Extended Sparse Matrices as Tools for Graph Computation

Adam Lugowski

UCSB

SIAM Annual Meeting July 11, 2012

Knowledge

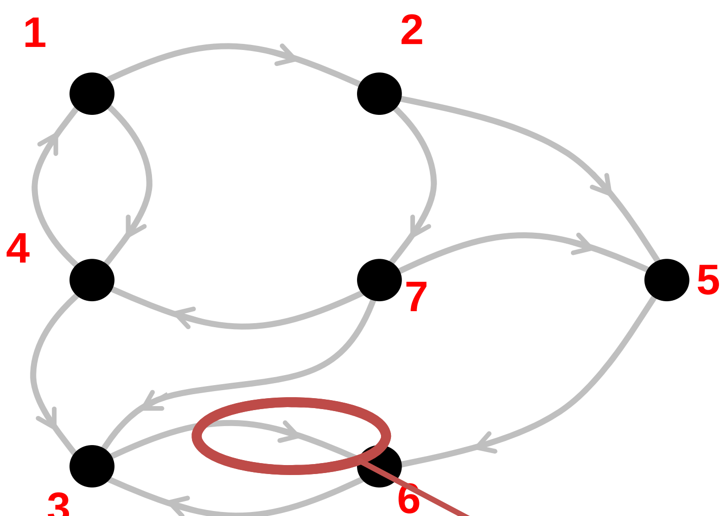
Discovery

Toolbox

kdt.sourceforge.net

KDT Graphs: distributed sparse matrices

G



Edge attributes can be arbitrary objects

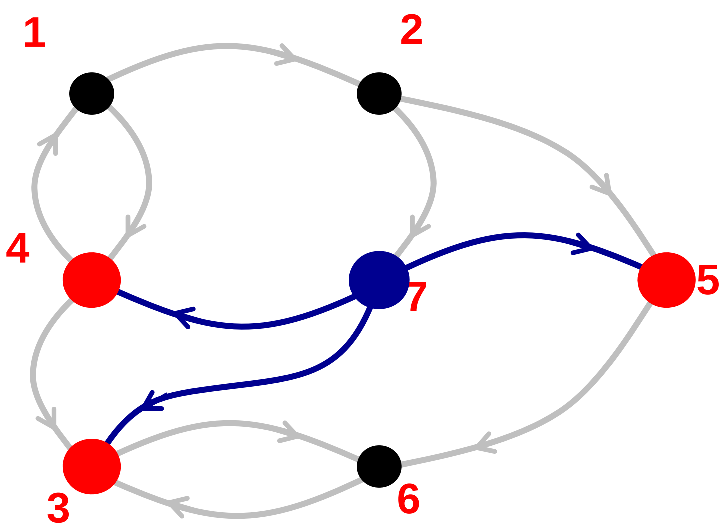
G

			1			
1						
			1		1	1
1						1
	1					1
		1		1		
	1					

Transposed Adjacency Matrix: sparse structure distributed in 2D layout

Graph Traversals are M×M or M×V

User-defined semirings on user-defined objects



$$\begin{array}{ccc} G & f_{in} & f_{out} \\ \begin{bmatrix} & & & 1 & & & \\ 1 & & & & & & \\ & & & 1 & & 1 & 1 \\ 1 & & & & & & 1 \\ & 1 & & & & & 1 \\ & & 1 & & 1 & & \\ & 1 & & & & & \end{bmatrix} & \times \begin{bmatrix} \\ \\ \\ \\ \\ \\ 7 \end{bmatrix} & = \begin{bmatrix} \\ \\ 7 \\ 7 \\ 7 \\ \\ \\ \end{bmatrix} \end{array}$$

