# Performance Measurement of Robotics Applications with ros2_benchmark

CY Chen | May 4th, 2023

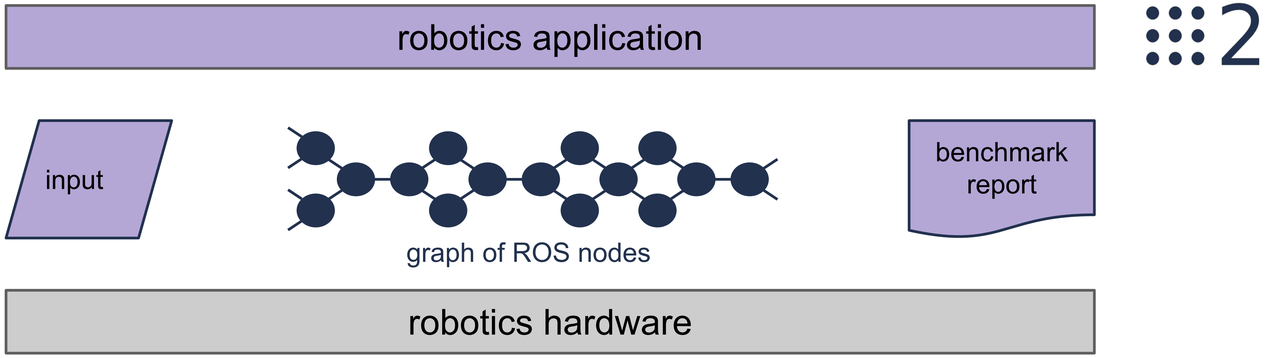
# Agenda

- Introduction to ros2_benchmark
- Introduction to r2b dataset 2023
- ros2_benchmark demo
- Benchmark Isaac ROS hardware accelerated graphs
- Create your own benchmark with ros2_benchmark

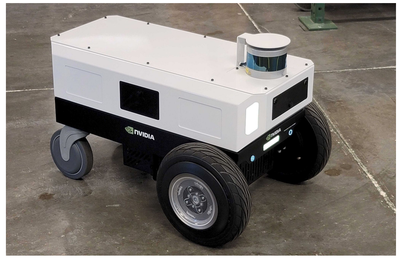
# Motivation

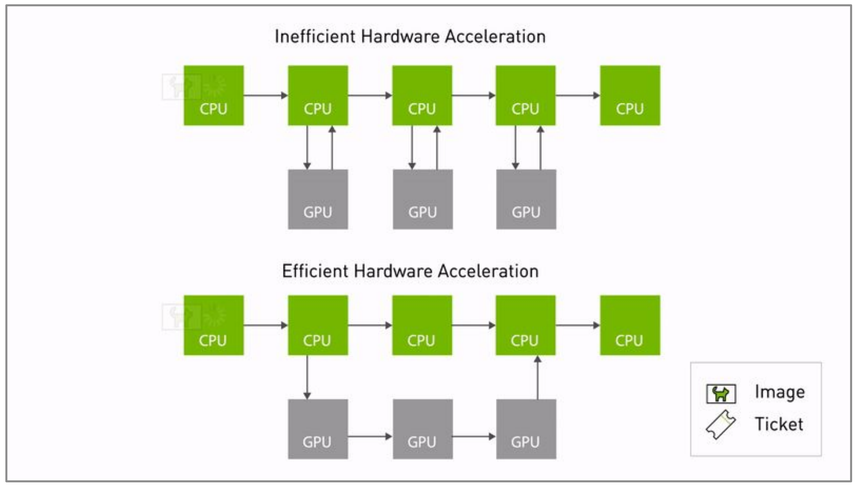
## Benchmarking ROS 2 Graphs

- Robots are real-time systems which require complex graphs of heterogeneous computation to perform perception, planning, and control
- These graphs of computation need to perform work deterministically and with known latency
- The computing platform has a fixed budget for heterogeneous computation (TOPS) and throughput; computation is typically performed on multiple CPUs, GPUs, and additional special purpose, fixed function hardware accelerators



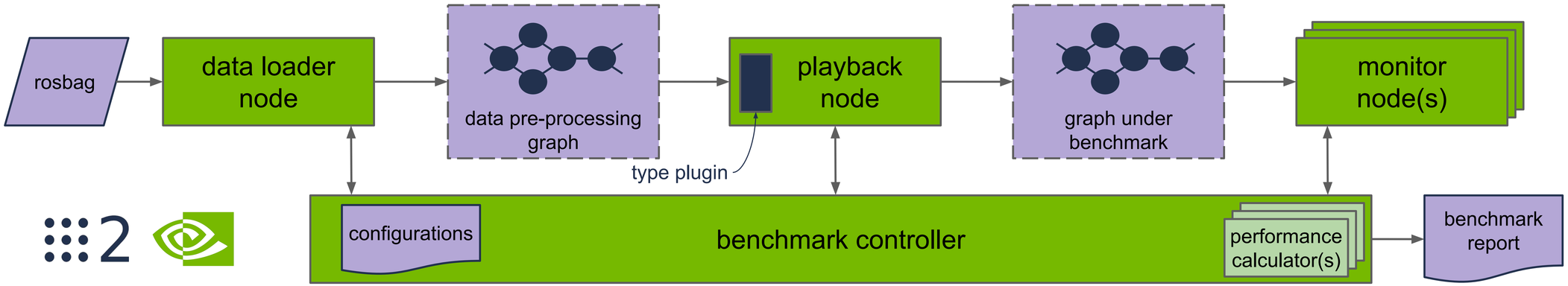

**Heterogeneous Computation on Robots**





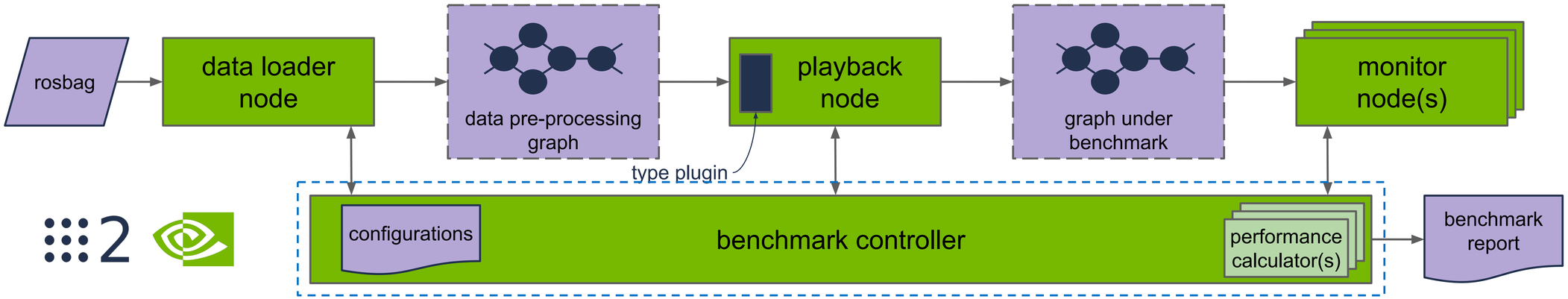
# ros2_benchmark

## Our Solution



# ros2_benchmark

## Benchmark Controller

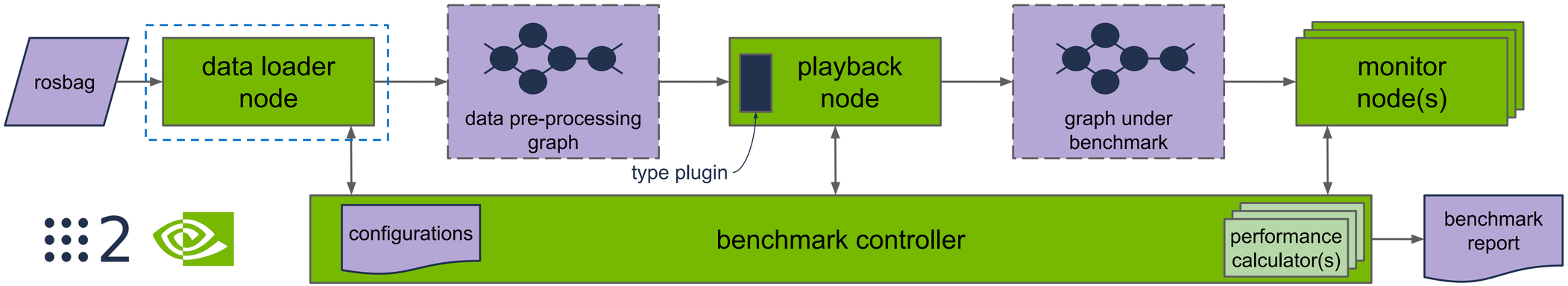



- **Benchmark Controller**
  - Orchestrates the benchmark flow
  - Loads benchmark configurations
  - Calculates performance results
  - Generates benchmark report
  - Started by using *launch_test*

