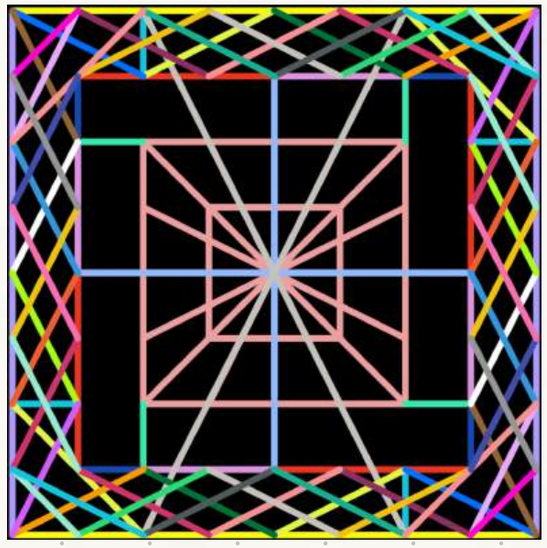
Understanding API Dispatching in NetworkX

About Me

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

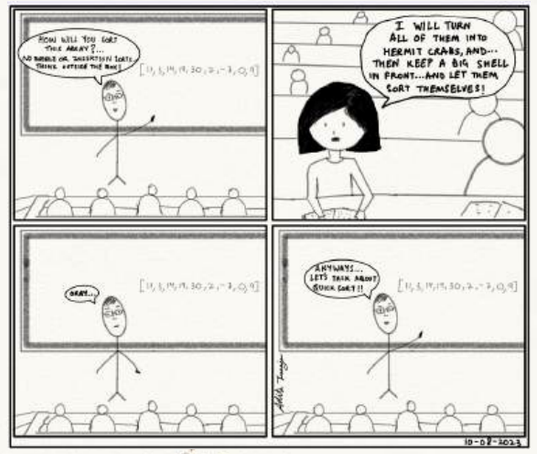


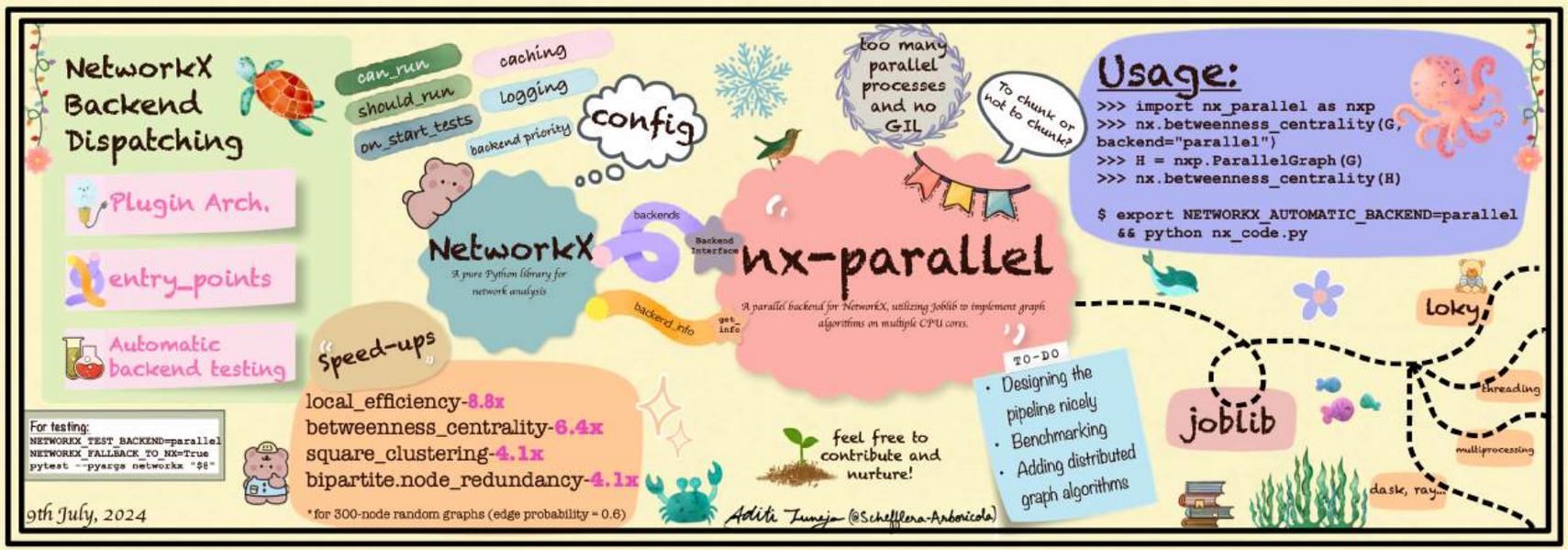
Slides - https://github.com/Schefflera-Arboricola/blogs/blob/main/archive

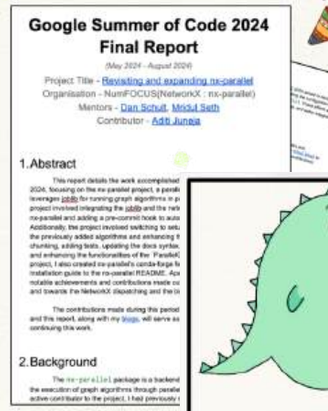
PyDelhi Meetups!

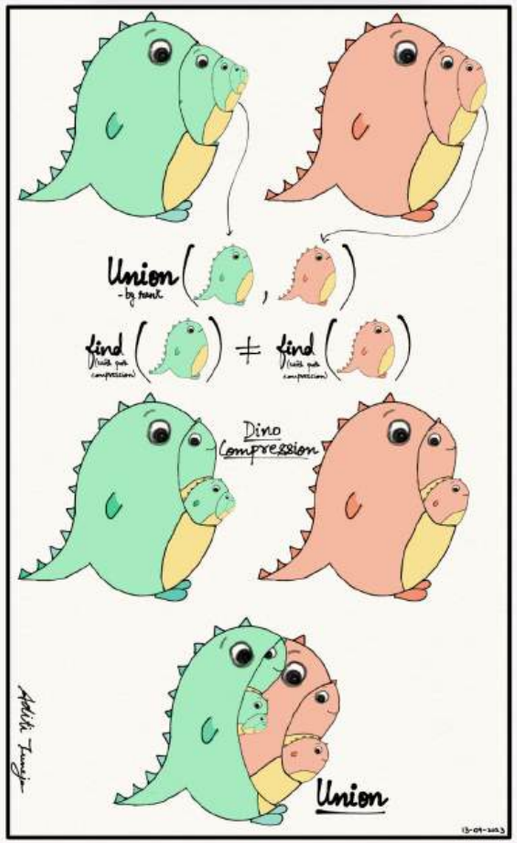
- In-person meetups conducted on 3rd Saturday of every month
- Submit your talk proposal here - https://github.com/pydelhi/talks/issues
- Find us on Telegram, WhatsApp, and know more at https://pydelhi.org/

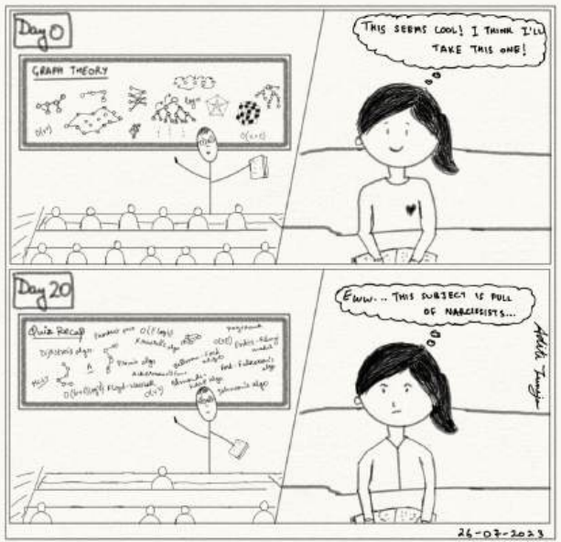
And...here is some of my stuff...

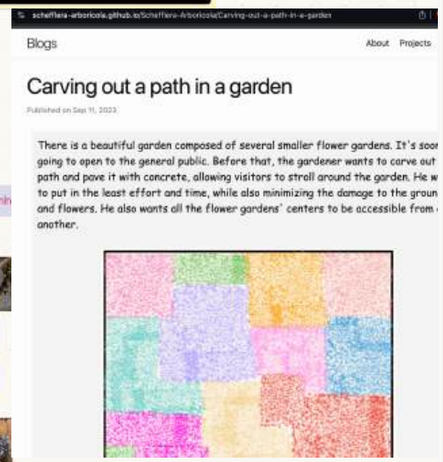












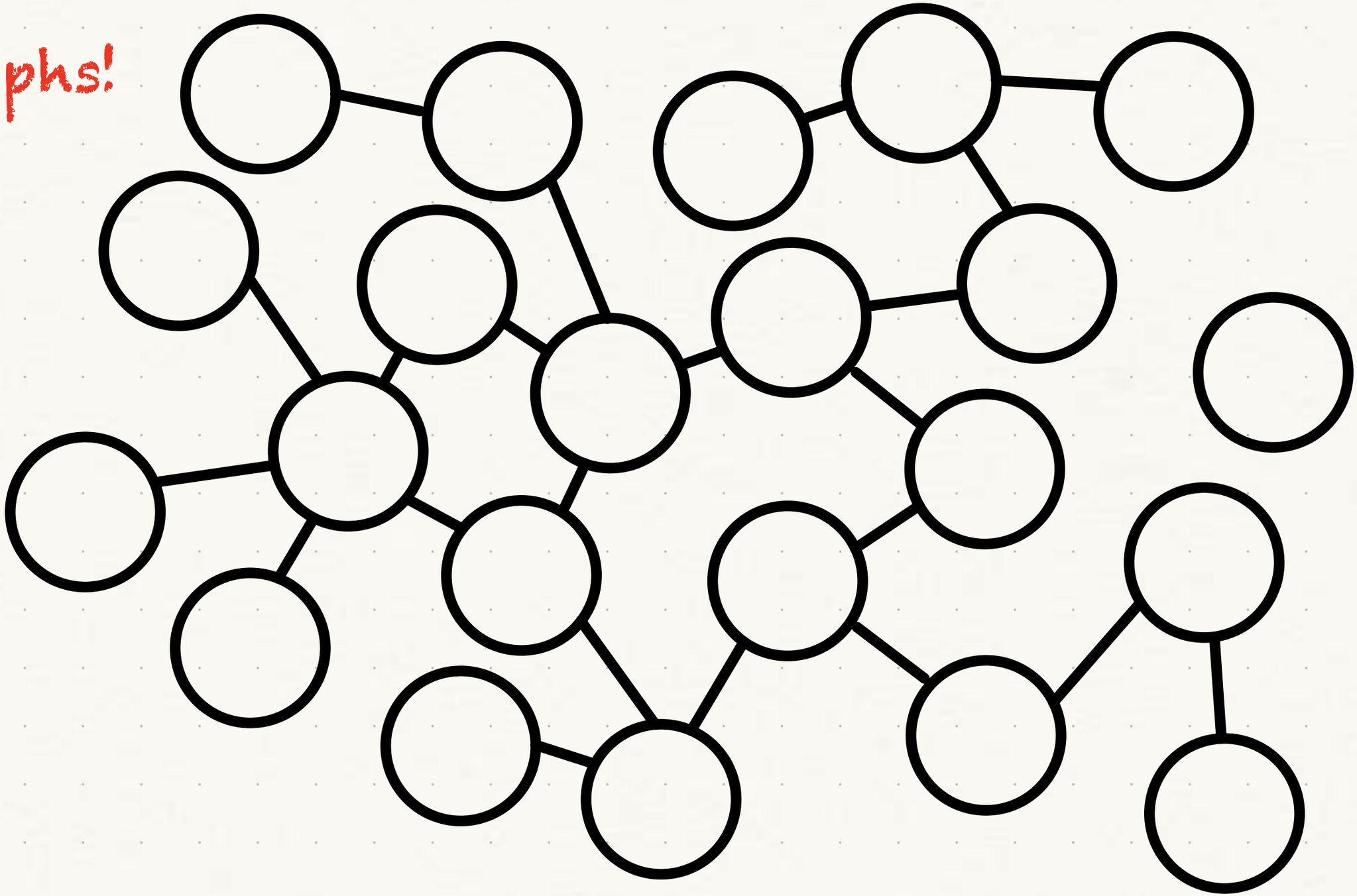
Understanding API Dispatching in NetworkX

