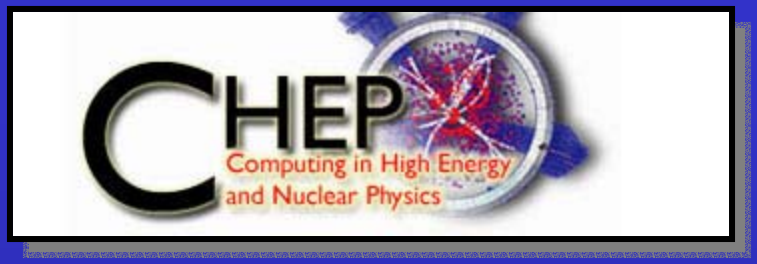
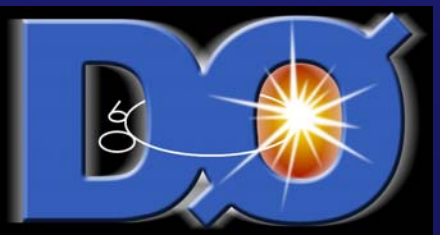
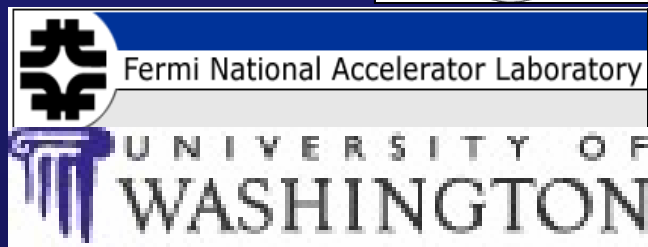


DAQ Monitoring and Auto Recovery at DØ

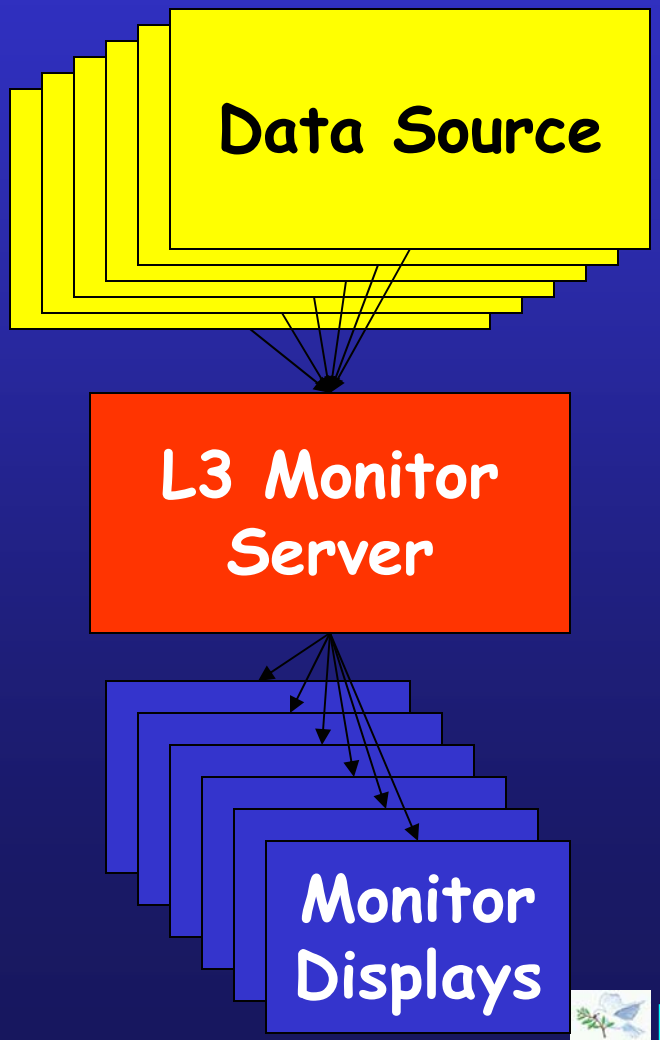
B. Angstadt, G. Brooijmans, D. Chapin, D. Charak, M. Clements, S. Fuess, A. Haas, R. Hauser, D. Leichtman, S. Mattingly, A. Kulyavtsev, M. Mulders, P. Padley, D. Petravick, R. Rechenmacher, ***G. Watts***, D. Zhang





Monitor Design

- The simple monitor system is easy to write
  - Perhaps not very interesting
  - Often gets left to last
  - Crucial for long term debugging.
- Only a few basic design questions
  - Push/Pull
  - Direct connect to data sources or server
  - Request Granularity
  - Data Format



Ours Was No Different

- The DØ DAQ Monitor
  - Wasn't written until it was clear we needed it
- General design principles were based on Run 1 experience
  - Too many displays affected system performance
  - Display maintained connections to every source
  - Data requested was only a large binary block.

