Pipit: Enabling Programmatic Analysis of Parallel Execution Traces

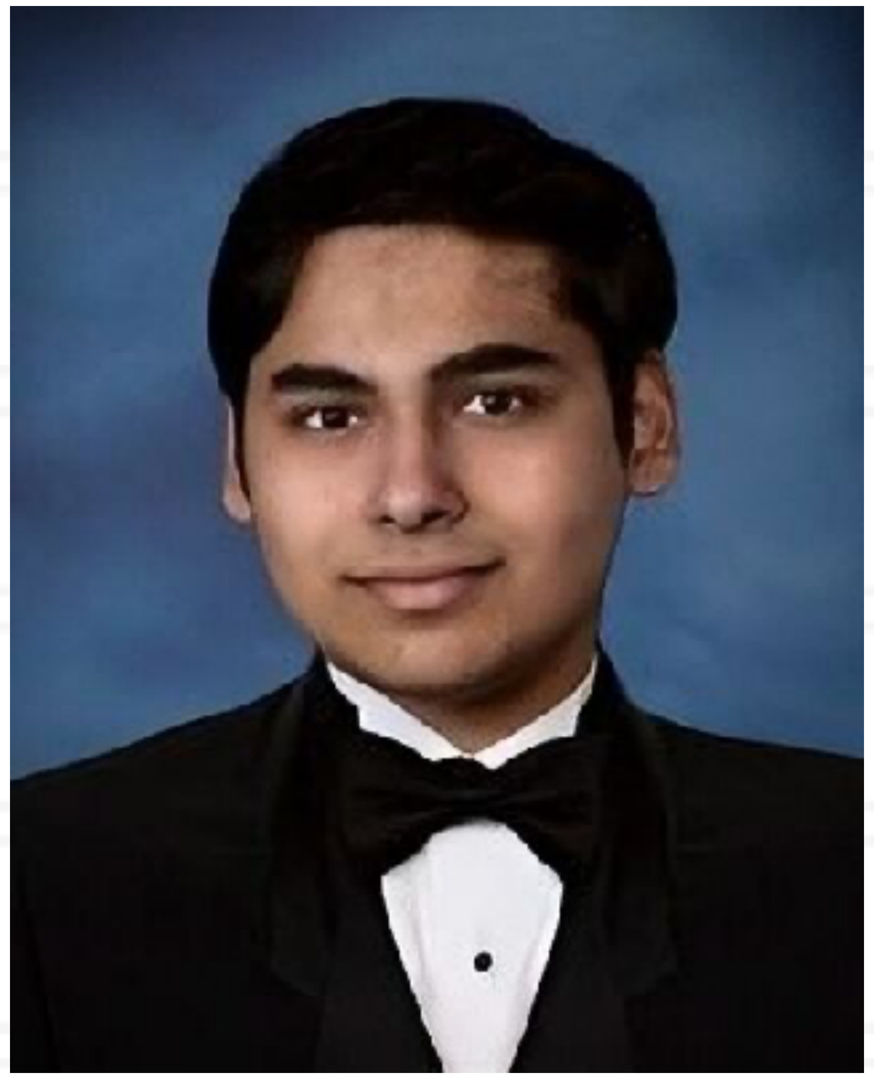
Abhinav Bhatele, Rakrish Dhakal, Alexander Movsesyan, Aditya Ranjan, Jordan Marry, Onur Cankur

Department of Computer Science, University of Maryland

PARALLEL SOFTWARE AND SYSTEMS GROUP

UNIVERSITY OF MARYLAND

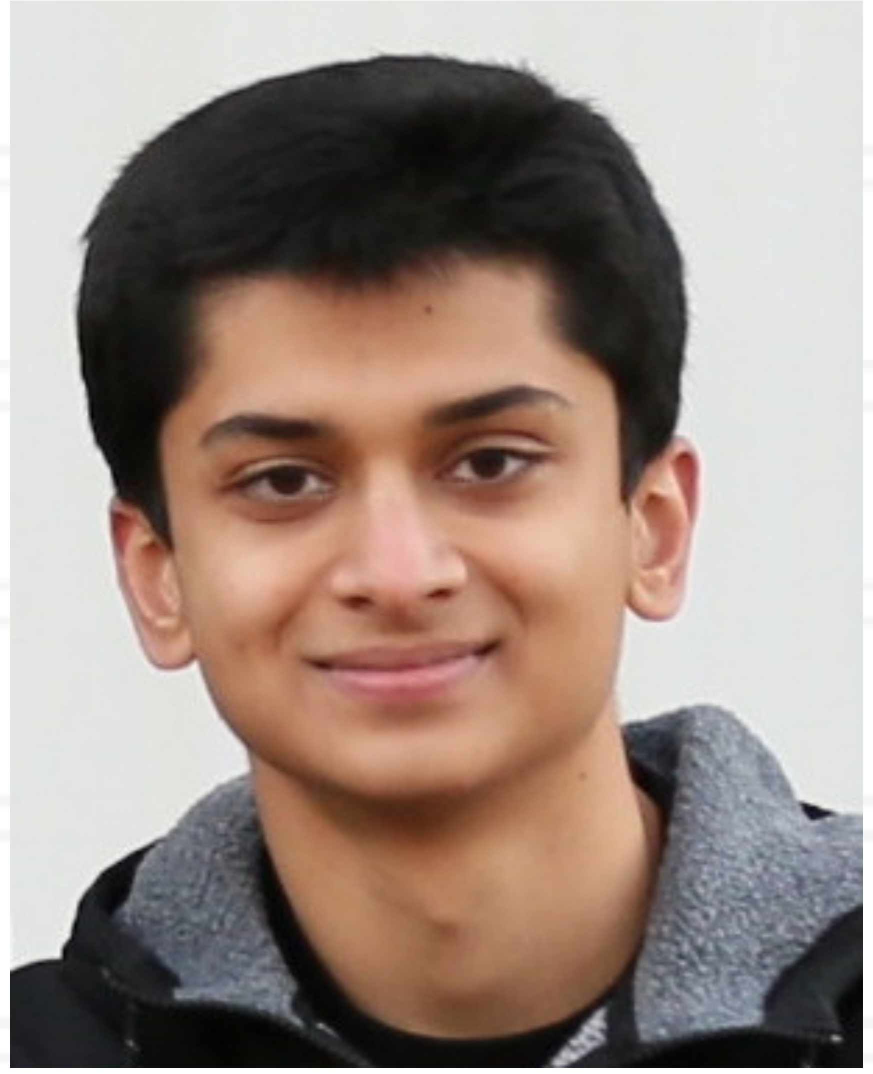
Primarily developed by UMDCS undergrads



Rakrish Dhakal

Jordan Marry

Alex Movsesyan



Aditya Ranjan

The name pipit … flicker fusion rate

- Frequency at which intermittent light stimulus appears to be completely steady
- Varies across species
- Much higher in birds of prey and passerines compared to humans (~129–137 Hz. vs. 60–75 Hz.)

https://www.sciencedaily.com/releases/2016/03/160318144548.htm

Nilgiri Pipit

The name pipit … flicker fusion rate

- Frequency at which intermittent light stimulus appears to be completely steady
- Varies across species
- Much higher in birds of prey and passerines compared to humans (~129–137 Hz. vs. 60–75 Hz.)

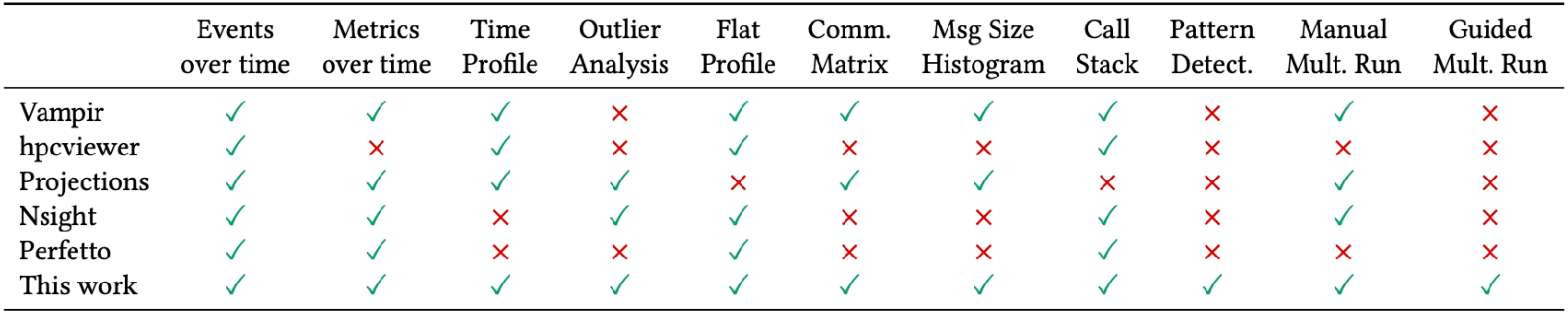
https://www.sciencedaily.com/releases/2016/03/160318144548.htm

Certain things can only be done with traces

- Analyzing utilization over time
- Messaging dependencies, critical paths
- Studying overlap of communication and computation
- Other time series analysis: repeating patterns, …

Limitations of current tools

- Each tool supports specific file formats
- Scripting and visualization are typically separate
- Easy comparisons of multiple executions are missing



	Events over time	Metrics over time	Time Profile	Outlier Analysis	Flat Profile	Comm. Matrix	Msg Size Histogram	Call Stack	Pattern Detect.	Manual Mult. Run	Guided Mult. Run
Vampir	✓	✓	✓	✗	✓	✓	✓	✓	✗	✓	✗
hpcviewer	✓	✗	✓	✗	✓	✗	✗	✓	✗	✗	✗
Projections	✓	✓	✓	✓	✗	✓	✓	✗	✗	✓	✗
Nsight	✓	✓	✗	✓	✓	✗	✗	✓	✗	✓	✗
Perfetto	✓	✓	✗	✗	✓	✗	✗	✓	✗	✗	✗
This work	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓

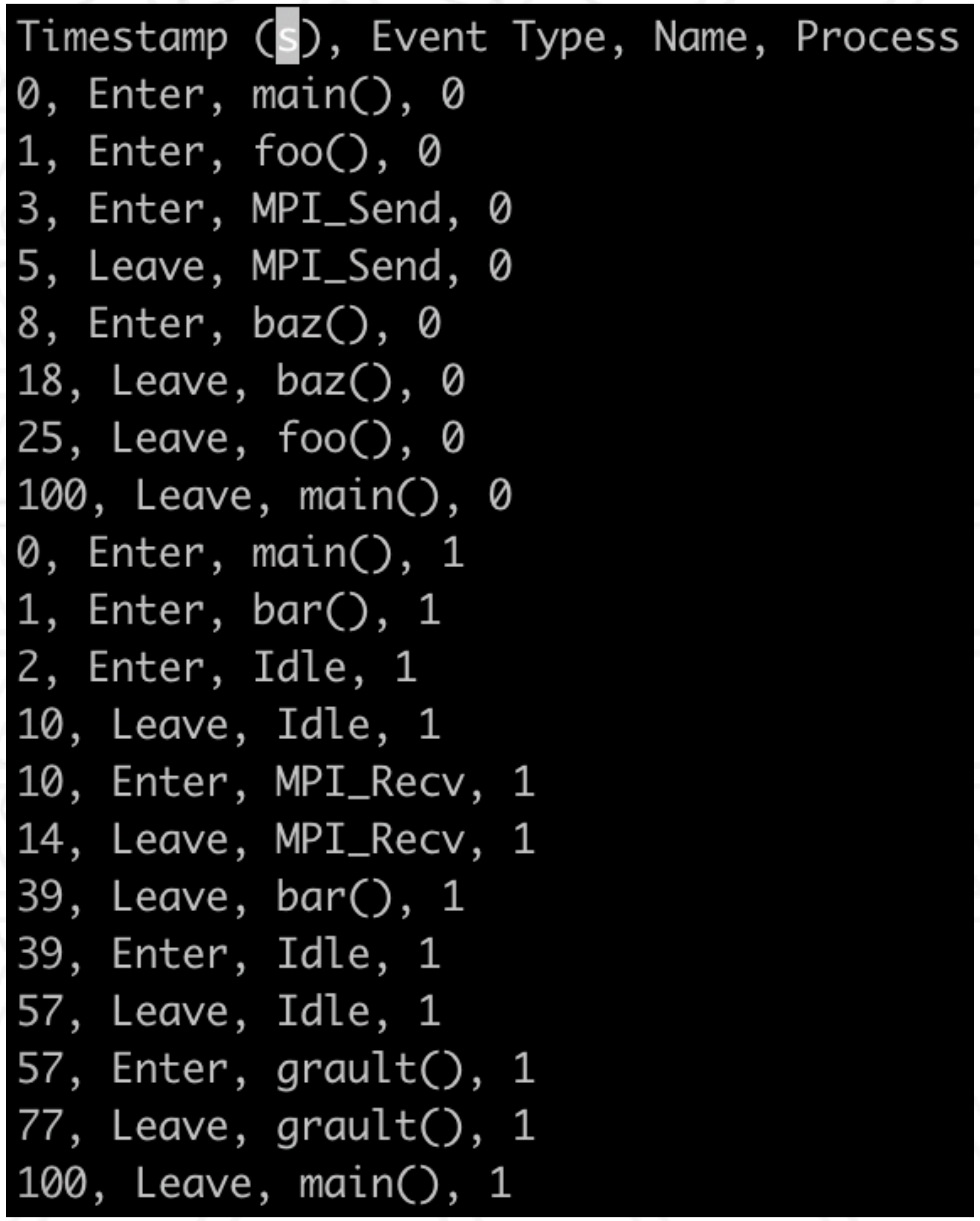
Goal: Scripting + visualization for traces

