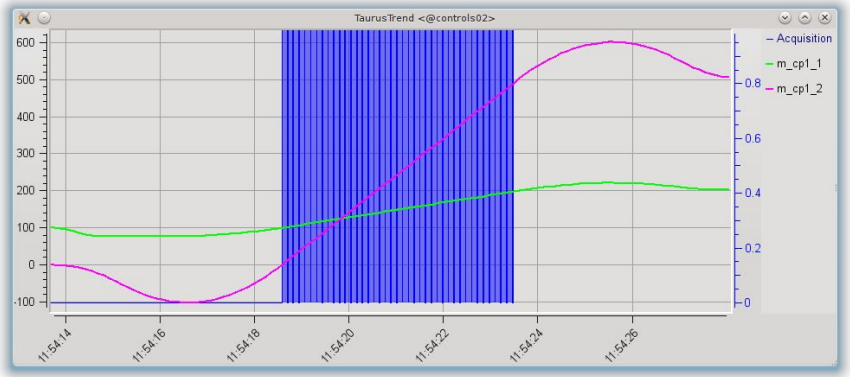
Sardana Trajectories:
Preliminary Core Design for Synchronized Motion and Acquisition

Jordi Aguilar
Roberto Homs
Zbigniew Reszela
Oriol Vallcorba

on behalf of
ALBA Synchrotron
Controls Section

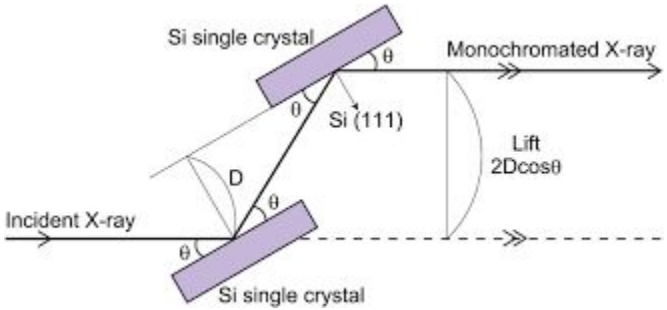
Pseudo-motors limitations

- Do not implement velocity and acceleration
- Sardana continuous scan framework implements constant velocity for all physical motors of one pseudo-motor, does not follow the trajectory.
- Pseudo-motors do not implement movement logic, only calculation.

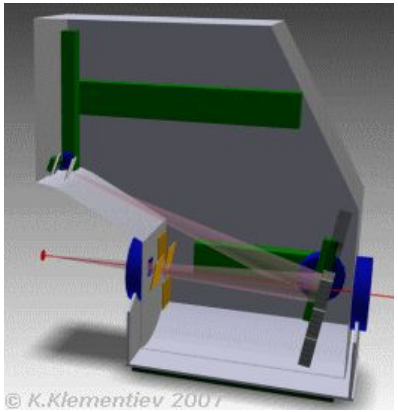


Motivations to implement trajectory

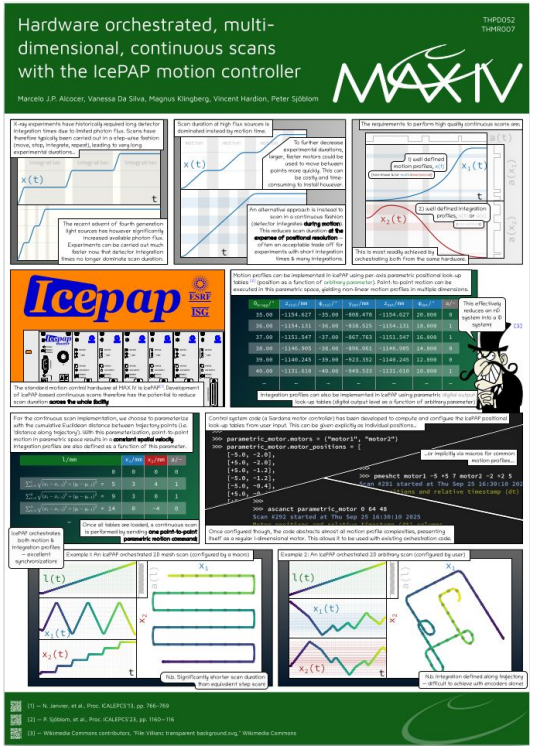
- **Avoid collisions**
- **Increase scan velocity**
- **Dynamic**: meshct, aXscanct, helical, etc
- **Static**: Pseudo-motors