Run 2 Environment

- Many more data sources
  - Of order 200
- Possibly many more displays
- Online System Firewall
  - Due to DOE security requirements.
- Primary debugging tool for shifters and remote experts

Design Principles

- Use a Monitor Server Architecture
  - Only connection to the data sources
  - Protocol is based on the pull
- Data Format is XML
  - Standard format that is easily extensible
  - Easily readable by any programming language.
- Bother monitor sources of data as infrequently as possible
  - Fine grained monitor data request
    - Some data may almost never be asked, but take significant resources to generate
  - Cache data in the monitor server for all data items.
- Be robust against crashes and other spurious effects
  - Auto reconnect built into clients and data sources
- Be able to handle a large number of data sources and displays
  - Heavily multithreaded
- Serve Real-time data
  - Cached data has a time stamp
  - Display data requests have an associated maximum age
    - Minimum time is one second

Monitor Server Implementation

- Written in C++
- Communication is TCP/IP, based on the ACE RT framework
- Data Sources connect to it
- Displays connect to it
- Caches data
  - One for every data source
- Highly multithreaded
  - Prevents blocking due to long latency displays

Performance

- Takes 10% of a 2 CPU machine under full Monitoring Load
  - PIII Xenon, 1 GHz.
- 4% of $\frac{1}{2}$ gig memory
- Serves about $\frac{1}{2}$ meg per second of monitor data
  - Cache Hit Rate is about 50%
  - Could setup second layer of a caching server if so desired
- Typical time to satisfy query is 50ms
- Relay to serve requests from outside online system
  - An access point through security firewall.
- 162 Data Sources, 35 displays

Wire Protocol

- All component connections are TCP/IP
- All run a common format
  - Message length word followed by the message text.
- XML format is simple!

This simplicity is perhaps the single biggest reason for this system's success.

Example Request To Client

```
<luminosity
  <d0_luminosity/> <b0_luminosity/>
  <beam_energy/><pbar_stack_size/>
  <outside_temperature/>
  <tev_protons/><tev_pbars/>
  <tev_beam/>
  <store_number/>
</luminosity>
```

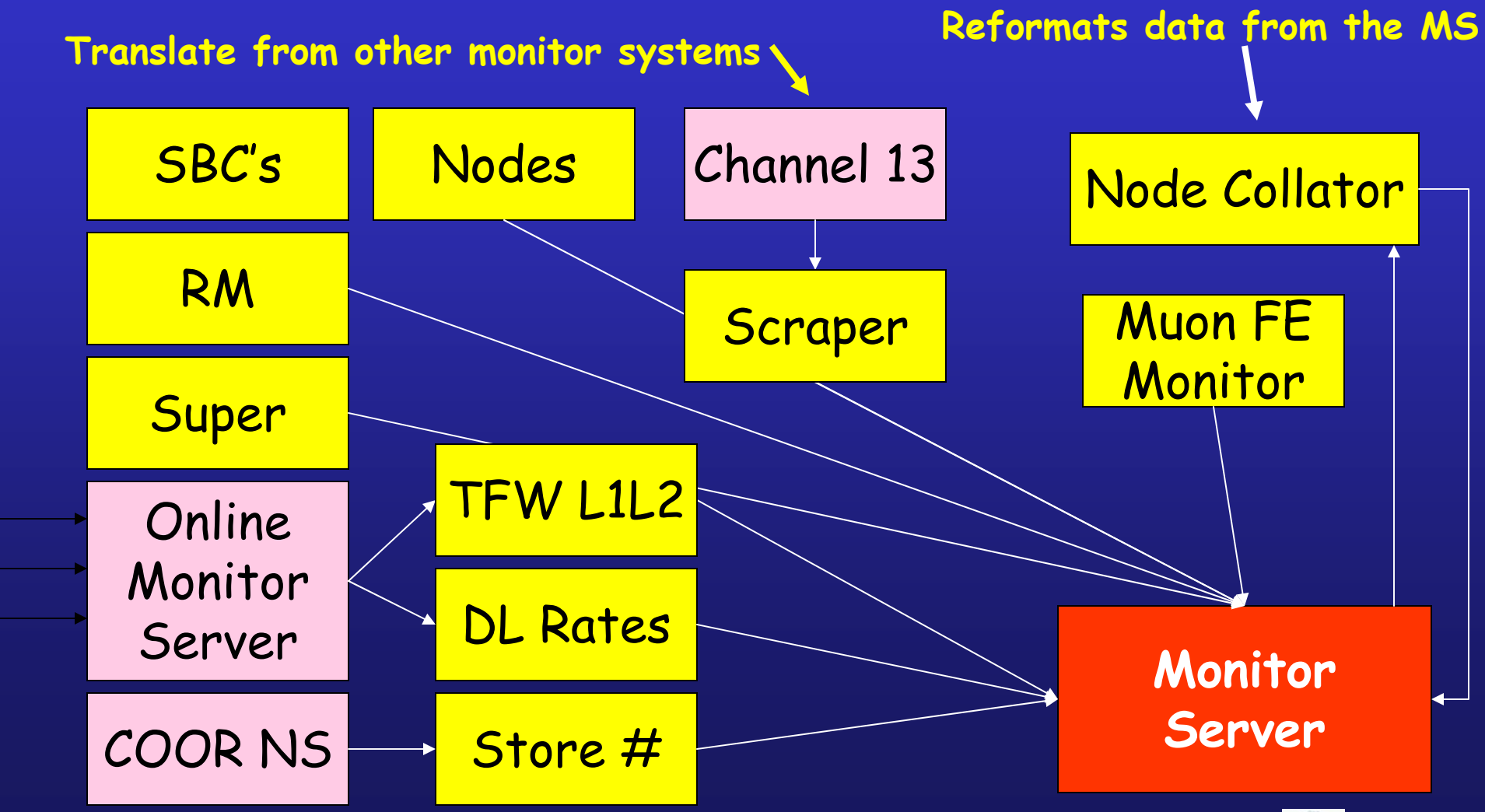
```
<luminosity
  <d0_luminosity>13.28 </d0_luminosity>
  <b0_luminosity> 24.58</b0_luminosity>
  <beam_energy> 987.43</beam_energy>
  <pbar_stack_size> 24.58</pbar_stack_size>
  <outside_temperature>41.92 </outside_temperature>
  <tev_protons>5869.87 </tev_protons>
  <tev_pbars>602.7</tev_pbars>
  <tev_beam>6.8</tev_beam>
  <store_number>1949 </store_number>
</luminosity>
```

