Redesign of CBC analysis software for advLIGO and further speedup by GPUs

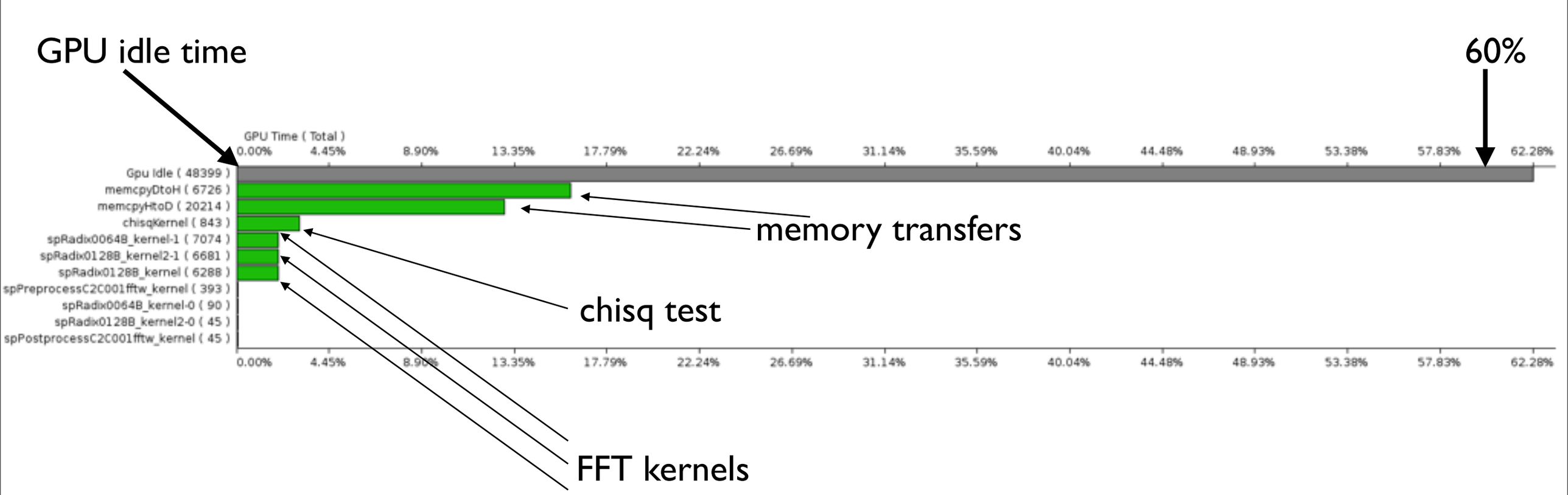
Karsten Wiesner

Albert Einstein Institute
Hannover

Date, Location

Previous work

- Acceleration of lalapps_inspiral by GPUs obtained a speedup of factor 15 by performing all FFTs and the chi-squared test on a GPU.
- Profiling showed clearly that the two main performance eaters are: GPU idle time and memory transfers of large data chunks to and from the GPU which ...



Date, Location

Previous work

- … leaded to the conclusion to implement the search engine (hot loop) completely on the GPU.

- The RMKI Virgo Group developed a low-latency, high performance many-core implementation of the matched-filter gravitational wave search algorithm, based on OpenCl. By implementing the search engine completely on the GPU the application achieved a **speedup factor of 92**
http://www.grid.kfki.hu/twiki/bin/view/RmkiVirgo/GPU_inspiral

- Reengineering of lalapps_inspiral showed that the current procedural code structure and the memory management did not fulfill the requirements of a sustainable framework for GPU accelerated CBC analysis focussing advanced Ligo.

