

When in-toto Met OPA

Turning Attestations into E2E Security & Compliance with Cloud-Native Policies

Snahil Singh and **Trishank Karthik Kuppusamy**

Open Source SecurityCon 2026

Introductions



Hi, I'm **Snahil**

Apple, Seattle

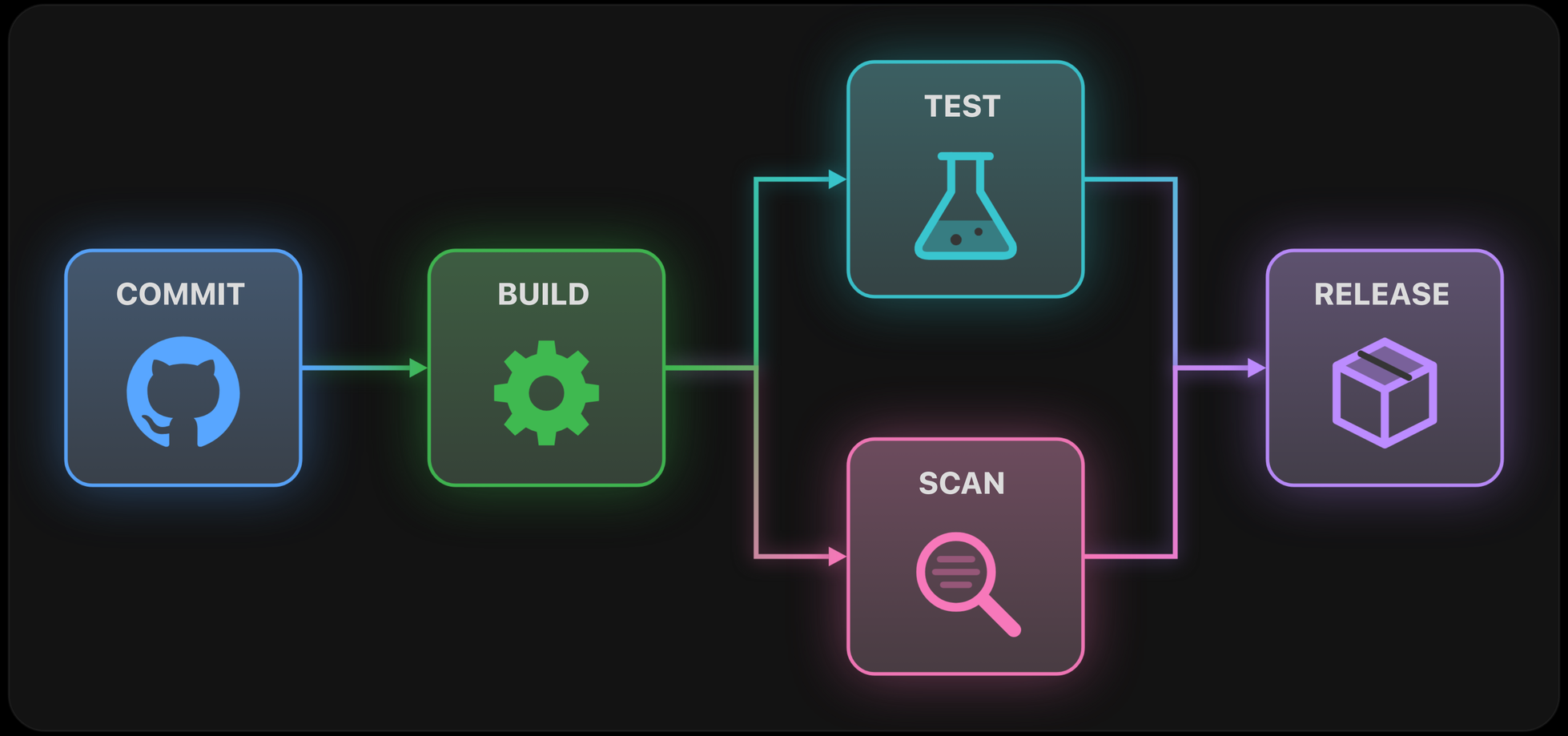


Hi, I'm **Trishank**

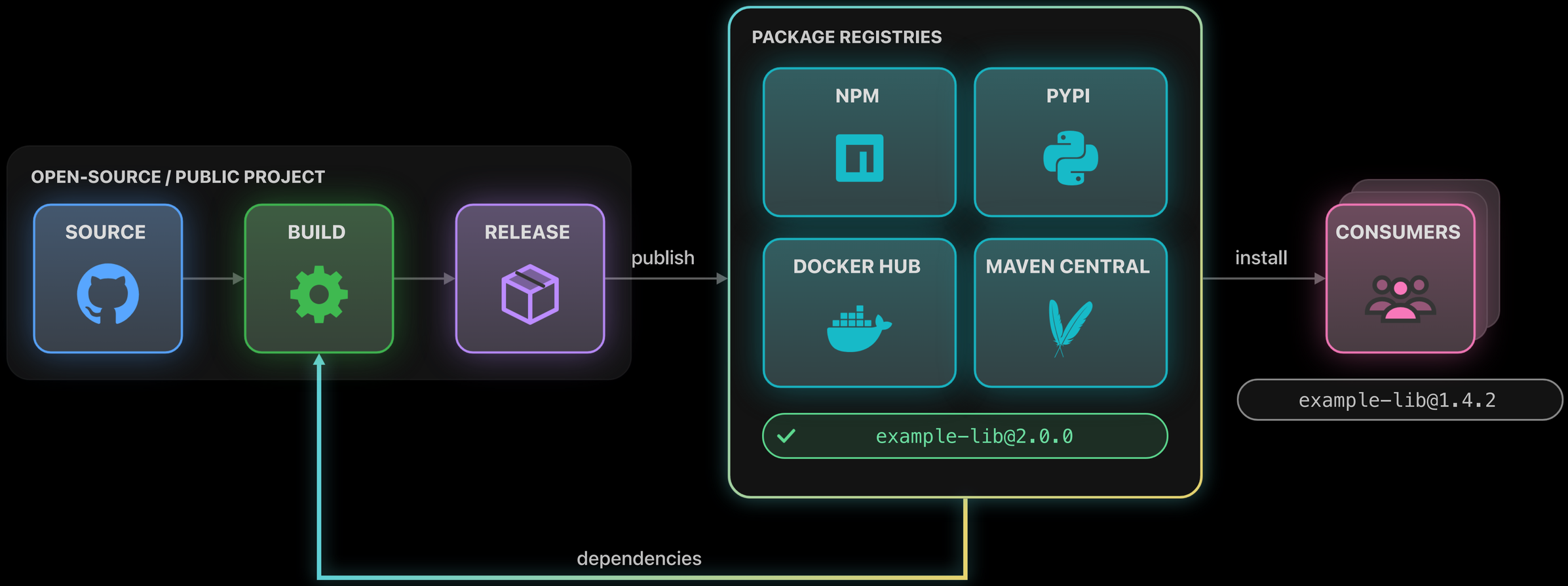
Apple, Seattle

Software Supply Chain Attacks

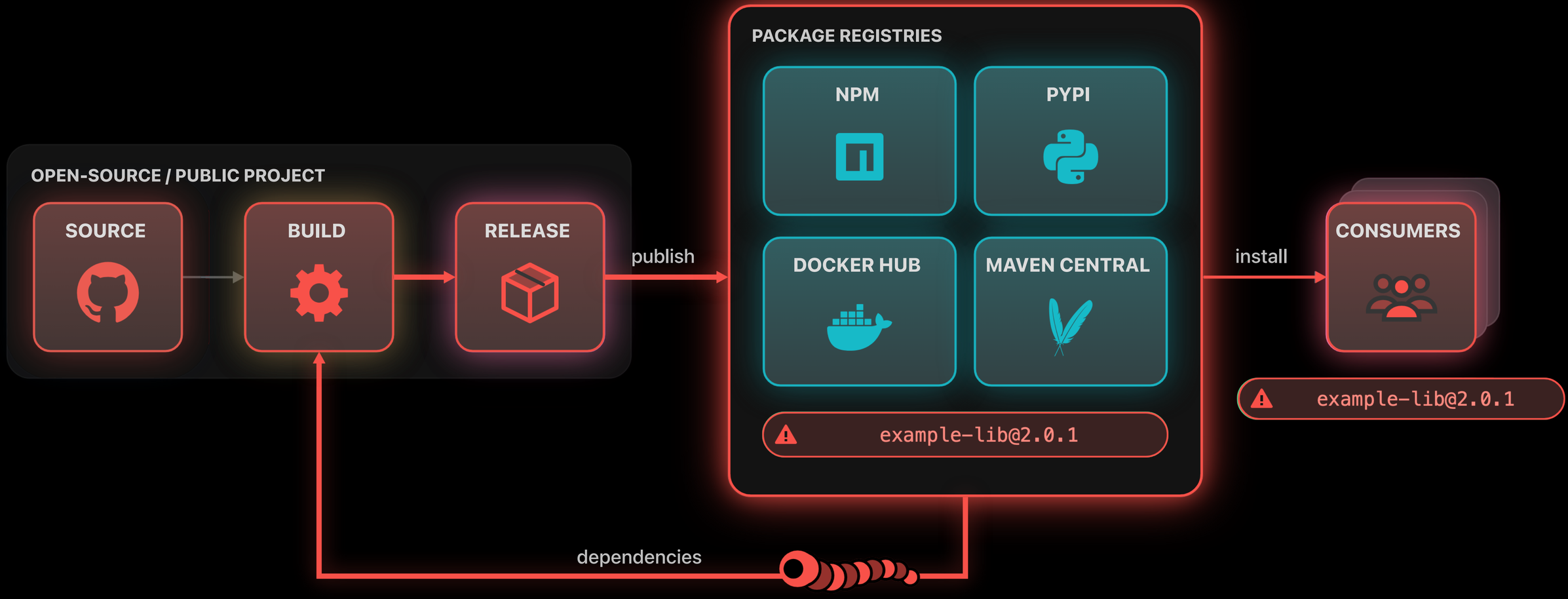
Software Supply Chains



Open-Source Dependencies



Supply Chain Attacks



Blast Radius

OPEN-SOURCE PACKAGES



e.g., libraries

SUPPLY CHAIN SECURITY TOOLS



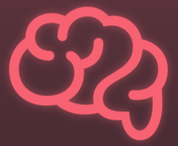
e.g., scanners

DEVELOPER TOOLS



e.g., IDE extensions

AI



e.g., models

FINANCE



e.g., cryptocurrencies

CYBERPHYSICAL INFRASTRUCTURE



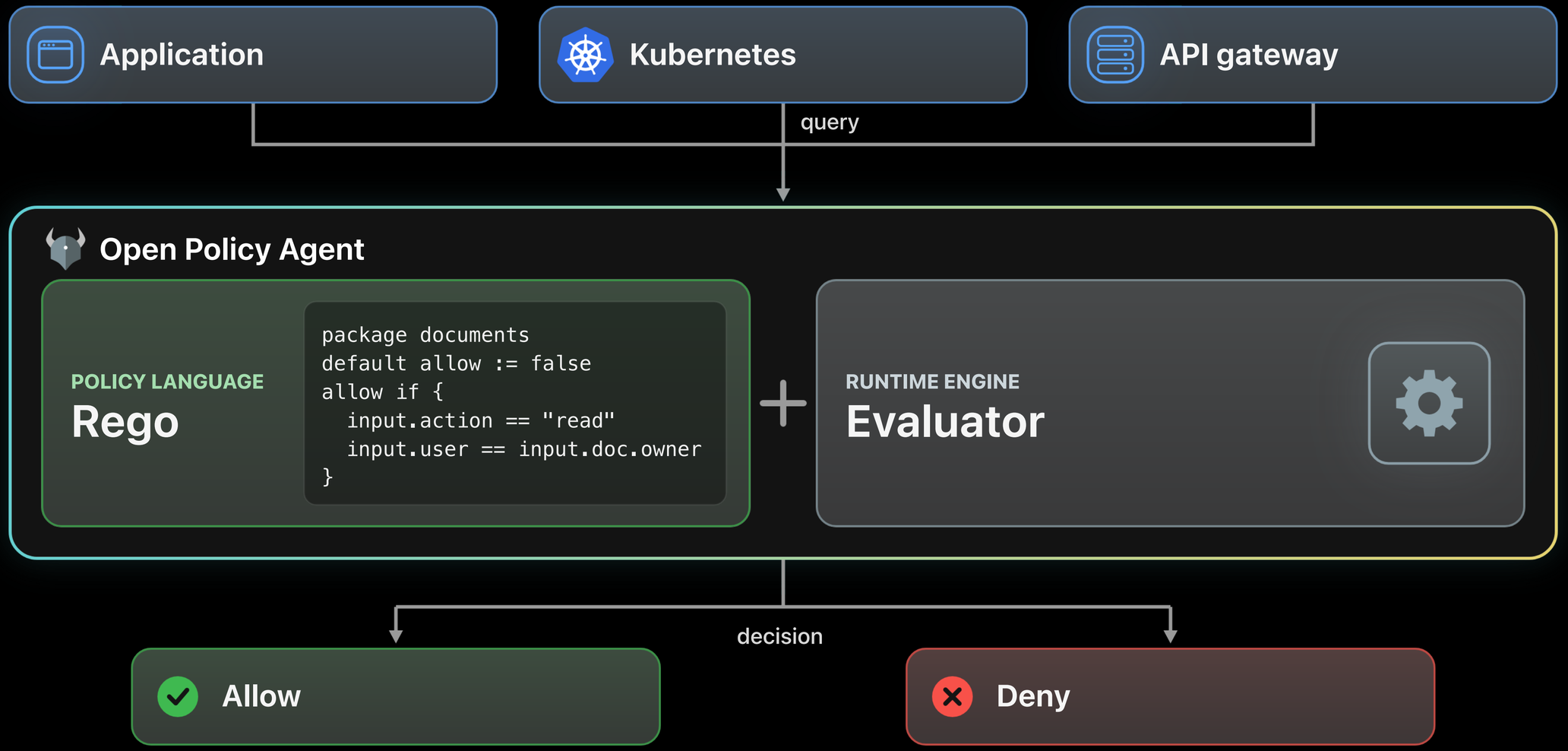
e.g., industrial control systems



in-toto-rego: Demo

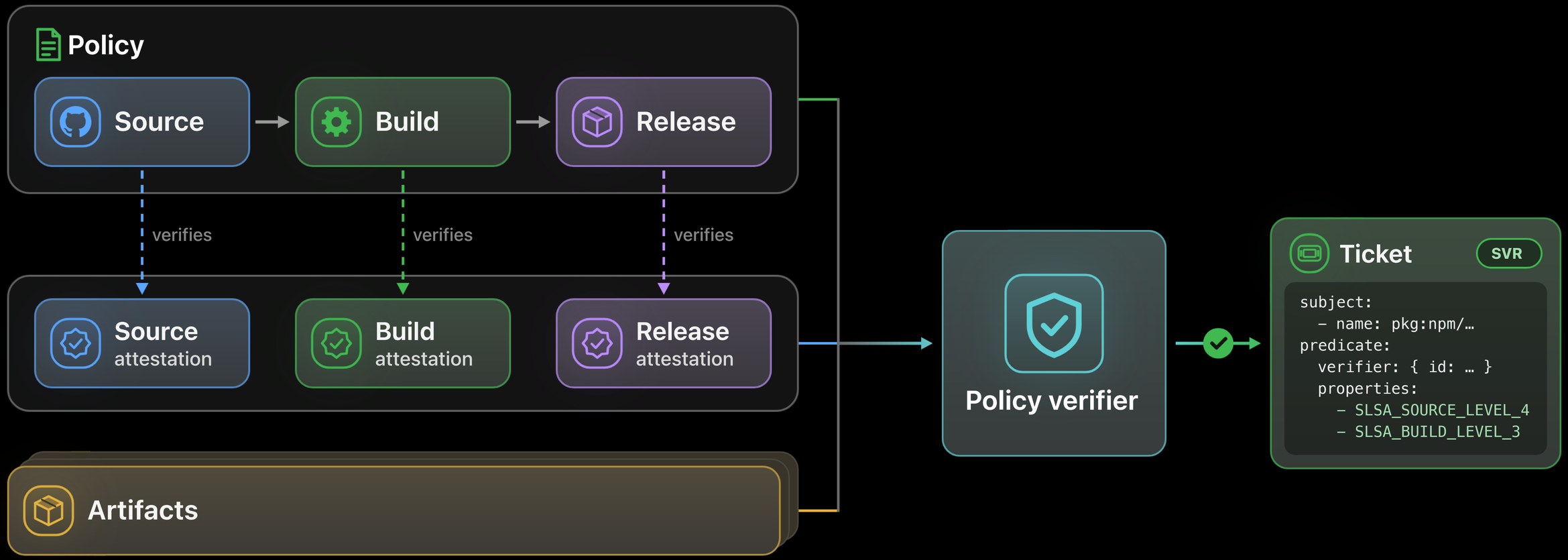
What Is OPA (Open Policy Agent)?