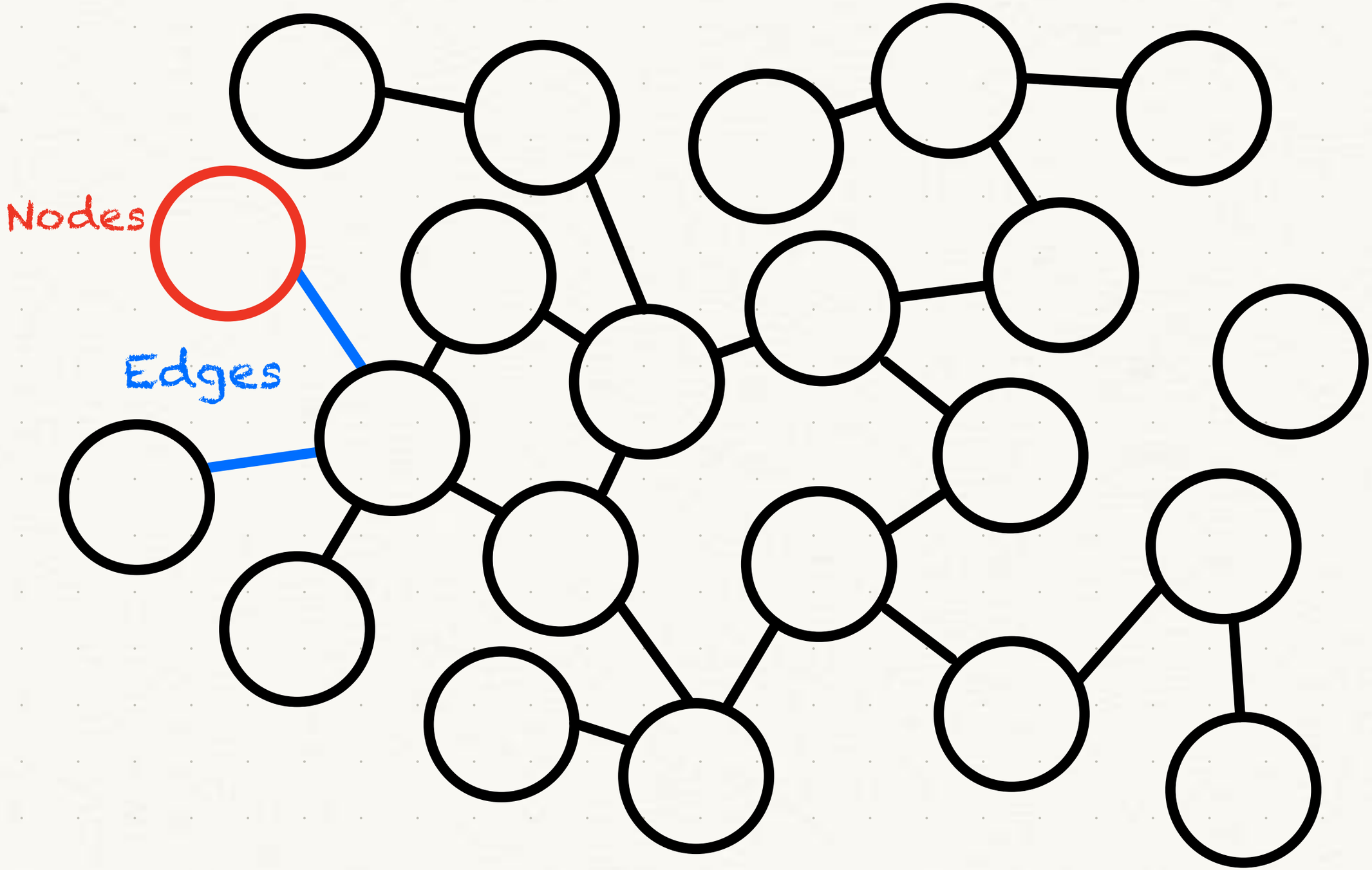
Understanding API Dispatching in NetworkX

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

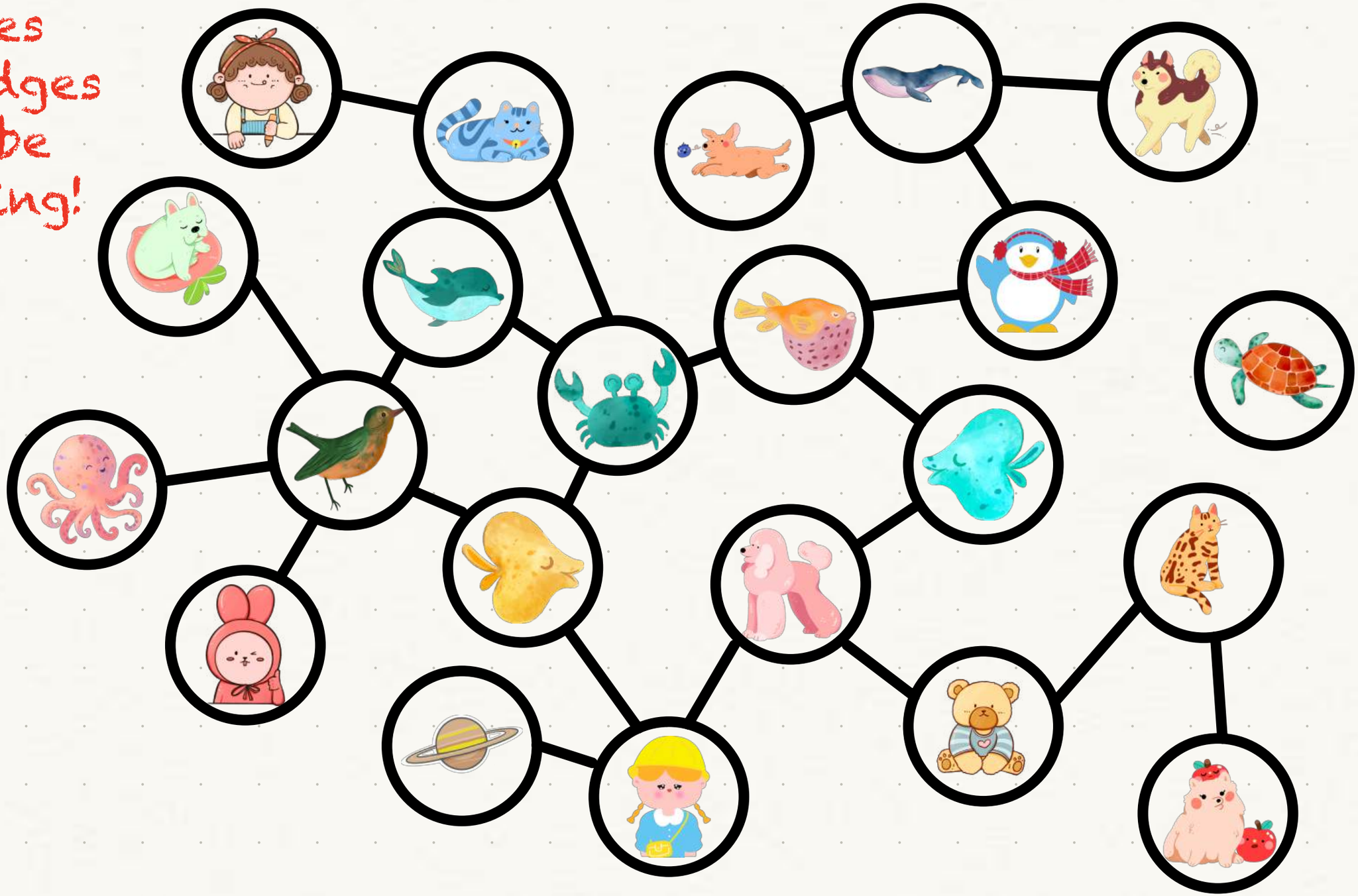
NOT these kind of graphs!

Graphs!

