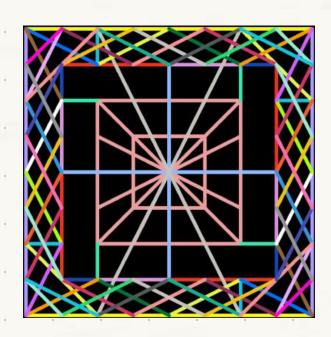


Understanding API Dispatching in NetworkX



About Me

- · Aditi Juneja (@Schefflera-Arboricola)
- Worked mostly around API Dispatching stuff and the nx-parallel backend
- · Network X's core developer
- · PyDelhi Volunteer Mostly proposal reviews



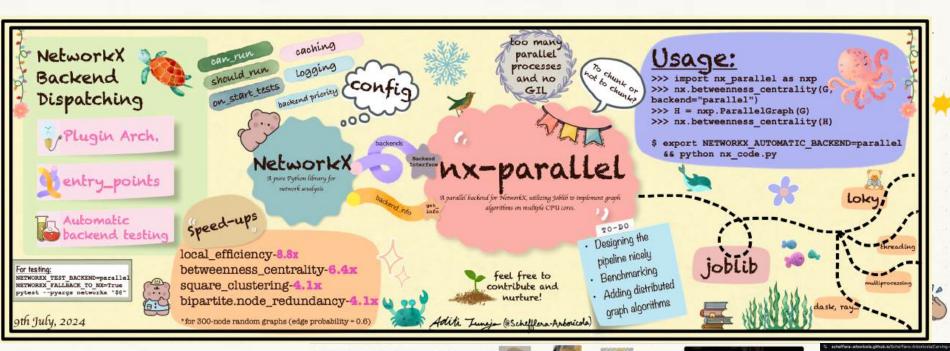
Slides - https://github.com/Schefflera-Arboricola/blogs/blob/main/archive

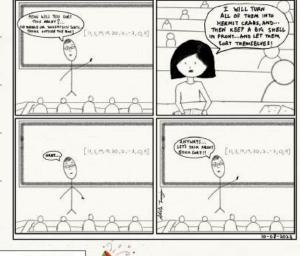
PyDelhi Meetups!

- In-person meetups conducted on 3rd Saturday of every month
- Submit your talk proposal here https://github.com/pydelhi/talks/issues
- Find us on Telegram, WhatsApp, and know more at https://pydelhi.org/

And...here is some of my stuff...







Google Summer of Code 2024 Final Report

Project Title - Beyinting and expanding me-paratiel Organisation - NumPCCUS(NotworkX: nx-paratiel) Mentors - Dan Schult, Mrisul Seth Contributor - Adb Junea.

1.Abstra

This report distalls the anche conceptioned 2005, Society on the re-peated project, a preside inversigne (2005). For sourcing grash algorithms are project inverside intellegating the goldy and the rate re-peated and adding an pre-control those is used A-collisionally, the impact invested selecting to set. the previously added algorithms and enhancing it department, as they stall, a polarity the data synthem and inflamming the furnity and their Synthesis, and inflamming the furnity and their synthesis and synthesis and their synthesis and syn

The contributions made studing this perior and this report, along with my blogs, will serve as sondraying this work.

2.Background

The mr-parellel package is a backened the execution of peph significant has been been a second or the control of the control



Carving out a path in a garden

Pull/listed on Says 11, 2023

There is a beautiful garden composed of several smaller flower gardens. It's soor going to open to the general public. Before that, the gardener wants to carve out path and pove it with concrete, allowing visitors to strall around the garden. He w to put in the least effort and time, while also minimizing the damage to the groun and flowers. He also wants all the flower gardens' centers to be accessible from nonther.









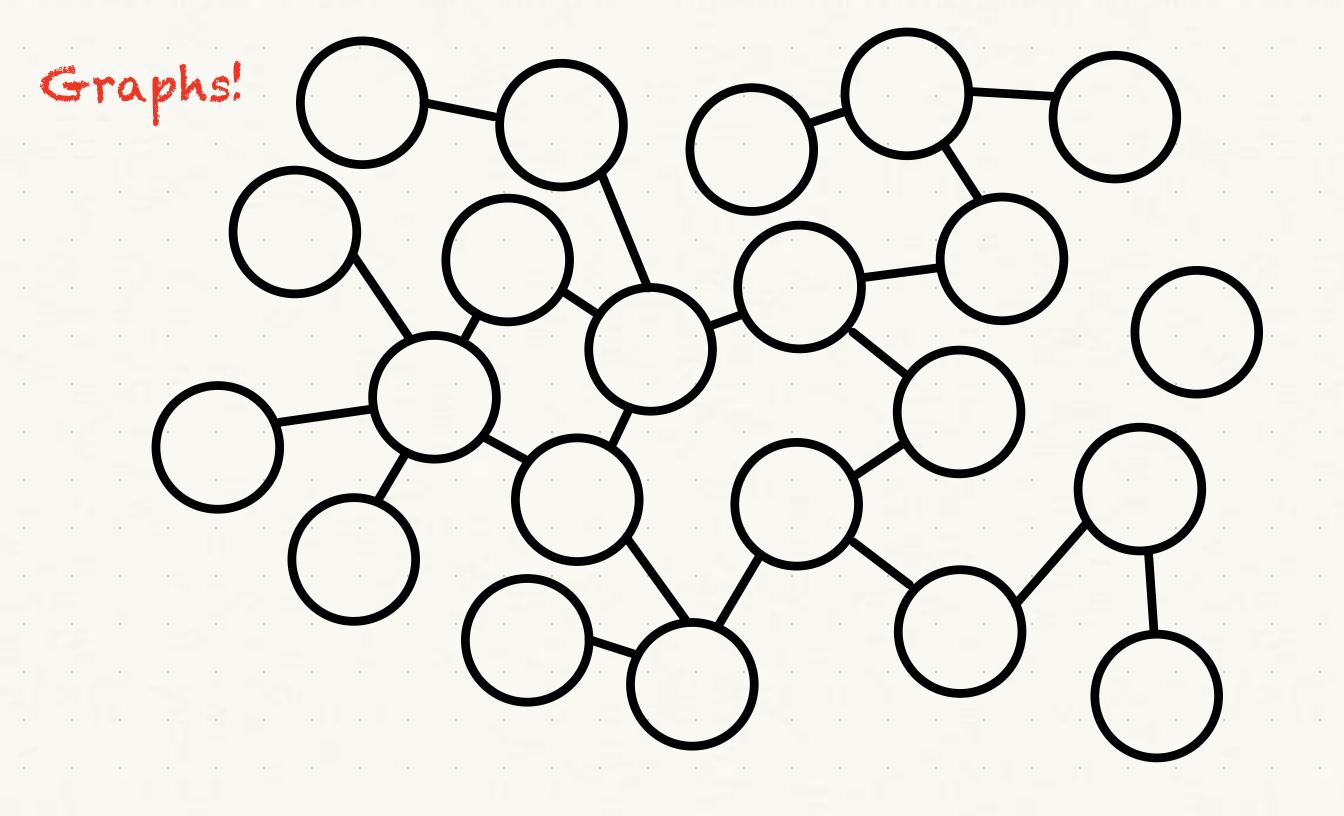
Understanding API Dispatching in NetworkX

