cdisc®

# 2026 Europe Interchange

THE FUTURE IS CONNECTED: STANDARDS AND AI POWERING DIGITAL TRANSFORMATION


# *From Data Traceability to Evidence Traceability: A Provenance Layer for CDISC Standards in Metadata-Driven SCEs*

Massimo Raineri, Arithmos
Darius Tetsa, UCB

# Agenda

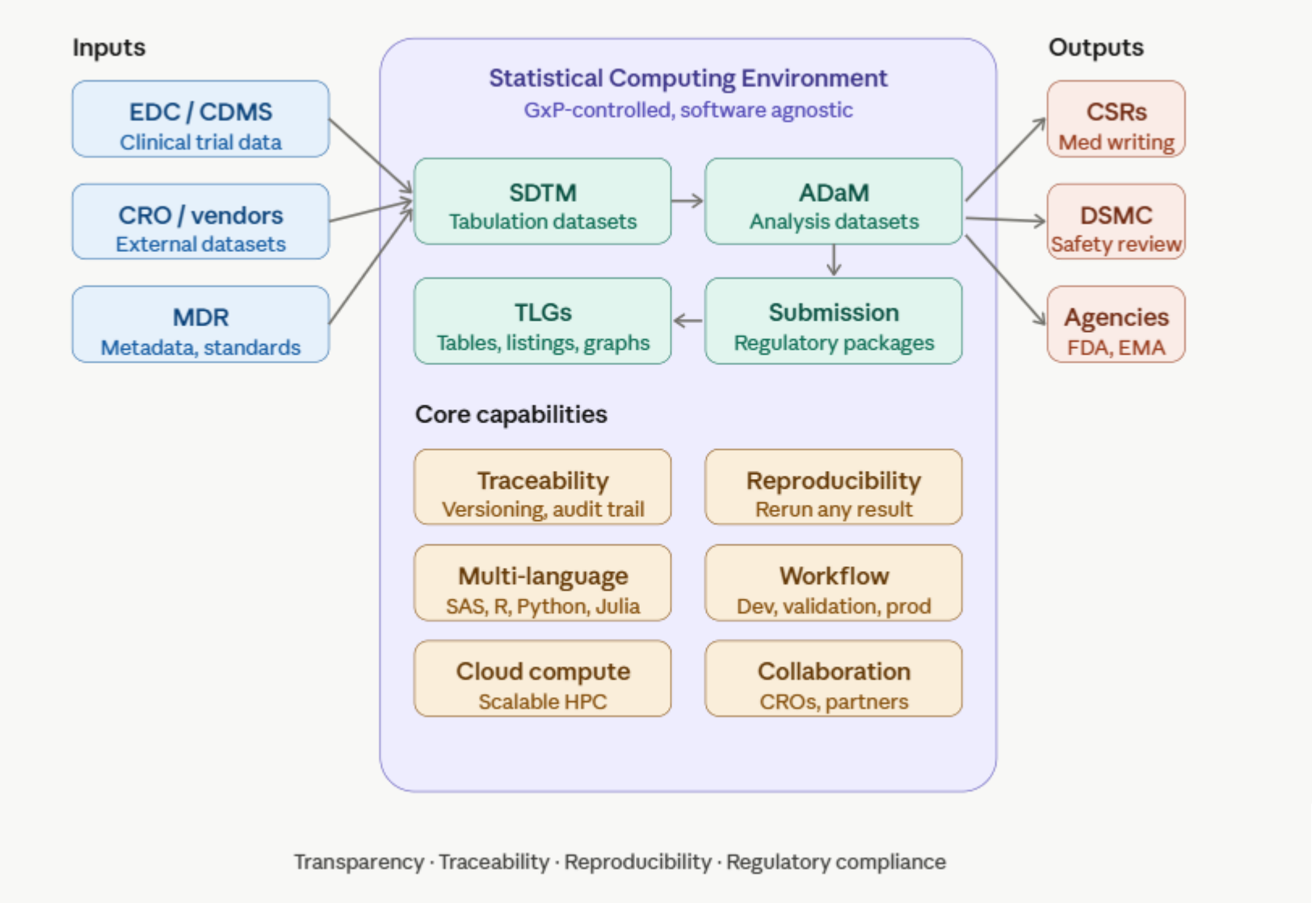
1. The context and the need
2. ARS: The Evolving Opportunities
3. ARS+ Provenance Bundle
4. Benefits and Next Steps

