Caliper:
Putting Performance Data in Context

9th Scalable Tools Workshop

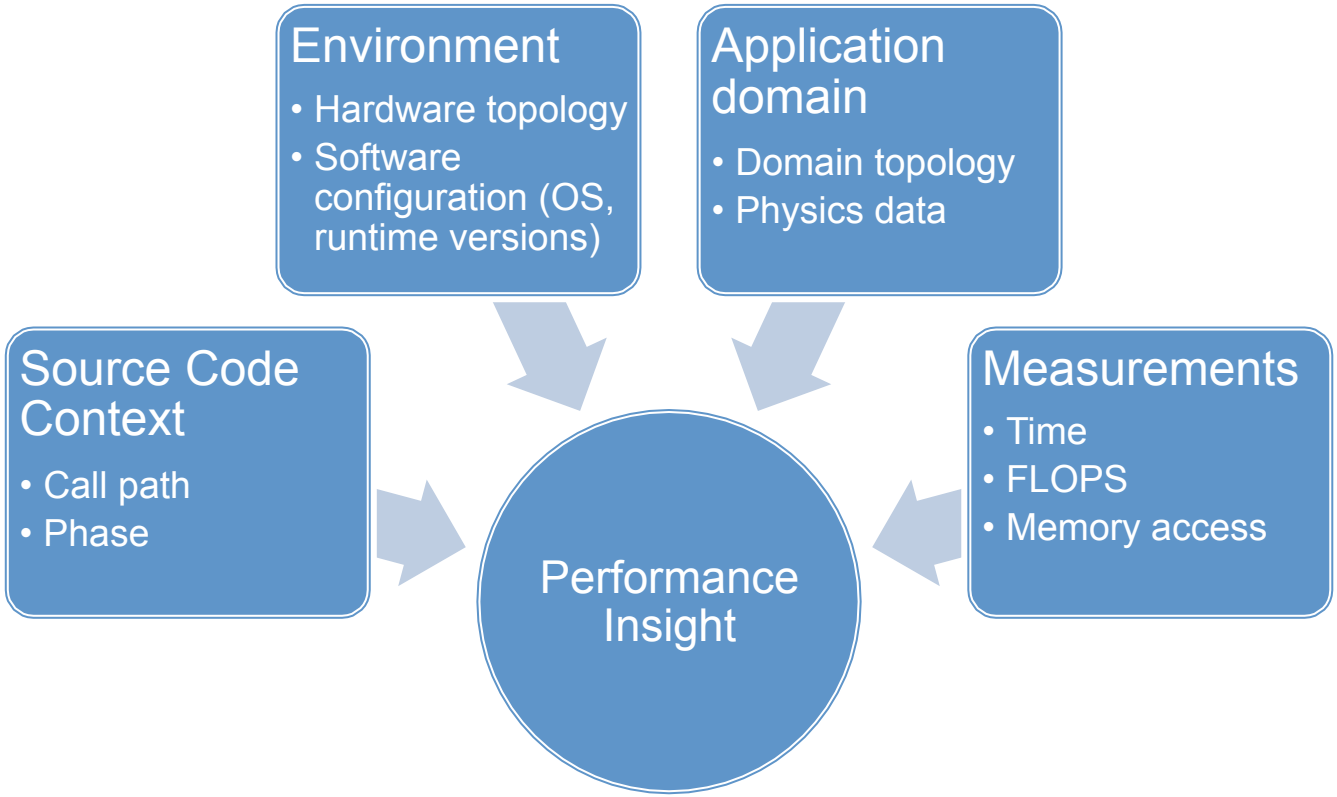
David Boehme
Todd Gamblin
Martin Schulz

August 3, 2015

LLNL-PRES-675459
This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract DE-AC52-07NA27344. Lawrence Livermore National Security, LLC