Complex Monitor Data

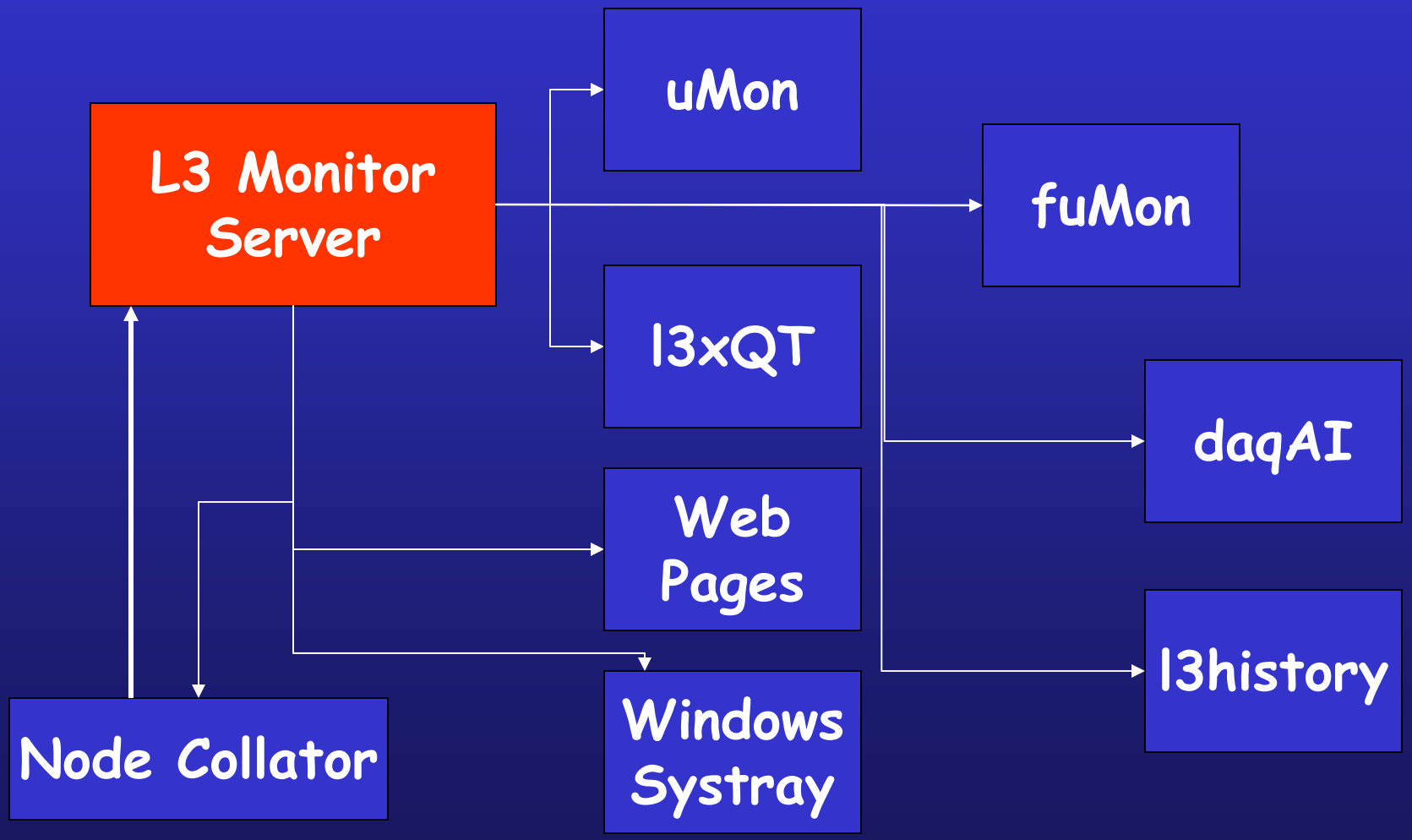
- What is sent between monitor item tags is up to data source
  - Can be arbitrarily complex.
  - Matches best with rest of system if in XML
    - But this is not a requirement!


```
  <evb_missing_crate_percents />
- <evb_missing_crates>
  - <crate>
      <id>0</id>
      <missing>0</missing>
      <required>0</required>
    </crate>
  - <crate>
      <id>1</id>
      <missing>0</missing>
      <required>0</required>
    </crate>
  - <crate>
      <id>2</id>
      <missing>0</missing>
      <required>0</required>
    </crate>
  - <crate>
      <id>3</id>
      <missing>0</missing>
      <required>0</required>
    </crate>
  - <crate>
      <id>4</id>
```

