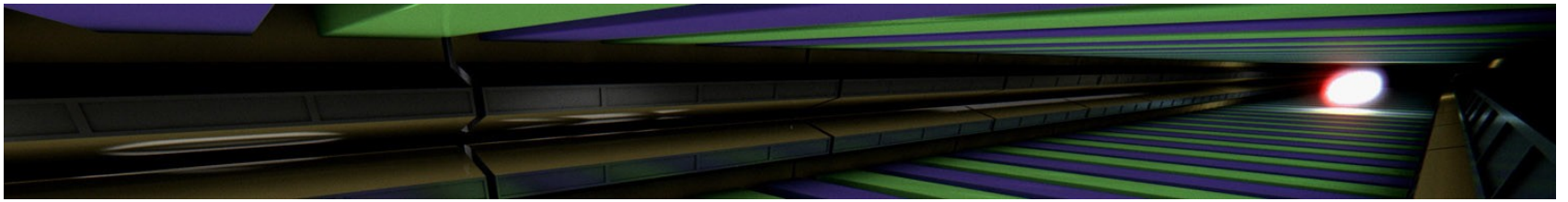
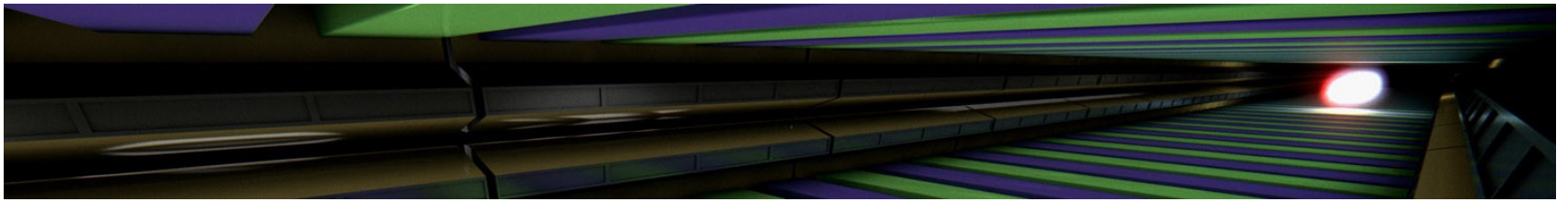


