Autonomous Vehicles with ROS2 using seL4

As part of the effort of

Assured Robotics Embedded Systems and
Applications in Ground Vehicles (ARSENAL)

Dave Lide

Trusted Science and Technology, Inc. (Trusted ST)

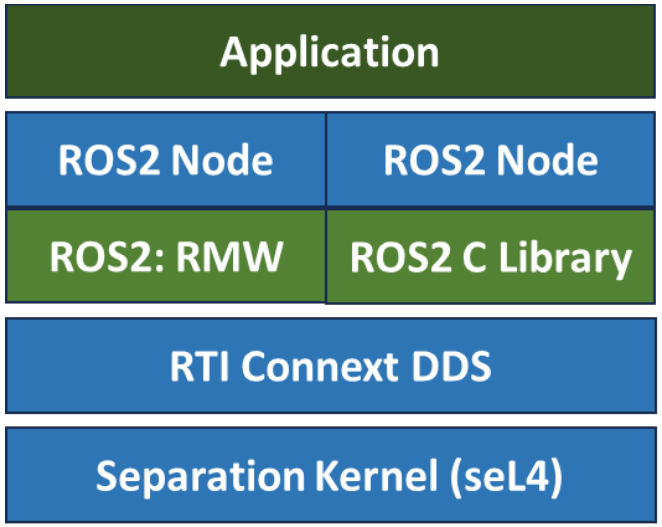
seL4 Summit, September, 2026

Background – ROS2

- ROS 2 (Robot Operating System 2) is an open-source, flexible middleware framework and software development kit (SDK) used to build advanced robotics applications
- Supported on Linux Operating System
- Provides:
  - Middleware to facilitate communication between robotic components (Nodes)
  - The pub/sub paradigm (topics), also client/server and actions
  - DDS (Data Distribution Service)
  - Many application components and examples that can be reused

General Vision

- Running Select ROS2 applications natively on seL4 w/ CAmkES
- ROS2 C library implements ROS functionality for the instantiated “ROS2 Nodes”
- ROS2 Nodes: ROS2 application built using ROS2 build system and then linked in during seL4 component build
- RMS (ROS middleware) allows a ROS2 node to communicate with the other application through the network interface
- The middleware communication transport is provided by a Data Distribution Service (DDS)
- We use RTI Connext DDS

