Karlsruhe Institute of Technology

MDC data generation code management

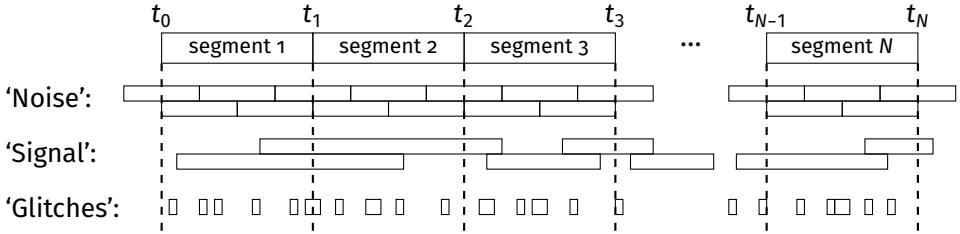
A. Tanasijczuk, **S. Hahn** for EIB Div1 | 27.05.2025

Simulation of detector response

Foundation

- detector response is simulated in independent segments i
- detector response is reproducible for any detector network

} light LAL[†] wrapper



[†]advantage: using approved reviewed implementations

Implemented ‘sources’

Info

- roughly classified by how response is added to output
- SGWB currently only available for L-shape

Noise:
[uncorrelated GC noise]

- based on PSD
- (quasi-)continuous

Signal:
[BNS, BBH, NSBH, Bursts]

- sampled from lists
- finite length

Glitches:
[blips]

- generated with GAN
- short, Poisson-distr.

⇒ a segment may contain any combination of any (overlapping) ‘component’, the ‘components’ need to be continuous over segment boundaries, and the distribution of ‘components’ needs to be proper

General overview

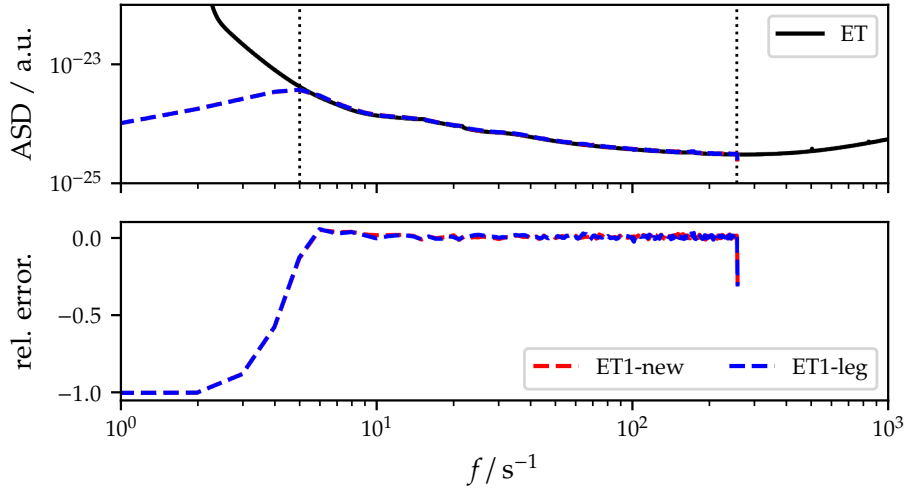
mdcpp

- C++ library + bindings to Python
- standard application to simulate all implemented components