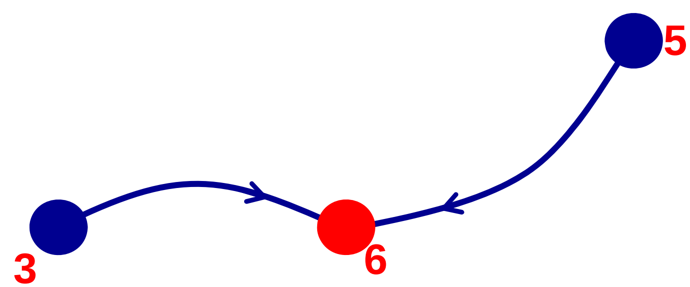
distance 1 from vertex 7

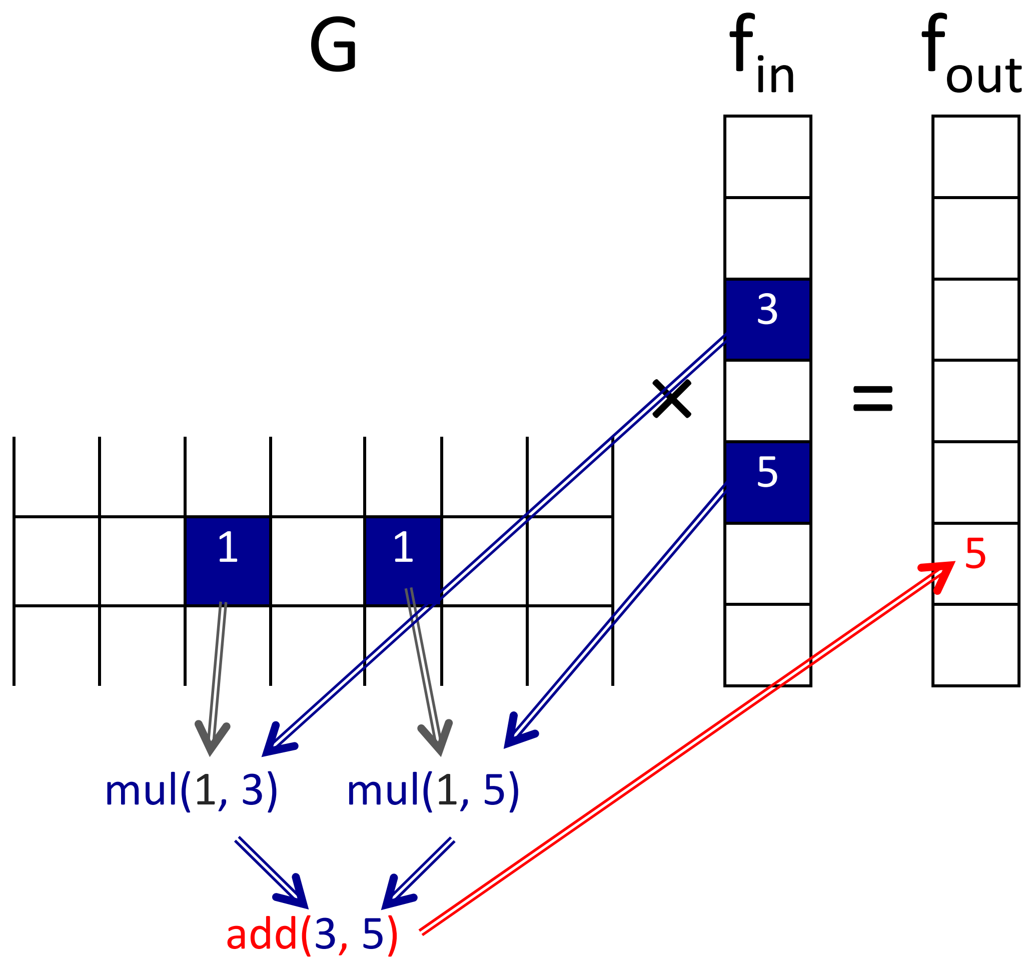
Algorithm logic in custom semirings

Semiring:

```
def mul(x, y):
    return y
```
traverse outgoing edges

```
def add(x, y):
    return y
```
choose among incoming edges





