Development of KAGRA Burst Pipeline

Kazuhiro Hayama
on behalf of KAGRA Burst Group

KAGRA Burst Group

- **Hayama**
- **Arima, Kanda,Yokozawa**

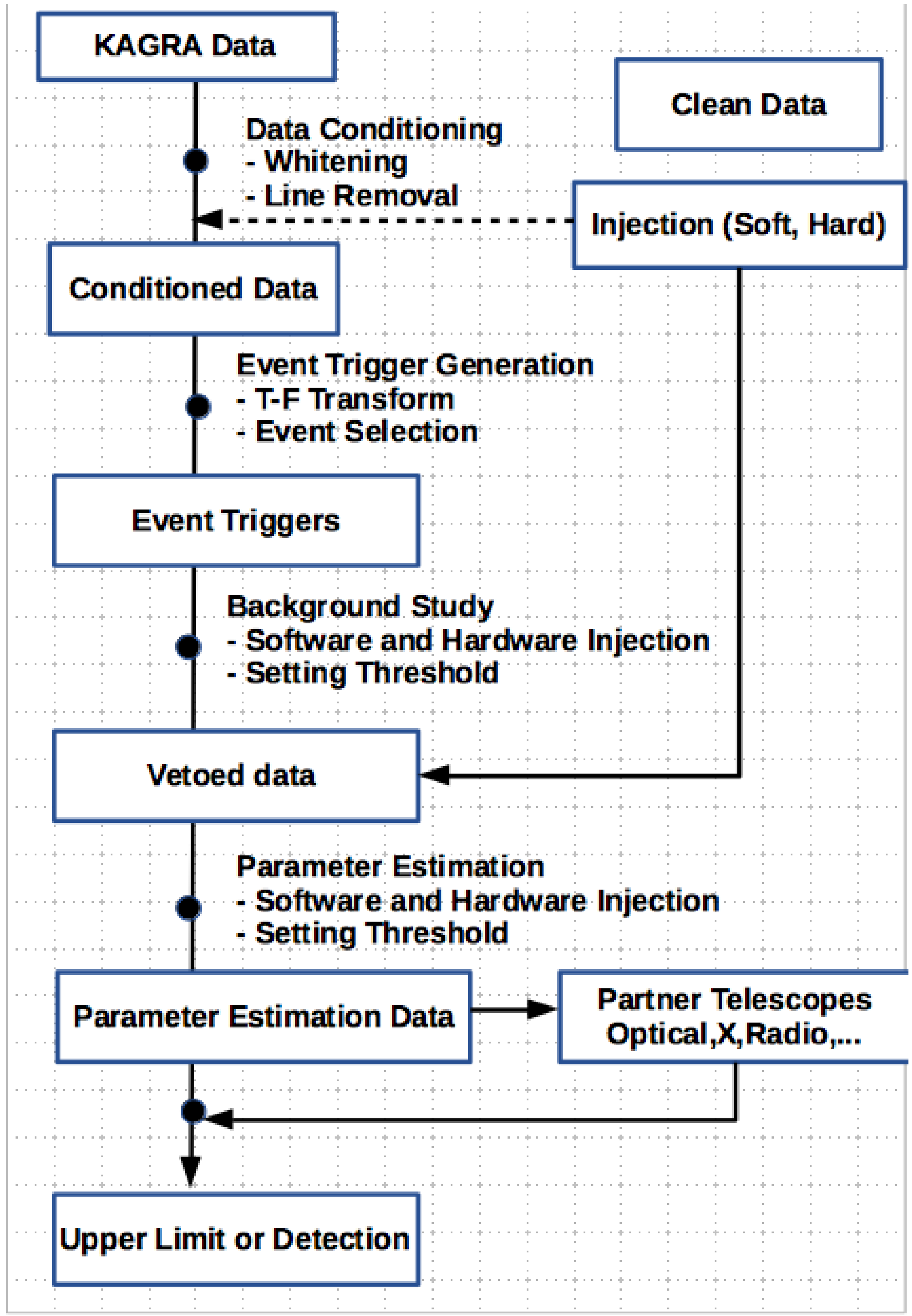
- **The burst pipeline is being developed using**
  - **KAGALI (C-based)**
  - **HasKAL (Haskel-based)**

    The role of HasKAL is mainly two
    - **As a wrapper of KAGALI library**
    - **As a detector characterization library**

    Speed is 1~2 x C

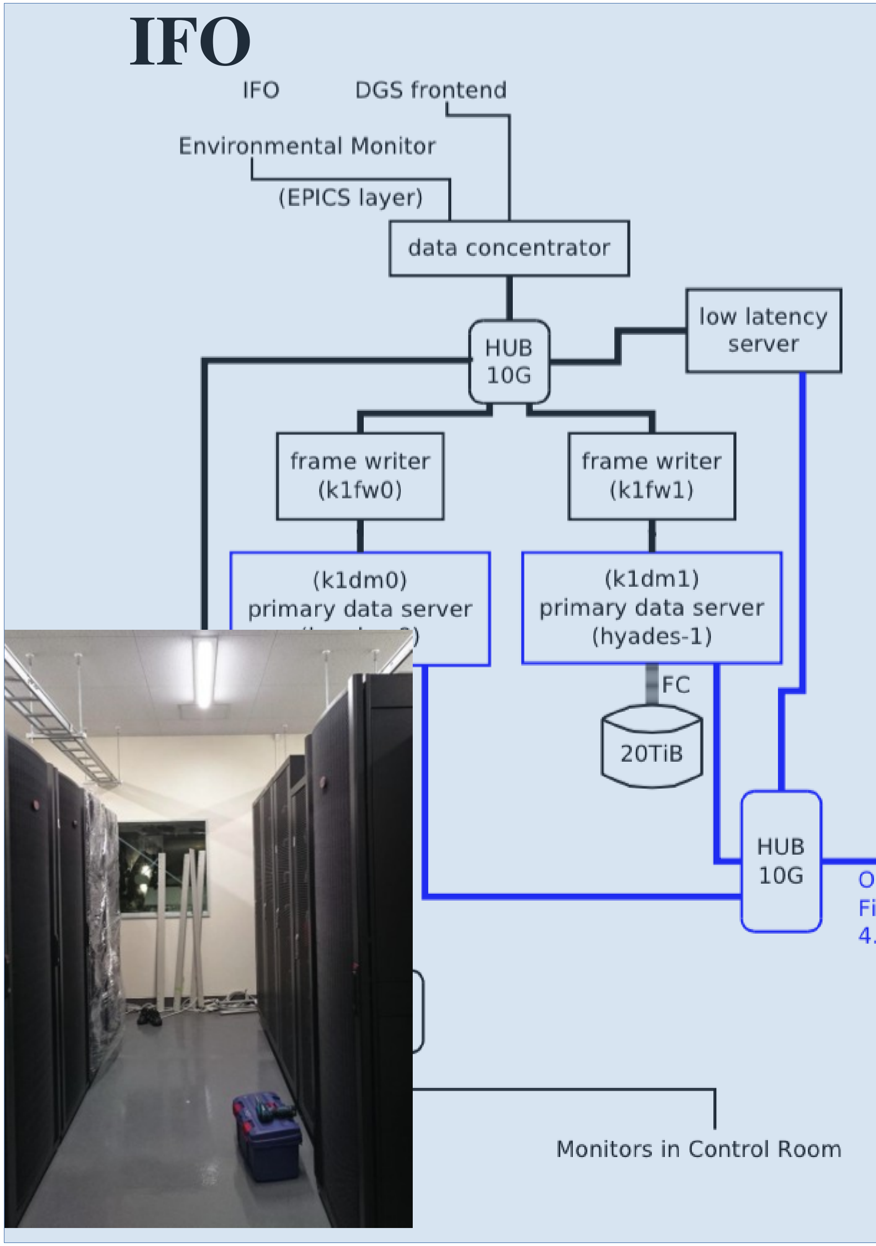
    Parallelization is very easy!

