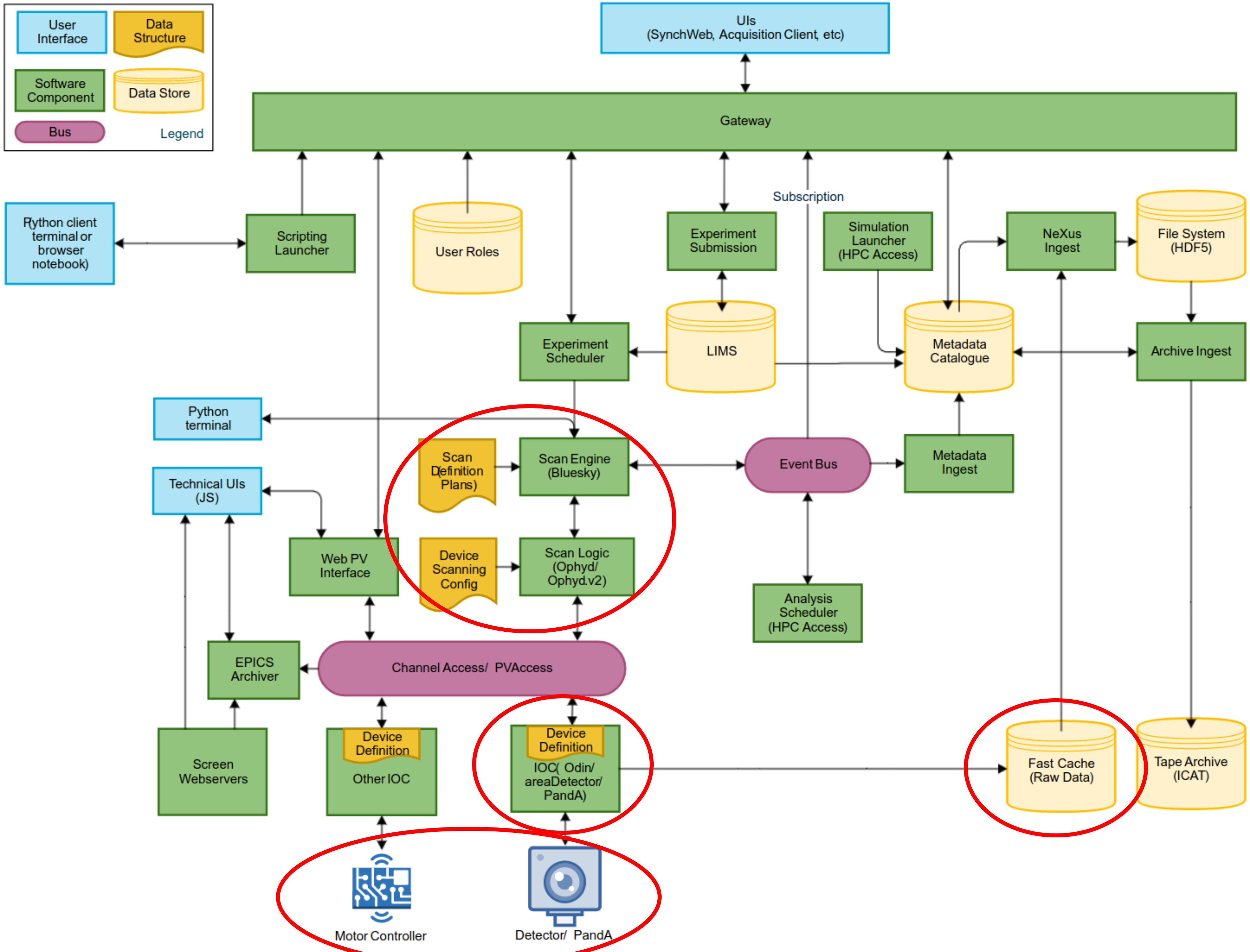
DLS approach to continuous scans

Tom Cobb

Beamline Controls Core Team Lead

Contents

- D-II facility upgrade
- Data writing
- PandA developments
- CS agnostic Device Server
- Bluesky/Ophyd
- Specific Strategies

