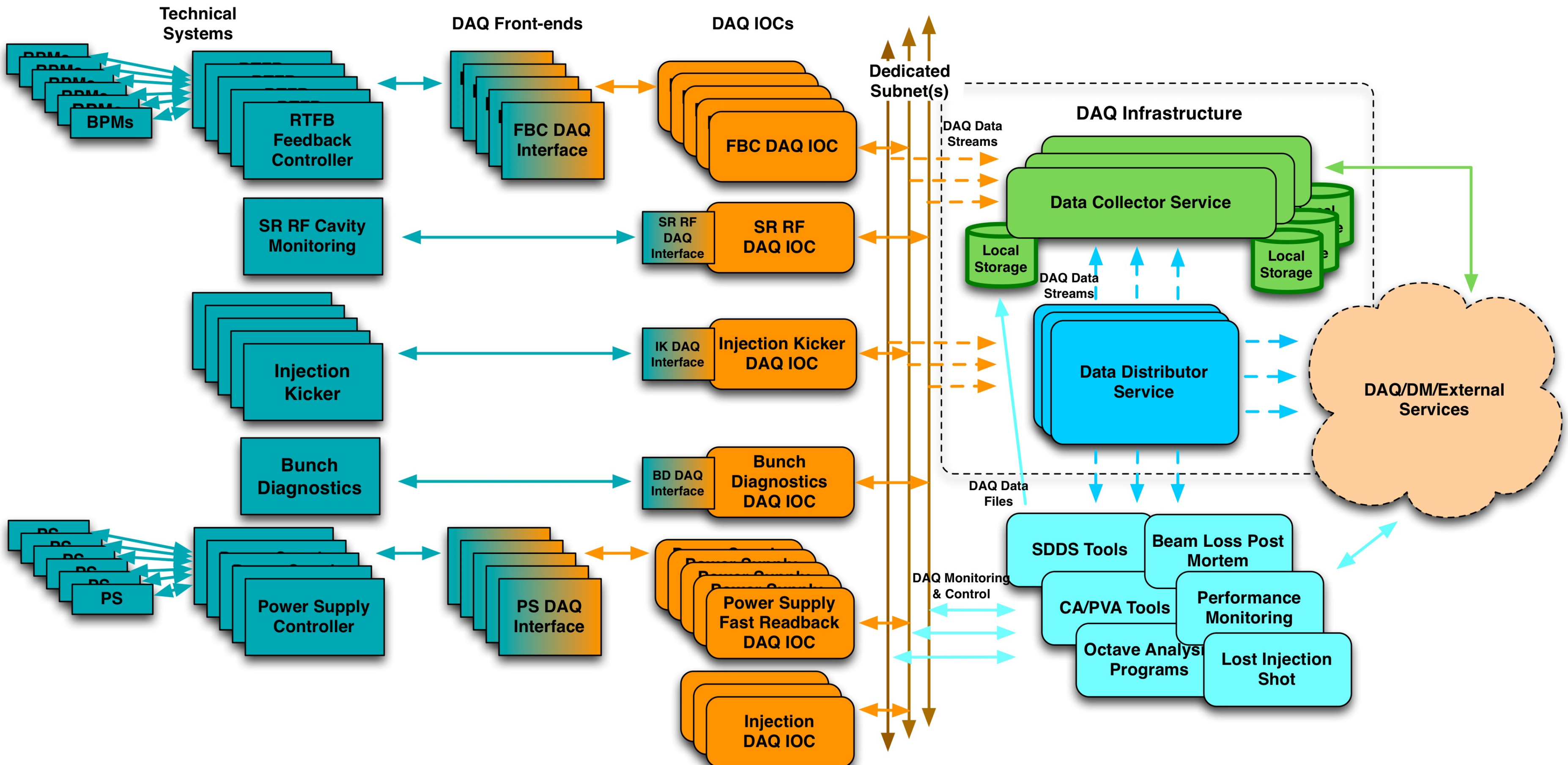
Using Area Detector As A General Purpose Processing Framework