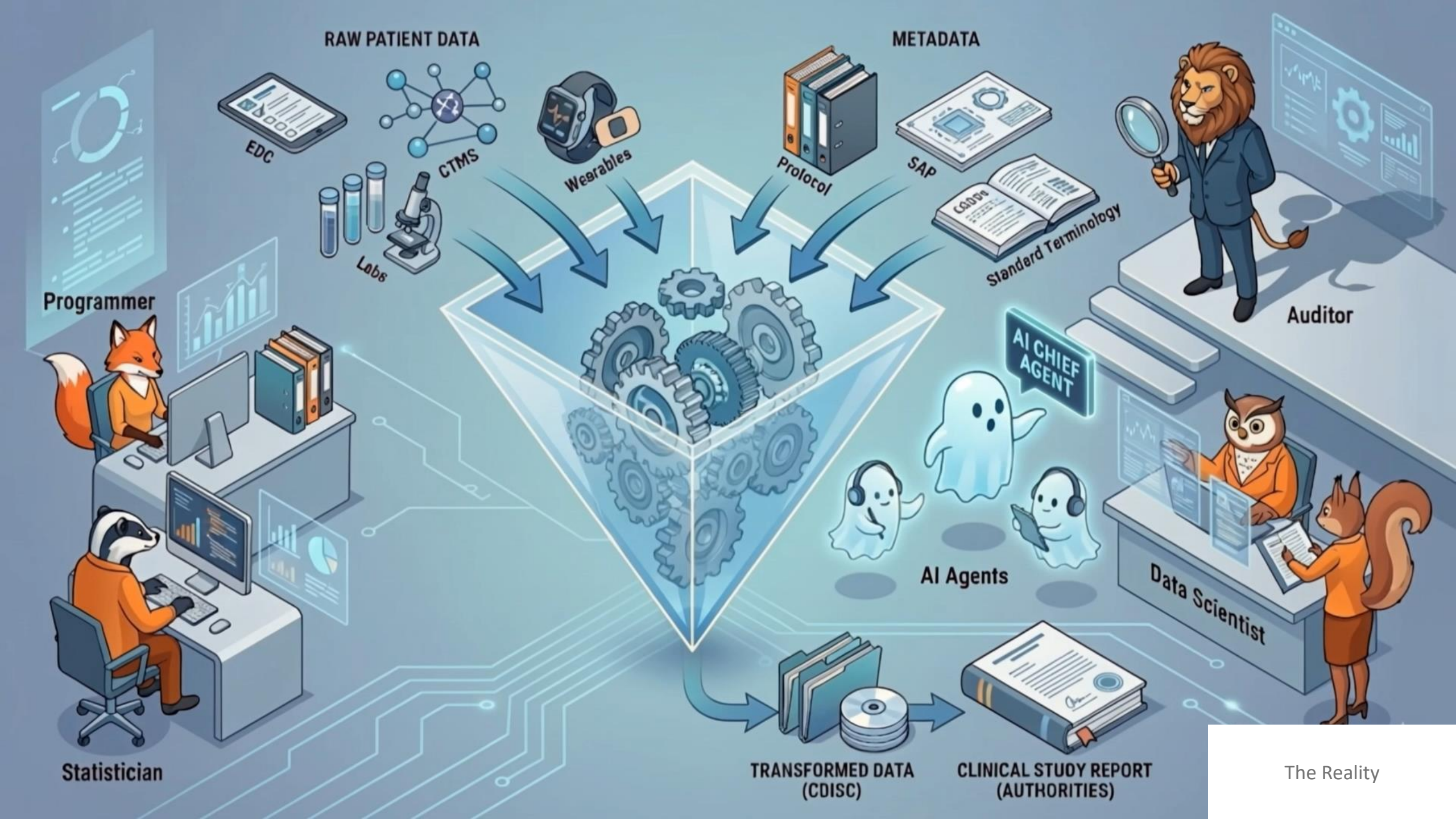
# Speakers

**Massimo Raineri**
Senior Consultant
Arithmos

**Darius Tetsa**
Tools & Standards Principal Specialist
UCB

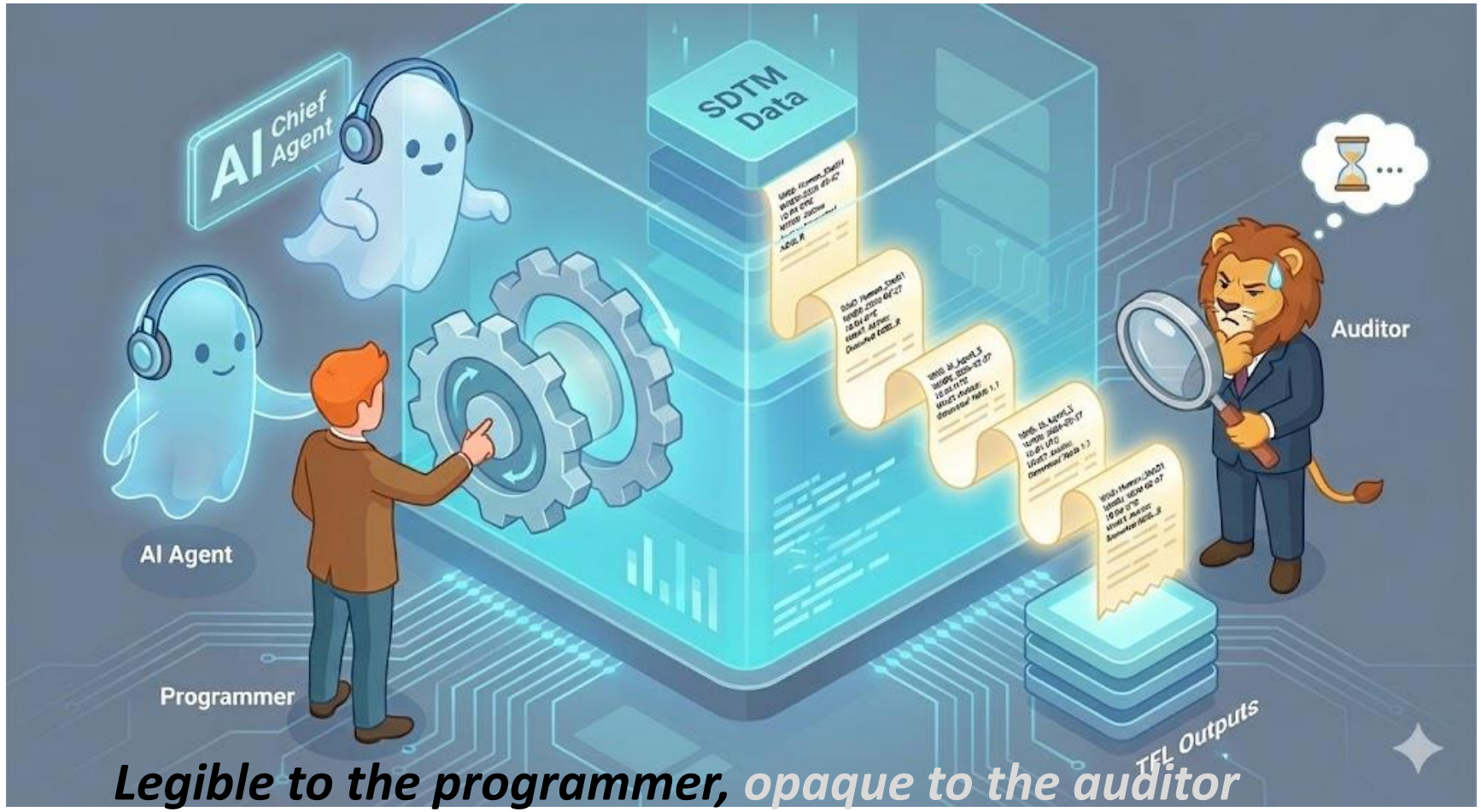
# Statistical Computing Environment




RAW PATIENT DATA
METADATA
EDC
CTMS
Wearables
Labs
Protocol
SAP
Standard Terminology
Programmer
Auditor
AI CHIEF AGENT
AI Agents
Data Scientist
Statistician
TRANSFORMED DATA (CDISC)
CLINICAL STUDY REPORT (AUTHORITIES)

