MODULAR SHOPFLOOR CONTROL SOFTWARE DESIGN

Presenter: Hanif Zaini, Development Scientist

Date: 8th March 2022

Managed by Advanced Remanufacturing and Technology Centre ARTC

Outline

- General Background

- Design Methodology
  - Software Architecture
  - Task system and dependencies
  - User-defined script
  - Process flows brief

- Results
  - Capabilities
  - Shortcomings

- Future Work

Background – Industry Project

- **Project with industry partner**
  - Info is generalized
  - Specific details are omitted

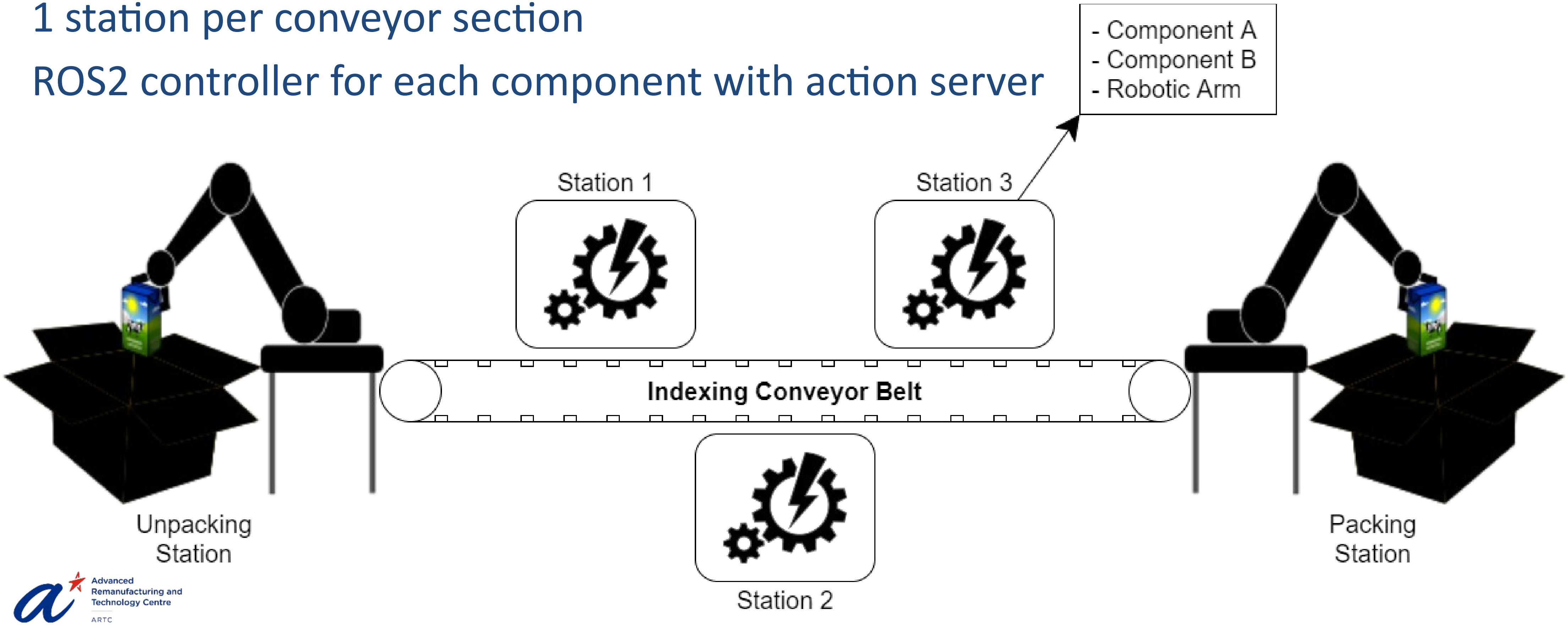
- **Task: Integrate multiple systems in a robotic cell**
  - Central controller/orchestrator to coordinate

- **Use ROS2 environment and framework**
  - Python

- **Not meant to be comprehensive solution for all similar scenarios**
  - Sharing of methods

