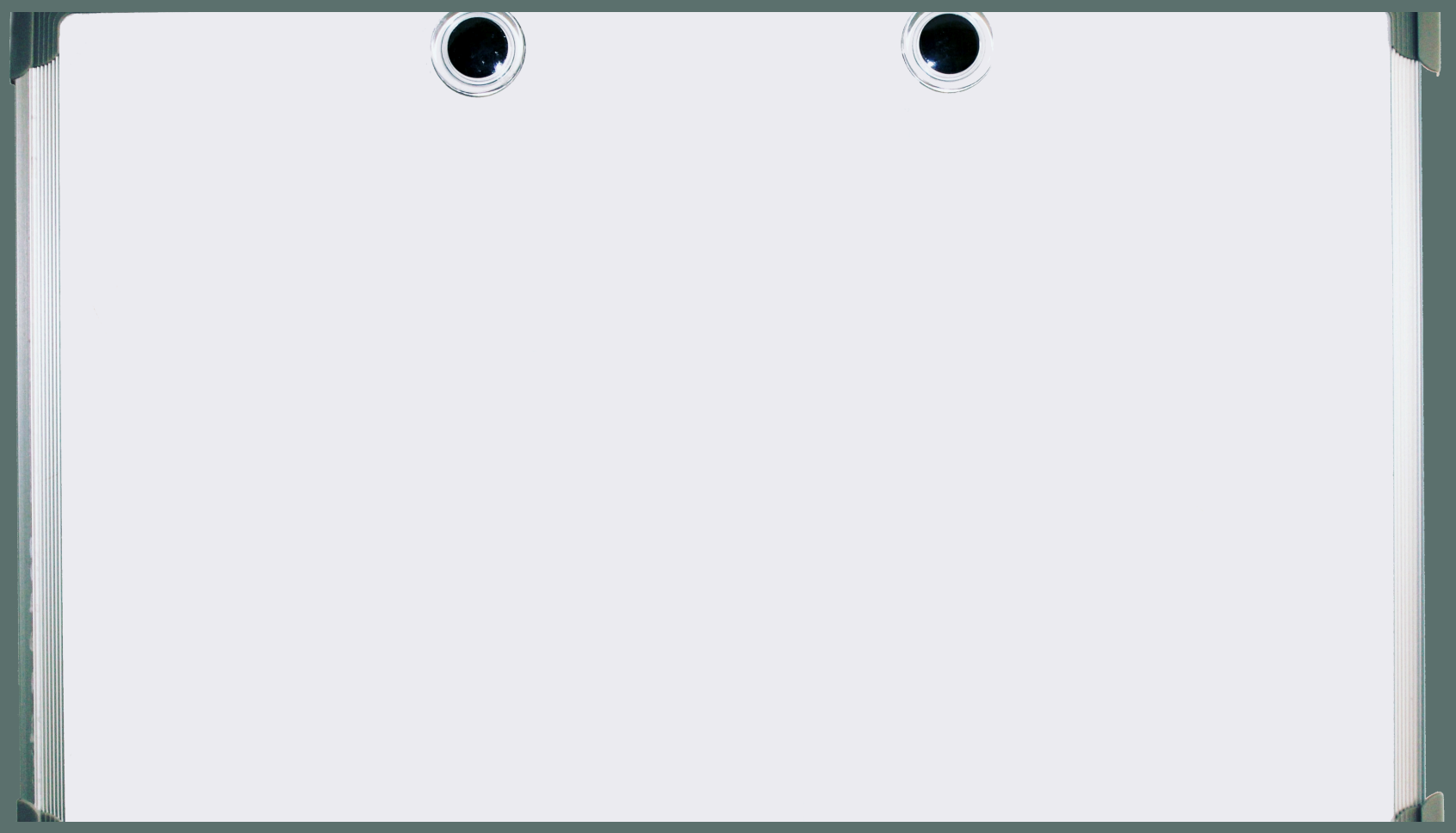
🎨 Systems Design: Production RAG System



Build a production RAG system that doesn't hallucinate

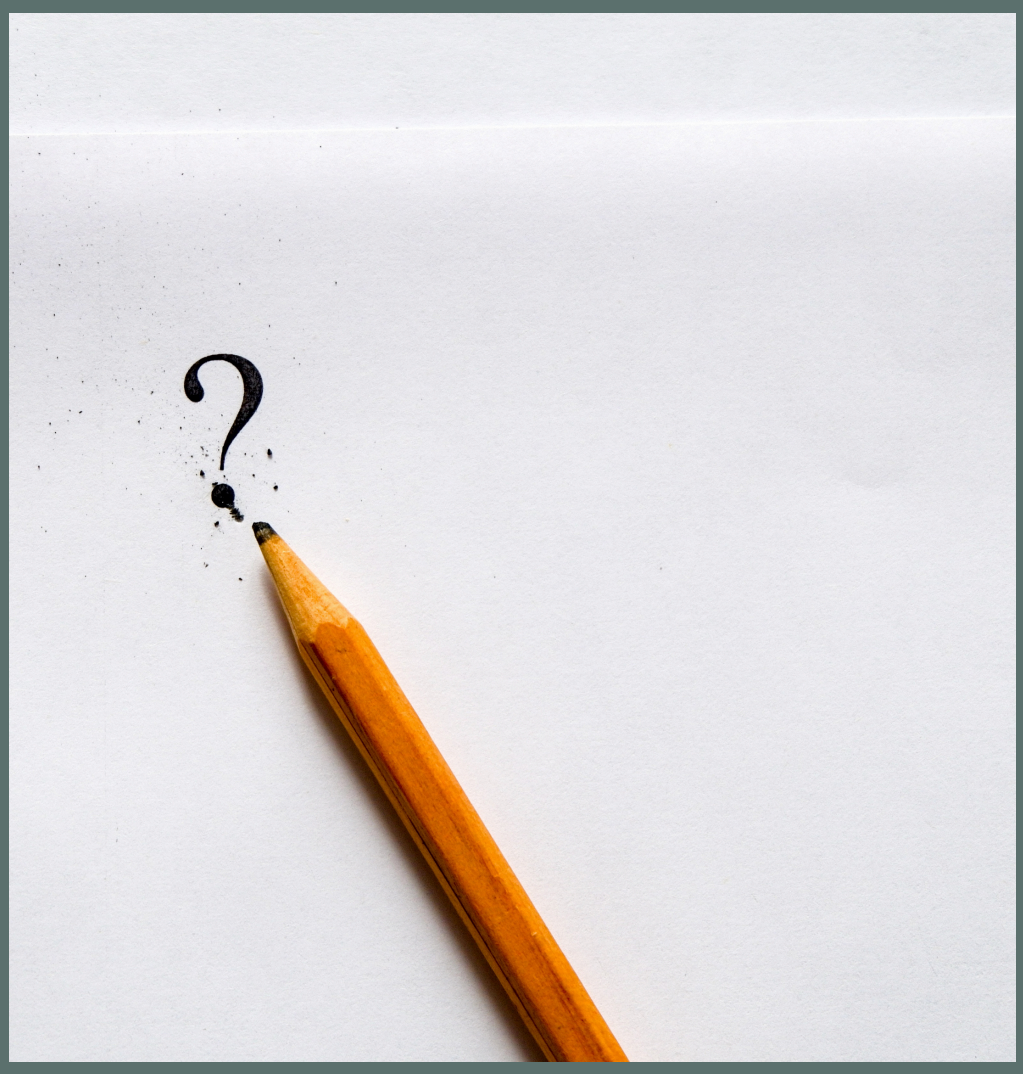
Where do you start?

Most people start with LangChain.

I start with constraints.

Here's the thinking process: 👇

📝 Step 1: What **Problem** Are We Solving?



How do we give an LLM access to private knowledge WITHOUT hallucinating?

This question determines everything:

→ Architecture

→ Tools

→ Failure modes

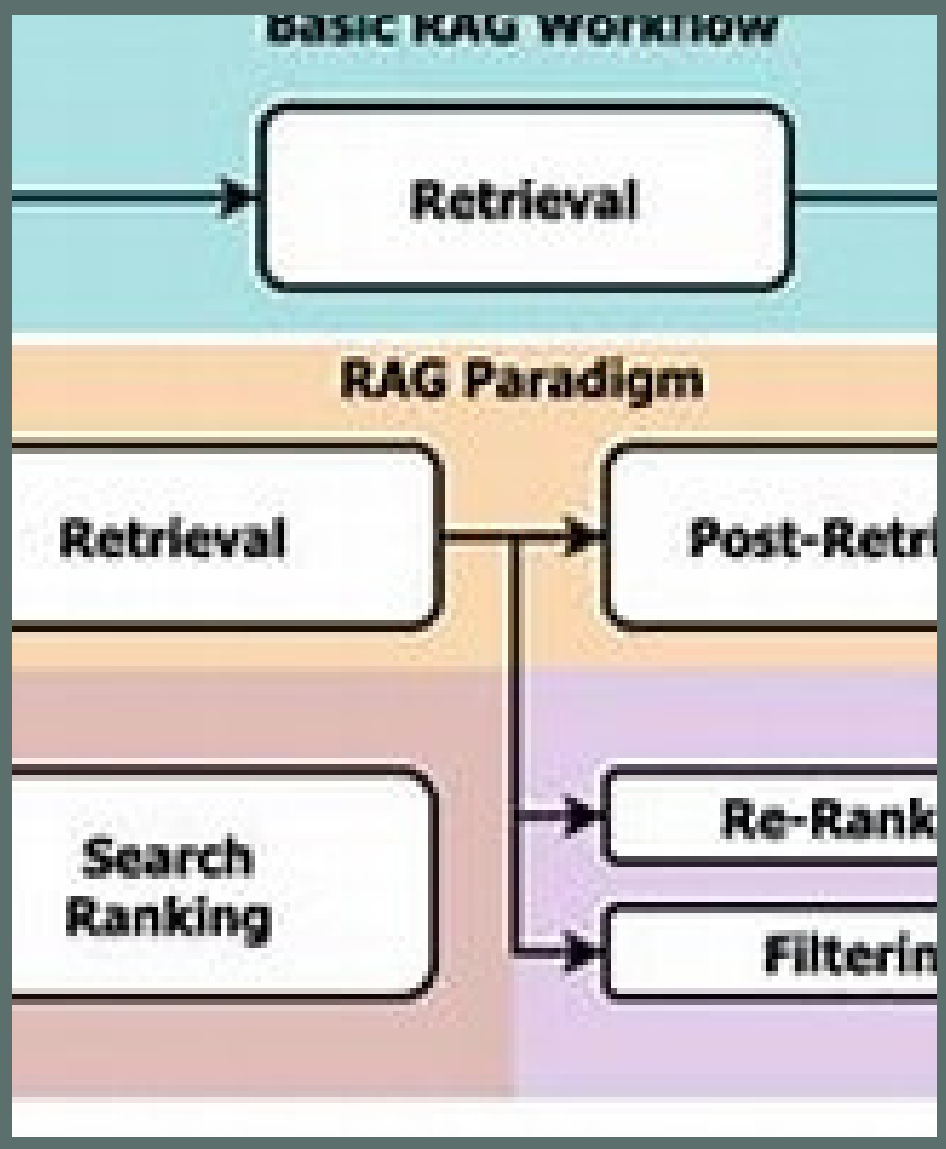
→ Success metrics

Wrong question = wrong system.

Right question = constraints become clear.

🔄 Step 2: Map the **Data Flow**

User Query → **[?]** → Response



Fill in the **[?]:**

→ Where does knowledge come from?

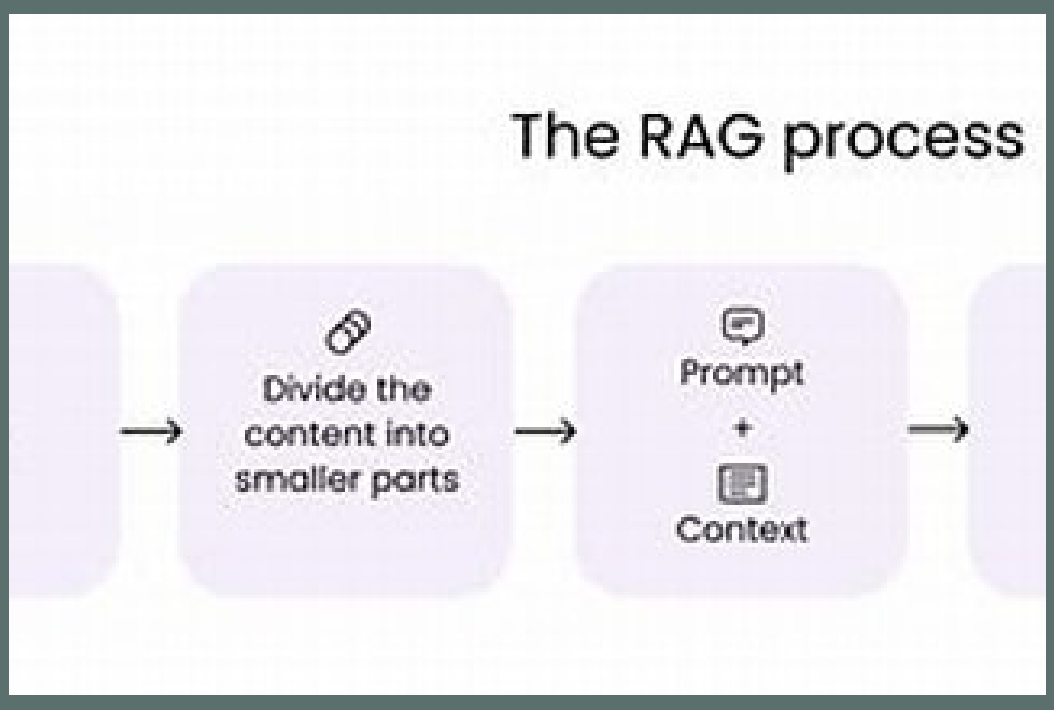
→ How do we find relevant context?

