

A photograph of a forest with a blue lightning bolt striking a tree trunk. The lightning bolt is bright blue and white, with multiple branches extending across the upper half of the image. The forest floor is covered in fallen leaves and branches, and the trees are tall and thin. The overall color palette is dominated by greens and browns, with the blue lightning bolt providing a stark contrast.

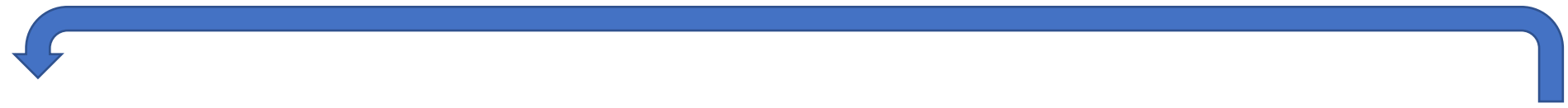
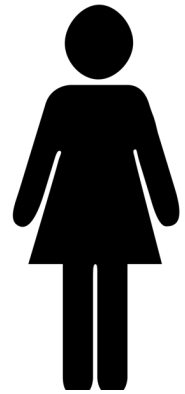
Scalable and Distributed Metadata Extraction with Xtract

Tyler J. Skluzacek (*and Kyle, Ian, Ryan, Zhuozhao, Yadu, Logan, Ben B., ...*)

Data Lifecycle and Scalable Workflows Group
Oak Ridge National Laboratory

ParslFest & FuncX Fest 2022

Scientists generate plentiful data artifacts along the research data lifecycle [Berman, '18]



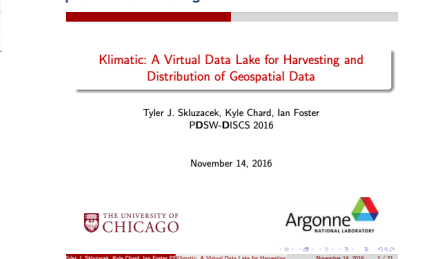
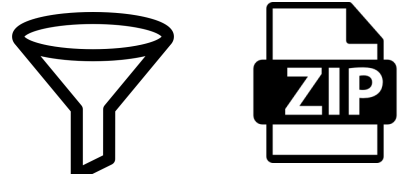
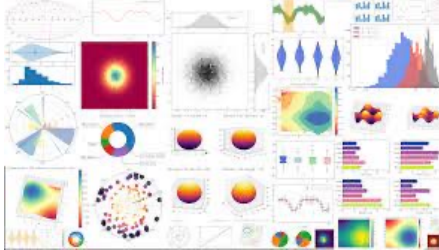
Acquire

