DAQ & data format

Peter Fischer

Institut für Technische Informatik, Universität Mannheim

Presentation given at the EUDET / JRA-1 review, 4.4.2006, Geneva

Introduction

- We have decided to use the present DEPFET DAQ as a starting point.

- Note: This DAQ has been written with a minimum effort, so
  - not enough documentation
  - still hard coded stuff
  - some error checking, no ‘recovery’ (but good enough)

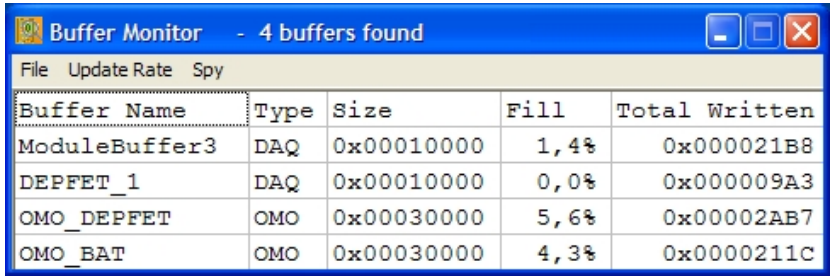
- This starting point must be improved:
  - need a clever way for process communication (across operating systems?)
  - need a ‘global controller’
  - need a better way to request Monitoring data
  - need ‘shared buffer through TCP/IP’
  - …

- This cannot be done by Mannheim (or Bonn), it is not our task, neither are we real experts…

Present DEPFET ‘Mini-DAQ’

- PC based, at the moment Windows based. Very light weight.

- DAQ Software is divided into many parallel tasks:
  - **several Producer** tasks read the hardware
  - **one FileWriter** task bundles events, writes to file and sends subsets for monitoring
  - There can be **several Online - Monitoring** tasks
  - **one Buffer Monitor** task allows to see what is going on
  - a **FileReader** can re-inject data into the monitoring

