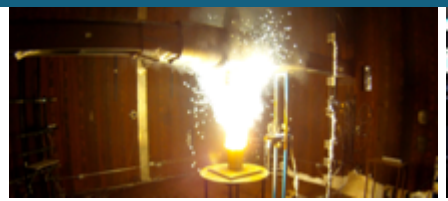
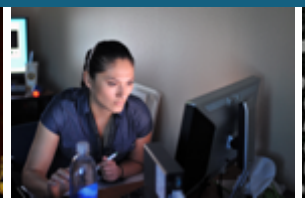
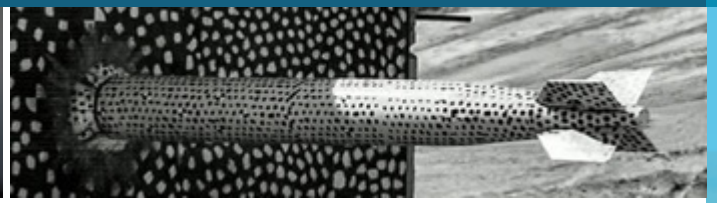
SAND2021-4475C

Sandia National Laboratories

Enabling Application and System Data Fusion







Ann Gentile, SNL
representing “Integrated System and Application Continuous Performance Monitoring and Analysis Capability” Team

ECP Annual Meeting 4/2021

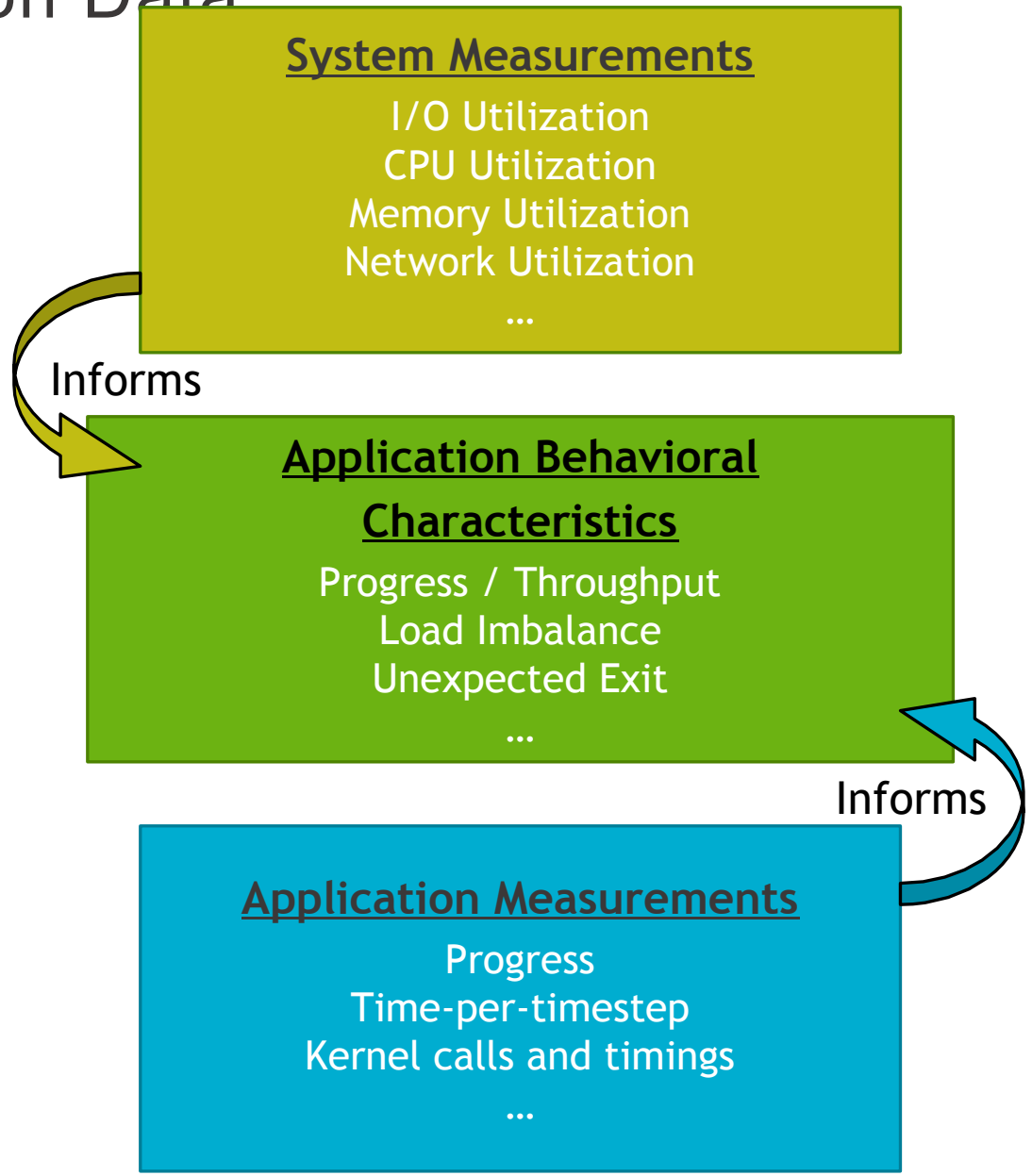
Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy’s National Nuclear Security Administration under contract DE-NA0003525.

