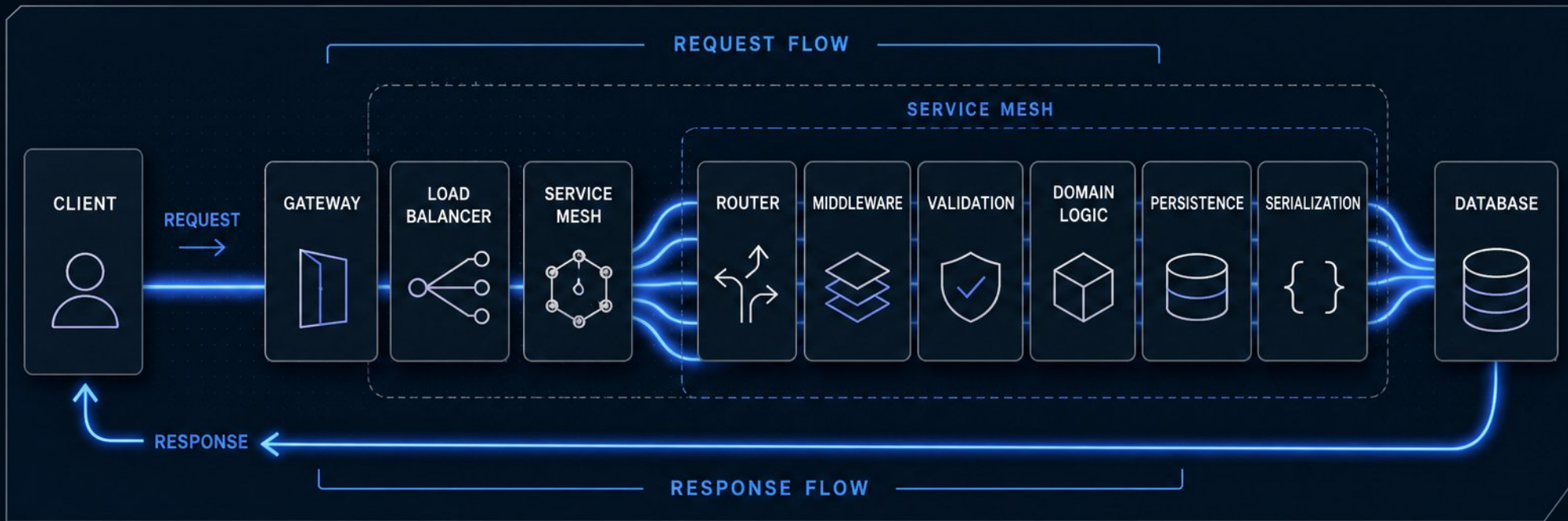


API Development Skills: Core Skills and Practice

- > Build production-grade skills across design, implementation, testing, security, operations
- > For backend and full-stack developers, software teams, and technical product managers
- > Production APIs must deliver correctness, consistency, evolvability, security, observability, performance
- > Lifecycle: contract, design review, implementation, testing, release, versioning, monitoring, deprecation
- > Failure modes: spec drift, inconsistent errors, missing authorization, unbounded retries, zombie endpoints
- > Course structure: concepts, patterns, labs, and delivery checklists



Background: How APIs Work and Where They Fit



> - Client-server request/response model; statelessness as the baseline for scaling



> - HTTP fundamentals: methods, status codes, headers, caching, safe vs. idempotent



> - Interaction styles by use case: REST, GraphQL, gRPC, events and webhooks



> - Request lifecycle: routing, middleware, validation, domain logic, persistence, serialization



> - Gateways, load balancers, and service meshes shape every request path



> - Internal, partner, and public APIs differ in contract stability and support

Choosing an API Style Per Boundary

Public & Partner
Clients



REST

HTTP / JSON

Internal Services



gRPC

Protobuf / HTTP/2

Multiple Clients,
Different Needs



GraphQL

Query One Graph

Producers &
Event Sources



**Events /
Webhooks**

Async & Push

- REST for public and partner APIs: universal clients, HTTP caching, curl-friendly debugging
- gRPC for internal service-to-service: binary protobuf, HTTP/2 multiplexing, compiler-checked contracts
- GraphQL when many clients need different slices of one graph; watch caching and N+1 cost
- tRPC for TypeScript monorepos; events and webhooks for async and push flows
- Pick per boundary, not per company: REST edge plus gRPC internally is normal
- Style choice is organizational: who is on the other side, do you own both ends?