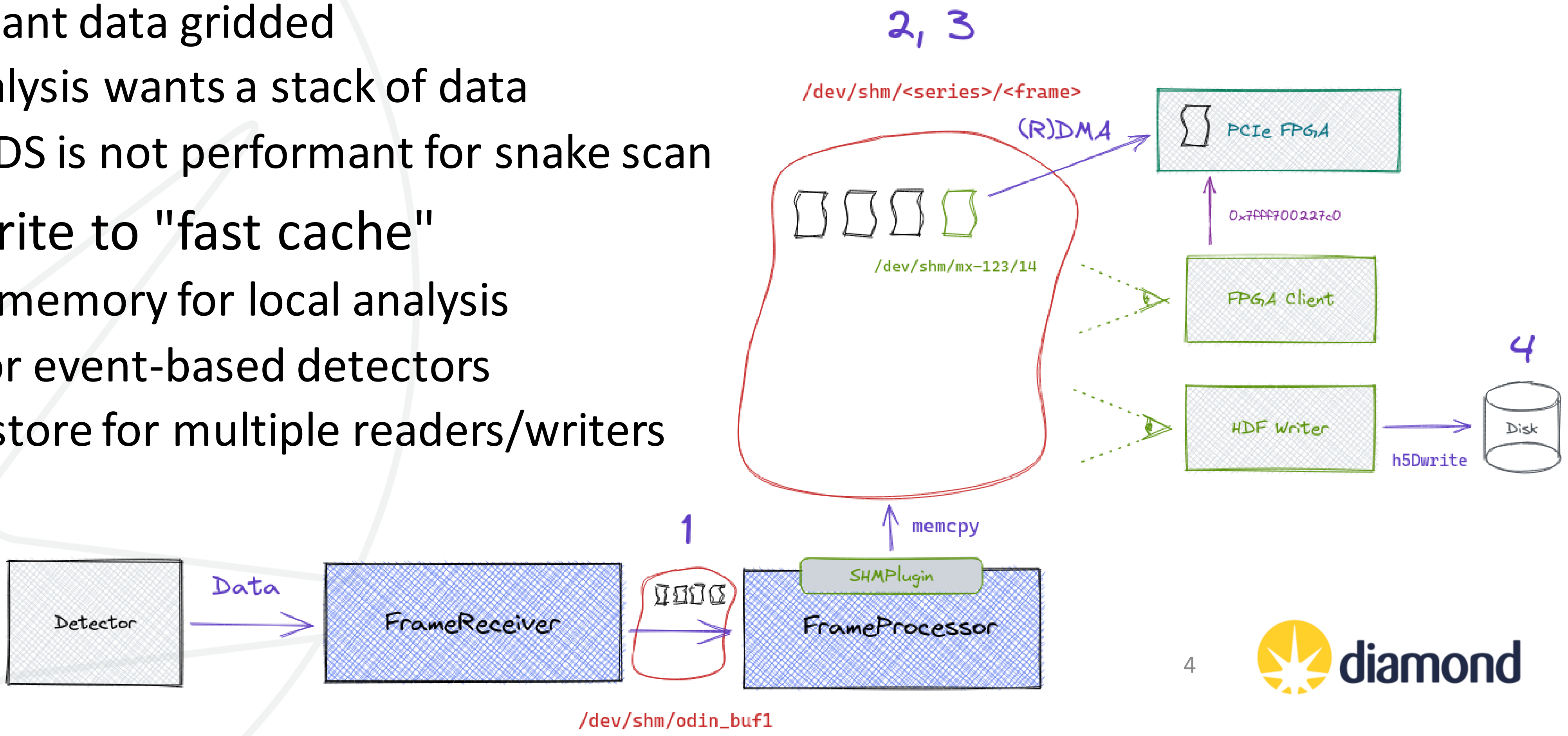


D-II facility upgrade

- 18-month dark period
- Starting end of 2027
- Opportunity to upgrade software architecture
- New beamlines will use new architecture from start
- Existing beamline will gradually be updated to new architecture

Data writing

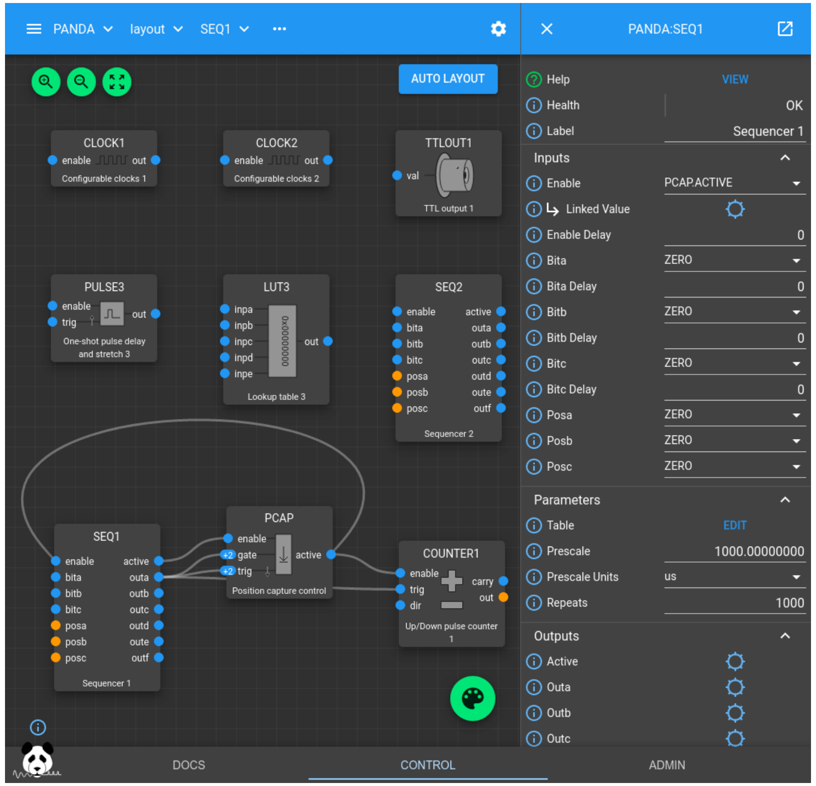
- Tension between final format and in-flight format
  - Users want data gridded
  - Live analysis wants a stack of data
  - HDF5 VDS is not performant for snake scan
- Instead write to "fast cache"
  - Shared memory for local analysis
  - Kafka for event-based detectors
  - Object store for multiple readers/writers