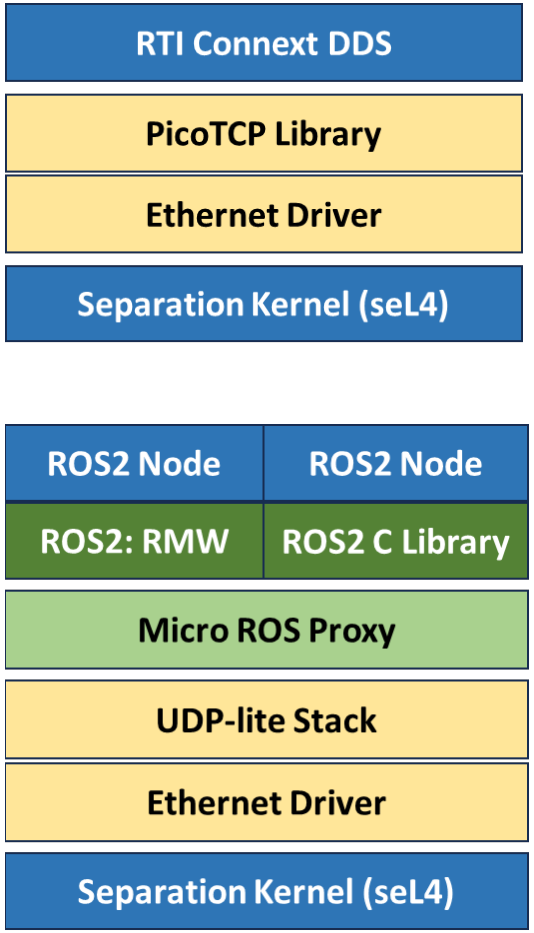


Split Approach

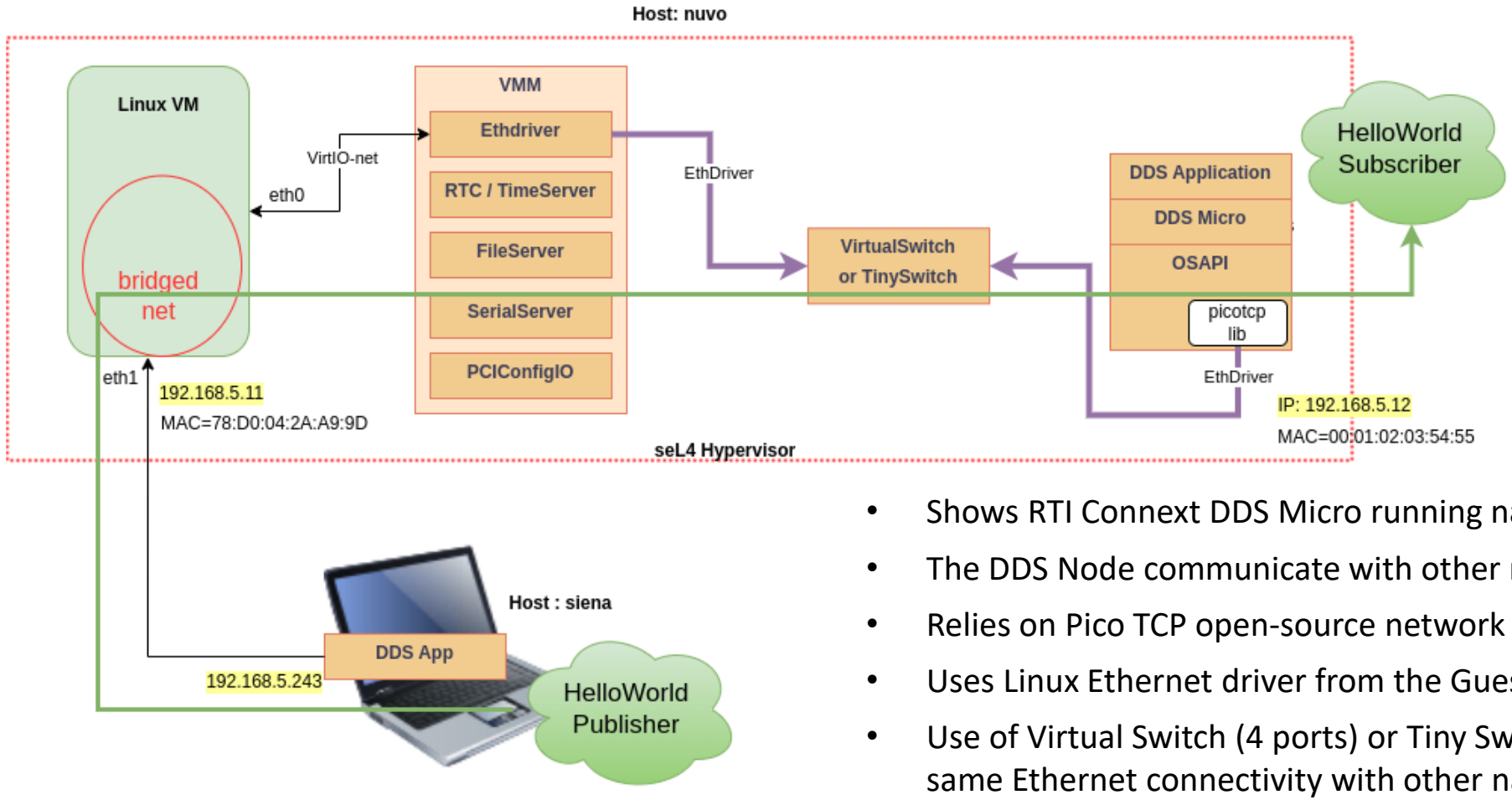
- It is quite challenging and time-consuming to make RTI Connext DDS function properly in an seL4 environment.
- The same is true to correctly port ROS2 (middleware and library) and run ROS2 nodes, which also depend on a working DDS such as the RTI Connext DDS.
- The team decided to ‘break’ the above two efforts into two Phases.
  - Phase 1: loosely-coupled development paths
  - Phase 2: integrate the outputs from the two development paths

Two Development Paths

- *Bottom-up:* porting the RTI Connext DDS to seL4, which basically starts with seL4 and involves all necessary drivers and resources to make the RTI Connext DDS fully functional.
- *Top-down:* making ROS2 work properly, which focuses on ROS2 porting “down” to connect with a proxy DDS (called Micro ROS proxy).



