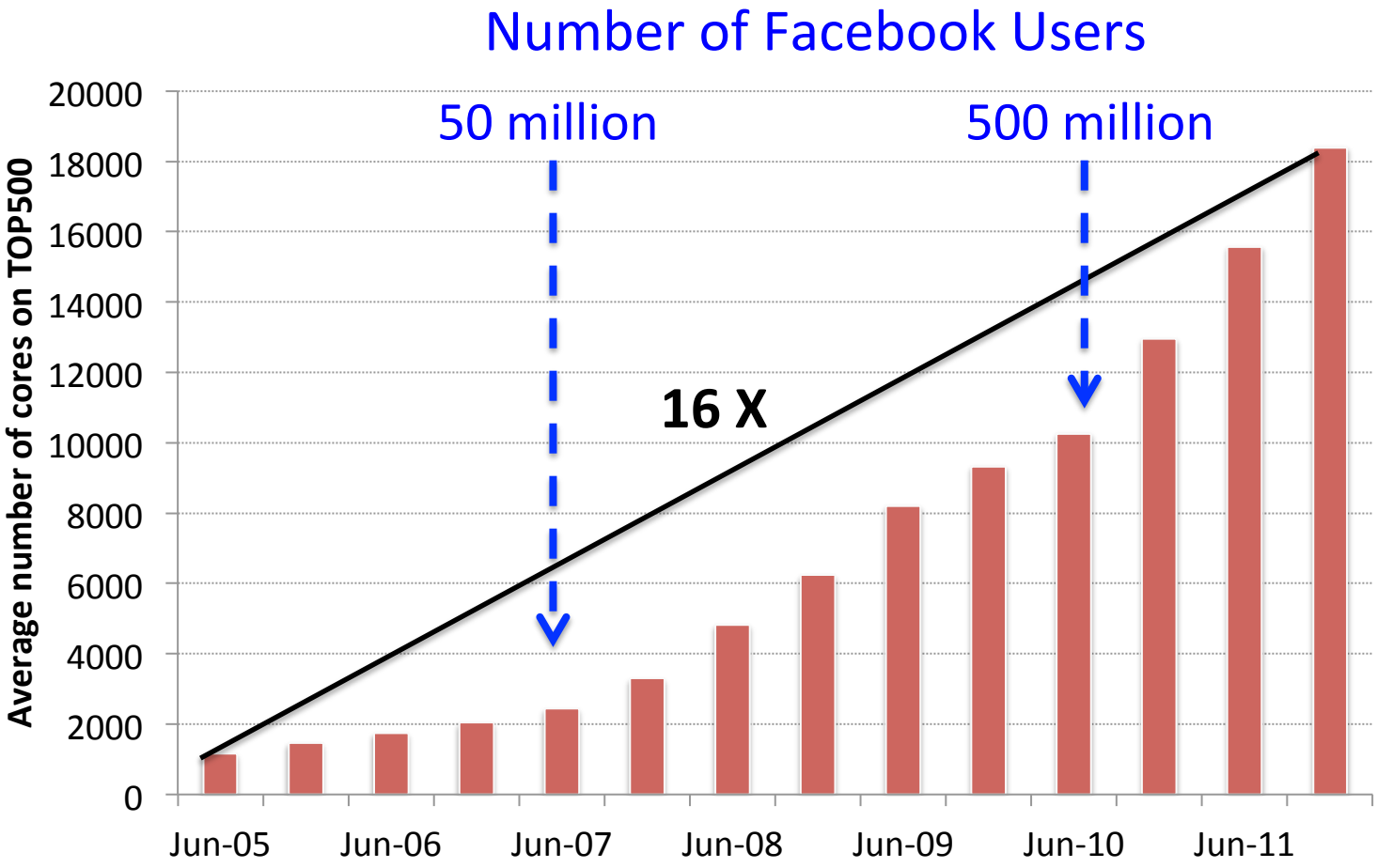
A sustainable software stack for parallel graph analysis

Aydın Buluç
Lawrence Berkeley National Laboratory
July 11, 2012

Trends in parallelism and data



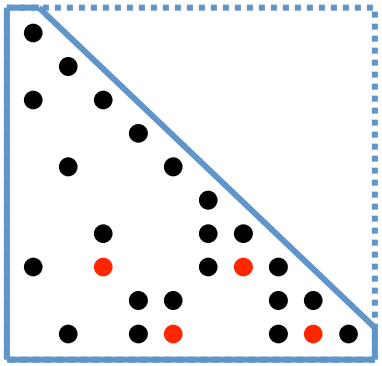
More cores and data → Need to extract <u>algorithmic</u> parallelism

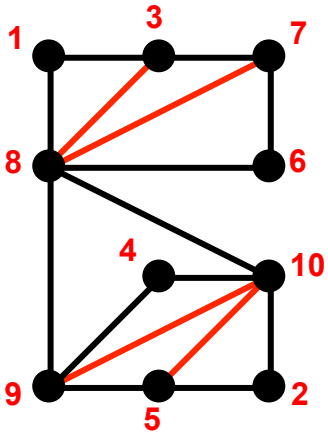
Graph abstractions in Computer Science

Compiler optimization:
Control flow graph : graph dominators
Register allocation: graph coloring

Scientific computing:
Preconditioning: support graphs, spanning trees
Sparse direct solvers: chordal graphs
Parallel computing: graph separators

Computer Networks:
Routing: shortest path algorithms
Web crawling: graph traversal
Interconnect design: Cayley graphs



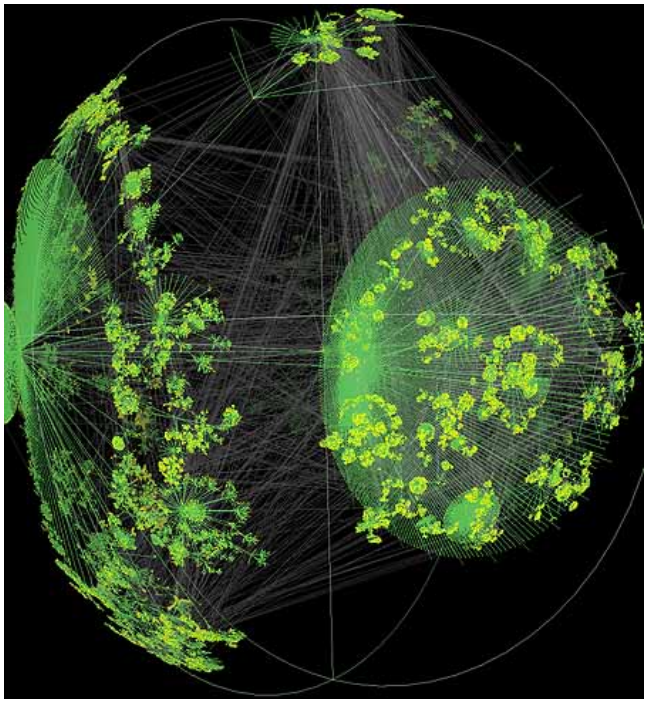


$G^+(A)$
[chordal]

Large graphs are everywhere

Internet structure

Social interactions

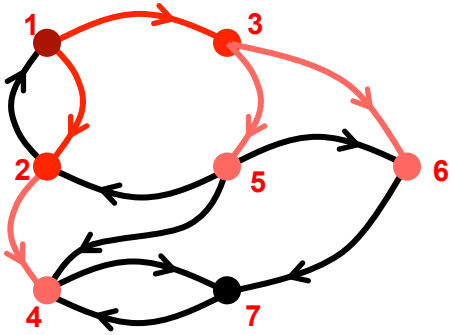


WWW snapshot, courtesy Y. Hyun

Scientific datasets: biological, chemical, cosmological, ecological, …

Yeast protein interaction network, courtesy H. Jeong

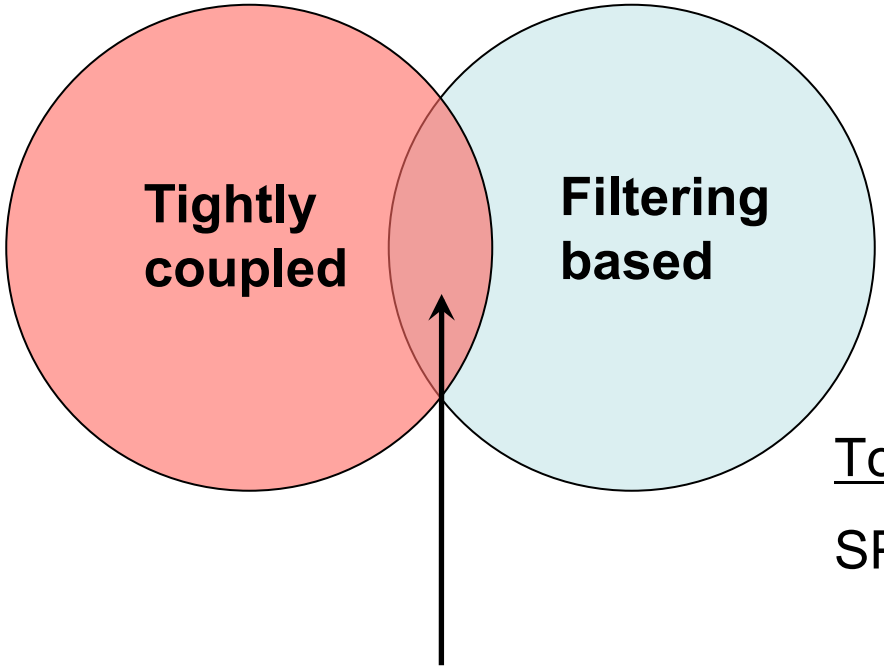
Types of graph computations



Tool: Graph Traversal

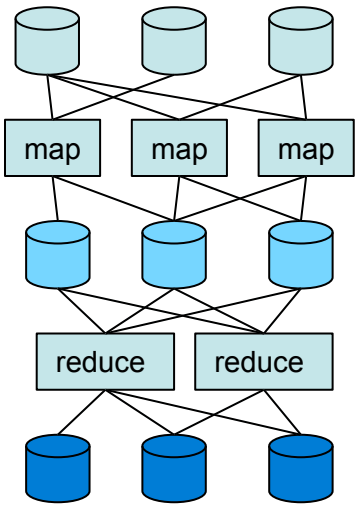
Examples:

- Centrality
- Shortest paths
- Network flows
- Strongly Connected Components



Fuzzy intersection

Examples: Clustering, Algebraic Multigrid

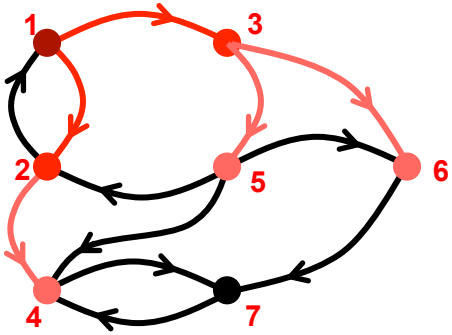


Tools: Map/Reduce, SPARQL engines

Examples:

- Loop and multi edge removal
- Triangle/Rectangle enumeration

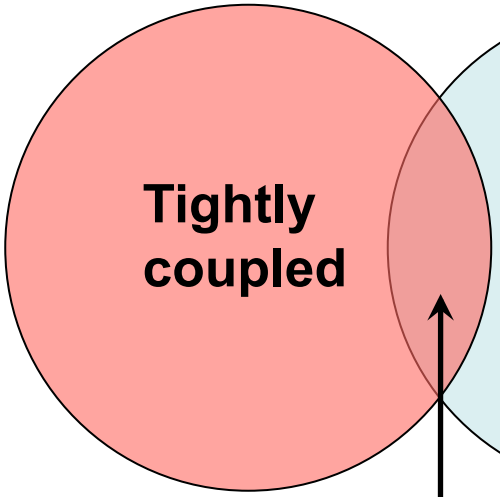
Types of graph computations



Tool: Graph Traversal

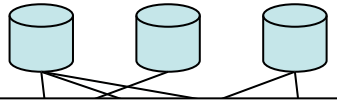
Examples:

- Centrality
- Shortest paths
- Network flows
- Strongly Connected Components



Fuzzy intersection

Examples: Clustering, Algebraic Multigrid



The left diagram is the focus of this talk.

Challenges :

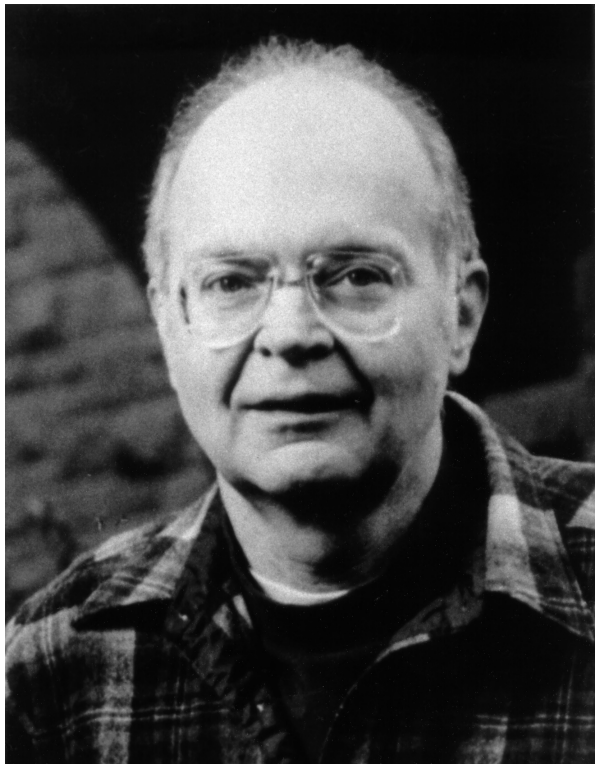
- Difficult to parallelize
- Very low arithmetic intensity
- Unpredictable data access patterns

Examples:

- Loop and multi edge removal
- Triangle/Rectangle enumeration

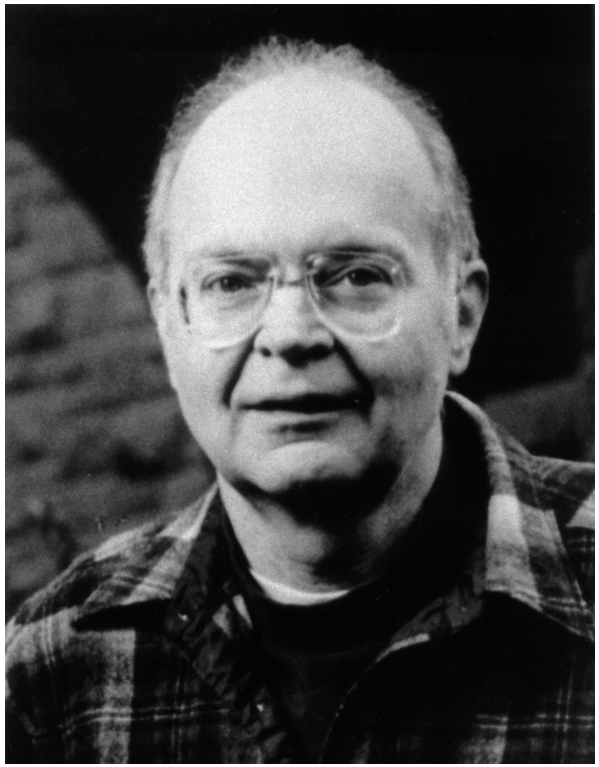
Software for Graph Computation

"...my main conclusion after spending ten years of my life on the TeX project is that software is hard. It's harder than anything else I've ever had to do"



Software for Graph Computation

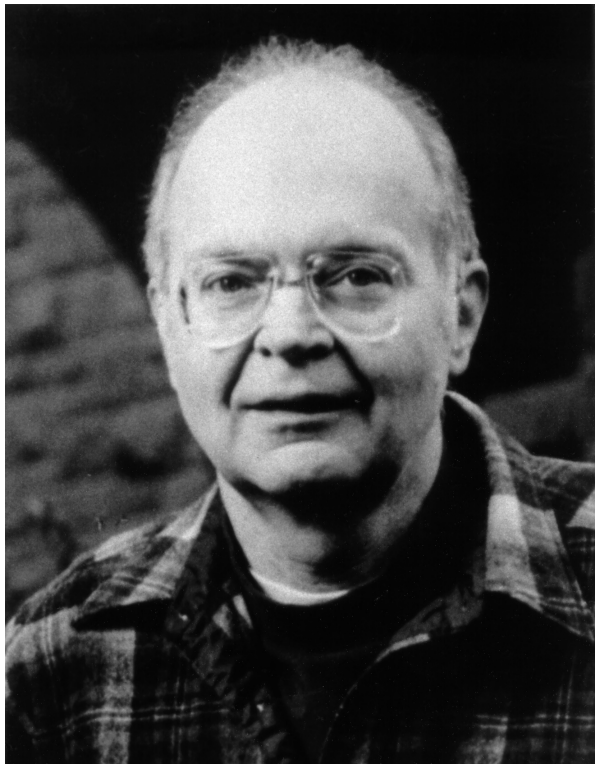
"...my main conclusion after spending ten years of my life on the TeX project is that software is hard. It's harder than anything else I've ever had to do"



Dealing with software is hard !

Software for Graph Computation

"...my main conclusion after spending ten years of my life on the TeX project is that software is hard. It's harder than anything else I've ever had to do"

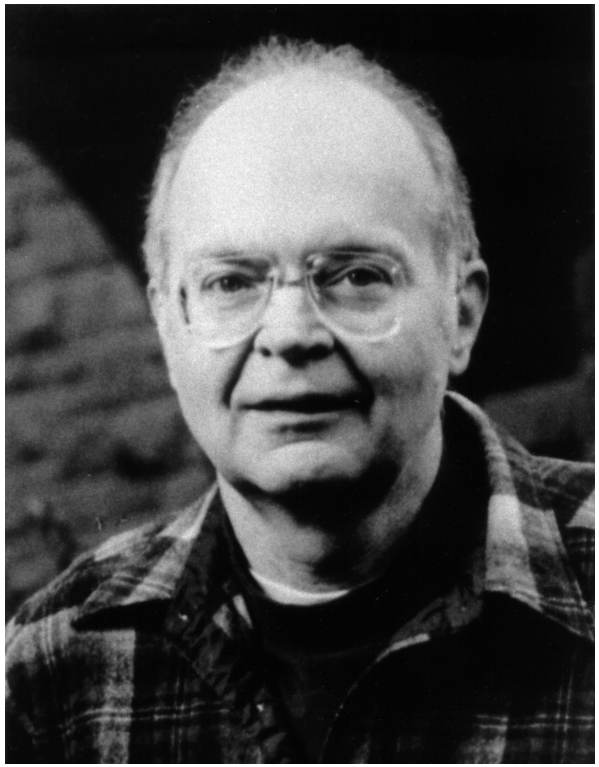


Dealing with software is hard !

High performance computing (HPC) software is harder !

Software for Graph Computation

"...my main conclusion after spending ten years of my life on the TeX project is that software is hard. It's harder than anything else I've ever had to do"



Dealing with software is hard !

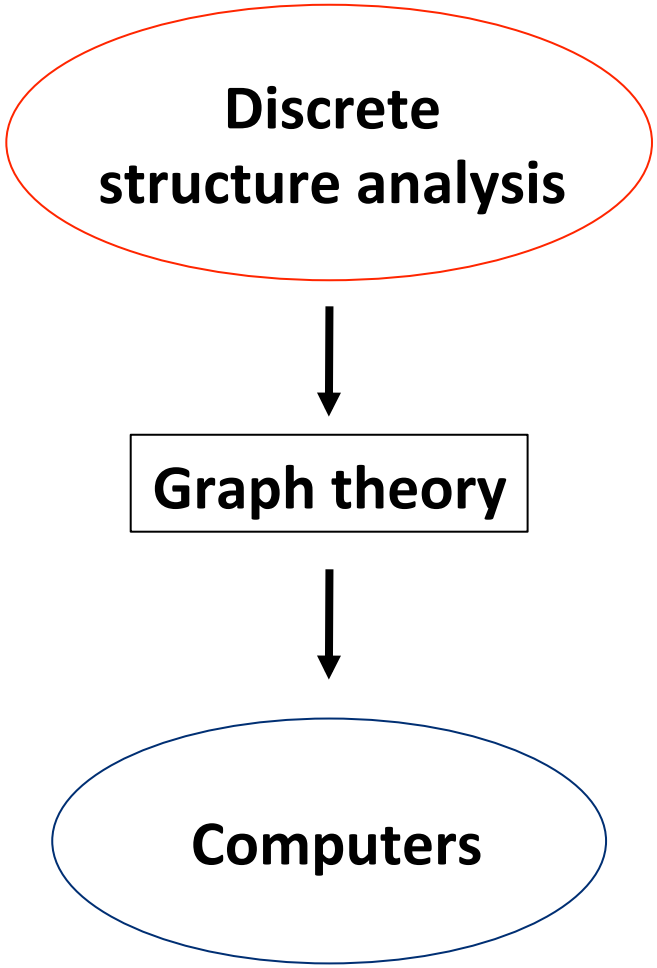
High performance computing (HPC) software is harder !

Deal with parallel HPC software?