Policy decisions, decoupled from the services that enforce them



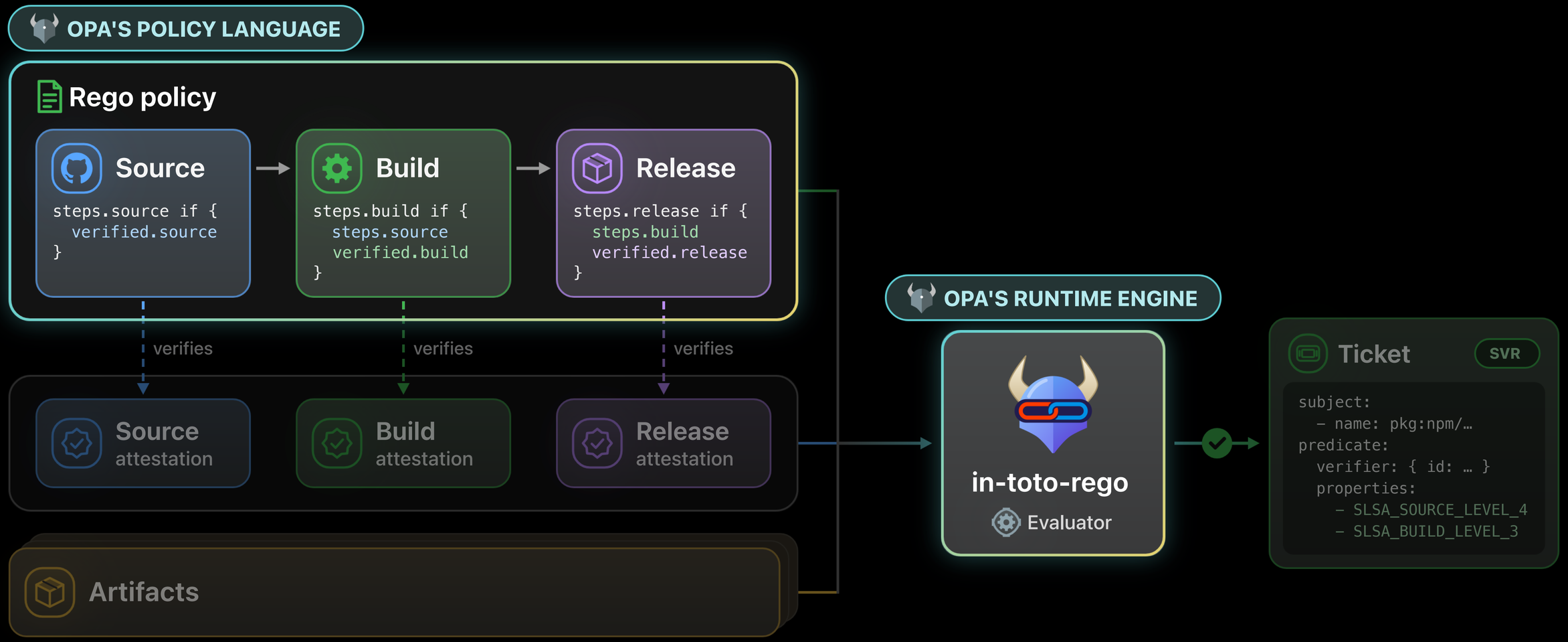
What Is in-toto?

Attestations for every step, and a policy that checks them

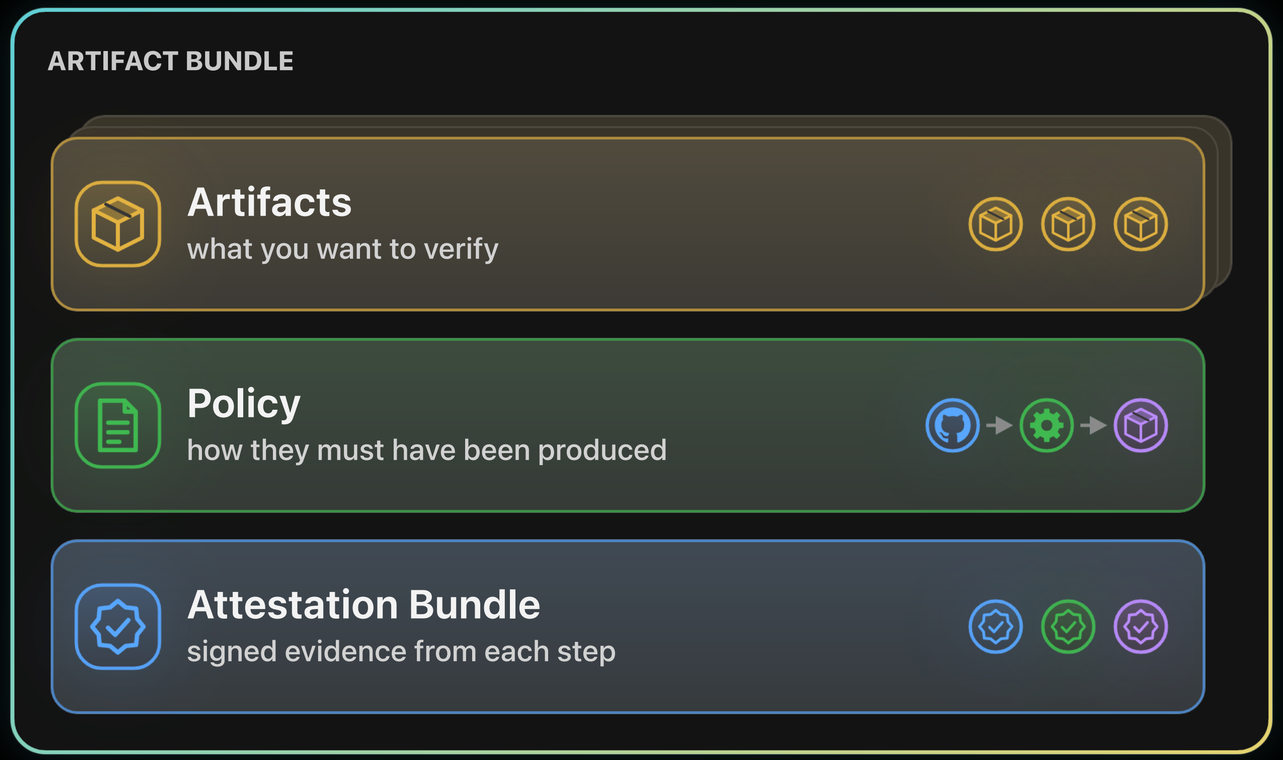


What Is in-toto-rego?

