Metagraph:
How to Make Graphs More Accessible

Stanley Seibert, on behalf of the Metagraph Team

March 7, 2022

Metagraph Team

- Jim Kitchen
- Paul Nguyen
- Stan Seibert
- Erik Welch

This work is funded via Intel as part of the DARPA HIVE project.

Outline

1 | The Problem with Graphs

2 | Metagraph

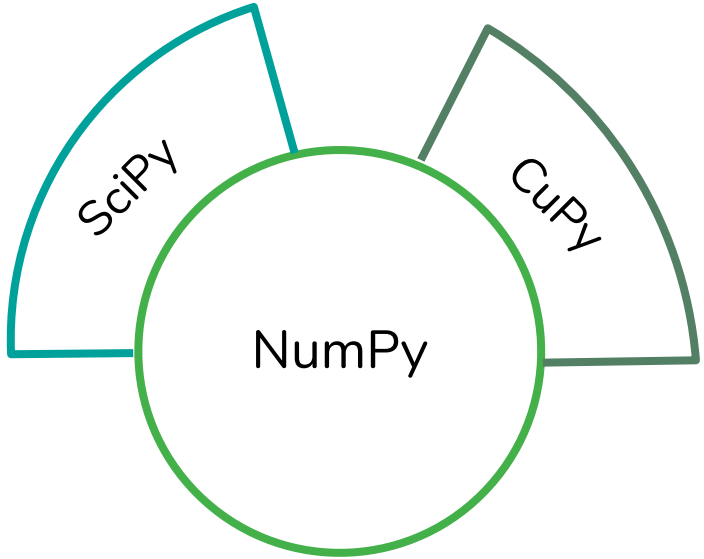
3 | GraphBLAS in Python

4 | Plans and Final Thoughts

The Problem with Graphs

Graphs are not a "monolithic" software domain

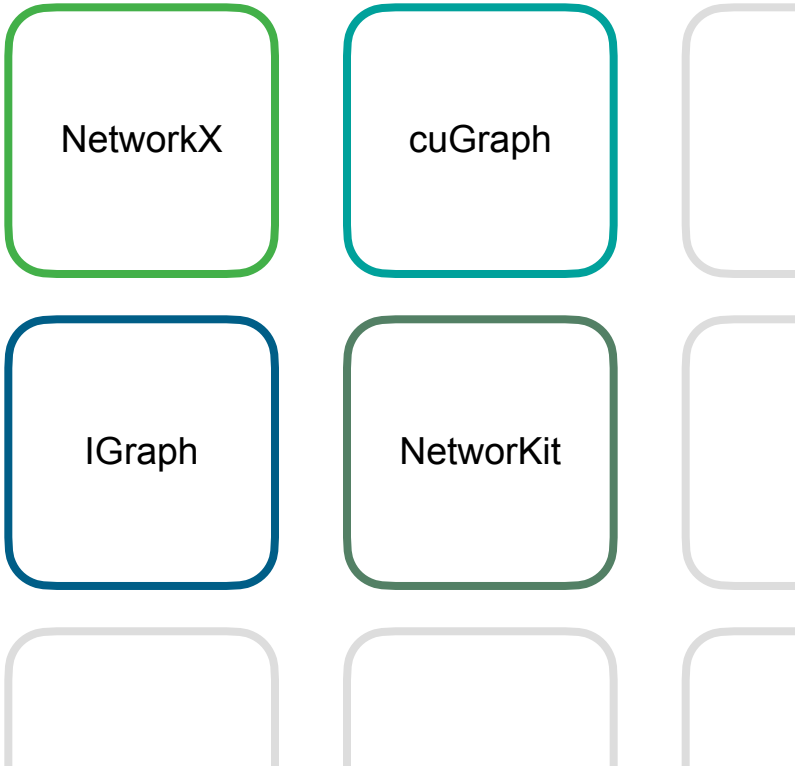
- Some software domains have:
  - One standard binary data representation
  - One (mostly) standard set of basic data operations and/or API
- Example: Array computing
  - Data: in-memory dense vector with shape, stride, and dtype
  - API: Whatever NumPy does
- New hardware target? New operations? Adopt existing data layout or API



Graphs are a software domain with no "center"

- *Many possible sets of basic operations:*
  - Vertex-centric, edge-centric, sparse linear algebra, ...
- *Many possible data representations:*
  - Dense, CSR, Block CSR, edge lists, dictionaries, ...
- *Many possible hardware targets:*
  - CPUs, GPUs, FPGAs, ...

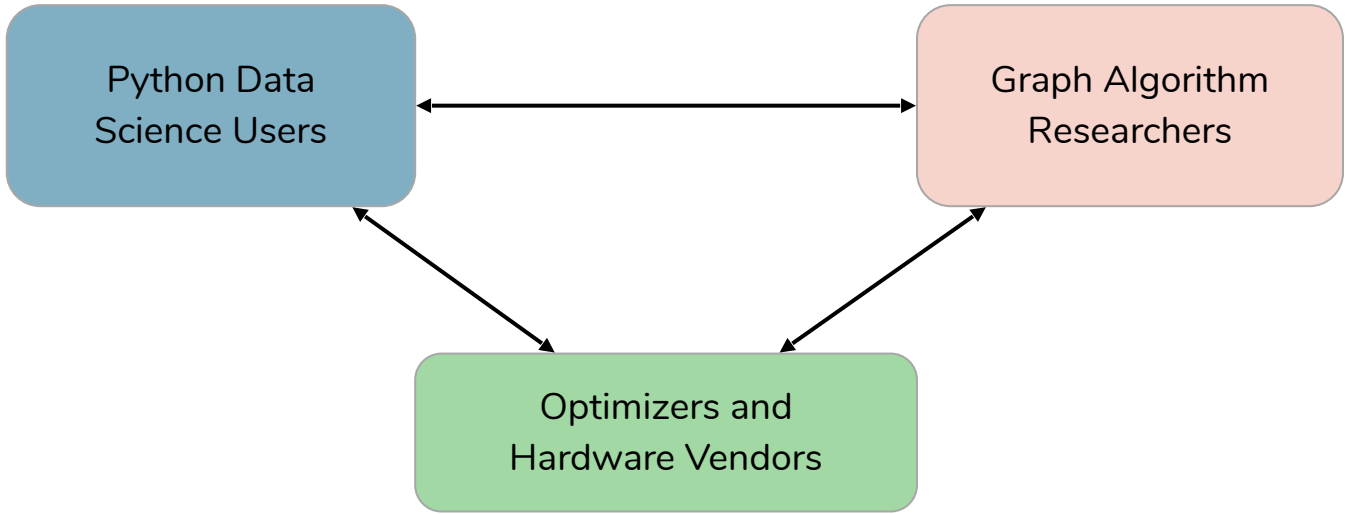
- Result: many interesting, but isolated projects!



Who are we solving for?

Run Conway's Law backwards:

- Design software to create the interactions that lead to long term innovation and success.