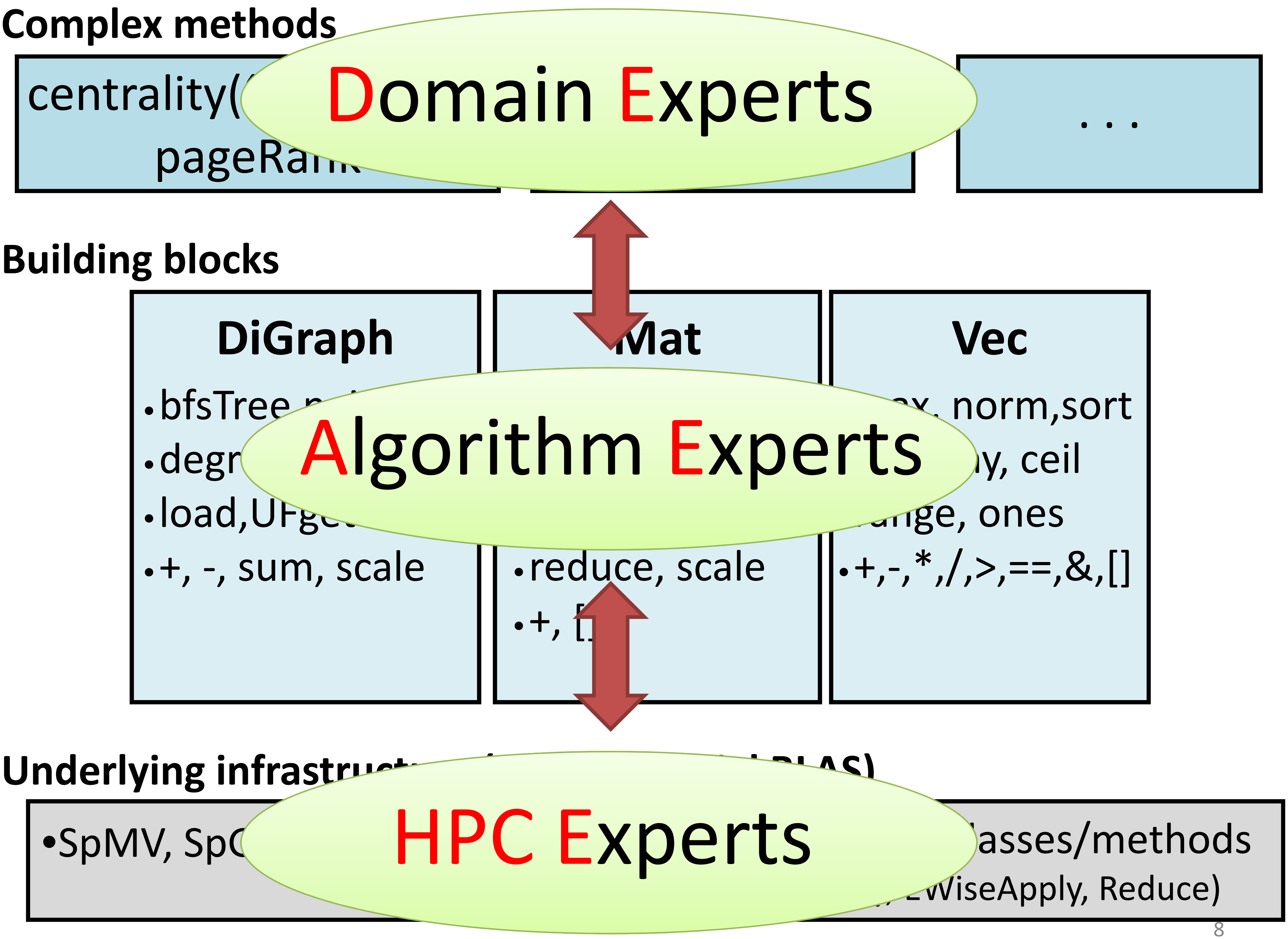
Sparse Matrix Operations

•Matrix-Matrix multiplication •Matrix-Vector multiplication •Element-Wise (eg. A .* B) •Scale by Vector	•Apply •Reduce •Prune •Find •Load/Save •Construct •Generate

All customizable with user-defined callbacks

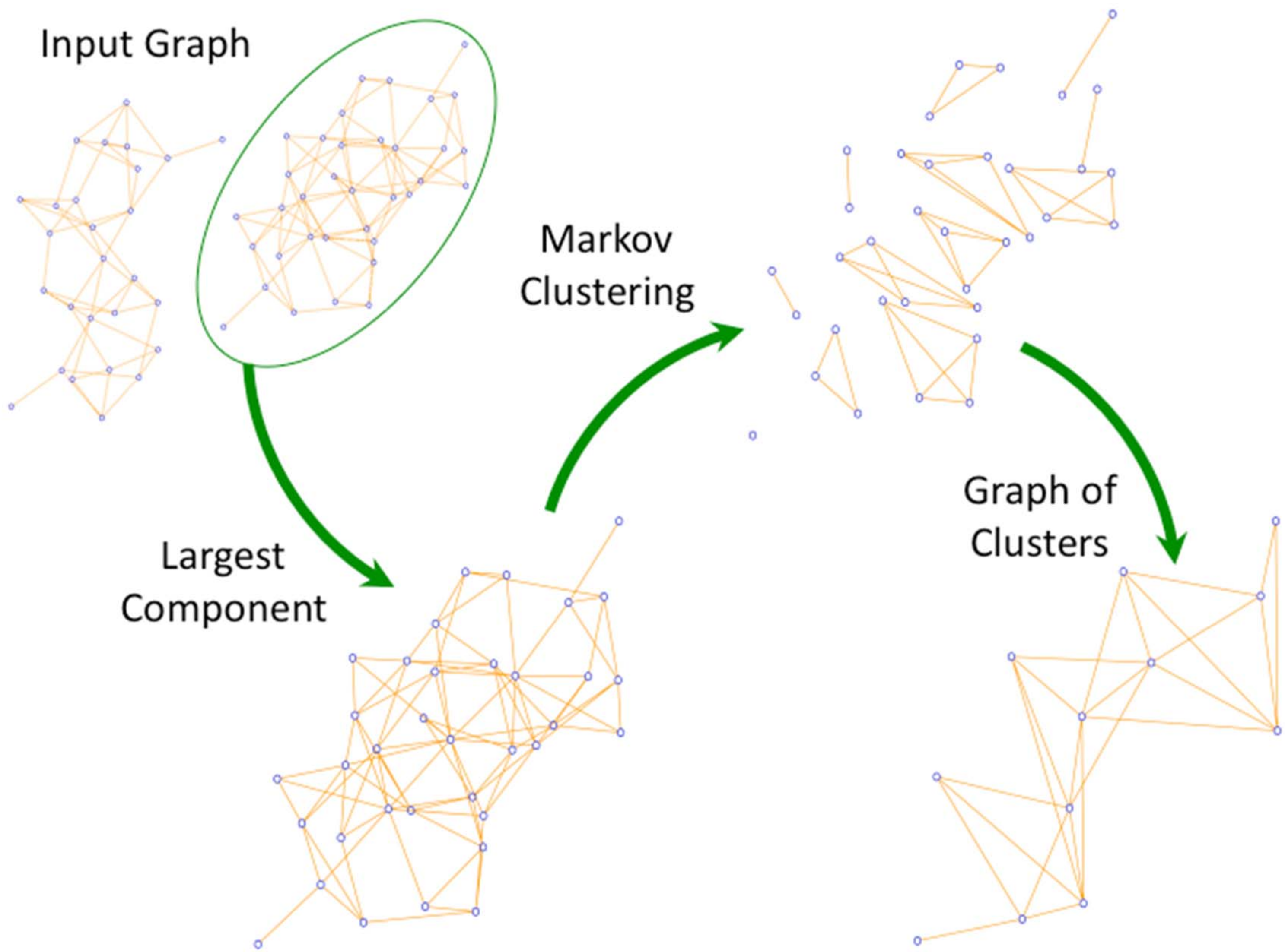
Why (sparse) adjacency matrices?

