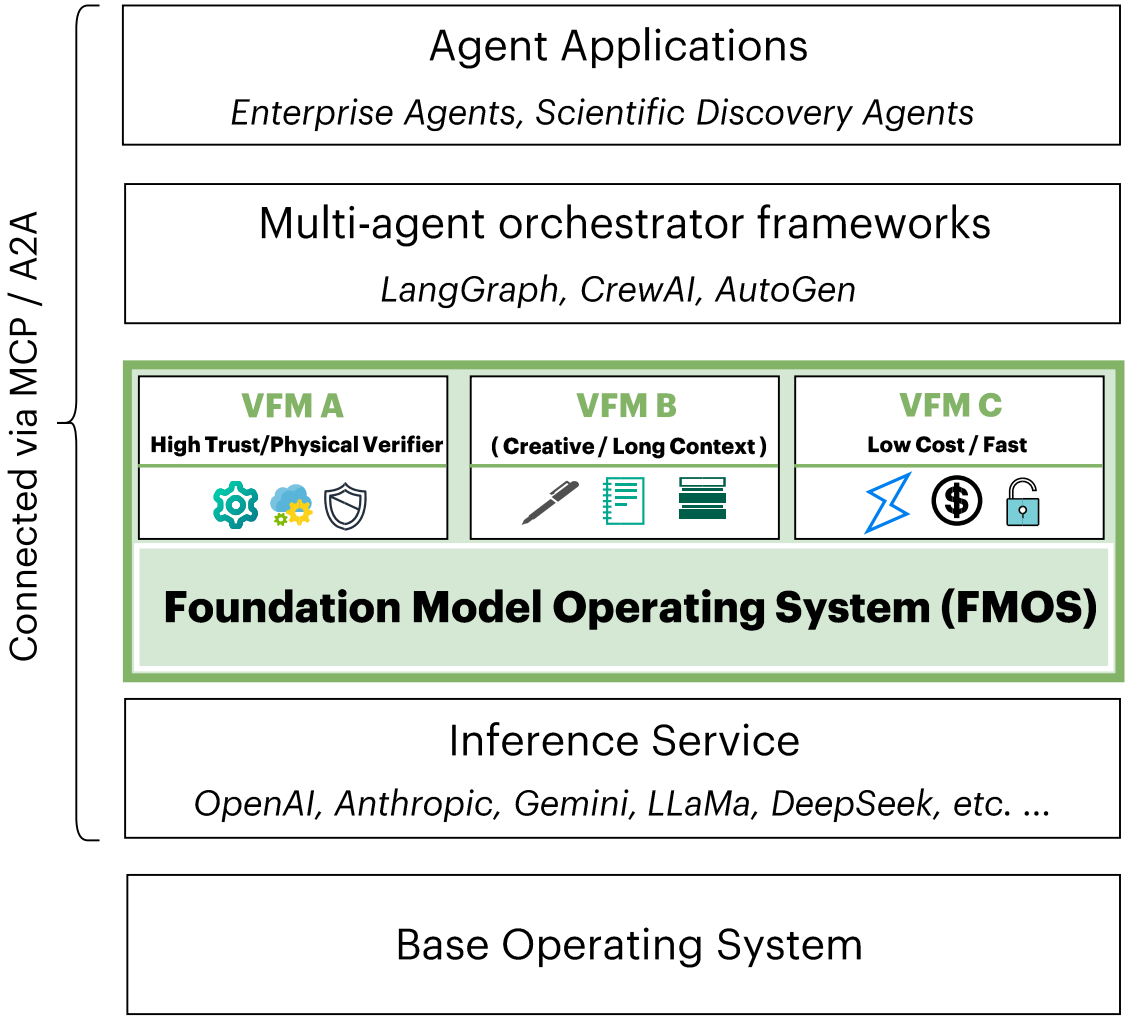
It is Time to Virtualize Foundation Models with a Self-evolving OS Layer

Suparna Bhattacharya, Tarun Kumar, Cong Xu, Satish Kumar Mopur, Jiahao Li, Ashish Mishra, Aalap Tripathy, Annmary Justine, Martin Foltin, Ian Foster[2,3]

HPE Labs, Argonne National Laboratory[2], University of Chicago[3]

