Status of
Event Display

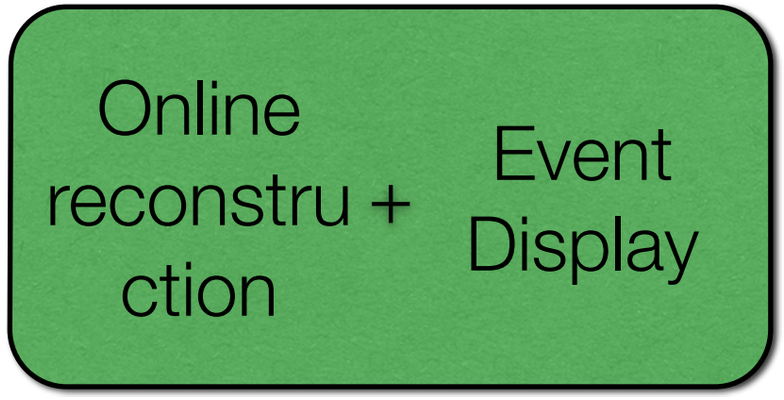
Jeremi Niedziela

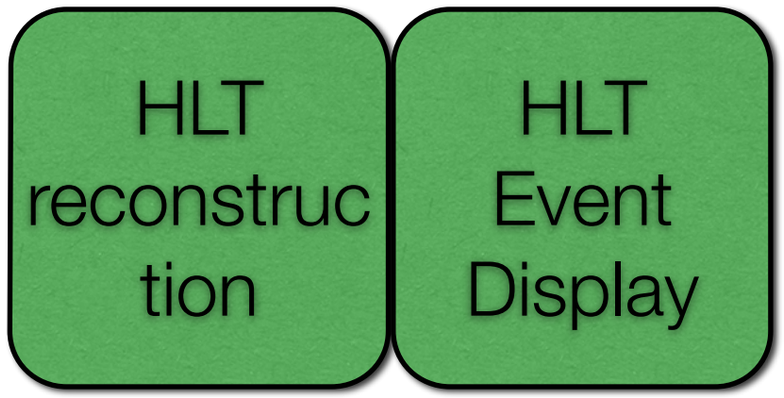
Outline

1. Event Display System architecture
2. Installation in P2
3. Status of work

Changes in architecture

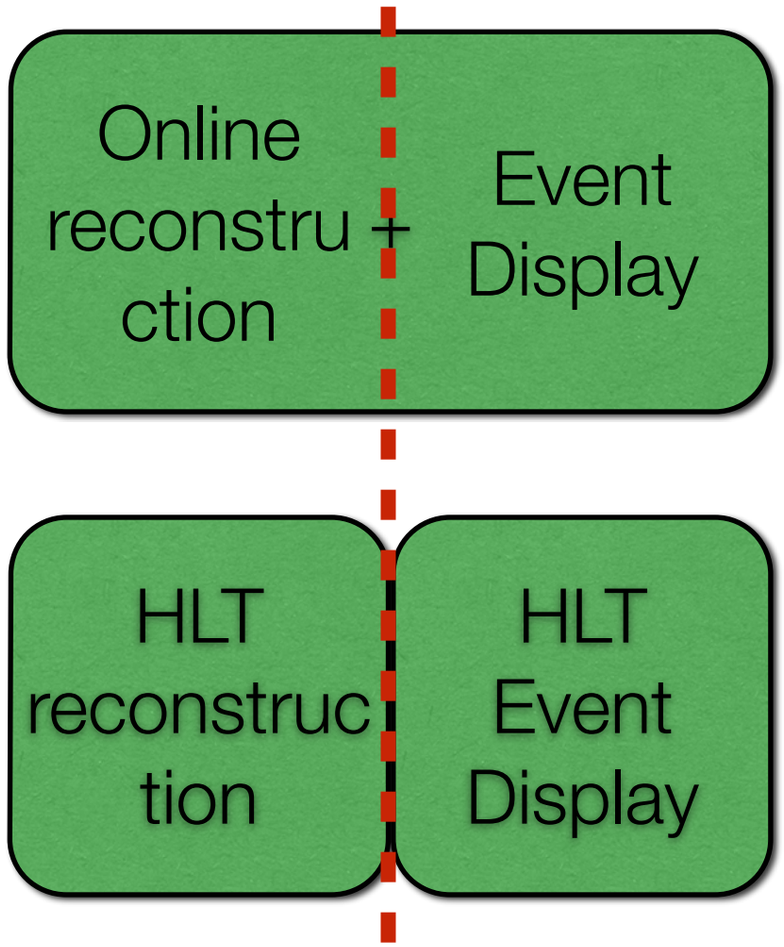
Run 1:





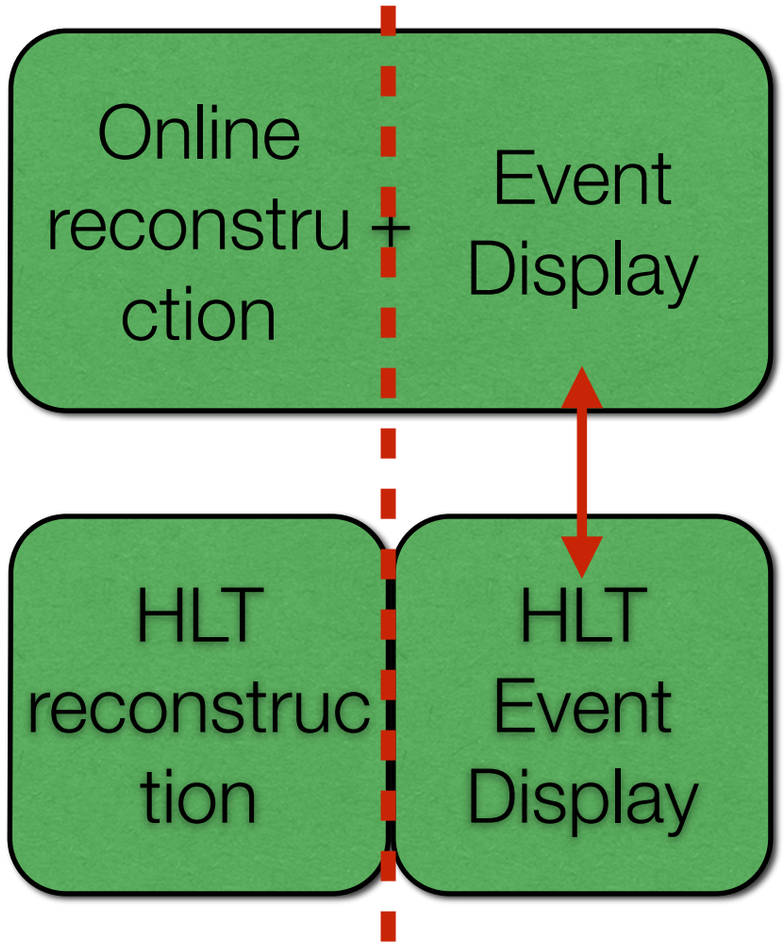
Changes in architecture

To gain **stability** ⟶ **separate** reconstruction from visualisation



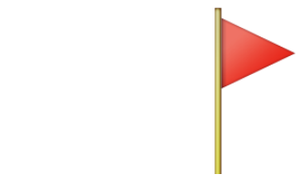
Changes in architecture

Event Display switching between **different sources** of events (i.a. Offline and HLT)



