DEVSECOPS DAY 2026 · BOLOGNA · 14 OTTOBRE

Oltre i ruoli: autorizzazione relazionale con OpenFGA

Il JWT dice chi sei, non cosa puoi fare

Francesco Montelli · Freelance Software Engineer
montelli.dev

slide e materiali

DEVSECOPS DAY 2026 · BOLOGNA · 14 OTTOBRE

Oltre i ruoli: autorizzazione relazionale con OpenFGA

Il JWT dice chi sei, non cosa puoi fare

Francesco Montelli · Freelance Software Engineer
montelli.dev

slide e materiali

DI COSA PARLIAMO OGGI

Quattro atti

01 **Autenticazione e autorizzazione**
Il punto di partenza, e dove i ruoli si rompono.

02 **Il modello relazionale (ReBAC)**
Relazioni, permessi calcolati, gruppi, eccezioni, permessi che scadono da soli.

03 **Caccia a un accesso di troppo**
Quando qualcosa va male: chi, da dove, da quando, e come non ripeterlo.

04 **Migrazione nella pratica**
Da un sistema a ruoli esistente, senza big bang.

Atto 1

Autenticazione e autorizzazione

Un dominio, un token, e dove i ruoli smettono di bastare.

ALL'INIZIO · UN'ACADEMY INTERNA

Una piattaforma di corsi

ADMIN

Gestisce l'intera piattaforma.

AUTORE

Un dipendente che crea i corsi.

STUDENTE

Ogni dipendente vede ogni corso.

IL PUNTO DI PARTENZA

Il JWT dice chi sei

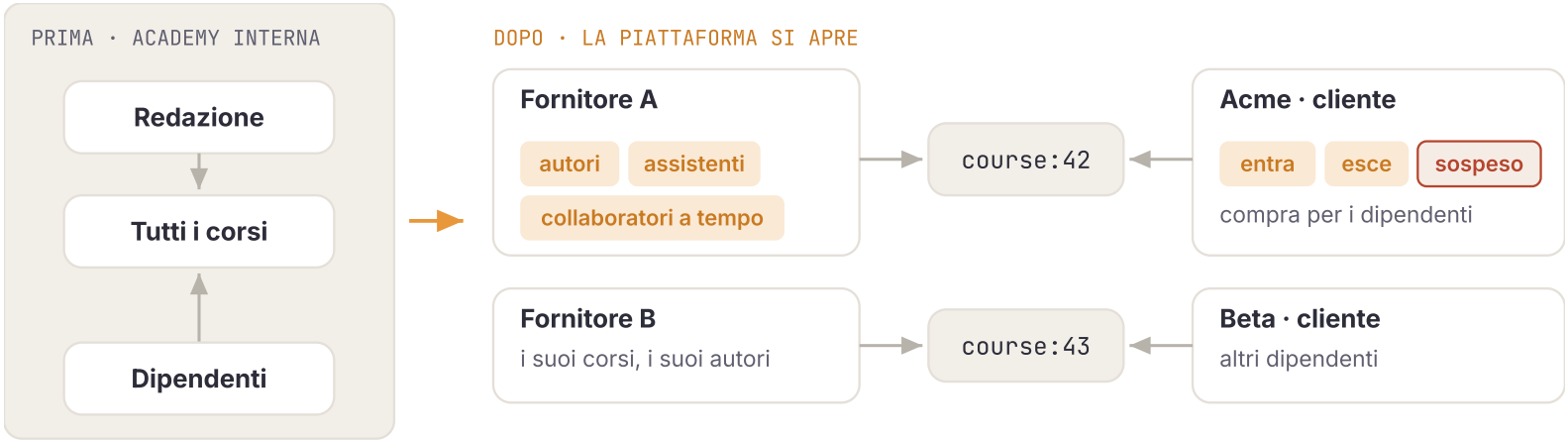


JWT · FIRMATO DALL'IDENTITY PROVIDER

```
{
  "sub": "3f9a…c21",
  "preferred_username": "mrossi",                    CHI SEI
  "name": "Mario Rossi",
  "realm_access": { "roles": ["author"] },
  "groups": ["/courses/authors"],                    A COSA APPARTIENI
  "iat": 1791972000,
  "exp": 1791975600                # +1 ora          FINO A QUANDO
}
```

LA NUOVA REALTÀ

La piattaforma si apre



DOVE IL TOKEN NON BASTA

Tre cose che un ruolo non sa dire

01 **Autore di cosa?**
Mario: il suo corso, non quello di un altro fornitore.

