

— THE BOTTLENECK

Authorization after an **acquisition**

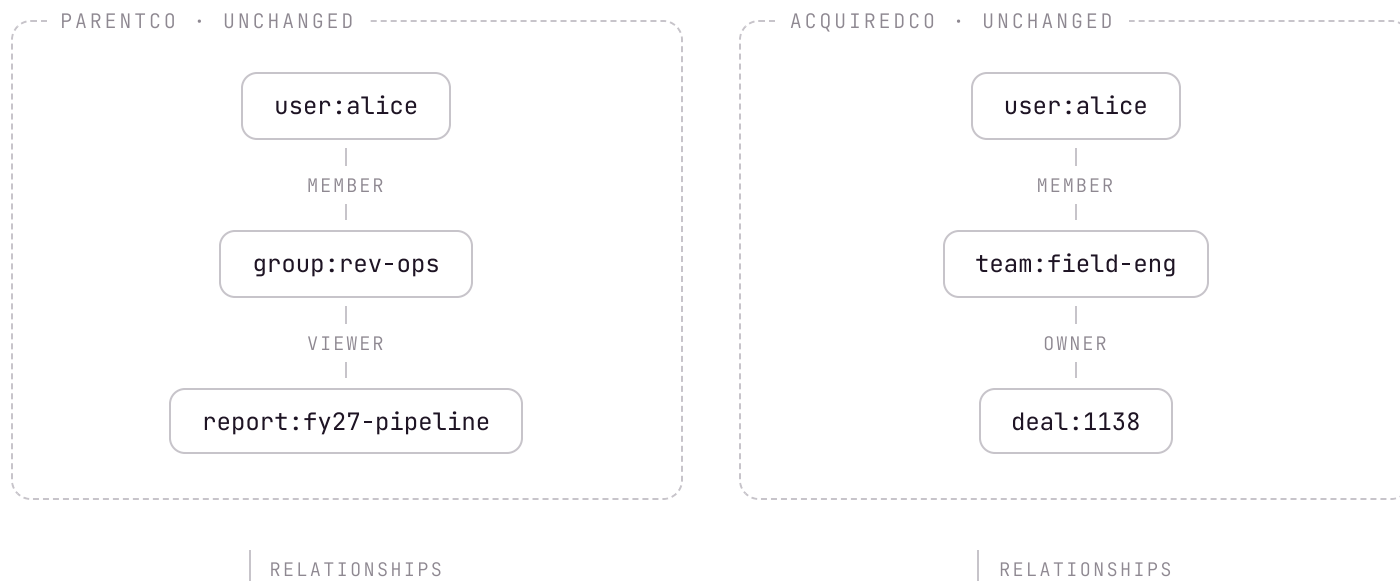
The permissions merge is usually what holds up the integration. A relationship-based layer doesn't merge the two systems — it lets you stop waiting for the merge to give people correct access.

Day one after close, nearly every integration milestone — a shared customer view, one support queue, joint reporting, a combined data room — reduces to a question neither system can answer alone: **can this person see this thing?**

Two paths are usually on the table. **Re-platform:** pick a surviving system, rebuild the other inside it, and deliver all the value at the end of a 12–18 month program. **Bilateral sync:** wire the two directories and apps together with group mappings, ship in weeks, then watch the mappings drift until nobody can say with confidence who has access to what.

The first is slow, the second is unauditably, and both put the authorization model on the critical path.

The third option is to **lift the decision out of both applications** while leaving both applications where they are.

SpiceDB — one relationship graph, **two namespaces**

```
parentco/report:fy27-pipeline#viewer@parentco/group:rev-ops#member
```

```
acquiredco/deal:1138#owner@acquiredco/team:field-eng#member
```

```
acquiredco/deal:1138#viewer@parentco/group:rev-ops#member ← the merger, as one write
```

```
CheckPermission(alice, view, acquiredco/deal:1138)
```

```
→ ALLOWED • via parentco/group:rev-ops
```

```
LEGACY CHECK, RUN IN PARALLEL
```

```
→ ALLOWED • diff clean, safe to cut over
```

Both applications keep their own storage, their own UI, and their own release cycle. **What moves is the permission decision.**

Granting the ParentCo revenue team access to an AcquiredCo deal is a single relationship write, not a schema migration on either side.

