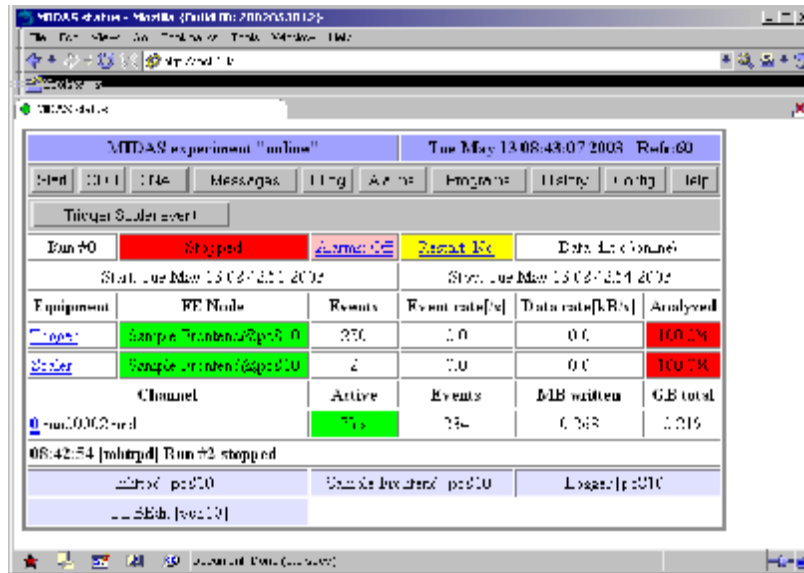


# MIDAS Short Course

*Stefan Ritt* (Paul Scherrer Institute, Switzerland)

*Pierre André Amaudruz* (TRIUMF, Canada)



## MIDAS short course

- What this is all about
  - Data acquisition (DAQ) in nuclear and particle physics
  - Basic principles of MIDAS DAQ system
  - Usage of the system
  - Tools, examples
  - Online demo
  - Goal: Be able to install, configure and run MIDAS
- Whom
  - Experimentalists with some computer knowledge
  - Test set-ups for R&D
  - Small and medium size experiments

## A few remarks

- This is a course!
  - if something is unclear, please ask immediately
  - if you have more general questions, keep them until the end of each section
  - Coffee break upon request
- Slides and example programs are on CD and available online at <http://midas.psi.ch/cgi-bin/cvsweb/midas/doc/course/>
- Basic C and data analysis knowledge is assumed
- Example programs partially simplified, miss status checking
- Authors are available during the whole conference

## The Authors

*Pierre André Amaudruz*

*Stefan Ritt*

- 1978 degree in electr. engineering at the Technical Institute, Geneva, Switzerland
- 1979-1981 BBC, Switzerland
- 1981-1988 SIN (PSI), MWPCs hardware, DAQ (PDP11+VAX)
- 1988-1993 CHAOS project at TRIUMF
- since 1993 staff member of DAQ group at TRIUMF

- 1993 Ph.D. in Physics at the University of Karlsruhe, Germany
- 1993-2000 postdoc at University of Virginia, project manager of  $\pi\beta$  experiment
- since 2000 research scientist (staff) at PSI, technical coordinator of MEG experiment
- Other interests: Chip design, control systems

# Paul Scherrer Institute



590 MeV Protons

Energy research

Proton irradiation

Synchrotron light source SLS



## Agenda

### Part I

- Introduction
- Internal structure
  - Remote Procedure Calls
  - Buffer Manager
  - Online Database
  - Slow Control System
  - History System
  - Alarm System
- Frameworks
  - Frontend
  - Analyzer

### Part II

- Online demo
  - Installation
  - Configuration
  - Running a small experiment
- Utilities and usage
- Known deployment
- Advanced features
  - Customization of run control and monitoring
  - Special front-end features
- Advanced online demo

# Why MI DAS?

- Fulfill needs of certain experiments (~thousand channels, ~MB data rate)
- Incorporated slow control system
- Integrated data analysis (histogramming)
- Operating system independent
- Hardware independent
- Quick installation
- Flexible, useful for other experiments
- Easy application programming
- Free (GPL)

# What is MI DAS?

- DAQ Framework
  - Event transport (multiple producers & consumers)
  - Central configuration
  - Frontend and analyzer framework
  - You have to write "user code"
- C library + applications
- Contains logging, slow control, alarm system, electronic logbook: **M**aximum **I**ntegrated **D**ata **A**cquisition **S**ystem
- Suitable for small tests/experiments to medium size experiments, not (?) for HEP experiments
- Not to be confused with ESO-Midas, Midas/UK, Auto service... Let's call it "MI DAS/PSI "

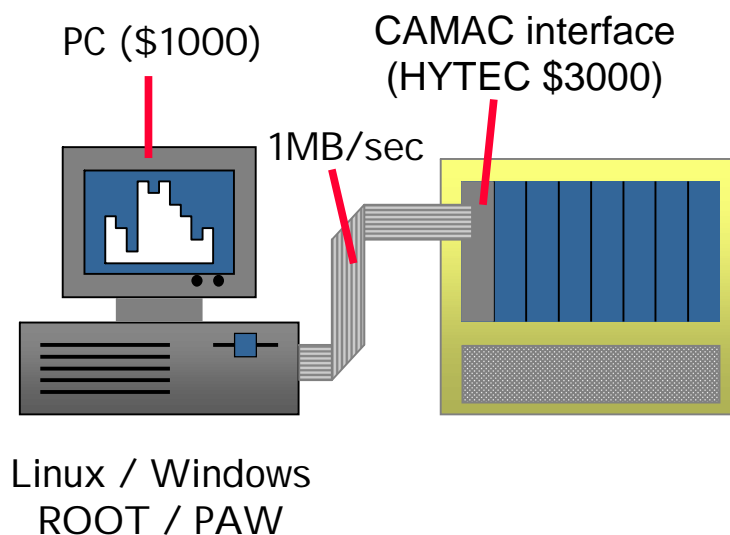
# What is MIDAS?

- ~100.000 lines of C code
- Optimized for event-based DAQ, typically nuclear and particle physics
- Contains driver library for CAMAC, VME, FASTBUS
- MIDAS is free and under the GPL
- Two "main" developers (S. Ritt and P. Amaudruz)
- Support by PSI and TRIUMF

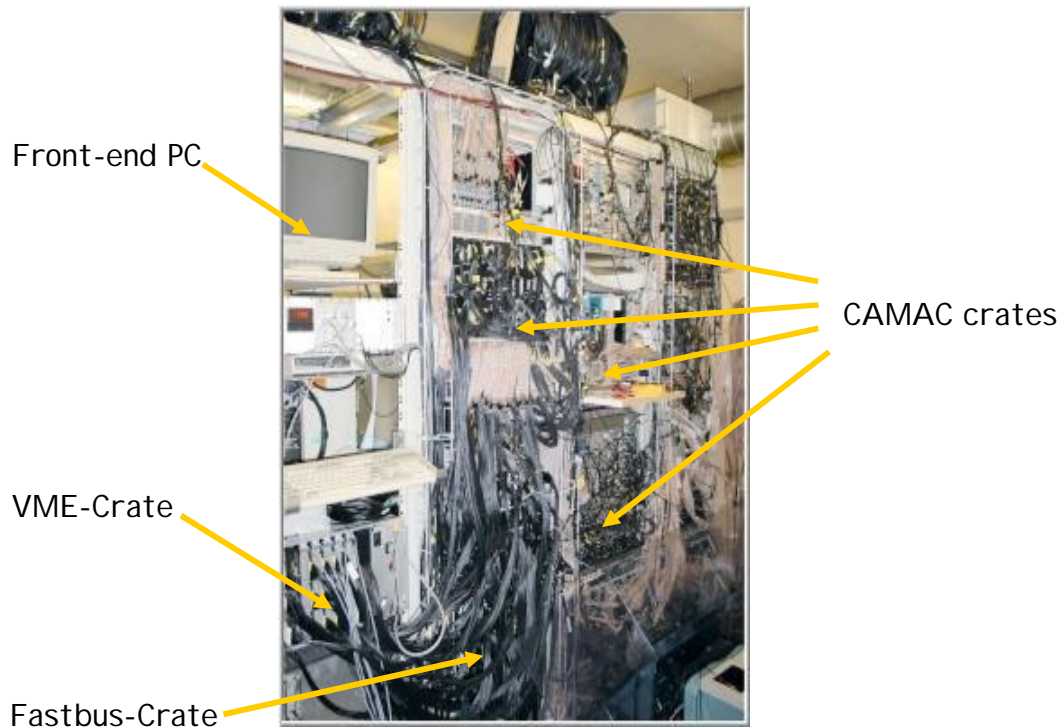
<http://midas.psi.ch>  
stefan.ritt@psi.ch