Flow Chart

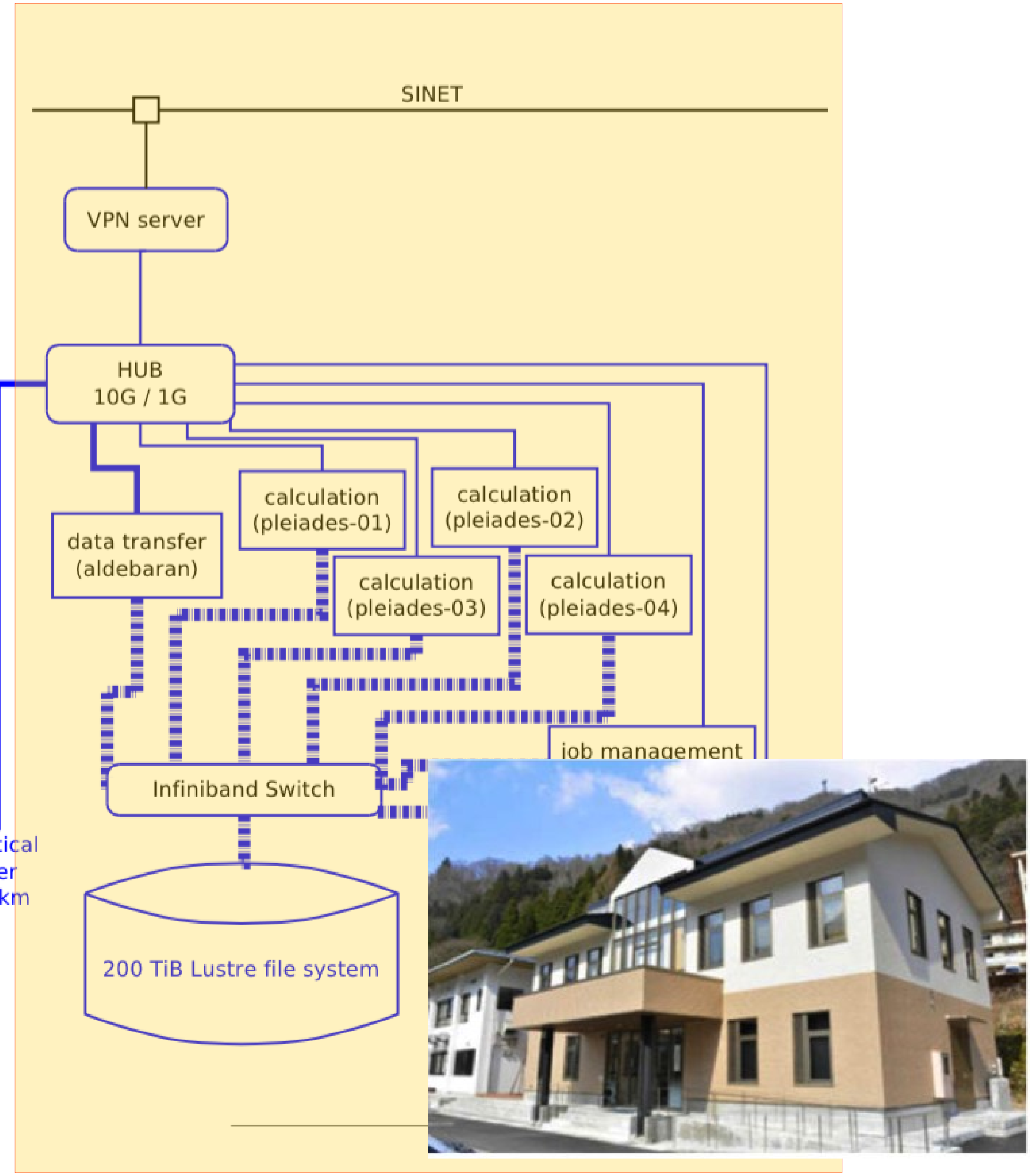


KAGRA Data
Data Conditioning
- Whitening
- Line Removal
Clean Data
Injection (Soft, Hard)
Conditioned Data
Event Trigger Generation
- T-F Transform
- Event Selection
Event Triggers
Background Study
- Software and Hardware Injection
- Setting Threshold
Vetoed data
Parameter Estimation
- Software and Hardware Injection
- Setting Threshold
Parameter Estimation Data
Partner Telescopes
Optical,X,Radio,...
Upper Limit or Detection

Data from KAGRA

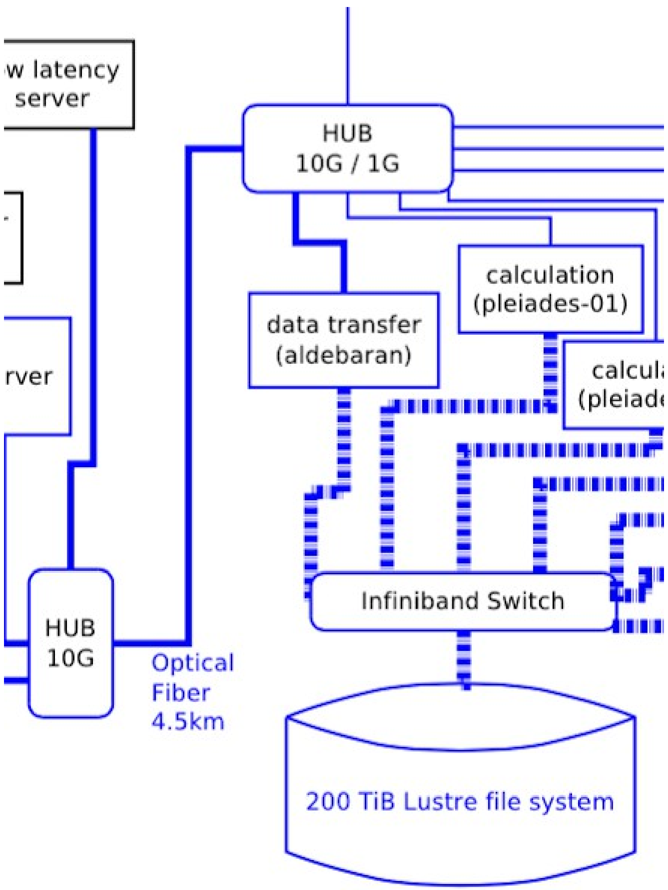


Kamioka Mine



Analysis Building

Data from KAGRA



- **Data is transferred to the analysis building via optical fiber lines.**
- **The data is first stored in a server with 200Tib storage.**
- **The data will be ~TByte/day. So we need to have a database system as a part of the KAGRA DAQ system.**

Database

Two methods are being developed.

