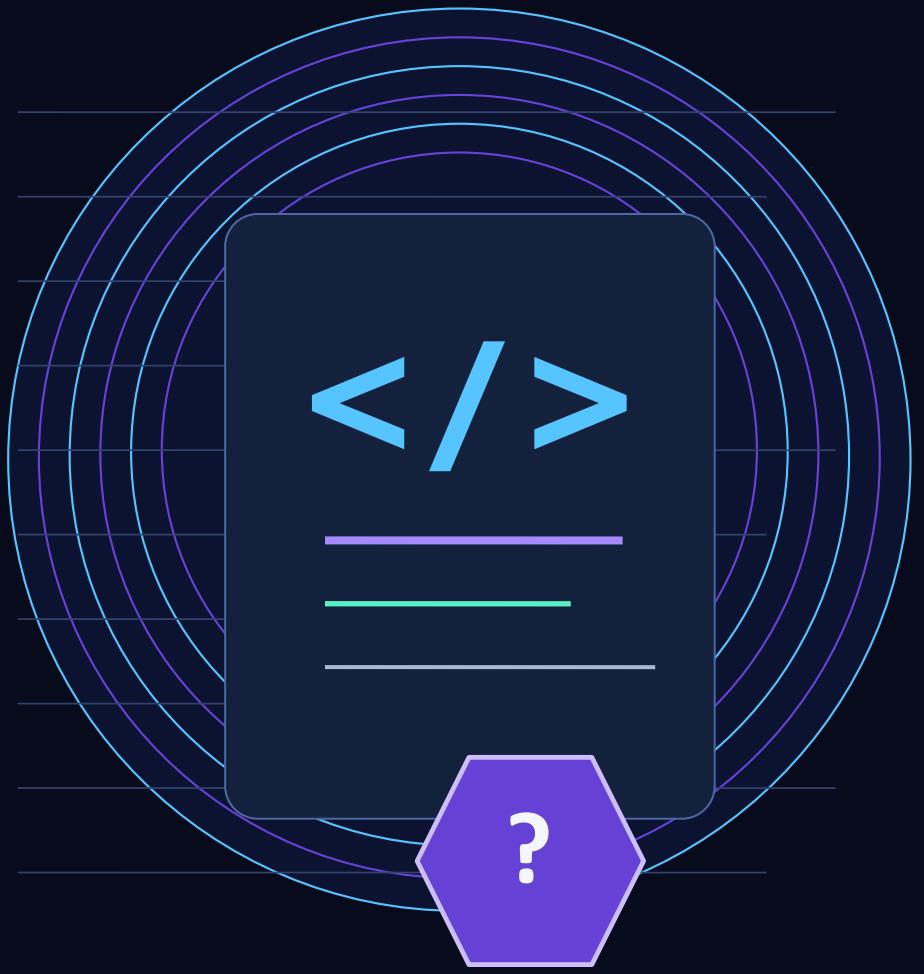
.NET CONF NEW YORK

AI WROTE THE CODE.

WHO MAKES SURE IT'S RIGHT?