Contract-First API

Design and Resource Modeling

- – Model resources, capabilities, and consumer workflows—not database tables
- – OpenAPI 3.1 spec as single source of truth, aligned with JSON Schema 2020-12
- – One spec drives docs, codegen, mocks, and runtime validation
- – Consistent naming, HTTP methods, cursor pagination, filtering, and field selection
- – Errors use RFC 9457 problem+json: type, title, status, detail, instance
- – Evolve additively; deprecate explicitly; review with lint, diff, and consumer sign-off

OpenAPI 3.1

openapi: 3.1.0

jsonSchemaDialect: "https://json-schema.org/draft/2020-12/schema"

Resources

GET	/customers	List customers ?cursor={cursor}&limit={limit} &filter={filter}&fields={fields}
GET	/customers/{id}	Get customer ?fields={fields}
POST	/customers	Create customer
PATCH	/customers/{id}	Update customer
DELETE	/customers/{id}	Delete customer

Error: RFC 9457 problem+json

```
{
  "type": "https://example.com/probs/invalid-input",
  "title": "Invalid input",
  "status": 400,
  "detail": "The 'email' field must be a valid email address.",
  "instance": "/customers"
}
```

Schemas (JSON Schema 2020-12)



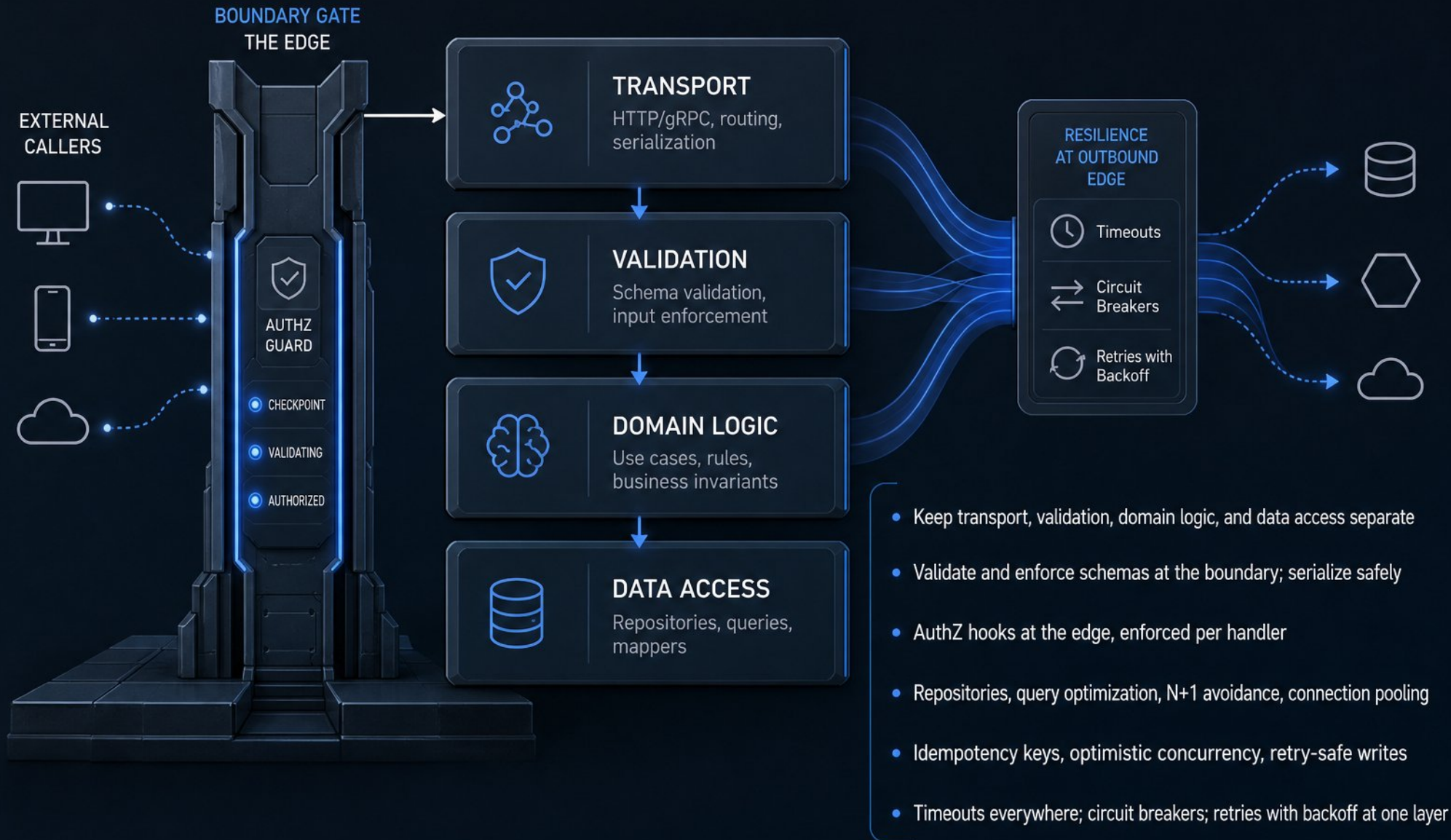
Consistent naming & HTTP methods

Cursor pagination, filtering, field selection

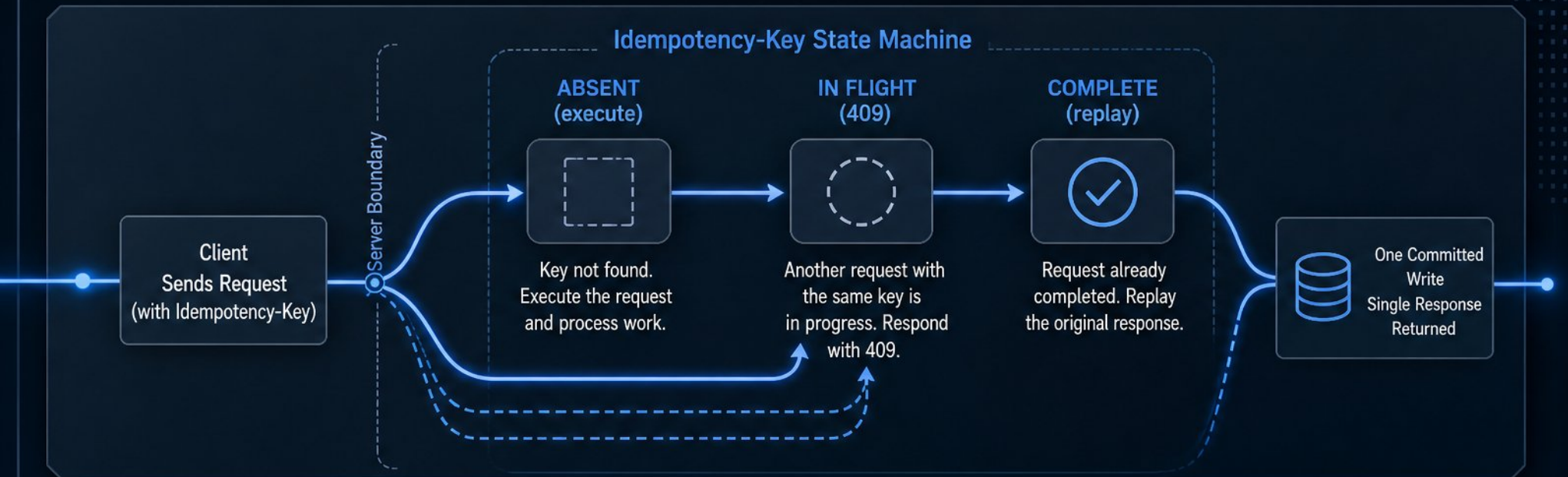
Problem details: type, title, status, detail, instance

