Analysis of physics analysis

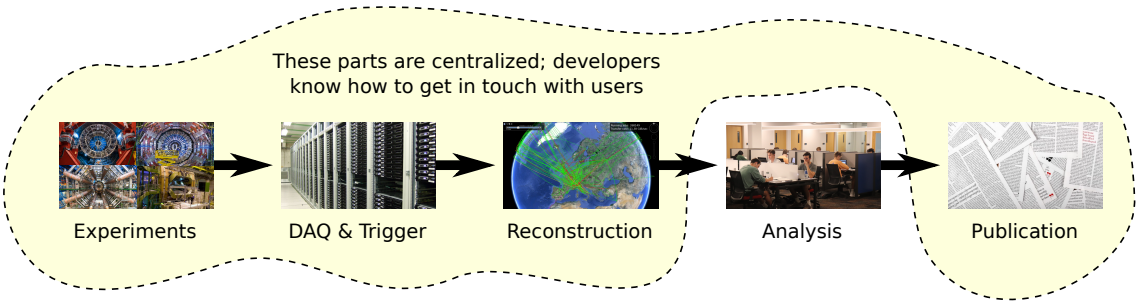
Jim Pivarski and Henry Schreiner

Princeton University – IRIS-HEP

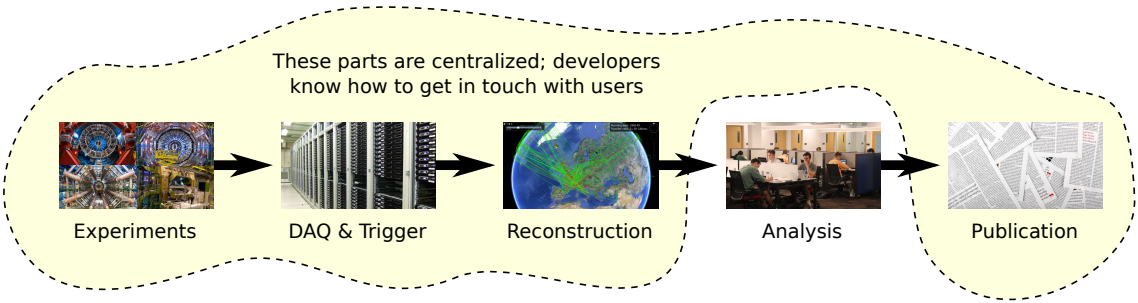
May 9, 2023

Dark computing

Dark computing



Dark computing



- The "analysis" step is the only one in the pipeline for which we don't even know *who* all the users are.

We don't do this

So what can we do instead?

Method	Good	Bad

So what can we do instead?

Method	Good	Bad
Bug-reports	Resolve immediate needs.	Only hear from proactive people.

So what can we do instead?

Method	Good	Bad
Bug-reports	Resolve immediate needs.	Only hear from proactive people.
Surveys	Can directly ask people what they think. Quantitative.	Are the people who didn't fill it out correlated with the questions?

So what can we do instead?

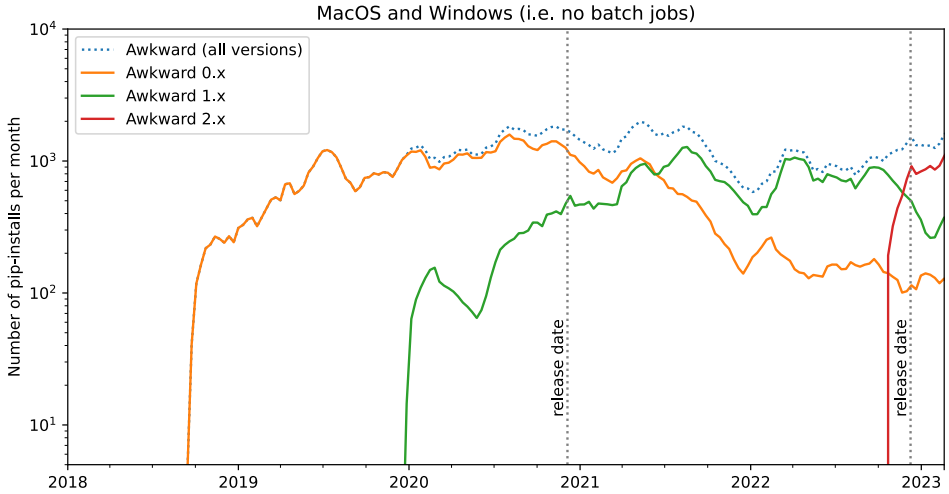
Method	Good	Bad
Bug-reports	Resolve immediate needs.	Only hear from proactive people.
Surveys	Can directly ask people what they think. Quantitative.	Are the people who didn't fill it out correlated with the questions?
Focus groups	As above, but open to free-form, generating new ideas.	Need to follow up from the small group to a large survey.

So what can we do instead?

Method	Good	Bad
Bug-reports	Resolve immediate needs.	Only hear from proactive people.
Surveys	Can directly ask people what they think. Quantitative.	Are the people who didn't fill it out correlated with the questions?
Focus groups	As above, but open to free-form, generating new ideas.	Need to follow up from the small group to a large survey.
Download stats	People vote with their feet. Quantitative.	Coarse-grained: only know package-level info. Skewed by batch jobs.

What download stats are good for (one slide)

Relative rates, such as new version adoption.



So what can we do instead?

