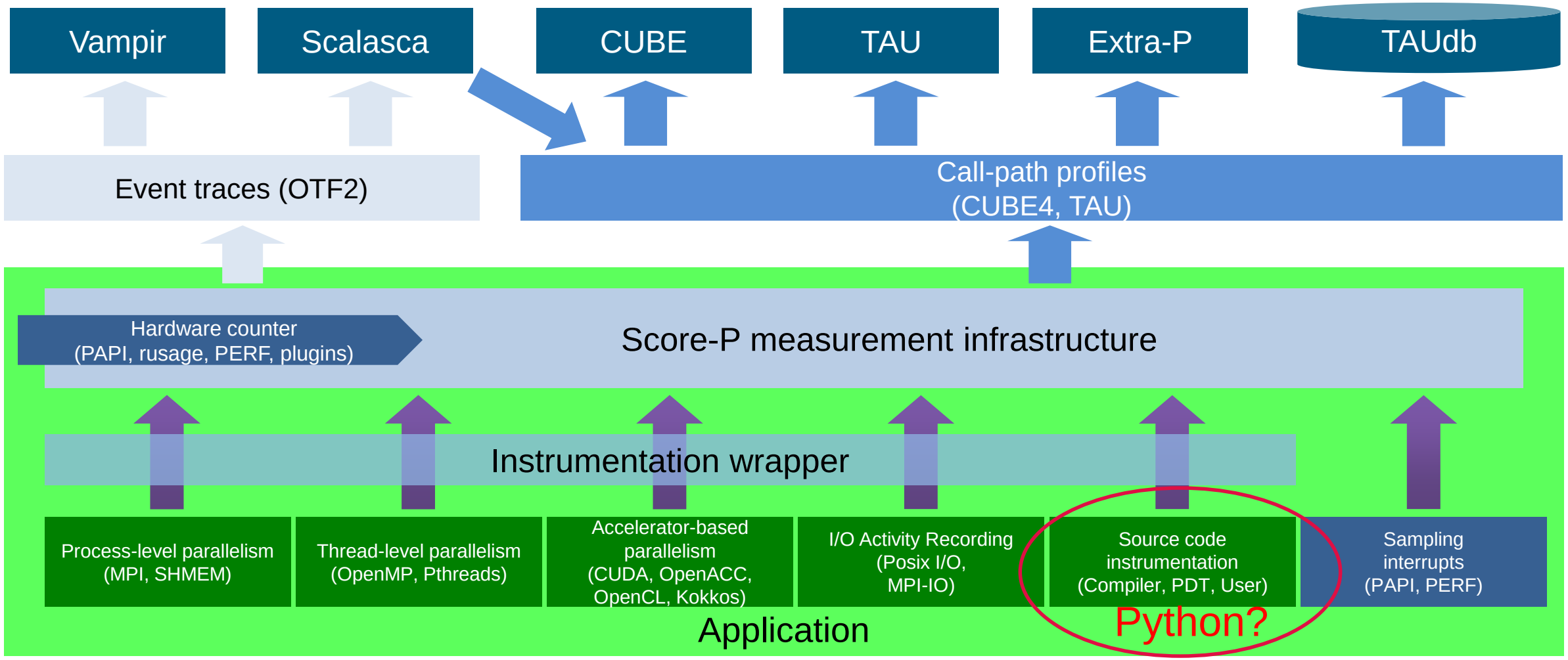
# PERFORMANCE ANALYSIS OF PYTHON CODES USING THE SCORE-P ECOSYSTEM

APRIL 8, 2022 I MICHAEL KNOBLOCH

# RECAP: Score-P ARCHITECTURE

JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE



# PYTHON SUPPORT IN SCORE-P

- Follow the Python approach
  - No Score-P adapter for Python
  - Provide an external Python module for Score-P bindings

- Available at GitHub https://github.com/score-p/scorep_binding_python
- Requires Score-P build with `--enable-shared`
- Simple installation via pip (in virtual environment)

```
git clone https://github.com/score-p/scorep_binding_python
cd scorep_python_bindings/
pip3 install .
```

# USAGE OF THE SCOREP MODULE

- Measure a Python script:

```
python -m scorep <script.py>
```

- Measure a Python script using MPI:

```
python -m scorep -mpp=mpi <script.py>
```

- Also works for multi-threaded code:

```
python -m scorep --mpp=mpi --thread=pthread <script.py>
```