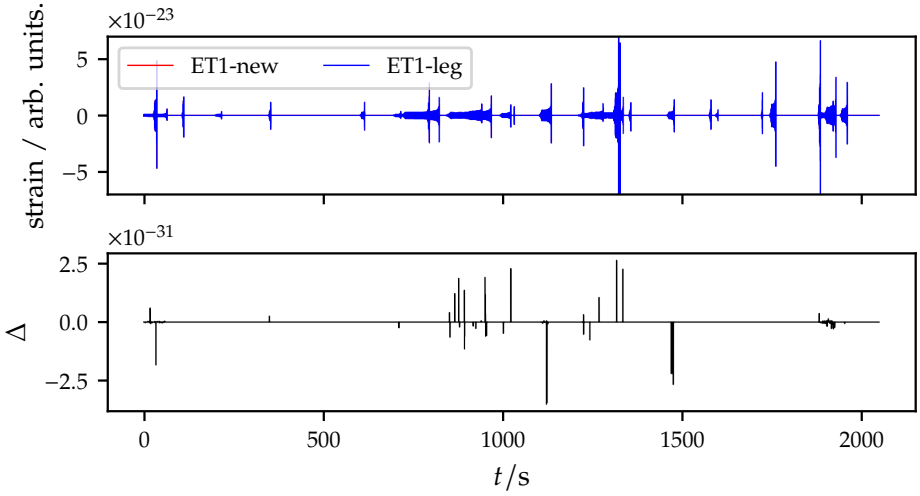
Example: simulate noise and blips [for a build using libtorch, job 0] with standard application `detsim` for a ET-triangle-like dummy network:

```
./bin/detsim -j 0                                          \
    --add_ET_dummy --gen_gcnoise --gen_blipgl              \
    --output_dir ./validation/output                       \
    --input_dir ./cfg                                      \
    --start_time 1000000000 --segment_duration 2048 \
    --output_format gwf
```