Double Crystal Monochromator



CLEAR Spectrometer
Core Level Emission Analyzer and Reflectometer
(by K. Klementiev)



Objective: Support different Motion Controller Hardware

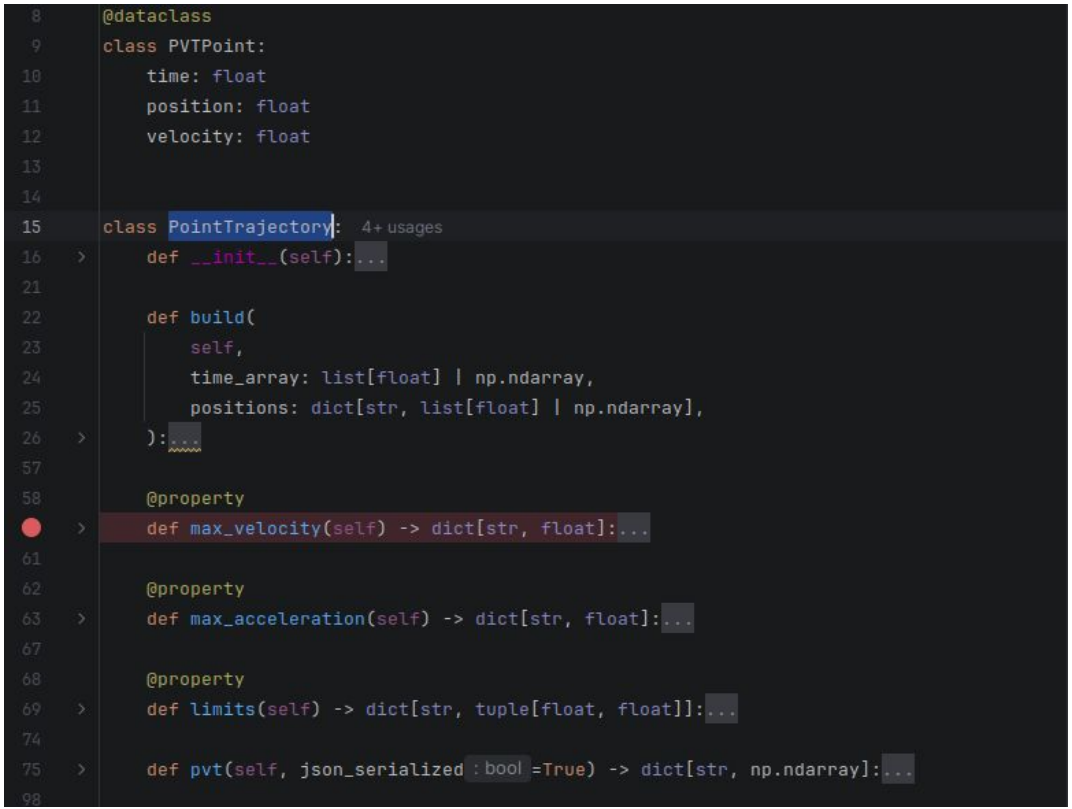
Aerotech
Delta-tau Pmac
Smaract MCS2
Icepap
Beckhoff
…..
Support: Virtual Axis, PVT/PT, G-Code.

