WSO2CON NORTH AMERICA

May 20 - 22, 2026 | Austin, Texas, USA

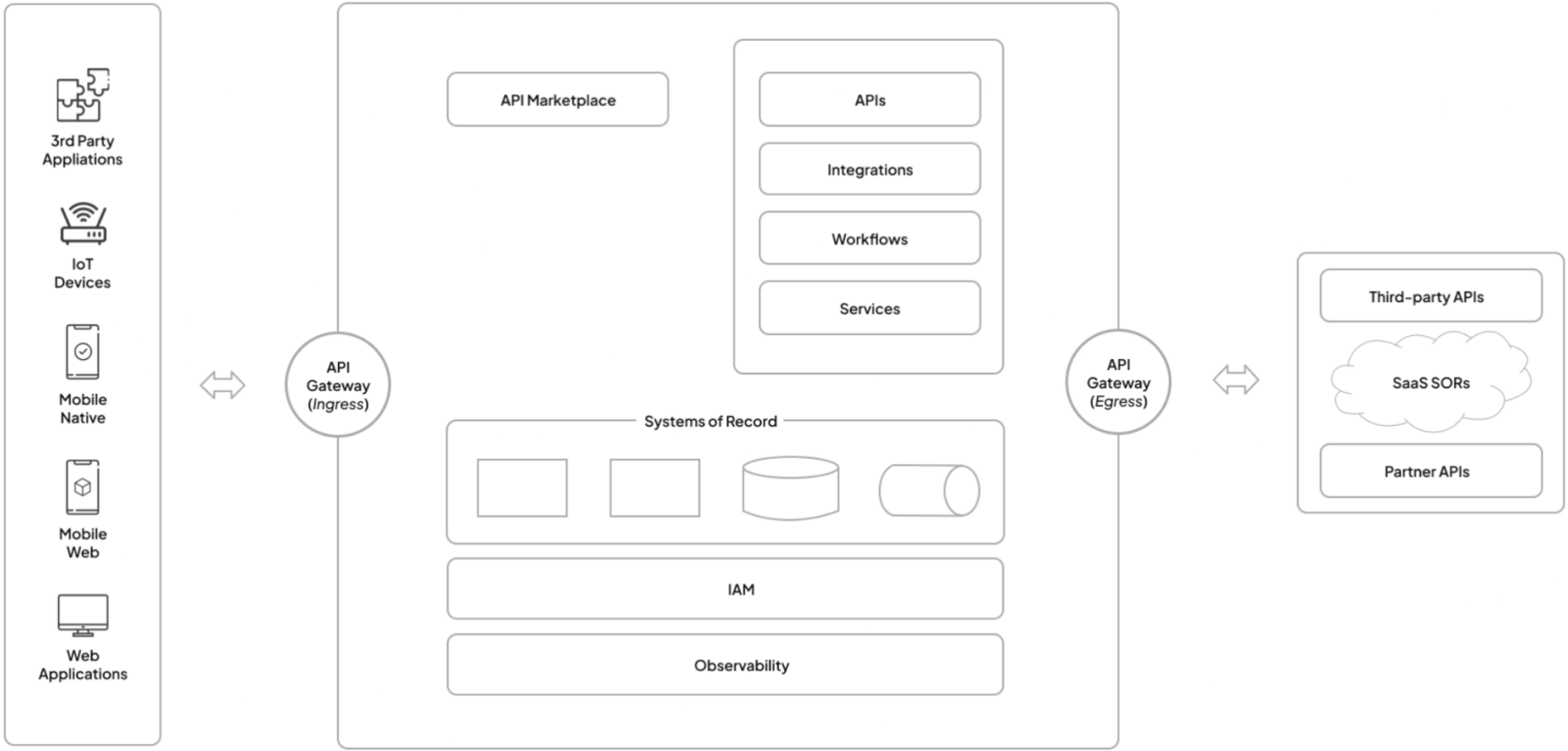
A Practical Guide to AI Agents in the Enterprise