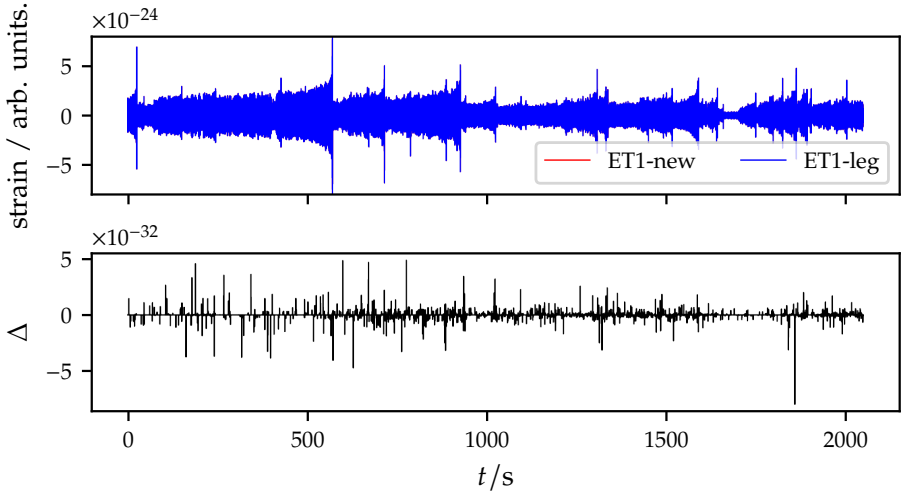
Consistency: old & new code: noise



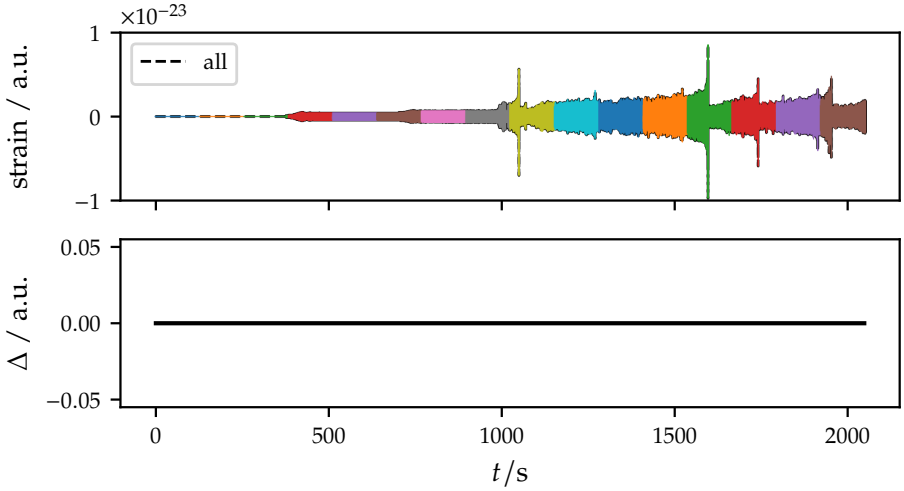
Consistency: old & new code: BBH



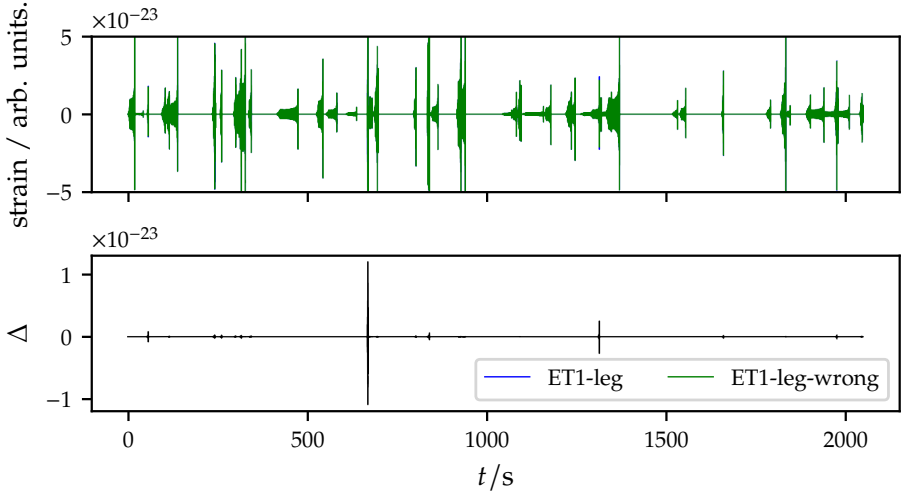
Consistency: old & new code: BHNS



Consistency: segments: BNS



Small bug in MDC1: BBH



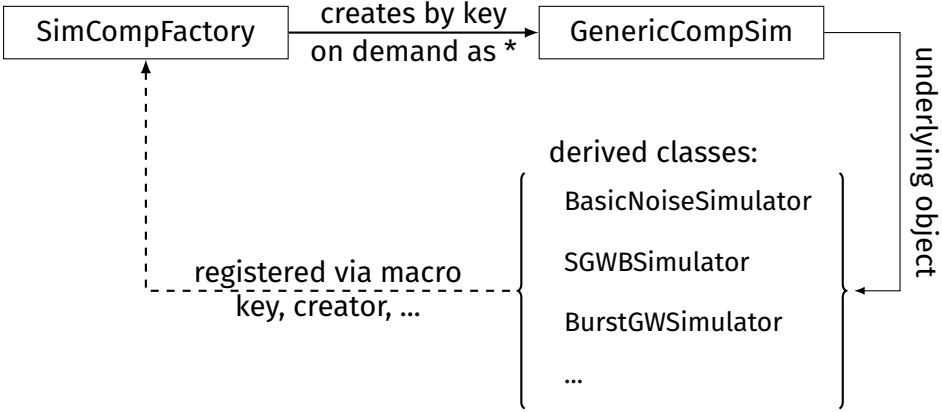
Some run statistics

- `htcondor`, container[†] (apptainer), 2Gb memory, 1CPU
- ET-triangle (8192 Hz), all-in-one, .gwf output
- debug build full output

Part	$\langle t_{\text{run}}/\text{s} \rangle$	$\langle t_{\text{setup}}/\text{s} \rangle$	Comment
Noise	13 ± 4	$<10^{-6}$	`FFTW_ESTIMATE`
SCBC[‡] (~60)	2196 ± 679	$<10^{-6}$	`FFTW_ESTIMATE`
Blip (~40)	2 ± 4	$<10^{-6}$	for GAN
Saving	23 ± 6	-	for .gwf

[†]current way of compiling/running
[‡]save all waveforms separatly, only BHNS

How does signal generation work?

