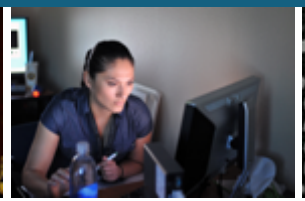
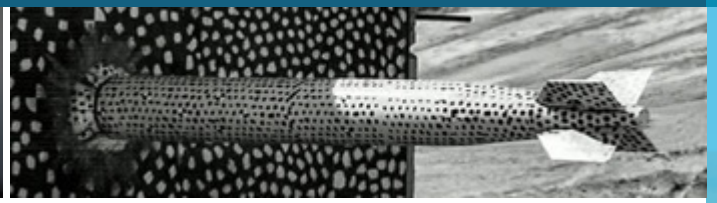
SAND2021-7996C

Sandia National Laboratories

# Integrating System State and Application Performance Monitoring: Network Contention Impact





Jim Brandt (SNL) presenting

*CUG 2021*

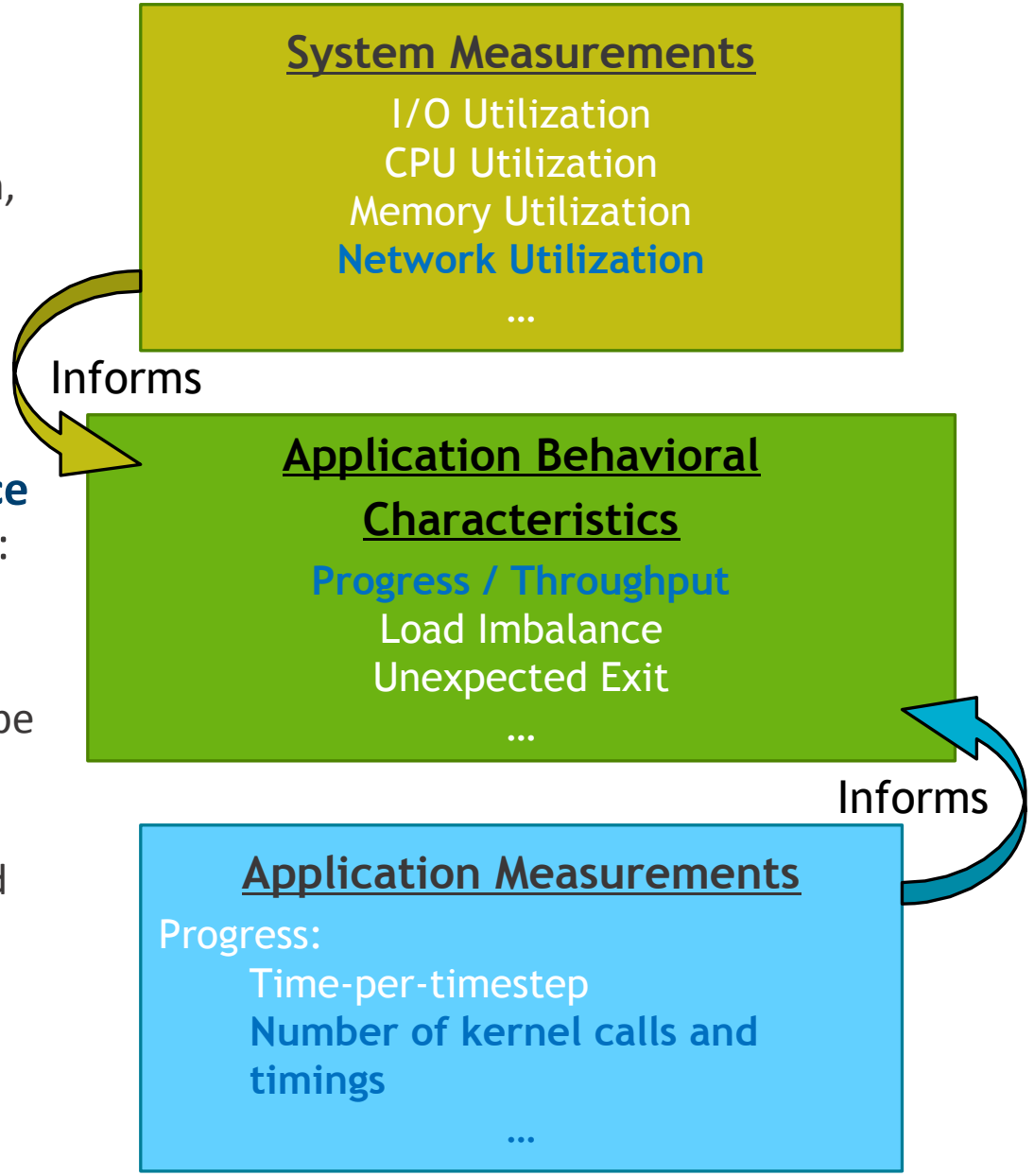
SAND

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.

