20 Years of Frontend Lessons

Architecture decisions that actually matter

CYC '26 • Dallas • Sept 3–4, 2026

ABOUT

Sayeed Mohammad

Principal Frontend Architect • Ascendion

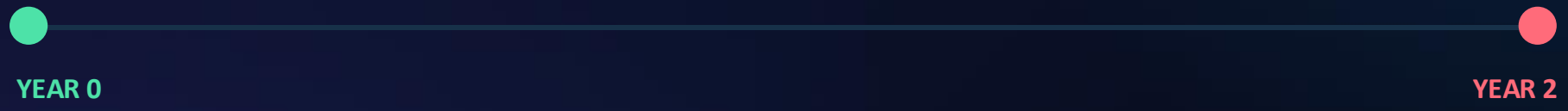
20+ years • Airlines • Healthcare • Financial services • Government

I help teams make

architecture decisions that keep change safe.

It began as a successful frontend.

Clear requirements. A small team. A clean architecture.



What could possibly go wrong?

Year 0: It started clean and simple.

One request path. Local ownership. A small blast radius.

0 YEARS	6 MONTHS	1 YEAR	2 YEARS

CHANGE COST — **Change was local. Confidence was high.**

Six months: Convenience began to spread.

Every shortcut felt reasonable. Together, they changed the shape of the frontend.

0 YEARS | **6 MONTHS** | 1 YEAR | 2 YEARS

CHANGE COST — **Shortcuts began to travel.**

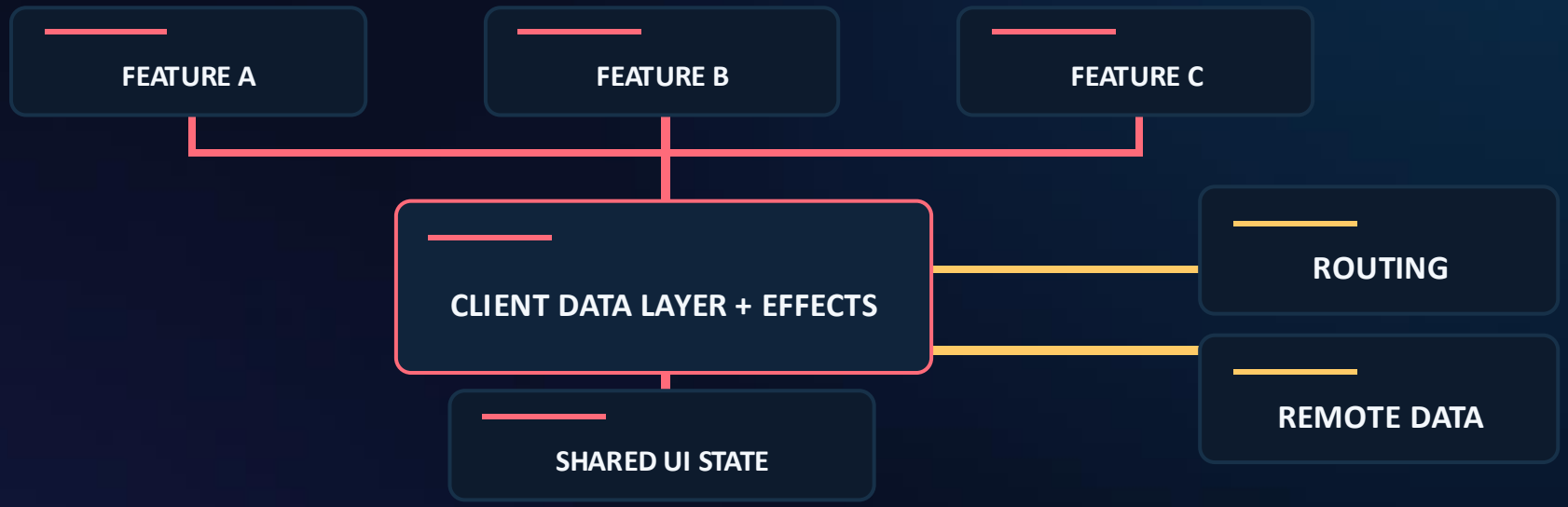
Year 1: Coordination dominated.