- Programmatic analysis of parallel execution traces
- Support a variety of file formats
- Provide basic operations to ingest/explore/reduce data
- Provide advanced operations to find scalability issues
- Support multi-run analysis

Pipit was hatched

- A Python-based library that uses pandas
- Load traces into a pandas dataframe
- Set of operators to calculate basic things such as caller-callee relationships, inclusive metrics, exclusive metrics, …
- Other operators to analyze overall performance, communication performance,
- Filter the trace to a more manageable size

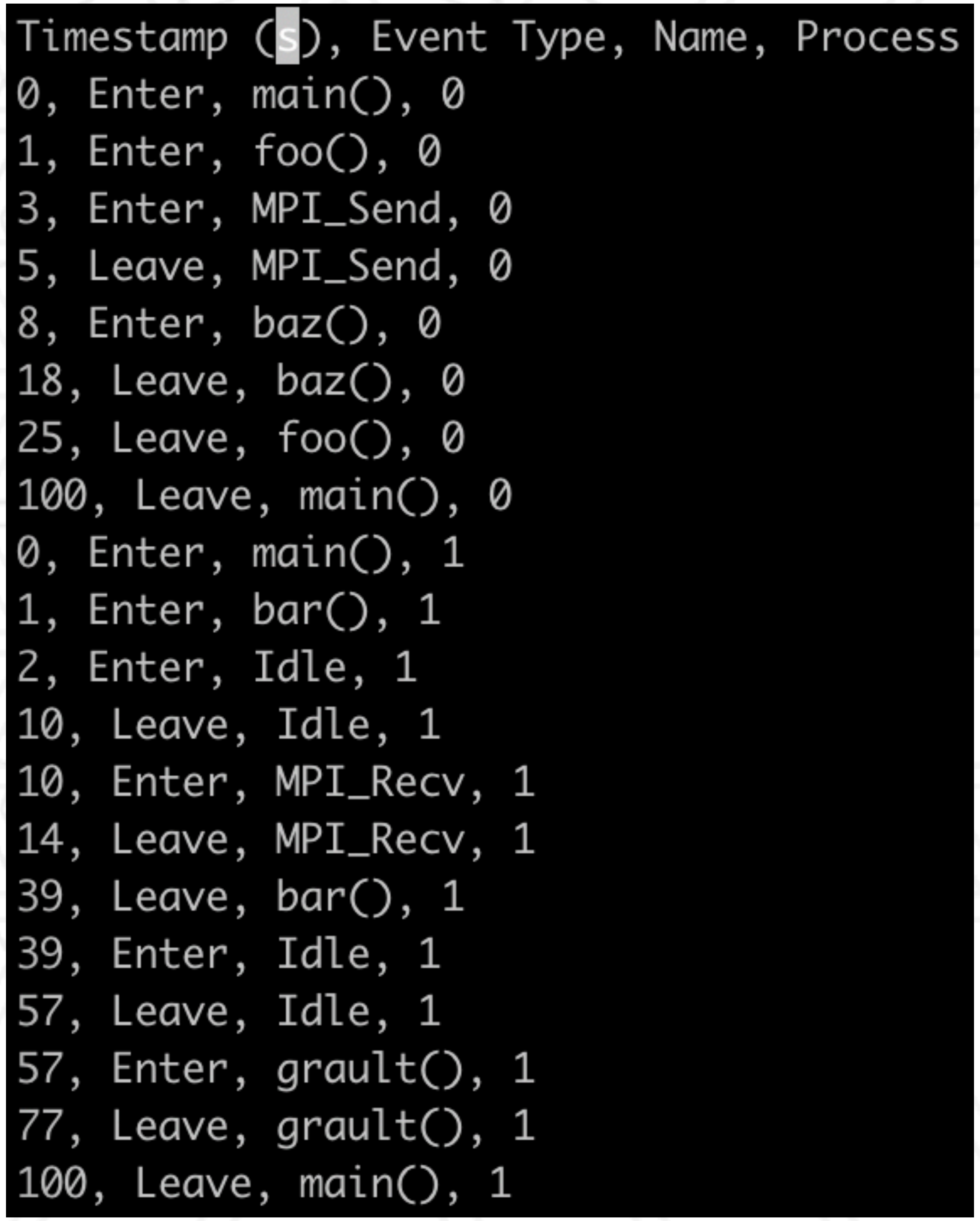
Data structures in pipit



```
Timestamp (s), Event Type, Name, Process
0, Enter, main(), 0
1, Enter, foo(), 0
3, Enter, MPI_Send, 0
5, Leave, MPI_Send, 0
8, Enter, baz(), 0
18, Leave, baz(), 0
25, Leave, foo(), 0
100, Leave, main(), 0
0, Enter, main(), 1
1, Enter, bar(), 1
2, Enter, Idle, 1
10, Leave, Idle, 1
10, Enter, MPI_Recv, 1
14, Leave, MPI_Recv, 1
39, Leave, bar(), 1
39, Enter, Idle, 1
57, Leave, Idle, 1
57, Enter, grault(), 1
77, Leave, grault(), 1
100, Leave, main(), 1
```

Data structures in pipit

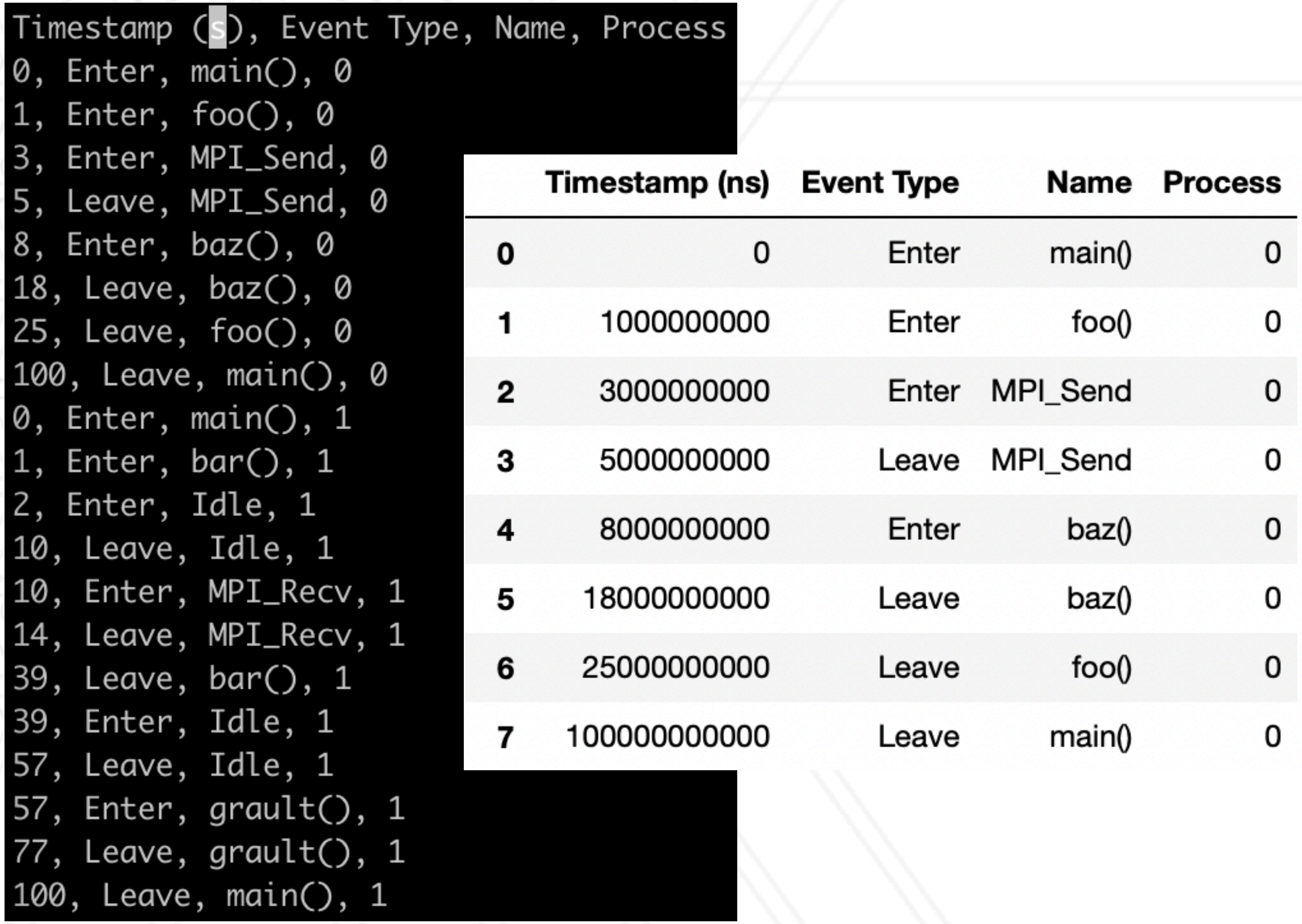
- (Events, timestamps) X (processes, threads) X (performance metrics)
- Store as dataframe
- Compute call graph aggregated over time and processes



```
Timestamp (s), Event Type, Name, Process
0, Enter, main(), 0
1, Enter, foo(), 0
3, Enter, MPI_Send, 0
5, Leave, MPI_Send, 0
8, Enter, baz(), 0
18, Leave, baz(), 0
25, Leave, foo(), 0
100, Leave, main(), 0
0, Enter, main(), 1
1, Enter, bar(), 1
2, Enter, Idle, 1
10, Leave, Idle, 1
10, Enter, MPI_Recv, 1
14, Leave, MPI_Recv, 1
39, Leave, bar(), 1
39, Enter, Idle, 1
57, Leave, Idle, 1
57, Enter, grault(), 1
77, Leave, grault(), 1
100, Leave, main(), 1
```

Data structures in pipit

- (Events, timestamps) X (processes, threads) X (performance metrics)
- Store as dataframe
- Compute call graph aggregated over time and processes



```
Timestamp (s), Event Type, Name, Process
0, Enter, main(), 0
1, Enter, foo(), 0
3, Enter, MPI_Send, 0
5, Leave, MPI_Send, 0
8, Enter, baz(), 0
18, Leave, baz(), 0
25, Leave, foo(), 0
100, Leave, main(), 0
0, Enter, main(), 1
1, Enter, bar(), 1
2, Enter, Idle, 1
10, Leave, Idle, 1
10, Enter, MPI_Recv, 1
14, Leave, MPI_Recv, 1
39, Leave, bar(), 1
39, Enter, Idle, 1
57, Leave, Idle, 1
57, Enter, grault(), 1
77, Leave, grault(), 1
100, Leave, main(), 1
```

	Timestamp (ns)	Event Type	Name	Process
0	0	Enter	main()	0
1	1000000000	Enter	foo()	0
2	3000000000	Enter	MPI_Send	0
3	5000000000	Leave	MPI_Send	0
4	8000000000	Enter	baz()	0
5	18000000000	Leave	baz()	0
6	25000000000	Leave	foo()	0
7	100000000000	Leave	main()	0

Reading in a trace dataset

- Pipit is available on GitHub:

http://github.com/hpcgroup/pipit

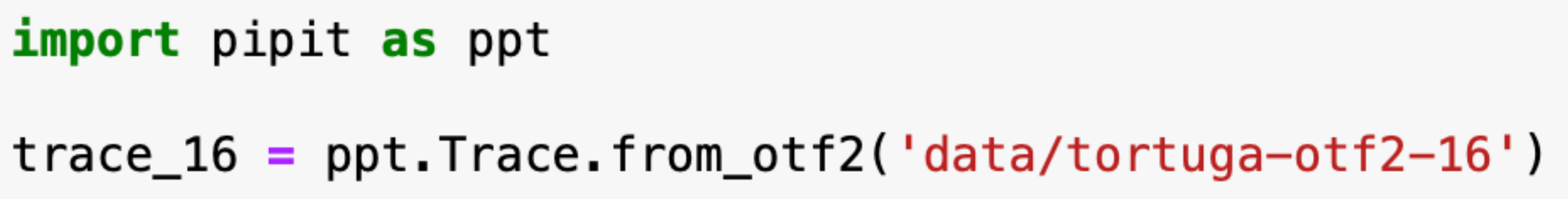
- We support several file formats already: OTF2, HPCToolkit (new format), Projections, Nsight (basic), CSV, …