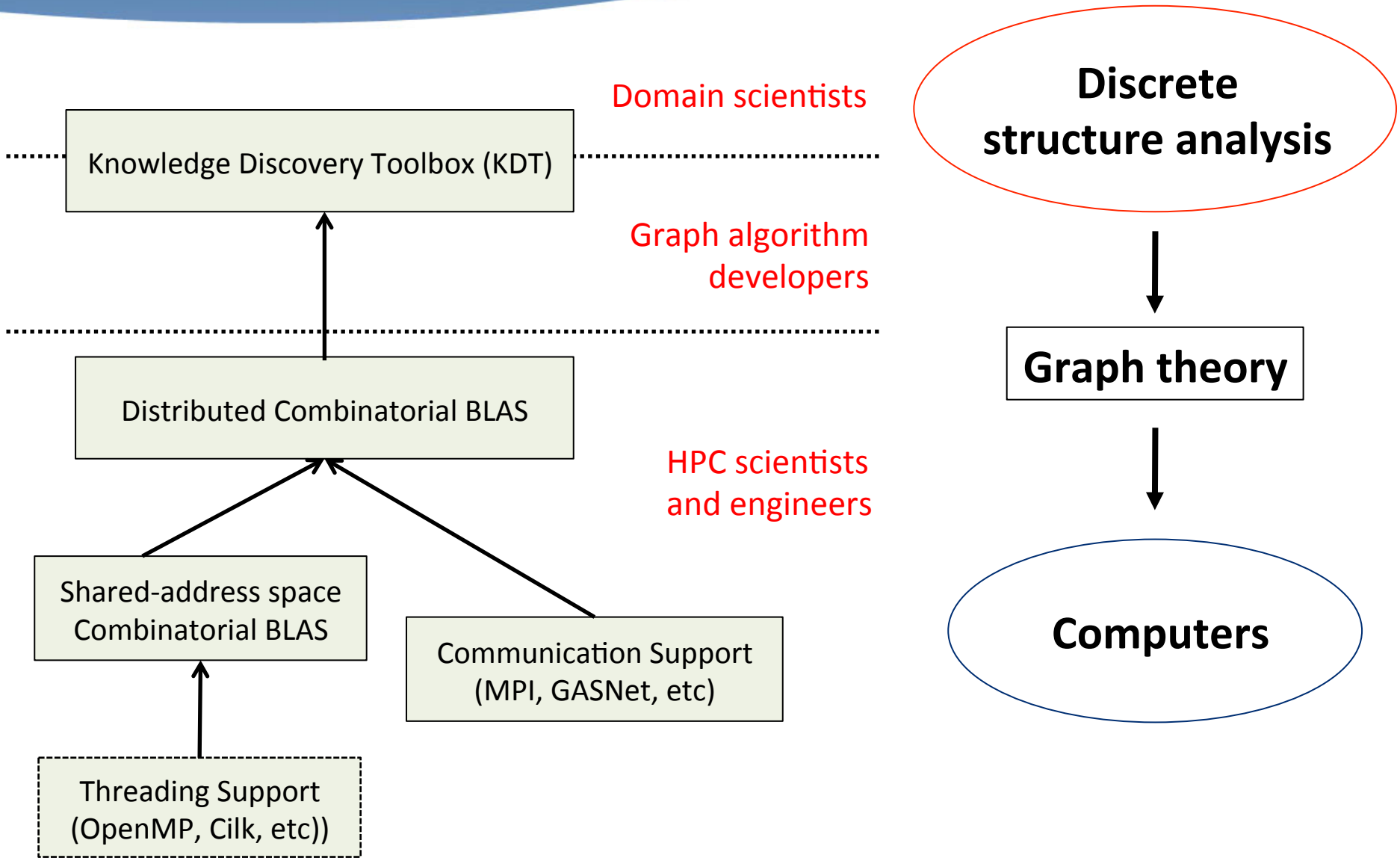
Outline

- **Sustainable graph libraries**
- Linear algebraic primitives for graphs
- Sparse triple product for graph subroutines
- Filtered semantic graphs
- Future challenges/directions

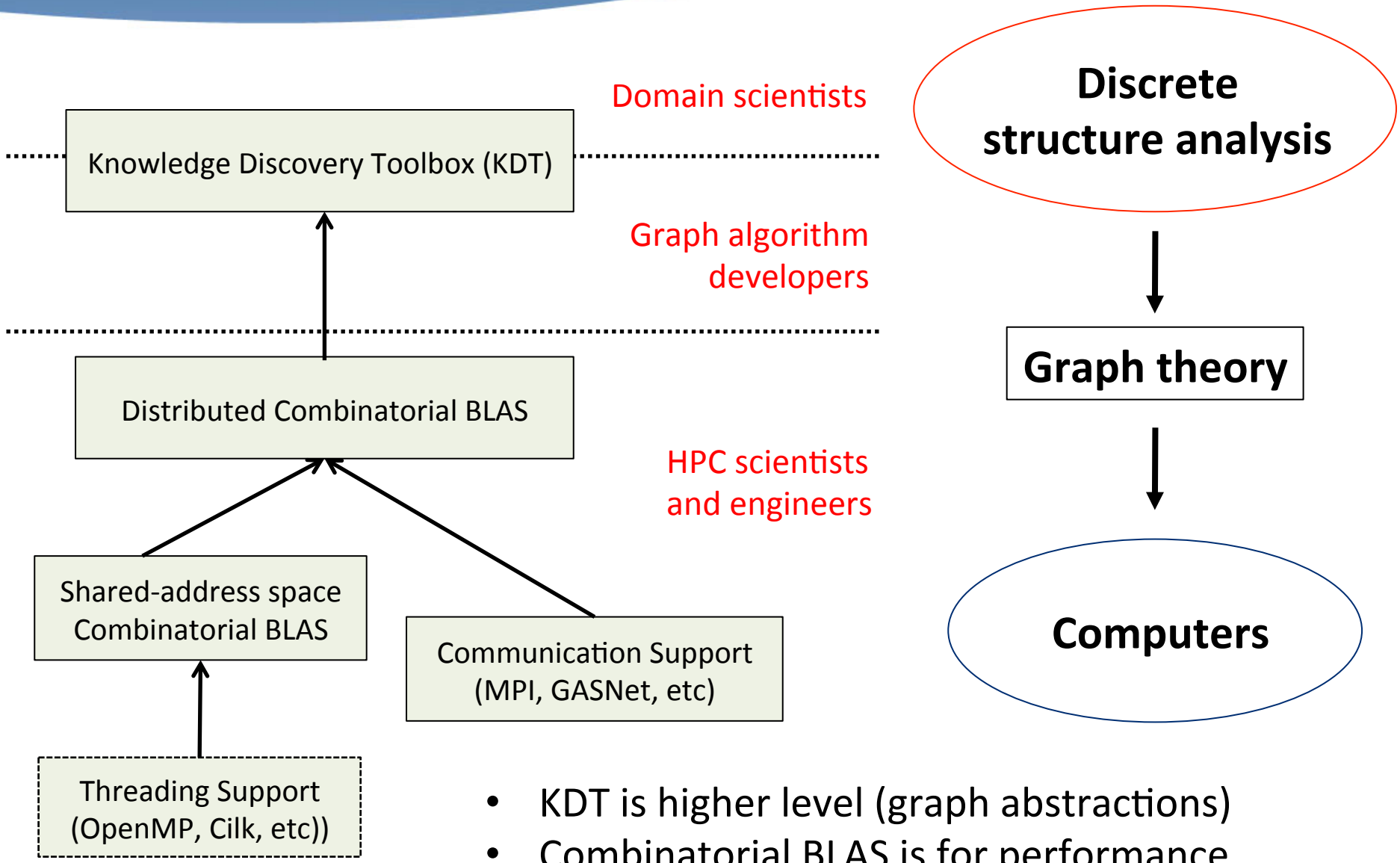
Parallel Graph Analysis Software



Parallel Graph Analysis Software



Parallel Graph Analysis Software

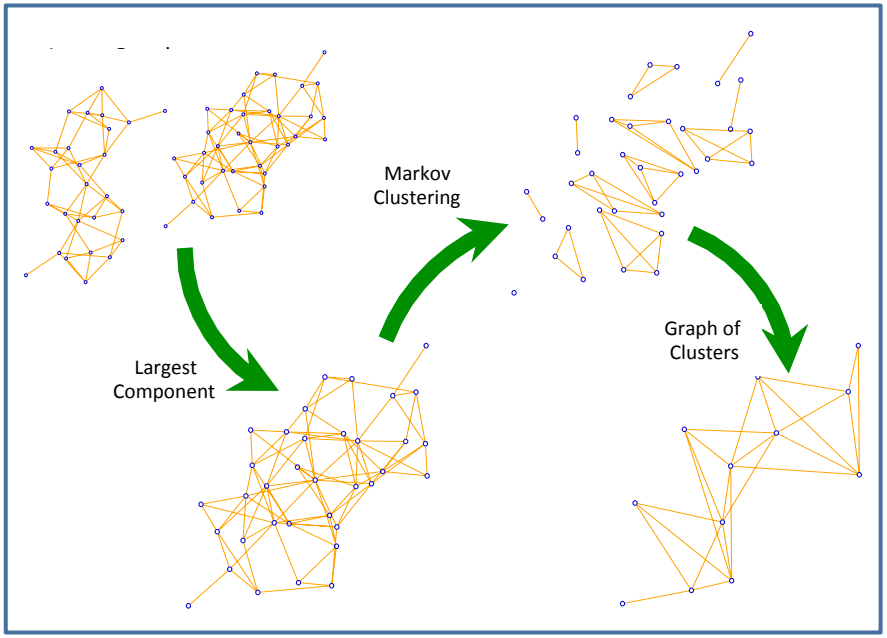


- KDT is higher level (graph abstractions)
- Combinatorial BLAS is for performance

Domain expert vs. Graph expert

- (Semantic) directed graphs
  - constructors, I/O
  - basic graph metrics (*e.g.*, `degree()`)
  - vectors
- Clustering / components
- Centrality / authority:

  betweenness centrality, PageRank



- Hypergraphs and sparse matrices
- Graph primitives (*e.g.*, `bfsTree()`)
- SpMV / SpGEMM on semirings

Domain expert vs. Graph expert

- (Semantic) directed graphs
  - constructors, I/O
  - basic graph metrics (*e.g.*, `degree()`)
  - vectors
- Clustering / components
- Centrality / authority:

  betweenness centrality, PageRank

```
# bigG contains the input graph
comp = bigG.connComp()
giantComp = comp.hist().argmax()
G = bigG.subgraph(comp==giantComp)

clus = G.cluster('Markov')

clusNedge = G.nedge(clus)

smallG = G.contract(clus)

# visualize
```

- Hypergraphs and sparse matrices
- Graph primitives (*e.g.*, `bfsTree()`)
- SpMV / SpGEMM on semirings

Domain expert vs. Graph expert

- (Semantic) directed graphs
  - constructors, I/O
  - basic graph metrics (*e.g.*, `degree()`)
  - vectors
- Clustering / components
- Centrality / authority:

  betweenness centrality, PageRank

```
# bigG contains the input graph
comp = bigG.connComp()
giantComp = comp.hist().argmax()
G = bigG.subgraph(comp==giantComp)

clus = G.cluster('Markov')

clusNedge = G.nedge(clus)

smallG = G.contract(clus)

# visualize
```

- Hypergraphs and sparse matrices
- Graph primitives (*e.g.*, `bfsTree()`)
- SpMV / SpGEMM on semirings

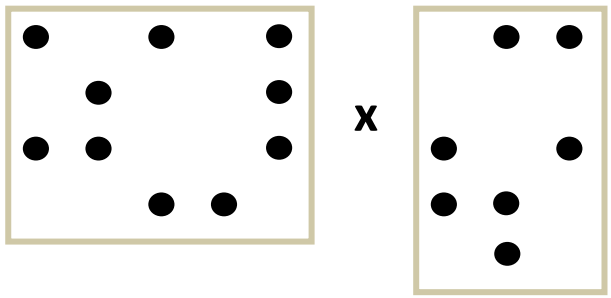
```
[…]
L = G.toSpParMat()
d = L.sum(kdt.SpParMat.Column)
L = -L
L.setDiag(d)
M = kdt.SpParMat.eye(G.nvert()) - mu*L
pos = kdt.ParVec.rand(G.nvert())
for i in range(nsteps):
    pos = M.SpMV(pos)
```

Outline

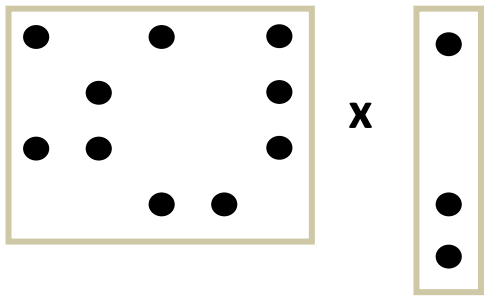
- Sustainable graph libraries
- **Linear algebraic primitives for graphs**
- Sparse triple product for graph subroutines
- Filtered semantic graphs
- Future challenges/directions

Linear-algebraic primitives

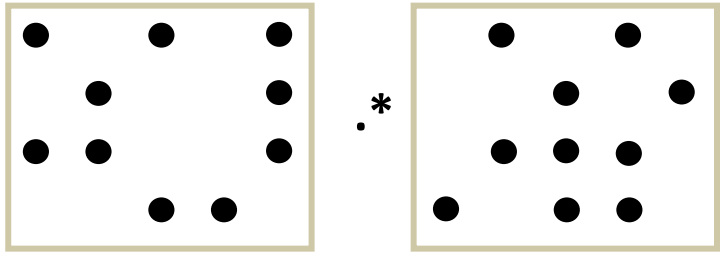
Sparse matrix-sparse matrix multiplication



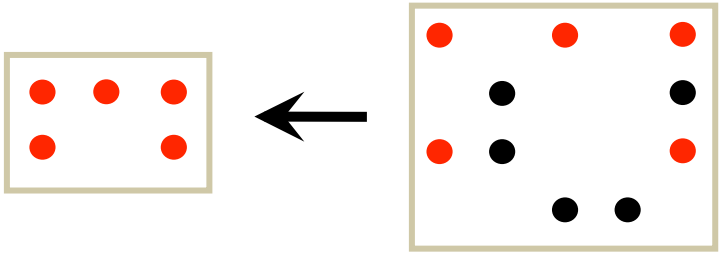
Sparse matrix-sparse vector multiplication



Element-wise operations



Sparse matrix indexing



The Combinatorial BLAS implements these, and more, on arbitrary semirings, e.g. (×, +), (and, or), (+, min)

The case for sparse matrices

Many irregular applications contain coarse-grained parallelism that can be exploited by abstractions at the proper level.

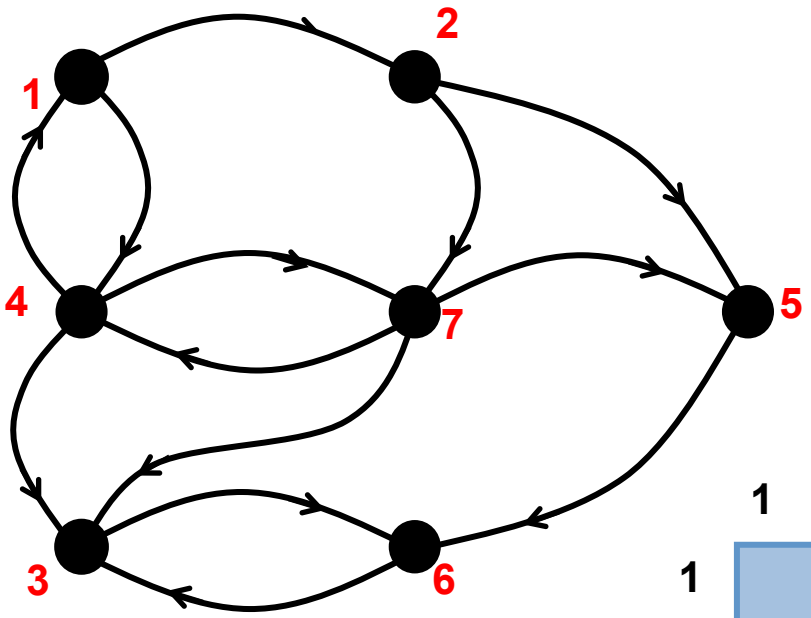
Traditional graph computations
Data driven, unpredictable communication.
Irregular and unstructured, poor locality of reference
Fine grained data accesses, dominated by latency

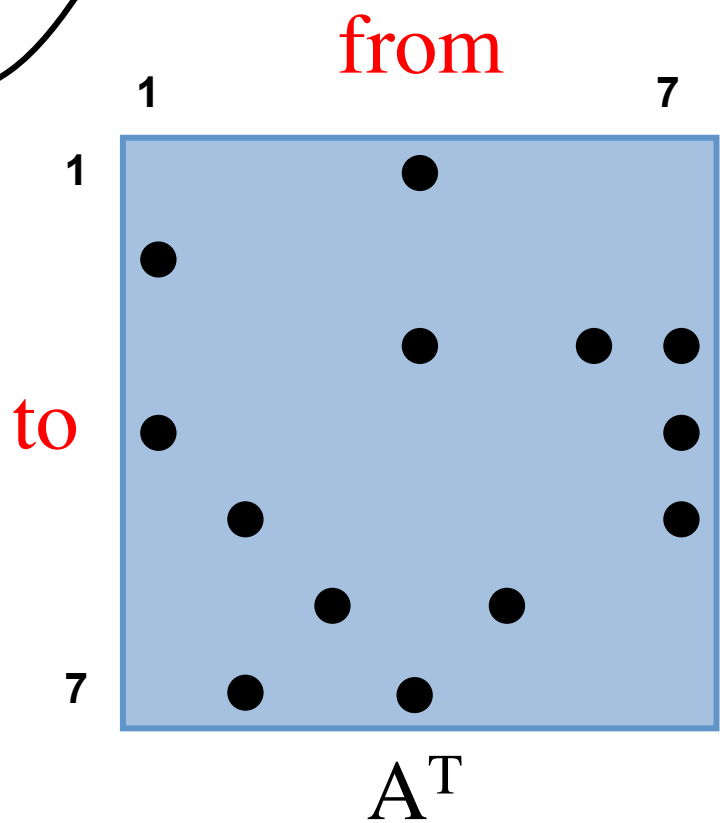
The case for sparse matrices

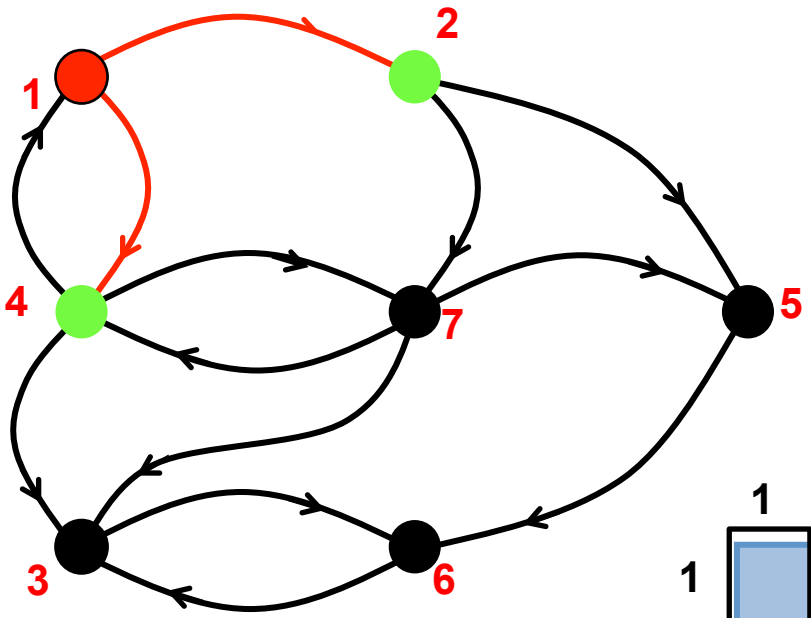
Many irregular applications contain coarse-grained parallelism that can be exploited by abstractions at the proper level.