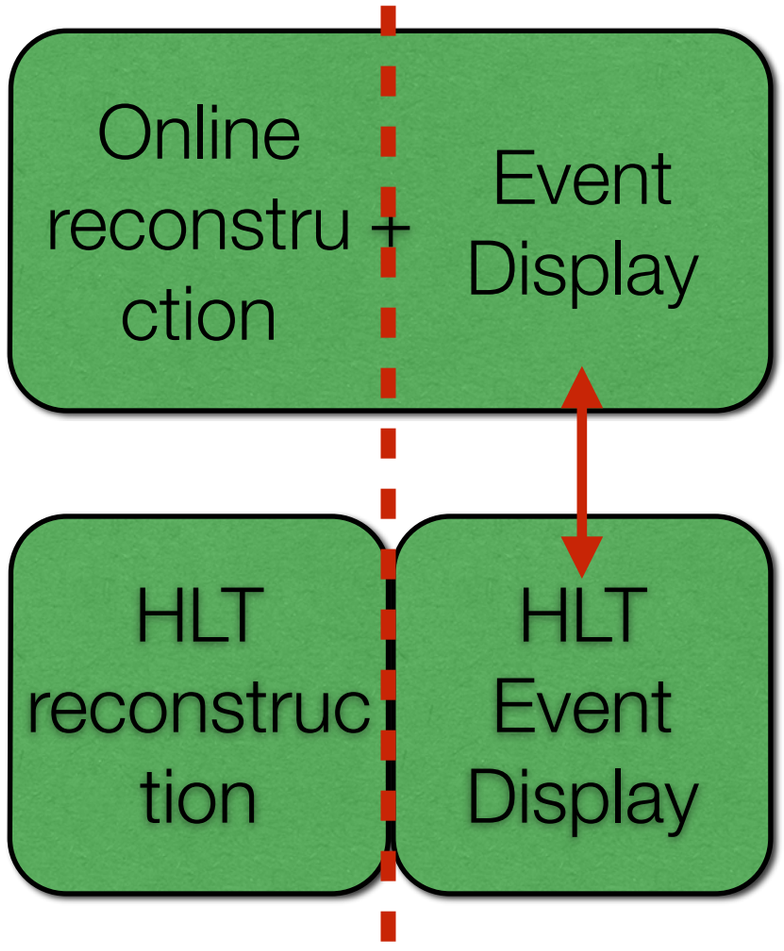
Changes in architecture

Add new **features**



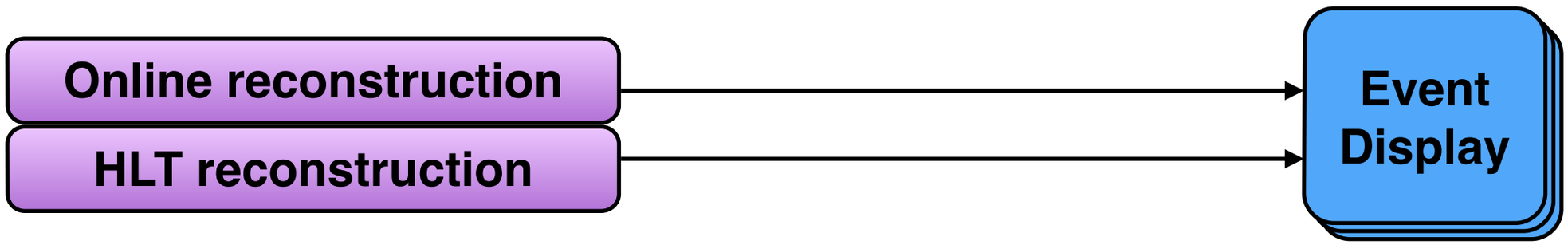
bookmarks

history browsing



Event Display architecture

Separation of visualisation and reconstruction + common Event Display for all data sources:



✓ separation of reconstruction and visualisation

✓ improved stability of Event Display

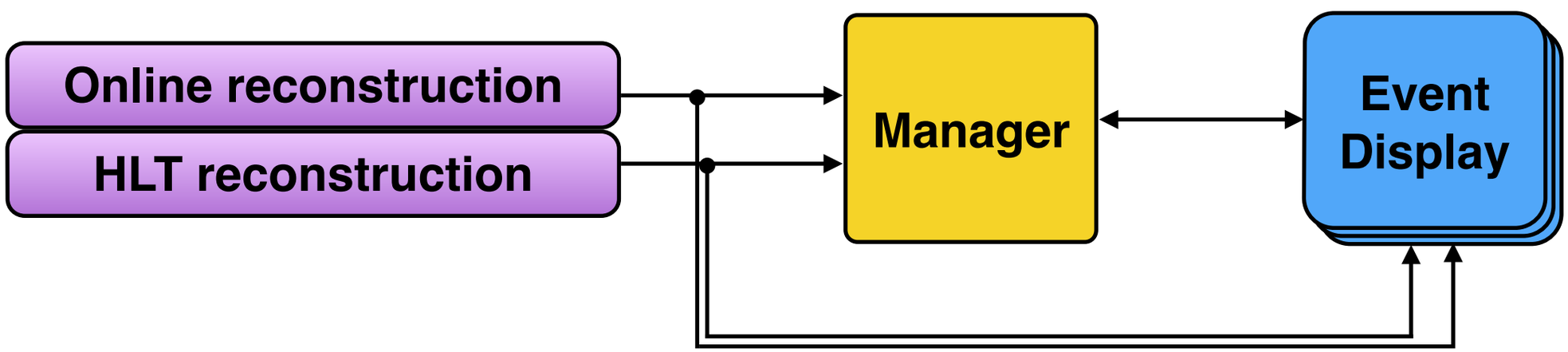
✓ Event Display and Online Reconstruction installed in P2