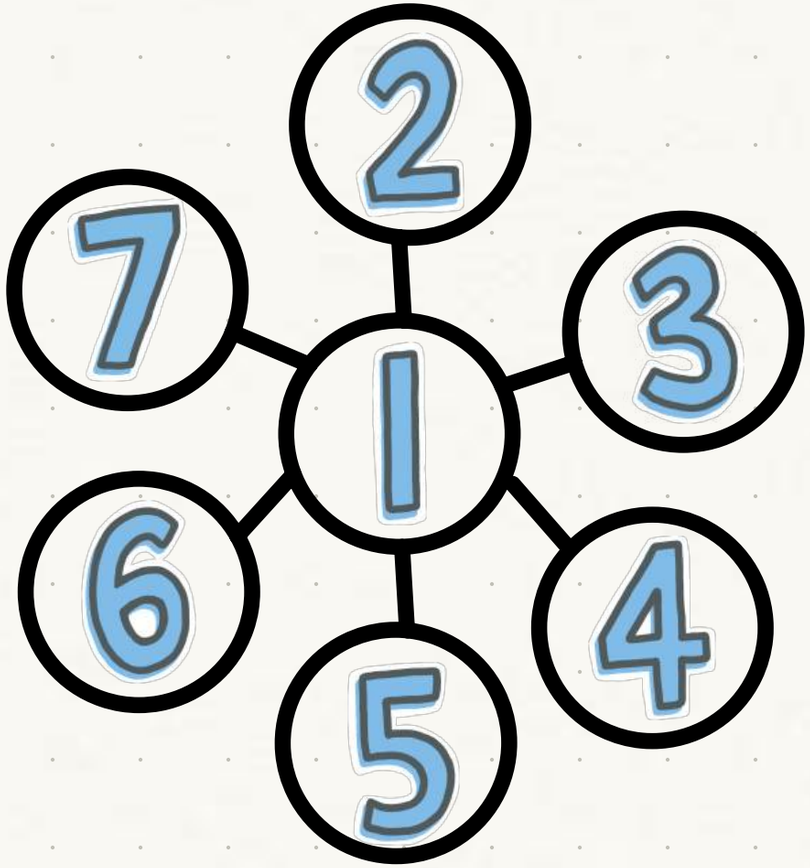



Nodes
Edges

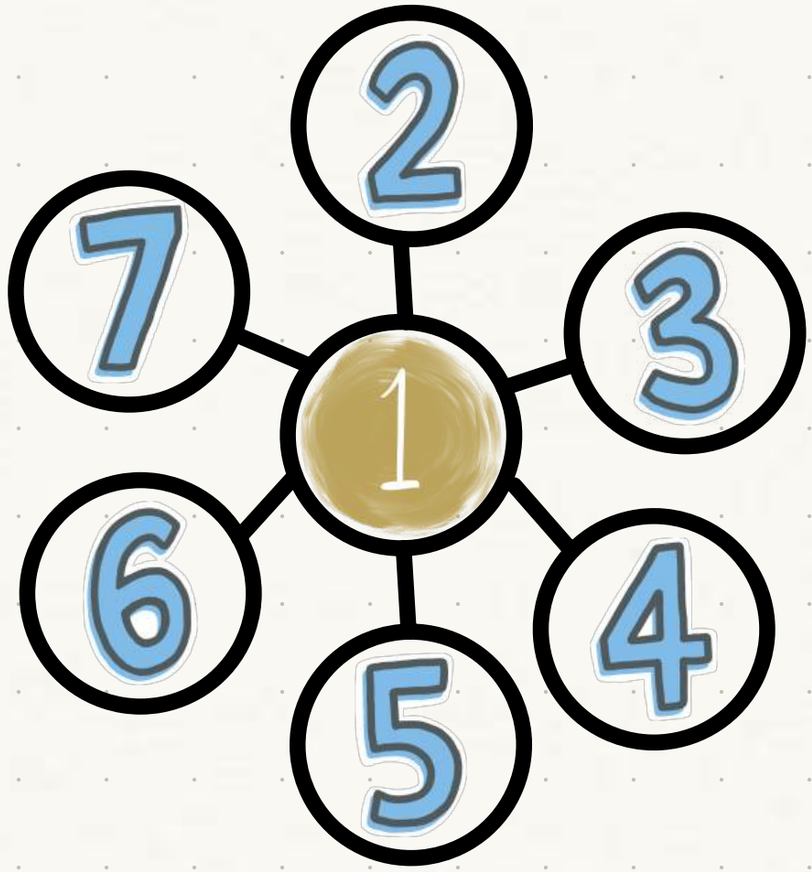
Nodes and edges can be anything!



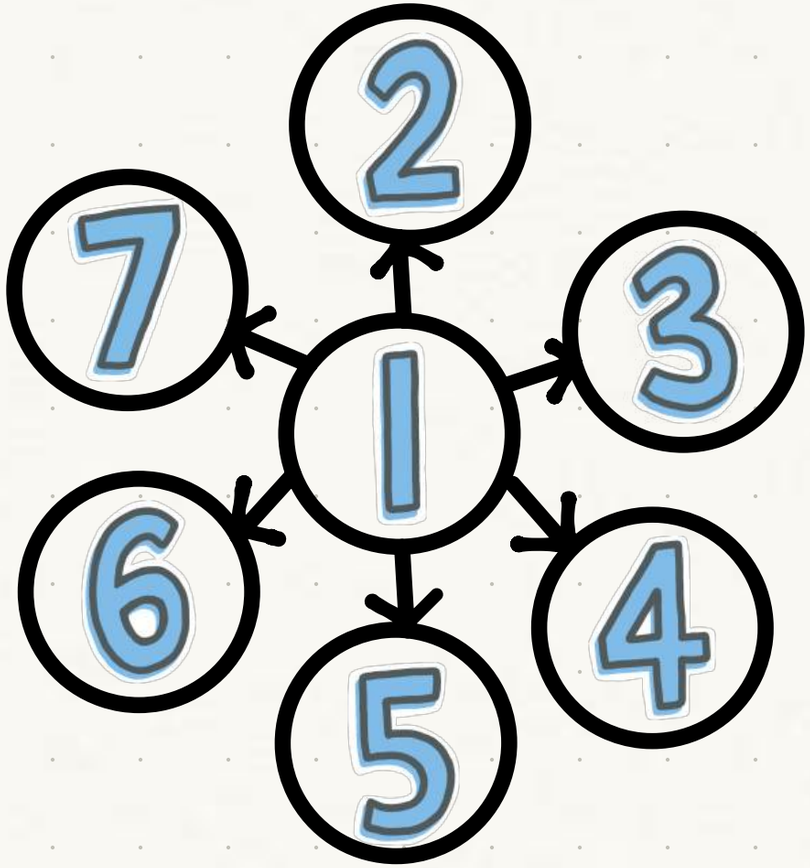
Which is the most important node?



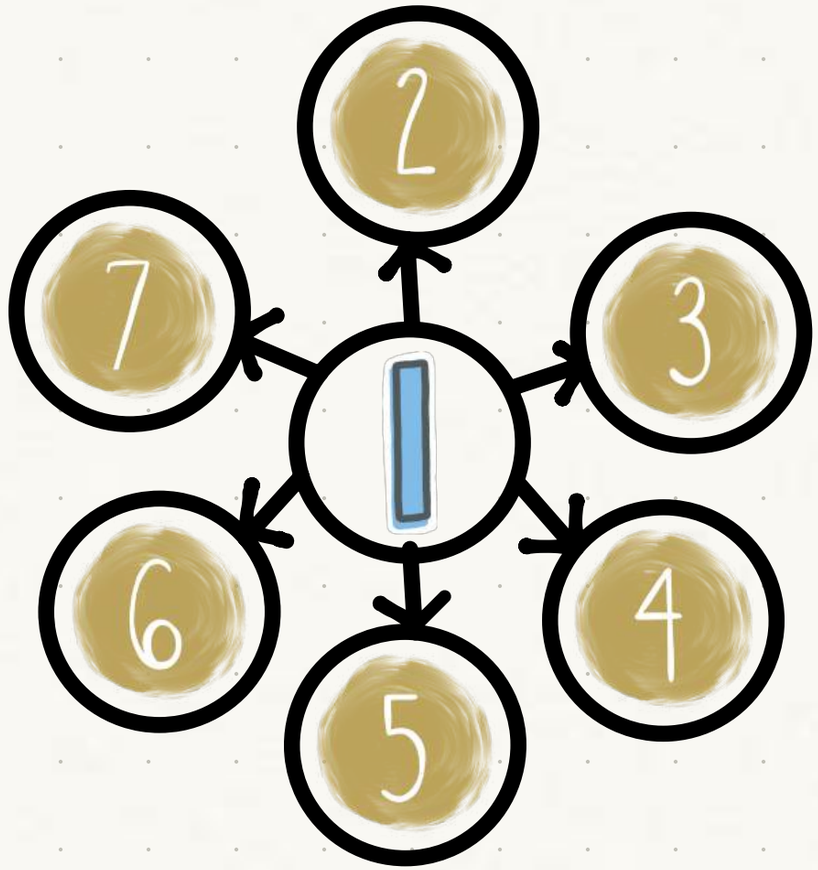
Which is the most important node?



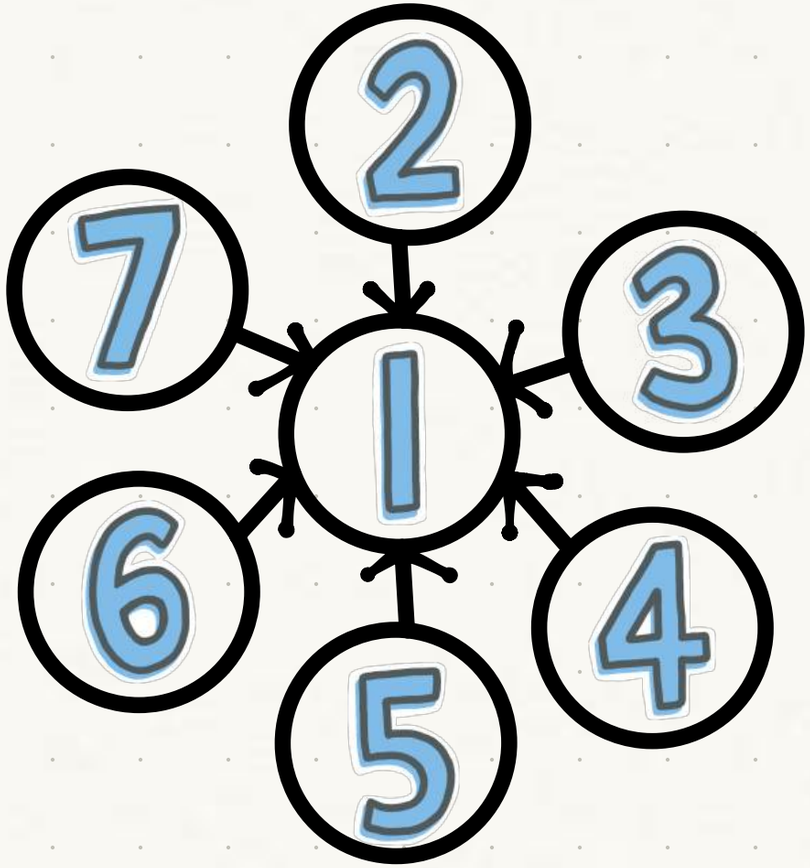
Which is the most important node?



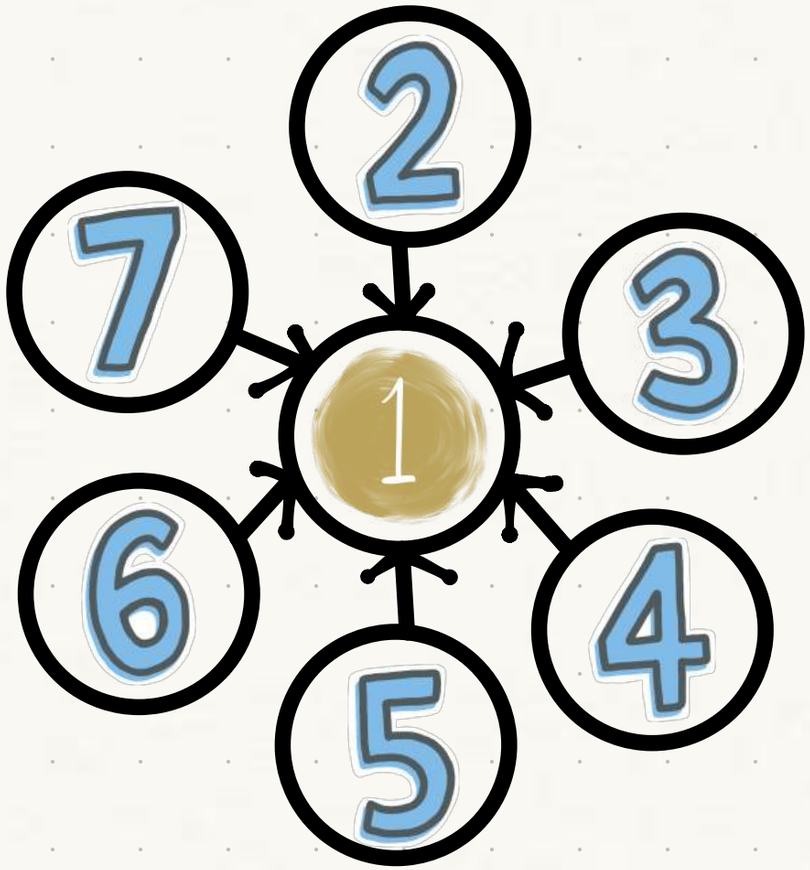
Which is the most important node?



Which is the most important node?



Which is the most important node?