Reading in a trace dataset

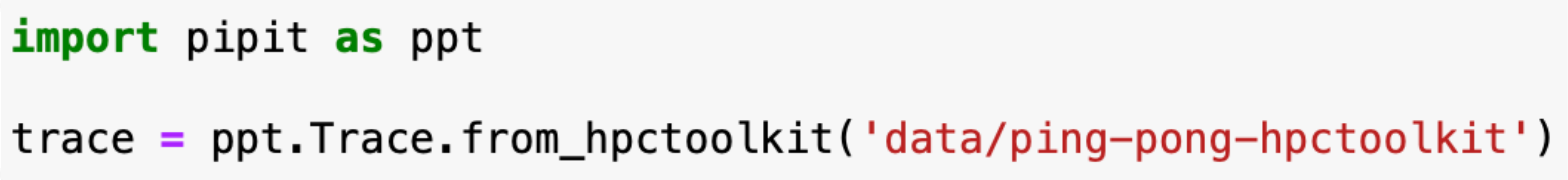
- Pipit is available on GitHub:

http://github.com/hpcgroup/pipit

- We support several file formats already: OTF2, HPCToolkit (new format), Projections, Nsight (basic), CSV, …



```
import pipit as ppt

trace_16 = ppt.Trace.from_otf2('data/tortuga-otf2-16')
```



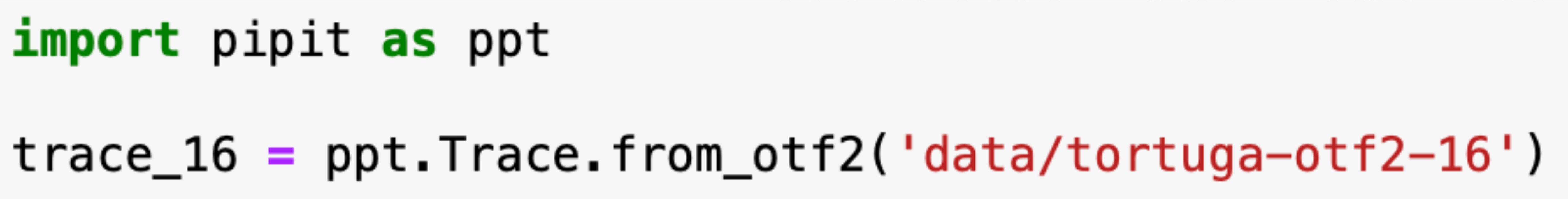
```
import pipit as ppt

trace = ppt.Trace.from_hpctoolkit('data/ping-pong-hpctoolkit')
```

Reading in a trace dataset

- Pipit is available on GitHub:

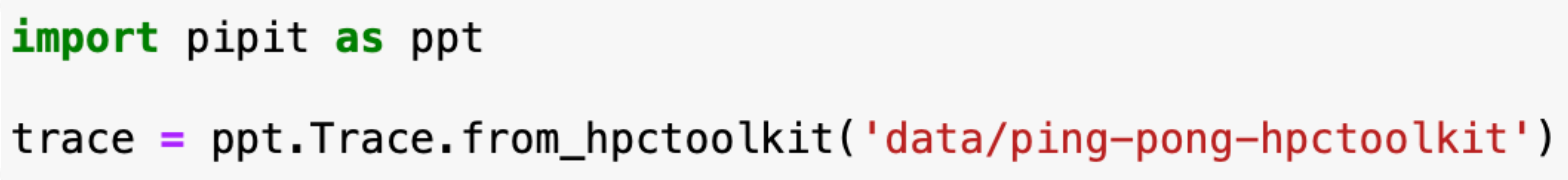
http://github.com/hpcgroup/pipit

- We support several file formats already: OTF2, HPCToolkit (new format), Projections, Nsight (basic), CSV, …



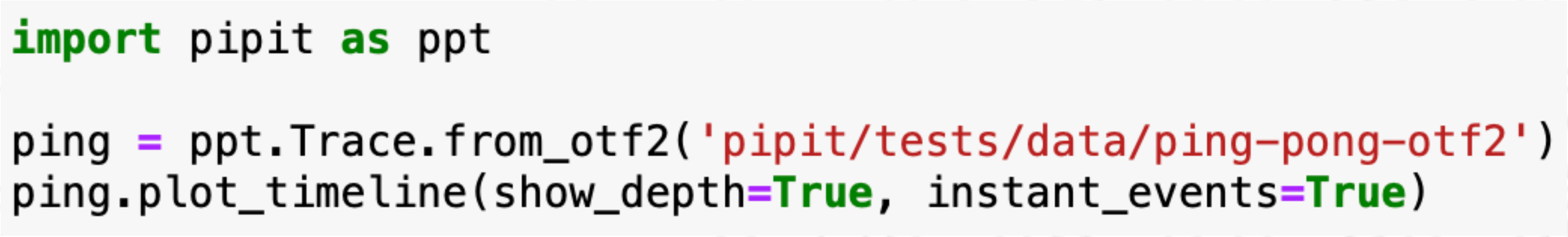
```
import pipit as ppt

trace_16 = ppt.Trace.from_otf2('data/tortuga-otf2-16')
```

Contributions Welcome!



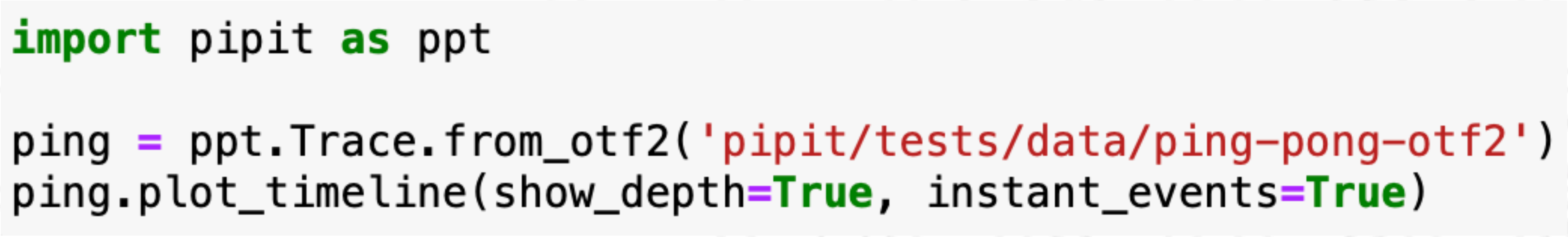
```
import pipit as ppt

trace = ppt.Trace.from_hpctoolkit('data/ping-pong-hpctoolkit')
```

Start exploring …

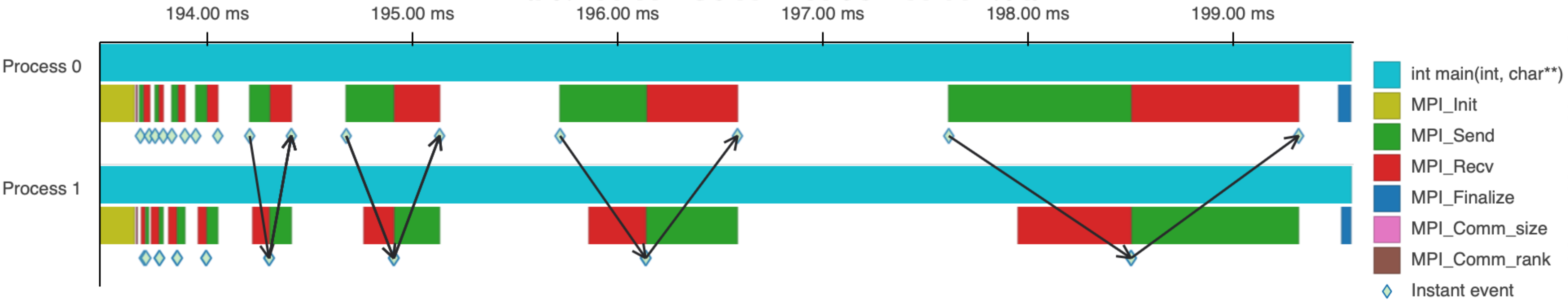


```
import pipit as ppt

ping = ppt.Trace.from_otf2('pipit/tests/data/ping-pong-otf2')
ping.plot_timeline(show_depth=True, instant_events=True)
```

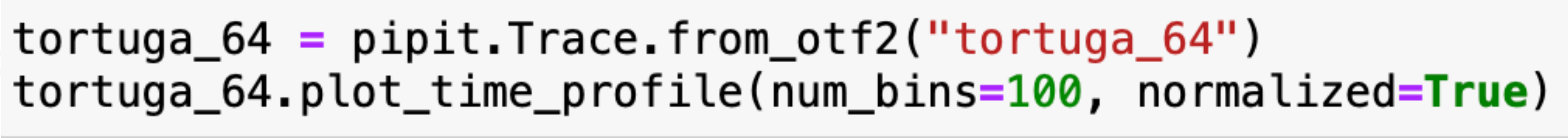
Start exploring …



```python
import pipit as ppt

ping = ppt.Trace.from_otf2('pipit/tests/data/ping-pong-otf2')
ping.plot_timeline(show_depth=True, instant_events=True)
```

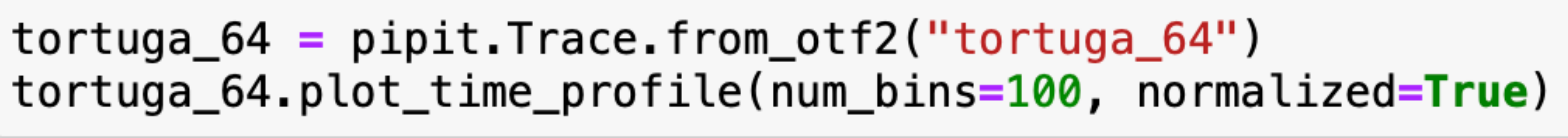


Time profile / utilization

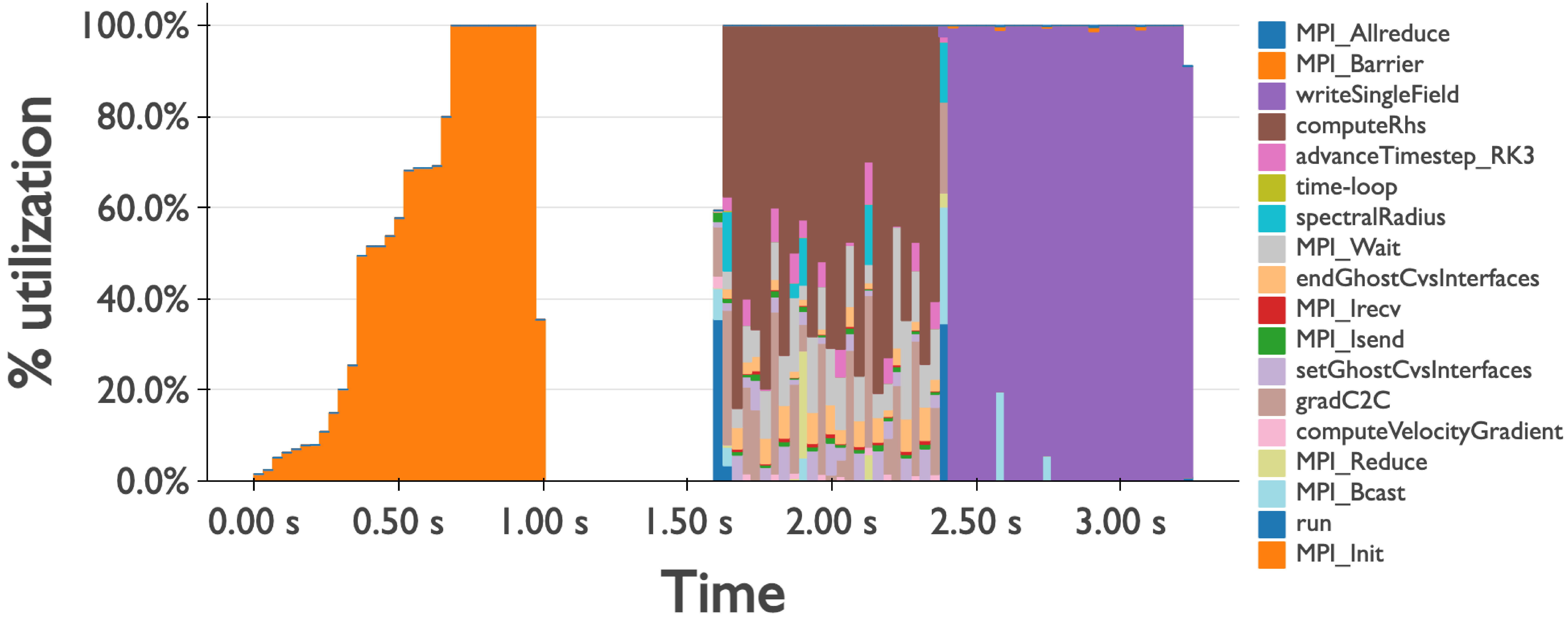


```
tortuga_64 = pipit.Trace.from_otf2("tortuga_64")
tortuga_64.plot_time_profile(num_bins=100, normalized=True)
```

