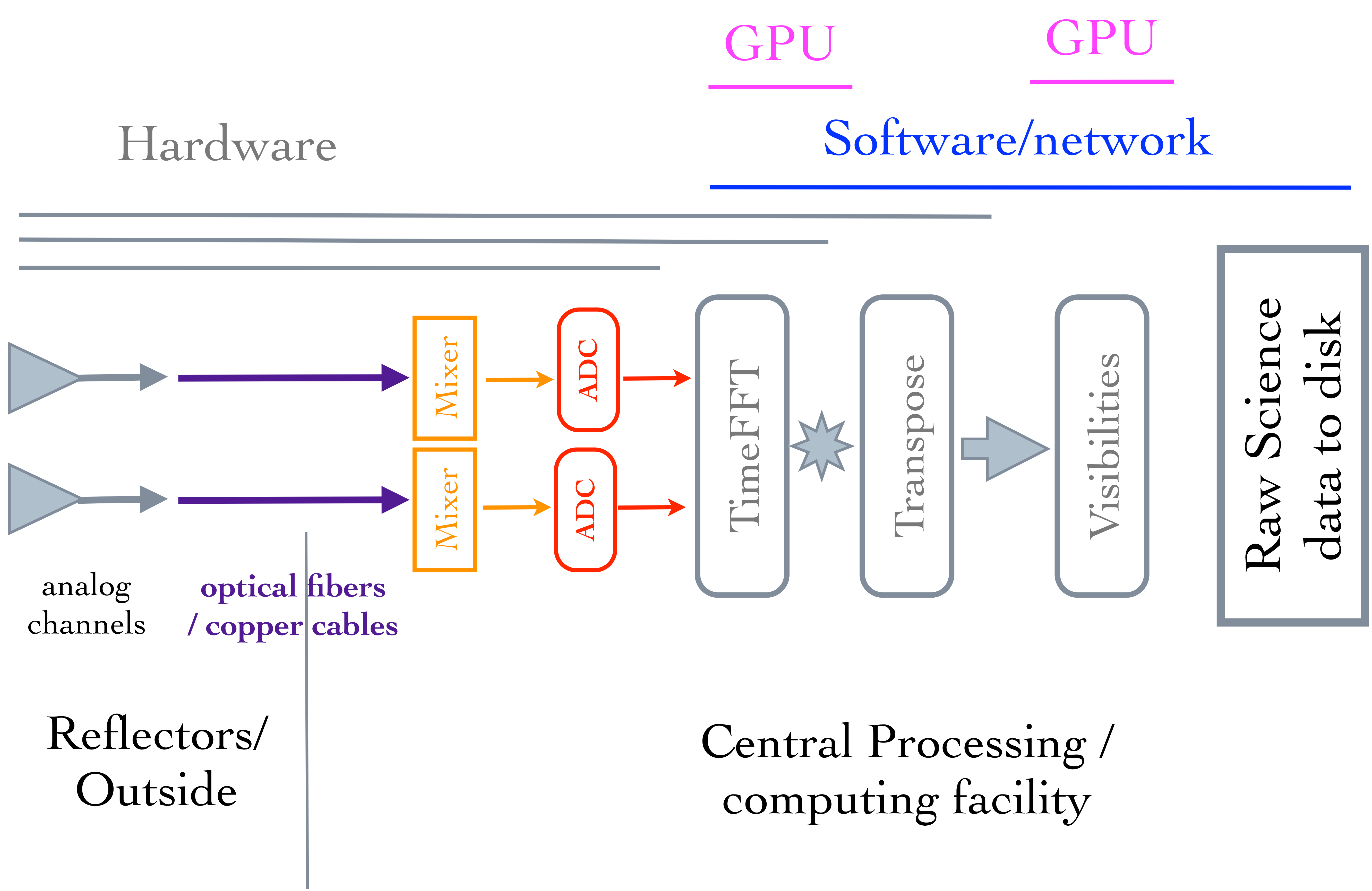
Tianlai data flow, processing, software architecture

