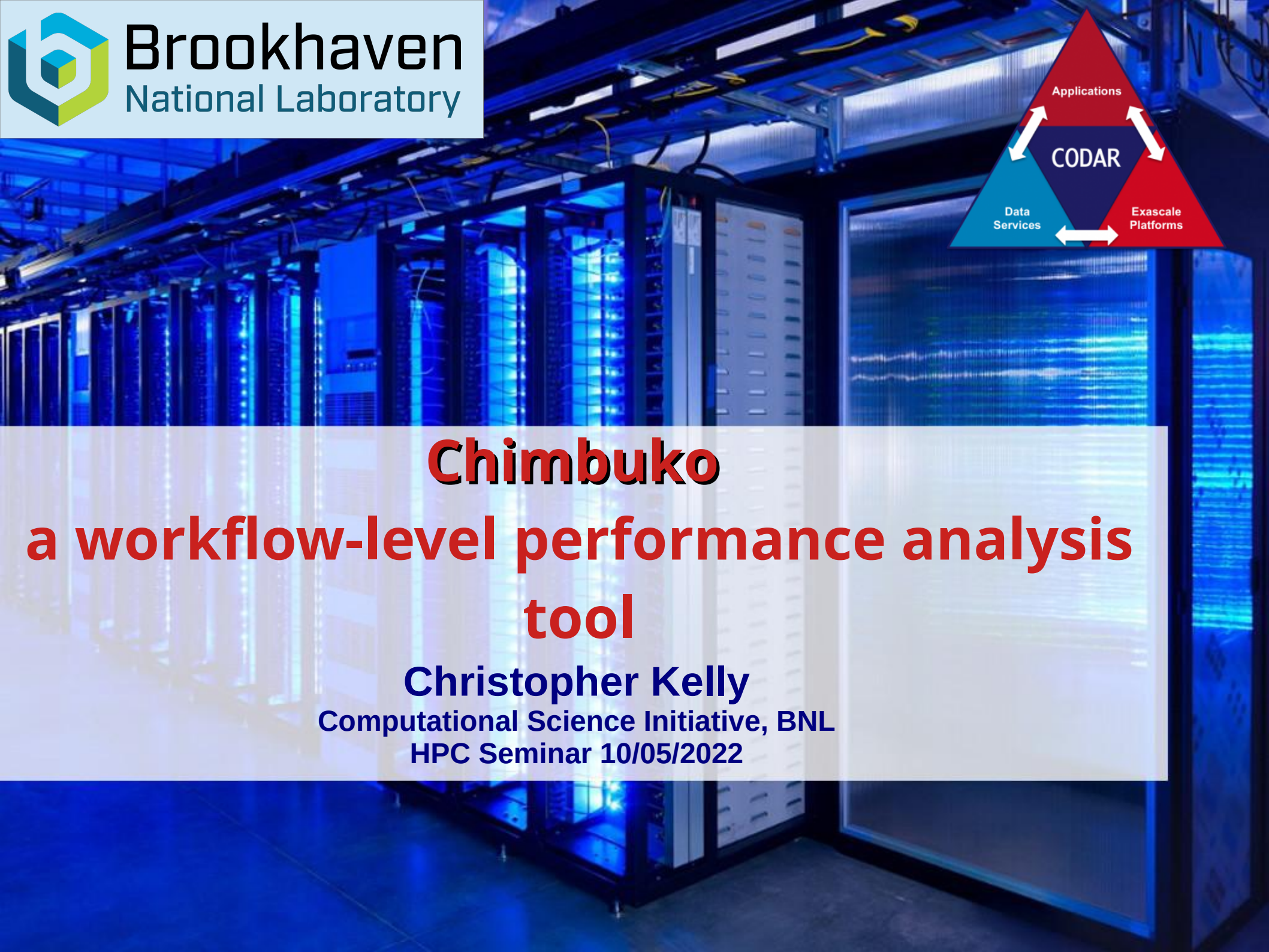
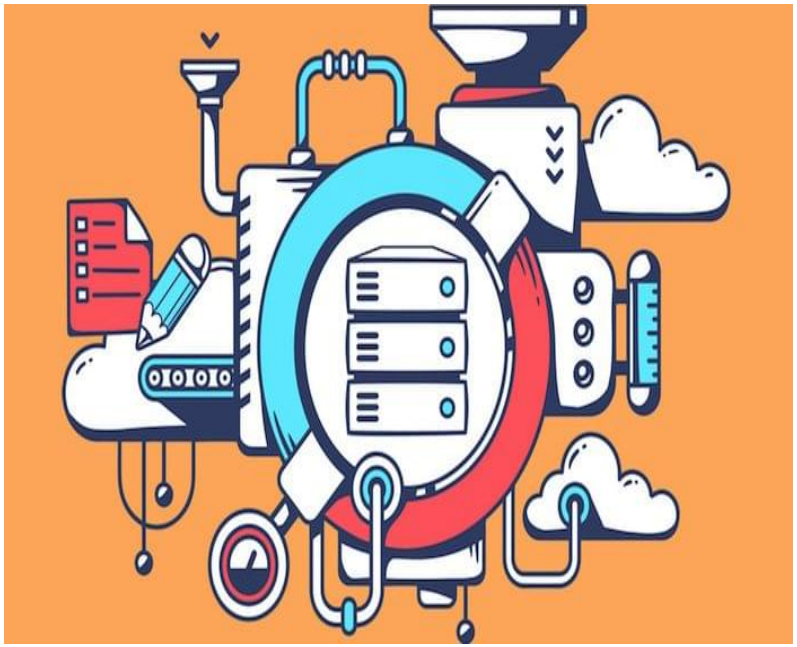


Chimbuko
a workflow-level performance analysis tool

Christopher Kelly
Computational Science Initiative, BNL
HPC Seminar 10/05/2022

The Workflow Challenge

- Many modern scientific applications form **workflows** comprising multiple interacting components.
- Components often **compete** for the same hardware resources.
- Potential for significant **performance issues** when run at-scale:
  - **Resource contention** with other workflow components or other users.
  - Temporary **bottlenecking** due to complex interactions with other components.
  - **Stochastic effects** in coupled system, e.g. interference from OS, power supply fluctuations, cosmic rays.
  - Intermittent **hardware faults**.
- Both users and system admins need to be able to identify and solve these issues.



Traditional diagnosis techniques

- Traditional tools fail:

  - **Benchmarking** and **profiling** individual components in isolation may not detect the problem.
  - **Manual timing** of critical code paths may miss the problem or if they do capture it, are unlikely to tell you *why* it happened.
  - Root cause analysis by storing **application traces** will quickly generate 100s of TB of data, and there are no practical ways to analyze it.

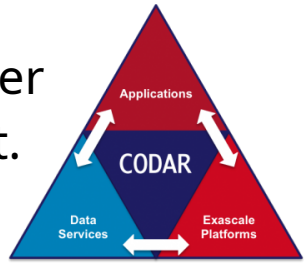
- What we need is a way to combine the **detail** of trace capture with the **ease** and small data volume of simple profiles.
- Enter **Chimbuko**.....