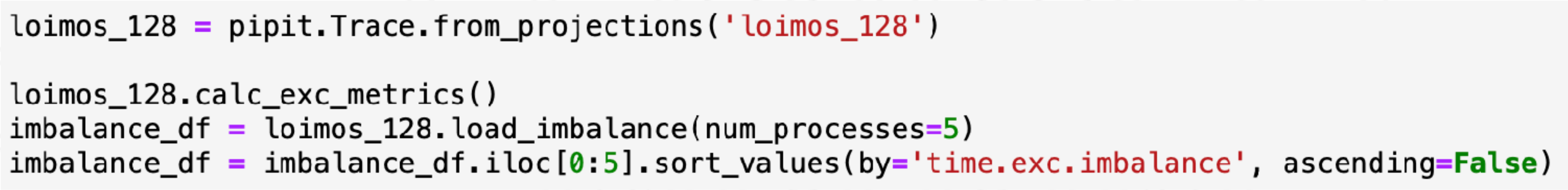
Time profile / utilization



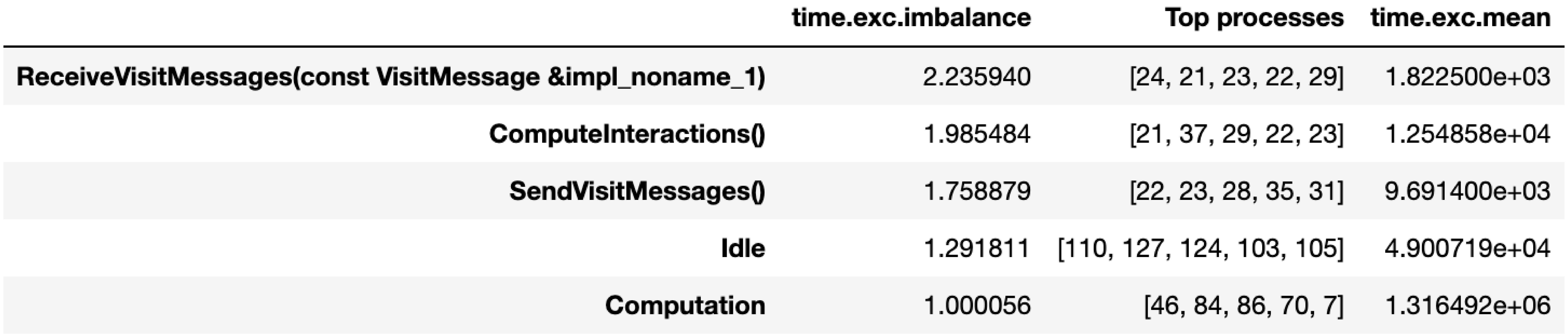
```python
tortuga_64 = pipit.Trace.from_otf2("tortuga_64")
tortuga_64.plot_time_profile(num_bins=100, normalized=True)
```



Load imbalance

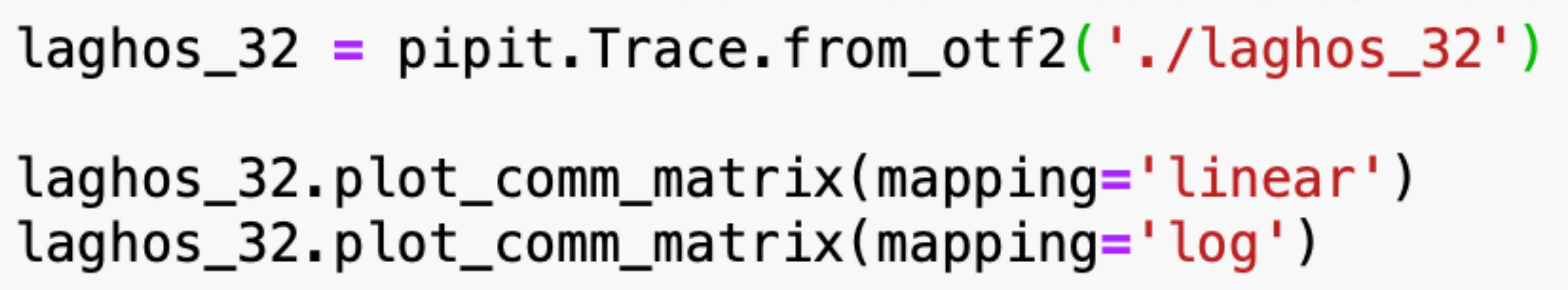


```
loimos_128 = pipit.Trace.from_projections('loimos_128')

loimos_128.calc_exc_metrics()
imbalance_df = loimos_128.load_imbalance(num_processes=5)
imbalance_df = imbalance_df.iloc[0:5].sort_values(by='time.exc.imbalance', ascending=False)
```

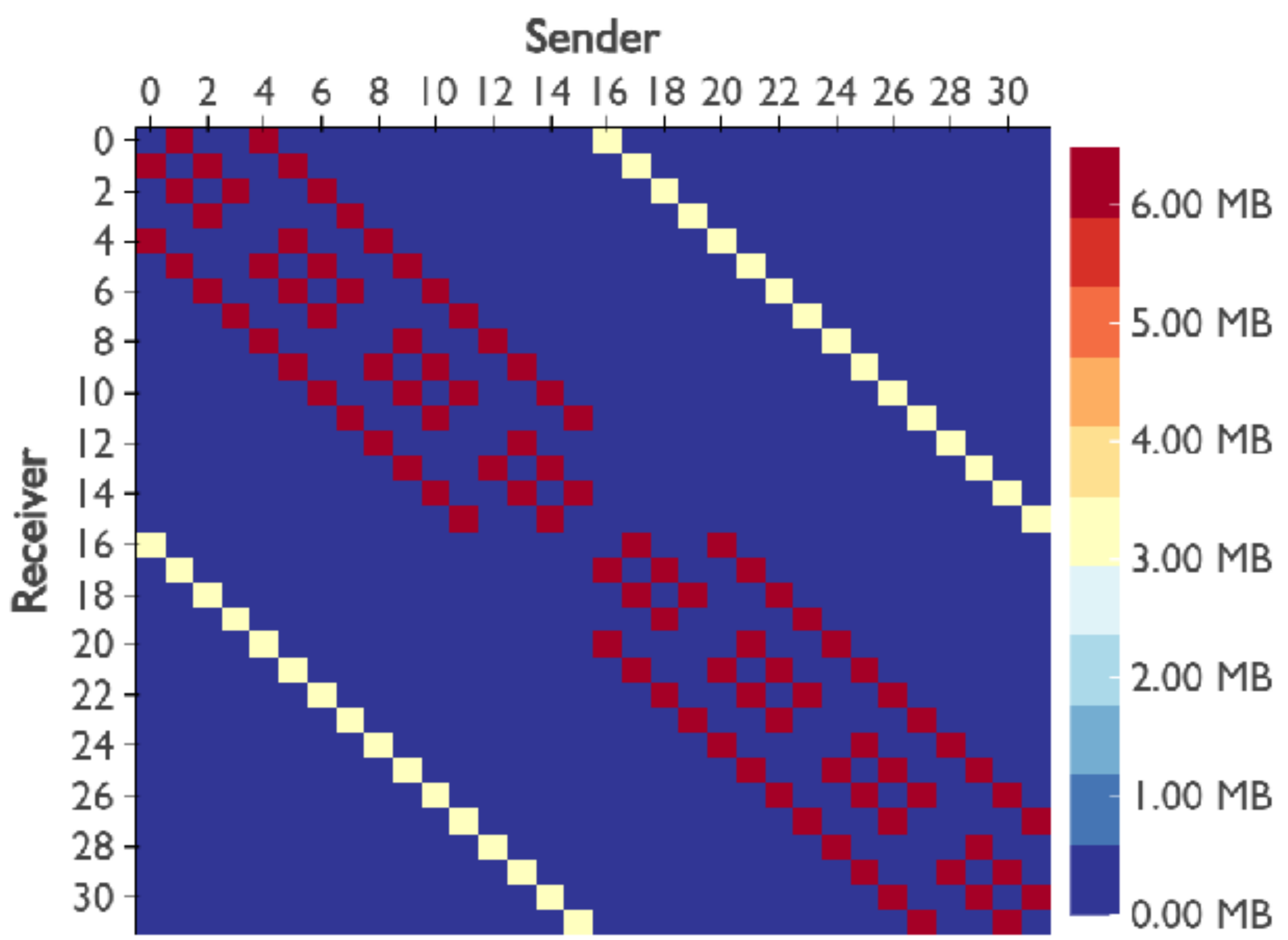


	time.exc.imbalance	Top processes	time.exc.mean
ReceiveVisitMessages(const VisitMessage &impl_noname_1)	2.235940	[24, 21, 23, 22, 29]	1.822500e+03
ComputeInteractions()	1.985484	[21, 37, 29, 22, 23]	1.254858e+04
SendVisitMessages()	1.758879	[22, 23, 28, 35, 31]	9.691400e+03
Idle	1.291811	[110, 127, 124, 103, 105]	4.900719e+04
Computation	1.000056	[46, 84, 86, 70, 7]	1.316492e+06

