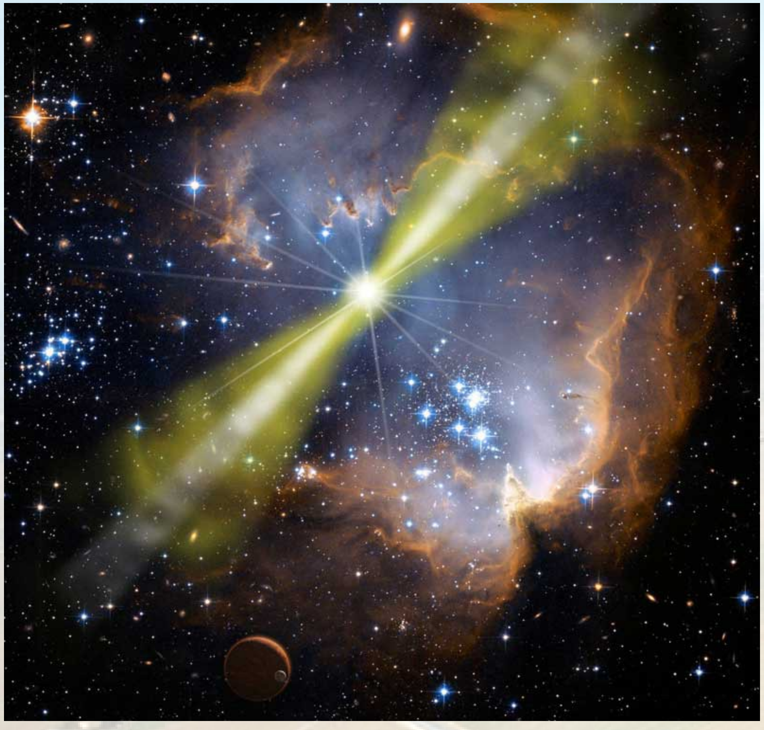
On the possibility of a GRB forecasting algorithm and alert system for future gravitational wave detectors

Gergely Debreczeni

Wigner Research Institute for Physics
Budapes,

(Debreczeni.Gergely@wigner.mta.hu)

Content



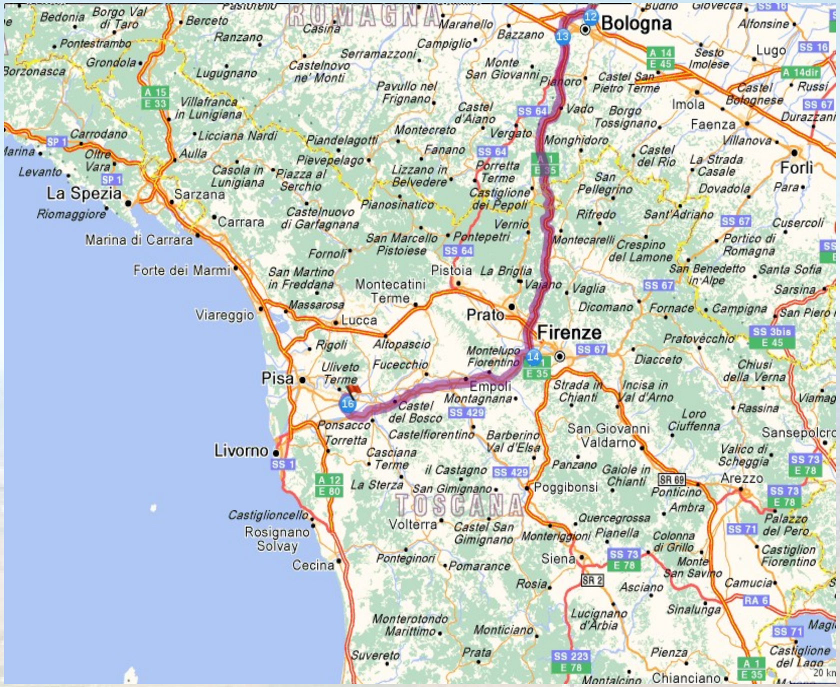
Artists's view of a gramma-ray burst

Credit: NASA/Swift/Mary Pat Hrybyk-Keith and John Jones

- **Gravitational wave detectors**
  - Short introduction of modern and next generation gravitational wave detectors
- **Binary neutron star coalescnce and gramma-ray bursts**
  - Types of GRB, production mechanisms
- **Scientific potential**
  - Open questions to be answered by this research
- **Requirements and chances**
  - What requirements have to be met and what are the chances of such observations
- **Expected precision**
  - Precision of the forecasting algorithm: arrival time and sky localisation
- **The Compute Backend tool - Wigner GPU Laboratory**
  - Universal GPU programing interface
- **Future plans**
  - Next steps, research drections, problems to face

The Virgo experiment

(as a possible future use-case)



- The Virgo detector is located in the site of the European Gravitational Observatory (EGO) in Cascina, near Pisa, Italy.

- Construction finished in 2003

- It is now a european collaboration including France, Italy, Hungary, Netherland, Poland

- Working together with LIGO (Laser Interferometer Gravitational-wave Observatory), synchronized observations and coordinated analysis

- So far, approixmately c.c 20 month of data taking

- Currently under upgrade, will start to collect scientific data in late 2016

About GRBs...

- Extremely energetic electromagnetic events.
- Can last from 10s of ms to minutes
- GRBs with duration < 2 s classified as short-GRBs
- The so called short GRBs believed to originate from merging binary neutron stars
- Inital burst usually followed by afterglow in longer wavelengths including optical and radio
- First afterglow observered in 1997
- Events are very far away, closest measeured-z short GRB had a distance of 33 Mpc
- Many satellite is designed for this purpose, BeppoSAX, HETE, Swift, Fermi, ...
- Gamma-Ray Burst Coordinate Network for coordinated observations
- On average 1 GRB per day
- Observing GRBs in the gravitational wave channel was just a preparatory exercise so far, since the sensitivity (horizon distance) of GW detectors was no big enough
- For advanced detectors with 200-400 Mpc horizon distance it we expect muxh more event inside the sensitivity distance
- Opening angle of the GRB's beam estimated between 2 and 20 degree

...and event rates.

