

1. For the text below, summarise the problem and suggest a software system architecture solution. The main stakeholders in this case are: - Software system architects - DevOps engineers - Developers

Problem, context, and related work

Robot Operating System 2 (ROS) is a popular and widely adopted robotics engineering framework. Its main goal is to provide robotics engineers a platform that allows for the development of robotics applications. Framework's infrastructure, which is open source, is based on numerous distributed packages. ROS supports a wide range of devices, starting from small embedded devices up to autonomous industry robots. ROS was developed with real-time capabilities in mind. Since this technology is in its infancy, the ROS community encourages industry and academic research. Academic research in ROS has covered many aspects of robotics development and has achieved quantifiable results regarding soft and hard real-time constraints, lowering latency, and in general, improving performance. However, because of its complicated vertical stack of technologies, there is still room where significant performance benefits could be extracted. The default way to extract performance with aim to lower execution time is by utilizing multi-cores. However, ROS with its complex infrastructure and requirement from robotics engineers to learn parallelization from C/C++ perspective, make it hard to utilize multi-cores properly. Therefore, utilizing multi-cores to lower execution time is one of the main motivators that guides research in ROS. By lowering execution time we gain performance benefits and allow other parts of ROS infrastructure, which are influenced by latency and real-time constraints, to be positively impacted as well. Robots are very heterogeneous devices. Therefore, robots may come with a vast array of different system requirements and system constraints. Some of the system constraints are critical when it comes to the correct functionality of the robots. These critical constraints deal with the overall performance of the robotics application. In the scope of overall performance of the robotics application, the main goal of robotics engineers is to lower execution time or make a particular function execute faster. To facilitate this, we refer to utilization of multi-core processors. However, required specialized knowledge regarding concurrency and expertise, which robotics engineers do not have, slows them down in building their robotics application and decreases efficiency. Robotics engineers are required to understand the architecture of the CPU and how it interacts with main memory. On top of that robotics engineers are required to understand the intricacies regarding caches and data locality. Complemented with this is a software stack that is utilizing the CPU. They need to learn how to use multi-threading libraries, which is not simple. Functions for creating and managing threads require thinking about multiple data-flows in software stack. As a consequence, robotics engineers are required to understand in low-level detail how multi-threading works and how prioritization, scheduling, and affinity combine to solve the problem of lowering execution time. To conclude, the main gap is that robotics engineers do not have an intuitive way of utilizing multi-cores to be able to lower execution time. They are required to know low-level details regarding parallelization and have to develop multi-threading applications. Additionally, robotics engineers do not have accessible interfaces to control Operating System mechanisms regarding prioritization, scheduler type, and affinity

Requirements

- SR.01: Framework to facilitate development of multithreaded applications in ROS2
- SR.02: Framework to assist robotics engineers in developing software that takes advantage of multi-cores in order to reduce average execution time, integrated in ROS2.
- SR.02: Pre-locking heap memory - The framework enables pre-locking of heap memory during initialization of the application in order to avoid failed memory allocations, which can occur if application requires significant amount of memory.
- SR.03: Process priority configuration - The framework provides interface for setting priorities of threads in order to avoid priority inversion.
- SR.04: Measurement of heap allocations - The framework offers a