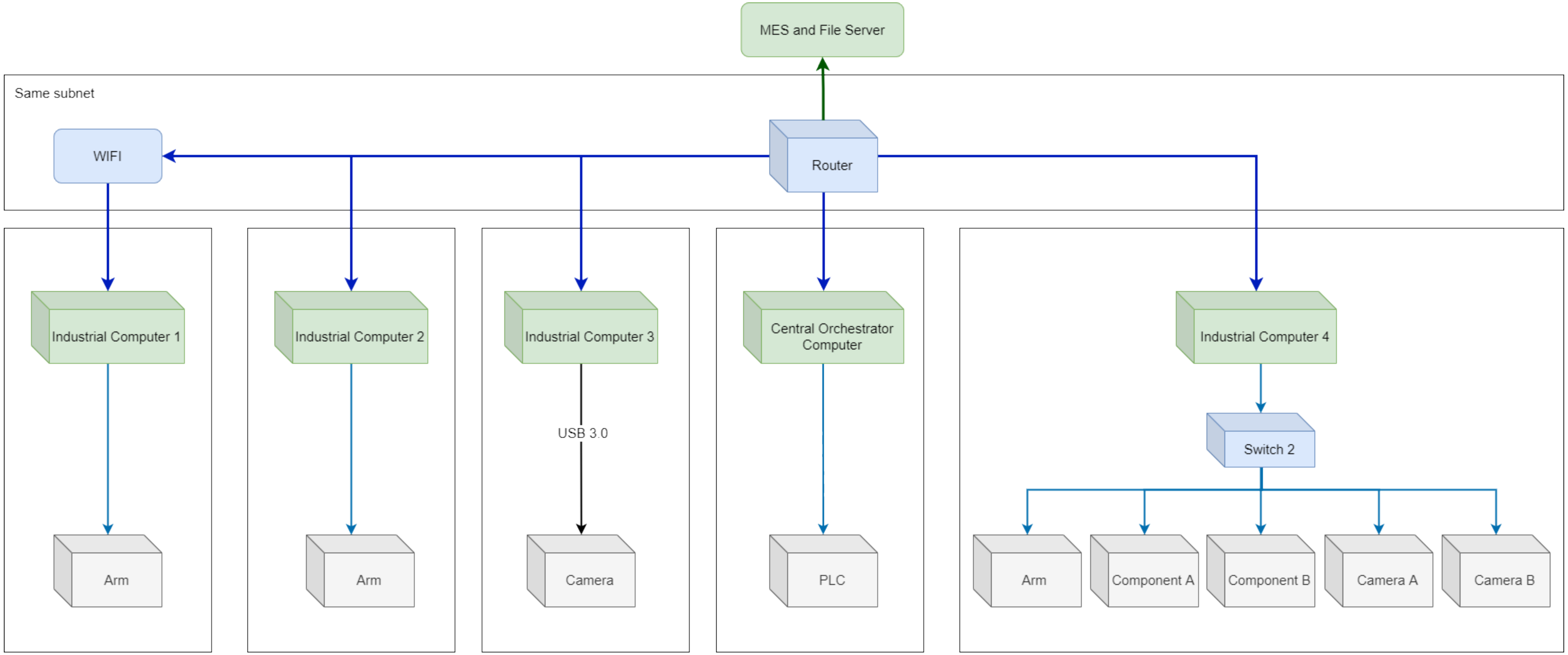
Background – Generalized Robotic Cell Layout

- Unpacking and packing stations
- Indexing conveyor belt
- 1 or more stations (Each with 1 or more components)
- 1 station per conveyor section
- ROS2 controller for each component with action server