Answering Key Operational Questions via Integrated System and Application Data

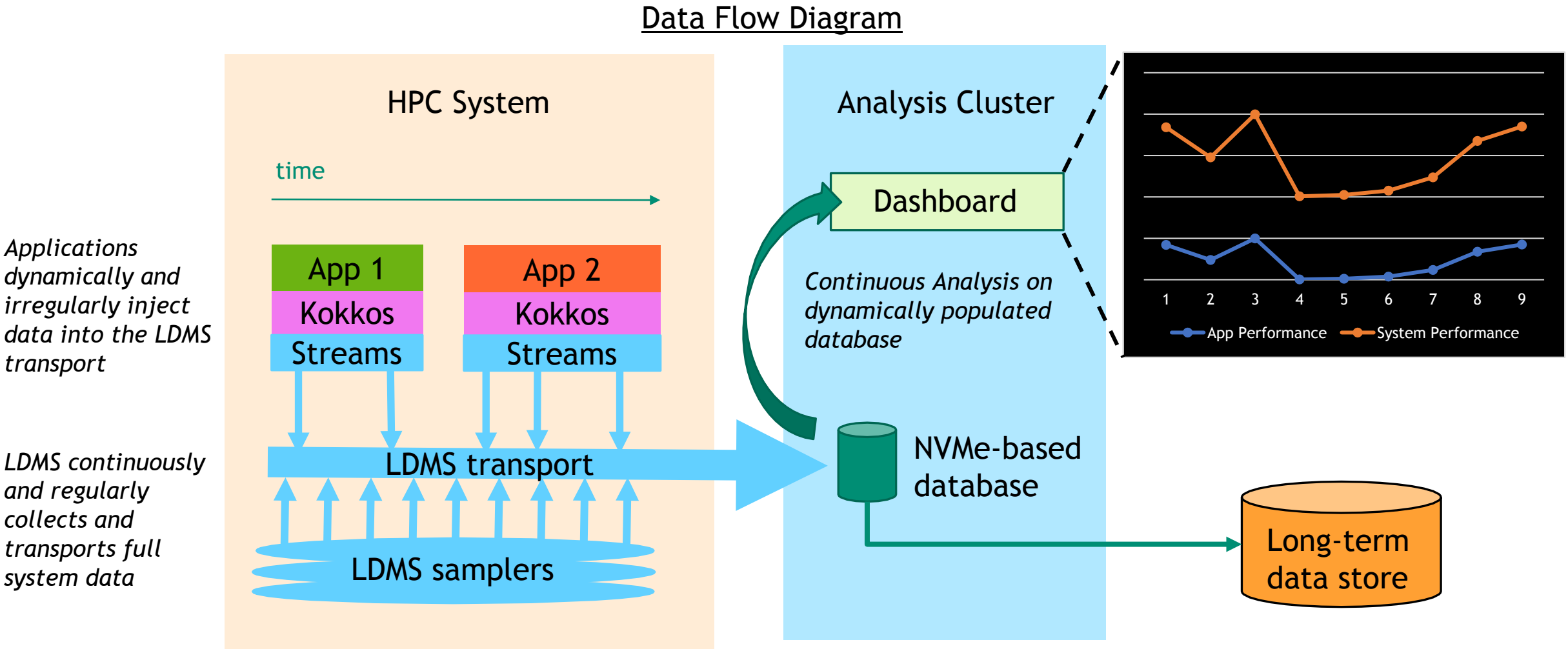
Is performance variation due to system conditions or code changes?