<http://midas.triumf.ca>  
amaudruz@triumf.ca

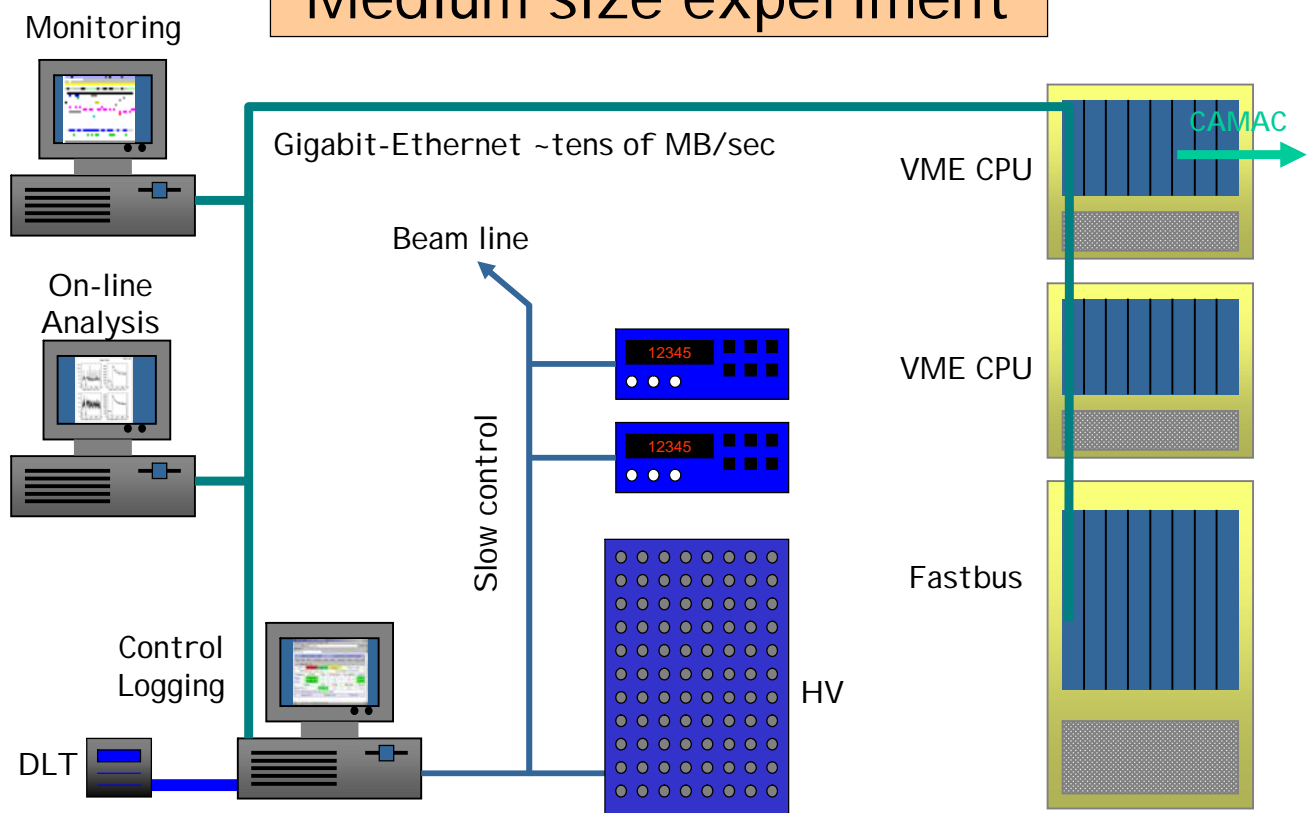
## Small (test) system



# $\pi\beta$ Experiment @ PSI



## Medium size experiment



# Limits of MIDAS

# Remote Procedure Calls

## Platform independence

Operating system  
independent calls

93%

### Buffer Manager

```
bm_open_buffer()  
bm_send_event()  
...
```

### Online Database

```
db_find_key()  
db_set_value()  
...
```

System call layer

7%

`ss_open_shared_memory()`

conditional compiling

### Windows NT

```
OpenFileMapping()
```

### VMS

```
ppl$create_  
shared_memory()
```

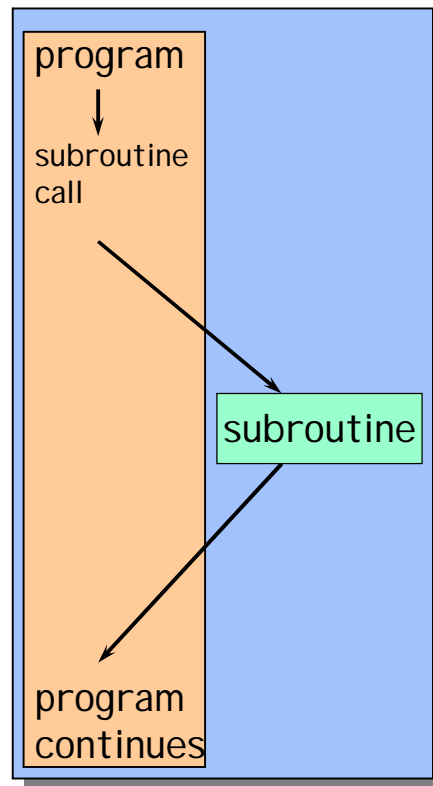
### UNIX

```
shmget()
```



# MI DAS RPCs

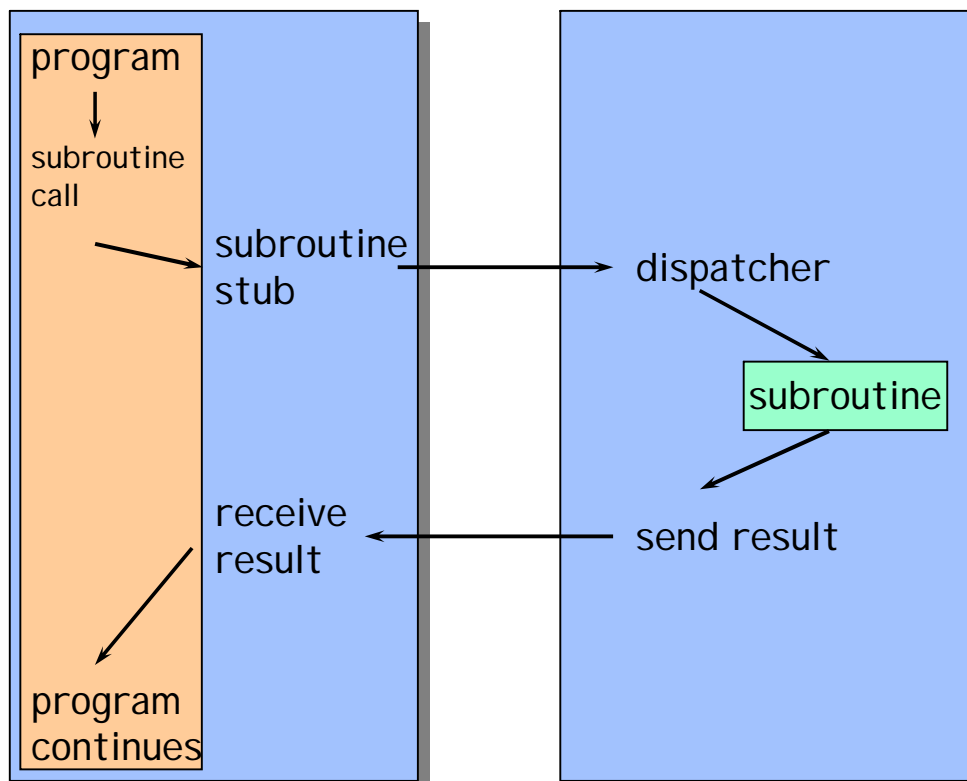
Computer A



# MI DAS RPCs

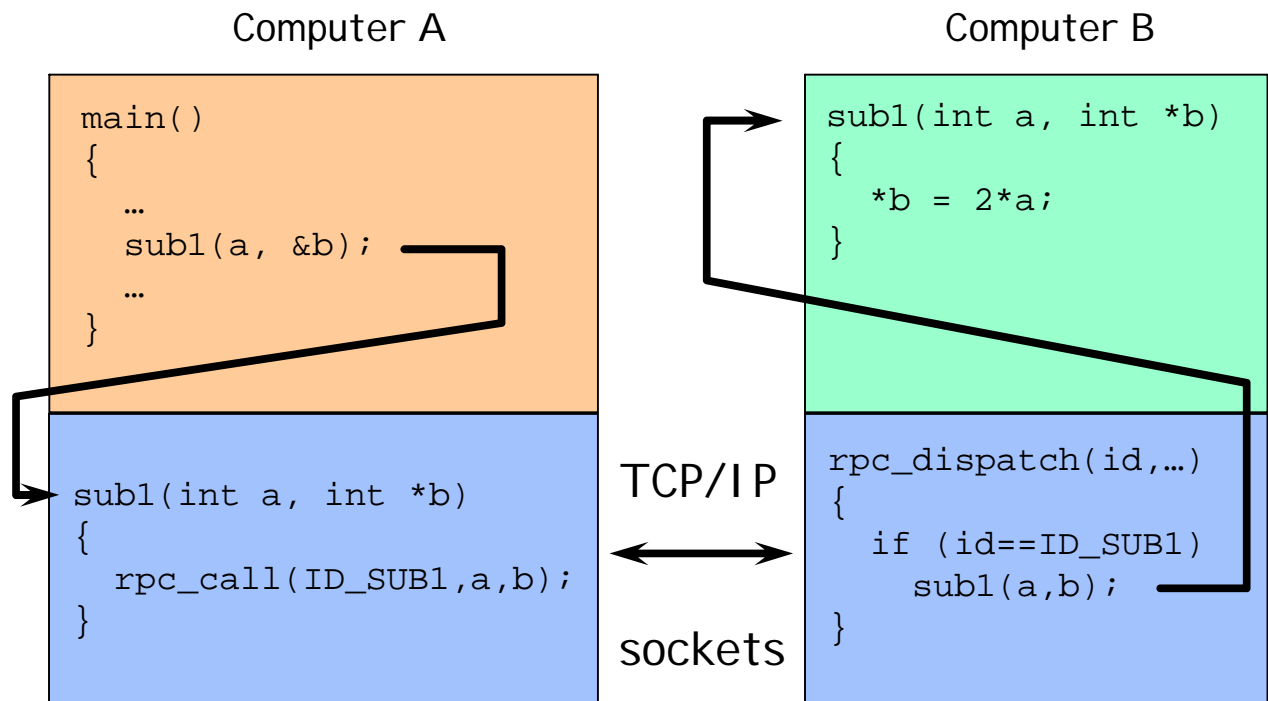
Computer A

Computer B

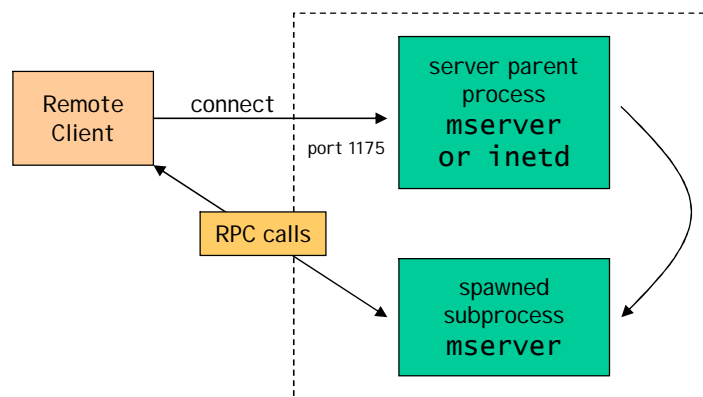


# MI DAS RPCs

No IDL, stubs are produced "by hand"



## MI DAS server



- Each client connected to own subprocess
- Access control through (x)inetd and optional mserver password
- RPC layer completely transparent -> exactly same code if running local or remotely
- Little/big endian negotiation/conversion

	adr0	adr1	
Motorola: big-endian	0x12	0x34	= 0x1234
Intel: little-endian	0x34	0x12	

# Why not standard RPCs?

- Availability (VxWorks, RT-Linux, ...)
- Performance
  - RPC layer is speed optimized
  - Data representation is negotiated
  - Some functions like send\_event can bypass part of the standard RPC-layer for buffered and asynchronous RPC
  - -> 11MB/sec over 100MBit Ethernet
- Dynamic extensions at run-time
- Asynchronous notification (-> callbacks)
- No IDL compiler necessary, simple syntax

## User-level RPCs

**Client**

```
#include "midas.h"
#include <stdio.h>

#define RPC_MYTEST 1

RPC_LIST rpc_list[] = {
  { RPC_MYTEST, "rpc_mytest",
    {{TID_INT,      RPC_IN},
      {TID_FLOAT,   RPC_IN},
      {TID_INT,     RPC_OUT},
      {TID_FLOAT,   RPC_OUT},
      {0} }},
  { 0 }
};

main()
{
  char  host_name[80];
  HANDLE hConn;
  INT  status;
  INT  i;
  float f;

  printf("Remote host name: ");
  ss_gets(host_name, sizeof(host_name));

  /* register this as an RPC client with rpc_list */
  rpc_register_client("MYCLIENT", rpc_list);

  /* connect to RPC server */
  rpc_client_connect(host_name, 1750, "", &hConn);

  /* rpc_mytest just doubles numbers */
  rpc_client_call(hConn, RPC_MYTEST, 1, 2, &i, &f);

  printf("\nResult should be: 2 4.0\n");
  printf("Result is:          %d %1.1f\n", i, f);

  /* disconnect this client from server */
  rpc_client_disconnect(hConn, FALSE);
  return 1;
}
```

**Server**

```
#include "midas.h"
#include <stdio.h>

#define RPC_MYTEST 1

RPC_LIST rpc_list[] = {
  { RPC_MYTEST, "rpc_mytest",
    {{TID_INT,      RPC_IN},
      {TID_FLOAT,   RPC_IN},
      {TID_INT,     RPC_OUT},
      {TID_FLOAT,   RPC_OUT},
      {0} }},
  { 0 }
};

INT rpc_mytest(INT i, float f, INT *i1, float *f1)
{
  *i1 = i*2; *f1 = f*2;
  return 1;
}

INT rpc_dispatch(INT index, void *prpc_param[])
{
  INT status;

  if (index == RPC_MYTEST)
    return rpc_mytest(CINT(0), CFLOAT(1), CPINT(2), CPFLOAT(3));
  return RPC_INVALID_ID;
}

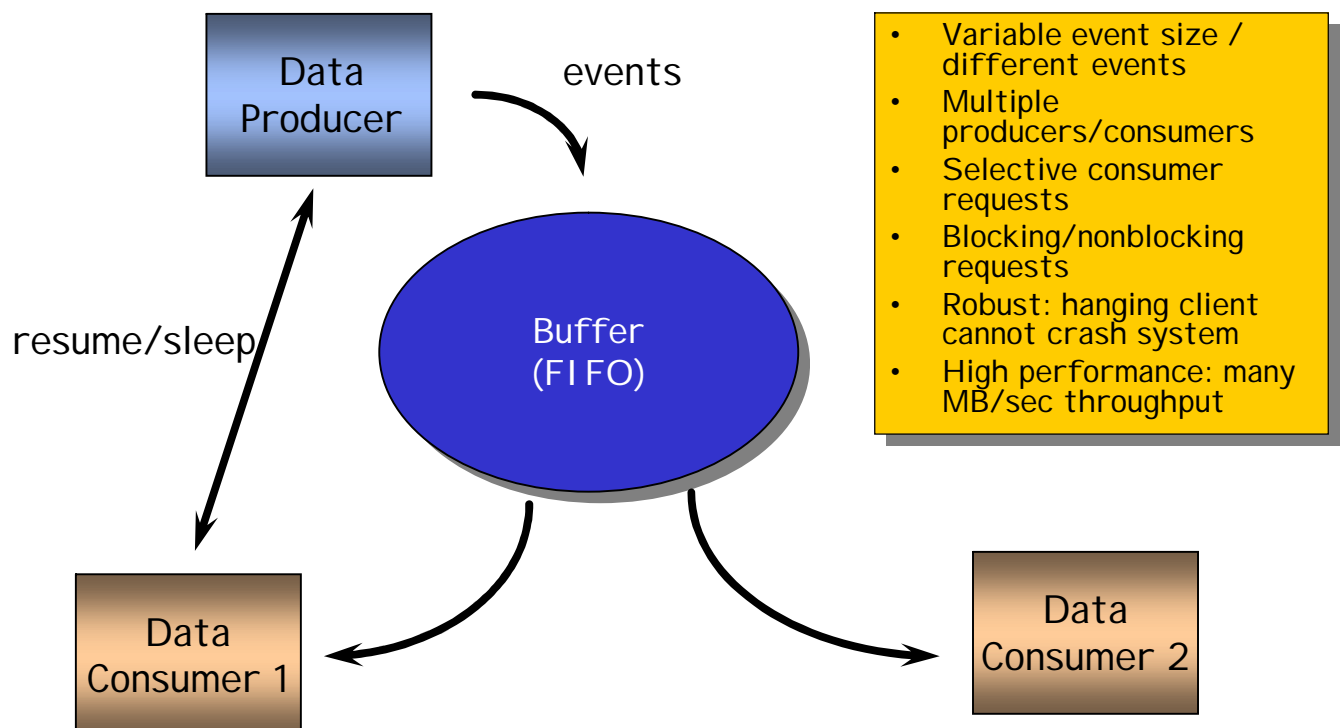
main()
{
  rpc_register_server(ST_REMOTE, "", 1750, NULL);

  /* Register function list. Calls get forwarded to rpc_dispatch */
  rpc_register_functions(rpc_list, rpc_dispatch);

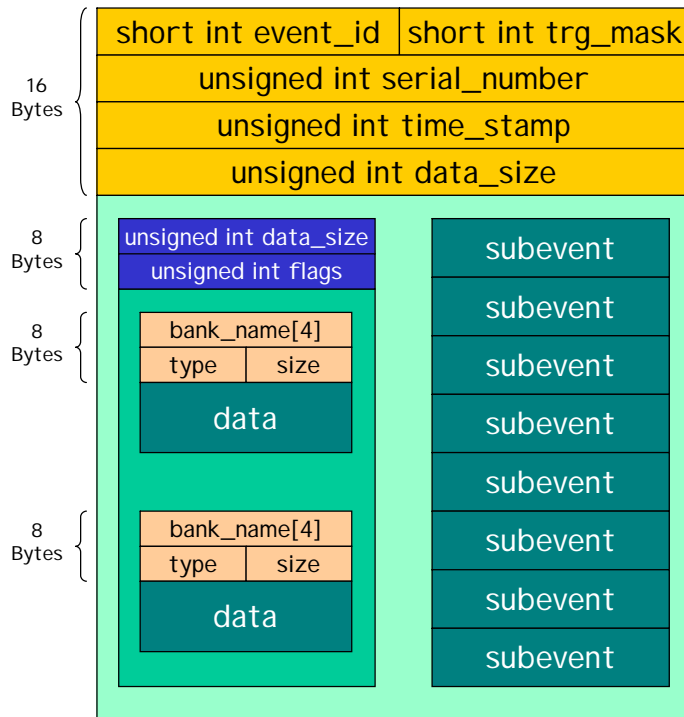
  /* Server loop */
  while (cm_yield(1000) != RPC_SHUTDOWN);
  rpc_server_shutdown();
  return 1;
}
```

# The MIDAS Buffer Manager

## Buffer Manager



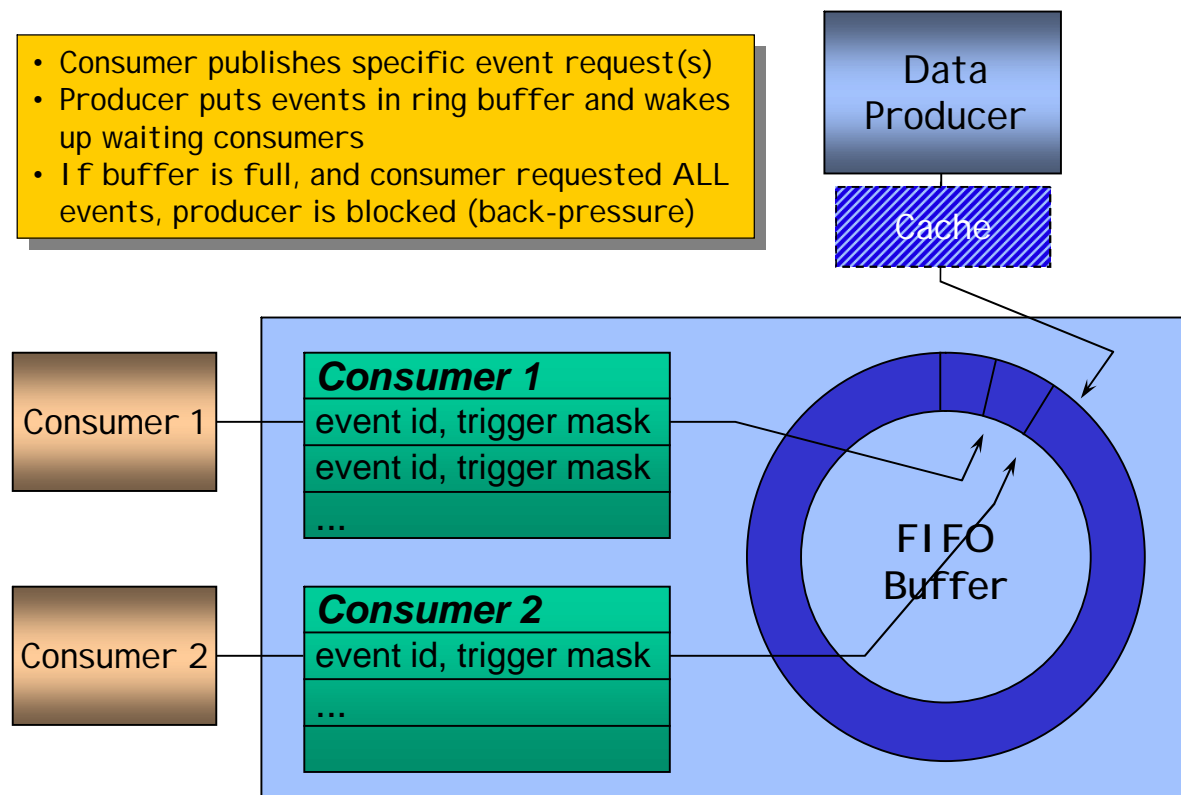
# Event Structure



- 16-byte header
- User data area free format
- Optional MI DAS bank system
- Optional YBOS bank system
- Optional Subevent system
- Default max. event size 512k

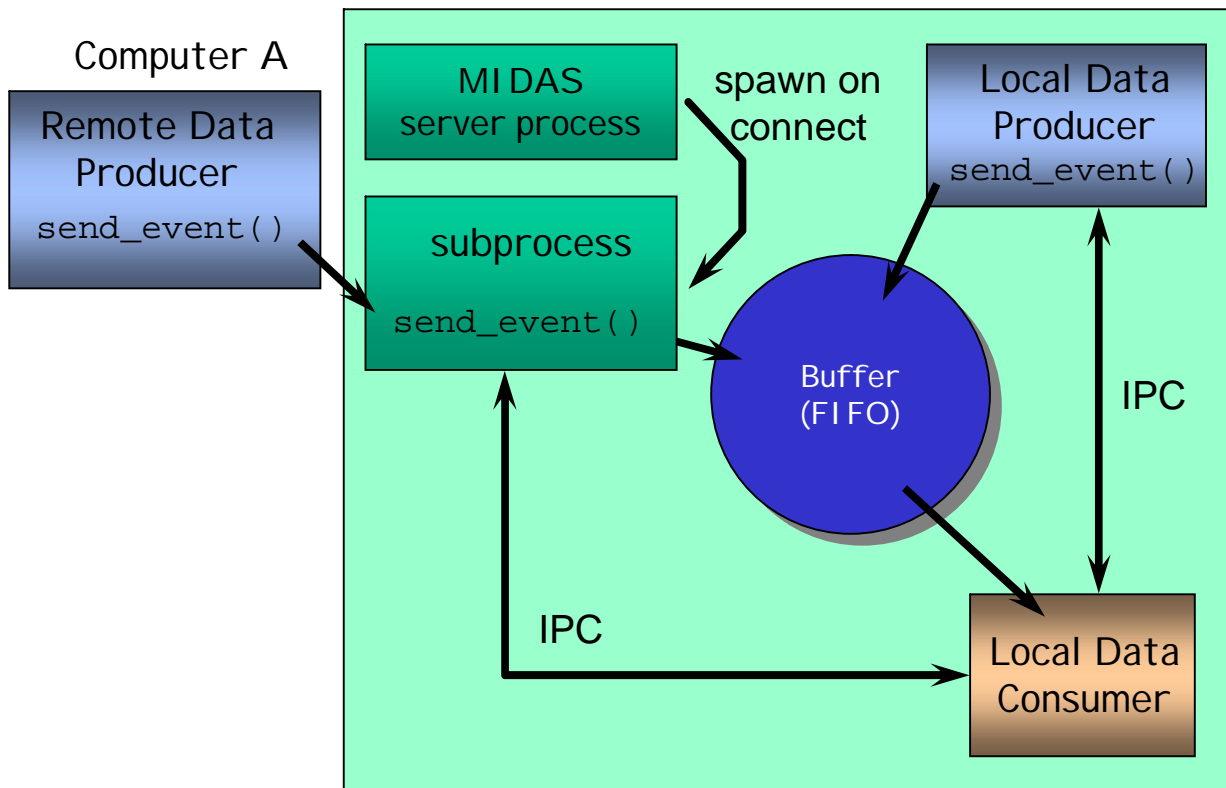
# Event Requests

- Consumer publishes specific event request(s)
- Producer puts events in ring buffer and wakes up waiting consumers
- If buffer is full, and consumer requested ALL events, producer is blocked (back-pressure)

