

[LOGO_PLACEHOLDER]

UET PLATFORM **MASTER BLUEPRINT v3.0**

The Definitive Architecture for a Knowledge Operating System

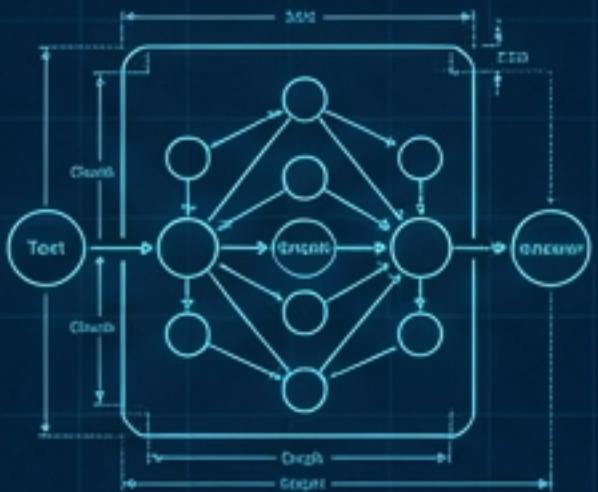
PURPOSE: เหตุผลที่ระบบนี้ต้องมีอยู่

ระบบ UET Platform ไม่ได้ถูกออกแบบมาเพื่อเป็นเพียง Chatbot แต่คือสถาปัตยกรรมที่ทำหน้าที่ 5 ประการหลัก:



Knowledge Operating System

ทำหน้าที่เป็นตัวกลางจัดการองค์ความรู้ทั้งหมดของผู้ใช้



Precision-RAG + Unified Knowledge Graph

ผ่าน Text → Chunk → Embed → Graph → Agent → Answer อย่างเป็นระบบ (Deterministic)



Agent Orchestration System

ไม่ใช่แค่ 'LLM Chat' แต่เป็นระบบควบคุมการทำงานของ Agent ทุกขั้นตอน



Execution-Graph AI

AI ที่มีกระบวนการคิดแบบ Pipeline มีสถานะ (State) และกฎเกณฑ์ที่ชัดเจน ไม่ใช่การเดาสุ่ม



Scalable Product

ออกแบบมาเพื่อรับรองรับผู้ใช้งานจำนวนมากและไม่ล่มเมื่อมีภาระงานสูง

THE LAWS OF PHYSICS: กฎเหล็กที่ทั้งระบบและทีมต้องปฏิบัติตาม

เพื่อให้สถาปัตยกรรมนี้คงอยู่ได้อย่างยั่งยืน เรามี 'Contract' สองระดับที่ทุกคนและทุกส่วนของโคดต้องยึดถือ



SYSTEM CONTRACT (กฎของระบบ)

'Everything Must Obey'

- ✓ 1. **Deterministic System:** ให้ Input เมื่อันกัน ต้องได้ผลลัพธ์เหมือนกันเสมอ
- ✓ 2. **Blueprint as Source of Truth:** การอัปเดตทุกอย่างต้องเริ่มที่ Master Blueprint และวิ่งกระจาย (cascade) ไปยังไฟล์ลูกทั้ง 25ไฟล์
- ✓ 3. **No Unauthorized Rewrites:** ห้ามเขียนทับ (rewrite) ไฟล์ลูกโดยไม่ว่าจะมาจาก Master Blueprint
- ✓ 4. **Aligned Components:** ไฟล์ลูกทั้ง 25 ไฟล์ต้องสอดคล้อง (align) กับพิมพ์เขียวที่แนบมา
- ✓ 5. **No Rogue Components:** ไม่มีไฟล์ลูกใดทำตัวเป็นเจ้าของระบบ



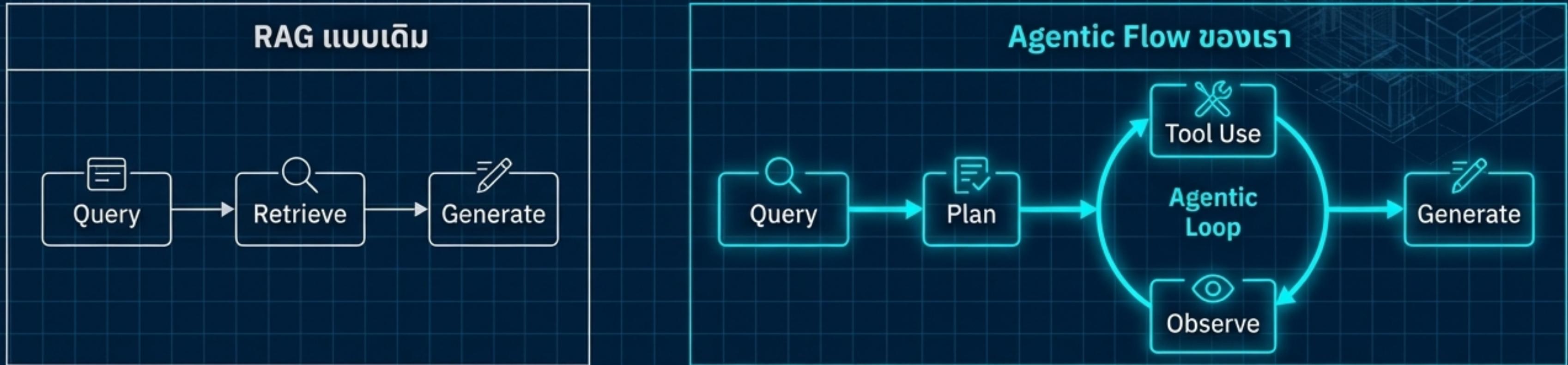
TEAM CONTRACT (กฎการทำงานร่วมกัน)

'Discipline Creates Clarity'