Epoch	Run Duration (months)	Burst range (LIGO) (Mpc)	Burst range (Virgo) (Mpc)	BNS range (LIGO) (Mpc)	BNS range (Virgo) (Mpc)	Number of BNS detections	% localized BNS soruces (5 deg)	% localize d BNS soruces (20 deg)
2015	3	40-60	-	40-80	-	0.0004-3	-	
2016-17	6	60-75	20-40	80-120	20-60	0.006-20	1-2	5-12
2017-18	9	75-90	40-50	120-170	60-85	0.04-100	1-2	10-12
2019+	(per year)	105	40-80	200	65-130	0.2-200	3-8	8-28
2022+ (India)	(per year)	105	80	200	130	0.4-400	17	48

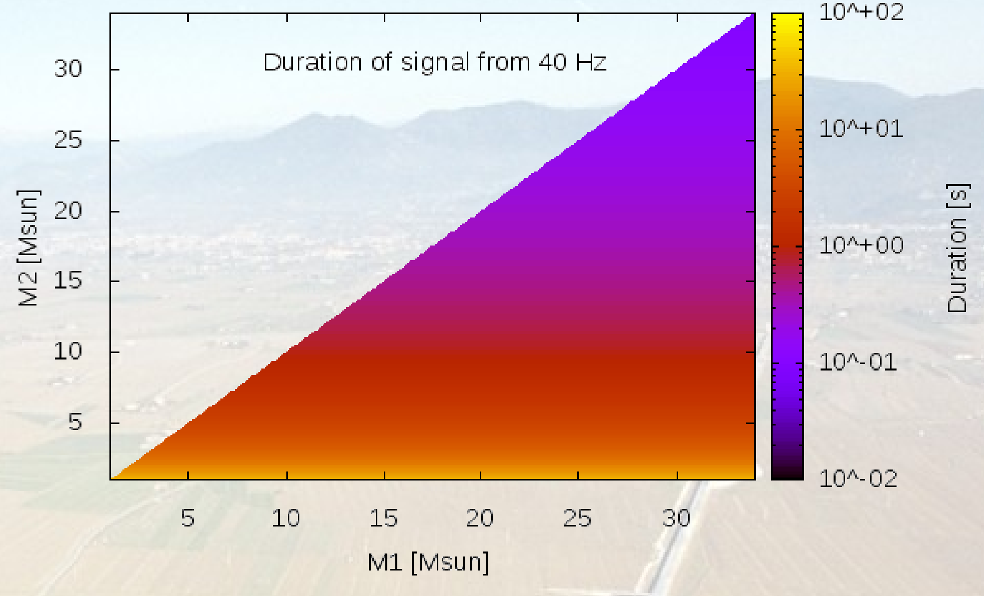
Questions to be answered
- that could be answered

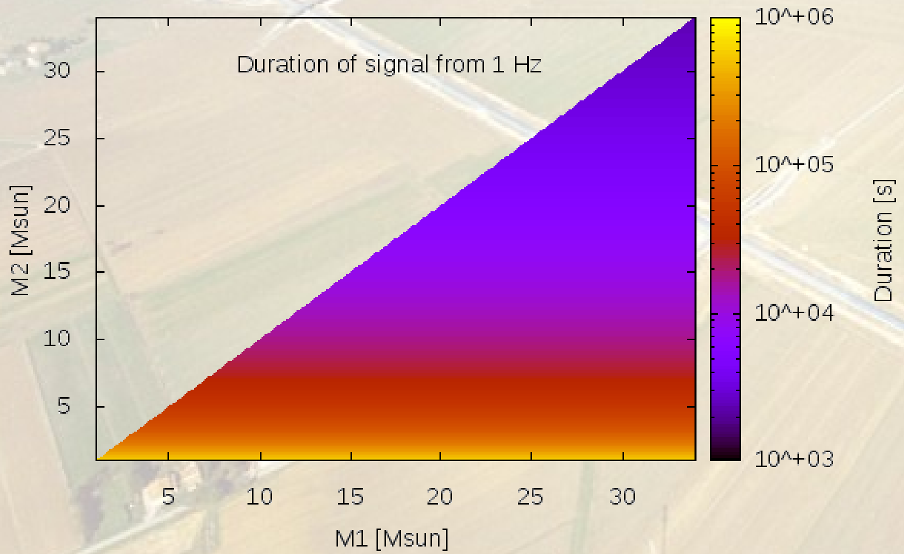
- **What fraction of binary neutron star coalescence ends as gamma-ray burst ? -** Requires some statistics, because of the focused features of the beam (hidden events), but a very important question.
- **What is the mass distribution of GRBs? -** As there is no direct observation of gamma ray burst sources in gravitational wave channel so far, this could be one of the first thing to be answered.
- **What is the precision and the quality of our understanding of the merger phase ? -**How well the extrpolated post-Newtonian evolution or the numerical simulation can predict the merging time.
- **How strictly the arrival time of the gravitational wave and the EM signal corresponds ? -** GW and EM waves supposed to travel with same speed... Easy use-case to test.
- **What is the opening angle of the EM beam ? -** Once we answered the above questions we can also measure this infromation. DIfficult, requires the approximate estimation of orbital plane.
- **Is there any property of the afterfglow which depends on other parameters of the system ? -** If we are able to observe the early phase of afterglow it could bring additional infromation.
- **What kind of additional physics can be extracted if we know the arrival time in advance ? -** Being able to perform prepared, targeted observations could enhance a lot our understanding of the process and contribute with high quality data.

