MCP Dev Summit Mumbai

Building a Context Engine for Multi-Agent Systems using MCP

Agents that **read**, **write**, and **reason** over shared, persistent context.

Presented by
Saradindu Sengupta

60 MINUTES, END TO END

Agenda

1	**Frame**	Why static RAG breaks in orchestrated, multi-session workflows	5m
2	**Fundamentals**	Entities, relationships, events, facts — knowledge graph → context graph	10m
3	**Build the graph**	Schema for a real workflow; ingest & normalize	15m
4	**Expose via MCP**	Server endpoints; query + update tool patterns	15m
5	**Wire up agents**	Agents reading & writing context; state across sessions	10m
6	**Scale**	Multi-agent coordination patterns & pitfalls	5m

THE PROBLEM, CONCRETELY

A decision was made. The reasoning vanished.

In the codebase

`Inference_timeout = 30s`
It used to say `10s`. Someone changed it.

`git blame` shows **who**. The **why** is gone.

In the CRM

A renewal agent approves a **20% discount** — policy caps it at 10%.

Incidents, escalations, a prior VP exception drove it.
The record keeps one fact: **“20% discount.”**

Infrastructure for what is true now exists— and almost nothing for **why it became true.**

FIRST PRINCIPLES

Context is the whole game

An LLM is not a pure function.

```
output = f(context) + Var
```

Short of training weights, the only lever on output quality is the quality of what you put in.
In-context learning is the paradigm: behaviour is steered by the window, not the weights.

An agent's context window should optimise for:

Correctness

no bad facts in the window

Completeness

nothing essential missing

Size

fewer tokens → better results

Trajectory

the right path through the work

Dong et al., "A Survey on In-context Learning" (arXiv:2301.00234) · Horthy, "Advanced Context Engineering for Agents"

THE DISCIPLINE

Treat the window as a budget, not a bucket

< 40%

target context utilisation
during long-running work

The worst things to put in a window, in order:

1. **Bad info**
   wrong facts poison every downstream step
2. **Missing info**
   the model can't reason about what it can't see
3. **Noise**
   irrelevant blobs crowd out what matters

Move work through `research` → `plan` → `implement`, compacting intentionally at each hand-off.

NECESSARY, BUT NOT SUFFICIENT

What RAG retrieves — and what agents actually need

RAG gives you

- Text chunks similar to your query
- Stores similarity, not meaning
- A static, read-only source
- Big JSON blobs that flood the window

Agents need to reason over

- Identity — who is this, across tools?
- Relationships — who owns / depends on what
- Temporal state — what was true, and when
- Why — the decisions behind the state

Marple (Graphlit), "The Context Layer AI Agents Actually Need" (Dec 2025)

FOUNDATIONS

From knowledge graph to context graph

Entities and relationships are table stakes. Events, facts, and decisions are the value.

TWO THINGS, OFTEN CONFLATED

Knowledge graph vs. context graph

Knowledge graph

A model of entities and the stable relationships between them. Answers “what is connected to what.” Largely a snapshot of current state.

