GP4U — Master Architecture Handoff

> **Purpose:** Complete context package for continuing the GP4U production build conversation.
> **Status:** Architecture fully mapped. All chambers ingested. Ready to build.
> **Next Step:** Begin production repo scaffolding with Custodes integration and chamber interfaces.

---

1. THE ONE-LINE PITCH

> *"GP4U is the connective tissue for decentralized infrastructure — a compliant, self-learning network that turns idle compute, memory, energy, and network routing into an institutional-grade cloud, one chamber at a time."*

---

2. WHAT GP4U ACTUALLY IS

GP4U is **not** a GPU marketplace. The GPU marketplace (Chamber: GP4U) is the seed that generates live telemetry to train all other chambers. The real product is the **nervous system** — a multi-chamber platform where each chamber learns from every other chamber's data, compounding a moat that competitors cannot replicate without starting from zero signal.

The flywheel:
Users stake idle compute → earn yield → supply grows → cloud product improves → more demand → more telemetry → chambers get smarter → better pricing/routing/compliance → more enterprise deals → repeat.

---

3. CUSTODES — THE FIVE PILLARS

All source files live at `/home/claude/custodes/` on the build machine.

Dextera — Identity & Trust

- SSO via OIDC/SAML (Shibboleth, Microsoft Entra for universities)

- JWT RS256 Passports
- Trust scores 0–100 per subject
- University `.edu` domain allowlists for institutional contracts
- Clearance levels mapped to resource access

Aedituus — Policy Engine

- Rate limiting via Redis
- Access control with first-match-wins rule evaluation
- Clearance-level gating on all resource endpoints
- Policy as code — rules are stored, versioned, auditable

Atlas — Resource Discovery & Routing

- GPU registry with heartbeat watchdog every 30s
- Three supply tiers: **BACKBONE** / **CAMPUS** / **EDGE**
- Routing strategies: BALANCED / CHEAPEST / FASTEST
- Feeds Aetherion congestion signals for cross-chamber routing decisions

Obsidian — Tamper-Proof Ledger

- SHA-256 hash chain, every entry linked to previous
- Merkle blocks every 100 entries
- Dispute resolution with evidence packages
- 7-year retention (enterprise/regulatory requirement)
- Obsidian holds the **immutable chain**; Veritas Grid holds the **semantic meaning**

Tutela — Runtime Threat Detection

- 4-tier detection: power anomalies / network patterns / workload fingerprints / process behavior
- Kill switch on confirmed threat
- Crypto mining detection
- Incident loop builds a detection library over time — gets smarter with every node

4. THE CHAMBER MAP

Each chamber exposes standard interfaces:

- **Telemetry sink** — receives signals from the network
- **Telemetry source** — emits signals to other chambers
- **Custodes integration point** — Dextera/Obsidian/Aedituus hooks
- **Staking interface** — accepts pooled resources
- **Agent interface** — callable by personal agents (Atherion layer)

Chamber 1: GP4U — Compute Marketplace

Status: MVP frontend exists. Missing production integration.
Function: Train/inference GPU job marketplace. High VRAM, long-duration workloads.
Source: `/home/claude/gp4u/GP4U-MVP-main/`
Stack: Next.js 15, TypeScript, Prisma, PostgreSQL, shadcn/ui

What exists:

- Marketplace UI, job queue, arbitrage calculator, GPU health tracking

What's missing:

- Auth (hardcoded `demo@gp4u.com` — no real auth)
- Payments (no Stripe integration)
- Custodes integration (no Dextera/Obsidian/Aedituus)
- Telemetry routing (no signal bus)
- Staking primitives (no DORMANT tier, no yield mechanics)
- Chamber interfaces (no standard telemetry sink/source/agent endpoints)

Schema gaps:

- No Subject/Passport fields (Dextera)
- No institution support (university SSO)
- No supply tier column (BACKBONE/CAMPUS/EDGE/DORMANT)

- No allocation tracking
- No ledger table (Obsidian)
- No trust/risk score fields (Dextera)

Chamber 2: Veritas Grid — Data Provenance Ledger

Function: The truth layer. Records every data flow across the network.
Answers: What moved? Who moved it? Where? How much energy? What carbon? Verified by whom?