A Vertical Spike Through the Problem Space

User-facing API and Integration Layer

Metagraph

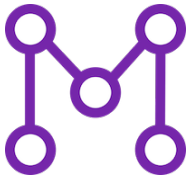
Building blocks for graph algorithms

GraphBLAS in Python: grblas

Framework for optimizing graph algorithms for multiple targets

MLIR GraphBLAS
(not talking about this today)

Metagraph



Metagraph: building a graph library from *other* libraries

- Metagraph is an experiment in blending graph libraries together:
  - *Multiple-dispatch function system*
    - Dispatching between implementations driven by extensible type system
  - *Automatic data translation*
    - Data is translated as needed to move from one function to the next
  - *Dask support*
    - Build end-to-end Dask computations that include Metagraph operations
  - *Dask DAG compilation*
    - Identify task subgraphs and fuse into JIT-compiled functions
  - **Everything is a plugin**
    - Types, APIs, implementations, translators, and compilers

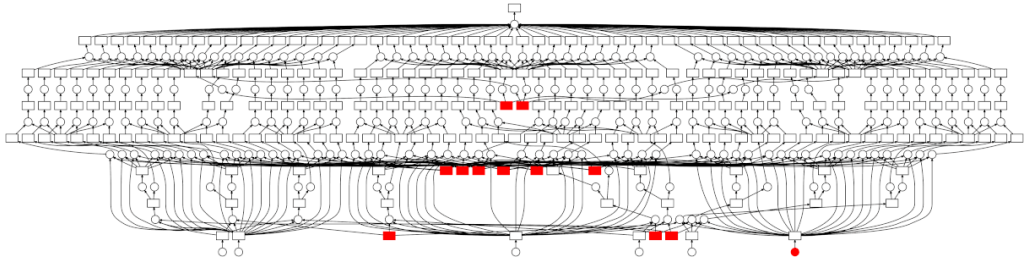
Metagraph Status

Plugins

- Ships with core package:
  - graphblas, scipy, pandas, numpy
- External:
  - igraph
  - cuda
  - stellargraph
  - karateclub
  - cogdl
  - katanagraph
  - numba
  - mlir

API categories

- Clustering / Community Detection
- Graph / Node Embedding
- Flow
- Graph Matching
- Subgraph Selection
- Traversal / Path Selection
- Vertex Ranking

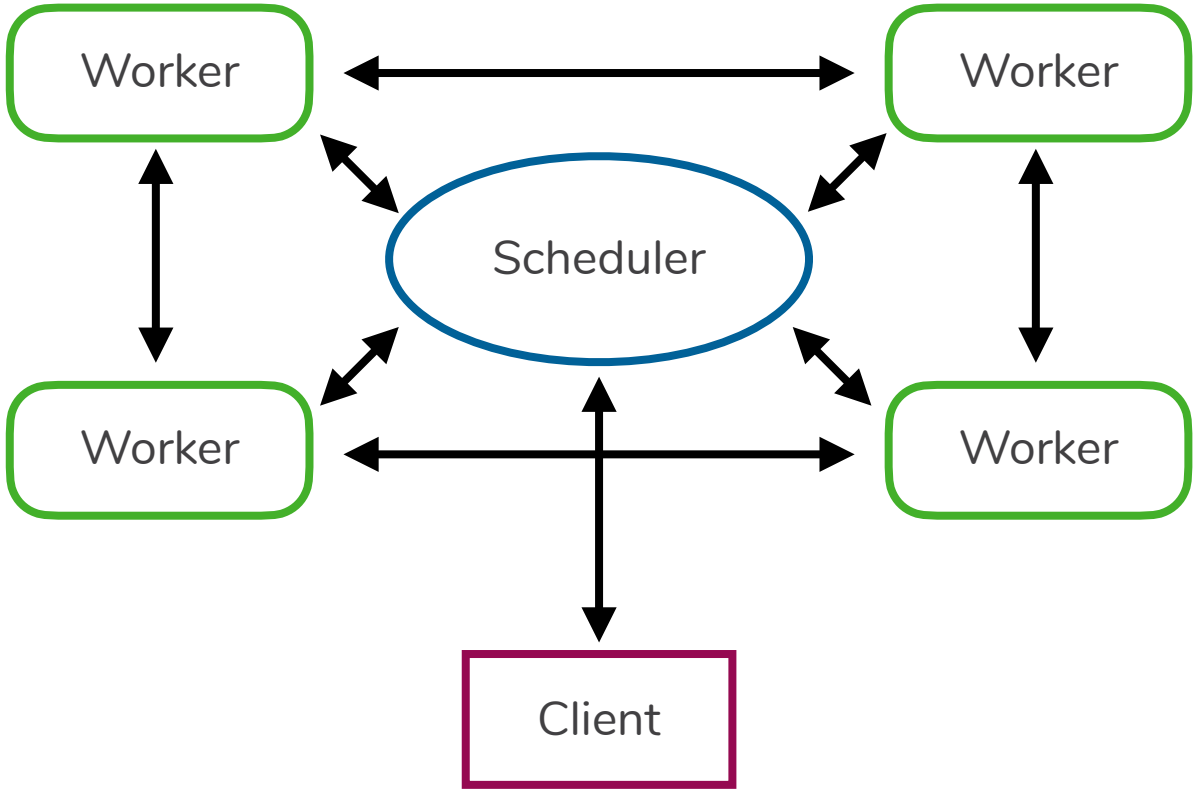


Easy, powerful parallel programming for Python

- **Familiar:** Implements parallel NumPy and Pandas objects
- **Fast:** Optimized for demanding for numerical applications
- **Flexible:** for sophisticated and messy algorithms
- **Scales up:** Runs resiliently on clusters of 100s of machines
- **Scales down:** Pragmatic in a single process on a laptop
- **Interactive:** Responsive and fast for interactive data science

http://dask.pydata.org

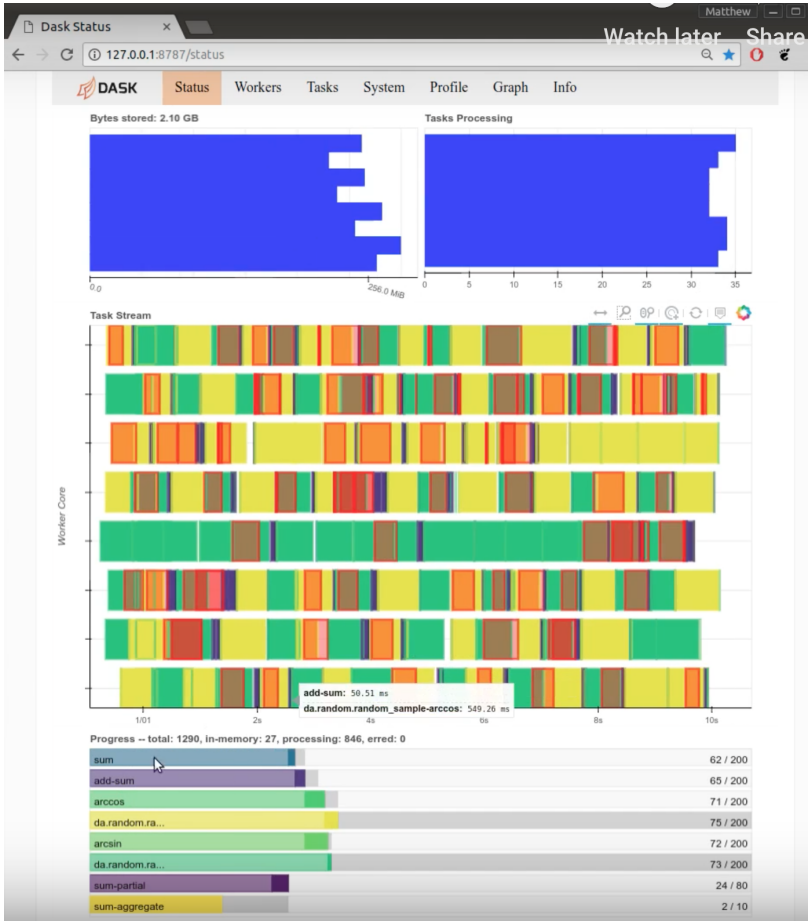
Dask for distributed computing



- Dask API works the same distributed as locally
- You can run the distributed scheduler on a single system!