- ✓ 1. **Markdown Only:** เอกสารทั้งหมดต้องเป็นฟอร์แมต `.md` เท่านั้น
- ✓ 2. **Consistent Updates:** อัปเดตที่ไฟล์เดิมเท่านั้น ห้ามสร้างไฟล์ part 2, part 3 หรือเวอร์ชันซ้อน
- ✓ 3. **Singular Knowledge:** ความรู้ใหม่ต้องเขียนทับ (override) ความรู้เดิมเท่านั้น ห้ามตีความหรือแต่งเติมเอง
- ✓ 4. **Language Consistency:** ใช้ 'ภาษาของ Santa' เป็นหลัก ห้ามใช้ศัพท์ที่ล้าสมัย

BEYOND RAG: ทำไมเราถึงเลือกสถาปัตยกรรมแบบ Agentic Orchestration

ระบบของเราท้าวข้าม RAG แบบเดิมๆ ไปสู่ Agentic AI ซึ่งมีความสามารถในการ ‘คิด-วางแผน-ลงมือทำ’ อย่างเป็นระบบ เหตุผลหลักคือ:



Agents as Specialized Workers (Agent คือผู้เชี่ยวชาญเฉพาะทาง)

Agent แต่ละตัวมีหน้าที่และความรับผิดชอบที่ชัดเจน (เช่น Research Agent, Analysis Agent, SQL Agent) ทำให้ง่ายต่อการจัดการและแก้ปัญหาที่ซับซ้อน (อ้างอิงแนวคิดจาก MarkTechPost และ DataTH)

“agent มันคือแอปพลิเคชันตัวหนึ่ง...ซึ่งมันมีความเป็นมีความคิด มีการวางแผน มีการใช้ reasoning มีเหตุมีผลภายในตัว เพื่อให้บรรลุเป้าหมาย” - DataTH

State-Driven Workflows (Workflow ที่ขับเคลื่อนด้วยสถานะ)

เราไม่ได้ใช้แค่ Chain แบบเส้นตรง แต่สร้างเป็น Graph ที่มีสถานะ (State) ทำให้สามารถสร้างเงื่อนไข (Conditional Edges) วนลูป (Cycles) และจัดการ Flow ที่ซับซ้อนได้ (อ้างอิงแนวคิดจาก LangGraph)

Tool-Use as a Core Capability (การใช้เครื่องมือคือหัวใจสำคัญ)

Agent สามารถเรียกใช้เครื่องมือภายนอกได้ (เช่น Web Search, Database Query, API) เพื่อหาข้อมูลเพิ่มเติมและทำงานที่ LLM อย่างเดียวทำไม่ได้ (อ้างอิงแนวคิดจาก Open Data Science Conference)

OUR FRAMEWORK OF CHOICE: LangGraph

เพื่อสร้างระบบ Agentic ที่เป็น Graph และมีสถานะ (Stateful) เราจึงเลือกใช้ LangGraph ซึ่งเป็นໄสารีสำหรับสร้างแอปพลิเคชันแบบ Multi-Actor โดยเฉพาะ

Nodes

គឺជាកិច្ចការឱ្យឯកសារ 'Agent' កំណត់ដោយខ្លួន



Edges

គឺជាលេខការណ៍ដែលរួចរាល់ពី Node មួយទៅ Node ផ្សេងៗ ដើម្បីផ្តល់សំណង់ទៅ State Object

Conditional Edges

គឺជាលេខការណ៍ដែលត្រូវបញ្ជាក់ថាបានការពិនិត្យក្នុងការបង្កើតការងារ ដើម្បីធ្វើការណ៍ដែលត្រូវបានការពិនិត្យ

Why LangGraph over simple Chains?



Supports Cycles: ជាដំឡើងក្រប់ក្រង់ការងារដែលត្រូវបានការពិនិត្យ ដើម្បីធ្វើការងារជាមួយគ្នា (Iterative processes) ដែលមានការការពិនិត្យ (Reasoning) និងការបន្ថែម (Acting).