- Detailed descriptions of this investigations as well as links to the GPU accelerated lalapps_inspiral code can be found in LIGO document: G1100401-v1; *Accelerating lalapps_inspiral by using GPUs*; https://dcc.ligo.org/cgi-bin/private/DocDB/ShowDocument?docid=40432

Requirements of a novel software framework for CBC analysis

- Modularity (code reuse vs. code duplication ig. coherent pipeline etc.)
- 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

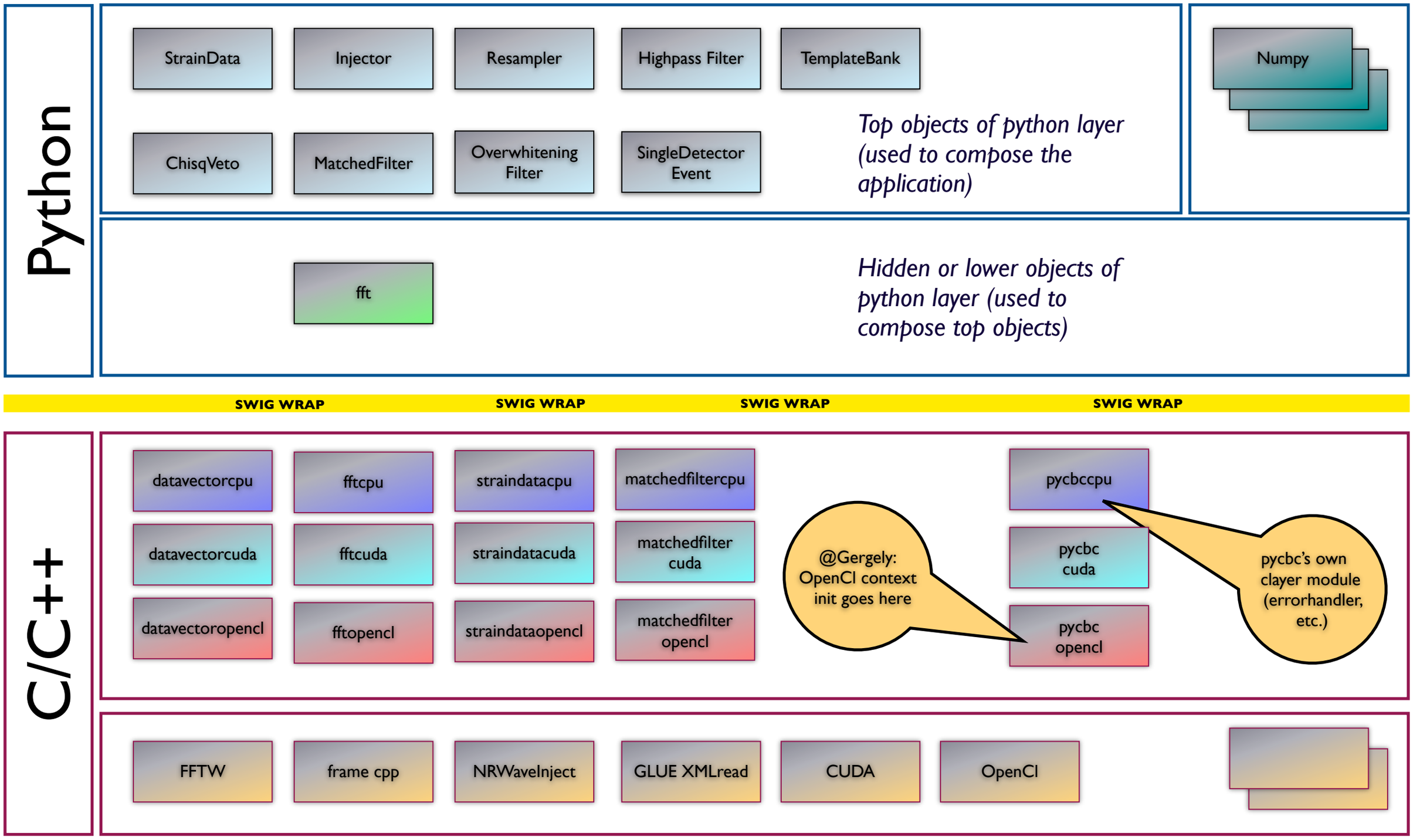
Date, Location

people and sites involved in Pycbc development

- TBD…
- add mailinglist, Redmine and wikies
- necessities to contribute to PyCBC

