Modern Dataflow in Experimental Nuclear Science (and Tcl).

Ron Fox, Giordano Cerizza
Sean Liddick, Aaron Chester

This material is based upon work supported by National Science Foundation.

Talk Outline

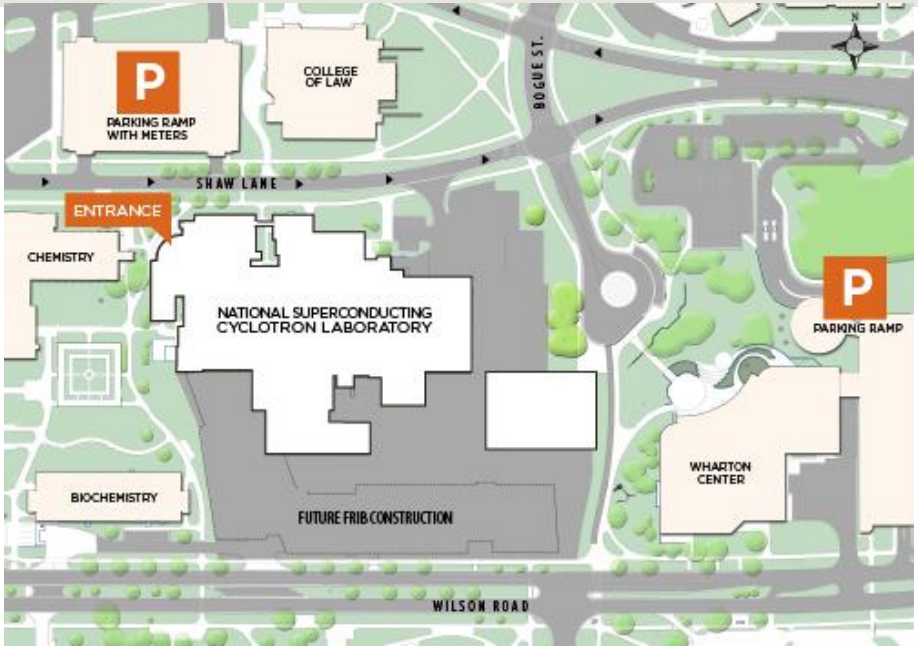
- A bit about me and my Tcl history
- What is the National Superconducting Cyclotron Laboratory (NSCL)
- How data taking has evolved in experimental nuclear science
- E17011 an experiment with modern electronics – why it's computationally demanding
- Parallel resources available to us
- Message Passing Interface (MPI) and Tcl
  - Intro to MPI
  - Existing Tcl support
  - Tcl-Ish support we did.
- Applying MPITcl to an existing application
- What this means for experimental nuclear science at the NSCL

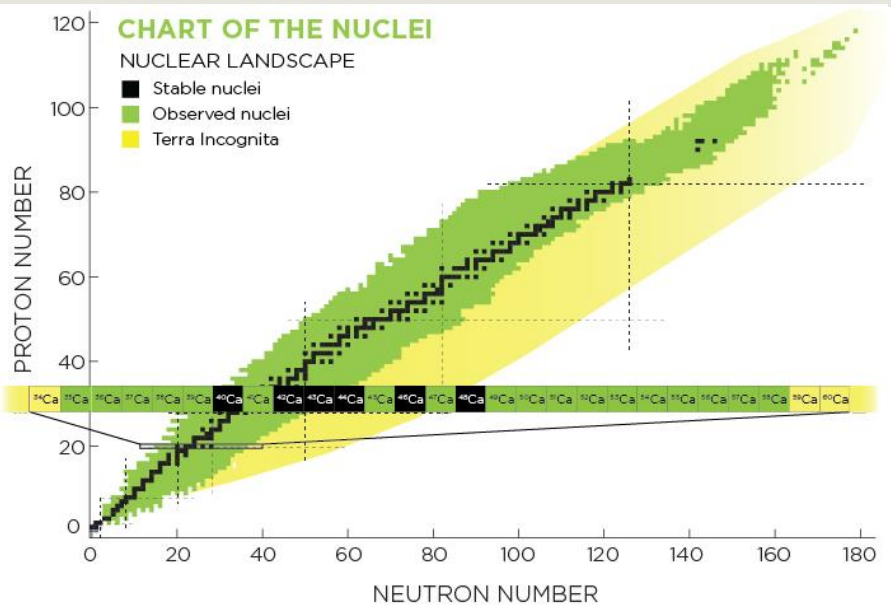
Tcl and me.

- Introduced Tcl/Tk at the National Superconducting Cyclotron Lab (NSCL) back in the 4.x days.
- Plugged into the community with a talk in New Orleans (Tcl 2004)
  - https://www.tcl.tk/community/tcl2004/Papers/RonFox/
  - NSCLSpecTcl – Histogramming package for experimental nuclear science.
- Tcl/Tk conference proceedings editor from Tcl2005 and on if memory serves.
- Tcl plays an important role in the NSCL experimental program.

NSCL

The National Superconducting Cyclotron Lab.



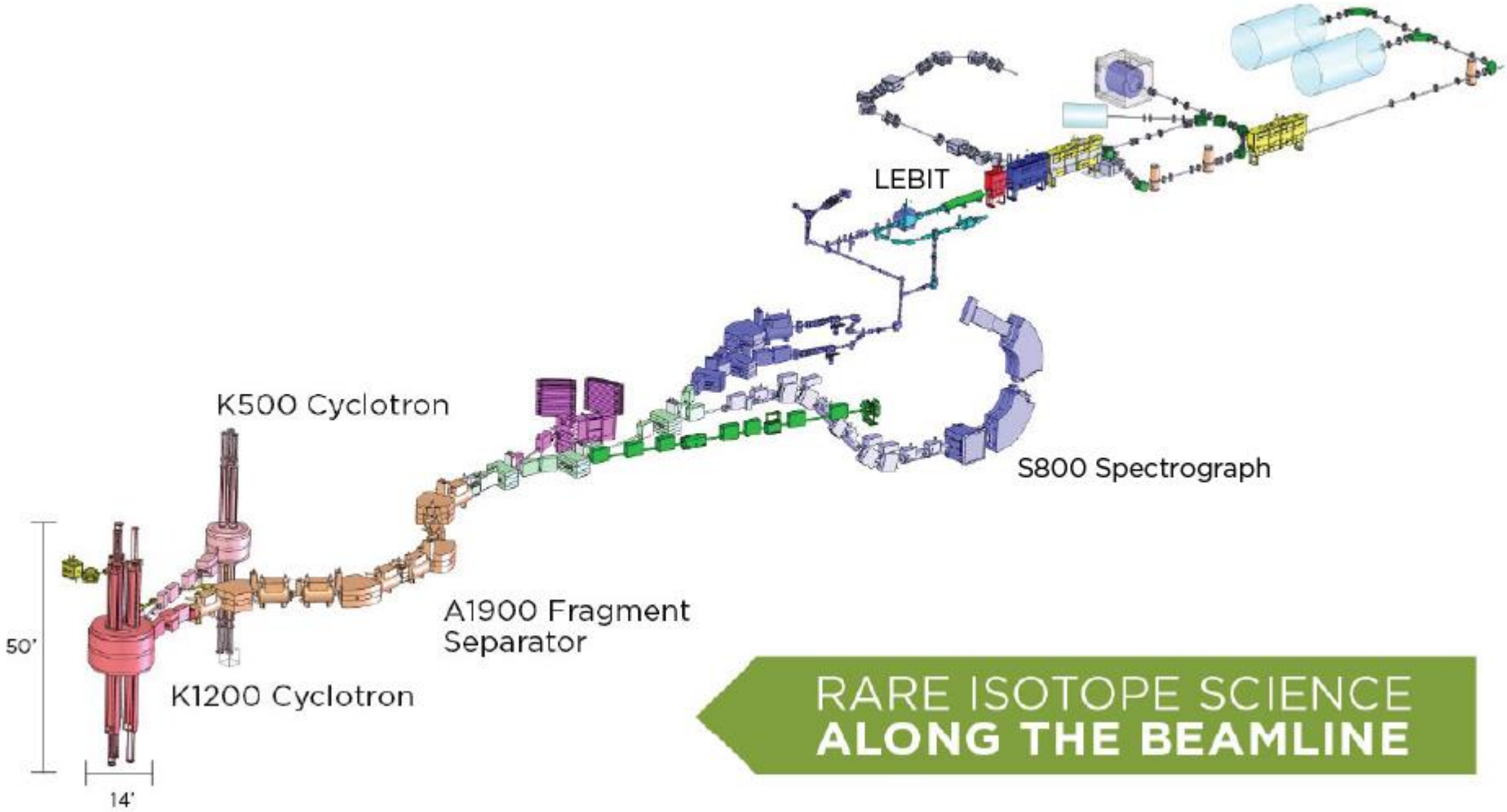


- Located at Michigan State University
- Funded by the National Science Foundation as a user facility

- Explore the properties of nuclear unstable nucleii
- Why and how do certain isotopes form.
- Where do the heavy elements come from?
- http://www.nscl.msu.edu

NSCL

NSCL Block Diagram



Science drivers for Rare Isotope Research

Science drivers (thrusts) from NRC RISAC 2007

Nuclear Structure	Nuclear Astrophysics	Tests of Fundamental Symmetries	Applications of Isotopes

Intellectual challenges from NRC Decadal Study 2013

How does subatomic matter organize itself and what phenomena emerge?	How did visible matter come into being and how does it evolve?	Are fundamental interactions that are basic to the structure of matter fully understood?	How can the knowledge and technological progress provided by nuclear physics best be used to benefit society?

