

Online Processing in the H1 Experiment at HERA

thanks to
R.Gerhards,C.Grab,S.Levonian,J.Martyniak,T.Mkrtchyan,C.Nowak,J.Nowak,
P.Fleischmann,M.Vorobiev

Outline

H1 Experiment

L45 trigger

New L45 scheme

Event Repository

Data Flow

Process and Corba Object Management

Monitoring - Online Histograms, Event display, Emergency Messages, Output Log

Calibration

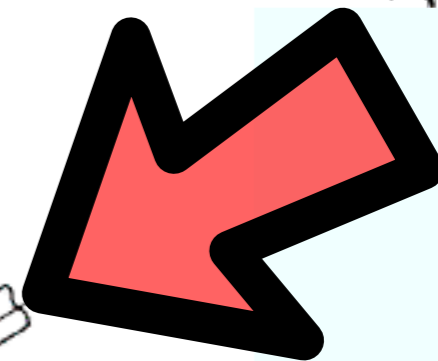
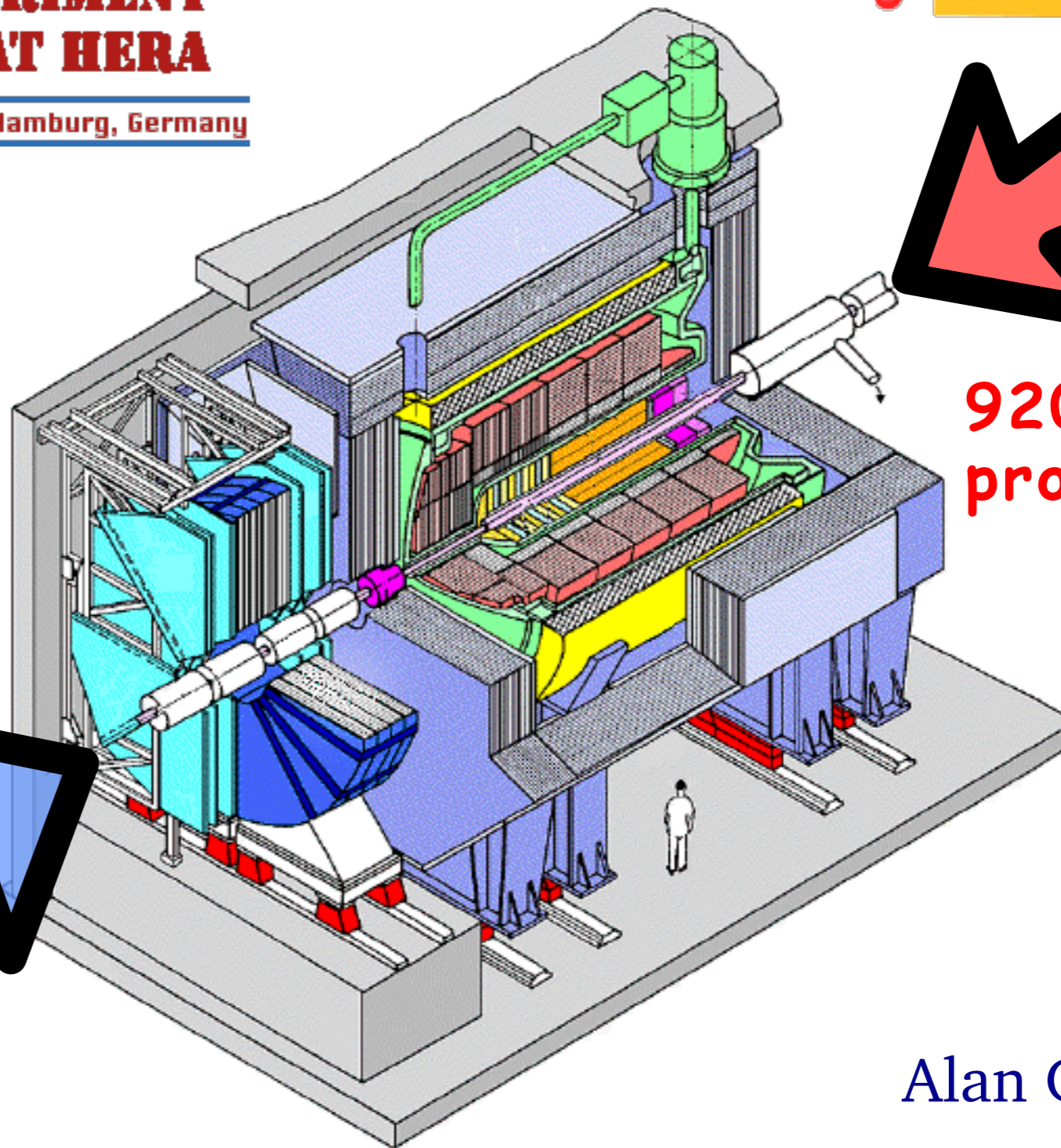
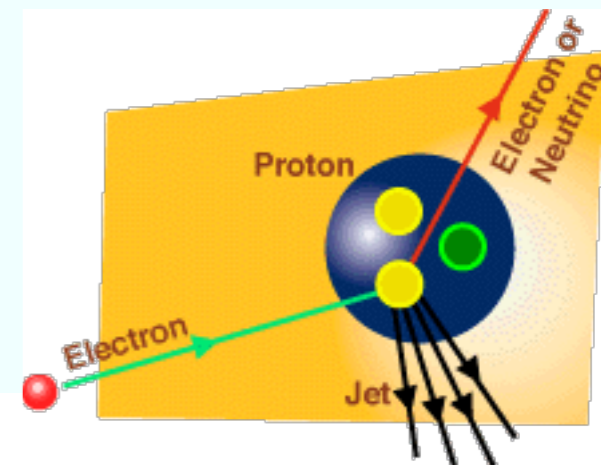
Trigger Algorithm Steering

