Queen Mary
University of London

The ToolDAQ DAQ Framework
and its uses (ANNIE, Hyper-K, E61, etc.)

Dr Benjamin Richards (b.richards@qmul.ac.uk)

ToolDAQ

ToolDAQ is an open source DAQ Framework developed in the UK.

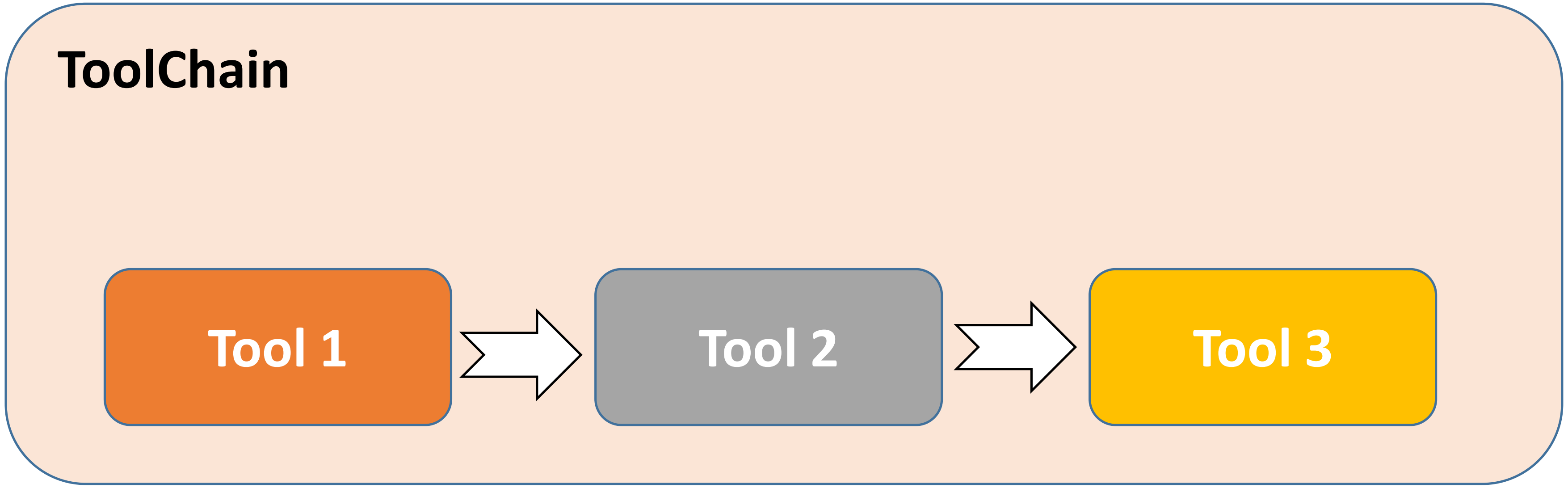
It was deigned to incorporate the best features of other DAQ whilst:

1. Being very easy and fast to develop DAQ implementations in a very modular way.
2. Including dynamic service discovery and scalable network infrastructure to allow its use on large scale experiments.

Features

- Pure C++
- Fast Development
- Very Lightweight
- Modular
- Highly Customisable / Hot swappable modules
- Scalable (built in service discovery and control)
- Fault tolerant (dynamic connectivity, discovery, message caching)
- Underlying transport mechanisms ZMQ (Multilanguage Bindings)
- JSON formatted message passing
- Few external dependencies (Boost, ZMQ)

How It Works: Structure / Nomenclature

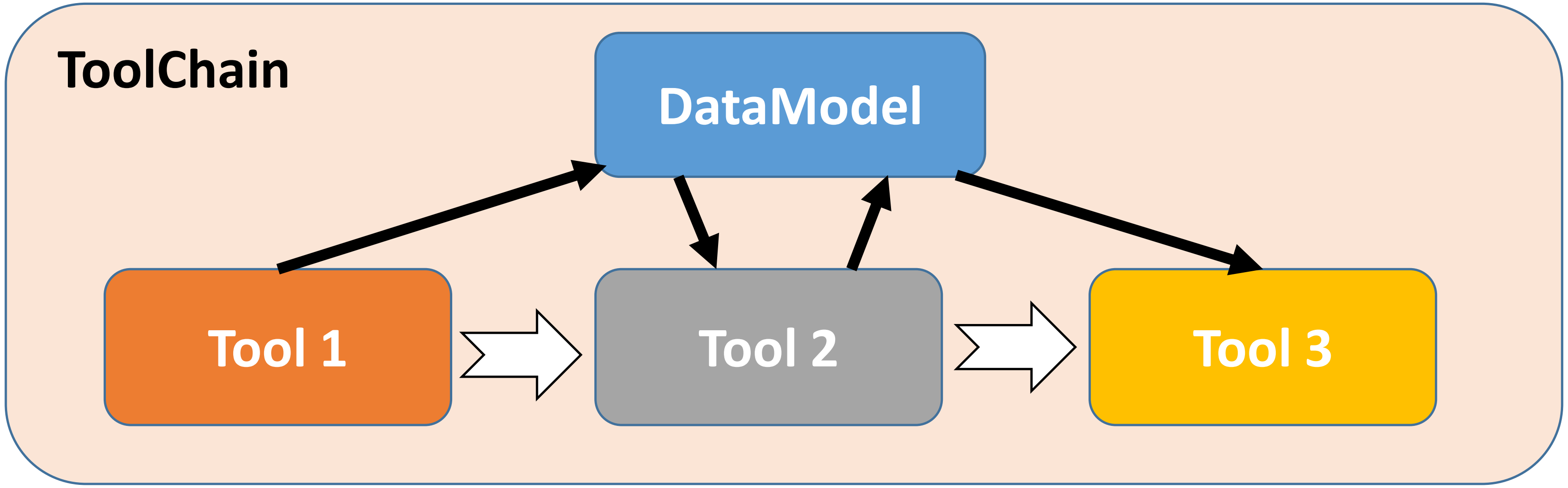


Tool = Modular classes that make up your program

ToolChain = Class that holds the modular Tools

Dynamic simpe ascii files determin which tools to run without compilation in which order. Similarly dynamic variables can be sent to each via simple acii files