Traditional graph computations	Graphs in the language of linear algebra
Data driven, unpredictable communication	Fixed communication patterns
Irregular and unstructured, poor locality of reference	Operations on matrix blocks exploit memory hierarchy
Fine grained data accesses, dominated by latency	Coarse grained parallelism, bandwidth limited



Complex methods
centrality(
pageRan
. . .
Domain Experts
Building blocks
DiGraph
•bfsTree
•degr
•load,UFge
•+, -, sum, scale
Mat
•reduce, scale
•+, [
Vec
ax, norm,sort
y, ceil
nge, ones
•+,-,*,/,>,==,&,[]
Algorithm Experts
Underlying infrastruc
BLAS)
•SpMV, SpG
lasses/methods
WiseApply, Reduce)
HPC Experts

Example workflow

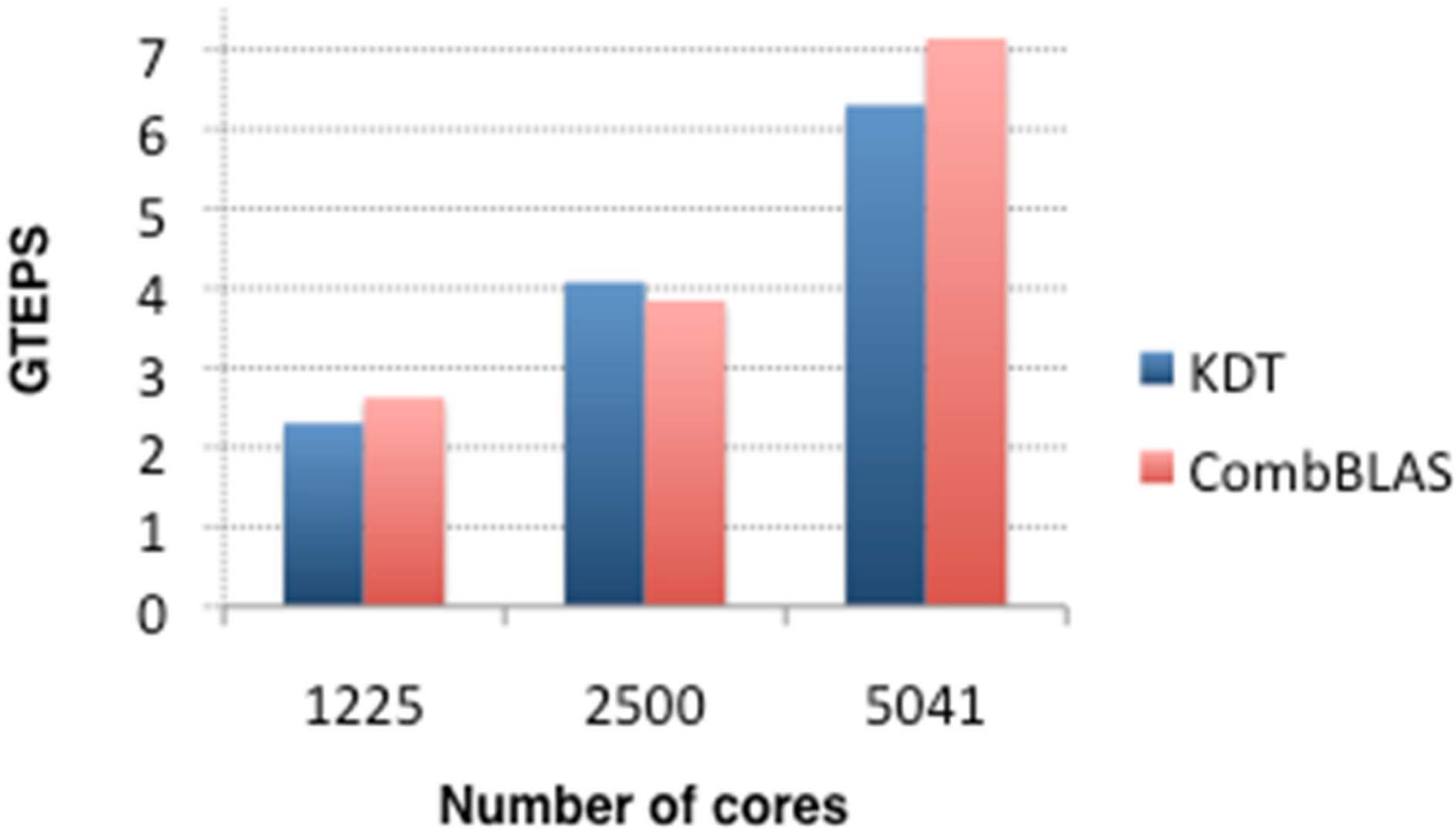


Example workflow KDT code

```
# the variable bigG contains the input graph
# find and select the giant component
comp = bigG.connComp()
giantComp = comp.hist().argmax()
G = bigG.subgraph(comp==giantComp)

# cluster the graph
clus = G.cluster('Markov')

# contract the clusters
smallG = G.contract(clus)
```

BFS on a Scale 29 RMAT graph

(500M vertices, 8B edges)



Machine: NERSC's Hopper

Ongoing work: High-performance Python

1. Speed up Python callbacks

1. Introducing runtime-defined types

Python is great at high-level operations, slow at inner loops.