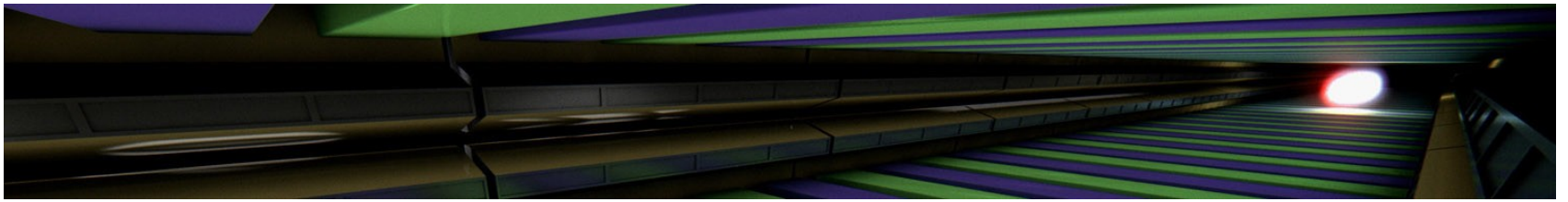
Concurrent Processing in Karabo

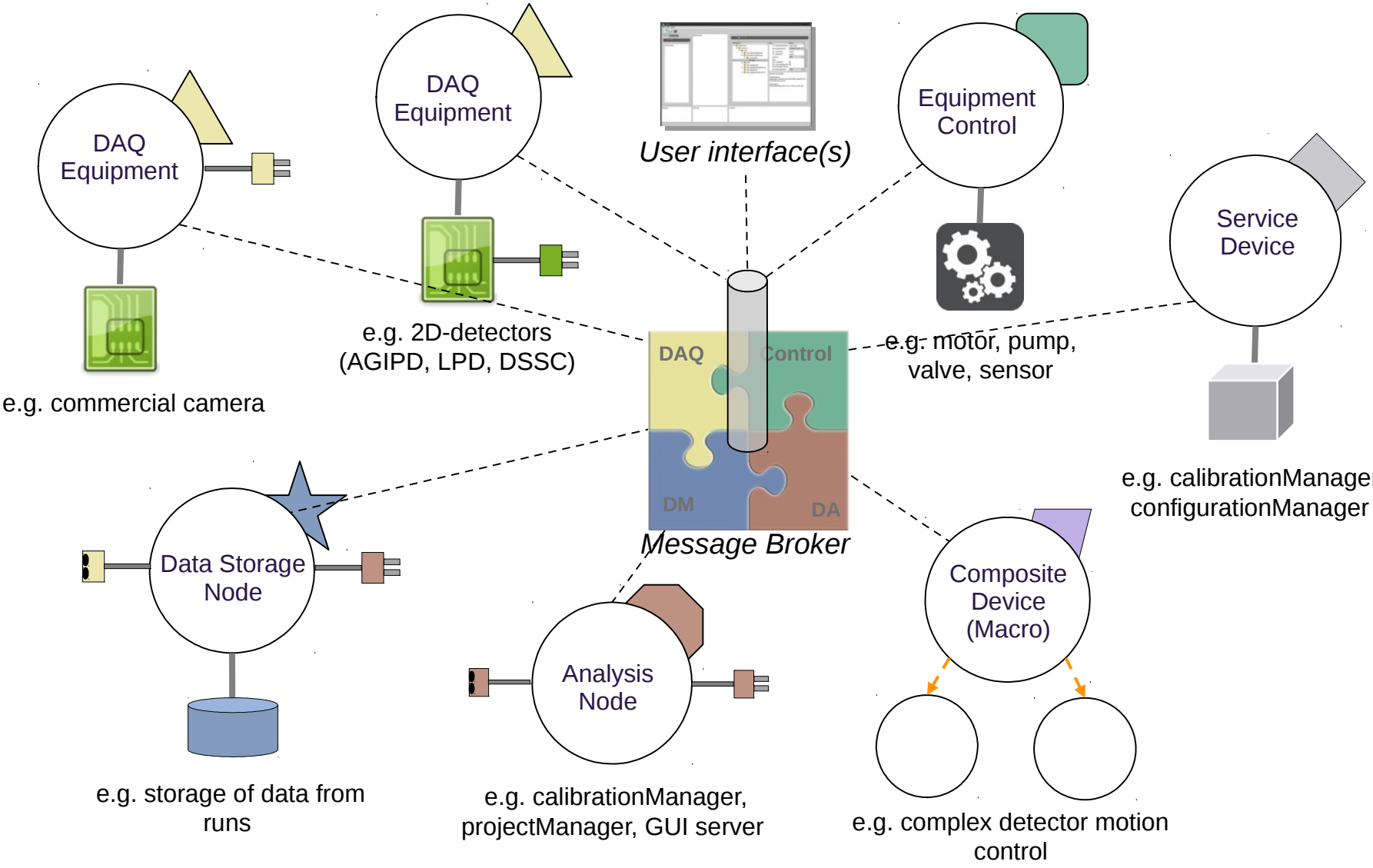
John Wiggins – XFEL CAS

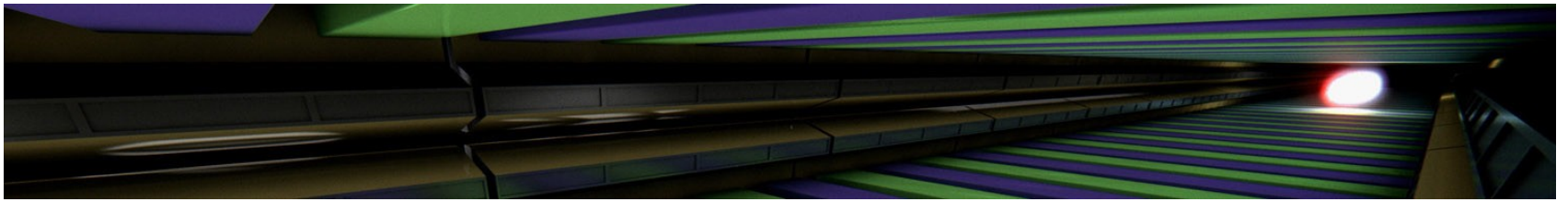


Karabo from 10,000ft (~3km)

- Devices and a Message Broker
- Communicating with Hashes, described by Schemata
- Device: A Karabo program running on a Server
- Message Broker: A central communication hub for Devices
- Hash: A container for data + metadata
- Schema: A way to describe the contents of a Hash

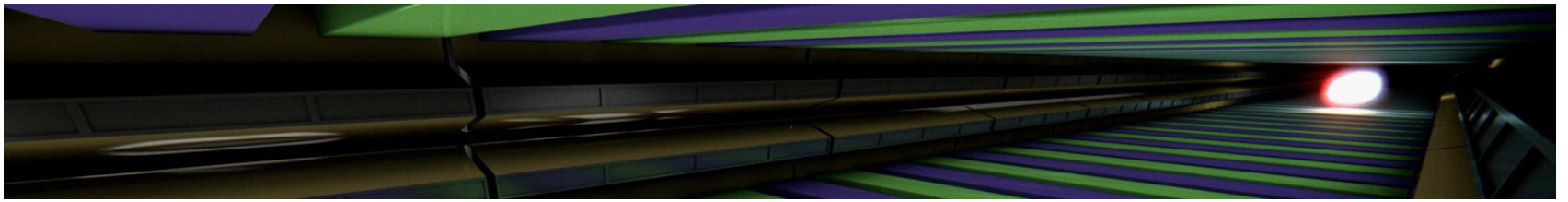



DAQ Equipment
DAQ Equipment
User interface(s)
Equipment Control
Service Device
e.g. commercial camera
e.g. 2D-detectors (AGIPD, LPD, DSSC)
e.g. motor, pump, valve, sensor
e.g. calibrationManager, configurationManager
DAQ
Control
DM
DA
Message Broker
Data Storage Node
Analysis Node
Composite Device (Macro)
e.g. storage of data from runs
e.g. calibrationManager, projectManager, GUI server
e.g. complex detector motion control



