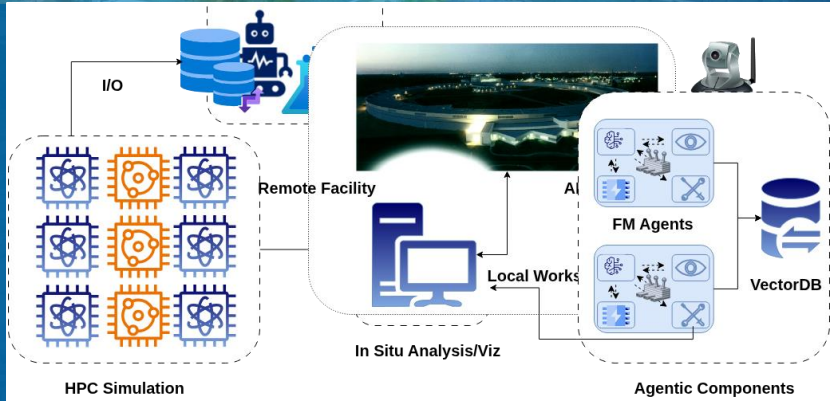
FEB 18, 2026

STABILITY IN MOTION: PERFORMANCE CHARACTERIZATION, RESILIENCE, AND TRUSTWORTHINESS OF CONTEMPORARY WORKFLOWS



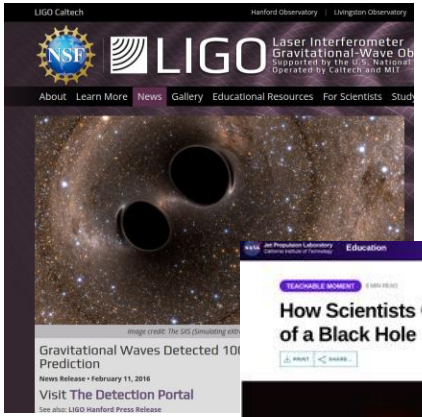
AMAL GUEROUDJI
Agueroudji@anl.gov
Postdoctoral Researcher
Argonne National Lab

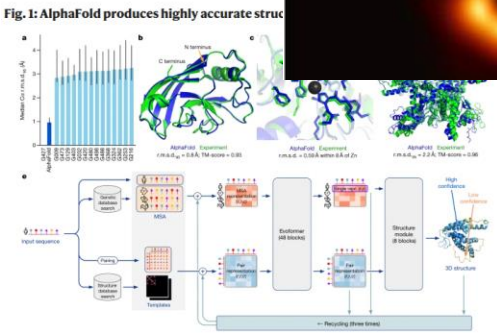
U.S. DEPARTMENT of ENERGY
Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.
Argonne NATIONAL LABORATORY

COMPUTATIONAL WORKFLOWS ARE THE BACKBONE OF MODERN DISCOVERY

Scientific computational workflows are powerful engines of discovery. They coordinate simulations, data movement, ML trainings, inferences, and real-time experimental streams at unprecedented scale.

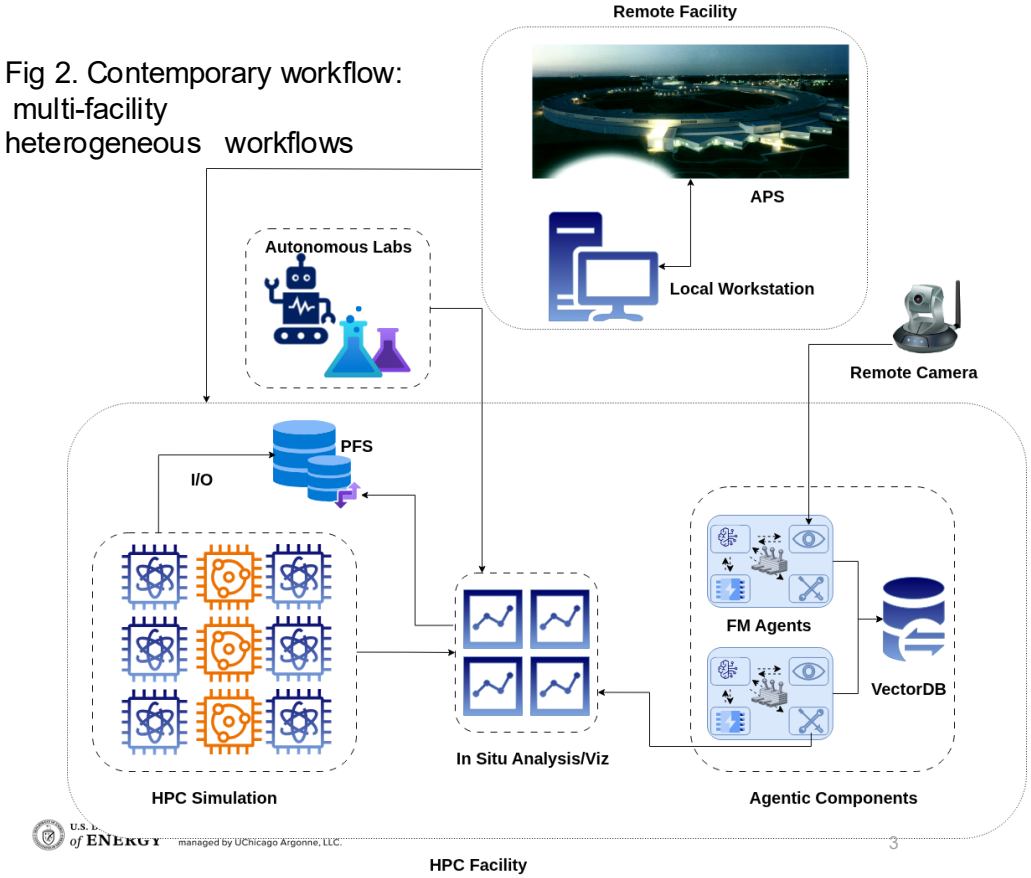
- The First Detection of Gravitational Waves (LIGO, 2015), 2017 Nobel Prize in Physics
- The First Image of a Black Hole (Event Horizon Telescope, 2019)
- AlphaFold2 & Protein Structure Explosion (2021–present), 2024 Nobel Prize in Chemistry
- ...





CONTEMPORARY WORKFLOWS: HIGH-DIMENSIONAL HETEROGENEITY

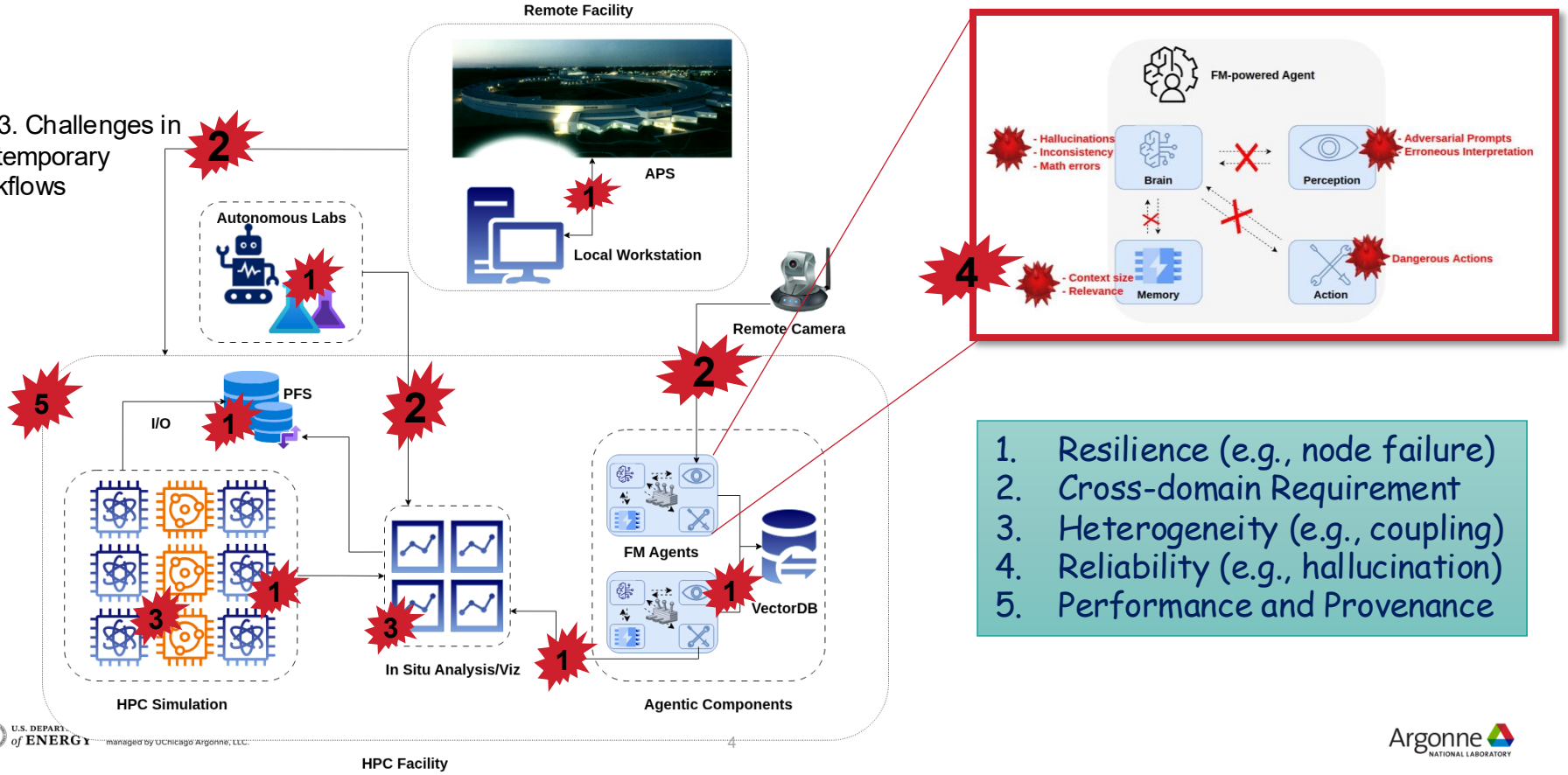
Fig 2. Contemporary workflow: multi-facility heterogeneous workflows