The way to make Python fast is to not use Python.

-- Me

SEJITS (A. Fox and S. Kamil)

- Selective Embedded Just-In-Time Specialization
  1. Take Python code
  2. Translate it to equivalent C++ code
  3. Compile with GCC
  4. Call compiled version instead of Python version

https://github.com/shoaibkamil/asp/wiki/

SEJITS: Speeding up Python with C++

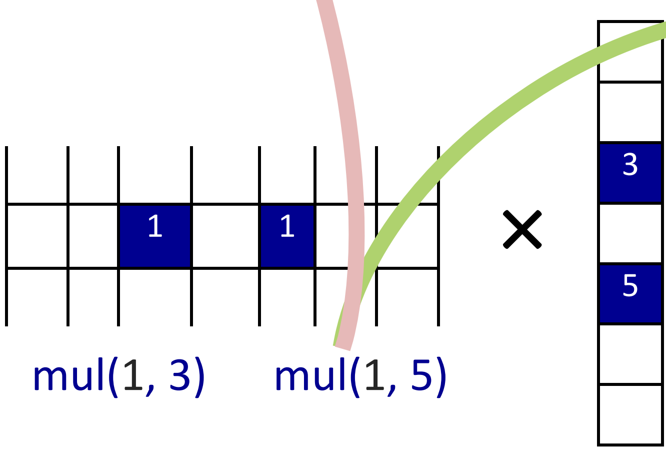
SEJITS converts
Python routine to C++

```python
def mul(x, y):
    return y
```

```cpp
double mul(const Obj2& arg1, double arg2)
{
    return arg2;
}
```

Compiles it (gcc) at runtime.

