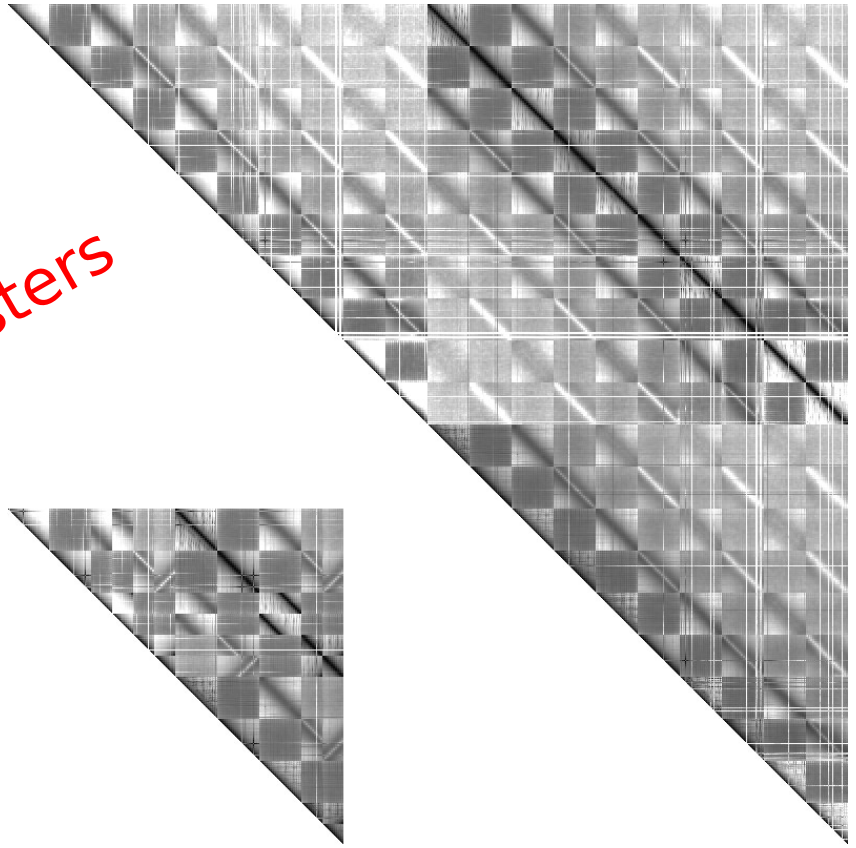
# UCESB

## unpack & check every single bit

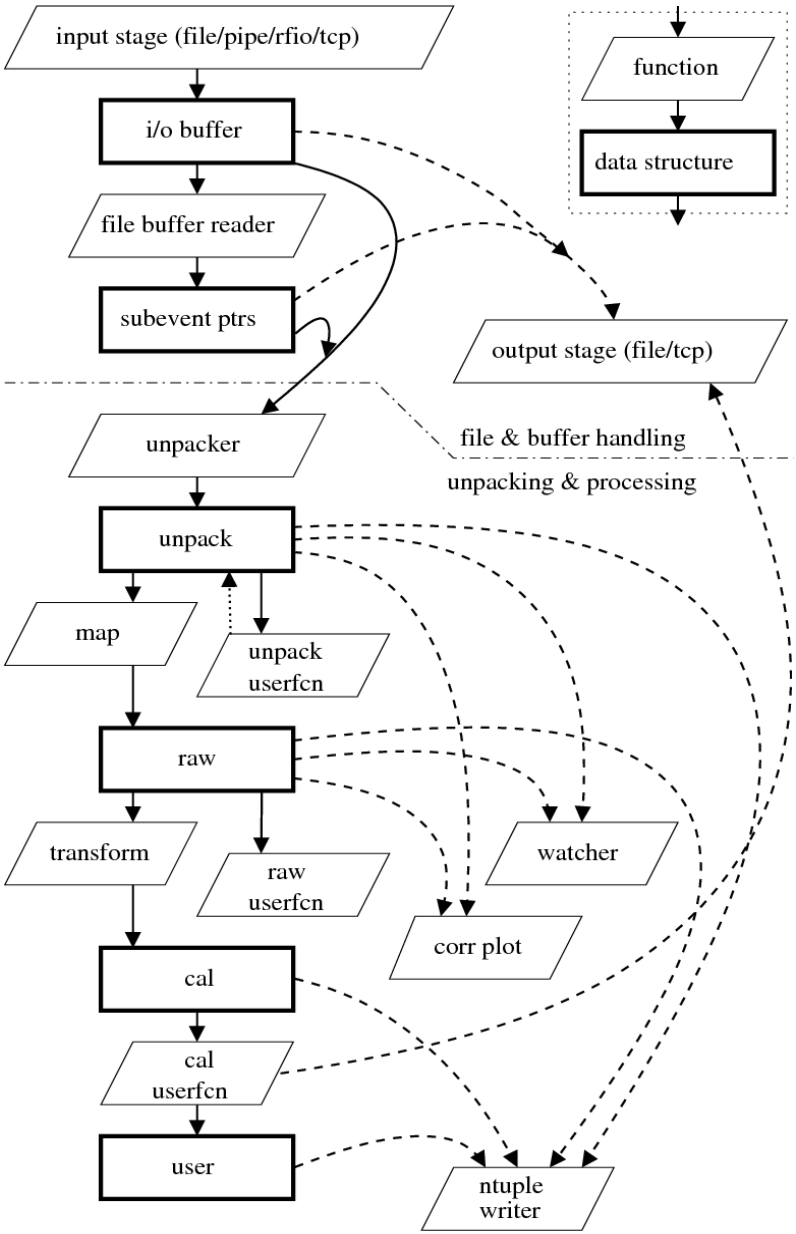
Unpacking for DAQsters



Håkan T. Johansson, Chalmers, Göteborg

GSI, Darmstadt, August 2009

# Primary scope of program




'Quick-n-dirty' generic unpacking and data 'quality' monitor

Unpack code generation from C structure-like specification:

```
SUPER_TDC(slot)
{
  UINT32 value;
}
SUBEVENT(ONE_CRATE)
{
  tdc1 = SUPER_TDC(slot=5);
  tdc2 = SUPER_TDC(slot=6);
}
EVENT
{
  crate1 = ONE_CRATE(type=5);
}
```

http://fy.chalmers.se/~f96hajo/ucesb/

# Command line driven

Compile:

```
make land
make land USE_EXT_WRITER=1 USE_CURSES=1
```

Run:

```
land/land /misc/scratch.land1/s245/lmd/r06_0308.lmd
```

Append options to manipulate the data:

```
--ntuple=RAW,r06_0308.ntu or --ntuple=RAW,r06_0308.root
```

More manipulations:

```
--watcher=POS1-2_1-4T:POS1-2_1-4E,N1_1-5_1-2T
```

List available options:

```
land/land --help
```

# Command line – Input options

```
 file://SRC          Read from file SRC.
 rfio://HOST:SRC     Read from rfio file SRC from HOST.
 event://HOST        Read from event server HOST.
 stream://HOST       Read from stream server HOST.
 trans://HOST        Read from transport HOST.
 --in-tuple=LVL,DET,FILE  Read data from ROOT-file/STRUCT. (new)
 --scramble          Toggle scrambling of data.
(--merge)            No support for overlapping sources compiled in.
```

Normal files can also be given just by name or `--file=`
For pipelines, `-` is treated as STDIN

RFIO will check and enforce that the files are staged

Read data on-line from MBS (or proxy) network servers

Insert data from ROOT-files, or (any) external program

(un)scrambling of 16 and 8-bit entries (usually NOT needed)

Sort the events from several files from multi-EB mode.

# Command line – Output options

```
--output=OPT,FILE     Save events in LMD file (native/net/...,size=nM).
--bad-events=FILE     Save events with unpack errors in LMD file.
--server=OPT          Data server (stream:port,trans:port,size=nM,hold).
--ntuple=LVL,DET,FILE  Dump data as PAW/ROOT ntuple.
```

Events can be written to new files
Creation of new sub-events
Events and sub-events can be selectively included
For pipelines, - is treated as STDOUT

Store events with unpack errors
(useful for DAQ diagnostics with on-line input)

Act as on-line data servers
(as proxy or from file / any input for dry-run tests)

Write data to HBOOK ntuples, ROOT trees,
or (any) external program

# Command line – Input Diagnostics

```
--print-buffer
--print
--data
--dump=LVL
--event-sizes
```

```
Print buffer headers.
Print event headers.
Print event data.
Text dump of data from data structures.
Show average sizes of events and subevents.
```

Print file & buffer headers:

```
Buffer         0, Size   16384 Used       0 Thu 1970-01-01 00:00:00.000 UTC
      Events   0 Type/Subtype 2000    1 FragEnd=0 FragBegin=0 LastSz       0
File header:
  Label    R06
  File     R06_0308.LMD
  User     land
  Time     27-Sep-01 09:13:59
  Run
  Exp
Buffer    1082774, Size   16384 Used   10032 Thu 2001-09-27 07:13:46.905 UTC
      Events   3 Type/Subtype   10    1 FragEnd=0 FragBegin=0 LastSz    1444
Buffer    1082775, Size   16384 Used   15868 Thu 2001-09-27 07:13:47.023 UTC
      Events   4 Type/Subtype   10    1 FragEnd=0 FragBegin=0 LastSz    1456
```

Print event & sub-event headers:

```
Event     7938336 Type/Subtype   10     1 Size    1444 Trigger  1
  SubEv ID      1 Type/Subtype   34 3100 Size     384 Ctrl   9 Subcrate   0
  SubEv ID      1 Type/Subtype   34 3200 Size     216 Ctrl   9 Subcrate   0
  SubEv ID      1 Type/Subtype   32 3100 Size     800 Ctrl   0 Subcrate   0
Event     7938338 Type/Subtype   10     1 Size     412 Trigger  7
  SubEv ID      1 Type/Subtype   34 3100 Size     392 Ctrl   9 Subcrate   0
```

Print unpacked & mapped data...
(not shown)

0xHEX-dump the raw data:

```
Event   270987188 Type/Subtype   10    1 Size     404 Trigger  2
  SubEv ID      1 Type/Subtype   34 3200 Size       8 Ctrl   9 Subcrate   2
 02000000 00004321
  SubEv ID      2 Type/Subtype   32 3130 Size      96 Ctrl   9 Subcrate   1
 28190af1 281c0a73 50910145 5096013e 5099016e b8b00157 68c00103 68c1014c
 68c2014a 68c3016a 68c40101 68c50142 68c60102 68c70129 68c80129 68c90144
 68ca0129 68cb0126 68cc013c 68cd0162 68ce011b 68cf0164 40b10240 40b4017a
...
```

```
 type/stype    id crt ctrl    min    max  avg(ev) avg(tot) head  occurances

trig  1:                  (     0   7232)          4236.6 100.6 (      7885)
   34/ 3100    10:  2:  1 (   384    384)   384.0    19.0   0.6 (       390)
   32/ 3130     8:  1:  2 (     4   1516)   180.5   180.5  12.0 (      7885)
   32/ 3130     8:  2:  2 (     8   1368)    19.8    19.8  12.0 (      7885)
   34/ 3200    10:  2:  1 (   124    124)   124.0   124.0  12.0 (      7885)
   34/ 3500    10:  2:  1 (     4     64)     8.0     8.0  12.0 (      7885)
   82/ 8200    10:  0:  3 (  3320   3960)  3479.0  3479.0  12.0 (      7885)
...
trig  2:                  (     0   6376)          3748.1  88.6 (      3549)
   34/ 3100    10:  2:  1 (   384    384)   384.0    19.2   0.6 (       177)
   32/ 3130     8:  1:  2 (     0   1048)    41.8    41.8  12.0 (      3549)
```