Sardana core requirements

- New motion states
- New attributes/method for configuration and execution on the Motor controller
- New motion loop action
- Adapt Generic Scan Framework

Out of the scope: Sardana is not responsible for synchronizing two controllers
e.g: Icepap-Aerotech, Icepap-Icepap

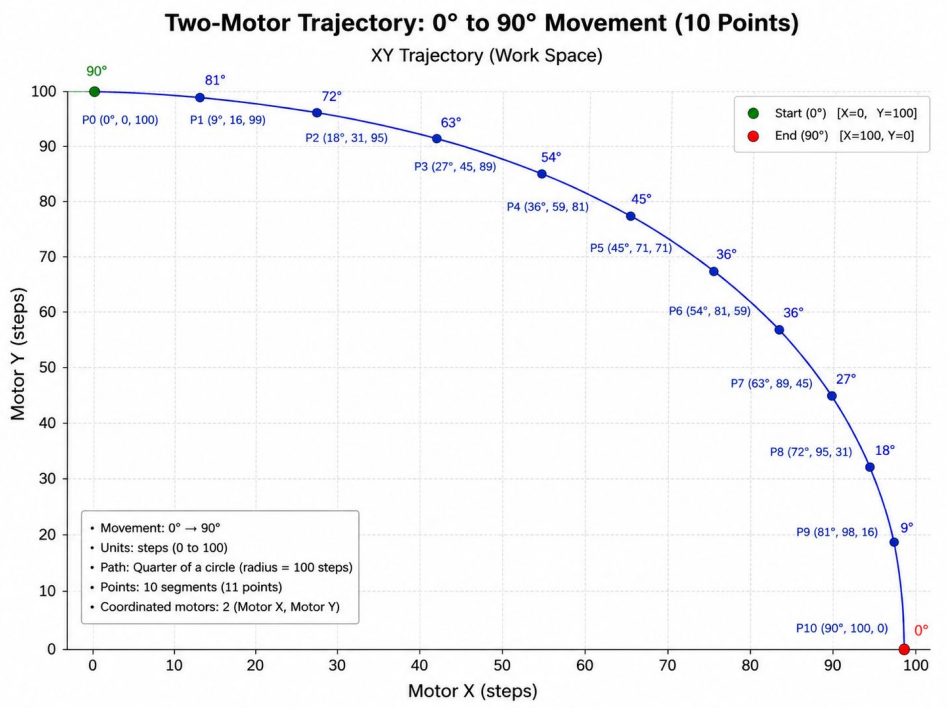
Helpers (sardana.util.motion.trajectory)



```python
@dataclass
class PVTPoint:
    time: float
    position: float
    velocity: float


class PointTrajectory:  4+ usages
    def __init__(self):...

    def build(
        self,
        time_array: list[float] | np.ndarray,
        positions: dict[str, list[float] | np.ndarray],
    ):...

    @property
    def max_velocity(self) -> dict[str, float]:...

    @property
    def max_acceleration(self) -> dict[str, float]:...

    @property
    def limits(self) -> dict[str, tuple[float, float]]:...

    def pvt(self, json_serialized : bool =True) -> dict[str, np.ndarray]:...
```

Helpers (sardana.util.motion.trajectory)

```python
@dataclass
class PVTPoint:
    time: float
    position: float
    velocity: float


class PointTrajectory:  4+ usages
    def __init__(self):...

    def build(
        self,
        time_array: list[float] | np.ndarray,
        positions: dict[str, list[float] | np.ndarray],
    ):...

    @property
    def max_velocity(self) -> dict[str, float]:...

    @property
    def max_acceleration(self) -> dict[str, float]:...

    @property
    def limits(self) -> dict[str, tuple[float, float]]:...

    def pvt(self, json_serialized : bool =True) -> dict[str, np.ndarray]:...
```