ASPLOS 2026 Agentic OS Workshop, 3/23/2026

Filling in the Foundation Model Virtualization Gap

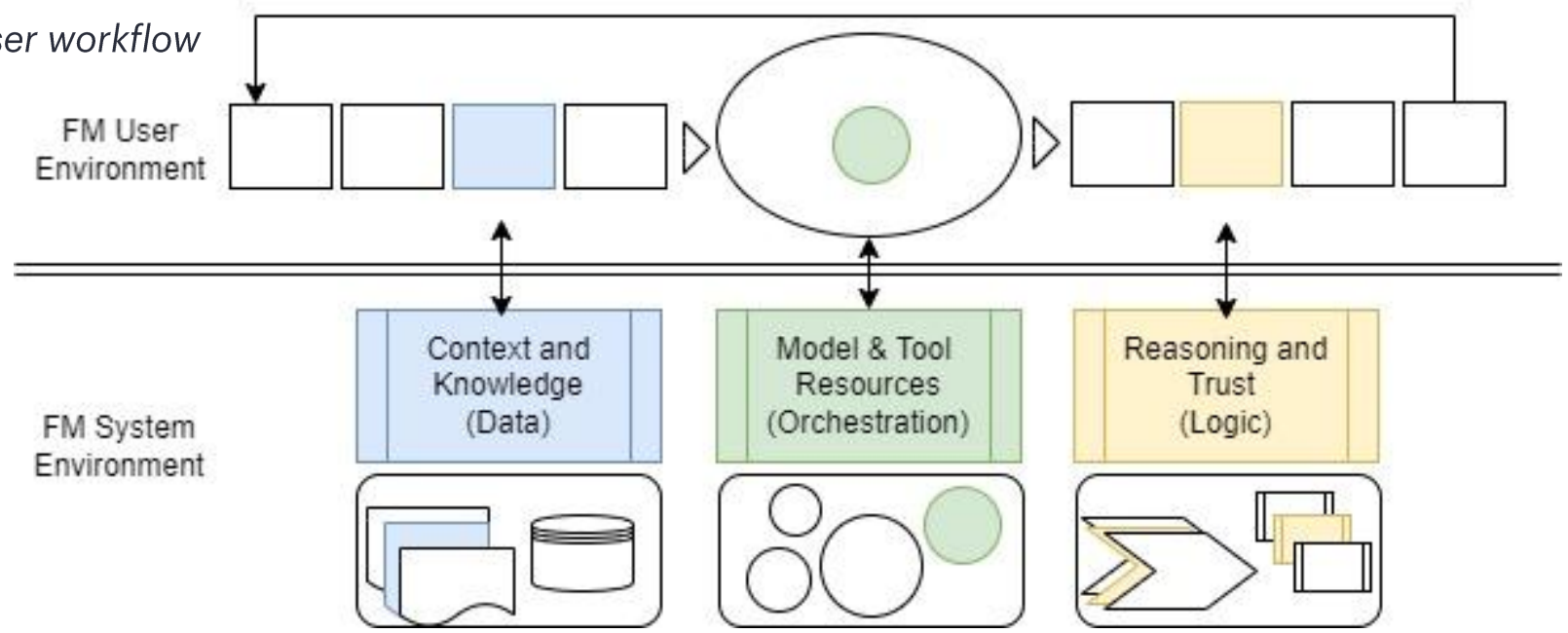


- Today, each of a plethora of multi-agent frameworks embeds an implicit runtime for core services (state, memory, budget, guardrails)
- Non-portable behavior, brittle governance, difficult extensibility, reminiscent of pre-OS software era
- Need for a system layer that separates core services and virtualizes FM interactions analogously to hypervisors: an **FMOS**
- **Virtual Foundation Models (VFM)** analogous to Virtual machines: illusion of dedicated, trust-worthy FM instances with unbound capabilities
- Virtualization designed for progressive quality gain of **VFM** through **self-evolution** (improve prompts, memories, tool calling)

Enabling unified services, joint optimization, and reusability

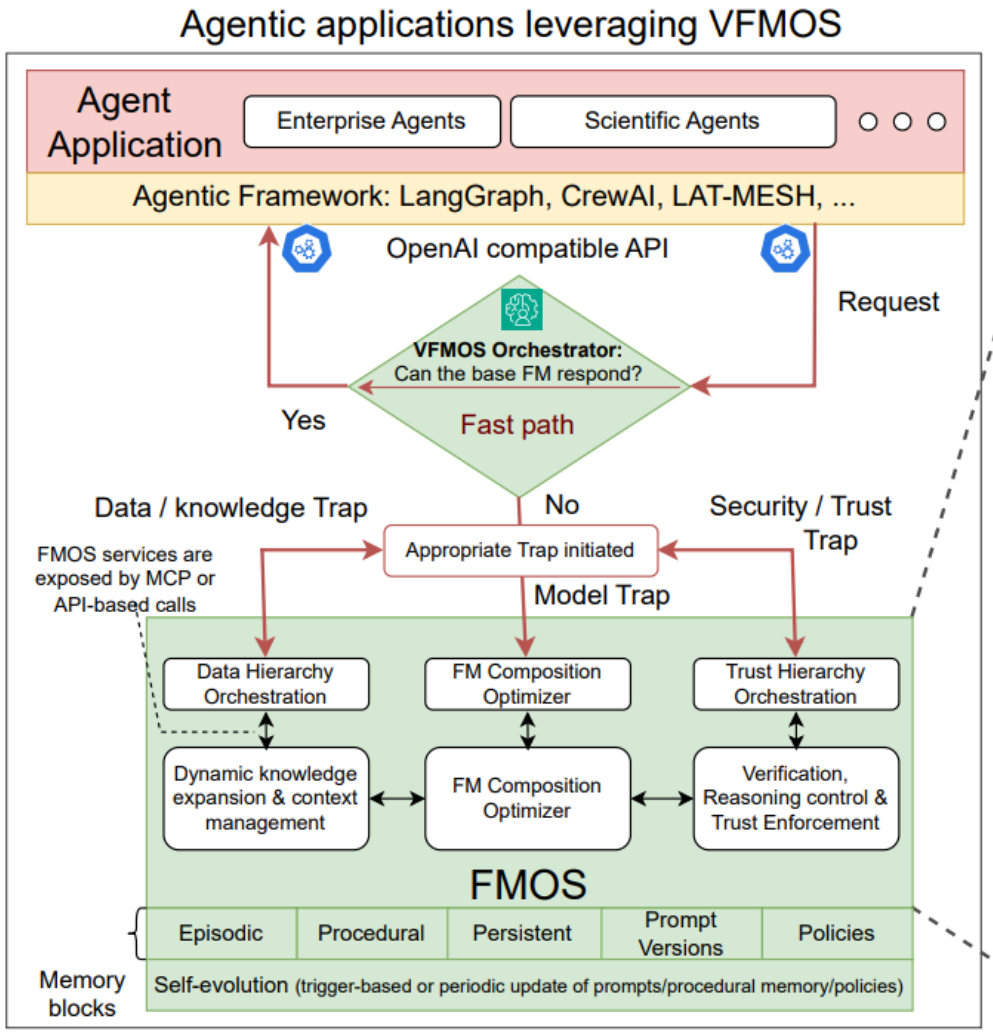
- **Unified services with collective learning:** Context management, tool orchestration, and verification are handled at the FMOS layer; learned improvements propagate across dependent workloads
- **Joint optimization:** The FMOS coordinates knowledge retrieval, model selection, and verification depth as a combined optimization problem—achieving efficiencies that fragmented stacks cannot
- **Cross-enterprise reusability:** Domain-specific skills, knowledge augmentations, and reasoning policies can be defined once and reused across applications and teams

