# TBB based multi-threading in Wire-Cell

Haiwang Yu (BNL)
for the Wire-Cell team

LArSoft Coordination Meeting

Nov. 19, 2019

# Wire-Cell – brief review

Wire-Cell is a Software project for LArTPC reconstruction
Lead by Brett Viren etc.

Wire-Cell uses "Data Flow Programing" paradigm

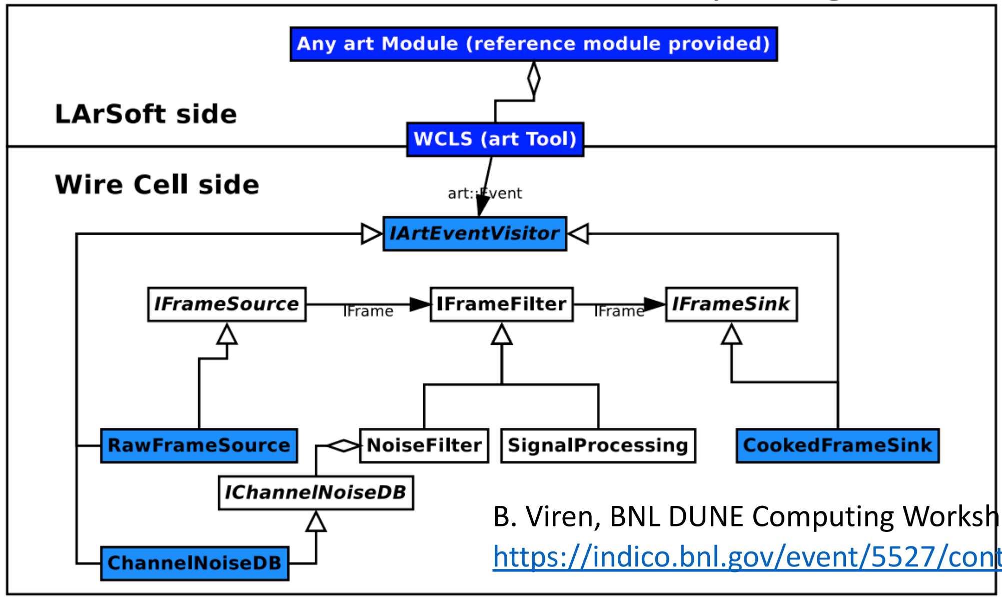
Wire-Cell ported graph

- computing *nodes* with defined input/output type
- ***const* data objects passed along edges**
- **no mutable global**
- run time configurable using *JSON/jsonnet*

Wire-Cell engine

- *Pgrapher* – single thread, no overhead
  - current default
- *TbbFlow* – multi-thread, memory sharing, some memory overhead
  - Not in wire-cell ups build yet

Interact with LArSoft via *larwirecell* package



B. Viren, BNL DUNE Computing Workshop, Jan. 2019
https://indico.bnl.gov/event/5527/contributions/25812/

References:

