LDX3 LONDON · JUNE 2026

Your problem isn't the monolith. It's the data.

Six years of monolith. Eighteen months of microservices.

Architecture endures when teams align on data lifecycles, not just APIs.

FIVE DISCIPLINES · WE'LL WALK ALL OF THEM

Rot doesn't start at the deployment boundary

It starts at the schema

🙋 Raise your hand if **none** of these are true in your ecosystem.

🔓 01

Writes to any table are allowed.

No persistence-layer gatekeeper.

🔗 02

Cross-domain joins in hot paths.

Read paths span domains.

⏳ 03

Retention rules vary across teams.

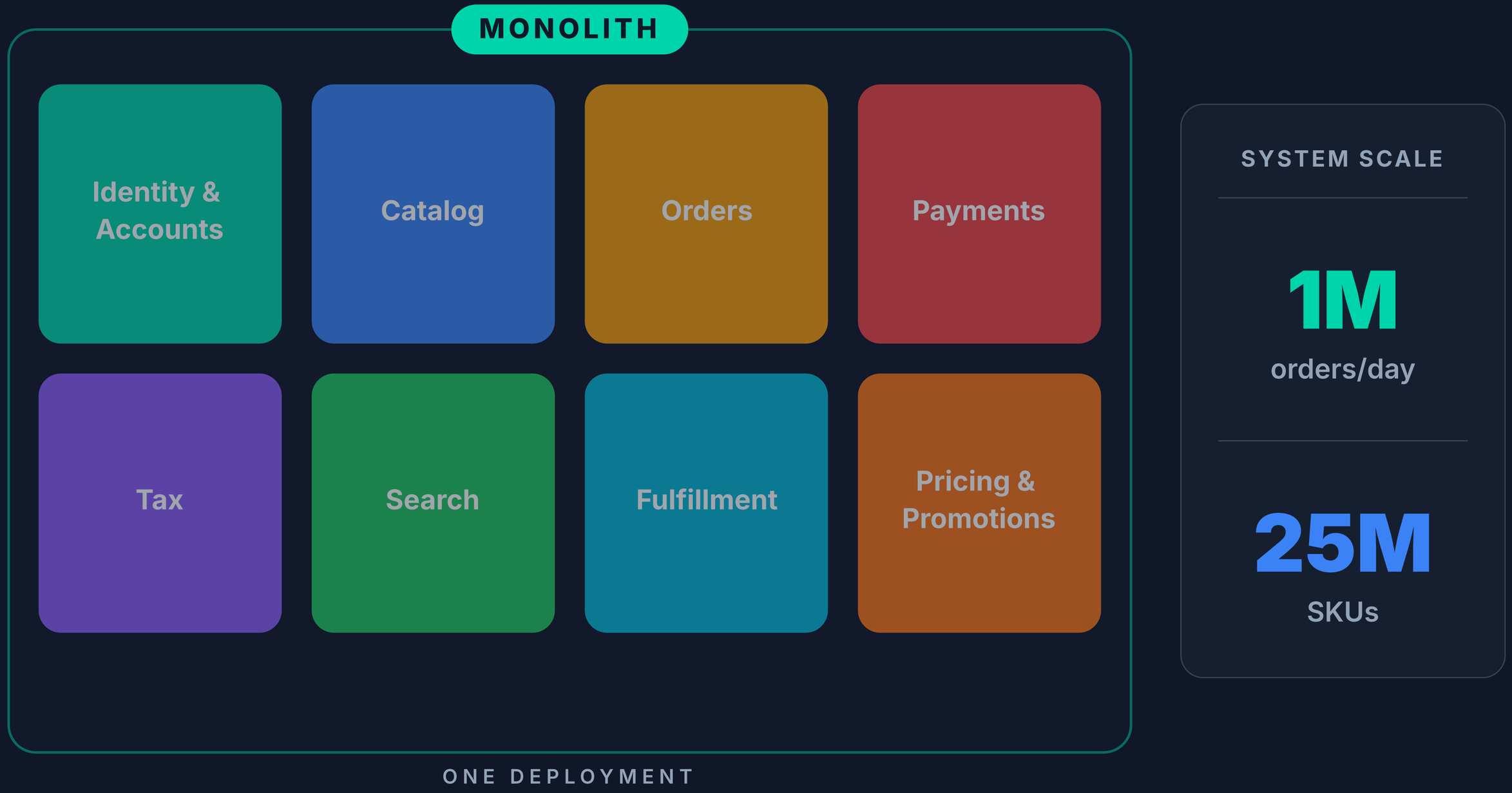
Per-team, per-table policies.