- **Data**: Large, Different sources, Different formats, Irregular
- **Communication**: e.g., MPI, RPC, Globus …
- **Programming models**: e.g., MPI, Task-based, CUDA, OpenMP ...
- **Orchestration**: e.g., Dynamic, Autonomous
- **Site**: e.g., Multi-facility, IoT

CONTEMPORARY WORKFLOWS' CHALLENGES

Fig 3. Challenges in contemporary workflows

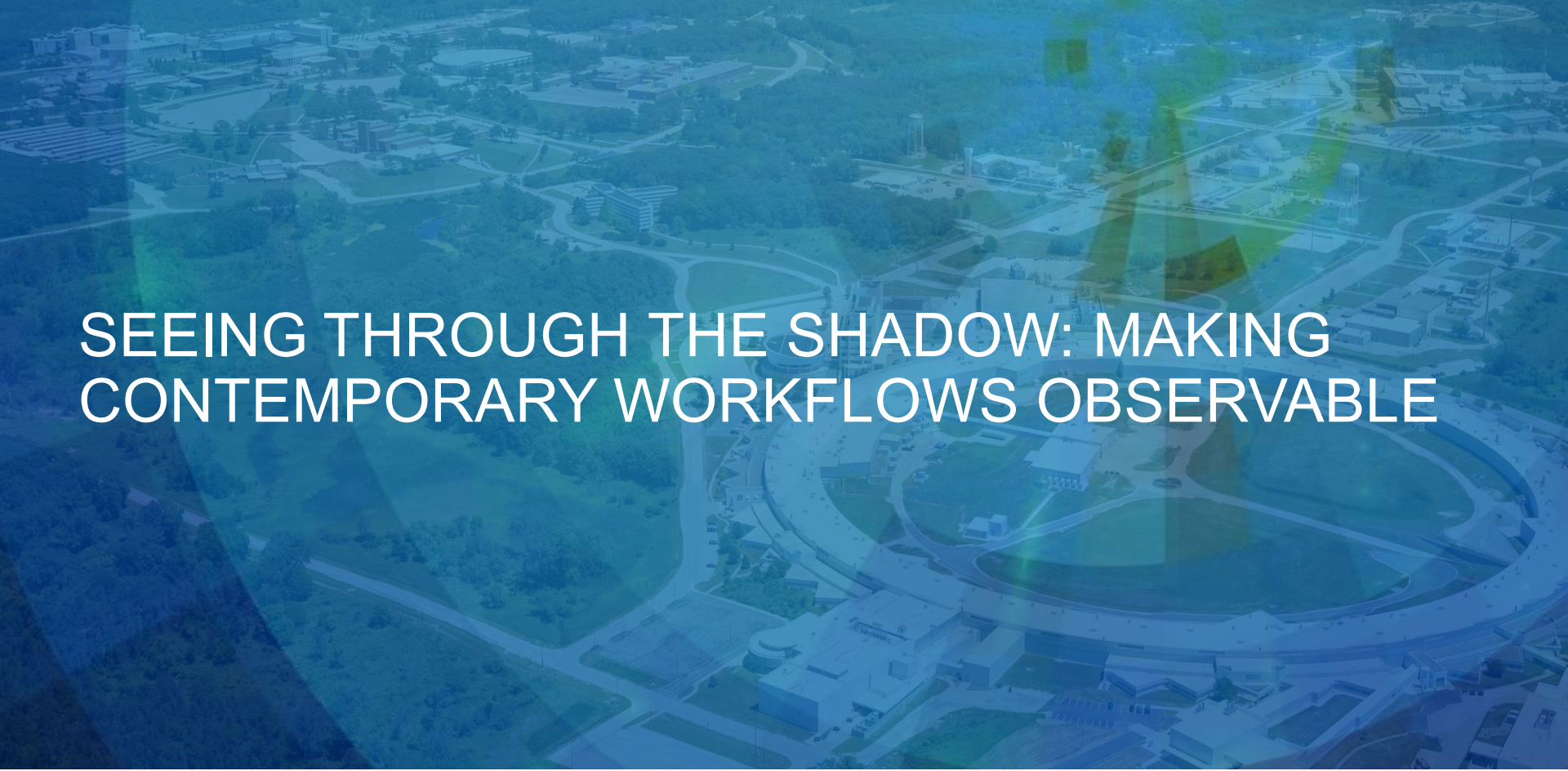


1. Resilience (e.g., node failure)
2. Cross-domain Requirement
3. Heterogeneity (e.g., coupling)
4. Reliability (e.g., hallucination)
5. Performance and Provenance

CONTEMPORARY WORKFLOWS' CHALLENGES

Contemporary workflows sit in the shadow of complexity and challenges: scale, heterogeneity, unpredictability.
At Argonne, my work builds the systems foundations to turn these challenges into a new generation of **trustworthy, resilient, *characterizable*** scientific instruments, by:

1. Seeking understandability in a world that is extremely hard to characterize
2. Offering resilience as a built-in feature
3. Enabling reliability (trustworthiness) in creative (a.k.a. agentic) systems



SEEING THROUGH THE SHADOW: MAKING CONTEMPORARY WORKFLOWS OBSERVABLE

U.S. DEPARTMENT of ENERGY

Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.

Argonne NATIONAL LABORATORY

WE CANNOT IMPROVE WHAT WE CANNOT MEASURE!

But why is it hard to characterize contemporary workflows?

- A landscape defined by heterogeneity
  - Different architectures, languages, programming models, scheduling
  - Different profiles, counters, traces and behavior
- An overabundance of characterization tools, each offers only a partial view of the behavior
  - System level, e.g., psutil, perf
  - I/O and storage, e.g., Darshan
  - Network and communication profilers, e.g., mpiP
  - Runtime-level telemetry, e.g., Dask dashboard
  - Facility Tools, e.g., vendor profilers, node monitors, job-level analytics
- Correlating signals/records/profiles across the stack is extremely hard:
  - Different formats
  - Different concepts
  - Different abstractions
  - No a priori knowledge about execution path (e.g., in dynamic runtimes)
  - …

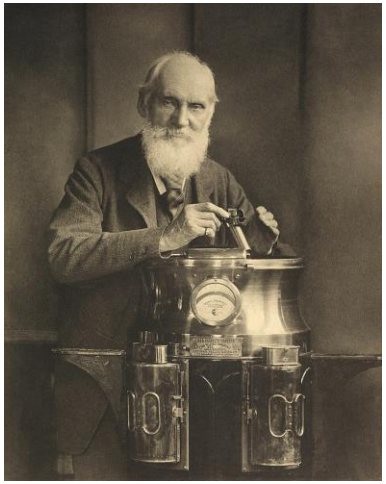


Fig 4. If you can't measure it, you can't improve it. Lord Kelvin (1824–1907) [Figure from Wikipedia]

A characterization gap: workflows behave **holistically**, but our tools observe them in **shards**.

A LAYERED OBSERVABILITY AND PROVENANCE STACK FOR DISTRIBUTED WORKFLOWS

- Performance and provenance data need to be **collected** at multiple levels:
  - **Workflow management level**:
    - tasks behavior: creation, state transition, inputs, outputs, timestamps,
    - scheduling,
    - work stealing,
    - communications ...
  - **Low-level**: system I/Os per workflow --> Node --> process --> (maybe per thread?), CPU/GPU metrics ...
  - **Job level**: configurations, allocations, logs …
- Data needs to be in an **easily searchable format**

1. Capture WMS data (workflow, task, scheduling ...)
2. Write/stream the data to a unified format

HPC profilers to capture system and I/O data

Parsing the files

Tabular format?

