HELMHOLTZ

ophyd | async
bluesky

ROCK-IT

# SUPPORT FOR THE TANGO CONTROL SYSTEM IN OPHYD-ASYNC AND BLUESKY

Author: Devin Burke – DESY Photon Science Experiment Control (FS-EC) group; ROCK-IT WP2
Co-Author: Yury Matveev – DESY FS-EC

DESY. HZB Helmholtz Zentrum Berlin HZDR KIT

# OPHYD-ASYNC DEVELOPMENT IS MOTIVATED BY THE ROCK-IT PROJECT

**Project Goals**

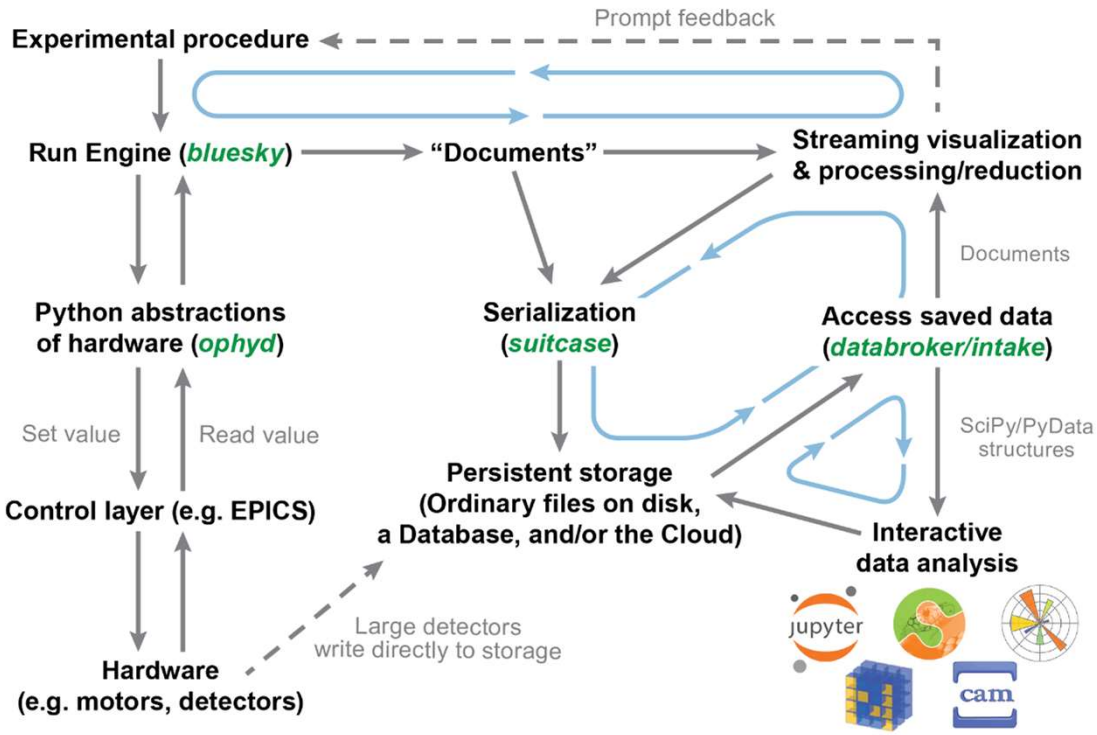
- **Create user-friendly automated experiment environments for non-experts and industry users**
- **Extend in-situ and operando mail-in experiment capabilities with remote access**
- **Improve Instrument accessibility by enabling remote experiment control via web browser**
- **Develop a general purpose set of tools which can be easily transferrable to new types of instruments and experiments**
- **Implement machine-learning for automated experiments, real-time analysis, and robotic sample handling**

Helmholtz Centers map image source:
https://upload.wikimedia.org/wikipedia/commons/3/3a/Map_of_19_member_centers_of_the_Helmholtz_association.png

