# ROS2: A Technical Overview

Robot Operating System 2 Architecture & Implementation

Architecture

**Distributed**

Security

**Built-in**

Performance

**Real-time**

Applications

**Multi-domain**

# Introduction & History of ROS

## 2006

### Origins

**Stanford AI Lab** project by Wyrobek & Berger

## 2006-07

### Core Goal

Eliminate **"reinventing wheel"** in robotics

## 2007-14

### Willow Garage Era

**PR2 robot** development • ROS maturation

## ROS1

### Limitations

**Single master** • No **real-time** • **Security** gaps

## 2014

### Transition to Open Robotics

After Willow Garage closure → **OSRF** took over development


*PR2 Robot - Key milestone in ROS development at Willow Garage*

# ROS2 Distributions

ROS2 distributions follow a **timed release schedule** with varying support periods

## Humble LTS

| | |
|---|---|
| Humble Hawksbill | May 2022 - May 2027 |
| Support Period | 5 years |

**Production Ready**

## Standard Releases

| | |
|---|---|
| Foxy Fitzroy | May 2020 - May 2023 |
| Galactic Geochelone | May 2021 - Nov 2022 |
| Iron Irwini | May 2023 - Nov 2024 |

**1.5-3 years support**

## Rolling

| | |
|---|---|
| Always Latest | Continuous updates |
| Development | Cutting-edge features |
| Testing | Not production-ready |

**Latest Features**

LTS (Long-Term Support) Standard Release Rolling Release

# Motivation for ROS2

### Real-time Performance

**Deterministic behavior** for industrial applications

### Multi-robot Systems

**Master-based** architecture limited scalability

### Security

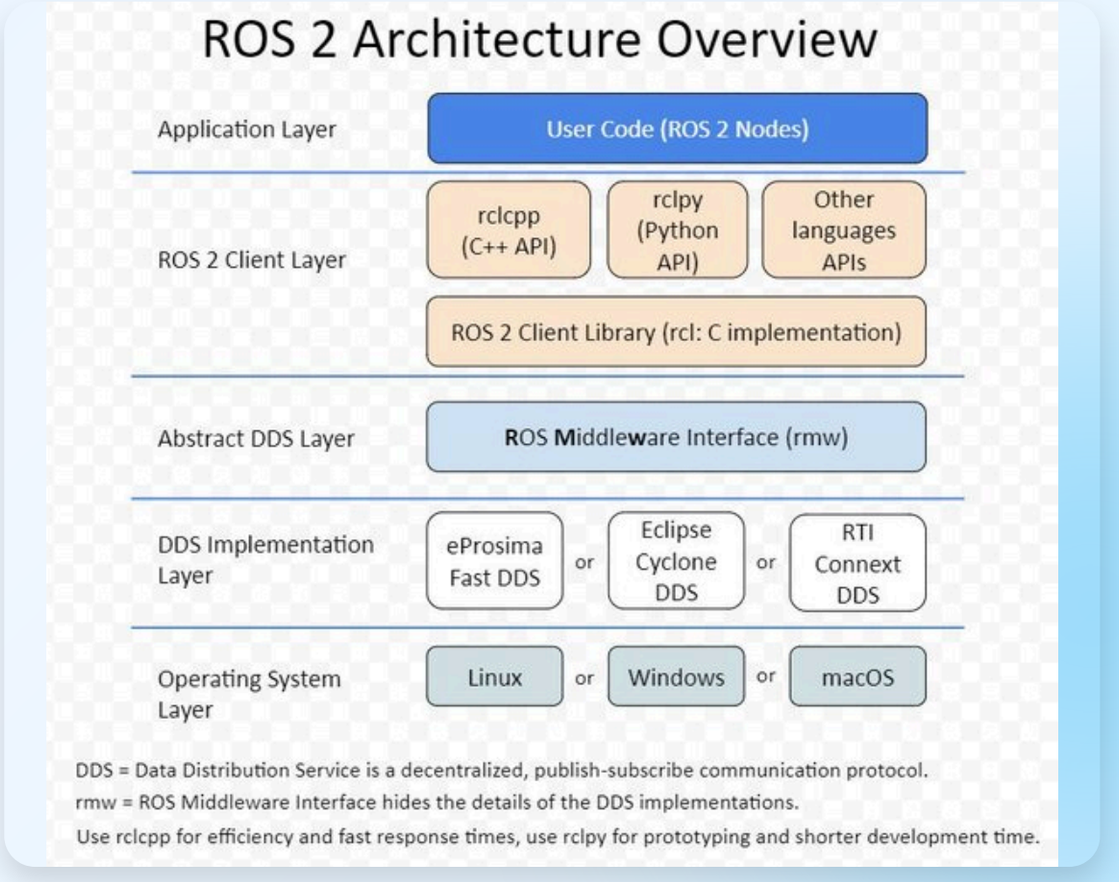
ROS1 lacked **built-in security** for production

