Slide to start the recording

Rider Pulse: Realtime Updates

Keeping rider state and backend truth aligned

We Removed Refresh — and Paid for It

- Manual refresh retired to avoid static-feeling UI.
- Smart polling (60s cadence) became the primary fallback.
- Backend request volume increased significantly.

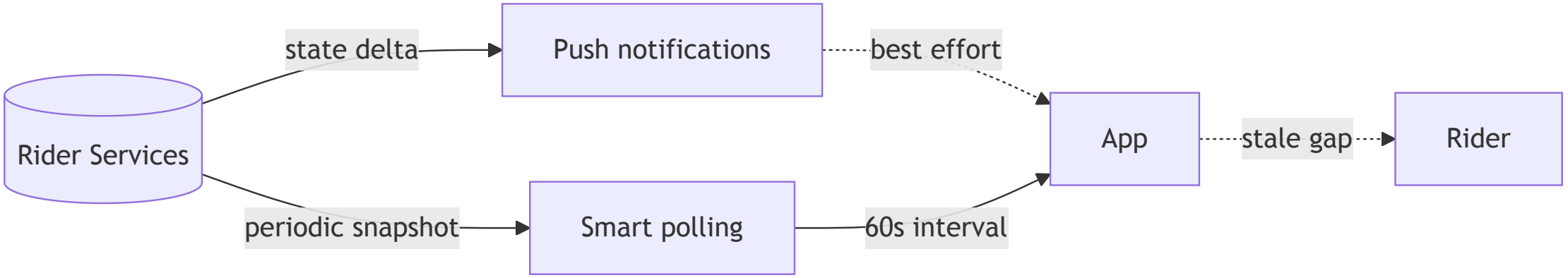
The Core Problem: Shared State Drift

- Distributed system with time-based UI updates.
- Worst-case UI lag of 60 seconds.
- Increased load without correctness guarantees.

Push Notifications Are Not a Consistency Mechanism

- Best-effort delivery and OS throttling.
- Limited scope and frequency by design.
- Missed push means waiting for the next poll.

Where Drift Happens Today



- Push and polling operate independently.
- Either channel can silently fail.

Current State: Works, But Scales Poorly

- Push plus polling achieves eventual parity.
- High backend load and persistent drift windows.

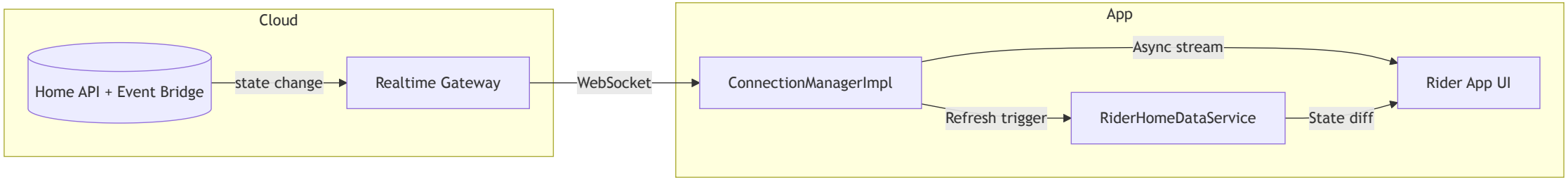
Introducing Rider Pulse

- A single, persistent connection for rider state signals.
- Event-driven freshness instead of time-driven refreshes.

PoC Scope / Limitations

- Only for iOS
- Only delivery state events are published

Architecture Overview



How Backend Posts Events to the App

- New backend service: `real-time-comms`.
- Producers call `sync(rider_id)` to trigger a client update.
- Payload is currently a freshness signal; state still comes from Home API.
- Supports multiple devices for single rider

Realtime Contract (v1)