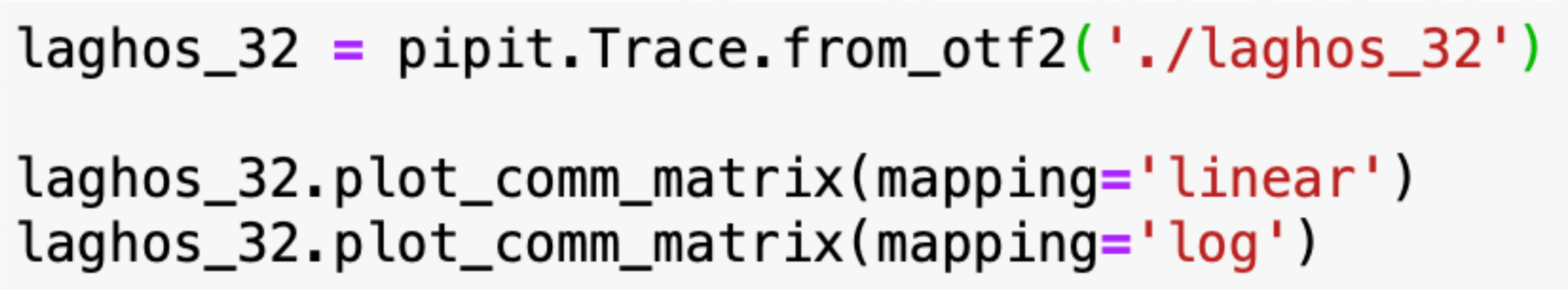
Communication analysis



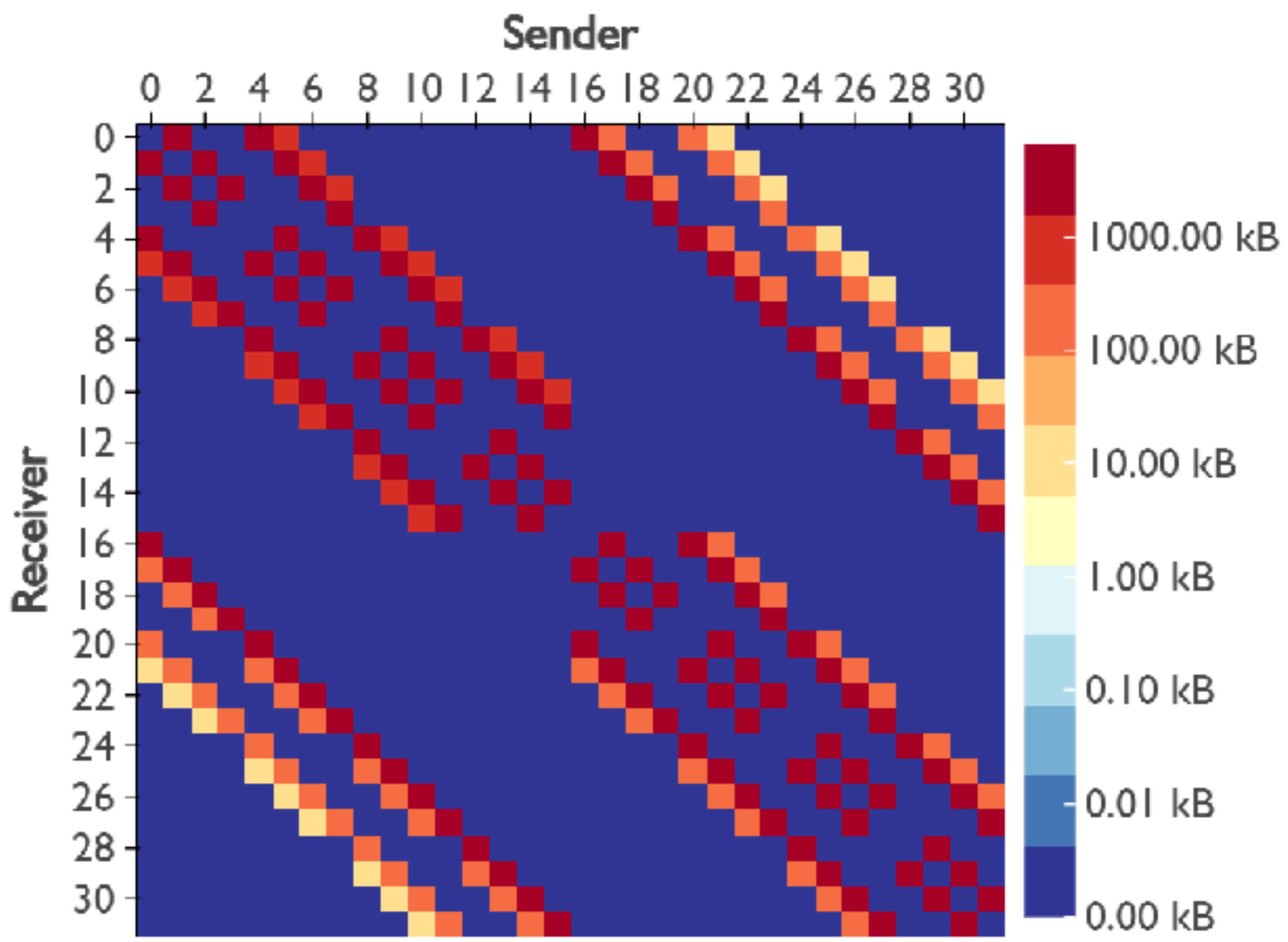
```
laghos_32 = pipit.Trace.from_otf2('./laghos_32')

laghos_32.plot_comm_matrix(mapping='linear')
laghos_32.plot_comm_matrix(mapping='log')
```



Communication analysis

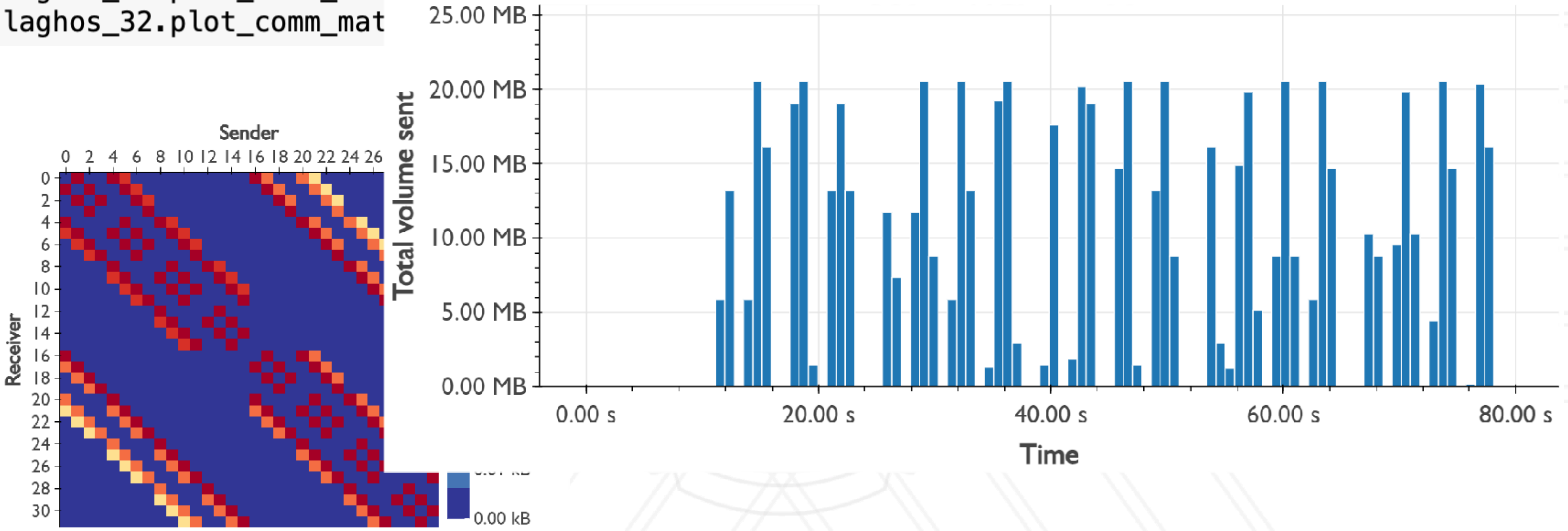


```
laghos_32 = pipit.Trace.from_otf2('./laghos_32')

laghos_32.plot_comm_matrix(mapping='linear')
laghos_32.plot_comm_matrix(mapping='log')
```



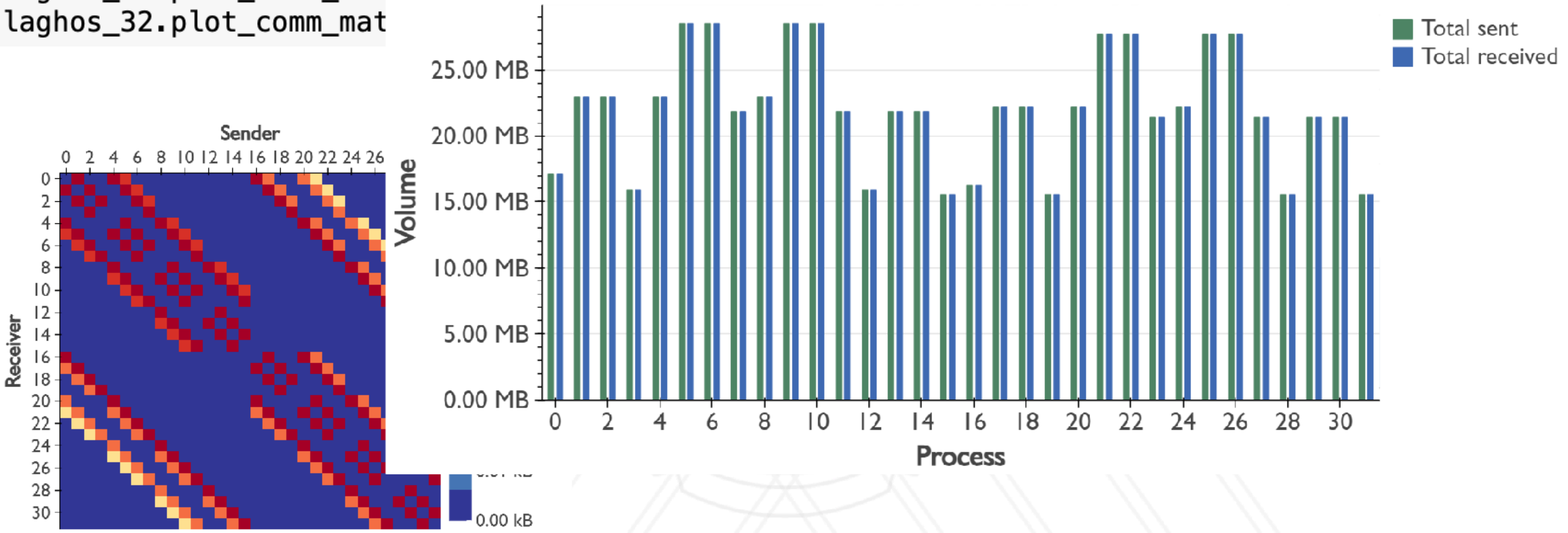
Communication analysis

```
laghos_32 = pipit.Trace.from_otf2('./laghos_32')

laghos_32.plot_comm_mat
laghos_32.plot_comm_mat
```

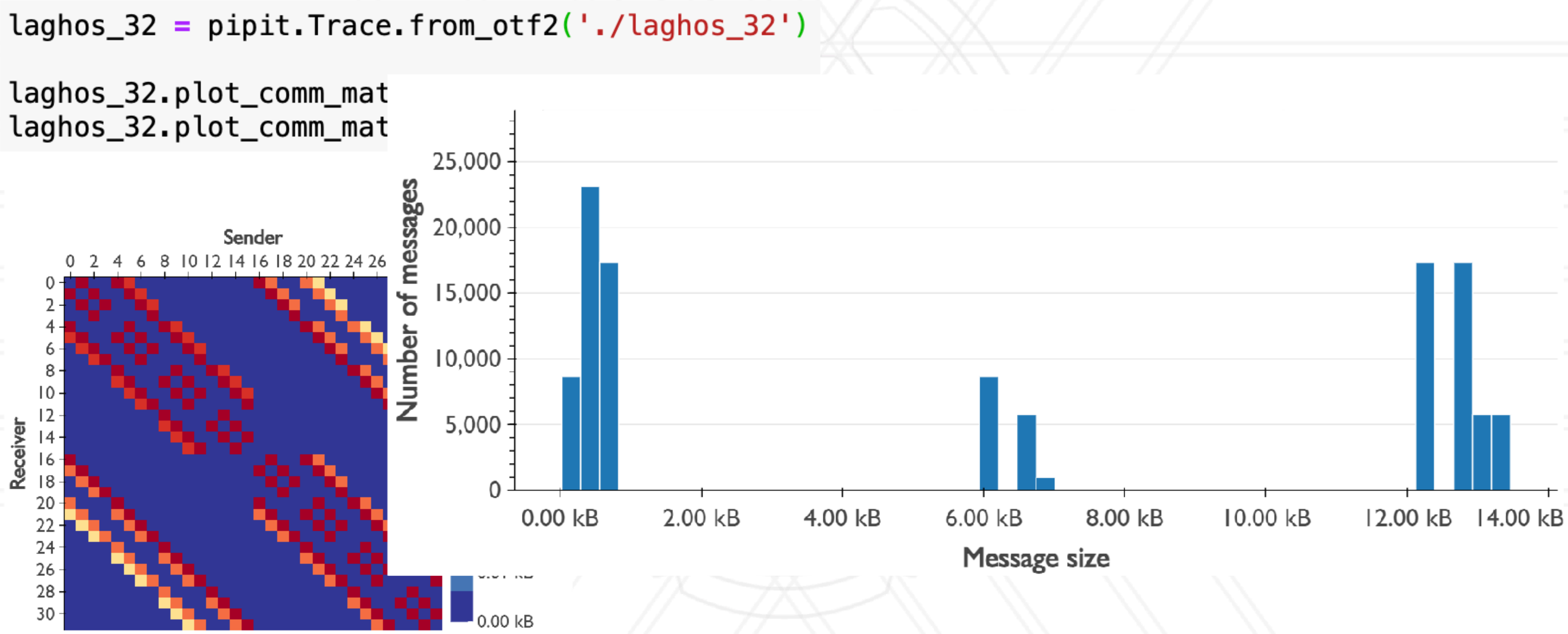


Communication analysis

```
laghos_32 = pipit.Trace.from_otf2('./laghos_32')

laghos_32.plot_comm_mat
laghos_32.plot_comm_mat
```

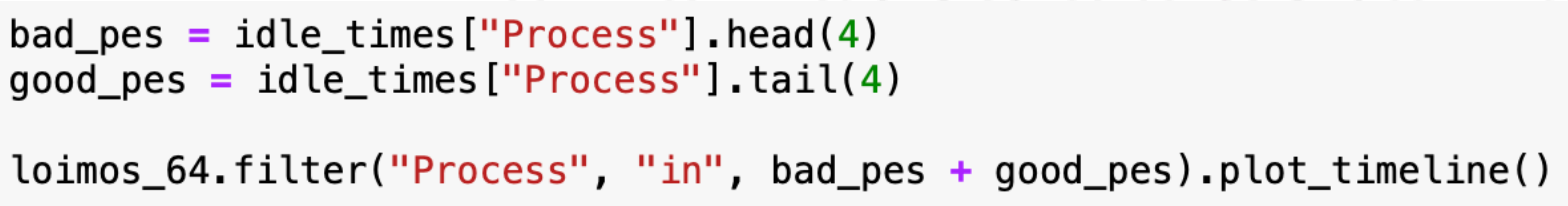


Communication analysis



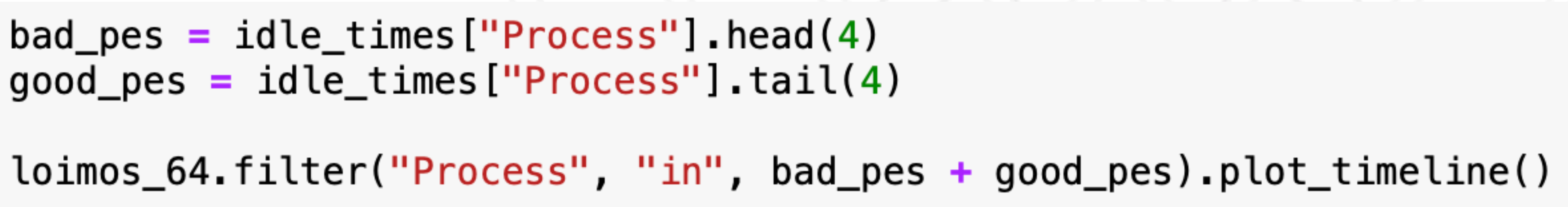
```
laghos_32 = pipit.Trace.from_otf2('./laghos_32')

laghos_32.plot_comm_mat
laghos_32.plot_comm_mat
```