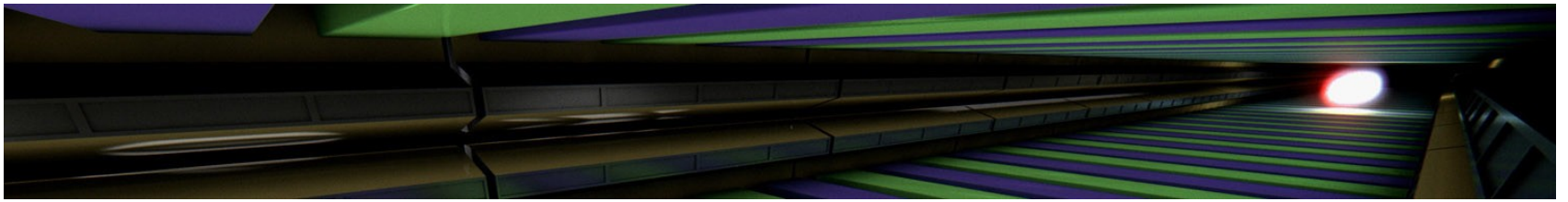
Karabo Device

- A Karabo Device is a program running on a Device Server and communicating with other Karabo Devices
- Some possible Device types:
  - Hardware control
  - DAQ
  - Services (project manager, broker monitor, …)
  - Processing



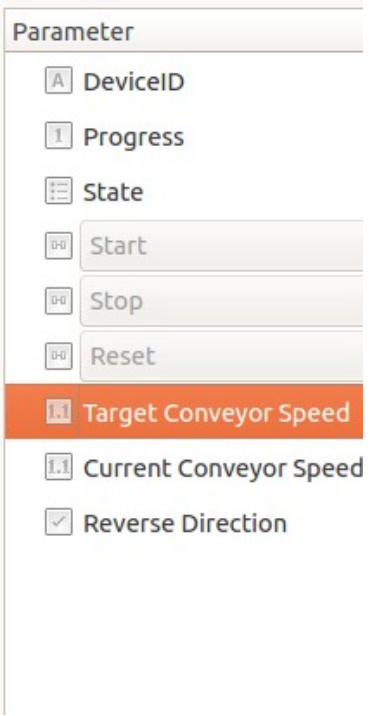
Karabo Hash

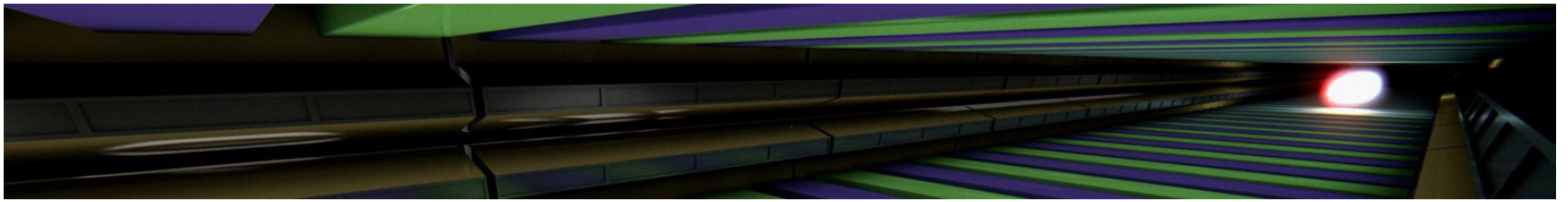
- The fundamental data structure of Karabo
- String-key, any-value associative container
- Keeps insertion order (iteration possible), hash performance for random lookup
- Provide (string-key, any-value) attributes per hash-key
- Fully recursive structure (i.e. Hashes of Hashes)
- Serialization: XML, Binary, HDF5
- Usage: configuration, device-state cache, message protocol, etc.



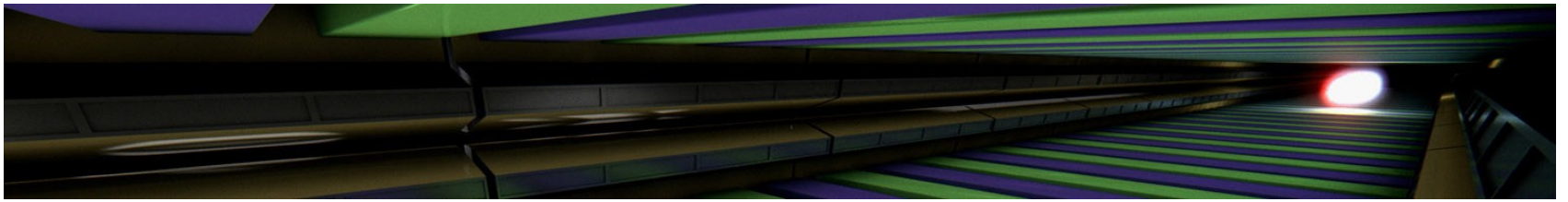
Karabo Schema

- A Schema describes the data contained in a Hash
- Analogous to an XSD document for an XML file
- In Karabo, every Device is described by a Schema:



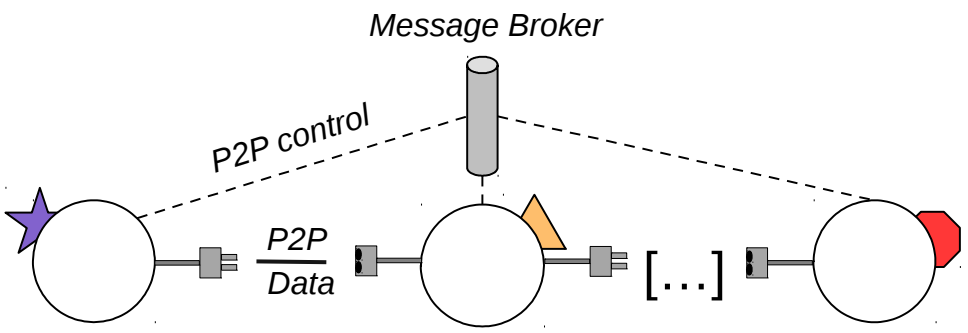


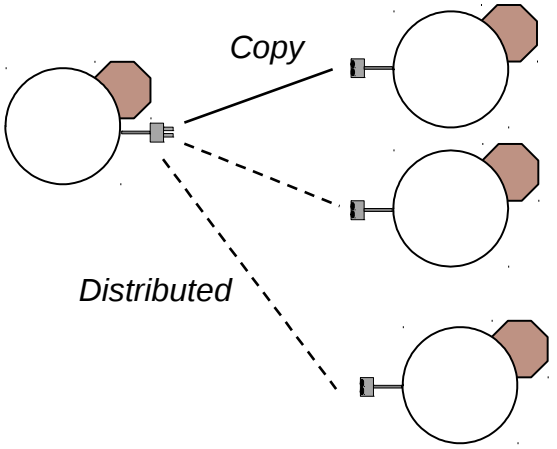
Karabo Point to Point

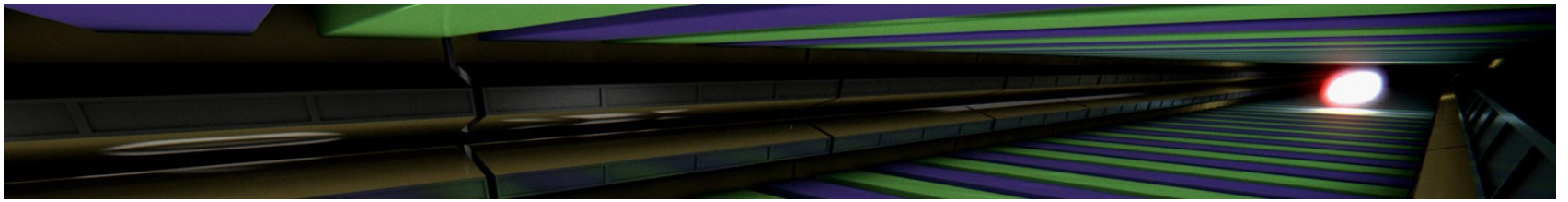


Karabo Point to Point from 5000ft

- Point-to-Point => P2P
- Devices should be directly connected when sharing large data.
  - => Don't burden the Broker
- Enables building of data flow networks with the Karabo GUI
  - Doesn't have to be a DAG; Cycles are allowed!

