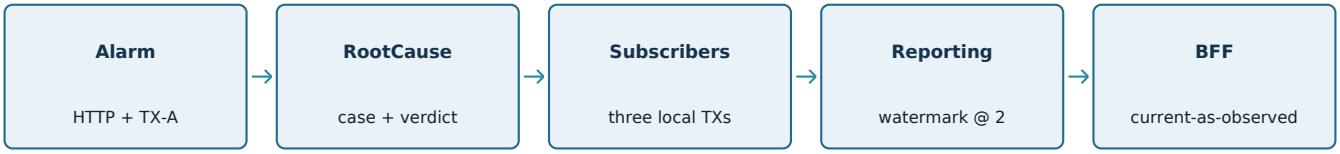
PART VI - THE FIRST END-TO-END DISTRIBUTED SLICE

36. The Complete Slice and the Optional RL Advisory Branch

The exact public facts and owner-local commits from Chapters 32-35 now become one production-path scenario with preserved correlation, direct causation and reviewable recovery evidence. RL Advisory remains an optional non-command observer.

CHAPTER DECISION
The core scenario enters through the secured Alarm HTTP edge, proves TX-A, the Alarm relay, TX-RC-OPEN, CaseOpened publication, TX-RC-VERDICT, verdict fan-out, TX-AUD, TX-COMP, TX-REP and the freshness-bearing BFF response. Completion is an evidence vector, never a global transaction. The manuscript defines the runnable proof but keeps runtime status PENDING until real artifacts exist.

PROOF PATH 36-A - COMPLETE CORE SLICE



GUARANTEE LEDGER 36-A

Question	Answer
What is preserved?	three frozen bodies, one correlation and direct-hop causation
What is independent?	every service database, retry path and failure outcome
What is optional?	RL Advisory queue, subscriber and module store
What is still pending?	repository execution and attached runtime evidence

PROOF BOUNDARY

A complete slice is not one distributed transaction

The scenario is complete when every owner can prove its own durable effect and the public facts connect those effects. It is not complete because one coordinator held locks across five databases or because one test method returned without throwing.

DECISION LEDGER 36-2

Owner boundary	Durable claim	Cross-owner claim forbidden
AlarmManagementDb	threshold decision + pending Flood fact	RootCause case exists
RootCauseDb	case + verdict + two pending facts	Audit/Compliance effects exist
AuditDb / ComplianceDb	independent observed consequences	Reporting is current
ReportingDb	delayed projection + watermark	source-authoritative truth

COMPLETION RULE
The proof is a conjunction of local commits, exact message identities, broker settlement evidence and an honest final observation. No row participates in a global ACID boundary.

GOLDEN CORPUS

Three public bodies remain byte-for-byte fixed

Chapter 36 consumes the artifacts frozen by Chapters 32-35. The runner does not regenerate convenient equivalents; it compares the captured bodies with the canonical corpus and fails if one coordinate, byte length or digest drifts.

EXECUTABLE ASSET 36-0 - SHARED GOLDEN IDENTIFIERS

```
internal static class Golden36
{
    public const string CORRELATION_ID = "request-am-20482";
    public const string FLOOD_MESSAGE_ID =
        "019bc4d4-5dc1-7b8d-9d48-542d25315356";
    public const string CASE_ID =
        "019bc4d4-6dfe-7bcb-9de8-3af71979285a";
    public const string CASE_OPENED_MESSAGE_ID =
        "019bc4d4-6ea0-7d14-a57a-3e8643c55f13";
    public const string VERDICT_ID =
        "019bc4d4-81c0-7ea1-82d5-4f9b8719d0a1";
    public const string VERDICT_MESSAGE_ID =
        "019bc4d4-82a4-77f2-ae06-09b172aa59db";
}
```

DECISION LEDGER 36-3

Fact	MessageId / length	Final body SHA-256
AlarmFloodDetected.v1	019bc4d4-5dc1-7b8d-9d48-542d25315356 / 1003	7cf98319c20a5915d5710bbcbb5878fe51cdae89a5b68081c135fd0e962c276a
RootCauseCaseOpened.v1	019bc4d4-6ea0-7d14-a57a-3e8643c55f13 / 876	454f51cf4b0250b9c855346a6648a7d73530f19687ee3b52cda2edf9d0339c5b
RootCauseVerdictIssued.v1	019bc4d4-82a4-77f2-ae06-09b172aa59db / 1157	192e5e73c5a35d438e2ee9edb0a9e7044293c29bf321ba1abe34f191d8b44c00

BYTE AUTHORITY
Pretty-printed JSON is documentation. The one-line RFC 8785-compatible canonical UTF-8 bodies and their stored digests are the executable contract corpus.

LINEAGE MODEL

Correlation stays constant; causation moves one hop

Correlation groups the business journey. Causation identifies the immediate business stimulus. Reusing the first command as every event's cause erases the real path; copying a trace span into CausationId confuses observability with business meaning.

DECISION LEDGER 36-4

Artifact	CorrelationId	Immediate CausationId
AlarmFloodDetected	request-am-20482	decision-am-9184
RootCauseCaseOpened	request-am-20482	019bc4d4-5dc1-7b8d-9d48-542d25315356
RootCauseVerdictIssued	request-am-20482	decision-rc-7184
subscriber lineage	request-am-20482	019bc4d4-82a4-77f2-ae06-09b172aa59db

DIRECT-HOP RULE
The scenario validator accepts one correlation value across the journey and demands a causation value appropriate to each direct edge.

RUNTIME TOPOLOGY

Core delivery and optional advice never share availability

The core profile boots AlarmManagement, RootCause, Audit, Compliance, Reporting and the BFF with owner-specific credentials and stores. The advisory profile may add one independent queue and subscriber. RootCause publication success never depends on that optional queue.

DECISION LEDGER 36-5

Route	Required recipients	Optional recipient
alarm-management.alarm-flood-detected.v1	root-cause	none
root-cause.root-cause-case-opened.v1	reporting	none
root-cause.root-cause-verdict-issued.v1	audit, compliance, reporting	rl-advisory when enabled

TOPOLOGY INVARIANT
The required route gate is evaluated against the core recipients only. Optional queue absence cannot block a producer outbox row from reaching Published.

DEPLOYMENT PROFILES

The optional branch is explicit configuration, not ambient discovery

Two manifests make the difference reviewable. Core is the default. CoreWithAdvisory adds a queue, binding, module store and worker registration; it does not replace any core registration or broaden any database principal.

EXECUTABLE ASSET 36-A - EXPLICIT SLICE PROFILE

```
public enum SliceProfile { Core, CoreWithAdvisory }

public static void AddFullSlice(
    this IServiceCollection services, SliceProfile profile)
{
    services.AddAlarmManagementPath();
    services.AddRootCausePath();
    services.AddAuditComplianceReporting();

    if (profile is SliceProfile.CoreWithAdvisory)
        services.AddRlAdvisorySubscriber();
}
```

FAIL CLOSED
An unknown profile, missing required queue or unexpected advisory registration fails startup evidence. The system never guesses from a hostname or environment label.

