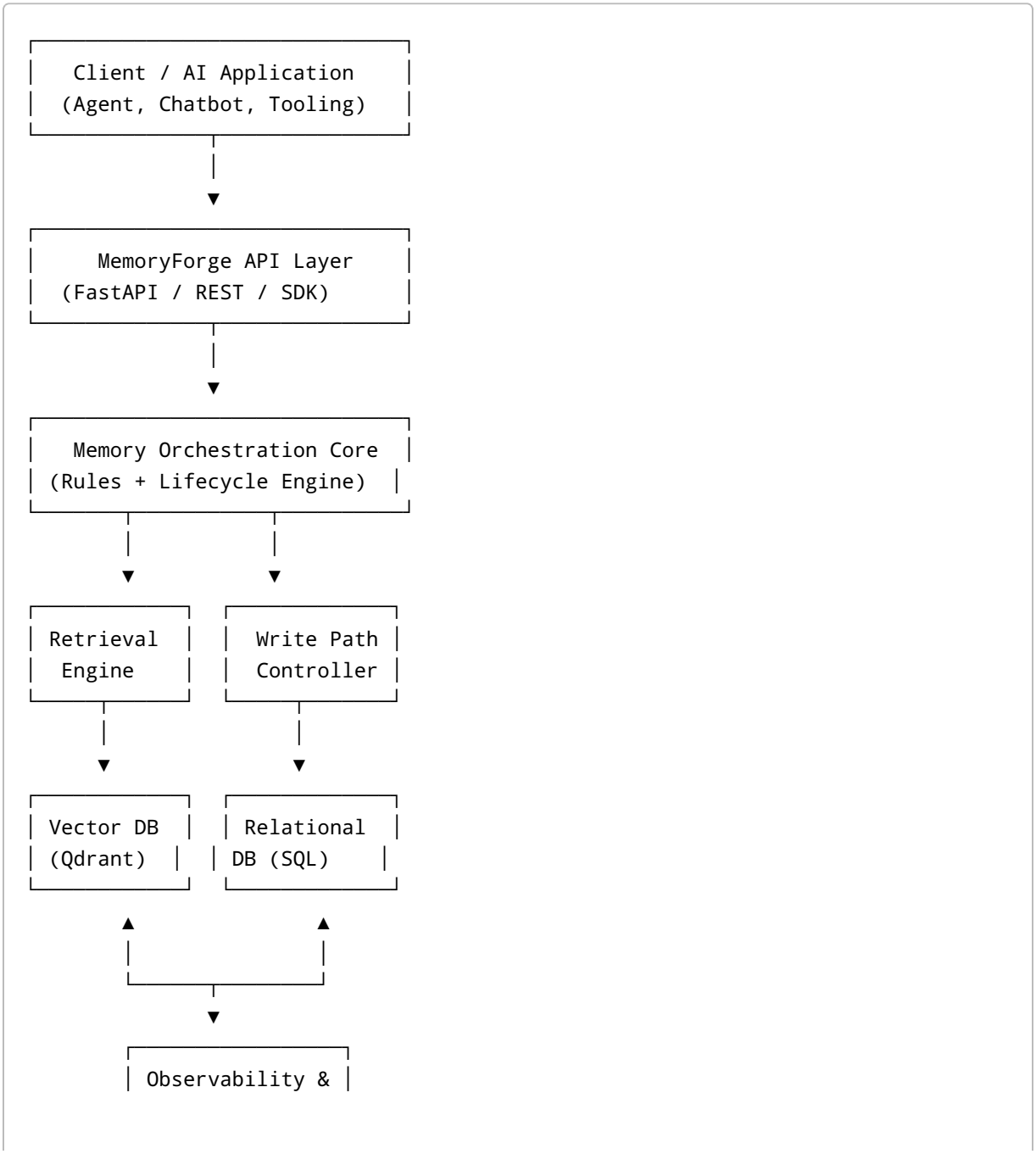
Smriti – System Architecture (Production-Ready)

This architecture is designed to be **buildable, debuggable, and trustworthy**. It avoids premature scale while enforcing correctness, memory control, and explainability.

---

1. High-Level Architecture (Mental Model)



```
        │ Audit Log       │
        └─────────────────┘
```

---

2. Core Architectural Principles (Non-Negotiable)

1. **Rules before LLMs**
2. **Memory is versioned, never overwritten**
3. **Reads are cheap, writes are expensive**
4. **Every decision is explainable**
5. **Failure is visible, not silent**

---

3. Component Breakdown

3.1 API Layer

Responsibilities:

- Authentication / namespace isolation
- Input validation
- Rate limiting
- SDK abstraction

Tech:

- FastAPI
- Pydantic schemas

No business logic here.

3.2 Memory Orchestration Core (Heart of V1)

This is the system's brain.

Responsibilities:

- Enforce memory lifecycle rules
- Decide read vs write vs update
- Route to correct memory type
- Enforce confidence thresholds

Important: This layer does NOT store data. It controls flow.

3.3 Memory Write Path (Controlled & Defensive)

```
Input → Validation → Classification → Rule Check → LLM Assist → Persist → Audit
```

Key rules:

- LLM suggestions are advisory
- Conflicts create new versions
- Deletes require explicit approval

Writes are slow by design.

3.4 Retrieval Engine (Context-Aware)

Capabilities:

- Memory-type filtering
- Temporal constraints
- Active/inactive filtering
- Relevance thresholding

Output:

- Ranked, minimal memory set
- Token-budget-aware

This replaces naive RAG.

---

4. Memory Types & Storage Mapping

Memory Type	Storage	Mutability
Working	In-memory + SQL	Mutable
Episodic	SQL	Immutable
Semantic	SQL + Vector	Versioned
Metadata	SQL	Mutable

Graph DB is intentionally excluded in V1.

---

5. Storage Layer

5.1 Relational DB (Source of Truth)

Stores:

- Memory records
- Versions
- Confidence scores
- Status (active/inactive)
- Source & timestamps

Why SQL:

- Strong consistency
- Easy audits
- Human debuggable

5.2 Vector DB (Similarity Only)

Stores:

- Embeddings for semantic memory

Used ONLY for:

- Candidate recall

Never used for authority.

---

6. LLM Interaction Layer

LLMs are used for:

- Summarization
- Classification
- Suggesting memory updates

LLMs are NOT used for:

- Deletion
- Final conflict resolution
- Truth authority

All LLM outputs must include confidence.

---

7. Contradiction Handling Flow

```
New Fact → Similarity Search → Conflict Detection
         → Version Creation
         → Resolution Policy Applied
         → One Active, Others Inactive
```

Nothing is lost.

---

8. Observability & Trust Layer

Every memory event logs:

- Input
- Decision path
- Rules triggered
- LLM output
- Final action

This is critical for production trust.

---

9. Deployment Model (V1)

Supported:

- Single-node deployment
- Docker-based
- Local-first

Explicitly NOT required:

- Kubernetes
- Kafka
- GPUs

---

10. Why This Architecture Will Work

Because it:

- Minimizes moving parts

- Prioritizes correctness over speed
- Treats memory as state, not text
- Gives humans control

Senior Engineer’s Rule of Thumb

If you can’t explain why a memory exists, the system is already broken.

This architecture prevents that.