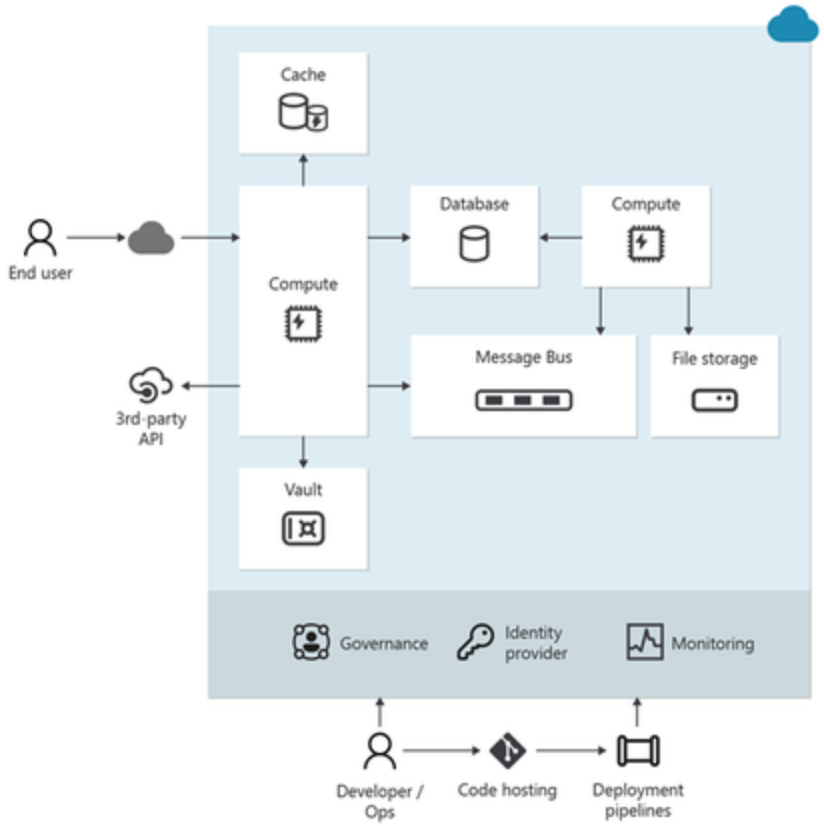
Resilient by Design: Building Reliable Workloads on Azure

Chris Ayers

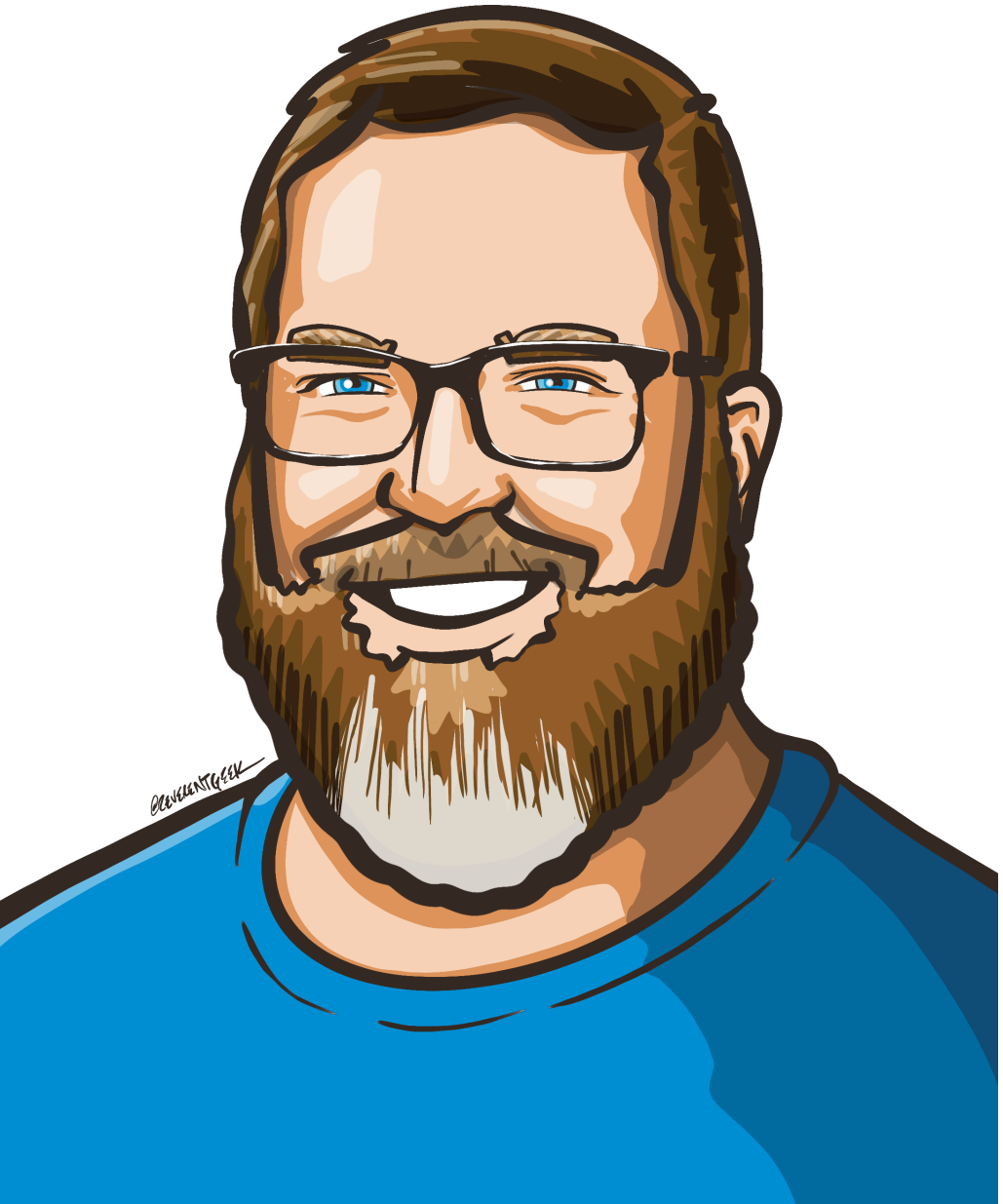
Principal Software Engineer

Microsoft

Azure EngOps AzRel



https://chris-ayers.com



Chris Ayers

Principal Software Engineer

Azure EngOps AzRel

Microsoft

BlueSky: @chris-ayers.com

LinkedIn: - chris-l-ayers

Blog: https://chris-ayers.com/

GitHub: Codebytes

Mastodon: @Chrisayers@hachyderm.io

~~Twitter: @Chris_L_Ayers~~

Reliability

Why Reliability Matters

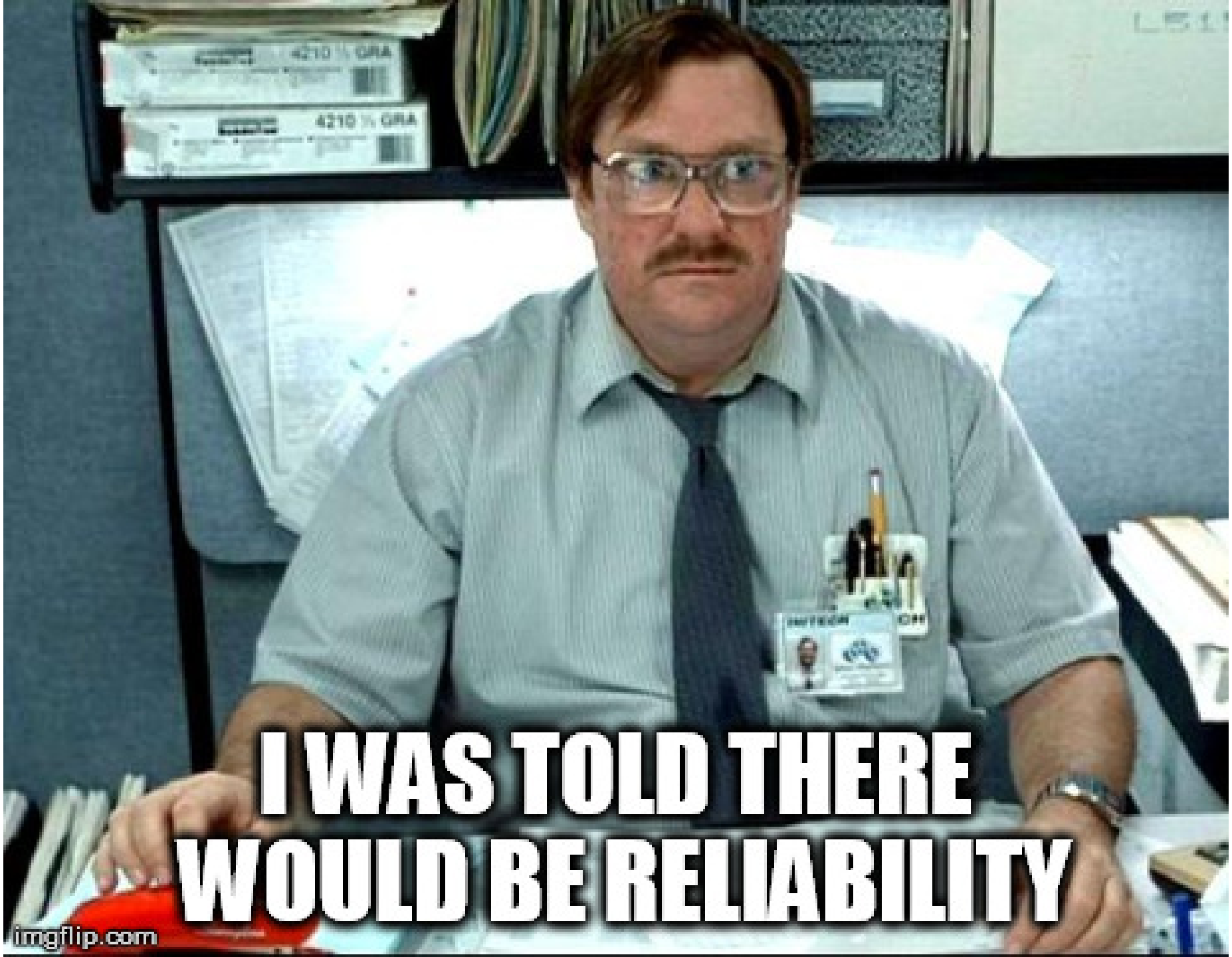
> A system is considered "reliable" if it can consistently serve users under normal or abnormal conditions.

- Reliability in distributed systems involves:
  - **Consistent performance** despite failures.
  - **Graceful degradation** when certain components become unavailable.
  - **Rapid recovery** within acceptable time limits.

Understanding Reliability, Resiliency & Recoverability

- Failures are **inevitable** in distributed systems
- The WAF frames reliability as three distinct properties:
  - **Resilient** — detect and **withstand** faults, degrade gracefully
  - **Recoverable** — **restore** within agreed **RTO/RPO** when resiliency is exceeded
  - **Available** — users access the workload at the promised quality
- Failures impact ***revenue***, ***reputation***, and ***customer trust***

FAILURE IS ALWAYS AN OPTION


I WAS TOLD THERE
WOULD BE RELIABILITY
imgflip.com

Reliability Levels

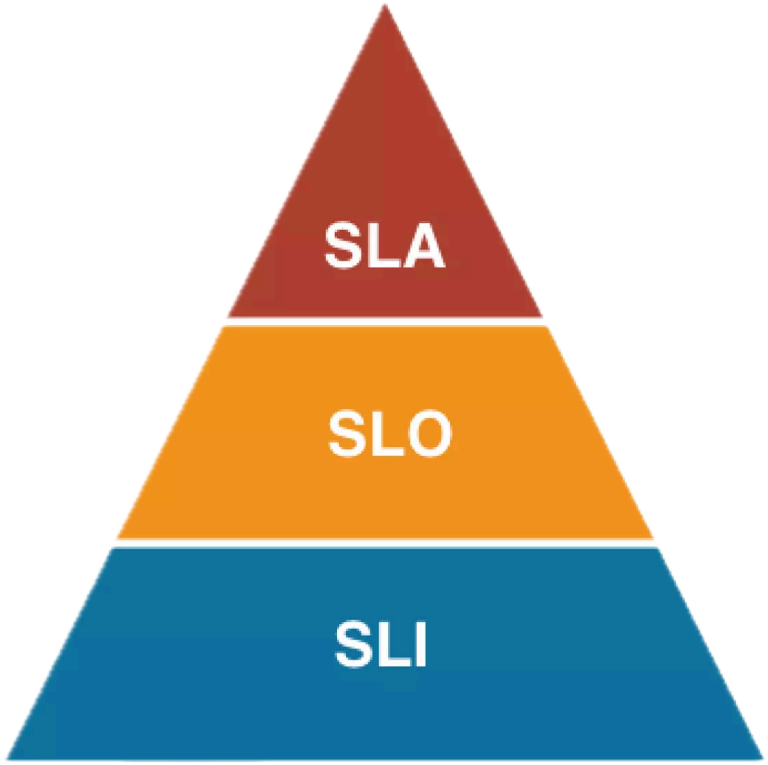
Level	Monthly Downtime	Annual Downtime	Cost
99.9%	43.8 minutes	8.75 hours	$
99.95%	21.9 minutes	4.375 hours	$$
99.99%	4.38 minutes	52.6 minutes	$$$
99.995%	2.19 minutes	26.3 minutes	$$$$
99.999%	26 seconds	5.25 minutes	$$$$$

> https://uptime.is/five-nines


ONE DOES NOT SIMPLY
ACHIEVE 100% UP TIME
makeameme.org

How Do We Measure Reliability?