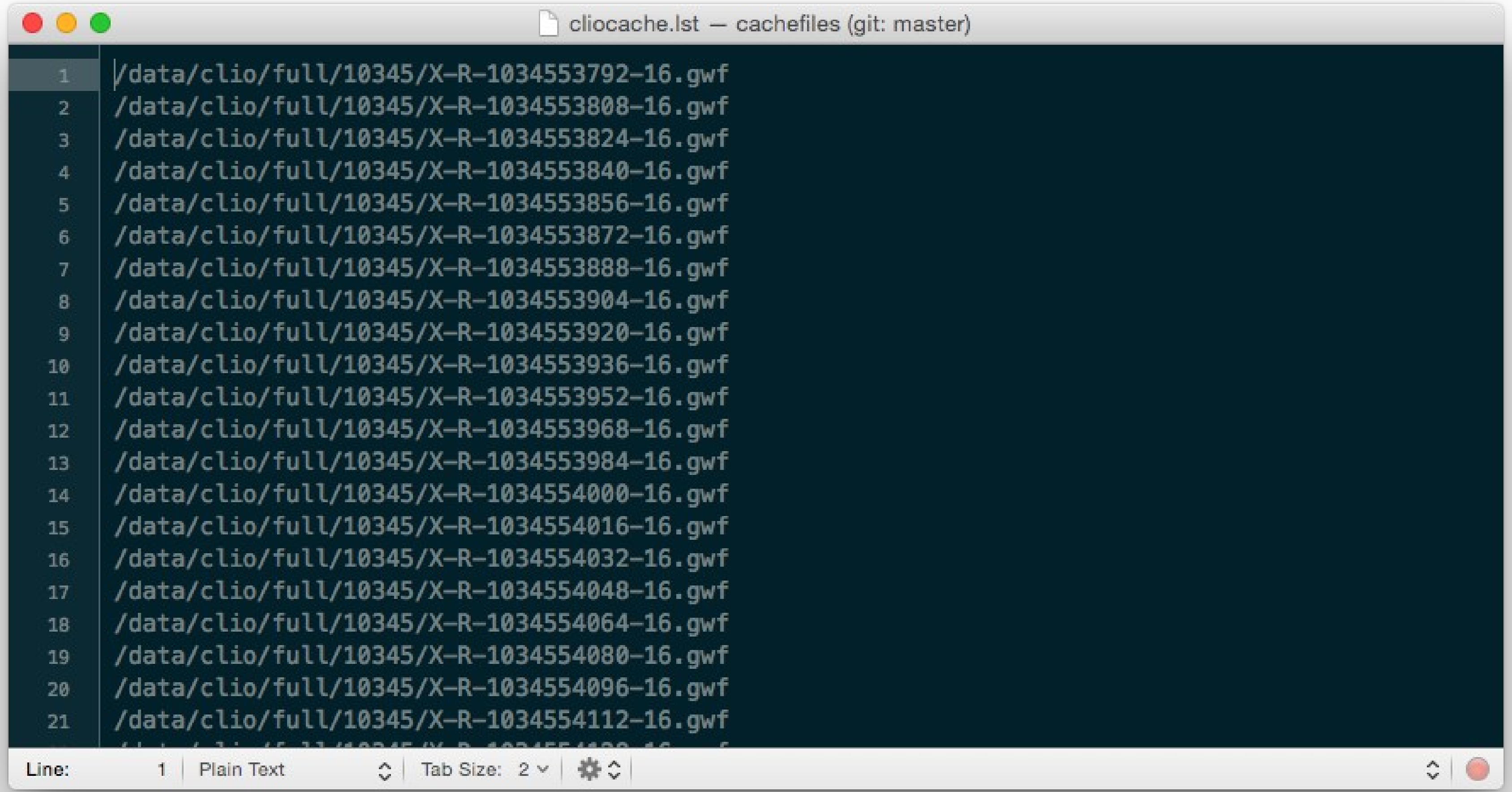
The approach to be adopted will be decided with DMG subsystem.

- **framecache**
- **Database**

framecache

Format of the framecache is not defined yet.

What parameters should be included?



Database

- **UpdateFrameDB**
  The information of the file is inserted into a database as soon as a frame file is stored in a specific directory.
- **The database engine is MySQL.**

2. mysql

```
mysql> select * from framedb;
```

frame_id	fname	gps_start	gps_stop	chname	sampling_rate	dq_flag
1	/data/kagra/xend/test/R0201/K-K1_R-1113209036-32.gwf	1113209036	1113209068	ADC1	2048	4
2	/data/kagra/xend/test/R0201/K-K1_R-1113209036-32.gwf	1113209036	1113209068	ADC2	2048	4
3	/data/kagra/xend/test/R0201/K-K1_R-1113209036-32.gwf	1113209036	1113209068	ADC3	2048	4
4	/data/kagra/xend/test/R0201/K-K1_R-1113209036-32.gwf	1113209036	1113209068	ADC4	2048	4
5	/data/kagra/xend/test/R0201/K-K1_R-1113209036-32.gwf	1113209036	1113209068	ADC5	2048	4
6	/data/kagra/xend/test/R0201/K-K1_R-1113209036-32.gwf	1113209036	1113209068	ADC6	2048	4
7	/data/kagra/xend/test/R0201/K-K1_R-1113209036-32.gwf	1113209036	1113209068	ADC7	2048	4
8	/data/kagra/xend/test/R0201/K-K1_R-1113209036-32.gwf	1113209036	1113209068	ADC8	2048	4
9	/data/kagra/xend/test/R0201/K-K1_R-1113209036-32.gwf	1113209036	1113209068	K1:PEM-EX_ACC_NO2_X_FLOOR	2048	4
10	/data/kagra/xend/test/R0201/K-K1_R-1113209036-32.gwf	1113209036	1113209068	K1:PEM-EX_ACC_NO2_Y_FLOOR	2048	4
11	/data/kagra/xend/test/R0201/K-K1_R-1113209036-32.gwf	1113209036	1113209068	K1:PEM-EX_ACC_NO2_Z_FLOOR	2048	4
12	/data/kagra/xend/test/R0201/K-K1_R-1113209036-32.gwf	1113209036	1113209068	K1:PEM-EX_MAG_X_FLOOR	2048	4
13	/data/kagra/xend/test/R0201/K-K1_R-1113209036-32.gwf	1113209036	1113209068	K1:PEM-EX_MAG_Y_FLOOR	2048	4
14	/data/kagra/xend/test/R0201/K-K1_R-1113209036-32.gwf	1113209036	1113209068	K1:PEM-EX_MAG_Z_FLOOR	2048	4
15	/data/kagra/xend/test/R0201/K-K1_R-1113209036-32.gwf	1113209036	1113209068	K1:PEM-EX_MIC_FLOOR	2048	4
16	/data/kagra/xend/test/R0201/K-K1_R-1113209036-32.gwf	1113209036	1113209068	K1:PEM-EX_REF	2048	4
17	/data/kagra/xend/test/R0201/K-K1_R-1113209036-32.gwf	1113209036	1113209068	ADC1-RAW	2048	4
18	/data/kagra/xend/test/R0201/K-K1_R-1113209036-32.gwf	1113209036	1113209068	ADC2-RAW	2048	4
19	/data/kagra/xend/test/R0201/K-K1_R-1113209036-32.gwf	1113209036	1113209068	ADC3-RAW	2048	4
20	/data/kagra/xend/test/R0201/K-K1_R-1113209036-32.gwf	1113209036	1113209068	ADC4-RAW	2048	4
21	/data/kagra/xend/test/R0201/K-K1_R-1113209036-32.gwf	1113209036	1113209068	ADC5-RAW	2048	4
22	/data/kagra/xend/test/R0201/K-K1_R-1113209036-32.gwf	1113209036	1113209068	ADC6-RAW	2048	4
23	/data/kagra/xend/test/R0201/K-K1_R-1113209036-32.gwf	1113209036	1113209068	ADC7-RAW	2048	4
24	/data/kagra/xend/test/R0201/K-K1_R-1113209036-32.gwf	1113209036	1113209068	ADC8-RAW	2048	4
25	/data/kagra/xend/test/R0201/K-K1_R-1113209036-32.gwf	1113209036	1113209068	K1:PEM-EX_ACC_NO2_X_FLOOR-RAW	2048	4
26	/data/kagra/xend/test/R0201/K-K1_R-1113209036-32.gwf	1113209036	1113209068	K1:PEM-EX_ACC_NO2_Y_FLOOR-RAW	2048	4
27	/data/kagra/xend/test/R0201/K-K1_R-1113209036-32.gwf	1113209036	1113209068	K1:PEM-EX_ACC_NO2_Z_FLOOR-RAW	2048	4
28	/data/kagra/xend/test/R0201/K-K1_R-1113209036-32.gwf	1113209036	1113209068	K1:PEM-EX_MAG_X_FLOOR-RAW	2048	4
29	/data/kagra/xend/test/R0201/K-K1_R-1113209036-32.gwf	1113209036	1113209068	K1:PEM-EX_MAG_Y_FLOOR-RAW	2048	4
30	/data/kagra/xend/test/R0201/K-K1_R-1113209036-32.gwf	1113209036	1113209068	K1:PEM-EX_MAG_Z_FLOOR-RAW	2048	4
31	/data/kagra/xend/test/R0201/K-K1_R-1113209036-32.gwf	1113209036	1113209068	K1:PEM-EX_MIC_FLOOR-RAW	2048	4
32	/data/kagra/xend/test/R0201/K-K1_R-1113209036-32.gwf	1113209036	1113209068	K1:PEM-EX_REF-RAW	2048	4