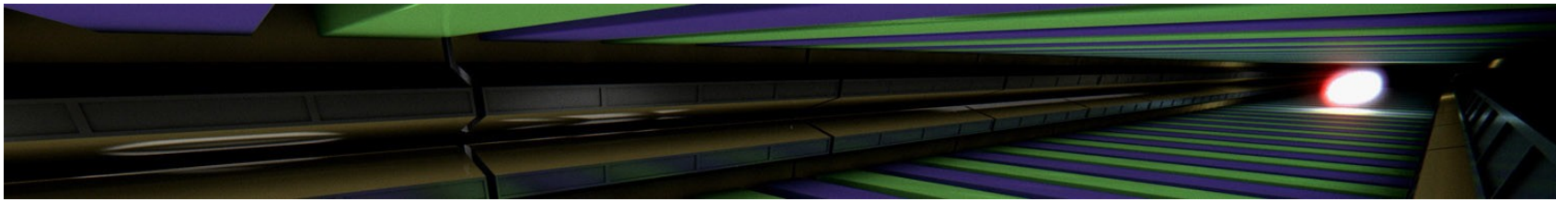




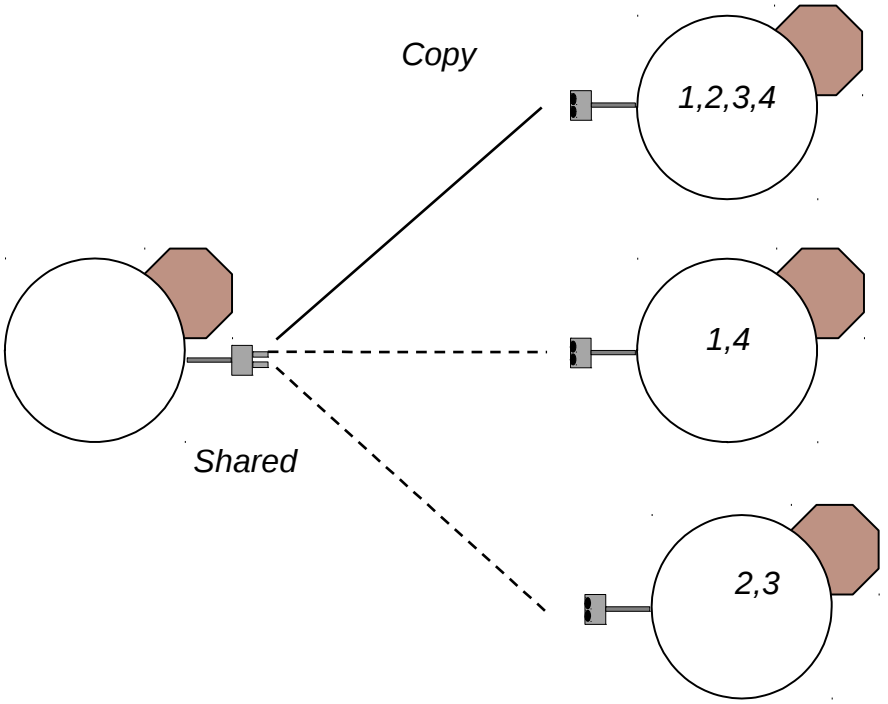


P2P Channels

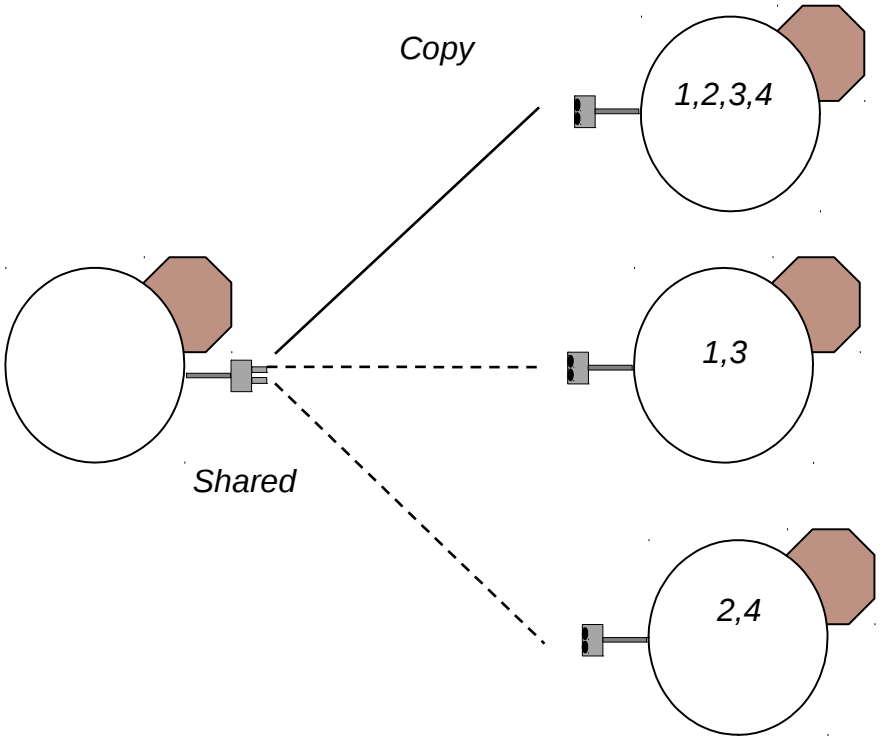
- A P2P channel is a TCP connection between Devices
- Described by a Schema on each end (input & output)
  - => Only one type of data structure per channel!
  - High-level types (Image, NDArray) are available
- Both Fan-in and Fan-out connections are allowed
- Policies determine how data tokens are passed between endpoints



