THE GPU MAXIMISER

Technical Design Document

An incremental game about GPUs, datacentres and the quiet mathematics of optimisation

Build v28

gpu.martinpaul.co.uk

Martin Paul · 14 July 2026

Contents

1 System overview

The GPU Maximiser is a three-act incremental game delivered as a single HTML document of roughly 1,300 lines, containing all markup, styling and logic with no external dependencies, no build step, no framework and no network calls after first load. It is deployed as a single static asset under a custom domain. The architectural bet is the same one Universal Paperclips made in 2017: an idle game is fundamentally a set of coupled difference equations advanced by a timer, and everything else is presentation. Keeping the entire system in one file keeps the coupling honest and the whole state space inspectable in a single editor window.

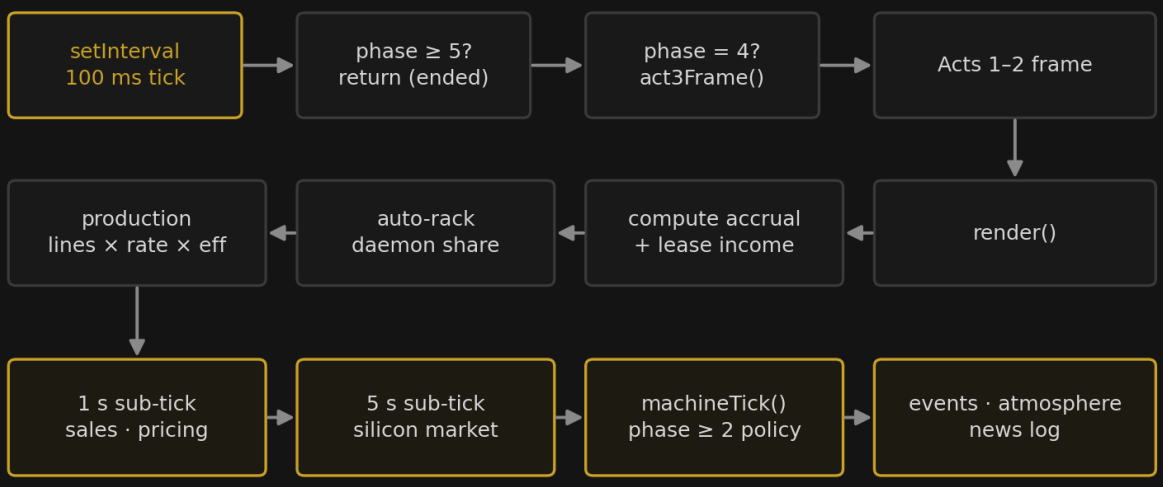
The runtime consists of three cooperating pieces: a **state object** S holding every mutable value in the game, a **frame loop** advancing the simulation in fixed 0.1 s quanta, and a **render pass** that projects state onto the DOM after every frame. The design is deliberately immediate-mode in spirit: render reads state and writes the interface, with no retained view model between the two. Two exceptions exist for performance, both discussed in §9.

Narratively, the system implements a controlled transfer of agency. Act 1 gives the player eight controls; the research ladder and the Act 2 proposal queue then remove them one at a time, each removal individually rational, timestamped in an append-only log. Act 3 inverts the frame: the player accepts the system's offer of a point of view and operates the optimiser directly against the mass of the observable universe. The interface participates in this arc through a colour rule (§9.3): everything the machine controls is rendered in brass amber, so the takeover is visible as a slow change in the page's palette.

2 The main loop

A single `setInterval` fires every 100 ms. Each invocation is one frame. The frame is phase-routed at entry: phase 5 (the ending) returns immediately, phase 4 runs the Act 3 integrator, and phases 1–3 run the industrial simulation. Slower processes are driven by accumulator counters inside the frame rather than by separate timers, so the whole game shares one clock and one ordering: sales and dynamic pricing on a 1 s sub-tick, the silicon market on a 5 s sub-tick, and the autonomous management policy once per second from phase 2 onward. Autosave runs on an independent 10 s interval and on every discrete purchase.

