PyCBC: a device independent approach to CBC analysis

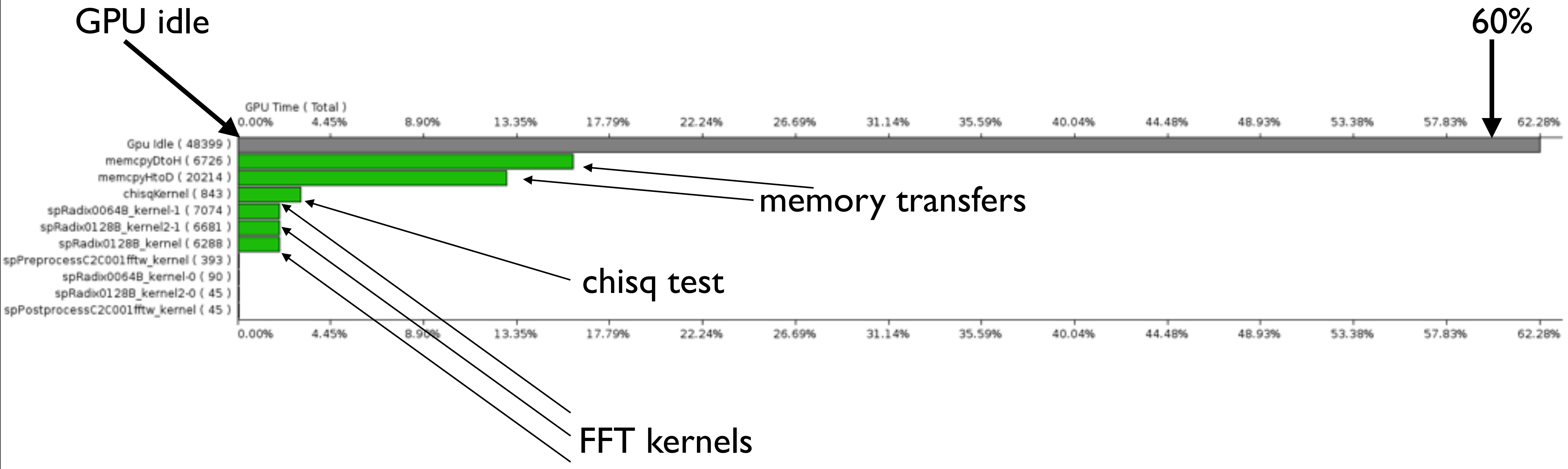
Karsten Wiesner

Albert Einstein Institute Hannover

Aug - 9 - 2011, Hannover

Previous work

- Acceleration of lalapps_inspiral by GPUs obtained a **speedup factor of 15** by performing all FFTs and the chi-squared test on a GPU. (CUDA) https://dcc.ligo.org/cgi-bin/private/DocDB/ShowDocument?docid=40432
- Largest portions of time spent:
  - transferring memory to and from GPU
  - running other computations on the CPU (GPU is idle)



Previous work

- To optimize this, we need to implement more of the search engine (i.e., the whole hot loop) completely on the GPU.
  - The RMKI Virgo Group has shown a possible **speedup factor of 92** by porting the search engine completely to the GPU. (OpenCl) http://www.grid.kfki.hu/twiki/bin/view/RmkiVirgo/GPU_inspiral
- Studying lalapps_inspiral, its procedural code structure and memory management do not meet the requirements of a sustainable framework for GPU accelerated CBC analysis.

Requirements of a novel software framework for CBC analysis

- Modularity (code reuse vs. code duplication eg. coherent pipeline etc.)
- Simple API (when creating a pipeline scientists doesn't necessarily have to touch w/ GPU code)
- Open for extension (closed for modification)
- Configurable (without too much deep code changes)
- Decouple algorithm (pipeline description) from implementation
- Very large parameter spaces - offline searches
- Encapsulation and management of memory
- Introduce object oriented design
- Few and simple, clear data structures
- Scalable to CPUs, GPUs, etc. (interfacing to independent batch systems)
- Better readability (code shall be self explanatory -> graph oriented processing framework)
- Auto documentation
- Robustness (extensive test suite as constrain to deployment)
- Bindings to lalsuite as well as to 3rd party libraries