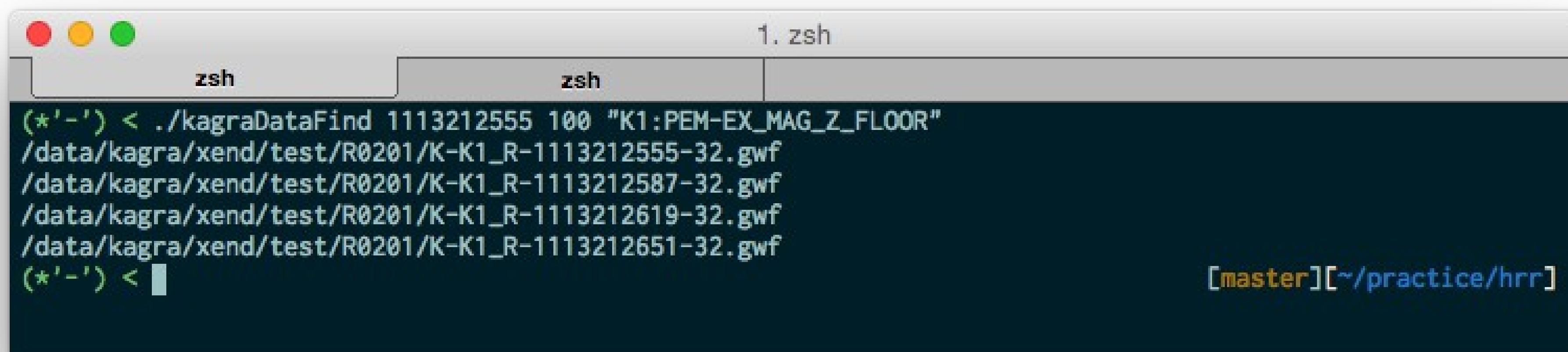
```
32 rows in set (0.00 sec)

mysql>
```

Accessing Database

- **Command line tools**
  - **kagraDataFind**
    If you give GPS start time, duration, channel_name, you can get a list of corresponding frame files.

e.g.
kagraDataFind 1113212555 100 "K1:PEM-EX_MAG_Z_FLOOR"



The software

- **MySQL 5.6.24**
  http://www.mysql.com
- **Generation of framecache**
  https://github.com/gw-analysis/detector-characterization/blob/master/HasKAL/src/HasKAL/FrameUtils/FileManipulation.hs
- **Database using MySQL**
  https://github.com/gw-analysis/detector-characterization/tree/master/HasKAL/src/HasKAL/DataBaseUtils

Data Structure

- **The data is defined below**

  https://github.com/gw-analysis/detector-characterization/tree/master/HasKAL/src/HasKAL/WaveUtils

- ```
  data WaveData = WaveData
    { detector :: Detector
    , dataType :: String
    , samplingFrequency :: Double
    , startGPSTime :: GPSTIME
    , stopGPSTime  :: GPSTIME
    , gwdata :: TimeSeries
    } deriving (Show, Eq, Read)


  data WaveProperty = WaveProperty
    { mean :: Vector Double -> Double
    , variance :: Vector Double -> Double
    , spectrum :: Vector Double -> Double -> [(Double, Double)]
    , spectrogram :: Vector Double -> Double -> [(Double, Double, Double)]
    }
  ```

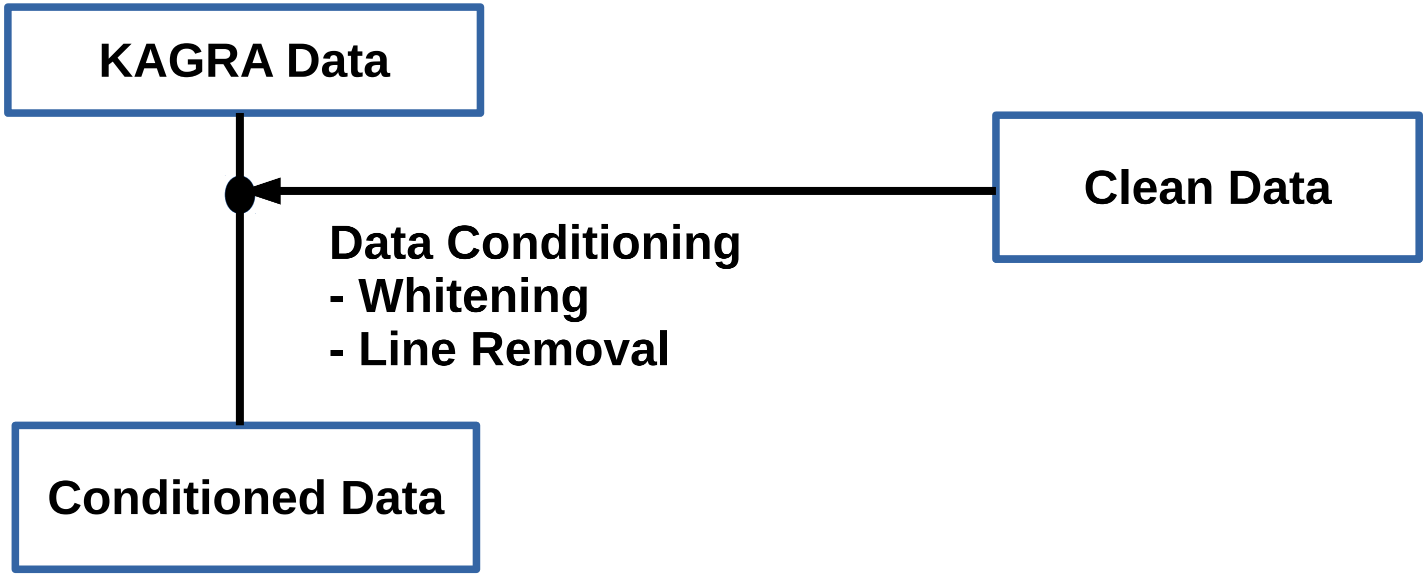
  https://github.com/gw-analysis/detector-characterization/blob/master/HasKAL/src/HasKAL/WaveUtils/Data.hs
  ```

Handling data

Every data processing comes with GPS time, …
So that we can avoid careless miss like wrong sampling frequency, time shift etc.

```haskell
updateWaveDatagwdata :: WaveData -> TimeSeries -> Maybe WaveData
updateWaveDatagwdata v w
  | dim (gwdata v)==dim w
    = Just $ mkWaveData (detector v) (dataType v) (samplingFrequency v) (startGPSTime v) (stopGPSTime v) w
  | otherwise = Nothing


dropWaveData :: Int -> WaveData -> WaveData
dropWaveData n x = do
  let t = (fromIntegral n) / (samplingFrequency x)
      newstartGPSTime = formatGPS $ deformatGPS (startGPSTime x) + t
      newgwdata = subVector n (dim (gwdata x) - n) (gwdata x)
  mkWaveData (detector x) (dataType x) (samplingFrequency x) newstartGPSTime (stopGPSTime x) newgwdata