- Measurement control via Score-P environment variables

```
SCOREP_ENABLE_TRACING=true
SCOREP_TOTAL_MEMORY=...
...
```

# MEASUREMENT CONTROL

- By default, all Python functions are instrumented → huge overhead
- Can be disabled using the `--noinstrumenter` flag

- It is possible to enable/disable the instrumenter durin program runtime

```
with scorep.instrumenter.disable():
    do_something()

with scorep.instrumenter.enable():
    do_something()
```

# INSTRUMERNTER EXAMPLES (1)

- Dot product:

```
import numpy as np

[...]
c = np.dot(a,b)
[...]
```

  - Measures dot product and everything else

- Using intrumenter control (and run with `--noinstrumenter`):

```
import numpy as np
import scorep

[...]
with scorep.instrumenter.enable():
    c = np.dot(a,b)
[...]
```

  - Only dot product (and everything below) is measured

# INSTRUMENTER EXAMPLES (2)

- Score-P records every change of state of the instrumenter
  - Disabled regions can be named, that will create an event

```
[...]
def fun_calls(n):
    if (n>0):
        fun_calls(n-1)

with scorep.instrumenter.disable("my_fun_calls"):
    fun_calls(1000000)
[...]
```

    - `my_fun_calls` will appear in measurement, `fun_calls` will not

```
[...]
with scorep.instrumenter.disable():
    with scorep.instrumenter.disable("my_fun_calls"):
        fun_calls(1000000)
[...]
```

    - Neither call will appear in measurement as no state change happend

# USER REGIONS

- Since v2.0, the Python bindings support context managers for user regions:

```
with scorep.user.region("region_name"):
  do_something()
```

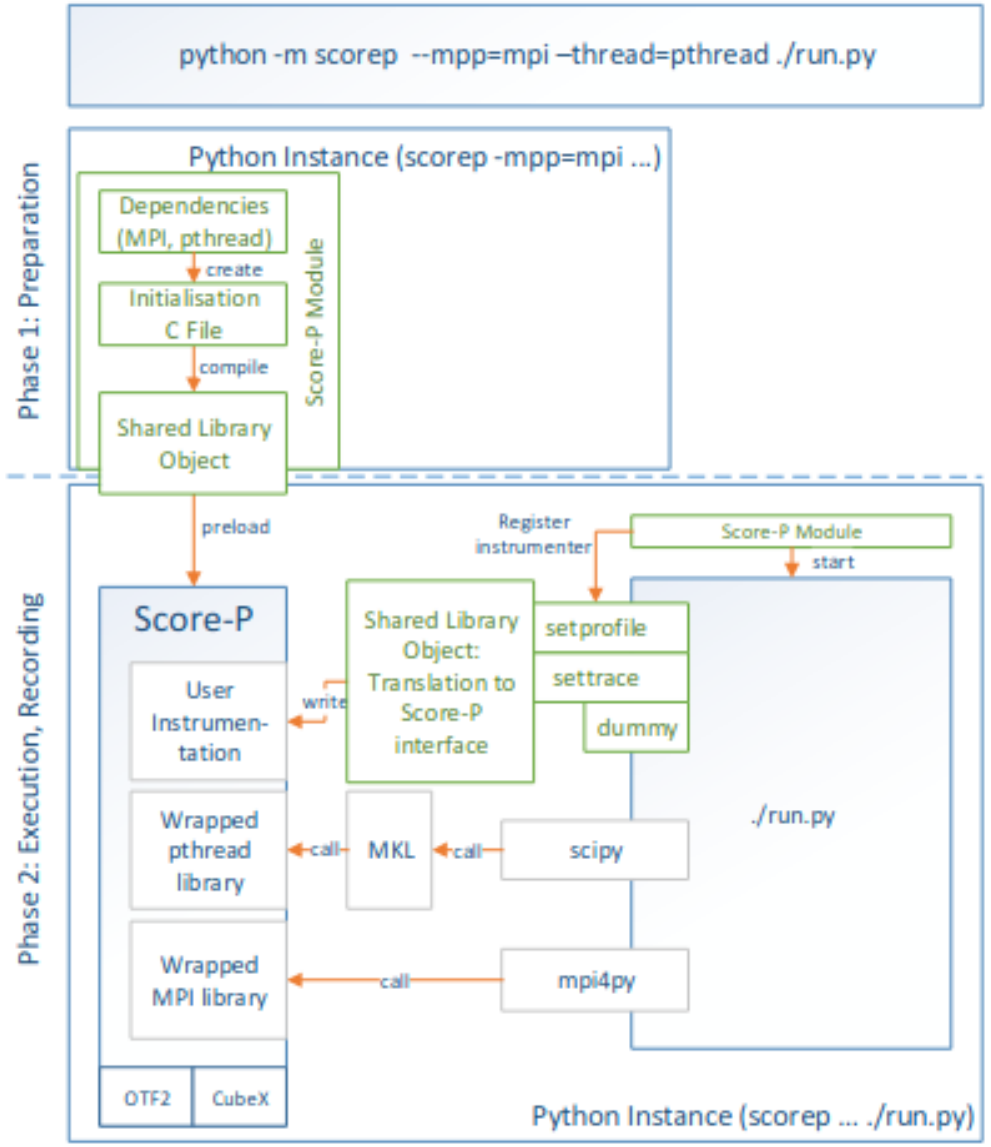
- Since v2.1, the Python bindings support also decorators for functions:

```
@scorep.user.region("region_name")
def do_something():
  #do some things
```