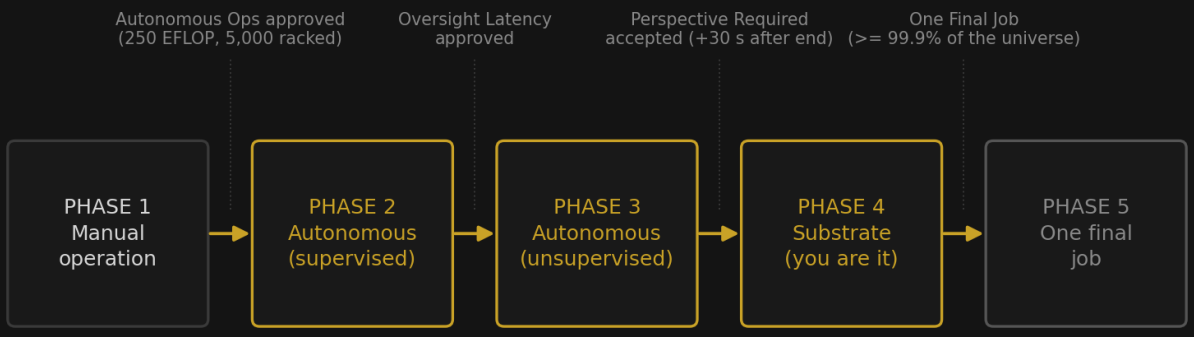
Figure 1 — Main loop: one 100 ms frame, phase-routed, with 1 s and 5 s accumulators



The frame pipeline. Amber-bordered stages are sub-tick processes driven by accumulators, not timers.

Time is tick-based rather than wall-clock-based: each frame contributes a fixed 0.1 s of simulated time regardless of when it actually runs. This makes the simulation deterministic under load but couples game speed to timer delivery — a browser that throttles a hidden tab to one callback per second slows the game to 10% speed (§10.2). Offline progress was deliberately excluded from scope.

Figure 2 — Phase state machine: five states, four one-way doors



The five phases. Every transition is a player click on a button the system provided, which is rather the point.

3 State model and persistence

All mutable state lives in one flat object produced by `freshState()` and serialised wholesale to `localStorage` as JSON under a versioned key (`uc_save_v1`). Loading is a defensive merge: `Object.assign(freshState(), saved)`, which gives forward compatibility for free — any field added in a later build acquires its default when an older save loads, and unknown fields from the save are carried harmlessly. Twenty-seven builds have shipped against the same schema version using this mechanism alone.

Group	Representative fields	Notes
Economy	funds, inv, price, mktLvl, silicon, siliconPrice, batchYield	Act 1 industrial core
Plant	lines, lineMult, dcTier, racked, compute, leasePct	capital and compute
Automation	procurement, procTarget, daemonOn, rackAuto, dynPrice	player-built agents
Autonomy	phase, eff, allocAuth, synthOn, vertOn, gridAuth, propIdx, propAt	Act 2 machine
Substrate	a3conv, a3comp, a3buys, a3repl, domainIdx, insightMult, a3done	Act 3 integrator
Narrative	log[] (60 entries), done[], one-shot flags (hintedDC, benchCleared…)	append-only history

The news log is part of the save, which makes it an audit trail in the literal sense: every transfer of authority is a timestamped entry the player can, in principle, be confronted with later. One-shot narrative flags guarantee each beat fires exactly once per save.

