Kafka as the Agent Action Log



Ron Kapoor · Current San Francisco 2026



Ron Kapoor

Technical product marketing at Conduktor

Before that: electronic trading systems at a bank (lots of tick data!), OLAP databases at CelerData.

This isn’t a product talk!

Raise your hand if a consumer ever processed the same message twice.

Keep it up if that behavior was expected.

Crash before commit, and Kafka delivers it again. Usually, that's fine.

the loop almost everyone writes

```
while True:
    msg = consumer.poll()
    process(msg)
    consumer.commit(msg)
```

✗ crash here
commit never runs, so the message comes back

FINE WHEN process() DOES THE SAME THING TWICE:

```
db.upsert(order_id, row)
```

```
index.put(doc_id, doc)
```

```
charge(card,
idempotency_key=id)
```

All fine, because the same input gives the **same answer**.

Where you commit decides what a crash costs

AT-MOST-ONCE

```
consumer.commit(msg)
process(msg)  # ✗ crash
```

✗ the message is lost

Often: metrics and telemetry, where the next reading replaces a lost one

AT-LEAST-ONCE · KAFKA'S DEFAULT

```
process(msg)  # ✗ crash
consumer.commit(msg)
```

↺ it runs twice

Often: most pipelines (orders, CDC, search indexing), with upserts or dedupe

EXACTLY-ONCE · p = producer

```
p.begin_transaction()
p.produce(output)
p.send_offsets_to_transaction()
p.commit_transaction()
```

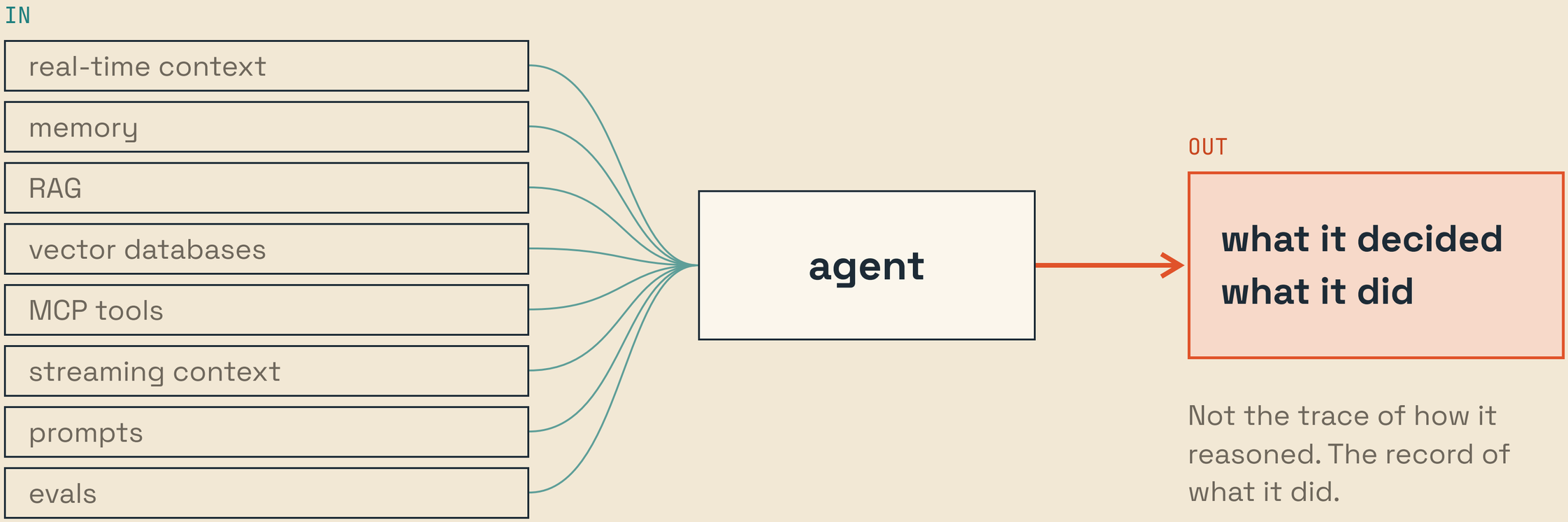
✓ all or nothing

Often: stream processing that counts or sums, like balances and billing

Kafka writes only. Into a DB: store the offset in the same transaction.

A refund API has no transaction to join. In practice, side effects run at-least-once.

We talk a lot about what goes into agents. Today, let's look at what comes out.



Who's building an agent? Where does it run?