Context graph

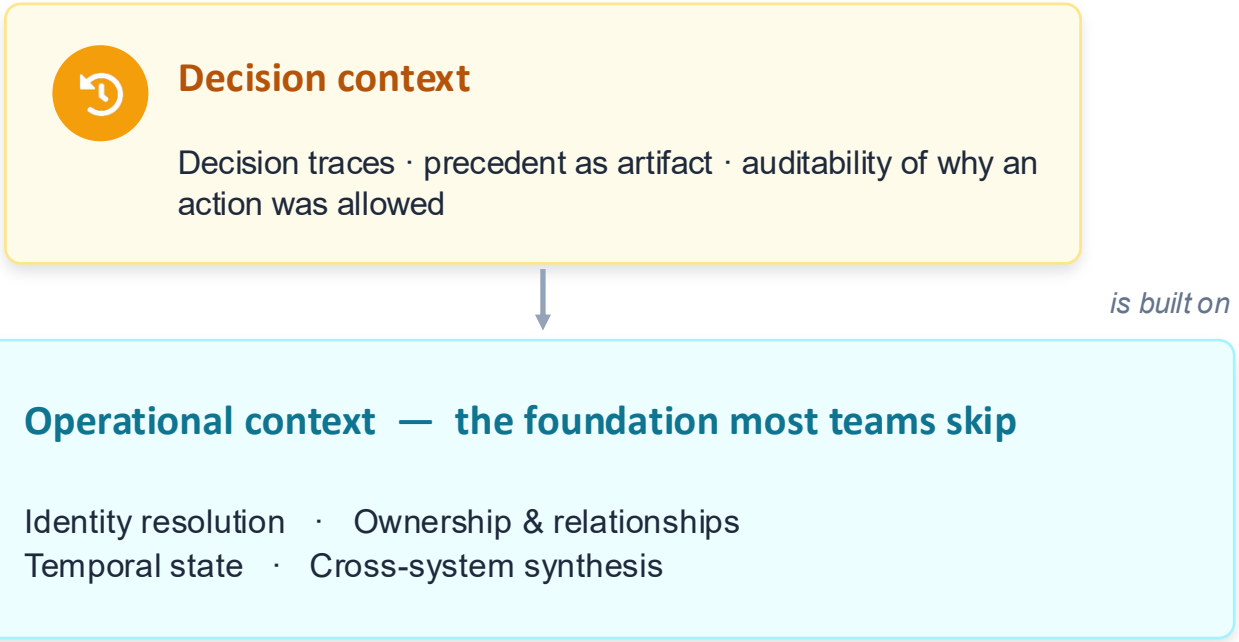
A living record of decision traces stitched across entities and time — so precedent becomes searchable. It explains not just what happened, but why it was allowed to.

The context graph is built *on top of* the knowledge graph — you can't capture **why** without first knowing **who** and **what**.

Foundation Capital (Gupta & Garg), “Context Graphs: AI's Trillion-Dollar Opportunity”

BUILD ORDER MATTERS

Two layers of context



Marple (Graphlit), Part 1 — operational context is the substrate decision traces depend on

THE DISTINCTION THAT UNLOCKS AUTONOMY

Rules vs. decision traces

RULES

What should happen in general

"Use official ARR for reporting."

Stable. Declarative. Necessary — but blind to the exception in front of you.

DECISION TRACES

What happened in this case

"Used X under policy v3.2, with a VP exception, based on precedent Z."

Specific. Replayable. The substance precedent is made of.

Agents don't just need the rulebook. They need the case law.

THE CORE IDEA

Two clocks

Every system has a state clock and an event clock. We over-built one of them.

A FRAMING WORTH STEALING

The state clock and the event clock

State clock

What is true right now.

Databases, CRMs, config files. We've built elaborate, trillion-dollar infrastructure here.

Well solved.

Event clock

What happened, in what order, with what reasoning.

The reasoning connecting observations to actions. It lived in heads, Slack threads, and meetings no one recorded.

Barely exists. This is the opportunity.

Koratana (PlayerZero), via Marple (Graphlit), "Building the Event Clock"

TIME IS ONE AXIS, NOT THE ONLY ONE

Context lives on five axes at once

The win is querying them **together**: “what did we discuss about Acme in NY-based meetings during Q3?”

THE IMPLEMENTATION SHAPE

Three layers: content → entities → facts

Facts	What content asserts — temporal claims about the world. This is the event clock.	*"Started Drug A on 2024-03-15."*
Entities	What content mentions — people, orgs, products. Where identity resolution happens.	*"Sarah Chen" = "S. Chen" = "@sarah"*
Content	The state clock — immutable source documents. The evidence trail, never edited.	*the raw email / transcript / commit*

Progressively more structured: raw evidence → identity → temporality + assertion.

THE UNLOCK

Make facts first-class, with validity in time

Fact

text	"Paula works at Microsoft as Principal Eng."
validAt	2024-03-15
invalidAt	null (still current)
status	Canonical | Superseded | Synthesized
mentions	Paula → Person ; Microsoft → Org
source	→ content #4821 (evidence chain)

- **Query, don't guess**

  "current employer?" = facts where invalidAt is null — not "retrieve recent docs and hope."

- **Synthesized facts**

  Derive "worked at Google 2020–2024" from point-in-time observations; keep the evidence chain.

- **A world model**

  Static models behave as if they learn — the evidence base grows, not the weights.

