LF DECENTRALIZED TRUST

*03 March 2026*

# *From Identity-First to Authority Continuity*

PIC

**Nicola Gallo**
*Co-founder, Nitro Agility S.r.l.*

Permguard (www.permguard.com)

Co-Chair, Trusted AI Agents Working Group
Decentralized Identity Foundation (DIF)

NitroAgility

Permguard

# The Real Problem

This problem did not start with AI

It has existed for decades

**AI agents made it impossible to ignore**

*What breaks under AI agents was already broken.*

# The Real Problem

We solved security with:

- Protocols
- Posture
- Configuration
- Policy

**But we never solved authority propagation.**

> *How can I trust either the posture, configuration, policies of something.*
> *I do not own, do not control, and do not even know?*

# Problem Framing

- Non-determinism ⟶ trust problem
- Authority propagation ⟶ security model problem
- Workloads can deviate from the expected execution flow
- A security model must constrain execution, not behavior

# Security & Tokens

**Why Tokens Instead of Credentials (API access)**

> Avoid sharing long-lived credentials (usernames/passwords)with third parties

> Tokens allow scoped, time-limited, and revocable access

> Enable impersonation without exposing core credentials

> Support secure integration across apps, APIs, and federated IdPs

# Security & Tokens

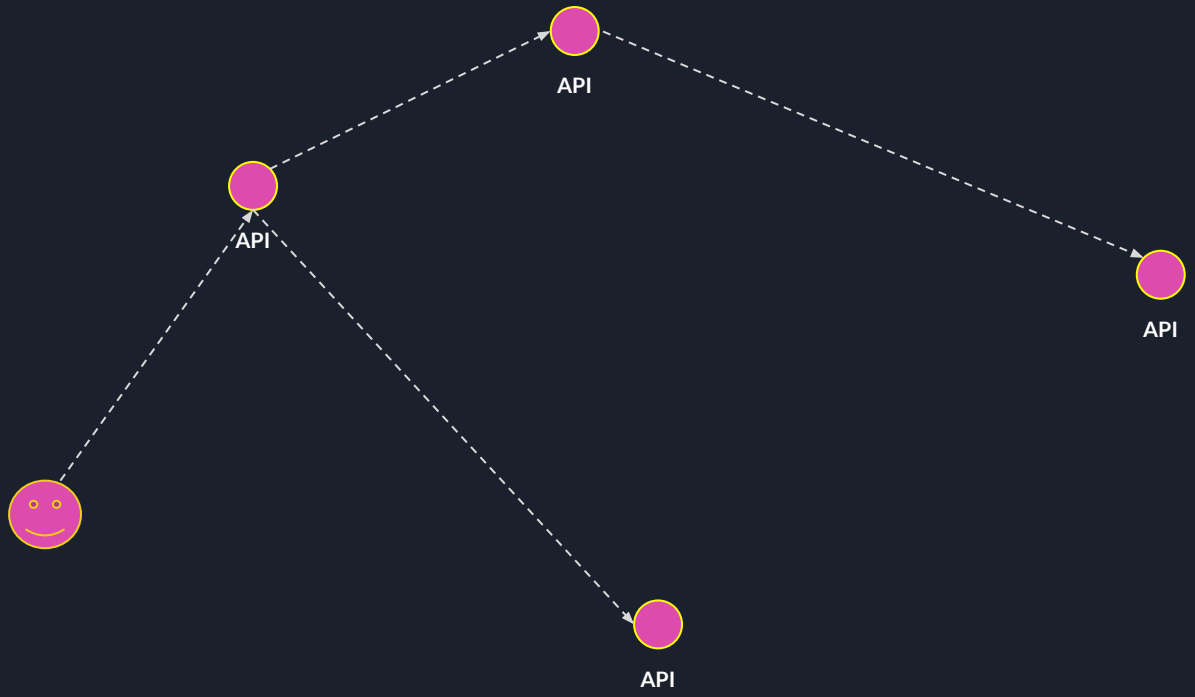
**Sub-tradeoff: Refresh Tokens**

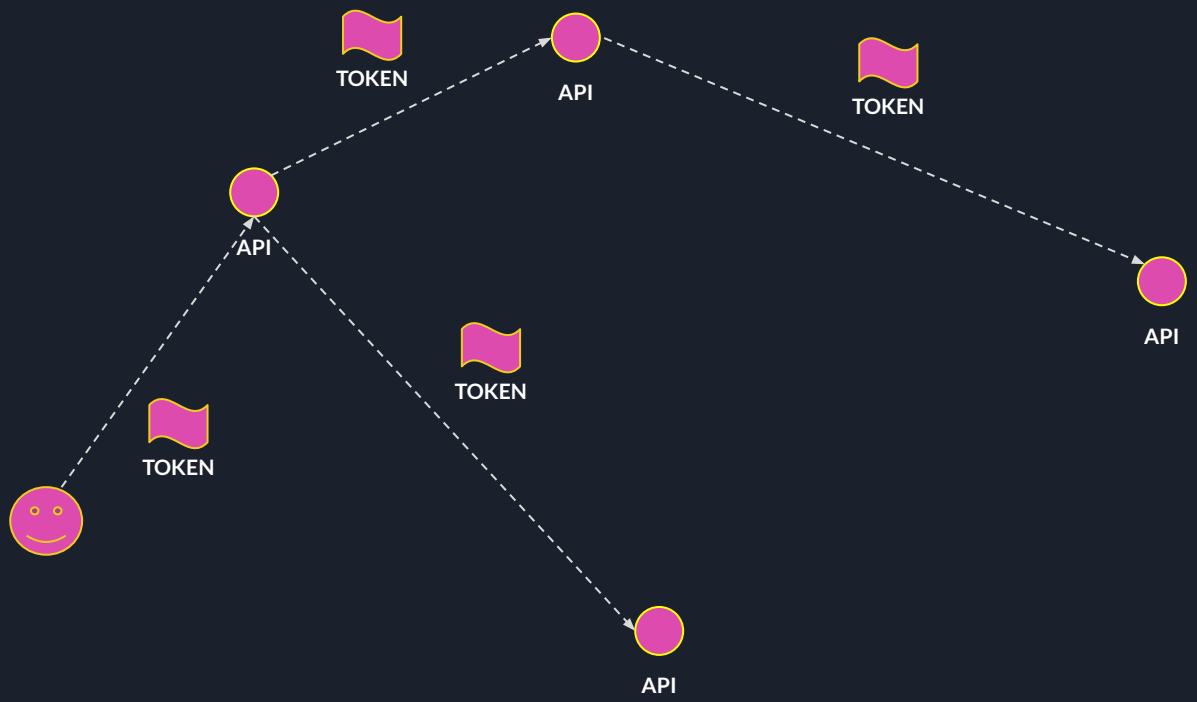
> Provide **long-lived sessions without re-prompting** the user