Gueroudji A. et al, "Performance Characterization and Provenance of Distributed Task-based Workflows on HPC Platforms", WORKS, SC24

DARSHAN: HPC I/O CHARACTERIZATION TOOL

Darshan usage in task-based workflows

- Darshan instrumentation captures coarse-grained I/O timing info (start/end timestamps for open, read, write, close operations)
- Darshan eXtended Tracing (DXT) provides full MPI-IO and POSIX read/write API.
- The traditional DXT is **not enough** to characterize I/O in task based models, **because a process may run multiple tasks in different threads simultaneously**
- There **is no way to map** the Dask tasks to the underlying I/O records collected by Darshan (N --> M mapping)

=> Add Pthread IDs to Darshan records in addition to timestamps and Process IDs

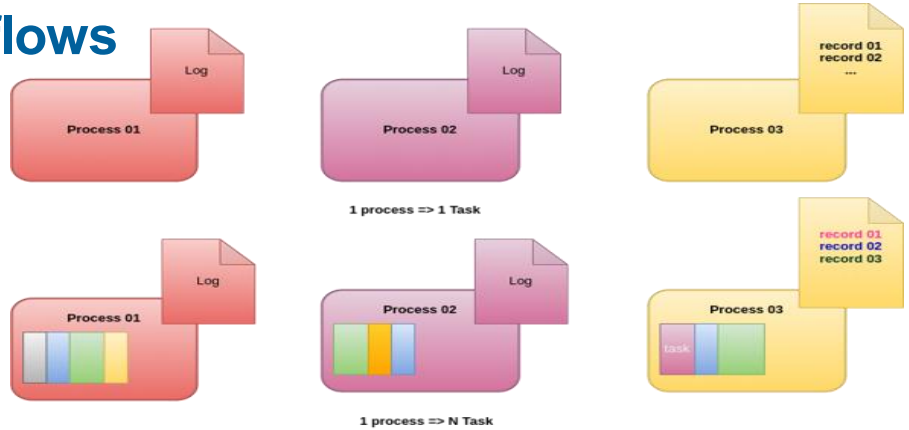


Fig 5. In a typical HPC job (top) each process executes essentially one “task” (i.e., its portion of a simulation). In Dask workflows (bottom), long-running worker processes execute a **variety of tasks** as the workflow unfolds. With the existing Darshan implementation, the I/O of all the tasks executed by a **single worker** is bundled into a **single log**, obfuscating useful connections between tasks and associated I/O.

Gueroudji A. et al "Performance Characterization and Provenance of Distributed Task-based Workflows on HPC Platforms", WORKS, SC24

EXAMPLE OF DASK DISTRIBUTED WORKFLOW CHARACTERISTICS

- Experiments in ALCF Polaris (560 nodes. Each node has one 2.8 GHz AMD EPYC Milan 7543P 32-core CPU with 512 GB of DDR4 RAM)
- 3 workflows (Image processing pipeline, fine-tuned ResNet152 batch prediction, XGBOOST regression model training)
- 10 runs per workflow
- 2 worker nodes (4 workers per node)

Workflows	ImageProcessing	ResNet152	XGBOOST
Task graphs	3	1	74
Task category	25	5	17
Distinct tasks	5440	8645	10348
Distinct files	151	3929	61
No. of I/O	5274-5287	2057-2302	867-1670
No. of transfers	3141-3247	3751-3976	1464-2027

Tab 2. Workflows characteristics summary

No prior data about communications

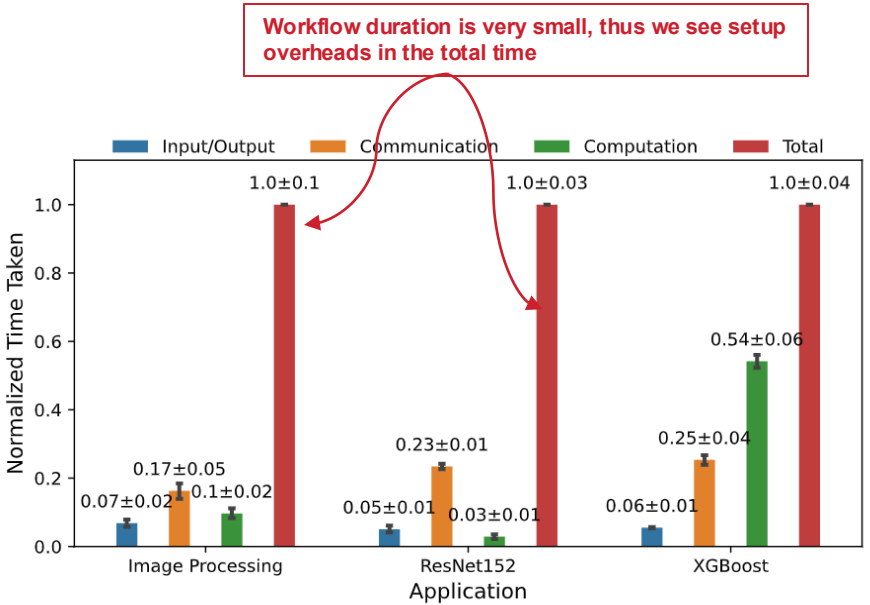


Fig 7. Normalized time spent per workflow in I/O, Communication, Computation, and total wall time.

EXAMPLE: CORRELATION BETWEEN TASK DURATION & WORKER RESPONSIVENESS

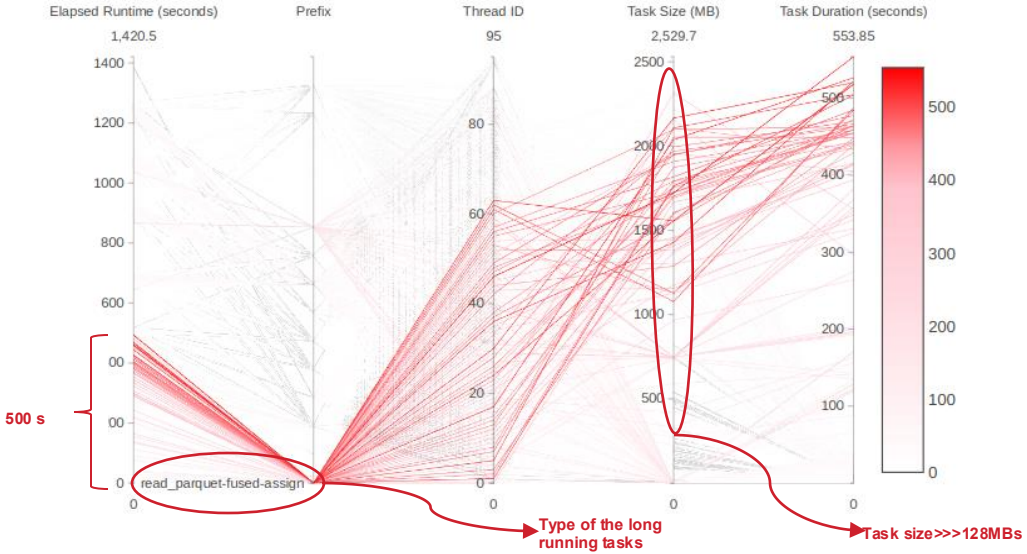


Fig 8. Task Characteristics parallel coordinate

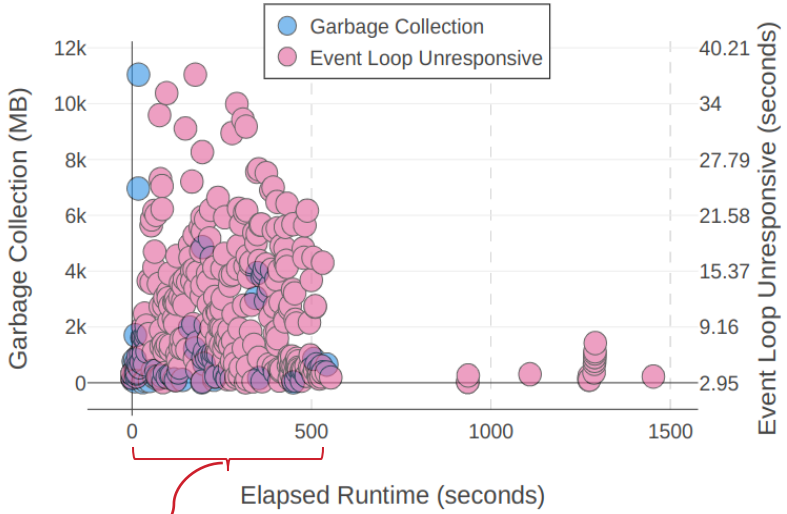


Fig 9. Distribution of scheduler warnings

Gueroudji A. et al "Performance Characterization and Provenance of Distributed Task-based Workflows on HPC Platforms"

MULTI-SOURCE TASK PROVENANCE & INTEGRATION WITH FLOWCEPT

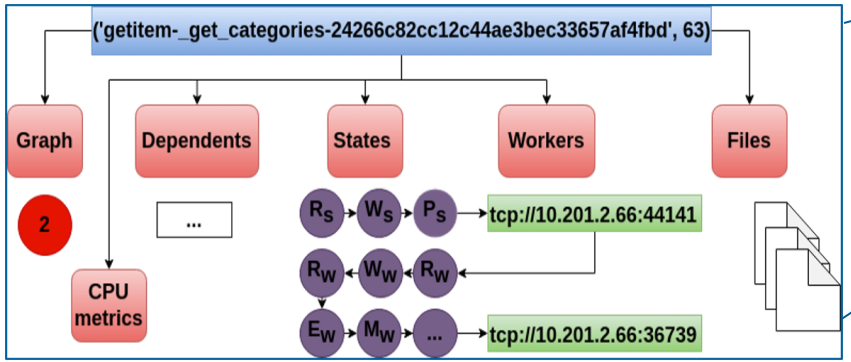


Fig 10. For each task, we collect data from multiple sources. We have a complete report about which graph the task is submitted from, the list of dependents and dependencies, all state transitions in the scheduler and workers, all data duplications/communications, CPU/GPU metrics, and all the I/Os performed by the task with underlying information (file path, PFS, hostname, I/O type and size, and timestamp).