💬 04

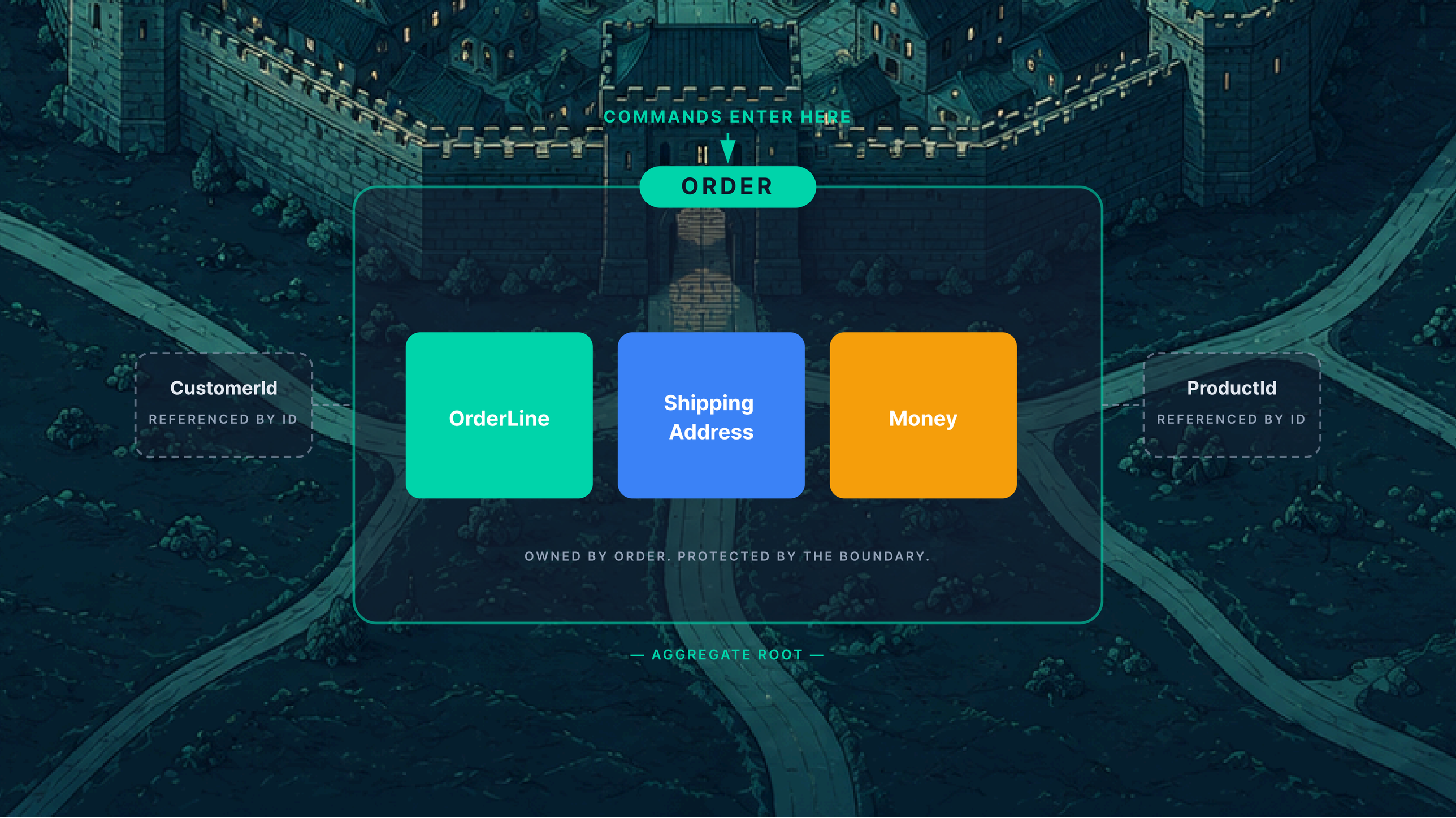
Shared database. Different vocabularies.