# Goal: Understanding Application Performance

- **Assertions:**
  - **Resource state** (e.g., utilizations, faults/failures, contention, performance limits) **affects application performance.**
  - The **dynamic nature** of system state over the time of an applications execution **makes effects** on application performance **difficult to quantify.**
- **Fusion of system and application state and performance metrics can provide insights into application behaviors**:
  - **Temporal association** of application progress with changes in system resource state
  - **Location** (e.g., spatial, temporal) of **behavior of interest** can be **expedited** through **examination** of an application's **run time progress**
  - **Quantify relationships** between application performance and **degree** of system resource **contention**

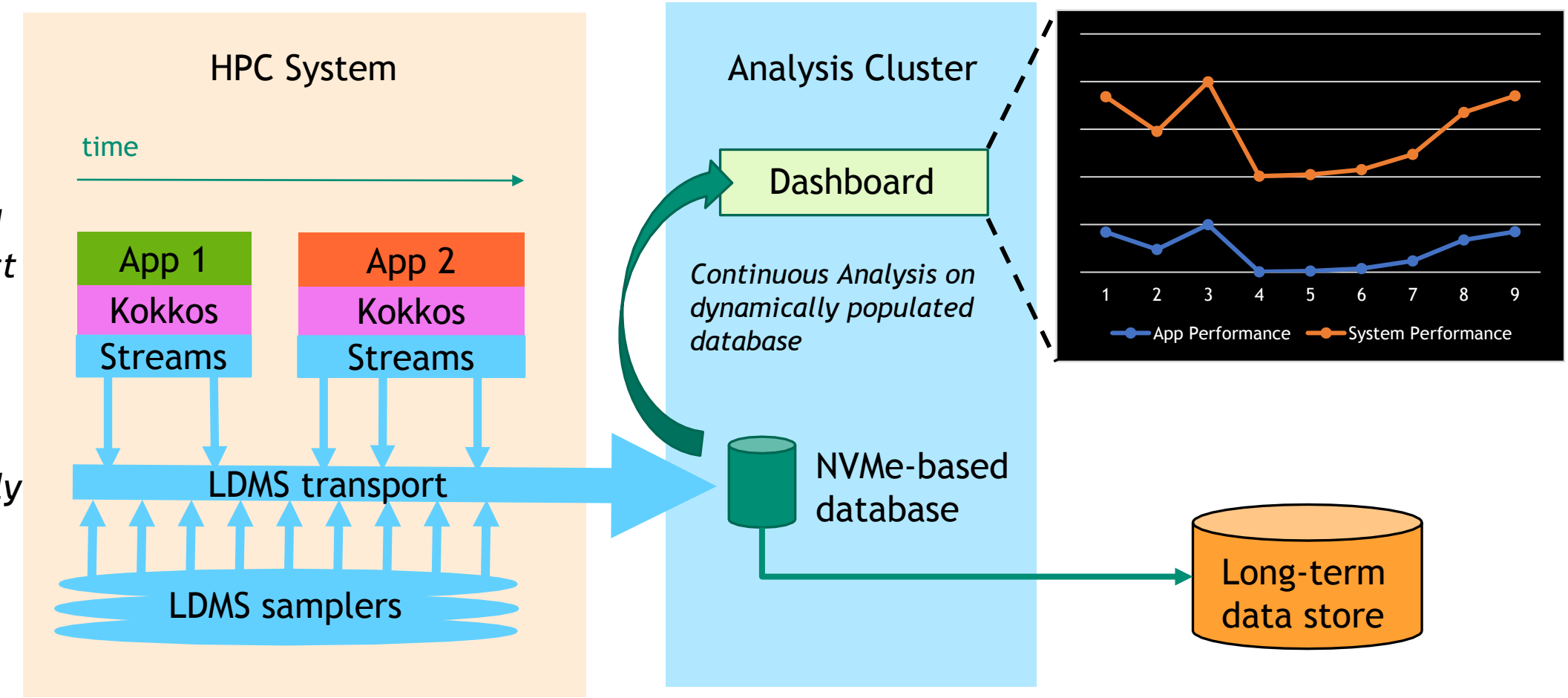


# Unified Framework for **Continuous, Run Time, Fused** System and Application Performance Monitoring and Analysis

Data Flow Diagram

***Applications*** *dynamically and irregularly inject data into the LDMS transport*

*LDMS continuously and regularly collects and transports full* ***system data***