X running online reconstruction with current runs

X sending HLT events and receiving them on Event Display side

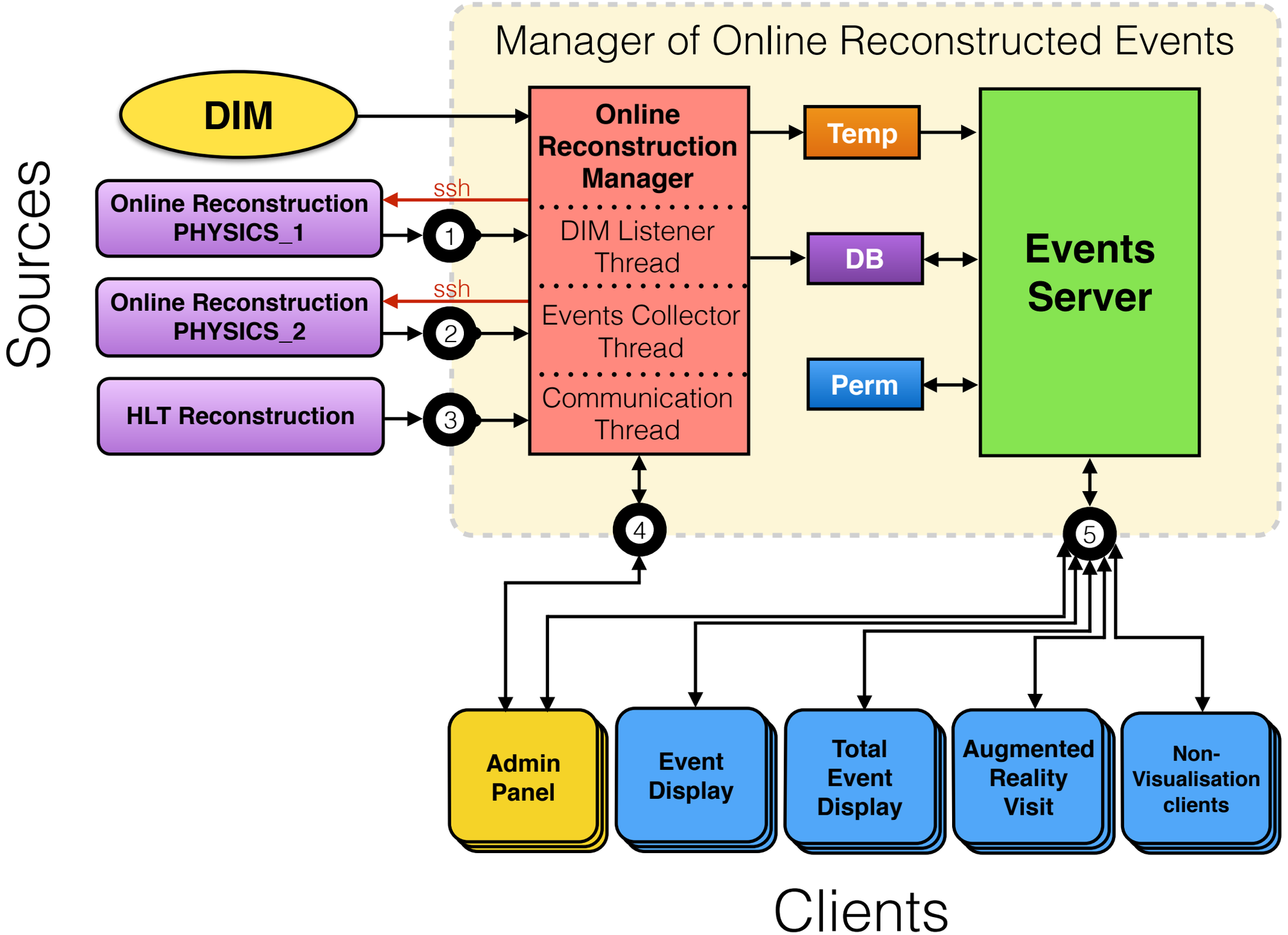
Event Display architecture

New features (**bookmarks, history**) by adding Manager of Online Reconstructed Events:



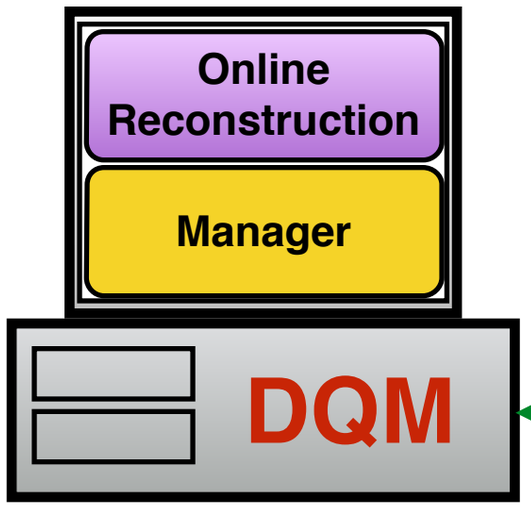
- ✓ Manager was written from scratch, it collects events, manages saving them on disk, serving and marking events on client's requests
- ✓ Event Display was adapted to take advantage of new functions
- ✓ Manager was also installed in P2

Recent change in Event Display system design



System's organisation in P2

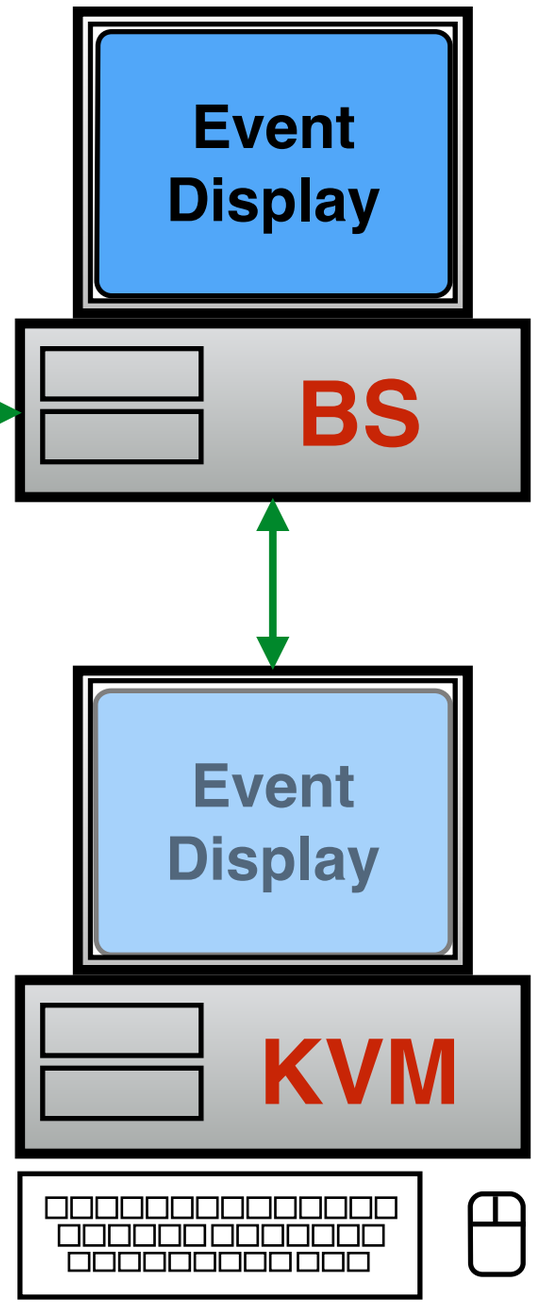
Counting Room



Manager of Online Reconstructed Events is now started by a Cron (check every one minute).

Online **Reconstruction** is managed by Manager (starting and killing)

ALICE Run Control Center



Event Display runs on the central big screen machine (monitored by Cron).