mul.o

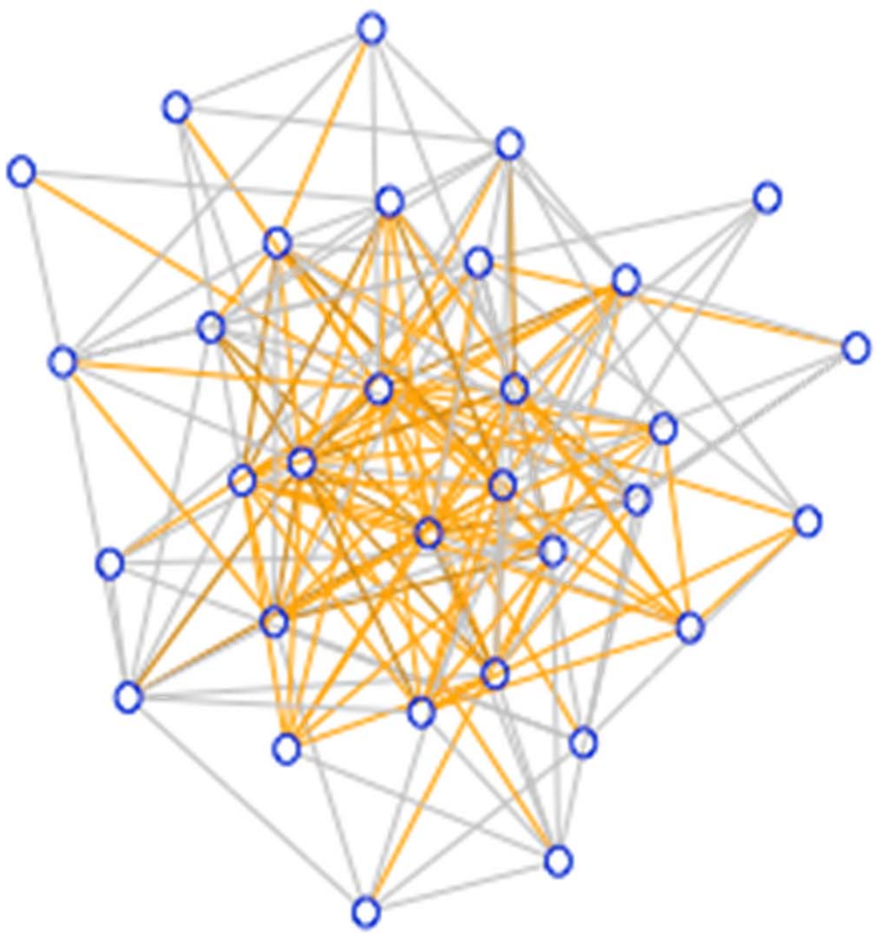


Compiled C++ routine called
instead of Python

SEJITS Integration into KDT: Filtering

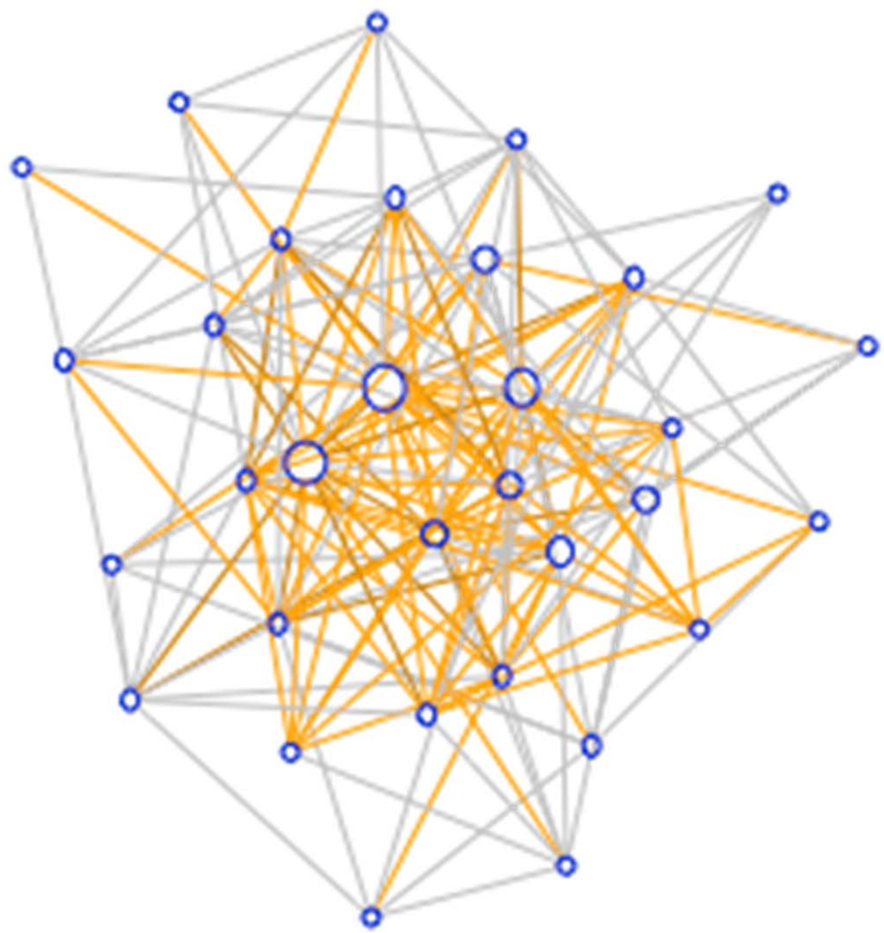
A filter is a predicate (Python function) which returns True if an edge is to be kept, False otherwise.

texts and phone calls



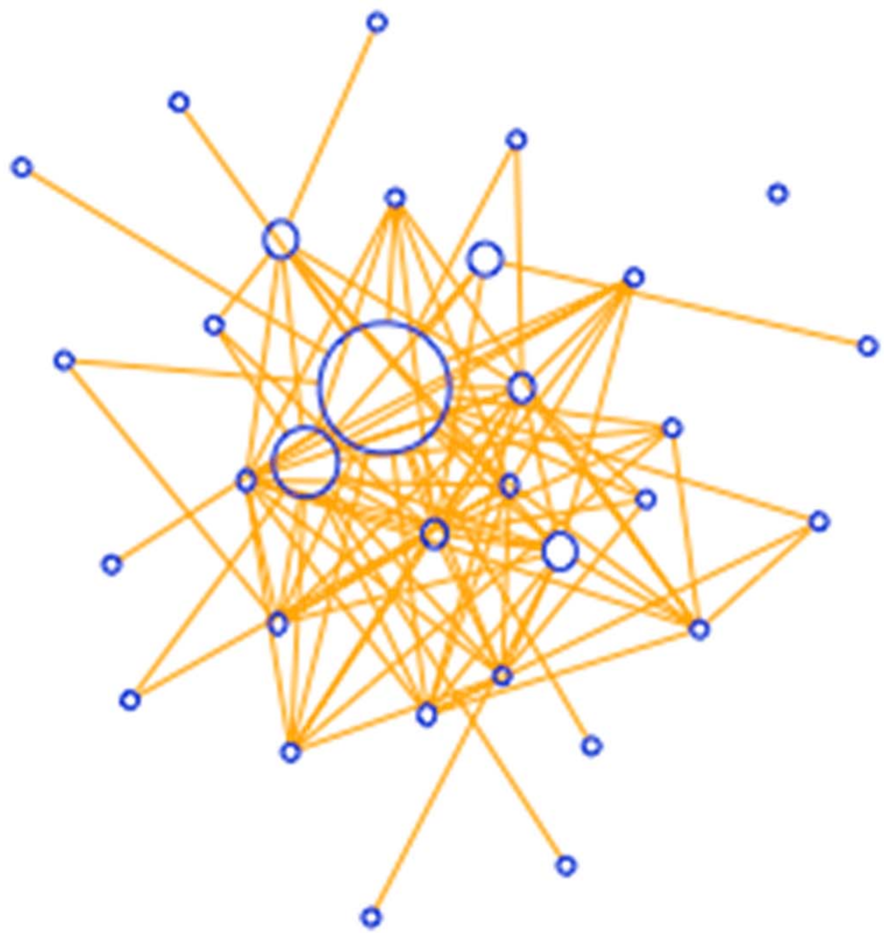
```
# draw graph
draw(G)

# Each edge has this attribute:
class edge_attr:
   isText
   isPhoneCall
   weight
```

