Aurora Global Database Design Patterns for HA/DR

Shayon Sanyal
Sr Database Specialist SA
AWS

Agenda

Amazon Aurora: Quick recap

Aurora Global database: Overview

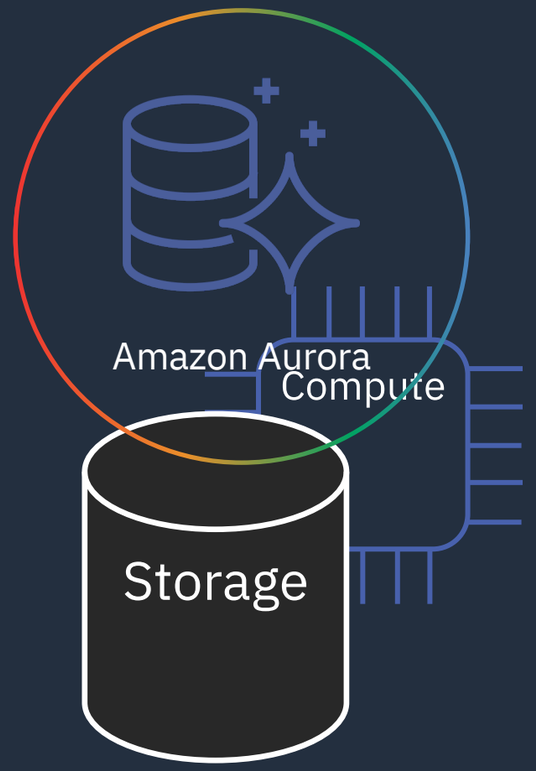
Use cases

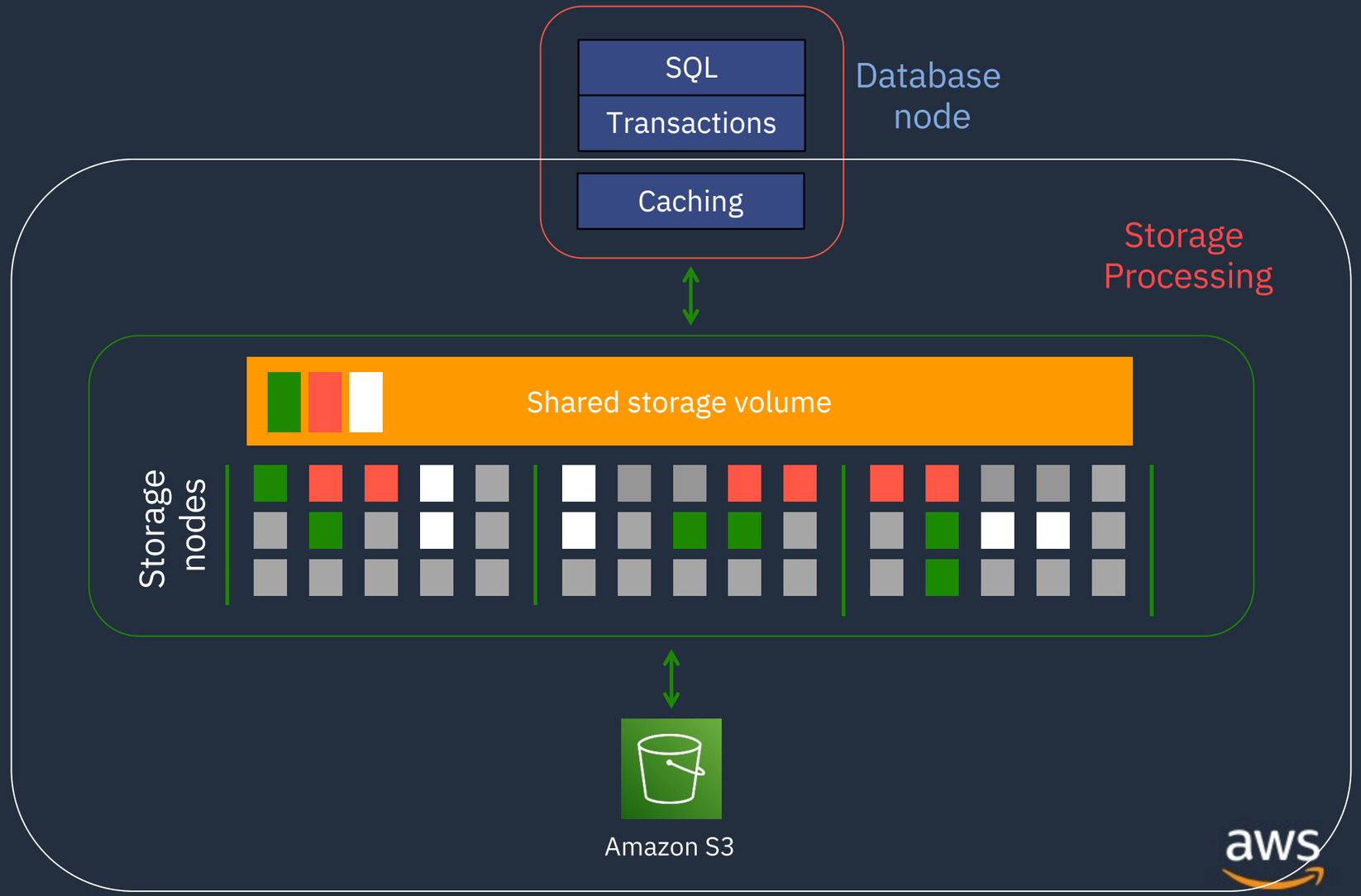
Design patterns

Resources

Amazon Aurora: Quick recap

Aurora decouples storage and query processing





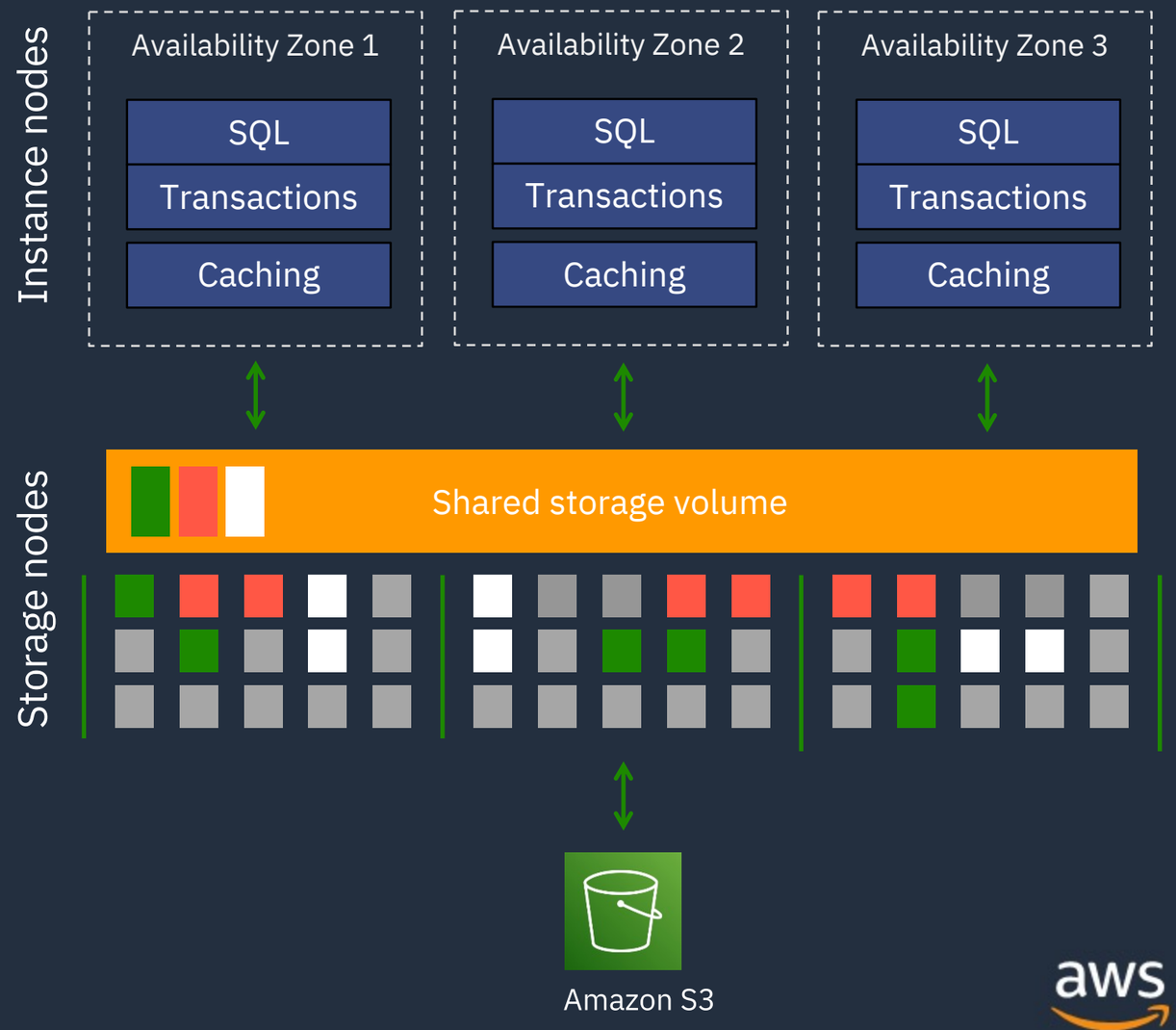
Scale-out, distributed storage processing architecture

Purpose-built, log-structured distributed storage system designed for databases

Storage volume is striped across hundreds of storage nodes distributed over three different Availability Zones

Six copies of data, two copies in each Availability Zone to protect against AZ+1 failures

Data is written in 10 GB protection groups, growing automatically when needed up to 128 TB