1

A workflow engine

`Temporal · Restate · Inngest`

The engine records every step and resumes from it.

`this is durable execution`

2

Inside Flink

`Flink Agents · Streaming Agents`

The runtime helps. Flink Agents records completed actions.

3

Your own consumer

`reads a message, calls a model`

Nothing records the decision unless you do.

`share groups: GA in Kafka 4.2`

So what happens when the thing processing your message is an AI, and Kafka retries it?



“That’s a deal, and that’s a legally binding offer - no takesies backsies.”

An agent is a loop: the model decides, your code acts

your code, around the model — = the harness

```
def run_agent(ticket):
    history = [ticket]
    while len(history) < MAX_STEPS:     ← and enforces limits
        step = model.decide(history)    ← model picks a step
        result = run_tool(step)         ← your code runs it
        history.append(result)          ← result goes back
```

TICKET #4521

“My blender arrived broken.”

1 decide → lookup_order("A-7731")
run_tool → delivered Tue · $40

2 decide → refund("A-7731", 40)
run_tool → ok · refund r_981

3 decide → send_email(customer)
run_tool → sent

every step can be an event, written ticket:step. step 2 of ticket 4521 is 4521:2.

4521:1 · lookup_order	**4521:2 · refund**	4521:3 · send_email	

Same ticket, asked twice. Two different answers.

terminal · illustration · same ticket, same prompt

```
$ python decide.py ticket-4521.json --temperature 0
refund 40

$ python decide.py ticket-4521.json --temperature 0
send_replacement
```

TICKET #4521

“My blender arrived broken.”

order A-7731 · $40

A retry may not repeat the work. It can change it.

A crash before the commit

worker A · consumer group

```
while True:
    msg = consumer.poll()
    decision = model.decide(msg)
    execute(decision)        # refund $40          ← ✗ crash right after the refund
    consumer.commit(msg)                           ← never runs
```

worker B · Kafka redelivered #4521

```
decision = model.decide(msg)   # asked again, may answer differently
```

Exactly-once aborts the transaction

worker A · exactly-once

```
while True:
    msg = consumer.poll()
    producer.begin_transaction()
    decision = model.decide(msg)
    execute(decision)        # refund $40          ← refund goes out: not a Kafka write
    producer.send_offsets_to_transaction(...)
    producer.commit_transaction()                  ← ↺ aborts: only Kafka writes undone
```

#4521 is processed again

```
decision = model.decide(msg)   # asked again, may answer differently
```

The model is slower than the lock

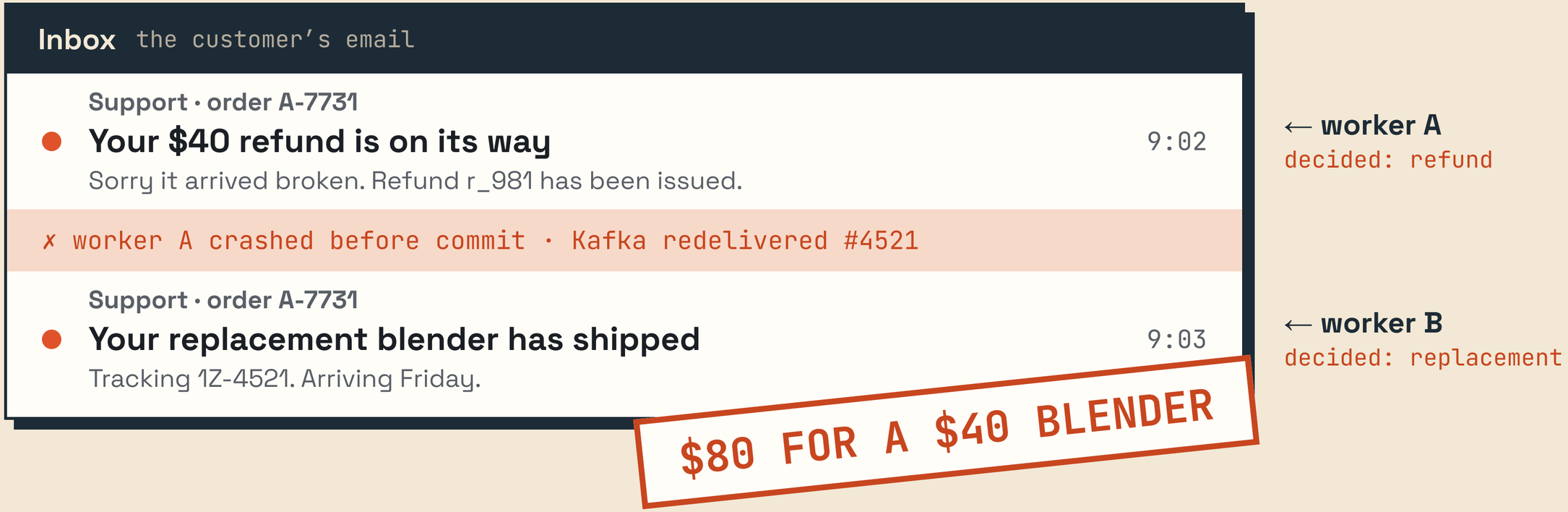
worker A · share group

```
while True:
    msg = share_consumer.poll()                    ← 30 s lock starts (the default)
    decision = model.decide(msg)                   ← ◷ still thinking at 30 s: lock runs out
    execute(decision)        # refund $40
    share_consumer.acknowledge(msg)
```

