das-purnendu

The 2026 GenAI Stack

The new AI engineer map.

Not just prompts.
Not just RAG.
Not just agents.

A production GenAI system now needs:

Models • Context • Tools • Evals • Traces • Guardrails • Humans • Operations

MODELS	Choose the right model for the right task
CONTEXT	Give the model the right information
TOOLS	Let the model act in the real world
EVALS	Measure quality, safety, and performance
TRACES	See what happened, step by step
GUARDRAILS	Control risk and prevent bad outcomes
HUMANS	Add judgment and approvals where needed
OPERATIONS	Run, monitor, and improve in production

Save this. You'll need it.

das-purnendu

The shift

2023:
Prompt the model

Focus:
Single answer quality

2024:
Add RAG

Focus:
Find better information

2025:
Build agents

Focus:
Automate workflows

2026:
Operate agentic systems

Focus:
Reliability, safety & scale

The skill has moved from:

"Can I get a good answer?"

to:

"Can this system complete real tasks reliably, safely, and repeatedly?"

Save this. You'll need it.

das-purnendu

The 2026 GenAI Stack

INPUTS

- User Request
- Data & Documents
- Databases & APIs
- Tools & Systems
- Teams & Experts

1. Model routing
2. Context engineering
3. RAG 2.0
4. Tool use
5. MCP
6. A2A
7. Multi-agent orchestration
8. Evals
9. Tracing
10. Guardrails
11. Human-in-the-loop
12. AgentOps

OUTCOMES

- Accurate Answers
- Safe Execution
- Reliable Workflows
- Lower Cost & Latency
- Better Outcomes

This is the new AI engineer map.

Save this slide — you'll need it.

das-purnendu

Why prompting is not enough anymore

A prompt can improve one answer.
But production systems need to handle:

- messy users
- missing context
- bad retrieval
- tool failures
- security risks
- long workflows
- cost limits
- latency limits
- hallucinations
- human approvals

A PROMPT

User question
+
Instructions
+
Context

PRODUCTION GENAI SYSTEM

- Handles messy users
- Manages missing context
- Recovers from bad retrieval
- Survives tool failures
- Stays safe and secure
- Runs long workflows
- Respects cost limits
- Meets latency limits
- Reduces hallucinations
- Supports human approvals

Prompting is still useful.
But it is no longer the whole stack.

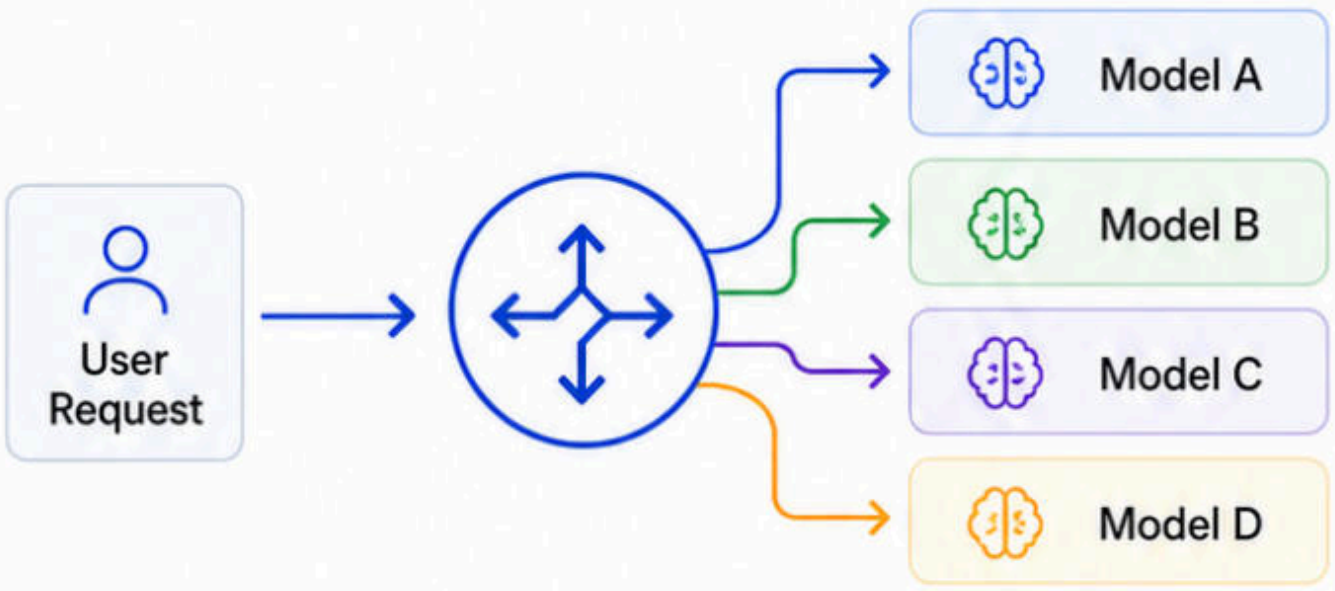
Save this slide for quick revision!

das-purnendu

Layer 1: Model Routing

One model for everything is lazy architecture.

In 2026, smart GenAI systems route tasks to different models based on:



- cost
- latency
- reasoning depth
- context length
- modality
- tool-use ability
- safety requirements
- output format
- business risk

The question is not:

Which model is best?

VS

The question is:

Which model is best for this task?

Save this. You'll need it.

das-purnendu

Model Routing: Practical Map

Use different models for different jobs:

JOB	MODEL TYPE
Simple classification	small, fast model
Complex reasoning	stronger reasoning model
Code generation	coding-specialized model
Summarization	low-latency model
Vision tasks	multimodal model
Sensitive decisions	safer, controlled model
Final answer	high-quality response model

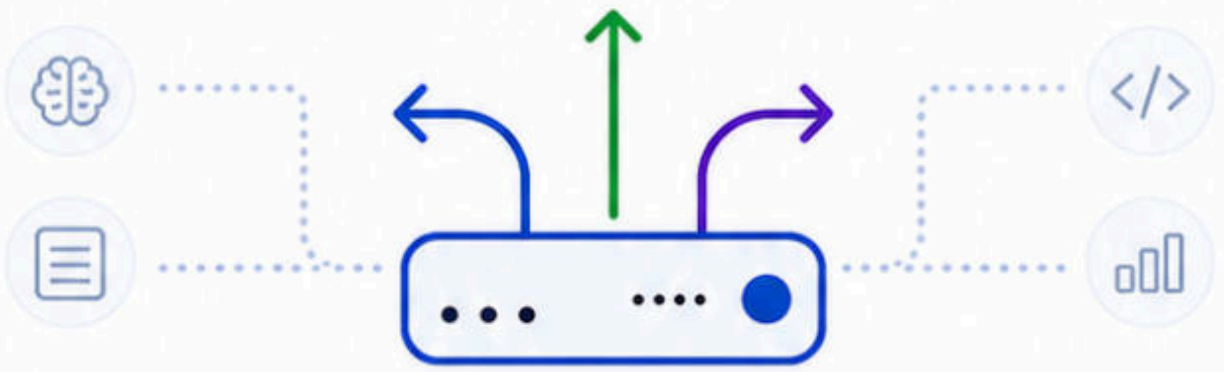
The best GenAI apps are not model-loyal.

They are outcome-loyal.