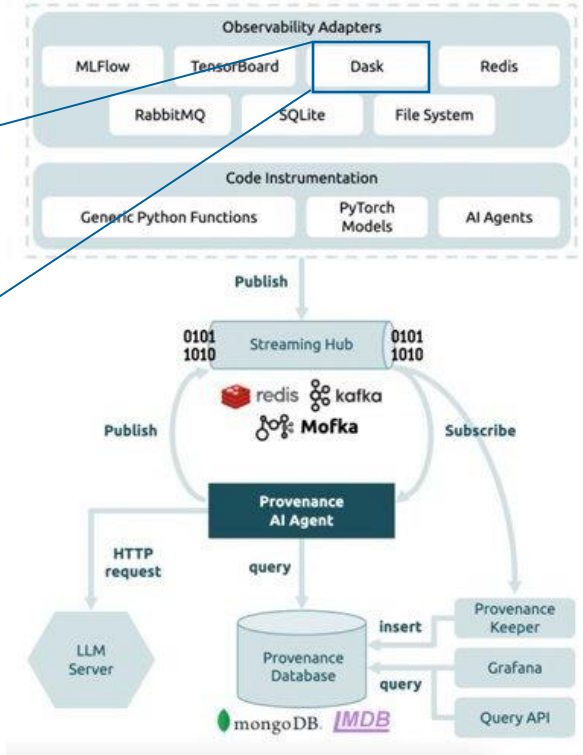


Fig 11. Flowcept Architecture

Flowcept figure from Souza R. et al, "LLM Agents for Interactive Workflow Provenance: Reference Architecture and Evaluation Methodology", WORKS, SC25

RESILIENCE AS A BUILT-IN FEATURE

PERSISTENT STREAMING FOR TRADITIONAL FAULTS, HETEROGENEITY

Why Resilience Matters?

Modern experiment-driven workflows, such as those used at large scientific facilities, operate under extreme constraints:

- Failures lead to lost data => delayed science (e.g., beamline time need proposal writing and a wait time in the order of months)
- Traditional pipelines (e.g., coupled with ZeroMQ) are:
  - tightly coupled
  - fragile under load
  - unable to buffer or recover
  - prone to cascading failures

Our goal is to turn resilience into a **built-in feature**, not an afterthought, so workflows behave like robust scientific instruments rather than fragile pipelines.

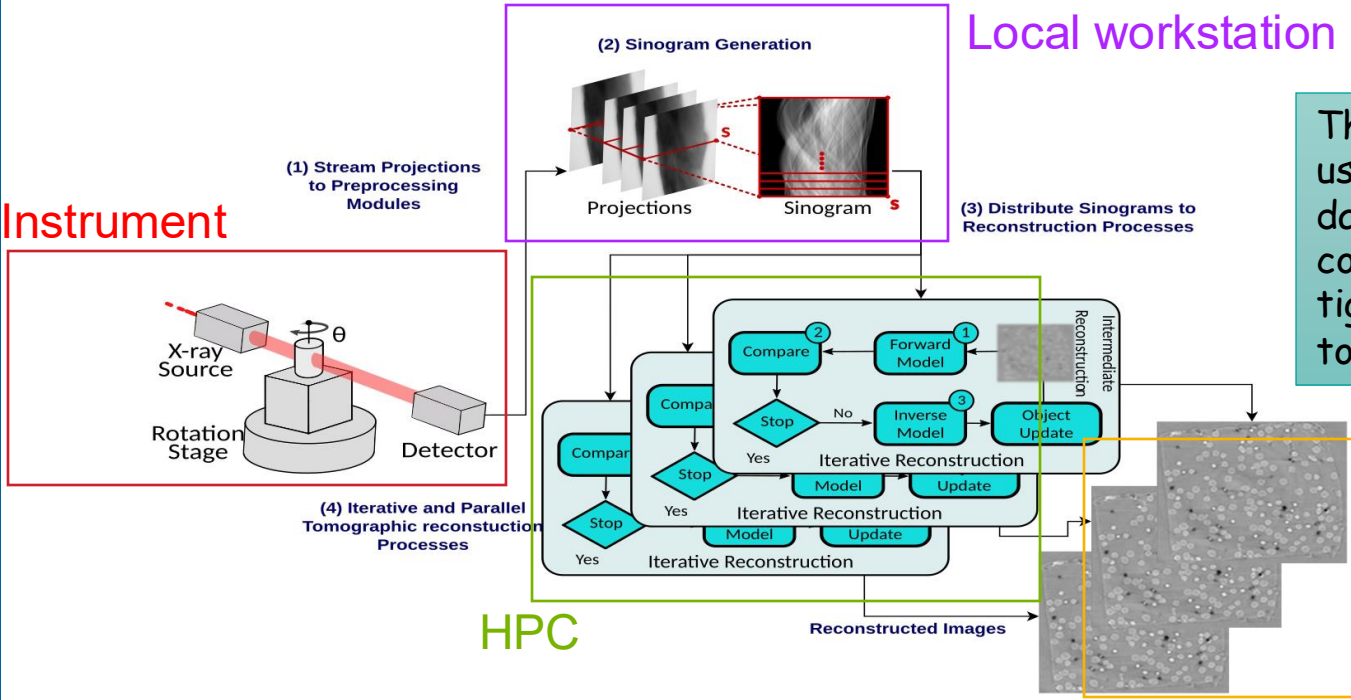
WHAT IS PERSISTENT STREAMING?

Persistent streaming is a data movement and communication model in which data flows continuously and durably between the components of a workflow.

- **Continuous flow**: Data is produced, transported, and consumed in *real time* or *near-real time* (no batch barriers).
- **Durability**: Each message or event is logged *persistently* (to a storage tier or log) so it can be replayed or recovered.
- **Decoupling**: Producers and consumers operate *independently* in time and scale. If one fails or slows down, the system still progresses.
- **Elasticity**: Components can join or leave the stream *dynamically* (ideal for scaling compute tiers or adapting to load).
- **Fault tolerance**: Because the stream state is persistent, a crashed component can restart and *resume without losing* context.

CASE OF STUDY

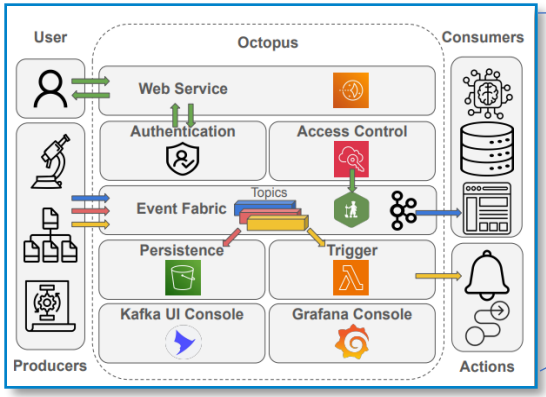
Tomographic Reconstruction Workflows



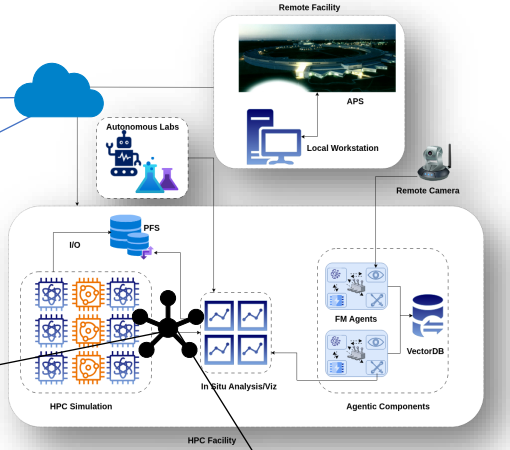
The original implementation uses ZeroMQ to send data between the workflow components (synchronous, tightly coupled, prone to failure)

Fig 13. Original tomographic reconstruction pipeline as described by Nguyen et al. 2025

PERSISTENT STREAMING SOLUTIONS (DIASPORA PROJECT, I. FOSTER)



Octopus



Mofka

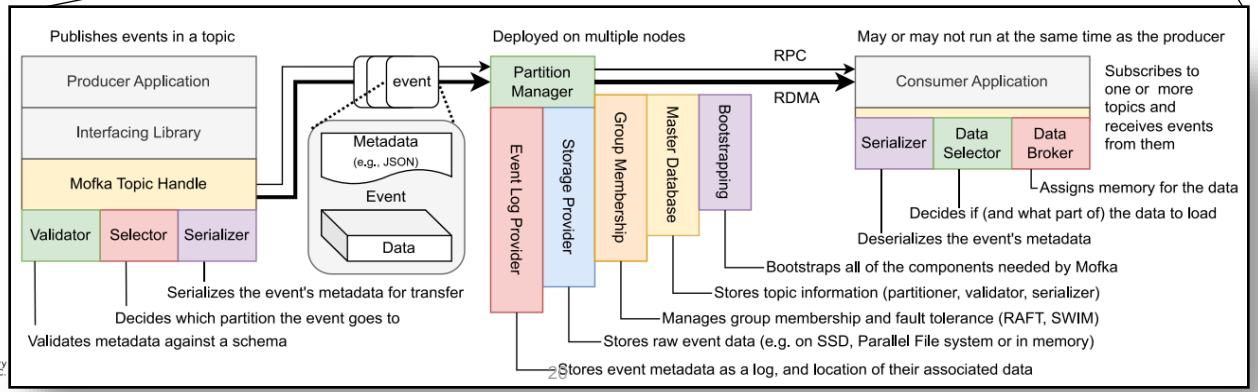


Fig 14. Persistent streaming solutions: Mofka for intra-facility and Octopus for inter-facility communication

MOFKA: HPC-TAILORED STREAMING SERVICE

Work led by Matthieu Dorier

- Mochi is a state-of-the-art open-source tool for **rapid** development of customized **data services for HPC**, big data, and large-scale learning
- Mofka is a **persistent streaming** service for HPC applications, built on mochi components. Analogous to Kafka but tailored for scientific computing.
- It leverages **HPC architectural features,** such as high-performance networks, RDMA, high-performance local NVMe storage devices, and high-concurrency multicore CPUs.
- Mofka **events** contain **large raw Data** and **small structured Metadata**. They are organized into Topics.
- Producers and Consumers **subscribe asynchronously** to the Topics they are interested in.