	Definition	Example
SLI	Metric of user experience	API success rate per request
SLO	Reliability target over window	99.95% successful responses (30d)
Error Budget	1 − SLO (consumable failure)	0.05% of eligible requests @ 99.95%
SLA	Contract with penalties	99.9% monthly w/ credits

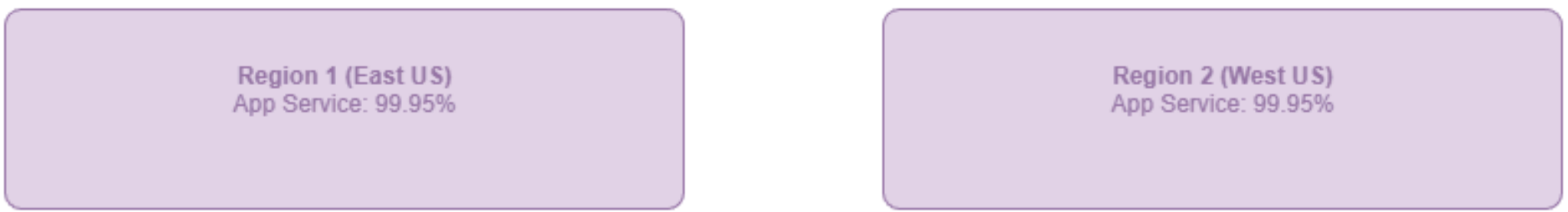


Composite SLA Calculation

Serial Dependencies (multiply)

99.99% × 99.95% × 99.99% × 99.99% = **99.92%**

Parallel Redundancy (increases availability)

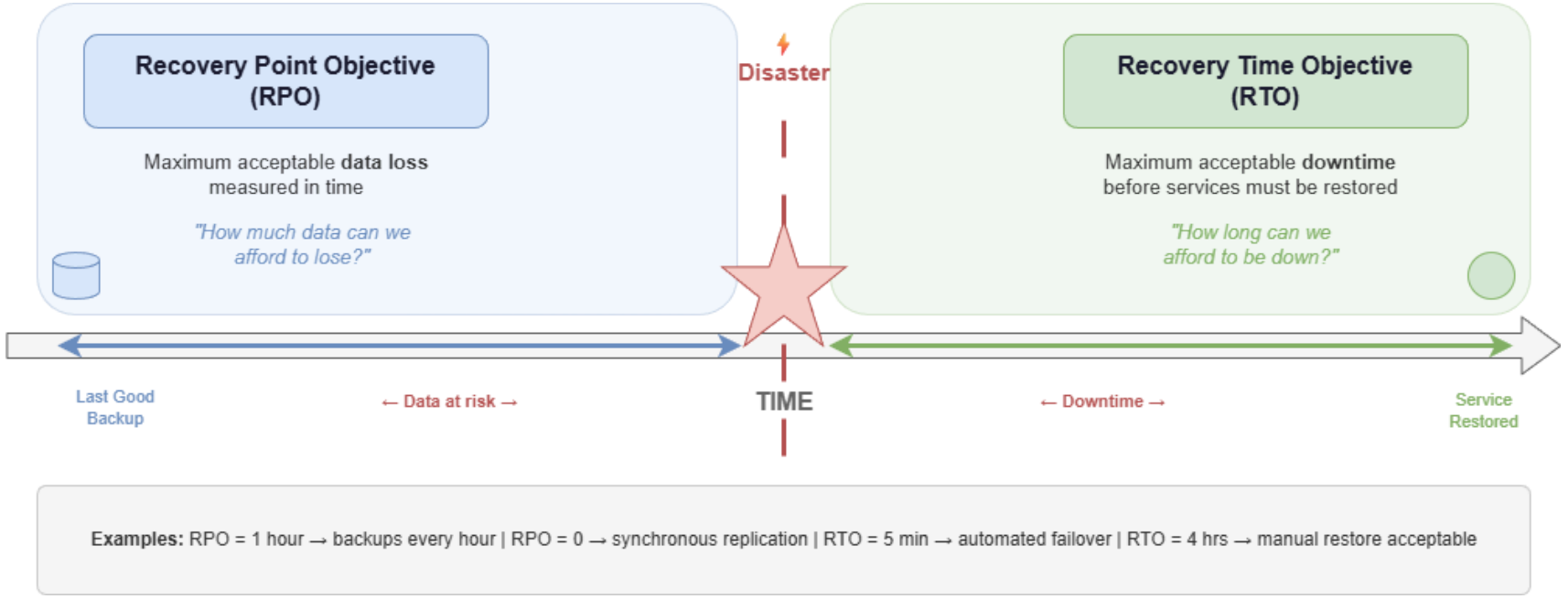


1 − (1 − 0.9995) × (1 − 0.9995) = **99.999975%**

> ⚠️ Serial dependencies **reduce** composite SLA | Parallel redundancy **increases** composite SLA

Understanding RPOs and RTOs

- **RTO**: Max acceptable **downtime** before services must be restored
- **RPO**: Max acceptable **data loss** measured in time



https://chris-ayers.com

Azure Well-Architected Framework

- Provides best practices and guidance for building high-quality Azure solutions.
- Ensures workloads are reliable, secure, efficient, and cost-effective.
- Azure Well-Architected Framework

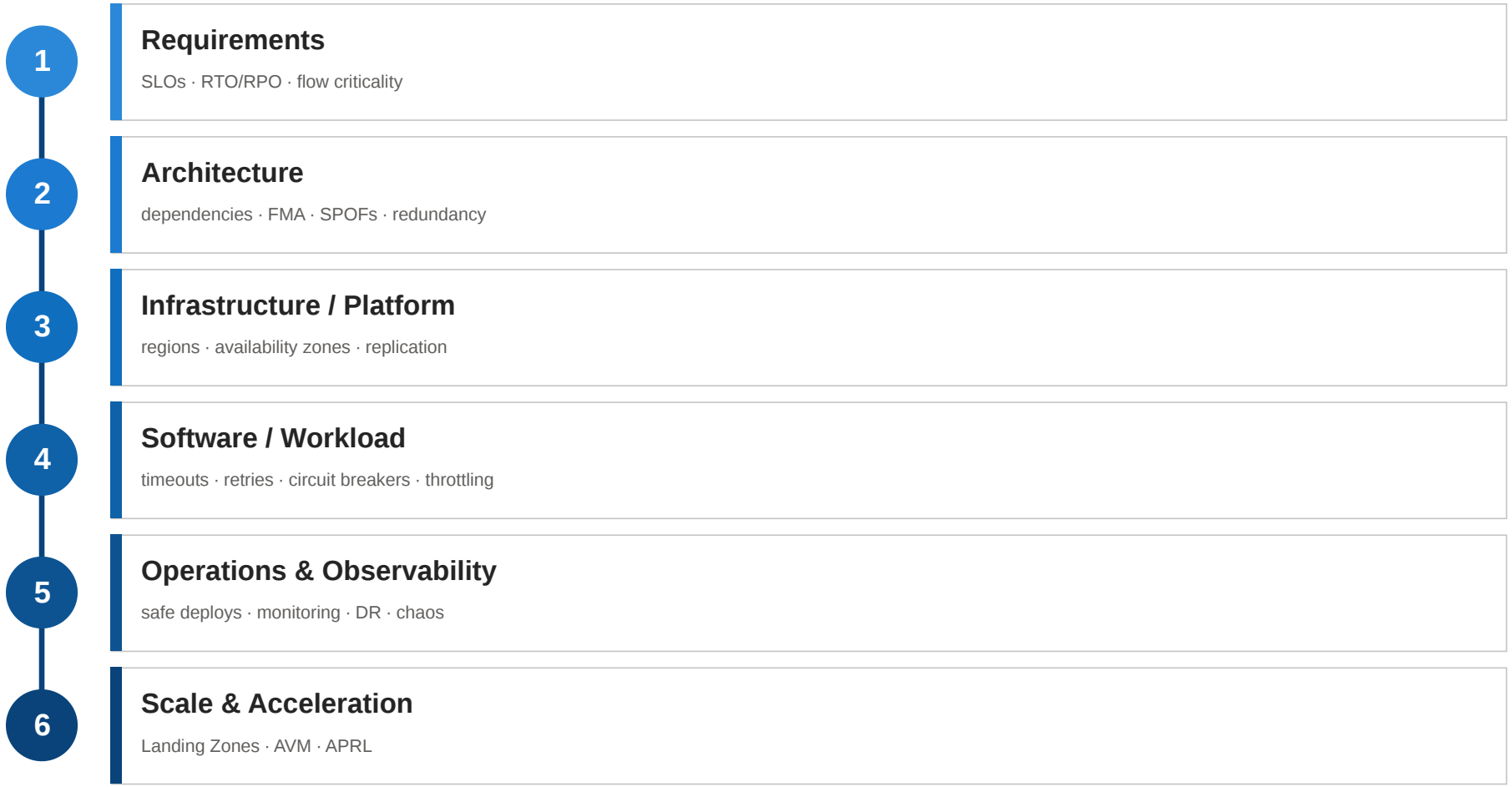
Design principles
Checklists
Recommendations
Tradeoffs

- Workload design
- Reference architectures
- Assessments
- Advisor recommendations
- Service guides

Microsoft Azure Well-Architected Framework Pillars

Reliability	Security	Cost Optimization	Operational Excellence	Performance Efficiency
Resiliency, availability, recovery	Protect data, detect threats, mitigate risks	Budgeting, reducing waste, efficiency	Observability, DevOps practices, safe deployments	Scalability, load testing, performance monitoring

The Reliability Journey



> We build reliability layer by layer — from requirements to scaling it across the org

Business & Non-Functional Requirements Layer

From Intent to Quantified Reliability

Establish resilience expectations before selecting technology:

- Define critical user journeys & classify components (critical vs. degradable)
- Quantify reliability targets (SLOs, error budgets, RTO/RPO)
- Align trade-offs early (cost, performance, security, operations)

Requirements: Resilience & Recovery

Business

- Quantify success targets for flows
- Understand platform SLAs, limits, constraints

Key Principle

> Design for **how you respond**, not just prevention

Recovery

- Recovery plans with regular drills
- Trusted backups & immutable copies
- Automated detection & self-healing

Resilience

- Classify **critical** vs. **degradable** components
- Design for fault isolation & graceful degradation
- Eliminate single points of failure

Identify & Rate User and System Flows

Flow Identification

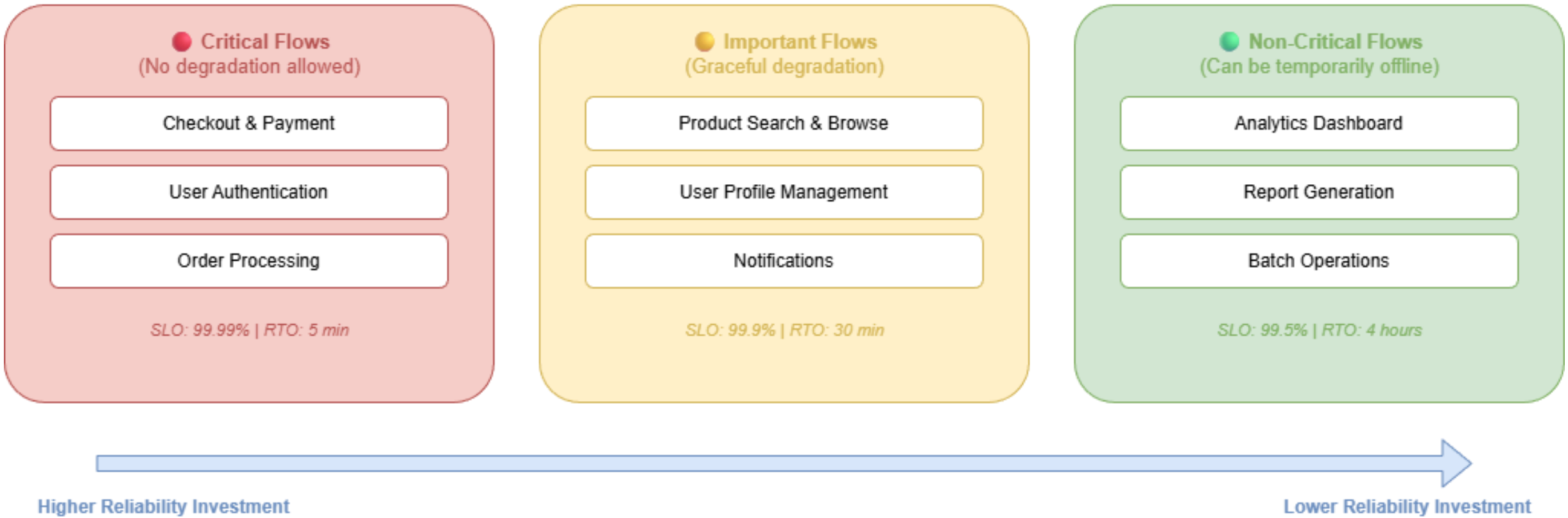
- Map **critical user journeys** end-to-end
- Identify all **system flows** (background jobs, sync, integrations)
- Rate each flow on a **criticality scale** tied to business impact

Criticality Classification

Tier	Reliability Target
Critical	99.99%, RTO < 5 min
Important	99.9%, RTO < 30 min
Non-Critical	99.5%, RTO < 4 hours

> Set SLOs and RTO/RPO **per flow**, not per system

User Flow Criticality Map



Operational Readiness

- Shift left to anticipate failures early
- Test failures in development
- Ensure cross-team visibility
- Use **health models** to track SLO attainment & workload health
- Use observability for rapid remediation

Strategy	Implementation
Observable Systems	Aggregate telemetry for holistic health views
Failure Simulation	Validate recovery metrics with realistic tests
Automation	Minimize human error and ensure consistency
Continuous Learning	Improve from real production incidents
Health Modeling	Map telemetry to SLO-based health state (Azure Monitor)
Proactive Monitoring	Prioritize alerts for active failures

Keep It Simple

- Avoid overengineering architecture, code, and operations
- Simplicity reduces inefficiencies and misconfigurations
- Maintain a balanced approach to avoid single points of failure

Trade-Offs

- Align trade-offs with business priorities
- Use scenario planning to assess impacts
- Continuously iterate and monitor performance