One schema, multiple consuming teams.

Penultimate Markets, 2022. One deployment. 120 engineers. 8 markets.


MONOLITH
Identity & Accounts
Catalog
Orders
Payments
Tax
Search
Fulfillment
Pricing & Promotions
ONE DEPLOYMENT
SYSTEM SCALE
1M
orders/day
25M
SKUs

Aggregates own their invariants.


COMMANDS ENTER HERE
ORDER
CustomerId
REFERENCED BY ID
OrderLine
Shipping Address
Money
ProductId
REFERENCED BY ID
OWNED BY ORDER. PROTECTED BY THE BOUNDARY.
— AGGREGATE ROOT —

01 · OWNERSHIP

What this looks like in practice

Somewhere in the tax codebase…

⊗ ASKING

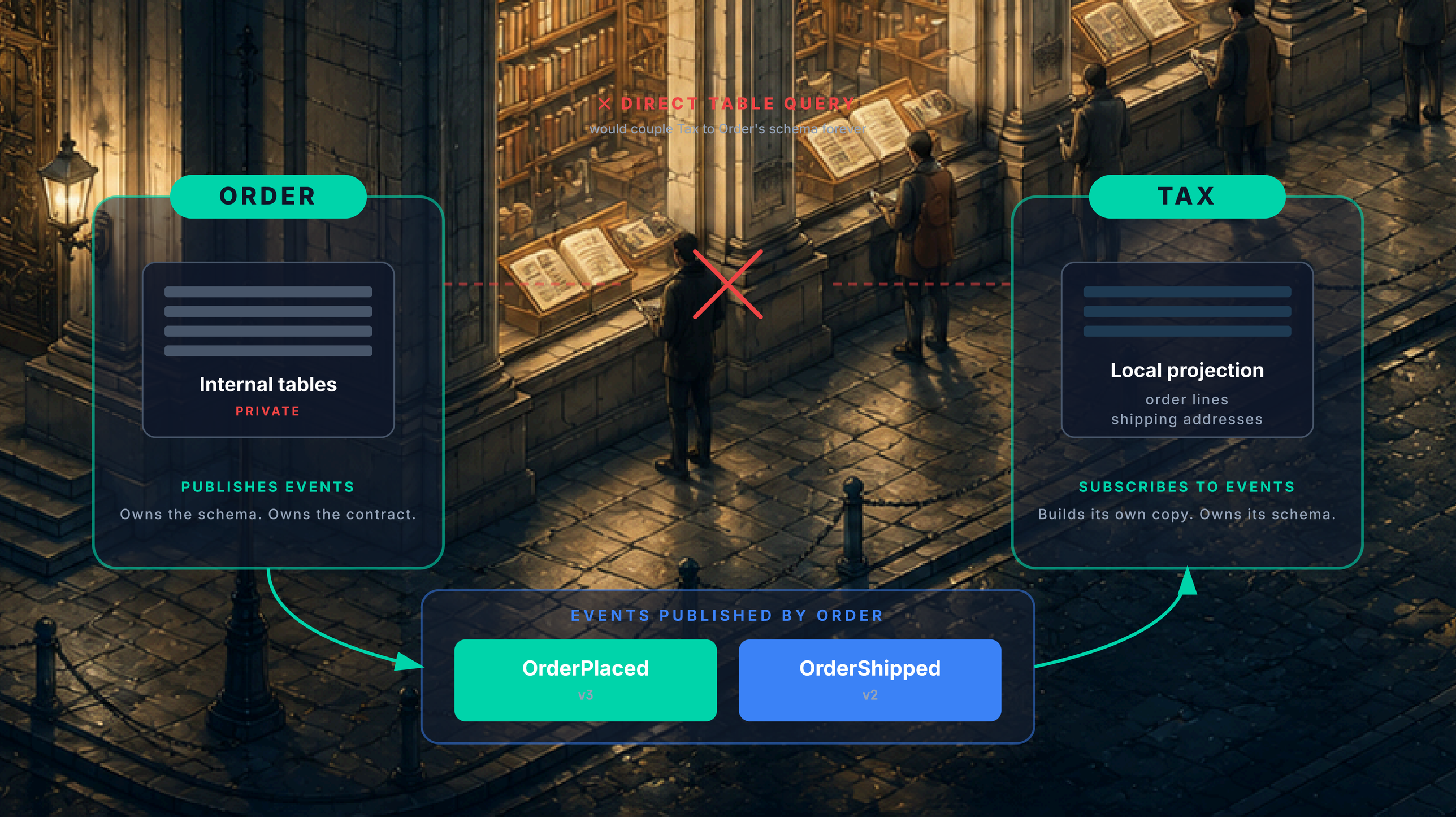
```
var order = orderRepo.findById(id);
if (order.state() ≠ DRAFT)
    throw new IllegalStateException();

var subtotal = order.lines().stream()
    .map(l → l.price().multiply(l.qty()))
    .reduce(ZERO, BigDecimal::add);

order.lines().forEach(l → {
    var rate = rateFor(order.address());
    l.setTaxRate(rate);
    l.setTaxAmount(l.price().multiply(rate));
});

order.setSubtotal(subtotal);
order.setState(TAXED);
orderRepo.save(order);
```