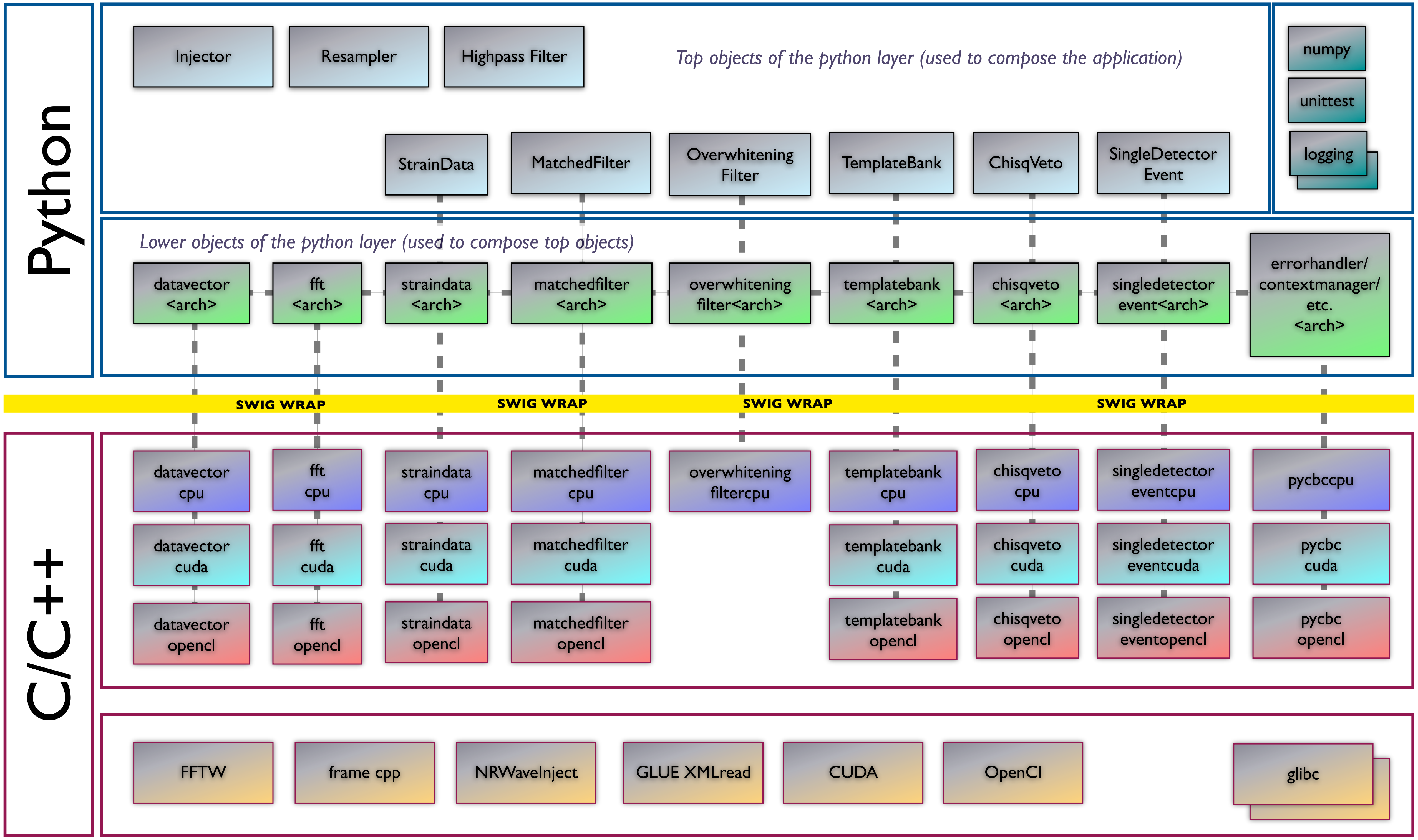
Aug - 9 - 2011, Hannover

People and sites involved in Pycbc development

- Albert Einstein Institute Hannover
  - Karsten Wiesner, Drew Keppel, Badri Krishnan
- Syracuse University
  - Duncan Brown, Alex Nitz,
- University of Wisconsin - Milwaukee
  - Adam Mercer
- Abilene Christian University
  - Josh Willis
- MTA KFKI RMKI Budapest
  - Bence Somhegyi, Gergely Debreczeni