worker B · got #4521 while A is still thinking

```
decision = model.decide(msg)   # asked again, may answer differently
```

A crash can make the agent decide twice



An idempotency key on the refund wouldn't help: **the second action wasn't a refund.**

On their own, the obvious fixes break

TEMPERATURE 0

```
decision = model.decide(ticket, temperature=0)
# ✗ still not guaranteed; changes with updates
```

CHECK IF IT ALREADY HAPPENED

```
if not shipping.sent(order): ship(order)
# ✗ B checks for its own answer, not A's
```

LOG WHAT THE AGENT DID

```
execute(decision); log.write(decision)
# ✗ crash in between: nothing to read
```

COMMIT BEFORE ACTING

```
consumer.commit(msg); execute(decision)
# ✗ crash after the commit: ticket lost
```

Those four lessons are the core of durable execution

1

fixes ✗ log it after

Record the decision first

```
decisions.write(…)
```

and wait for the ack

2

fixes ✗ check action

Give every step an ID

```
"4521:2"
```

ticket 4521 · step 2

3

fixes ✗ temperature 0

Read it back on a retry

```
decisions.get(…)
```

keep the answer you got

4

fixes ✗ commit first

Make the action safe to repeat

```
idempotency_key=
```

the tool’s job

Temporal, Restate and Flink Agents build on these same ideas, under different names.

Write the decision down before you act

white: yours · orange: the fix

```
def run_step(ticket, step):
    decision = decisions.get(ticket.id, step)
    if decision is None:
        decision = model.decide(ticket)
        # returns only after Kafka acks
        decisions.write(ticket.id, step, decision)
    execute(decision, idempotency_key=f"{ticket.id}:{step}")
```

1 · called for each step
here: ticket 4521, step 2

2 · look up 4521:2
1st time: nothing · retry: “refund”

3 · nothing recorded? ask
a retry skips the model here

4 · the only model call
it says: “refund”

5 · write 4521:2 = refund
to agent.decisions, wait for the ack

6 · only now: the refund
key 4521:2 · repeat → “already done”

decisions = a small class, not a Kafka call
a table of ticket:step → decision, rebuilt from the topic at startup
get() reads it · write() appends to the topic, waits for the ack

Decide, write, wait for the ack.
Then act.

Same crash, with the decision log

Inbox the customer's email

Support · order A-7731

Your $40 refund is on its way 9:02

Sorry it arrived broken. Refund r_981 has been issued.