### Reliability

Need for **fault-tolerant** communication

### DDS-based Communication

Adoption of **industry-standard** middleware




*ROS2 Architecture with DDS Middleware Layer*

# Why ROS2? (Key Advantages)

### Distributed Architecture

No single point of failure

**Decentralized**

### Modularity

Component isolation

**Independent**

### Real-time Capability

Deterministic communication

**Configurable QoS**

### Security Features

Authentication built-in

**Encrypted**

### Scalability

Large-scale systems

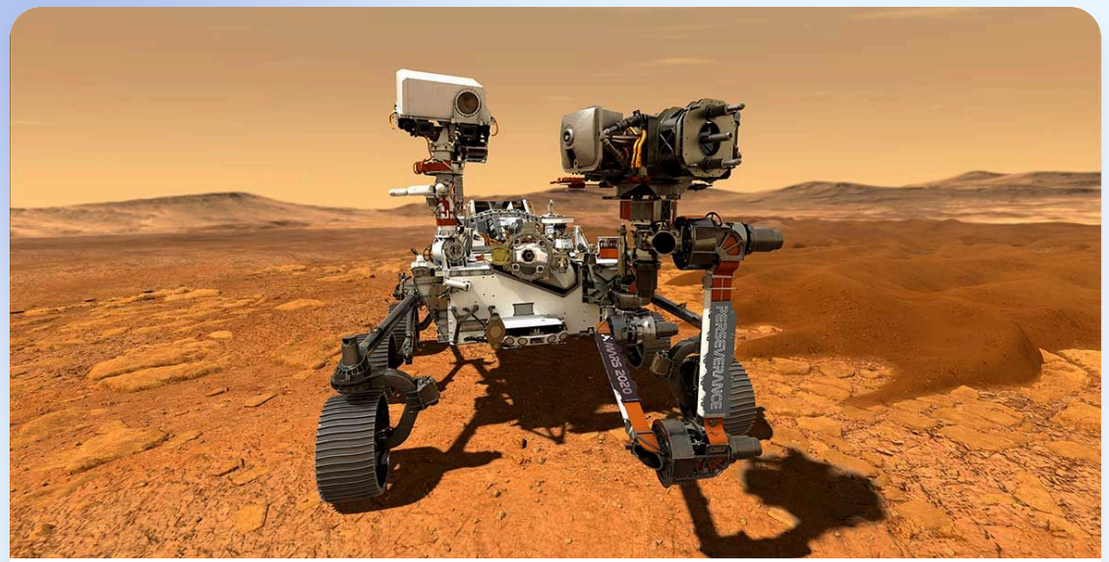
**Hundreds of nodes**

### Industrial-grade Middleware

DDS reliability

**High performance**

# Applications & Domains - Part 1




## Space Robotics

- **Space ROS** : NASA-certified
- RASSOR lunar excavator
- Martian rover navigation