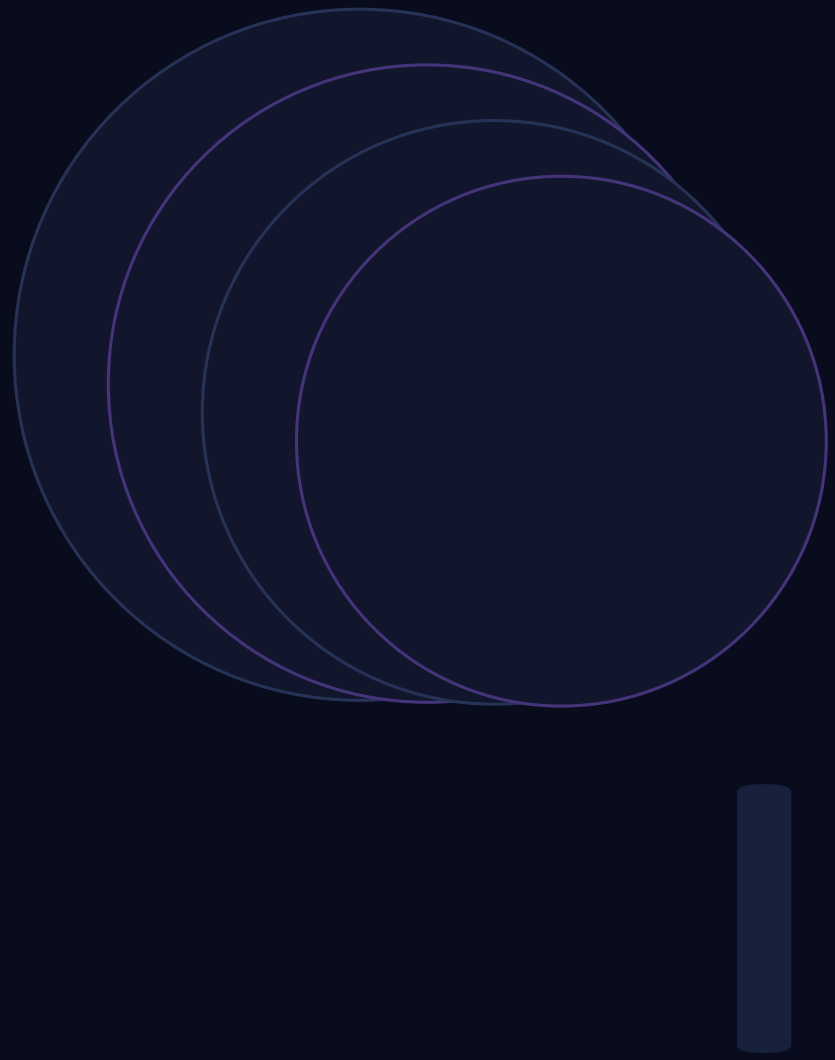


Engineering trust in the age of AI • .NET 11 + C# 15

ACT I

THE ILLUSION OF CORRECTNESS

When AI writes the code—and the evidence.



We automated agreeing with ourselves.

The same agent generates the implementation, the tests and the review.



We built a very efficient consensus machine.

The PR is green. The confidence is synthetic.

A realistic fictional PR dashboard.

PR #1842

Add account-to-account transfer API

✓ Build	PASSED
✓ Unit tests	24 / 24 PASSED
✓ AI review	NO ISSUES FOUND

UNKNOWN

Does it match the business rule?

No independent evidence

Would you merge it?

What your pipeline knows versus what your customers need.

EVIDENCE WE HAVE

- Code compiles
- 24 tests pass
- Static checks are green
- Change looks idiomatic

EVIDENCE WE NEED

- Requirements are explicit
- Invalid inputs are rejected
- Retries do not duplicate money
- Concurrency preserves invariants

Merge / Request changes / Need more context

ACT II

THE BUG NOBODY TESTED

It compiles. It passes. It moves money backwards.