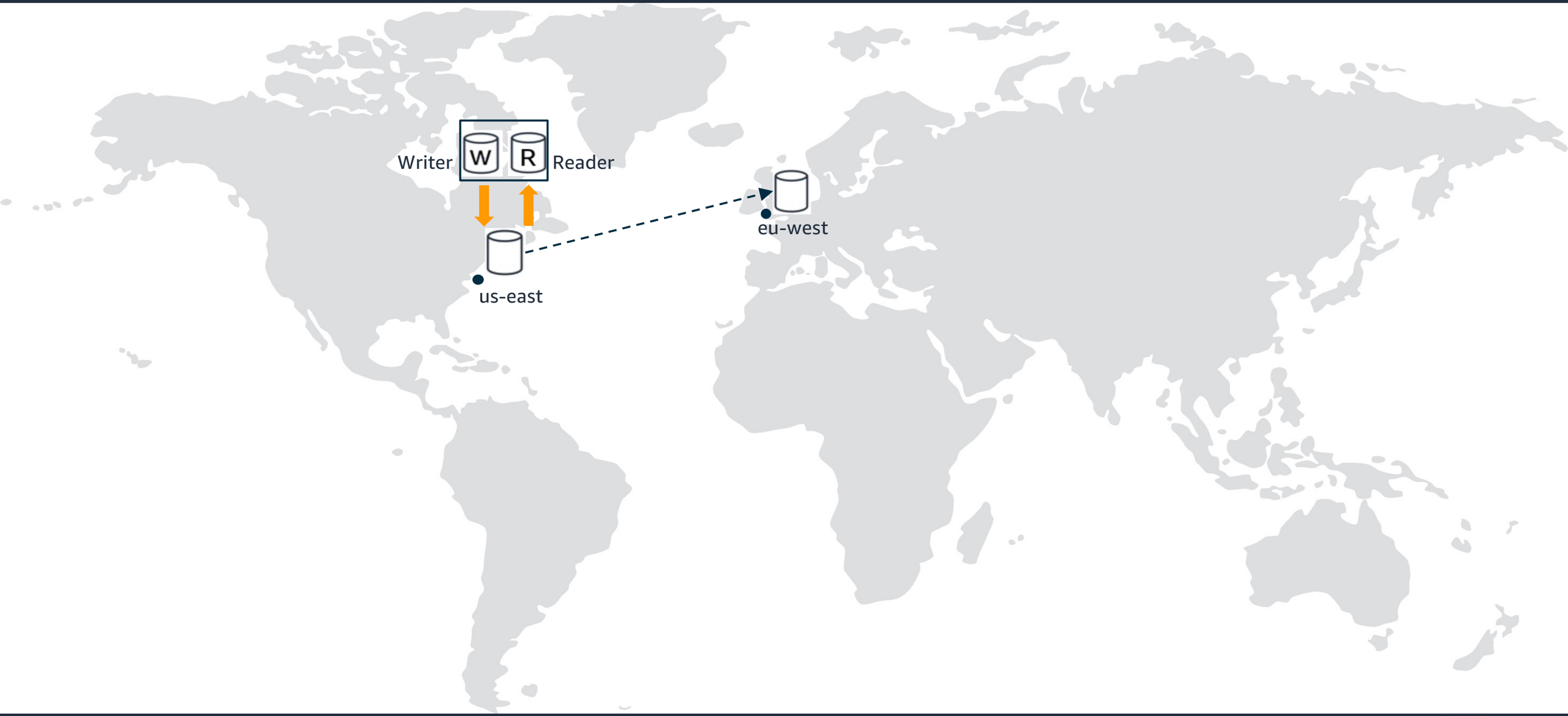
Amazon Aurora Global database: Overview

Use cases

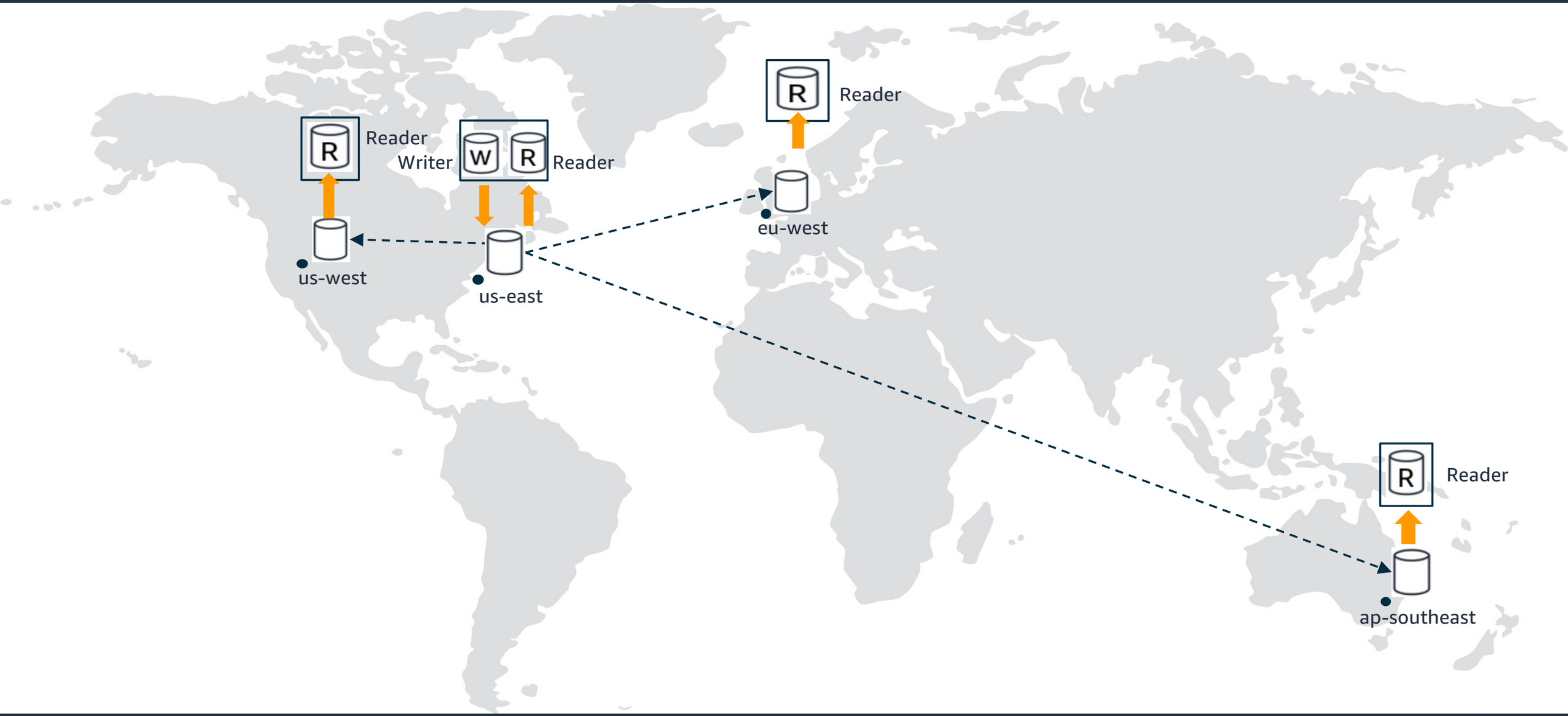
- **Disaster recovery** – promote remote databases to a primary for faster recovery in the event of a disaster
- **Data locality** – bring data closer to users in different Regions to enable faster reads

