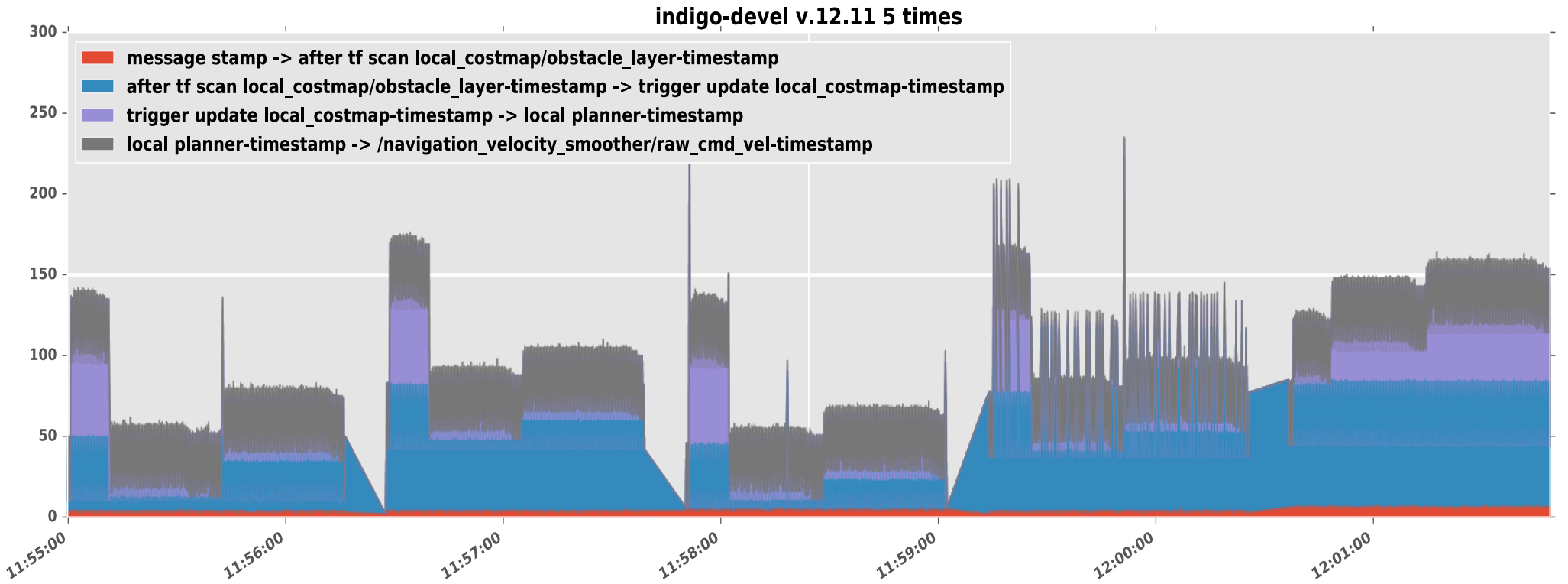
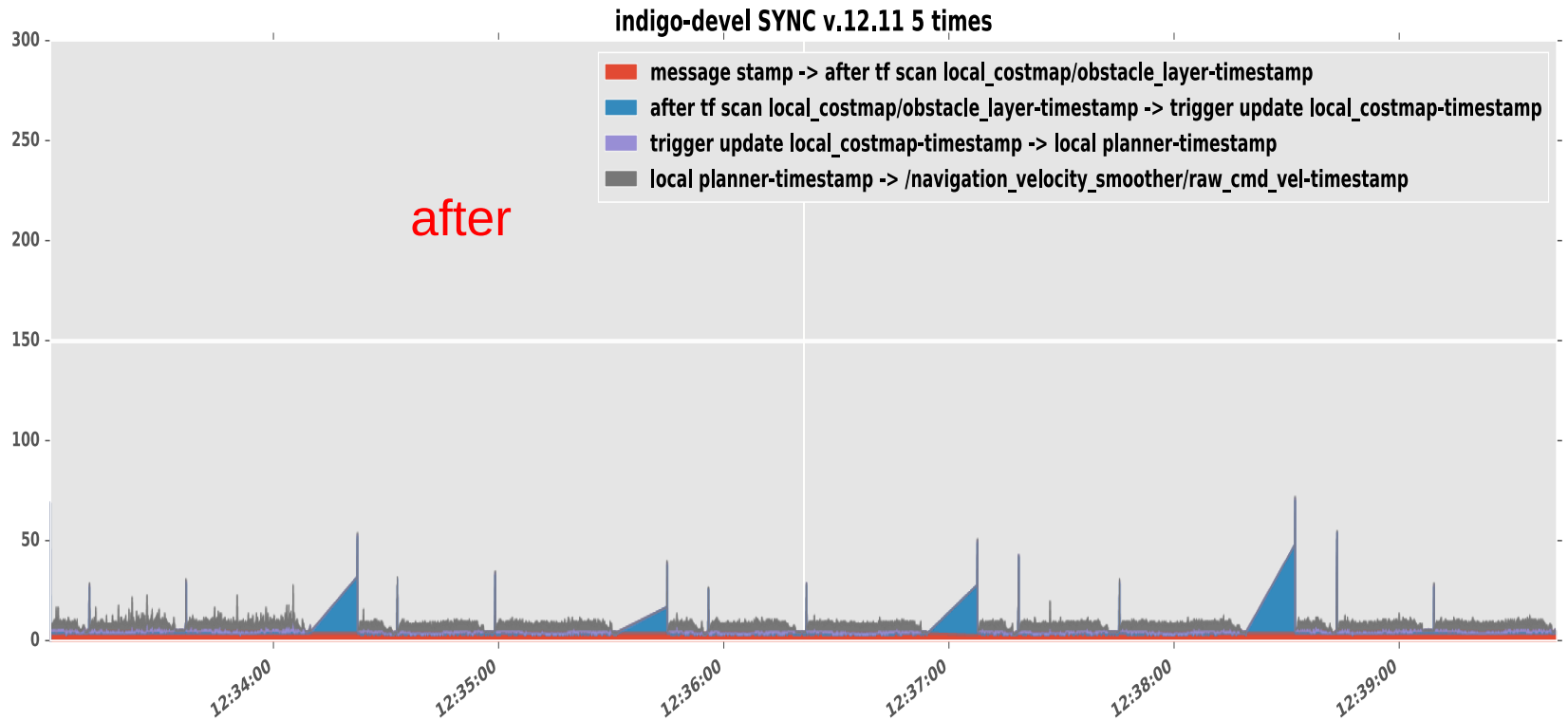
ROS2 TRACING