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

Breadth-first search in Combinatorial BLAS



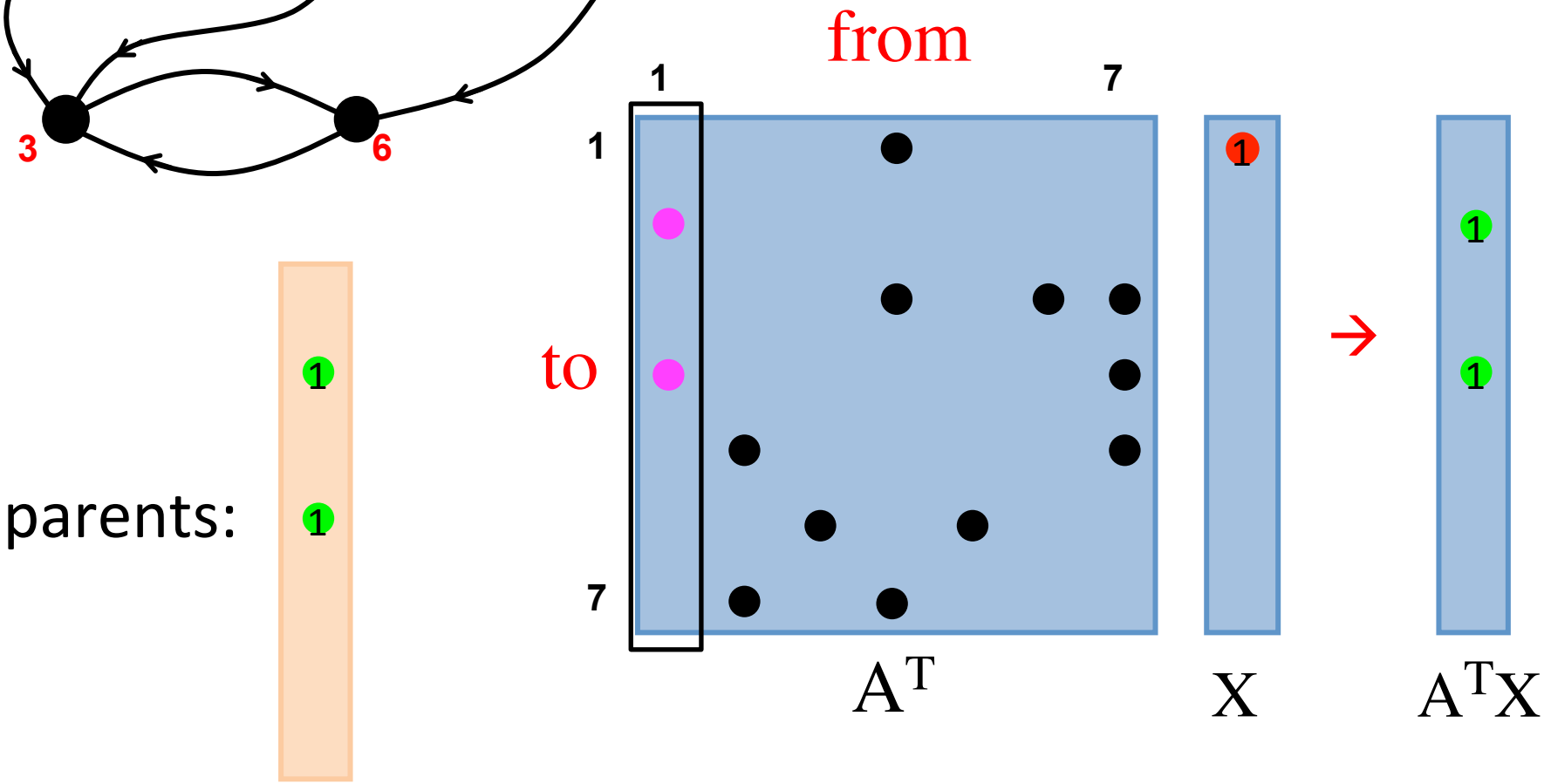


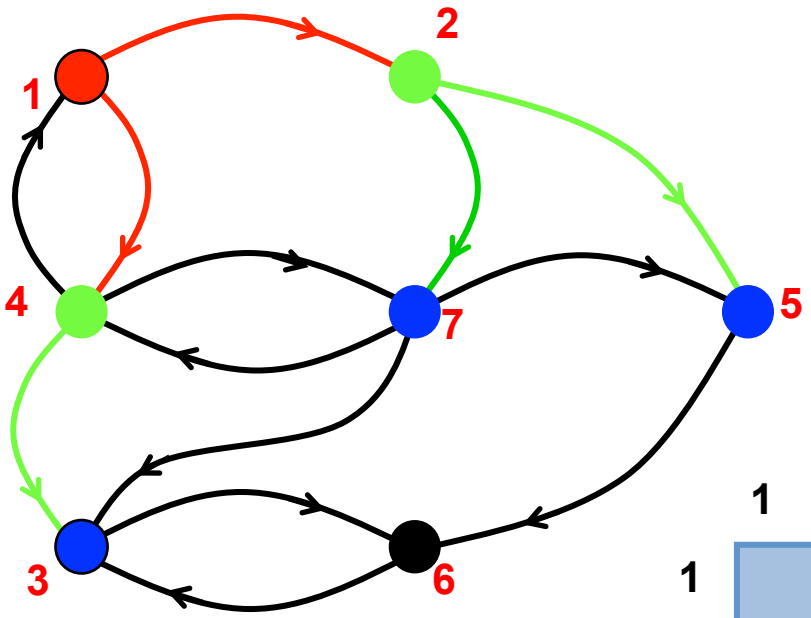


Particular semiring operations:

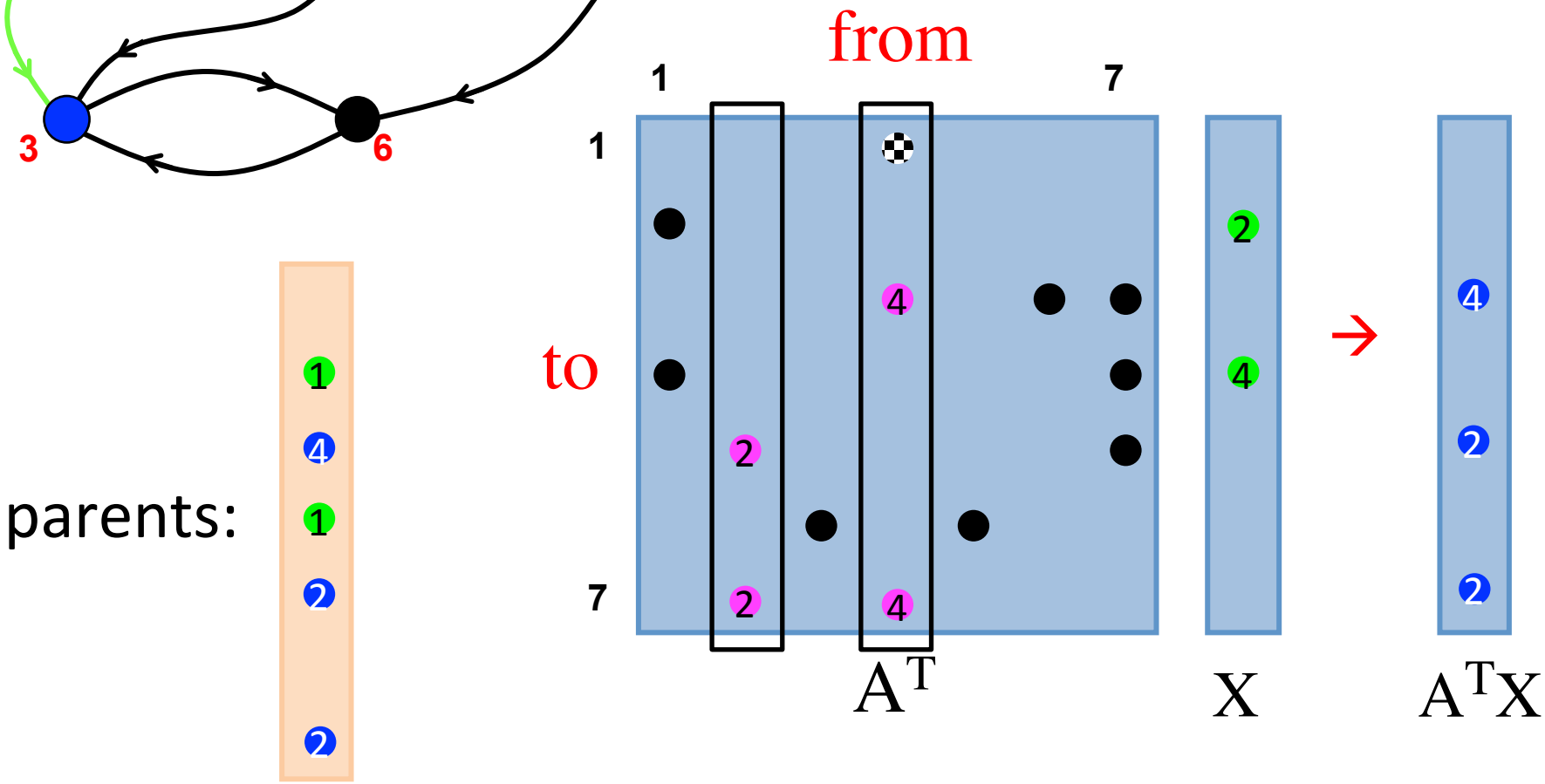
Multiply: select

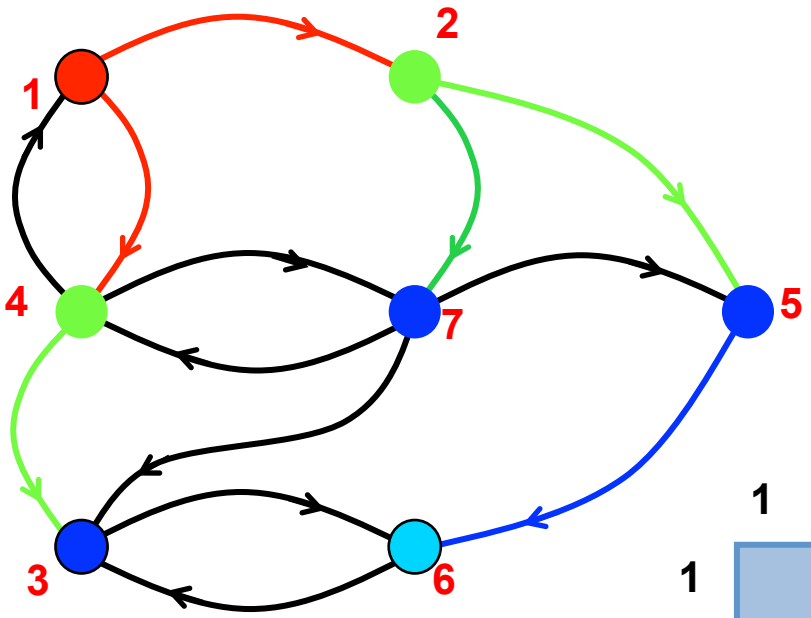
Add: minimum



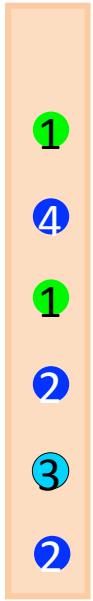


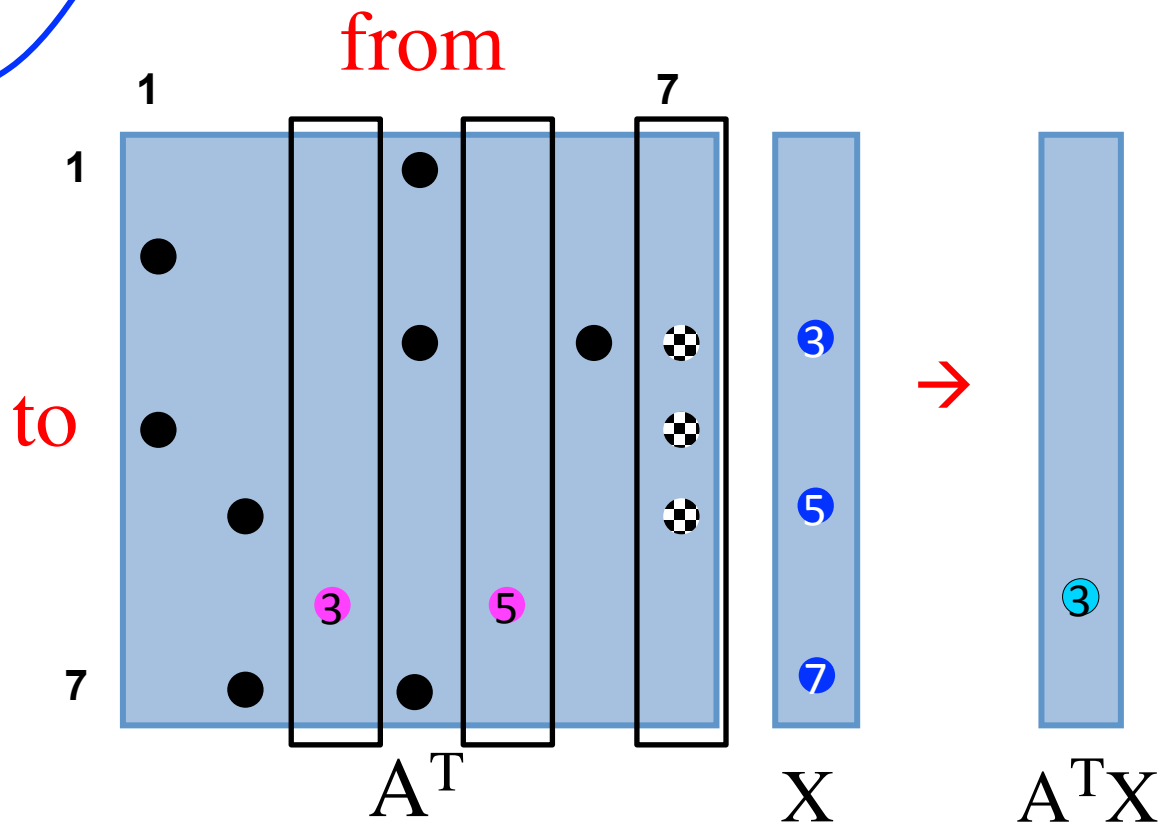
Select vertex with <u>minimum</u> label as parent

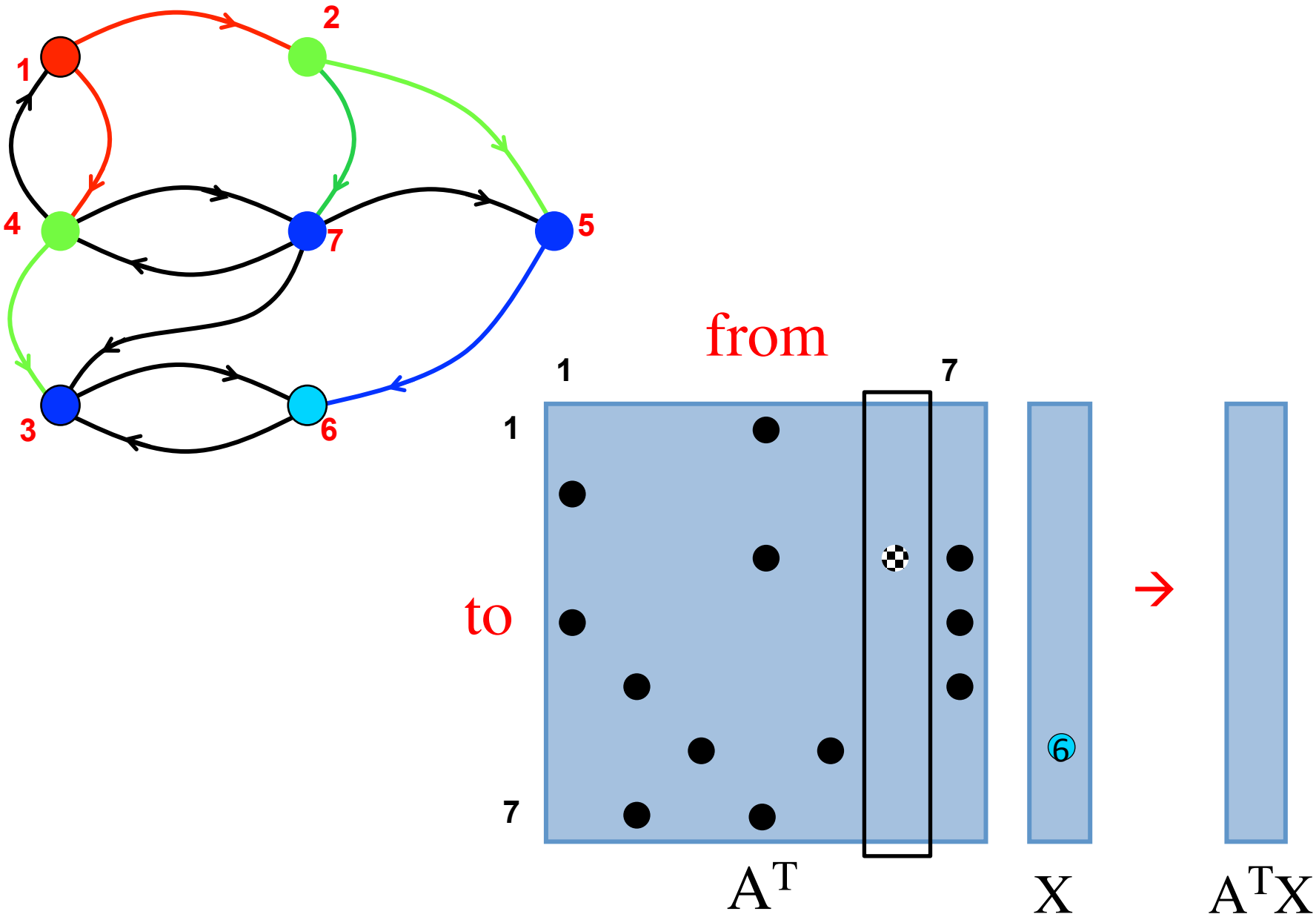




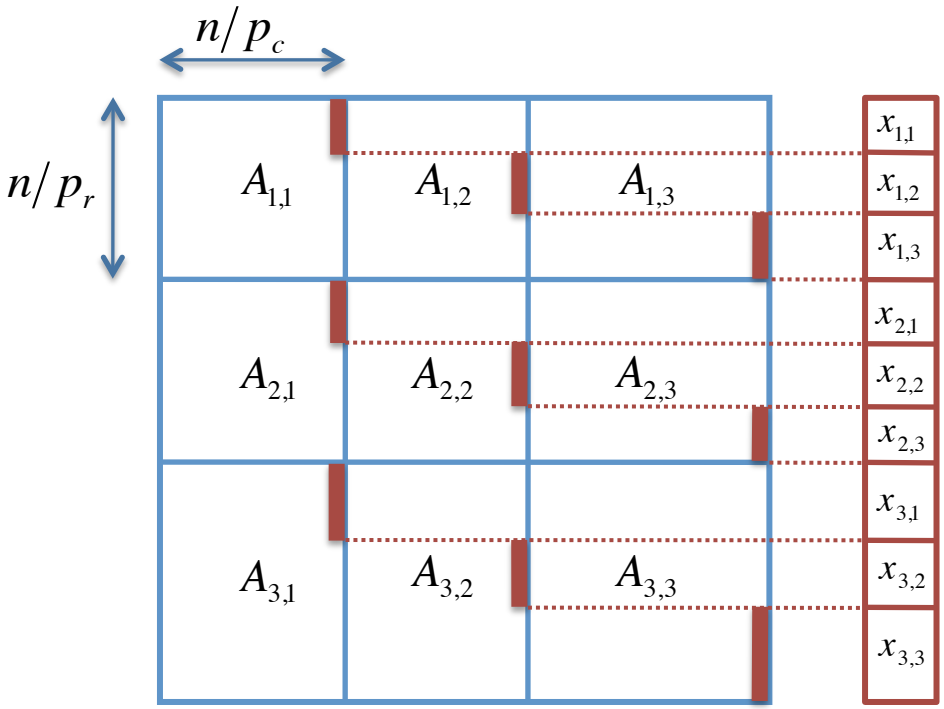
parents:






1
2
3
4
5
6
7
from
to
1
7
1
7
6
A^T
X
A^TX

2D layout for sparse matrices & vectors



Matrix/vector distributions, interleaved on each other.

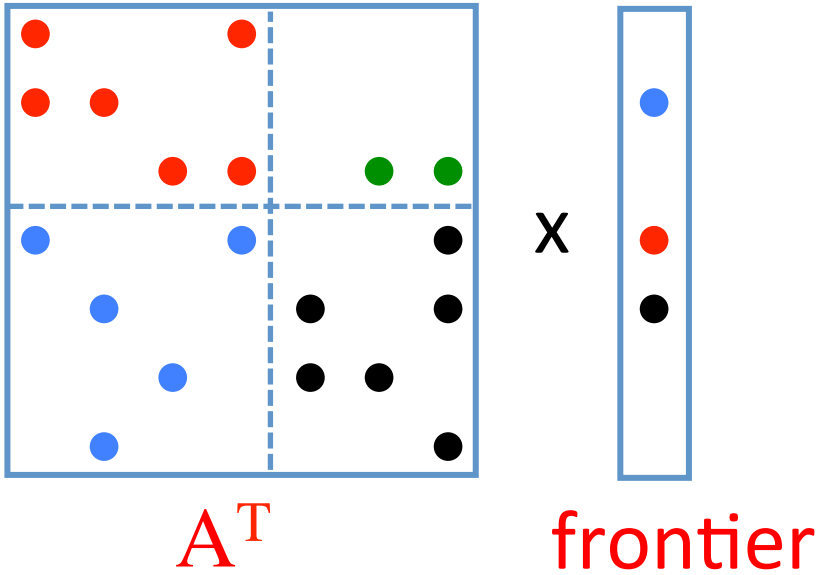
Default distribution in **Combinatorial BLAS**.

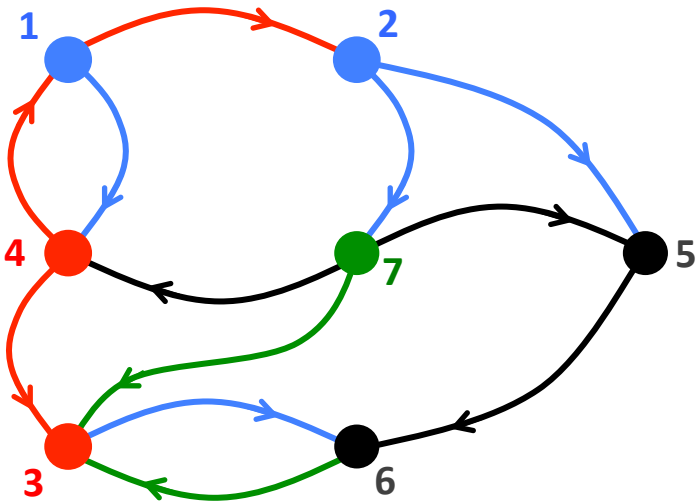
Scalable with increasing number of processes

- 2D matrix layout wins over 1D with large core counts and with limited bandwidth/compute
- 2D vector layout sometimes important for load balance

B., Gilbert. The Combinatorial BLAS: Design, implementation, and applications. *The International Journal of High Performance Computing Applications*, 2011.