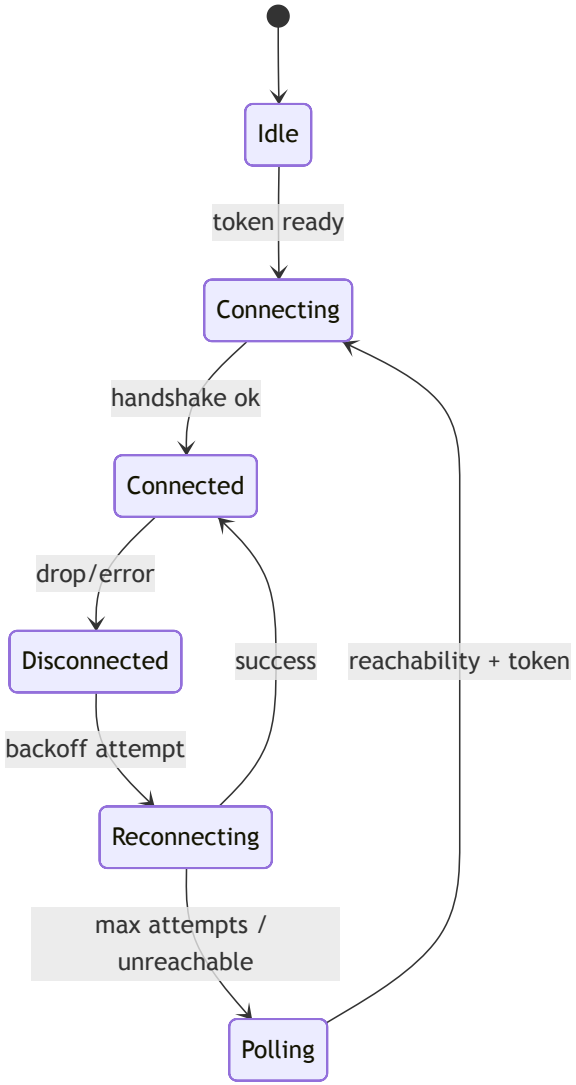
- Delivers signals, not authoritative state.
- Guarantees freshness when connected.
- Automatically falls back to polling.
- Never blocks rider workflows.

Client-Side Implementation Highlights

- Keep the connection open during shift on background and foreground.
- Off-shift only in foreground.
- Exponential backoff on reconnect attempts.
- Reconnect on token expiration with a fresh auth token.

Background and Lifecycle Support

- Socket remains active during shifts.
- Lifecycle-aware reconnect behavior.



Failure Handling and Resiliency

- Reachability-aware reconnect with backoff.
- Explicit fallback to smart polling.
- UI marked as potentially stale when needed.