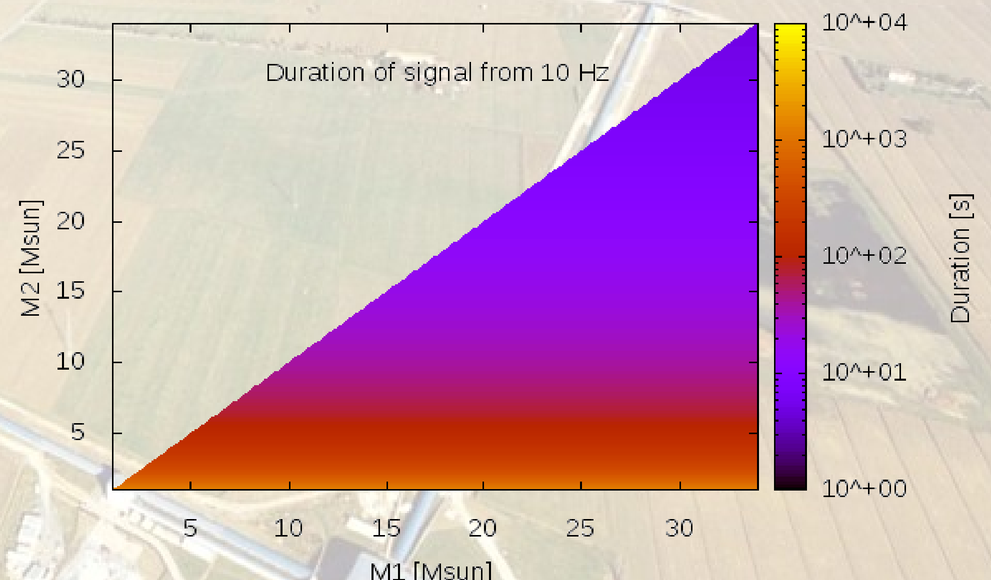
Signal durations

$$T(f_{low}) = \frac{5}{256\eta}\frac{GM}{c^3}\left[v_{low}^{-8} + \left(\frac{743}{252} + \frac{11}{3}\eta\right)v_{low}^{-6} - \frac{32\pi}{5}v_{low}^{-5} + \left(\frac{3058673}{508032} + \frac{5429}{504}\eta + \frac{617}{72}\eta^2\right)v_{low}^{-4}\right]$$

- Longest signal for current detectors (40 Hz lower freq. cutoff c.c 44 sec) - **not sufficient**
- Late-advanced detectors (10 Hz lower freq cutoff c.c 1500 sec) ~ **very good**
- Einstein Telescope (1 Hz lower freq cutoff, very long !)
- At least 60 second preparation time is necessary for EM telescopes (the longer the better) !