Smorgasbord…



Current *Displays…*



Coding…

C++ *Data Source* Code

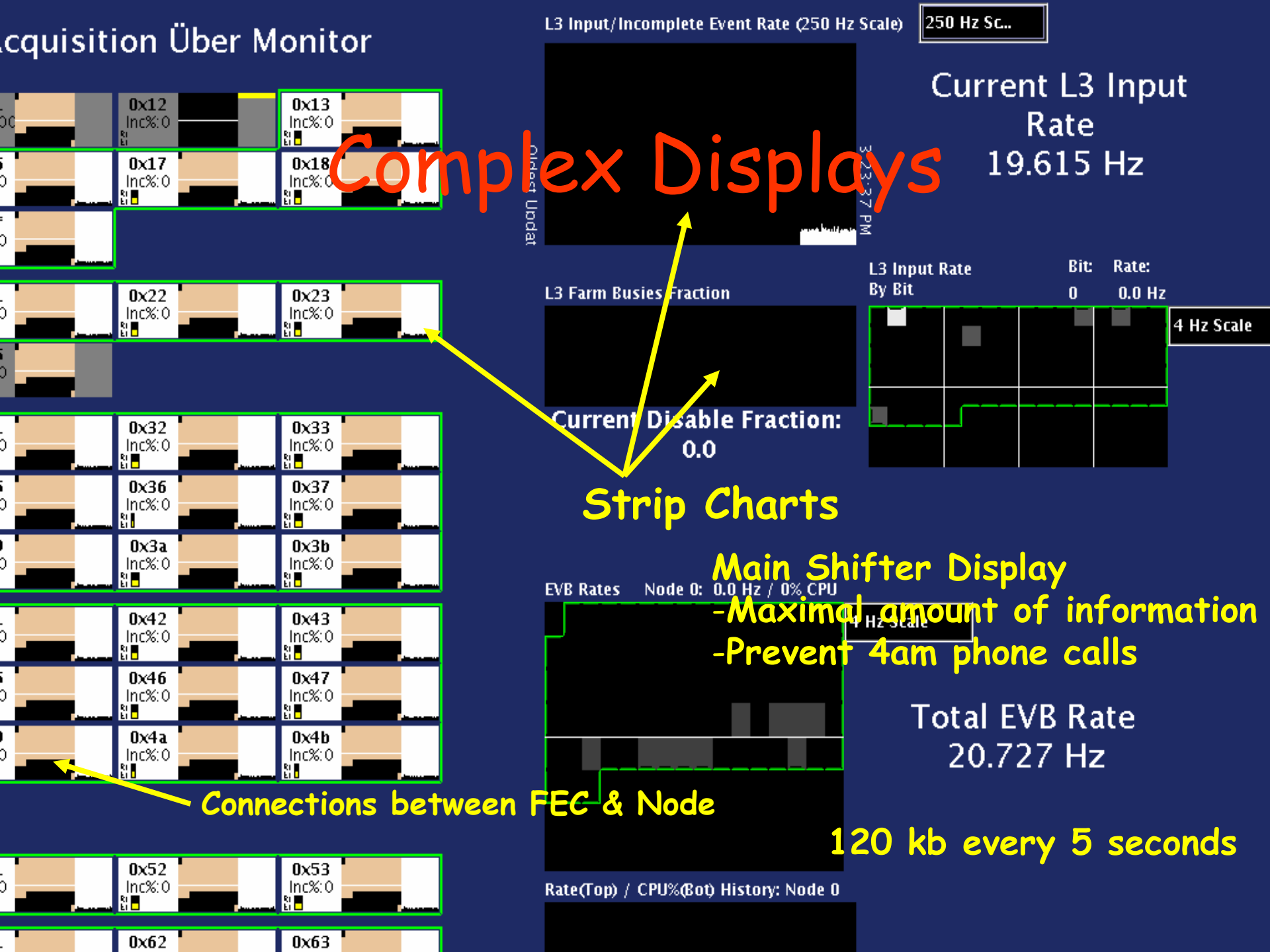
```
l3_monitor_util_reporter_ACE monitor_callback
("tester");
  L3_monitor_connector<l3_monitor_util_reporter_ACE>
ms_connection ("d0l3mon.fnal.gov",
DEFAULT_L3MONITOR_CLIENT_PORT,
&monitor_callback);
  ms_connection.connect_to_server (true);
  l3_monitor_object_op<int> counter ("count");
  while (1) {
    counter++;
    std::cout << "Doing iteration " << counter <<
std::endl;
    ACE_OS::sleep(1);
  }
  return 0;
}
```