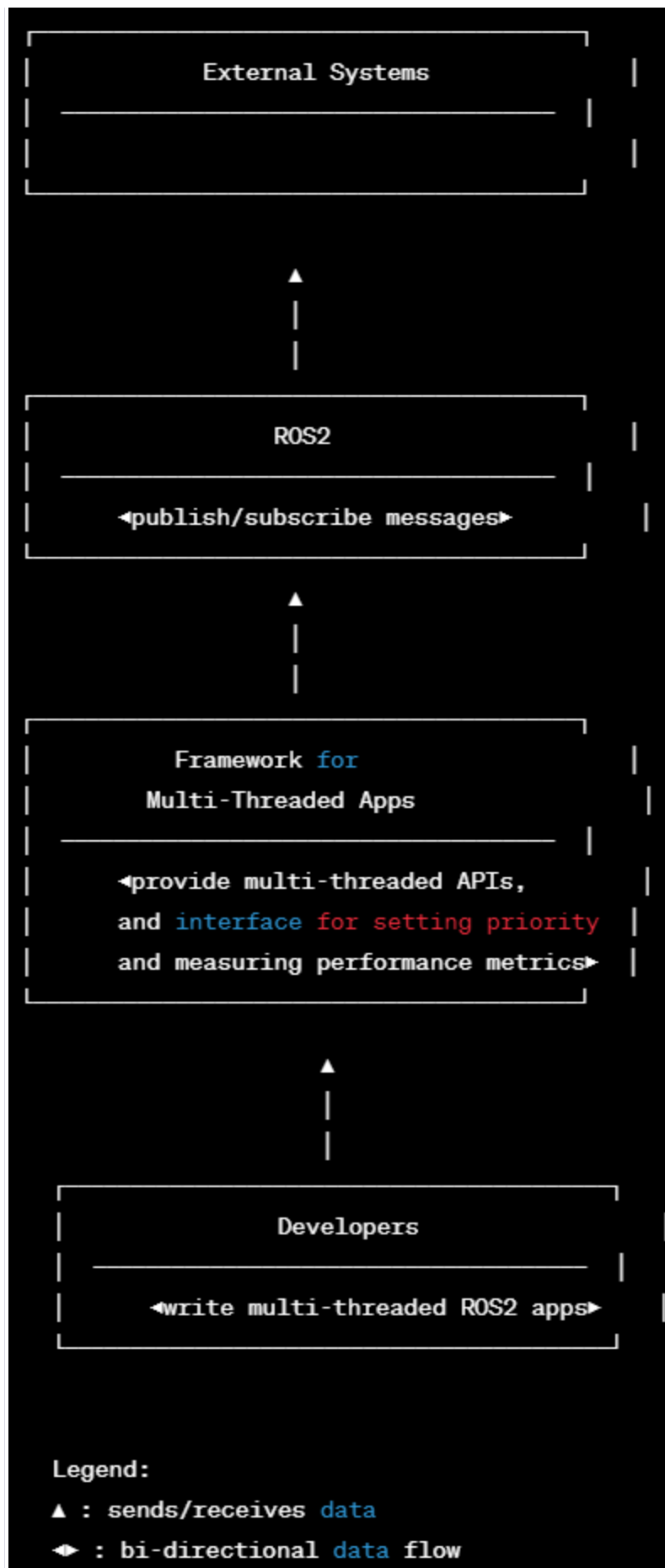
visualization on how many memory allocations happened, and it should be easily accessible to the robotics engineers • SR.05: Measurement of latency. - The framework has a mechanism that measures latency • SR.06: Measurement of jitter - The framework has a mechanism that measures jitter • SR.07: Measurement of CPU caches usage & The framework has a mechanism that measures CPU cache usage • SR.08: Measurement of memory fetches. - The framework has a mechanism which measures memory fetches. • SR.09: Executor type - Robotics engineers write nodes' configuration as input to the framework. The framework outputs the best performing executor type configuration containing nodes. & ROS executor • SR.10: Node to process assignment - Robotics engineers write nodes' configuration as input to the framework. The framework outputs the best performing configuration. & node assignment, thread, process • SR.11: Allocator type - The framework accepts the message data types that are used throughout the application and it advises robotics engineers on which allocators to use. • SR.12: Easy to use user interface - All information and capabilities of the framework should be gathered in easy-to-use user interface. • SR.13: Guideline - Robotics engineers will be able to express their intentions of parallelizing the code in the framework. The framework will output guidelines for robotics engineers. • SR.14: Parallel Configuration - The framework will accept non parallelized code and will turn it into parallel configuration. & Parallel Configuration Quality requirements • SR.01: Increased CPU Utilization - The framework facilitates the development of multi-threaded applications, and as a result, increased CPU utilization. If the number of cores are n , (and $n > 1$), then utilization is at least $0.7 * n$. • SR.02: Decreased Heap Allocation - The framework facilitates the mechanism with pre-locking memory during the start of the execution of the application. At least 50 percent decrease in heap allocations regarding memory is expected. • SR.03: Increased general performance of the application - The framework facilitates the mechanism for process priority configuration. We expect an increase in general performance in a region of 10 percent. • SR.04: Decreased average execution time - The framework facilitates the mechanism for decreasing average execution time by utilizing multi-threading. • SR.05: Increased CPU Cache utilization - We expect an increase in CPU Cache utilization by 30 percent. • SR.06: Decreased Memory Fetches - We expect to have a decreased memory fetches by 20 percent as a result of increased CPU Cache utilization. • SR.07: Increased productivity of robotics engineers - The facilities of the framework will output parallelized configurations. Compared with by-hand optimization, this process will be expected to take 30 percent less time. & increased productivity • SR.08: Usability - The framework will include an easy to use user interface. We expect robotics engineers to quickly understand the framework and use it efficiently. • SR.09: Modularity - The framework's design will closely follow ROS design philosophy. We expect the framework to be at least modular as ROS and to fit on top of ROS. & modularity Use cases • UC.01: Direct Inverse Kinematics - Sensors, Actuators, Moveable joints, Rotating Joints & Inverse kinematics is a field which concerns itself with correct positioning of the joints in order for end-actuator to reach desired position. As the name indicates, the positioning is worked out from the end-actuator backwards to the first-actuator. • UC.02: Detecting human faces and torso orientation - People in a scene, robot scanning the scene - Image processing is a very important field in robotics engineering. This use case tries to identify the way of doing image processing in robotics, so we can learn how to parallelize common functions. • UC.03: Kalman Filter & A specific robot does sensor fusion and uses kalman filter for prediction - Prediction of the measurements. Kalman filtering is a mathematical approach on how to improve our measures that come from the sensors. • UC.04: Assessing execution time of real-time loop - Real-Time Loops, Function, Time & Since ROS was built with real-time capabilities in mind, measuring real-time loop execution time can give us an insight on performance improvements.