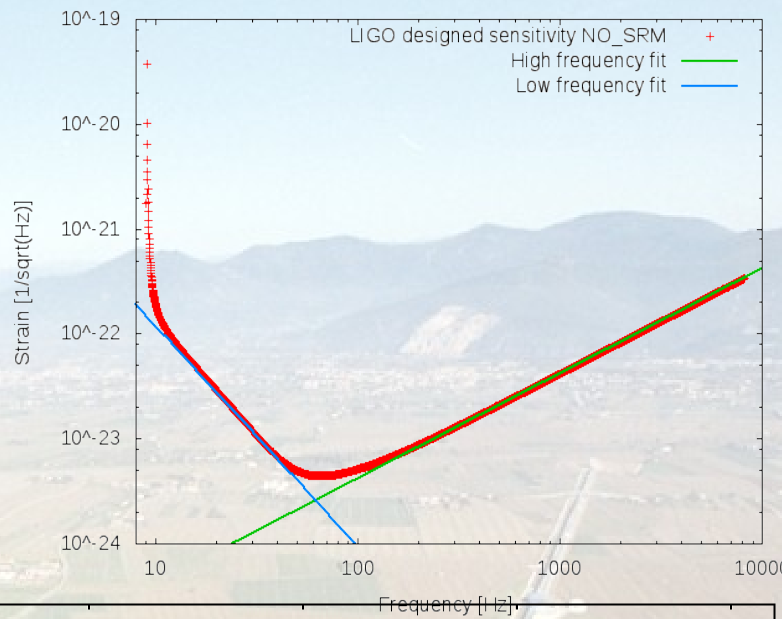


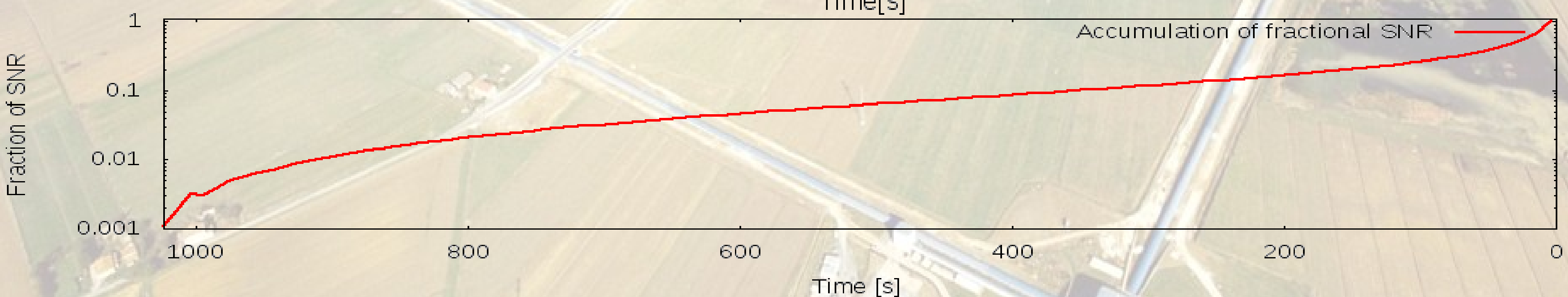




Accumulation of signal-to-noise ratio

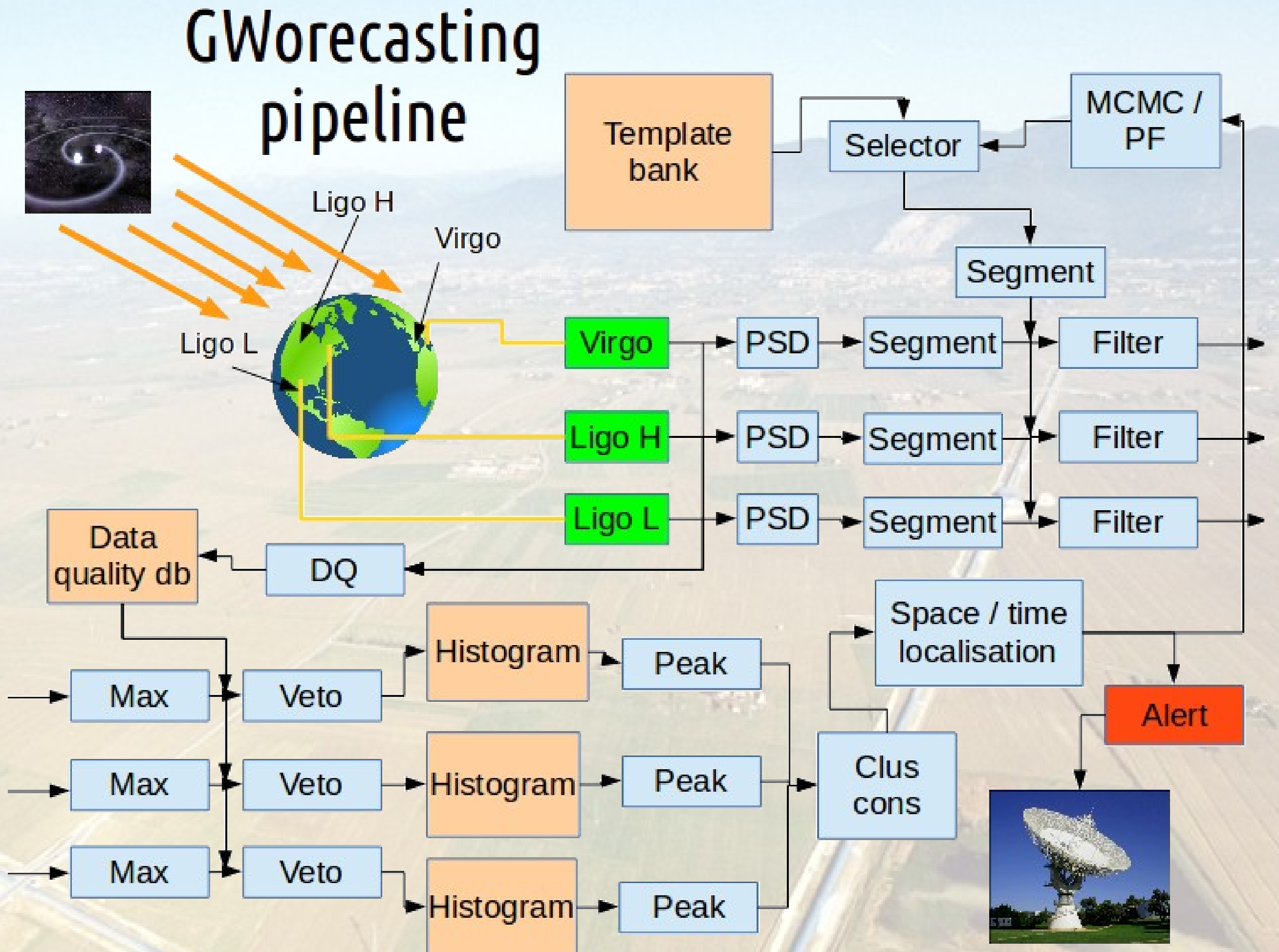
- Detector has strongly varying sensitivity as the function of frequency
- Sufficient SNR has to be accumulated before any trigger or alert can be generated
- Only scenarios where the final SNR is over 8 is considered.
- Chirp signal's spectral amplitude goes as f^(-7/6).
- Has to be weighted with the sensitivity curve.





The algorithm - the complete system

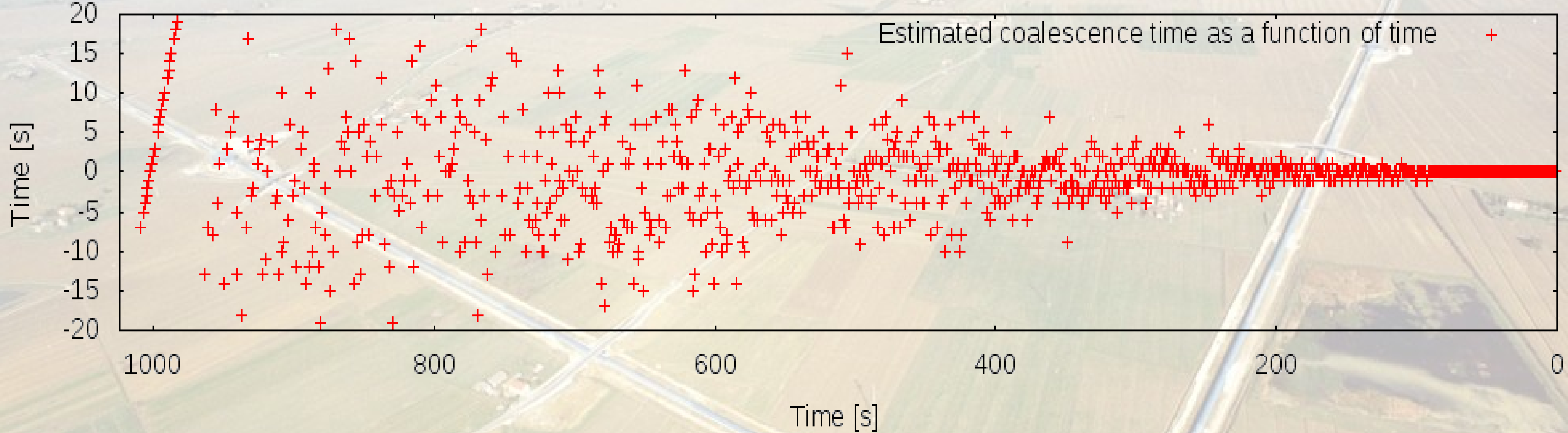
- Performs early detection of 'wavelets', i.e. incomplete wave chunks.
- Matches over the threshold are histogrammed in multi-dimensional parameter space
- Wihen consisten accumulation of peak is detected a **trigger** is generated
- Arrival time can be deduced from a single detector, however
- it is necessary to handle triggers from multiple detectors for sky localisation
- Multi detector parameter matching and enviromental noise crosscheck have to be performed before sending an **alert.**