→ How do we prevent hallucinations?

This is the **skeleton**.

Now **add the organs**.

🧩 Step 3: Core Components



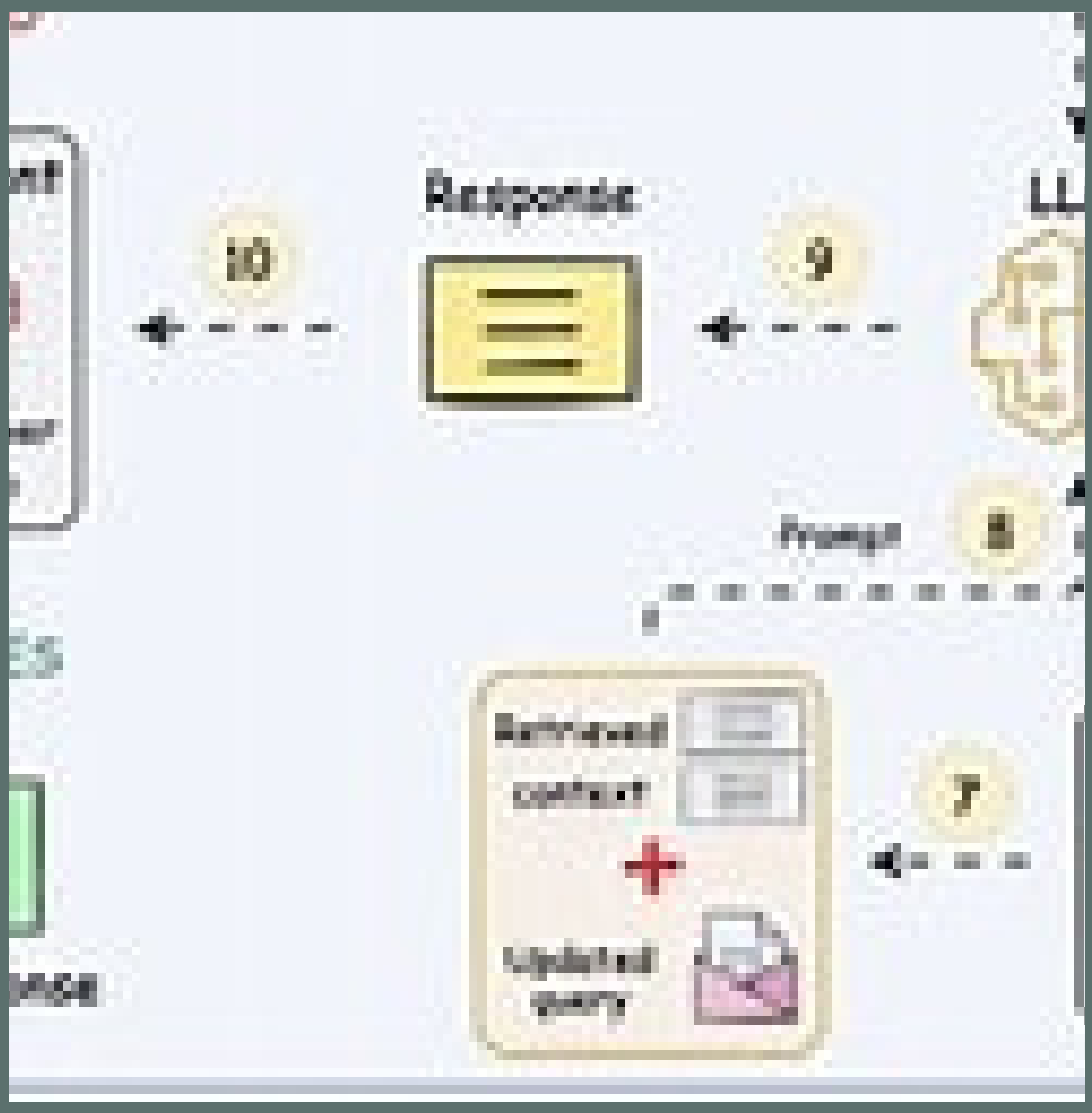
User Query
→ Embedding
→ Vector Search
→ Context
→ LLM
→ Response