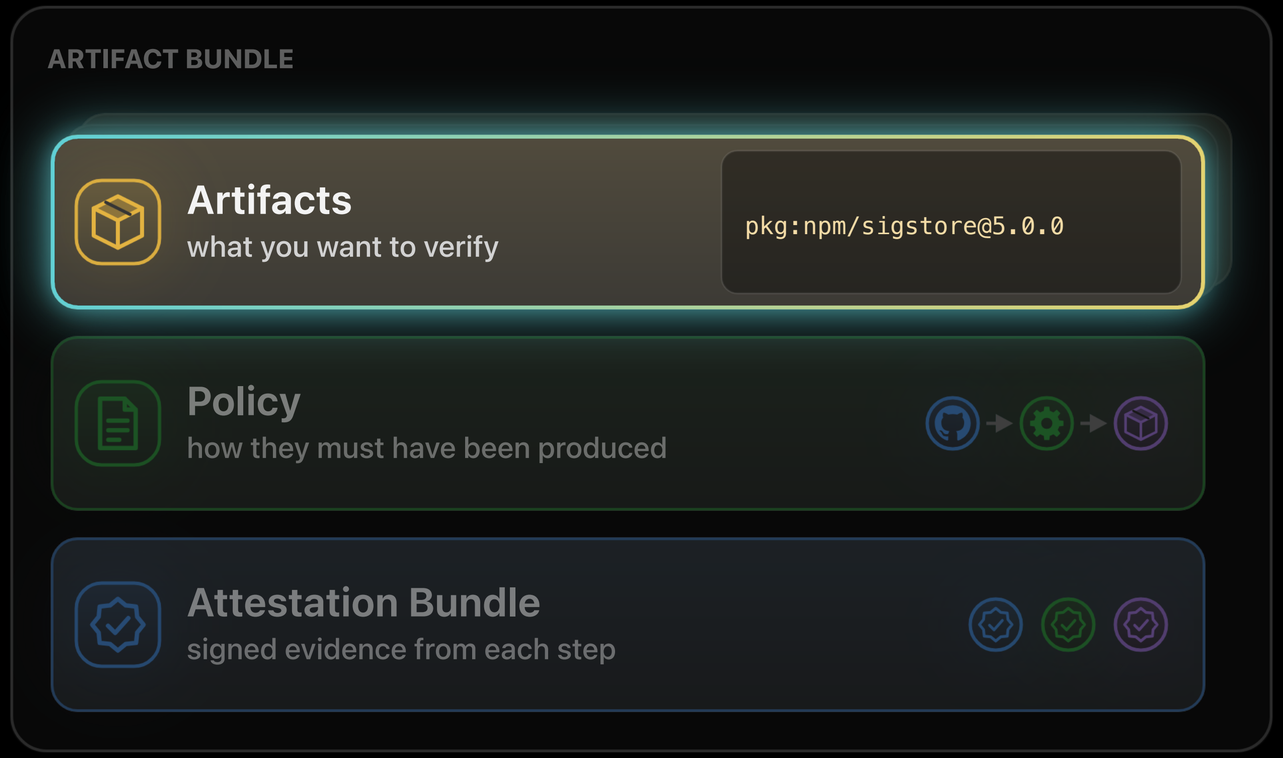
Giving in-toto a home in OPA



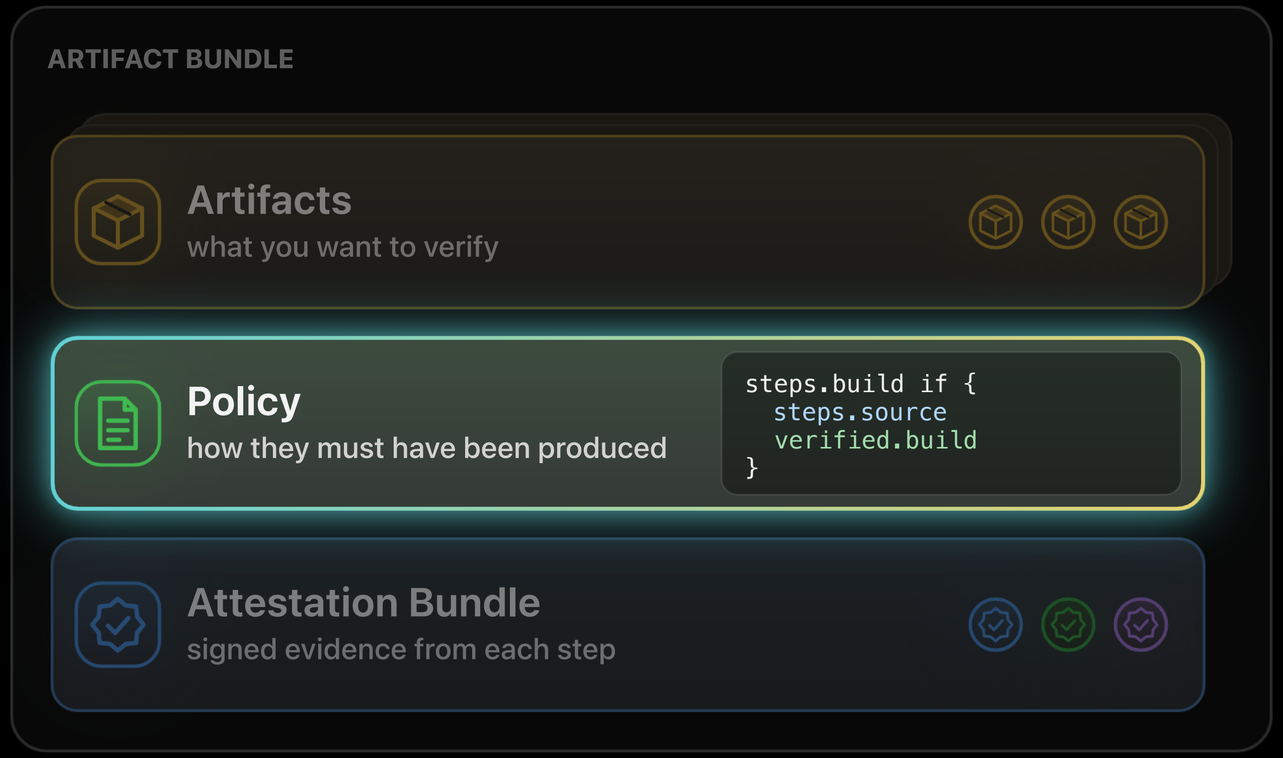
How in-toto-rego Works



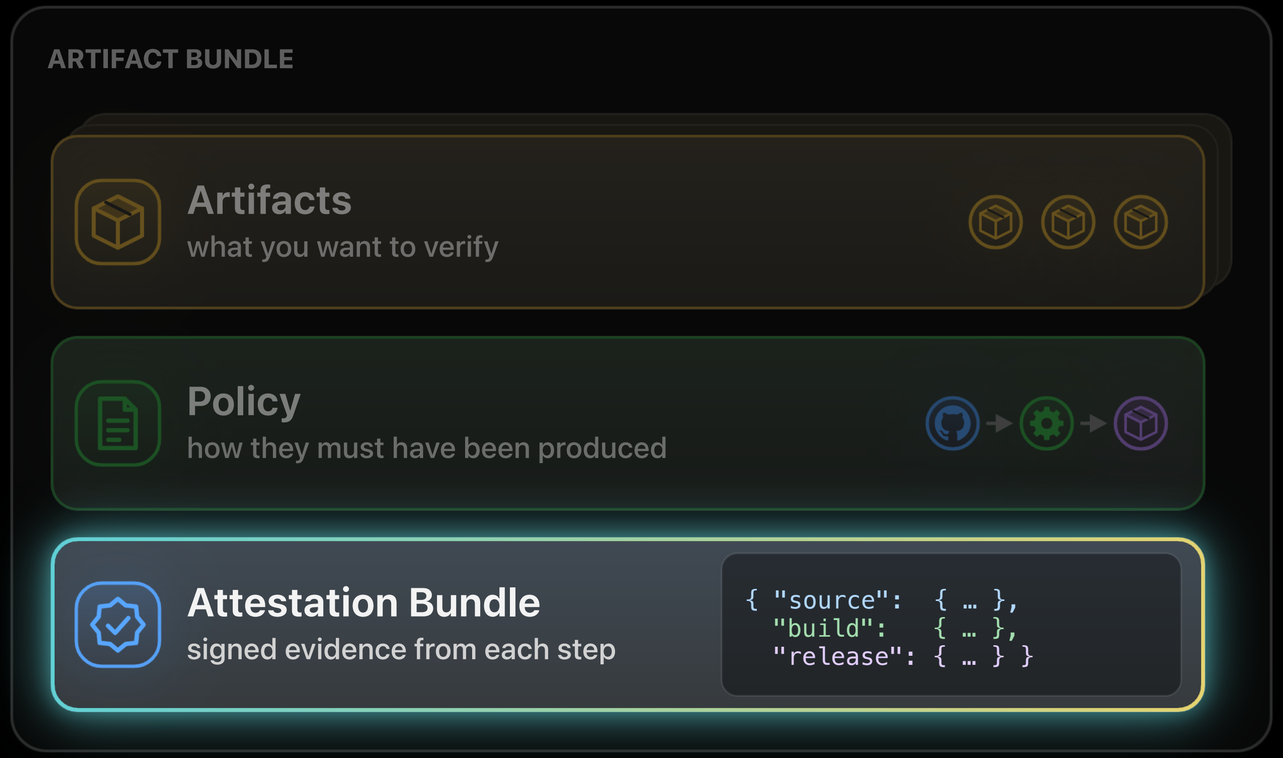
How in-toto-rego Works



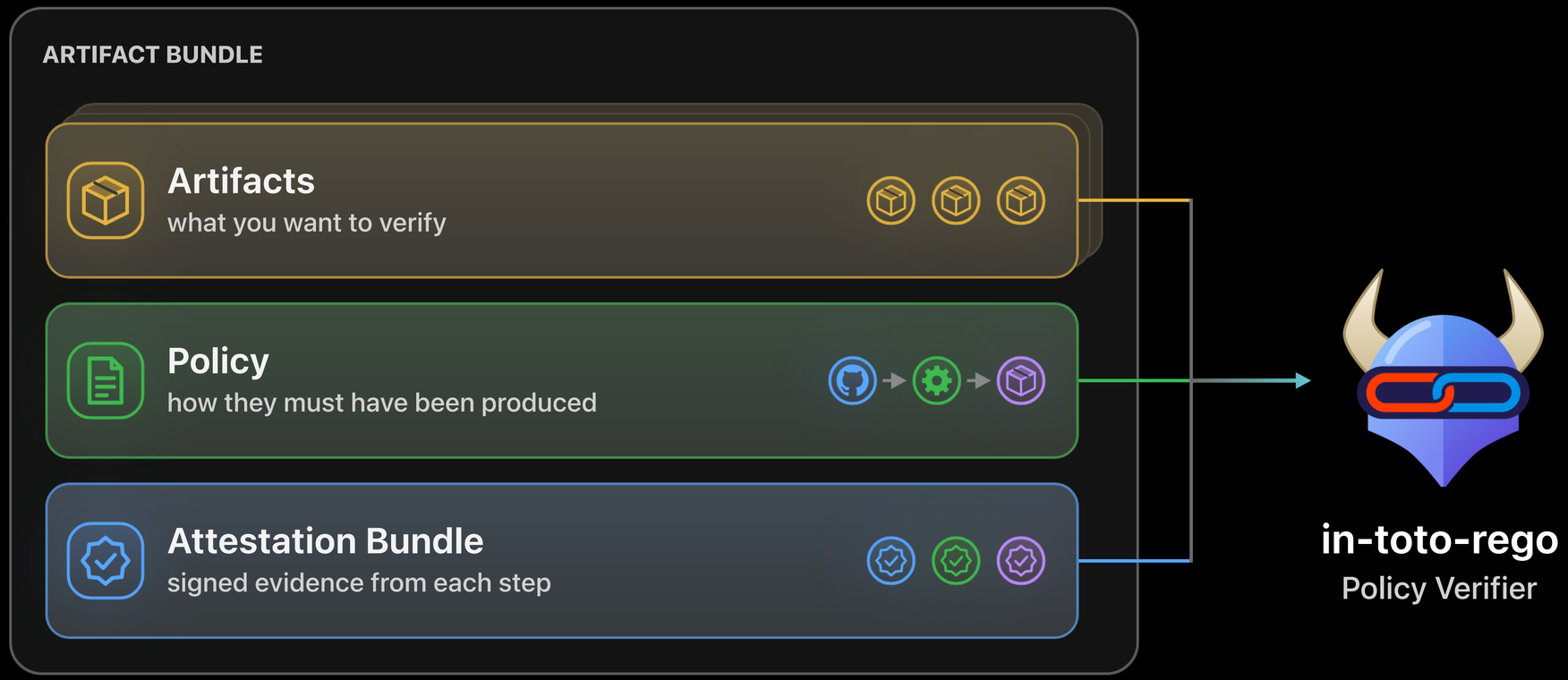
How in-toto-rego Works



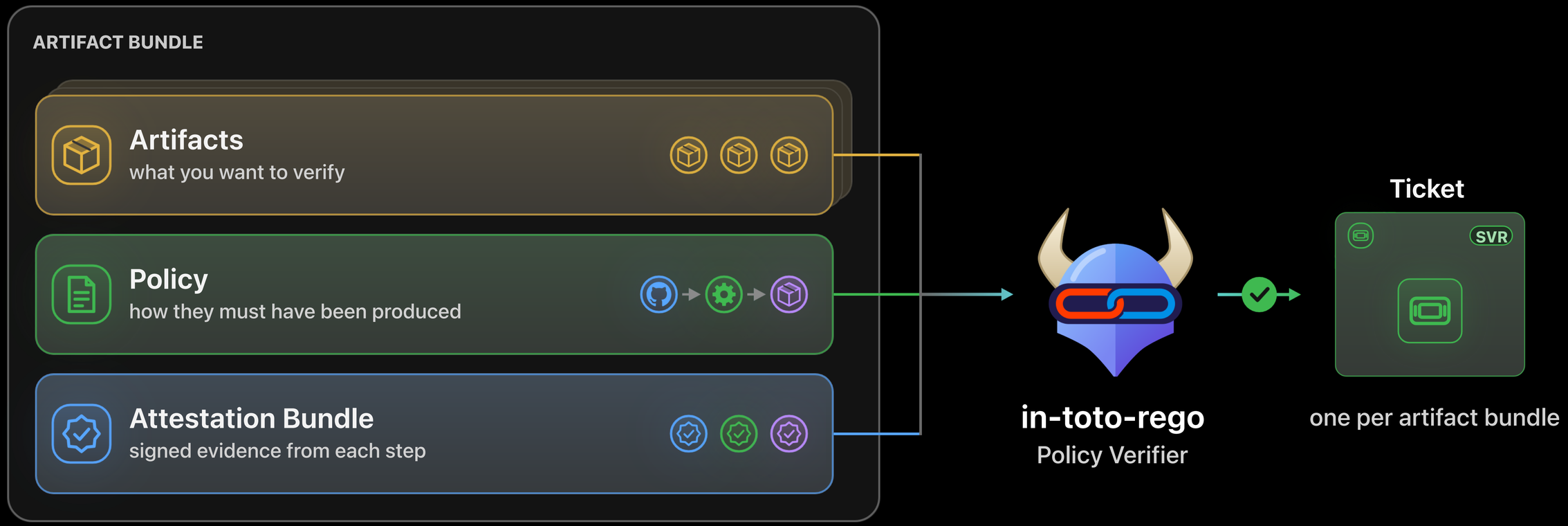
How in-toto-rego Works



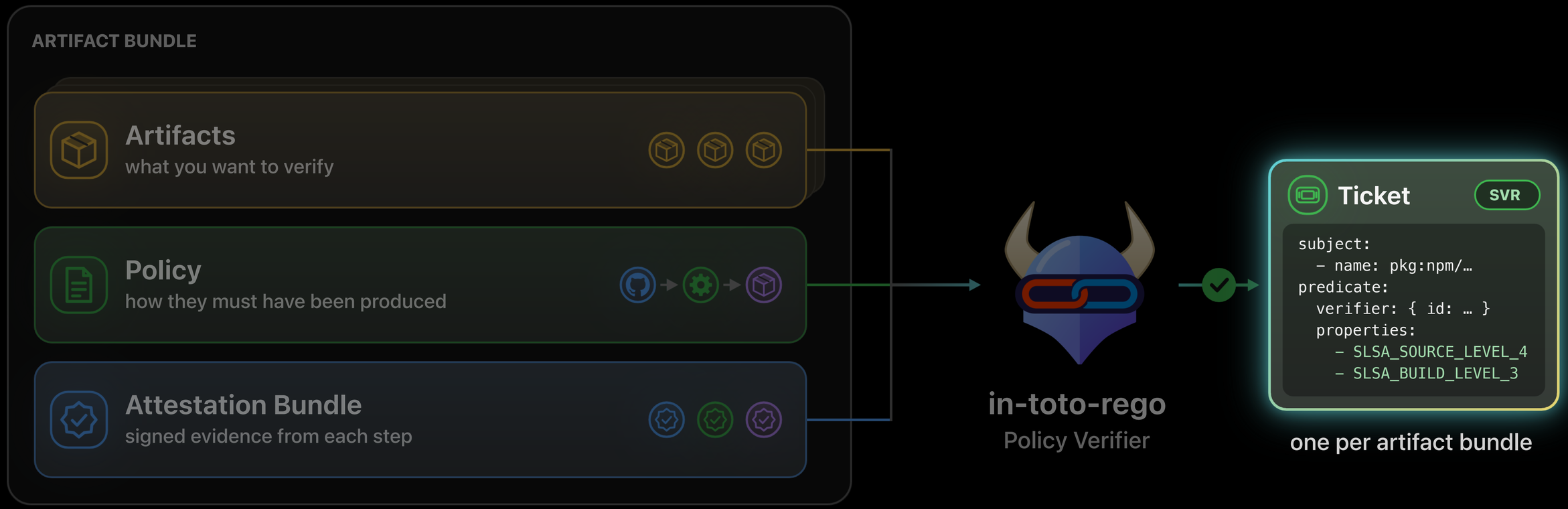
How in-toto-rego Works



How in-toto-rego Works



How in-toto-rego Works



in-toto-rego · npm supply-chain verifier

Before you install a package: was its source reviewed, was it built by its own release workflow, and is it what npm published?

in-toto · SLSA · Sigstore · OPA · Rego

Demonstrate attacks | Expert mode

1 Choose a package

Package name: such as sigstore

Find versions

Version: find the versions first

Press Enter or Find versions to read this package's versions from the npm registry.

2 Choose a policy

build-to-release build → publish

source-to-release source → build → publish

3 Choose a scenario

LEGITIMATE
A legitimate release
The real attestations from npm, and a source attestation of what GitHub records about the commit.

ATTACK
Published from a developer's laptop with a stolen npm token
No release workflow built it, so there is no build provenance for this version. npm's publish attestation is still there, because the publish itself was real.

ATTACK
A pull request from a fork triggered the release workflow
pull_request_target ran the release workflow with its secrets, so only the event that started the build gives it away. Simulated by reading the real provenance's event as pull_request_target.

ATTACK
A compromised GitHub account pushed a malicious commit
The real release workflow built and published it, so the build provenance is genuine. Only two-party review on the source would have stopped it.

Verify ▶ Needs a package name, a version.

BEFORE THE POLICY · DEMO TOOLING

PREPARE Fetch — npm registry — waiting ▶ PREPARE Trust — expected parameters — waiting ▶ PREPARE Evidence — artifact bundle — waiting ▶

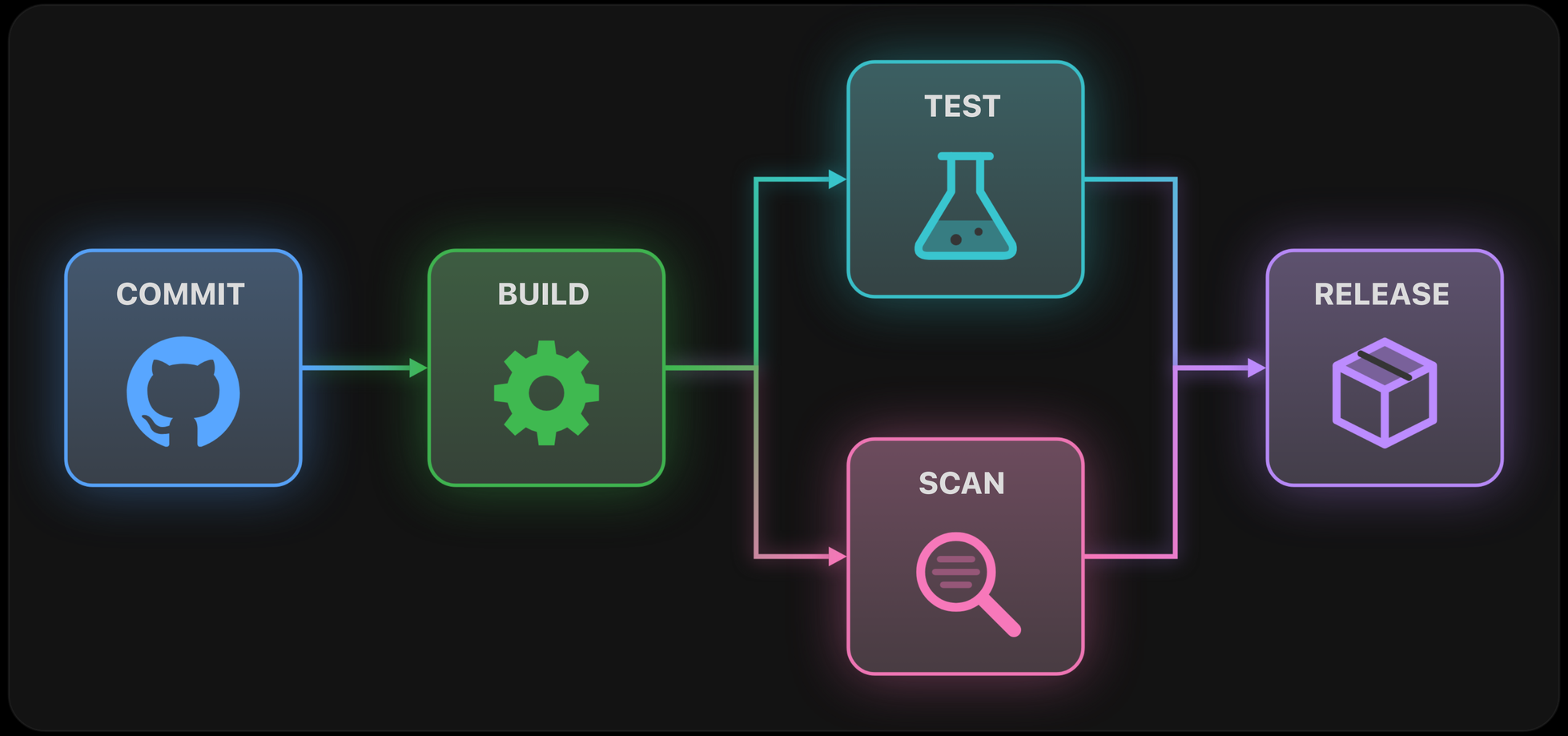
IN-TOTO-REGO

STEP Source — reviewed commit — waiting ▶ STEP Build — GitHub Actions — waiting ▶ STEP Publish — npm — waiting ▶ SVR BINDING Artifact — your tarball — waiting