Dask dashboard

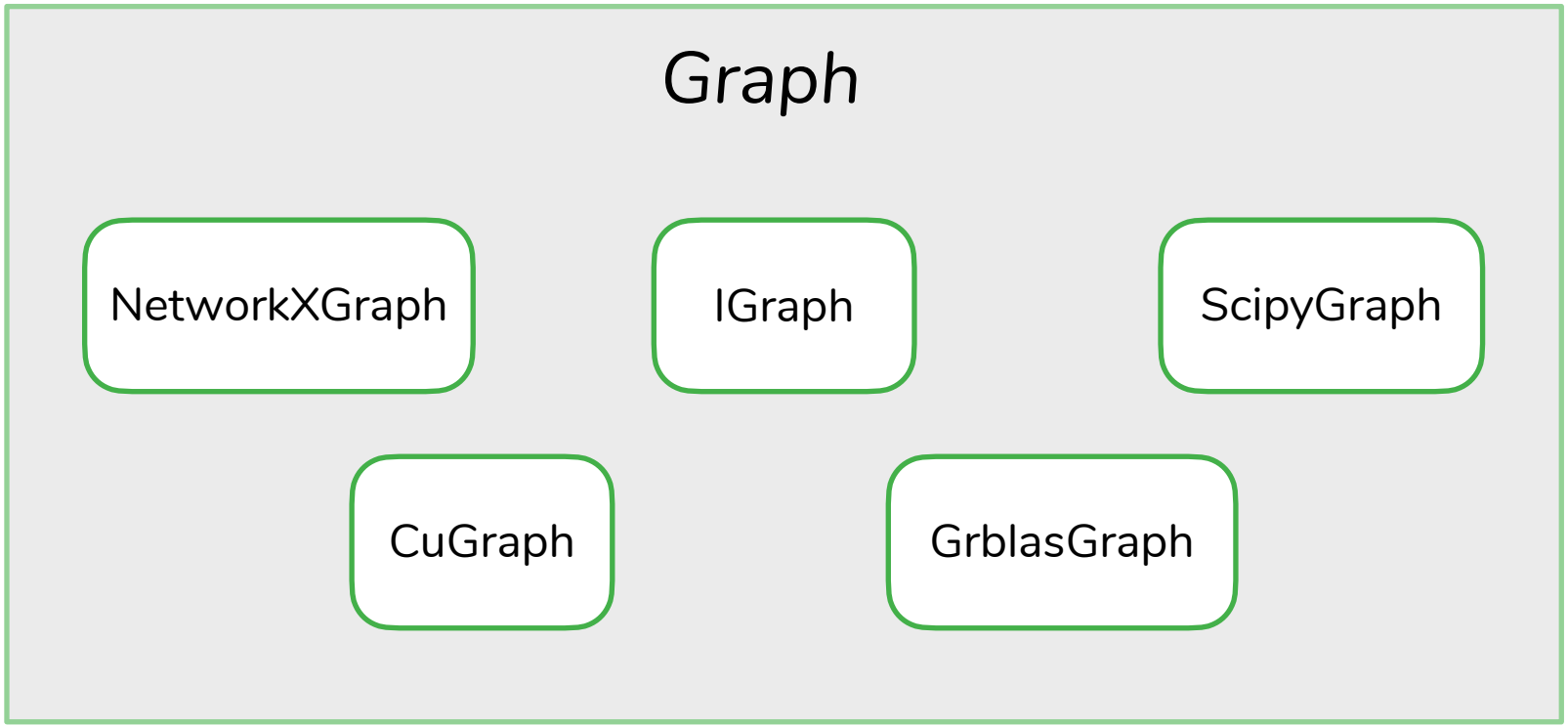
- Web interface provided by scheduler showing diagnostics on the cluster
- What tasks are running where?
- Progress on current computations
- Resource usage
- Profiling data



Why is Metagraph using Dask?

- **Surprise: it isn't for partitioned graph computation!**
  - Distributed graphs are hard, leave it to the professionals (i.e. the library backends)
- Dask provides:
  - Robust support for arrays and dataframes
    - Where do graphs come from? Arrays and dataframes.
  - Excellent distributed I/O support for scalable data loading
  - Parallel task scheduling
  - "Definition by execution" of task graphs
    - Lazy evaluation creates opportunities for inter-task optimization

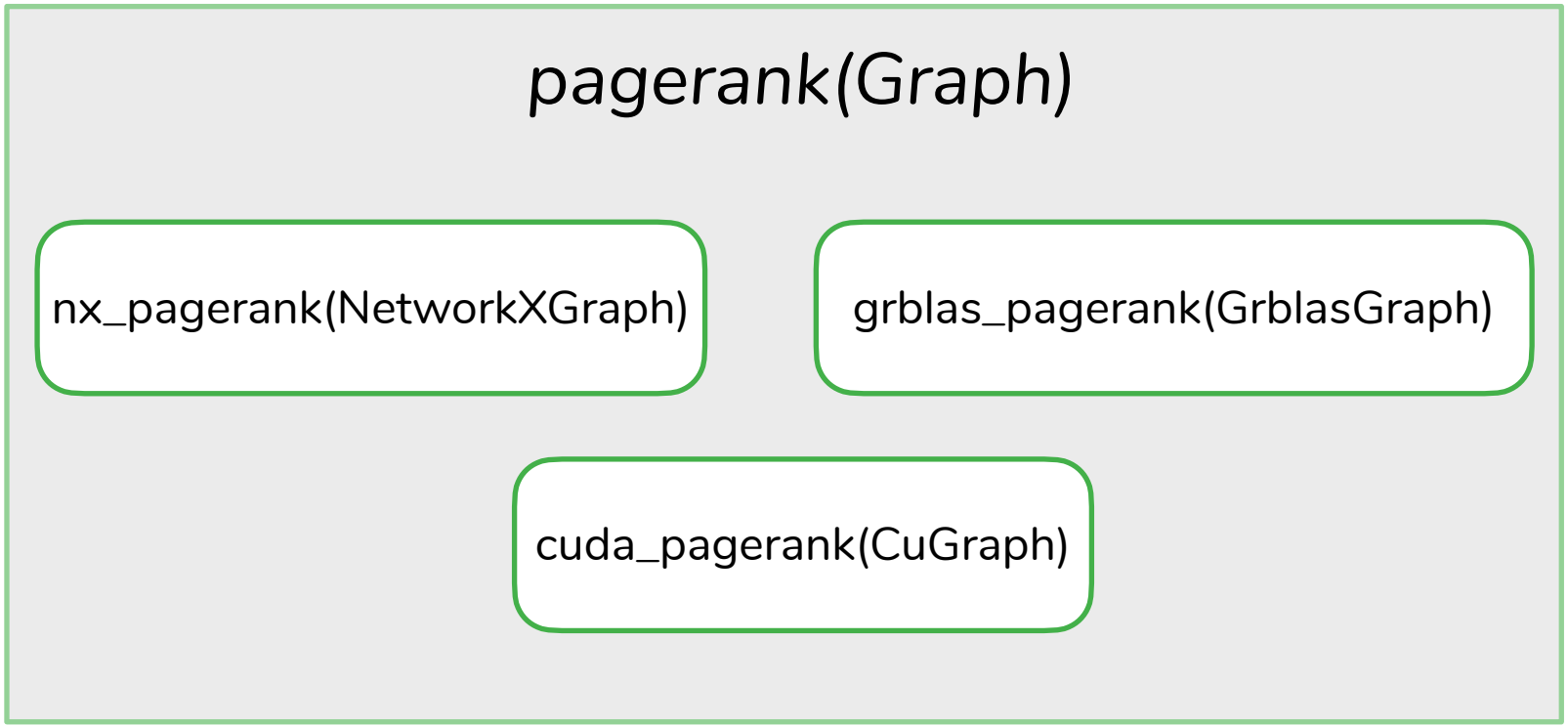
Stitching together separate libraries: Types



Notes on Types

- Abstract Types:
  - Equivalence classes of Concrete Types, not a "base class"
  - Define a category of types that could potentially be converted between each other
- Concrete Types:
  - Define an in-memory data layout in a specific memory space
  - Ex: "SuiteSparse adjacency matrix in CPU memory" or "cuGraph in GPU memory"
- Property system that allows both abstract and concrete types to have specialization without multiplying distinct types