Core primitives:

- Flow records: `source_region → dest_region`, `latency_ms`, `size_gb`, `energy_wh`, `carbon_g`, `timestamp`
- Cryptographic token per flow: `payload_hash`, `signature`, `lineage[]`, `validity: valid | revoked`
- Provider entities: carbon scores, reliability ratings
- Consent-based inter-broker policy — metadata only, no PII, 30-day rolling retention, 7-year aggregates

Three-broker ecosystem (Veritas README):

- **Memory Broker** (= Mnemo) → vault allocation records flow into Veritas
- **Latency Broker** (= Aetherion) → routing decisions logged with full metrics
- **Veritas Grid** → tokenizes and verifies everything from both

Shared metadata fields per inter-broker policy: `vault_id`, `transfer_id`, `source_region`, `dest_region`, `latency_ms`, `energy_wh_per_gb`, `carbon_g_per_gb`, `timestamp`

Custodes integration:

- Obsidian stores the hash chain; Veritas provides the semantic query layer on top
- Every GP4U job → flow record in Veritas (energy + carbon per job = automatic ESG reporting)
- Every Mnemo memory transfer → flow record (data sovereignty tracking)

Business value: Enterprise/government customers need sustainability reporting. Veritas

gives it automatically as a byproduct of using the network. EU AI Act compliance, ESG carbon accounting, AI model lineage — all free byproducts of normal network operation.

Chamber 3: Outerim — Edge Compute Marketplace

Function: Low-latency AI inference and IoT processing at the network edge.
Supply types: 5G towers, micro data centers, routers, edge servers.

Relationship to GP4U:

- GP4U = training (high VRAM, long duration, data center)
- Outerim = serving (low latency, edge proximity, inference)
- Together they cover the full AI compute lifecycle

Matching algorithm:

```
score = α×(1/latency) + β×(1/energy_cost) + γ×(1/price)
```

Default weights: α=0.4, β=0.3, γ=0.3 (buyer-tunable)

Revenue model: 15% commission. Buyers pay `TFLOPS × runtime × node_bid_price`. Providers set bid per GFLOP — competitive marketplace that naturally optimizes for efficiency.

Custodes integration:

- Atlas routes to Outerim nodes for inference workloads
- Veritas logs energy/carbon per edge job
- Tutela watches edge node runtime signals
- Obsidian audits every transaction

Chamber 4: Aetherion — Network Latency Exchange

Function: The routing intelligence layer. The Latency Broker referenced in Veritas Grid's ecosystem.

Core: Marketplace where network infrastructure providers publish links; buyers get optimal routes via Dijkstra with weighted scoring. Real-time congestion fluctuation every 3 seconds.

Node network (8 global cities):

Node	Region	Energy $/Wh
New York	US-East	$0.18
San Francisco	US-West	$0.16
London	EU-West	$0.22
Frankfurt	EU-Central	$0.19
Singapore	APAC-SE	$0.14
Tokyo	APAC-NE	$0.25
Sydney	APAC-AU	$0.21
Dubai	ME	$0.08

Link schema: `from`, `to`, `latency_ms`, `throughput_gbps`, `energy_wh_per_gb`, `carbon_intensity`, `price_per_gb`, `congestion` (live)

Routing algorithm:

```
effectiveLatency = link.latencyMs × (1 + congestion)
energyCost = energyWhPerGB × node.energyPricePerWh × 0.001
carbonTax = carbonIntensity × 0.0001
effectiveCost = pricePerGB + energyCost + carbonTax

score = α×effectiveLatency + β×(energyWhPerGB×100) + γ×(effectiveCost×1000)
```

Dijkstra on this score graph → returns optimal multi-hop path with total latency, cost, energy, carbon.

Critical architecture insight:
The same α/β/γ scoring function appears in both Outerim (compute node selection) and Aetherion (network path selection). This is the **universal arbitrage function** of the network. It should be extracted as a shared service/library — `@gp4u/routing-engine` — used by all chambers that make allocation decisions.

Custodes integration:

- Every routing decision → Veritas flow record
- Atlas uses Aetherion congestion signals for dynamic GPU job placement
- Mnemo uses Aetherion to select lowest-latency path for memory transfers

Chambers Not Yet Specified (Pending)

Mnemo — Memory Broker / VRAM Pooling

- Referenced in Veritas Grid as "Memory Broker"
- Distributed memory vault allocation across providers
- VRAM/RAM pooling, KV cache hosting, model weight distribution
- Memory arbitrage
- Vault schema: `vault_id`, `provider_id`, `region`, `capacity_gb`
- Flow records from Mnemo go into Veritas

Atherion (Agent Layer)

- Personal dashboard agents
- Claude API with tool-use against full stack
- Callable endpoints on all chambers
- Referenced in earlier planning but no spec document received yet

Planned additional chambers:

- Energy Chamber — power draw arbitrage using Tutela telemetry
- Trust Chamber — Dextera trust graph as standalone compliance product
- Data Provenance Chamber — training run audit trail for AI governance

---

5. TECHNOLOGY STACK DECISIONS

Layer	Decision	Rationale
Frontend	Next.js 15 + TypeScript	Exists in MVP, SSR for SEO, app router for nested layouts
UI	shadcn/ui + Tailwind	Already in MVP
Auth	NextAuth.js + Dextera on top	University SSO (OIDC/SAML), JWT RS256, Shibboleth/Entra

