WebRTC-to-HLS bridge — decision sheet

Five production patterns side by side, latency budgets, and the 2026 decision tree for picking the right one.

A. Five bridge patterns at a glance

Pattern	Composition	Live/Batch	Glass-to-glass	Layout flexibility	Operator cost	Best fit
1. Track recording	None — per-track files (.mjr / WebM).	Batch	Not live; minutes-to-hours after the room.	None until post-processing.	Lowest	Telemedicine compliance, training archives, deposition copies.
2. Server-side compositor	Bridge worker; GStreamer / FFmpeg with compositor and hlssink2.	Live	2–5 s LL-HLS · 8–15 s classic.	Limited; code change per layout.	Low/room at scale	High-volume standard-layout broadcasts, sports betting, live shopping at 10k+ viewers.
3. Headless browser	Bridge spins up headless Chrome rendering an HTML/CSS template.	Live	3–5 s LL-HLS · 8–15 s classic.	Unlimited; CSS/JS layout.	Medium	Bespoke layouts, brand overlays, custom transitions, SaaS conferencing.
4. WebRTC → RTMP → live encoder	Bridge composes; downstream encoder (MediaLive, Mux, IVS, Wowza) handles HLS.	Live	6–12 s typical.	Limited on bridge side.	Low at bridge; encoder billed separately	Teams already running a live-encoder cluster who want to add a WebRTC origin.
5. Managed bridge SaaS	Vendor handles everything; operator picks template.	Live	3–8 s typical.	Vendor templates.	Highest per minute	Time-to-market wins; small to medium volume; standard layouts.

B. LL-HLS Pattern 2 latency budget — stage by stage (intra-DC, 200 ms parts, 2-part player buffer)

Speaker capture 30 ms · encoder pacing 30 · RTP speaker→SFU 40 · SFU→bridge intra-DC 5 · bridge decode (1 frame) 33 · compositor 8 · bridge encode 33 · packager close 200 ms part 200 · origin→CDN edge 80 · player buffer (2 parts) 400 · viewer decode 100 ⇒ Total ≈ 959 ms. Doubling the player buffer to 4 parts pushes the total to ≈ 1.4–1.6 s; classic HLS with 4 s segments and 3-segment buffer ≈ 14–16 s.

C. Decision tree (in order)

Q1. Recording a live audience watches, or only a stored archive? → Archive only: Pattern 1 — Track recording. Per-track files; post-process to HLS-VOD or MP4 later.

Q2. Live broadcast — hundreds or thousands of viewers? → Hundreds: Pattern 5 — Managed bridge / WHEP direct. Skip HLS entirely if viewer count permits.

Q3. Thousands+ viewers — standard or bespoke layout? Standard at scale → Pattern 2 — GStreamer compositor. Bespoke → Pattern 3 — Headless browser (HTML template). Already run a live-encoder cluster → Pattern 4 — WebRTC → RTMP → live encoder.

D. Five pitfalls every bridge team hits

- Pin the bridge subscriber to the top layer — otherwise SFU congestion control silently downgrades the broadcast tail's resolution.
- GStreamer's webrtcbin does not replace the SFU — bridge must speak the SFU's room protocol before RTP enters the pipeline.
- Cloudflare WHIP-in + HLS-out is not one step (mid-2026) — route through RTMP to a Cloudflare Stream Live input if HLS is required.
- Pattern 4's RTMP hop adds ~1.5 s on top of the downstream encoder's segmenting latency. Budget 6–12 s glass-to-glass total.
- Pattern 3 is overkill for standard layouts — headless Chrome is 1.5–2 GB RAM per room. Pattern 2 is one-third the footprint when layout never changes.