Wire-Cell main repository:
https://github.com/WireCell/wire-cell-toolkit

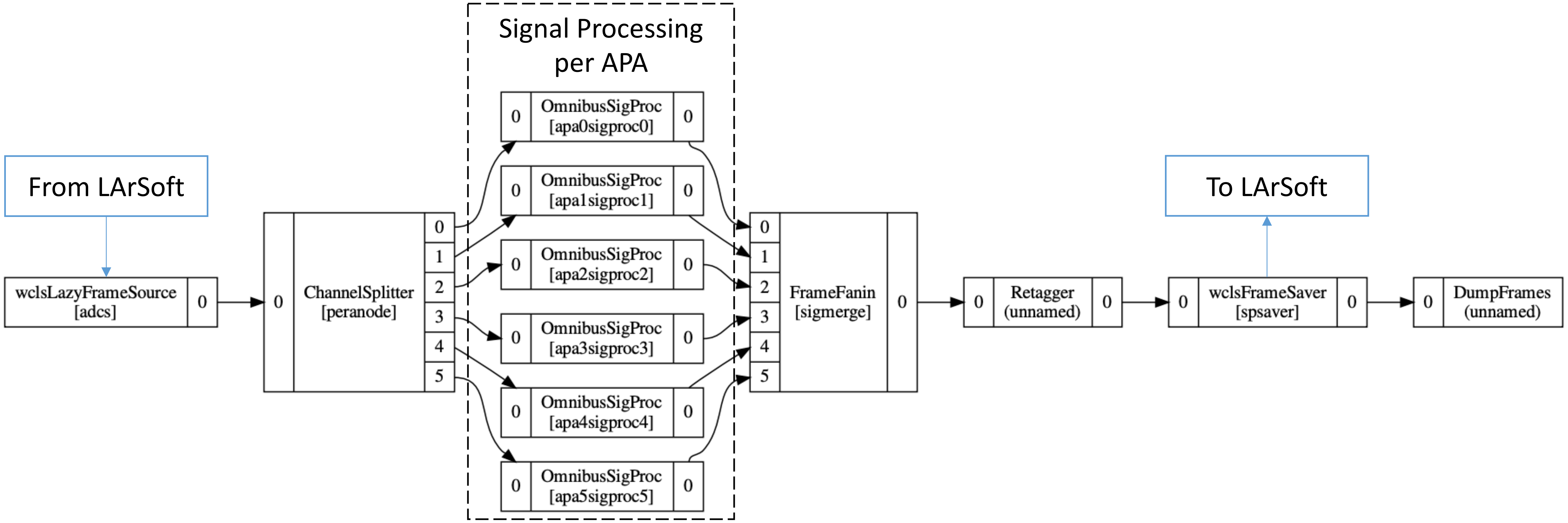
B. Viren, BNL DUNE Computing Workshop, Jan. 2019
https://indico.bnl.gov/event/5527/contributions/25812/

Manual, blog, Doxygen
https://wirecell.github.io/

Tutorial website by C. Zhang etc.
https://czczc.github.io/wire-cell-tutorial/

# Example of Wire-Cell graph: Signal Processing

Configured by https://github.com/HaiwangYu/wct-analysis/blob/master/exp_data/dec-to-sig.jsonnet
*Pgrapher* and *TbbFlow* share same jsonnet configuration file



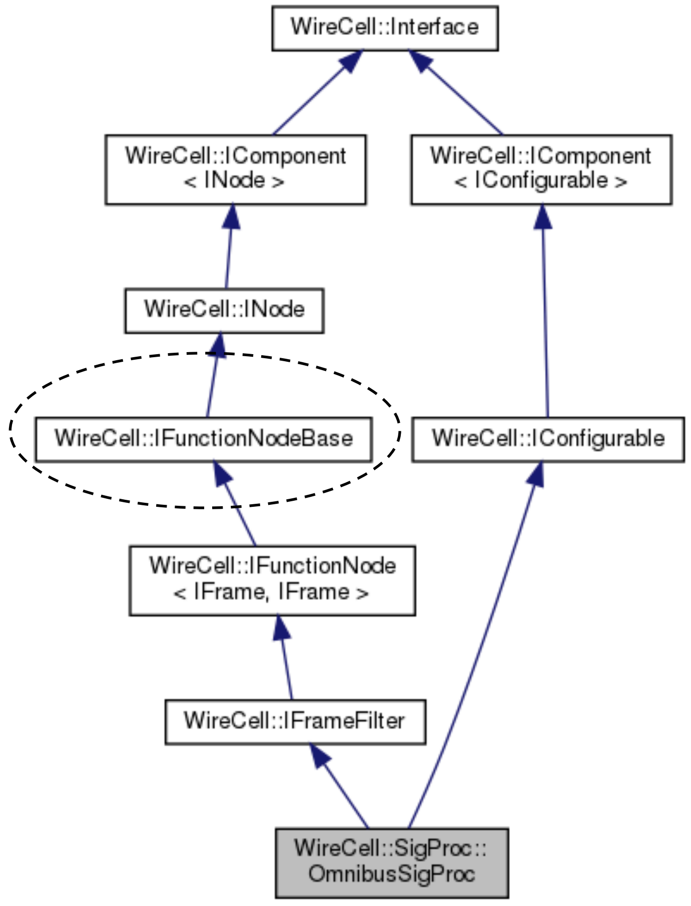
Signal Processing per APA
From LArSoft
To LArSoft
wclsLazyFrameSource [adcs]
ChannelSplitter [peranode]
OmnibusSigProc [apa0sigproc0]
OmnibusSigProc [apa1sigproc1]
OmnibusSigProc [apa2sigproc2]
OmnibusSigProc [apa3sigproc3]
OmnibusSigProc [apa4sigproc4]
OmnibusSigProc [apa5sigproc5]
FrameFanin [sigmerge]
Retagger (unnamed)
wclsFrameSaver [spsaver]
DumpFrames (unnamed)