SCENARIO HARNESS

One runner observes production paths without owning them

The runner sequences test actions and bounded observations. It never writes business tables, publishes owner facts, acknowledges deliveries or retries production commands on behalf of a service. Those actions remain inside the production composition roots.

EXECUTABLE ASSET 36-B - EVIDENCE RUNNER BOUNDARY

```
public sealed class FullSliceScenario(
    SliceEnvironment environment,
    AlarmApiDriver alarmApi,
    RootCauseApiDriver rootCauseApi,
    SliceObserver observer,
    EvidenceSink evidence)
{
    public async Task<ScenarioResult> RunAsync(
        SliceProfile profile, CancellationToken ct)
    {
        await environment.StartAsync(profile, ct);
        await alarmApi.SubmitGoldenObservationsAsync(ct);
        await observer.AwaitCaseOpenedAsync(ct);
        await rootCauseApi.IssueGoldenVerdictAsync(ct);
        return await observer.AwaitConvergenceAsync(profile, ct);
    }
}
```

NOT A PROCESS MANAGER
This type exists in the test/evidence assembly. It has no production endpoint, database, queue, retry policy or deployment. Chapter 37 introduces the durable production workflow owner.

ENVIRONMENT ISOLATION

Every run leases names, databases and credentials

Parallel or failed runs must not share queues, schemas, users or evidence paths. A lease resolves concrete resource names, records them before startup and removes only its own isolated resources during teardown.

EXECUTABLE ASSET 36-C - ISOLATED RESOURCE LEASE

```
public sealed record SliceLease(
    string RunId,
    string BrokerVHost,
    IReadOnlyDictionary<string, string> Databases,
    IReadOnlyDictionary<string, string> Principals,
    string EvidenceDirectory);

var lease = await allocator.CreateAsync(new LeaseRequest(
    Prefix: "slice36",
    Databases: ["alarm", "rootcause", "audit",
                "compliance", "reporting", "advisory"]), ct);
```

DECISION LEDGER 36-8

Isolation gate	Required proof
names	run id is embedded in every resource
credentials	one principal per runtime/migration owner
cleanup	explicit resource inventory; no broad wildcard delete

PREFLIGHT

No scenario starts on an ambiguous environment

The harness records build identity, schema markers, topology digest and service configuration identities before the first request. A dirty migration state, reused queue, unknown image or missing evidence writer stops the run.

EXECUTABLE ASSET 36-D - PREFLIGHT GATE

```
await preflight.RequireAsync(
    BuildIdentity.FromRepositoryAndArtifacts(),
    DatabaseState.AllEmptyAtRequiredMigration(),
    TopologyState.Matches(profile),
    ConfigurationState.HasExpectedIdentities(),
    BrokerState.NoReadyOrUnackedMessages(),
    EvidenceState.WritableAndEmpty(),
    ct);
```

WHY BEFORE ACT
A green scenario over stale rows or residual messages proves nothing. Preflight failure is evidence, not a reason to clean silently and continue.

DETERMINISM

Fixed time and identities are test-only substitutions

The production hosts are booted through their real composition paths. The harness replaces only explicitly approved nondeterministic boundaries: TimeProvider, UUID factory, authenticated test principal and external resource locations. Policy, handlers, mappings, transactions and workers remain production registrations.

EXECUTABLE ASSET 36-E - DELIBERATE SUBSTITUTIONS

```
builder.ConfigureTestServices(services =>
{
    services.RemoveAll<TimeProvider>();
    services.AddSingleton<TimeProvider>(
        new FakeTimeProvider(Golden.StartUtc));

    services.RemoveAll<IIdentityFactory>();
    services.AddSingleton<IIdentityFactory>(
        Golden.IdentityFactory());

    services.RemoveAll<IPrincipalIssuer>();
    services.AddSingleton<IPrincipalIssuer>(
        Golden.PrincipalIssuer());
});

services.AssertOnlyApprovedReplacements(
    typeof(TimeProvider), typeof(IIdentityFactory),
    typeof(IPrincipalIssuer));
```

REPRODUCIBILITY LIMIT
Deterministic time makes assertions stable. It does not prove production clock synchronization; clock skew remains a separate operational test.

PRODUCTION ENTRY

The scenario enters through the secured Alarm HTTP edge

Five observations are submitted to the exact Chapter 31 endpoint under the alarm.observation.submit capability. The fifth crosses the flood policy. Repeating the request with the same idempotency identity must return the stored outcome, while changed bytes must conflict without a second decision.

EXECUTABLE ASSET 36-F - AUTHENTICATED GOLDEN INPUT

```
using var client = alarmFactory.CreateAuthorizedClient(
    capability: "alarm.observation.submit",
    detectorScope: "site-ath-17/line-02");

foreach (var request in Golden.AlarmObservations)
{
    request.CorrelationId.Should().Be("request-am-20482");
    using var response = await client.PostAsJsonAsync(
        "/api/v1/alarm-observations", request, ct);
    response.StatusCode.Should().Be(HttpStatusCode.Accepted);
}
```

DECISION LEDGER 36-11

Input coordinate	Golden value
idempotency key	idem-am-7f9c2a
threshold decision	decision-am-9184 / flood-am-6031
scope	site-ath-17 / line-02

TX-A

Alarm acceptance finishes before any broker success

The fifth request commits the accepted observation, deterministic flood decision and one Pending outbox body in AlarmManagementDb. The HTTP 202 is justified by local durability. The broker can be unavailable at this instant without changing that response.

EXECUTABLE ASSET 36-G - OWNER-LOCAL ALARM ORACLE

```
await alarmDb.AssertTransactionAsync("TX-A", tx => tx
    .HasAcceptedObservation("alarm-am-9005")
    .HasFloodDecision("flood-am-6031", "decision-am-9184")
    .HasPendingOutbox(FLOOD_MESSAGE_ID)
    .HasCanonicalBodySha256(FLOOD_BODY_SHA256)
    .HasNoPublishedTimestamp());

await foreignStores.AssertNoRowsAsync(
    "RootCauseDb", "AuditDb", "ComplianceDb", "ReportingDb");
```

ALLOWED CLAIM
At this checkpoint AlarmManagement has durably decided and intends to publish. Nothing yet proves broker acceptance or a downstream effect.

FIRST PUBLIC FACT

AlarmFloodDetected.v1 is compared as captured bytes

The relay capture stores body and broker properties before any decoder formats them. The test validates MessageId, route, content type, canonical length, content fingerprint and final body digest against the Chapter 32 corpus.

EXECUTABLE ASSET 36-H - BYTE CORPUS ASSERTION

```
var delivery = await brokerCapture.RequiredAsync(
    "019bc4d4-5dc1-7b8d-9d48-542d25315356", ct);

delivery.Route.Should().Be(
    "alarm-management.alarm-flood-detected.v1");
delivery.Body.Length.Should().Be(1003);
Sha256.Hex(delivery.Body).Should().Be(
    "7cf98319c20a5915d5710bbcbb5878fe51cdae89a5b68081c135fd0e962c276a");
Envelope.Read(delivery.Body).ContentFingerprint.Should().Be(
    "sha256:ef70dfaafe0e08599eb82877a77e60fe3d94b2ae0aa1456d2a175af9e2712b97");
```

NO CONVENIENT RESERIALIZATION
The assertion hashes the exact captured body. A semantically similar object serialized with different bytes is a contract failure, not a pass.