Chimbuko

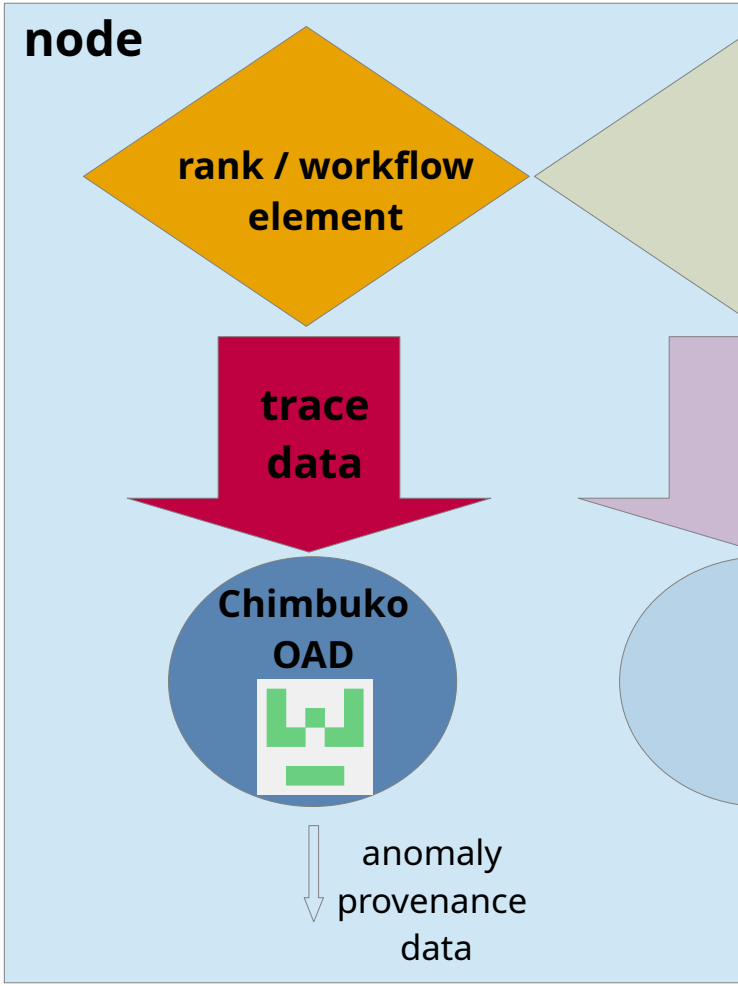
(“Provenance / origin” in Swahili)

[https://github.com/CODARcode/Chimbuko]

- Chimbuko is a tool developed under the Codesign Center for Online Data Analysis and Reduction (CODAR) project.
- Sponsored by the Exascale Computing Project (ECP).
- It succeeds in the aforementioned goals by performing **real-time *in situ* analysis** of trace data.
- The tool dynamically builds a model of the application profile of each component of the workflow.
- The model is used to isolate **anomalous behavior**, utilizing streaming anomaly detection algorithms.
- Only detailed information on the anomalies are captured and stored.
- By focusing only on anomalies we achieve **very high reduction** in data volume while retaining key information required for causal analysis.

