Easing the complex with

By Adriano Marques Garcia

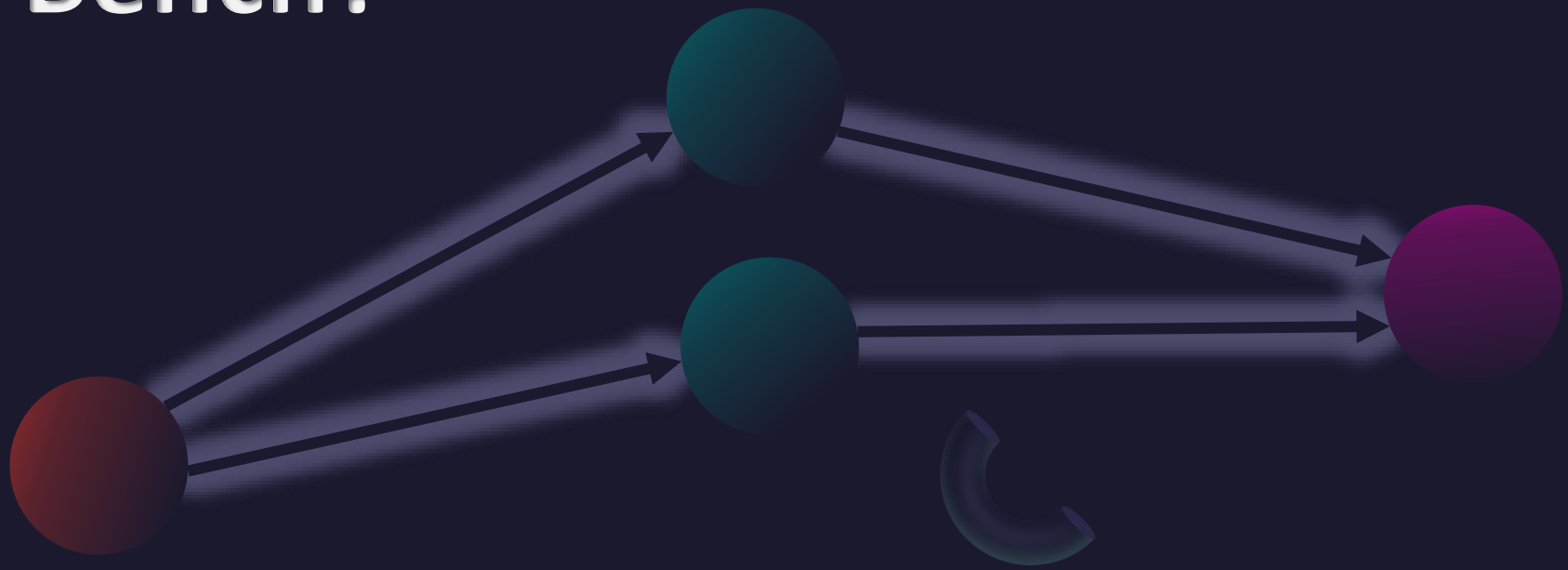
Stream Processing Benchmark

A framework for streamlining benchmarking of C++ stream processing

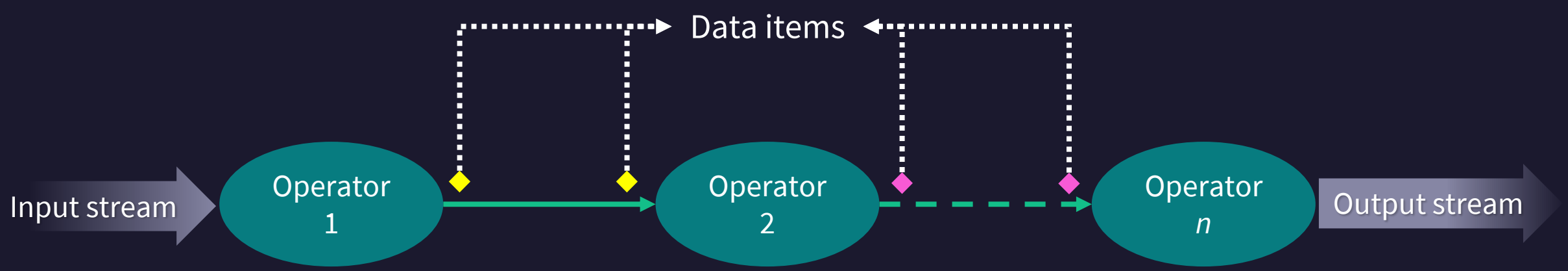
Stream Processing Benchmark

It provides a set of real-world stream processing applications for the C++ community in a high-level, reusable, and cleaner abstraction so that users can with minimal effort build custom benchmarks for evaluating different approaches or technologies in this context.

Why SPBench?



Basic concepts



To implement stream processing users must split the original application into multiple operators and identify all data dependencies.

What are the operators?

```
DIR *pd = m_dir_stack.top();
struct dirent *ent = NULL;
int res = 0;
struct stat st;
int path_len = strlen(m_path);
ent = readdir(pd);
if (ent == NULL) {
          closedir(pd);
          m_path[m_path_stack.top()] = 0;
          m_path_stack.pop();
          m_dir_stack.pop();
          if(m_dir_stack.empty()) {
                    stream_end = true;
                    break;
          }
}
strcat(m_path, ent->d_name);
res = stat(m_path, &st);
if (res != 0) {
          perror("Error:");
          stream_end = true;
          break;}
if (S_ISREG(st.st_mode)) {
          ret = file_helper(m_path);
          m_path[path_len]=0;
          input_found = true;
} else if (S_ISDIR(st.st_mode)) {
          m_path[path_len]=0;
          push_dir(ent->d_name);
} else
          m_path[path_len]=0;
ret->second.seg.name = ret->first.load.name;
ret->second.seg.width = ret-
>first.load.width;
ret->second.seg.height = ret-
>first.load.height;
ret->second.seg.HSV = ret->first.load.HSV;
image_segment((void**)&ret->second.seg.mask,
          &ret->second.seg.nrgn,
          ret->first.load.RGB,
          ret->first.load.width,
          ret->first.load.height);
free(ret->first.load.RGB);
cass_query_t query;
query = item.query_batch[num_item];
ret->second.vec.name = ret->extract.name;
query.flags = CASS_RESULT_LISTS |
CASS_RESULT_USERMEM;
ret->second.vec.ds = query.dataset = &ret-
>extract.ds;
query.vecset_id = 0;
query.vec_dist_id = vec_dist_id;
query.vecset_dist_id = vecset_dist_id;
query.topk = 2*top_K;
query.extra_params = extra_params;
```

?

?

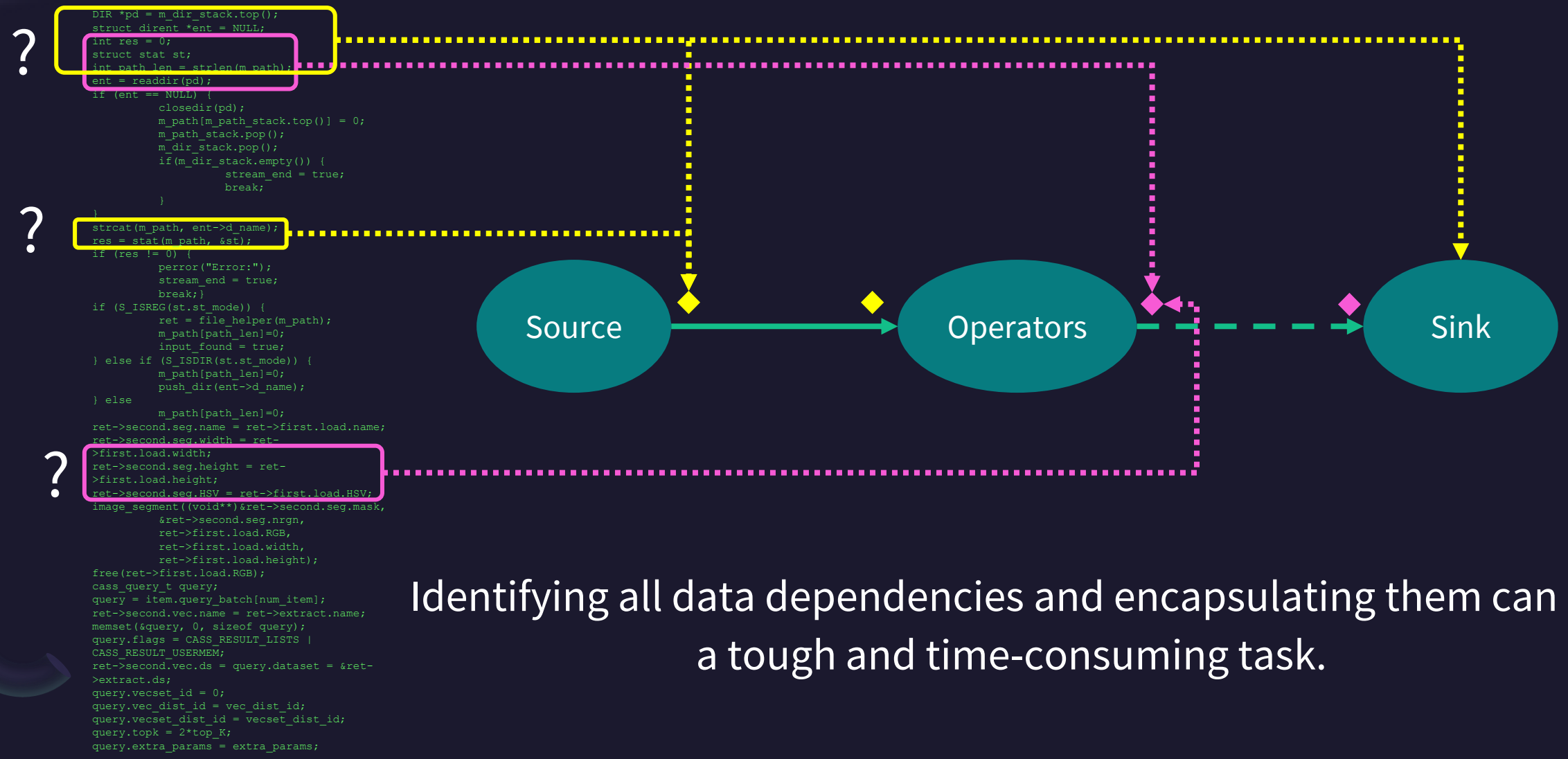
?

It may not be easy to find the beginning and end of an operator. These operators can range from a few lines up to hundreds or thousands lines of code.

What are the operators?

In SPBench, specific low-level details of each application are abstracted away, and it presents the user with the core of each application in a few lines of code.

What are the data items?



Identifying all data dependencies and encapsulating them can be a tough and time-consuming task.

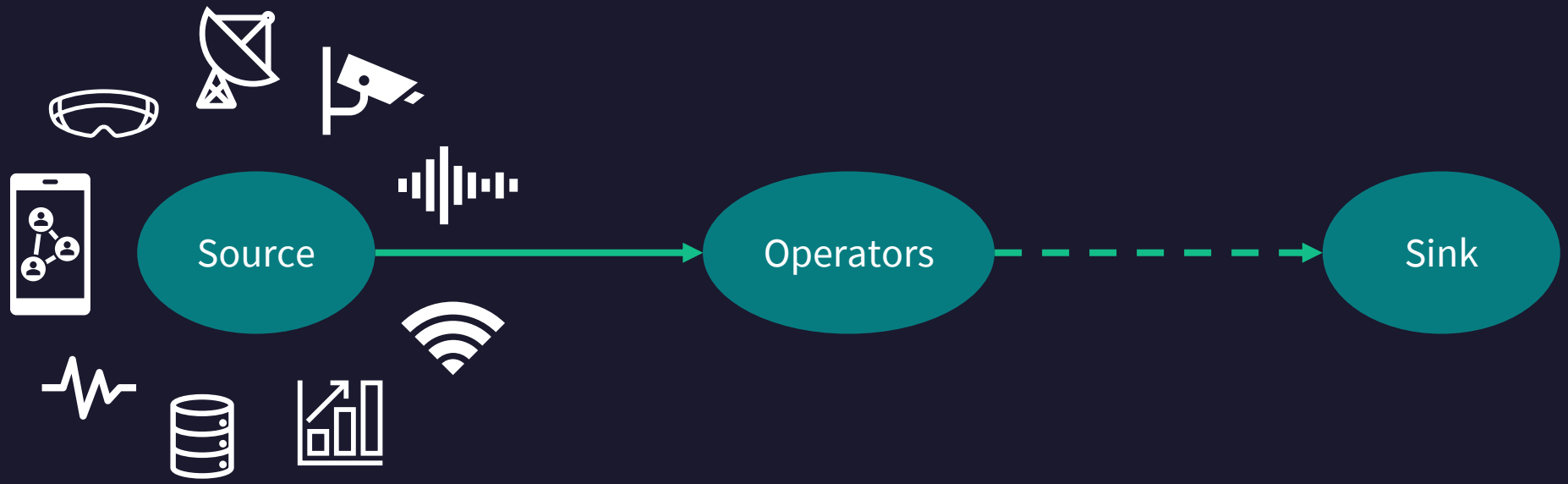
What are the data items?

```
While(/*condition*/){
    Item item;
    Source(item);
    Operators(item);
    Sink(item);
}
```

SPBench automatically encapsulates data in a standard, simple, and generic way for users.

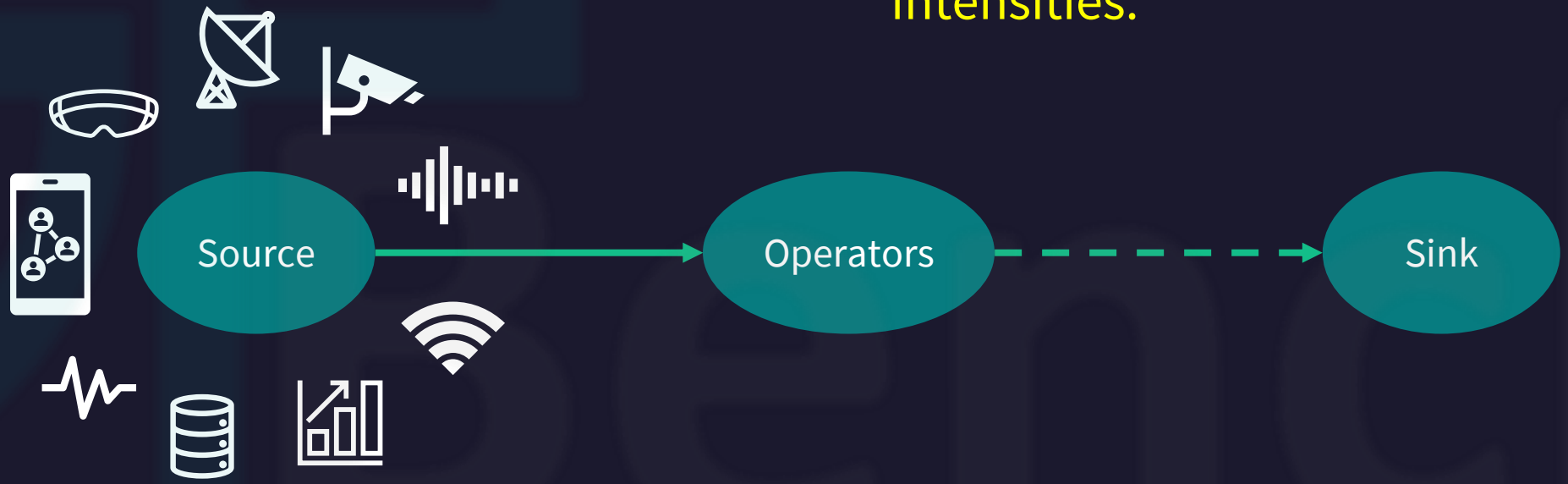
What and how is the input data stream?

