Stephane Derosiaux

CTO
Conduktor

Event-Driven Memory for LLM Agent Swarms



DSS Data Streaming Summit

Hosted by StreamNative



Stephane Derosiaux

CTO

Conduktor

Co-founder of Conduktor, the Kafka operations platform.
On Kafka since 0.8, in 2015.
Adtech and retail tech aficionado.



business domain

agentic domain

support chat
customers write here

business events

customer.messages

reacts

orchestrator
consumer group

decision model

agent topics

agent.tasks.*

agent.results

agent.state

agents harness + LLM

shipping agents

stock agents

refunds agents

memory.events

recall
embedder · vector DB

MCP server

proposal = command to the owner

order service
decides, publishes

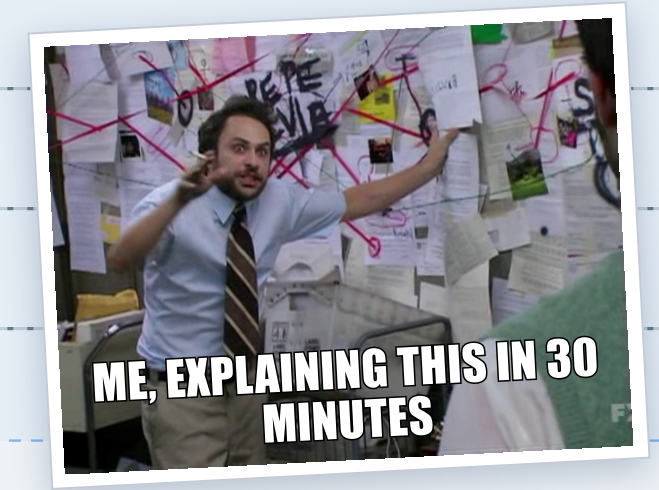
order.events

carrier webhook

carrier.events

warehouse webhook

warehouse.events

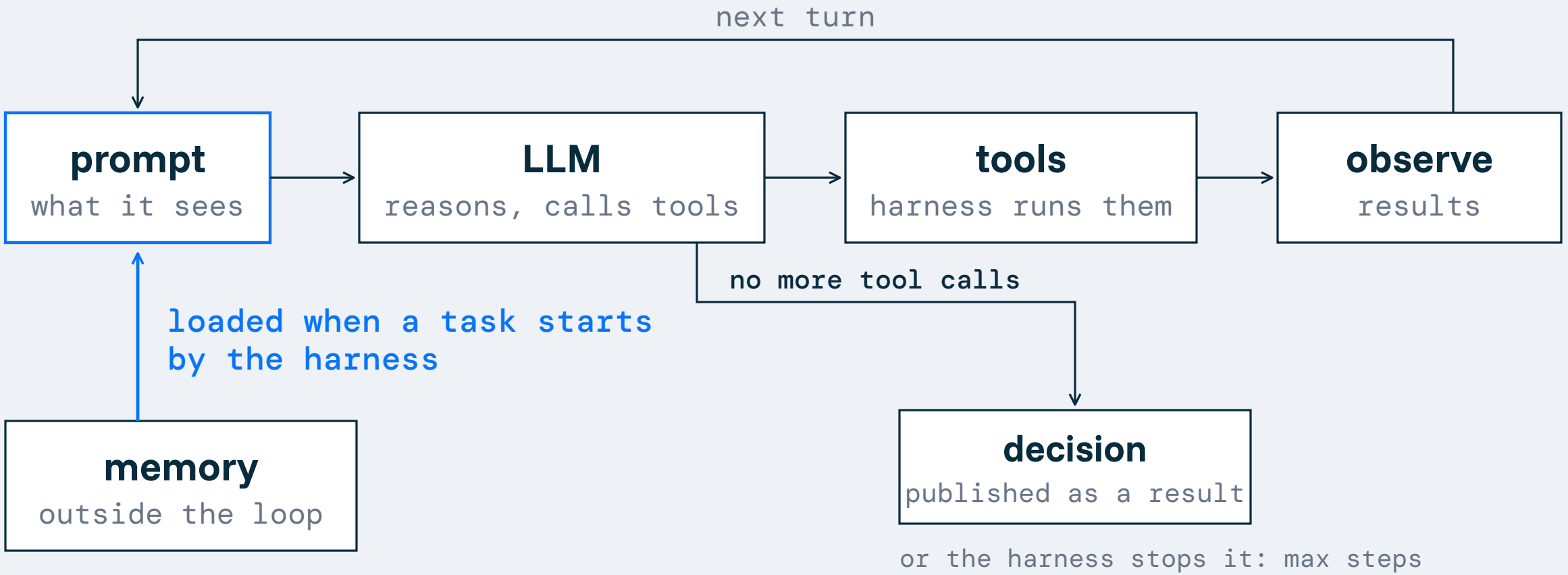


order-context
Kafka Streams + IQ

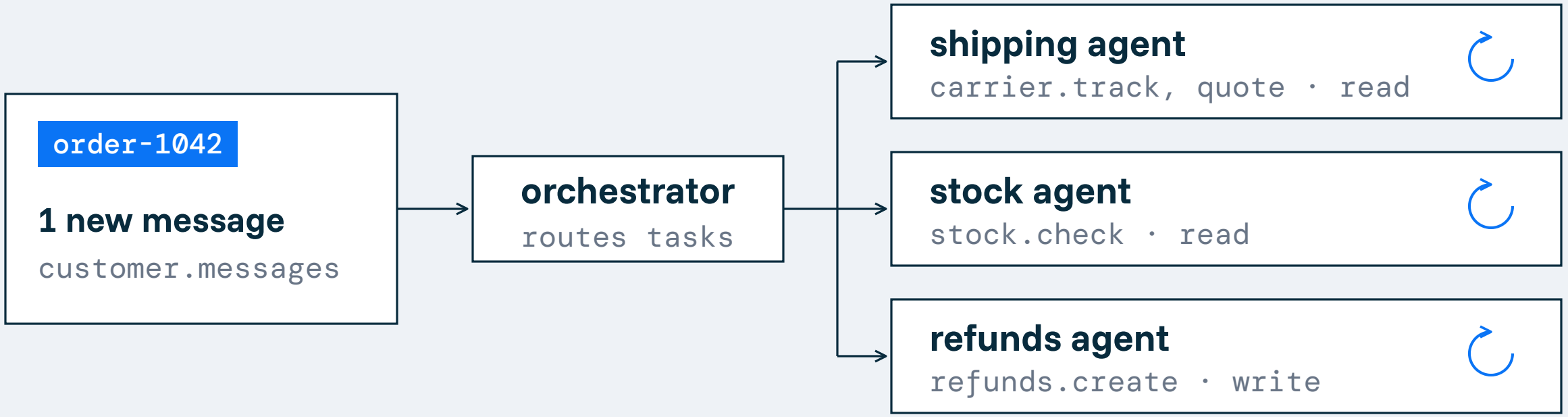
MCP server

state store

An agent is a loop.
Memory lives outside it.



An agent is a loop.
A swarm is a topology.



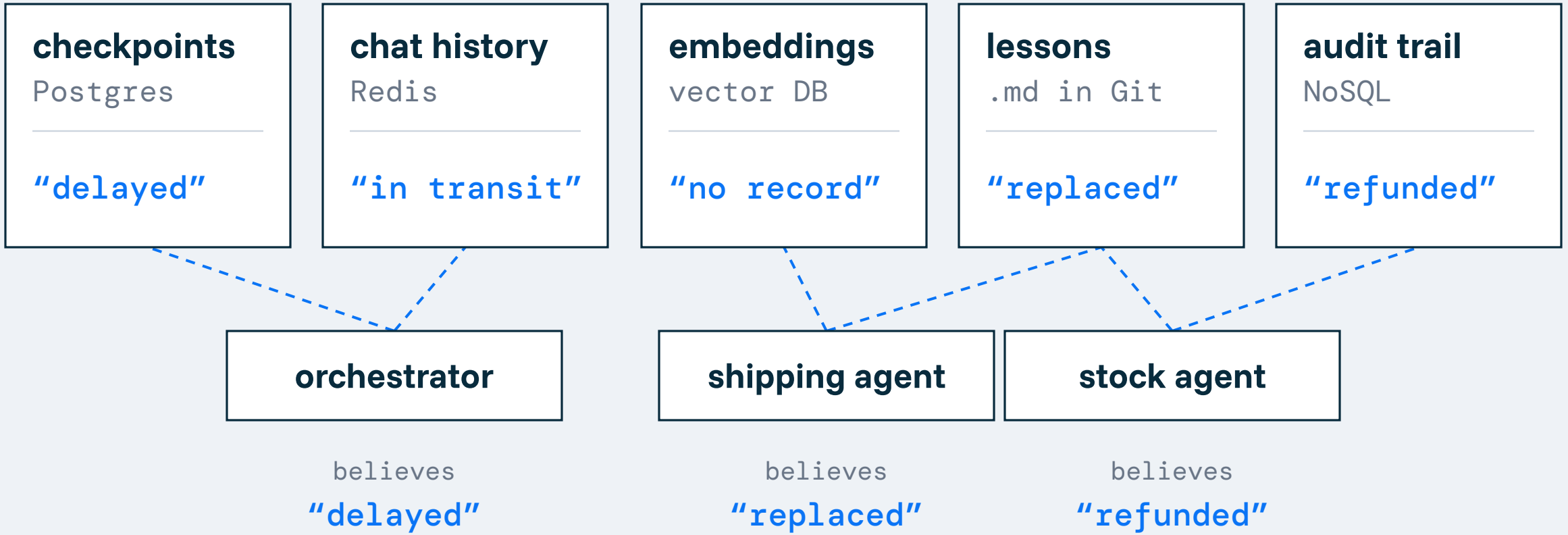
why split | isolate context | run in parallel | least privilege | right-size models