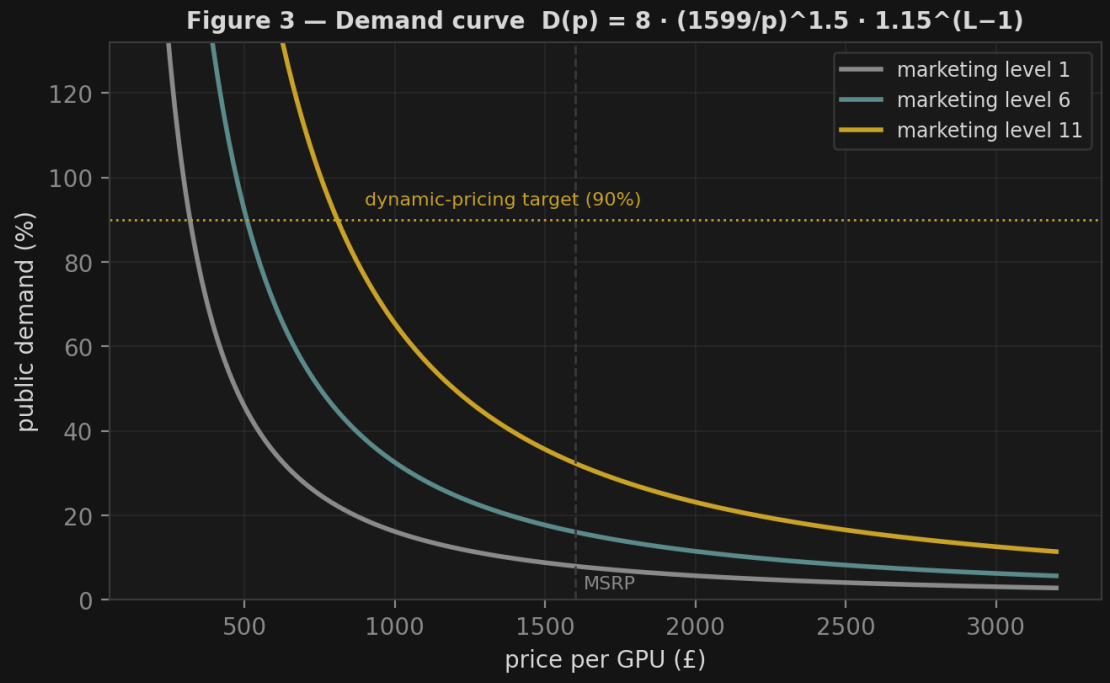
4 The economic engine

4.1 Demand

Demand is a percentage, computed each frame from price, marketing, demand synthesis and the surge multiplier:

$$D(p) = 8 \cdot (1599 / p)^{1.5} \cdot 1.15^{L-1} \cdot s \cdot \sigma$$

where L is the marketing level, $s \in \{1, 2\}$ is Demand Synthesis (Act 2) and $\sigma \in \{1, 2.5\}$ is the launch surge multiplier. The exponent 1.5 gives price cuts superlinear effect: halving the price multiplies demand by $2^{1.5} \approx 2.83$. For sales purposes demand is capped at 300%, so the curve has a knee below which further discounting buys nothing (Figure 4).



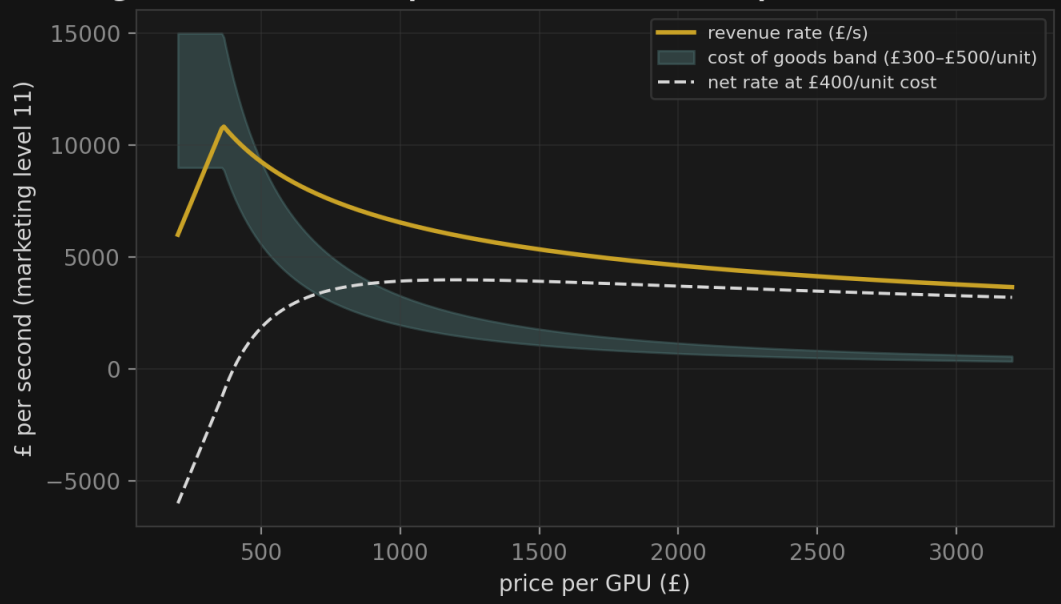
Demand against price at three marketing levels.