Generate summary of event & sub-event sizes vs. trigger type

# Command line – Error handling

```
--allow-errors      Allow errors.
--broken-files      Allow errors again after bad files.
--debug             Print events causing errors.
--io-error-fatal    Any I/O error is fatal.
```

UCESB unpackers normally abort after 10 errors

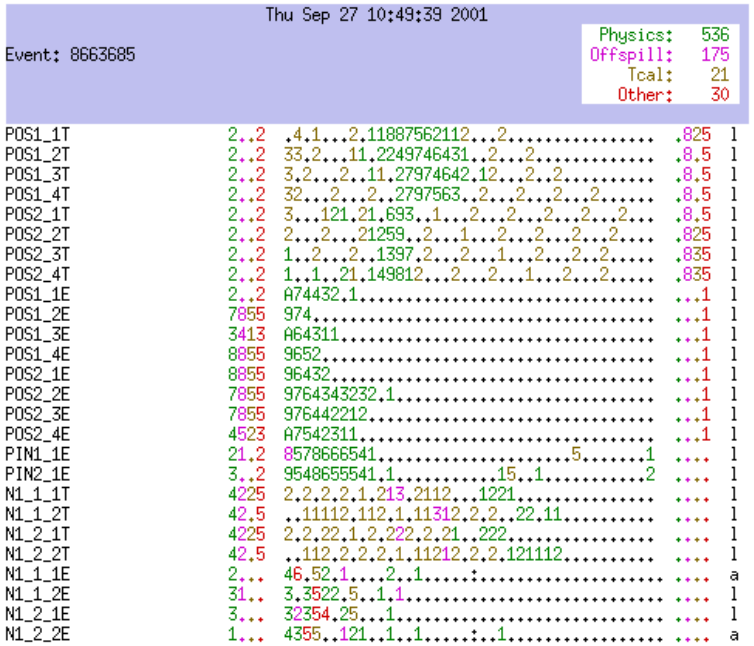
Processing can continue after broken files

0xHEX-dump the raw data for broken events

# Command line – Processing & Monitoring

```
--watcher=DET
--corr=TRIG,DET,FILE
```

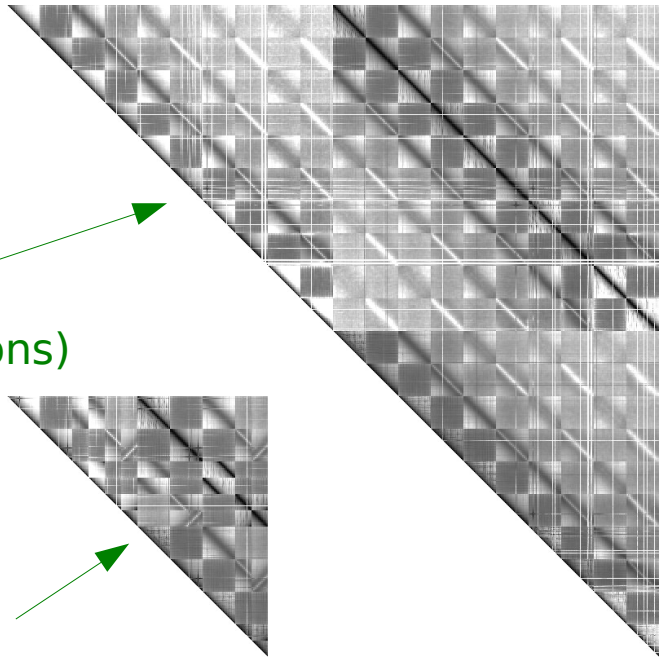
Do ncurses-based data viewing.
Create 2D correlation plot.




## DAQ-scope:

On/off-line 'movie' view of data
Shows usage of each channel's range

Usually 1 spill per 'frame'



LAND
(cosmic muons)

Correlations between data in neighbouring channels checks mapping

artificial mismaps

# Command line – Miscellaneous

```
--show-members
--calib=FILE
--max-events=N
--help
```

```
Show members of all data structures.
Extra input file with mapping/calibration parameters.
Limit number of events processed.
Print this usage information and quit.
```

Print a list of all available variables:

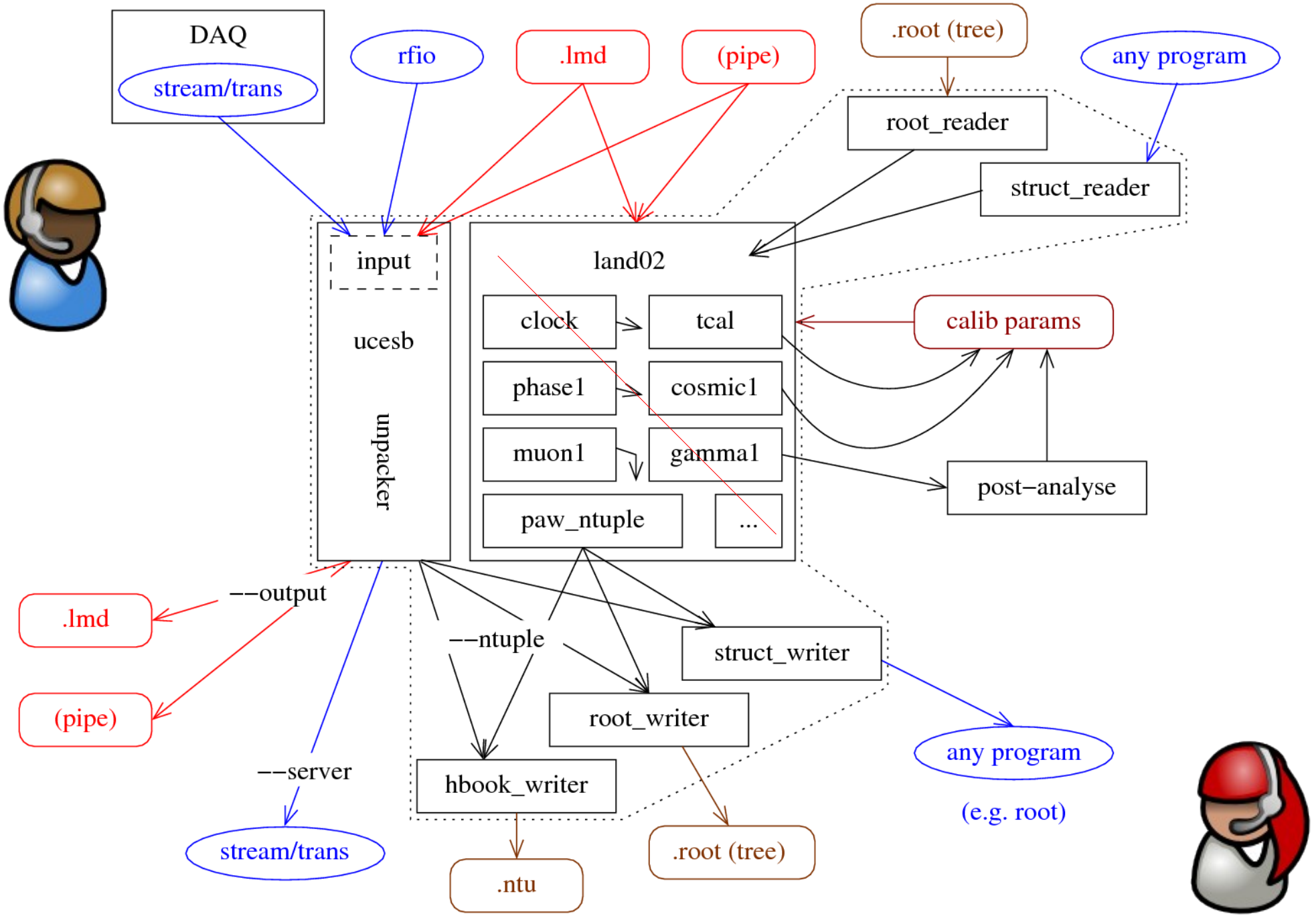
```
UNPACK_vme_tdc3data32           .UNPACK.vme.tdc[3].data[z:32]
UNPACK_vme_tdc3eob_u32          .UNPACK.vme.tdc[3].eob.u32
UNPACK_vme_scaler0_data32       .UNPACK.vme.scaler0.data[z:32]
UNPACK_vme_scaler0_header_u32   .UNPACK.vme.scaler0.header.u32
UNPACK_vme_adc5data32           .UNPACK.vme.adc[5].data[z:32]
UNPACK_vme_adc5eob_u32          .UNPACK.vme.adc[5].eob.u32
UNPACK_vme_header_failure_u32   .UNPACK.vme.header.failure.u32
UNPACK_vme_header_time_stamp    .UNPACK.vme.header.time_stamp
UNPACK_vme_header_multi_events  .UNPACK.vme.header.multi_events
...
RAW_BACK2E                      .RAW.BACK[2].E
RAW_MONE_E                      .RAW.MONE.E
RAW_MONDE_E                     .RAW.MONDE.E
RAW_MONTGT_E                    .RAW.MONTGT.E
RAW_DSSSD2F32E                  .RAW.DSSSD[2].F[z:32].E
RAW_DSSSD2F32T                  .RAW.DSSSD[2].F[z:32].T
RAW_DSSSD2B32E                  .RAW.DSSSD[2].B[z:32].E
RAW_DSSSD2B32T                  .RAW.DSSSD[2].B[z:32].T
RAW_DSSSD2FT                    .RAW.DSSSD[2].FT
RAW_DSSSD2BT                    .RAW.DSSSD[2].BT
...
```