Stitching together separate libraries: Algorithms



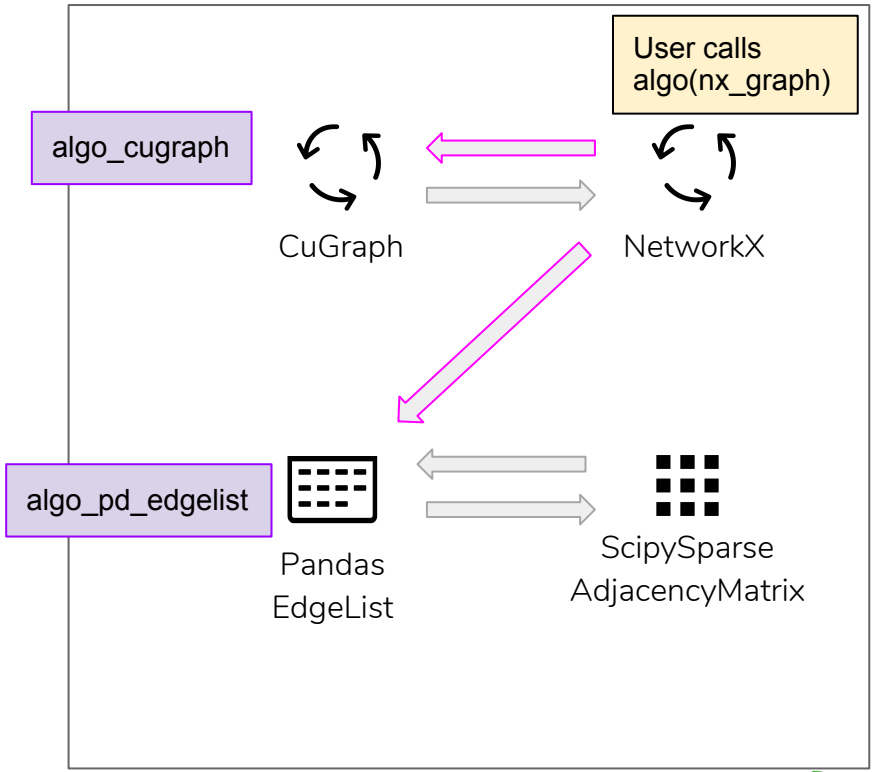
Notes on Algorithms

- Abstract Algorithms:
  - Type signature can only refer to Abstract Types (or common Python types like lists of Abstract types, numeric scalars, strings)
- Concrete Algorithms:
  - Indicate which abstract algorithm they implement
  - Type signature can only refer to Concrete Types (or common Python types)
- Both use Python type annotations to indicate this metadata

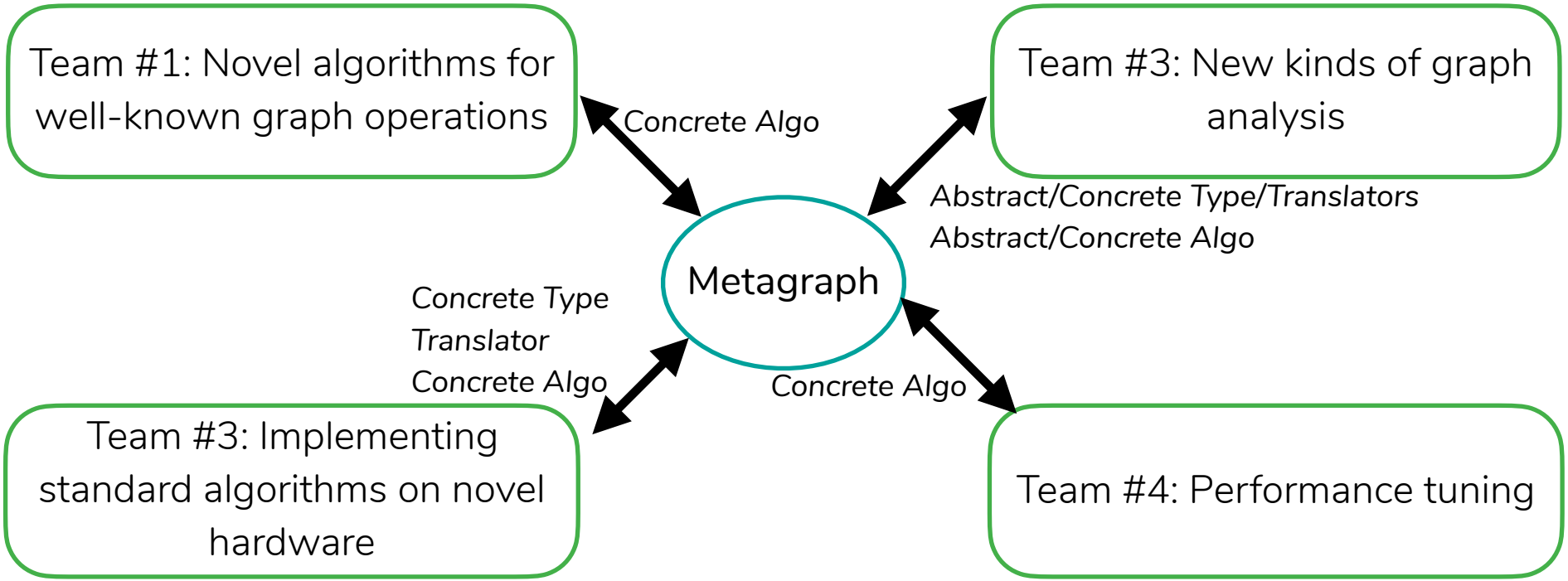
Automatic data translation + multiple dispatch

- The glue that binds the system together
- Dispatcher has list of translators between the concrete types
- Multi-stage translation allowed
- Current dispatcher heuristic tries to minimize translation steps for all arguments
  - Will likely need to get more sophisticated in future

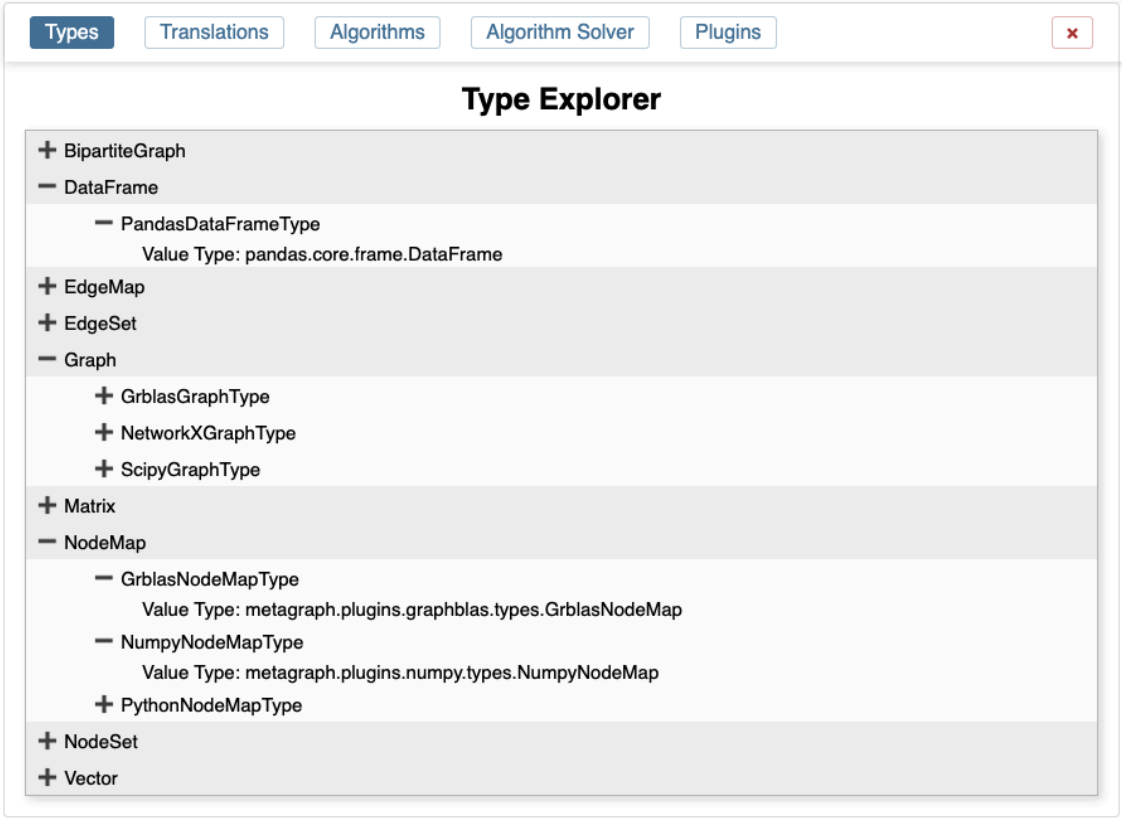
- *User can always force a specific algorithm backend if desired*