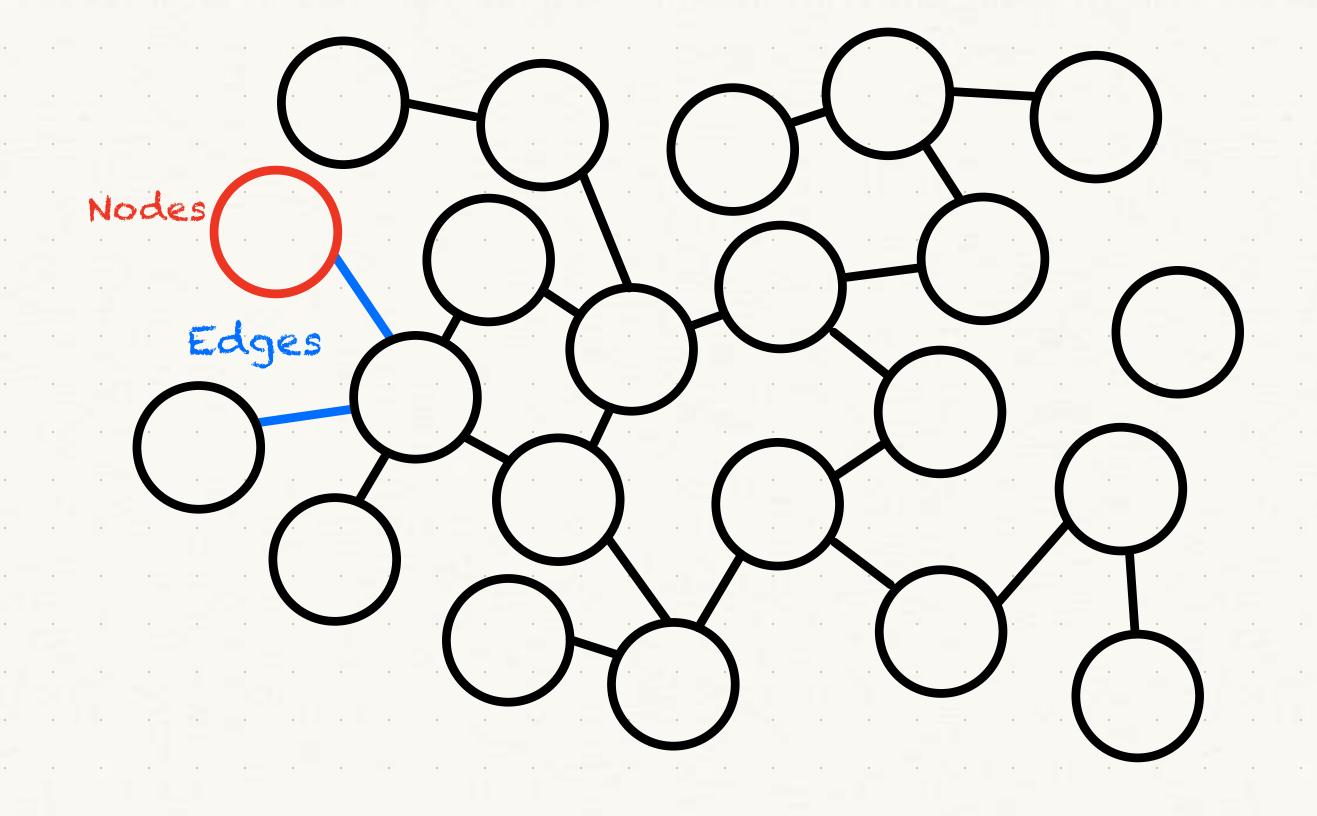


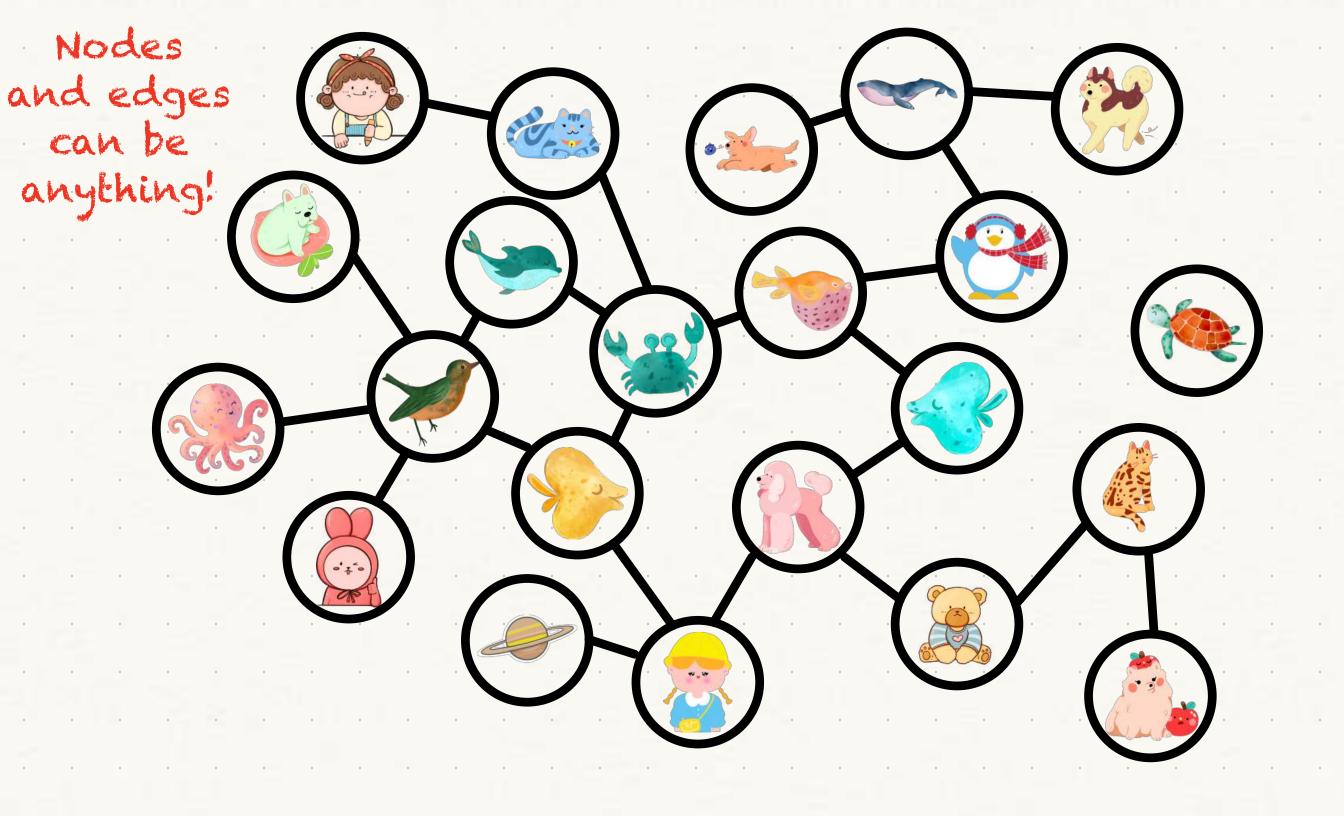
NetworkX is a **graph** (aka network) analysis
Python library