Clean

Use/
Reuse

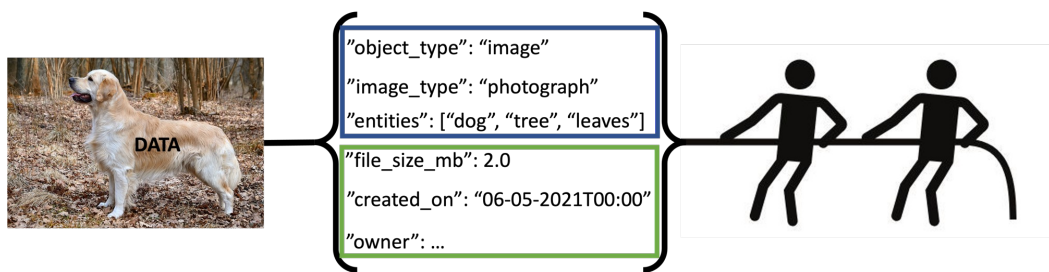
Publish

Preserve/
Destroy

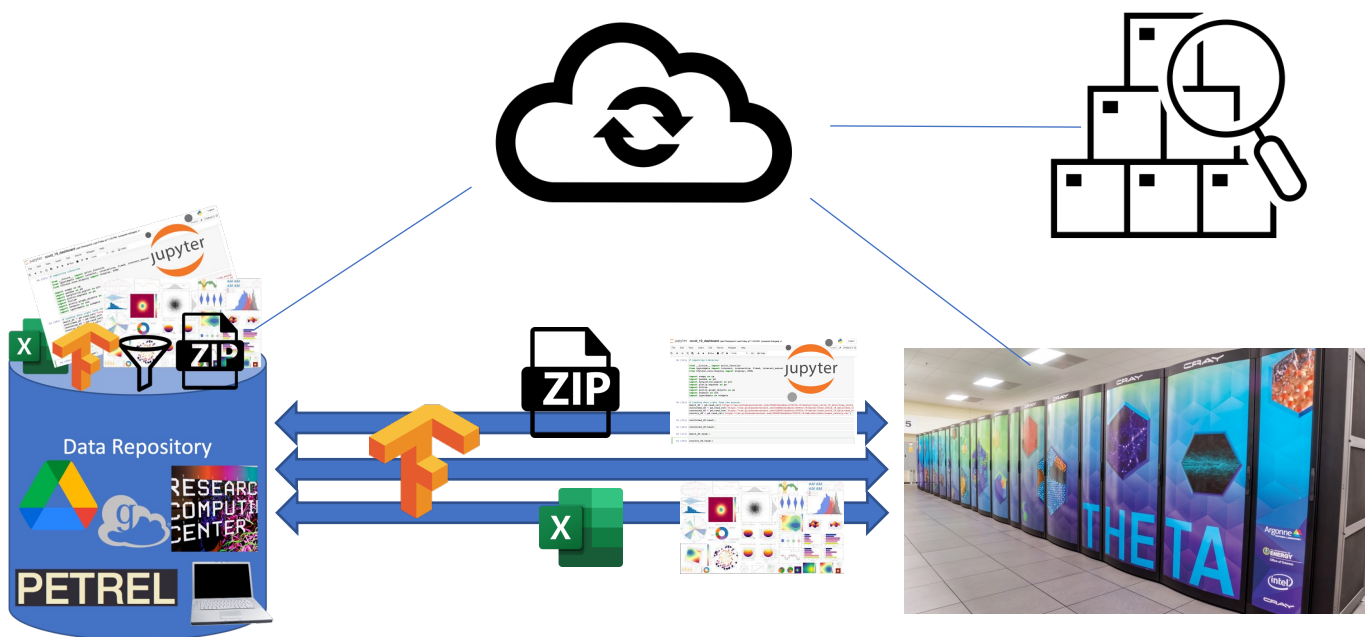


Metadata extraction can help make these data navigable

Option 1: Manually
(olden days)
“Human annotation”