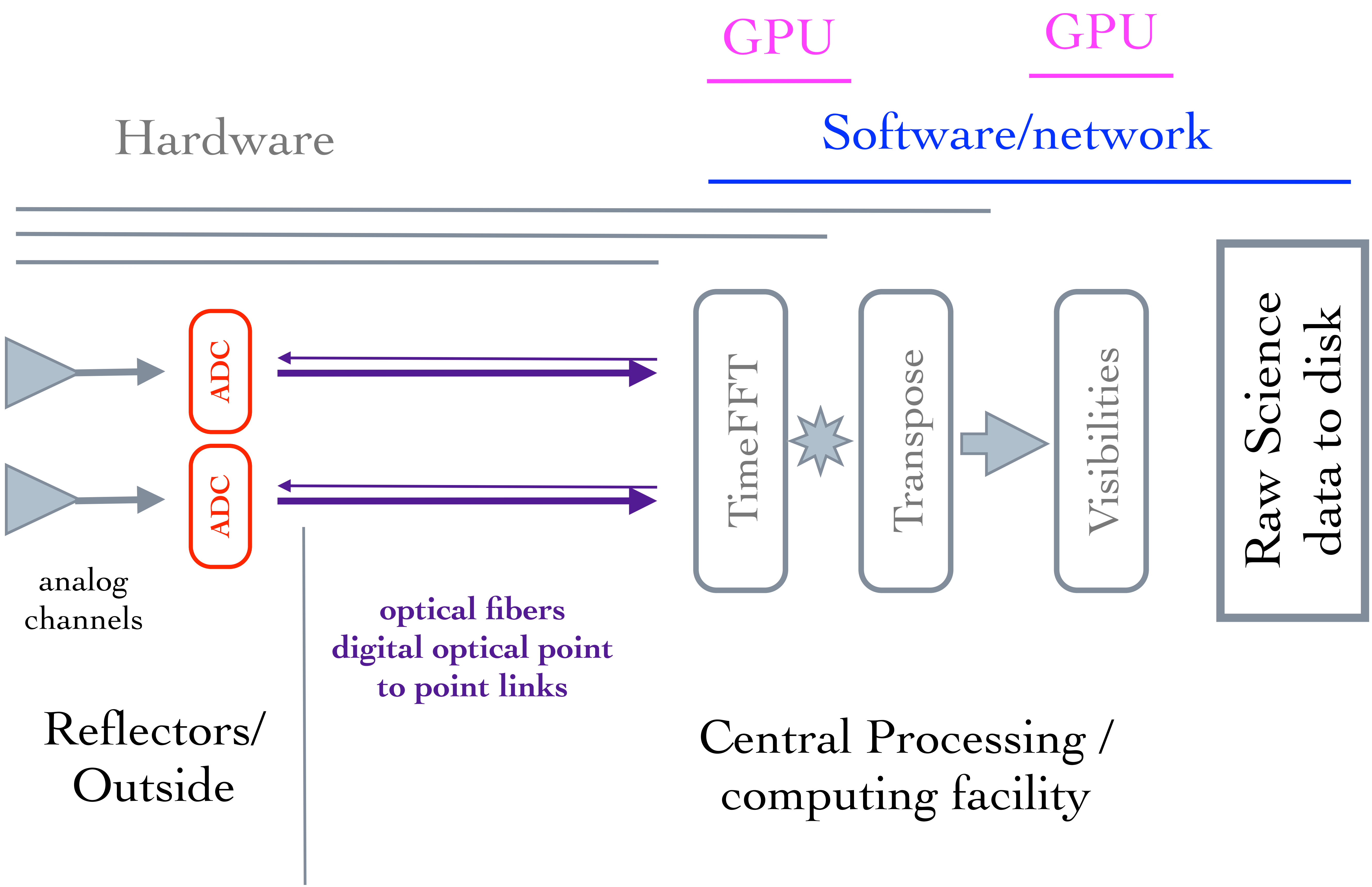
R. Ansari

Tianlai Electronic chain overview (1)



R. Ansari - May 2013

Tianlai Electronic chain overview (2)



R. Ansari - May 2013

Main data processing levels

- Level 1 : Data acquisition and on-line processing
  - ➠ raw science data (wave form or visibilities …)
- Level 2 : cleaning, calibration
  - ➠ calibrated, cleaned / flagged science data (visibilities)
- Level 3.a : map making
  - ➠ visibilities to 3D sky maps (data cubes)
- Level 3.b : component separation (foreground removal) power spectrum computation …
- Level 4 : from level 3 data to science products