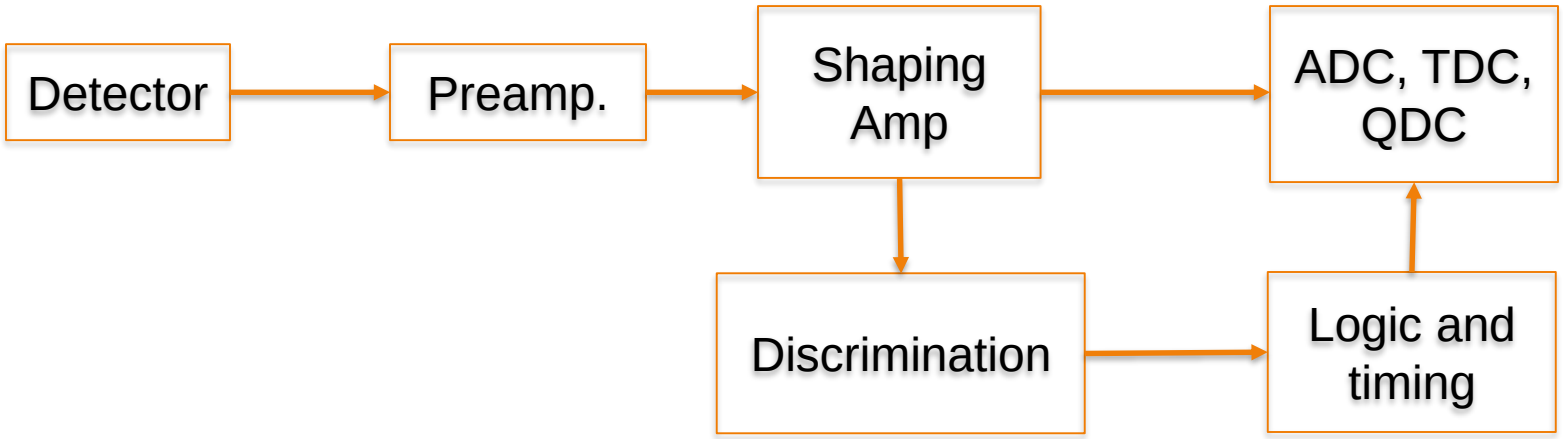
Overarching questions from NSAC Long Range Plan 2015

How are nuclei made and organized? What is the nature of dense nuclear matter?	Where do nuclei and elements come from? What combinations of neutrons and protons can form a bound atomic nucleus? How do neutrinos affect element synthesis?	Are neutrinos their own antiparticles? Why is there more matter than antimatter in the present universe?	What are practical and scientific uses of nuclei?

Overarching questions are answered by rare isotope research

NSCL

Data Acquisition – old school (analog)



Important point – dead-times for a conversion are microseconds

Data Acquisition – old school (analog)

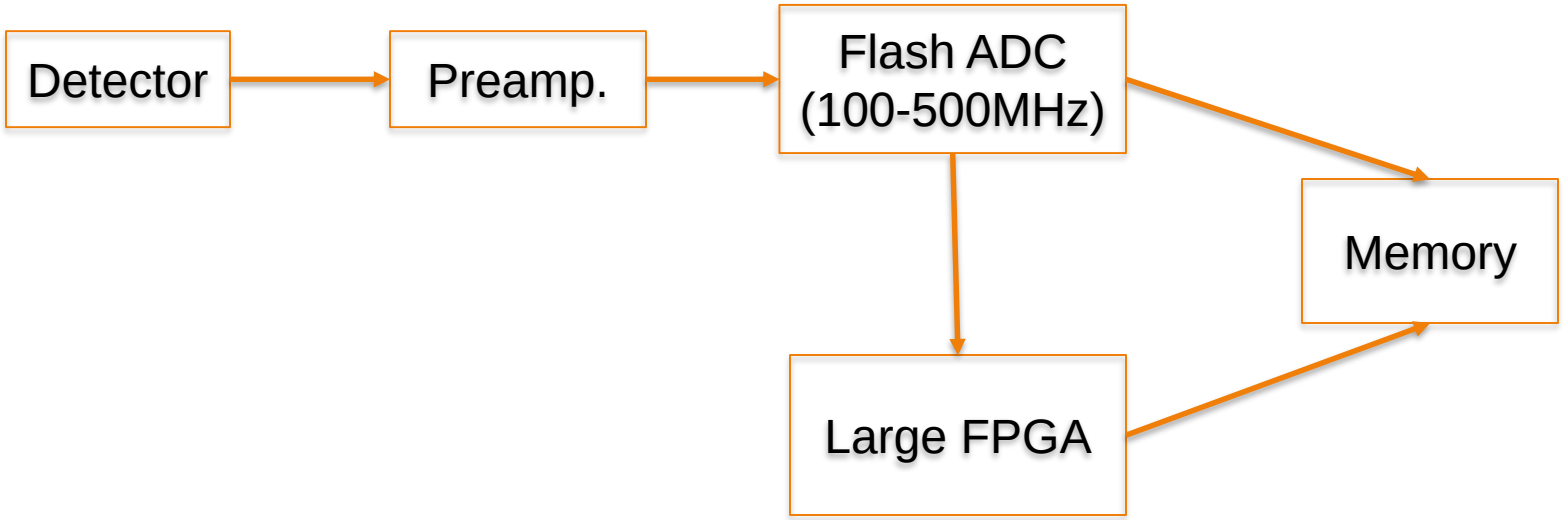
- Detector signals
- Pre-amplification
- Shaping/amplification
- Timing/triggering
- Digitizing modules

Each digitizing module
Gives one value per input:

- Pulse height
- Pulse charge integration
- Pulse timing relative to some reference time.

NSCL

Modern Data Acquisition (digital)



Modern data acquisition (100MHz – 500MHz)

- Detector Signals
- Preamplification
- Digitization

- Firmware can extract
  - Pulse ht.
  - Charge integral
  - Timing
- Keeping waveforms allows experiments that can't be done with analog electronics.
- Wave form analysis is computationally demanding
  Wave forms bloat the data

NSCL

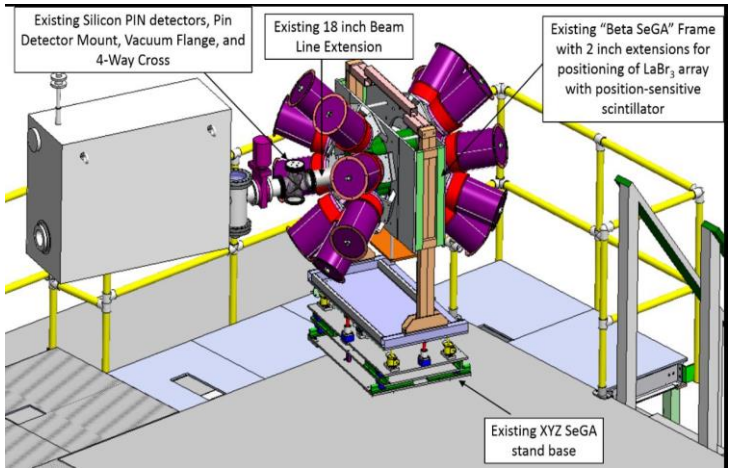
E17011

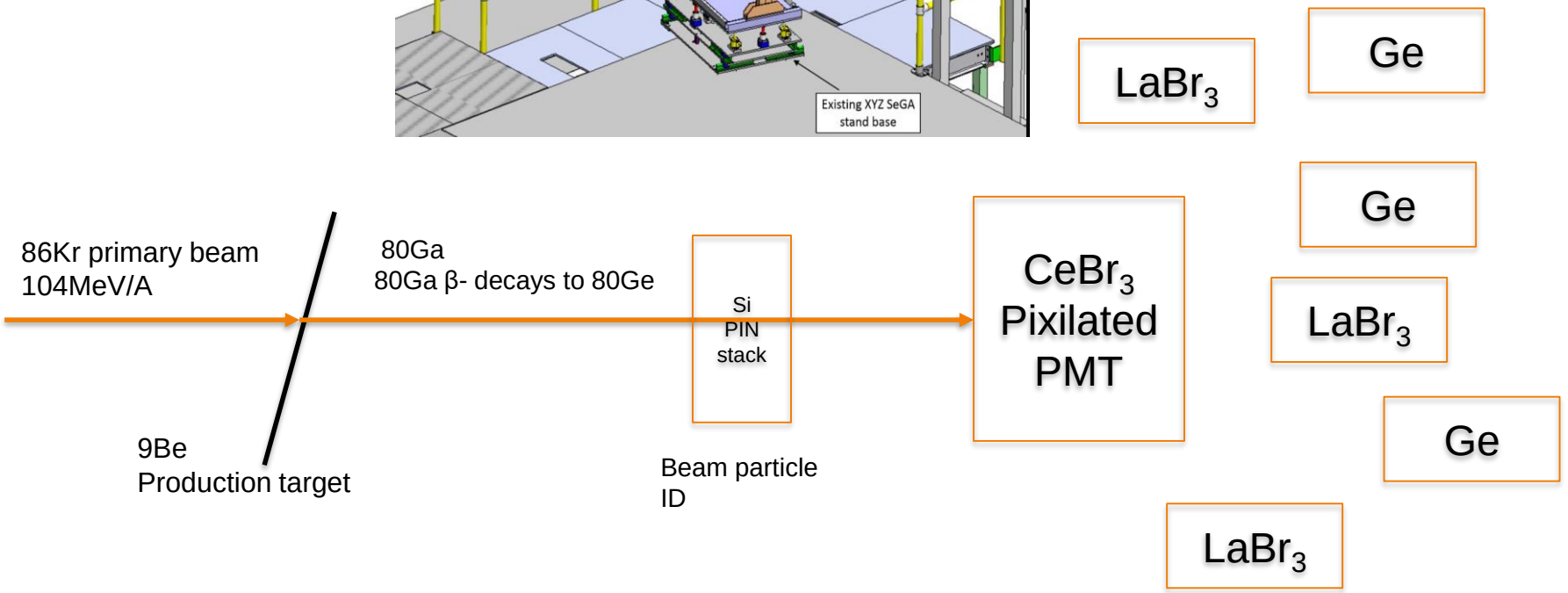
- Scheduled to run in January.
  - Look at beta decay of 80Ga -> 80Ge
  - Look at the lifetime of the 0_2^+ -> 0_1^+
  - Lifetime tell us something about the difference in the radius of the charge distribution of the two states.
- 200MB/second sustained – though modest trigger rate (~3KHz).
- Will take 100TB+ of data
- Need good online and nearline analysis:
  - Are the detectors working.
  - Are we seeing what we think we should be seeing.
  - Should we ask for additional (discretionary time).

