EE3801 Data Engineering Principles

Optimizing Parallel Processing

Optimizing Parallel Processing

Lab 7 Techniques

Lab 7

For this lab, you can opt to use the scripts you created in Lab 6 if you did not have any problems. However, we have also added a version of the scripts to the PyHipp repository, so if you would like to use those versions instead, you will need to rename your version so it does not create a conflict when you merge the changes in the upstream repository into your local repository (remember to create a cluster from your EC2 instance and not your computer):

```
(env1) [ec2-user@ip-10-0-5-43 PyHipp] $ mv rplparallel-slurm.sh myrplparallel-slurm.sh
```

The same goes for these other scripts you created in Lab 6:

```
rplsplit-slurm.sh   checkfiles.sh
rs1-slurm.sh        removefiles.sh
rs2-slurm.sh        checkfiles2.sh
rs3-slurm.sh        pipe2.sh
rs4-slurm.sh
```

Warning

If you choose to use the files in the upstream repository, remember to replace the AWS account number used in these scripts with your own account number.

You can then merge your repository with the changes in the upstream repository:

```
(env1) [ec2-user@ip-10-0-5-43 PyHipp] $ git fetch upstream

(env1) [ec2-user@ip-10-0-5-43 PyHipp] $ git checkout main

(env1) [ec2-user@ip-10-0-5-43 PyHipp] $ git merge upstream/main
```

You can add your versions of the scripts to your GitHub repository (remember to set the default editor to nano as in Lab 6):

```
(env1) [ec2-user@ip-10-0-5-43 PyHipp] $ git add my*.sh

(env1) [ec2-user@ip-10-0-5-43 PyHipp] $ git commit
```

Optimizing Parallel Processing

Lab 6 Techniques

Lab 6

Create a personal access token (Classic) to use with GitHub here (scroll down to find the section titled "Creating a personal access token (classic)").

You can push the pipeline-slurm.sh file to GitHub by doing:

```
(env1) [ec2-user@ip-10-0-5-43 PyHipp] $ git config --global core.editor "nano"

(env1) [ec2-user@ip-10-0-5-43 PyHipp] $ git add pipeline-slurm.sh

(env1) [ec2-user@ip-10-0-5-43 PyHipp] $ git commit
```

At this point, a nano editor window will open up for you to add a commit message. It is good practice to add a concise description of what changes were made in the nano window so in the future, you can look back and understand what was done. Once you are done, you can exit nano the usual way, making sure to save the file with the suggested filename. After that, you can push the scripts to your GitHub repository by doing:

```
(env1) [ec2-user@ip-10-0-5-43 PyHipp] $ git push
```

You will then be prompted to enter your GitHub username and token.

Optimizing Parallel Processing

Lab 7 Techniques

- Bash
  - Use multiple conda environments to work around software with file locking
  - Create and use Python program to manage conda environments

Optimizing Parallel Processing

Resource Management

```
#!/bin/bash

# Submit this script with: sbatch <this-filename>

#SBATCH --time=24:00:00   # walltime
#SBATCH --ntasks=1   # number of processor cores (i.e. tasks)
#SBATCH --nodes=1   # number of nodes
#SBATCH -J "rplhps"   # job name

## /SBATCH -p general # partition (queue)
#SBATCH -o rplhps-slurm.%N.%j.out # STDOUT
#SBATCH -e rplhps-slurm.%N.%j.err # STDERR

# LOAD MODULES, INSERT CODE, AND RUN YOUR PROGRAMS HERE
/data/miniconda3/bin/conda init
source ~/.bashrc
envarg=`/data/src/PyHipp/envlist.py`
conda activate $envarg

python -u -c "import PyHipp as pyh; import time; pyh.RPLHighPass(saveLevel = 1);
 from PyHipp import mountain_batch; mountain_batch.mountain_batch(); from PyHipp
 import export_mountain_cells; export_mountain_cells.export_mountain_cells(); print(time.localtime());"

conda deactivate
/data/src/PyHipp/envlist.py $envarg
```

Optimizing Parallel Processing

Resource Management

- Create Python program to manage environments: envlist.py
- Usage 1: envlist.py env_prefix number_of_environments
- Create list of environment names in Python
  - envlist.py cenv 64
  - [cenv0, cenv1, cenv2, … cenv63]
  - saved in /data/picasso/envlist.hkl

Optimizing Parallel Processing

Resource Management

```
with lock:
    if pmode == RESET_MODE:
        clist1 = [*range(0,int(nenvs),1)]
        clist = [envprefix + str(s) for s in clist1]
    else:
        # load hickle file
        clist = hickle.load(file_path)

        if pmode == WRITE_MODE:
            # append item to end of list
            clist.append(env)
        else:
            # pop first item off list
            env = clist.pop(0)
            # return env name
            print(env)

    # save hickle file
    hickle.dump(clist, file_path, mode="w")
```

Optimizing Parallel Processing

Resource Management

- Usage 2: envlist.py
- Use file locking to make sure only 1 job can access list at a time to get environment name
  - removes first name from the list, and returns it
  - envarg=`envlist.py` Runs the command within `` and sets the output to the variable envarg
  - echo $envarg
  - cenv0
  - [cenv1, cenv2, … cenv63]

Optimizing Parallel Processing

Resource Management

```
with lock:
    if pmode == RESET_MODE:
        clist1 = [*range(0,int(nenvs),1)]
        clist = [envprefix + str(s) for s in clist1]
    else:
        # load hickle file
        clist = hickle.load(file_path)

        if pmode == WRITE_MODE:
            # append item to end of list
            clist.append(env)
        else:
            # pop first item off list
            env = clist.pop(0)
            # return env name
            print(env)

    # save hickle file
    hickle.dump(clist, file_path, mode="w")
```

Optimizing Parallel Processing

Resource Management

- Usage 3: envlist.py env_name_to_return
- Use file locking to make sure only 1 job can access list at a time to return environment name
  - initial list: [cenv1, cenv2, … cenv63]
  - envlist.py cenv0
  - [cenv1, cenv2, … cenv63, cenv0]
  - appends name to the end

Optimizing Parallel Processing

Resource Management

```
with lock:
    if pmode == RESET_MODE:
        clist1 = [*range(0,int(nenvs),1)]
        clist = [envprefix + str(s) for s in clist1]
    else:
        # load hickle file
        clist = hickle.load(file_path)

        if pmode == WRITE_MODE:
            # append item to end of list
            clist.append(env)
        else:
            # pop first item off list
            env = clist.pop(0)
            # return env name
            print(env)

    # save hickle file
    hickle.dump(clist, file_path, mode="w")
```

Optimizing Parallel Processing

Lab 7 Techniques

- Bash
  - Use multiple conda environments to work around software with file locking
  - Create and use Python program to manage conda environments
  - Use sort command to organize output
    - `find session01 -name "hps*out" -or -name "lfp*out" | xargs tail -n 1 | sort`

Optimizing Parallel Processing

Lab 7 Techniques

- Bash
  - Use multiple conda environments to work around software with file locking
  - Create and use Python program to manage conda environments
  - Use sort command to organize output
    - `find session01 -name "hps*out" -or -name "lfp*out" | xargs tail -n 1 | sort`
  - Use cut command to extract output
    - `echo $filename`
    - `./session01/array04/channel123/rplhighpass_b59f.hkl`
    - `echo $filename | cut -d "/" -f 1-4`
    - `./session01/array04/channel123`

Optimizing Parallel Processing

Lab 7 Techniques

- Bash
  - Use comm command to find missing output
    - `find . -name "channel*" | grep -v -e eye -e mountain | sort > chs.txt`
    - `find . -name "rplhighpass*hkl" | grep -v -e eye | sort | cut -d "/" -f 1-4 > hps.txt`
    - `comm -23 chs.txt hps.txt`

Optimizing Parallel Processing

Lab 7 Techniques

https://man7.org/linux/man-pages/man1/comm.1.html

man7.org > Linux > man-pages Linux/UNIX system programming traini

comm(1) — Linux manual page

NAME | SYNOPSIS | DESCRIPTION | EXAMPLES | AUTHOR | REPORTING BUGS | COPYRIGHT | SEE ALSO | COLOPHON

Search online pages

COMM(1) User Commands COMM(1)

NAME top

comm - compare two sorted files line by line

SYNOPSIS top

comm [*OPTION*]... *FILE1 FILE2*

DESCRIPTION top

Compare sorted files FILE1 and FILE2 line by line.

When FILE1 or FILE2 (not both) is -, read standard input.

With no options, produce three-column output. Column one contains lines unique to FILE1, column two contains lines unique to FILE2, and column three contains lines common to both files.

-1 suppress column 1 (lines unique to FILE1)

-2 suppress column 2 (lines unique to FILE2)

-3 suppress column 3 (lines that appear in both files)

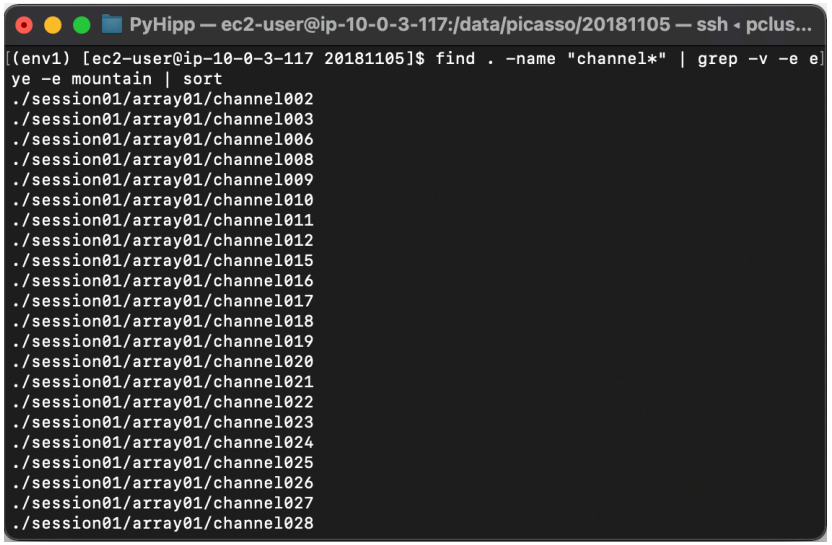
Optimizing Parallel Processing

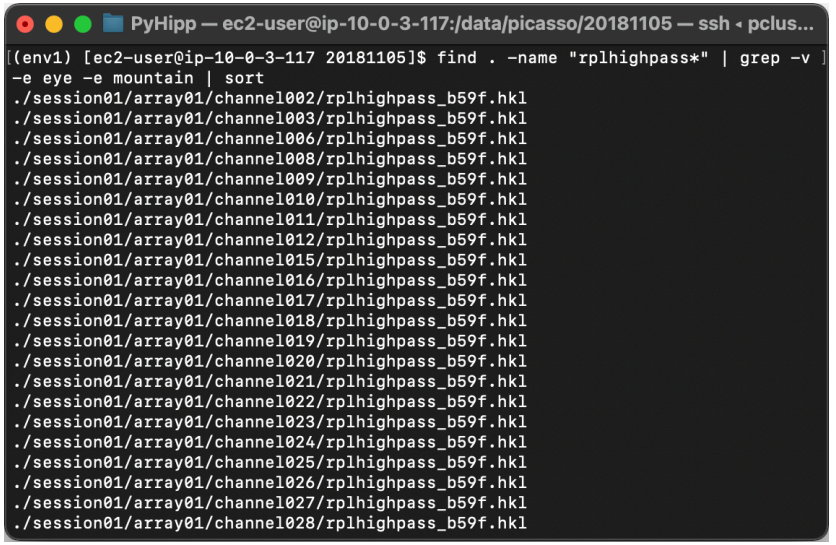
Lab 7 Techniques

- Bash
  - Use comm command to find missing output
    - `find . -name "channel*" | grep -v -e eye -e mountain | sort > chs.txt`
    - `find . -name "rplhighpass*hkl" | grep -v -e eye | sort | cut -d "/" -f 1-4 > hps.txt`
    - `comm -23 chs.txt hps.txt`
  - Using for-loops
    - `cwd=`pwd`; for i in `comm -23 chs.txt hps.txt`; do echo $i; cd $i; sbatch /data/src/PyHipp/rplhighpass-sort-slurm.sh; cd $cwd; done`
    - `for i in 2018110[12]; do echo $i; cd $i; bash /data/src/PyHipp/pipe2a.sh; cd ..; done`
    - `for i in 20180??? 201810??; do echo $i; cd $i; sbatch /data/src/PyHipp/rplparallel-slurm.sh; cd ..; done`

