Layer1Labs Silicon

LAYER1LABS SILICON INC. · DOCUMENT

The Traffic Demo, In Depth

How a Lingua Franca program times the Layer1Labs traffic demo — and the measurements behind it.

Version 1.0 · October 26, 2026 · Layer1Labs Silicon Inc.

The traffic demo on the Layer1Labs website runs two copies of the same city grid side by side. One grid's lights run on fixed timers. The other grid's lights are timed, live, by a program written in Lingua Franca and executed by the ChronoHive engine. This document explains that program completely — what it decides, what it does not decide, and how the two are kept from drifting apart — and it sets out the measurements the demo stands on: the tests that proved the program returns the same plans as the planner it replaced, the two-day continuous run that followed, and the configuration sweep that chose the settings the demo opens with. The weak results are in here too, by name.

See it live: the demo this document explains — `https://layer1labs.ai/how-it-works/`

Figures in this document are drawn from the project's measurement records; each is cited where it appears. The soak census printed in Section 7 is a formal completion record taken on October 26, 2026, 11:11 AM EDT.

© 2026 Layer1Labs Silicon Inc. All rights reserved. See the Notices at the end of this document.

Contents

1 What this demo is **3**

2 The program that times the lights **3**
2.1 Admission: the round must be allowed to happen 5

3 From program to running plan **5**

4 Inside the engine: how a request becomes a plan **6**
4.1 Admission first 6
4.2 Dispatch by name 7
4.3 The round's context 7
4.4 The emit check, and the answer 8
4.5 The manifest: what the program may call 8
4.6 Integration: the same program, two tiers 9

5 The honest boundary: what the program decides, and what it does not **9**
5.1 Fail-closed, demonstrated 10

6 Measured: the LF path against the previous planner **10**
6.1 The equivalence battery 10
6.2 The full campaign, re-run 11

7 Measured: two days of continuous running **11**

8 Why the demo opens where it opens **12**
8.1 The opening 12
8.2 The ceiling did not move 14
8.3 Weak cells, by name 14

9 How to read the demo **15**

10 Limits, stated plainly **16**

A Appendix A: TrafficGrid.lf, complete **17**

Colophon **18**

Notices **19**

1 What this demo is

The demo on the "How It Works" page of layer1labs.ai — live at `https://layer1labs.ai/how-it-works/` — shows two copies of the same signaled city grid — three avenues crossed by three streets, nine intersections — running side by side on exactly the same stream of cars. The left grid's lights run on fixed timers: each light follows its own pre-set cycle, whatever the traffic is doing. The right grid's timings are computed against the whole grid's state — where the queues are, which movements are waiting, what is approaching — by the system this document describes.

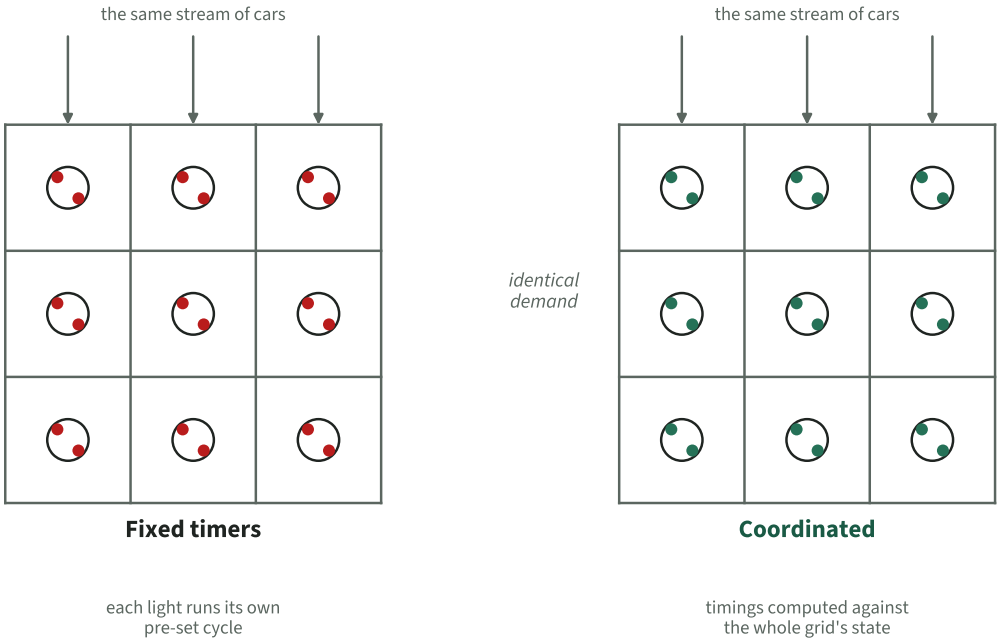


Figure 1: The demo's comparison. Both grids are offered the same trips; only the timing differs. Source: the demo page itself.

Identical demand is enforced, not assumed. Both grids are offered the same trips, drawn from the same list (the *trip manifest*), and in every measured run the two manifests are checked identical by hash. In the reference run for the current energy model, the manifest hashed to `dbaf04cc`, carried 1,369 trips, and neither grid dropped a single trip. The configuration sweep in Section 8 checked the same identity in all 180 of its runs (180/180 equal).

One labeling rule applies throughout this document: the fuel and CO_2 figures are **modeled**. They are computed from published U.S. Environmental Protection Agency and Department of Energy factors applied to the simulated driving — they are labeled MODELED wherever they appear, and they are never measurements of real cars.

2 The program that times the lights

The coordinated grid is timed by a single program, `TrafficGrid.lf`, written in Lingua Franca (LF), a language for programs built from **reactors**. A reactor is a component with inputs, outputs, and *reactions* — pieces of code that run when an input arrives. Reactors are wired together with connections, and the wiring fixes the order things happen in.

One execution of the program is one **plan round**: read the grid's state, time all nine intersections, apply

the entry rule, emit one plan. Appendix A prints the complete source, with annotations. The program has four reactor types:

- **PlanDriver** starts the round. Its reaction calls `sense_snapshot`, which reads the grid snapshot the website sent with the request — the queued cars on every approach of every intersection, and what is approaching. The program states its parsing limits at the call site: queues are clamped at 999 and arrival estimates at 9,999, so a malformed snapshot cannot inject absurd values.
- **Intersection**, instantiated nine times (named `i00` to `i22`, one per crossing), times its own intersection. Its reaction calls `time_intersection`, which decides how the signal cycle is split between the crossing streets, whether the left turn gets a lead phase, and the offset that lets green lights progress across the grid. The policy constants are stated at the call site: the cycle must land between 36 and 72 ticks (a tick is the demo's unit of simulated time) with 2 ticks of clearance; a left-turn lead is warranted when left-turn pressure reaches 3, against a left-crossing figure of 24; progression assumes 116 ticks of travel between crossings.
- **Metering** applies the entry rule to the timed grid: `meter_entries` limits how much traffic is admitted from the entry segments, with the entry segment cap stated as 20. Cars held at the edge wait outside the grid instead of flooding in and jamming the interior — a deliberate trade that returns in Section 8, where it explains one honest negative figure.
- **Emit** assembles the nine timed intersections and the metering decisions into the single plan returned to the website.