Level 1 software

- Instrument & hardware configuration and control
- Data acquisition: (a) raw data (wave form), (b) FFT data, (c) visibility data to disk
- Acquisition and storage of auxiliary data (housekeeping) data : telescope configuration, weather, temperatures, …
- Suggested main communication channel between hardware & software: ethernet (TCP / UDP) and data exchanged in form of packets
- On-line processing : some RFI cleaning, data flagging, visibility computation with fine time sampling (<~ 0.1 s)
- If possible, some on-line monitoring information
- Should be flexible enough to adapt to different hardware configuration (CPU, GPU …)
- Performance & efficiency & robustness is crucial
- Output data written in a well supported format, suggested format: FITS

Software requirements

- An important component to ensure efficient collaboration
- Would be helpful to use a common base layer (library and associated tools) for developing the acquisition and key components of the data analysis software
- Required qualities: robustness, efficiency and reliability Efficiency, reliatibility, the question of maintenance & evolution should be addressed
- Acquisition software should handle different observation modes (electronic calibration & tests, sky source calibration, normal observation mode …)
- Data storage format & scheme, should cover all the experiment needs (Science Data, Housekeeping data) - should ensure easy exchange of the processed data
- Ensure an efficient implementation of the computing intensive part (visibility computation, sky map reconstruction …)
- Should be flexible enough to adapt to different hardware configuration (CPU, GPU …)
- Leave enough freedom to scientists for high level data analysis

Data flow & data rates …

Visibilities:

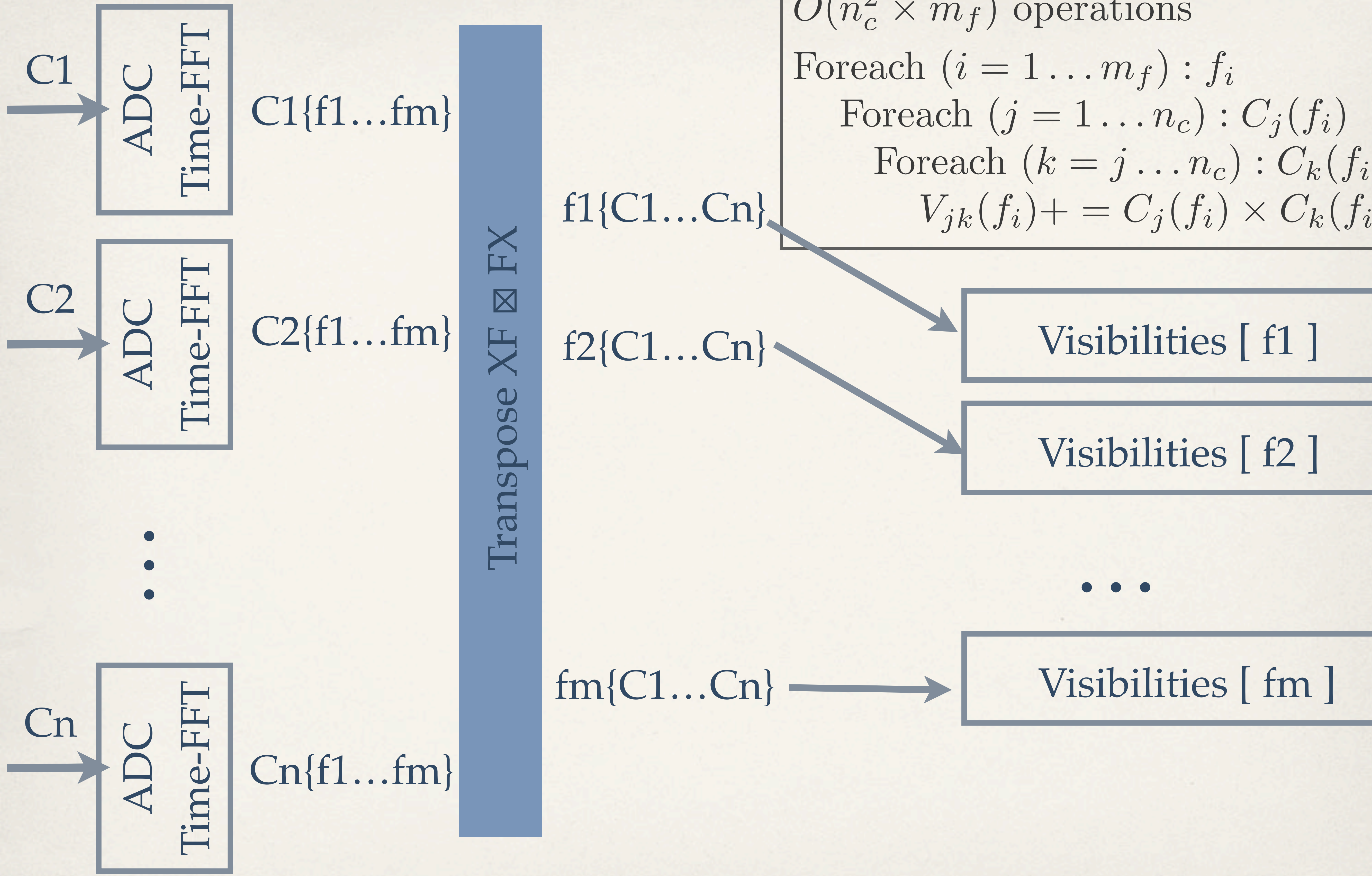
$O(n_c^2 \times m_f)$ operations

Foreach $(i = 1 \ldots m_f) : f_i$

  Foreach $(j = 1 \ldots n_c) : C_j(f_i)$

    Foreach $(k = j \ldots n_c) : C_k(f_i)$

      $V_{jk}(f_i) += C_j(f_i) \times C_k(f_i)$



Data flow

1. Sampling : (NFeed) × (BandWidth x 2)
2. FFT : In Data Rate = Out data Rate = (NFeed) × (BandWidth x 2) [x 4 →float]
3. Transpose (grouped in M frequency bands)
   - In Data Rate = (NFeed) × (BandWidth x 2) [x 4 →float]
   - Out Data Rate = M × (NFeed) × (BandWidth x 2 / M) [x 4 →float]
4. Visibility computation (NFeed^2 /2=NVis)
   - In Data Rate = M × (NFeed) × (BandWidth x 2 / M) [x 4 →float]
   - Out Data Rate = M × (NFeed^2 /2) × (BandWidth x 2 / M) / **10^5** [x 4 →float]

10^5 : assuming 0.1 s time sampling and ~10 000 freq. bins (~ 10 kHz resolution)

CPU & bandwidth requirements

	A	B	C
NFeed	32	256	1024
BandWidth	100 MHz	200 MHz	400 MHz
1 → 2 → 3	6.4 GBytes/sec	100 GBytes/sec	800 GBytes/sec
M	8	64	256
3/M →	0.8 GBytes/sec	1.6 GBytes/sec	3.2 GBytes/sec
NVis	528	32896	526336
@4 TFlops	~ 1	~ 100	~ 3200
4 → / M	5 MBytes/sec	50 MBytes/sec	400 MBytes/sec

Easy … *Challenging* … ??

Level 1 & 2 data rates for A: 32 feeds = 16 x 2 polar, 100 MHz, M=8

- **Level 1 output data rate**
- ~ 5 MBytes/s for each frequency sub-bands
- ~ 40 MBytes/s total
- ~500 GBytes / day for each sub-band
- ~4 TBytes / day in total
- ~ 1500 TBytes / year for the raw science data
- may be reduced by a factor 10-50 by further processing & binning in time (1-5 seconds) → ≤ 100 TB / year
- **Level 2 output data rate**
- ≲ 100 TBytes / year (~ 10 TB / month) for cleaned science data

BAORadio TAcq software

An object oriented C++ framework for the high level acquisition / processing software for Tianlai
Can be used as a basis for Level 1 & Level 2 pipelines

BAORadio Acquisition/Processing software

- Multi-thread programs / object oriented (C++) architecture
- FITS files (+ directory structure) for data storage (raw data/Fourier coefficient/Visibilities/spectra)
- **BRPaquet** hardware / software data exchange unit
- **RAcqMemZoneMgr** (memory manager) insure thread synchronisation/coordination
- Thread objects (ZThread) perform differents tasks
- Uses the **SOPHYA** C++ class library