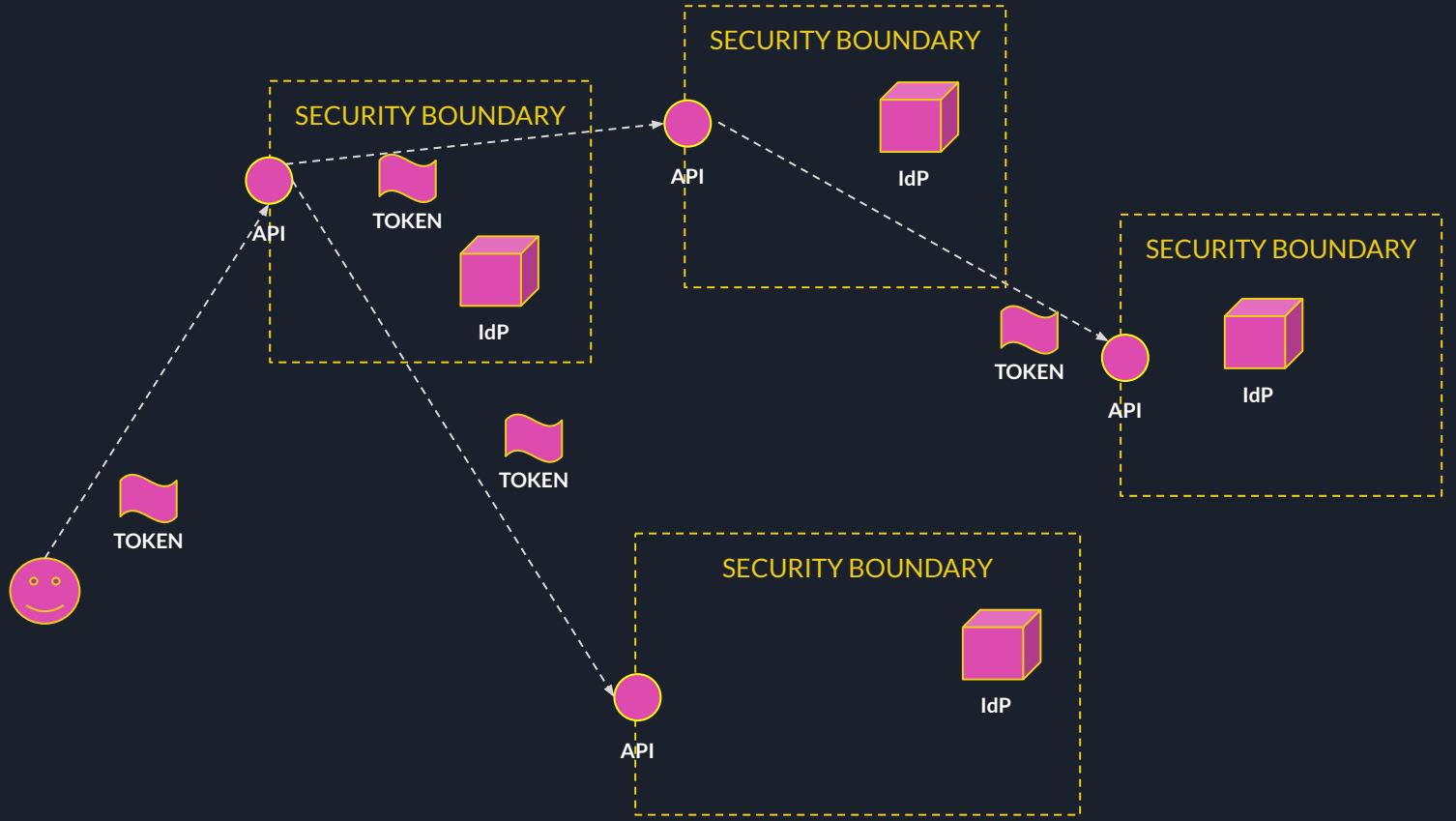
> But **reintroduce** a **long-lived secret** that must **be secured**

>**Risks mitigated** with: **rotation, reuse detection**, sender-constrained tokens