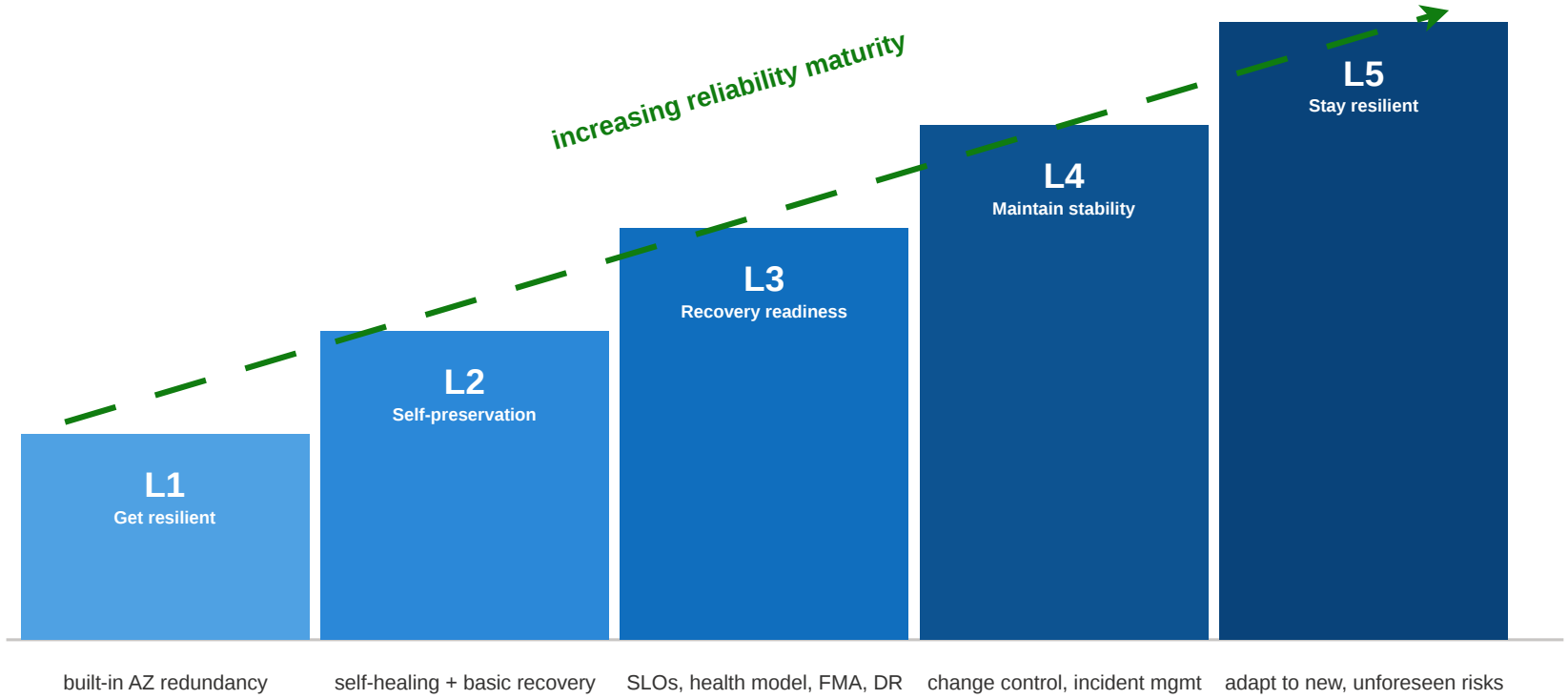

ARCHITECTS
RELIABILITY
COST OPTIMIZATION

Reliability Trade-Offs with Other Pillars

Pillar	More Reliability Means...	Less Reliability Means...
Cost	Redundancy, monitoring, over-provisioning	Right-sized, single-region, simpler ops
Security	Larger attack surface, may delay patches	Strict controls, frequent patching
Ops Excellence	Complex architecture, extensive DR testing	Simpler systems, less overhead
Performance	Replication latency, failover overhead	Lean deployments, speed-focused

Reliability Maturity Model

Assess your **current posture** and follow a staged path — five levels, each building on the previous.



Reliability maturity model

Architecture Layer

Structuring for Failure Containment

Focus on failure domains, redundancy strategy, and dependency design before implementation.

Dependency Management

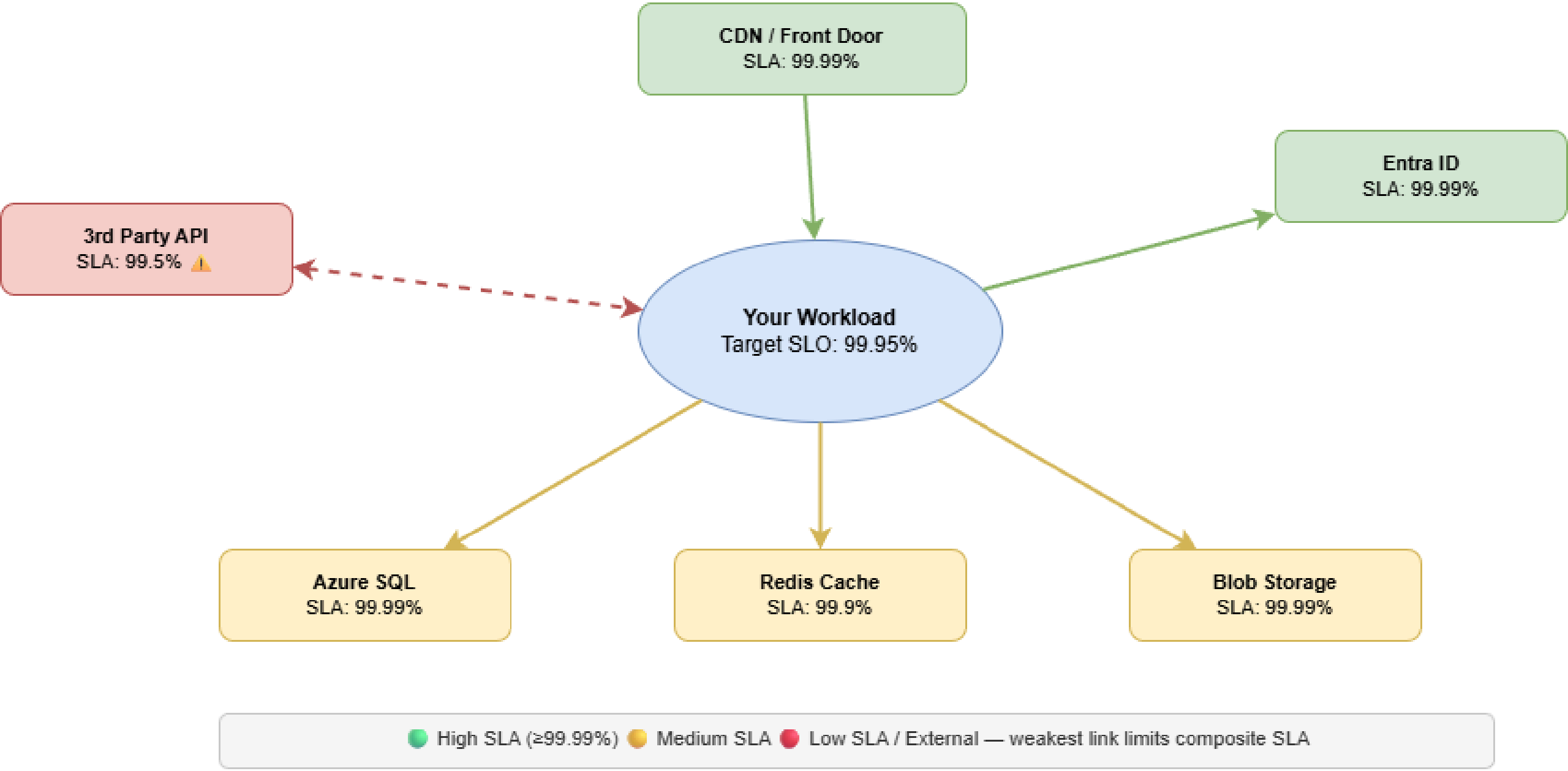
Map Your Dependencies

- Identify all **upstream** and **downstream** dependencies
- Document each dependency's **SLA and failure modes**
- Classify: **strong** (required) vs. **weak** (degradable)

Mitigation Strategies

Strategy	When to Use
Caching	Tolerate downtime for reads
Async messaging	Decouple from unreliable services
Fallback values	Defaults when dependency fails
Circuit breaker	Prevent cascading failures
Timeout + retry	Handle transient issues

Dependency Mapping & SLA Impact



Failure Mode Analysis (FMA)

Proactive Identification

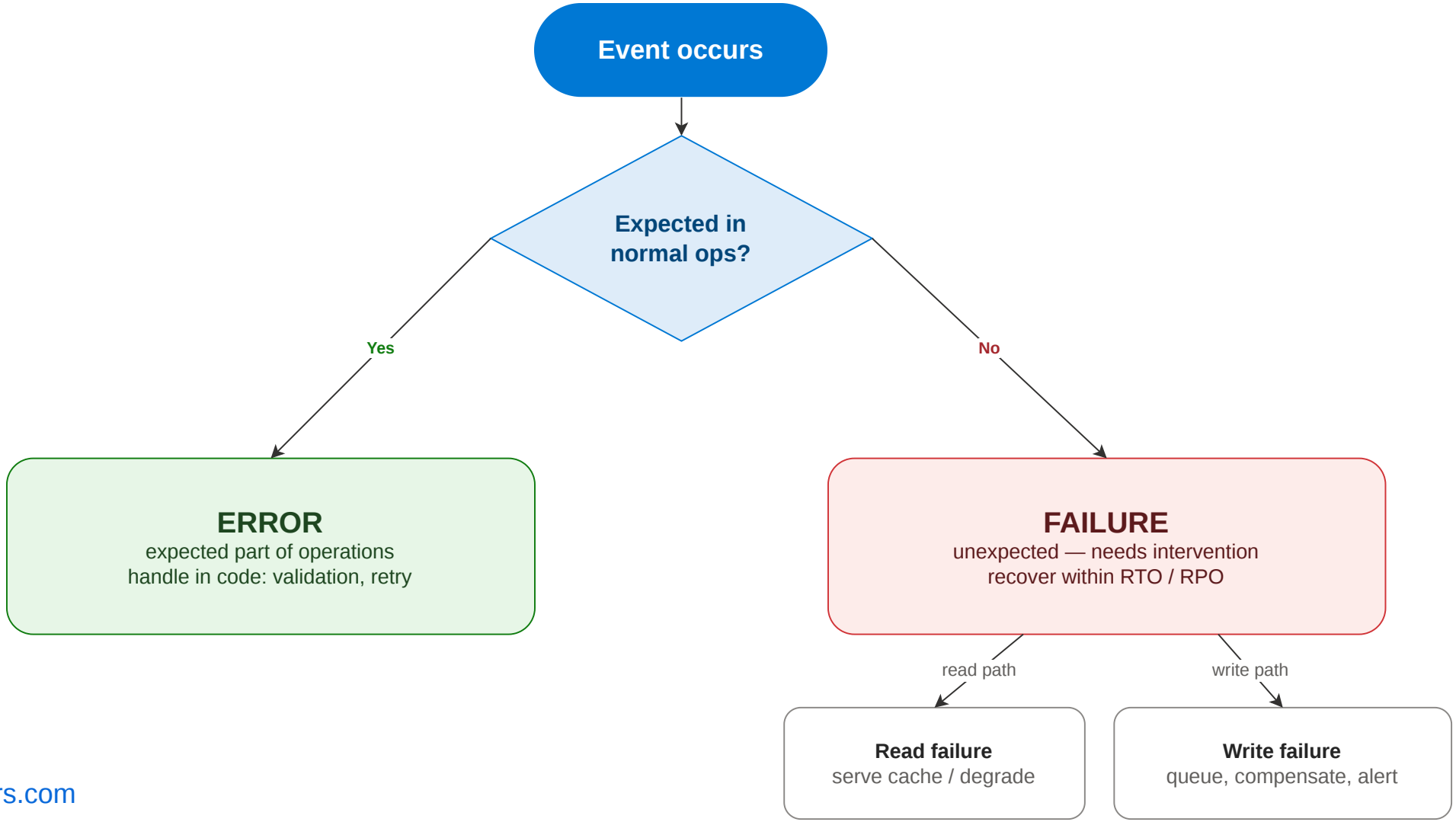
- Distinguish **failures** (unexpected, need intervention) from **errors** (expected in normal ops)
- Analyze **read** vs. **write** failures separately — impact & mitigation differ
- Prioritize by user impact & blast radius; use checklists, post-mortems, dependency maps

> Failure Examples

Effective Mitigation

- Architect graceful degradation & fallbacks
- Instrument for fast anomaly detection
- Automate failover & data replication where feasible

FMA: Failure vs. Error



Single Points of Failure (SPOFs)

Understanding SPOFs

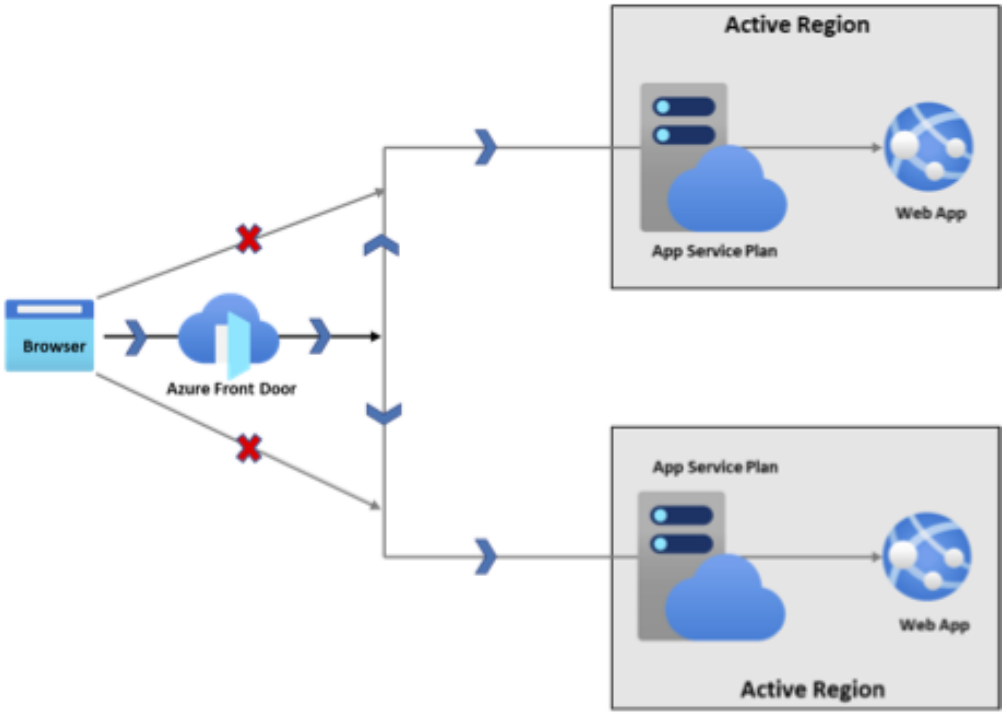
- Single component whose failure halts critical flow
- Occur in infra, data, code paths, people/process
- Early detection reduces mean time to mitigation