Betweenness Centrality



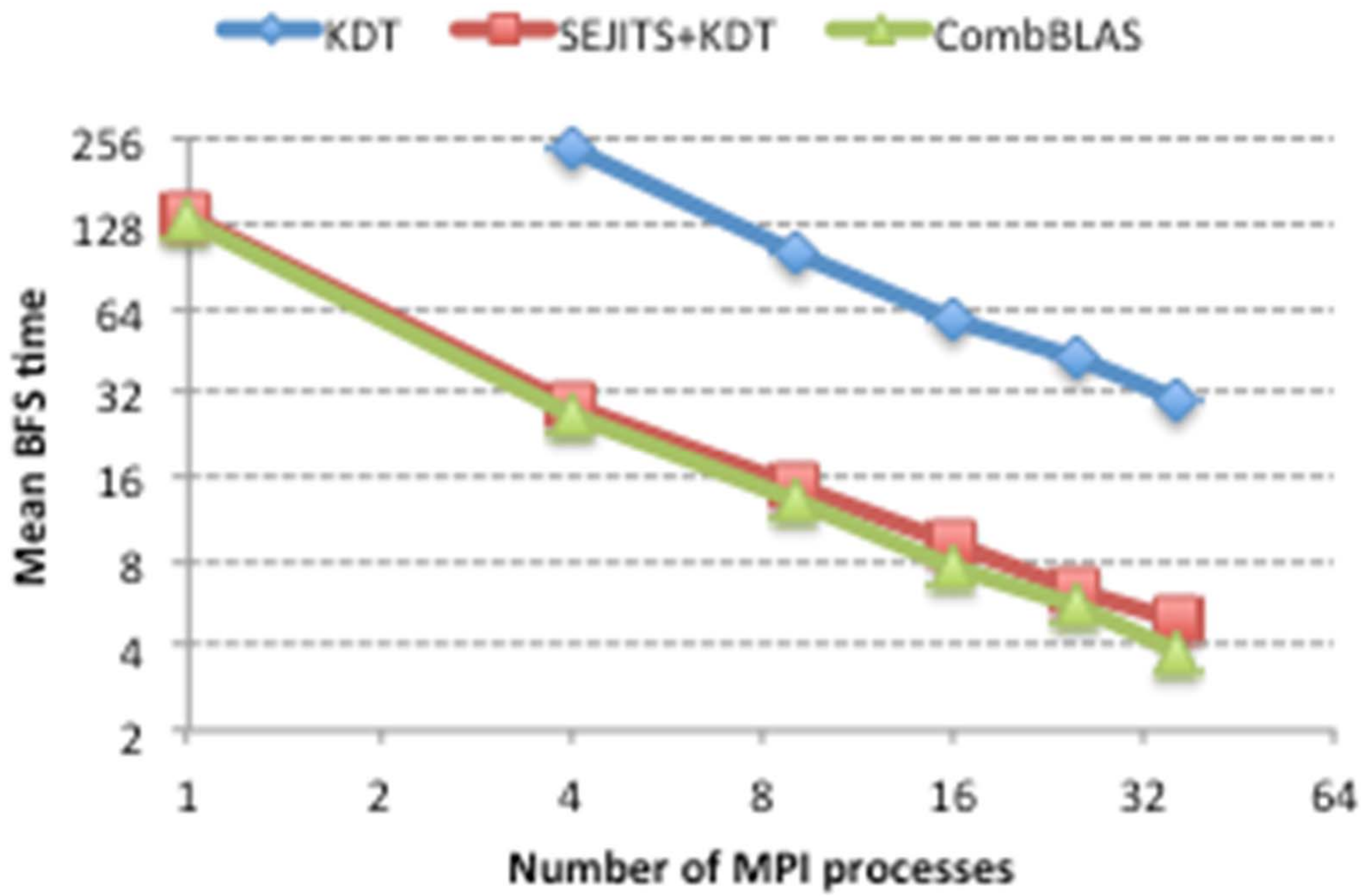
```
bc = G.centrality("approxBC")
# draw graph with node sizes
# proportional to BC score
draw(G, bc)
```