Data reduction / filtering



```
bad_pes = idle_times["Process"].head(4)
good_pes = idle_times["Process"].tail(4)

loimos_64.filter("Process", "in", bad_pes + good_pes).plot_timeline()
```

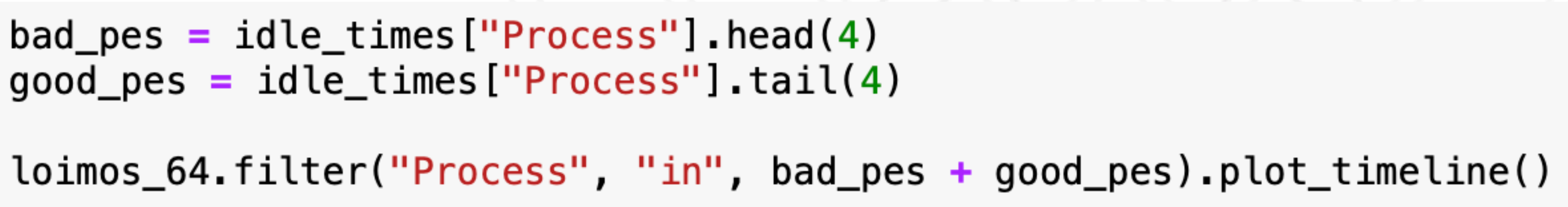
Data reduction / filtering



```
bad_pes = idle_times["Process"].head(4)
good_pes = idle_times["Process"].tail(4)

loimos_64.filter("Process", "in", bad_pes + good_pes).plot_timeline()
```

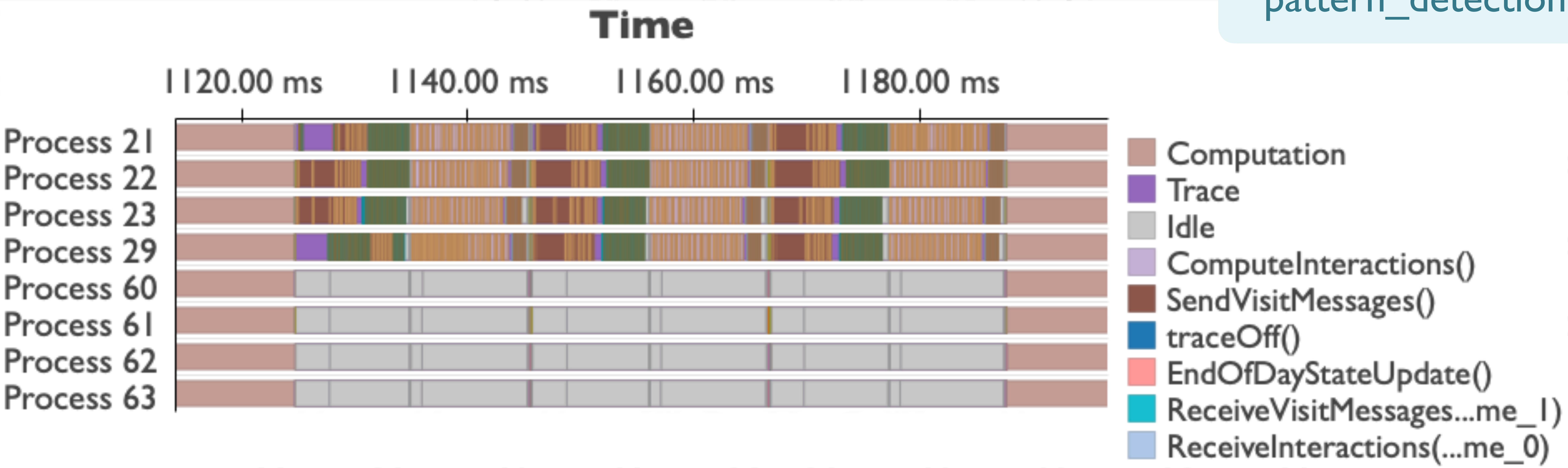
idle_time()
outlier_detection()
pattern_detection()

Data reduction / filtering

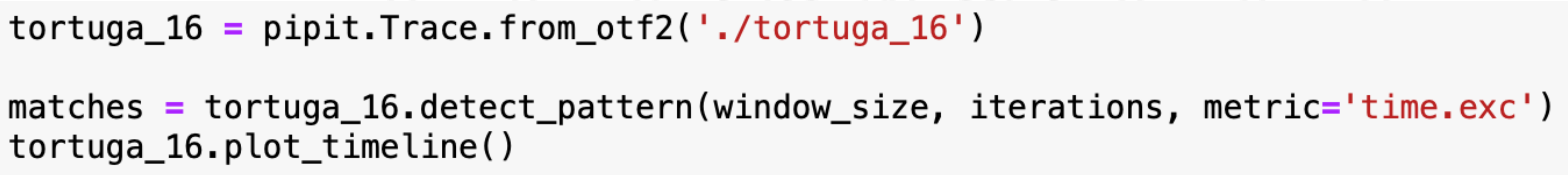


```
bad_pes = idle_times["Process"].head(4)
good_pes = idle_times["Process"].tail(4)

loimos_64.filter("Process", "in", bad_pes + good_pes).plot_timeline()
```

idle_time()
outlier_detection()
pattern_detection()

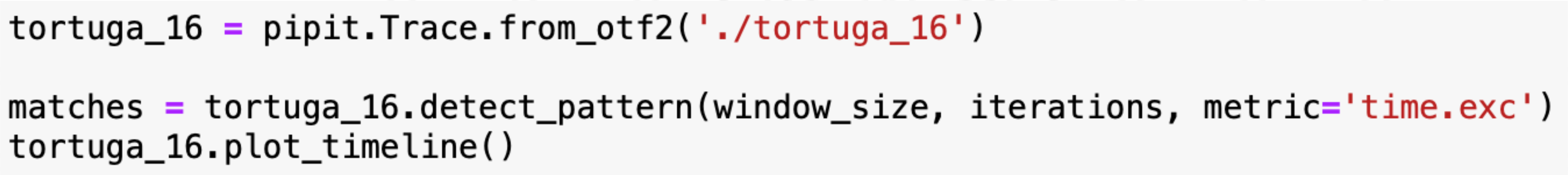


Pattern detection

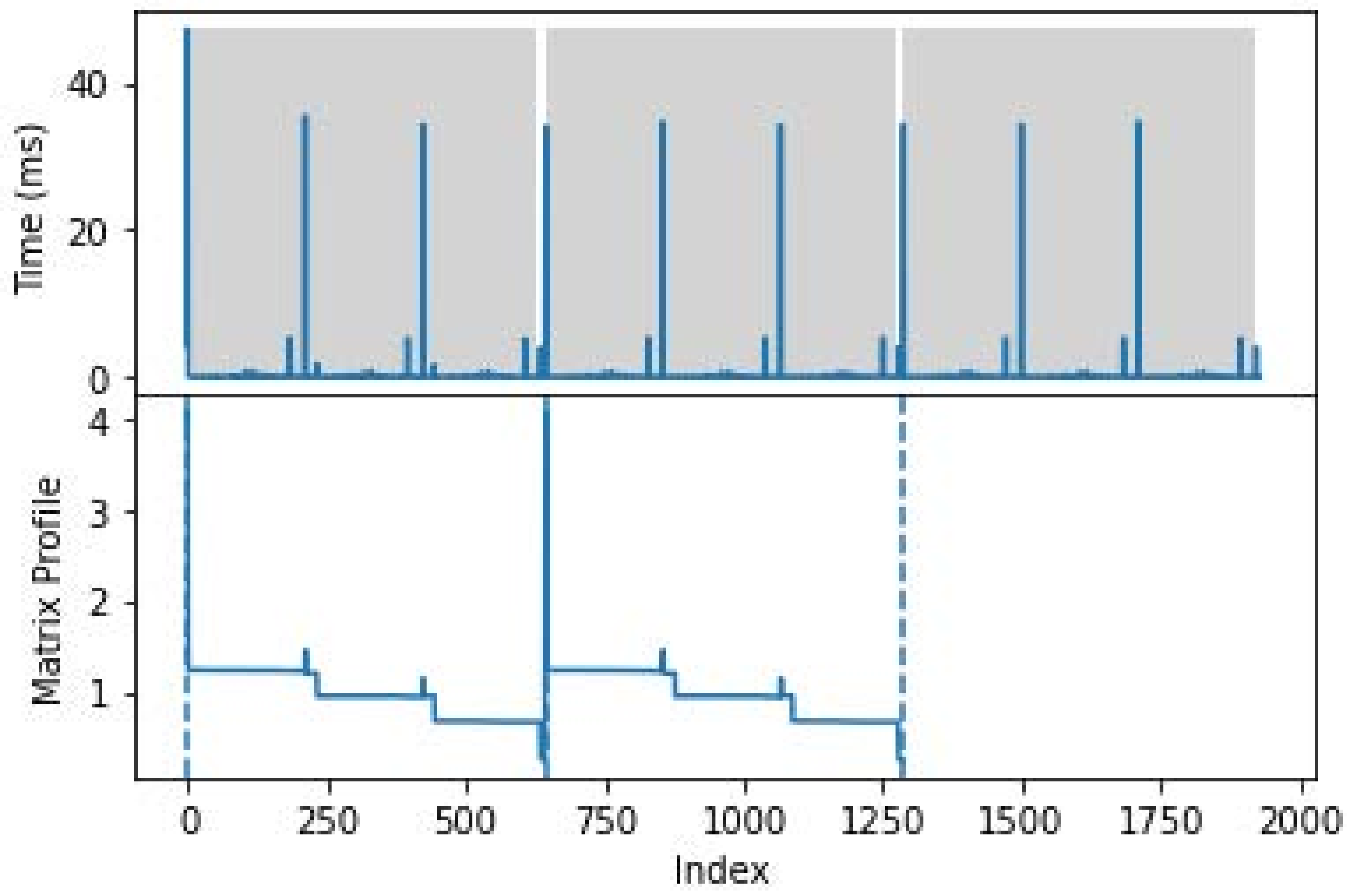


```python
tortuga_16 = pipit.Trace.from_otf2('./tortuga_16')

matches = tortuga_16.detect_pattern(window_size, iterations, metric='time.exc')
tortuga_16.plot_timeline()
```