How It Works: Structure / Nomenclature



Tool = Modular classes that make up your program

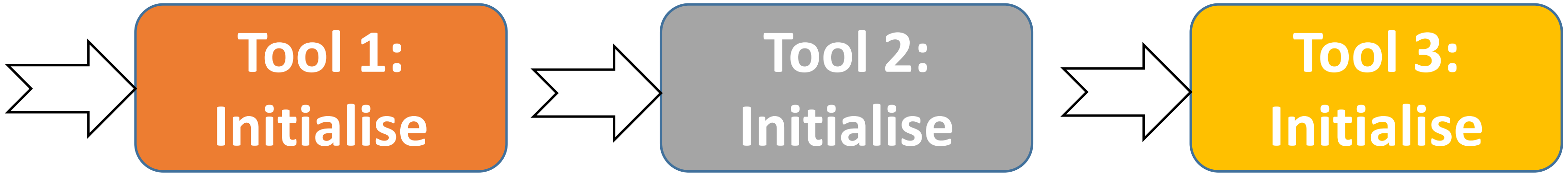
ToolChain = Class that holds the modular Tools

DataModel = Shared / transient data class. Any object/variable/instance in the DataModel class is shared between all tools

Operation

ToolDAQ works by Initialising, Executing and Finalising each tool sequentially

Initialise: (use to initialise variables, create data structures, open files)

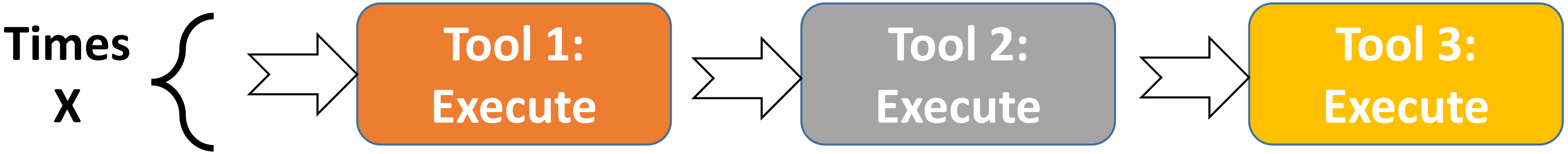


Tool 1: Initialise

Tool 2: Initialise

Tool 3: Initialise

Execute: (use to perform the operation on data, either one entry or all)



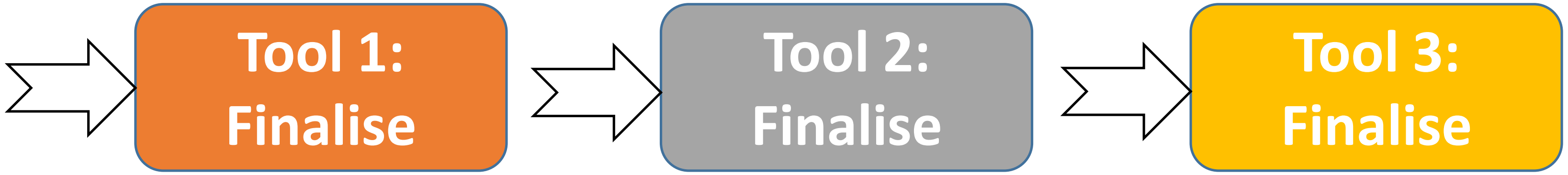
Times X

Tool 1: Execute

Tool 2: Execute

Tool 3: Execute

Finalise: (use to close files, delete and clean up)

