End-to-End Testing Radio Pipelines

Liquidshop 6 — Attila Györffy

A talk about why testing audio output retroactively beats inspecting events.

The paradox

- The system said: **"DJ is on air."**
- The listener heard the song that should have ended.
- Both were telling the truth — about different layers.

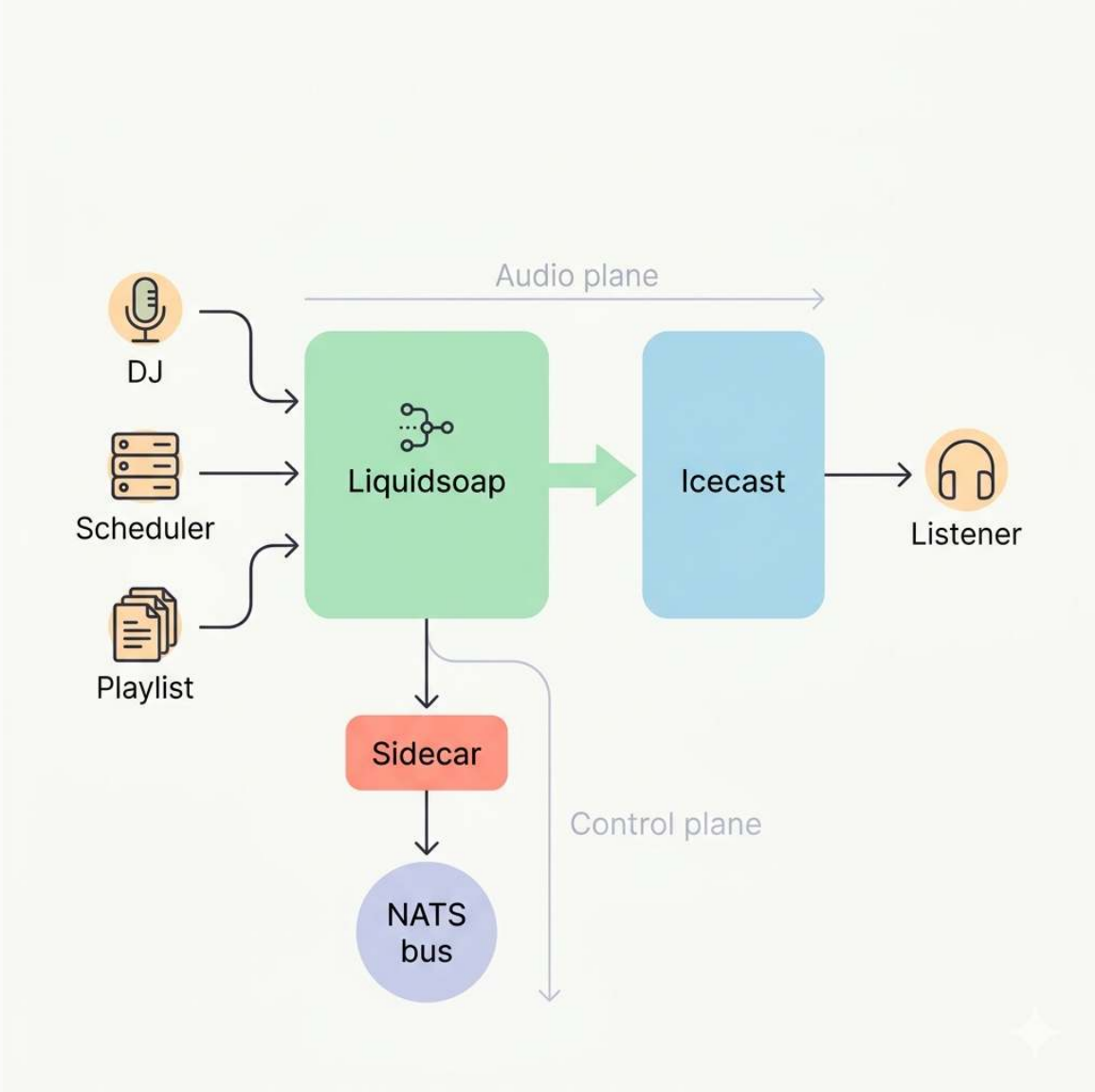
Who I am

Attila Györffy

- 20+ years writing reliable software
- TDD enthusiast — outside-in testing is the default in the web world
- Built a homelab radio station (Liquidsoap + Icecast + Go)
- My tests couldn't catch the bug a *listener* could