Optimizing Parallel Processing

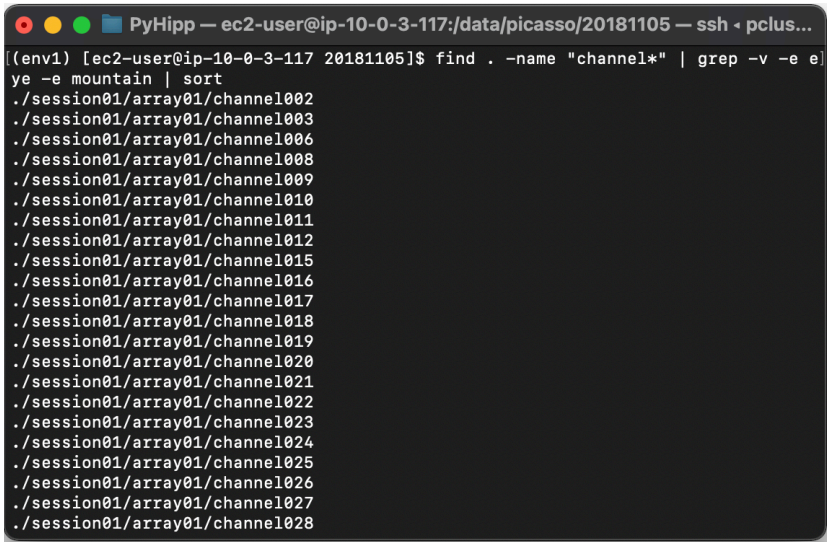
Lab 7 Techniques

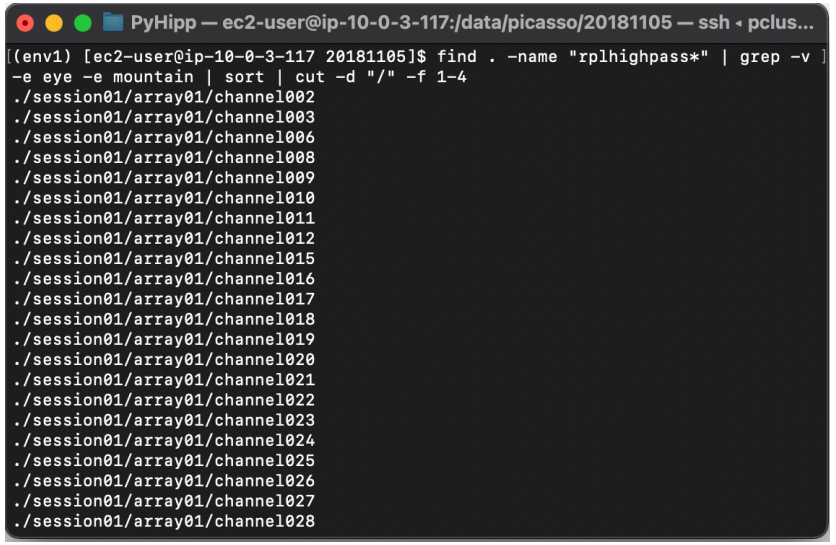




Optimizing Parallel Processing

Lab 7 Techniques





Optimizing Parallel Processing

Lab 7 Techniques

PyHipp — ec2-user@ip-10-0-3-117:/data/picasso/20181105 — ssh ◂ pclus...

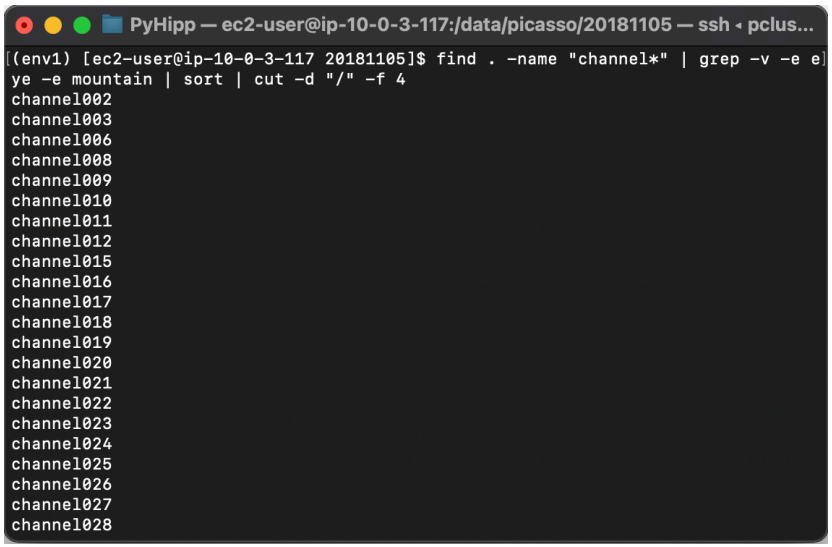
```
(env1) [ec2-user@ip-10-0-3-117 20181105]$ find . -name "channel*" | grep -v -e e
ye -e mountain | sort
./session01/array01/channel002
./session01/array01/channel003
./session01/array01/channel006
./session01/array01/channel008
./session01/array01/channel009
./session01/array01/channel010
./session01/array01/channel011
./session01/array01/channel012
./session01/array01/channel015
./session01/array01/channel016
./session01/array01/channel017
./session01/array01/channel018
./session01/array01/channel019
./session01/array01/channel020
./session01/array01/channel021
./session01/array01/channel022
./session01/array01/channel023
./session01/array01/channel024
./session01/array01/channel025
./session01/array01/channel026
./session01/array01/channel027
./session01/array01/channel028
```

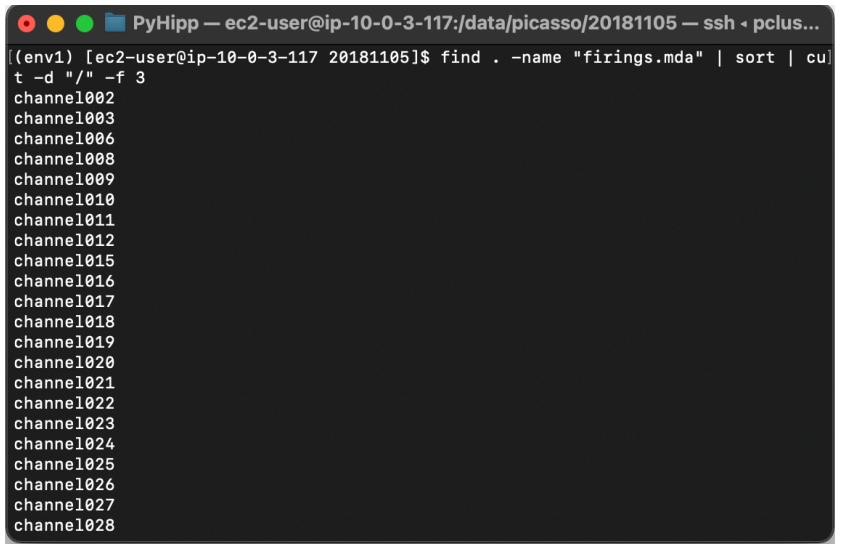
PyHipp — ec2-user@ip-10-0-3-117:/data/picasso/20181105 — ssh ◂ pclus...

```
(env1) [ec2-user@ip-10-0-3-117 20181105]$ find . -name "firings.mda" | sort
./mountains/channel002/output/firings.mda
./mountains/channel003/output/firings.mda
./mountains/channel006/output/firings.mda
./mountains/channel008/output/firings.mda
./mountains/channel009/output/firings.mda
./mountains/channel010/output/firings.mda
./mountains/channel011/output/firings.mda
./mountains/channel012/output/firings.mda
./mountains/channel015/output/firings.mda
./mountains/channel016/output/firings.mda
./mountains/channel017/output/firings.mda
./mountains/channel018/output/firings.mda
./mountains/channel019/output/firings.mda
./mountains/channel020/output/firings.mda
./mountains/channel021/output/firings.mda
./mountains/channel022/output/firings.mda
./mountains/channel023/output/firings.mda
./mountains/channel024/output/firings.mda
./mountains/channel025/output/firings.mda
./mountains/channel026/output/firings.mda
./mountains/channel027/output/firings.mda
./mountains/channel028/output/firings.mda
./mountains/channel029/output/firings.mda
```

Optimizing Parallel Processing

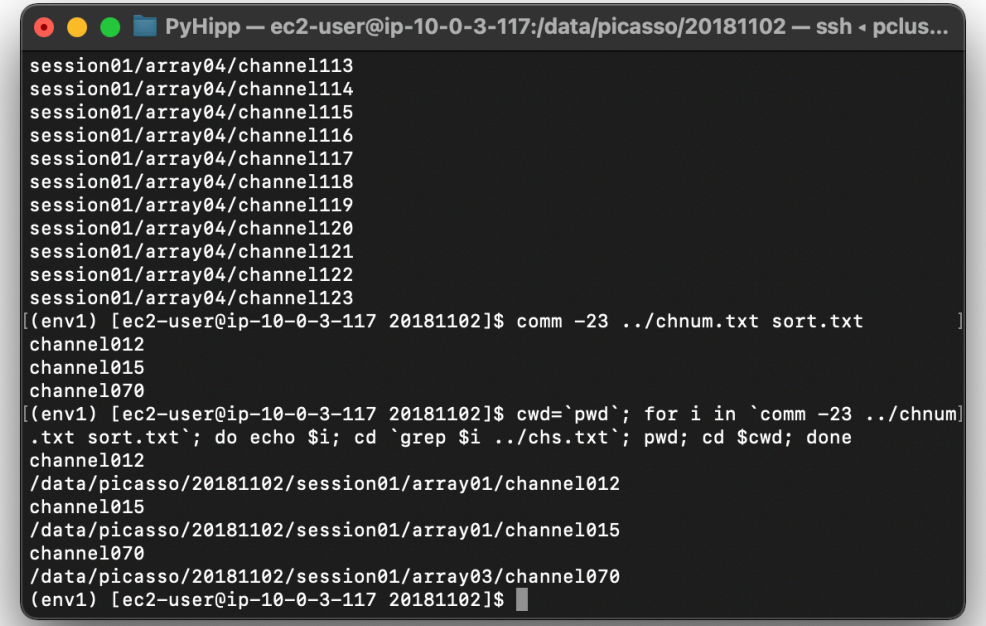
Lab 7 Techniques





Optimizing Parallel Processing

Lab 7 Techniques



Optimizing Parallel Processing

Coarse-Grained Parallel Processing (5 jobs)

Time

RPLParallel	RPLSplit		RPLSplit		RPLSplit		RPLSplit	
Unity	session01	channel001	session01	channel033	session01	channel065	session01	channel097
EDFSplit	sessioneye	channel002	sessioneye	channel034	sessioneye	channel066	sessioneye	channel098
aligning_objects		…		…		…		…
raycast		channel032		channel064		channel096		channel124
	RPLLFP		RPLLFP		RPLLFP		RPLLFP	
	session01	channel001	session01	channel033	session01	channel065	session01	channel097
	sessioneye	channel002	sessioneye	channel034	sessioneye	channel066	sessioneye	channel098
		…		…		…		…
		channel032		channel064		channel096		channel124
	RPLHighPass		RPLHighPass		RPLHighPass		RPLHighPass	
	session01	channel001	session01	channel033	session01	channel065	session01	channel097
	sessioneye	channel002	sessioneye	channel034	sessioneye	channel066	sessioneye	channel098
		…		…		…		…
		channel032		channel064		channel096		channel124
	mountain_batch		mountain_batch		mountain_batch		mountain_batch	
	session01	channel001	session01	channel033	session01	channel065	session01	channel097
		channel002		channel034		channel066		channel098
		…		…		…		…
		channel032		channel064		channel096		channel124