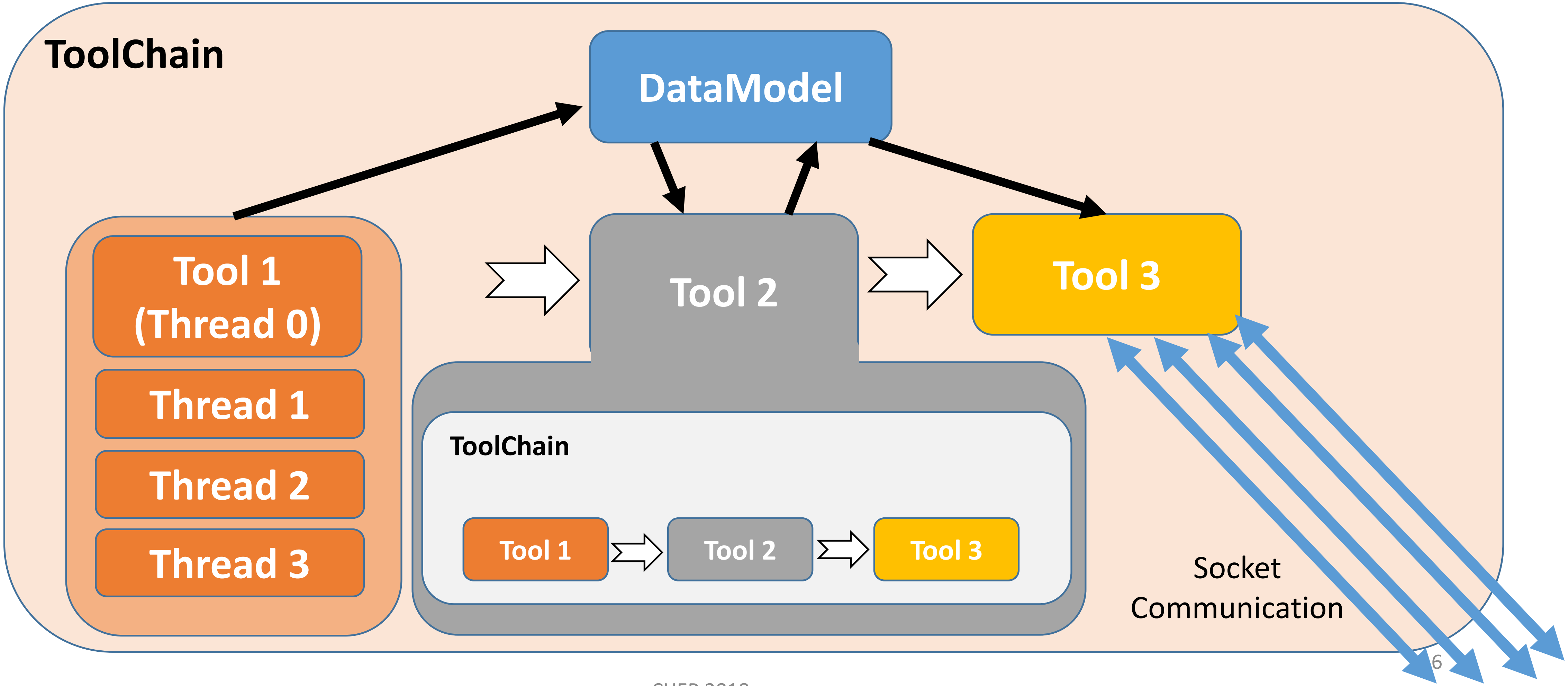


Tool 1: Finalise

Tool 2: Finalise

Tool 3: Finalise

Tools

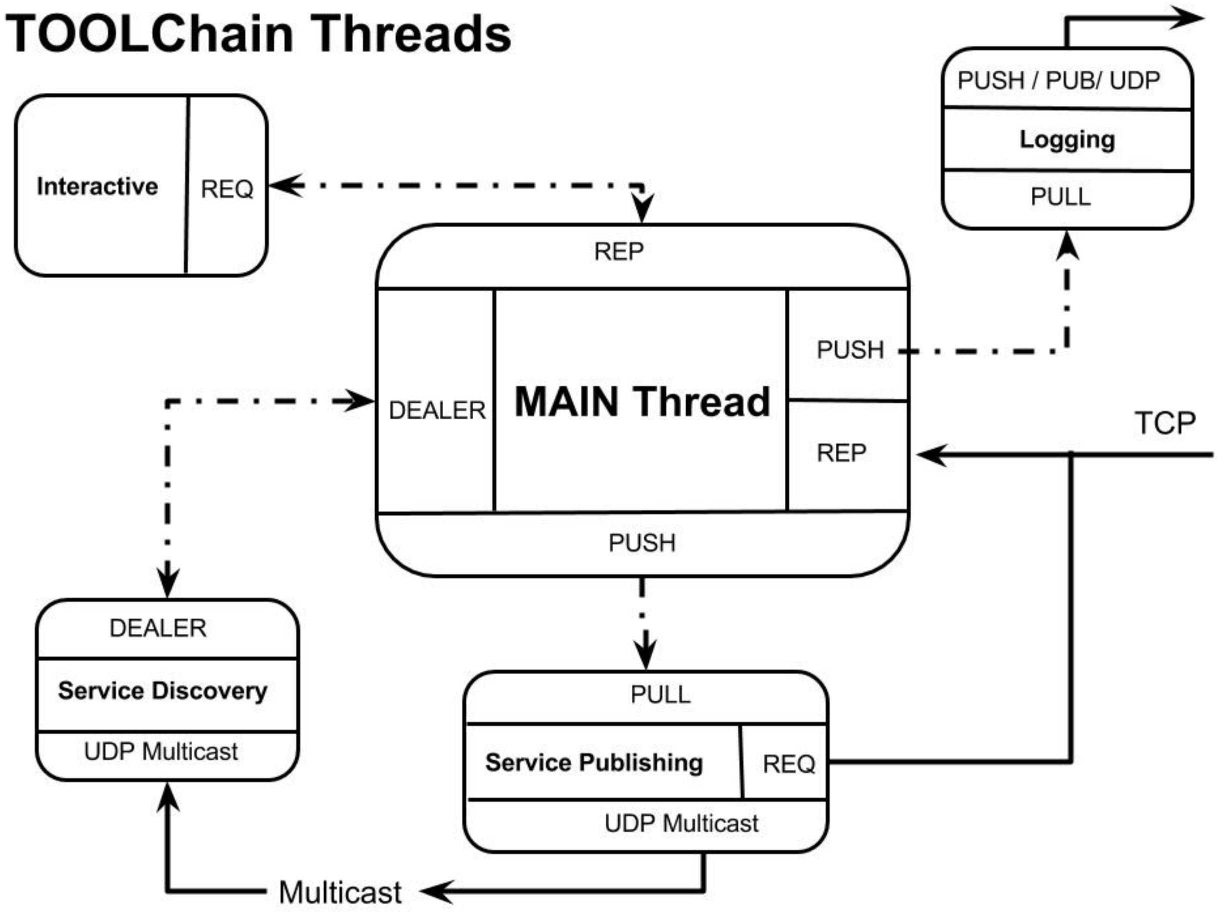

ToolChain
DataModel
Tool 1
(Thread 0)
Thread 1
Thread 2
Thread 3
Tool 2
Tool 3
ToolChain
Tool 1
Tool 2
Tool 3
Socket
Communication

ToolDAQ in Detail

All Users need do is write their own modular Tools for hardware control and data processing while under the hood the software provides a powerful set of features out of the box.