MOFKA ARCHITECTURE

Work led by Matthieu Dorier

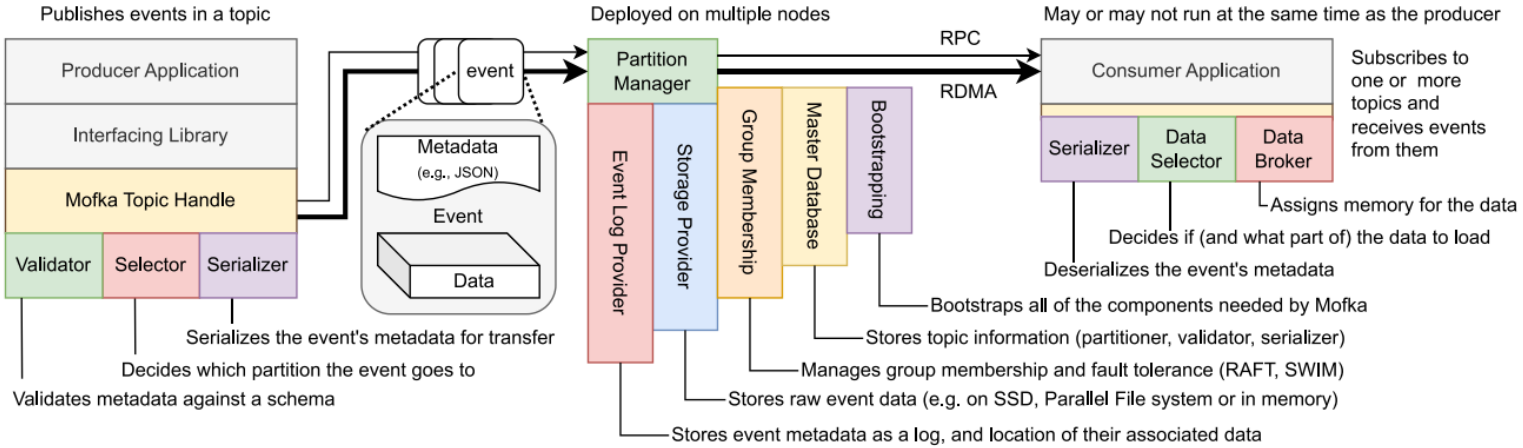


Fig 15. Overview of the Mofka architecture, including client libraries (producer and consumer). Mofka is an assemblage of Mochi components handling distinct aspects of the service, from bootstrapping, to storing the metadata and data that compose events, to managing group membership and global information

Figure from Dorier M. et al., "Toward a persistent event-streaming system for high-performance computing applications", Frontier 2025

RESILIO: MOFKA-ENABLED ARCHITECTURE

- DAQ: Data Aquisition
- DIST: Preprocessing
- SIRT: Simultaneous Iterative Reconstruction Technique
- DEN: Denoiser

- Fully decoupled
- Extra-resilience layer
- Elastic and scalable by design (inherited from Mochi)

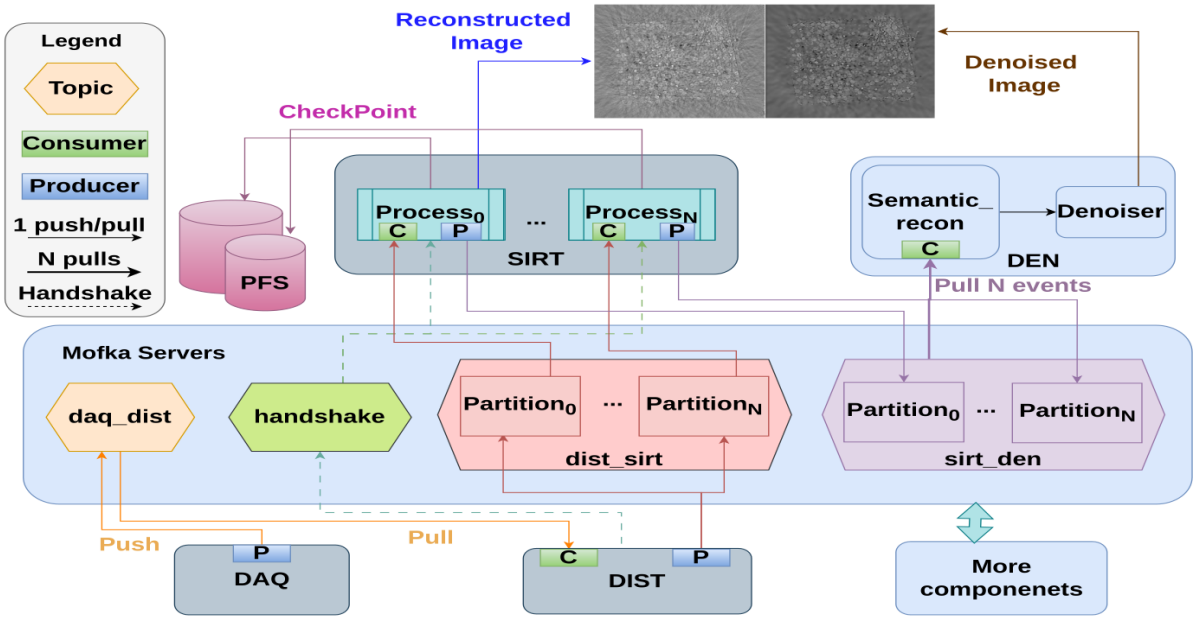


Fig 16. Mofka-Enabled architecture. The four components are represented: DAQ, DIST(bottom), SIRT, and DEN (upper part). The Mofka server, the topics, and different partitions are shown in the middle layer

Figure from Gueroudji A. et al., "RESILIO : A Scalable and Composable Architecture for Tomographic Reconstruction Workflows", WORKS, SC25

TOMOGRAPHIC WORKFLOW CHARACTERISTICS

Tab 2. Component configuration for DAQ, DIST, SIRT and DEN. The [P] corresponds to data producers and [C] to data consumers

	Component	Nodes	Processes	Partitions	Data	Metadata	Total events
	DAQ [P] Python	1	1	1	Proj 20 KB	30 B	1,500
	DIST [C] Python	1	1	1	20 KB	30 B	1,500
	DIST [P] Python	1	1	2–64	Sino 10 KB	JSON 148 B	3,000–96,000
MPI+OMP	SIRT [C] C++	1–32	2–64	2–64	10 KB	148 B	3,000–96,000
MPI+OMP	SIRT [P] C++	1–32	2–64	1 / 2–64	Recon 25 MB	290 B	188–6,016
TensorFlow	DEN [C] Python	1	1	1 / 2–64	25 MB	290 B	188–6,016
MPI	Mofka server C++	1	2–64	9–129	4.64–47.8 GB	0.518–15.25 MB	7,867–103,516

Sources of heterogeneity:

- Programming models
- Programming languages
- Data types
- Data sizes
- Scalability

These and the following results are from:

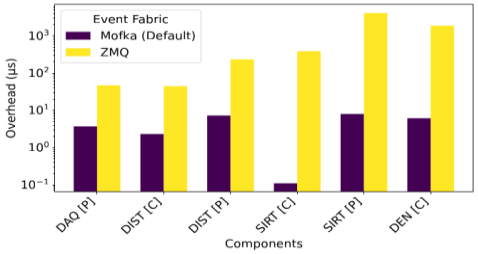
- Gueroudji A. et al, RESILIO: A Scalable and Composable Architecture for Tomographic Reconstruction Workflows
- Dorier M. et al, Toward a persistent event-streaming system for high-performance computing applications

MOFKA VS ZMQ PERFORMANCE

At least a 12x reduction in overhead across all components

Tab 3. Producer/consumer per-event overhead. Mofka vs ZeroMQ. The [P] corresponds to data producers and [C] to data consumers. Overhead measured in microseconds (Default asynchronous mode)

Operator	Mofka (Memory)				Mofka (Default)				ZMQ		
	Best Config	Min	Med	Max	Best Config	Min	Med	Max	Min	Med	Max
DAQ [P]	64	2.73	3.77	1.2e6	64	3.32	3.71	4.6e6	40.1	46.7	260
DIST [C]	64	2.00	2.25	8.84	16	1.99	2.31	8.9e4	10.2	44.5	1.7e5
DIST [P]	64	3.57	7.06	1.1e6	16	3.65	7.24	1.5e7	103.4	232.6	1.2e6
SIRT [C]	1	0.02	0.10	5.2e4	1	0.02	0.11	3.5e5	7.31	384	6.5e5
SIRT [P]	64	7.39	8.24	2.6e5	32	7.15	7.97	2.3e6	3,373	4,062	4,108
DEN [C]	8	4.07	5.92	2.7e8	64	4.34	6.13	1.5e9	1,186	1,865	6,865



Smaller than a complete RPC round trip in Polaris

Up to 500x better performance than ZeroMQ

PYTHON PRODUCER SCALABILITY EXAMPLE: DIST COMPONENET

Computed throughput for end-to-end send call (including persistence)