# Example of Wire-Cell nodes

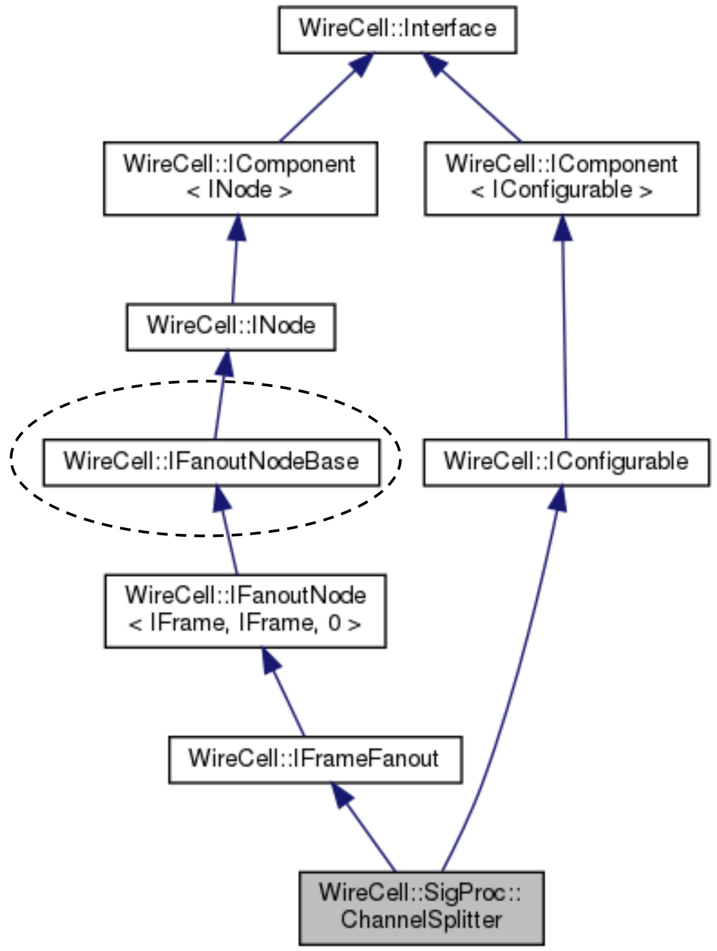
*INode* types:

- Source
- Sink
- Function
- Fanout
- Fanin
- etc.