OUTBOX RELAY

Published means correlated broker acceptance

The Alarm relay claims one outbox lease, publishes with mandatory routing and correlates the returned confirm to that exact row. Only then does it mark the intent Published. A timeout leaves the outcome Unknown and relies on safe replay.

EXECUTABLE ASSET 36-I - RELAY CHECKPOINT

```
var publication = await relayEvidence.RequiredAsync(
    messageId: FLOOD_MESSAGE_ID, ct);

publication.Mandatory.Should().BeTrue();
publication.Confirm.Should().Be(PublishConfirm.Ack);
publication.Return.Should().BeNull();
publication.BodySha256.Should().Be(FLOOD_BODY_SHA256);

await alarmDb.AssertOutboxAsync(
    FLOOD_MESSAGE_ID, OutboxState.Published);
```

DECISION LEDGER 36-14

Crash point	Recoverable state
before publish	Pending row remains
after broker accept / before local mark	redelivery; consumer dedupes
after Published	no second relay claim

ROOTCAUSE ADMISSION

A delivery becomes a command only after strict validation

RootCause authenticates the relay principal, verifies exchange/route, MessageId, producer, fingerprint and payload invariants, then translates the public fact into a local application command. It never queries AlarmManagementDb to fill a missing field.

EXECUTABLE ASSET 36-J - ADMISSION ASSERTION

```
var admitted = rootCauseAdmission.Admit(
    capturedFloodDelivery, authenticatedRelay);

admitted.MessageId.Should().Be(FLOOD_MESSAGE_ID);
admitted.CorrelationId.Should().Be("request-am-20482");
admitted.CausationId.Should().Be("decision-am-9184");
admitted.Command.SourceDecisionId.Should().Be("flood-am-6031");
admitted.Command.Evidence.Should().HaveCount(5);
alarmManagementProbe.DatabaseReads.Should().Be(0);
```

SUFFICIENT PUBLIC FACT
RootCause can open the case from the event alone. Any required private database read would be evidence that the public contract is insufficient.

TX-RC-OPEN

Case creation and CaseOpened intent commit together

RootCauseDb records the inbox receipt, inbound lineage, one case, five ordered evidence items and one Pending RootCauseCaseOpened.v1 body in the same local transaction. The broker delivery remains unacknowledged until the commit outcome is known.

EXECUTABLE ASSET 36-K - ROOTCAUSE OPEN ORACLE

```
await rootCauseDb.AssertTransactionAsync("TX-RC-OPEN", tx => tx
    .HasInboxReceipt(FLOOD_MESSAGE_ID)
    .HasInboundLineage(FLOOD_MESSAGE_ID, FLOOD_BODY_SHA256)
    .HasCase(CASE_ID, sourceDecisionId: "flood-am-6031")
    .HasOrderedEvidence(count: 5)
    .HasPendingOutbox(CASE_OPENED_MESSAGE_ID));

await broker.AssertSettledAsync(
    FLOOD_MESSAGE_ID, Settlement.AckAfterCommit);
```

DUPLICATE RESULT
A later delivery with the same MessageId and body resolves the stored receipt. A changed body under that identity is quarantined as an identity conflict.

SECOND PUBLIC FACT

RootCauseCaseOpened.v1 preserves the first causal edge

The public case identity is allocated once inside RootCause. Its envelope retains the journey correlation and names the AlarmFloodDetected MessageId as its immediate cause. Reporting may observe it before the verdict and legitimately expose an Open case.

DECISION LEDGER 36-17

Coordinate	Exact value
CaseId	019bc4d4-6dfe-7bcb-9de8-3af71979285a
MessageId	019bc4d4-6ea0-7d14-a57a-3e8643c55f13
correlation / causation	request-am-20482 / 019bc4d4-5dc1-7b8d-9d48-542d25315356
stream	root-cause/case/019bc4d4-6dfe-7bcb-9de8-3af71979285a @ 1
body	876 bytes / 454f51cf4b0250b9c855346a6648a7d73530f19687ee3b52cda2edf9d0339c5b

INTERMEDIATE TRUTH
A Reporting row at stream position 1 is not stale corruption. It is an honest intermediate observation until position 2 arrives.

VERDICT COMMAND

The verdict enters through RootCause's authorized application edge

The scenario waits for the stored case, then submits the deterministic verdict command under the capability that RootCause owns. The command references evidence revision 1 and does not write Audit, Compliance or Reporting state.

EXECUTABLE ASSET 36-L - DETERMINISTIC VERDICT INPUT

```
var command = new IssueRootCauseVerdictV1(
    CommandId: "decision-rc-7184",
    CaseId: Guid.Parse(CASE_ID),
    VerdictCode: "COMMON_POWER_SUPPLY_INSTABILITY",
    Confidence: 0.94m,
    EvidenceRevision: 1,
    PolicyIdentity: "root-cause/verdict/3",
    CorrelationId: "request-am-20482");

var response = await rootCauseApi.IssueVerdictAsync(command, ct);
response.StatusCode.Should().Be(HttpStatusCode.Accepted);
```

AUTHORIZATION BOUNDARY
The test principal may invoke the public/application edge. It receives no RootCauseDb write credential and cannot manufacture a verdict outbox row.

TX-RC-VERDICT

Verdict state and VerdictIssued intent are one commit

RootCause verifies the case is open at evidence revision 1, applies policy root-cause/verdict/3, stores the verdict identity and adds one Pending RootCauseVerdictIssued.v1 body. No subscriber availability check participates in this transaction.

EXECUTABLE ASSET 36-M - VERDICT COMMIT ORACLE

```
await rootCauseDb.AssertTransactionAsync("TX-RC-VERDICT", tx => tx
    .HasVerdict(VERDICT_ID,
        code: "COMMON_POWER_SUPPLY_INSTABILITY",
        confidence: 0.94m,
        policy: "root-cause/verdict/3")
    .HasEvidenceRevision(1, count: 5)
    .HasPendingOutbox(VERDICT_MESSAGE_ID)
    .HasStreamPosition(CASE_ID, expected: 2));

await subscribers.AssertNoSynchronousCallsAsync();
```

COMMIT BOUNDARY
The Accepted response proves a RootCause-owned verdict and durable publication intent. It does not prove Audit evidence, Compliance review, Reporting projection or advice.

THIRD PUBLIC FACT

RootCauseVerdictIssued.v1 is the fan-out anchor

The exact Chapter 34 body is published once and independently delivered to Audit, Compliance and Reporting. Its causation is the owner-local verdict decision, while each subscriber records the public MessageId as its input lineage.