Shifter can **interact** with Event Display using KVM machine.

Summary

What has been done:

- ✓ separation of Event Display System into visualisation, reconstruction and manager,
- ✓ stability improvements,
- ✓ new features implementation,
- ✓ installation in P2,
- ✓ agreement on integration with HLT reconstruction

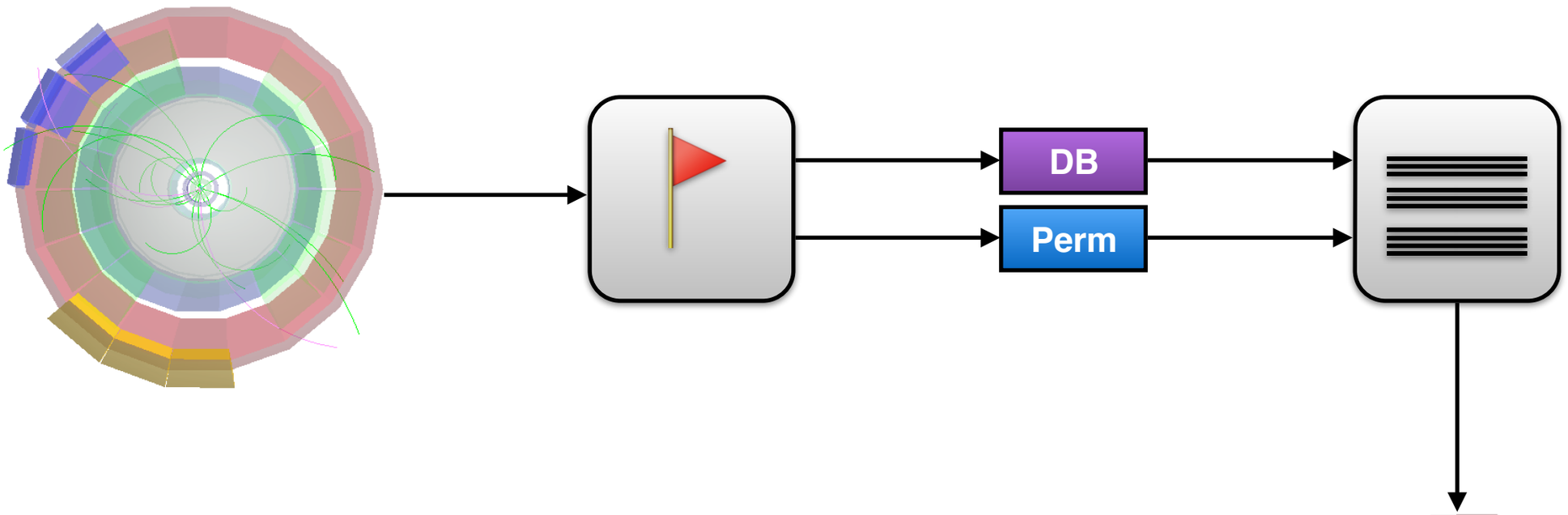
Summary

What is to been done:

- X tests of reconstruction with global runs,
- X automatisation of installation in P2,
- X minor bugs fixing,
- X appearance improvements.

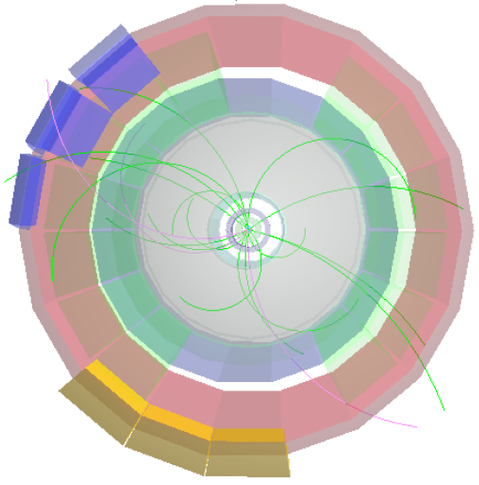
Backup

User point of view



User gets:

- **One Event Display** for different sources of data (not crashing due to reconstruction failures and not freezing when communication with server is broken),
- Complete **history** of recent events, which can be browsed and filtered (event by number, next, last, matching some query…),
- **Bookmarks**



Event Display architecture

API can be easily extended for different data types.