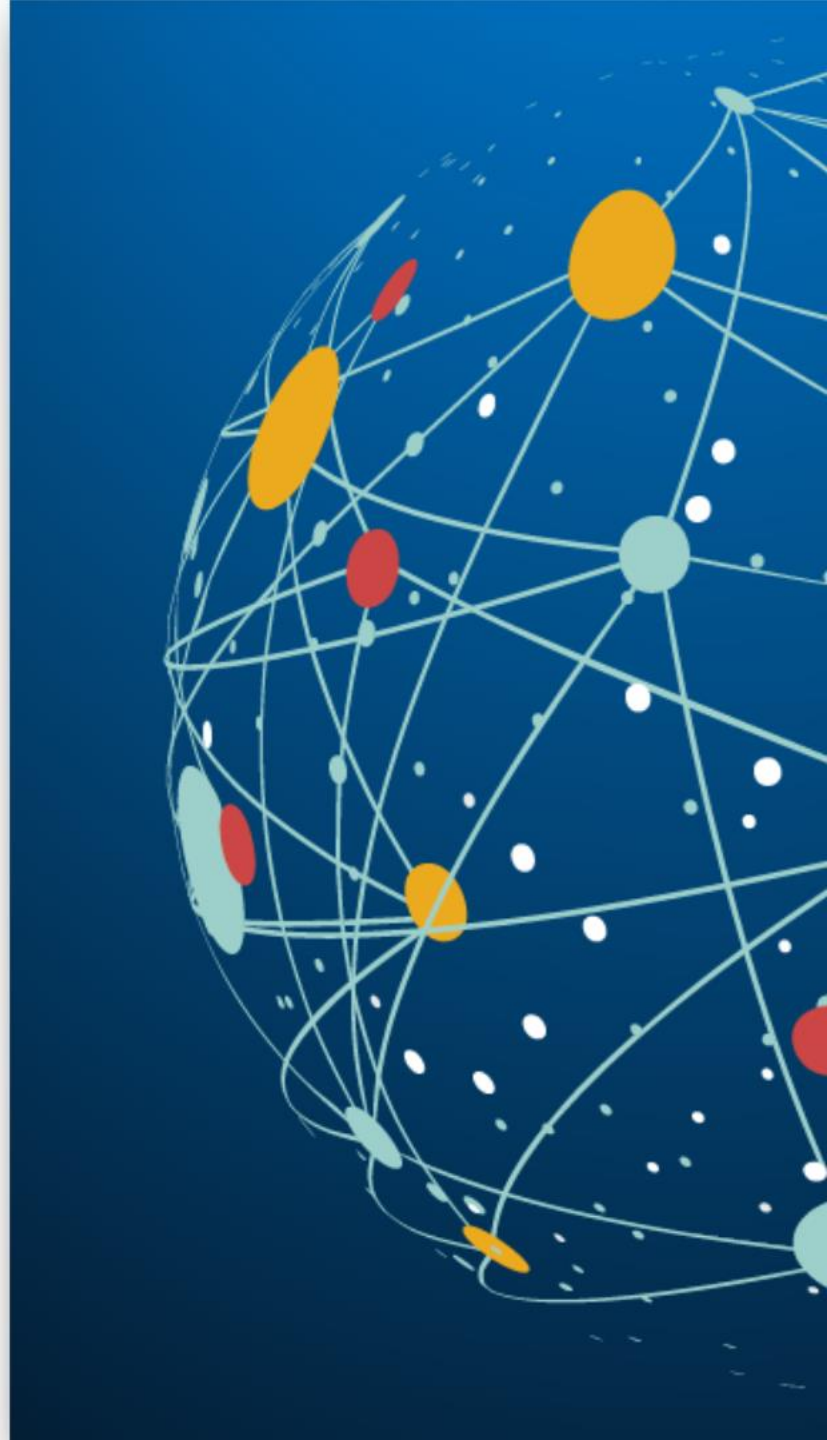
The Reality

# Today's audit trail: a stack of receipts that trace data. But do they trace evidence?



# Rethinking the Audit Trail

| Event-Centric vs State-Centric | |
|---|---|
| **Event-Centric *(today)*** | **State-Centric *(needed)*** |
| Logs *what happened* | Captures *the full state* |
| Answers: *"what happened?"* | Answers: *"what produced this?"* |
| Fragmented across systems | Unified snapshot, machine-readable |
| Sufficient for process audit | Required for result reconstruction |



# The Audit Trail Limits


1. **Traceability Limit:** we can trace the *data* (SDTM → ADaM → CSR) but not the *how.*

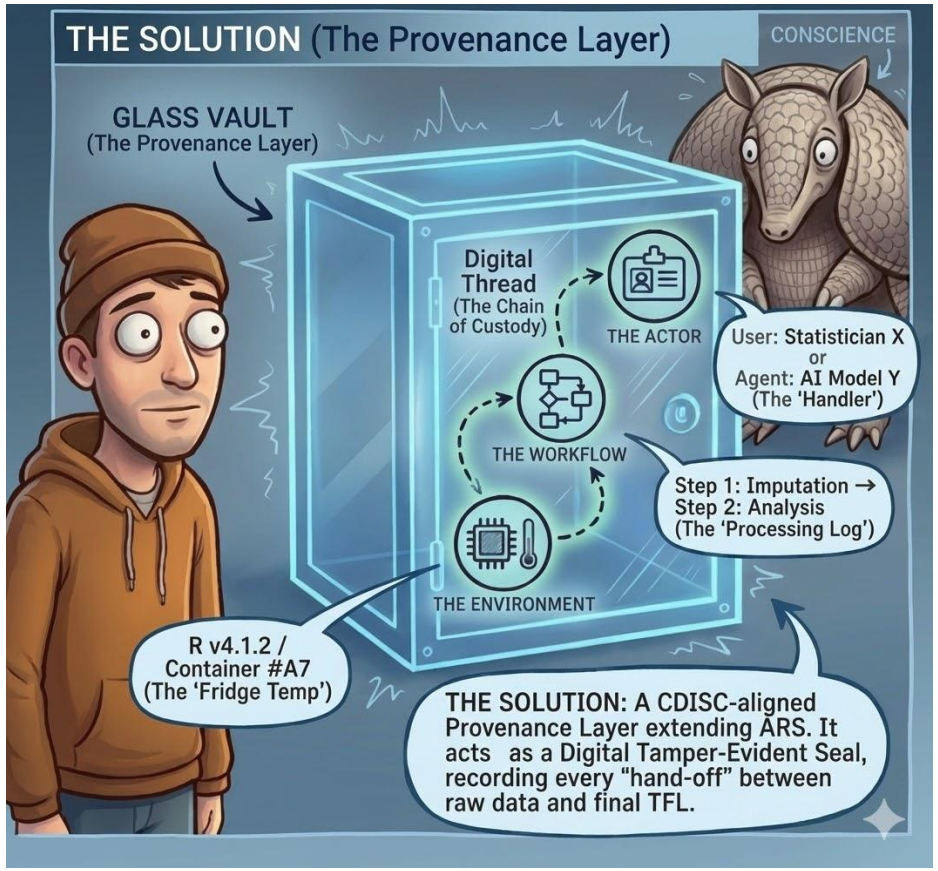
2. **Polyglot Challenge:** multi-language execution fragments the audit trail.