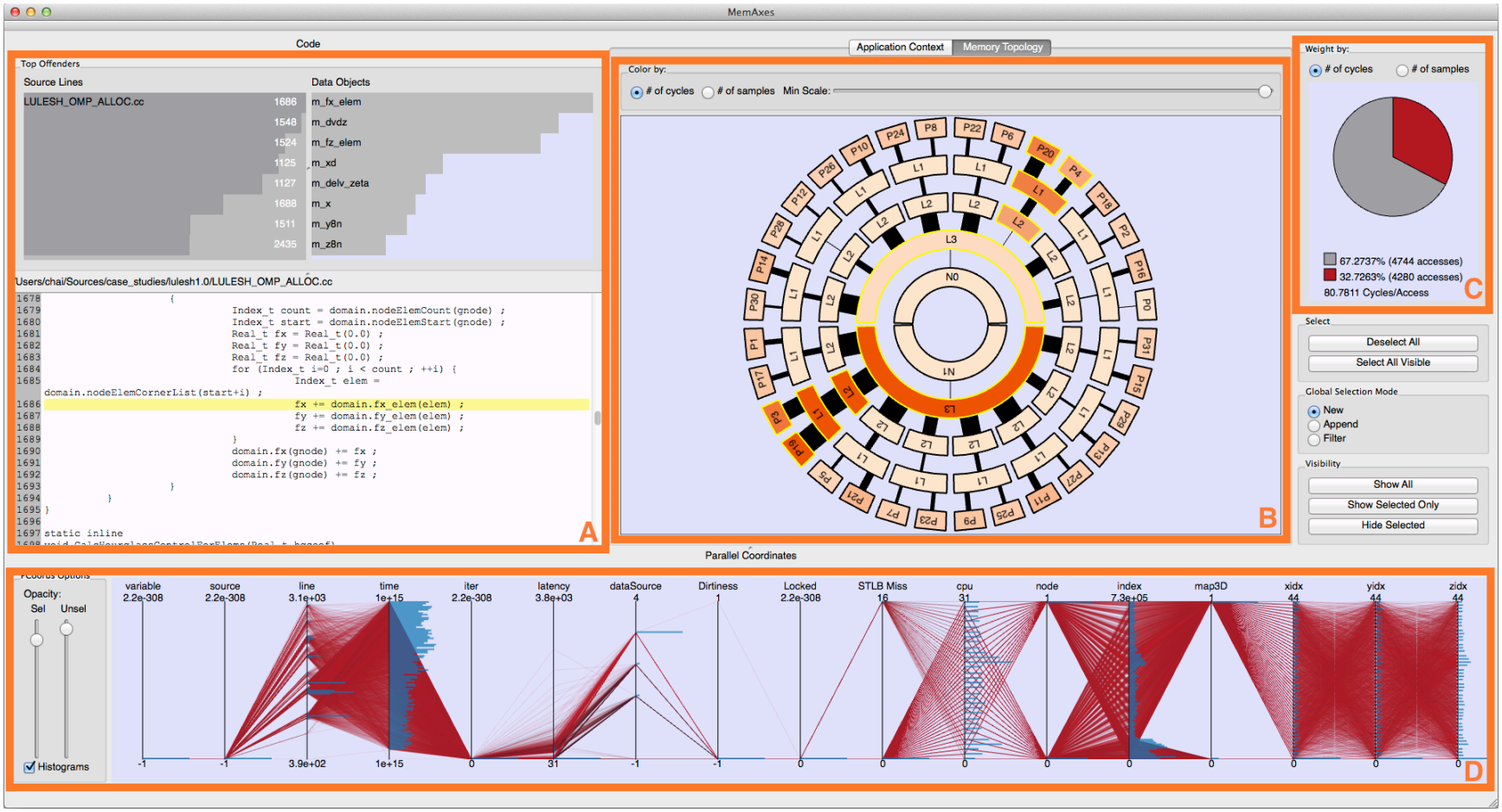
Lawrence Livermore National Laboratory

Performance Analysis requires Data Correlation



No interface to collect generic context data!