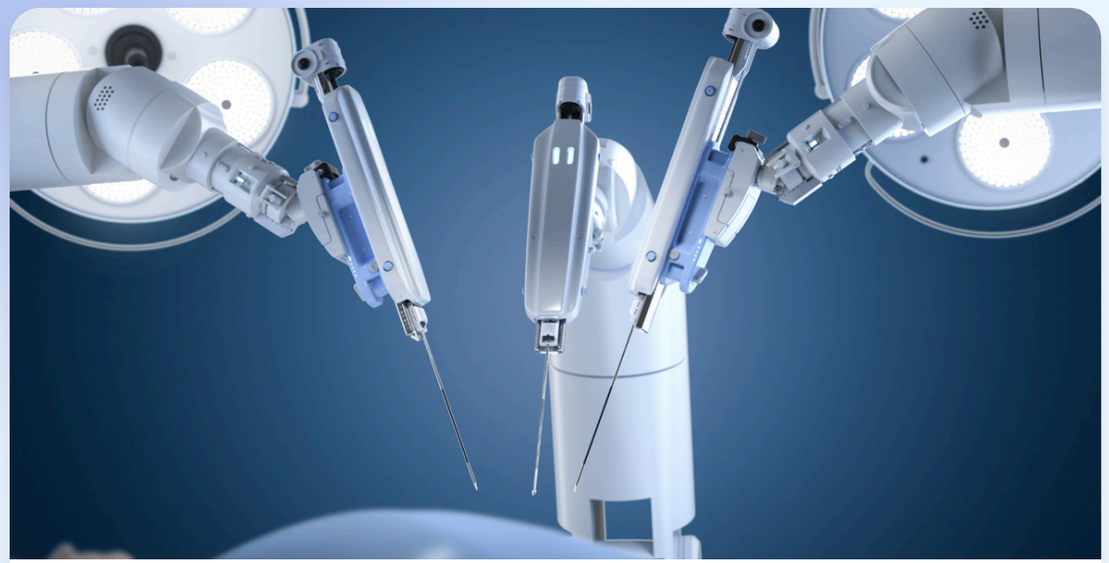
**Extreme environments**

## Military & Defense

- **ROS-M** : Military ecosystem
- Autonomous ground vehicles
- Surveillance & reconnaissance

**Mission-critical systems**

# Applications & Domains - Part 2



## Medical Robotics

- **ROS-MED**: Medical framework
- Surgical assistance systems
- Rehabilitation devices

**Precision medicine**

## Industrial Automation

- **Collaborative robots** (cobots)
- Manufacturing assembly lines
- Quality inspection systems

**Production efficiency**

# Structural Concepts

## Workspace

- Directory containing **ROS2 packages**
- Built with **colcon** build tool

src/ build/ install/ log/

### Organized development

```
# Create a workspace
mkdir -p ~/ros2_ws/src
cd ~/ros2_ws
colcon build
```

## Package

- **Basic unit** of ROS2 software
- Contains: **nodes**, libraries, launch files
- Metadata in **package.xml**
- Build instructions in **CMakeLists.txt**

### Modular development

```
# Create a package
ros2 pkg create --build-type ament_python
my_package
```

# Core Communication Concepts

## Node

- **Independent** process
- Modular, testable

**Fault-isolated**

## Topic

- **Pub/Sub** pattern
- **Asynchronous** flow

**One-to-many messaging**

## Service

- **Request-response**
- **Synchronous** operation

**Client-server model**

## Action

- **Long-running** tasks
- **Preemptable** (cancellable)

**Provides feedback**

# Node Concept in ROS2

## Node

- **Independent** process in ROS2 graph
- Modular, testable, **fault-isolated**

Unconfigured | Inactive | Active | Finalized

## Node Example

```python
import rclpy
from rclpy.node import Node

class MinimalNode(Node):
    def __init__(self):
        super().__init__('minimal_node')
        # Node initialization code

def main(args=None):
    rclpy.init(args=args)
    minimal_node = MinimalNode()
    rclpy.spin(minimal_node)
    minimal_node.destroy_node()
    rclpy.shutdown()
```

# Topic Concept in ROS2

## Topic

- **Pub/Sub** communication pattern
- **Asynchronous** data flow
- **One-to-many** messaging


**Loose coupling**

## Publisher & Subscriber Example

```python
# Publisher
self.publisher = self.create_publisher(String,
'topic', 10)
timer_period = 0.5
self.timer = self.create_timer(timer_period,
self.timer_callback)

# Subscriber
self.subscription = self.create_subscription(
    String, 'topic', self.listener_callback, 10)

# Callback functions
def timer_callback(self):
    msg = String()
    msg.data = 'Hello World'
    self.publisher.publish(msg)

def listener_callback(self, msg):
    self.get_logger().info('I heard: "%s"' %
msg.data)
```

# Launch Files

## Purpose

**Orchestrate** and coordinate multiple nodes

**System coordination**

## Features

- **Parameter** passing
- **Remapping** topics/services
- **Configuration** management

**Flexible deployment**

Python Example | XML Example

```python
# Create a launch description
from launch import LaunchDescription
from launch_ros.actions import Node

def generate_launch_description():
    return LaunchDescription([
        Node(
            package='my_package',
            executable='talker',
            name='my_talker',
            output='screen',
            parameters=[{'use_sim_time': True}]
        ),
        Node(
            package='my_package',
            executable='listener',
            name='my_listener',
            output='screen'
        )
    ])
```