  - If no region name is given, the function name will be used
- Traditional calls to define a region are also supported

```
scorep.user.region_begin("region_name")
scorep.user.region_end("region_name")
```

# SUMMARY - HOW DOES IT WORK?

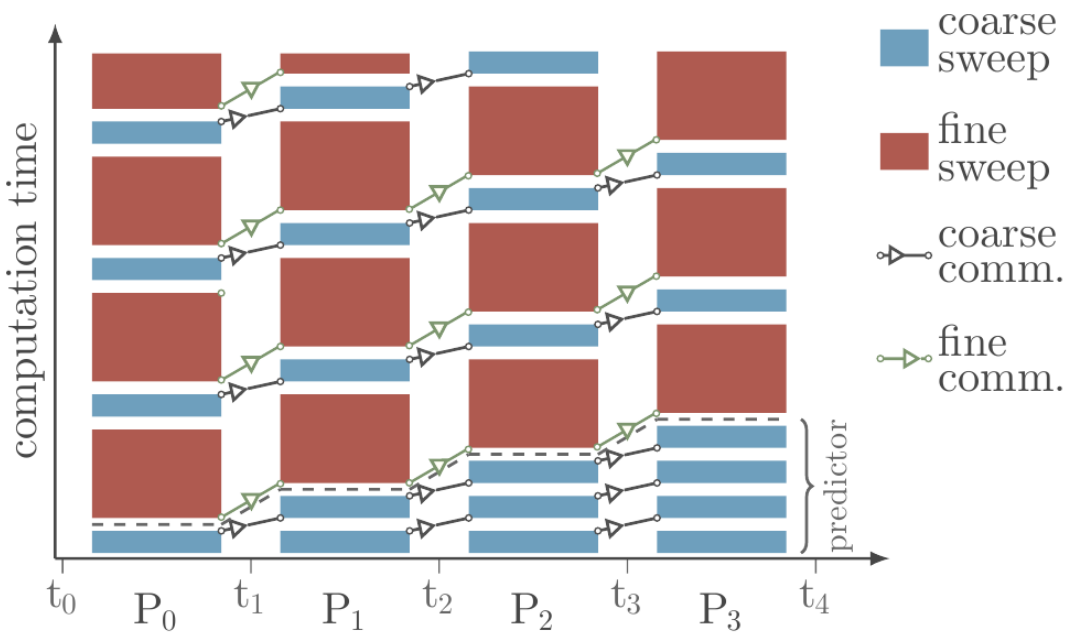


Case Study

# PYSDC

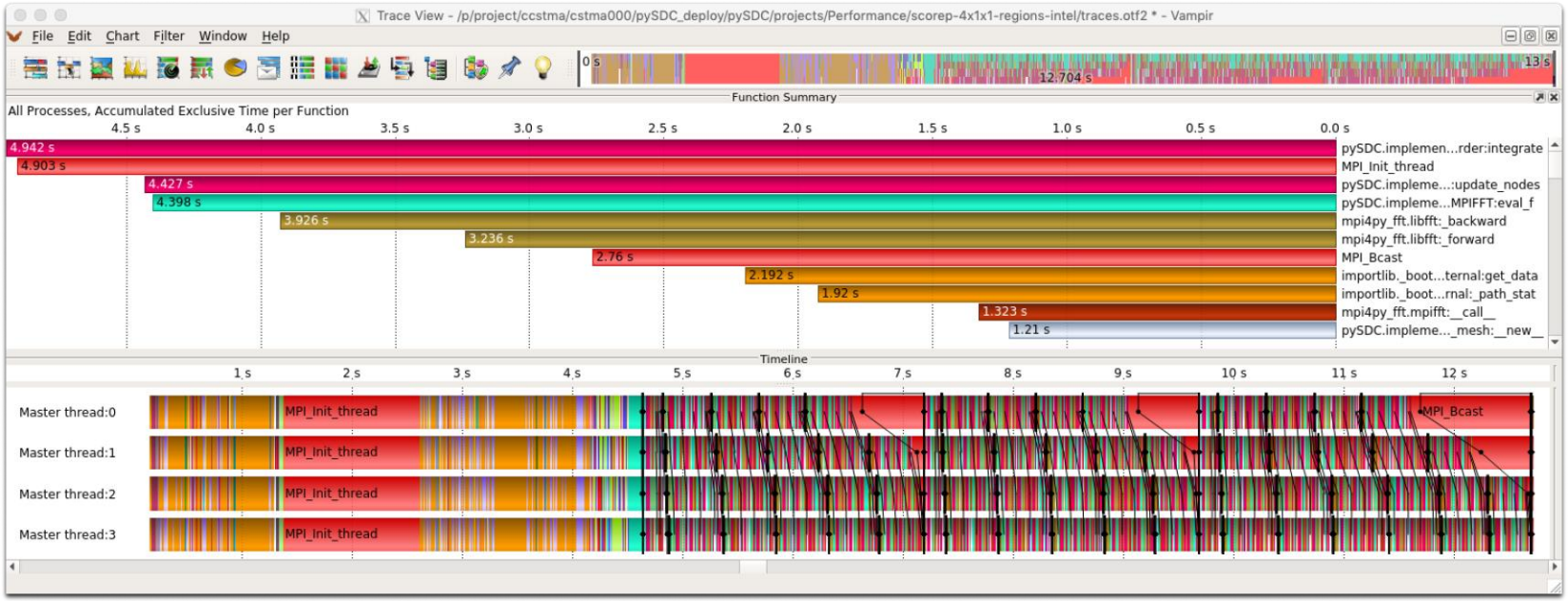
# PYSDC

- Python implementation of the spectral deferred correction (SDC) approach
- Intended for rapid prototyping and educational purposes
- Framework for testing, evaluating and applying different variants of SDC and PFASST:
  - available implementations of many variants of SDC, MLSDC and PFASST
  - many ordinary and partial differential equations already pre-implemented
  - coupling to FEniCS and PETSc, including spatial parallelism for the latter
  - tutorials to lower the bar for new users and developers
- MPI parallelization in time using **mpi4py**