CONTRACT CARD: API CONTRACT
AS SINGLE SOURCE OF TRUTH

Implementation Patterns and Backend Architecture



Safe Retries and Idempotent Write Design



- - Retries are inevitable; at-least-once delivery means duplicates are certain, not hypothetical
- - Client generates Idempotency-Key; server reserves it atomically before doing any work
- - Three states, not two: absent (execute), in flight (409), complete (replay)
- - Scope keys to caller and endpoint; store fingerprint, reject mismatches with 422
- - Commit business write and key record in one transaction; lease stale in-flight rows
- - Never cache 5xx; retain keys past the longest retry window, commonly 24 hours

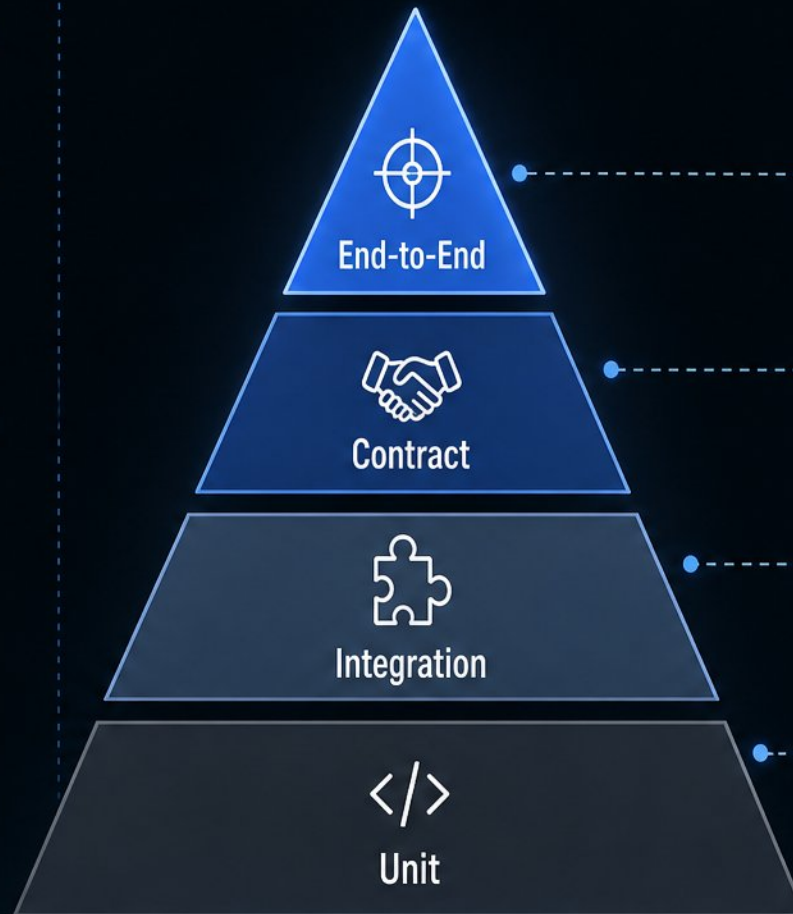
Security, Authentication, and Authorization

- - OWASP API Top 10 (2023): BOLA #1, then broken authentication, BOPLA
- - Check object-level authorization in every handler; return 404, not 403
- - OAuth 2.0/OIDC: Authorization Code + PKCE, exact redirect URIs, short tokens, refresh rotation
- - Validate JWTs with algorithm allowlist plus iss, aud, exp, nbf
- - Avoid localStorage; use HttpOnly cookies or a backend-for-frontend
- - TLS everywhere, managed secrets, DPoP or mTLS, rate limiting



Testing, Contract Verification, and CI Quality Gates

- Test pyramid: unit, integration, contract, and end-to-end each catch distinct failures
- Spec-driven contract tests with Schemathesis, Dredd, and example validators
- Consumer-driven contracts with Pact; bi-directional testing when OpenAPI exists
- Mock servers let consumers build before the provider is finished
- CI gates: spec linting, schema validation, security scans, and coverage floors
- Run server-free checks first, then live checks needing a running service