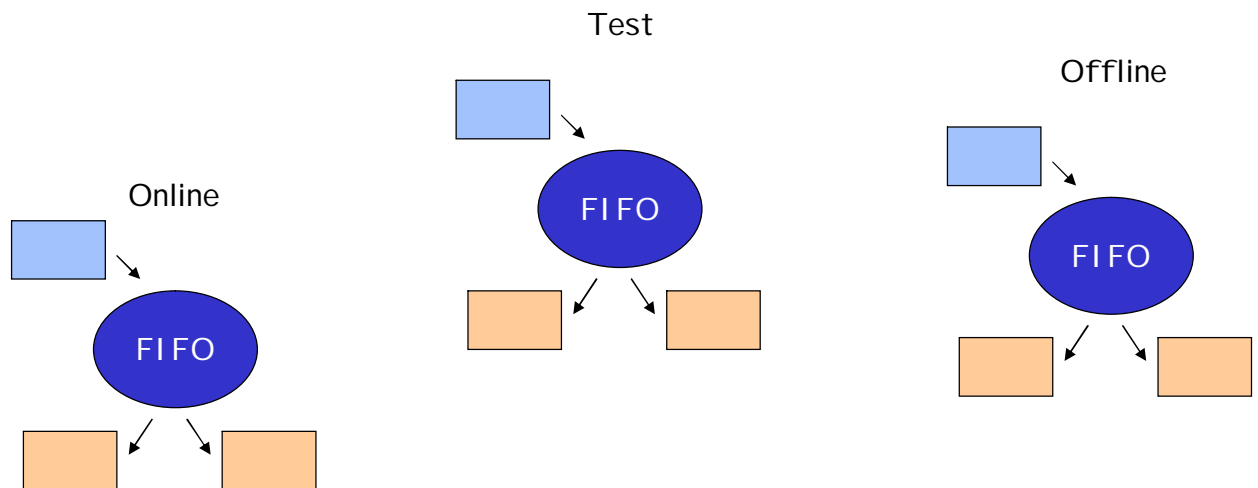


# Remote Access to BM

Computer B



## Multiple Experiments



/etc/exptab

```
online /home/midas/online user1
offline /home/midas/offline user1
test /home/test test
```

- Multiple independent experiments
- Experiment name associated with directory on file system
- Explicitly connect to experiment

# Watchdog Scheme



```
#include <stdio.h>
#include <stdlib.h>
#include "midas.h"

#define EVENT_SIZE 1000

main()
{
    INT hBuf, status, start, stop, bytes_sent, serial_number=1;
    char event[EVENT_SIZE], host_name[256];

    printf("Host to connect: ");
    ss_gets(host_name, 256);

    /* connect to experiment */
    cm_connect_experiment(host_name, "", "Producer", NULL);

    /* open the event buffer with default size */
    bm_open_buffer("SYSTEM", 0x100000, &hBuf);

    /* set the buffer write cache size */
    bm_set_cache_size(hBuf, 0, 50000);

    do
    {
        start = ss_millitime();
        bytes_sent = 0;

        do
        {
            /* compose an event header with increasing serial number */
            bm_compose_event((EVENT_HEADER *) event, 1, 1,
                           EVENT_SIZE, serial_number++);

            /* send event */
            rpc_send_event(hBuf, event, EVENT_SIZE+sizeof(EVENT_HEADER), SYNC);

            bytes_sent += EVENT_SIZE;

            /* repeat this loop for 1s */
        } while (ss_millitime() - start < 1000);

        /* Now calculate the rate */
        stop = ss_millitime();
        if (stop != start)
            printf("Rate: %1.21f MB/sec\n",
                  bytes_sent/1024.0/1024.0/(stop/1000.0 - start/1000.0));

        status = cm_yield(0);
    } while (status != RPC_SHUTDOWN && status != SS_ABORT);

    cm_disconnect_experiment();
    return 1;
}
```

```
#include <stdio.h>
#include <stdlib.h>
#include "midas.h"

void process_event(HANDLE hBuf, HANDLE request_id, EVENT_HEADER *pheader,
                  void *pevent)
{
    static INT bytes_received, start=0, stop;

    /* accumulate received event size */
    bytes_received += pheader->data_size;

    /* calculate rates each second */
    if (ss_millitime() - start > 1000)
    {
        stop = ss_millitime();
        if (stop != start)
            printf("Rate: %1.21f MB/sec\n",
                  bytes_received/1024.0/1024.0/(stop/1000.0 - start/1000.0));

        start = stop;
        bytes_received = 0;
    }
}

main()
{
    INT hBuf, status, request_id;

    /* connect to experiment */
    cm_connect_experiment("", "", "Consumer", NULL);

    /* open the event buffer */
    bm_open_buffer("SYSTEM", 0x100000, &hBuf);

    /* set the buffer cache size */
    bm_set_cache_size(hBuf, 50000, 0);

    /* place a request for a specific event id "1" */
    bm_request_event(hBuf, 1, TRIGGER_ALL, GET_ALL, &request_id,
                    process_event);

    do
    {
        status = cm_yield(1000);
    } while (status != RPC_SHUTDOWN && status != SS_ABORT);

    cm_disconnect_experiment();
    return 1;
}
```

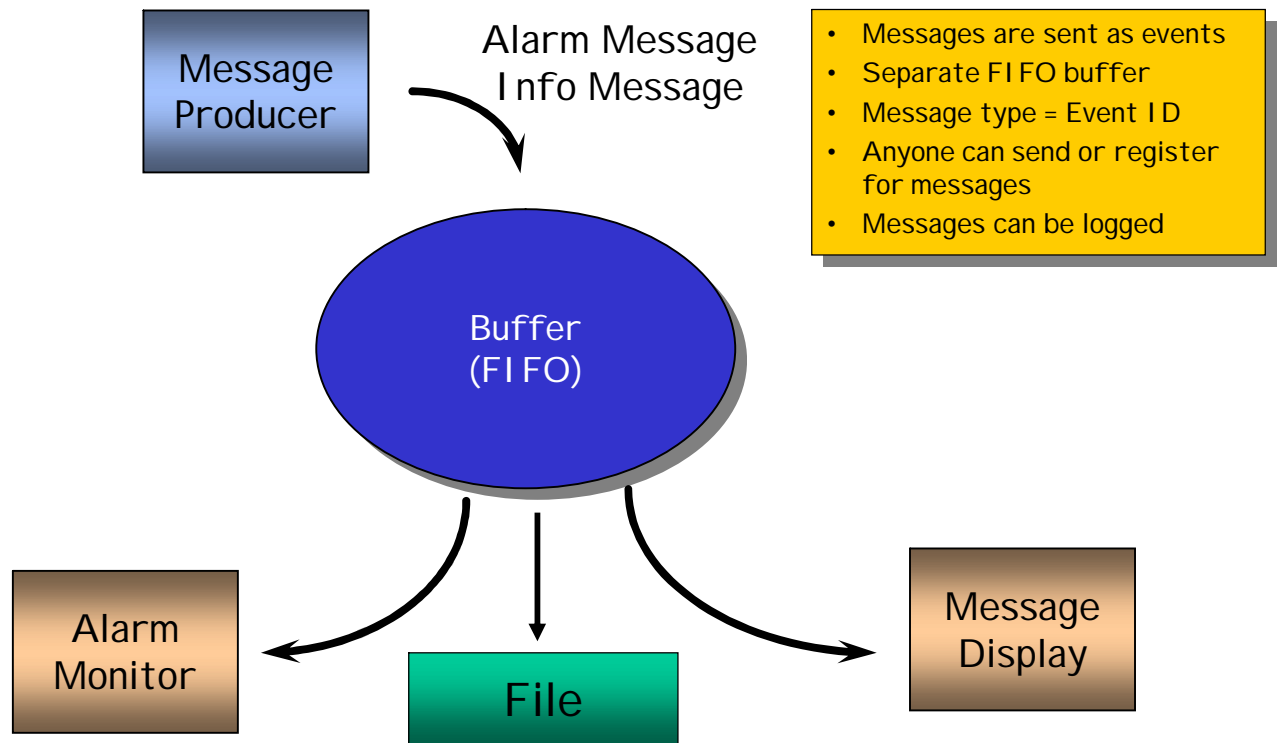


## Performance

- Locally (1.4 GHz Athlon, 1kB event size)
  - 160 MB/sec (one producer, one consumer)
  - 80 MB/sec (one producer, two consumers)
  - 5 MB/sec (10 Byte events)
- 100Mbit Ethernet
  - 10MB/sec
- 1Gbit Ethernet:
  - limited by TCP/IP stack
  - socket API : 30-50MB/sec
  - zero-copy TCP/IP stack (send\_file): under development, 2x possible



# Message System



## Online Database (ODB)

# Why database?

- Store all variable experiment data
  - Run number
  - Logging channel information
  - Event definition
  - Front-end parameters (e.g. crate address)
  - Analyzer parameters (e.g. pedestal values)
  - Info about running programs
  - Slow control values (HV, beam line)
- Central database advantages
  - "Single point of configuration"
  - Avoid inconsistencies
  - Easy backup
  - Simple API for controlling the whole experiment

# Requirements

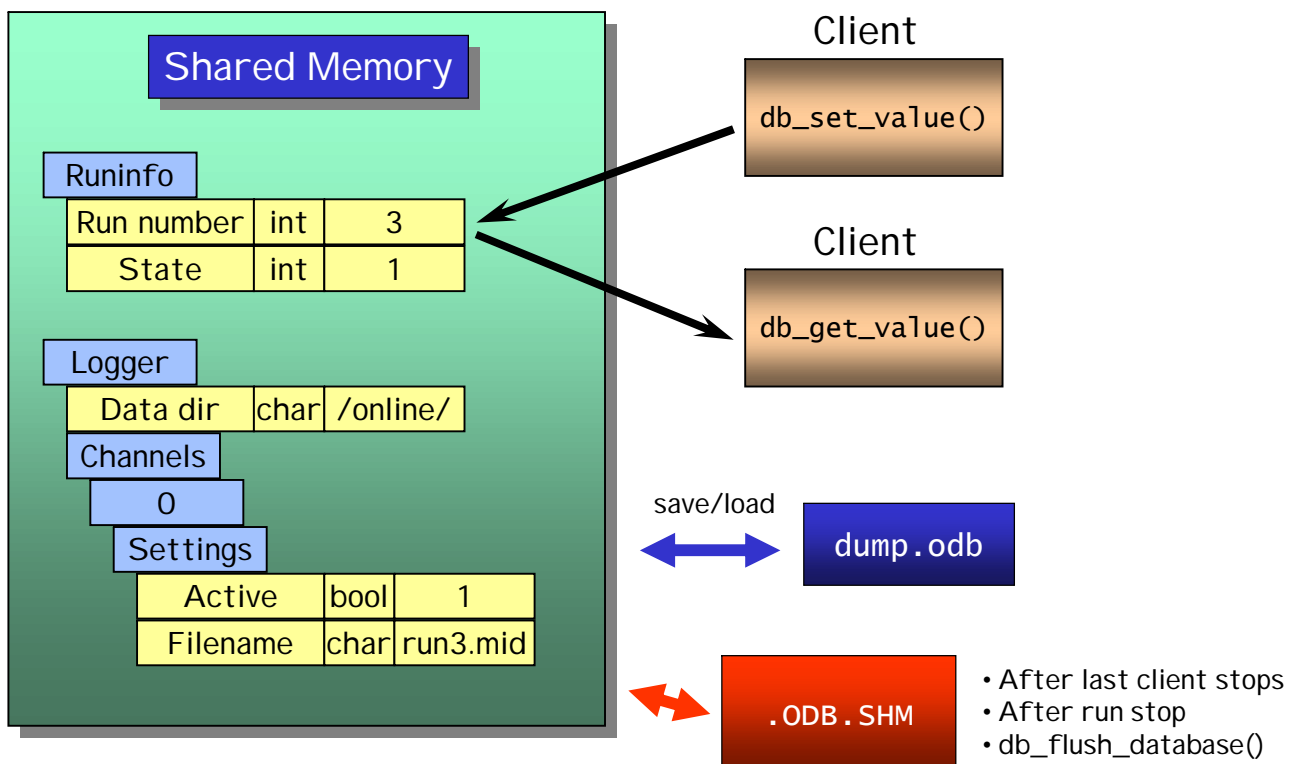
- Fast access (>100.000/s), remote access
- Tagged data: name, type, value
- Hierarchical structure (/proc, Windows registry)
- Dynamical creation and deletion of keys
- ASCII import/export
- Automatic notification on data change

*-> Requirements are not met by standard  
(disk based) database*

# Solution: ODB

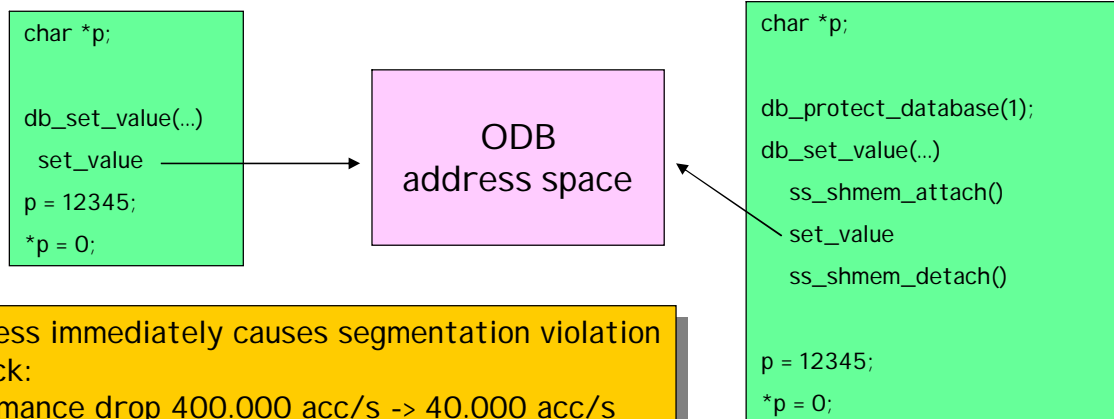
- Resides completely in shared memory
- Access via MIDAS RPCs
- Similar to file system
  - File name -> "key"
  - File contents -> key value (single value or array)
  - Directory -> key list
  - Symbolic link -> key link
  - Access flags: read/write/delete
- Data change notification via RPC callback
- Persistency through disk image
- Access through API and tools (ODBCedit)

## ODB Structure



# Corrupting the ODB

- ODB has to be mapped into process address space
- Misbehaving process (e.g. invalid pointer) can corrupt ODB
- For debugging, all db\_xxx routines can map/unmap address space



Bad access immediately causes segmentation violation  
 Drawback:  
 Performance drop 400.000 acc/s -> 40.000 acc/s  
 Watchdog scheme doesn't work anymore

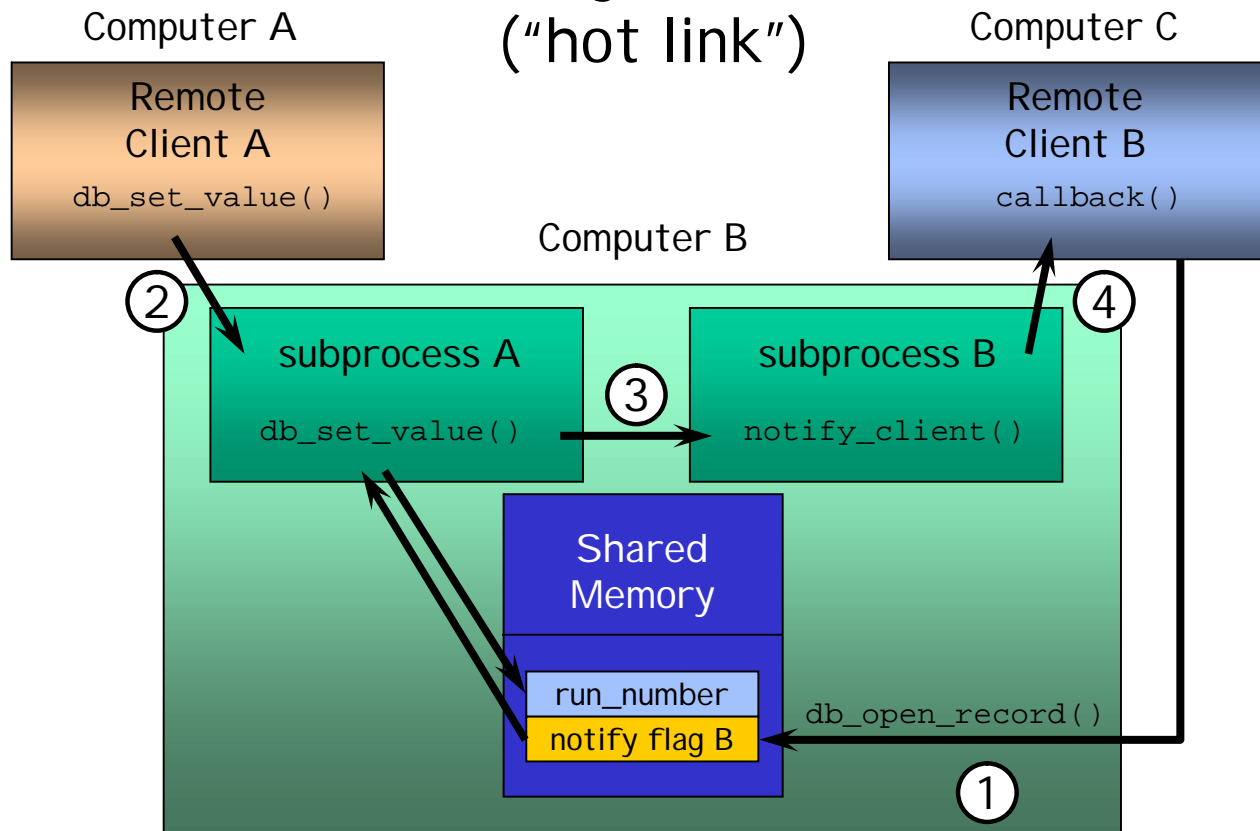
# ODB ASCII format

```
[Experiment/Runinfo]
State           = INT : 1
Online          = BOOL : 1
Run number      = DWORD : 10
Start time      = STRING : [25] Wed Sep 04 10:03:47 1996

[Experiment/Runinfo/Flags]
Clear histos    = BOOL : 1
Save histos     = BOOL : 1

[Experiment/Analyzer parameters]
ADC pedestal    = DOUBLE[4] :
1.2
4.3
2.5
5.6
Offset          = DOUBLE : 9.5
```