There is a wide range of data that can be processed by stream processing applications. This data often arrives at different intensities, such as varying sizes and frequencies.

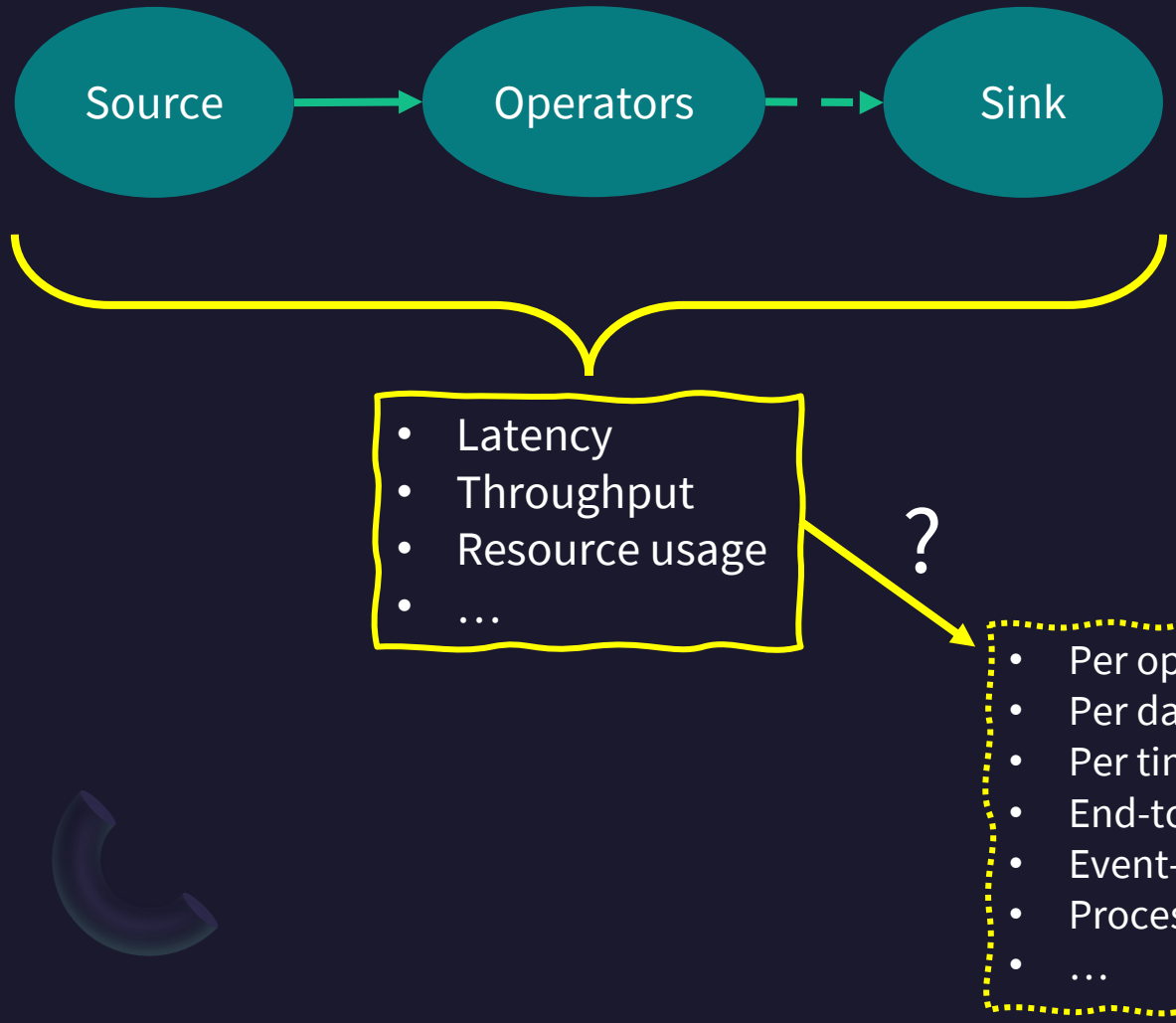


What and how is the input data stream?

SPBench provides tools for users to modify workloads in various ways to simulate different behaviors and data intensities.

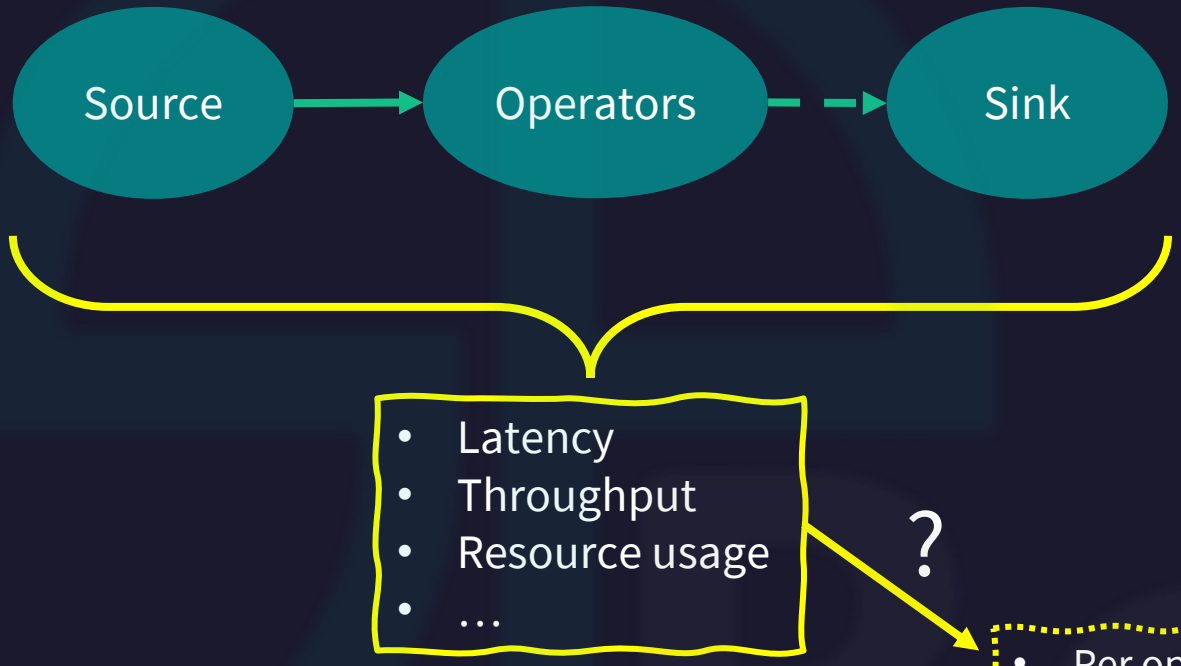


How to evaluate it?

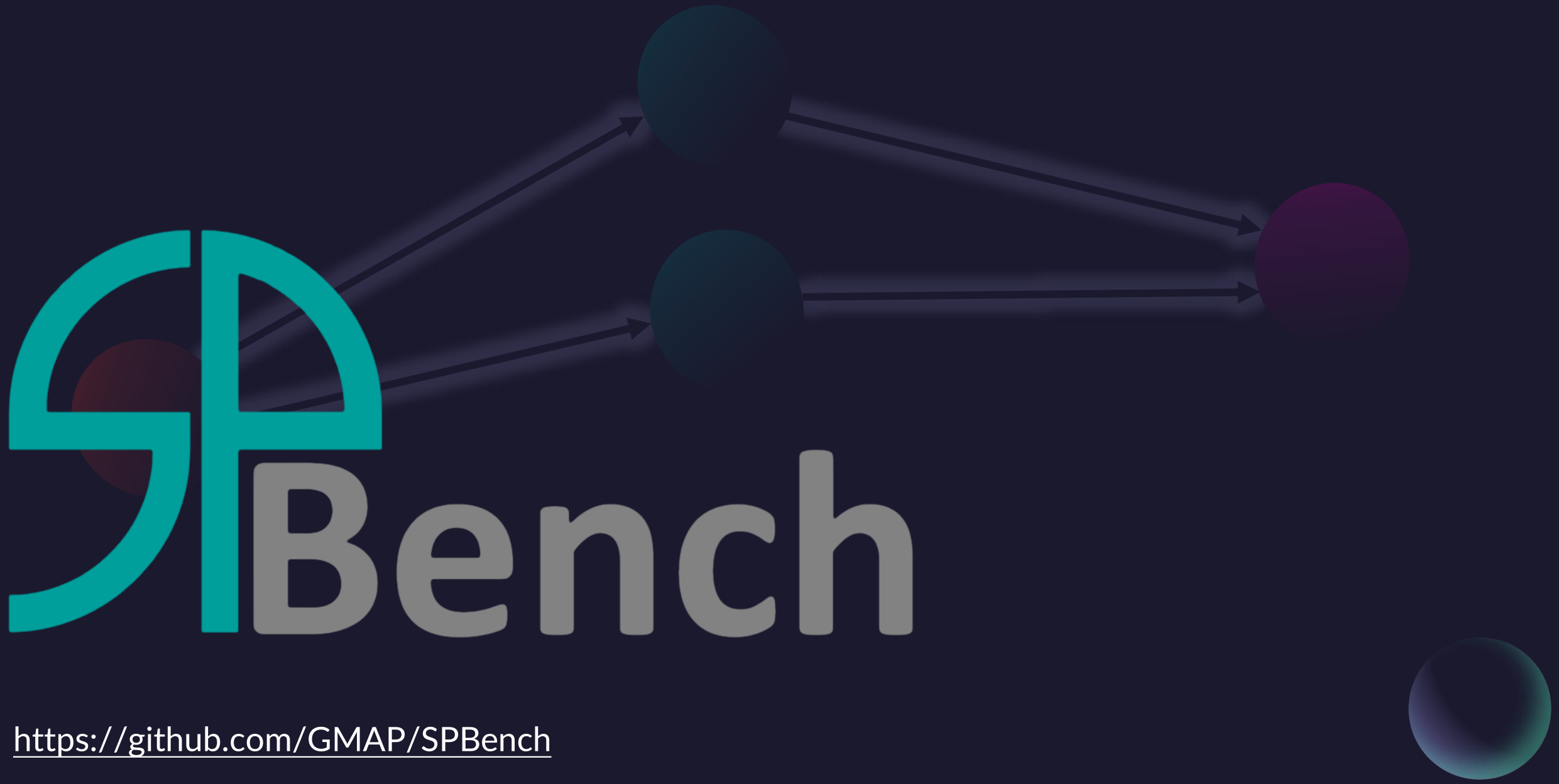


Evaluating stream processing applications can be difficult, since performance metrics can have unclear definitions and different granularity and depth degrees.

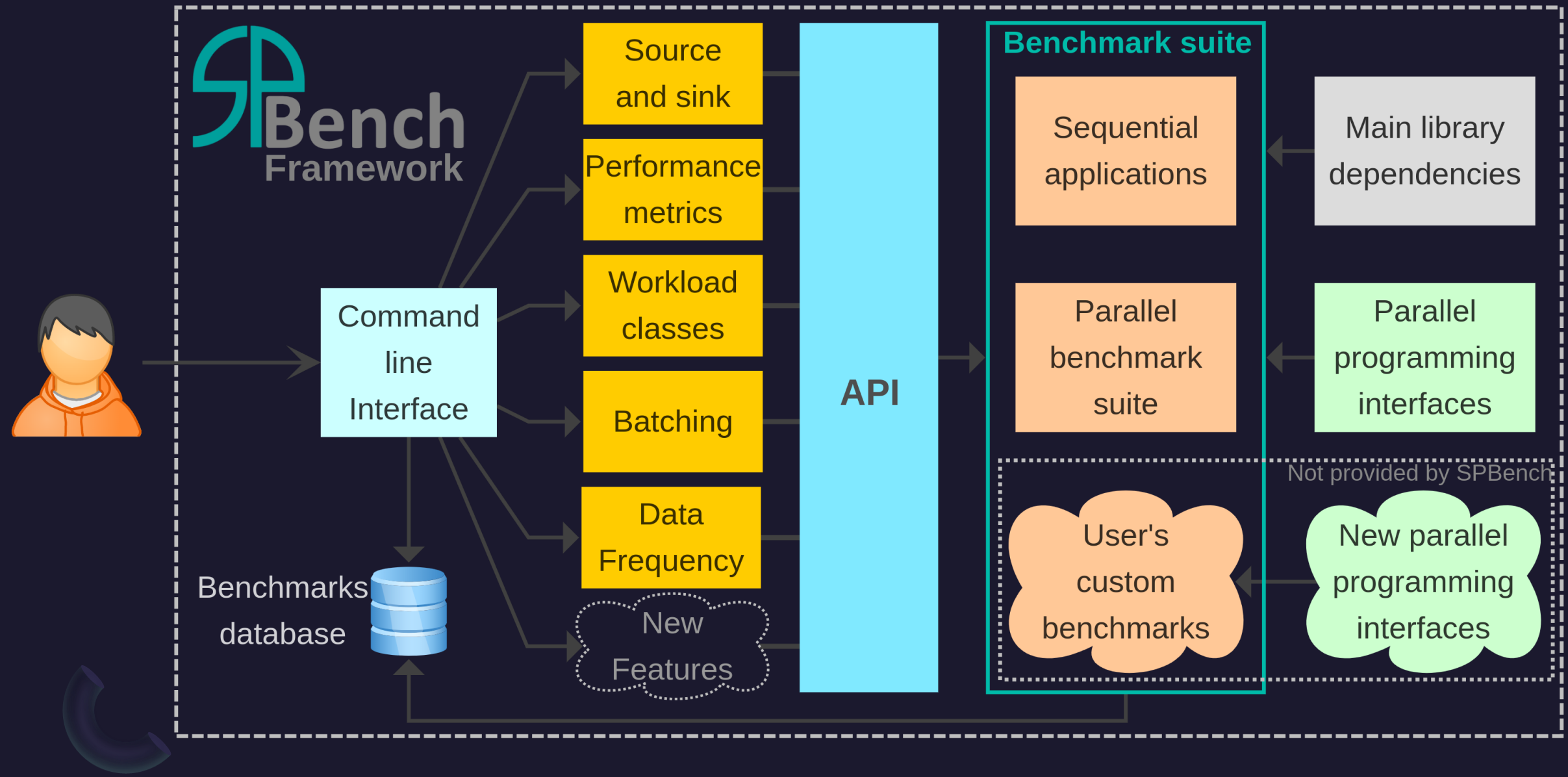
How to evaluate it?