3. **Black Box Automation:** auditors can see inputs and outputs, but not the transformation.

# Forensic Analogy: The Chain of Custody

| From the Courtroom to the SCE | | |
|---|---|---|
| **The find:** DNA at the crime scene | **The trial:** prove who handled it, when, where, how | **The verdict:** no context = inadmissible |

# The Solution: A CDISC-Aligned Provenance Layer




**ACTOR:** *Who ran it*

**WORKFLOW:** *How it was produced*

**ENVIRONMENT:** *Where it ran*

**Every hand-off between raw data and final TLF is recorded, linked to its context, machine-readable.**

# ARS: The Evolving Opportunities

# ARS (Analysis Result Standard)

Standardized Results

Traceable Analysis

Data Transparency

Consistent, Reproducible, and Transparent Analysis Results

# Evolving Opportunities for ARS

## ARS Today: A Strong Foundation for the Next Phase of Provenance

**Broader Provenance Coverage**

ARS offers a strong provenance foundation that can be extended to support end-to-end traceability.

**AI-Ready Transparency**

As AI adoption grows, ARS can evolve to better document AI reasoning and inference.

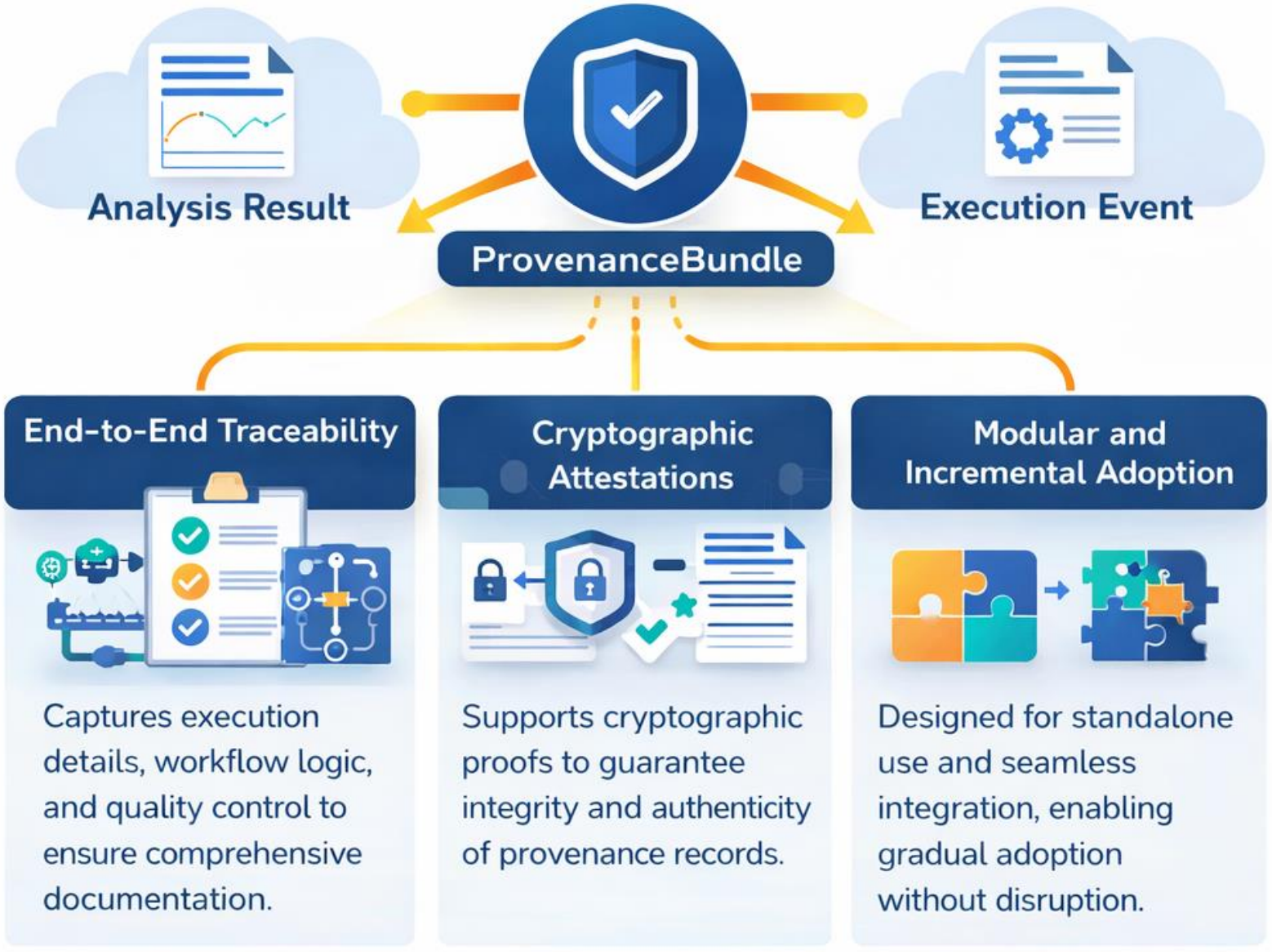
**Enhanced Audit & Compliance**

Richer environment and workflow metadata will enhance audit readiness, reproducibility, and regulatory alignment.