Conclusions

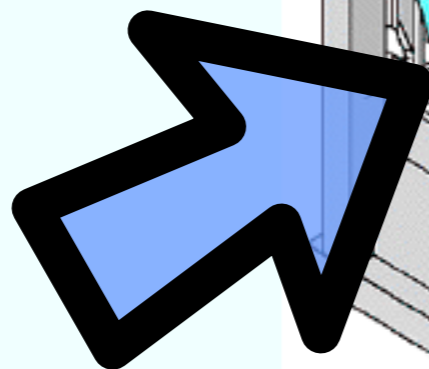


EXPERIMENT AT HERA

H1 Collaboration at DESY, Notkestr.85, D-22607 Hamburg, Germany



920 GeV proton

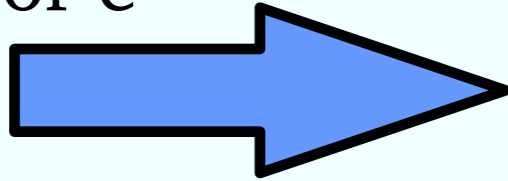


30 GeV positron or electron

Alan Campbell

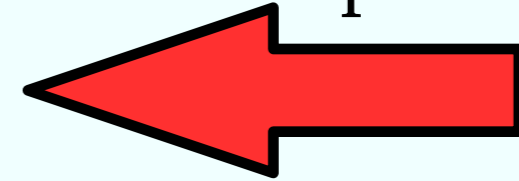
L45

e^+ or e^-



H1

p



96ns

between beam crossings

H1 Events

60Hz

1kB-1MB , ~100kB

average

~6 MB/s sustained average

L45 trigger

Full event reconstruction

Background rejection (eg vertex position)

Event Classification (assign to output stream)

Finders - select special physics channels

Downscale soft physics

Monitor & Calibration



Storage

~3 MB/s

RAW

DST

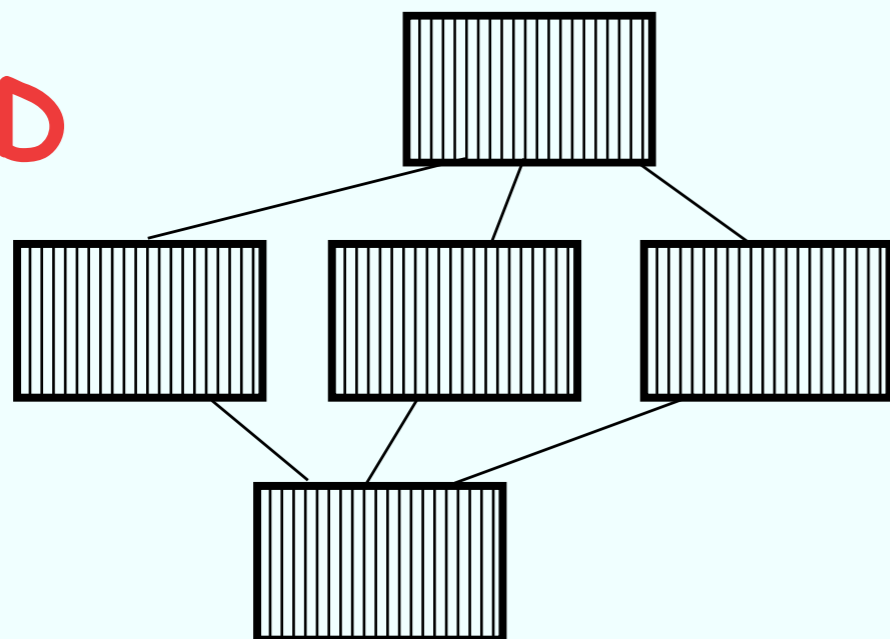
RANDOM

LED...

Data taking in "runs" of few minutes to one hour.
A run has fixed trigger and readout settings.

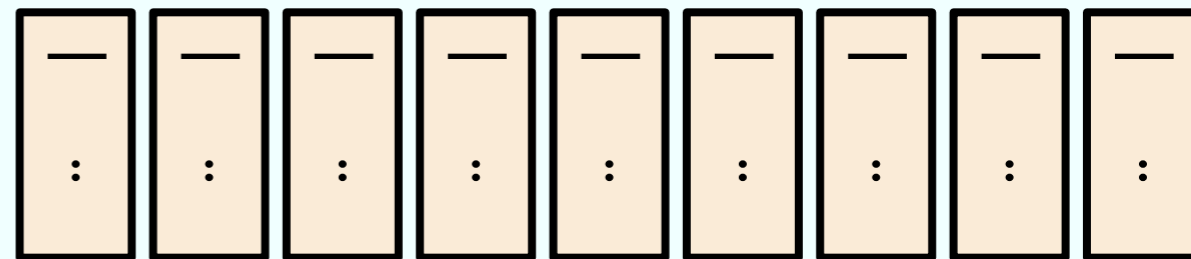
L45 upgrade HERAII

OLD



VMEbus processors
standalone programs

NEW



Networked PCs

- Same platform as offline computing (identical software) -resources can be shared
- lower cost -expandable network & cpu -same framework for reprocessing

Approach:

standards based event distribution framework
CORBA for data and control -> multiple language bindings
C++/Fortran/C -> link binary code from H1 standard libraries
Python -> control & setup scripts
Java -> histogram collection & display, run control

Investigations showed that data transfer via CORBA is fast (>7MB/s on 100Mbit network) with <1ms/call overhead => transfer in chunks of ~500kB.

What is CORBA ?

Common Object Request Broker Architecture

open, vendor-independent, standardised architecture and infrastructure for remote method call via network

standard protocol IIOP over TCP

multiple implementations including open source

any computer, operating system, programming language

IDL interface definition language independent of programming language

IDL compiler produces stubs (client) and skeleton (server) code in target language

IDL

```
#include "logger.idl"
#include "event.idl"
#include "perf_monitor.idl"

typedef sequence < Event > EventSeq;

interface eRepository {
    // Raised when trying to register to a repository which is already
    // closing ( the eofmarker has been read )
    exception ClosedException{
        boolean notified;
    };

    void storeEvent(inout Event e, in unsigned long tId) raises( ClosedException );
    void storeSeq(inout EventSeq eSeq, in unsigned long tId) raises( ClosedException );
    void storeBarrier(inout Event e, in unsigned long tId) raises( ClosedException );
};
```

C++

```
inline
void MReader::storeSequence( EventSeq_var & seq ){
    try{
        _writingReps[0]->storeSeq( seq.inout() , _gId );
    } catch( ... ) {
        cout << _name << "Exception while storing sequence." <<
endl;
        throw;
    }
}
```