Date, Location

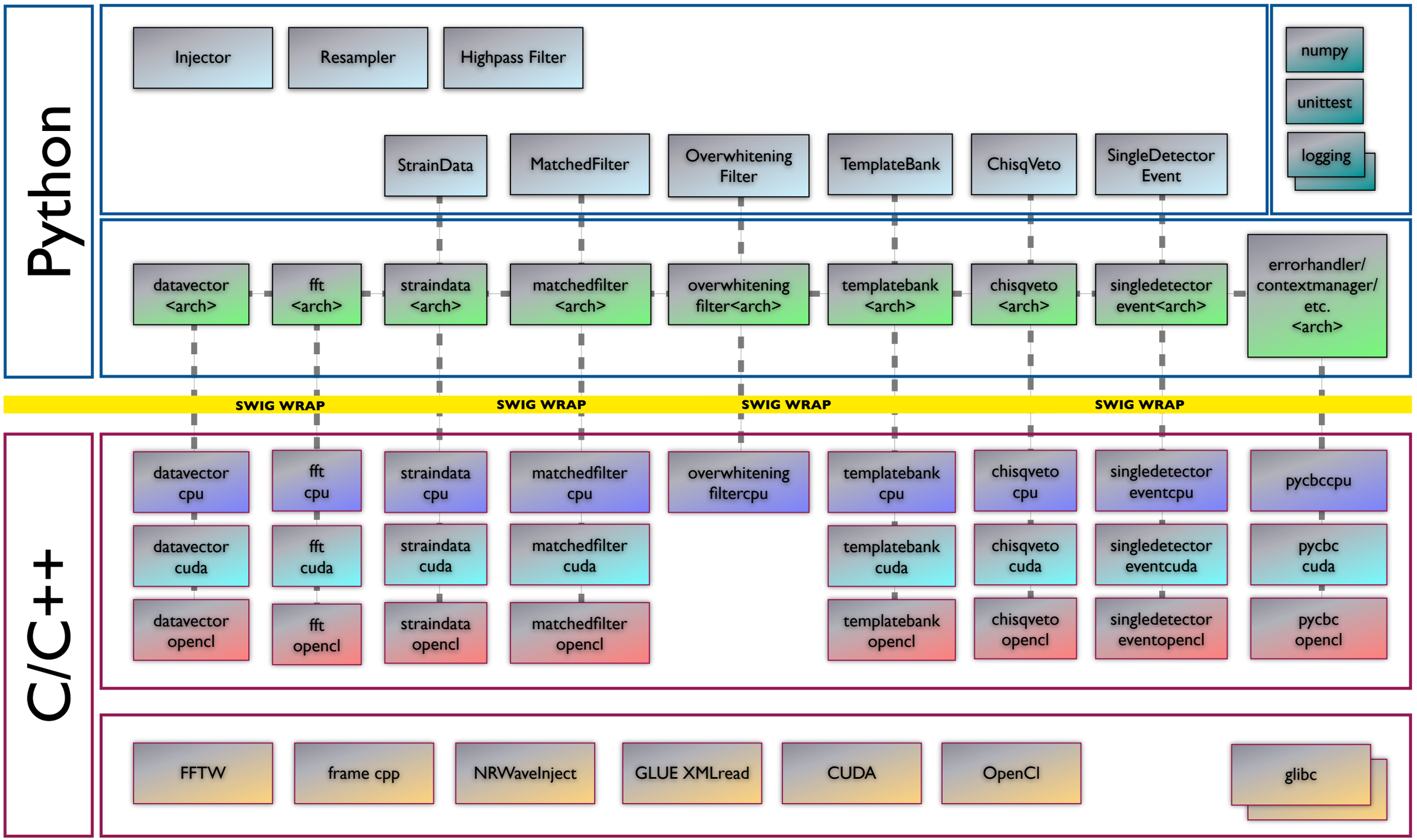
Working slide - Pycbc Layers and packages for the minimal pipeline (gets hidden for the official talk)