takeWaveData :: Int -> WaveData -> WaveData
takeWaveData n x = do
  let t = (fromIntegral n) / (samplingFrequency x)
      newstopGPSTime = formatGPS $ deformatGPS (startGPSTime x) + t
      newgwdata = subVector 0 n (gwdata x)
  mkWaveData (detector x) (dataType x) (samplingFrequency x) (startGPSTime x) newstopGPSTime newgwdata
```

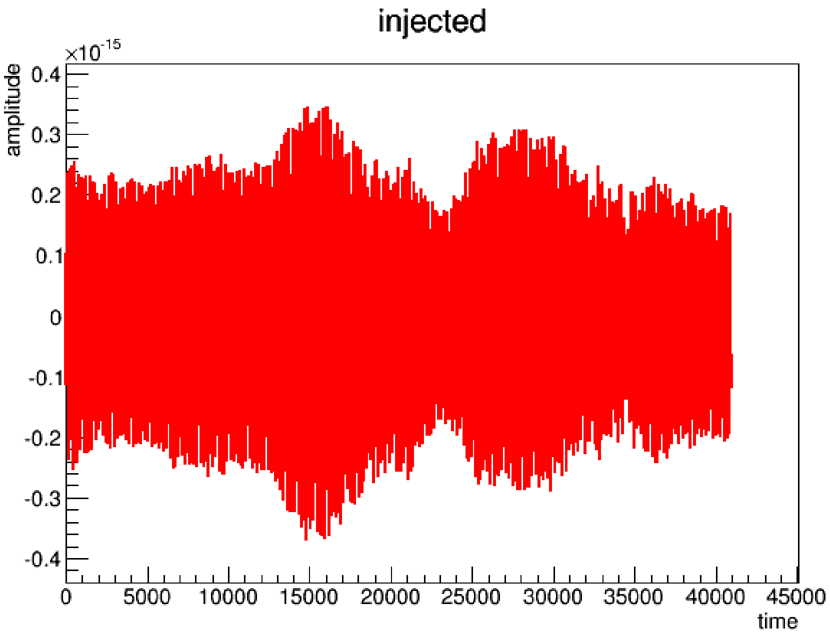
Data Conditioning

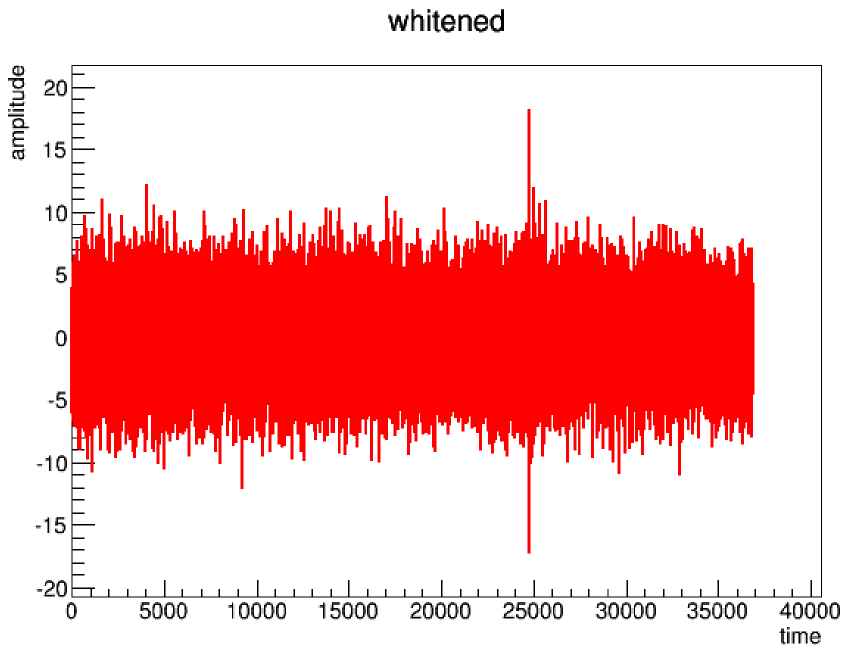


- **Clean Data Finder (Yokozawa)**
  To find “stationary” data without any tangients
- **Whitening (Hayama)**
  To remove frequency dependency from the data
- **Line Removal (Asano)**
  To remove narrow-band artifacts s.t. violin modes

Data Conditioning

- Linear Prediction Error Filter
  - Estimating IIR filter coefficients that obtain a transfer function having inverse of the PSD.
  - Very stable





The software

```
t9 <- getCurrentTime
print "{- whitening filter coefficients -}"
let trlen = truncate fs
    trdat = take trlen $ toList $ gwdata injected
    whnParam = lpefCoeff nC (gwpsd trdat nfft fs)

print (snd whnParam)
t10 <- getCurrentTime
print $ diffUTCTime t10 t9

t11 <- getCurrentTime
print "{- apply whitening filter -}"
let whnWaveData = dropWaveData (2*nC) $ whiteningWaveData whnParam injected
```

```
{- exposed functions -}
lpefCoeff :: Int -> [(Double,Double)] -> ([Double],Double)
lpefCoeff p psddat = (out,rho)
  where
    (out,rho) = levinson p r
    r = toList.fst.fromComplex.ifft.fromList
      $ [fs*nn*x/nn:+0|x<-(snd.unzip) psddat]
    fs = last.fst.unzip $ psddat
    nn = fromIntegral $ length psddat :: Double
```

```
whiteningWaveData :: ([Double],Double) -> WaveData -> WaveData
whiteningWaveData (whnb,rho) x = do
  let y = map (/sqrt rho) $ fir whnb $ toList (gwdata x)
  fromJust $ updateWaveDatagwdata x $ fromList y
```

https://github.com/gw-analysis/detector-characterization/blob/master/HasKAL/src/HasKAL/SignalProcessingUtils/LinearPrediction.hs

Line removal :
https://github.com/gw-analysis/detector-characterization/tree/master/HasKAL/src/HasKAL/LineUtils/LineRemoval

As a wrapper of KAGALI

FIR, IIR Filter
by Ueno

Functio

ImpulseResponse.c File Reference

Functions of FIR and IIR filters. More...