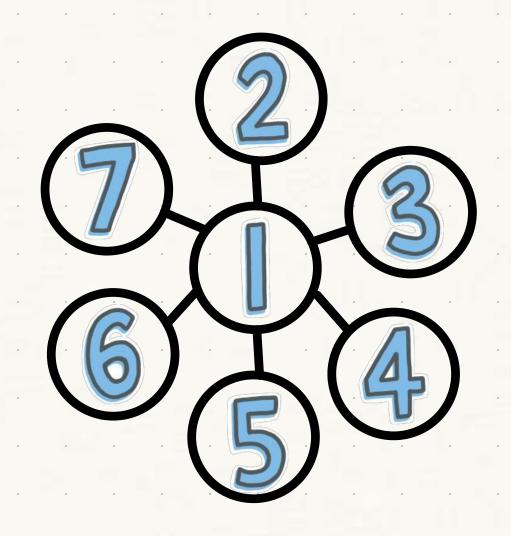


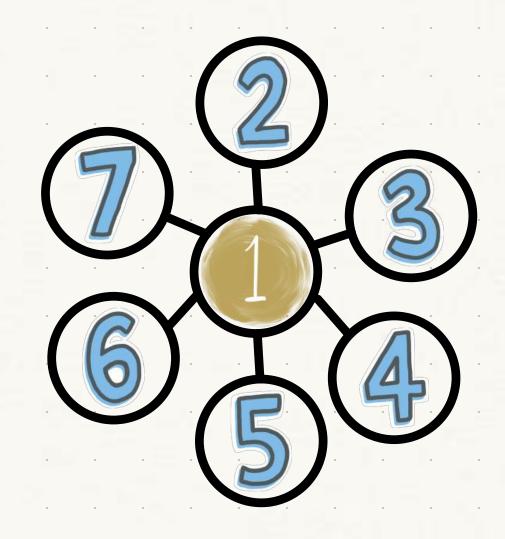
NOT these kind of graphs!

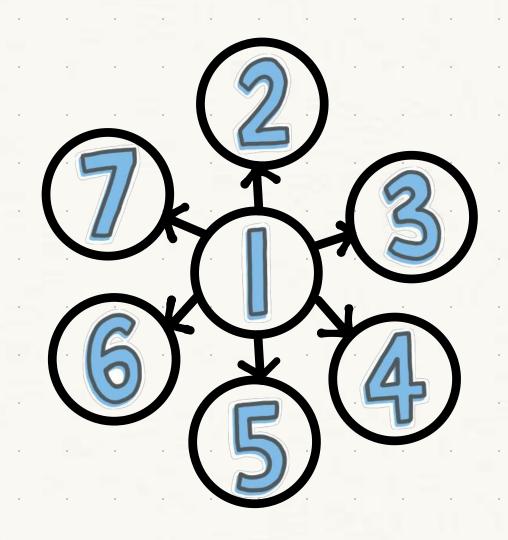


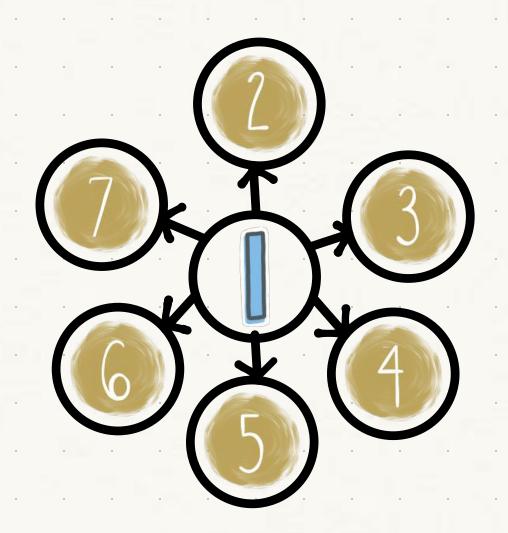


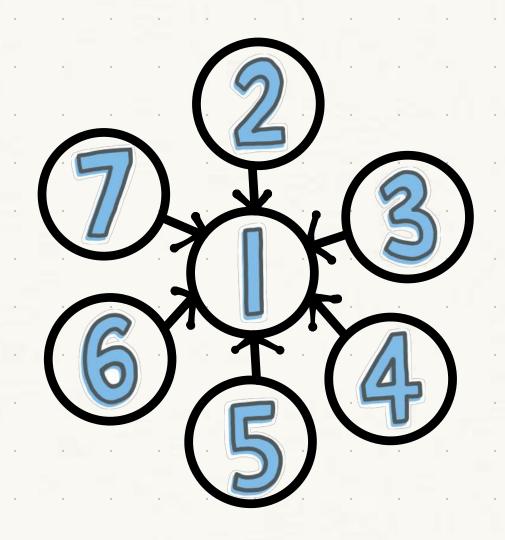


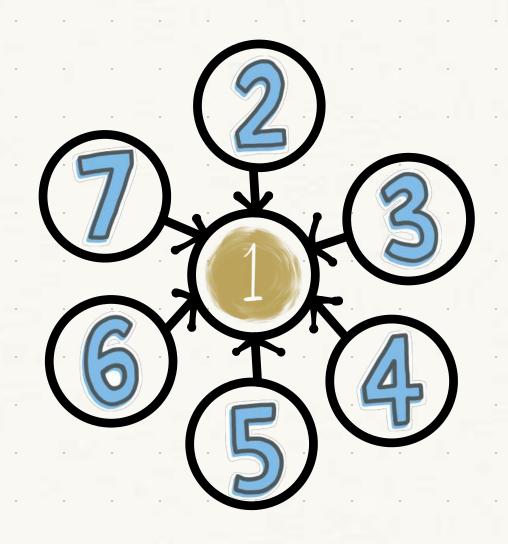












But, what is this "importance"?