Initial result - arrival time estimation

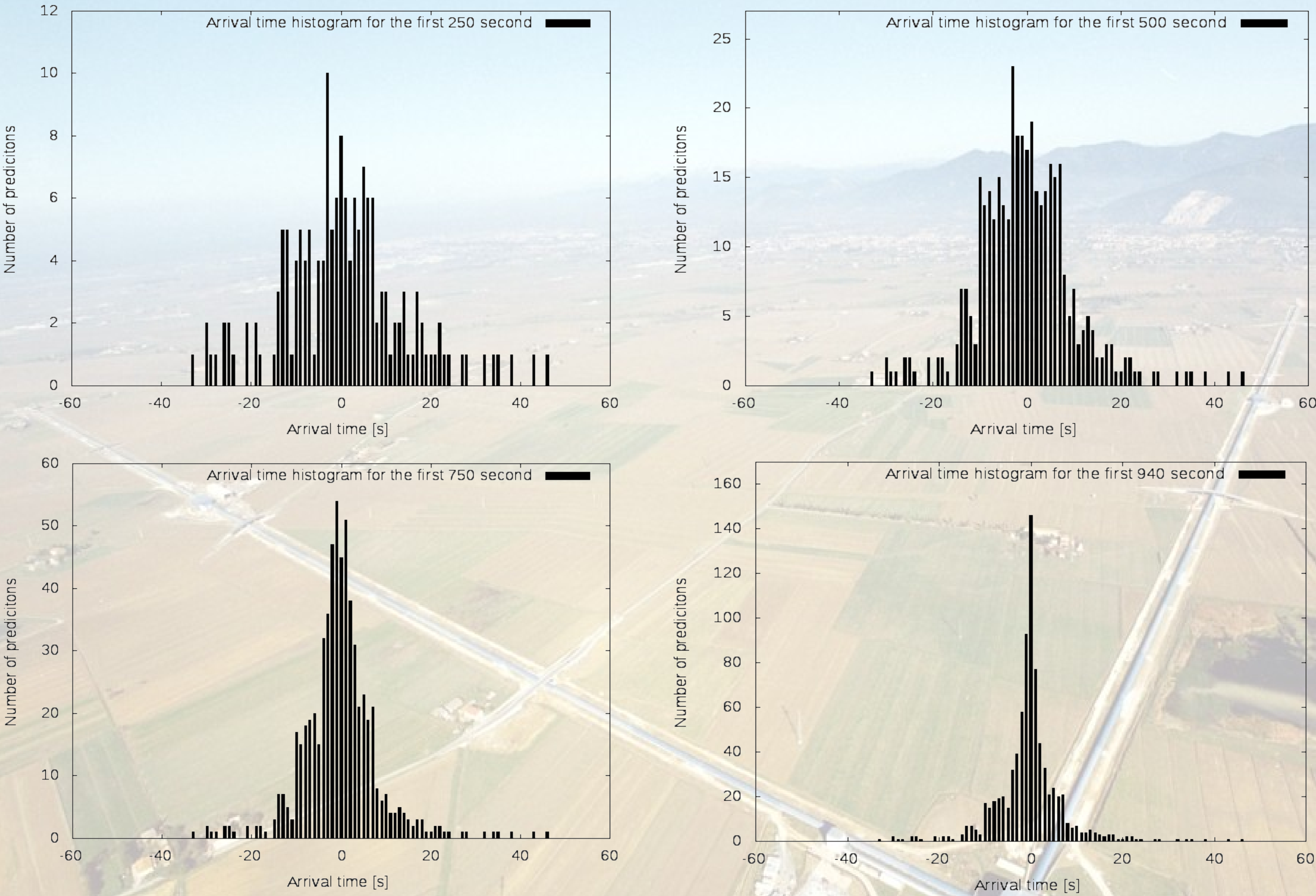
- Performs early detection of 'wavelets', i.e. incomplete wave chunks.
- Matches over the threshold are histogrammed in multi-dimensional parameter space
- Wihen consisten accumulation of peak is detected a **trigger** is generated

- Arrival time can be deduced from a single detector, however
- it is necessary to handle triggers from multiple detectors for sky localisation
- Multi detector parameter matching and enviromental noise crosscheck have to be performed before sending an **alert.**



An example for an inspiraling 1.4 - 1.4 Msun binary system entering into sensitivity band ar 10 Hz. Estimated arrival times as a function of time when the data is queried with only one template.

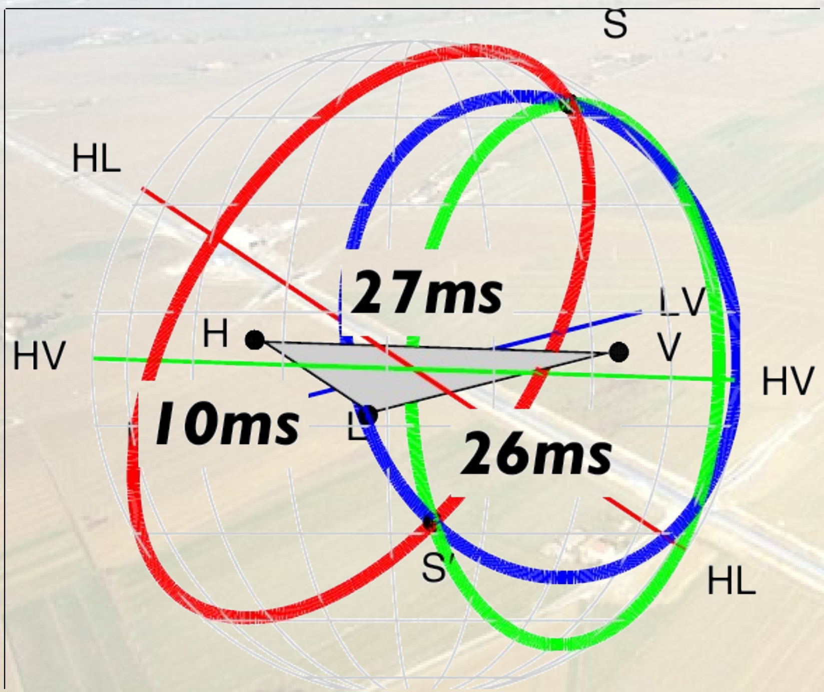
Initial result - arrival time histograms

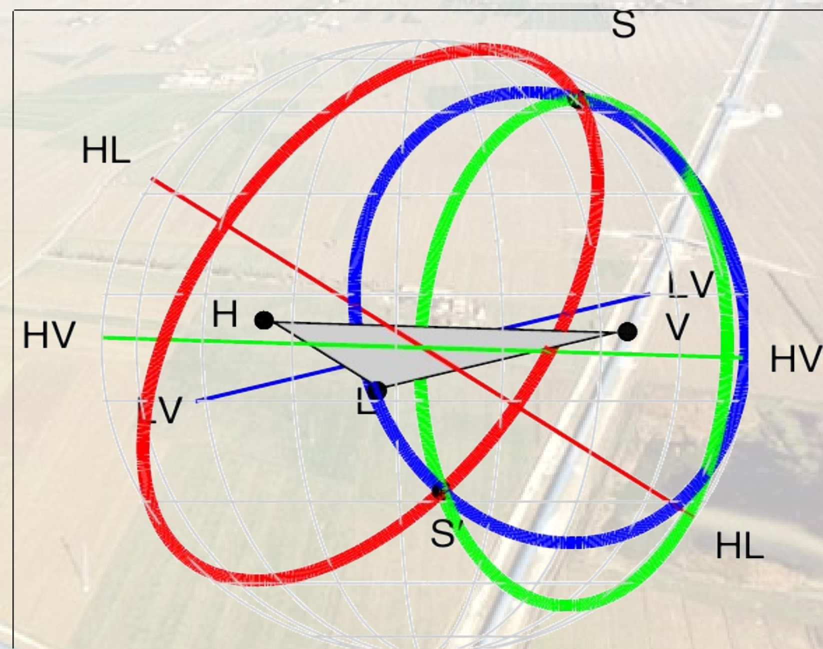


Initial result - sky localisation

- Sky localisation from timing infromation only is doable but migh not be as accurate as desired
- Two detector pair spans a circle in the sky.
- The circle width depends on detector basline, frequency resolution, sampling rate

