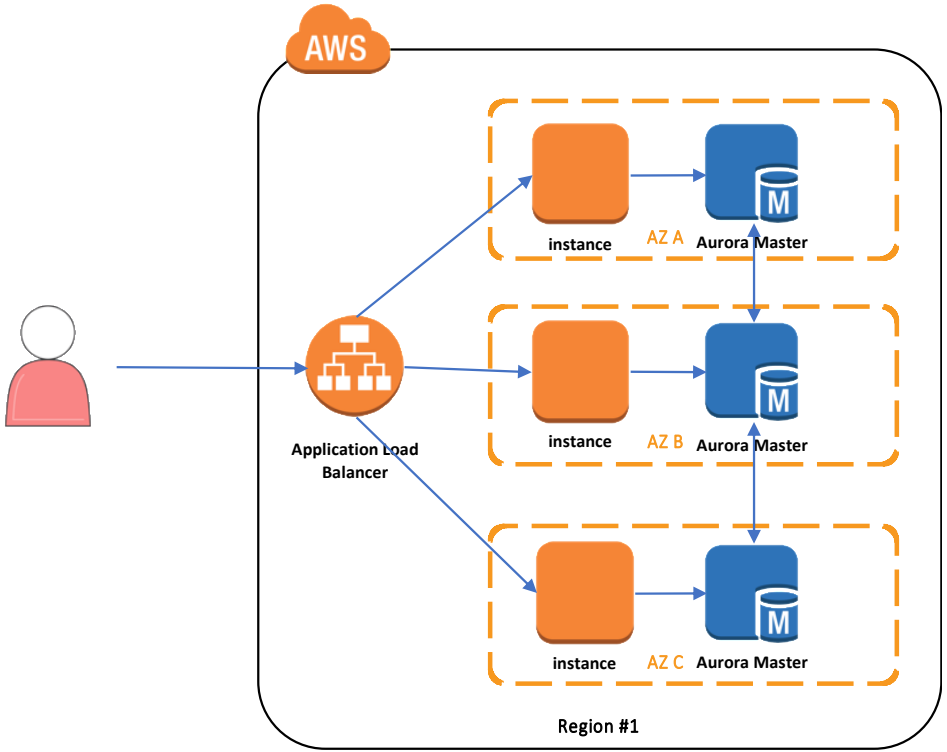
- A completely autonomous environment
  - Single or multi region / account
  - Has its own configuration
  - Responds to its own constraints



- Contains one or multiple customers
  - (But) One customer is usually hosted in a single cell

(also, much simpler to say than compartment)

Cells remains largely unchanged

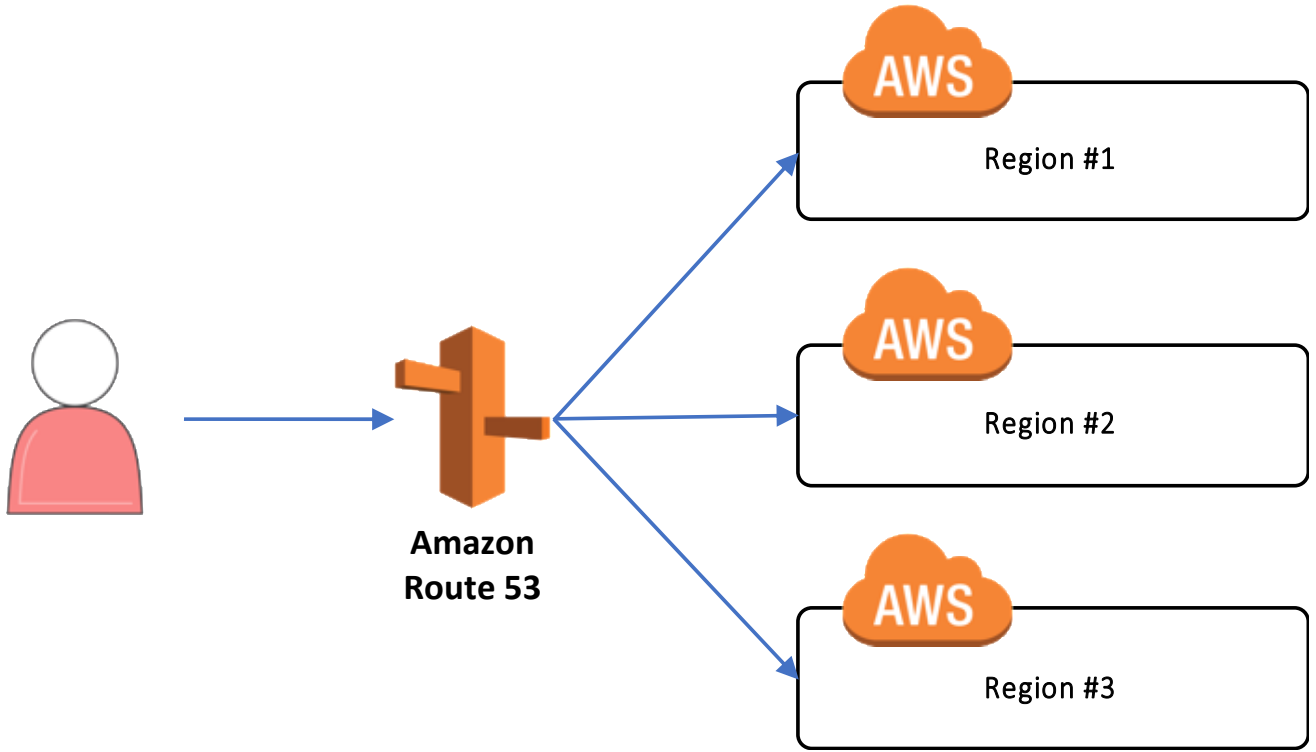


What features we need?

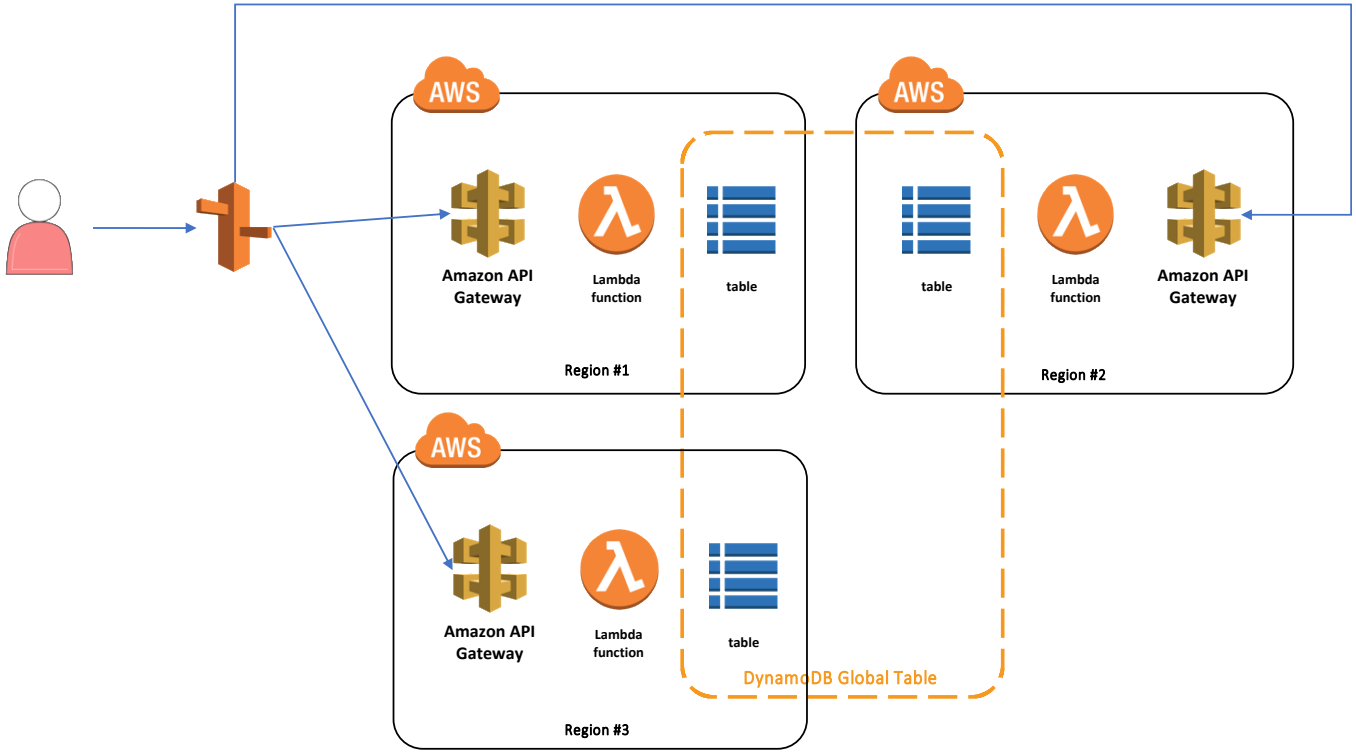
- A routing layer, assigning customers to cells
- Create and update a cell
- Migrate a customer between cells