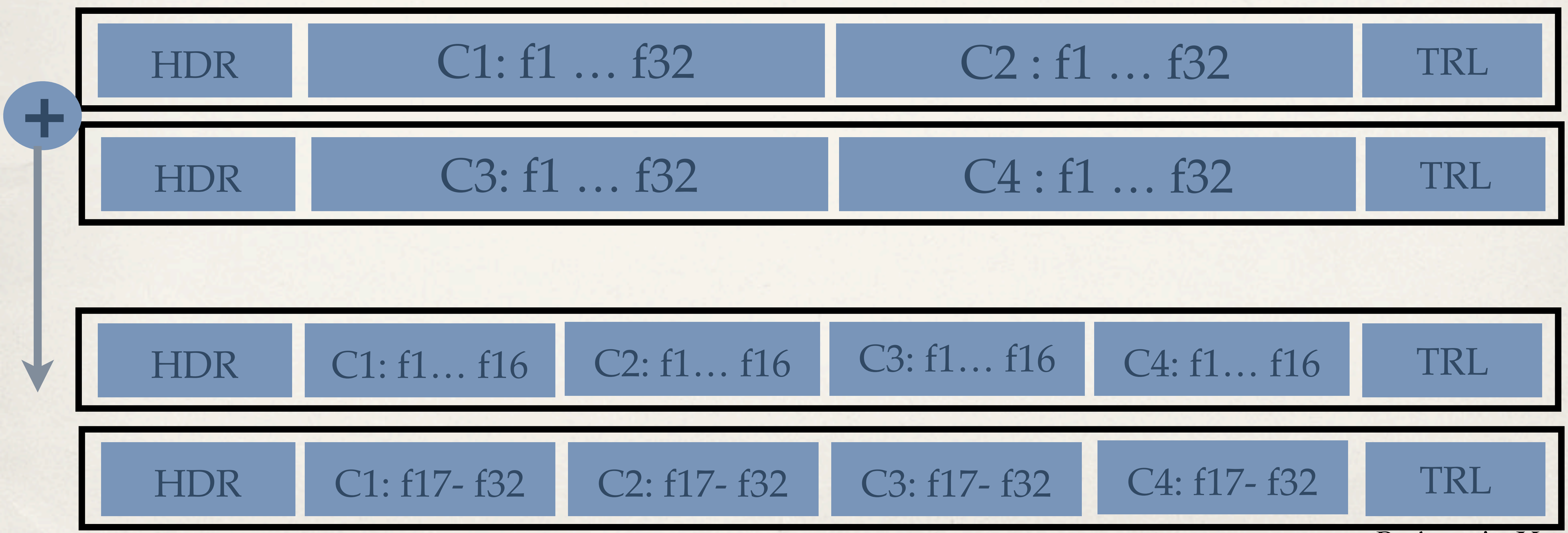
http://www.sophya.org

BRPaquet

- HDR/TRL: Data description, HW Time Tag, paquet length …
- Data : Raw or FFT coefficients, one or multiple channels
- HDR+TRL = 40 Bytes, Paquet length (variable)

HDR :

- Packet length
- DataDesc
- TimeTag
- FrameCounter



FITS format & data files

```
SIMPLE  =                    T / file does conform to FITS standard
BITPIX  =                    8 / number of bits per data pixel
NAXIS   =                    2 / number of data axes
NAXIS1  =                16424 / nb of pixels along X = PaquetSize
NAXIS2  =                 5120 / nb of rows = NumberOfPaquets
DATEOBS = '2010-07-06T12:59:48.0' /  Observation Time (YYYY-MM-DDThh:mm:ss UT)
TMSTART = '2010-07-06T12:59:48.0' /  File Acqu. Start Time/Date
ACQVER  =                  7.1 / BAORadio Acq Software version
ACQMODE = 'rawlc' /  BAORadio Acq run  mode
BRPAQCFM= 'BR_Copy' /  BAORadio BRPaquet DataFormatConversion
FIBERNUM=                    1 /  Fiber number/id
SKYSOURC= 'UGC04358-RA=8h21m26-DEC=-00d25m08' /  Source identification
TMEND   = '2010-07-06T12:59:48.0' /  File Acqu. End Time/Date
FCFIRST =                60173 /  First valid frame counter in file
FCLAST  =                66188 /  Last valid frame counter in file
TTFIRST =         178382123753 /  First valid timetag in file
COMMENT  BAO-Radio / MiniFITSFile
END
```