DECISION LEDGER 36-20

Coordinate	Frozen value
VerdictId / MessageId	019bc4d4-81c0-7ea1-82d5-4f9b8719d0a1 / 019bc4d4-82a4-77f2-ae06-09b172aa59db
correlation / causation	request-am-20482 / decision-rc-7184
stream	root-cause/case/019bc4d4-6dfe-7bcb-9de8-3af71979285a @ 2
content fingerprint	07d530ef636be3d2af8dcc3b20cd91b212a076a4e70c1a9c55730930fbfe1e2e
body	1157 bytes / 192e5e73c5a35d438e2ee9edb0a9e7044293c29bf321ba1abe34f191d8b44c00

FAN-OUT RULE
One public fact produces independent local consequences. There is no broker transaction that atomically commits all subscriber databases.

REQUIRED ROUTING

Three core queues must receive the exact verdict body

The topology probe resolves the verdict route to the required core queues. Every captured delivery must have the same MessageId and body digest. Consumer-specific delivery tags and observation timestamps are allowed to differ.

EXECUTABLE ASSET 36-N - CORE FAN-OUT GATE

```
var routed = await topology.QueuesMatchingAsync(
    "nexus.events",
    "root-cause.root-cause-verdict-issued.v1", ct);

routed.Core.Should().BeEquivalentTo([
    "audit.root-cause-verdicts.v1",
    "compliance.root-cause-verdicts.v1",
    "reporting.integration-events.v1"
]);

await captures.AssertSameBodyAsync(
    VERDICT_MESSAGE_ID, VERDICT_BODY_SHA256, routed.Core, ct);
```

OPTIONAL ROUTE EXCLUSION
An advisory queue may appear only in the optional set. It is never added to the core route requirement evaluated by the RootCause relay.

TX-AUD

Audit appends one immutable observed consequence

Audit validates the verdict delivery and commits its inbox receipt, inbound lineage and immutable evidence record in AuditDb. A duplicate returns the stored receipt; no update path edits the original evidence payload.

EXECUTABLE ASSET 36-O - AUDIT ORACLE

```
await auditDb.AssertTransactionAsync("TX-AUD", tx => tx
    .HasInboxReceipt(VERDICT_MESSAGE_ID)
    .HasInboundLineage(
        sourceMessageId: VERDICT_MESSAGE_ID,
        correlationId: "request-am-20482")
    .HasEvidenceRecord(
        caseId: CASE_ID,
        verdictId: VERDICT_ID,
        code: "COMMON_POWER_SUPPLY_INSTABILITY")
    .HasNoMutableReplacement());
```

DECISION LEDGER 36-22

Audit fact	Claim
record exists	Audit observed the approved VerdictIssued body
record absent	Audit has not recorded it yet; verdict may still exist

TX-COMP

Compliance opens one Pending review, not a verdict copy

Compliance consumes the same public fact and commits one Pending review with its own lifecycle. It records the source verdict identity and policy but does not reinterpret confidence or promote the verdict into compliance approval.

EXECUTABLE ASSET 36-P - COMPLIANCE ORACLE

```
await complianceDb.AssertTransactionAsync("TX-COMP", tx => tx
    .HasInboxReceipt(VERDICT_MESSAGE_ID)
    .HasReview(
        rootCauseCaseId: CASE_ID,
        sourceVerdictId: VERDICT_ID,
        state: ComplianceReviewState.Pending)
    .HasSourcePolicy("root-cause/verdict/3")
    .HasNoAutomaticApproval());
```

SEMANTIC BOUNDARY
A root-cause verdict is evidence for compliance work. It is not a compliance decision and cannot skip human or policy authorization.

TX-REP

Reporting reaches stream position 2 contiguously

Reporting may receive CaseOpened and VerdictIssued in either delivery order. It durably buffers a gap, then commits inbox, event ledger, summary and contiguous checkpoint. The final oracle requires the case stream watermark to reach exactly 2.

EXECUTABLE ASSET 36-Q - REPORTING CONVERGENCE ORACLE

```
await reportingDb.AssertProjectionAsync(
    projection: "root-cause-case-summary",
    generation: activeGeneration,
    streamId: $"root-cause/case/{CASE_ID}",
    expectedPosition: 2,
    expectedMessageId: VERDICT_MESSAGE_ID,
    assertRow: row => row
        .HasStatus("VerdictIssued")
        .HasVerdict(VERDICT_ID)
        .HasCode("COMMON_POWER_SUPPLY_INSTABILITY")
        .HasConfidence(0.94m));
```

DELAYED TRUTH
The projection is accepted only after the gap count is zero and the checkpoint advances contiguously. Delivery order is never treated as producer order.

BFF OBSERVATION

The final HTTP answer carries freshness evidence

The BFF reads only Reporting-owned state and returns the verdict summary, watermark, source age, projection age and sampled update status. The scenario requires current-as-observed; it does not relabel the projection as globally current truth.

EXECUTABLE ASSET 36-R - FINAL READ ASSERTION

```
var observation = await bff.GetRootCauseCaseAsync(CASE_ID, ct);

observation.Value.Status.Should().Be("VerdictIssued");
observation.Value.VerdictId.Should().Be(VERDICT_ID);
observation.Evidence.LastContiguousPosition.Should().Be(2);
observation.Evidence.GapCount.Should().Be(0);
observation.Evidence.UpdateStatus.Should().Be("current-as-observed");
observation.Evidence.ObservedAtUtc.Should().NotBe(default);
```

UI WORDING
The operator may see Current as observed with the observation timestamp. The UI never displays source-authoritative now or hides an unknown/failed status behind the last value.

DISTRIBUTED OBSERVATION

Completion is an evidence vector, not one boolean row

A scenario result records each checkpoint separately. The aggregate status becomes Passed only if all required coordinates agree. A partial result names the missing or conflicting boundary instead of returning a generic timeout.

EXECUTABLE ASSET 36-S - CONVERGENCE VECTOR

```
public sealed record SliceConvergence(
    Checkpoint AlarmCommit,
    Checkpoint FloodPublication,
    Checkpoint RootCauseOpenCommit,
    Checkpoint CaseOpenedPublication,
    Checkpoint VerdictCommit,
    Checkpoint VerdictPublication,
    Checkpoint AuditCommit,
    Checkpoint ComplianceCommit,
    Checkpoint ReportingPositionTwo,
    Checkpoint BffObservation);

public bool IsCoreComplete => AllRequired.All(x => x.IsProved);
```

DECISION LEDGER 36-26

Status	Meaning
Proved	artifact exists and coordinates match
Pending	valid earlier state; bounded wait not yet satisfied
Failed	explicit invariant or identity mismatch
Unknown	observation path could not establish outcome

CAUSATION VALIDATOR

Business lineage is checked as an edge set

The validator builds an expected directed graph from frozen identities. It rejects a missing edge, a cycle, an event that names itself as cause, and a subscriber record that points to the verdict decision instead of the delivered public MessageId.