# ros2_benchmark

## Data Loader Node

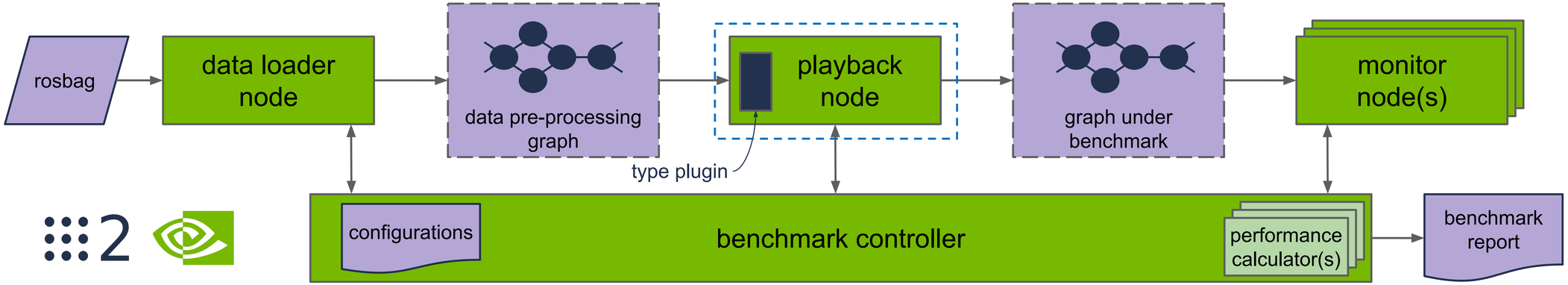
- **Data Loader Node**
  - Loads a rosbag
  - Publishes messages from specified time range



# ros2_benchmark

## Playback Node

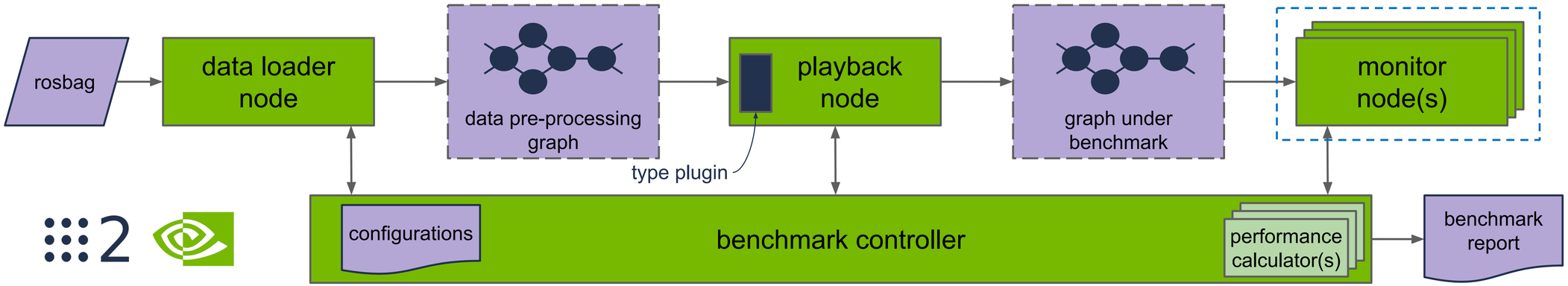
- **Playback Node**
  - Buffers messages
  - Plays messages in two modes
  - Records message start time points



# ros2_benchmark

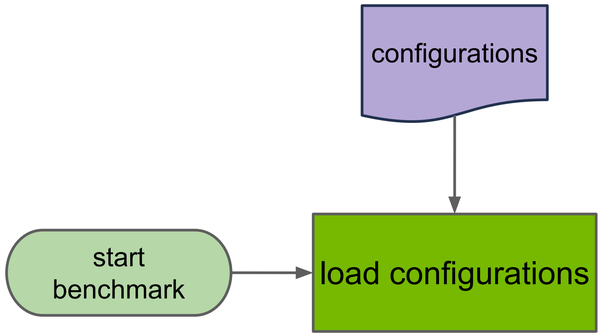
## Monitor Node

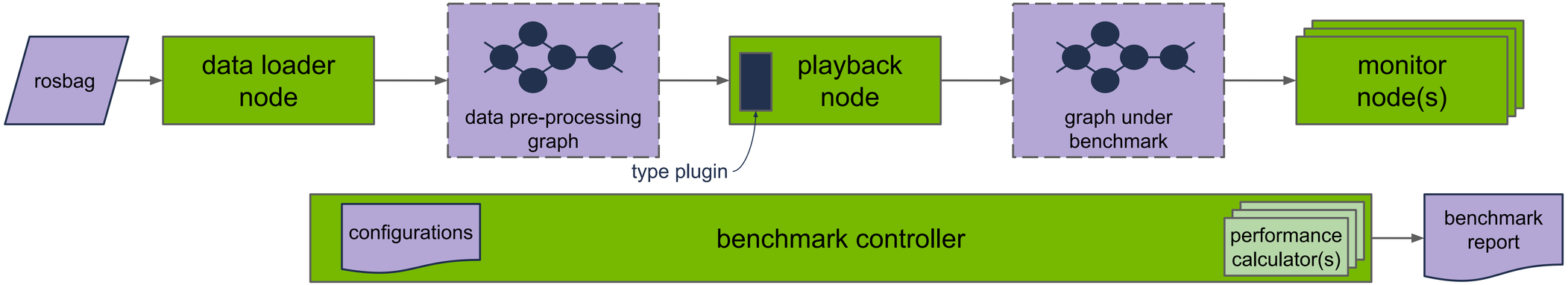
- **Monitor Node**
  - Records message arrival time points



# Benchmark Flow

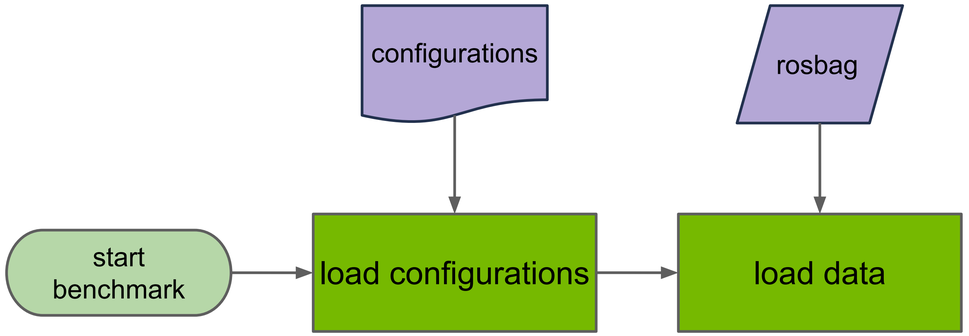
## Communication between Benchmark Nodes