- Three execution modes (Interactive, Inline and Remote)
- Built in distributed Network DAQ control through command line or Web Interface
- Built in Dynamic Service Discovery and Publishing
- Remote or Local Logging modes
- Simple threaded scalable Fault tolerant NtoN Networking technology provided by ZMQ
- Configuration file tools using a universal data storage class

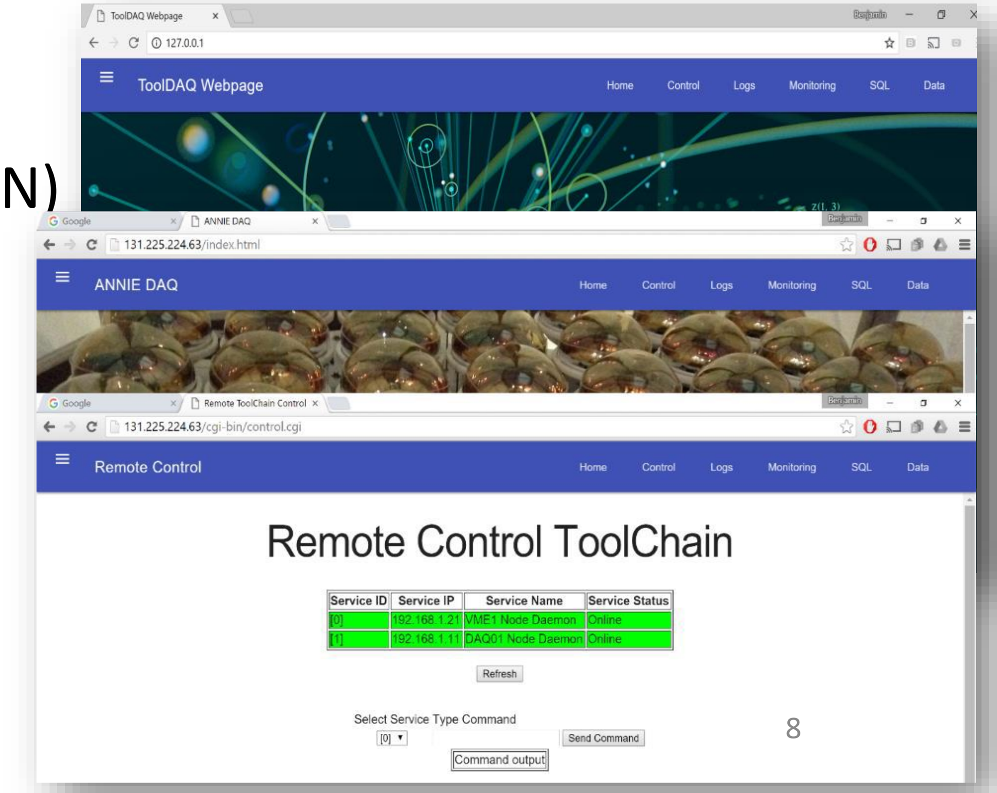
And more…



Control

- Each ToolChain can be run in three modes (determined by asci config file)
  - Interactive: Console command based control (start stop status pause execute initialise etc.)
  - Inline: Fixed or dynamic execution cycles
  - Remote: Remote over network commands (JSON)

- Remote Control can be achieved by:
  - Console based program
  - Web interface



Remote Control ToolChain

Service ID	Service IP	Service Name	Service Status
[0]	192.168.1.21	VME1 Node Daemon	Online
[1]	192.168.1.11	DAQ01 Node Daemon	Online

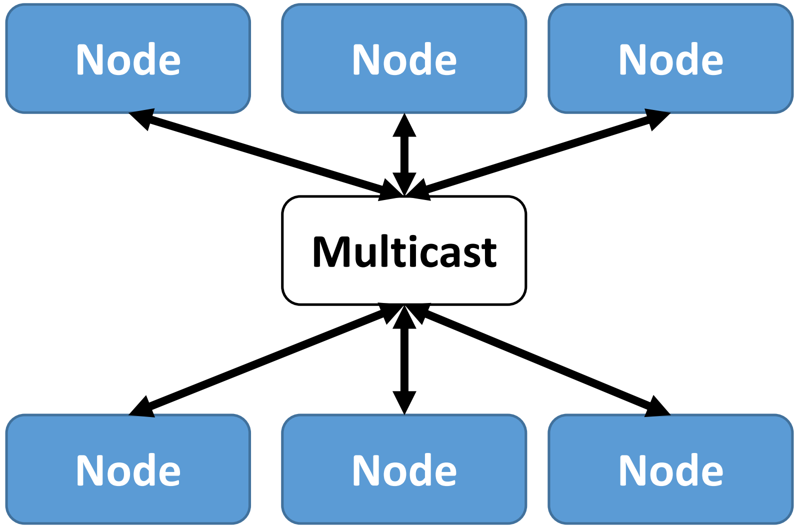
Select Service Type Command
[0] ▾

Dynamic Service Discovery

The core Framework has multiple threads that run both in the ToolChains and NodeDaemons that take care of all the control systems, service discovery, etc…

Dynamic service discovery lets every single Node Daemon, ToolChain and service know about each other via use of multicast beacons.