*IConfigurable* provides JSON/jsonnet configuration interface



WireCell::Interface
WireCell::IComponent < INode >
WireCell::IComponent < IConfigurable >
WireCell::INode
WireCell::IFunctionNodeBase
WireCell::IConfigurable
WireCell::IFunctionNode < IFrame, IFrame >
WireCell::IFrameFilter
WireCell::SigProc::OmnibusSigProc



WireCell::Interface
WireCell::IComponent < INode >
WireCell::IComponent < IConfigurable >
WireCell::INode
WireCell::IFanoutNodeBase
WireCell::IConfigurable
WireCell::IFanoutNode < IFrame, IFrame, 0 >
WireCell::IFrameFanout
WireCell::SigProc::ChannelSplitter

# Wire-Cell data objects

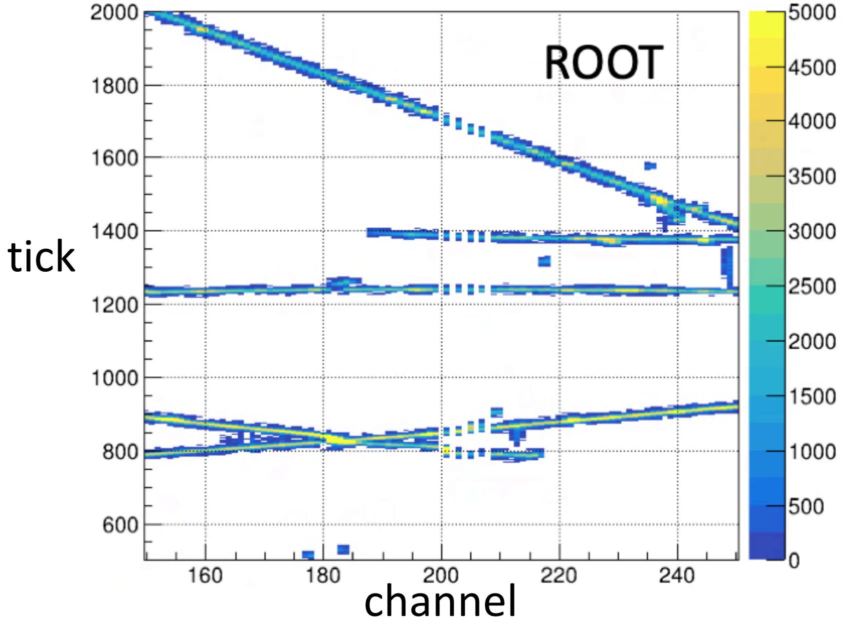
Initially designed for transient only

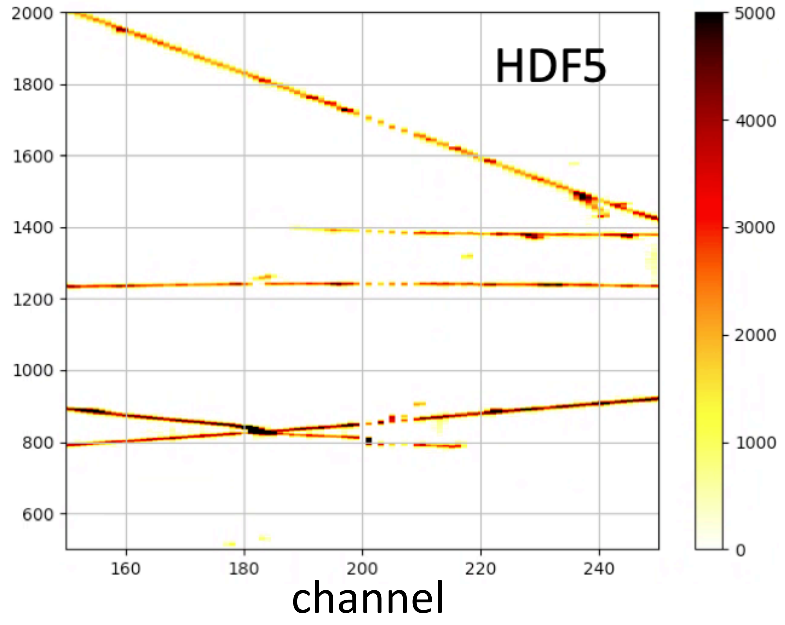
- Only considered interfacing to LArSoft objects