Option 2: Automatically
(Today's wonder years)
Metadata Extraction Systems

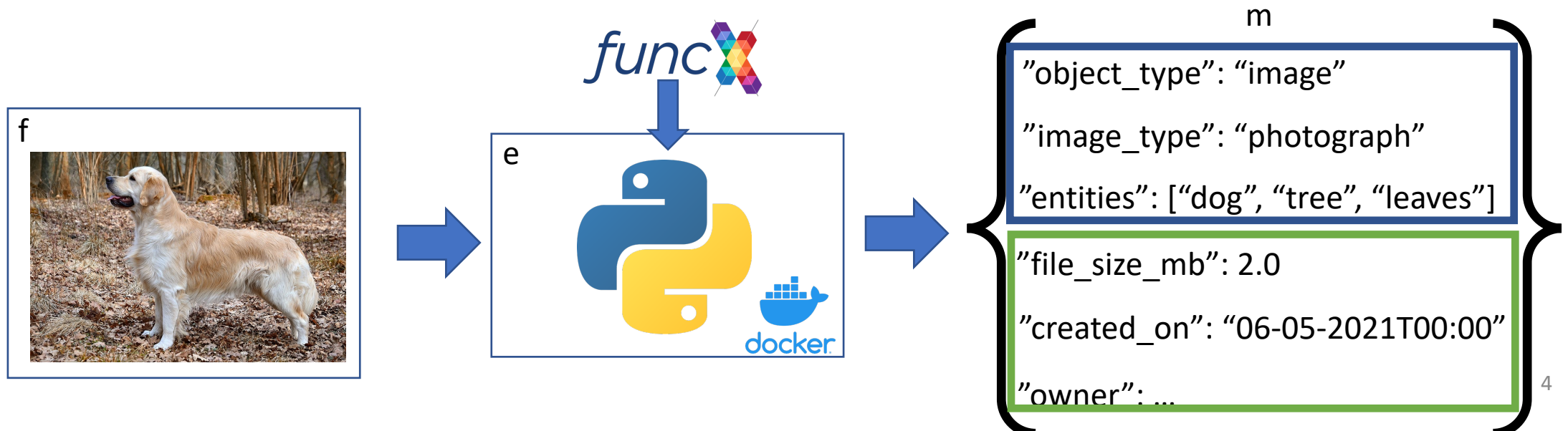


How to extract metadata from files of very different types?

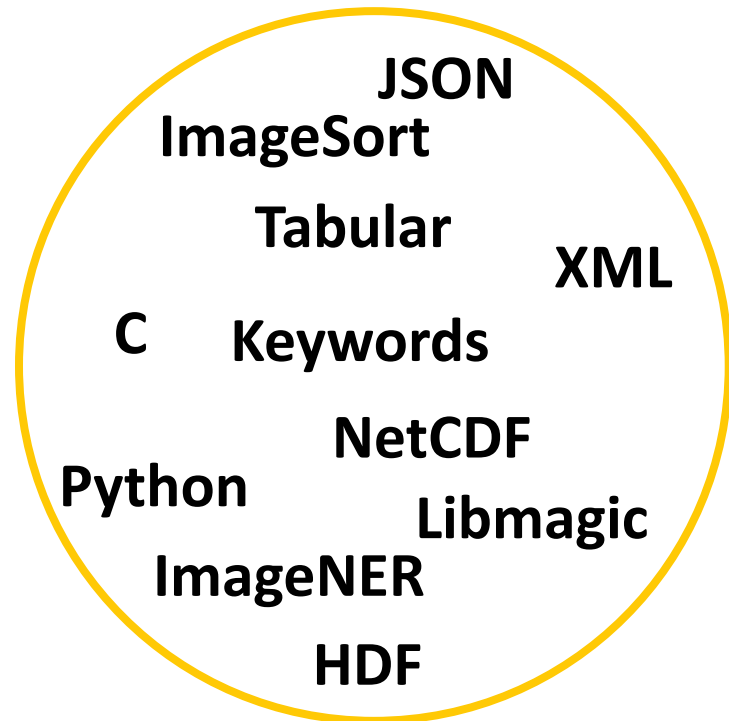
Repository R has collection of files $f \in R$

M_f is universe of all possible metadata for f

Set of **extractors** ε where $e \in \varepsilon$ is a function $e(f)$ that returns a (potentially empty) set of new metadata elements $m \in M_f$



We have developed a broad extractor library that illuminates the long-tail of science data