The problem identified is that robotics engineers face challenges in utilizing multi-core processors to reduce execution time in the Robot Operating System 2 (ROS2) framework. This is

due to the need for specialized knowledge regarding concurrency, low-level details of parallelization, and multi-threading applications. The solution proposed is a framework that facilitates the development of multi-threaded applications, with features such as pre-locking heap memory, process priority configuration, measurement of heap allocations, latency, jitter, CPU cache usage, and memory fetches. The solution also includes a user-friendly interface and output guidelines for parallelizing code. The proposed solution aims to increase CPU utilization, decrease heap allocation, improve general performance, decrease average execution time, increase CPU cache utilization and productivity, and improve usability and modularity. Use cases for the proposed solution include direct inverse kinematics, image processing, sensor fusion using the Kalman filter, and assessing real-time loop execution time in ROS2.

2. Suggest architecture of a conceptual solution for this problem. We need a C4 system context diagram (<https://c4model.com/#SystemContextDiagram>). Please use text (arrows, dots, different shapes, and different colours) to visualise the solution. Consider the following: - Identify relevant external systems (if any) and add them to the diagram. - Identify relevant external stakeholders (if any) and add them to the diagram. - Identify high level abstract data types transferred between components and stakeholders. - Summarise data types transmitted between components. Add this information to the connections and interfaces between components.