Bottom-up Prototype Architecture



- Shows RTI Connext DDS Micro running natively on seL4 w/CAmkES
- The DDS Node communicate with other nodes outside the test machine
- Relies on Pico TCP open-source network stack
- Uses Linux Ethernet driver from the Guest OS running on top of seL4
- Use of Virtual Switch (4 ports) or Tiny Switch (2 ports) allows sharing the same Ethernet connectivity with other native seL4 components
- Nuvo machine DDS application subscribes to *HelloWorld* topic
- An external PC ('siena') publishes sample data on the same topic

Bottom-up: RTI DDS on seL4

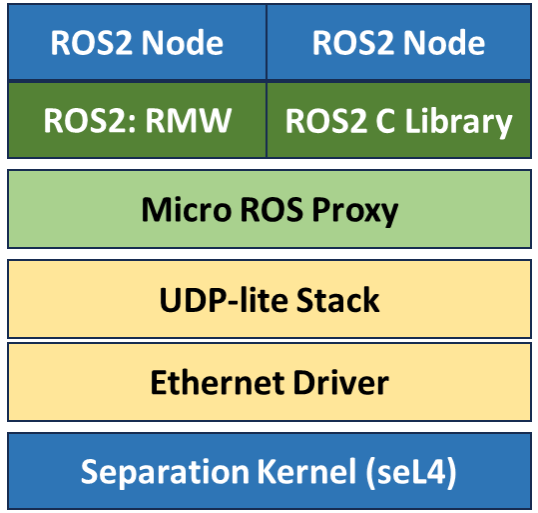
- Utilized seL4/CAmkES on NUVO industrial PC
- Created a CAmkES component that contained:
  - RTI Connext DDS layer
  - Sample DDS application (Subscriber)
  - Pico-TCP stack
  - VETH virtual ethernet driver (CAmkES interface)
- RTI Connext DDS layer
- Sample DDS application (Subscriber)
- Pico-TCP stack
- VETH virtual ethernet driver: allows RTI DDS component to communicate with external Linux PC also running the same RTI/DDS example

Bottom-up: Challenges and Solutions

Challenges	Solutions
Logging	Utilize standard seL4 print services, connecting to the SerialServer CAmkES component.
Timers: ability to trigger a particular function at a particular time (single shot or periodic)	Utilize CAmkES Time Server to provide a 10 msc ‘tick’ Notification Event.
Threads: ability to create arbitrary threads	Use CAmkES to provide a set of Notification elements in the assembly. CAmkES will automatically create a corresponding thread for each element. These can then be used as a “pool” of threads for DDS stack to use.
Synchronization primitives: Semaphores, Mutex, binary semaphores	A pool Semaphores was defined in the assembly; this allowed CAmkES to create these objects for our purposes.
Network stack (specifically access to the UDP layer)	We decided to integrate picotcp library directly in the component. Having an external TCP/IP Server component running picotcp had performance issues.