What are the architectural requirements given the site's workloads?

- Resource state (e.g., utilizations, faults/failures, contention, performance limits) affect application performance.
- Integrating system and application measurements can provide insights into application behaviors:
  - Is an application's progress change associated with a simultaneous change resource state?
  - Can load imbalance among ranks in the application be associated with inconsistent utilization (e.g., CPU, memory) across nodes?
  - Can examination of application progress information narrow location (e.g., special, temporal) of behavior of interest?
  - Are our applications in contention for resources?



Integrated System and Application Continuous Performance Monitoring and Analysis Capability



Enabling Application Data Injection: LDMS Background

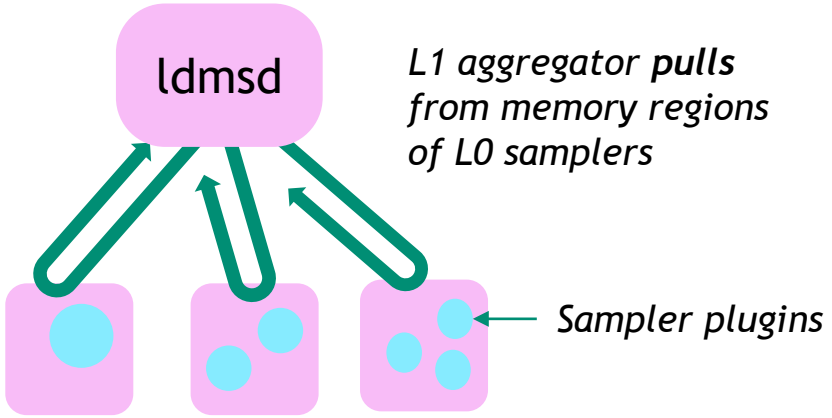
LDMS - low-overhead data collection, transport, and storage capability designed for continuous monitoring supporting runtime analytics and feedback.

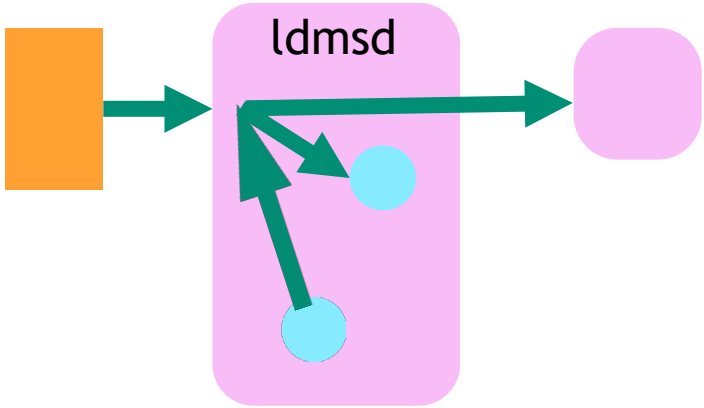
- LDMS transport is low-overhead (e.g., RDMA vs typical message bus IP)
- System data collection is typically synchronous at regular (e.g., second or less) intervals
- Structured data format (i.e., metric set) designed to minimize data movement
- Transport is typically *pull* based to minimize CPU interference,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