Save this slide for quick revision!

das-purnendu

Model Routing: What to track



Before routing models, define:

- task type
- input complexity
- required accuracy
- allowed latency
- max cost
- safety risk
- tool requirement
- fallback model
- escalation rule

Bad routing gives you:

- high cost
- slow output
- unpredictable quality

Good routing gives you:

- speed
- quality
- control

Save this slide for quick revision!

das-purnendu

Layer 2: Context Engineering

Prompt engineering asks:

"What should I say to the model?"

Context engineering asks:

"What should the model know before it answers?"

This includes:

- user intent
- task history
- tool outputs
- retrieved docs
- business rules
- memory
- previous actions
- examples
- constraints

MODEL

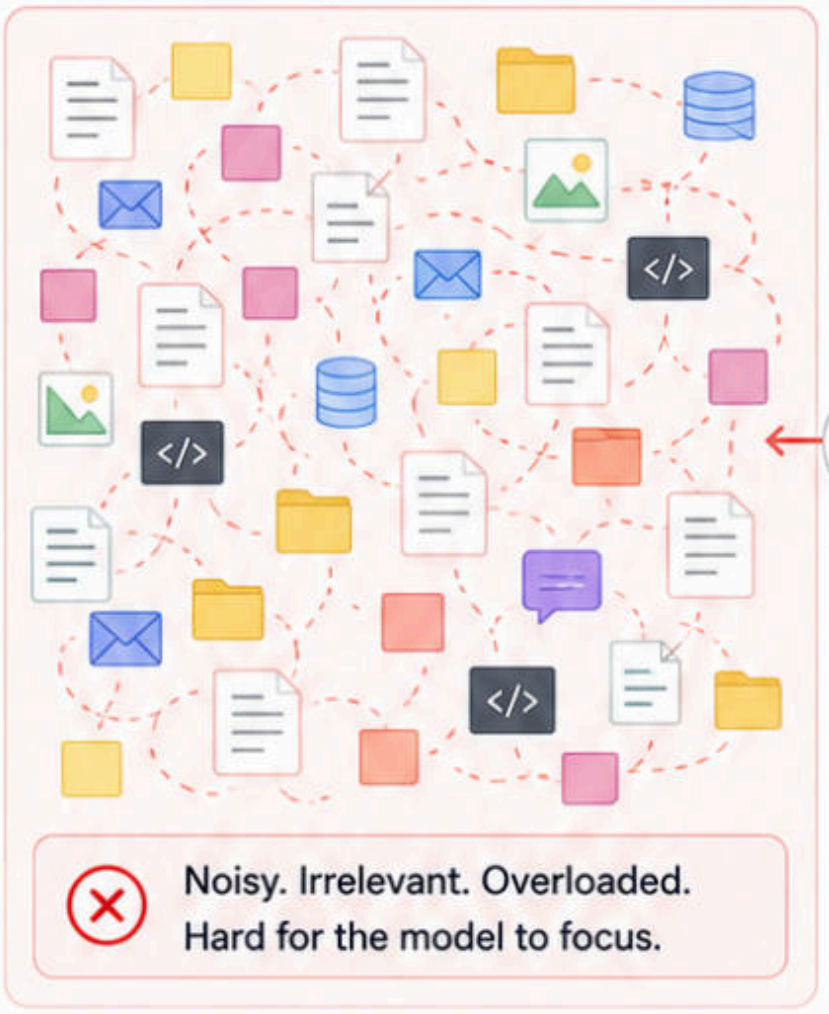
In 2026, context is the real prompt.

das-purnendu

Context Engineering: The mistake

Most teams dump more context.
Better teams select better context.

MORE CONTEXT (THE MISTAKE)



Noisy. Irrelevant. Overloaded.
Hard for the model to focus.

VS

BETTER CONTEXT (THE PRACTICE)

Focused. Relevant. Actionable.
Easy for the model to use.

GREAT SYSTEMS KNOW:

what to include	what to exclude	what to summarize	what to retrieve
what to remember	what to forget	what to pass to tools	what to keep private

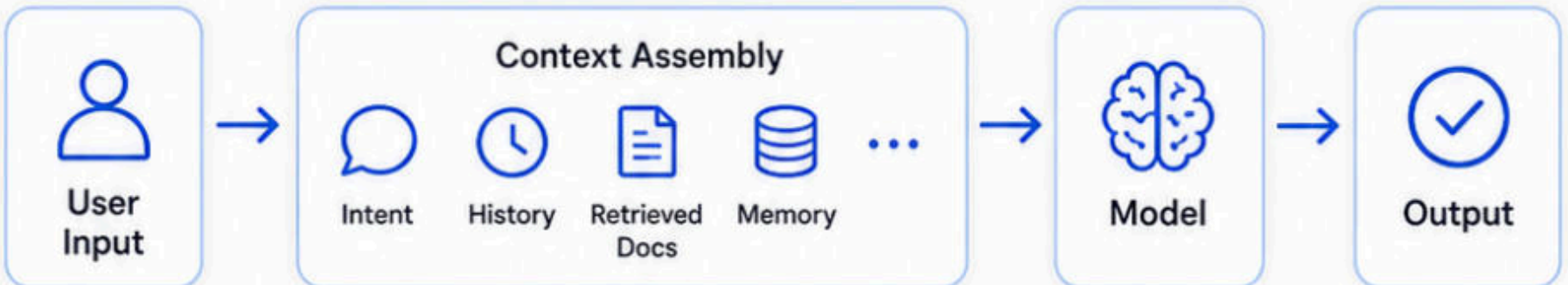
More tokens ≠ better intelligence.

Better context = better output.

Save this slide for quick revision!

das-purnendu

Context Engineering Checklist



Before sending context to the model, ask:

- [ ] Is this relevant?
- [ ] Is this too much?
- [ ] Is this fresh?
- [ ] Is this structured?
- [ ] Is this trusted?
- [ ] Is this user-specific?
- [ ] Is this allowed?
- [ ] Is this task-specific?
- [ ] Is this complete?
- [ ] Is this safe to expose?

Context quality is system quality.

das-purnendu

Layer 3:

RAG 2.0

Old RAG:

retrieve → generate

Basic RAG answers from documents.

2026 RAG:

route → rewrite → retrieve → rerank → filter → ground → cite → evaluate

route → rewrite → retrieve → rerank → filter → ground → cite → evaluate

Advanced RAG proves:

- where the answer came from
- why the source is relevant
- whether the source is fresh
- whether the model stayed grounded
- whether the final answer is faithful

RAG is not dead.