```
#include <kagali/KGLStdlib.h>
#include <kagali/ImpulseResponse.h>
```

Go to the source code of this file.

Functions

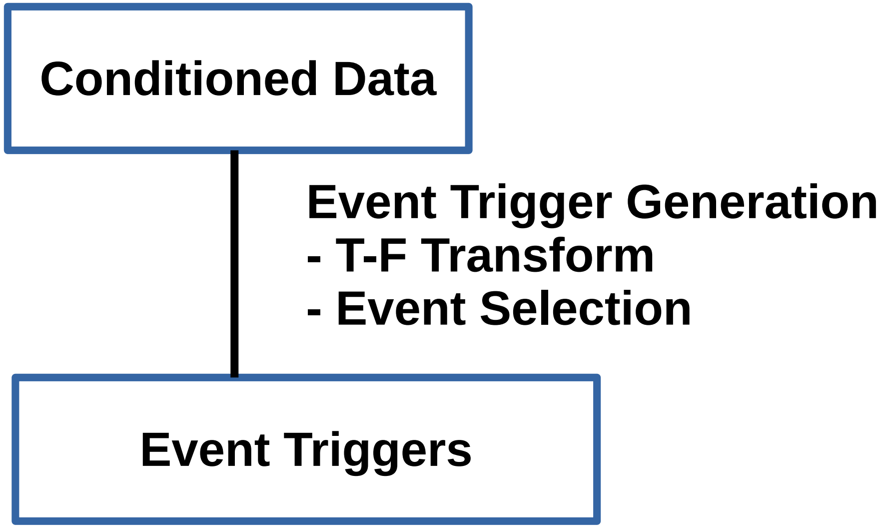
void	**KGLFIRFilterCore** (**KGLStatus** *status, double *output, double *input, unsigned inputlen, double fir_coeff[], double fir_buffer[], unsigned *indexn, unsigned nKernel)
void	**KGLFIRFilter** (**KGLStatus** *status, double *output, double *input, unsigned inputlen, double fir_coeff[], unsigned nKernel)
void	**KGLIIRFilterCore** (**KGLStatus** *status, double *output, double *input, unsigned inputlen, double num_coeff[], double denom_coeff[], unsigned nKernel, double init_coeff[])
void	**KGLIIRFilter** (**KGLStatus** *status, double *output, double *input, unsigned inputlen, double num_coeff[], double denom_coeff[], unsigned nKernel)

Parallelization in Haskell

- **If you replace computeS to computeP, then codes are parallelized!**

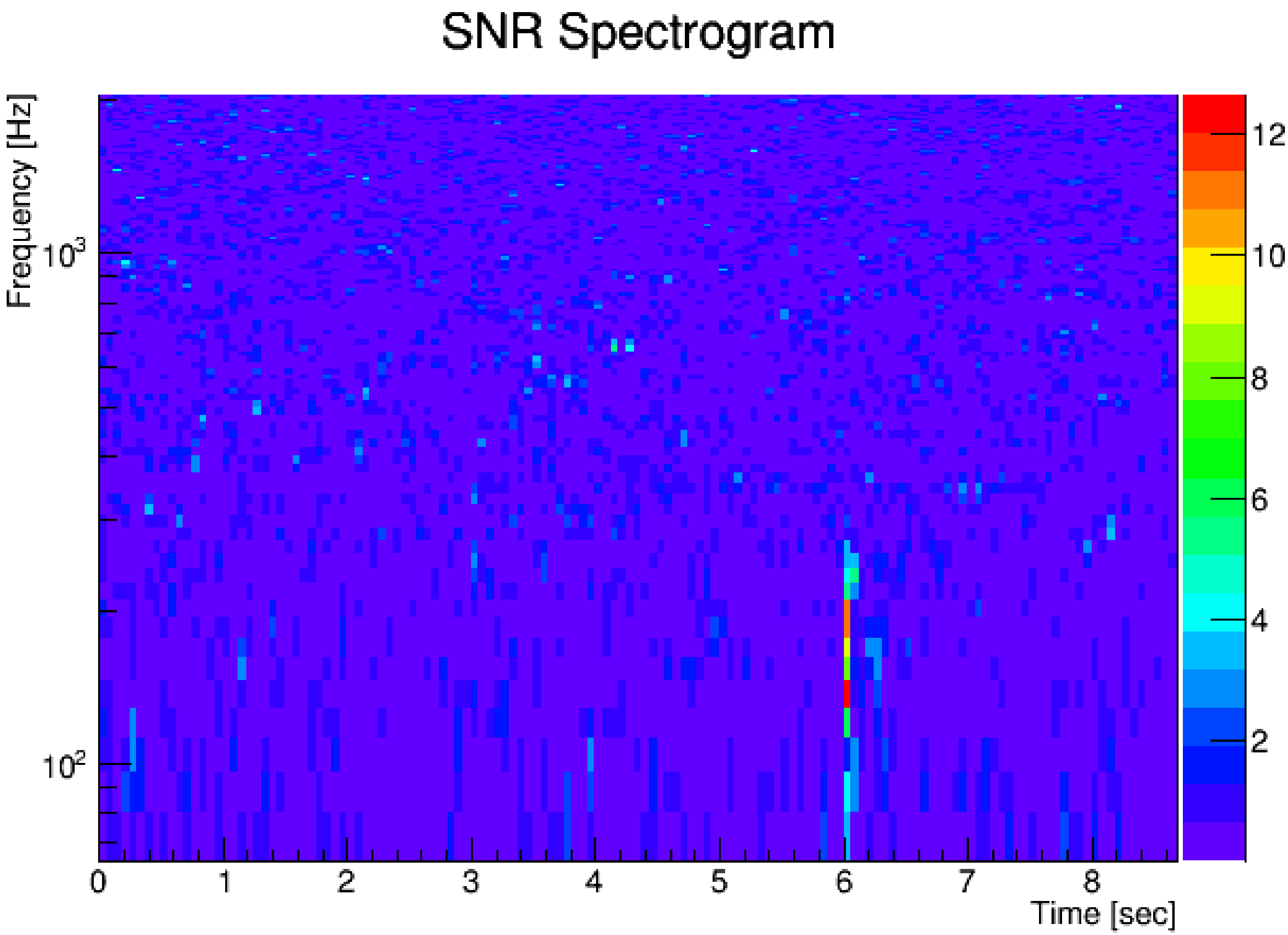
```haskell
fir'0repa :: Repa.Array Repa.U Repa.DIM1 Double -> Repa.Array Repa.U Repa.DIM1 Double -> Repa.Array Repa.U Repa.DIM1 Double ->
fir'0repa rh rw rx
  | rx == fromListUnboxed (Z:.(1::Int)) [] = do
    y' <- computeP (rh *^ rw) :: IO (Array U DIM1 Double)
    sumy <- sumAllP y'