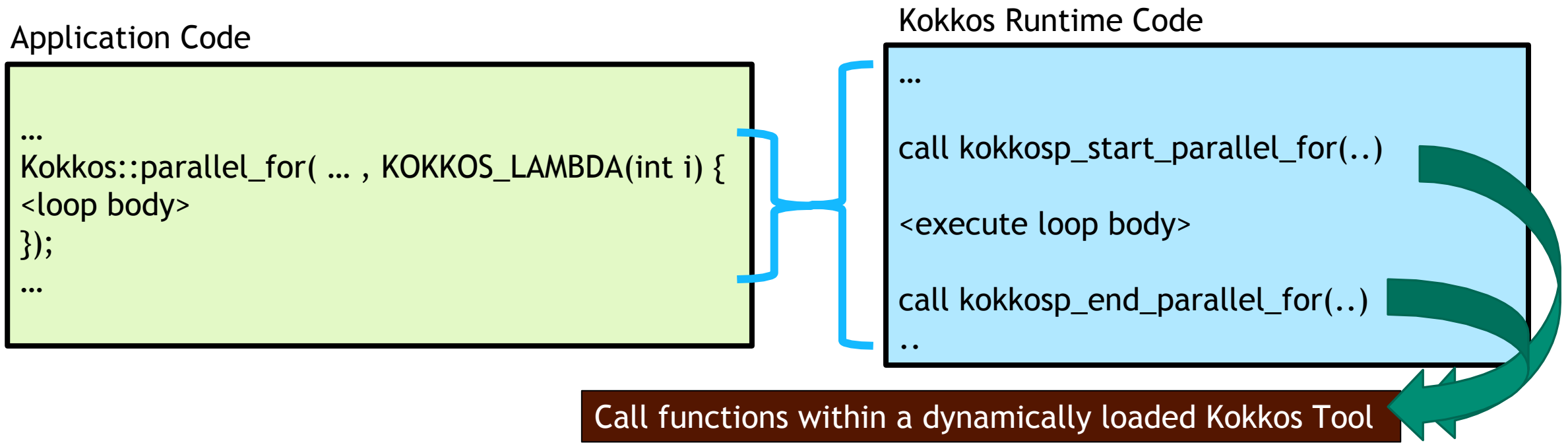
- Transport is push based and supports asynchronous event data (e.g. scheduler and log data)
- Unstructured data
- Leverages all the features of the LDMS transport (e.g., security)