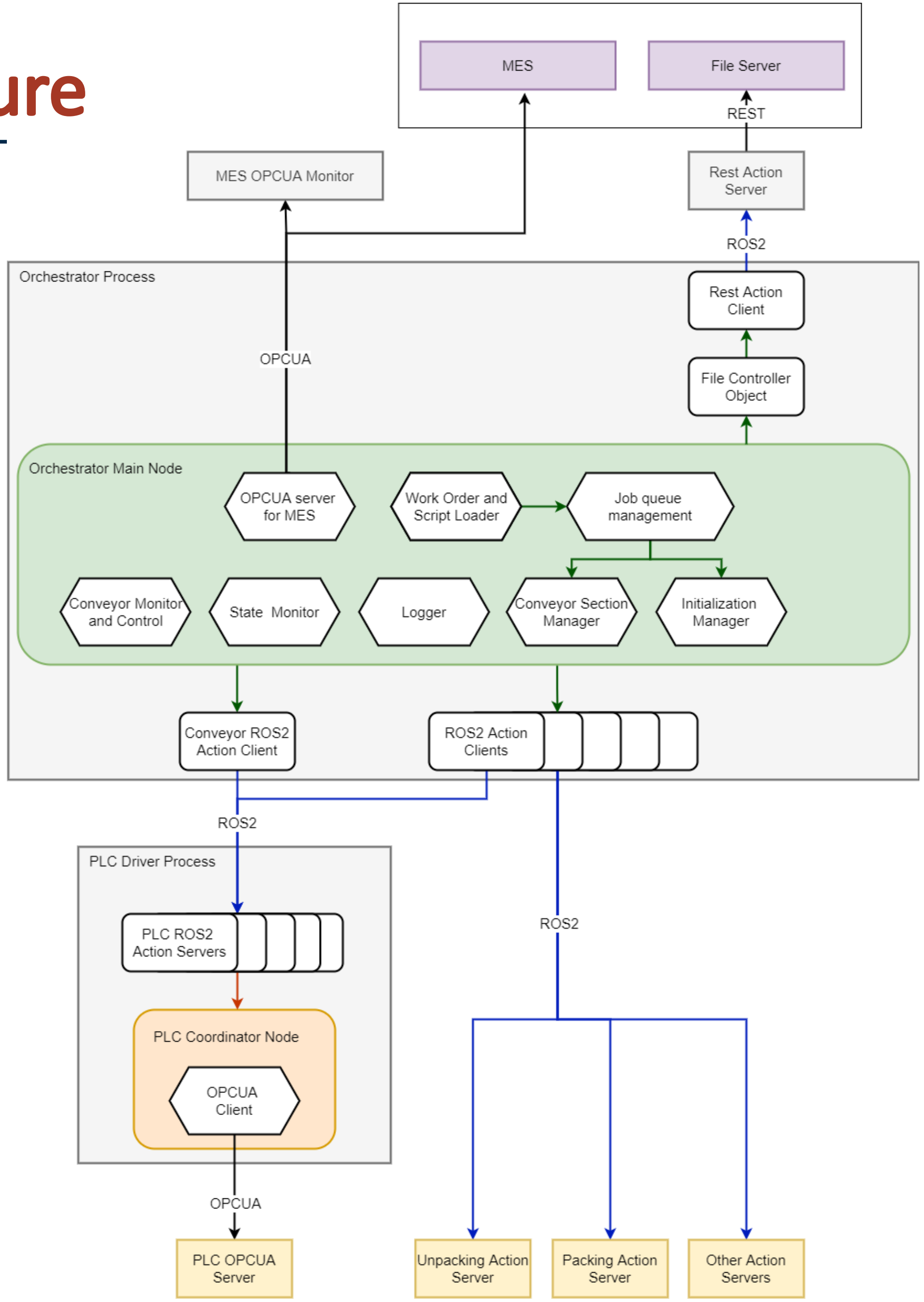
Background – Network Architecture

- ROS2 controllers and action servers run on multiple computers
  - All computers on the same subnet
  - Controlled components are not on the same subnet



Design – Software Architecture

- **Core of Orchestrator – Job Queue management system**
  - Loads and pops conveyor section and initialization manager objects
  - Tracks job completion
- **ROS2 Action clients**
  - Can be loaded from list at runtime using templated action client
  - Require only name and type
- **Multi-threaded executor in orchestrator main node**
  - To run multiple clients and timer loops concurrently



Some details in next slides

Managed by Advanced Remanufacturing and Technology Centre ARTC

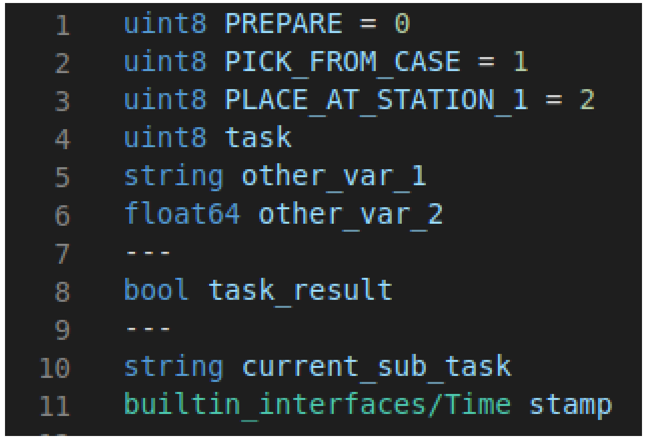
Design – Templated Action Client Class

- **Parse action type to import action class**
  - E.g. this_project_msgs/action/TypeOfAction

- **Create action client using passed main node object**
  - Stored in dictionary with action name as key

- **Action type definition consists of**
  - “task” variable with predefined task constants
  - Other variables
  - Result may also include other variables in addition to “task_result”, e.g. item presence check by a camera component

Example Action Definition



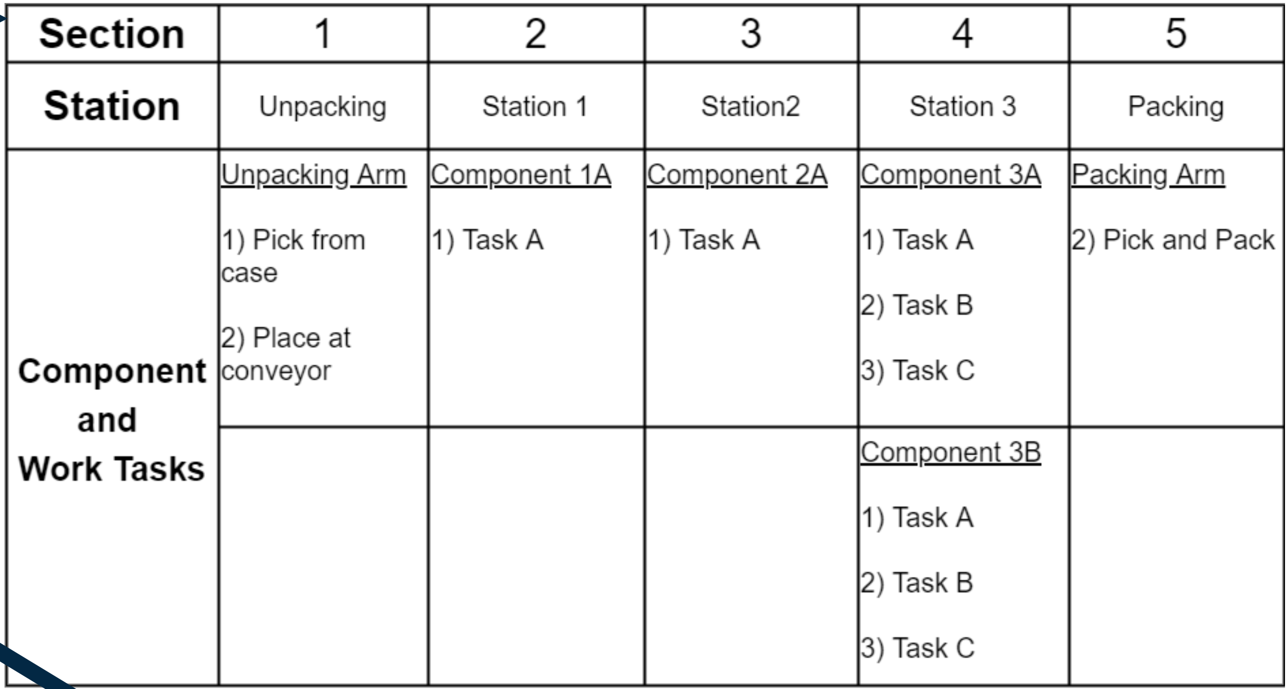
```
uint8 PREPARE = 0
uint8 PICK_FROM_CASE = 1
uint8 PLACE_AT_STATION_1 = 2
uint8 task
string other_var_1
float64 other_var_2
---
bool task_result
---
string current_sub_task
builtin_interfaces/Time stamp
```