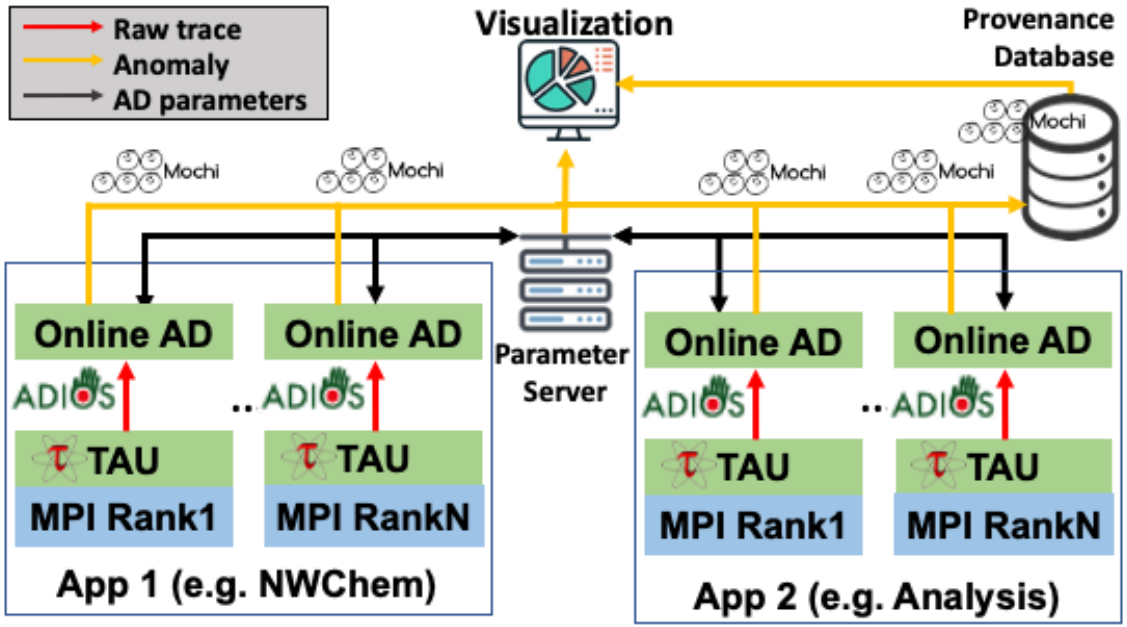


EXASCALE COMPUTING PROJECT

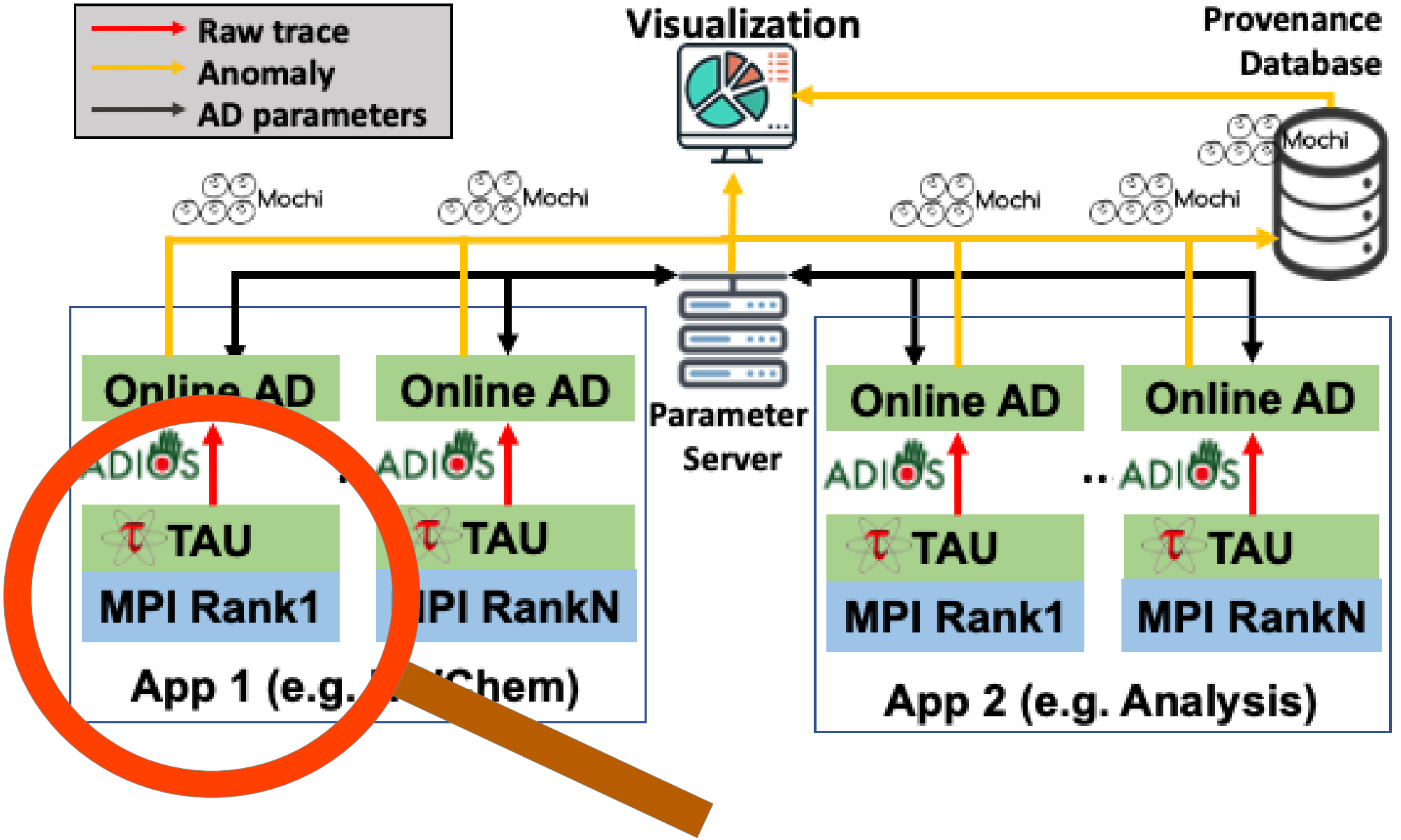


Chimbuko design overview

- **TAU Performance System:**
  - Generates real-time traces and sends to OAD.
- **Online AD**:
  - Trains anomaly detection algorithm on incoming trace data.
  - Applies algorithm to filter out anomalies
  - Gathers detailed provenance information on each anomaly.
- **Provenance database:**
  - Acts as a centralized database provenance information.
- **Parameter server:**
  - Aggregates / synchronizes parameters of AD model between OAD instances.
  - Exploit coexistence of many identical instances of workflow components.
  - Allows very rapid training that *improves with job scale.*



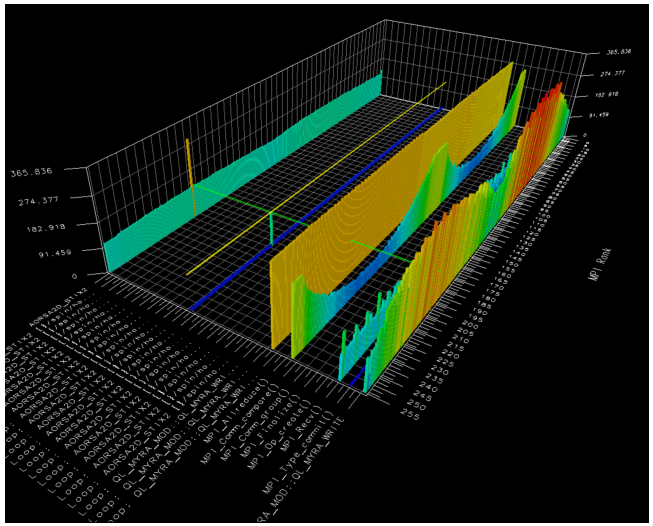
- Aggregates global process information / statistics and forwards to visualization.
- **Visualization**:
  - Displays real-time information on captured anomalies.
  - GUI allows manual interaction with the provenance database as the analysis is being performed.

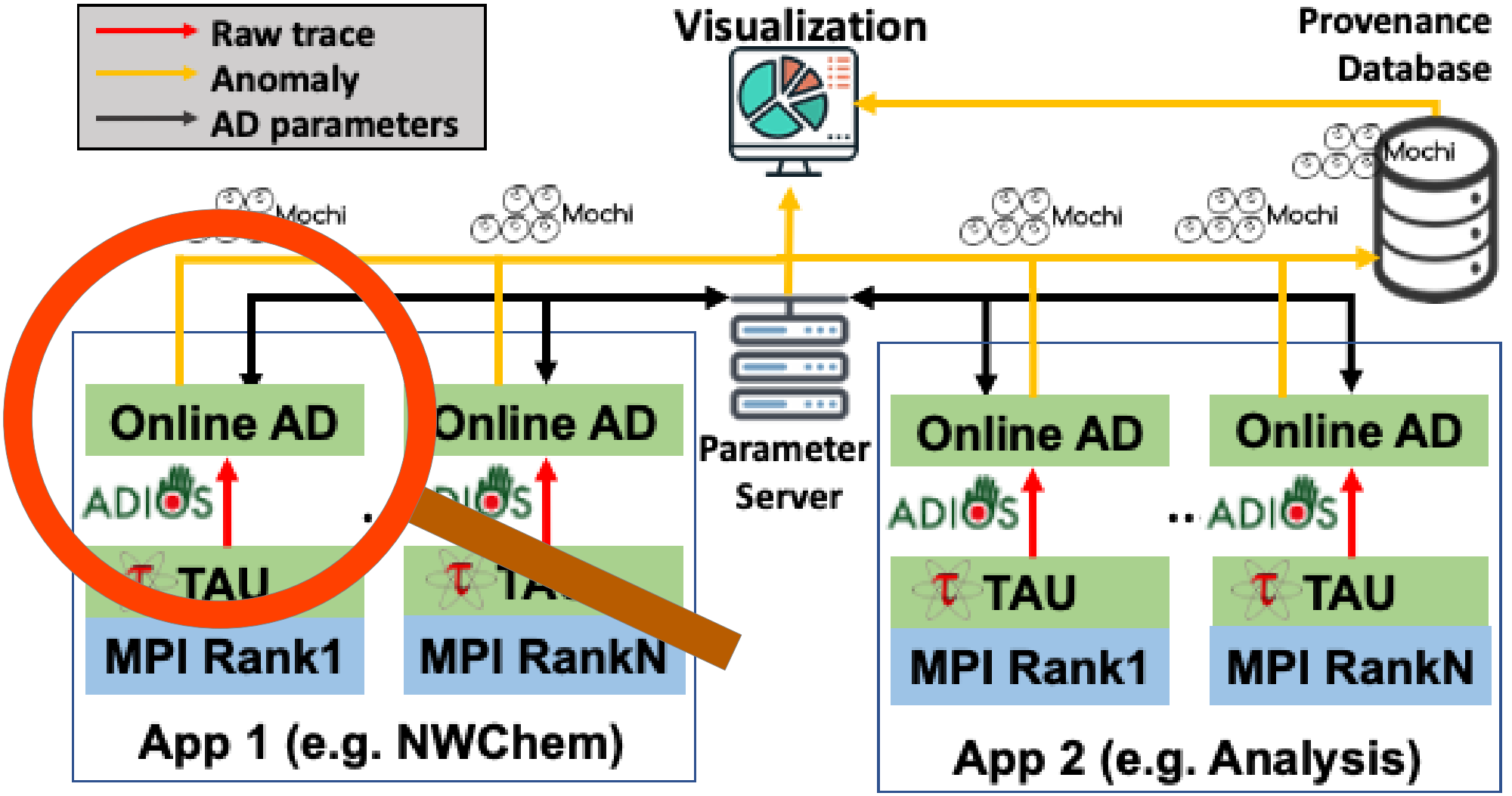

Raw trace
Anomaly
AD parameters
Visualization
Provenance Database
Mochi
Mochi
Mochi
Mochi
Mochi
Online AD
Online AD
Online AD
Online AD
Parameter Server
ADIOS
ADIOS
ADIOS
ADIOS
TAU
TAU
TAU
TAU
MPI Rank1
MPI RankN
MPI Rank1
MPI RankN
App 1 (e.g. GTC)
App 2 (e.g. Analysis)