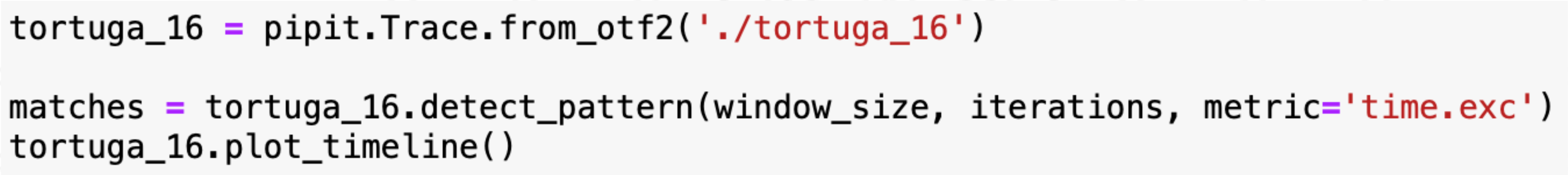
Pattern detection



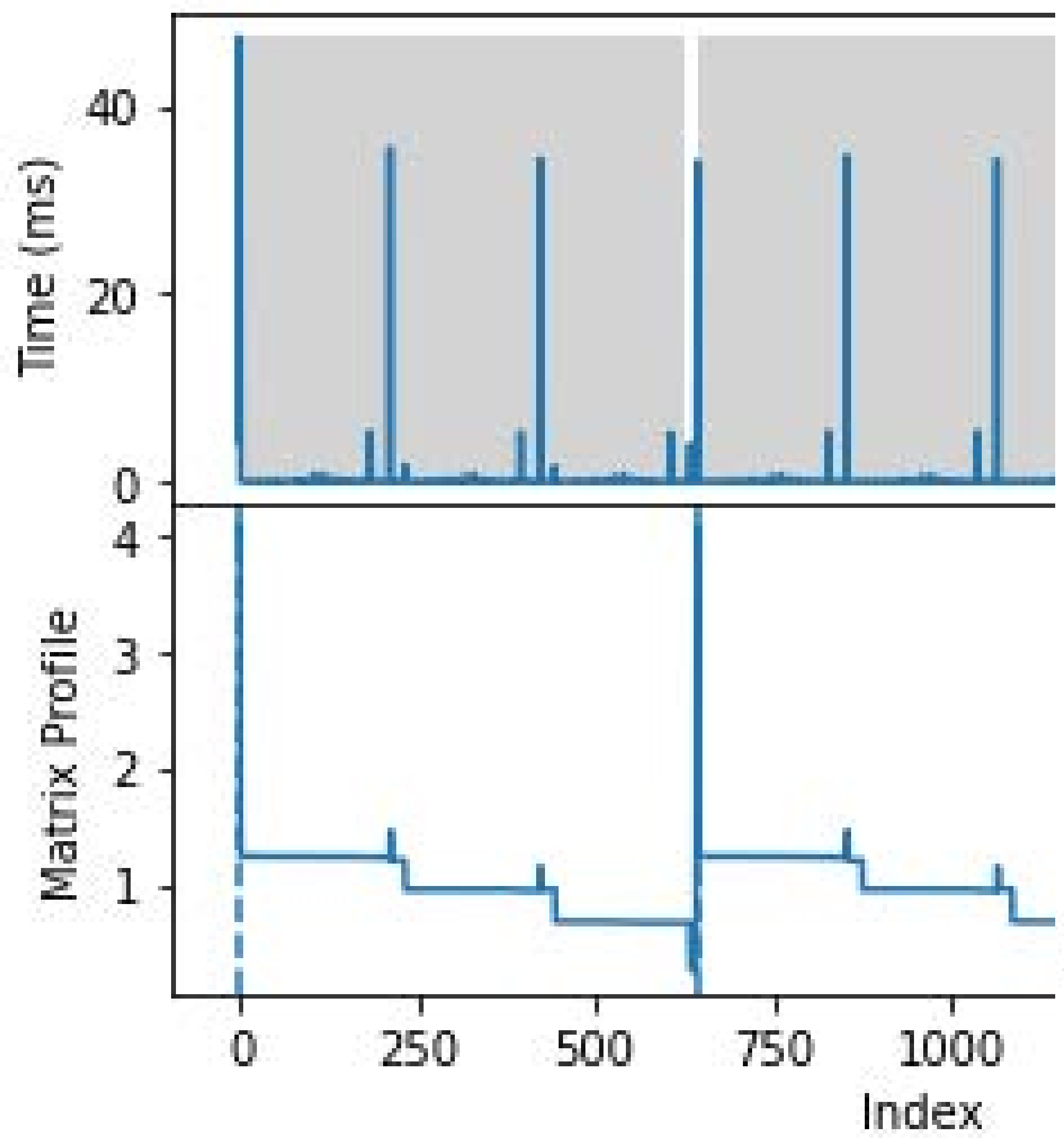
```
tortuga_16 = pipit.Trace.from_otf2('./tortuga_16')

matches = tortuga_16.detect_pattern(window_size, iterations, metric='time.exc')
tortuga_16.plot_timeline()
```

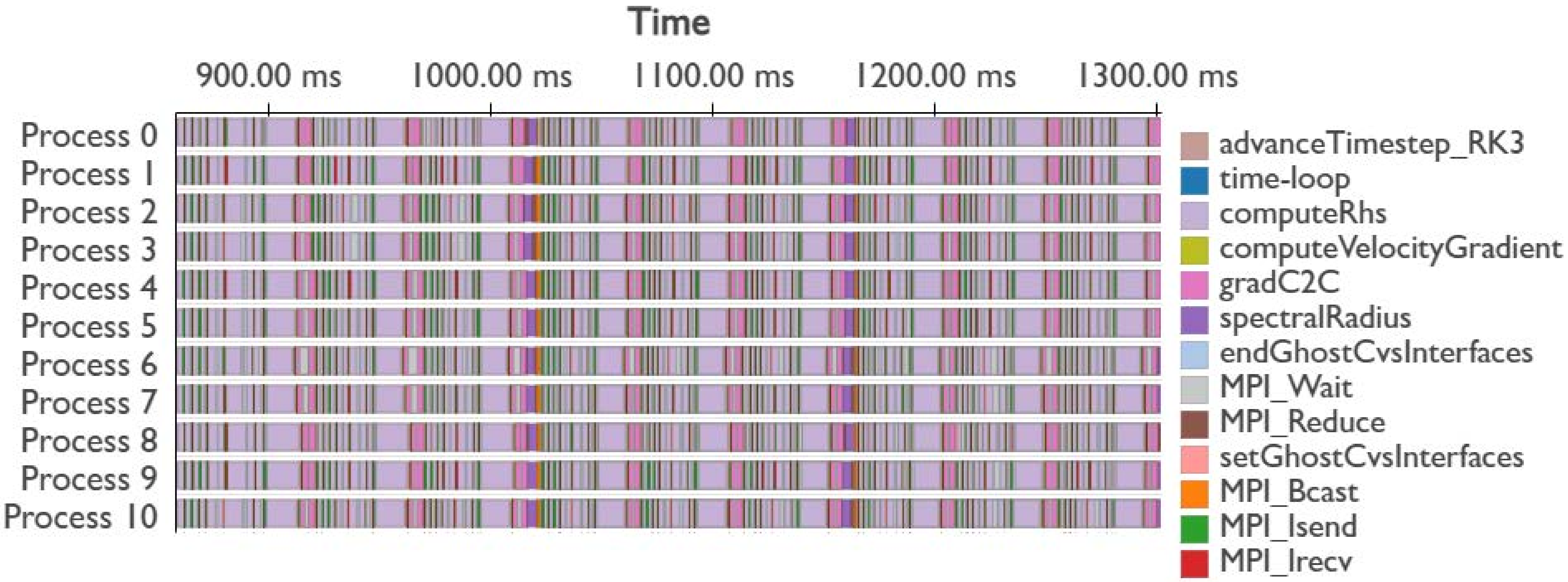


Pattern detection

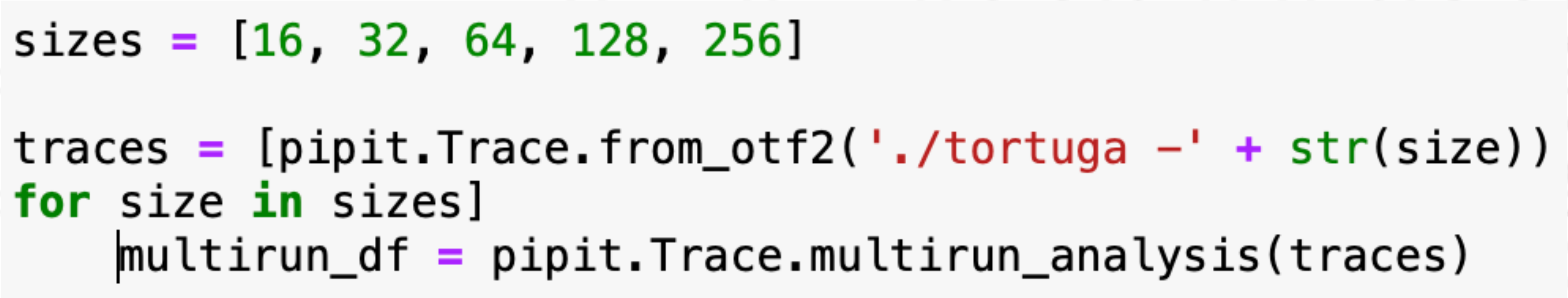


```
tortuga_16 = pipit.Trace.from_otf2('./tortuga_16')

matches = tortuga_16.detect_pattern(window_size, iterations, metric='time.exc')
tortuga_16.plot_timeline()
```



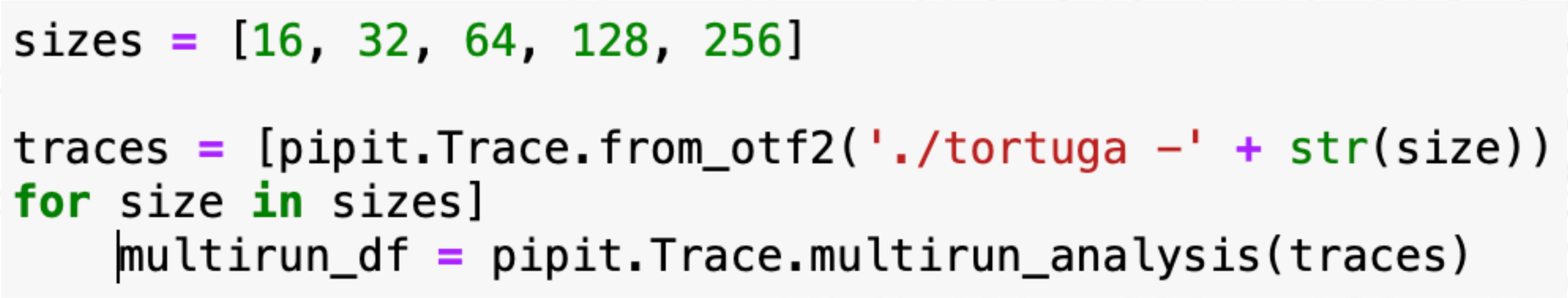


Multi-run analysis

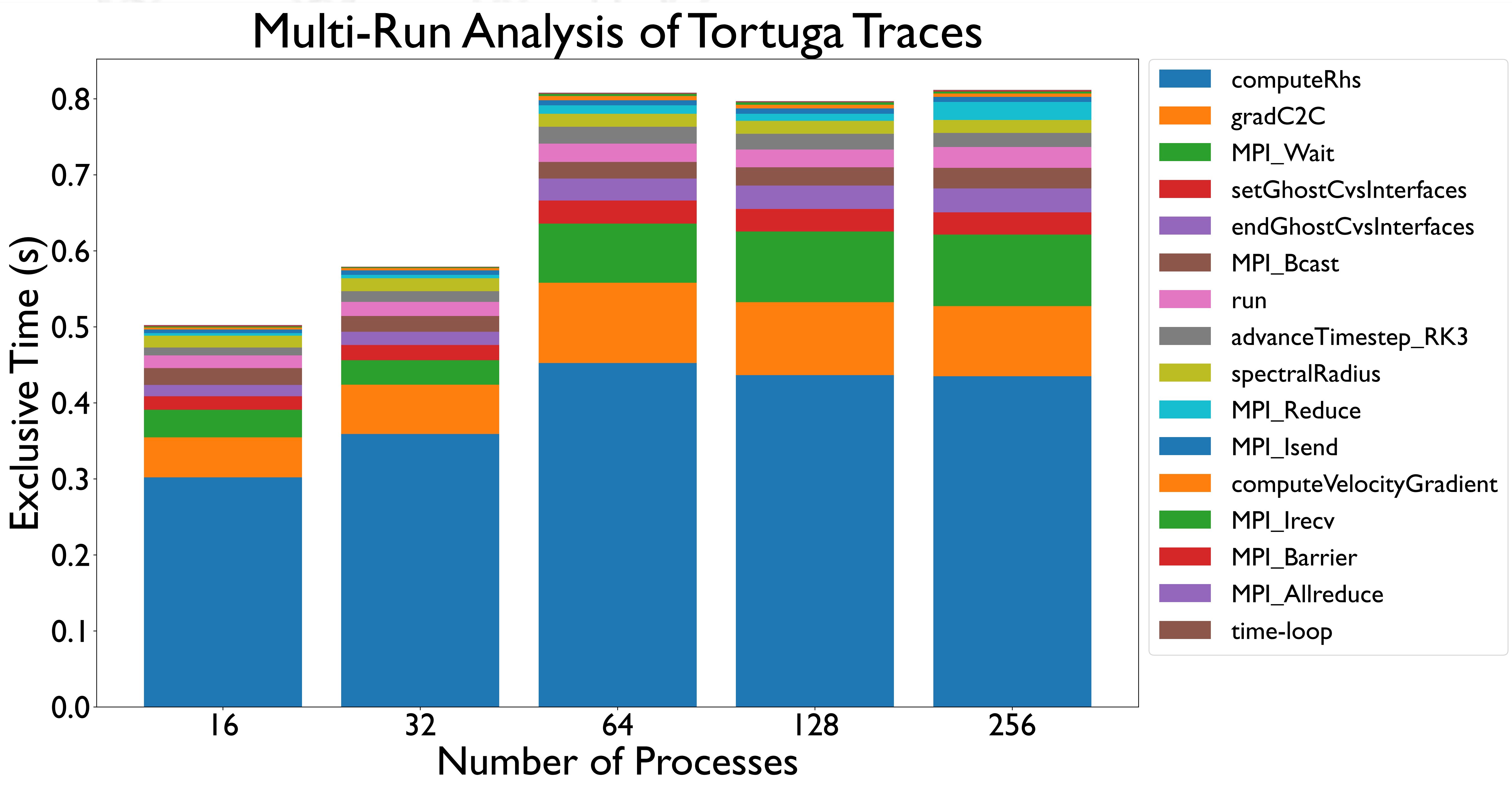


```
sizes = [16, 32, 64, 128, 256]

traces = [pipit.Trace.from_otf2('./tortuga -' + str(size))
for size in sizes]
    multirun_df = pipit.Trace.multirun_analysis(traces)
```