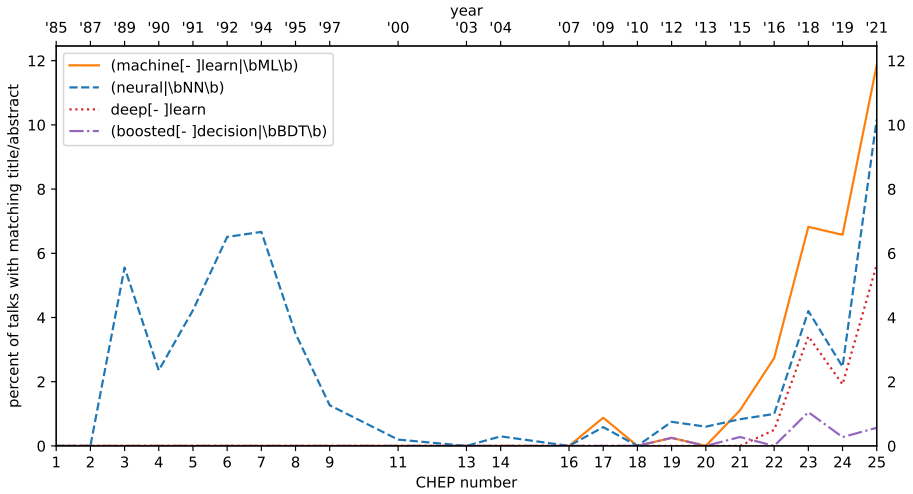
Method	Good	Bad
Bug-reports	Resolve immediate needs.	Only hear from proactive people.
Surveys	Can directly ask people what they think. Quantitative.	Are the people who didn't fill it out correlated with the questions?
Focus groups	As above, but open to free-form, generating new ideas.	Need to follow up from the small group to a large survey.
Download stats	People vote with their feet. Quantitative.	Coarse-grained: only know package-level info. Skewed by batch jobs.

So what can we do instead?

Method	Good	Bad
Bug-reports	Resolve immediate needs.	Only hear from proactive people.
Surveys	Can directly ask people what they think. Quantitative.	Are the people who didn't fill it out correlated with the questions?
Focus groups	As above, but open to free-form, generating new ideas.	Need to follow up from the small group to a large survey.
Download stats	People vote with their feet. Quantitative.	Coarse-grained: only know package-level info. Skewed by batch jobs.
Textual analysis of CHEP/ACAT	Long-view historical trends.	Only for those who give talks, and what they choose to talk about.

What textual analysis of CHEP/ACAT is good for (one slide)

Discovering trends and changing interests.



So what can we do instead?

Method	Good	Bad
Bug-reports	Resolve immediate needs.	Only hear from proactive people.
Surveys	Can directly ask people what they think. Quantitative.	Are the people who didn't fill it out correlated with the questions?
Focus groups	As above, but open to free-form, generating new ideas.	Need to follow up from the small group to a large survey.
Download stats	People vote with their feet. Quantitative.	Coarse-grained: only know package-level info. Skewed by batch jobs.
Textual analysis of CHEP/ACAT	Long-view historical trends.	Only for those who give talks, and what they choose to talk about.

So what can we do instead?

Method	Good	Bad
Bug-reports	Resolve immediate needs.	Only hear from proactive people.
Surveys	Can directly ask people what they think. Quantitative.	Are the people who didn't fill it out correlated with the questions?
Focus groups	As above, but open to free-form, generating new ideas.	Need to follow up from the small group to a large survey.
Download stats	People vote with their feet. Quantitative.	Coarse-grained: only know package-level info. Skewed by batch jobs.
Textual analysis of CHEP/ACAT	Long-view historical trends.	Only for those who give talks, and what they choose to talk about.
Analysis of source code online	Fine-grained, quantitative, average over many users.	Only public repos, have to identify demographics with some seed: how to define "particle physicists"?

Analysis of source code online (the rest of this talk)

A few years ago (2019), Jim stumbled upon a good technique:

- CMSSW has been on GitHub since 2013.
- Many CMS physicists have to fork CMSSW at some point.
- Very few non-physicists would fork CMSSW.

Analysis of source code online (the rest of this talk)

A few years ago (2019), Jim stumbled upon a good technique:

- CMSSW has been on GitHub since 2013.
- Many CMS physicists have to fork CMSSW at some point.
- Very few non-physicists would fork CMSSW.

So the technique is: select GitHub users who forked CMSSW ("CMS physicists") and look at all of their non-fork repos. 3 697 people, 22 961 repos over 10 years.

Analysis of source code online (the rest of this talk)

A few years ago (2019), Jim stumbled upon a good technique:

- CMSSW has been on GitHub since 2013.
- Many CMS physicists have to fork CMSSW at some point.
- Very few non-physicists would fork CMSSW.

So the technique is: select GitHub users who forked CMSSW ("CMS physicists") and look at all of their non-fork repos. 3 697 people, 22 961 repos over 10 years.

But what about experiments other than CMS?

A complementary dataset

- GitHub Archive (**https://www.gharchive.org/**) has been collecting all fork, PR, issue, wiki, watch, and comment events since 2017. We can get a list of GitHub users who have had any interaction at all with a specified repo.
- **https://github.com/root-project/root** seems like a logical choice to define "particle physicists."
- (Could also consider a set of repos.)
- (We can get a list of 13 069 root-forum users, but not their GitHub userids.)

A complementary dataset

- GitHub Archive (**https://www.gharchive.org/**) has been collecting all fork, PR, issue, wiki, watch, and comment events since 2017. We can get a list of GitHub users who have had any interaction at all with a specified repo.
- **https://github.com/root-project/root** seems like a logical choice to define "particle physicists."
- (Could also consider a set of repos.)
- (We can get a list of 13 069 root-forum users, but not their GitHub userids.)

So: select GitHub users who interacted with the ROOT repo ("particle physicists") and look at all of their non-fork repos. 2 824 people, 17 334 repos over 6 years.

A complementary dataset

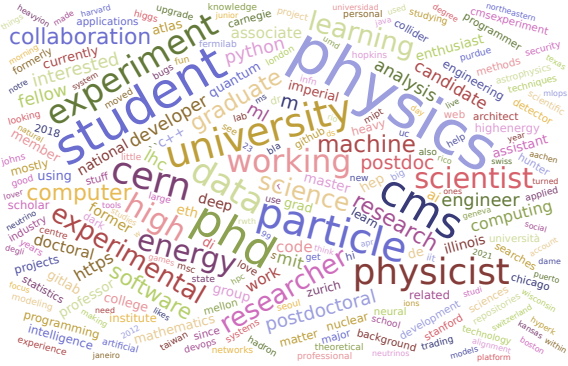
- GitHub Archive (**https://www.gharchive.org/**) has been collecting all fork, PR, issue, wiki, watch, and comment events since 2017. We can get a list of GitHub users who have had any interaction at all with a specified repo.
- **https://github.com/root-project/root** seems like a logical choice to define "particle physicists."
- (Could also consider a set of repos.)
- (We can get a list of 13 069 root-forum users, but not their GitHub userids.)