What are threads ?

threads are multiple programs running on same data
omniorb is multi-threaded ... it allows multiple simultaneous calls
user code must care for synchronisation where necessary

ORB takes care of setting up tcp connections and reusing them where appropriate
omniorb is careful not to make unnecessary context switches

synchronisation

MUTEX - mutual exclusion - prevents a block of code from being executed simultaneously in several threads

RECURSIVE MUTEX - needed for thread safe wrapper around legacy fortran library

CONDITION VARIABLES - allow threads to know when a condition is reached (thrown)

C++

```
pthread_cond_init( &_confirmCond, NULL );
pthread_mutex_init( &_commonLock, NULL );
}
//-----
-
void BarrierSrv_i::unRegisterFromBarrierServer( unsigned long iId
){
    MutexGuard guard( &_commonLock );

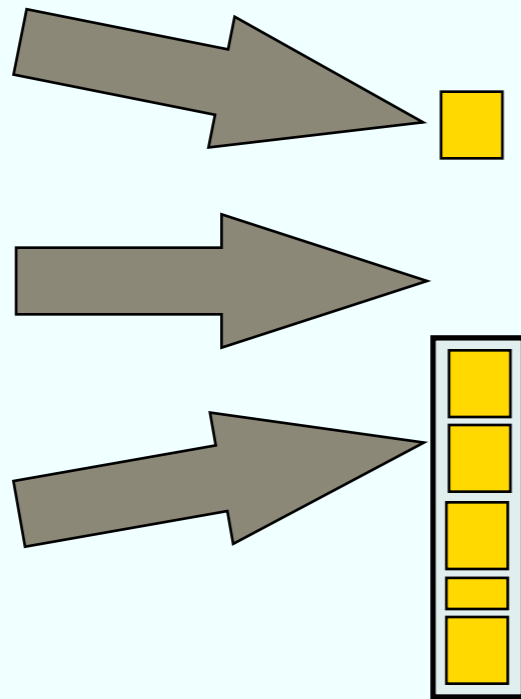
    delete _orbiMap[ iId ];
    _orbiMap.erase( iId );

    --_registeredNum;
    cout << "BSRV: input with iId: " << iId << " removed." << endl;

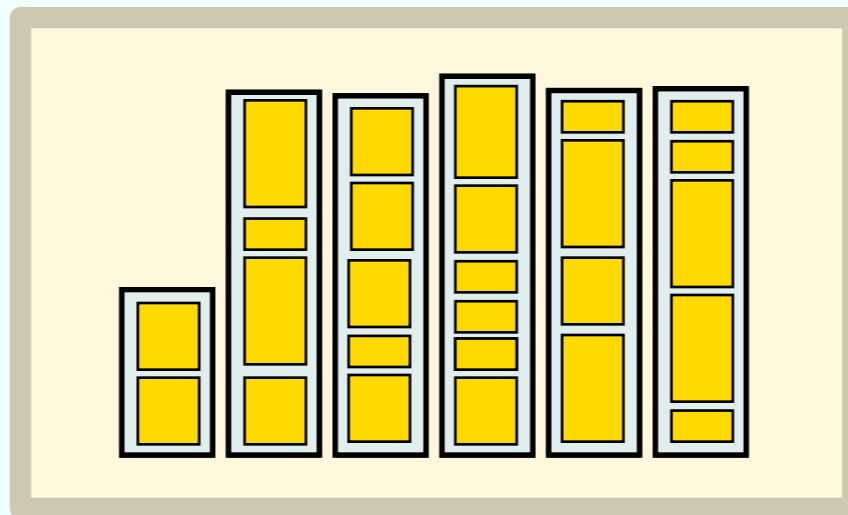
    if( !_registeredNum ){
        cout << "BSRV: all inputs left, stopping the barrier server."
<< endl;
        _stop = true;
        pthread_cond_signal( &_confirmCond );
    }
}
```