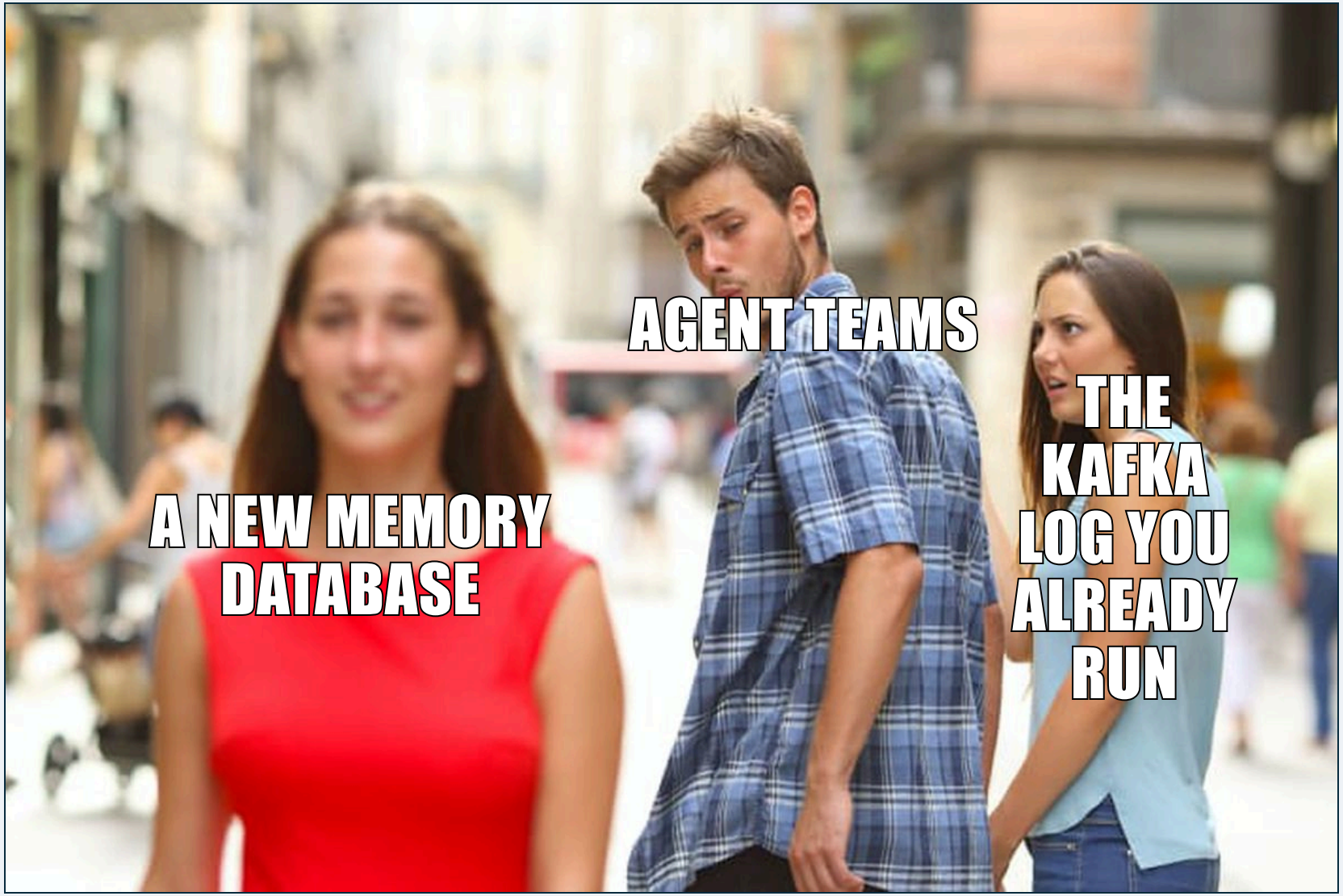
switch models with one line: OpenRouter or LiteLLM


WE ARE THE BORG.
RESISTANCE IS FUTILE.