Every feature now crossed the same data and state path.

0 YEARS	6 MONTHS	1 YEAR	2 YEARS

CHANGE COST More features required shared coordination.

Year 2: The system still worked.



The app still worked. But every frontend change crossed the same shared path.

0 YEARS	6 MONTHS	1 YEAR	2 YEARS

THE REAL COST **Change had stopped being cheap.**

CREDIBILITY, FAST

Four industries. The same pressure.

FINANCE

Trust

HEALTHCARE

Uptime

AIRLINES

Journey

GOVERNMENT

Change

20+ YEARS

FRONTEND ARCHITECTURE

Building frontend systems where mistakes travel far.

YOUR TURN

Think of a system you were afraid to change.

01 What made change expensive?

02 Which decision would you revisit today?

YOUR SYSTEM **Keep it in mind. Use it to test the five judgment calls.**

AGENDA

Five judgment calls that outlast frameworks

01	**CHANGE**	Will this survive year two?
02	**ABSTRACTION**	Have we earned it?
03	**STATE**	Can it stay local?
04	**TRUST**	Who could be surprised?
05	**LEADERSHIP**	What can the team decide?

? **PICK ONE** — **Which one costs your team the most today?**

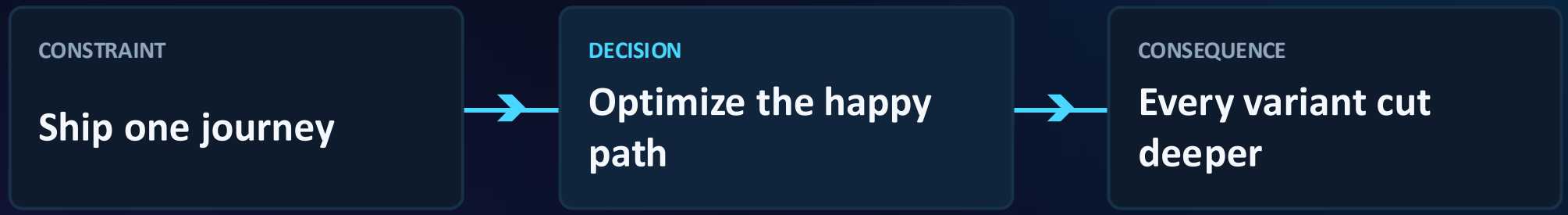
Launch day is the easy part.

FINANCIAL SERVICES

The team optimized one workflow for launch, not for the changes that followed.

The first version proved delivery. The next version tested the design.

The second feature revealed the design.



KEY POINT Clean code is not the same as cheap change.

THE DURABLE PRINCIPLE

Design for the next change.

Map likely change paths and keep their blast radius small, even when launch pressure is high.

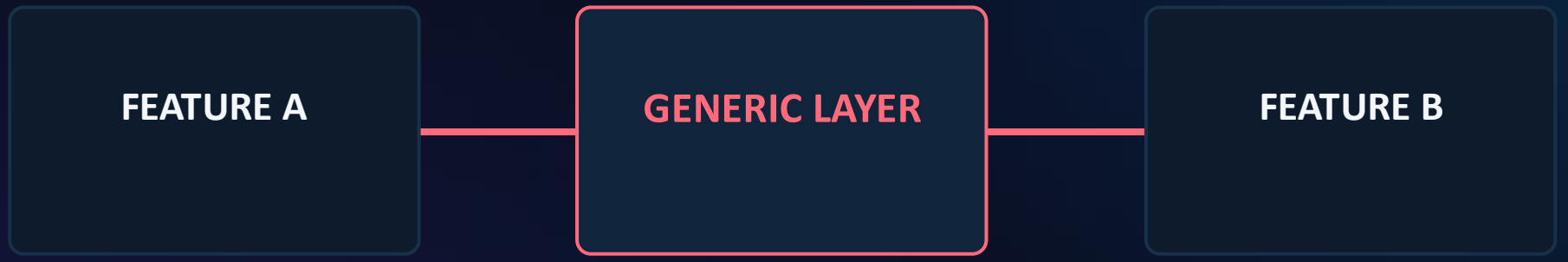
The abstraction looked perfect.