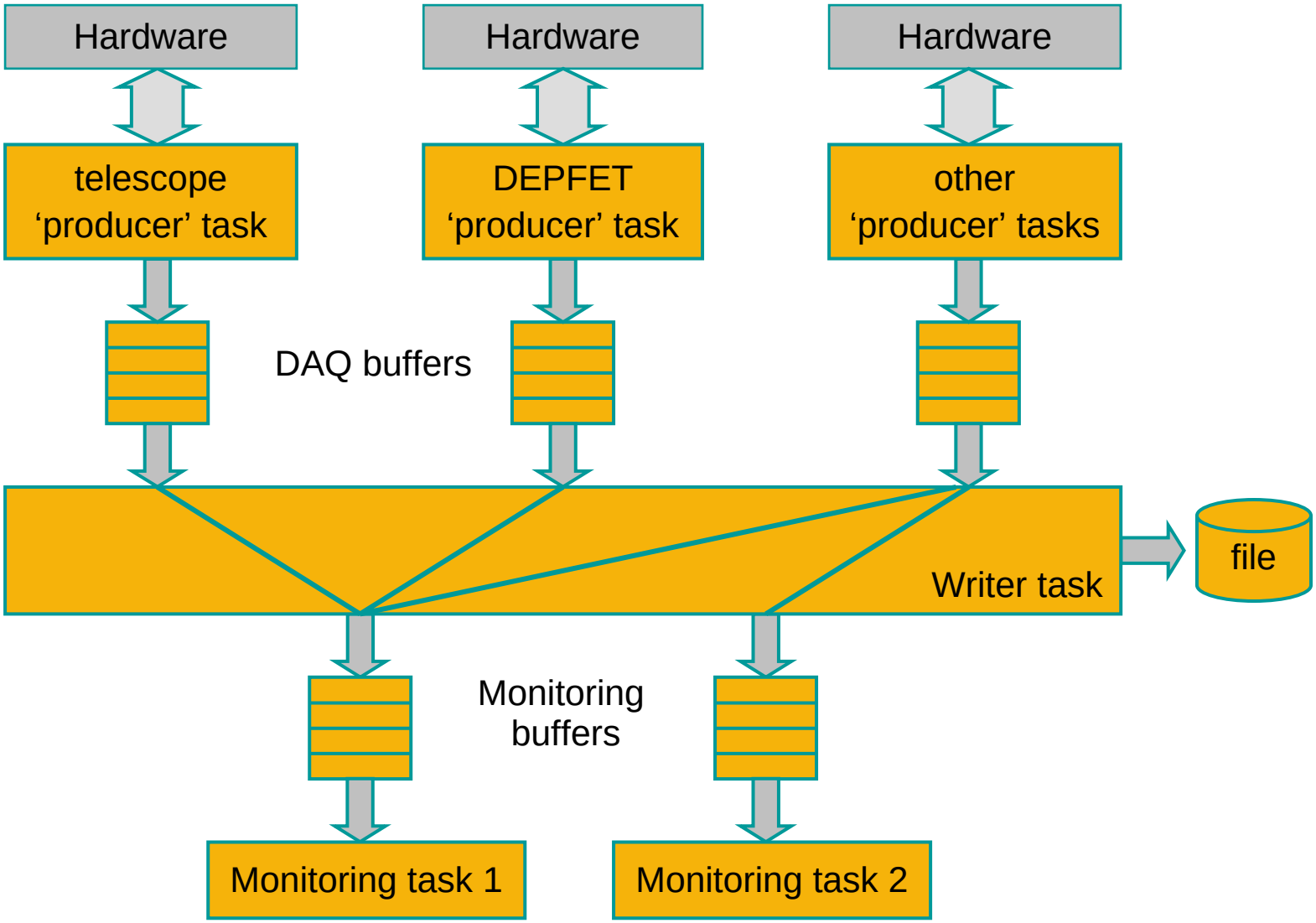

Buffer Monitor - 4 buffers found

File Update Rate Spy

Buffer Name	Type	Size	Fill	Total Written
ModuleBuffer3	DAQ	0x00010000	1,4%	0x000021B8
DEPFET_1	DAQ	0x00010000	0,0%	0x000009A3
OMO_DEPFET	OMO	0x00030000	5,6%	0x00002AB7
OMO_BAT	OMO	0x00030000	4,3%	0x0000211C

- Central data flow element: ‘shared buffers’
  - Used between Producers and Writer and between Writer and Monitors
  - Events have unambiguous trigger ID in the buffers for later sorting
  - Mutex elements are used to control access of the buffers
  - Uses begin-of-run (BORE), data (DATA), end-of-run (EORE) events.

Present DEPFET ‘Mini-DAQ’



Shared Buffers

- Organize data transport between tasks.
- Two types:
  - DAQ buffers from DAQ tasks to WRITER
  - Monitoring Buffers from WRITER to MONITORING tasks
- Two port buffer: one task can write, one task can read (no checks!)

- Very simple & efficient:
  - ~300 lines of source code for class implementation
  - writing & reading can be done simultaneously, arbitration only needed at start / end of transfers.
  - data blocks are always contiguous blocks of data so that tasks do not have to bother with wrapping pointers in ring buffers
- Can be extended to send data from PC to PC (via TCP/IP)

- Buffers are created by the writing task. (If they already exist, the task connects to the existing buffer)
- Buffers automatically disappear when nobody refers to them any more.

Shared Buffer header (part)

```
class TSharedBuffer {

static const unsigned int MAX_SHARED_BUF;

public:
   TSharedBuffer (AnsiString Name, unsigned int Size, SBufferType Type = BUFTYPE_DAQ);
  ~TSharedBuffer ();

  bool            Created;                                      // flags if buffer was newly
                                                                   created or already existed
  bool            IsEmpty         (void);
  float           GetFillLevel    (void);                       // return fill level

  unsigned int *  GetWriteAddress (unsigned int BlockSize);     // request a pointer to write
  void            FinishWrite     (unsigned int * NextPos);     // flag that we are done

  unsigned int    GetDataSize     (void);                       // see how much data is there
  unsigned int *  GetReadAddress  (unsigned int * Size);        // get data address (and size)
  void            FinishRead      (unsigned int * NextPos);     // release data until NextPos

  void            Clear           (void);                       // delete all data
  ...
}
```