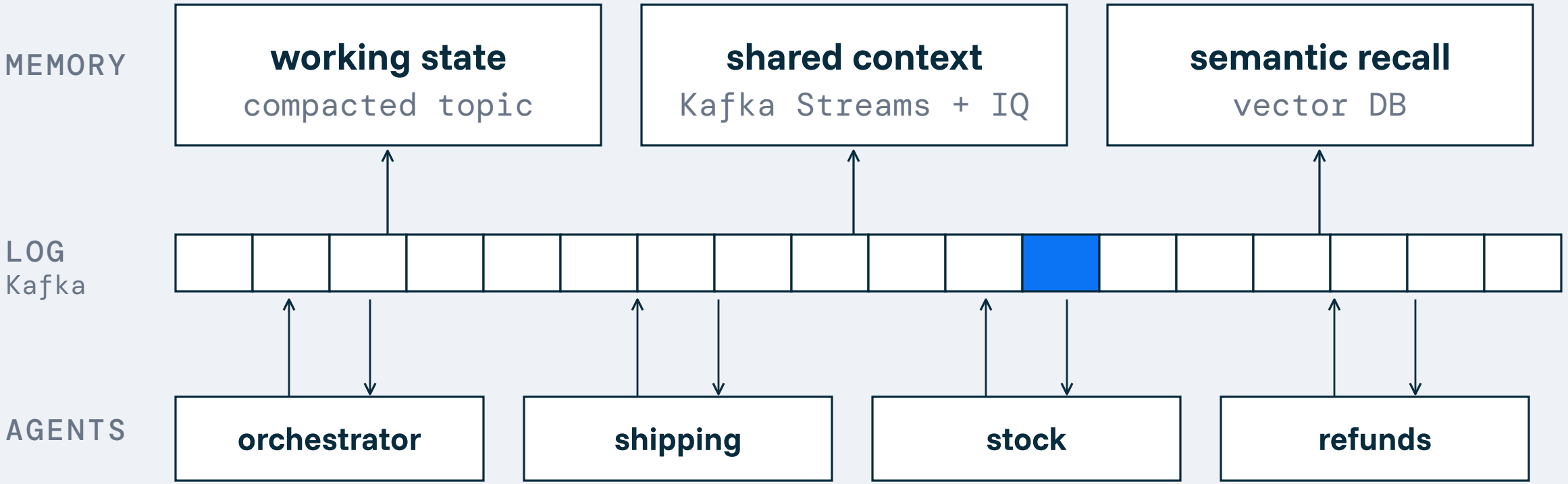
Where does your swarm remember?

Ask each store: what is the status of order-1042?

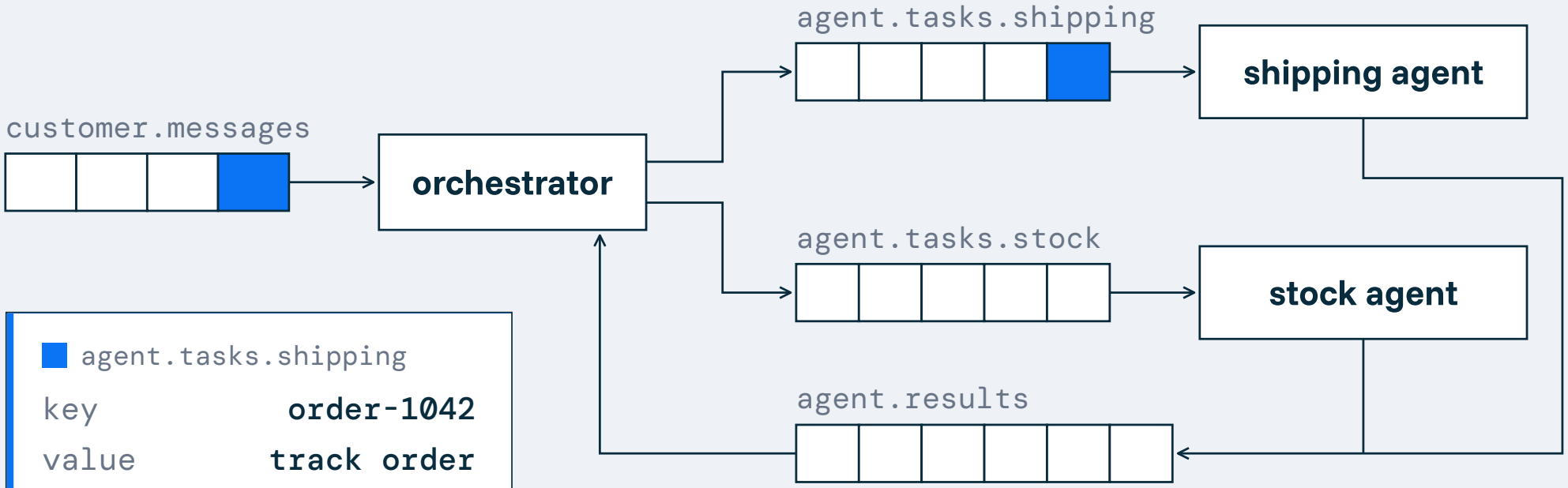



AGENT TEAMS
A NEW MEMORY DATABASE
THE KAFKA LOG YOU ALREADY RUN

Turning agent memory inside out



Agents don't call each other. They publish records.



agent.tasks.shipping

key	order-1042
value	track order

HEADERS

correlation_id	run-7f3
causation_id	msg-881
task_id	t-2
traceparent	00-4bf9…-01

same pattern, other tracks

transport	Pulsar · Redis Streams · NATS JetStream · SQS
orchestration	Restate · LittleHorse · Temporal · Flink Agents