Ned Arnold
Siniša Veseli
Argonne National Laboratory

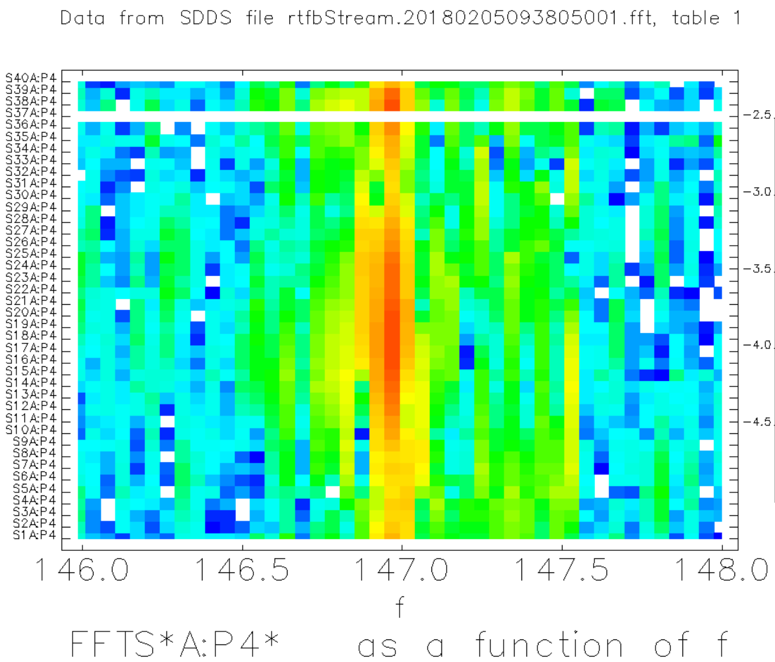
EPICS Collaboration Meeting
June 2018

APSU Data Acquisition System

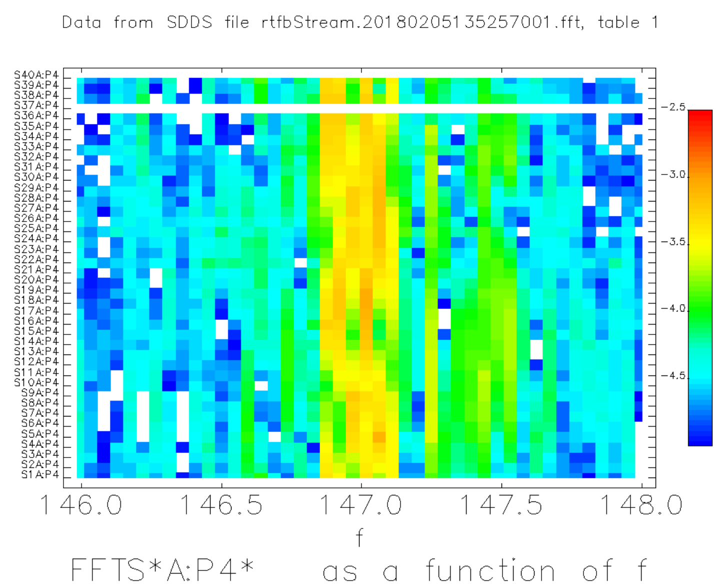


- Provides time correlated/synchronously sampled data
- Can be used for commissioning, troubleshooting, performance monitoring and early fault detection
- Separated from operational systems to allow troubleshooting during user operations
- Can acquire data from several subsystems at various sample rates
- Supports continuous/triggered acquisition, static parameters, slow (in development) and fast data
- Scalability
- Ability to route data to any number of applications

DAQ Usage Example (L. Emery)