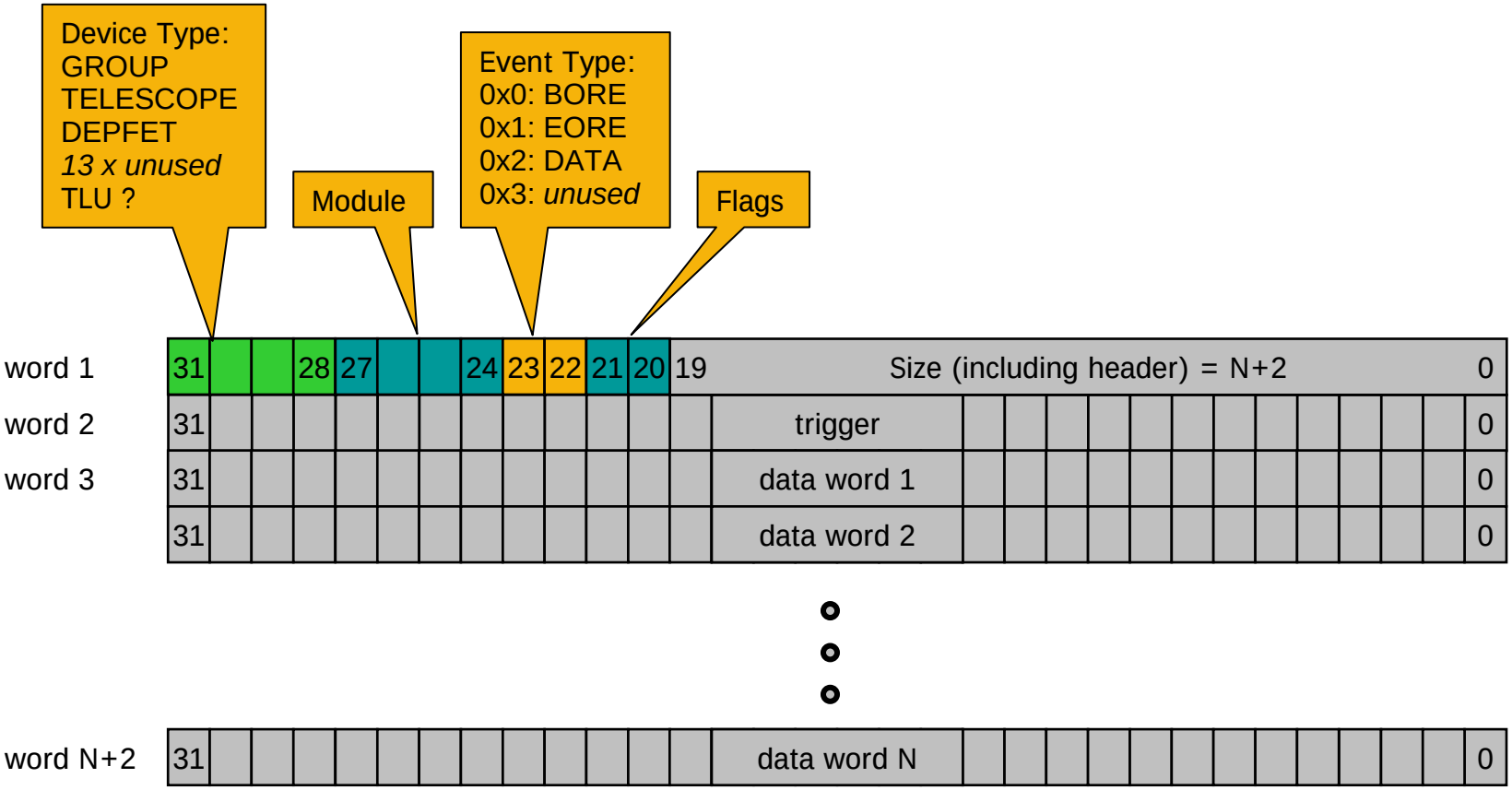
data structure in the shared buffers

- one word = unsigned integer (32 bit)

- data is organized in events with a **2 word header**:

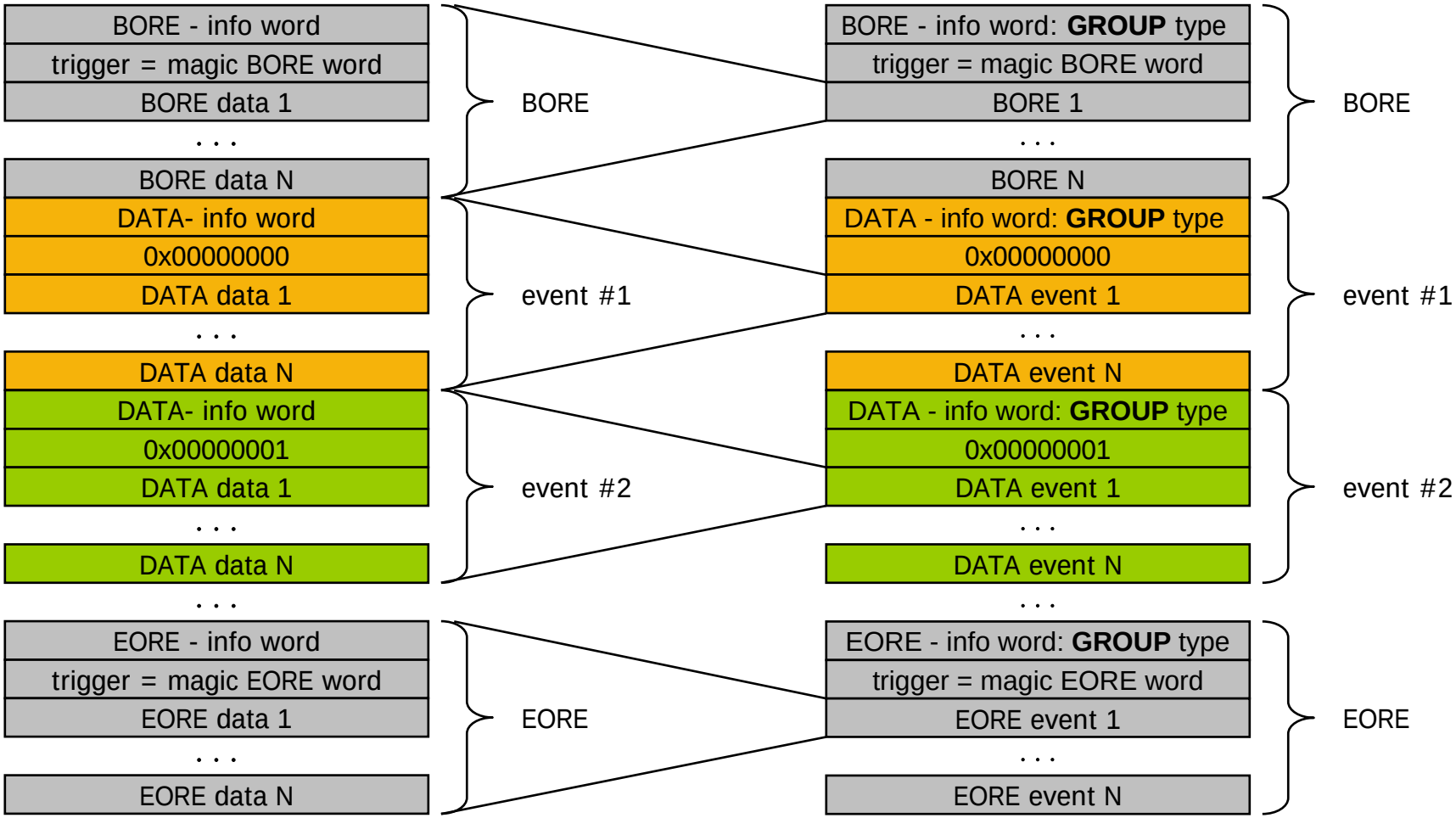
- word 0:	event **type**	begin of run	BORE
		end of run	EORE
		data	DATA
	event **size**	(including header)	size
	event **content**	group data	GROUP
		detector data	device / module / flags
- word 1:	**trigger** number	start with 0, magic numbers for BORE/EORE	
- word 2…size:	size-2 data words		

present bit allocation



packets in the buffers

DAQ buffer (single events):