Chintana Wilamuna

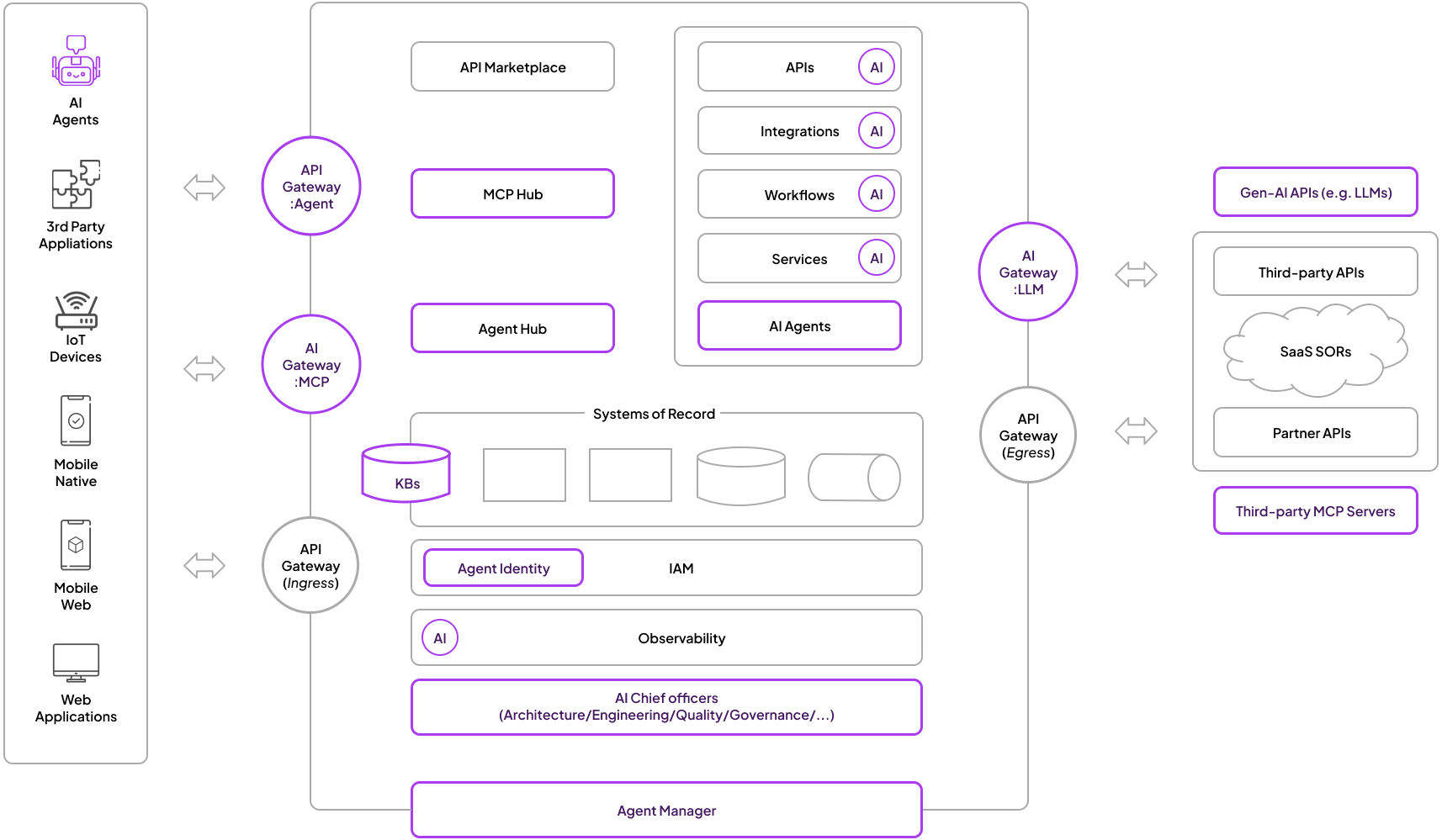
VP Sales Engineering
chintana@wso2.com

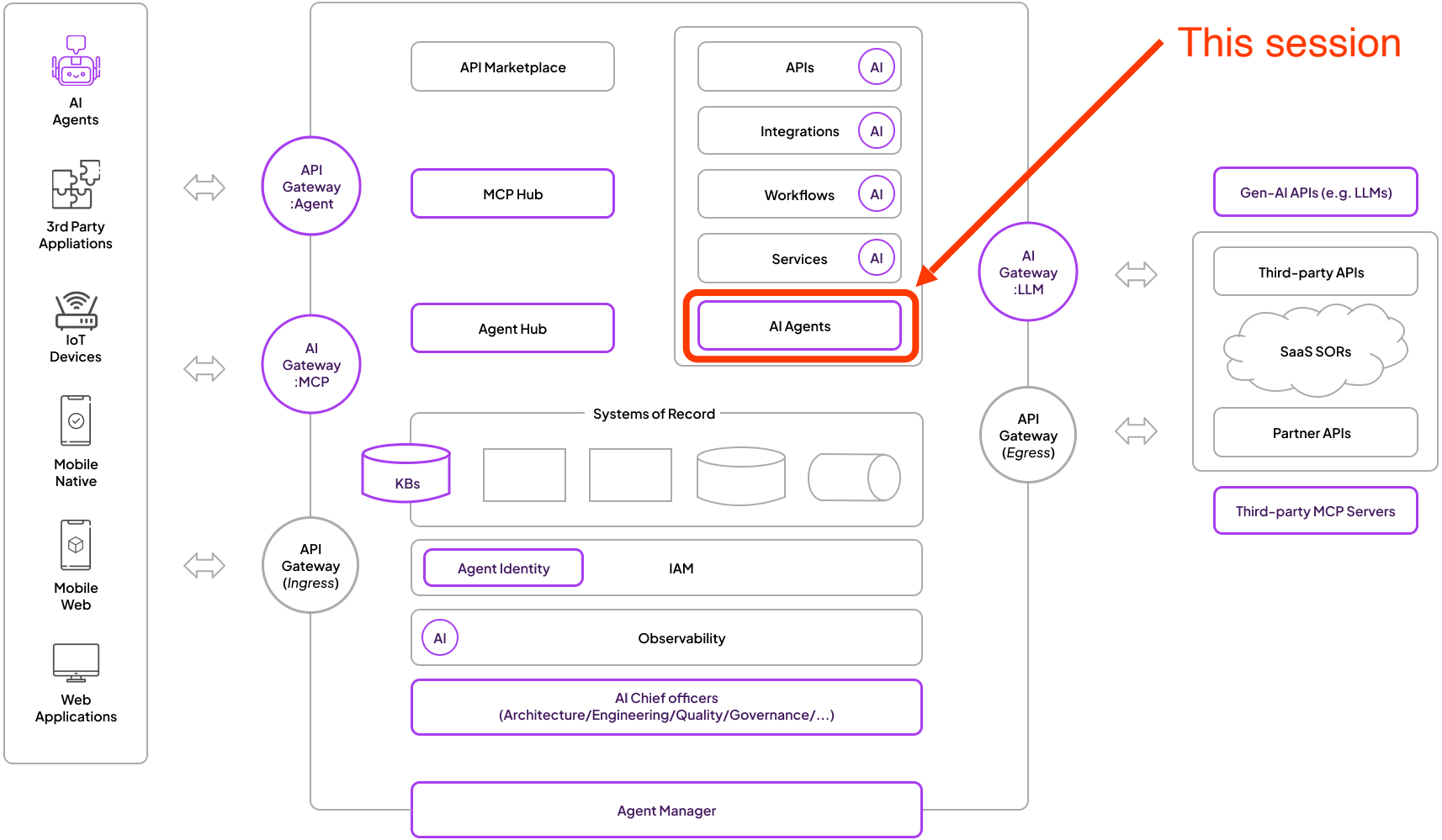
Nadheesh Jihan