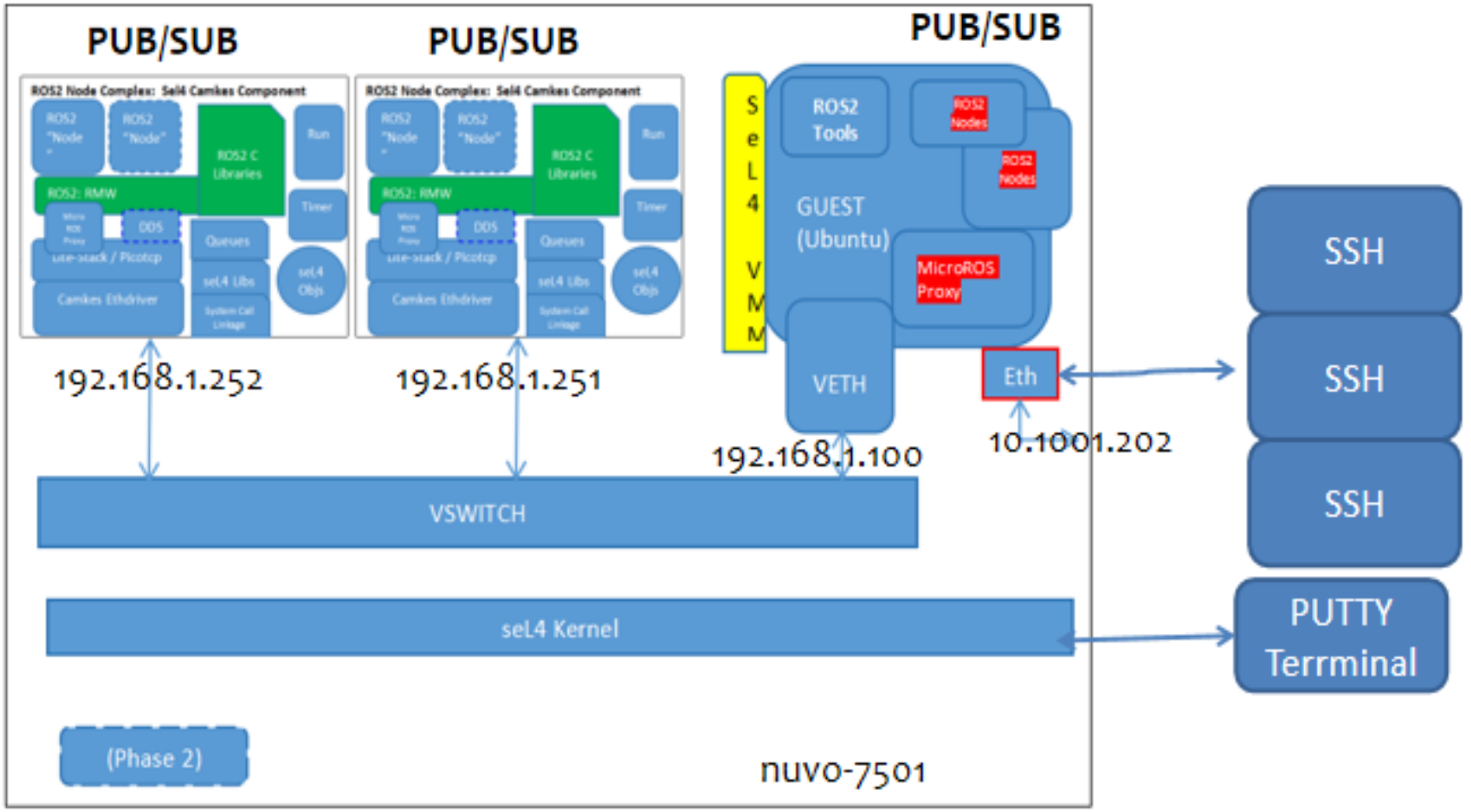
Top-down Development Path

- The goal of the Top-down effort was on getting a ROS2 application and necessary libraries integrated into a seL4/Camkes component.
- We used the ROS2 micro-ros *proxy* approach with **micro-ros-proxy** app on guest providing full DDS functionality, and a simpler protocol used between ROS2 Node Complexes and proxy application.
- Another goal was to show that these ROS2 applications running natively in seL4 could still interact with the ROS2 environment on the Guest.