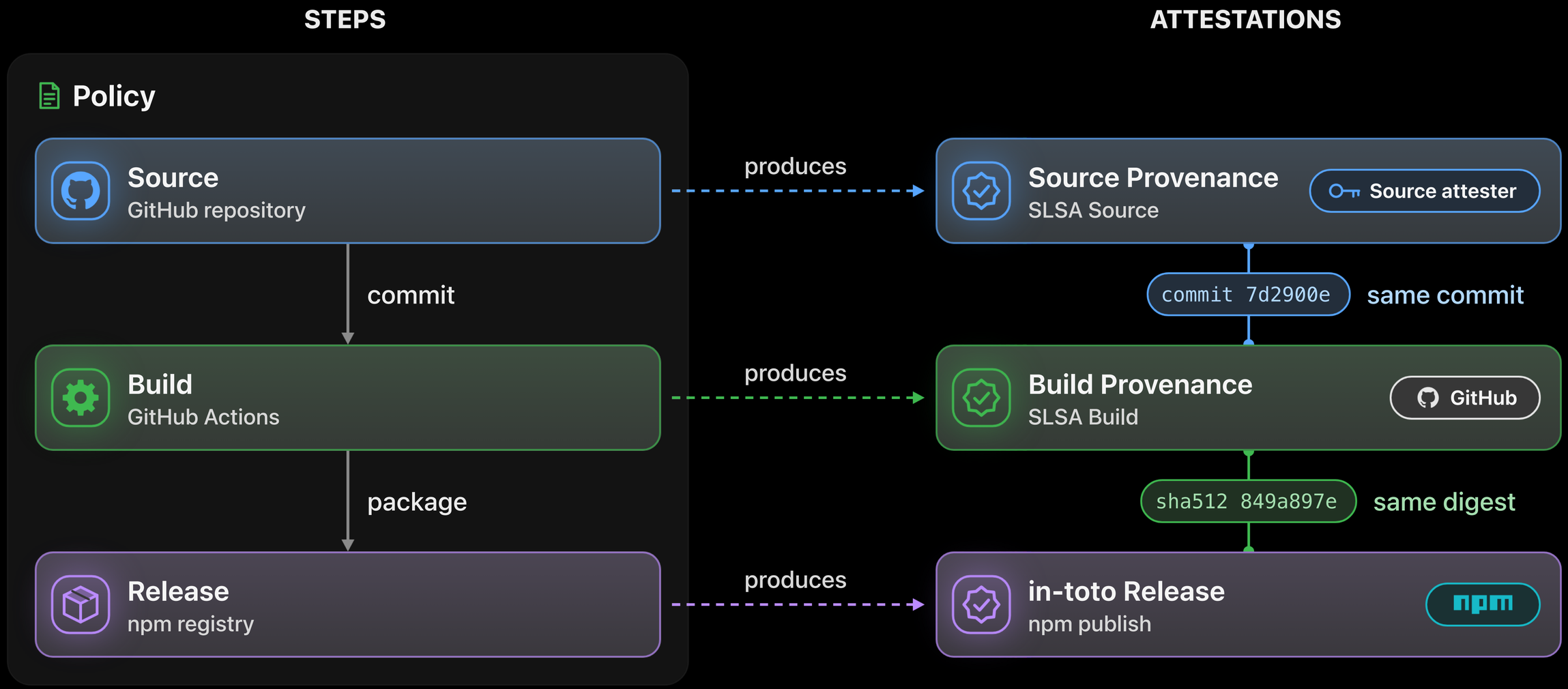
Logs | Outputs | hide

Attestations

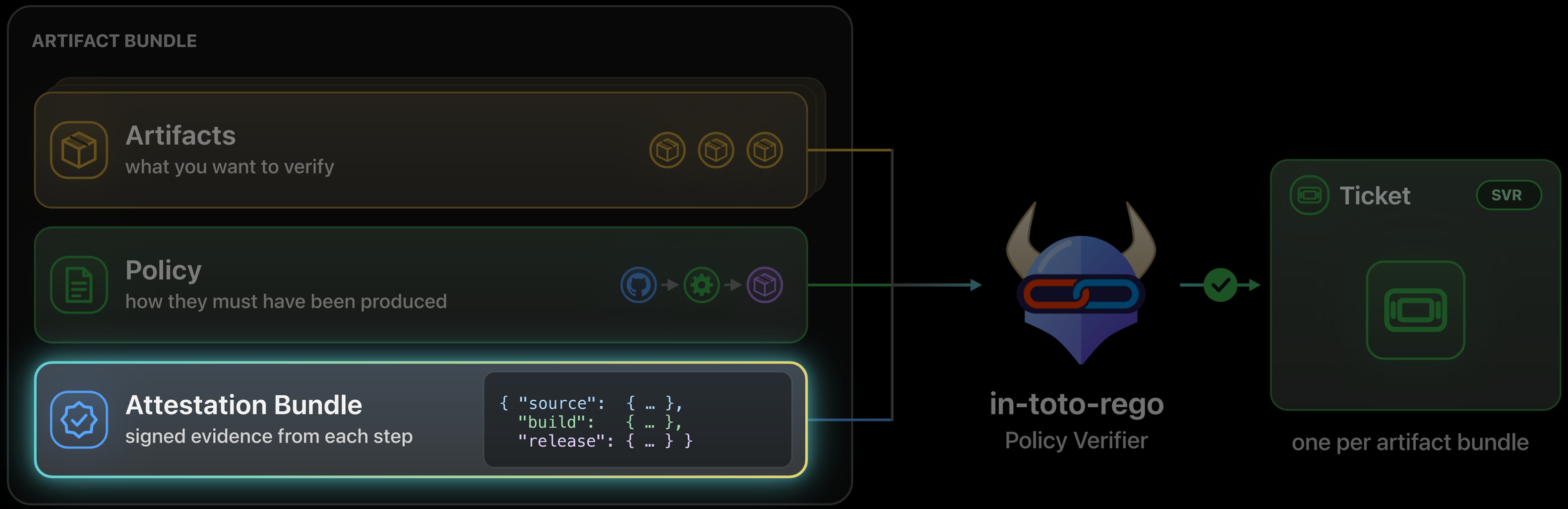
A Supply Chain Has Many Steps



Each Step Has an Attestation

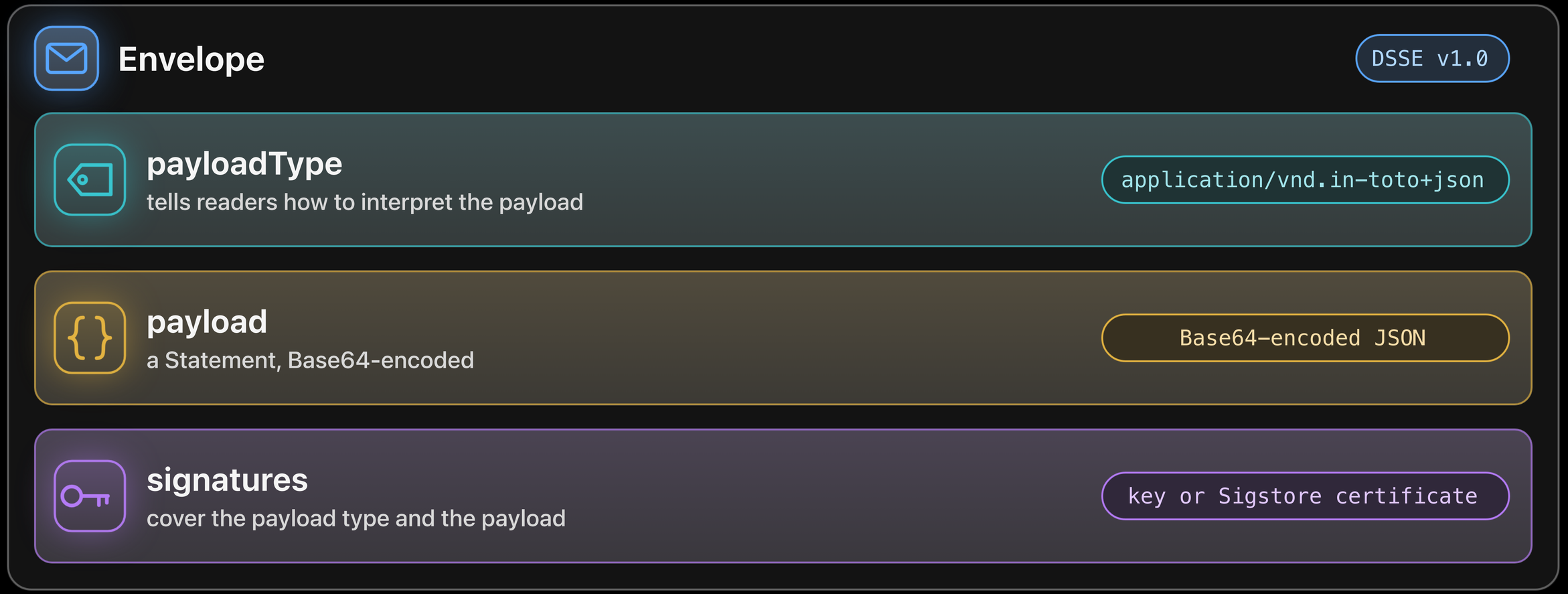


Attestation Bundle



Anatomy of an Attestation

LAYER 1 · THE ENVELOPE



Anatomy of an Attestation

LAYER 2 · THE STATEMENT

Envelope

DSSE v1.0

payloadType

tells readers how to interpret the payload

`application/vnd.in-toto+json`

Statement

the decoded payload

`Base64-encoded JSON`

`"_type": "https://in-toto.io/Statement/v1"`	which version of the Statement spec
`"subject": [ { "name": …, "digest": … } ]`	what the claim is about
`"predicateType": "<URI>"`	what kind of claim it is
`"predicate": { … }`	the claim itself

Anatomy of an Attestation

LAYER 3 · SUBJECT AND PREDICATE

Statement

Base64-encoded JSON

```
"subject": [ … ]
```

```
"predicateType": "<URI>"
```

```
"predicate": { … }
```

Example subject

what the claim is about

```
{
  "name": "pkg:npm/sigstore@5.0.0",
  "digest": {
    "sha512": "849a897e81bf7b8a…"
  }
}
```

Example predicate

SLSA build provenance

```
"predicateType":
  "https://slsa.dev/provenance/v1",
"predicate": {
  "buildDefinition": { … },
  "runDetails": { … }
}
```

Example: Source Verification Summary

Source VSA

SLSA Source

```
{
  "_type": "https://in-toto.io/Statement/v1",
  "subject": [{
    "uri": "git+https://github.com/sigstore/sigstore-js@refs/heads/main",
    "digest": { "gitCommit": "7d2900eca1c22b3f87c13987c8d4b7c9a29b733a" }
  }],
  "predicateType": "https://slsa.dev/verification_summary/v1",
  "predicate": { "verifiedLevels": ["SLSA_SOURCE_LEVEL_4"], … }
}
```



Example: Build Provenance

Build Provenance

SLSA Build


```
{
  "_type": "https://in-toto.io/Statement/v1",
  "subject": [{
    "name": "pkg:npm/sigstore@5.0.0",
    "digest": { "sha512": "849a897e81bf7b8a…b5559597fbd5f80e" }
  }],
  "predicateType": "https://slsa.dev/provenance/v1",
  "predicate": { "buildDefinition": { "resolvedDependencies": [{
    "uri": "git+https://github.com/sigstore/sigstore-js@refs/heads/main",
    "digest": { "gitCommit": "7d2900eca1c22b3f87c13987c8d4b7c9a29b733a" }
  }] }, … }
}
```




Example: Release

in-toto Release

npm publish

```
{
  "_type": "https://in-toto.io/Statement/v0.1",
  "subject": [{
    "name": "pkg:npm/sigstore@5.0.0",
    "digest": { "sha512": "849a897e81bf7b8a…b5559597fbd5f80e" }
  }],
  "predicateType":
    "https://github.com/npm/attestation/tree/main/specs/publish/v0.1",
  "predicate": { "name": "sigstore", "version": "5.0.0", … }
}
```



Standard Predicate Types



CycloneDX

CycloneDX BOM for software artifacts.



Link

For migration from in-toto 0.9.



Reference

References documents that
are relevant to some resource.



Release

Details an artifact that is part of a given release version.



Runtime Traces

Captures runtime traces of
software supply chain operations.



SCAI Report

Evidence-based assertions about software
artifact and supply chain attributes or behavior.

Standard Predicate Types, Continued



SLSA Provenance

Describes how an artifact or set of artifacts was produced.



SLSA Verification Summary

SLSA verification decision about a software artifact.



SPDX2 and SPDX3

SPDX-formatted BOM for software artifacts.



Simple Verification Result

Evidence that an artifact has been evaluated against one or more policies.



Test Result

A generic schema to express results of any type of tests.



VULNS

Defines the metadata to share the results of vulnerability scanning on software artifacts.

Bring Your Own Predicate

You can also define and use your private predicate types.

Your own predicate type

```
{
  "_type": "https://in-toto.io/Statement/v1",
  "subject": [ … ],
  "predicateType":
    "https://example.com/my-predicate/v1",
  "predicate": { … }
}
```

No permission required
Nobody has to approve your type.

No standardization required
It doesn't need to become a standard.