Routing layer

Routing layer



Routing layer

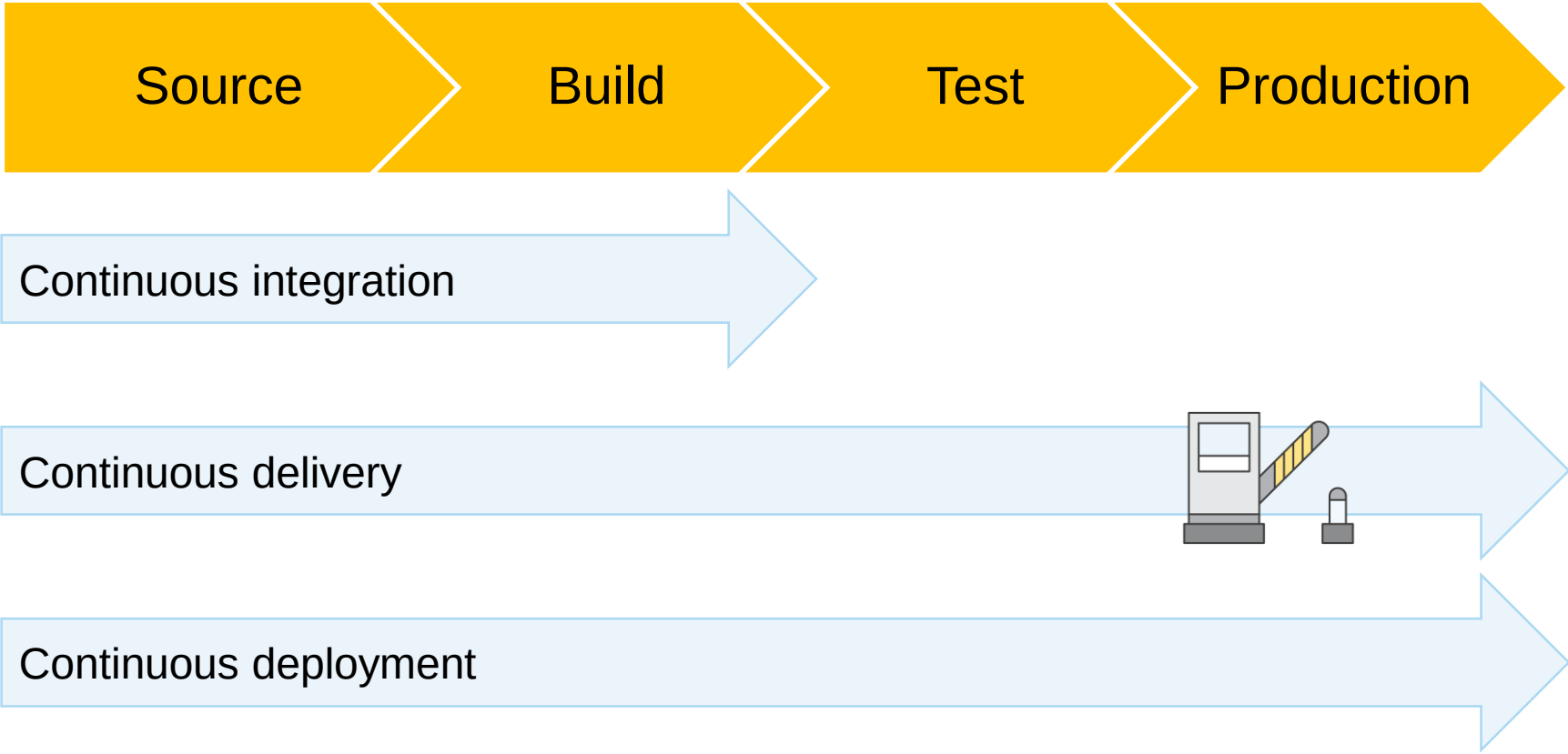


Deploying cells

Key points for deployment

- Fully automated
- Used everywhere, from dev to prod
- Configurable

CI / CD


Source
Build
Test
Production
Continuous integration
Continuous delivery
Continuous deployment

How to deploy?

Pipelines: automated actions and transitions; from check-in to production

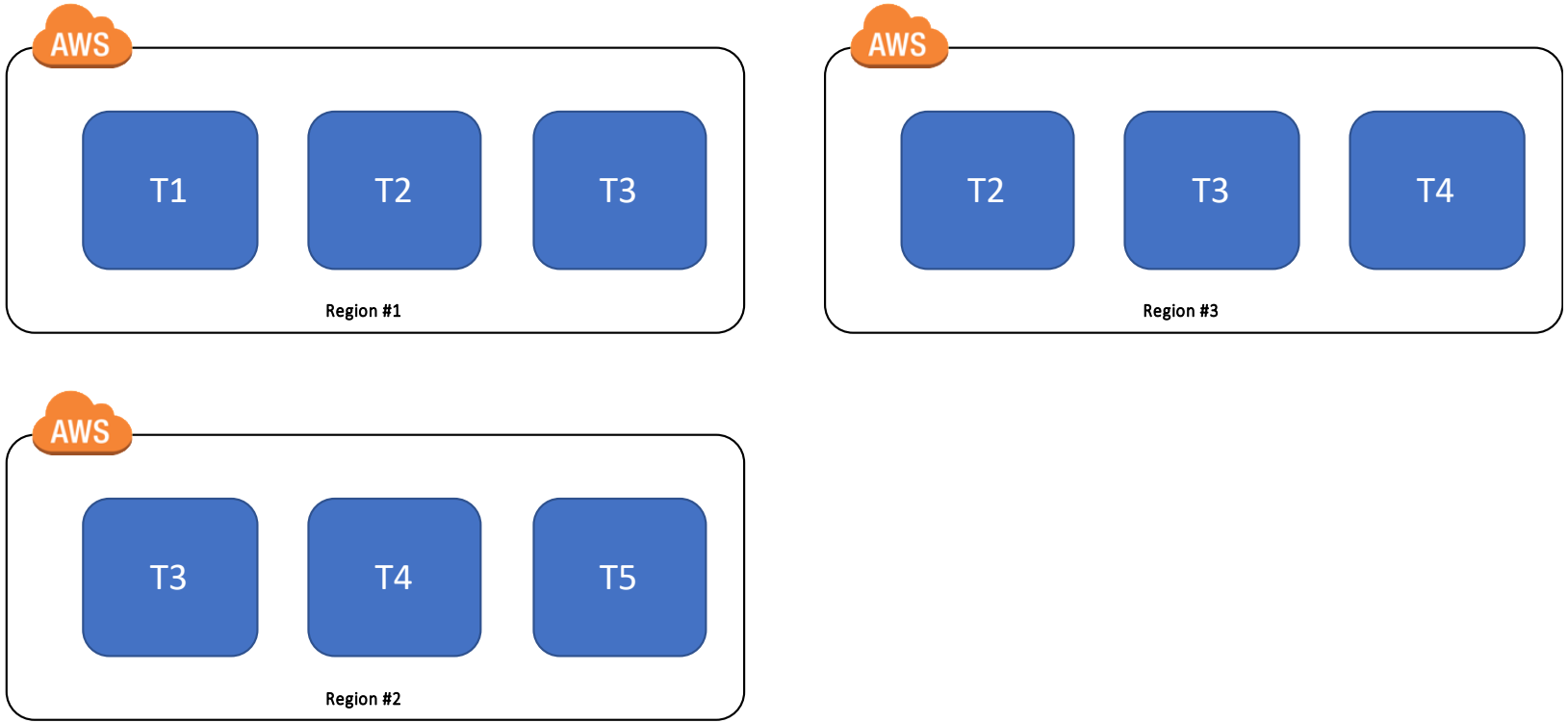
➔ Think 100x (still)