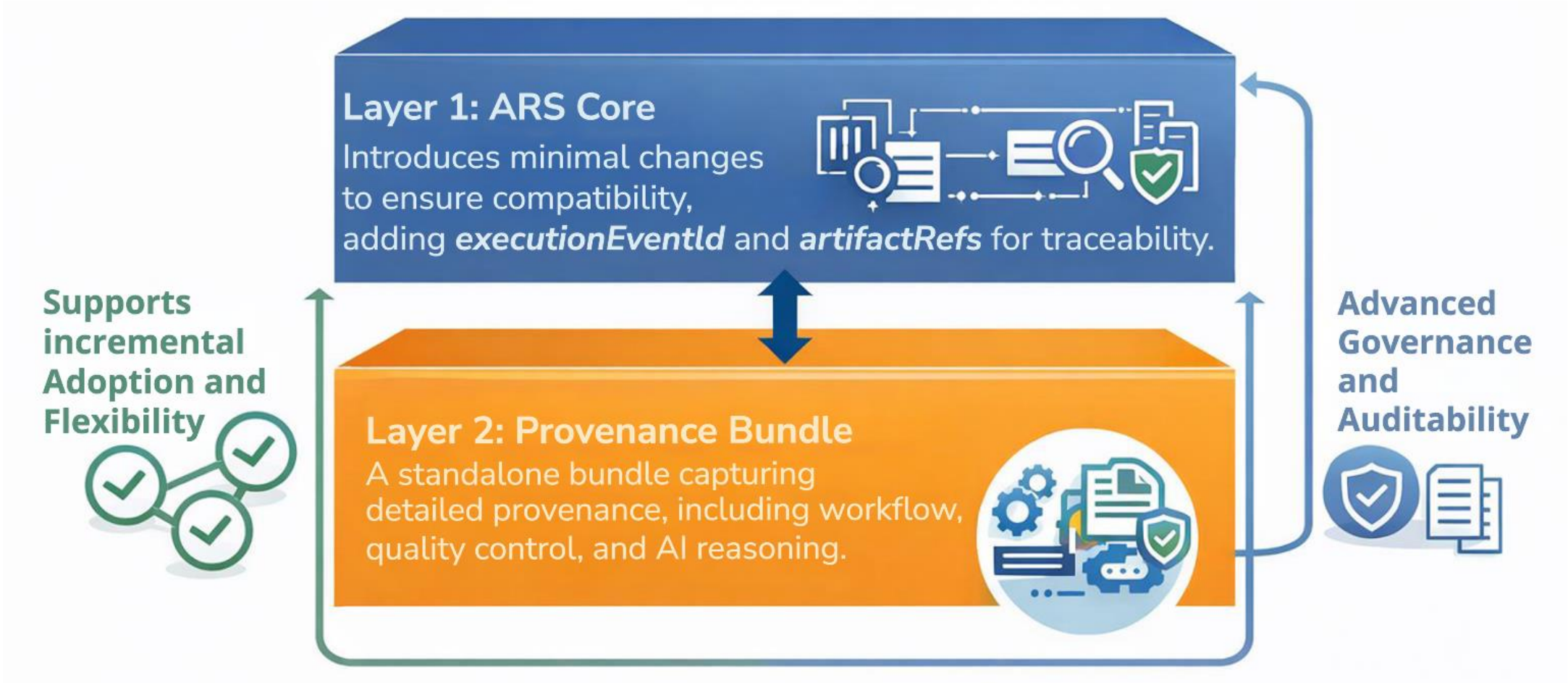
# ARS+ Provenance Bundle:

Extending ARS with End-to-End, Machine-Readable Provenance

# Extending ARS with a Provenance Layer




# Two-Layer Architecture for Provenance




Supports auditability, reproducibility, and regulatory compliance with detailed execution and review documentation.

# Layer 2:
# ARS+ Provenance Bundle Components

- Workflow & Dependency
- Quality & Governance
- AI & Agentic Context

# ExecutionEvent: *Workflow & Dependency*

## Capturing What & When

Used to document the precise circumstances of each execution

- **Core attributes**
  - executionEventId (UUID)
  - startedAt, endedAt
  - triggerType (manual / scheduled / dependency-change / release)
  - executor (human/agent/service account)
  - environmentRef → ***ExecutionEnvironment***
  - workflowRunRef → ***WorkflowRun***
  - materials → list of produced artifacts + hashes
  - attestations → list of cryptographic attestations (optional but recommended)


```
"executionEvents": [
  {
    "executionEventId": "EXE-9f2c7e3a",
    "startedAt": "2026-02-24T08:11:02Z",
    "endedAt": "2026-02-24T08:14:55Z",
    "triggerType": "dependency-change",
    "executor": { "type": "service", "id": "svc-sce-runner" },
    "environmentRef": "ENV-validated-r42",
    "workflowRunRef": "WR-1a22b9",
    "materials": [
      { "uri": "s3://.../t1411.rtf", "sha256": "8f0c...aa1e" },
      { "uri": "s3://.../t1411.log", "sha256": "11bd...c02a" }
    ],
    "attestations": [
      { "type": "build-attestation", "ref": "WORM://attest/EXE-9f2c7e3a" }
    ]
  }
],
```


Example of ExecitionEvent in JSON

# ExecutionEnvironment: *Workflow & Dependency*

## Documenting Where

Used to capture information about the execution environment. This information is reusable across multiple executions and is critical for ensuring reproducibility and auditability.