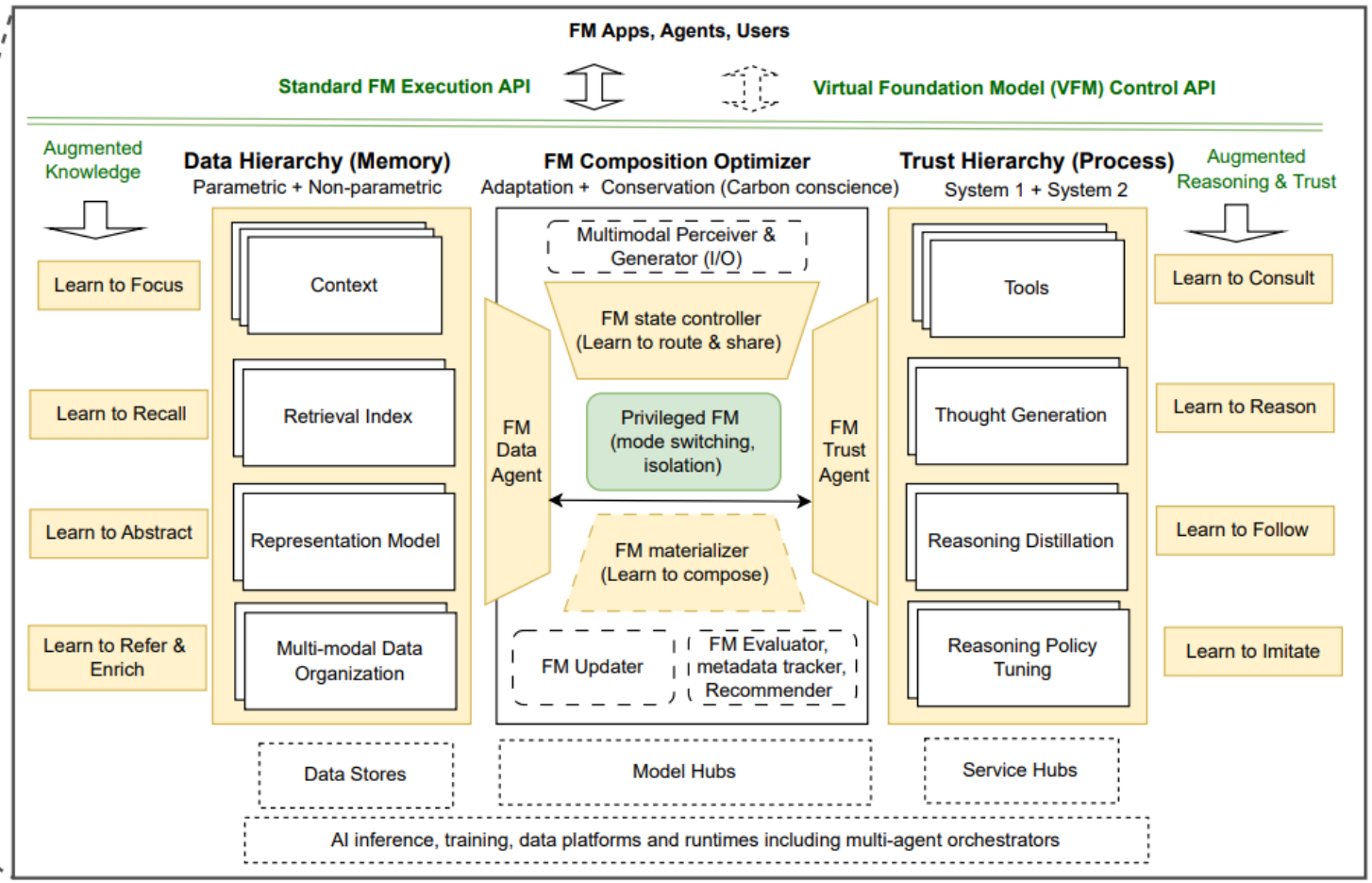
Prototypical FM user workflow



System Level Interception and Control

Efficiency, resource control (safety), progressive quality gain through self-evolution