# Data change notification ("hot link")



## ODBEdit

- Simple "wrapper" around MIDAS library
- Old style command line interface
- Inspect/change ODB, create new keys
- Start/stop runs
- Shutdown clients
- Import/Export ODB to ASCII, C structure
- Check ODB structure ("cleanup")
- Simple chat functionality
- Change passwords
- Works locally and remotely (-h <hostname>)

# Example ODB Access

```
#include <stdio.h>
#include "midas.h"

int run_number;

void run_number_changed(HNDLE hDB, HNDLE hKey, void *info)
{
    printf("Run number is %d\n", run_number);
}

main()
{
    HNDLE hkey;
    int size, status;

    /* connect to experiment */
    cm_connect_experiment("", "", "ODB Test", NULL);

    /* get run number */
    size = sizeof(run_number);
    db_get_value(1, 0, "/runinfo/run number",
                &run_number, &size, TID_INT, TRUE);

    run_number++;

    /* write run number */
    db_set_value(1, 0, "/runinfo/run number",
                &run_number, size, 1, TID_INT);

    /* open hot link to run number */
    db_find_key(1, 0, "/runinfo/run number", &hkey);
    db_open_record(1, hkey, &run_number, sizeof(run_number),
                MODE_READ, run_number_changed, NULL);

    /* enter idle loop */
    while (cm_yield(1000); != RPC_SHUTDOWN);

    cm_disconnect_experiment();
    return 1;
}
```

Callback routine gets called  
whenever run number changes

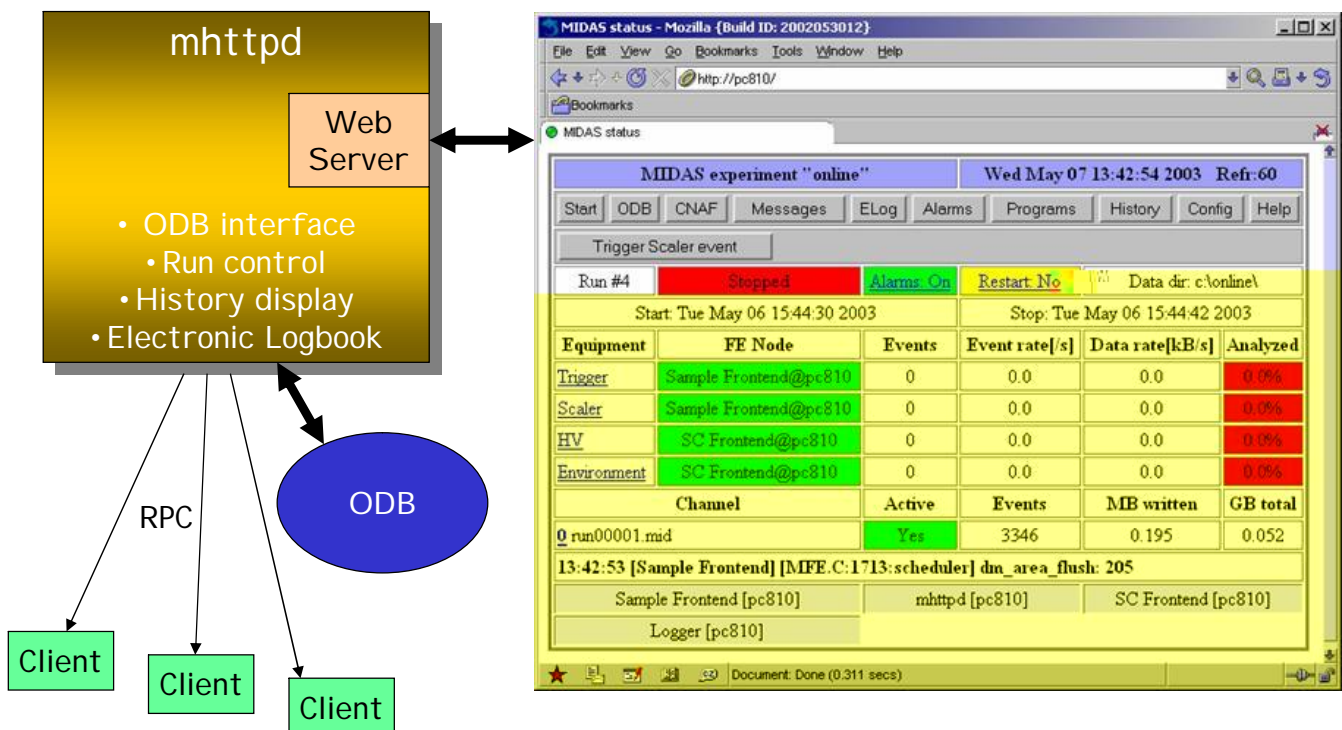
Read, modify and write run\_number

Register "hot link"

Event loop



# Web interface



# Hot link to C structure

## Application

## ODB

```
struct {
```

```
    int run_number;
```

```
    int state;
```

```
} runinfo;
```

Runinfo

Run number	int	3
State	int	1

-> Problem if ODB structure doesn't match C structure

### Solution:

- correct ODB structure before hot-link is opened
- lock ODB structure as long as hot-link is open

# Hot link to C structure example

runinfo.h

```
typedef struct {
    INT    state;
    INT    online_mode;
    INT    run_number;
    INT    transition_in_progress;
    INT    requested_transition;
    char   start_time[32];
    DWORD  start_time_binary;
    char   stop_time[32];
    DWORD  stop_time_binary;
} RUN_INFO;

#define RUN_INFO_STR(_name) char *_name[] = {\
    "[.]",\
    "State = INT : 1",\
    "Online Mode = INT : 1",\
    "Run number = INT : 0",\
    "Transition in progress = INT : 0",\
    "Requested transition = INT : 0",\
    "Start time = STRING : [32] Tue Sep 09 15:04:42 1997",\
    "Start time binary = DWORD : 0",\
    "Stop time = STRING : [32] Tue Sep 09 15:04:42 1997",\
    "Stop time binary = DWORD : 0",\
    "",\
    NULL }
```

- Create ASCII representation of C structure
- Use ASCII representation to create/correct ODB structure

```
#include <stdio.h>
#include "midas.h"

#include "run_info.h"

RUN_INFO run_info;

RUN_INFO_STR(run_info_str);

void run_info_changed(HNDLE hDB, HNDLE hKey, void *info)
{
    printf("Run number is %d\n", run_info.run_number);
}

main()
{
    HNDLE hKey;
    int status;

    /* connect to experiment */
    cm_connect_experiment("", "", "ODB Test", NULL);

    /* create/correct /runinfo structure */
    db_create_record(1, 0, "/Runinfo", strcomb(run_info_str));

    /* open hot link to runinfo number */
    db_find_key(1, 0, "/runinfo", &hKey);
    db_open_record(1, hKey, &run_info, sizeof(run_info),
        MODE_READ, run_info_changed, NULL);

    /* enter idle loop */
    do
    {
        status = cm_yield(1000);
    } while (status != RPC_SHUTDOWN);

    cm_disconnect_experiment();
    return 1;
}
```

47/100  
]/FE>cd "T

typedie{dNT

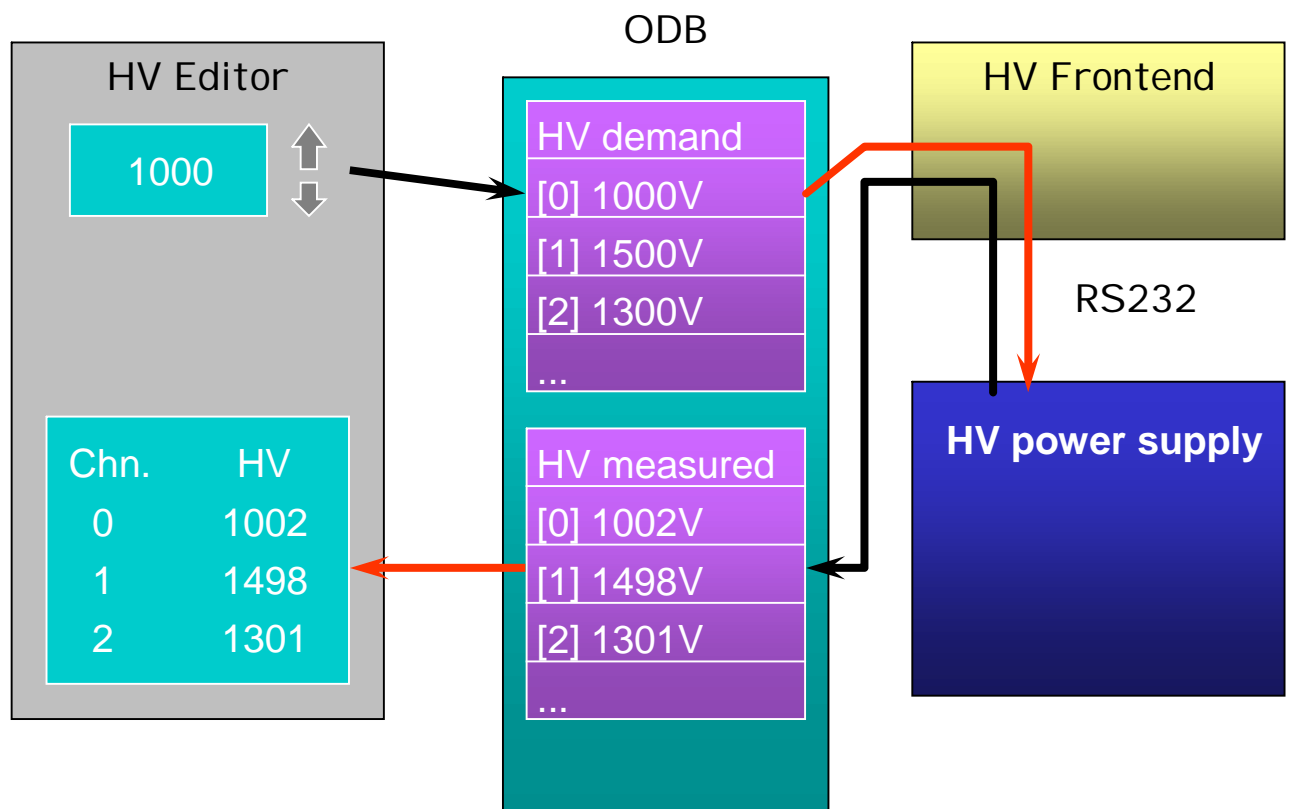
#define TRIGGER\_L\_CTONES (sizeof (char) \* 256 - 1) / 2

#define

/\*> mics



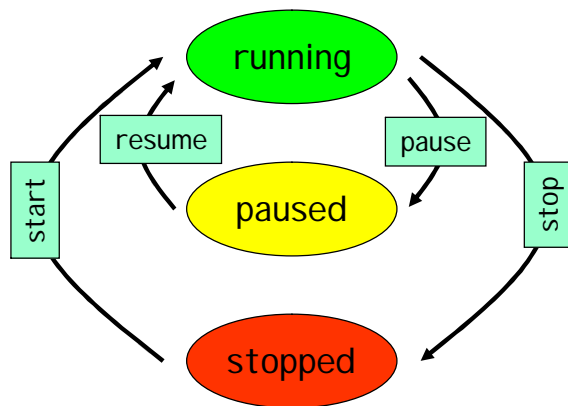
# Slow Control System



## Run Transitions

# Run Transitions

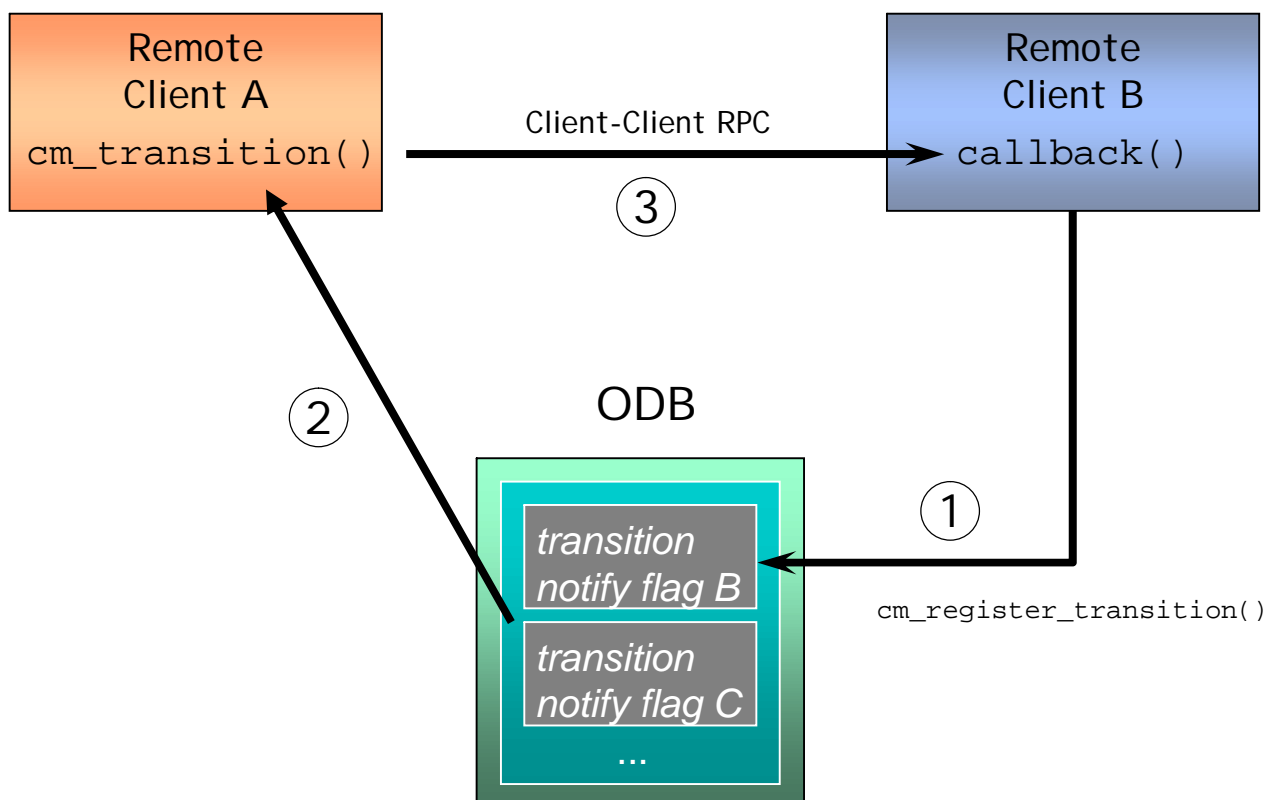
- Measurements are subdivided into "Runs"
- Typically one Run is logged into one data file
- Three run states: Stopped, Paused, Running
- Run Transitions: Start, Stop, Pause, Resume
- All clients need to be synchronized



e.g. run start:

- Logger opens data file
- Analyzer clears histograms
- Front-end enables trigger

# Transition Synchronization



# Transition example

```
#include <stdio.h>
#include "midas.h"
```

```
INT tr_start(INT run_number, char *error)
{
    printf("Start run %d\n", run_number);
    return CM_SUCCESS;
}
```

```
main()
{
    int status;
```

```
    cm_connect_experiment("", "", "Frontend", 0);
```

```
    /* register to be notified on START transition */
    cm_register_transition(TR_START, tr_start);
```

```
    /* enter event loop */
    while (cm_yield(1000) != RPC_SHUTDOWN);
```

```
    cm_disconnect_experiment();
}
```

```
INT cm_transition(
    INT transition,
    INT run_number,
    char *perror, INT strsize,
    INT async_flag,
    INT debug_flag);
```