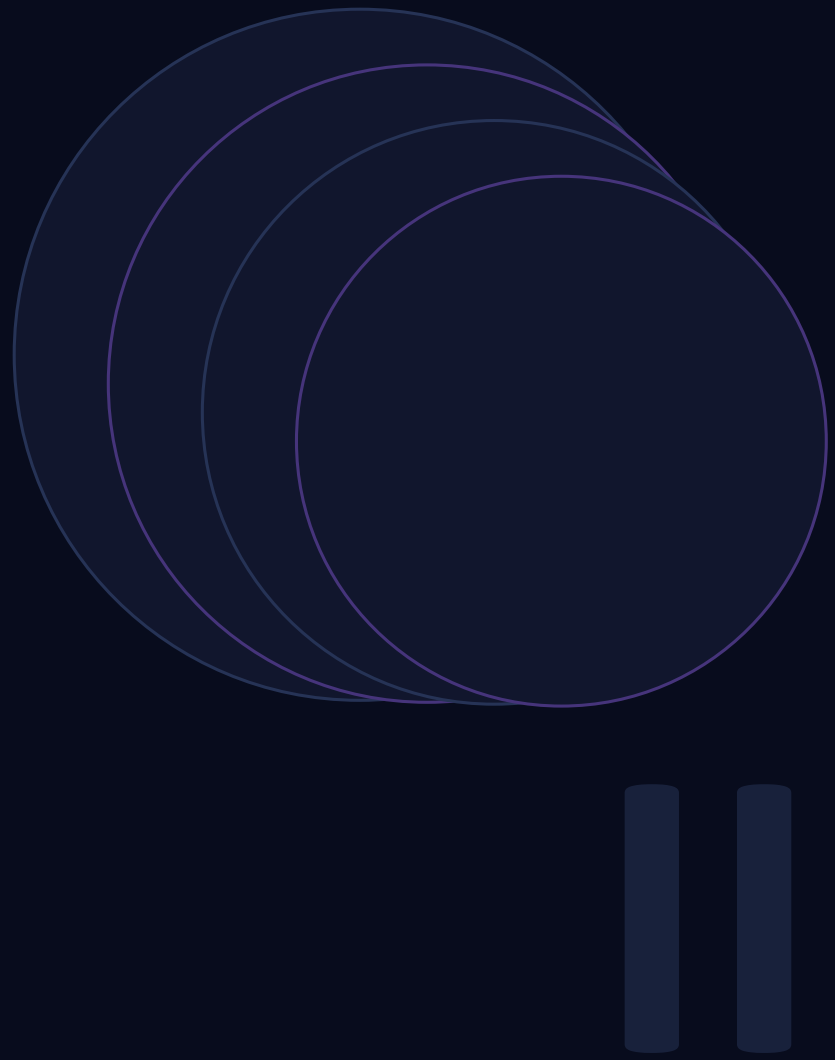


Exhibit A: the AI-generated transfer method

Nothing unusual. No compiler warnings. One missing precondition.

TransferService.cs

```
01 public TransferResult Transfer(
02     Account sender, Account receiver,
03     decimal amount)
04 {
05     if (sender.Balance < amount)
06         return TransferResult.Declined;
07     sender.Balance -= amount;
08     receiver.Balance += amount;
09     return TransferResult.Success;
10 }
```

THE MISSING RULE

amount must be positive

This is illustrative domain code, not a complete transaction implementation.

Negative $100. Everything goes the wrong way.

The invariant “total balance stays constant” still holds.