Example: Visualization in MemAxes

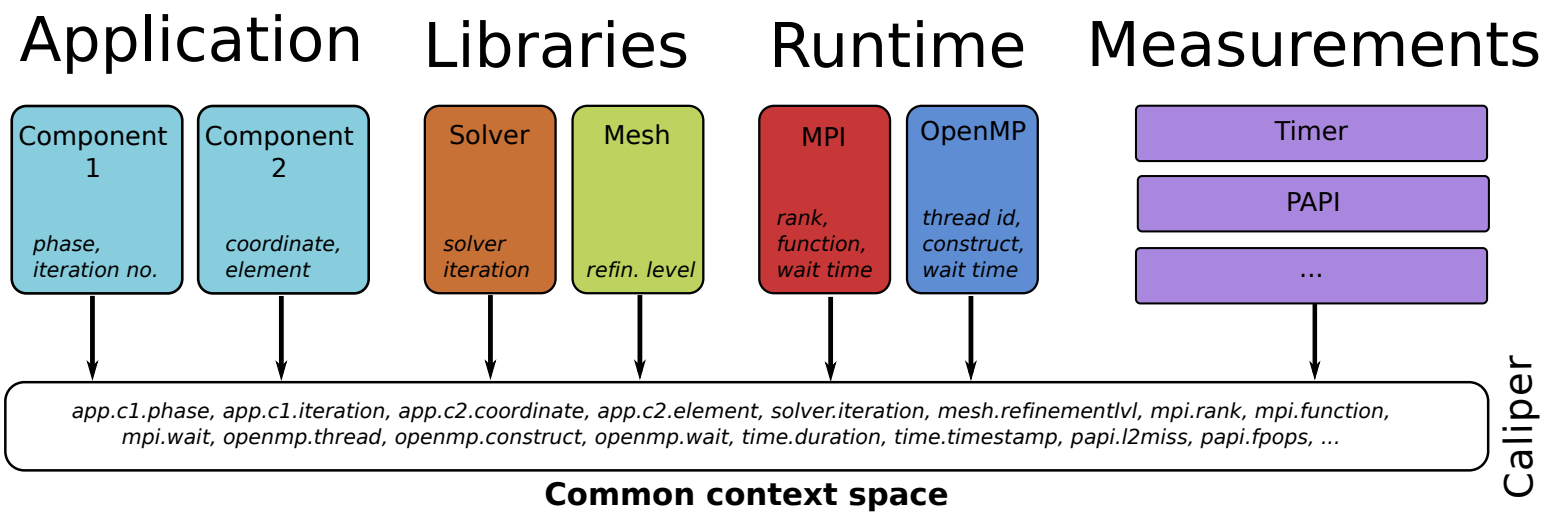


Caliper Contributions

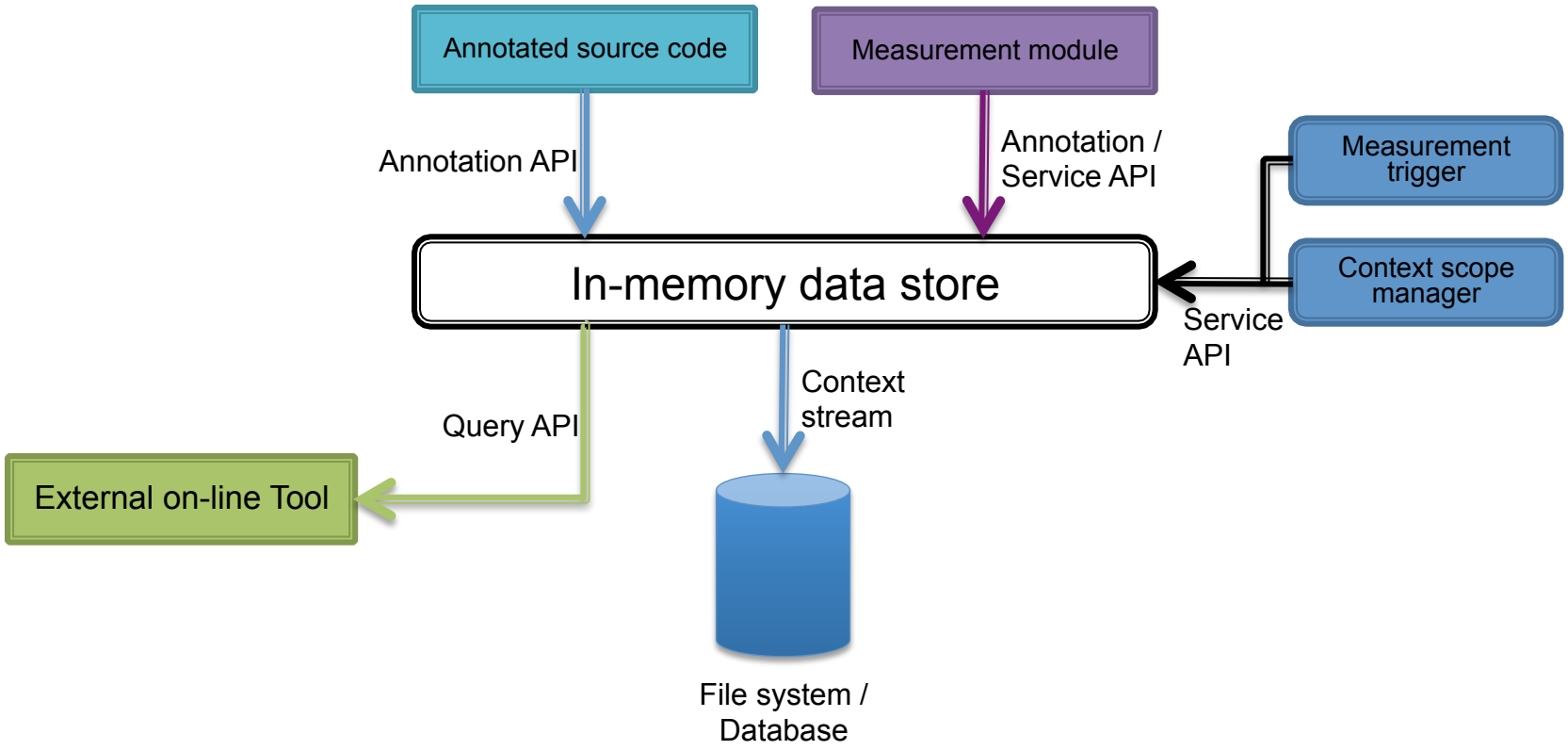
1. Interface for **applications** to provide arbitrary context information
2. Interface for plug-in tools that provide **measurement** data
3. Enable **composition** of context annotations and measurement providers