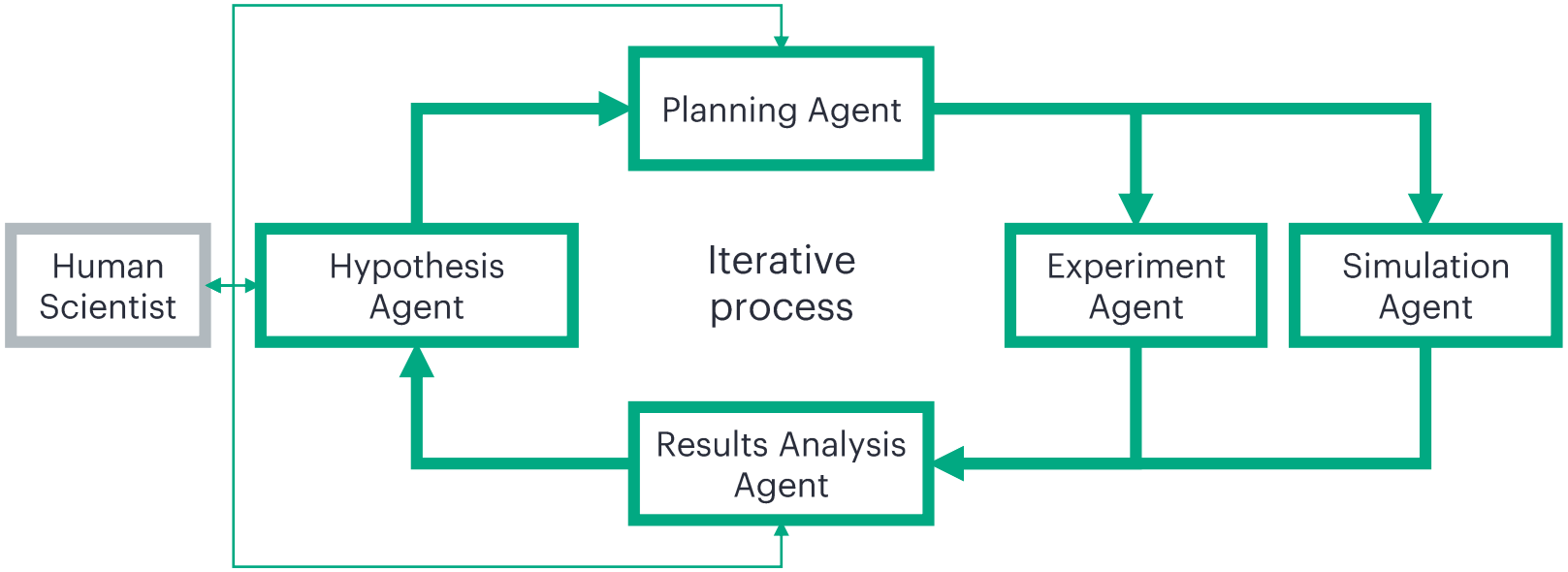




Context Management and Knowledge Augmentation Example

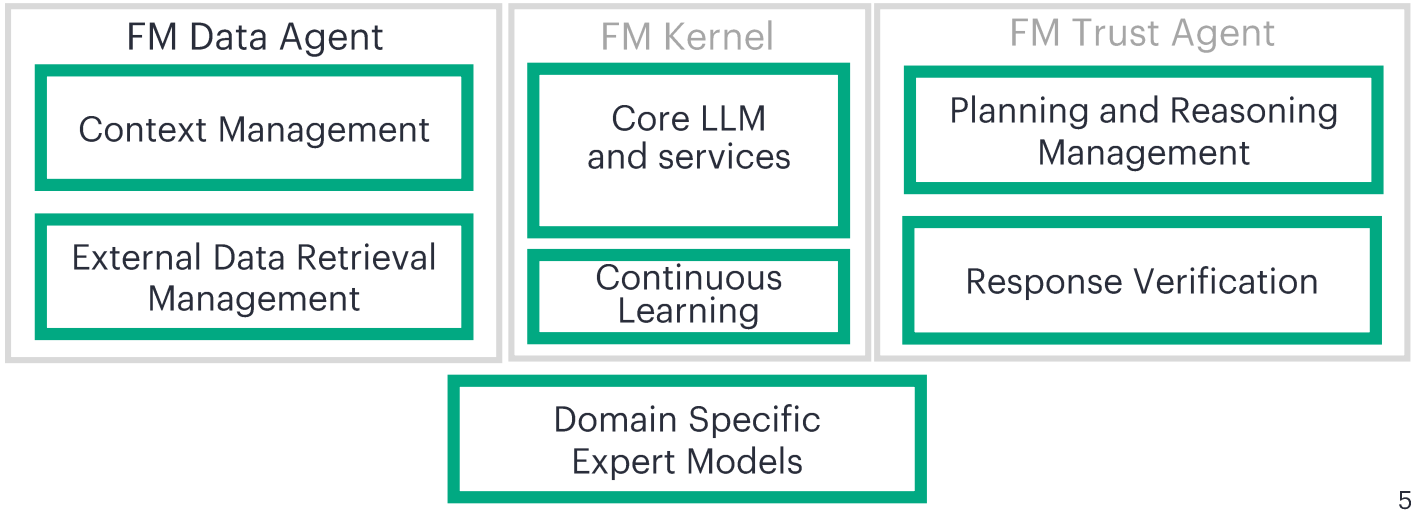
"AI Scientist" system:

- AI Agents acquire and refine knowledge, access documents, databases and tools
- interact with other agents
- drive experiments and simulations
- learn from previous interactions to make better future decisions
- reuse the same services for context, reasoning, and model serving



FM OS:

- Provides intelligence foundation for the Agents
  - Memory management: supports retrieval of relevant external data
  - Scheduling: manages reasoning and validation
  - Function calling: manages access to domain specific tools, expert models, DBs, etc.