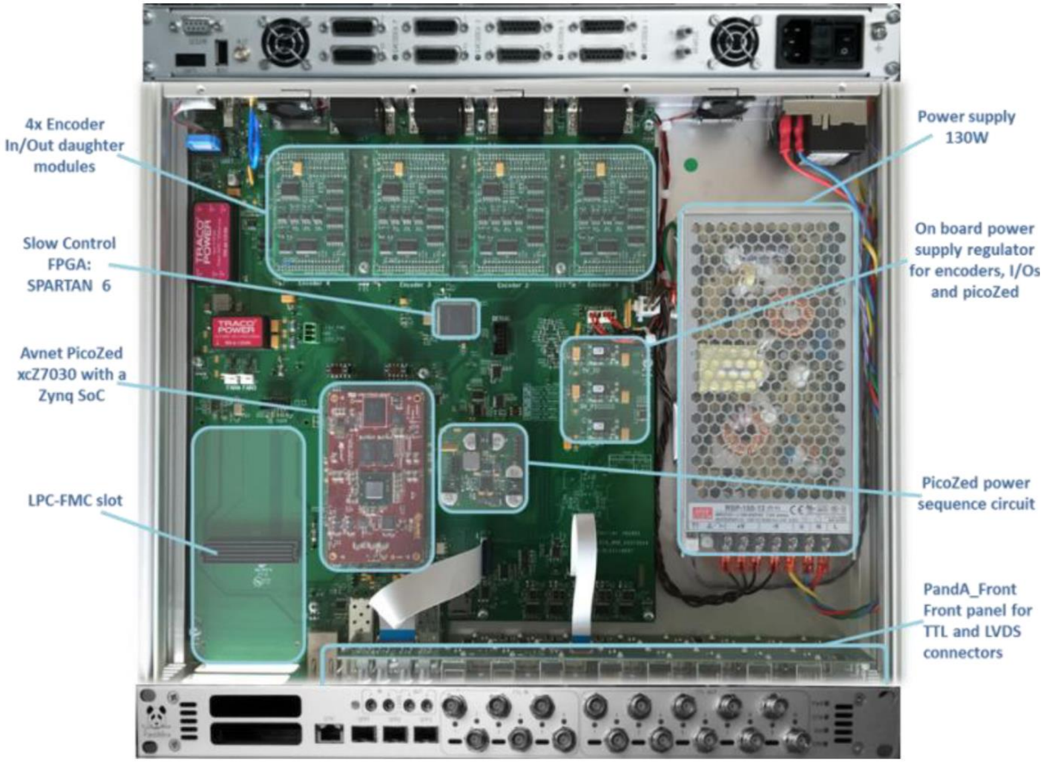


diamond

What is PandABox?