GOVERNMENT

Standardization pushed different workflows behind a generic layer.

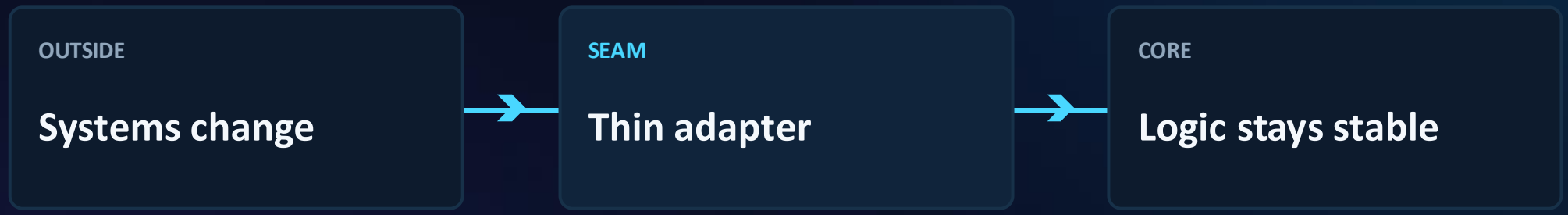
The diagram became simpler. The developer experience did not.

The abstraction became the architecture.



Every exception made the reusable layer harder to understand.

Abstract at the seam—not at the center.



The seam is where things change independently.

THE DURABLE PRINCIPLE

Wait for the pattern.
Then abstract at the seam.

Use real examples to discover what varies; place the abstraction where systems change independently.

Centralized state felt safer.

HEALTHCARE

Reliability concerns pushed transient UI state into a global model.

Centralized looked controlled. It also connected everything.

State should tell the truth.

LYING STATE

isLoading: true
isLoaded: true
hasError: true
errorMessage: null

HONEST STATE

```
status: 'idle' | 'loading'
      | 'success' | 'error'
```

KEY POINT

If the model allows nonsense, the UI will eventually produce it.

THE DURABLE PRINCIPLE

Keep state local and make it honest.

Promote it only for real coordination; model mutually exclusive states explicitly.

The API was right. The plan was not.

AIRLINES

Frontend, backend, and product optimized for different definitions of done.

Nobody failed their task. The journey still surprised everyone.

Earn trust before you set direction.

WITH BACKEND

Learn their constraints

WITH PRODUCT

Translate risk into delivery impact

WITH YOUR TEAM

Prototype. Review generously.

KEY POINT

Good direction needs trust, not just authority.

THE DURABLE PRINCIPLE

Trust grows when surprises shrink.

Share risks and tradeoffs before implementation locks them in.

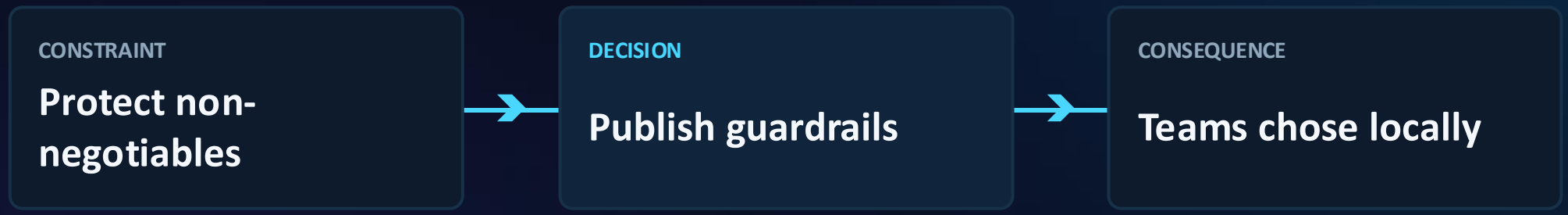
The standard became a mandate.

