UCSB

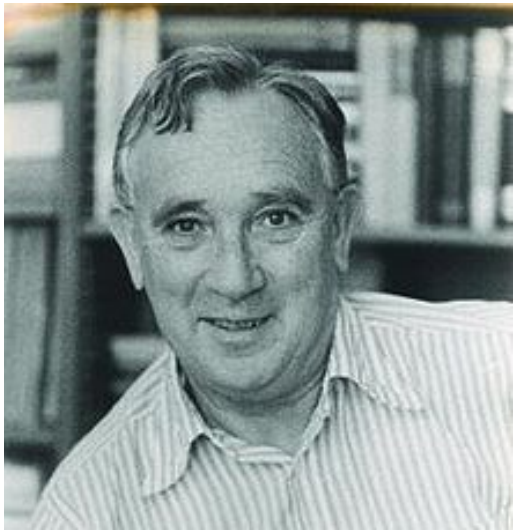
Graph Algorithms in the Language of Linear Algebra: How did we get here, where do we go next?

John R. Gilbert
University of California, Santa Barbara

IPDPS Graph Algorithms Building Blocks
May 21, 2018

Support: Intel, Microsoft, DOE Office of Science, NSF

George Pólya on how to give a mathematical talk

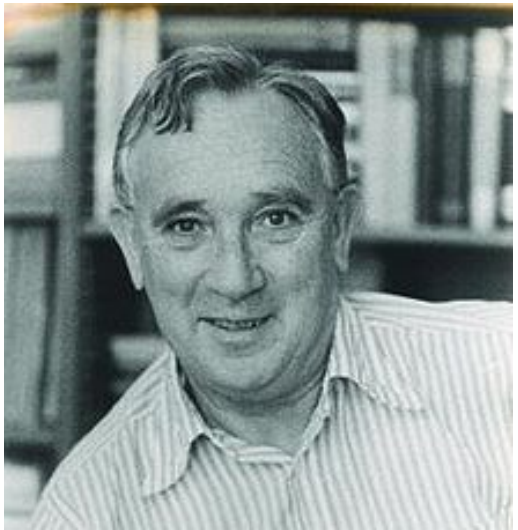


(as described by John Todd)

"Pólya's recipe was as follows: The first quarter should be understandable to absolutely everyone, the second quarter should include kind words about your friends (especially those in the audience), and then it doesn't matter what you say in the last half hour."

UCSB

George Pólya on how to give a mathematical talk



(as described by John Todd)

"Pólya's recipe was as follows: The first quarter should be understandable to absolutely everyone, the second quarter should include kind words about your friends (especially those in the audience), and then it doesn't matter what you say in the last half hour."

"I [Todd] adjust this by adding, sit down after a quarter hour."

UCSB

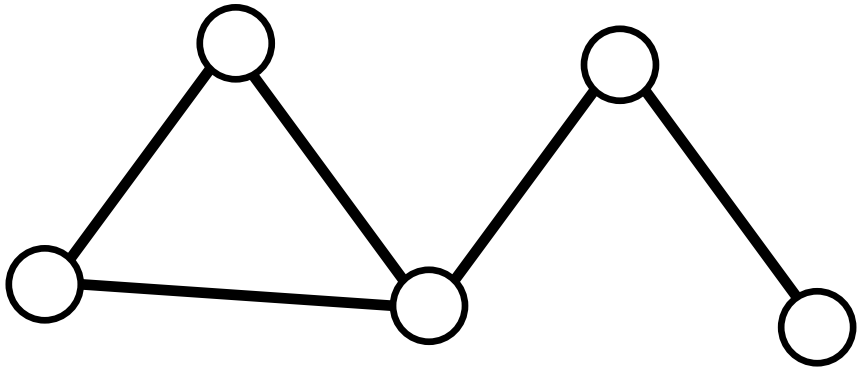
Prehistory

In the year 1961 ...

UCSB

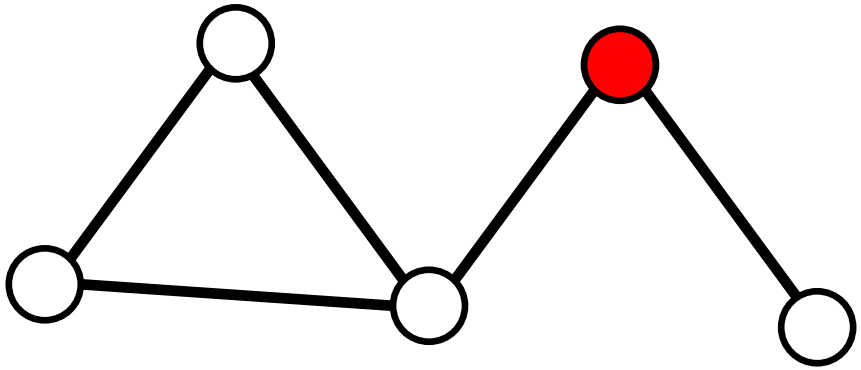
Prehistory: A 1-person game on graphs

[S. Parter 1961]