- Sub 10 ms timing resolution is necessary for rough sky localisation
- This is achievable only very close to coalescence
- Can be improved with higher sampling rate or amplitude / phase consistency between sites



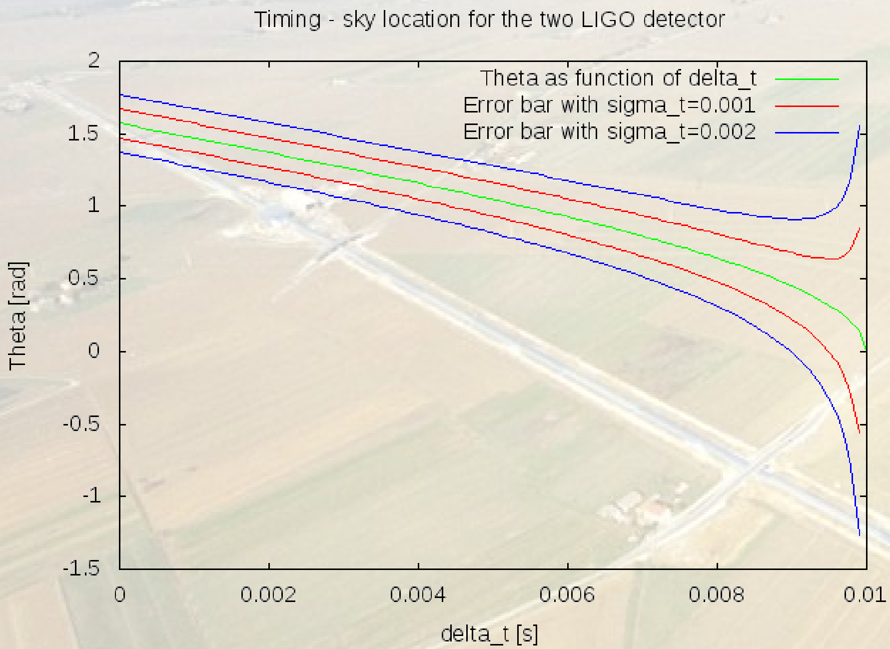


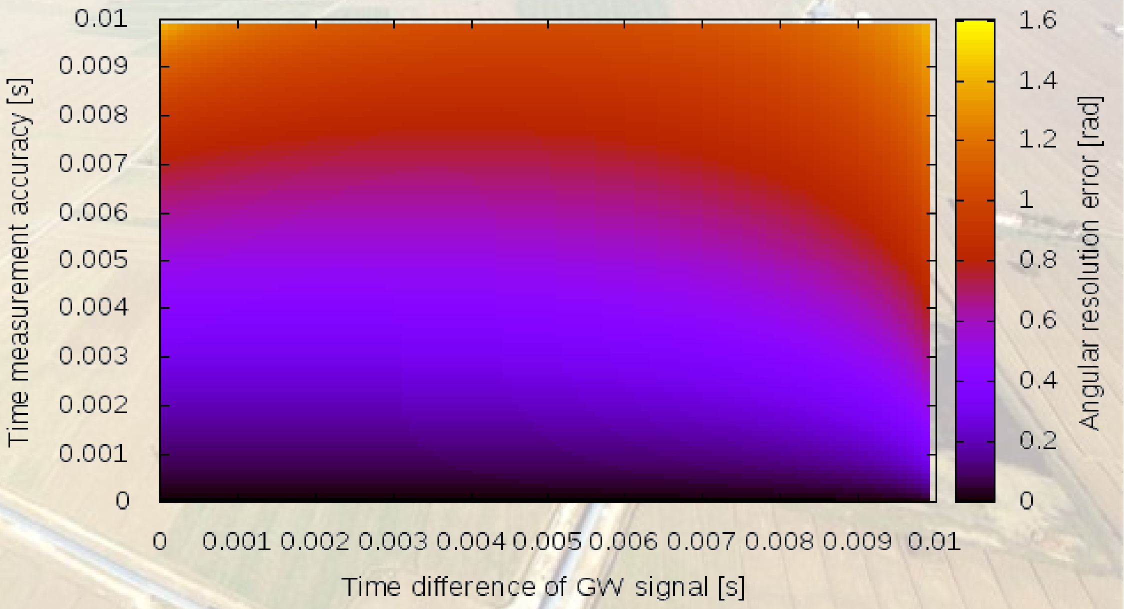
Images courstesy of S.F. NJP 2009, CQG 2011

Initial result - sky localisation

- H and L Ligo detectors are d = 3000 km away
- 10 ms time-of-flight
- A simplified, one-dimensional time difference - sky position plot using interaural equation is shown below.
- cos(theta) = delta_t * c / d