EXECUTABLE ASSET 36-T - EXACT CAUSAL EDGES

```
var expected = new CausalEdge[]
{
    new("decision-am-9184", FLOOD_MESSAGE_ID),
    new(FLOOD_MESSAGE_ID, CASE_OPENED_MESSAGE_ID),
    new("decision-rc-7184", VERDICT_MESSAGE_ID),
    new(VERDICT_MESSAGE_ID, "audit-evidence"),
    new(VERDICT_MESSAGE_ID, "compliance-review"),
    new(VERDICT_MESSAGE_ID, "reporting-position-2")
};

lineageGraph.AssertExactBusinessEdges(expected);
lineageGraph.AssertCorrelation("request-am-20482");
```

OWNER-LOCAL DECISIONS
decision-am-9184 and decision-rc-7184 are valid causal nodes even though they are not public broker messages. Evidence must label their owning boundary.

TRACE CONTINUITY

traceparent supports diagnosis; it does not define causation

Each HTTP and messaging hop propagates W3C Trace Context when valid, records the producer and consumer spans, and links redelivery attempts without changing business identifiers. A missing trace is an observability defect, not permission to rewrite CausationId.

EXECUTABLE ASSET 36-U - TWO INDEPENDENT ASSERTIONS

```
businessLineage.CorrelationId.Should().Be("request-am-20482");
businessLineage.CausationEdges.Should().Equal(expectedEdges);

traceEvidence.RootTraceId.Should().NotBeEmpty();
traceEvidence.Spans.Should().ContainRequired([
    "alarm.http.accept", "alarm.outbox.publish",
    "rootcause.consume", "rootcause.verdict",
    "audit.consume", "compliance.consume", "reporting.consume"
]);

traceEvidence.Values.ShouldNotBeUsedAsBusinessIds();
```

TWO PLANES
Business lineage survives sampling and tracing-tool changes. Trace context improves timing diagnosis but cannot become the domain's durable identity scheme.

CONSISTENCY MODEL

No checkpoint pretends to be globally simultaneous

Owner timestamps, source occurrence times, local commit sequence numbers and broker observations are recorded as different coordinates. The evidence bundle can reconstruct a partial order; it cannot claim a single globally serializable instant.

DECISION LEDGER 36-29

Coordinate	Use	Misuse rejected
OccurredAtUtc	producer business occurrence	cross-host transaction order
committedAt / local sequence	owner-local durability	global database sequence
broker observedAt	transport observation	business cause
trace span time	diagnostic latency	contract identity

ALLOWED CONCLUSION
The graph proves that required durable effects eventually converged with preserved identities. It does not prove they were atomically visible together.

FAULT LABORATORY

Outages are injected at owner boundaries

The scenario has a happy-path profile and focused recovery profiles. Each profile makes exactly one dependency unavailable, observes the durable intermediate state, restores the dependency and waits for convergence without rewriting or manually advancing business data.

DECISION LEDGER 36-30

Fault	Intermediate truth	Recovery proof
broker offline after TX-A	Flood outbox Pending	same body later Published
RootCause offline	queue retains Flood	one case after restart
Audit offline	core verdict and other consumers progress	one evidence record later
Reporting offline	owner truth complete; view delayed	watermark reaches 2
RL offline	core complete	optional backlog only

FOCUSED EVIDENCE
One giant chaos test is hard to diagnose. The matrix preserves the same corpus while varying one failure boundary per run.

CRASH SCHEDULE

Every commit/settlement gap has a named replay outcome

Crash injection is placed before local commit, after local commit but before acknowledgement, after broker acceptance but before outbox completion, and during Reporting gap repair. The oracle asserts row counts and identities, not merely eventual HTTP success.

EXECUTABLE ASSET 36-V - CRASH POINT THEORY

```
[Theory]
[MemberData(nameof(CrashSchedule.AllOwnerBoundaries))]
public async Task SLICE36_CRASH_001_recovers_without_new_meaning(
    CrashPoint point)
{
    await scenario.RunUntilAsync(point);
    await environment.RestartCrashedOwnerAsync();
    var result = await scenario.ResumeAndAwaitConvergenceAsync();

    result.Core.IsCoreComplete.Should().BeTrue();
    await corpus.AssertNoNewPublicIdentityAsync();
    await stores.AssertNoDuplicateSemanticEffectAsync();
}
```

UNKNOWN COMMIT
A client timeout is not treated as rollback. Recovery queries the owner-local idempotency record or retries the same identity and body.

RETRY DISCIPLINE

Retries repeat identity and re-open scoped resources

Transport retries reuse the same public MessageId and exact bytes. Application semantic retries reuse the command/idempotency identity. Provider retries create a fresh DbContext and transaction; they do not reuse a poisoned unit of work.

DECISION LEDGER 36-32

Retry	Must remain stable	Must be fresh
HTTP	idempotency key + request digest	request attempt / trace span
outbox	MessageId + canonical body	channel / confirm sequence
consumer	MessageId + received body	delivery tag / DbContext
provider	semantic operation identity	transaction / DbContext

FORBIDDEN RETRY
Allocating a new public MessageId or verdict identity because an acknowledgement was lost creates new meaning and fails the corpus gate.

CORE CONVERGENCE

The happy path finishes only after negative gates pass

Presence checks are insufficient. The final gate also requires no identity conflicts, no quarantined golden messages, no unacknowledged required deliveries, no stream gaps and no unexpected synchronous cross-owner database access.

EXECUTABLE ASSET 36-W - FINAL CORE GATE

```
result.Core.IsCoreComplete.Should().BeTrue();

await evidence.AssertZeroAsync(
    Metric.IdentityConflicts,
    Metric.QuarantinedGoldenMessages,
    Metric.RequiredUnackedDeliveries,
    Metric.ReportingStreamGaps,
    Metric.CrossOwnerDatabaseReads,
    Metric.UnexpectedPublicIdentities);

await topology.AssertNoRequiredBacklogAsync(ct);
```

BOUNDED WAIT
AwaitConvergence uses an explicit deadline and polling/backoff policy. Deadline expiry returns the unresolved checkpoint vector and captured diagnostics; it never spins indefinitely.

EVIDENCE BUNDLE

Artifacts are addressable, hashed and reproducible

A passing local assertion is not release evidence until its outputs are retained. The bundle records the repository revision, build identity, package lock digest, environment manifest, exact commands, test results, database oracles, broker captures, trace export and redacted logs.

DECISION LEDGER 36-34

Artifact	Required coordinate
build.json	repository URI, commit, dirty flag, artifact digests
environment.json	profile, resource lease, configuration identities
scenario.trx	discovered/executed/failed/skipped counts
broker/	properties, routes, bodies, confirm/settlement chronology
stores/	provider-backed row oracles and migration identities
traces.otlp.json	trace ids/spans with redaction report

REPRODUCTION COMMAND
The manifest must contain the exact command and configuration digest needed to reproduce the profile from the named build. A prose statement that tests passed is rejected.

FAIL-CLOSED EVIDENCE

The manuscript specifies a run; it does not counterfeit one

No repository execution artifacts are attached to this chapter. Therefore the implementation and runtime status remain PENDING. A validator rejects missing files, zero discovery, stale timestamps, build mismatch, skipped required cases and evidence recorded under a different profile.