BOSCH



Agenda

1. Problems in performance analysis and execution monitoring
2. What is tracing?
3. ros2_tracing
    1. Installation
    2. Getting Started
    3. Built-in tracepoint recording and analysis
    4. Custom tracepoint recording
4. Outlook

BOSCH

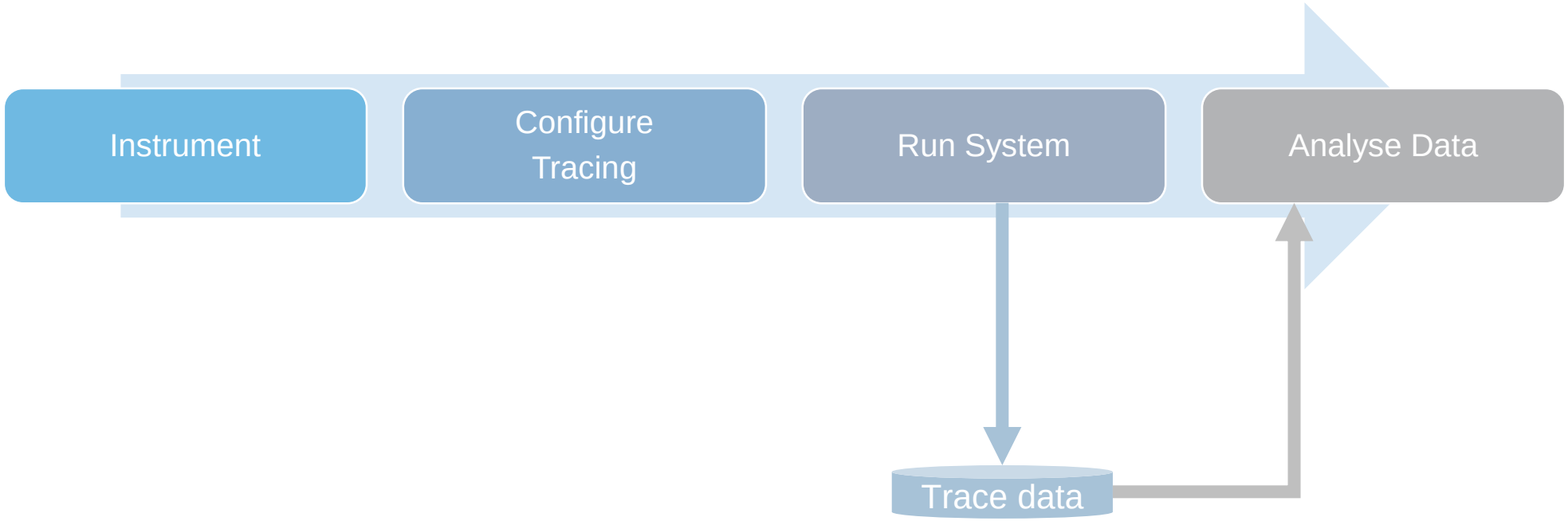
ROS2 Tracing
Problems in performance analysis and execution monitoring