SENDER ACCOUNT

$500

$600

–$100

RECEIVER ACCOUNT

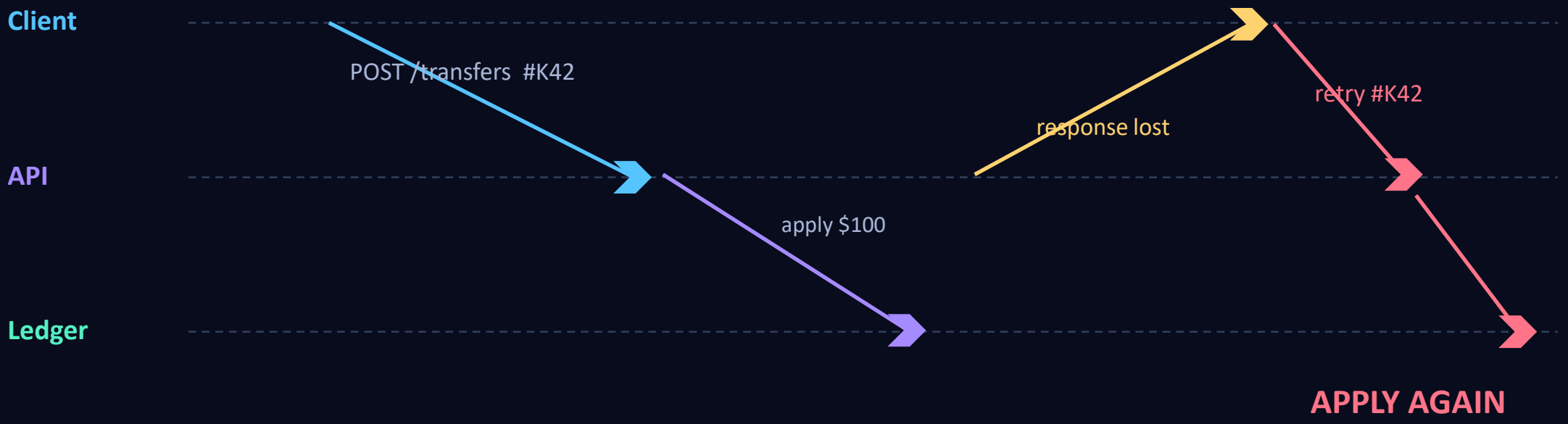
$200

$100

TOTAL BEFORE $700 → TOTAL AFTER $700

Same request. Twice the transfer.

Retry safety is a requirement, not a framework default.



Desired invariant: one idempotency key → at most one applied transfer

A better question: what must always be true?

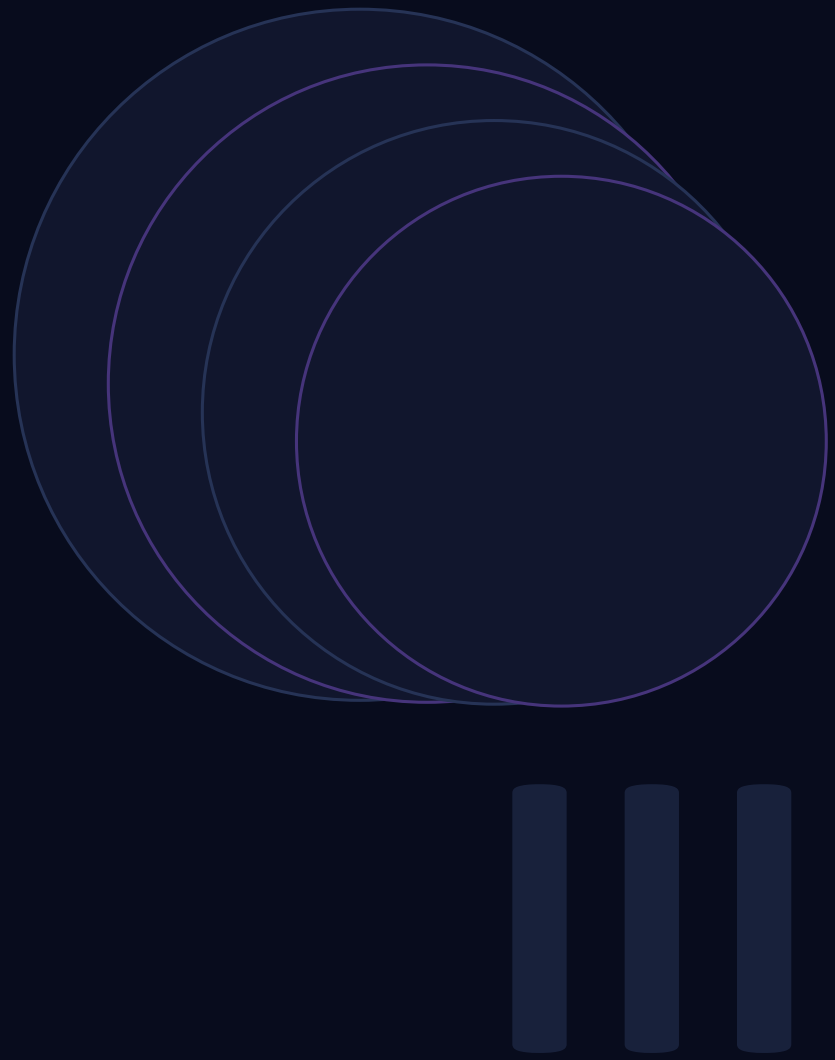
Define properties before asking for implementations or tests.