betweenness_centrality

pagerank

degree

closeness_centrality

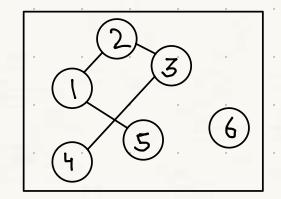
eigenvector_centrality

square_clustering

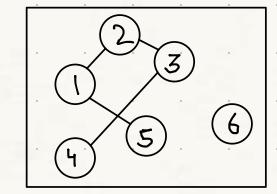
But, graph analysis is more than this...



```
>>> import networkx as nx
>>> G = nx.Graph()
>>> G.add_edges_from([(1,2), (3,4), (2,3), (5,1)])
>>> G.add_node(6)
>>> nx.betweenness_centrality(G)
```



```
>>> import networkx as nx
>>> G = nx.Graph()
>>> G.add_edges_from([(1,2), (3,4), (2,3),
 (5,1)])
>>> G.add_node(6)
>>> nx.betweenness_centrality(G)
{1: 0.3000000000000004, 2: 0.4, 3:
 0.300000000000004, 4: 0.0, 5: 0.0, 6: 0.0}
```



Lets try this...



```
>>> import networkx as nx
>>> G = nx.Graph()
>>> G.add_edges_from([(1,2), (3,4), (2,3),
 (5,1)])
>>> G.add_node(6)
>>> nx.betweenness_centrality(G)
>>> G = nx.fast_gnp_random_graph(1000000, 0.5)
>>> nx.betweenness_centrality(G)
```

```
>>> import networkx as nx
>>> G = nx.Graph()
>>> G.add_edges_from([(1,2), (3,4), (2,3),
 (5,1)])
>>> G.add_node(6)
>>> nx.betweenness_centrality(G)
>>> G = nx.fast_gnp_random_graph(1000000, 0.5)
>>> nx.betweenness_centrality(G)
 ...takes forever 😩 😩 😩 😭 😭
```

```
>>> import networkx as nx
>>> G = nx.Graph()
>>> G.add_edges_from([(1,2), (3,4), (2,3),
 (5,1)])
>>> G.add_node(6)
>>> nx.betweenness_centrality(G)
>>> G = nx.fast_gnp_random_graph(1000000, 0.5)
>>> nx.betweenness_centrality(G)
 ...takes forever 答 答 答 😭 😭
```





Understanding API Dispatching in NetworkX



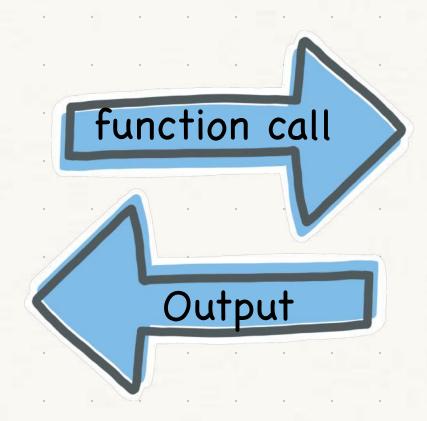
Understanding API Dispatching in NetworkX



What is Dispatching?

Usually...



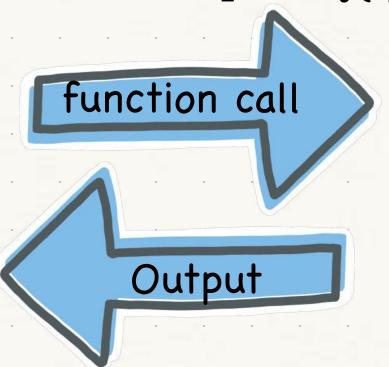


The main Library

Usually...



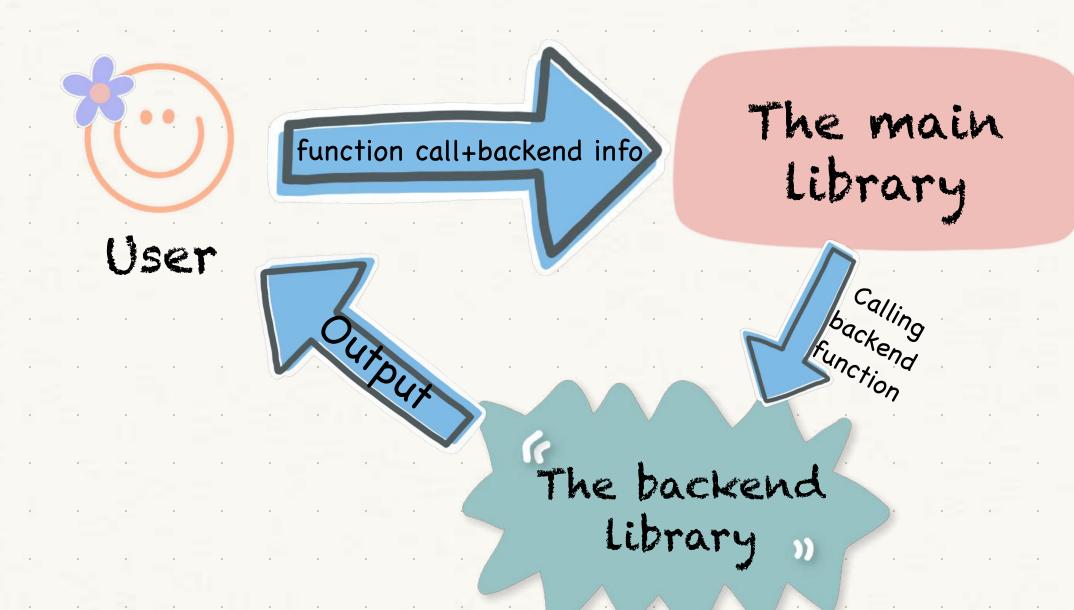
nx.betweenness_centrality(G)



The main Library

{1: 0.3000000000000004, 2: 0.4, 3:

0.3000000000000004, 4: 0.0, 5: 0.0, 6: 0.0}





function call+backend info

The main library

Calling backend Function

8

Here, backend info can be inside the function call or stored globally as an environment variable, or in the Graph object itself. The backend library