BORE and EORE events

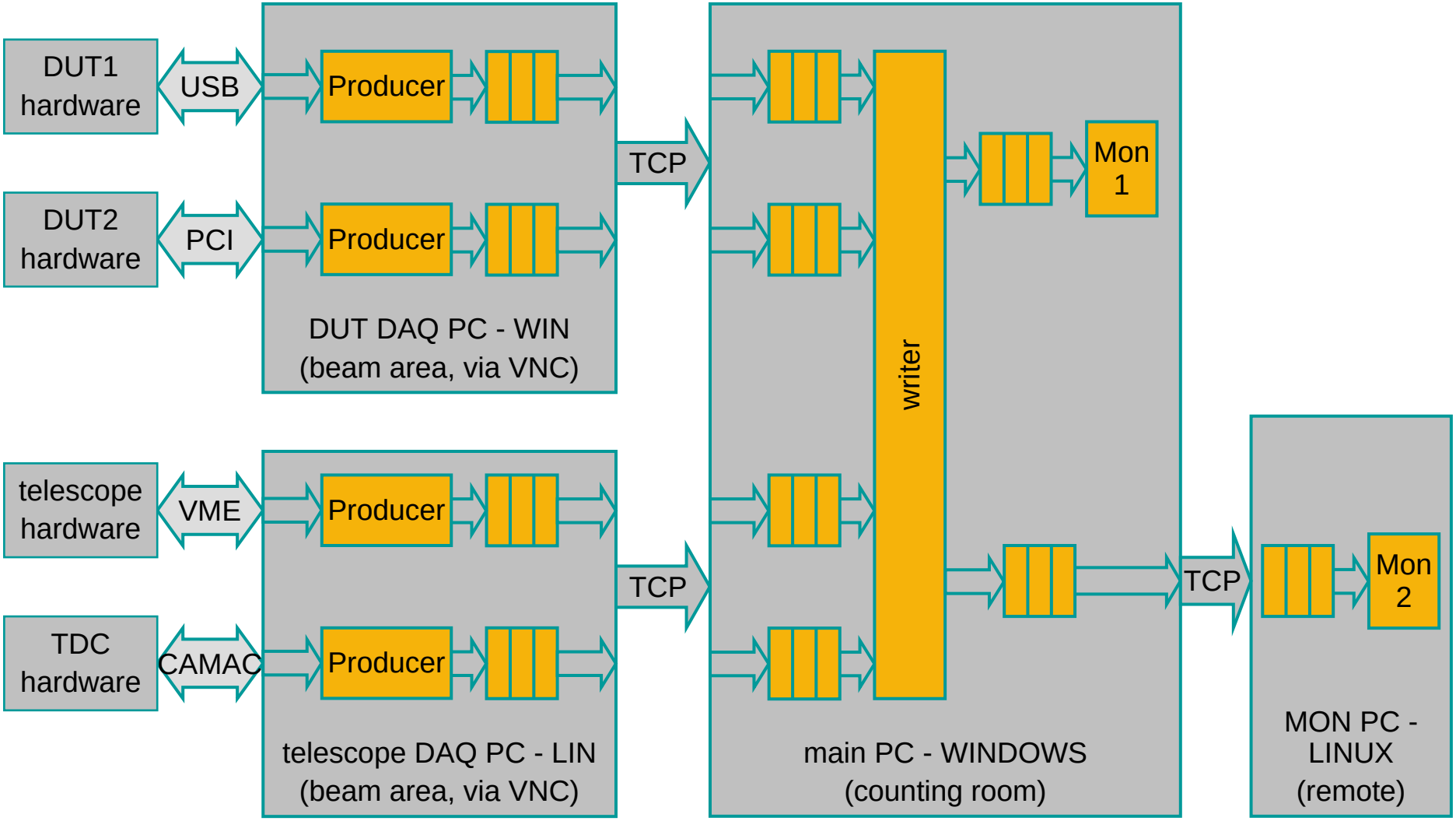
- BORE (Begin Of Run Event):
  - time, date, …
  - detector configuration data (bias values, thresholds, pedestals, …)

- EORE (End Of Run Event):
  - error flags
  - statistics for consistency checks (total number of data words,…)
  - …

GROUP events

- GROUP events are used to bundle data for ONE TRIGGER from several sources.
- They are written to file and sent to the monitoring tasks.
- data events within GROUP events are **identical** to full single events (i.e. headers are not stripped)
  - this gives a small overhead, as trigger ID is repeated
  - but it makes copying and analysis much simpler

example of possible ‘advanced’ data flow



Demo

1. Buffer Monitor + file writer
   - fill levels, raw data format
2. Add file writer
   - Observe emptying of buffers and BORE, EORE mechanism
3. Add monitoring task
   - Modify Producer parameters
   - restart a new run
   - two monitoring tasks

4. Have a look at source code (?)

Deliverables (to Daniel, David, Gilles,… ?)

- shared buffer class
- event class, derived DEPFET event class

- Applications written in C++ under Borland CBuilder (Windows):
  - BufferMonitor
  - DEPFET_DummyProducer
  - FileWriter
  - DEPFET_OMO
  - TEST_OMO

- A fair amount of comments in the code…

Summary

- Pipelined operation
- Small tasks keep units simple
- Clearly defined interface at buffer ends
- Extensions are simple

- flexible choice of operating system
- Several small monitoring tasks can run in parallel

- Later:
  - Preprocessed data (tracks) can be included into the data flow and be used by monitoring tasks. This requires some modifications, but sounds like a very interesting option!
  - Send buffers through TCP/IP (work in progress) to relief main DAQ computer

- Still needed:
  - Overall control task (flush all buffers, start/stop run,…)
  - Sending stuff over TCP/IP
  - Better control over detector configuration
  - Logfile
  - …