- Typical questions
  - How long does my system take to react?
    - Corollary: Is my system always reacting fast enough?
  - How much resources is my system consuming → sizing compute hardware
    - Corollary: Where does the resource use come from?
  - Is my system still within its expected resource corridor?

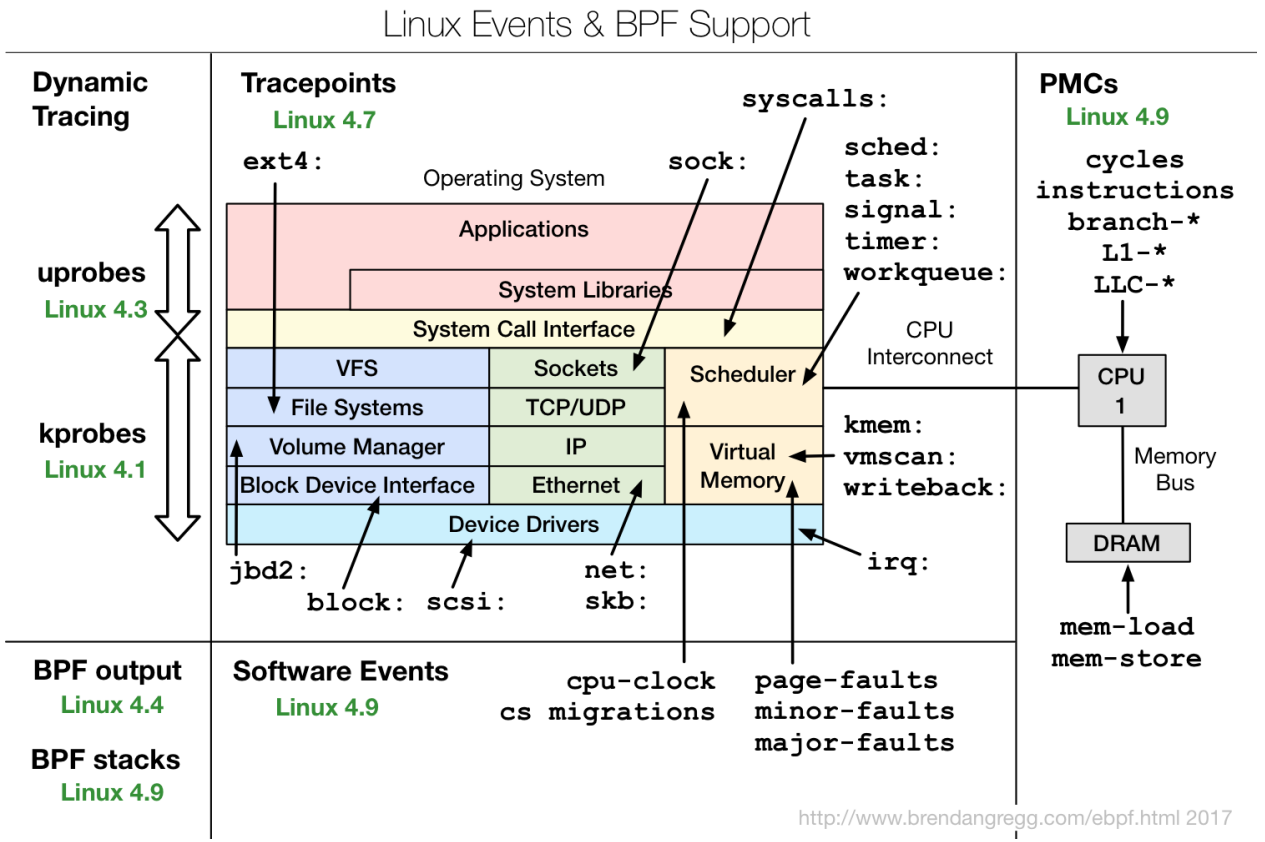
- Complicating factors
  - Distributed systems
    - Many nodes running concurrently
  - Repetitive periodic processing
    - Not all of which are equal
  - Performance Analysis: Low overhead important for correct data
  - Execution Monitoring: Low overhead paramount