Date, Location

Pycbc Layers and packages

Lower objects of the python layer (proposed to compose top objects)



Date, Location

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

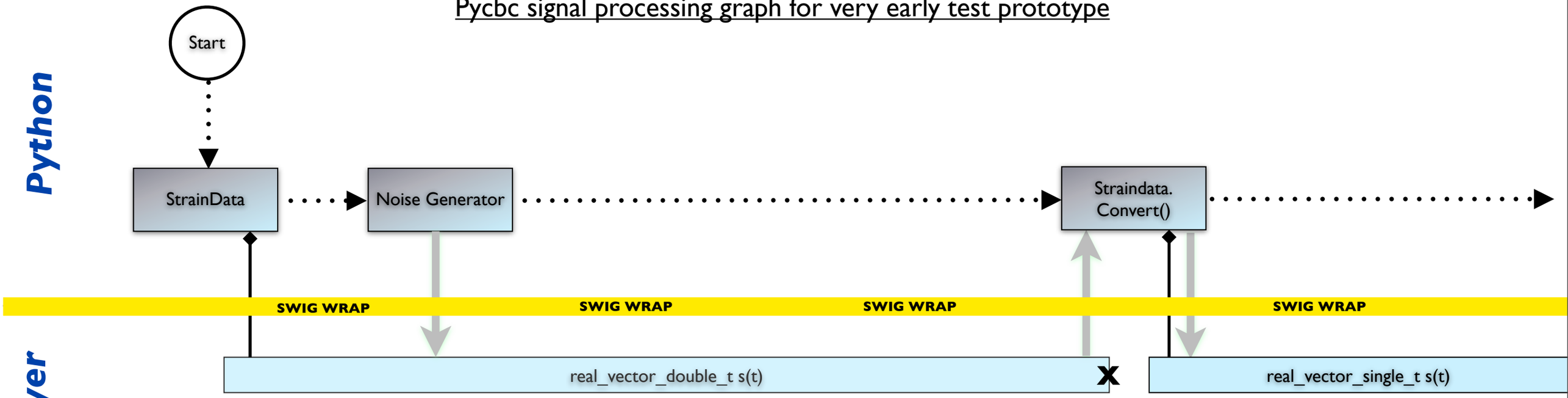
Python

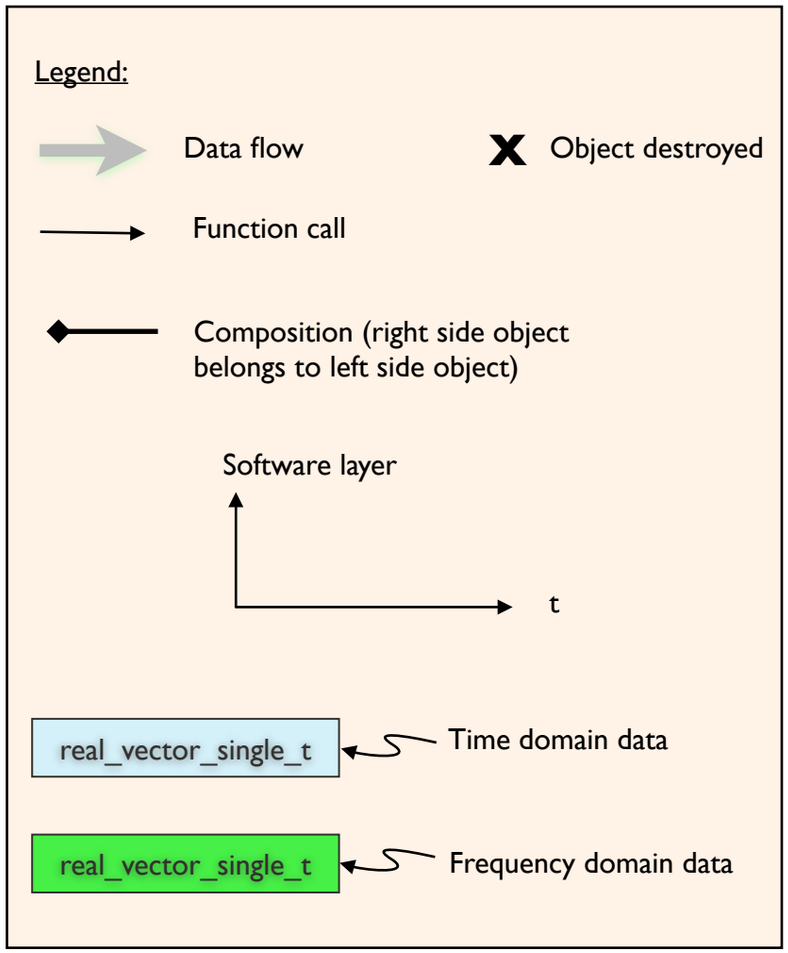
clayer

GPU

Lal

files





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)

[typ. 15 segments freq data from typ 256s timedomain data -> (N=2^20)] complex_vector_single_t stilde(f) (Python list of datavector_cpu)

FFTs of overlapping input data

render

[ ] complex_vector_single_t stilde(f)

clay

GPU

GLUE/
Lal

files

Pycbc signal processing graph for very early test prototype

Python