Composite Data Collection

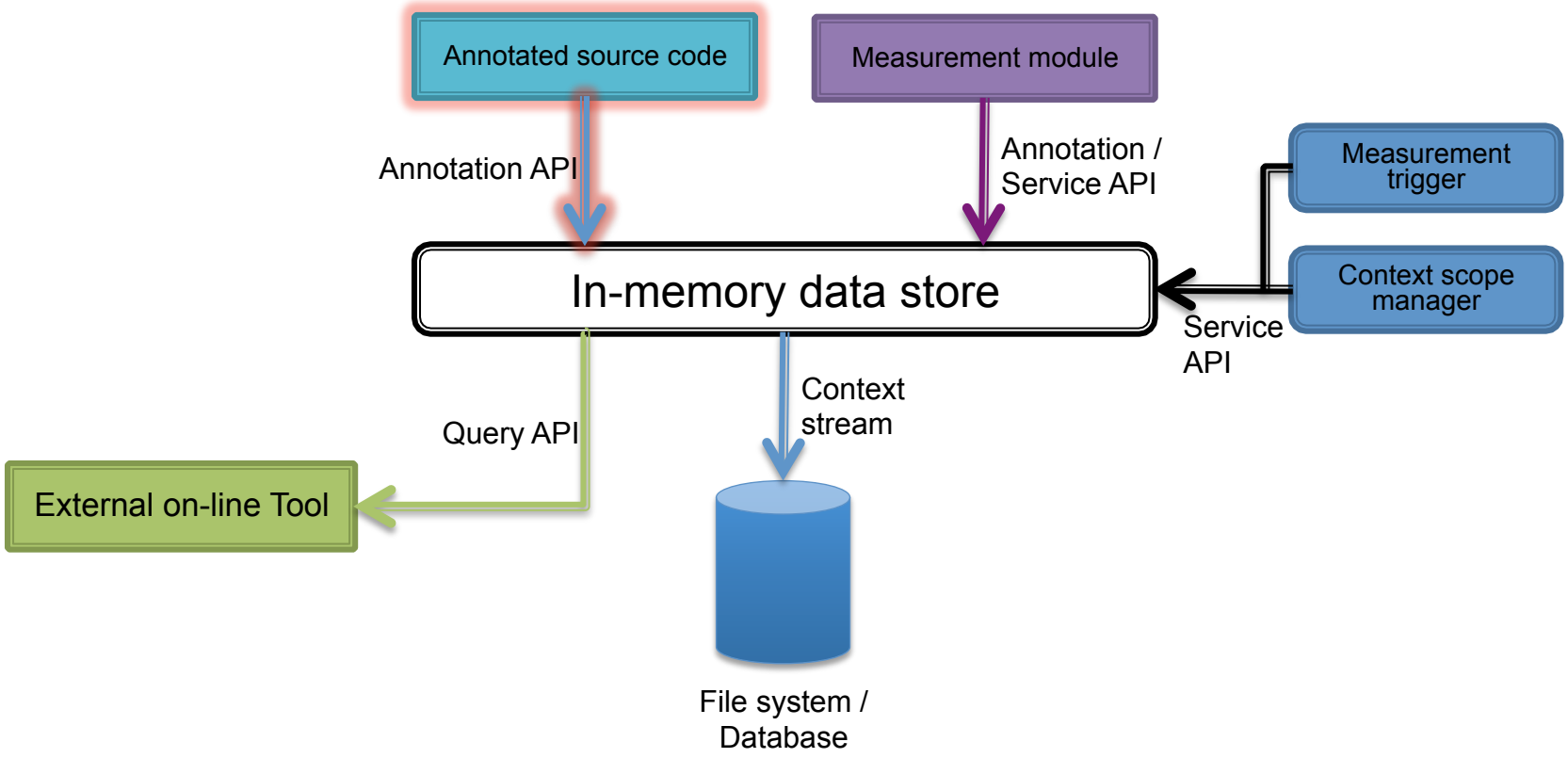
- Generic *attribute:value* data model
  - Allows storage of any type or kind of data: not limited to pre-defined context categories
- In-memory data store
  - Data sources update context / measurement data *independently*
  - Combines data across the software stack