Simple linear calibrations at RAW → CAL conversion

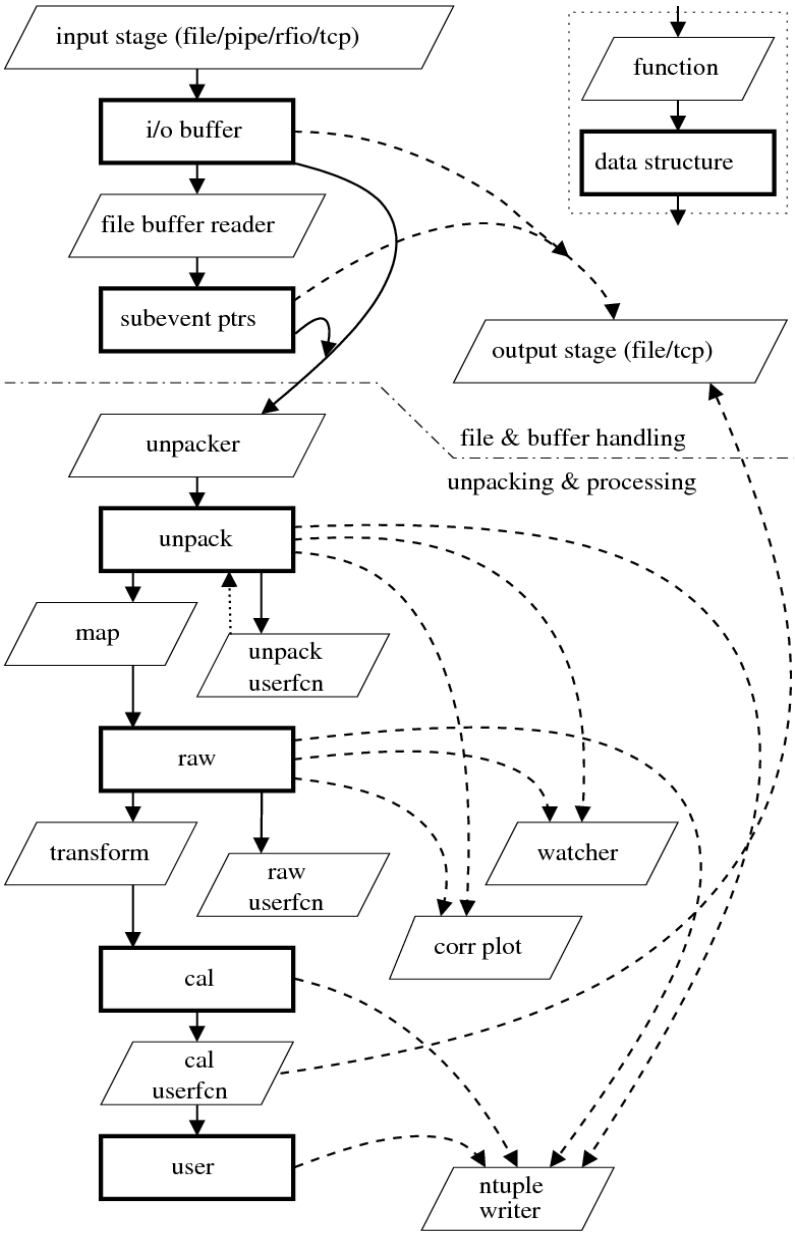
Stop processing after N events.

Too many options?

# ucesb/(land02) interaction



# UCESB – unpack & check every single bit




'Quick-n-dirty' generic unpacking and data 'quality' monitor

Unpack code generation from C structure-like specification:

```
SUPER_TDC(slot)
{
  UINT32 value;
}
SUBEVENT(ONE_CRATE)
{
  tdc1 = SUPER_TDC(slot=5);
  tdc2 = SUPER_TDC(slot=6);
}
EVENT
{
  crate1 = ONE_CRATE(type=5);
}
```

http://fy.chalmers.se/~f96hajo/ucesb/

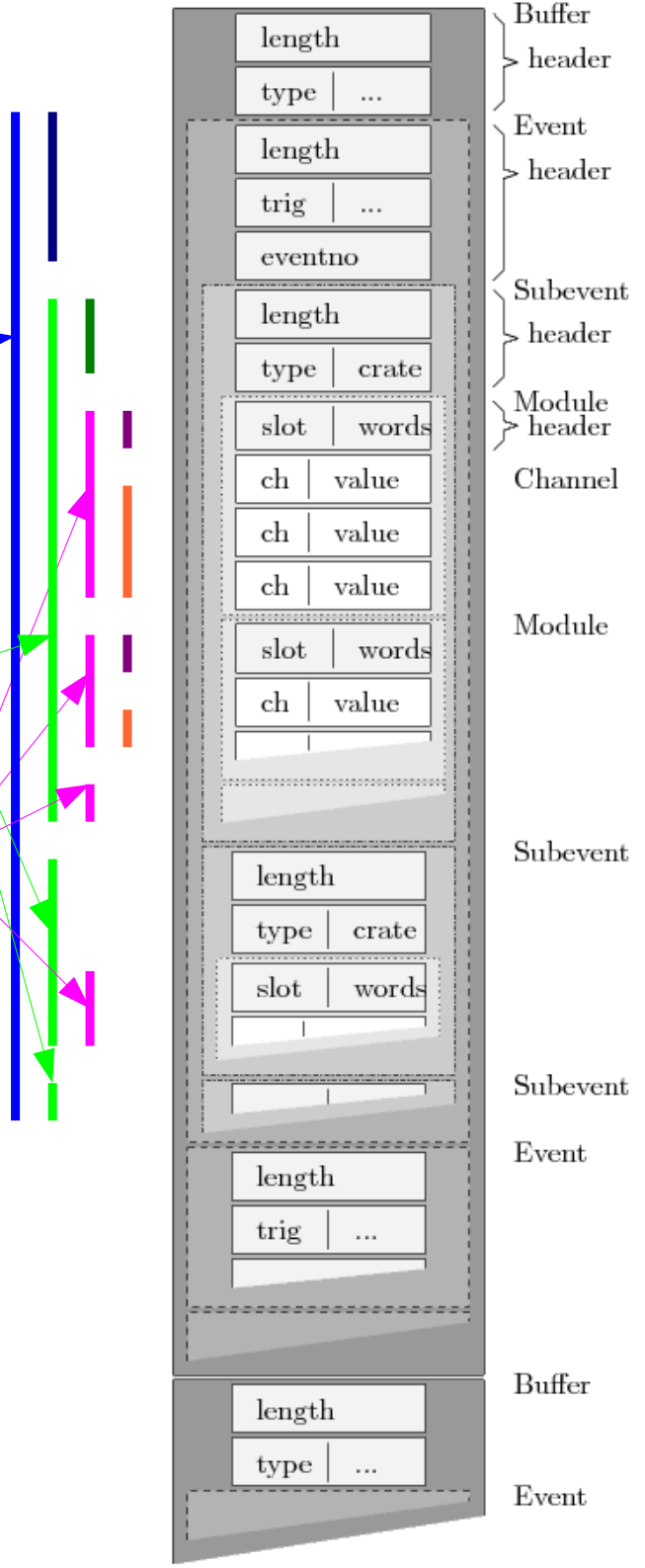
# Anatomy of raw data files (.lmd)

Each file has many events:
a header, and...

Each event has one or more sub-events:
a header, and...

Each sub-event has data from acquisition modules (TDCs, QDCs, ADCs, scalers...):
often a header per module, and...

Each data word is a value, sometimes with indexing information



# Module .spec structure

```
VME_CAEN_V775(geom,crate)
{
  MEMBER(DATA12_OVERFLOW data[32] ZERO_SUPPRESS);

  UINT32 header NOENCODE {
      8_13:  count;
      16_23: crate = MATCH(crate);
      24_26: 0b010;
      27_31: geom = MATCH(geom);
  }
  list(0<=index<header.count) {
      UINT32 ch_data NOENCODE {
          0_11:  value;

          12:    overflow;
          13:    underflow;
          14:    valid;

          16_20: channel;

          24_26: 0b000;
          27_31: geom = CHECK(geom);

          ENCODE(data[channel],(value=value,overflow=overflow));
      }
  }
  UINT32 eob {
      0_23:  event_number;
      24_26: 0b100;
      27_31: geom = CHECK(geom);
  }
}
```

## CAEN V775 (TDC) data format:

**Header**

| 31 - 27 | 26-24 | 23 - 16 | | 13 - 8 | |
|---|---|---|---|---|---|
| geom | 0 1 0 | crate | | count | |

**Data word**

| 31 - 27 | 26-24 | | 20 - 16 | | 14 | 13 | 12 | 11 - 0 |
|---|---|---|---|---|---|---|---|---|
| geom | 0 0 0 | | channel | | V | U | O | value |

**End marker**

| 31 - 27 | 26-24 | 23 - 0 |
|---|---|---|
| geom | 1 0 0 | event_number |

# .spec: item blocks, SUBEVENT, EVENT

'Object oriented' – structures in structures

```
SUPER_TDC(slot)
{
  UINT32 value;
}

SUBEVENT(ONE_CRATE)
{
  tdc1 = SUPER_TDC(slot=5);
  tdc2 = SUPER_TDC(slot=6);
}

EVENT
{
  crate1 = ONE_CRATE(type=5);
}
```

```
class SUPER_TDC
{
  uint32 value;
};
class ONE_CRATE
  : public unpack_subevent_base
{
  SUPER_TDC tdc1;
  SUPER_TDC tdc2;
};
class unpack_event
  : public unpack_event_base
{
  ONE_CRATE crate1;
};
```

# .spec: select several

When (several) blocks/items occur in arbitrary order:

```
select
select several
```

Then usually MATCH within the blocks

```
class LECROY_1885
{
  // ...
};
class FASTBUS_CRATE
  : public unpack_subevent_base
{
  LECROY_1885 adc0;
  LECROY_1885 qdc[3];
};
```

```
SUBEVENT(FASTBUS_CRATE)
{
  select several
  {
    adc0 = LECROY_1885(geom=10,channels=96);
    qdc[0] = LECROY_1885(geom=13,channels=48);
    qdc[1] = LECROY_1885(geom=17,channels=48);
    qdc[2] = LECROY_1885(geom=15,channels=48);
  }
}
```

# .spec: bitfields

```
BITFIELD()
{
  UINT16 data
  {
    0_11:  value;
    12_14: channel;
    15:    overflow;
  }
}
```

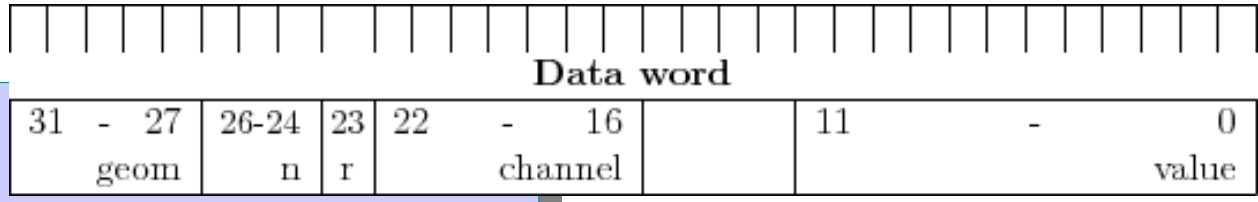
```
class BITFIELD
{
  union
  {
    struct
    {
      uint16 value:12;
      uint16 channel:3;
      uint16 overflow:1;
    };
    uint16 u16;
  } data;
};
```

Bit-packed items are specified with their locations

Generated as bit-fields
(actually twice, for big and little endian unpacker program host *machines*
- i.e. this is not the byte-swapping;
it is C bit-field declaration conventions)

Access to the full word

# .spec: zero-suppression, NOENCODE




```
LECROY_1885(geom,channels)
{
  MEMBER(DATA12_RANGE data[96] ZERO_SUPPRESS);

  UINT32 ch_data NOENCODE
    {
      0_11:  value;
      16_22: channel = RANGE(0,channels);
      23:    range;
      24_26: n = 0;
      27_31: geom = MATCH(geom);

      ENCODE(data[channel],(value=value,
                            range=range));
    }
}

SUBEVENT(FASTBUS_CRATE) {
  select several {
    adc = LECROY_1885(geom=10,
                      channels=64);
  }
}
```

```
class LECROY_1885
{
  raw_array_zero_suppress <
    DATA12_RANGE,
    DATA12_RANGE, 96 > data;
};
```

