

MCP Dev Summit Nairobi



Payment Intents, Not Payment Calls

An execution layer for autonomous agent transactions

Chris Ibiam | AI/ML Engineer, Kora

Jime Akpusugh | Head of Innovation, Kora

The agent was right. The airline was down. The money had already moved.

What should the system do next?



Why a payment API call isn't enough for an agent

1. **Retries**
   A timed-out call is ambiguous. Did the money move? A blind retry can pay twice.

2. **Partial state**
   The booking is confirmed but the payment is unknown, or the reverse.

3. **Humans mid-flow**
   A PIN or approval arrives minutes later, outside the model's context.

4. **A confident wrong call**
   The model picks the wrong amount or payee, and nothing sits between it and the rail.

Decide, authorize, execute, settle: four separate steps

1

Decide

LLM dispatcher

Plans the tool calls and proposes an intent. It cannot move money.

2

Authorize

Human and policy

PIN approval or a spending limit. Outside the model.

3

Execute

Adapters

Reserve the airline hold first, then take the payment.

4

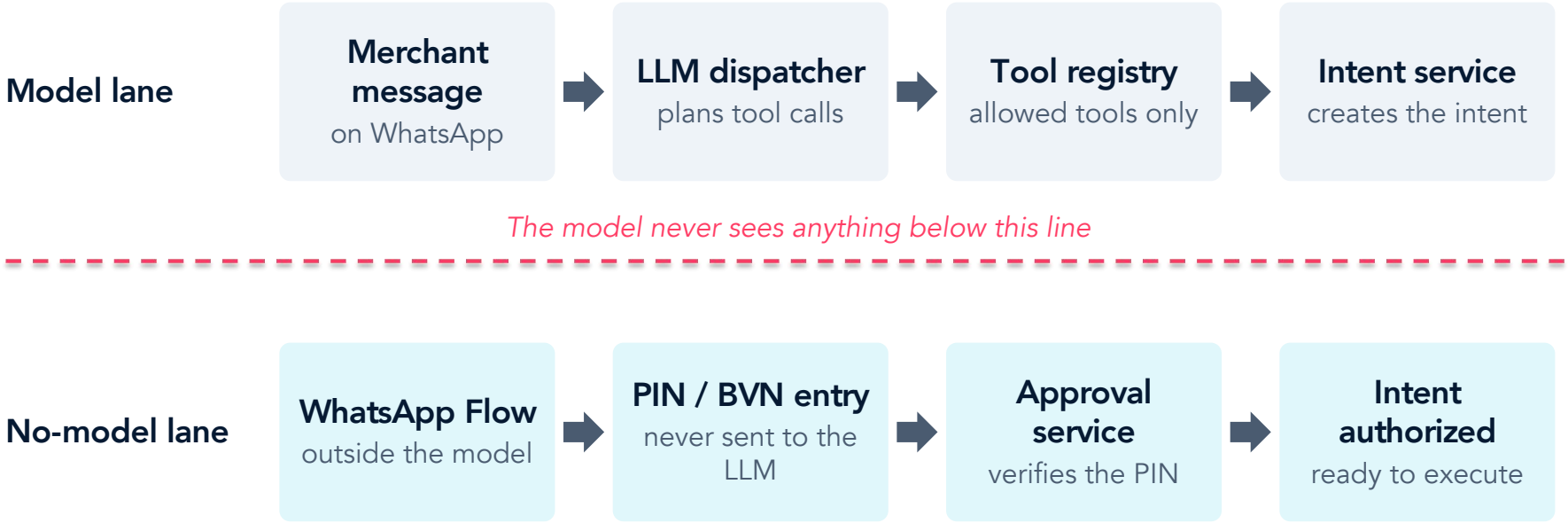
Settle

Status poller

Confirm against the rail's state, not the call's return value.

One payment intent carries its state through all four.

The LLM plans. It never touches the PIN.



The boundary is enforced by routing, not by a prompt instruction.

Scoped tokens and spending limits bound the agent

1 Scoped payment token

- Scoped to one merchant, amount range, rail and expiry
- Signed and tamper-evident
- No raw credentials leave the agent boundary
- Prepared and reserved before any PIN exists

2 Policy-bounded autonomy

- Agent wallets carry spending limits and approval rules
- Inside the bounds, the agent acts on its own
- Outside the bounds, a human approves

Book first, settle second: failures stay recoverable

Reserve
airline hold



Settle
money moves



Confirm
booking

Settlement fails before money moves

The hold is cancelled automatically. Nothing to unwind.

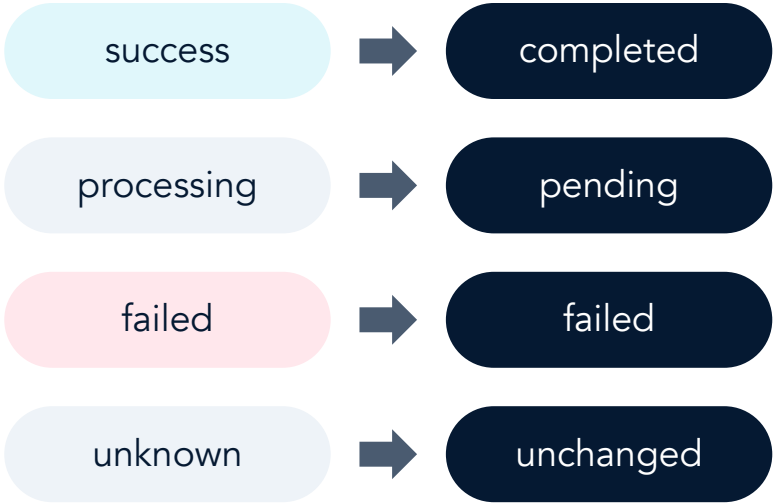
Booking fails after settlement

Goes to manual review. No automatic refund.

Auto-refunding on ambiguous state is how you pay twice.

Settlement truth comes from the rail, not the call

Rail status becomes intent status



Why not trust the call?

- A timeout says nothing about the rail.
- A poller re-reads rail state on a schedule.
- An unknown status keeps the last known state. We never guess.
- If the status API fails, we fall back to our stored state.

Demo: Lagos to Abuja, then a failure we cause on purpose

Mock airline adapter on Kora test rails and no real money.

What you'll see in the demo

1. Ask in plain language: Lagos to Abuja, Friday
2. Pick a flight and add passenger details
3. Approve with a PIN. The model is not in this step
4. Hold, settle, confirm
5. The ticket confirmation arrives on WhatsApp

Mock airline adapter on Kora test rails and no real money

Then the failure run

We force the airline adapter to fail at settlement.

The hold is cancelled and the user gets a clear message.

What we got wrong, and the patterns that travel

What we got wrong

We built our own PIN-confirmation service.

Then we found the approval flow we already had, and deleted ours.

Two implementations of one security path is a bug waiting to happen.

Patterns that travel beyond payments

1. Separate deciding, authorising and executing
2. Keep the model away from secrets
3. Reserve before you spend
4. Trust the rail's state, not the call's return

Where this stands today

WORKING IN CODE

- Payment intent state machine
- Scoped payment tokens
- Policy-bounded agent wallets
- Settlement status poller

TESTED SO FAR

- Mock airline adapter
- Kora test rails
- Actively speaking to and integrating with two real merchants

NEXT

- Merchant integrations.
- Interoperability with ACP and AP2





MCP
Dev Summit
Nairobi

Questions?

Chris Ibiam | AI/ML Engineer, Kora