# Enabling Application Data Injection: LDMS Background

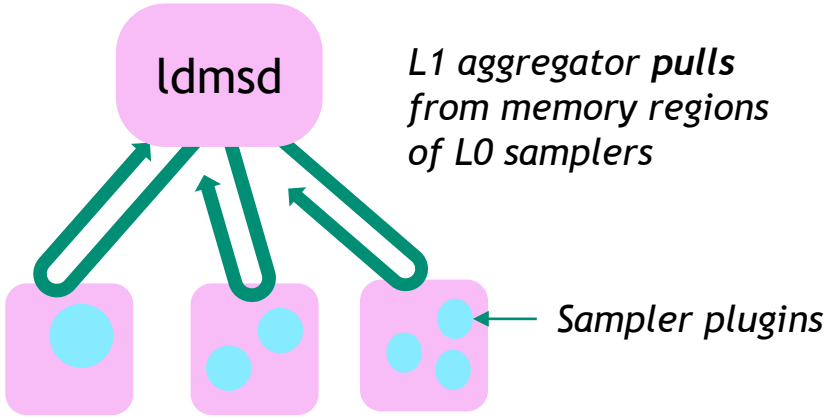
LDMS - low-overhead data collection, transport, and storage capability designed for continuous monitoring supporting runtime analytics and feedback.

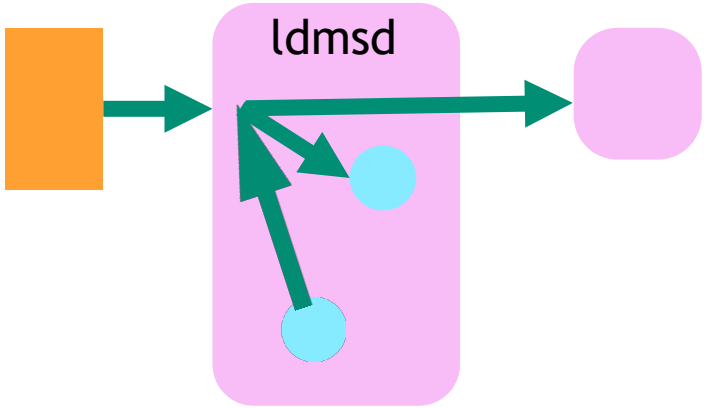
- LDMS transport is low-overhead (e.g., RDMA vs typical message bus IP)
- System data collection is typically synchronous at regular (e.g., second or less) intervals
- Structured data format (i.e., metric set) designed to minimize data movement
- Transport is typically *pull* based to minimize CPU interference, but also supports *push* based for asynchronous structured data

**GOAL: Leverage the efficient and secure LDMS transport to support Application Data Injection**

LDMS Streams – **on demand publication** of loosely formatted information to subscribers

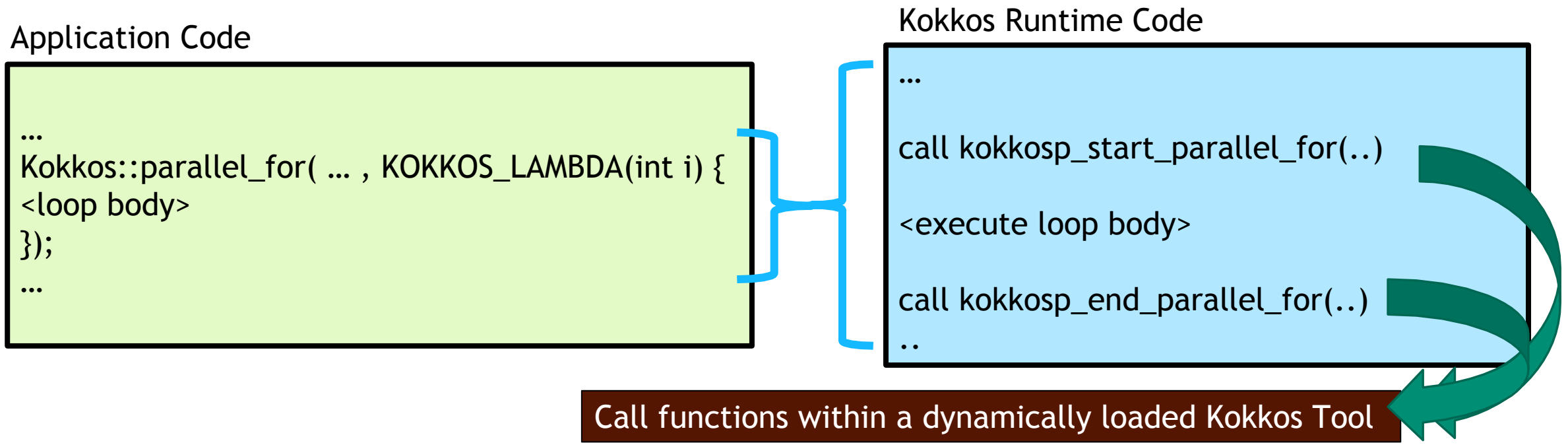
- Transport is push based and supports asynchronous event data (e.g. scheduler and log data)
- Unstructured data
- Leverages all features of the LDMS transport (e.g., security, RDMA)