TAU tracing

https://github.com/UO-OACISS/tau2
https://github.com/ornladios/ADIOS2

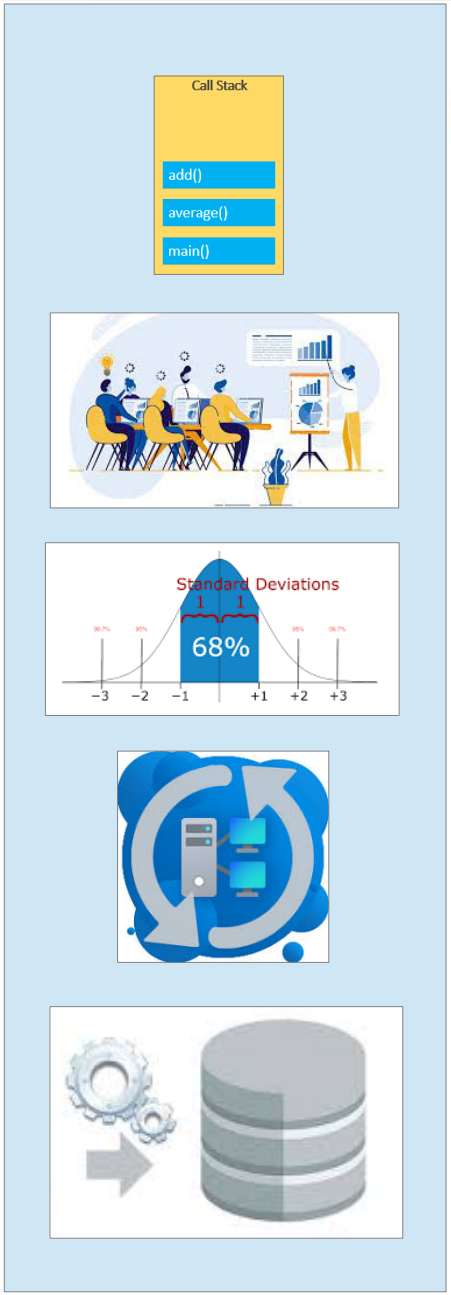
- Application performance traces are provided by TAU.
- **GPU kernel traces** supported for all major platforms via vendor APIs
- **CPU traces** require instrumentation of source code for compiled languages:
  - TAU PDT tool for Fortran, C, old C++
  - TAU LLVM plugin for modern C++ (backup Python tool)
  - TAU LLVM backend for auto-instrumenting source allows C++ instrumentation with other compilers (experimental)
  - Compiler instrumentation for most languages and compilers is available but often instruments *everything* ; larger overheads
- **Python support** available but also instruments everything
- (Use selective instrumentation to reduce overheads)
- Additional data:
  - **PAPI counters** (TAU compilation option)
  - **Node state** (memory/cpu utilization,etc) from */proc/pid* (monitoring plugin).
- TAU trace data output **in batches** (typically 1/s) using *ADIOS2* library; either via memory (SST) or disk (BPFile/BP4/BP5).




Raw trace
Anomaly
AD parameters
Visualization
Provenance Database
Mochi
Mochi
Mochi
Mochi
Mochi
Online AD
Online AD
Online AD
Online AD
Parameter Server
ADIOS
ADIOS
ADIOS
ADIOS
TAU
TAU
TAU
MPI Rank1
MPI RankN
MPI Rank1
MPI RankN
App 1 (e.g. NWChem)
App 2 (e.g. Analysis)

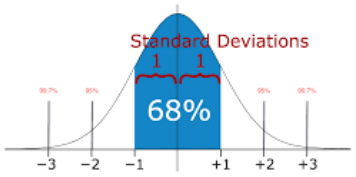
Online Anomaly Detection

- 1 instance of OAD per application rank / workflow component instance.
- **Roles**:
  - **Parse trace data** into call stack, assigning counters and MPI events to function executions.
  - **Train** local AD model and synchronize with parameter server to obtain global model.
  - Apply global model to function execution times to **identify anomalies.**
  - **Gather provenance data** and send to provDB.
  - **Compute batch statistics** and send to parameter server.
- **Requirements:**
  - Support O(100k+) trace events per second.
  - Complete all activities on batch *within batch receive period.*
  - Low overheads to avoid interfering with application.
- **Features:**
  - Highly configurable with numerous options for customizing the analysis.
  - Flexible launch: can be launched with or without MPI.

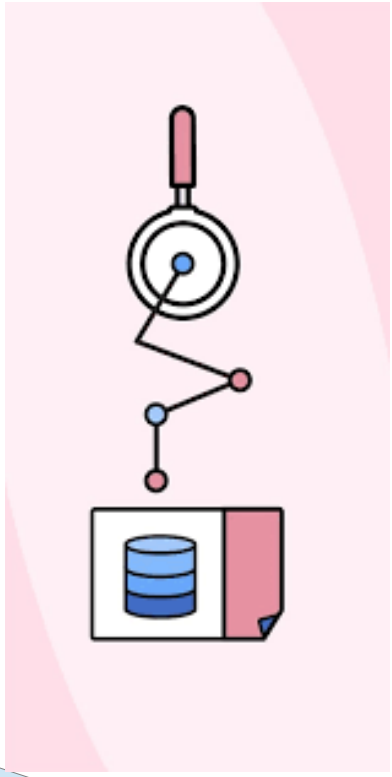


Anomaly Detection Algorithms

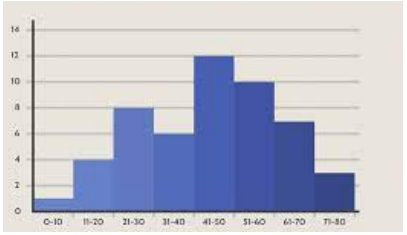
- Due to **extreme data volume** we cannot utilize many common anomaly detection algorithms which act on the entire dataset.
- Instead we require **streaming** (batched) algorithms.
- Algorithms must be **unsupervised** with **minimal hyperparameters.**
- Currently provide 3 algorithms based on the **function execution time.**



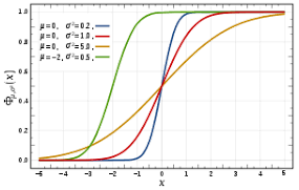
- **SSTD (Sample STandard Deviation):**
  - Compute moments (mean, variance) for each function execution.
  - Synchronization combines across ranks to obtain global moments.
  - Anomalies are assigned based on number of std. deviations from mean.