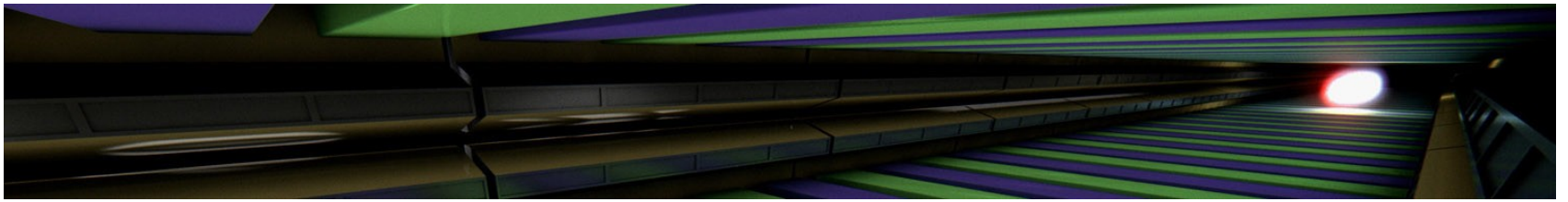
P2P Channel Policies



Load-balanced Distribution



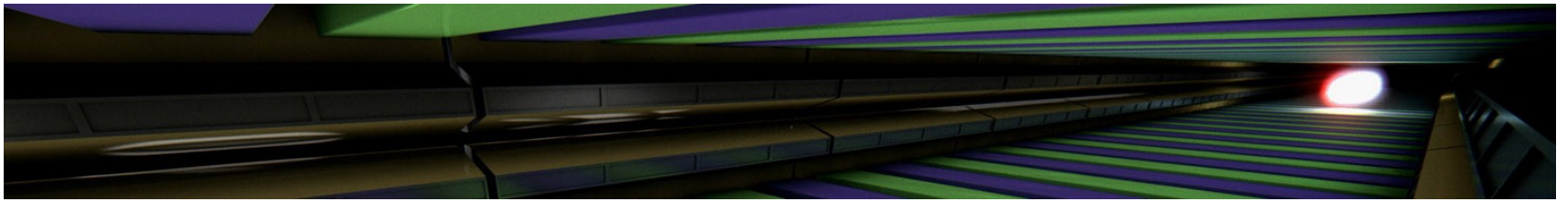
Round-robin Distribution



P2P Channel Policies

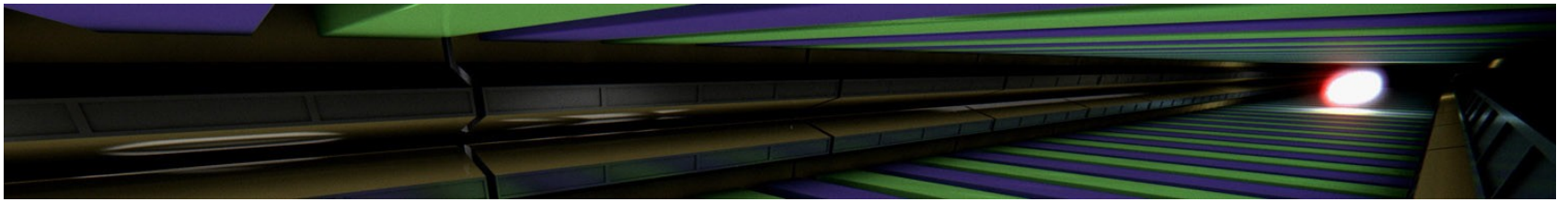
- Input Channels
  - Distribution mode: Copy, Shared
    - Copy: Every token; Shared: Determined by output distribution mode
  - Slow Input (“onSlowness”): drop, queue, throw, wait
    - Used when the distribution mode is “Copy”

- Output Channels
  - Distribution mode: load-balanced, round-robin
    - load-balanced: Send to next ready shared input
    - round-robin: Send to next shared input
  - Slow Input (“noInputShared”): drop, queue, throw, wait
    - Determines how inputs with “Shared” distribution mode are treated



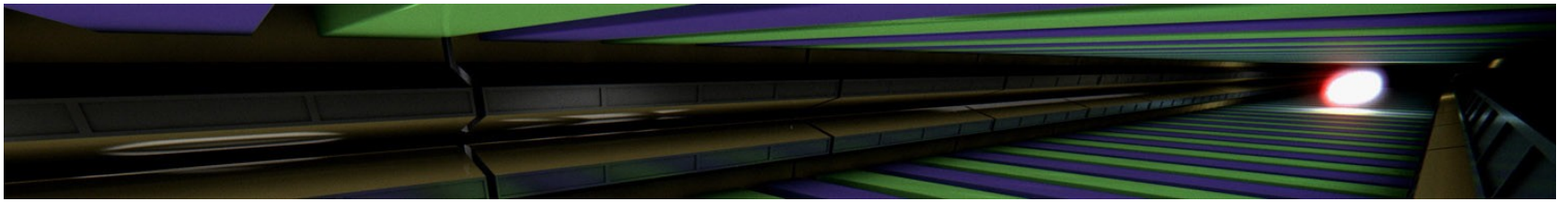
P2P Channel Policies Cont.

- Drop: Drop the data
- Wait: Wait until the receiving channel is available
- Queue: Add data to the receiving channel's queue
- Throw: Throw an exception on the OutputChannel Device



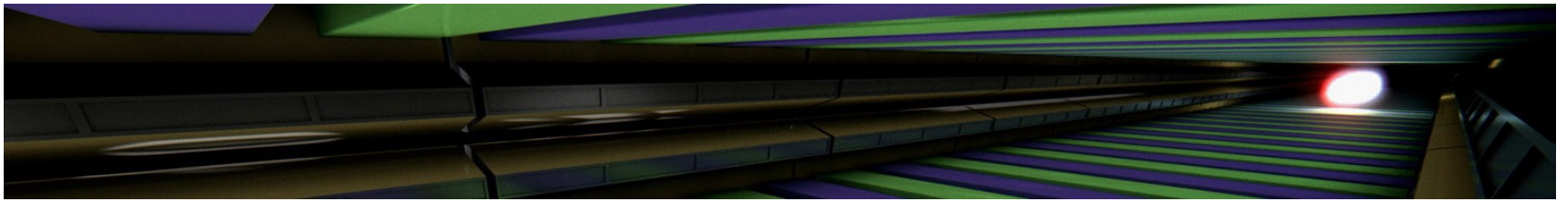
Other Parallel Processing Techniques

- IPython parallel usage will be described by Steffen
- GPU processing will be available at XFEL
- MPI can (theoretically) be mixed with P2P.
  - There isn't a pre-made solution for starting the MPI job from a device.
  - We foresee the master node being a Karabo device and other nodes not connected to message broker.