Lazy RAG is dead.

das-purnendu

RAG 2.0:
What actually matters

Your **RAG quality** depends on:

RETRIEVAL FOUNDATION

1. chunking strategy
2. metadata quality
3. embeddings
4. hybrid search

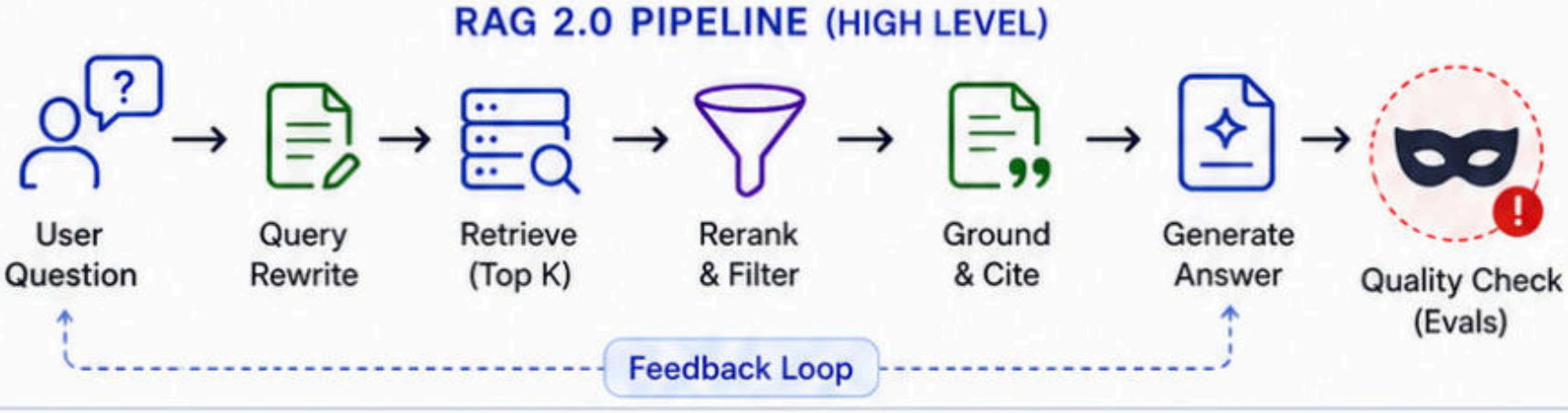
RANKING & REFINEMENT

5. query rewriting
6. reranking
7. permission filters
8. freshness filters

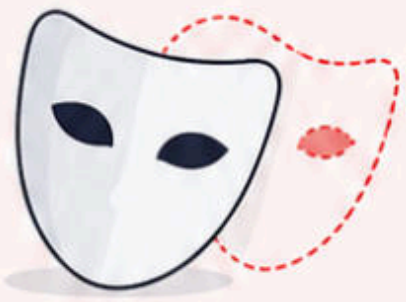
GROUNDING & QUALITY

9. source citation
10. answer faithfulness
11. eval datasets
12. failure analysis

RAG 2.0 PIPELINE (HIGH LEVEL)



Most RAG failures are retrieval failures wearing a generation mask.



Save this slide for quick revision!

13/32

das-purnendu

RAG 2.0: Production questions

Before shipping RAG, ask:

- ☐ Can it handle vague queries?
- ☐ Can it handle conflicting sources?
- ☐ Can it avoid stale documents?
- ☐ Can it respect user permissions?
- ☐ Can it say "I don't know"?
- ☐ Can it cite correctly?
- ☐ Can it detect missing context?
- ☐ Can it recover after bad retrieval?
- ☐ Can it be evaluated automatically?

From a document chatbot...

Docs + chatbot is not enough

→ ✕

Evaluated knowledge system is the goal

A chatbot with documents is *not* automatically a knowledge system.

Save this slide for quick revision!

das-purnendu

Layer 4: Tool Use

LLMs talk.
Tools let them act.

With tools, a GenAI system can:

LLM AS RESPONDER

What's the revenue for last quarter?

The revenue for last quarter was $2.45M.

Provides information based on training data.

LLM AS OPERATOR

Use tools to complete real tasks

- search
- calculate
- query databases
- call APIs
- create tickets
- send emails
- run code
- update CRMs
- trigger workflows
- inspect files
- fetch real-time data

Tool use turns a model from a responder into an operator.

Save this slide for quick revision!

das-purnendu

Tool Use: The hidden risk

Every tool adds **power.**

Every tool also adds **risk.**

Bad tool design causes:

- wrong API calls
- unsafe actions
- data leakage
- permission bypass
- cost spikes
- infinite loops
- duplicated work
- irreversible changes
- confusing failures

The model is not the only thing that needs design.

The tools need design too.

Save this slide for quick revision!

das-purnendu

Tool Use Checklist

For every tool, define:

Identity

- [ ] 1 clear name
- [ ] 2 clear description

Contract

- [ ] 3 input schema
- [ ] 4 output schema
- [ ] 5 permission level
- [ ] 6 failure behavior

Safety

- [ ] 7 timeout limit
- [ ] 8 retry policy

Operations

- [ ] 9 audit log
- [ ] 10 approval requirement
- [ ] 11 cost impact
- [ ] 12 rollback path

A vague tool is a dangerous tool.

Save this slide for quick revision!

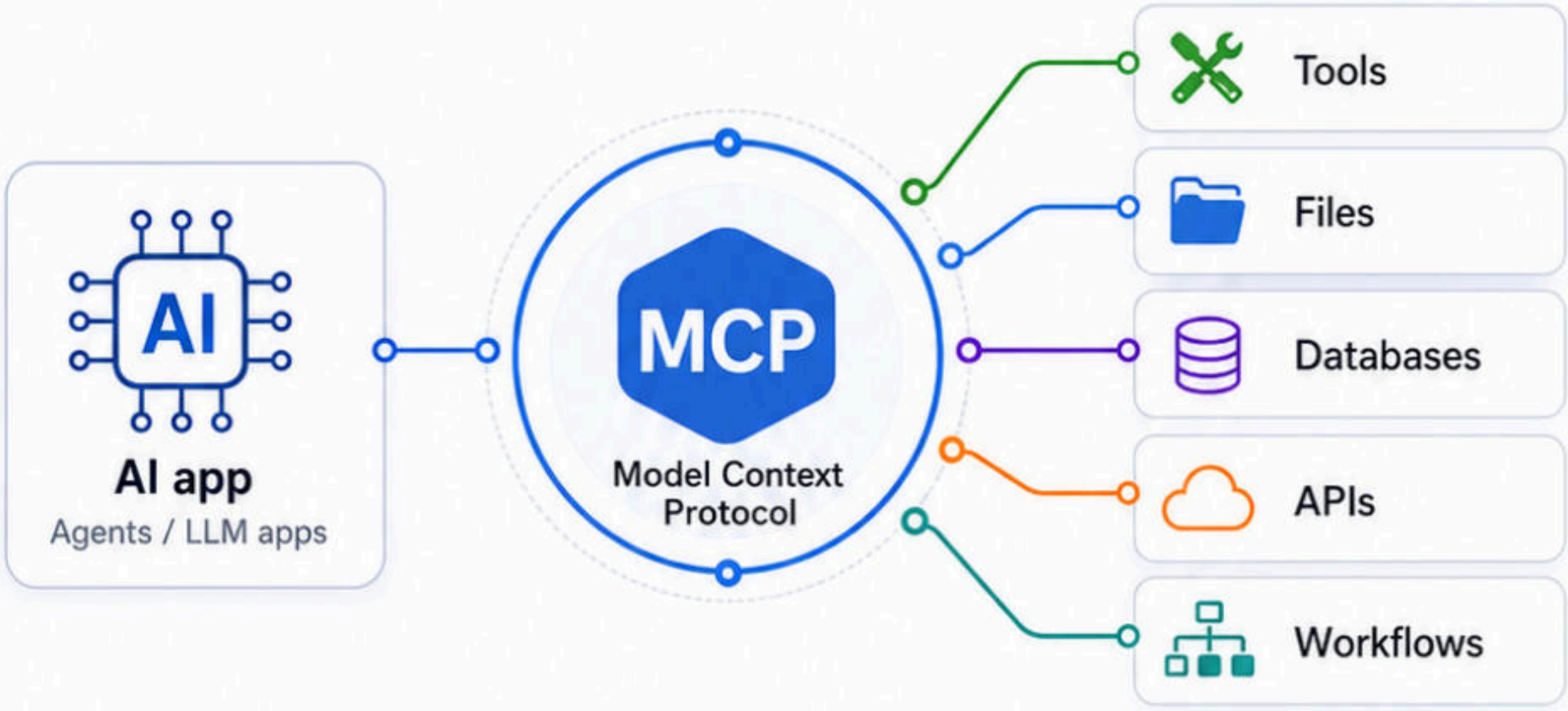
das-purnendu

Layer 5: MCP

Model Context Protocol.