*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

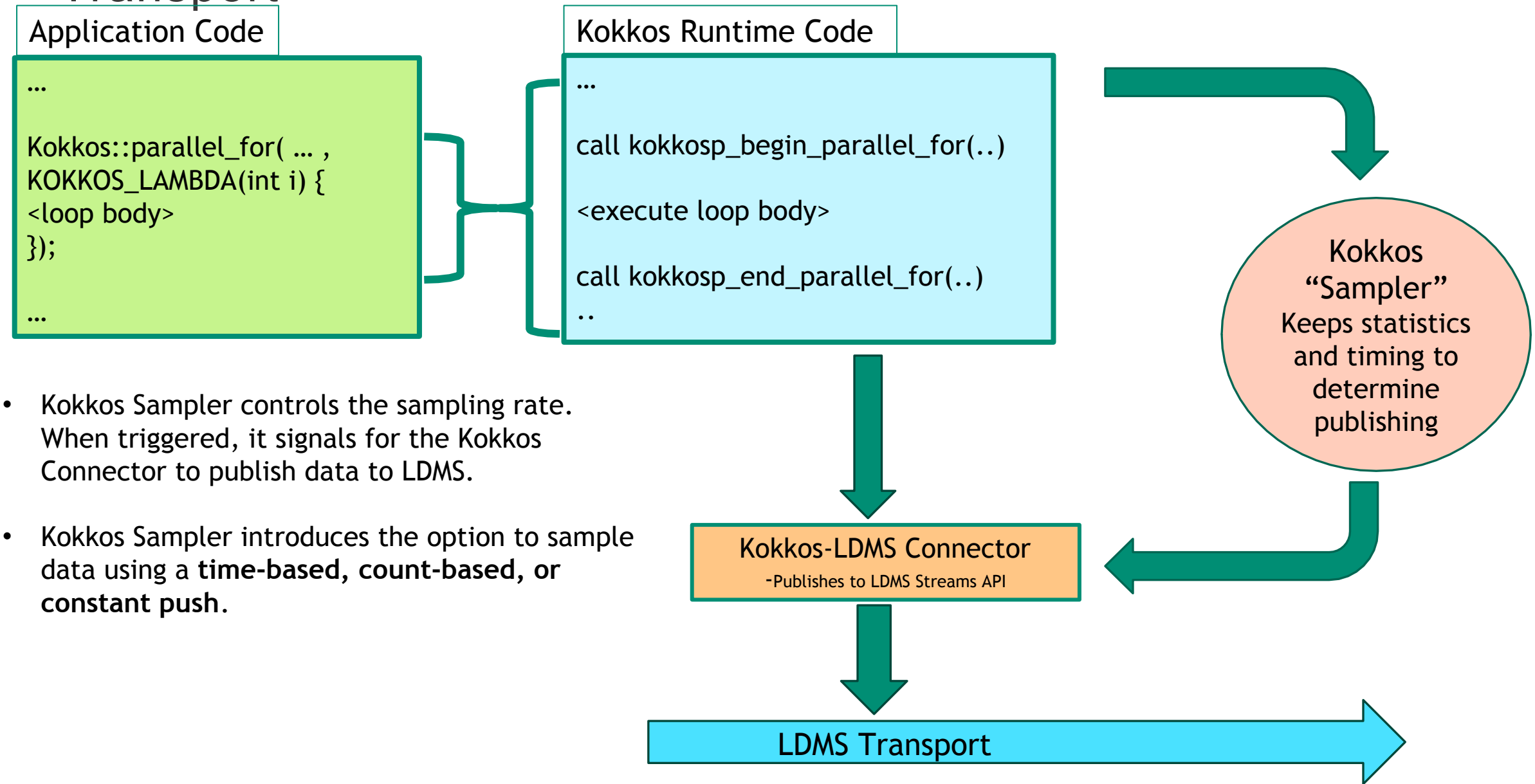


Kokkos Tool External Connectivity - kernels and Teuchos timers within Trilinos are configured to dynamically load a Kokkos supplied “connector”. This requires **no recompilation** for profile enabled code and can be used for any Kokkos application (not just Trilinos, EMPIRE etc)

Kokkos Tool Internal Connectivity – 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)

