Enabling ROS 2 Benchmarks: A Medical Robotics Perspective

Marco Pedretti
Zoltan Resi
Shashank Sharma

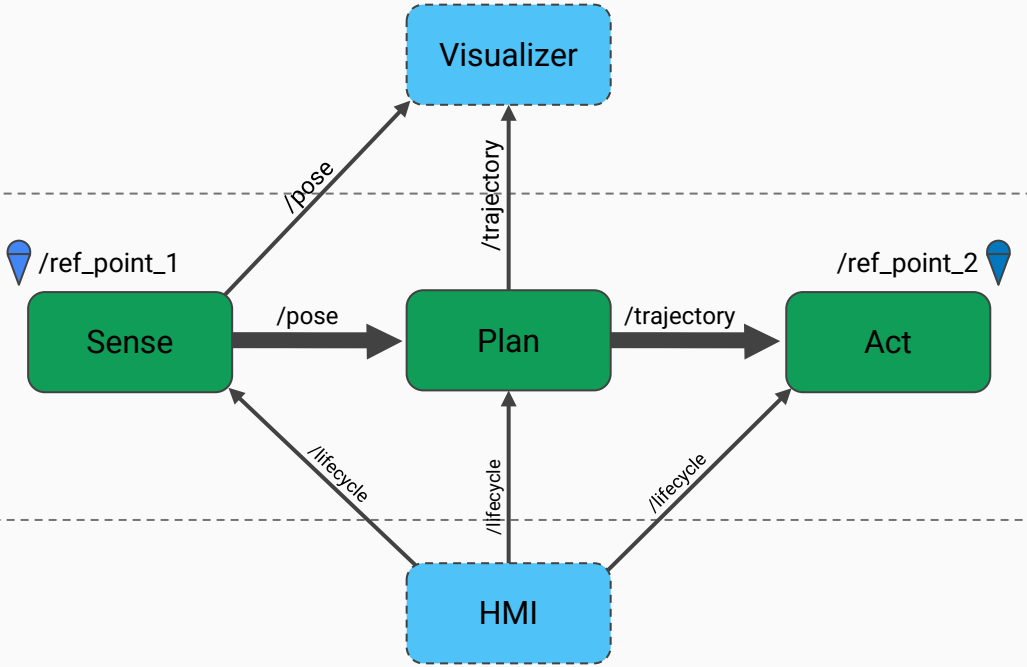
Use Case

System

Example ROS 2 application defined by the launch configuration

Subsystem

Nodes of the system that have to be tested for real-time requirements, defined by the profiling configuration



Existing Tools

Disclaimer: it is not our intention to provide a verdict on which tool is the best, the user should pick the most suitable tool for their use case.

ApexAI/performance_test

- × Focuses on benchmarking RMW implementations on a single topic communication

iRobot/ros2-performance

- × Focuses on benchmarking RMW implementations on a configurable synthetic system