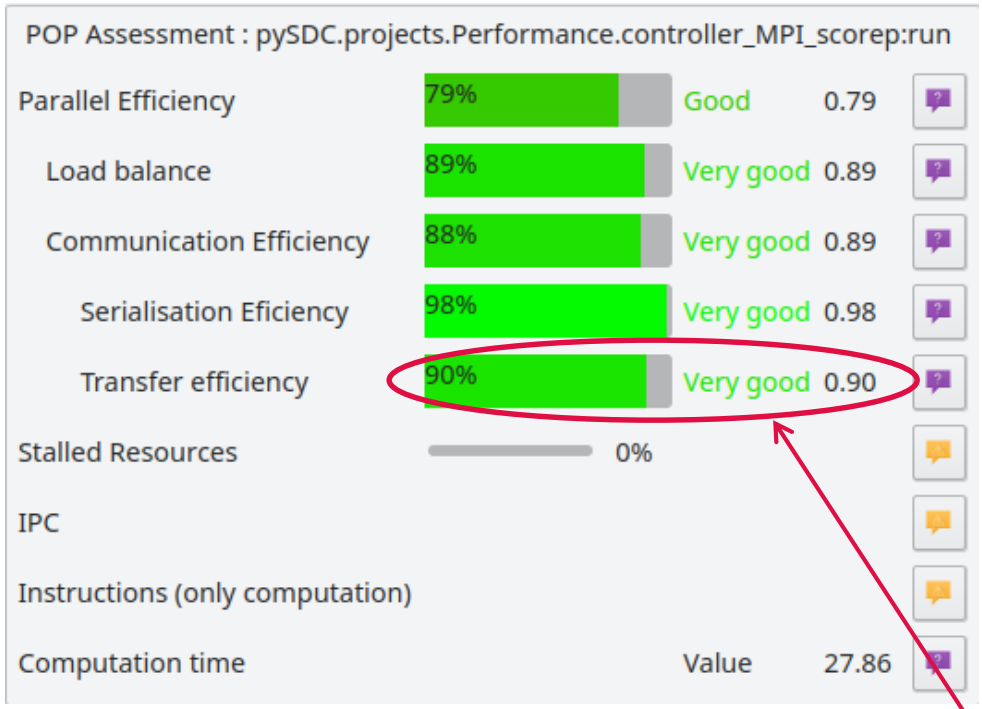
# PYSDC FIRST MEASUREMENTS

- Plan: Use Intel compilers and Intel MPI
  - Failed due to an issue with the Intel compiler and the Score-P python bindings
  - Switched to GNU compilers and ParaStationMPI
- First trace was confusing – overwhelming with information
  - Switched to manual instrumentation

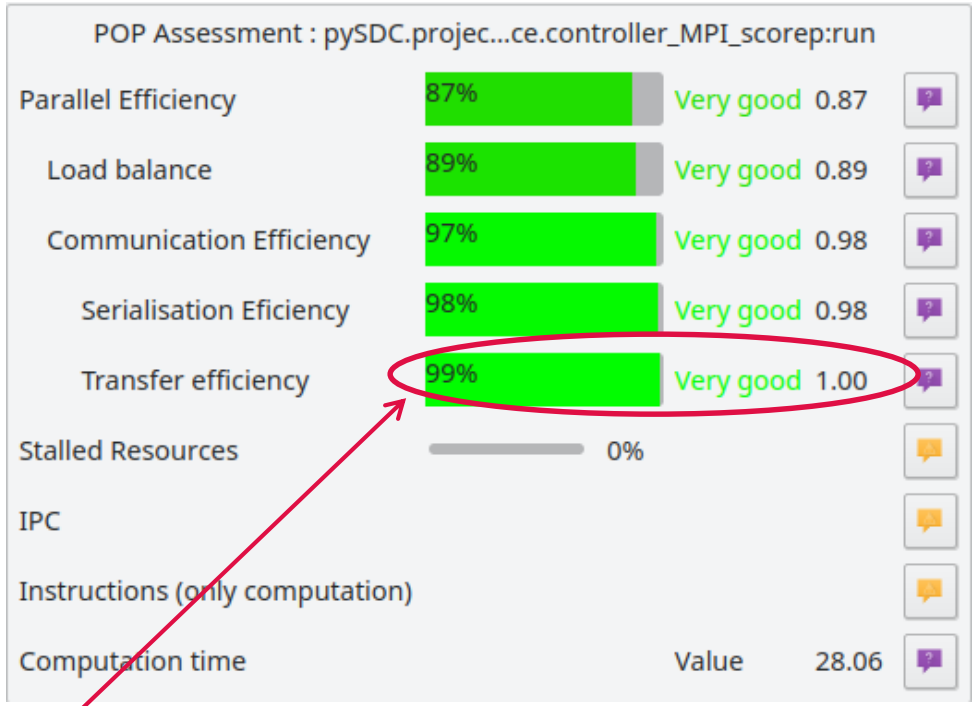



```python
from mpi4py import MPI
from pySDC.core.Controller import controller

import scorep.user as spu

...

def run_pfasst(*args, **kwargs):
    ...

    while not done:
        ...
        name = f'REGION -- IT_FINE -- {my_rank}'
        spu.region_begin(name)
        controller.do_fine_sweep()
        spu.region_end(name)
        ...
        name = f'REGION -- IT_DOWN -- {my_rank}'
        spu.region_begin(name)
        controller.transfer_down()
        spu.region_end(name)
        ...
        name = f'REGION -- IT_COARSE -- {my_rank}'
        spu.region_begin(name)
        controller.do_coarse_sweep()
        spu.region_end(name)
        ...
        name = f'REGION -- IT_UP -- {my_rank}'
        spu.region_begin(name)
        controller.transfer_up()
        spu.region_end(name)
        ...
        name = f'REGION -- IT_CHECK -- {my_rank}'
        spu.region_begin(name)
        controller.check_convergence()
        spu.region_end(name)
        ...

    ...

...
```