This is how remote control is achieved anywhere on the network



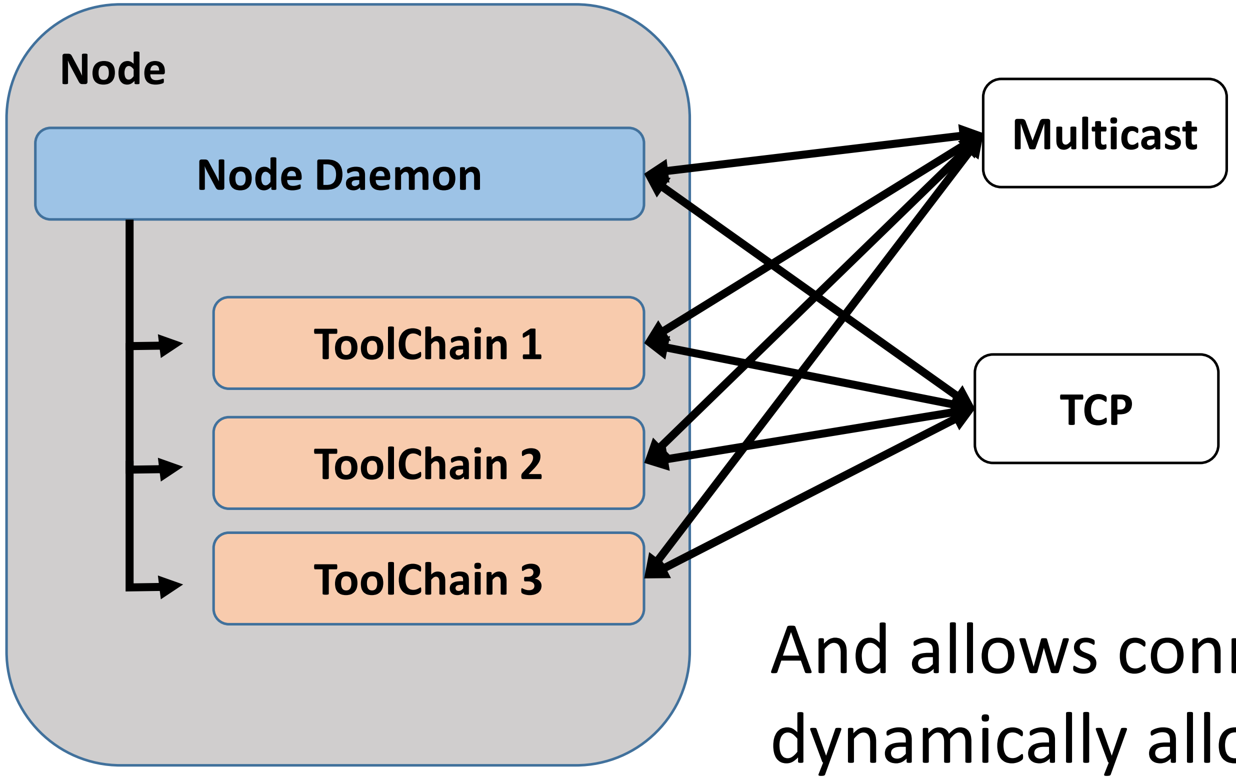
[UUID, Name , IP, Service, Port, Status, Timestamp]

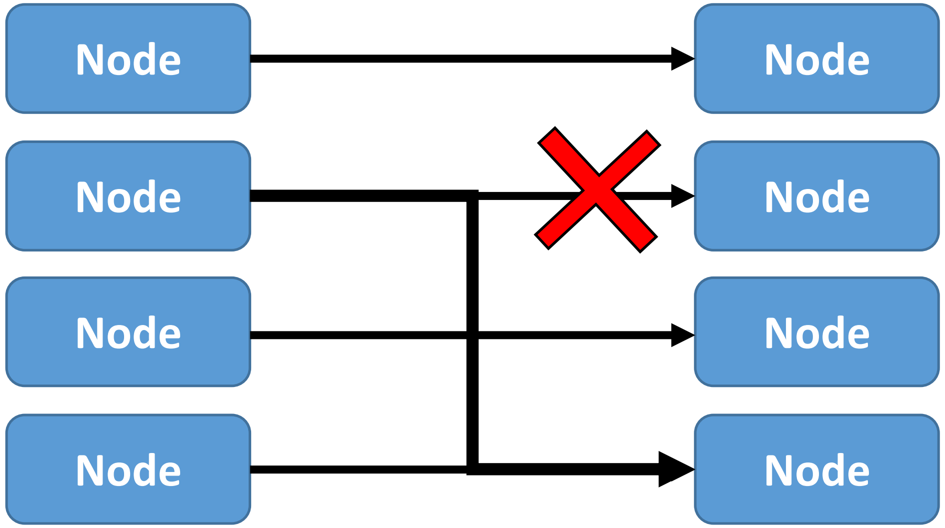
Distributed Node Management & Hot Swapping

Most DAQ systems will require multiple distributed nodes

Each can have multiple ToolChains running on them

So ToolDAQ has a node control and monitoring system





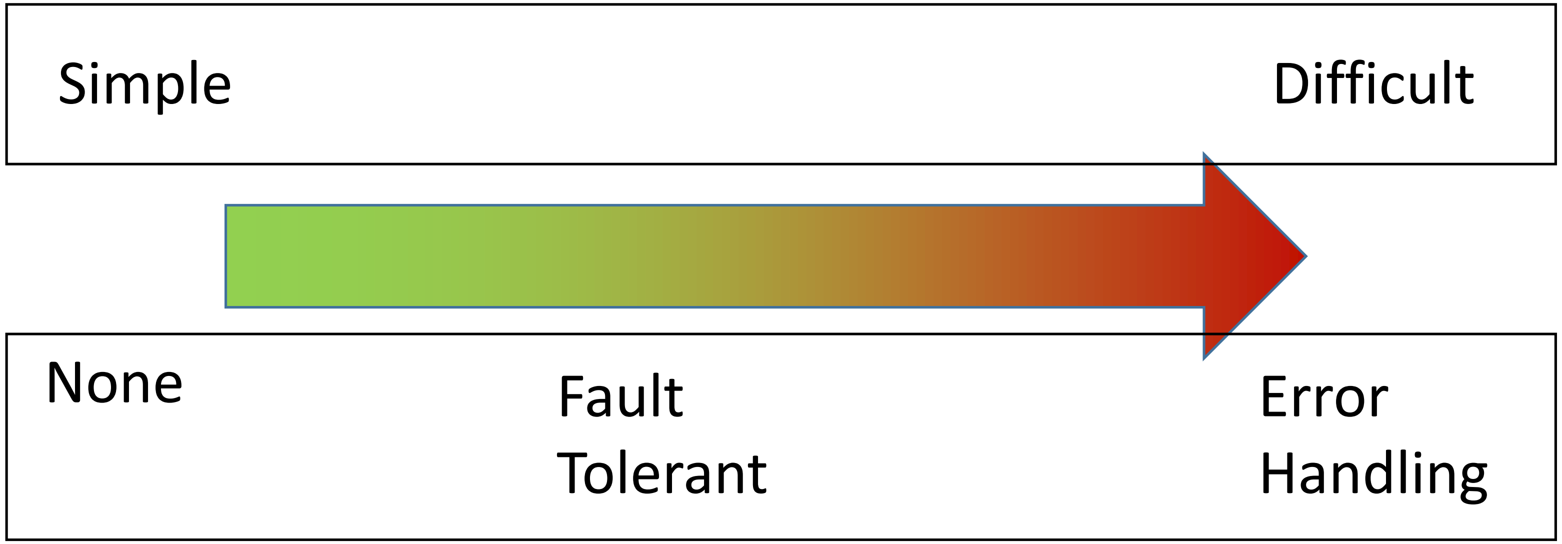
And allows connections and data flow to rerouted dynamically allowing for hot swapping

Fault Tolerance And Error Correcting

- ToolDAQ makes use of ZMQ to provide a fault tolerant scalable messaging

- Use of ZMQ, message buffering and Service discovery allows for creation of DAQ that can not just be fault tolerant but handle errors.