So: select GitHub users who interacted with the ROOT repo ("particle physicists") and look at all of their non-fork repos. 2 824 people, 17 334 repos over 6 years.

Interestingly, only 143 are in both (3.9% of CMSSW and 5.1% of ROOT).

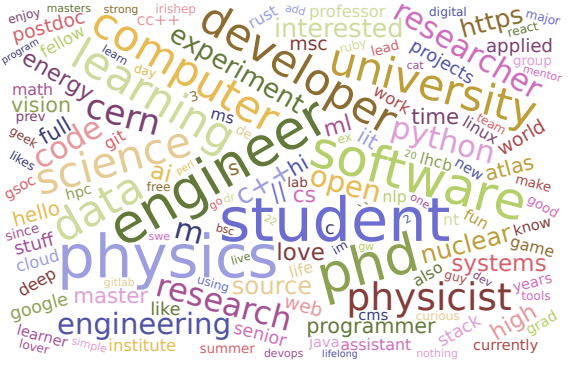
What they said in their profile bios

Selected by CMSSW fork



A lot of “physics,” “student,” “particle,” “physicist,” “PhD,” “CERN,” and “CMS.”

Selected by ROOT interaction



A little more “software,” “engineer,” but still lots of “physics,” “student,” and “PhD.”

What can we do once we have the repos?

Previously, Jim regex-searched them for "`import XYZ`" and "`#include<XYZ>`".

What can we do once we have the repos?

Previously, Jim regex-searched them for "`import XYZ`" and "`#include<XYZ>`".

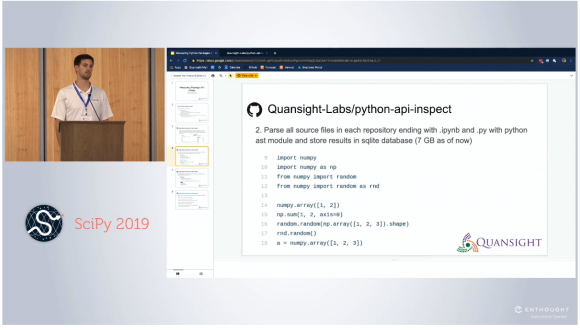
For this talk, we wanted to go further and build ASTs/statically analyze all of the code.

What can we do once we have the repos?

Previously, Jim regex-searched them for "`import XYZ`" and "`#include<XYZ>`".

For this talk, we wanted to go further and build ASTs/statically analyze all of the code.

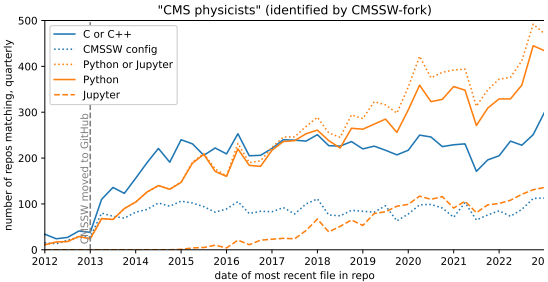
Prior art: see Chris Ostrouchov's Measuring API Usage (2019).

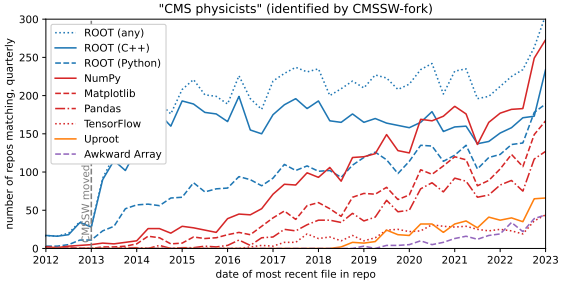


Day 2 Lightning Talks | SciPy 2019 |

But first, reproducing the previous studies

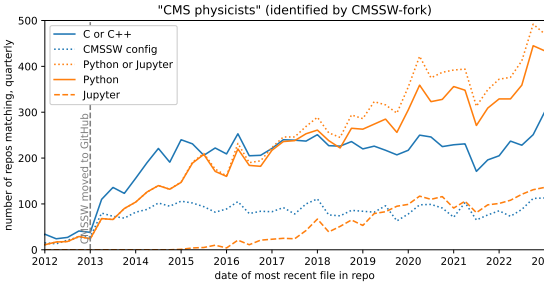
Note that the data have changed: more GitHub users have forked CMSSW since the last time we looked, which adds even their past history to the plot, and the date of a repo is set by the latest file, which causes them to migrate bins (forward in time).

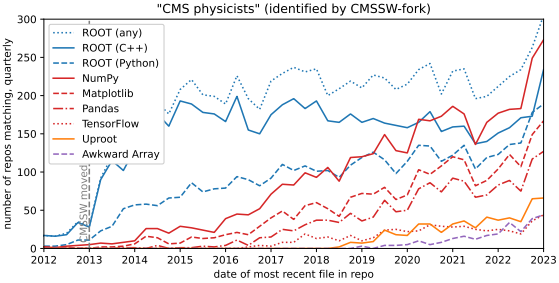




But first, reproducing the previous studies

Note that the data have changed: more GitHub users have forked CMSSW since the last time we looked, which adds even their past history to the plot, and the date of a repo is set by the latest file, which causes them to migrate bins (forward in time).

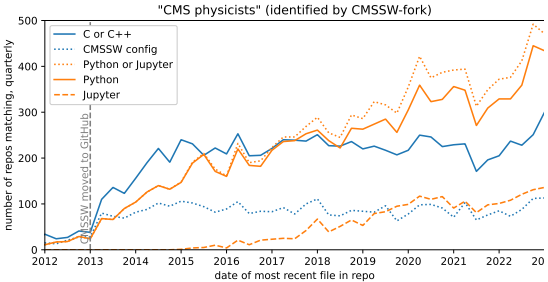




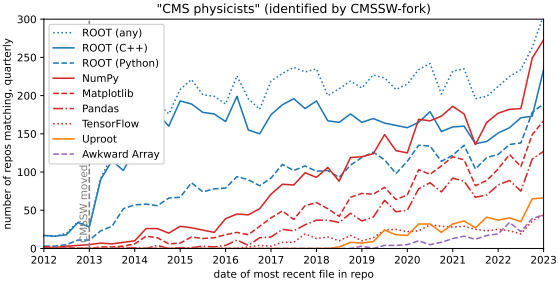
Conclusion is the same: C++ and CMSSW config (Python with `import FWCore`) are flat, while Python and Jupyter (Python) increase.