02 **E un'eccezione?**
L'account di Paolo è sospeso per un mese, ma resta in Acme.

03 **Fino a quando?**
Luca serve solo fino al 30 giugno.

Atto 2

Il modello relazionale

Relazioni, permessi calcolati, gruppi, eccezioni, permessi che scadono da soli.

CAMBIO DI PROSPETTIVA

Il mondo è fatto di relazioni

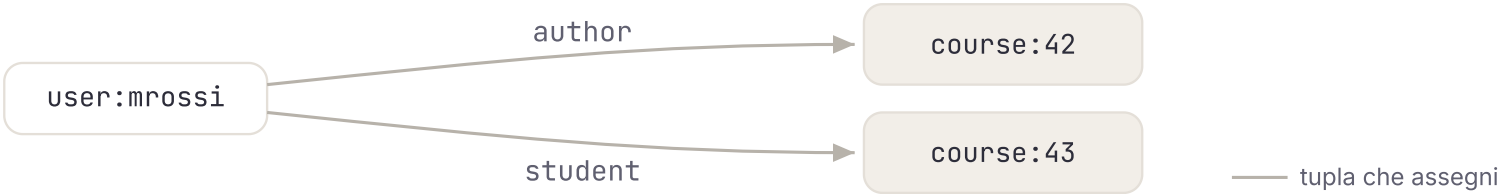
01 Mario **ha scritto** `course:42` → lo può **modificare**.

02 Paolo è **sospeso da** `course:42` → non lo può **vedere**.

03 Luca **corregge** `course:42` fino al 30/06 → dal 1° luglio, **niente**.

La relazione vive su un oggetto

> «Mario è **autore di questo corso**, e solo **studente di quell'altro**.»



MODELLO

```
type user
type course
  relations
    define author: [user]
    define assistant: [user]
    define student: [user]
```

TUPLE

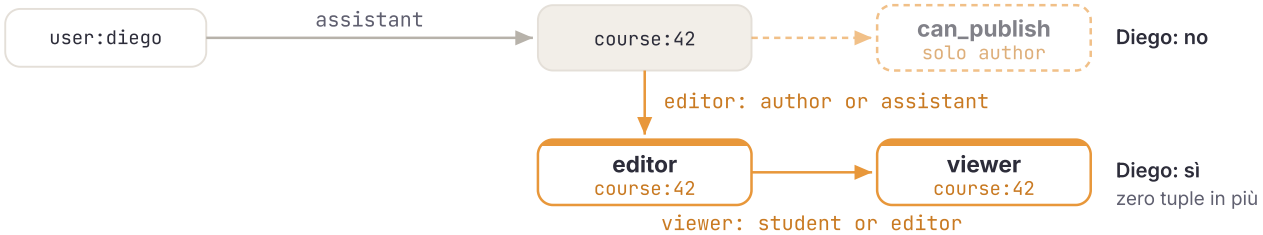
USER	RELATION	OBJECT
user:mrossi	author	course:42
user:mrossi	student	course:43

CHECK

```
check(user:mrossi, author, course:42) → true
check(user:mrossi, author, course:43) → false
```

Modificare si calcola

> «Diego è assistente: **valuta**, ma **non pubblica**.»