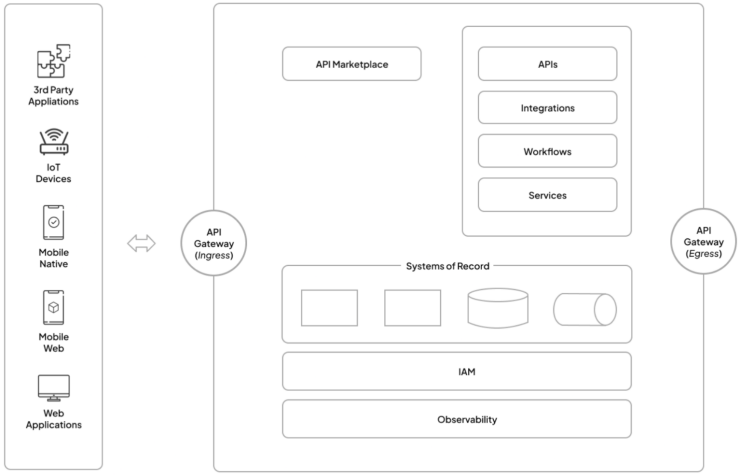
Senior Technical Lead
nadheesh@wso2.com



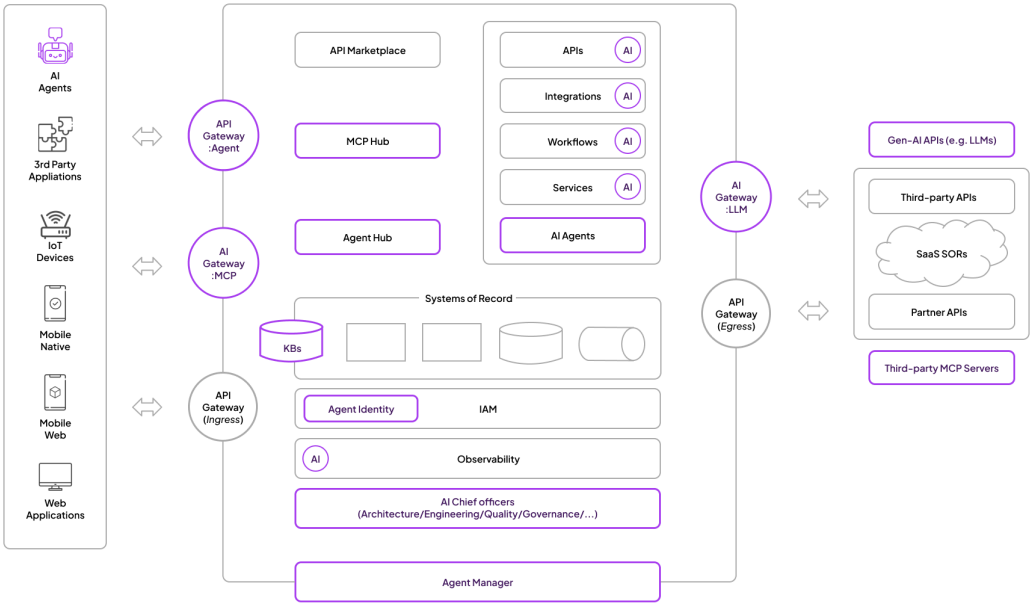
Ref: https://wso2.com/library/blogs/agentic-enterprise-with-wso2/