Eliminating SPOFs

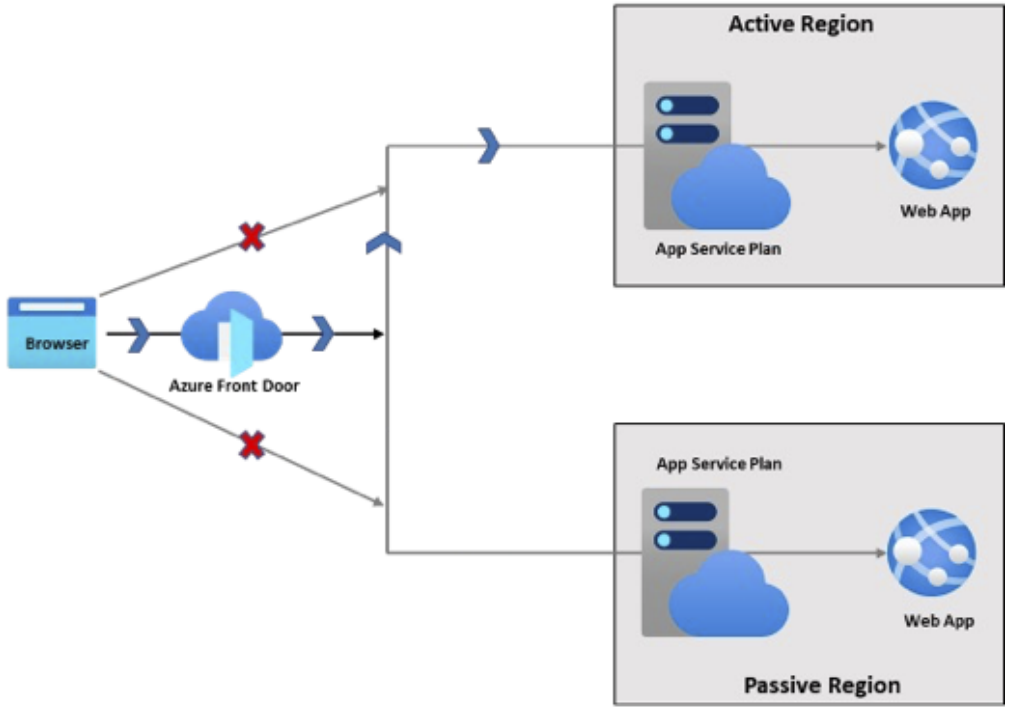
- Redundancy & diversity (multi-zone / multi-instance)
- Load balancing & partitioning
- Automated failover runbooks
- **Delete protection** via Azure resource locks on redundant components

Active-Active vs. Active-Passive

Active-Active: Multiple instances process requests simultaneously.



Active-Passive: Primary instance processes traffic; secondary is on standby.

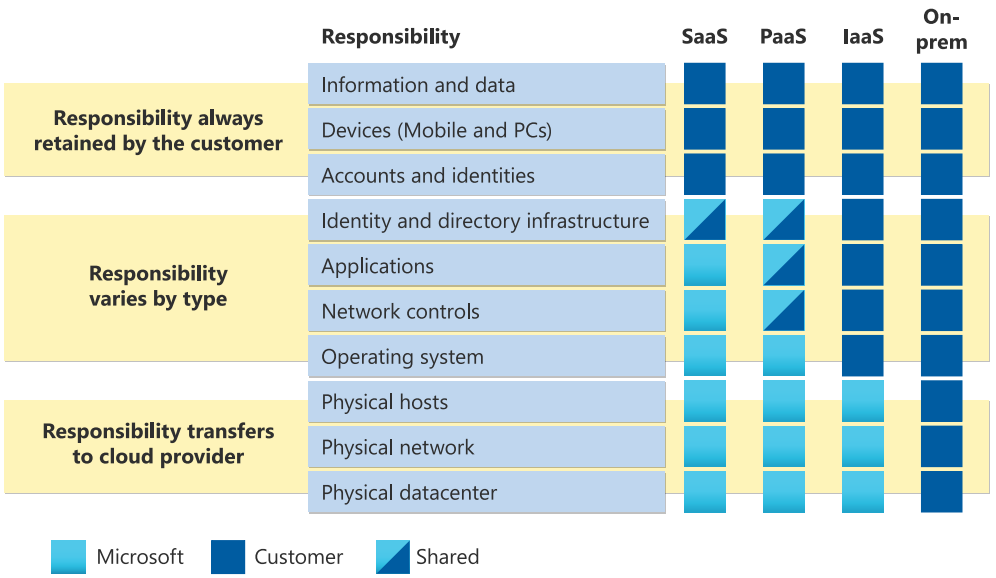


Infrastructure / Platform Layer

Foundation for Consistent Resilience

Automated, policy-driven environments (Landing Zones, AVM, APRL) reduce variance & misconfiguration risk.

Azure-Customer Shared Responsibility Model



> Azure provides service SLAs; you own workload reliability

Your Reliability Responsibilities

Layer	Customer Owns
Data	Backup strategy, replication, encryption
Application	Retry logic, circuit breakers, health probes
Identity	MFA, conditional access, break-glass accounts
Network	Redundant paths, ExpressRoute resilience
Infra Config	Zone/region selection, scaling rules

Azure Regions and Regional Strategy

Region Selection Criteria

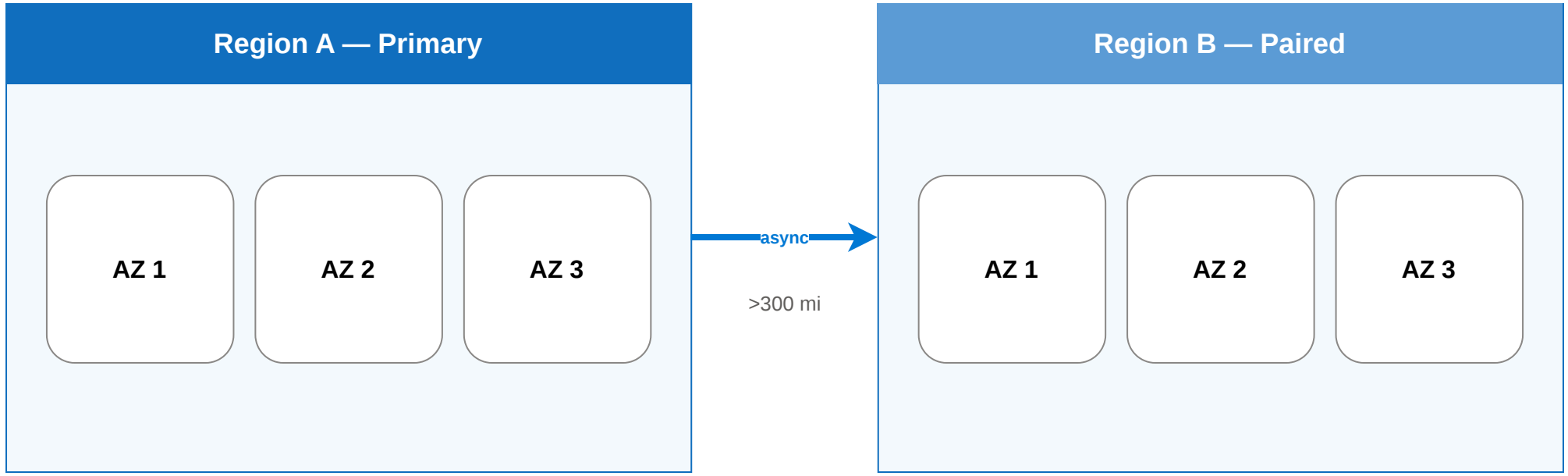
- **Latency**: Proximity to users
- **Compliance**: Data residency, sovereignty
- **Service Availability**: Not all services in all regions
- **Cost**: Pricing varies by region

Multi-Region Considerations

Factor	Impact on Design
Region Pairs	Sequential updates, prioritized recovery
Non-paired	Flexible, but plan your own DR
Sovereign Clouds	Isolated (Gov, China)
Distance	Affects replication lag & latency

> Pairing ≠ automatic failover—you must design for it

Azure Region Pairs

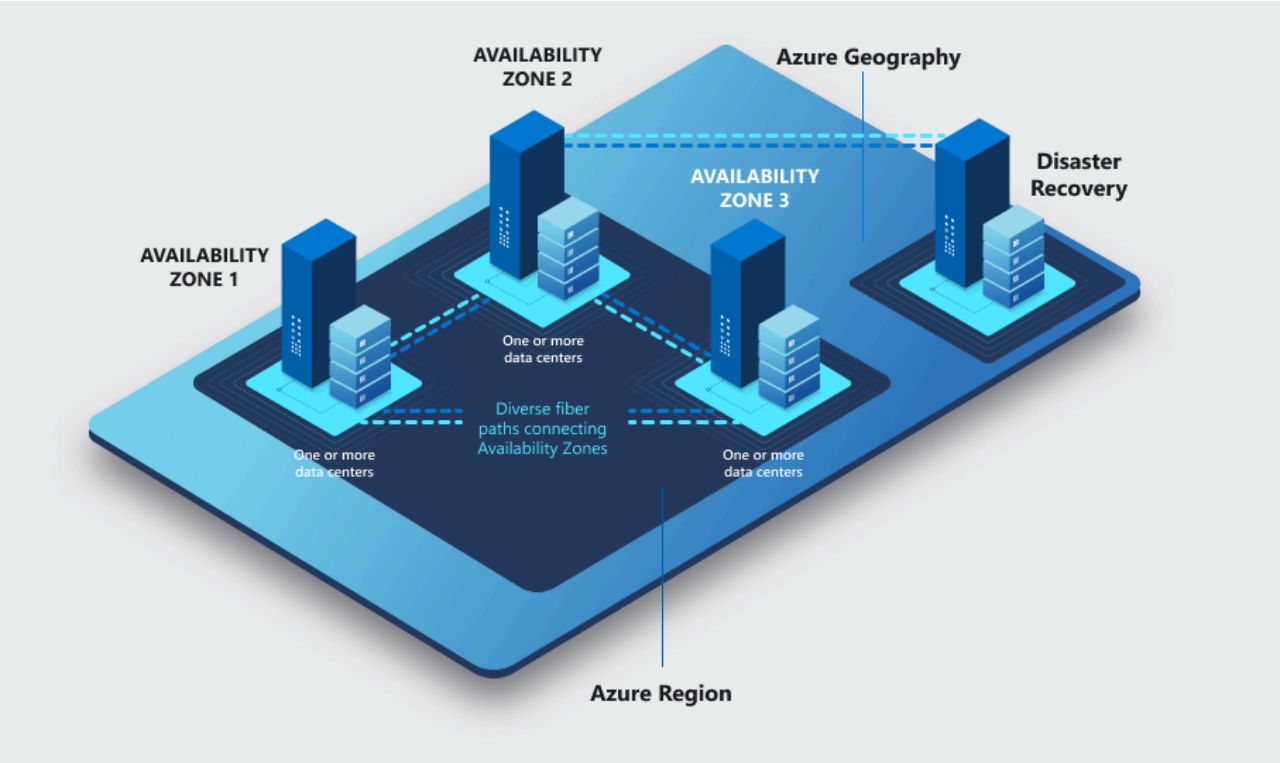


Region pairs: sequential platform updates · prioritized recovery · data residency in-geography

Pairing is not automatic failover — you design and trigger it

Azure Availability Zones

- Physically separate datacenters within a region
- Independent power, cooling, and networking
- Low-latency connections; measure for your workload
- Maintenance and failover behavior vary by service
- View Region Support



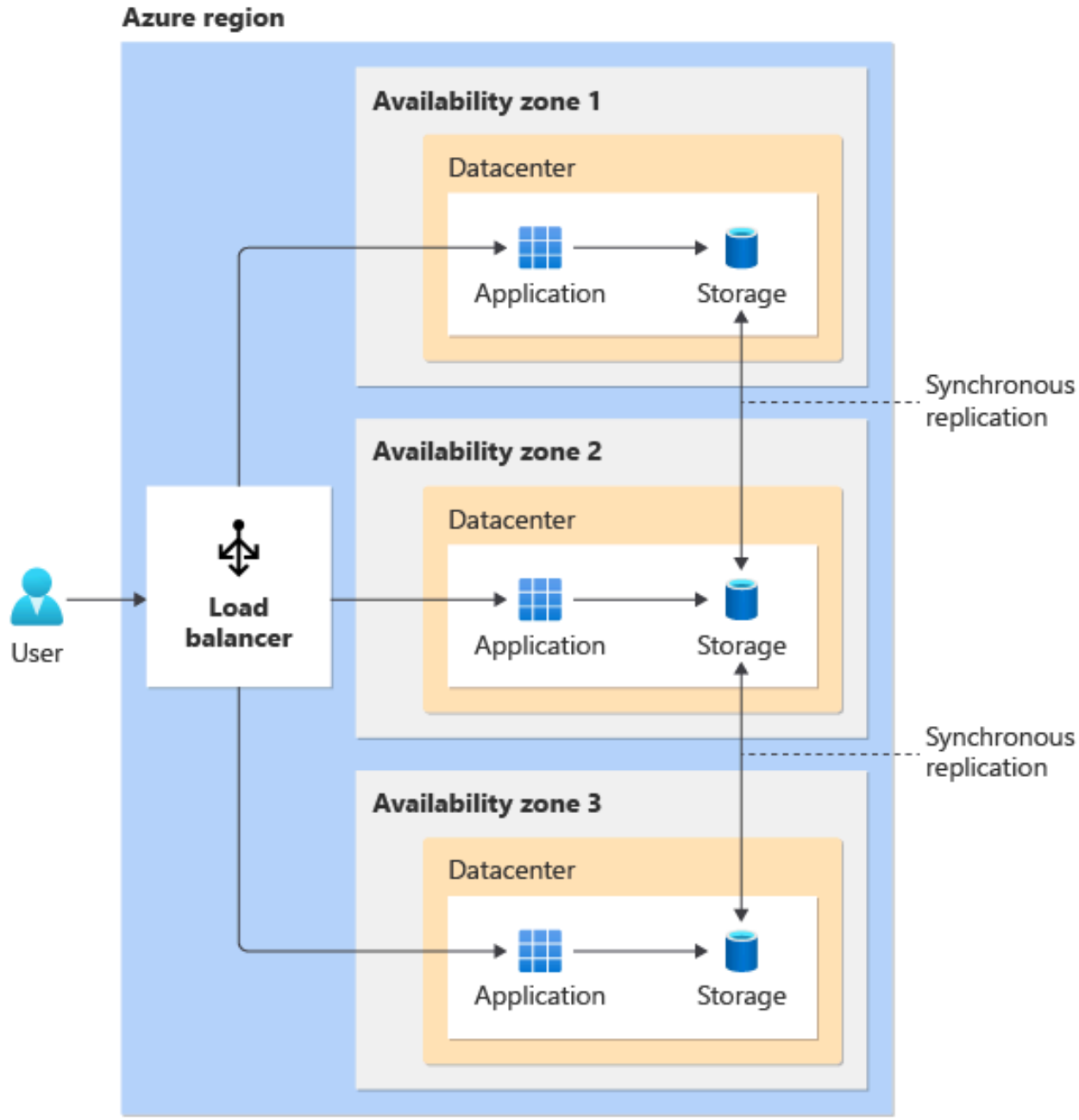
Types of Availability Zone Support

Type	Deployment	Failover	Who Manages
Zonal	Pinned to one zone	You handle failover	Customer
Zone-Redundant	Spread across zones	Automatic	Microsoft
Zone-Resilient	Zonal or zone-redundant	Survives zone outage	Varies
Non-Zonal (Regional)	No zone affinity	Service-dependent	Service-dependent