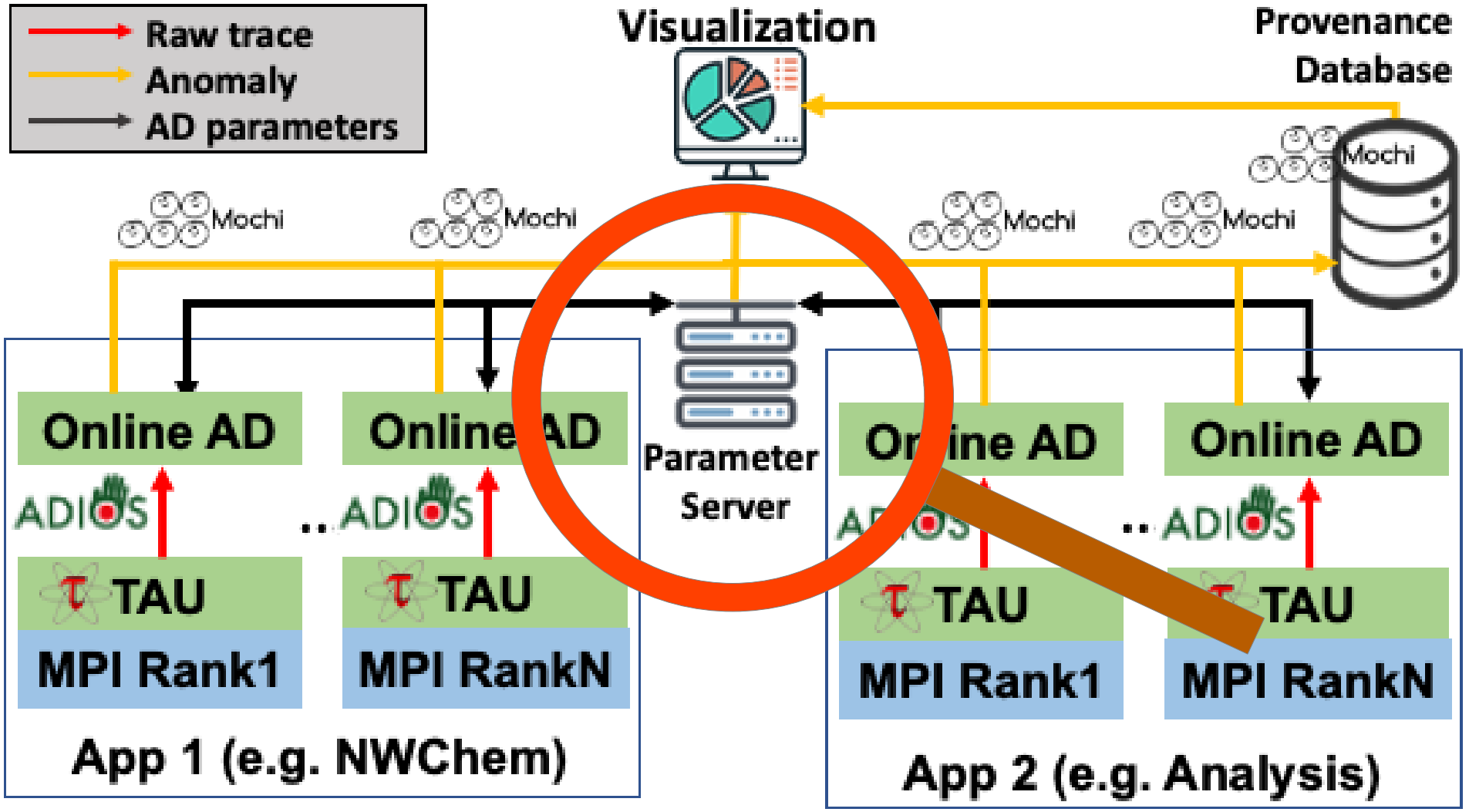
- **HBOS (Histogram Based Outlier Selection):** (Goldstein, Dengel 2012)
  - Generate a local histogram for each function.
  - Synchronization merges histogram across ranks.
  - Scores are assigned based on bin probability of each event
  - Dynamic threshold seeks to isolate only extreme outliers.



HBOS best choice for multimodal distributions



- **COPOD (Copula Based Outlier Detection):** (Li 2009)
  - Use histograms in same way as HBOS but utilizes copula (generalization of CDF) to assign event scores.

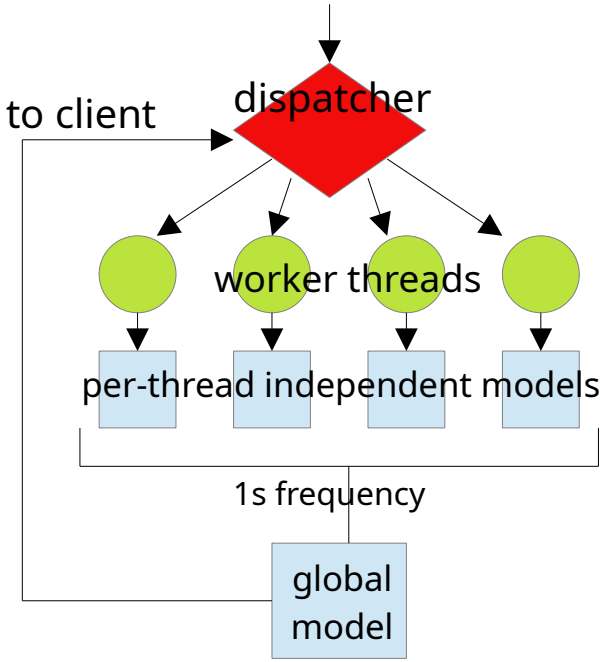

Raw trace
Anomaly
AD parameters
Visualization
Provenance Database
Mochi
Mochi
Mochi
Mochi
Mochi
Online AD
Online AD
Online AD
Online AD
Parameter Server
ADIOS
ADIOS
ADIOS
ADIOS
TAU
TAU
TAU
TAU
MPI Rank1
MPI RankN
MPI Rank1
MPI RankN
App 1 (e.g. NWChem)
App 2 (e.g. Analysis)