REGULATED PLATFORMS

A technically sound direction failed because the team never owned it.

Compliance increased. Commitment disappeared.

Guardrails beat approvals.



KEY POINT Direction without autonomy becomes dependency.

THE DURABLE PRINCIPLE

Lead with constraints, not control.

Make the why and non-negotiables clear; let the team shape the how.

TAKEAWAY • USE IN YOUR NEXT REVIEW

Five questions for every architecture review

CHANGE — What gets harder in year two?

ABSTRACTION — What pattern have we observed?

STATE — Can this stay local?

TRUST — Who could be surprised?

LEADERSHIP — What can the team decide?

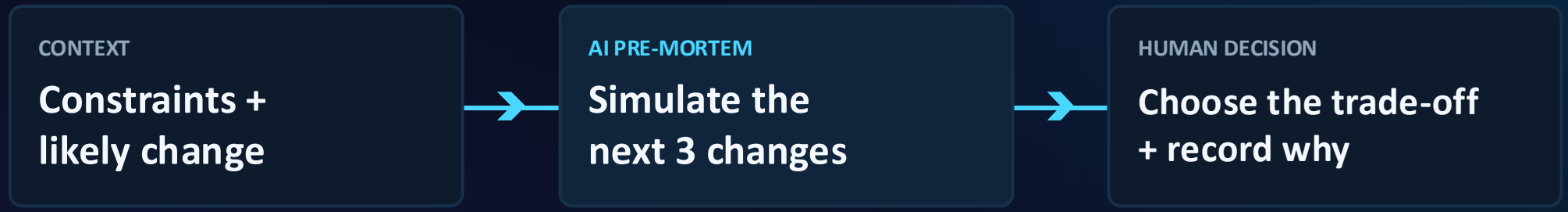
AI can accelerate the audit.

Judgment still owns the decision.

Use AI to reveal risks, challenge assumptions, and trace complexity—then let people decide the trade-offs.

Use AI before architecture decisions harden.

NEW PROJECT • INITIAL PHASE



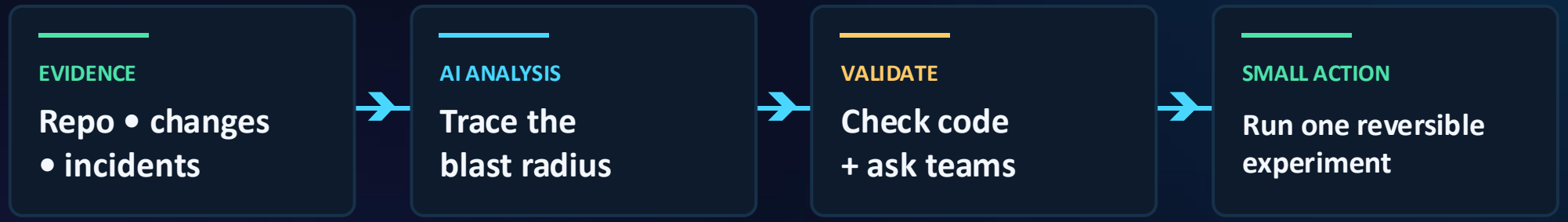
Use AI to expose questions—not to approve architecture.

Test every finding against change • abstraction • state • trust • leadership

AI-ASSISTED AUDIT

Audit change friction—not just code quality.

EXISTING SYSTEM



AI proposes hypotheses. Teams validate them with evidence.

Approved tools only • No secrets or PII • Evidence required

Architecture is

the cost of change.

Code change × coordination × confidence

CALL TO ACTION

Choose one question for Monday.

CHANGE • ABSTRACTION • STATE • TRUST • LEADERSHIP

Add it to your next design review.

Build for your future team.

Thank you. Questions?

QUESTIONS LATER?

sayeedmd05@gmail.com

More frontend lessons: sayeed.in