But this is **TOO** simple.

What's missing?
→ Where do documents come from?
→ How are they processed?
→ What if context is wrong?
→ How do we know if it works?

Add the layers that production needs

📥 Step 4: **Data Sources** & Processing



Documents (PDFs, wikis, APIs)

↓

Chunking + Metadata Extraction

↓

Embedding Model

↓

Vector Database

Questions to answer:

→ **Chunk size?** (512 tokens? 1024?)

→ **Overlap?** (50 tokens? 100?)

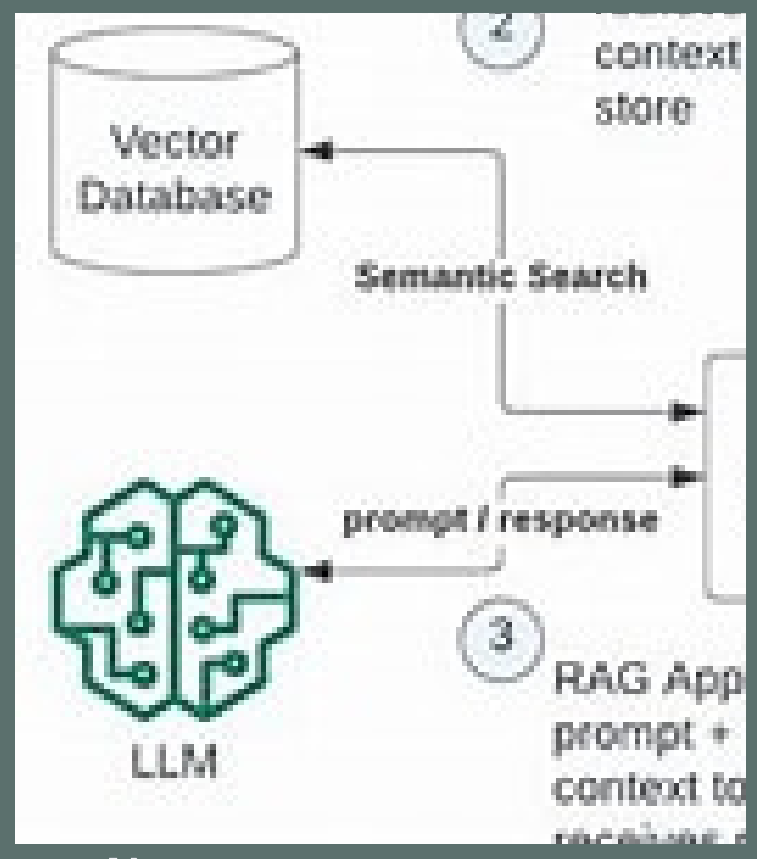
→ **Metadata?** (source, date, author?)

→ **Update frequency?** (real-time? batch?)

This layer determines retrieval quality.

🧠 Step 5: Vector Storage & Search

Vector Database (Pinecone/Weaviate/Chroma)
→ Stores embeddings
→ Semantic search
→ Metadata filtering
→ Hybrid search (vector + keyword)



This isn't just storage. This is **WHERE RETRIEVAL LIVES.**

Bad retrieval = LLM hallucinates (even if model is perfect)

Most **RAG failures happen here**.