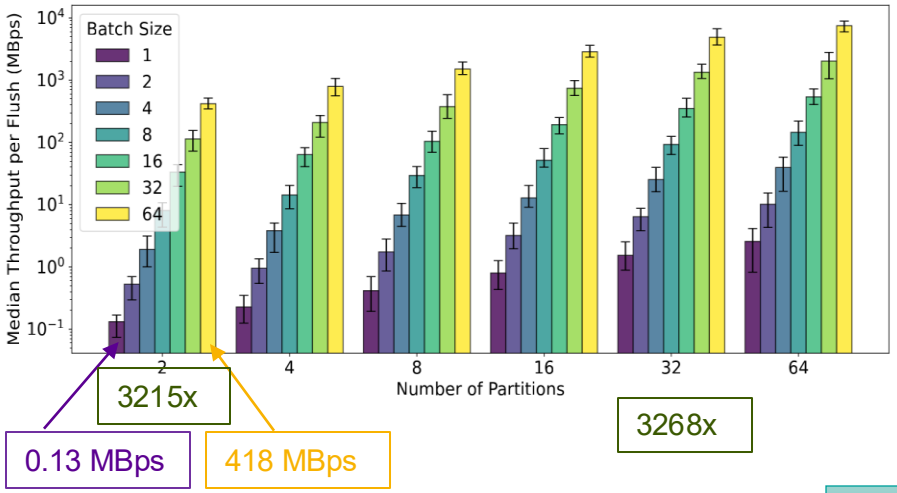


Fig 17. DIST Send throughput while varying the batch size and scaling the number of partitions from 2 to 64. The size of the event is fixed and the number of events increases with the number of partitions (Blocking operations)

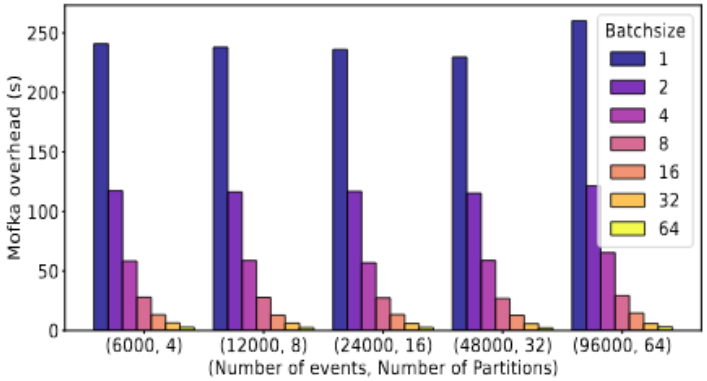
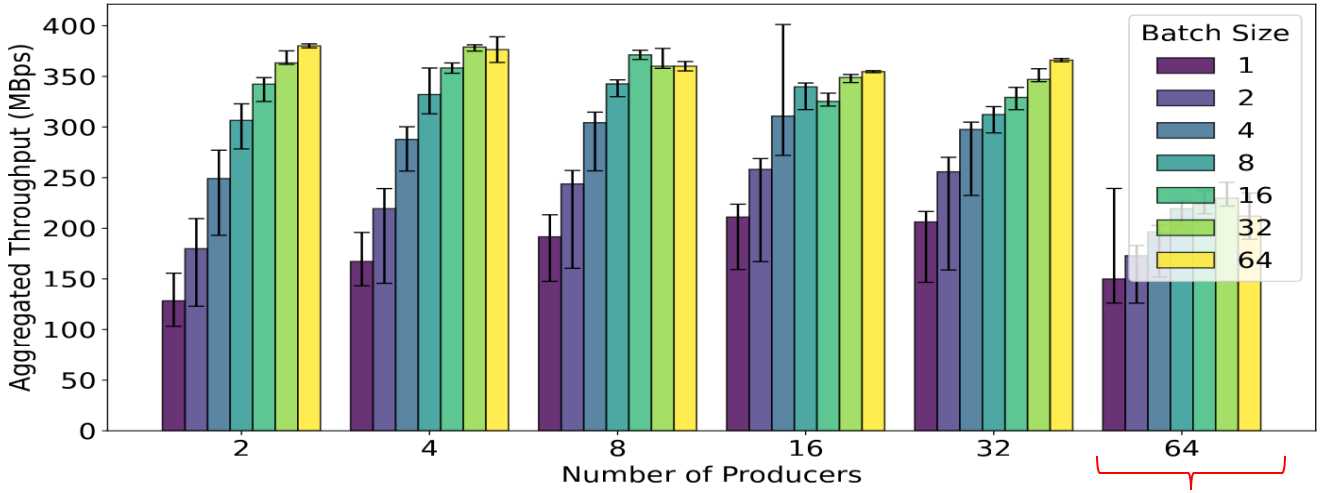


Fig 18. Overhead for non-blocking operations

- Using large batches for small events is beneficial
- Python API has a lot of overhead
- More partitions <=> More parallelism

C++/MPI PRODUCERS SCALABILITY EXAMPLE: SIRT COMPONENT

Computed throughput for end-to-end send call (including persistence)



- Less partitions <=> Less parallelism (1 partition is used here)
- Large events do not benefit from batching

Fig 19. SIRT producers' evaluation while varying the batch size and scaling the number of processes from 2 to 64. The size of the event is fixed, and the number of events increases with the number of processes (results for a single partition in the server)

C++/MPI PRODUCERS SCALABILITY EXAMPLE

End-to-end blocking vs non-blocking send overheads

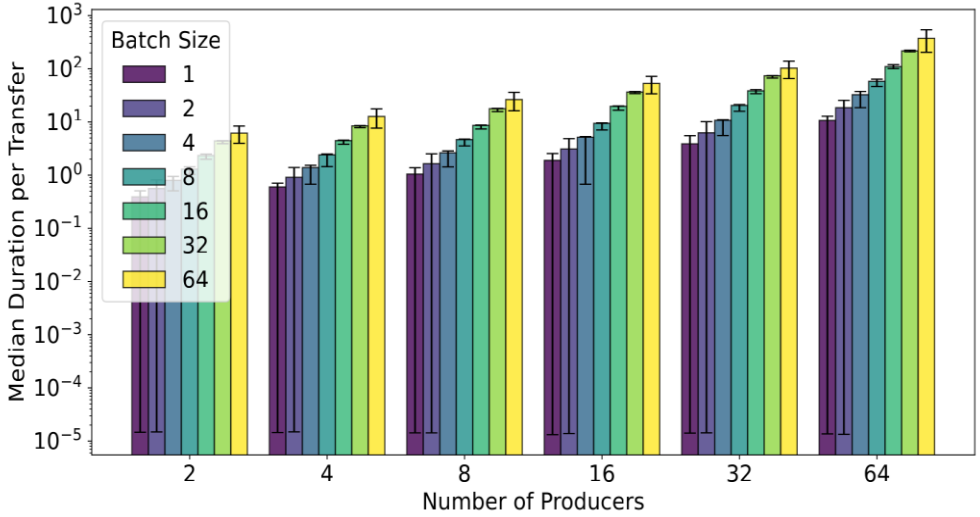


Fig 20. SIRT producer evaluation varying the batch size and the number of processes from 2 to 64. The size of the event is fixed, and the number of events increases with the number of processes (1 partition in the server)

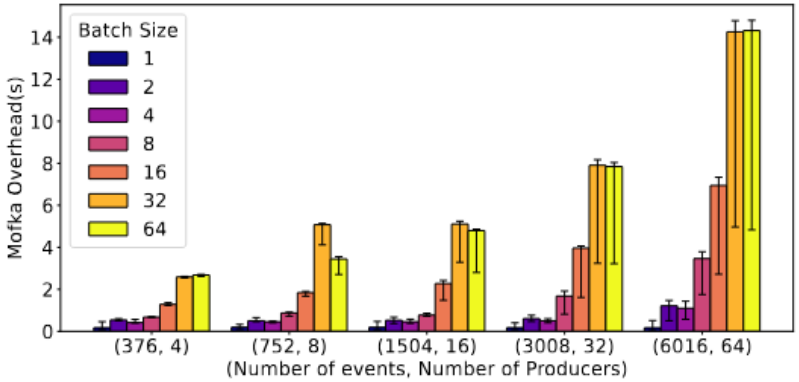


Fig 21. Default asynchronous mode overhead (blocks only if there are more than 2 queued batches) (The number of partitions scales with the number of processes)

Use blocking mode only for extra resilience requirement

ENHANCING TRUSTWORTHINESS IN AGENTIC WORKFLOWS

U.S. DEPARTMENT of ENERGY

Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.

Argonne NATIONAL LABORATORY

WHAT IS AN AGENTIC WORKFLOW?

Why are they fragile?

Autonomy without reliability erodes scientific trust :/

- AI Agents were first introduced in 1999[*]
- An AI Agent comprises 4 fundamental components:
  - A brain
  - A memory
  - Perception tools
  - Action mechanisms
- Agentic workflows incorporate AI agents into traditional workflows (alongside simulations, data analysis, surrogate models, ...)