for 5 templates in bank:

one waveform

TemplateBank

for 15 Segments in data:

MatchedFilter

If(max>thresh)

SWIG WRAP

SWIG WRAP

SWIG WRAP

SWIG WRAP

clayer

[] complex_vector_single_t stilde(f) (Python list of datavector_cuda)

complex_vector_single_t htilde

GPU

Generate_snr

Find_maximum

complex_vector_single_t qtilde (snr tilde)

FFT

real_vector_single_t q (snr)

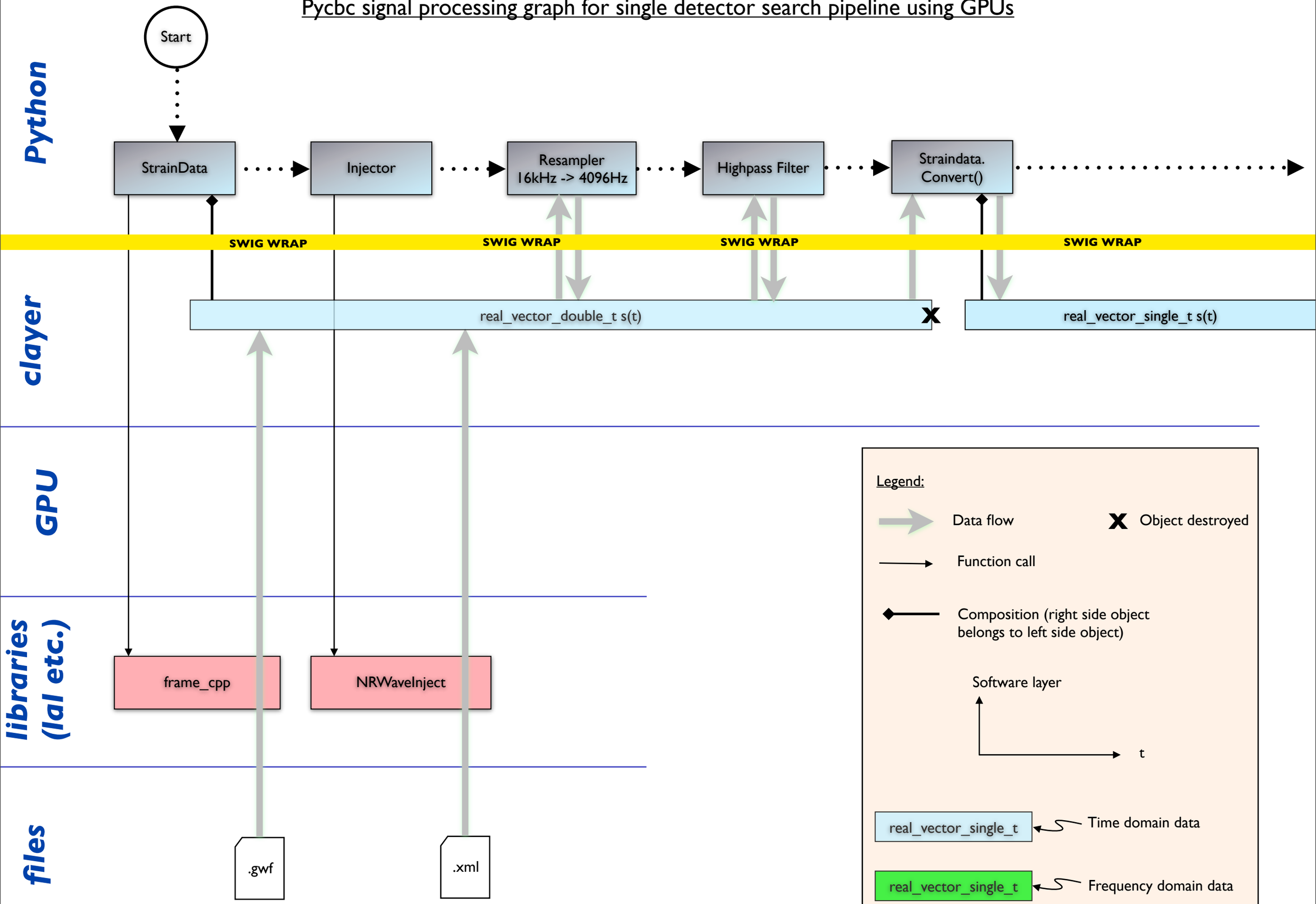
```
# filter the data against the template bank
for template in bank:
    htilde = bank.perform_generate_waveform(template)
    for stilde in strain_data:
        matched_filter.perform_generate_snr(snr, stilde, htilde)
        matched_filter.perform_find_max(max, snr)
```