CI QUALITY GATES



Spec linting



Schema validation



Contract tests



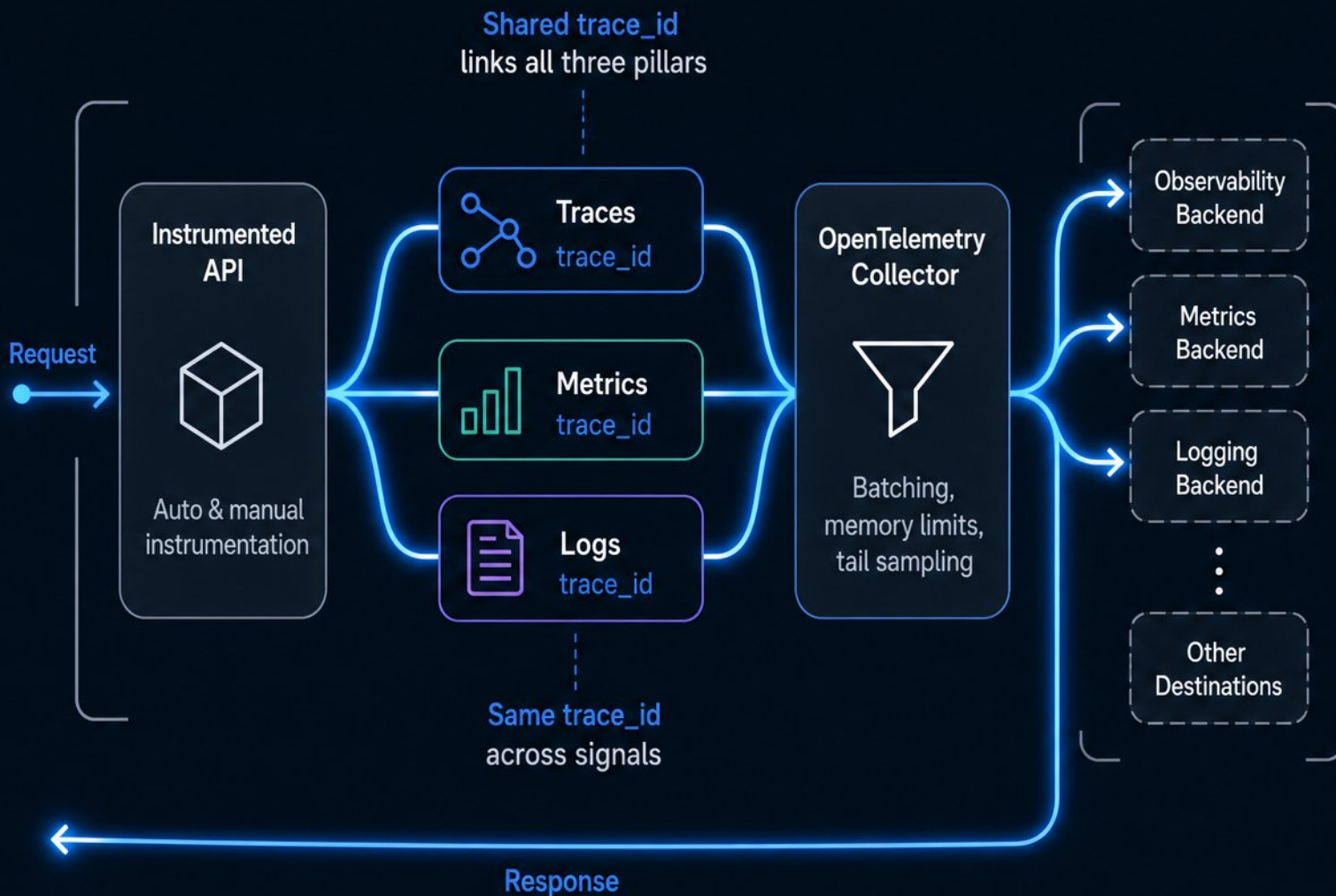
Security scans



Coverage floors

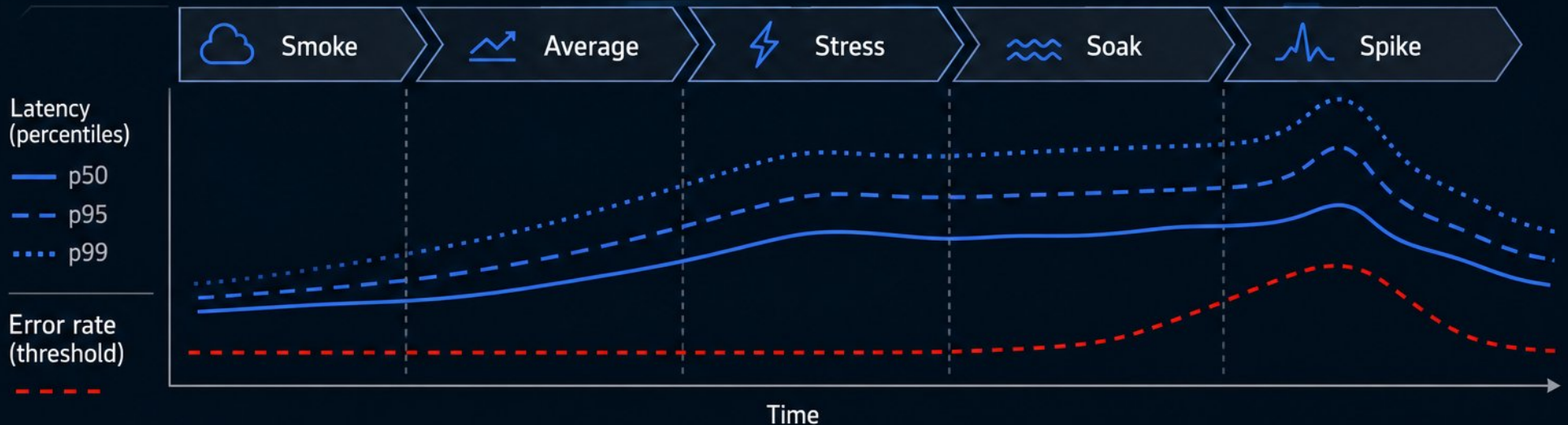
Observability with OpenTelemetry

- - Instrument once with OTel; export traces, metrics, and logs anywhere via OTLP
- - Shared trace_id links all three pillars for fast root-cause analysis
- - Auto-instrument frameworks, databases, and HTTP clients; add manual spans for business logic
- - Track golden signals: latency percentiles, traffic, errors, and saturation
- - Collector handles batch, memory limits, and tail sampling
- - Sample 1-10% of normal traces; keep 100% of errors and slow traces