NSCL

E17011 – block diagram

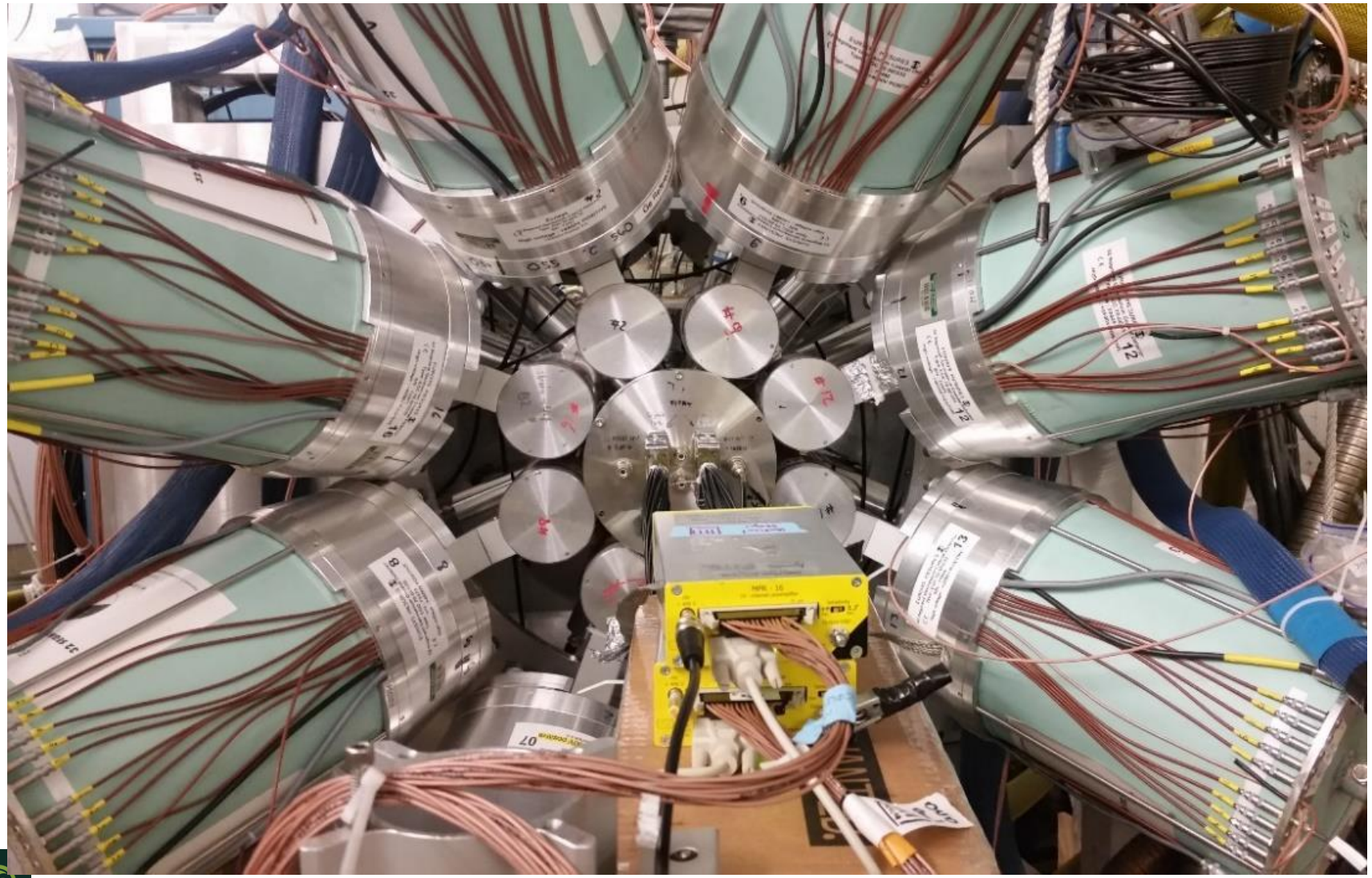
Sketch of experiment





Pictures pictures ($CeBr_3$ and $LaBr_3$ array)

More pictures Ge Array (SeGA)

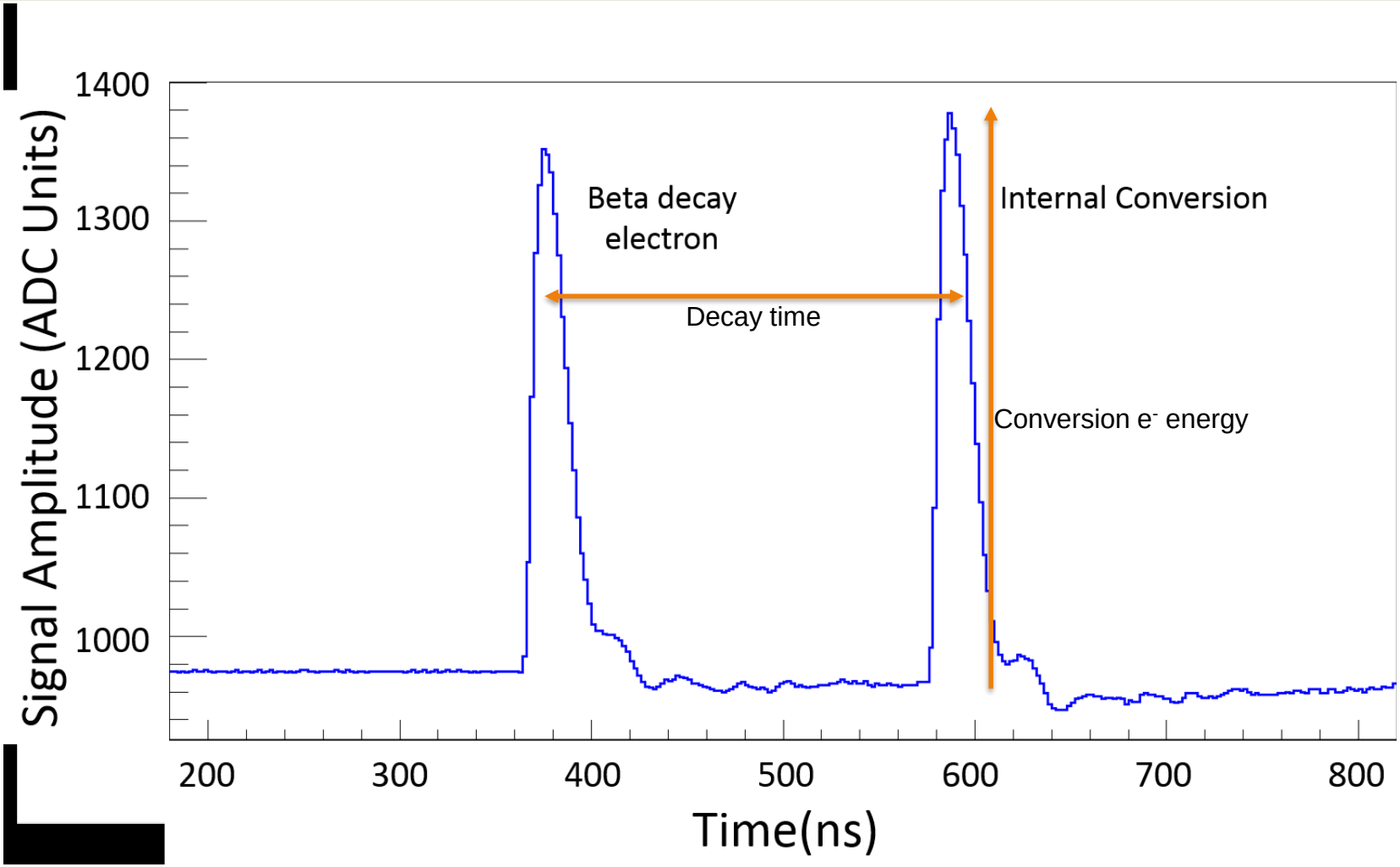


What happens to the implanted ions.