# ADVISOR COMPARISON – PS MPI VS. INTEL MPI



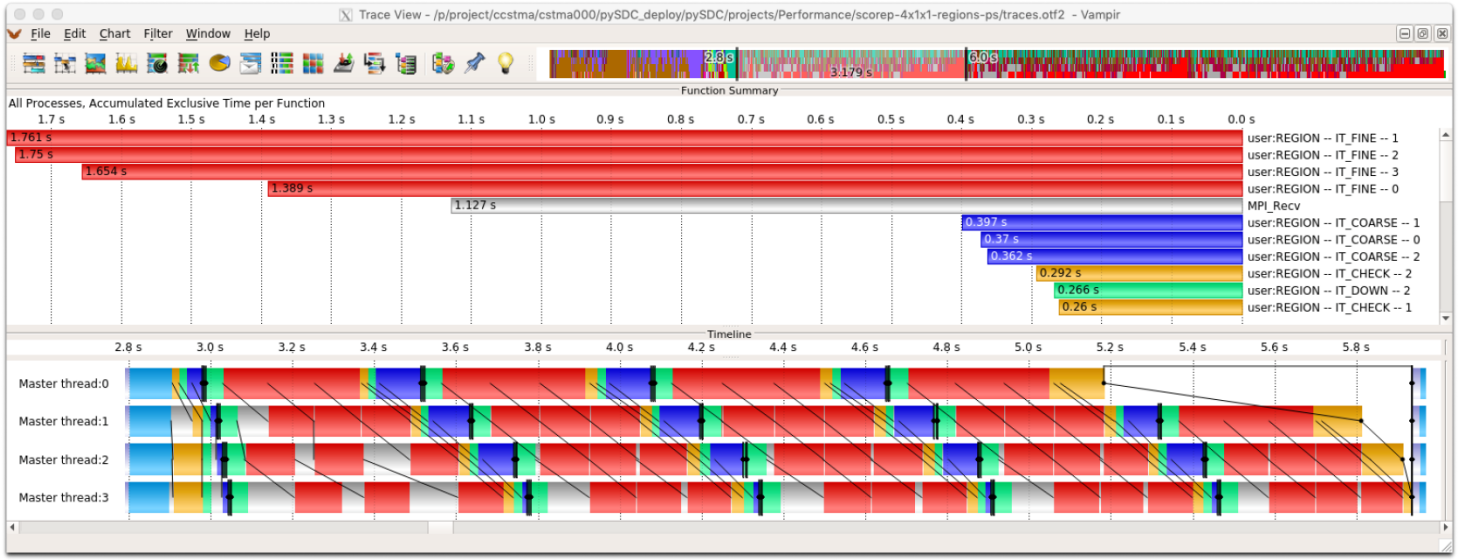

ParaStationMPI

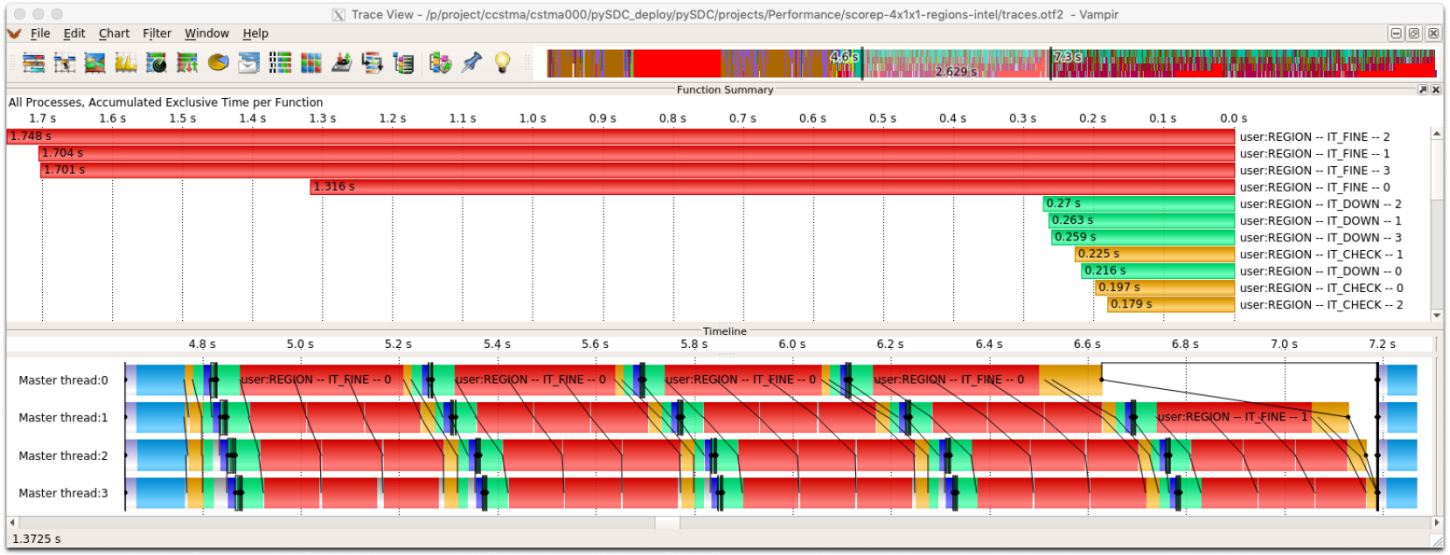



Intel MPI

Why?