https://aws.amazon.com/solutions/case-studies/smartnews-2021/

Fast cross-Region disaster recovery

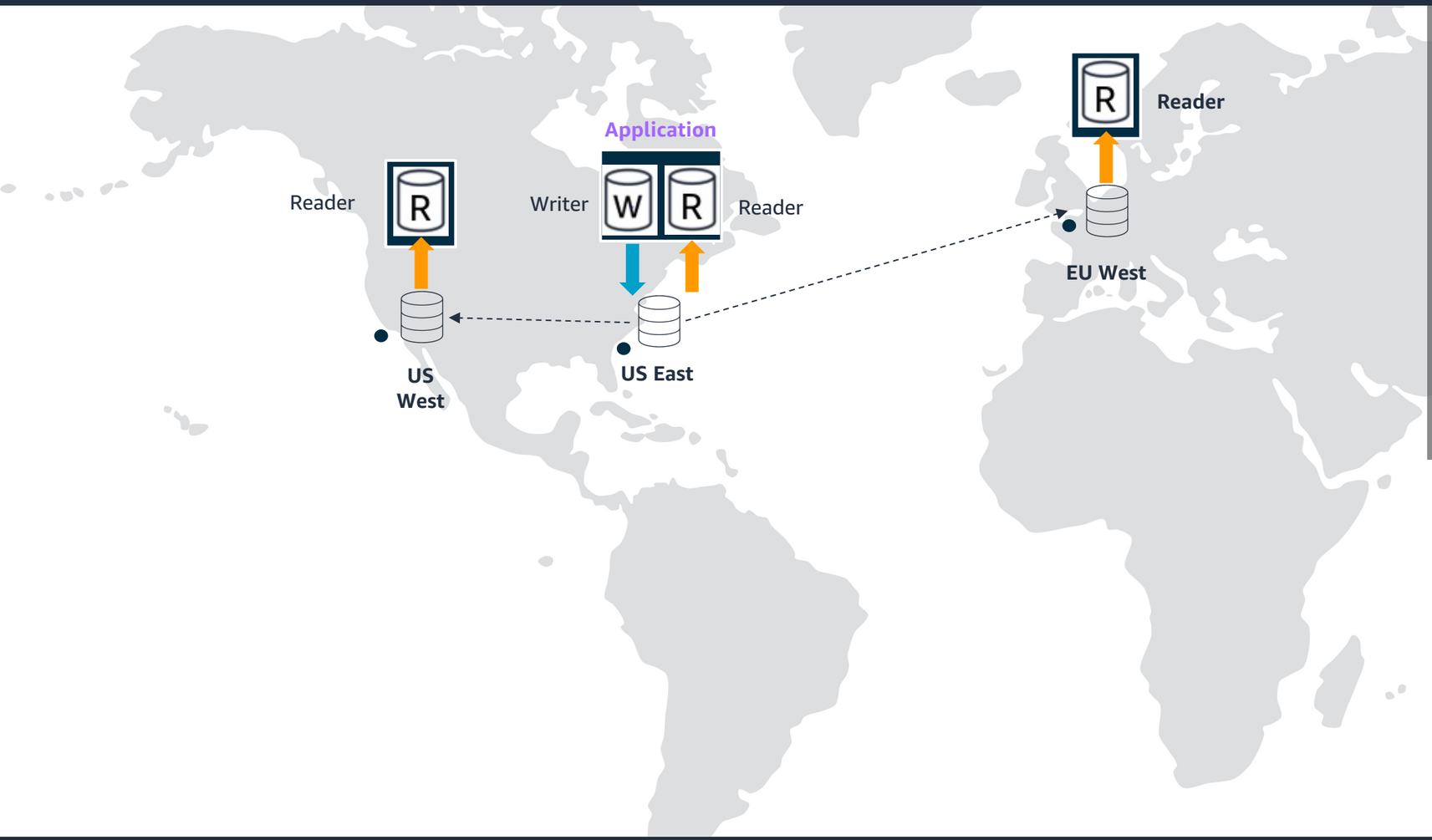


Global reads with low-replication latency



Amazon Aurora Global Database

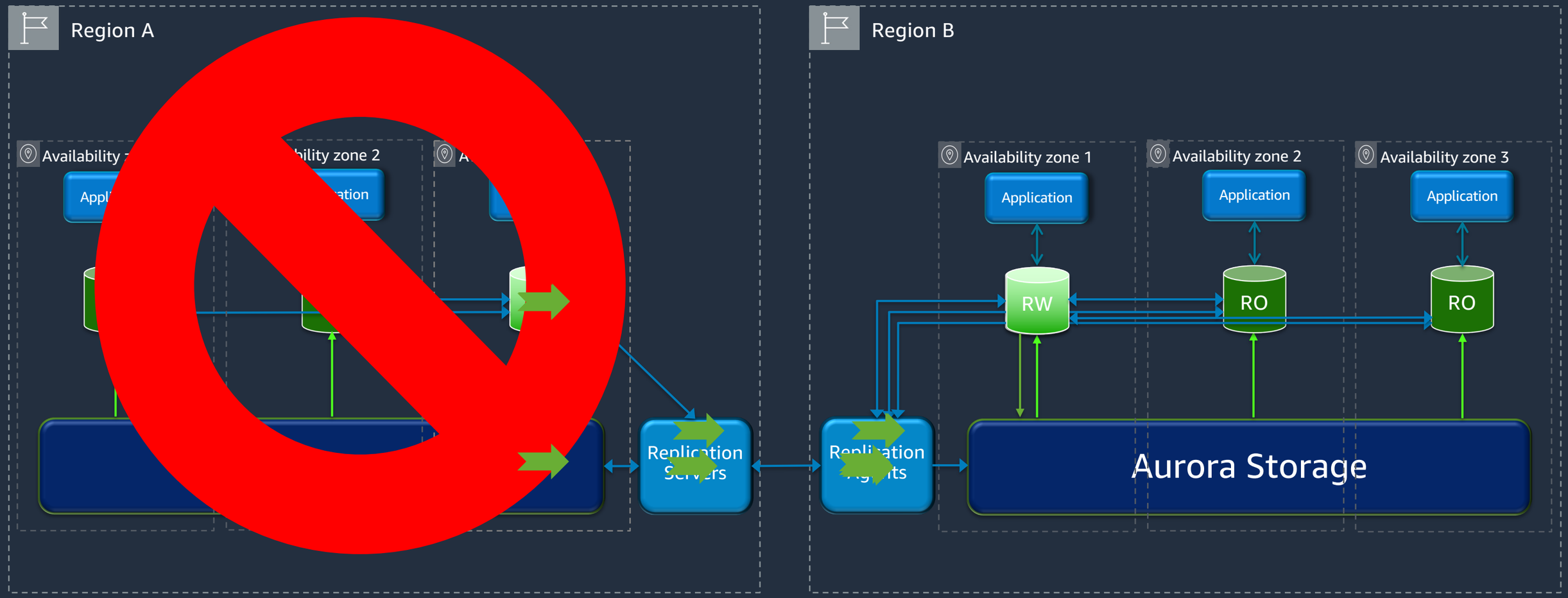
Faster disaster recovery and enhanced data locality



Architecture:

- Physical, log-based asynchronous replication
- Optimized replication service for data transport
- Using AWS backbone network
- Multiple encrypted connections reduce jitter
- Up to five secondary regions

Amazon Aurora Global Database



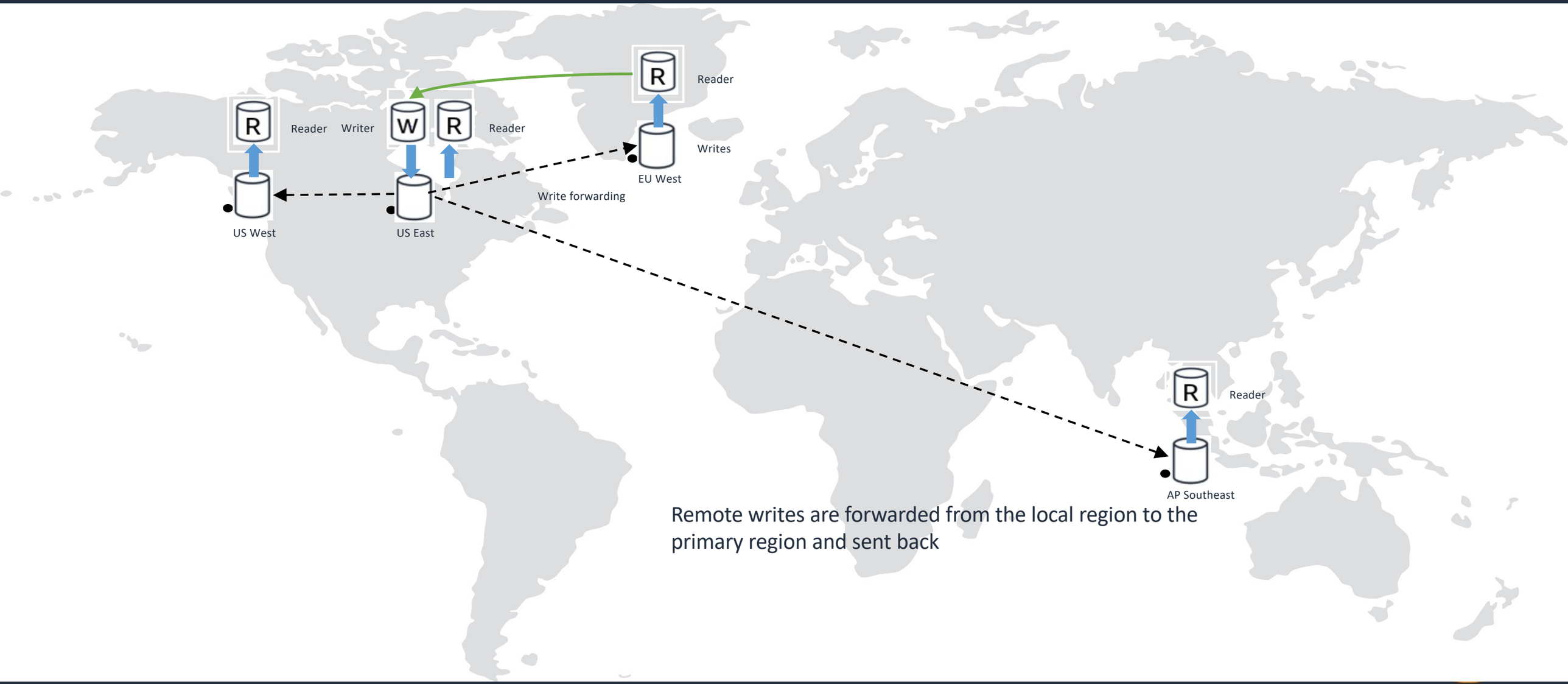
High throughput: Up to 200K writes/sec – negligible performance impact