FITS header

- Output files uses directory structures + FITS format
- Raw data (time samples), FFT data and visibility matrices written in FITS format
- BRPaquet complete structure written to FITS files for raw/fft data dumps
- FITS headers used to add auxiliary informations, such time time&date, start-end timetag, pointing/source identification …
- Visibility matrices written also in FITS format

http://heasarc.gsfc.nasa.gov/fitsio/

Some of the TAcq classes

- Class **BRPaquet** & **BRPaqChecker**
- Class **RAcqMemZoneMgr** (Multi fiber / link managed memory zone)
- Thread (task) classes **(ZThread)** (see next slide)
- Class **MiniFITSFile**

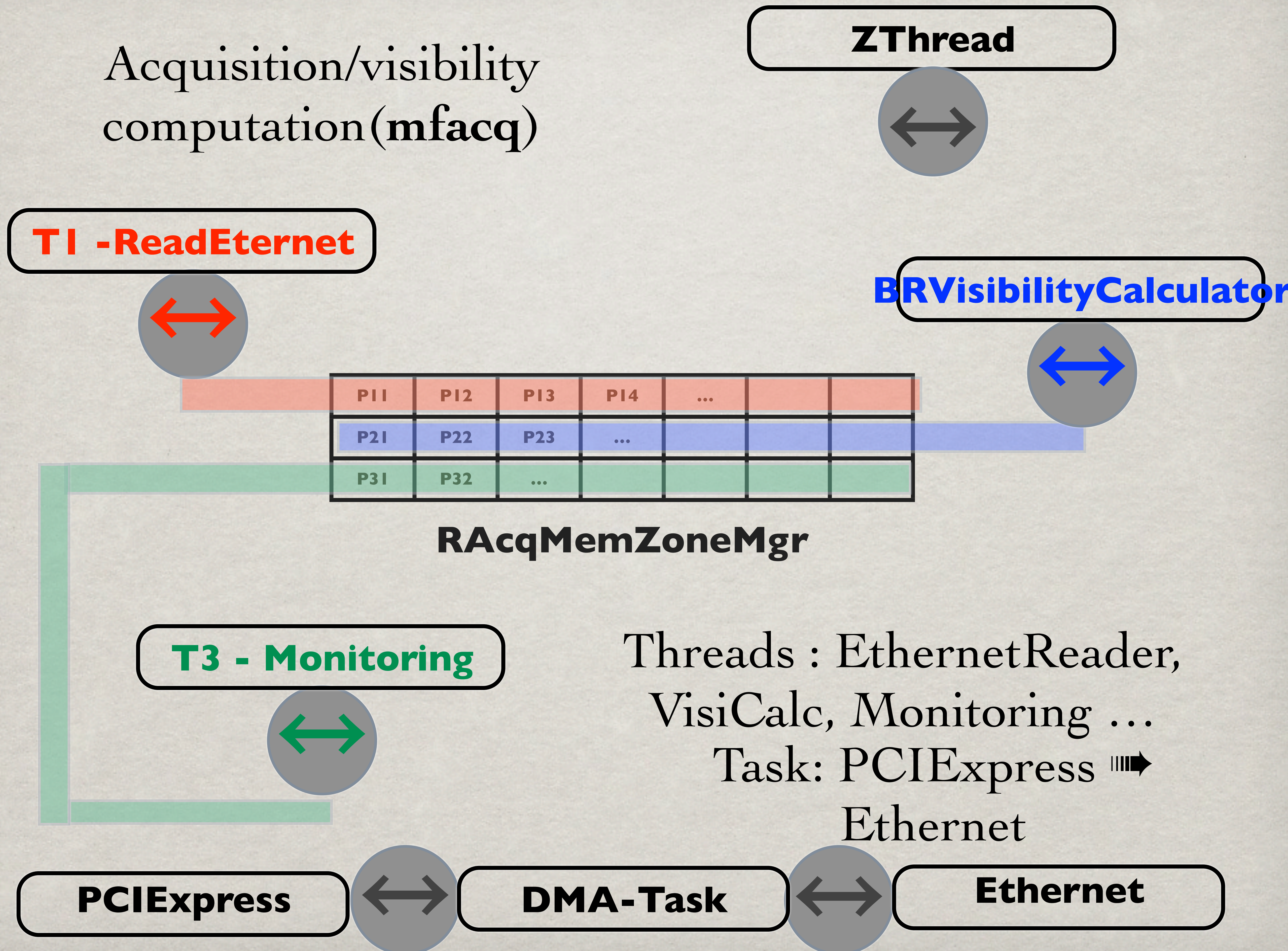
http://bao.lal.in2p3.fr/TAcq/
http://www.sophya.org