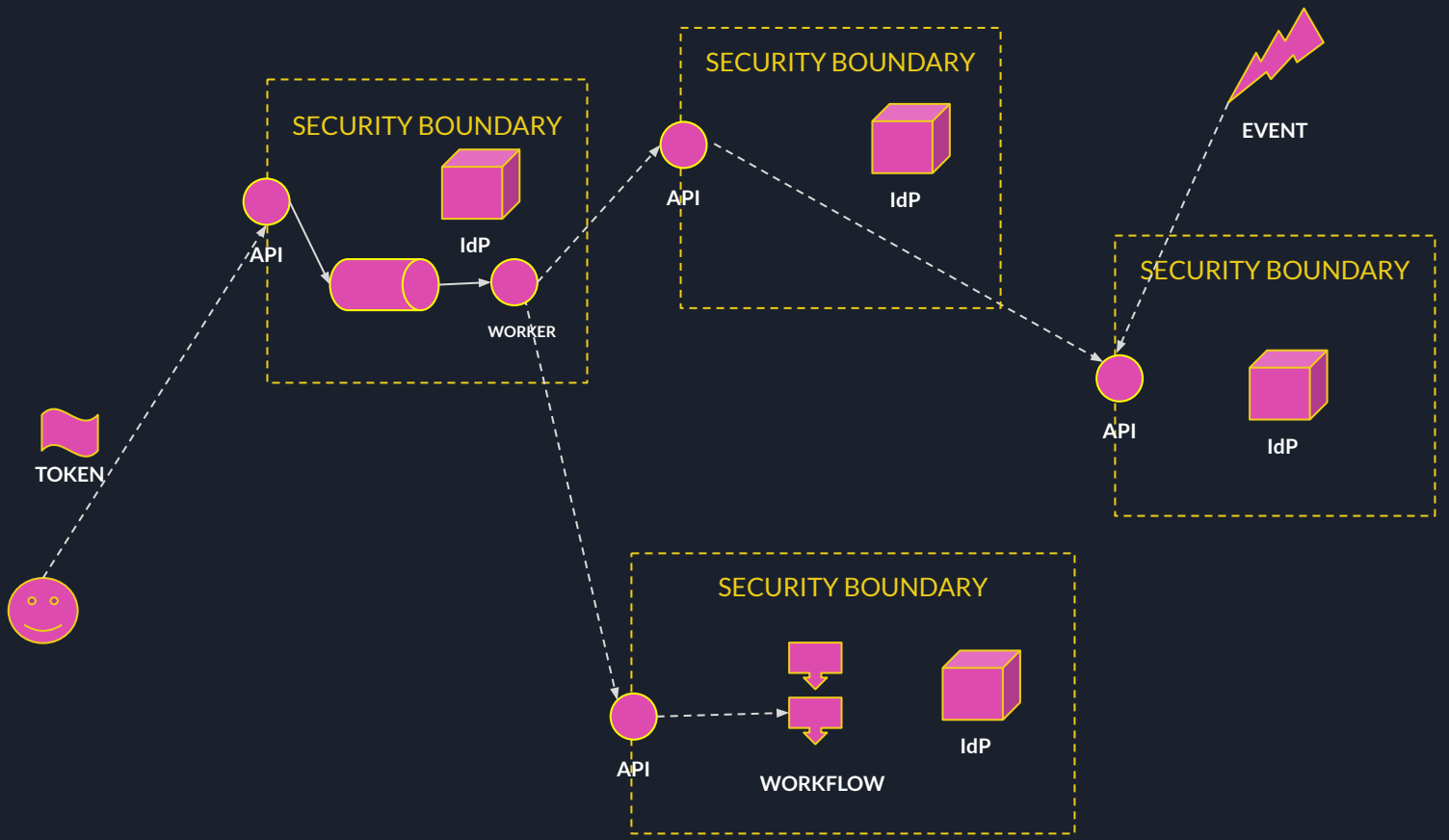
> **Balance** between **usability** (fewer logins) and **security** (token theft risk)