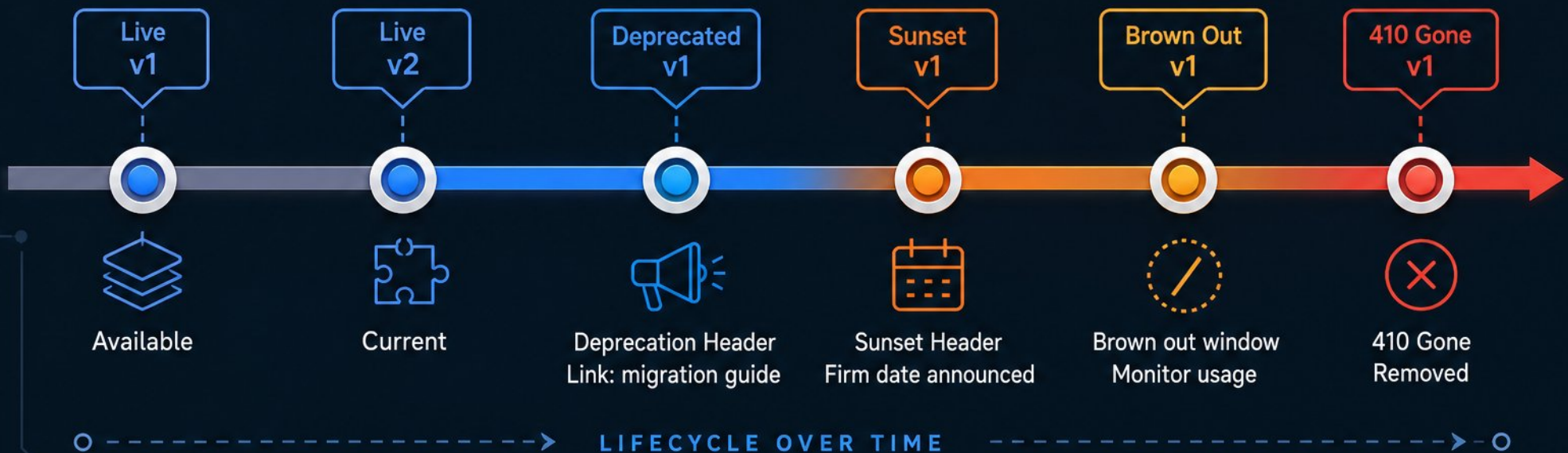
Performance, Caching, and Load Testing

	Measure first: bottlenecks are often slow external calls, not the database	
	Quick wins: parallelize independent calls and cache stable data with clear TTLs	
	Caching layers: HTTP ETags and cache-control, app caches, disciplined invalidation	