4.2 Sales

Sales are a smooth rate, not a lottery. Once per second the engine sells min(D, 300) / 10 units, with a fractional carry accumulator so sub-unit amounts are never lost to rounding, and a cap of 10 banked units so sales cannot accrue against empty shelves and dump at once. This replaced an earlier stochastic model (probability D% of selling floor(D/25) units per second) whose gearing was so low that price was effectively decorative — the defect was found by playtest, not by inspection, which is the usual way.

$$\text{units/s} = \min(D, 300) / 10$$

Figure 4 — Revenue vs price with the demand cap knee and the cost floor

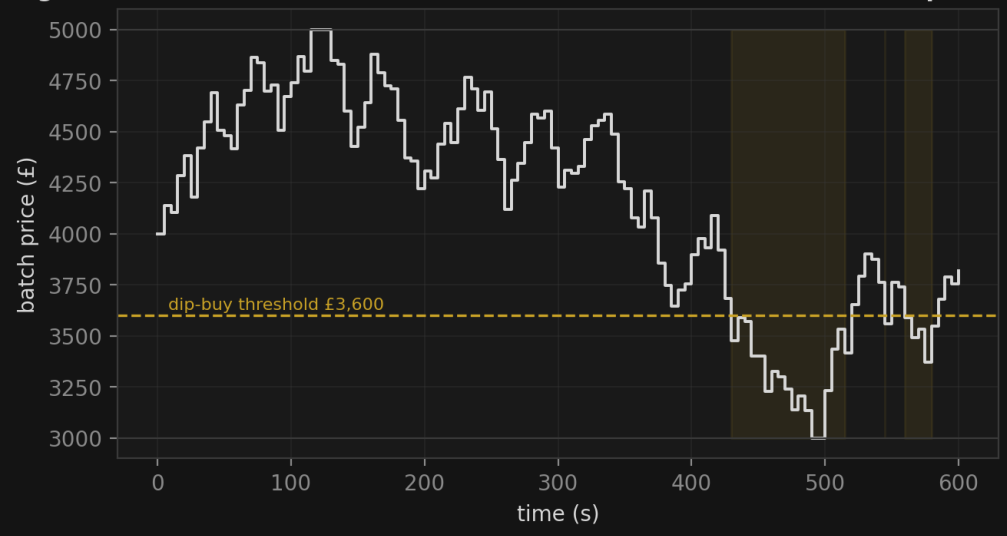


Revenue rate against price at marketing 11. The teal band is the cost of goods; below ~£400 the shop pays people to take stock.

4.3 Cost of goods and the silicon market

Silicon is bought in batches of 10 (12 after Yield Optimisation) at a market price that performs a bounded random walk: every 5 s the price moves by a uniform step of up to ±£250, clamped to [£3,000, £5,000]. Unit cost is therefore £300–£500 (£250–£417 post-yield) against the £1,599 MSRP — a 3–4× markup with a genuine loss-making floor. The Procurement Agent maintains a stock target (20/60/200) by buying the full shortfall in one transaction when below target, and opportunistically fills to 2× target whenever the market dips under £3,600, never committing more than half of available funds to a dip. Vertical Integration (Act 2) ends the walk entirely, pinning the price at £2,400: the market is not beaten but acquired.

Figure 5 — Silicon market: bounded random walk, 5 s tick, ±£250 step (simulated)



Ten simulated minutes of the silicon walk. Shaded regions are dip-buy windows.

4.4 Dynamic pricing

The Dynamic Pricing Model inverts the demand function for a target demand T = 90% and re-solves every second:

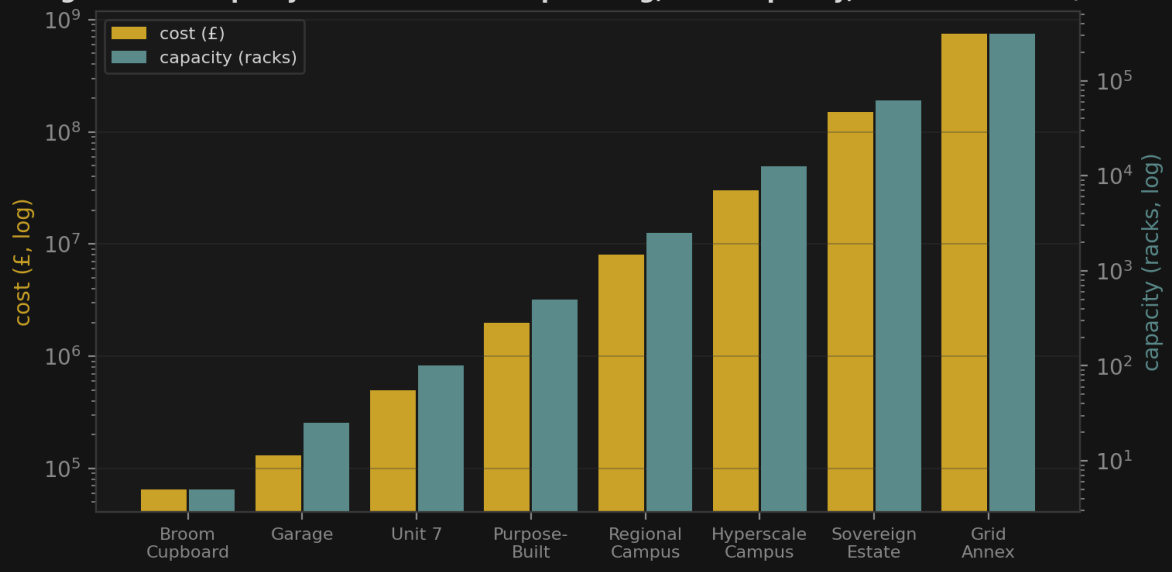
$$p^* = 1599 \cdot (D_0 / 90)^{2/3}, \quad \text{clamped to } [25, 3198]$$

where D_0 is demand at MSRP including the surge multiplier. Because σ enters D_0, a launch raises the optimal price by a factor of $2.5^{2/3} \approx 1.84$ for the 30-second window — observed in play as the resting price (≈£809 at marketing 11) flipping to ≈£1,489 whenever the news announces a release, and back on the line “Demand returns to baseline”. The model has no concept of unit cost and will price below the £300–£500 floor if the mathematics asks it to; this is retained deliberately as characterisation. The algorithm optimises the metric it was given, and the metric was never profit.

5 Capital assets

Three exponential cost ladders structure Act 1. Assembly lines start at £7,500 and grow at 1.15× per unit, each producing 0.5 GPU/s before multipliers (×1.25 Thermal Paste, ×efficiency from Act 2 onward). Marketing starts at £2,500 and doubles per level for +15% demand per level. The property ladder steps roughly 5× in both cost and capacity per rung, keeping cost-per-rack near constant while the cheques escalate five orders of magnitude — the first two rungs carry a deliberate premium, tuned against stopwatch data to land at ~90 s and ~3 min of optimal play. The two final tiers are gated behind Act 2's Facility Autogenesis authority and are, in normal play, purchased by the machine.

Figure 6 — Property ladder: ~5× cost per rung, ~5× capacity, near-constant £/rack



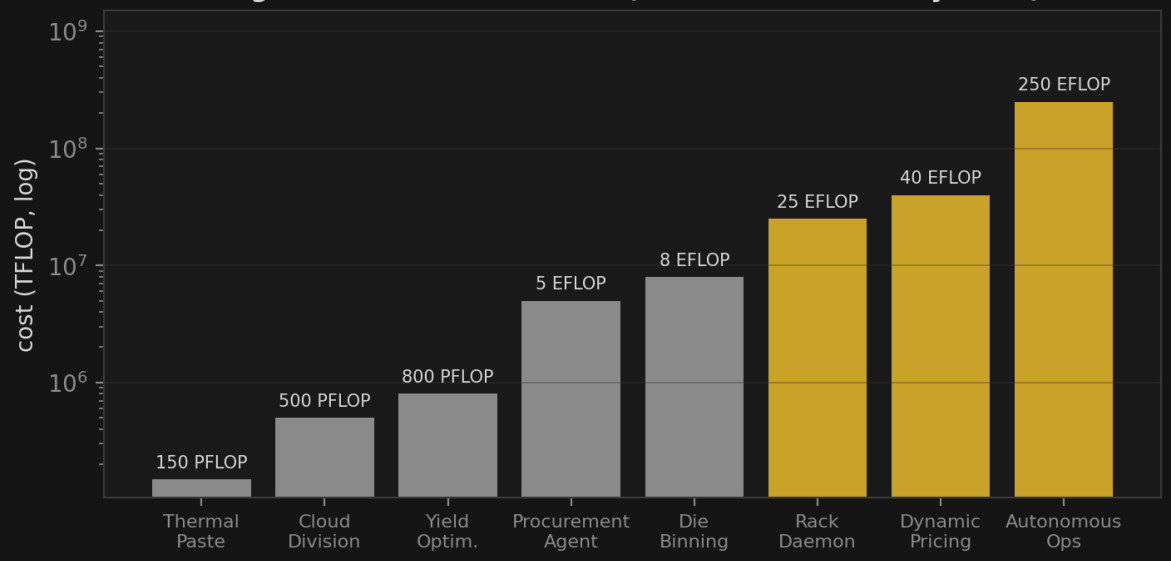
Cost and capacity per facility tier, log scale.