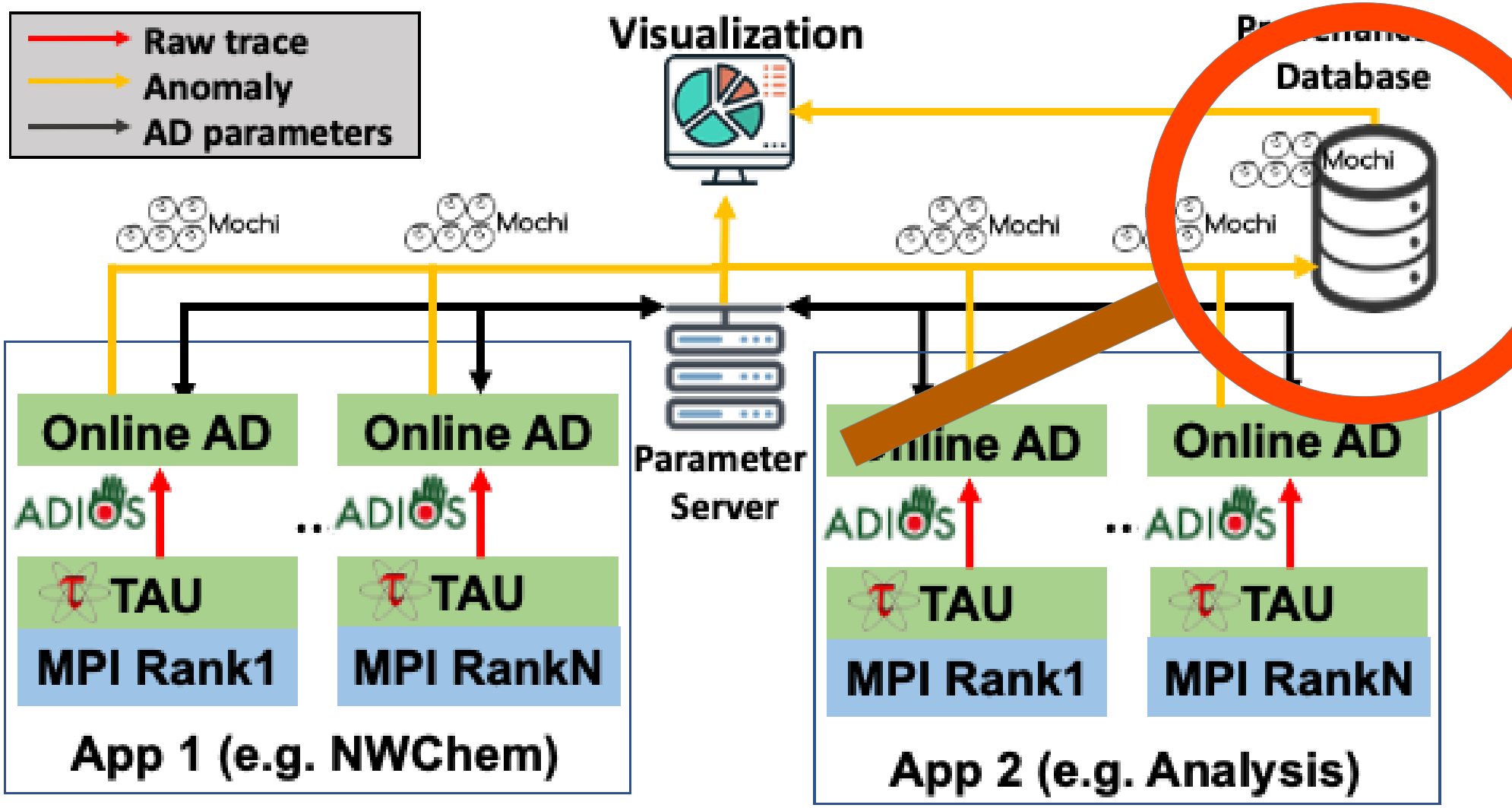
Parameter server

https://zeromq.org/

https://uscilab.github.io/cereal/

- **Roles:**
  - Maintains and synchronizes the AD model with the clients.
  - Aggregates **profile information**, and **statistics** on anomalies, counters, etc.
    - These are forwarded to the provDB at the end of the run.
    - Also optionally forwarded in realtime to the visualization.
- **Requirements:**
  - Must accept connections from remote clients across many possible network types.
  - Must be **scalable** to support 1000s of clients.
  - Synchronization is **blocking;** require **minimal latency (<<1s).**
- **Design:**
  - Custom RPC server implementation using **ZeroMQ** with **Cereal** serialization. Supports many client workers.
  - Workers update independent instances of AD model **maximizing parallelization**.
  - These instances are combined by a separate external thread once per second to ensure the global model is current.

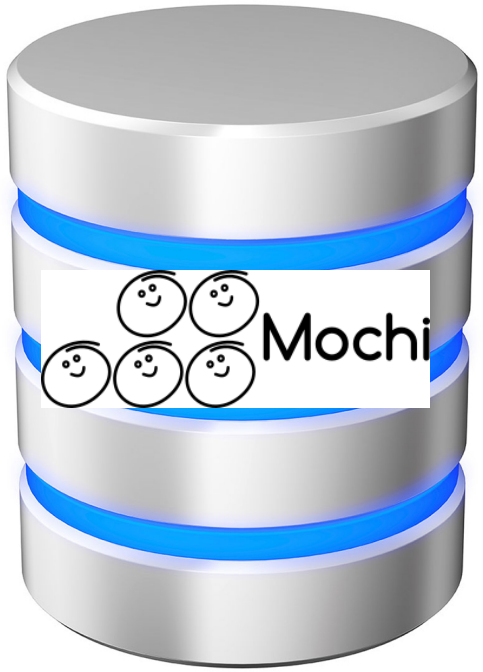



Raw trace
Anomaly
AD parameters
Visualization
Database
Mochi
Mochi
Mochi
Mochi
Mochi
Online AD
Online AD
Online AD
Online AD
Parameter Server
ADIOS
ADIOS
ADIOS
ADIOS
TAU
TAU
TAU
TAU
MPI Rank1
MPI RankN
MPI Rank1
MPI RankN
App 1 (e.g. NWChem)
App 2 (e.g. Analysis)

Provenance database

https://mercury-hpc.github.io/
https://github.com/pmodels/argobots
https://github.com/mochi-hpc/mochi-sonata

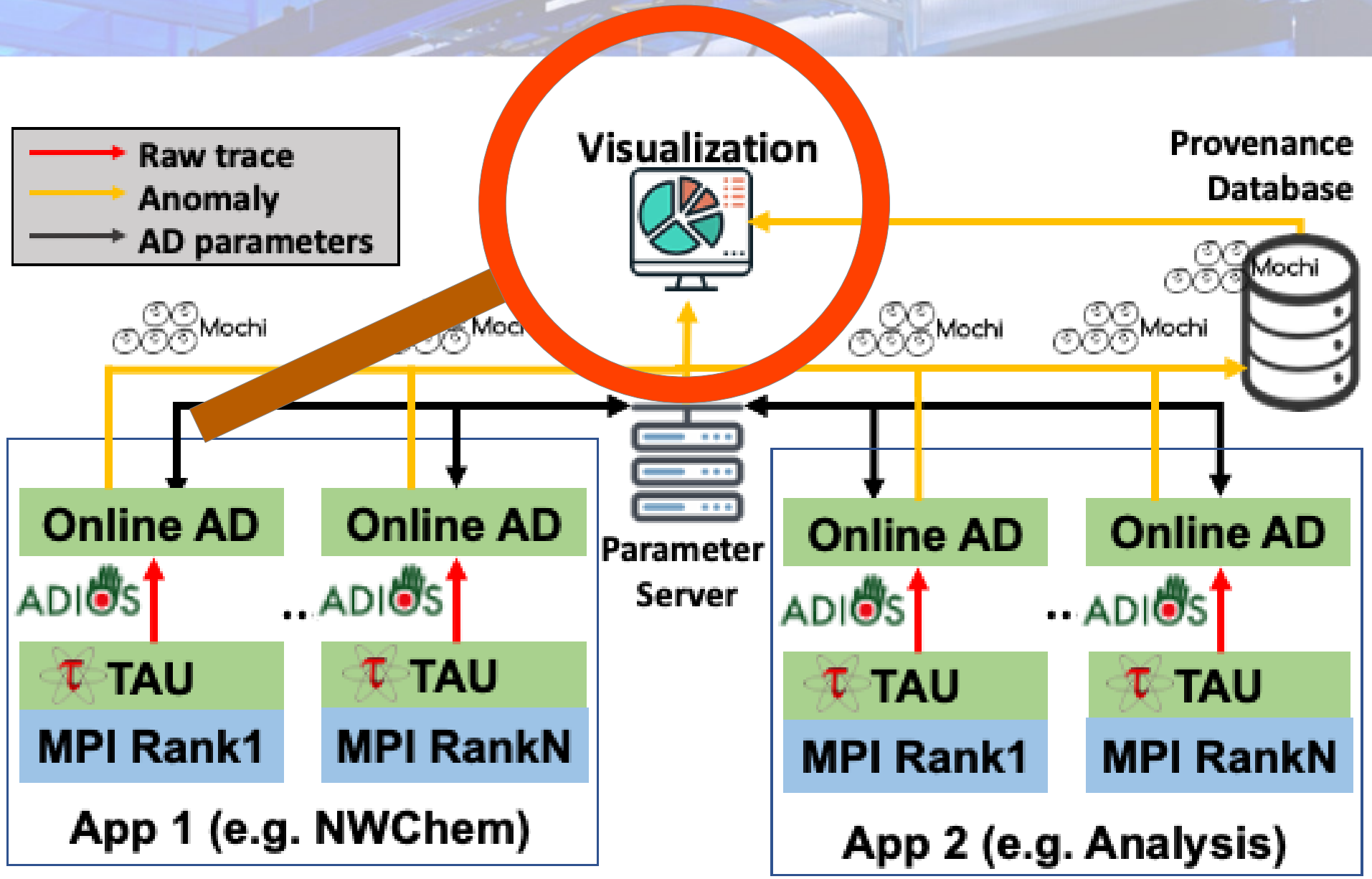
- **Role:**
  - Store and allow querying of anomaly provenance data from clients.
  - Store and allow querying of global information (profiles, final AD model, statistics) from parameter server.
  - All records are stored as JSON documents.
- **Requirements:**
  - Accept connections from remote clients across many possible network types.
  - **Scalable** to support 1000s of clients.
  - Convenient API for search and filtering by user and visualization.
  - Support asynchronous sends from clients for latency independence.
- **Design:**
  - Co-designed with the ECP Mochi team.
  - **Mochi** stack implements a generalized RPC server/client built on top of **Mercury** (RPC) and **Argobots** (threading).
  - Mochi and Chimbuko teams codesigned a flexible JSON remote database solution, **Sonata**
  - Achieve arbitrary **scalability** through database sharding for thread scalability and multiple servers if necessary.
  - Offline query tools programmed through convenient Python and C++ APIs.