Infrastructure as code

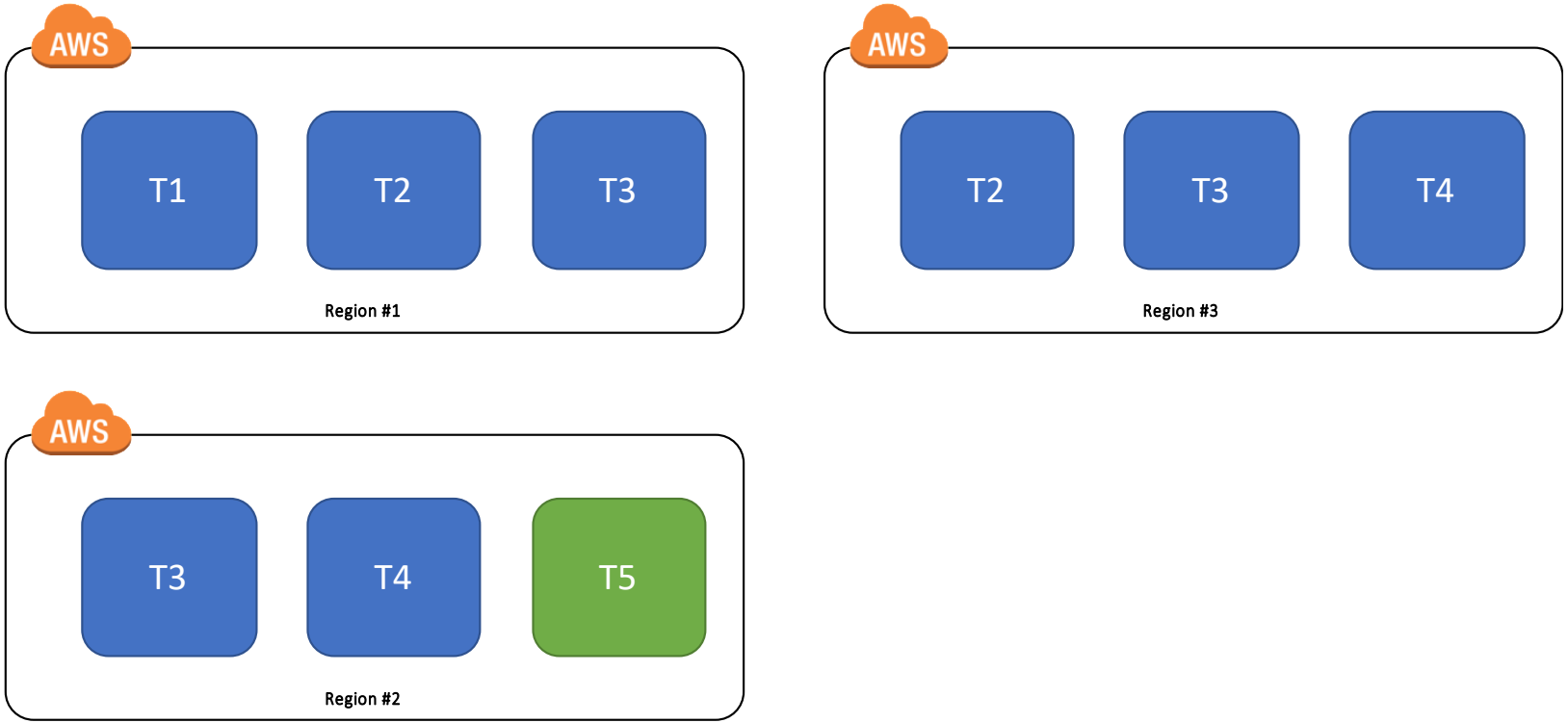
For everything, including the deployment process, and across multiple regions

```
"Resources":{
  "Ec2Instance" : {
    "Type" : "AWS::EC2::Instance",
    "Properties" : {
      "ImageId" : { "Fn::FindInMap" : [ "RegionMap", { "Ref" : "AWS::Region" }, "AMI" ]},
      "KeyName" : { "Ref" : "KeyName" },
      "NetworkInterfaces": [ {
        "AssociatePublicIpAddress": "true",
        "DeviceIndex": "0",
        "GroupSet": [{ "Ref" : "myVPCEC2SecurityGroup" }],
        "SubnetId": { "Ref" : "PublicSubnet" }
      } ]
    }
  }
}
```