The system

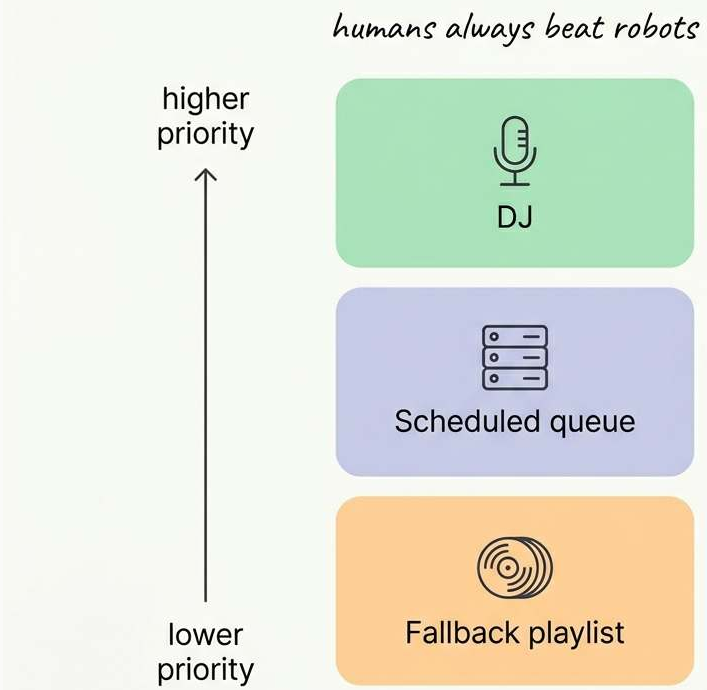
- **Audio plane**: SRT → Liquidsoap → Icecast → listener
- **Control plane**: webhooks → sidecar → NATS bus
- Tests usually only watch one of these planes. That's the bug.