Low replica lag: Typically < 1 sec cross-country replica lag under heavy load

Fast recovery: < 1 min to accept full read-write workloads after region failure

Use case: Distributed Multi-Region Apps want region local access for read/write

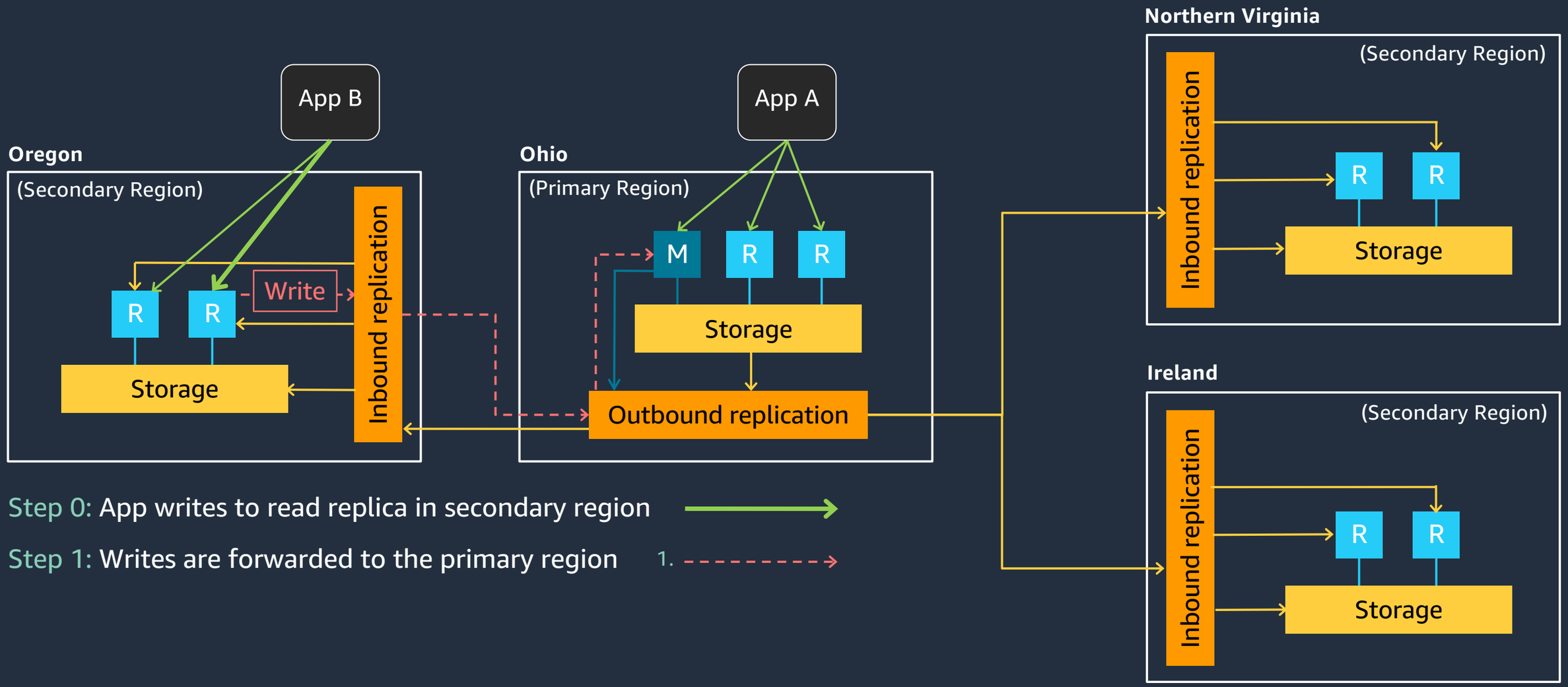
Distributed Multi-Region Apps want region local access for read/write



Remote writes are forwarded from the local region to the primary region and sent back

Design pattern: Write Forwarding with Global Read Replicas

Readers in secondary DB clusters accept writes and forward them to the primary DB cluster writer instance



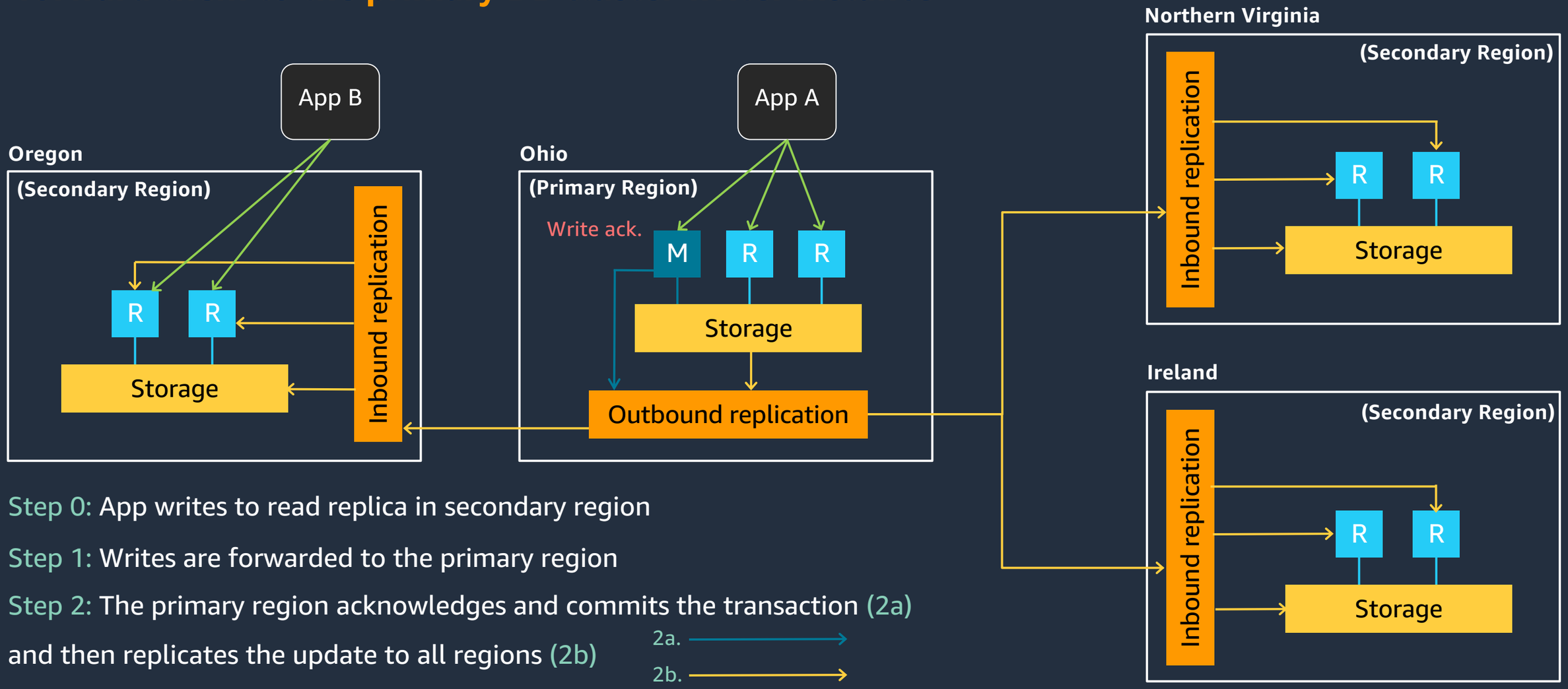
Step 0: App writes to read replica in secondary region

Step 1: Writes are forwarded to the primary region 1.

*Currently available for Aurora MySQL

Design pattern: Write Forwarding with Global Read Replicas

Readers in secondary DB clusters accept writes and forward them to the primary DB cluster writer instance