NOENCODE data words are consumed by the generated unpacker

The same module can be entered many times: revisit / or not: norevisit

usually without module header

usually with header

# .spec: lists and conditionals

```
GROUP_DATA(group)
{
  MEMBER(DATA16 data[64] NO_INDEX_LIST);

  UINT16 header NOENCODE
    {
      0_7:   group = MATCH(group);
      8_13:  item_count;
      14_15: 0b01;
    }
  list (0 <= index < header.item_count)
    {
      UINT16 value NOENCODE;

      ENCODE(data APPEND_LIST,(value=value));
    }
  if (!(header.item_count & 1))
    {
      // Padding needed to keep 32-bit alignment
      UINT16 pad NOENCODE { 0_15: 0; }
    }
}
```

Use a `list` when the number of items is known

Conditional

# Input stage, etc...

Techno-babble...:

Zero copy (unpack directly from `mmap()`ed I/O-buffers)
(only fragmented *subevents* are copied)

Byte-swapping on the fly - templated unpacker functions

Event-driven

Separate thread for network input

Transparent fork of gzip / gunzip / bzip2 / bunzip2 / lzma

Bucket-sort strategies for mapping – fixed size structures
(as default)

---

Separate thread for network output

ntuple / ROOT tree / struct writer as separate process

# File formats

Common code for low-level I/O-buffers

---

Several file packaging readers (buffers etc):

LMD – .lmd – list mode data (GSI standard)

PAX – KVI data (ESN unpacking)

EBYEDATA – (Daresbury) MIDAS

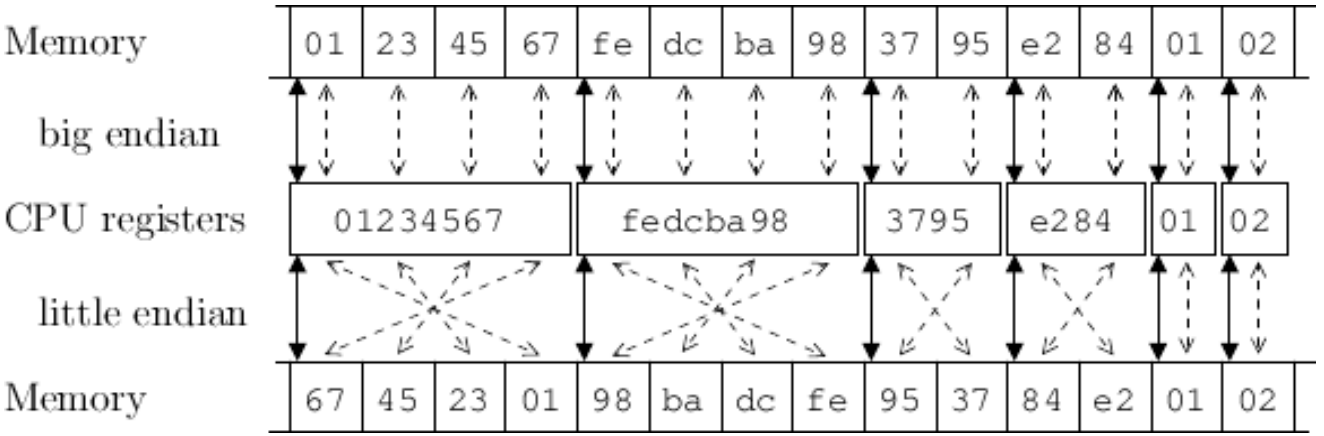
HLD – Hades files

(more - easily possible... suggestions? (i.e. have any interesting files?))

---

Event handling (including unpacking) is common code

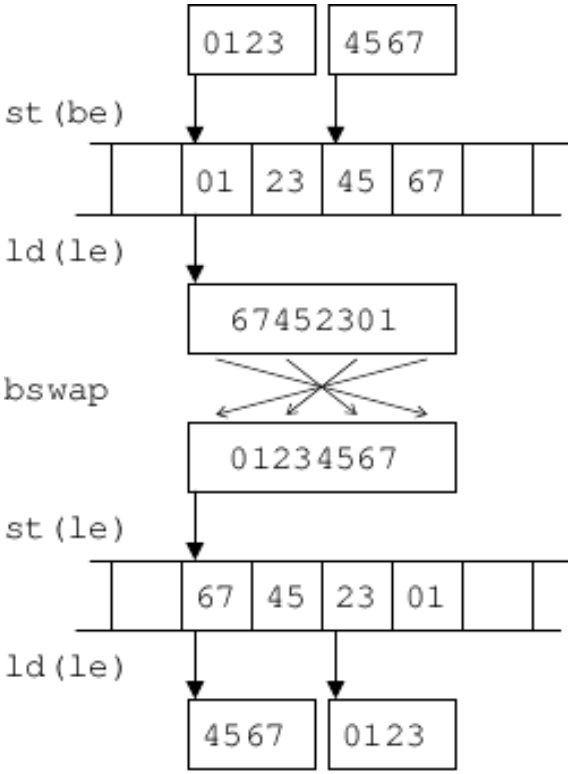
# Byte swapping - endianess & scrambling




Endianess differences and byte-swapping are unavoidable → deal with it → markers!

UCESB knows what to do!
(given marked file formats... - workaround for others)

Byte-swapping with the wrong size scrambles the data




st = store (CPU → memory)
ld = load (memory → CPU)
be = big endian
le = little endian

# Mapping modules to detectors

Each signal mapping has
a source at UNPACK level,
a destination at RAW level,
and a type (same as at UNPACK)

Corresponding C structure
is generated

```
struct raw_event_DETA
{
  DATA12 T[2];
  DATA12 E;
};
struct raw_event_DETB
{
  DATA12 T;
};
struct raw_event
  : public raw_event_base
{
  raw_event_DETA DETA[1];
  raw_event_DETB DETB[5];
};
```

```
// Declarations of single detector signals
SIGNAL(DETA_1_T2,  vme.tdc[1].data[6],  DATA12);
SIGNAL(DETA_1_E,   vme.qdc[2].data[12], DATA12);

// Create an item in the raw level without source
SIGNAL(DETB_5_T, , DATA12);
```

# Zero-suppressed mappings

```
struct raw_event_DETC_FRONT
{
  DATA12 T;
};
struct raw_event_DETC
{
  raw_array_zero_suppress < raw_event_DETC_FRONT,
                            raw_event_DETC_FRONT, 16 > FRONT;
};
struct raw_event
  : public raw_event_base
{
  raw_event_DETC DETC[2];
};
```

Templated encapsulation-classes implement zero-suppression

```
// A list of mappings/items
SIGNAL(DETC_2_FRONT_1_T,  vme.tdc[2].data[0],
       DETC_2_FRONT_16_T, vme.tdc[2].data[15], DATA12);

// Make an array zero-suppressed
SIGNAL(ZERO_SUPPRESS: DETC_1_FRONT_1);
```

# CAL level and units

```
struct raw_event_DETD
{
  DATA12 E UNIT ("ch");
};
struct raw_event
  : public raw_event_base
{
  raw_event_DETD DETD[1];
  DATA32 SCALER[1];
};
```

Optional units for the conversions

```
struct cal_event_DETD
{
  float E UNIT ("#MeV");
};
struct cal_event
  : public cal_event_base
{
  cal_event_DETD DETD[1];
};
```

CAL level has same layout as RAW level

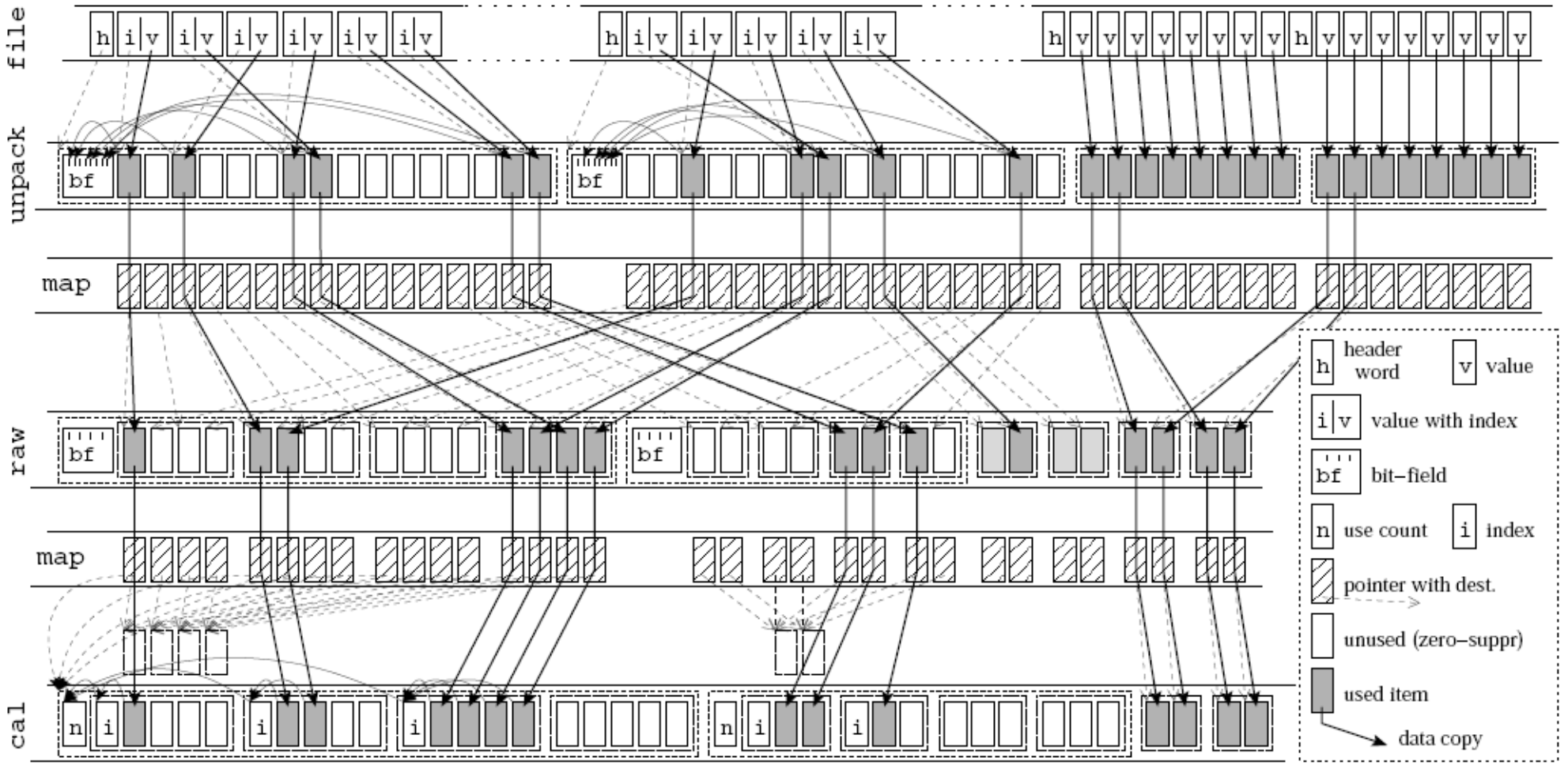
```
// An item with cal level entry, and units
SIGNAL(DETD_1_E, vme.qdc[3].data[0], (DATA12 "ch",float "#MeV"));

// An item which is only mapped for the last physical event in a
// multi-event unpacker
SIGNAL(LAST_EVENT: SCALER_1, vme.scaler.data[0], DATA32);
```

# Data structures

“Show me your code and conceal your data structures, and I shall continue to be mystified. Show me your data structures, and I won't usually need your code; it'll be obvious.”