ROS2 Tracing
What is tracing?

- Tracing: „record information about system execution“

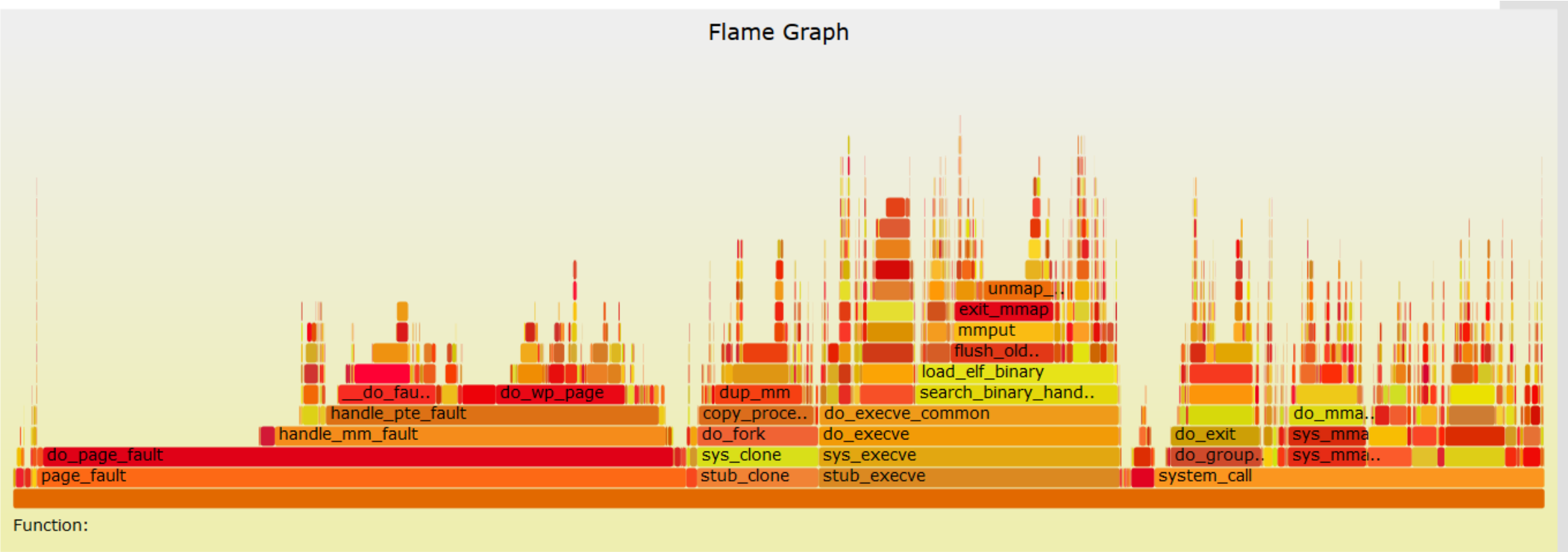


ROS2 Tracing
What kind of information can we record?