2

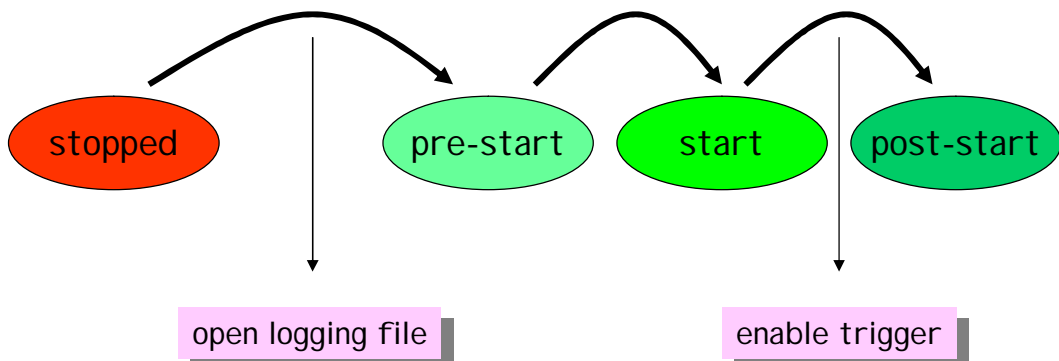
1



## Pre- Post- Transitions

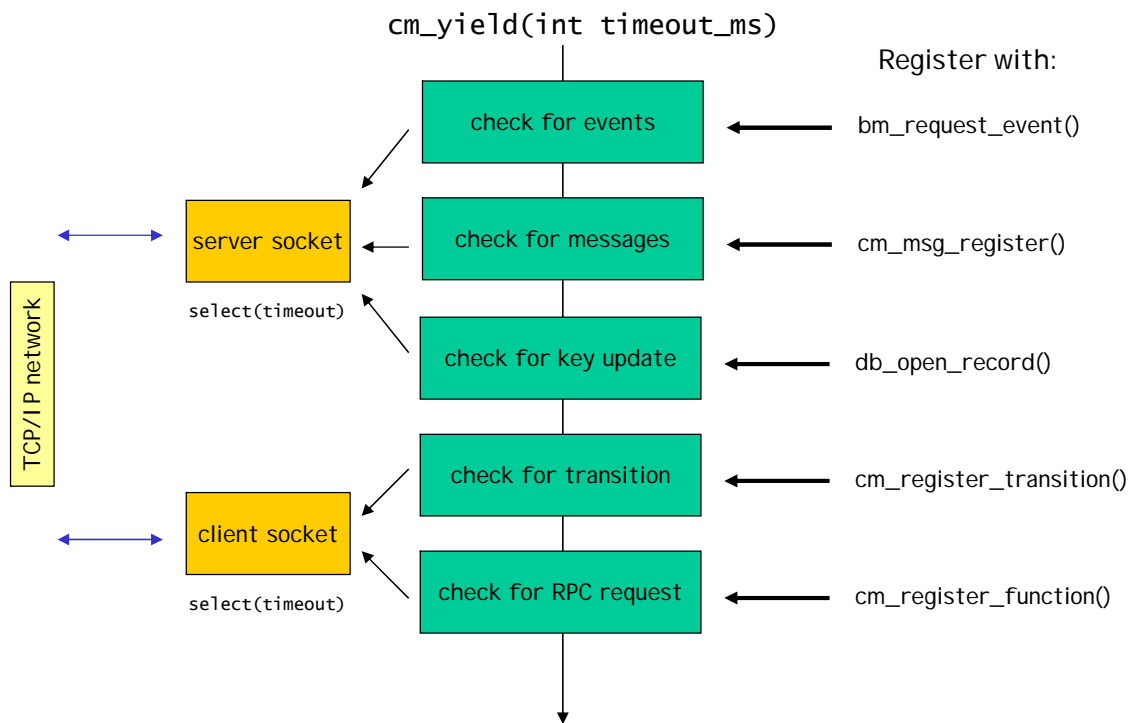
Certain order necessary:

e.g. first open logging file, then enable trigger



Only when all clients have successfully made one transition, continue with next

# MIDAS event loop

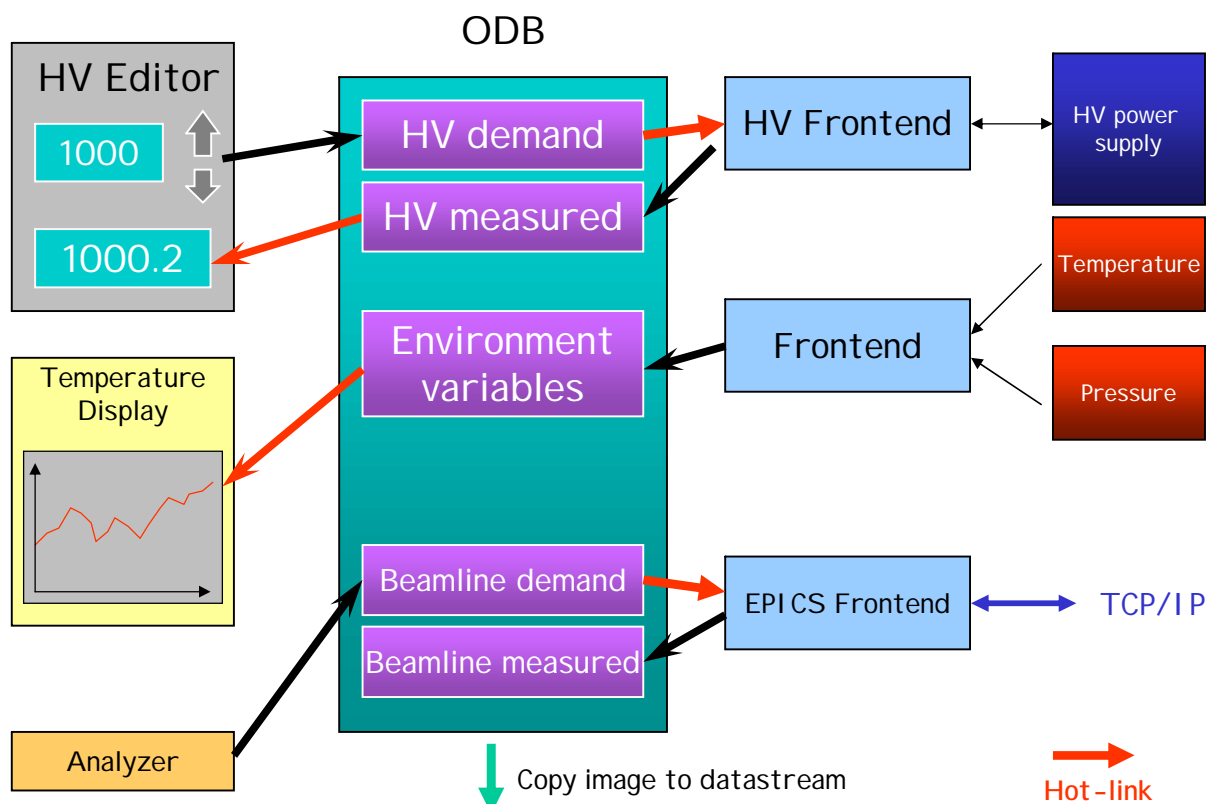


## Slow Control System

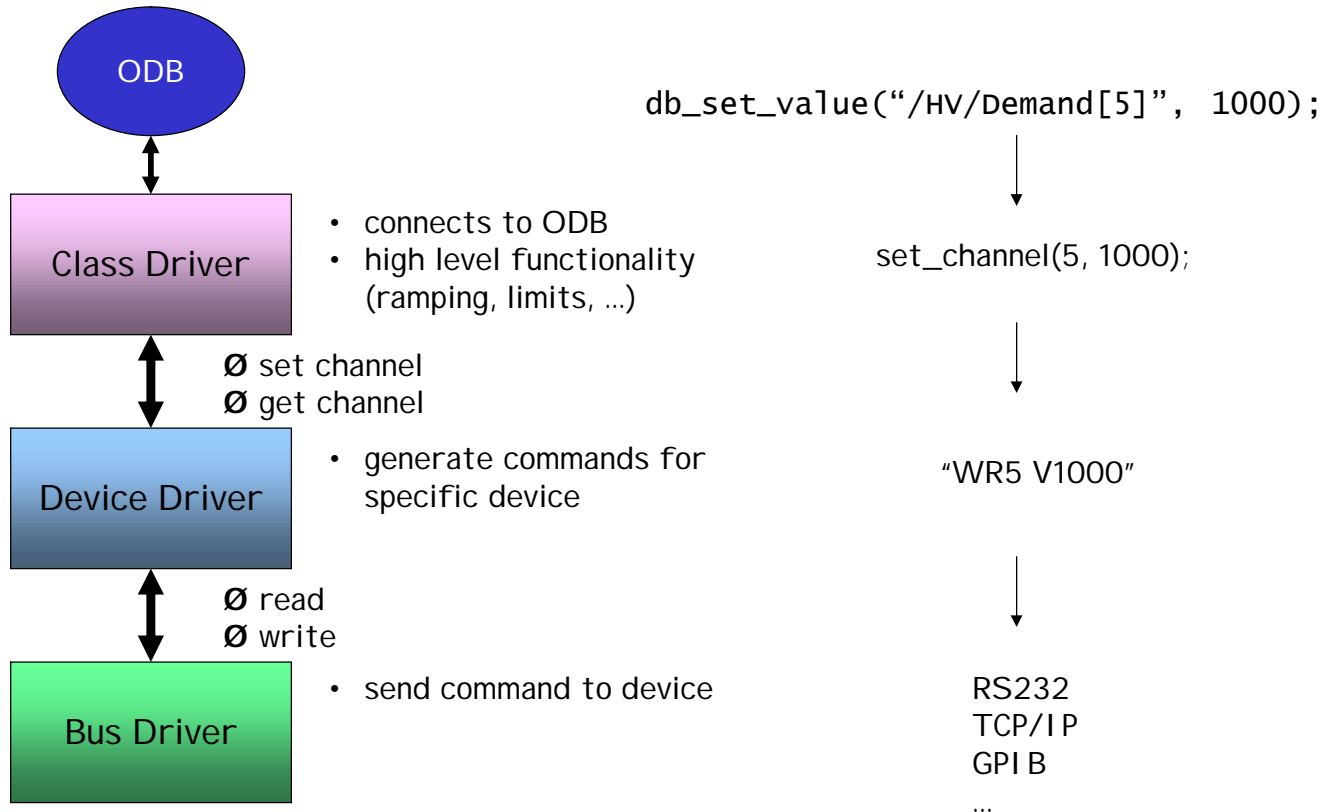
# Slow Control

- "Slow Control": Everything not related to event-based readout
  - High voltage
  - Environment (temperature, pressure)
  - Beam line magnets/slits
  - Gas systems, ...
- Requirements
  - Have all slow control data with event data on tape
  - Access all slow control equipment in a standardized way
  - Every program should have access to slow control equipment (analyzer, tools)
  - History system (on-line display, date-to-date query)

## Slow Control through ODB



# Three-layer driver concept



## Advantages

- Class driver does not have to know anything about specific device
- Device driver does not have to know how to physically connect to device
- Bus driver does not have to know anything about device
- Layers can be exchanged easily
- Device driver very simple
- New features in class driver are inherited by all devices automatically
- Class driver can encapsulate slow control data in events for taping

# Slow Control Drivers

## Available devices:

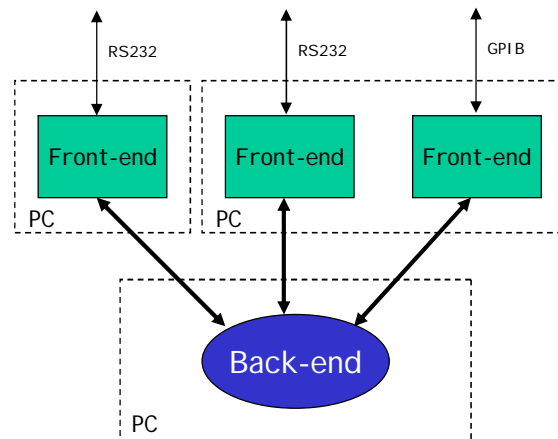
- LeCroy HV
- Keithley DAS cards
- Epics Channel Access

midas/drivers/class/  
midas/drivers/device/  
midas/drivers/bus/

## Problem:

Devices are read out sequentially, slow devices may block others

- > "slowdev" class driver
- > separate front-ends
- > faster devices



# Writing your own device driver

```

INT lrs1440_set(LRS1440_INFO *info, INT channel, float value)
{
    char str[80];

    sprintf(str, "%04.0f C%02d\r", value, channel);

    BD_PUTS(str);
    BD_GETS(str, sizeof(str), "\r\n\r\n", 1000);

    return FE_SUCCESS;
}

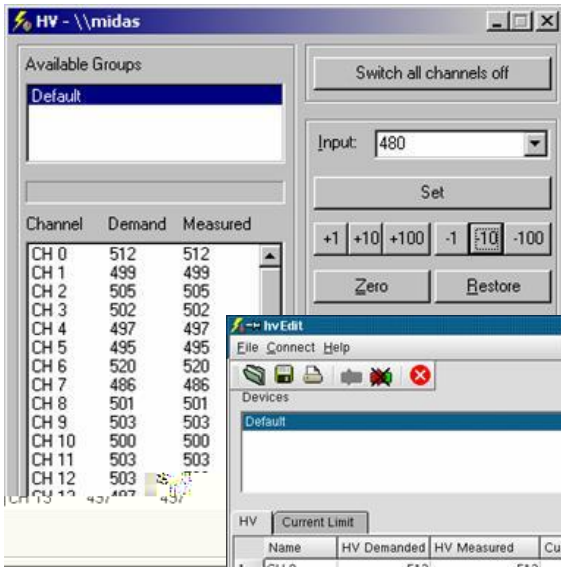
INT lrs1440_get(LRS1440_INFO *info, INT channel, float *pvalue)
{
    int value;
    char str[256];

    sprintf(str, "R V C%02d\r", channel);
    BD_PUTS(str);
    BD_GETS(str, sizeof(str), "Act ", 1000);
    BD_GETS(str, sizeof(str), "\r\n\r\n", 1000);
    sscanf(str+1, "%d", &value);

    *pvalue = (float) value;

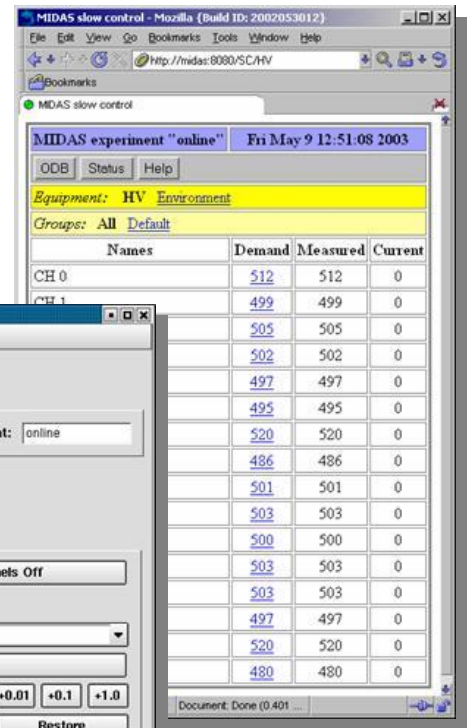
    return FE_SUCCESS;
}
  
```

- Use existing driver as template
- Implement device specific commands
- Put device specific settings in C structure
- Use BD\_XXX routines for communication