MODELLO

```
type user
type course
  relations
    define author: [user]
    define assistant: [user]
    define student: [user]
+   define editor: author or assistant  # modificare
+   define viewer: student or editor    # vedere
+   define can_publish: author          # pubblicare
```

TUPLE

USER	RELATION	OBJECT
user:diego	assistant	course:42

CHECK

```
check(user:diego, editor, course:42)       → true
check(user:diego, can_publish, course:42) → false
```

SALTO 3 DI 5 · I GRUPPI

Il legame vale per un gruppo

> «Acme compra il 42 **per tutti**, anche per chi arriva **domani**.»



MODELLO

```
  type user
+ type org
+   relations
+     define member: [user]
  type course
    relations
      define author: [user]
      define assistant: [user]
      define student: [user, org#member]
      define editor: author or assistant
      define viewer: student or editor
```

TUPLE

	USER	RELATION	OBJECT
+	org:acme#member	student	course:42
+	user:anna	member	org:acme
+	user:paolo	member	org:acme
+	**user:giulia**	**member**	**org:acme**

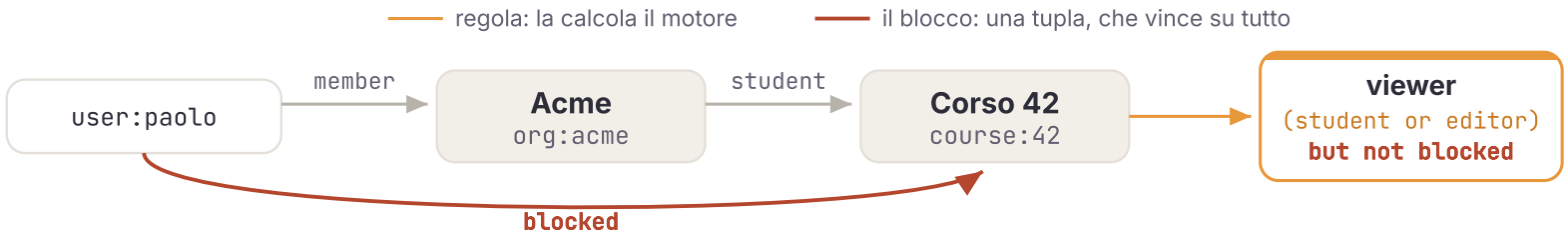
CHECK

```
check(user:giulia, viewer, course:42) → true
```

SALTO 4 DI 5 · L'ECCEZIONE

Un permesso si può togliere

> «Paolo è di Acme, ma **sospeso**: il corso non lo vede.»



MODELLO

```
type user
type org
  relations
    define member: [user]
type course
  relations
    define author: [user]
    define assistant: [user]
    define student: [user, org#member]
+   define blocked: [user]
    define editor: author or assistant
    define viewer: (student or editor) but not blocked
```

TUPLE

	USER	RELATION	OBJECT
	user:paolo	member	org:acme
+	user:paolo	blocked	course:42

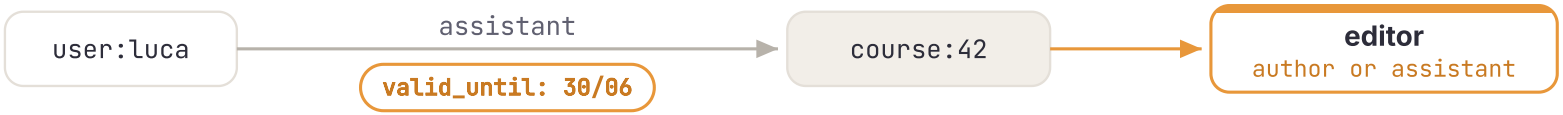
CHECK

```
check(user:paolo, viewer, course:42) → false
check(user:anna,  viewer, course:42) → true
```

SALTO 5 DI 5 · IL CONTESTO

Un permesso che scade da solo

> «Luca fa da assistente su questo corso, ma **solo fino al 30 giugno**.»