Currently supported types:

- bool
- long
- custom structures
- vector of structures
- AliESDEvent objects

To add **new type**, simply override Send() method of Event Manager.

If **new socket** needed, add it in enum StorageSockets and create in CreateSocket() method of Event Manager.

Extending event manager

AliStorageEventManager class:

```
class AliStorageEventManager
{
public:
   static AliStorageEventManager* GetEventManagerInstance();

   void Send(std::vector<serverListStruct> list,storageSockets socket);
   void Send(struct serverRequestStruct *request,storageSockets socket);
   bool Send(struct clientRequestStruct *request,storageSockets socket,int timeout = -1);
   void Send(AliESDEvent *event,storageSockets socket);
   void Send(long message,storageSockets socket);
   void Send(bool message,storageSockets socket);
   void SendAsXml(AliESDEvent *event,storageSockets socket);

   std::vector<serverListStruct> GetServerListVector(storageSockets socket);
   AliESDEvent* GetEvent(storageSockets socket,int timeout=-1,TTree **tmpTree=0);
   struct serverRequestStruct* GetServerStruct(storageSockets socket);
   struct clientRequestStruct* GetClientStruct(storageSockets socket);
   long GetLong(storageSockets socket);
   bool GetBool(storageSockets socket);


   bool CreateSocket(storageSockets socket);
//...
}
```

Extending event manager

Send method for **bool**:

```
void AliStorageEventManager::Send(bool message,storageSockets socket)
{
   char *buffer;
   if(message==true)
   {
    buffer = (char*)("true");
   }
   else
   {
    buffer = (char*)("false");
   }
   message_t *replyMessage = new message_t((void*)buffer,sizeof(buffer),freeBuff);
   fSockets[socket]->send(*replyMessage);
   delete replyMessage;
}
```

Send method for **AliESDEvent**:

```
void AliStorageEventManager::Send(AliESDEvent *event, storageSockets socket)
{
   TMessage tmess(kMESS_OBJECT);
   tmess.Reset();
   tmess.WriteObject(event);
   TMessage::EnableSchemaEvolutionForAll(kTRUE);

   int bufsize = tmess.Length();
   char* buf = (char*) malloc(bufsize * sizeof(char));
   memcpy(buf, tmess.Buffer(), bufsize);

   message_t message((void*)buf, bufsize, freeBuff);
   fSockets[socket]->send(message);
}
```

Extending event manager

storageSockets enumerated type:

```
enum storageSockets{
   SERVER_COMMUNICATION_REQ=0,
   SERVER_COMMUNICATION_REP,
   CLIENT_COMMUNICATION_REQ,
   CLIENT_COMMUNICATION_REP,
   EVENTS_SERVER_PUB,
   EVENTS_SERVER_SUB,
   XML_PUB,
   NUMBER_OF_SOCKETS
};
```