✓ TELLING

```
orderService.applyTaxes(
    orderId, taxBreakdown);
```

Tell, don't ask.

Other teams consume contracts, not tables.


✕ DIRECT TABLE QUERY
would couple Tax to Order's schema forever
ORDER
Internal tables
PRIVATE
PUBLISHES EVENTS
Owns the schema. Owns the contract.
TAX
Local projection
order lines
shipping addresses
SUBSCRIBES TO EVENTS
Builds its own copy. Owns its schema.
EVENTS PUBLISHED BY ORDER
OrderPlaced
v3
OrderShipped
v2

02 · CONTRACTS

The contract is the boundary.

INTERNAL · ORDER'S PRIVATE TABLES

```
-- private to orders-team
CREATE TABLE orders (
  order_id       UUID        PRIMARY KEY,
  customer_id    UUID        NOT NULL,
  state          order_state NOT NULL,
  placed_at      TIMESTAMPTZ NOT NULL,
  currency       CHAR(3)     NOT NULL,
  total_minor    BIGINT      NOT NULL,
  fraud_score    SMALLINT,         -- internal
  channel        TEXT,
  channel_meta   JSONB,            -- internal
  fulfillment_id UUID,
  ml_segment     TEXT,             -- internal
  updated_at     TIMESTAMPTZ NOT NULL
);
CREATE TABLE order_lines       ( ... );
CREATE TABLE order_status_log  ( ... );
CREATE INDEX ON orders (customer_id, placed_at DESC);
```

PUBLIC · ORDERPLACED EVENT

```
syntax = "proto3";
package penultimate.orders.events.v3;

// Owner: orders-team
// Compatibility: BACKWARD (Buf in CI)
message OrderPlaced {
  reserved 9, 10;   // removed: discount_code, promo_id

  string             order_id        = 1;
  string             customer_id     = 2;
  int64              placed_at_ms    = 3;
  string             currency        = 4;
  Address            shipping        = 5;
  repeated OrderLine lines           = 6;
  Money              total_amount    = 7;
  Source             source          = 8;
  optional string    idempotency_key = 11;
}
// internal-only fields excluded:
//   fraud_score, channel_meta, ml_segment
```

Versioned. Owned by orders-team. Published with a guarantee.