MODELLO

```
  type user
  type course
    relations
      define author: [user]
      define assistant: [user, user with non_expired]
      define editor: author or assistant

+ condition non_expired(current_time: timestamp, valid_until: timestamp) {
+   current_time < valid_until
+ }
```

TUPLE

	USER	RELATION	OBJECT
+	user:luca	assistant	course:42

valid_until: 2026-06-30

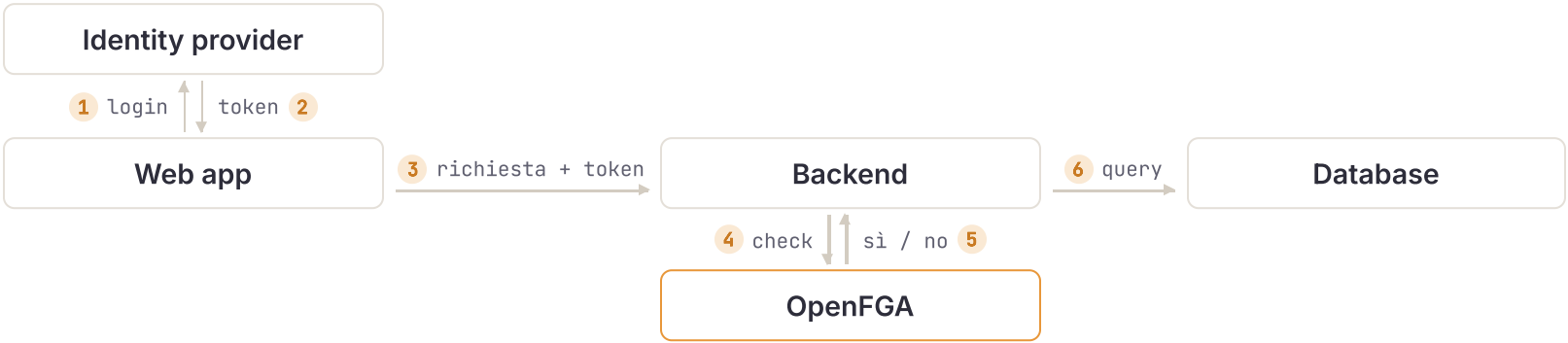
CHECK

```
check(user:luca, editor, course:42,
  { current_time: 15/06 }) → true

check(user:luca, editor, course:42,
  { current_time: 02/07 }) → false
```

NEL SISTEMA CHE AVETE GIÀ

Il token resta, cambia chi risponde



DAL TOKEN `username "mrossi" → user:mrossi`

AL MOTORE `check(user:mrossi, editor, course:42) → true`

Atto 3

Caccia a un accesso di troppo

Auditing: non se qualcosa va male, ma quando.

UN VENERDÌ POMERIGGIO

«Un collaboratore doveva restare fino a giugno. Apre ancora i nostri corsi.»

01

Chi altro è esposto, adesso?

02

Da dove arriva l'accesso?

03

Da quando, e perché non è mai sparito?

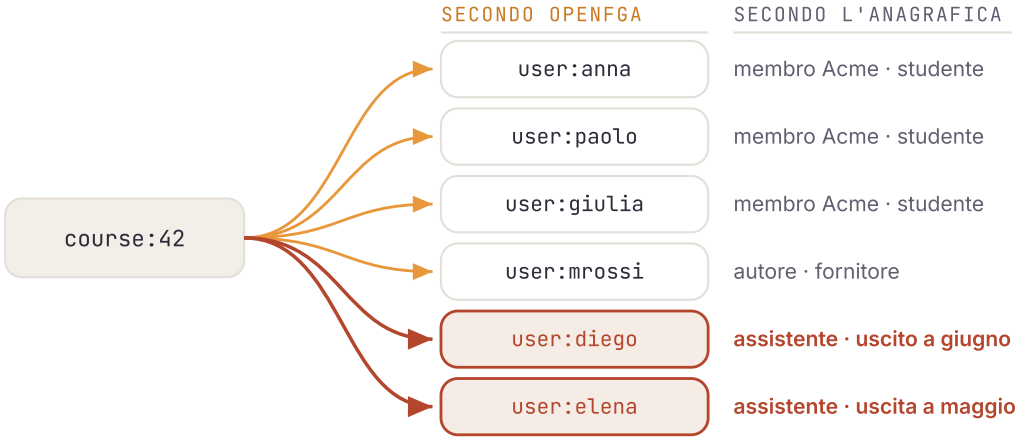
PASSO 1 DI 3 · CONTENERE

Chi può vederlo, adesso

```
QUERY   listUsers(viewer, course:42, context: { current_time: now })
```