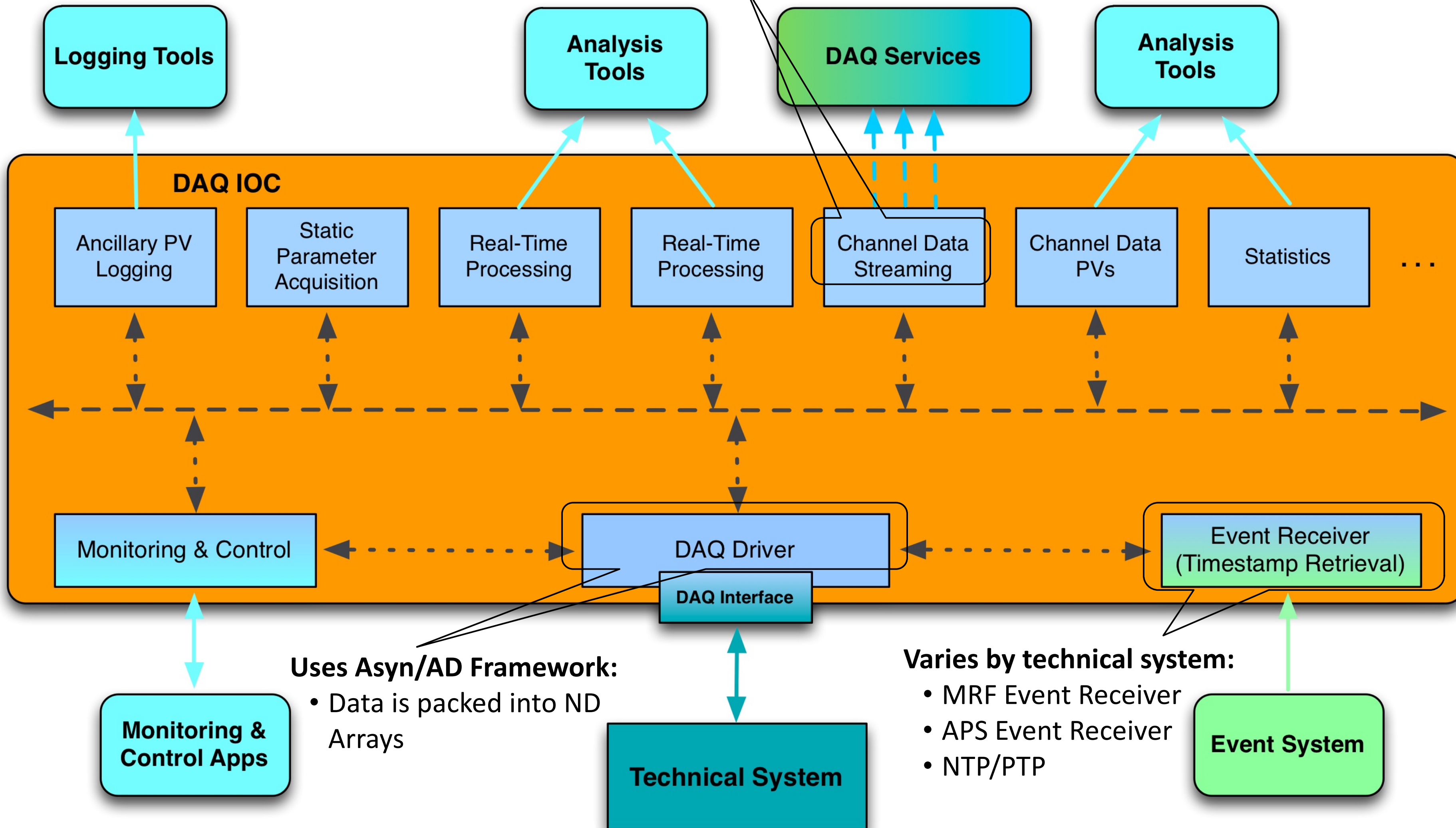
- Suppression of 147Hz vibration source in the ring using the DAQ system + post-processing with FFT
- Vacuum chamber was vibrating and introduced a Bx field; shims were inserted between poles and vacuum chamber (S37AQ3, S37AQ2)



- Identification of the nearest quadrupoles required 400 channels, 20 seconds of continuous DAQ data to get 0.5Hz precision
- This allowed separating line frequencies of 20 pumps
- Figure on the top is showing data before shimming, while the one on the right is showing results after shimming

DAQ IOC

Prototyped several options:
- Custom TCP
- EPICS V4 PVA
- AMQP (QPID)