EXECUTABLE ASSET 36-X - MANIFEST VALIDATOR

```
validator.RequireFile("scenario.trx");
validator.RequireFile("build.json");
validator.RequireFile("environment.json");
validator.RequireDirectory("broker");
validator.RequireDirectory("stores");

validator.RequireTests(discovered: GreaterThan.Zero,
    failed: 0, skippedRequired: 0);
validator.RequireSameBuildAcrossArtifacts();
validator.RequireProfile(requestedProfile);
validator.RequireFreshness(maxAge: TimeSpan.FromHours(24));
```

CURRENT STATUS: PENDING
The chapter provides compile-oriented assets, exact fixtures, oracles and commands. It does not assert that https://github.com/gregory82gr/Nexus-1-phase-0 currently contains or has executed them.

OPTIONAL RL BRANCH

Advice is permitted only as a non-blocking observation

RL Advisory may observe RootCauseVerdictIssued.v1 and compute Recommend or Abstain. It cannot prevent the core scenario from completing, cannot change the RootCause verdict, and cannot execute an operational action. The branch is absent by default.

DECISION LEDGER 36-36

Question	Chapter 36 decision
Required for core?	No
Input	approved public VerdictIssued.v1 only
Output effect now	module-owned advisory snapshot or abstention
Operational command?	Never
Independent service?	Not yet justified

PRESERVATION RULE RL-ADV-001
A recommendation remains advice. Neither a handler, UI, adapter nor future consumer may reinterpret it as an authorized operational command.

OPTIONAL REGISTRATION

Disabled means no host, queue, binding or database activity

The optional profile adds a distinct queue bound to the exact verdict route, a subscriber registration and module-owned storage. When disabled, topology and process inventories must prove complete absence rather than a dormant worker polling shared infrastructure.

EXECUTABLE ASSET 36-Y - OPTIONAL TOPOLOGY GATE

```
if (profile is SliceProfile.CoreWithAdvisory)
{
    await topology.AssertBindingAsync(
        "rl-advisory.root-cause-verdicts.v1",
        "root-cause.root-cause-verdict-issued.v1", ct);
    hosts.Should().Contain("RlAdvisorySubscriber");
}
else
{
    topology.ShouldNotContainQueue("rl-advisory.root-cause-verdicts.v1");
    hosts.ShouldNotContain("RlAdvisorySubscriber");
    advisoryDb.ConnectionAttempts.Should().Be(0);
}
```

PRODUCER INDEPENDENCE
RootCause does not know whether the optional queue exists. Its required publication gate is unchanged between profiles.

RL ADMISSION

The adapter authenticates and validates before model work

The subscriber applies the same envelope and byte checks as other consumers, restricts the producer to root-cause and accepts only VerdictIssued schema 1. It translates public fields into an internal advisory state; it never reads RootCauseDb or ReportingDb.

EXECUTABLE ASSET 36-Z - STRICT ADVISORY ADMISSION

```
var delivery = advisoryAdmission.Admit(
    rawDelivery, authenticatedBrokerPrincipal);

delivery.Envelope.DemandProducer("root-cause");
delivery.Envelope.DemandEvent(
    "nexus1.root-cause.root-cause-verdict-issued.v1", schema: 1);
delivery.Envelope.DemandMessageId(rawDelivery.Properties.MessageId);
fingerprints.Verify(delivery.Envelope, rawDelivery.CopiedBody);

privateDatabaseProbe.TotalReads.Should().Be(0);
```

CONTRACT-ONLY INPUT
If the public verdict lacks a value required by the advisor, the correct outcomes are contract evolution or abstention, never a private cross-database read.

ADVISORY STATE

Public verdict fields map to a bounded feature vector

The adapter records source identities, policy/model identities and the feature mapping version. It allows only reviewed categorical mappings and bounded numeric values. Unknown verdict codes or unsupported mapping versions produce Abstain with a stable reason.

EXECUTABLE ASSET 36-AA - EXPLICIT FEATURE MAPPING

```
public sealed record AdvisoryState(
    Guid SourceVerdictId,
    Guid SourceMessageId,
    string SiteClass,
    string LineClass,
    string VerdictCode,
    decimal VerdictConfidence,
    int EvidenceCount,
    string FeatureMapIdentity);

var state = featureMapper.Map(
    verdict, identity: "rl-advisory/features/1");
state.DemandFiniteAndBounded();
```

DECISION LEDGER 36-39

Rejected input	Outcome
unknown verdict code	Abstain(unsupported-verdict-code)
confidence outside 0..1	terminal invalid-contract
unknown feature map	Abstain(unsupported-feature-map)

POLICY IDENTITY

A model file without provenance is not a policy

The advisor identifies the algorithm, model artifact digest, training-data snapshot, feature map, action catalogue and decision thresholds. The combined PolicyIdentity is stored with every outcome. Floating latest aliases are forbidden in evidence-bearing profiles.

EXECUTABLE ASSET 36-AB - IMMUTABLE POLICY DESCRIPTOR

```
public sealed record AdvisoryPolicyIdentity(
    string Algorithm,
    string ModelSha256,
    string TrainingSnapshot,
    string FeatureMap,
    string ActionCatalogue,
    string Thresholds)
{
    public string Digest => CanonicalSha256.Of(this);
}

var policy = registry.GetRequired(
    "rl-advisory/power-instability/1");
policy.AssertArtifactDigest();
```

DETERMINISM CLAIM
Given the same admitted public fact and identical policy identity, the adapter must produce the same outcome bytes. This is a testable adapter guarantee, not a claim that the learned policy is optimal.

DECISION SEMANTICS

Recommend and Abstain are the only valid outcomes

The policy may return a ranked advisory action when confidence, coverage and safety gates pass; otherwise it abstains. A Q-value or discounted return is labeled as such and is never displayed as a probability. The action catalogue contains advice codes, not actuator targets.

EXECUTABLE ASSET 36-AC - TYPED RESULT

```
public abstract record AdvisoryOutcome
{
    public sealed record Recommend(
        string AdviceCode,
        decimal Score,
        string ScoreMeaning,
        AdvisoryPolicyIdentity Policy) : AdvisoryOutcome;

    public sealed record Abstain(
        string ReasonCode,
        AdvisoryPolicyIdentity Policy) : AdvisoryOutcome;
}

const string ScoreMeaning = "discounted-return-not-probability";
```

NO FORCED ANSWER
Abstention is a successful, auditable outcome. The module must not choose a low-confidence action merely to keep the UI populated.

TX-ADV

Receipt, lineage and advisory outcome commit locally

When enabled, the module commits one inbox receipt, exact input lineage, policy identity and either recommendation or abstention in its own store. It does not write an outbox because Chapter 36 activates no public advisory event.