Intended behaviour

The principle: **humans always beat robots.**

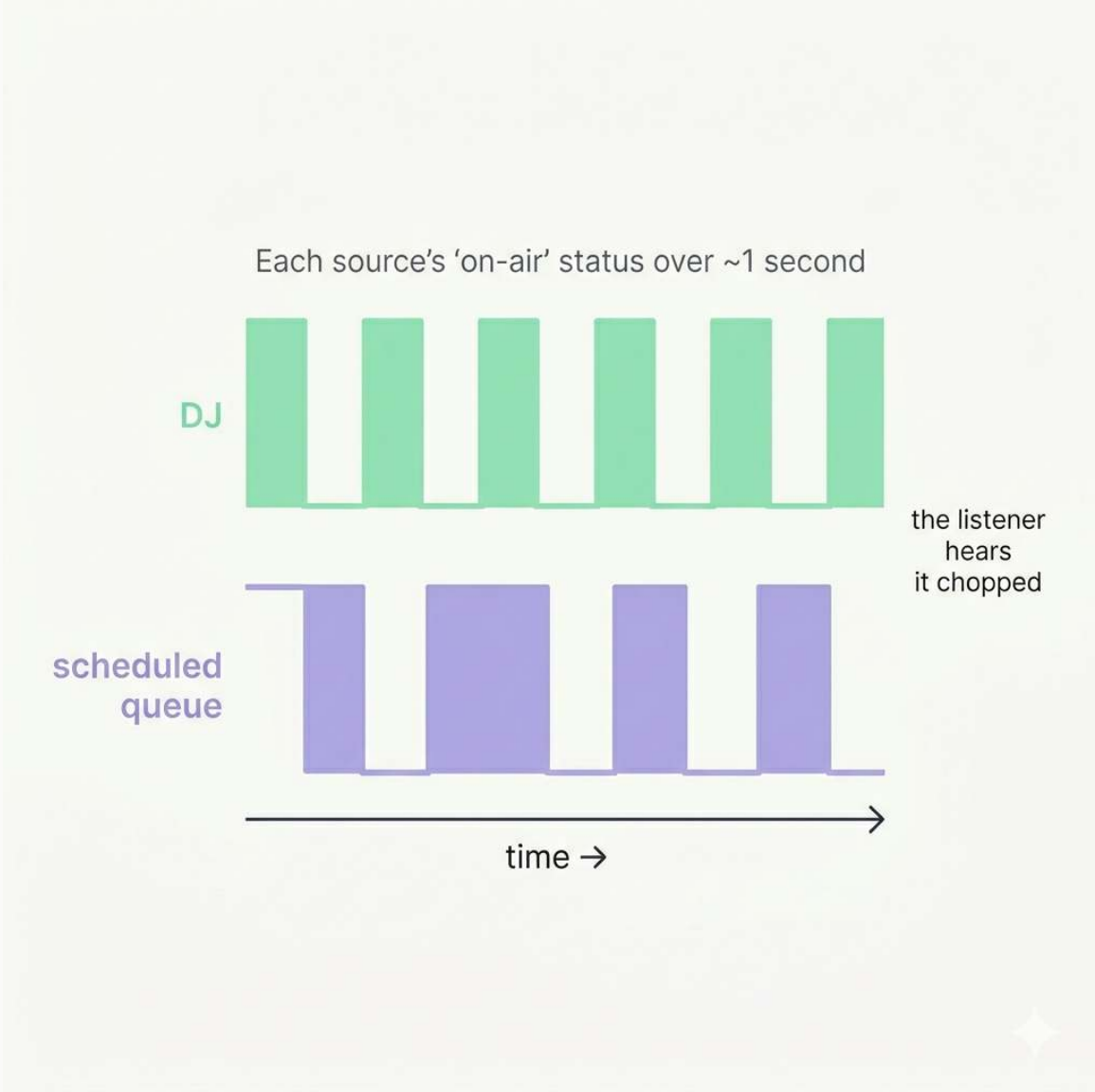
- Highest-priority *ready* source wins
- Preemption is mid-track
- Drop down on un-ready



When the priority chain can't decide

The buffer briefly empties. The chain reads it as "DJ disconnected" and drops to the queue.

Networking calls this **flapping**.
We patched it. The stutter went away.



Three reasons that wasn't enough

1. **We hadn't measured.** Tests went green; we never listened.
2. **Regressions are silent.** The next change could undo the fix.
3. **Other bugs have the same shape.** Codec hiccups, encoder glitches — wrong audio, control plane green.

> *We thought we'd fixed it. The tests said we had. But how would we know it's actually gone?*

First attempt: test the control plane

```
 1.  Fallback playlist plays on startup    → webhook log
 2.  Icecast metadata matches webhook      → webhook log
 3.  Telnet queue.push triggers metadata   → webhook log
 4.  Telnet skip                           → webhook log
 5.  Live source connect                   → webhook log
 6.  Live handback to fallback             → webhook log
 7.  Anti-flapping                         → webhook log
 8.  Blank detection failover              → webhook log
...                                        → webhook log
```

> *This suite ran in CI on every push. It went green every time.*
>
> *We could still hear the bug. The suite couldn't.*

What is testing, anyway?

Testing is **observing what we expected**.

If anything between input and output breaks, the expected observation doesn't appear.

These are **end-to-end tests** — every layer the listener depends on.

They run in CI, before we ship to production.