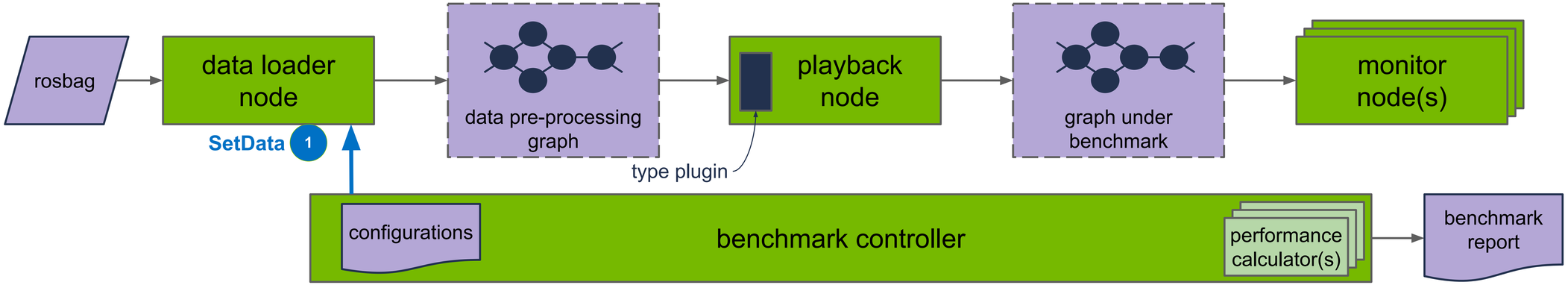


# Benchmark Flow

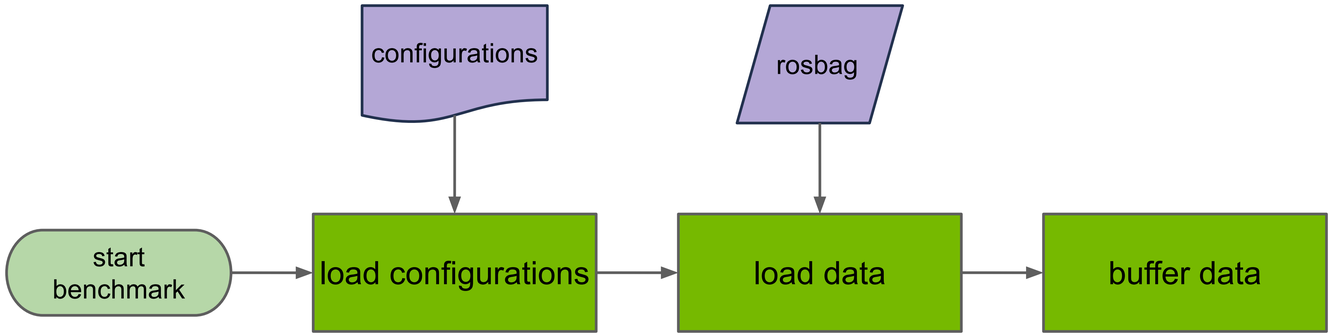
## Communication between Benchmark Nodes

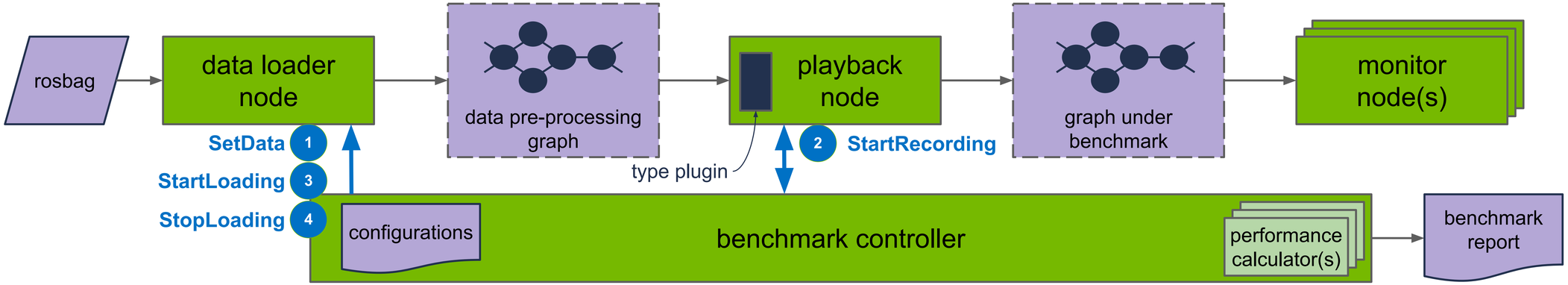


# Benchmark Flow

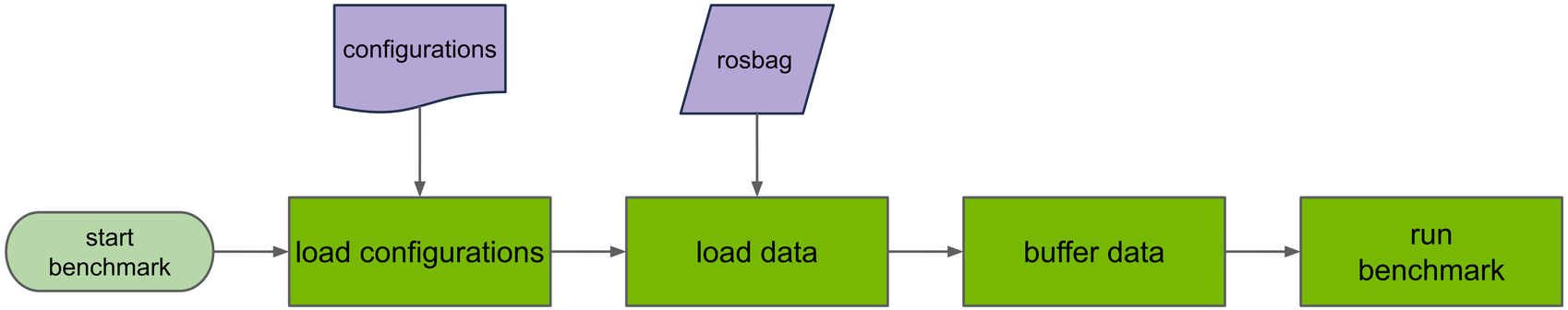
## Communication between Benchmark Nodes

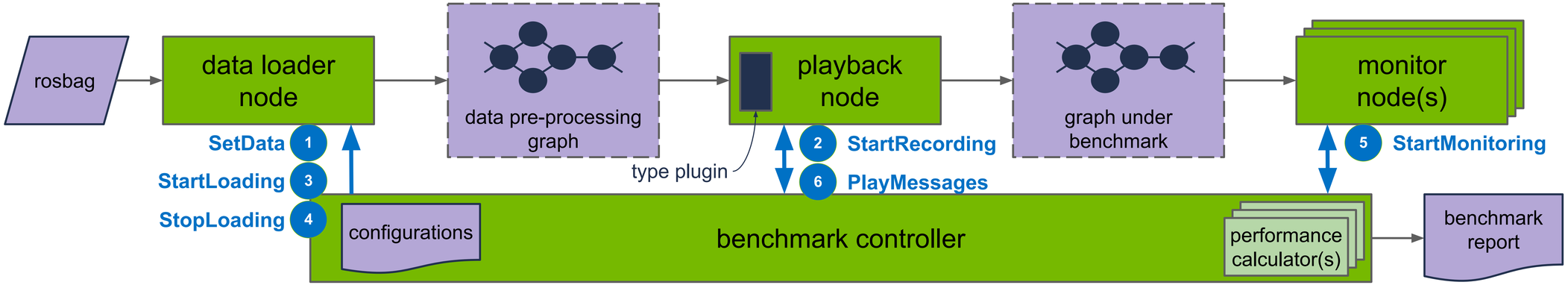


# Benchmark Flow

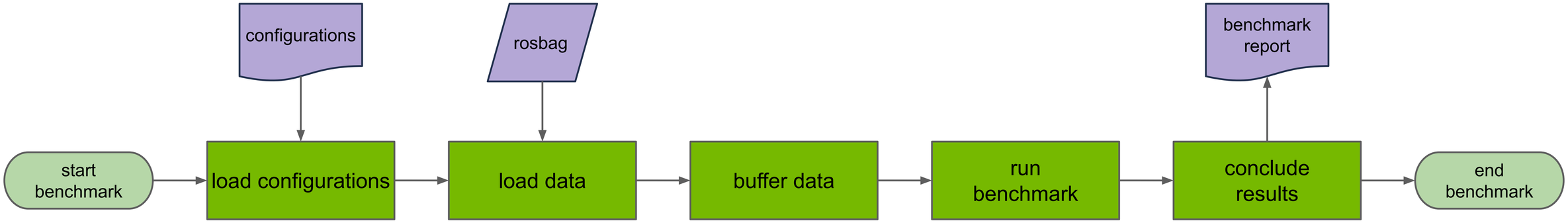
## Communication between Benchmark Nodes