- **Core attributes**
  - runtimeStack (language + interpreter versions)
  - packages (name/version/source/hash)
  - OS (distribution, kernel)
  - container (image name, digest, build provenance)
  - infrastructure (cloud region/cluster, VM type, CPU, memory)

```
"executionEnvironments": [
  {
    "environmentId": "ENV-validated-r42",
    "runtimeStack": [
      { "language": "R", "version": "4.2.3" }
    ],
    "packages": [
      { "name": "dplyr", "version": "1.1.4", "source": "CRAN", "sha256": "..." },
      { "name": "ggplot2", "version": "3.5.0", "source": "CRAN", "sha256": "..." }
    ],
    "os": { "name": "Ubuntu", "version": "22.04", "kernel": "5.15.0-91" },
    "container": {
      "image": "registry/org/sce:r42-validated",
      "digest": "sha256:3b1d...f9c2",
      "buildProvenanceRef": "WORM://build/sha256:3b1d...f9c2"
    },
    "infrastructure": {
      "orchestrator": "Kubernetes",
      "clusterId": "prod-sce-01",
      "nodeType": "Standard_D8s_v5",
      "region": "uksouth"
    }
  }
],
```

Example of ExecitionEnvironment in JSON

# WorkflowDefinition: *Workflow & Dependency*

A machine-readable description of the pipeline logic

- **Core attributes**
  - workflowId,
  - name,
  - version
  - steps[] each step has:
    - stepId,
    - type (data-load, derive, impute, model-fit, render, QC-check, publish)
    - commandRef (script/macro/function)
    - inputs[], outputs[]
    - parameters (immutable set used)

```
"workflowDefinitions": [
  {
    "workflowId": "WF-tlf-prod",
    "version": "2.7.0",
    "steps": [
      { "stepId": "S1", "type": "data-load", "commandRef": "git://repo#scripts/load.R" },
      { "stepId": "S2", "type": "derive", "commandRef": "git://repo#scripts/adam_derivations.R" },
      { "stepId": "S3", "type": "analysis", "commandRef": "git://repo#scripts/primary_endpoint.R" },
      { "stepId": "S4", "type": "render", "commandRef": "git://repo#scripts/render_tlf.R" }
    ]
  }
],
```

Example of WorkflowDefinition in JSON

# WorkflowRun: *Workflow & Dependency*

A realized execution of a workflow definition.

- **Core attributes**
  - workflowRunId
  - workflowDefinitionRef → ***WorkflowDefinition***
  - resolvedDependencies (snapshot of dependency graph at runtime)
  - executionEvents[] (one per step, or one per run; you can support both patterns)


```
"workflowRuns": [
  {
    "workflowRunId": "WR-1a22b9",
    "workflowDefinitionRef": "WF-tlf-prod@2.7.0",
    "resolvedDependencies": "DEP-77c9",
    "executionEvents": ["EXE-9f2c7e3a"]
  }
],
```


Example of WorkflowRun in JSON

**Note:** Workflow Definition is separate from Run.

***Workflow run** = **Workflow definition** + **dependencies** + **Execution Event**.*

# DependencyManifest: *Workflow & Dependency*

A canonical listing of all upstream dependencies with versions and hashes. This enables dependency checks, automatic re-run on change and deterministic rebuilds.

- **Core attributes**
  - dependencyManifestId
  - policy
  - inputs
  - code
  - config

```
"dependencyManifests": [
  {
    "dependencyManifestId": "DEP-77c9",
    "policy": { "rerunOnHashChange": true },
    "inputs": [
      { "type": "dataset", "name": "ADSL", "uri": "s3://.../ADSL.xpt", "sha256": "..." },
      { "type": "dataset", "name": "ADAE", "uri": "s3://.../ADAE.xpt", "sha256": "..." }
    ],
    "code": [
      { "type": "repo", "uri": "git://repo", "commit": "a13f9d2", "dirty": false },
      { "type": "macro", "uri": "s3://.../macros.sas", "sha256": "..." }
    ],
    "config": [
      { "uri": "s3://.../params.yml", "sha256": "..." }
    ]
  }
],
```

Example of DependencyManifest in JSON

**Note:** Dependency manifest is the single source of re-run truth. Any mismatch triggers a new ExecutionEvent and new output hashes.

# QCAttestation: Quality & Governance

Captures the QC method applied and its outcome.