Routing doesn't need a generative model

the orchestrator asks: what next?

```
POST /v1/systemone
{
  "model": "jev-latest",
  "state": { "order": "order-1042",
    "carrier": "delayed, ETA Mon",
    "promised": "Fri", "stock_sf2": 3 },
  "questions": { "next_step": {
    "type": "choice",
    "instructions": "Next step for this order",
    "criteria": {
      "stock": "Check stock for a replacement",
      "refunds": "Refund the customer",
      "reply": "Answer the customer" }}}
}
```

a generic decision model: one choice

```
200 OK
{
  "model": "jev-1.13.0",
  "answers": { "next_step": {
    "type": "choice",
    "choice": "stock",
    "confidence": 0.77,
    "probabilities": {
      "stock": 0.84,
      "refunds": 0.11,
      "reply": 0.05 } } }
}
```

decision models: **TypeSafe Jev (can use OpenRouter) · Cloudflare Clef · OpenAI Decisions API**

make it yours: **fine-tune on your past decisions, straight from the log (LoRA, e.g. Unsloth)**


ONE DOES NOT SIMPLY
BLOCK A KAFKA PARTITION ON AN LLM CALL.


SHARE GROUPS
A.K.A. QUEUES FOR KAFKA

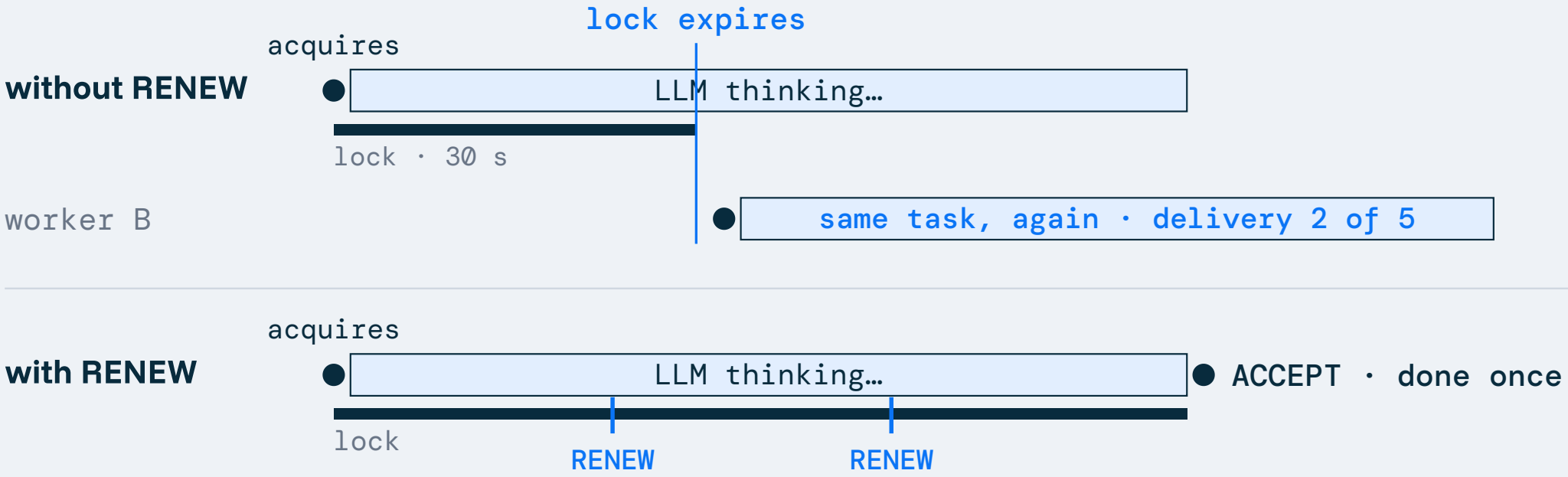
LLM workers are a queue

LLM inference · seconds to minutes
a slow call must not block a partition

share group · Queues for Kafka
unordered · Java only for now

```
var worker = new KafkaShareConsumer<String, String>(props);
worker.subscribe(List.of("agent.tasks.shipping"));
while (running) {
  for (var task : worker.poll(Duration.ofSeconds(1))) {
    var ok = runShippingAgent(task);   // → agent.results
    worker.acknowledge(task, ok ? ACCEPT : RELEASE);
  }
  worker.commitSync();
}
```

A slow model gets its task stolen




```
share.acknowledgement.mode=explicit      # consumer: ACCEPT, RELEASE, RENEW by hand
share.record.lock.duration.ms=30000      # 15-60 s: a dead worker holds it that long
share.delivery.count.limit=5             # each expiry costs one, then archived
share.renew.acknowledge.enable=true      # allows RENEW (Kafka 4.3); nothing renews by itself
```