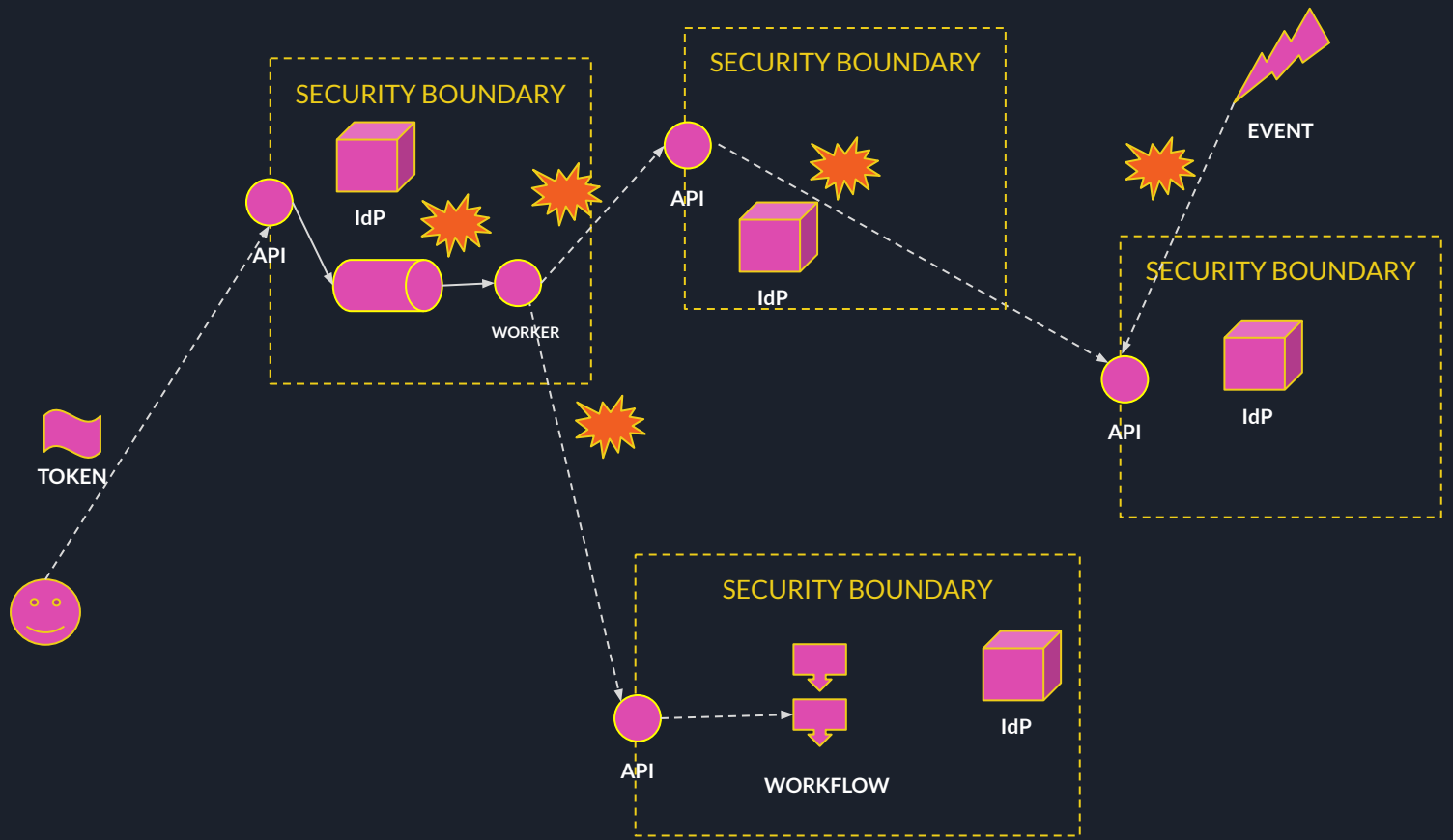






# Reality is more COMPLEX the it seems!





# Microservices

*Containers exploded the surface. We built patterns to cope. The execution model didn't change.*