End-to-end testing in the wild

```
# Web (Playwright / Selenium)
page.goto(url)
page.click("Buy")
assert page.has_text("Order confirmed")

# Mobile UI (XCUITest / Espresso)
app.tap("Settings")
assert app.label("Dark mode").isVisible

# Video player
frames = capture_screen(player, seconds=10)
assert frames_decode_cleanly(frames)
assert audio_synced_with_video(frames)
```

End-to-end, applied to radio

```
recording = capture(icecast_url, seconds=85)

assert no_dead_air(recording)
assert dj_audio_present(recording, during=live_window)
assert scheduled_audio_absent(recording, during=live_window)
```

- The "browser" is the **Icecast stream**
- The "DOM" is the **recorded audio**
- Don't peek inside. Test what the listener actually hears.

The test stack

```
nats:        nats:latest --jetstream
icecast:     real Icecast2
liquidsoap:  real savonet/liquidsoap, real radio.liq
sidecar:     real Go service
test-runner: Go + ffmpeg (SRT-capable)
             – synthesises DJ streams (sine → SRT)
             – records the Icecast output as WAV
```

- The test-runner is the **only** fake
- **~3 minutes** per CI run
- **Runs before deploy**

One continuous broadcast

t	What happens	Audio plane	Control plane
0–10 s	Fallback playlist	random track	metadata event
10–20 s	Scheduler queues track A	scheduled track A	metadata for A
20–25 s	DJ connects	crossfade A → DJ	`LiveDetected`
25–60 s	**Sustained live session**	**DJ, uninterrupted, 35 s**	(no events)
60–75 s	DJ disconnects, handback	crossfade DJ → scheduled B	`LiveLost` + metadata B
75–85 s	Post-live track	scheduled track B	metadata stable

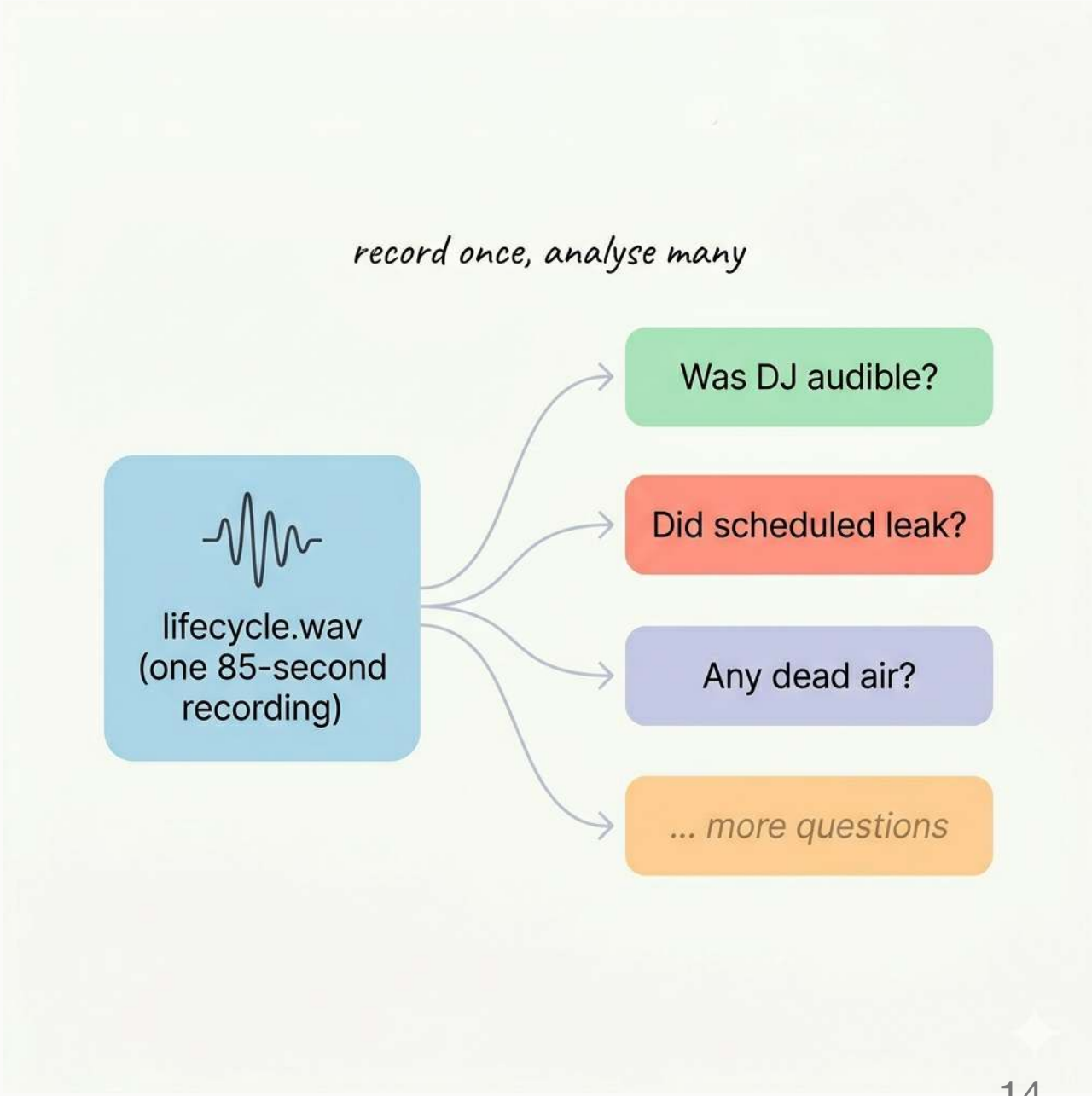
One WAV recording. All assertions run *retroactively* against it.