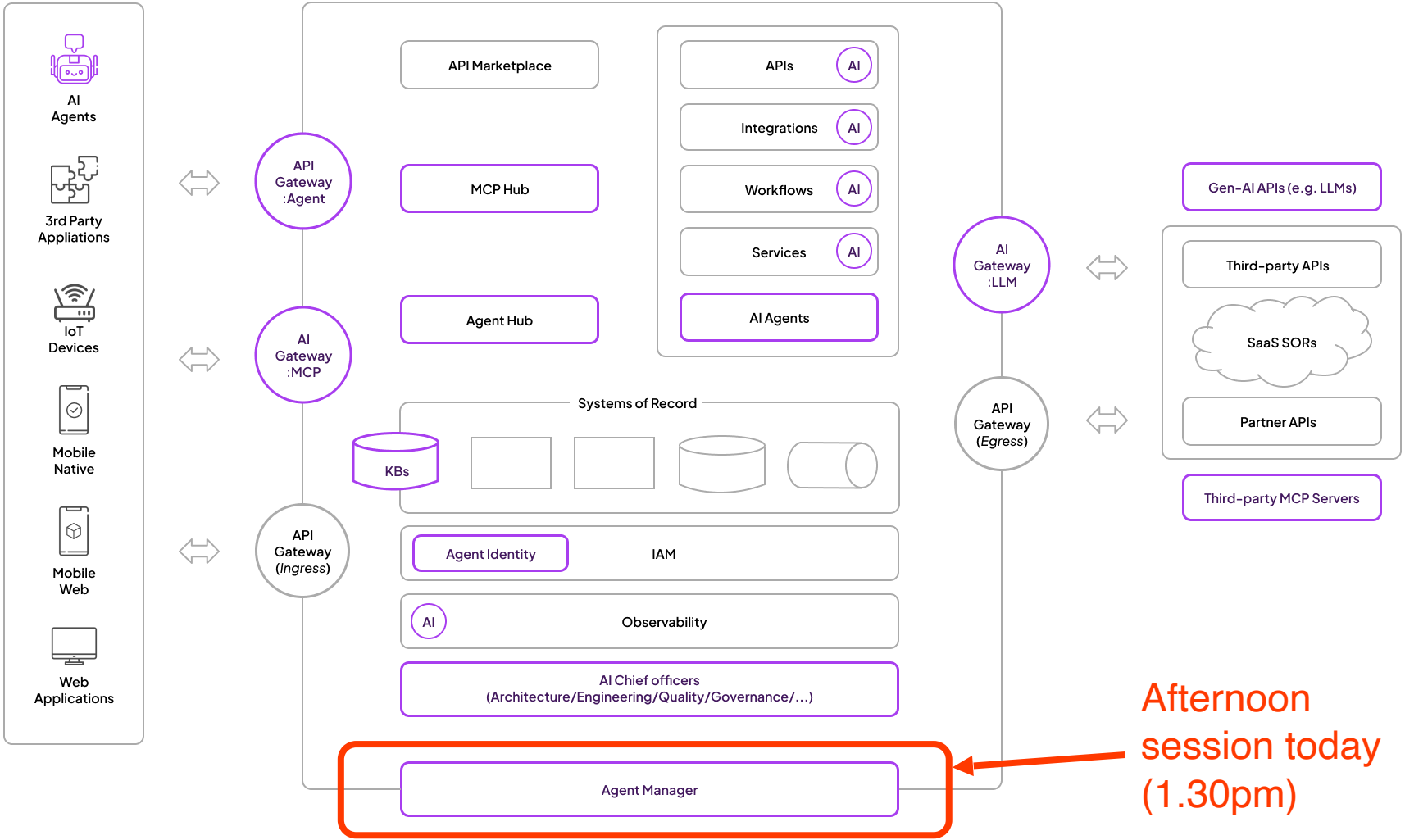

AI Agents
3rd Party Appliations
IoT Devices
Mobile Native
Mobile Web
Web Applications
API Gateway :Agent
AI Gateway :MCP
API Gateway (Ingress)
API Marketplace
MCP Hub
Agent Hub
APIs
AI
Integrations
AI
Workflows
AI
Services
AI
AI Agents
Systems of Record
KBs
Agent Identity
IAM
AI
Observability
AI Chief officers
(Architecture/Engineering/Quality/Governance/...)
Agent Manager
AI Gateway :LLM
API Gateway (Egress)
Gen-AI APIs (e.g. LLMs)
Third-party APIs
SaaS SORs
Partner APIs
Third-party MCP Servers