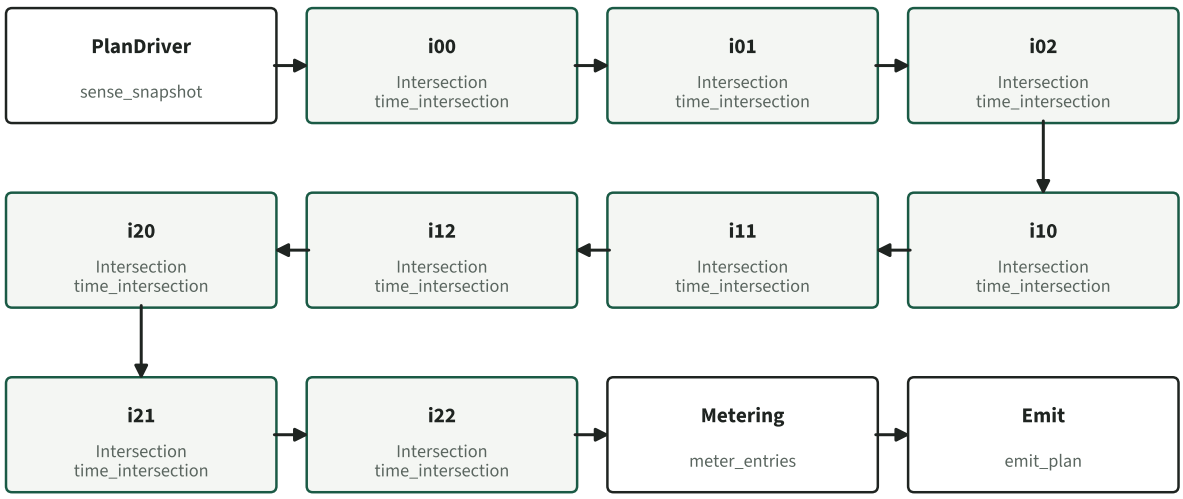


Figure 2: The reactor chain, drawn from the program's own connection list (parsed from the source by script). Source: `TrafficGrid.lf`, SHA-256 `8c4ef47b…`.

The reactors are *chained* — driver, then `i00` through `i22`, then metering, then emit — rather than fanned in to a single collection point. That shape is forced, not chosen: LF forbids one input appearing twice on the right side of a connection, so the nine intersections cannot all feed Metering directly; the compiler rejects that form outright, as the project found when it first tried. The chain has a useful side

effect: it fixes the timing order as row-major — top row left to right, then the next row — the same order every round. Section 3 shows this is one root of the system's determinism.

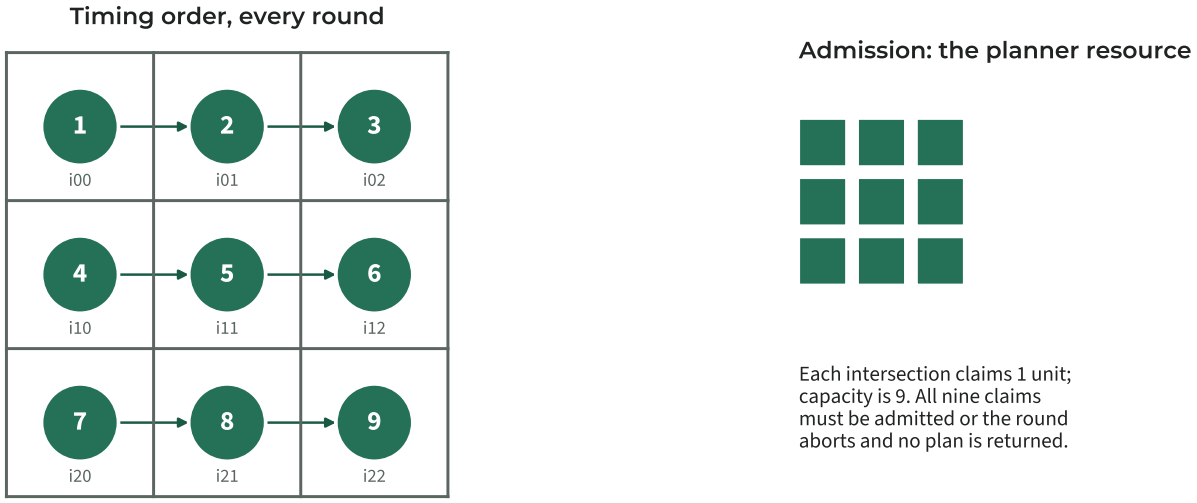


Figure 3: One reactor per crossing, timed in row-major order. Admission (below) makes the order a checked contract, not a convention. Source: `TrafficGrid.lf`.

2.1 Admission: the round must be allowed to happen

Each Intersection claims one unit of a resource called `planner` against a capacity of 9, and every operation in the program is **must-admit**: before any timing runs, the engine's kernel must admit the whole round's schedule against that capacity. A round that cannot be admitted aborts, and the route returns no plan at all rather than improvising one. This is why the path fails the way it does — refused and disclosed, never silently substituted — as the drill in Section 5 shows.

3 From program to running plan

The program is compiled once, ahead of time — never per request. The compiler is **chronoc**, built on the project's pinned fork of the Lingua Franca compiler (`layer1labs/lingua-franca` @ `7eccd811a8ad…`, compiler version `lfc 0.14.0-SNAPSHOT`, chronoc 0.1.0, target `Chrono`). Compilation lowers the program to one **constraint specification** — a compiled form, file suffix `.cspec`, that captures the reactor graph, the ordering between steps, and the resource claims. For this program the specification is 1,563 bytes: 12 operations, 1 step.

The specification carries the source's SHA-256 hash *inside it*, and the demo page displays the same hash beside the printed program; the page build fails if the two ever disagree. What a visitor reads is what runs. The toolchain pin is itself recorded, and two independent compilation routes were run as a check: both produced byte-identical specifications.

Per request, the engine executes the compiled specification through its kernel. The order the kernel runs is the chain's order:

How the kernel runs that order — admission, dispatch, and the round's shared context — is Section 4.

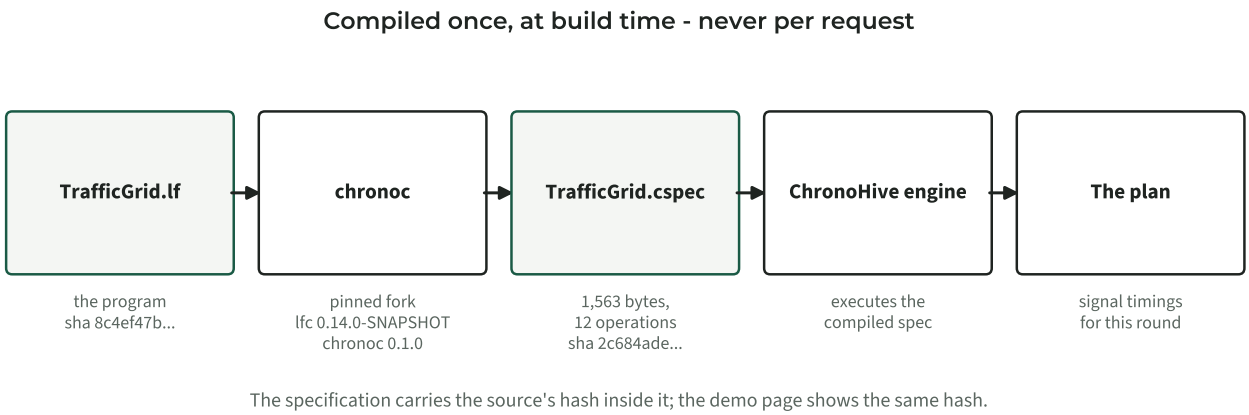


Figure 4: Compile once, execute per request. Both SHA-256 hashes are printed in full in the colophon. Source: the compiled specification itself, measured at this document's build.