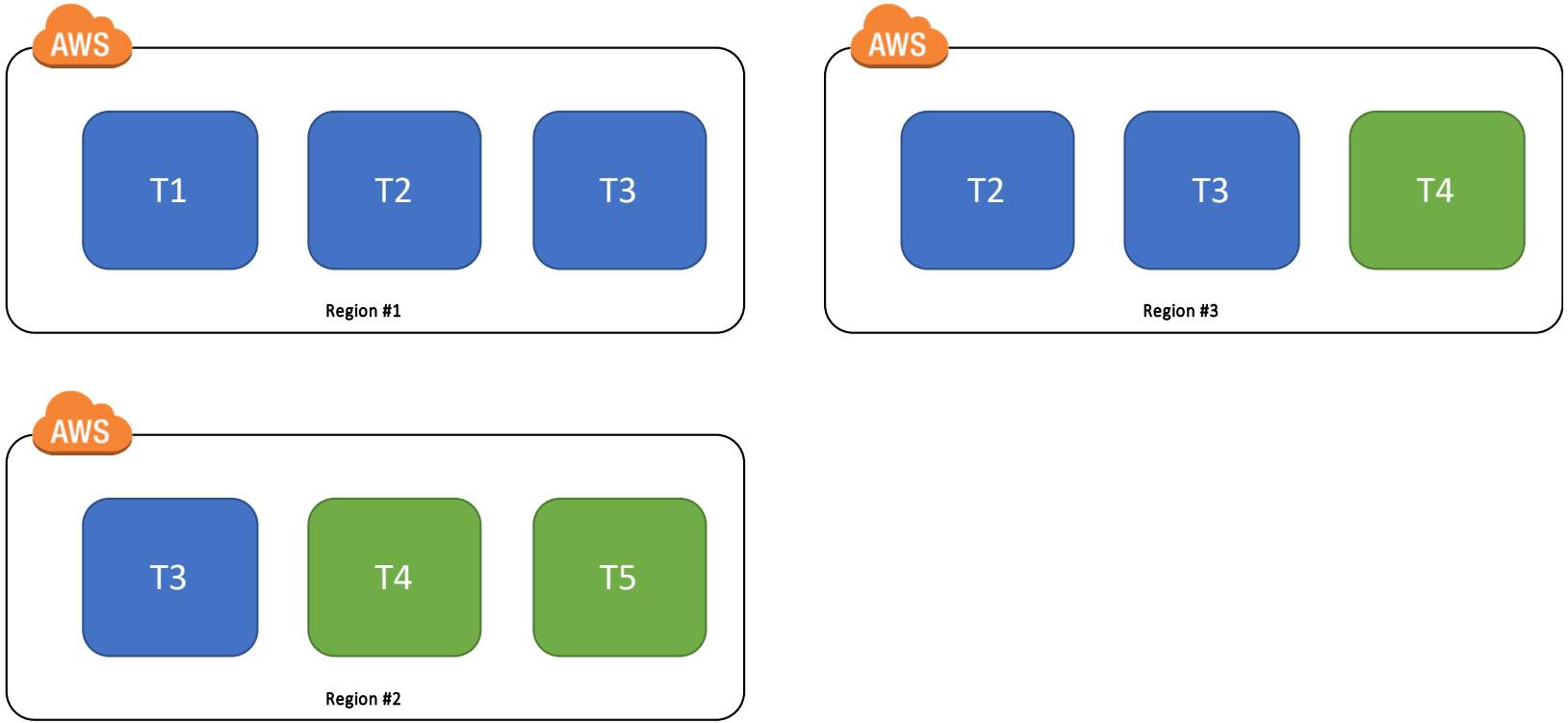
Deployment



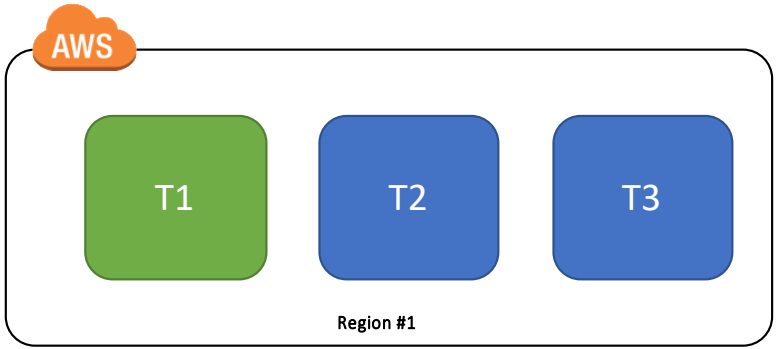
Test in 1 cell in 1 region

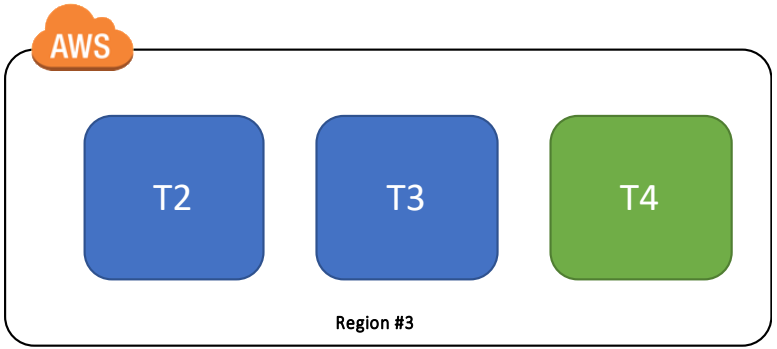


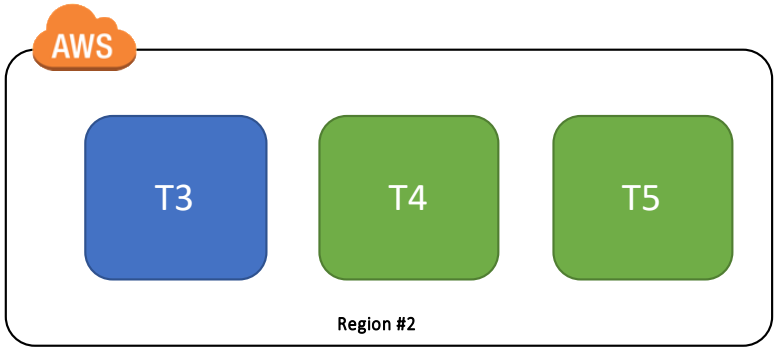
Expand to another region



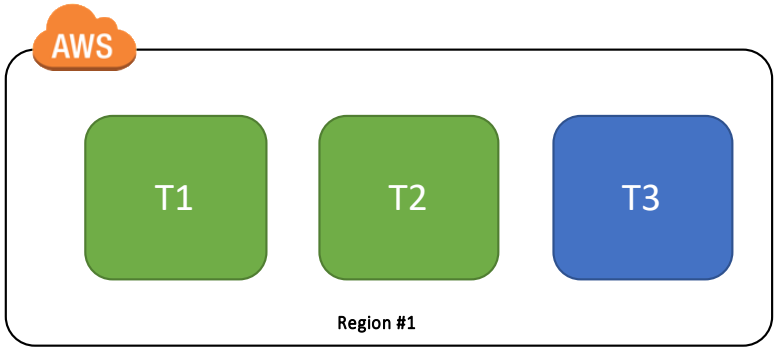
Keep on expanding …

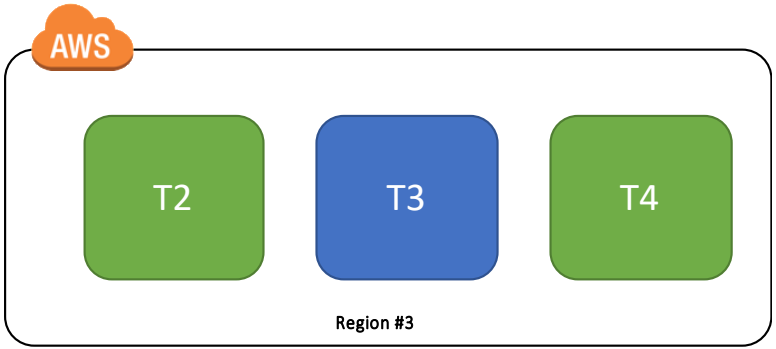


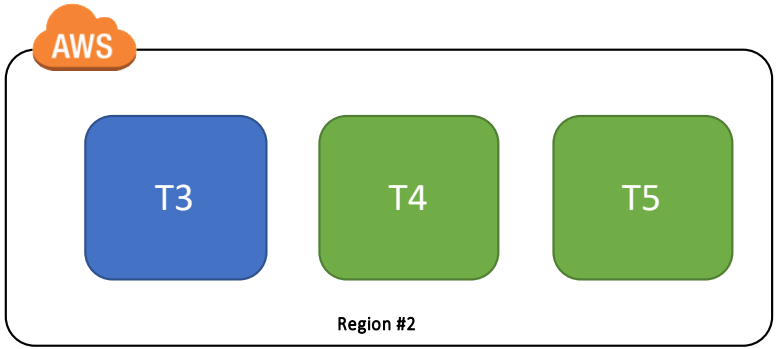




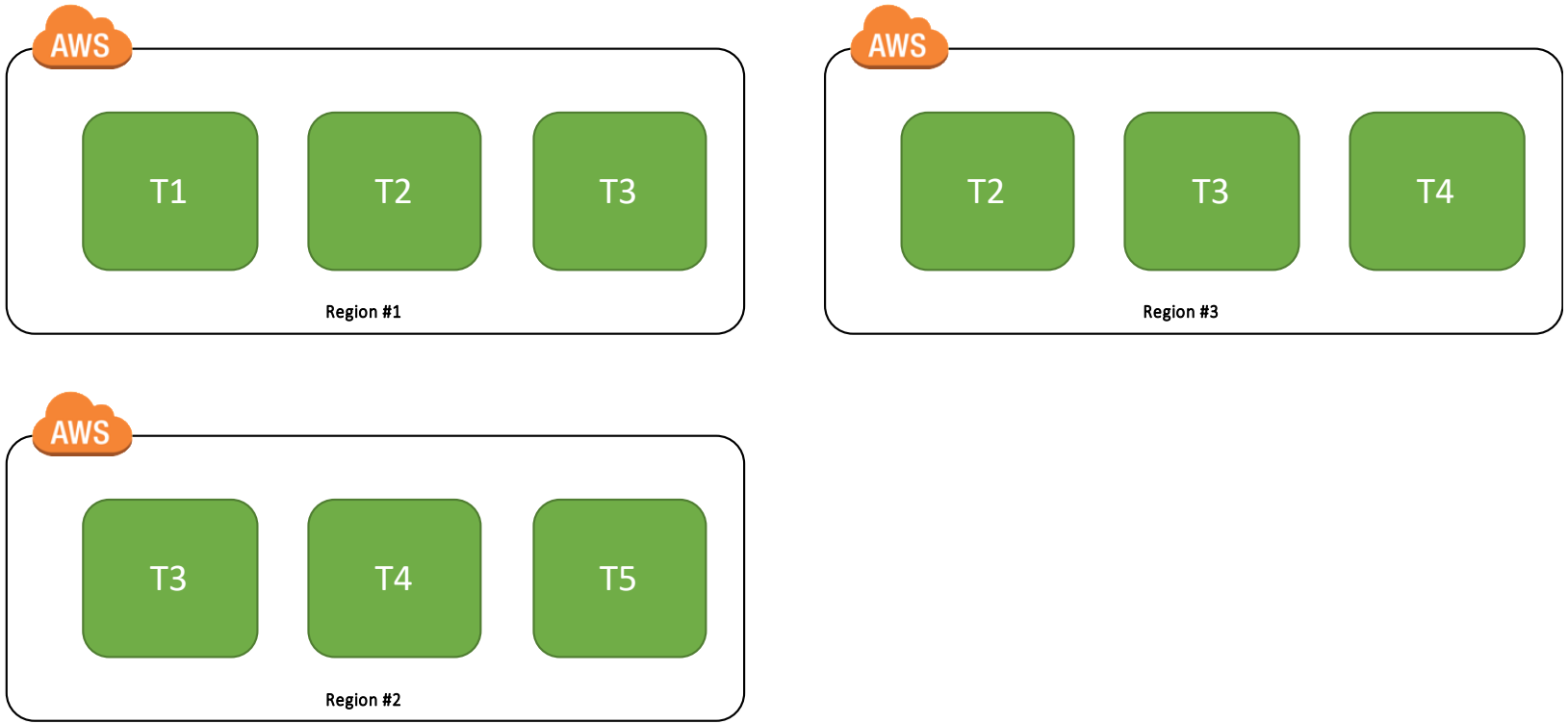
… and again …





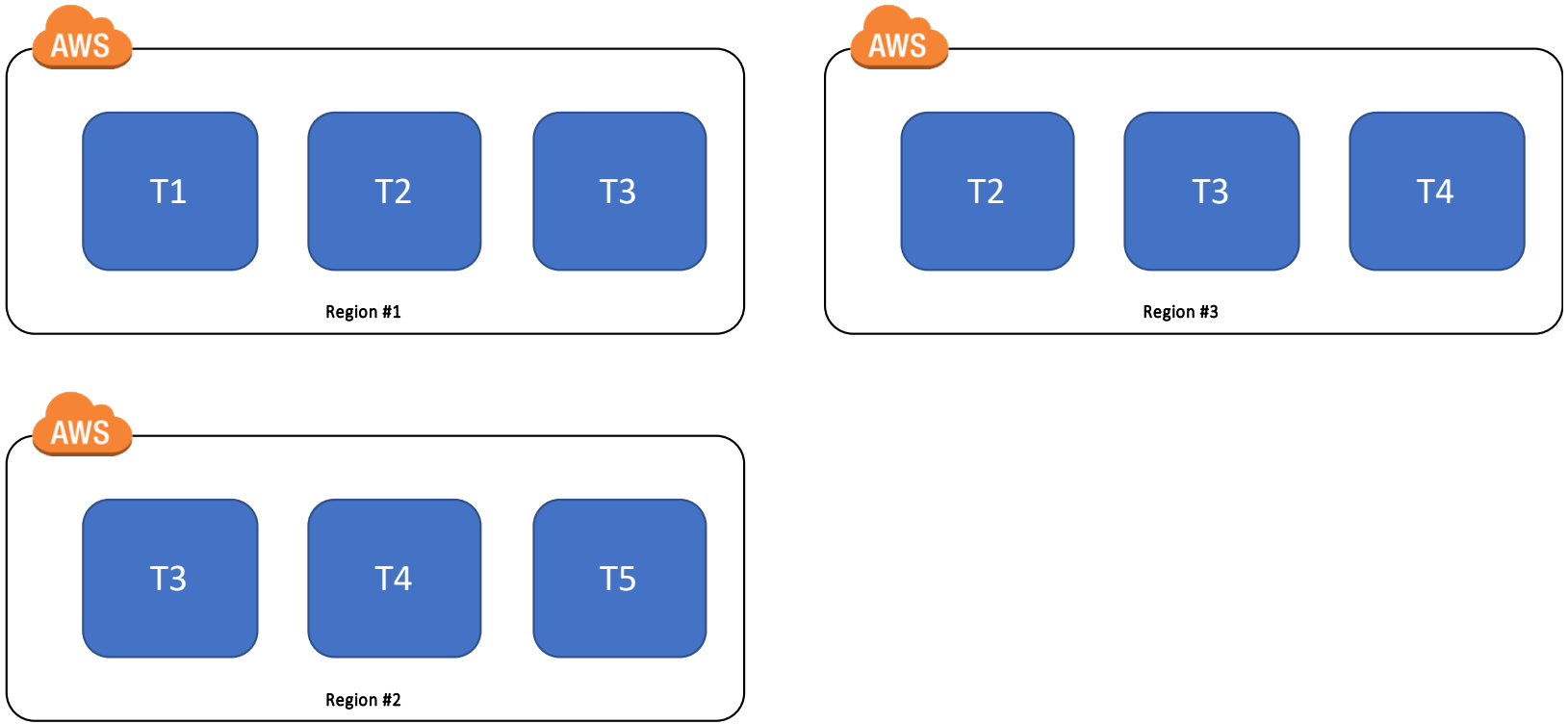