Difficulty



Simple

Difficult

None

Fault Tolerant

Error Handling

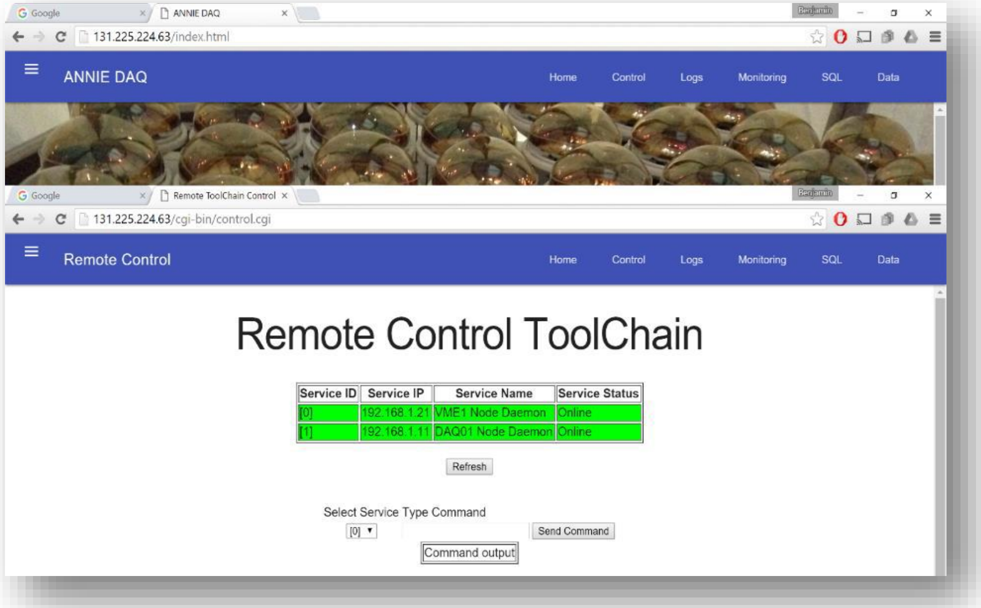
Network Error behaviour

Logging And Monitoring

- Facilities exist for Logging both locally on each node and centrally via a network

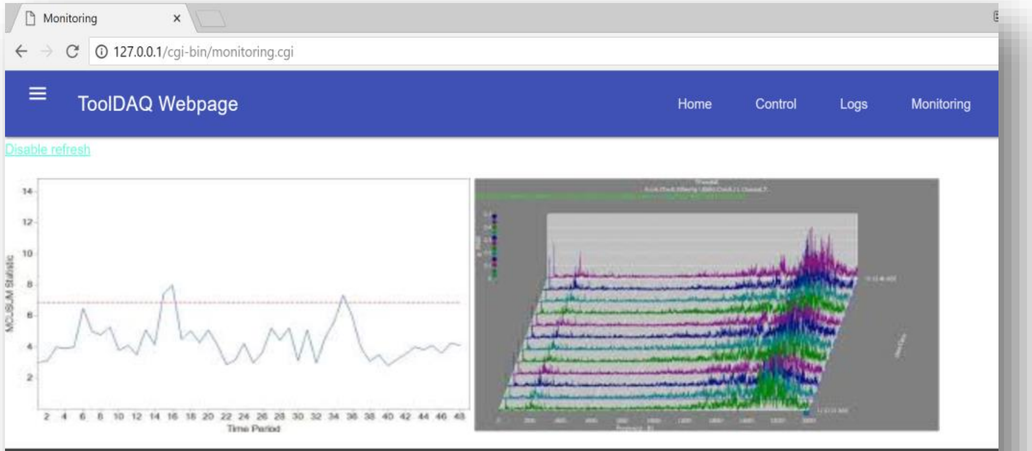
- Also Monitoring of both each nodes NodeDaemon, ToolChains and services status is included with the framework

- Monitoring of data flow, data quality and other phyiscs plots can also be achieved via the Webpage and a seperate ToolChain for monitoring



Remote Control ToolChain

Service ID	Service IP	Service Name	Service Status
[0]	192.168.1.21	VME1 Node Daemon	Online
[1]	192.168.1.11	DAQ01 Node Daemon	Online



Store

ToolDAQ comes with two universal storage classes.

- These act like maps where the value can be of any type within the same object.
- Anything can be stored from basic types, stl containers and custom classes
- These stores are serialisable (ascii, binary and json) and portable
- They can also be used for multi event storage similar to a TTree

Standard std::map

Key	Value
“Val1”	467.4
“Val2”	234.56
“Val3”	235.623

Store

Key	Value
“Val1”	“Hello world”
“Val2”	234.356
“Val3”	MyClass
“Val4”	std::vector<float>

Easy Installation, Tool Sharing And Docker

- The software is all open source and hosted on GitHub
- There are installation scripts to install the software and all dependencies
- As well as docker images of tagged builds and the latest branches

- Due to the modular nature Tools developed for one application can be shared between others. Meaning that the library of available tools keeps growing adding fuctionality