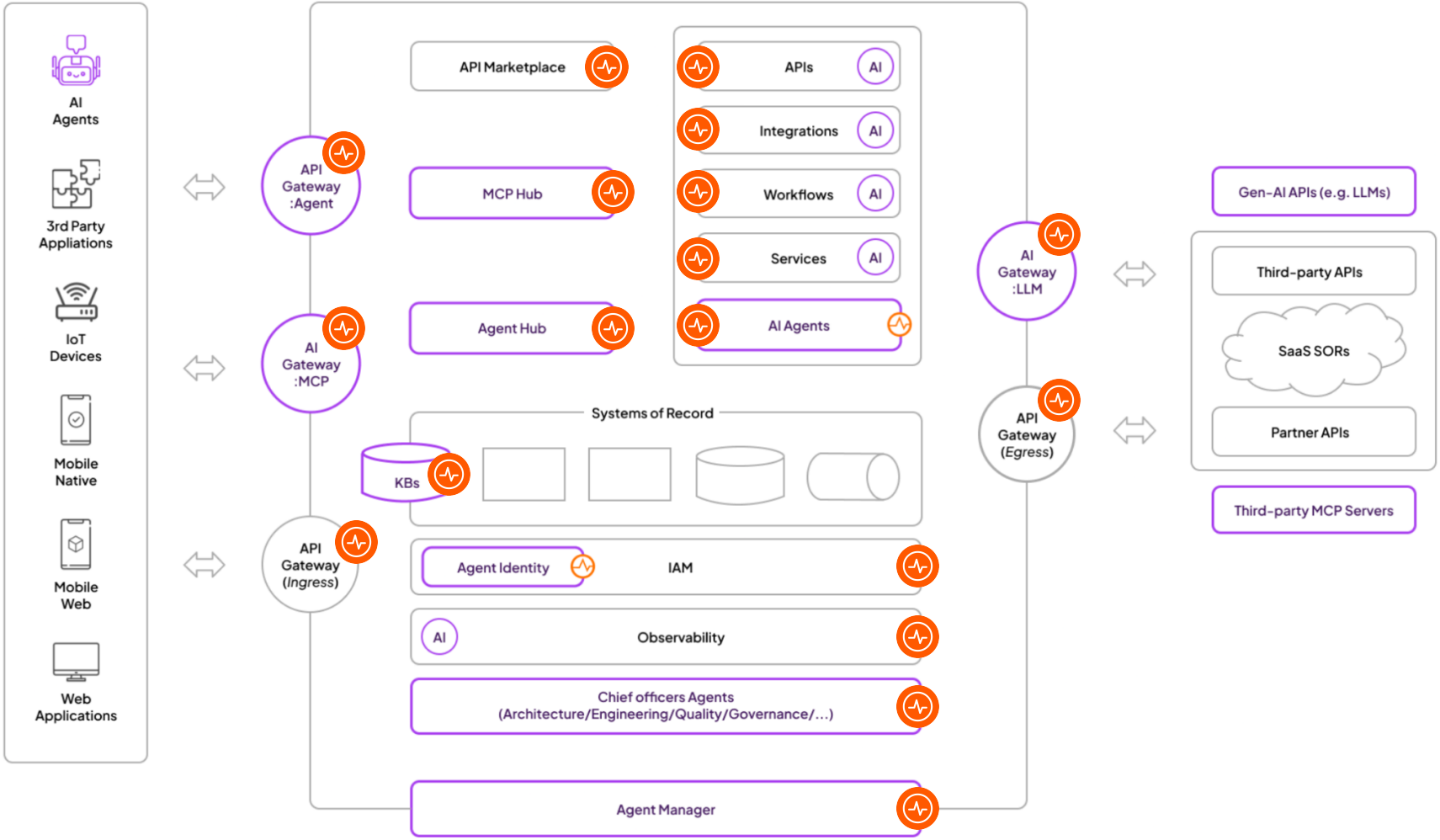

AI Agents
3rd Party Appliations
IoT Devices
Mobile Native
Mobile Web
Web Applications
API Gateway :Agent
AI Gateway :MCP
API Gateway (Ingress)
API Marketplace
MCP Hub
Agent Hub
APIs
AI
Integrations
Workflows
Services
AI Agents
Systems of Record
KBs
Agent Identity
IAM
Observability
AI Chief officers (Architecture/Engineering/Quality/Governance/...)
Agent Manager
AI Gateway :LLM
API Gateway (Egress)
This session
Gen-AI APIs (e.g. LLMs)
Third-party APIs
SaaS SORs
Partner APIs
Third-party MCP Servers



Next session (11am)




AI Agents
3rd Party Appliations
IoT Devices
Mobile Native
Mobile Web
Web Applications
API Gateway :Agent
AI Gateway :MCP
API Gateway (Ingress)
API Marketplace
MCP Hub
Agent Hub
APIs
AI
Integrations
AI
Workflows
AI
Services
AI
AI Agents
Systems of Record
KBs
Agent Identity
IAM
AI
Observability
AI Chief officers
(Architecture/Engineering/Quality/Governance/...)
Agent Manager
AI Gateway :LLM
API Gateway (Egress)
Gen-AI APIs (e.g. LLMs)
Third-party APIs
SaaS SORs
Partner APIs
Third-party MCP Servers
Afternoon session today (1.30pm)


AI Agents
3rd Party Appliations
IoT Devices
Mobile Native
Mobile Web
Web Applications
API Gateway :Agent
AI Gateway :MCP
API Gateway (Ingress)
API Marketplace
MCP Hub
Agent Hub
APIs
AI
Integrations
AI
Workflows
AI
Services
AI
AI Agents
Systems of Record
KBs
Agent Identity
IAM
AI
Observability
Chief officers Agents
(Architecture/Engineering/Quality/Governance/...)
Agent Manager
AI Gateway :LLM
API Gateway (Egress)
Gen-AI APIs (e.g. LLMs)
Third-party APIs
SaaS SORs
Partner APIs
Third-party MCP Servers

Agenda

- What an agent actually is
- The six production building blocks
- Unifying pattern: atomic code, intelligent tools & skills
- Live demo
- Recap and Q&A

What is an agent?

WSO2

What is an agent?

