



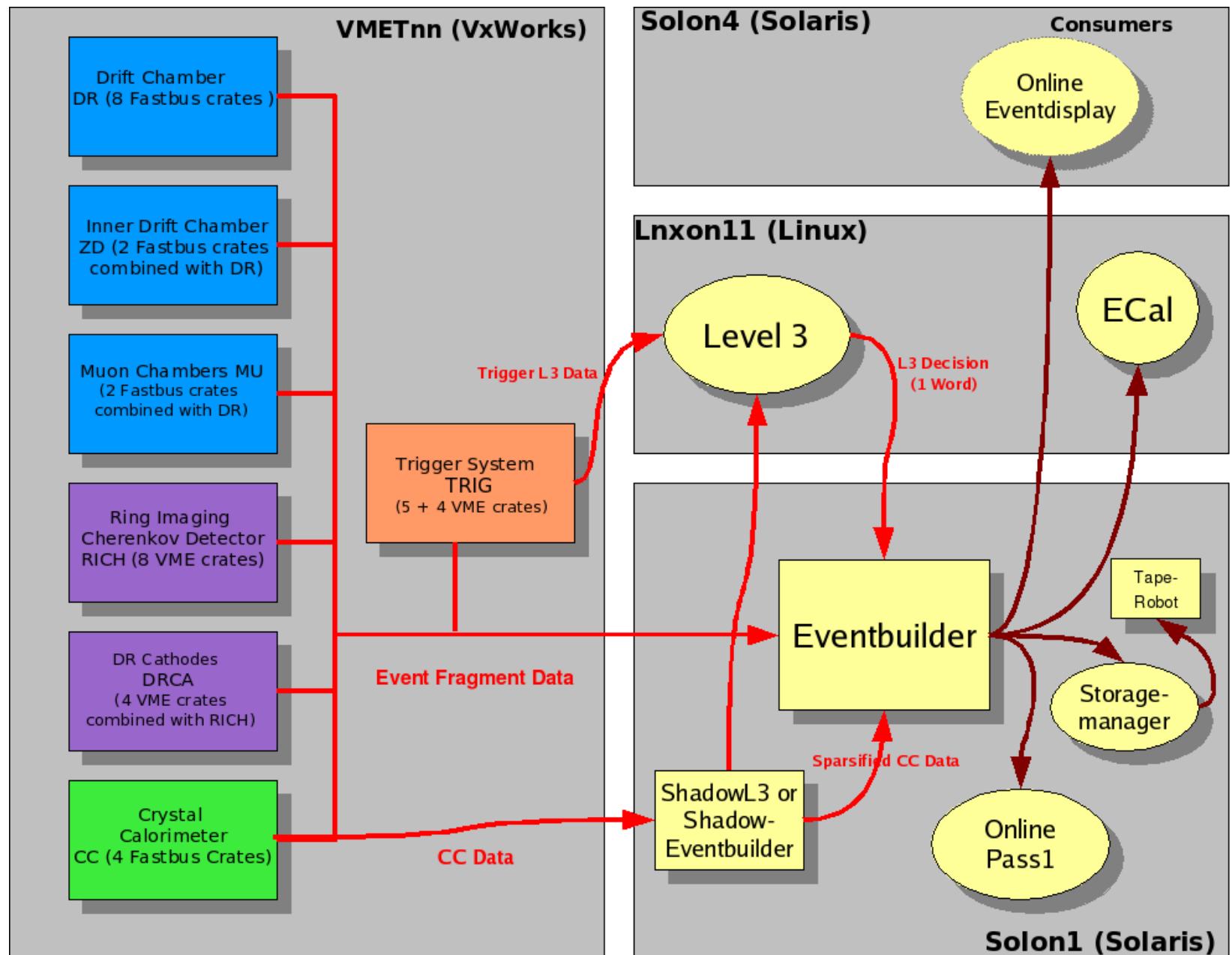
Klaus Honscheid DES SISPI Workshop

RT 95 (Michigan State): The Plan

CLEO III Data Acquisition

- ~400k channels
 - Trigger Rate 1000 Hz, 25-50 KB/event, data rate ~ 9MB/s
 - keep it simple
 - Use “standards”; most of the hardware can be bought (later!),
 - 3 hardware projects underway
 - » **FRITZ VME-FB Interface**
 - » **Databoard Buffer Manager**
 - » **Optical PCI-PCI/PMC Link**
 - Modsim simulation
 - Object oriented approach to Run Control (CORBA)
- ⇒ Done in 2 years (?)

CLEO III DAQ Architecture



Readout Controller

VME

Solution: PowerPC + VxWorks

Let Motorola worry about improvements...