03 · CI FITNESS

03 · CI FITNESS

Fitness functions for data boundaries.

```java
@AnalyzeClasses(packages = "com.penultimate")
public class DataBoundaryFitnessFunctions {

  @ArchTest
  static final ArchRule no_cross_domain_persistence_access =
    noClasses().that().resideInAPackage("..tax..")
      .should().accessClassesThat()
      .resideInAPackage("..orders.persistence..")
      .because("Tax reads OrderPlaced events. Tax does not query Order's tables.");

  @ArchTest
  static final ArchRule writes_only_via_aggregate_root =
    noClasses().that().resideOutsideOfPackage("..orders.domain..")
      .should().callMethod(OrderRepository.class, "save", Order.class)
      .because("Order's invariants live in Order. No exceptions.");

  @ArchTest
  static final ArchRule events_versioned_and_immutable =
    classes().that().areAnnotatedWith(DomainEvent.class)
      .should().haveSimpleNameEndingWith("V1")
      .orShould().haveSimpleNameEndingWith("V2")
      .andShould().haveOnlyFinalFields();
}
```

Three rules. One CI build. Architecture, as code.

03 · CI FITNESS

Other fitness functions you'll add

1 Schema migration linter

Expand / contract enforcement.

2 Contract stability test

Pact, schema-registry compatibility.

3 Cross-aggregate join detector

SQL log analysis or query plan inspection.

4 Event schema validation

In CI, against registry.

5 Lifecycle / retention assertions

Per-table TTL declared, enforced.

+

YOUR NEXT ONE

04 · SAFE MIGRATIONS

04 · SAFE MIGRATIONS

Migrations expand. Then contract.

Every step is reversible. None of them is a flag day.

01 EXPAND

Add the new column nullable. Dual-write from the app.

↺ *Drop the column. Old reads were never touched.*

02 BACKFILL

→ Populate new column from old. Idempotent, restartable.

↺ *Re-run. Or pause and resume; the script is a fact, not a flag.*

03 SWITCH READS

Reads move to the new column behind a feature flag.

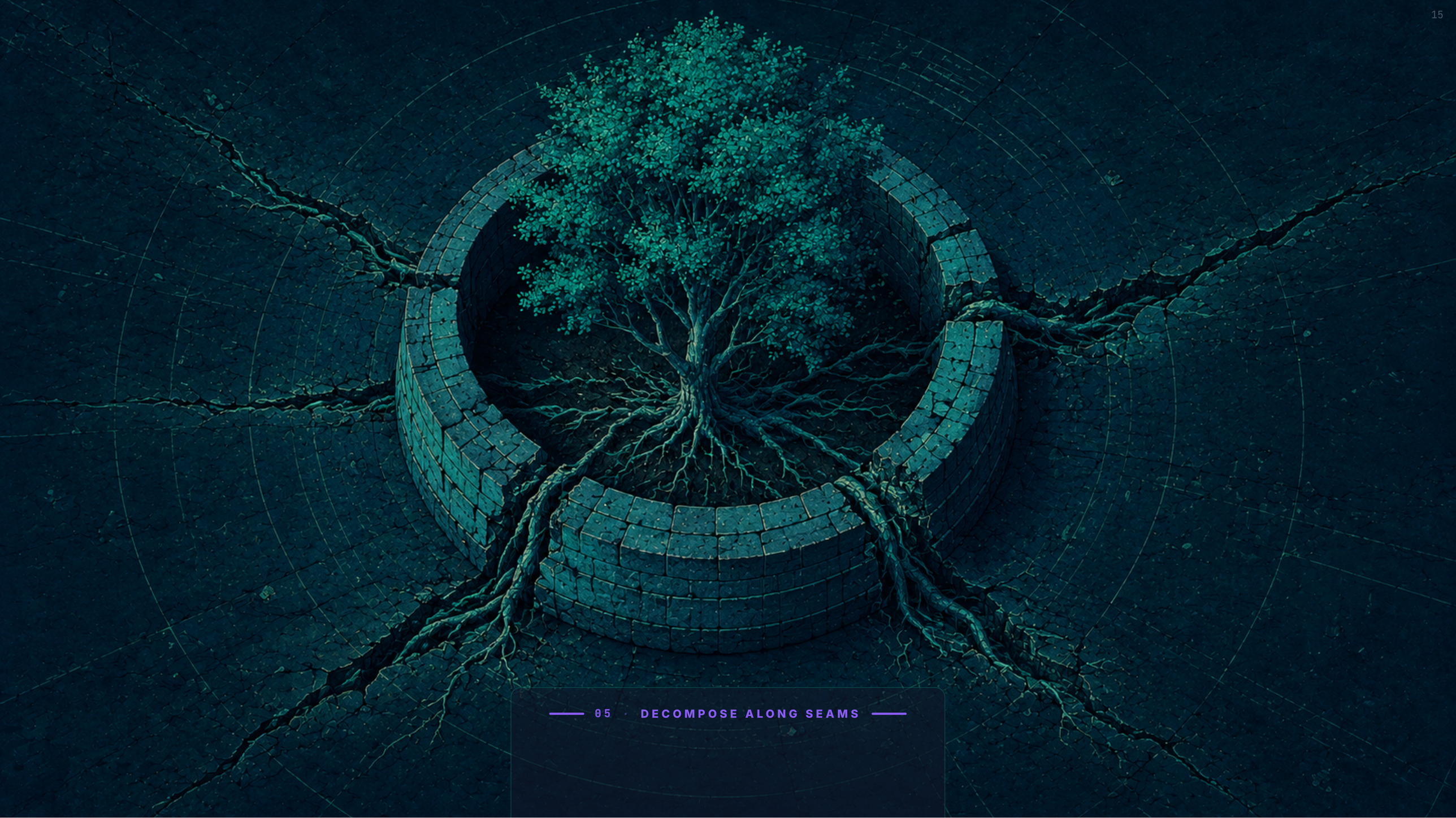
↺ *Flip the flag back. Both columns still hold the truth.*