Some persistent mechanism in Wire-Cell

- ROOT – Magnify utilities
- Initial exploration on HDF5 with H5Cpp
  - https://github.com/WireCell/wire-cell-toolkit/pull/10

Wire-Cell 'Frame' serialized to ROOT and HDF5 format





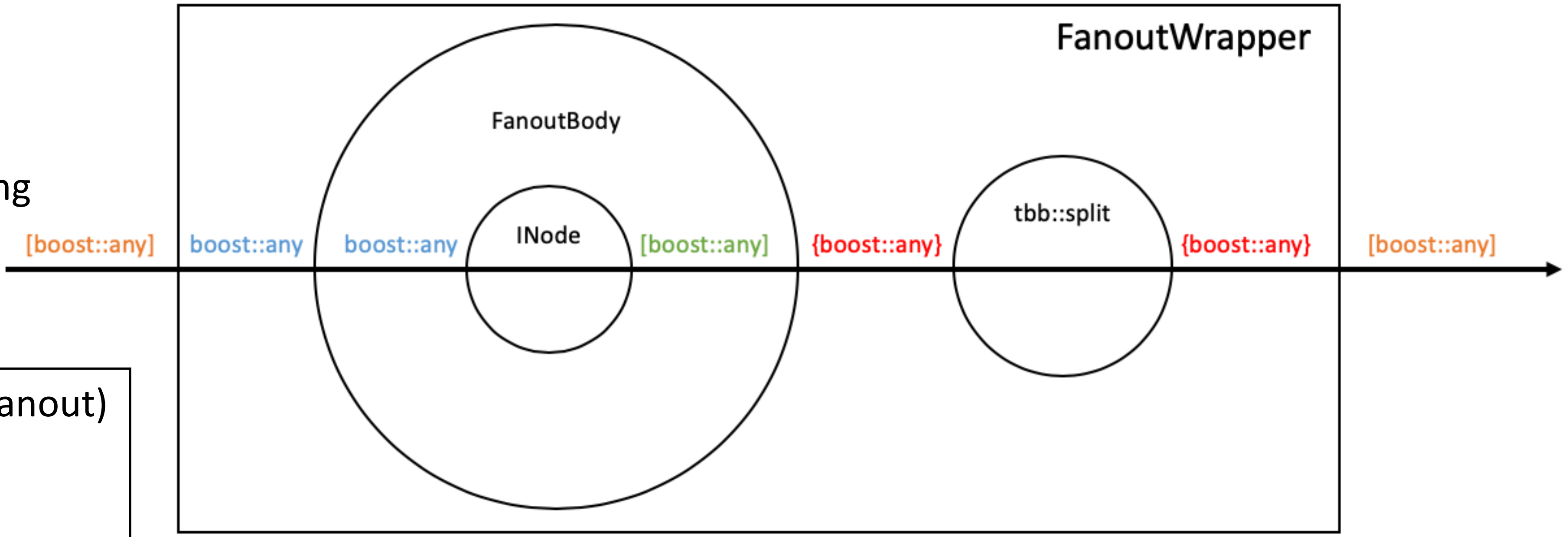
# Wire-Cell node ⟷ TBB node

nodes with multiple input/output ports

- Wire-Cell nodes use STL containers that could have **run-time variable length**
- TBB nodes use *std::tuple* which has **compile-time variable length**
- Some efforts made for this adaption
- More in: https://github.com/WireCell/wire-cell-toolkit/tree/master/tbb

type flow for the `*FanoutWrapper*`, an object is responsible for any type conversion at its boundaries

*INode* handles the logic splitting
*tbb::split* handles the thread splitting



*INode*: Wire-Cell node (Fanout)
[]: *std::vector*
{}: *std::tuple*
types are color coded