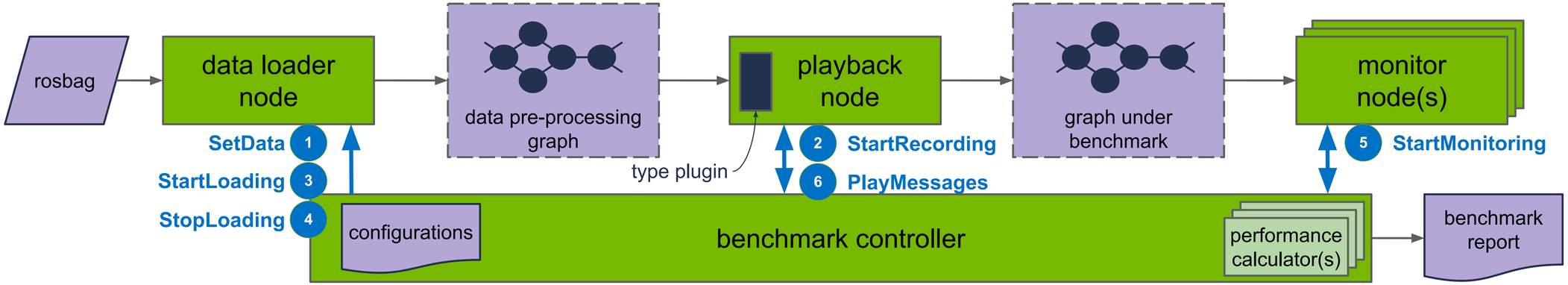


# Benchmark Flow

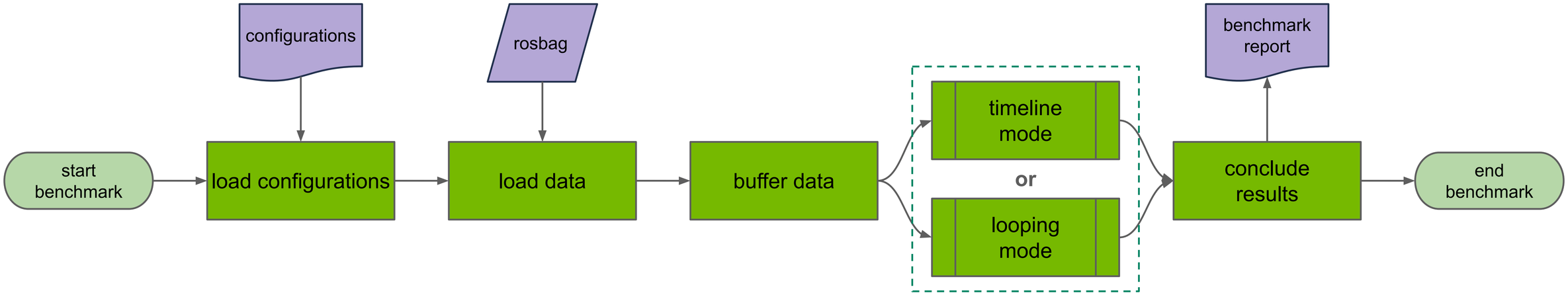
## Communication between Benchmark Nodes



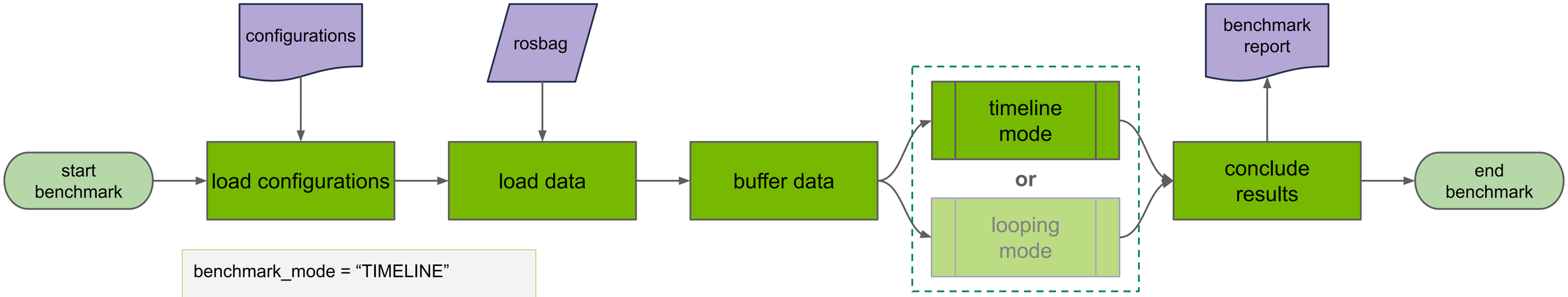
# Benchmark Modes

## Timeline + Looping



# Timeline Benchmark Mode

## Play Messages Based on Recorded Timeline



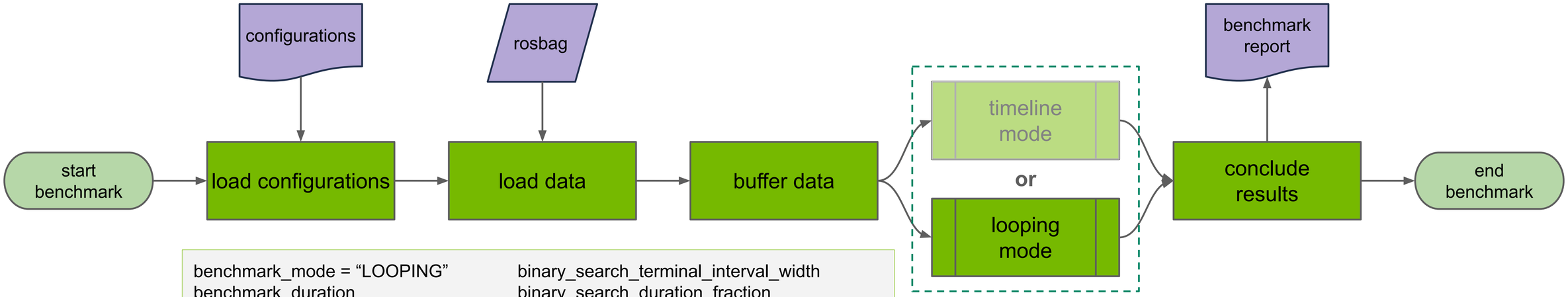

```
benchmark_mode = “TIMELINE”

input_data_path
input_data_start_time
input_data_end_time
publish_tf_static_messages_in_set_data
playback_message_buffer_size = 0
```

- **Timeline Mode**
  - Play buffered messages based on the recording timeline

# Looping Benchmark Mode

## Play Buffered Messages




benchmark_mode = “LOOPING”
benchmark_duration
test_iterations
publisher_upper_frequency
publisher_lower_frequency
additional_fixed_publisher_rate_tests