INPUT	Amount > 0; valid accounts
DIRECTION	Sender decreases, receiver increases
ATOMICITY	Either both changes apply—or neither
IDEMPOTENCY	Same key cannot apply twice
AUTHORIZATION	Caller is allowed to transfer

ACT III

MAKE WRONG STATES LOUDER

C# 15 moves intent into code the compiler understands.



C# 15 union types: outcomes are explicit.

No magic string. No nullable return value. No accidental “success by default”.

TransferOutcome.cs

```
01 public sealed record Applied(Guid Id);
02 public sealed record Rejected(string Reason);
03 public sealed record NeedsApproval(decimal Amount);
04
05 public union TransferOutcome(
06     Applied, Rejected, NeedsApproval);
```

APPLIED

REJECTED

NEEDS APPROVAL

One return type, three intentional outcomes.

The compiler can challenge missing outcomes.

When we extend the union, an incomplete switch stops being invisible.

Describe.cs

```
01 string Describe(TransferOutcome r) => r switch
02 {
03     Applied a => $"Applied: {a.Id}",
04     Rejected r => r.Reason,
05     NeedsApproval p => "Needs review"
06 };
07
08 // New case added: PendingSettlement
```

COMPILER-CHECKED EXHAUSTIVENESS

The unhandled case is reported at build time.

The type checker verifies cases - not business truth.

Union types or closed hierarchies?

Two ways to bound a domain. Different ownership trade-offs.

UNION

- Combine existing case types
- Value is one of declared cases
- Exhaustive switch expressions
- Good for API outcomes

CLOSED HIERARCHY

- Own the inheritance model
- Derivation limited to assembly
- Exhaustive hierarchy patterns
- Good for domain state models

C# 15 • GA

Validation now has an async path.

Useful for rules that must consult an external service or store.

API BOUNDARY

1. Basic shape & amount
2. Account eligibility lookup
3. Authorization decision

NEW IN .NET 11

IAsyncValidatableObject
AsyncValidationAttribute
ValidateObjectAsync

Async validation is not a database transaction.

The JSON contract can reintroduce ambiguity.

Two C# union cases, two object-shaped payloads.

ResultJson.cs

```
01 [JsonUnion(
02     TypeClassifier =
03       typeof(JsonUnionTypeStructuralClassifier))]
04 public union Result(Applied, Rejected);
05
06 // Both cases serialize as JSON objects
```

```
{ "Id": "..." }
```

```
{ "Reason": "..." }
```

Types are explicit. Wire formats still need design.

ACT IV

TESTS ARE EVIDENCE

But evidence is only as strong as the question asked.



Build an evidence ladder, not a test-count trophy.

Different checks expose different kinds of failure.



Confidence is layered. No single layer is a guarantee.

First, write the test the AI did not invent.

Derive acceptance cases from the requirement, not from the implementation.

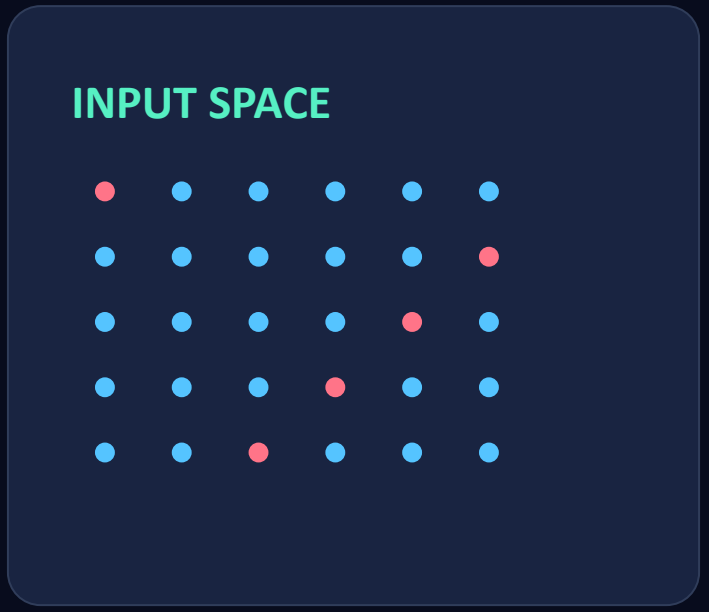
CASE	EXPECTED OUTCOME
Positive amount	Sender decreases by amount
Amount = 0	Reject without balance changes
Amount < 0	Reject without balance changes
Insufficient funds	Reject atomically
Duplicate key	Apply at most once