Aug - 9 - 2011, Hannover

Pycbc Layers and packages



Aug - 9 - 2011, Hannover

Code example:Application layer of the test prototype

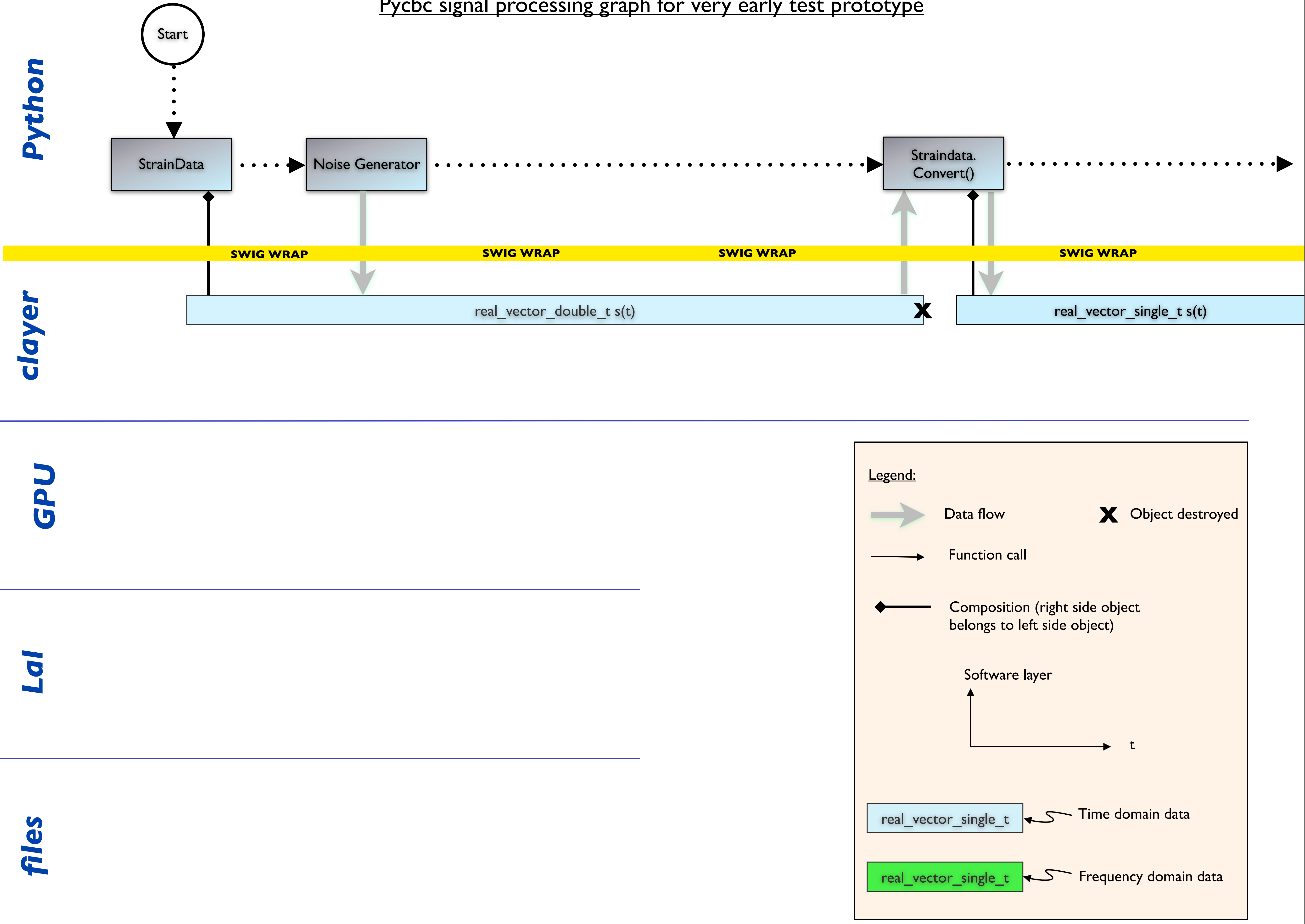
```
import sys
import random

# preliminary hard coded path to packages
sys.path.append('/Users/kawies/dev/src/pycbc')

from pycbc.pycbc import OpenClDeviceContext as ProccessingTargetContext

from pycbc.straindata.straindata_cpu import StrainDataCpu as StrainData
from pycbc.templatebank.templatebank_cpu import TemplateBankCpu as TemplateBank
from pycbc.matchedfilter.matchedfilter_cpu import MatchedFilterCpu as
MatchedFilter
from pycbc.datavector.datavectorcpu import real_vector_single_t as
SnrResultTimeSeries

import logging

logging.basicConfig(level=logging.DEBUG,
                    format='%(name)s %(asctime)s %(levelname)s %(message)s',
                    filename='pycbc_min_pipeline.log',
                    filemode='w')

logger= logging.getLogger('pycbc.main_script')


start_message = 'Starting pycbc single detector minimal pipeline ...'
logger.debug(start_message)
print start_message

# setup straindata
search_time = 128 # typ design spec: 2048
sample_freq = 256 # typ design spec: 4096
length =      search_time * sample_freq
segments = 15
gps_start_time= 871147532
gps_end_time= gps_start_time + search_time
interferometer = "H1"

with ProccessingTargetContext(1) as context:

    strain_data= StrainData(gps_start_time, gps_end_time,
                            segments, sample_freq,
                            interferometer)

    # initialize straindata w/ white noise
    for i in range(length):
        tmp= random.uniform(-1,1)
        strain_data.time_series[i] = tmp

    # convert straindata to single precision
    strain_data.convert_to_single_preci()

    # segmenting straindata and transform into frequency domain
    strain_data.perform_fft_segments()


    # transfer straindata to appropriate memory space on target device
    strain_data.render()
```

```
with ProccessingTargetContext(1) as context:

    strain_data= StrainData(gps_start_time, gps_end_time,
                            segments, sample_freq,
                            interferometer)

    # initialize straindata w/ white noise
    for i in range(length):
        tmp= random.uniform(-1,1)
        strain_data.time_series[i] = tmp

    # convert straindata to single precision
    strain_data.convert_to_single_preci()

    # segmenting straindata and transform into frequency domain
    strain_data.perform_fft_segments()


    # transfer straindata to appropriate memory space on target device
    strain_data.render()


    # create 5 templates (testing the iterator of TemplateBank)
    bank = TemplateBank( 5, strain_data.segments_length,
                            strain_data.segments_delta_x )
    logger.debug("instanciated TemplateBank w/ waveform length: {0}"
                  .format(bank.waveform_length))

    # create matched filter (only generate_snr() has to be implemented
    # for the minimal pipeline)
    matched_filter = MatchedFilter(strain_data.segments_length)
    logger.debug("instanciated MatchedFilter w/ length: {0}".format
    (matched_filter.length))

    # instanciate result vectors
    snr = SnrResultTimeSeries(strain_data.segments_length,
                              strain_data.segments_delta_x)
    logger.debug("instanciated SnrResultTimeSeries as {0}".format(repr(snr)))

    # filter the data against the template bank
    for template in bank:
        htilde = bank.perform_generate_waveform(template)
        for stilde in strain_data:
            matched_filter.perform_generate_snr(stilde, htilde, snr)

    # prepare to leave the processing context
    del(strain_data)
    del(bank)
    del(matched_filter)
    del(snr)
    del(htilde)

    # leaving the ProccessingTargetContext NOW (destroy the device context)
    ############################################################################

end_message = '... end of pycbc single detector minimal pipeline.'
logger.debug(end_message)
print end_message
```