But, what is this "importance"?

betweenness_centrality

pagerank

degree

closeness_centrality

eigenvector_centrality

square_clustering

But, graph analysis is more than this...

connectivity

traversals

triads

isomorphism

flow

communities

trees

cycles

density

shortest paths

matching

dependency

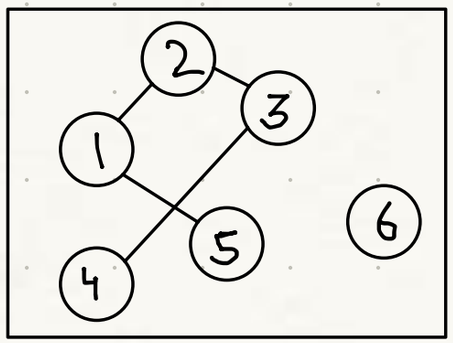
efficiency

link prediction

temporal graphs

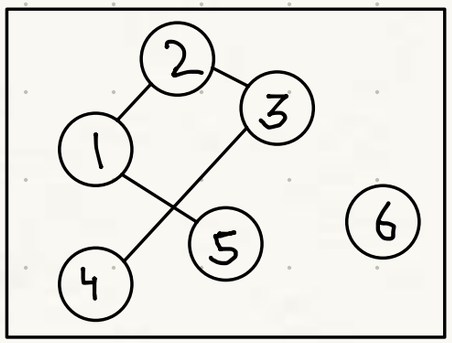
NetworkX API

```
>>> 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)
```



NetworkX API

```
>>> 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: 0.30000000000000004, 2: 0.4, 3:
0.30000000000000004, 4: 0.0, 5: 0.0, 6: 0.0}
```



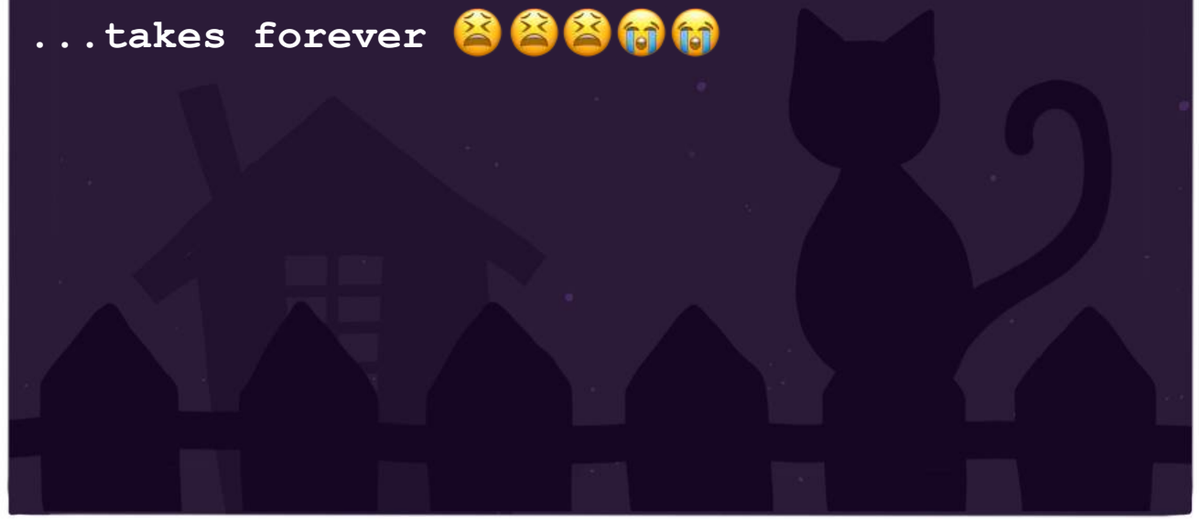
NetworkX API

```
>>> 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)
```

Lets try this...

NetworkX API

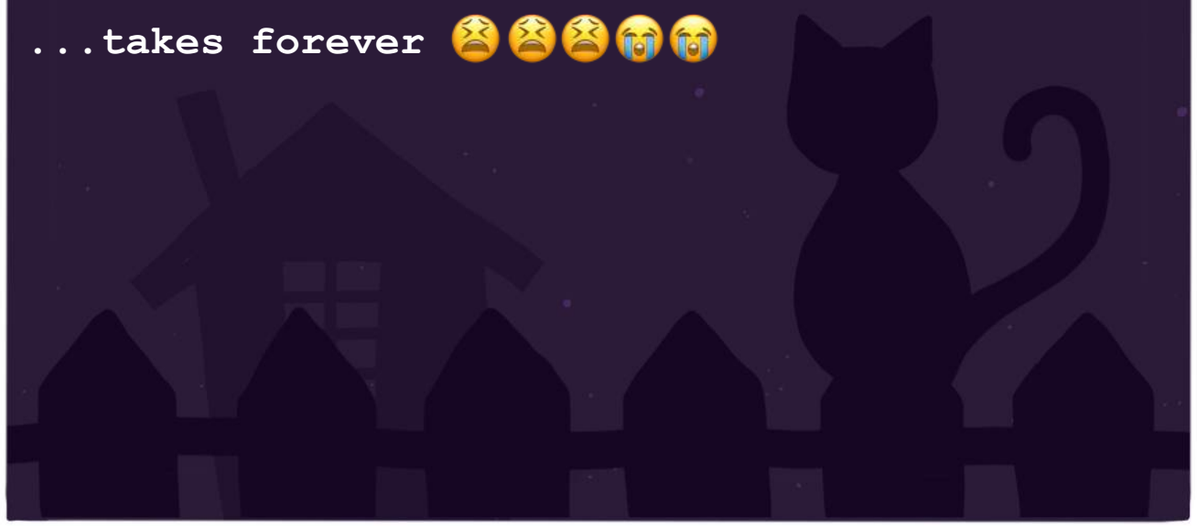
```
>>> import networkx as nx

>>> 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 😫😫😫😭😭
```