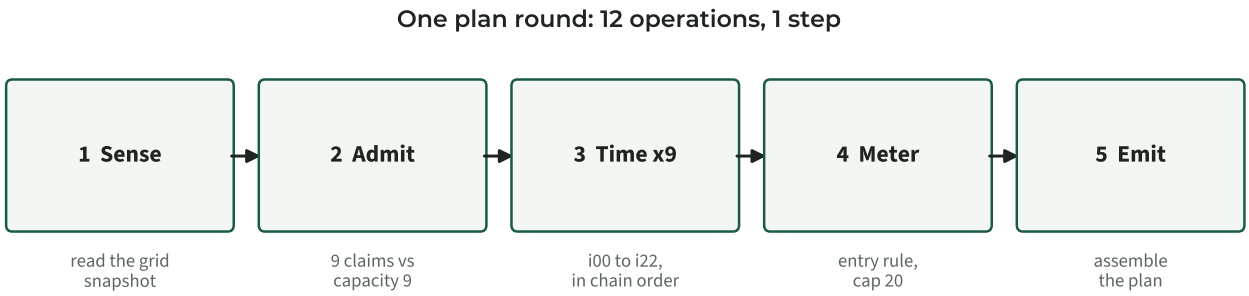


Figure 5: One plan round, in order. Source: `TrafficGrid.lf`.

Because the chain fixes the order and nothing in the round depends on wall-clock chance, the same snapshot produces the same plan every time. This **determinism** was verified directly: all 420 snapshots in the equivalence battery (Section 6) were executed twice, with 0 divergences.

4 Inside the engine: how a request becomes a plan

Sections 2 and 3 described the program and its compiled form. This section describes what the engine does with that form when a request arrives — the mechanism, at the level of what runs, in what order, under what checks. The engine's internal arithmetic is a different matter, and Section 5 draws that line.

4.1 Admission first

A plan request reaches the engine carrying the grid snapshot the website sent. The engine's **kernel** — the part that runs compiled specifications — takes the specification's schedule: for this program, one step holding twelve operations (one sense, nine intersection solves, one metering, one emit). Before any of them runs, the kernel **admits** the step: it adds up the step's resource demands — each intersection solve

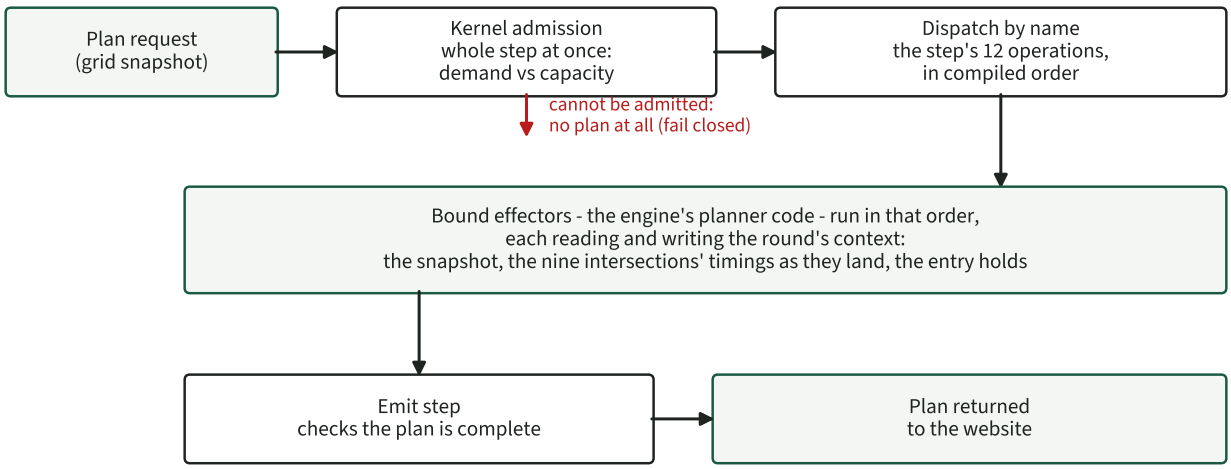
claims one unit of the `planner` resource, against the capacity of 9 the program was compiled with — and checks the total against capacity. The step is admitted as a whole or not at all. A step that cannot be admitted is refused, and because every operation in this program is **must-admit**, a refusal ends the round: the route returns no plan. Nothing runs on a maybe.

4.2 Dispatch by name

Admission granted, the kernel works through the step in the compiled order — the chain's order, driver first, emit last. Each operation in the specification is a *name*, not code: the kernel invokes whatever the engine has **bound** to that name for this program. The bindings connect the four intrinsic names to the engine's planner functions: the sense name to the snapshot parser, the intersection name to the per-intersection planner, the metering name to the entry-hold computation, the emit name to the assembly check (Section 5). If a bound function raises, or the kernel drops an operation, the round fails the same way an unadmitted one does — closed, with no plan.

4.3 The round's context

The bound functions do not pass their results hand to hand. They read and write one shared **execution context** the engine keeps for the round — per request, created when the request arrives and discarded when it completes. The sense operation writes the parsed snapshot and the backbone cycle into it. Each intersection operation reads the snapshot and its own intersection's place in the grid from the context, and writes its timed plan into its slot. The metering operation reads the snapshot and writes the entry holds. Nothing is kept between requests: the next round starts from an empty context and its own snapshot.



An incomplete plan is refused the same way as an unadmitted one: nothing is returned.

Figure 6: One request through the engine: admission, dispatch by name, the round's context, and the emit check. An architecture diagram — it shows the mechanism, not measurements. Source: the engine's executor and effector binding, as described in this section.

4.4 The emit check, and the answer

The last operation is the emit step, and it is a check, not a formality: it verifies that all nine intersection plans and the entry holds are actually present in the context before anything is returned. If the grid is not fully planned, the emit step raises, and the request ends with no plan — a partially assembled plan is never sent and silently completed. A complete plan is returned to the website as the round's answer.

4.5 The manifest: what the program may call

The names the kernel dispatches are not improvised. The compiler knows a table of **intrinsics** — named operations a program may call — and a call to anything not in the table is a compile error. The compiler carries a built-in table of its own (operations from the engine's original storage and checkpoint work: a training step, a checkpoint admission, a prefetch). A program that needs its own operations declares them in a **manifest**: a file versioned beside the source that either replaces the built-in table or, with `extend_default = true`, extends it. `TrafficGrid`'s manifest extends it with the four traffic operations. Each entry declares the operation's name, the effector it binds to, its refusal policy, and the arguments every call site must state:

```
extend_default = true

[[intrinsic]]
name = "sense_snapshot"
effector = "sense_snapshot"
on_refuse = "must-admit"
args = "any"
required_args = ["cycle_min", "cycle_max",
                 "clearance_ticks", "queue_clamp", "eta_clamp"]

[[intrinsic]]
name = "time_intersection"
effector = "time_intersection"
on_refuse = "must-admit"
args = "any"
required_args = ["row", "col", "units",
                 "left_pressure_min", "left_cross", "travel_ticks"]

[[intrinsic.demand]]
resource = "planner"
arg = "units"
scale = 1

[[intrinsic]]
name = "meter_entries"
effector = "meter_entries"
on_refuse = "must-admit"
args = "any"
required_args = ["entry_segment_cap"]

[[intrinsic]]
name = "emit_plan"
effector = "emit_plan"
on_refuse = "must-admit"
args = "none"
```

(The manifest's own comments are omitted from this excerpt, the same treatment as Appendix A's plate; the entries are complete.) Two entries deserve a reading note. The `time_intersection` entry carries the round's only resource demand: one unit of `planner`, taken from the `units` argument the call site states — that is the claim the kernel checks at admission. And `emit_plan` declares `args = "none"`: it takes no arguments because everything it checks is already in the round's context.