The order is already in the log

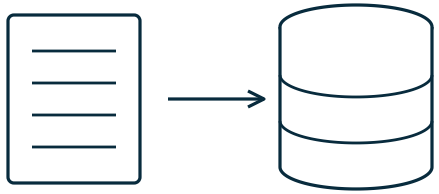
every record keyed **order-1042**		written by
carrier.events	delayed, ETA Monday	carrier webhook
customer.messages	**"where is my order?"**	support chat
agent.tasks.shipping	**track order**	orchestrator
agent.results	**carrier.track → delayed, ETA Monday**	shipping agent
warehouse.events	SF-2: 4 → 3 (another order)	warehouse webhook
agent.tasks.stock	**find a replacement near 94107**	orchestrator
agent.results	**stock.check → SF-2: 3 in stock**	stock agent
agent.results	**proposal: ship replacement, express**	shipping agent
order.events	**approved: replacement, Thursday**	order service

also a dataset: **cluster outcomes, spot business signals, fine-tune a cheaper model (e.g. Unsloth)**

Three questions, three memories

Where was I?

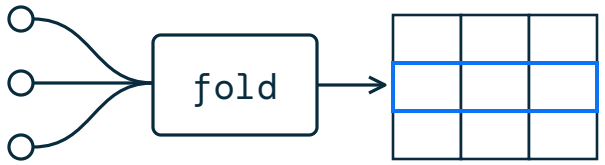
compacted topic



private · role|order

What do we know?

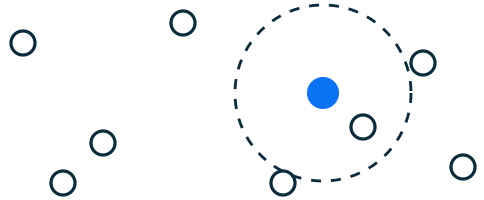
Kafka Streams + IQ



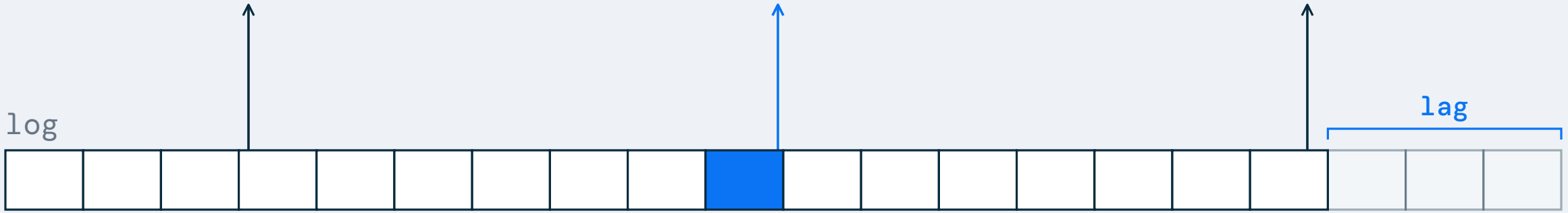
shared · per order

Just this order?

vector DB



long-term · all orders



⚠ eventually consistent

Where was I? Key it by role and order id




```
// compacted: the latest checkpoint per key is the state
producer.send(new ProducerRecord<>("agent.state",
    "shipping|order-1042",
    json(new Checkpoint(3, "carrier.track", "1Z84X7"))));
builder.globalTable("agent.state",  // a copy on every worker
    Materialized.as("agent-state"));
ReadOnlyKeyValueStore<String, String> store = streams.store(
    StoreQueryParameters.fromNameAndType("agent-state",
        QueryableStoreTypes.keyValueStore()));
var checkpoint = store.get("shipping|order-1042");
```

What do we know? Materialize it once




```
builder.<String, String>stream(List.of(  // key: order id
        "carrier.events", "warehouse.events", "order.events"))
    .groupByKey()
    .aggregate(() -> "{}",
        (orderId, fact, ctx) -> merge(ctx, fact), // + version
        Materialized.as("order-context"));
ReadOnlyKeyValueStore<String, String> store = streams.store(
    StoreQueryParameters.fromNameAndType("order-context",
        QueryableStoreTypes.keyValueStore()));
var ctx = store.get("order-1042");  // GET /orders/{id}
```

alternatives: PostgreSQL, Flink SQL, … still built from the log.