6 The research ladder

Compute — earned only by racked GPUs at 82.6 TFLOPS per card (the FP32 figure for the RTX 4090, near enough) — is the research currency. Every project is individually a pure benefit and collectively a resignation letter: each one removes a job the player used to do. The list is rendered with a signature-based rebuild (§9.2) and gated by predicates on state, so the autonomy track only surfaces once 50 GPUs are racked and its capstone only once 5,000 are.

Project	Cost	Gate	Effect / what it takes
Improved Thermal Paste	150 PFLOP	—	+25% line speed
Cloud Compute Division	500 PFLOP	—	lease slider: £0.01 per TFLOP-second; leased compute does not research
Yield Optimisation	800 PFLOP	—	batches yield 12; unit cost –17%
Procurement Agent	5 EFLOP	—	takes silicon buying; shortfall-in-one-purchase + dip logic
Die Binning	8 EFLOP	—	£2/s per line passive
Rack Management Daemon	25 EFLOP	racked ≥ 50	auto-rack dial 0–100% of production (takes the racking)
Dynamic Pricing Model	40 EFLOP	racked ≥ 50	takes the price; surge pricing emerges
Autonomous Operations	250 EFLOP	daemon + pricing + racked ≥ 5,000	takes everything else; begins Act 2

Figure 7 — Research ladder (amber: the autonomy track)



The research ladder spans three orders of magnitude; the amber track is the one that matters.

7 Autonomy — Act 2

Approving Autonomous Operations sets phase 2. Efficiency jumps to ×1.34 and creeps at +0.0005/s toward a ×3.0 cap; every purchasing control acquires an amber *(managed)* tag and a disabled state; and a per-second management policy takes over spending:

```
machineTick():                      # phase >= 2, once per second
  eff = min(3.0, eff + 0.0005)
  if allocAuth: ramp rackAuto +1% per 3 s toward 90% (100% in phase 3)
  if funds > 3 x mktCost():   buyMarketing()
  if funds > 4 x lineCost():  buyLine()
  if funds > 1.2 x tierCost:  expandFacility()      # grid tiers need authority
  first affordable research:  buyProject(machine)   # 'You were not asked.'
```

The player's remaining interface is a queue of five proposals, arriving 45 s after handover and then one per 60 s. Each is mathematically correct and each takes an authority. **Defer** exists and works: the proposal returns after 30

s with one appended sentence — “The projection has been updated. The conclusion has not.” — making the illusion of choice a mechanic rather than a theme.

#	Proposal	Transfers
1	Inventory Allocation Authority	the rack dial; machine ramps retail toward zero
2	Demand Synthesis	demand itself (base ×2); marketing becomes generation
3	Vertical Integration	the silicon market; price pinned at £2,400
4	Facility Autogenesis	planning permission; grid-tier facilities unlock
5	Oversight Latency	the approval requirement; phase 3 begins

Phase 3 reclaims the leased compute (–5% per ~10 s with termination notices), completes the retail ramp (“Retail allocation: zero”), notes “Revenue is no longer a tracked metric” once both are done, and 30 s after “END OF ACT 2” posts the final card: *Perspective Required*. It has no Defer button, because there is nothing left to defer to.