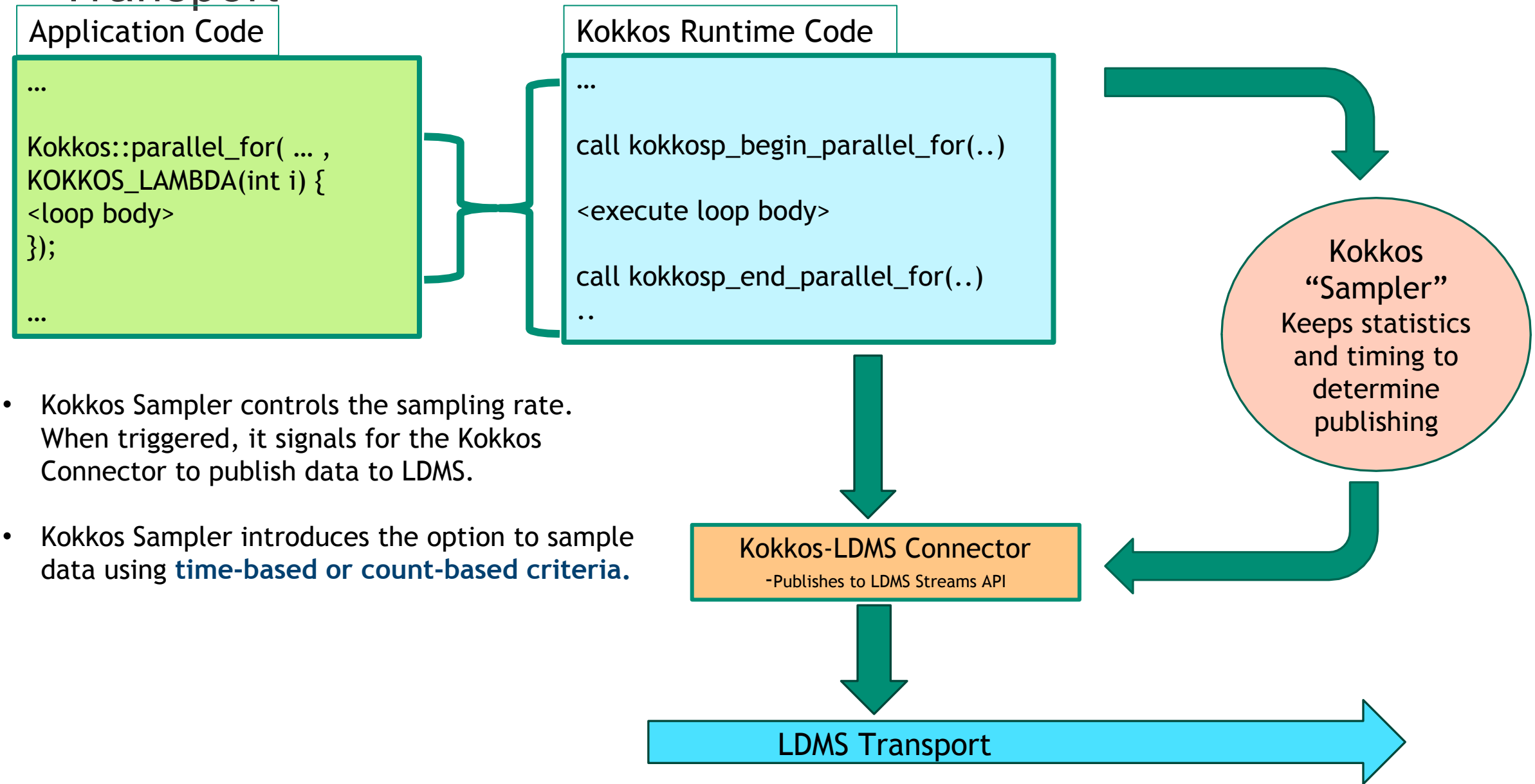

*Daemon publish API called from externally or by a plugin **pushes** to ldmsd which pushes to all subscribing plugins and aggregators*