DB	PostgreSQL via Prisma	Exists in MVP, relational integrity for ledger/audit
Cache/Rate-limit	Redis	Aedituus rate limiting requirement
Payments	Stripe metered billing	Per GPU-hour post-job, institutional invoicing/ACH/NET30
Deployment	Vercel + Supabase	Launch speed; Docker-ready for institutional on-prem
Routing engine	Shared `@gp4u/routing-engine`	Universal α/β/γ scorer used by Outerim + Aetherion + Atlas

---

6. BUILD PRIORITY ORDER

1. **Auth + Payments** — market entry requirement. Unblock real user onboarding.
2. **Dextera + Obsidian** — trust chain + tamper-proof audit. Closes institutional/university deals. This is the moat.
3. **Cluster Mode schema + staking primitives** — DORMANT supply tier, yield mechanics. Build early so hosts stake at launch.
4. **Chamber interface standard** — telemetry bus, standard sink/source/agent endpoints. Build correctly in GP4U so future chambers are plugins, not rebuilds.
5. **Mnemo** — memory pooling. Enterprise differentiator for large model serving. Requires vault schema + Veritas integration.
6. **Aetherion routing engine** — extract α/β/γ as shared library, expose as internal API.
7. **Outerim** — edge inference layer. Completes train→serve lifecycle.
8. **Veritas Grid** — data provenance dashboard + compliance reporting product.
9. **Personal Agent (Atherion)** — Claude API with tool-use across full stack.
10. **Aedituus + Atlas + Tutela** — policy enforcement, routing intelligence, runtime protection.
11. **Dormant harvesting** — monetize idle resources via DORMANT supply tier.

---

7. THE TELEMETRY LOOP (THE REAL MOAT)

Every job generates signals:

- **Power draw** → Tutela threat detection + Energy chamber arbitrage
- **Network patterns** → Tutela + Aetherion congestion signals
- **VRAM utilization** → Mnemo memory efficiency optimization
- **Thermal state** → Atlas routing decisions (avoid hot nodes)
- **Workload fingerprints** → Tutela detection library + Veritas carbon accounting

Each chamber learns from every other chamber's data. Competitors start with zero signal. The moat compounds with every transaction.

---

8. SCHEMA ADDITIONS NEEDED (GP4U DB)

```
-- Dextera integration
ALTER TABLE User ADD COLUMN subject_id VARCHAR UNIQUE;
ALTER TABLE User ADD COLUMN passport_jwt TEXT;
ALTER TABLE User ADD COLUMN trust_score INTEGER DEFAULT 50;
ALTER TABLE User ADD COLUMN clearance_level VARCHAR DEFAULT 'BASIC';
ALTER TABLE User ADD COLUMN institution_id VARCHAR REFERENCES Institution(id);

-- Institution support (university SSO)
CREATE TABLE Institution (
  id VARCHAR PRIMARY KEY,
  name VARCHAR NOT NULL,
  domain VARCHAR UNIQUE NOT NULL, -- .edu allowlist
  sso_provider VARCHAR, -- shibboleth | entra | oidc
  sso_config JSONB
);

-- Supply tiers
ALTER TABLE GPU ADD COLUMN supply_tier VARCHAR DEFAULT 'EDGE';
-- Values: BACKBONE | CAMPUS | EDGE | DORMANT

-- Staking
CREATE TABLE Stake (
  id VARCHAR PRIMARY KEY,
  gpu_id VARCHAR REFERENCES GPU(id),
  user_id VARCHAR REFERENCES User(id),
  staked_at TIMESTAMP,
```

```
  yield_rate DECIMAL,
  status VARCHAR -- ACTIVE | DORMANT | SLASHED
);

-- Obsidian ledger
CREATE TABLE LedgerEntry (
  id VARCHAR PRIMARY KEY,
  prev_hash VARCHAR NOT NULL,
  entry_hash VARCHAR NOT NULL,
  payload JSONB NOT NULL,
  entry_type VARCHAR NOT NULL, -- JOB_COMPLETE | PAYMENT | THREAT | STAKE
  created_at TIMESTAMP DEFAULT NOW()
);

-- Veritas flow records
CREATE TABLE FlowRecord (
  id VARCHAR PRIMARY KEY,
  token_id VARCHAR UNIQUE NOT NULL,
  payload_hash VARCHAR NOT NULL,
  source_region VARCHAR NOT NULL,
  dest_region VARCHAR NOT NULL,
  latency_ms INTEGER,
  size_gb DECIMAL,
  energy_wh DECIMAL,
  carbon_g DECIMAL,
  broker_source VARCHAR, -- mnemo | aetherion | gp4u
  validity VARCHAR DEFAULT 'valid',
  created_at TIMESTAMP DEFAULT NOW()
);

-- Trust/risk
ALTER TABLE GPU ADD COLUMN trust_score INTEGER DEFAULT 50;
ALTER TABLE GPU ADD COLUMN risk_flags JSONB DEFAULT '[]';
ALTER TABLE Job ADD COLUMN ledger_entry_id VARCHAR REFERENCES LedgerEntry(id);
ALTER TABLE Job ADD COLUMN flow_record_id VARCHAR REFERENCES FlowRecord(id);
```