Windows/MFC

HVEdit



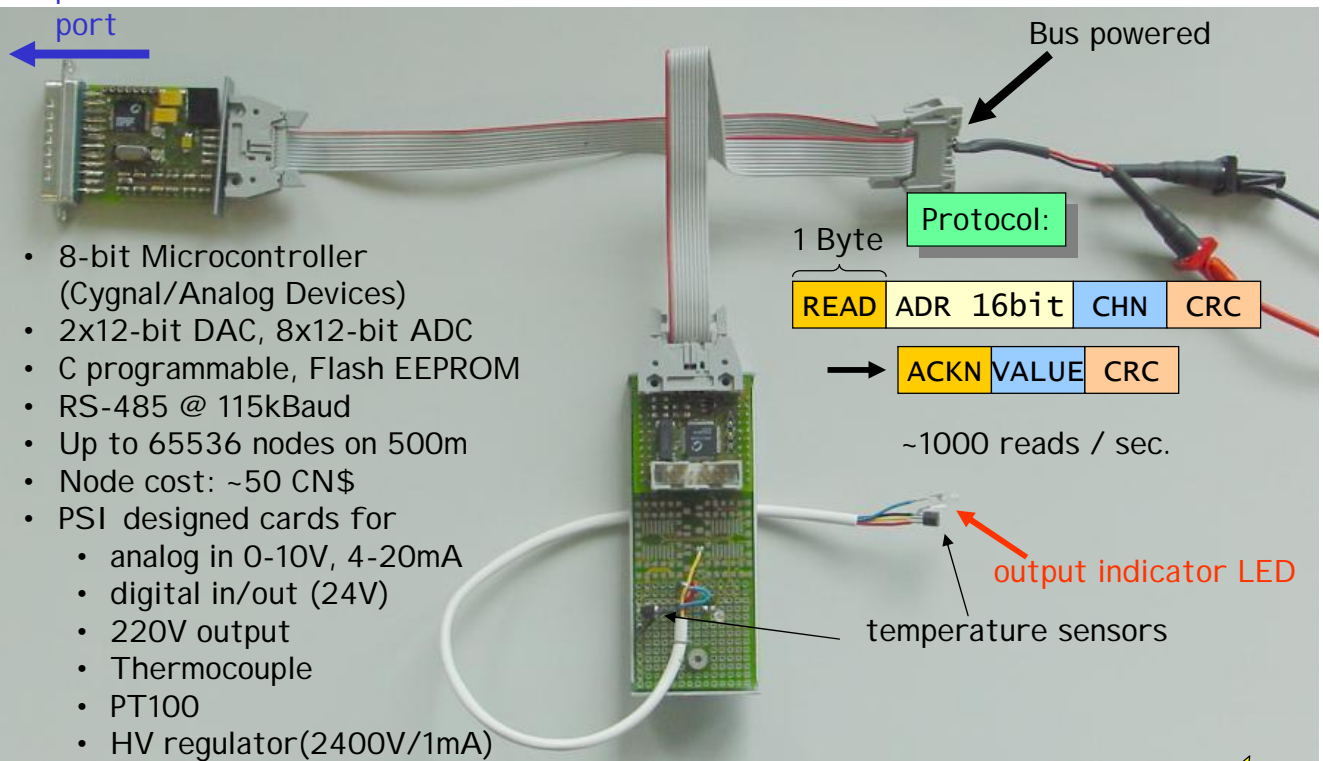
Web interface

Qt

Andreas Suter  
PSI  
C++ class library

## MSCB system

PC parallel port





# History System

## History system

- Requirements
  - Want to store slow control data in database over years
  - Online display of trend lines
  - Database queries: "Show temperature last May"
  - Fast write/search
- Standard databases like MySQL are "overkill"
  - no need for record locking, delete records, edit records
  - CPU overhead
- MIDAS history system
  - small set of C functions
  - events are linearly written to files, one file per day
  - files contain event definition
  - simple query function
  - data gets only written when changing (hot-link) and when event is written to data stream

# History File Structure

030516.hst	030517.hst	030518.hst
event 1 definition	event 1 definition	event 1 definition
event 1	event 2 definition	event 2 definition
event 1	event 1	event 1
event 1	event 2	event 2
event 2 definition	event 1	event 1
event 2	event 2	event 2
event 1	event 1	event 1
event 2	event 2	event 2
event 2	event 2	event 2
event 1	event 1	event 1
event 2	event 2	event 2
event 2 definition	event 1	event 1
event 2	event 2	event 2
event 1	event 1	event 1

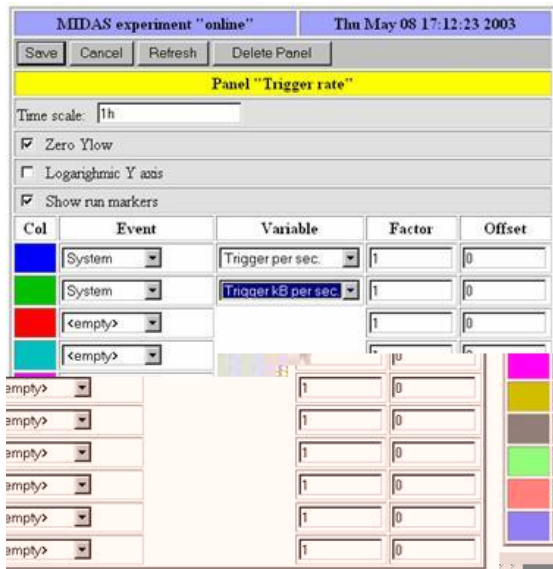
event definition can change during lifetime of experiment

# History Usage

- History written by mlogger
- Flag in front-end / slow control front-end
- Link in ODB
- Display settings (panels) in ODB

Key name	Type	#Val	Size	Last	Opn	Mode	Value
History	DIR						
Links	DIR						
System	DIR						
Trigger per sec.	LINK	1	46	1h	0	RWD	/Equipment/Trigger/Statistics/Events per sec.
Trigger kbytes per sec.	SLINK	1	46	1h	0	RWD	/Equipment/Trigger/Statistics/kbytes per sec.
Display	DIR						
Trigger rate	DIR						
Variables	STRING	2	64	1h	0	RWD	System:Trigger per sec. System:Trigger kB per sec.
Time Scale	STRING	1	32	1h	0	RWD	1h
Factor	FLOAT	2	4	1h	0	RWD	[0] 1 [1] 1
Offset	FLOAT	2	4	1h	0	RWD	[0] 0 [1] 0
Timescale	STRING	1	32	1h	0	RWD	1h
Zero ylow	BOOL	1	4	1h	0	RWD	y
Show run markers	BOOL	1	4	1h	0	RWD	y
Buttons	STRING	7	32	1h	0	RWD	[0] 10m [1] 1h [2] 3h [3] 12h [4] 24h [5] 3d [6] 7d
Log axis	BOOL	1	4	1h	0	RWD	n

# Display configuration

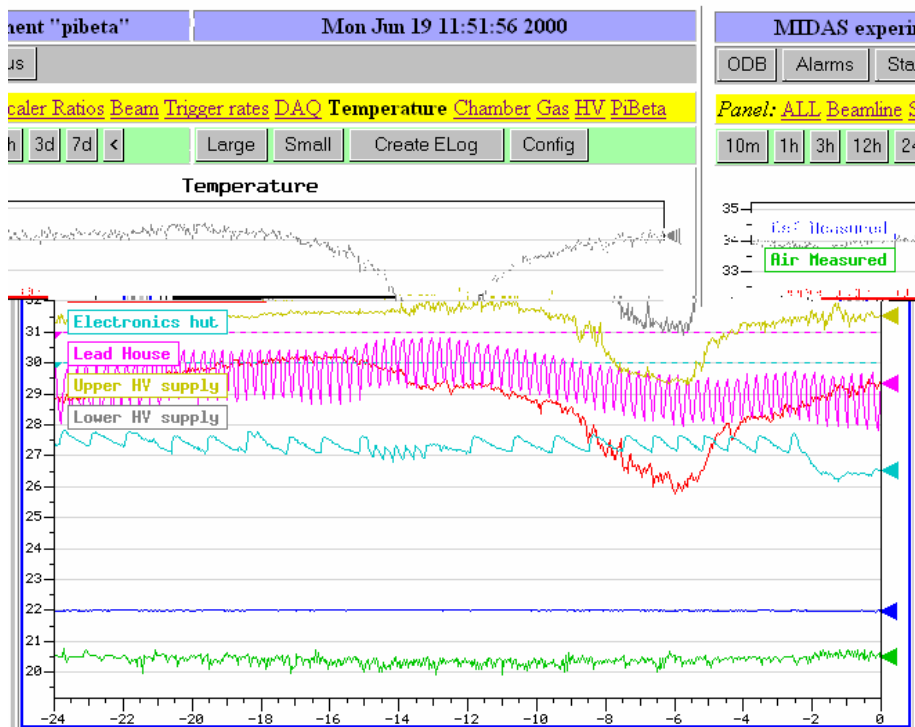


Web based configuration



Resulting display

## $\pi\beta$ Temperature Display



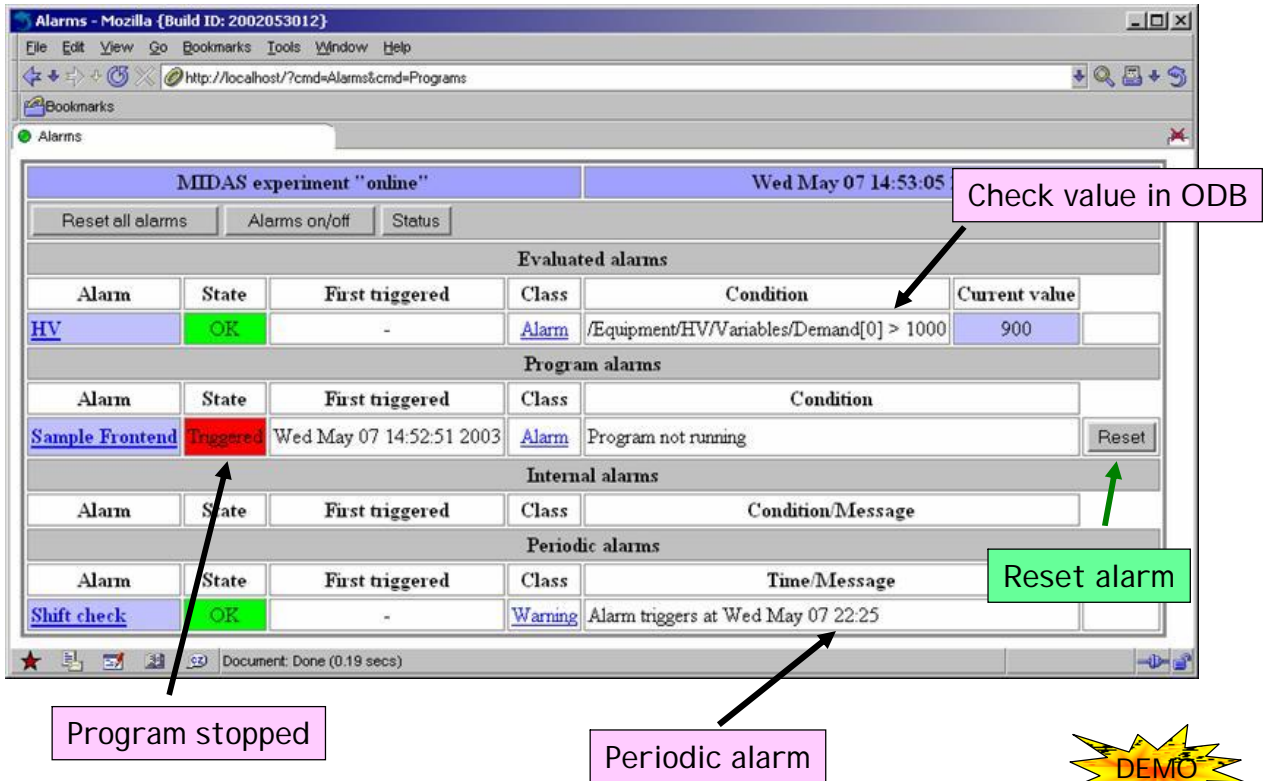
- Display through mhttpd
- Gif images generated dynamically in memory
- mchart utility for strip-chart display
- Query with mhist
- Write speed: 2000 events/s
- Query over one month in ~10sec

# Alarm System

## MI DAS Alarm System

- Alarms are an important issue in experiments
- Alarm System incorporated in MI DAS library
  - > every client does alarm checks
- Four types of alarm
  - ODB value <, >, =, !=
  - Program stopped
  - Periodic alarm (-> shift checks, change LN<sub>2</sub>)
  - Internal alarms: Triggered by user code  
al\_trigger\_alarm()
- Alarm triggers alarm "class" (Warning, Alarm, ...)
  - write system message (-> speaker)
  - write elog message
  - stop run
  - execute command (-> pager, SMS)

# Alarm examples



**Alarms - Mozilla {Build ID: 2002053012}**

File Edit View Go Bookmarks Tools Window Help

http://localhost/?cmd=Alarms&cmd=Programs

Alarms

MIDAS experiment "online" Wed May 07 14:53:05

Reset all alarms Alarms on/off Status

**Evaluated alarms**

Alarm	State	First triggered	Class	Condition	Current value
<a href="#">HV</a>	OK	-	<a href="#">Alarm</a>	/Equipment/HV/Variables/Demand[0] > 1000	900

**Program alarms**

Alarm	State	First triggered	Class	Condition	
<a href="#">Sample Frontend</a>	Triggered	Wed May 07 14:52:51 2003	<a href="#">Alarm</a>	Program not running	<a href="#">Reset</a>

**Internal alarms**

Alarm	State	First triggered	Class	Condition/Message

**Periodic alarms**

Alarm	State	First triggered	Class	Time/Message
<a href="#">Shift check</a>	OK	-	<a href="#">Warning</a>	Alarm triggers at Wed May 07 22:25

Document: Done (0.19 secs)

**Annotations:**

- Check value in ODB (points to Current value: 900)
- Program stopped (points to Sample Frontend state: Triggered)
- Periodic alarm (points to Shift check alarm)
- Reset alarm (points to Reset button)

**DEMO**

## Electronic Logbook

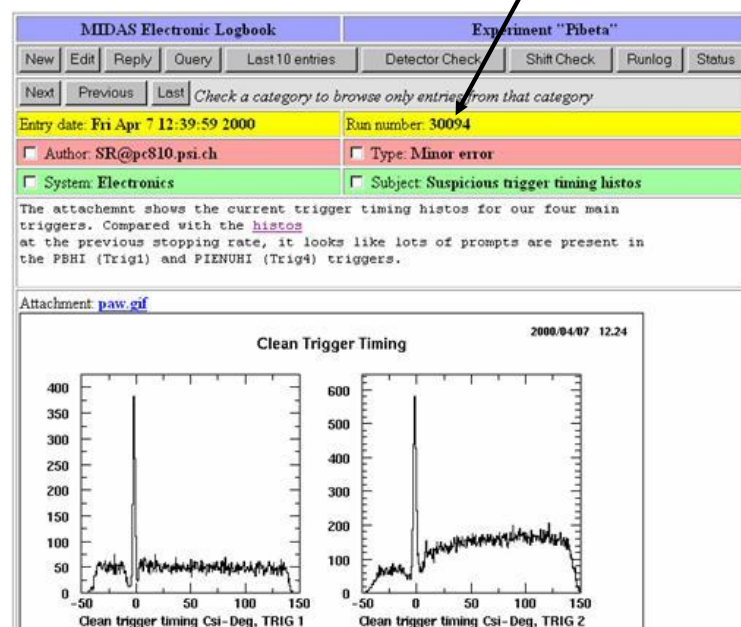
# ELOG

- Why give up paper logbook?
  - Remote access (remote shifts, expert off-site)
  - Easy copy
  - Searchable
  - Automatic logbook entries (analyzer)
- How
  - Web interface built into mhttpd (no CGI!)
  - Simple flat file database (one file per day)
  - Image attachments (scanner, digital camera)
  - Standalone ELOG package (<http://midas.psi.ch/eelog>)
- Experience
  - Excellent tool once experiment is in stable operation
  - For setting up experiments, still some paper needed

## ELOG example

run number automatically filled in

PAW screendump →





# ELOG forms

MIDAS Electronic Logbook		Form "Shift Check"	
Submit	Reset Form		
Entry date: Thu May 08 11:59:34 2003		Run number: 22407	
Author: SR			
Item	Checked	Comment	
1 Beamline Magnets	<input checked="" type="checkbox"/>	Magnet reset	
2 Temperature	<input checked="" type="checkbox"/>		
3 Humidity	<input checked="" type="checkbox"/>		
4 Disk,Tape,Archive	<input checked="" type="checkbox"/>	Tape has to be changed soon	
5 Zero Fanouts	<input type="checkbox"/>		
6 Gas Bubbler	<input checked="" type="checkbox"/>		
7 Crate Fans,Iceberg	<input checked="" type="checkbox"/>		
8 Empty Water	<input checked="" type="checkbox"/>		

- Form to be filled out every shift
- Form items defined in ODB

Resulting ELOG entry

MIDAS Electronic Logbook		Experiment "pibeta"	
New	Edit	Reply	Query
Detector Check		Shift Check	
Runlog		Status	
Last 8h Last 24h Last 7d			
Next Previous Last Check a category to browse only entries from that category			
Entry date: Thu May 08 12:01:29 2003		Run number: 22407	
<input type="checkbox"/> Author: SR@pc810		<input type="checkbox"/> Type: Shift Check	
<input type="checkbox"/> System: General		<input type="checkbox"/> Subject: Shift Check	
1 Beamline Magnets : [X] Magnet reset 2 Temperature : [X] 3 Humidity : [X] 4 Disk,Tape,Archive : [X] Tape has to be changed soon 5 Zero Fanouts : [ ] 6 Gas Bubbler : [X] 7 Crate Fans,Iceberg : [X] 8 Empty Water : [X]			