Event Repository - 1.Basics

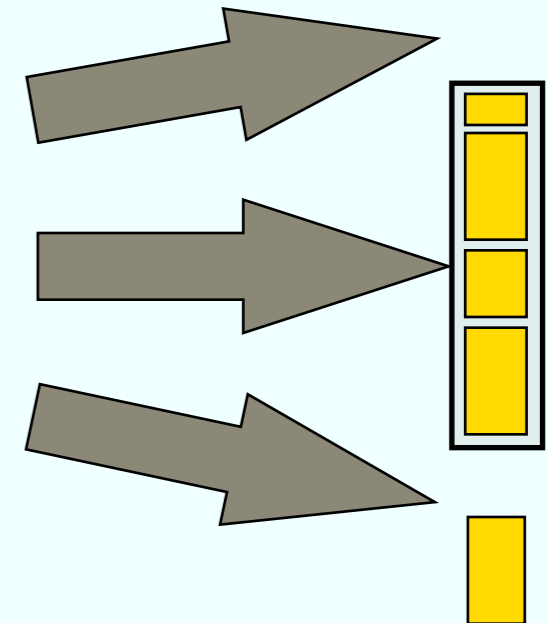
writer tasks



Event Repository FIFO



reader tasks



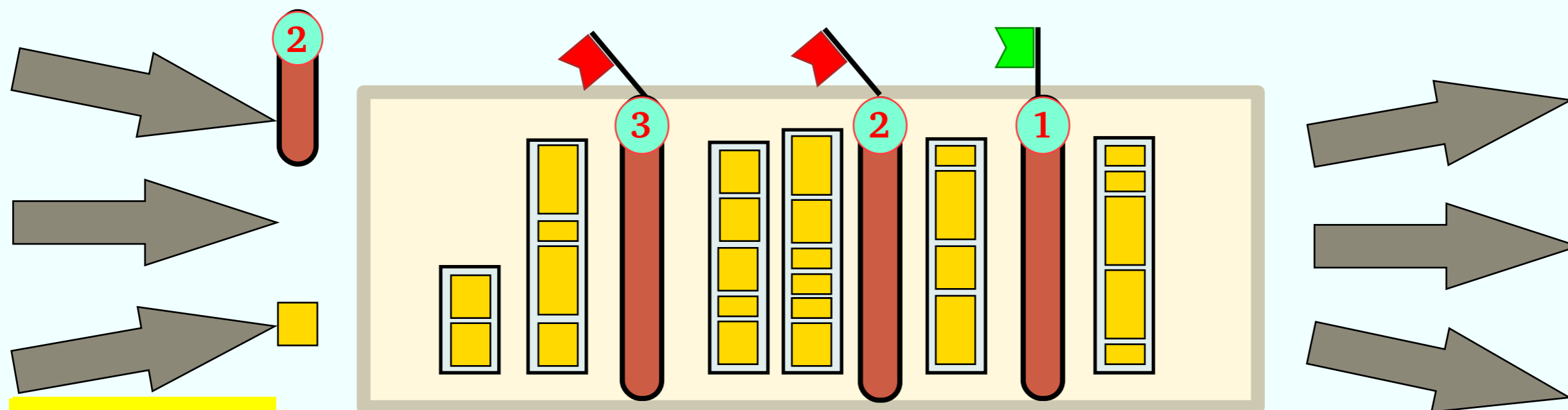
write_event
write_sequence

read_event
read_sequence

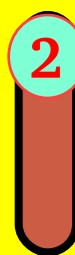
a first-in first-out event store
read and write similar to sequential file access
events stored in repository as suitably sized sequences
sequences are created when single events are inserted
only entire sequences are transferred between repositories via network
multiple writers & readers simultaneously - multi-threaded orb
no expensive data copy required
used both for event collection and distribution

Event Repository - 2.Synchronisation

Event Repository FIFO



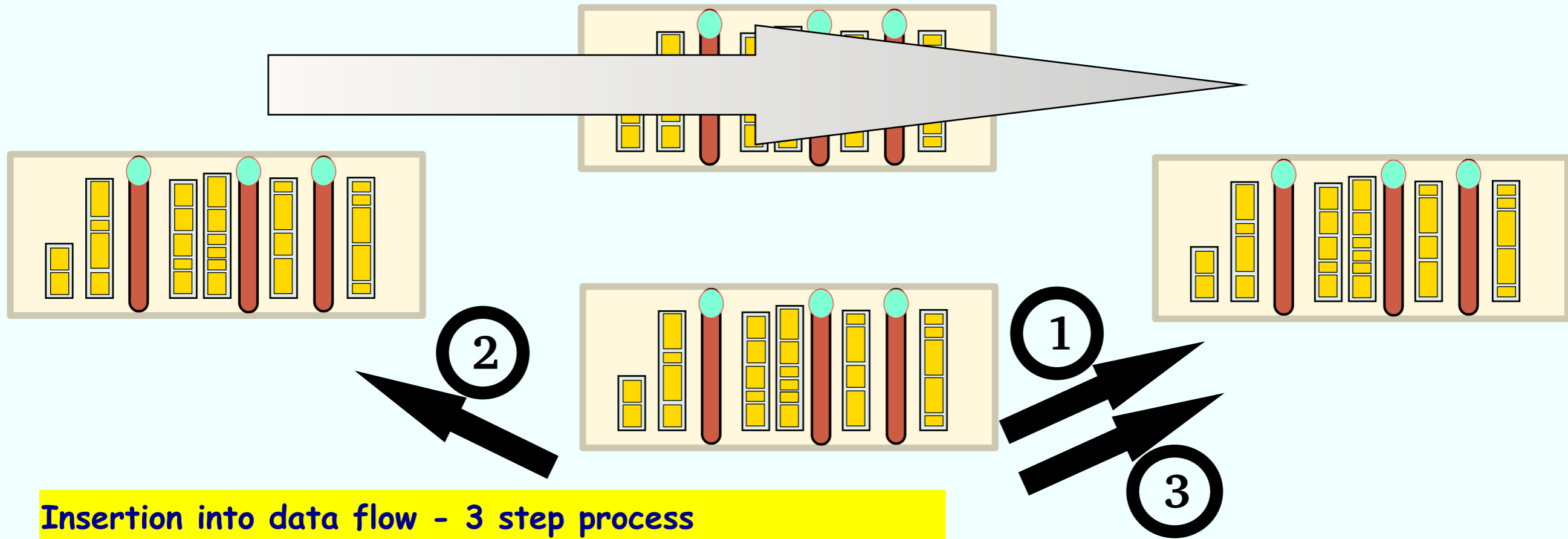
Barriers



separate event flow into blocks (eg runs), events stay within a block
all output events in a block arrive before any event from the next block
events may be dropped from (eg trigger reject) or added to (eg extra calibration data) a block
barriers are eg run start/stop, file begin/end, nth event calibration trigger

Barriers assigned incremental number when first written to its source repository
all writers must write a barrier before any readers can read it
writers write event sequences in front of a barrier which they have not yet written
as soon as all writers have written a barrier it becomes readable
barriers are not removed until read by all readers
a reader can read sequences behind barriers he has already read
each barrier is distributed to all repositories
repositories must be large and barriers infrequent to not hinder data flow

Event Repository - 3. Linking



Insertion into data flow - 3 step process

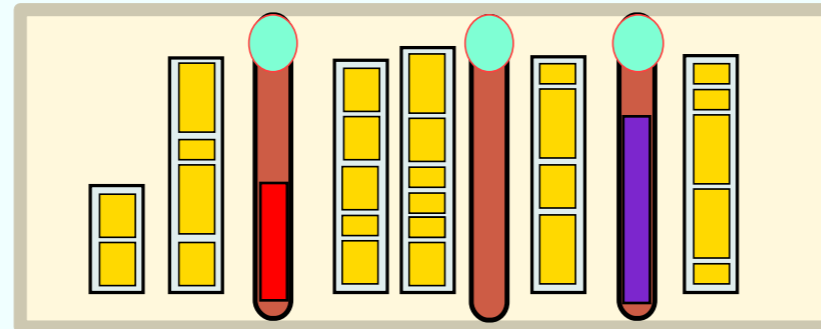
- 1 tell sink repository we will deliver front-most non-readable barrier
- 2 tell source repository we will read from front-most barrier
- 3 tell sink we will deliver only from barrier number given by source repository

Similar care needed to detach from flow normally or abruptly

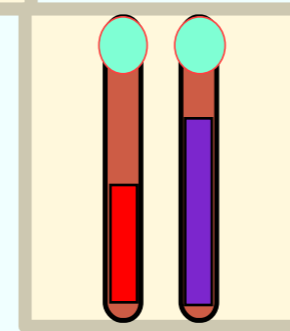
Event Repository - 4. Barriers with Data

Barriers are "broadcast" to all repositories and mark timestamp in dataflow.
Data Attached to barriers provides distribution of constants.