- ^{80}Ga decays to ^{80}Ge by β- decay.
  - This decay is also detected in the $CeBr_3$ detector
  - This decay populates several energy levels of ^{80}Ge
- Of interest are the decays that populate the 0_2^+ state.
  - This eventually de-excites to the 0_1^+ state emitting a γ-ray (detected by the $LaBr_3$ array and/or SeGA) and and a conversion electron.
  - The conversion electron produced by that decay is sensed by the $CeBr_3$
- Well it's not actually eventually.
  - Similar de-excitations have half lives of about 50ns.
  - We want the actual ½ life.
- This is a short ½ life. How to measure it.
  - Digitize the pulses in the $CeBr_3$
    - » Sum signal at 500MHz
    - » pixels at 250MHz
    - » Trace lengths of a few microseconds (on order 100 samples).

NSCL

Sample trace from a similar experiment



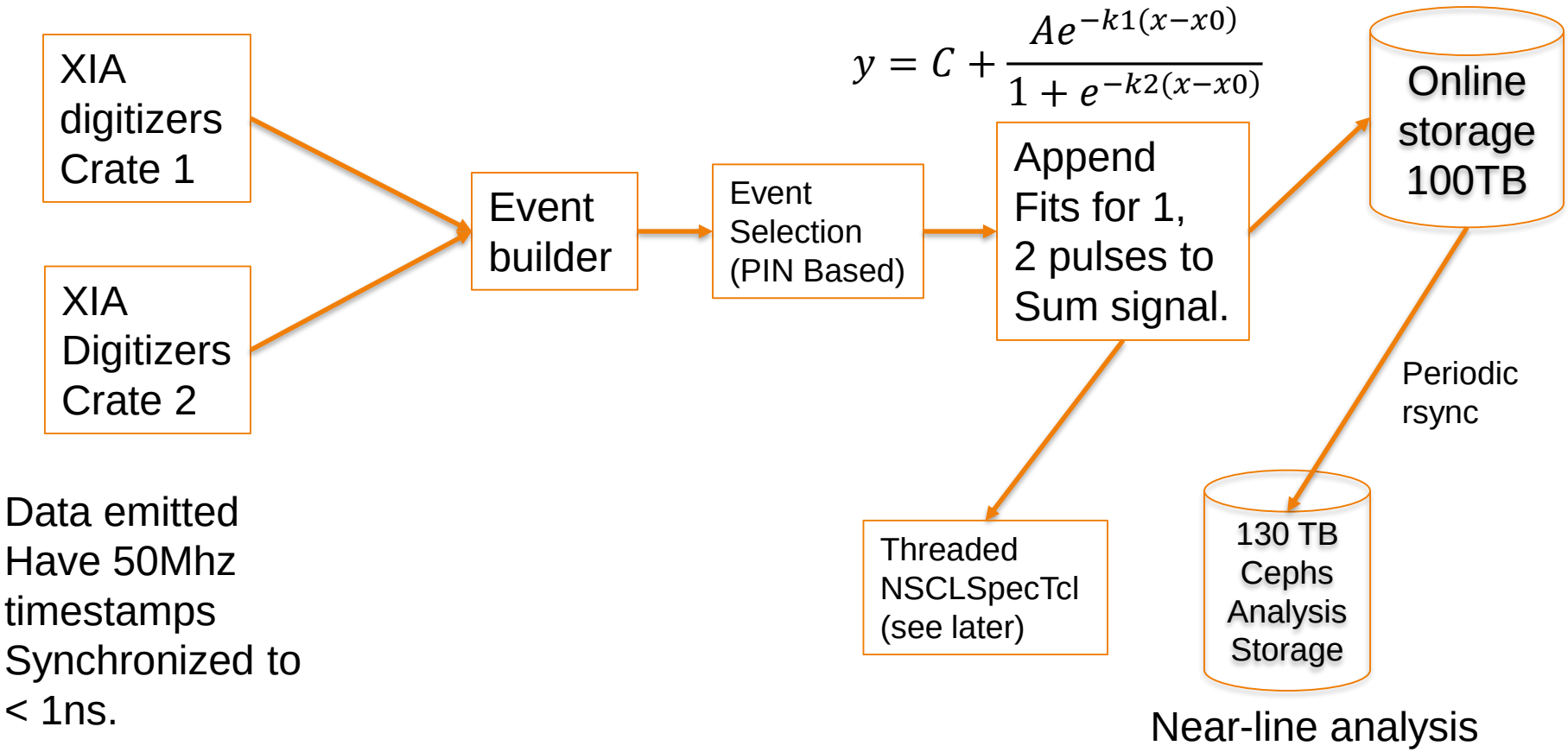
Where does that 200MB/sec come from?

- Since most of the CeBr3 detector lights up for a hit we about 200traces/event (maximal pixel is ‘where’ the event occurred).
- The data rate is dominated by traces from the CeBr3.
- Trigger rates may be 3KHz (modest)
- Data transfer rates will be a sustained 200MB/seconds.
- To see if the experiment is “working” we need to do some processing on all this stuff.
  - Determine if traces are single or double pulses.
  - Determine the characteristics of the pulse(s) – time and height.
- Good news though: Taking traces meas we can do the experiment.

This experiment is *really hard* to do
with old school electronics.

NSCL

Data Flow:



Data emitted Have 50Mhz timestamps Synchronized to < 1ns.

Online analysis

- Fit the sum traces from the CeBr3.
  - Fit for both single and double pulses.
  - Use a heuristic to determine if the pulses are single or double.
- Make a pile of histograms (NSCLSpecTcl) and look at them online
- Keep up with the incoming data rate.

NOTE: Each fit costs 3.5ms to do using GSL's Levenberg-Marquardt.

Serial code isn't going to cut it.

Near-line Analysis – want to keep up with incoming data rate or better

- Fit the remaining traces in the $CeBr_3$
  - Are they single or double pulses (heuristic)?
  - If double pulses extract the time difference as a parameter for histogramming.
- Correlate implantation events with decay events.
  - Using position and particle ID information
  - Timing between implantation and decay.
- These are computationally intensive (e.g. the fit is about 3.5ms/event). To make decisions about the experiment we need to analyze the data already taken faster than acquisition.
- Serial code isn't going to cut it ~2500 cores just for fitting all traces.

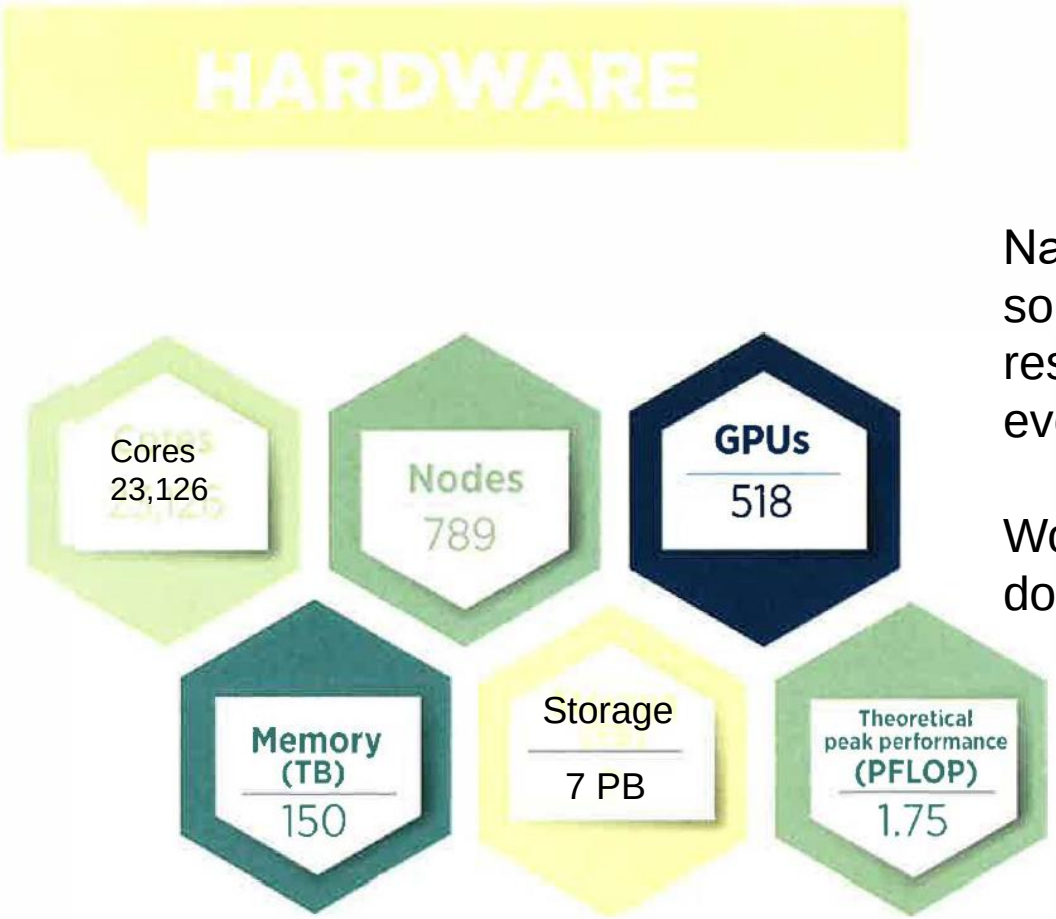
NSCL

Parallel resources at the NSCL available to E17011

- Three high core count systems:
  - 1 26 core system. (Xeon E5-2690 v4 @ 2.60GHz)
  - 2 40 core systems (Xeon Gold 6148 @ 2.4GHz) – bought for this experiment
  - Used for online data flow and interactive ‘near-line’ analysis.
- Modest Linux cluster
  - 360 cores of various ages
  - Used for non-interactive ‘near-line’ partial analysis.
- That’s not going to be enough (to do the fitting of all signals at data rates needs about 2500cores).
- no GPU coprocessors ☹