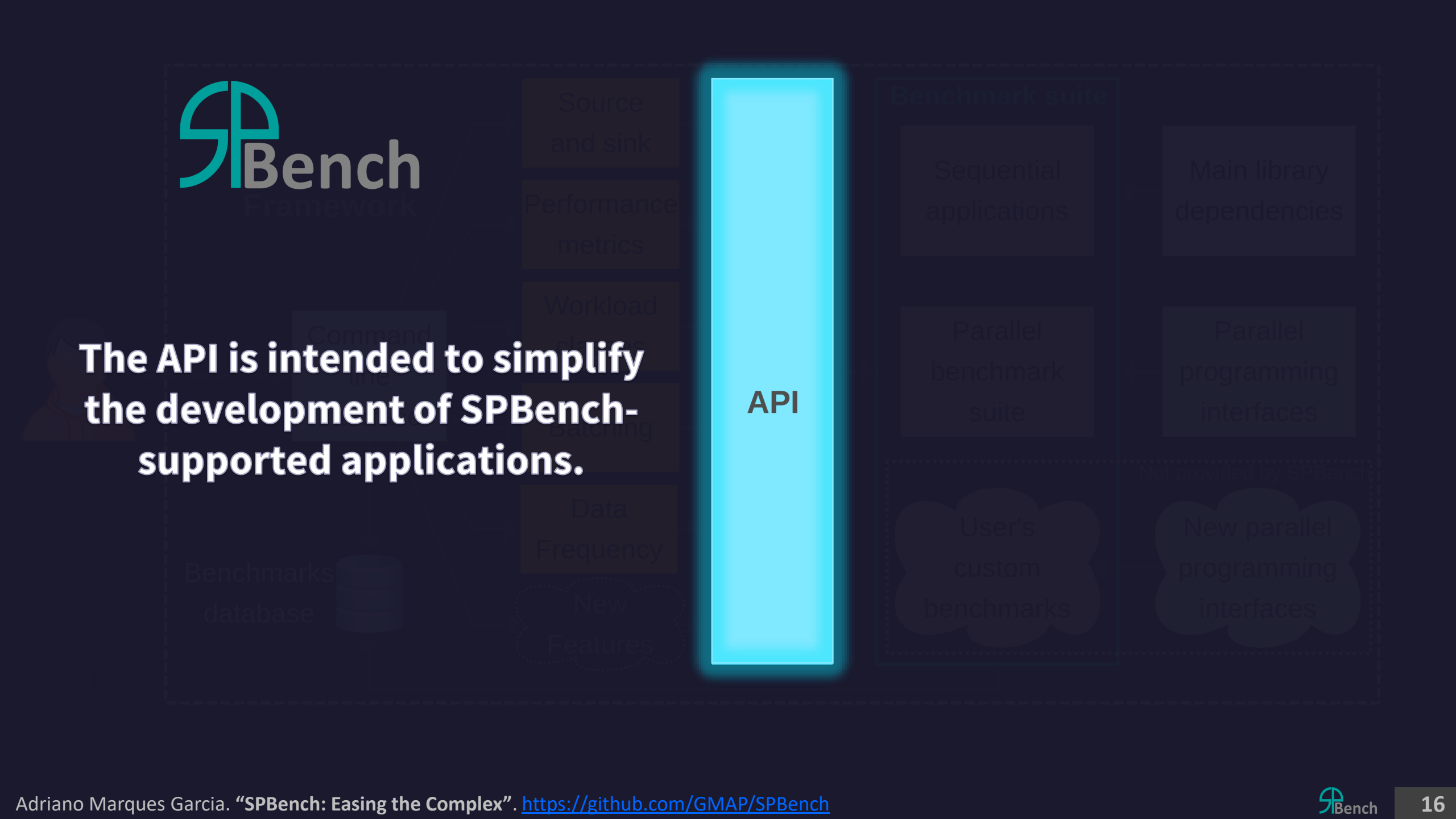


SPBench natively implements across all benchmarks most of the metrics used in the literature.



https://github.com/GMAP/SPBench


SPBench
Framework
Source and sink
Performance metrics
Workload classes
Batching
Data Frequency
New Features
Command line Interface
Benchmarks database
API
Benchmark suite
Sequential applications
Parallel benchmark suite
User's custom benchmarks
Main library dependencies
Parallel programming interfaces
Not provided by SPBench
New parallel programming interfaces



The API is intended to simplify the development of SPBench-supported applications.

```
DIR *pd = m_dir_stack.top();
struct dirent *ent = NULL;
int res = 0;
struct stat st;
int path_len = strlen(m_path);
ent = readdir(pd);
if (ent == NULL) {
            closedir(pd);
            m_path[m_path_stack.top()] = 0;
            m_path_stack.pop();
            m_dir_stack.pop();
            if(m_dir_stack.empty()) {
                        stream_end = true;
                        break;
            }
}
strcat(m_path, ent->d_name);
res = stat(m_path, &st);
if (res != 0) {
            perror("Error:");
            stream_end = true;
            break;}
if (S_ISREG(st.st_mode)) {
            ret = file_helper(m_path);
            m_path[path_len]=0;
            input_found = true;
} else if (S_ISDIR(st.st_mode)) {
            m_path[path_len]=0;
            push_dir(ent->d_name);
} else
            m_path[path_len]=0;
ret->second.seg.name = ret->first.load.name;
ret->second.seg.width = ret->first.load.width;
ret->second.seg.height = ret->first.load.height;
ret->second.seg.HSV = ret->first.load.HSV;
image_segment((void**)&ret->second.seg.mask,
            &ret->second.seg.nrgn,
            ret->first.load.RGB,
            ret->first.load.width,
            ret->first.load.height);
free(ret->first.load.RGB);
cass_query_t query;
query = item.query_batch[num_item];
ret->second.vec.name = ret->extract.name;
memset(&query, 0, sizeof query);
query.flags = CASS_RESULT_LISTS | CASS_RESULT_USERMEM;
ret->second.vec.ds = query.dataset = &ret->extract.ds;
query.vecset_id = 0;
query.vec_dist_id = vec_dist_id;
query.vecset_dist_id = vecset_dist_id;
query.topk = 2*top_K;
query.extra_params = extra_params;
memset(&query, 0, sizeof query);
query.flags = CASS_RESULT_LISTS | CASS_RESULT_USERMEM;
ret->second.vec.ds = query.dataset = &ret->extract.ds;
query.vecset_id = 0;
if (S_ISREG(st.st_mode)) {
            ret = file_helper(m_path);
            m_path[path_len]=0;
            input_found = true;
}
struct dirent *ent = NULL;
int res = 0;
struct stat st;
int path_len = strlen(m_path);
res = stat(m_path, &st);
if (res != 0) {
            perror("Error:");
            stream_end = true;
            break;
}
query.topk = 2*top_K;
query.extra_params = extra_params;
memset(&query, 0, sizeof query);
&ret->second.seg.nrgn,
ret->first.load.RGB,
ret->first.load.width,
ret->first.load.height);
if (ent == NULL) {
            closedir(pd);
            m_path[m_path_stack.top()] = 0;
            m_path_stack.pop();
            m_dir_stack.pop();
}
```

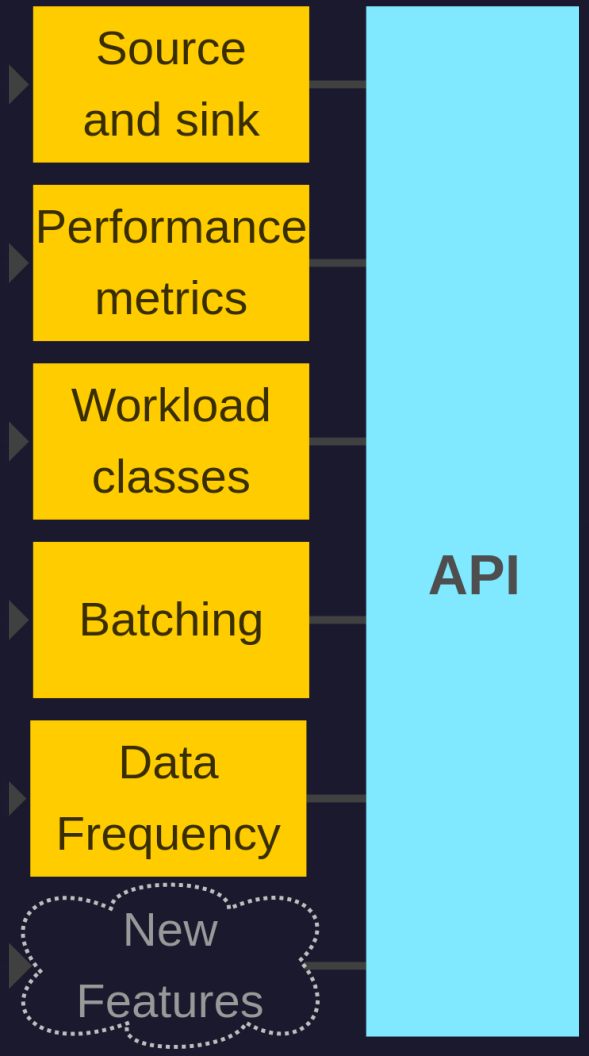
It hides not only the low-level code of the application but also the metrics management and other mechanisms.

API

```
while(/*condition*/){
    Item item;
    source(item);
    operator_1(item);
    ...
    operator_n(item);
    sink(item);
}
```

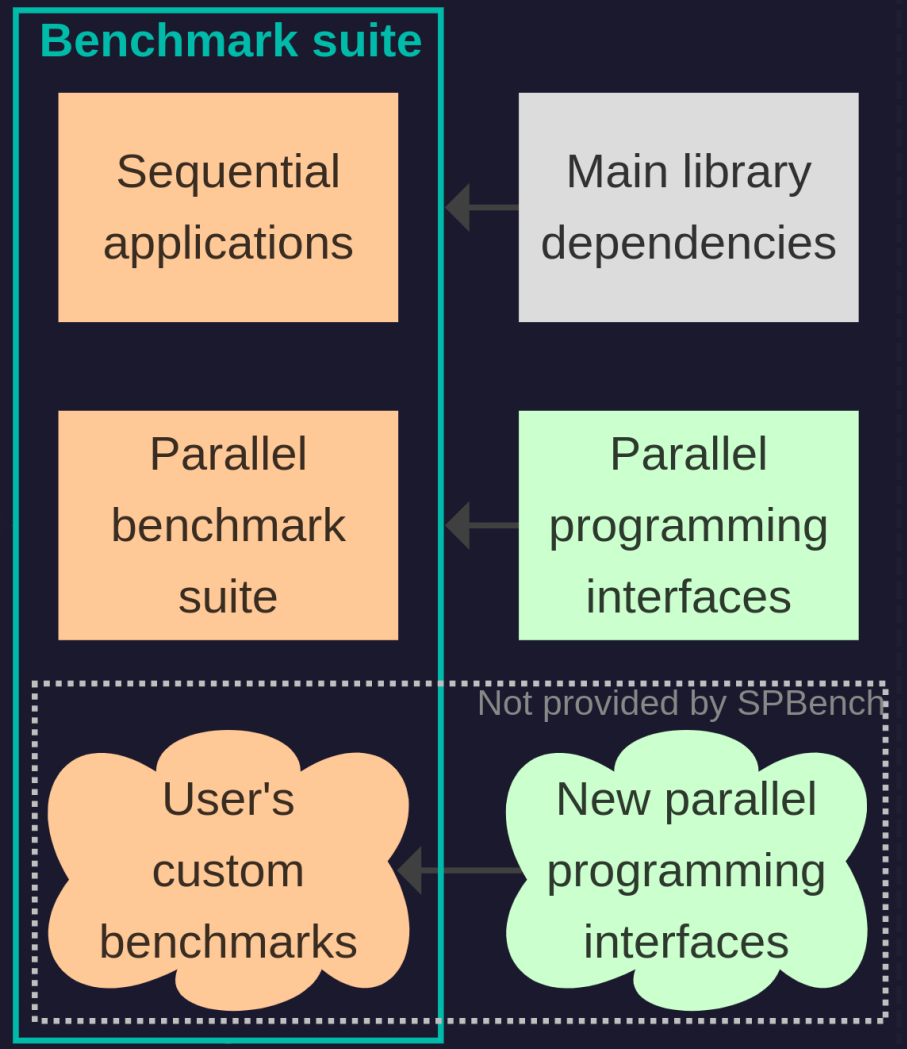
SPBench

Main features supported by the API



The SPBench benchmark suite includes the sequential applications and some parallel implementations.

Users can modify the available benchmarks or create new ones.

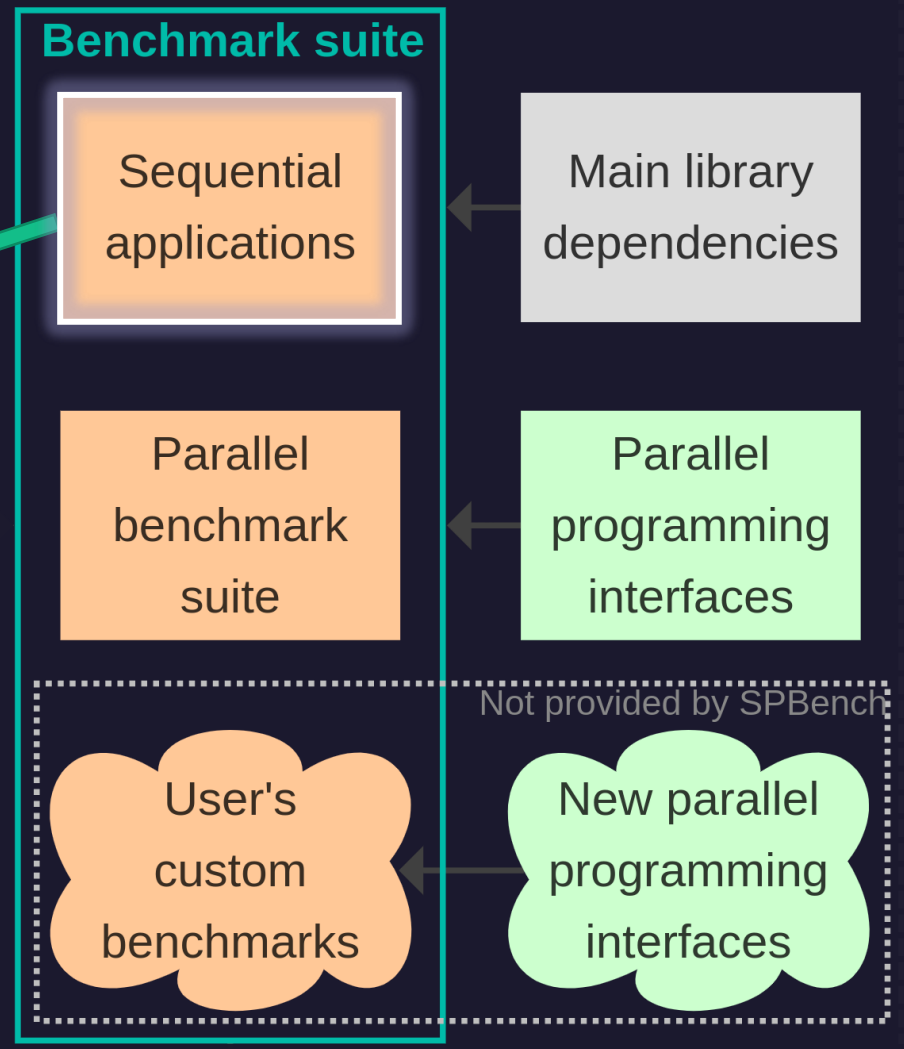


https://spbench-doc.rtfd.io/en/latest/SPBench_benchmarks.html

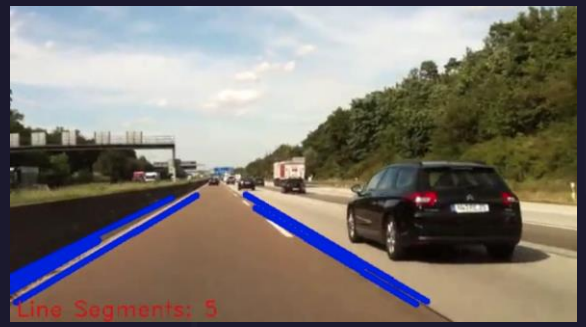
SPBench

```
while(/*condition*/){

    Item item;

    source(item);

    operator_1(item);
    ...
    operator_n(item);

    sink(item);
}
```