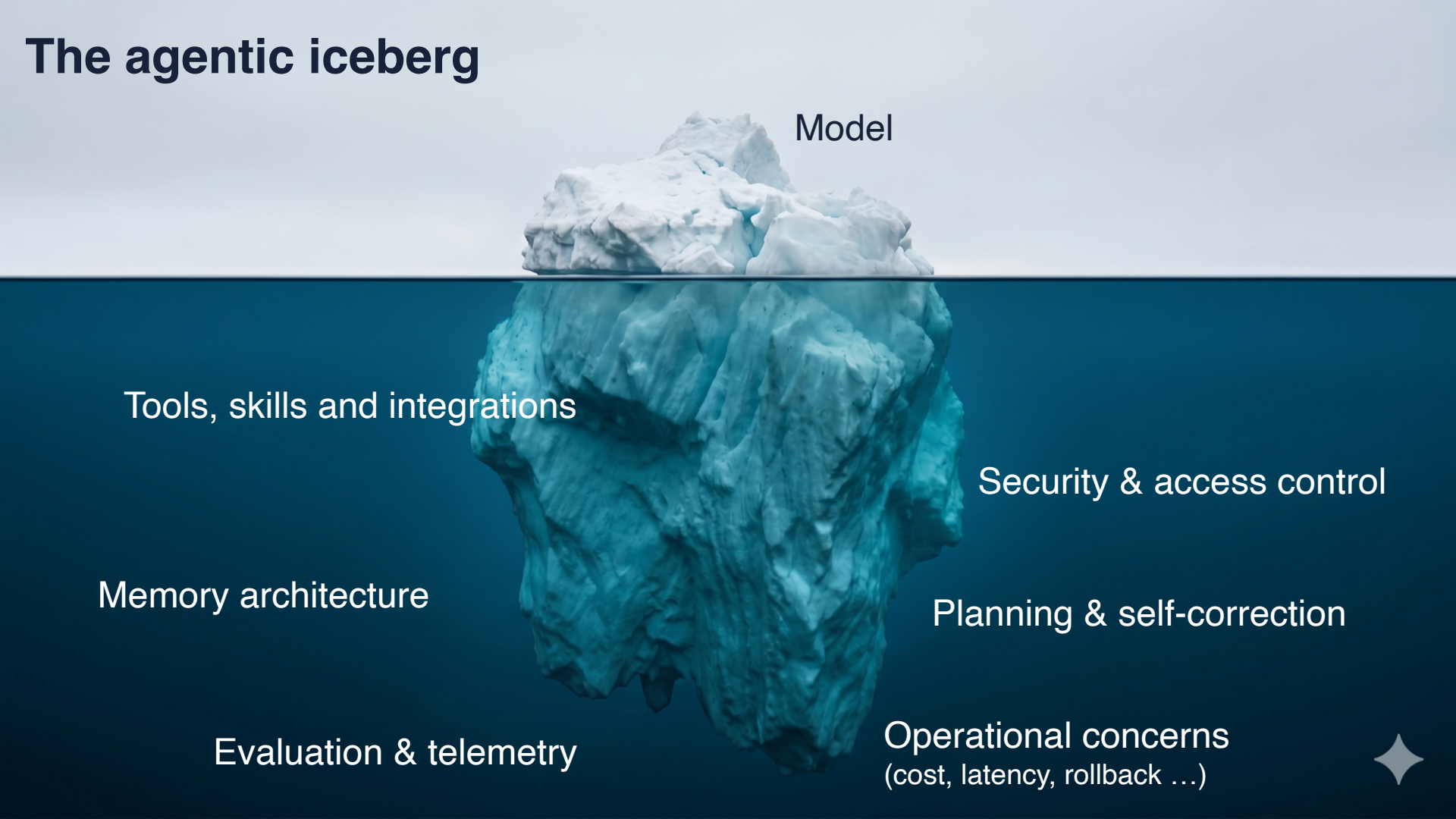
An agent is an **LLM** running in a **loop**, with **tools**, working towards a goal it can **decide** when it has met.

- **Loop** - the model gets to act multiple times
- **Tools** - the model can interact with the external world
- **Decide** - the model controls flow, not the developer (non-deterministic)

6 Production Building Blocks

WSO2

The agentic iceberg



1 Tools, skills & Integration

WSO2

Tools, skills & integrations

This is the highest-leverage design surface

Two truths

- A great model with bad tools is a bad agent
- A mediocre model with great tools is often a fine agent

Tools and skills

- **Tool** - capabilities an agent can invoke at runtime
- **Skills** - Instructions, retrieved into context that shape how the agent thinks and acts

Tool design principles

- Few tools, well shaped
  - Usually ~10 but experiment
  - Consolidate by intent, not by API endpoint
- Names are part of the prompt
  - **find_customer_by_by_email** is better than **query_users_v3**
- Descriptions are skills
  - Tell the model when to use it, not just what it does
- Inputs match how the model thinks
  - Natural-language slots over IDs, resolve IDs inside the tool
- Outputs guide the next step
  - Return structured data and a one-line “what to do next” hint when useful
- Errors are teachable
  - “Oder not found. Try search_order_by_customer first”

Dynamic skills

OpenClaw model

- Fixed agent harness
- Pluggable custom behavior using SKILL.md text files
- No code changes in the agent

2 Memory Architecture

WSO2

Memory architecture

Memory is not just to “give it more context”. It’s choosing what to remember, where and for how long

Memory types

- Working memory - current loop, current task
- Conversational memory - this session’s history
- Long-term semantic memory - facts, learned preferences, embeddings over past interactions
- Episodic memory - “what happened last time we tried this”

Memory patterns and pitfalls

Patterns

- Summarize on overflow, don’t truncate
- Separate stores by lifetime (turn / session / user / global)
- Retrieve, then inject - don’t dump the whole memory into context
- Be careful about write paths - be deliberate about what gets remembered

Pitfalls

- “Just stuff it all in the context window” - context rot is real, long contexts degrade reasoning
- Letting the agent freely write to long-term memory (poisoning)
- No invalidation strategy (memories that contradict each other)