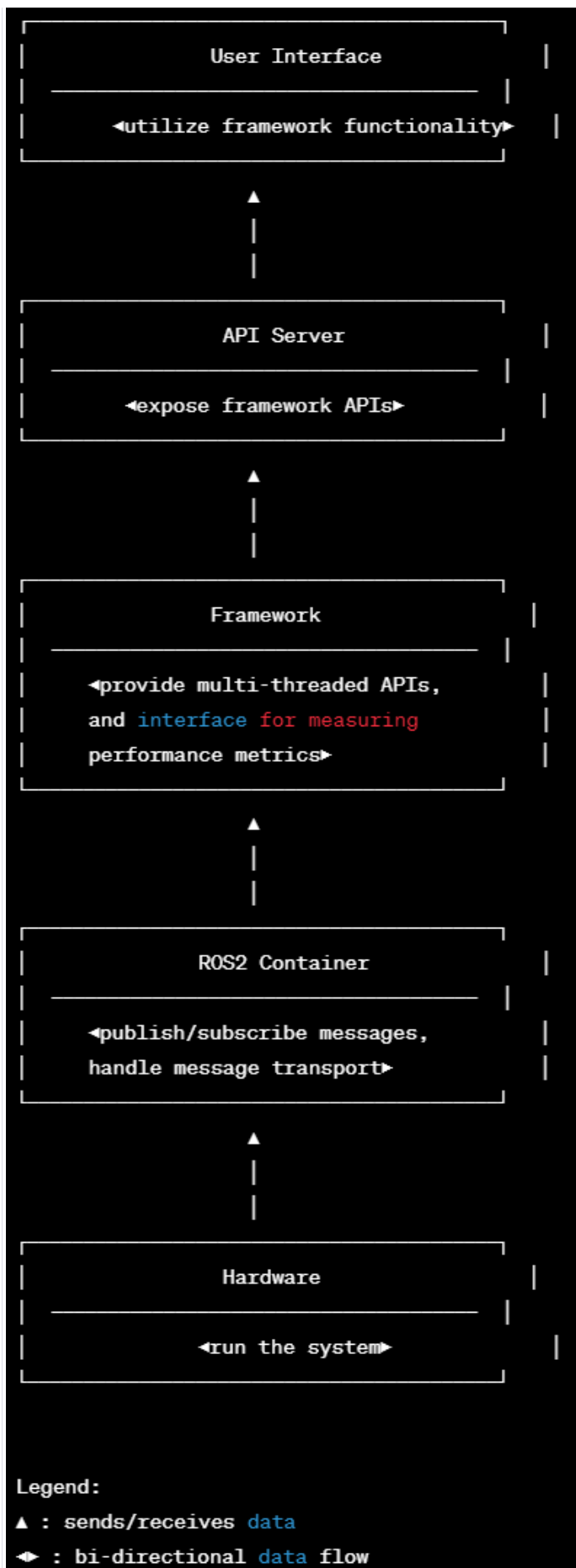
The following is a proposed architecture for the solution to the problem of facilitating the utilization of multi-core processors in the ROS2 framework:



In this conceptual solution, there are no external stakeholders involved. However, there are external systems that interact with the ROS2 framework. The proposed solution introduces a framework for multi-threaded applications that sits on top of the ROS2 framework. The framework provides multi-threaded APIs, interfaces for setting thread priority, and measuring performance metrics. Developers write multi-threaded ROS2 applications that utilize the framework to take advantage of multi-core processors. The high-level abstract data types transferred between components include messages, ROS2 topics, and API calls. The connections between components involve the transmission of messages, topics, and API calls. The interfaces between components involve setting thread priority, measuring performance metrics, and utilizing multi-threaded APIs.

3. Suggest architectural components that we need to solve this problem. We need a C4 container solution diagram (<https://c4model.com/#ContainerDiagram>). Please use text (arrows, dots, different shapes, and different colours) to visualise the solution. Consider the following: - Define how the main stakeholder will use the solution. Add it to the diagram.. - Which quality properties are important? Suggest adequate architectural solutions to address them (architectural style, design patterns, design tactics, components and their relationships). Structure the components according to it. - Define data types transferred between components. - Summarise data types transmitted between components. Add this information to the connections and interfaces between components. - Define interfaces between components. Add information about required and provided interfaces to the components, along with the data types. - Document which anti-patterns we need to avoid when implementing this solution.