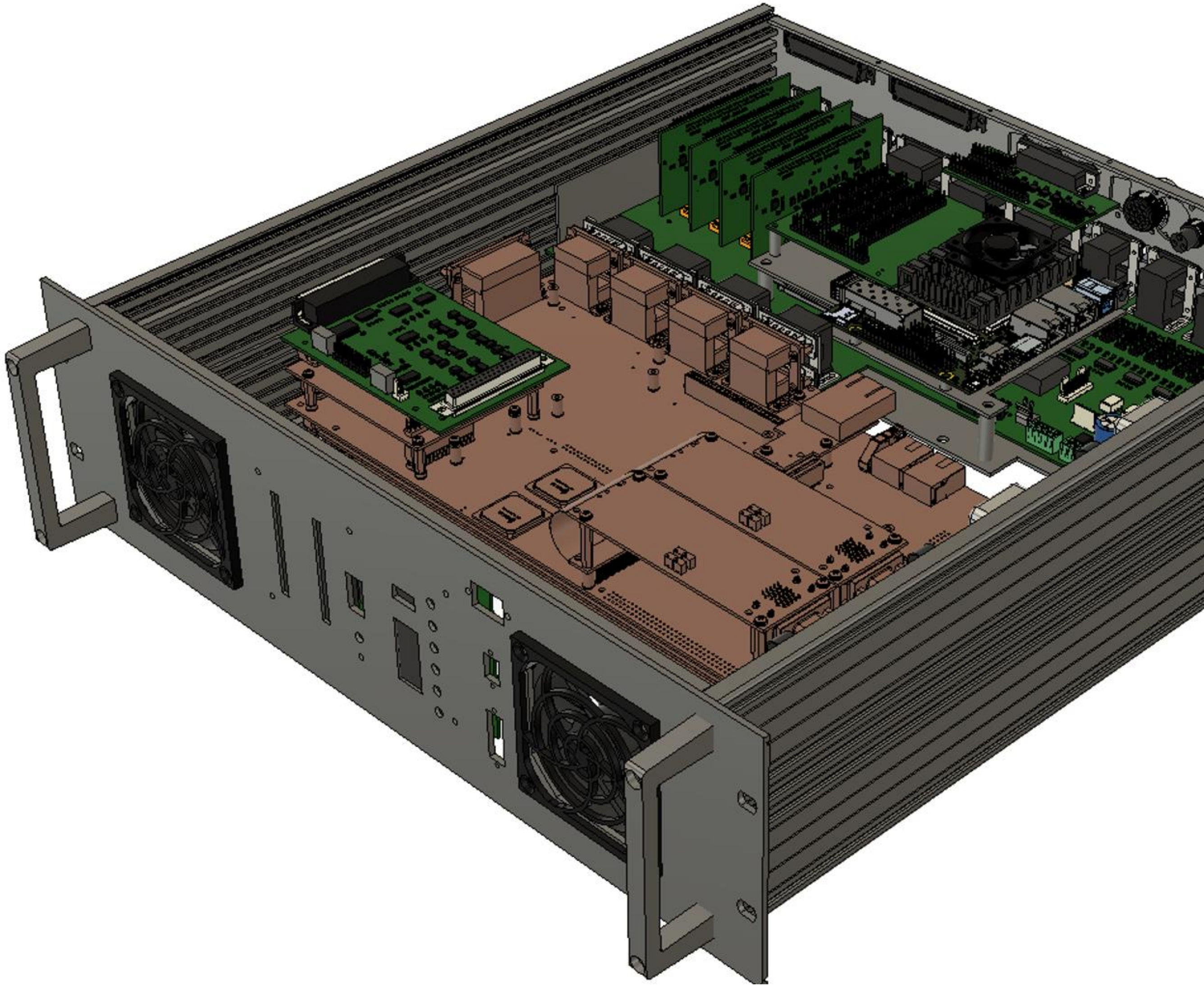
- Position and Acquisition box
- FPGA with rewirable, configurable processing blocks like pulse generator, position compare, position capture
- CPU with TCP server for control system integration and web server for easy setup
- Currently collaboration between DLS, SOLEIL, MAX-IV





PandA Developments

- PandABrick
  - PandA and PowerBrick
  - In one box
  - Still 2 systems
  - Prototype delivered
- PandASwitch
  - PandA firmware
  - Running on White Rabbit Switch V4
  - Prototype in 2024 (hopefully)
- PandABox Ultra?
  - Driven by SOLEIL and LEAPS

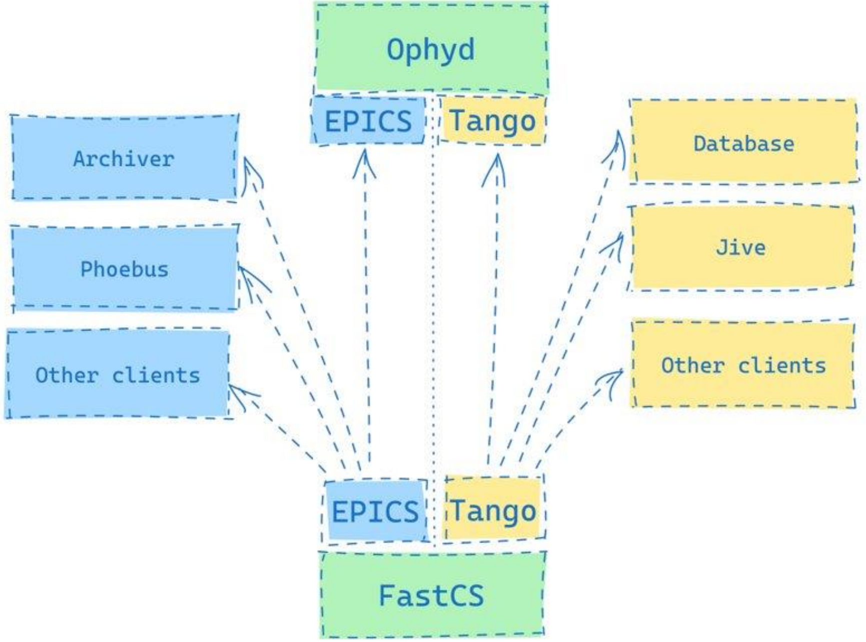


FastCS: CS Agnostic Device Server

- Python framework for making control system devices, fast
- Similar in concept to FastAPI, remove boilerplate, support type hints
- Plugins for EPICS, Tango and (maybe in the future) GraphQL
- EPICS plugin makes pythonSoftIOC + screens + ophyd interface