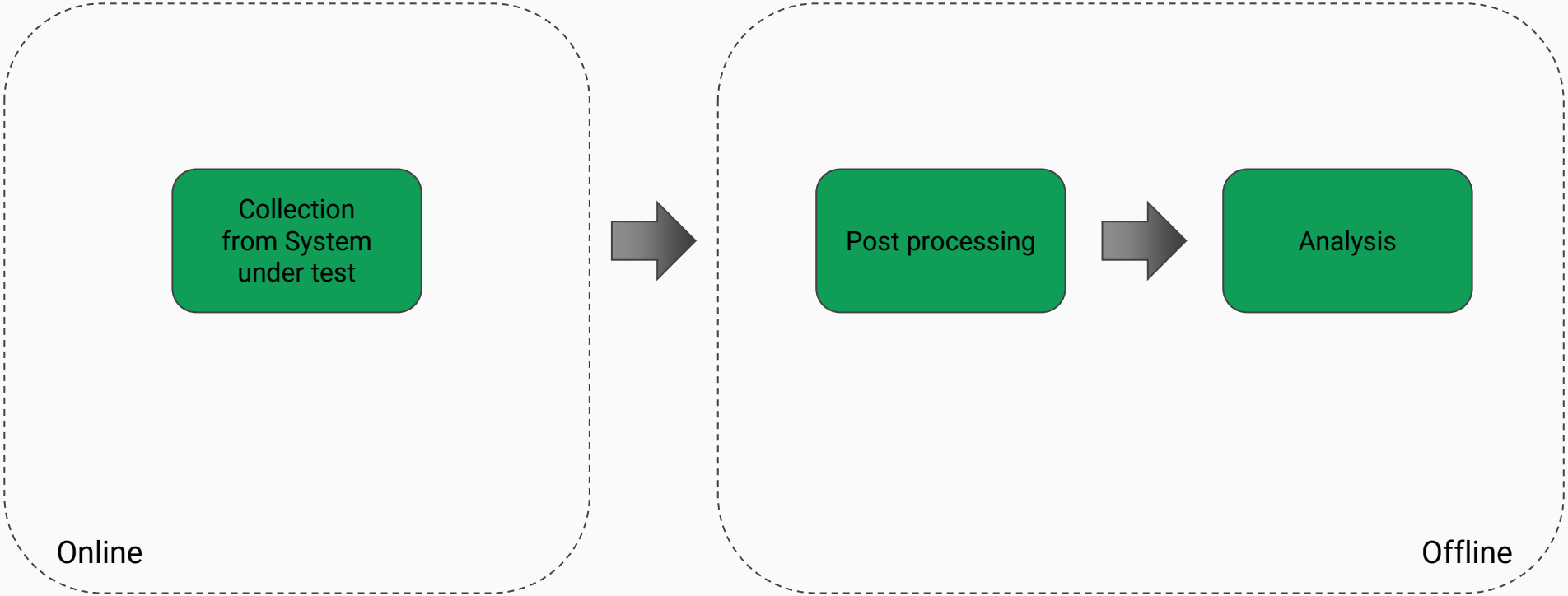
TierIV/caret

- ✓ Focuses on application level
- × Requires custom ROS 2 forks

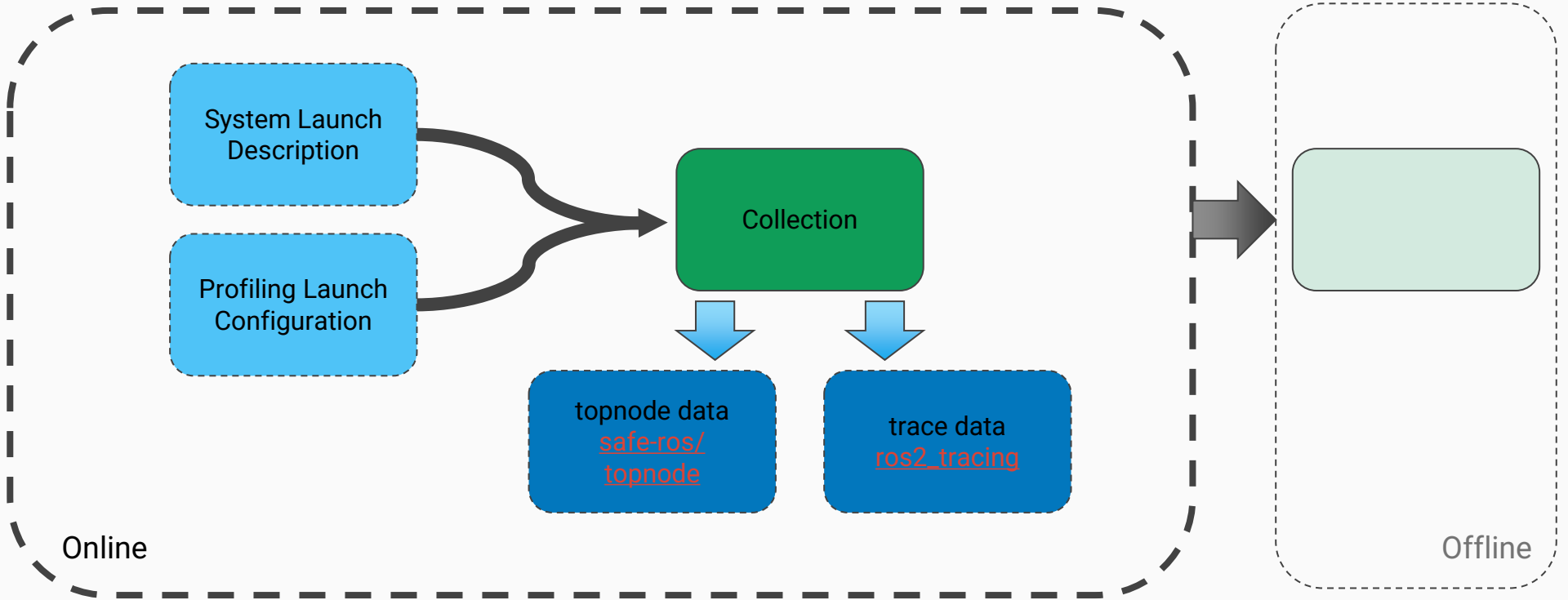
ROS 2 Profiling – Goals

- Ability to instrument arbitrary ROS 2 systems
  - System-wide metrics
  - Fine grained metrics
- Ability to analyze the collected metrics
  - Manual - Jupyter notebook, Python scripts (plots, diagrams, etc…)
  - Automated - Pytest-style assertions
- Minimize the load on the system under test

ROS 2 Profiling – Flowchart



ROS 2 Profiling – Flowchart – Collection



ROS 2 Profiling – Collection – launch

ros2 profile launch:

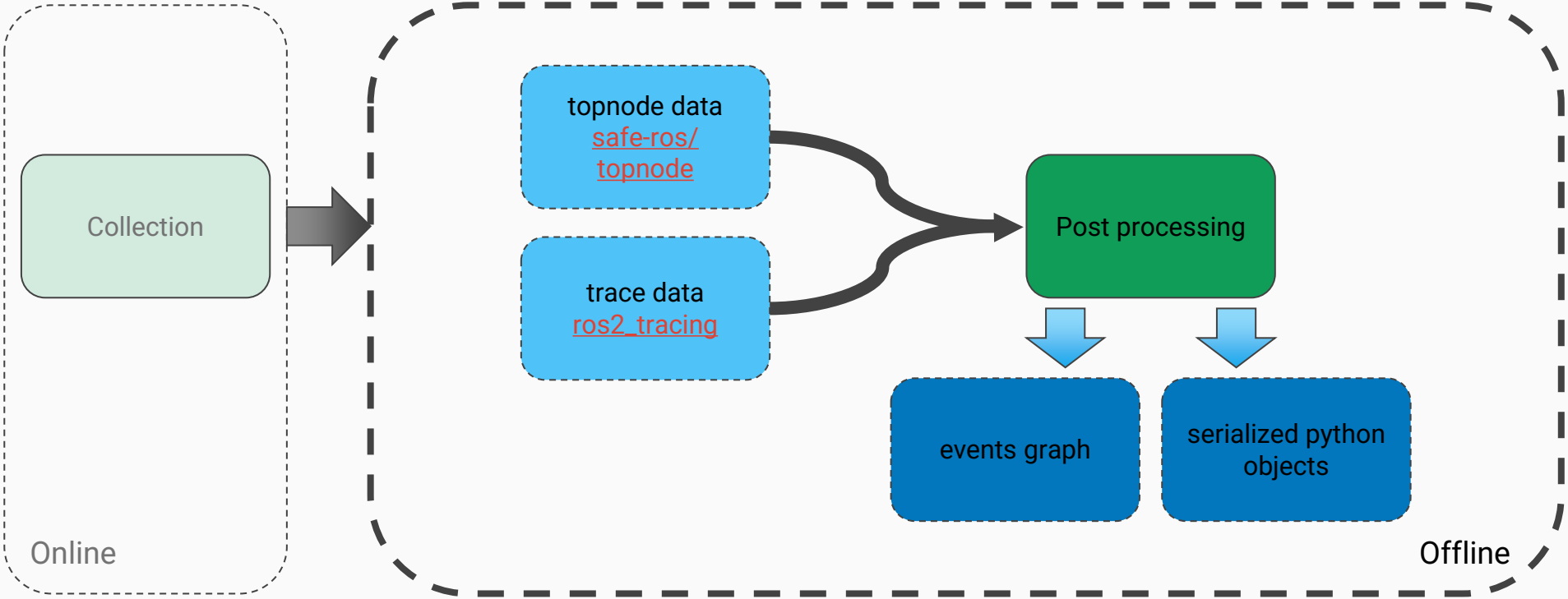
- Launch a ROS 2 application
- Add topnode
- Add Trace action

```
launch_ros.actions.Node(
name=f'{node_name}_monitor',
namespace='',
package='topnode',
executable='resource_monitor',
output='screen',
parameters=[{
    "publish_period_ms": 500,
    "record_cpu_memory_usage": True,
    "record_memory_state": True,
    "record_io_stats": True,
    "record_stat": True,
    "record_file": f'{output_dir}{event.action.node_name}_{pid}.mcap',
    "pid": pid
}])
```

```
ros2 profile launch \
  --launch-file \
  ./src/reference_system/launch/reference_system.launch.py \
  --config-file \
  ./src/ros2_profiling/ros2_profiling_demo/config/reference_system.yaml
```

```
launch_description.add_action(action=Trace(
    session_name=session_name,
    base_path=output_dir,
    append_timestamp=append_timestamp,
    events_kernel=events_kernel,
    events_ust=events_ust,
    context_fields=context_fields,
    subbuffer_size_ust=subbuffer_size_ust,
    subbuffer_size_kernel=subbuffer_size_kernel
))
```