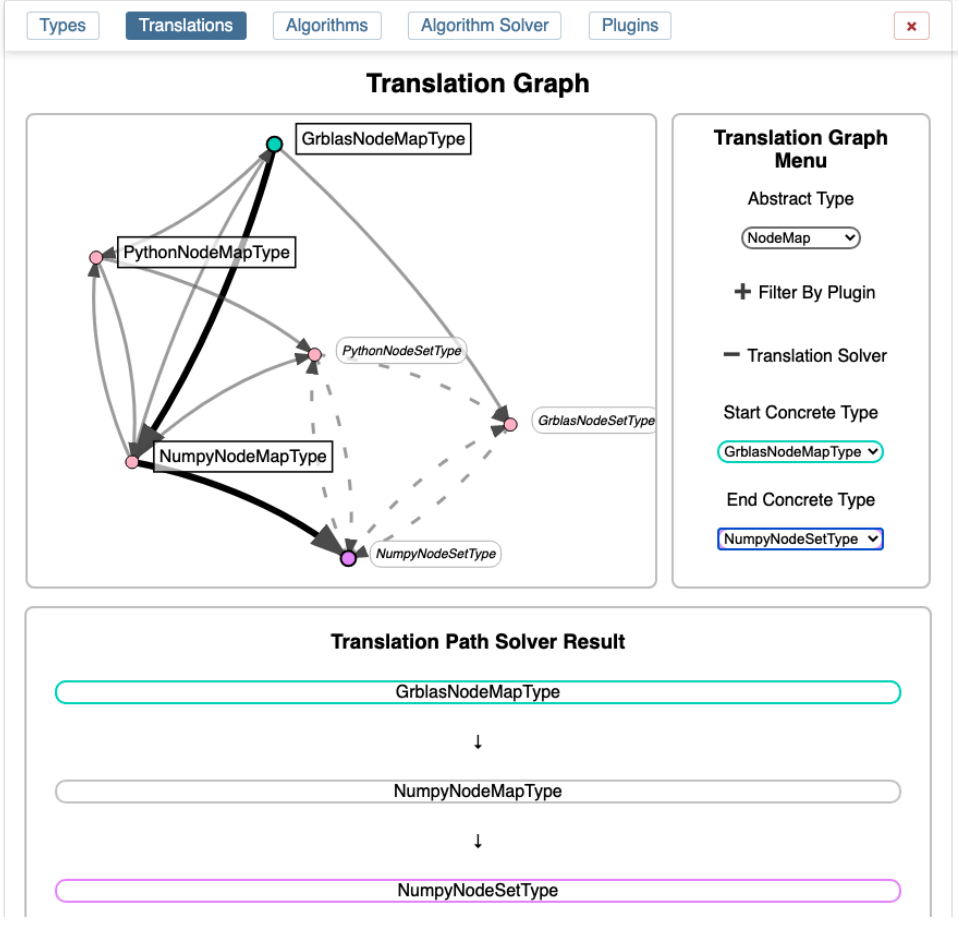
Designing for Community Interactions



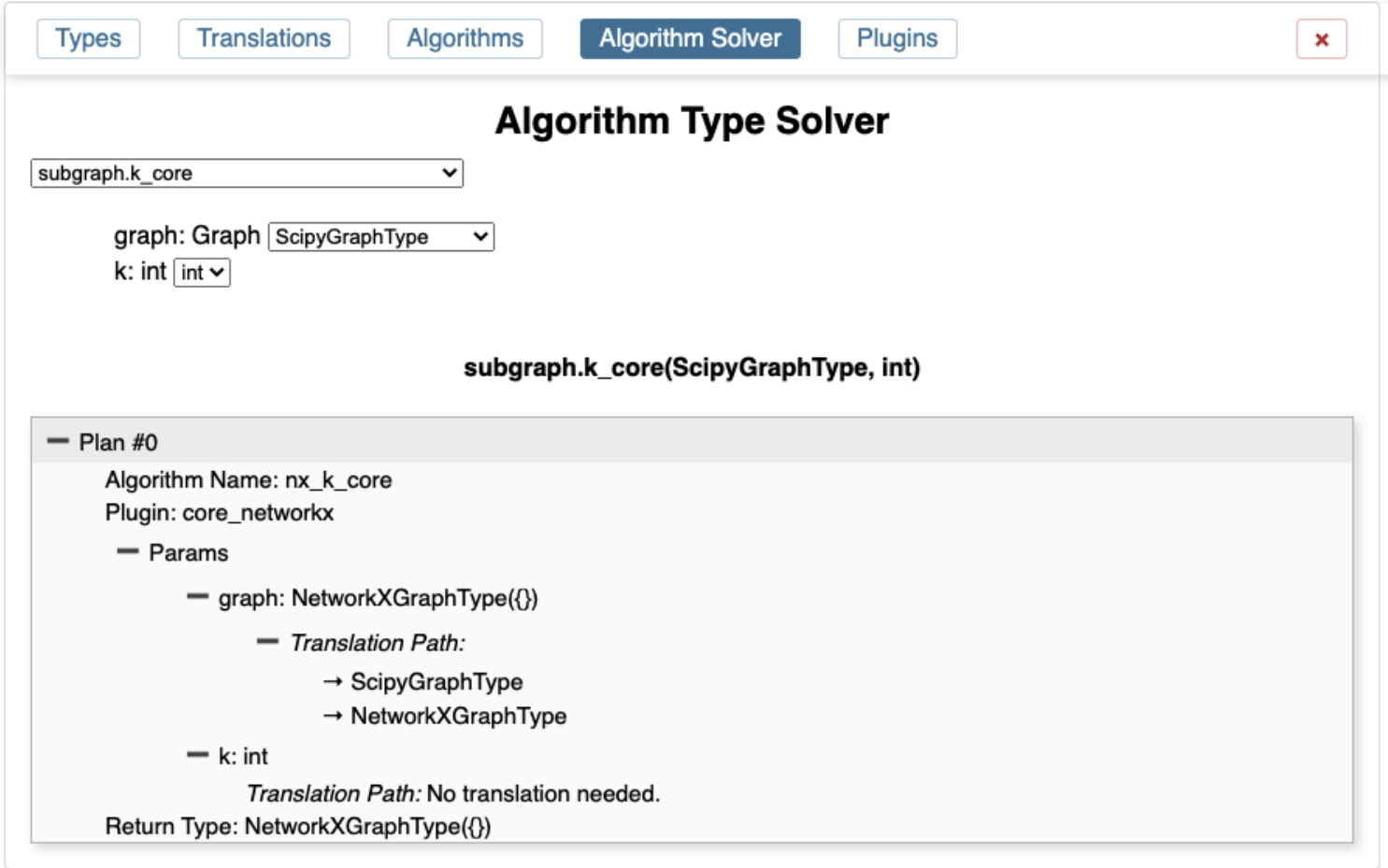
Metagraph Explorer: Types



Metagraph Explorer: Translations



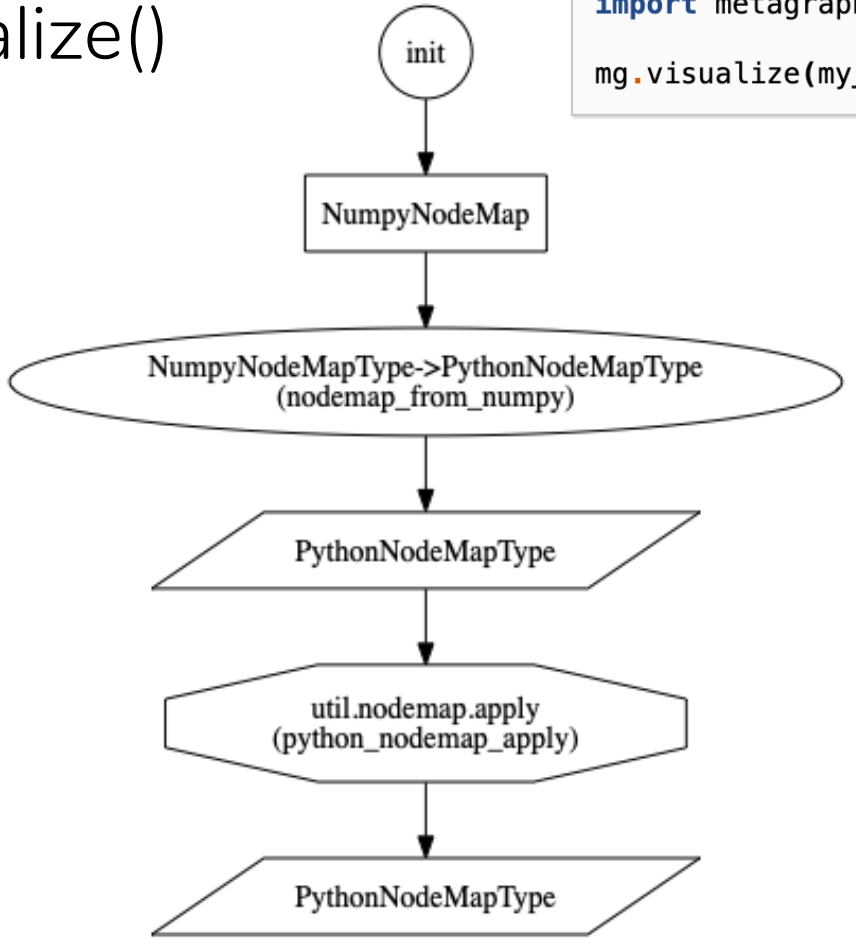
Metagraph Explorer: Algorithm Solver