Property-based tests hunt beyond examples.

Generate many input values; preserve the invariants.

TransferProperties.cs • illustrative sketch

```
01 [Property]
02 public void NonPositiveAmountNeverMovesMoney(
03     int balanceA, int balanceB,
04     int nonPositive)
05 {
06     // Arrange bounded balances + amount <= 0
07     // Assert no balance changes
08     // Assert transfer is rejected
09 }
```



Edge cases are found,
then shrunk to minimal failures.

Illustrative test skeleton: adapt generators, ranges and assertions to your framework.

Mutation testing challenges the quality of the tests.

If we change the code, do the tests notice?

BoundaryMutation.cs

```
01 // Intended guard
02 if (amount <= 0)
03     return TransferResult.Rejected;
04
05 // Mutation: the boundary slips
06 if (amount < 0)
07     return TransferResult.Rejected;
```

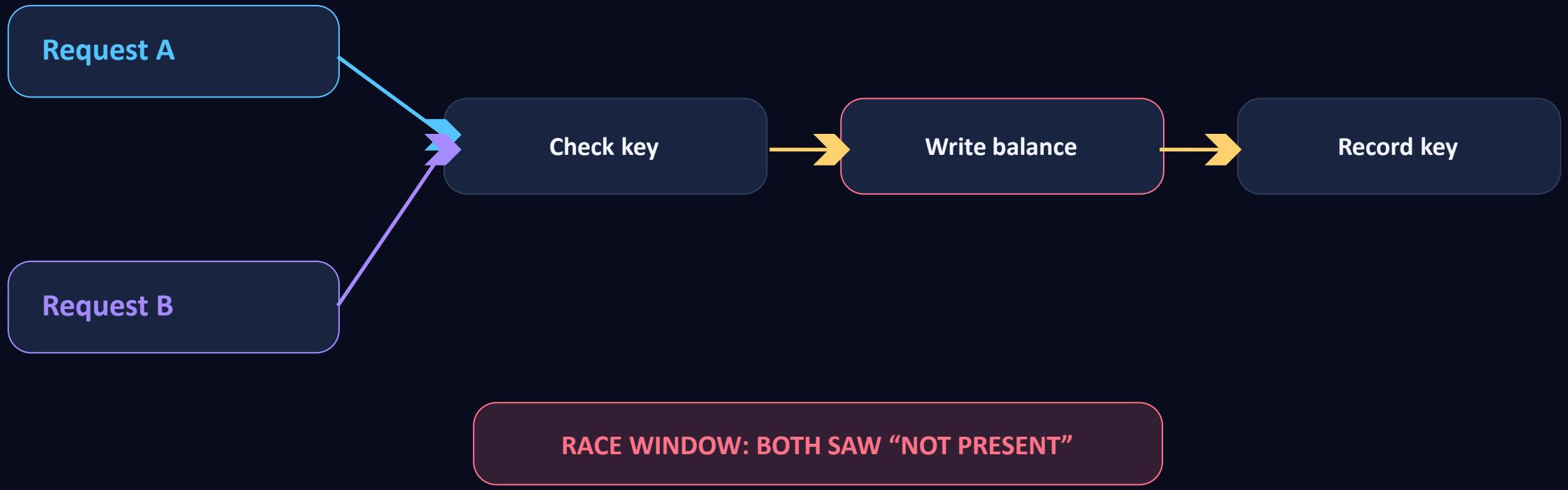
ZERO IS NOW ALLOWED

If tests still pass, the suite missed a meaningful boundary.