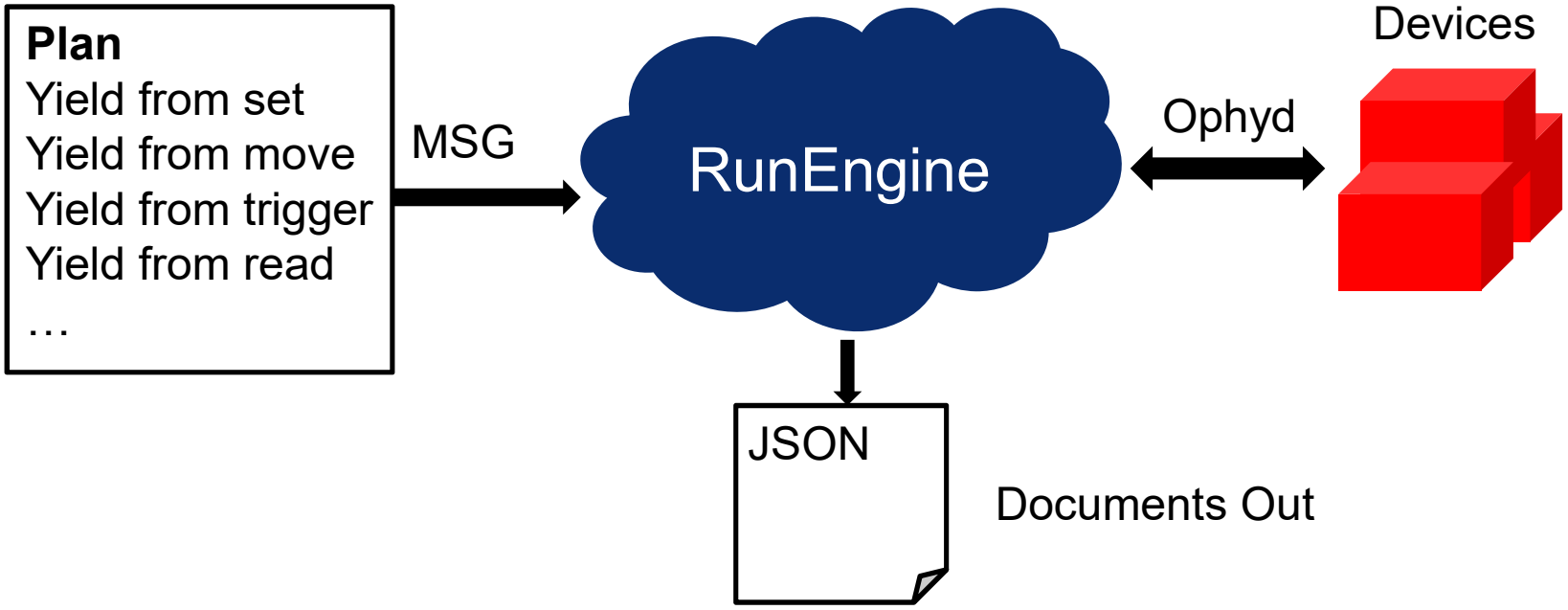
bluesky

# EXPERIMENT ORCHESTRATION

ROCK-IT



# BLUESKY PROCESSES MESSAGES TO ORCHESTRATE DEVICES AND PRODUCES JSON-LIKE “DOCUMENTS”



## BLUESKY IS FEATURE-RICH AND FLEXIBLE


- Interrupts and Error Handling
- Conditional run suspension/abort/resume
- Asynchronous operation
- Live visualization and processing
- Automatic handling of experiment and device metadata
- Debugging and logging tools
- IPython “Magics” for SPEC-like operation
- Document publication to subscribers
- Automatic device baseline measurements before/after run
- Asynchronous monitoring