Eric Raymond



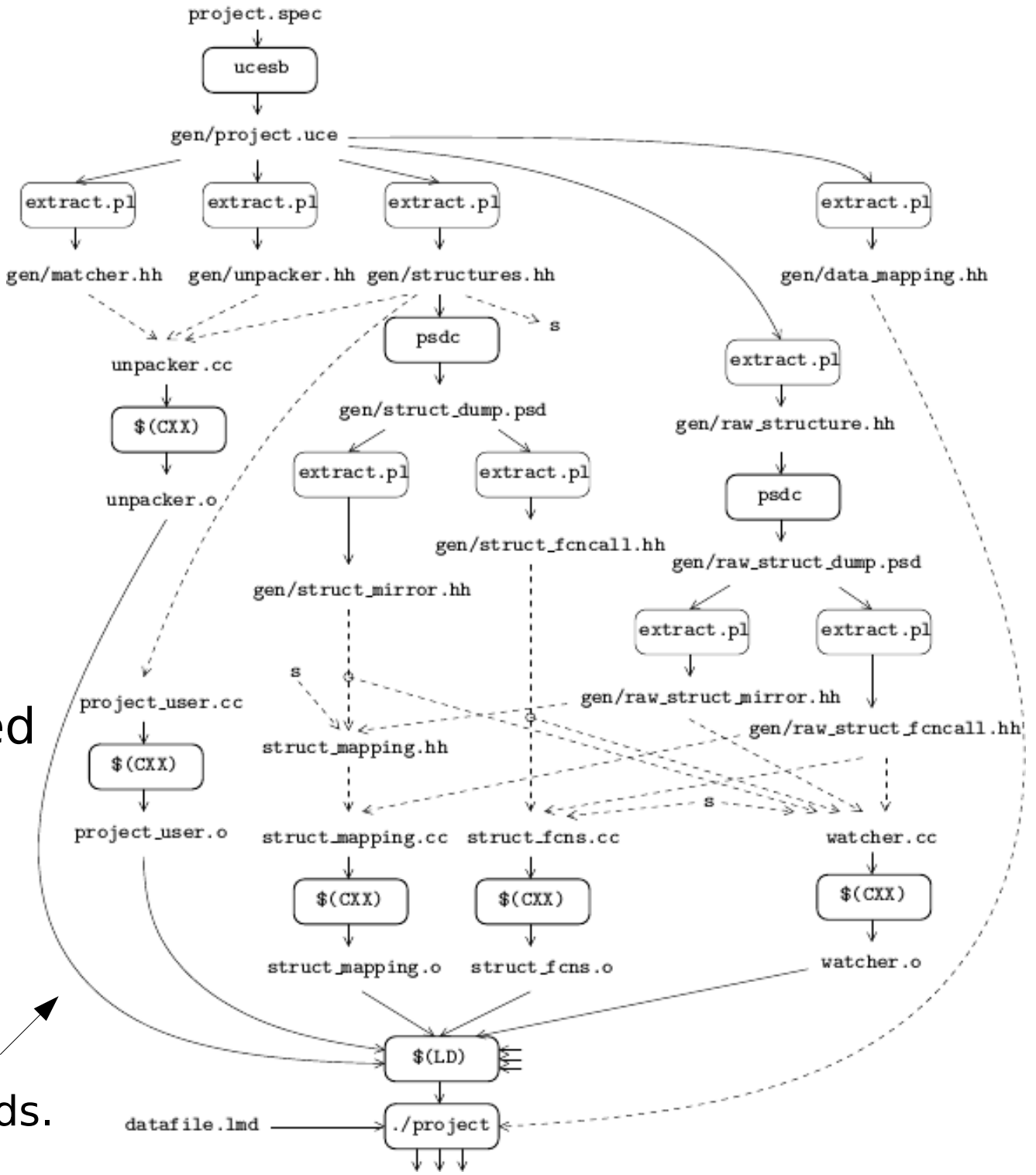
# Code generation & compilation

Use the compiler - that's what it is for!

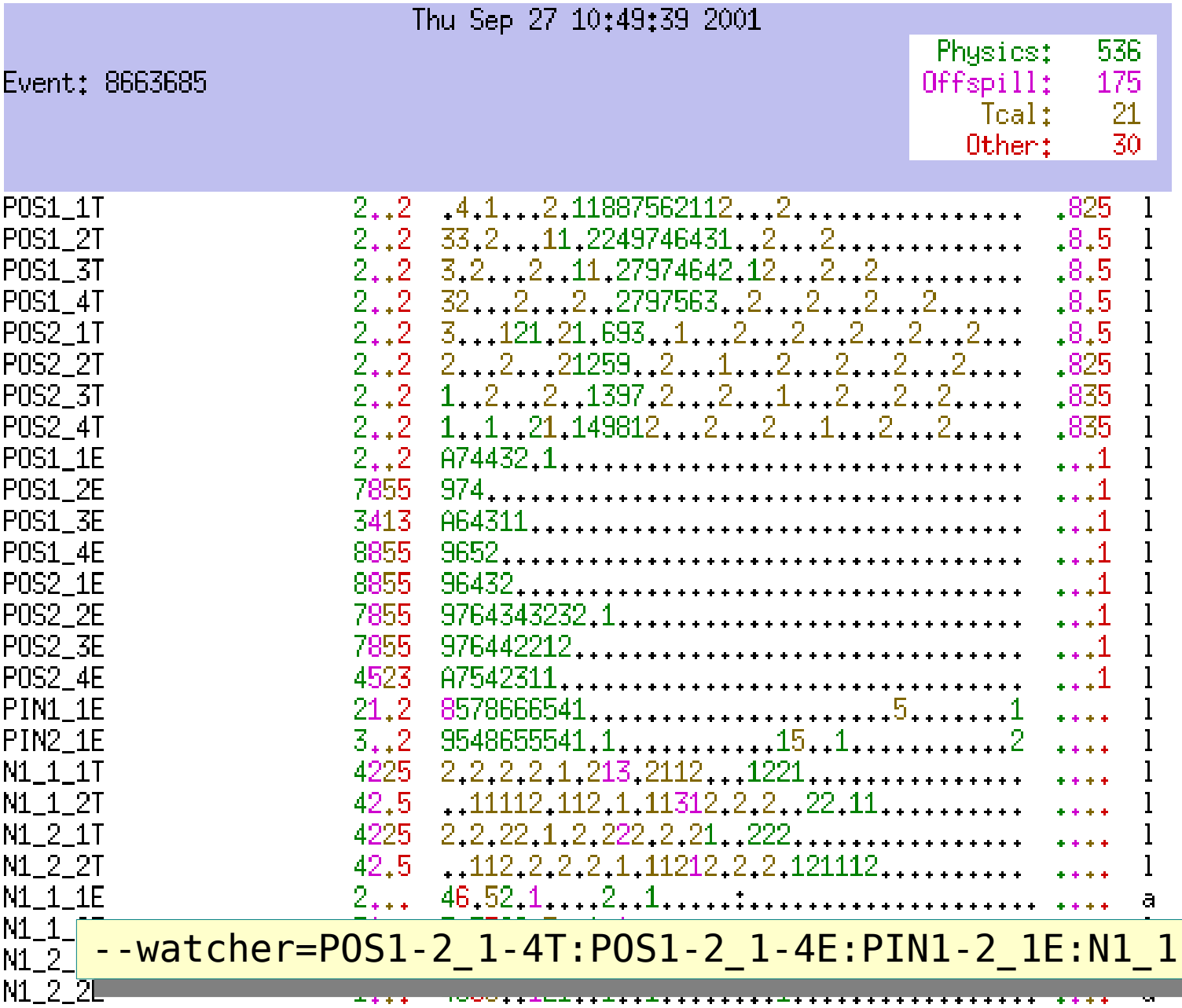
Build process handled by a Makefile

Just type: `make`

The compilation **gui**tar - the *only* 'GUI' ucesb needs.



# Watcher – the DAQscope




Each line is a histogram for one raw channel

Values are $\log_2$ of bin content

Stored zeros and overflow

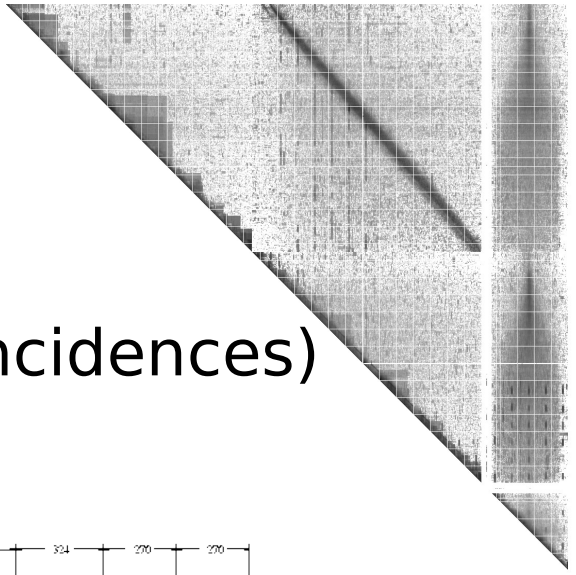
Colour by most contributing trigger type

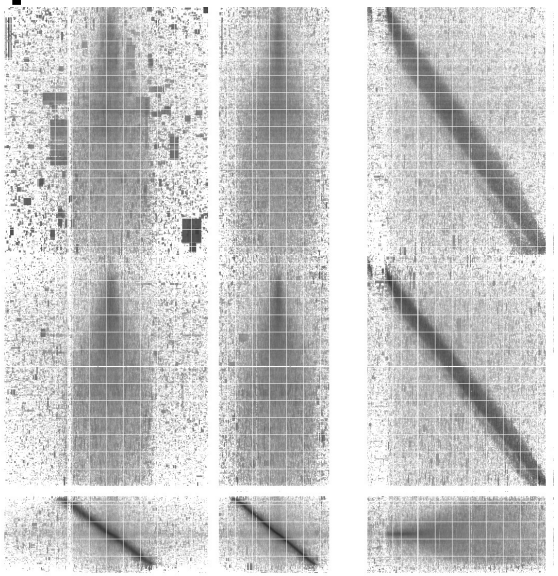
Spill synchronized

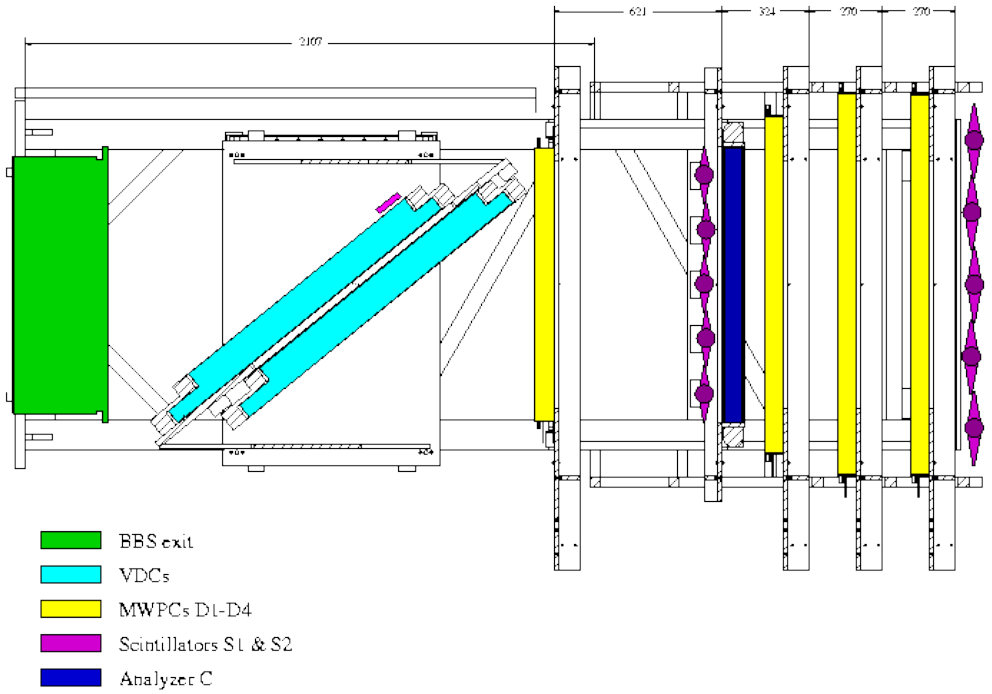
# Correlation plots – EuroSuperNova (ESN) @KVI MWPC maps