Dispatching in NetworkX

nx-parallel

def betweenness_centrality(G, ...):

nx.betweenness_centrality(G, backend="parallel")-

nx.betweenness_centrality(cug)

with nx.config(backend_priority=[graphblas,]):
nx.betweenness_centrality(G) = = = =

NetworkX

@_dispatchable 🛥🗗

def betweenness_centrality(G, ...):

nx-cugraph

def betweenness_centrality(G, ...):

python-graphblas

def betweenness_centrality(G, ...):

Dispatching in NetworkX

nx-parallel

def betweenness_centrality(G, ...):

nx.betweenness_centrality(G, backend="parallel")-

NetworkX

def betweenness_centrality(G, ...):

@_dispatchable 🗕 🗗

nx-cugraph

def betweenness_centrality(G, ...):

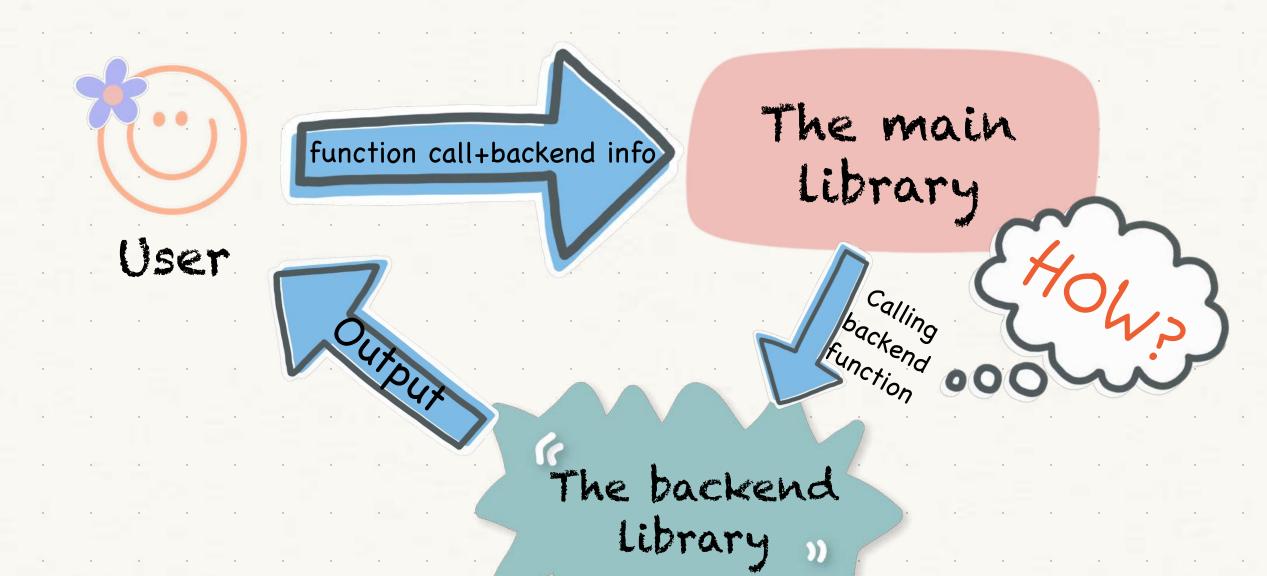
nx.betweenness_centrality(cug) -

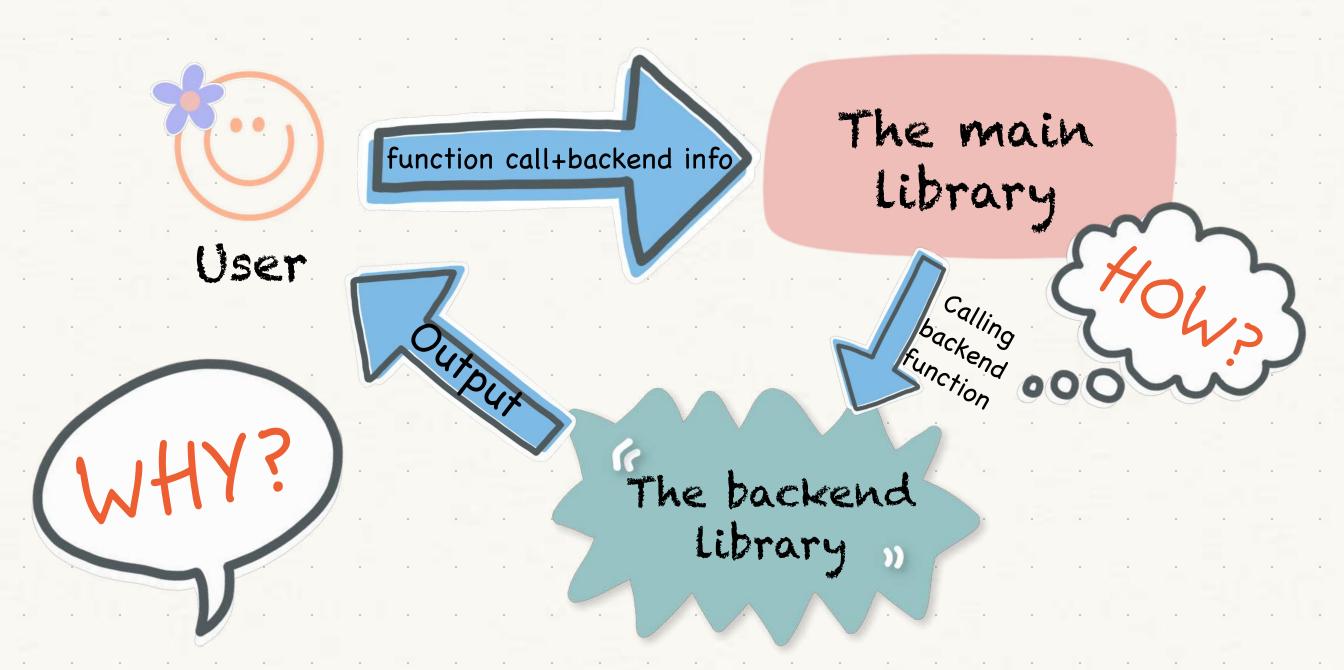
with nx.config(backend_priority=[graphblas,]):
nx.betweenness_centrality(G) - - - -

in an attribute of the backend graph object (type based dispatching).

python-graphblas

def betweenness_centrality(G, ...):