	Define SLOs first, then run smoke, load, stress, soak, and spike tests	
	Model think time, arrival rate, auth, and varied representative data	
	Wire k6 thresholds into CI so regressions fail the pipeline	



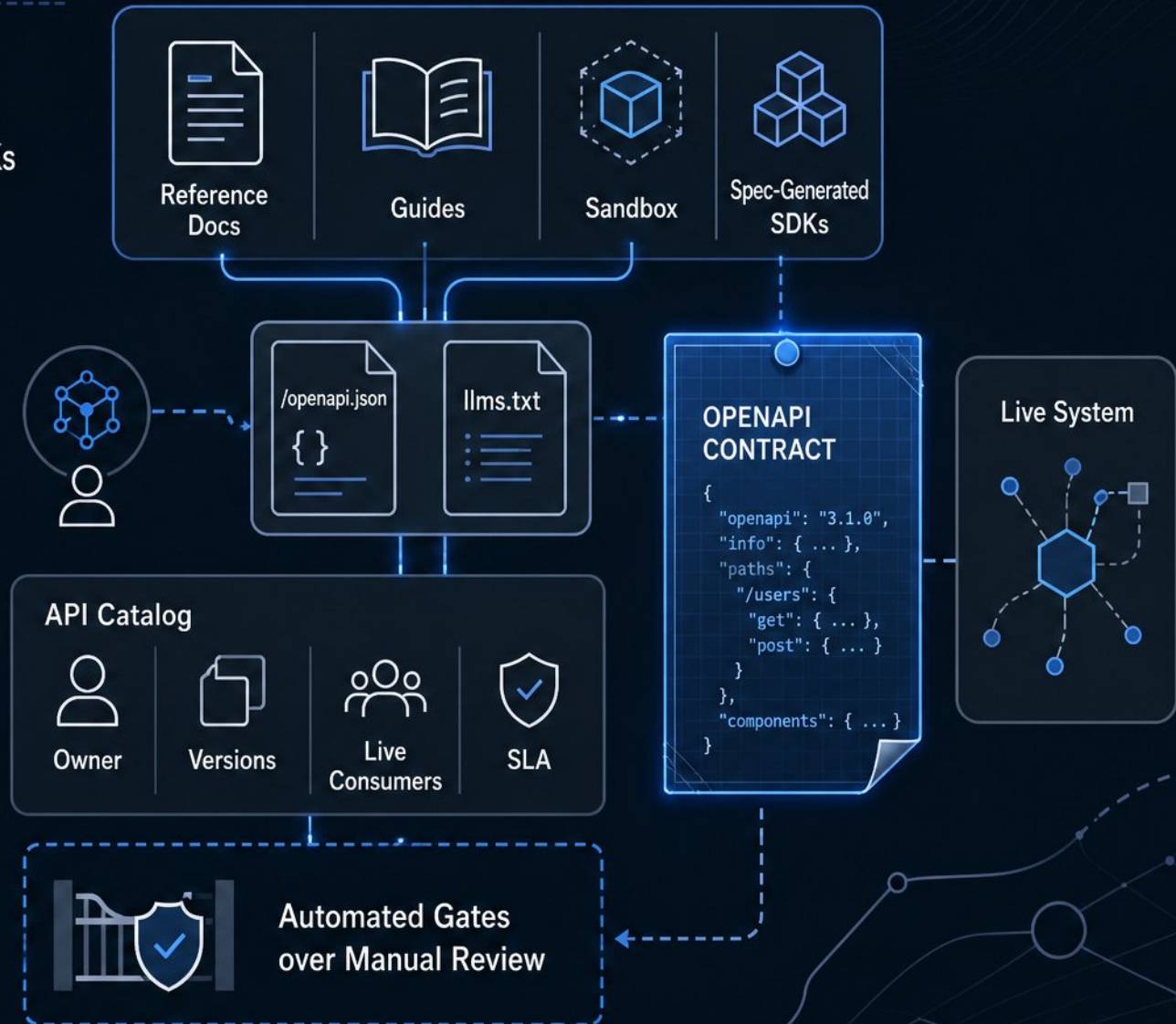
Versioning, Deprecation, and Lifecycle Management