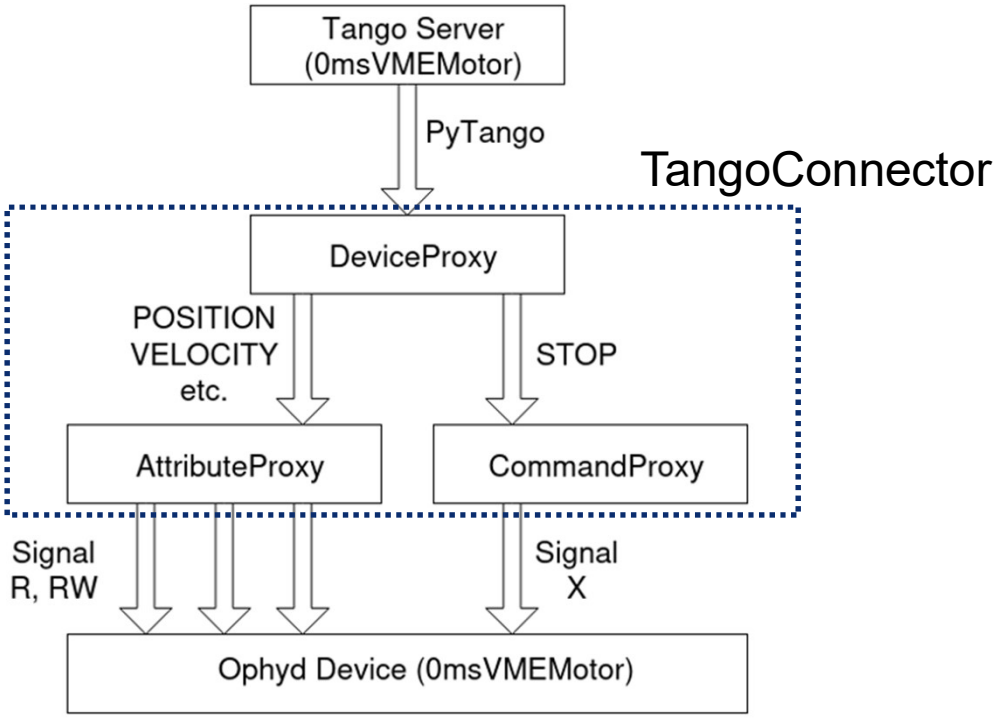
## OPHYD-ASYNC DELIVERS POWERFUL FEATURES AS STANDALONE SOFTWARE


- Classes are “Device” hierarchies.
- Child Signals/Devices are self-describing
- Arbitrary metadata can be associated with Devices/Signals
- Arbitrary Callbacks can be subscribed to signals
- Derived signals with definable read/write behavior supported in Ophyd-Async
- Signals abstract Tango/EPICS/Labview/etc. Devices may have signals with different backend connections.

# OPHYD-ASYNC IS A SUCCESSOR TO OPHYD WHICH TAKES ADVANTAGE OF PYTHON ASYNCIO

ROCK-IT

ophyd | async
bluesky



# OPHYD(-ASYNC) DELIVERS A FLEXIBLE AND GENERIC API FOR DEVICE CONTROL


Set() →

Read() →

Trigger() →

Prepare() →

Describe() →

…

# OPHYD-ASYNC PROVIDES A UNIFIED API TO SHARE COMMUNITY TOOLS


**APS Tools**
- AreaDetector
- NXWriterAPS
- Document callbacks
- Custom scan libraries

**Bluesky Core**
- Common scan library
- Scan component library
- Visualization callbacks
- Document serialization to publishers like ZMQ
- Flyscanning

**Diamond Light Source**
- Definition of scan paths via ScanSpec
- PVT Trajectory scanning in Delta Tau motion controllers
- Position compare and capture using a PandABox

**Community Interfaces**
- Australian light source web-based interface
- Jupyter lab/notebooks
- Queueserver and API

**ROCK-IT**
- bluesky_nexus
- bouquet

# SIMPLE DEVICES CAN BE BUILT AUTOMATICALLY BY SIMPLY PASSING THE TRL TO A TANGOREADABLE DEVICE

```
class Counter(TangoReadable):
    """Very simple devices can be automatically created from a TRL"""
    pass
counter = Counter(trl='p09/counter/eh.01', name='counter')
await counter.connect()
print(counter._child_devices)
```

counter._child_devices =

dict_keys(['Counts', 'Init', 'Offset', 'Read', 'Reset', 'State', 'Status'])