Code frameworks in Python and C++

Python *Display* Code

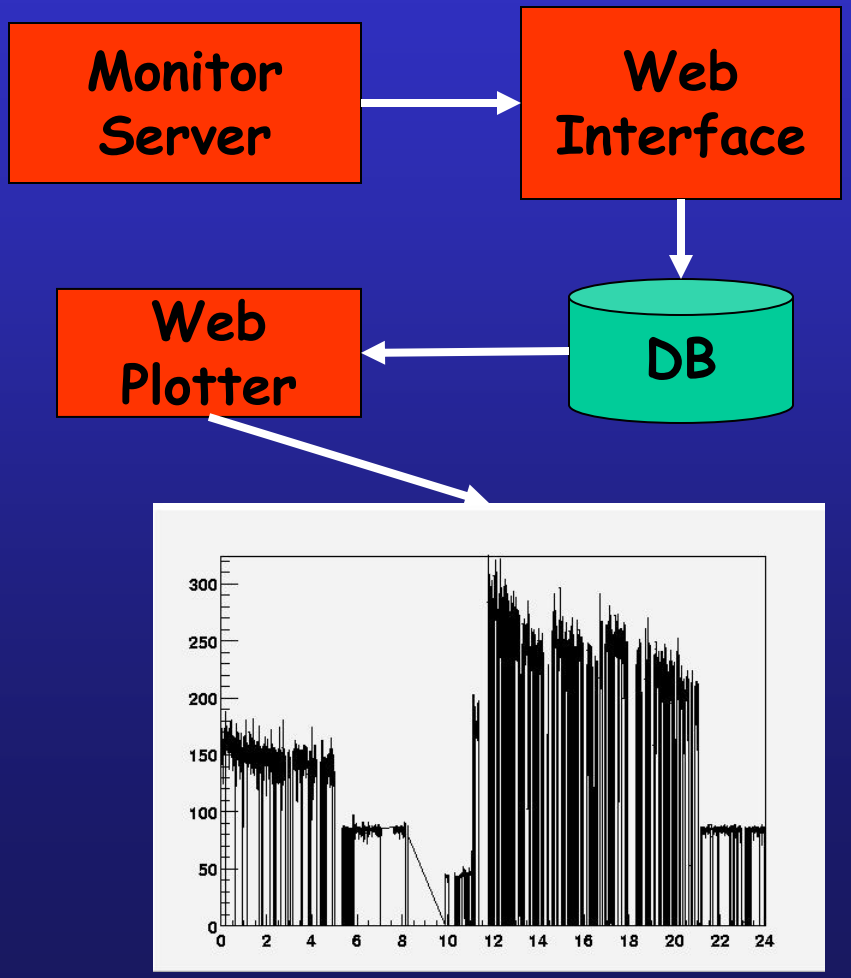
```
import time
import l3xmonitor_util_module
disp = l3xmonitor_util_module.monitor_display
test_item = disp.get_item("tester", "count")
while 1:
  disp.query_monitor_server()
  print "Count value is %s" % test_item[0]
  time.sleep(1)
```