- **General send_goal method:**
  - That runs “send_goal_async” method of the action client
  - Argument values for the other variables specific to job are pulled if exists

- **Other standardized features:**
  - Cancel goal, state message subscriber, result and response handling

Design – Task System

- **Each component has a set of tasks**
  - Defined in its action type
- **User-defined script loaded at runtime upon receipt of job work order**
  - Defines sequence of tasks for each section
- **Job registration loads section and initialization manager objects into job queue**
  - Each manager object manages execution of task list and alter job and system state variables accordingly
  - Tasks parameters can be set: continue-if-fail, requires conveyor, quantity change, special arguments
  - Completion of each cycle releases the conveyor to move one index



Section	1	2	3	4	5
Station	Unpacking	Station 1	Station2	Station 3	Packing
Component and Work Tasks	Unpacking Arm 1) Pick from case 2) Place at conveyor	Component 1A 1) Task A	Component 2A 1) Task A	Component 3A 1) Task A 2) Task B 3) Task C	Packing Arm 2) Pick and Pack
				Component 3B 1) Task A 2) Task B 3) Task C	

Example task list for section 4 (Station 3)

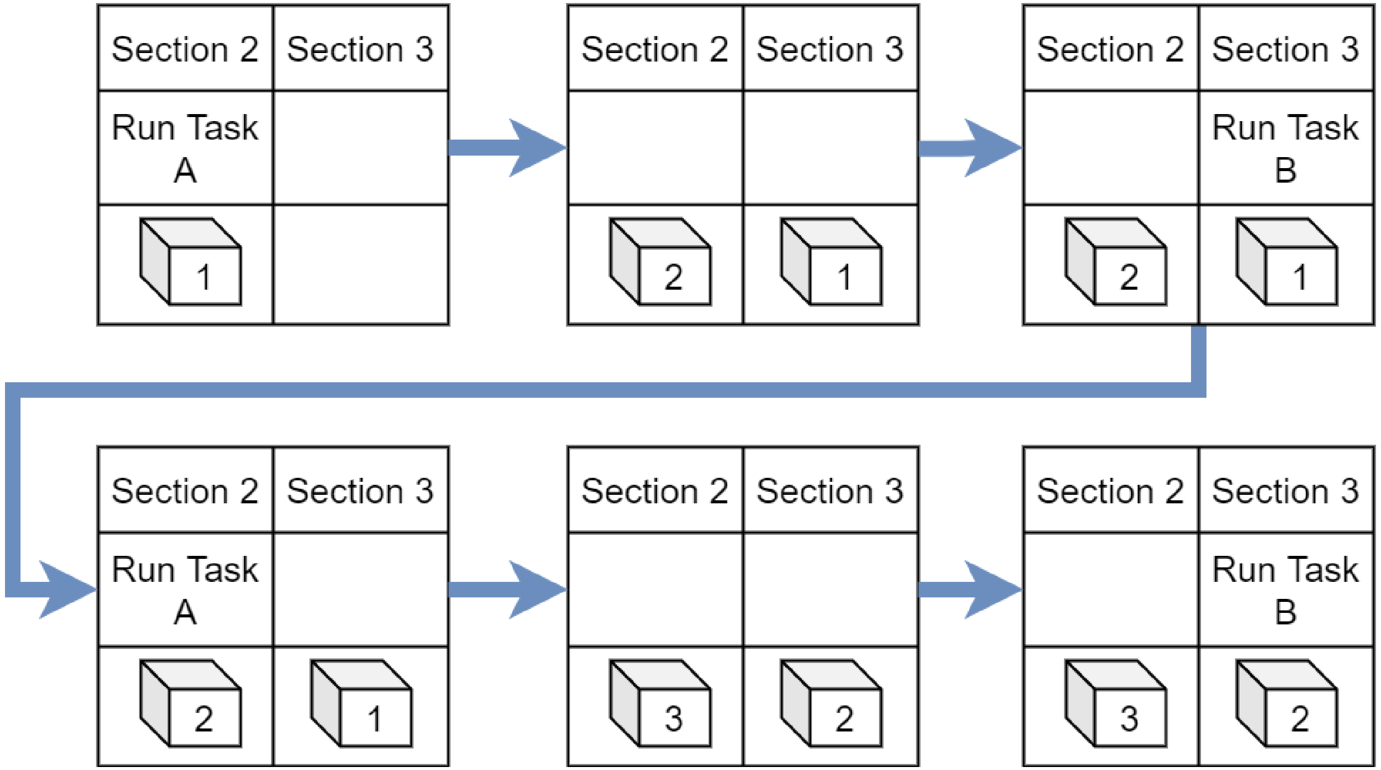
1) 3A – Task A
2) 3B – Task A
3) 3B – Task B
4) 3C – Task A