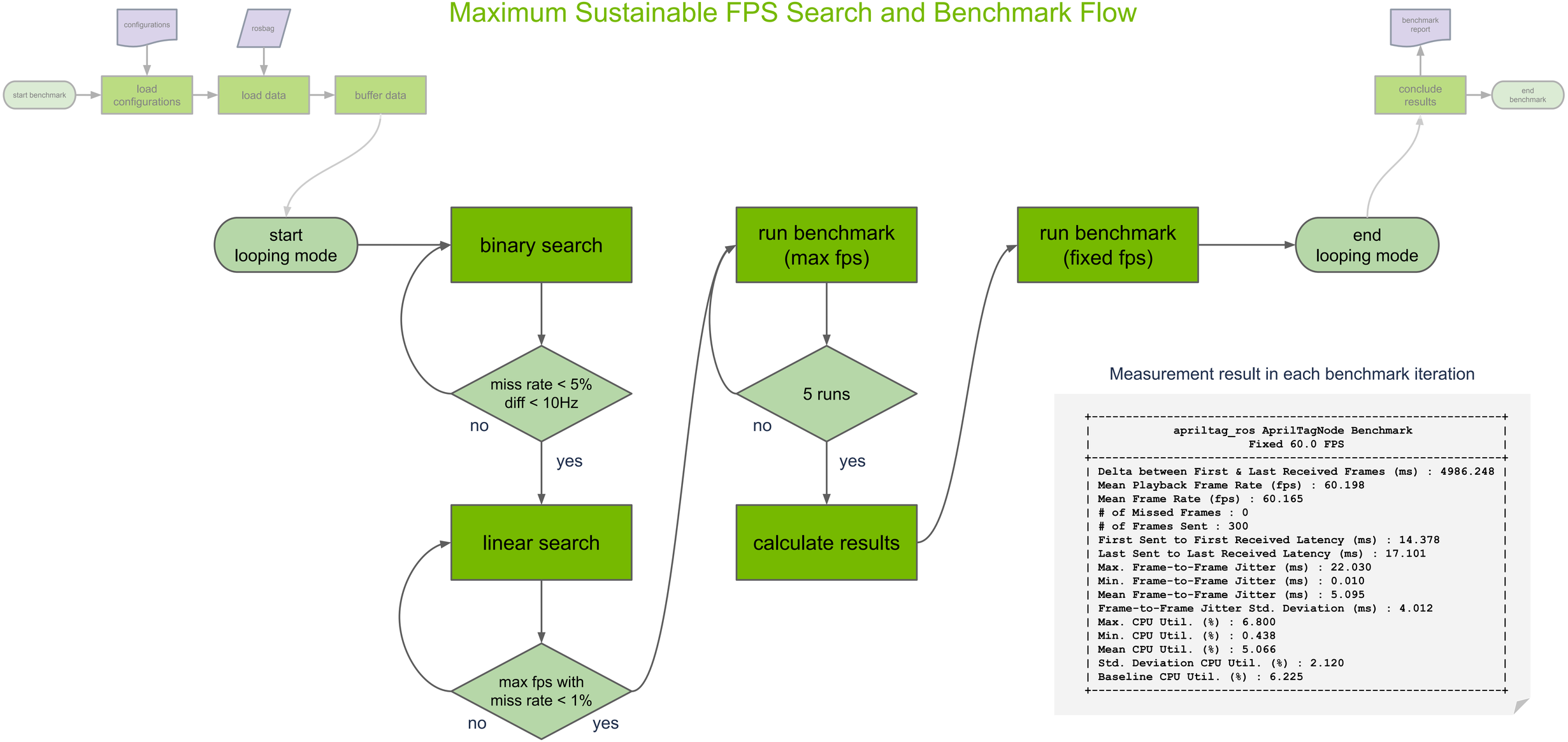
input_data_path
input_data_start_time
input_data_end_time
publish_tf_static_messages_in_set_data
playback_message_buffer_size

binary_search_terminal_interval_width
binary_search_duration_fraction
binary_search_acceptable_frame_loss_fraction
binary_search_acceptable_frame_rate_drop
linear_scan_step_size
linear_scan_duration_fraction
linear_scan_acceptable_frame_loss_fraction
linear_scan_acceptable_frame_rate_drop

- **Looping Mode**
  - Searches the maximum sustainable fps
  - Plays buffered messages at a fixed rate

# Looping Benchmark Mode

## Maximum Sustainable FPS Search and Benchmark Flow




Measurement result in each benchmark iteration

```
+-------------------------------------------------------------+
|            apriltag_ros AprilTagNode Benchmark              |
|                        Fixed 60.0 FPS                       |
+-------------------------------------------------------------+
| Delta between First & Last Received Frames (ms) : 4986.248 |
| Mean Playback Frame Rate (fps) : 60.198                     |
| Mean Frame Rate (fps) : 60.165                              |
| # of Missed Frames : 0                                      |
| # of Frames Sent : 300                                      |
| First Sent to First Received Latency (ms) : 14.378          |
| Last Sent to Last Received Latency (ms) : 17.101            |
| Max. Frame-to-Frame Jitter (ms) : 22.030                    |
| Min. Frame-to-Frame Jitter (ms) : 0.010                     |
| Mean Frame-to-Frame Jitter (ms) : 5.095                     |
| Frame-to-Frame Jitter Std. Deviation (ms) : 4.012           |
| Max. CPU Util. (%) : 6.800                                  |
| Min. CPU Util. (%) : 0.438                                  |
| Mean CPU Util. (%) : 5.066                                  |
| Std. Deviation CPU Util. (%) : 2.120                        |
| Baseline CPU Util. (%) : 6.225                              |
+-------------------------------------------------------------+
```

# Benchmark Report

## Exported in JSON

Benchmark Performance Conclusion

Fixed Rate Benchmark Results

User-defined Key-values

Benchmark Metadata

```
+-----------------------------------------------------------------------------------------------+
|                              apriltag_ros AprilTagNode Benchmark                              |
|                                          Final Report                                         |
+-----------------------------------------------------------------------------------------------+
| Delta between First & Last Received Frames (ms) : 4986.822                                    |
| Mean Playback Frame Rate (fps) : 102.387                                                      |
| Mean Frame Rate (fps) : 99.827                                                                |
| # of Missed Frames : 12.333                                                                   |
| # of Frames Sent : 510.000                                                                    |
| First Sent to First Received Latency (ms) : 10.216                                            |
| Last Sent to Last Received Latency (ms) : 15.833                                              |
| Max. Frame-to-Frame Jitter (ms) : 6.377                                                       |
| Min. Frame-to-Frame Jitter (ms) : 0.000                                                       |
| Mean Frame-to-Frame Jitter (ms) : 0.747                                                       |
| Frame-to-Frame Jitter Std. Deviation (ms) : 0.835                                             |
| Max. CPU Util. (%) : 9.162                                                                    |
| Min. CPU Util. (%) : 0.188                                                                    |
| Mean CPU Util. (%) : 4.086                                                                    |
| Std. Deviation CPU Util. (%) : 3.854                                                          |
| Baseline CPU Util. (%) : 8.390                                                                |
+-----------------------------------------------------------------------------------------------+
| [60.0fps] Delta between First & Last Received Frames (ms) : 4986.248                          |
| [60.0fps] Mean Playback Frame Rate (fps) : 60.198                                             |
| [60.0fps] Mean Frame Rate (fps) : 60.165                                                      |
| [60.0fps] # of Missed Frames : 0                                                              |
| [60.0fps] # of Frames Sent : 300                                                              |
| [60.0fps] First Sent to First Received Latency (ms) : 14.378                                  |
| [60.0fps] Last Sent to Last Received Latency (ms) : 17.101                                    |
| [60.0fps] Max. Frame-to-Frame Jitter (ms) : 22.030                                            |
| [60.0fps] Min. Frame-to-Frame Jitter (ms) : 0.010                                             |
| [60.0fps] Mean Frame-to-Frame Jitter (ms) : 5.095                                             |
| [60.0fps] Frame-to-Frame Jitter Std. Deviation (ms) : 4.012                                   |
| [60.0fps] Max. CPU Util. (%) : 6.800                                                          |
| [60.0fps] Min. CPU Util. (%) : 0.438                                                          |
| [60.0fps] Mean CPU Util. (%) : 5.066                                                          |
| [60.0fps] Std. Deviation CPU Util. (%) : 2.120                                                |
| [60.0fps] Baseline CPU Util. (%) : 6.225                                                      |
+-----------------------------------------------------------------------------------------------+
| [custom] data_resolution : HD (1280,720)                                                      |
+-----------------------------------------------------------------------------------------------+
| [metadata] Test Name : apriltag_ros AprilTagNode Benchmark                                    |
| [metadata] Test File Path : /workspaces/isaac_ros-dev/ros_ws/src/ros2_benchmark/scripts/ap    |
| [metadata] Test Datetime : 2023-04-06T07:09:44Z                                               |
| [metadata] Device Hostname : ros                                                              |
| [metadata] Device Architecture : x86_64                                                       |
| [metadata] Device OS : Linux 5.4.0-131-generic #147-Ubuntu SMP Fri Oct 14 17:07:22 UTC 202    |
| [metadata] Benchmark Mode : 1                                                                 |
| [metadata] Peak Throughput Prediction (Hz) : 102.188                                          |
| [metadata] Input Data Path : /workspaces/isaac_ros-dev/ros_ws/src/ros2_benchmark/assets/da    |
| [metadata] Input Data Size (bytes) : 3087908864                                               |
| [metadata] Input Data Hash : 5e8f11201fe10dbac7307a8628553e94                                 |
| [metadata] Input Data Start Time (s) : 3.000                                                  |
| [metadata] Input Data End Time (s) : 3.500                                                    |
+-----------------------------------------------------------------------------------------------+
```