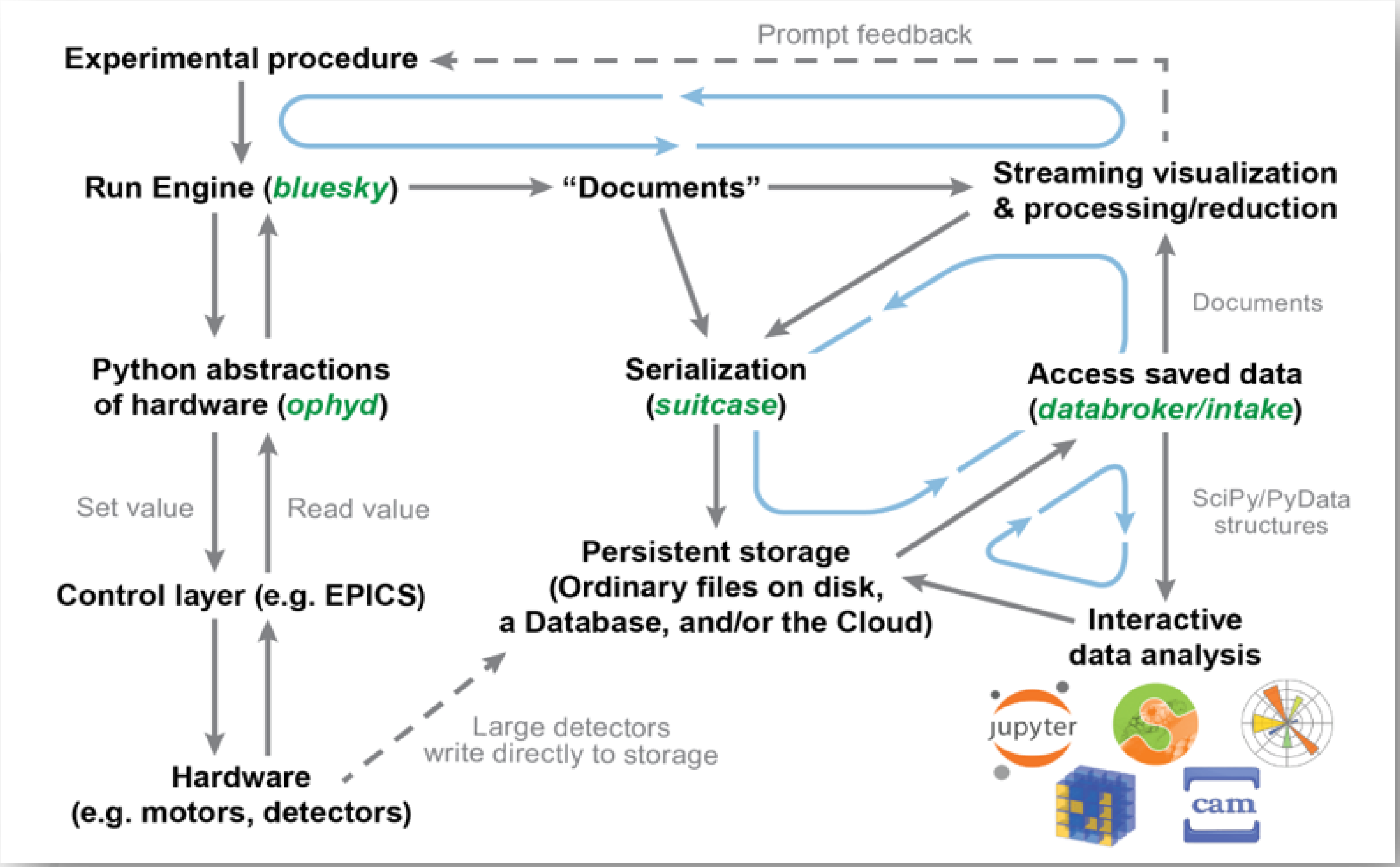

```python
@dataclass
class TempHandler:
    cmd: str
    async def put(self, controller, value):
        await controller.conn.send_command(
            f"{self.cmd}={value}\r\n")

class TempController(Controller):
    start = AttrRW(Int(), handler=TempHandler("S"))
    end = AttrRW(Int(), handler=TempHandler("E"))
    current = AttrR(Float(prec=3), handler=TempHandler("T"))
    enabled = AttrRW(Bool("Off", "On"), handler=TempHandler("N"))
```

```python
from typing import Union

from fastapi import FastAPI

app = FastAPI()


@app.get("/")
def read_root():
    return {"Hello": "World"}


@app.get("/items/{item_id}")
def read_item(item_id: int, q: Union[str, None] = None):
    return {"item_id": item_id, "q": q}
```



Bluesky

- Lightweight clean architecture
- Python3 based
- Event model for data
- Ecosystem of complementary modules
- Proven at NSLS-II
- Adopted by several other facilities
- Our prototypes were leading to a similar architecture
- Part adoption