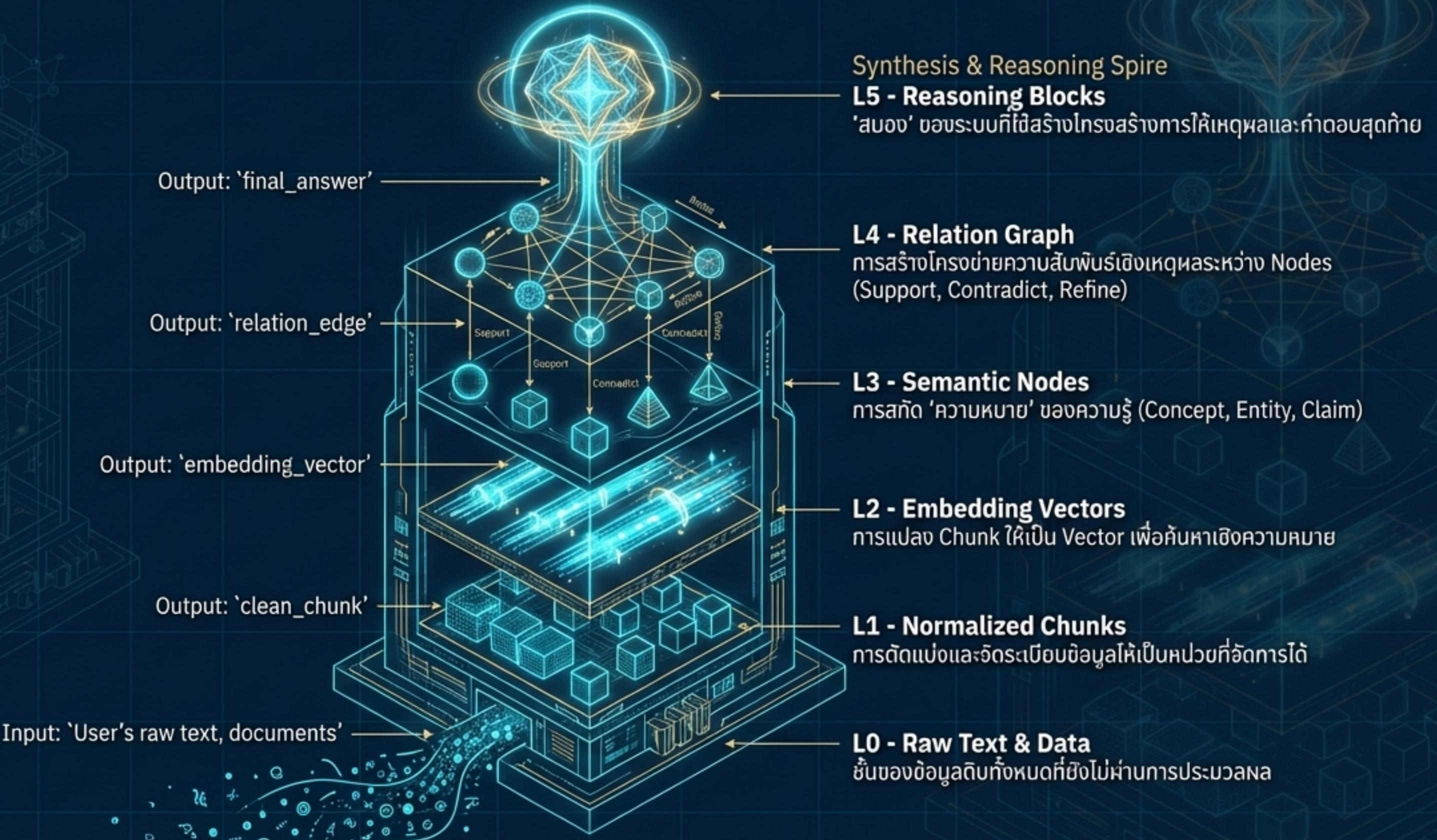


State Management: ការបង្កើតការងារដែលមានការផ្តល់សំណង់ទៅគ្រប់គ្រងការងារ ដើម្បីបង្កើតការងារដែលមានការផ្តល់សំណង់ទៅគ្រប់គ្រងការងារ.



Control & Visibility: ការបង្កើតការងារដែលមានការផ្តល់សំណង់ទៅគ្រប់គ្រងការងារ ដើម្បីបង្កើតការងារដែលមានការផ្តល់សំណង់ទៅគ្រប់គ្រងការងារ.

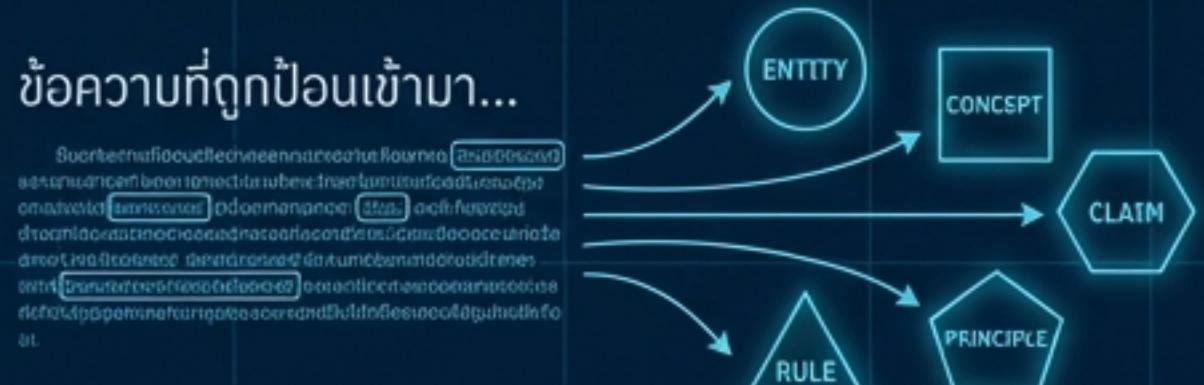
THE ARCHITECTURE: โครงสร้าง 6 ชั้น จากข้อมูลดิบสู่สมองของระบบ



DEEP DIVE: The Knowledge Core (L3-L5)

สามชั้นบนสุดคือส่วนที่เปลี่ยน ‘ข้อมูล’ ให้กลายเป็น ‘ปัญญา’ ที่สามารถให้เหตุผลได้

ข้อความที่ถูกป้อนเข้ามา...

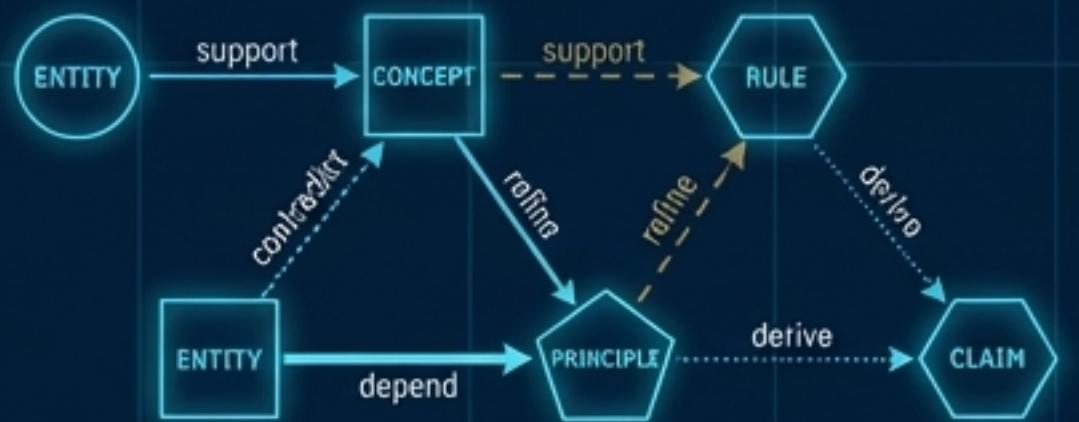


L3: Semantic Nodes - The Atoms of Meaning

What it does សកែដុល្លារទិន្នន័យទីតាំងរបស់ភ្នែកជាមួយគ្នា

Examples ‘concept’, ‘entity’, ‘claim’, ‘rule’, ‘principle’