Caliper Framework

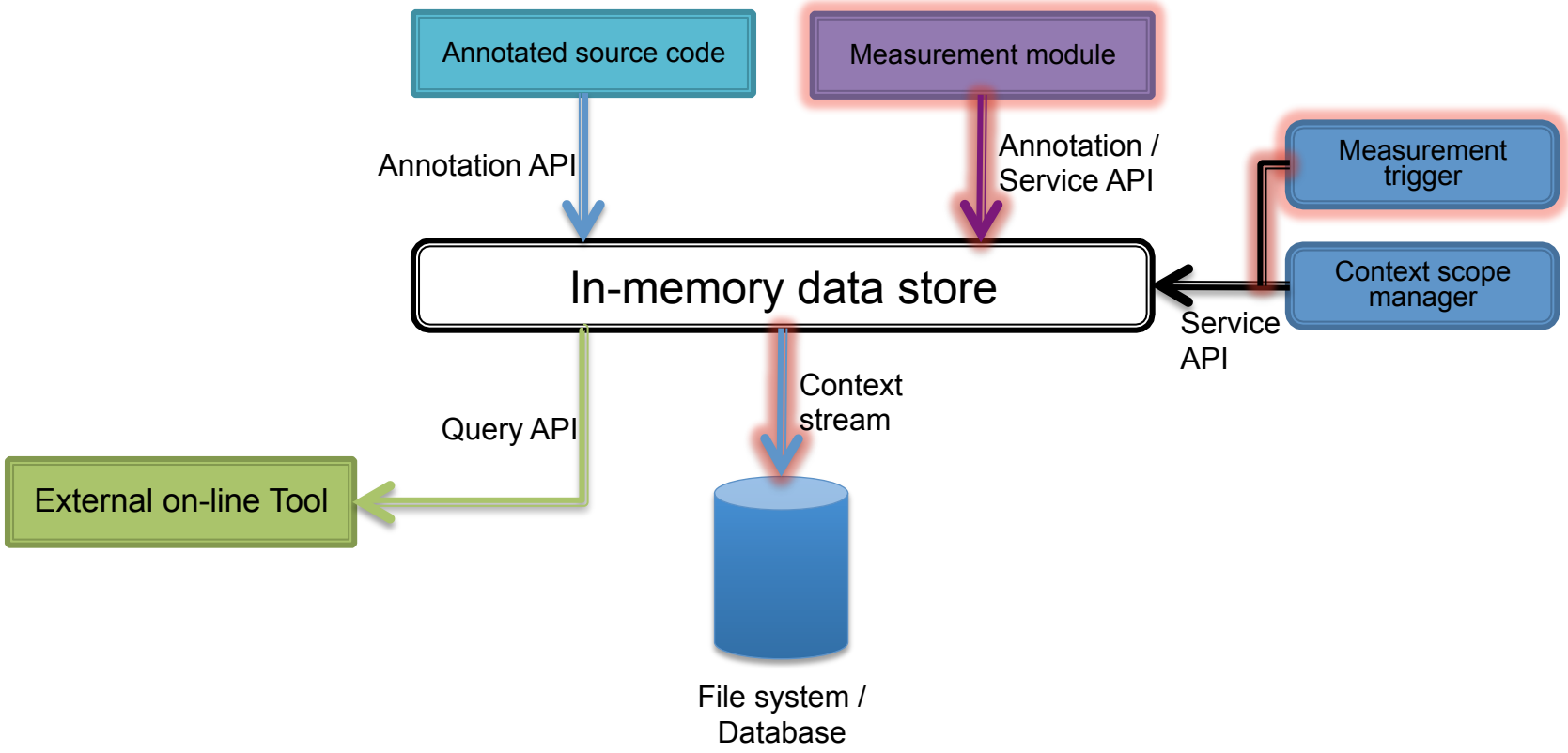


Caliper Workflow (1)