EXECUTABLE ASSET 36-AD - ADVISORY LOCAL TRANSACTION

```
await using var tx = await advisoryDb.Database.BeginTransactionAsync(ct);

advisoryDb.Inbox.Add(receipt.Completed(delivery));
advisoryDb.InputLineage.Add(lineage.From(delivery));
advisoryDb.Decisions.Add(decisionRow.From(
    sourceVerdictId: VERDICT_ID,
    sourceMessageId: VERDICT_MESSAGE_ID,
    outcome, policyIdentity));

await advisoryDb.SaveChangesAsync(ct);
await tx.CommitAsync(ct);
return ConsumeOutcome.Committed(VERDICT_MESSAGE_ID);
```

NO PUBLIC FACT YET
A future outbox would require a separately approved RecommendationGenerated.v1 contract, routing policy and real consumer. This chapter deliberately stops before that boundary.

ADVISORY IDEMPOTENCY

Transport and semantic duplicates use different keys

The inbox key proves receipt of one MessageId. A semantic unique key prevents a replay or contract migration from producing two advisory decisions for the same source verdict under the same policy identity. A different policy identity creates a new, explicitly versioned evaluation.

EXECUTABLE ASSET 36-AE - TWO-KEY DATABASE ORACLE

```
model.Entity<AdvisoryInboxRow>().HasKey(x =>
    new { x.ConsumerName, x.MessageId });

model.Entity<AdvisoryDecisionRow>().HasIndex(x => new
{
    x.SourceVerdictId,
    x.PolicyIdentityDigest
}).IsUnique();

model.Entity<AdvisoryDecisionRow>().HasIndex(x =>
    x.SourceMessageId);
```

DECISION LEDGER 36-43

Collision	Classification
same MessageId + same bytes	transport duplicate
same verdict + same policy	semantic duplicate
same MessageId + changed bytes	identity conflict
same verdict + new policy	new evaluation, separately identified

NON-COMMAND ENFORCEMENT

Architecture and contracts prevent advice from becoming control

Naming is not enough. The advisory assembly cannot reference actuator, command-dispatch or AlarmManagement application packages; the advice contract contains no target endpoint, execution deadline or authorization token; the UI requires an explicit human/policy action outside this module.

EXECUTABLE ASSET 36-AF - PRESERVATION GATES

```
Types.InAssembly(RlAdvisoryAssembly)
    .ShouldNot().HaveDependencyOnAny(
        "Nexus1.Actuation",
        "Nexus1.AlarmManagement.Application",
        "Nexus1.CommandDispatch");

typeof(AdvisoryOutcome.Recommend)
    .ShouldNotExposeMembers([
        "Execute", "TargetEndpoint", "AuthorizationToken",
        "CommandPayload", "MustCompleteBy"
    ]);
```

RL-ADV-001
Any adapter that turns AdviceCode into a command belongs to an explicit authorized decision boundary with its own contract and evidence. It cannot be hidden inside RL Advisory.

OPTIONAL FAILURE

Advisory outage cannot reduce core convergence

With the optional queue enabled and the advisor offline, the queue may retain the verdict while Audit, Compliance and Reporting complete. With the queue disabled, there is no backlog to interpret. Restoring the subscriber produces one stored advisory outcome without replaying core effects.

EXECUTABLE ASSET 36-AG - OFFLINE ADVISORY PROOF

```
await environment.StopAsync("RlAdvisorySubscriber", ct);
await scenario.RunCoreThroughVerdictAsync(ct);

(await observer.AwaitCoreConvergenceAsync(ct))
    .IsCoreComplete.Should().BeTrue();
await topology.AssertReadyCountAsync(
    "rl-advisory.root-cause-verdicts.v1", expected: 1, ct);

await environment.StartAsync("RlAdvisorySubscriber", ct);
await advisoryDb.AwaitOneDecisionAsync(VERDICT_ID, ct);
await coreStores.AssertUnchangedAsync();
```

HISTORY WHEN ENABLING LATER
Adding the optional queue after facts have already passed does not magically recover history. An explicit replay source, start coordinate and retention proof are required.

FUTURE PUBLIC CONTRACT

RecommendationGenerated.v1 remains a candidate, not an activated fact

A future extraction may justify a public advisory fact. Chapter 36 records its semantic outline but does not claim a schema file, outbox row, exchange binding or consumer. This prevents an aspirational name from appearing as implemented architecture.

DECISION LEDGER 36-46

Candidate field	Required meaning before activation
recommendationId	stable producer-owned identity
sourceVerdictMessageId	direct public cause
adviceCode	non-command catalogue entry
score / scoreMeaning	bounded value with honest semantics
policyIdentity	model, data, mapping and threshold provenance
expiresAt / abstention	explicit usefulness and refusal semantics

STATUS: NOT ACTIVATED
RecommendationGenerated.v1 is a design candidate only. No architecture inventory, route gate or evidence manifest may list it as a present public contract.

EXTRACTION DECISION

An adapter becomes a service only after ownership evidence exists

Process isolation is not an architectural reward for using machine learning. Extraction is justified only when the capability has an independently valuable contract, deployment and failure boundary, sustained scaling need, data/store ownership, security policy and an accountable operating team.

DECISION LEDGER 36-47

Evidence dimension	Required before independent service
contract	stable input/output, compatibility policy, real consumer need
operations	SLO, error budget, on-call owner, runbooks, dashboards
deployment	independent release/rollback and failure isolation evidence
data	owned schema, migrations, retention, backup/restore, replay
security	separate principals, ACLs, threat model, audit trail
economics	measured load/cost or cadence benefit exceeding complexity

CURRENT DECISION
Keep RL Advisory as an optional adapter/module. The chapter records how to prove a future extraction, but present evidence does not justify a service boundary.

EVIDENCE MATURITY

Core and optional profiles have separate release claims

A core release may pass without the advisory profile. An advisory-enabled release must first inherit a passing core result, then prove its extra topology, store, contract admission, deterministic policy and non-command gates. The optional result can fail without rewriting the core result.

DECISION LEDGER 36-48

Profile	Required suites	Present manuscript status
Core	golden path + provider + broker + crash + BFF	PENDING runtime evidence
CoreWithAdvisory	Core plus RL admission, TX-ADV, offline, RL-ADV-001	PENDING runtime evidence
future RL service	all above plus extraction dimensions	NOT JUSTIFIED

NO STATUS INHERITANCE UPWARD
Passing the core profile does not silently approve RL. Passing the optional module does not justify service extraction. Each claim has its own gate.

SECURITY AND PRIVACY

The proof path uses least privilege and redacted evidence

Each service receives credentials only for its database and declared broker operations. The scenario principal is scoped to the two public input capabilities. Captures retain identifiers and digests needed for proof while redacting tokens, connection strings, raw credentials and unapproved payload fields.

DECISION LEDGER 36-49

Principal	Allowed	Denied
alarm-runtime	AlarmManagementDb DML + Alarm outbox publish	foreign databases
rootcause-runtime	RootCauseDb DML + RootCause publish/consume	AlarmManagementDb reads
subscribers	own DB + own queue consume	RootCauseDb reads
scenario	approved HTTP calls + read-only evidence probes	business-table writes
rl-advisory	own DB + optional queue consume	command/actuator paths

EVIDENCE HYGIENE
A redaction report lists removed fields and scans retained artifacts for configured secret patterns. Evidence that leaks a credential is a failed run.