Why it matters ทำให้เราเข้าใจว่า ‘ครก็จะอะไรก็ได้’ และ ‘แนวคิดหลักคืออะไร’



L4: Relation Graph - The Web of Logic

What it does เดี๋ยวจะมาดู Semantic Nodes กันต่อไป

Edge Types `support`, `contradict`, `refine`, `depend`, `derive`

Why it matters สร้าง ‘Knowledge Web’ ที่เป็นเหตุเป็นผลอธิบาย ไปใช้แค่ Graph ที่เชื่อมกันมั่วๆ



L5: Reasoning Blocks - The Engine of Synthesis

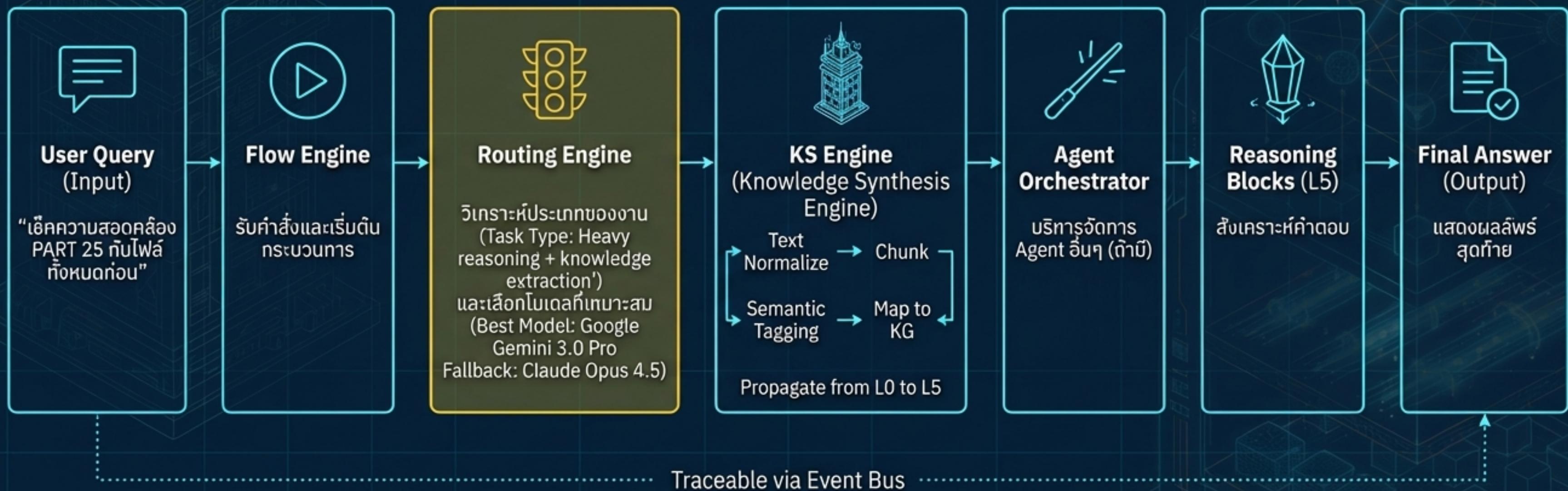
What it does นำ Knowledge Graph มาสั่งเคราะห์และสร้างเป็นค่าตอบสุดท้าย

Components ‘argument structure’, ‘reasoning pattern’, ‘synthesis block’

Why it matters นี่คือ 'สมอง' ที่ประกอบขึ้นส่วนความรู้ก็ยังคงให้กลไกเป็นคำตอบที่สมบูรณ์

A DAY IN THE LIFE: End-to-End Deterministic Flow

นี่คือเส้นทางการเดินทางของคำสั่ง (Query) ผ่านสถาปัตยกรรมของเรา ทุกขั้นตอนสามารถติดตามและตรวจสอบได้ (Traceable) ผ่าน Event Bus