Betweenness Centrality on texts



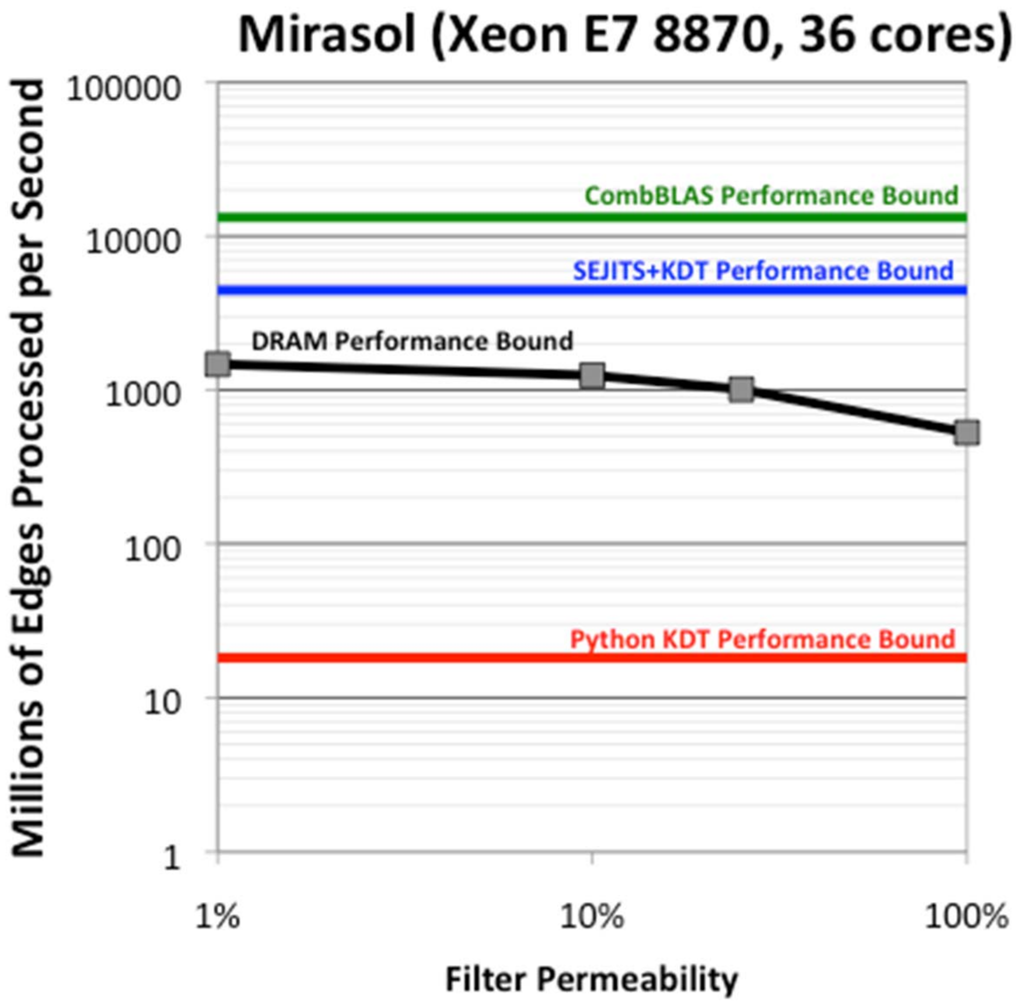
```
# BC only on text edges
G.addEFilter(
      lambda e: e.isText)
bc = G.centrality("approxBC")
# draw graph with node sizes
# proportional to BC score
draw(G, bc)
```

SEJITS brings performance back



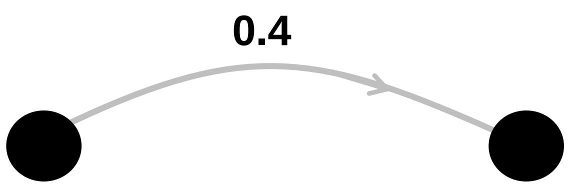
Time (in seconds) for a single BFS iteration on Scale 23 RMAT (8M vertices, 130M edges) with 10% of elements passing filter. Machine is Mirasol.

Roofline analysis: why this works

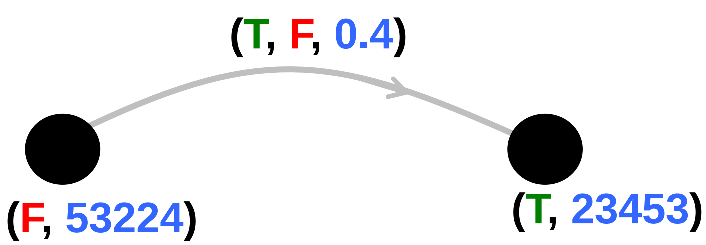


Attributes

“Graph”



“Weighted Graph”



“Semantic Graph”

Extended Attribute Support

- Completely remove user-written C++ code
  - User friendliness, allows systemwide installs
- adds flexibility
  - remove limitations on number of types allowed
  - remove limitation on assumption of what an object is
  - allows definition of well-formatted datafiles

Extended Attribute Support

- Requirements:
  - Type defined in Python
    - Fixed-size
  - Memory allocated in C++, object used in Python
  - Be able to operate on Python-defined structure through C++
    - `For SEJTIS`

Regular Python objects too general

Extended Attribute Support

- Inspiration from ctypes.Structure:

```
class MyEdge(Structure):
    _fields_ = [("weight", c_double),
                ("isPhoneCall", c_bool),
                ("isText", c_bool)]
```

Acts like Python, C++ friendly

Python:

```
e = MyEdge()
e.weight = 10
```

But also have:

- sizeof, addressof, offset, type
- placement new

Can generate translations at runtime,
performance equivalent to compile time-defined structs

Conclusion

- KDT is a high-performance graph analysis toolkit written for a high-productivity language
- Possible to write callbacks in high-level language while retaining low-level language performance
- Possible to define datatypes at runtime

Thank You

kdt.sourceforge.net