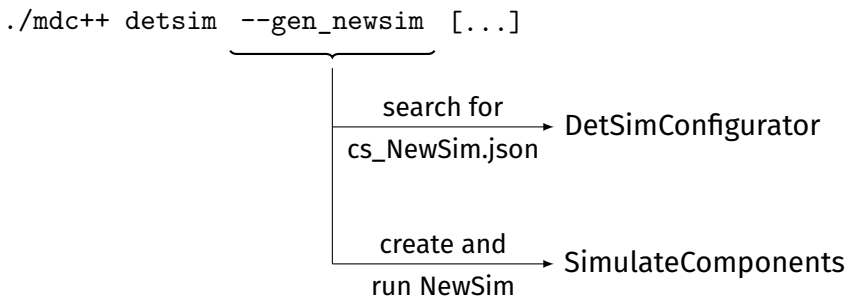


How to add new signal simulator [in C++]?

Creating a child class and register it:

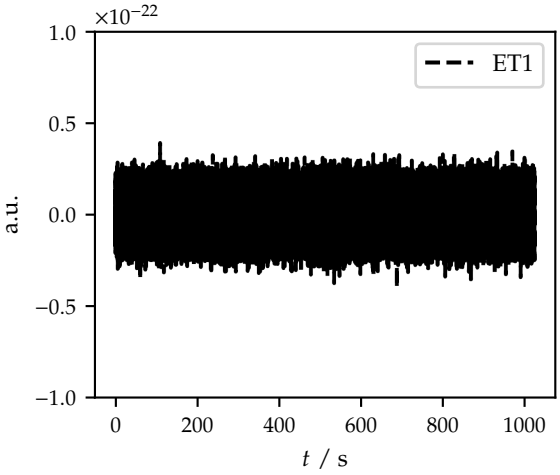
```
class NewSimulator : public GenericComponentSimulator {
 public:
  void Setup(const cfg::DetSimOptions&) override;
  void AddTo(det::DetectorNetwork&) override;
  ...

 private:
  ...

  REGISTER_CLASS_IN_COMPSIMFACTORY(NewSimulator, "NewSim", "newsim");
};
```

How to invoke?



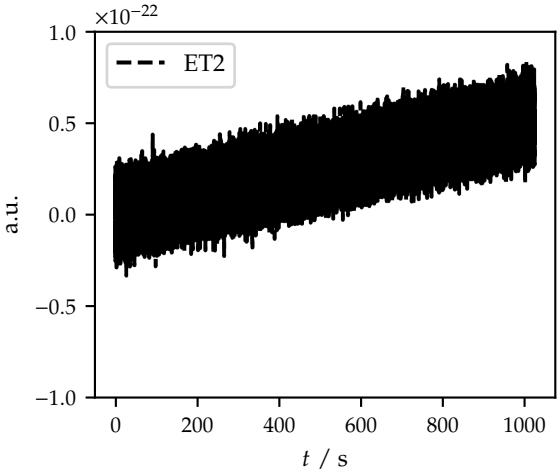
Python bindings - example I

```python
if __name__ == '__main__':
    options = pymdcpp.DetSimOptions()  # get options object

    options.add_et_dummy = True  # short cut for an ET-like detector network
    options.selected_comp_sims = set(['BasicNoise'])

    options.input_dir = './cfg'
    options.output_dir = './validation/output'

    options.component_strategy = pymdcpp.ComponentStrategy.separate_everything
    options.output_format = pymdcpp.OutputFormat.gwframe

    # load manager & setup nn (ET-like)
    mng = pymdcpp.DetSimManager(options)
    net = pymdcpp.InitializeDetectorNetwork(mng)

    # setup & run & save
    mng.Setup()
    mng.SimulateAllActiveComponents(net)
    mng.SaveSubMeasurement(net, "ET1", "BasicNoise")
```