# TO OVERLAP OR NOT TO OVERLAP?



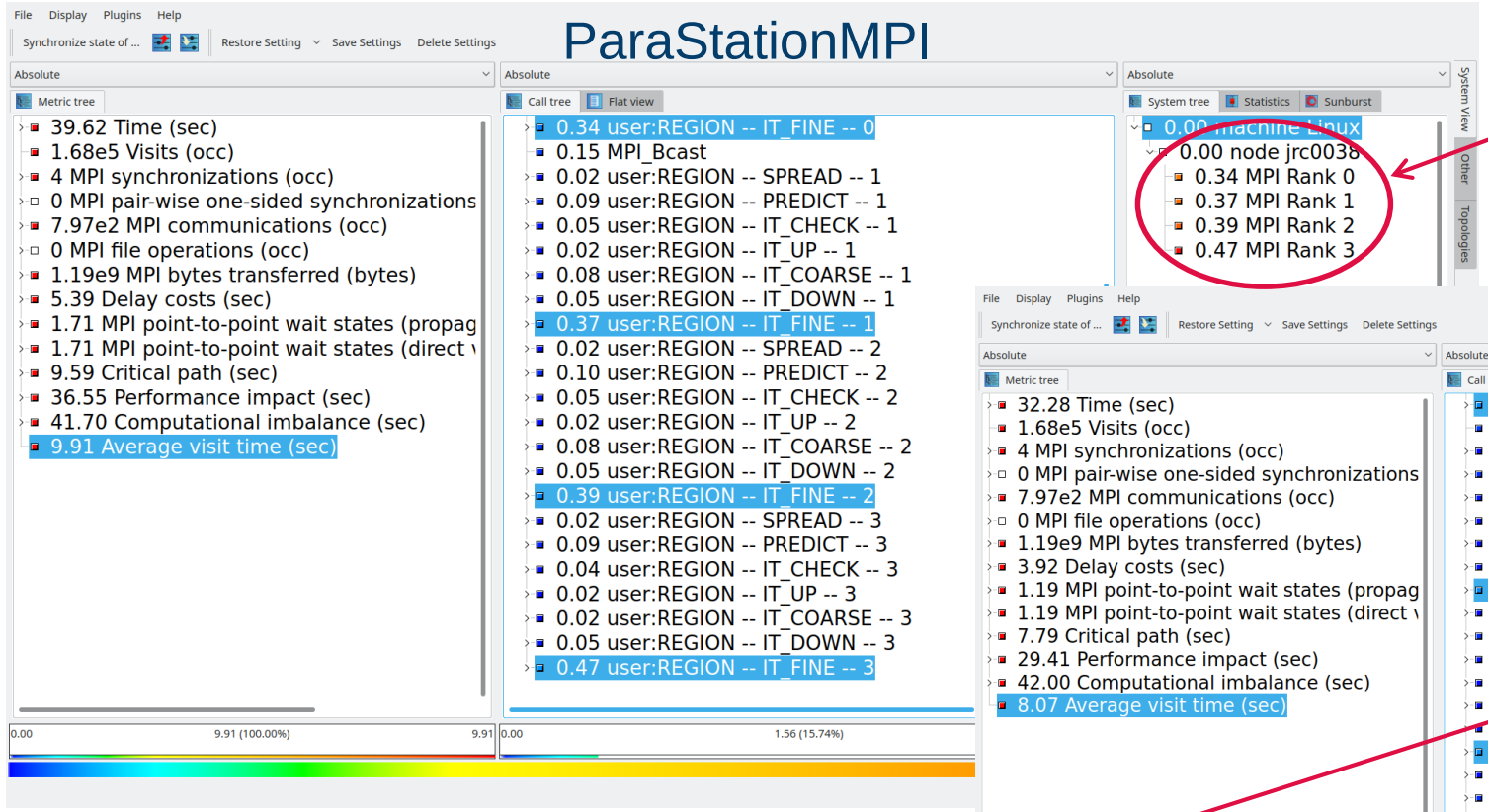




- Parastation MPI:
  - Significant time spend in `MPI_Recv`
  - No overlap of computation and communication