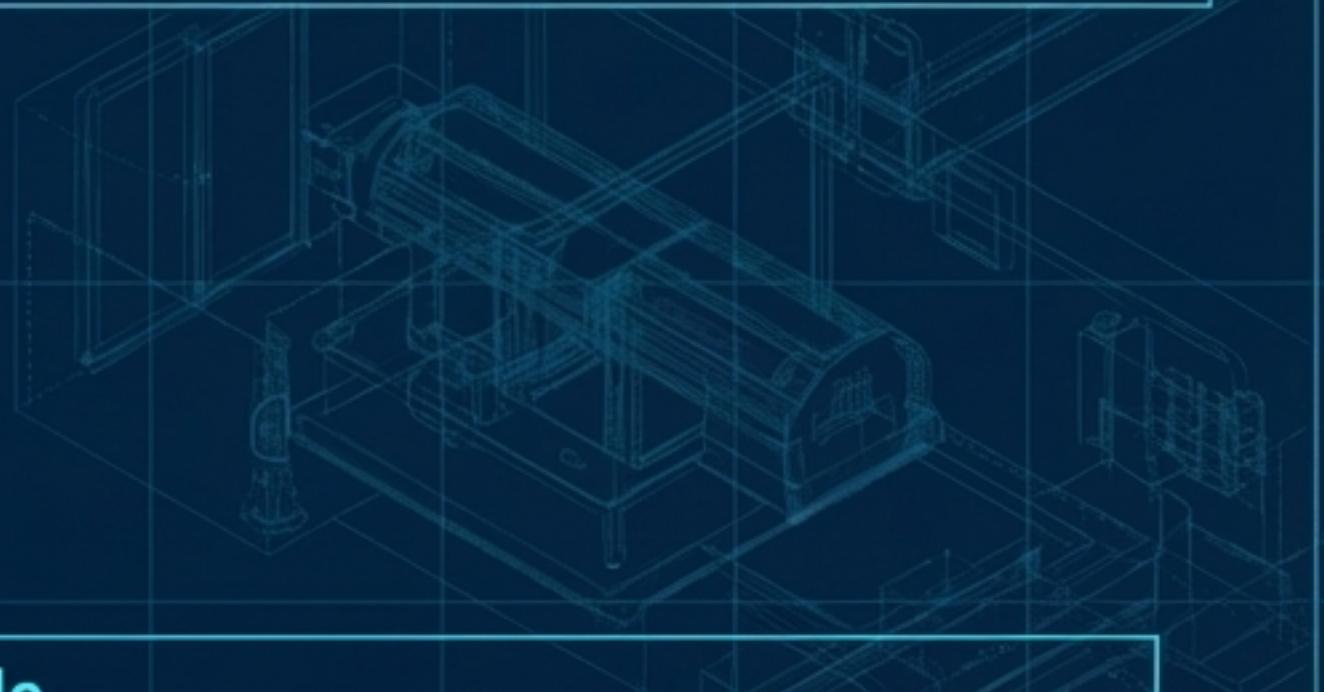
INTELLIGENT ROUTING: The Right Tool for the Right Job

เราไม่ใช้โมเดลเดียวสำหรับทุกงาน Routing Engine ทำหน้าที่เป็น 'ผู้จัดสรรงาน' อัจฉริยะเพื่อเลือกโมเดลที่เหมาะสมที่สุดตามลักษณะของงาน ซึ่งเป็น Best Practice ในการบริหารจัดการต้นทุนและประสิทธิภาพ

“Different models carry very different cost profiles... Instead of sending everything to a vision model (expensive...), we routed text inputs to a lighter, cost-effective text model. This simple design choice reduced costs significantly without compromising accuracy.” - Talentica Software

Decision Matrix

-  **'Classification'**: ประเภทของงานคืออะไร?
-  **'Cost Model'**: ต้นทุนที่ยอมรับได้
-  **'Latency Requirements'**: ความเร็วในการตอบสนองที่ต้องการ
-  **'Accuracy Requirements'**: ระดับความแม่นยำที่ต้องการ



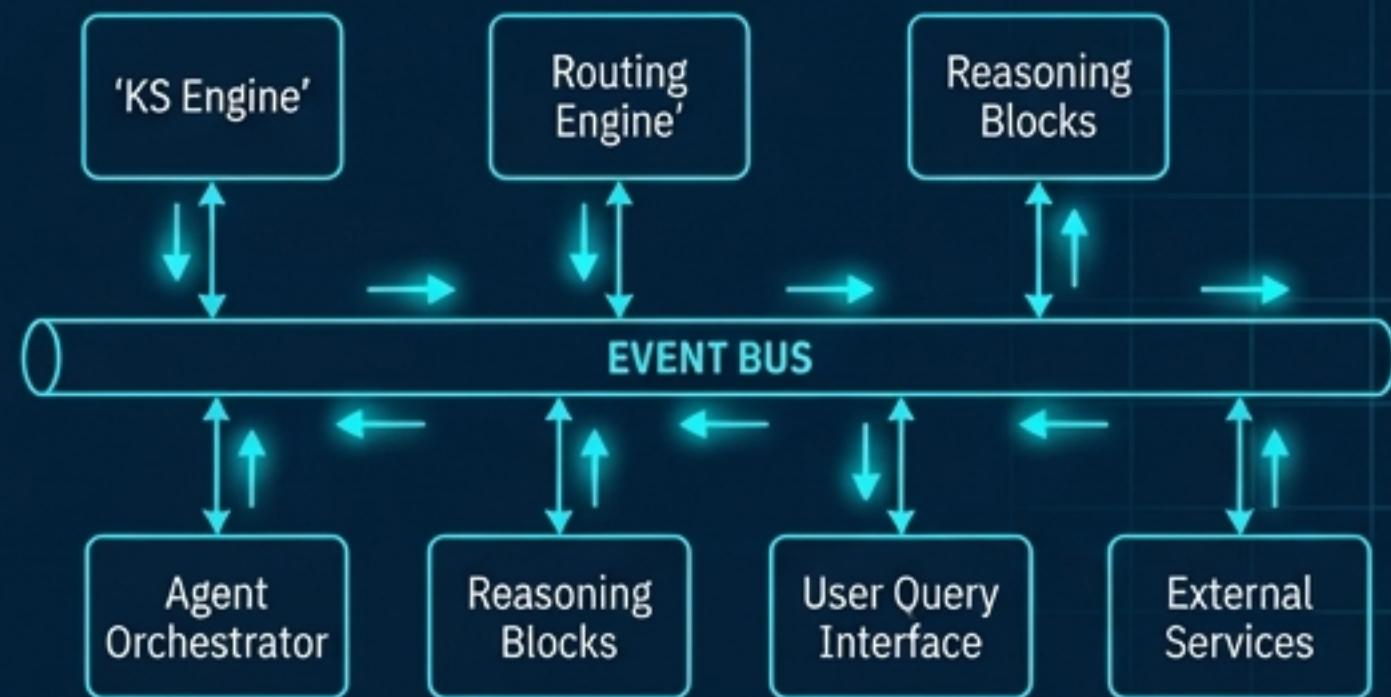
Task Type	Selected Model	Rationale
Reasoning-Heavy	GPT-5.xx (Placeholder)	Best for complex logic & synthesis
RAG-Heavy	Claude Series	Excels at finding info in large contexts
Summarization	Gemini Series	Efficient and accurate for summarization
Tool-Call	Local Model	Low latency, cost-effective for simple tasks