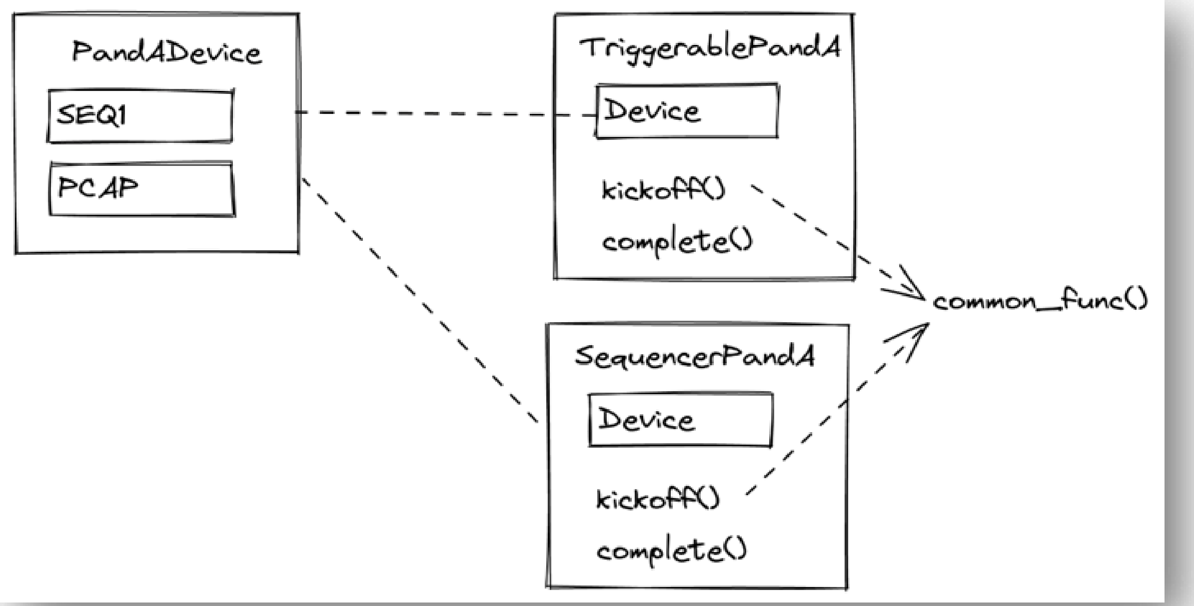
diamond

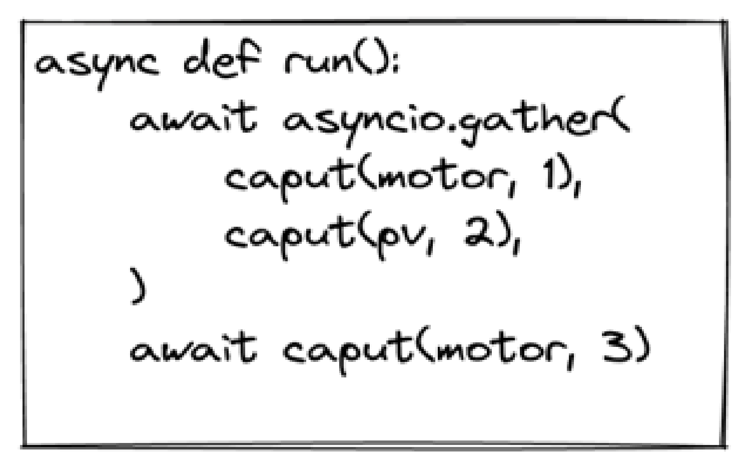
Focus on protocols

- Devices support certain "verbs"
  - read(), trigger(), set()…
- User writes a plan that uses building blocks to send messages to Bluesky
  - move(my_device, 1)
- Bluesky interprets the messages and calls the right verbs on Devices
- Data is emitted as Events, which contain data or references to data

Ophyd-async

- PandA is single Device which can be used in many ways
  - Ophyd doesn't separate interface and logic
  - Big inheritance hierarchy
- Common logic in flyscans
  - Ophyd does this via multiple inheritance
  - Lack of composition impairs readability
- Async functions improve legibility
  - Ophyd uses threads and callbacks
- Writing ophyd-async in collaboration with NSLS-II to solve