# Run Wire-Cell with LArSoft/*art*

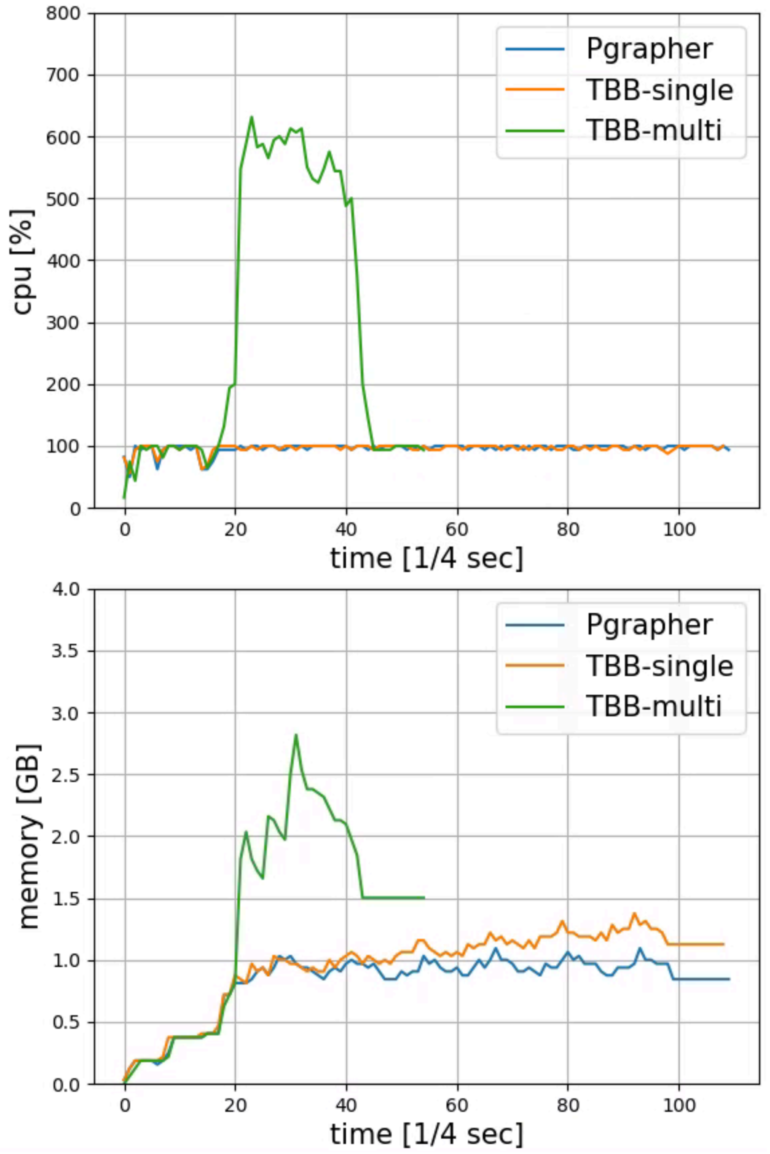
As pointed out in K. Knoepfel, LArSoft Workshop 2019
https://indico.fnal.gov/event/20453/session/8/contribution/12/material/slides/0.pdf
Current (?) LArSoft uses many 'Legacy' services
Chains including them can only run in single threaded mode

We isolated a section does not need any 'Legacy' services for testing:

- Decoded digits $\Longrightarrow$ Signal Processing
- https://github.com/HaiwangYu/wct-analysis/blob/master/exp_data/dec-to-sig.fcl