Optimizing Parallel Processing

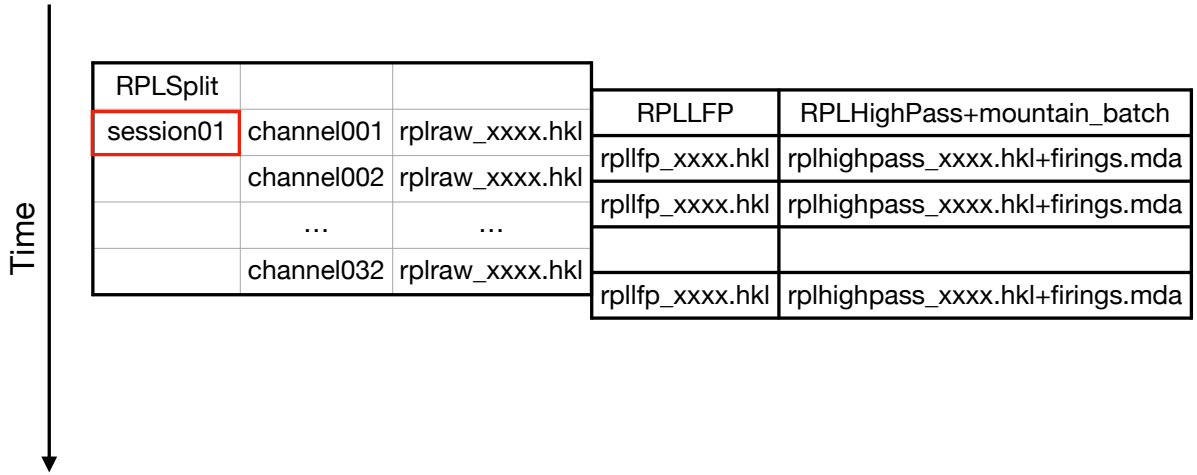
Fine-Grained Parallel Processing

rs1-slurm.sh

```
# LOAD MODULES, INSERT CODE, AND RUN YOUR PROGRAMS HERE
python -u -c "import PyHipp as pyh; \
import DataProcessingTools as DPT; \
import os; \
import time; \
t0 = time.time(); \
print(time.localtime()); \
DPT.objects.processDirs(dirs=None, objtype=pyh.RPLSplit, channel=[*range(1,33)]); \
DPT.objects.processDirs(dirs=['sessioneye/array01','session01/array01'], cmd='import
PyHipp as pyh; import DataProcessingTools as DPT; DPT.objects.processDirs(None,
pyh.RPLLFP, saveLevel=1); DPT.objects.processDirs(None, pyh.RPLHighPass, saveLevel=1);');
\
os.chdir('session01/array01'); \
DPT.objects.processDirs(level='channel', cmd='import PyHipp as pyh; from PyHipp import
mountain_batch; mountain_batch.mountain_batch(); from PyHipp import export_mountain_cells;
export_mountain_cells.export_mountain_cells();'); \
print(time.localtime()); \
print(time.time()-t0);"
```

Optimizing Parallel Processing

Fine-Grained Parallel Processing



Data Processing on AWS

Ripple Low-Pass & High-Pass Data

Data Type	Input	Function	Output
Low-Pass Neural Data	rplraw_xxxx.hkl	RPLLFP	rpllfp_xxxx.hkl
High-Pass Neural Data	rplraw_xxxx.hkl	RPLHighPass	rplhighpass_xxxx.hkl

RPLLFP (from rpllfp.py)
Called from the Channel Directory

array01		
	channel001	
		rplraw_xxxx.hkl
		rpllfp_xxxx.hkl
	…	
array02		
	channel033	
		rplraw_xxxx.hkl
		rpllfp_xxxx.hkl
	…	
…		

RPLHighPass (from rplhighpass.py)
Called from the Channel Directory

array01		
	channel001	
		rplraw_xxxx.hkl
		rpllfp_xxxx.hkl
		rplhighpass_xxxx.hkl
	…	
array02		
	channel033	
		rplraw_xxxx.hkl
		rpllfp_xxxx.hkl
		rplhighpass_xxxx.hkl
…		

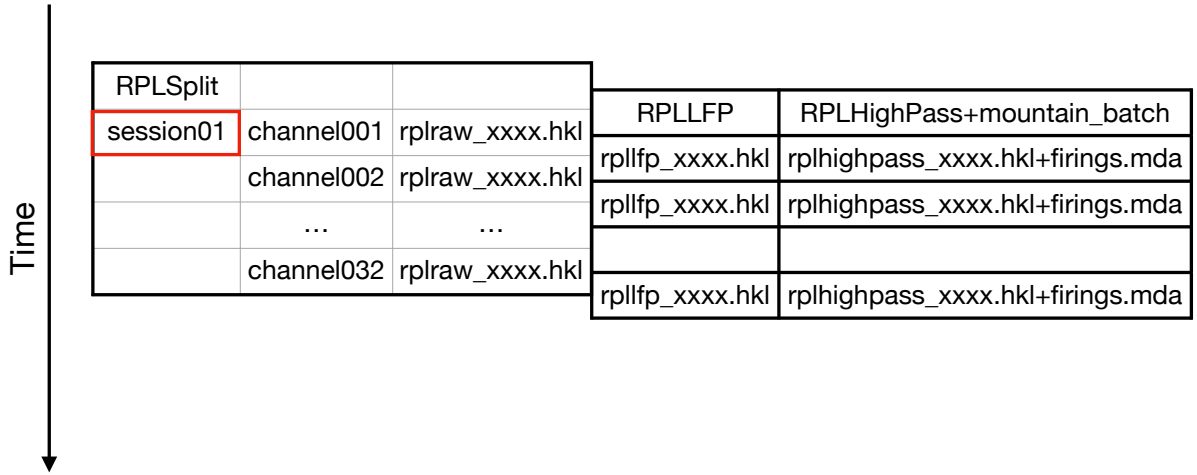
Data Processing on AWS

Spike Sorted Data

session01				
	array01			
		channel001		
			…	
			rplhighpass_xxxx.hkl	
			cell01	
				spiketrain_xxxx.hkl
				spiketrain.csv
sessioneye			mountain_batch (from mountain_batch.py) Called from the Channel Directory in session01 directory	
	array01			
		channel001		
			…	
			rplhighpass_xxxx.hkl	
			cell01	
				spiketrain_xxxx.hkl
				spiketrain.csv

Optimizing Parallel Processing

Fine-Grained Parallel Processing



Optimizing Parallel Processing

Fine-Grained Parallel Processing (226 jobs)

Just-in-time job submission

DPT.objects.processDirs(dirs=None, objtype=pyh.RPLSplit, channel=[*range(1,33)], SkipHPC=False, HPCScriptsDir = '/data/src/PyHipp/', SkipLFP=False, SkipHighPass=False, SkipSort=False);

rplsplit.py — Free Mode

~/Dropbox/Work/Python/PyHipp/PyHipp/rplsplit.py — main — (no function selected)

```
82              os.chdir(channelDir)
83              print('Calling RPLRaw for channel {:03d}'.format(channelNumber))
84              rplraw.RPLRaw(analogData = data, analogInfo = analogInfo, saveLevel = 1)
85              if self.args['SkipHPC']:
86                  if not self.args['SkipLFP']:
87                      print('Calling RPLLFP for channel {:03d}'.format(channelNumber))
88                      rpllfp.RPLLFP(saveLevel = 1)
89                  if not self.args['SkipHighPass']:
90                      print('Calling RPLHighPass for channel {:03d}'.format(channelNumber))
91                      rplhighpass.RPLHighPass(saveLevel = 1)
92                  if DPT.levels.get_level_name('session', os.getcwd()) != 'sessioneye':
93                      if not self.args['SkipSort']:
94                          print('Calling Mountain Sort for channel {:03d}'.format(channelNumber))
95                          # mountain_batch()
96                          # export_mountain_cells()
97              else:
98                  if 'HPCScriptsDir' not in kwargs.keys():
99                      kwargs['HPCScriptsDir'] = ''
100                 if not self.args['SkipLFP']:
101                     print('Adding RPLLFP slurm script for channel {:03d} to job queue'.format(channelNumber))
102                     os.system('sbatch ' + kwargs['HPCScriptsDir'] + 'rpllfp-slurm.sh')
103                 if not self.args['SkipHighPass']:
104                     if not self.args['SkipSort']:
105                         print('Adding RPLHighPass and Mountain Sort slurm script for channel {:03d} to job queue'.format(channelNumber))
106                         os.system('sbatch '+ kwargs['HPCScriptsDir'] + 'rplhighpass-sort-slurm.sh')
107                     else:
108                         print('Adding RPLHighPass slurm script for channel {:03d} to job queue'.format(channelNumber))
109                         os.system('sbatch ' + kwargs['HPCScriptsDir'] + 'rplhighpass-slurm.sh')
```

L: 153 C: 1 — Python — Unicode (UTF-8) — Unix (LF) — Saved: 11/16/20, 5:13:37 PM — 6,438 / 687 / 153 — 100%

Optimizing Parallel Processing

Fine-Grained Parallel Processing (226 jobs)

6. In addition, create the scripts rpllfp-slurm.sh and rplhighpass-sort-slurm.sh so they can be used by RPLSplit to submit jobs from the channel directory once the rplraw files are created:

```
python -u -c "import PyHipp as pyh; \
import time; \
pyh.RPLLFP(saveLevel=1); \
print(time.localtime());"

python -u -c "import PyHipp as pyh; \
import time; \
pyh.RPLHighPass(saveLevel=1); \
from PyHipp import mountain_batch; \
mountain_batch.mountain_batch(); \
from PyHipp import export_mountain_cells; \
export_mountain_cells.export_mountain_cells(); \
print(time.localtime());"
```

Optimizing Parallel Processing

Lab 7 Techniques

- Slurm
  - Just-in-time job submission to allow jobs to start when data is ready
  - Creating many jobs allows full use of all available vCPUs

	Serial Job	2 Parallel Jobs	5 Parallel Jobs	226 Parallel Jobs
1 Day Neural Data	10 hours	9.5 hours	3 hours	2.5 hours

- AWS
  - Using combination of instance types for compute nodes to maximize vCPUs
  - Possible bottlenecks caused by lack of resources on the head node

Questions?

EE3801 Data Engineering Principles

High-Throughput Visualization

High-Throughput Visualization
OOP Code Reuse

- DataProcessingTools (DPT)
  - DPT.objects.processDirs(dirs=None, objtype=pyh.RPLParallel, saveLevel=1)
  - DPT.objects.processDirs(dirs=None, objtype=pyh.RPLSplit, saveLevel=1)
  - DPT.objects.processDirs(dirs=None, objtype=pyh.RPLLFP, saveLevel=1)
  - DPT.objects.processDirs(dirs=None, objtype=pyh.RPLHighPass, saveLevel=1)
  - DPT.objects.processDirs(dirs=None, objtype=pyh.Unity, saveLevel=1)

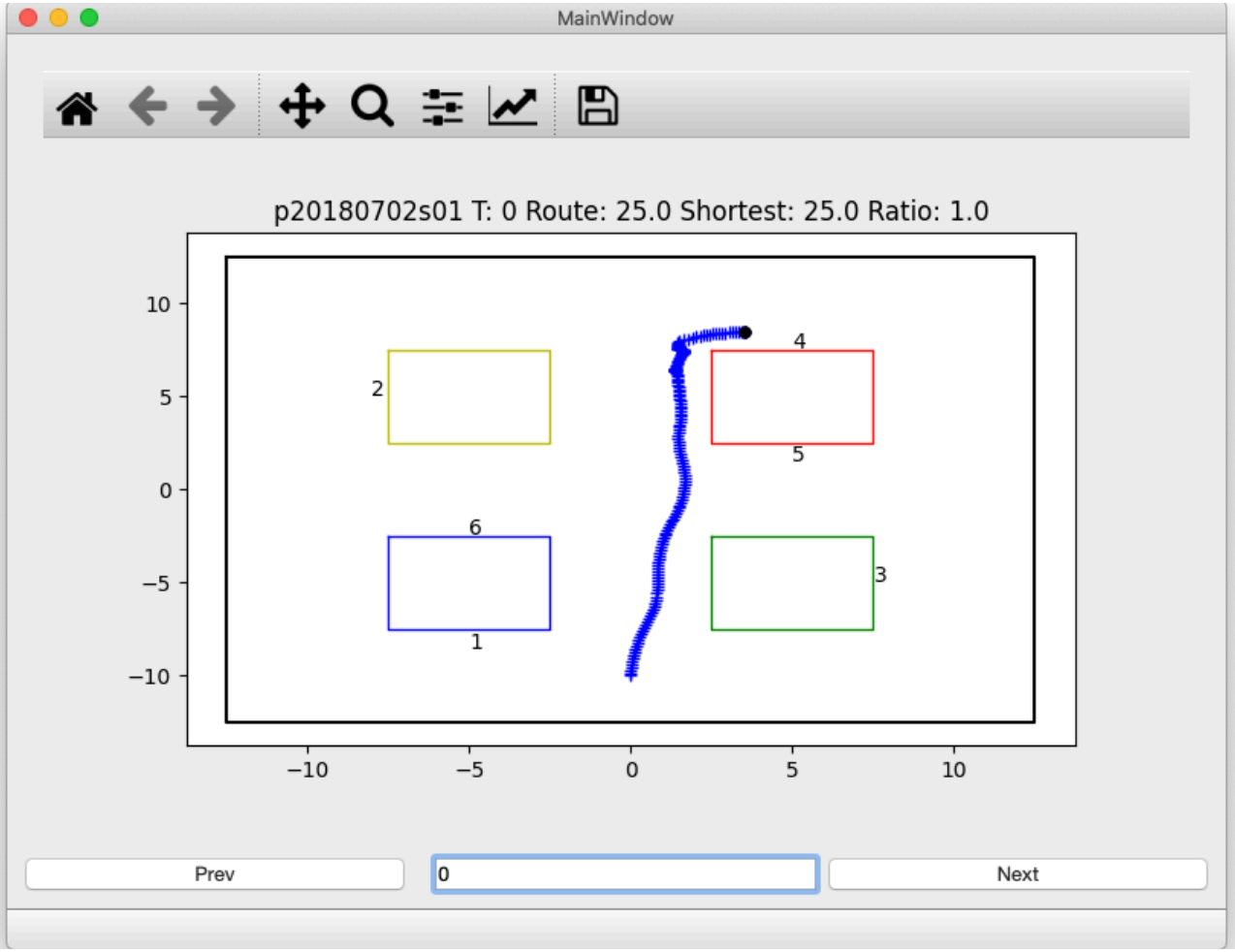
High-Throughput Visualization

OOP Code Reuse

- PanGUI
  - PanGUI.create_window(pyh.Unity())
  - PanGUI.create_window(pyh.Eyelink())
  - PanGUI.create_window(pyh.VMRaw())
  - PanGUI.create_window(pyh.VMLFP())
  - PanGUI.create_window(pyh.VMHighPass())