Example of sequential benchmark



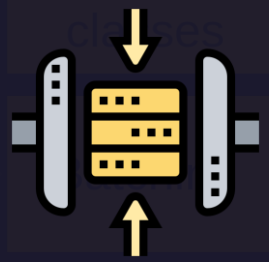
https://spbench-doc.rtfd.io/en/latest/SPBench_benchmarks.html



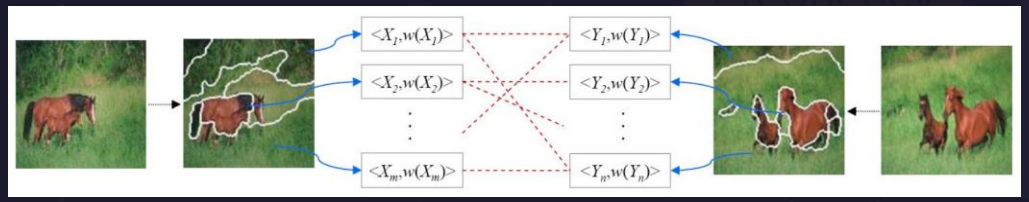
Lane Detection

Face Recognition

Fraud Detection



Bzip2



Content similarity search (Ferret - PARSEC)

Benchmark suite

Sequential applications

Main library dependencies

Parallel benchmark suite

Parallel programming interfaces

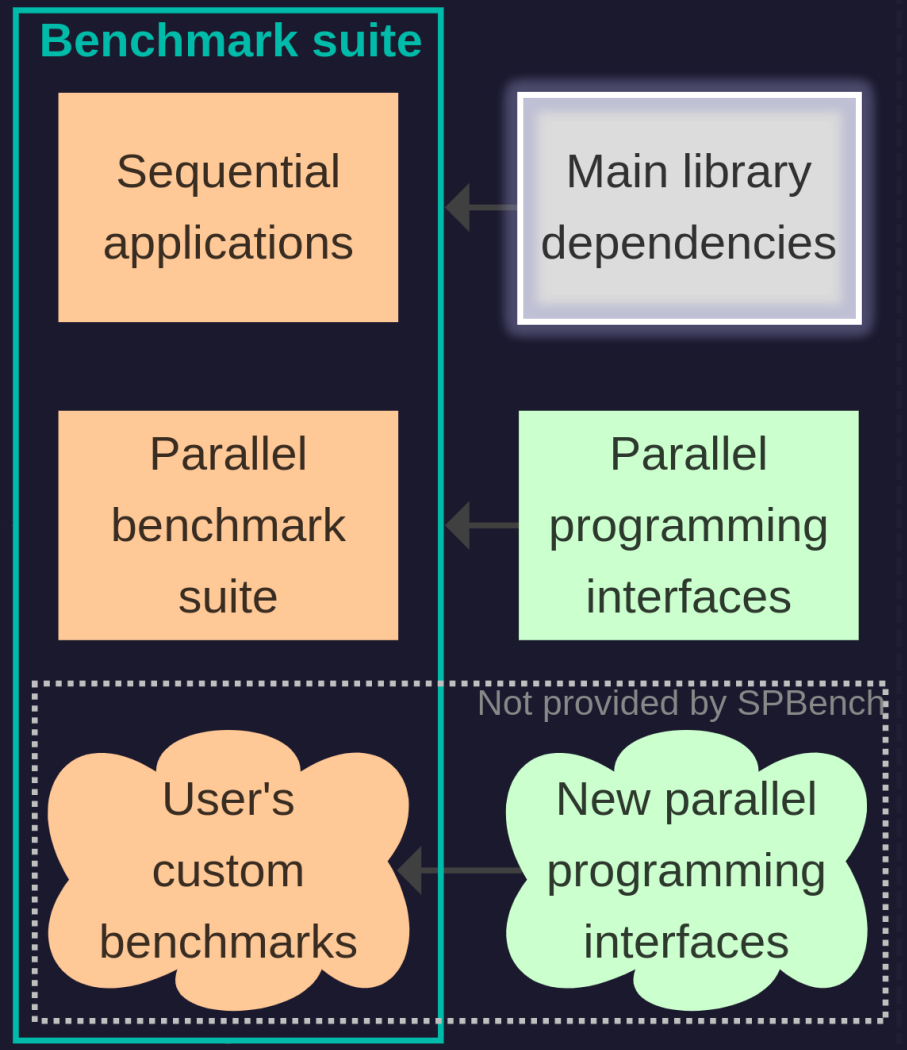
Not provided by SPBench

User's custom benchmarks

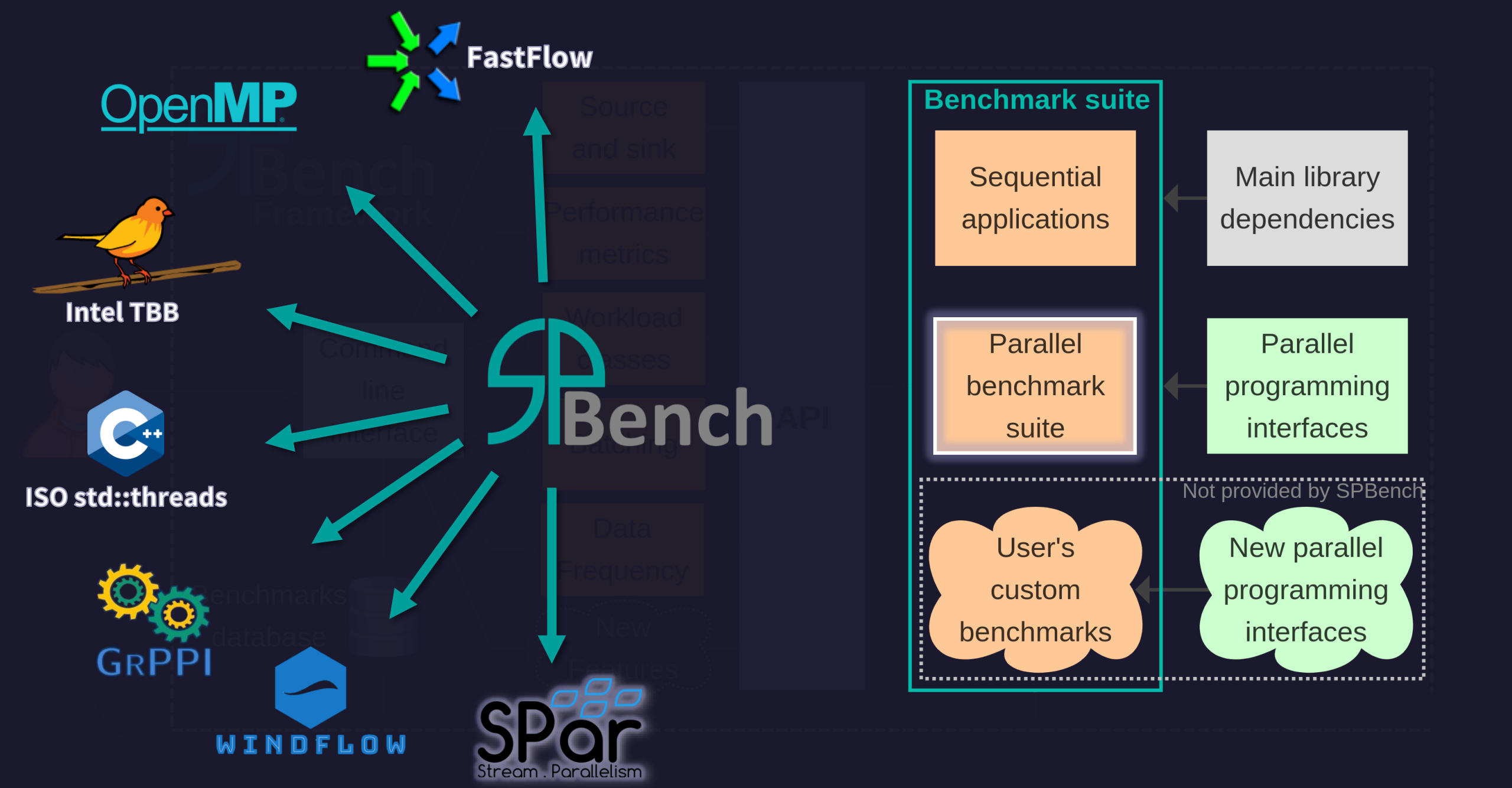
New parallel programming interfaces

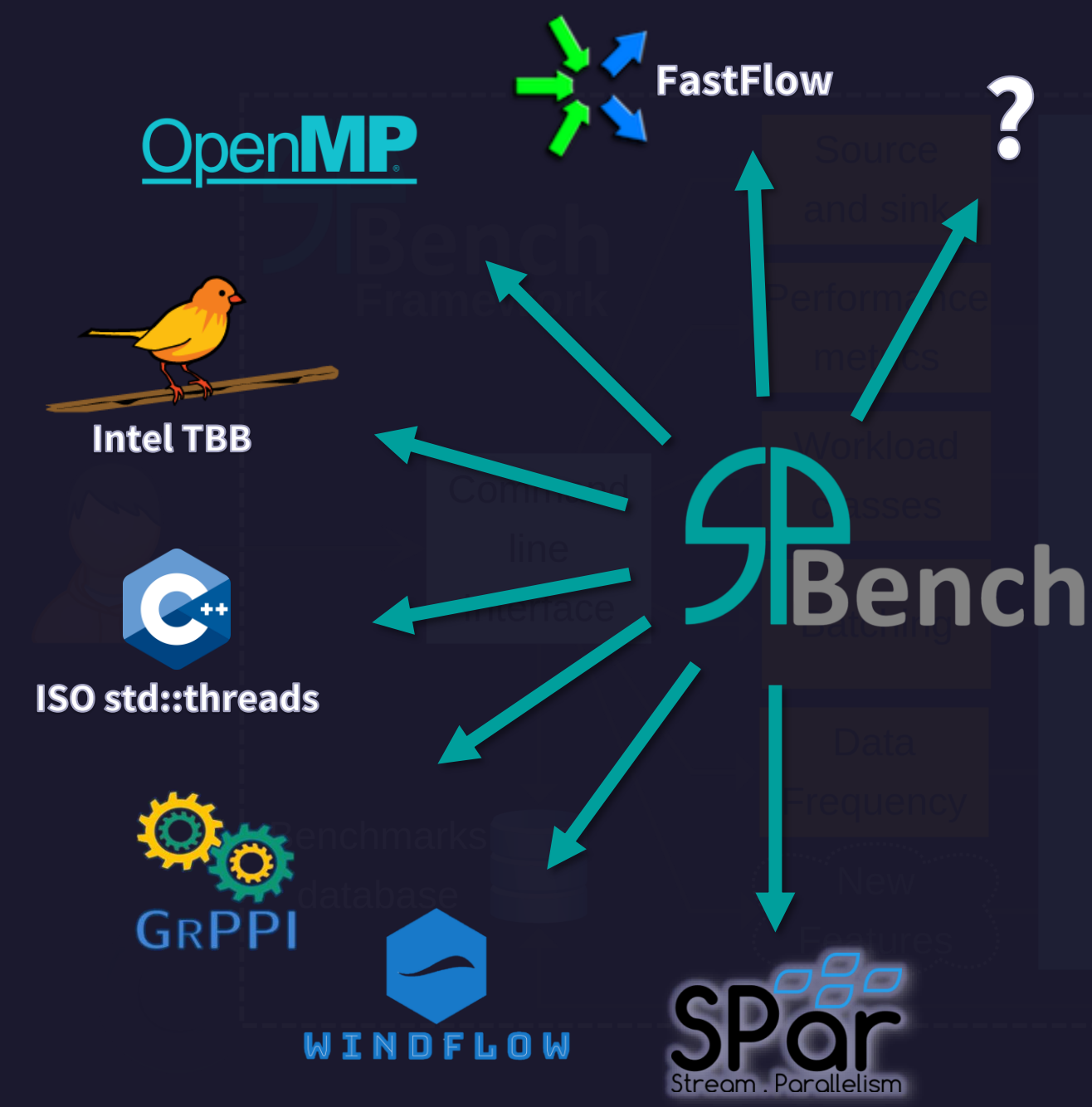
https://spbench-doc.rtfd.io/en/latest/SPBench_applications.html

SPBench provides and install all specific library dependencies



https://spbench-doc.rtfd.io/en/latest/install.html


FastFlow
OpenMP
Intel TBB
ISO std::threads
GrPPI
WINDFLOW
SPar
Stream . Parallelism
SPBench
Benchmark suite
Sequential applications
Main library dependencies
Parallel benchmark suite
Parallel programming interfaces
Not provided by SPBench
User's custom benchmarks
New parallel programming interfaces