Where It’s Being Used

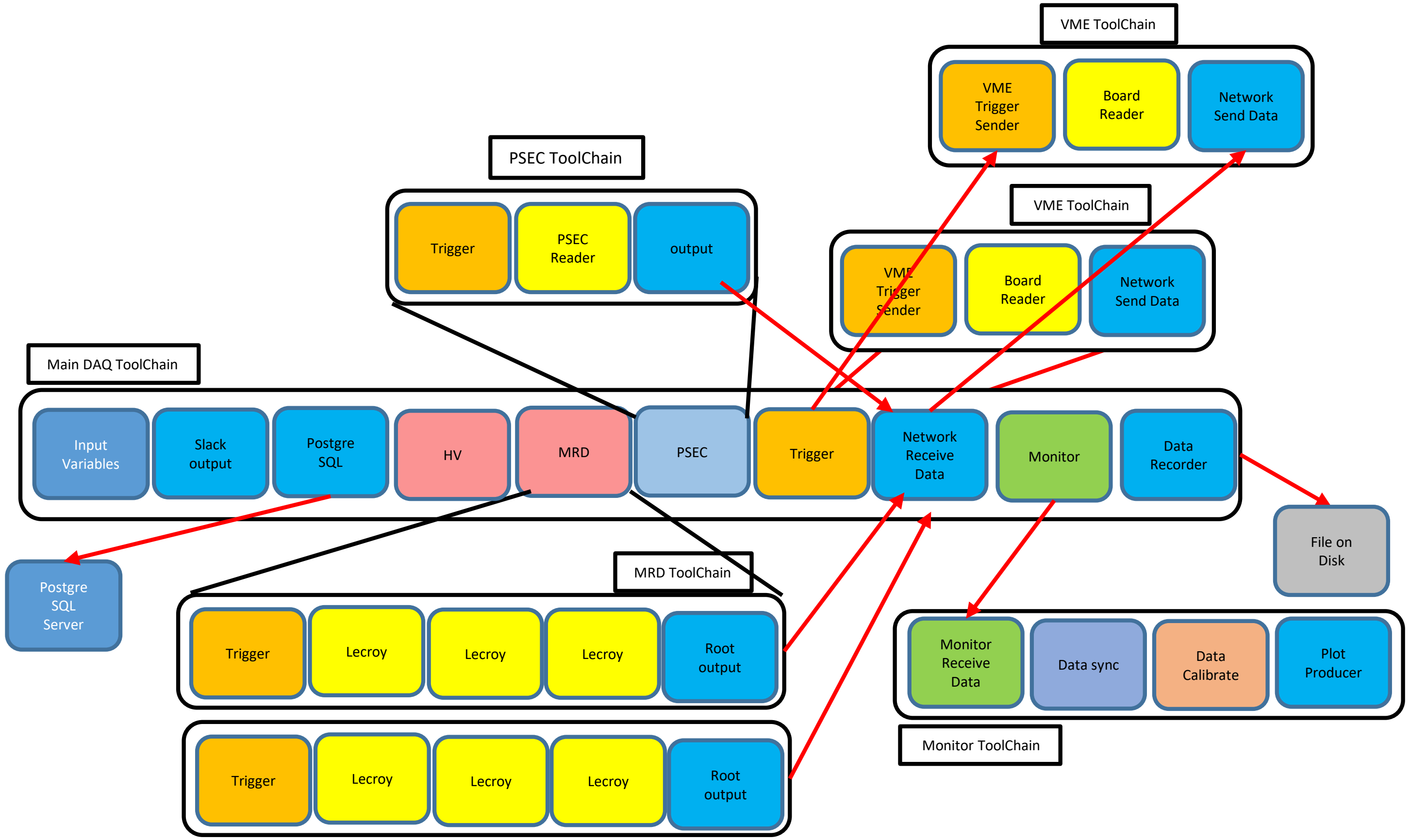
- **ANNIE** (~200 PMTs)
- **Intermediate Water Cherenkov** (~7,500 PMTs [365 MPMTs])
- **Hyper-K** (~40,000 PMTs)
- **Hardware test stands**