    return (fromListUnboxed (Z :.(1::Int)) [sumy])  :: IO (Array U DIM1 Double)
  | otherwise = do
    y' <- computeP (rh *^ rw) :: IO (Array U DIM1 Double)
    let y = (fromListUnboxed (Z :.(1::Int)) [sumAllS y']) :: Array U DIM1 Double
    rw' <- computeP (extract (Z:.(0::Int)) (Z:.(1::Int)) rx
      Repa.++ (extract (Z :.(1::Int)) (Z :.(size (extent rw)-1)) rw)) :: IO (Array U DIM1 Double)
    rx' <- computeP (extract (Z:.(1::Int)) (Z:.(size (extent rx)-1)) rx) :: IO (Array U DIM1 Double)
    output <- fir'0repa rh rw' rx' :: IO (Array U DIM1 Double)
    computeP (y Repa.++ output) :: IO (Array U DIM1 Double)
```

Event Trigger Generation



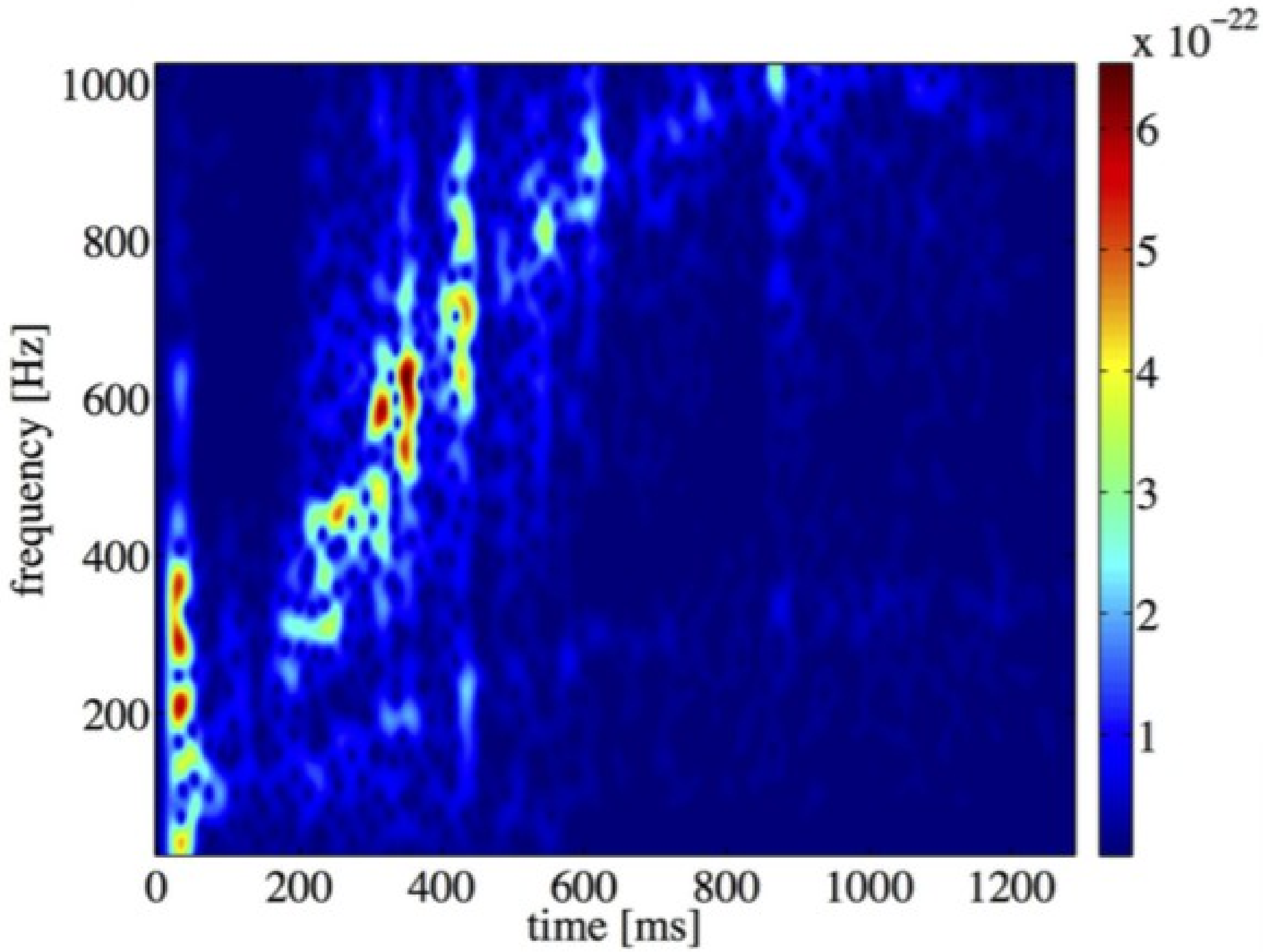
- Excess power based method to find signal
  - Detection on Time-Frequency maps
    - Short Fourier Transform (Hayama)
    - Constant Q-Transform (Hayama)
    - Wavelet Packet Transform (Yokozawa)
  - Event Selection
    - Pixel clustering
    - Other method needed!

Event Trigger Generation

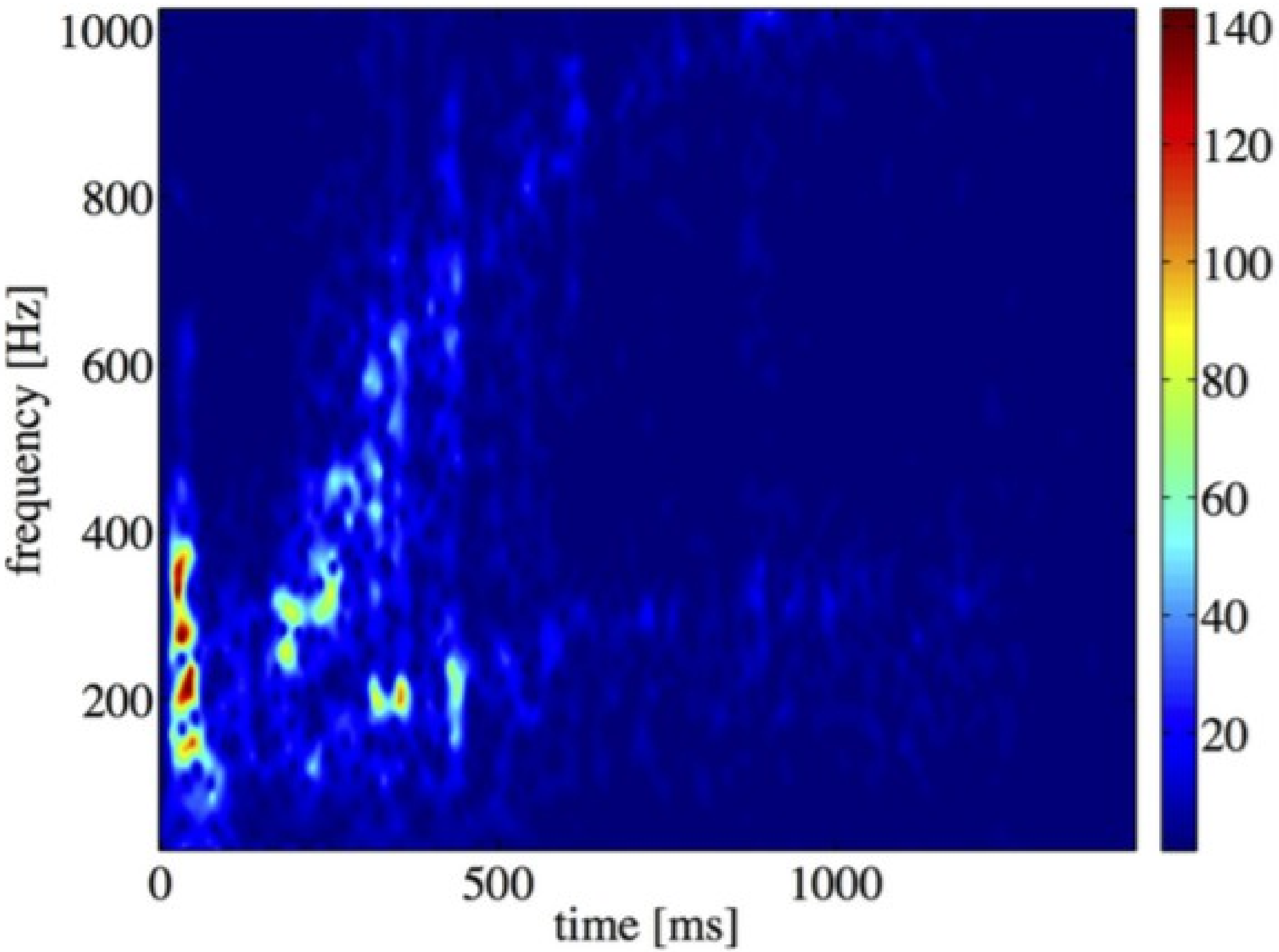


Event Selection: Simple clustering method may not work

Injection



SNR TF plot

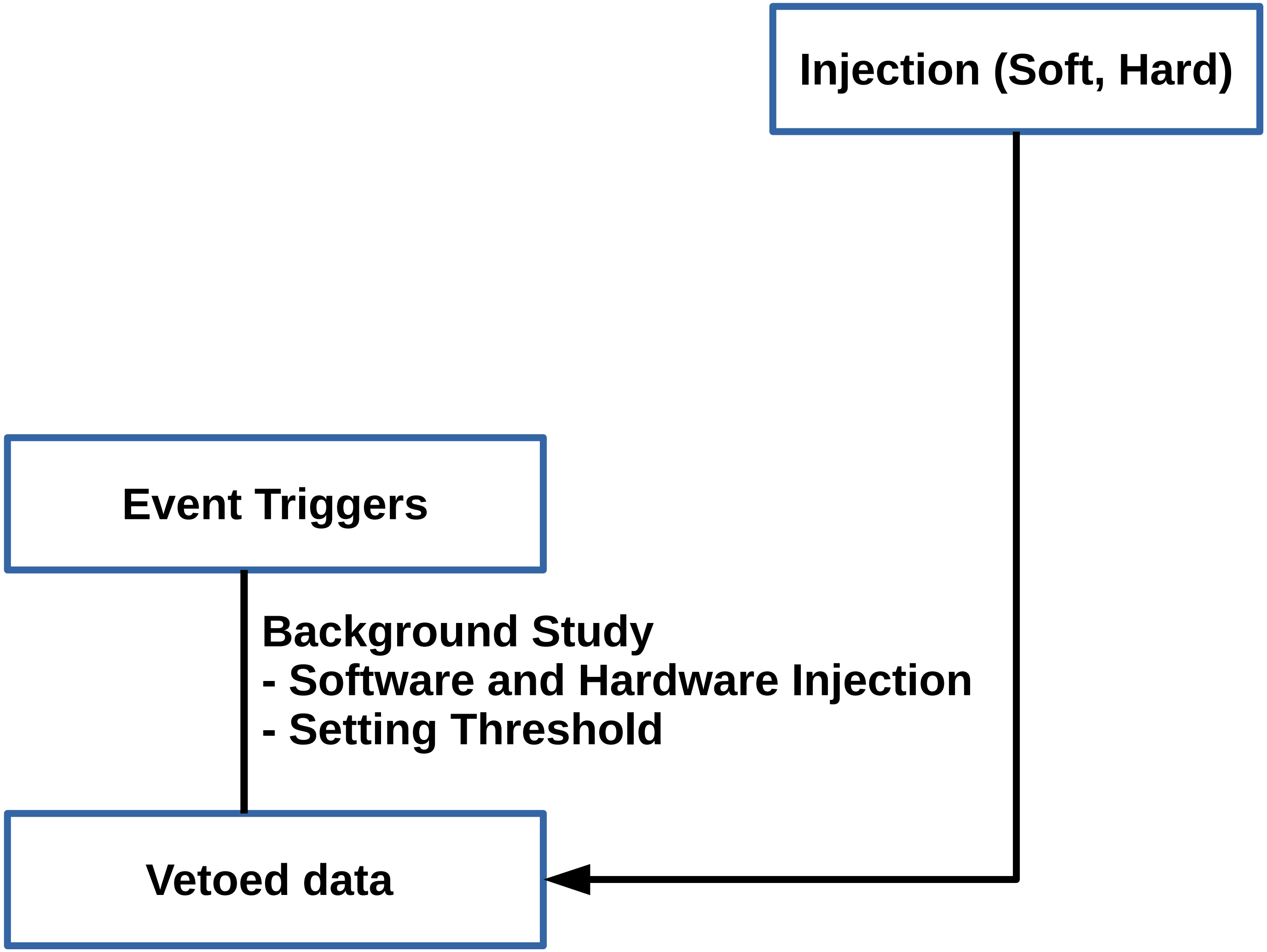


1000
800
600
400
200
frequency [Hz]
0
500
1000
time [ms]
140
120
100
80
60
40
20

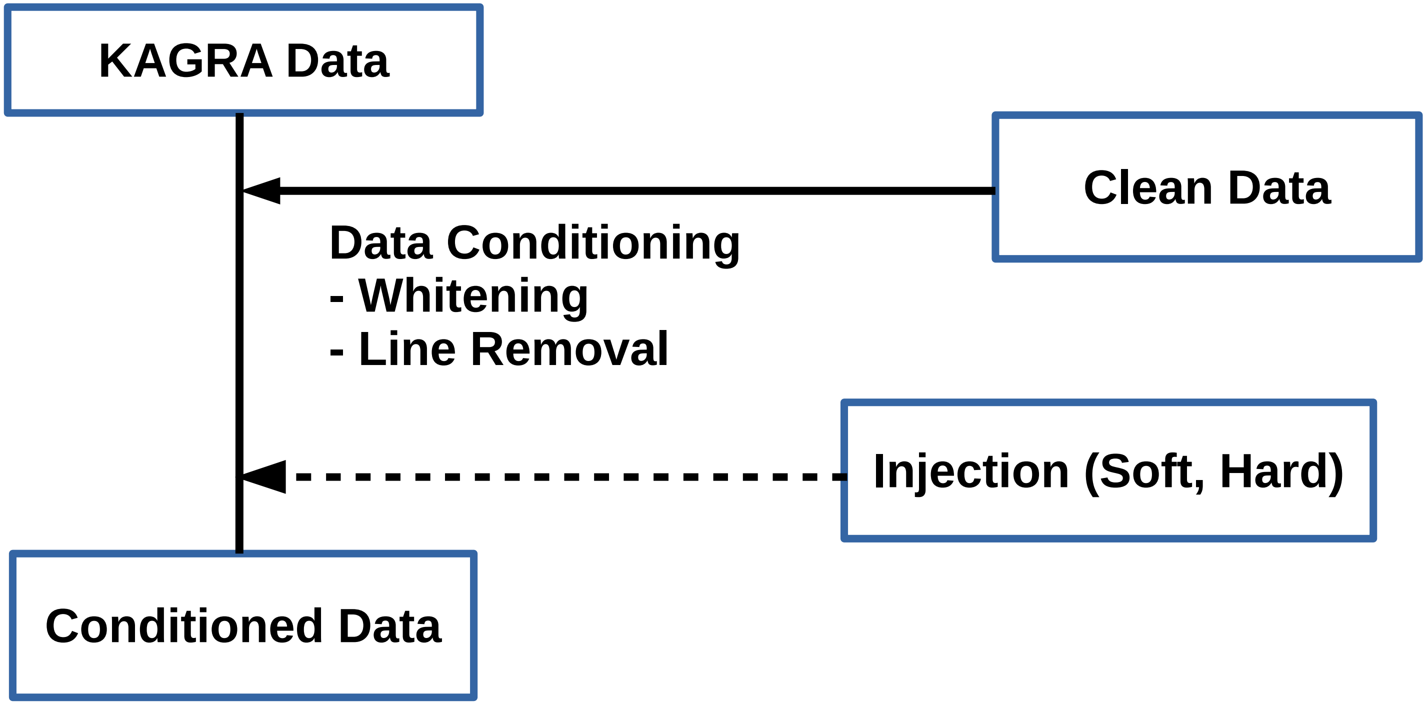
The software