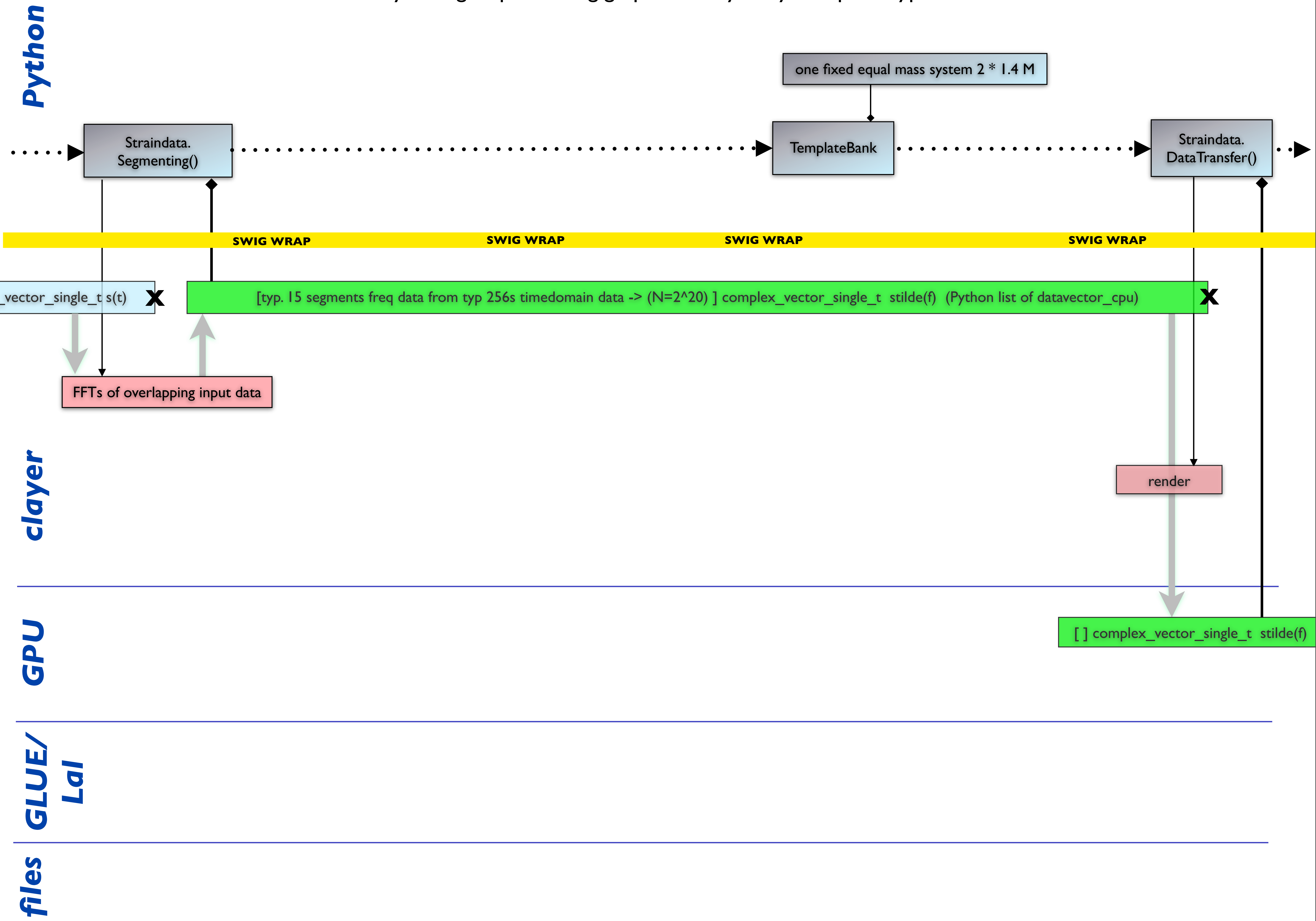
Pycbc signal processing graphs

- Shows processing graph from the application programmers viewpoint
- Shows how functions and kernels are called and where they live (layers)
- Shows how data flows
- Shows memory objects
    - Where they live
    - Who is the owner
    - Which kind of data do they hold
    - How long they live