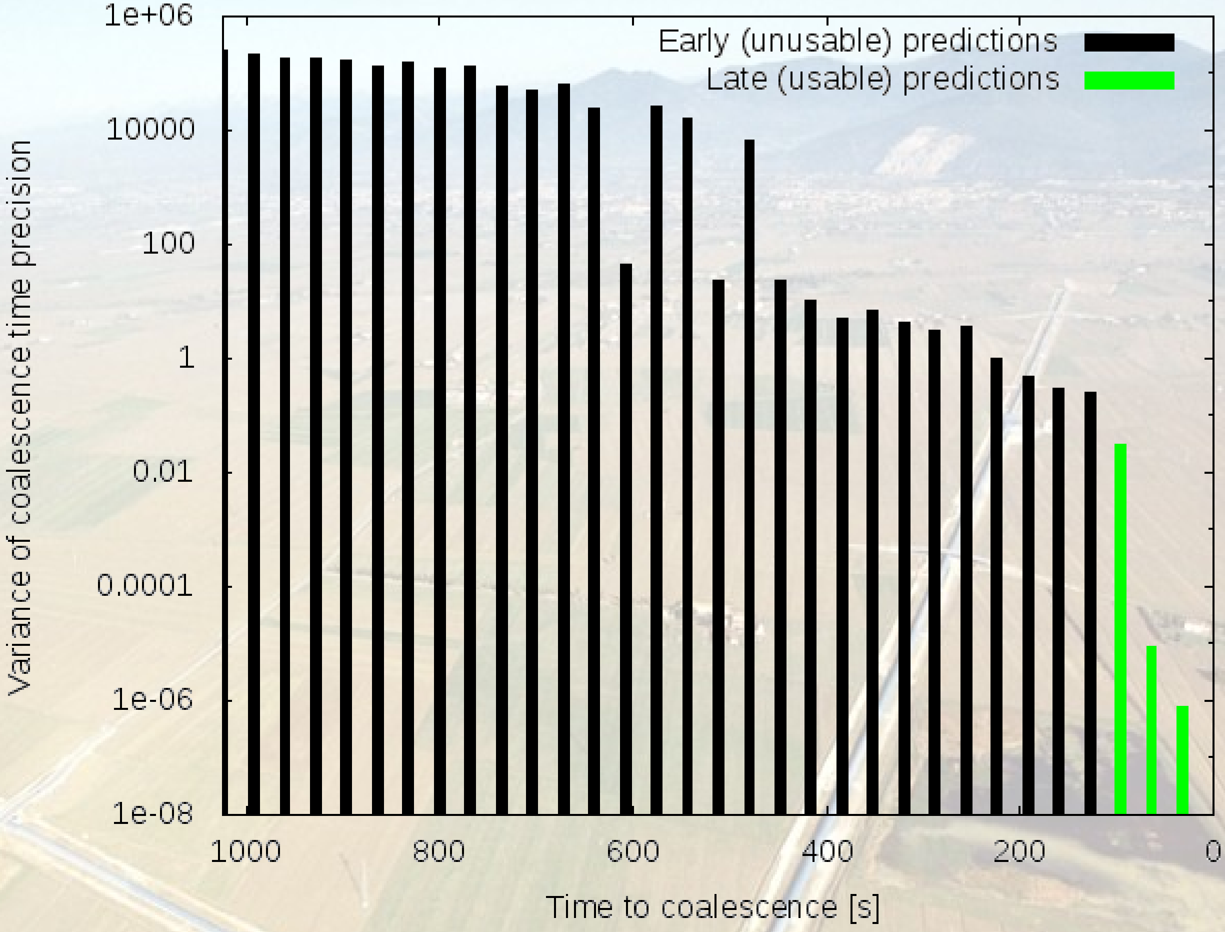
- It is seen that even small timing errors causes large localization problems
- Sky location information will be available in the last moments...
- For an octant coverage of sky one needs 2 ms timing precision....





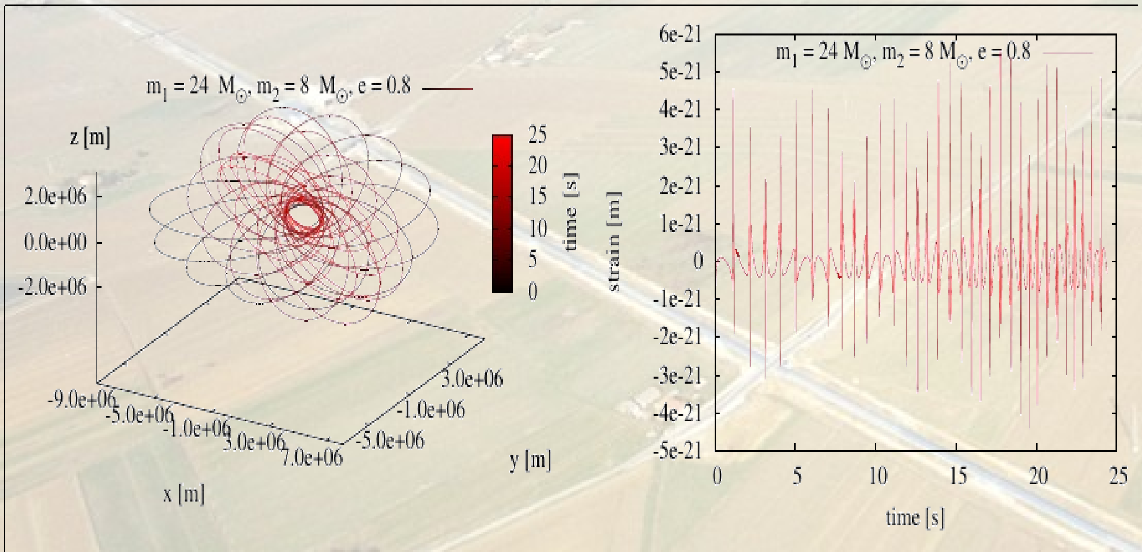
Initial result - sky localisation

- Only the very last minute will give sky location information
- Must be improved
- With the current algorithm there will be no time to re-position telescopes but instead, an alert should be sent only to the ones which looks in the right directions.
- Requires bi-directional communication



Next step: orbital plane estimation (?)

- Splinless binaries have no precession, orbital plane orientation contributes with a constant factor to the amplitude
- Binaries with high spin have very strong dynamic at merger, difficult to say anything about final orientation...
- Small spin, slowly precessing binaries could be used for orbital plane estimation



Orbital precession and waveform mudulation of a strongly eccentric, double spinning binary system

Computational and coordinational challanges

- Several millions of template have to be matched if not restricting to binary NSs with low masses.
- FFT, vector multiplication, maximum finding, clustering can be efficiently performed on GPUs - typical gain: x80-100.
- In case of faint signals - for some parameter space - , using only phase and dropping amplitude information there can be faster methods than FFT
- Reliable communication channel have to be maintained
- ROC (Receiver Operating Characteristic) curves of filters should include operational and manpower costs

The Compute Backend (CB)

The problem

- For several reason (cost efficiency, manpower, future hardwares, etc..) the analysis code has to be generic
- It is always a subject of debate which language to use to program GPUs.
- Double coding for multiple interface is a waste of time and manpower.

The solution:

- THE **COMPUTE BACKEND** (CB) IS ADDRESSING THIS PROBLEM BY PROVIDING UNIFIED INTERFACE FOR VARIOUS GPU PROGRAMING LANGUAGES, SUCH AS CUDA AND OPENCL !
- It levreages the burden of host-side double coding and the very same code can be used to run on CUDA (NVidia) or OpenCL (AMD, Intel, Samsung, etc...) devices...