Very preliminary results - more profiling on-going

1 event



# Next: improve FFT with *FFTW*

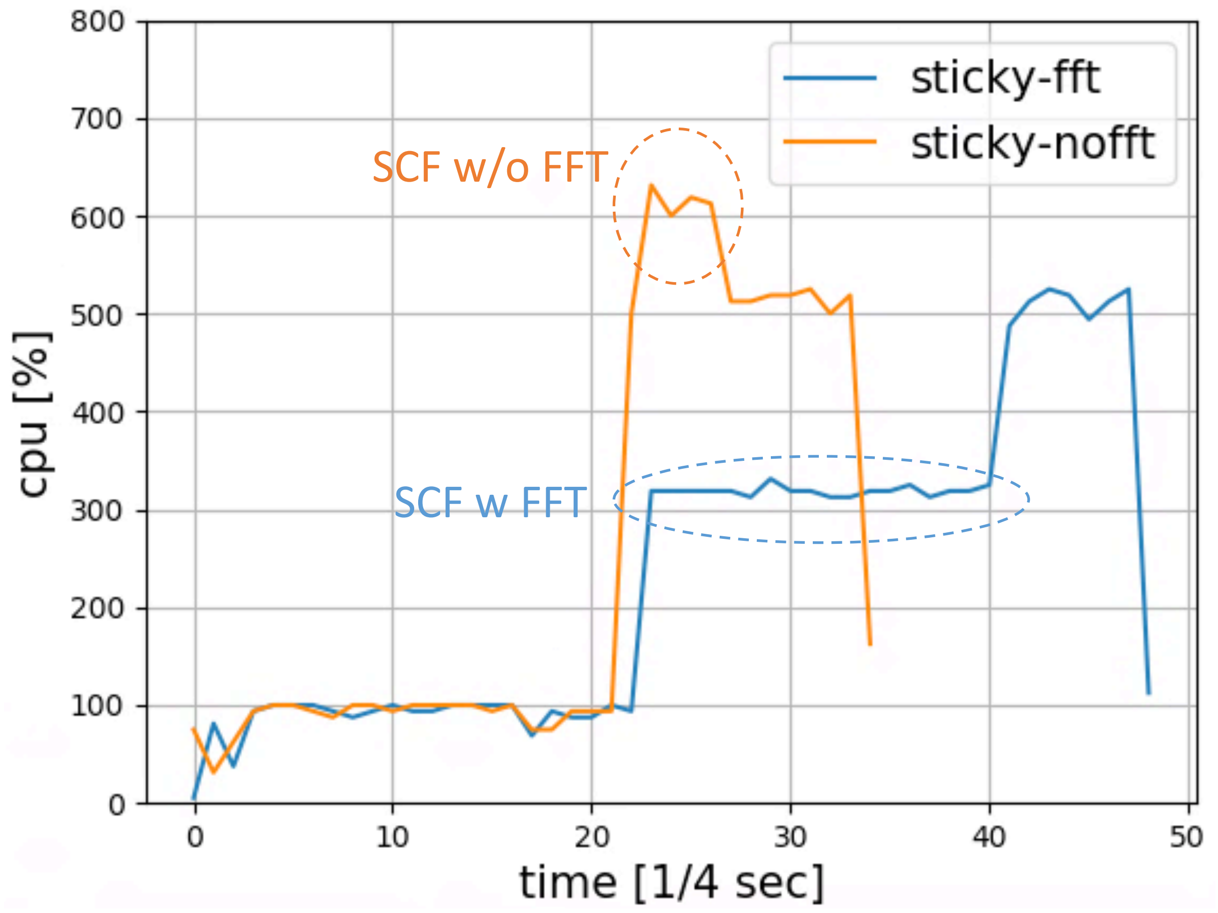
We use *FFTW* as backend of Eigen FFT operations
The FFTW execution is thread safe while the **planner is NOT**

Thanks to the LArSoft team !!
for quickly adding *libfftw*_threads.so* in v3_3_8a
so we could have this test with ups products

For now we use this to add locks around planner calls
*void fftw_make_planner_thread_safe(void)*
This is limiting the CPU usage efficiency in some cases

Will fix this with per-thread planner call instead of locks

Noise filtering with sticky code fix (SCF)


cpu [%]
time [1/4 sec]
sticky-fft
sticky-nofft
SCF w/o FFT
SCF w FFT