… and again

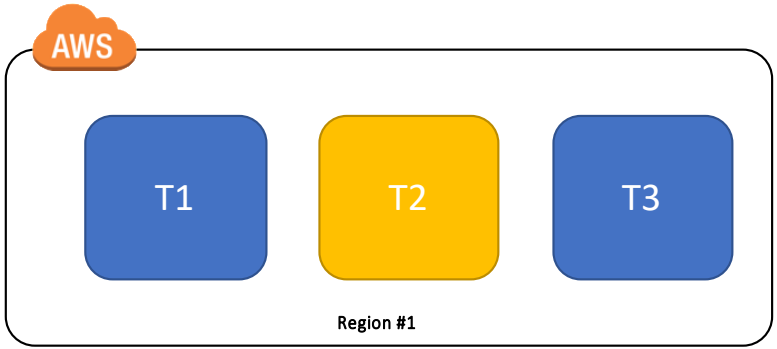


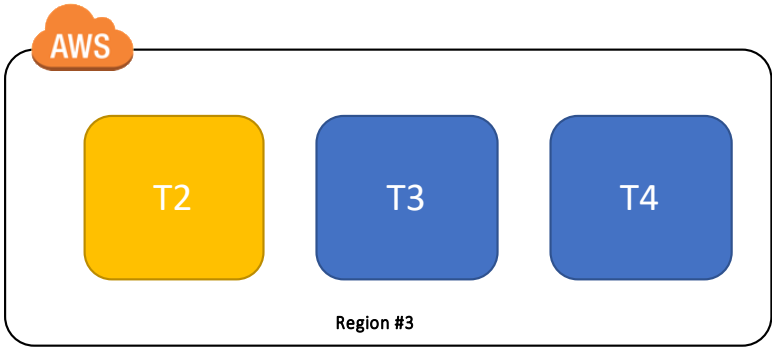
A/B Testing

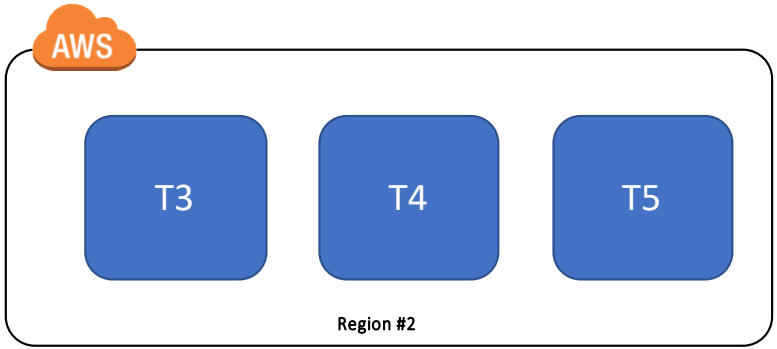
A/B Testing



A/B Testing (test phase)

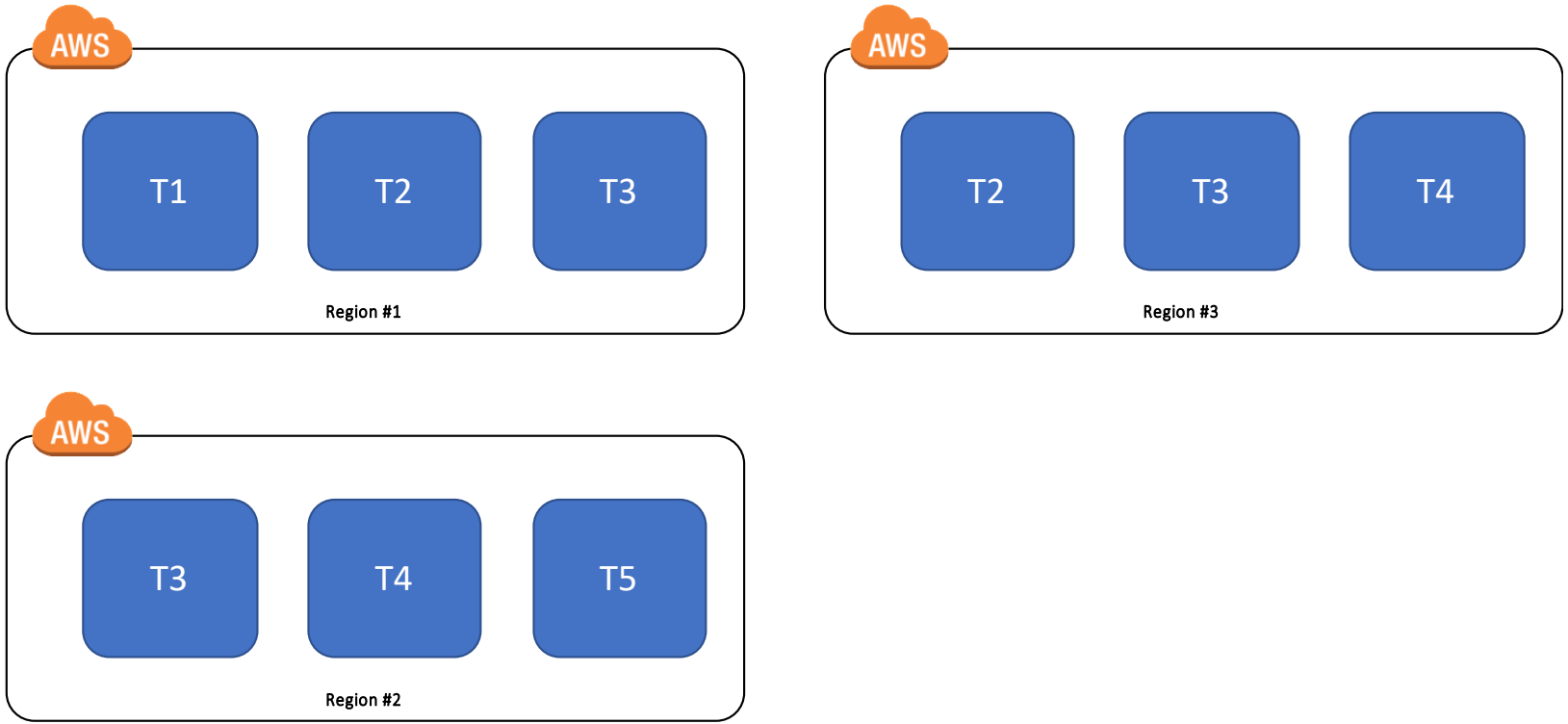






A/B Testing (final phase)

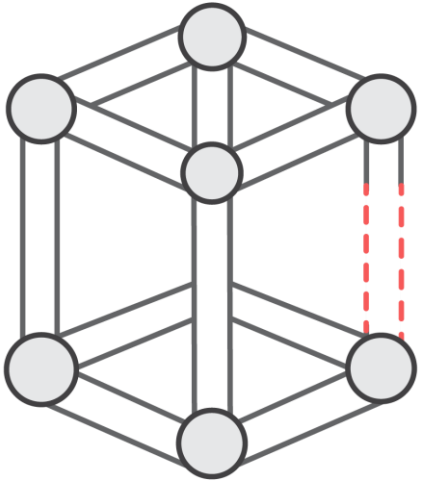
A/B Testing (rollback)



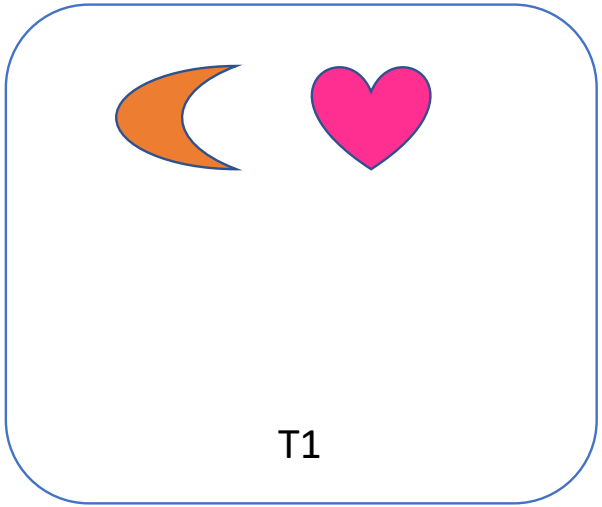
Migrating customers

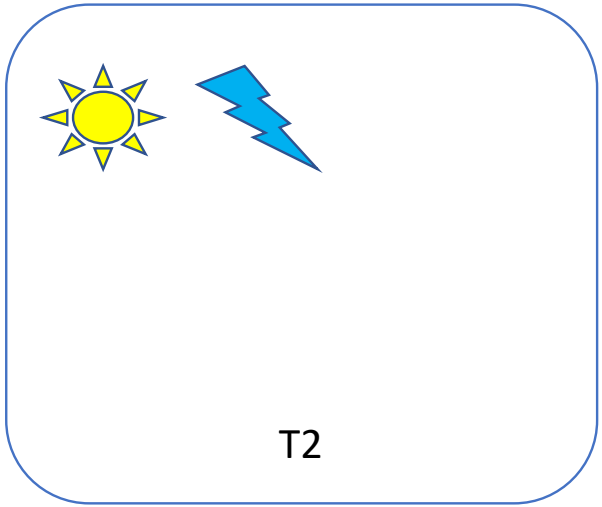
Why migration?