Code frameworks in Python, C++, Java, and C#


Acquisition Über Monitor
L3 Input/Incomplete Event Rate (250 Hz Scale)
250 Hz Sc...
Current L3 Input Rate 19.615 Hz
Complex Displays
L3 Farm Busies Fraction
Current Disable Fraction: 0.0
L3 Input Rate By Bit
Bit: 0
Rate: 0.0 Hz
4 Hz Scale
Strip Charts
Main Shifter Display
-Maximal amount of information
-Prevent 4am phone calls
EVB Rates
Node 0: 0.0 Hz / 0% CPU
Total EVB Rate 20.727 Hz
Connections between FEC & Node
120 kb every 5 seconds
Rate(Top) / CPU%(Bot) History: Node 0
0x12 Inc%:0
0x13 Inc%:0
0x17 Inc%:0
0x18 Inc%:0
0x22 Inc%:0
0x23 Inc%:0
0x32 Inc%:0
0x33 Inc%:0
0x36 Inc%:0
0x37 Inc%:0
0x3a Inc%:0
0x3b Inc%:0
0x42 Inc%:0
0x43 Inc%:0
0x46 Inc%:0
0x47 Inc%:0
0x4a Inc%:0
0x4b Inc%:0
0x52 Inc%:0
0x53 Inc%:0
0x62
0x63

History Logger

- Keep track of data over long periods of time
  - Data Storage in Root
  - Data Storage in Oracle
- XML Format
  - Automatically parsed into ntuple like forms
- Change in data automatically reflected in data stored in DB
  - Web interface to change what is logged
- Make Plots in Real time using web interface
- Written in .NET



Lessons and Out Look

- Problems:
  - Didn't understand how to build strings efficiently in C++
  - Was generating a monolithic block of data in a single monitor item
  - Multi-threaded code that stays up for weeks at a time is hard
- Text vs Binary
  - Only once have we had to encode information in Text for compression reasons
- The code is stand alone
  - We rely on ACE for the monitor server
  - Display code frequently relies on Xerces and also ACE.
  - Data Source code relies on ACE and Xerces
- Wire protocol is very simple
  - Could write directly to it and skip dependencies.

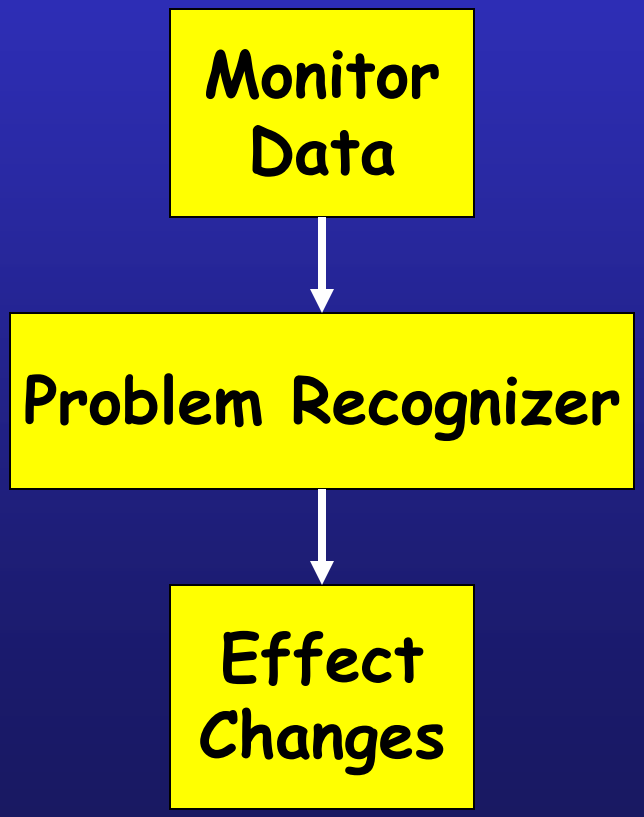
Simple To Use → **Everyone Uses It**

Complex Systems…

- Experiment has many 1000's of computer controlled components.
- Problems fixable
  - Long term
    - Visa problem!
  - Work around
  - Decrease data taking efficiency.
- Can this be addressed?

Called "daqAI" ← **Bad Name!**

Simple Design…



The Components

New problems often require the addition of new data to monitor system

Monitor Data

Hard!

Problem Recognizer

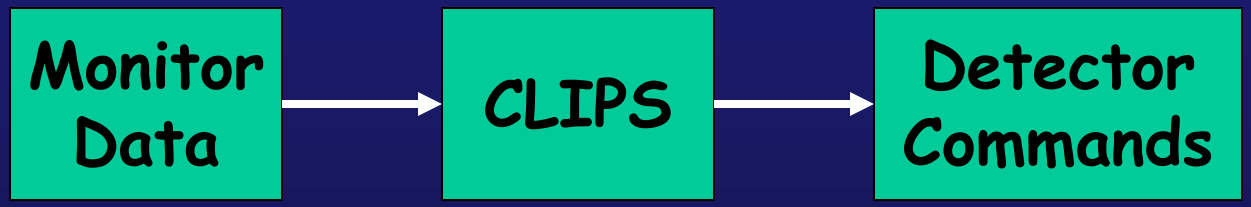
- All of detector is computer controlled
- Control is decentralized in DØ
- Safety/Political allowing computer to send commands

Effect Changes

Recognizing The Problem

- Must be dynamic
  - Each new problem should not require major update to code
  - Problems change
  - What problems symptoms indicate change
- Must be safe
  - Feedback loops are bad!
  - Should not cause harm or increase dead time!