04 CONTRACT

→ Stop writing the old column. Drop it in a later deploy.

↺ *Only safe once no reader has touched old for a full retention window.*

Enforced in CI: an Expand without its Contract is a failed build.



05 · DECOMPOSE ALONG SEAMS

05 · DECOMPOSE ALONG SEAMS

The pressure was not traffic. It was lifecycle.

Three teams. Three incidents. Three different shapes of pressure.

	Search Y5 Q1 · PUBLISH HIT 800 MS	Tax Y5 Q3 · RETENTION DRIFT	Payments Y6 Q1 · CASCADE OUTAGE
CONSISTENCY	●○○○○ *eventual*	●●●●● *strong*	●●●●● *strong*
LATENCY	●●●●● *tight*	●○○○○ *relaxed*	●●●●● *tight*
RETENTION	●○○○○ *short*	●●●●● *7 years*	●●●○○ *medium*
ISOLATION	●●○○○ *regional*	●●●○○ *regulated*	●●●●● *bulkheaded*
	→ Cache at the edge.	→ Append-only ledger.	→ Bulkhead for blast radius.

05 · DECOMPOSE ALONG SEAMS

Three shapes of cut.

Same colors. Same domains. Same teams. Different boundaries.

YEAR 4 8 domains, 1 deployment → YEAR 7 3 extracted, 5 still together

Search

YEAR 5 · MOVE 1

AGGREGATE BECOMES EXTRACTION BOUNDARY.

Already consumed events. Already had its own projection. The boundary was already there — extraction was a deployment change, not a redesign. Same code, different transport.

Tax

YEAR 6 · MOVE 2

READ MODEL BECOMES PUBLISHED PRODUCT.

The Tax read model existed before extraction. After: a name, a version, an SLA, a deprecation window, consumers it didn't have before. Implicit contracts made explicit.

Payments

YEAR 6–7 · MOVE 3

PROCESS MANAGER REPLACES TRANSACTION.

A five-table transaction became a saga with explicit compensation. Choreography for simple paths, orchestration for the multi-step ones. Never call a saga a transaction in front of a database person.

05 · DECOMPOSE ALONG SEAMS

Duplication owned by a contract is not debt. It's autonomy.

IN 2026

The cost compounds.

57%

AGENT TOOLS

over-expose data through naively converted APIs.

+322%

AI-ASSISTED CODE

privilege-escalation paths. +153% design flaws.

4×

COMMIT VOLUME

more commits in fewer, larger PRs. Review attention didn't scale.