Custom Dask visualize()

```
import metagraph as mg

mg.visualize(my_dask_object)
```



input value

translator

typed value

algorithm

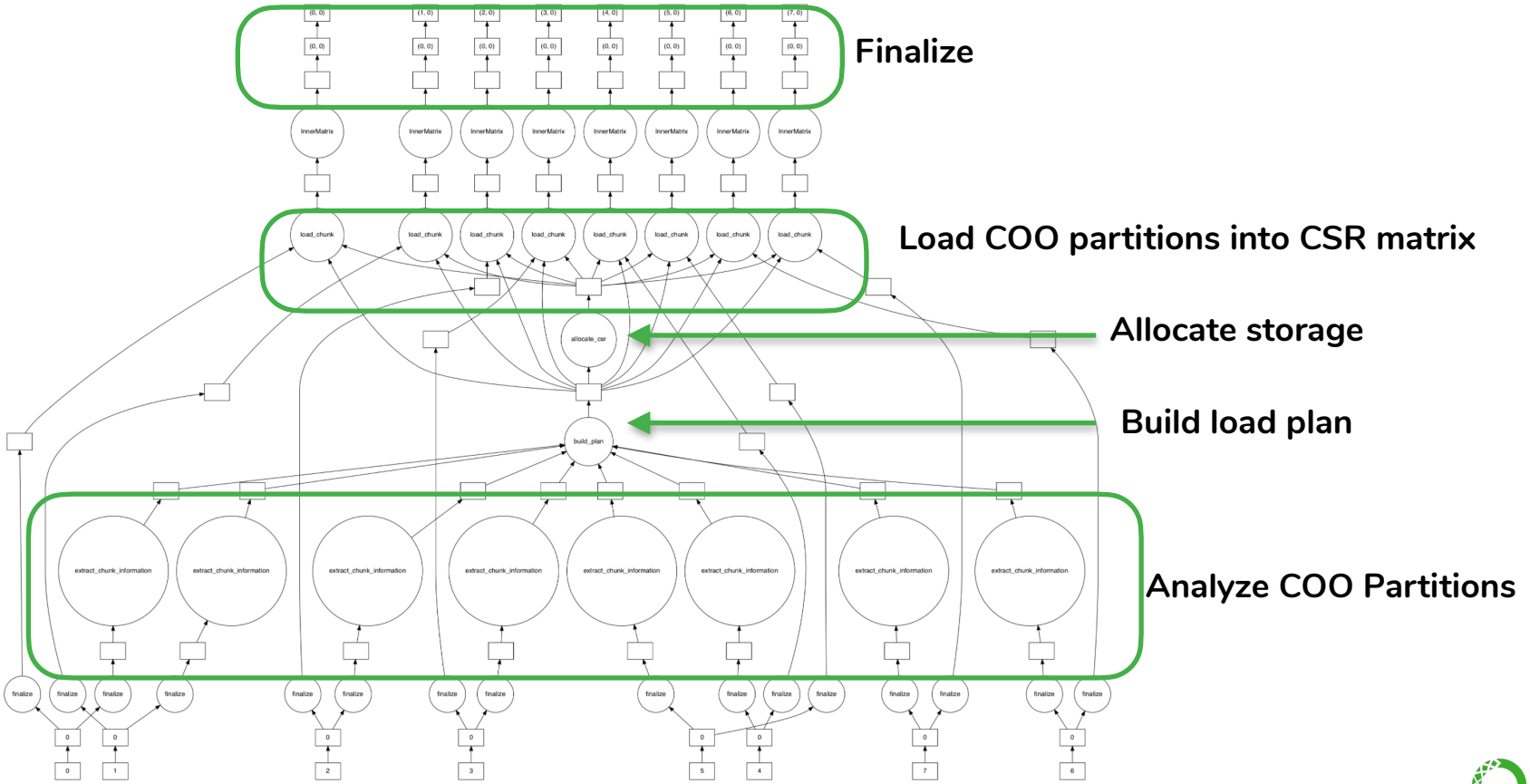
typed value

Built-in Dask visualize() is very extensible with custom graphviz attributes:

`function_attributes`:
task key → dict

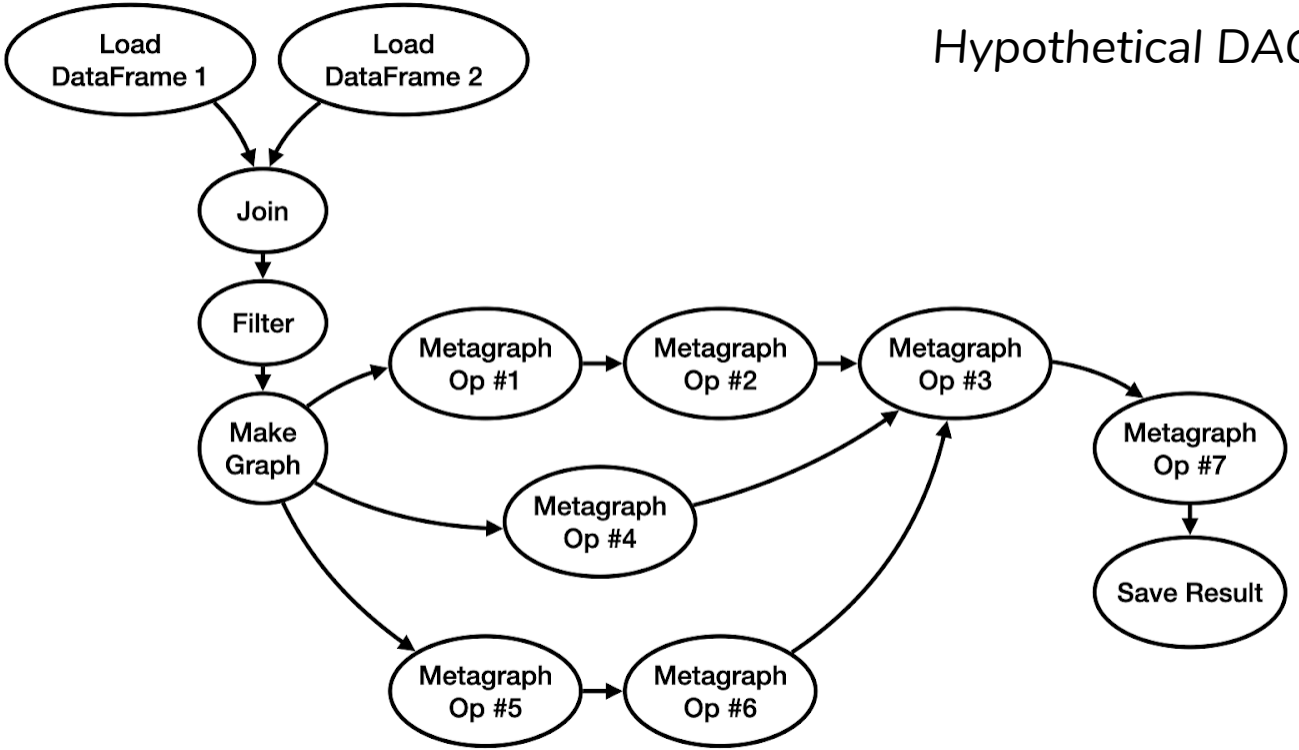
`data_attributes`:
task key → dict

Distributed COO->CSR loading



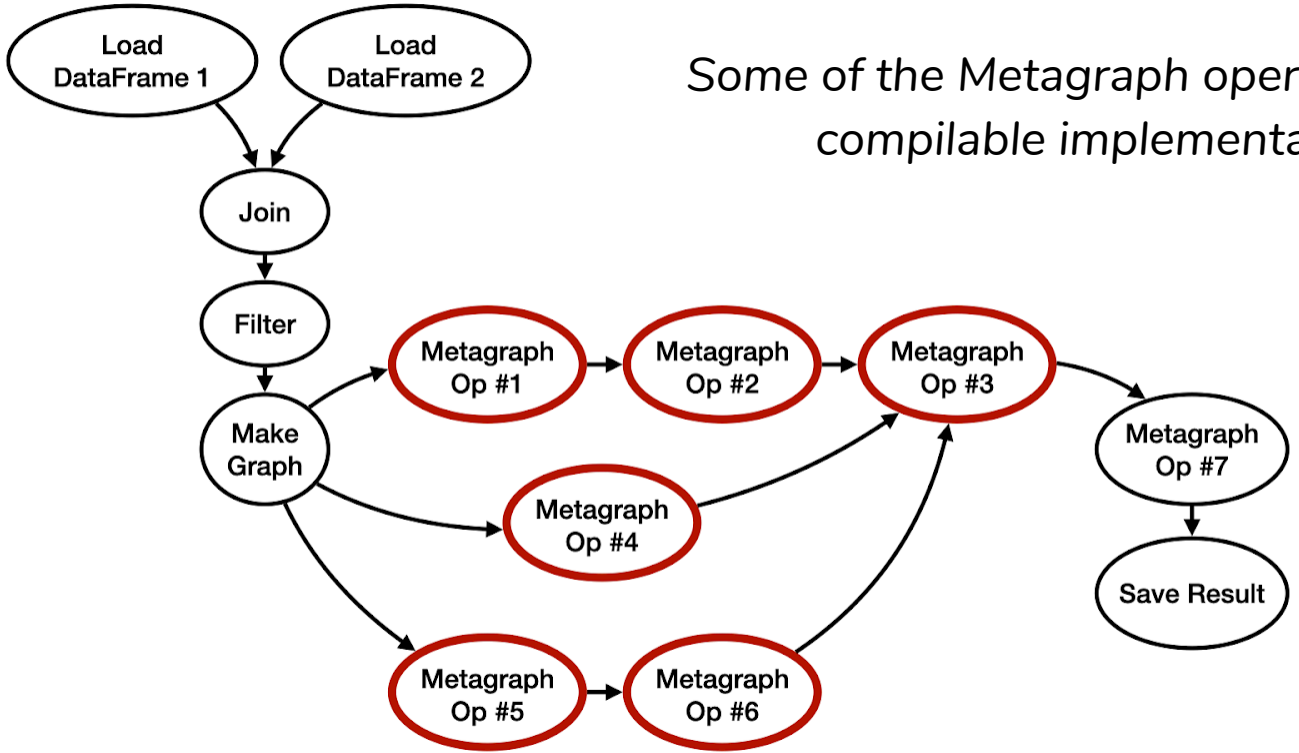
Compiling DAGs

Compiling DAGs



Hypothetical DAG

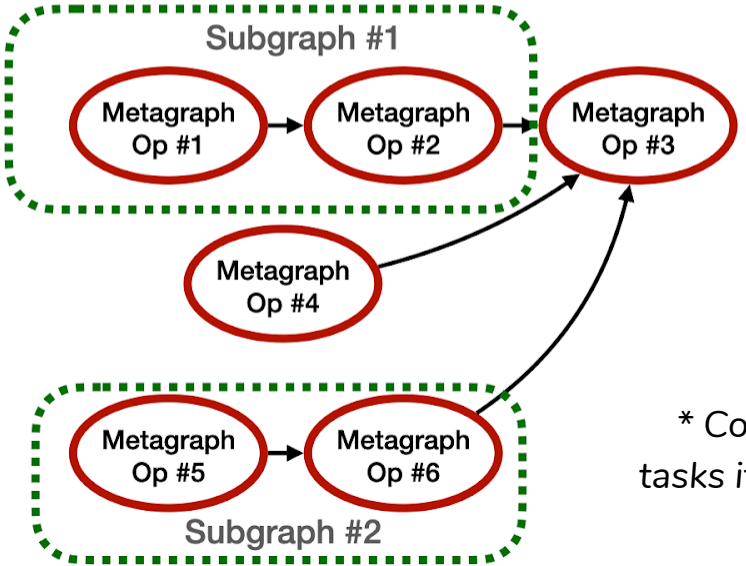
Compiling DAGs



Some of the Metagraph operations have compilable implementations

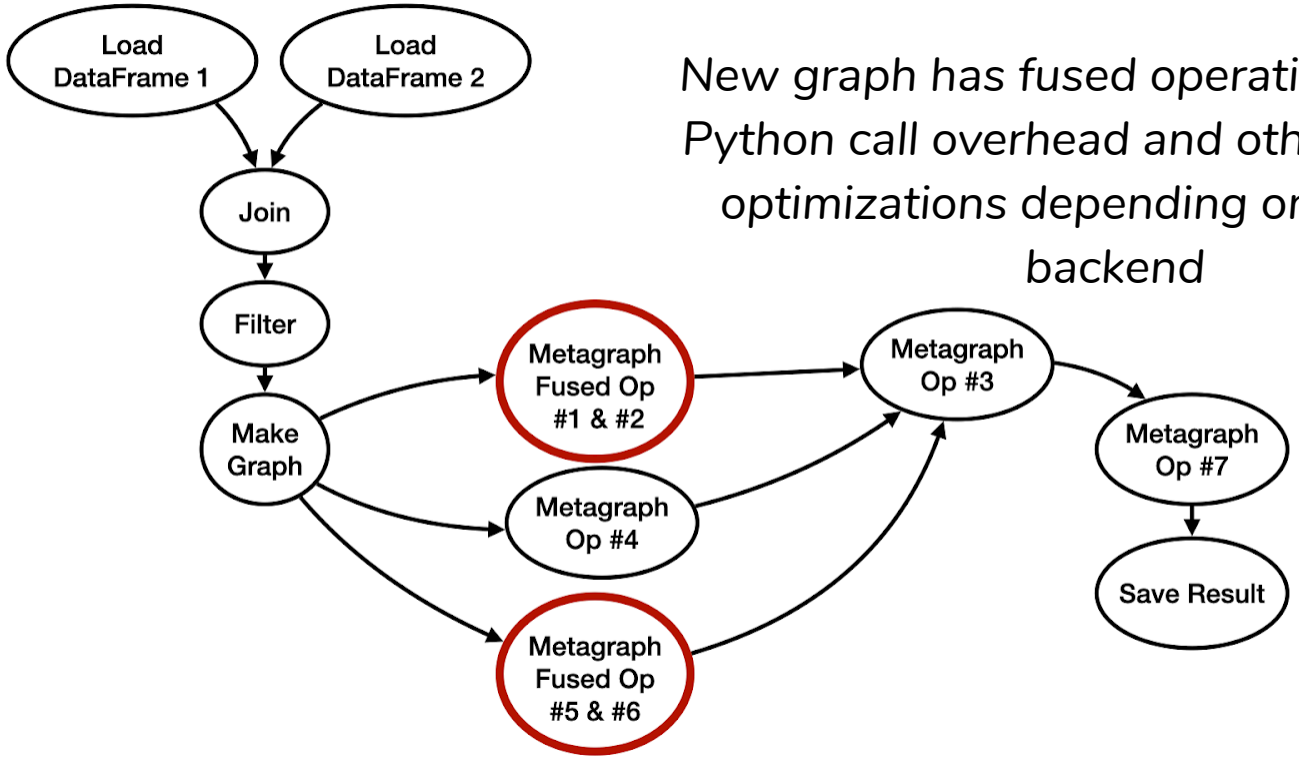
Compiling DAGs

*Extract the subgraphs which form linear chains and will not reduce overall parallelism**