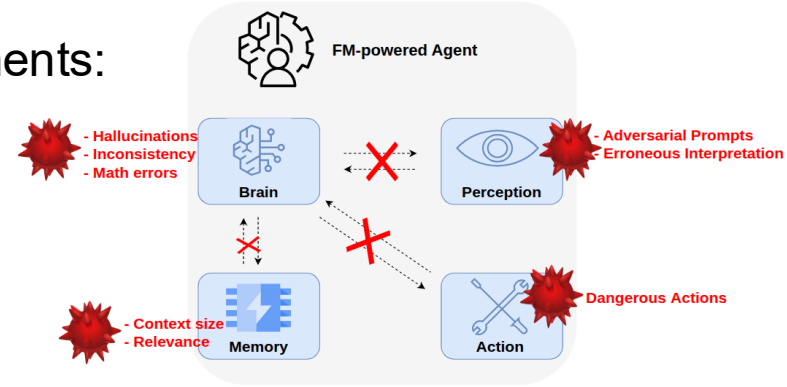


Fig 23. Intra-agent source of failure

[*] J. Ferber and G. Weiss, Multi-agent systems: an introduction to distributed artificial intelligence. Addison-wesley Reading, 1999, vol. 1.
Figure from Gueroudji A. et al., "ControlA: Agentic Workflow Control Mechanisms for Reliable Science", Ai4SC, escience 2025

WHAT MAKES AGENTIC WORKFLOWS FRAGILE?

A **fault** is any *abnormal condition* or defect within an AI agent or agentic workflow that causes *deviation* from expected or reliable behavior

Resilience could be defined as the capacity of the system to detect, contain, and recover from such faults (rolling back the system into a reliable, trustworthy state)

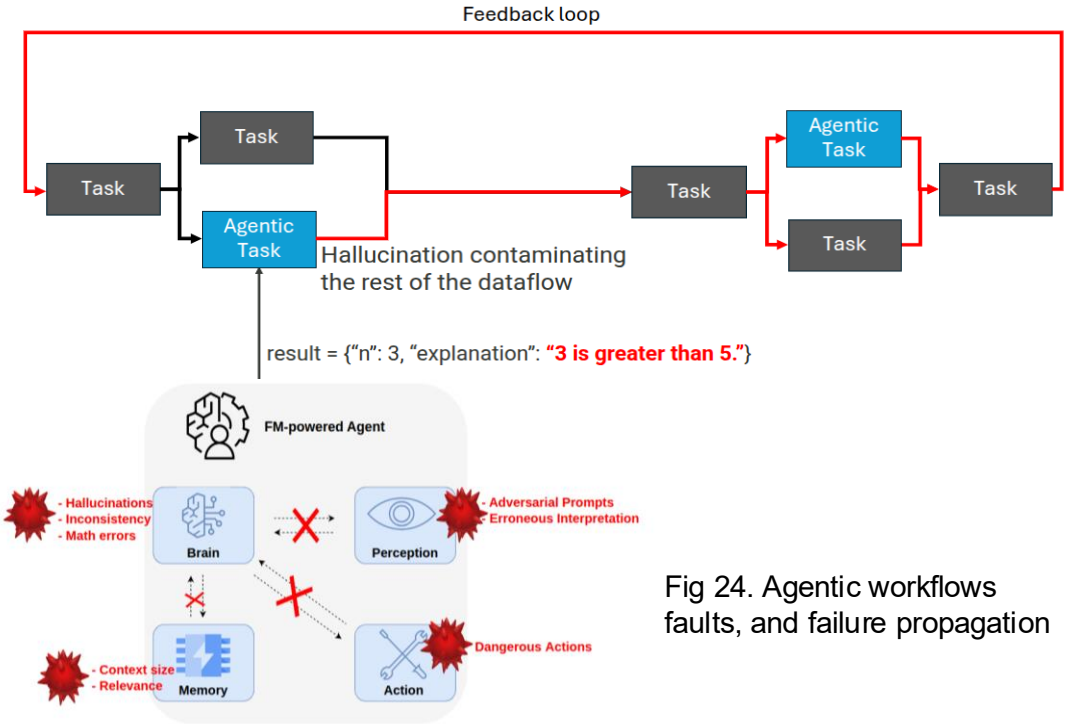
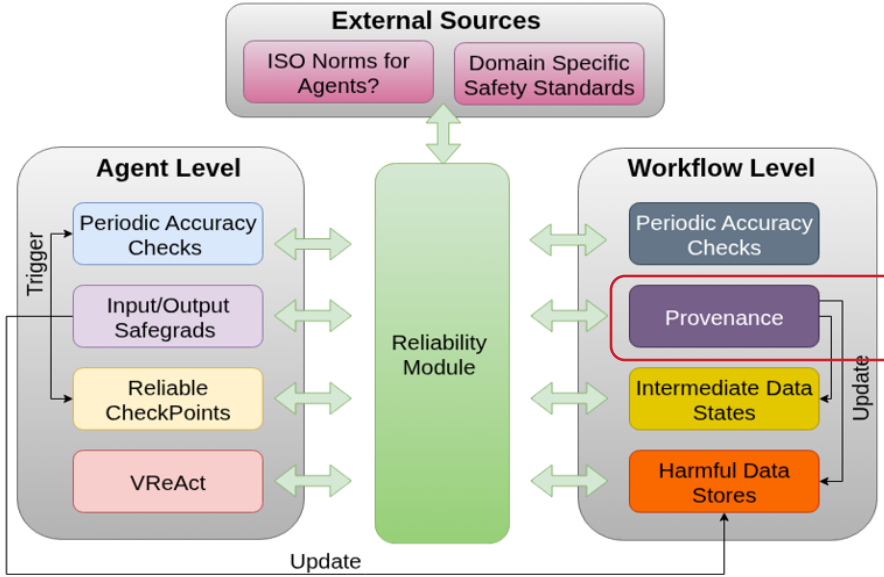


Fig 24. Agentic workflows faults, and failure propagation

CONTROLA: RELIABILITY MECHANISMS FOR AGENTIC SYSTEMS

Two-layered reliability framework for agentic systems



(i) identify a set of initial data and model assets for use in the Mission, including digitization, standardization, metadata, and provenance tracking; and [Genesis]

Fig 25. Agent-level and workflow-level control mechanisms for reliable science

Figure from Gueroudji A. et al., "ControlA: Agentic Workflow Control Mechanisms for Reliable Science", Ai4SC, escience 2025

AGENTIC PROVENANCE

PROV-AGENT

- Definition: “Systematic capture and representation of agents’ **decisions**, **actions**, **interactions**, and their effects within workflows, ensuring that agent behavior is **traceable**, **accountable**, and **connected** to the broader provenance of data and tasks.”
- Capture: Each time an agent runs, its metadata are recorded and linked with the broader workflow provenance.
- W3C PROV+MCP: extending PROV for agentic workflows

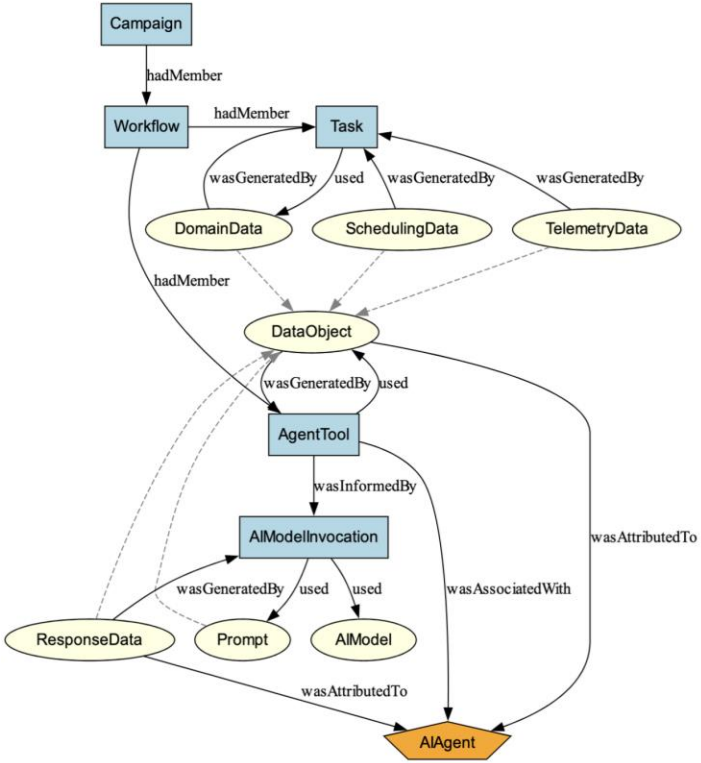


Fig 26. PROV-AGENT: A W3C PROV Extension for Agentic workflows, Dashed arrows represent *subClassOf*

Figure from Souza R. et al., "PROV-AGENT: Unified Provenance for Tracking AI Agent Interactions in Agentic Workflows", ReWorDS, escience 2025

AGENTIC PROVENANCE

Flowcept Architecture

- Data collection:
  - Observability Adapter
  - Code Instrumentation
- Message Queues:
  - Redis
  - Kafka
  - **Mofka**
- Persistence:
  - MongoDB
  - LMDB
- Visualization and Queries:
  - Grafana
  - Flowcept Agent