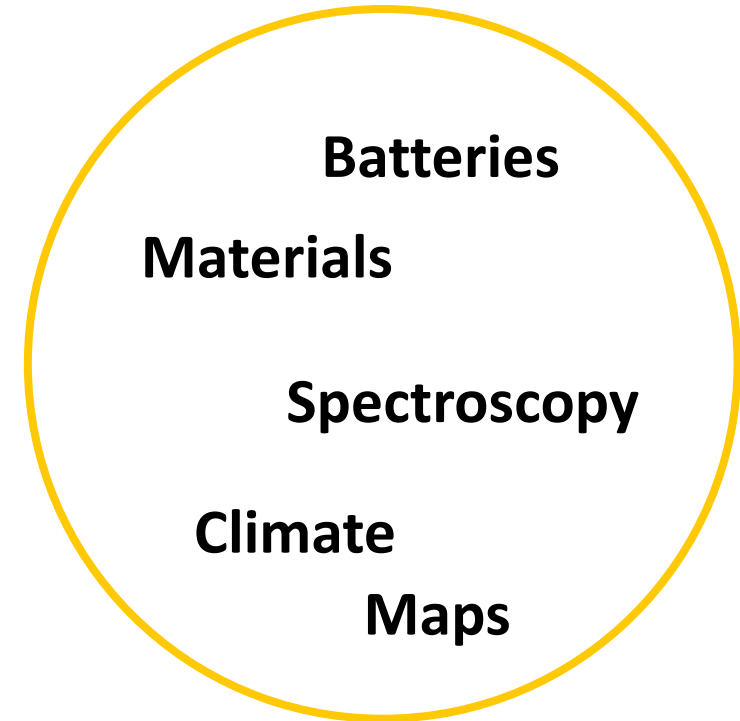


Principles for Extractor Design

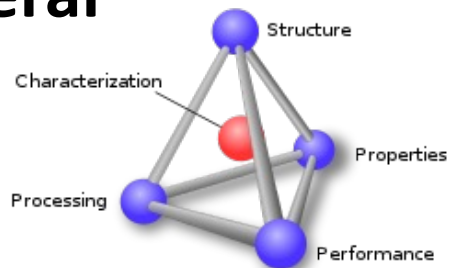
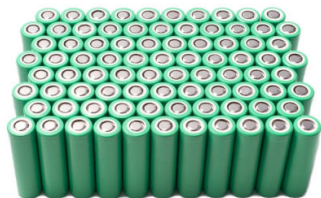
1. Relevant metadata
2. Correct metadata
3. Lightweight execution
4. Flexible to similar schema
5. Modular execution



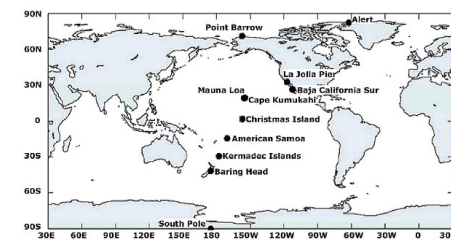
see Tyler's thesis for more information



General



Domain



Extractors as *funcX* functions

```
1 def base_extractor(event):
2     from xtract_sdk.agent.xtract import XtractAgent
3
4     # Load endpoint configuration. Init the XtractAgent.
5     xtra = XtractAgent(xtract_dir=event['xtract_dir'],
6                       sys_path_add=event['sys_path_add'],
7                       module_path=event['module_path'],
8                       recursion_depth=event['recursion_limit'],
9                       metadata_write_path=event['metadata_write_path'])
10
11     # Execute the extractor on the family_batch.
12     xtra.execute_extractions(family_batch=event['fam_batch'], input_type=event['type'])
13
14     # All metadata are held in XtractAgent's memory. Flush to disk!
15     paths = xtra.flush_metadata_to_files(writer=event['writer'])
16     stats = xtra.get_completion_stats()
17     stats['mdata_paths'] = paths
18
19     return stats
```

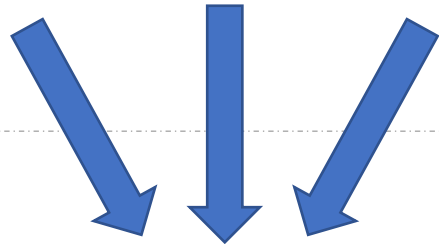
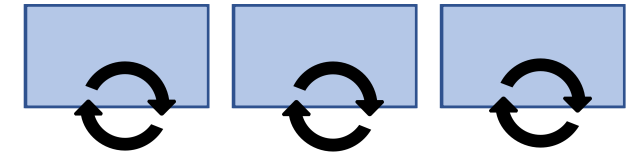
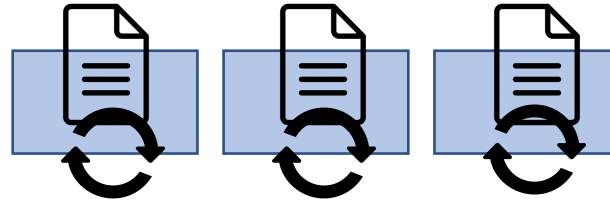
Uses funcX “container” workers