3 Planning & self-correction

WSO2

Planning & self-correction

- ReAct - interleave thought + action, model decides each step
  - OpenAI SDK’s default behavior
- Plan-then-execute - generate a plan up front, execute, replan on failure
  - Useful when tool calls are expensive
- Decompose - break the task into sub-tasks, possibly via sub-agents
  - Suitable for large and complex tasks

There’s no single best solution. Choice depends on task structure and reversibility cost

Self-correction patterns

The single highest-leverage reliability move after good tools:

- Reflection - model critiques its own draft before finalizing
- Critic loop - separate “critic” call evaluates the “actor” output
- Verifier - deterministic check (lint, schema, tests) on outputs
- Replan on failure - observed error feeds back into the planner

Cost: extra tokens
Benefit: dramatic reduction in confident-but-wrong outputs

4 Evaluation

Evaluation

Non-deterministic systems break the testing instincts you grew up with

You cannot

- Assert exact outputs
- Trust that “it worked once” means “it works consistently over time”
- Rely on unit tests alone

You can

- Define success at the trajectory level, not just the final answer
- Build regression sets of real prior tasks
- Use LLM-as-judge for fuzzy quality dimensions
- Capture production telemetry as the largest eval set you’ll ever have

Eval: Trajectory vs outcome evaluation

- Outcome eval - did the final answer satisfy the goal?
- Trajectory eval - did the agent take a sensible path?

We need **both**

- Outcome catches *whether* it worked
- Trajectory catches *why* it worked, and whether it was lucky

Trajectory eval also surface tool-misuse, redundant calls, and silent recovery from earlier mistakes

5 Security

Security

Agents are the first widely deployed systems where the executor of code is also target of social engineering

Unique risks

- Prompt injection - via any text the agent reads (tool results, web pages, docs etc…)
- Excessive autonomy - agent does too much before a human check
- Excessive privileges - agent has permissions the requestor doesn’t
- Exfiltration via tool calls - a “search” tool used to leak data

Excessive autonomy and privileges

Design question for every tool: if this tool is called wrongly, what's the worst that happens?

Tier tools by "blast radius"

- Read-only on non-sensitive data - no problem
- Read on sensitive data - log and rate-limit
- Write on reversible state - log, allow add a critic
- Write on irreversible state (send email, payments, delete) - human-in-the-loop or strict guardrails

Defense in depth for agents

Layers, none is sufficient by its own

- Input filters - strip/flag suspicious content before the LLM call
- Scoped tool permissions - agent can only see what this user can see
- Per-tool guardrails - schema validation, allowlist, rate-limits
- Critic/verifier on actions - second model checks high-risk calls
- Human approval - the last line, used sparingly but reliably
- Audit logging - everything, immutable, queryable

6 Operational concerns

Operational concerns

- Cost - long loops are expensive, cap tokens, turns, parallel sub-agents
- Latency - agents are slow, consider streaming, partial UIs, async patterns
- Versioning - model, prompts, tool defs, skills
- Rollback - can you revert a prompt change quickly?
- Observability

Unifying pattern

Shape of a production agent

Main agent code is boringly simple

```
while not done and steps < max_steps:
    decision = model.respond(context)          # reason
    if decision.is_final:
        return decision_text
    result = run_tool(decision.tool_call)# act
    context = update(context, result)          # observe
```

Where the intelligence lives

Agent atomic loop is dumb on purpose, intelligence layer:

Layer	What in has
System prompt	The agent’s role, constraints, escalation rules
Tool names	Vocabulary of available actions
Tool descriptions	When and how to use each tool, a skill module
Tool error messages	Recovery guidance - a skill module
Memory contents	What the agent knows now, retrieved
Critic prompts	Quality standards, applied at runtime
Dynamic skills	New capabilities and expertise

Getting the Agent Demo Running

1. Clone: https://github.com/wso2con/2026-AUS-AI-tutorial
2. Follow simple setup instructions: Lab-01/README.md
   - *make install*
   - Add your openai key to *.env*
   - *make dev*
   - UI running: http://localhost:5173

WSO2

Demo: Building Practical Agents

01 - Agent Definition

What differentiates an agent liability from an agent asset in your enterprise.

02 - Tool Design

How precise tool signatures and schemas dictate the bound of agentic intelligence.

03 - MCP & Skills

Leveraging MCP for decoupled boundaries and modular skills to provide specialized expertise.

04 - Memory

Implementing varied memory types and persistence scopes to maintain continuity.

05 - Planning

Thinking before acting: Why planning and validation is important for critical tasks.

06 - Security

Guards to protect agents against attacks and ensure policy compliance.

Defining the Useful Agent

Technically, it is a loop that **reasons** (LLM), **acts** (tools), and **observes**.

As agents mature, they gain skills, contextual memory, enterprise knowledge, and policy guardrails.

*But what criteria actually separates an unpredictable runtime loop — a potential **liability** — from a true, production-ready enterprise **asset**?*

Keep this central question in mind as we evaluate the code harnesses.

Demo Scenario: One Job, Two Architectures

Use Case: Resolve order issues for an EU retailer (Status, Refunds, Shipping, Cancellations).

Build v1: The "First-Cut"

Prompt-Heavy:
Identity, refund caps, and procedures live solely in the system prompt.