But first, reproducing the previous studies

Note that the data have changed: more GitHub users have forked CMSSW since the last time we looked, which adds even their past history to the plot, and the date of a repo is set by the latest file, which causes them to migrate bins (forward in time).



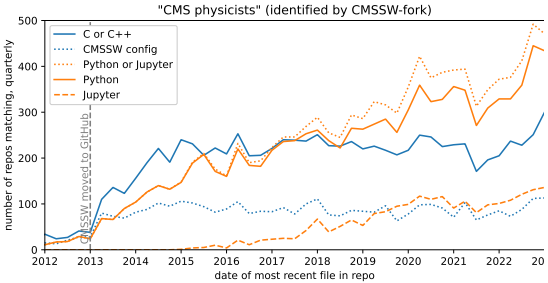
Conclusion is the same: C++ and CMSSW config (Python with `import FWCore`) are flat, while Python and Jupyter (Python) increase.



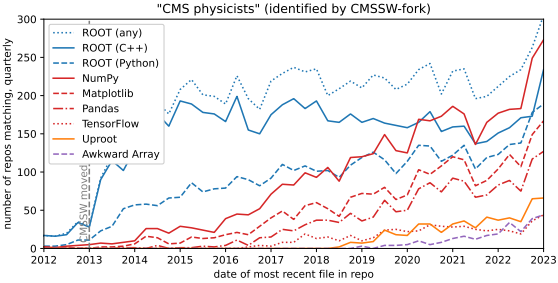
Conclusion is the same: ROOT-C++ usage is flat while PyROOT and especially NumPy, Matplotlib, Pandas, TensorFlow are increasing.

But first, reproducing the previous studies

Note that the data have changed: more GitHub users have forked CMSSW since the last time we looked, which adds even their past history to the plot, and the date of a repo is set by the latest file, which causes them to migrate bins (forward in time).

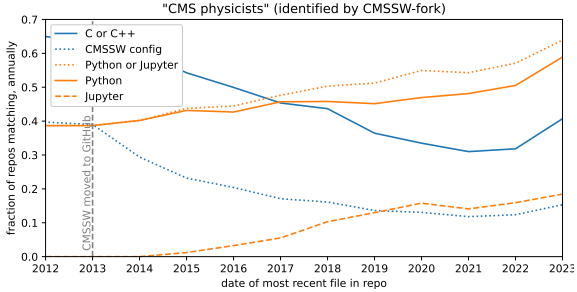


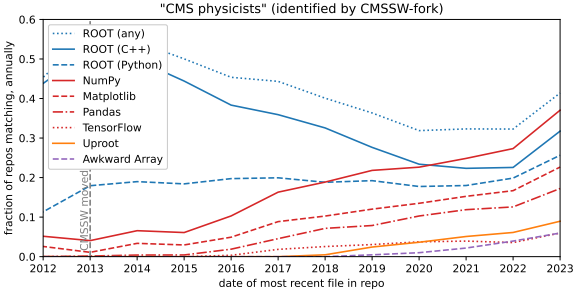
Conclusion is the same: C++ and CMSSW config (Python with `import FWCore`) are flat, while Python and Jupyter (Python) increase.



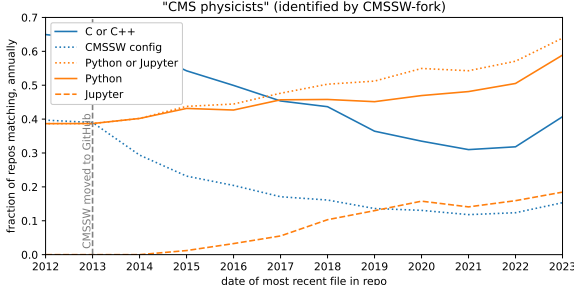
Conclusion is the same: ROOT-C++ usage is flat while PyROOT and especially NumPy, Matplotlib, Pandas, TensorFlow are increasing. Uproot/Awkward usage $\sim$ TensorFlow usage.

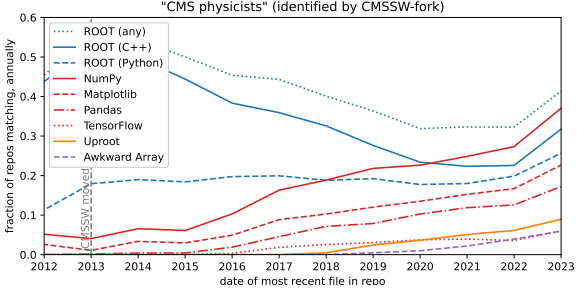
Better represented as fractions: #matching repos/#total repos