Check out our extractor library
to see how hotdogs are made

<https://github.com/xtracthub/xtract-tabular>
<https://github.com/xtracthub/xtract-keyword>
<https://github.com/xtracthub/xtract-python>
<https://github.com/xtracthub/xtract-images>
<https://github.com/xtracthub/xtract-matio>
... and more!

the edge



the computing center

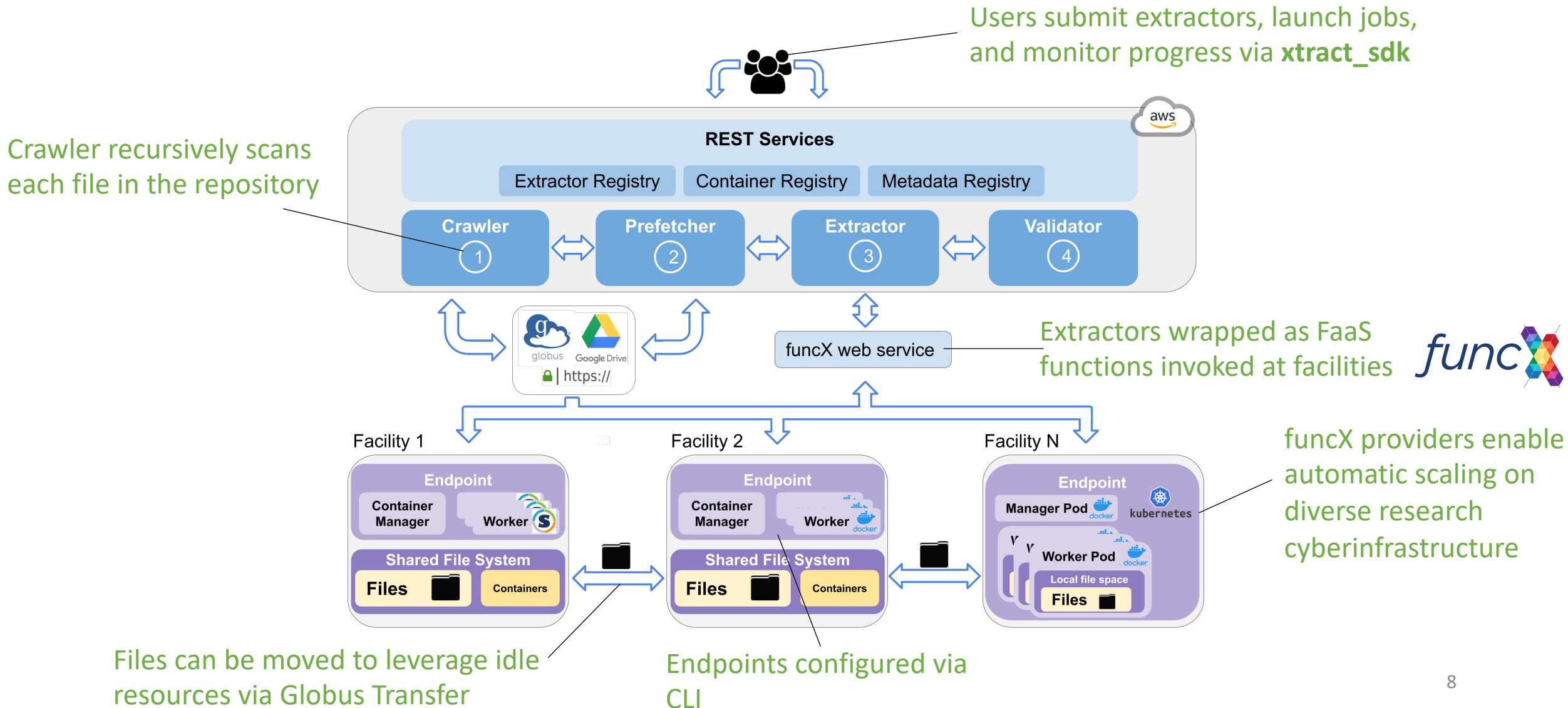


extract here

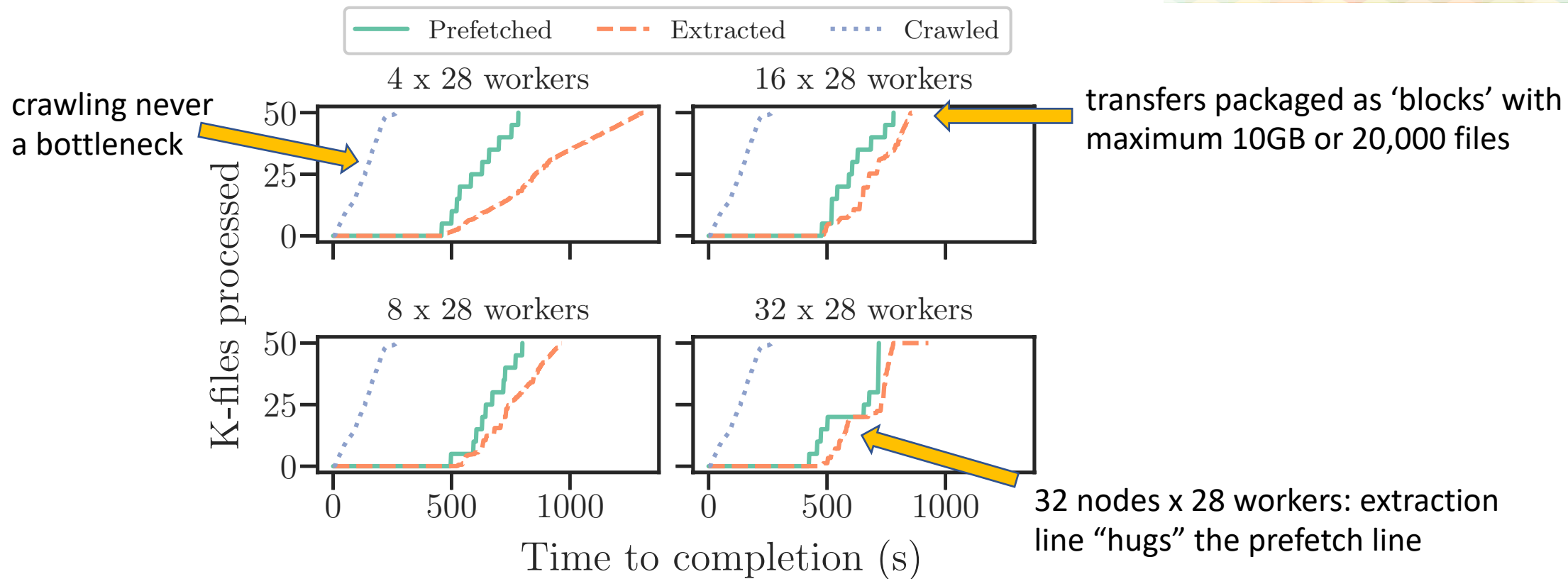
extract there

extract anywhere

Xtract: the metadata extraction system for science