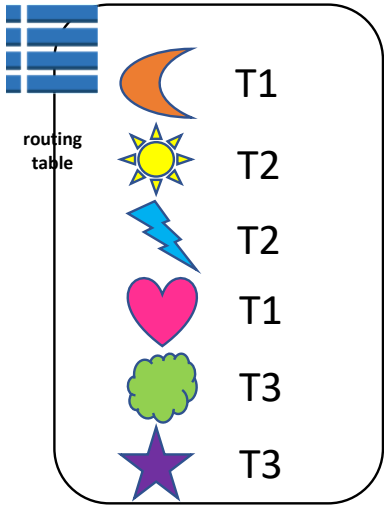
- For technical reasons
  - Denial of service
  - Failure management
- For business reasons
  - Customer will change during their lifecycle
  - Free tier → S/M/L/XL customers



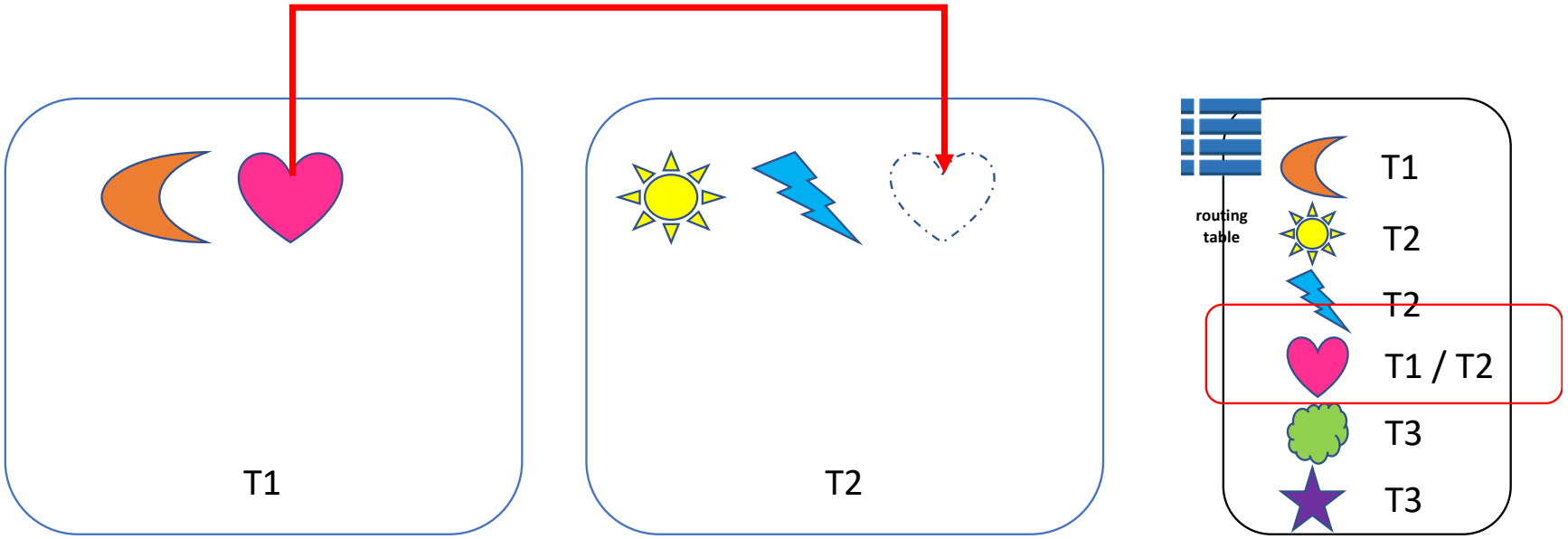
Migrate a customer between cells



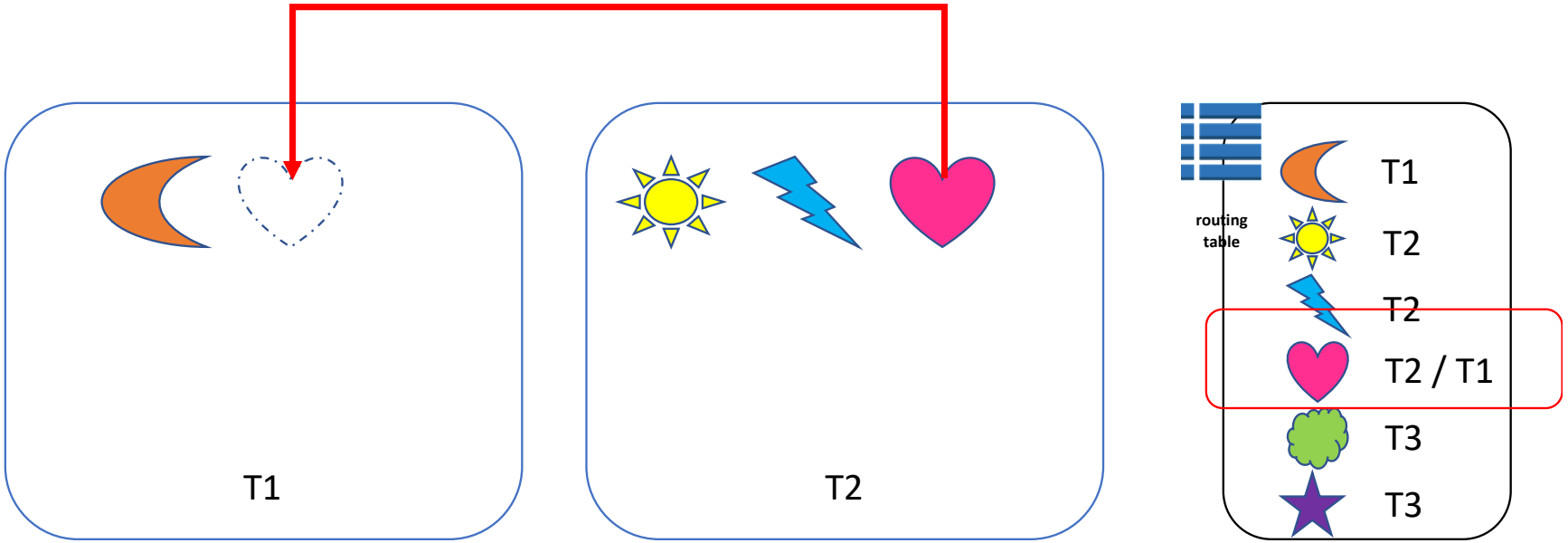




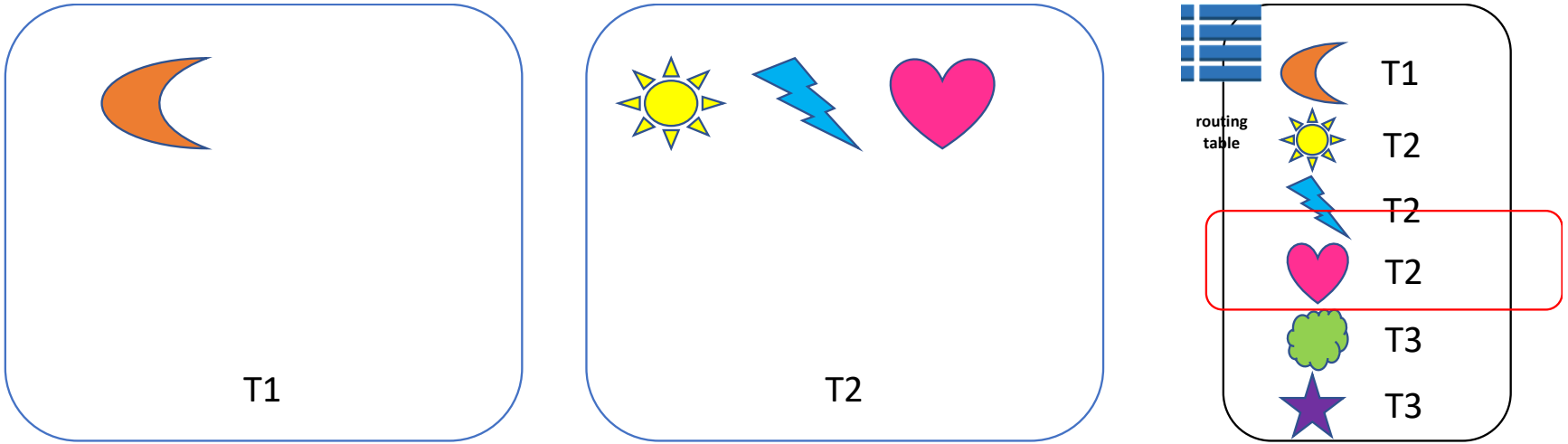
Migrate a customer between cells



Migrate a customer between cells

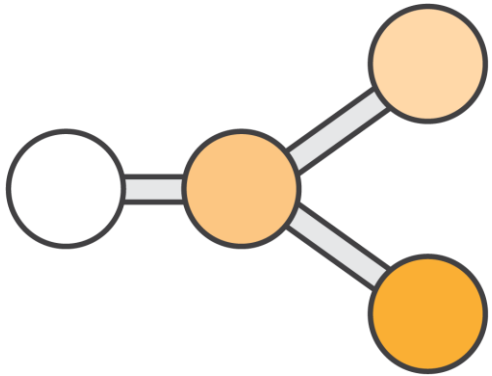


Migrate a customer between cells



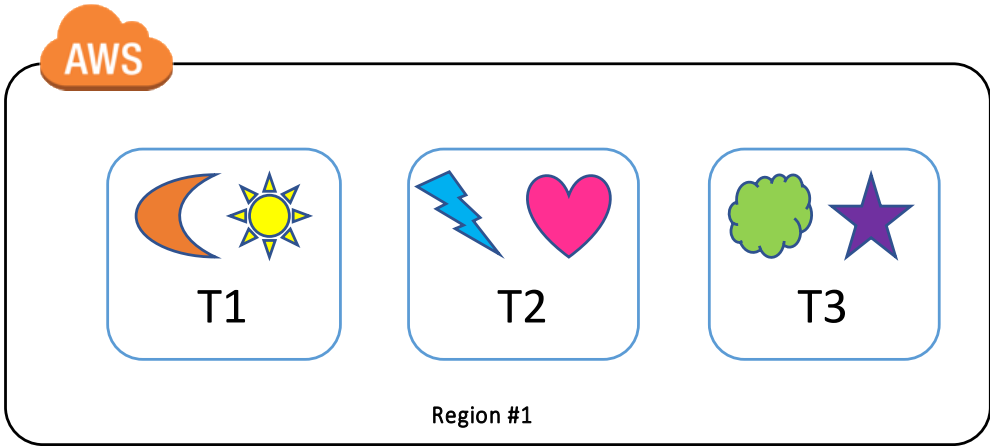
Key points for migration

- Migration can be within or across regions / accounts
- Migration happens as a backend process
- Some customers will exist in multiple cells

