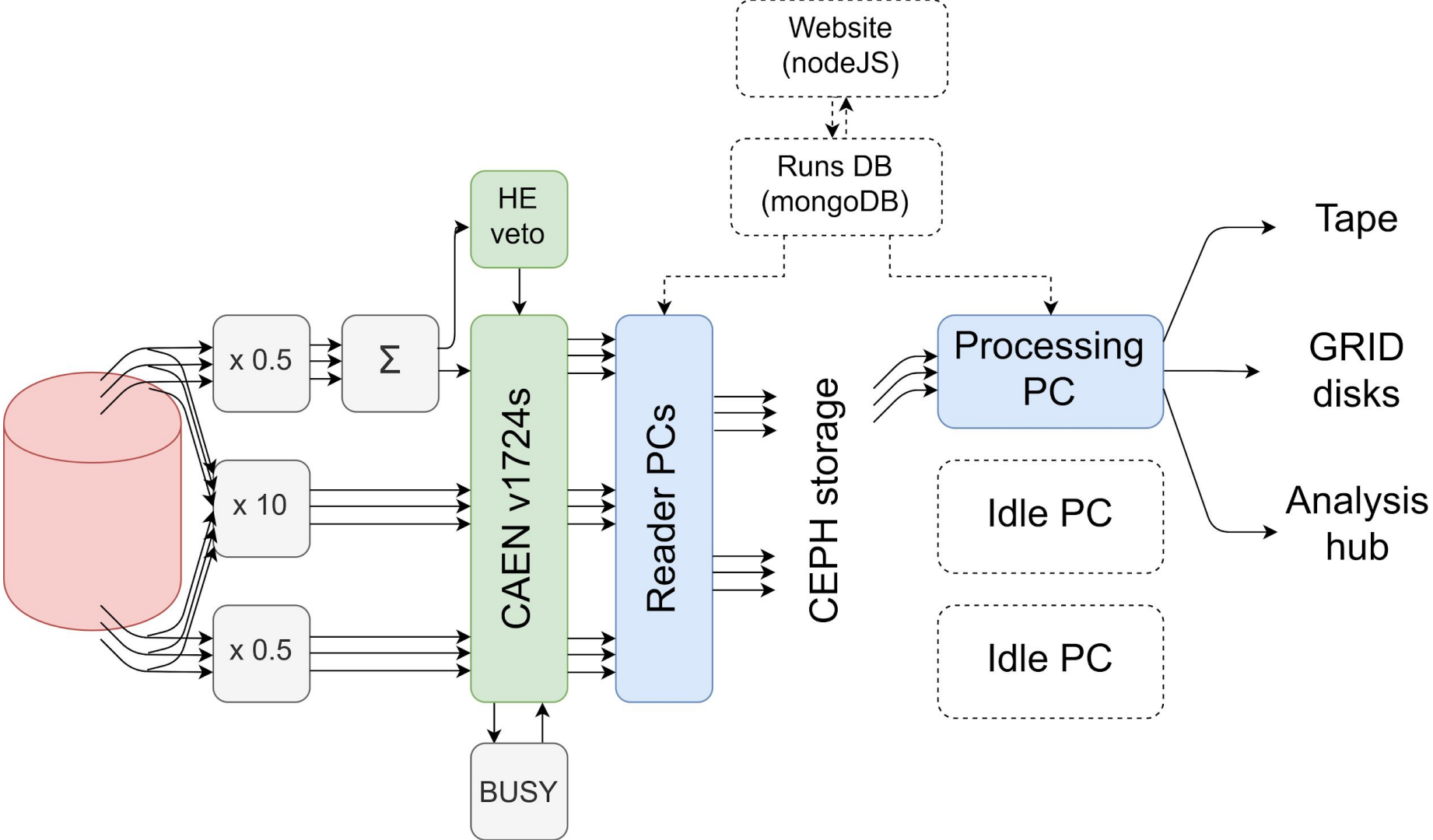
XENONnT DAQ and processing

Jelle Aalbers

Work with Daniel Coderre, Chris Tunnell, Darryl Masson, Peter Gaemers, Joran Angevaare, and several others.