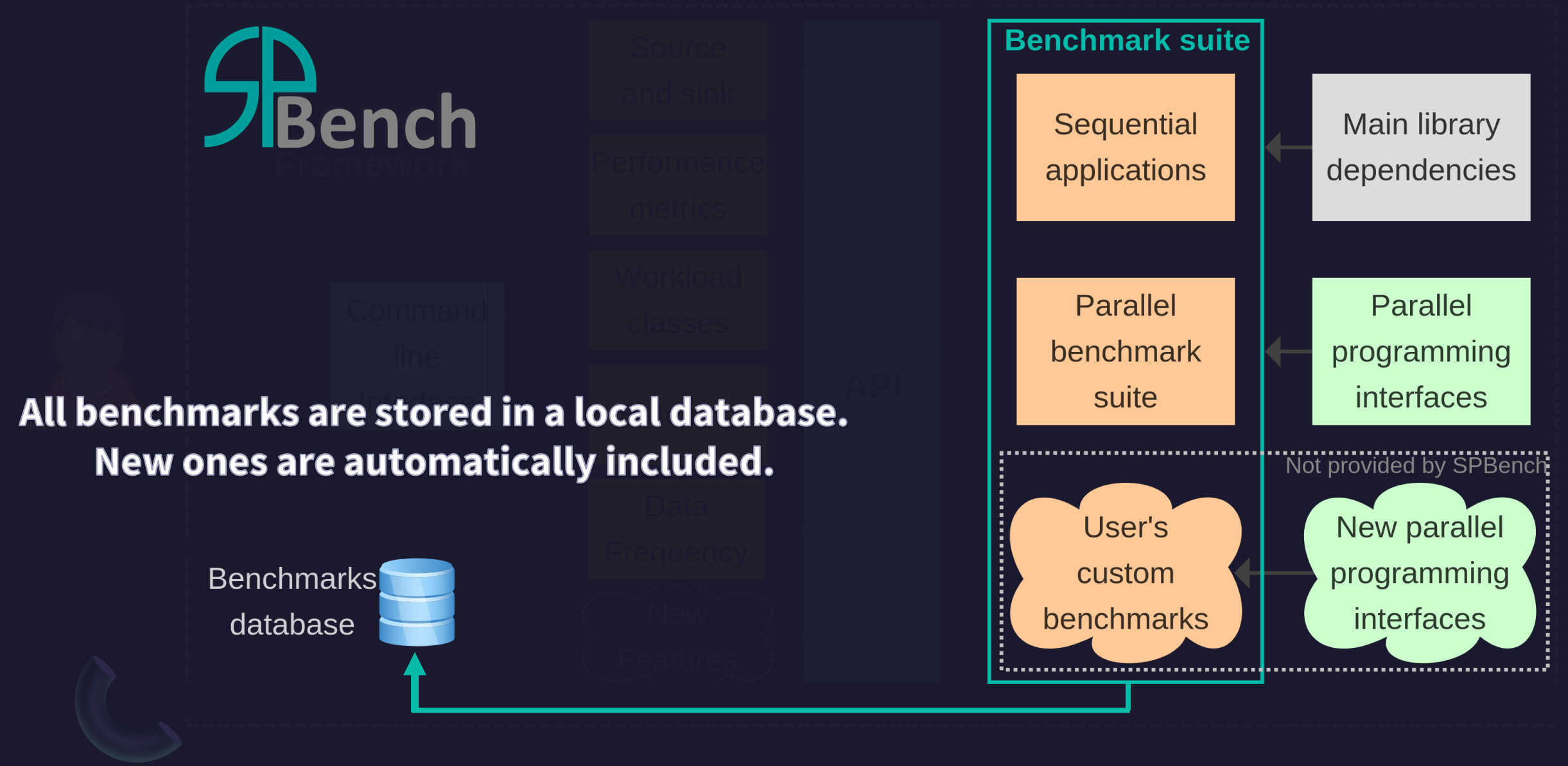


Benchmark suite

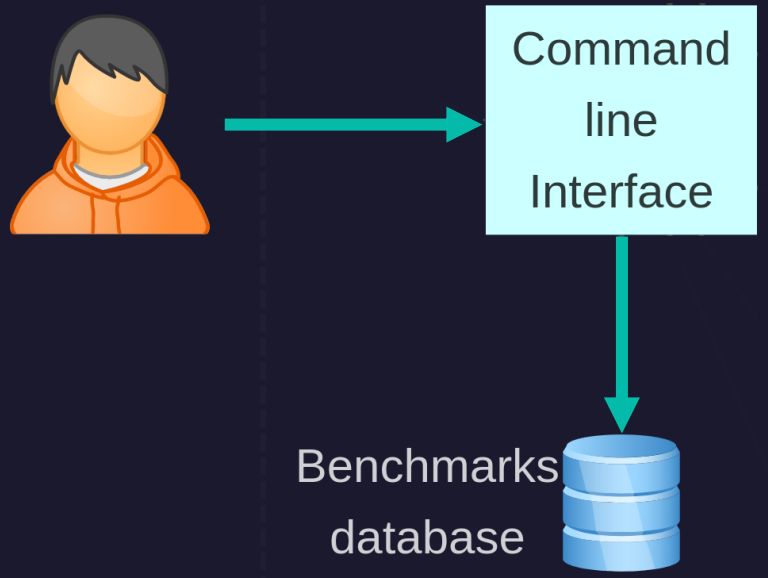
- Sequential applications
- Main library dependencies
- Parallel benchmark suite
- Parallel programming interfaces

Not provided by SPBench

- User's custom benchmarks
- New parallel programming interfaces

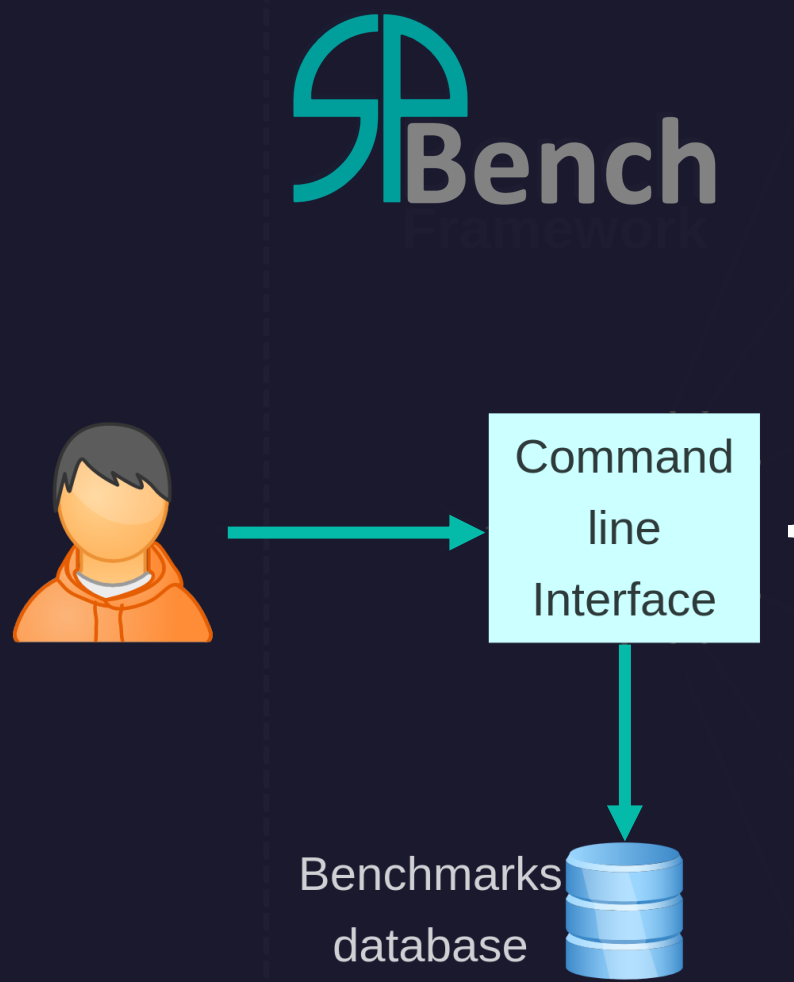


All benchmarks are stored in a local database.
New ones are automatically included.



The management and access to the benchmarks can be done through the SPBench CLI.

https://spbench-doc.rtfd.io/en/latest/CLI.html



- **new-app**
- **new-input**
- **new-bench**
- **edit**
- **configure**
- **compile**
- **exec**

Main commands currently supported by the CLI for managing the benchmarks.

https://spbench-doc.rtfd.io/en/latest/CLI.html

SPBench

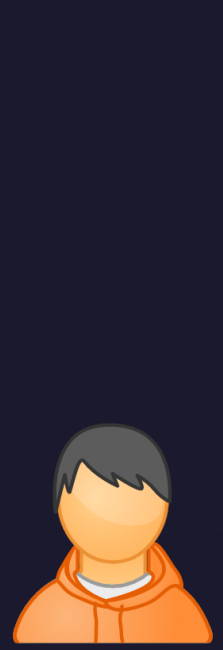
Command line Interface

Benchmarks database

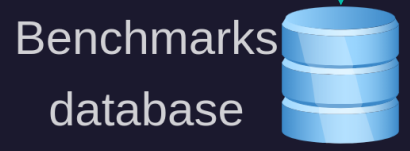
- new-app
- new-input
- new-bench
- edit
- **configure**
- compile
- exec

```
{
  "CXX": "g++ -std=c++1y",
  "CXX_FLAGS":"-O3",
  "PPI_CXX": "g++ -std=c++1y",
  "PPI_CXX_FLAGS":"-O3",
  "MACROS": "-DBLOCKING_MODE -DNO_DEFAULT_MAPPING",
  "PKG-CONFIG": {
     "opencv" :"pkg-config --cflags --libs opencv"
  },
  "INCLUDES": {
     "fastflow": "-I $SPB_HOME/ppis/fastflow/",
     "UPL" : "-I $SPB_HOME/libs/upl/include/upl/"
  },
  "LIBS": {
     "UPL" : "-L $SPB_HOME/libs/upl/lib/x86 -lupl"
  },
  "LDFLAGS": "-lpthread"
}
```

https://spbench-doc.rtfd.io/en/latest/CLI.html



Command line Interface

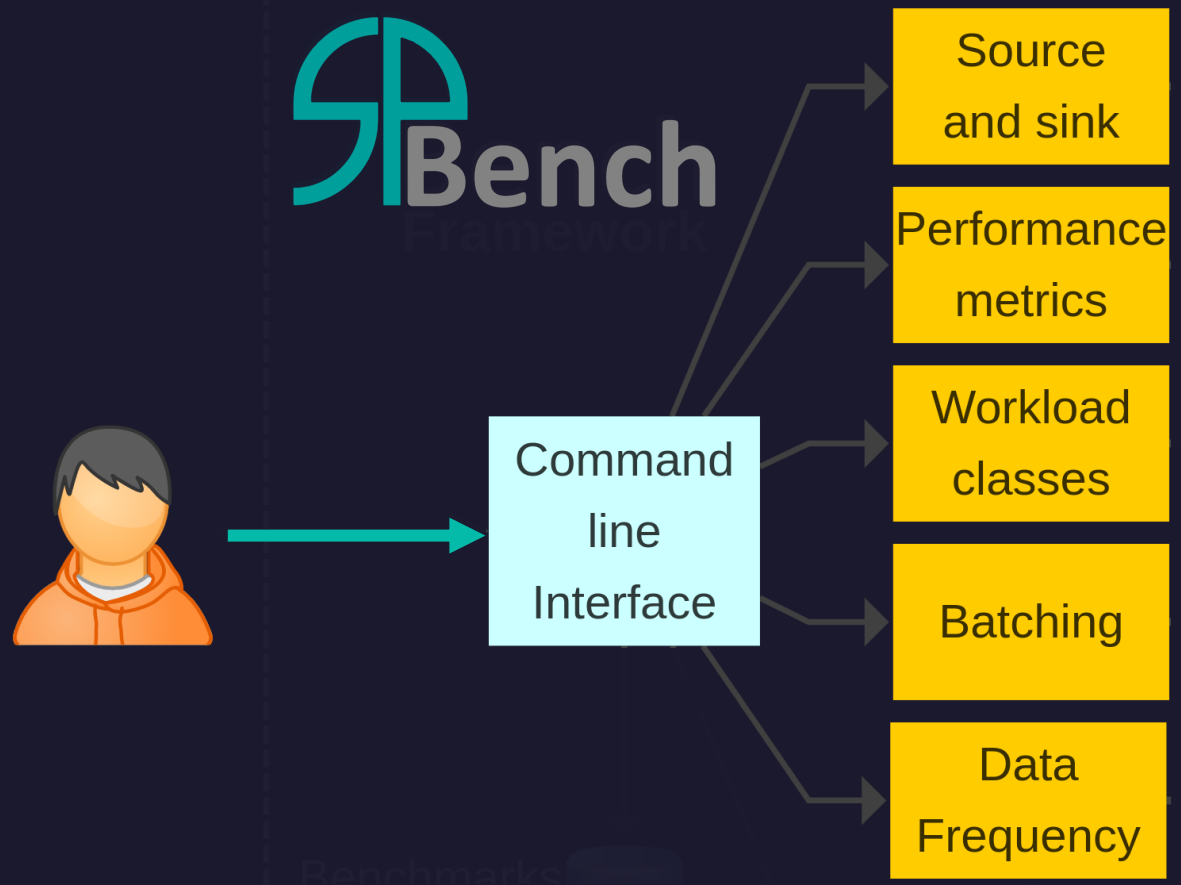


Benchmarks database

- new-app
- new-input
- new-bench
- edit
- configure
- compile
- **exec**

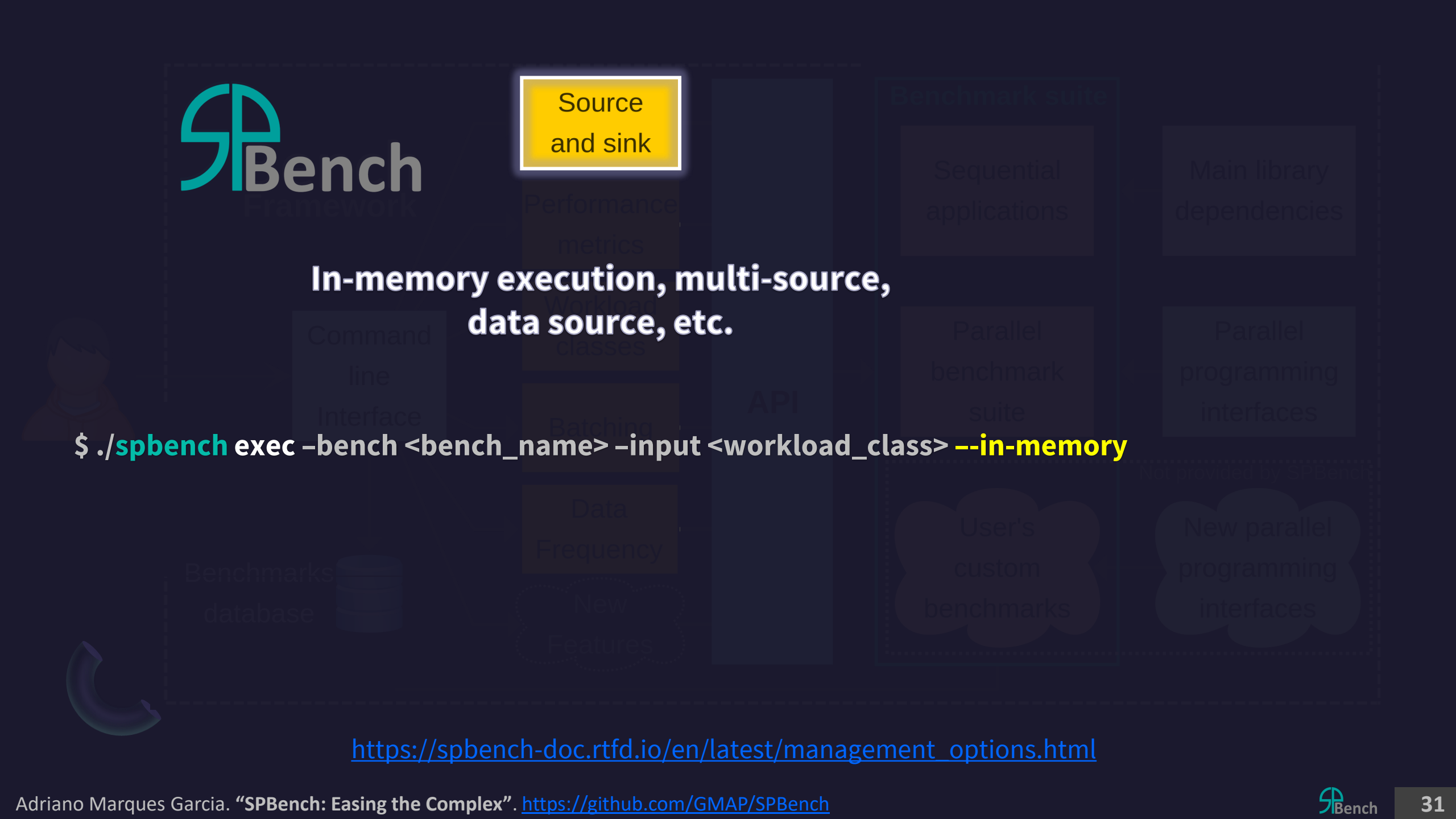
$ ./spbench exec –bench <bench_name> … –repeat 10 –nthreads 1:5:30