Marple (Graphlit), "Building the Event Clock" — facts with validAt / invalidAt

DON'T FALL INTO THE TRAP

Schema: adopt foundations, then learn what's novel

Prescribed

Palantir's bespoke way

Hand-built ontology per customer. Works, but slow and expensive.

Learned

“the ontology emerges”

Structure emerges from agent trajectories. Elegant — but ignores 20 years of prior art.

Adopt + extend

the pragmatic third path

Reuse Schema.org, CDM, FHIR, FIBO. Extend where the domain needs it.

Entities (Person, Account, Service) are solved. Spend your modelling effort where it's genuinely unsolved: **temporal validity, decision traces, fact resolution.**

Marple (Graphlit), “What the Ontology Debate Gets Wrong” · structure is learnable: Grover & Leskovec, node2vec (arXiv:1607.00653)

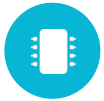
FROM IDEA TO PROTOCOL

Memory, and why MCP

The window is RAM. The context graph is disk. MCP is the system call between them.

BORROWED FROM OPERATING SYSTEMS

The agent as an LLM operating system



Context window = RAM

Fast, tiny, expensive. Holds the system prompt, tool schemas, memory blocks, files, and the message buffer.

Memory blocks (from MemGPT) are reserved, size-limited, self-editable slots of persistent context.

Context graph = disk

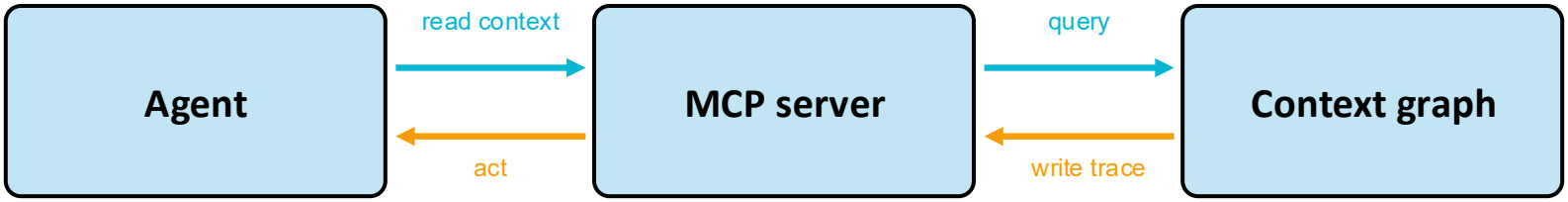
Large, durable, cheap. Holds entities, relationships, facts, and decision traces across sessions and agents.

Tools are the system calls that page context in — and write results back out.

Packer et al., MemGPT (arXiv:2310.08560) · Letta, "Anatomy of a Context Window"

THE POINT OF TODAY

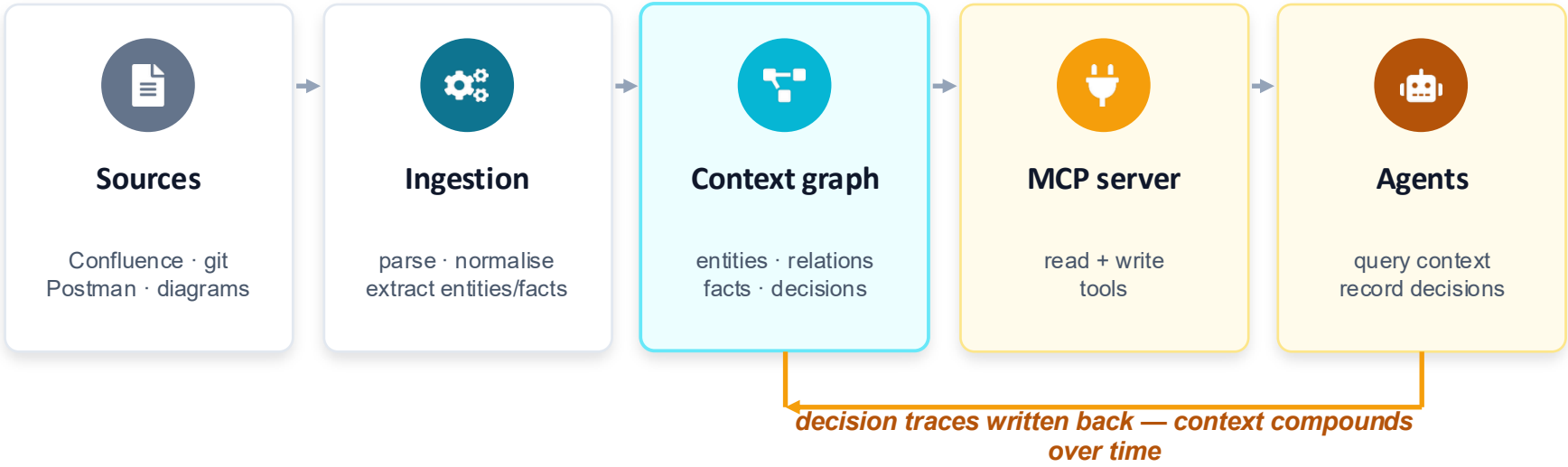
MCP closes the loop: read and write context