ROS2 Tracing

Example analyses: Brendan Gregg's famous flame graphs



Source: http://www.brendangregg.com/FlameGraphs/cpuflamegraphs.html

BOSCH

ROS2 Tracing
Tracepoints in ROS 2 (as of 12/2019)

- Initialization
- Invocation: 가

	User code				
rclcpp	Executor				
	Node	Subscription 가	Publisher 가	Service 가	Timer 가
rcl	rcl_node	rcl_subscription 가	rcl_publisher	rcl_service	rcl_timer
rmw	rmw_node	rmw_subscription	rmw_publisher	rmw_service	
DDS	Participant	Reader	Writer	Service	
Linux	OS Kernel				

- RMW-Layer is not very interesting, but could be added easily
- DDS has many implementations – not sure how to proceed here

ROS2 Tracing
Static tracing

- Static tracing: *Compile-time defined tracepoints in the source code, inserted by developers*
  - Pro: Encodes developer knowledge about what is important
  - Pro: Has direct access to all the data
  - Con: Takes effort to add for each tracepoint
  - Con: Possibly dependent on specific tracing framework

```
213 TRACEPOINT_EVENT(
214   TRACEPOINT_PROVIDER,
215   callback_start,
216   TP_ARGS(
217     const void *, callback_arg,
218     int, is_intra_process_arg
219   ),
220   TP_FIELDS(
221     ctf_integer_hex(const void *, callback, callback_arg)
222     ctf_integer(int, is_intra_process, is_intra_process_arg)
223   )
224 )
```

```
157   void dispatch(
158     std::shared_ptr<MessageT> message, const rmw_message_info_t & message_info)
159   {
160     TRACEPOINT(callback_start, (const void *)this, false);
```

ROS2 Tracing
Aside: Dynamic tracing

- Dynamic tracing: *Run-time defined tracepoints, configured by analyst*
  - Pro: Can be attached to any of the event sources with relatively little effort
  - Con: For uprobes, need to know the symbol you're attaching to → requires in-depth knowledge of code
  - Con: Currently only supported by kernel-based tracers → context-switching overhead

- Hint: You can often use dynamic tracing for your own code to add some extra info

ROS2 Tracing
Installation

- This is an excerpt from https://micro-ros.github.io/docs/tutorials/advanced/tracing/

```
sudo apt-add-repository ppa:lttng/stable-2.10
sudo apt-get update
sudo apt-get install lttng-tools lttng-modules-dkms liblttng-ust-dev
```

- Pre-Requisites
  - Linux-Trace-Toolkit ng
  - Babeltrace

```
wget https://gitlab.com/micro-ROS/ros_tracing/ros2_tracing/raw/master/tracing.repos
vcs import src < tracing.repos
```

```
colcon build --symlink-install --cmake-args " -DWITH_LTTNG=ON"
source ./install/local_setup.bash
```

- We have a repo-list for use with vcs
- Main thing: Build with –DWITH_LTTNG=ON!
- By the magic of dynamic linking, now you can trace every ROS 2 application ;-)