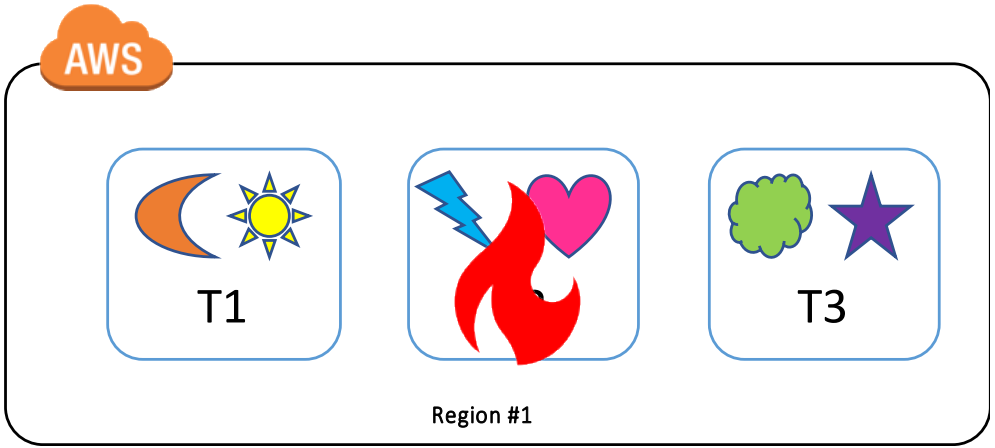


Handling failures

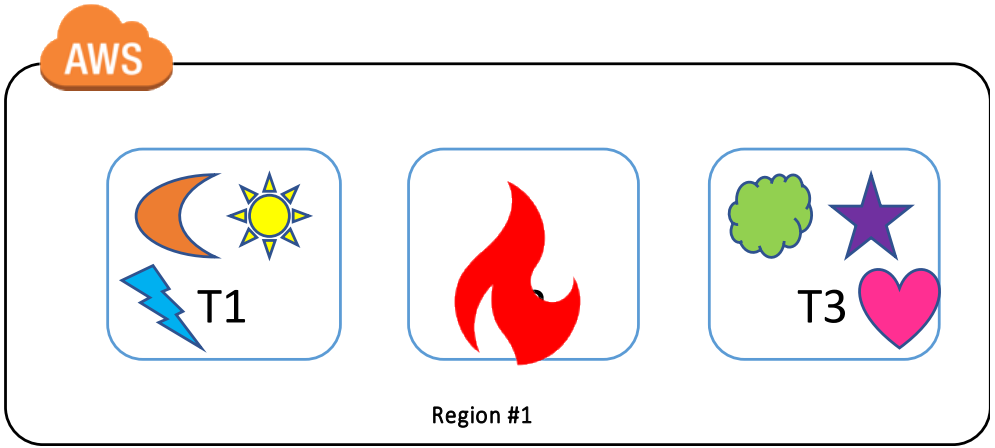
Handling failure



Handling failure

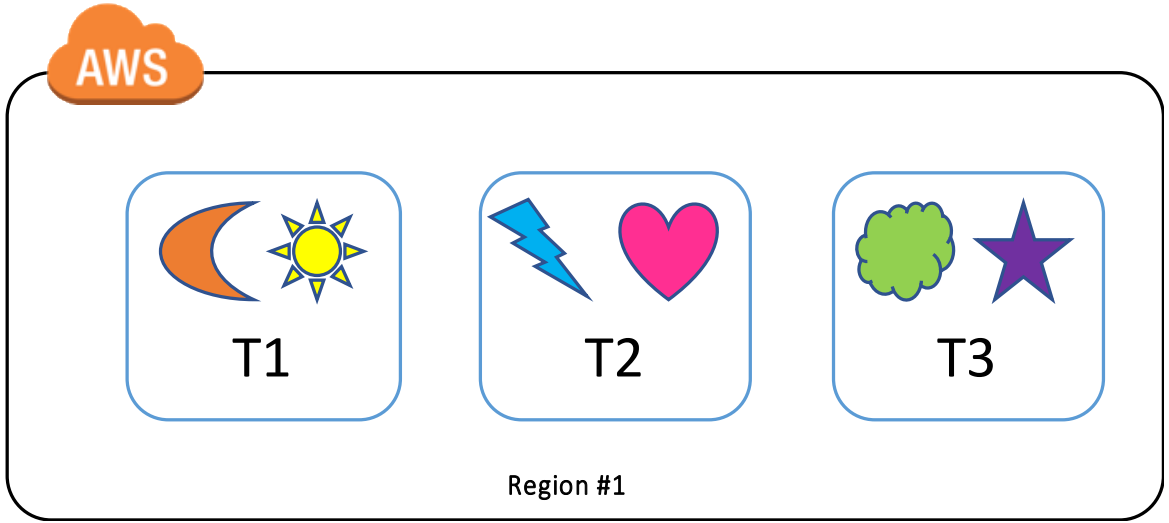


Recovering from Failure

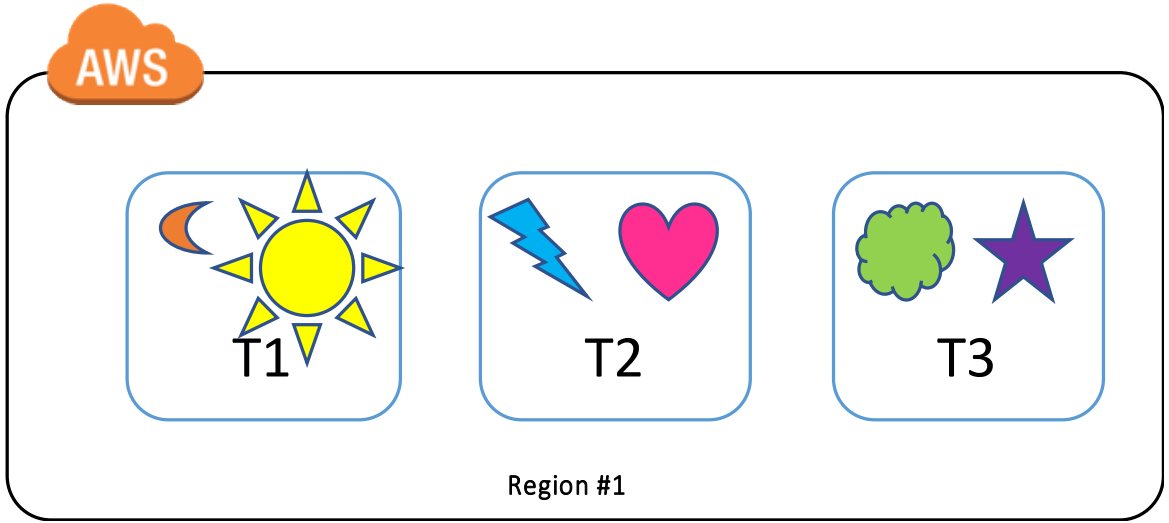


Balancing load

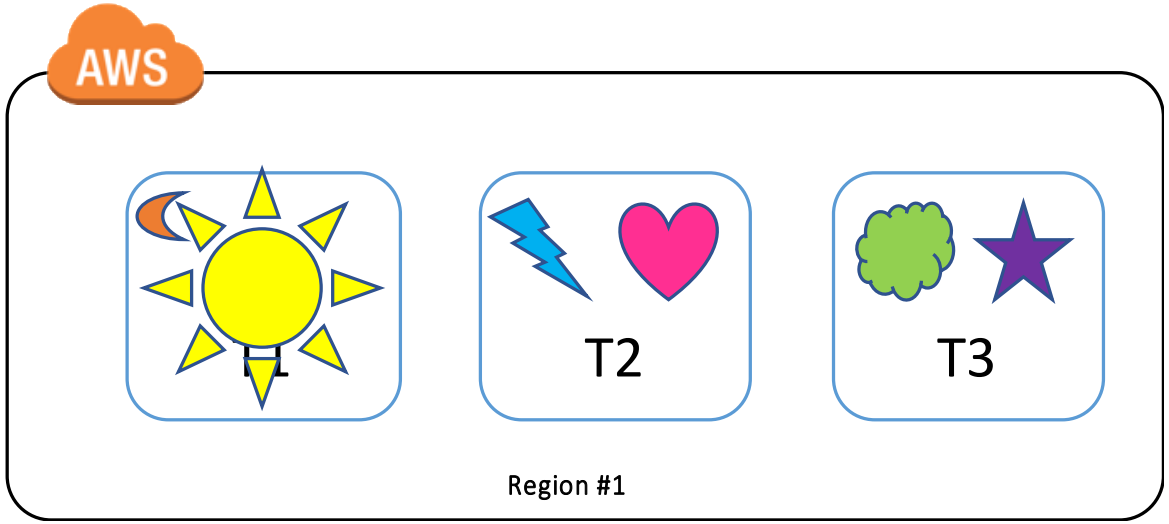
Balancing load



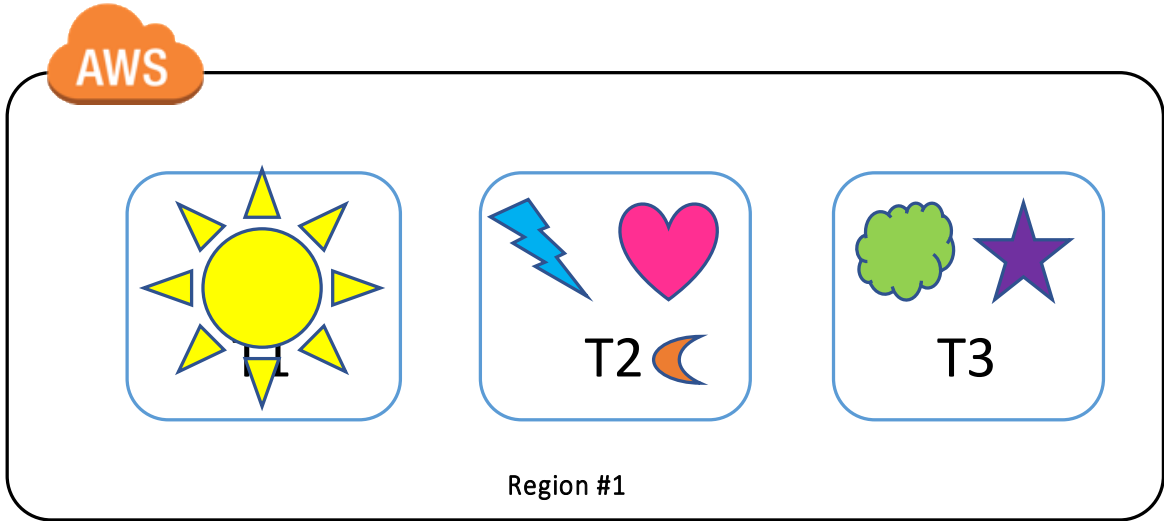
Balancing load



Balancing load



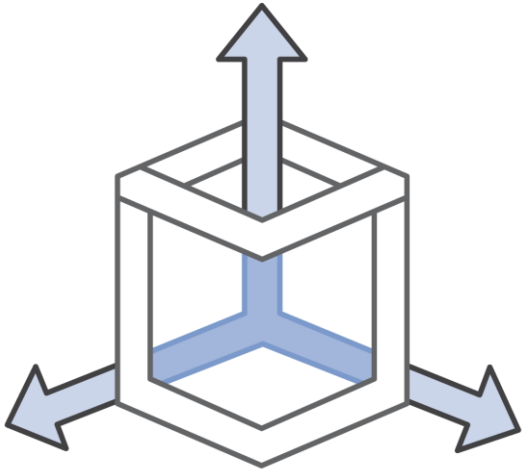
Balancing load



Designing cells

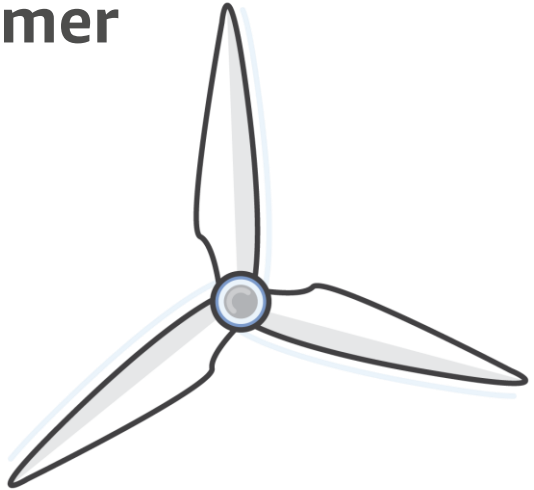
Key points for designing cells

- How small is too small?
- How big is too big?
- What justifies the creation of a cell?
- Should all services have the same cell design?



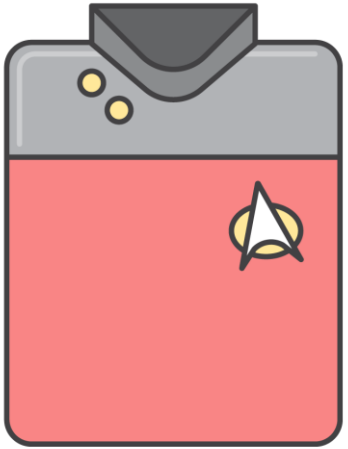
How to get to this design