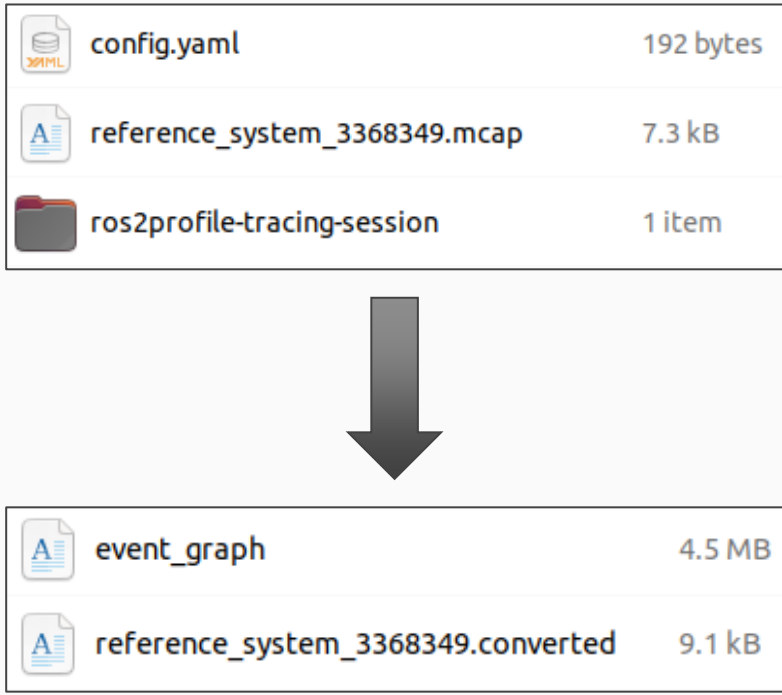
ROS 2 Profiling – Flowchart – Post-Processing