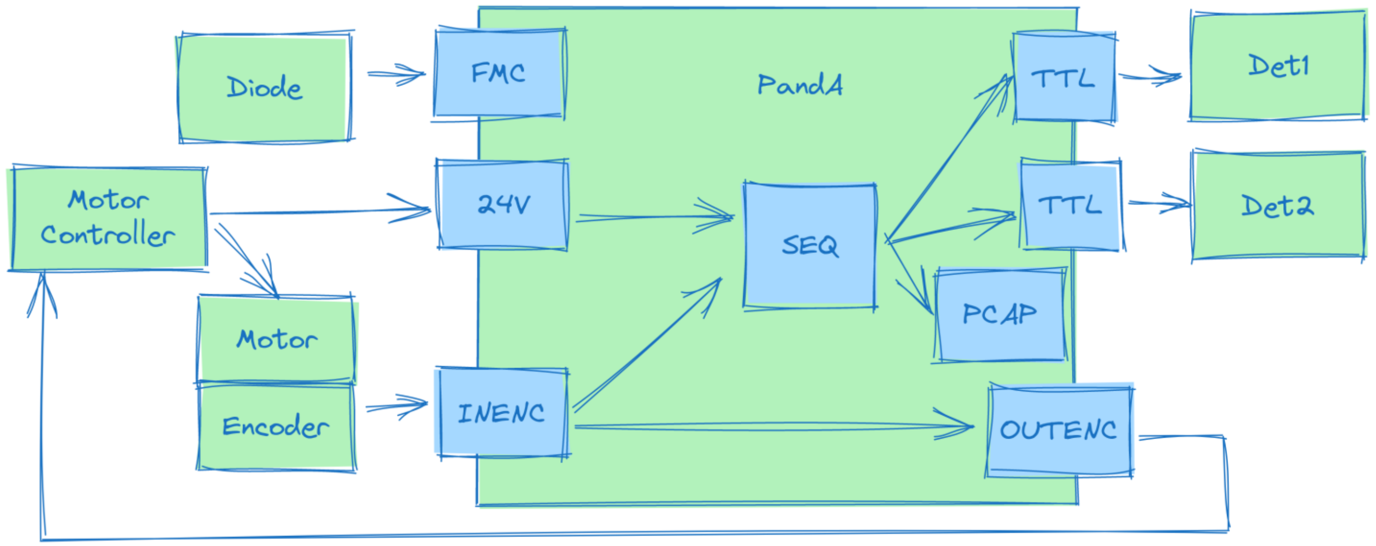
diamond

Specific strategies

- Push synchronization down to the hardware wherever possible
  - Co-ordinate systems in the motor controller
  - Position compare and time-based triggering in the PandA
  - Machine events into PandA
- Will cover the specifics in the requested format

Synchronization descriptions

- Bring all signals to PandA
- Manufacture trigger signals
- Fan out to detectors and internally captured signals
- Bluesky changes internal wiring of PandA based on saved configs
- PandA now has MRF EVR support



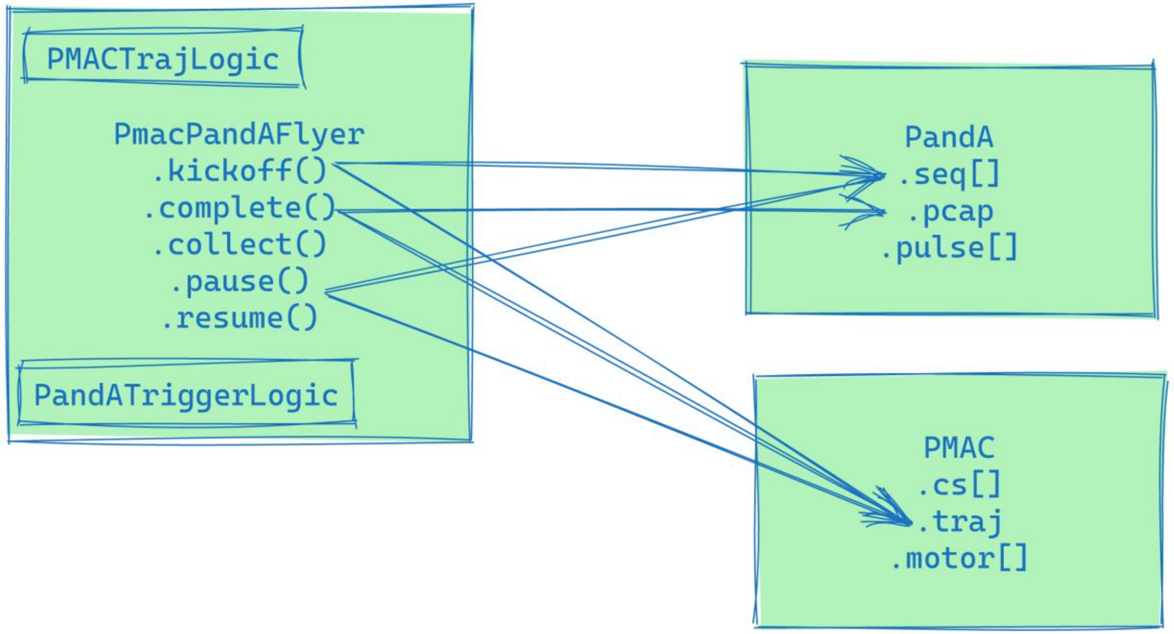
Trajectory control

- EPICS Trajectory control for Delta Tau Turbo/Power PMAC
- Points can be appended while the scan is in progress
- Co-ordinate systems implemented on the controller as kinematics
- Bluesky interfaces to co-ordinate system motor
- Post processing to calculate co-ordinate system positions from captured raw motor positions