The following is a proposed C4 container diagram for the solution to the problem of facilitating the utilization of multi-core processors in the ROS2 framework:



The proposed solution involves a user interface, an API server, a framework for multi-threaded applications, a ROS2 container, and hardware to run the system. The user interface allows the software system architects, DevOps engineers, and developers to interact with the solution, utilizing the functionality provided by the framework. The API server exposes the framework's APIs to the developers. The framework provides multi-threaded APIs, interfaces for measuring performance metrics, and other features to enable the utilization of multi-core processors. The ROS2 container is responsible for message transport and publishes/subscribes to messages. The hardware runs the system.

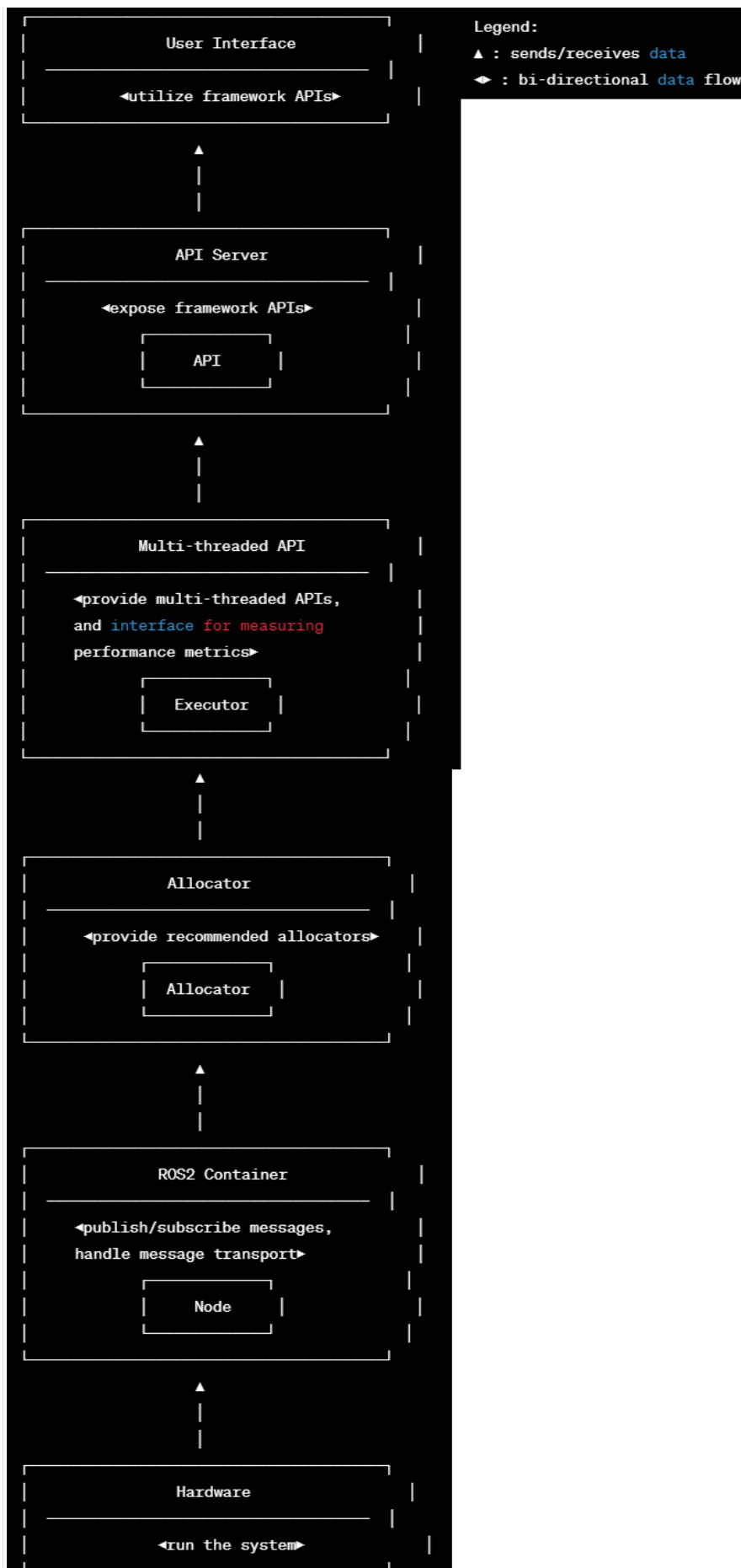
The important quality properties for this solution include increased CPU utilization, decreased heap allocation, improved general performance, decreased average execution time, increased CPU cache utilization and productivity, and improved usability and modularity. To address these quality properties, the following architectural solutions can be implemented:

- Architectural Style: Microservices architecture style can be used to ensure modularity, scalability, and maintainability of the system.
- Design Patterns: Singleton pattern can be used for the framework and API server components to ensure only one instance is running, preventing unnecessary overhead and resources.
- Design Tactics: Caching can be used to increase CPU cache utilization and reduce memory fetches. Pre-locking memory can be used to reduce heap allocation.
- Components and Their Relationships: The API server communicates with the framework to expose its APIs to the developers. The framework communicates with the ROS2 container to handle message transport. The user interface utilizes the framework's functionality.
- Data Types: The data types include messages, topics, APIs, and performance metrics.
- Connections and Interfaces: The connections involve message transport between the ROS2 container and framework, API calls between the API

4. Suggest architectural components that we need to solve this problem. We need a C4 component solution diagram (<https://c4model.com/#ComponentDiagram>). Please use text (arrows, dots, different shapes, and different colours) to visualise the solution. Consider following:

- Define how the main stakeholder will use the solution. Add it to the diagram.
- Which quality properties are important? Suggest adequate architectural solutions to address them (architectural style, design patterns, design tactics, components and their relationships). Structure the components according to it.
- Define data types transferred between components.
- Summarise data types transmitted between components. Add this information to the connections and interfaces between components.
- Define interfaces between components. Add information about required and provided interfaces to the components, along with the data types.
- Document which anti-patterns we need to avoid when implementing this solution.

The following is a proposed C4 component diagram for the solution to the problem of facilitating the utilization of multi-core processors in the ROS2 framework:



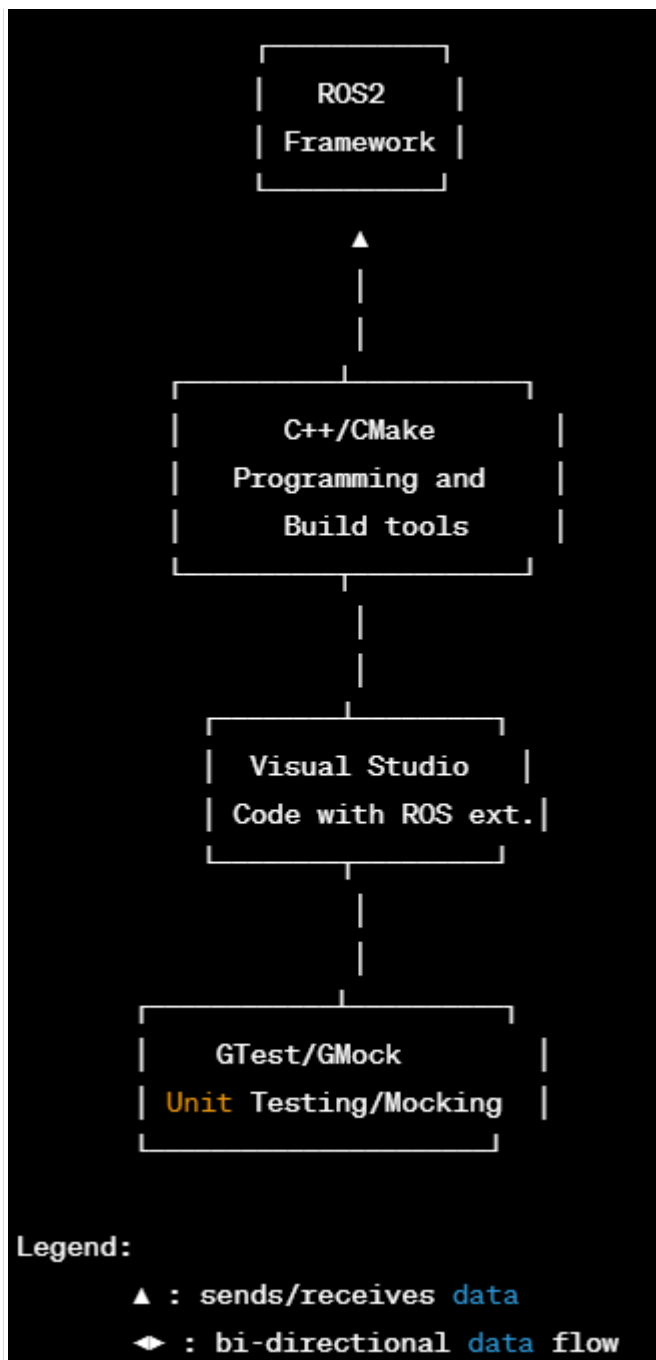
The proposed solution involves a user interface, an API server, a multi-threaded API component, an allocator component, a ROS2 container, and hardware to run the system. The user interface allows the software system architects, DevOps engineers, and developers to interact with the solution, utilizing the functionality provided by the multi-threaded API and allocator components. The API server exposes the multi-threaded and allocator APIs to the developers. The multi-threaded API provides multi-threaded APIs, interfaces for measuring performance metrics, and other features to enable the utilization of multi-core processors. The allocator component provides recommended allocators based on the message data types used

5. Please suggest a toolchain to implement the solution. Consider which tools we can reuse. For the components we need to develop, please suggest a programming language. Please use text (arrows and dots) to visualise the solution. Define data types transferred between components. Summarise data types transmitted between components. Add this information to the connections and interfaces between components.

For the proposed solution, we can use the following toolchain:

- ROS2 framework: ROS2 is the base framework used for the proposed solution.
- C++ programming language: C++ is the programming language used for developing the proposed components, as it is a commonly used language in the ROS2 community and is well-suited for the performance-sensitive nature of the problem.
- CMake: CMake is used for building the proposed components, as it is the standard build system used in ROS2.
- Visual Studio Code: Visual Studio Code is used as an integrated development environment (IDE) for developing the proposed components, as it has a ROS extension that provides helpful features for developing ROS2 applications.
- GTest/GMock: GTest and GMock are used for unit testing and mocking in the proposed solution.

The following is a visualisation of the proposed toolchain:



For data types transferred between components, the proposed solution involves message data types passed between ROS2 nodes, as well as data structures passed between the multi-threaded API and the allocator component. The multi-threaded API component may also pass performance metrics data types to the user interface component. The proposed solution uses the standard ROS2 message data types, such as `std_msgs::String`, `std_msgs::Int32`, etc.

In summary, the proposed solution involves the use of ROS2 framework, C++ programming language with CMake for programming and building, Visual Studio Code with ROS extension for development, and GTest/GMock for unit testing and mocking. Data types transferred between components include message data types passed between ROS2 nodes and data structures passed between the multi-threaded API and the allocator component. Performance metrics data

types may also be passed from the multi-threaded API component to the user interface component.