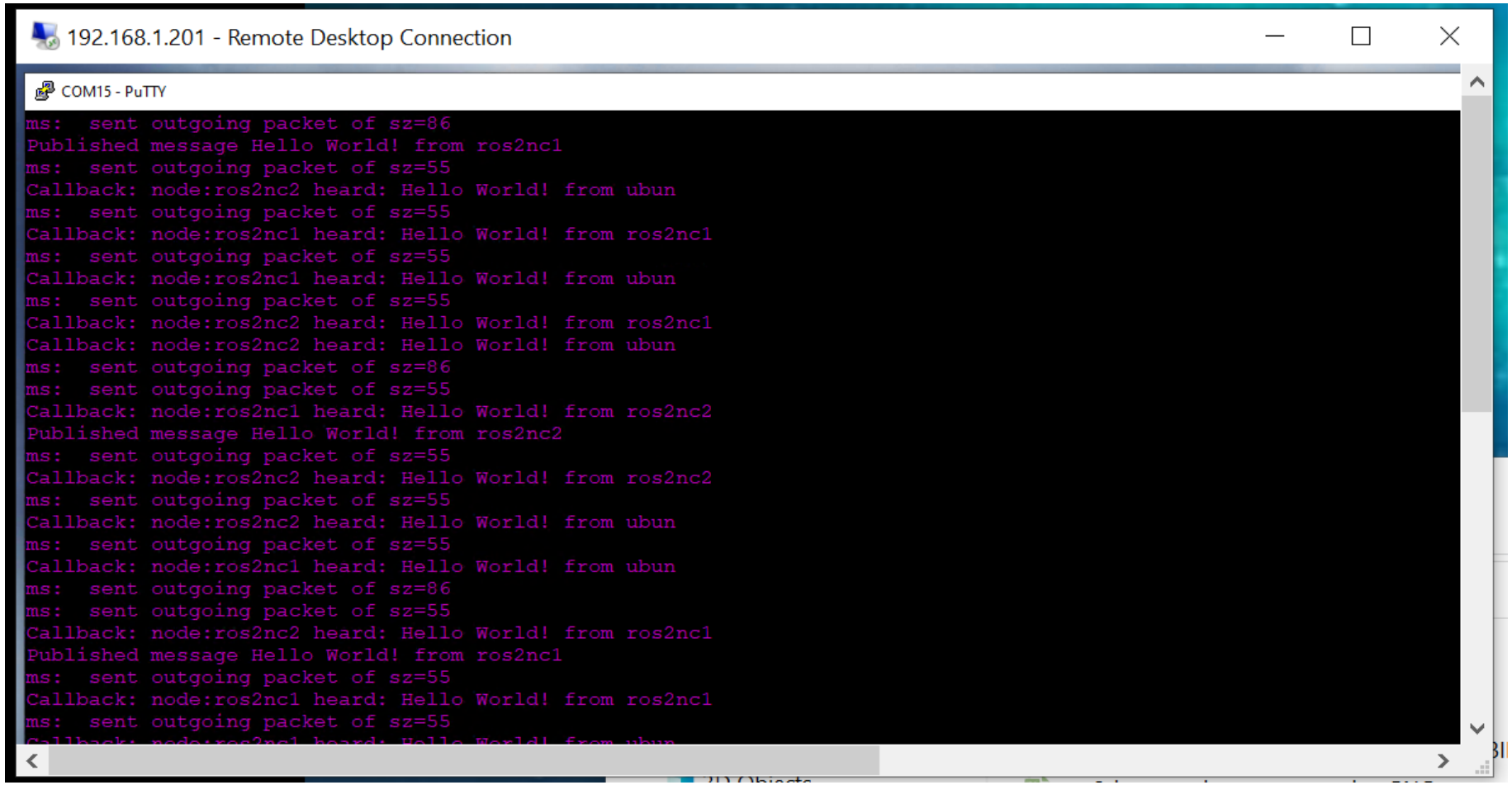
Top-down Development Architecture

ROS2
PUB/SUB
Example



Top-down Results: ROS2NC

Screen Shot: ROS2NC nodes in action (seL4 debug output). Print when topic published and received. (Node 'ubun' is running on guest, next slide)