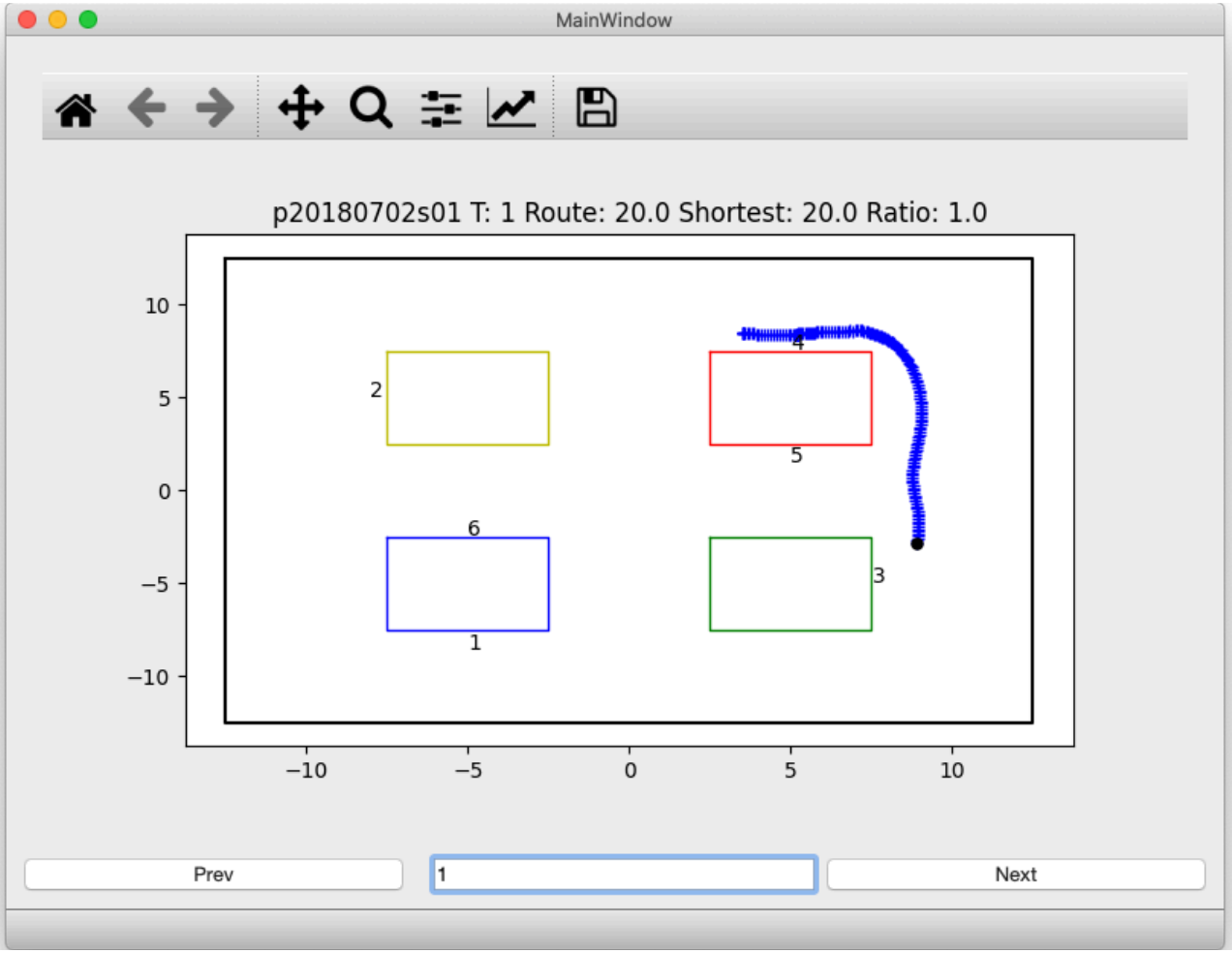
High-Throughput Visualization

Interactive Plots



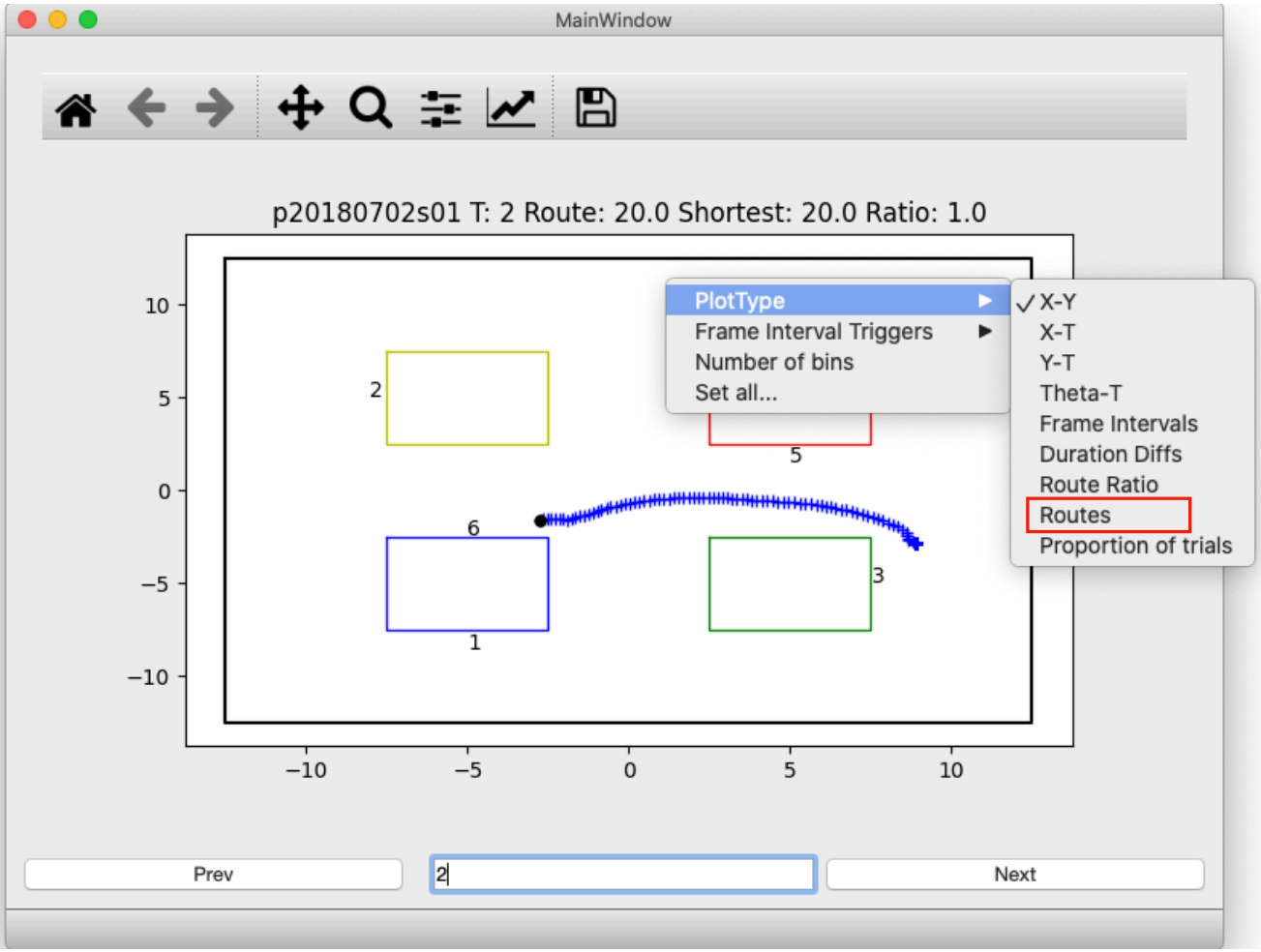
High-Throughput Visualization

Interactive Plots



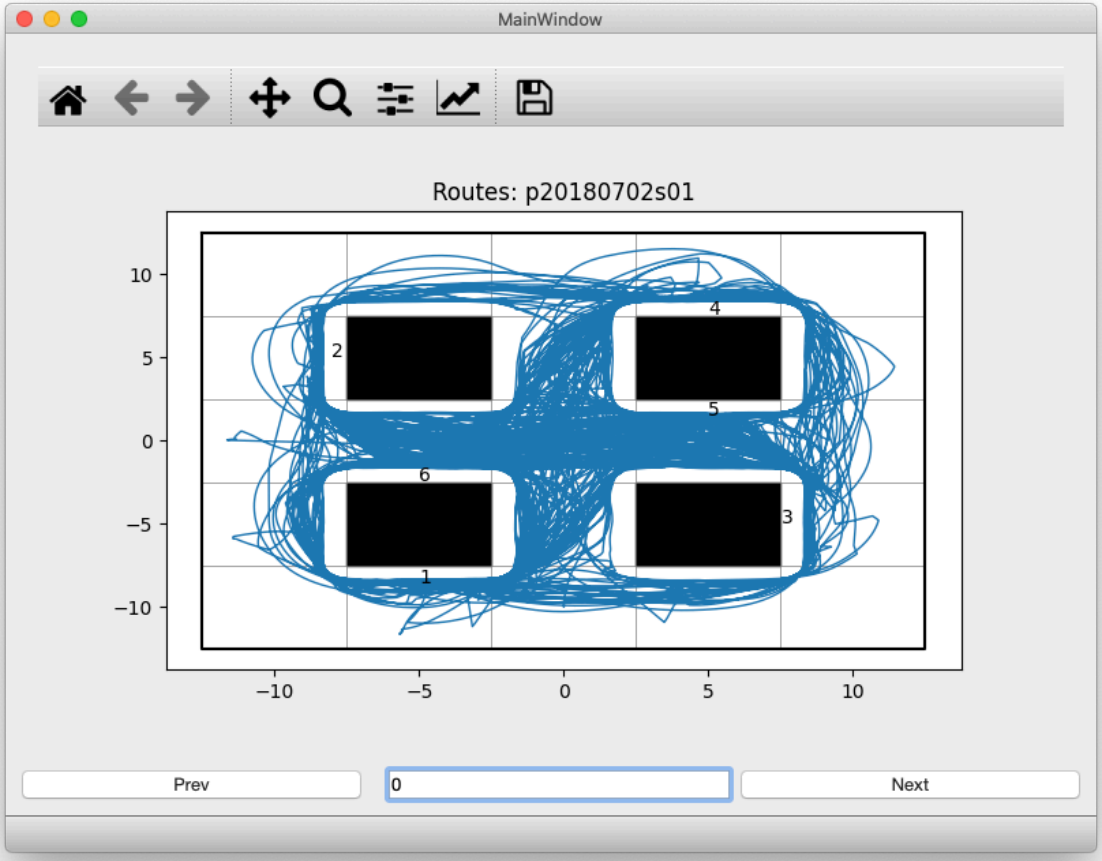
High-Throughput Visualization

Interactive Plots



High-Throughput Visualization

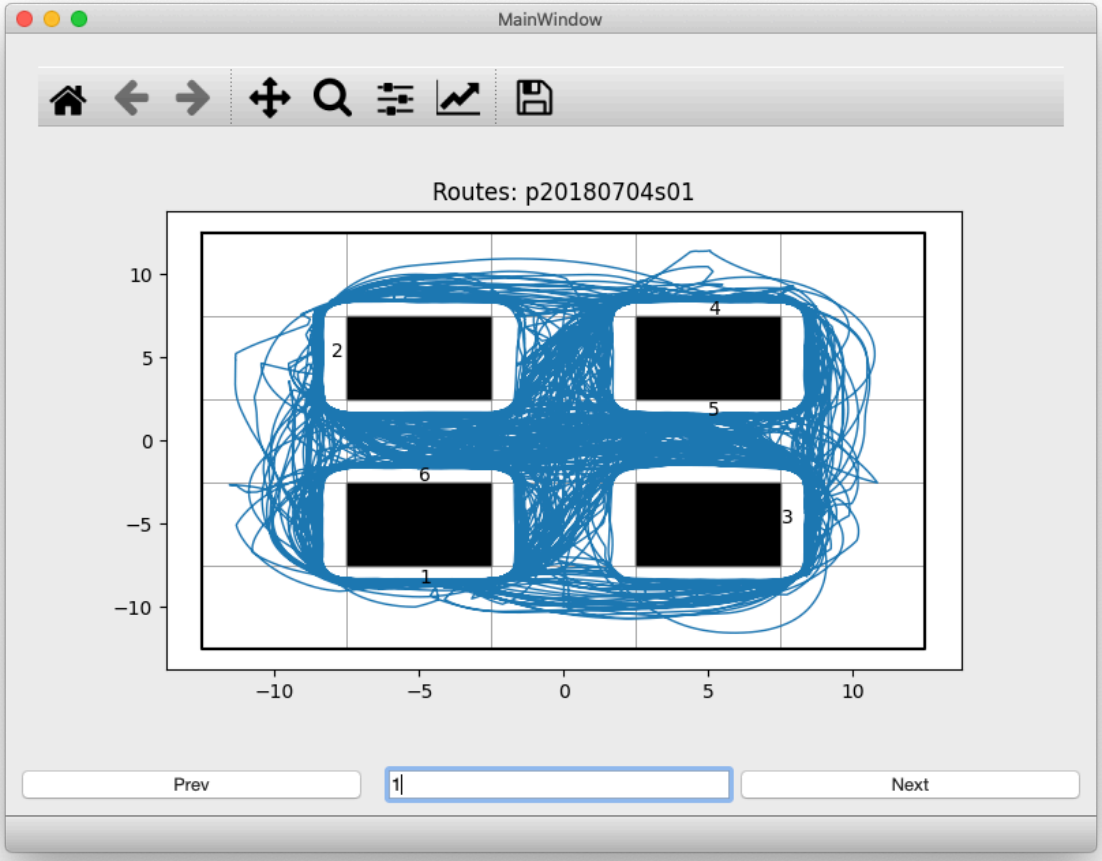
Interactive Plots



403 Trials

High-Throughput Visualization

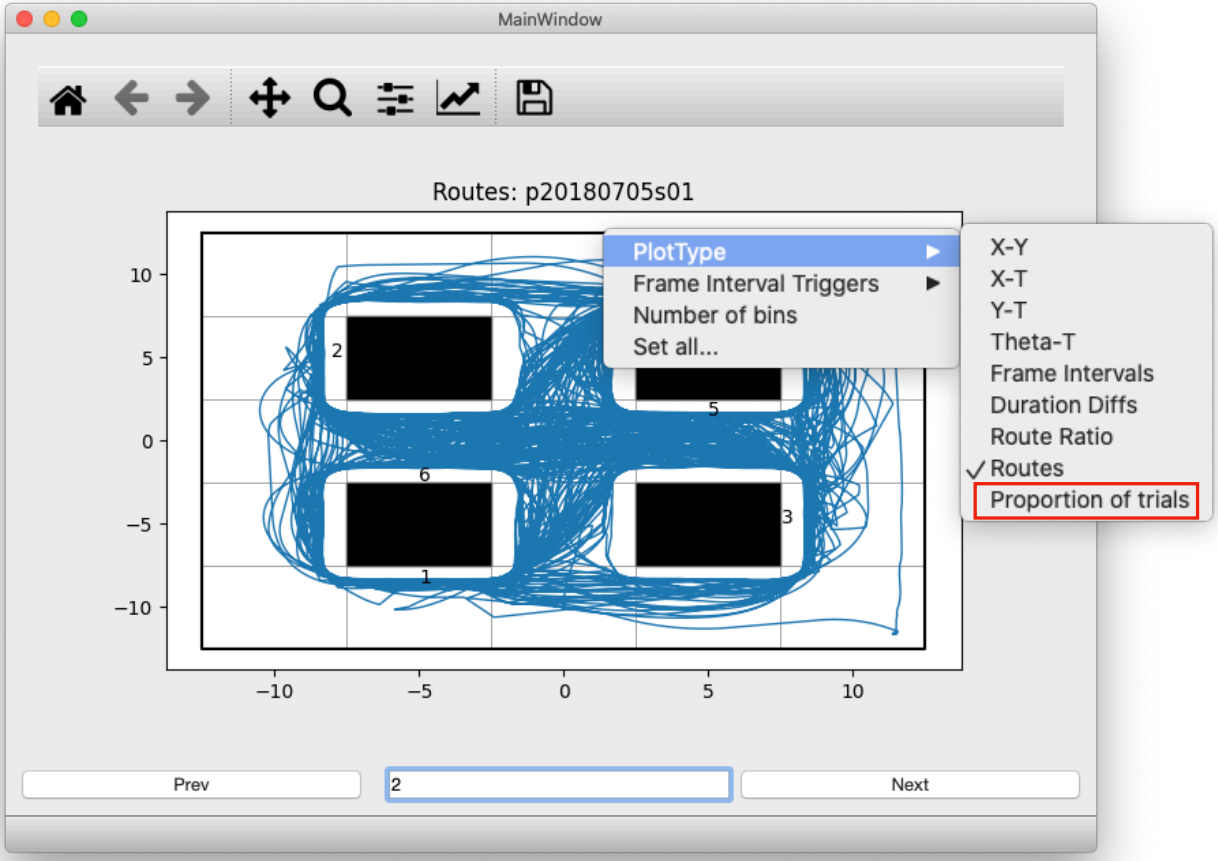
Interactive Plots



403 Trials

High-Throughput Visualization

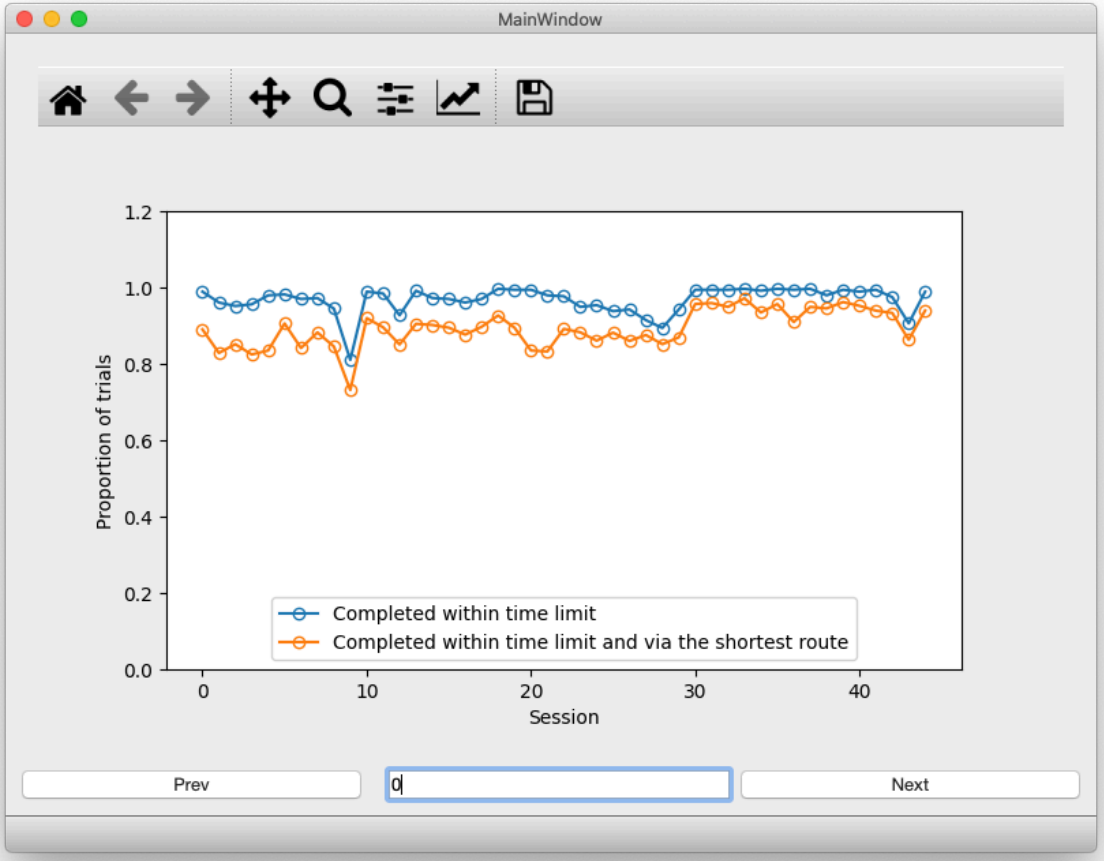
Interactive Plots



403 Trials

High-Throughput Visualization

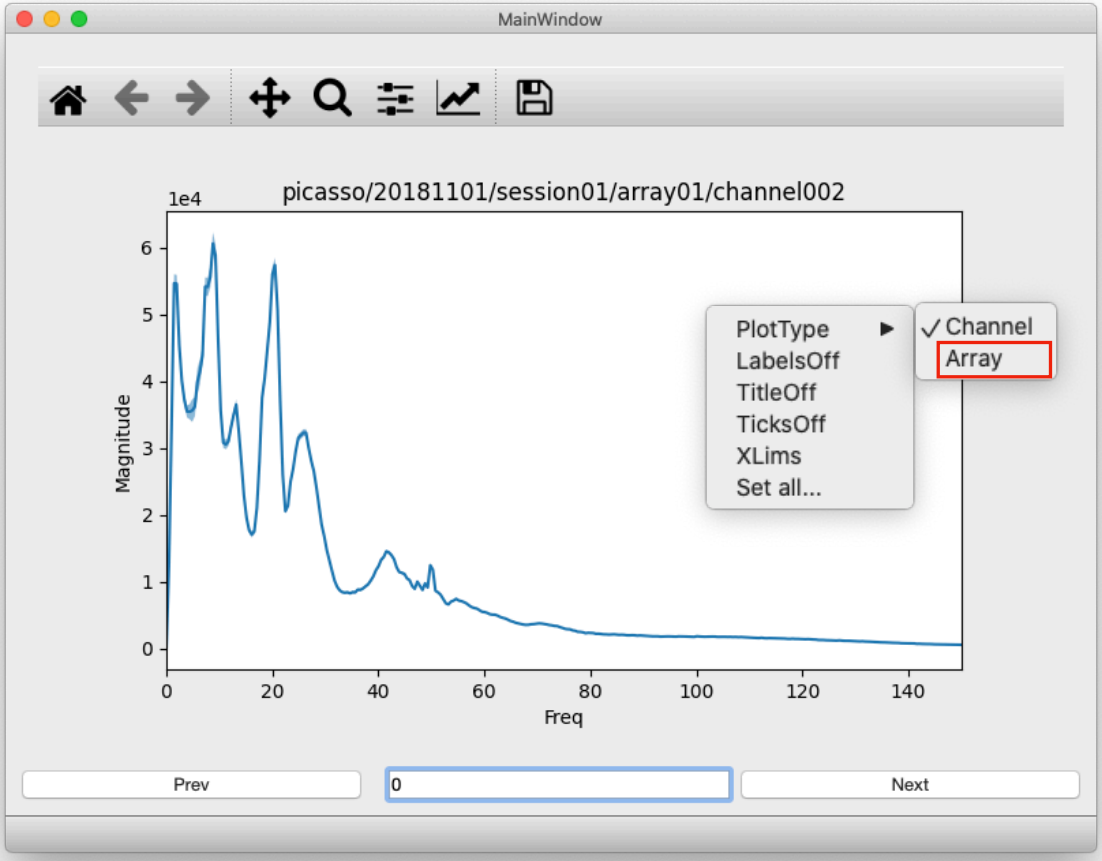
Interactive Plots



18,305 Trials

High-Throughput Visualization

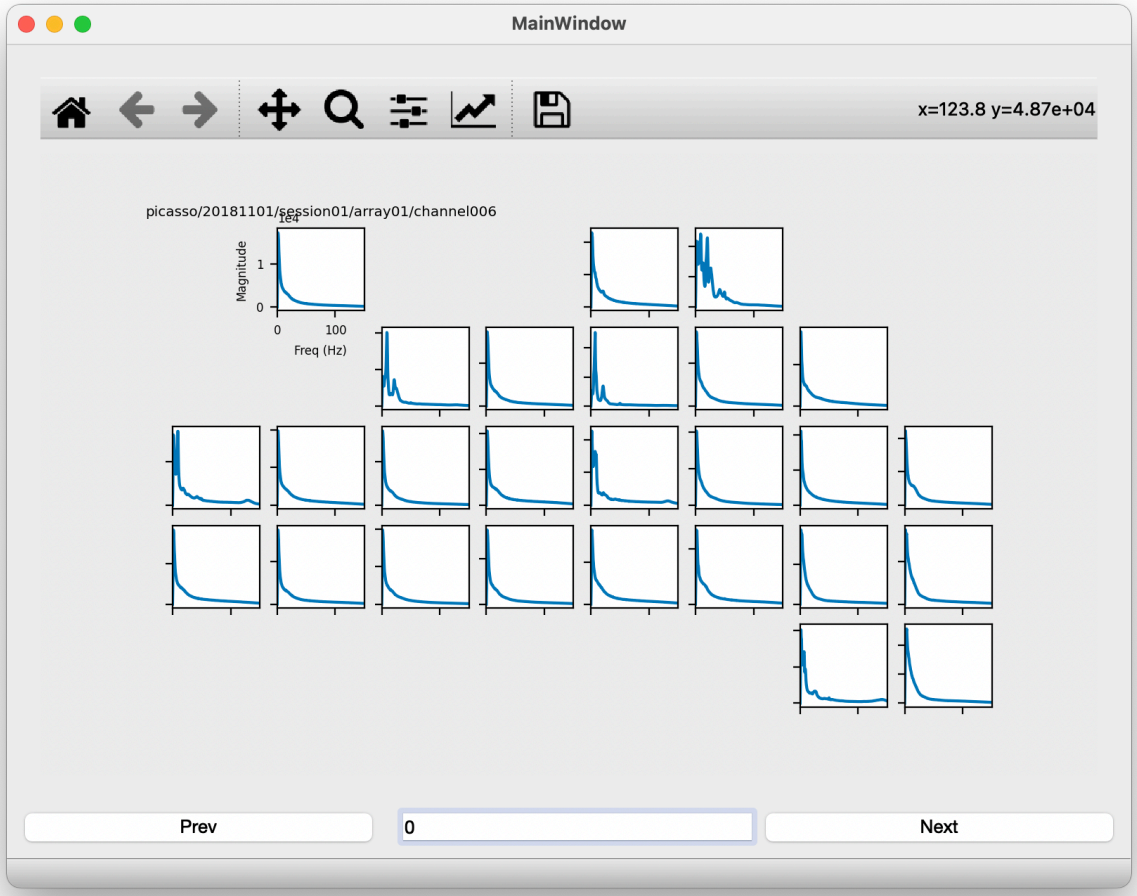
Interactive Plots



1 channel

High-Throughput Visualization

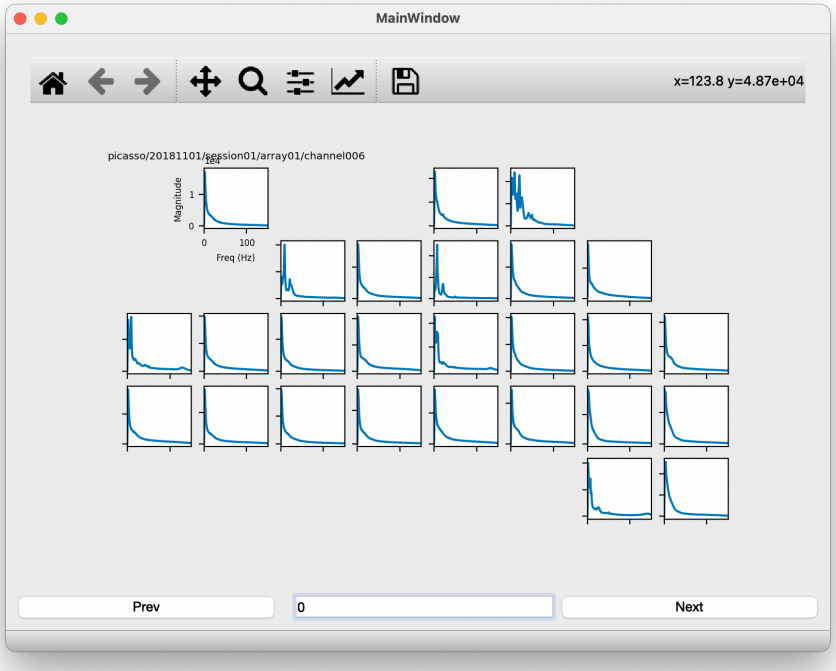