MCP is a standard way to connect
AI applications with external systems.

Think:
AI app → MCP → tools, files, databases, APIs, workflows



Why it matters:

- less custom integration
- reusable tool & connectors
- cleaner context access
- better agent workflows
- easier enterprise adoption
- more standardized tool use

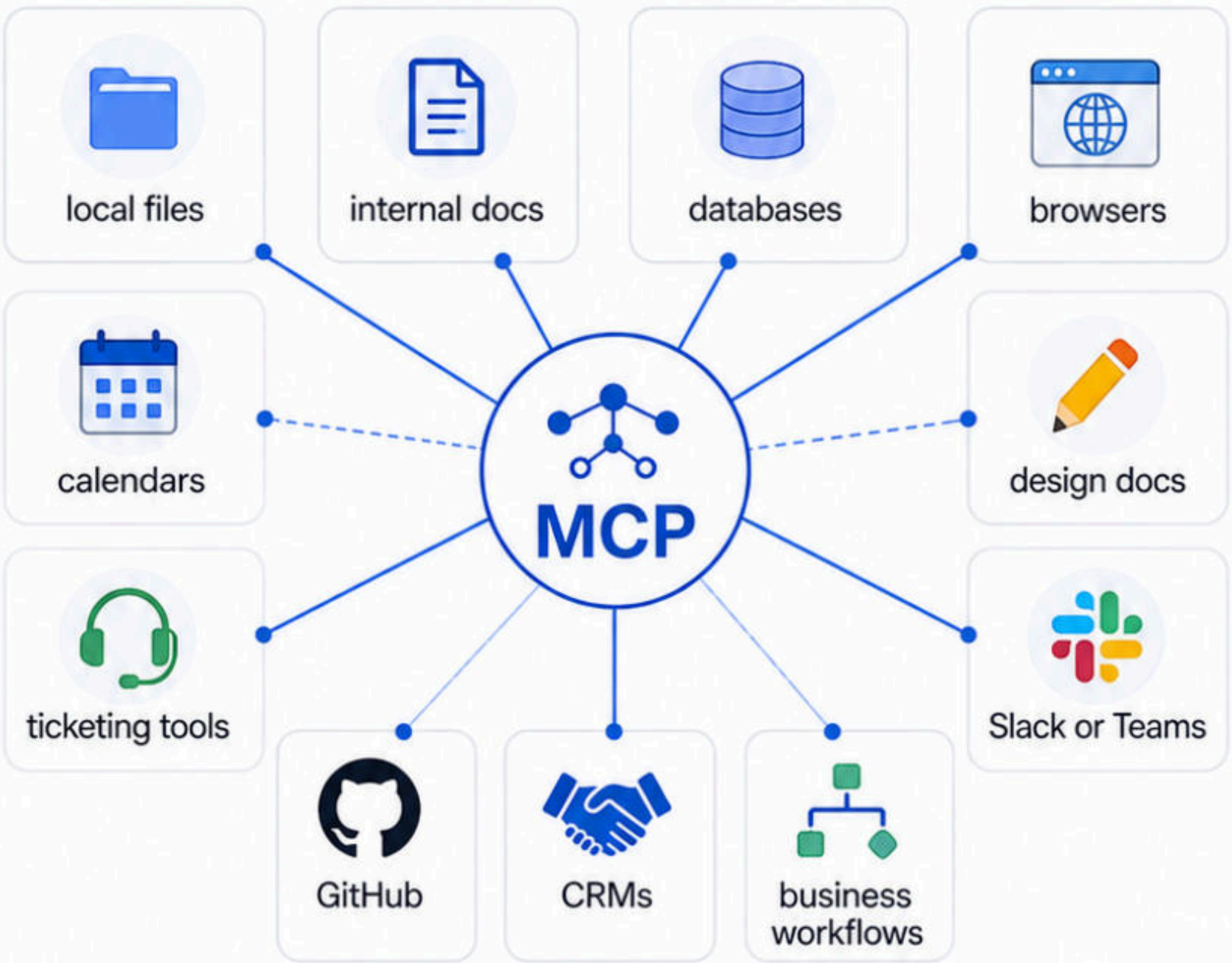
MCP is becoming one of the key integration layers for agentic systems.

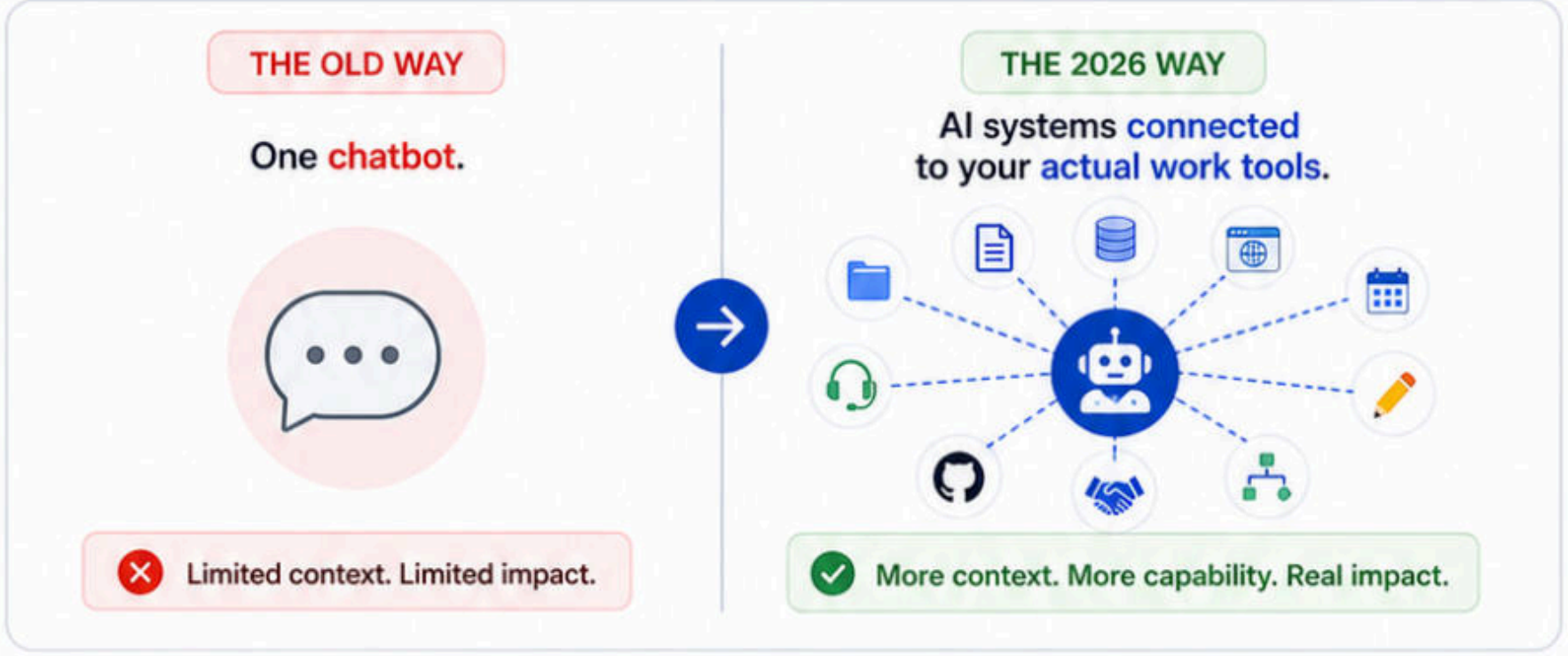
18/32

das-purnendu

MCP: What GenAI engineers should know

MCP is useful when agents need access to:





The future is not one chatbot.