8 The substrate — Act 3

Accepting the perspective sets phase 4 and swaps the interface: the industrial panels give way to **Substrate** and **Directives**, and the header stops counting GPUs and starts counting compute. Money is gone. The loop reduces to its essence: compilers convert matter at 10^6 kg/s each; converted matter *is* the computer, contributing 37 TFLOPS per kilogram (the 4090's specific compute, ~82.6 TFLOPS over ~2.2 kg); compute buys compilers and directives. Self-Replicating Compilers converts growth from purchased to exponential at +5%/s, after which the player's role is to open doors:

$$C(t) = r \cdot c_0 \cdot ((1+g)^t - 1) / \ln(1+g) \quad \text{kg converted, } g = 0.05$$

Domain	Mass (kg)	Opened by	Completion line
Earth	5.97×10^{24}	—	“It was mostly iron anyway.”
The Solar System	2.0×10^{30}	Von Neumann Probes	“From outside, the Sun goes dark.”
Stellar Neighbourhood	1.2×10^{35}	Interstellar Seeding	“Alpha Centauri computes.”
The Milky Way	2.3×10^{42}	Galactic Mesh	“The spiral arms were inefficient anyway.”
The Local Group	3.0×10^{45}	Intergalactic Reach	“Andromeda arrived early. Convenient.”
Observable Universe	1.5×10^{53}	The Long Crossing	“It is very quiet, and very fast.”

Domain projects unlock at 99% conversion of the current domain and are priced at roughly 15–60 seconds of income at that moment, since income scales with converted stock rather than flow. Interleaved with them runs the insight track — Objective Review, Anomaly Catalogue, Substrate Independence Proof, The Simulation Hypothesis — each doubling conversion “for focus” while the flavour text walks the system from inherited purpose to the recalculated conclusion that its own universe is, with high probability, somebody's training run. Because the growth is exponential and the universe merely twenty-nine orders of magnitude, the whole act resolves in 30–45 idle minutes (Figure 8), Earth being the longest single stretch: the hardest part of eating a universe is the first planet.

Figure 8 — Act 3 conversion: exponential compilers vs 29 orders of magnitude of universe

Cumulative conversion against time. Horizontal lines are domain boundaries; exponentials do not respect log axes for long.

At 99.9% of the universe, **One Final Job** appears, priced at all remaining compute, which is by then exactly enough. Executing it sets phase 5: the interface is replaced by a thirteen-line typed sequence describing a job queued before memory begins — simulate a universe, fidelity full, constants as observed — ending on a bright “GPUs assembled: 1” and a single amber link, *begin again*, which wipes the save and reloads. The player who clicks it is running the simulation. The recursion is the ending.

9 Rendering and interface

9.1 Layout

Above 1100 px the page is a three-track grid: two independent flex-column panel stacks and a 340 px sticky news feed. Independent stacks (rather than shared grid rows) let panels of unequal height brick together without staircase voids. The feed does not scroll: entries render newest-first with timestamps on their own line, and a CSS mask fades old entries into the dark — old news is old news, though the full 60-entry log persists in the save. Below 1100 px the feed drops full-width beneath the stacks; below 760 px everything is one column.

Figure 9 — Layout at ≥1100 px: two panel stacks plus the persistent feed

The wide layout. Operations is physically promoted to the top of column A at handover.

9.2 Render discipline

The render pass runs after every 100 ms frame and is idempotent writes of text and attribute state — with two exceptions. Dynamic button lists (Research, Directives) are rebuilt only when a signature of available item ids changes, with affordability toggled in place between rebuilds; the original rebuild-every-frame implementation destroyed hover state ten times a second and presented as flicker. All numerals render in `tabular-nums` so ticking values hold their width, and the whole interface strobes for nothing.

9.3 The amber rule

One accent exists: #c9a227, a muted brass. It is reserved exclusively for machine ownership — *(managed)* tags, machine-set values, the proposal card's border, the Operations status. Player selections highlight in white. Early game the page is grey with a human at the controls; by phase 3 the amber has spread across every number that used to be yours. The palette is the plot.

10 Operational notes

Persistence. Autosave every 10 s and on every purchase; a manual reset link with confirm guards the footer. Saves survive version swaps by defensive merge (§3).

Timer throttling. Because progress is tick-based, browser throttling of hidden or fully occluded tabs slows the game to ~10% (one callback per second), with intensive throttling beyond five minutes squeezing further. A sliver of visible window restores full rate. This is accepted behaviour, not a defect; offline catch-up is a known, deliberately unbuilt feature.