- First step has to be easy: 1 cell per customer
- Automate deployment
- Work on routing and migration
- Update your services
- Measure, tune, repeat



What do we gain?

- Controlling the growth of your environment
- Agility, adaptability and performance
- Reliability
- Cost management



Services making it easier

Amazon Route 53

Amazon API Gateway

AWS Lambda

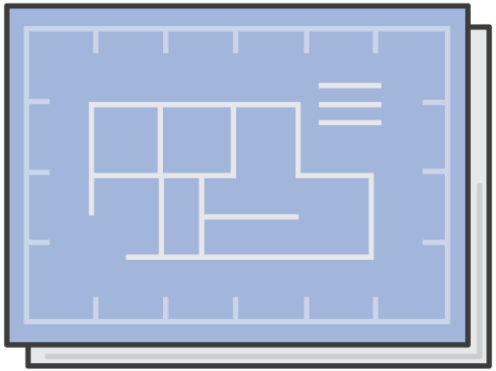
Amazon RDS

AWS Organizations

AWS CodePipeline

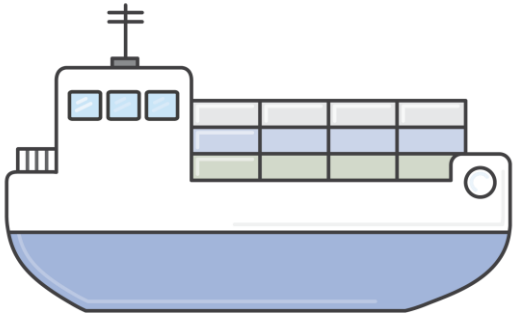
AWS CloudTrail

AWS Config



There’s more to it

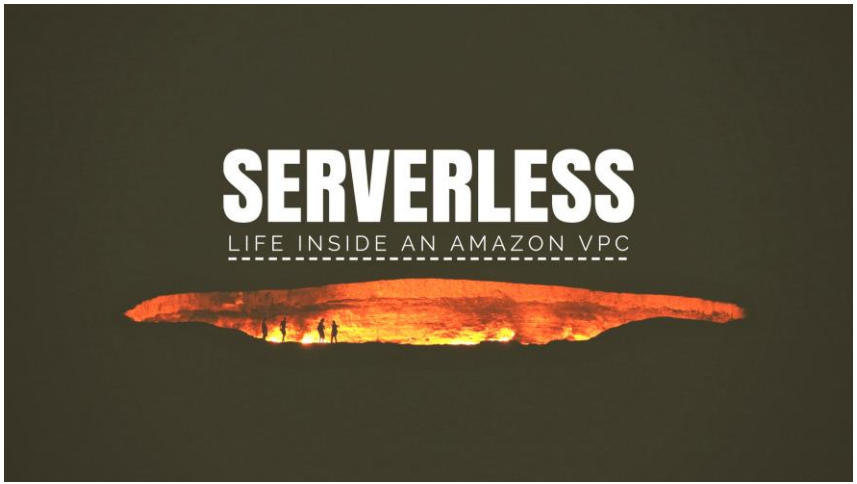
- Networking
- Encryption
- Cost optimization
- Observability



AWS Well-Architected

References

https://bit.ly/2J7NvBT



https://bit.ly/2sv21cu

aws SUMMIT

Thank you!