# Kokkos **Performance Portability Layer**: Background




- Kernels and Teuchos timers within Trilinos are configured to **dynamically load a Kokkos supplied "connector"**. This requires **no recompilation** for profile enabled code and can be used for any Kokkos application (not just Trilinos, EMPIRE, etc.)
- Hook points already exist for **kernels** (parallel-for, reduce, scan), **"regions"** (arbitrary points in code which can stack) and **"sections"** (arbitrary points in code which may overlap)
- Already have a good idea of what the **valuable profiling information** would be (**doesn't require user input**)

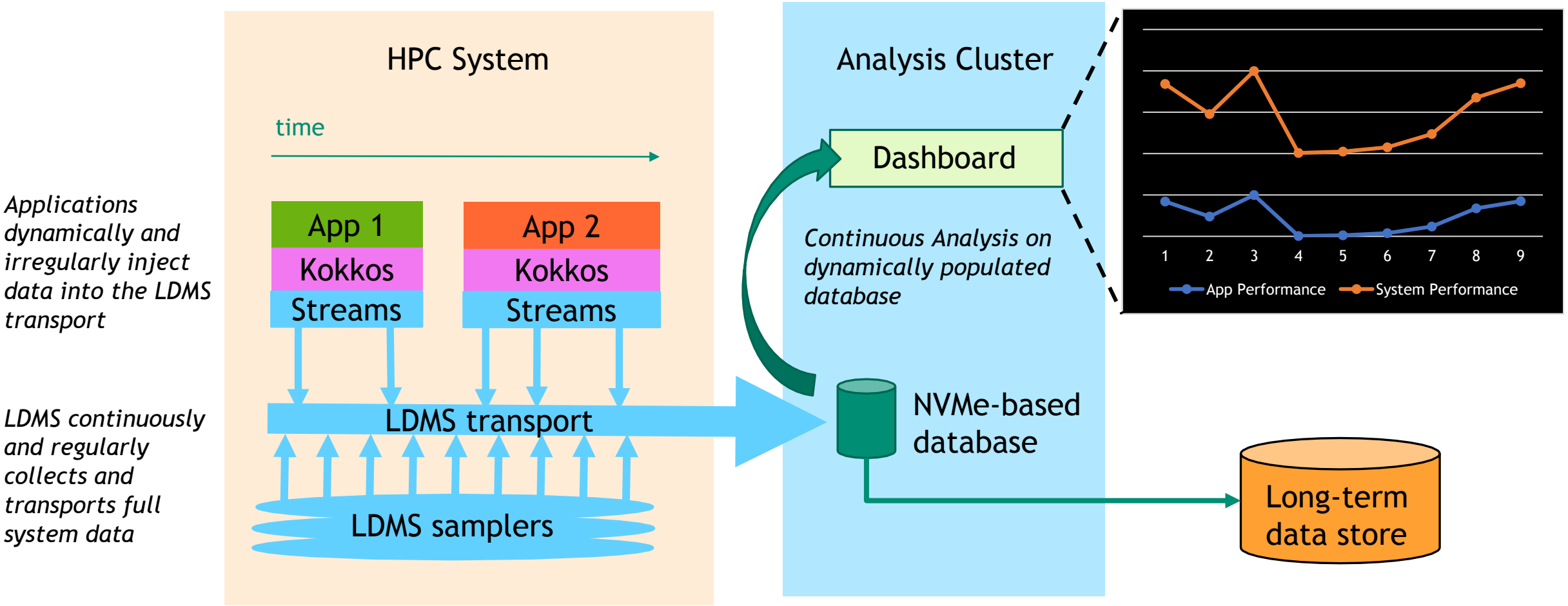
# Run time Injection of Application Data into the LDMS Transport




- Kokkos Sampler controls the sampling rate. When triggered, it signals for the Kokkos Connector to publish data to LDMS.
- Kokkos Sampler introduces the option to sample data using **time-based or count-based criteria.**