NetworkX API

```
>>> 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 😫😫😫😭😭

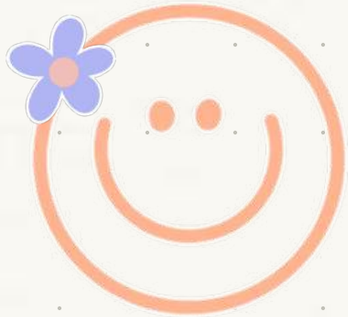


Understanding API Dispatching in NetworkX

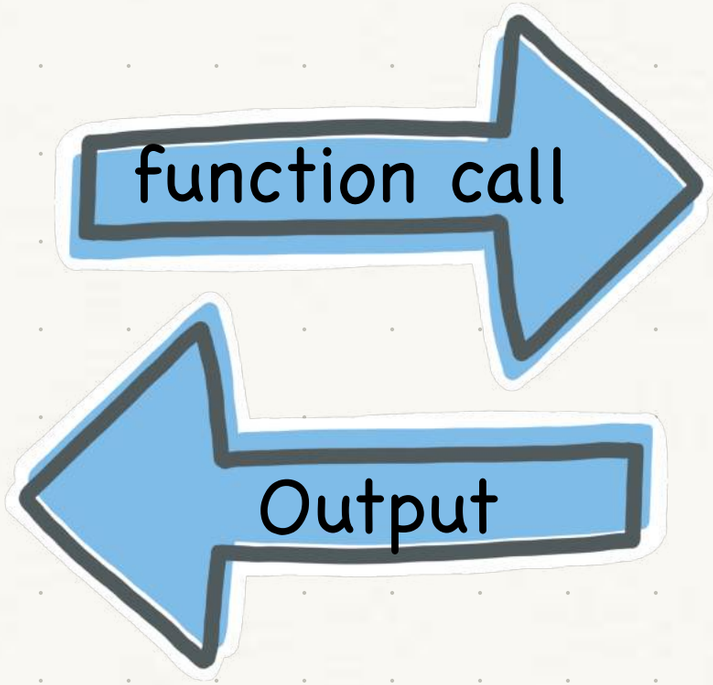
Understanding API Dispatching in NetworkX

"What is Dispatching?"

Usually...



User



The main library

Usually...

User

nx.betweenness_centrality(G)

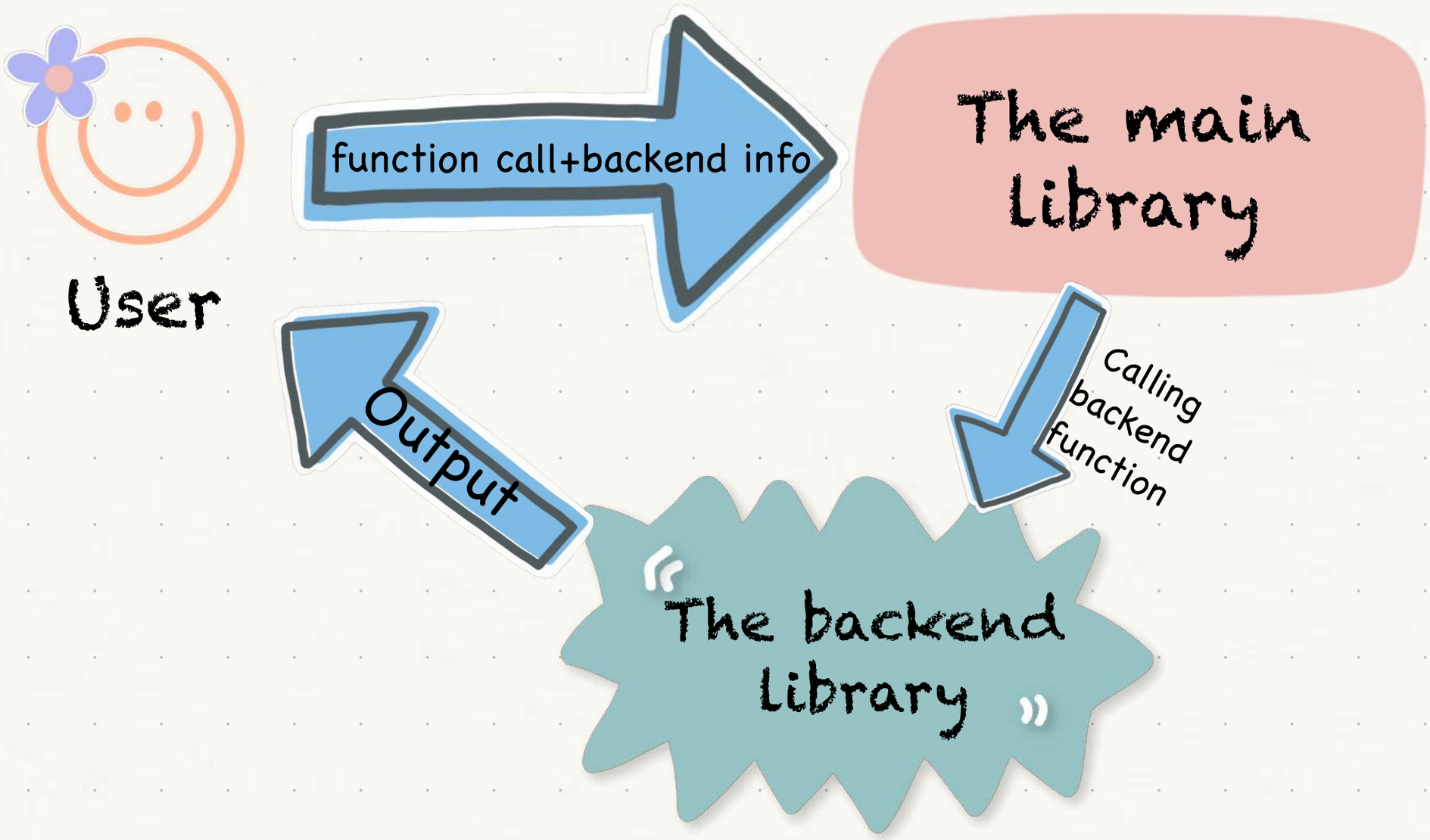
function call

Output

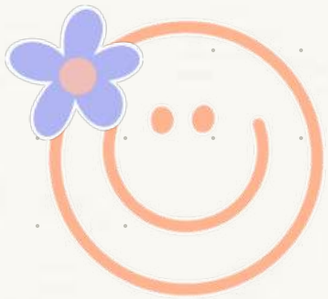
The main library

{1: 0.30000000000000004, 2: 0.4, 3: 0.30000000000000004, 4: 0.0, 5: 0.0, 6: 0.0}

Dispatching



Dispatching



User

function call+backend info

The main library