Pycbc signal processing graph for very early test prototype

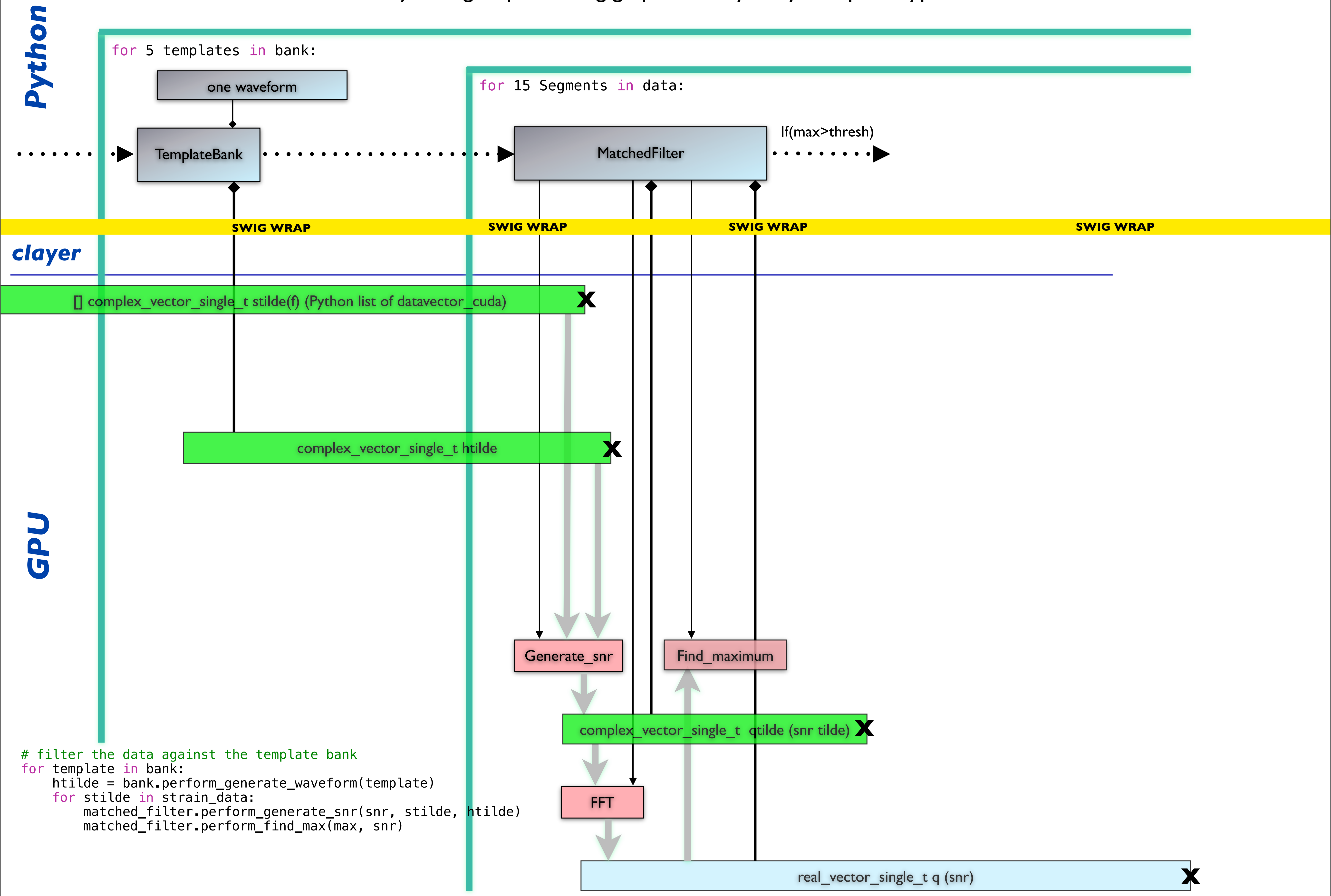


Pycbc signal processing graph for very early test prototype

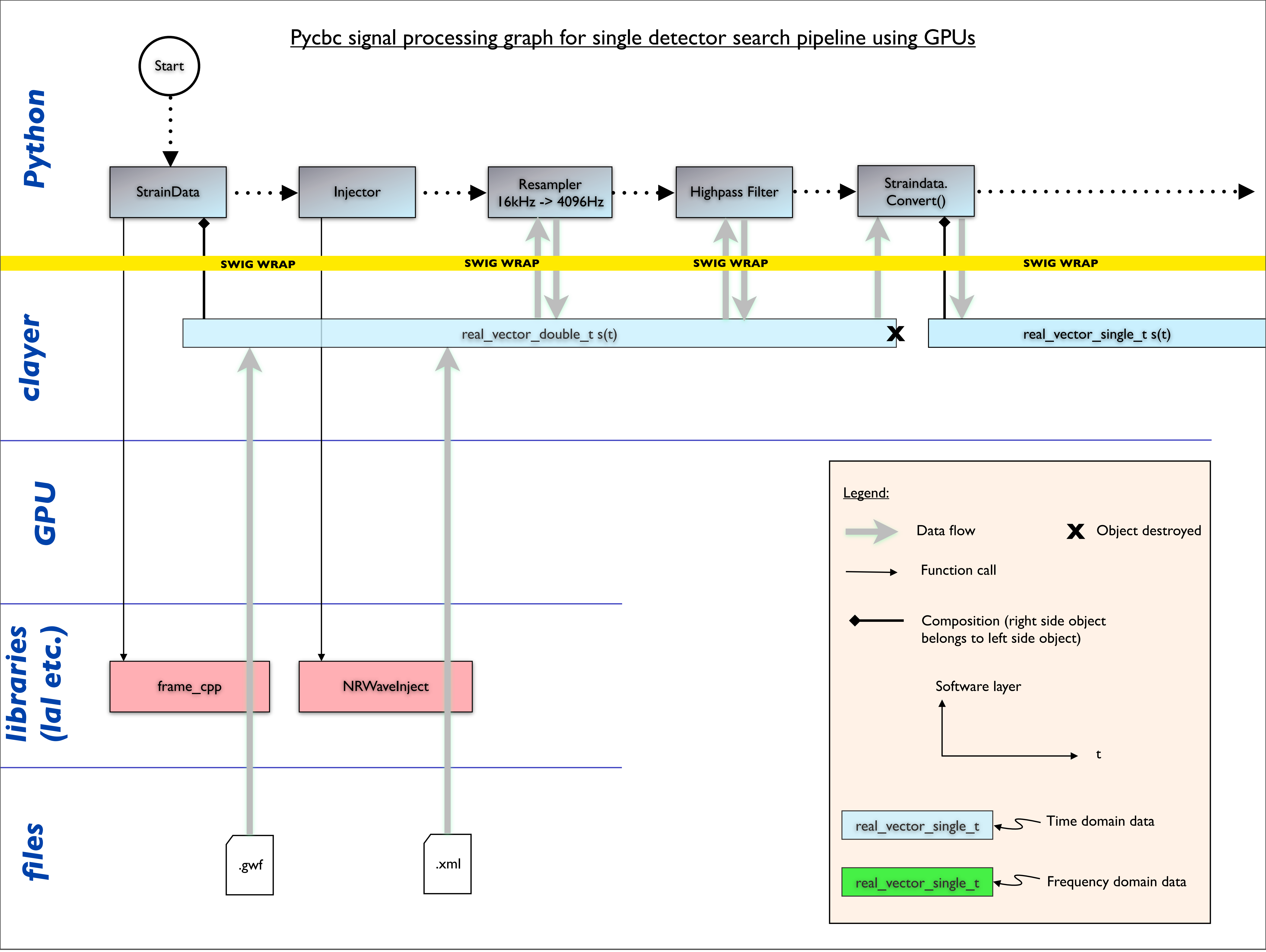


Python
one fixed equal mass system 2 * 1.4 M
Straindata. Segmenting()
TemplateBank
Straindata. DataTransfer()
SWIG WRAP
SWIG WRAP
SWIG WRAP
SWIG WRAP
_vector_single_t s(t)
X
[typ. 15 segments freq data from typ 256s timedomain data -> (N=2^20)] complex_vector_single_t stilde(f) (Python list of datavector_cpu)