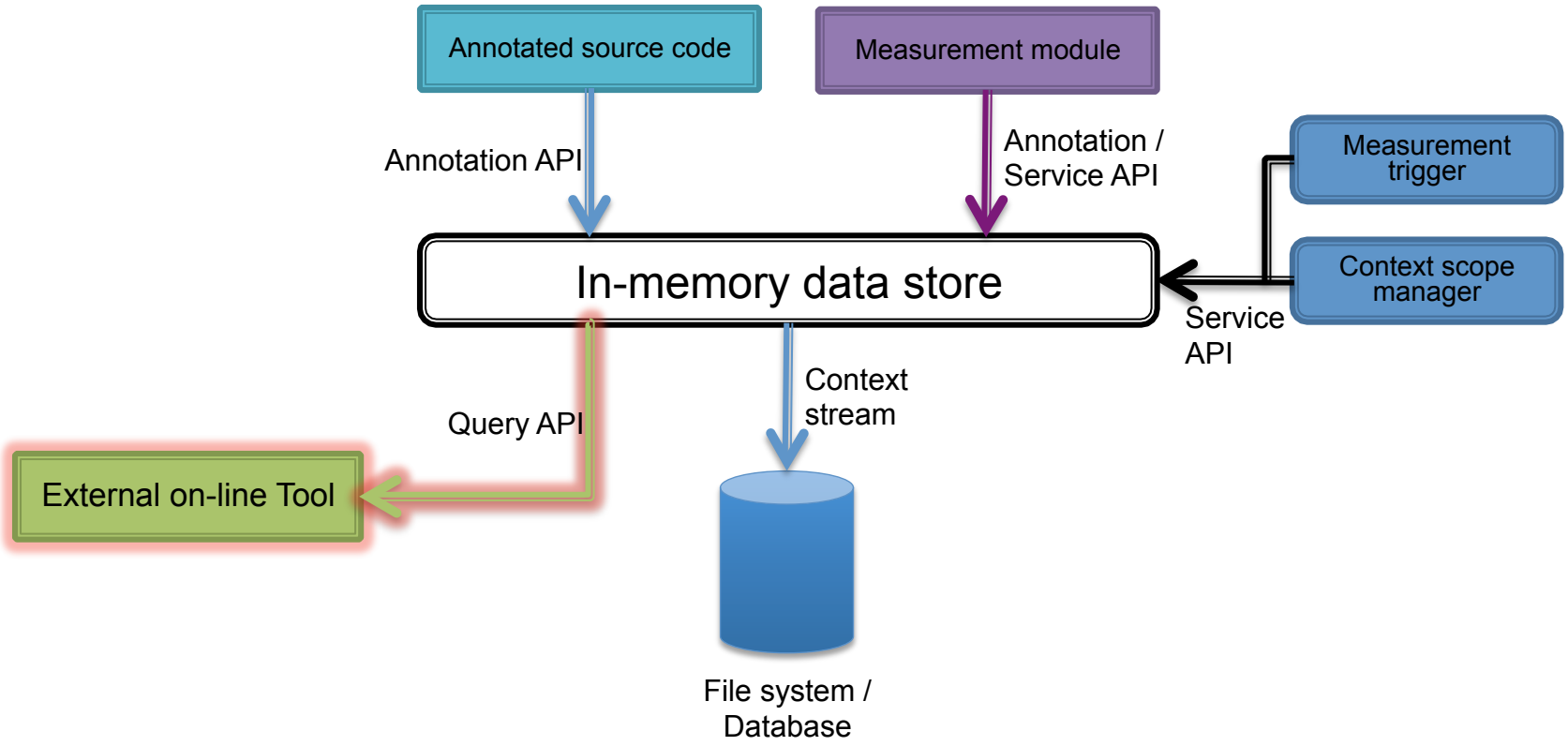
- Instrumented modules independently update context information

Caliper Workflow (2)



- Trigger creates record with *context snapshot* and measurement data

Caliper Workflow (3)



- Alternatively, external tool pulls context and measurement snapshot on-line

Data Model

- Fully flexible *attribute:value* format

  ```
  app.phase="solve",mpi.rank=42,time.duration=1234,mesh.level=3
  ```

- Attributes contain
  - Unique name
  - Data type (integer, floating point, string, binary blob)
    - Future extension: JSON description for complex types
  - Scope (process, thread, or task)