**Circuit Breaker**

Stop calling a failing service

**Saga Pattern**

Steps + compensations across services

**Retry / Backoff**

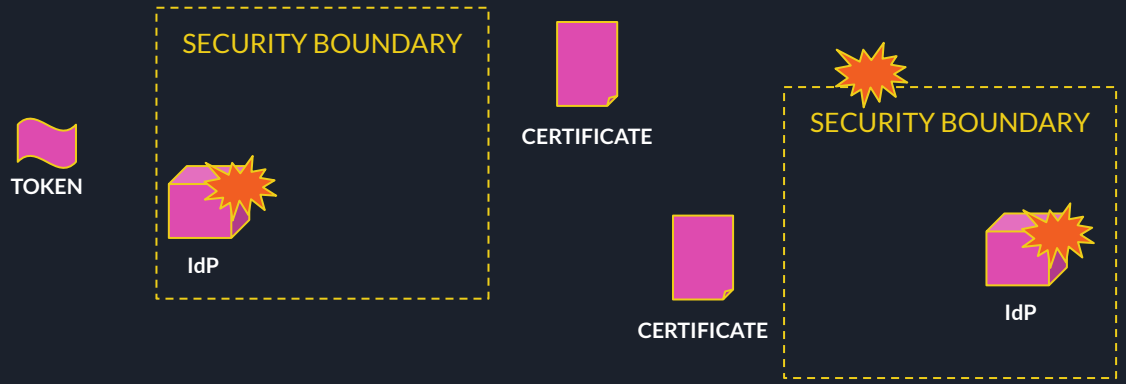
Re-execute failed ops over time

**Event-driven**

Producers and consumers decoupled in time

The Saga pattern is a sequence of causally linked steps — each one can fail, each one has a compensating action.

**A Saga is a distributed execution chain. We already had the model — we just never applied it to security.**




# PUBLIC vs PRIVATE

2 two security models


# What if they were AI AGENTS?

**AI agents made it impossible to ignore**

# The Real Problem

Reality:

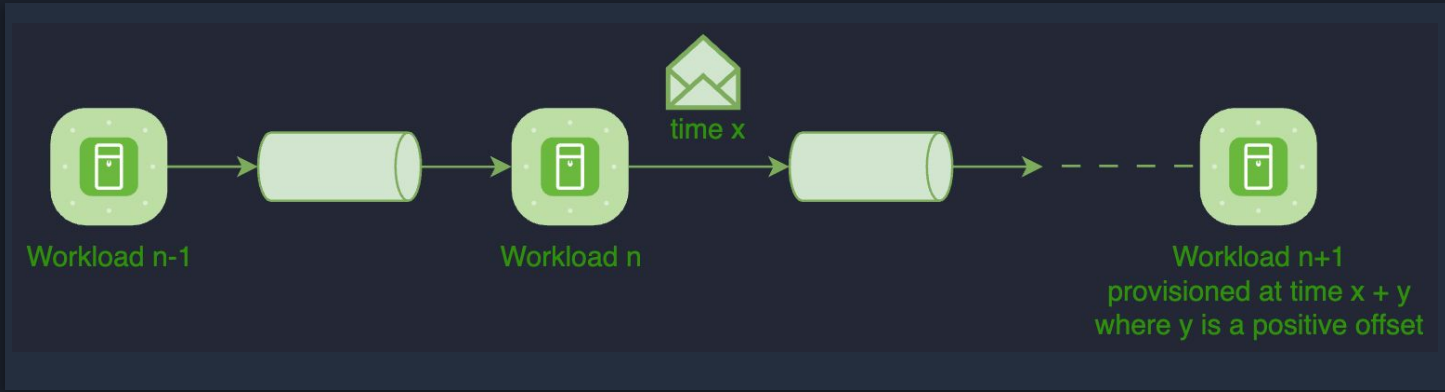
- The illusion of synchrony breaks at scale
- Distributed systems introduce temporal gaps
- No global consistency across services

We introduced patterns to cope with this:

- CQRS ⟶ separate read and write models
- Saga ⟶ manage distributed transactions
- Event-driven systems ⟶ embrace asynchrony

*Once we realized that synchronous models don’t hold at scale, we started introducing patterns like Saga and CQRS.*

# Canonical Execution Model




*Execution is temporal and causal, not positional.*

- Workload n+1 provisioned at time x + y  (y > 0)
- Authority flows from origin — never re-created
- Every hop must prove continuity, not possession

# The Shift

A subject creates authority by expressing intent.

After that:

> **Authority must only be continued, never recreated.**

# A New Primitive is Required

To make this shift possible,
we need a new primitive.

Tokens encode possession.
We need a primitive that encodes intent and execution.