# Unified Framework for **Continuous, Run Time, Fused** System and Application Performance Monitoring and Analysis

Data Flow Diagram

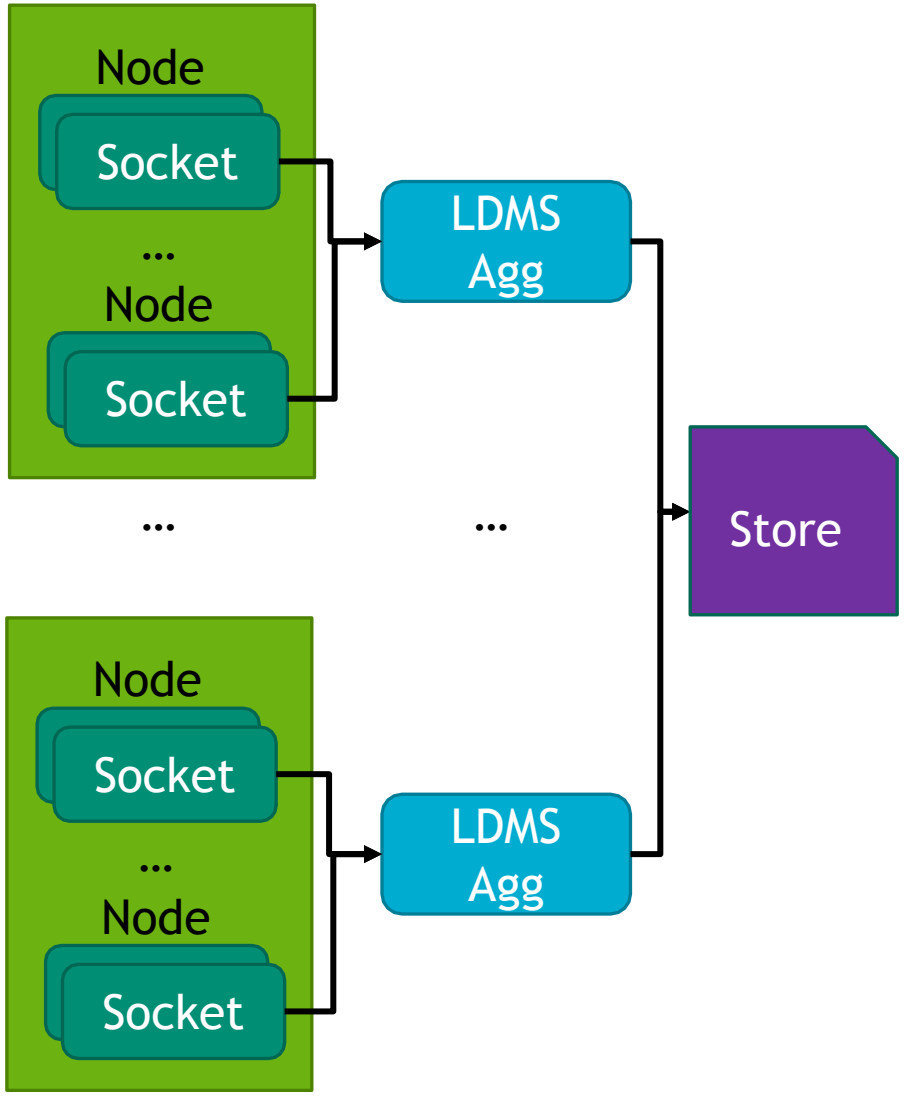



*Applications dynamically and irregularly inject data into the LDMS transport*

*LDMS continuously and regularly collects and transports full system data*

# Application and LDMS Configuration

- Voltrino (Cray XC40) 54 nodes
- Target recording ~1% of kernel execution events (e.g., one or more instances of {kernel name, kernel executions count, time})
  - Provide **reasonable representation of execution behavior** while having **little instrumentation overhead** (can dial in whatever % desired)
- Format is currently JSON
  - Investigating unpacking performance effects
- Store will eventually be a distributed database.
  - Currently CSV




**Recorded information per message:**

```
rank,timestamp,job-id,kokkos-perf-data:time,kokkos-perf-data:type,kokkos-perf-data:name,kokkos-perf-data:count
```

```
0,100907.012310,8290750,0.000003,0,"Kokkos::View::initialization [Kokkos::Random_XorShift64::state]",2
0,100907.012360,8290750,0.000008,0,"Kokkos::View::initialization [DualView::modified_flags]",5
0,100907.012400,8290750,0.000014,0,"Kokkos::View::initialization [SurfCollide:nsingle]",4
```

# Characterizing Impact of Network Traffic on Applications

Approach:

- Characterize application performance in the context of network congestion
  - Utilize the GPCNeT application to **create an environment with varying network interference** on an application
  - Historically **NO Network Hardware Performance Counters** have been shown to provide strong correlations between the “level of congestion” that an application is experiencing (e.g., stall to flit ratios, percent time stalled) and the progress that it is making (e.g., number of kernel executions per second for a given phase). There are **weak correlations at best.**

Challenge:

- Acquiring a **tangible and realistic** run time **representation of network congestion** that can be utilized to gain understanding (quantify) of how it affects application performance for any given application and input deck.