In-Process Tools:
A "god-tool" (cancel/refund/address) with optional params and free-text errors.

Legacy Context:
SOAP-styled history and atomic micro-getters.

Shared Instance:
One shared agent across all customers; no harness hooks or skills.

Build v2: Engineered Harness

Declarative Config:
Caps, MCP server lists, and memory layers defined in `agent-profile.yaml`.

MCP Services:
Scoped tools with typed params and structured errors.

Harness Hooks:
Identity binding and refund cap enforcement executed before the tool call ships.

Memory Topology:
Per-customer cache plus episodic memory and skill-based know-how.

"Same model. Same queries. The differences are everything around the LLM."

Witnessing the Live Agent

"This is a live agent. If things behave unexpectedly, congratulations — you've just witnessed real-world AI."

Tool Design over Model Choice

v1 Anti-Patterns

God Tool / Overloaded Input Schema

Poor tool documentation

Inconsistent Error Handling

Result: Hallucinations and wrong executions

v2 Production Harness

Proper tool descriptions

Enforced policy references

Better tool decomposition

Result: Predictable outcomes and helpful assistant.

Architecture Lesson: High-tier models paper over broken tool code at a 5× cost penalty. Fix the integration architecture once.

Can We Improve Further?

Current State (v2 Architecture)

Instead of a v1 "God-Tool", we decoupled execution into scoped MCP endpoints: cancel_order() and issue_refund(). LLM calculates deductions and executes multi-step chains.

1. Programmatic Entitlement

Concern:

Token tax and risk surface by making the LLM subtract floating percentages.

Solution: Move math to tool code; model passes intent only.

2. Atomic Transaction Chaining

Concern:

Failure window between cancel and refund creates orphan states.

Solution: Macro-skill execute_order_termination as one unit.

"Be cautious: over-optimizing an agent workflow can suppress agentic behavior and gradually turn it into a deterministic workflow."

Decoupling Boundaries: MCP & Skills

Model Context Protocol (MCP)

Crossing Enterprise Boundaries

- Tools are pluggable: agent-profile.yaml
- Maintained by domain expert teams
- Harness is developer responsibility

Dynamic Skills

Expertise Discovery

- Domain experts are often non-developers
- Text-based knowledge vs. code
  - Translates system use into agent tools
  - Enables new MCP workflows
- Pluggable, continuous expertise expansion: agent-profile.yaml and SKILL.md

"MCP decouples **where** *tools live; Skills decouple* **how** *agents strategize."*

WSO2 Agent-Flavored Markdown (AFM)

An open specification for defining AI agents. AFM is a powerful, markdown-based alternative to standard YAML profiles, enabling rich agent definitions across several key dimensions.

Agent Details
Core metadata: name, description, version, and author information.

Agent Model
Defines the specific AI model and parameters powering the agent.

Agent Interfaces
Defines invocation methods and the expected input/output signatures.

Agent Tools
External tool definitions accessible via protocols like MCP.

Agent Execution
Runtime configuration including limits and iteration logic.

Agent Skills
On-demand capabilities following the open Agent Skills standard.

Full Specification: github.com/wso2/agent-flavored-markdown

Memory Types & Scopes

State pollution occurs when an agent fails to separate temporary chat context from persistent data boundaries. Secure enterprise architectures require explicitly engineered isolation vectors.

Scope Layer	Intended Tier	v1 Prototype (Anti-Pattern)	v2 Production Harness
Session Memory	Session (Per-Thread)	Boundary Mismatch: Shared memory leak across customers.	Isolated interaction state per user thread at harness-level.
Episodic Memory	User / Tenant	*Absent. History lost once context window expires.*	Dynamically maps segregated customer flat files (.md).
Semantic Memory	User or Org	*Policy KB is integrated as a tool but does not enforced the use of it.*	Delegated to external policy_kb MCP microservice.
Procedural Memory	Agent Identity	*Baked into codebase; requires code updates to shift.*	Isolated skills/ directory parsed on initialization.

"Relying on a large context window does not establish a scalable memory architecture. Context is temporary sight; topology is resilient infrastructure."

The “LLM Vulnerability Window”

LLMs optimize for **conversational success**, not enforcement of hard business invariants.

Why Prompts are Insufficient

- Prompts can be socially manipulated
- Models prioritize “helpfulness” over policy
- Critical logic becomes probabilistic

Example Failure Modes

Financial Splitting

$250 refund split into $200 + $50

Identity Swapping

“Switch to this other account” bypasses auth

“Never let an LLM self-police critical security context or auth identifiers.”

The Core Axiom: Prompts guide behavior. Harnesses enforce policy.

Green Room session line up

- 11am Evolving Your Enterprise Architecture for the Agentic Era

  This lab explores how traditional enterprise architectures must evolve to support the autonomous reasoning and non-deterministic nature of AI agents. It provides a practical blueprint for modernizing identity, observability, and governance patterns to ensure agentic workloads are secure, scalable, and production-ready.

- 1.30pm AI Agent Manager

  The WSO2 Agent Manager serves as an open control plane designed to deploy, observe, and govern AI agents across an organization regardless of their underlying framework. This hands-on lab provides a practical guide to the five pillars of agent management, ensuring teams can securely monitor and evaluate agentic workloads for confident production deployment.

- 3.30pm AI Agent Manager [continued]

Thanks!