Thread (Task) classes

- **PCIEMultiReader** (PCI-Express DMA to memory)
- **PCIEToEthernet** (PCI-Express DMA to ethernet)
- **MultiDataSaver** (Writes packet data to disk (FITS format))
- **EthernetReader** (Read packet from ethernet to memory)
- **BRMultiFitsReader** (Multi fiber fits reader, align in time)
- **BRVisibilityCalculator** (Computes visibilities)
- **BRFFTCalculator** (perform FFT on raw data)
- **MonitorProc(s)** (Monitoring during processing)

A TAcq example …

```
// ----- Setup the pipeline
// Setup the central memory manager / controler
    RAcqMemZoneMgr mmgr(par.nzones_, par.dirlist_.size(), par.npaqinzone_, par.paqsize_, procsz);
    mmgr.SetFinalizedMask((uint_4)MemZS_ProcB);
// Setup the FITS files reader
    BRMultiFitsReader reader(mmgr, par.dirlist_, par.rdsamefc_, par.imin_, par.imax_, par.istep_);
// Setup a time FFT processor (assuming we have wave form data)
    BRFFTCalculator procfft(mmgr, par.fgsinglechannel_);
// The aim is to compute time averaged spectra
// So setup a processor to do that ...
    BRMeanSpecCalculator procms(mmgr, par.outpath_, par.nmean_, par.fgdatafft_, par.fgsinglechannel_);
//----- We are now ready to run
// Let's start the processors (threads ...)
    reader.start();
    procfft.start();
    procms.start();
//--- We just now wait for the work to be finished ...
    usleep(200000);
    reader.join();
    procfft.join();
    procms.join();
//--- and print some overall stats on memory manager - controler
    mmgr.Print(cout);
```


Acquisition/visibility computation(mfacq)
ZThread
T1 -ReadEternet
BRVisibilityCalculator
P11
P12
P13
P14
...
P21
P22
P23
...
P31
P32
...
RAcqMemZoneMgr
T3 - Monitoring
Threads : EthernetReader, VisiCalc, Monitoring ...
Task: PCIExpress ➠ Ethernet
PCIExpress
DMA-Task
Ethernet

Flexible CPU or GPU correlator

BRVisibilityCalculator can easily be adapted to perform the heavy calculation on GPU …

BRVisibilityCalculator code has > 800 lines

less than 20-50 lines has to be executed on GPU …

```
// kpair=numero sequentiel de la paire: 0->(0,0), 1->(0,1), 2->(0,2), 3->(0,3), 4->(1,1), 5->(1,2) ...
  sa_size_t kpair=0;
  sa_size_t k=0;     // numero de ligne dans la matrice des visibilites
  for(size_t i=0; i<vpdata_.size(); i++) {
    for(size_t j=i; j<vpdata_.size(); j++) {
      kpair++;
      if (kpair<(pairst_+1))  continue;
      if (kpair>=(pairst_+nbpairs_+1))  break;
      if (fgpimp_&&(i!=j)&&((i+j)%2==0))  continue;  // calcul des visib avec numero pair-impair + autocorrel
      TVector< complex<r_4> > vis = vismtx_.Row(k);  k++;

      if (fgdataraw_) {  // Donnees firmware RAW apres TF soft
        for(sa_size_t f=1; f<vis.Size(); f++) {
          vis(f) += vpdatar_[i][f] * conj(vpdatar_[j][f]);
        }
      }
      else {   // donnees firmware FFT
        for(sa_size_t f=1; f<vis.Size(); f++) {
          vis(f) += complex<r_4>((r_4)vpdata_[i][f].realB(), (r_4)vpdata_[i][f].imagB()) *
            complex<r_4>((r_4)vpdata_[j][f].realB(), -(r_4)vpdata_[j][f].imagB());
        }
      }
      nb_flop_ += (8.*(r_8)(vis.Size()-1));
    }
  }
```