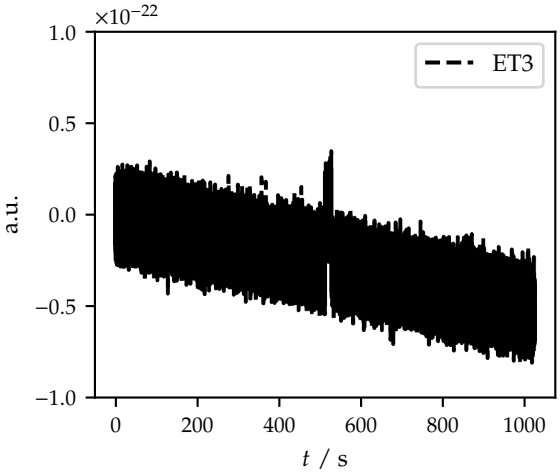
Python bindings - example II

```python
# adding some numpy array to detection
interferometer = find_interferometer(net, 'ET2')
time_series = interferometer.GetTimeSeries('BasicNoise', True)
dummy_array = np.arange(time_series.Size()).astype(np.float64) / 1e28

# example: adding a np-array to a time series bin by bin
pymdcpp.XLALAddREAL8TimeSeriesTestWrapperNDArr(time_series, dummy_array)
mng.SaveSubMeasurement(net, "ET2", "BasicNoise")
```



Python bindings - example III

```python
# adding a mock waveform to the detection
interferometer = find_interferometer(net, 'ET3')
time_series = interferometer.GetTimeSeries('BasicNoise', True)

# example: adding (again) a np-array to a time series bin by bin
pymdcpp.XLALAddREAL8TimeSeriesTestWrapperNDArr(time_series, dummy_array)

gps = pymdcpp.GpsTime(options.start_time + options.segment_duration / 2)
mock_wf = pymdcpp.GenerateMockWavefield(512.0)  # wf_+ = 1e-22 = const

additional_signal = interferometer.ComputeSignalFromWavefield(
    mock_wf.plus, mock_wf.cross, gps, -1.800372, -0.876021, 4.262495, 1)

interferometer.AddResponse(time_series, additional_signal)
mng.SaveSubMeasurement(net, "ET3", "BasicNoise")
```



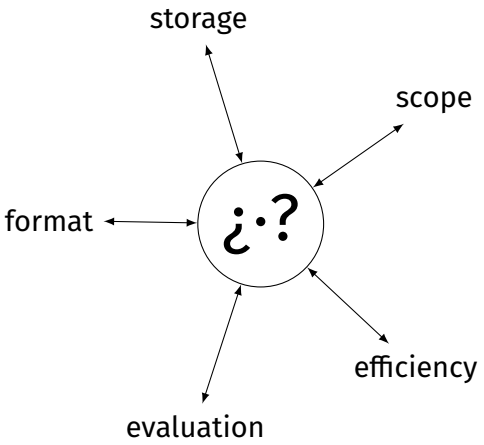
Take away

Summary: mdcpp

- consistent with MDC-1
- (more) modular & parallelizable
- bindings for more flexibility

Open questions

- userbase: expert or general tool
- efficiency: any concerns?
- produced data: combined, split?
- output format: .gwf, .hdf, ...
- evaluation of MDC-X?

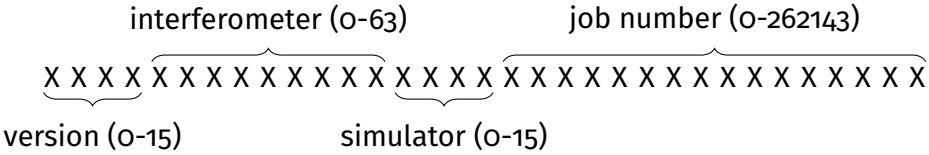


BACKUP

Consistent seeds

Requirements

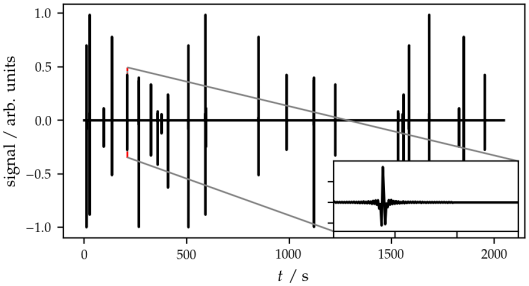
For each segment (job number) we need a different random number for each interferometer, e.g., ET1, and for each simulator, e.g., GCNoise. Most RNGs use 32-bit integers as seeds:



Implementation - blip glitches

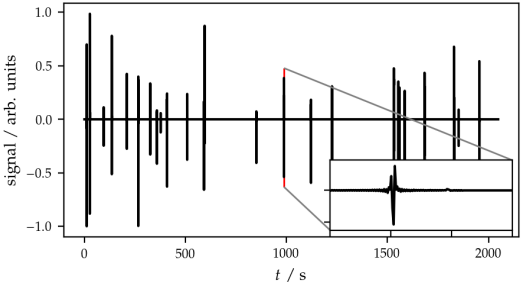
Circular buffer

- all glitches have same size
- load all into continuous memory
- iterate over memory



Invoking GAN

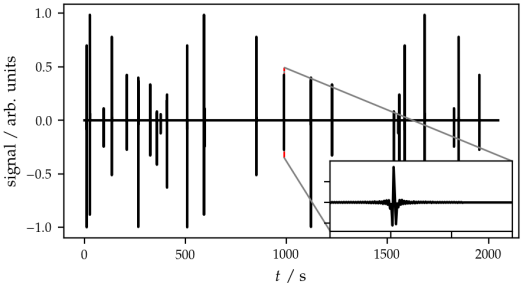
- PyTorch model to TorchScript
- compile with PyTorch C++ [~200Mb]
- invoke within framework



Implementation - blip glitches

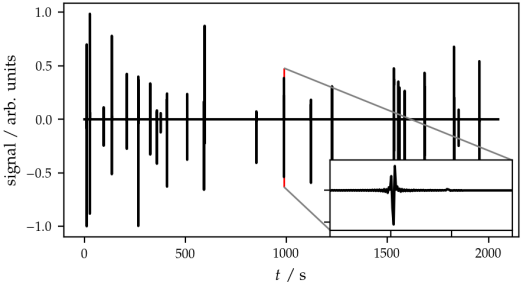
Circular buffer

- all glitches have same size
- load all into continuous memory
- iterate over memory

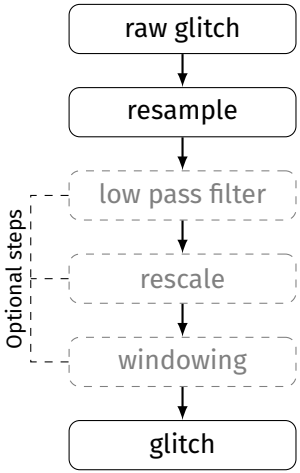


Invoking GAN

- PyTorch model to TorchScript
- compile with PyTorch C++ [~200Mb]
- invoke within framework



Implementation - some details



Take raw glitch from circular buffer/GAN

GAN uses sampling rate of 4096!