ROS2 Tracing
Tracing your system

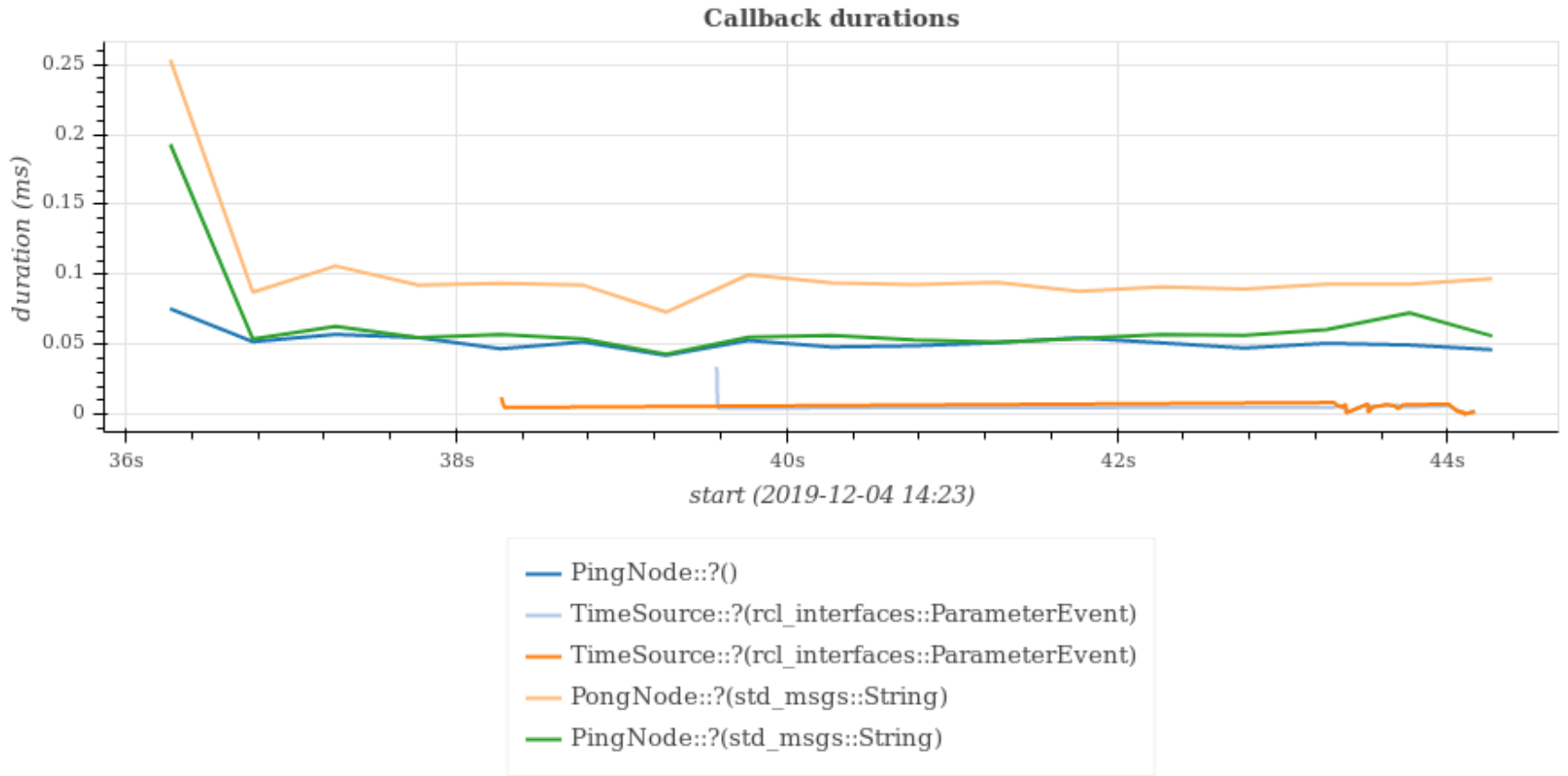
- Option 1: „ros2 trace“
- Option 2: „Trace“ action in launch file

```python
from launch import LaunchDescription
from launch_ros.actions import Node
from tracetools_launch.action import Trace


def generate_launch_description():
    return LaunchDescription([
        Trace(
            session_name='my-tracing-session',
        ),
        Node(
            package='tracetools_test',
            node_executable='test_ping',
            output='screen',
        ),
        Node(
            package='tracetools_test',
            node_executable='test_pong',
            output='screen',
        ),
    ])
```

ROS2 Tracing
Analyzing the trace part 1: CLI

```
lui3si@RNGX7819:~/src/tracing_ws$ ros2 run tracetools_analysis cb_durations ~/.ros/tracing/my-tracing-session/converted

   Count       Sum      Mean       Std                                                Name
3     17  1.731224  0.101837  0.039629                          PongNode::?(std_msgs::String)
4     17  1.093302  0.064312  0.033667                          PingNode::?(std_msgs::String)
0     17  0.882704  0.051924  0.007078                                        PingNode::?()
1     57  0.143689  0.002521  0.004859  TimeSource::?(rcl_interfaces::ParameterEvent)
2     58  0.134670  0.002322  0.002937  TimeSource::?(rcl_interfaces::ParameterEvent)
```