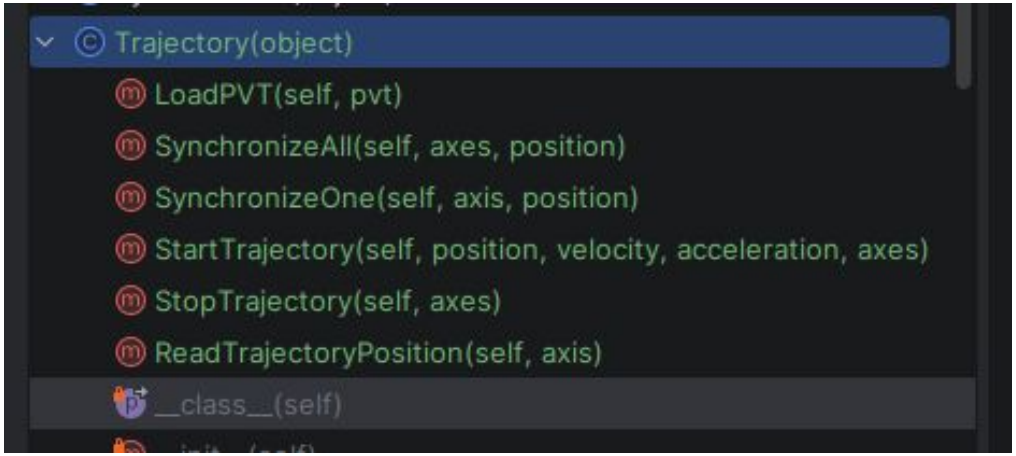
Helpers (sardana.util.motion.trajectory)

```
15
16  def createTrajectory():
17      alpha = numpy.linspace(0,90,10)
18      times = numpy.linspace(0,10,10)
19      x_positions = numpy.array(numpy.cos(numpy.deg2rad(alpha))*100, int)
20      y_positions = numpy.array(numpy.sin(numpy.deg2rad(alpha))*100, int)
21      test_traj_ipap = PointTrajectory()
22      test_traj_ipap.build(times, positions: {TRAJ_MOT01: x_positions, TRAJ_MOT02: y_positions,
23                                              'param': alpha})
24      return test_traj_ipap
25
```

Helpers (sardana.util.motion.trajectory)

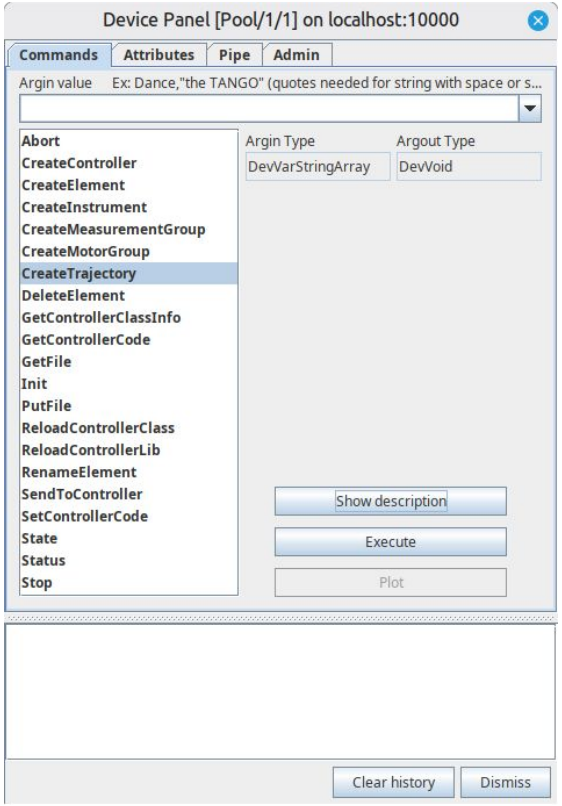
```python
class MeshScanTrajectory:
    def __init__(
        self,
        x_start: float,
        x_stop: float,
        x_points: int,
        y_start: float,
        y_stop: float,
        y_points: int,
        point_time: float,
        snake: bool = True,
    ):...

    def generate_points(self) -> tuple[np.ndarray, dict[str, np.ndarray]]:...

    def build_trajectory(self) -> PointTrajectory:...
```