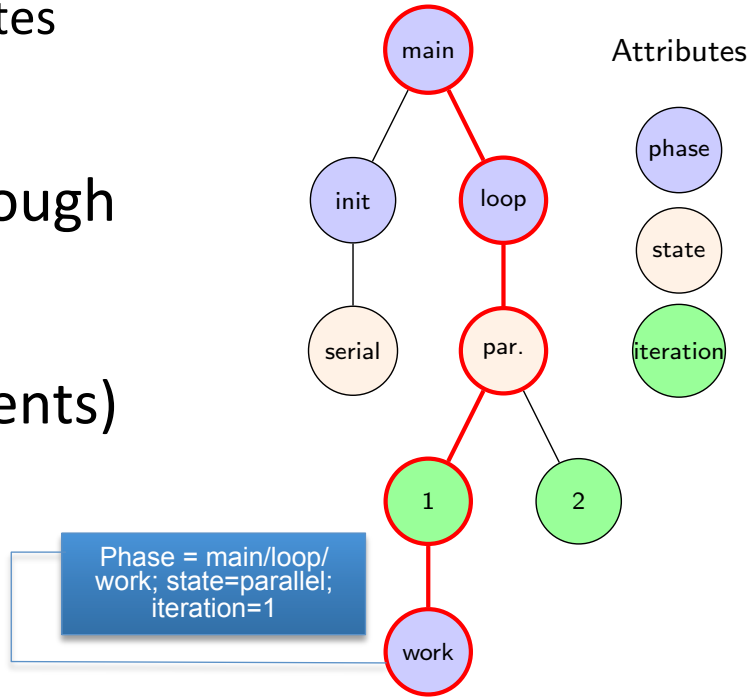
- Entries can be hierarchical (e.g., for call paths)

- Automatic scoping
  - Caliper keeps separate entries per thread or tasks

- Efficient tree-based data representation

Efficient Context Representation

- Build up *context tree*
  - Stores values from multiple attributes
  - Transparent to the user
- Represent context snapshot through single node
- Non-repetitive data (measurements) stored explicitly



Data Format

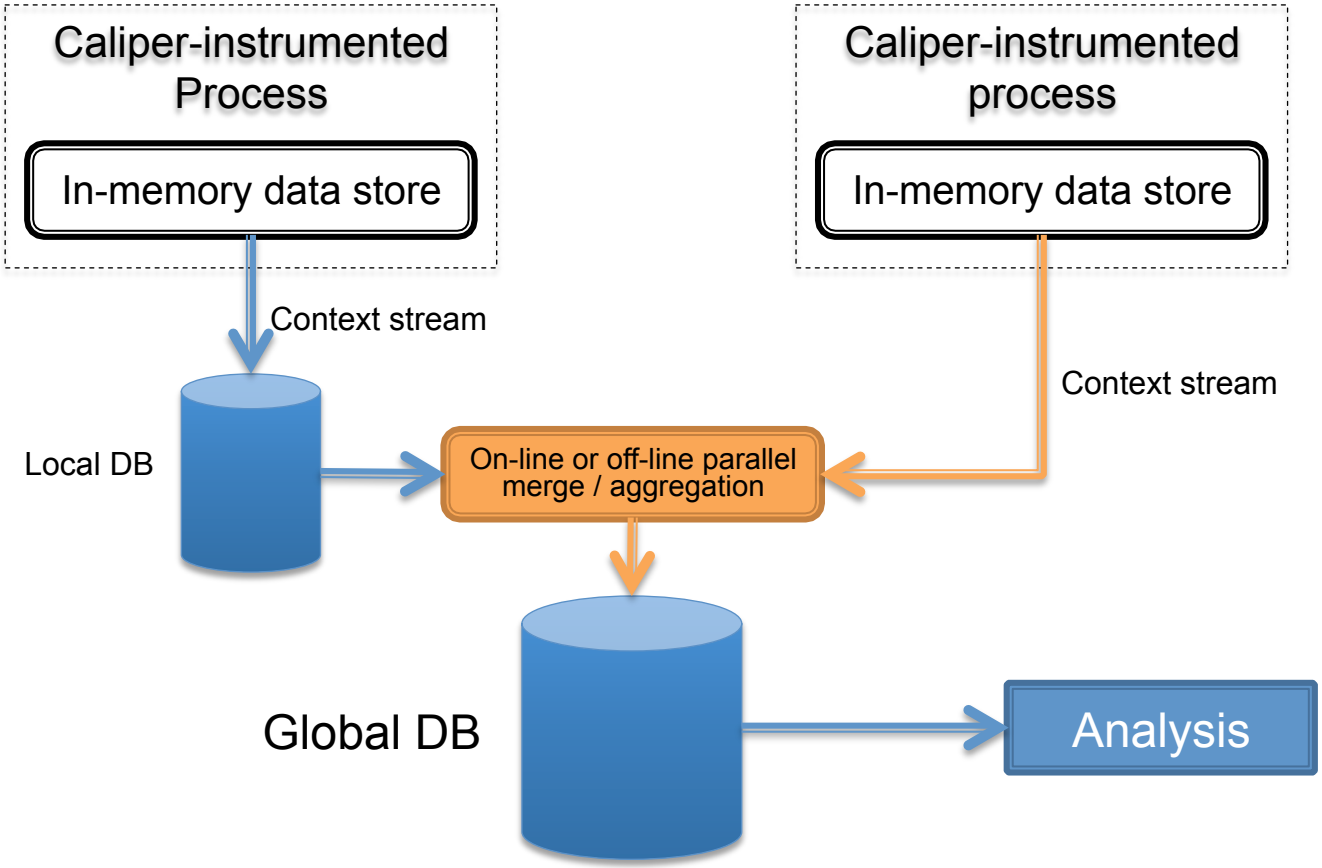
- **Context streams** include performance/context and metadata records of a single Caliper instance
- **Node records** describe context tree and metadata (attributes)

```
__rec=node,id=24,attr=8,data=iter,parent=23
```