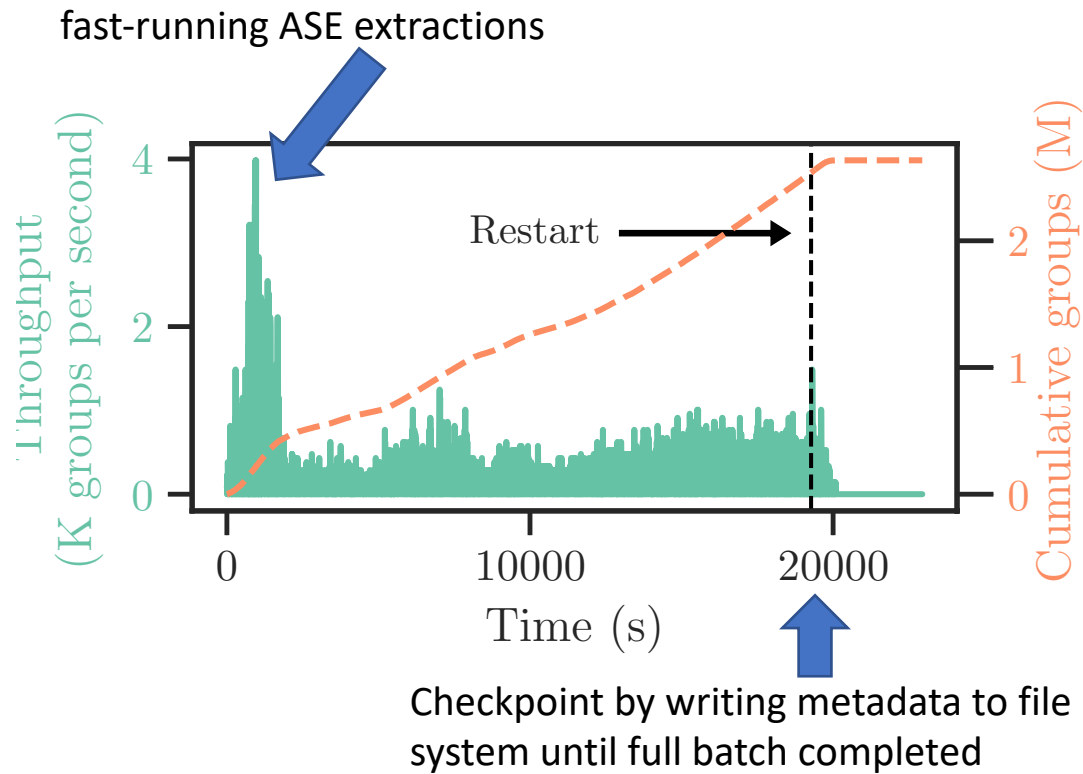


Extract here: full repository transfer

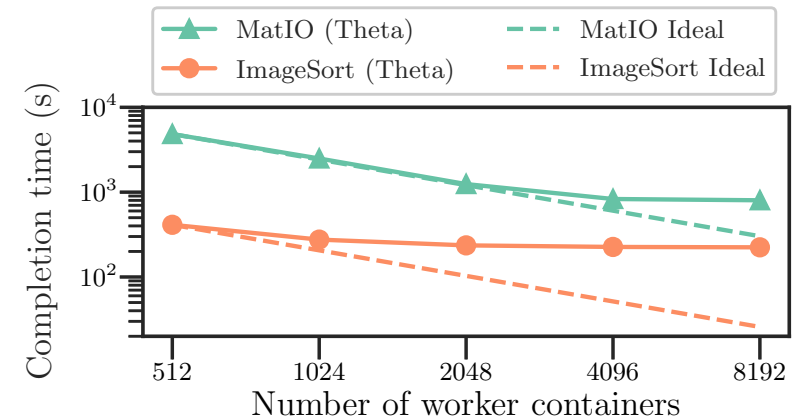


Bulk metadata extraction times for an MDF subset (50,000 files) processed on 4—32 UChicago Midway2 nodes.