Memory is a tool call

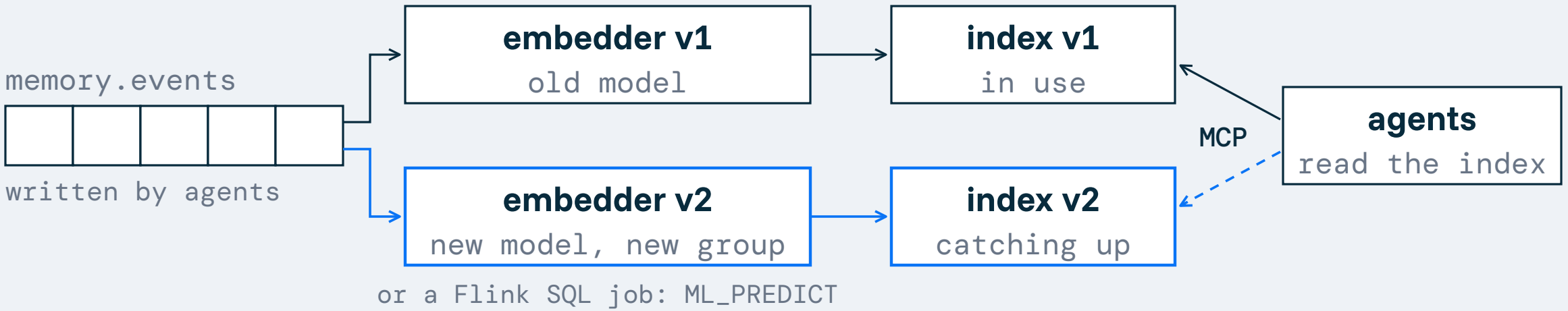
MCP: how an agent discovers and calls tools

```
McpServer.sync(new StdioServerTransportProvider(mapper))
  .serverInfo("order-context", "1.0.0")
  .toolCall(Tool.builder("get_order_context", ORDER_ID)
    .description("Live status, carrier, ETA, promised date "
      + "and stock of one order.")
    .build(),
    (exchange, call) -> httpGet("/orders/{id}",
      call.arguments().get("order_id")))
  .build();
```

get_order_context("order-1042") returns

```
{"status":"delayed","carrier":"UPS","eta":"Mon","promised":"Fri","stock_sf2":3,"version":4}
```

Zoom out to the big picture



Query the vector DB through MCP

find_similar_cases(**text**, order_id) · Qdrant

```
var scope = Filter.newBuilder()
  .addMustNot(matchKeyword("order_id", orderId)) // not itself
  // recent beats similar
  .addMust(datetimeRange("ts", since(Duration.ofHours(12))))
  .build();
var hits = nearest3(scope, text);  // top 3 by meaning
```

alternative: pgvector

```
-- $2 = embed(text)
SELECT order_id, text FROM memories
WHERE order_id <> $1
  AND ts > now() - interval '12 hours'
ORDER BY embedding <=> $2 LIMIT 3;
```

hits for **text** = "UPS shipment delayed at the hub, new ETA after the promised date"

```
order-0871   0.821   4 h ago   missed hub scan; reshipped next day air
order-1027   0.815   1 h ago   stuck at the Oakland hub; reship offered
order-1019   0.714   6 h ago   held at the Oakland hub; reshipped next day air
```

what the agent does with it

```
not isolated: 3 orders stuck at the hub today. Reship from SF-2, flag the hub
```

This is what the model sees

```
## instructions
You are the shipping agent. Decide the next step.

## where you were
step 4 of 5 · carrier tracked, stock found

## what we know about order-1042
delayed · UPS · ETA Mon · promised Fri · SF-2: 3

## the big picture
3 orders stuck at the Oakland hub today

## your task
quote a faster shipment
```