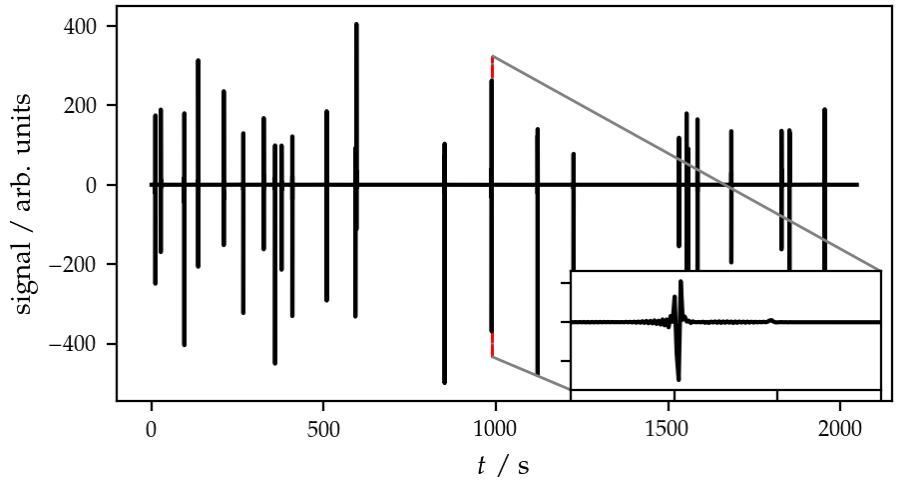
Remove low frequencies

SNR rescaling

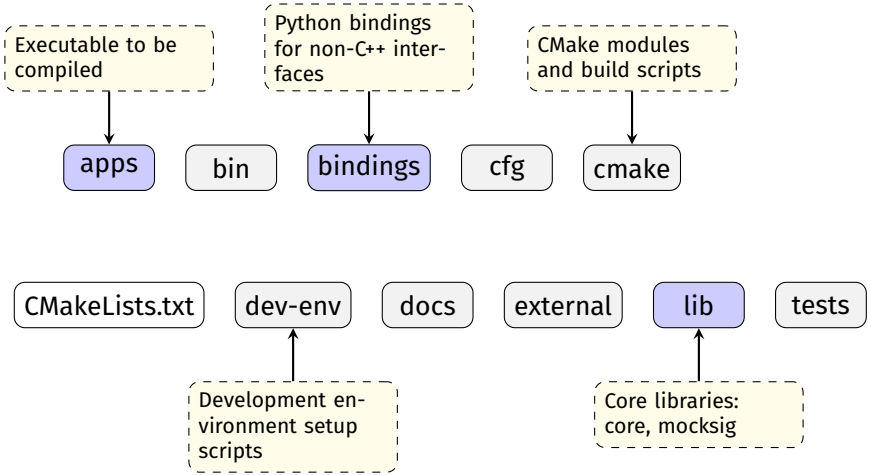
windowing to remove edge effects

final glitch written into output

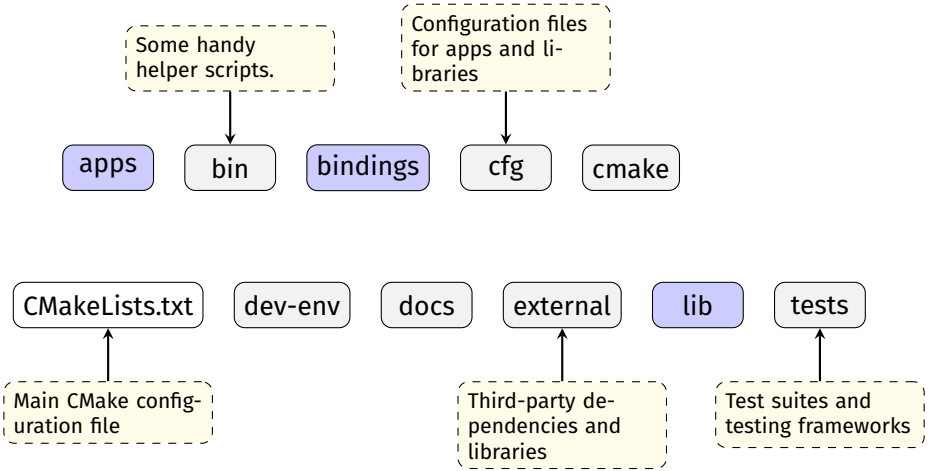
Implementation - throwing everything in



Overview - repository



Overview - repository



Example compilation

```
cd $SOURCE_DIR/dev-env

./build-containers.sh  # builds apptainer container for
   dependencies
# ./build-containers.sh pytorch  # builds apptainer container
   with pytorch
./build-containers.sh basic-submit.def  # for submission

# print help of detsim
apptainer run containers/basic-submit.sif --help
```

⇒ makes it very easy for submission to cluster