Adding an operation to a program follows the same shape, in this order: implement the operation once, in the engine, as an effector; declare it in the program's manifest (name, binding, refusal policy, required arguments, any resource demand); call it from the program, stating its constants at the call site; recompile. The compiled specification then carries the new operation by name, and the engine binds it exactly as it binds the existing four.

4.6 Integration: the same program, two tiers

Putting a program into service is the pipeline of Section 3 plus the binding step above: write the program and its manifest; compile with chronoc to one constraint specification; bind the specification's intrinsic names to effectors in the engine; serve requests. The compiled specification is the deployment contract, and it is the same contract on both of the architecture's tiers. The ChronoHive engine — the software tier, and the one this document's measurements describe — runs it with effectors bound to software functions. ChronoFabric, the hardware tier, runs the identical specification with the same effectors bound to FPGA and hardware endpoints instead. Nothing else about the program changes: admission, sequencing, and the determinism guarantee carry over, because what is being enforced is the specification, not a reimplementation of it. (As the ChronoFabric page puts it: only the effector binding changes.)

> **Why there is no code inside the reactions.** In ordinary Lingua Franca, a reaction's body holds code written in the target language — C, Python, and so on — and the compiler emits that code into the running program. The Chrono target is deliberately different: a reaction body may only *call* an intrinsic by name. No implementation travels inside the program, so nothing about the arithmetic can hide in it. The compiled specification stays what it is in this document — a pure schedule of named operations, their order, their stated constants, and their resource demands — which is what makes it portable between the software engine and the hardware tier, and what makes it auditable: a reader with the specification and the manifest knows every operation a round can perform, because there is nowhere else for one to be.

5 The honest boundary: what the program decides, and what it does not

This section states the claim boundary as plainly in this document as it stands on the project's record, because the boundary is part of the product.

The LF program *is* the schedule the engine executes for a plan round. It decides which operations run, in what order, with what stated constants, and under what admission rule. The claim this document makes — and the claim the demo page makes — is: *this is the program that times these lights.*

The four operations (`sense_snapshot`, `time_intersection`, `meter_entries`, `emit_plan`) are the **intrinsics** of Section 4: operations the program calls by name, declared in the manifest that ships beside the source. Each is bound to the engine's existing planner code as an **effector** — the point where the program touches the system that does the arithmetic. The arithmetic inside the four operations is therefore the engine's planner code, as it was before this program existed. Nothing in this document claims the arithmetic is written in LF.

The link between the program and the arithmetic cannot drift silently. The constants stated at the program's call sites (the cycle bounds, the clearance, the left-turn warrant, the travel time, the entry cap) are pinned to the planner's own constants by a test that derives both sides from the artifacts themselves — the compiled program on one side, the planner it binds to on the other — and fails on any divergence. The few constants the LF subset used here cannot state — a floating-point split gain and its clamps — live

only in the planner, and this document says so rather than approximating them in the program to make it look more complete.

5.1 Fail-closed, demonstrated

If the compiled specification is missing or corrupt, the engine refuses: the route answers with an error (status 503) and no plan. It never substitutes the other planner silently. This was demonstrated by drill, not asserted: a deliberately truncated specification was forced on the test key, and the LF path refused — the two probes were answered in 1,134 ms and 1,224 ms, with no plan — while the website honestly showed its fallback state (during the drill the website's serving layer answered from its built-in fallback, returning 9 plans and 24 holds, labeled as fallback). The production key was unaffected throughout.

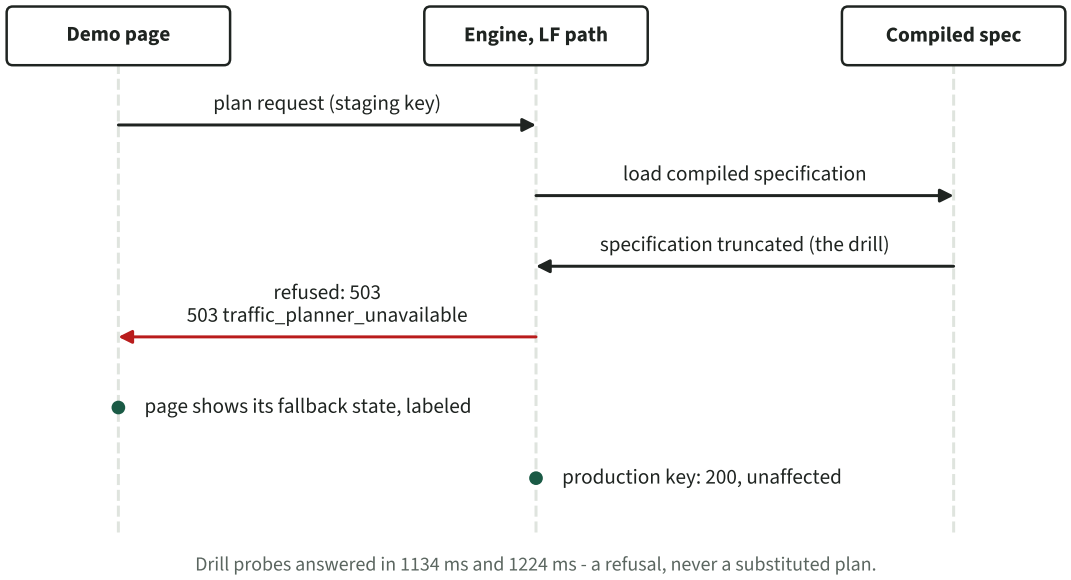


Figure 7: The fail-closed drill. A corrupt specification produces a refusal and an on-page disclosure, never a quietly substituted plan. Source: the project's drill record, measured on the staging key.

Rollback is equally un-dramatic: which path a key uses is a flag the engine reads per request, so switching a key between the LF path and the previous planner takes effect immediately, with no restart. The full flip-and-back drill took about 50 seconds. Section 7 records the day that switch was thrown for production.

6 Measured: the LF path against the previous planner

Before the LF program could time the production demo, it had to prove it returned the same plans as the planner it replaced — the same, or better, was the bar set for the swap, and "the same" is what the evidence shows, exactly.

6.1 The equivalence battery

The battery used 420 snapshots drawn round-robin from a harvested corpus of 4,994 unique grid states. On every one, the LF path and the previous planner returned **identical plans and identical holds — 420 of 420** — and the constants-pin test (Section 5) was green. The determinism double-run of the same battery found 0 divergences (Section 3).