Interactive Plots



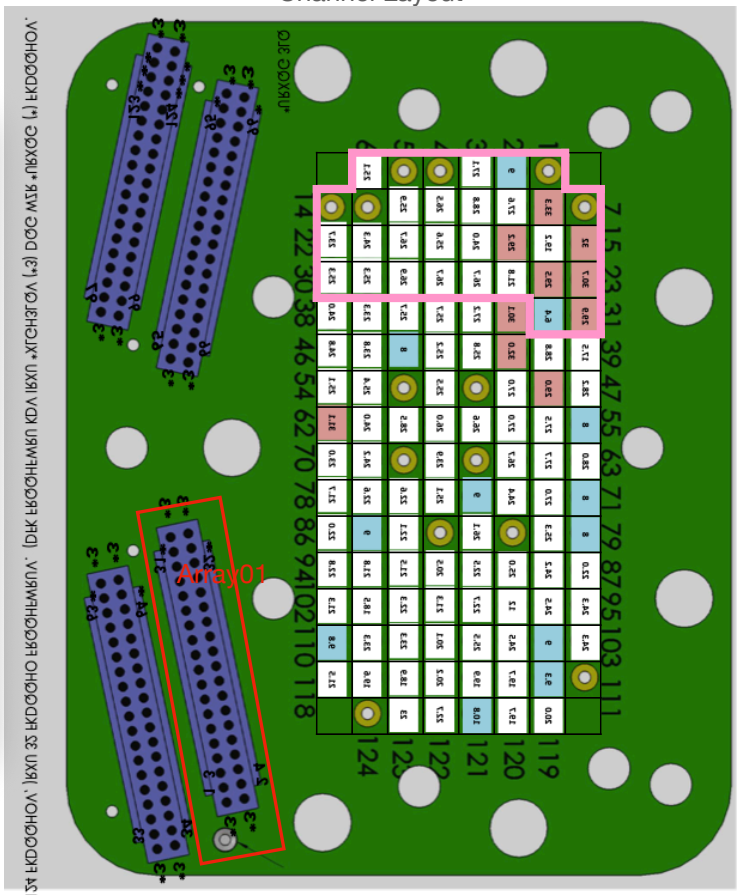
32 channels

High-Throughput Visualization

Interactive Plots

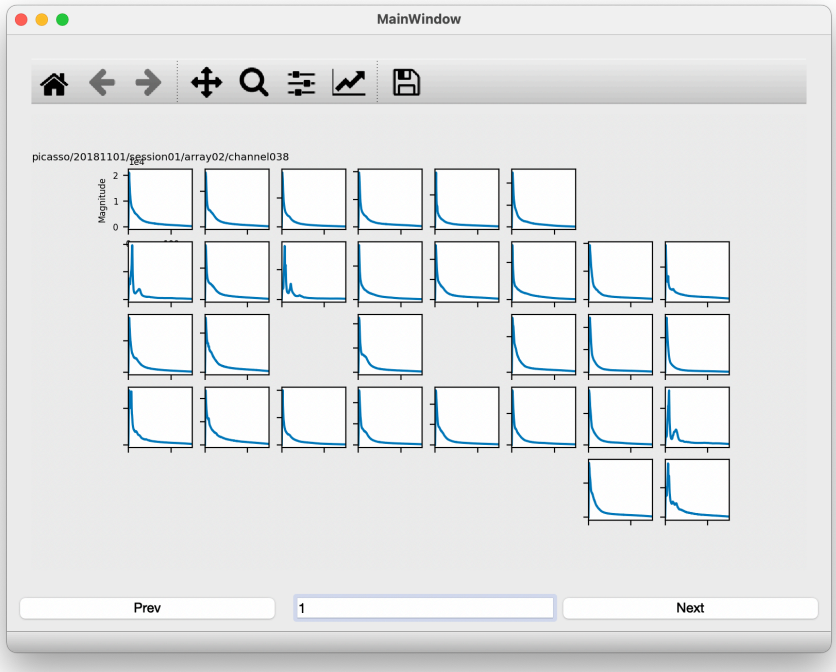


Channel Layout

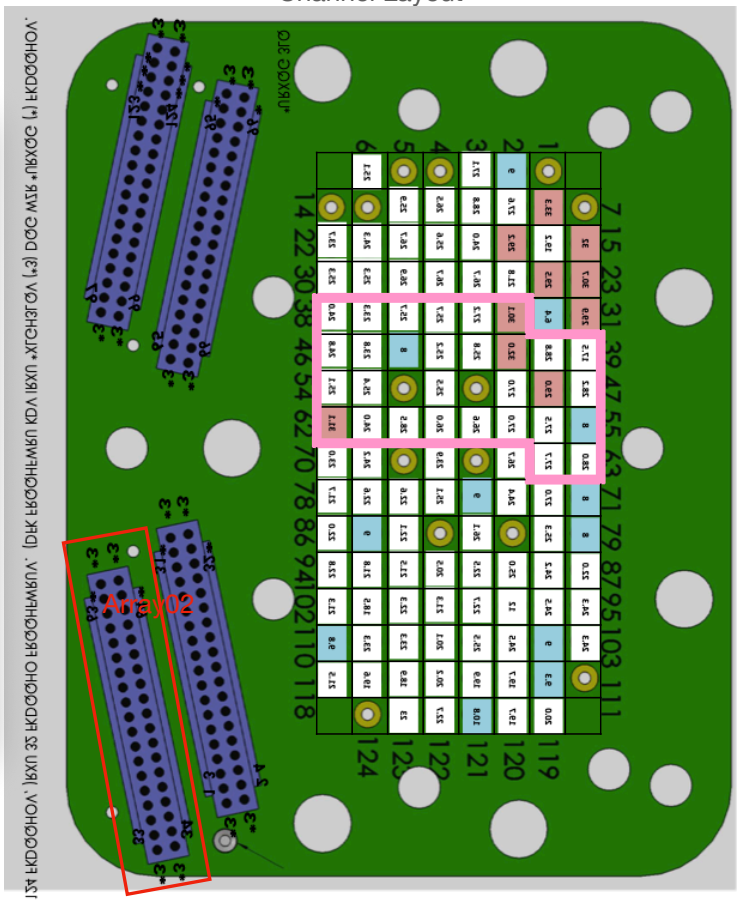


High-Throughput Visualization

Interactive Plots

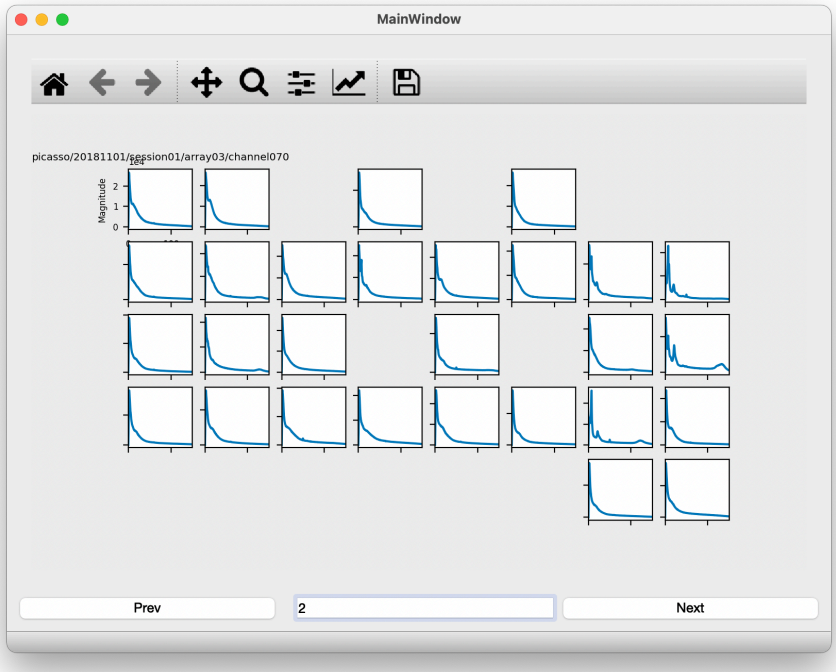


Channel Layout

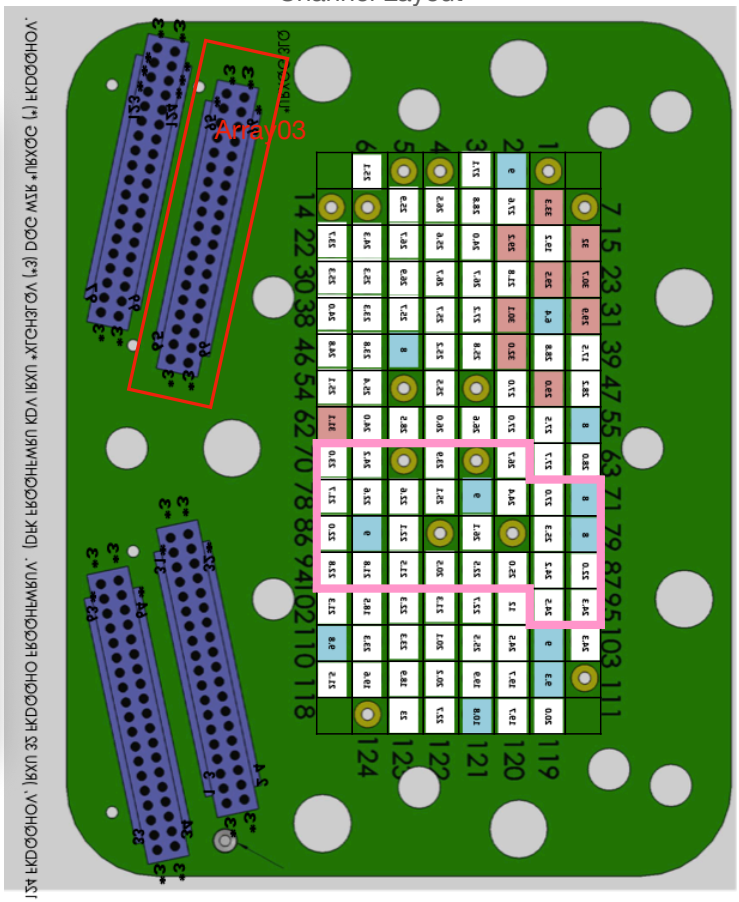


High-Throughput Visualization

Interactive Plots

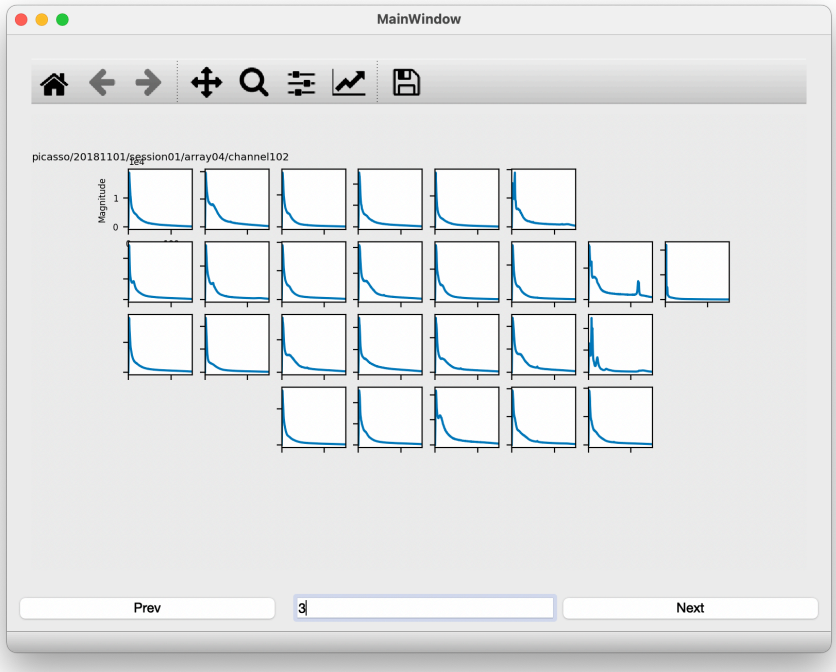


Channel Layout

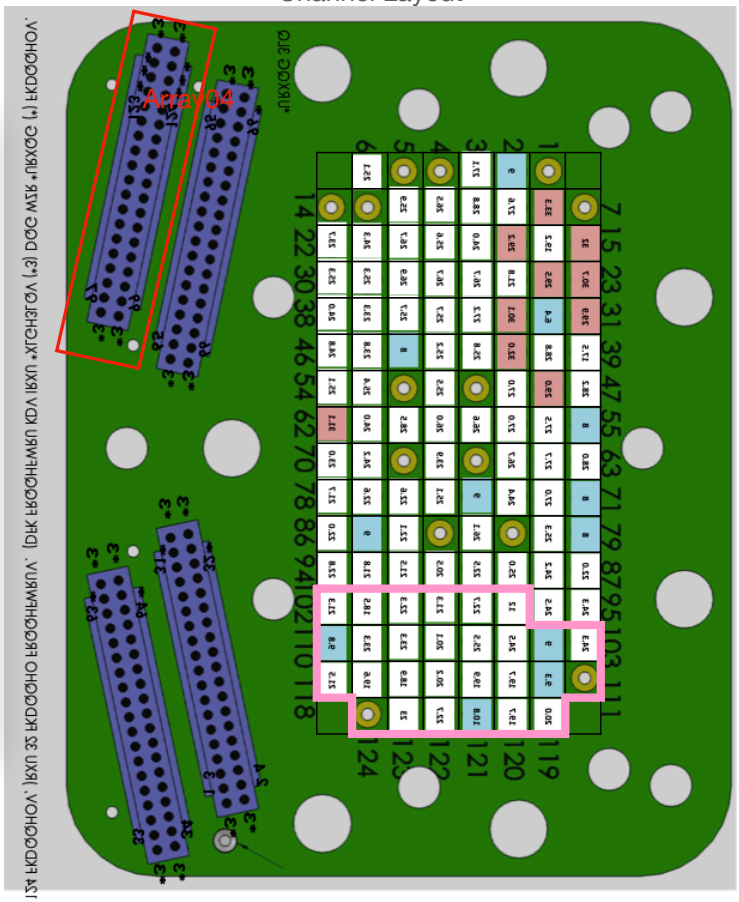


High-Throughput Visualization

Interactive Plots

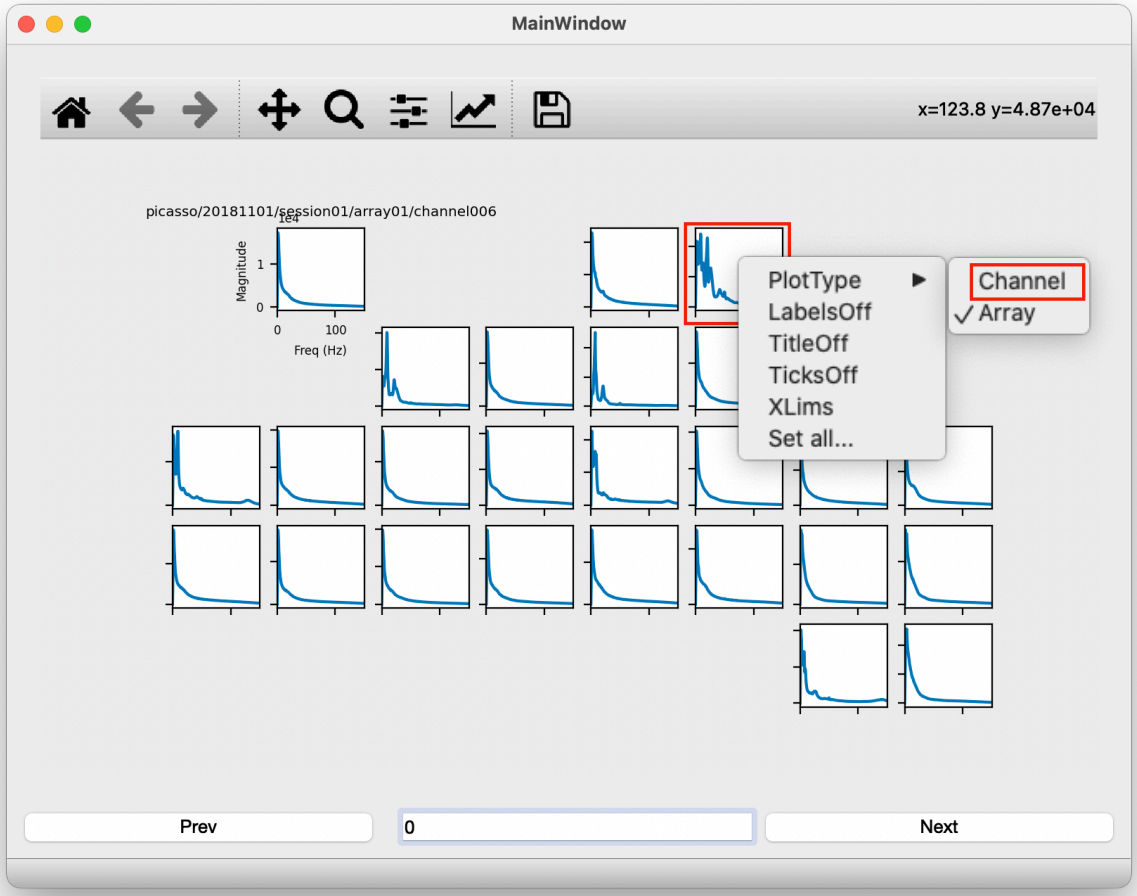


Channel Layout



High-Throughput Visualization

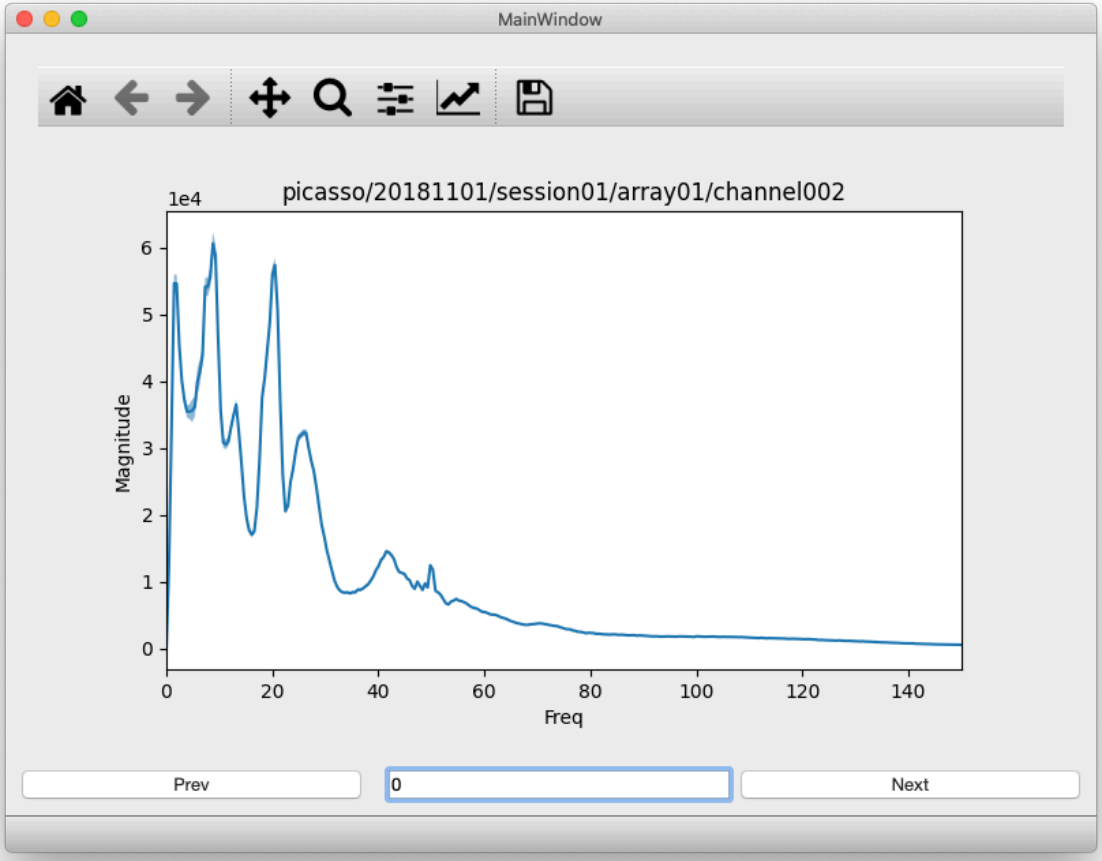
Interactive Plots



32 channels

High-Throughput Visualization

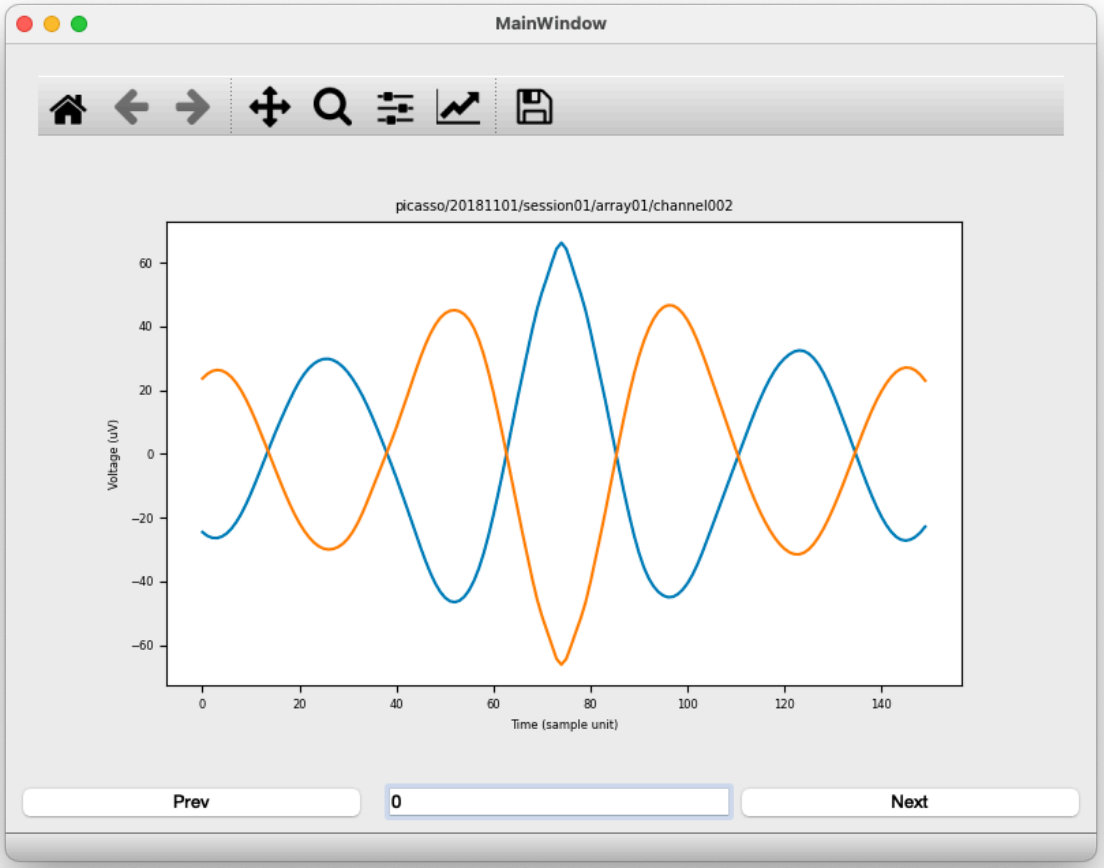
Interactive Plots



1 channel