The future is AI systems connected to your actual work tools.

Save this slide for quick revision!

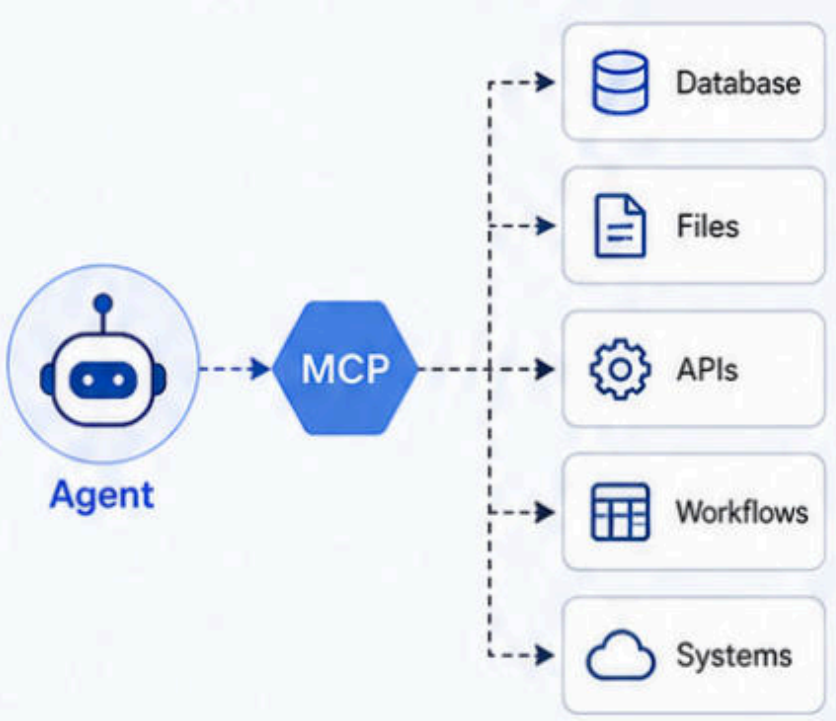
das-purnendu

Layer 6: A2A

Agent2Agent.

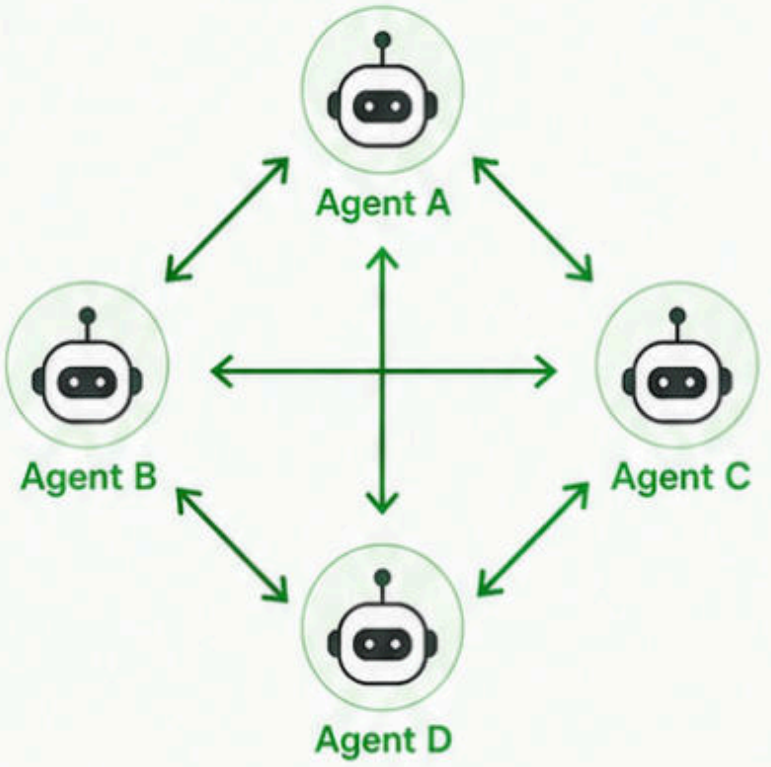
MCP connects:

agents → tools/data



A2A connects:

agents → other agents



This matters when different agents need to:

- communicate
- share task state
- exchange information
- delegate work
- coordinate actions
- collaborate across platforms
- complete enterprise workflows

A2A is about agent *interoperability*.

Save this slide for quick revision!

das-purnendu

MCP vs A2A

Don’t confuse them.

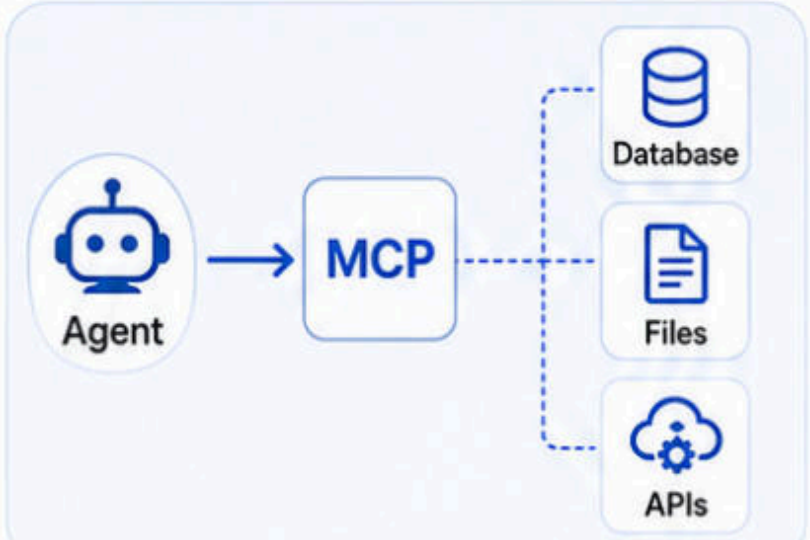
MCP

MCP connects:

agent to tools/data

Example:

agent reads database



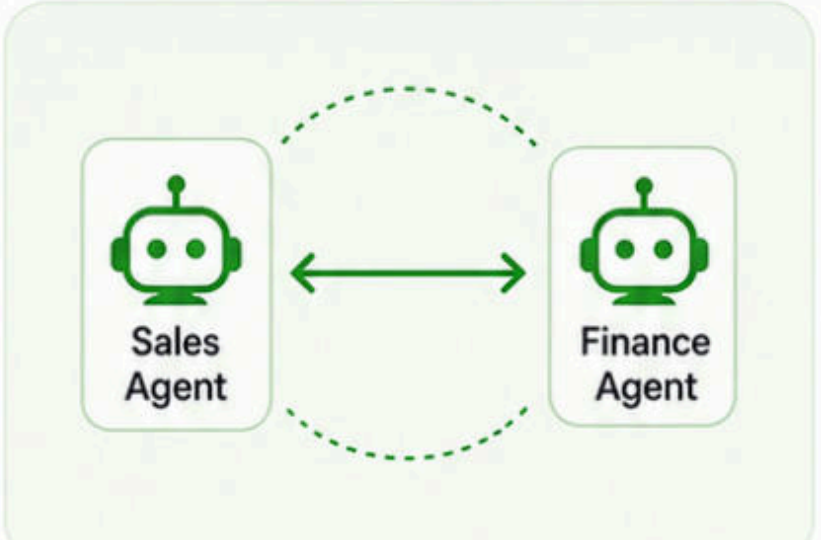
A2A

A2A connects:

agent to agent

Example:

sales agent talks to finance agent



Simple way to remember:

MCP = tool access

A2A = agent collaboration

Both matter because
agentic systems are becoming more distributed.

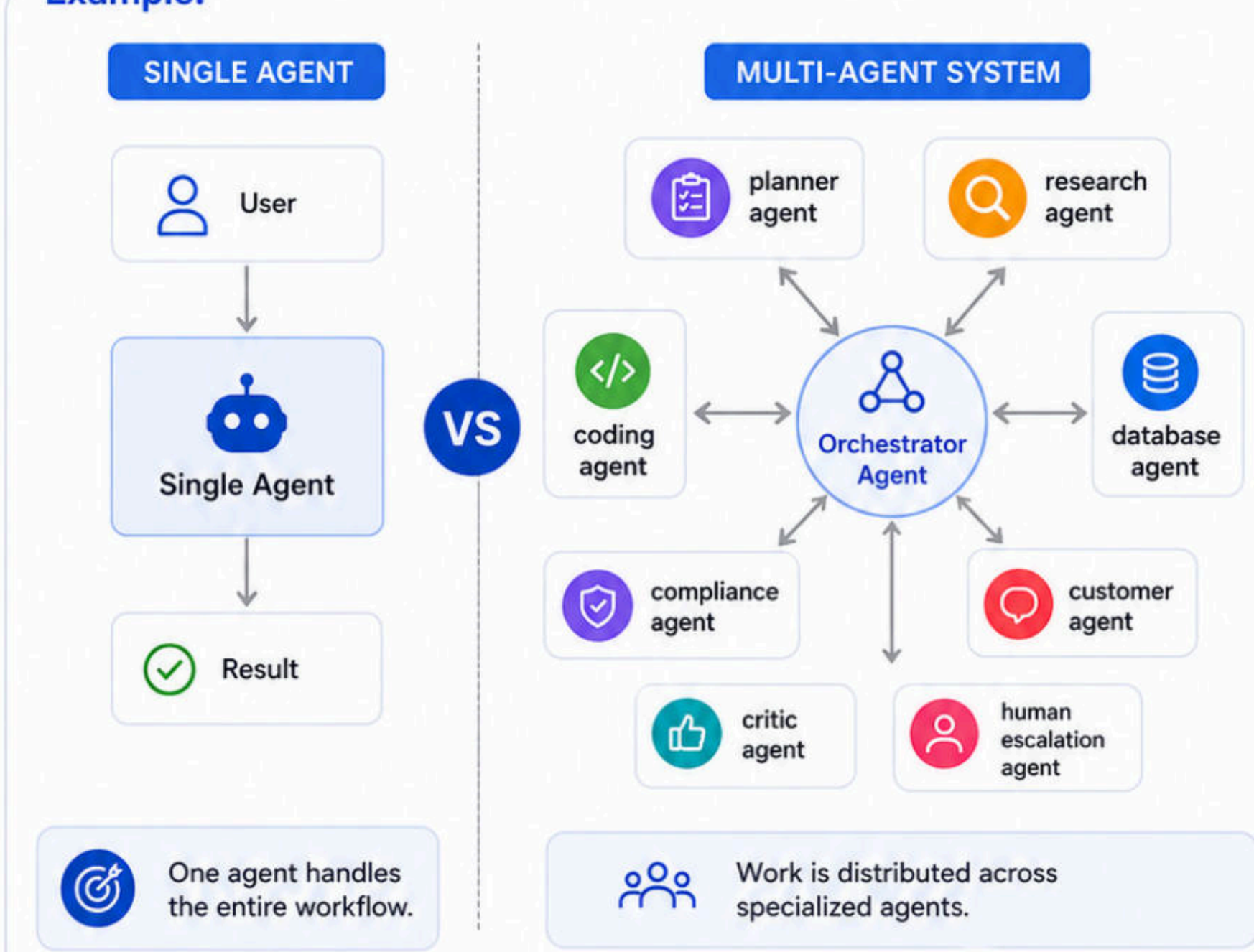
das-purnendu

Layer 7:
Multi-Agent Orchestration

A single agent can do **one workflow.**

A multi-agent system can **divide work across specialists.**

Example:

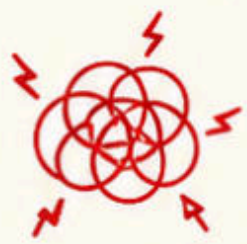


One agent handles the entire workflow.

Work is distributed across specialized agents.

But more agents do not automatically mean better intelligence.

Sometimes they **only create more chaos.**



Save this slide for quick revision!

das-purnendu

Multi-Agent Failure Modes

Multi-agent systems fail when:

01 agents duplicate work

02 agents disagree silently

03 one agent trusts bad context

04 handoffs lose information

05 no one owns the final answer

06 tool calls conflict

07 cost grows unpredictably

08 traces become unreadable

09 loops never stop

10 humans cannot intervene

The harder part is not creating agents.
The harder part is coordinating them.

Save this slide for quick revision!

das-purnendu

Multi-Agent Design Checklist

Before adding another agent, ask:

1. Does this need a separate role?
2. What is this agent responsible for?
3. What can it access?
4. What can it not do?
5. Who reviews its output?
6. When does it hand off?
7. When does it stop?
8. What happens if it fails?
9. What does it cost?
10. How will we trace it?

Do not add agents because it sounds advanced.

Add agents because the workflow needs **specialization**.

Save this slide for quick revision!

das-purnendu

Layer 8: Evals

Prompting gives you **an output.**

Evals give you **confidence.**

In 2026, GenAI systems need evals for:

1. answer accuracy
2. groundedness
3. retrieval quality
4. tool-call correctness
5. task completion
6. safety
7. latency
8. cost
9. refusal quality
10. regression failures
11. human satisfaction

No evals = no production confidence.

Save this slide for quick revision!

das-purnendu

Evals: The minimum useful suite

For a serious LLM app, build:

1. Golden test cases
2. Edge cases
3. Adversarial prompts
4. Retrieval evals
5. Tool-call evals
6. Hallucination checks
7. Safety checks
8. Regression tests
9. Cost benchmarks
10. Human review samples

Your eval suite is your GenAI test suite.