6.2 The full campaign, re-run

The demo's whole measured-results campaign was then re-run against the LF path. The two engine paths were **bit-for-bit identical on all 90 per-seed margins**. The campaign census: 38,872 engine calls, 100% answered with status 200, zero fail-closed events. Latency, measured interleaved on the same 300 snapshots per path so neither side got the quiet machine:

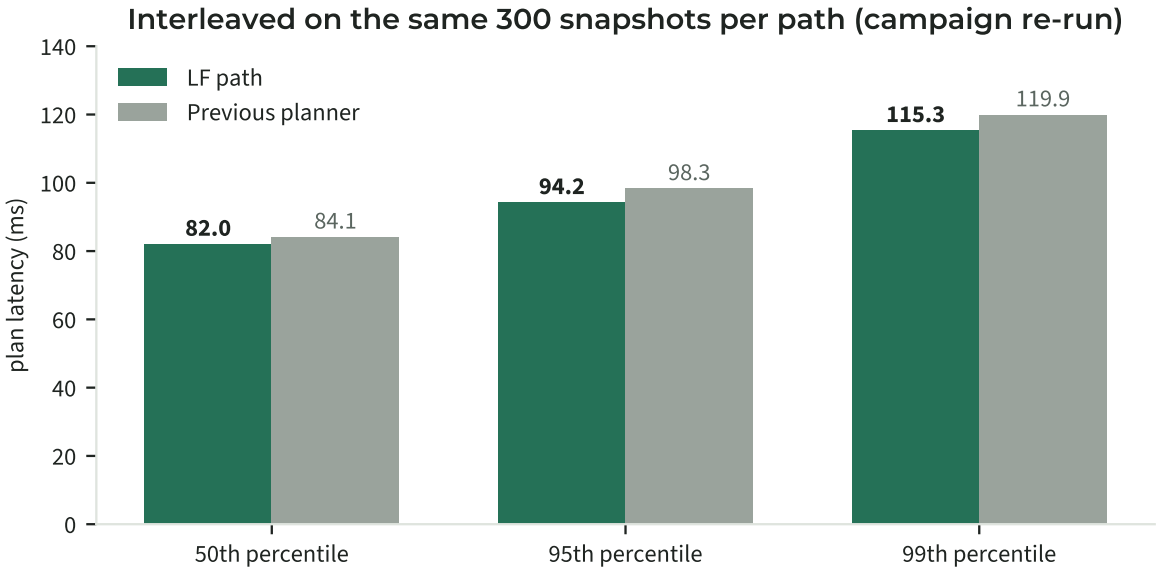


Figure 8: Plan latency by percentile. The LF path is at or faster than the previous planner at every reported percentile — 50th: 82.0 vs 84.1 ms; 95th: 94.2 vs 98.3 ms; 99th: 115.3 vs 119.9 ms. Source: the project's campaign re-run record.

The campaign's history deserves one honest paragraph. The first campaign run disagreed with the project's existing record on four cells. A red team traced the disagreement to the *measurement harness*: its flush step returned when the engine's bytes arrived rather than when the reply chain had settled, so plan-adoption ticks were being randomized by a race in the test rig — the engine paths themselves were identical. The four record figures were corrected openly (the originals remain on the record, marked superseded), the record was re-based by the project owner's ruling, and the re-run passed 18 of 18 checks against the corrected record. The document reports the corrected figures throughout.

7 Measured: two days of continuous running

Equivalence in a battery is one thing; running unattended for two days is another. After the campaign re-run, the LF path was put on a **soak**: one real plan request every 30 seconds, under a dedicated test key, with request bodies cycling the equivalence battery's corpus. The pass criterion was fixed in advance: 48 hours elapsed *and* 5,000 plans served, whichever completes later, with an error budget of zero LF-path errors.

The census printed in Figure 9 is the run's formal completion record (October 26, 2026, 11:11 AM EDT): it records 5,271 call entries: **5,152 plans served**, 119 transport errors, and **0 LF-path errors**, over 48.0 hours elapsed. The run completed at the 48-hour mark on October 26, 2026, with its formal verdict: PASS — both halves of the criterion met, and the error budget untouched.

On the morning of October 26, 2026 — with the soak still running in parallel, under the owner's decision — the production demo's key was switched to the LF path (07:44:19 EDT). Post-switch verification: 8

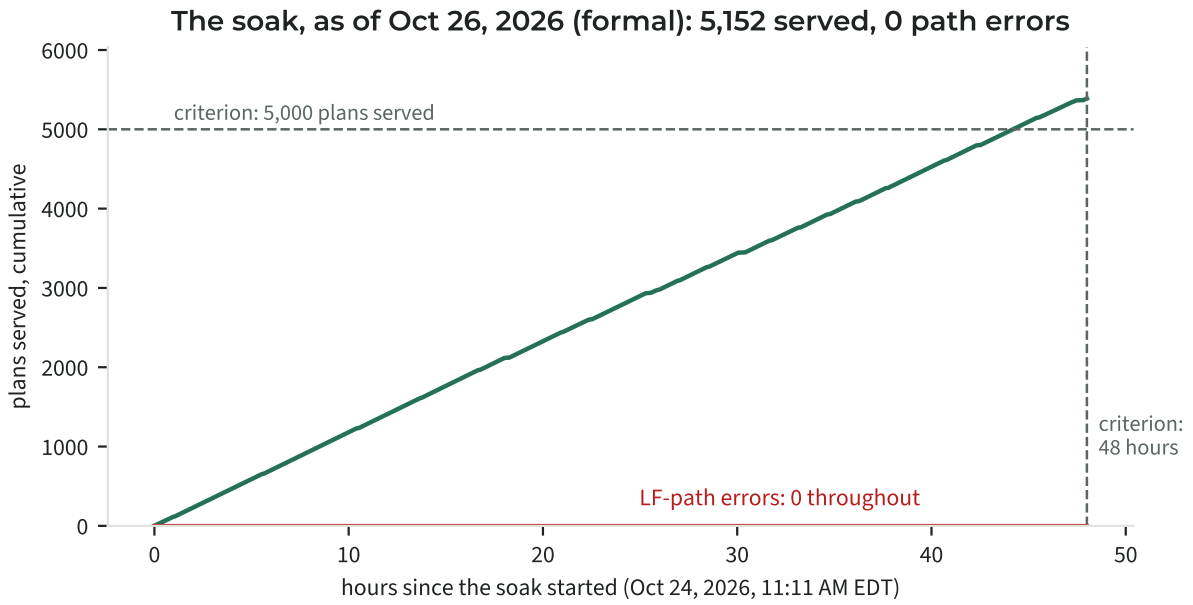


Figure 9: The soak, generated from the soak's own append-only log at build time. Transport errors are network-side failures (request never reached the engine), classified in the log and counted here, never hidden. Source: the soak's own log, complete run of October 24–26, 2026.

of 8 production endpoint calls returned plans from the engine, engine-side 35–45 ms, zero server errors. The page itself did not change (its SHA-256 before and after: `2b03068b…`). From that switch onward, the provenance line beside the program on the demo page — the program printed there is the program that times the lights — is literally true of the live demo.

8 Why the demo opens where it opens

The demo has settings: how much demand, how much turning, what mix of vehicles. The demand settings are named **detents** — Light (28), Typical (34), Heavy (37), in the demo's demand units. Which combination the demo *opens* with is a measured choice, and this section shows the measurement — including the configurations where coordination barely helps, which stay one click away from the opening and are named here.

The sweep measured 36 configurations (steady demand or a simulated day, crossed with the three detents, three turning splits, and two fleets) at 5 seeds each: 180 runs of the shipped demo code, scored with the project's record formulas. Figure 10 is the whole sweep; nothing is cropped out of it.

8.1 The opening

The demo opens at: steady demand, **Typical (34), Low turning**, Typical vehicle mix, events off. Measured there (median of 5 seeds unless stated):

- Delay margin **+44.3%** (seeds +35.2% to +51.2%) — the coordinated grid's delay improvement over fixed timers.
- Average-wait margin **+43.6%** (admitted basis, Section 9).
- Throughput **+27.4%** — more trips completed in the same simulated time.