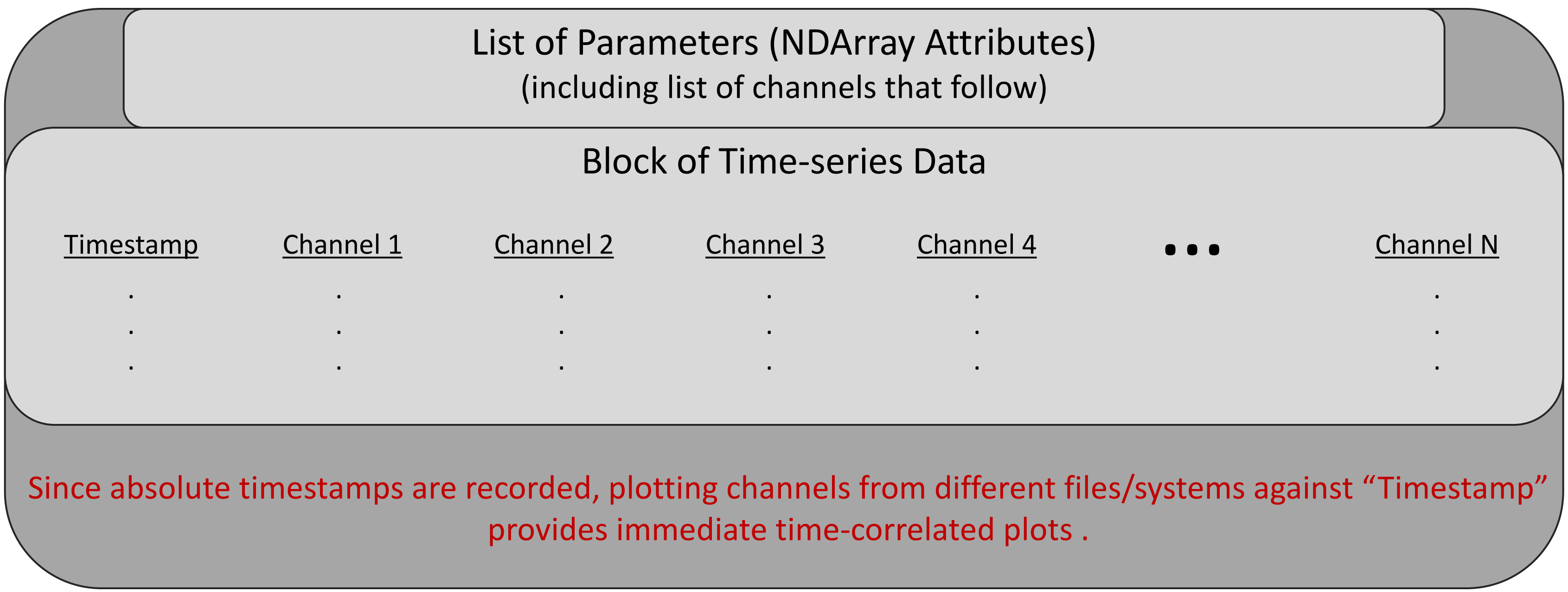


Uses Asyn/AD Framework:
- Data is packed into ND Arrays

Varies by technical system:
- MRF Event Receiver
- APS Event Receiver
- NTP/PTP

Asyn/AD Framework Usage: Initial Approach

- Use Asyn/AD Framework to collect and process data from technical subsystems
  - Driver packs channel data into ND Arrays, and passes it to real-time processing plugins
  - ND Array attributes describe the content (i.e., what data is in ND Array, how is it packed)
  - Communication plugin streams ND Arrays to remote Data Collector service where it is unpacked, stored, and (possibly) forwarded to the Data Distribution Service (Message Broker)
- DAQ Data Packet



Challenges

- How do we pack data?
  - ND Array was designed for homogeneous data; DAQ must handle multiple data types
  - Timestamps are doubles, channel data may not be
  - DAQ channels may have different data types
- How do we handle runtime configuration changes?
  - Must be able to turn on/off different channels without restarting various system components
  - Configuration changes result in ND Array packing scheme changes
- How do we provide DAQ users with easy access to individual channel data?
  - Cannot be done without custom clients that know how ND Array was packed
- How do we handle slow vs fast data?
  - Must avoid significant overhead in memory/network bandwidth
- How do we efficiently access/process channel data in real-time DAQ processing plugins?
  - Unpacking ND Array data in plugins themselves is very inefficient

Possible Solution

- Modify AD framework to pass DAQ structures through plugins; those could then be easily exposed as EPICS v4 structures over PV Access protocol

DAQ AD Core Extension

- Goals:
  1) Minimal modification of AD core code that allows us to pass arbitrary data through processing plugins (without having to pack/unpack ND arrays in plugins themselves)
  2) Backwards compatibility: no existing AD plugins need to change
  3) Ability to retrieve data from IOC via standard PVA APIs and tools like pvget
- Strategy: use NDArray/NDArrayPool as base classes for extending the AD Core functionality
- AD Core modifications: 6 lines of code in NDArray.h and about 25 lines of code in NDArrayPool.cpp:
  - New NDArrayPool class methods for management of extended ND arrays
  - Modifications to keep track of ELL node offset
- Custom DAQ Code:
  - RtfbNDArray (derived from NDArray, incorporates custom v4 structure) and RtfbNDArrayPool classes (derived from NDArrayPool, manages RtfbNDArrays)
  - Driver code uses custom pool and manipulates RtfbNDArray
  - RtfbNDPluginPva exposes RtfbNDArray via PV Access channel (plugin uses dynamic cast to convert NDArray pointer to RtfbNDArray pointer)

DAQ AD Core Extension

- Advantages:
  - Approach is completely backwards compatible (no need to change existing plugins)
  - Requires minimal modifications to AD Core
  - No loss in performance due to packing/unpacking ND Arrays in plugins
  - Custom plugins can expose data as v4 structures easily; those can be accessed using standard client tools:

```
$ pvget rtfb_ext_ndarray
rtfb_ext_ndarray
structure
    uint ArrayId 0
    double[] TimeStamp []
    float[] PosX_0 []
    float[] PosY_0 []
    float[] PosX_1 []
    float[] PosY_1 []
    …
```

- Drawbacks:
  - Custom plugins that use classes derived from NDArray must downcast (performance hit)
  - Design not quite suitable for a general purpose processing framework
  - Better solution would require AD Core refactoring/redesign, and would not be backwards compatible

Final Comments

- Current DAQ production code:
  - Uses EPICS 3.15.x, AD Core 2.5
  - Mixture of old (use NDArray) and new (use DAQ extension) style IOCs
- DAQ development will transition to EPICS7, AD Core 3.2
- PR #324 for DAQ AD Core (merged recently): will allow us to keep up with AD Core changes