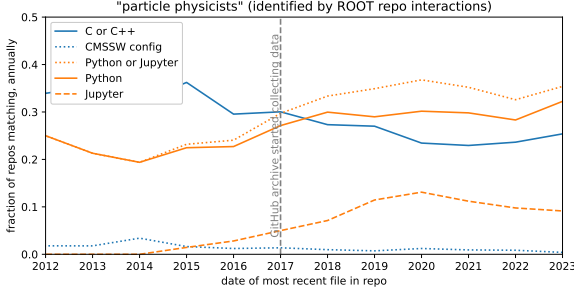


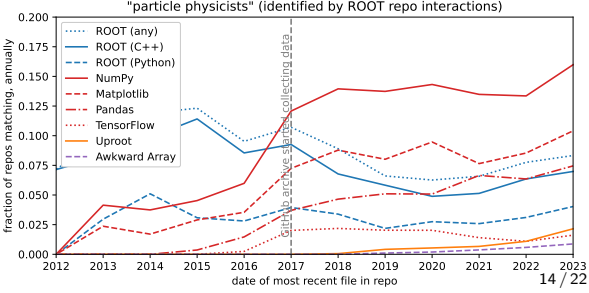


Better represented as fractions: #matching repos/#total repos

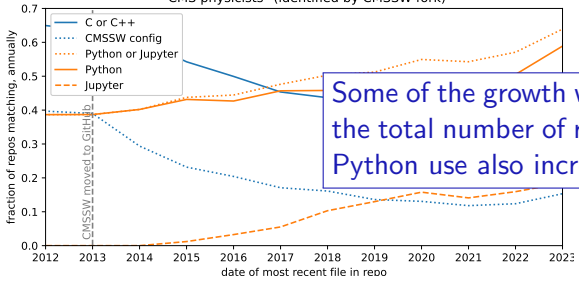


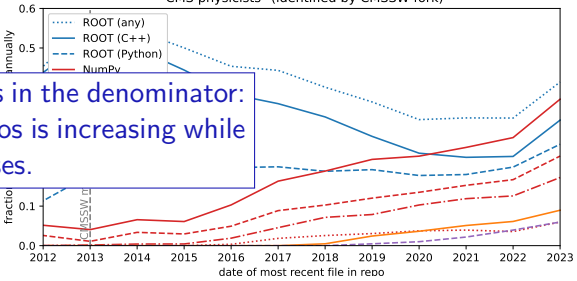




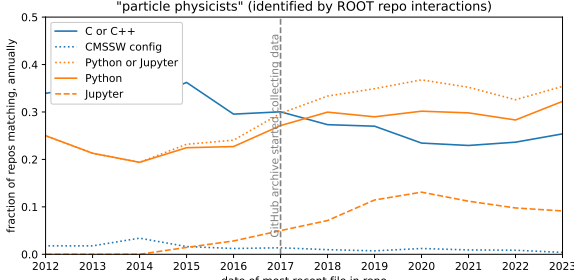


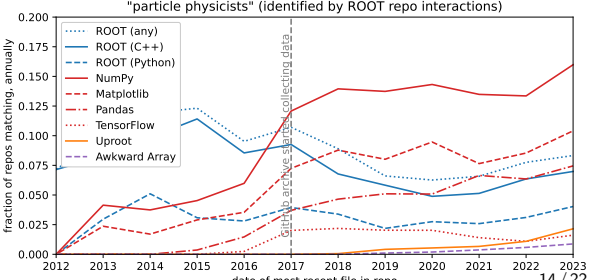
Better represented as fractions: #matching repos/#total repos



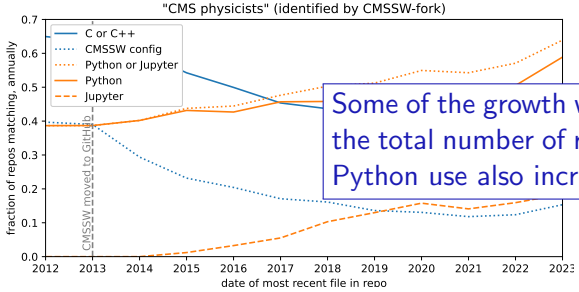


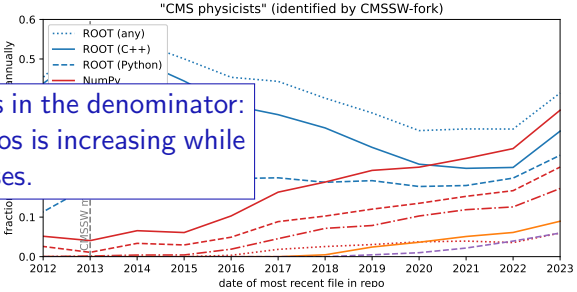
Some of the growth was in the denominator: the total number of repos is increasing while Python use also increases.



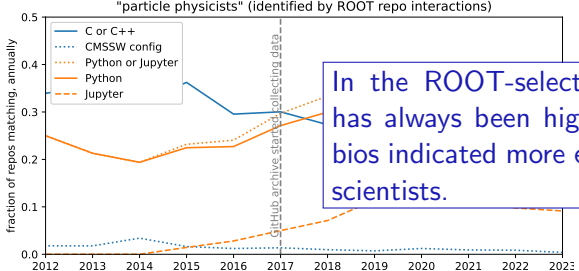


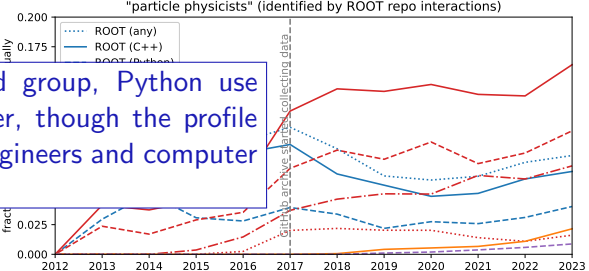
Better represented as fractions: #matching repos/#total repos





Some of the growth was in the denominator: the total number of repos is increasing while Python use also increases.

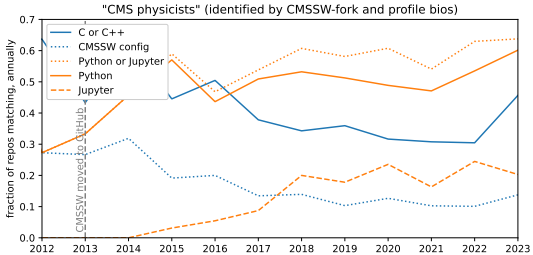


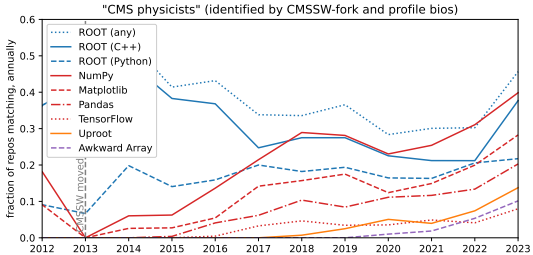


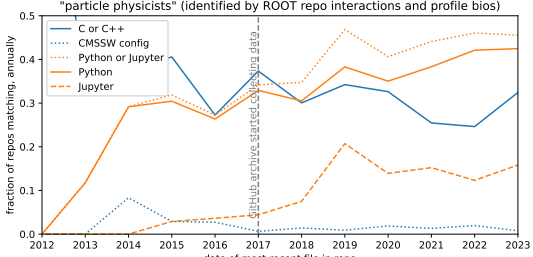
In the ROOT-selected group, Python use has always been higher, though the profile bios indicated more engineers and computer scientists.