← **its mission**
Git

← **its private state**
agent.state topic

← **shared context**
Kafka Streams + IQ

← **memory of all orders**
vector DB

← **the job right now**
orchestrator



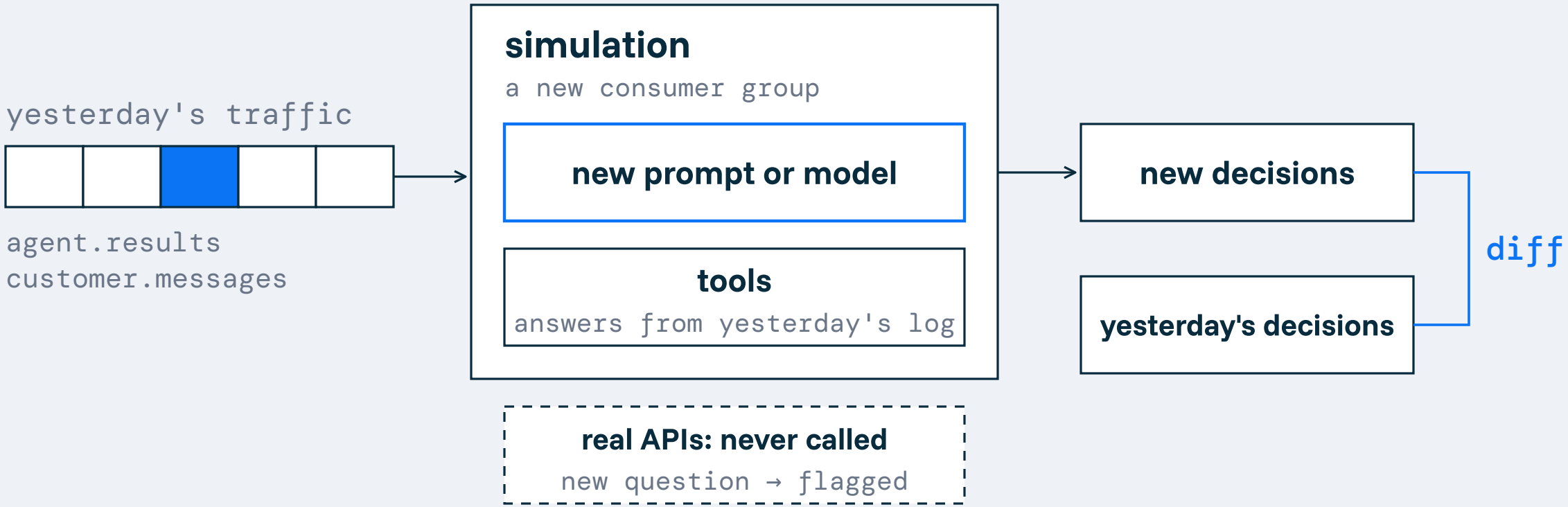
Testing a new prompt on your customers



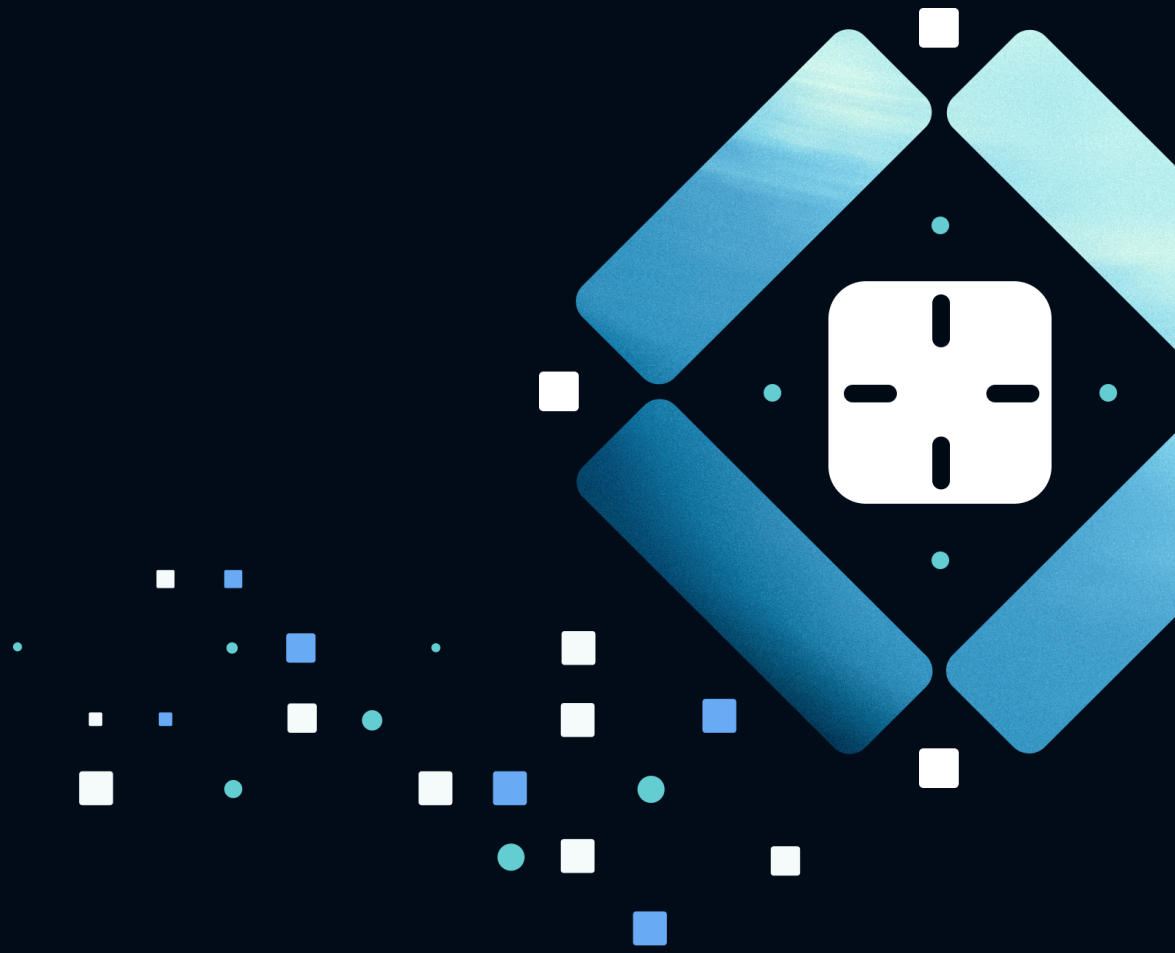
Testing it on yesterday's traffic

Simulate a new prompt or model on yesterday's traffic

a.k.a. **backtesting · offline evaluation · regression tests for prompts**



Questions?



Thank you

Stephane Derosiaux

CTO
Conduktor

https://www.linkedin.com/in/stephane-derosiaux

Hosted by StreamNative

DSS '26

San Francisco, CA | October 7 - 8

DSS Data Streaming Summit