🎯 Step 6: Context Building & Enrichment

Retrieved Chunks

↓

Re-ranking (best → worst)

↓

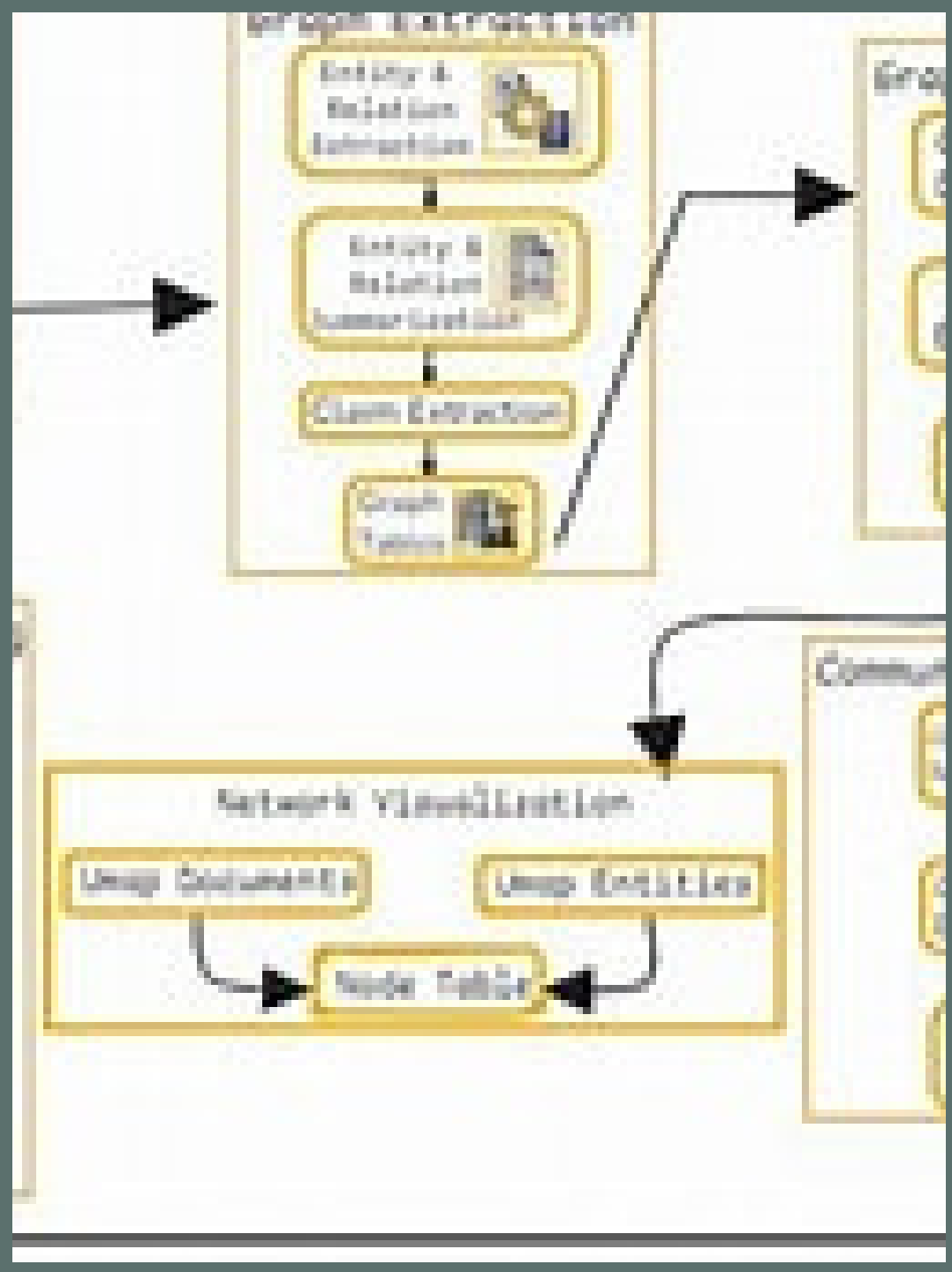
Summarization (if too long)