Event Repository FIFO



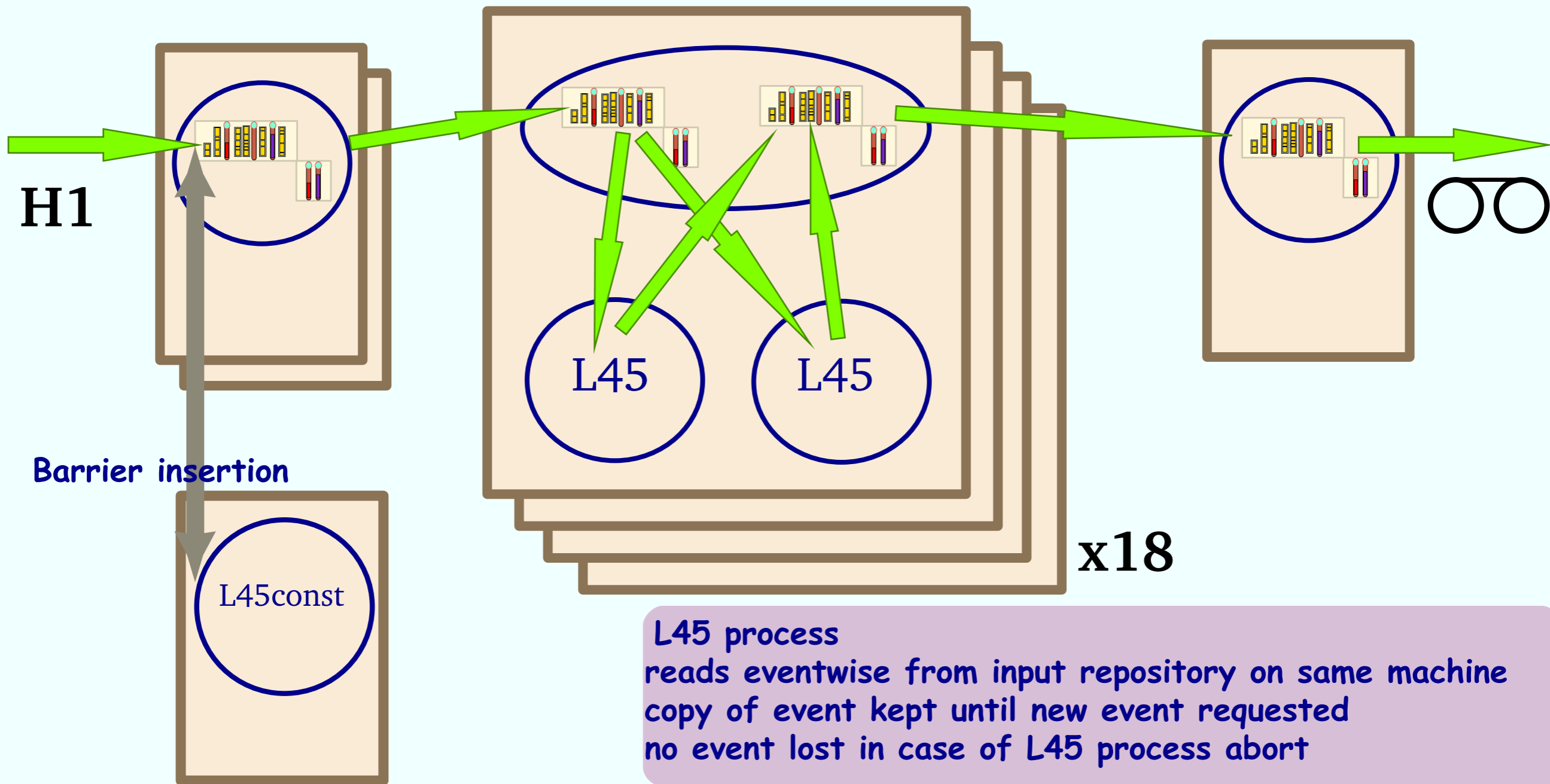
ensures same constants for same block
on all processing nodes
avoids multiple access to database
timely distribution of run-start records
ie run settings



Persistent Barrier Cache

When barrier with data is removed from repository it enters a cache, replacing last barrier of this type.
New readers first read the barriers in the cache to obtain all current constants.

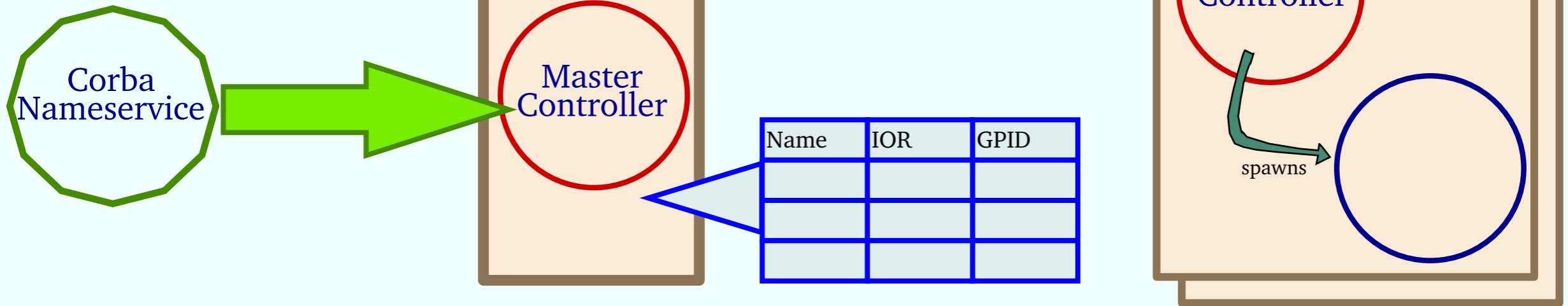
Overall Dataflow



L45 process
reads eventwise from input repository on same machine
copy of event kept until new event requested
no event lost in case of L45 process abort