# r2b Dataset 2023

## a collection of sequences stored in rosbags

https://catalog.ngc.nvidia.com/orgs/nvidia/teams/isaac/resources/r2bdataset2023

NVIDIA. NGC | CATALOG

Welcome Guest

Catalog > Resources > r2b dataset 2023

### r2b dataset 2023

Download

Overview | Version History | File Browser | Release Notes | Related Collections

**Description**

The r2b dataset 2023 is a collection of live recorded sequences stored in rosbags (a ROS 2 native format used for message data) including time-synchronized sensor captures from a robot.

**Publisher**

NVIDIA

**Use Case**

Other

**Framework**

Other

**Latest Version**

1

**Modified**

April 19, 2023

**Compressed Size**

14.49 GB

### r2b Dataset 2023

#### Dataset Overview

The r2b dataset 2023 is a collection of sequences stored in rosbags, a ROS 2 native format used for message data including time synchronized sensor captures from a robot. The dataset can be loaded and played back into an application running a ROS graph of nodes for testing and performance benchmarking of image processing, perception and localization. Each sequence consists of a single sequence in a scene, providing a small diversity of data across the collection of sequences in this dataset.

Several sequences contain sensor multi-second captures including LIDAR, stereo camera, and IMU data from HAWK stereo camera, RealSense D455, and XT32 with highly precise time synchronization. Other sequences contain data recorded from a simulated scene in Isaac Sim. The `r2b_hope` sequence is distinct as it contains D415 RGB data.

| Sequence | Visual | Description |
|---|---|---|
| `r2b_lounge` |  | Lounge sequence containing couch, table, chairs, and staircase with natural planted background wall. |
| `r2b_storage` |  | Storage sequence including person, AprilTag, shoe, shelving, boxes, pallets, with moving obstacles. |
| `r2b_hallway` |  | Hallway sequence with walking persons, low to no feature not-perpendicular walls, specular highlights, and reflections. |
| `r2b_datacenter` |  | Datacenter sequence with tall vertical corridor, repetitive low-feature surfaces, and little color. |
| |  | Café sequence including table, chairs, stools, reflective |

r2b_lounge

r2b_storage

r2b_hallway

r2b_datacenter

r2b_cafe

r2b_hope

r2b_hideaway

r2b_mezzanine

NVIDIA

# r2b Dataset 2023

a collection of sequences stored in rosbags

| Sequence | Topics Provided | Message Type |
|---|---|---|
| r2b_lounge | d455_1_depth | sensor_msgs/msg/Image |
| r2b_storage | d455_1_imu | sensor_msgs/msg/Imu |
| r2b_hallway | d455_1_left_ir_camera_info | sensor_msgs/msg/CameraInfo |
| r2b_datacenter | d455_1_left_ir_image | sensor_msgs/msg/Image |
| r2b_cafe | d455_1_rgb_camera_info | sensor_msgs/msg/CameraInfo |
| r2b_hideaway | d455_1_rgb_image | sensor_msgs/msg/Image |
| r2b_mezzanine | d455_1_right_ir_camera_info | sensor_msgs/msg/CameraInfo |
| | d455_1_right_ir_image | sensor_msgs/msg/Image |
| | hawk_0_left_rgb_camera_info | sensor_msgs/msg/CameraInfo |
| | hawk_0_left_rgb_image | sensor_msgs/msg/Image |
| | hawk_0_right_rgb_camera_info | sensor_msgs/msg/CameraInfo |
| | hawk_0_right_rgb_image | sensor_msgs/msg/Image |
| | pandar_xt_32_0_lidar | sensor_msgs/msg/PointCloud2 |
| r2b_hope | image | sensor_msgs/msg/Image |


r2b_lounge


r2b_storage


r2b_hallway


r2b_datacenter


r2b_cafe


r2b_hope


r2b_hideaway


r2b_mezzanine

# ros2_benchmark Quickstart

## Run apriltag_ros AprilTagNode Benchmark

1. launch official Docker container with ROS 2 Humble pre-installed

   *docker run -it ros:humble*

2. Setup environment and install utility tools
3. Clone ros2_benchmark and demo package (apriltag_ros) and install dependencies
4. Clone, patch, and build image_proc package with required, backported fix for image resize (reference)
5. Pull down r2b dataset 2023
6. Build ros2_benchmark and demo package

   *colcon build --packages-up-to ros2_benchmark apriltag_ros*

7. Start the AprilTag benchmark

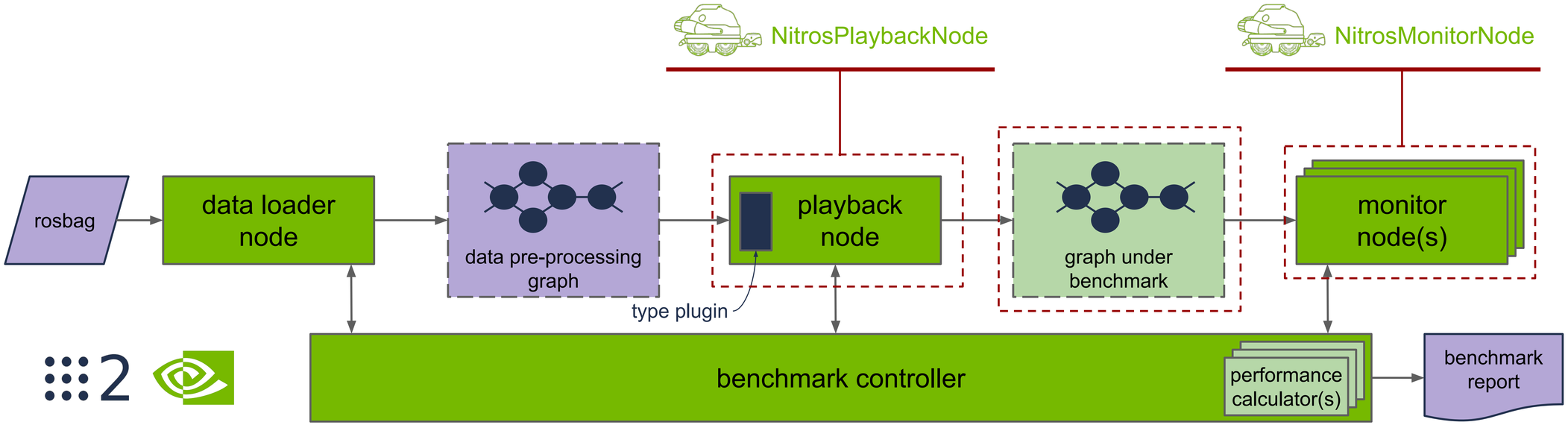
   *launch_test src/ros2_benchmark/scripts/apriltag_ros_apriltag_node.py*