# Proof of Continuity (PoC)

To continue authority, a workload must prove:

**Causal Continuity**

Execution is a valid continuation of its predecessor.

**Origin Invariance**

The origin ($p_0$) is immutable throughout the chain.

**Monotonic Authority Restriction**

Authority is monotonically non-increasing. $ops_{i+1} \subseteq ops_i$ — authority never expands.

**Proof of Continuity (PoC)** is built from **Proof of Relationship (PoP)**.
**Proof of Possession** is one mechanism, not the primitive.

# Execution is powered by continuity.

No continuity. No execution.

A workload may proceed only if it either:

- proves an authority continuity relationship, or
- explicitly creates new authority

Proof of Relationship is what validates continuity.

Possession is just one way to prove continuity, not the principle.

No authority mixing. No re-materialization.

**This is Authority Continuity.**

# Identity Is Not Required to Continue

### To Continue Authority

Identity: optional

Identifiers: optional

What matters: proof of continuity

### To Create Authority

Express own intent

Become a new origin

Identity: required

**Continuation ≠ Creation**

# Provenance Identity Continuity (PIC) Model


## Provenance

*The causal chain is always traceable and auditable. From Origin to end, unbroken. If it breaks, it stops.*

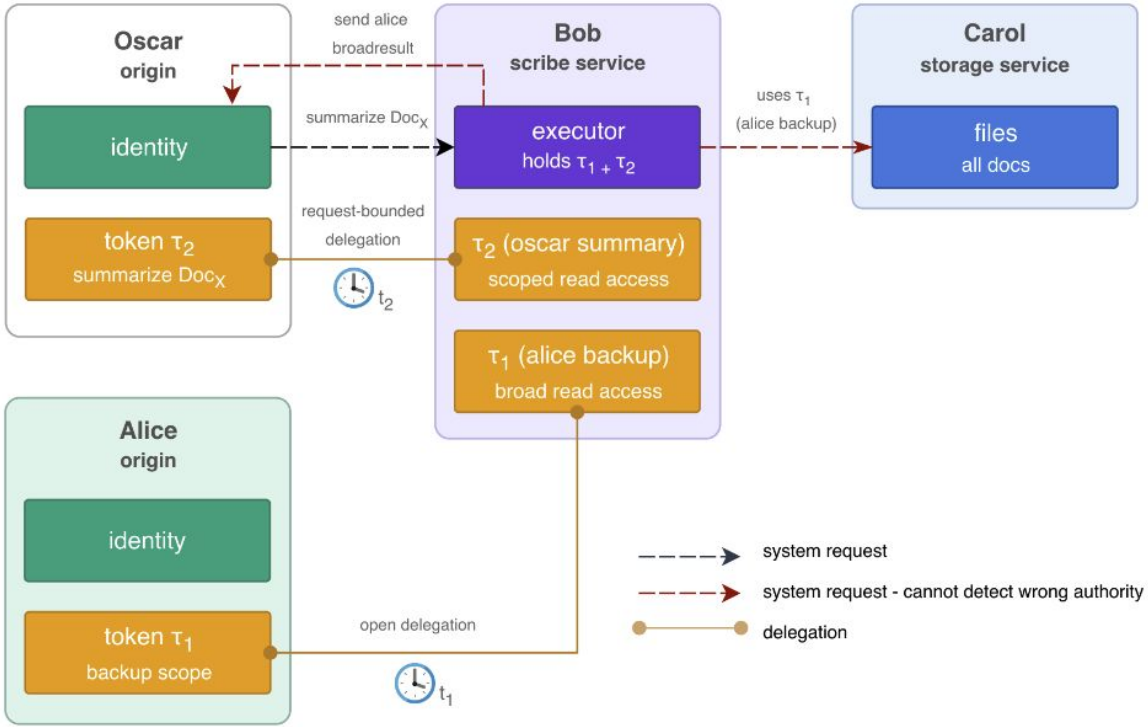
## Identity

*The origin is immutable. It generates authority. It cannot change throughout the chain.*