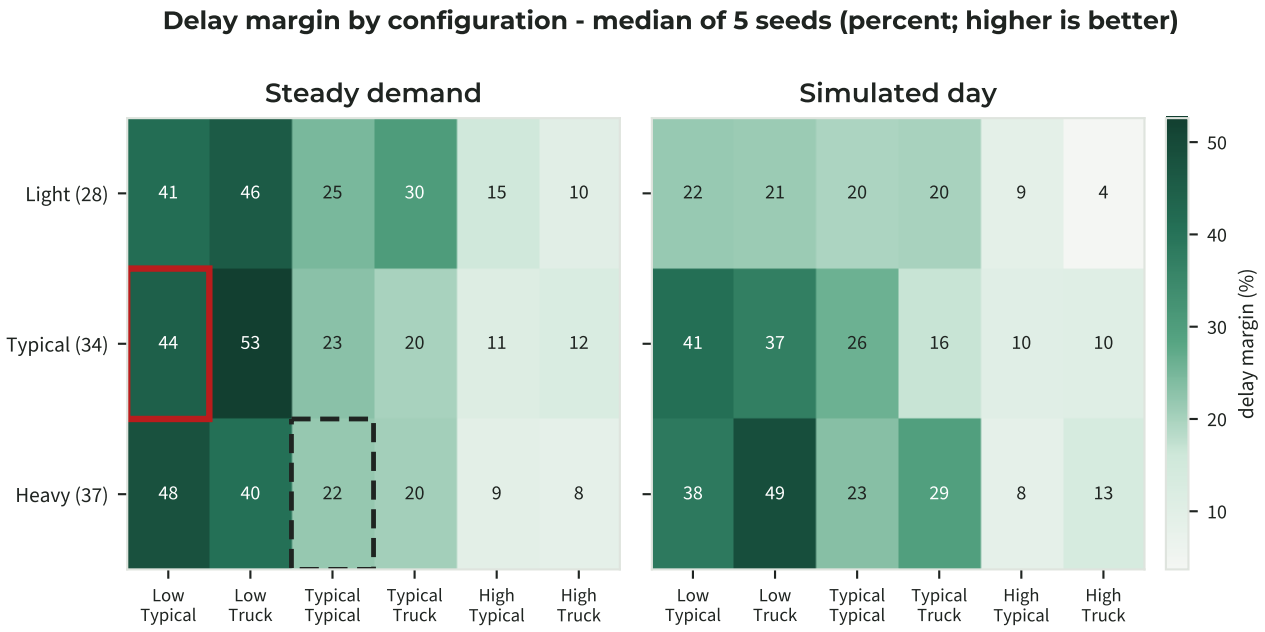


Figure 10: The full sweep. The strongest cells are real, and so are the weak corners on the same chart: High-turning columns and truck-heavy columns are weak everywhere. Source: the project's sweep record (36 configurations, 5 seeds each).

- Per-trip energy saved, by class, on two bases (means of 12 seeds; Section 9 defines the bases): per trip so far / per finished trip — gasoline **+18.1% / +23.7%**, diesel **+17.2% / +23.0%**, hybrid **+18.1% / +23.1%**, electric **-0.5% / +9.7%**; fleet CO_2 per trip **+18%** on the per-trip-so-far basis.

The electric row needs its working shown. Per finished trip, coordination saves an electric trip **+9.7%** on average across twelve seeds — positive on every seed (+0.6% to +16.3%) — against about 23% for the other classes (gasoline +23.7%, diesel +23.0%, hybrid +23.1%). The gap is real, and it is the expected one: an electric motor draws nothing at a standstill beyond a modest auxiliary load, and regenerative braking refunds part of every stop, so a traffic stream full of stops wastes far less of an electric car's energy to begin with. In the model's terms, stops and standing are 64% of a gasoline trip's energy but only 27% of an electric trip's; the rest of the electric figure is distance, which is identical on both grids and beyond any signal's reach.

The running figure — per trip so far, counting every trip from the moment it sets out — can nevertheless show electric trips behind mid-run: it measures about even at this configuration (-0.5% across twelve seeds, individual seeds from -13.4% to +16.5%). That is an artifact of the accounting basis, not of the physics: unfinished trips count at what they have burned so far, and at any moment the coordinated grid's trips have driven further along the same routes — about 14% more miles per trip at this configuration — so they have burned more of their distance energy already, while the fixed grid's trips still owe theirs. Decomposed, coordination's genuine electric savings at this configuration (stops −9.7 points, standing −1.3 points, of which edge waiting is −0.3) are almost exactly cancelled by that +10.5-point progress term.

The natural guess — that metering holds and auxiliary load penalize electric cars — does not survive the decomposition: coordinated electric trips wait about a quarter as long at the grid's edge as fixed-timer trips (34 against 123 ticks per trip), and setting the auxiliary load to zero makes the running figure worse, not better.

© 2026 Layer1Labs Silicon Inc.

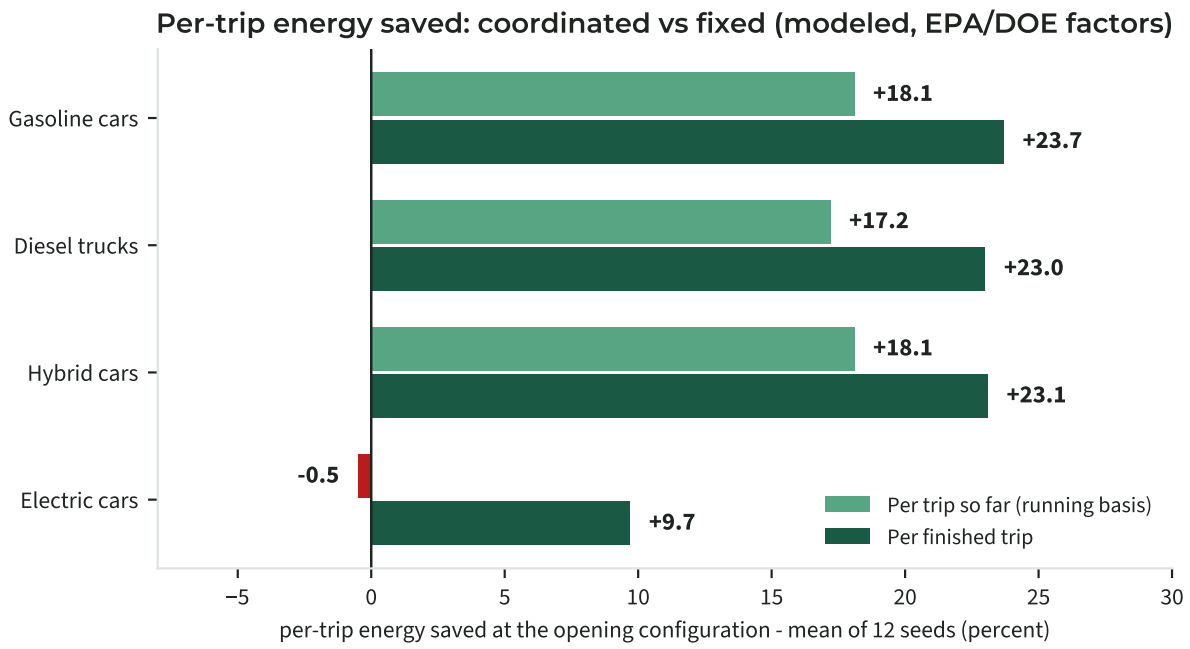


Figure 11: Per-trip energy saved at the opening configuration, on two bases: per trip so far, and per finished trip. The electric row is the one figure in this section that needs its working shown; the surrounding text gives it. Source: the project's re-measurement of the shipped demo at this configuration (12 seeds, October 2026).