https://github.com/NVIDIA-ISAAC-ROS/ros2_benchmark#quickstart

# Demo: ros2_benchmark

# Benchmark Isaac ROS Hardware Accelerated Graphs

## Isaac ROS Benchmark




| | | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| Apriltag node | Apriltag graph | freespace segmentation node | freespace segmentation graph | Bi3D node | centerpose graph | detectnet graph | disparity node | disparity graph |
| DNN image encoder node | DOPE graph | ESS node | ESS graph | occupancy grid localizer node | h264 decoder node | h264 encoder (iframe) node | h264 encoder (pframe) node | Nvblox node |
| rectify node | Tensor RT DOPE node | Tensor RT PeopleSemSegnet node | Triton DOPE node | Triton PeopleSemSegnet node | UNET graph | VSLAM node | | |

https://github.com/NVIDIA-ISAAC-ROS/isaac_ros_benchmark

# isaac_ros_benchmark Quickstart

## Run Isaac ROS AprilTagNode Benchmark

1. Setup Isaac ROS environment and install utility tools by following the instructions in isaac_ros_common (instructions)
2. launch Isaac ROS Docker container with ROS 2 Humble and dependencies pre-installed

   ```
   cd ~/workspaces/isaac_ros-dev/src/isaac_ros_common && \
   ./scripts/run_dev.sh
   ```

3. Clone ros2_benchmark, isaac_ros_benchmark and demo packages and install dependencies
4. Pull down r2b dataset 2023
5. Build isaac_ros_benchmark and demo packages

   ```
   colcon build --symlink-install --packages-up-to isaac_ros_benchmark isaac_ros_apriltag
   ```

6. Start the AprilTag benchmark

   ```
   launch_test src/isaac_ros_benchmark/scripts/isaac_ros_apriltag_node.py
   ```

https://github.com/NVIDIA-ISAAC-ROS/isaac_ros_benchmark#quickstart

# Create Your Own Benchmark

witht ros2_benchmark: high level script template overview

Benchmark Graph Declarations

Benchmark Configurations

```python
from launch_ros.actions import ComposableNodeContainer
from launch_ros.descriptions import ComposableNode

from ros2_benchmark import ImageResolution
from ros2_benchmark import ROS2BenchmarkConfig, ROS2BenchmarkTest

def launch_setup(container_prefix, container_sigterm_timeout):
    """Graph setup for benchmarking your custom graph."""
    # Insert your composable node declarations

    # Required DataLoaderNode

    # Insert your custom preprocessor graph if needed

    # Required PlaybackNode

    # Required MonitorNode

    # Required composable node container

    return [composable_node_container]

def generate_test_description():
    return TestCustomGraph.generate_test_description_with_nsys(launch_setup)

class TestCustomGraph(ROS2BenchmarkTest):
    """Performance test for your custom graph."""

    # Insert custom configurations

    def test_benchmark(self):
        self.run_benchmark()
```

https://github.com/NVIDIA-ISAAC-ROS/ros2_benchmark#creating-custom-benchmark

# Create Your Own Benchmark

witht ros2_benchmark

1. Insert your custom graph (e.g., composable nodes) in the launch_setup method.

```python
from launch_ros.actions import ComposableNodeContainer
from launch_ros.descriptions import ComposableNode

from ros2_benchmark import ImageResolution
from ros2_benchmark import ROS2BenchmarkConfig, ROS2BenchmarkTest

def launch_setup(container_prefix, container_sigterm_timeout):
    """Graph setup for benchmarking your custom graph."""

    # Insert your composable node declarations

    # Required DataLoaderNode
    data_loader_node = ComposableNode(
        name='DataLoaderNode',
        namespace=TestCustomGraph.generate_namespace(),
        package='ros2_benchmark',
        plugin='ros2_benchmark::DataLoaderNode',
        # Insert remappings if necessary
    )

    # Insert your custom preprocessor graph if needed

    # Required PlaybackNode
    playback_node = ComposableNode(
        name='PlaybackNode',
        namespace=TestCustomGraph.generate_namespace(),
        package='ros2_benchmark',
```

```python
from launch_ros.actions import ComposableNodeContainer
from launch_ros.descriptions import ComposableNode

from ros2_benchmark import ImageResolution
from ros2_benchmark import ROS2BenchmarkConfig, ROS2BenchmarkTest

def launch_setup(container_prefix, container_sigterm_timeout):
    """Graph setup for benchmarking your custom graph."""
    Insert your composable node declarations

    # Required DataLoaderNode

    # Insert your custom preprocessor graph if needed

    # Required PlaybackNode

    # Required MonitorNode

    # Required composable node container

    return [composable_node_container]

def generate_test_description():
    return TestCustomGraph.generate_test_description_with_nsys(launch_setup)

class TestCustomGraph(ROS2BenchmarkTest):
    """Performance test for your custom graph."""

    # Insert custom configurations

    def test_benchmark(self):
        self.run_benchmark()
```

# Create Your Own Benchmark

witht ros2_benchmark

1. Insert your custom graph (e.g., composable nodes) in the launch_setup method.
2. Revise remappings in the data loader node to connect rosbag topics to either your preprocessor graph or a playback node.

```
from ros2_benchmark import ImageResolution
from ros2_benchmark import ROS2BenchmarkConfig, ROS2BenchmarkTest

def launch_setup(container_prefix, container_sigterm_timeout):
    """Graph setup for benchmarking your custom graph."""

    # Insert your composable node declarations

    # Required DataLoaderNode
    data_loader_node = ComposableNode(
        name='DataLoaderNode',
        namespace=TestCustomGraph.generate_namespace(),
        package='ros2_benchmark',
        plugin='ros2_benchmark::DataLoaderNode',
        # Insert remappings if necessary
    )

    # Insert your custom preprocessor graph if needed

    # Required PlaybackNode
    playback_node = ComposableNode(
        name='PlaybackNode',
        namespace=TestCustomGraph.generate_namespace(),
        package='ros2_benchmark',
```

2

```
from launch_ros.actions import ComposableNodeContainer
from launch_ros.descriptions import ComposableNode

from ros2_benchmark import ImageResolution
from ros2_benchmark import ROS2BenchmarkConfig, ROS2BenchmarkTest

def launch_setup(container_prefix, container_sigterm_timeout):
    """Graph setup for benchmarking your custom graph."""
    # Insert your composable node declarations

    # Required DataLoaderNode

    # Insert your custom preprocessor graph if needed

    # Required PlaybackNode

    # Required MonitorNode

    # Required composable node container

    return [composable_node_container]

def generate_test_description():
    return TestCustomGraph.generate_test_description_with_nsys(launch_setup)

class TestCustomGraph(ROS2BenchmarkTest):
    """Performance test for your custom graph."""

    # Insert custom configurations

    def test_benchmark(self):
        self.run_benchmark()
```