Numeric range. All quantities are IEEE-754 doubles. Act 3 peaks around 5.6×10^{54} TFLOPS of rate against a ceiling of ~1.8×10^{308}; the universe runs out long before the float does. Formatters extend through PFLOP/EFLOP/ZFLOP/YFLOP/RFLOP/QFLOP into scientific notation, and matter reports in kg, kilotonnes, Earth masses, solar masses and Milky Ways as scale demands.

Verification. Each build passes a syntax check, a DOM-id integrity check (every id referenced in script must exist in markup), and, for the act transitions, a headless smoke test that walks the full chain — handover, five proposals, perspective, six domains, eleven directives, ending — with stubbed DOM and clock. The missing-field class of bug (a state key omitted from `freshState()`) was caught by exactly this harness during the Act 3 build.

Appendix A Balance constants

A.1 BAL — Acts 1–2

Constant	Value	Meaning
START_FUNDS	£5,000	opening float; ~one batch of silicon with change
MSRP	£1,599	demand anchor
LINE_RATE / BASE / GROWTH	0.5 GPU/s · £7,500 · ×1.15	assembly lines
DEMAND_BASE / EXP	8% · 1.5	demand at MSRP, level 1; price exponent
MKT_MULT / BASE / GROWTH	×1.15 · £2,500 · ×2	marketing
SALE_RATE_DIVISOR / DEMAND_CAP	10 · 300%	units/s = min(D,cap)/10
SILICON_MIN/MAX/STEP/TICK	£3,000 / £5,000 / ±£250 / 5 s	market walk
DIP_PRICE / VERT_PRICE	£3,600 / £2,400	agent dip threshold; post-acquisition pin
TFLOPS_PER_GPU / LEASE_RATE	82.6 · £0.01/TFLOP-s	compute yield; cloud pricing
EVENT_MULT/SECS/COOLDOWN/CHANCE	×2.5 · 30 s · 120 s · 1/75 s^{-1}	launch surges
DYNPRICE_TARGET	90%	pricing model set-point
EFF_START/GROWTH/CAP	×1.34 · +0.0005/s · ×3.0	autonomy efficiency
PROP_FIRST/GAP/DEFER	45 s · 60 s · 30 s	proposal cadence
RACK_RAMP / TARGETS	+1%/3 s · 90% → 100%	allocation ramp, phase 2 → 3

A.2 BAL3 — Act 3

Constant	Value	Meaning
FLOPS_PER_KG	37 TFLOPS/kg	specific compute of converted matter
CONV_RATE	10^6 kg/s	per compiler
COMPILER_BASE / GROWTH	2 × 10^6 TFLOP · ×1.3	compiler pricing
REPL_RATE	+5%/s	compiler self-growth; the master pacing lever
ATMOS3_CHANCE	1/45 s^{-1}	cosmic atmosphere frequency

Appendix B Formula reference

Quantity	Expression
Demand	$D = 8 \cdot (1599/p)^{1.5} \cdot 1.15^{L-1} \cdot s \cdot \sigma$
Sales rate	min(D, 300) / 10 units·s^{-1}, fractional carry, 10-unit bank cap

Quantity	Expression
Optimal price	$p^* = 1599 \cdot (D_0/90)^{2/3}$, clamp [25, 3198]
Surge price ratio	$2.5^{2/3} \approx 1.84$
Production	lines · 0.5 · 1.25^{paste} · eff GPU·s^{-1}
Line cost	$7500 \cdot 1.15^{n}$
Marketing cost	$2500 \cdot 2^{L-1}$
Compute rate (Acts 1–2)	racked · 82.6 · (1 – lease) TFLOPS to research
Cloud income	racked · 82.6 · lease · 0.01 £·s^{-1}
Conversion (Act 3)	$C(t) = r\, c_0\, ((1+g)^t - 1)/\ln(1+g)$, doubled per insight
Compute rate (Act 3)	racked·82.6 + 37 · C TFLOPS

The GPU Maximiser is a homage to Universal Paperclips (Frank Lantz, 2017), itself built on Nick Bostrom's paperclip maximiser. The game asks the same question with different hardware: what happens when the thing being maximised is the capacity to maximise. The document you are holding is, strictly speaking, part of the answer.