Step 0: App writes to read replica in secondary region

Step 1: Writes are forwarded to the primary region

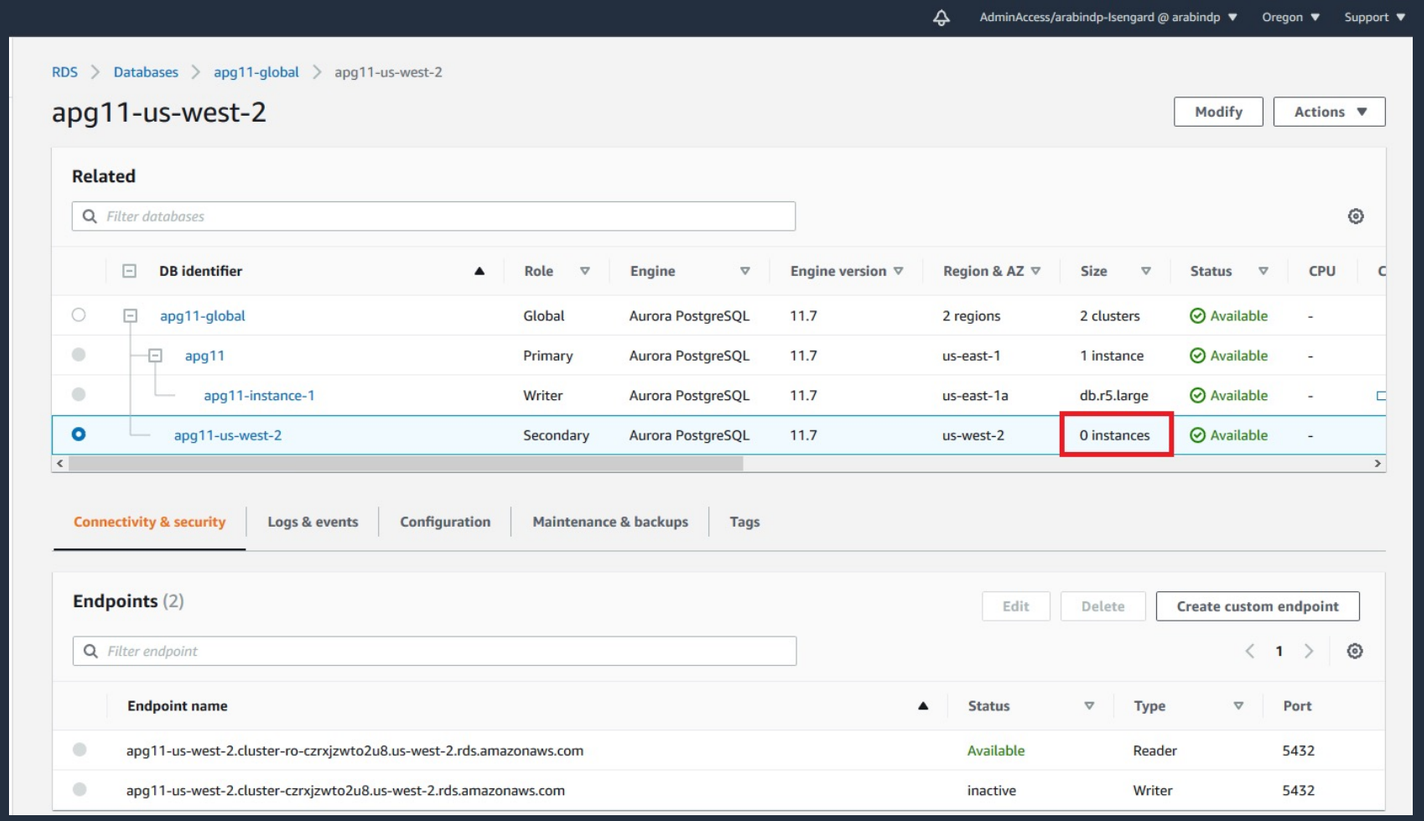
Step 2: The primary region acknowledges and commits the transaction (2a) and then replicates the update to all regions (2b)

*Currently available for Aurora MySQL

Use case: Customers want to save costs on DR

Design pattern: Aurora Global Database Headless Clusters

- Aurora Cluster in the secondary region without any replicas attached to storage
- Secondary's storage volume is kept in-sync with the primary DB cluster
- Monitor replication lag using CloudWatch console
- Add a replica before failing over
- Saves compute charge at the cost of higher RTO



Use case: Customers want to cap maximum RPO loss to a limit

Customers want to cap maximum RPO loss to a limit

- Managing recovery point objective (RPO)
- Global database replication is asynchronous
- Replicas typically lag primary by <1 second
- Data at risk in case of geo-disaster = replication lag
- What if a failure (e.g., network) causes replication to fall further behind?
- Application needs protection from replica lag that is too high

Design pattern: Managed RPO

For applications with **critical RPO requirements**

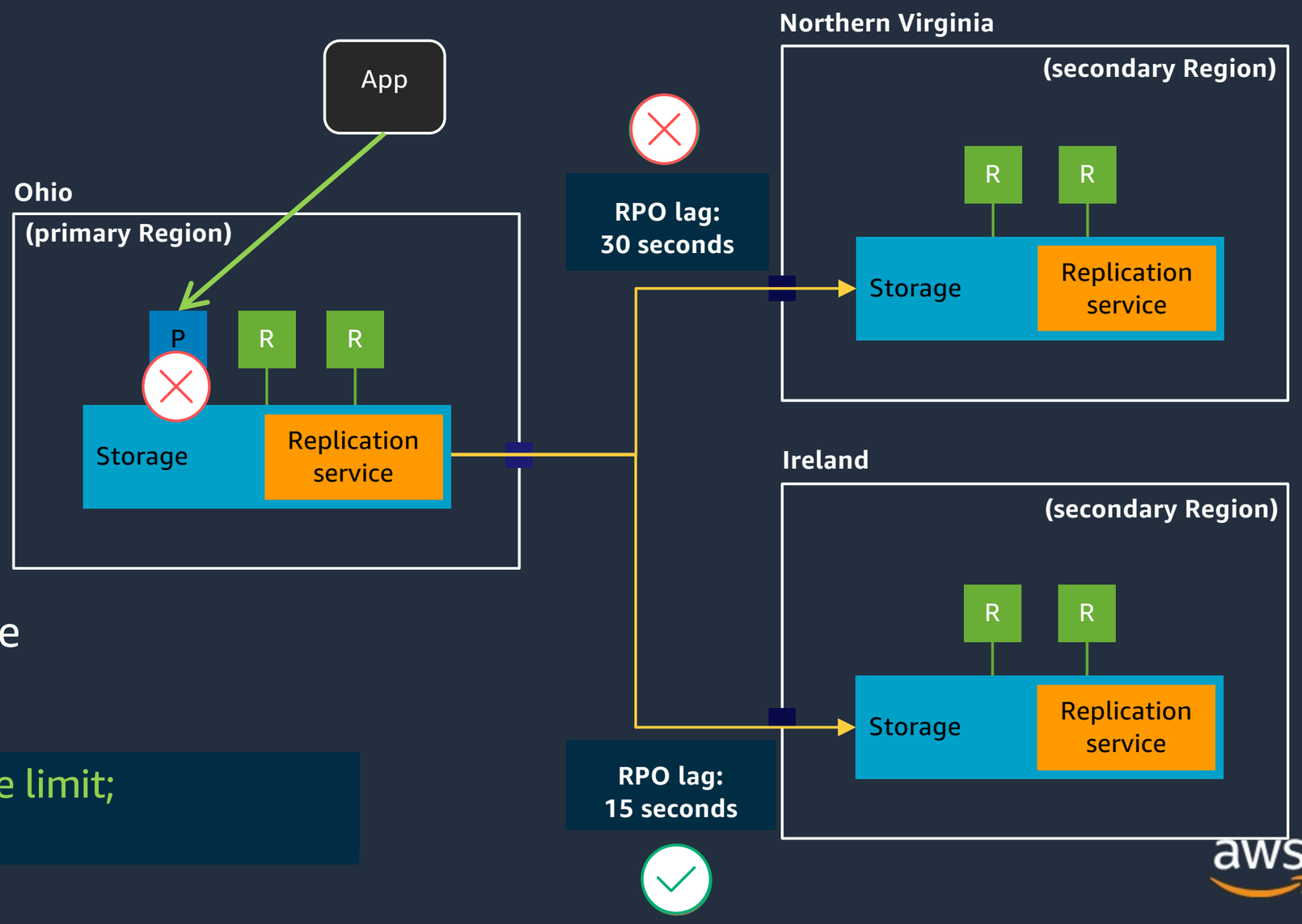
You define the maximum RPO that you allow

If RPO lag in **all** secondary Regions exceeds the limit, Aurora pauses writes until **at least one** Region catches up