2D parallel BFS algorithm

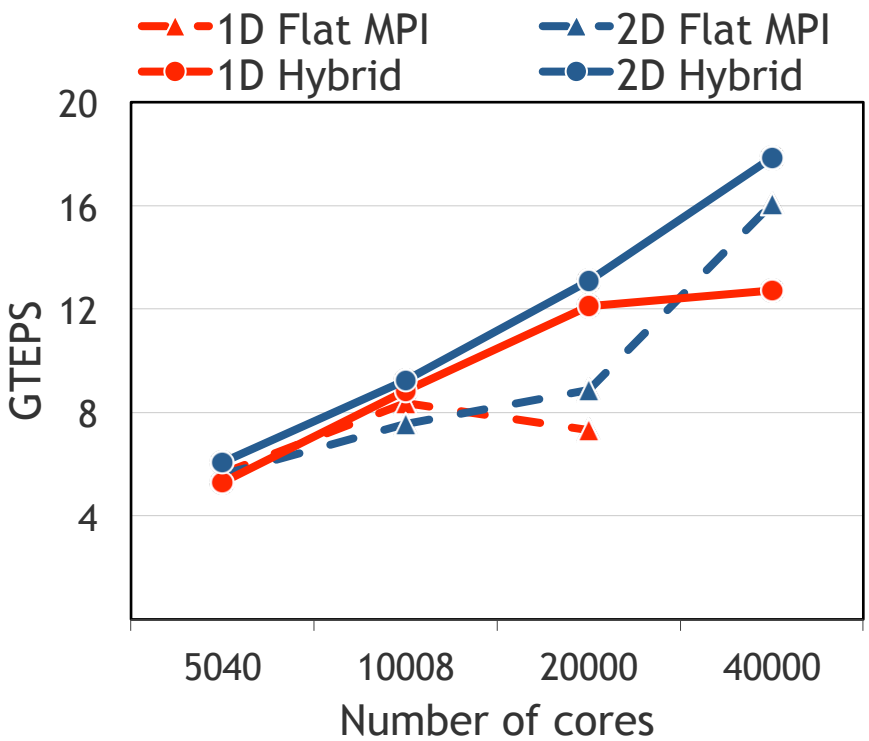


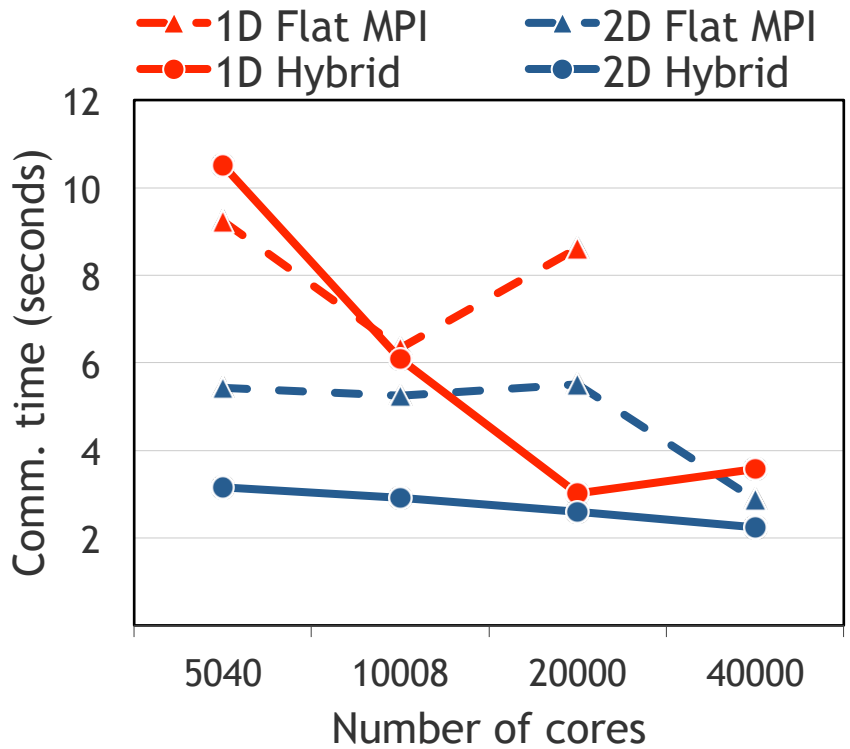


ALGORITHM:

1. Gather vertices in *processor column* **[communication]**
2. Find owners of the current frontier's adjacency [computation]
3. Exchange adjacencies in *processor row* **[communication]**
4. Update distances/parents for unvisited vertices. [computation]

BFS strong scaling





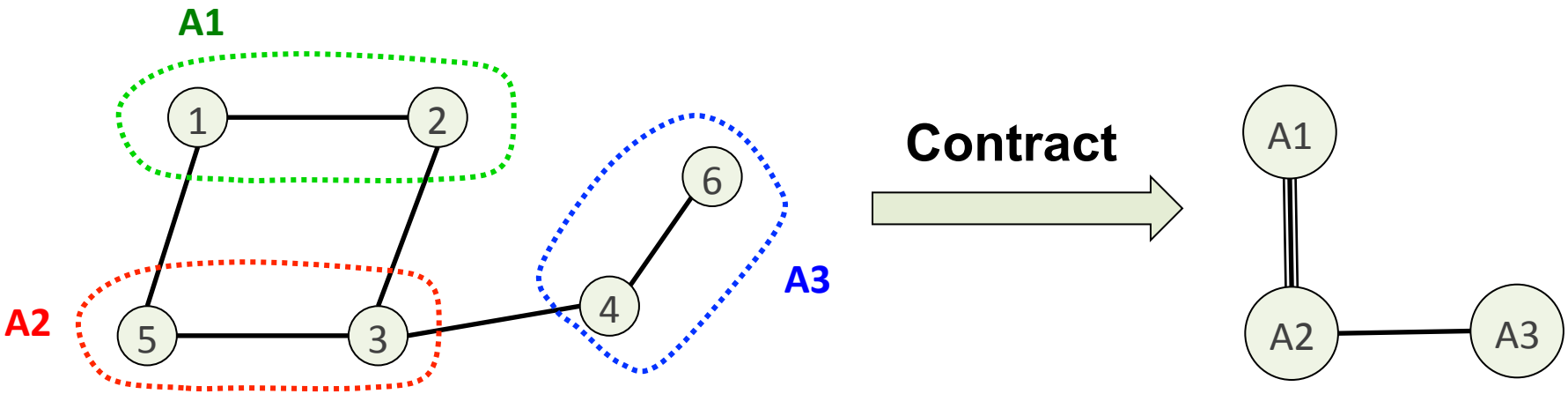
- NERSC Hopper (Cray XE6, Gemini interconnect AMD Magny-Cours)
- Hybrid: In-node 6-way OpenMP multithreading
- Graph500 (R-MAT): 4 billion vertices and 64 billion edges.

B., Madduri. Parallel breadth-first search on distributed memory systems. *Supercomputing*, 2011.

Outline

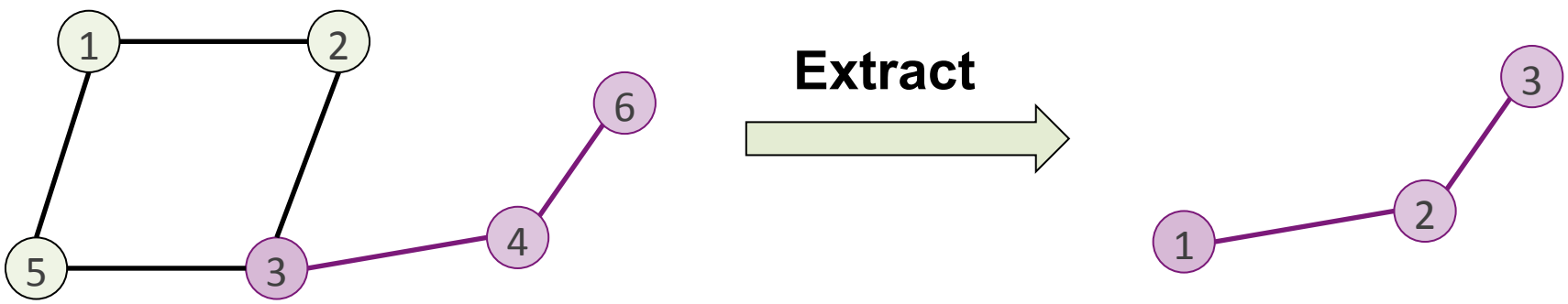
- Sustainable graph libraries
- Linear algebraic primitives for graphs
- **Sparse triple product for graph subroutines**
- Filtered semantic graphs
- Future challenges/directions

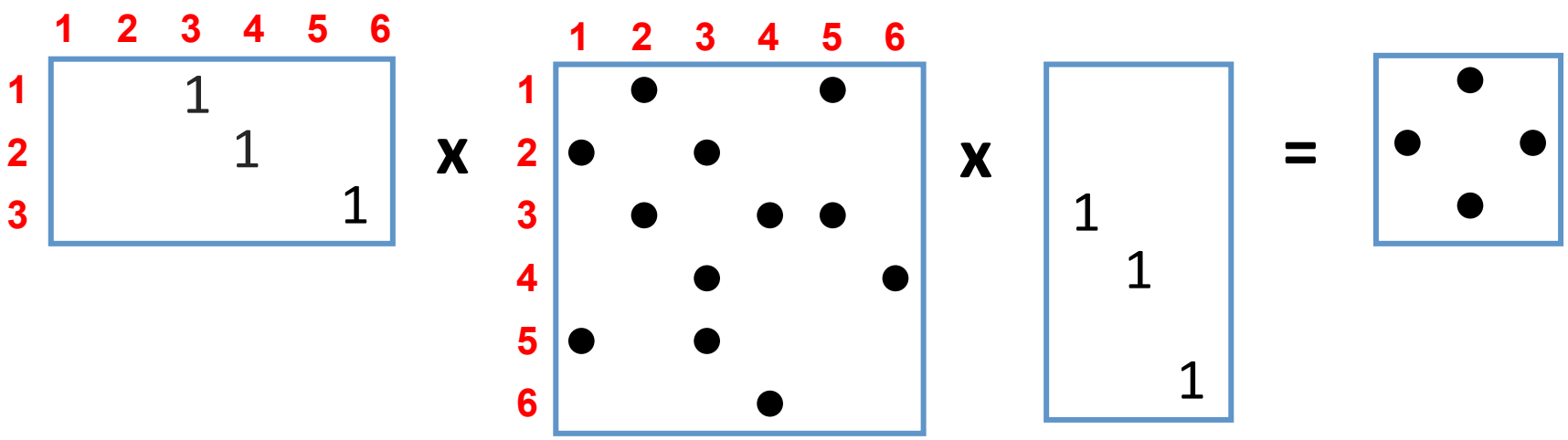
Graph contraction via sparse triple product



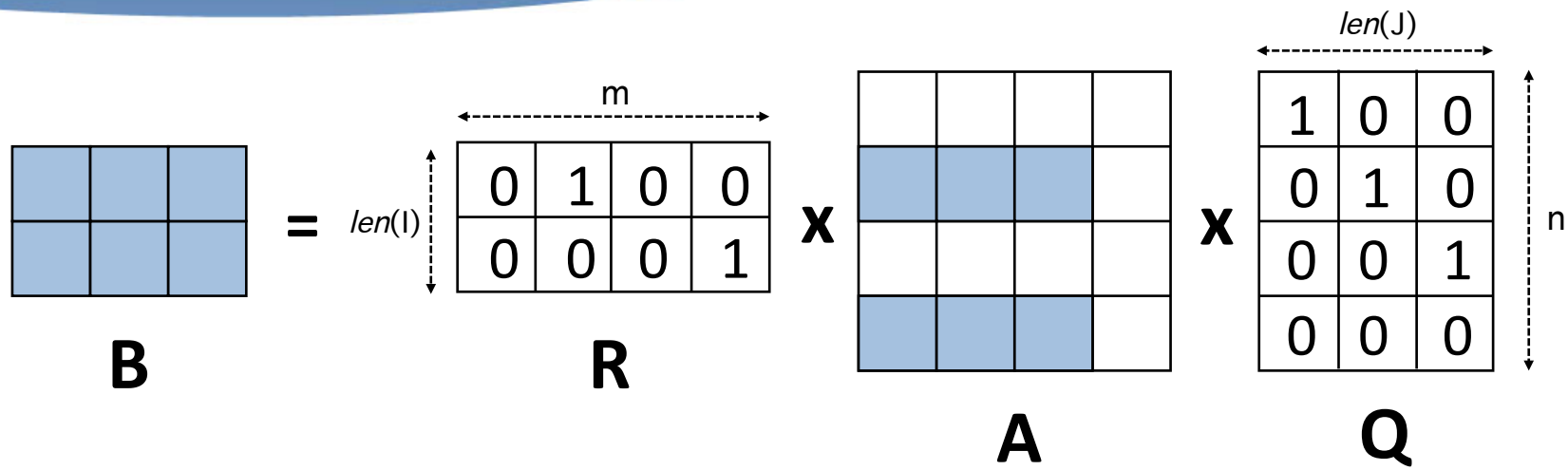
$$
\begin{array}{c|cccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
1 & 1 & 1 & & & & \\
2 & & & 1 & & 1 & \\
3 & & & & 1 & & 1
\end{array}
\quad \mathbf{x} \quad
\begin{array}{c|cccccc}
 & 1 & 2 & 3 & 4 & 5 & 6 \\
\hline
1 & & \bullet & & & \bullet & \\
2 & \bullet & & \bullet & & & \\
3 & & \bullet & & \bullet & \bullet & \\
4 & & & \bullet & & & \bullet \\
5 & \bullet & & \bullet & & & \\
6 & & & & \bullet & &
\end{array}
\quad \mathbf{x} \quad
\begin{array}{cccccc}
1 & & & & & \\
1 & & & & & \\
 & 1 & & & & \\
 & & & 1 & & \\
 & 1 & & & & \\
 & & & & 1 &
\end{array}
\quad = \quad
\begin{array}{cccccc}
 & & \bullet & & & \\
\bullet & & & & & \bullet \\
 & & \bullet & & &
\end{array}
$$

Subgraph extraction via sparse triple product





Sparse matrix indexing (SpRef)



SpRef using mixed-mode sparse matrix-matrix multiplication (**SpGEMM**). Ex: B = A([2,4], [1,2,3])

$$T_{comm} = \Theta\left(\alpha \cdot \sqrt{p} + \beta \cdot \frac{nnz(A)}{\sqrt{p}}\right)$$

Algorithmic structure makes general case analysis possible

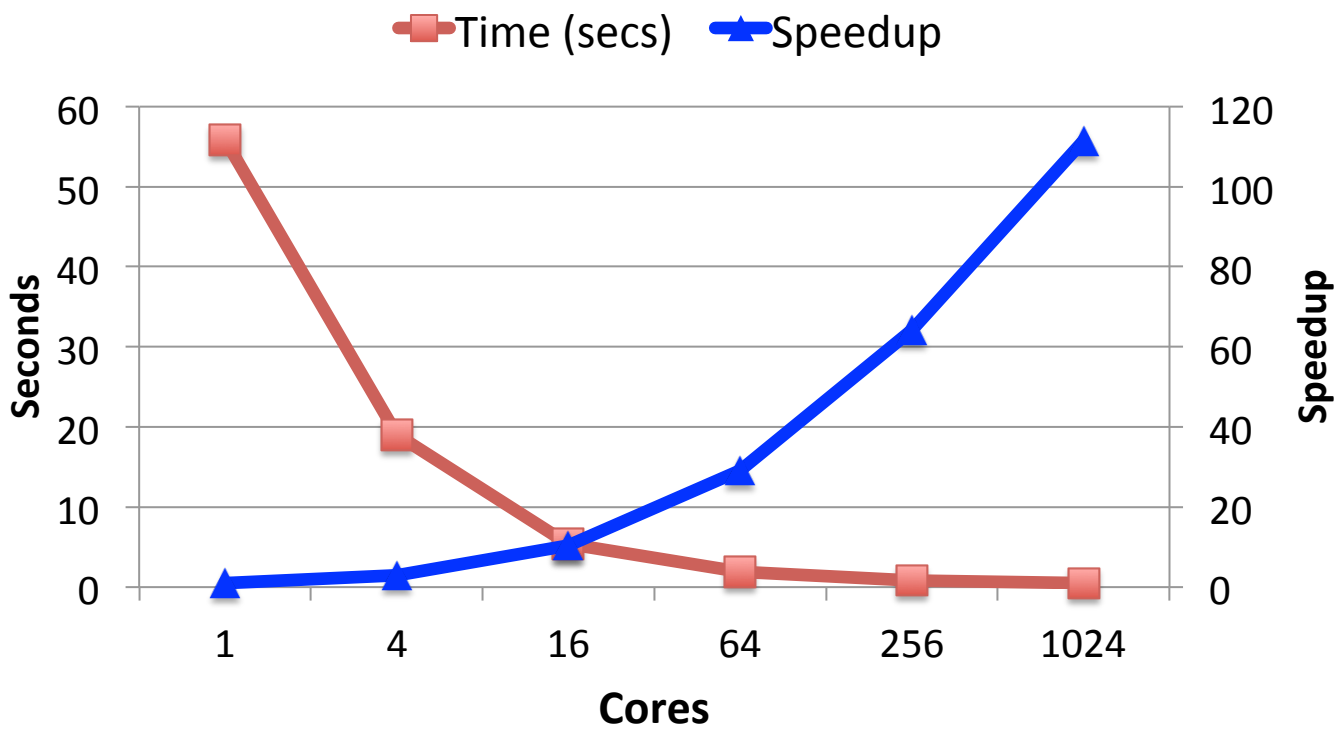
Dominated by **SpGEMM**

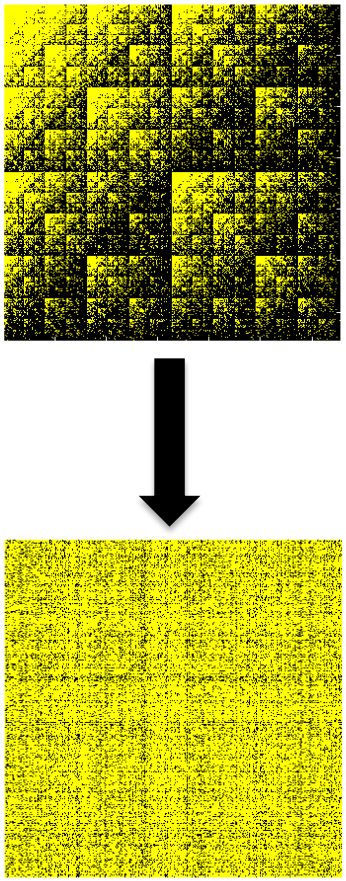
Bottleneck: bandwidth costs

Speedup: $\Theta\left(\sqrt{p}\right)$

B., Gilbert. Parallel sparse matrix-matrix multiplication and indexing: Implementation and experiments. *SIAM Journal of Scientific Computing*, 2012 (to appear)

Strong scaling of vertex relabeling

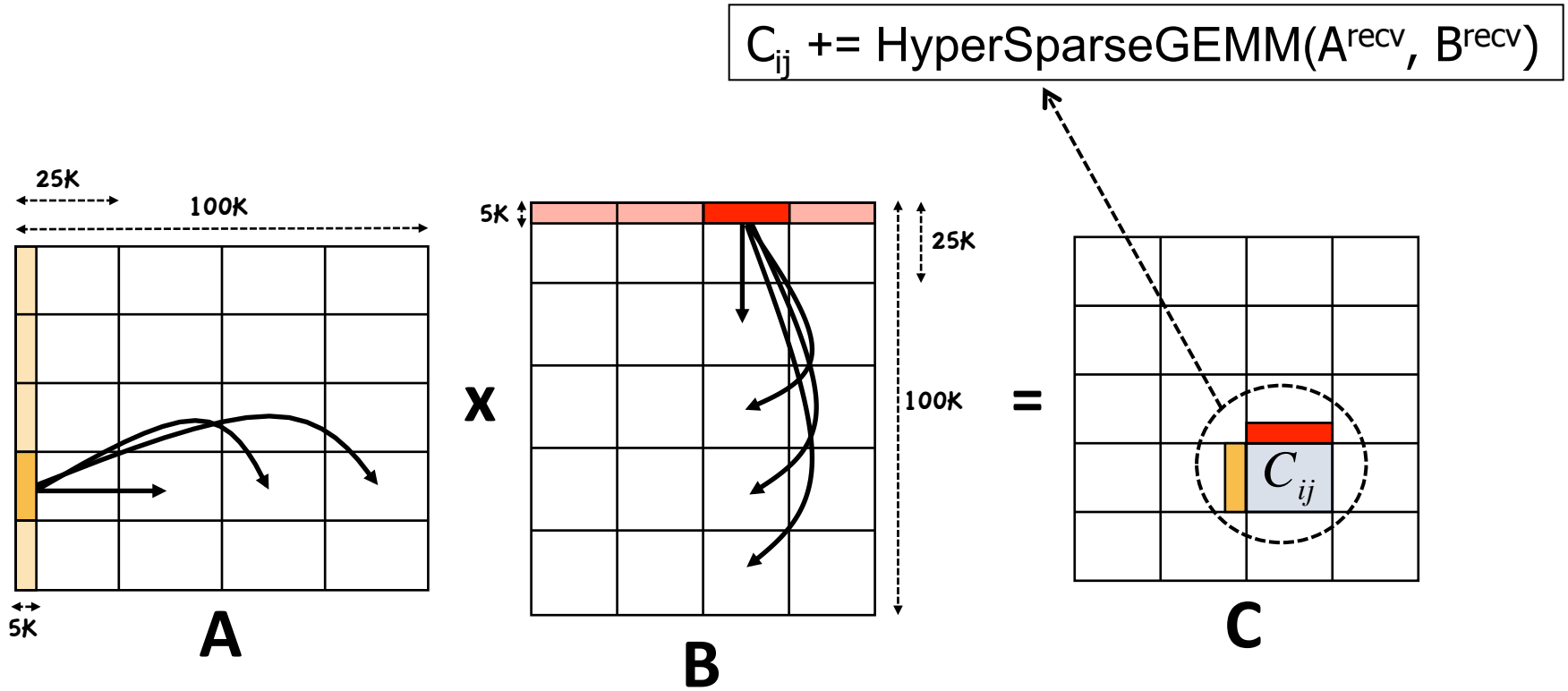




random symmetric permutation ⇔ relabeling graph vertices

- RMAT Scale 22; edge factor=8; a=.6, b=c=d=.4/3
- Franklin/NERSC, each node is a quad-core AMD Budapest