Rule Based Recognizer



CLIPS Sample

Monitor Data Converted to facts

```
(l3xetg-rate_events 55)
(coormon-store_number 2326)
```

Rules…

```
(defrule rate_is_low "See if the daq rate is less than 30 Hz"
        (l3xetg-rate_events ?num&:(< ?num 30))
        =>
        (assert (b_daq_rate_is_low ?num)))
(defrule rate_low "Rate is too low. Fix with reset"
        (b_daq_rate_is_low ?r)
        (coormon-store_number ?s&:(> ?s 0))
        =>
        (log_reason "Rate is low")
        (talk "Rate is low. SCL Init issued")
        (issue_coor_request "scl_init"))
```

If less than 30 Hz…

Down Time Log

Tell Shifters What Is Up

Attempt to Fix It

Produces Error Reports

- Had to produce a report of what it had done.
- All problem detection is associated with a particular problem
- Shift Problem Reports
  - Can see how often something goes wrong and how much downtime it causes

```
daqAI Shift report covering the period 2003-03-24-11:16:
  1 times 'Bad A/O BOT Rate in term(193): muon' took 00:
  2 times 'Crate 0x61 is FEB' took 00:10:56 (328 secs o
  1 times 'Crate 0x64 is FEB' took 00:00:24 (24 secs on
  3 times 'Crate 0x6b is FEB' took 00:19:21 (387 secs o
  1 times 'L3 RM is asserting its disable' took 00:00:0
  1 times 'Muon Crate 0x18 has fatal error' took 00:00:
  1 times 'Muon Crate 0x30 has fatal error' took 00:00:
  21 times 'Other' took 00:43:37 (124.619 secs on avg)

Timer Status:
  Timer in_store: 04:39:04 (off)
  Timer l3_configured: 04:00:12 (off)
    86.0726% of in_store
  Timer daq_configured: 03:27:14 (off)
    74.2594% of in_store
    86.2753% of l3_configured
  Timer good_data_flow_to_l3: 03:11:38 (off)
    68.6694% of in_store
    79.7807% of l3_configured
    92.4723% of daq_configured
  Timer l3_disable: 00:00:16 (off)
    0.0955566% of in_store
```