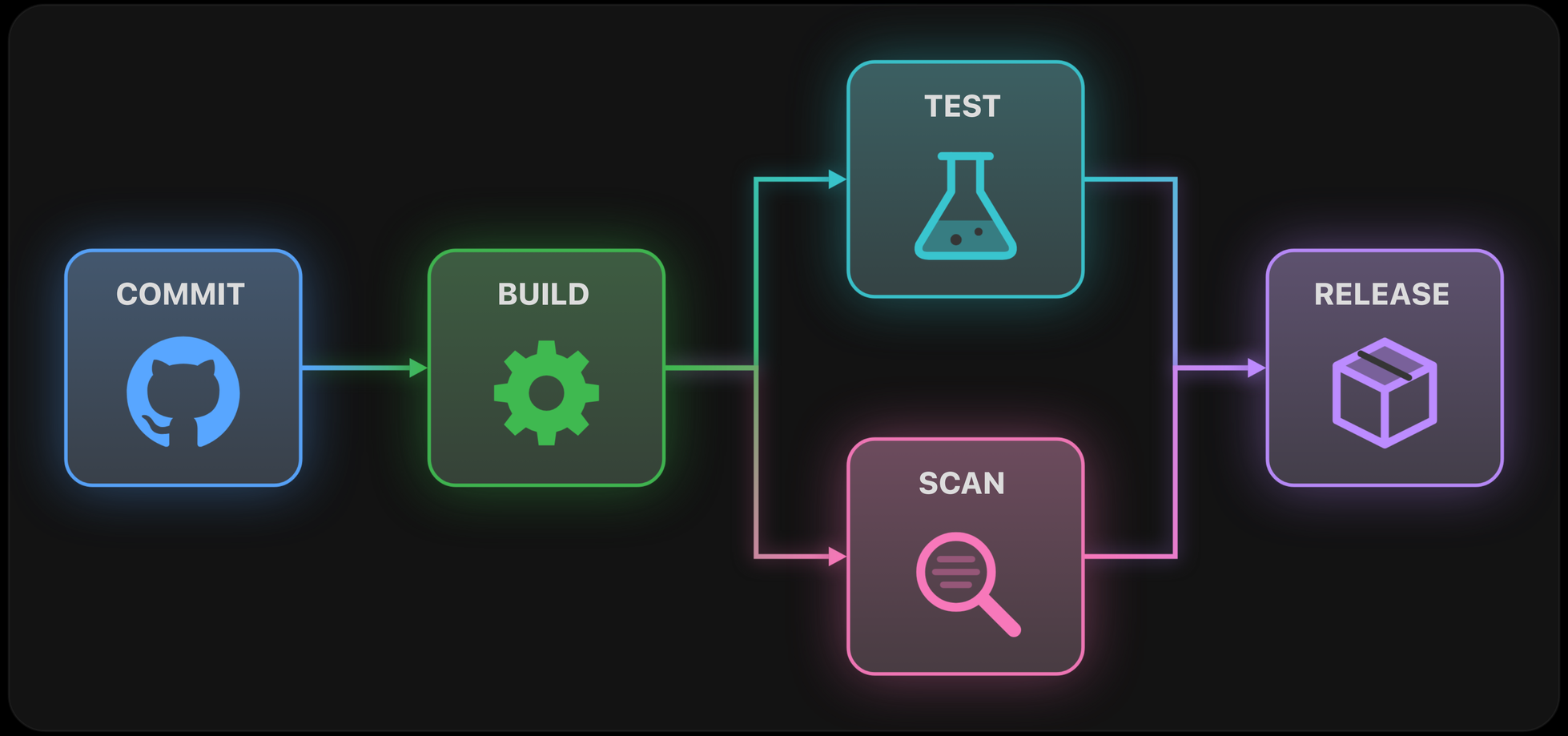
No open-sourcing required
Keep the schema private if you like.

Policies

Policies: Turning Attestations into Compliance



A Supply Chain Has Many Steps



A Policy Is a Few Things

Policy

signed by cryptographic identities

THE SIGNED PAYLOAD HAS FOUR PARTS

Expiration

After this time, the policy should no longer be trusted.

Identities

Used to verify the attestations in any attestation bundle.

Steps

Each step in the supply chain produces or emits an attestation.

Inspections

Bind an artifact and its attestation bundle to this policy.

A Policy Has Many Steps

Policy

signed by cryptographic identities

THE SIGNED PAYLOAD HAS FOUR PARTS

Expiration

After this time, the policy should no longer be trusted.

Identities

Used to verify the attestations in any attestation bundle.

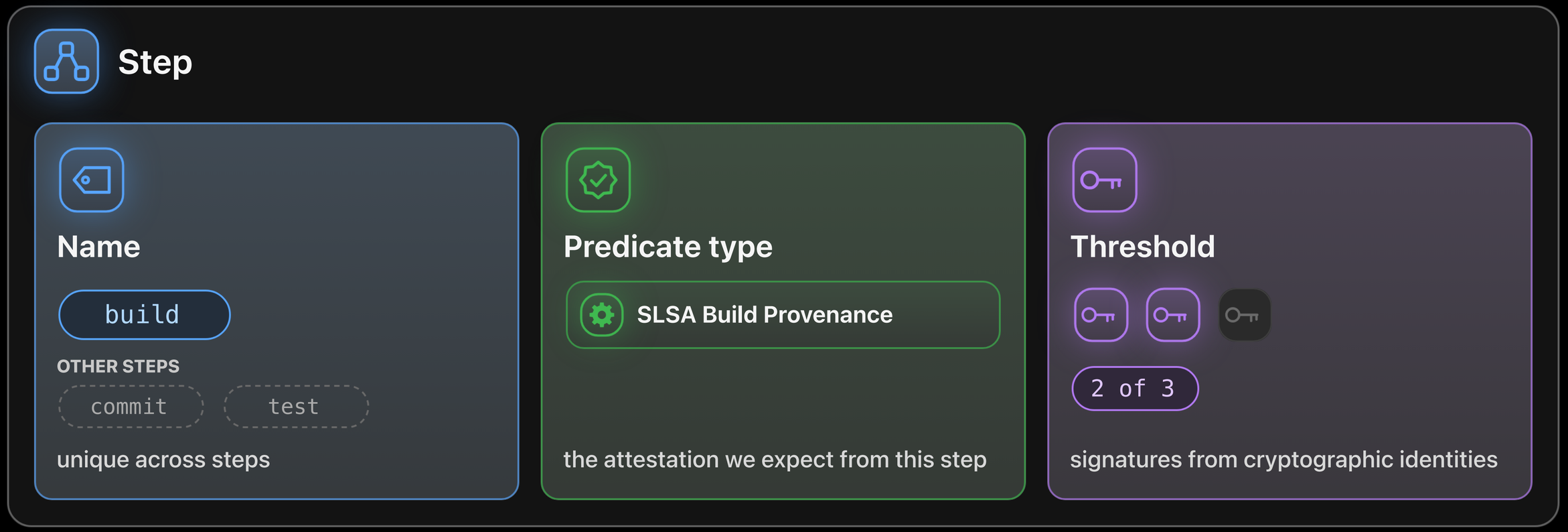
Steps

Each step in the supply chain produces or emits an attestation.

Inspections

Bind an artifact and its attestation bundle to this policy.

What Is a Step?



Predicate Validators: Checking Within a Step

Source VSA
two-party review

```
"SLSA_SOURCE_LEVEL_4" in predicate.verifiedLevels
```

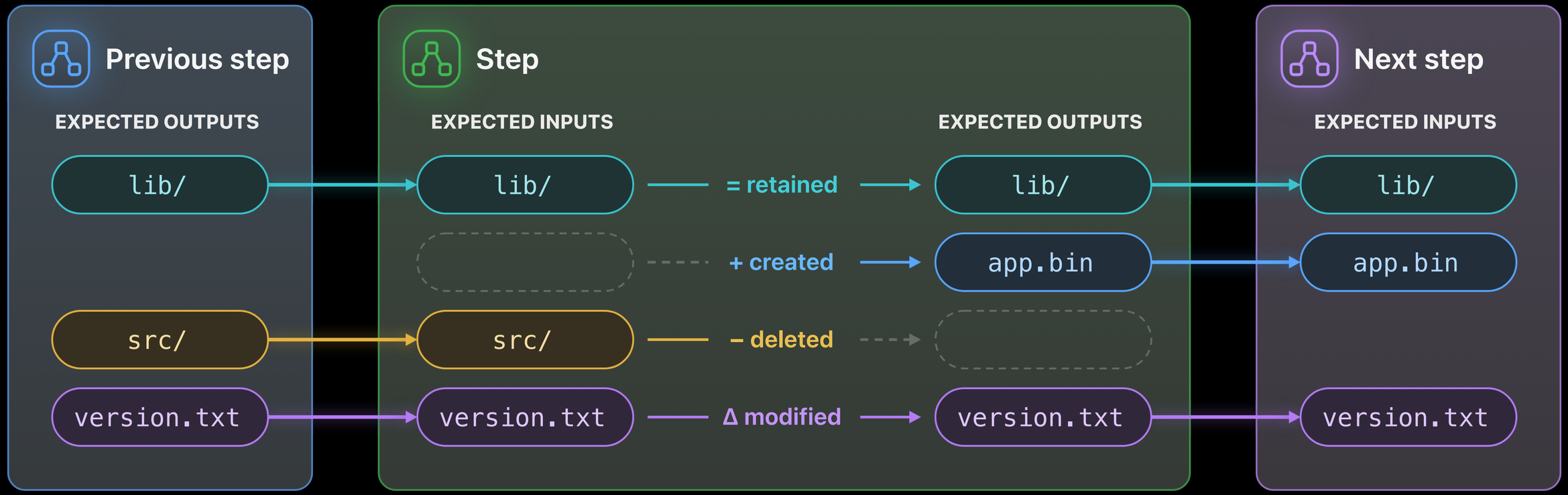
Build Provenance
expected repository and workflow

```
workflow.repository == "https://github.com/sigstore/sigstore-js"
workflow.path == ".github/workflows/release.yml"
```

in-toto Release
package, version and registry

```
predicate.name == "sigstore"
predicate.version == "5.0.0"
predicate.registry == "https://registry.npmjs.org"
```

Tracing Artifact Flow Between Steps



Artifact Rules: Linking Between Steps

`ALLOW`	Pattern must appear in Inputs/Outputs
`CREATE`	Pattern must be in Outputs not Inputs
`DELETE`	Pattern must be in Inputs not Outputs
`DISALLOW`	Pattern must not appear in Inputs/Outputs
`MATCH`	Preserve artifacts across Steps
`MODIFY`	Artifacts changed across Steps
`REQUIRE`	Name must appear in Inputs/Outputs
	**Applied in order like Firewall rules**

A Policy Has Inspections

Policy

signed by cryptographic identities

THE SIGNED PAYLOAD HAS FOUR PARTS

Expiration

After this time, the policy should no longer be trusted.

Identities

Used to verify the attestations in any attestation bundle.

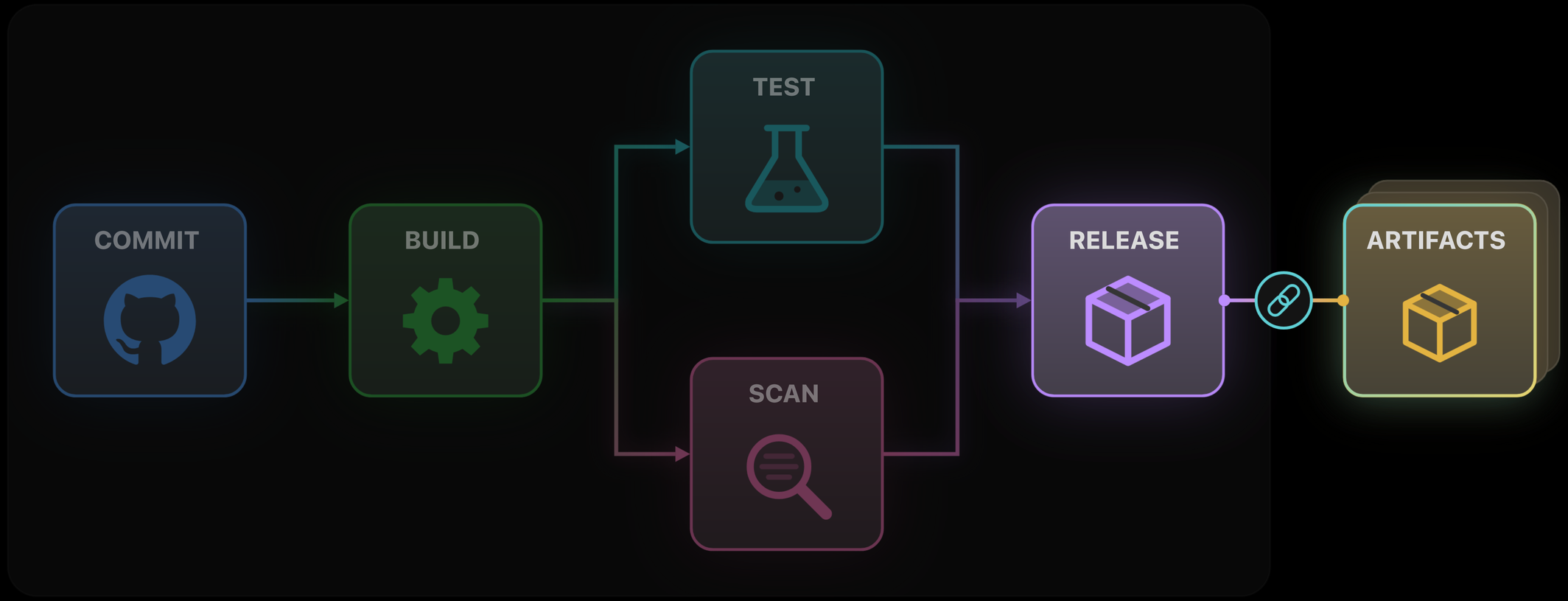
Steps

Each step in the supply chain produces or emits an attestation.

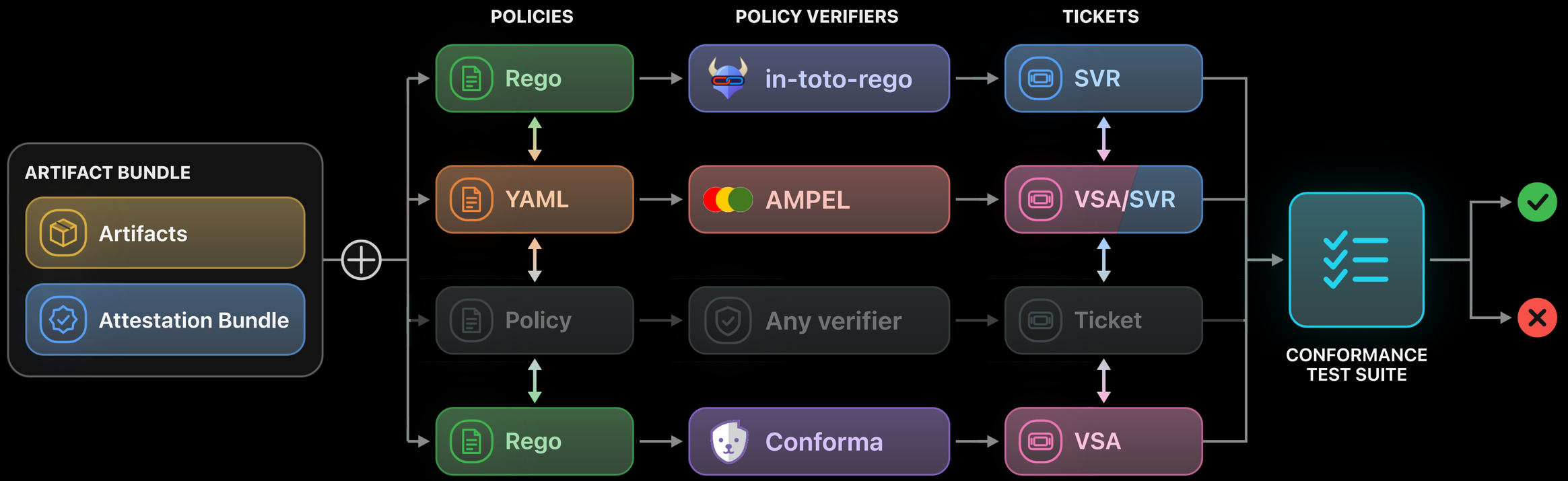
Inspections

Bind an artifact and its attestation bundle to this policy.