Extract there: process 60TB (2.2 million groups) using the Theta supercomputer in **just over 6 hours**



MatIO sees *reasonable* scaling up to 4,096 workers

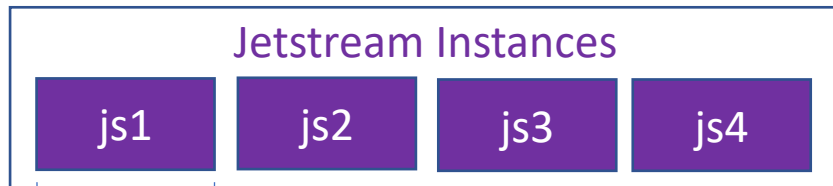
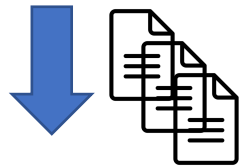


(a) Strong scaling



(b) Weak scaling

Extract anywhere: automatically offload files to underutilized compute facilities



Each Jetstream instance either invoking Xtract extractors (via funcX) OR is running the same number of workers on Tika

System	Percentage Transferred (%)	Transfer Time (s)	Completion Time (s)
Xtract	0%	0	1696
	10%	374	1560
	20%	655	1662
Apache Tika	0%	0	2032
	10%	384	1868
	20%	649	1935

10% is optimal, as Midway2 and Jetstream finish at approximately the same time

Global Portal Framework / Search

search box

facets

Creator

239548
154761
106487
96513
87349
85074
69962
54800
47109
40022
39335
2187
20101
17248
13881
12299
10630
10472
8500
6130

Results

1299232 datasets found

relevant dataset count

clickable dataset names

preview information

Parent Folder
Cycle
Publication Year

2020-2
2022

Parent Folder
Cycle
Publication Year

2020-2
2022

Parent Folder
Cycle
Publication Year

2020-1
2022

(shoutout to Nick Saint)

Spectroscopy: Globus Search Portal Interface

Battery Data Retrieval

```
In [2]: from batteryarchive.battery_sdk import Battery
```

Python SDK

Capability 1: Discover relevant datasets

We first provide the ability to 'drill down' into Battery to determine what data are even available, and fetch the relevant data into memory. To do this, we have a `count_and_collect(operator_ls, groupby)` function that accepts a list of AND-style clauses to return the count of matching records from a metadata database. For instance, you can count the number of cycle-testing files greater than 33 degrees using the anode 'graphite', as follows:

```
In [3]: # Create a Battery object, print the tallies for each file.
xb = Battery()
xb.count_and_collect([('min_temp_c', '>=', 32.),
                    ('max_temp_c', '<=', 48.)], groupby='cathode')

Out[3]: {'groups': [{'LCO': 7}, {'NCA': 2}, {'NMC': 4}], 'count': 13}
```

relevant dataset count

Capability 2: List metadata

For exploratory purposes, we might want to list the metadata counted in a call to `xb.count_and_collect()`. To accomplish this, we provide the `df_dump()` endpoint that converts these contents to a pandas dataframe. It does not take any arguments, but does require first running `xb.count_and_collect()` to identify data to list.

```
In [4]: df = xb.df_dump()
print(df)
```

dataset information

	id	filename	cathode	anode
0	5	18650_NMC_35C_0-100_0.5-1C_b_timeseries.csv	NMC	graphite
1	8	4_pouch_LCO_40C_0-100_2-1.84C_d_timeseri...	LCO	graphite
2	14	5_pouch_LCO_40C_0-100_2-1.84C_f_timeseri...	LCO	graphite
3	17	18650_NMC_35C_0-100_0.5-1C_d_timeseries.csv	NMC	graphite
4	26	18650_NMC_35C_0-100_0.5-1C_e_timeseries.csv	NMC	graphite

Capability 3: Construct graphs

```
In [4]: xb.get_charge_discharge_curves(67, cycles=[50, 100], graph_type='discharge', x_val='capacity')
xb.get_charge_discharge_curves(105, cycles=[50, 100], graph_type='discharge', x_val='capacity')
```

graph support

Battery Modeling: Python SDK on JupyterHub

Things I'm particularly excited about

- the funcX container service
- service-owned endpoints
- Minnesota Vikings 2023 Super Bowl run

Xtract is (soon-to-be, again) available for use

Future work

- Shared micro-extractor utilities to support shared extraction logic
- Combine 'workflow' and file metadata
- **Application:** AI-enhanced storage to support ML applications

Want to learn more?

skluzacektj@ornl.gov

We're building in the open:

<https://github.com/xtracthub>