diamond

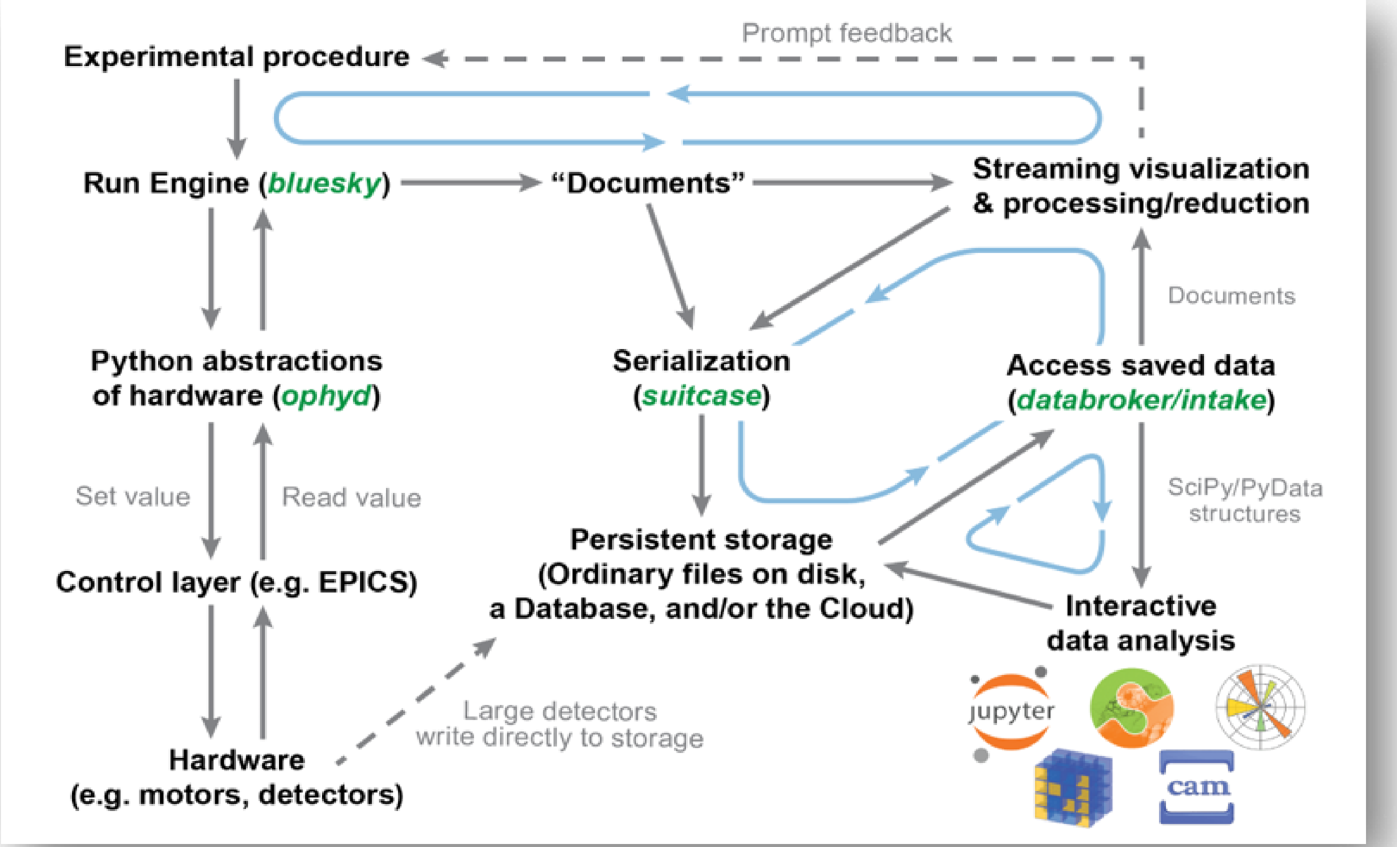
Multiple controllers

- Bluesky has the concept of Flyable devices which support kickoff and complete verbs
- We intend to model a flyscan as a single Flyable so pause/resume can be sequenced correctly
- We will encapsulate the PandA and PMAC specific logic in internal classes and call them from the top level Flyable



High speed scans

- Stream data to one of:
  - GPFS using SWMR (current)
  - Shared memory on a single machine (prototyped)
  - Object store (investigating)
  - Kafka (investigating)
- Pass references to this data via Bluesky event model



diamond

Other issues

- When to record encoder position -> PandA averages over a gate
- Latency time -> Detector reports deadtime
- Measurement group -> Model them as a single custom Flyer
- Decoupling channels -> Use subscriptions for slow data
- Add timeout to motion -> Plan to pause and resume
- Add early timeout -> Monitor for stall and end early

Conclusion

- D-II facility upgrade gives software architecture upgrade opportunity
- Data writing to be more diverse
- PandA developments on different hardware platforms
- CS agnostic Device Server to aid collaboration
- Bluesky/Ophyd to run all scanning
- Pushing down to hardware wherever possible
- Thanks for listening