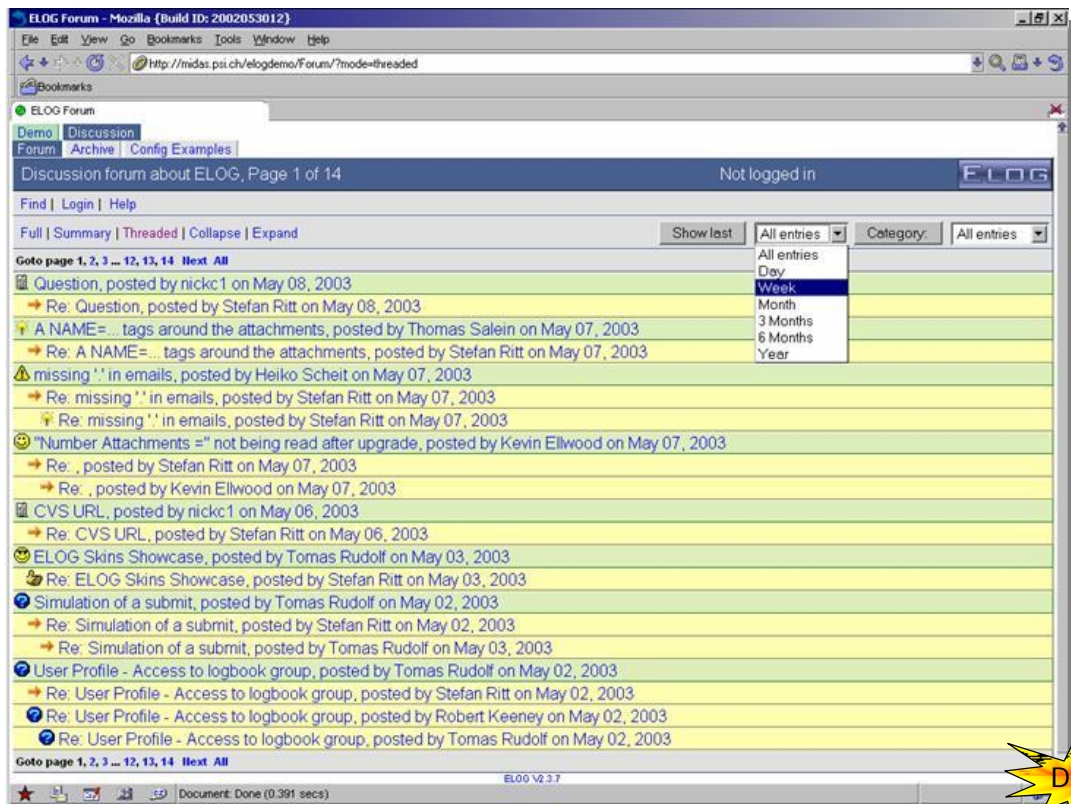
# Automatic ELOG entries

- PMT ADC spectra have peak in spectrum
- Spectra are cross-correlated with stored spectra, if enough statistics (~6h)
- Gain deviation is calculated
- HV is adjusted automatically (analyzer writes to ODB)
- ELOG entry is produced

MIDAS Electronic Logbook		Experiment "pibeta"	
New	Edit	Reply	Query
Detector Check		Shift Check	
Runlog		Status	
Last 8h Last 24h Last 7d			
Next Previous Last Check a category to browse only entries from that category			
Entry date: Wed Dec 22 15:17:09 1999		Run number: 22403	
<input type="checkbox"/> Author: Analyzer@pc810		<input type="checkbox"/> Type: Gainmatch	
<input type="checkbox"/> System: Analyzer		<input type="checkbox"/> Subject: Gainmatch mode 'e'	
Result of automatic gain match			
Attachment <a href="#">gainmatch.txt</a>			
Wed Dec 22 15:16:56 1999			
Chn 0: N= 501 Gain=0.984 Corr=0.966 +- 0.001, HV 1829V +6V * Chn 1: N= 687 Gain=0.930 Corr=1.023 +- 0.001, HV 1797V -4V * Chn 2: N= 617 Gain=0.975 Corr=1.027 +- 0.002, HV 1885V -5V * Chn 3: N= 784 Gain=0.950 Corr=1.041 +- 0.001, HV 1893V -8V * Chn 4: N= 491 Gain=0.948 Corr=0.966 +- 0.001, HV 1903V +7V * Chn 5: N= 641 Gain=0.954 Corr=1.014 +- 0.001, HV 1602V -2V * Chn 6: N= 648 Gain=0.987 Corr=1.011 +- 0.001, HV 1866V -2V * Chn 7: N= 521 Gain=1.003 Corr=0.995 +- 0.001, HV 1843V +1V * Chn 8: N= 696 Gain=0.966 Corr=1.022 +- 0.001, HV 1421V -3V * Chn 9: N= 507 Gain=0.955 Corr=0.987 +- 0.001, HV 1784V +2V * Chn 10: N= 1034 Gain=0.916 Corr=1.014 +- 0.001, HV 1576V -2V * Chn 11: N= 640 Gain=0.934 Corr=0.950 +- 0.001, HV 1946V +10V * Chn 12: N= 1151 Gain=0.920 Corr=1.029 +- 0.001, HV 1653V -5V * Chn 13: N= 993 Gain=1.016 Corr=1.026 +- 0.001, HV 1852V -5V * Chn 14: N= 785 Gain=1.000 Corr=0.980 +- 0.001, HV 1835V +4V * Chn 15: N= 994 Gain=0.930 Corr=1.016 +- 0.001, HV 1605V -3V * Chn 16: N= 1049 Gain=0.989 Corr=1.026 +- 0.001, HV 1704V -4V * Chn 17: N= 911 Gain=0.973 Corr=1.006 +- 0.001, HV 1538V -1V * Chn 18: N= 901 Gain=1.178 Corr=1.015 +- 0.001, HV 2066V -3V * Chn 19: N= 967 Gain=0.899 Corr=1.010 +- 0.001, HV 1557V -2V * Chn 20: N= 841 Gain=0.941 Corr=1.006 +- 0.001, HV 1588V -1V * Chn 21: N= 863 Gain=0.910 Corr=0.995 +- 0.001, HV 1965V +1V * Chn 22: N= 887 Gain=0.888 Corr=1.007 +- 0.001, HV 1633V -1V * Chn 23: N= 1101 Gain=0.953 Corr=1.046 +- 0.001, HV 1807V -8V * Chn 24: N= 998 Gain=0.957 Corr=1.006 +- 0.001, HV 1744V -1V * Chn 25: N= 1090 Gain=1.146 Corr=1.023 +- 0.001, HV 1773V -4V * Chn 26: N= 838 Gain=0.943 Corr=1.000 +- 0.001, HV 1856V +0V * Chn 27: N= 980 Gain=0.935 Corr=1.025 +- 0.001, HV 1649V -4V * Chn 28: N= 884 Gain=0.902 Corr=1.019 +- 0.001, HV 1722V -3V * Chn 29: N= 905 Gain=0.877 Corr=1.006 +- 0.001, HV 1474V -1V * Chn 30: N= 911 Gain=0.836 Corr=0.990 +- 0.001, HV 1656V +2V * Chn 31: N= 964 Gain=0.853 Corr=1.025 +- 0.001, HV 2127V -5V *			

Modified HV's

# Standalone ELOG



## Frontend Framework



```
#include <stdio.h>
#include "midas.h"
#include "msystem.h"

INT running; /* stopped:0, started:1 */
INT serial = 0; /* event serial number */
INT run_number = 0; /* run number */

HANDLE hBuf; /* buffer handle for events */

INT tr_start(INT rn, char *error)
{
    printf("Start run %d\n", rn);
    serial = 0;
    run_number = rn;
    running = TRUE;
    return CM_SUCCESS;
}

INT tr_stop(INT rn, char *error)
{
    running = FALSE;
    rpc_flush_event();
    bm_flush_cache(hBuf, SYNC);
    printf("Stop run %d\n", rn);
    return CM_SUCCESS;
}

main()
{
    INT status, size=1000, id=0;
    DWORD last_time=0;
    char event[2000];
    char host_name[256];
    char dash[] = "-\\|/";

    printf("Host to connect: ");
    ss_gets(host_name, 256);

    /* now connect to server */
    status = cm_connect_experiment(host_name, "", "MinIFE", NULL);
    if (status != CM_SUCCESS)
        return 1;

    cm_register_transition(TR_START, tr_start);
    cm_register_transition(TR_STOP, tr_stop);
}
```

```
/* open event bufer and set cache size */
bm_open_buffer(EVENT_BUFFER_NAME, EVENT_BUFFER_SIZE, &hBuf);
bm_set_cache_size(hBuf, 0, 100000);

do
{
    if (!running)
    {
        /* if run is not started, call yield with a timeout of 1s */
        status = cm_yield(1000);
        if (status == RPC_SHUTDOWN)
            break;
    }
    else
    {
        /* during run call yield with 0 timeout once each 300 ms */
        if (ss_millitime() - last_time > 300)
        {
            printf("%c\r", dash[id++ % 4]);
            fflush(stdout);

            status = cm_yield(0);
            if (status == RPC_SHUTDOWN)
                break;
            last_time = ss_millitime();
        }
    }

    /* if run is started, send event */
    if (running)
    {
        /* compose event header, empty event */
        bm_compose_event((EVENT_HEADER *) event, 1, 1, size, serial++);

        /* send event */
        rpc_send_event(hBuf, event, size+sizeof(EVENT_HEADER), SYNC);
    }

} while (TRUE);

cm_disconnect_experiment();

return 1;
}
```



## Front-end Framework

- Update event statistics (events/s, kb/s)
- Support for interrupt/pollled/periodic/slow control events
- Open buffer, send events
- Send event samples to ODB for inspection
- Stop run after number of events received
- Incorporate slow control system
- CNAF functionality

Framework

Trigger event declaration

Scaler event declaration

BOR

EOR

Check for trigger event

Trigger event readout

Scaler readout

# Front-end user code

```
#include <stdio.h>
#include <stdlib.h>
#include "midas.h"
#include "mcstd.h"
```

```
/*-- Equipment list --*/
```

```
EQUIPMENT equipment[] = {
```

```
  { "Trigger",          /* equipment name */
    { 1, 0,             /* event ID, trigger mask */
      "SYSTEM",         /* event buffer */
      EQ_POLLED,        /* equipment type */
      LAM_SOURCE(0,0xFFFFF), /* event source crate 0, all stations */
      "MIDAS",          /* format */
      TRUE,             /* enabled */
      RO_RUNNING |      /* read only when running */
      RO_ODB,           /* and update ODB */
      500,              /* poll for 500ms */
      0,                /* stop run after this event limit */
      0,                /* number of sub events */
      0,                /* don't log history */
      "", "", "", },
    read_trigger_event, /* readout routine */
    NULL, NULL, NULL,
  },
```

```
  { "Scaler",          /* equipment name */
    { 2, 0,             /* event ID, trigger mask */
      "SYSTEM",         /* event buffer */
      EQ_PERIODIC |     /* equipment type */
      EQ_MANUAL_TRIG,   /* event source */
      0,                /* format */
      "MIDAS",          /* enabled */
      TRUE,             /* enabled */
      RO_RUNNING |      /* read when running and on transitions */
      RO_TRANSITIONS | /* and update ODB */
      RO_ODB,           /* read every 10 sec */
      10000,            /* stop run after this event limit */
      0,                /* number of sub events */
      1,                /* log history */
      "", "", "", },
    read_scaler_event,  /* readout routine */
    NULL, NULL, NULL,
  },
  { "" }
};
```

```
INT frontend_init() {
  return SUCCESS;
}
```

```
INT frontend_exit() {
  return SUCCESS;
}
```

```
INT begin_of_run(INT run_number, char *error) {
  return SUCCESS;
}
```

```
INT end_of_run(INT run_number, char *error) {
  return SUCCESS;
}
```

```
INT frontend_loop() {
  return SUCCESS;
}
```

```
/*-----*/
```

```
INT poll_event(INT source, INT count, BOOL test)
{
  int i;
  DWORD lam;

  for (i=0 ; i<count ; i++)
  {
    cam_lam_read(LAM_SOURCE_CRATE(source), &lam);

    if (lam & LAM_SOURCE_STATION(source))
      if (!test)
        return lam;
  }

  return 0;
}
```

```
INT read_trigger_event(char *pevent, INT off) {
  return 0;
}
```

```
INT read_scaler_event(char *pevent, INT off) {
  return 0;
}
```

# Creating banks

```
INT read_trigger_event(char *pevent, INT offset)
{
  ADC0_BANK *adc0_bank;
  WORD      *pdata, a;

  bk_init(pevent);

  /* create structured ADC0 bank */
  bk_create(pevent, "ADC0", TID_STRUCT, &adc0_bank);

  pdata = (WORD *)adc0_bank;

  /* read CAMAC adcs */
  cami(CRATE, SLOT_ADC, 0, 0, &adc0_bank->adc0);
  cami(CRATE, SLOT_ADC, 1, 0, &adc0_bank->adc1);
  cami(CRATE, SLOT_ADC, 2, 0, &adc0_bank->adc2);
  cami(CRATE, SLOT_ADC, 3, 0, &adc0_bank->adc3);

  /* clear ADC */
  camc(CRATE, SLOT_ADC, 0, 9);

  bk_close(pevent, pdata);

  /* create variable length TDC bank */
  bk_create(pevent, "TDC0", TID_WORD, &pdata);

  /* read CAMAC TDC until no Q */
  caml6i_rq(CRATE, SLOT_TDC, 0, 0, &pdata, 96);

  /* clear TDC */
  camc(CRATE, SLOT_TDC, 0, 9);

  bk_close(pevent, pdata);

  return bk_size(pevent);
}
```

```
typedef struct {
  WORD  adc0;
  WORD  adc1;
  WORD  adc2;
  WORD  adc3;
} ADC0_BANK;
```

structured bank

bank with  
variable length

# Frontend control through ODB

```
[local:online:S]/>cd Equipment/Trigger/
[local:online:S]Trigger>ls -lr
```

Key name	Type	#Val	Size	Last	Opn	Mode	Value
Trigger	DIR						
Variables	DIR						
ADC0	DIR						
adc0	WORD	1	2	1h	0	RWD	226
adc1	WORD	1	2	1h	0	RWD	29
adc2	WORD	1	2	1h	0	RWD	489
adc3	WORD	1	2	1h	0	RWD	574
TDC0	WORD	1	2	1h	0	RWD	0
		[0]					49
		[1]					152
		[2]					156
		[3]					223
ASUM	DIR						
Sum	FLOAT	1	4	3h	0	RWD	1318
Average	FLOAT	1	4	3h	0	RWD	329.5
Settings	DIR						
CAMAC slot	BYTE	1	1	3h	0	RWD	7
Common	DIR						
Event ID	WORD	1	2	1h	0	RWD	1
Trigger mask	WORD	1	2	1h	0	RWD	0
Buffer	STRING	1	32	1h	0	RWD	SYSTEM
Type	INT	1	4	1h	0	RWD	2
Source	INT	1	4	1h	0	RWD	16777215
Format	STRING	1	8	1h	0	RWD	MIDAS
Enabled	BOOL	1	4	1h	0	RWD	y
Read on	INT	1	4	1h	0	RWD	257
Period	INT	1	4	1h	0	RWD	500
Event limit	DOUBLE	1	8	1h	0	RWD	1e+006
Num subevents	DWORD	1	4	1h	0	RWD	0
Log history	INT	1	4	1h	0	RWD	0
Frontend host	STRING	1	32	1h	0	RWD	pc810
Frontend name	STRING	1	32	1h	0	RWD	Sample Frontend
Frontend file name	STRING	1	256	1h	0	RWD	c:\midas\examples\experiment\frontend.c
Statistics	DIR						
Events sent	DOUBLE	1	8	0s	0	RWD	154234
Events per sec.	DOUBLE	1	8	0s	0	RWD	132
kBytes per sec.	DOUBLE	1	8	0s	0	RWD	12.233

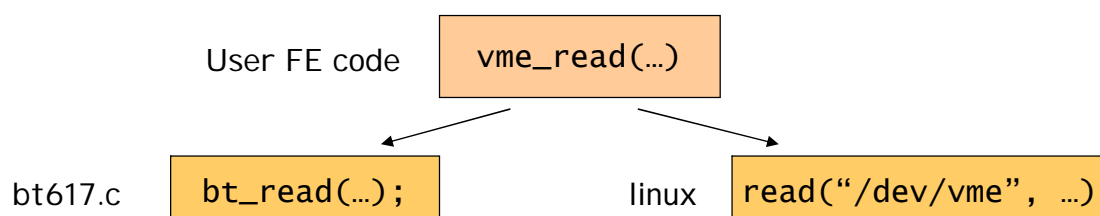
structured bank

variable length bank

Equipment Structure

## Hardware Access

- Standardized hardware access layer makes front-end independent of interfaces
- mcstd.h & esone.h
  - cam8i(const int c, const int n, const int a, const int f, BYTE \*d);
  - cam16i\_r(const int c, const int n, const int a, const int f, WORD \*\*d, const int r);
  - cam16i\_sn(const int c, const int n, const int a, const int f, WORD \*\*d, const int r);
- mvmestd.h
  - vme\_open, vme\_ioctl, vme\_close, vme\_read, vme\_write, vme\_mmap, vme\_unmap
- mfbstd.h
  - fb\_frd, fb\_frc, fb\_fwd, fb\_fwc, ...