1995: mv1600

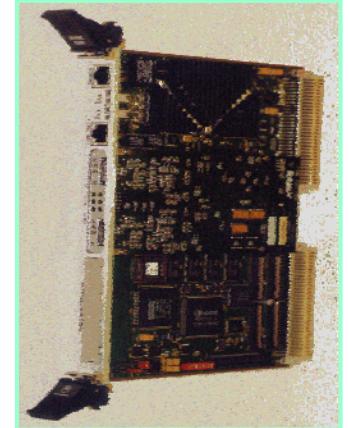
1997: mv2600

1998: mv2300

1999: mv2400

Price

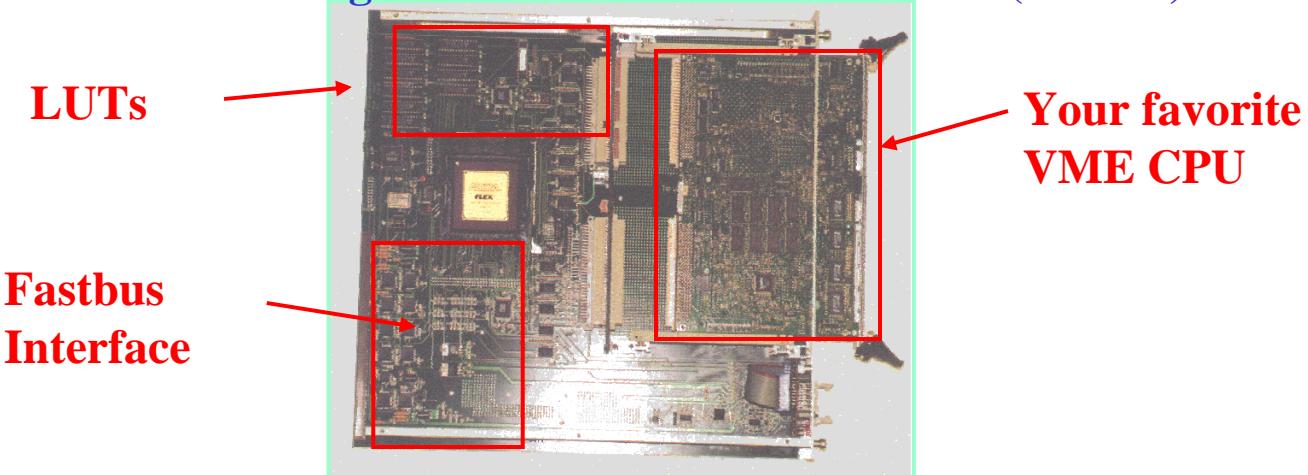
Performance



Fastbus

Not much of a market!

Solution: Design own VME-Fastbus Interface (FRITZ)



Software Choices

- Realtime OS: VxWorks
- OS: Solaris, NT (XP),
Abandoned Digital Unix after 1 year
- Middleware: CORBA
- GUI: JAVA, JAS
- Database: Objectivity
- Scripting: Tkl (configuration)

Step 1: Hardware

```
#define ADC_CHANNEL_REG 0x8100
#define ADC_VALUE_REG    0x8104
```

Step 3: Server

```
#include "VmeADC_s.hh"
class ADCImpl : public _sk_VME::_sk_ADC
{public:
    ADCImpl(const char *obj_name=NULL) :
        _sk_VME::_sk_ADC(obj_name)
    {}

    void selectChannel(CORBA::Long channel)
    {ADC_CHANNEL_REG = channel;}

    void read(CORBA::Long& value)
    {value = *ADC_VALUE_REG;}
}
```

Step 2: IDL

```
module VME
{
    interface ADC
    {
        boolean selectChannel(in long channel);
        boolean read(out long value);
    };
}
```

Step 4: Client

```
#include "VmeADC_c.hh"
int main(int argc, char ** argv)
{// get ORB reference
 // Bind to an interface
 VME::ADC_var adc;
 adc = VME::ADC::_bind(obj_name);
 adc->selectChannel(2);
 adc->read(value);
 cout << "ADC Value " << value << endl;
 return 0;
}
```

Standard Services Built on top of CORBA

Database

- » CORBA hides implementation from user
- » Objectivity 5 under Solaris (NT)

Configuration

- » Load programs from DB
- » Partition
- » Load Constants from DB

Interlocks

- » Any component can become Interlock (Source)
- » Any component can react to status changes of an Interlock
- » Trigger parallel actions (such as load constants) and wait for everyone to set interlock.