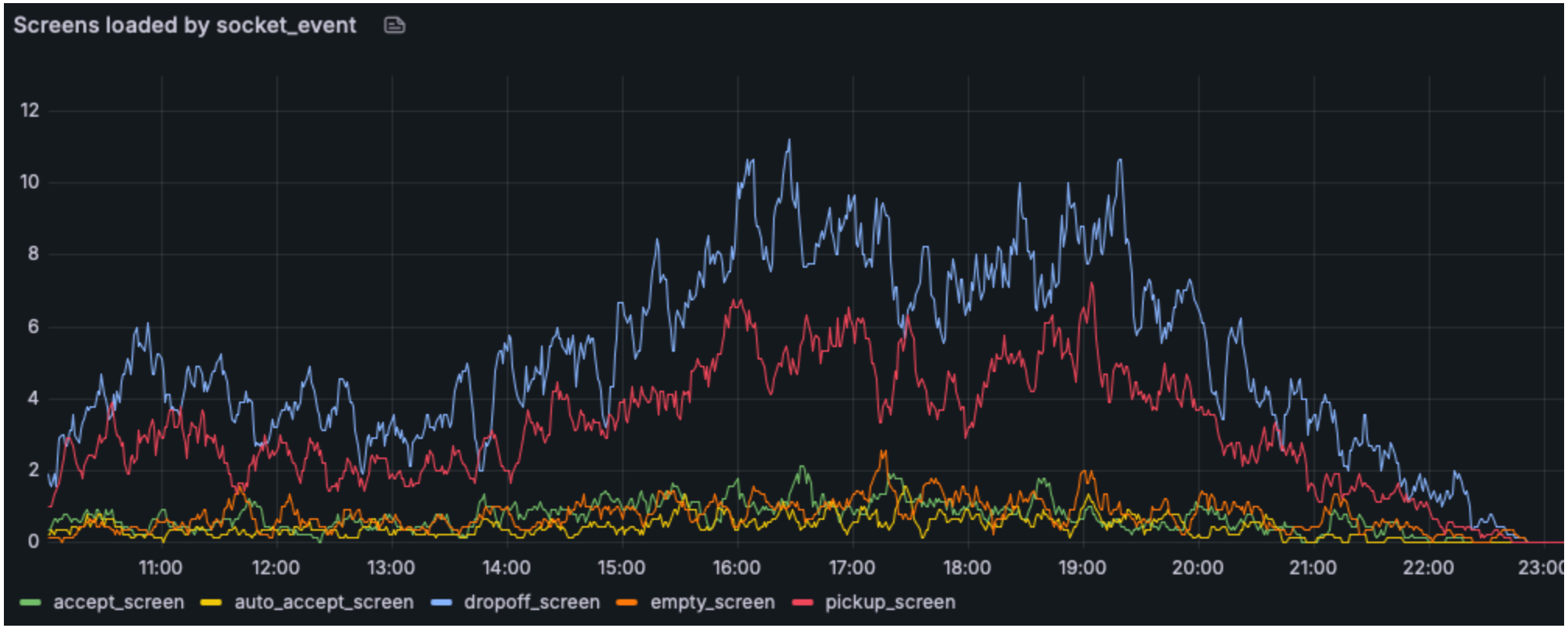
How Rider Home API Refreshes

- Realtime event triggers Home API refresh.
- Dedicated `socket_event` refresh reason.


0:00 / 0:12

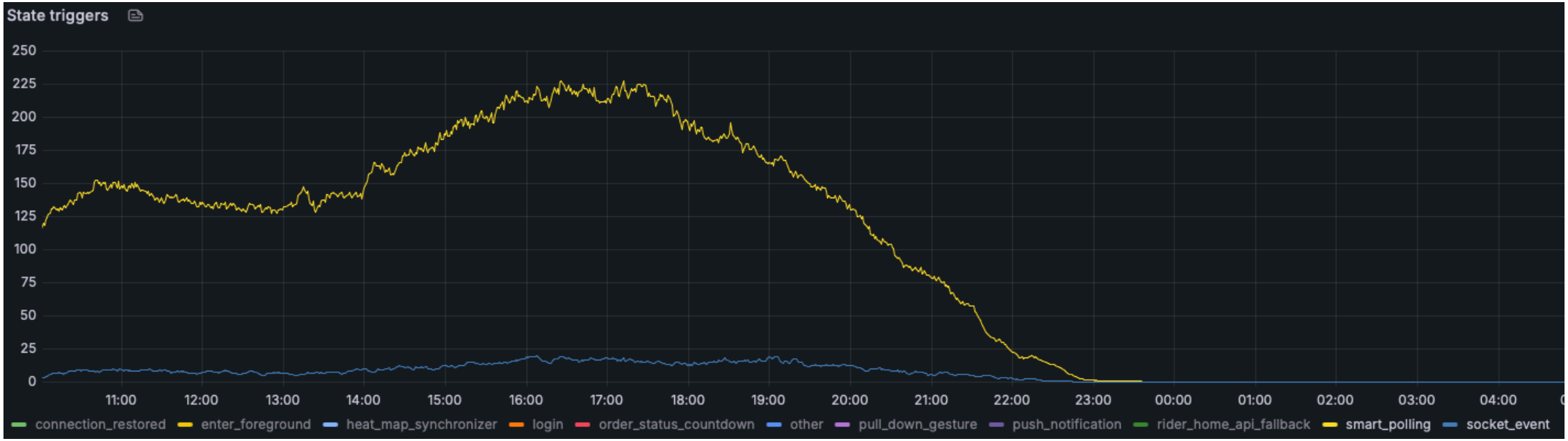
Polling behavior

Socket behavior



Nubmer of reloads

- 1/4 of riders are using the socket connection



Notifications: Today and Next

- Home API refresh drives local notifications.
- Long-term goal: stop sending remote push when socket is connected.

Future Vision: Delta-Driven Client Updates

- Move from “signal then refetch” to “apply delta then reconcile.”
- Client maintains the last known state and applies server deltas safely.
- Home API becomes a periodic integrity check and fallback, not the hot path.

Finland A/B Test: Early Results

- Home API traffic down 42%.
- Delivery Not Seen incidents down 16%.
- Reaction time improved from 6.6s to 5.4s.
- Acceptance rate up, reassignment down.

Battery Impact

- Mean drain delta ~1.5% vs control.
- Guardrail at 2% with kill switch ready.

Next Steps: Q1 2026

- Platform: larger-country A/B + offline indicator.
- Consumers: Android client, Live Activity integration.
- Infrastructure: notification logic, shared shift state.
- Partnership with Fundamentals for follow-up experiments.

Beyond Q1 2026

- Stream GPS updates from the client via realtime.
- Stop all other refreshes if the socket is active (coming to the foreground, etc.).
- Expand realtime coverage to other use cases (quests, work opportunities, etc.).

Thank You

Project Folder

Comms Dashboard

Polling Dashboard

A/B Test Monitoring

Slack Channel

Time for questions