```
t13 <- getCurrentTime
print "{- Time-Frequency SNR Map -}"
let noverlap = 0 :: Int
    nfreq = 256 :: Int
    ntime = 140 :: Int
    fs2 = floor $ ((fromIntegral nfreq)/2) :: Int

    nrefset = 100 :: Int
    refpsd = snd $ gwpsdV (subVector 0 (nfreq*nrefset) (gwdata whnWaveData)) nfreq fs
    refpsd2 = scale sigma $ subVector 0 fs2 refpsd
    refpsd2s= subVector 0 fs2 refpsd

HR.plot HR.LogXY HR.Line 1 HR.RED ("frequency",  "Spectrum") 0.05 "ref psd" "testburst_refpsd.png" ((0, 0), (0, 0))
  $ zip [0..] (toList refpsd2)
  -- todo : functionalization
let snrMatF = scale (fs/fromIntegral nfreq) $ linspace nfreq (0, fromIntegral nfreq)
    snrMatT = scale ((fromIntegral nfreq)/fs) $ fromList [0.0, 1.0..(fromIntegral ntime -1)]
    snrMatP = fromColumns
      $ map (\i->zipVectorWith (/)
      (
      subVector 0 fs2 $ snd $ gwpsdV (subVector (nfreq*i) nfreq (gwdata whnWaveData)) nfreq fs)
      refpsd2
      ) [0..ntime-1]
    snrMat = (snrMatT, snrMatF, snrMatP)
    nrow = rows snrMatP
    ncol = cols snrMatP
t14 <- getCurrentTime
print $ diffUTCTime t14 t13
```

Background Study



Injection



- **Injection (Hayama)**
  To estimate background noise and set detection threshold.

  - **Software Injection**
    Currently implemented using S5 burstMDC
  - **Hardware Injection**

    To be discussed with DGS group (?)

The Software

```
data SOURCE_TYPE = SOURCE_TYPE
  { sigType :: SigType
  , longitude :: Double
  , latitude :: Double
  , psi :: Double
  , fs :: Double
  , hrss :: Double
  } deriving (Show, Eq)
```

```
injDetectorResponse :: Detector -> SOURCE_TYPE -> GPSTIME -> WaveData
injDetectorResponse detName srcType gps = do
  let detparam
        | detName == LIGO_Hanford = ligoHanford
        | detName == LIGO_Livingston = ligoLivingston
        | detName == KAGRA = kagra
        | otherwise = error "not recognized"

      (antennaPattern, tauS) =
        fplusfcrossts detparam (longitude srcType) (latitude srcType) (psi srcType)

      detresp = genDetectorResponse antennaPattern $ getPolarizations srcType

      startGPSTime' = fromIntegral (fst gps) + 1E-9 * fromIntegral (snd gps) + tauS
  WaveData { detector = detName
           , dataType = "SoftwareInjection"
           , samplingFrequency = fs srcType
           , startGPSTime = formatGPS startGPSTime'
           , stopGPSTime  = formatGPS $ startGPSTime'+(fromIntegral (dim detresp)-1)/(fs srcType)
           , gwdata = detresp
           }
```

- **https://github.com/gw-analysis/detector-characterization/tree/master/HasKAL/src/HasKAL/SimulationUtils/Injection**

Calculation of antenna pattern (Hayama)

```haskell
fplusfcrossts :: DetectorParam -> Double -> Double -> Double -> (AntennaPattern, Double)
fplusfcrossts detname phi theta psi = do
  -- Detector Tensor
  let d = calcd (tuple2mat (deta detname)) (tuple2mat (detb detname))
      rr = rz (90+psi) <> ry (90-theta) <> rz (phi)
      dtensor = rr <> d <> trans rr
      fplus = (dtensor @@> (0, 0) - dtensor @@> (1, 1)) / 2.0
      fcross= -(dtensor @@> (0, 1) + dtensor @@> (1, 0)) / 2.0
      tau = (tuple2mat (detr detname) <> (unitVector phi theta) / (scalar speedofLight)) @@> (0, 0)
  ((fplus, fcross), tau)
```

```haskell
data DetectorParam =
  DetectorParam { name :: Detector
                , detr :: (Double, Double, Double)
                , deta :: (Double, Double, Double)
                , detb :: (Double, Double, Double)
                } deriving (Show)

ligoHanford :: DetectorParam
ligoHanford = DetectorParam { name = LIGO_Hanford
                            , detr = (-2.161414928E6, -3.834695183E6, 4.600350224E6)
                            , deta = (-0.223891216, 0.799830697, 0.556905359)
                            , detb = (-0.913978490, 0.026095321, -0.404922650)
                            }
```

https://github.com/gw-analysis/detector-characterization/tree/master/HasKAL/src/HasKAL/DetectorUtils

Parameter Estimation

- **Not yet implemented**
- **One of featured parameter estimation of the burst analysis in KAGRA is Hilbert-Huang Transform. (Kaneyama)**

Alert

- **Not yet determined.**
- **VOEvent is one of good candidates.**
  - **Duration, Power, Frequency, Waveform, Sky region**