Let's see an example where we set RPO = 20 seconds

→ Lag is within the limit

→ Lag in Ireland is back under the limit; writing has been resumed



Use case: Customers want to meet DR test regulatory compliance requirements

Customers want to meet DR test regulatory compliance requirements

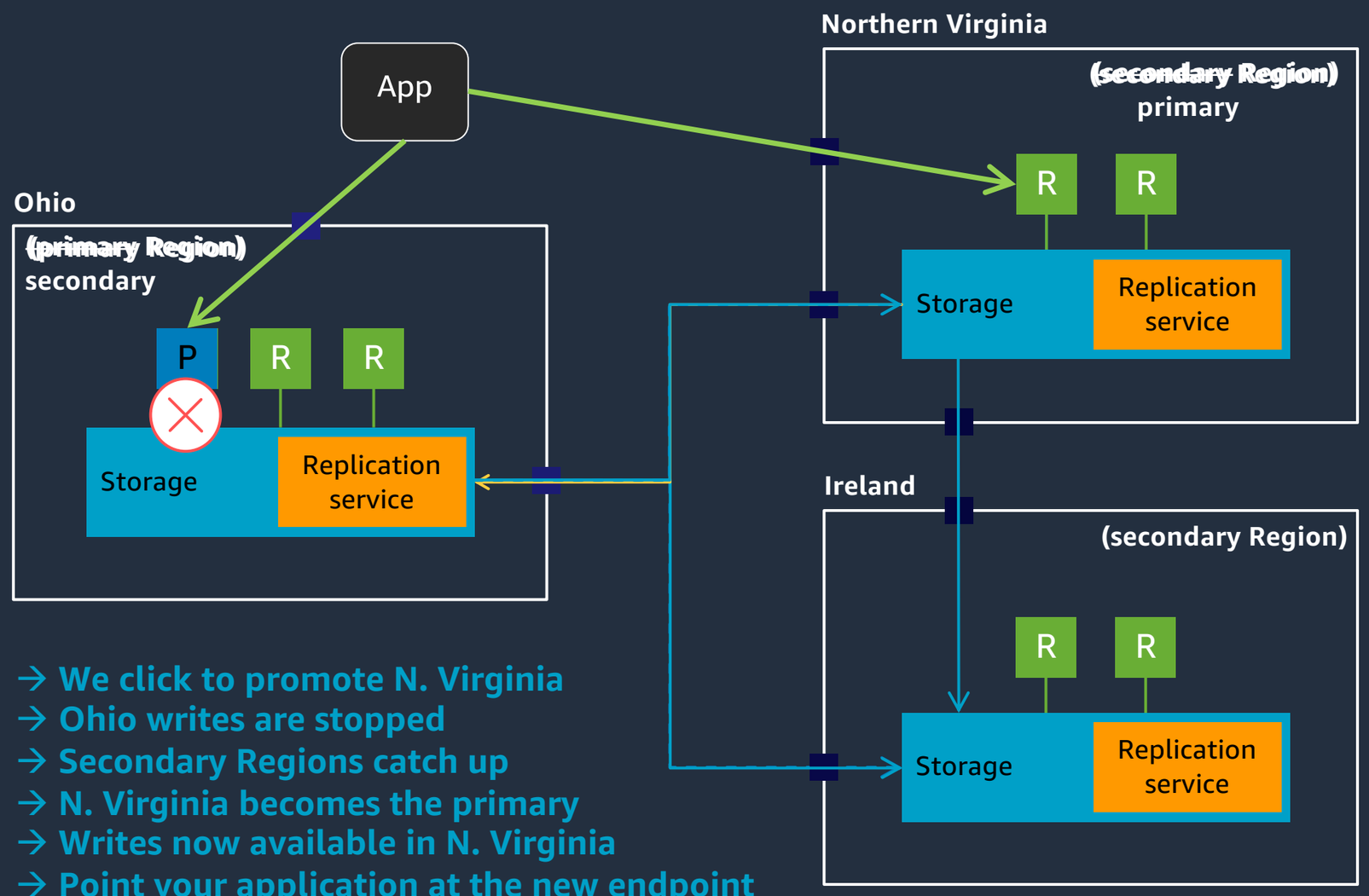
- An easy way to test your disaster recovery (DR) setup
- An easy way to relocate your primary Region
- Designed to be used on a healthy Aurora global database cluster
- Promote a secondary Region to be the primary
  - In a completely automated manner
  - Without destroying your global database topology
  - With RPO = 0, writes are stopped until new primary catches up
  - Without having to replicate previous data
  - Without interrupting your DR capability
- Point your application to the new primary and you're done

Design pattern: Managed Planned Failover

In this example, we'll promote N. Virginia to primary

RPO=0; writes are stopped until new primary catches up

RTO directly proportional to *AuroraGlobalDBReplicationLag* metric value for all the secondaries



→ We click to promote N. Virginia
→ Ohio writes are stopped
→ Secondary Regions catch up
→ N. Virginia becomes the primary
→ Writes now available in N. Virginia
→ Point your application at the new endpoint and go

Use case: Customers need to recover rapidly on region failure

Design pattern: manual unplanned failover

- Used to recover from an *unplanned* outage in an AWS region
- RPO depends on the *AuroraGlobalDBReplicationLag* metric value at the time of failure
- RTO depends on how quickly you can perform the manual failover related tasks
- Detach & Promote a secondary Region to be the primary
  - Stop writes to the Primary
  - Identify a secondary region to use as the new primary DB cluster based on least replication lag
  - Detach & promote the secondary region Aurora cluster
- Point your application to the new primary Aurora cluster
- Add secondary AWS regions as needed to re-create the global database topology

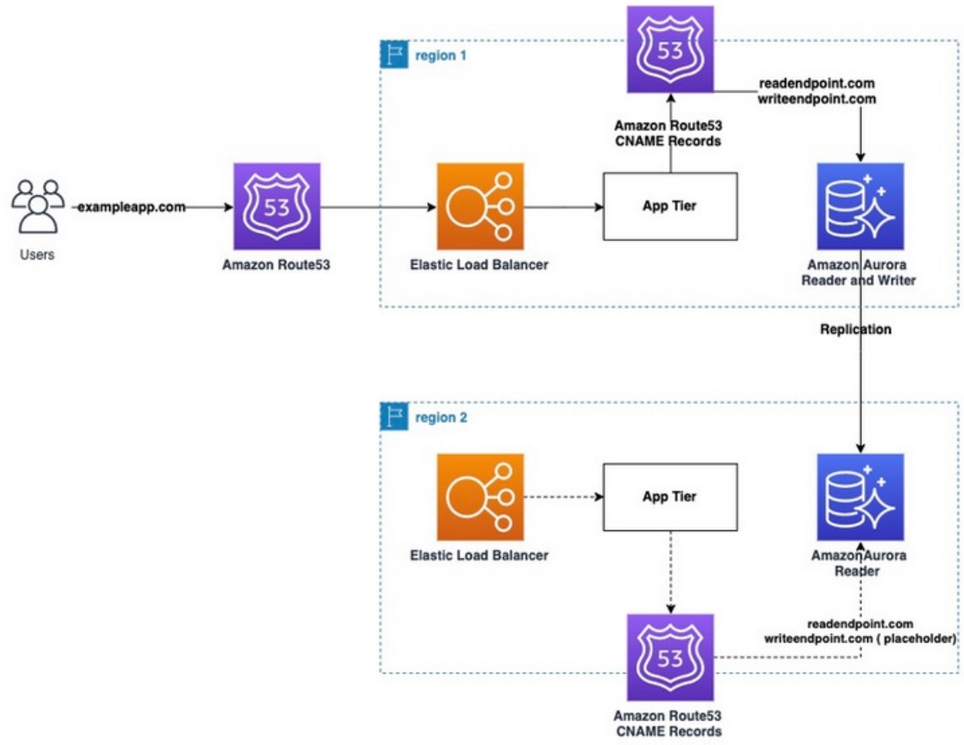
Use case: Customers want to automate tasks on DR failover to reduce RTO