28 October 2019

https://github.com/AxFoundation/strax
https://github.com/XENONnT/straxen
https://github.com/coderdj/redax


Website
(nodeJS)
Runs DB
(mongoDB)
HE
veto
x 0.5
Σ
x 10
x 0.5
CAEN v1724s
BUSY
Reader PCs
CEPH storage
Processing
PC
Idle PC
Idle PC
Tape
GRID
disks
Analysis
hub

raw_records

PMT waveforms

Fixed-length fragments

O(PB/year) compressed

records

Factor ~4 reduction:

removed single el. trains

and baseline samples

peaks

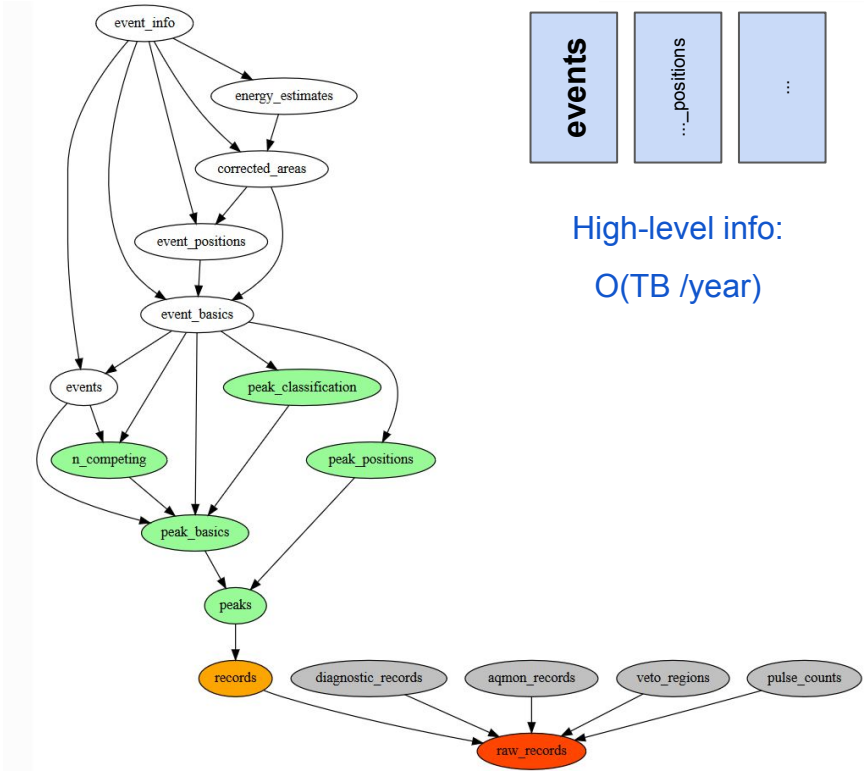
S1/S2 sum wf.s / hitpatterns

O(20 TB/year)

peak_basics

..._positions

...



events

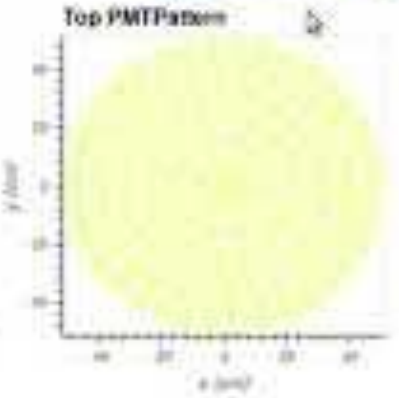
..._positions

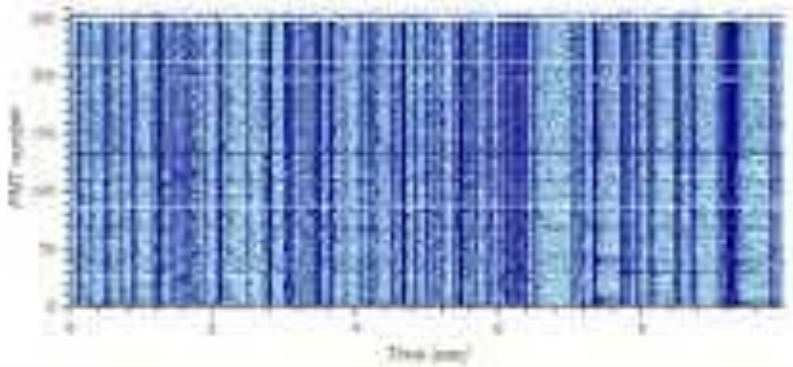
...

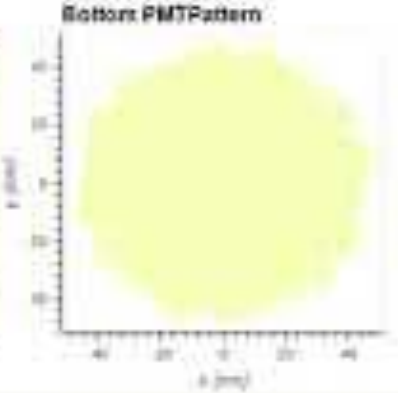
High-level info:

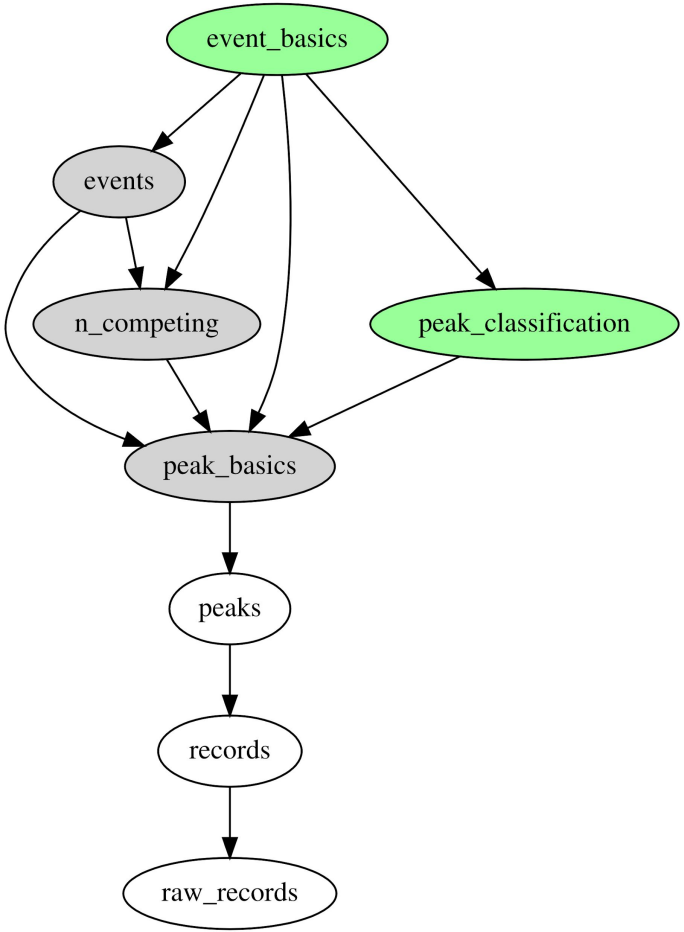
O(TB /year)


PeakSum Waveform


Top PMT Pattern




Bottom PMT Pattern



Analyst:

“Load `event_basics` for run 142,
with S1 coincidence requirement = 2”

Strax, behind the scenes:

1. Build graph of dependencies
2. Find which data types can be loaded
3. Compute others that are needed

Analyst receives dataframe

Computed dataframes are now stored
(with tracking of the custom options)

Simple arrays >> lists of custom objects

Object creation has a penalty even in C

Example: make a histogram of tau-tau jets in CMS

0.018 MHz	full framework (CMSSW, single-threaded C++)
0.029 MHz	load all 95 jet branches in ROOT
2.8 MHz	load jet p_T branch (and no others) in ROOT
12 MHz	allocate C++ objects on heap, fill, delete
31 MHz	allocate C++ objects on stack, fill histogram
250 MHz	minimal "for" loop in memory (single-threaded C)

Pivarski, J. et. al. "Toward real-time data query systems in HEP" ACAT 2017 proceedings, arXiv:1711.01229

In python, object-level code is particularly slow

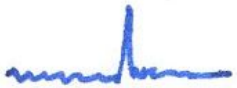
... but numpy and numba allow array ops at native performance

Avoiding variable-length objects

XENON1T / pax: **pulses**

Time Data	Channel	
130214564	24	
130214575	12	
130214580	192	
130214590	87	

XENONnT / strax: **"records" / "fragments"**

Time	Channel	index	Data
130214564	24	0	
130214575	12	0	
130214580	192	0	
130214590	87	0	
130214590	87	1	
130214590	87	2	

Baselining/hitfinding take this splitting into account. Sum waveform is indifferent.

hax

```python
class LargestS2Area(hax.minitrees.TreeMaker):
    """Find the largest S2 area in the event.

    Provides:
     - largest_s2_area: Area (PE) of largest S2 in the event
    """
    __version__ = '0.1'
    branch_selection = ['peaks.area', 'peaks.type']

    def extract_data(self, event):

        s2_areas = []
        for p in event.peaks:
            if p.type != 's2':
                continue
            s2_areas.append(p.area)

        result = 0
        if len(s2_areas):
            result = max(s2_areas)

        return dict(largest_s2_area=result)
```

Return type in comment (hopefully)
Specify branch selection (or run slow)
Data is in nested objects (unlike in analysis)

strax

```python
class LargestS2Area(strax.LoopPlugin):
    """Find the largest S2 area in the event.
    """
    __version__ = '0.1'
    depends_on = ('events', 'peak_basics', 'peak_classification')

    dtype = [
        ('largest_s2_area', np.float32,
            'Area (PE) of largest S2 in event')]

    def compute_loop(self, event, peaks):

        s2s = peaks[peaks['type'] == 2]

        result = 0
        if len(s2s):
            result = s2s['area'].max()

        return dict(largest_s2_area=result)
```

Return type declared: searchable
Dependencies declared: trackable
Data is in numpy arrays: exactly like analysis