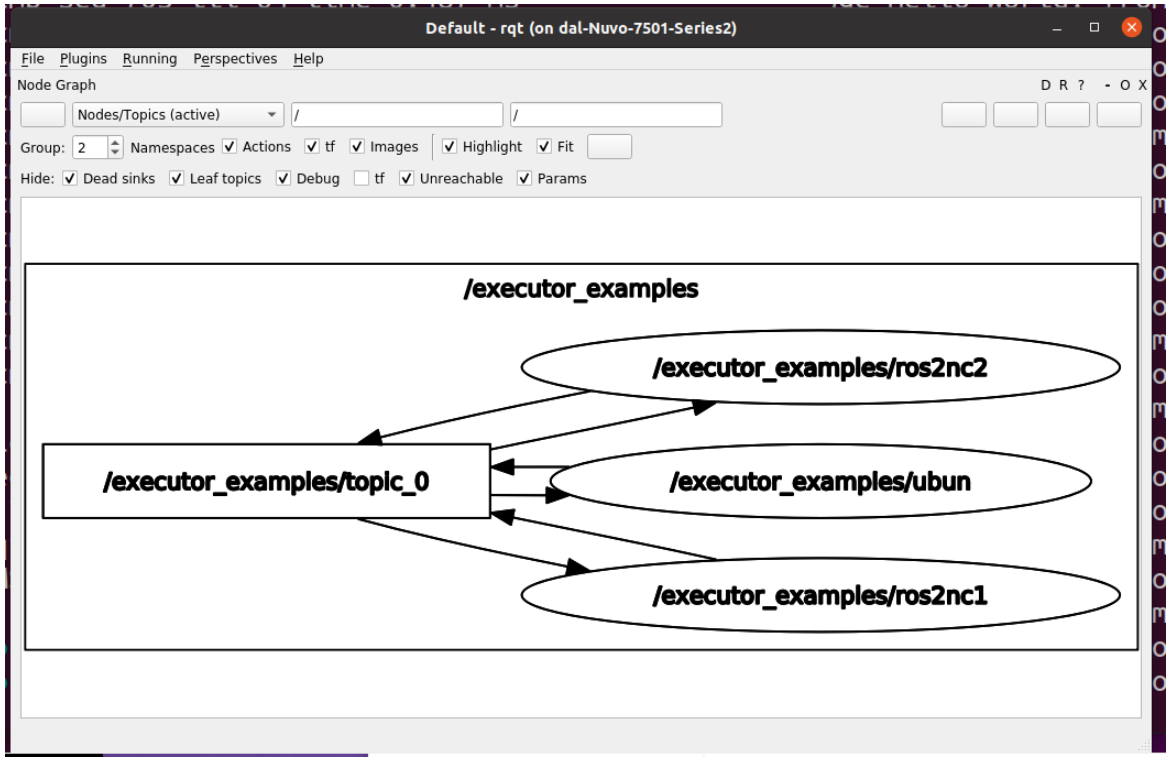
Top-down Results: ROS2 Example on Guest

Screen Shot: ROS2 Example on Guest (same application as on ROS2NCx, node name = “ubun”)

```
rclpub: after rmw_create_pub
Created timer with timeout 1000 ms.
ct 1
ct 3a
ct 3b
xrcesess 3
xrcesess 4
ct 4
xrcesess 3
xrcesess 4
xrcesess 3
xrcesess 4
Created subscriber topic_0:
Debug: number of DDS handles: 2
Published message Hello World! from ubun
Callback: node:ubun heard: Hello World! from ubun
Published message Hello World! from ubun
Callback: node:ubun heard: Hello World! from ubun
Callback: node:ubun heard: Hello World! from ros2nc2
Published message Hello World! from ubun
Callback: node:ubun heard: Hello World! from ubun
Callback: node:ubun heard: Hello World! from ros2nc1
Published message Hello World! from ubun
Callback: node:ubun heard: Hello World! from ubun
Callback: node:ubun heard: Hello World! from ros2nc2
Published message Hello World! from ubun
Callback: node:ubun heard: Hello World! from ubun
Callback: node:ubun heard: Hello World! from ros2nc1
Callback: node:ubun heard: Hello World! from ros2nc2
Published message Hello World! from ubun
Callback: node:ubun heard: Hello World! from ubun
Published message Hello World! from ubun
Callback: node:ubun heard: Hello World! from ubun
Callback: node:ubun heard: Hello World! from ros2nc1
Callback: node:ubun heard: Hello World! from ros2nc2
Published message Hello World! from ubun
```

Top-down Results: ROS2 Tool on Guest

Left: rosqt graphical tools shows nodes/topics that are active. Right: “ros2” command line tool shows list of nodes and can see into the topic stream (‘echo’ command)