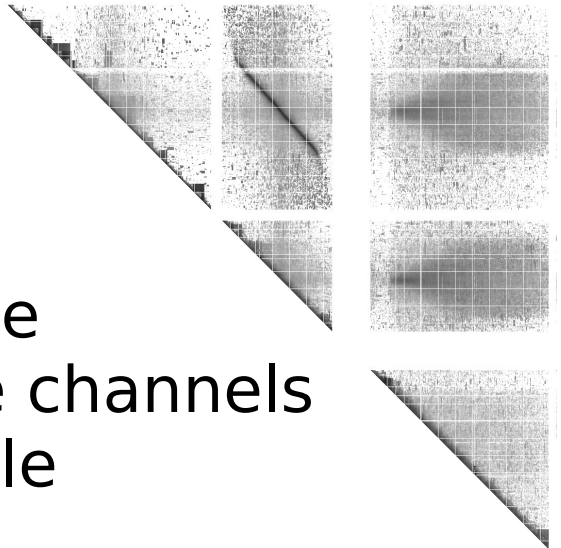
Every row/column = 1 channel

Intensity = $\log_2$(#coincidences)





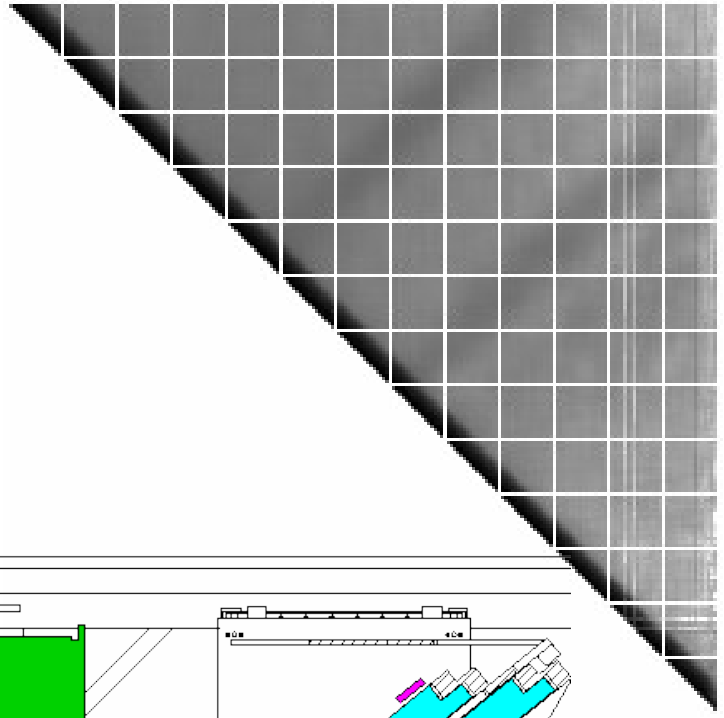


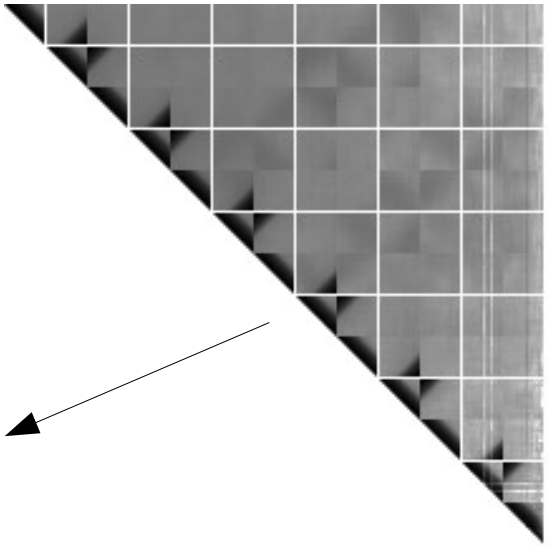



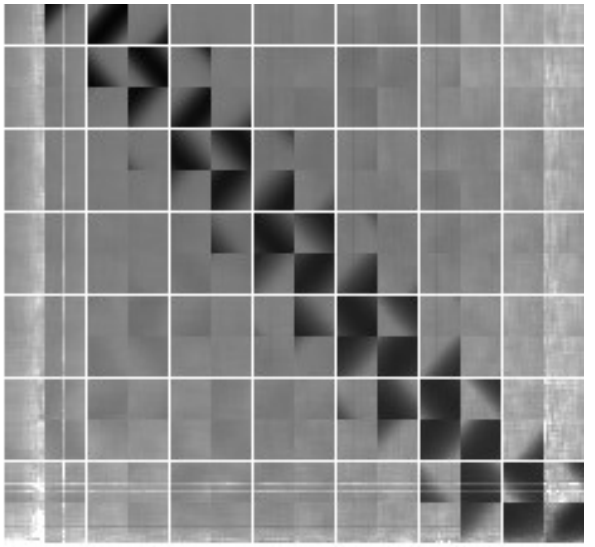
Plot shows the ~2500 active channels of 32k possible

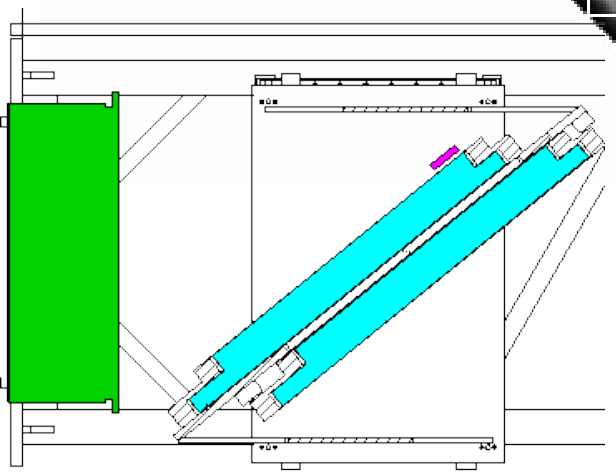
http://www.kvi.nl/~snovadoc/

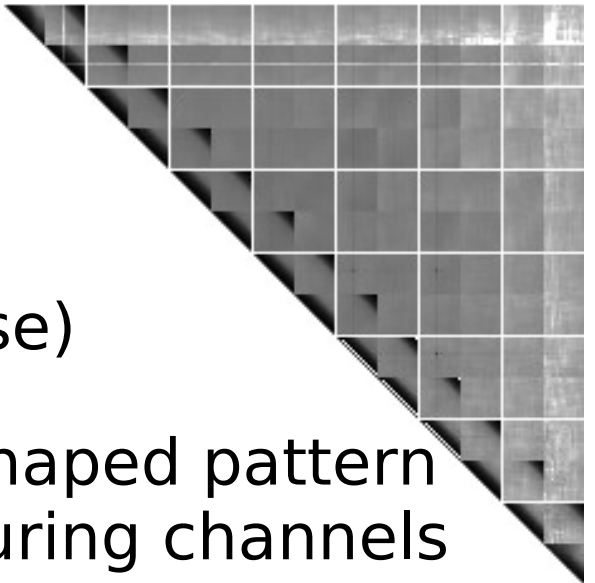
# Correlation plots – ESN VDC maps









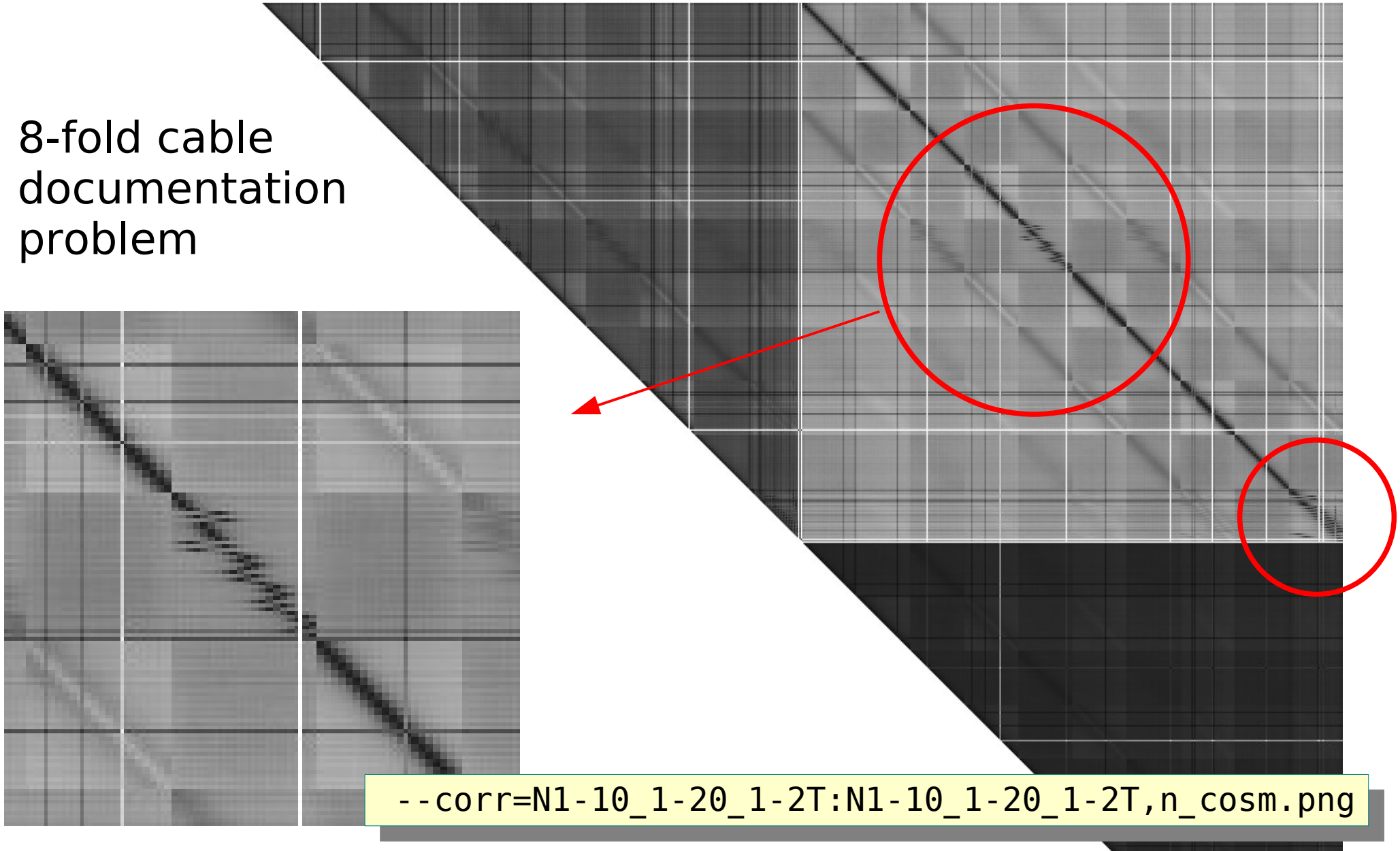



Every second 16-fold
cable inverted (on purpose)

Every ion track gives V-shaped pattern
(in time) in ~10 neighbouring channels

# Quickly finding LAND cable mismap

8-fold cable documentation problem



```
--corr=N1-10_1-20_1-2T:N1-10_1-20_1-2T,n_cosm.png
```

# ntuple & ROOT tree generation

Select which data-levels to include.