Provenance information

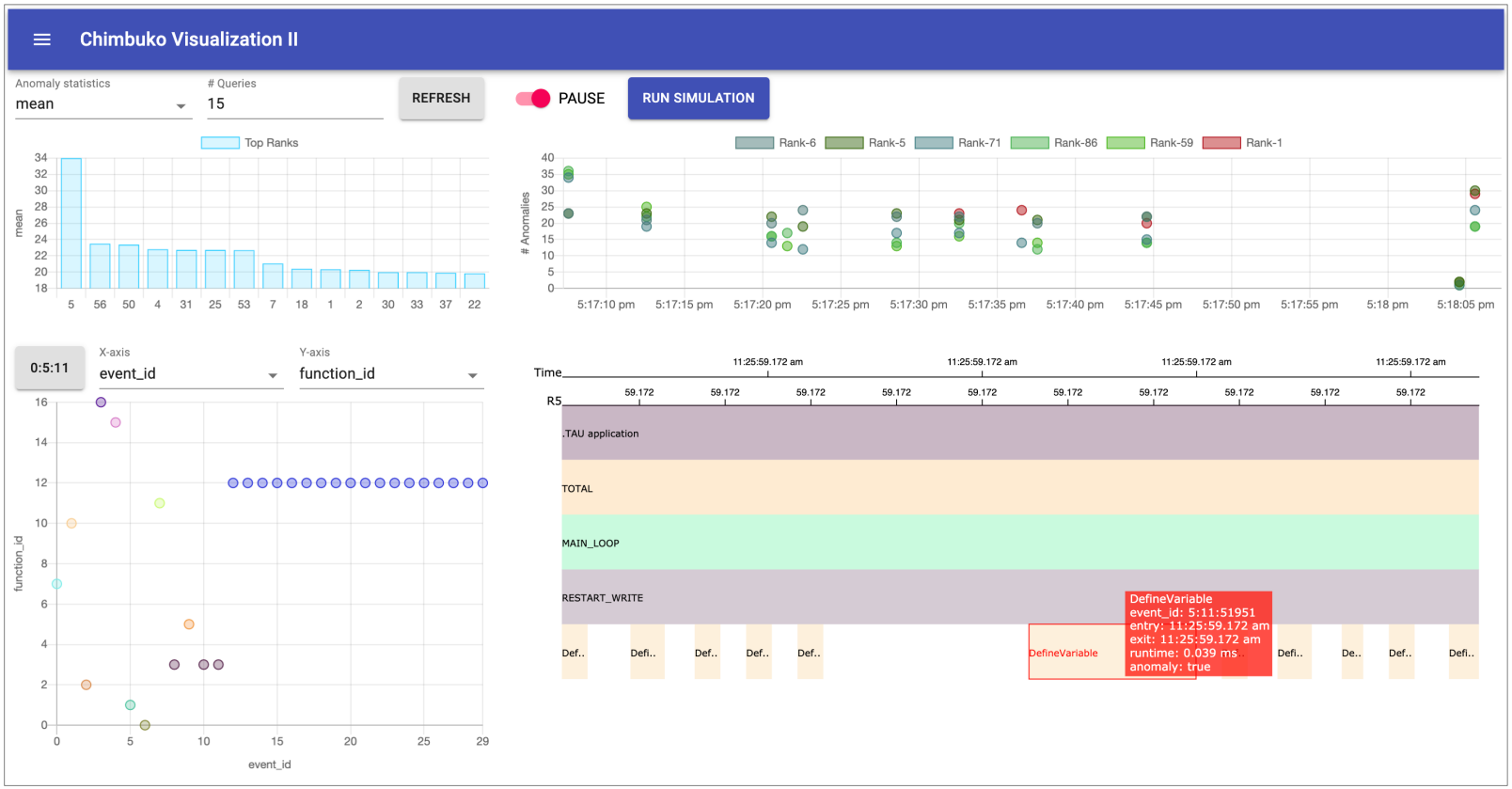
(cf. https://chimbuko-performance-analysis.readthedocs.io/en/ckelly_develop/io_schema/schema.html)

- The goal is to allow a user to identify
  - **How important** the anomaly was:
    - Anomaly metrics associated with likelihood and importance
    - The algorithm parameters at the time of identification
  - **Where** the anomaly occurred:
    - Rank, device/thread, date/time
    - Call-stack of function execution
    - Window snapshot of executions before/after
    - For GPU events, which CPU-side execution launched the kernel
  - **What** happened during the anomalous execution:
    - Execution inclusive/exclusive runtime.
    - MPI events occurring during execution.
    - Counters from PAPI or other sources if supported by TAU.
    - Node state including CPU usage, memory usage, cache misses (*/proc/pid)*
    - A selection of non-anomalous events are also collected for comparison.
- Hopefully together this enables the user to identify **why**