- Most MCP servers are read-only vector wrappers.
- Bidirectional servers let agents update their own state — closing the loop between action and memory.
- A specialised, graph-backed MCP server becomes long-term memory for the whole agent ecosystem.

THE SYSTEM WE BUILD TODAY

Anatomy of a context engine



Reads pull structured context in; writes persist new decisions — the loop the workshop builds end to end.

PART

01

HANDS-ON · 15 MIN

Build a minimal context graph

Schema for a real workflow, then ingest and normalise.

HANDS-ON PART 1

A context graph for an engineering workflow

Our domain

A small microservices system — services, owners, deploys, incidents, and the decisions behind them. Real domain data, not toy examples.

Schema sketch

```
(:Service)-[:OWNED_BY]->(:Team)
(:Service)-[:DEPENDS_ON]->(:Service)
(:Incident)-[:AFFECTS]->(:Service)
(:Decision)-[:ABOUT]->(:Service)
(:Fact {validAt, invalidAt})
```

What you'll do

1. Model entities & relationships (the knowledge graph).
2. Add Fact nodes with validAt / invalidAt (the event clock).
3. Add Decision nodes linking inputs → action → rationale.
4. Load the seed dataset; normalise & resolve identities.
5. Sanity-check with a few Cypher reads.

Outcome: a queryable graph that knows who owns what, what changed when, and why.

PART

02

HANDS-ON · 15 MIN

Expose context via MCP

Stand up the server. Design tools for query and update.

HANDS-ON PART 2

Tools, not blobs: the MCP surface

Server tool surface

`get_service_state`	live facts + open incidents
`get_event_timeline`	ordered deploys, incidents, decisions
`find_precedent`	prior decisions + outcomes
`record_decision`	write a decision trace back
`add_fact`	append a temporal fact

cyan = read **amber = write**

Design patterns

- Return shaped results, not raw JSON dumps — blobs flood the window.
- Scope every read (entity + hop limit + time) so payloads stay small.
- Make writes explicit and structured: inputs, action, rationale, actor.
- Stamp provenance on every write — source + timestamp + confidence.

Stack: Python + FastMCP over the graph store. Keep tool returns terse — the window is a budget.