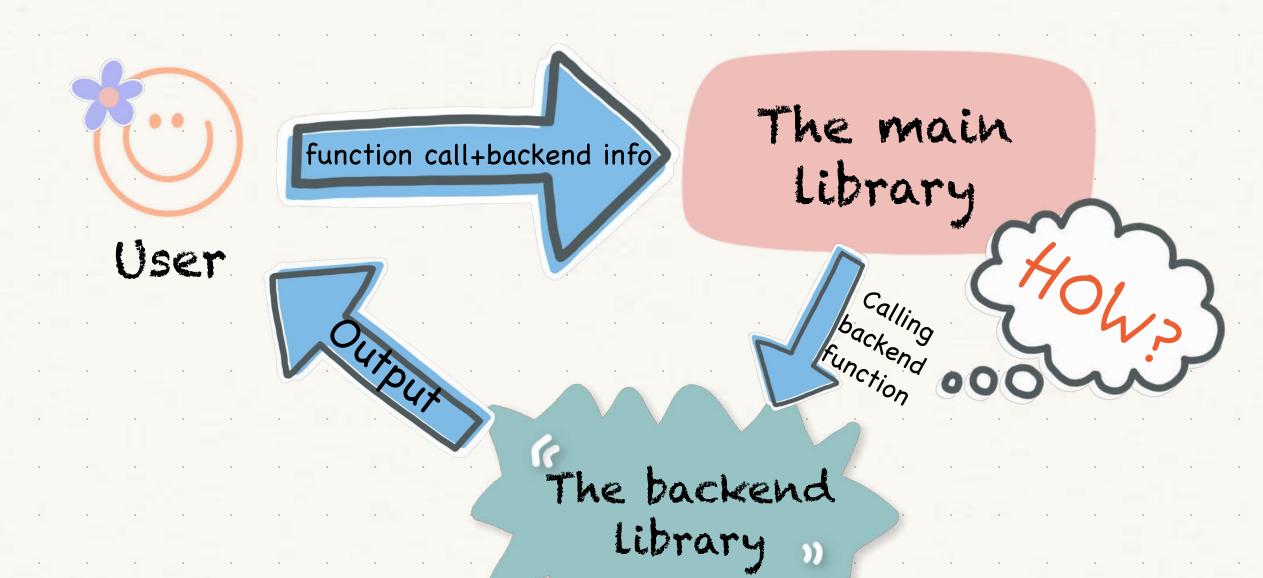




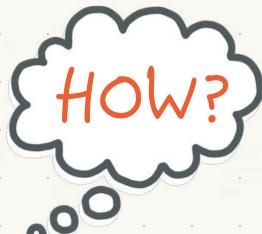




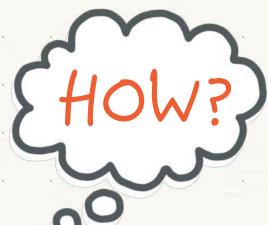
- NetworkX is a big and old project...
 - Pure Python + no dependencies -> "first-time-contributor"-friendly
 - Consistency
 - Maintenance
- No major changes to existing codebase
- Faster code with the same user-API
- Will this API dispatching be good for your project? IDK...







```
if backend=="parallel":
    # import nx_parallel
    # load and call func in nx_parallel
elif backend=="graphblas":
    # import graphblas
    # load and call func in graphblas
...
...
else:
    # backend not found!
```



if backend=="parallel":
 # import nx_parallel
 # load and call func in nx_parallel
elif backend=="graphblas":
 # import graphblas
 # load and call func in graphblas
...
...
else:
 # backend not found!

not a very scalable solution!

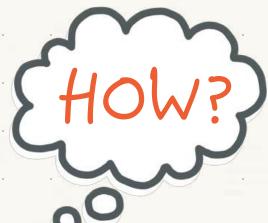
not very flexible!

NetworkX decides

backend's existence

populates the frontend library with a lot of backends' info

What about private/
experimental
backends?



```
if backend=="parallel":

# import nx_parallel

# load and call func in nx_parallel
elif backend=="graphblas":

# import graphblas

# load and call func in graphblas

...

...

else:

# backend not found!
```

not a very scalable solution!

not very flexible!