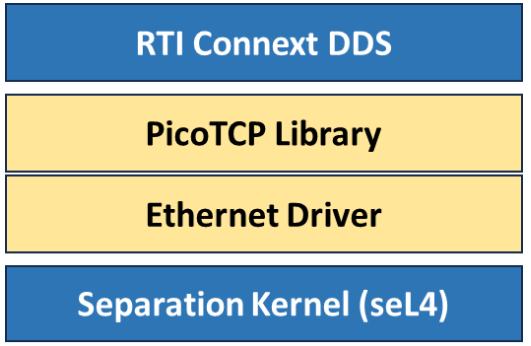


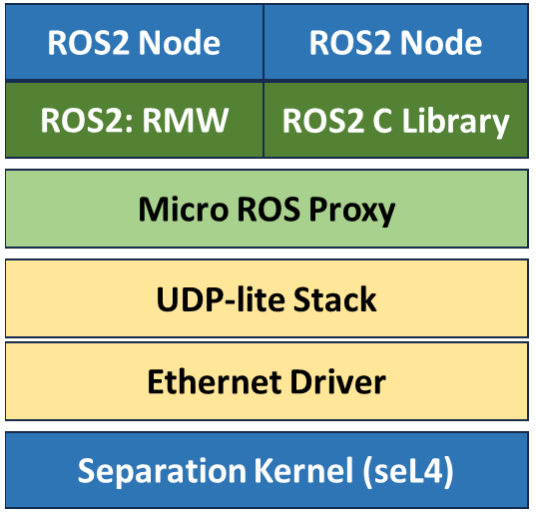

```
dal@dal-Nuvo-7501-Series2:~$ ros2 node list
/executor_examples/ros2nc1
/executor_examples/ros2nc2
/executor_examples/ubun
```

```
dal@dal-Nuvo-7501-Series2:~$ ros2 topic echo /exec
2025-04-18 15:56:36.342 [RTPS_MSG_OUT Error] error
me -> Function init
data: Hello World! from ubun
---
data: Hello World! from ubun
---
data: Hello World! from ros2nc1
---
data: Hello World! from ros2nc2
---
data: Hello World! from ubun
---
data: Hello World! from ubun
---
data: Hello World! from ros2nc1
---
data: Hello World! from ros2nc2
---
```

Split Approach Concluding Remarks

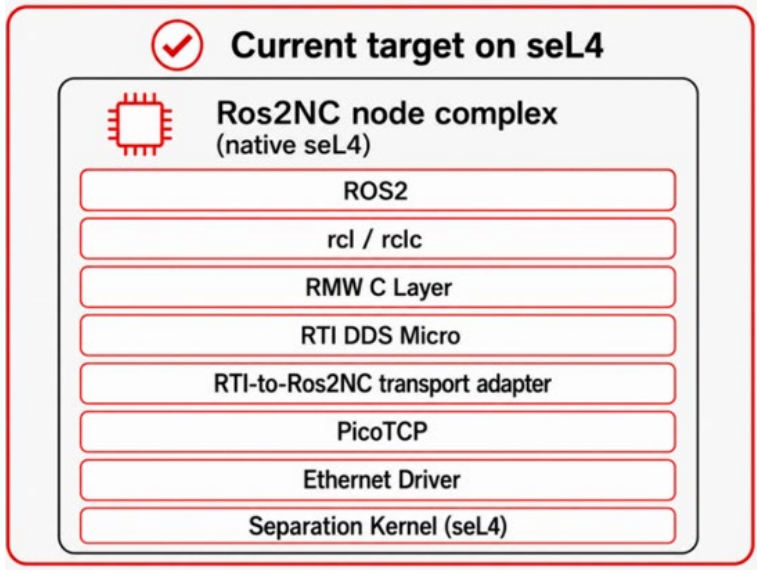
- The team has successfully achieved the Bottom-up development goal – making RTI Connext DDS Micro work properly and natively in an seL4 environment.
- The team has successfully achieved the Top-down development goal – making ROS2 work properly and natively in an seL4 environment.
- Integration of these two development paths is Work in Progress.