PART

03

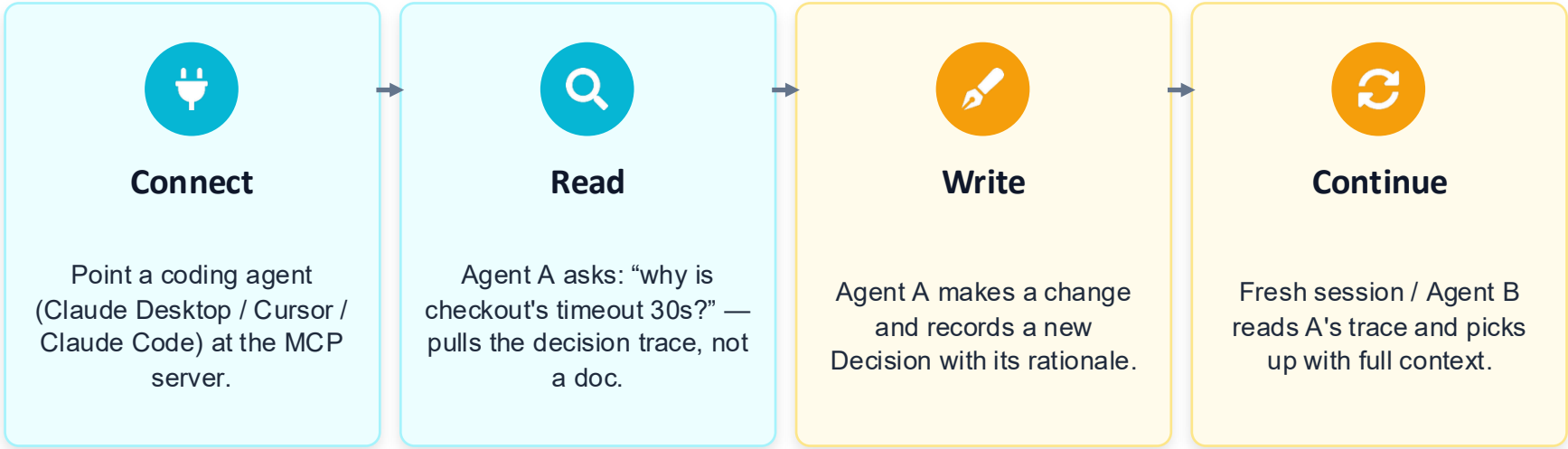
HANDS-ON · 10 MIN

Wire up the agents

Read context, write a trace, prove continuity across sessions.

HANDS-ON PART 3

Closing the loop across sessions & agents



Continuity across sessions and agent boundaries is the payoff — and the thing chat history alone can't give you.

PRODUCTION

Scaling for orchestration

Patterns and pitfalls for shared context across many agents.

MULTI-AGENT COORDINATION

Patterns that scale, pitfalls that don't

Patterns

- One shared context layer; agents read & write it
- Sub-agents for context isolation, not org-charts
- Compact at every hand-off (research/plan/impl)
- Provenance + supersession to resolve conflicts

Pitfalls

- Tool returns that dump huge JSON into context
- Adding/removing tools mid-run (breaks KV-cache)
- Volatile prompt prefixes (timestamps, ordering)
- Hiding errors — keep failures in context to adapt

Synthesised from Horthy (HumanLayer) and Ji, "Context Engineering for AI Agents — Lessons from Building Manus"

THE TAKEAWAY CHECKLIST

A context-engineering playbook

- Engineer the window: correctness > completeness > size > trajectory.
- Move work through research → plan → implement; compact intentionally.
- Keep the prefix stable and context append-only (protect the KV-cache).
- Prefer facts with validity over retrieved chunks; query, don't guess.
- Adopt standard ontologies; learn only the temporal & decision layers.
- Write decision traces at commit time — context that compounds.

IF YOU REMEMBER FIVE THINGS

Key takeaways

Context is the lever

Output quality is bounded by what's in the window — engineer it deliberately.

Graph over chunks

Model identity, relationships, time, and decisions — not just similar text.

Capture the event clock

Persist decision traces with temporal validity so precedent is queryable.

MCP makes it bidirectional

Read and write context through tools; close the loop between action and memory.

Context compounds

Shared, persistent context is what makes multi-agent orchestration reliable.

Future Work

Reading list

- Foundation Capital — Context Graphs: AI's Trillion-Dollar Opportunity (decision traces; systems of record for decisions).
- Graphlit (Kirk Marple), 3-part series — The Context Layer AI Agents Need; Building the Event Clock; What the Ontology Debate Gets Wrong.
- Letta — Anatomy of a Context Window: A Guide to Context Engineering (LLM-OS; memory blocks).
- Manus — Context Engineering for AI Agents: Lessons from Building Manus (KV-cache, mask-don't-remove, file system as context, recitation, keep errors in, don't get few-shotted).
- Dex Horthy (HumanLayer) — Advanced Context Engineering for Agents (research→plan→implement; intentional compaction; sub-agents; <40% utilization).
- Papers:
    - MemGPT: Towards LLMs as Operating Systems (2310.08560);
    - node2vec: Scalable Feature Learning for Networks (1607.00653);
    - A Survey on In-Context Learning (2301.00234).

Workshop Repo: https://github.com/saradindusengupta/context-engine

Questions

Github LinkedIn Website

Thank You