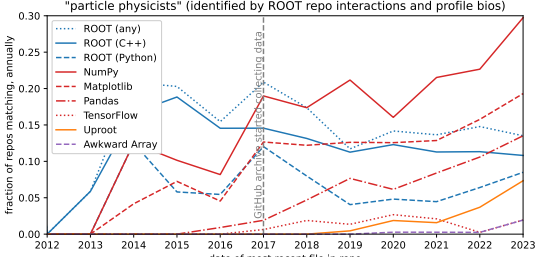
Narrow in on physicists, selecting by their profile bios

Regex `(phys|analy|hep|particle|cern|cms|atlas|alice|lhc)` selects 7.6% of users.



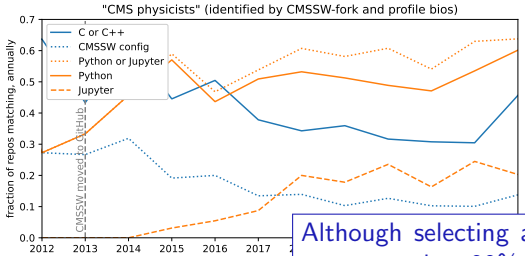


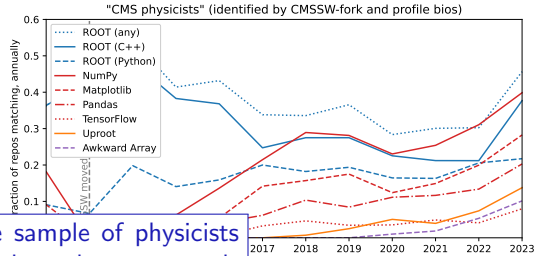




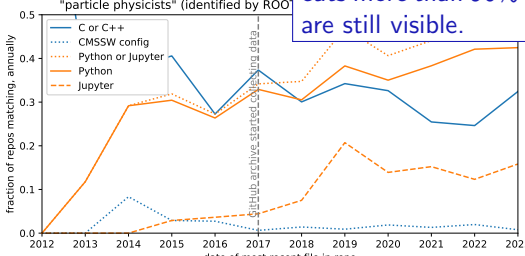
Narrow in on physicists, selecting by their profile bios

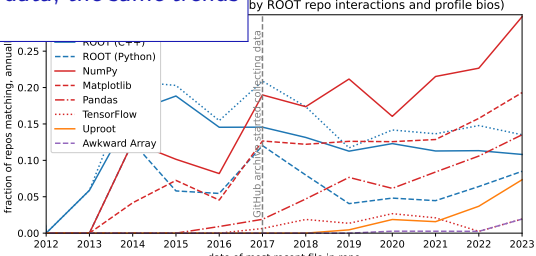
Regex `(phys|analy|hep|particle|cern|cms|atlas|alice|lhc)` selects 7.6% of users.



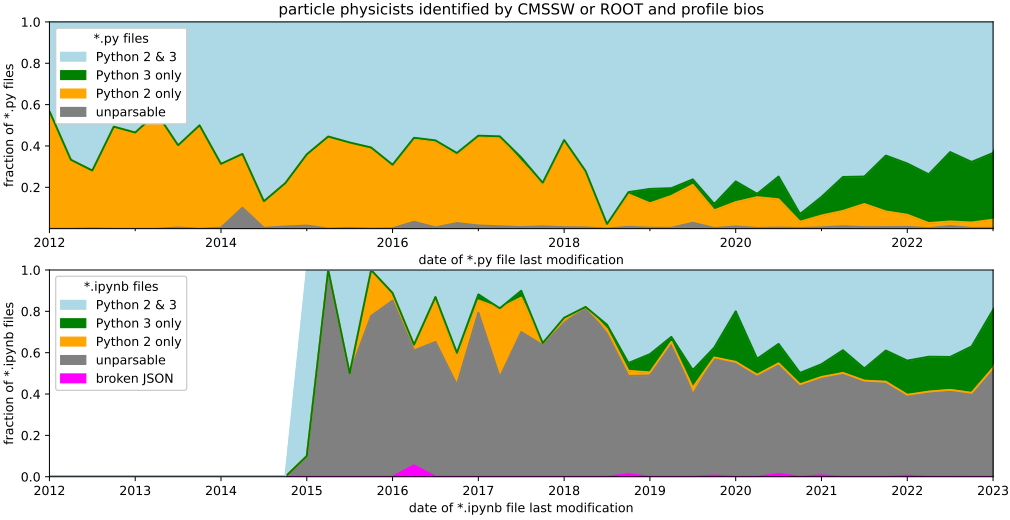


Although selecting a pure sample of physicists cuts more than 90% of the data, the same trends are still visible.

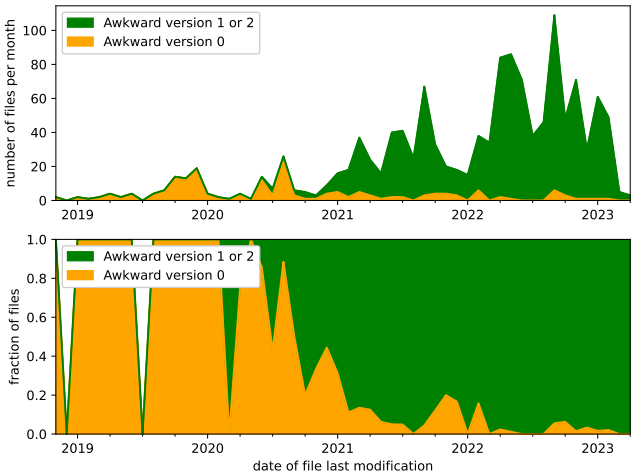




Now actually parse the repos: Python 3 adoption among physicists



Ask specific questions: Awkward 1 adoption by function name




```
def is_awkward0(obj):
    return obj.function.name.startswith(
        "ak.JaggedArray"
    ) or obj.function.name.startswith(
        "ak.array.jagged.JaggedArray"
    ) or obj.function.name in (
            "ak.IndexedArray",
            "ak.Table",
            "ak.fromarrow",
            "ak.fromiter",
            "ak.hdf5",
            "ak.load",
            "ak.save",
            "ak.toarrow",
            "ak.topandas",
            "ak.util.concatenate",
        )
```