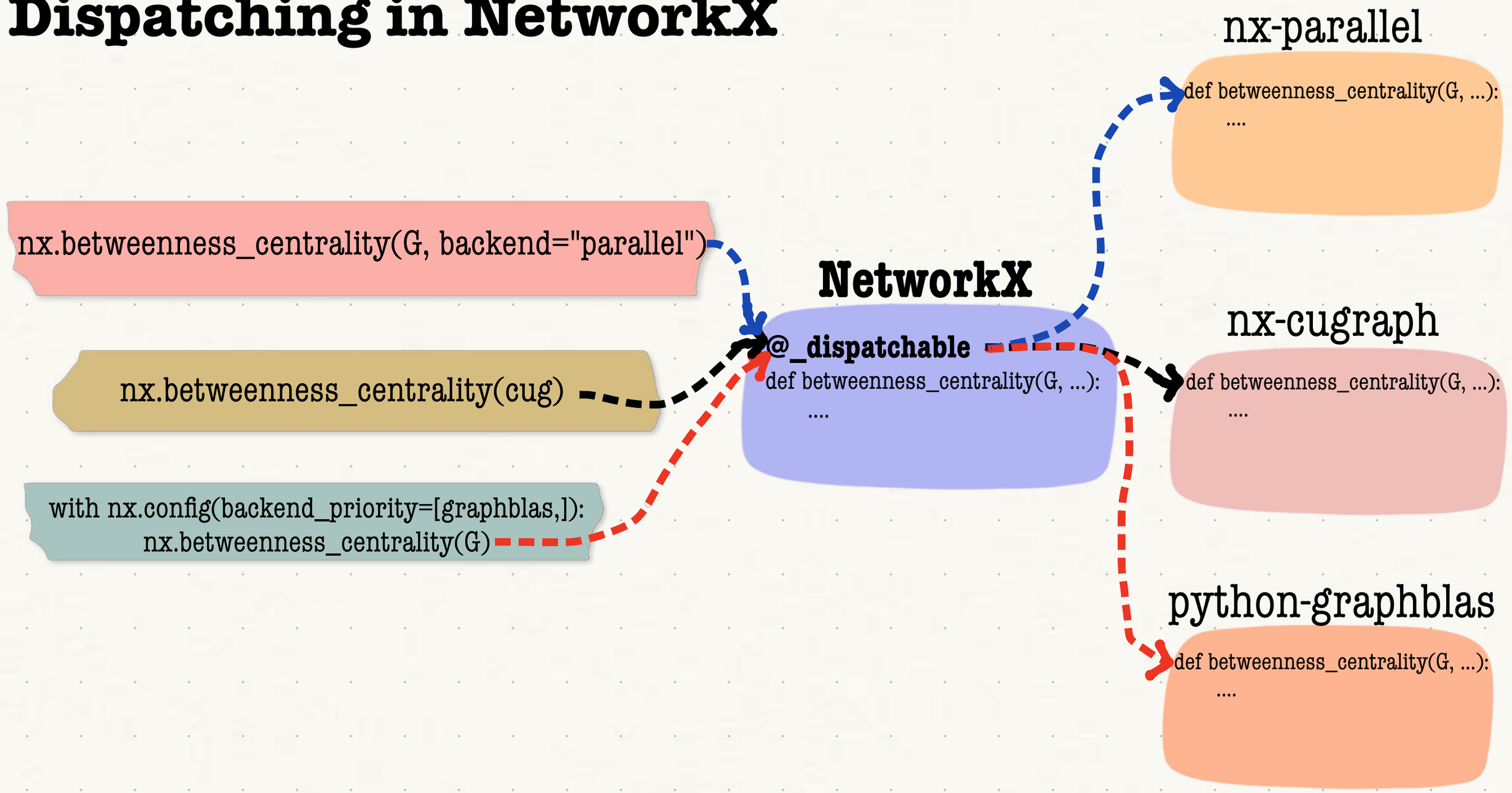
Calling backend function

Output

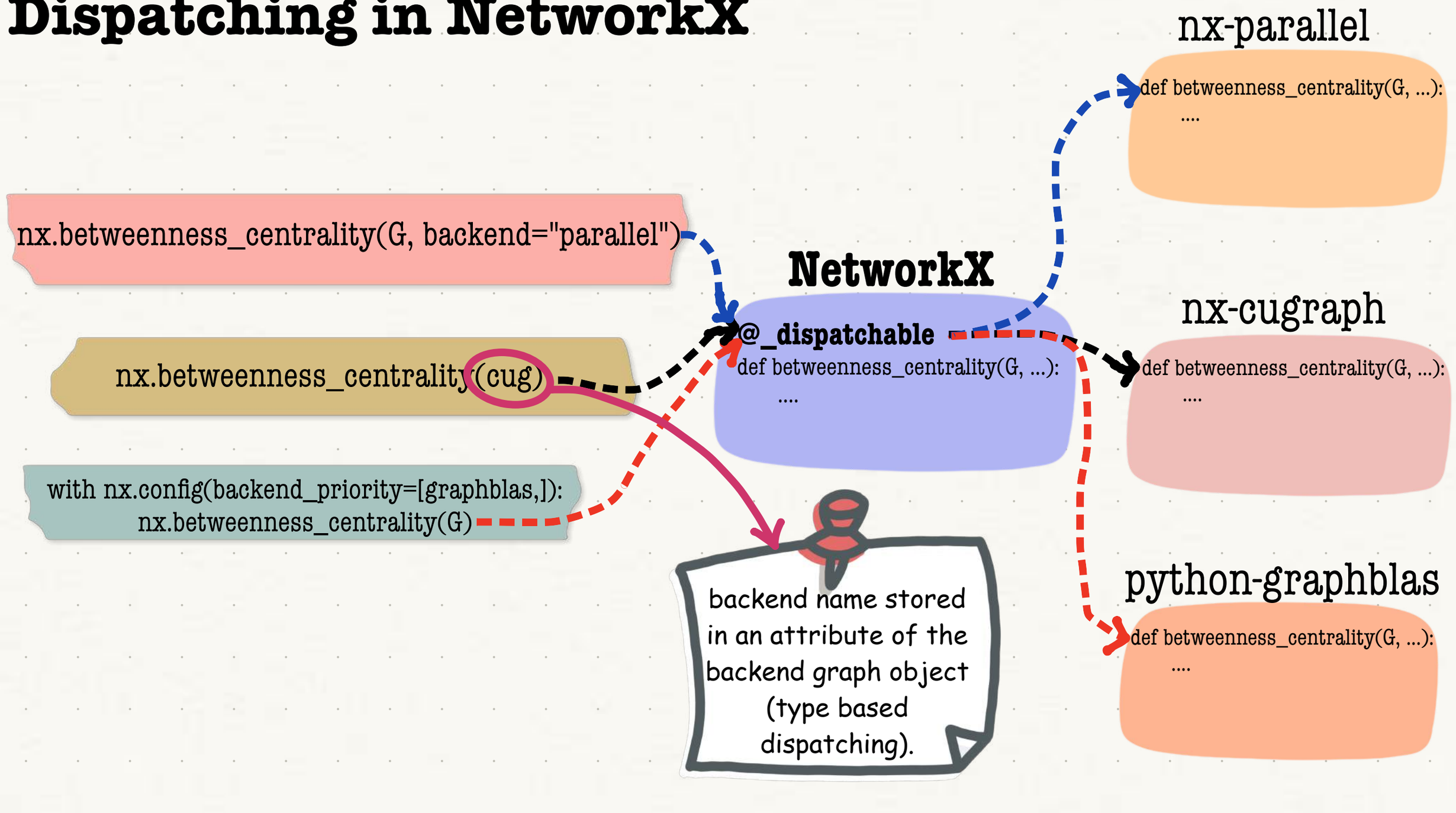
"The backend library"

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

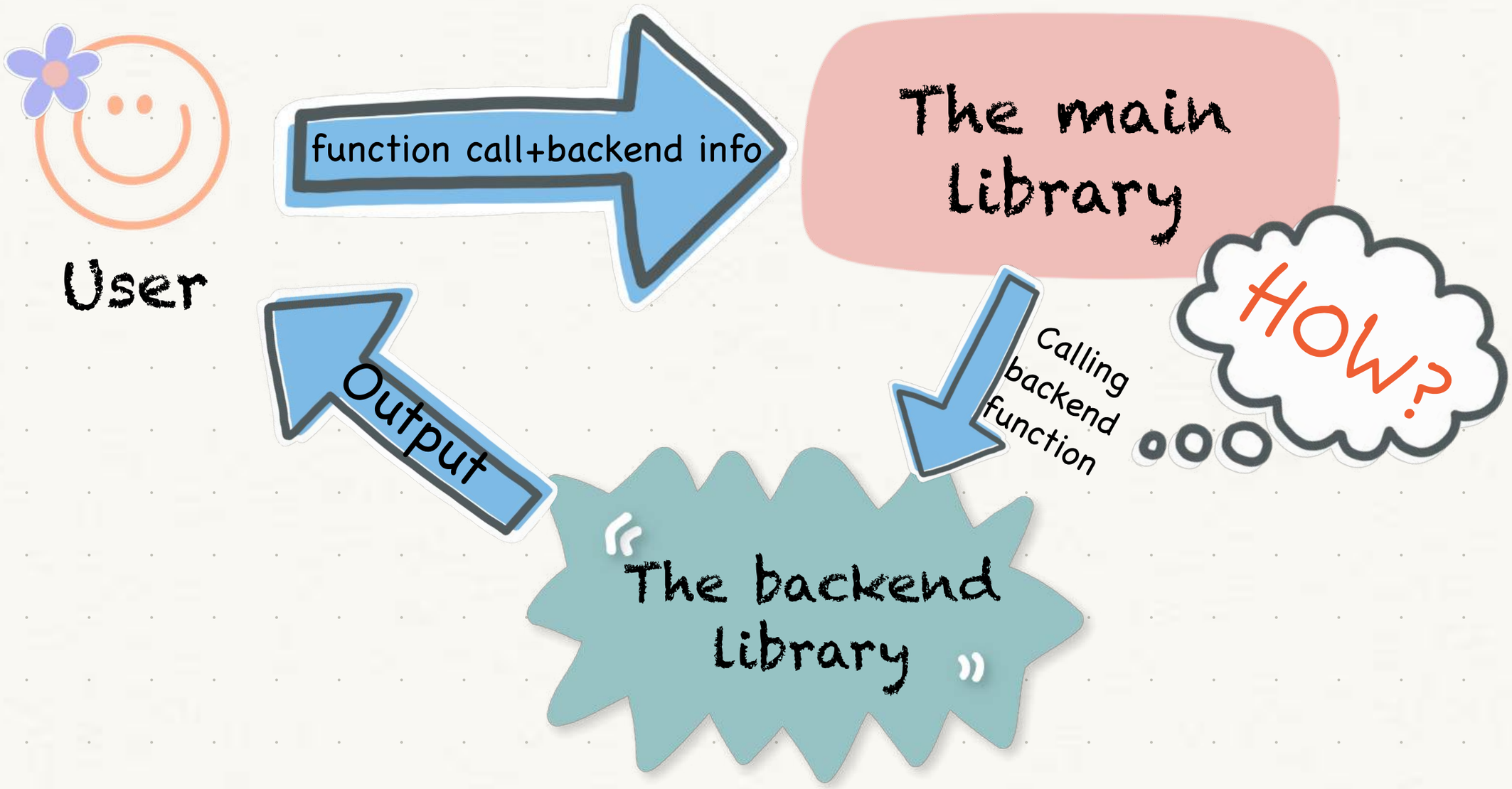
Dispatching in NetworkX



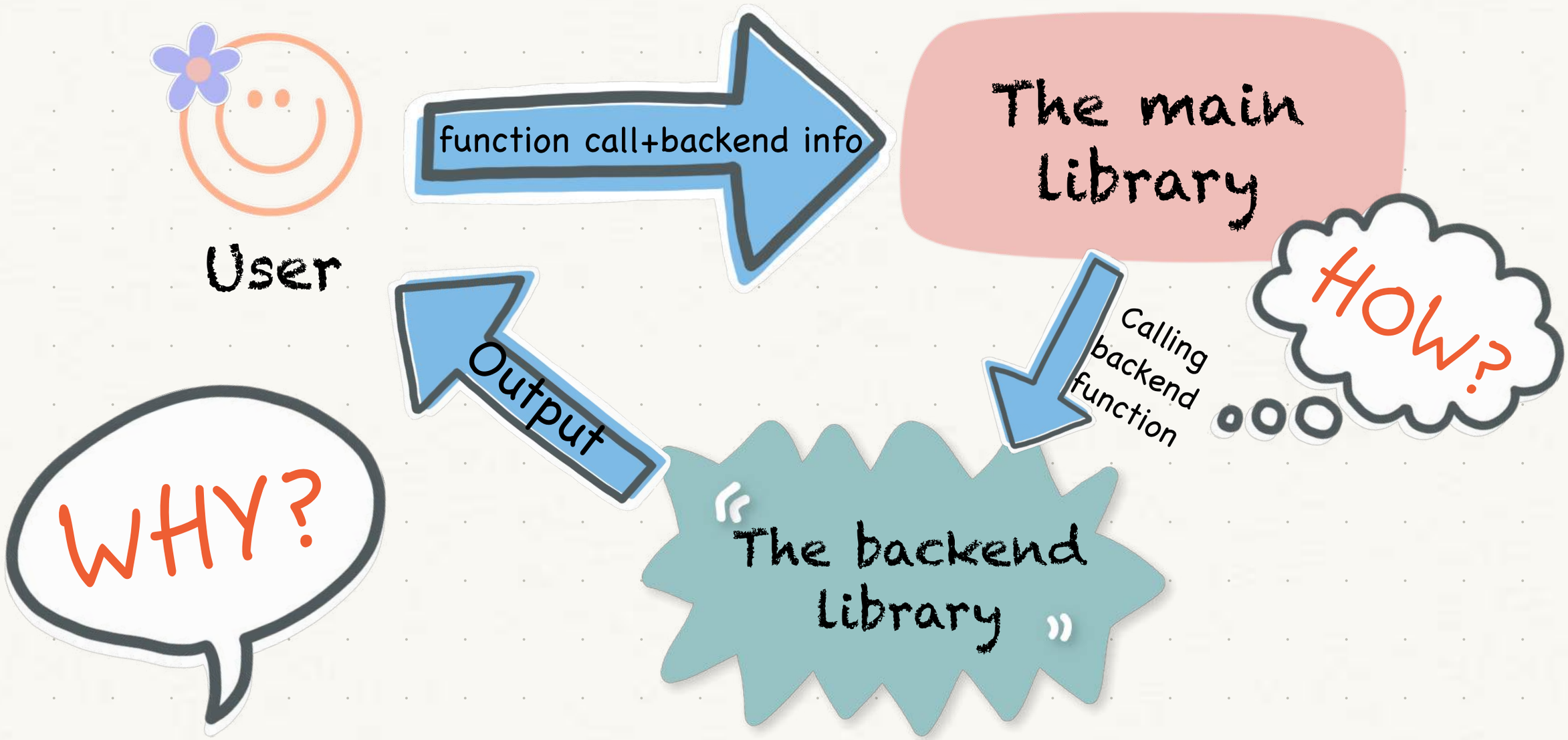
Dispatching in NetworkX



Dispatching



Dispatching

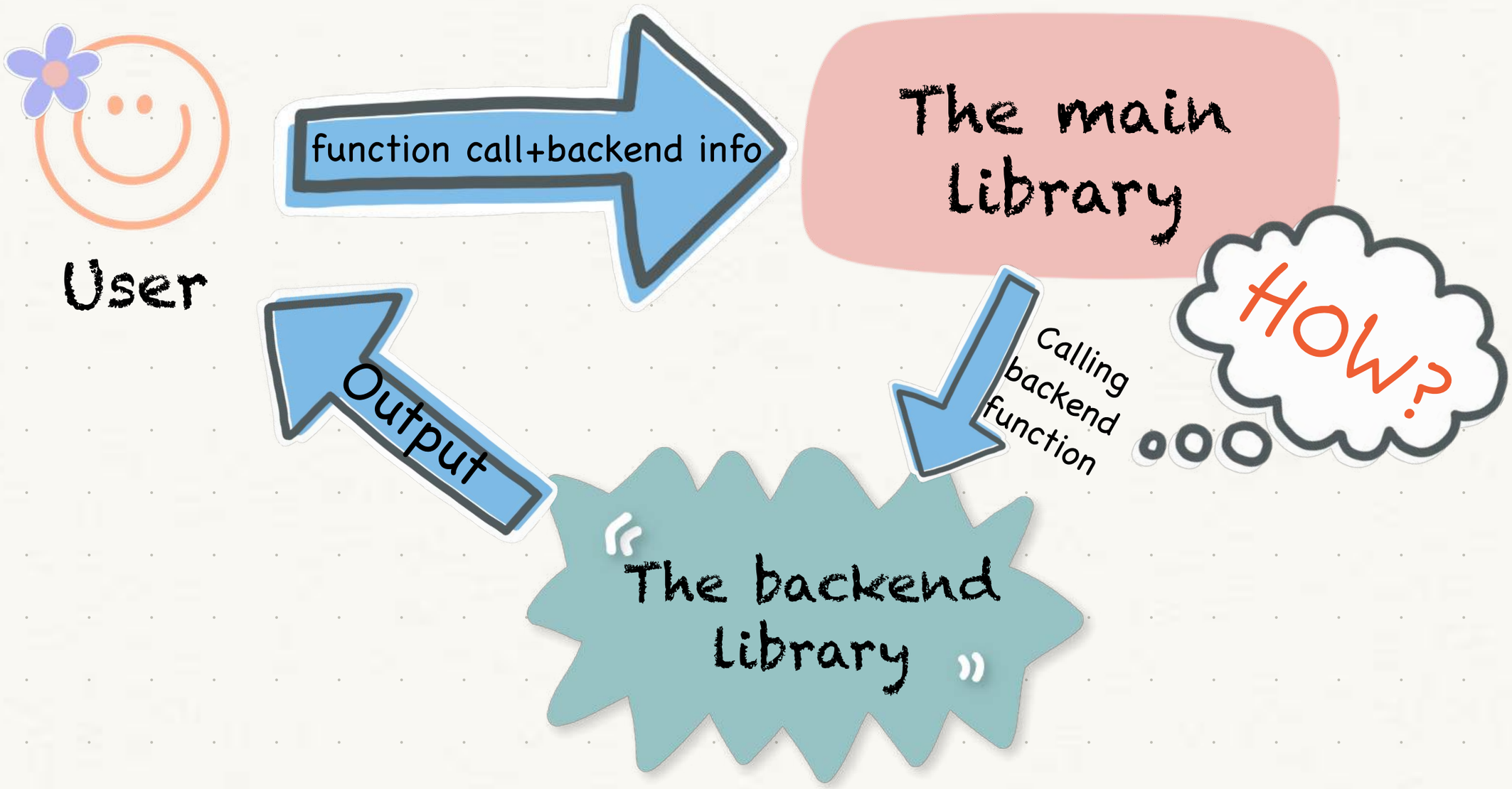


dispatching?

dispatching?

- 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...

Dispatching



does the main library calls the function in the backend library?

does the main library calls the function in the backend library?