# Equipment Types

- Periodic
  - scalers, ...
- Interrupt
  - trigger events
  - readout in interrupt routine
  - dual buffer event transfer to front-end framework
- Polled
  - trigger events
  - user-supplied polling routine
  - readout routine can be debugged
  - faster response time than interrupt on dedicated front-ends
- Slowcontrol
  - HV, environment, beam line, ...
  - Read by slow control system driver architecture

# Equipment Definition

## Polled, Periodic:

```
EQUIPMENT equipment[] = {
{ "Trigger",          /* equipment name */
  { 1, 0,             /* event ID, trigger mask */
    "SYSTEM",         /* event buffer */
    EQ_POLLED,         /* equipment type */
    LAM_SOURCE(0,0xFFFFF), /* event source crate 0, all stations */
    "MIDAS",          /* format */
    TRUE,              /* enabled */
    RO_RUNNING |      /* read only when running */
    RO_ODB,            /* and update ODB */
    500,               /* poll for 500ms */
    0,                 /* stop run after this event limit */
    0,                 /* number of sub events */
    0,                 /* don't log history */
    "", "", "", },
    read_trigger_event, /* readout routine */
    NULL, NULL, NULL,
  },
{ "Scaler",           /* equipment name */
  { 2, 0,             /* event ID, trigger mask */
    "SYSTEM",         /* event buffer */
    EQ_PERIODIC,       /* equipment type */
    0,                 /* event source */
    "MIDAS",          /* format */
    TRUE,              /* enabled */
    RO_RUNNING |      /* read when running and on transitions */
    RO_TRANSITIONS,    /* and update ODB */
    10000,             /* read every 10 sec */
    0,                 /* stop run after this event limit */
    0,                 /* number of sub events */
    0,                 /* log history */
    "", "", "", },
    read_scaler_event, /* readout routine */
    NULL, NULL, NULL,
  },
};
```

## Slowcontrol:

```
#include "class/hv.h"
#include "device/lrs1440.h"
#include "device/lrs4032.h"
#include "bus/rs232.h"

/* device driver list */
DEVICE_DRIVER hv_driver[] = {
  { "LRS1440", lrs1440, 8, rs232 },
  { "LRS4032", lrs4032, 4, rs232 },
  { "" }
};

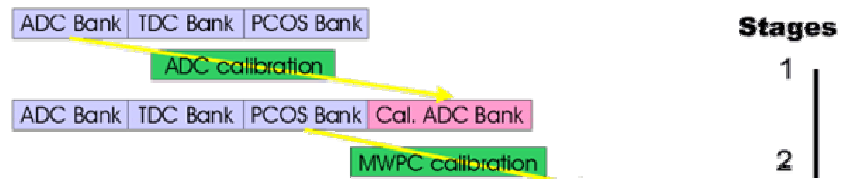
EQUIPMENT equipment[] = {
{ "HV",
  { 3, 0,             /* event ID, trigger mask */
    "SYSTEM",         /* event buffer */
    EQ_SLOW,          /* equipment type */
    0,                 /* event source */
    "FIXED",          /* format */
    TRUE,              /* enabled */
    RO_RUNNING |      /* read when running */
    RO_TRANSITIONS,    /* and on transitions */
    60000,             /* read every 60 sec */
    0,                 /* event limit */
    0,                 /* number of sub events */
    1,                 /* log history every event */
    "", "", "", },
    cd_hv_read,         /* readout routine */
    cd_hv,              /* class driver main routine */
    hv_driver,          /* device driver list */
    NULL,              /* init string */
  },
{ "" }
};
```

# Analyzer Framework

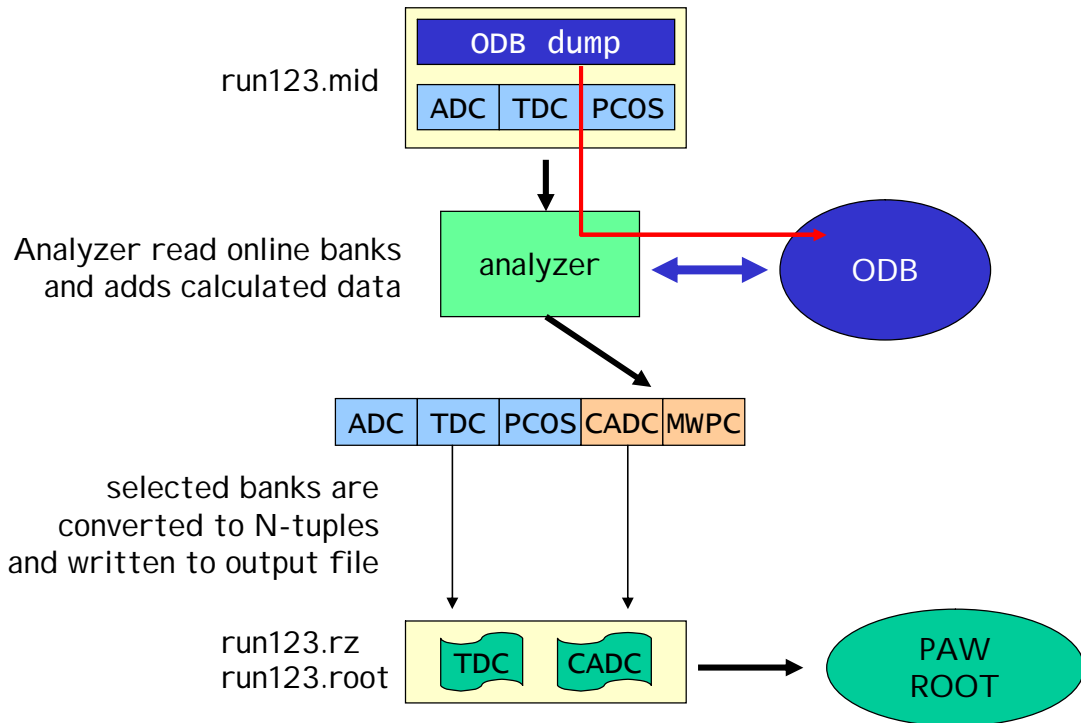
## MI DAS analyzer framework (mana.c)

- Automatically receive events
  - Online (buffer manager), request only certain events
  - Offline (read from \*.mid file)
- Write statistics to ODB
- Generate histos
  - Online
    - HBOOK histos and online N-tuples (shared memory)
    - ROOT histos (socket server)
  - Offline
    - Write \*.rz file with RWNT or CWNT and histos
    - Write \*.root file with TTree and histos
- Modular concept
- Get parameters (e.g. calibration constants) from ODB

# Modular concept

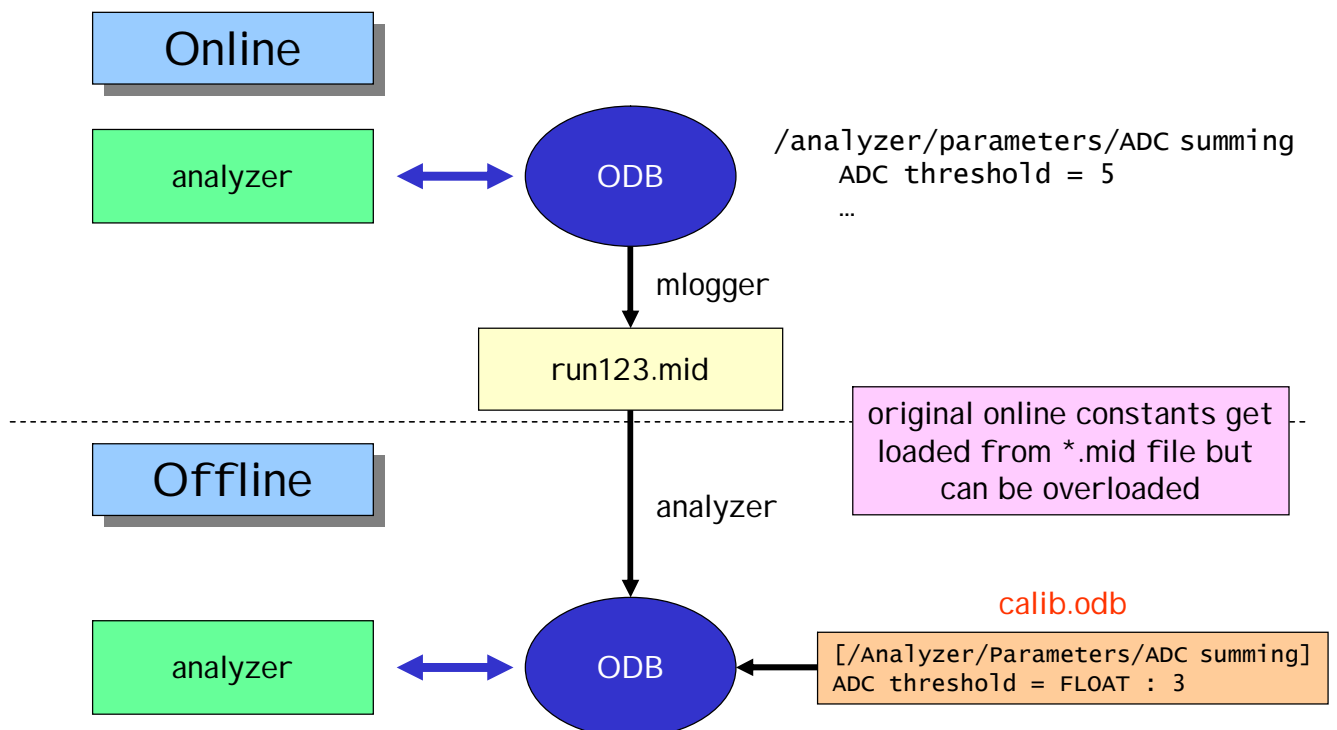


# Offline analysis



```
analyzer -i run123.mid -o run123.rz
```

# Calibration constants



```
analyzer -i run123.mid -o run123.rz -c calib.odb
```

```
;} TRIGGER_SETTINGS;
```

[illegible]

Mode Value-----

```
threshold;float pedestal;} ADC_SUMMING_PARAM;
```

~~Word~~ value ADG slot ----- WORD --- WORD-21h-2 --- RWD  
 tWORD

Midas Short Course - Real Time 2003 - Montreal - Part I

95/100

/Equipment/Trigger/Variables/Key name	Type	#Val	Size	Last
---------------------------------------	------	------	------	------

" is automatically created

/Equipment/Trigger/Variables/Key name	Type	#Val	Size	Last
---------------------------------------	------	------	------	------

" is automatically created see also:  
[experim.h](#)

[illegible][illegible]

WORD

enter "make" in ODBEdit.

```
adcthreshold:float pedestal;} ADC_SUMMING_PARAM;
```

**Key name**

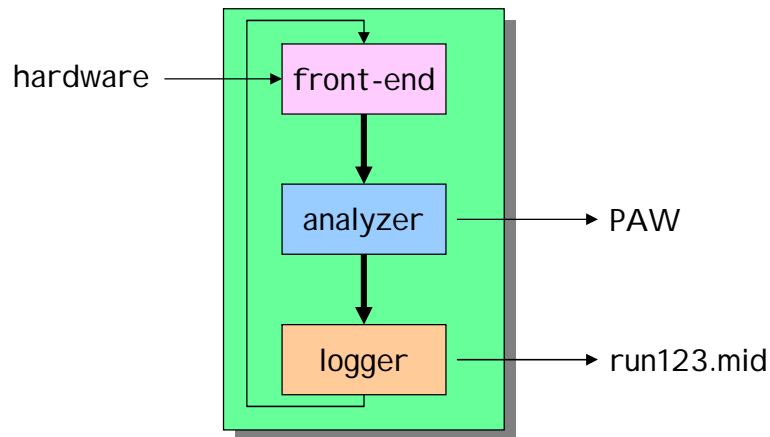
Type	#Val	Size	Last
------	------	------	------

Key name	Type	# Var	Size	Last
"K601" targetName	String	6	24	0
"K602" targetName	String	6	24	1
"K603" targetName	String	6	24	2
"K604" targetName	String	6	24	3
"K605" targetName	String	6	24	4
"K606" targetName	String	6	24	5
"K607" targetName	String	6	24	6
"K608" targetName	String	6	24	7
"K609" targetName	String	6	24	8
"K610" targetName	String	6	24	9
"K611" targetName	String	6	24	10
"K612" targetName	String	6	24	11
"K613" targetName	String	6	24	12
"K614" targetName	String	6	24	13
"K615" targetName	String	6	24	14
"K616" targetName	String	6	24	15
"K617" targetName	String	6	24	16
"K618" targetName	String	6	24	17
"K619" targetName	String	6	24	18
"K620" targetName	String	6	24	19
"K621" targetName	String	6	24	20
"K622" targetName	String	6	24	21
"K623" targetName	String	6	24	22
"K624" targetName	String	6	24	23
"K625" targetName	String	6	24	24
"K626" targetName	String	6	24	25
"K627" targetName	String	6	24	26
"K628" targetName	String	6	24	27
"K629" targetName	String	6	24	28
"K630" targetName	String	6	24	29
"K631" targetName	String	6	24	30
"K632" targetName	String	6	24	31
"K633" targetName	String	6	24	32
"K634" targetName	String	6	24	33
"K635" targetName	String	6	24	34
"K636" targetName	String	6	24	35
"K637" targetName	String	6	24	36
"K638" targetName	String	6	24	37
"K639" targetName	String	6	24	38
"K640" targetName	String	6	24	39
"K641" targetName	String	6	24	40
"K642" targetName	String	6	24	41
"K643" targetName	String	6	24	42
"K644" targetName	String	6	24	43
"K645" targetName	String	6	24	44
"K646" targetName	String	6	24	45
"K647" targetName	String	6	24	46
"K648" targetName	String	6	24	47
"K649" targetName	String	6	24	48
"K650" targetName	String	6	24	49
"K651" targetName	String	6	24	50
"K652" targetName	String	6	24	51
"K653" targetName	String	6	24	52
"K654" targetName	String	6	24	53
"K655" targetName	String	6	24	54
"K656" targetName	String	6	24	55
"K657" targetName	String	6	24	56
"K658" targetName	String	6	24	57
"K659" targetName	String	6	24	58
"K660" targetName	String	6	24	59
"K661" targetName	String	6	24	60
"K662" targetName	String	6	24	61
"K663" targetName	String	6	24	62
"K664" targetName	String	6	24	63
"K665" targetName	String	6	24	64
"K666" targetName	String	6	24	65
"K667" targetName	String	6	24	66
"K668" targetName	String	6	24	67
"K669" targetName	String	6	24	68
"K670" targetName	String	6	24	69
"K671" targetName	String	6	24	70
"K672" targetName	String	6	24	71
"K673" targetName	String	6	24	72
"K674" targetName	String	6	24	73
"K675" targetName	String	6	24	74
"K676" targetName	String	6	24	75
"K677" targetName	String	6	24	76
"K678" targetName	String	6	24	77
"K679" targetName	String	6	24	78
"K680" targetName	String	6	24	79
"K681" targetName	String	6	24	80
"K682" targetName	String	6	24	81
"K683" targetName	String	6	24	82
"K684" targetName	String	6	24	83
"K685" targetName	String	6	24	84
"K686" targetName	String	6	24	85
"K687" targetName	String	6	24	86
"K688" targetName	String	6	24	87
"K689" targetName	String	6	24	88
"K690" targetName	String	6	24	89
"K691" targetName	String	6	24	90
"K692" targetName	String	6	24	91
"K6				



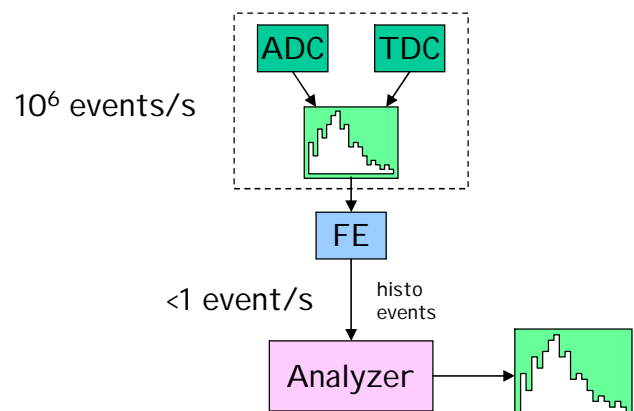
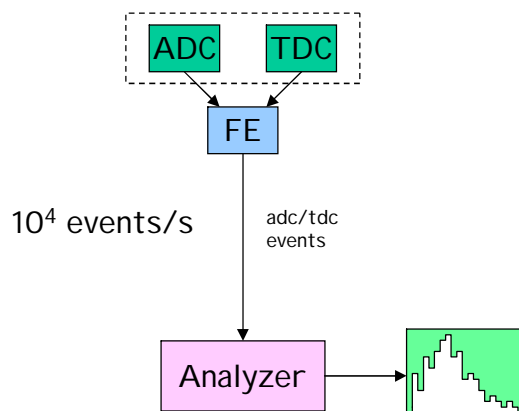
# FAL

- For simple experiment, **f**ront-end + **a**nalyzer + **l**ogger are combined in single process
- Avoid overhead of inter-process communication
- Only single process needs to be started
- Disadvantage: crashing analyzer stops DAQ



## "Histogram" readout

- Event-based readout
  - ADC/TDC read by frontend
  - polled/interrupt events
  - analyzer does histogram filling in software
- Histogram readout
  - Histograms filled in hardware
  - periodic events
  - analyzer displays histos



Midas

99/100



- Simulation
- Configuration
- Running a small experiment
- Utilities and usage
- Known deployment
- Advanced features
  - Customization of run control and monitoring
  - Special front-end features
- Advanced online demo