Motor Controller Interface



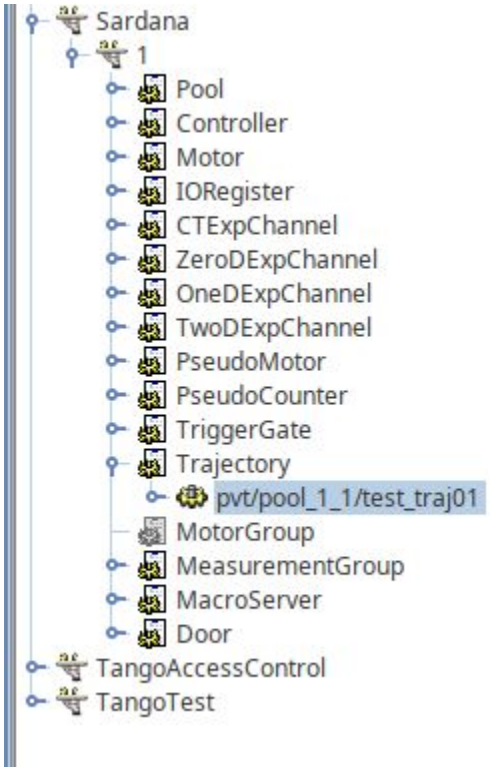

```
class IcepapController(MotorController, Trajectory):  5+ usages  Roberto Javier Homs Puron +9 *
    """
    This class is the Sardana motor controller for the ICEPAP motor controller.
    As part from the standard Pool motor interface per axis, it provides extra
    attributes for some firmware attributes of the driver.
    """
```