Release conveyor then repeat cycle for each item from shipping case

Design – Lateral Task Dependency

- **Some downstream tasks require completion of upstream task before executing**
  - Specified in user-defined script
  - Flags stored in orchestrator main node (not manager objects) coordinate execution sequence

- **Flag system brief**:
  - Upstream task sets flag “True” after execution
  - Downstream task executes only if upstream flag is “True”, sets to “False” after
  - Upstream task only executes again if its flag is “False”
  - Repeat

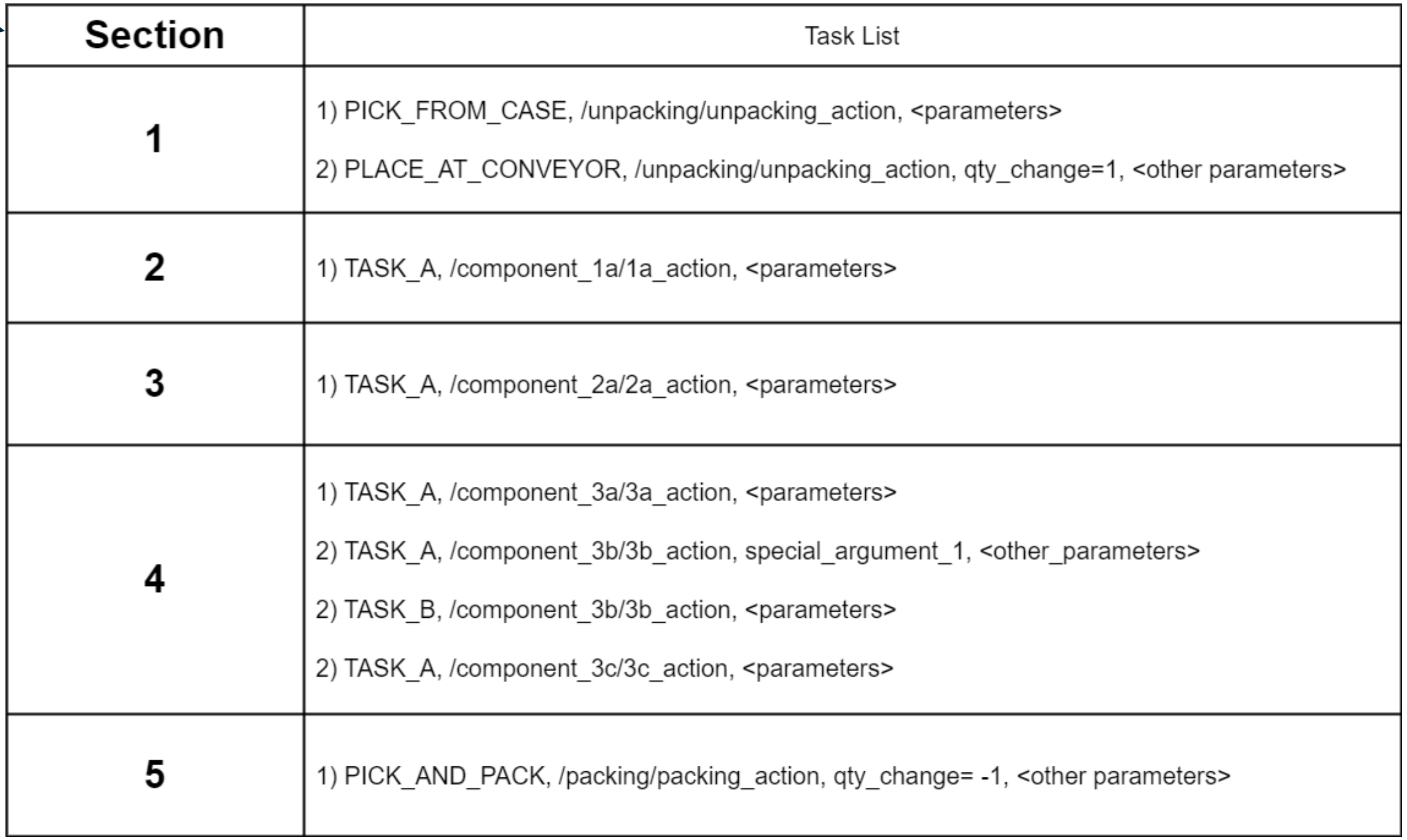


Reasons for this system:
1) Data dependency from upstream task (no buffer)
2) Avoid physical collision