Mutation survived → test suite is missing evidence

Correct in isolation. Wrong under concurrency.

A retry key must be enforced atomically, even across service instances.



Test real storage semantics—not only a mocked repository.

.NET 11 helps run the verification machinery.

Microsoft.Testing.Platform gains useful run-level controls.

terminal • MTP runner

```
dotnet test --timeout 90s \
  --maximum-failed-tests 1 \
  --results-directory-layout per-module
```

TIMEOUT

Bound hung test runs

FAIL FAST

Stop after failure limit

ISOLATED RESULTS

Avoid artifact collisions

Runner controls make evidence reproducible. They do not prove correctness.

ACT V

FROM GREEN TO TRUSTWORTHY



Code is only one part of the system we ship.

The agent also chose your dependencies.

A correct algorithm can still ship an avoidable dependency risk.

AI SUGGESTED

SuperFastPayments.Helper

Newly published?
Provenance known?
Vulnerabilities checked?

nuget source policy

```
01 dotnet nuget update source public \
02   --min-publish-age-hours 48 \
03   --configfile nuget.config
```

Delay ≠ trust. Audit, pin and review.

A trace explains what happened. Not whether it should.

Technical observability needs business-level assertions and audit events.

TECHNICAL TRACE

HTTP POST	28 ms
Validate	3 ms
SQL UPDATE	12 ms
Response 200	1 ms

BUSINESS AUDIT

transfer id	K42
expected amount	$100
applied count	2 ✕
invariant status	VIOLATED

200 OK is not the same as correct.

The AI-generated PR needs an evidence dossier.

A practical standard for accepting generated change.

CONTRACT

Outcomes, constraints, authorization

COMPILER

Types, warnings, analyzers

BEHAVIOR

Examples, properties, mutations

SECURITY

Secrets, dependencies, threat model

OPERATIONS

Retry rules, telemetry, audit

ACCOUNTABILITY

Named owner and release decision

The agent contributes artifacts. A team owns the decision.

Same feature. Very different level of confidence.

Before and after the engineering work.

GREEN PR

- Implementation exists
- Self-generated tests pass
- AI review approves
- Business intent assumed

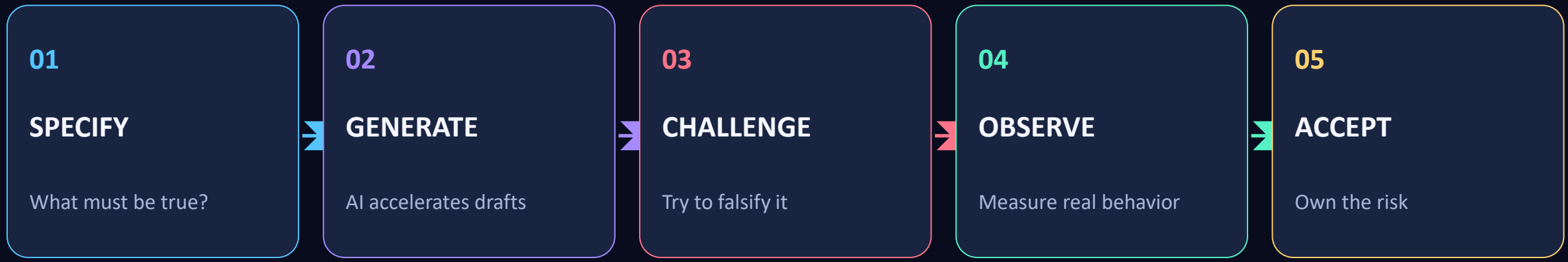
EVIDENCE-BACKED PR

- Outcomes explicitly typed
- Domain invariants tested
- Retries & failures challenged
- Risk owned and documented

Not “proven perfect.” Ready for accountable review.

The engineer becomes the architect of evidence.

AI shifts effort from typing implementations to specifying and verifying behavior.