SUPPORTING INFRASTRUCTURE: The Backbone of the System

เพื่อให้สถาปัตยกรรมหลักทำงานได้อย่างราบรื่นและขยายตัวได้ เรา มีระบบสนับสนุนที่แข็งแกร่ง 2 ส่วน

EVENT BUS SYSTEM - The Central Nervous System

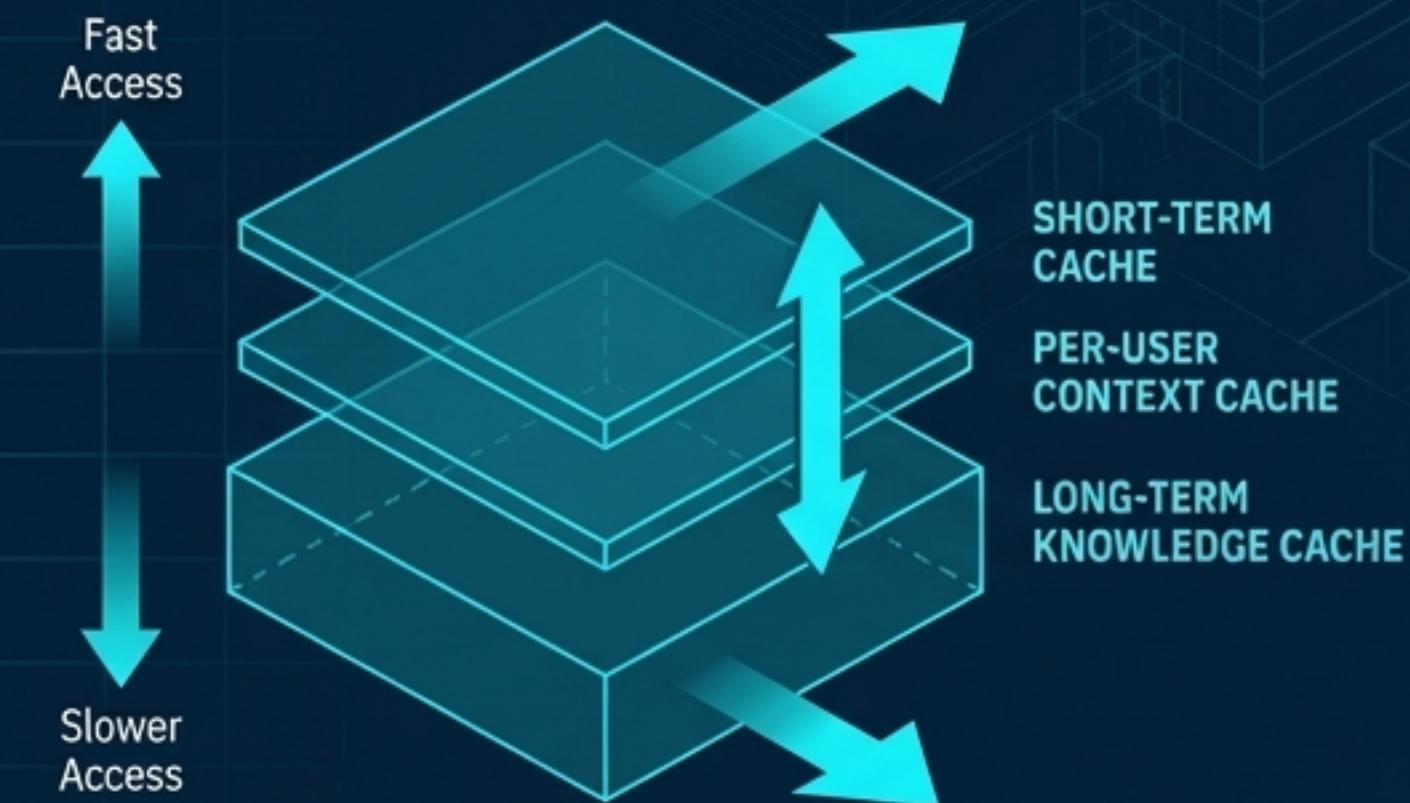
ระบบสื่อสารภายในแบบ Publish/Subscribe



- 📢 **Event Publish/Subscribe:** แจ้งเดือนเหตุการณ์ระหว่างส่วนประกอบต่างๆ
- ⚙️ **Background Job:** จัดการงานที่ทำงานอยู่เบื้องหลัง
- 📈 **Metric Push:** ส่งข้อมูลการวัดผลไปยังระบบ Monitoring
- 📝 **Log & Trace:** หัวใจของการทำให้ระบบ 'Traceable' สามารถติดตามการทำงานทุกขั้นตอนได้