Signatures, and one recording

The scheduled tracks are **test fixtures** — known inputs we can cross-check against the output.

Each fixture gets a pure sine tone as its signature.

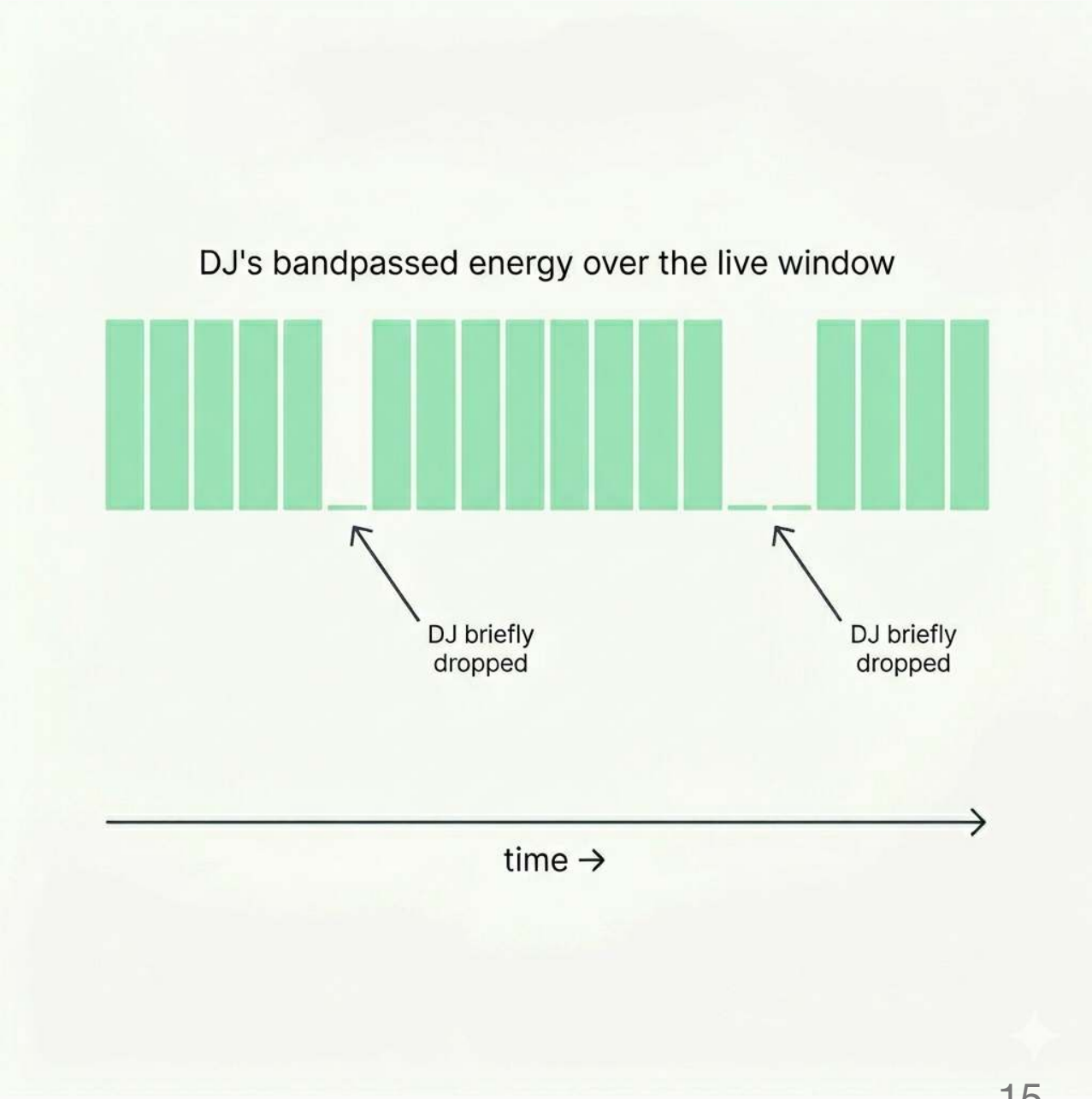
A pure tone is unmistakable.