Alarms

- » central (CORBA) alarm server hides database
- » connected to Interlock system to stop data taking

Run Control

Run Statistics

- » unified interface to extract (push and/or pull) information
- » histograms
- » dynamically view any variable, histogram in the system

Local Slow Control

Common Framework:

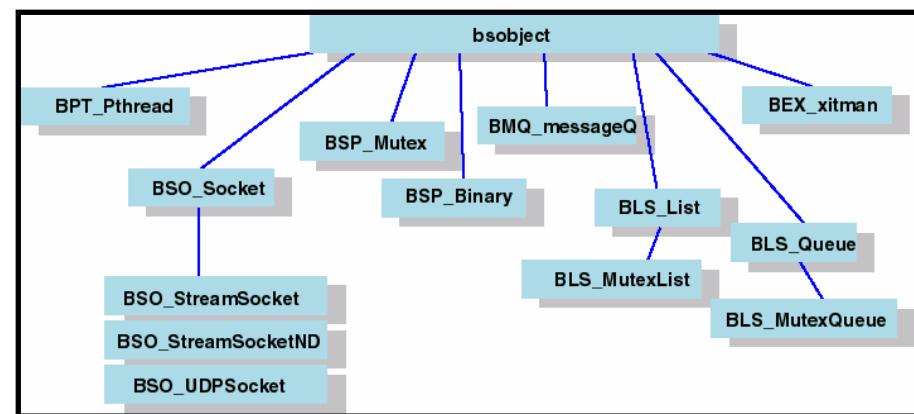
“Component Frame”

- simplifies user code development
- hides interaction with system components

Common Library:

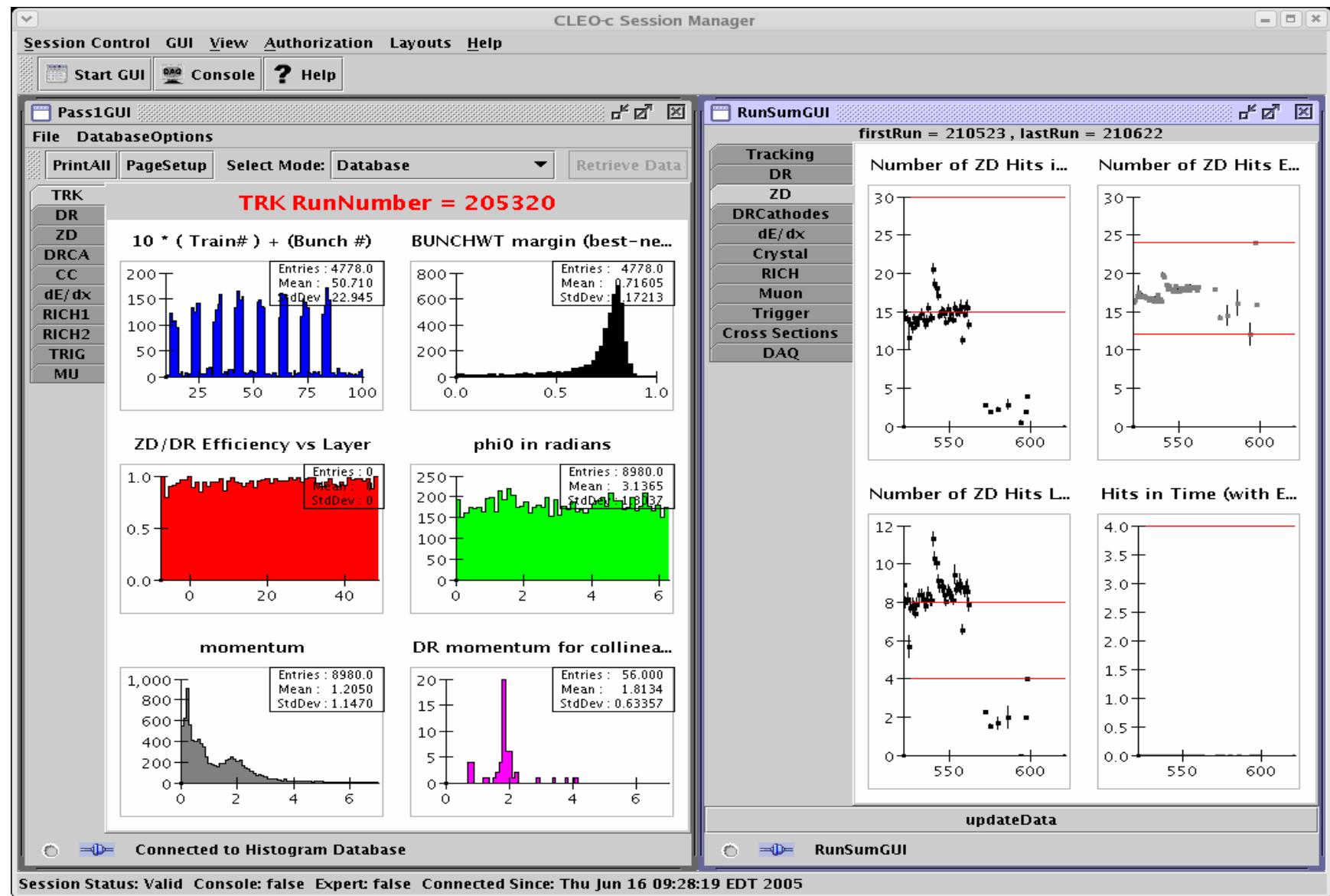
“BASICS class library”