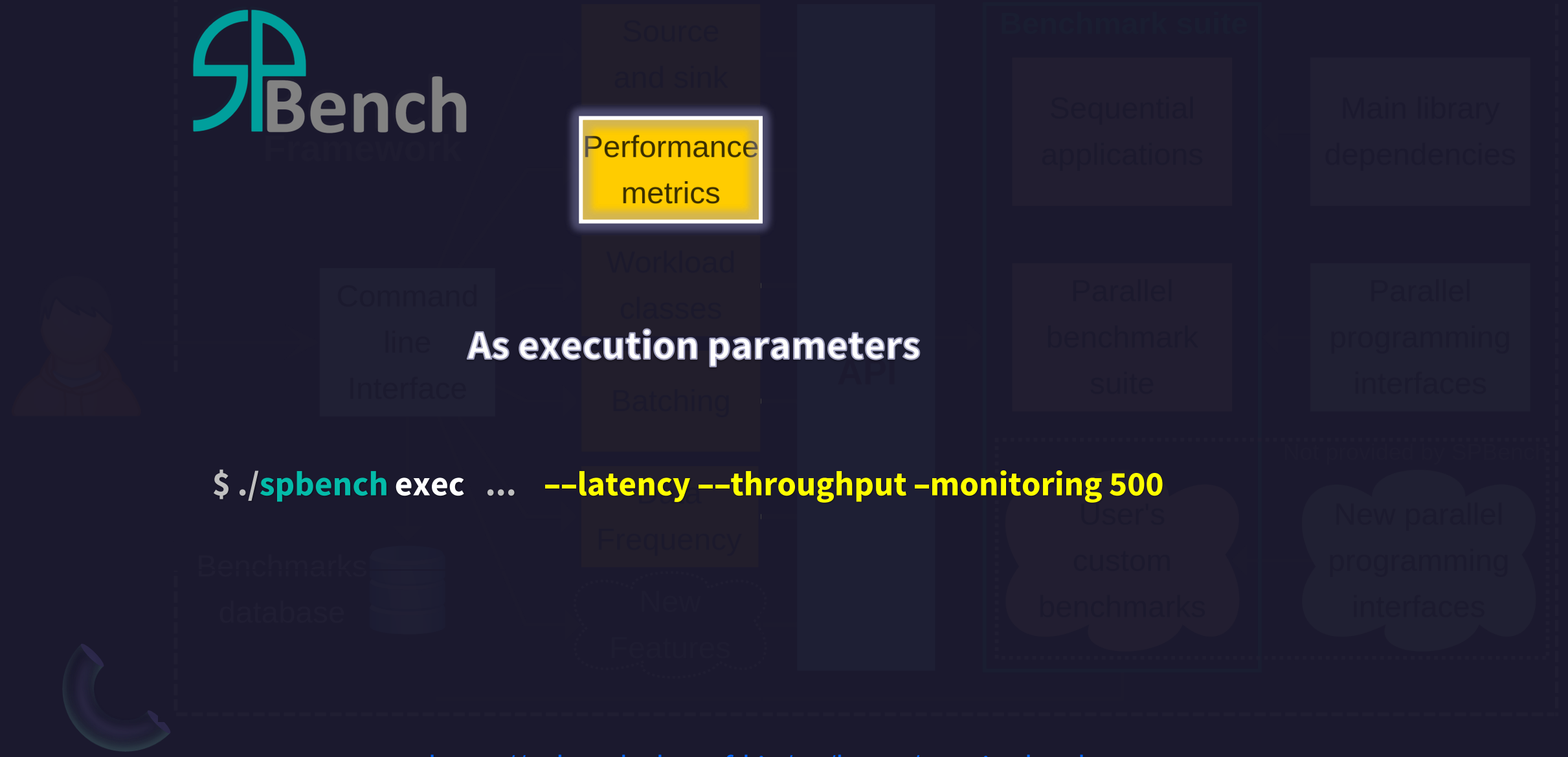
https://spbench-doc.rtfd.io/en/latest/CLI.html



The CLI also allows for managing all execution parameters.

https://spbench-doc.rtfd.io/en/latest/CLI.html

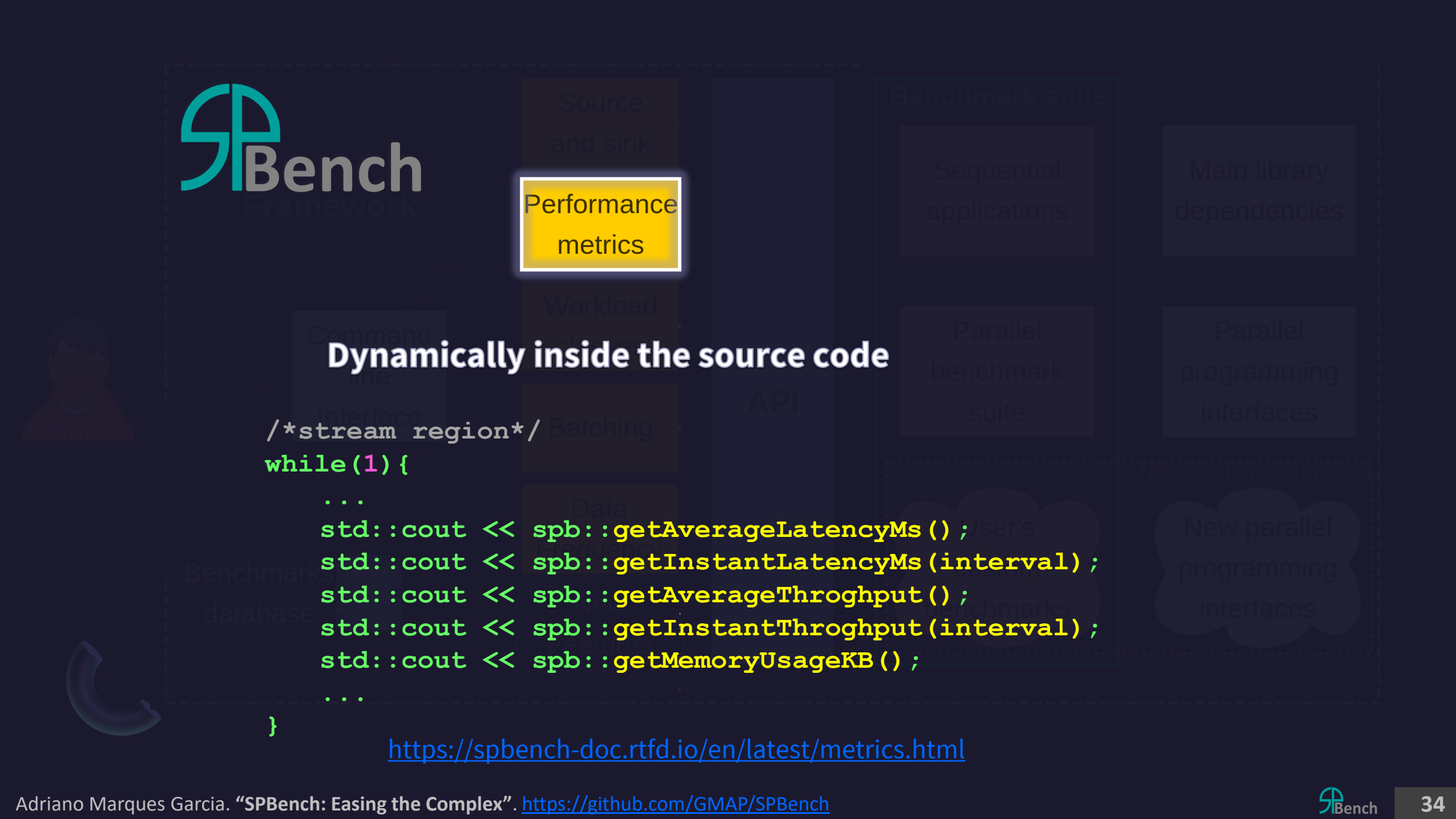

SPBench
Source and sink
In-memory execution, multi-source, data source, etc.
$ ./spbench exec –bench <bench_name> –input <workload_class> --in-memory
https://spbench-doc.rtfd.io/en/latest/management_options.html

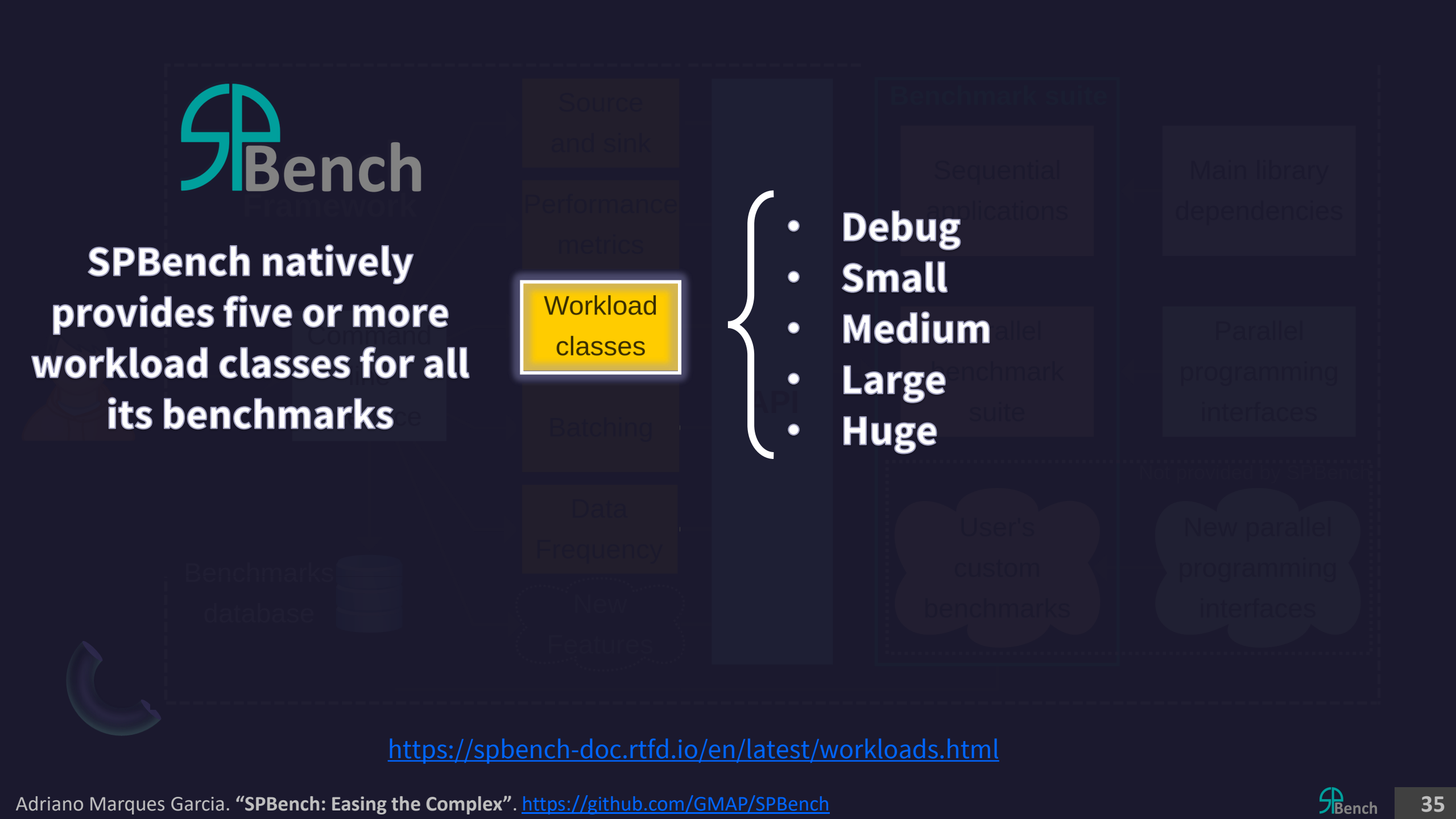


As execution parameters

```
$ ./spbench exec   ...   ––latency ––throughput –monitoring 500
```

https://spbench-doc.rtfd.io/en/latest/metrics.html


SPBench
Performance metrics
Dynamically inside the source code
/*stream region*/
while(1){
...
std::cout << spb::getAverageLatencyMs();
std::cout << spb::getInstantLatencyMs(interval);
std::cout << spb::getAverageThroghput();
std::cout << spb::getInstantThroghput(interval);
std::cout << spb::getMemoryUsageKB();
...
}
https://spbench-doc.rtfd.io/en/latest/metrics.html


SPBench
SPBench natively provides five or more workload classes for all its benchmarks
Workload classes
Debug
Small
Medium
Large
Huge
https://spbench-doc.rtfd.io/en/latest/workloads.html

SPBench

Workload classes

$ ./spbench new-input ...

Inputs database

Users can also register new input workloads

https://spbench-doc.rtfd.io/en/latest/workloads.html

SPBench

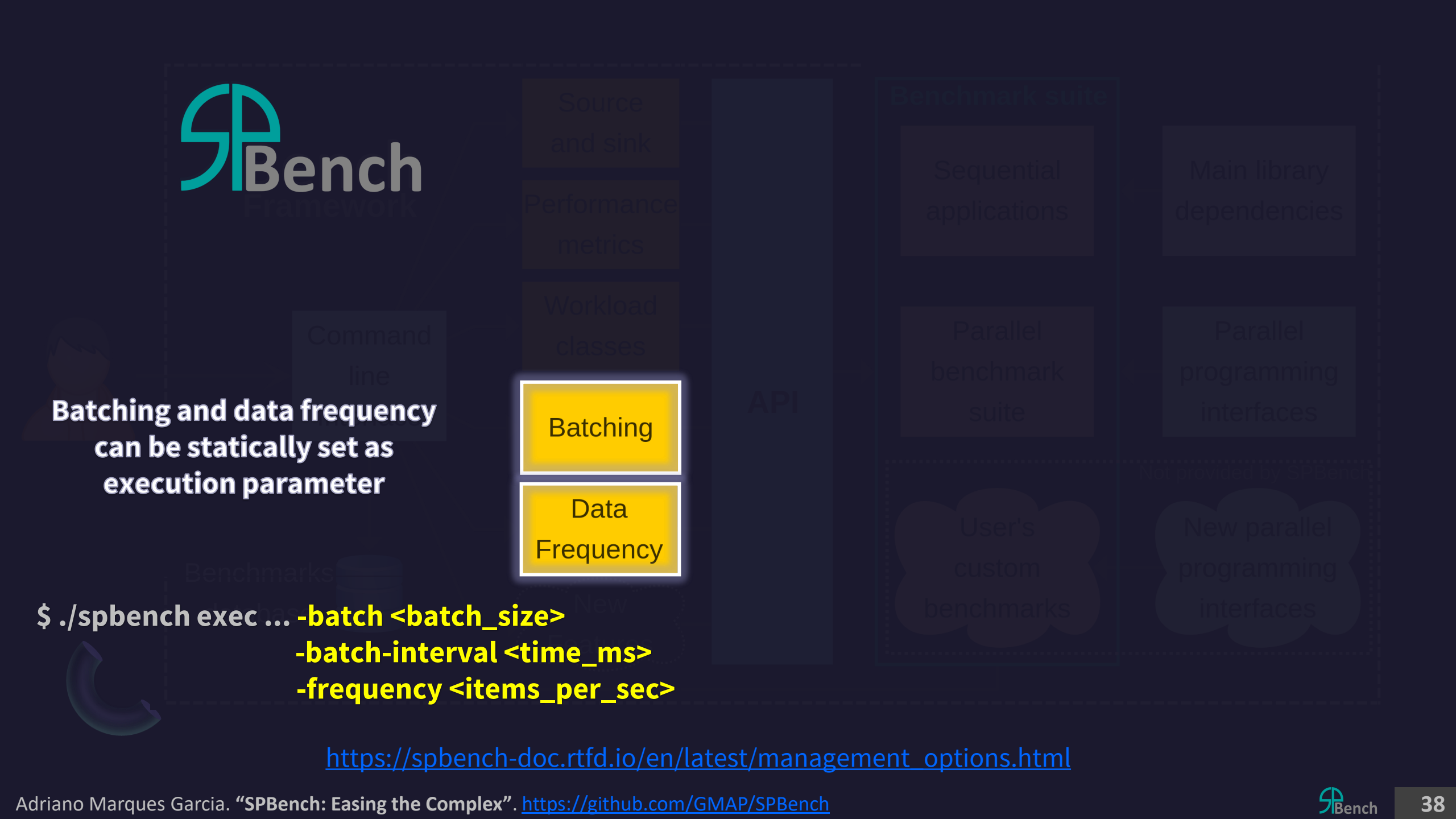
Workload classes

It supports correctness testing

$ ./spbench new-input –md5 <md5_hash_to_validate> ...