- **Core attributes**
  - qcId
  - qcMethod (double programming / code review / unit tests / golden dataset / statistical review)
  - qcScope (result-level / workflow-run-level / step-level)
  - qcOutcome (pass/fail/conditional)
  - evidenceRefs[] (logs, diff reports, unit test output, review record)
  - reviewers[] → ***ReviewerAction***


Example of QCAttestation in JSON

# ReviewerAction: Quality & Governance

A human sign-off with traceable identity.

- **Core attributes**
  - reviewerId (mapped to org identity)
  - actionType (reviewed / approved / rejected)
  - timestamp
  - digitalSignature (public key or e-signature reference)
  - comment


Example of ReviewerAction in JSON

# AgentSession: AI & Agentic Context

Represents an AI agent's "Ask → Plan → Retrieve → Execute" session that contributed to a result.

- **Core attributes**
  - agentSessionId
  - agentIdentity (agent name/version, vendor, policy)
  - modelIdentity (model family, version, endpoint, deployment id)
  - sessionStart, sessionEnd
  - governance (policy version, allowed tools, safety filters)


```
"agentSessions": [
  {
    "agentSessionId": "AS-0091",
    "agentIdentity": { "name": "SCE-Agent", "version": "1.4.0" },
    "modelIdentity": { "provider": "AzureOpenAI", "deploymentId": "gpt-x-prod-03" },
    "sessionStart": "2026-02-24T08:05:00Z",
    "sessionEnd": "2026-02-24T08:10:21Z",
    "governance": { "policyVersion": "AI-GOV-7.2", "toolsAllowed": ["retrieval","code-exec"] }
  }
],
```


Example of AgentSession in JSON

# AgentTrace: AI & Agentic Context

Stores *structured* reasoning steps without needing to expose sensitive internal “thoughts”.

- **Core attributes**
  - userGoal (high-level request)
  - planSteps[] (explicit steps the agent intended to do)
  - toolCalls[] (retrieval queries, code execution, dataset access)
  - prompts[] (the exact prompts sent to the model)
  - responses[] (model outputs used)
  - redactions (what was removed and why, for privacy/IP)


```
"agentTraces": [
  {
    "agentSessionId": "AS-0091",
    "userGoal": "Generate primary endpoint TLF with validated pipeline",
    "planSteps": [
      "Retrieve latest ADaM snapshot",
      "Run validated workflow WF-tlf-prod",
      "Collect logs and compute hashes"
    ],
    "toolCalls": [
      { "tool": "retrieval", "query": "latest ADSL ADAE snapshot", "timestamp": "2026-02-24T08:05:14Z" },
      { "tool": "execute", "target": "WF-tlf-prod@2.7.0", "timestamp": "2026-02-24T08:10:10Z" }
    ],
    "prompts": [
      { "role": "system", "contentHash": "sha256:...", "timestamp": "2026-02-24T08:05:01Z" },
      { "role": "user", "contentHash": "sha256:...", "timestamp": "2026-02-24T08:05:02Z" }
    ],
    "responses": [
      { "contentHash": "sha256:...", "usedInDecision": true }
    ],
    "redactions": [
      { "field": "prompt", "reason": "PHI/PII minimization" }
    ]
  }
],
```


Example of AgentTrace in JSON

**Note**: This captures “reasoning context” in an auditable way without requiring raw chain-of-thought.

# InferenceEvent : AI & Agentic Context

## Captures non-deterministic inference details

- **Core attributes**
  - inferenceEventId
  - agentTraceId
  - timestamp
  - modelIdentity (exact deployment)
  - Parameters (temperature/top_p/seed)
  - inputHash (hash of prompt + retrieved context)
  - outputHash (hash of output)
  - attestationRef (immutable store/WORM log reference)

```
"inferenceEvents": [
  {
    "inferenceEventId": "INF-a82b",
    "agentSessionId": "AS-0091",
    "timestamp": "2026-02-24T08:05:09Z",
    "modelIdentity": { "deploymentId": "gpt-x-prod-03" },
    "parameters": { "temperature": 0.2, "top_p": 0.95 },
    "inputHash": "sha256:...",
    "outputHash": "sha256:...",
    "attestationRef": "WORM://ai/INF-a82b"
  }
]
```

Example of InferenceEvent in JSON

# Benefits and Next Steps for Implementation

ARS+ Provenance Bundle

# Benefits and Next Steps

ARS+ delivers **end-to-end, machine-readable** traceability of programming executions by linking results to their **full technical, quality, and AI context**, enabling **reproducible analyses, auditable compliance, and future-ready governance.**

| Agree on Core Extensions | Pilot Implementation | Align with Policies | Engage Stakeholders |
|---|---|---|---|
| Agree on minimal **ARS** core extensions to ensure compatibility and ease of adoption. | Pilot the ***ARS+ ProvenanceBundle*** in controlled environment. | Ensure the ***ARS+ ProvenanceBundle*** aligns with quality control, validation, and AI governance policies. | Involve **IT, QA, regulatory, and AI governance teams** to validate approach and address challenges. |

# Thank You!

# Questions?