** Could consider fusing parallel tasks if targeting parallel hardware?*

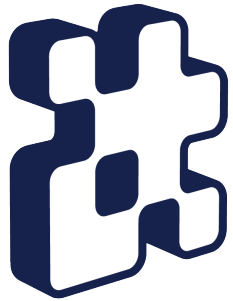
Compiling DAGs



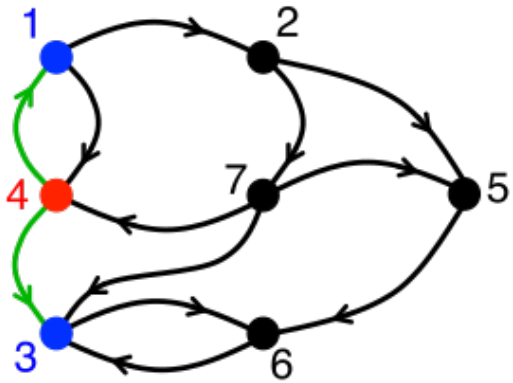
New graph has fused operations with no Python call overhead and other potential optimizations depending on compiler backend

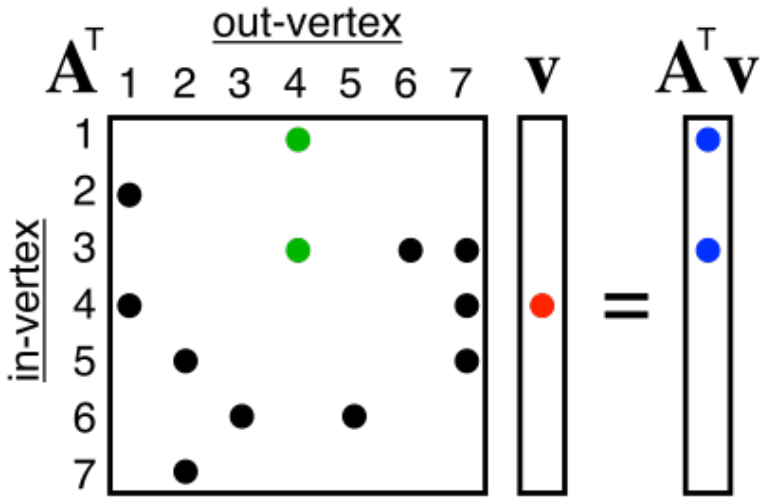
GraphBLAS & Python

GraphBLAS



- A set of graph primitives defined in the language of linear algebra





https://graphblas.org/

Basics of GraphBLAS

- Sparse matrix / vector operations with optional masking and update / accumulation in-place
  - Matrix * Matrix, Matrix * Vector, Vector * Vector
  - Elementwise binary operations, and unary apply operation
  - Reductions
  - Extract ranges from matrices/vectors and assign into matrices/vectors
- Missing data is not the same as zero!
- Linear algebra with semirings
  - $\oplus$: Monoid binary operator & identity (*associative* & *commutative*)
  - $\otimes$: Binary operator

grblas: a Python interpretation of GraphBLAS

- Code: https://github.com/metagraph-dev/grblas
- Install: **pip install grblas** or **conda install -c conda-forge grblas**
- Pure Python
  - We worked with Michel Pelletier to create **python-suitesparse-graphblas**
  - Now **grblas** and **pygraphblas** both share the same base and are always compatible
- Currently wraps SuiteSparse:GraphBLAS
  - But we plan to support others
- Apache 2 license
- 26 months old
- Thoroughly tested (~100% code coverage)
- Authors: Erik Welch and Jim Kitchen

- Emphasis on usability, interactivity, and good error messages

- Many new features, enhancements, and improved syntax in the last year

- We expect the API to be stable going forward

- See full changelog here: https://github.com/metagraph-dev/grblas/releases

A “High Level” API? What does that look like?

Let’s dive into the anatomy of a GraphBLAS call

C

```
GrB_mxm(C, M, GrB_MIN_FP64, GrB_MIN_PLUS_SEMIRING_FP64, A, B, GrB_DESC_RST0T1);
```

Python

```
C(mask=M.S, replace=True, accum=op.min) << semiring.min_plus(A.T @ B.T)
```

Python (alt)

```
C(M.S, replace, op.min) << A.T.mxm(B.T, semiring.min_plus)
```

Assign and sub-assign examples

```
C(M.V)[rows, cols] << E.T
```

```
GrB_Matrix_assign(C, M, NULL, E, (GrB_Index[]){0, 1, 2, 3}, 4,
                  (GrB_Index[]){0, 1, 2, 3}, 4, GrB_DESC_T0)
```

```
C[rows, cols](E.S) << 1
```

```
GxB_Matrix_subassign_INT64(C, E, NULL, 1, (GrB_Index[]){0, 1, 2, 3}, 4,
                           (GrB_Index[]){0, 1, 2, 3}, 4, GrB_DESC_S);
```

A “High Level” API? What does that look like?

grblas decomposes a monolithic C call into a composition of expressions

- grblas is like the GraphBLAS math notation
  - But instead of pseudo-code, it is actual code!
  - We construct expressions **lazily** to avoid **expensive** and **unnecessary** computations
- ***Anything*** expressible in GraphBLAS C can be expressed ***naturally*** in grblas
- One should *never* need to specify descriptors explicitly
  - A.T for transpose; A.S and ~A.V for structural and complemented value mask descriptors
- grblas is a more natural way to read, write, learn, and ***think*** about GraphBLAS
  - Because math notation is much more natural than the C API
  - ***No confusion*** about how to translate between C API and grblas (both ways!)
  - With the Recorder, grblas can emit the equivalent C calls!

```
with Recorder() as rec:
    C(M.S)[0, cols] << w
```

```
GrB_Matrix_assign(C, M, NULL, (GrB_Matrix)w, (GrB_Index[]){0}, 1,
                  (GrB_Index[]){0, 1, 2, 3}, 4, GrB_DESC_ST0);
```

Example: SSSP

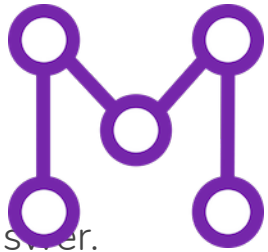
```
# matrix A is the graph adjacency matrix
v = Vector.new(m.dtype, m.nrows)
v[1] << 0  # initialize starting vertex (1) with distance of zero
v_dup = gb.Vector.new(v.dtype, size=N)
i = 0
while True:
    i += 1
    v_dup << v
    v(op.min) << A.mxv(v, op.min_plus)
    if v.isequal(v_dup):
        break
```


Plans and Final Thoughts

Next Steps

- Metagraph:
  - Can we upstream some of concepts into Dask itself?
  - Other packages that need plugins?
  - Look for more user feedback
- Grblas
  - Promote GraphBLAS more in Python communities
  - File formats and I/O?

Conclusions



- Graphs are a hard problem! We must design assuming we don't have a complete answer.
- The graph community includes several difference constituencies:
  - **Users:**
    - Looking for integration and flexibility to solve complete workflows
  - **Algorithm developers:**
    - Want to focus on the performance of specific algorithms without solving for every user concern
  - **Optimizers and hardware vendors:**
    - Need hooks to swap out individual or sets of operations
- All the projects described here are open source:
  - **Metagraph:** https://metagraph.readthedocs.io/
  - **grblas;** https://github.com/metagraph-dev/grblas