- View Services Support

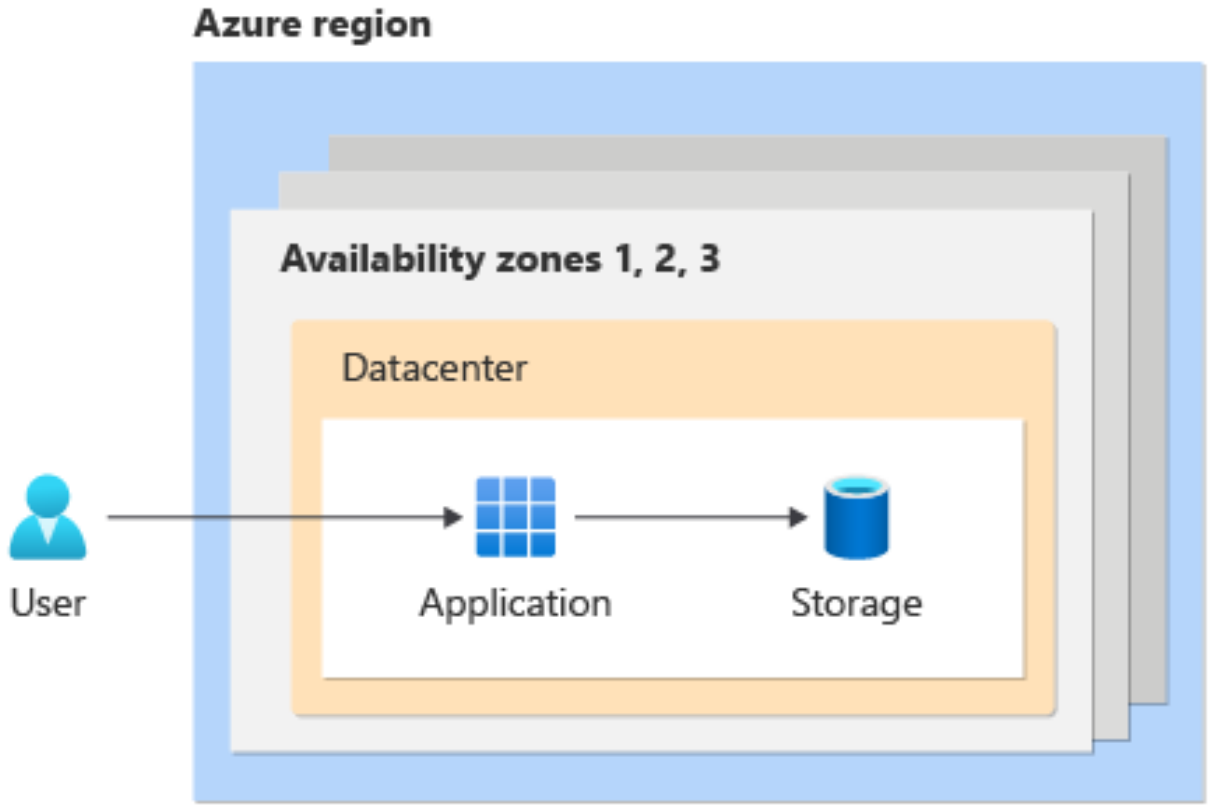
Zonal Resources

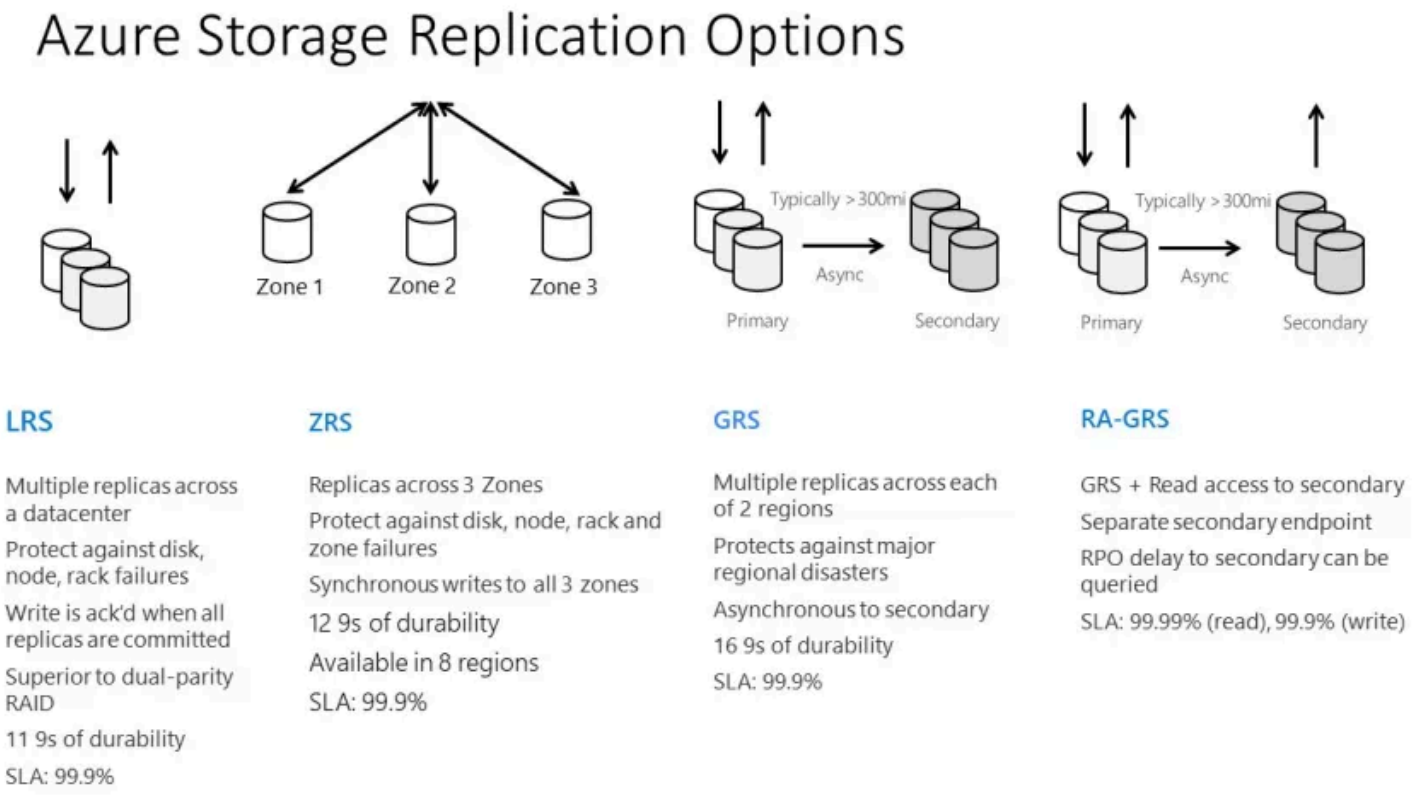
- Pinned to a specific availability zone
- Combine multiple zonal deployments for high reliability
- **You** manage replication, load balancing, and failover
- Deploy across zones for zone-level protection



Zone-Redundant Resources

- Spread automatically across multiple availability zones.
- Microsoft manages request distribution and data replication.
- Automatic failover if a zone goes down.

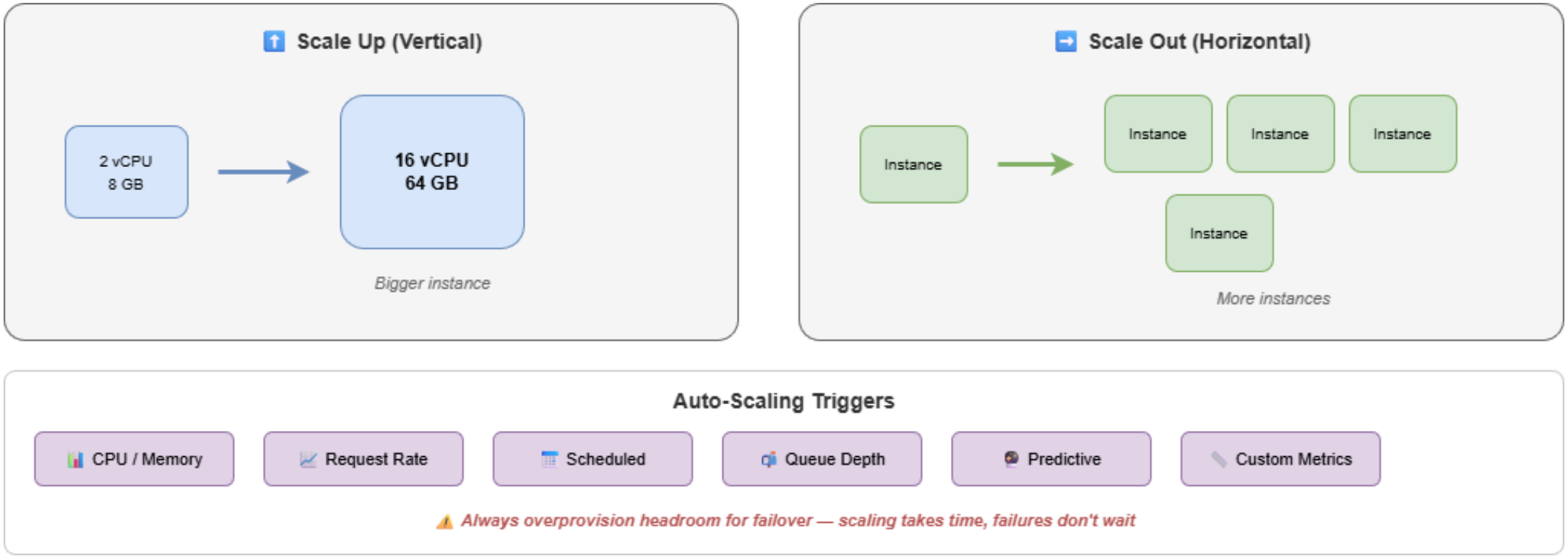




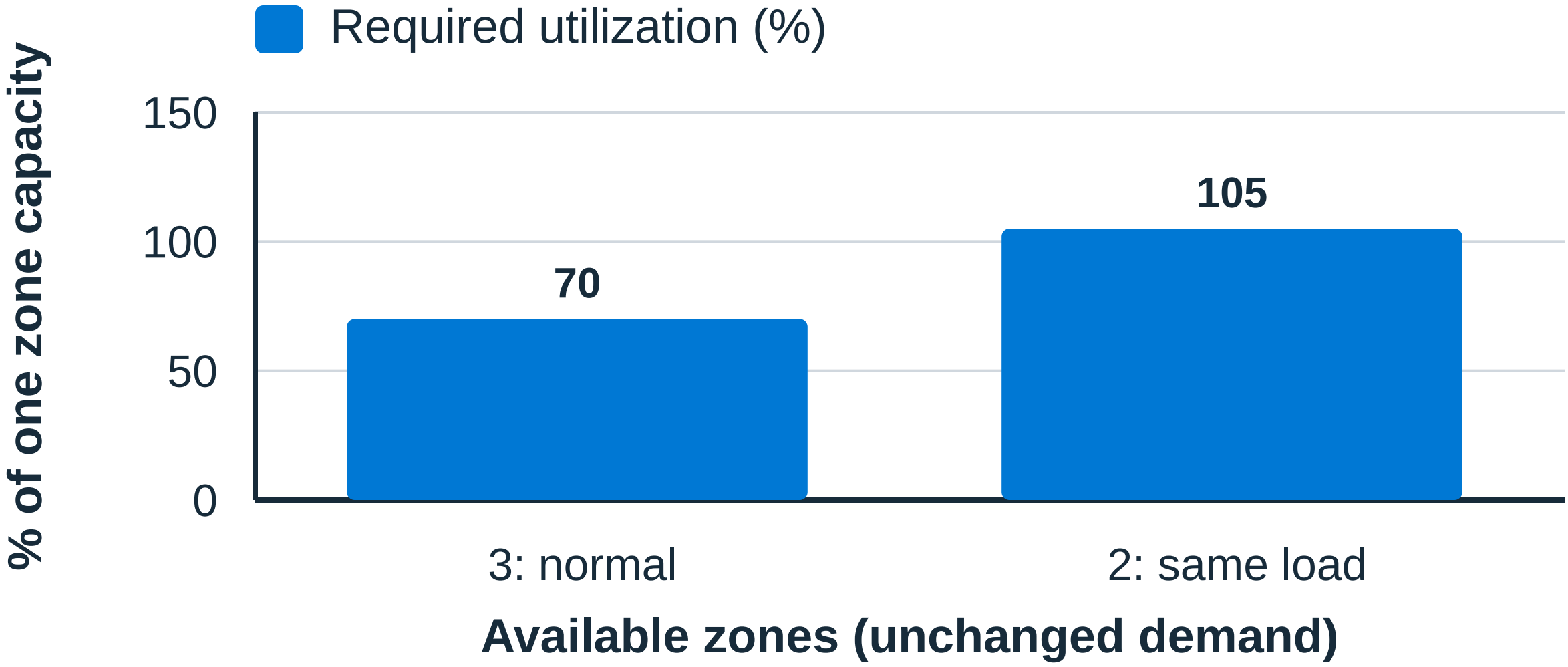
Data Replication: Storage Options

- **Replication:** LRS/ZRS synchronous; GRS/GZRS geo-replication asynchronous. Verify lag and recoverable state.
- **Recovery:** Hot/Cool/Archive tiering changes restore time; immutable storage protects against ransomware.