6
viewer del corso 42

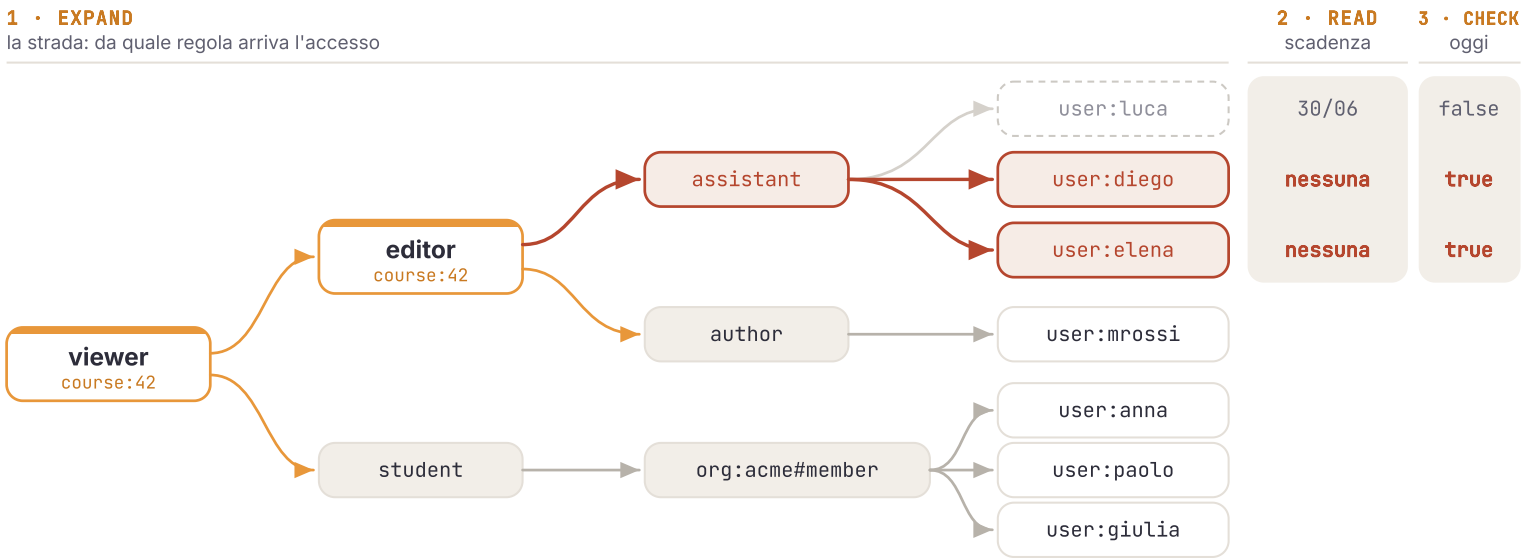
2
senza diritto



PASSO 2 DI 3 · DIAGNOSTICARE

Da dove arriva l'accesso

```
QUERY   1 expand(viewer, course:42)   2 read(assistant, course:42)   3 check(…, { current_time: now })
```



PASSO 3 DI 3 · RICOSTRUIRE, CHIUDERE, VERIFICARE

Una concessione mai revocata

1 · READCHANGES quando: ogni scrittura, con data

OPERATION	TIMESTAMP	TUPLE_KEY			CONDITION
WRITE	2026-02-09	user:diego	assistant	course:42	— nessuna
WRITE	2026-02-09	user:elena	assistant	course:42	— nessuna
WRITE	2026-04-02	user:luca	assistant	course:42	non_expired · 30/06
DELETE	—	**nessuno, fino a oggi**			

2 · DELETE chiudere

```
delete(user:diego, assistant, course:42)
delete(user:elena, assistant, course:42)
```

3 · CHECK verificare, adesso

```
check(user:diego, viewer, course:42) → false
check(user:elena, viewer, course:42) → false
```

DOPO LA CHIUSURA

Correggere la causa, non il caso

> «Il modello permetteva **un assistente senza scadenza**.»