# Characterizing Impact of Network Traffic on Applications

Our current solution is to utilize **ms level fidelity** data to perform **run time comparisons** of:

- Approximate network congestion from statistically sampling latencies among an application's compute nodes utilizing ping-pong packets and/or Cray Aries latency counters.
- Approximate application progress using a statistical sampling of kernel calls

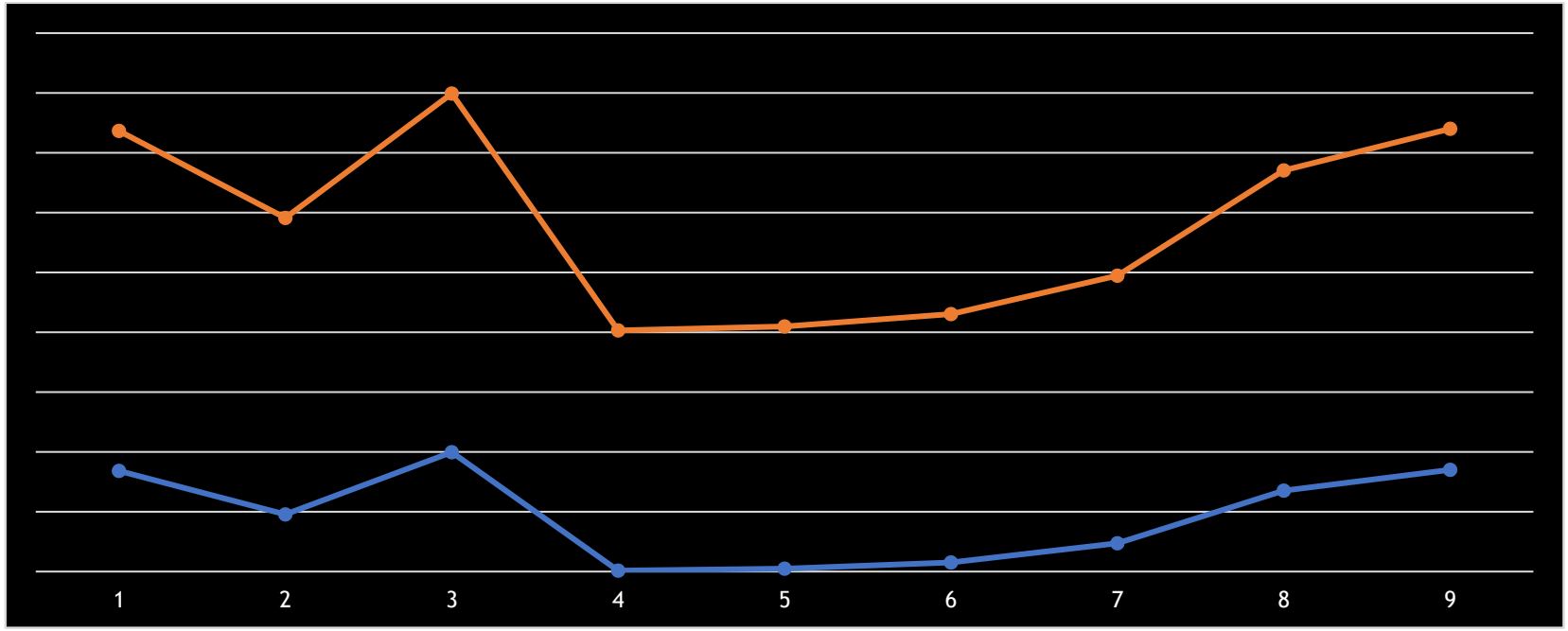



Fig. 1: Comparing application progress (blue) with evolving latency (system state (orange))