- hides platform dependencies

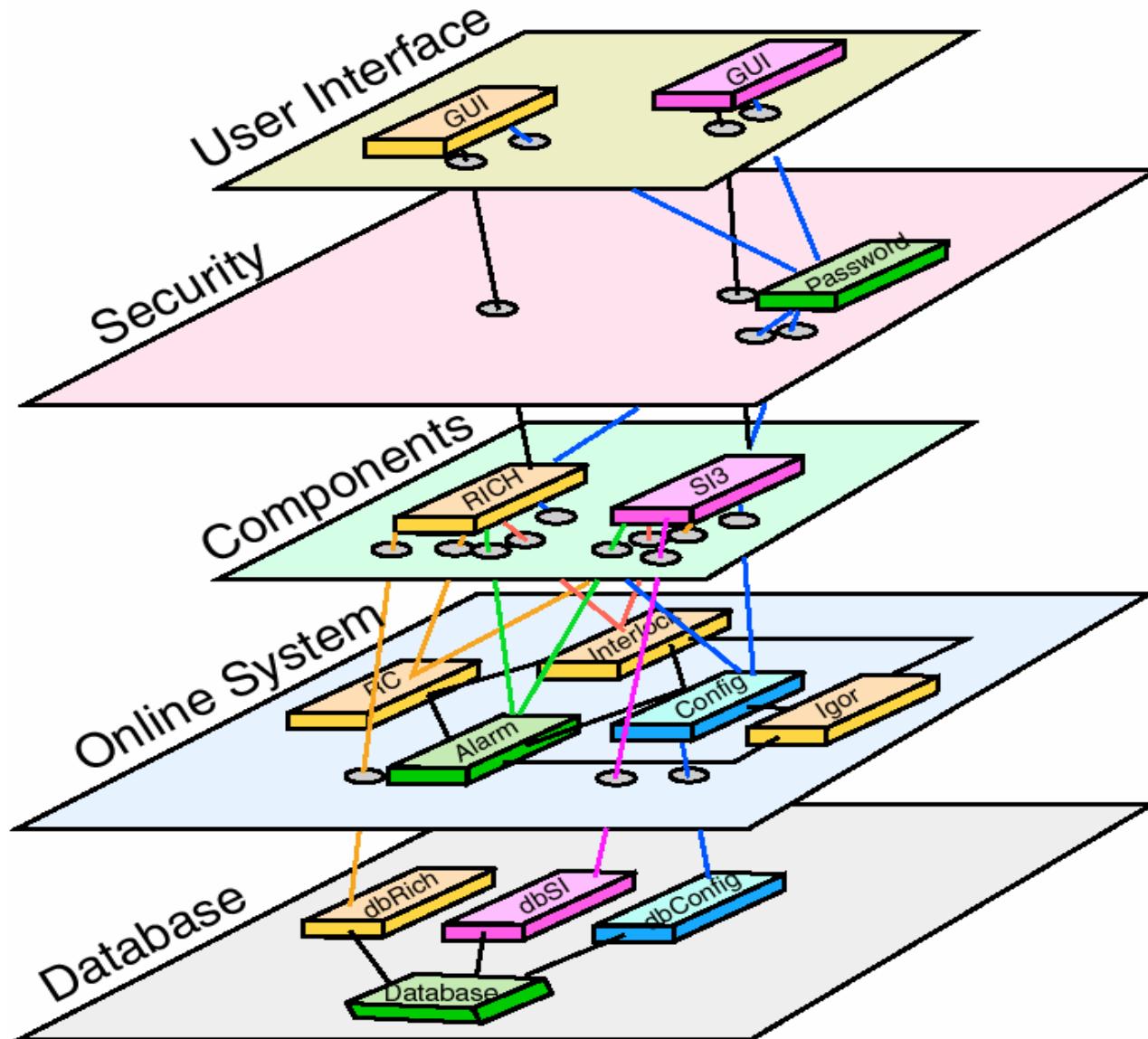


Status: complete on NT, Solaris, VxWorks

Graphical User Interfaces



CLEO III Online Software Structure



Ten Years Later: Lessons Learnt

- Use standards; most of the hardware can be bought later.
 - Choose wisely (PPC vs 68k, VME, TCPIP vs custom link)
 - Don't lock in to early (DEC Alpha)
 - Great success (Network speed, CPU speed)
 - Commercial Software ??? (Objectivity, CORBA, JAVA)
- A small, focused team can accomplish a lot
 - ~4 FTE, 4 yrs
 - free to make design choices
 - Test stand support
- Maintenance (1 FTE)