# THE DEVICE API IS BUILT BY INHERITING PROTOCOLS LIKE MOVABLE AND STOPPABLE

ROCK-IT

```python
class OmsVME58Motor(TangoReadable, Movable, Stoppable):
    """Signals can be defined by class annotations or built in __init__ method"""
    Position: A[SignalRW[float], Format.HINTED_UNCACHED_SIGNAL]
    State: A[SignalR[DevState], TangoPolling(0.1)]
    SlewRate: A[SignalRW[float], Format.CONFIG_SIGNAL]
    SlewRateMax: A[SignalRW[float], Format.CONFIG_SIGNAL]
    Conversion: A[SignalRW[float], Format.CONFIG_SIGNAL]
    Acceleration: A[SignalRW[float], Format.CONFIG_SIGNAL]
    StopMove: SignalX

    async def set(self, value, timeout):
        ...

    async def stop(self, success=False):
        ...
```

# FILL OUT ACTIONS USING HELPFUL SIGNAL UTILITIES

```python
class OmsVME58Motor(TangoReadable, Movable, Stoppable):
    """Signals can be defined by class annotations or built in __init__ method"""
    Position: A[SignalRW[float], Format.HINTED_UNCACHED_SIGNAL]
    State: A[SignalR[DevState], TangoPolling(0.1)]
    SlewRate: A[SignalRW[float], Format.CONFIG_SIGNAL]
    SlewRateMax: A[SignalRW[float], Format.CONFIG_SIGNAL]
    Conversion: A[SignalRW[float], Format.CONFIG_SIGNAL]
    Acceleration: A[SignalRW[float], Format.CONFIG_SIGNAL]
    StopMove: SignalX

    @AsyncStatus.wrap
    async def set(self, value, timeout=10):
        await self.Position.set(value, timeout=timeout)
        await wait_for_value(self.State, match="ON", timeout=timeout)

    def stop(self, success=False):
        self._set_success = success
        return self.StopMove.trigger()
```

# NOW LET'S BUILD A SIMPLE FLYSCANNER

ROCK-IT

```
class Flyscanner(StandardReadable, Preparable, Flyable, EventCollectable):
    def __init__(self, motor_trl, counter_trl, name):
        with self.add_children_as_readables():
            self.motor = OmsVME58Motor(motor_trl)
            self.counter = Counter(counter_trl)
        super().__init__(name=name)
        self._start_position = 0.0
        self._end_position = 0.0
        self.data = []
        self.data_task = None

    async def describe_collect(self):
        print("Calling describe_collect")
        description = await super().describe()
        return {'simple_flyer': description}
```

# SIMPLE COROUTINE TO TAKE A MEASUREMENT

```python
async def _measurement(self):
    try:
        print("Starting measurement")
        while True:
            count = await self.counter.Counts.read()
            count_name = self.counter.Counts.name
            position = await self.motor.Position.read()
            position_name = self.motor.Position.name
            partial_event = {
                'data': {
                    count_name: count[count_name]['value'],
                    position_name: position[position_name]['value']
                    },
                'time': time.time(),
                'timestamps': {
                    count_name: count[count_name]['timestamp'],
                    position_name: position[position_name]['timestamp']
                    },
                }
            self.data.append(partial_event)
            await asyncio.sleep(1)
    except asyncio.CancelledError:
        print("Measurement stopped")
```

# DEFINE REMAINING ABSTRACT METHODS

ROCK-IT

```python
@AsyncStatus.wrap
async def prepare(self, value):
    """Prepares a device for scanning"""
    print("Preparing device")
    self._start_position = value[0]
    self._end_position = value[1]
    await self.motor.SlewRate.set(15000)

@AsyncStatus.wrap
async def kickoff(self):
    "Returns a status object marked as done when acquisition has begun"
    print("Kickoff acquisition")
    await self.motor.set(self._start_position, timeout=None)
    await self.motor.Position.set(self._end_position, wait=False)
    # Schedule a task run the measurement coroutine
    self.data_task = asyncio.create_task(self._measurement())
```