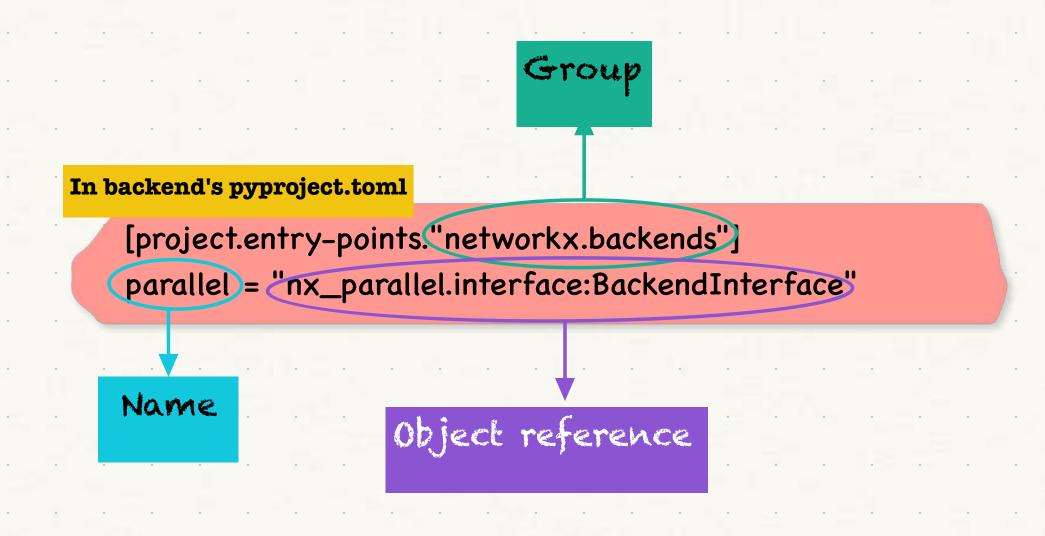
NetworkX decides

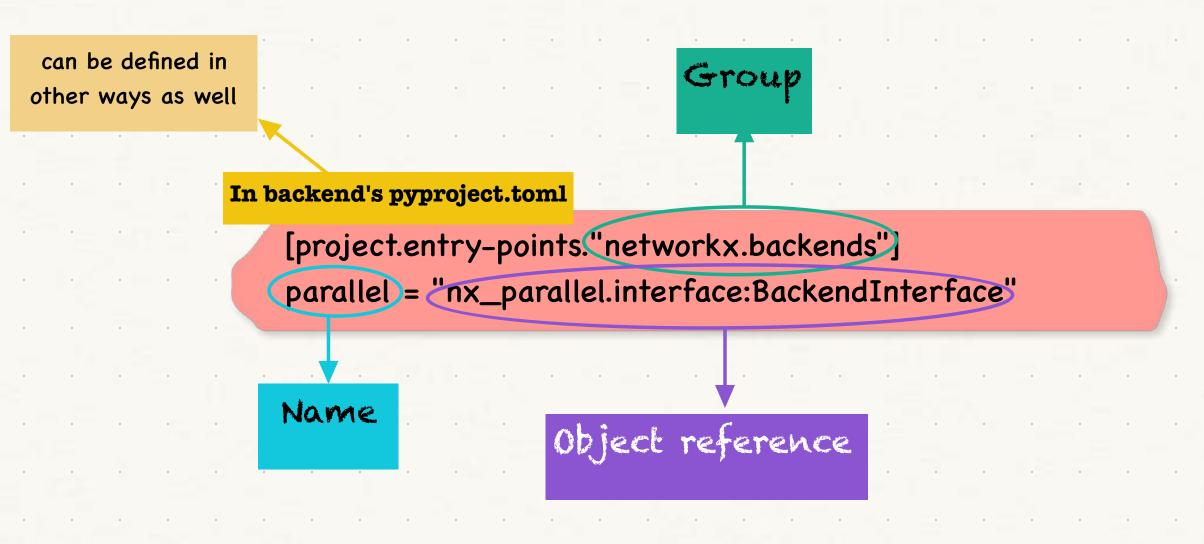
backend's existence

Better way : entry-points

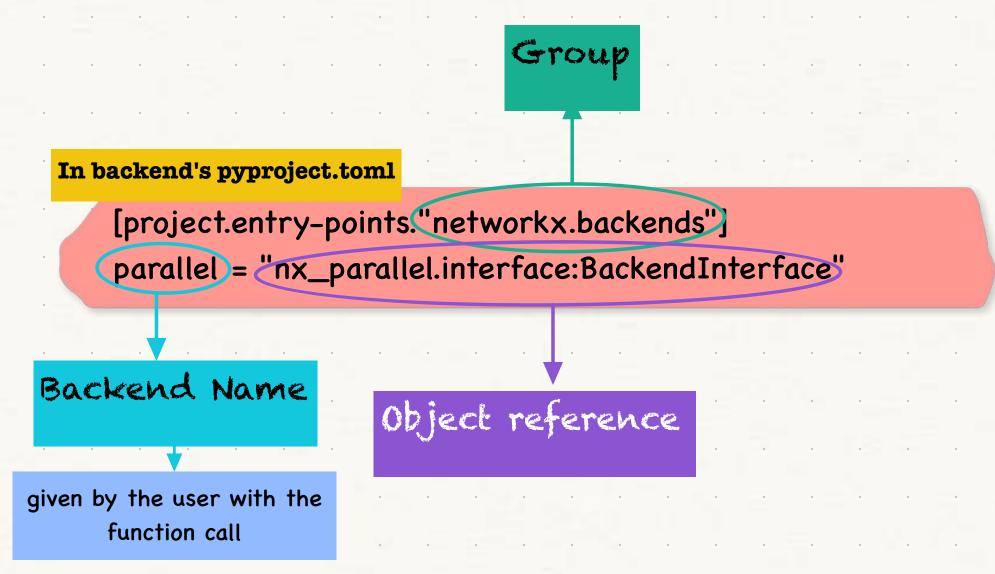
populates the frontend library with a lot of backends' info

What about private/
experimental
backends?