L45const - extra barrier creation process
allows for multiple input processes
inserts additional barriers in data flow if new constants available

Controller



Controller - process creation and corba object handling (python & omniipy)

Master controller

starts slave controllers via ssh

provides global process ids

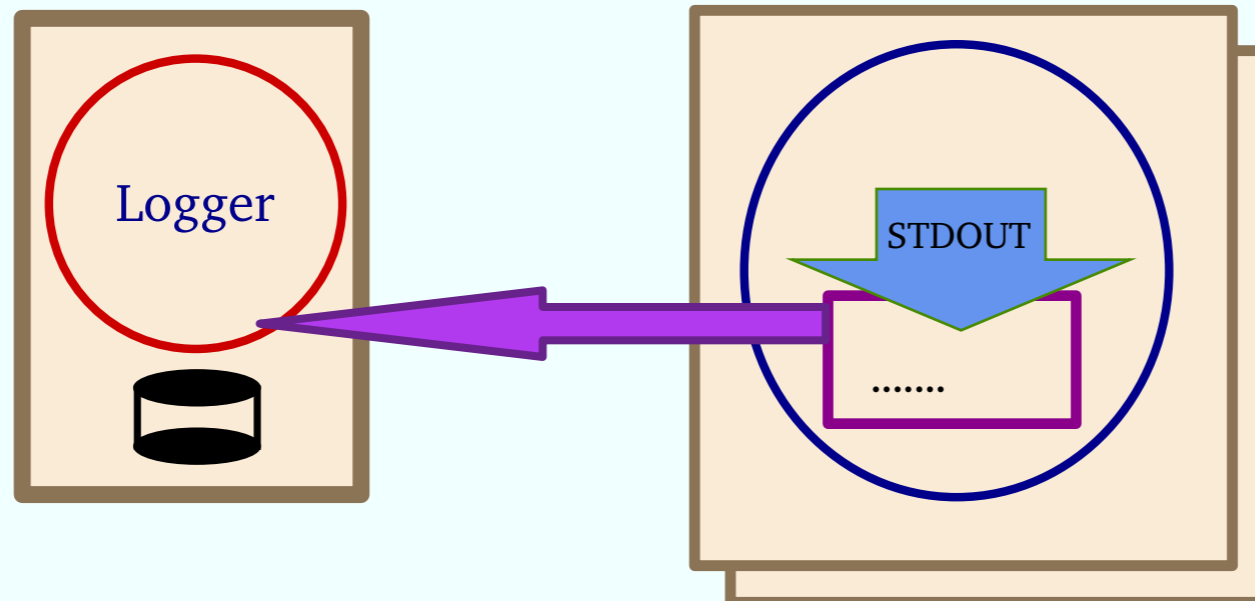
maintains lists of object references eg of event repositories, histogram servers

maintains list of repositories and their reader/writers

Slave controllers start processes for master, check & restart processes

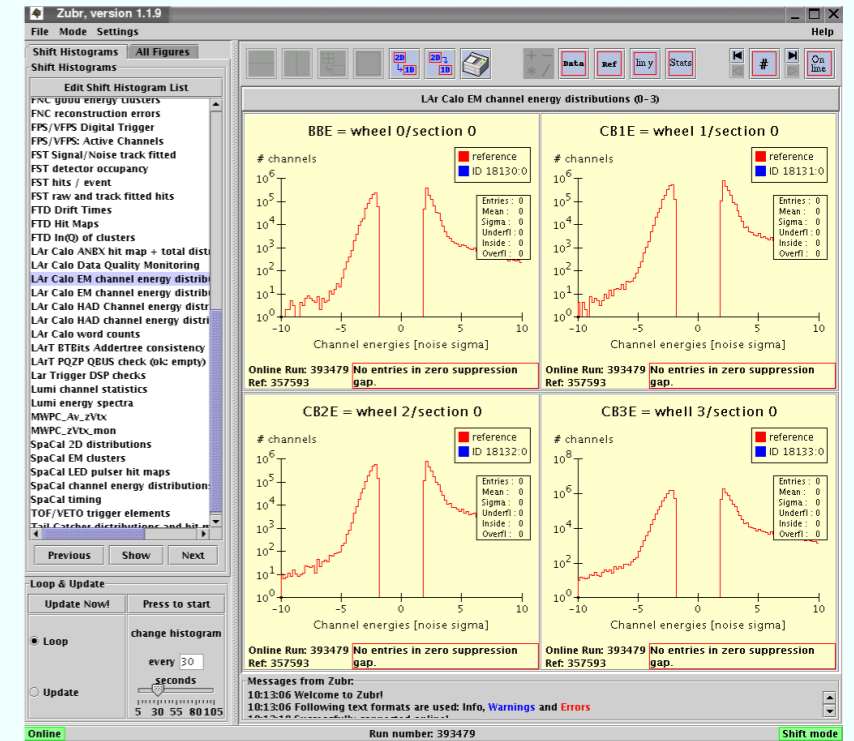
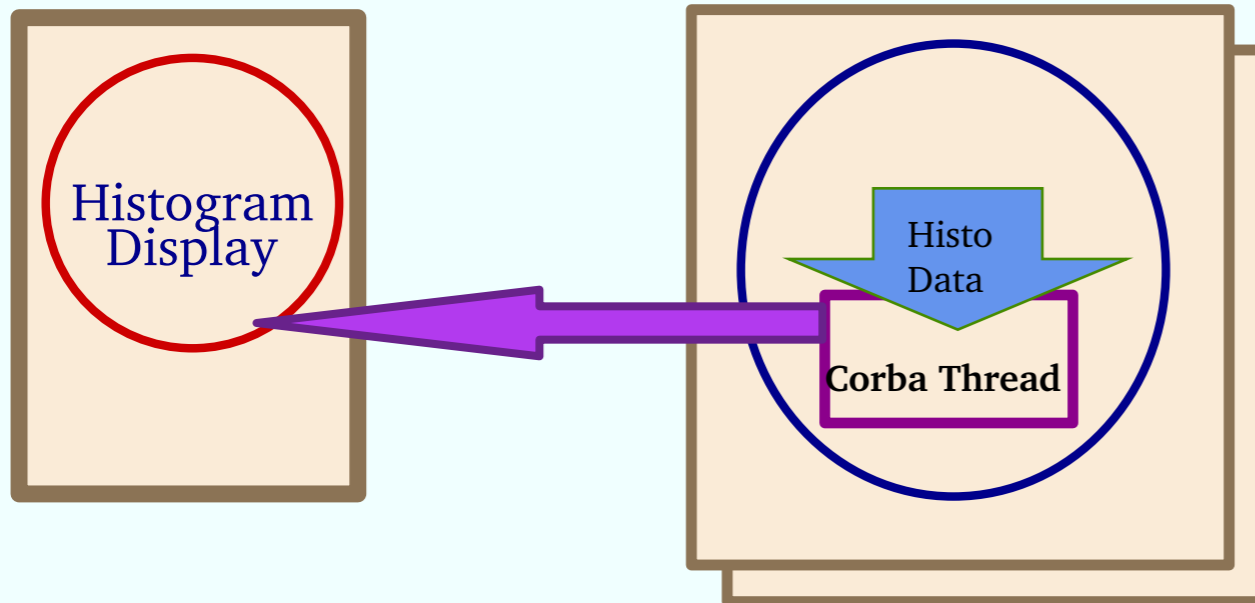
controllers are restartable