Parallel sparse matrix-matrix multiplication algorithms



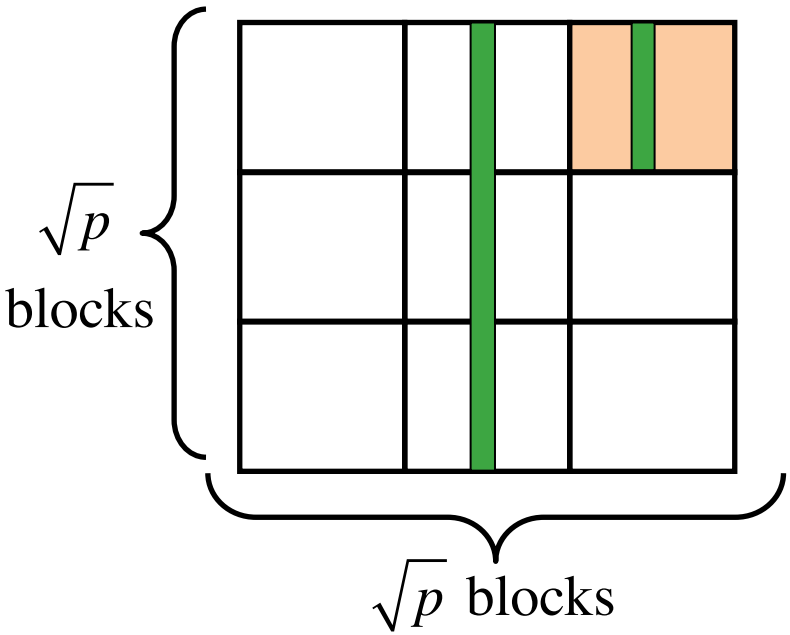
2D algorithm: Sparse SUMMA (based on dense SUMMA)

General implementation that handles rectangular matrices

B., Gilbert. Challenges and advances in parallel sparse matrix-matrix multiplication. In *ICPP'08*

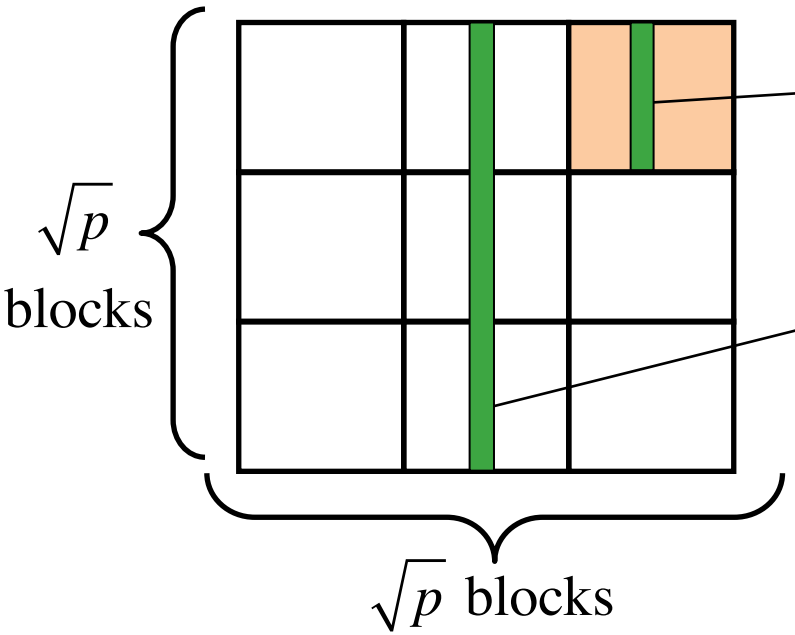
Local submatrix storage

Submatrices are "*hypersparse*" *(i.e. nnz << n)*



Local submatrix storage

Submatrices are "*hypersparse*" *(i.e. nnz << n)*

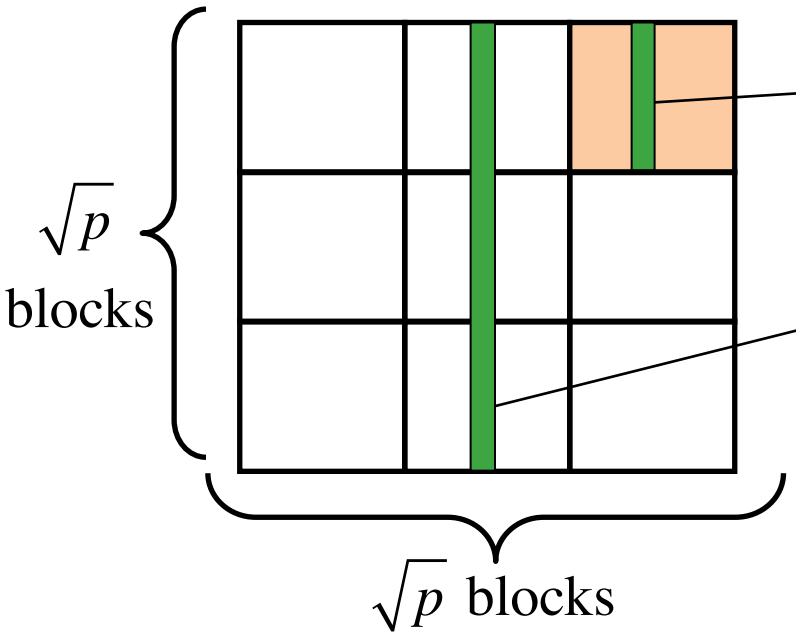


$$nnz' = \frac{c}{\sqrt{p}} \rightarrow 0$$

Average of c nonzeros per column

Local submatrix storage

Submatrices are "*hypersparse*" *(i.e. nnz << n)*



$$nnz' = \frac{c}{\sqrt{p}} \rightarrow 0$$

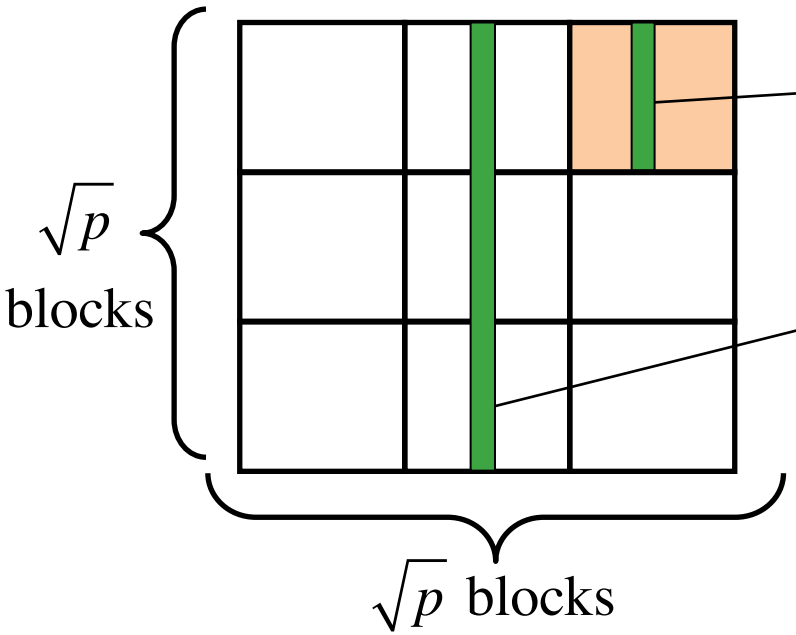
Average of c nonzeros per column

<u>Total Storage</u>:

$$O(n + nnz) \rightarrow O(n\sqrt{p} + nnz)$$

Local submatrix storage

Submatrices are "*hypersparse*" *(i.e. nnz << n)*



$$nnz' = \frac{c}{\sqrt{p}} \rightarrow 0$$

Average of c nonzeros per column

<u>Total Storage</u>:

$$O(n + nnz) \rightarrow O(n\sqrt{p} + nnz)$$

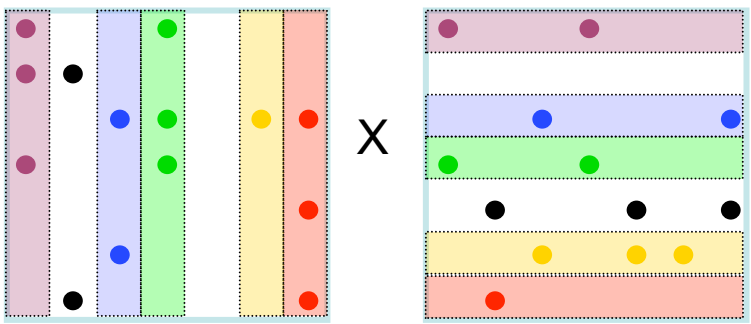
- A data structure or algorithm that depends on matrix dimension n (e.g. CSR or CSC) is asymptotically too wasteful for submatrices
- Use doubly-compressed (DCSC) data structures or compressed sparse blocks (CSB) instead

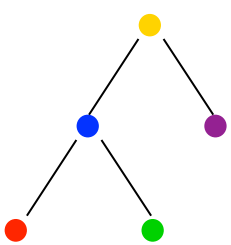
Sequential "hypersparse" kernel

Operates on the strictly O(nnz) DCSC data structure

Sparse outer-product formulation with multi-way merging

Efficient in parallel, i.e. T(1) ≈ p T(p)





Time complexity:

$$O(flops \cdot \lg ni + nzc(A) + nzr(B))$$

- independent of dimension

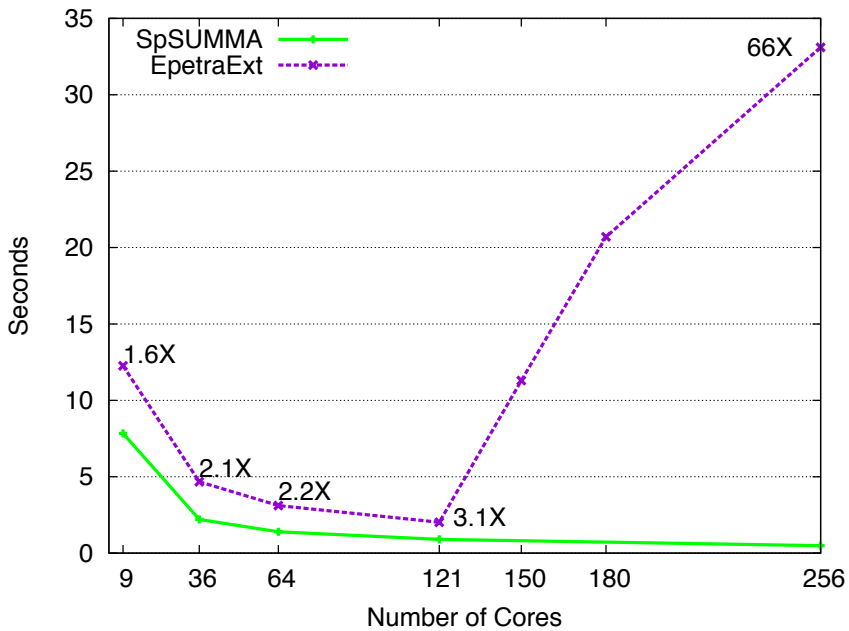
Space complexity:

$$O(nnz(A) + nnz(B) + nnz(C))$$

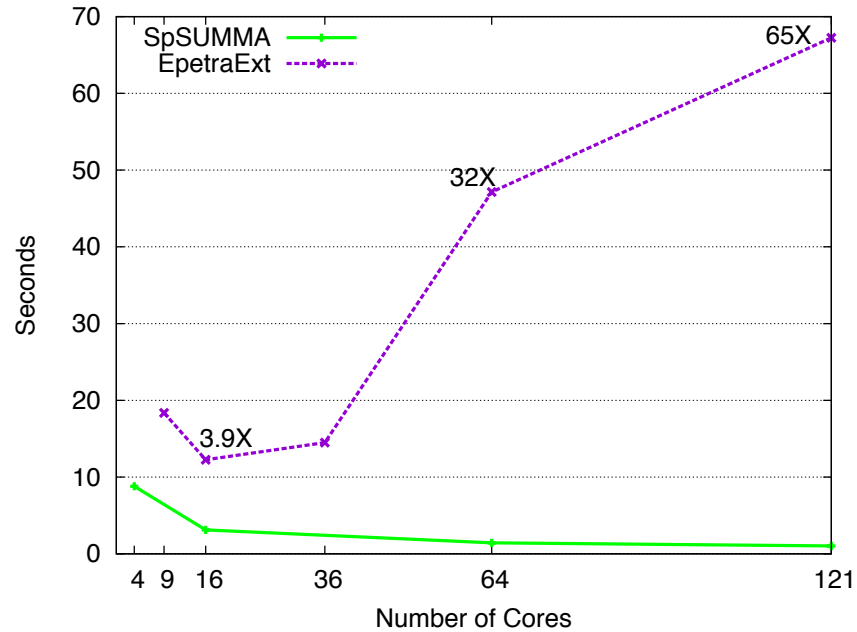
- independent of flops

B., Gilbert. On the Representation and Multiplication of Hypersparse Matrices. In *IPDPS*, 2008.

1D vs. 2D scaling for sparse matrix-matrix multiplication



(a) R-MAT × R-MAT product (scale 21).



(b) Multiplication of an R-MAT matrix of scale 23 with the restriction operator of order 8.

SpSUMMA = 2-D data layout (Combinatorial BLAS)
EpetraExt = 1-D data layout (Trilinos)

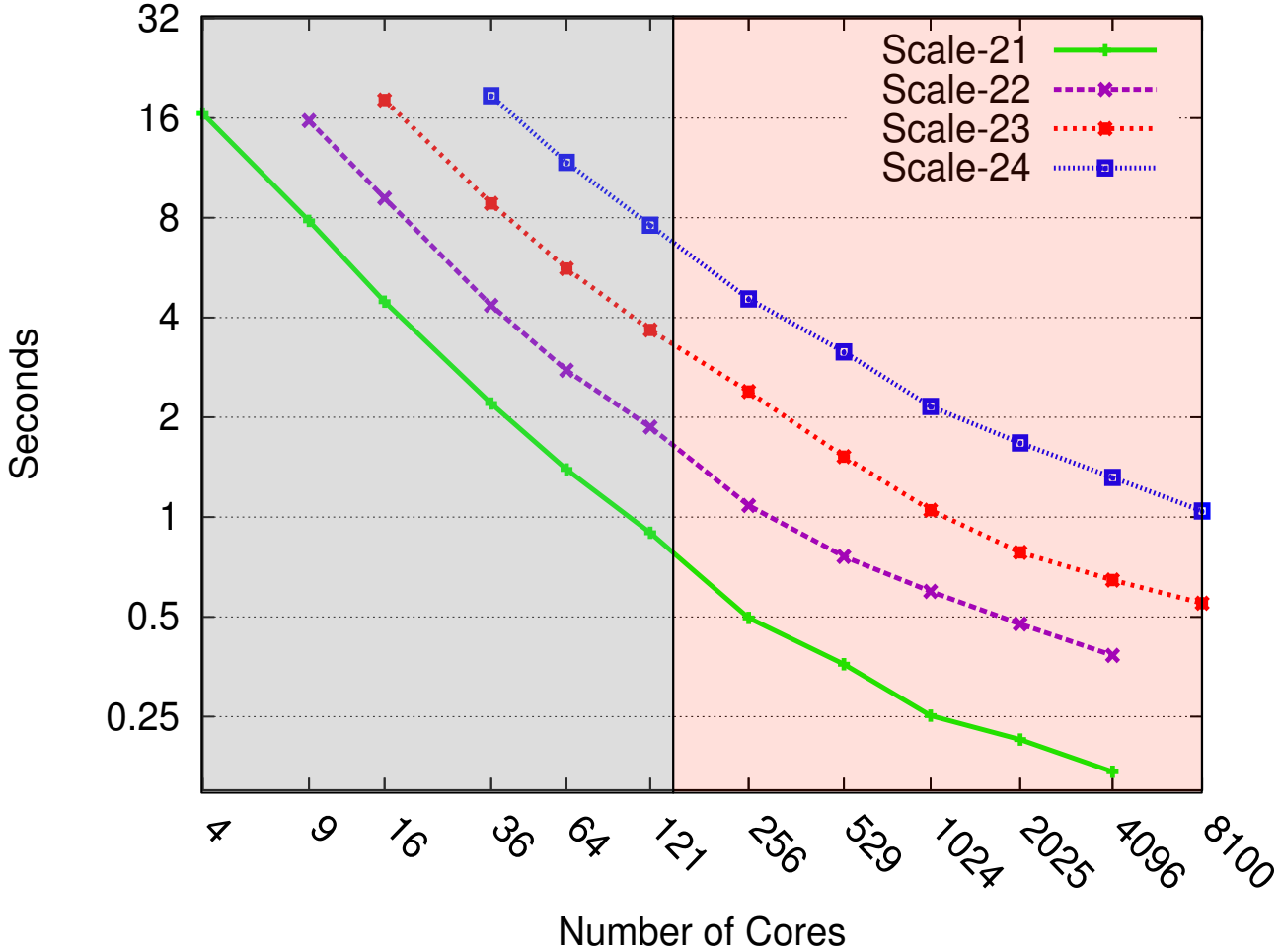
In practice, 2D algorithms have <u>the potential</u> to scale, but not linearly

$$T_{comm}(2\mathrm{D}) = \alpha p\sqrt{p} + \beta cn\sqrt{p}$$

$$T_{comp}(optimal) = c^2 n$$

Square sparse matrix multiplication

Almost linear scaling until bandwidth costs starts to dominate



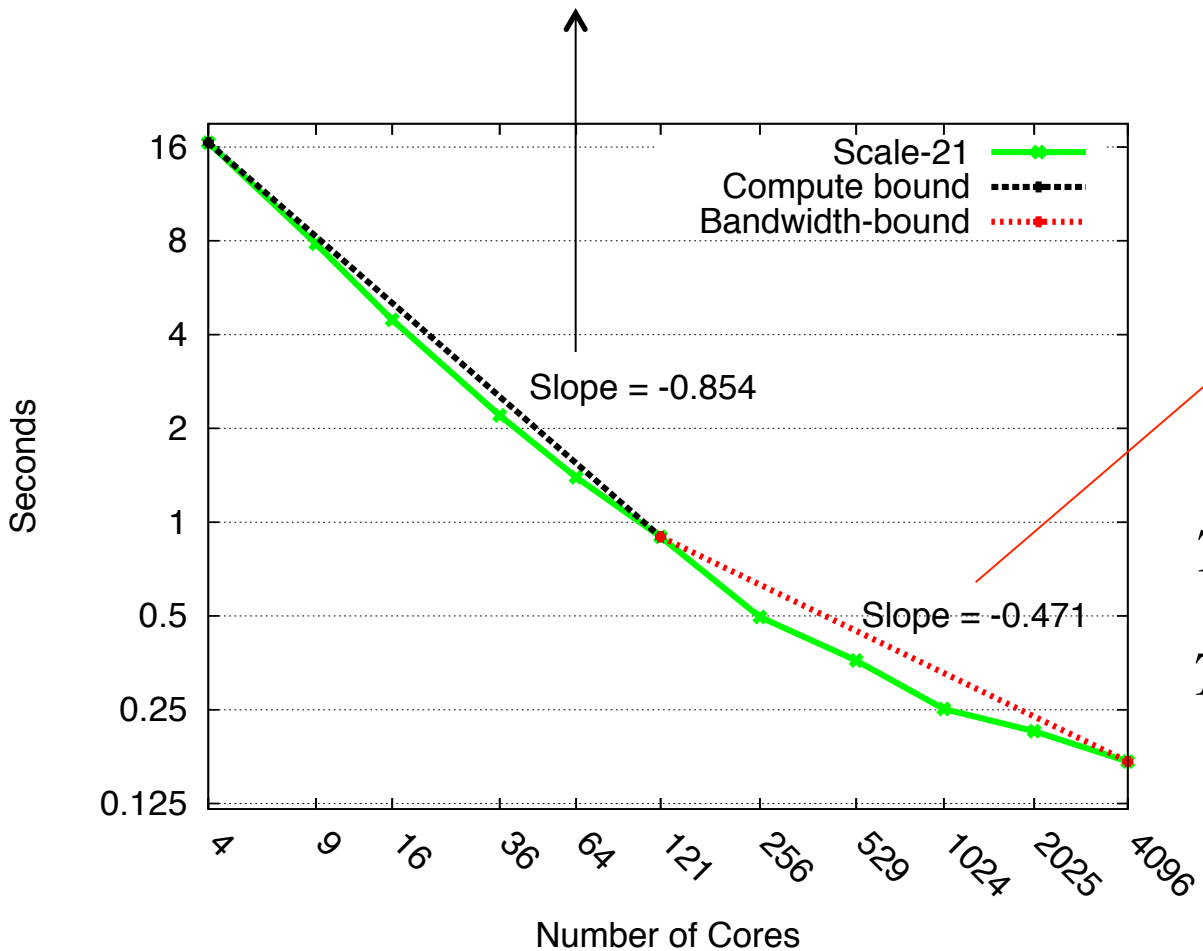
Scaling proportional to √p afterwards

NERSC/Franklin
Cray XT4

R-MAT, edgefactor: 8
a=0.6, b=c=d=0.4/3

Square sparse matrix multiplication

Almost linear scaling until bandwidth costs starts to dominate



Scaling proportional to √p afterwards

$$T_{comm} = \alpha p\sqrt{p} + \beta cn\sqrt{p}$$

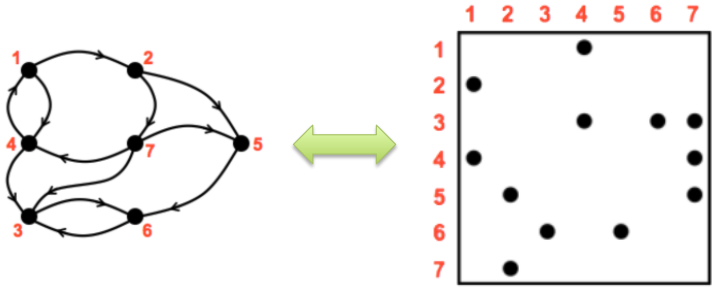
$$T_{comp}(optimal) = c^2 n$$

Outline

- Sustainable graph libraries
- Linear algebraic primitives for graphs
- Sparse triple product for graph subroutines
- **Filtered semantic graphs**
- Future challenges/directions