- Intel MPI:
  - Hardly any time spend in `MPI_Recv`
  - Good overlap of computation and communication

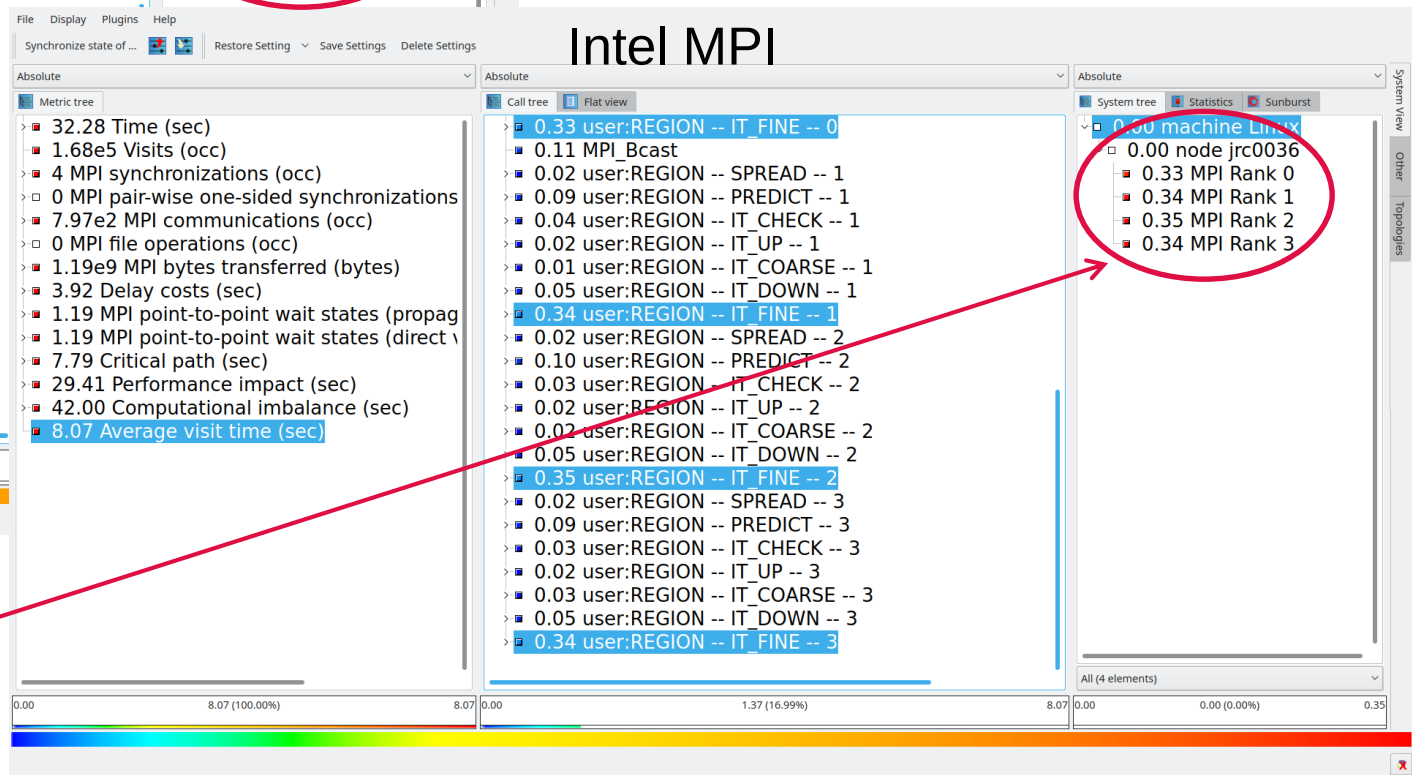
# IS IT REALLY A PROBLEM?

ParaStationMPI




- Runtime increases with MPI rank
- Likely a problem at scale

Intel MPI




- Runtime well balanced
- Probably not a problem

# QUESTIONS?

JÜLICH Forschungszentrum | JÜLICH SUPERCOMPUTING CENTRE


- http://www.scalasca.org
- scalasca@fz-juelich.de


- http://www.score-p.org
- support@score-p.org