↓

Metadata enrichment

↓

Formatted Context → LLM

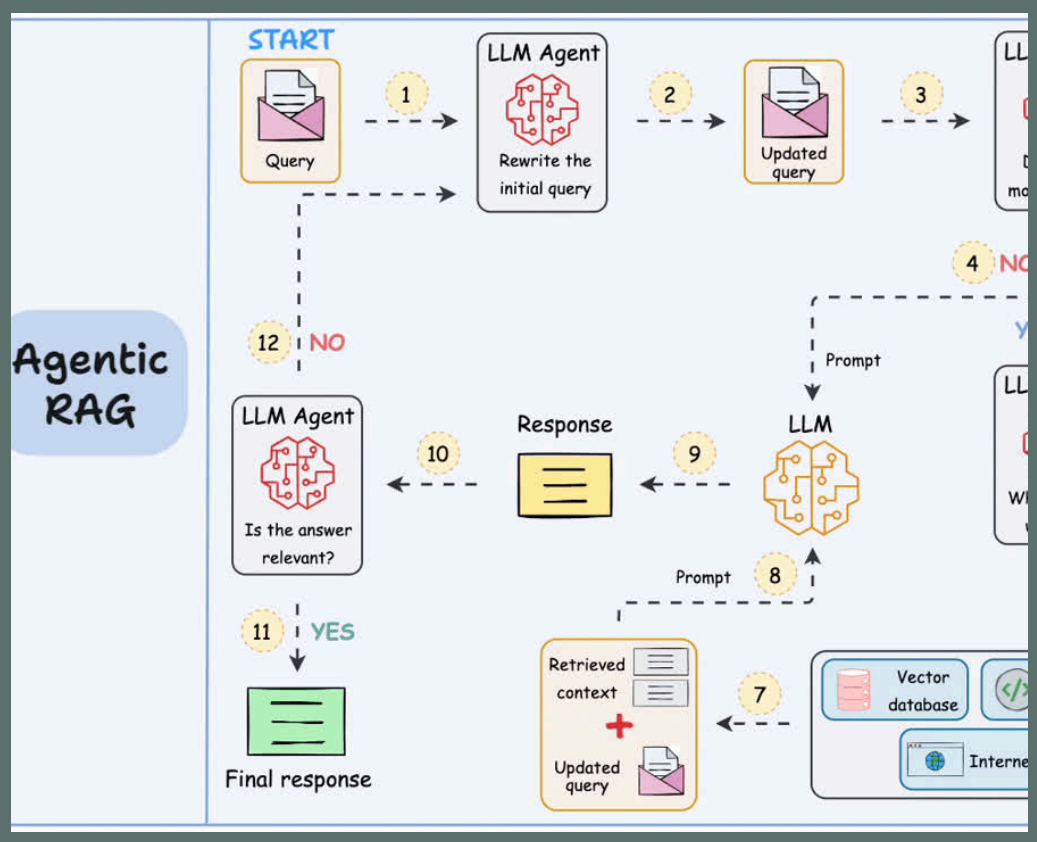


Models only see what you feed them.

Context quality = Output quality.

This is where **LlamaIndex**/**LangChain** add value.

📊 Step 7: Monitoring & Guardrails



Track:

→ Retrieval accuracy (did we get right docs?)

→ Hallucination rate (is LLM making things up?)

→ Latency (user experience)

Tools: Arize, Langfuse, Helicone

→ Context relevance (are chunks useful?)

Production RAG without monitoring = **gambling.**

You can't improve what you don't measure.

⚠️ Step 8: Design for Failure

Mark the failure points BEFORE you build.

Where does this system break ?

→ Stale embeddings (documents updated, vectors not)
→ Poor chunking (context split across chunks)
→ Context overflow (too many tokens for LLM)
→ Retrieval drift (relevance degrades over time)

Elite systems plan for failure.

Then design defenses:

→ Embedding refresh jobs
→ Smart chunking with overlap
→ Context window management
→ Continuous monitoring

✅ The Complete Production RAG System

Data Sources

↓

Processing (Chunking + Embedding)

↓

Vector Database (Search + Storage)

↓

Context Building (Re-rank + Enrich)

↓

LLM (Generate Response)

↓

Monitoring (Track Everything)

This is production RAG.
Not "add LangChain and hope."

7 layers.

Each critical.

Most teams:
→ Start with the LLM
→ Add retrieval as afterthought
→ Skip monitoring
→ Wonder why it hallucinates

1% teams:
→ Design data flow first
→ Optimize retrieval layer
→ Monitor everything
→ RAG works reliably

The difference: Systems thinking.

Next in series: Multi-Cloud MLOps Pipeline
👇

#SystemsDesign #RAG #MLOps #Production