Design – User-defined Script

- **Main process task list example**
  - Tasks are identified by task name and action name
  - Action name is sufficient to look up action client in orchestrator main

- **Others:**
  - Initialization task list
  - Lateral task dependency
  - Initialization task dependency
  - Special argument variables corresponding to defined action type (to be pulled by respective action client)
  - Shipping case quantity

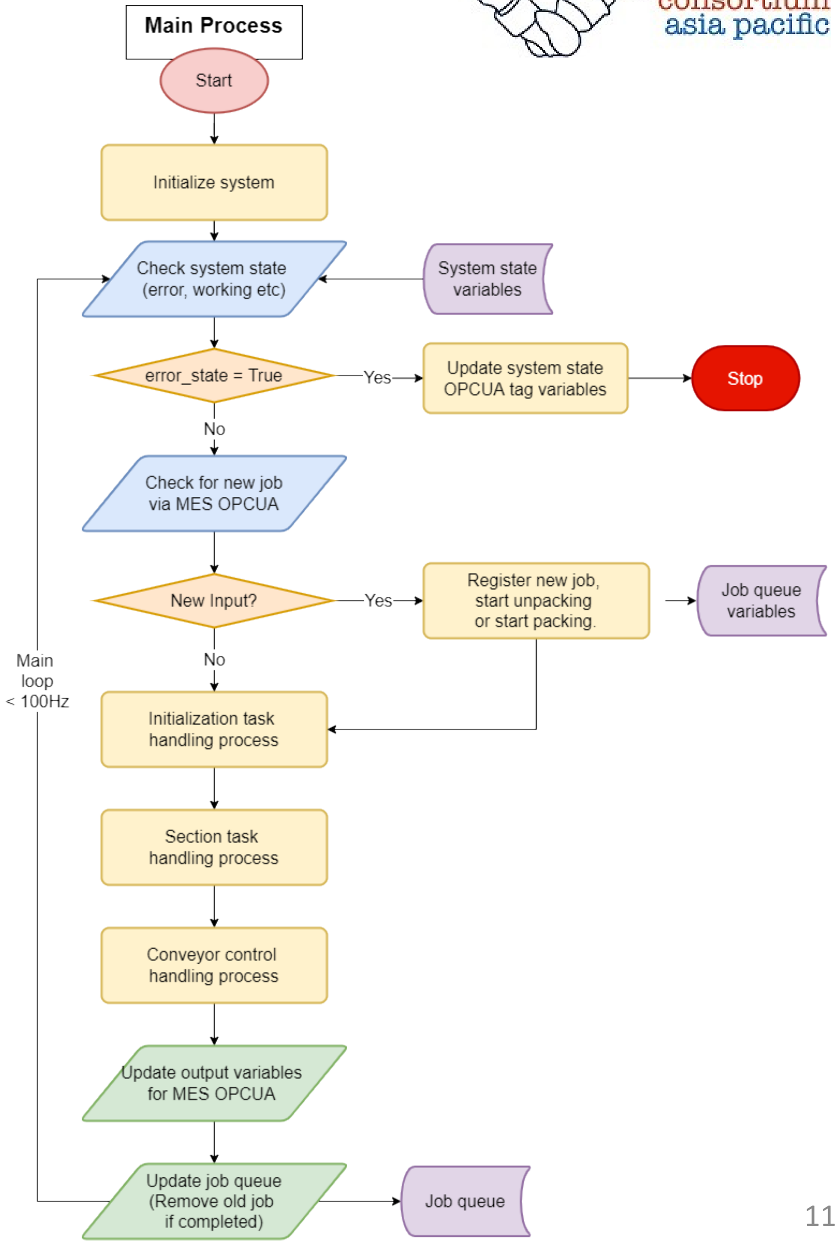


Section	Task List
1	1) PICK_FROM_CASE, /unpacking/unpacking_action, <parameters> 2) PLACE_AT_CONVEYOR, /unpacking/unpacking_action, qty_change=1, <other parameters>
2	1) TASK_A, /component_1a/1a_action, <parameters>
3	1) TASK_A, /component_2a/2a_action, <parameters>
4	1) TASK_A, /component_3a/3a_action, <parameters> 2) TASK_A, /component_3b/3b_action, special_argument_1, <other_parameters> 2) TASK_B, /component_3b/3b_action, <parameters> 2) TASK_A, /component_3c/3c_action, <parameters>
5	1) PICK_AND_PACK, /packing/packing_action, qty_change= -1, <other parameters>

Design – Main Process Flow

- **The main process is a simple loop**
  - Each loop checks through inputs and states
  - Then runs task management processes
  - Lastly, updates states