Optionally limit which detectors / channels are included (also with indices).

Output file type selected by extension.

```
--ntuple=UNPACK,file.ntu

--ntuple=RAW,file.ntu

--ntuple=UNPACK,RAW,file.root

--ntuple=RAW,POS2,N,TFW,TOF,file.root

--ntuple=UNPACK,fastbus,camac,file.root
```

# Pipelining

Left | as | exercise | to | the > reader

# 'Any' program processing

```
#include "ext_data_client.h"
#include "ext_h101.h"
#include <stdlib.h>
#include <stdio.h>

int main(int argc,char *argv[])
{
  struct ext_data_client *client;

  EXT_STR_h101 event;
  EXT_STR_h101_layout event_layout = EXT_STR_h101_LAYOUT_INIT;

  if (argc < 2)
    {
      fprintf (stderr,"No server name given, usage: %s SERVER\n",argv[0]);
      exit(1);
    }

  client = ext_data_connect_stderr(argv[1]);

  if (client == NULL)
    exit(1);

  if (ext_data_setup_stderr(client,
                  &event_layout,sizeof(event_layout),
                  sizeof(event)))
    {
      for ( ; ; )
        {
          if (!ext_data_fetch_event_stderr(client,&event,sizeof(event)))
            break;

          /* Do whatever is wanted with the data. */

          printf ("%10d: %2d\n",event.EVENTNO,event.TRIGGER);
        }
    }
  ext_data_close_stderr(client);
  return 0;
}
```

Using the same structures as generated (virtually) for ntuples and ROOT trees

# 'Any' program processing

```
empty/empty /dev/null \
  --ntuple=UNPACK,STRUCT_HH,ext_h101.h
```

```
typedef struct EXT_STR_h101_t
{
  // UNPACK
  uint32_t TRIGGER;
  int32_t  EVENTNO;

} EXT_STR_h101;
```

Also as network server (any # clients)

```
cc -g -O3 -o ext_reader_h101_stderr -I. ext_data_reader_stderr.c hbook/ext_data_client.o
```

```
empty/empty INFILE.lmd --ntuple=UNPACK,STRUCT,- | \
  ./ext_reader_h101_stderr -
```

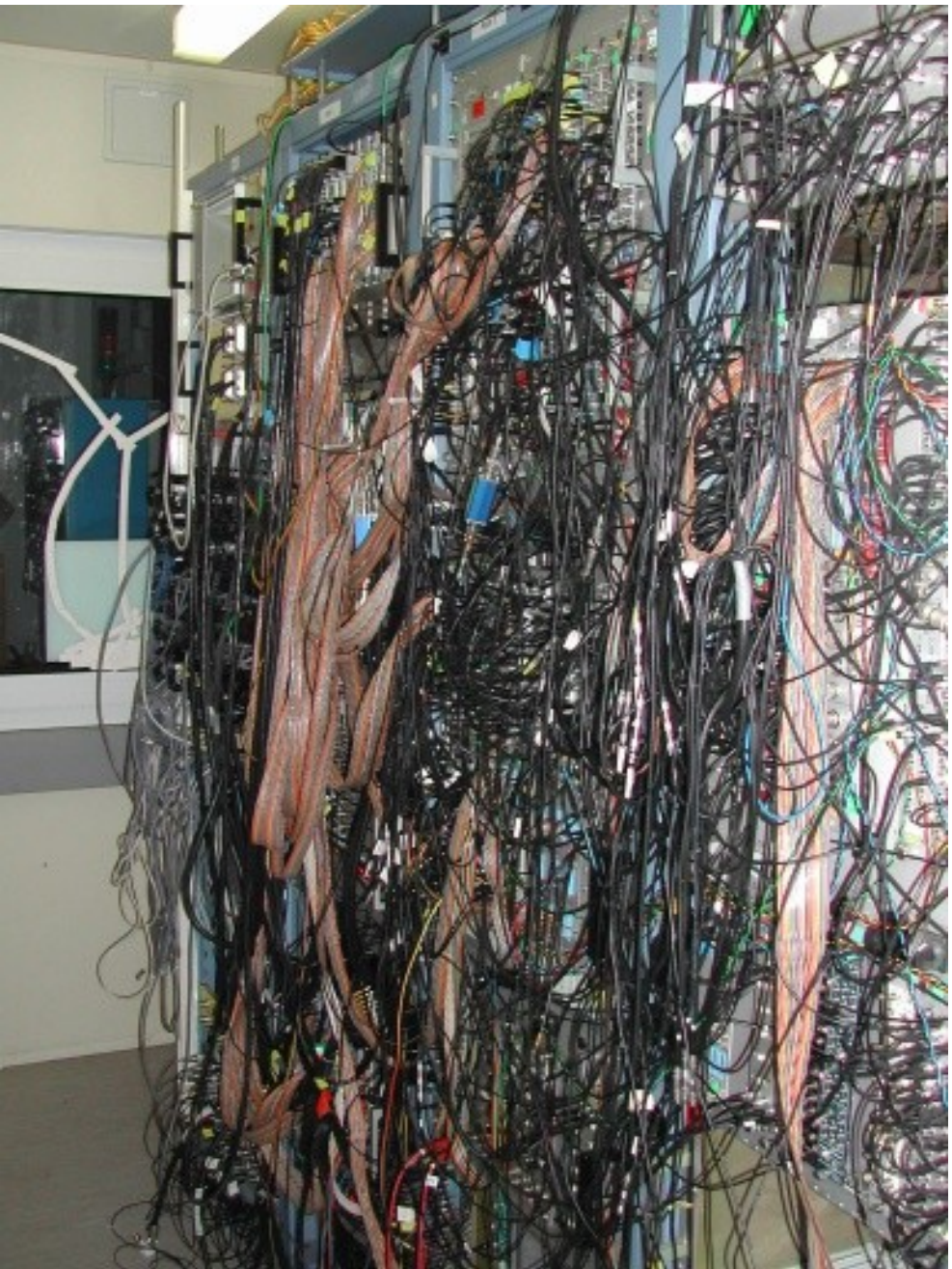
# Finale!

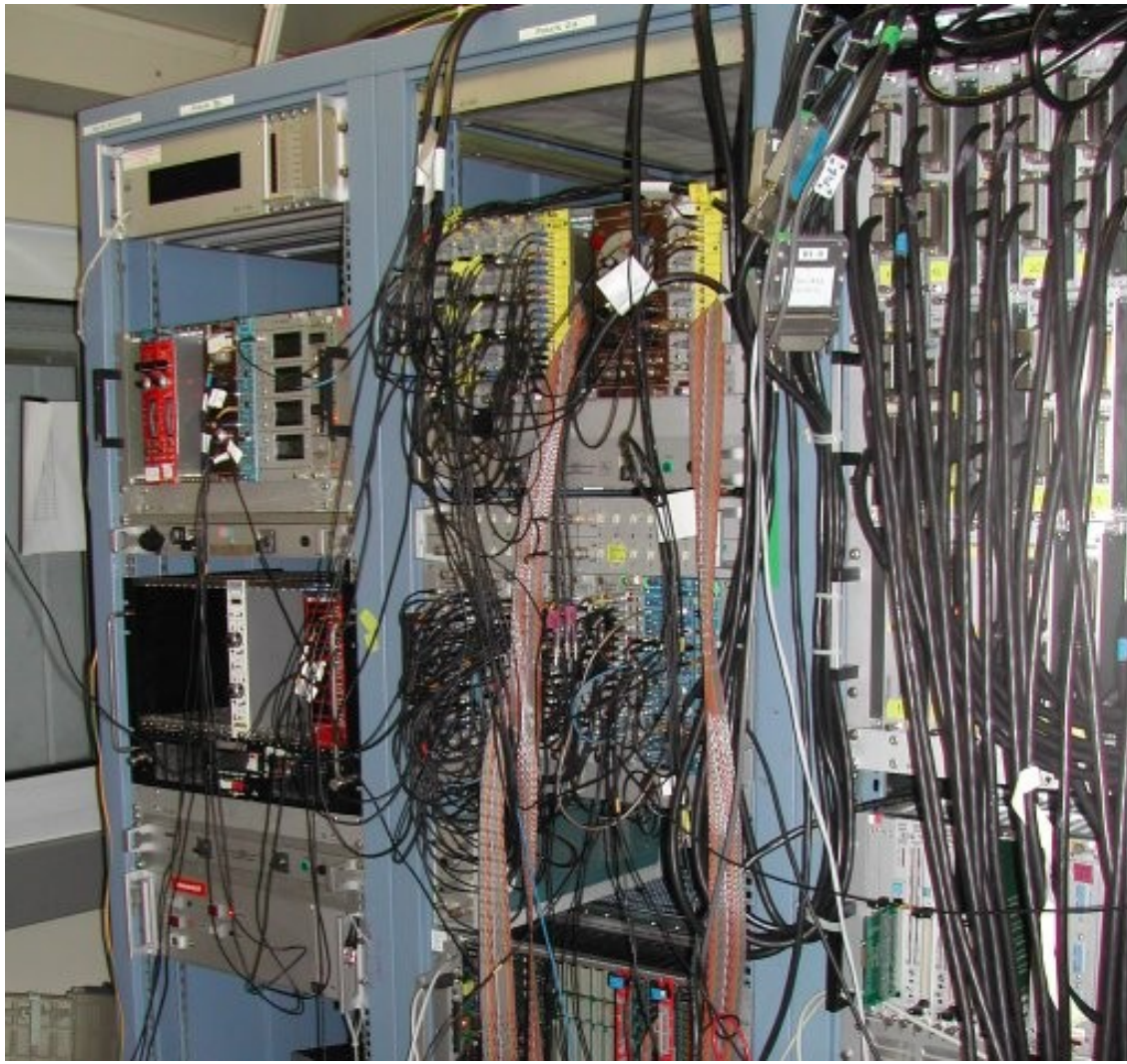
A great deal of FUN?

Thank you!