Knowledge Discovery Toolbox

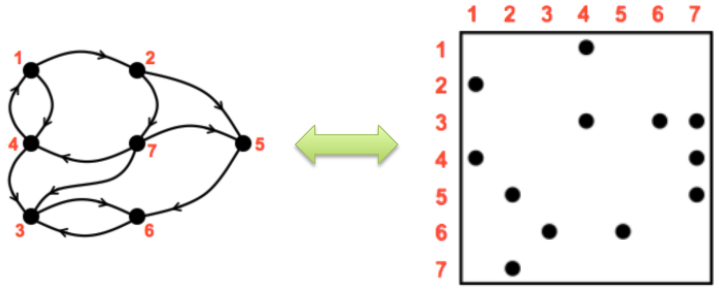
http://kdt.sourceforge.net/



A general graph library with operations based on linear algebraic primitives

Knowledge Discovery Toolbox

http://kdt.sourceforge.net/

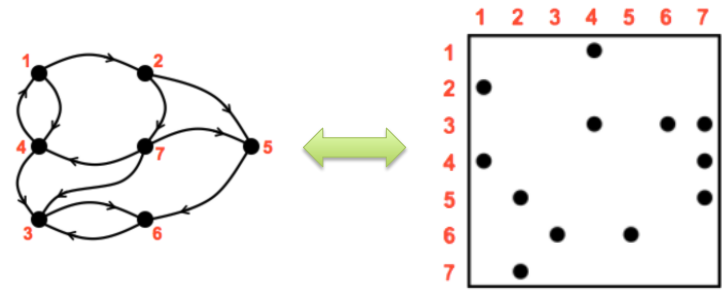


A general graph library with operations based on linear algebraic primitives

- Aimed at domain experts who know their problem well but don't know how to program a supercomputer
- Easy-to-use Python interface
- Runs on a laptop as well as a cluster with 10,000 processors

Knowledge Discovery Toolbox

http://kdt.sourceforge.net/



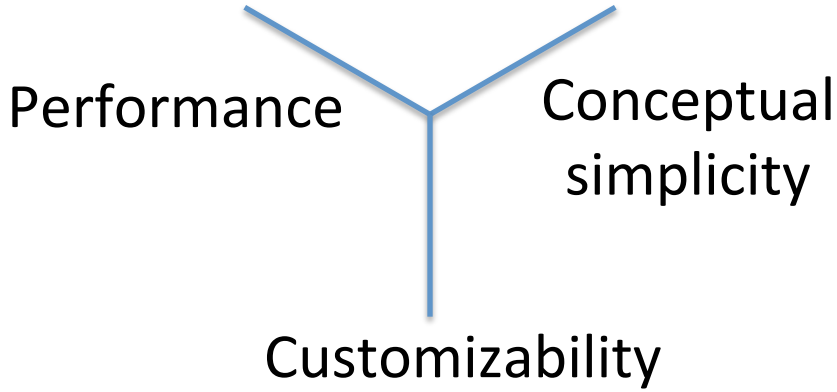
A general graph library with operations based on linear algebraic primitives

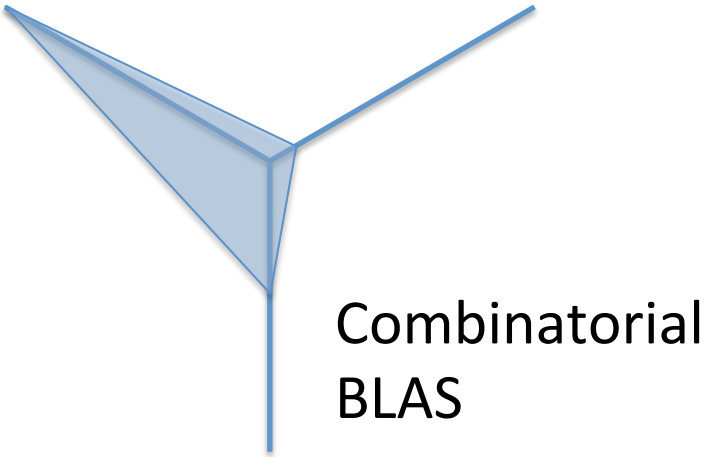
- Aimed at domain experts who know their problem well but don't know how to program a supercomputer
- Easy-to-use Python interface
- Runs on a laptop as well as a cluster with 10,000 processors

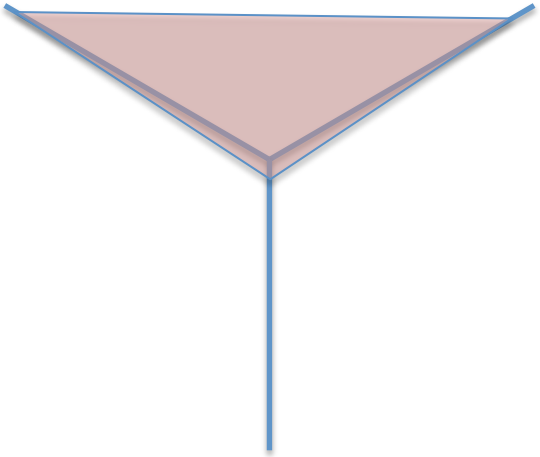
- A collaboration among LBNL, UCSB, UC Berkeley, and Cray Inc.
- Open source software, released under New BSD license
- v0.1 released March 2011; v0.2 released March 2012

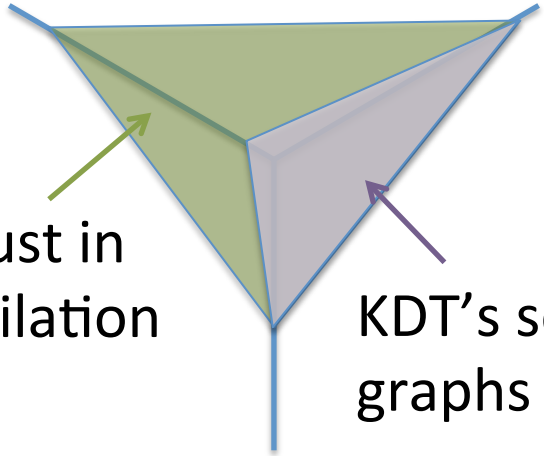
Lugowski, Alber, **B.**, Gilbert, Reinhardt, Teng, and Waranis. A flexible open-source toolbox for scalable complex graph analysis. *SIAM Conference on Data Mining (SDM),*2012

How to target "domain experts"?



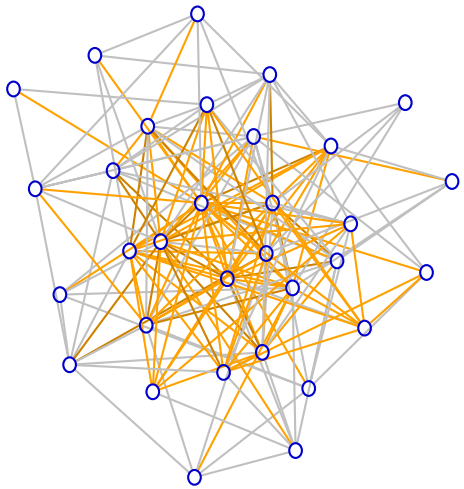




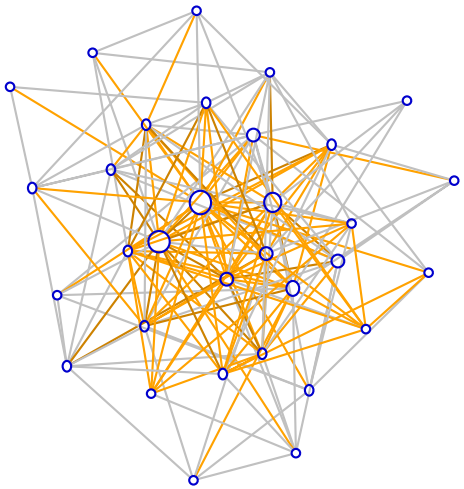


The need for filters

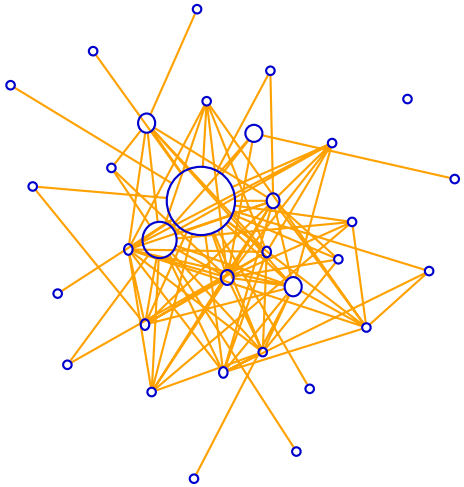
Graph of text & phone calls



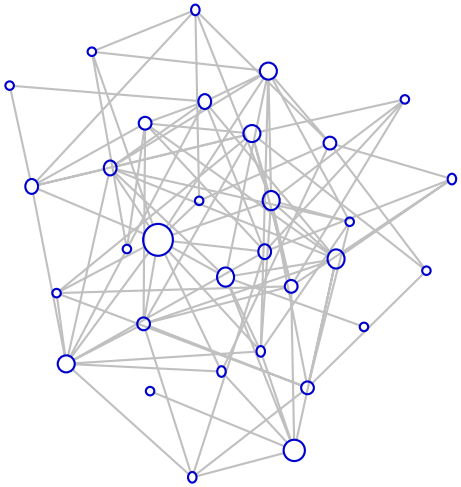
Betweenness centrality



Betweenness centrality on text messages



Betweenness centrality on phone calls



Attributed semantic graphs and filters

Example:

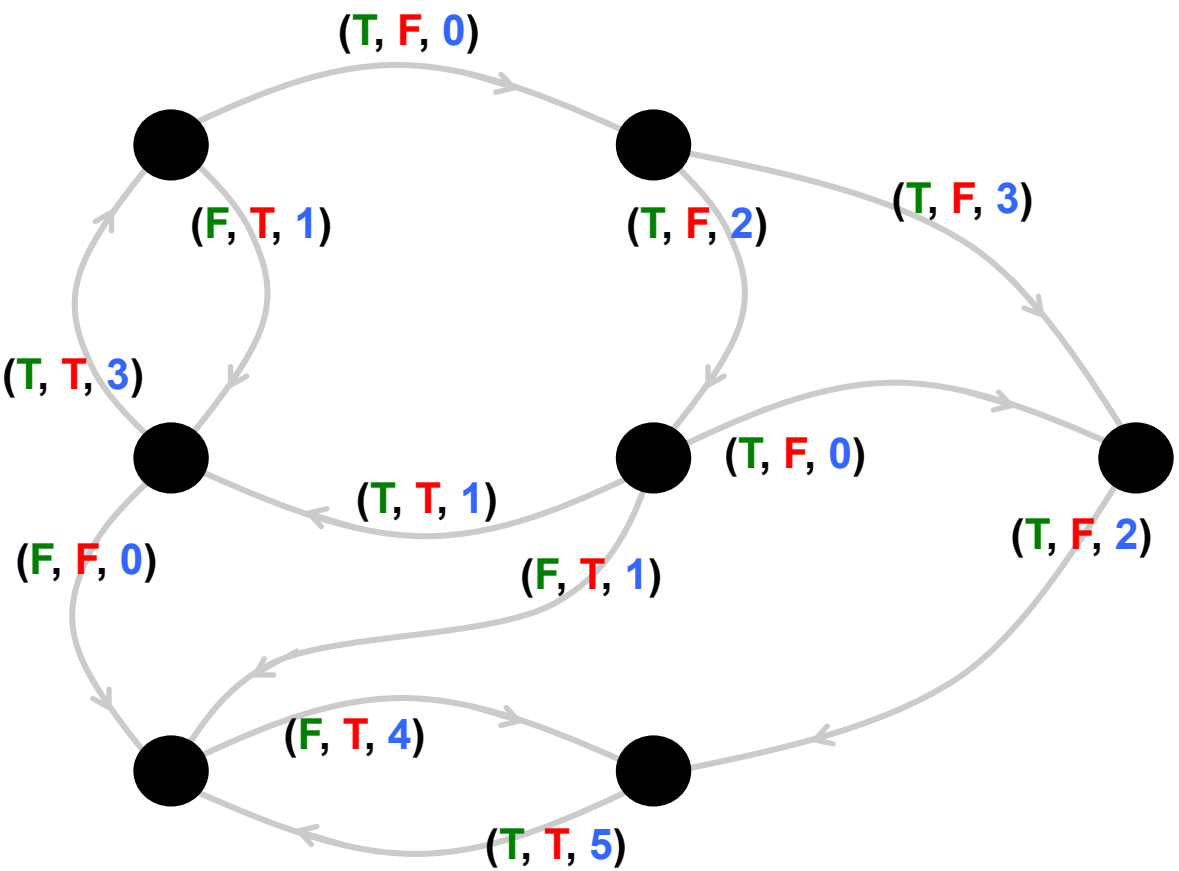
- Vertex types: Person, Phone, Camera, Gene, Pathway
- Edge types: PhoneCall, TextMessage, CoLocation, Sequence Similarity
- Edge attributes: StartTime, EndTime

- Calculate centrality just for emails among engineers sent between times sTime and eTime

```
def onlyEngineers(self):
    return self.position == Engineer

def timedEmail (self, sTime, eTime):
    return ((self.type == email) and
            (self.Time > sTime) and
            (self.Time < eTime))

start = dt.now() - dt.timedelta(days=30)
end = dt.now()

# G denotes the graph
G.addVFilter(onlyEngineers)
G.addEFilter(timedEmail(start, end))

# rank via centrality based on recent
email transactions among engineers
bc = G.rank('approxBC')
```


- Lugowski, **B.**, , Gilbert, Reinhardt. Scalable complex graph analysis with the knowledge discovery toolbox. In *ICASSP*, 2012

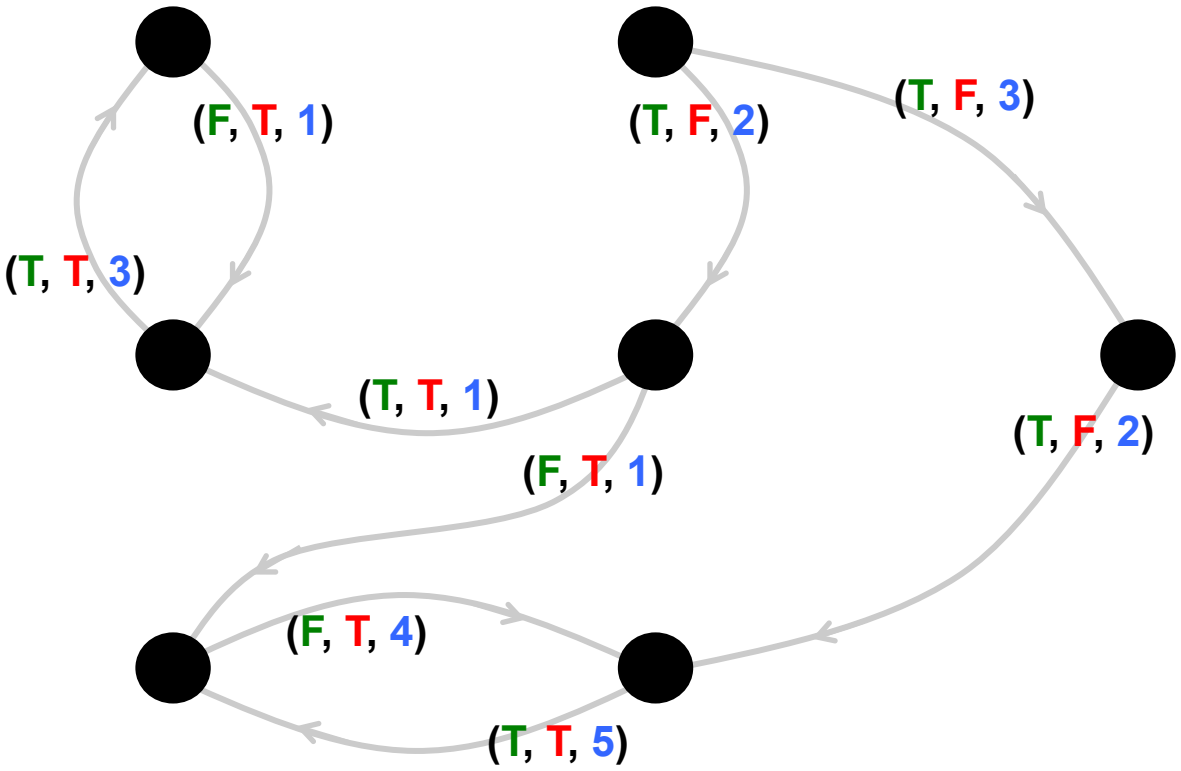
Edge filter illustration




```
class edge_attr:
    isText
    isPhoneCall
    weight
```

Edge filter illustration

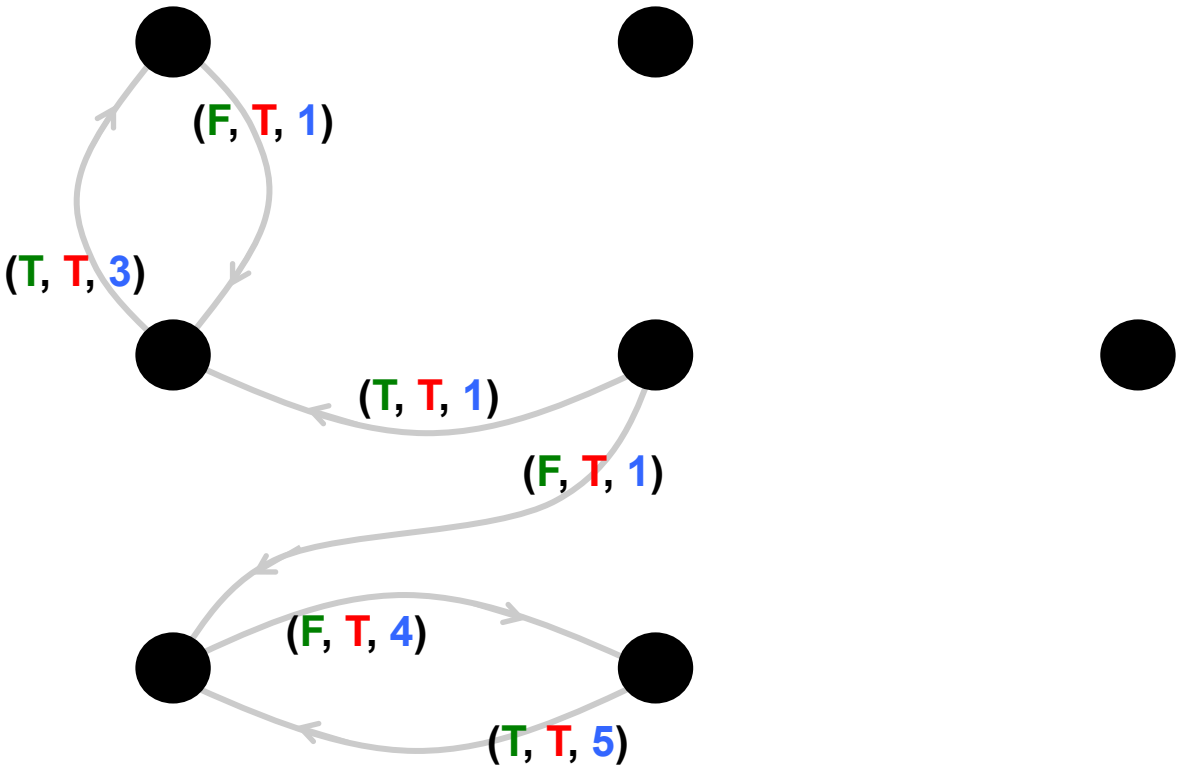
```
G.addEFilter(lambda e: e.weight > 0)
```




```
class edge_attr:
    isText
    isPhoneCall
    weight
```