<u>Discussing development for:</u>

- WATCHMAN
- ND280
- SNO+

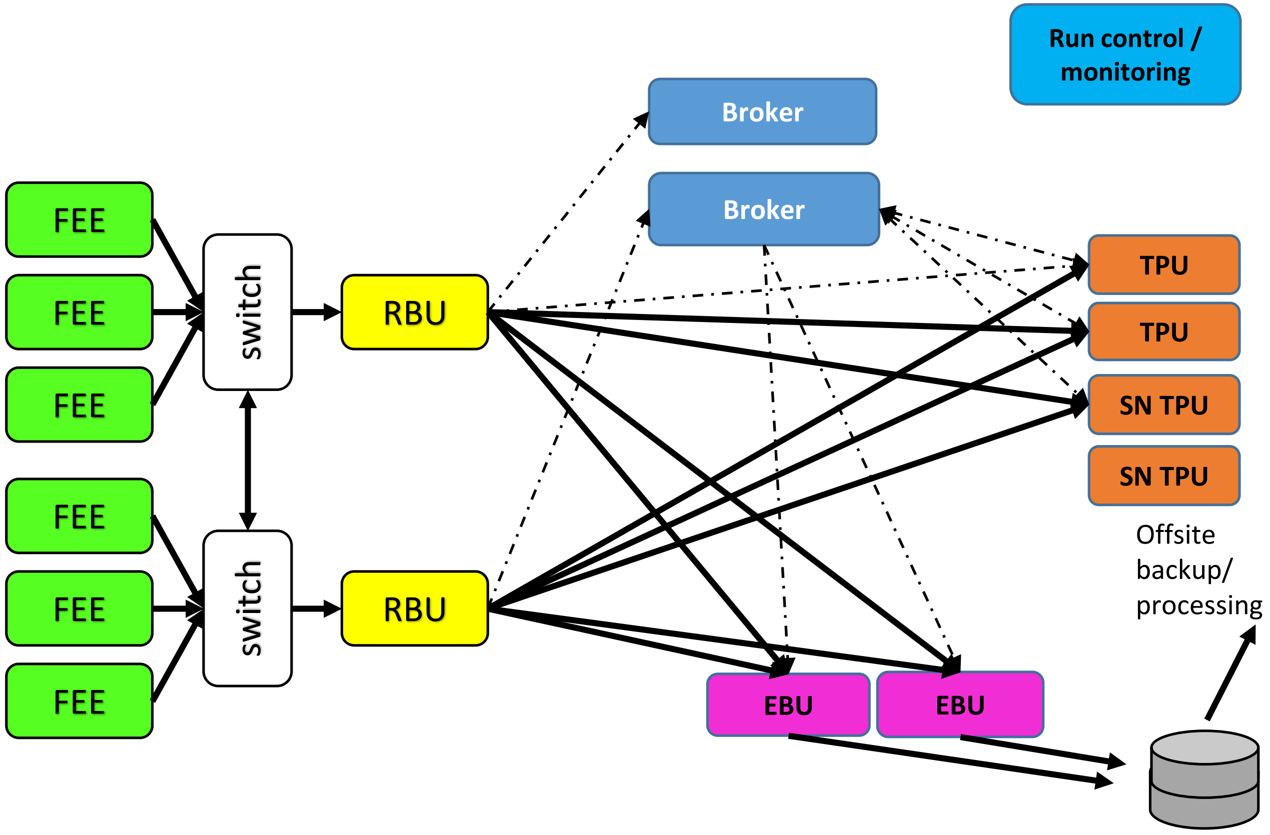
ANNIE

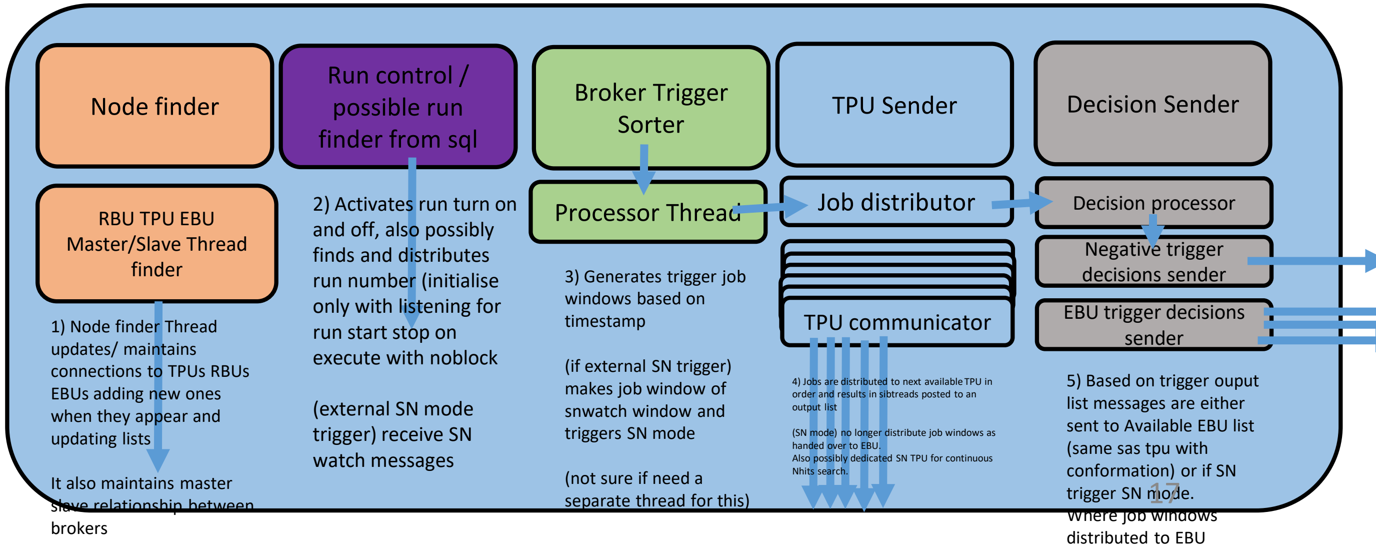
- Multiple asynchronous data sources MRD, Veto, PMTs LAPPDs
  - ADC Koto Boards
  - Camac TDC
  - PSec LAPPDs
  - Trigger stream
- Fault tolerant
- Flexible to changes in data and trigger
- Also used for analysis/reconstruction



Hyper-K

- Larger Beast
  - 40,000 channels
  - 2000 FEEs
  - 150 computing nodes
  - Dead timeless
  - Separate GPU Trigger farm
  - Readout buffering
  - Event Building
- Self maintaining
- Highly fault tolerant
- Also starting to use ToolFrame work for simulated triggering




Node finder
Run control / possible run finder from sql
Broker Trigger Sorter
TPU Sender
Decision Sender
RBU TPU EBU Master/Slave Thread finder
Processor Thread
Job distributor
Decision processor
Negative trigger decisions sender
EBU trigger decisions sender
TPU communicator

1) Node finder Thread updates/ maintains connections to TPUs RBUs EBUs adding new ones when they appear and updating lists

It also maintains master slave relationship between brokers

2) Activates run turn on and off, also possibly finds and distributes run number (initialise only with listening for run start stop on execute with noblock

(external SN mode trigger) receive SN watch messages

3) Generates trigger job windows based on timestamp

(if external SN trigger) makes job window of snwatch window and triggers SN mode

(not sure if need a separate thread for this)

4) Jobs are distributed to next available TPU in order and results in sibtreads posted to an output list

(SN mode) no longer distribute job windows as handed over to EBU.
Also possibly dedicated SN TPU for continuous Nhits search.

5) Based on trigger ouput list messages are either sent to Available EBU list (same sas tpu with conformation) or if SN trigger SN mode. Where job windows distributed to EBU

Summary

- ToolDAQ is lightweight and highly scalable DAQ framework
- It allows for dynamic service discover reconfiguration and high fault tolerance
- It’s currently being employed by a few experiments. With a few more exploring its use
- The library of Tools is growing constantly
- Please check out the code and contact me if your interested (b.richards@qmul.ac.uk)