ROS 2 Profiling – Post-Processing

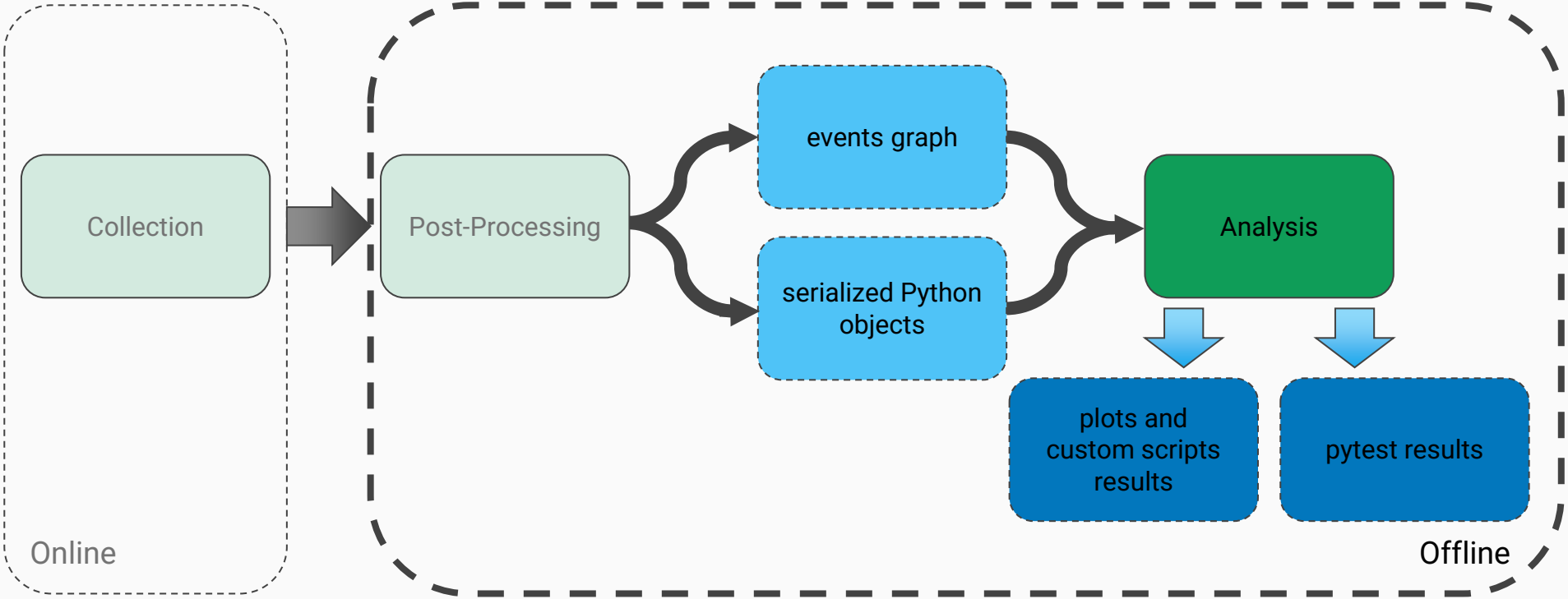
ros2 profile process:

- Convert mcap data (from topnode)
- Create events graph (from trace points)
  - context, nodes, pubs, subs, timers
  - events association
    - pubs and subs on same topic
    - pubs from timer cb
    - pubs from subs cb
    - ros2_profile_extensions




```
ros2 profile process ~/.ros/profile/reference_system-20230117095738/
```

ROS 2 Profiling – Flowchart – Analysis



ROS 2 Profiling – Analysis

jupyter notebook

- more interactive analysis
- testing/assertion development

ros2 profile run_test

- perform testing/assertion

```
ros2 profile run_test \
  ~/.ros/profile/reference_system-20230117104350/ \
  ./src/ros2_profiling/ros2_profiling_demo/test/test_profile.py
```

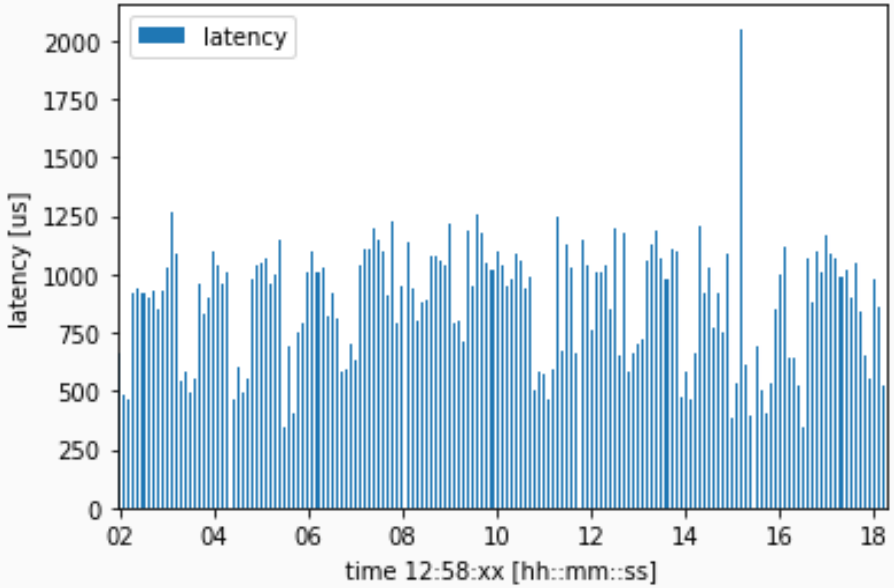
```
%pylab inline

import os

from ros2profile.api.process import load_mcap_data
```