Edge filter illustration

```
G.addEFilter(lambda e: e.weight > 0)
G.addEFilter(lambda e: e.isPhoneCall)
```




```
class edge_attr:
    isText
    isPhoneCall
    weight
```

Filter options and implementation

- Filter defined as unary predicates, checked in order they were added
- **Each KDT object** maintains **a stack of filter predicates**
- All operations respect filters, enabling **filter-ignorant algorithm design**

On-the-fly filters	Materialized filters
Edges are retained	Edges are pruned on copy
Check predicate on each edge/vertex traversal	Check predicate once on materialization
Cheap but done on each run	Expensive but done once

SEJITS for filter acceleration

Selective Embedded Just-In-Time Specialization

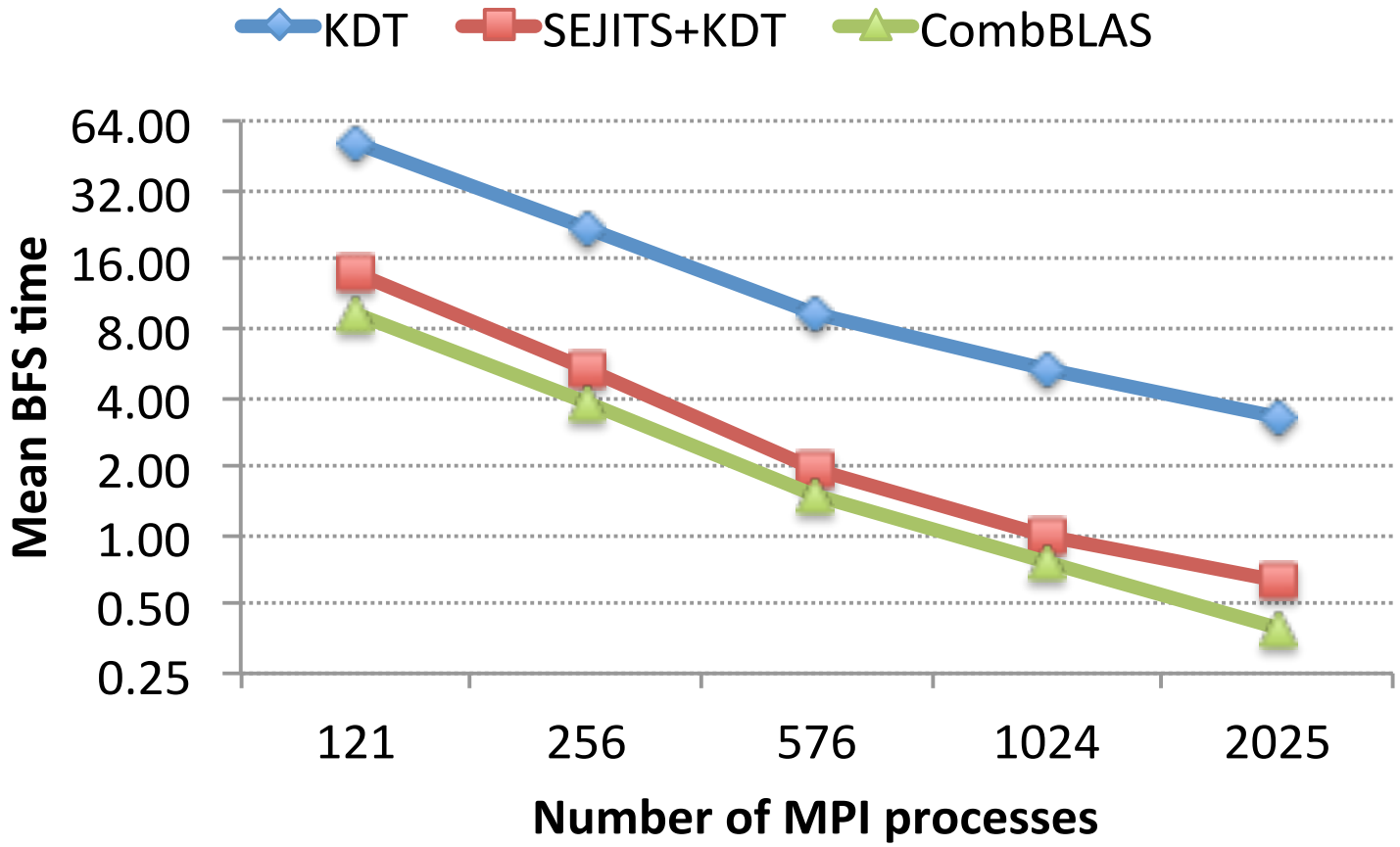
- Take Python code
- Translate it to equivalent C++ code
- Compile with GCC
- Call compiled version instead of Python version

On-the-fly filters need SEJITS for fast execution:

- In plain KDT, filters are pure Python functions.
- Requires a per-vertex or per-edge upcall into Python
- Can be as slow as 80X compared to pure C++

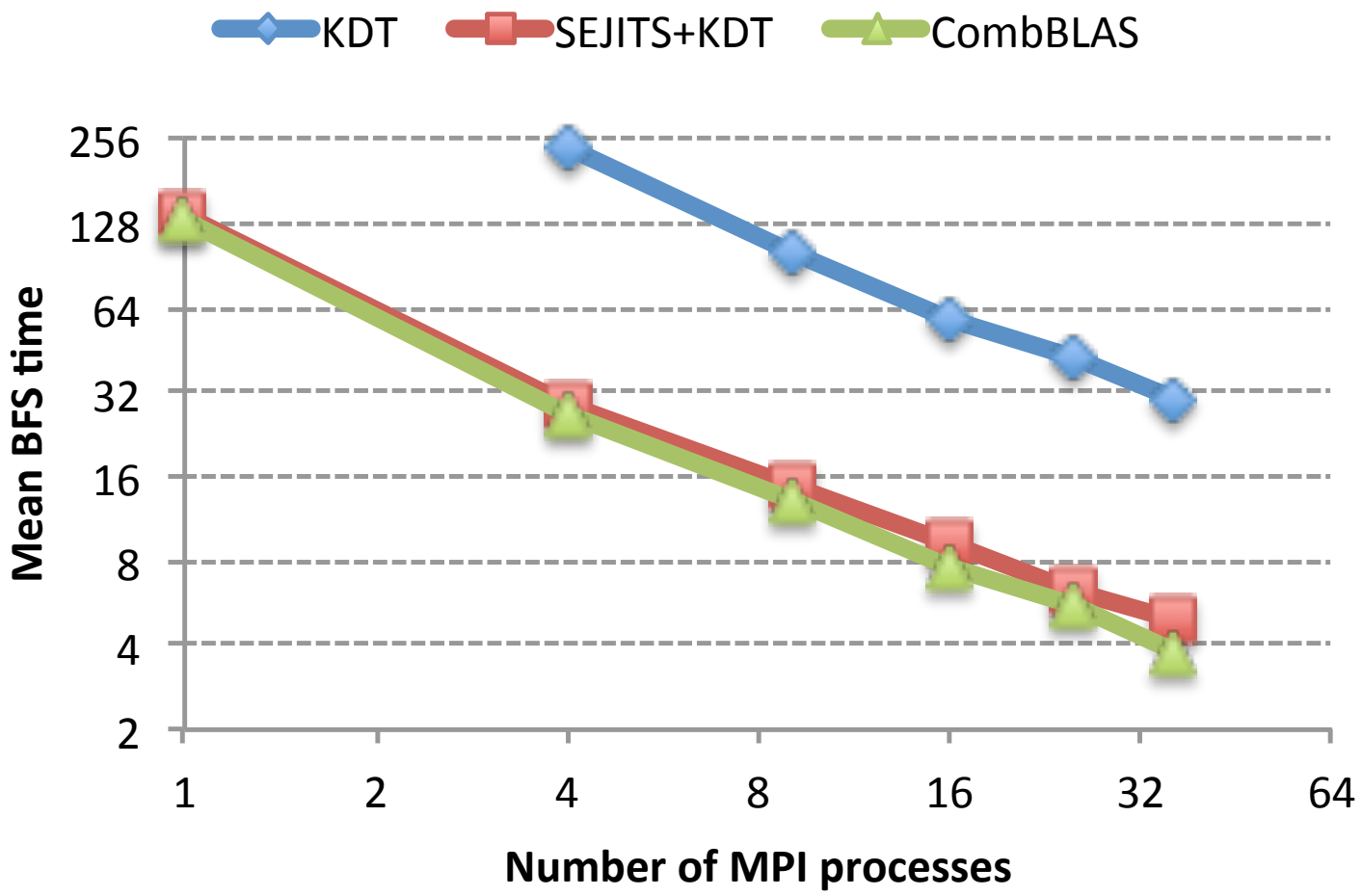
- **B.**, Gilbert, Fox, Kamil, Lugowski, Oliker, Williams. High-Performance Analysis of Filtered Semantic Graphs*, Tech Report UCB/EECS-2012-61 (Extended Abstract at PACT'12)*

Filtered BFS with SEJITS



Time (in seconds) for a single BFS iteration on Scale 25 RMAT (33M vertices, 500M edges) with 10% of elements passing filter. Machine is NERSC's Hopper.

Filtered BFS with SEJITS

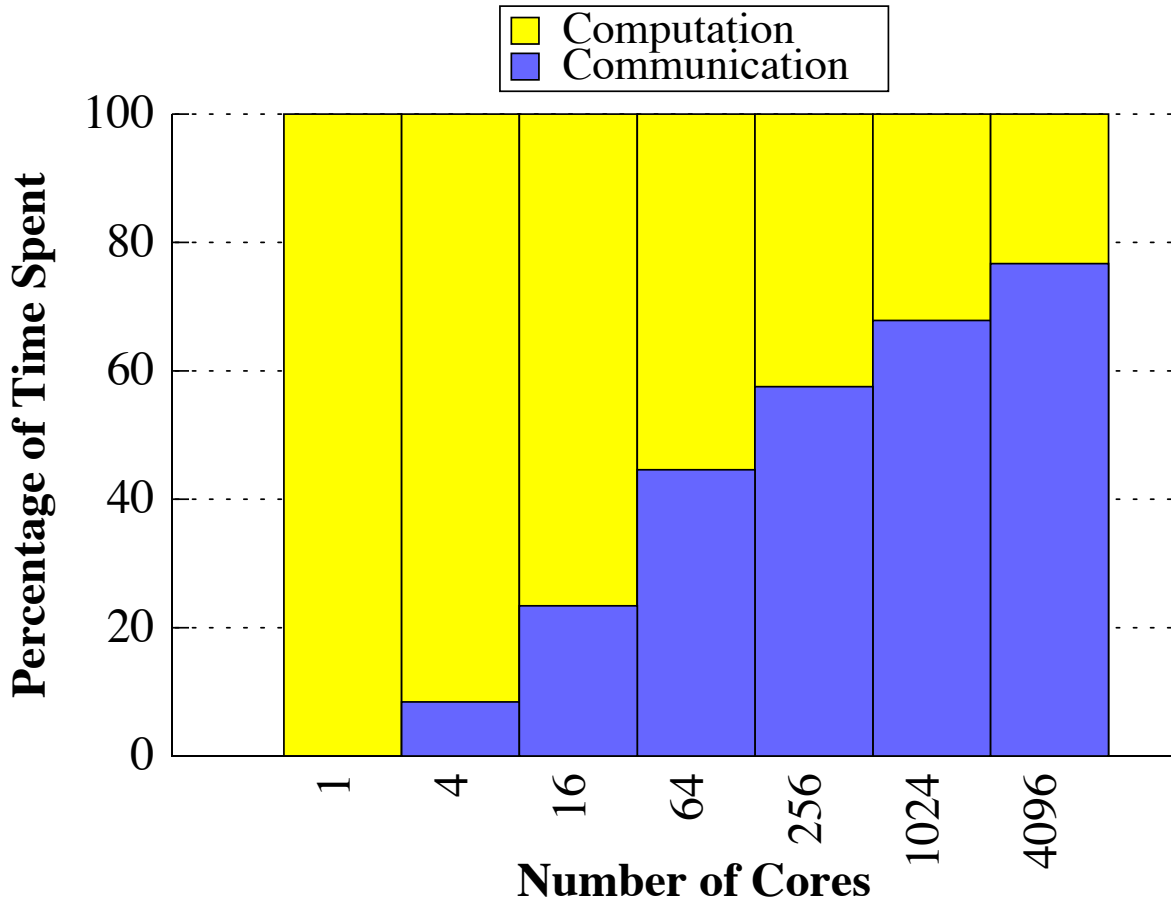


Time (in seconds) for a single BFS iteration on Scale 23 RMAT (8M vertices, 130M edges) with 10% of elements passing filter. Machine is Mirasol.

Outline

- Sustainable graph libraries
- Linear algebraic primitives for graphs
- Sparse triple product for graph subroutines
- Filtered semantic graphs
- **Future challenges/directions**

Challenges #1: Communication

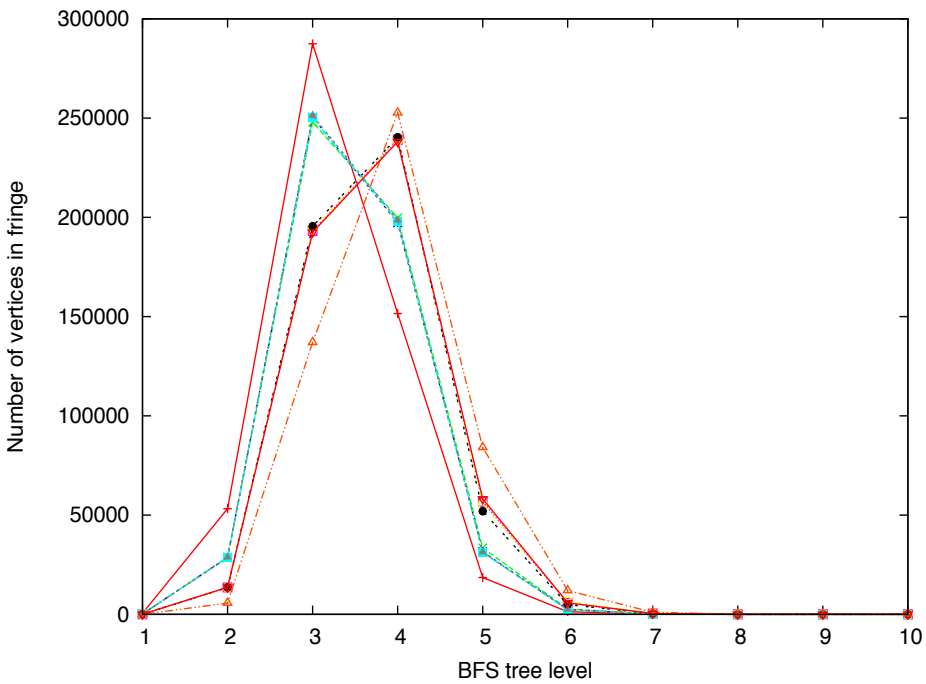


Scale 23 R-MAT **times** restriction operator of order 4

Communication is the bottleneck !

2.5D Algorithms:
Using extra memory (c replicas) reduces communication volume by a factor of $c^{1/2}$ compared to 2D

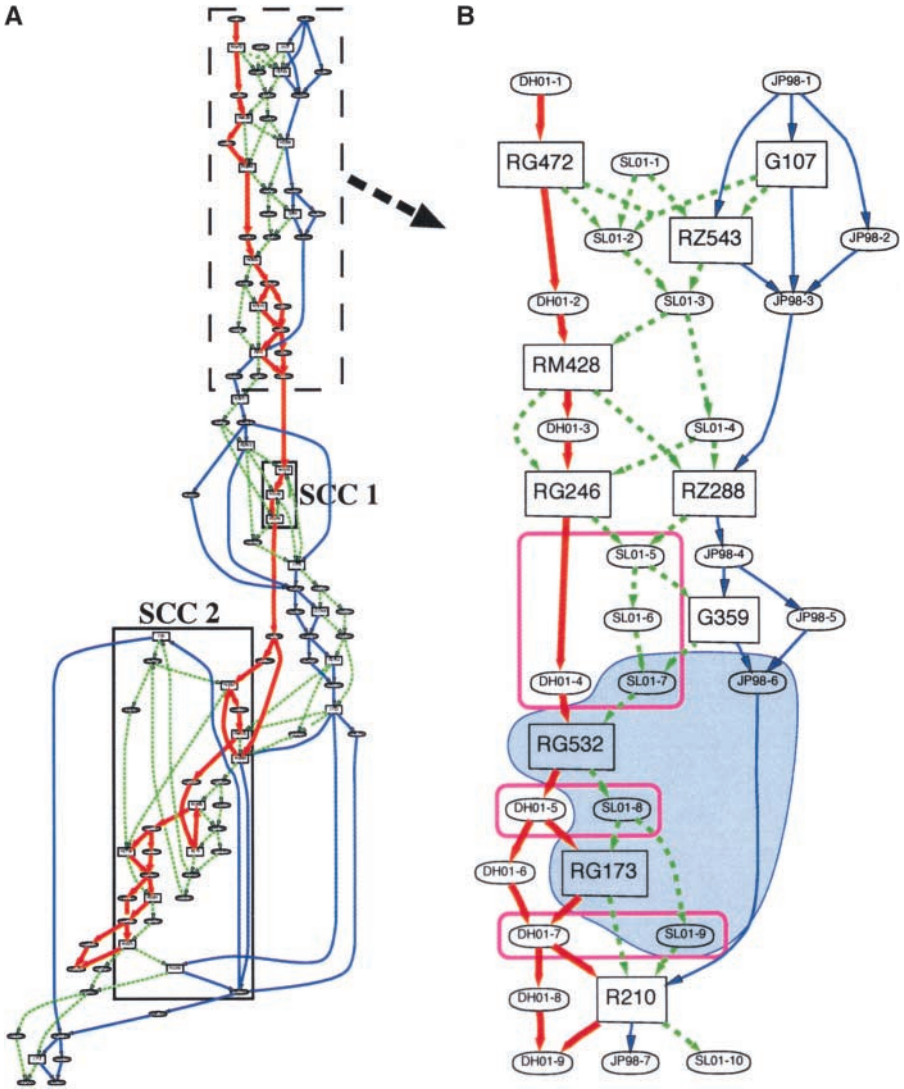
Challenges #2: Data diversity



Low diameter graph (R-MAT)

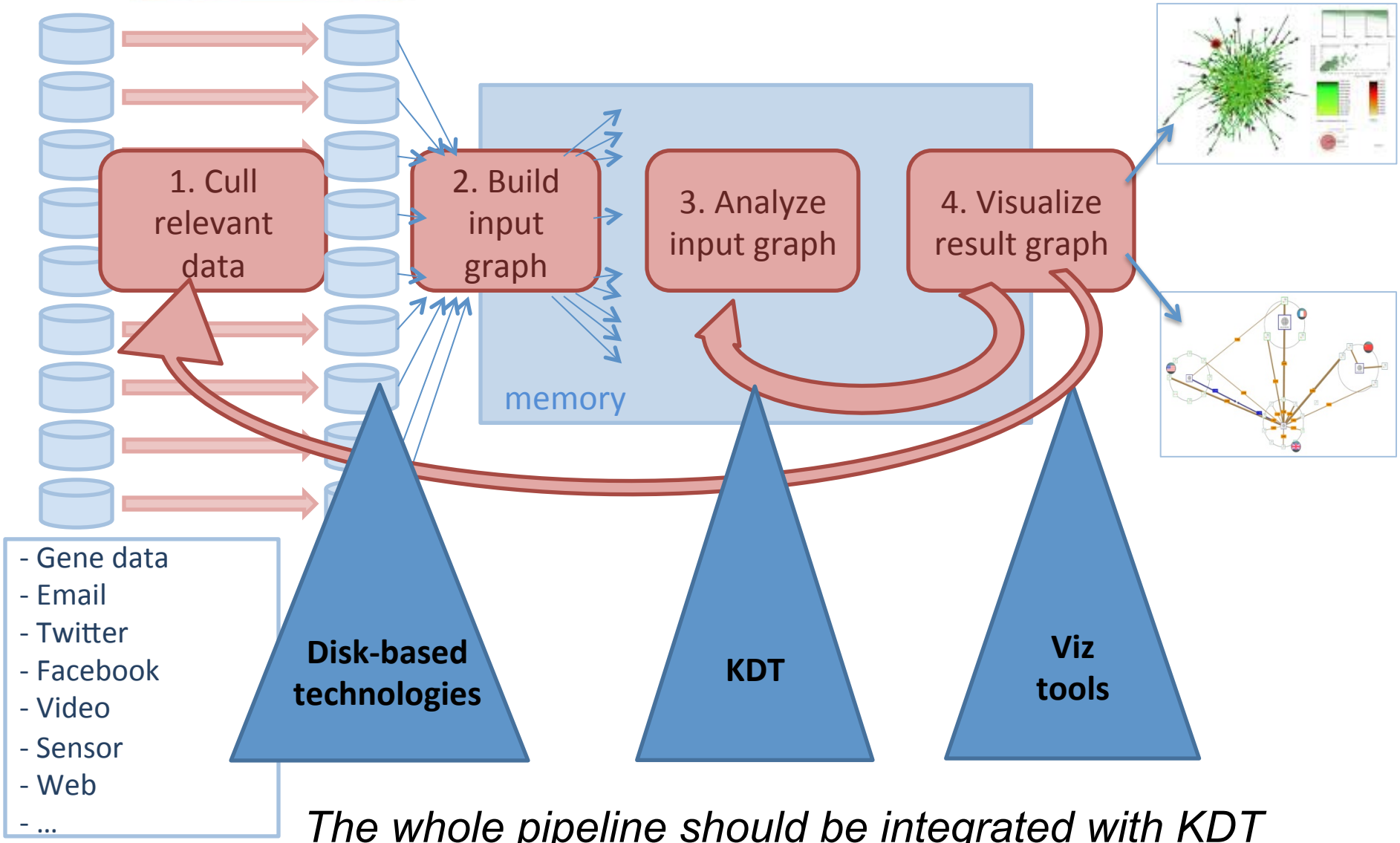
vs.