Scaling Strategies



Zone loss: 105% required



Source: Illustrative capacity model: 70% x 3 / 2

Azure Reliability Services

Traffic & Load Balancing

Service	Purpose
Azure Front Door	Global HTTP LB, failover, WAF
Traffic Manager	DNS-based global routing
Load Balancer	Regional L4 LB
Application Gateway	Regional L7 LB, WAF

Backup & Recovery

Service	Purpose
Azure Site Recovery	DR replication & failover
Azure Backup	Managed backup (VMs, SQL, files)
Service Health	Platform outage alerts
Resource Health	Individual resource status
Azure Advisor	Proactive recommendations

Software / Workload Layer

Runtime Resilience Behaviors

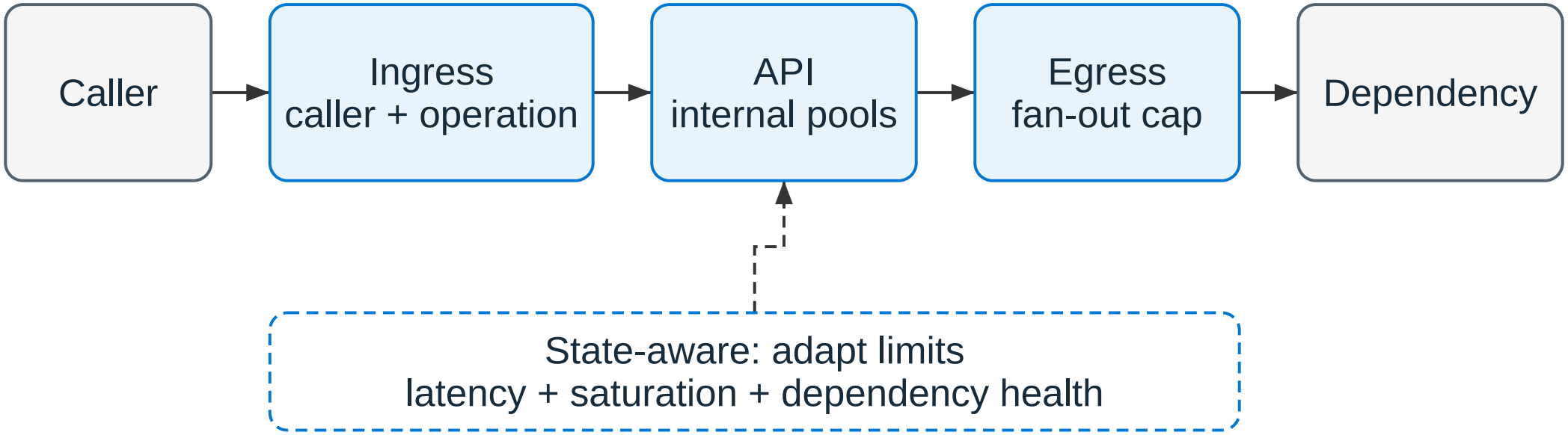
Embed failure-aware logic: timeouts, retries, backoff, bulkheads, circuit breakers, idempotency, hedging.

Resilience Patterns

Pattern	Problem Solved	Key Considerations
Timeout	Prevent hanging on slow dependency	Use measured latency and the end-to-end deadline
Retry + Backoff + Jitter	Transient faults (throttling, blips)	Cap attempts; honor server delays and idempotency
Circuit Breaker	Failing dependency cascading	Trip on error rate/latency; half-open probes
Bulkhead Isolation	One noisy component	Resource partitioning
Dead Letter Queue	Bad messages blocking progress	Monitor & replay with alerting
Throttling / Rate Limiting	Protect service from overload & noisy neighbors	Bound load; give retry guidance when safe
Graceful Degradation	Maintain partial service	Feature flags, fallback data

> Throttling design guide

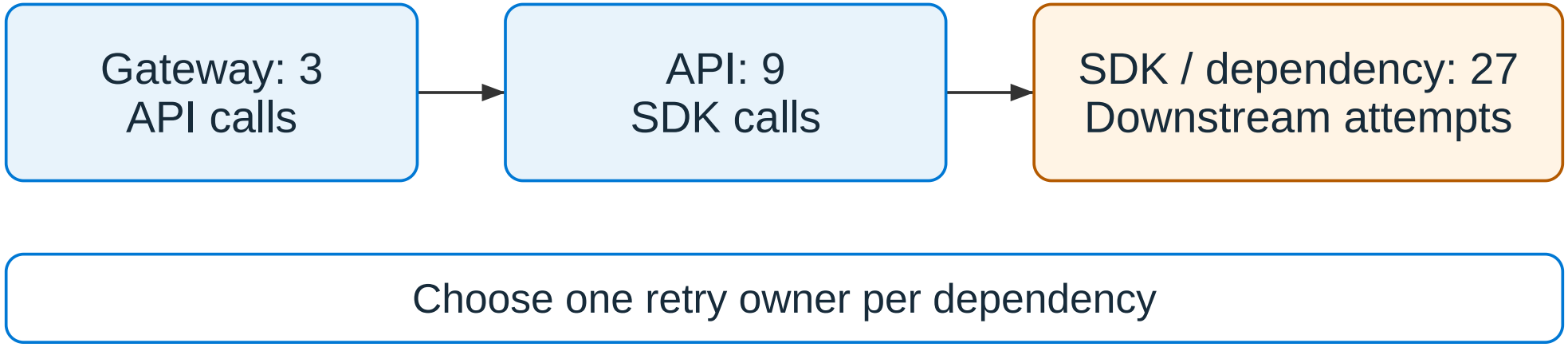
Throttling Belongs Along the Whole Path



Ingress → **internal** → **egress**, with **state-aware** limits across boundaries.

Three Retry Layers Can Make 27 Attempts

1 logical request | 3 total attempts per layer
Initial attempt + up to 2 retries at each layer



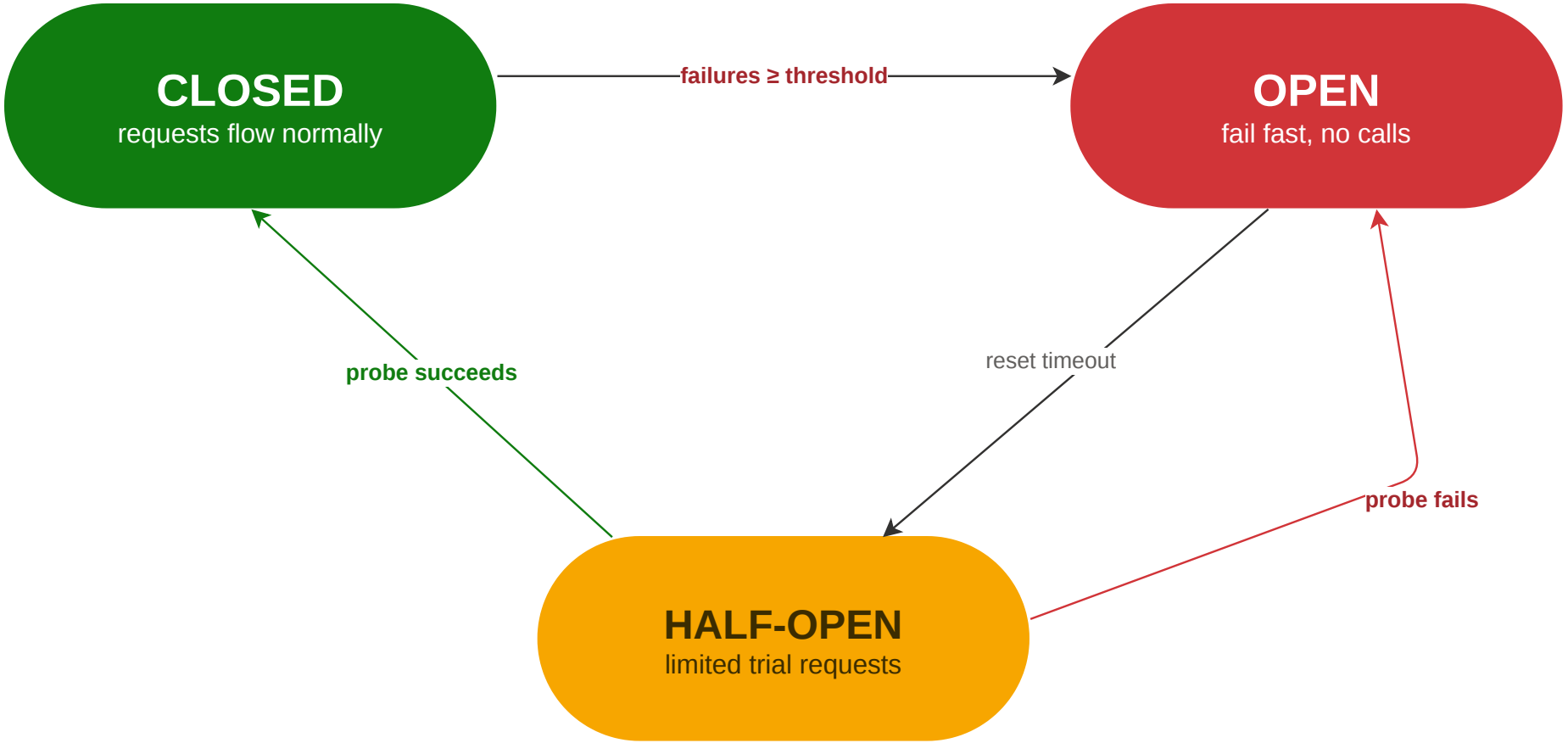
3 × 3 × 3 = 27 downstream attempts for one logical operation.

Make Throttling a Retry Contract

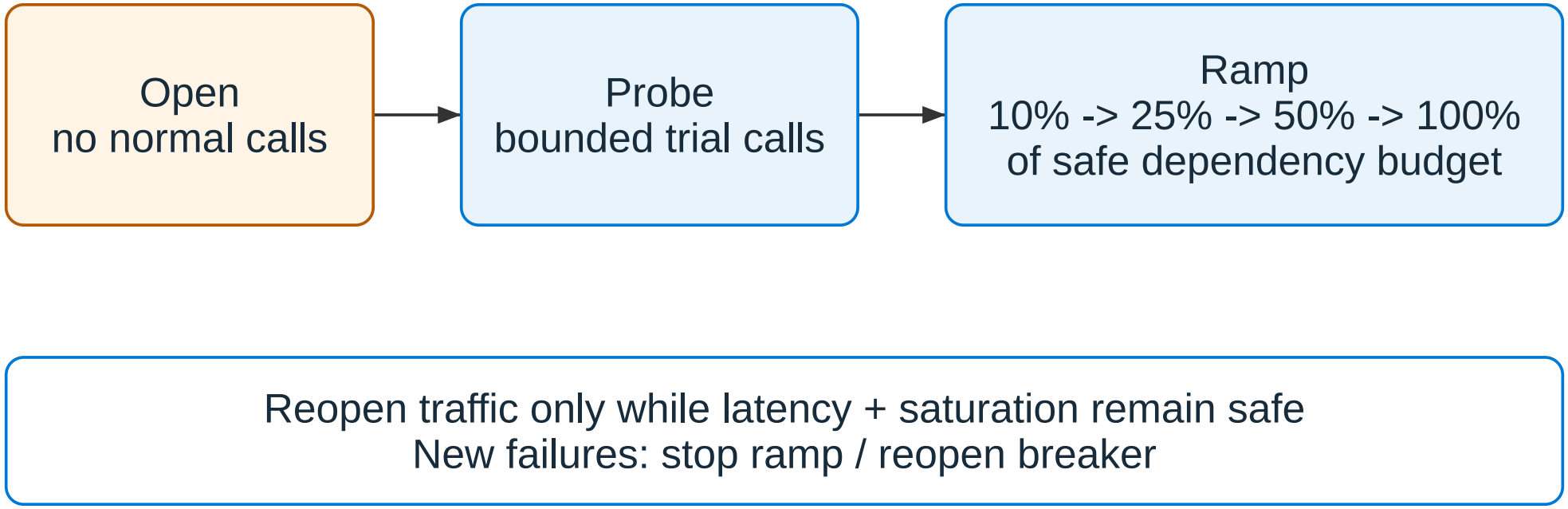
Before another attempt	Required decision
Transient?	Follow this dependency's error code and retry guidance
Safe?	Confirm idempotency, deduplication, or the previous outcome
Attempts left?	Count SDK and application attempts together
Time left?	Fit the server delay and next attempt inside the deadline

Retry-After: 2 s with 1.1 s left? Do not retry inside that request.