Profiling – 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

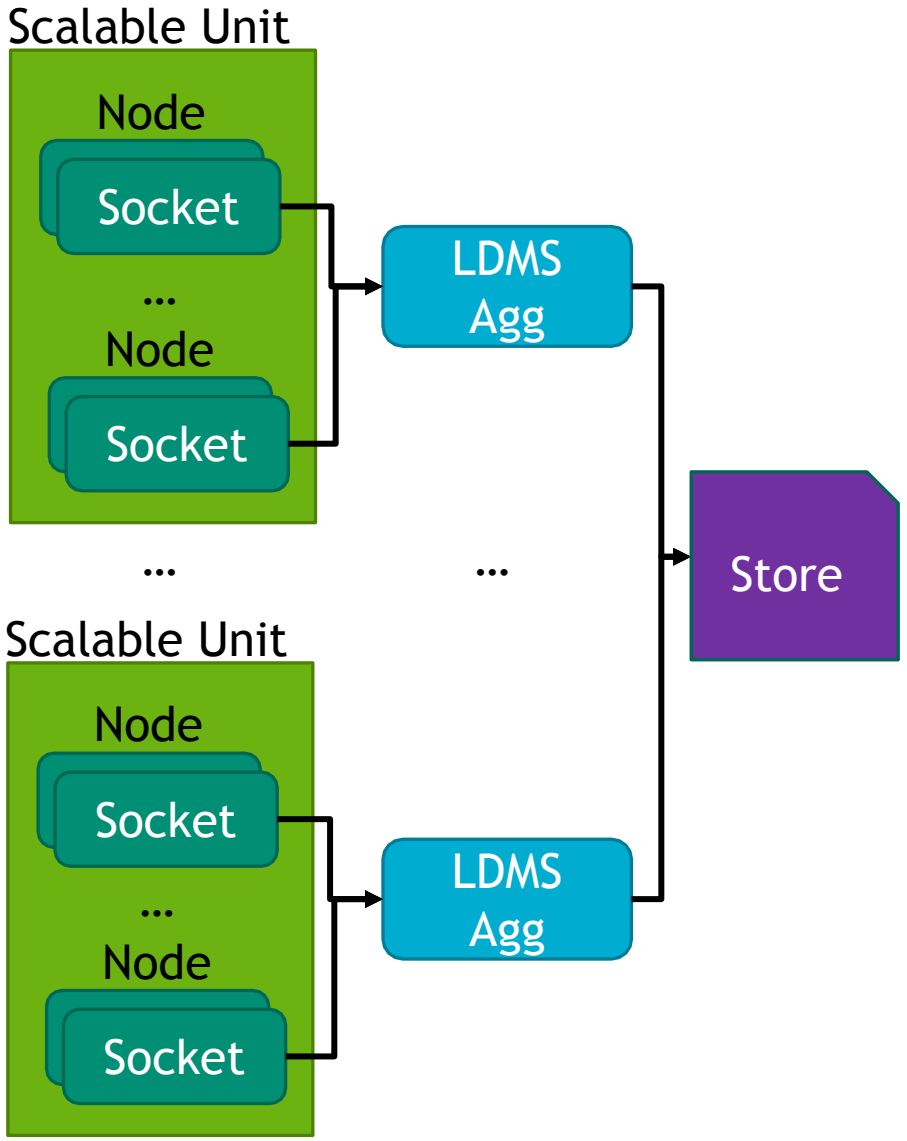


- Kokkos Sampler controls the sampling rate. When triggered, it signals for the Kokkos Connector to publish data to LDMS.
- Kokkos Sampler introduces the option to sample data using a **time-based, count-based, or constant push**.

Application and LDMS Configuration

- Eclipse (CTS1) ~1500 Nodes
- Code target (EMPIRE) configuration: 1 rank/socket
- 1 first level aggregator/SU
- Target recording 1 publication per 10ms per rank with 1000 multi-part data values (e.g., {kernel name, count, time})
- Format currently JSON, currently investigating unpacking performance effects
- Store will eventually be distributed database. Currently CSV

```
#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
```

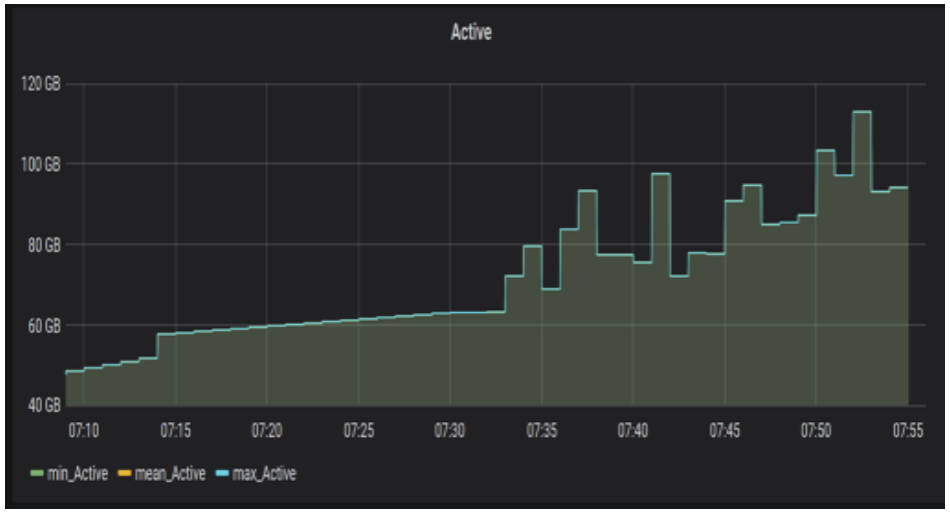


Analysis and Visualization Plan

Plot derived application performance metrics in conjunction with relevant system performance metrics to produce dashboards that can provide a fused view of system performance and application performance

- Create Python analyses to parse application performance data and derive throughput and performance metrics
  - Hot data retention policy is two weeks
- Create queries and dashboards that make intuitive sense to visualize the derived application metrics