CACHE STRATEGY - The Scalability Engine

กลยุทธ์การทำ Cache เพื่อเพิ่มความเร็วและรองรับการขยายตัว

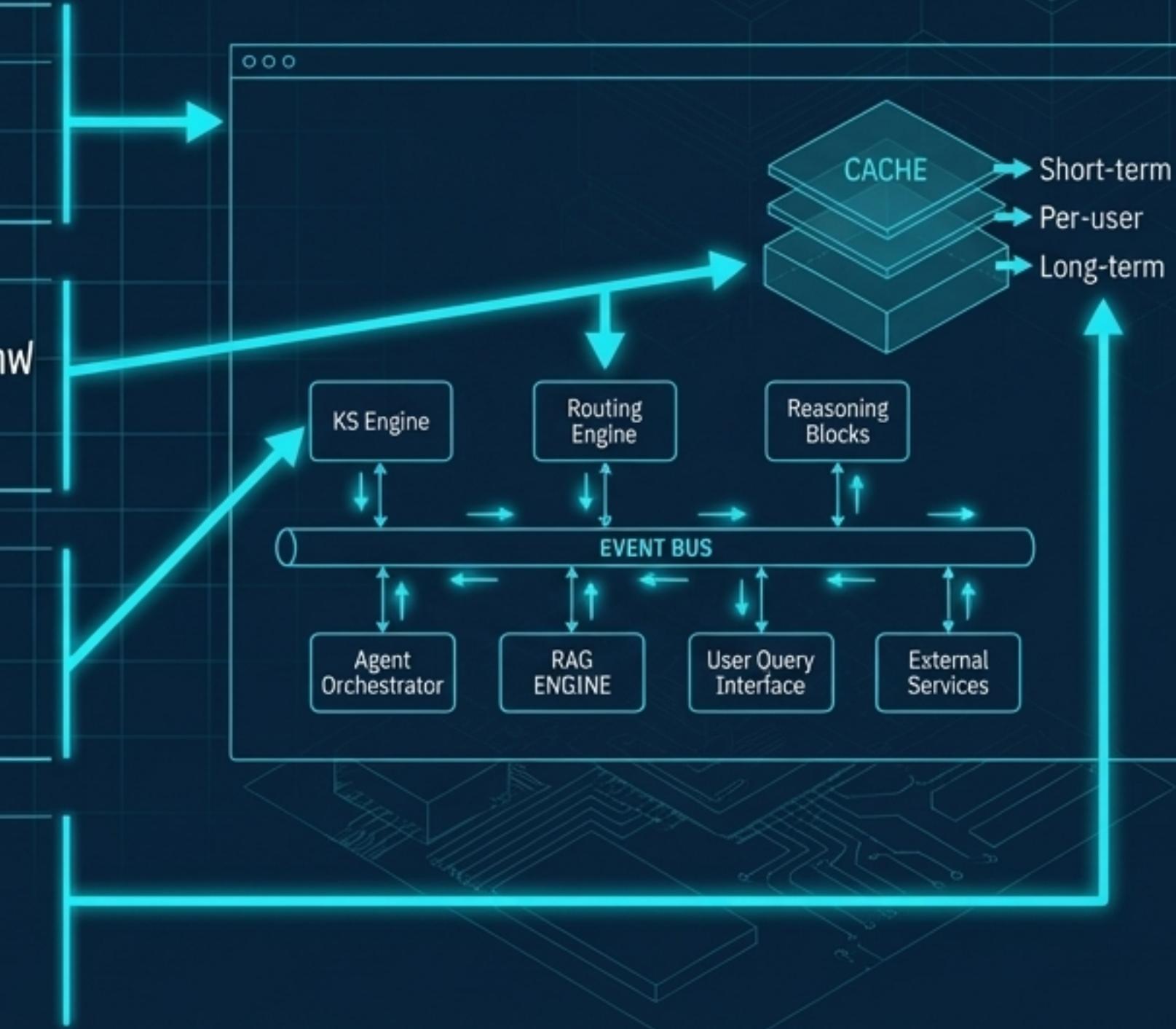


- ⌚ **Short-term Cache:** สำหรับข้อมูลที่เรียกใช้บ่อยในระยะสั้น
- 👤 **Per-user Context Cache:** เก็บ Context ของผู้ใช้แต่ละคน
- 🕒 **Long-term Knowledge Cache:** เก็บผลลัพธ์ขององค์ความรู้ที่สังเคราะห์แล้ว
- 🔧 **Rule-based Invalidation:** กฎเกณฑ์ในการลบ Cache ที่ล้าสมัย

FROM BLUEPRINT TO REALITY: The 25 Foundational Files

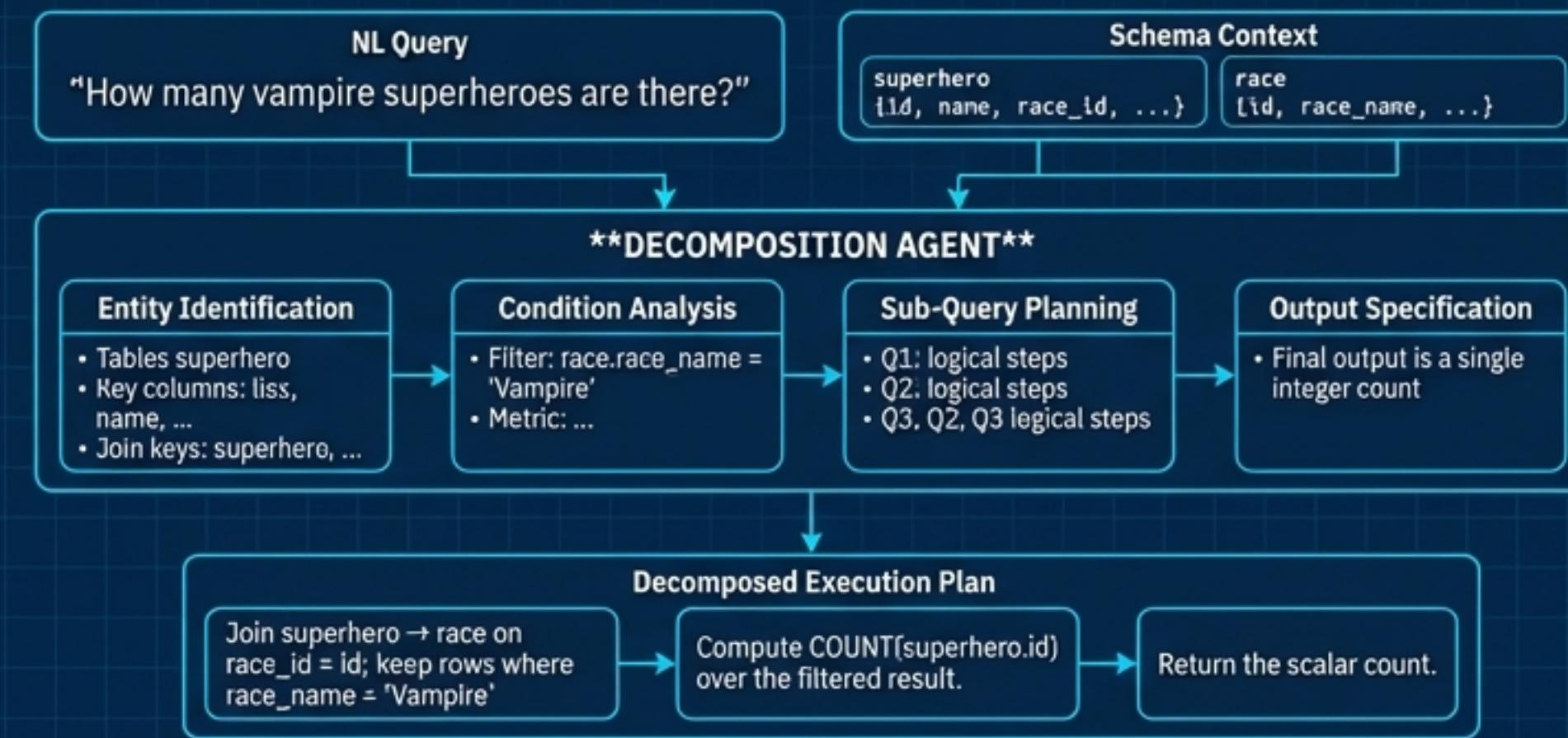
ทุกแนวคิดใน Master Blueprint นี้ ถูกจับคู่แบบ 1:1 กับไฟล์ในโปรเจกต์ นี่คือแผนที่ที่เชื่อมระหว่างสถาปัตยกรรมและโค้ด