- **Context records** combine context and immediate data entries of a context snapshot

```
__rec=ctx,ref=25,attr=19=24,data=13=1
```

Data Processing / Analysis Stack



Annotation API

- `cali::Annotation`
  - Encapsulates attribute
- `begin()`
  - Append new value
- `set()`
  - Set (overwrite) value
- `end()`
  - Remove last value

```
#include <Annotation.h>

int main(int argc, char* argv[])
{
    cali::Annotation phase_ann("phase");

    phase_ann.begin("init");
    // Perform initialization
    initialize();
    phase_ann.end(); // ends “init”

    phase_ann.begin("loop");
#pragma omp parallel for
   for (int i; i < MAX; ++i) {
        cali::Annotation("iteration").set(i);
        do_work(i);
    }
    phase_ann.end(); // ends “loop”
}
```

Service API

- Instrumentation + context query for measurement services and third-party tools
  - `push_context()`
    - Trigger snapshot and write to stream
  - `pull_context()`
    - Trigger and pull snapshot
  - `create_attribute()`
    - Creates attribute
  - `begin() / end() / set()`
    - Set values
  - (Query API) (not defined yet)
- Callback functions for various events
  - E.g. snapshot triggered, attribute created, value changed, …

Measurement Services

Service	
Timer	Timestamps and/or time duration
Callpath	Performs stack unwinding to retrieve call path
MPI	Wraps MPI functions and provides MPI rank
OMPT	OpenMP tools interface, provides thread ID, state, and OpenMP construct wrappers
PAPI	PAPI hardware counters

Usage (1): Configure, run

- Link caliper library
- Configure
  - Add measurement services, set output flags

```
$ export CALI_SERVICES_ENABLE=recorder:timestamp
```

- Run

```
$ ./test/cali-basic
== CALIPER: Registered recorder service
== CALIPER: Registered timestamp service
== CALIPER: Initialized
== CALIPER: Wrote 38 records.
== CALIPER: Finished
$ ls *.cali
150724-073336_479_ytM1by52l3yV.cali
```

Usage (2): Examine

- *cali-query* expands records and provides basic aggregation, filter, and merge functionality
- Export to your favorite data analytics / visualization tool

```
$ cali-query –e *.cali
                                  time.duration=221
                phase=main        time.duration=140
                phase=main/init   time.duration=15
                phase=main        time.duration=12
                phase=main/loop   time.duration=18
iteration=0     phase=main/loop   time.duration=15
iteration=1     phase=main/loop   time.duration=7
iteration=2     phase=main/loop   time.duration=6
iteration=3     phase=main/loop   time.duration=10
                phase=main/loop   time.duration=13
                phase=main        time.duration=7
```

Ongoing Work

- On-line aggregation
  - Turns Caliper into a “real” profiler
- Complex datatypes
  - Describes layout of binary blobs
- Scalable cross-process on-line merge / aggregation
  - Use MRNet

Further Information

- Available on ~~github~~ LLNL LC Stash:

  https://lc.llnl.gov/stash/projects/PIPER/repos/caliper/browse

- BSD License
- Release version available soon
- Contact:

  David Boehme
  boehme3@llnl.gov