Logger



C++ thread reads STDOUT
collects lines
transfers to logger via corba
logger process dumps to disk

Online Histograms

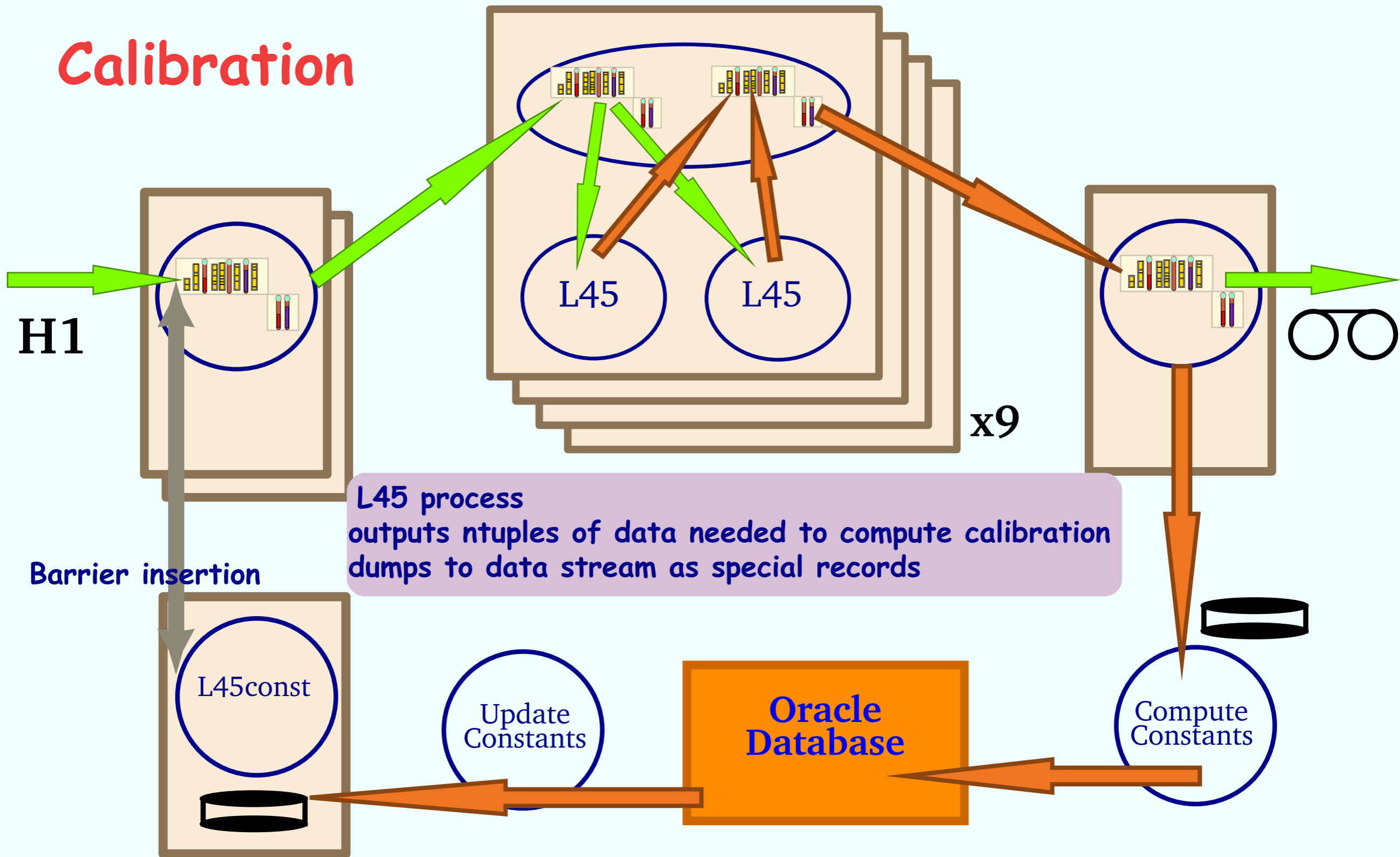


Corba thread C++ calls fortran histogram package (LOOK)
LOOK modified to add recursive mutex
Histogram display fetches and sums data from all processes(Java, JAS, corba)
New: web access via corba web server in python (omnipy, biggles, svg)