ROS2 Tracing
Analyzing the trace part 2: Jupyter Notebook



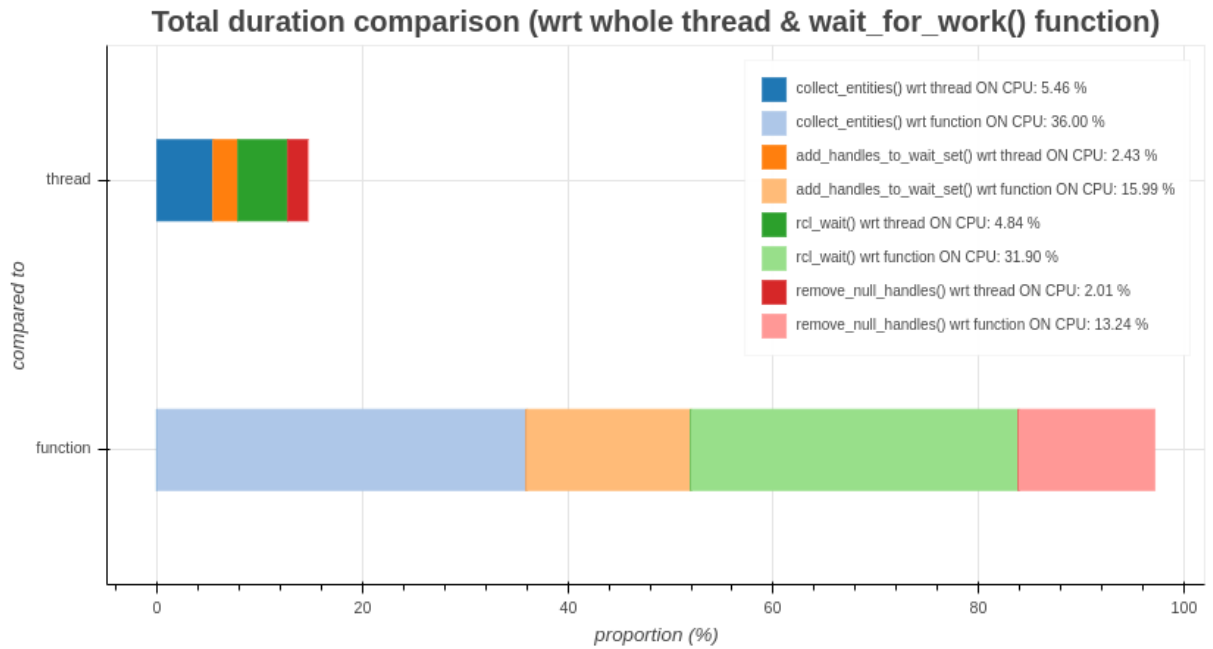
ROS2 Tracing
Custom traces: Function instrumentation

- Sometimes, a more detailed view is needed → custom tracepoints
- A simple approach is function-instrumentation with –finstrument-functions
  - By default, this has too much overhead
  - The “instrument-attribute-gcc-plugin” by Christophe Bourque Bedard addresses this
- Usage:
  - Add instrumentation attribute
  - Compile with -fplugin=./instrument_attribute.so
- This is essentially a selective form of profiling

```
void __attribute__((instrument_function)) instrumented_function()
{
  printf("this is instrumented\n");
}
```

ROS2 Tracing
Custom Trace Example: Executor profiling

- This summer, several people noticed that the ROS 2 SingleThreadedExecutor can have significant CPU overhead. See https://discourse.ros.org/t/singlethreadedexecutor-creates-a-high-cpu-overhead-in-ros-2/10077/10
- We traced this using custom tracepoints in the executor
- Result:



ROS2 Tracing
Outlook

- Tracepoints for services etc.
- More analyses provided out-of-the-box
- Performance improvement for analysis
- Live tracing
- Capturing data from ebpf-tracing

ROS2 Tracing
Conclusion

- Tracing is an excellent infrastructure for system-level analysis
  - Both for kernel and user-space
- Bosch has contributed initial tracing support for ROS 2
  - Tracepoints in framework
  - Integration with tooling
  - Basic analysis scripts

- We have many interesting internship / thesis projects in this area