Extending event manager

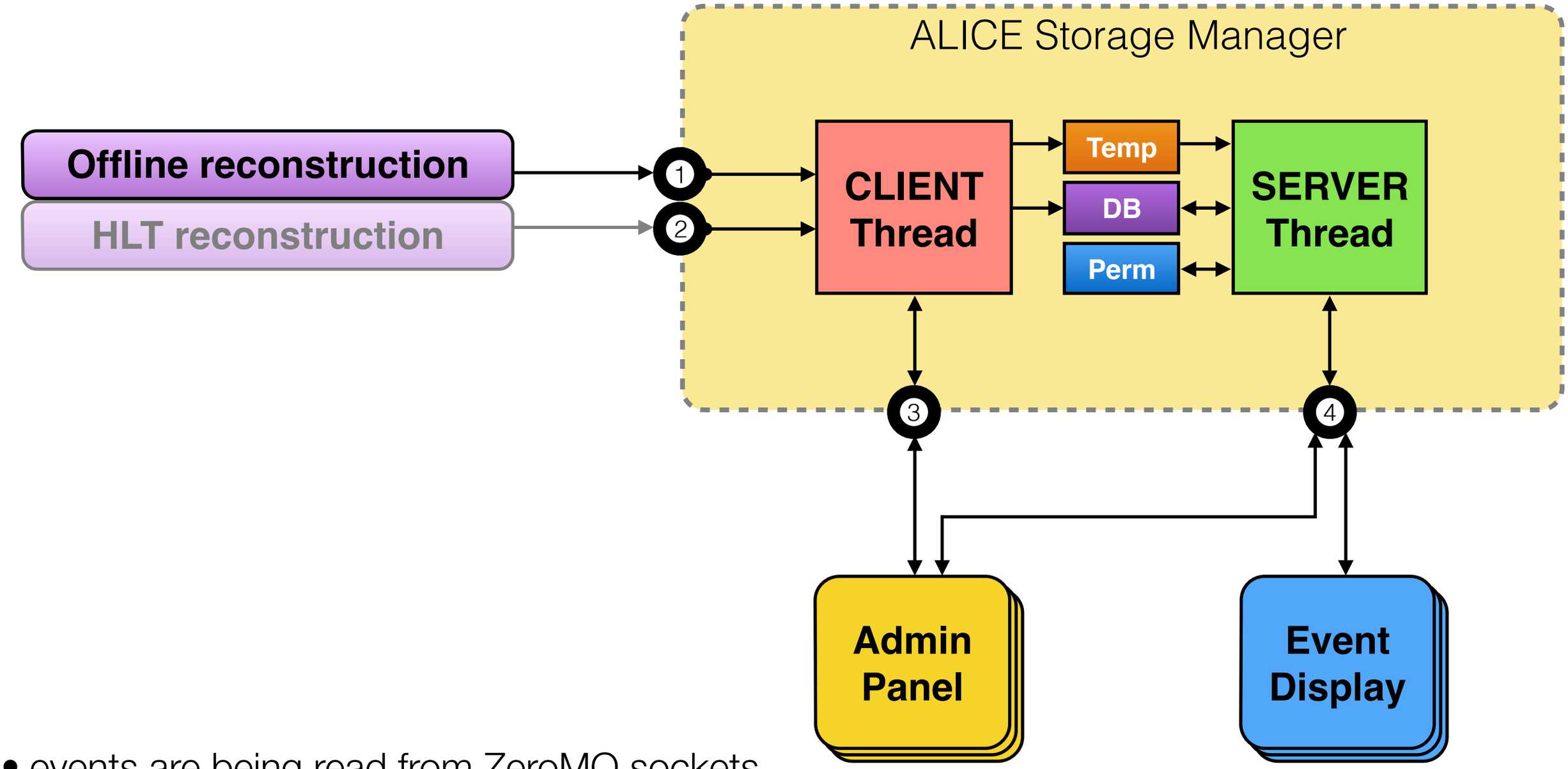
CreateSocket method:

```
bool AliStorageEventManager::CreateSocket(storageSockets socket)
{
    cout<<"Creating socket:"<<socket<<endl;

    switch(socket)
    {
        case SERVER_COMMUNICATION_REQ:
        {
            fSockets[SERVER_COMMUNICATION_REQ] =
            new socket_t(*fContexts[SERVER_COMMUNICATION_REQ],ZMQ_REQ);
            try
            {
                fSockets[SERVER_COMMUNICATION_REQ]->connect(Form("tcp://%s:%d",fStorageServer.c_str(),fStorageServerPort));

            }
            catch (const zmq::error_t& e)
            {
                cout<<"MANAGER -- "<<e.what()<<endl;
                return 0;
            }
        }
            break;
        case SERVER_COMMUNICATION_REP:
        {
            fSockets[SERVER_COMMUNICATION_REP] =
            new socket_t(*fContexts[SERVER_COMMUNICATION_REP],ZMQ_REP);
            try
            {
                fSockets[SERVER_COMMUNICATION_REP]->bind(Form("tcp://*:%d",fStorageServerPort));
            }
            catch (const zmq::error_t& e)
            {
                cout<<"MANAGER -- "<<e.what()<<endl;
                return 0;
            }
        }
            break;
//...
    }
}
```

Event Display architecture



- events are being read from ZeroMQ sockets,
- Storage Manager provides events (next, previous, last, by number, matching query, marked). Early version of API was prepared and its development is under discussion with HLT,
- Admin Panel (for experts) controls parameters of Storage Manager (occupation levels, size of chunks).