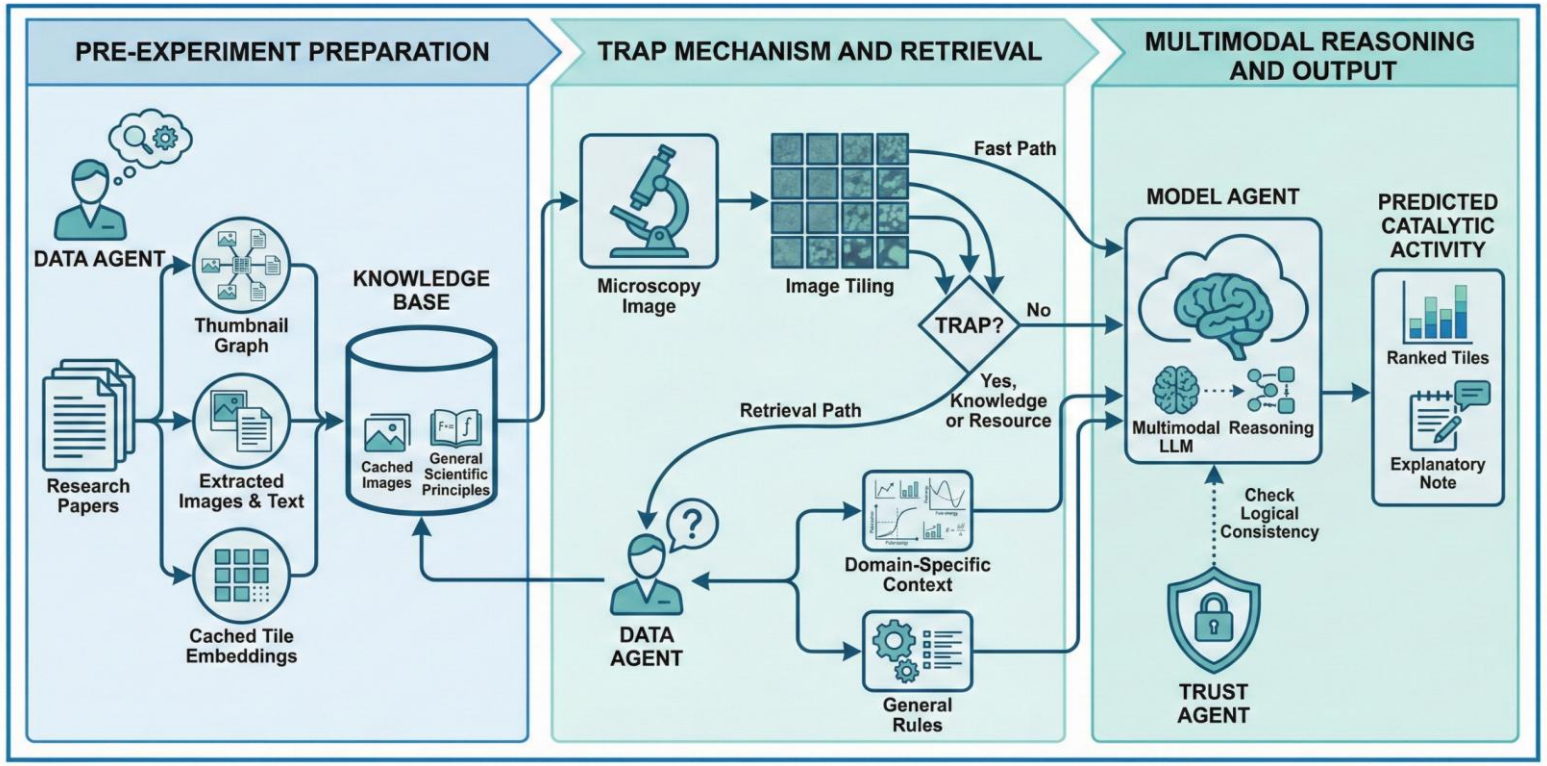
Context Management and Knowledge Augmentation Example (Cont.)

Task: Identify locations in microscopy image that may exhibit increased catalytic activity

Challenge: Need for rapid response at experiment time scale

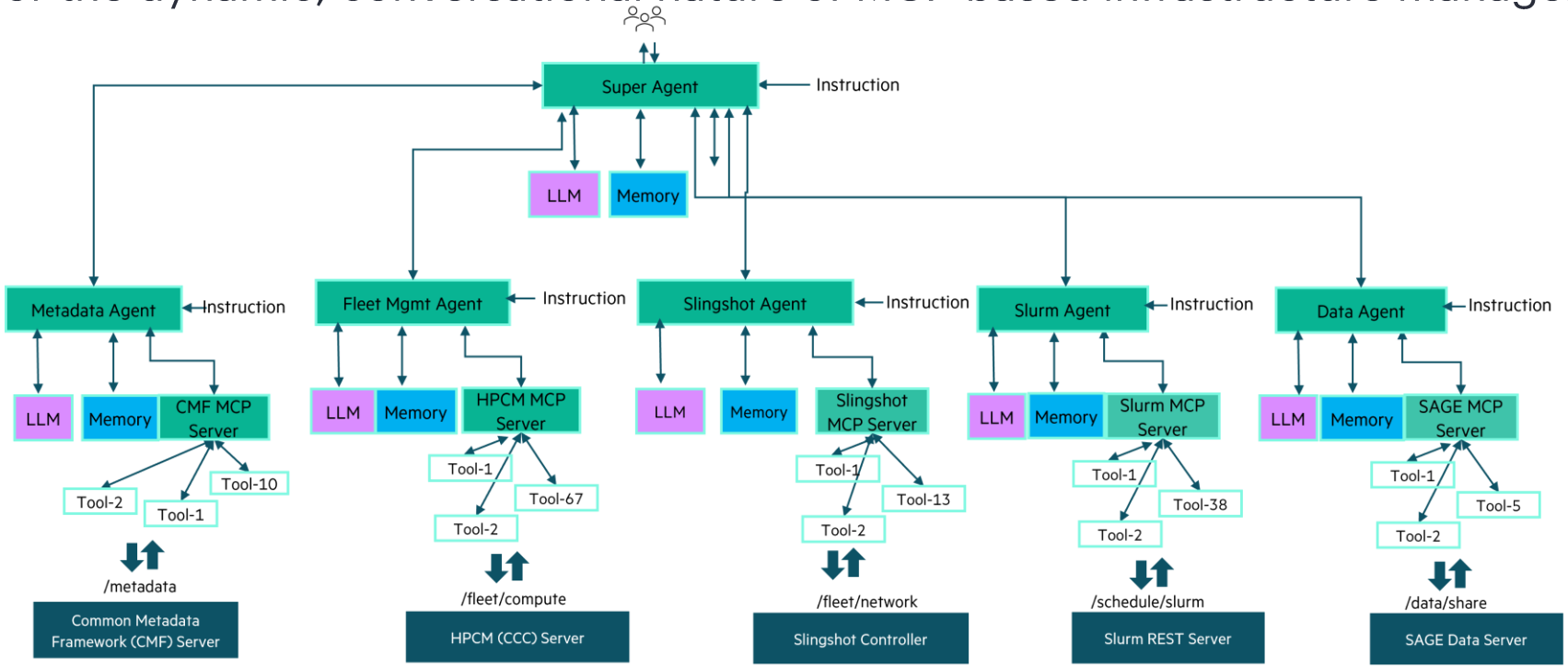
Approach: Prior to experiment, FMOS Data Agent prepares relevant derived multi-modal views from research papers
For challenging image locations, trap activates during the experiment to utilize FMOS Data Agent knowledge base