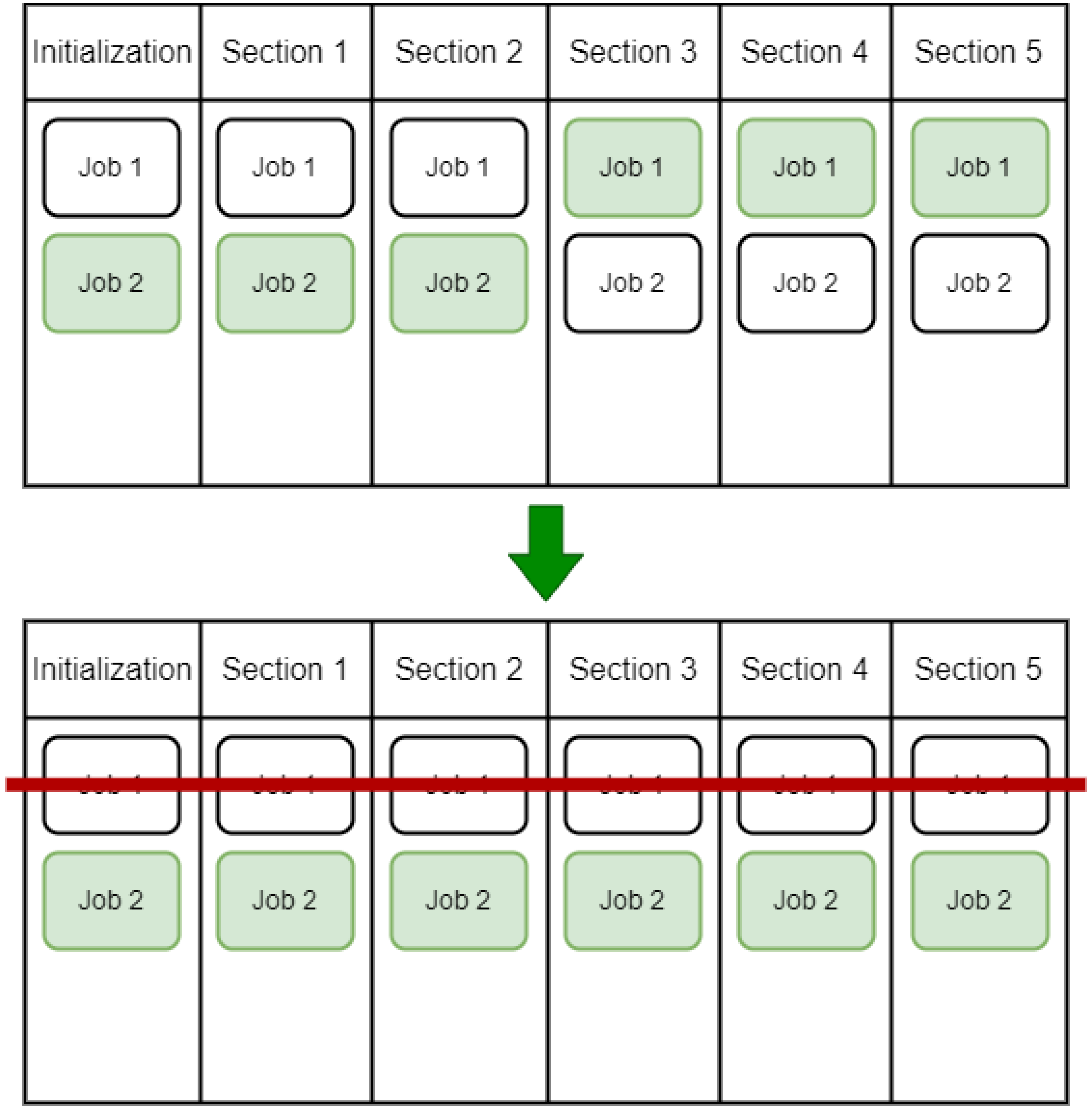
- **Not limited to single loop**
  - May separate into multiple loops using separate ROS2 timers
  - But reduces need for thread locking (avoid race condition)


Main Process
Start
Initialize system
Check system state (error, working etc)
System state variables
error_state = True
Yes
Update system state OPCUA tag variables
Stop
No
Check for new job via MES OPCUA
New Input?
Yes
Register new job, start unpacking or start packing.
Job queue variables
No
Main loop < 100Hz
Initialization task handling process
Section task handling process
Conveyor control handling process
Update output variables for MES OPCUA
Update job queue (Remove old job if completed)
Job queue

Managed by Advanced Remanufacturing and Technology Centre ARTC

Design – Job Queue Processing

- **Each conveyor section and the initialization stage has a queue**
  - New jobs result in creation of initialization and section manager objects that are added to the queue
- **The manager object for the job that is first in queue is active unless it has finished processing the job.**
  - These objects execute respective tasks and track/update job status
  - The queue switches to process on next job in queue if the previous is finished (smooth continuation)
- **When all sections have completed the job, the manager objects are popped/deleted from the queue**



Initialization	Section 1	Section 2	Section 3	Section 4	Section 5
Job 1	Job 1	Job 1	Job 1	Job 1	Job 1
Job 2	Job 2	Job 2	Job 2	Job 2	Job 2

Initialization	Section 1	Section 2	Section 3	Section 4	Section 5
~~Job 1~~	~~Job 1~~	~~Job 1~~	~~Job 1~~	~~Job 1~~	~~Job 1~~
Job 2	Job 2	Job 2	Job 2	Job 2	Job 2