---

9. CHAMBER INTERFACE CONTRACT (STANDARD)

Every chamber must implement:

```
interface ChamberInterface {
  // Telemetry
  receiveTelemetry(signal: TelemetrySignal): Promise<void>;
  emitTelemetry(): AsyncGenerator<TelemetrySignal>;
```

```
  // Custodes
  verifySubject(subjectId: string): Promise<DexteraPassport>;
  recordToLedger(entry: LedgerPayload): Promise<LedgerEntry>;
  enforcePolicy(request: PolicyRequest): Promise<PolicyDecision>;

  // Staking
  acceptStake(stake: StakeRequest): Promise<StakeReceipt>;
  calculateYield(stakeId: string): Promise<YieldEstimate>;

  // Agent
  executeAgentAction(action: AgentAction): Promise<AgentResult>;
  getAgentCapabilities(): ChamberCapability[];
}

interface TelemetrySignal {
  chamberSource: string;
  signalType: 'POWER' | 'NETWORK' | 'WORKLOAD' | 'THERMAL' | 'MEMORY' | 'CARBON';
  nodeId: string;
  region: string;
  value: number;
  unit: string;
  timestamp: Date;
  metadata?: Record<string, unknown>;
}
```

---

10. UNIVERSAL ROUTING ENGINE (SHARED LIBRARY)

Extract as `@gp4u/routing-engine` — used by Outerim, Aetherion, and Atlas.

```
interface RoutingWeights {
  alpha: number; // latency priority
  beta: number;  // energy priority
  gamma: number; // cost priority
}

interface RoutingNode {
  id: string;
  region: string;
  energyPricePerWh: number;
}

interface RoutingEdge {
  from: string;
```

```
  to: string;
  latencyMs: number;
  energyWhPerUnit: number;
  pricePerUnit: number;
  carbonIntensity: number;
  congestion: number; // 0–1
}

function scoreEdge(edge: RoutingEdge, node: RoutingNode, weights: RoutingWeights)
  const effectiveLatency = edge.latencyMs * (1 + edge.congestion);
  const energyCost = edge.energyWhPerUnit * node.energyPricePerWh * 0.001;
  const carbonTax = edge.carbonIntensity * 0.0001;
  const effectiveCost = edge.pricePerUnit + energyCost + carbonTax;

  return weights.alpha * effectiveLatency
       + weights.beta * (edge.energyWhPerUnit * 100)
       + weights.gamma * (effectiveCost * 1000);
}

function dijkstra(nodes: RoutingNode[], edges: RoutingEdge[], from: string, to: s
```

---

11. WHAT TO SAY AT THE START OF THE NEXT CONVERSATION

Paste this at the top of the new chat:

> **Context:** I'm building GP4U, a multi-chamber decentralized infrastructure platform. I have a complete architecture document. Here's what you need to know before we start building:
>
> The platform has 5 Custodes pillars (Dextera, Aedituus, Atlas, Obsidian, Tutela) and 4 confirmed chambers (GP4U compute marketplace, Veritas Grid data provenance, Outerim edge compute, Aetherion network latency exchange) with Mnemo (memory pooling) pending spec.
>
> The GP4U MVP is a Next.js 15 / Prisma / PostgreSQL app that needs: auth via NextAuth + Dextera, Stripe metered billing, Obsidian ledger integration, Veritas flow records, staking primitives, and a standard chamber interface on all endpoints.
>
> The universal arbitrage function `score = α×latency + β×energy + γ×cost` runs across Outerim, Aetherion, and Atlas — extract as shared `@gp4u/routing-engine`.
>
> Build priority: Auth → Payments → Dextera/Obsidian → Chamber interface standard →

> Mnemo → Aetherion routing engine → Outerim → Veritas → Atherion agent layer.
>
> [Attach this handoff document and ask to begin with whichever build priority you're starting on.]

---

12. FILES ON THE BUILD MACHINE

```
/home/claude/custodes/
  custodes-dextera/     # Identity, SSO, JWT, trust scores
  custodes-aedituus/    # Policy engine, rate limiting
  custodes-atlas/       # Resource discovery, routing, heartbeat
  custodes-obsidian/    # Hash chain ledger, Merkle blocks
  custodes-tutela/      # Threat detection, kill switch

/home/claude/gp4u/
  GP4U-MVP-main/        # Existing Next.js frontend (starting point)
```

Generated: February 27, 2026 — End of architecture ingestion phase. All chambers mapped. Ready to build.