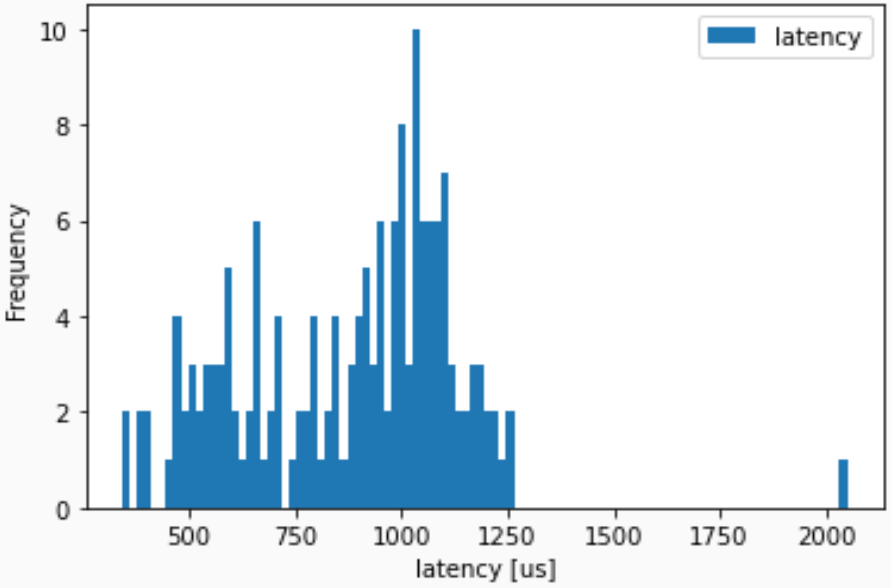
```
PROFILE_RUN = os.path.expanduser('~/.ros/profile/reference_system-20230310113449/')

topnode_data = load_mcap_data(PROFILE_RUN)
```

```
recorded_processes = list(topnode_data.keys())

cpu_per_proc = {}
memory_per_proc = {}

for proc in recorded_processes:
    s = proc.split('_')
    proc_name = '_'.join(s[0:-1])
    proc_pid = int(s[-1])
    print(f'Process: {proc_name} ({proc_pid})')
    recorded_msgs = topnode_data[recorded_processes[0]]
    for msg in recorded_msgs.keys():
        print(f' * {msg} ({len(recorded_msgs[msg])} msgs)')
```

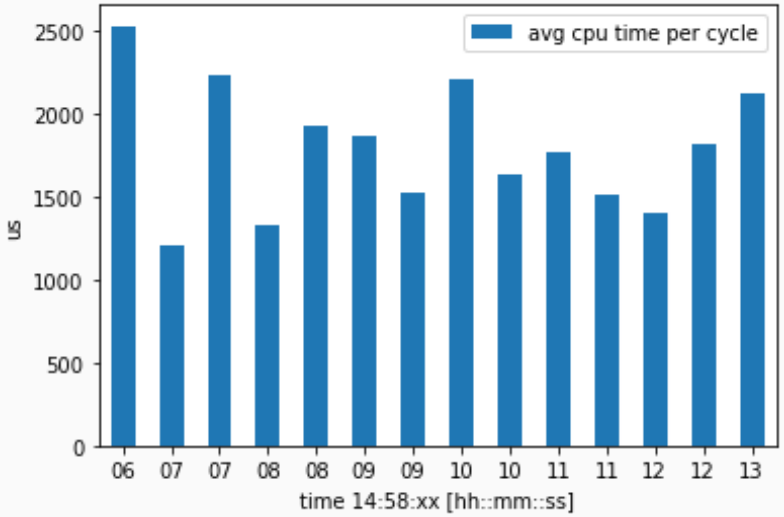
```
Process: reference_system_robot (1333493)
 * ~/cpu_memory_usage (137 msgs)
 * ~/memory_state (137 msgs)
 * ~/io_stats (137 msgs)
 * ~/stat (137 msgs)
Process: reference_system_control (1333495)
 * ~/cpu_memory_usage (137 msgs)
 * ~/memory_state (137 msgs)
 * ~/io_stats (137 msgs)
 * ~/stat (137 msgs)
```