Code example:
Application (python) 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.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"

strain_data= StrainData(gps_start_time, gps_end_time,
                        segments, sample_freq,
                        interferometer)
logger.debug("instanciated StrainData w/ segment length: {0}".format
(strain_data.segments_length))

# 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 )
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)
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)


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

Pycbc signal processing graph for single detector search pipeline using GPUs



Pycbc signal processing graph for single detector search pipeline using GPUs

Python

[typ. 10^3] ParameterSpace

Straindata. Segmenting()

Overwhitening Filter

Straindata *= Overw.-Filt

TemplateBank

TemplateBank. PreConditionData()

Overwhiten render()

Straindata. render()

SWIG WRAP

SWIG WRAP

SWIG WRAP

SWIG WRAP

_vector_single_t s(t)

[typ. 15 segments freq data from typ 256s timedomain data -> (N=2^20)] complex_vector_single_t stilde(f) (Python list of datavector_cpu)

FFTs of overlapping input data

real_vector_single_t 1/Sn(f) alias inverse PSD alias overwhitening filter N=2^20

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

clay

GPU

GLUE/ Lal

files

Pycbc signal processing graph for single detector search pipeline using GPUs

Python

GPU

clayer

```
for template in bank:
```

```
for stilde in data:
```

[ ] mParameterSpace

TemplateBank

ChisqVeto

MatchedFilter

If(max>thresh)

ChisqVeto

SingleDetector Event

SWIG WRAP

SWIG WRAP

SWIG WRAP

SWIG WRAP

[] complex_vector_single_t stilde(f) (Python list of datavector_cuda)

real_vector_single_t 1/Sn(f) (N=2^20)

Generate_filter_waveform

complex_vector_single_t htilde

Compute_freq_bins

complex_vector_single_t qtilde bin (snr tilde)

Generate_snr

Find_maximum

Generate_chisq

Find_Events

complex_vector_single_t qtilde (snr tilde)

real_vector_single_t (chisq)

FFT

real_vector_single_t q (snr)

```
# 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