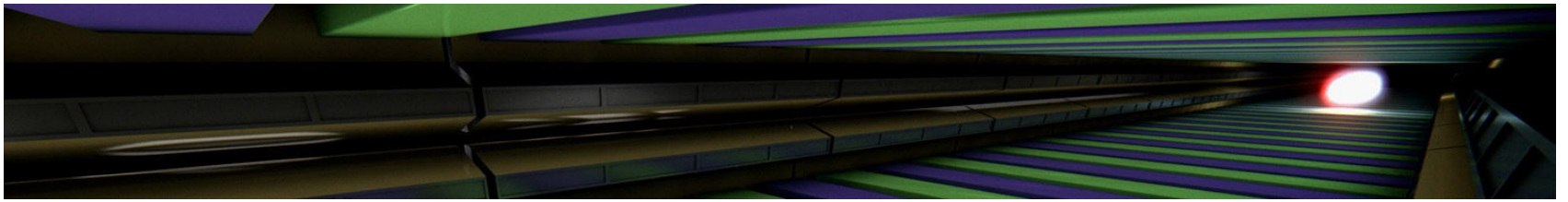
Conclusion

- Data processing will focus on the Python language
- Karabo will provide some means for distributed, parallel data pipelines
- Other existing parallelization techniques will be supported, especially if they stay thread or process parallel and do not involve external (networked) computers.
- For offline computing, direct access to calibrated HDF5 files will be granted



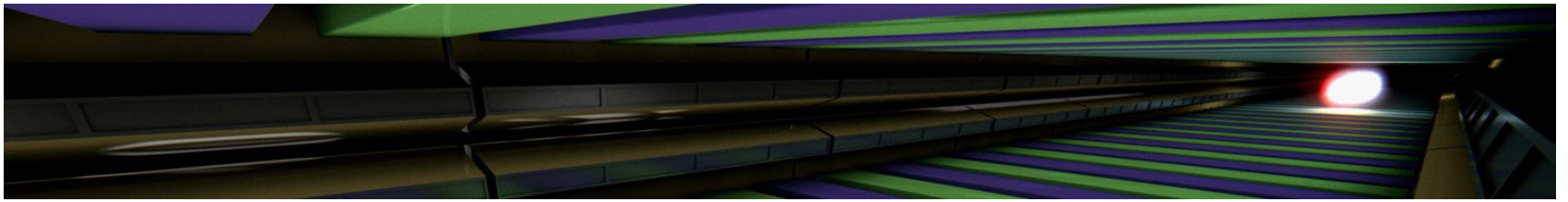
Hash Example

```
from karabo.api_1 import Hash

h = Hash()
h["some_bool"] = True
h["an_int"] = 4
h["bad_pi_approx"] = 3.14

h["some_bool", "magic_attr"] = "this can be anything"
h["an_int", "max_value"] = 4
h["bad_pi_approx", "precision"] = 2

bool_attrs = h['some_bool', ...]
print(bool_attrs['magic_attr'])  # Prints out: this can be anything
```



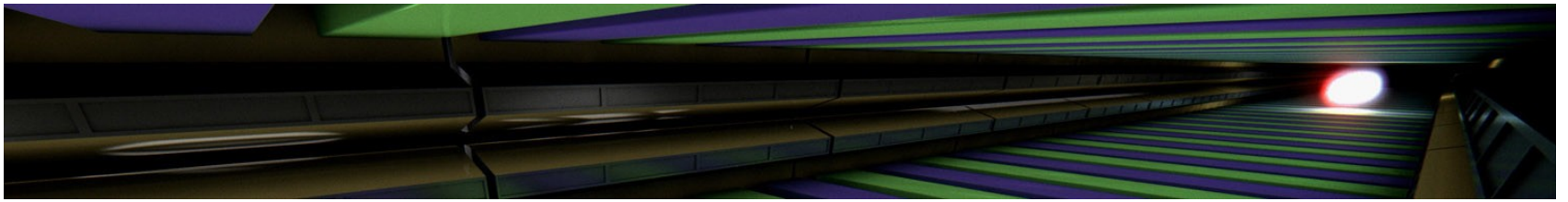
Schema Example

```
from karabo.api_1 import (
    Schema, INT32_ELEMENT, STRING_ELEMENT, VECTOR_INT32_ELEMENT
)


# … When defining the Schema for a Device
data = Schema()
d = INT32_ELEMENT(data).key("dataId")
d.readOnly().commit()

d = STRING_ELEMENT(data).key("flow")
d.readOnly().commit()

d = NDARRAY_ELEMENT(data).key("data")
d.readOnly().commit()
```



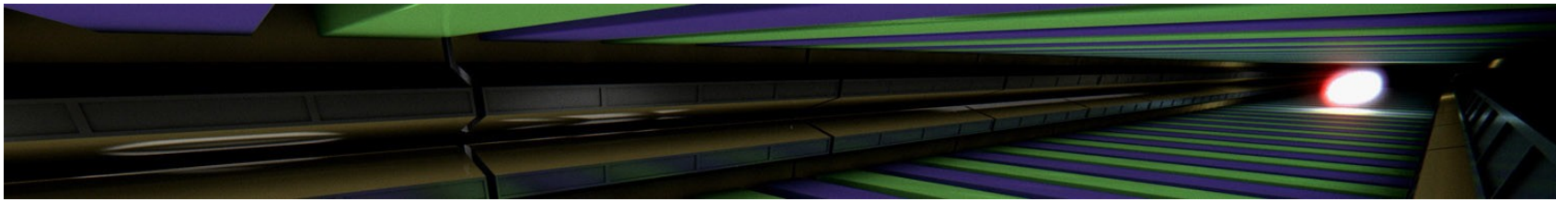
P2P Example Device [Output]

```
class OutputDevice(Device):

    @staticmethod
    def expectedParameters(expected):

        data = Schema()

        d = INT32_ELEMENT(data).key("dataId")
        d.readOnly().commit()

        d = NDARRAY_ELEMENT(data).key("data")
        d.readOnly().commit()

        e = SLOT_ELEMENT(expected).key("write")
        e.displayedName("Write").description("Write some data")
        e.allowedStates("Idle, Finished").commit()

        e = OUTPUT_CHANNEL(expected).key("output")
        e.displayedName("Output")
        e.dataSchema(data).commit()
```