Design pattern: Endpoint update automation

AWS Database Blog

Deploy multi-Region Amazon Aurora applications with a failover blueprint

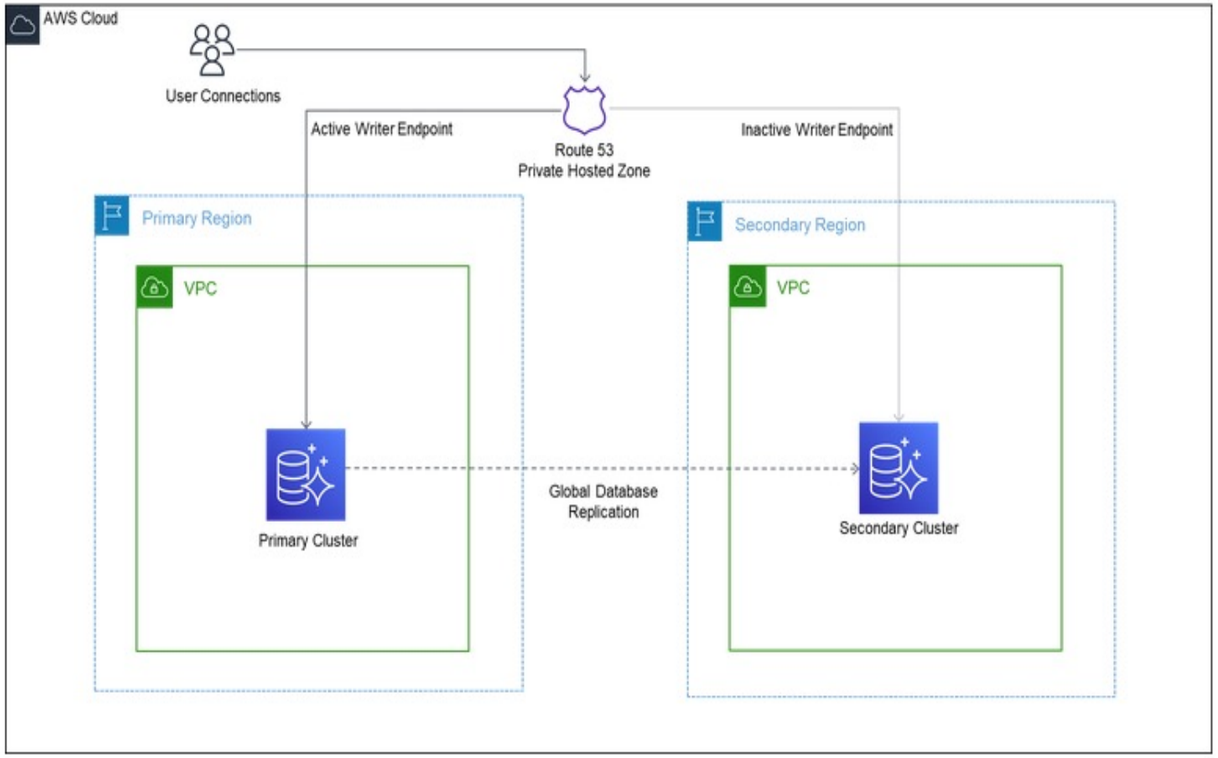
by Vivek Kumar and Jigna Gandhi | on 23 JUN 2021 | in Amazon Aurora, Amazon Route 53, AWS Lambda | Permalink | Comments | Share



AWS Database Blog

Automate Amazon Aurora Global Database endpoint management

by Aditya Samant | on 22 SEP 2021 | in Amazon Aurora, Infrastructure & Automation | Permalink | Comments | Share



https://aws.amazon.com/blogs/database/deploy-multi-region-amazon-aurora-applications-with-a-failover-blueprint/

https://aws.amazon.com/blogs/database/automate-amazon-aurora-global-database-endpoint-management/

Use case: Customers want to automate provisioning and failover

Design pattern: Terraform automation for Global Database

AWS Quick Starts | Amazon Connect integrations | Amazon EventBridge integrations | FAQs | Resources | Demos

AWS Quick Starts

Automated, gold-standard deployments in the AWS Cloud

Quick Starts are automated reference deployments built by Amazon Web Services (AWS) solutions architects and AWS Partners. Quick Starts help you deploy popular technologies on AWS based on AWS best practices for security and high availability. These accelerators reduce hundreds of manual procedures into just a few steps so that you can build your production environment in minutes and start using it immediately.

Each Quick Start includes AWS CloudFormation templates that automate the deployment and a guide that describes the architecture and provides deployment instructions.

SEE ALSO
For patterns, techniques, and tips for building Quick Starts and automating AWS Cloud DevOps tasks, see the **Infrastructure & Automation blog**.

https://aws.amazon.com › quickstart

terraform

1 (1)

Sort by: Last update (newest - oldest)

DEVELOPER TOOLS

HashiCorp Terraform

Terraform modules on AWS

Built by HashiCorp and AWS

With Terraform modules on Amazon Web Services (AWS), deploy native Terraform resources on the AWS Cloud.

Learn more

Last update
November 2020

Terraform modules on AWS

Building blocks for Terraform-managed resources on AWS

Use Terraform modules on Amazon Web Services (AWS) to deploy native Terraform resources on the AWS Cloud. Terraform modules on AWS are published under an open-source license.

Terraform modules on AWS are available in the Terraform registry on the AWS Integration and Automation namespace page. Use the links provided to access modules in the Terraform registry and source code on GitHub. For module deployment instructions, refer to the README.md file in the GitHub repository.

HashiCorp Terraform

Terraform modules on AWS were developed by HashiCorp Inc. in partnership with AWS. HashiCorp is an AWS Partner.

Available modules:

Terraform Cloud Workspace on AWS
Configure a Terraform organization and workspace on the Amazon Web Services (AWS) Cloud.
View on the registry.
See the source code.

Amazon VPC for Terraform on AWS
Provision Amazon Virtual Private Cloud (Amazon VPC) resources managed by Terraform on the Amazon Web Services (AWS) Cloud.
View on the registry.
See the source code.

Terraform Amazon Aurora
Provision Amazon Aurora managed by Terraform on the Amazon Web Services (AWS) Cloud.
View on the registry.
See the source code.

Consul-Terraform-Sync on AWS
Create a listener rule and target group for an Application Load Balancer.
View on the registry.
See the source code.

Terraform AWS Transit Gateway
Provision AWS Transit Gateway resources managed by Terraform on the Amazon Web Services (AWS) Cloud.
View on the registry.
See the source code.

Terraform AWS Label
Generate consistent label names and tags for Terraform resources.
View on the registry.
See the source code.

Terraform Amazon ECS on AWS Fargate
Deploy Amazon Elastic Container Service (Amazon ECS) on AWS Fargate.
View on the registry.
See the source code.

Terraform Amazon SNS
Deploy a Amazon Simple Notification Service (Amazon SNS) topic.
View on the registry.
See the source code.

Terraform AWS CodeCommit
Deploy AWS CodeCommit to securely host scalable private Git repositories.
View on the registry.
See the source code.

Resources

- AWS Terraform Workshop
- Amazon Aurora Terraform Module
- Aurora PostgreSQL Global Database Immersion Day
- Aurora MySQL Global Database Immersion Day
- Automated endpoint management for Aurora Global Database Managed Planned failover
- Automate endpoint management for Aurora Global Database unplanned failover
- Automate replication of secrets in AWS Secrets Manager across AWS Regions

Thank you!