NSCL

MSU Institute for Cyber Enabled Research (ICER)

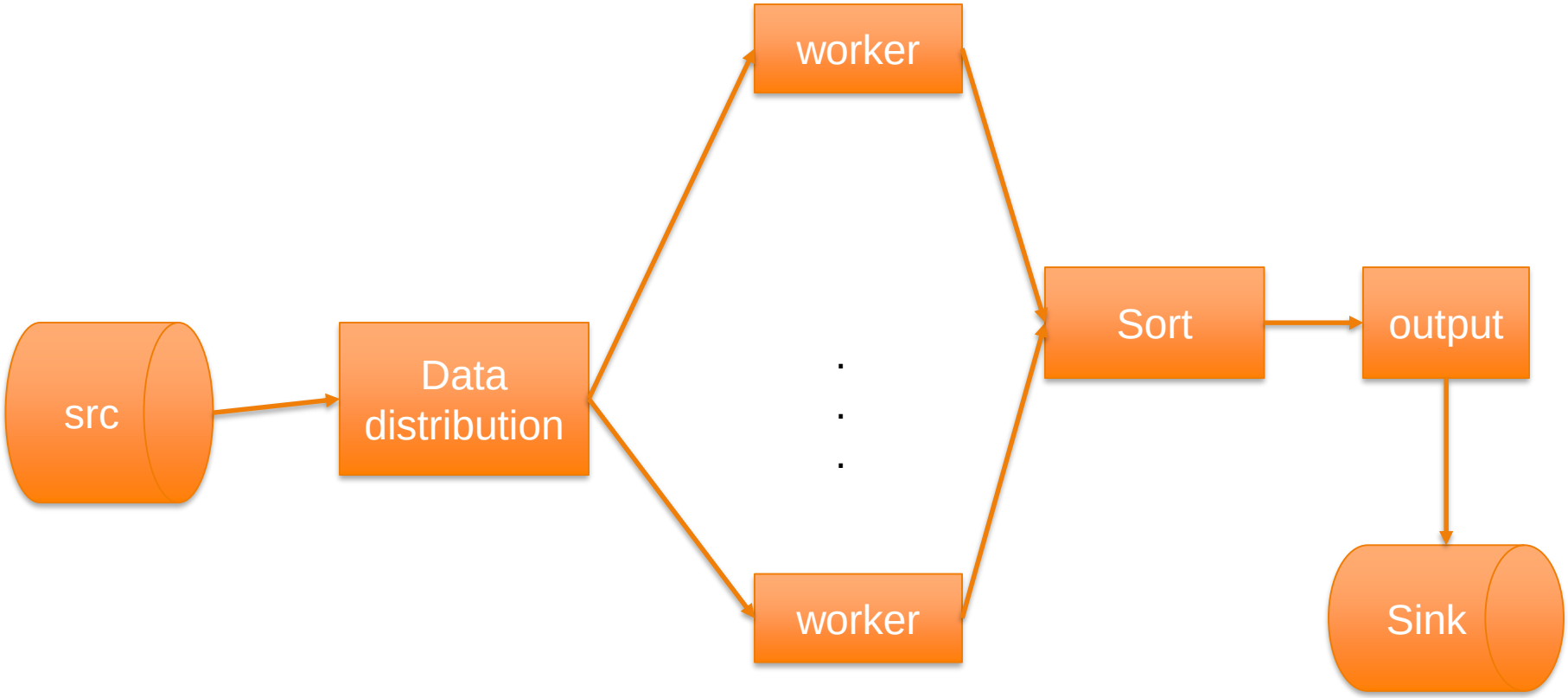


Naturally we've ~~lusted after~~ sought ways to leverage this resource for near-line and maybe even online analysis.

Work to containerize our apps is done (thank you singularity)

Scheduling, however can be an issue: NSCL resources can be dedicated to E17011, ICER is shared across all university users.

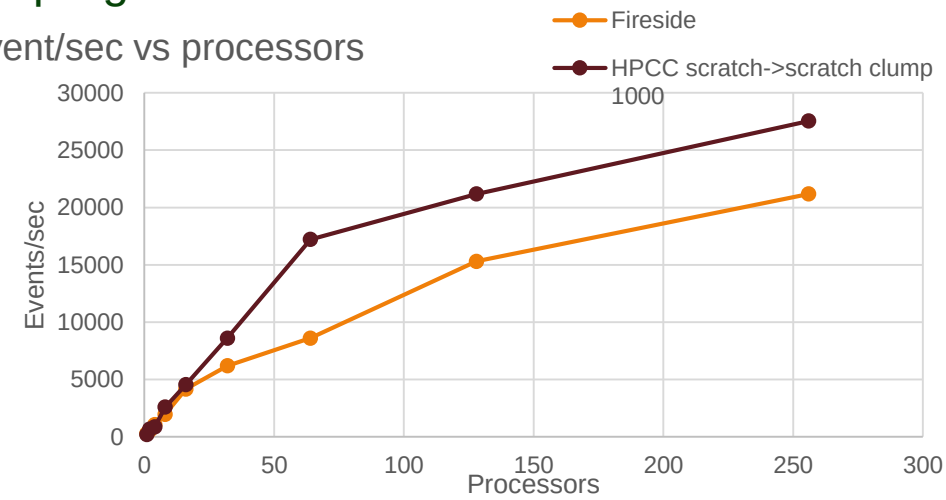
Structure of event analysis parallel programs



Meeting these needs.

- Different types of parallelism
  - Threaded parallelism for the online/interactive stuff.
  - Distributed parallelism for near-line non-interactive stuff.
- Tools to make parallelization simpler
- Fitting:
  - Support for GPU ‘accelerated’ fitting residual and Jacobian computation ☹
  - Machine learning for single/double pulse determination – most traces are single pulses

Example trace fitting the sum signal: same program threaded/cluster



MPI – cluster distributed parallel computing

- MPI - Message Passing Interface standard for writing distributed parallel programs.
  - OpenMPI https://www.open-mpi.org/
  - MPICH https://www.mpich.org/
- Multiple instances of the same process run in parallel.
  - Each process as a *rank* identifying it.
  - Processes can target messages to specific ranks.
  - Communicators can be formed to link groups of processes together.
- Messages require:
  - Rank – who we're sending to but:
  - Communicator – defines the process group in which the rank has meaning (MPI_COMM_WORLD – the entire application is pre-defined).
  - A Tag (integer)
  - Message data.
  - Type of data in the message (message data are strongly typed)
  - Number of items of that type being sent.

NSCL

The MPI API is large and complex:

MPI
MPIX_Allgather_init
MPIX_Allgatherv_init
MPIX_Allreduce_init
MPIX_Alltoall_init
MPIX_Alltoallv_init
MPIX_Alltoallw_init
MPIX_Barrier_init
MPIX_Bcast_init
MPIX_Exscan_init
MPIX_Gather_init
MPIX_Gatherv_init
MPIX_Neighbor_allgather_init
MPIX_Neighbor_allgatherv_init
MPIX_Neighbor_alltoall_init
MPIX_Neighbor_alltoallv_init
MPIX_Neighbor_alltoallw_init
MPIX_Query_cuda_support
MPIX_Reduce_init
MPIX_Reduce_scatter_block_init
MPIX_Reduce_scatter_init
MPIX_Scan_init
MPIX_Scatter_init
MPIX_Scatterv_init
MPI_Abort
MPI_Accumulate
MPI_Add_error_class
MPI_Add_error_code
MPI_Add_error_string
MPI_Address
MPI_Aint_add
MPI_Aint_diff
MPI_Allgather
MPI_Allgatherv
MPI_Alloc_mem
MPI_Allreduce
MPI_Alltoall
MPI_Alltoallv
MPI_Alltoallw
MPI_Attr_delete
MPI_Attr_get
MPI_Attr_put
MPI_File_call_errhandler
MPI_File_close
MPI_File_create_errhandler
MPI_File_delete
MPI_File_f2c
MPI_File_get_amode
MPI_File_get_atomicity
MPI_File_get_byte_offset
MPI_File_get_errhandler
MPI_File_get_group
MPI_File_get_info
MPI_File_get_position
MPI_File_get_position_shared
MPI_File_get_size
MPI_File_get_type_extent
MPI_File_get_view
MPI_File_iread
MPI_File_iread_all
MPI_File_iread_at
MPI_File_iread_at_all
MPI_File_iread_shared
MPI_File_iwrite
MPI_File_iwrite_all
MPI_File_iwrite_at
MPI_File_iwrite_at_all
MPI_File_iwrite_shared
MPI_File_open
MPI_File_preallocate
MPI_File_read
MPI_File_read_all
MPI_File_read_all_begin
MPI_File_read_all_end
MPI_File_read_at
MPI_File_read_at_all
MPI_File_read_at_all_begin
MPI_File_read_at_all_end
MPI_File_read_ordered
MPI_File_read_ordered_begin
MPI_File_read_ordered_end
MPI_File_read_shared
MPI_File_seek
MPI_File_seek_shared
MPI_Ineighbor_allgather
MPI_Ineighbor_allgatherv
MPI_Ineighbor_alltoall
MPI_Ineighbor_alltoallv
MPI_Ineighbor_alltoallw
MPI_Info_c2f
MPI_Info_create
MPI_Info_delete
MPI_Info_dup
MPI_Info_env
MPI_Info_f2c
MPI_Info_free
MPI_Info_get
MPI_Info_get_nkeys
MPI_Info_get_nthkey
MPI_Info_get_valuelen
MPI_Info_set
MPI_Init
MPI_Init_thread
MPI_Initialized
MPI_Intercomm_create
MPI_Intercomm_merge
MPI_Iprobe
MPI_Irecv
MPI_Ireduce
MPI_Ireduce_scatter
MPI_Ireduce_scatter_block
MPI_Irsend
MPI_Is_thread_main
MPI_Iscan
MPI_Iscatter
MPI_Iscatterv
MPI_Isend
MPI_Issend
MPI_Keyval_create
MPI_Keyval_free
MPI_Lookup_name
MPI_Message_c2f
MPI_Message_f2c
MPI_Mprobe
MPI_Mrecv
MPI_Neighbor_allgather
MPI_T_init_thread
MPI_T_pvar_get_info
MPI_T_pvar_get_num
MPI_T_pvar_handle_alloc
MPI_T_pvar_handle_free
MPI_T_pvar_read
MPI_T_pvar_readreset
MPI_T_pvar_reset
MPI_T_pvar_session_create
MPI_T_pvar_session_free
MPI_T_pvar_start
MPI_T_pvar_stop
MPI_T_pvar_write
MPI_Test
MPI_Test_cancelled
MPI_Testall
MPI_Testany
MPI_Testsome
MPI_Topo_test
MPI_Type_c2f
MPI_Type_commit
MPI_Type_contiguous
MPI_Type_create_darray
MPI_Type_create_f90_comp
MPI_Type_create_f90_integ
MPI_Type_create_f90_real
MPI_Type_create_hindexed
MPI_Type_create_hindexed_
MPI_Type_create_hvector
MPI_Type_create_indexed_b
MPI_Type_create_keyval
MPI_Type_create_resized
MPI_Type_create_struct
MPI_Type_create_subarray
MPI_Type_delete_attr
MPI_Type_dup
MPI_Type_extent
MPI_Type_f2c
MPI_Type_free
MPI_Type_free_keyval
MPI_Type_get_attr
MPI_Type_get_contents
MPI_Bcast
MPI_Bsend
MPI_Bsend_init
MPI_Buffer_attach
MPI_Buffer_detach
MPI_Cancel
MPI_Cart_coords
MPI_Cart_create
MPI_Cart_get
MPI_Cart_map
MPI_Cart_rank
MPI_Cart_shift
MPI_Cart_sub
MPI_Cartdim_get
MPI_Close_port
MPI_Comm_accept
MPI_Comm_c2f
MPI_Comm_call_errhandler
MPI_Comm_compare
MPI_Comm_connect
MPI_Comm_create
MPI_Comm_create_errhandler
MPI_Comm_create_group
MPI_Comm_create_keyval
MPI_Comm_delete_attr
MPI_Comm_disconnect
MPI_Comm_dup
MPI_Comm_dup_with_info
MPI_Comm_f2c
MPI_Comm_free
MPI_Comm_free_keyval
MPI_Comm_get_attr
MPI_Comm_get_errhandler
MPI_Comm_get_info
MPI_Comm_get_name
MPI_Comm_get_parent
MPI_Comm_group
MPI_Comm_idup
MPI_Comm_join
MPI_Comm_rank
MPI_Comm_remote_group
MPI_Comm_remote_size
MPI_Comm_set_attr
MPI_Comm_set_errhandler
MPI_Comm_set_info
MPI_Comm_set_name
MPI_Comm_size
MPI_Comm_spawn
MPI_Comm_spawn_multiple
MPI_Comm_split
MPI_Comm_split_type
MPI_File_set_errhandler
MPI_File_set_info
MPI_File_set_size
MPI_File_set_view
MPI_File_sync
MPI_File_write
MPI_File_write_all
MPI_File_write_all_begin
MPI_File_write_all_end
MPI_File_write_at
MPI_File_write_at_all
MPI_File_write_at_all_begin
MPI_File_write_at_all_end
MPI_File_write_ordered
MPI_File_write_ordered_begin
MPI_File_write_ordered_end
MPI_File_write_shared
MPI_Finalize
MPI_Finalized
MPI_Free_mem
MPI_Gather
MPI_Gatherv
MPI_Get
MPI_Get_accumulate
MPI_Get_address
MPI_Get_count
MPI_Get_elements
MPI_Get_elements_x
MPI_Get_library_version
MPI_Get_processor_name
MPI_Get_version
MPI_Graph_create
MPI_Graph_get
MPI_Graph_map
MPI_Graph_neighbors
MPI_Graph_neighbors_count
MPI_Graphdims_get
MPI_Grequest_complete
MPI_Grequest_start
MPI_Group_c2f
MPI_Group_compare
MPI_Group_difference
MPI_Group_excl
MPI_Group_f2c
MPI_Group_free
MPI_Group_incl
MPI_Group_intersection
MPI_Group_range_excl
MPI_Group_range_incl
MPI_Group_rank
MPI_Group_size
MPI_Neighbor_alltoall
MPI_Neighbor_alltoallv
MPI_Neighbor_alltoallw
MPI_Op_c2f
MPI_Op_commutative
MPI_Op_create
MPI_Op_f2c
MPI_Op_free
MPI_Open_port
MPI_Pack
MPI_Pack_external
MPI_Pack_external_size
MPI_Pack_size
MPI_Pcontrol
MPI_Probe
MPI_Publish_name
MPI_Put
MPI_Query_thread
MPI_Raccumulate
MPI_Recv
MPI_Recv_init
MPI_Reduce
MPI_Reduce_local
MPI_Reduce_scatter
MPI_Reduce_scatter_block
MPI_Register_datarep
MPI_Request_c2f
MPI_Request_f2c
MPI_Request_free
MPI_Request_get_status
MPI_Rget
MPI_Rget_accumulate
MPI_Rput
MPI_Rsend
MPI_Rsend_init
MPI_Scan
MPI_Scatter
MPI_Scatterv
MPI_Send
MPI_Send_init
MPI_Sendrecv
MPI_Sendrecv_replace
MPI_Sizeof
MPI_Ssend
MPI_Ssend_init
MPI_Start
MPI_Startall
MPI_Status_c2f
MPI_Status_f2c
MPI_Status_set_cancelled
MPI_Status_set_elements

… and there's more…much more.

Approaches for encapsulating MPI Subset wrapping

- Straight encapsulation of the MPI function interface.
  - Approach taken by Axel Kohlmeyer for the mpi package
  - See e.g. https://core.tcl-lang.org/jenglish/gutter/packages/mpi.html

- Still captures the flavor of the MPI API
- Still exposes explicitly the MPI API subset
- Still exposes the complexity.

What do Tcl MPI applications want to do:

- Send scripts executed in other ranks.
  - Special case of send to all or all others.
- Send data that can be handled by other ranks via callbacks.
  - Again special case of send to all or to others.
- Tcl – We know in advance: Everything has a string representation.
- Binary data – may be sent around by the C/C++ part of the application for C/C++ parts of the application to work on needs a way for that code to shove the TclMPI event handling stuff aside and take over.

NSCL

mpitcl - MPI aware tcl shell.

- Provides MPI aware tclsh.
  - Must be run from mpirun.
- All ranks run this.
- Rank 0 is special – the ‘master’ interpreter it takes input from stdin (normally a file for cluster batch jobs).
- Provides all processes with the mpi namespace in which the mpi command ensemble lives.
- Rank 0 runs a thread to hoist MPI messages received to the event loop (vwait).
- Ranks other than zero run a main loop that accepts MPI messages and act on them under the assumption they come from mpitcl.
- MPI Tags are used to transparently dispatch messages to appropriate handlers.

NSCL

mpi::mpi command ensemble subcommands:

size - How many processes are in the application.
rank – What is my rank in MPI_COMM_WORLD
execute *where script* – Executes a *script* in the rank(s) defined by *where* where is a rank number “all” or “others”
send *where data* - sends the *data* to *where*
handle *script* - specifies *script* to handle data received. The script receives two parameters: sender rank and the data.
stopnotifier - only legal in rank 0 – stops the event notifier thread.
startnotifier - only legal in rank 0 – starts the notifier thread again.