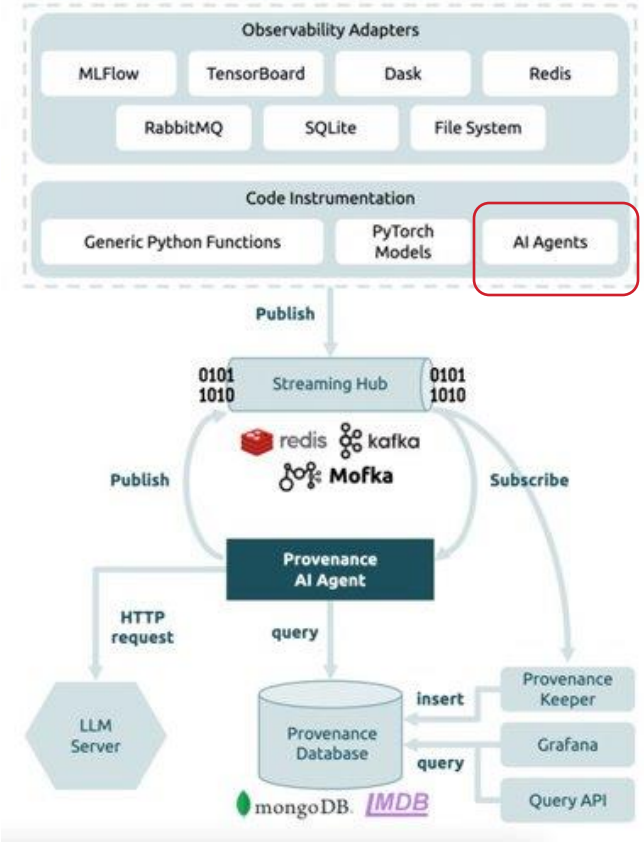


Fig 27. Flowcept architecture, from data collections (top) to analysis and visualization (Bottom), passing by streaming services (Redis, Kafka, Mofka)

Figure from Souza R. et al, "LLM Agents for Interactive Workflow Provenance: Reference Architecture and Evaluation Methodology", WORKS, SC2025

CONCLUSIONS AND FUTURE VISION

CONCLUSIONS AND FUTURE VISION

Contemporary workflows are complex in different ways, this talk presented coping mechanisms on:

- Performance characterization to understand end-to-end workflow behavior
- Resilience as a built-in feature through persistent streaming to survive real-world conditions
- Agentic reliability to recover trust autonomy at scale in a world of creative components

Together, they can fulfill what a scientific workflow could and should be: an observable, resilient, trustworthy instrument of discovery.

CONCLUSIONS AND FUTURE VISION

From pipelines to instruments, from execution to intelligence

Re-thinking workflows for an AI-centric era of science

- **From data to memory:**
  - How do we capture provenance, semantics, and evolution of AI-centric data (embeddings, traces, reasoning artifacts) so that, alongside, vector databases and knowledge stores, they become auditable scientific memory?
- **From fault tolerance to trustworthiness:**
  - How do we redefine resilience when failures are no longer only crashes, but hallucinations, unsafe actions, and reasoning drift, and make reliability a first-class property of agentic systems?
- **From isolated assistants to federated co-scientists:**
  - How do we design agentic systems that collaborate across facilities, instruments, and communities, while respecting policies and scientific accountability?
- **From static execution to resource-aware intelligence:**
  - How do workflows continuously balance energy, time, and scientific accuracy, turning performance telemetry and provenance into closed-loop, self-optimizing control?

REFERENCES

- "Performance Characterization and Provenance of Distributed Task-based Workflows on HPC Platforms", **Amal Gueroudji** et al.
- "A Terminology for Scientific Workflow Systems", Frederic Suter et al.
- "Toward a persistent event-streaming system for high-performance computing applications", Matthieu Dorier et al.
- "Octopus: Experiences with a Hybrid Event-Driven Architecture for Distributed Scientific Computing". Haochen Pan et al.
- "RESILIO : A Scalable and Composable Architecture for Tomographic Reconstruction Workflows", **Amal Gueroudji** et al.
- "ControlA: Agentic Workflow Control Mechanisms for Reliable Science", **Amal Gueroudji** et al.
- "PROV-AGENT: Unified Provenance for Tracking AI Agent Interactions in Agentic Workflows" Renan Souza et al.
- "LLM Agents for Interactive Workflow Provenance: Reference Architecture and Evaluation Methodology", Souza R. et al.
- "Agent Smith: A single image can jailbreak one million multimodal LLM agents exponentially fast", X. Gu, et al.
- "Multi-agent systems: an introduction to distributed artificial intelligence", J. Ferber and G. Weiss.
- "Beyond Centralized Labs: Federating the Co-scientist", ACX, HPCAsia 2026, **Amal Gueroudji** et al.
- Image Gravitational wave
- Image AlphaFold
- Image black hole


Argonne
NATIONAL LABORATORY
U.S. DEPARTMENT of ENERGY

OCTOPUS: CLOUD-BASED STREAMING SERVICE

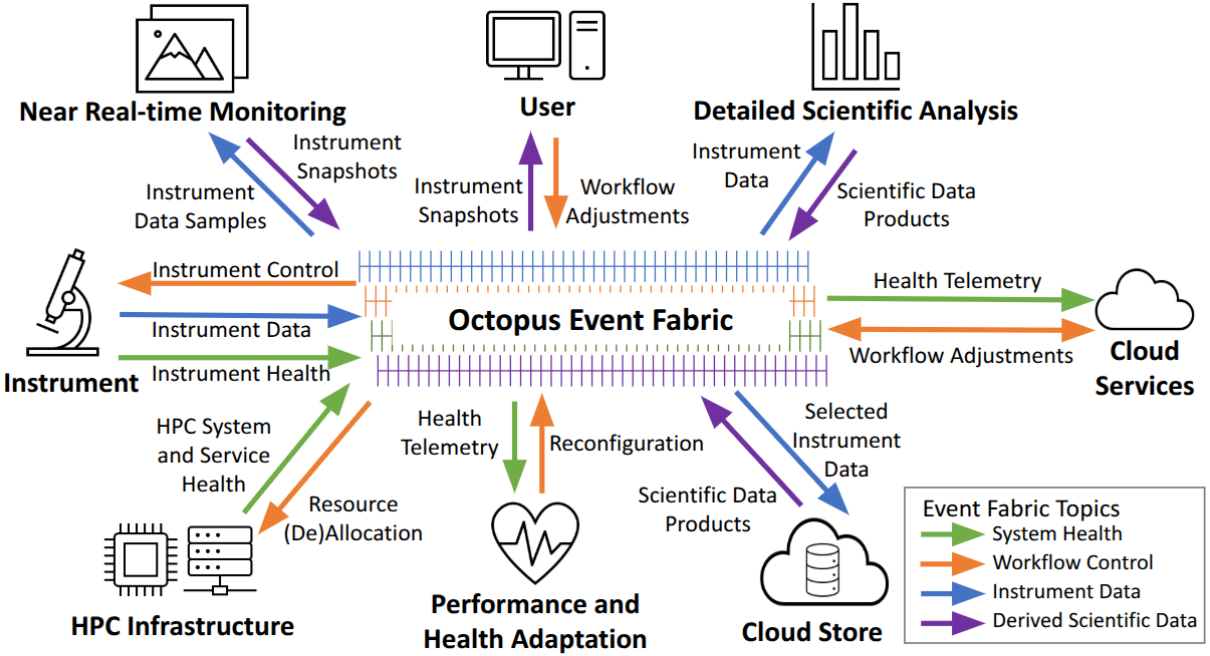


Fig 8. The Octopus event fabric spans locations and provides an available and resilient basis for developing applications. Various event topics allow filtered views by consumers.

Figure from "Octopus: Experiences with a Hybrid Event-Driven Architecture for Distributed Scientific Computing". Haochen Pan et al.

OCTOPUS ARCHITECTURE

- Event Fabric: Built on AWS Managed Streaming for Kafka (MSK) for high-throughput, scalable event streaming.
- Web Service: A RESTful service for managing user authentication, topic ownership and trigger deployment
- Trigger: User-defined AWS Lambda functions with optional event filters, enabling automated and real-time actions to specific event conditions.

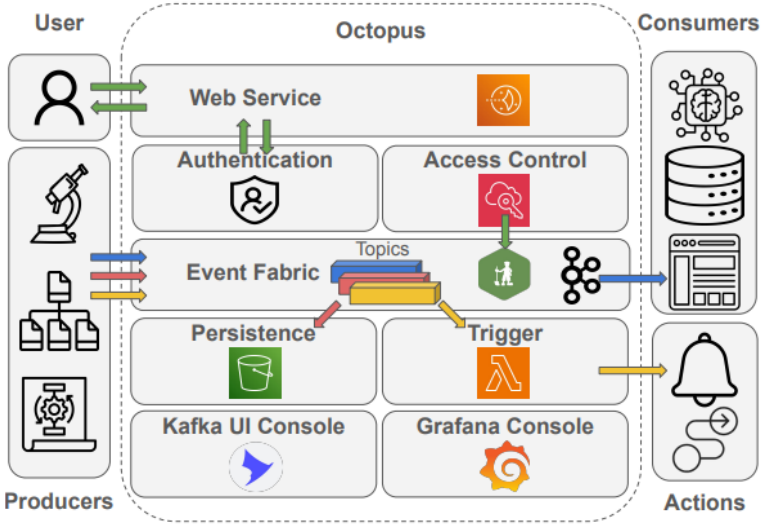


Fig 9. Octopus architecture. Users interact with the web service (green arrows) to acquire credentials, where their identity is shared with the event fabric. Producers (left) and consumers (right) communicate with the event fabric to publish and receive events, respectively (blue arrows). Events can also be persisted to reliable cloud storage when enabled (red arrows). Monitoring consoles are available for admins to monitor the system's live status and historical statistics.

Figure from "Octopus: Experiences with a Hybrid Event-Driven Architecture for Distributed Scientific Computing". Haochen Pan et al.