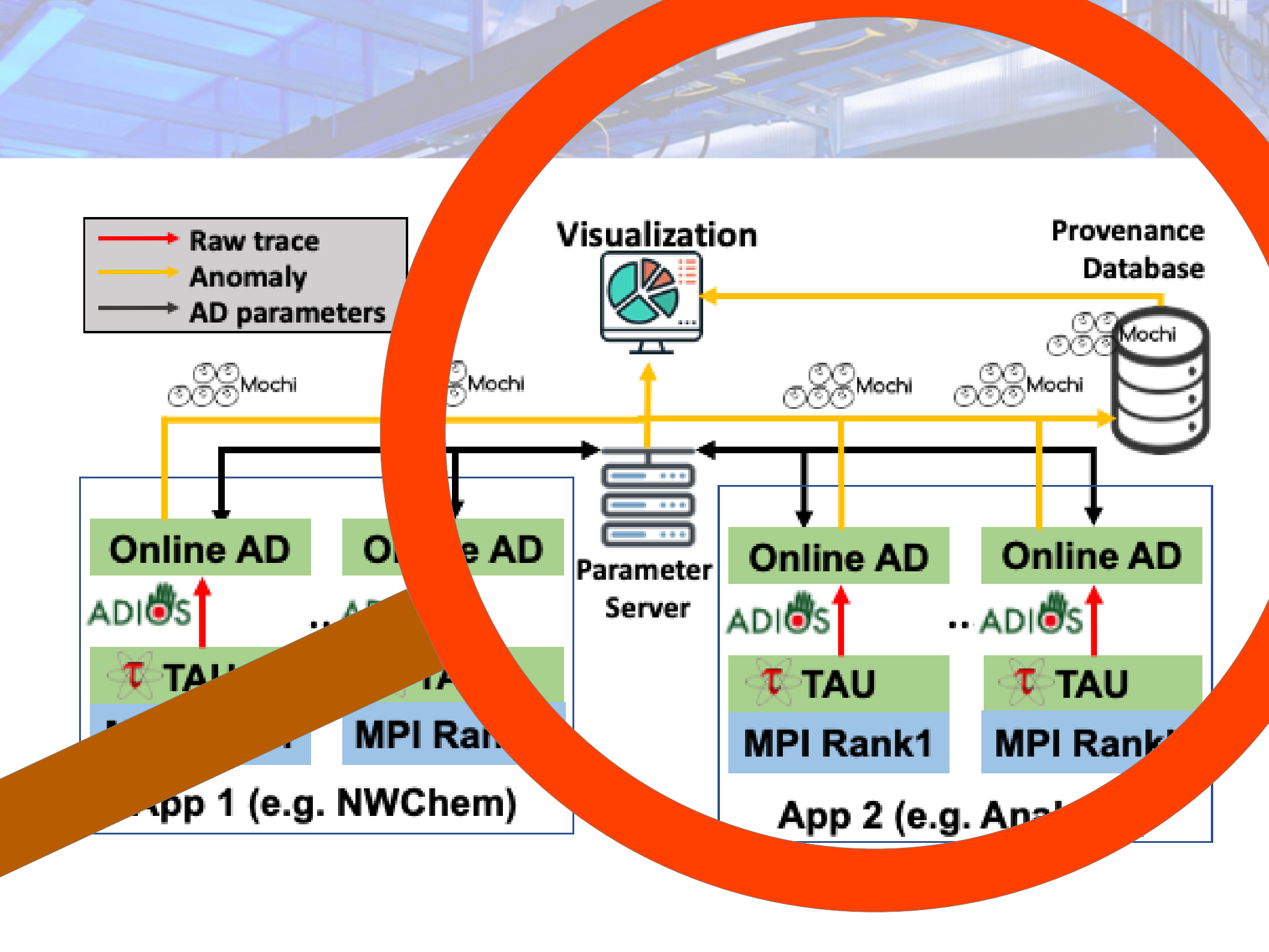

Raw trace
Anomaly
AD parameters
Visualization
Provenance Database
Mochi
Mochi
Mochi
Mochi
Online AD
Online AD
Online AD
Online AD
Parameter Server
ADIOS
ADIOS
ADIOS
ADIOS
TAU
TAU
TAU
TAU
MPI Rank1
MPI RankN
MPI Rank1
MPI RankN
App 1 (e.g. NWChem)
App 2 (e.g. Analysis)

Chimbuko visualization

https://github.com/CODARcode/ChimbukoVisualizationII



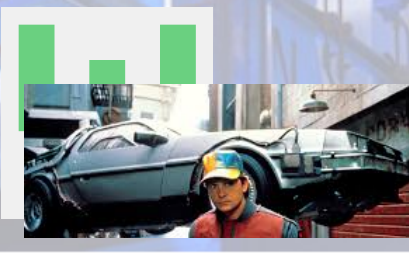
- **Roles:**
  - Obtain and display real-time statistics information to monitor the application status.
  - Allow user to focus in increasing detail on specific ranks / anomalies, querying the provDB where appropriate.
- **Design:**
  - Browser-based interactive frontend built using JavaScript/React with Python backend (+Redis/Celery).
  - Dynamic access to provenance database via Sonata API.
  - Connect to backend server running on job head node via ssh tunnel.


Raw trace
Anomaly
AD parameters
Visualization
Provenance Database
Mochi
Mochi
Mochi
Mochi
Mochi
Online AD
Online AD
Online AD
Parameter Server
ADIOS
ADIOS
ADIOS
TAU
TAU
MPI Rank1
App 1 (e.g. NWChem)
App 2 (e.g. Ana

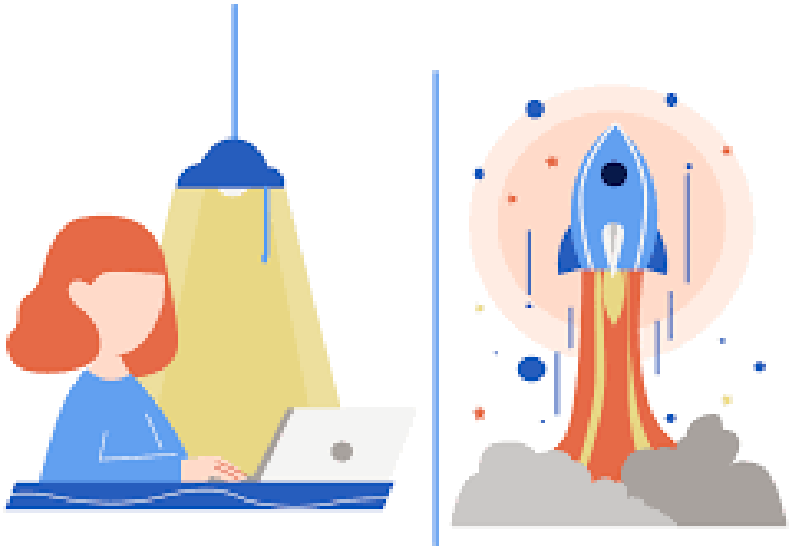
Chimbuko current features

- Analyze arbitrary workflows at-scale with **robust components** tested to scales of 100s of ranks on HPC installations (e.g. Summit).
- Supports workflow components written in **most major languages** (C/C++, Python, Fortran) and vendor-specific GPU paradigms (Cuda, HIP, DPC++) and common APIs (MPI, OpenMP).
- Launch of service components controlled by unified interface with a single configuration script.
- Online AD component can be launched via MPI or by hand in a wrapper script. Can also be integrated into workflow tools (e.g. Radical Cybertools).
- Supports **multiple AD algorithms** and offers **flexible control** for filtering out uninteresting data (e.g. filter-out functions, choose minimum anomaly time) and for tuning hyperparameters.
- **Detailed anomaly provenance** stored in a queryable database accessible with command line tools and full Python API for user analysis scripts.
- Online monitoring tool allows **real-time insights** into workflow performance.
- **Offline analysis tools** (currently rudimentary) offer at-a-glance analysis of profile and anomalies for single runs and between multiple runs.

Chimbuko planned features

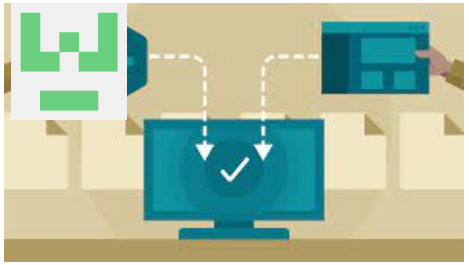


- Aim to expand on the capabilities of Chimbuko into the future:
  - **Incorporate counter information into anomaly detection** rather than just execution time.
  - Expand on **offline analysis tools** with command line and GUI-based tools for exploring the database and performing causal analysis.
  - Explore **closer integration with TAU** to simplify launch procedure.
  - Explore options to remove the parameter server as a bottleneck to **arbitrary scalability**.
- And more!



Obtaining and running Chimbuko

- Recommended way to install Chimbuko is through the **Spack** package manager.



- Spack repo available in source:

```
git clone https://github.com/CODARcode/PerformanceAnalysis.git
git clone https://github.com/mochi-hpc/mochi-spack-packages.git
spack repo add PerformanceAnalysis/spack/repo/chimbuko mochi-spack-packages
spack install chimbuko
```

- Provide **Spack environment configurations** for several HPC installations (Summit, Crusher, Spock, HPC1) to utilize system libraries, MPI and GPU APIs.
- **Docker images** for experimentation are available:

```
docker pull chimbuko/run_examples:latest
```

- **Detailed documentation** available including instructions on basic running as well as running on some major HPC installations:
  - https://chimbuko-performance-analysis.readthedocs.io/en/ckelly_develop/index.html
- **Numerous examples** of run and configure scripts can be found in the PerformanceAnalysis source code:
  - PerformanceAnalysis/benchmark_suite/

If you are interested in trying out Chimbuko, feel free to contact me!
<ckelly@bnl.gov>

https://chimbuko-performance-analysis.readthedocs.io/en/ckelly_develop/index.html

https://github.com/CODARcode/Chimbuko (`ckelly_develop` branch for bleeding edge)