```
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!
```

does the main library calls the function in the backend library?

```
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!
- populates the frontend library with a lot of backends' info
- not very flexible! NetworkX decides backend's existence
- What about private/ experimental backends?

HOW? does the main library calls the function in the backend library?

```
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!

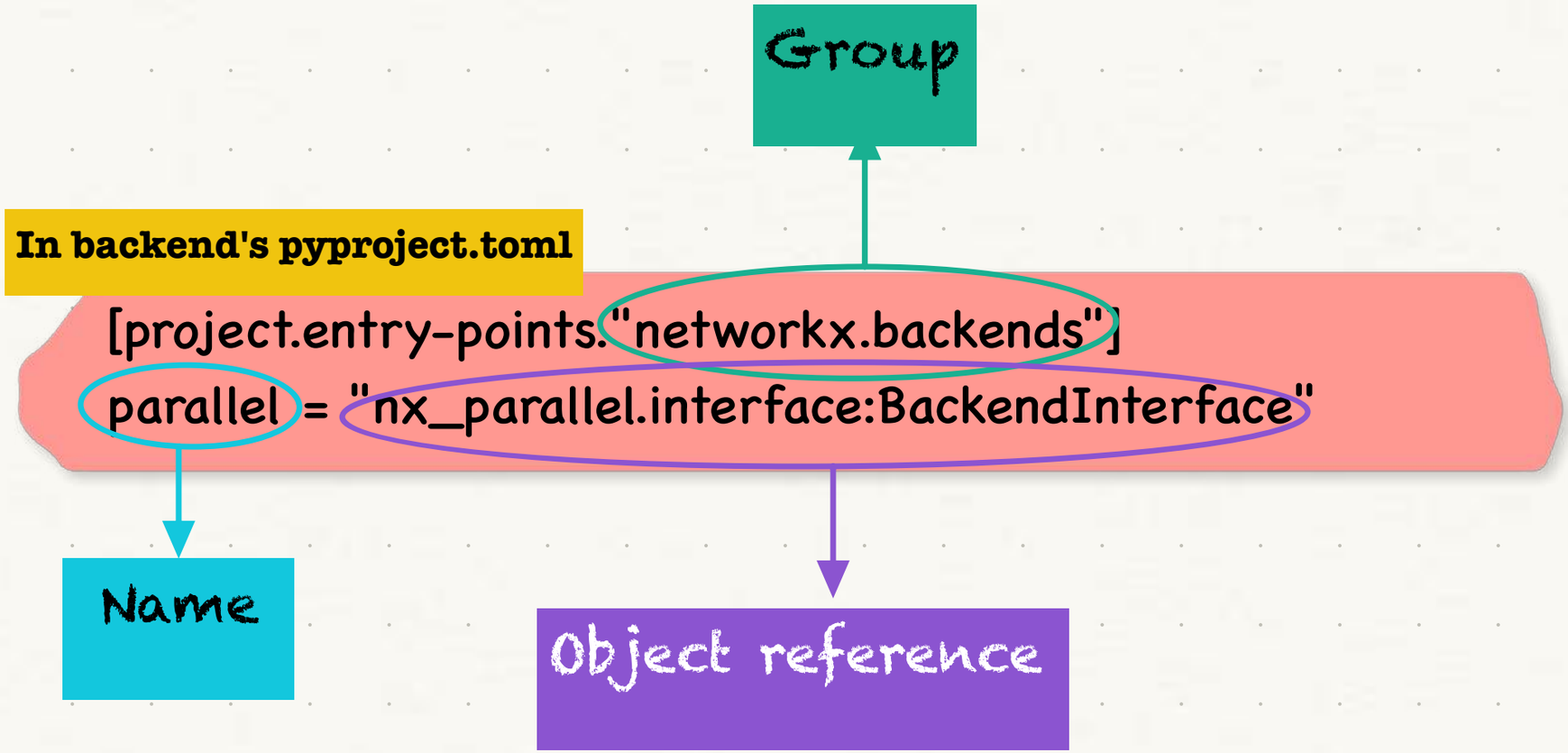
✗ populates the frontend library with a lot of backends' info

✗ not very flexible! NetworkX decides backend's existence

✗ What about private/ experimental backends?

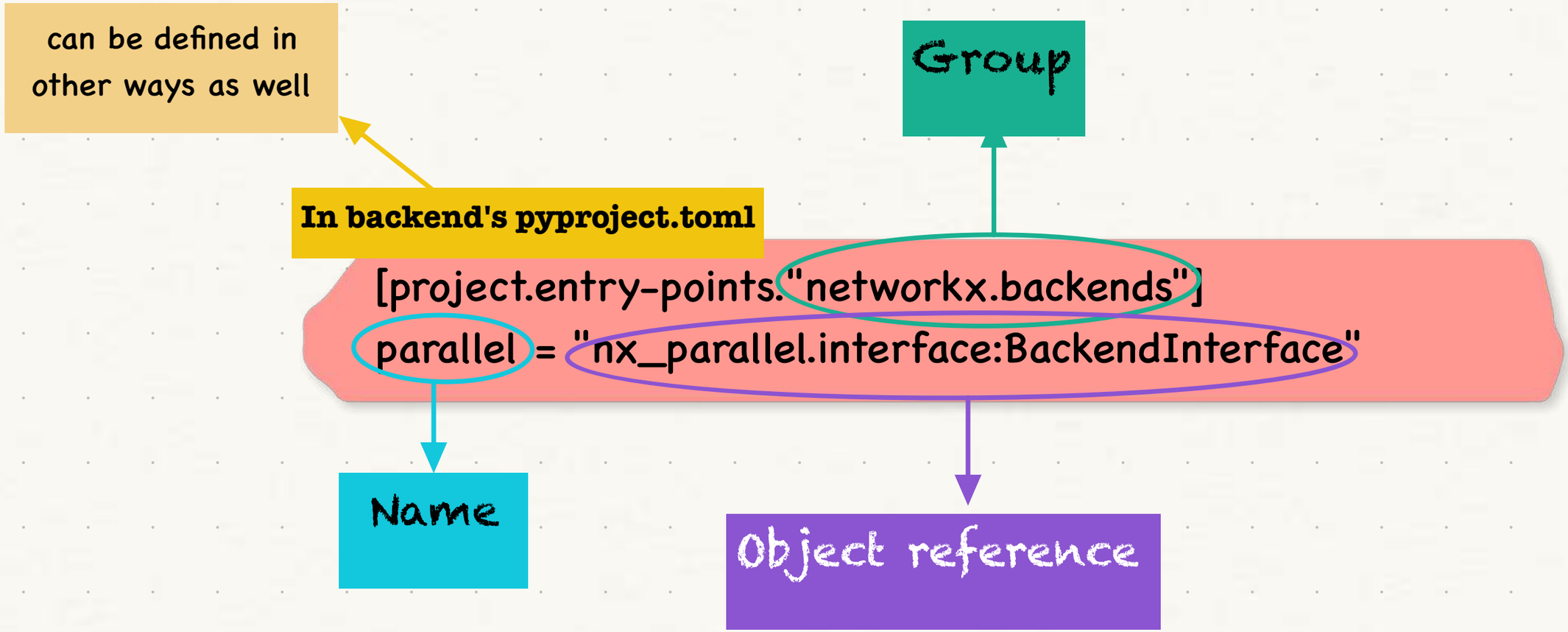
✓ Better way : entry-points

Defining a Python entry_point



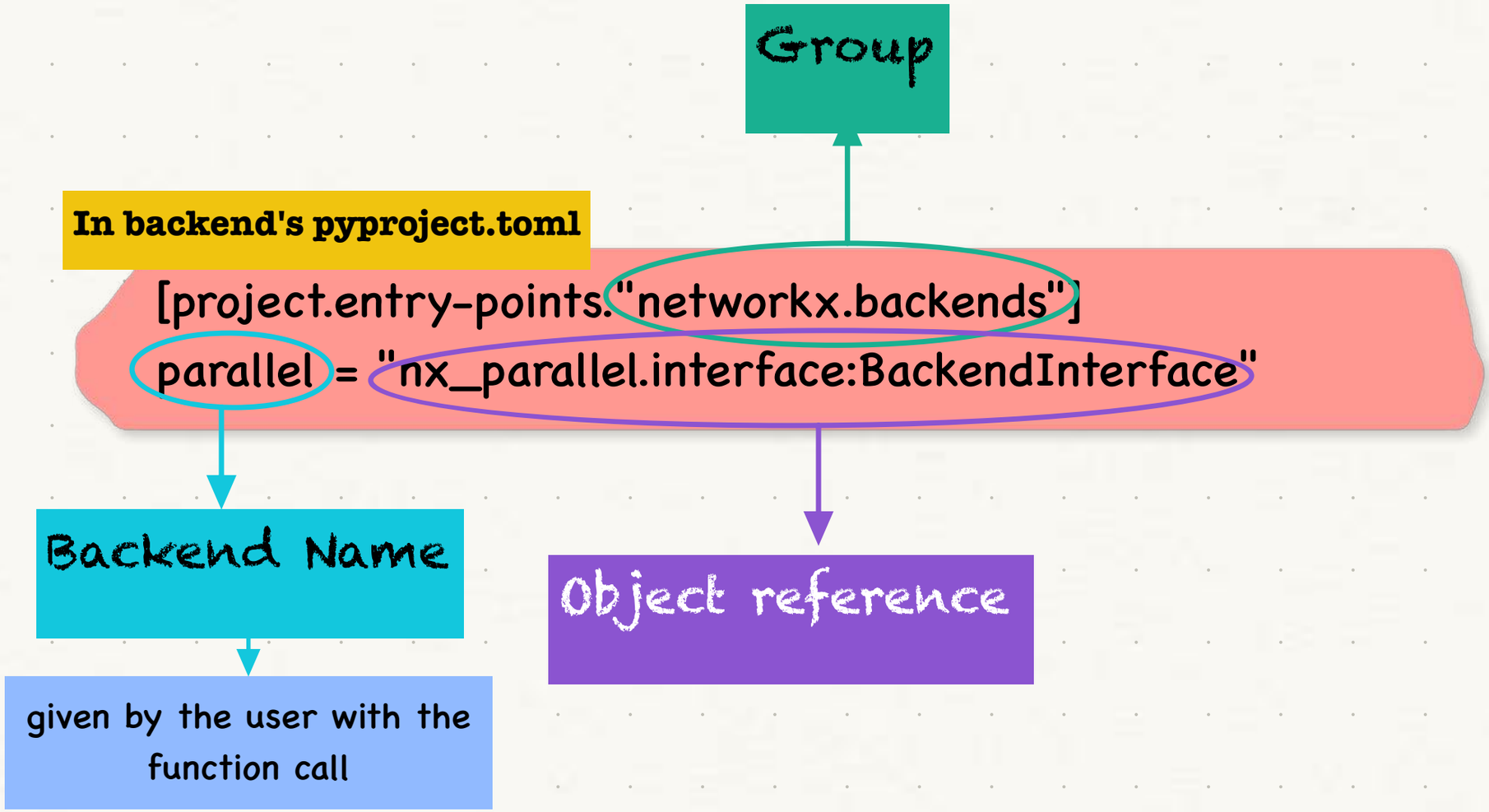
ref. https://packaging.python.org/en/latest/specifications/entry-points/

Defining a Python entry_point



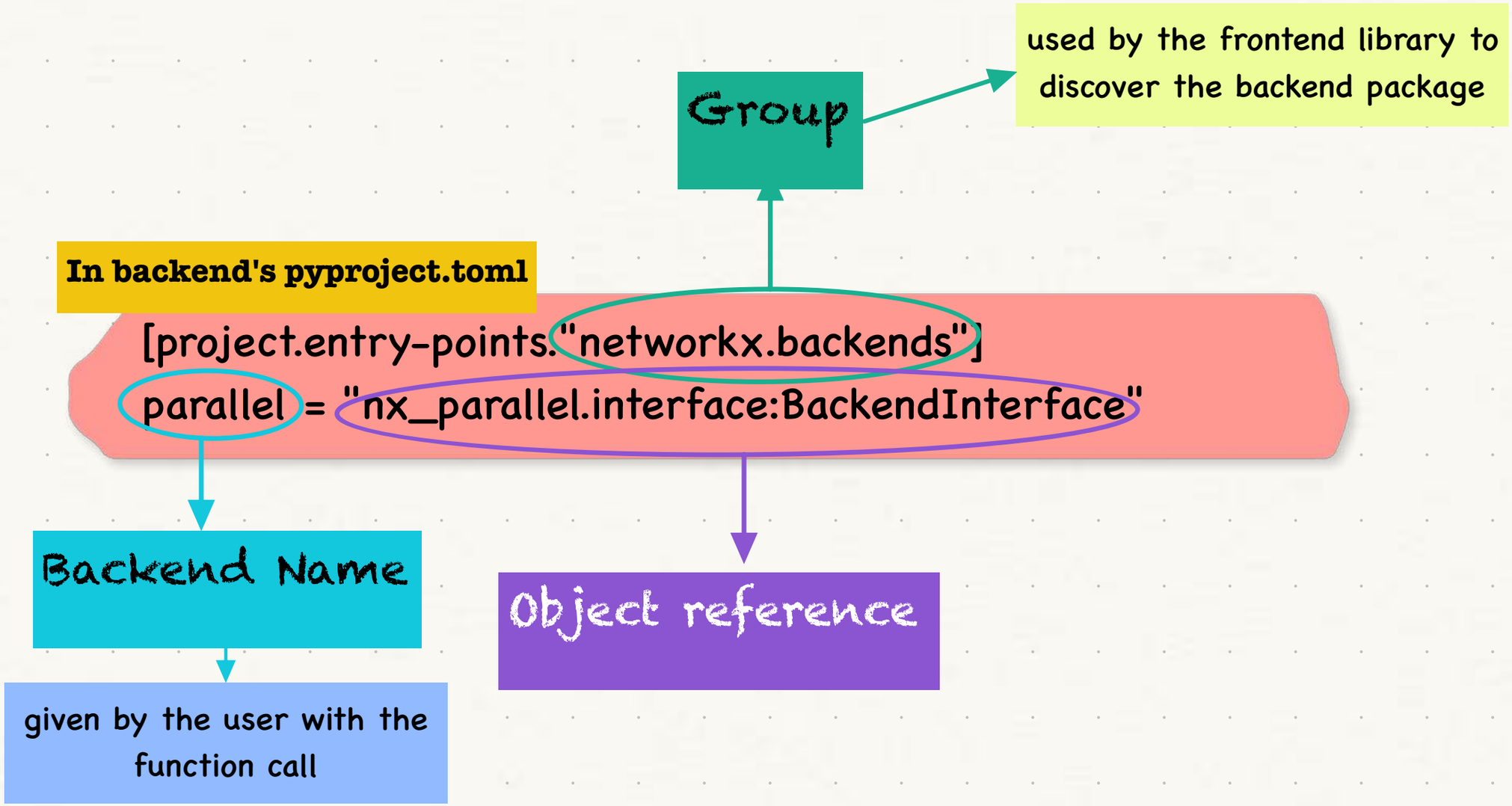
ref. https://packaging.python.org/en/latest/specifications/entry-points/

Defining a Python entry_point



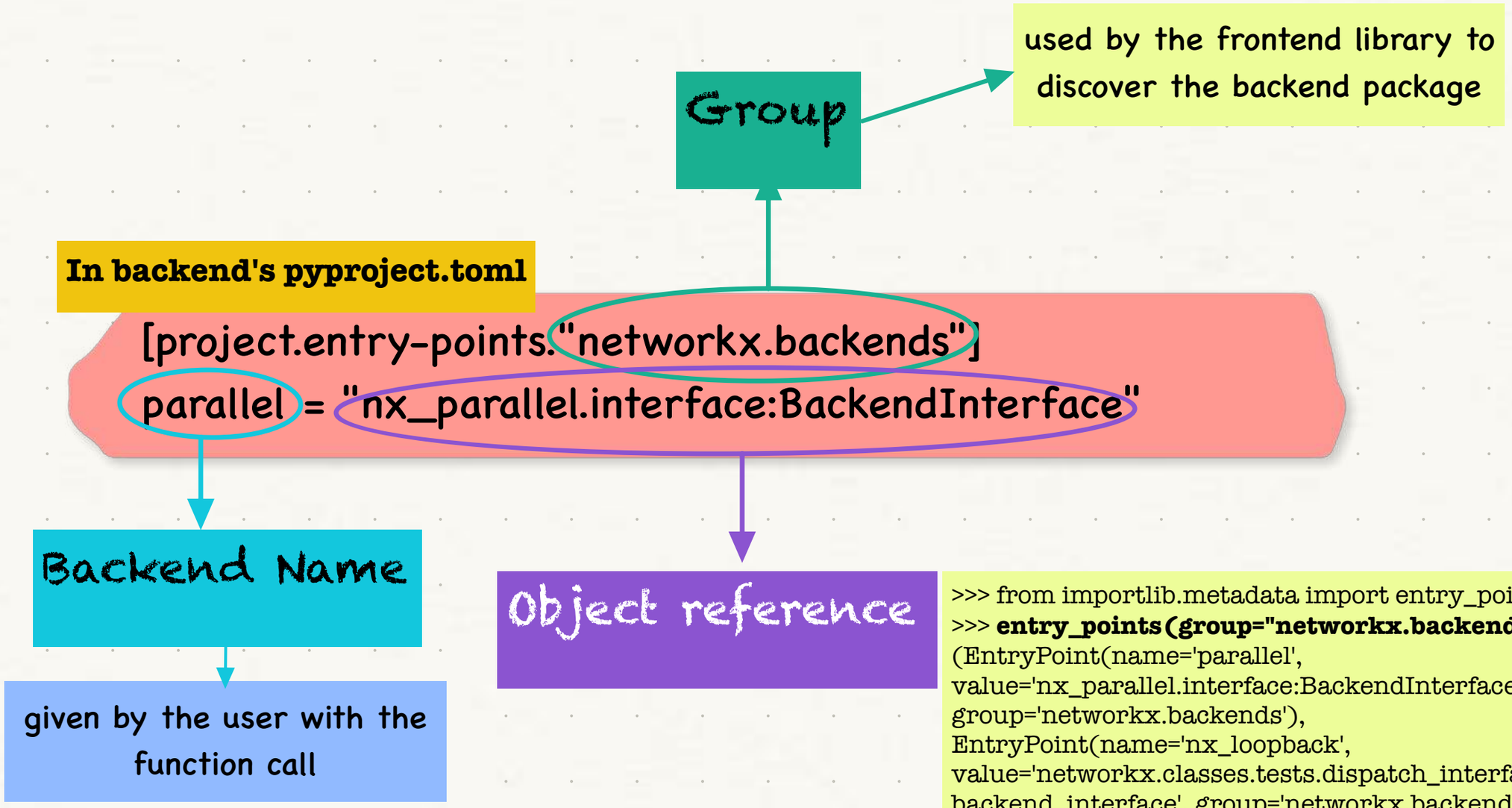
ref. https://packaging.python.org/en/latest/specifications/entry-points/

Defining a Python entry_point



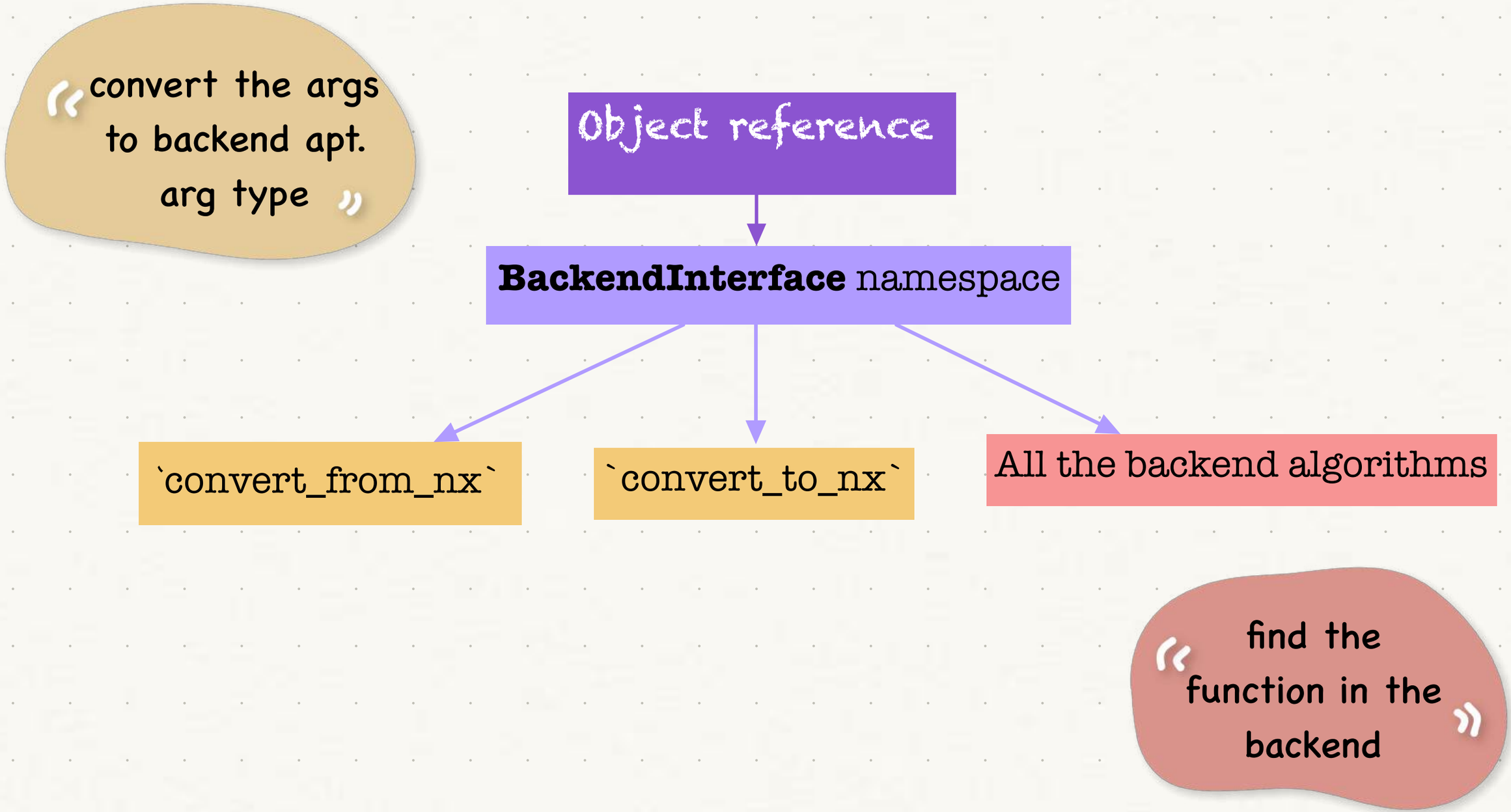
ref. https://packaging.python.org/en/latest/specifications/entry-points/

Defining a Python entry_point