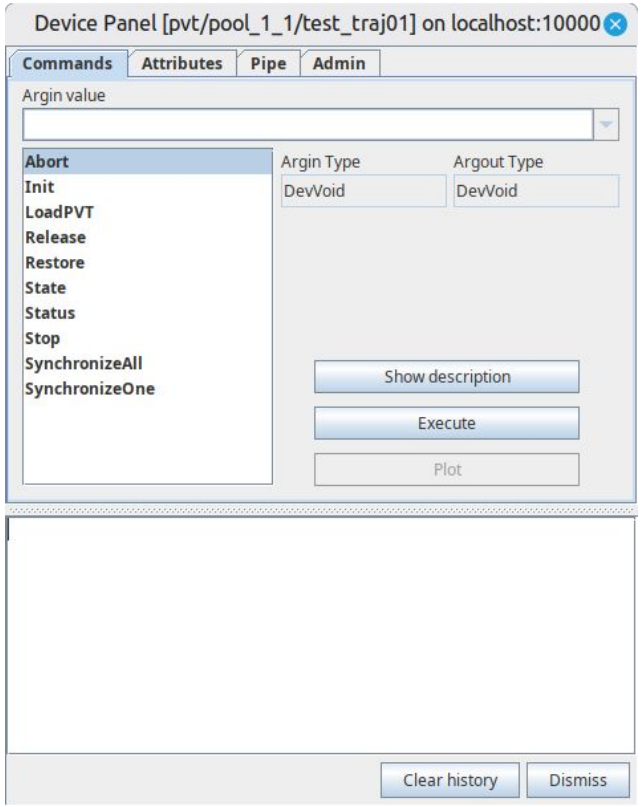
Tango Interface

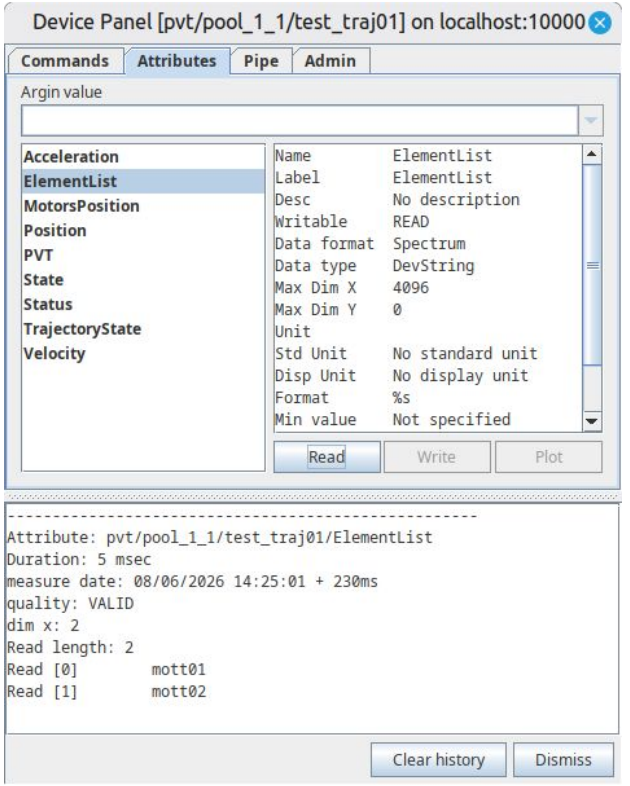


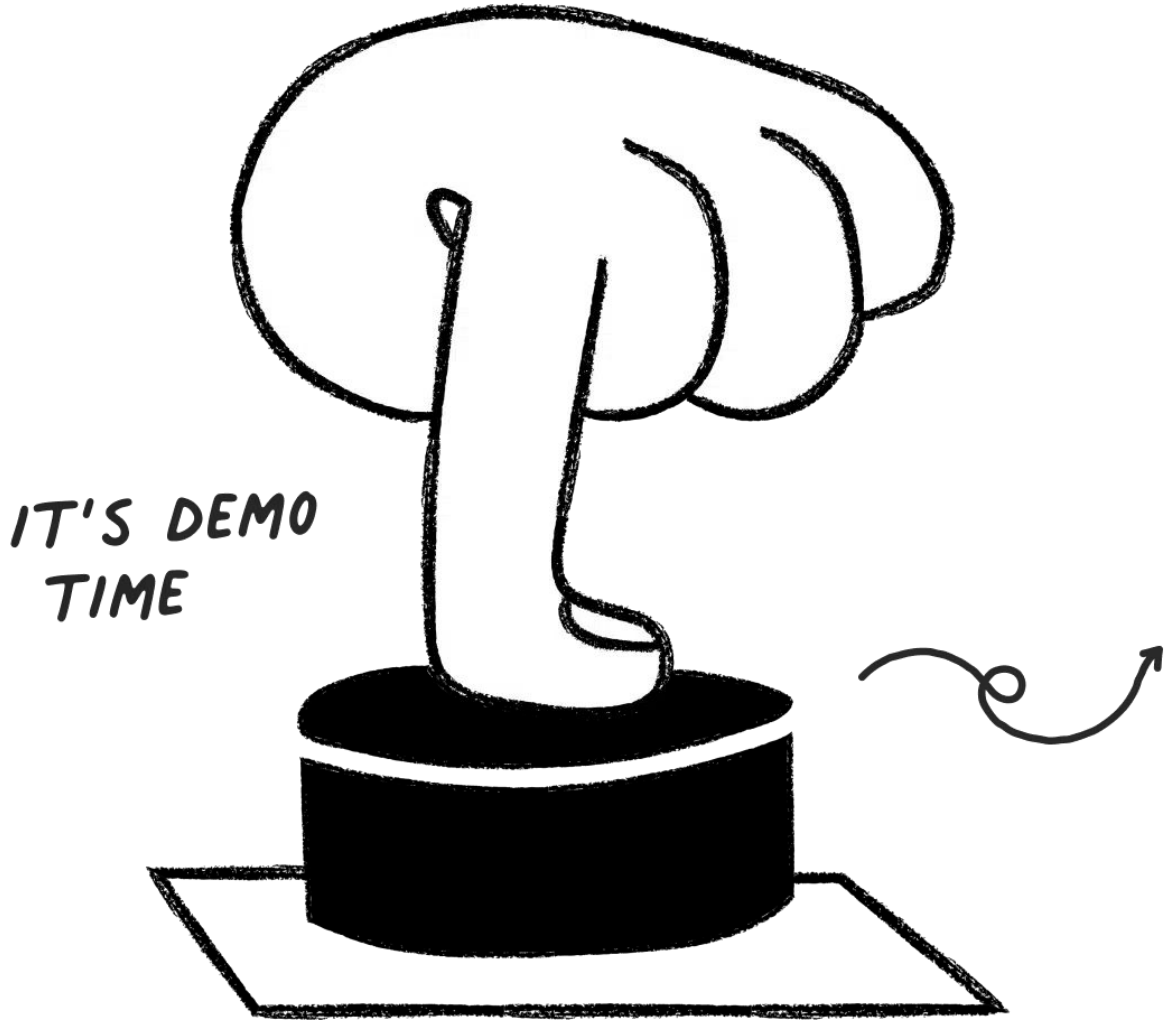

```
if __name__ == "__main__":
    pool = tango.DeviceProxy('tango://localhost:10000/pool/1/1')
    pool.CreateTrajectory([TRAJ_NAME, TRAJ_MOT01, TRAJ_MOT02])
```