- ▶ URI path versioning is the default; header and date-based suit mature programs
- ▶ Version only breaking changes: removals, renames, type, status, or default shifts
- ▶ Additive changes ship without a version bump; keep at most 2-3 live versions
- ▶ Emit Deprecation and Sunset headers plus a Link to the migration guide
- ▶ Lifecycle: announce with a firm date, monitor usage, brown out, then 410 Gone



Documentation, Developer Experience, and Governance

- - Docs are a product deliverable: guides, sandboxes, spec-generated SDKs
- - Serve `/openapi.json` and `llms.txt` so agents discover the API
- - Agent-readiness helps humans too: idempotency keys, clear operationIds, scoped tokens
- - API catalog tracks owner, versions, live consumers, and SLA
- - Automated gates over manual review; measure breaking-change escape rate



Hands-On Practice:

Designing and Shipping an API Slice

	- Lab scenario: design a small, realistic API for a defined consumer	
	- Produce an OpenAPI 3.1 contract; lint it; review errors, pagination, and idempotency	
	- Implement endpoints with validation, auth, object-level authorization, and RFC 9457 errors	
	- Add contract and integration tests; pass CI lint and breaking-change diff gates	
	- Instrument with OpenTelemetry; confirm one trace spans the full request path	
	- Run a mini load test, fix one bottleneck, then version and document for a consumer	



Team Delivery Checklist and Next Steps

READINESS ACROSS SEVEN DIMENSIONS



RELIABILITY

Design for resilience and steady behavior.



OBSERVABILITY

See what matters, end to end.



ROLLBACK

Safe to revert, fast to recover.



SECURITY

Secure by design, least privilege.



CAPACITY

Sustainable performance at scale.



INCIDENT RESPONSE

Prepare, detect, respond, learn.



DATA

Protect, back up, and restore confidently.

- ✓ - Definition of done: contract, security, tests, observability, docs, rollback plan
- ✓ - Readiness spans seven dimensions, from resilience to tested restores
- ✓ - What fails most: untested rollback, missing timeout, never-restored backup
- ✓ - Watch for no owner, inconsistent standards, undocumented breaks, zombie endpoints
- ✓ - Adopt a living style guide, templates, review cadence, automated gates

API LIFECYCLE RAIL



PersonWise.

PersonWise • AI digital-human course platform



Scan the code or click anywhere to watch this course live, with voice Q&A

personwise.ai/t/51cae6e2/public