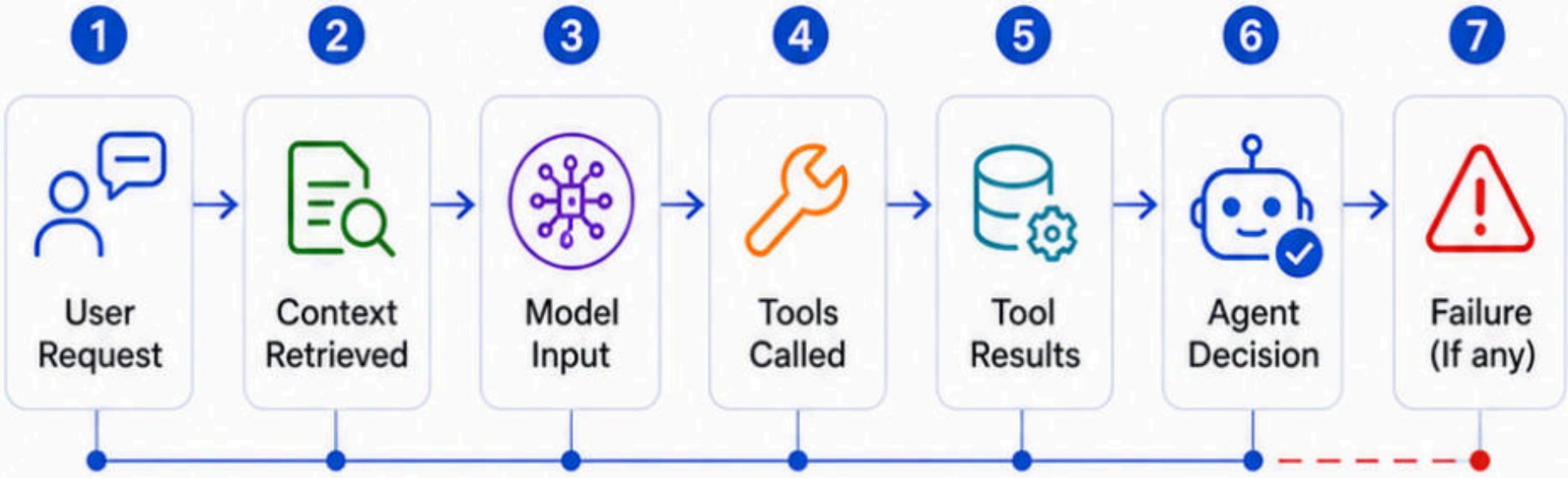
Would you ship code without tests?
Then don't ship prompts without evals.

Save this slide for quick revision!

das-purnendu

Layer 9: Tracing

When an agent fails, you need to see the path.



Tracing answers:

1. what did the user ask?
2. what context was retrieved?
3. what did the model see?
4. what tools were called?
5. what did each tool return?
6. which agent made the decision?
7. where did latency increase?
8. where did cost increase?
9. where did the system go wrong?

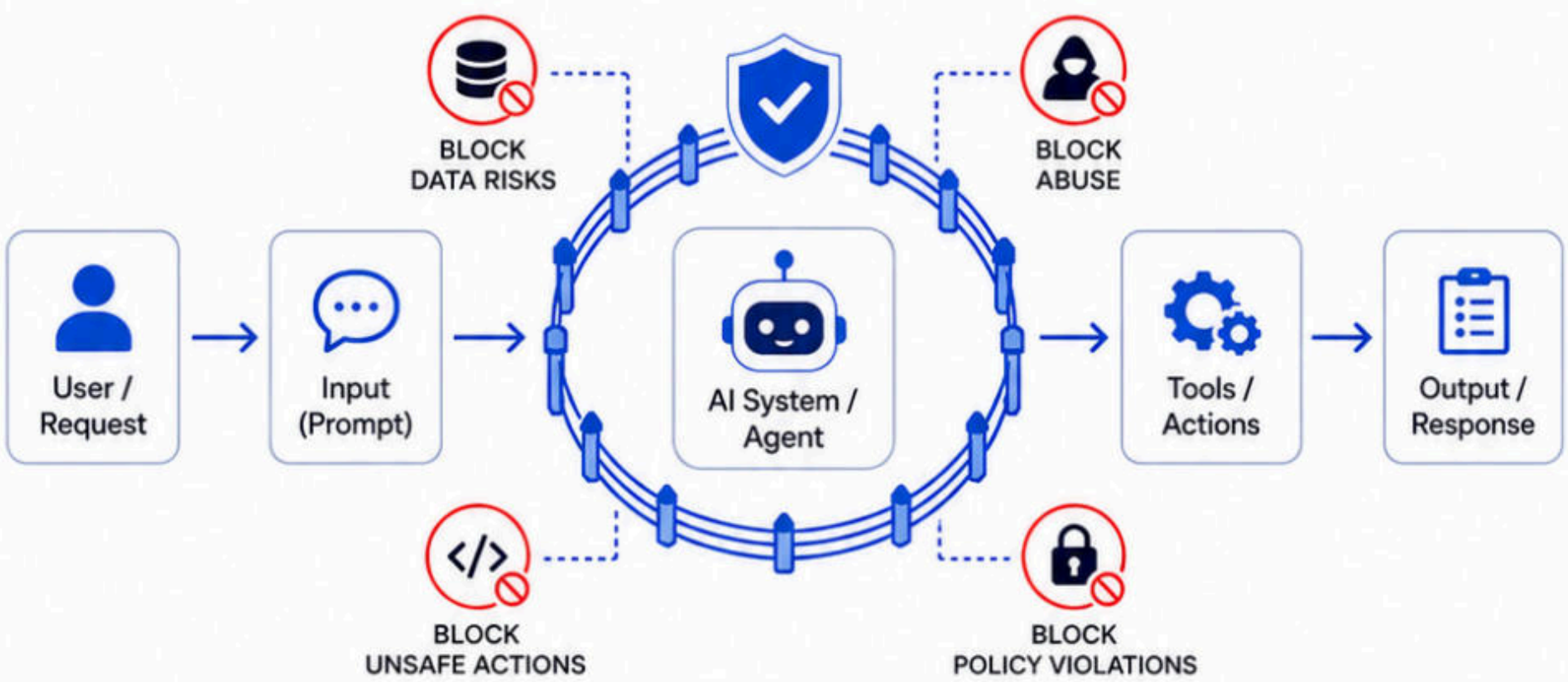
No traces = no debugging.

Save this slide for quick revision!

das-purnendu

Layer 10: Guardrails

Guardrails define what the system should never do.



You need guardrails around:

1. data access
2. tool permissions
3. prompt injection
4. unsafe actions
5. financial actions
6. medical/legal claims
7. Customer communication
8. private data
9. compliance workflows
10. autonomous decisions

The more autonomy you give an agent, the more guardrails you need.