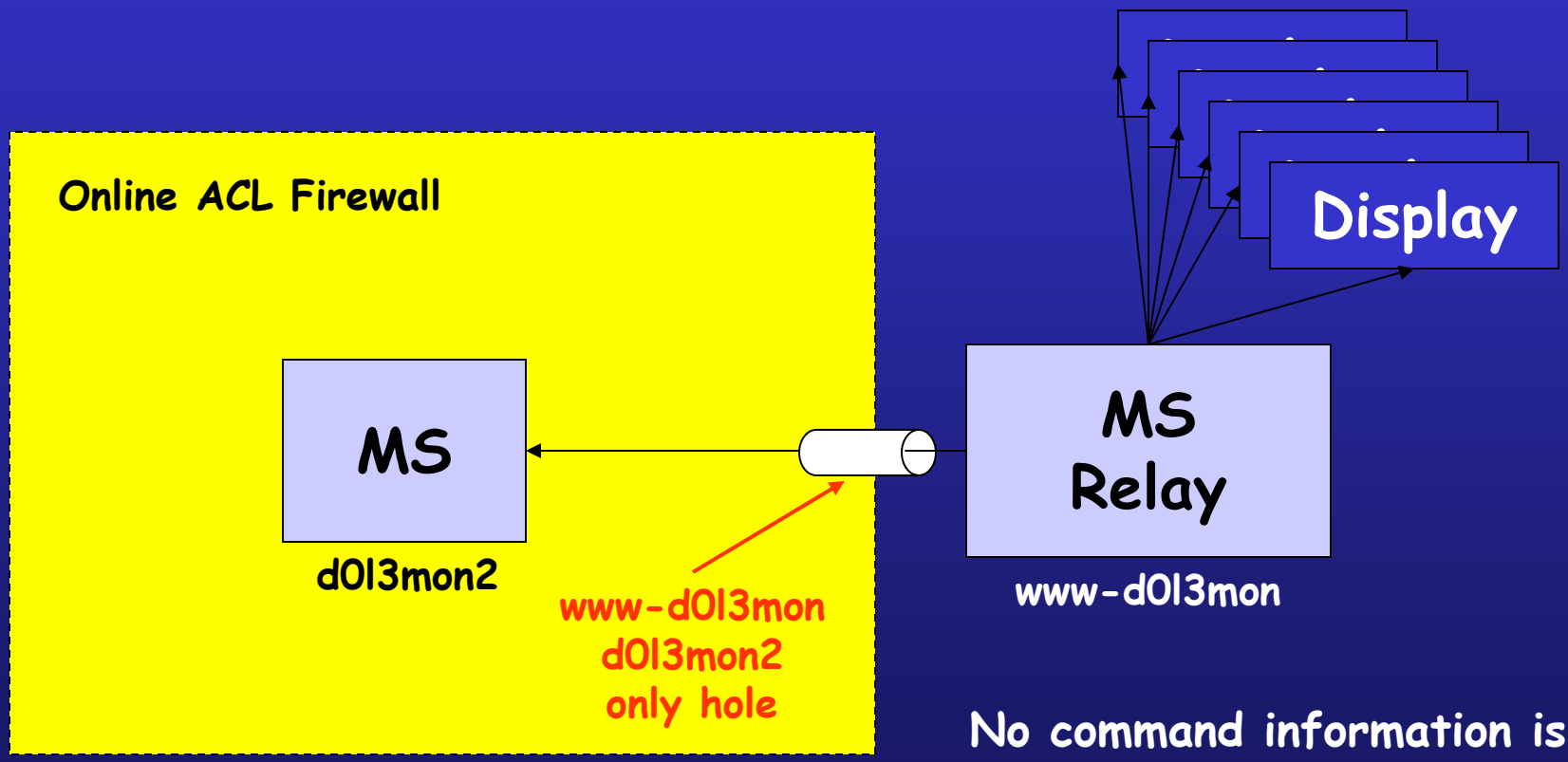
Lessons…

- Successful
  - In the sense it dramatically decreased the downtime for common problems
- Not Successful
  - Requires constant up keep by myself
  - Must convince people this is the right thing to do
  - Adding new conditions is not as easy as it should be
    - Rule based is easier, but there must be a better way!
    - Have to be careful of feedback and unintended consequences

Conclusions

- The Monitor Server has been wildly successful
  - XML format along with simple wire protocol were key to this
  - A easily usable and unified monitoring system allows projects like the auto detection to be easily written.
- The daqAI Project was successful
  - Big help reducing and cataloging DØ's down time.
- Future
  - Monitor Server is stable and not under active development.
  - Looking for a smarter way to implement the problem recognizer for daqAI.

The Outside World



No command information is passed back and forth, just request and query

Only displays can run outside…

Adding your data…

Simplified road map…

1. Choose a name for your data source
   "sbc" "daqAI" "luminosity"
2. Choose the language you'll source the data
   C/C++ Python
3. Write Code!
4. Run behind online ACL

→ No configuration changes required in MS

l3monitor_utils/test/test_monitor_obj_client.cpp

C++ Client

Comments removed!!

```
#include "l3monitor_utils/client_objects/l3_monitor_object_op.hpp"
#include "l3monitor_utils/client_objects/l3_monitor_util_reporter_ACE.hpp"
int main()
{
  l3_monitor_util_reporter_ACE monitor_callback ("tester");
  L3_monitor_connector<l3_monitor_util_reporter_ACE> ms_connection ("d0l3mon.fnal.gov",
DEFAULT_L3MONITOR_CLIENT_PORT, &monitor_callback);
  ms_connection.connect_to_server (true);
  l3_monitor_object_op<int> counter ("count");

  while (1) {
    counter++;
    std::cout << "Doing iteration " << counter << std::endl;
    ACE_OS::sleep(1);
  }
  return 0;
}
```

Client Name

Connect to MS

Declare Integer to be monitored

Change it once per second for fun using usual C++ syntax

l3xmonitor_util_module/test/test_monitor_client.py *

Python Client

Does the same thing

```
import time
import l3xmonitor_util_module
ms = l3xmonitor_util_module.monitor_client("tester")
count=0
while 1:
  ms.set("count", count)
  time.sleep(1)
  count += 1
```

Connect to MS

Set monitor item count

Must be run inside online ACLs!!

Python Display

```
import time
import l3xmonitor_util_module
disp = l3xmonitor_util_module.monitor_display_relay()

test_item = disp.get_item("tester", "count")

while 1:
  disp.query_monitor_server()
  if len(test_item) == 0:
    print "No data from the MS"
  else:
    print "Count value is %s" % test_item[0]
  time.sleep(1)
```

Get connection to MS via ACL relay

Tell system you want the count monitor variable

Load latest data from MS

Make sure data came back from MS & Print!

(MS down, Client Down…)