Sample mpitcl scripts:

```
mpi::mpi stop notifier
mpi::mpi send all exit
```

Mimimal MPI script

```
proc receiver {rank data} {
    puts "Received from $rank '$data'"
    incr ::slaves -1
}

set slaves [mpi::mpi size]
incr slaves -1;          # number of slave processes.

mpi::mpi handle receiver

mpi::mpi execute others {
    mpi::mpi send 0 "Rank [mpi::mpi rank] is alive"
}

while {$slaves} {
    vwait slaves
}

mpi::mpi stopnotifier
mpi::mpi execute all exit
```

Soliciting/getting data from workers

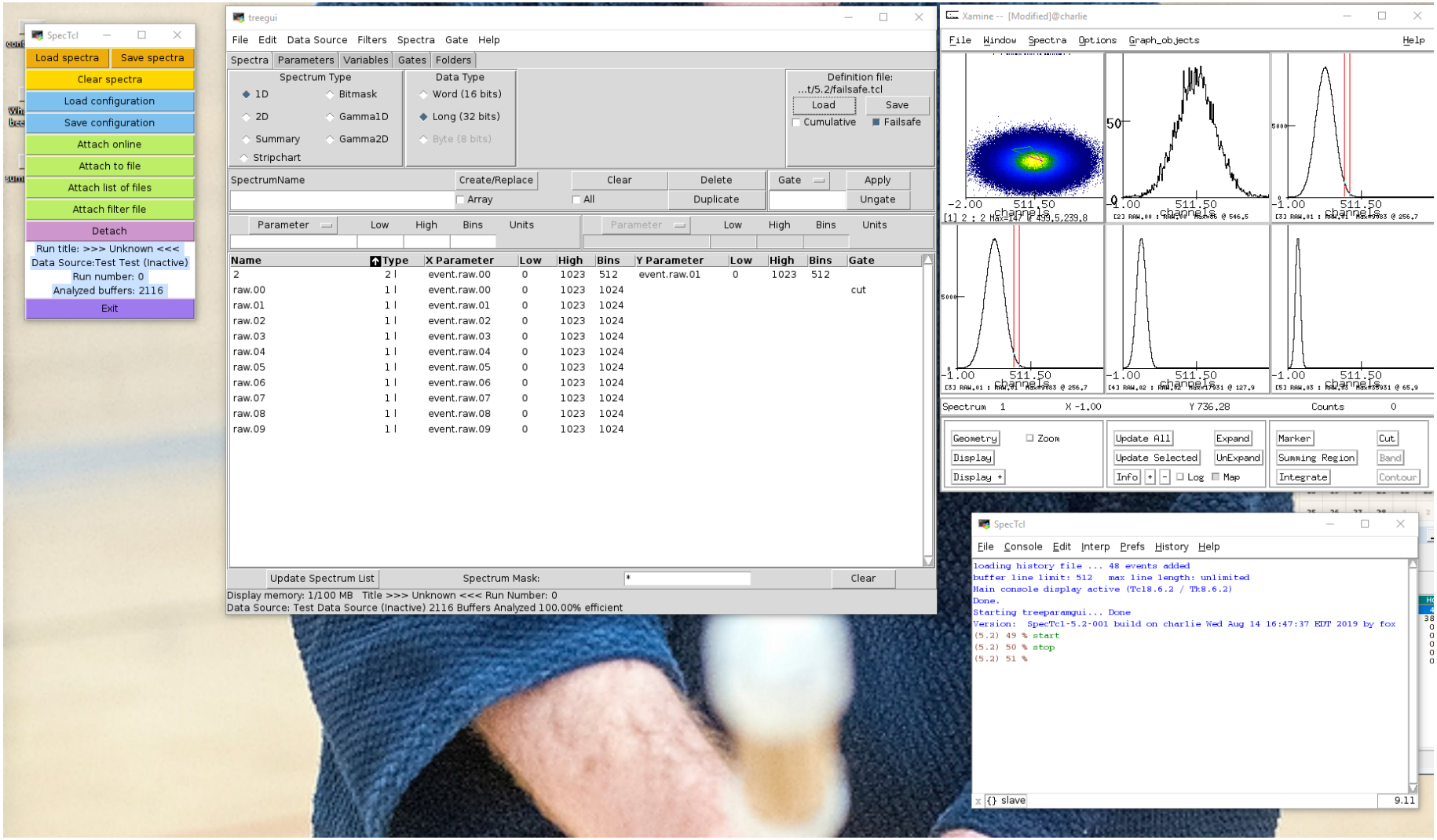
NSCL

Applying tclmpi – The Circle is complete.

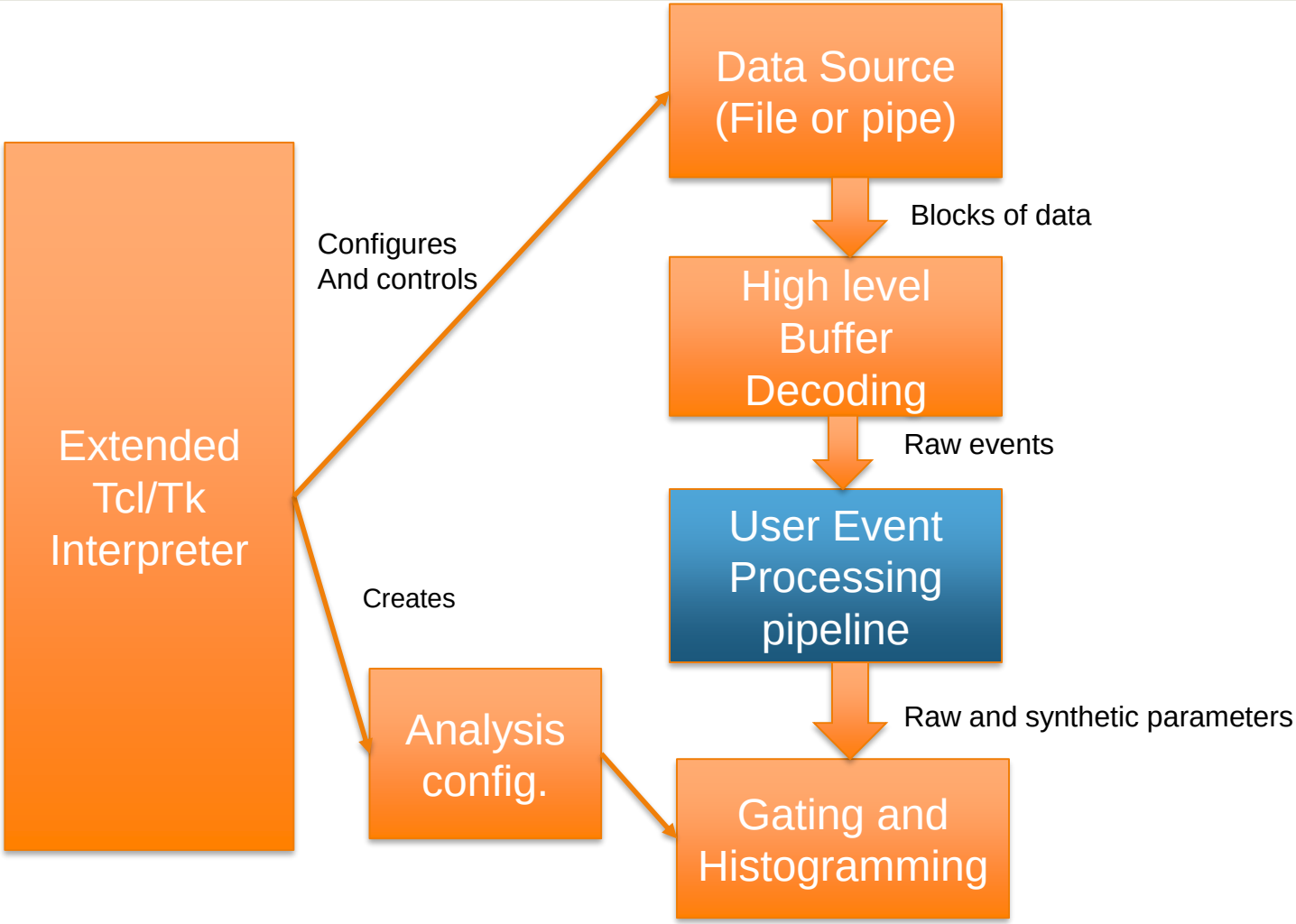
- NSCLSpectcl
- Structure
- Parallelization of the interactive version (threaded).
- Creating a batch NSCLSpecTcl and using it with tclmpi.