Inputs database

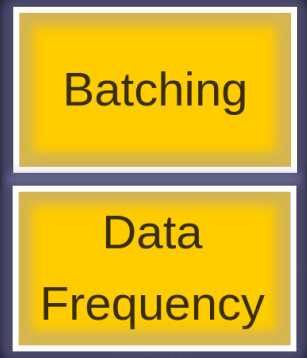
https://spbench-doc.rtfd.io/en/latest/workloads.html



https://spbench-doc.rtfd.io/en/latest/management_options.html

SPBench

Batching and data frequency can be statically set as execution parameter




```
$ ./spbench exec ... -batch <batch_size>
                     -batch-interval <time_ms>
                     -frequency <items_per_sec>
```

But they can also be dynamically set at any point during execution.

```
spb::setBatchSize(5);
spb::setFrequency(50000);

/*stream region*/
while(1){
    Item item;
    ...
    if(item.index > 1000){
        spb::setBatchSize(1);
        spb::setFrequency(10000);
    }
    ...
}
```

https://spbench-doc.rtfd.io/en/latest/management_options.html


```
Frequency pattern: wave

   #  #                  #  #  - - - - - - - - -> Maximum: 60 items per second
 #      #              #     #
          #          #        #
             #    #              #        #
                #  #                  #  # - - -> Minimum: 10 items per second
                   |_____________________| -> Periods: 30 seconds
```

```
Frequency pattern: binary

          # # # # #        # # # #  - -> Maximum: 60 items per second
          #       #        #
          #       #        #
# # # # #         # # # # #  - - - - -> Minimum: 10 items per second
                  |_______________| - -> Periods: 30 seconds
```

Different frequency patterns can be set or even combined

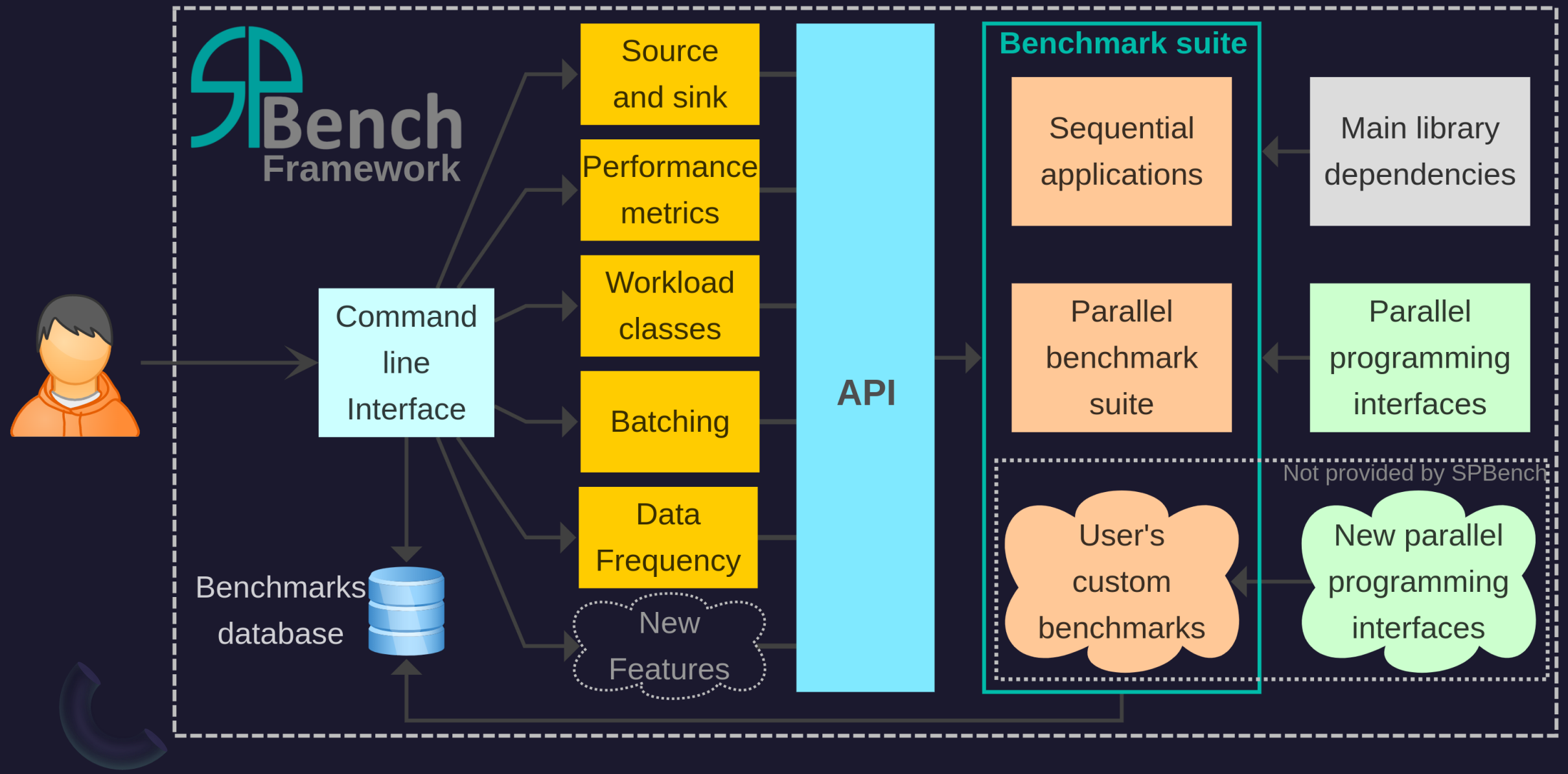
Data Frequency

```
Frequency pattern: spike

            #             #  - -> Maximum: 60 items per second
           ##            ##
          # #           # #
# # # # #  # # # # # #  # # #-> Minimum: 10 items per second
                      |____| - -> Spikes:  3.0 seconds
             |_____________| - -> Periods: 30 seconds
```

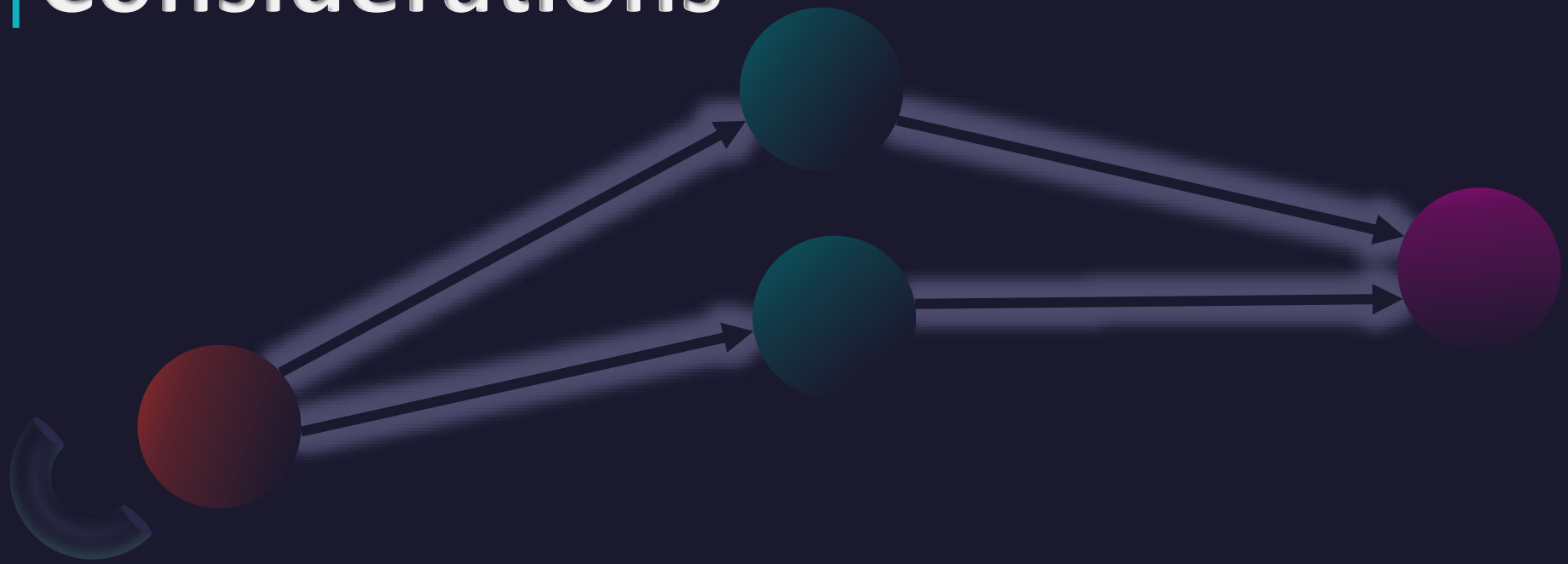
$ ./spbench exec ... -freq-pattern <pattern, period, min, max>

```
spb::setFrequencyPattern(pattern, period, min, max);
```

https://spbench-doc.rtfd.io/en/latest/management_options.html

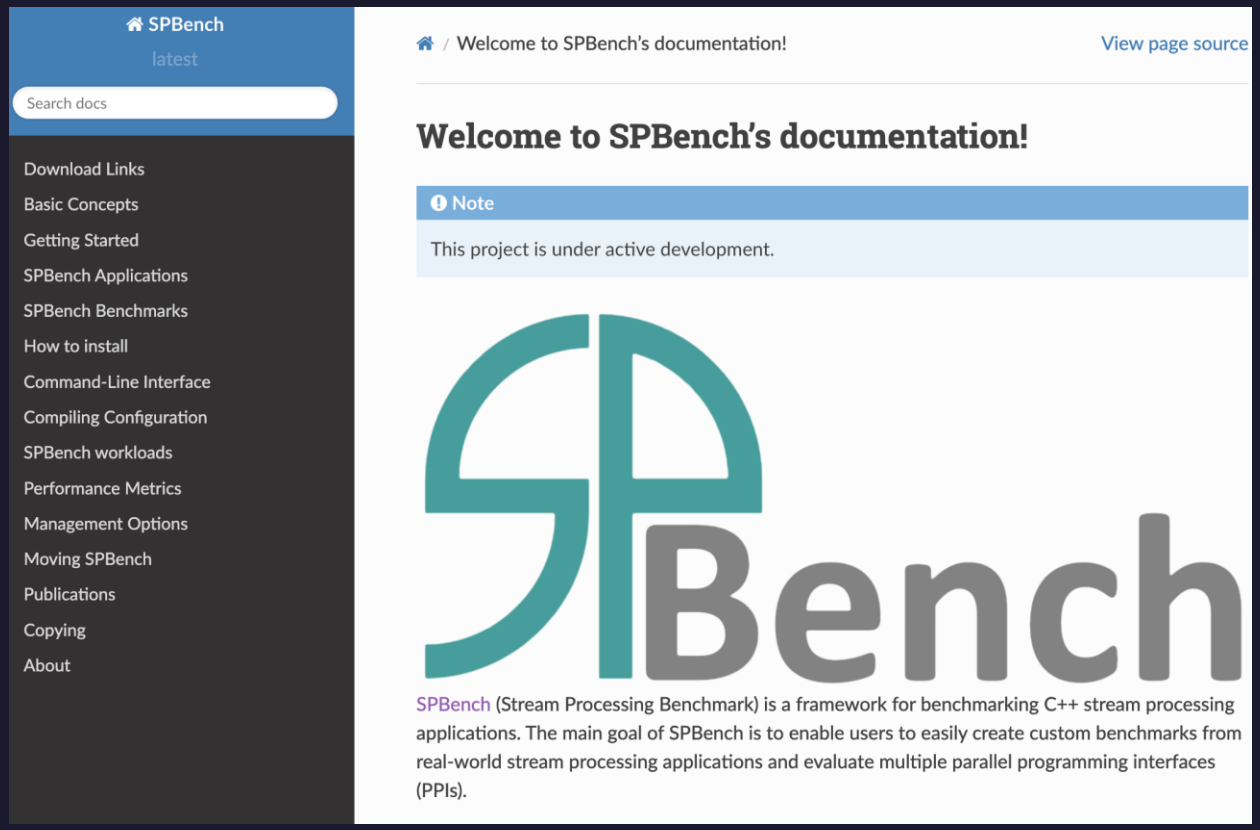

SPBench
Framework
Source and sink
Performance metrics
Workload classes
Batching
Data Frequency
New Features
Command line Interface
Benchmarks database
API
Benchmark suite
Sequential applications
Main library dependencies
Parallel benchmark suite
Parallel programming interfaces
Not provided by SPBench
User's custom benchmarks
New parallel programming interfaces

Final Considerations



Documentation

https://spbench-doc.rtfd.io



SPBench Benchmarks — View page source

SPBench Benchmarks

Table of Contents

Single source benchmarks

These are the commom benchmarks. They read data from a single input and generate a single stream of input and output data. The source operator runs inside the stream region, as shown in the example below.

```
int main (int argc, char* argv[]){

   /* Init of stream region */
   while(1){
      spb::Item item;
      if(!spb::Source::op(item)) break;
      spb::First::op(item);
      spb::Second::op(item);
      spb::Nth::op(item);
      spb::Sink::op(item);
   } // End of stream region

   return 0;
}
```