Circuit Breaker — State Machine

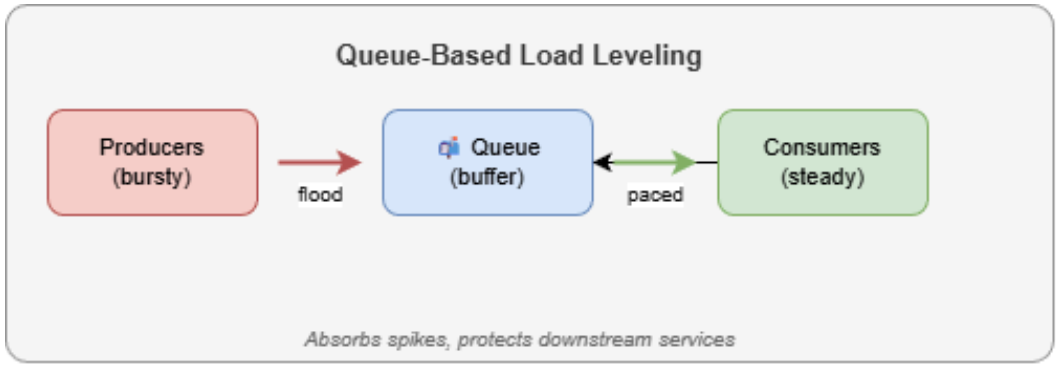


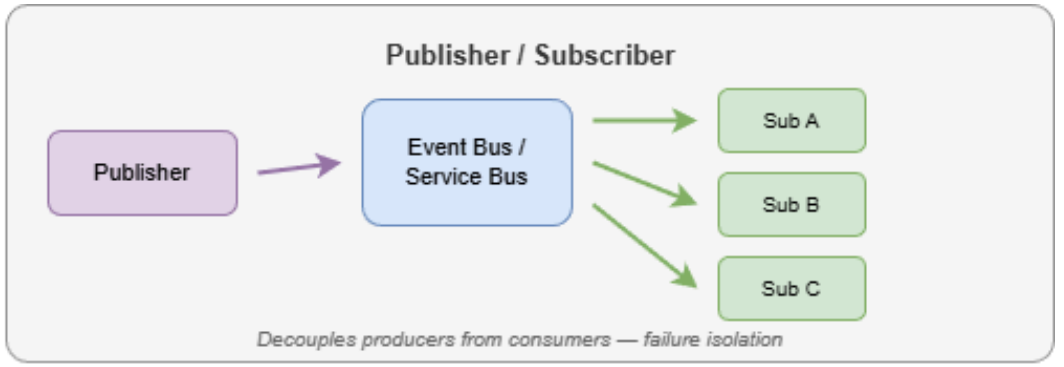
Recovery Needs a Rate Limit Too

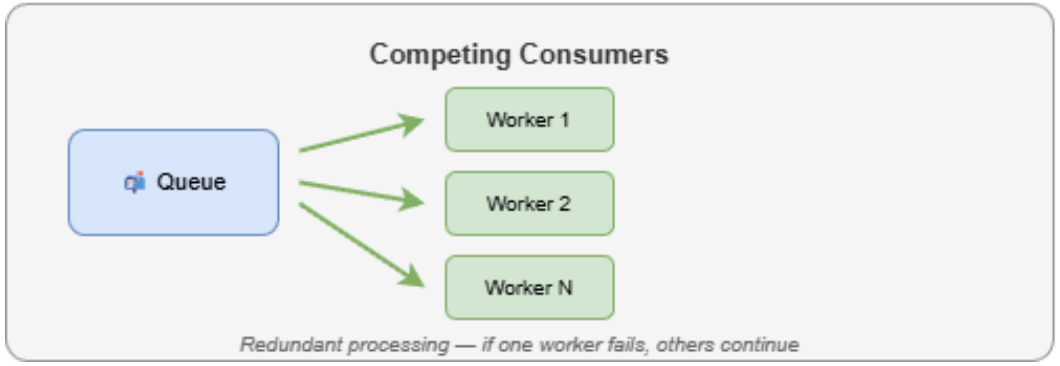


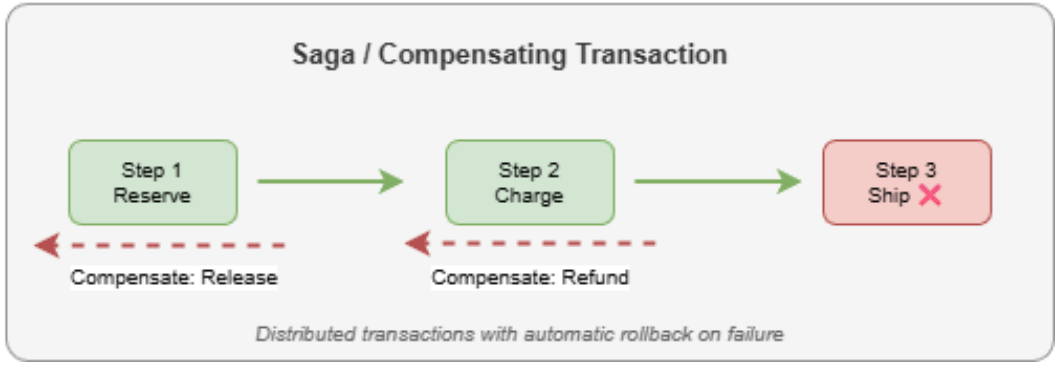
Do not release **new work + retries + backlog + cache fills** all at once.

Async & Event-Driven Patterns









Azure SDK Resiliency Best Practices

Built-in SDK Features

- **Retry** — Service-specific policies; inspect defaults
- **Timeouts** — Check each timeout's scope
- **Throttling** — Service-specific errors and delay hints
- **Idempotency** — Only where explicitly supported

Your Responsibilities

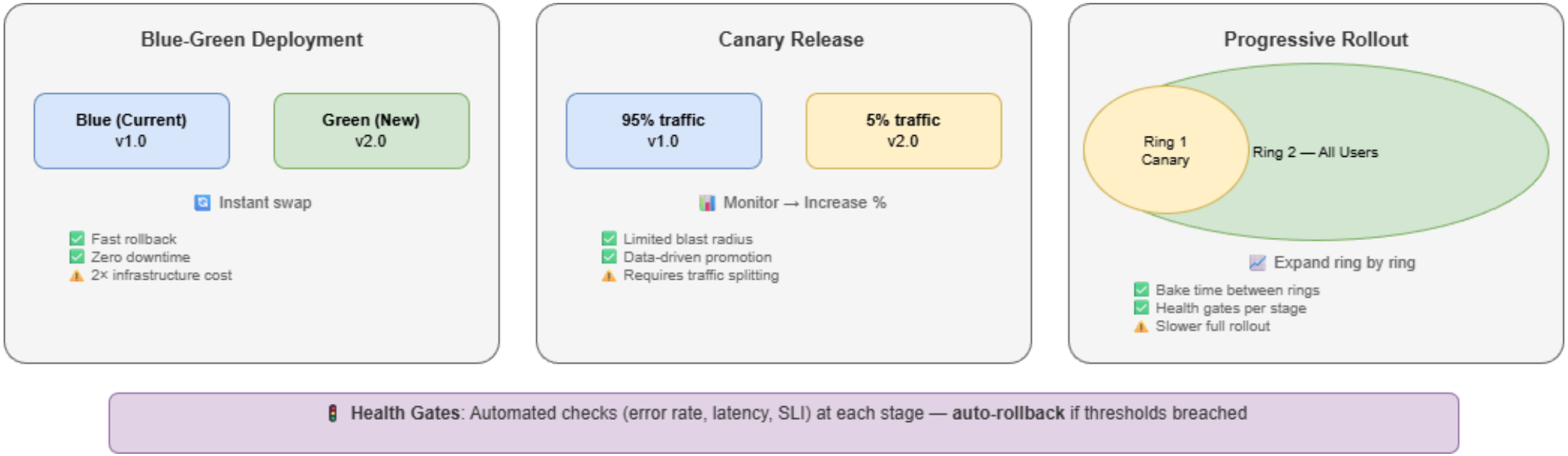
- **Circuit Breaking** — Polly / resilience libs
- **Connection Reuse** — Reuse clients (thread-safe)
- **Backpressure** — Bounded channels, fail fast
- **Observability** — Correlate traces, log retries

Operations & Observability Layer

Detect, Respond, Learn

Emphasize fast detection, validated recovery paths, and continuous improvement through data & drills.

Safe Deployment Practices



Health Gates: Automated checks (error rate, latency, SLI) at each stage — **auto-rollback** if thresholds breached

OpenTelemetry

- Open standard for generating, collecting, and exporting telemetry data (traces, metrics, logs)
- Supported by major cloud providers, including Azure Monitor, AWS CloudWatch, and Google Cloud Operations
- Enables vendor-neutral instrumentation and observability across distributed systems

Azure Service Groups

Health Metrics Aggregation

- **Cross-environment health**: Aggregate health metrics from multiple apps and environments
- **Resource type inventory**: Unified views across subscriptions
- **Unified monitoring**: Single Service Group provides health visibility across management groups

Least-Privilege Access

- **Selective permissions**: Service Groups don't inherit member permissions
- **Role-based viewing**: Assign different users access to same Service Group with varying resource visibility
- **Monitoring metrics**: Apply minimal privileges for metrics access without resource management rights

Metrics & Error Budgets

Concept	Formula / Definition	Example
Availability SLI	(Successful Requests) / (Total Requests)	99,950 / 100,000 = 99.95%
Latency SLI	% of requests under threshold	95% < 250ms (p95), 99% < 400ms
Error Budget	1 - SLO	SLO 99.9% => 0.1% budget
Burn Rate	Bad-request fraction / (1 - SLO)	0.2% / 0.1% = 2× burn
MTTR	Avg restore time for incidents	Track trend downwards

- Burn Rate **> 4×** for 1h: Freeze deploys; incident review
- Burn Rate **2×** sustained: Reduce change volume
- Burn Rate **< 1×**: Continue roadmap; schedule chaos tests

Error Budget Burn-Down

Validating Resilience

Why Validate?

- **Early Issue Detection**: Find problems before customers do
- **Increased Confidence**: Better team preparedness for incidents
- **Reduced Recovery Time**: Faster MTTR during real outages
- **Trust Building**: Demonstrate resilience to stakeholders

How to Validate

- **Start Simple**: Begin with critical paths and core functions
- **Game Days**: Schedule cross-team incident response exercises
- **Iterate**: Evolve tests as your architecture changes
- **Track**: Document findings and improvements

Load Testing

Key Benefits

- **Prevents Surprises**: Identify capacity issues before production
- **Validates Scaling**: Ensure your auto-scaling works properly
- **Finds Weaknesses**: Spot resource exhaustion and memory leaks

Implementation

- **Cost-Effective**: Much cheaper than emergency scaling during incidents
- **Azure Tools**: Azure Load Testing service with JMeter support
- **Best Practice**: Test with realistic user patterns and data volumes

Chaos Engineering

Core Principles

- **Controlled Failure**: Introduce failures in controlled environments
- **Best Practice**: Start small with clear abort conditions
- **Continuous Process**: Build complexity as confidence increases

Key Azure Scenarios

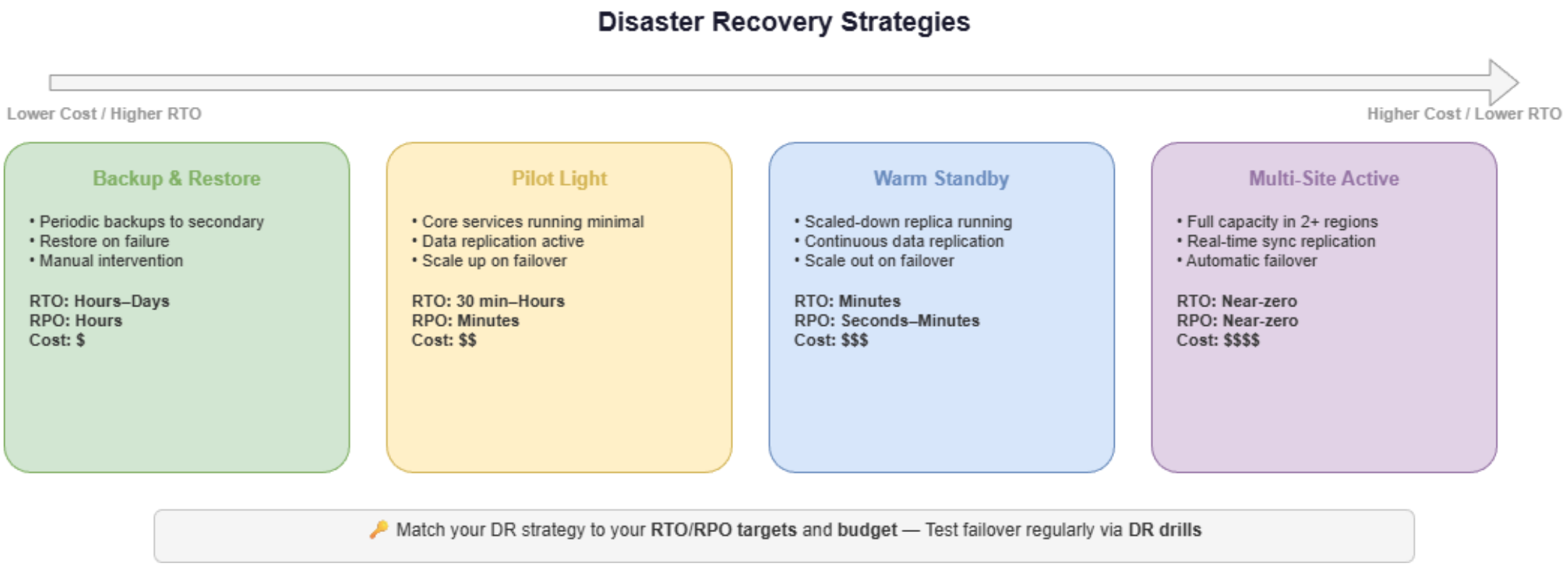
- **Availability Zone Outages**: Test region resiliency
- **Network Latency**: Simulate connectivity issues
- **Service Throttling**: Test quota and limit handling
- **Identity Failures**: Test credential expiration response

Combine Load and Failure Tests

Experiment	Example setup or acceptance criterion
Baseline	Three equal test instances near 50% capacity
Inject	Remove one instance; make a controlled dependency return 429
Bound work	At most 2 total attempts; 2 s deadline; 80 in-flight calls across replicas
Protect data	No duplicate side effects or lost acknowledged operations
Recover	Remove the fault; regain the flow SLO without a second surge

Illustrative test values—not Azure limits or measured results.