1

2

# Create Your Own Benchmark

witht ros2_benchmark

1. Insert your custom graph (e.g., composable nodes) in the launch_setup method.
2. Revise remappings in the data loader node to connect rosbag topics to either your preprocessor graph or a playback node.
3. [optional] Insert your preprocessor graph in the launch_setup method if required.

```
from ros2_benchmark import ROS2BenchmarkConfig, ROS2BenchmarkTest

def launch_setup(container_prefix, container_sigterm_timeout):
    """Graph setup for benchmarking your custom graph."""

    # Insert your composable node declarations

    # Required DataLoaderNode
    data_loader_node = ComposableNode(
        name='DataLoaderNode',
        namespace=TestCustomGraph.generate_namespace(),
        package='ros2_benchmark',
        plugin='ros2_benchmark::DataLoaderNode',
        # Insert remappings if necessary
    )

3   # Insert your custom preprocessor graph if needed

    # Required PlaybackNode
    playback_node = ComposableNode(
        name='PlaybackNode',
        namespace=TestCustomGraph.generate_namespace(),
        package='ros2_benchmark',
```

```
from launch_ros.actions import ComposableNodeContainer
from launch_ros.descriptions import ComposableNode

from ros2_benchmark import ImageResolution
from ros2_benchmark import ROS2BenchmarkConfig, ROS2BenchmarkTest

def launch_setup(container_prefix, container_sigterm_timeout):
    """Graph setup for benchmarking your custom graph."""
1   Insert your composable node declarations

    # Required DataLoaderNode

2   Insert your 3 custom preprocessor graph if needed

    # Required PlaybackNode

    # Required MonitorNode

    # Required composable node container

    return [composable_node_container]

def generate_test_description():
    return TestCustomGraph.generate_test_description_with_nsys(launch_setup)

class TestCustomGraph(ROS2BenchmarkTest):
    """Performance test for your custom graph."""

    # Insert custom configurations

    def test_benchmark(self):
        self.run_benchmark()
```

# Create Your Own Benchmark

witht ros2_benchmark

1. Insert your custom graph (e.g., composable nodes) in the launch_setup method.
2. Revise remappings in the data loader node to connect rosbag topics to either your preprocessor graph or a playback node.
3. [optional] Insert your preprocessor graph in the launch_setup method if required.
4. Revise data_formats and remappings in the playback and monitor nodes to connect to the loaded/preprocessed data and your custom graph.

```
# Insert your custom preprocessor graph if needed

# Required PlaybackNode
playback_node = ComposableNode(
    name='PlaybackNode',
    namespace=TestCustomGraph.generate_namespace(),
    package='ros2_benchmark',
    plugin='ros2_benchmark::PlaybackNode',
    # Revise "data_formats" based on your graph
    parameters=[{
        'data_formats': [
            'sensor_msgs/msg/Image',
            'sensor_msgs/msg/CameraInfo'
        ],
    }],
    # Revise "remapping" based on your graph
    remappings=[
        ('buffer/input0', 'data_loader_node/image'),
        ('input0', 'image'),
        ('buffer/input1', 'data_loader_node/camera_info'),
        ('input1', 'camera_info')
    ]
```

4

```
from launch_ros.actions import ComposableNodeContainer
from launch_ros.descriptions import ComposableNode

from ros2_benchmark import ImageResolution
from ros2_benchmark import ROS2BenchmarkConfig, ROS2BenchmarkTest

def launch_setup(container_prefix, container_sigterm_timeout):
    """Graph setup for benchmarking your custom graph."""
    # Insert your composable node declarations

    # Required DataLoaderNode

    # Insert your custom preprocessor graph if needed

    # Required PlaybackNode

    # Required MonitorNode

    # Required composable node container

    return [composable_node_container]

def generate_test_description():
    return TestCustomGraph.generate_test_description_with_nsys(launch_setup)

class TestCustomGraph(ROS2BenchmarkTest):
    """Performance test for your custom graph."""

    # Insert custom configurations

    def test_benchmark(self):
        self.run_benchmark()
```

1 2 3 4

# Create Your Own Benchmark

witht ros2_benchmark

1. Insert your custom graph (e.g., composable nodes) in the launch_setup method.
2. Revise remappings in the data loader node to connect rosbag topics to either your preprocessor graph or a playback node.
3. [optional] Insert your preprocessor graph in the launch_setup method if required.
4. Revise data_formats and remappings in the playback and monitor nodes to connect to the loaded/preprocessed data and your custom graph.
5. Insert your custom nodes declared in step 1 to the composable node container.

```
)

# Required composable node container
# Insert your composable nodes in the "composable_node_descriptions" list.
composable_node_container = ComposableNodeContainer(
    name='container',
    namespace=TestCustomGraph.generate_namespace(),
    package='rclcpp_components',
    executable='component_container_mt',
    prefix=container_prefix,
    sigterm_timeout=container_sigterm_timeout,
    composable_node_descriptions=[
        data_loader_node,
        playback_node,
        monitor_node,
        # Insert custom nodes here
    ],
    output='screen'
)
```

5

```
from launch_ros.actions import ComposableNodeContainer
from launch_ros.descriptions import ComposableNode

from ros2_benchmark import ImageResolution
from ros2_benchmark import ROS2BenchmarkConfig, ROS2BenchmarkTest

def launch_setup(container_prefix, container_sigterm_timeout):
    """Graph setup for benchmarking your custom graph."""
    # Insert your composable node declarations

    # Required DataLoaderNode

    # Insert your custom preprocessor graph if needed

    # Required PlaybackNode

    # Required MonitorNode

    # Required composable node container

    return [composable_node_container]

def generate_test_description():
    return TestCustomGraph.generate_test_description_with_nsys(launch_setup)

class TestCustomGraph(ROS2BenchmarkTest):
    """Performance test for your custom graph."""

    # Insert custom configurations

    def test_benchmark(self):
        self.run_benchmark()
```

1 2 3 4 5

# Create Your Own Benchmark

witht ros2_benchmark

1. Insert your custom graph (e.g., composable nodes) in the launch_setup method.
2. Revise remappings in the data loader node to connect rosbag topics to either your preprocessor graph or a playback node.
3. [optional] Insert your preprocessor graph in the launch_setup method if required.
4. Revise data_formats and remappings in the playback and monitor nodes to connect to the loaded/preprocessed data and your custom graph.
5. Insert your custom nodes declared in step 1 to the composable node container.
6. Revise/add benchmark configurations under ROS2BenchmarkConfig declaration based on your custom graph.

```
    return TestCustomGraph.generate_test_description_with_nsys(launch_setup)

class TestCustomGraph(ROS2BenchmarkTest):
    """Performance test for your custom graph."""

    # Custom configurations
    config = ROS2BenchmarkConfig(
        # Insert your custom benchmark configurations
        benchmark_name='Custom Graph Benchmark',
        input_data_path='datasets/your_custom_rosbag_directory_path',
        publisher_upper_frequency=100.0,
        publisher_lower_frequency=10.0,
        playback_message_buffer_size=10
    )

    def test_benchmark(self):
        self.run_benchmark()
```

6

```
from launch_ros.actions import ComposableNodeContainer
from launch_ros.descriptions import ComposableNode

from ros2_benchmark import ImageResolution
from ros2_benchmark import ROS2BenchmarkConfig, ROS2BenchmarkTest

def launch_setup(container_prefix, container_sigterm_timeout):
    """Graph setup for benchmarking your custom graph."""
    # Insert your composable node declarations

    # Required DataLoaderNode

    # Insert your custom preprocessor graph if needed

    # Required PlaybackNode

    # Required MonitorNode

    # Required composable node container

    return [composable_node_container]

def generate_test_description():
    return TestCustomGraph.generate_test_description_with_nsys(launch_setup)

class TestCustomGraph(ROS2BenchmarkTest):
    """Performance test for your custom graph."""

    # Insert custom configurations

    def test_benchmark(self):
        self.run_benchmark()
```

1 2 3 4 5 6


NVIDIA

# NVIDIA Isaac ROS Survey

Help us get better and win a prize

- Our mission with Isaac ROS is to empower ROS developers with accelerated computing packages and tools needed to develop high performance and power efficient robotics applications.

- To help us continue to improve Isaac ROS and meet the needs of developers like you, we're asking for **your feedback on how you're using Isaac ROS and how we can make it even better in the future**.

- All entries will be entered into a **raffle** for a chance for three winners to receive an NVIDIA Jetson AGX Orin Developer Kit or an NVIDIA Jetson Orin Nano Developer Kit.

- Please share your feedback here in this 1-minute survey >

**https://developer.nvidia.com/isaac-ros/survey**