X
FFTs of overlapping input data
clayer
render
GPU
[ ] complex_vector_single_t stilde(f)
GLUE/ Lal
files

Pycbc signal processing graph for very early test prototype

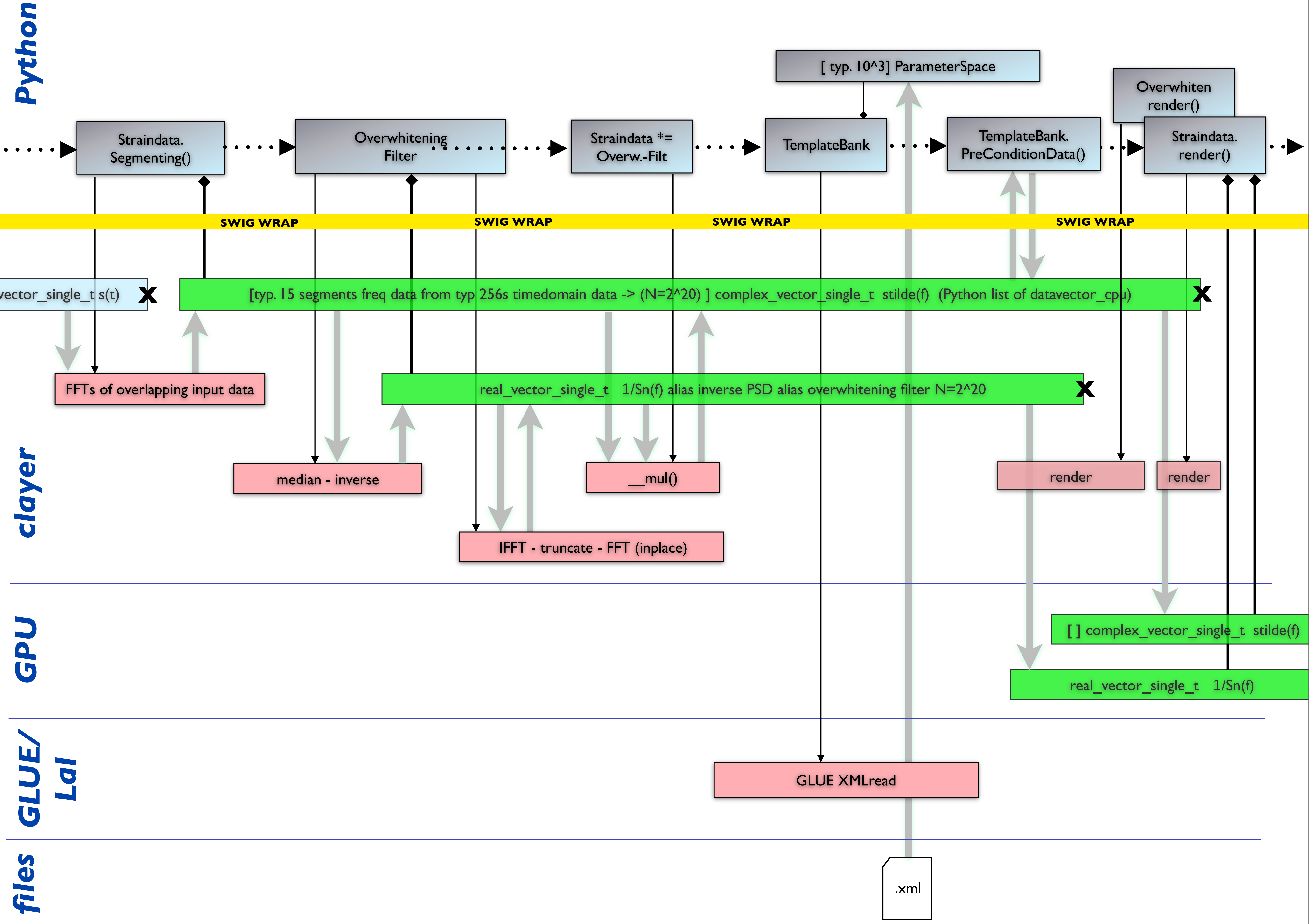


```
# filter the data against the template bank
for template in bank:
    htilde = bank.perform_generate_waveform(template)
    for stilde in strain_data:
        matched_filter.perform_generate_snr(snr, stilde, htilde)
        matched_filter.perform_find_max(max, snr)
```