Disaster Recovery Strategies



Incident Response & Continuous Learning

Scale & Acceleration Layer

Making Reliability Repeatable

- Scale the reliability you designed across many workloads and teams
- Cut the effort with reusable modules (AVM), proactive assessment (APRL), and standardized landing zones

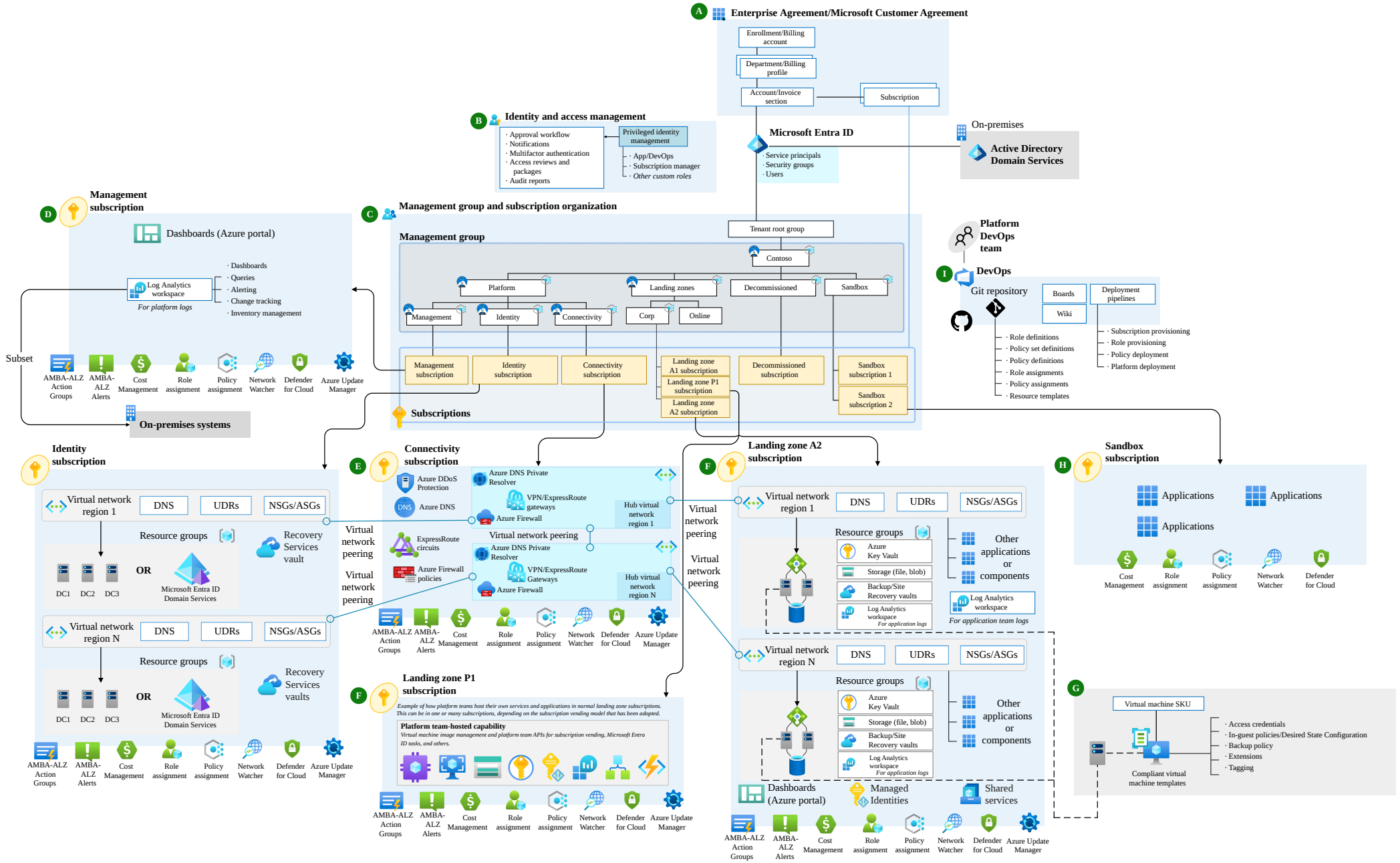
Azure Landing Zones

Secure Structure

- Azure Landing Zones create a secure, organized foundation for Azure environments.
- Enforce identity, network, and governance policies at scale.

Reliable Implementation

- A standardized environment supports consistent deployments.
- Simplifies resource management and reduces misconfigurations.



Microsoft Azure

Version: 2025-02-19

Reference Architectures & Guidance

Azure Architecture Center

- Best practices, reference architectures, design patterns

Mission-Critical Workloads

- Designing for max availability & performance

Well-Architected Workloads

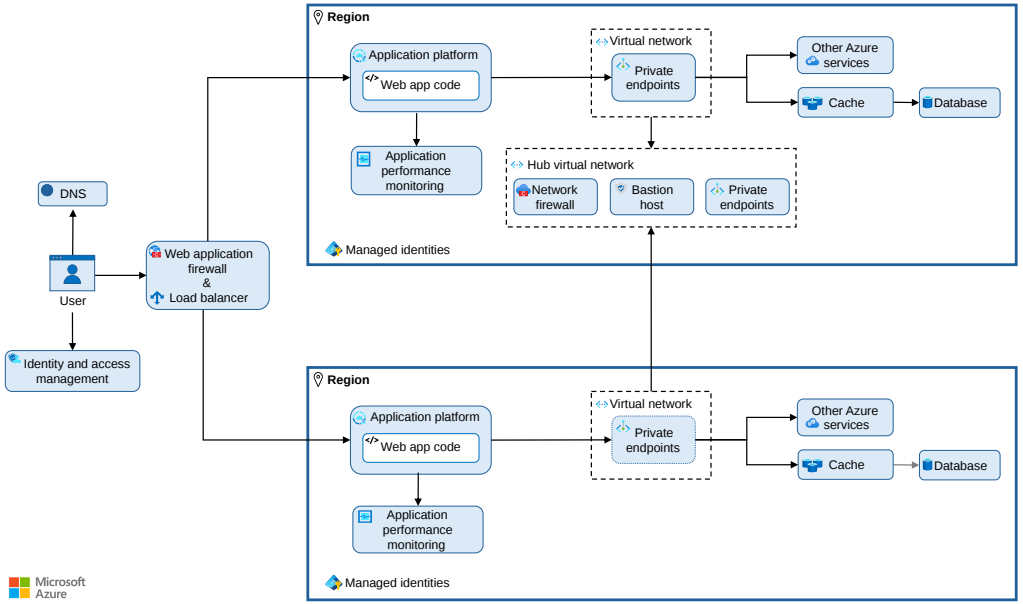
- Workload-specific WAF guidance & trade-offs

Enterprise Web App Patterns

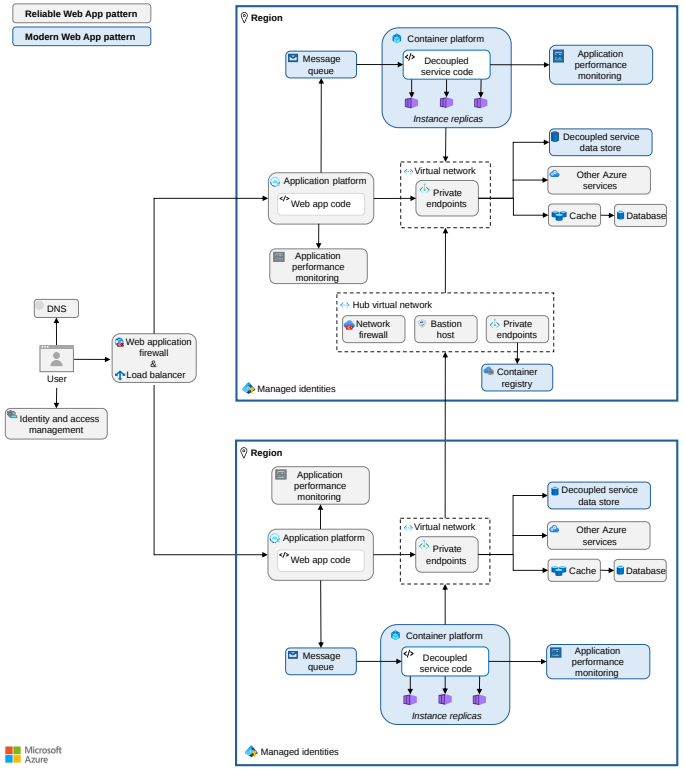
- Prescriptive architecture, code, and configuration

Web App Patterns

Reliable Web App



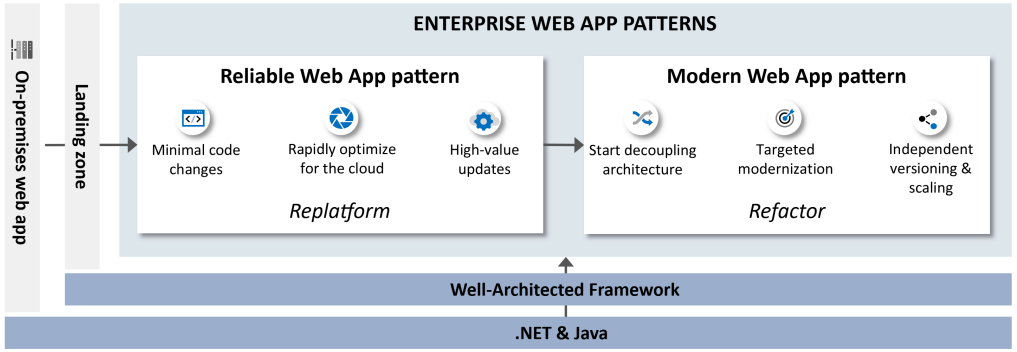
Modern Web App



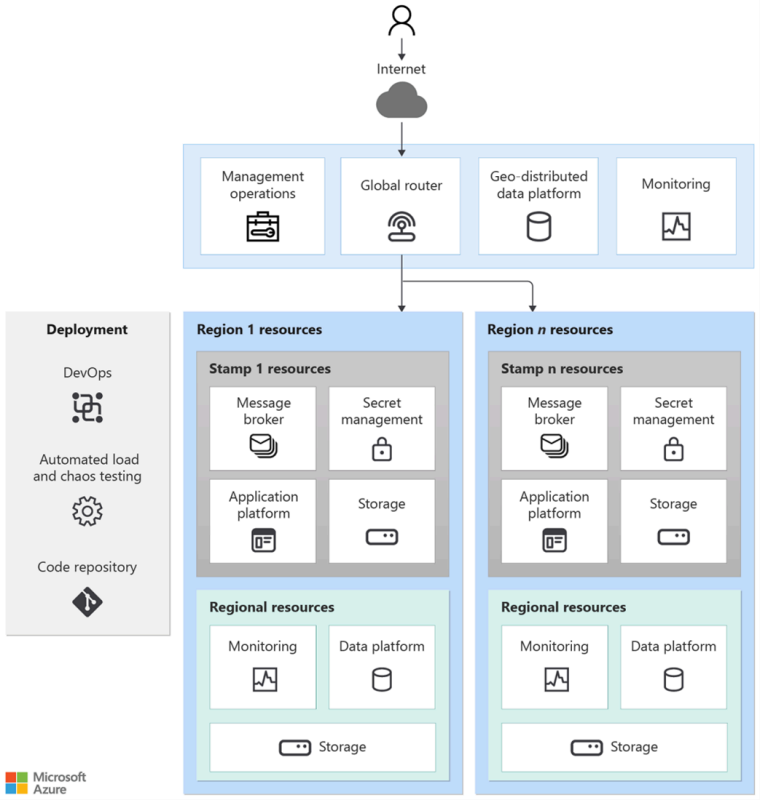
https://chris-ayers.com

Enterprise & Mission-Critical

Enterprise App



Mission-Critical



https://chris-ayers.com

WAF Design Guides (Design Essentials)

Prescriptive, **cross-pillar** guidance for specific practices — a newer addition to the framework.

- Regions & availability zones
- Handle transient faults
- Throttling
- Background jobs
- Build a monitoring system
- Health modeling
- Disaster recovery (multi-region)
- Incident management

> Bridge principles → implementation

Azure Verified Modules (AVM)

What is AVM?

- **Official Microsoft IaC initiative** — Bicep, Terraform
- **WAF-aligned** — review module and version defaults
- Configure AZs, monitoring, and hardening for your workload

Module Types

- 🏗️ **Resource** — Single resource, secure defaults
- 🎯 **Pattern** — Multi-resource proven architectures
- ⚙️ **Utility** — Reusable helpers (Preview)

> Reduce misconfigurations with standardized, production-tested templates

APRL: Azure Proactive Resiliency Library

What is APRL?

- **Curated catalog** of resiliency recommendations for 100+ Azure services
- **Azure Resource Graph queries** to find non-compliant resources
- **WAF-aligned** reliability best practices
- **Open source** — 75+ contributors

What's Inside

- 🏗️ **Resource-specific** configuration guidance
- 🔧 **Specialized workload** patterns
- 📊 **Ready-to-use ARG queries** for compliance
- 🛡️ **Assessment & review** tools

Azure Review Checklists

What They Are

- **Design validation tool** for Azure architectures
- **Community-driven** best practice checklists
- **Proactive issue detection** before deployment

Coverage

- Landing Zones, AKS, App Service, SQL
- Networking, Cost Optimization, DevOps
- AI Landing Zone, Spring Apps, SAP
- Standardized reviews across teams & projects

Conclusion

- **Design Principles**: Focus on resilience, recovery, operations, and simplicity
- **Know Your Flows**: Identify critical user journeys, set per-flow SLOs, and understand composite SLAs
- **Trade-offs**: Every decision impacts cost, security, operational excellence, and performance
- **Proactive Reliability**: Use FMA, dependency mapping, safe deployments, and tested DR plans
- **Continuous Improvement**: Chaos engineering, load testing, incident response, and blameless postmortems

Questions?

https://chris-ayers.com

Thank You!

Links

- Azure Well-Architected Framework
- APRL
- Azure Verified Modules
- Azure Review Checklists
- Enterprise Web Apps

Chris Ayers

Principal Software Engineer

Azure CXP AzRel

Microsoft

BlueSky: @chris-ayers.com

LinkedIn: - chris-l-ayers

Blog: https://chris-ayers.com/

GitHub: Codebytes

Mastodon: @Chrisayers@hachyderm.io

~~Twitter: @Chris_L_Ayers~~