Most common function calls/argument patterns

	Awkward Array		Uproot
2832	`ak.flatten(?)`	2150	`uproot.open(?)`
2498	`ak.num(?)`	889	`uproot.open('str')`
2193	`ak.to_numpy(?)`	198	`uproot.recreate(?)`
874	`ak.sum(?, axis=1)`	179	`uproot.tree.TBranchMethods.array(?)`
865	`ak.flatten(?, axis=None)`	74	`uproot.lazy(?)`
564	`ak.sum(?)`	58	`uproot.newtree(?)`
455	`ak.ones_like(?)`	57	`uproot.pandas.iterate(?, 'str', ['stri`
406	`ak.Array(?)`	44	`uproot.open(?, xrootdsource=?)`
283	`ak.concatenate(?)`	23	`uproot.lazy(?, filter_name=?)`
265	`ak.singletons(?)`	22	`uproot.recreate('str')`
248	`ak.num(?, axis=1)`	18	`uproot.create(?)`
246	`ak.concatenate(?, axis=1)`	15	`uproot.recreate(?, compression=?)`
235	`ak.any(?, axis=1)`	13	`uproot.newbranch(?, size='str')`
234	`ak.zip(?, with_name='str')`	11	`uproot.numentries(?, ?)`
233	`ak.to_pandas(?)`	11	`uproot.ArrayCache('str')`
226	`ak.unzip(?)`	10	`uproot.numentries(?, ?, total=False)`
221	`ak.firsts(?)`	10	`uproot.numentries(?, ?, executor=?, to`

Most common function calls/argument patterns

	Awkward Array
2832	`ak.flatten(?)`
2498	`ak.num(?)`
2193	`ak.to_numpy(?)`
874	`ak.sum(?, axis=1)`
865	`ak.flatten(?, axis=None)`
564	`ak.sum(?)`
455	`ak.ones_like(?)`
406	`ak.Array(?)`
283	`ak.concatenate(?)`
265	`ak.singletons(?)`
248	`ak.num(?, axis=1)`
246	`ak.concatenate(?, axis=1)`
235	`ak.any(?, axis=1)`
234	`ak.zip(?, with_name='str')`
233	`ak.to_pandas(?)`
226	`ak.unzip(?)`
221	`ak.firsts(?)`

	Uproot
2150	`uproot.open(?)`
889	`uproot.open('str')`
198	`uproot.recreate(?)`
179	`uproot.tree.TBranchMethods.array(?)`
74	`uproot.lazy(?)`
58	`u`
57	`u` stri
44	`u`
23	`u`
22	`uproot.recreate('str')`
18	`uproot.create(?)`
15	`uproot.recreate(?, compression=?)`
13	`uproot.newbranch(?, size='str')`
11	`uproot.numentries(?, ?)`
11	`uproot.ArrayCache('str')`
10	`uproot.numentries(?, ?, total=False)`
10	`uproot.numentries(?, ?, executor=?, to`

Uproot relies more on object methods. We'd have to statically analyze *object types*, not functions on global modules, which is hard in a dynamically typed language.

Most common function calls/argument patterns

Awkward Array

2832	`ak.flatten(?)`
2498	`ak.num(?)`
2193	`ak.to_numpy(?)`
874	`ak.sum(?, axis=1)`
865	`ak.flatten(?, axis=None)`
564	`ak.sum(?)`
455	`ak.ones_like(?)`
406	`ak.Array(?)`
283	`ak.concatenate(?)`
265	`ak.singletons(?)`
248	`ak.num(?, axis=1)`
246	`ak.concatenate(?, axis=1)`
235	`ak.any(?, axis=1)`
234	`ak.zip(?, with_name='str')`
233	`ak.to_pandas(?)`
226	`ak.unzip(?)`
221	`ak.firsts(?)`

Compare to web traffic on awkward-array.org...

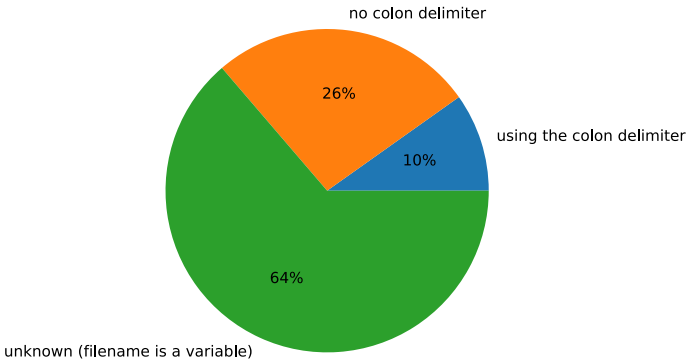
function	#unique visitors	#views	avg. time
ak.Array	785	1100	3m33s
ak.concatenate	223	293	4m35s
ak.count	210	265	4m20s
ak.flatten	203	242	4m23s
ak.where	202	262	3m54s
ak.num	184	235	3m07s
ak.to_numpy	181	218	3m25s
ak.mask	178	231	3m52s
ak.zip	163	221	5m02s
ak.fill_none	162	214	3m11s
ak.broadcast_arrays	156	210	4m20s
ak.combinations	136	171	3m58s
ak.sum	136	165	4m42s
ak.behavior	125	152	6m25s
ak.ArrayBuilder	124	161	3m02s
ak.cartesian	121	159	3m09s
ak.pad_none	114	146	3m00s

Use feature adoption to make decisions about deprecation

The colon in `uproot.open("file.root:dir/tree")` causes many problems:

- **Test a URL with an HTTP port number** scikit-hep/uproot5#47
- **uproot4 unable to open ROOT file with colons in the name that uproot3 can** scikit-hep/uproot5#79
- **Simplify file path/object splitter** scikit-hep/uproot5#80
- **Removed the colon-parsing and replaced it with dicts.** scikit-hep/uproot5#81
- **Reading in multiple root files into pandas/dask DataFrame** scikit-hep/uproot5#129
- **Reading an object with colon in its name** scikit-hep/uproot5#365
- **Not possible to escape colon in filenames / avoid object-in-file path syntax (?)** scikit-hep/uproot5#541
- **Aliases and cuts when reading ROOT file** scikit-hep/uproot5#543
- **Accessing ROOT files with colons and double slashes in the path** scikit-hep/uproot5#669
- **pathlib.Path drops '//' (naturally), but it's sometimes used for URLs** scikit-hep/uproot5#670
- **fix: Ignore semicolon in EOS token when separating file name and object name** scikit-hep/uproot5#875

Use feature adoption to make decisions about deprecation

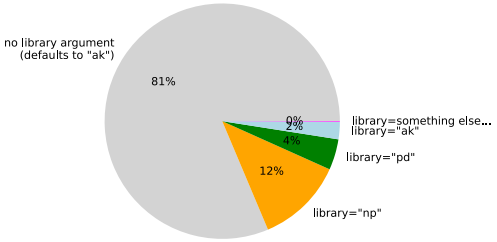


But removing it would upset at least 10% of workflows.

The deprecation period has to be *long*, if it is to be removed at all.

Focus on Uproot's array-fetching functions

How do people use the `library="??"` argument?



When it's used, it's much more often used for NumPy than for Pandas.

```
if isinstance(tree, ast.Call):
    name = ast.unparse(tree.func)
    if (   # select Uproot functions only
        (name.endswith(".array") and name not in (
            "np.array", "np.ma.array", "numpy.array",
            "NUMPY_LIB.array", "array.array",
            "self.NUMPY_LIB.array", "cupy.array",
        ))
        or name.endswith(".arrays")
        or name.endswith(".iterate")
        or (name.endswith(".concatenate") and name not
            "np.concatenate", "ak.concatenate",
            "awk.concatenate", "awkward.concatenate",
            "awkward.JaggedArray.concatenate",
            "JaggedArray.concatenate",
            "tf.concatenate",
        ))
        or name.endswith(".dask")
    ):
        matches.append(tree)
```

What libraries are Awkward and Uproot used *with*?

Awkward Array

numpy	90.5%	torch	4.2%
uproot	56.9%	seaborn	3.7%
matplotlib	49.8%	yahist	3.6%
coffea	35.6%	xgboost	3.2%
pandas	31.2%	sklearn	2.9%
mplhep	20.4%	h5py	2.9%
ROOT	11.9%	memory_profiler	2.6%
numba	11.8%	pympler	2.3%
hist	8.8%	psutil	2.1%
uproot_methods	8.4%	correctionlib	1.9%
yaml	8.2%	sortedcontainers	1.8%
utils	7.4%	cycler	1.7%
tqdm	6.7%	networkx	1.7%
boost_histogram	5.8%	pylab	1.5%
tensorflow	5.0%	PIL	1.5%
scipy	4.8%	helpers	1.4%
vector	4.3%	tabulate	1.4%

Uproot

numpy	88.5%	seaborn	3.9%
matplotlib	59.4%	hist	3.9%
pandas	46.5%	boost_histogram	3.9%
awkward	31.7%	keras	3.5%
ROOT	23.6%	CMS_lumi	3.5%
coffea	14.0%	histo_utilities	3.1%
mplhep	13.8%	analysis_utilities	3.1%
tqdm	11.0%	torch	2.9%
tensorflow	9.4%	h5py	2.8%
scipy	8.2%	progressBar	2.8%
sklearn	7.0%	cebefo_style	2.3%
uproot_methods	6.2%	lumi_utilities	2.1%
xgboost	6.0%	yahist	1.9%
yaml	5.8%	common	1.8%
numba	5.8%	config	1.8%
utils	5.1%	root_pandas	1.8%
root_numpy	4.5%	psutil	1.6%

Conclusions

- A lot of physics analysis code is public on GitHub and GitLab.

Conclusions

- A lot of physics analysis code is public on GitHub and GitLab.
- We can find it by crawling the network, seeded by a package that is well-known in the community, such as CMSSW or ROOT.

Conclusions

- A lot of physics analysis code is public on GitHub and GitLab.
- We can find it by crawling the network, seeded by a package that is well-known in the community, such as CMSSW or ROOT.
- Studying tens of thousands of git repos is a modest data analysis (TB scale).
  - Dask was very helpful!

Conclusions

- A lot of physics analysis code is public on GitHub and GitLab.
- We can find it by crawling the network, seeded by a package that is well-known in the community, such as CMSSW or ROOT.
- Studying tens of thousands of git repos is a modest data analysis (TB scale).
  - Dask was very helpful!
- We can learn things that are useful for software library maintenance:
  - user adoption of new versions
  - most common function-call patterns
  - decide if and when a feature can be deprecated
  - discover which libraries are being used together, maybe motivate integrations

Conclusions

- A lot of physics analysis code is public on GitHub and GitLab.
- We can find it by crawling the network, seeded by a package that is well-known in the community, such as CMSSW or ROOT.
- Studying tens of thousands of git repos is a modest data analysis (TB scale).
  - Dask was very helpful!
- We can learn things that are useful for software library maintenance:
  - user adoption of new versions
  - most common function-call patterns
  - decide if and when a feature can be deprecated
  - discover which libraries are being used together, maybe motivate integrations
- It's hard to identify class method calls in a dynamically typed language!

Conclusions

- A lot of physics analysis code is public on GitHub and GitLab.
- We can find it by crawling the network, seeded by a package that is well-known in the community, such as CMSSW or ROOT.
- Studying tens of thousands of git repos is a modest data analysis (TB scale).
  - Dask was very helpful!
- We can learn things that are useful for software library maintenance:
  - user adoption of new versions
  - most common function-call patterns
  - decide if and when a feature can be deprecated
  - discover which libraries are being used together, maybe motivate integrations
- It's hard to identify class method calls in a dynamically typed language!

How to get the analysis code (source data are in public S3 buckets):

https://github.com/jpivarski-talks/2023-05-09-chep23-analysis-of-physicists