# DEFINE REMAINING ABSTRACT METHODS

ROCK-IT

```python
@AsyncStatus.wrap
async def complete(self):
    """Returns a status object marked as done when acquisition has completed"""
    print("Completing acquisition")
    await wait_for_value(self.motor.Position, match=self._end_position, timeout=None)
    if self.data_task:
        self.data_task.cancel()

def collect(self):
    print("Collecting data")
    # Iterate and yield self.data
    for partial_event in self.data:
        yield partial_event
```

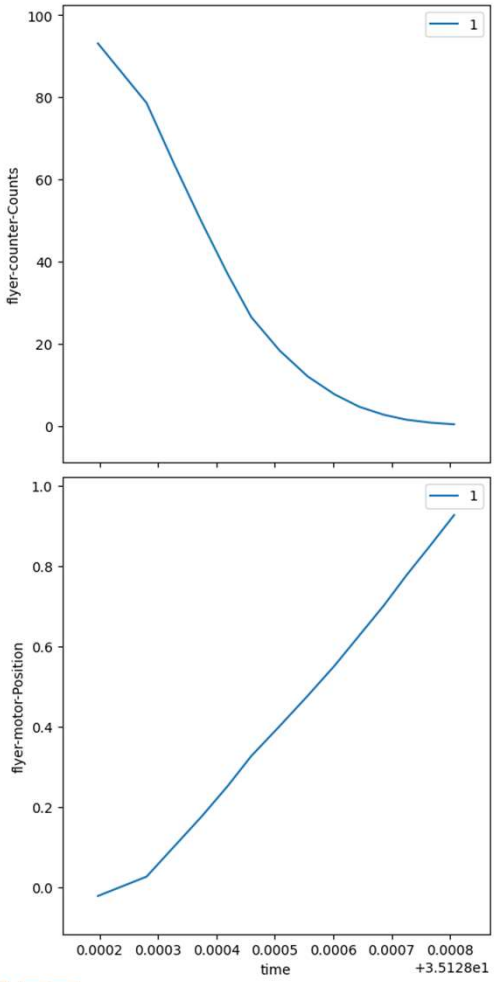
# NOW WE CAN USE ANY PLAN THAT EXPECTS FLYABLES

```
bec = BestEffortCallback()
RE = RunEngine()
RE.subscribe(bec)

RE(bps.prepare(flyer, (0.0, 1.0)), print)
RE(fly([flyer]))
```

```
Preparing device


Transient Scan ID: 1     Time: 2025-05-20 12:11:51
Persistent Unique Scan ID: 'd373ed86-d364-436c-84b4-36b5a64af92f'
Kickoff acquisition
Starting measurement
Completing acquisition
Measurement stopped
Calling describe_collect
New stream: 'simple_flyer'
Collecting data
```




# BLUESKY DOCUMENTS ARE THEN PROCESSED AUTOMATICALLY BY CALLBACKS SUBSCRIBED TO THE RUNENGINE

## PRE-1.0 RELEASE LIMITATIONS


1. Typed commands (SignalX) are not yet supported.
   - A pre-release workaround is implemented.
   - Tango commands with input/output types that are not null are instantiated as SignalRW and treated like attributes.

2. Tango attributes with different input/output types are not supported.

3. Pipes are not supported.

# THANK YOU FOR LISTENING

**Links**:
Ophyd-Async: https://github.com/bluesky/ophyd-async
Bluesky Project Page: https://blueskyproject.io/
Bluesky Mattermost: https://mattermost.hzdr.de/
ROCK-IT Project: https://www.rock-it-project.de/

**Author Contact**:
Dr. Devin Burke – DESY FS-EC; ROCK-IT WP2
devin.burke@desy.de