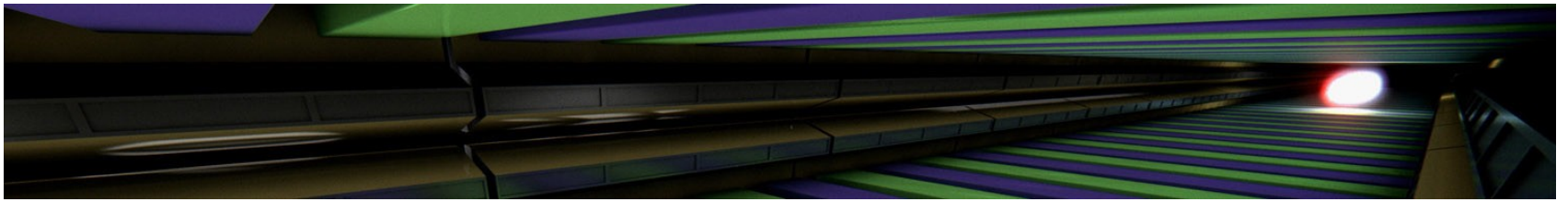
P2P Example Device [Output] cont.

```
def write(self):

    self.updateState("Writing")

    while True:
        data = Data()
        data.set('dataId', self.currentDataId)
        data.set('data', np.ones(1000))
        self.writeChannel("output", data)

        if self.someStopCondition:
            break

    self.signalEndOfStream("output")

    self.updateState("Finished")
```



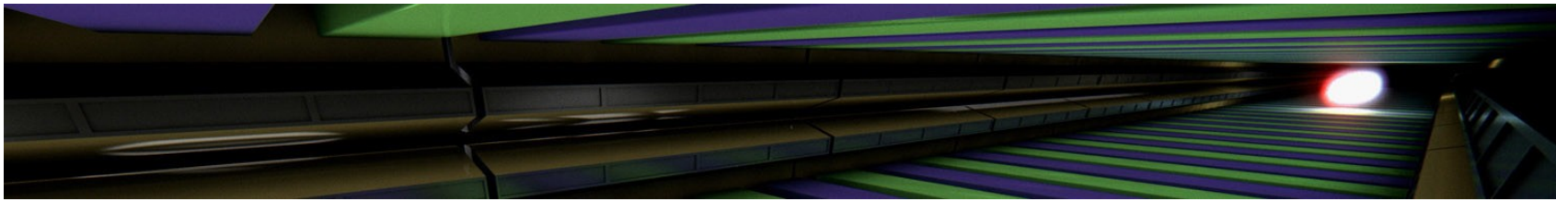
P2P Example Device [Input]

```
class InputDevice(Device):

    @staticmethod
    def expectedParameters(expected):

        data = Schema()

        d = INT32_ELEMENT(data).key("dataId")
        d.readOnly().commit()

        d = NDARRAY_ELEMENT(data).key("data")
        d.readOnly().commit()

        e = INPUT_CHANNEL(expected).key("input")
        e.displayedName("Input")
        e.dataSchema(data).commit()
```



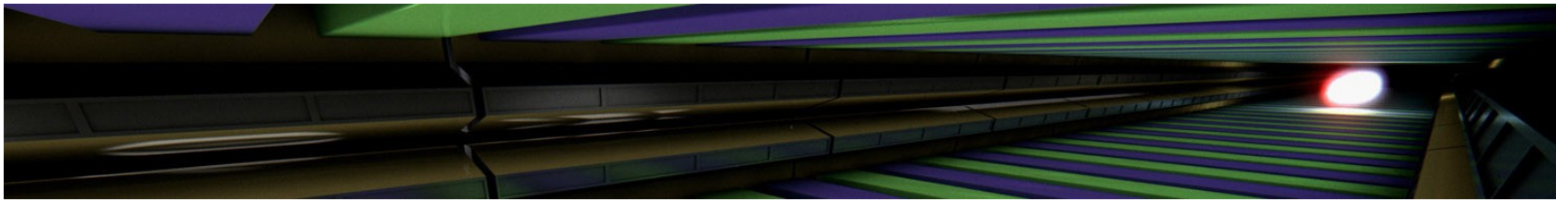
P2P Example Device [Input] cont.

```
def __init__(self, configuration):
    super(InputDevice, self).__init__(configuration)
    self.registerInitialFunction(self.initialization)

def initialization(self):
    self.KARABO_ON_DATA("input", self.onData)
    self.KARABO_ON_EOS("input", self.onEndOfStream)
    self.updateState("Idle")
```



P2P Example Device [Input] cont.

```
def onData(self, data):
    self.updateState("Computing")

    print("Received data with ID:", data["dataId"])
    array = data["data"]
    print("Data average value is:", array.mean())

    self.updateState("WaitingIO")

def onEndOfStream(self, input):
    print("Input <{0}> finished".format(input))
    self.updateState("Finished")
```



Online & Offline Processing

- The type of processing being done – online or offline – can inform which policies are used on a P2P connection
- Online processing:
  - Calibration
  - Data QC during experiment
- Offline processing:
  - Analysis of experimental data at XFEL