Design – Task Execution Process Flow

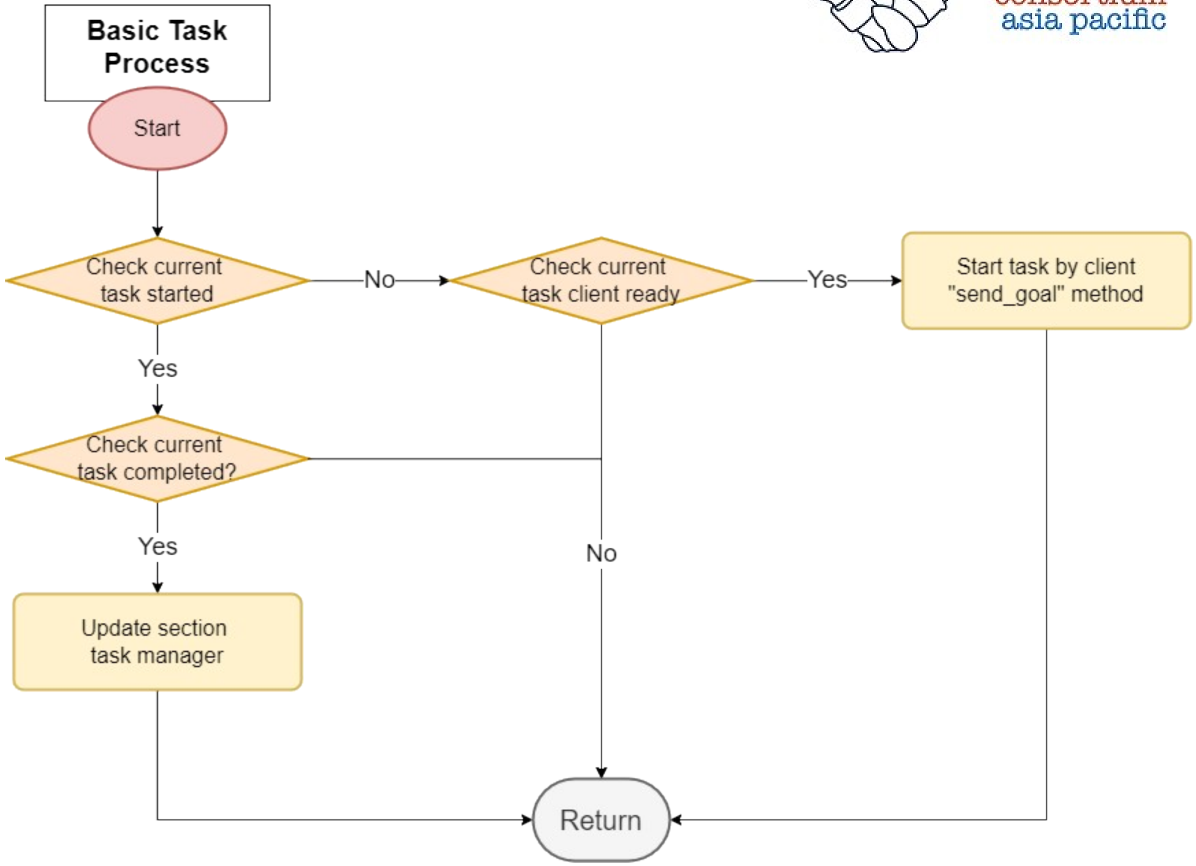
ROS industrial consortium asia pacific

- The basic task execution process
  - For each section, initialization stage and conveyor control
  - Each type has its own additional checks and sequencing not shown here
  - Each cycle of the orchestrator main process runs this process

- If not running task,
  - Run task and return

- If task is running,
  - Check for task status and update
  - Then return

- Minimize blocking



Basic Task Process

Start

Check current task started

No

Check current task client ready

Yes

Start task by client "send_goal" method

Yes

Check current task completed?

Yes

No

Update section task manager

Return

Result – System Capabilities

- **Applicable to similar layouts**
  - Unpacking and packing at ends of conveyor
  - Indexing conveyor
  - Processing stations at conveyor sections

- **Easy code reconfiguration**
  - Action clients loaded by template class and are standardized.
  - Script requires task name and action name only to execute tasks via action clients

- **Queue multiple jobs and execute back-to-back**

- **Task dependency system**
  - Configurable to avoid collision or sequencing for data (E.g. camera in upstream section)

Result – Current Shortcomings

- **Extraction and storing of information for display, logging or output to communication interface is not simple**
  - The OPCUA output for MES currently pulls information from various locations and not in a standardized way
  - Requires modification of templated action client for certain outputs

- **Job and queue data storage is not yet standardized**
  - The current version pulls information from both main process, queue data and manager objects
  - Requires revisit and refactoring

- **Scripts creation not automated**
  - Simplified but manual

Future Work

- **Refactoring for reusability**
  - Reorganize job and queue data storage
  - Standardize logging and data output methods

- **Script auto-generation**
  - Based on a given work order and the configuration of system

- **User interface**
  - Rudimentary interface created but requires more work to make it end-user friendly

THANK YOU!

Managed by

Advanced Remanufacturing and Technology Centre
ARTC