System Visualization



Timeseries plot of active memory of an application

Application Visualization

Timeseries plot of function/second for specific miniEM function by rank

Analysis and Visualization: Architecture

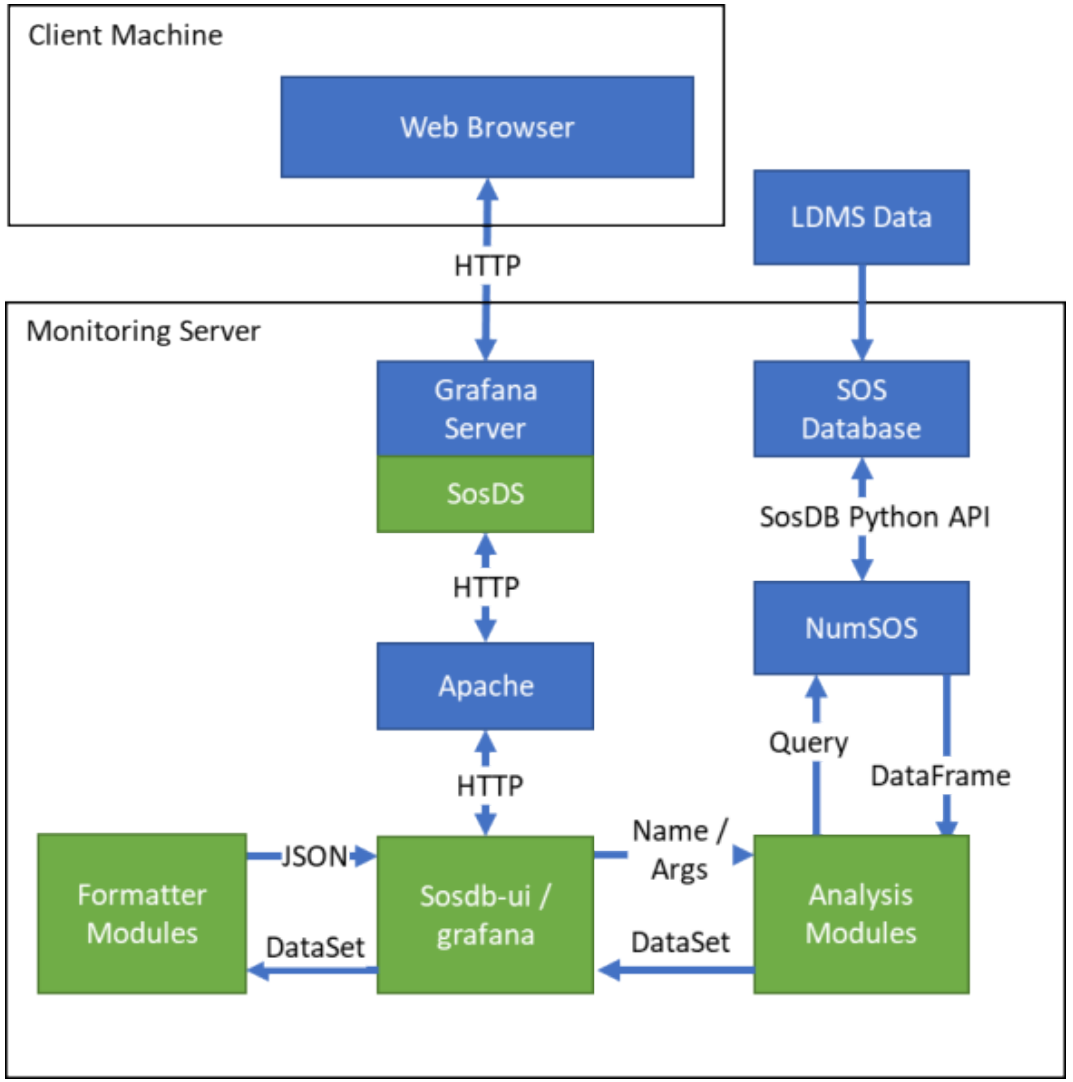
Grafana interface for analyzing and visualizing integrated application and system data

Data can be directly queried from a database or have a python module perform analysis alongside the query

- 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 with analysis

```
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
```



Current Work

- System and Application data fusion will provide unprecedented run time insight into performance and resource utilization features
- Challenges:
  - Data format to support ingest rate
  - Features of interest we expect to resolve are at 100ms-few seconds (e.g., our supported system state data rate)
  - Data and events of interest to applications and from the system
  - Meaningful application progress and system state visualizations
  - Statistical and ML techniques
- Upcoming Scale Run and Initial Data Set: 24 hr code run ~300 nodes