Long skinny graph (genomics)



Gene linkage map, courtesy Yan et al.

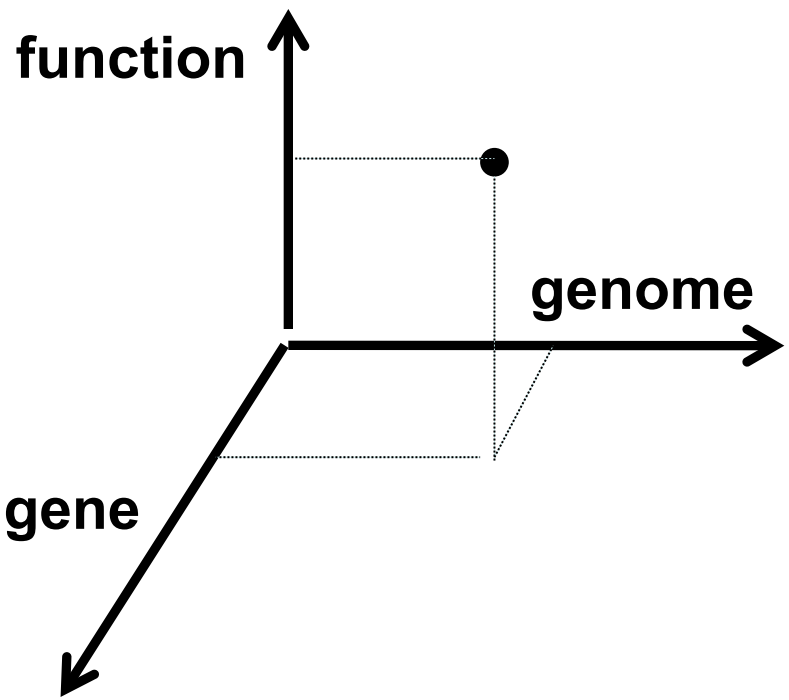
Challenges #3: End to end solution



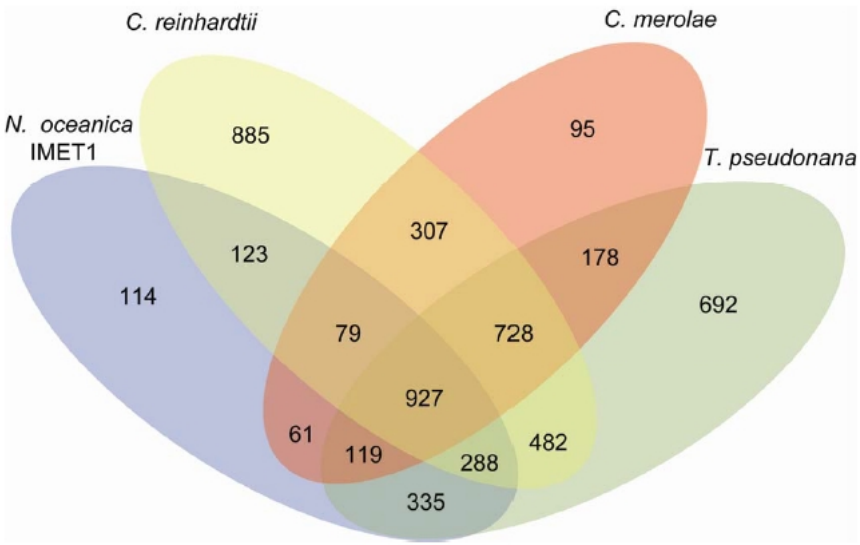
The whole pipeline should be integrated with KDT

Challenges #4: Attack the yet undefined

- High-performance graph analysis should help define yet "undefined" data analysis problems in sciences.
- Key: Collaborating directly with domain scientists.



Example: Pan-genome analysis



Microbial pan-genome, courtesy K. Ning

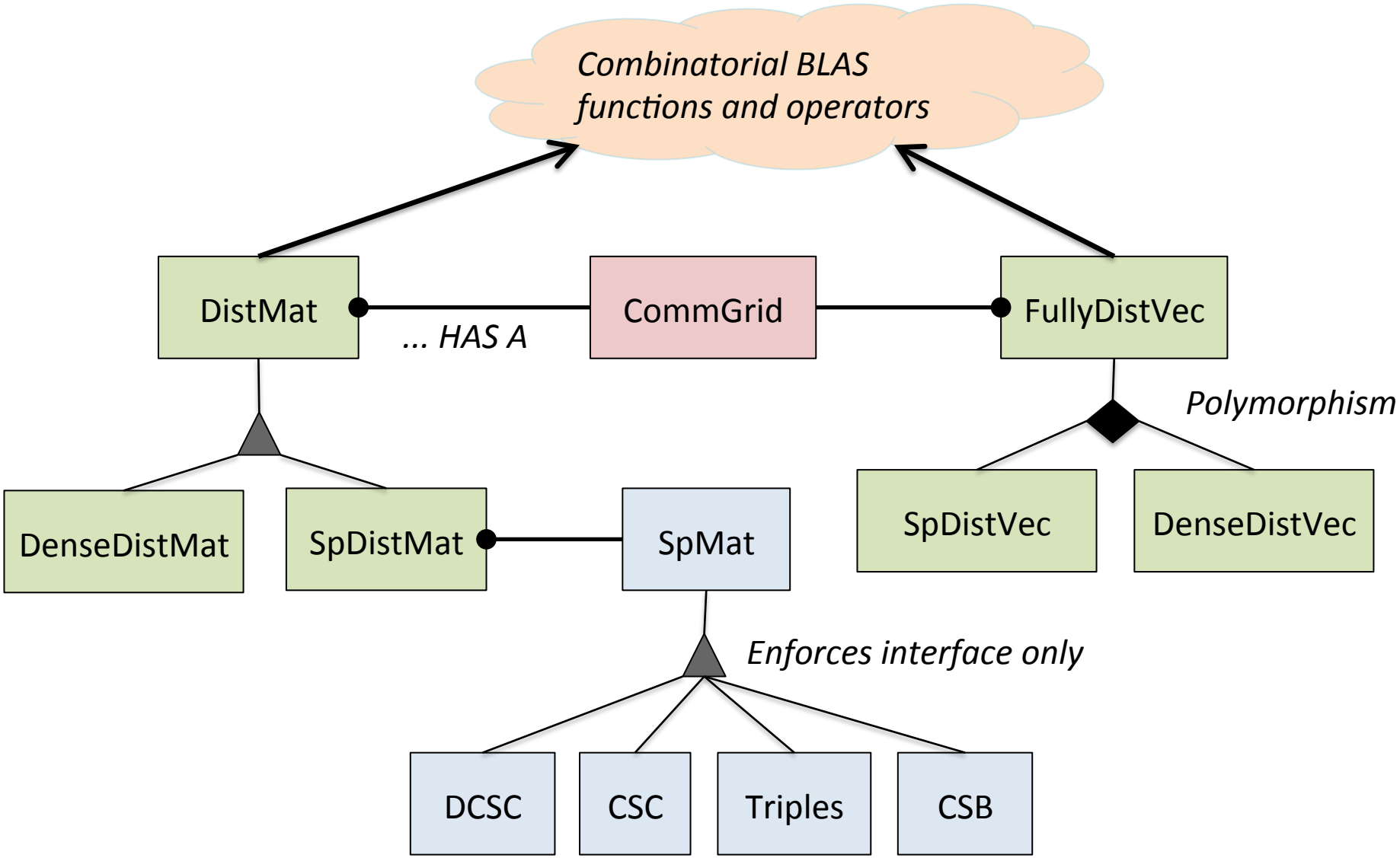
Conclusions

- **KDT + Combinatorial BLAS**: Making parallel graph analysis accessible to domain scientists.
- Layered **software architecture** allows concurrent advances in performance and functionality.
- Carefully chosen and optimized **primitives** are the key to high performance and scalability in graph computations.
- **Linear algebraic primitives** of the Combinatorial BLAS are general enough to be widely useful and compact enough to be heavily optimized.
- High-performance **filtered semantic graph processing** is possible without changes from the graph algorithm developer.

Acknowledgments…

David Bader, Grey Ballard, Scott Beamer, Ceren Budak, James Demmel, Jeremy Fineman, Matteo Frigo, Armando Fox, John Gilbert, Laura Grigori, Bruce Hendrickson, Shoaib Kamil, Jeremy Kepner, Charles Leiserson, Adam Lugowski, Kamesh Madduri, Lenny Oliker, Steve Reinhardt, Viral Shah, Oded Schwartz, Edgar Solomonik, Sam Williams

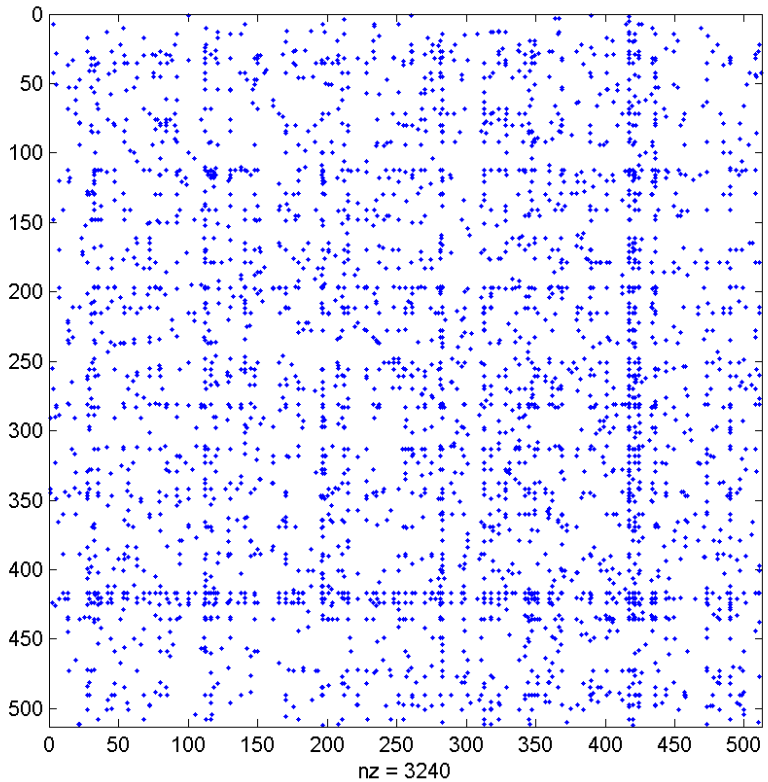
Combinatorial BLAS class hierarchy

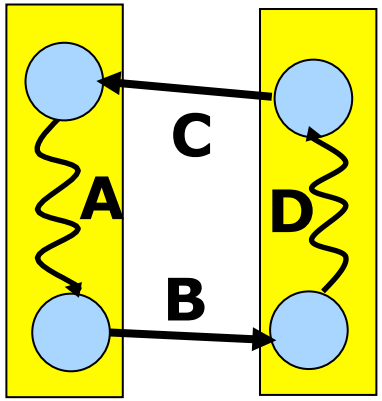


Some Combinatorial BLAS functions

Function	Applies to	Parameters		Returns	Matlab Phrasing
SpGEMM	Sparse Matrix (as friend)	**A**, **B**: trA: trB:	sparse matrices transpose **A** if true transpose **B** if true	Sparse Matrix	$\mathbf{C} = \mathbf{A} * \mathbf{B}$
SpMV	Sparse Matrix (as friend)	**A**: **x**: trA:	sparse matrices sparse or dense vector(s) transpose **A** if true	Sparse or Dense Vector(s)	$\mathbf{y} = \mathbf{A} * \mathbf{x}$
SpEWiseX	Sparse Matrices (as friend)	**A**, **B**: notA: notB:	sparse matrices negate **A** if true negate **B** if true	Sparse Matrix	$\mathbf{C} = \mathbf{A} * \mathbf{B}$
Reduce	Any Matrix (as method)	dim: *binop*:	dimension to reduce reduction operator	Dense Vector	sum(**A**)
SpRef	Sparse Matrix (as method)	**p**: **q**:	row indices vector column indices vector	Sparse Matrix	$\mathbf{B} = \mathbf{A}(\mathbf{p}, \mathbf{q})$
SpAsgn	Sparse Matrix (as method)	**p**: **q**: **B**:	row indices vector column indices vector matrix to assign	none	$\mathbf{A}(\mathbf{p}, \mathbf{q}) = \mathbf{B}$
Scale	Any Matrix (as method)	**rhs**:	any object (except a sparse matrix)	none	Check guiding principles 3 and 4
Scale	Any Vector (as method)	**rhs**:	any vector	none	none
Apply	Any Object (as method)	*unop*:	unary operator (applied to non-zeros)	None	none

Recursive all-pairs shortest paths





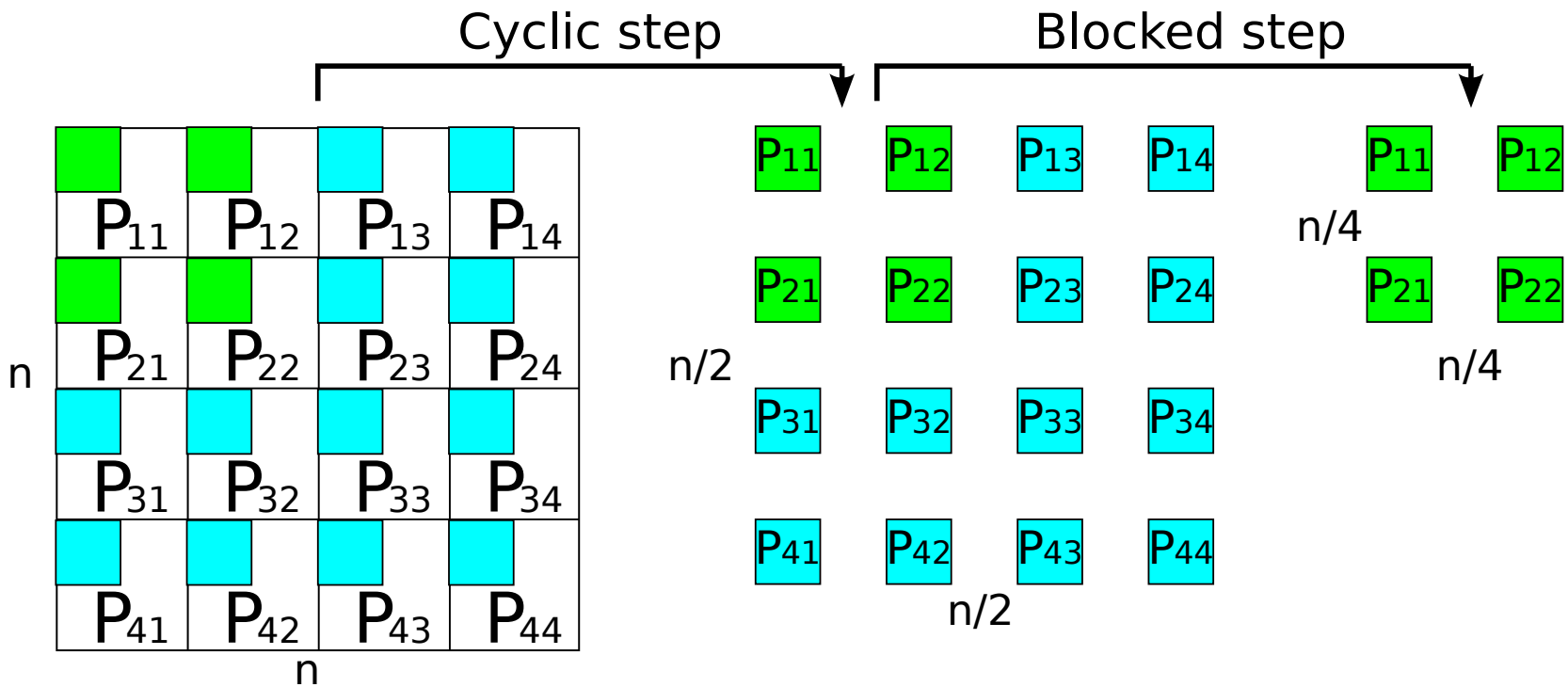
A	B
C	D

+ is "min", **×** is "add"

```
A = A*;      % recursive call
B = AB;  C = CA;
D = D + CB;
D = D*;      % recursive call
B = BD;  C = DC;
A = A + BC;
```

Novel 2.5D APSP algorithm

[Solomonik, B., Demmel; 2012]



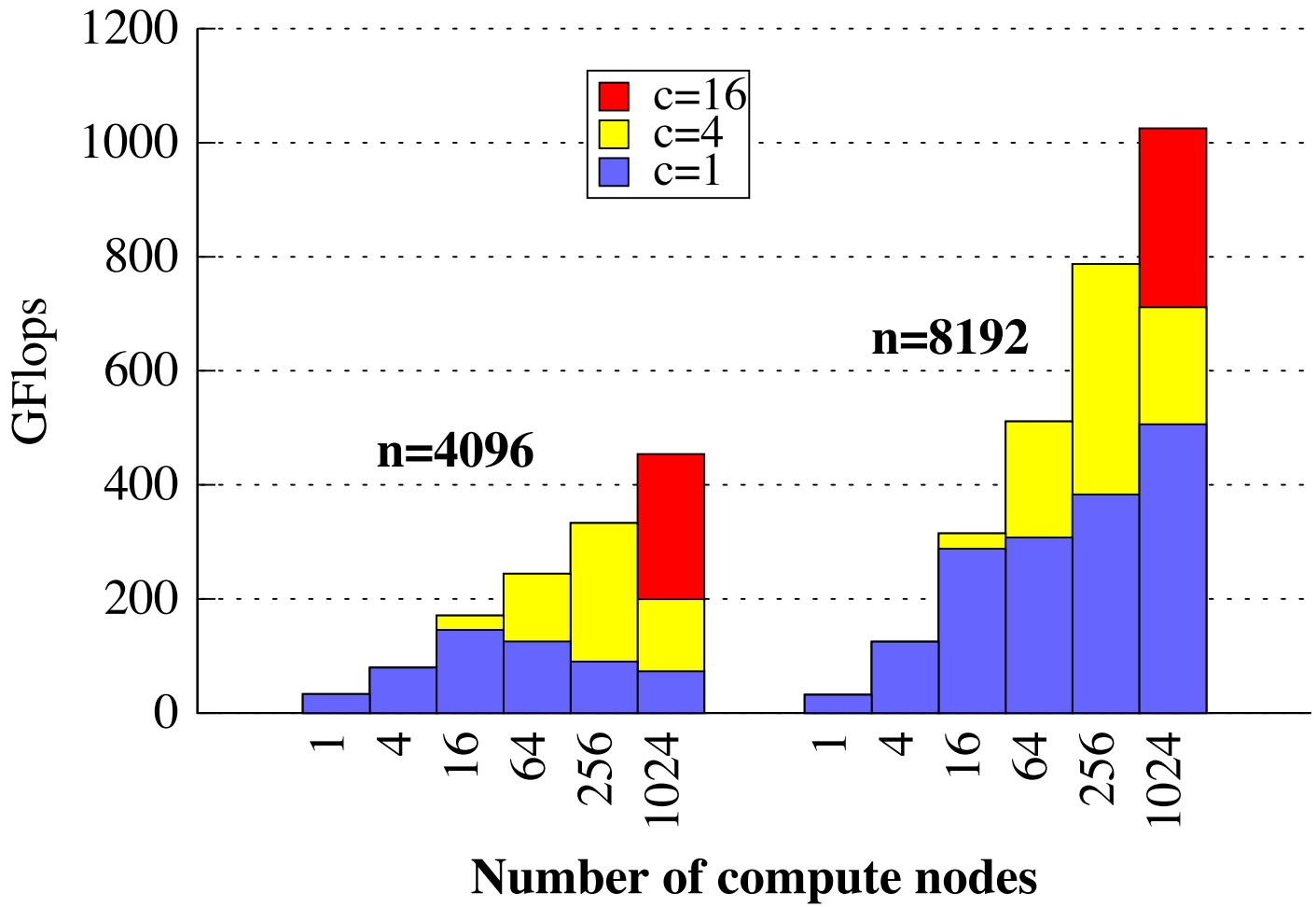
Bandwidth: $W_{\text{bc-2.5D}}(n,p) = O(n^2/\sqrt{cp})$

c: number of replicas

Latency: $S_{\text{bc-2.5D}}(p) = O\left(\sqrt{cp}\log^2(p)\right)$

Optimal for any memory size !

Novel 2.5D APSP algorithm



Complex methods

centrality(‘exactBC’)
centrality(‘approxBC’)

cluster(‘Markov’)

pageRank

Building blocks

DiGraph

- bfsTree,neighbor
- degree,subgraph
- load,UFget
- +, -, sum, scale

HyGraph

- toDiGraph
- load, Ufget
- bfsTree
- degree

SpMatrix

- SpMV
- SpGEMM
- load, eye
- reduce, scale
- +, []

SpVec

- max, norm,sort
- abs, any, ceil
- range, ones
- +,-,*,/,>,==,&,[]

Underlying infrastructure (Combinatorial BLAS)

- SpMV, SpMV_SemiRing
- SpGEMM, SpGEMM_SemiRing

Sparse-matrix classes/methods
(*e.g.*, Apply, EWiseApply, Reduce)