ROS 2 Profiling – Example Output



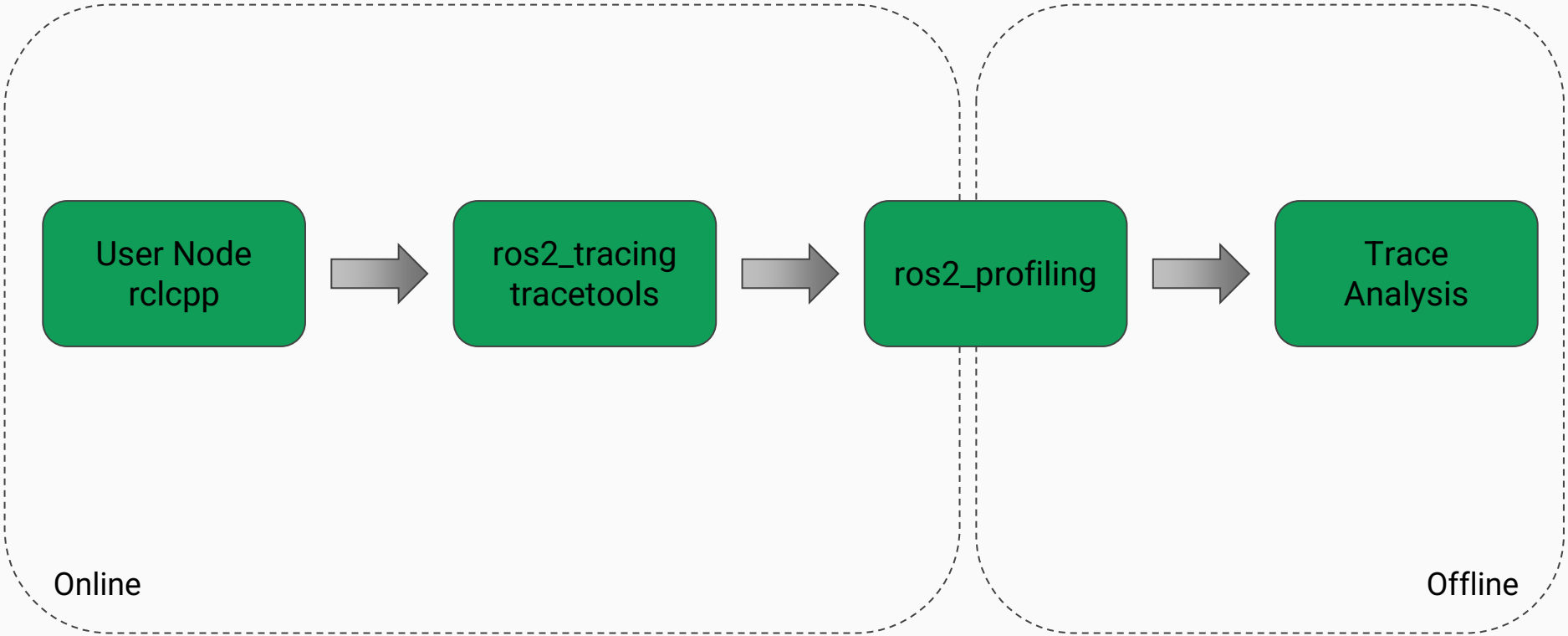


ROS 2 Profiling – Example Output

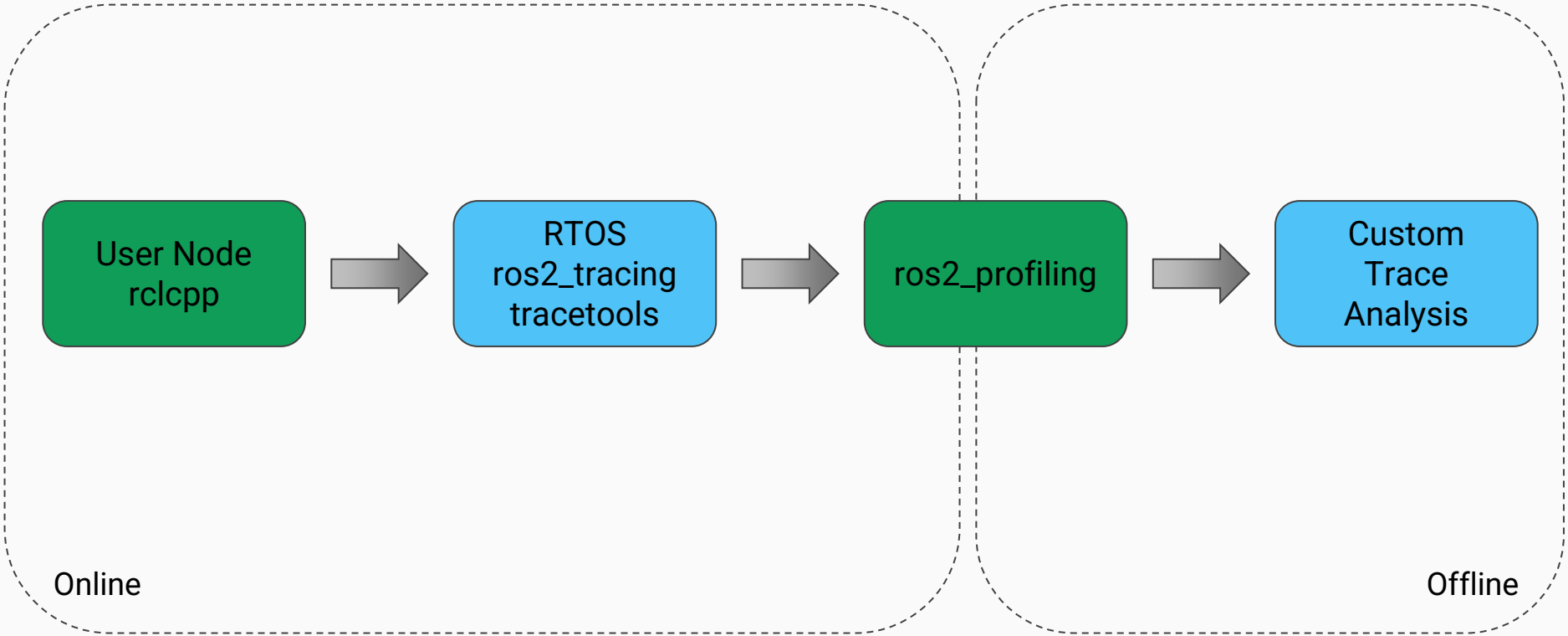



```
mapedretti@mylaptop:~/development/safe_ros$ ros2 profile run_test \
  ~/.ros/profile/reference_system-20240923145801/ \
  ./src/ros2_profiling/ros2_profiling_demo/test/test_metrics.py
============================= test session starts ==============================
collected 9 items

src/ros2_profiling/ros2_profiling_demo/test/test_metrics.py ..F......    [100%]

=================================== FAILURES ===================================
_____________________________ test_chain_stdev _____________________________
src/ros2_profiling/ros2_profiling_demo/test/test_metrics.py:54: AssertionError
=========================== short test summary info ============================
FAILED src/ros2_profiling/ros2_profiling_demo/test/test_metrics.py::test_chain_stdev - assert 254.12609211822
97 < 100
========================= 1 failed, 8 passed in 3.93s ==========================
```

Profiling Toolchain



Profiling Toolchain



Take Home Messages

- Possibility to use a common profiling workflow for different ROS 2 applications and operating systems
- From trace analysis to certification artifacts generation, which are critical for any safety critical application (medical robotics, aerospace, automotive, etc.)
- In our quest towards SW Defined Medical Robotics, we aspire faster development to deployment cycle by automated certification artifacts generation

Potential Community Contribution

- Improving the Analysis API
- Explore new metrics and/or new tests/validations which might be leveraged in different ROS 2 applications
- Use these tools in different specific setups and provide feedback on what could be improved

Questions?

Marco Pedretti
Zoltan Resi
Shashank Sharma