Inspections: Binding Artifacts to End Steps

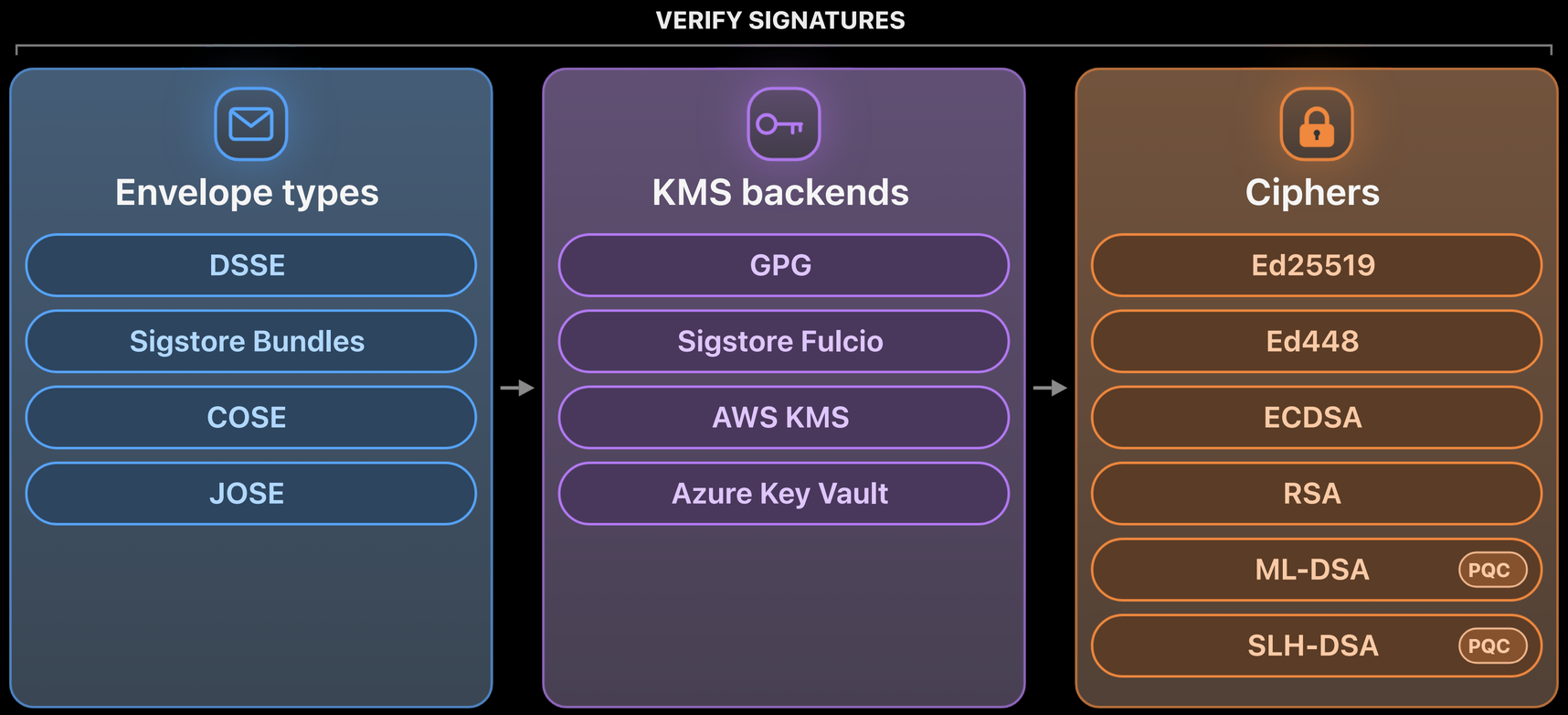


Policy Framework: Different Languages, One Ticket



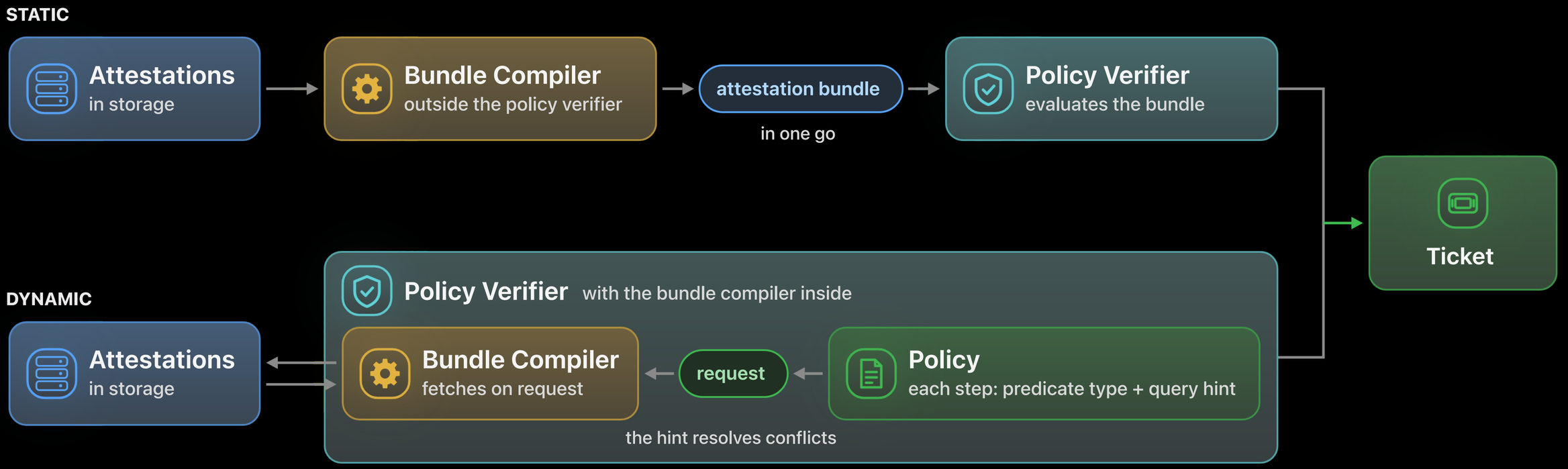
Implementation Choice: Bring Your Own Crypto

Any envelope type, KMS backend, or cipher, with the same results



Implementation Choice: Bundle Compilation

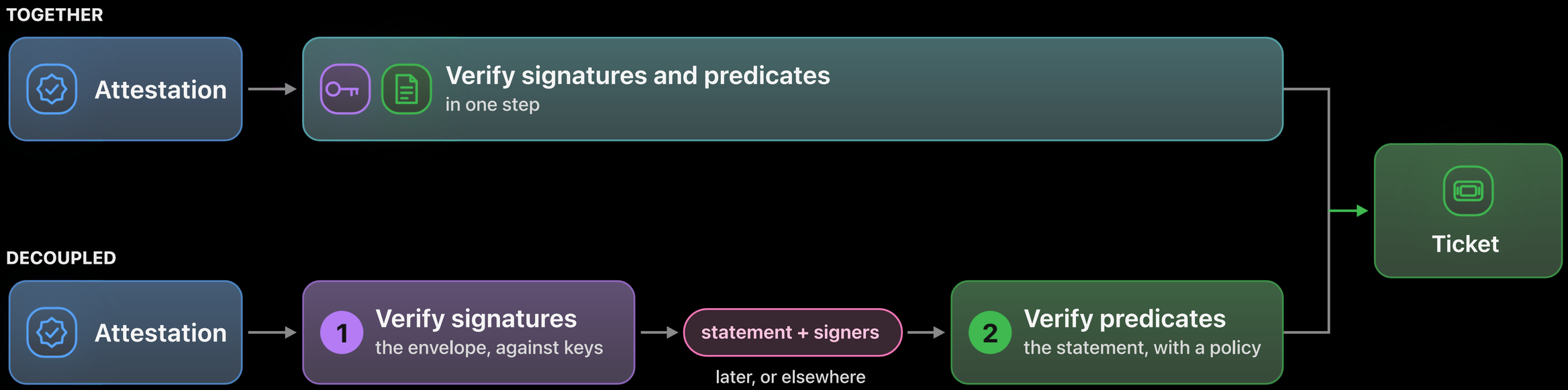
Compile attestation bundles outside the verifier, or inside it, on demand



Hand the verifier a whole bundle at once, or let it compile one from each step's query hints.

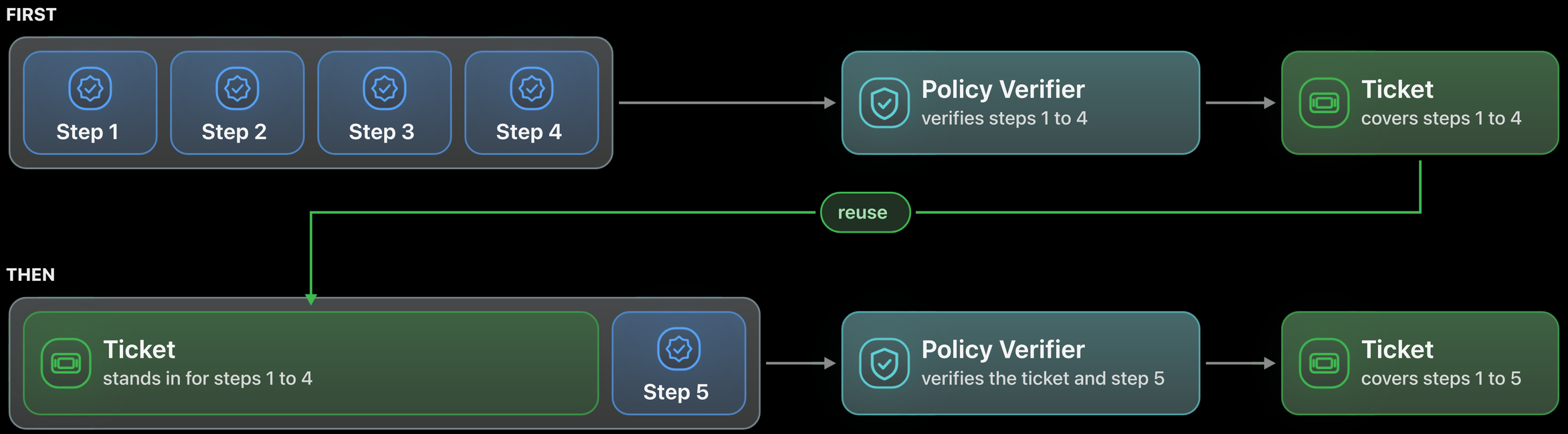
Implementation Choice: Decoupled Verification

Verify signatures first, then predicates, together or apart



Implementation Choice: Partial Verification

Verify in parts, reusing the tickets of the parts already verified



Policy Verifier: in-toto-rego

Policy Framework Compliance



Implementation Choices

OPA DEFAULTS

Policy language
Rego

Policy runtime
OPA

Bundle compilation
Static (ahead of time)

BUILT FOR IN-TOTO-REGO

Keyless and File-Backed Classical Crypto
DSSE, Sigstore Bundles; Cryptosigner, Fulcio; Ed25519, ECDSA, RSA

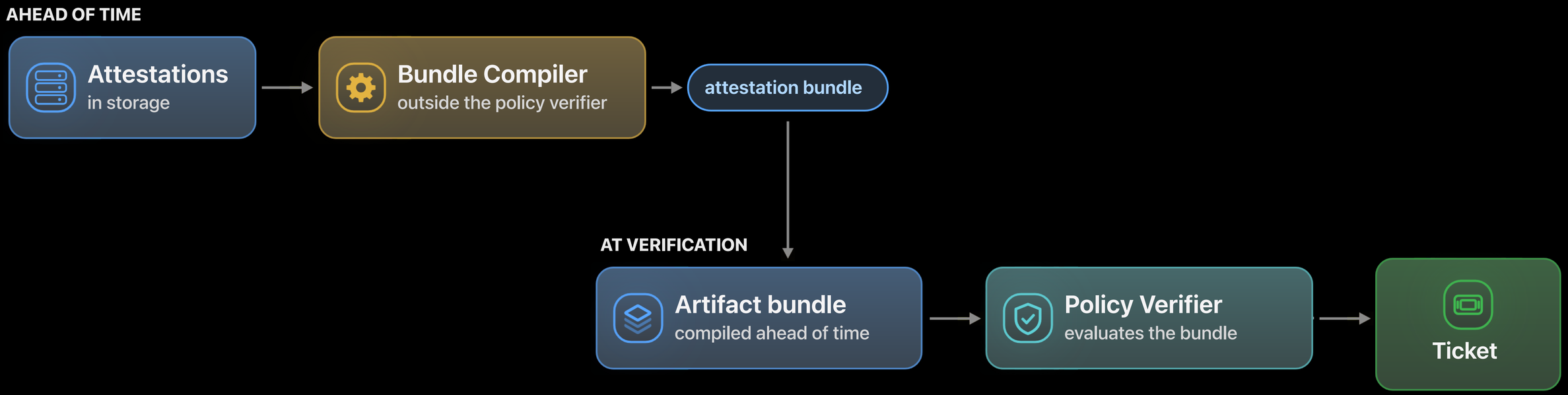
Artifact Rules
List/set comprehensions, consuming artifacts, returning leftovers, chaining rules

Policies
Policies as code

Ticket predicate type
Simple Verification Result (SVR)

Static Bundle Compilation

Compile the bundle once, ahead of time, then hand it to the verifier



Keyless and File-Backed Classical Crypto

With sigstore-go and go-securesystemslib



Artifact Rules in Rego

List or set comprehensions over artifacts

Artifact rules ALLOW CREATE DELETE DISALLOW MATCH MODIFY REQUIRE

```
allow(field, pattern, artifacts) := leftovers if {
    _valid_descriptors(field, artifacts)
    leftovers := [artifact |
        some artifact in artifacts
        not glob.match(pattern, null, artifact[field])
    ]
}
```

Artifact Rules in Rego

Mandatory instead of optional fields in ResourceDescriptors

Artifact rules ALLOW CREATE DELETE DISALLOW MATCH MODIFY REQUIRE

```
allow(field, pattern, artifacts) := leftovers if {
    _valid_descriptors(field, artifacts)
    leftovers := [artifact |
        some artifact in artifacts
        not glob.match(pattern, null, artifact[field])
    ]
}

_valid_descriptors(field, artifacts) if {
    field in {"name", "uri"}
    every artifact in artifacts {
        json.match_schema(artifact, data.in_toto.schemas.resource_descriptor)[0]
        is_string(artifact[field])
    }
}
```

Artifact Rules in Rego

Consuming artifacts, returning leftovers, chaining rules

Artifact rules ALLOW CREATE DELETE DISALLOW MATCH MODIFY REQUIRE

```
allow(field, pattern, artifacts) := leftovers if {
    _valid_descriptors(field, artifacts)
    leftovers := [artifact |
        some artifact in artifacts
        not glob.match(pattern, null, artifact[field])
    ]
}

disallow(field, pattern, artifacts) := artifacts if {
    _valid_descriptors(field, artifacts)
    every artifact in artifacts {
        not glob.match(pattern, null, artifact[field])
    }
}

rules.disallow("name", "*", rules.allow("name", "dist/*", outputs))
```