# Analysis and Visualization: Architecture

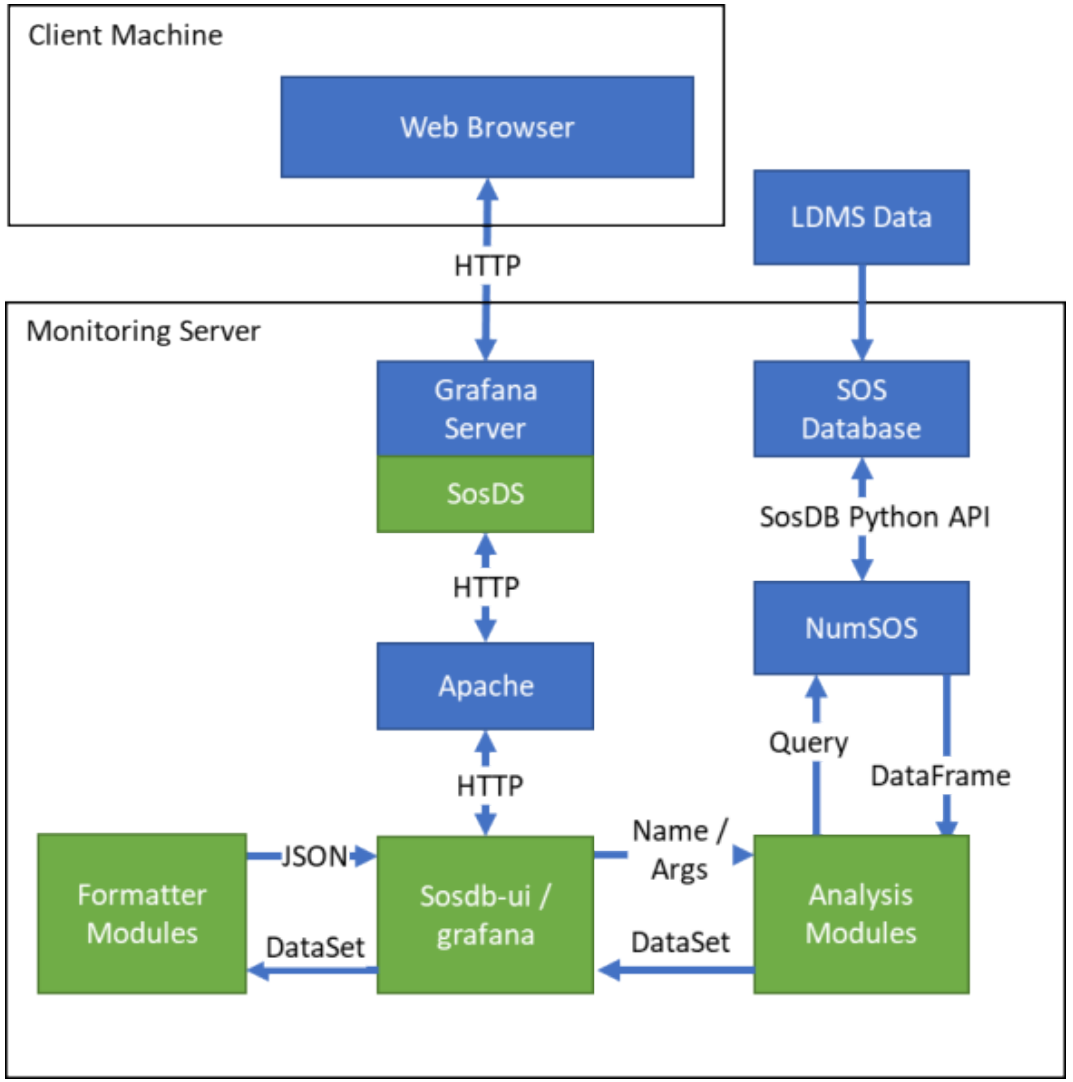
Grafana interface for analyzing and visualizing integrated application and system data

Data can be directly queried from a database or have a python module perform analyses alongside queries

- Allows for **flexible development of visualizations** as **analysis only happens during a query** rather than over all data
- Any user with appropriate permissions can add and change analytics and create their own queries/analyses

```
def _get_max(self,metrics):
    where_ = [ [ 'timestamp', Sos.COND_GE, self.start ] ]
    if self.end > 0:
        where_.append([ 'timestamp', Sos.COND_LE, self.end ])
    self.src.select(['timestamp'] + metrics,
                    from_ = [ self.schema ],
                    where = where_,
                    order_by = 'time_job_comp'
        )

    self.xfrm = Transform(self.src, None, limit=self.mdp)
    resp = self.xfrm.begin()
    while resp is not None:
            resp = self.xfrm.next()
            if resp is not None:
                    self.xfrm.concat()
    self.xfrm.mean(metrics)
    data = self.xfrm.pop()
    df = pd.DataFrame(data.tolist(), columns=metrics)
    res = DataSet()
    res.append_array(len(df.columns), 'metric', df.columns)
    res.append_array(len(df.iloc[0]), 'max', df.iloc[0])
    return res
```



# Conclusions

System and Application data fusion will provide **unprecedented run time insight** into performance and resource utilization features

**BUT** there are challenges:

- **Fidelity** (are we correctly characterizing the resource) **and completeness** (do we have coverage of connections in our sampling (e.g., is a sampling of 1% of connections sufficient?)) of latency measurements
- Meaningful and useful **congestion metrics**
- Meaningful and useful **application progress/performance metrics**
- Application phases change (do we have enough data points in each phase to characterize the application overall?)
- **Application** of these insights and capabilities **to other shared resources**
- Data format to support desired ingest rates
- **Features** of interest we expect to resolve **are at 10s of ms to a few seconds** (e.g., our supported system state data rate)