Three ffmpeg primitives

- `atrim` — carve a time window
- `bandpass` — isolate one frequency (the prism)
- `silencedetect` — log every quiet stretch

"Was the DJ audible?" Any silence range = a dropout. We want zero.



Same chain, inverted

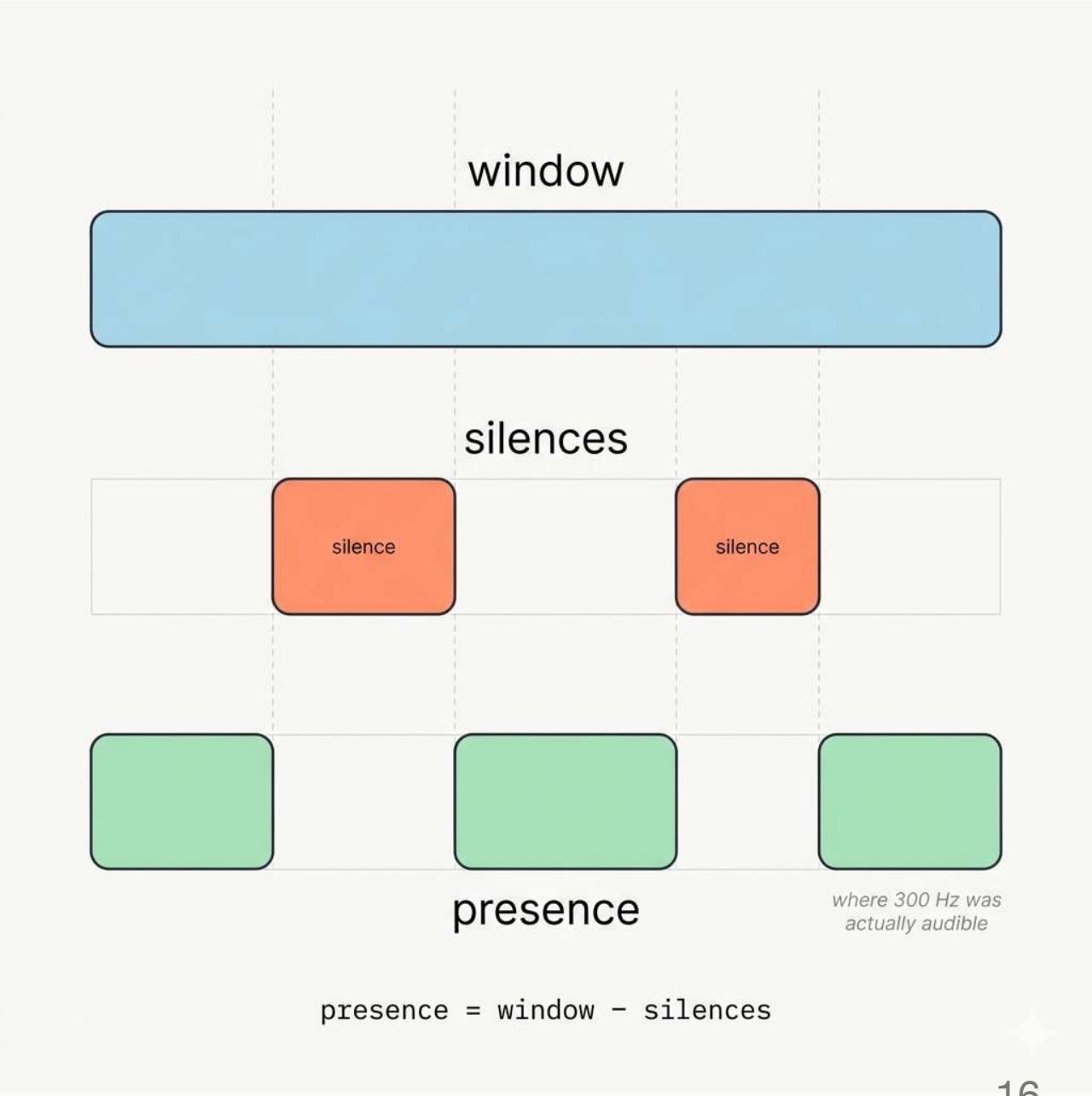
"Did the scheduled track leak when it shouldn't?"

ffmpeg ships a *silence* detector, not a *presence* detector.

So we ask "when was 300 Hz quiet?" and take the complement.

```
presence = window − silences
```

Any presence in the live window = a leak.