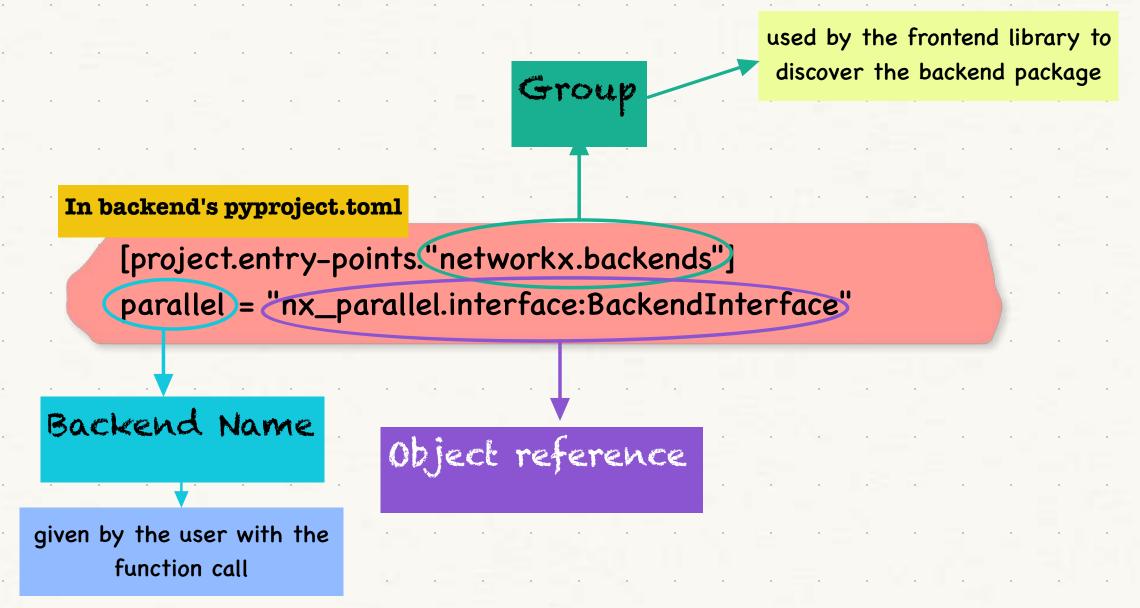




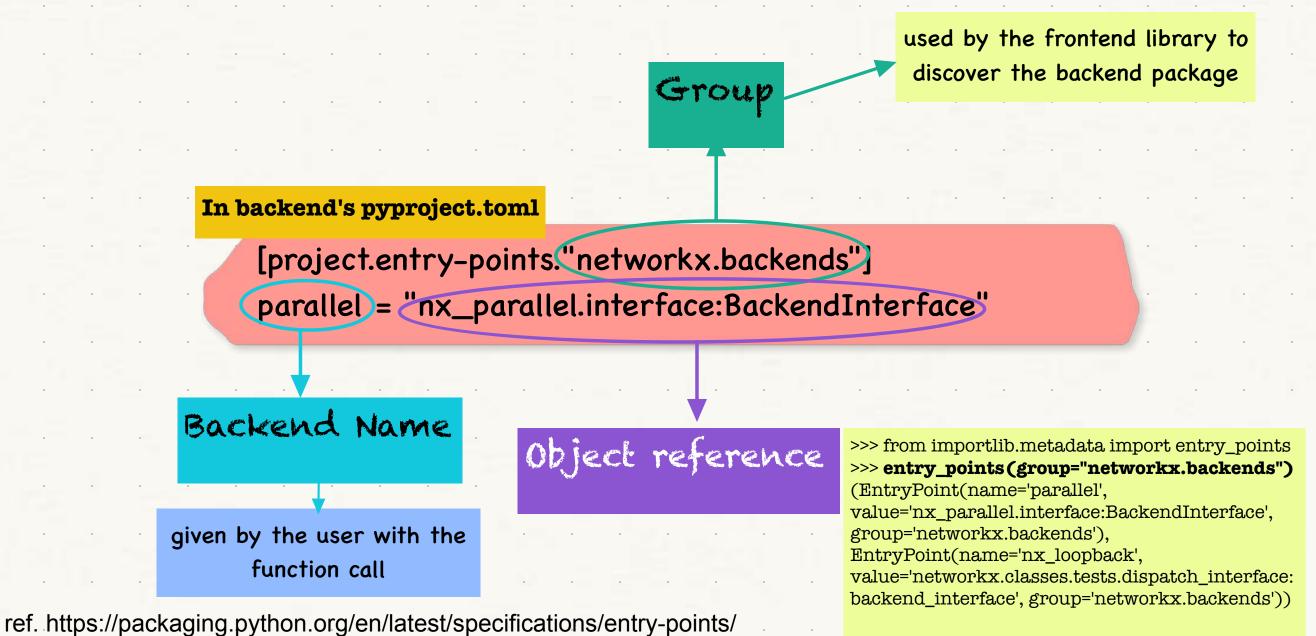
ref. https://packaging.python.org/en/latest/specifications/entry-points/

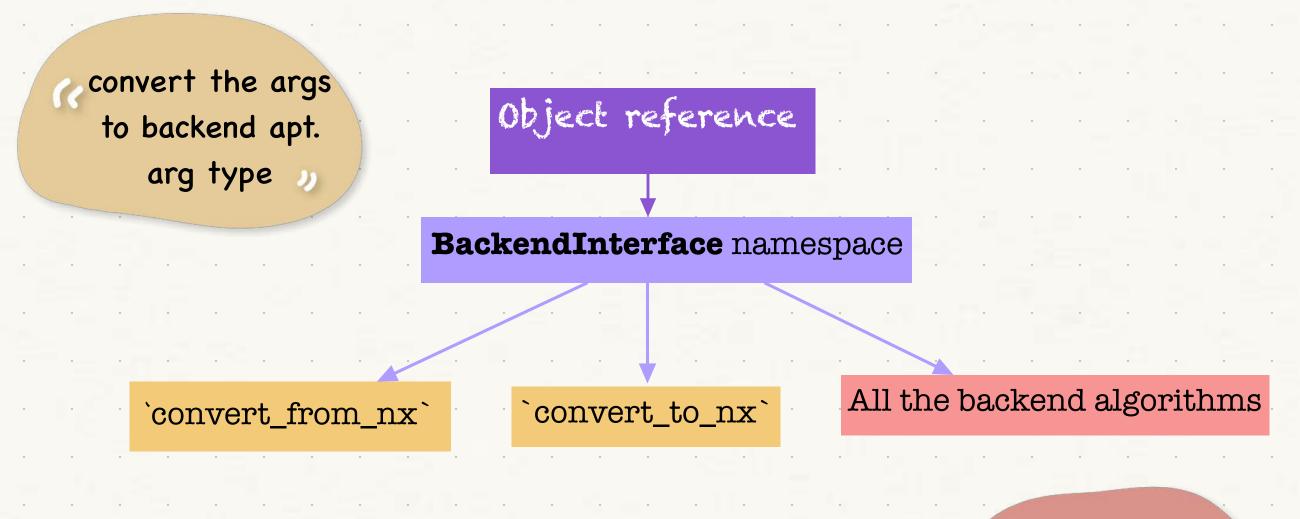


ref. https://packaging.python.org/en/latest/specifications/entry-points/



ref. https://packaging.python.org/en/latest/specifications/entry-points/





find the function in the backend

So, to summarise...

- 1. User calls the function along the "backend name"
- 2. Backend discovery using entry-points
 - A. frontend library gets all the packages in 'networkx.backends' group
 - B. find the backend with the given backend name
- 3. Load the 'BackendInterface' namespace of the backend
 - A. check if the backend has the called function
 - B. convert the arguments using `convert_from_nx`
- 4. Call the backend function with the converted args n return the result!

Some more dispatching-related stuff:

logging

Automatic testing

NETWORKX_TEST_BACKEND=parallel NETWORKX_FALLBACK_TO_NX=True pytest --pyargs networkx

Configurations

nx.config.backends.parallel.n_jobs = 8
nx.config.backends.parallel.verbose = 10

Additional backend args

>>> nx.betweenness_centrality(G, backend="parallel", get_chunks=get_chunks)

2nd entry point

[project.entrypoints."networkx.backend_info"] parallel = "_nx_parallel:get_info"

 $source-https://networkx.org/documentation/stable/reference/algorithms/generated/networkx.algorithms.shortest_paths.weighted.all_pairs_bellman_ford_path_length.html$

fallback option

Additional backends implement this function

cugraph: GPU-accelerated backend.

Negative cycles are not yet supported. NotImplementedError will be raised if there are negative edge weights. We plan to support negative edge weights soon. Also, callable weight argument is not supported.

Additional parameters:

dtype: dtype or None, optional

The data type (np.float32, np.float64, or None) to use for the edge weights in the algorithm. If None, then dtype is determined by the edge reduce.

graphblas: OpenMP-enabled sparse linear algebra backend.

Additional parameters:

chunksize : int or str, optional

Split the computation into chunks; may specify size as string or number of rows. Default "10 MiB"

parallel: Parallel backend for NetworkX algorithms

The parallel implementation first divides the nodes into chunks and then creates a generator to lazily compute shortest paths lengths for each node in node_chunk, and then employs joblib's Parallel function to execute these

Caching conversion

Is dispatching just to run algorithms faster?

Is dispatching just to run algorithms faster?

No...

- database backend: nx-arangodb
- visualisation backend: ??
- more to come...

Dispatching and other projects

- scikit-image
- sklearn
- numpy
- ?

Dispatching and other projects

- scikit-image
- sklearn
- numpy
- ?

what are your challenges?

Dispatching discussions:

- Weekly dispatch meetings: https://scientific-python.org/calendars/networkx.ics
- Dispatching channel in Scientific Python discord: https://discord.com/invite/vur45CbwMz
- `spatch` repository: https://github.com/scientificpython/spatch
- SPEC-2 (API Dispatching) document: https://scientific-python.org/specs/spec-0002/
- Scientific Python discuss: https://discuss.scientific-python.org/t/spec-2-api-dispatch/173