Your Monday Morning Actions

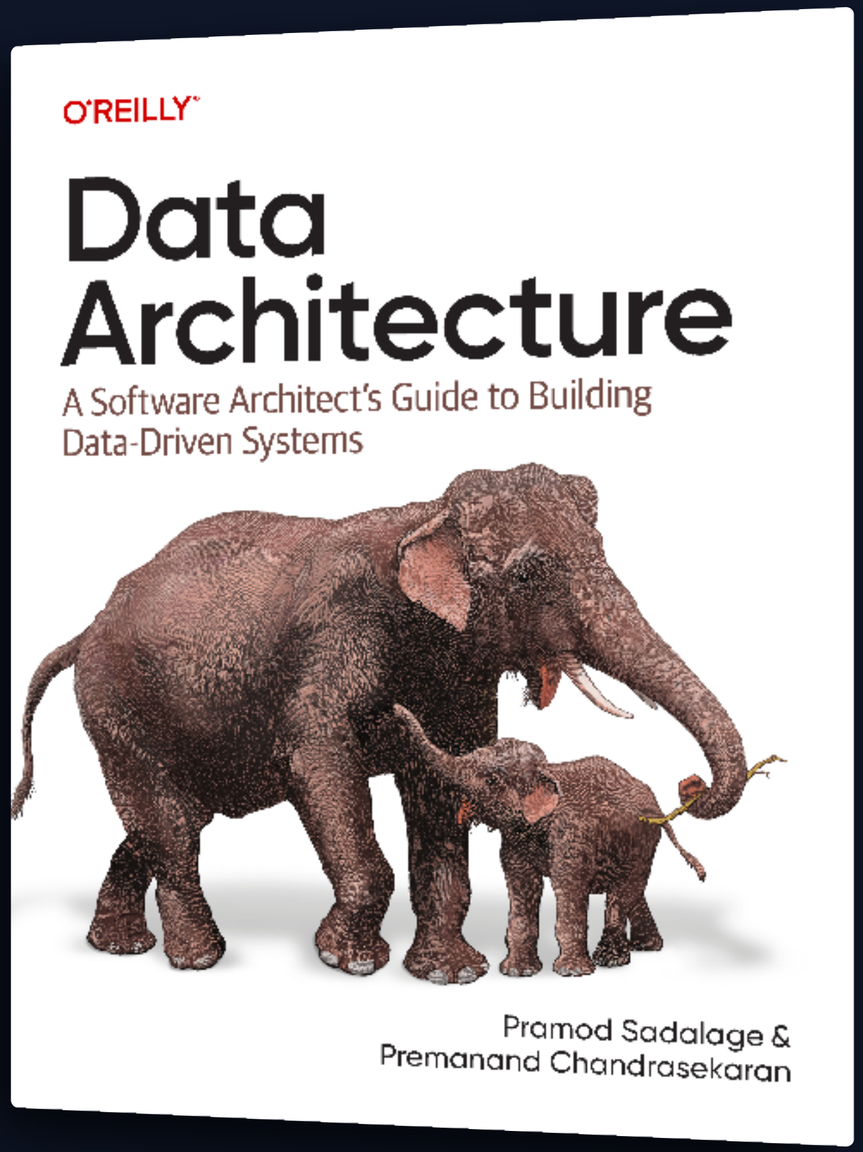
1. **Encode ownership.**
   Which team owns which aggregates? Write it down this week.

2. **Publish contracts.**
   Events and read models, versioned and owned.

3. **Verify in CI.**
   Fitness functions for boundaries, contracts, migrations.

4. **Make migrations safe by design.**
   Expand, then contract. Always.

5. **Decompose along seams when lifecycle pulls.**
   Not when traffic pulls. Not when fashion pulls.



Your monolith will stay modular longer.
Your decompositions will be evolutionary, not traumatic.

WHAT THAT LOOKED LIKE AT PENULTIMATE

2/wk → 50/day
DEPLOY FREQUENCY

−68%
ON-CALL ESCALATIONS

~3 mo
EXTRACTION → GA

0
EXTRACTION-RELATED OUTAGES

Year 4 → Year 7. Same headcount. Same teams.