Multi-run analysis

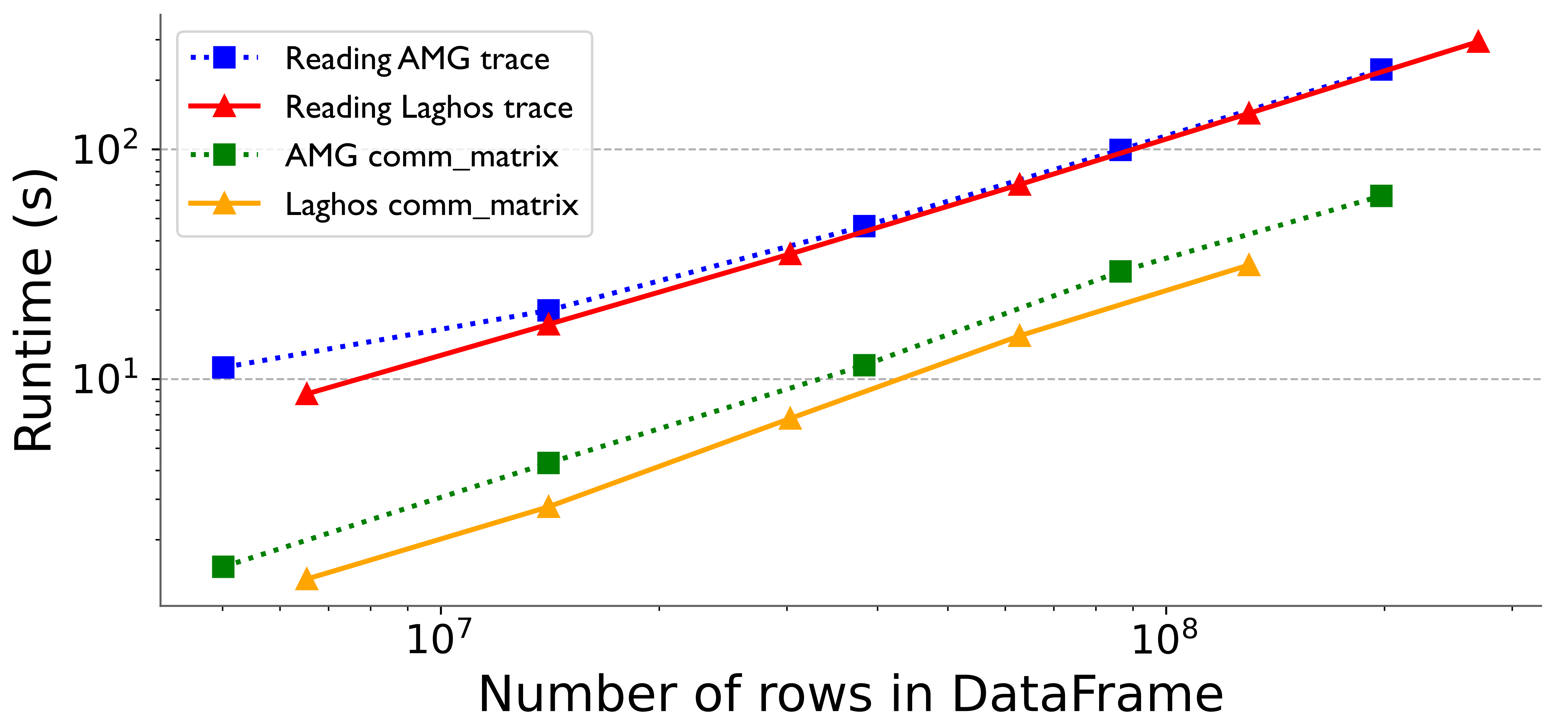


```
sizes = [16, 32, 64, 128, 256]

traces = [pipit.Trace.from_otf2('./tortuga -' + str(size))
for size in sizes]
    multirun_df = pipit.Trace.multirun_analysis(traces)
```



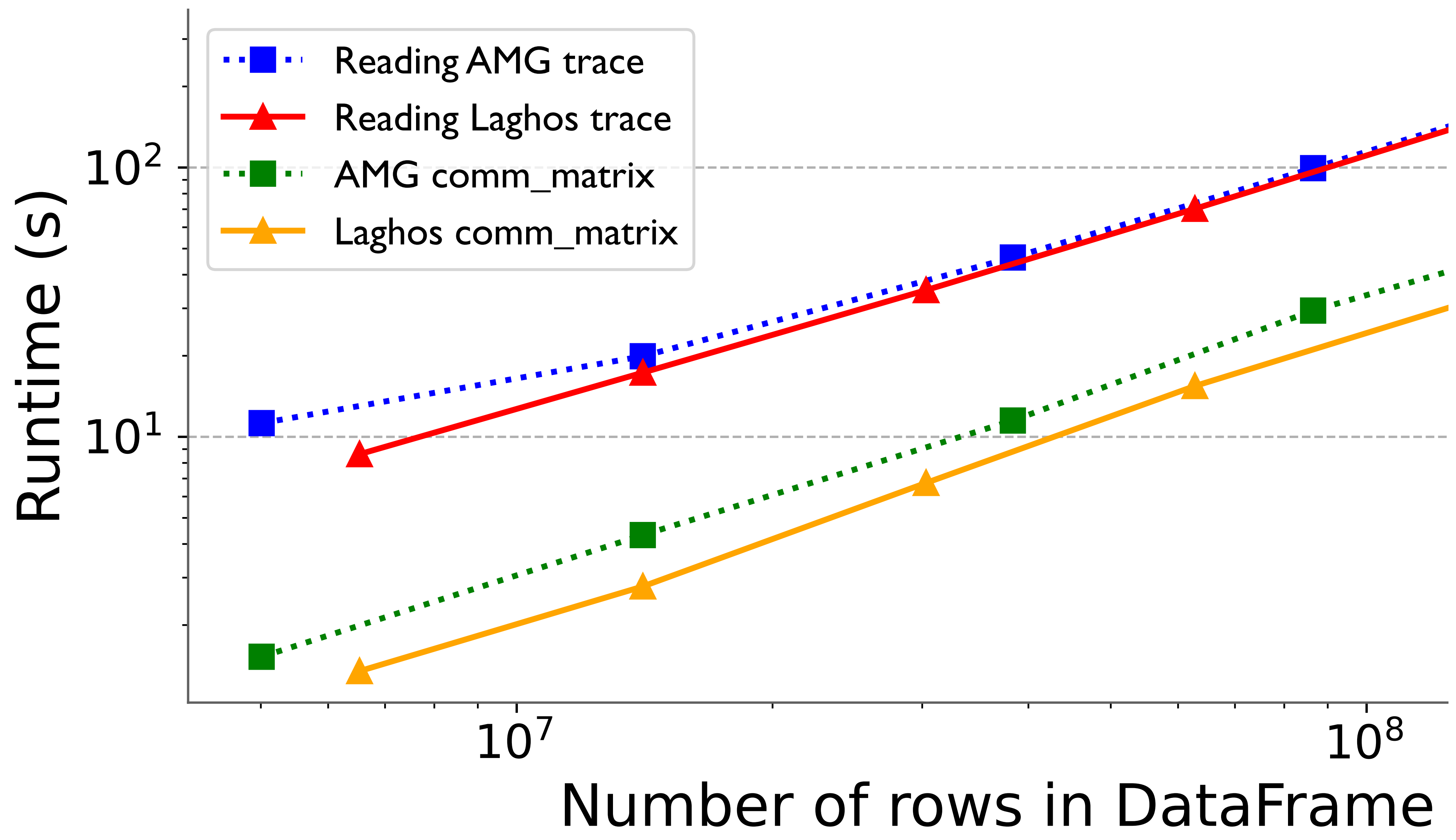
Scalability

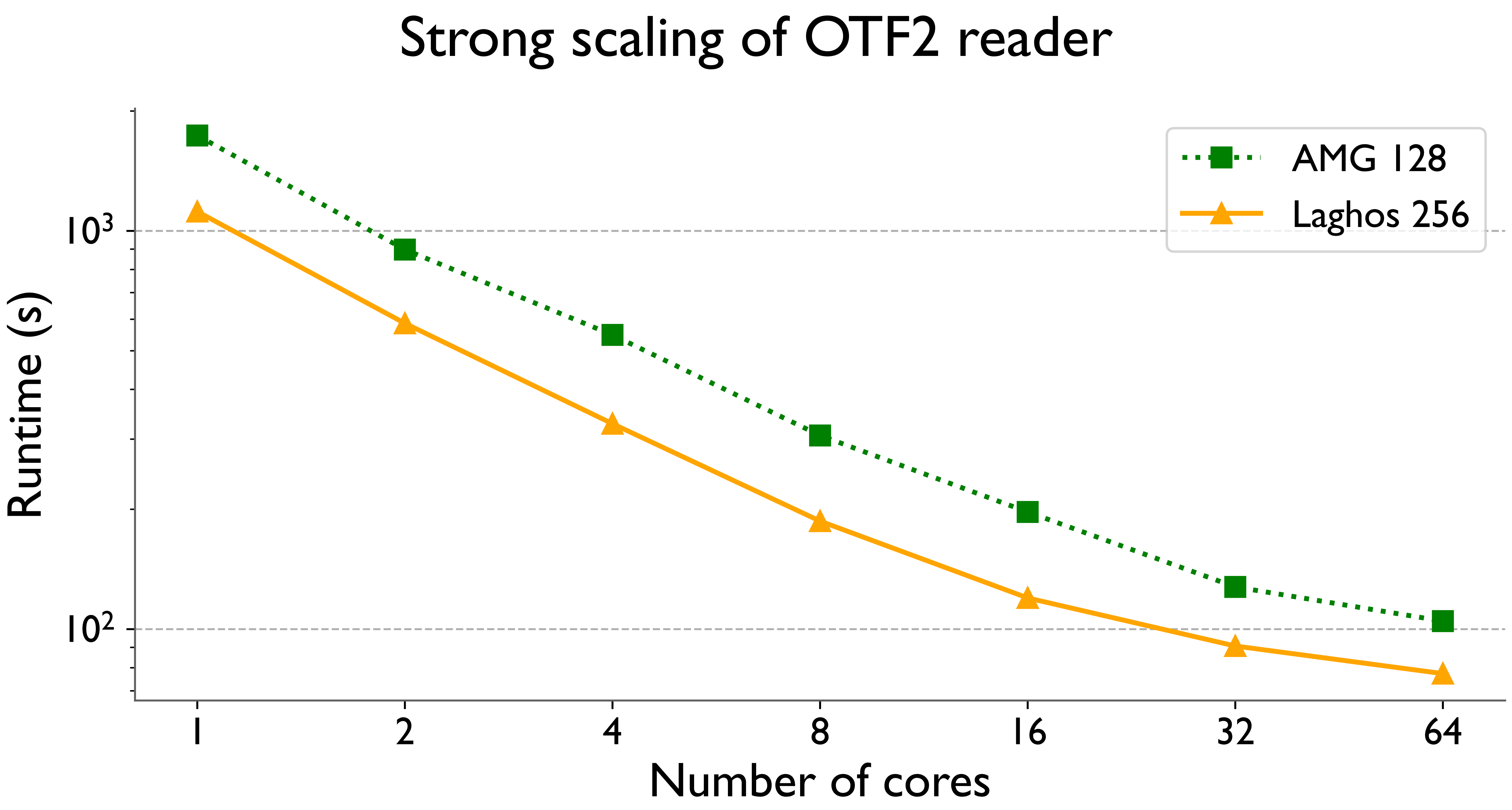
- Very much a work in progress



Scalability

- Very much a work in progress





Summary

- Pipit provides an API for programmatic analysis of parallel traces
- Scripting + visualization can simplify performance analysis and save effort, time, make it more powerful …
- Future work:
  - Scalability of the tool: parallel reading, parallel operations
  - Scalability of the visualization

Code: http://github.com/hpcgroup/pipit

Paper preprint: https://arxiv.org/abs/2306.11177

Abhinav Bhatele

Parallel Software and Systems Group

5218 Brendan Iribe Center (IRB) / College Park, MD 20742

phone: 301.405.4507 / e-mail: bhatele@cs.umd.edu