Compute Backend (CB) features:

- C and C++ API (fortran, python and c# on the way...)
- CUDA and OpenCL backends (ComputeGl, RenderScript considered)
- Single host-side code for multiple backend
- Runs under Linux/Windows/MacOS
- Compatible with CMake, Autoconf, MSVC, etc.
- Academic license is available
- User support around the clock

Available at: http://gpu.wigner.mta.hu

The Compute Backend - the C API

```c
#include <stdio.h>
#include <stdlib.h>
#include <cb.h>

int main() {

    // Auxiliary variables
    int err ;
    int i;

    // Sets the log level
    cb_log_level = 5;

    // Get some buffer
    unsigned int num_elements = 1024;
    unsigned int size = num_elements * sizeof(float);

    // ... and also on the host side
    float * h_buffer1 = (float *) malloc(size);
    float * h_buffer2 = (float *) malloc(size);
    float * h_buffer3 = (float *) malloc(size);

    // ... fill up the buffers
    for (i = 0; i < num_elements; i++) {h_buffer1[i] = 4; h_buffer2[i] = 11;}

    // The C API
    // A compute backend
    cb_backend   backend;
    cb_program   prog;
    cb_kernel    kernel1, kernel2, kernel3;
    cb_buffer    buffer1, buffer2, buffer3;

    // Get the compute backend
    err = cbGetComputeBackend(&backend);

    // Get a program
    err = cbGetProgram(&backend, "/home/me/testt", &prog);

    // Get the kernel
    err = cbGetKernel(&prog, "test_kernel",    &kernel1);
    err = cbGetKernel(&prog, "simple_kernel",  &kernel2);
    err = cbGetKernel(&prog, "buffer_kernel",  &kernel3);
```

```c
    err = cbCreateBuffer(&backend, CB_READ_WRITE, size, NULL, &buffer1);
    err = cbCreateBuffer(&backend, CB_READ_WRITE, size, NULL, &buffer2);
    err = cbCreateBuffer(&backend, CB_READ_WRITE, size, NULL, &buffer3);


    // Send some data to device
    err = cbWriteBuffer(&backend.queues[0], &buffer1, size, h_buffer1, true);
    err = cbWriteBuffer(&backend.queues[0], &buffer2, size, h_buffer2, true);

    // Set the kernel sizes
    cbExtent g_size = cbSetExtent(1,1024);
    cbExtent l_size  = cbSetExtent(1, 32);

    // Execute the kernel
    cbParam b1_arg = cbBuffer(&buffer1);
    cbParam b2_arg = cbBuffer(&buffer2);
    cbParam b3_arg = cbBuffer(&buffer3);
    cbParam n_arg   = cbInt(100);

    err = cbExecuteKernel(&backend.queues[0], &kernel3, g_size, l_size, 4,
&b1_arg, &b2_arg, &n_arg, &b3_arg);

    // Read back the result
    err = cbReadBuffer(&backend.queues[0], &buffer3, size, h_buffer3, true);

    // Printing the result
    for (i = 0; i < 10; i++) printf("%f ", h_buffer3[i]);
    printf("\n\n");

    // Releasing stuff
    free(h_buffer1);
    free(h_buffer2);
    free(h_buffer3);

    // Exit
    return err;
}
```

The Compute Backend - the C++ API

```
#include <stdio.h>
#include <stdlib.h>
#include <iostream>
#include <cb.hpp>

int main() {

    // Sets the log level
    cb_log_level = 5;
    int err ;
    int i;

    // Get some buffer on the host side
    unsigned int num_elements = 1024;
    unsigned int size = num_elements * sizeof(float);

    float * h_buffer1 = new float[num_elements];
    float * h_buffer2 = new float[num_elements];
    float * h_buffer3 = new float[num_elements];

    // ... fill in the buffers
    for (i = 0; i < num_elements; i++) {h_buffer1[i] = 4; h_buffer2[i] = 11;}

    // Construction Backend, Program, Kernel and Buffers
    cb::Backend bck;
    cb::Program prg(bck, "/home/me/test");
    cb::Kernel  TestKernel(prg, "test_kernel");
    cb::Kernel  SimpleKernel(prg, "simple_kernel");
    cb::Kernel  BufferKernel(prg, "buffer_kernel");

    // Initializing the buffers
    cb::Buffer  b1(bck, CB_READ_WRITE, size, NULL);
    cb::Buffer  b2(bck, CB_READ_WRITE, size, NULL);
    cb::Buffer  b3(bck, CB_READ_WRITE, size, NULL);

    // Send data to device
    b1.Write(bck.GetQueue(), h_buffer1);
    b2.Write(bck.GetQueue(), h_buffer2);

    // Set the kernel sizes
    cb::Extent g(num_elements);
    cb::Extent l(32);
```

```
   // Create kernel arguments
    cbParam buff1_arg = cbBuffer(b1);
    cbParam buff2_arg = cbBuffer(b2);
    cbParam buff3_arg = cbBuffer(b3);
    cbParam numarg    = cbInt(100);

    // Execute the buffer kernel
    BufferKernel(bck.GetQueue(), g, l, 4, &buff1_arg, &buff2_arg, &numarg, &buff3_arg);

    // Read back the result
    b3.Read(bck.GetQueue(), h_buffer3);

    // Some output for checking the result
    for (int i = 0; i < 10; i++) {
        std::cout << h_buffer1[i] << " " <<  h_buffer2[i] << " " <<  h_buffer3[i];
    }

    // Releasing stuff
    delete h_buffer1;
    delete h_buffer2;
    delete h_buffer3;

    // Exiting
    exit(0);
}
```

Compile for CUDA:

```
cd build
cmake -DOPENCL_BACKEND=1 ../
make
```

Compile for OpenCL:

```
cd build
cmake -DCUDA_BACKEND=1 ../
make
```

Conclusion and future plans

What is done

- Performed a feasibility study of an algorithm which is capable to predict coalescence time of GRBs
- Identified parameter space where the task is doable
- Measured coalescence time estimating accuracy
- Measured sky location estimation accuracy
- Assesed computational requirements
- Unified Compute Backend used for CUDA and OpenCL

Problems

- Problems of late and uncertain sky localization to be solved !
- Better computational implementation is needed

Future plans

- Orbital alignment estimation has to be checked
- Estimating other parameters
- Compare performance of alternative implementations
- Optimize the algorithm and window size for real, measured noisy detector data
- Test the algorithm on historical GW data and Swift lightcurves