High-Throughput Visualization

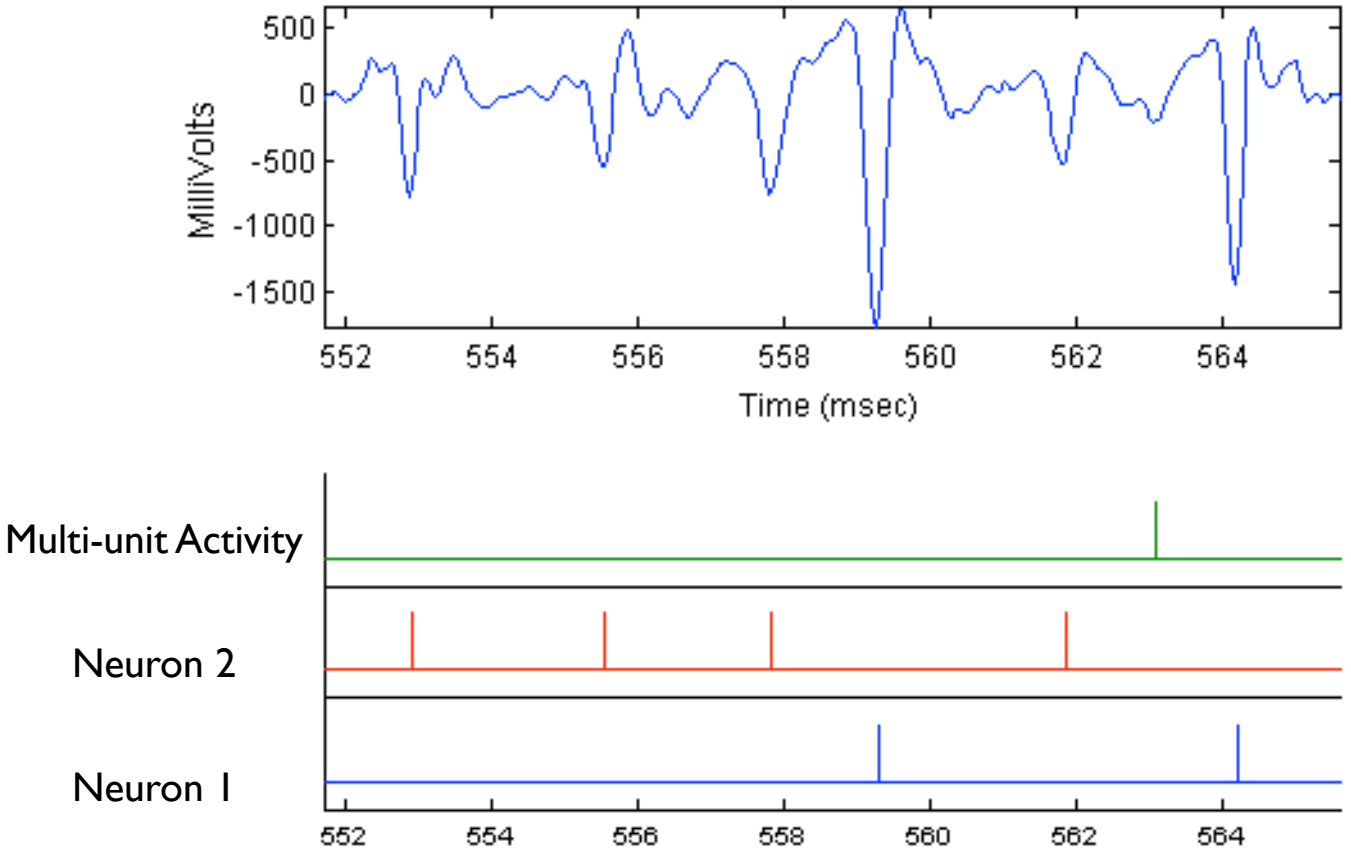
Interactive Plots



waveform.py

Data Acquisition & Processing

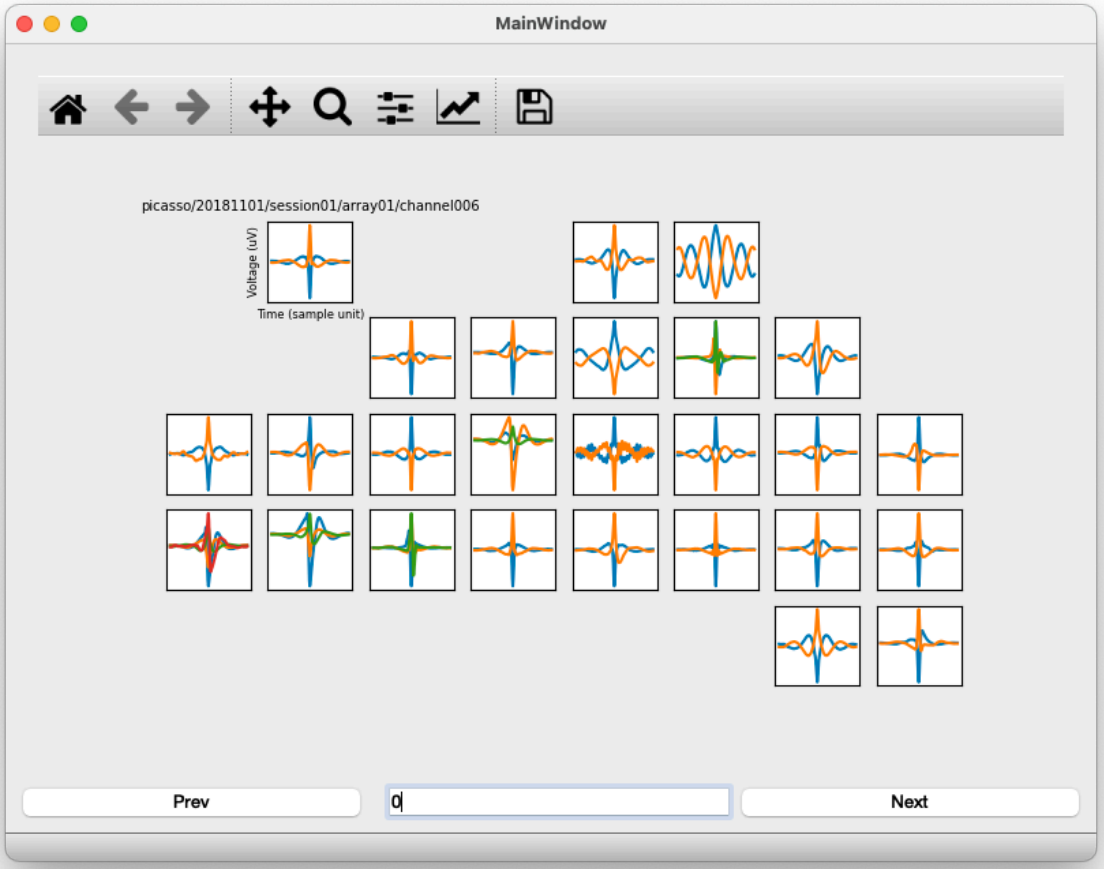
Neural Data (Ripple)



After spike sorting

High-Throughput Visualization

Interactive Plots



waveform.py

NUS HPC

NUS National University of Singapore | Information Technology

HPC@NUS IT

myEmail Staff Portal Student Portal nTouch

Home Milestones Getting Started Services HPC Newsletter

About HPC

In a research environment, High Performance Computing (HPC) is the use of hardware, software, tools and programming techniques to accelerate research computation, which in turn will enable the execution of large cutting-edge research simulation that accelerates new discoveries. At the National University of Singapore, a combination of centralized and distributed approaches has been adopted in the provisioning and support of High Performance Computing (HPC) for research computation.