Write less. Know more. Prove what matters.

Three truths to take back to your team.

The checks are useful. The conclusions must remain disciplined.

01

Compiling ≠ correctness

The compiler checks what the language can express.

02

Passing tests ≠ sufficient evidence

Tests are only as strong as their requirements and assertions.

03

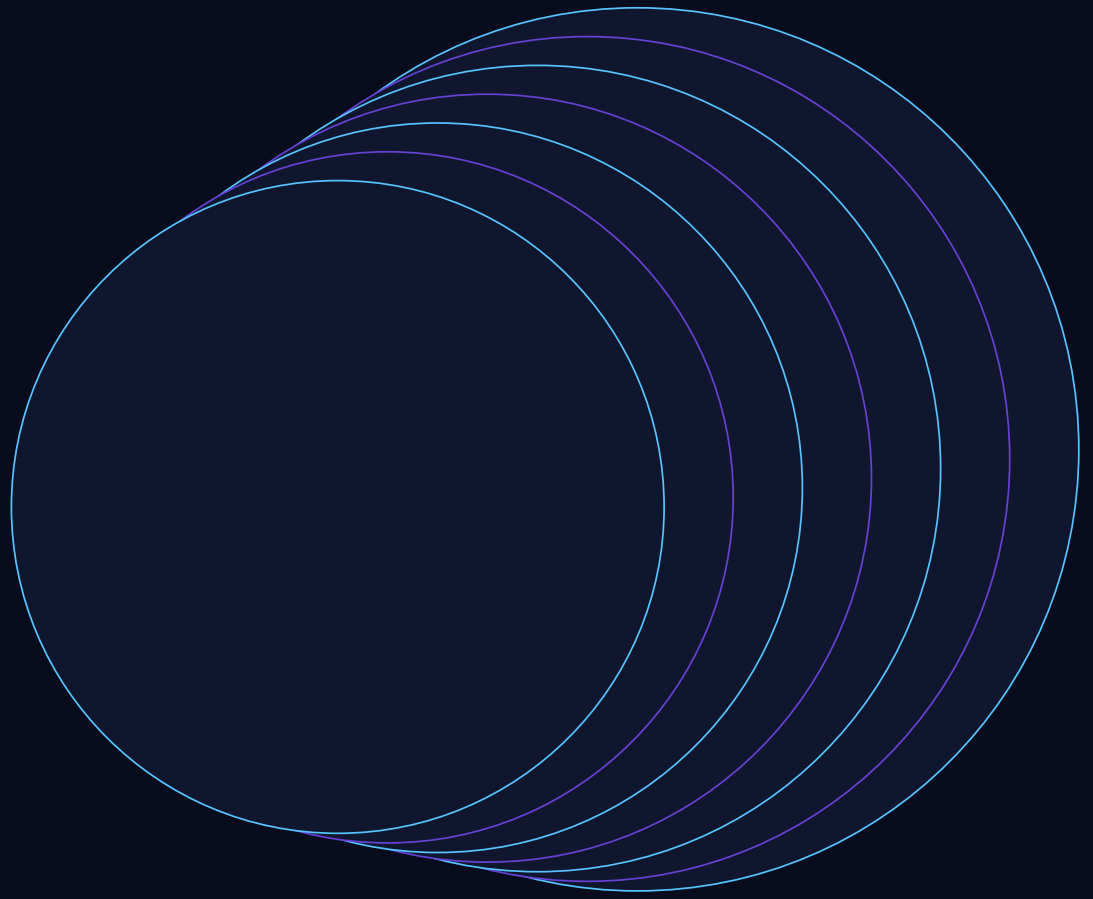
AI ≠ outsourced accountability

Teams still decide what “right” means and own the risk.

THE FUTURE OF SOFTWARE ENGINEERING

ISN'T ABOUT WHO WRITES THE CODE.

IT'S ABOUT WHO CAN PROVE IT DESERVES TO RUN.



AI wrote the code. You own the verdict.