File Name	Description
Group 1: Foundational Contracts & Structure 01 SYSTEM CONTRACT 02 ARCHITECTURE 03 PROJECT STRUCTURE	หลักการและกฎเหล็ก ภาพใหญ่ของระบบ โครงสร้างไฟล์เดอร์
Group 2: Data & Knowledge Layer 04 DATA SCHEMA 05 UNIFIED GRAPH 07 CONSTRAINT 09 KS ENGINE	โครงสร้างตารางข้อมูล โครงสร้าง Node/Edge ของกราฟ กฎความสัมพันธ์ของข้อมูล ระบบจัดการและยิงค์ความรู้
Group 3: Core Engines 10 RAG ENGINE 11 AGENT ENGINE 12 FLOW ENGINE 13 ROUTING	ระบบ Retrieve ข้อมูล ระบบ Orchestrate Agents ระบบจัดลำดับการทำงาน ระบบเลือกโมเดล
Group 4: Infrastructure & API 14 EVENT BUS 15 CACHE 18 API SPEC 23 DEPLOY	ระบบสื่อสารภายใน ระบบเร่งความเร็ว บัญชี Endpoints ขั้นตอนการ Deploy



USE CASE IN ACTION: The Hybrid NL2SQL Agent

หนึ่งใน Agent ที่ทรงพลังที่สุดที่เราสามารถสร้างบนสถาปัตยกรรมนี้คือ Agent แปลงภาษาคนเป็น SQL (NL2SQL) ซึ่งเป็นตัวอย่างที่ดีของการทำงานร่วมกันของ Agent เฉพาะทางหลายตัว



1. Query Decomposer Agent



Task: รับคำถาม ("How many vampire superheroes are there?") และแยกองค์ประกอบออกเป็นแผนการทำงาน (Execution Plan)
Process: Entity Identification → Condition Analysis → Sub-Query Planning → Output Specification

2. SQL Generator Agent



Task: แปลง Execution Plan ให้กลายเป็น SQL Query
Mechanism: ใช้ SLM (Small Language Model) ที่ Fine-tune มาเพื่องานนี้โดยเฉพาะ และมี Fallback ไปหา LLM (เช่น GPT-4o) หากเกิดข้อผิดพลาด

3. Executor & Validator Agent



Task: นำ SQL ที่สร้างขึ้นไปรันกับฐานข้อมูลจริง และตรวจสอบความถูกต้อง (Syntax & Semantics)
Feedback Loop: หากเกิด Error จะส่งข้อมูลกลับไปให้ Generator Agent เพื่อแก้ไข (Self-correction)

Key takeaway: สถาปัตยกรรมของเราไม่ได้เป็นแค่ 'กล่องดำ' แต่เป็นระบบ Agent ที่ทำงานร่วมกัน มีการตรวจสอบและแก้ไขตัวเองได้

THE OUTCOME: A System That is Deterministic, Scalable, and Reliable

Master Blueprint นี้ไม่ใช่แค่เอกสาร แต่เป็นรากฐานที่นำเราไปสู่ระบบที่มีคุณสมบัติ 3 ประการสำคัญ:



DETERMINISTIC (เป็นระบบที่คาดเดาได้)

- ทุกการทำงานมีกฎเกณฑ์และเส้นทางที่ชัดเจน
- สามารถติดตาม (Trace) และดีบัก (Debug) ได้ง่าย
- ลดความ ‘มัว’ และความไม่แน่นอนของ AI



SCALABLE (ขยายตัวได้)

- สถาปัตยกรรมแบบ Micro-services และ Agent ทำให้ง่ายต่อการเพิ่มความสามารถใหม่ๆ
- Cache Strategy และ Routing Engine ช่วยรองรับภาระงานที่เพิ่มขึ้น



RELIABLE (เชื่อถือได้)

- System Contracts และ Validation loops ช่วยป้องกันข้อผิดพลาด
- การออกแบบที่โปร่งใสทำให้ทุกคนเข้าใจ และเชื่อมั่นในผลลัพธ์

This is not just code; it is disciplined architecture.