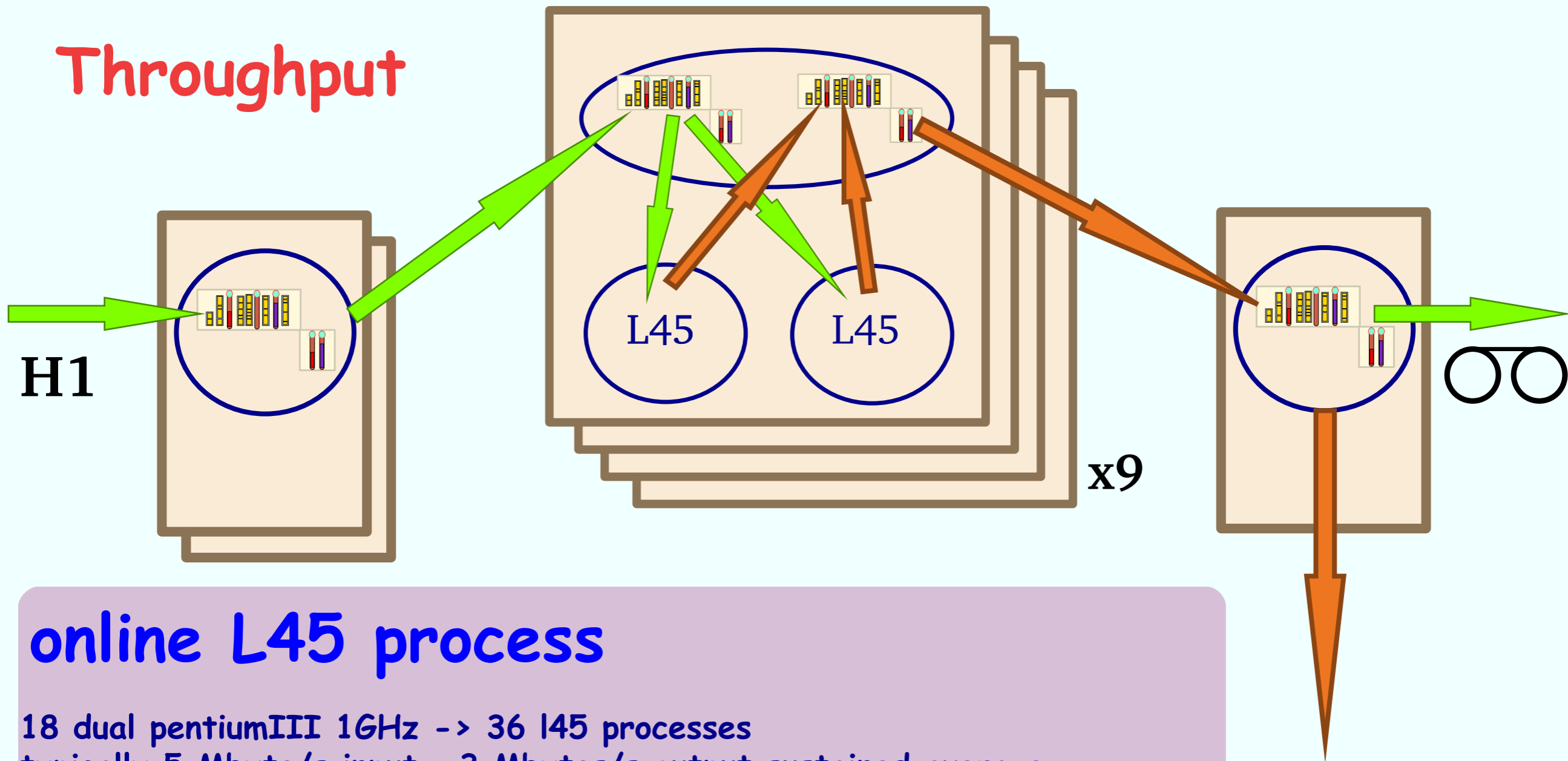
Online Event Display

rare events may be selected for the online event display
selection criteria are sent as 'constants' via I45const
events selected in L45 are written as special records
latest event is kept in logging process and fetched for display via corba

Calibration



Throughput



online L45 process

18 dual pentiumIII 1GHz -> 36 l45 processes
typically 5 Mbyte/s input , 3 Mbytes/s output sustained average
50 Hz of 100kbyte raw data events

reprocessing

input from dcache (with prestage requests))
13 dual Xeon 2.8GHz machines -> 52 repro processes
typically 3.4 million events/day (40 Hz) 6.5 MB/s output

Trigger Algorithm Steering

algorithm defined in text
stored in database

```
'MODULES;'  
'L1=L_RNDM,L_L1_L5SKIP;'  
'QT=I_QT_NHITCRJE,#L1;'  
'CJC=I_CJC_NUPSTRTRACK,#QT;'  
  
'L1RESET;'  
'L_L1_ALLST      = 0-127;'  
  
'TRIGGER;'  
'* Accept Very High ETJET events;'  
'L4_HS_VHETJET  =  L_L1_VHETJET      &'  
'                R_CJC_ZVX>-60.      &'  
'                R_CJC_ZVX<110.      &'  
'                R_ENFL_ETJET>20.0   &'  
'                L_L1_SETSKIP        &'  
'                L_L1_ECBIT07        &'  
'                :ACCEPT:0.00:CONTINUE;'  
'***** reject obvious background *****;'  
'L4_CJC_VTXZ_DOWN= L_L1_ALLST      &'  
'                R_CJC_ZVX>100.0:RESET_ALLST:0.1;'  
  
'ACCEPT;'  
'H1REC      :1.0;'  
  
'HISTOGRAMS;'  
'MAIN      = 50,0.,5000000.# 1,0;'  
'QT        = 25,0., 50000.# 1000,0;'  
'L1        = 25,0., 100000.# 2000,0;'  
'CJC       = 50,0.,2500000.# 4000,0;'  
'R_FNC_ESUM = L4_EFS_FNC_Q2   :60, 10.,1210.;'  
'R_CJC_ZVX  =                  200,-250.,150.;'
```

definition of processing modules
variables which can be calculated by module
dependencies between modules

definition of trigger masks

statements executed until accept/reject
modules called only if variable needed
statement aborts if a subcondition is false
if all subconditions true action taken
ACCEPT/REJECT/RESET_MASK
RESET_MASK is reject only if no bits remain
CONTINUE specifies test statement
fraction specifies monitor/scaledown
histograms per statement records result

extra processing if accept

module timing histograms
variable histograms
variable histograms for individual
statements

Summary & Conclusions

H1 Experiment High Level Filter

- * Event Repository organises data flow and synchronisation
- * multithreads overlap data transfer and processing
- * successful mix legacy FORTRAN and with C++
- * very stable system
- * CORBA gives language & network independence
- * entirely open standards, open source