Policies

Three steps, each signed, bound into one chain

POLICY `in_toto.policies.source_to_release`

Source

```
steps.source := {"outputs": outputs} if {
    outputs := verified.source.statement.subject
}
```

Build

```
steps.build := {"inputs": inputs, "outputs": outputs} if {
    inputs := verified.build.statement.predicate.buildDefinition.resolvedDependencies
    rules.match(…, inputs, …, steps.source.outputs)
    outputs := verified.build.statement.subject
}
```

Publish

```
steps.publish := {"inputs": inputs, "outputs": outputs} if {
    inputs := verified.publish.statement.subject
    rules.match(…, inputs, …, steps.build.outputs)
    outputs := inputs
}
```

Inspection

```
svr := predicates.svr(…, steps.publish.outputs, …)
```

Policies

Each step trusts exactly one signer

POLICY `in_toto.policies.source_to_release`

Source

```
verified.source.statement := dsse.verify_dsse_1(
    input.attestations.source, source_attester_key)
    # identity: the key 59acbde357d575dc…74368d7c45b50d2e
steps.source := {"outputs": outputs} if {
    outputs := verified.source.statement.subject
}
```

the source attester's key

Build

```
verified.build.statement := sigstore.verify_sigstore_dsse_bundle_fulcio(
    input.attestations.build, constants.gha_oidc_issuer, identity)
    # identity: https://github.com/sigstore/sigstore-js/.github/workflows/release.yml@refs/heads/main
steps.build := {"inputs": inputs, "outputs": outputs} if {
    inputs := verified.build.statement.predicate.buildDefinition.resolvedDependencies
    outputs := verified.build.statement.subject
}
```

GitHub Actions certificate

Publish

```
verified.publish.statement := sigstore.verify_sigstore_dsse_bundle_npm(
    input.attestations.publish)
    # identity: npm's key SHA256:DhQ8wR5APBvFHLF/+Tc+AYvPOdTpcIDqOhxsBHRwC7U, which the bundle names
steps.publish := {"inputs": inputs, "outputs": outputs} if {
    inputs := verified.publish.statement.subject
    outputs := inputs
}
```

npm's attestation key

Policies

In each statement: the in-toto statement version, the predicate type, and a predicate validator

POLICY `in_toto.policies.source_to_release`

Source

```
verified.source.statement := statement if {
    statement := statements.get_statement(
        dsse.verify_dsse_1(input.attestations.source, source_attester_key),
        constants.statement_v1,
        constants.vsa_v1,
    )
    "SLSA_SOURCE_LEVEL_4" in statement.predicate.verifiedLevels
}
```

predicate type

predicate validator

Build

```
verified.build.statement := statement if {
    statement := statements.get_statement(
        sigstore.verify_sigstore_dsse_bundle_fulcio(
            input.attestations.build,
            constants.gha_oidc_issuer,
            identity,
        ),
        constants.statement_v1,
        constants.provenance_v1,
    )
    statement.predicate.runDetails.builder.id == constants.gha_builder_id
}
```

predicate type

predicate validator

Publish

```
verified.publish.statement := statement if {
    statement := statements.get_statement(
        sigstore.verify_sigstore_dsse_bundle_npm(input.attestations.publish),
        constants.statement_v0_1,
        constants.publish_v1,
    )
    statement.predicate.name == parameters.package_name
}
```

predicate type

predicate validator

Policies

The build used the commit the source vouches for

POLICY `in_toto.policies.source_to_release`

Source

```
steps.source := {"outputs": outputs} if {
    outputs := verified.source.statement.subject
}
```

Build

```
steps.build := {"inputs": inputs, "outputs": outputs} if {
    rules.match(…, inputs, …, steps.source.outputs)
    outputs := verified.build.statement.subject
}
```

same commit

Publish

```
steps.publish := {"inputs": inputs, "outputs": outputs} if {
    rules.match(…, inputs, …, steps.build.outputs)
    outputs := inputs
}
```

Inspection

```
svr := predicates.svr(…, steps.publish.outputs, …)
```

Policies

npm published the digest the build made

POLICY `in_toto.policies.source_to_release`

Source

```
steps.source := {"outputs": outputs} if {
    outputs := verified.source.statement.subject
}
```

Build

```
steps.build := {"inputs": inputs, "outputs": outputs} if {
    rules.match(…, inputs, …, steps.source.outputs)
    outputs := verified.build.statement.subject
}
```

Publish

```
steps.publish := {"inputs": inputs, "outputs": outputs} if {
    rules.match(…, inputs, …, steps.build.outputs)
    outputs := inputs
}
```

same digest

Inspection

```
svr := predicates.svr(…, steps.publish.outputs, …)
```

Policies

The SVR covers only what npm published

POLICY `in_toto.policies.source_to_release`

Source

```
steps.source := {"outputs": outputs} if {
    outputs := verified.source.statement.subject
}
```

Build

```
steps.build := {"inputs": inputs, "outputs": outputs} if {
    rules.match(…, inputs, …, steps.source.outputs)
    outputs := verified.build.statement.subject
}
```

Publish

```
steps.publish := {"inputs": inputs, "outputs": outputs} if {
    rules.match(…, inputs, …, steps.build.outputs)
    outputs := inputs
}
```

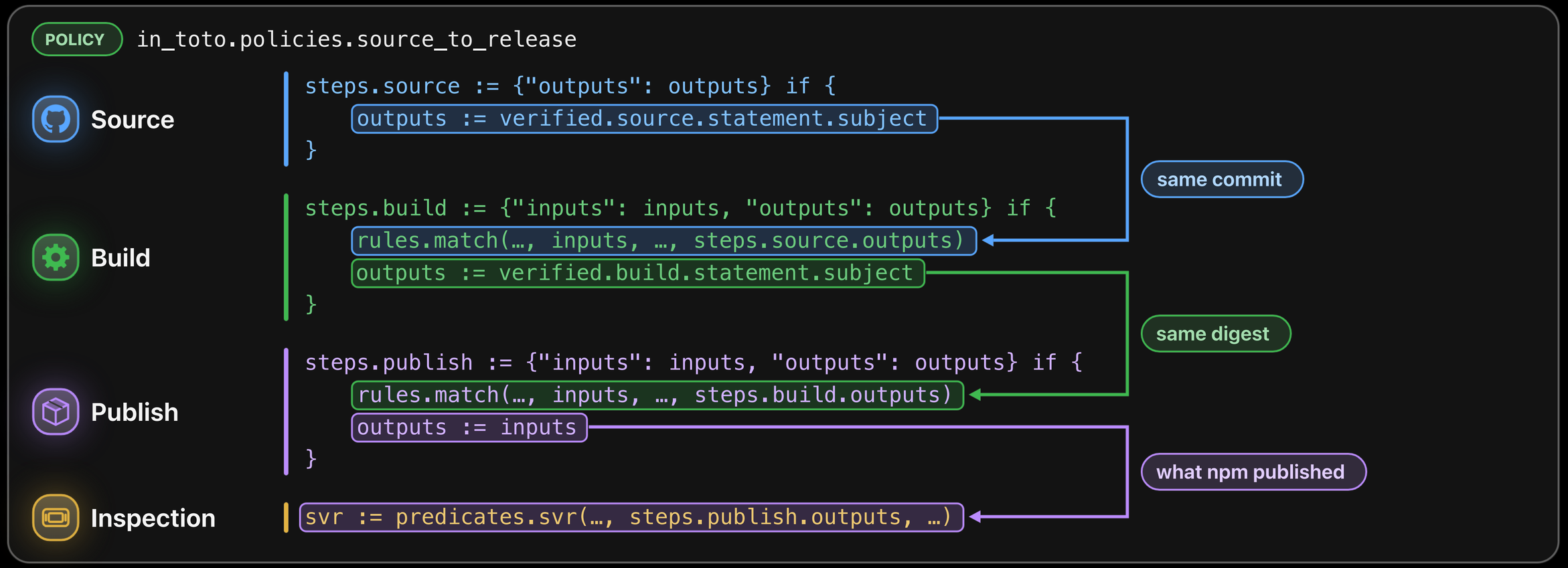
what npm published

Inspection

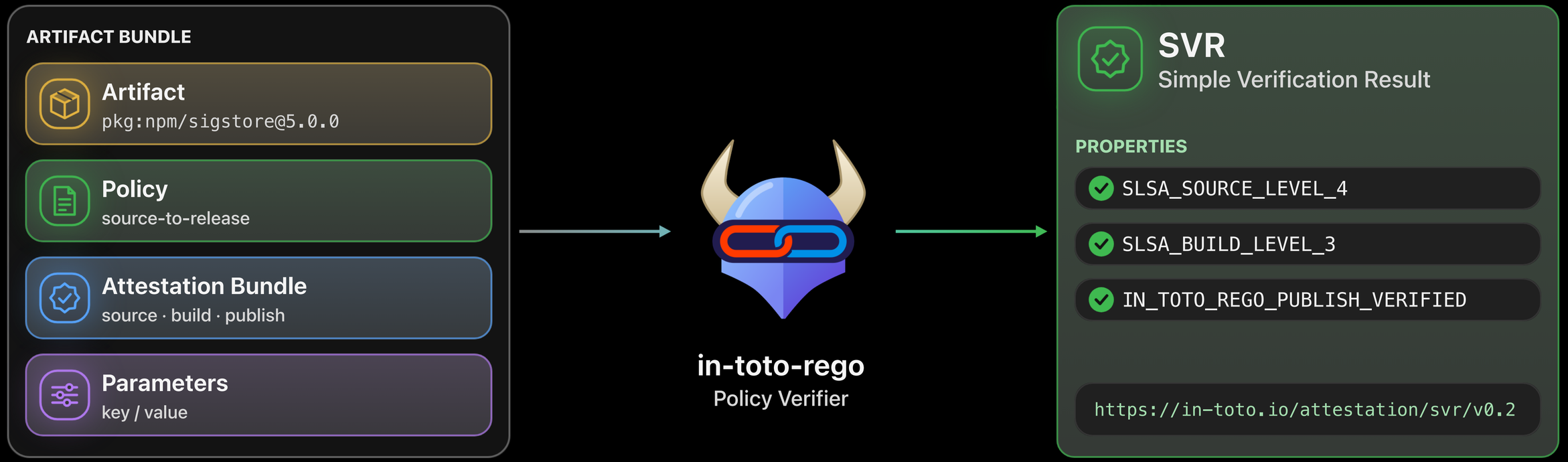
```
svr := predicates.svr(…, steps.publish.outputs, …)
```

Policies

Every link holds, so in-toto-rego mints the SVR



Tickets: Simple Verification Results (SVRs)



Tickets: Verifier and Properties

Which verifier ran, and what it attests about the artifact

SVR https://in-toto.io/attestation/svr/v0.2

```
{
  "subject": [{
    "name": "pkg:npm/sigstore@5.0.0",
    "digest": { "sha512": "849a897e81bf7b8a…b5559597fbd5f80e" }
  }],
  "predicate": {
    "properties": [
      "SLSA_SOURCE_LEVEL_4",
      "SLSA_BUILD_LEVEL_3",
      "IN_TOTO_REGO_PUBLISH_VERIFIED"
    ],
    "verifier": {
      "id": "https://github.com/open-policy-agent/in-toto-rego",
      "policies": [{ "name": "source-to-release", "digest": { "sha256": "ccb4bfb99531f399…7002fc741b182fb0" } }]
    }
  }
}
```

Artifacts

Properties

Verifier

Tickets: Artifact Bundles

What it records, so anyone can re-run the verification

SVR `https://in-toto.io/attestation/svr/v0.2`

```
{
  "subject": [{
    "name": "pkg:npm/sigstore@5.0.0",
    "digest": { "sha512": "849a897e81bf7b8a…b5559597fbd5f80e" },
    "annotations": {
      "attestations": {
        "source": { "payloadType": "application/vnd.in-toto+json", "payload": "eyJfdHlwZSI6…", … },
        "build": { "mediaType": "application/vnd.dev.sigstore.bundle.v0.3+json", … },
        "publish": { "mediaType": "application/vnd.dev.sigstore.bundle+json;version=0.2", … }
      },
      "parameters": { "package_name": "sigstore", "github_repository": "sigstore/sigstore-js", … }
    }
  }],
  "predicate": {
    "verifier": { "policies": [{ "name": "source-to-release", "digest": { "sha256": "ccb4bfb9…" } }] }, …
  }
}
```

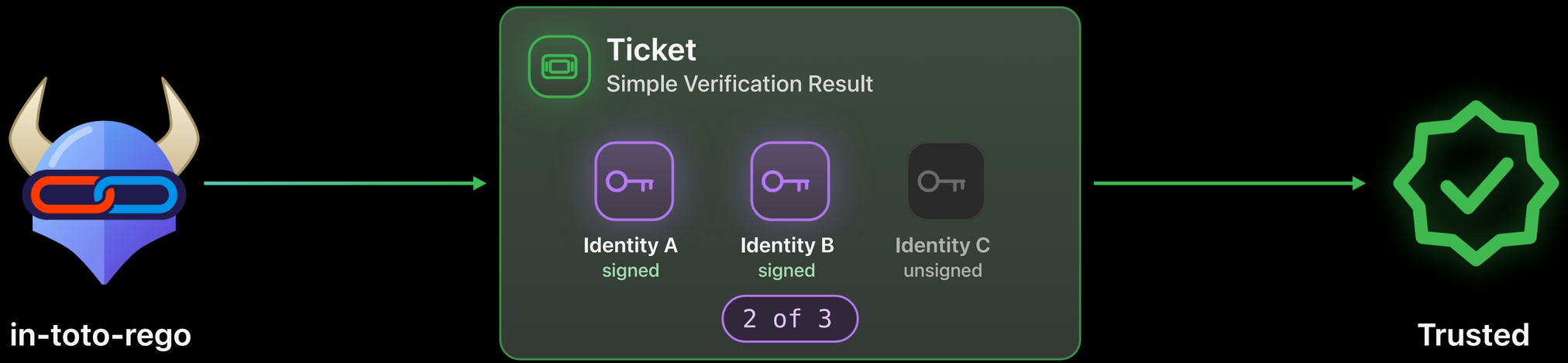
Artifact

Attestation Bundle

Parameters

Policy

Tickets: Threshold Signatures



in-toto-rego × OPA Gatekeeper

A Pod is admitted only if a Simple Verification Result for its image verifies.