In this hybrid environment, some departments and research groups are providing local resources to support their users, whereas the HPC team at the NUS IT is focusing on providing central HPC resources and services to support all staff and student researchers across campus. NUS IT is also collaborating with the National Supercomputing Centre (NSCC) in providing high-end HPC resources to NUS researchers to enable large-scale and complex research simulations.

About NUS IT HPC

Central HPC support at NUS IT began in 1989 when a shared vector supercomputer was made available to the NUS research community. Since then, a variety of HPC systems and technologies had been introduced to meet wide range of research computing requirements. In our recent development, we have been focusing on developing a data-centric HPC environment since the beginning of 2010s. From mid-2010s, additional focus on AI related resources and technologies has been initiated to keep pace with emerging Machine Learning/Deep Learning research requirements. We are expecting the HPC-AI era in research computing to be in full swing in 2020s.

NUS HPC

AWS vs NUS HPC

	AWS	NUS HPC
Job Manager	Slurm	PBS
Submit	sbatch	qsub
Queue	squeue	qstat
Delete	scancel	qdel
Cost	$	Free
Head Node	Created on demand	atlas[6-9].nus.edu.sg

Optimizing Parallel Processing

Lab Instructions

- Lab 8 Instructions:
  - https://ee3801.github.io/Lab8/instruction.html
- Submit to Canvas (Lab 8->Lab 8A & Lab 8->Lab 8B)
- Submit in PDF format
- Name the files Lab8A_YourName.pdf and Lab8B_YourName.pdf
- Part A due on Monday (Nov 6) 2 pm
- Part B due on Wednesday (Nov 8) 9 pm

Questions?