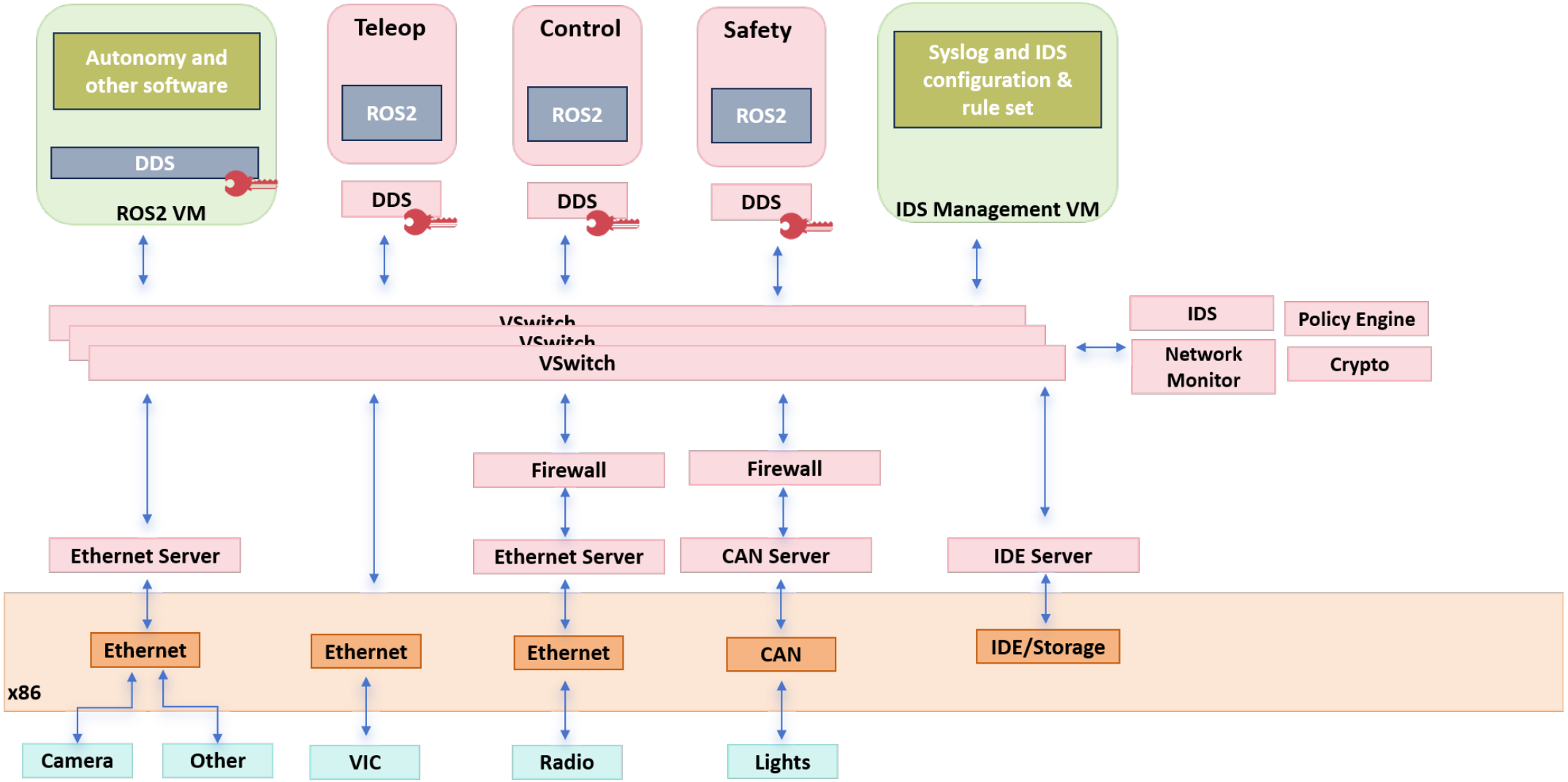


Work In Progress

- Merge two development paths
  - Requires “C” RMW layer for RTI DDS to be developed
  - Initial integration is complete: able to use full RTI Connext DDS with native seL4 ROS2 Node
- Support for C++ ROS2 applications on seL4
  - C++ stdlib !!!
- Upgrade software
  - CAmkES and seL4
  - ROS2 foxy to humble
  - Upgrade Guest Linux version
  - RTI DDS V3 to V4
- Security design and implementation



ROS2, DDS and Security



Q&A