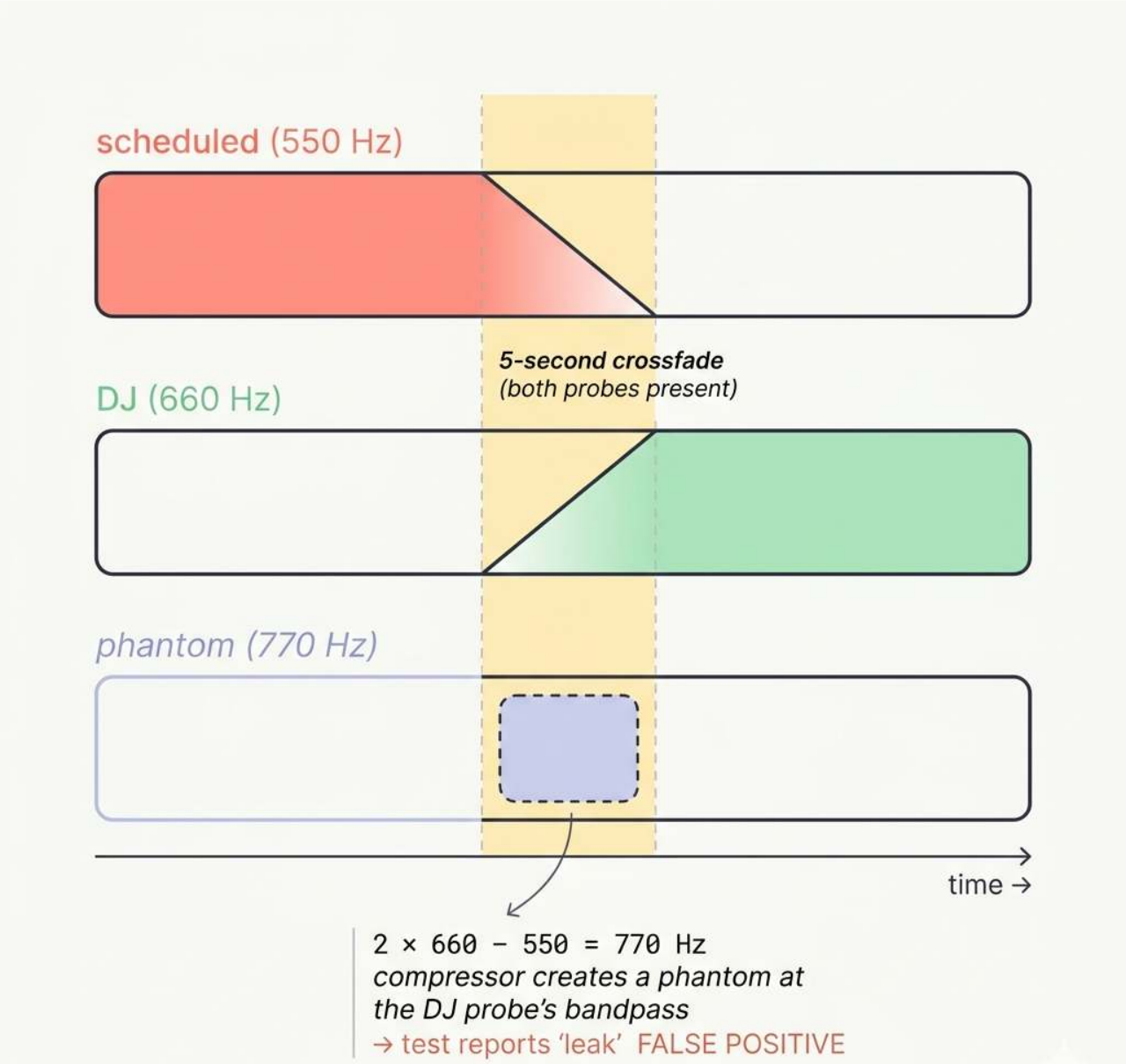
The twist

1. Three probes: 550, 660, 770 Hz.

2. The test failed. The 770 Hz bandpass lit up when nothing was supposed to play at 770 Hz. *What?*

3. The crossfades. During each 5-second crossfade, two probes mix. The compressor (non-linear) creates a phantom: 2 × 660 – 550 = 770 Hz.

Fix: 300, 2000, 5000 Hz.



The test signal is part of the system

```
550 + 660  → 2 × 660 − 550 = 770 Hz   ← collides ✗
660 + 770  → 2 × 660 − 770 = 550 Hz   ← collides ✗

300 + 2000   → 3700 Hz   ✓ off-probe
2000 + 5000  → 8000 Hz   ✓ off-probe
300 + 5000   → 9700 Hz   ✓ off-probe
```

Same shape elsewhere: image compression artifacts, video codec smearing, database query plans on test fixtures.

> ***The test signal is part of the system under test.***

Take-homes

- **Test the outside of the system.** Webhooks are not the air.
- **Record once, analyse many.** Crossfades, silence-trim, ducking — same primitives.
- **The test signal is part of the system under test.**
- **This is testing, not monitoring.** Different problem, different tools.

> *If what matters is what the listener hears, that's what the test has to listen to.*

Thanks

Attila Györffy

attilagyorffy.com

Bluesky · @attilagyorffy.com

X · @attilagyorffy

Mastodon · @attila@m.attilagyorffy.net