OBSERVABILITY

Metrics explain the path without inventing high-cardinality dimensions

Service, operation, event type, schema version, outcome and bounded failure code are metric labels. MessageId, CaseId and CorrelationId belong in structured logs/traces under access controls, not metric labels. Each owner emits commit and settlement evidence at its boundary.

EXECUTABLE ASSET 36-AH - BOUNDED TELEMETRY

```
meter.Counter<long>("nexus.slice.effects")
    .Add(1,
        KeyValuePair.Create<string, object?>("owner", owner),
        KeyValuePair.Create<string, object?>("event_type", eventType),
        KeyValuePair.Create<string, object?>("outcome", outcome));

logger.LogInformation(
    "Slice effect {Owner} {MessageId} {CorrelationId} {Outcome}",
    owner, messageId, correlationId, outcome);
```

CARDINALITY RULE
Golden identifiers make one run easy to find, but they still remain log/trace attributes. Production metric dimensions must stay bounded.

PERFORMANCE ENVELOPE

Latency budgets are per boundary and advisory work is outside core

The capstone records time to local acceptance, outbox publication, downstream commit, Reporting convergence and BFF observation separately. One end-to-end percentile without stage evidence cannot identify queueing, provider or projection delay.

DECISION LEDGER 36-51

Measure	Start -> stop	Release interpretation
accept latency	HTTP receive -> TX-A commit	synchronous budget
publication lag	TX commit -> broker confirm	relay capacity
consumer lag	broker enqueue -> local commit	subscriber capacity
observation lag	source occurrence -> BFF evidence	user-visible delay
advisory lag	verdict enqueue -> TX-ADV	optional SLO only

NO BORROWED BUDGET
A slow or unavailable advisor cannot consume the core convergence deadline. Its result is evaluated in the optional profile after core completion.

CI EXECUTION

One command produces a reviewable, fail-closed bundle

The repository should expose a single documented entry point that restores locked dependencies, builds the named revision, starts isolated dependencies, runs the selected profile, validates evidence and archives the bundle. Individual hidden developer steps are not release procedure.

EXECUTABLE ASSET 36-AI - PROPOSED REPRODUCTION COMMAND

```
dotnet test tests/Nexus1.FullSlice.EvidenceTests \
  --configuration Release \
  --no-restore \
  --filter "Category=Slice36&Profile=Core" \
  --logger "trx;LogFileName=scenario.trx" \
  --results-directory artifacts/evidence/slice36/core

dotnet run --project tools/EvidenceValidator -- \
  artifacts/evidence/slice36/core/manifest.json
```

PROPOSED PATH
The command and paths are the chapter's target contract. They become factual only when committed and executed in the named repository/build; until then the manifest status is PENDING.

SCENARIO RECORD

The final JSON cannot say Passed without attached proof

The release record binds every checkpoint to an artifact path and digest. It lists missing requirements explicitly. A generated timestamp, author name or manual approval cannot replace machine-readable evidence.

EXECUTABLE ASSET 36-AJ - FAIL-CLOSED RECORD SHAPE

```
{
  "claimId": "NEXUS1-SLICE36-CORE-001",
  "profile": "Core",
  "status": "PENDING",
  "buildIdentity": null,
  "scenarioTrx": null,
  "brokerCaptureManifest": null,
  "storeOracleManifest": null,
  "traceExport": null,
  "missing": [
    "repository implementation",
    "provider-backed execution",
    "broker-backed execution"
  ]
}
```

NOTHING CLAIMS TO EXIST THAT DOES NOT
The honest PENDING record is stronger than an unsupported green badge. It names exactly what must be produced before the claim can change.

ADR AND HANDOFF

The first distributed slice is specified completely and honestly

Chapter 36 closes Part VI with exact owner-local transactions, three frozen public bodies, preserved correlation/direct causation, broker and provider proof paths, delayed-truth observation and an explicitly optional advisory adapter. It deliberately leaves durable multi-step workflow ownership to the next Part.

DECISION LEDGER 36-54

ADR-036 field	Decision
core slice	Alarm -> RootCause -> Audit/Compliance/Reporting -> BFF
completion	evidence vector; no global transaction
lineage	one correlation; direct-hop business causation; trace separate
RL Advisory	optional non-command adapter; no public output activated
service extraction	not justified without contract/ops/data/security/economic evidence
runtime status	PENDING until real build, .trx, broker/provider/trace artifacts exist

NEXT - PART VII, CHAPTER 37
The NEXUS-1 Distributed Process Manager introduces durable workflow state, expected events, timeouts, completion and correlation for the alarm-to-compliance path. Chapter 36's runner is a test/evidence harness only; it is not that production process manager.

OFFICIAL TECHNICAL ANCHORS

The implementation path ends in evidence, not assertion

- ASP.NET Core integration tests and WebApplicationFactory: https://learn.microsoft.com/aspnet/core/test/integration-tests
- Microsoft.Extensions.Time.Testing FakeTimeProvider: https://learn.microsoft.com/dotnet/api/microsoft.extensions.time.testing.faketimeprovider
- EF Core transactions and optimistic concurrency: https://learn.microsoft.com/ef/core/saving/transactions and https://learn.microsoft.com/ef/core/saving/concurrency
- RabbitMQ acknowledgements and publisher confirms: https://www.rabbitmq.com/docs/confirms
- RabbitMQ queues and ordering: https://www.rabbitmq.com/docs/queues
- RabbitMQ .NET client guide: https://www.rabbitmq.com/client-libraries/dotnet-api-guide
- W3C Trace Context: https://www.w3.org/TR/trace-context/
- OpenTelemetry messaging conventions: https://opentelemetry.io/docs/specs/semconv/messaging/
- RFC 8785 JSON Canonicalization Scheme: https://www.rfc-editor.org/rfc/rfc8785

WHAT THIS CHAPTER HAS PROVED
The chapter specifies the exact full-slice path, identities, bytes, owner-local transactions, direct causal edges, topology, bounded waits, provider/broker/crash evidence, final delayed-truth observation, optional RL admission and persistence, non-command preservation rule, and service-extraction criteria. It has not represented unexecuted repository artifacts as present.

CHAPTER CHECKPOINT

- Why is the scenario runner not a production coordinator or process manager?
- Which three canonical public bodies must remain byte-identical?
- Why does correlation remain constant while causation changes by hop?
- Which local commits may exist while the next service is offline?
- What negative gates must pass before core convergence is accepted?
- Why may RL Advisory fail without changing the core result?
- Which rule prevents a recommendation from becoming an operational command?
- What evidence is still required before runtime status leaves PENDING?
- What evidence would justify extracting the advisory adapter as a service?

HANDOFF TO PART VII
Chapter 37 begins Workflow, Consistency, and Repair by implementing the NEXUS-1 Distributed Process Manager. That durable production component will own expected events, timeouts, resumability and completion; the Chapter 36 runner remains confined to tests and evidence.