1 · IL MODELLO, IN UNA PR

```
  type course
    relations
-     define assistant: [user, user with non_expired]
+     define assistant: [user with non_expired]
```

2 · DA OGGI, LA SCRITTURA

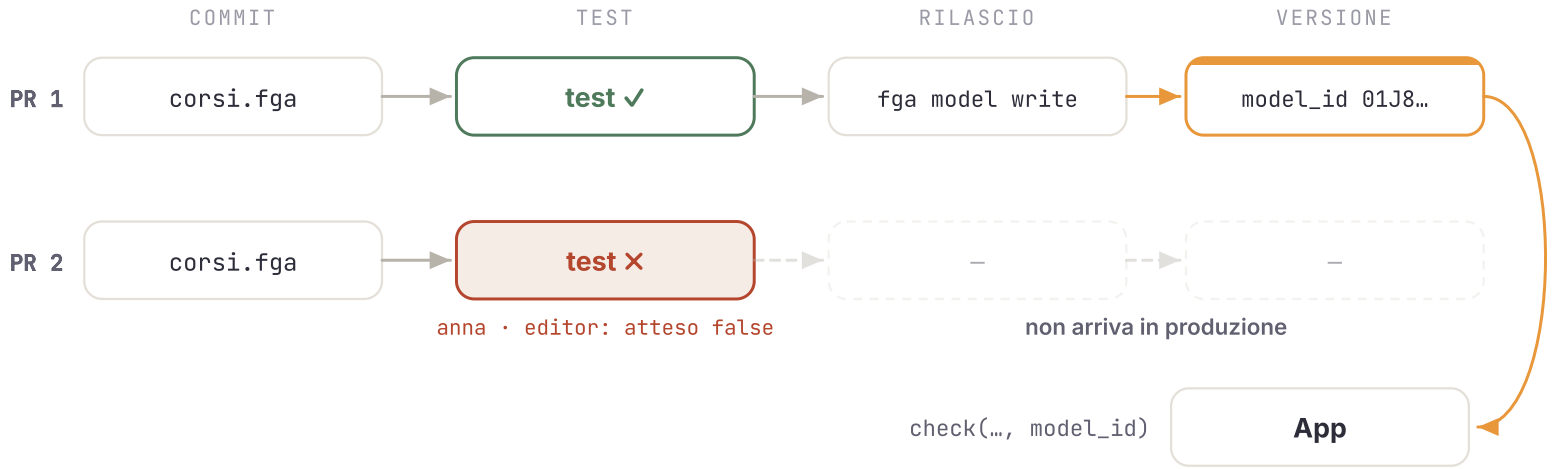
```
write(user:sara, assistant, course:42)
  → rifiutata: manca la scadenza

write(user:sara, assistant, course:42,
  non_expired { valid_until: 2026-12-31 })
  → ok
```

PRIMA riscrivere con una data le tuple assistant già scritte: per il nuovo modello non valgono più

IL MODELLO È CODICE

Passa dalla pipeline, come il resto



Atto 4

Migrazione nella pratica

Rinunciare al «si è sempre fatto così», senza chiedere fiducia.