The stability mattered as much as the size: this cell's delay range across seeds (16.0 points) is the tightest of the top tier, no seed flips sign, and its worst seed (+35.2%) still beats the previous opening's median delay margin. A visitor's single run lands between +35.2% and +51.2% on delay, every time, on this seed family.

At this configuration, no car is trapped. The dedicated starvation suites — which track whether any car can be stuck in the grid indefinitely — pass all four regime/seed combinations at the opening:

8.2 The ceiling did not move

The previous opening — steady, Heavy (37), Typical turning — sits at the planner's *proven ceiling*. At full saturation this planner produces its improvement of roughly twenty-something percent and no presentation changes that: on the sweep build it measured +24.2 delay / +27.1 wait / +33.5% throughput (the project record's family: +20.2/+26.4/+30.0). This section's opening moved the demo to a regime where the same planner honestly produces more. It did not raise the ceiling, and the demo does not claim otherwise anywhere.

8.3 Weak cells, by name

- **Truck-heavy fleets** are the record's softest family: delay margins of +9.92 at Typical demand and +14.68 at Heavy (both figures as corrected in the 2026-10-24 re-base).
- **The default construction event** measures +15.37 against its own +25 clause — it does not meet it, and its post-event residue (+8.1 against a ± 4 bound) does not meet its bound either. The adversarial construction fixture passes (+26.14). Nobody clicks "adversarial" first; the document names the default.

© 2026 Layer1Labs Silicon Inc.

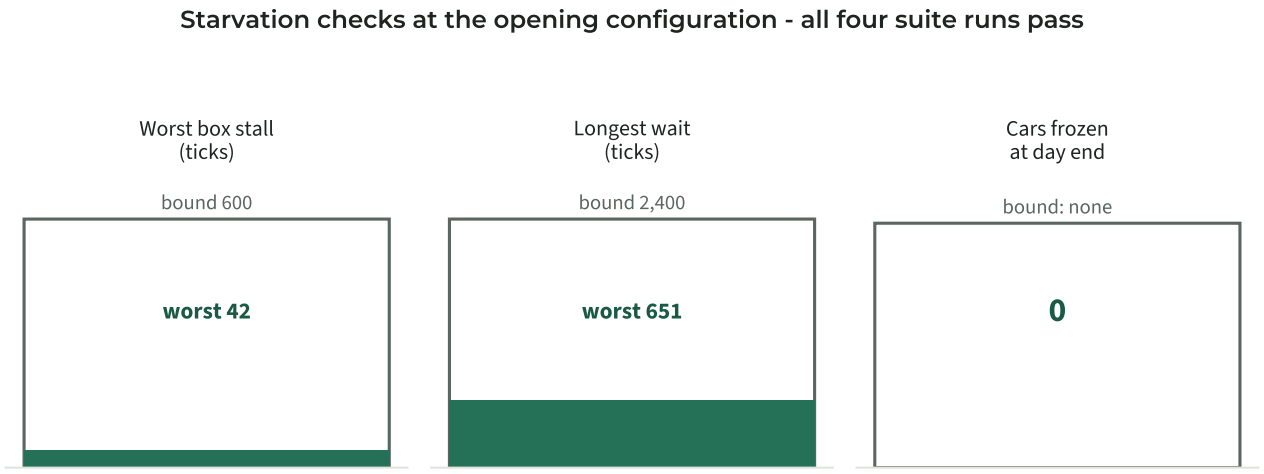


Figure 12: Starvation checks at the opening configuration: worst measured box stall 42 ticks against a bound of 600; longest wait 651 ticks against 2,400; zero cars frozen at day end. Source: the project's starvation-suite record.

- **High-turning corners** compress the margins to medians of +9 to +15, and the sweep flags some of those cells unstable (seed-dependent sign).
- **The electric per-trip figure** is about even on the running basis at the opening (-0.5% across twelve seeds) and positive over finished trips (+9.7%); at the old Heavy opening the running figure was -19. The cause is the accounting basis, stated openly above: unfinished trips count at what they have burned so far, and the coordinated grid's trips have driven further along the same routes. It is one row of four on the scoreboard, and it is honest.

9 How to read the demo

The scoreboard under the demo is the page's own summary, and it is built to be read literally. Its header names the running configuration — every figure under it belongs to the regime named there, and changing a control re-labels the figures with the new regime.

Three reading rules carry the whole page:

Admitted basis. Every comparison counts every car either grid has taken in, finished or not — finished trips at their final figure, cars still on the road at what they have used so far. This is the one framing that cannot be gamed by holding cars back or trapping them: every admitted car is in the numbers from the moment it is taken in.

Per-trip first. Energy is reported as per-class *average cost per trip*. Totals are compared only after both grids have brought the same manifest of trips home — so a totals line can never compare unequal work. Mid-run, the coordinated grid has usually moved more cars, so its raw fuel and CO_2 totals-so-far can read *higher* while its per-trip costs are lower. Read per-trip first; this misreading has happened to a careful reader (the owner), and the gauges now lead with per-trip figures because of it.

Figures settle. Early in a run, figures can favor the fixed side: in one recorded early snapshot (1,200 ticks, at the previous defaults), the coordinated electric figure was 0.205 against the fixed side's 0.174 kWh per trip. The scoreboard's basis labeling exists so a reader can tell a settled figure from an early one.

And one disclosure the page itself makes: if the engine is ever unreachable, the demo holds the last engine plan, then switches to a built-in *fallback planner* — and says so, in the coordinated grid's status line, for as long as the fallback is running. The fallback is different code from the engine planner; on the

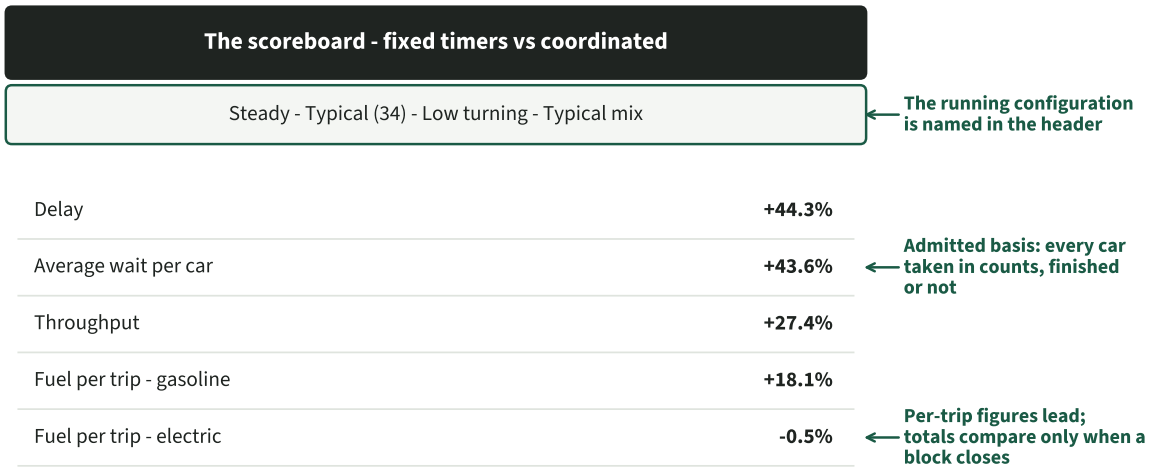


Illustration of the live scoreboard (values: the opening configuration, from the project's measurements).