Tango Interface

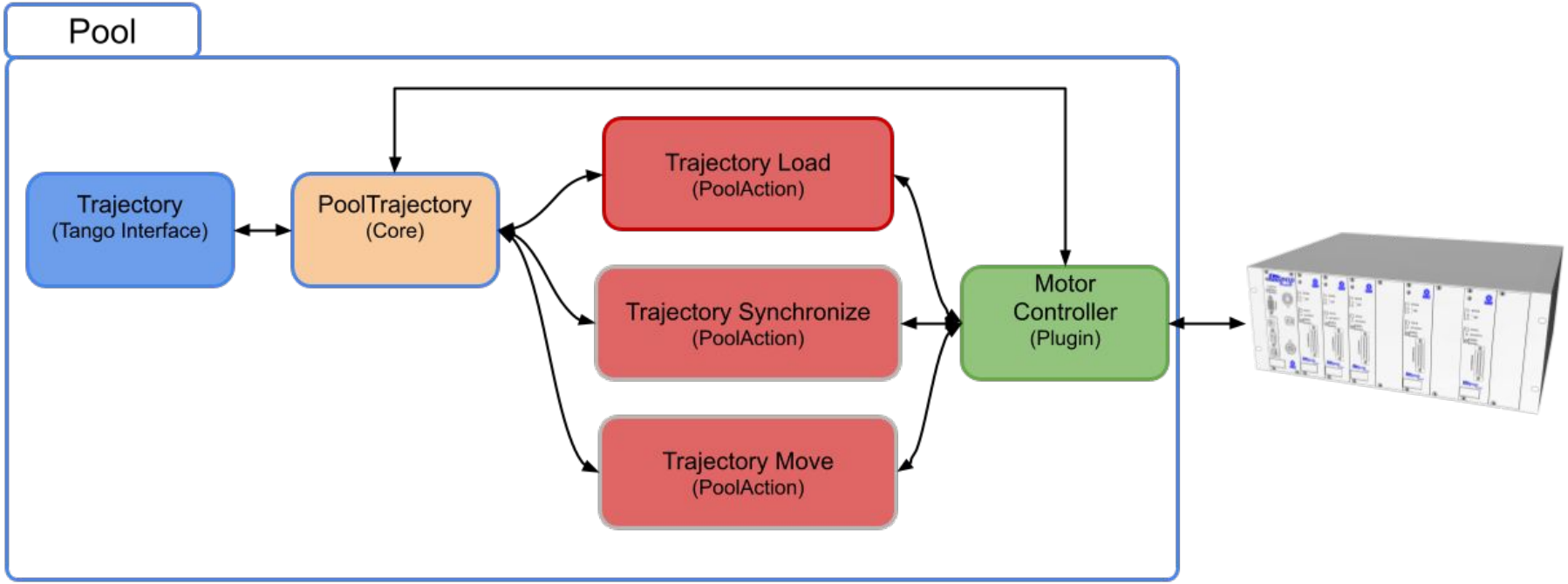







IT'S DEMO
TIME

What's new?


Pool
Trajectory
(Tango Interface)
PoolTrajectory
(Core)
Trajectory Load
(PoolAction)
Trajectory Synchronize
(PoolAction)
Trajectory Move
(PoolAction)
Motor
Controller
(Plugin)

Next steps….

- First MR:
  - Events generation (positions, state machine, …)
  - Adapt/re-define TrajectoryState attribute to Tango Device Server State or keep it
  - Position, Velocity and Acceleration validation
  - Transformation from user unit to motor units
  - Create meshtraj macro (ideas for name are welcome)
  - Documentation :)
- Design solution for pseudo-motors
- Validate PoC with other controllers (Aerotech, Beckoff,....)

Thank You for Your Attention