NSCL

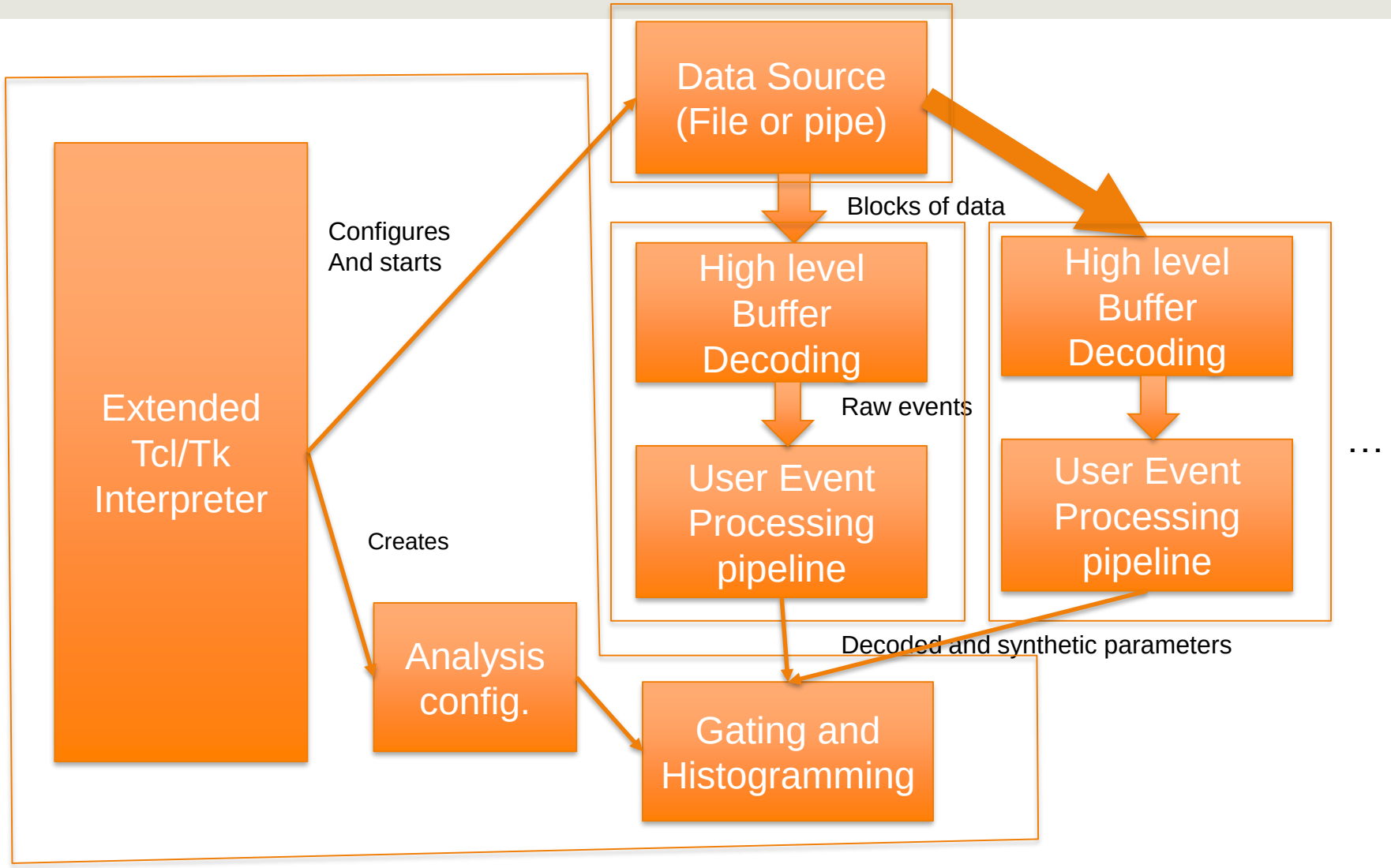
NSCLSpecTcl is highly interactive



Simplified NSCLSpecTcl structure and parallelization approaches.

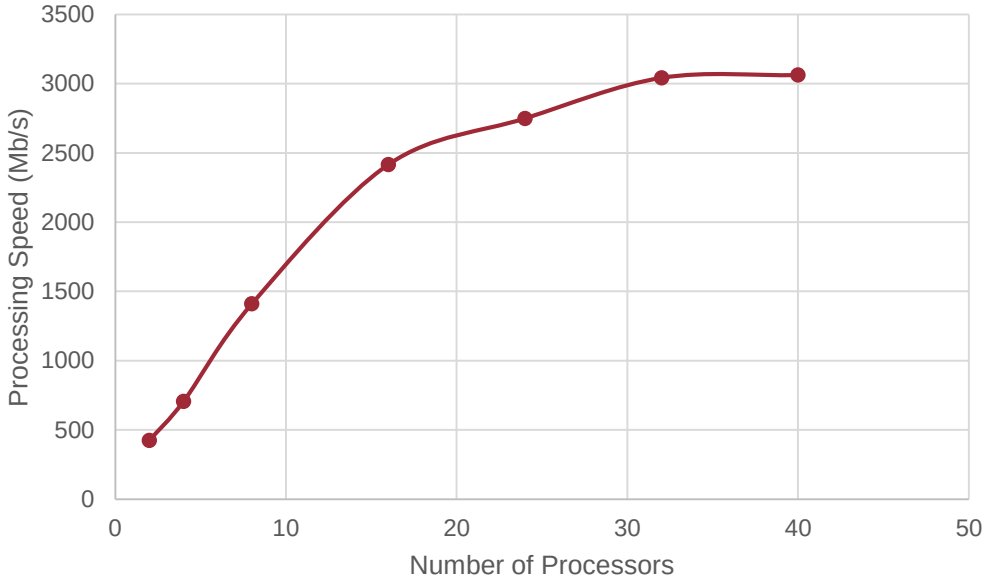


Threaded Parallelism (Giordano Cerizza)



User code must be thread-safe

Threaded Spectcl performance



- Roll off is at the performance limit of the SSD that contained the data.
- Analysis pipeline for this case is simple compared with E17011's.
- Good scaling up until SSD transfer limits.

MPI parallelism?

- We can recruit more cores if the application scales.
- We don’t have to worry about thread safety since it’s process parallelism.
- With an assist from container technology (e.g. singularity) we can get outside the NSCL to supercomputer centers (or ICER e.g.) with even more cores.
- BUT In almost all cases cluster computing doesn’t allow dynamic interactivity.
  - Needed to turn NSCLSpecTcl into a batch program.
  - Needed to figure out how to easily parallelize it.
  - This is the original target of mpitcl.

MPI NSCLSpecTcl

- Each process is a complete batch NSCLSpecTcl
  - Batch NSCLSpecTcl is three packages
    - » spectcl – the base application code.
    - » mpispectcl MPI data sources and sinks.
    - » A user supplied package implementing the processing pipeline.
  - Batch/MPI NSCLSpecTcl has generalized data sources and sinks. **analyze** command sends blocks of events from source to sink. I've implemented:
    - » Source –file.
    - » Source – MPI (for workers – requests block of data from rank 0).
    - » Sink -- Analysis
    - » Sink – MPI (Distributes blocks of data to workers using MPI source).
- Rank 0 :
  - Tells each process (including itself) to read in the configuration scripts.
  - Tells other process to use an MPI Source and Analysis sink
  - Tells itself to use a file data source and MPI sink.
  - Tells everyone to start analyzing data.
- When analysis is complete Rank 0
  - Tells all other processes to send it spectrum data.
  - Sums the spectra into total spectra
  - Writes them out for visualization.

NSCL

What this looks like:

```
mpi::mpi execute all {
    package require spectcl
    package require mpispectcl
    package require MyPipeline;   # User event processing code is here.
    source defs.tcl
}

mpi::mpi execute others {
    mpisource
    analysissink
}

filesource run-0003-00.evt
mpisink
mpi::mpi stopnotifier

mpi::mpi execute others analyze
analyze

mpi::mpi startnotifier
```

Getting the data back:

```
set l [spectrum -list]
set f [open spectra.dat w]

foreach spectrum $l {
    set name [lindex $spectrum 1]
    getSpectrumFromWorkers $name
    swrite -format ascii $f $name;   # Writes a histo to file.
}

close $f
```

Summing a spectrum from worker nodes.

```
proc addData {name src data} {;          # sums the histo from 1 worker into the local histo.
    foreach datum $data {
        set value [lindex $datum end]
        set coords [lrange $datum 0 end-1]
        set current [channel -get $name $coords]

        incr current $value
        channel -set $name $coords $current
    }
    incr ::expected -1

}

proc getSpectrumFromWorkers name {
    mpi::mpi handle [list addData $name]
    set ::expected [mpi::mpi size]
    incr ::expected -1;

    set script "mpi::mpi send 0;    # Care must be taken to ensure substitutions are done
    append script "\[“          ;    # in the right process.
    append script "scontents $name]"
    mpi::mpi execute others $script
    while {$::expected > 0} {
        vwait ::expected;
    }
}
```

NSCL

Does this work?

10Gbyte file with 2,754,450 events.					
Workers	Time	MB/sec	Events/sec		
1	25.4	403.1496	108442.91		
2	13.5	758.5185	204033.33		
4	6.6	1551.515	417340.91		
8	3.9	2625.641	706269.23		

- 2.6Gbytes/sec is interconnect saturation (dual 10GBit/sec ethernet)
- Event processing in the actual experiment
  will be more complex but we can scale to more workers.

Conclusions

- NSCL's Transition to modern, digital nuclear electronics poses problems for online and near-line data analysis.
- Experiments will increasingly require parallelism in online and near-line data handling:
  - Software tools to make it easy for naïve users to make use of parallelism by plugging in their event analysis code.
  - Large core count systems for interactive, online analysis (threaded parallel).
  - Clusters dedicated to the running experiment for near-line analysis (distributed parallel).
  - High speed interconnects to support the data flow bandwidth.
  - Large, multi-petabyte storage that's fast with fast interconnects.
- E17011 provides a laboratory to explore the techniques we'll need to apply to modern experiments.
- mpitcl is one technique to simplify parallel programming for "the masses"
  - Easly retrofitted a highly interactive, complex serial analysis program (2 days work)
  - Got scaling up to the interconnect bandwidth.

NSCL