Multiple source benchmarks

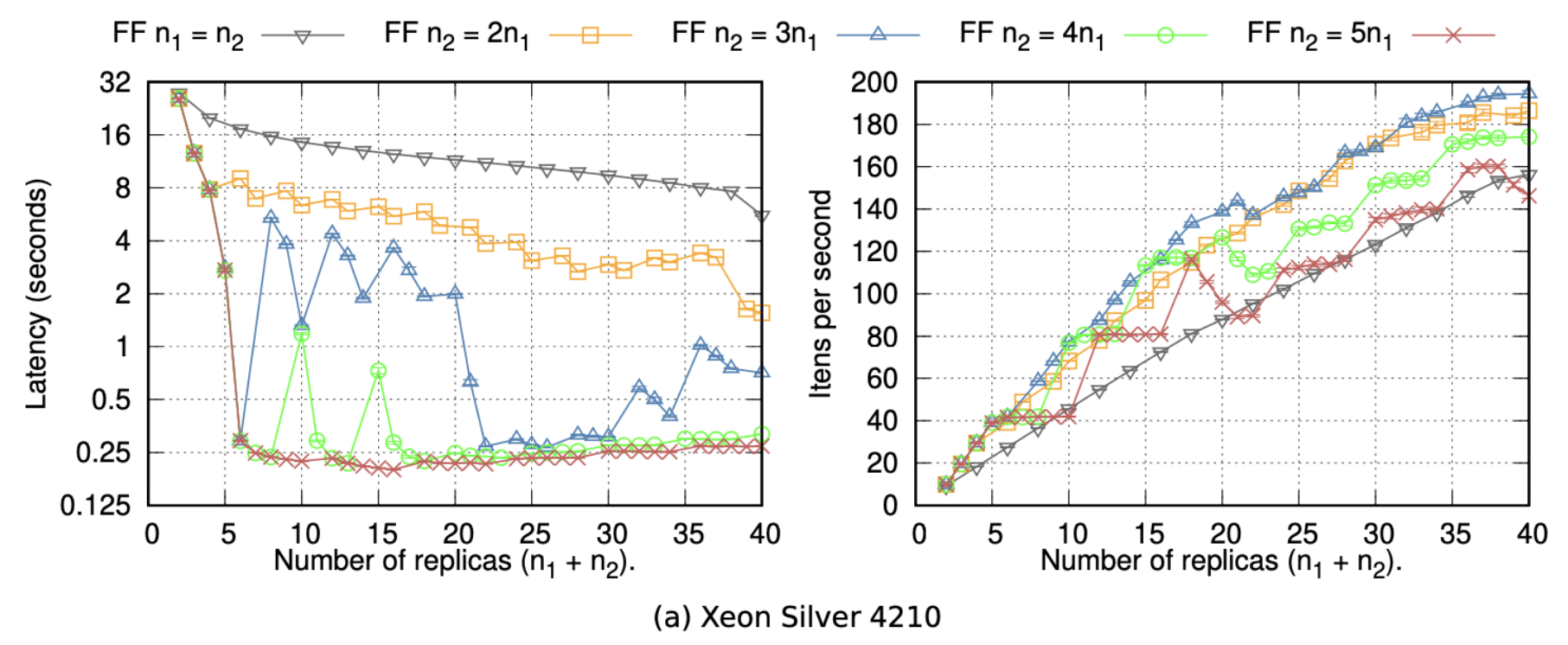
Main characteristics of the SPBench

- It is self-contained
- It is easy to use
- Enables fast code replication
- Highly configurable and parameterizable
- Offers wide range of performance metrics
- Provides a benchmark suite with multiple parallel programming interfaces

SPBench is intended for three main audiences:

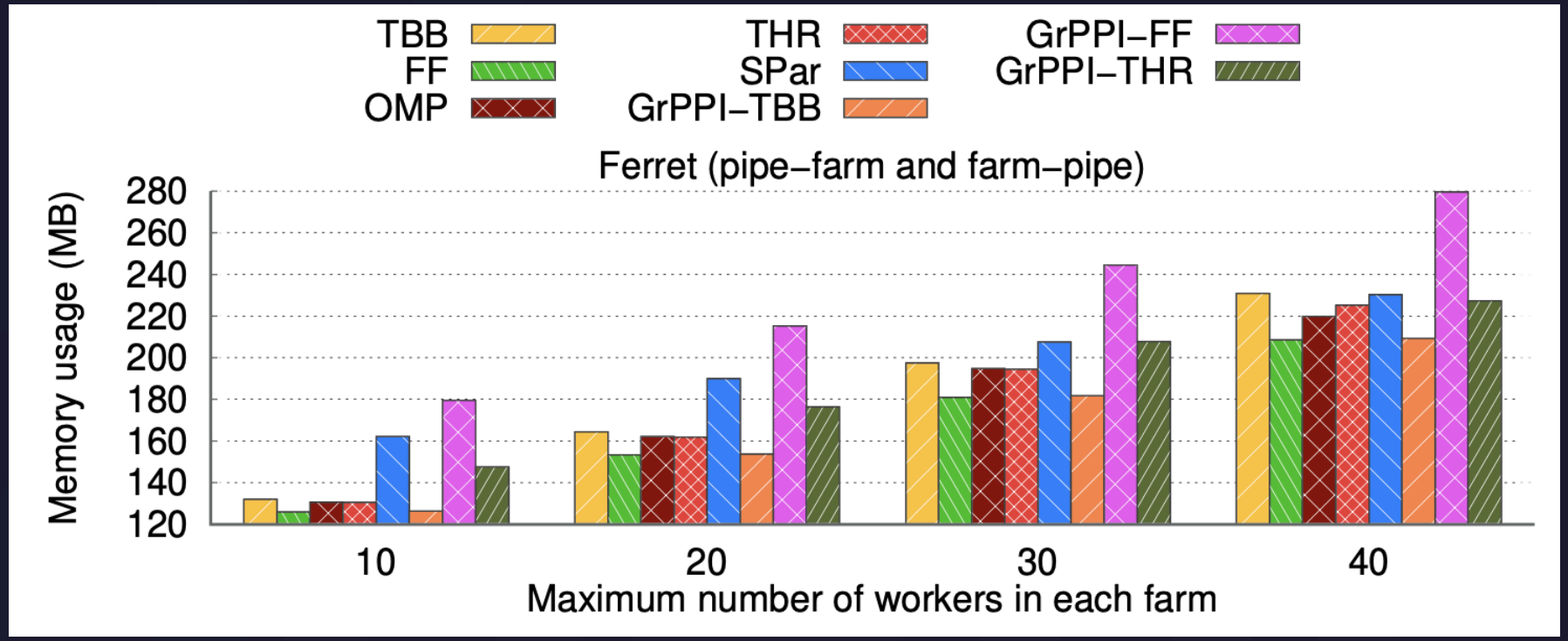
- Users who want to run performance tests with the SPBench benchmarks or custom ones.
- Researchers and developers who want to test and evaluate new technologies and solutions for parallel stream processing.
- Students and teachers who want to learn/teach stream processing.

SPBench in the literature



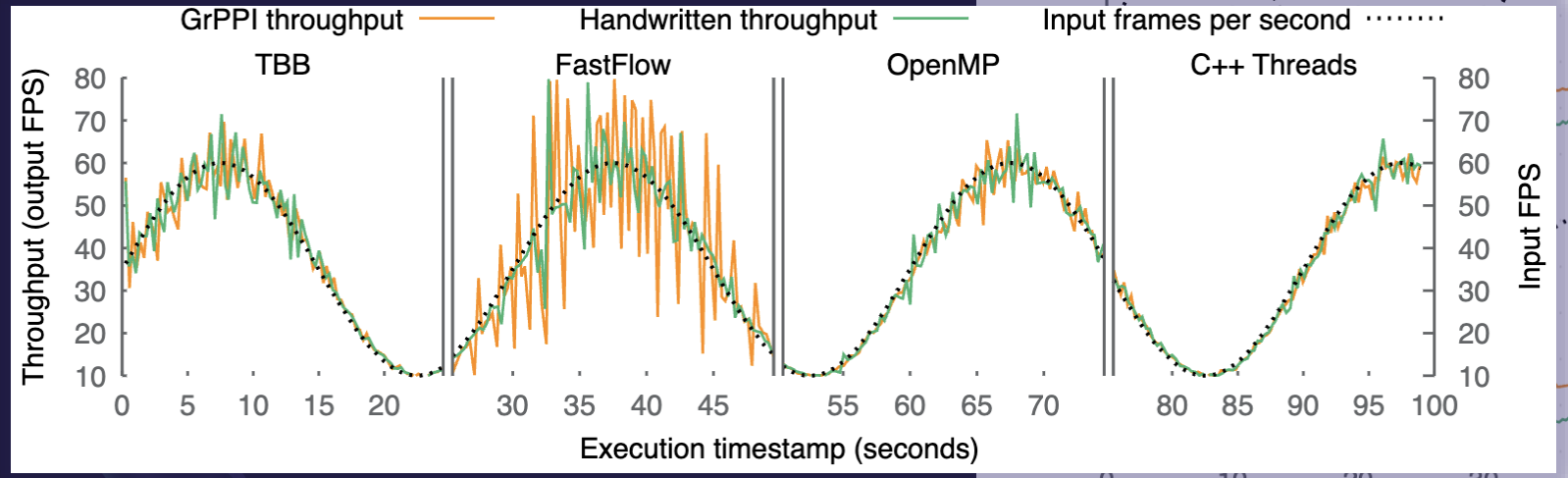
GARCIA, A. M.; GRIEBLER, D. J.; SCHEPKE, C.; FERNANDES, L. G. “SPBench: A Framework for Creating Benchmarks of Stream Processing Applications”. COMPUTING, v. 1, p. 1, 2021, doi: 10.1007/s00607-021-01025-6.

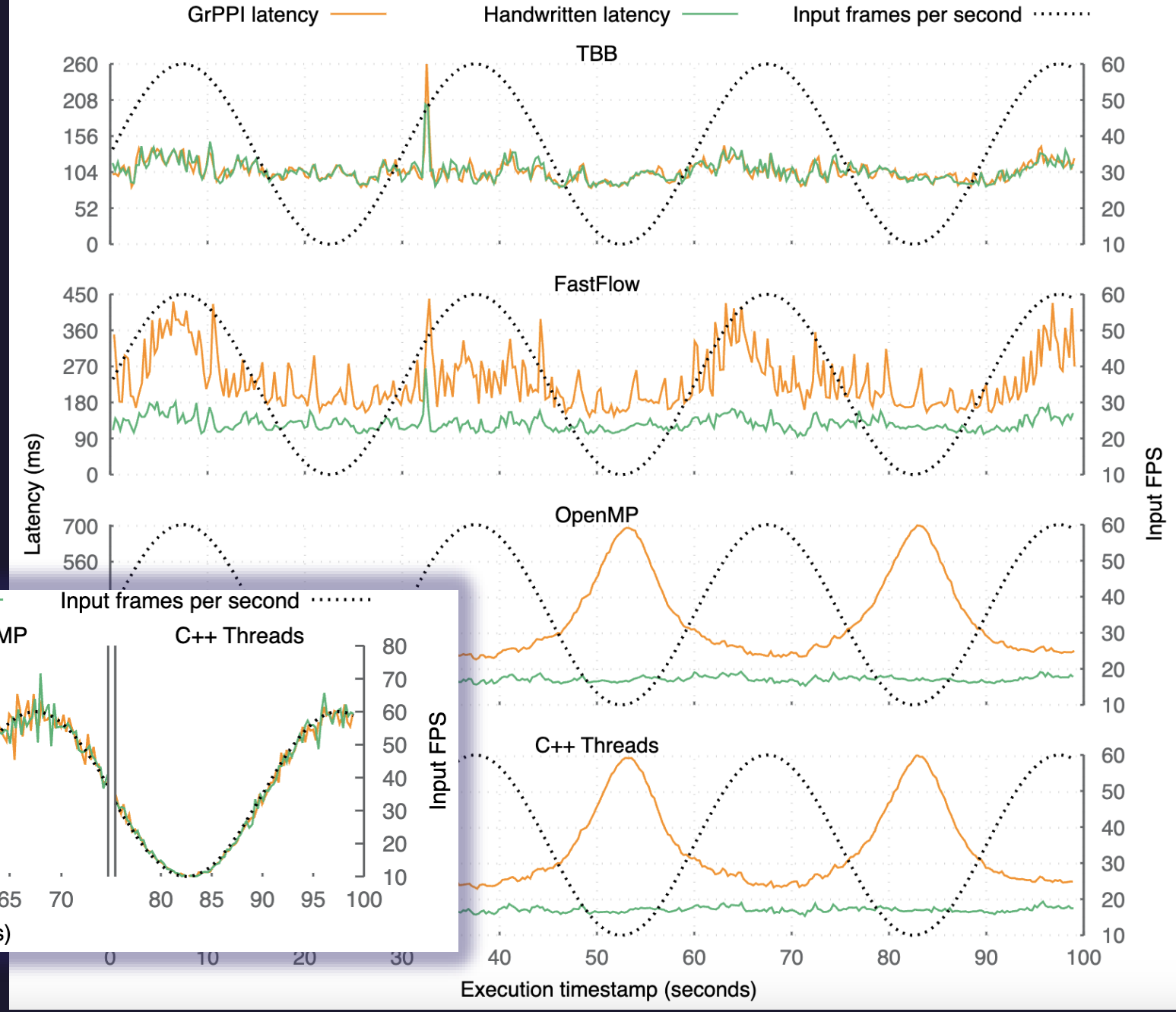
SPBench in the literature



GARCIA, A. M.; et al., "A Latency, Throughput, and Programmability Perspective of GrPPI for Streaming on Multi-cores". In: 2023 31st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), Naples, Italy, 2023, pp. 164-168, doi: 10.1109/PDP59025.2023.00033.

SPBench in the literature





GARCIA, A. M.; et al., “A Latency, Throughput, and Programmability Perspective of GrPPI for Streaming on Multi-cores”. In: 2023 31st Euromicro International Conference on Parallel, Distributed and Network-Based Processing (PDP), Naples, Italy, 2023, pp. 164-168, doi: 10.1109/PDP59025.2023.00033.

SPBench in the literature

Device	Application	Average Power Dissipation (W)					
		Seq	SPar	Fastflow	TBB	Threads	OpenMP
Jetson	Bzip2	3.11	5.43	5.48	5.46	5.44	5.45
	Ferret	3.09	5.62	5.63	5.66	5.75	5.70
	Lane	2.79	4.53	4.53	4.53	4.53	4.53
	Face	3.21	5.78	5.73	5.85	5.77	5.74
Raspberry	Bzip2	4.32	5.44	5.50	5.51	5.46	5.46
	Ferret	4.32	5.32	5.36	5.38	5.34	5.32
	Lane	4.55	5.55	5.60	5.57	5.56	5.49
	Face	4.55	5.48	5.48	5.46	5.46	5.45
Odroid	Bzip2	3.24	5.74	5.68	5.79	5.68	5.76
	Ferret	3.08	6.55	6.48	6.58	6.50	6.54
	Lane	3.30	6.65	6.63	6.76	6.85	6.88
	Face	3.15	6.14	6.17	6.31	6.29	6.33
Total		3.56	5.69	5.69	5.74	5.72	5.72

Hoffmann Filho, R. B., et al. “Impacts of parallel programming on limited-resource hardware”. Masters Dissertation. PUCRS, 2023.

Easing the Complex with SPBench

A framework for streamlining benchmarking of C++ stream processing

Public release: https://github.com/GMAP/SPBench

Author: Adriano Marques Garcia
Contact: adriano1mg@gmail.com
Website: https://alpha.di.unito.it/adriano-garcia