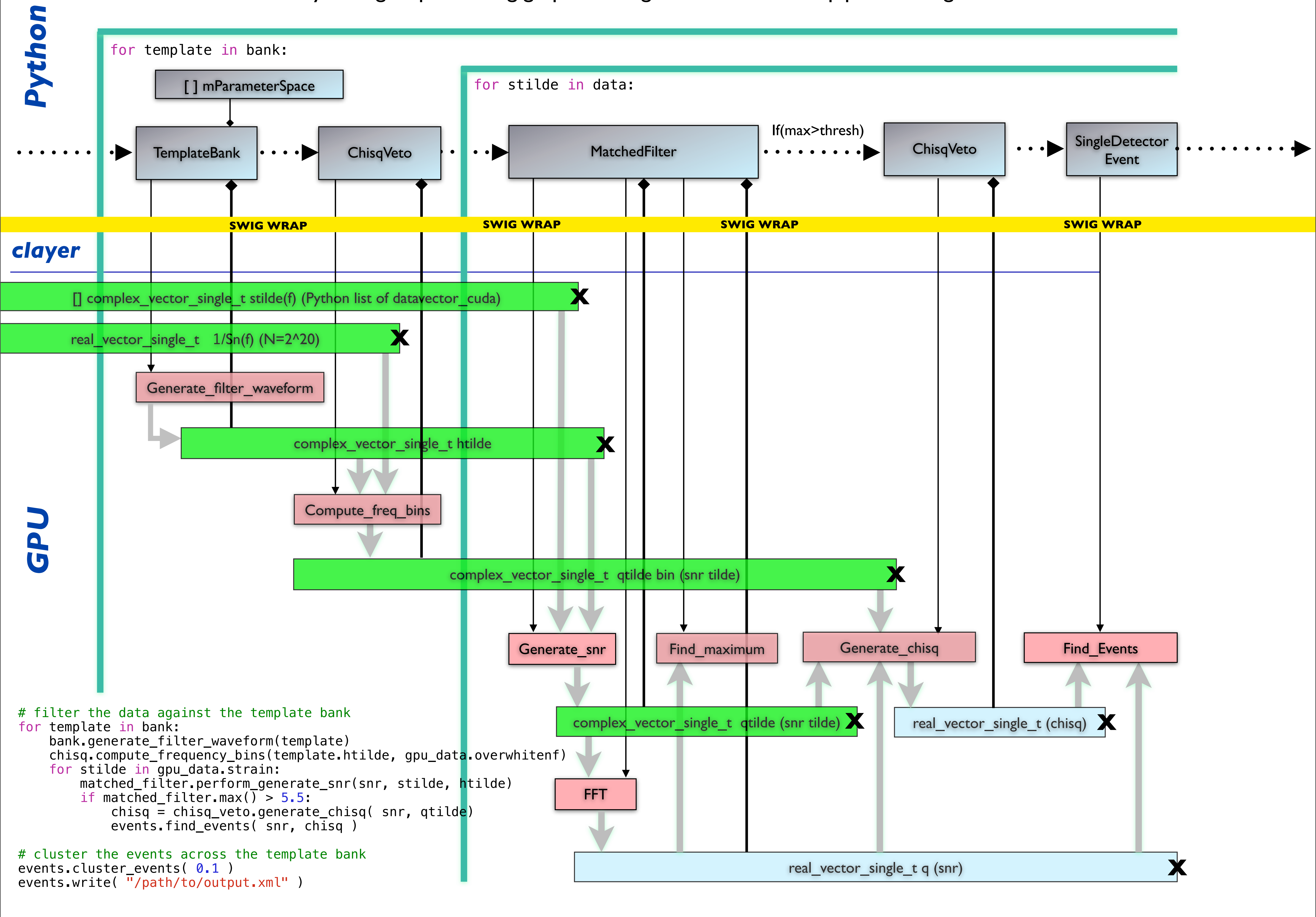
Monday, August 8, 2011

Pycbc signal processing graph for single detector search pipeline using GPUs



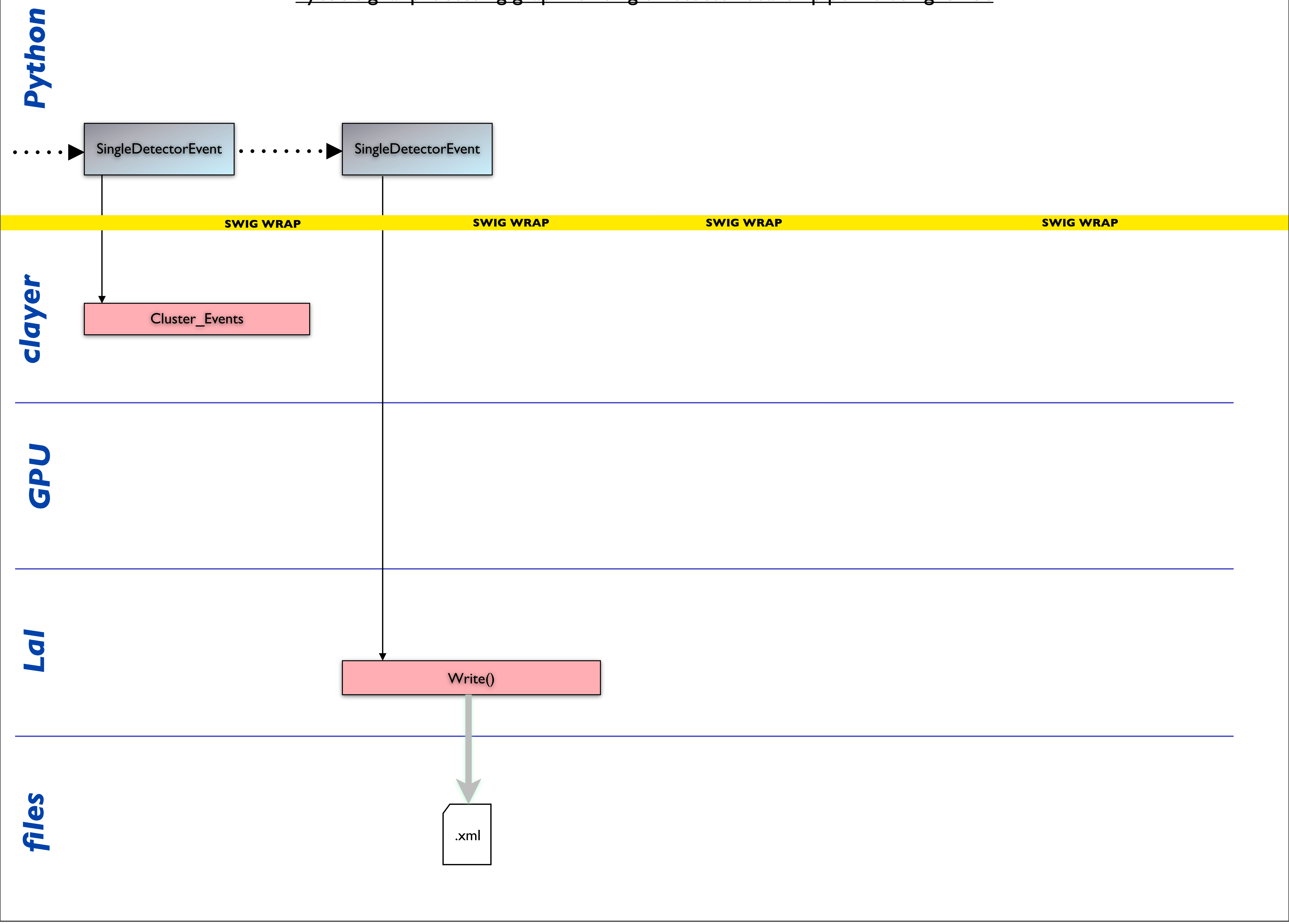
Python
clayer
GPU
GLUE/ Lal
files
[typ. 10^3] ParameterSpace
Overwhiten render()
Straindata. Segmenting()
Overwhitening Filter
Straindata *= Overw.-Filt
TemplateBank
TemplateBank. PreConditionData()
Straindata. render()
SWIG WRAP
SWIG WRAP
SWIG WRAP
SWIG WRAP
_vector_single_t s(t)
x
[typ. 15 segments freq data from typ 256s timedomain data -> (N=2^20)] complex_vector_single_t stilde(f) (Python list of datavector_cpu)
x
FFTs of overlapping input data
real_vector_single_t 1/Sn(f) alias inverse PSD alias overwhitening filter N=2^20
x
median - inverse
__mul()
render
render
IFFT - truncate - FFT (inplace)
[ ] complex_vector_single_t stilde(f)
real_vector_single_t 1/Sn(f)
GLUE XMLread
.xml

Pycbc signal processing graph for single detector search pipeline using GPUs



```
# filter the data against the template bank
for template in bank:
    bank.generate_filter_waveform(template)
    chisq.compute_frequency_bins(template.htilde, gpu_data.overwhitenf)
    for stilde in gpu_data.strain:
        matched_filter.perform_generate_snr(snr, stilde, htilde)
        if matched_filter.max() > 5.5:
            chisq = chisq_veto.generate_chisq( snr, qtilde)
            events.find_events( snr, chisq )

# cluster the events across the template bank
events.cluster_events( 0.1 )
events.write( "/path/to/output.xml" )
```

Pycbc signal processing graph for single detector search pipeline using GPUs


Python
SingleDetectorEvent
SingleDetectorEvent
SWIG WRAP
SWIG WRAP
SWIG WRAP
SWIG WRAP
clayer
Cluster_Events
GPU
Lal
Write()
files
.xml