Next: Bonus slides – cable documentation

# Just a few cables...





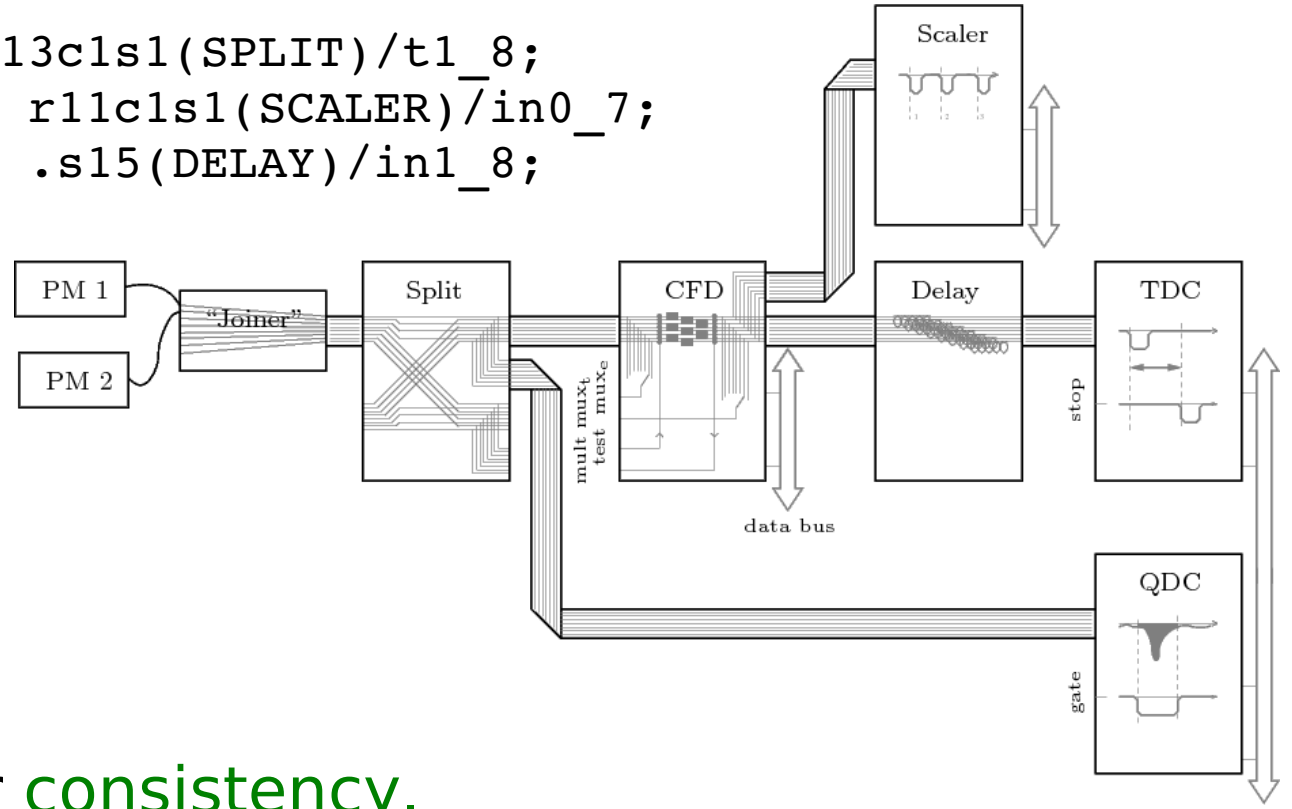
Solution / workaround ---------->

# Support tool: cable documentation

```
CF8103(r12c2s1)
{
  SERIAL("LCF6343");  // Comments

 in1_8: "N11 CFTN1" <- , r13c1s1(SPLIT)/t1_8;
 th1_8: "1/1"         -> , r11c1s1(SCALER)/in0_7;
 tb1_8: "CR2 SL1"     -> , .s15(DELAY)/in1_8;

 m:            .c11s3/in1;
 test:         .s23/out1;
 mux_tb:       .s22/in1a;
 mux_e:        .s22/in5a;
 mux_mon: .s22/in9a;
}
```

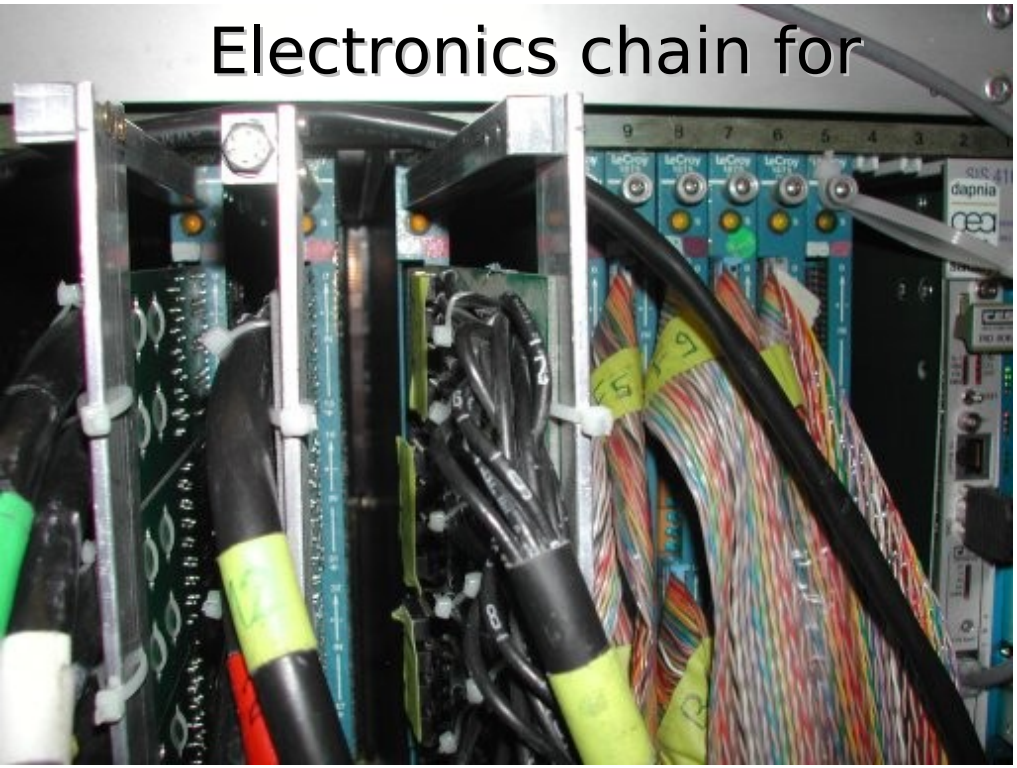


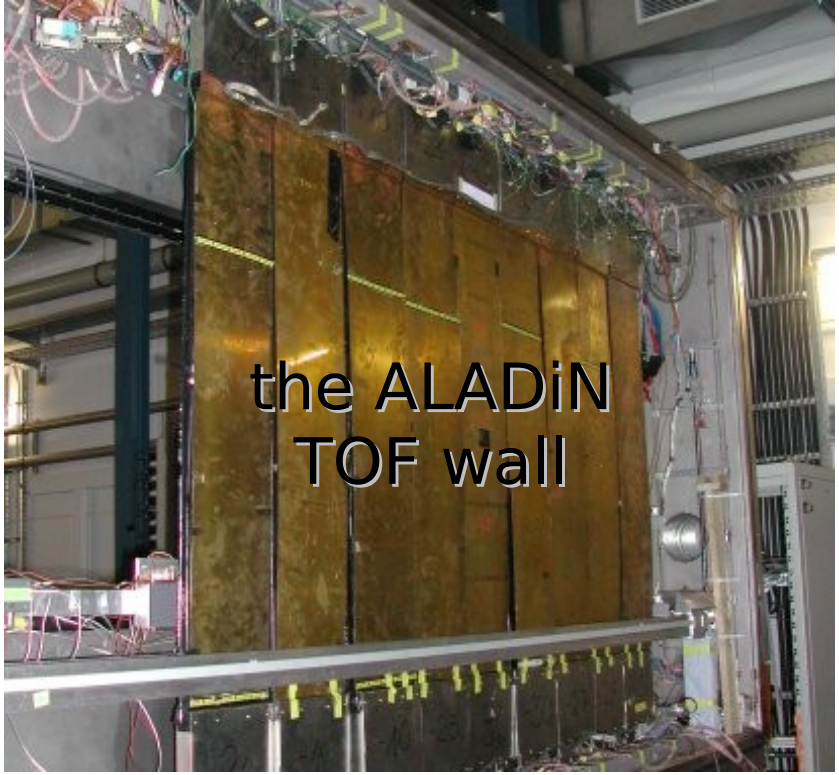

C-like text format.

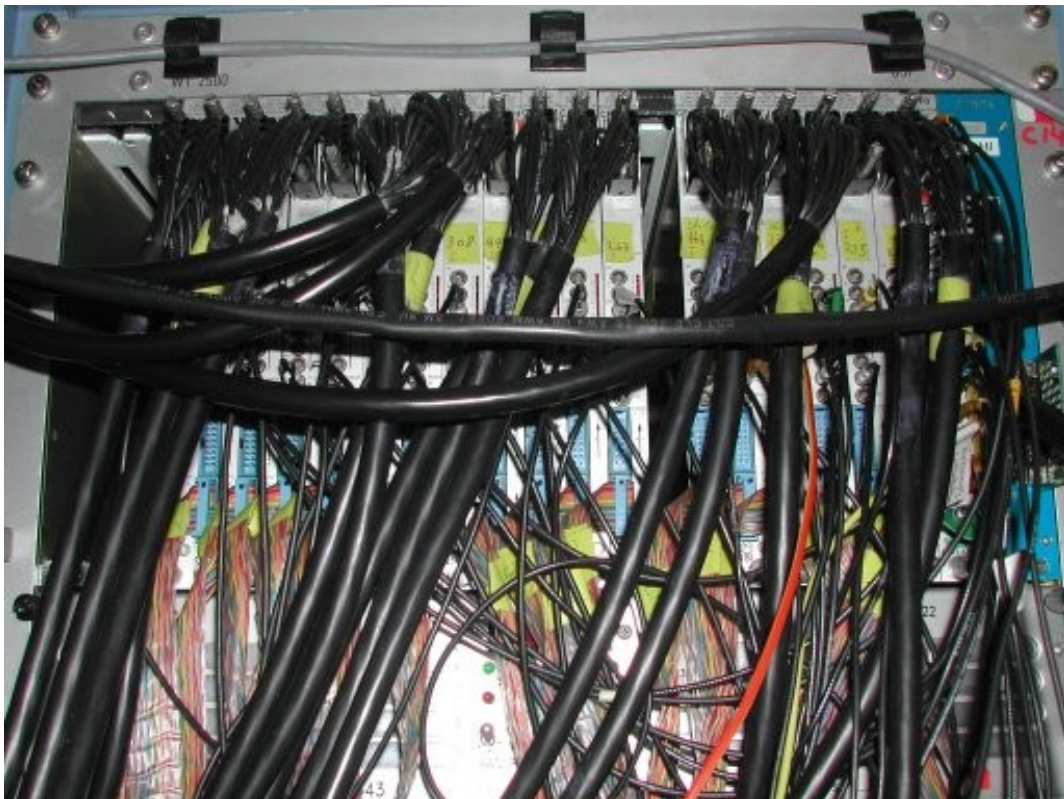
Parsed and checked for consistency.
(Every cable documented twice – at both ends.)

Checker generates tables for unpacking and slow-control.

# S304 cable doc



1 PhD,  < 1 week

(= a few days)

→ working unpacking
and mapping