✗ worker A crashed before commit · Kafka redelivered #4521

No second email

4521:2 = refund (read, not re-asked) · refund system: already done

$40 FOR A $40 BLENDER

← **worker A**
wrote "refund" first

← **worker B**
read the log, didn't ask the model

1 refund · 0 repeated model calls

Think in parallel, act in order

share group

A's lock ran out at 30 s, so B got #4521 too

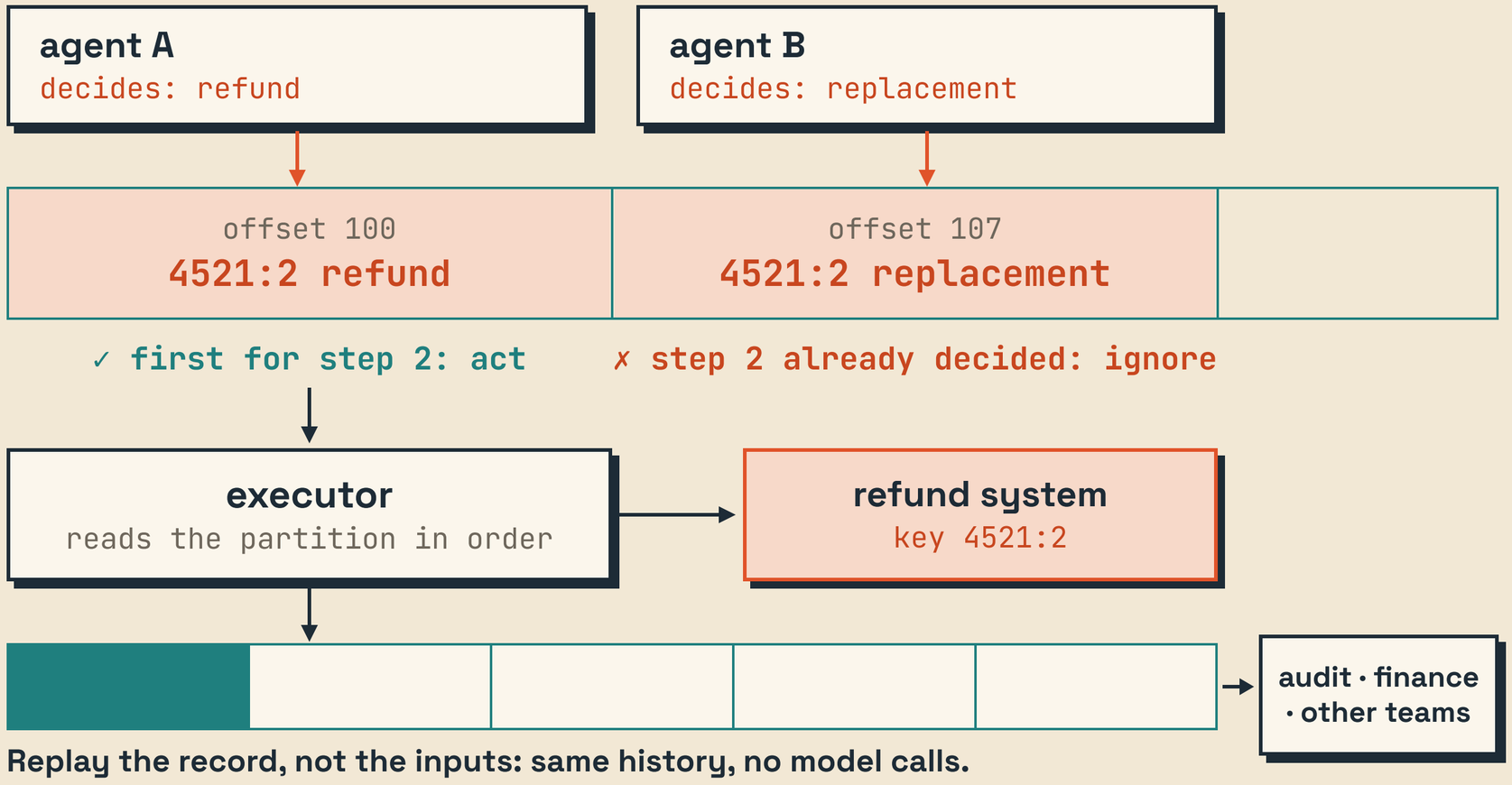
agent.decisions

partition 3: every record for ticket 4521

B gets no error: a topic has no write-if-absent

agent.results

the audit trail and the replay source



Replay the record, not the inputs: same history, no model calls.

The executor: first decision per step wins

executor · simplified

```
for msg in consume("agent.decisions", group="executor"):
    step_id = f"{msg.ticket}:{msg.step}"
    winner = winners.setdefault(step_id, msg.offset)
    if msg.offset > winner:
        continue
    result = execute(msg.decision, idempotency_key=step_id)
    produce("agent.results", key=msg.ticket, value=result)
    commit(msg)
```

← **1 · its own consumer group**
one executor reads 4521, in order

← **2 · first offset per step wins**
offset 100: 4521:2 refund

← **3 · later one, same step?**
offset 107: replacement, skipped

← **4 · act once, with the key**
then record it in agent.results

In production, winners is a state store, rebuilt from the topic after a restart.

Build the small version, or adopt a framework

Build it on Kafka when…

- A few steps, seconds long
- Triggered by Kafka events
- Tools take an idempotency key

Adopt a workflow engine when…

- Runs for hours or days
- Waits on a human
- Needs timers or rollbacks across steps

Either way, I'd put the record of what your agent did on Kafka!

Next steps

github.com/ronkapoor86

Thank you!