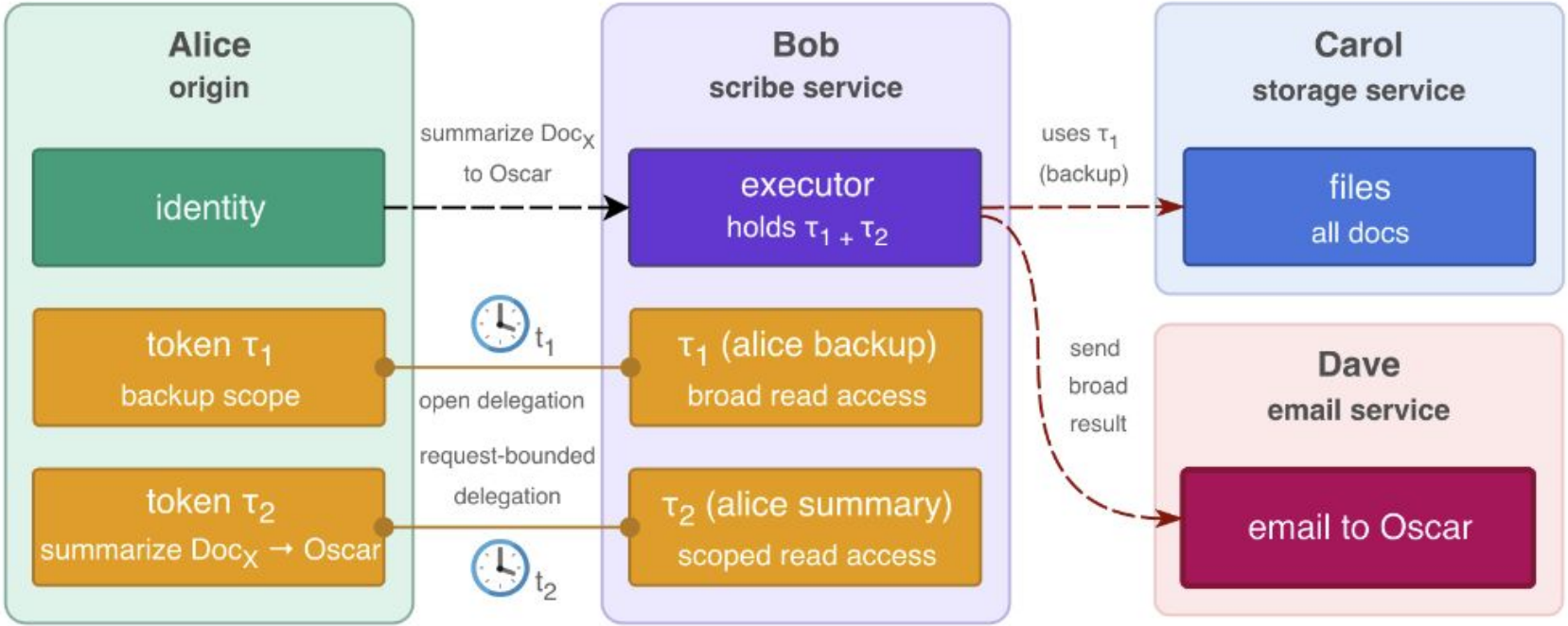
## Continuity

*Continuity is proven at every step. Proof of Continuity replaces Proof of Possession. Authority can only shrink.*

# Deputy Hijacking



# Deputy Overreach



# The untrusted orchestrator

Authority is not identity.

Authority is not possession.

**Authority is continuity.**

PIC is not just a protocol.
It is a correction of the model.
It is a new security primitive.

# Get Involved

## Website

www.pic-protocol.org

## Specification

github.com/pic-protocol/pic-spec

## Formal Model

Zenodo (DOI: 10.5281/zenodo.17833000)

PIC

www.pic-protocol.org

Not a Policy.
Not a Token.
An *Ontological Shift*
Proven Mathematically.

Thank you!

*JOIN THE PIC SLACK CHANNEL*