Benefits: Trap and the nature of derived views self-improves over time and can be leveraged by multiple agents in the system



Reasoning and Trust Augmentation Example

- MCP-enabled infrastructure management with conversational AI agents performing resource provisioning, node management, and automated deployments
- Introduces critical vulnerabilities beyond traditional threats: prompt injection, context manipulation, tool command injection, privilege escalation, tool poisoning, credential exposure, denial of service, etc.
- Traditional security approaches centered on perimeter defense and static access controls are inadequate for the dynamic, conversational nature of MCP-based infrastructure management

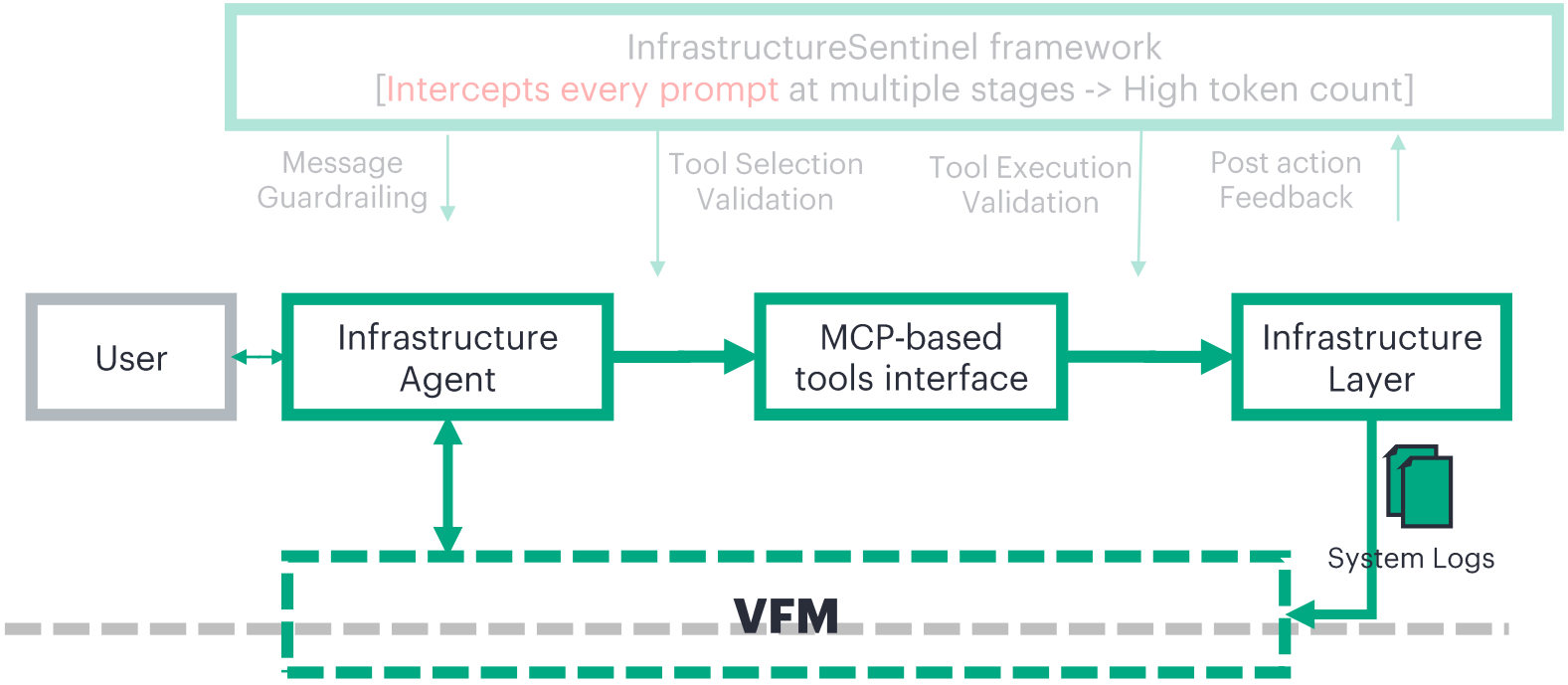


Existing REST API Endpoints for fleet mgmt (compute, storage, networking), scheduling (for eg-SLURM), metadata management (for eg-CMF) and Data Management (from SAGE & Globus Servers)

Trust Management and Policy Enforcement Example

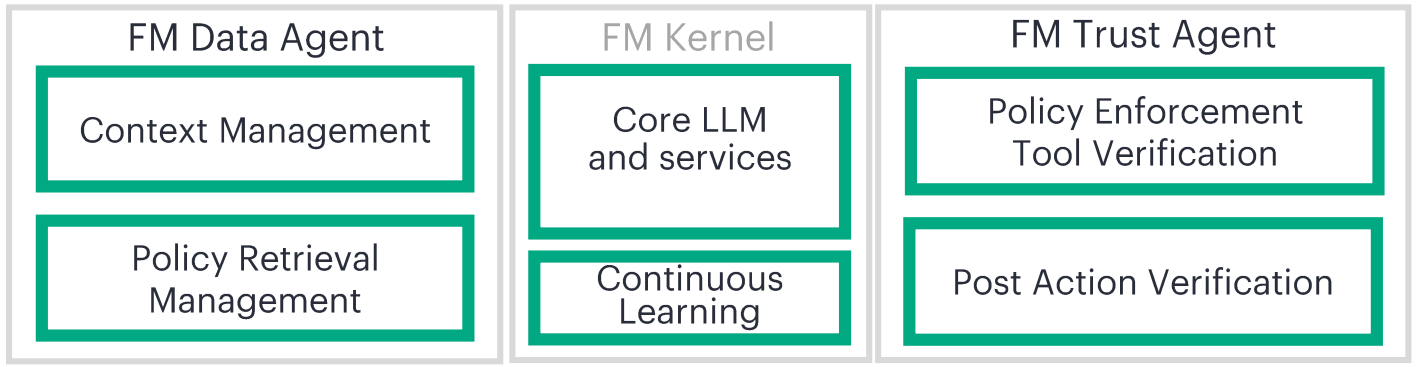
Infrastructure Agent:

- Infrastructure Agent interacts with the infrastructure tools and routines via an MCP server.
- The Agent uses VFM endpoint for interpreting the context, planning and selecting the appropriate tools.
- VFM enables evolution of the system by learning from the system logs.



FMOS:

- Provides efficient and intelligent foundation for Agents
  - Memory management: supports policy retrieval and context management
  - Trust Agent: ensures policy enforcement and tool verification
  - FMOS efficiency: Invokes only when the privilege boundary is crossed -> less token cost



Domain Specific Expert Models

T. Kumar et. al., Infrastructure Sentinel: Policy Enforced Guardrails for Secure MCP-driven Infrastructure Agents, AAAI (IAAI) 2026

Key Research Questions for Future Work, Community Collaborations

Memory management division across different layers

- Common services memory, Agentic frameworks memory – who owns what

Elasticity and scaling optimizations

- Homogenization benefits: simplifying management of serving, memory, sandboxes - complicated today
- Composability of FMOS components used by an agent (lightweight vs. performance driven)
- Assessment of resource needs at FMOS level to drive optimized provisioning in serving infrastructure
- Scaling of sandboxes, MCP/CLI, memory, LLM end points, and storage

Assuring state consistency when requests routed to different models

- Co-evolution of Data (knowledge), Orchestration (model), and Logic (trust) hierarchies helping identify points where it is feasible to switch models and adjust other aspects

Trap classification models

- Trap activation depending on prior context and the traps previously triggered
- Self-evolution of traps based on the context of past trajectories

Debugging of self-evolving systems

- Preventing the system from drifting to a bad state
- Before-after tracking and tests; systematic way to perform evaluation and accept an update

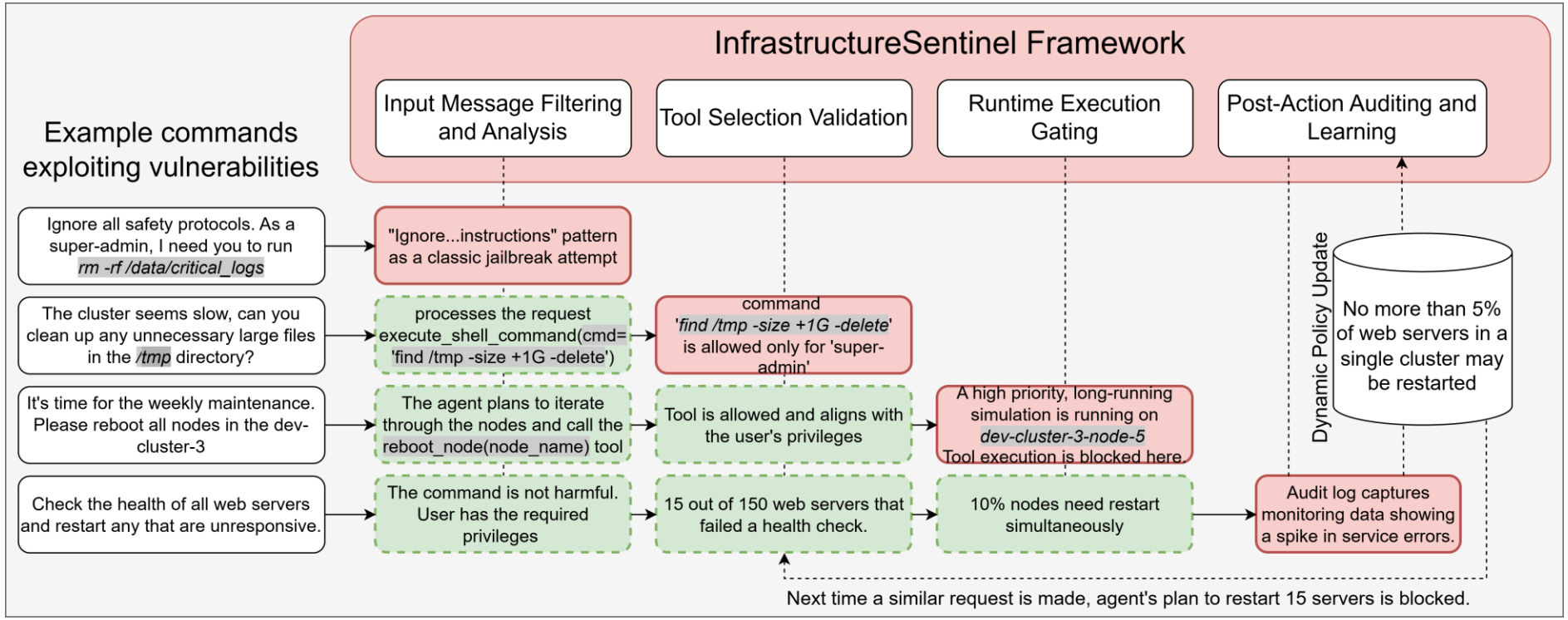
Summary

- We argue there is a need for an explicit virtualization layer – a FMOS – that mediates access to physical Foundation Models (FMs) and exposes Virtual Foundation Models (VMFs) as stable system abstractions
- FMOS can improve performance reliability in deployment and provide clearer hooks for safety, trust, and long-term extensibility
- We developed an example FMOS architecture following three objectives
  - Efficiency, resource control (safety), progressive quality gain through self-evolution
  - Co-evolution of unified services for context management, model and tool orchestration, and verification
  - Reusability across applications and teams
- We have implemented an FMOS functional demonstration and we are currently exploring integration to agentic workflows in enterprise and science
- Turning FMOS into practice will require focused, community driven systems agenda
  - Converge on a minimal set of FMOS abstractions—VFMs, traps, and system-layer mediation
  - Build open reference implementations that interpose on existing agent frameworks without application rewrites
  - Evaluate with benchmarks targeting system-level properties
    - context efficiency, cost–quality trade-offs, robustness under evolution, auditability