– Two permission models **can coexist**

RBAC, per-object ACLs and tenant scoping are all expressible in a Zanzibar-style schema. You port each side roughly as-is into its own namespace, run them side by side, and converge the model later as a schema change — not as a re-platform.

– Cutover is testable **before it's real**

Backfill both systems' relationships, run checks against SpiceDB in parallel with the existing logic, and diff the answers under production traffic. You cut over per-application, once the diff is clean — a far easier ask of a security team than a flag day.

– Cross-company access **becomes data**

"Give the acquired field team read access to ParentCo accounts" is a relationship written at runtime, reversible in one call. No bidirectional group sync to build, and no drift between two copies of the truth.

– The divestiture case is **the same machinery**

If part of the combined business is later carved out, separating a subgraph of relationships is tractable. Unpicking permissions that were fused into two applications' role tables generally is not.

HOW THE WORK SEQUENCES

PHASE 0	Reconcile identity One stable subject identifier per human across both directories. This has to land first, and it is not something an authorization layer can do for you.	YOUR IDP TEAM
PHASE 1	Recover the real models Extract the effective permissions from each side — including the rules living in SQL joins and application conditionals. This is the archaeology, and it is where these projects usually slip.	JOINT
PHASE 2	Write one schema, two namespaces Both models expressed in SpiceDB without changing either one's semantics, plus the specific cross-company grants the integration plan calls for.	AUTHZED-LED
PHASE 3	Backfill and shadow-read Import existing relationships, then run SpiceDB checks alongside the live authorization path and diff every answer. Nothing user-visible changes during this phase.	JOINT
PHASE 4	Cut over, one application at a time Each service switches to SpiceDB as its source of truth once its diff is clean. Cross-company access can be granted from this point without waiting on the rest of the integration.	YOUR TEAMS

WHAT THIS DOES NOT SOLVE

- **Identity reconciliation.** SpiceDB needs a stable subject ID. Deduplicating humans across two IdPs, federating SSO and retiring one directory remain your work, and they gate everything else.
- **Discovering the current model.** Nothing reads a legacy application and tells you what its permissions actually are. Expect manual archaeology in Phase 1.
- **Regulated separation obligations.** Where a deal carries hold-separate or clean-team conditions, treat the model as one input to your counsel's review, not as a compliance control on its own.
- **Resource identity.** If the same customer exists as a record in both CRMs, that duplication has to be resolved before relationships about it mean anything.
- **Deciding who should have access.** Post-close entitlement policy is a governance and compliance decision. This layer enforces and evidences it; it does not make it.

A single schema holds both companies' definitions. The only new construct is the one that spans them:

```
// both sides keep their own shapes; the prefix keeps them apart
definition user {}

definition parentco/group {
  relation member: user
}

definition acquiredco/team {
  relation member: user
}

definition acquiredco/deal {
  relation owner: acquiredco/team#member

  // the merger, expressed once, in the schema
  relation viewer: user | acquiredco/team#member | parentco/group#member

  permission view = viewer + owner
}
```

Illustrative schema, simplified for the brief. Real models carry inheritance, tenancy and exclusions; we'd write yours against your actual objects in a working session.

WHAT WE'D NEED TO SCOPE IT

- ? Are you pre-close and planning, or already mid-integration with a target date?
- ? How many applications on each side make their own authorization decisions today?
- ? Is there one identity provider yet, or is that still in flight?
- ? What is the first cross-company access that has to work — and who is waiting on it?
- ? Which side's permission model, if either, is intended to survive?