```
>>> from importlib.metadata import entry_points
>>> entry_points(group="networkx.backends")
(EntryPoint(name='parallel',
value='nx_parallel.interface:BackendInterface',
group='networkx.backends'),
EntryPoint(name='nx_loopback',
value='networkx.classes.tests.dispatch_interface:
backend_interface', group='networkx.backends'))
```

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


convert the args to backend apt. arg type
Object reference
BackendInterface namespace
`convert_from_nx`
`convert_to_nx`
All the backend algorithms
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

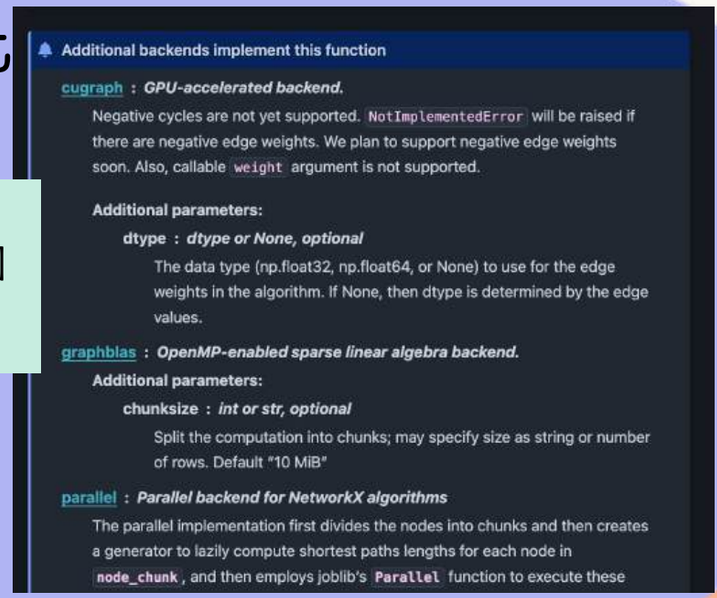
```
NETWORKX_TEST_BACKEND=parallel
NETWORKX_FALLBACK_TO_NX=True
pytest --pyargs networkx
```

Additional backend args

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

2nd entry point

```
[project.entry-
points."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

Configurations

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

fallback option

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/scientific-python/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


“
Thank you :)
”