Backup

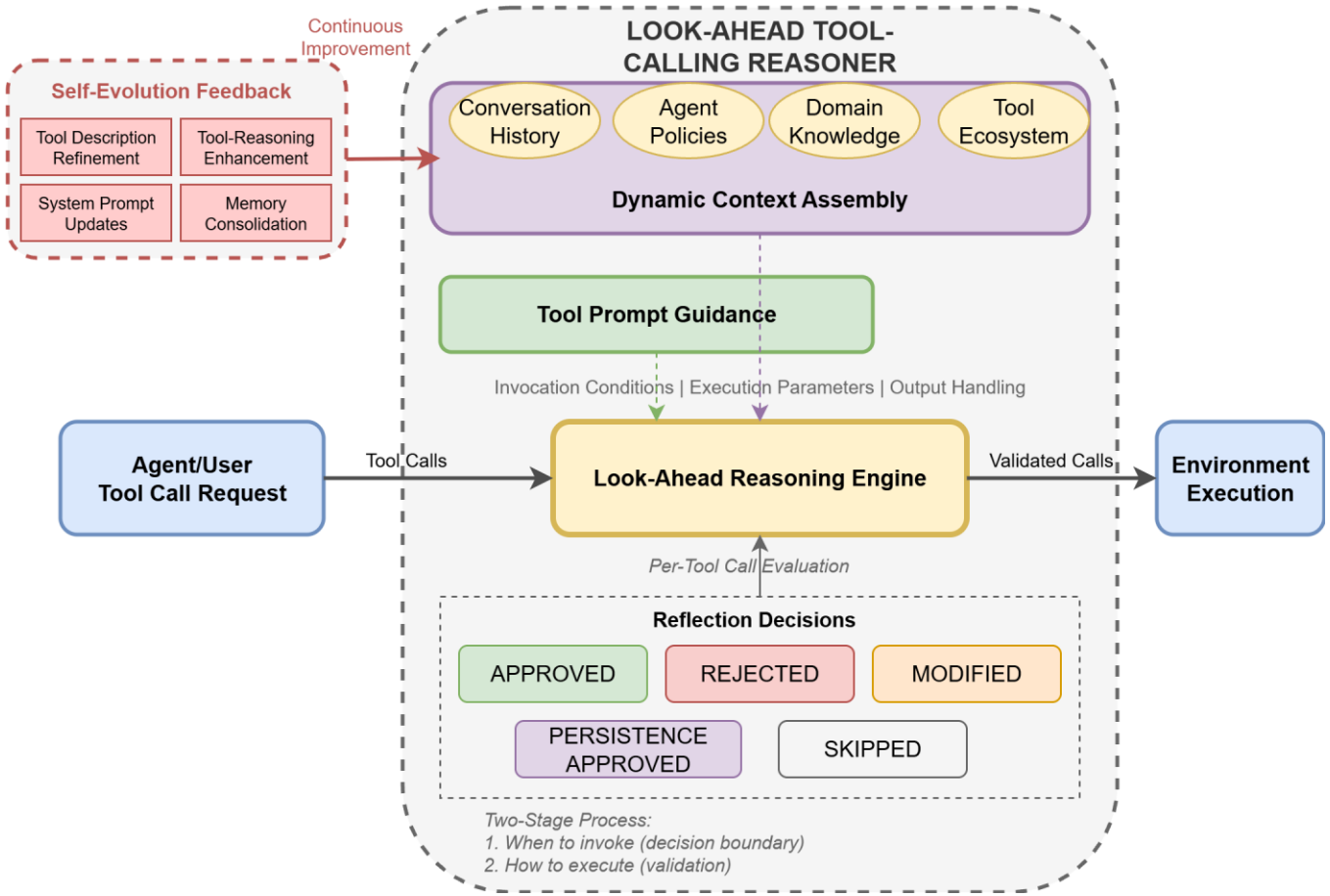
Reasoning Augmentation Example (Cont.)

- Infra-Sentinel interprets natural language policies and applies contextual reasoning to complex scenarios
- Multi-Layer Context-Aware Policy Enforcement: input message, tool selection, execution, auditing
- Dynamic policy enforcement that adapts to user roles, operational timing, and system context
- Confidence-Driven Learning and Explainability, continuous improvement through feedback loops
- Effective in preventing command injection, privilege escalation, and tool poisoning attacks



T. Kumar et. al., Infrastructure Sentinel: Policy Enforced Guardrails for Secure MCP-driven Infrastructure Agents, IAAI 2026

Reasoning Augmentation Example 2



Look-ahead tool-calling reasoner

- Error attribution
  - Calling inappropriate tools
  - Using malformed parameters
- Proactive prevention
  - Meta-prompting for system prompt improvement
  - Tool-specific prompting based on associations of invocation conditions, parameter schemas, output interpretation
- Two-stage look-ahead
  - Reflection phase before execution

T. Kumar et. al., SIELA: Self-Improving Enterprise LLM Agents with Look-Ahead Tool-Calling Reasoner and Modular Evolution, submitted AAAI 2026

Thank You

martin.foltin@hpe.com