Pod kubectl run — **API server** admission — **Gatekeeper** K8sSVR Rego — **svr-provider** DSSE · schema — **Registry** SVR referrer

1
Attested
`checkout:1.4.2` by digest
Built by the pipeline. Its SVR is signed by the verifier this cluster trusts.

2
No SVR
`checkout:1.4.3-hotfix` by digest
A hotfix pushed straight from a laptop. Nothing vouches for it.

3
Forged SVR
`checkout:1.4.2-evil` by digest
Someone copied 1.4.2's SVR and edited it to name this image instead.

4
Borrowed SVR
`checkout:1.5.0-rc1` by digest
Someone attached 1.4.2's genuine SVR, unchanged, to a different image.

5
Rogue verifier
`checkout:1.4.4` by digest
A well-formed SVR, signed by a verifier this cluster does not trust.

6
Tag, not digest
`checkout:1.4.2` by tag
The attested image again, but named by a tag, which can move after admission.

READY

$ press 1–6 to create a Pod in namespace demo

GATEKEEPER waiting for a Pod

SVR ATTACHED TO CHECKOUT:1.4.2

Claims | Envelope | Raw

This Pod's image	localhost:5001/shop/checkout@sha256:94b66950a21c…
SVR subject	localhost:5001/shop/checkout@sha256:94b66950a21c… THIS IMAGE
Policy	oci sha256:d6692b4c2a2f…
Properties	SLSA_BUILD_LEVEL_3 SLSA_SOURCE_LEVEL_4
Verifier	https://github.com/open-policy-agent/in-toto-rego
Created	2026-11-05T03:34:21Z

PODS IN NAMESPACE DEMO

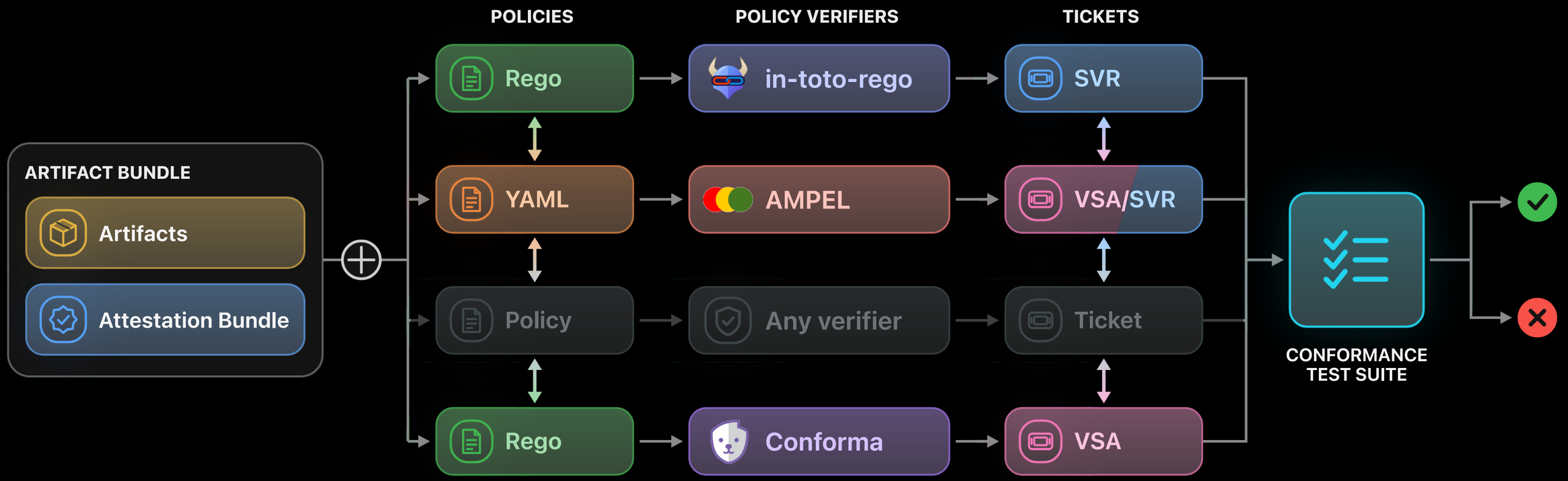
Reset

No Pods yet.

1–6 deploy · ← → inspect · C E J tabs · R reset

Every verdict on this page is Gatekeeper's. The inspector only shows what is attached.

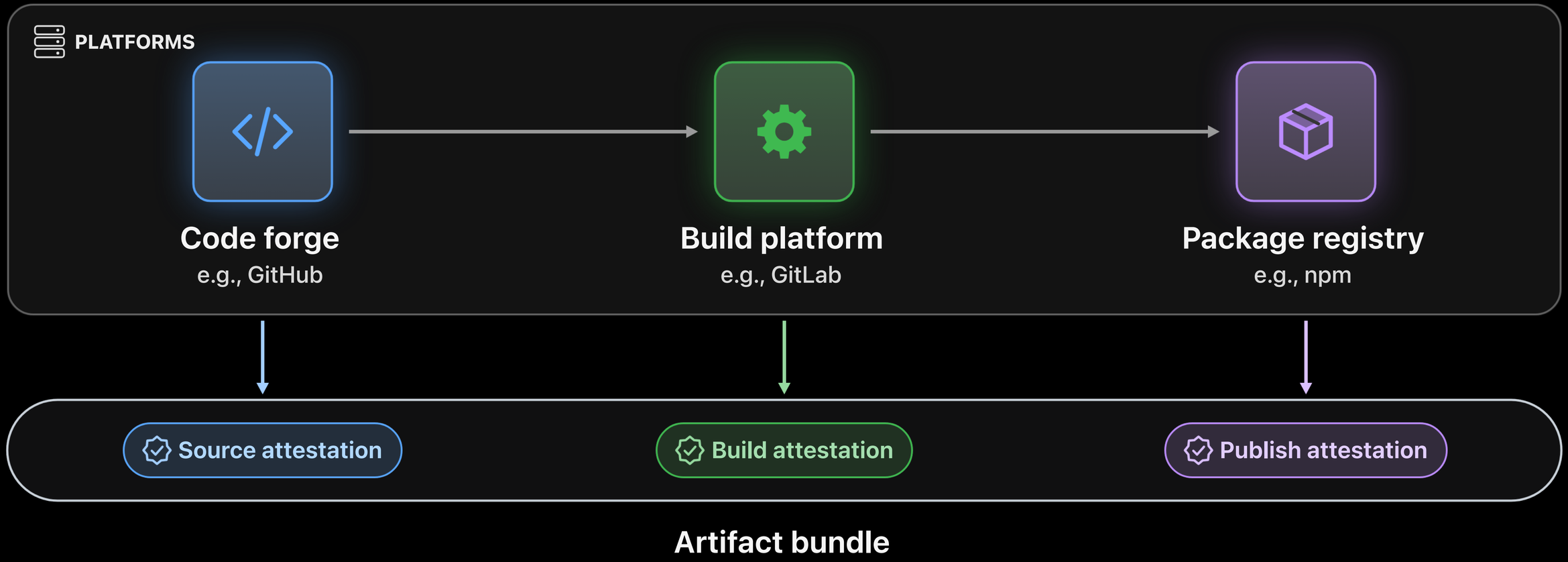
Policy Framework: Different Languages, One Ticket



Conclusions

How in-toto-rego can help secure open-source

Platforms produce attestations, which together make an artifact bundle



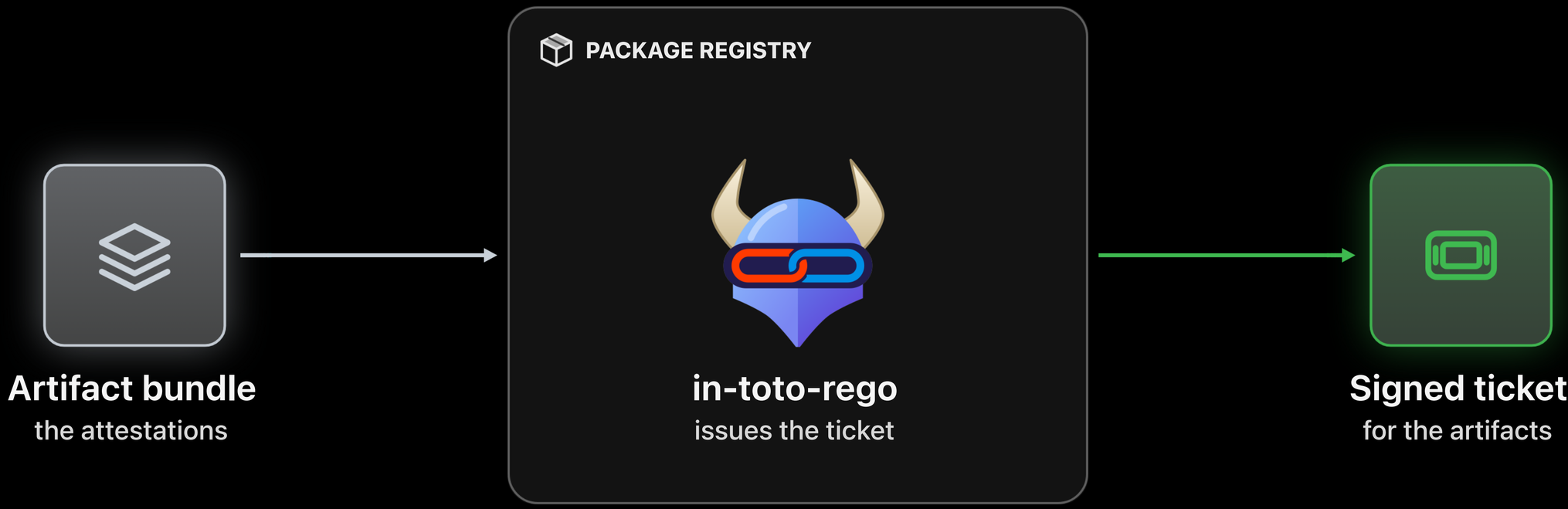
How in-toto-rego can help secure open-source

Maintainers opt in to these attestations for their own artifacts



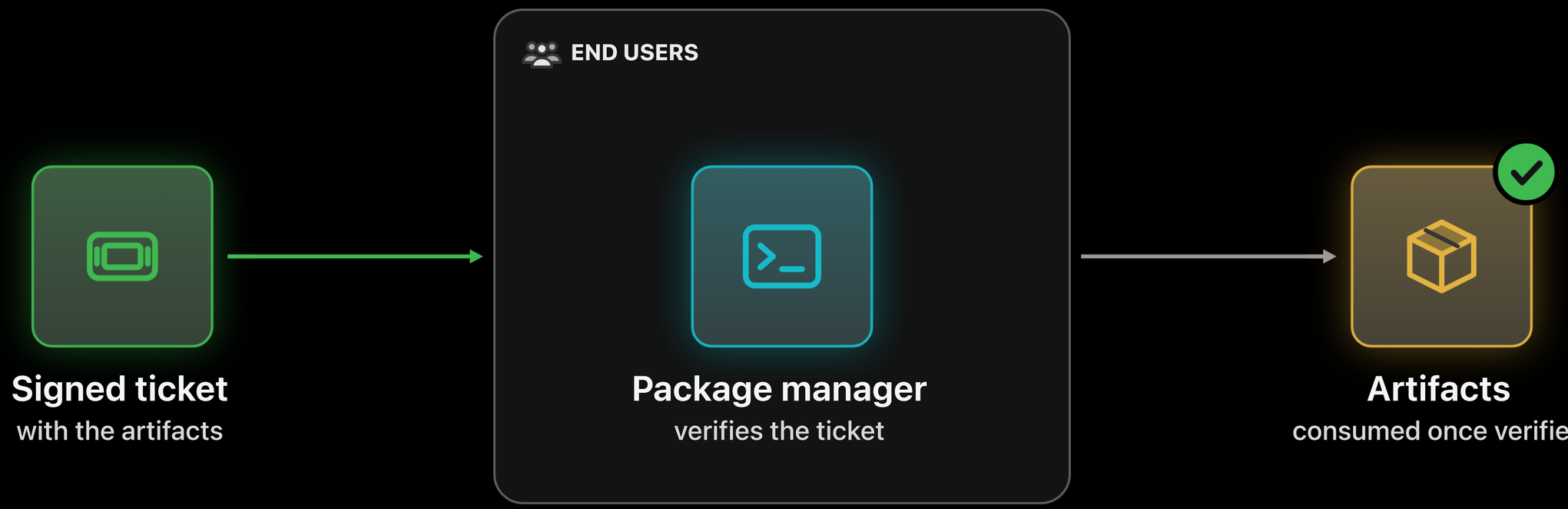
How in-toto-rego can help secure open-source

Package registries use in-toto-rego to issue tickets for these artifacts



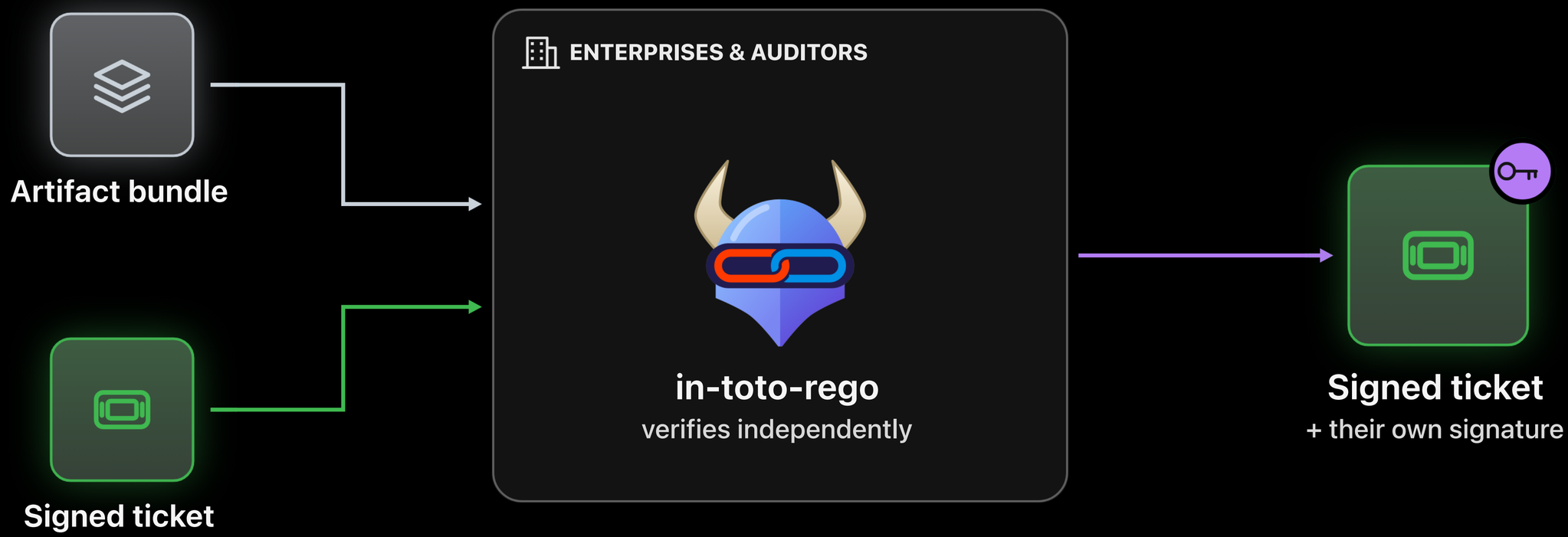
How in-toto-rego can help secure open-source

End users' package managers verify tickets transparently before consuming artifacts



How in-toto-rego can help secure open-source

Enterprises and auditors verify tickets independently and add their own signatures



Thank You



in-toto
attestations and policies



Open Policy Agent
policy engine



Sigstore
transparency



SLSA
supply-chain levels



securesystemslib
keys and signing



TUF
secure updates

Apple Services Engineering Security Is Hiring

Security Platform Automation Engineer

200685550-3337

Senior Software Engineer - Security

200688648-3337

Senior Software Engineer - Security Automation

200690613-3337

Senior Security Software Engineer, Software Supply Chain Security

200679771-3337





Slack

#opa-in-toto-rego
slack.openpolicyagent.org





GitHub

github.com/open-policy-agent/
in-toto-rego



TM and © 2026 Apple Inc. All rights reserved.