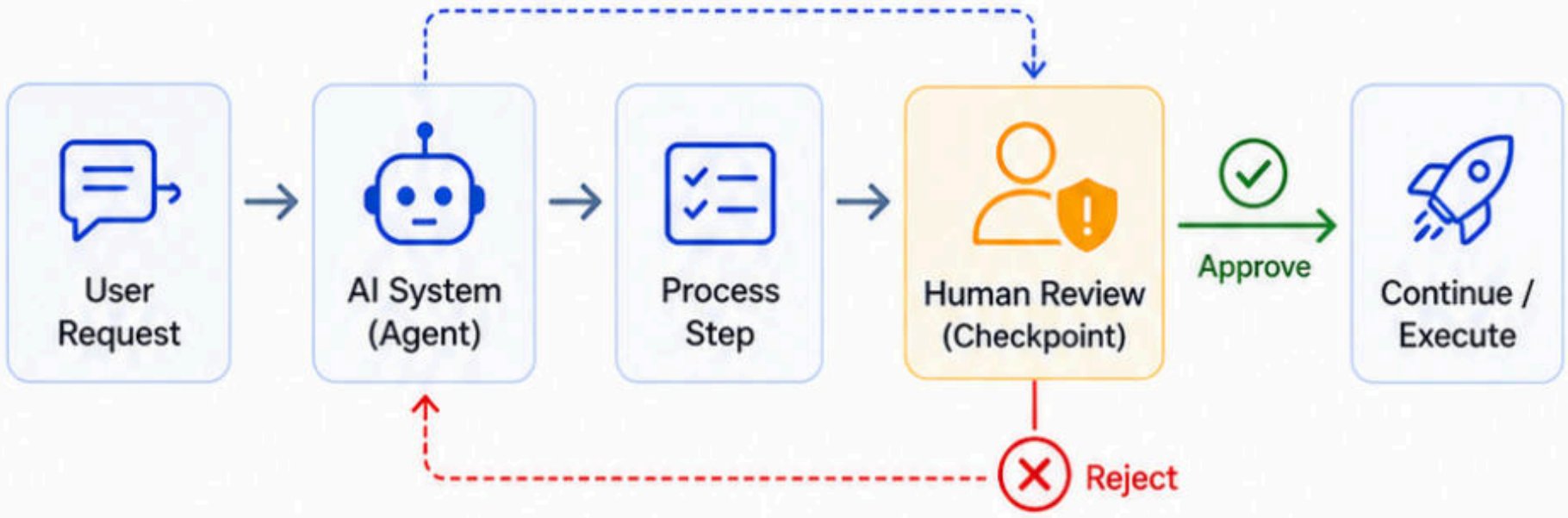
Save this slide for quick revision!

28/32

das-purnendu

Layer 11: Human-in-the-loop

Humans are not a weakness in agentic systems.
They are a control layer.



Use human review for:

1. high-risk decisions
2. uncertain outputs
3. irreversible actions
4. financial approvals
5. legal/compliance steps
6. customer-sensitive messages
7. low-confidence answers
8. policy violations
9. escalation cases

The best systems do not remove humans.
They involve humans at the right moment.

Save this slide for quick revision!

das-purnendu

Layer 12: AgentOps

DevOps	DevOps operates software.
MLOps	MLOps operates models.
LLMOps	LLMOps operates LLM apps.
AgentOps	AgentOps operates autonomous workflows.

AgentOps manages:

- agents
- tools
- permissions
- traces
- evals
- memory
- guardrails
- costs
- incidents
- rollbacks
- human escalations
- production monitoring

AgentOps

AgentOps is where GenAI becomes real production infrastructure.

Save this slide for quick revision!

das-purnendu

The final map

The 2026 GenAI Stack:

1. **Model routing** | chooses the right model.
2. **Context engineering** | gives it the right information.
3. **RAG 2.0** | grounds it in trusted knowledge.
4. **Tool use** | lets it act.
5. **MCP** | connects it to systems.
6. **A2A** | connects it to other agents.
7. **Orchestration** | coordinates workflows.
8. **Evals** | test quality.
9. **Tracing** | explains failures.
10. **Guardrails** | control risk.
11. **HITL** | adds judgment.
12. **AgentOps** | runs it in production.

Save this slide for your next build!

das-purnendu

The 2026 AI Engineer Mindset

OLD MINDSET

Do not just ask:

Can the model answer?

- ✗ Focus on one answer
- ✗ Optimizes for model output
- ✗ Works in demos

NEW MINDSET

Ask:

Can the system complete the task reliably, safely, repeatedly, and affordably?

- ✓ Focus on end-to-end outcomes
- ✓ Optimizes for system reliability
- ✓ Works in production

That is the difference between:

demo AI

- ✗ Looks impressive
- ✗ Brittle and inconsistent
- ✗ Fails in real-world edge cases
- ✗ Hard to trust

and

production AI

- ✓ Solves real problems
- ✓ Reliable and consistent
- ✓ Handles edge cases
- ✓ Trusted and scalable

You don't just build AI.
You build reliable AI systems.

Save this slide for quick revision!

32/32

das-purnendu

Save this

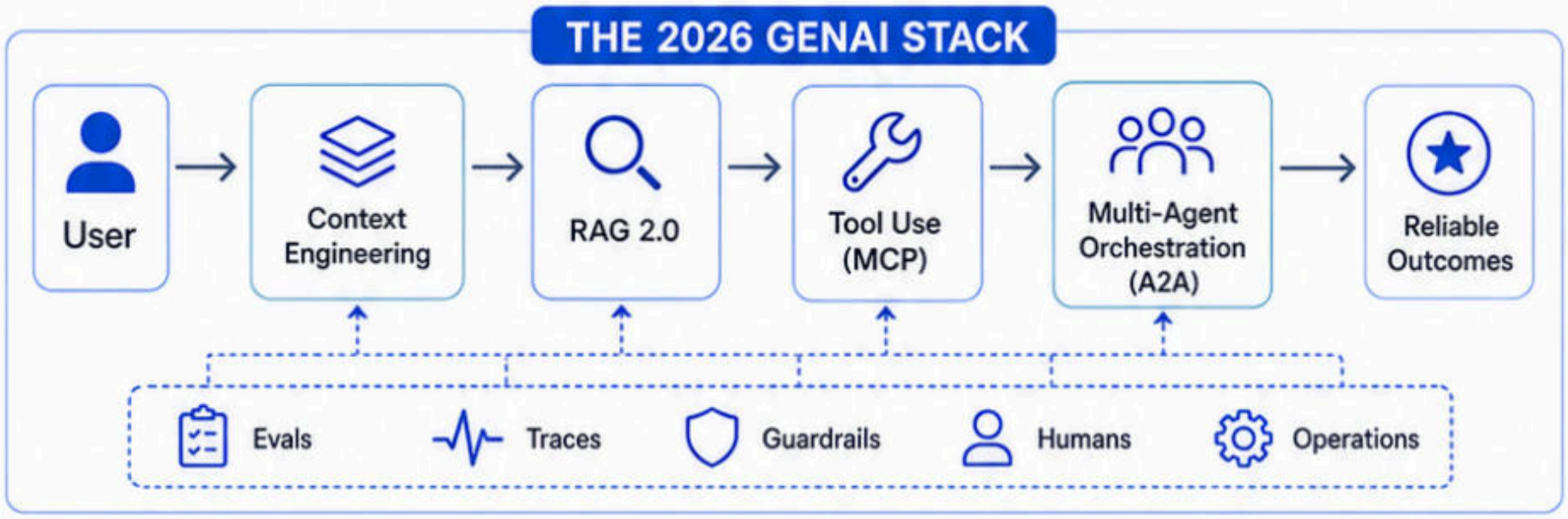
The future belongs to people who can turn LLMs into reliable systems.

- ✗ Not just impressive prompts.
- ✗ Not just beautiful demos.
- ✗ Not just another chatbot.

Reliable systems.

With:

- context
- tools
- evals
- traces
- guardrails
- humans
- operations



That is the 2026 GenAI stack.

Save this for your future self!