NON DA UN GIORNO ALL'ALTRO

Quattro fasi, per endpoint

00

Backfill

il motore sa quello che sa il database

01

Shadow

nessun utente se ne accorge

02

Compare

le differenze diventano numeri

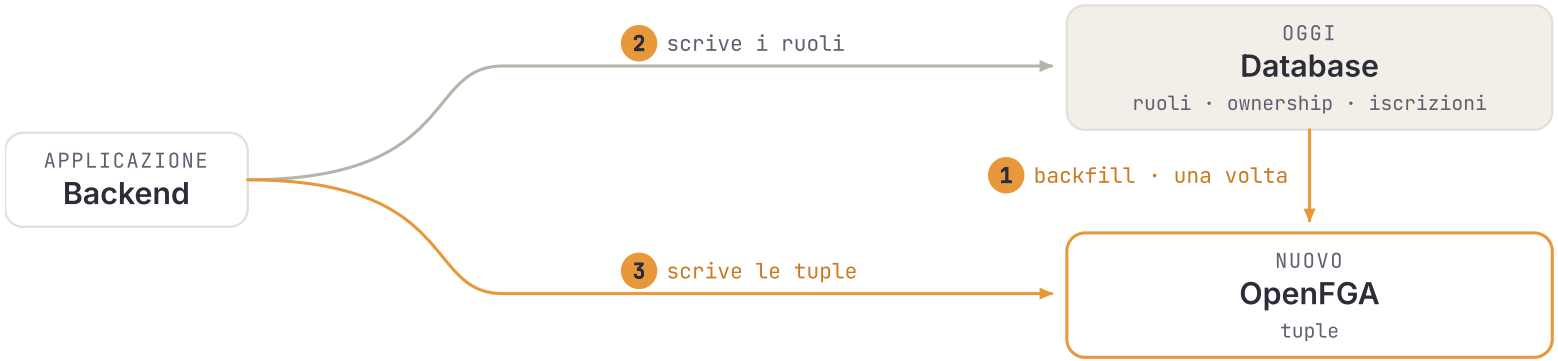
03

Cutover

si torna indietro con un flag

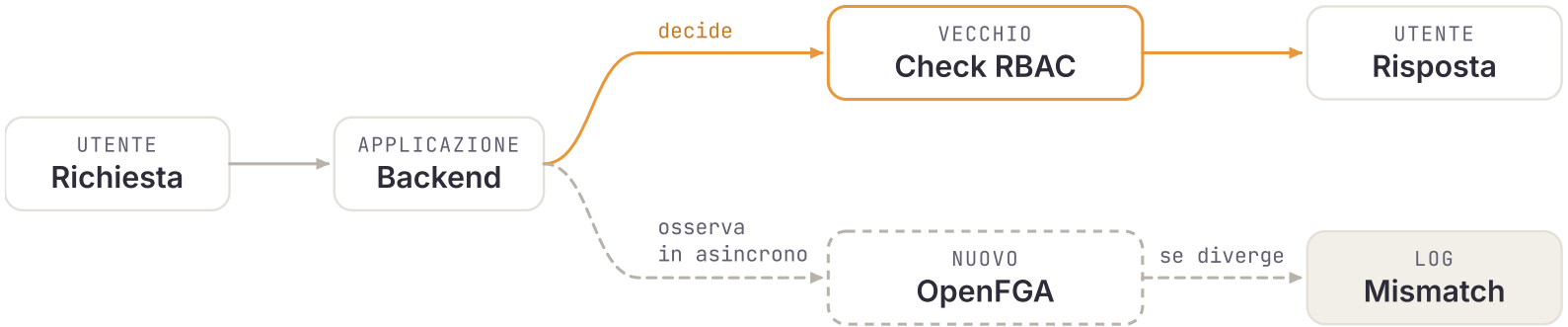
FASE 00

Backfill e dual-write



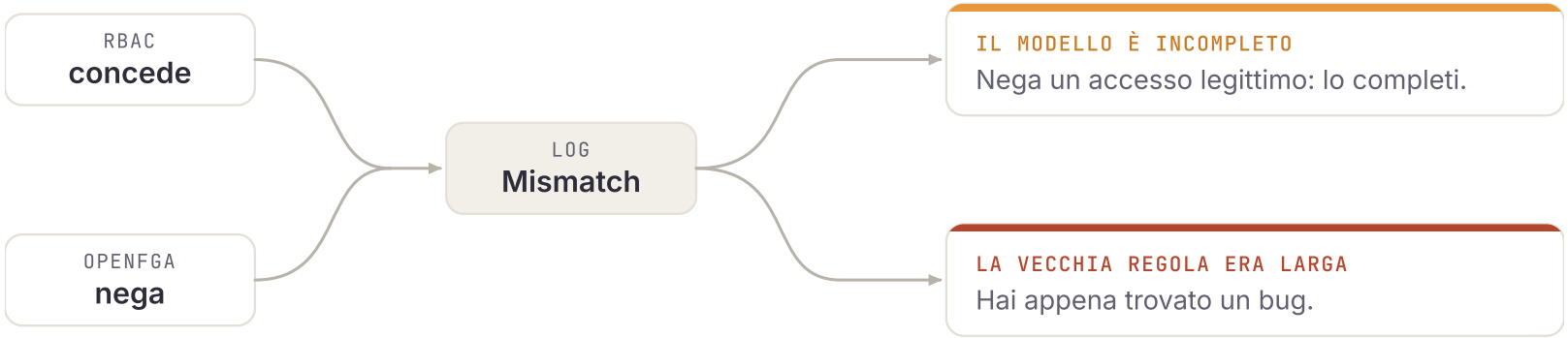
FASE 01

Shadow: il motore osserva



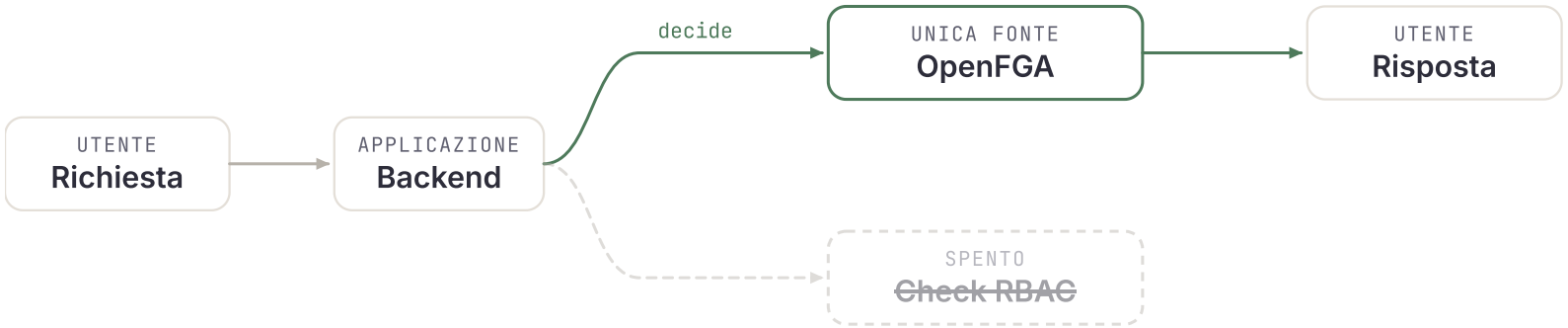
FASE 02

Compare: ogni mismatch è un indizio



FASE 03

Cutover: decide il motore



CONCLUSIONI · ONESTÀ INTELLETTUALE

Quando serve, e quando no

Restate sui ruoli

- pochi tipi di risorse
- nessuna gerarchia
- pochi ruoli, una matrice

→ quando nascono ruoli come `autore-del-42`

Passate alle relazioni

- multi-tenant
- gerarchie e gruppi
- eccezioni
- scadenze
- chi vede cosa

Spesso convivono: ruoli per le regole a grana grossa, relazioni per le singole risorse.

IN CHIUSURA

Tre cose da portare a casa

01

Parti da una frase: chi, cosa, su quale oggetto.

02

Descrivi le relazioni: i permessi ne discendono.

03

Il modello è codice: si testa, si versiona, si interroga.

Oltre i ruoli: autorizzazione relazionale.

Grazie, domande?

francesco@montelli.dev · montelli.dev
github.com/monte97 · linkedin.com/in/francesco-montelli

Lascia un feedback