Figure 13: How to read the scoreboard. Source: the shipped demo page's scoreboard, drawn as an illustration.

harvested corpus the two returned different contents on 14.3% of snapshots. The measured campaigns' trajectories did not visit those states, but this document does not claim the fallback is the product — it is a labeled stand-in, and the divergence is an open item on the project record (Section 10).

10 Limits, stated plainly

What this document does not establish, collected in one place:

- The demo is an illustration: a simplified grid, simulated cars and lights in the browser, modeled fuel and CO_2 (EPA/DOE factors). It is not a field deployment, and its margins are not a forecast for any real city.
- The planner has a measured ceiling at full saturation (Section 8). The opening configuration's figures do not raise it; they show the same planner in a regime below saturation.
- Equivalence is proven on the batteries and campaigns cited — 420 snapshots, the 90-margin campaign, the soak corpus. States outside those are not thereby proven; in particular the fallback-divergence states (Section 9) remain an open owner decision on the record.
- The starvation record has two owner-accepted misses, both at Heavy configurations neighboring — not at — the opening, both at seed 101: a 1,042-tick fixed-grid box stall in the baseline regime, and a 602-tick coordinated stall at High turning in the defaults regime. They are named here because this document names weak things.
- Nothing in this document describes the engine's internal computation. That is deliberate, and permanent for public surfaces.

© 2026 Layer1Labs Silicon Inc.

A Appendix A: TrafficGrid.lf, complete

The complete program, exactly as compiled (SHA-256 `8c4ef47b82da0b91469e4b43a9a304970a3b8420a46e5d221fd9f943bcb7969f`; verified at this document's build time — the build fails on drift). The printing starts at `target Chrono`: the source file's leading comment header is omitted, the same treatment as the program's printing on the demo page itself. The hash above still covers the complete file, header included — nothing in the runnable program is omitted. Annotations follow the plate.

```
target Chrono

reactor PlanDriver(rounds=1) {
    output go
    state c = 0
    timer tick(0, 1 ms)

    reaction(tick) -> go {=
        sense_snapshot(cycle_min=36, cycle_max=72, clearance_ticks=2, queue_clamp=999, eta_clamp=9999)
        c = c + 1
        go.set({"round": c})
        if c >= rounds:
            request_stop()
    =}
}

reactor Intersection(row=0, col=0) {
    input sense
    output timed
    reaction(sense) -> timed {=
        time_intersection(row=row, col=col, units=1, left_pressure_min=3, left_cross=24, travel_ticks=116)
        timed.set({"row": row, "col": col})
    =}
}

reactor Metering {
    input timed_in
    output metered
    reaction(timed_in) -> metered {=
        meter_entries(entry_segment_cap=20)
        metered.set({"metered": 1})
    =}
}

reactor Emit {
    input metered_in
    reaction(metered_in) {=
        emit_plan()
    =}
}

main reactor TrafficGrid(rounds=1) {
    d = new PlanDriver(rounds=rounds)
    i00 = new Intersection(row=0, col=0)
    i01 = new Intersection(row=0, col=1)
    i02 = new Intersection(row=0, col=2)
    i10 = new Intersection(row=1, col=0)
    i11 = new Intersection(row=1, col=1)
    i12 = new Intersection(row=1, col=2)
    i20 = new Intersection(row=2, col=0)
    i21 = new Intersection(row=2, col=1)
    i22 = new Intersection(row=2, col=2)
    m = new Metering()
    e = new Emit()
    d.go -> i00.sense
    i00.timed -> i01.sense
    i01.timed -> i02.sense
    i02.timed -> i10.sense
```

```
    i10.timed -> i11.sense
    i11.timed -> i12.sense
    i12.timed -> i20.sense
    i20.timed -> i21.sense
    i21.timed -> i22.sense
    i22.timed -> m.timed_in
    m.metered -> e.metered_in
}
```

Annotations

- **target Chrono** — the declaration that this program compiles to a ChronoHive constraint specification.
- **PlanDriver** — `sense_snapshot` carries the cycle bounds (36–72), clearance (2), and parse clamps (999 / 9,999) at its call site; `rounds=1` makes one execution one plan round.
- **Intersection** — `time_intersection` carries the left-turn warrant (pressure 3, left-cross 24), the progression travel time (116 ticks), and the resource claim (`units=1` of `planner`, capacity 9).
- **Metering** — `meter_entries` carries the entry segment cap (20).
- **The chain** — the main reactor's connections, in order: the row-major solve order the engine executes (Sections 2 and 4).
- **Emit** — `emit_plan` assembles the plan; it is the chain's last link, so nothing runs after the plan exists.

Colophon

Sources. Every figure in this document is drawn from the project's measurement records, on the bases stated where they appear. The compilation, the equivalence battery, and the campaign re-run were measured between October 23 and 26, 2026, on the pinned toolchain (compilation section, Section 3). The fail-closed and rollback drills were measured on the project's staging key in the same period; the production switch (Section 7) was verified against the live endpoint. The soak census is the run's formal completion record (Section 7). The configuration sweep, the ceiling measurements, the weak cells, and the starvation suites were measured on the shipped demo client in October 2026 (Section 8); the figures the record corrected on October 24, 2026 are printed in their corrected form throughout. The energy model's basis and wording follow the project's measured-results records for the demo's current client. The program printed in Appendix A and its compiled specification are identified by the hashes below; the live pages are layer1labs.ai/how-it-works/ (the demo) and layer1labs.ai/chronofabric/ (the hardware tier described in Section 4).

Hashes. Program source, SHA-256: `8c4ef47b82da0b91469e4b43a9a304970a3b8420a46e5d221fd9f943bcb7969f`. Compiled specification, SHA-256: `2c684ade6a491fb4c6b6d60e0c6ea2407a707f149824e1912f55c9a24edb3607` (1,563 bytes).

Build. Version 1.0, October 26, 2026 — the first published edition of this document. Built with the project's TeX toolchain, on the Layer1Labs document templates. Figures were generated by script from the source data files named in their captions — no figure value was typed by hand. The soak census is stated on the basis given in Section 7.

Notices

Copyright. © 2026 Layer1Labs Silicon Inc. All rights reserved.

Sharing. This document may be shared in its complete, unmodified form, with attribution to Layer1Labs Silicon Inc. No other reproduction, modification, adaptation, or derivative use — in whole or in part — is permitted without the prior written permission of Layer1Labs Silicon Inc.

No license. Nothing in this document grants, by implication or otherwise, any license or right under any patent, copyright, trademark, or other intellectual property of Layer1Labs Silicon Inc. Publication of this document is a disclosure of information only.

Patents. The technology described in this document is the subject of pending patent applications, including 9 provisional patent applications filed with the U.S. Patent and Trademark Office. No statement in this document is a representation as to the scope, validity, or enforceability of any application or of any patent that may issue from one.

Trademarks. Layer1Labs, ChronoHive, and ChronoFabric are trademarks of Layer1Labs Silicon Inc. Other names mentioned in this document may be trademarks of their respective owners; their use here is for identification only and claims no affiliation or endorsement. In particular, Lingua Franca is an open-source research project of its own community.

Measurements and basis. Figures in this document are drawn from the project's measurement records and are cited where they appear; where a figure is modeled rather than measured, it is labeled as modeled. This document is provided for information only, on an "as is" basis, without warranty of any kind, express or implied, including as to accuracy, completeness, or fitness for a particular purpose. It describes a demonstration system; it is not a specification, an offer, or a forecast of results in any other setting.

Current version. Documents in this family are revised in place under version numbers. The current version of this document is the one published at layer1labs.ai; earlier versions are superseded.