UCSB

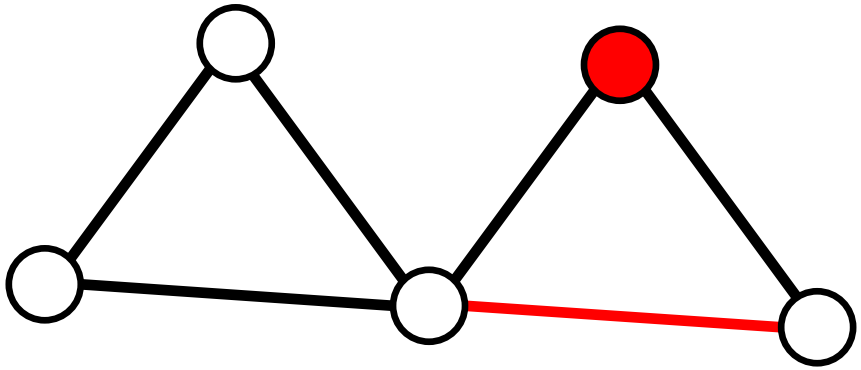
Prehistory: A 1-person game on graphs



- Mark a vertex.

UCSB

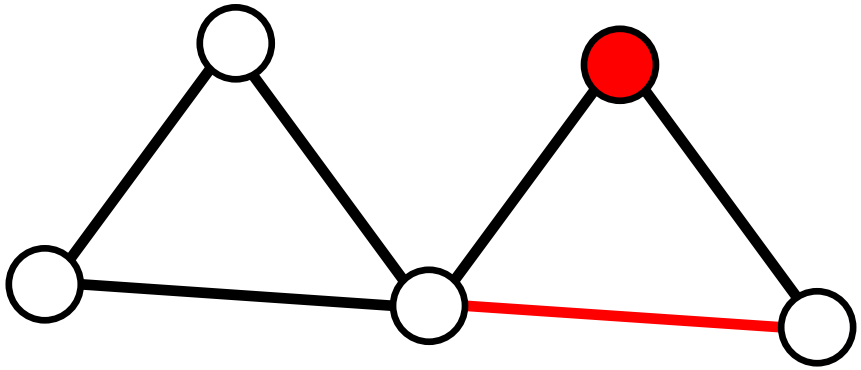
Prehistory: A 1-person game on graphs



- Mark a vertex.
- Connect its unmarked neighbors.

UCSB

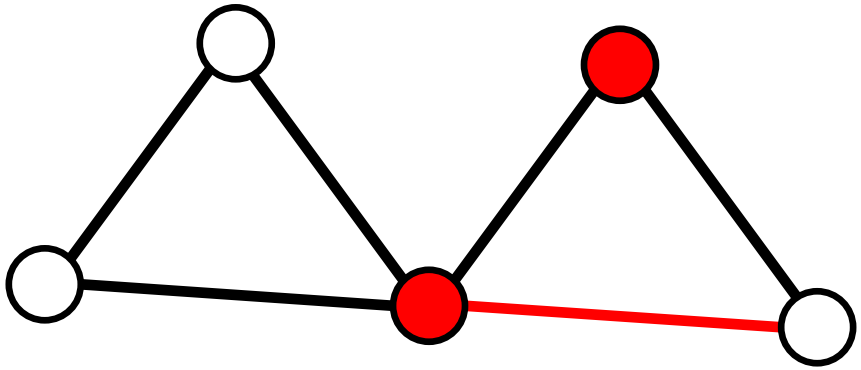
Prehistory: A 1-person game on graphs



- Mark a vertex.
- Connect its unmarked neighbors.
- Repeat.

UCSB

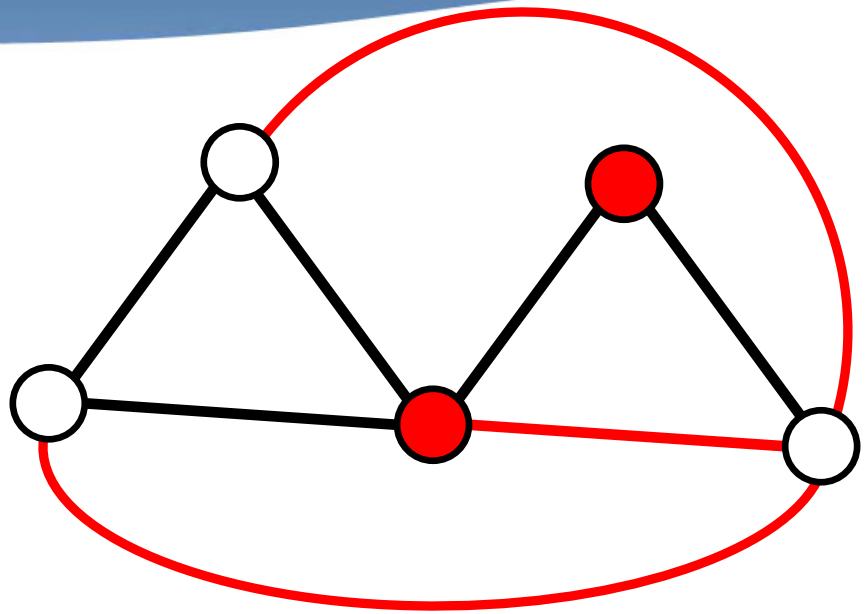
Prehistory: A 1-person game on graphs



- Mark a vertex.
- Connect its unmarked neighbors.
- Repeat.

UCSB

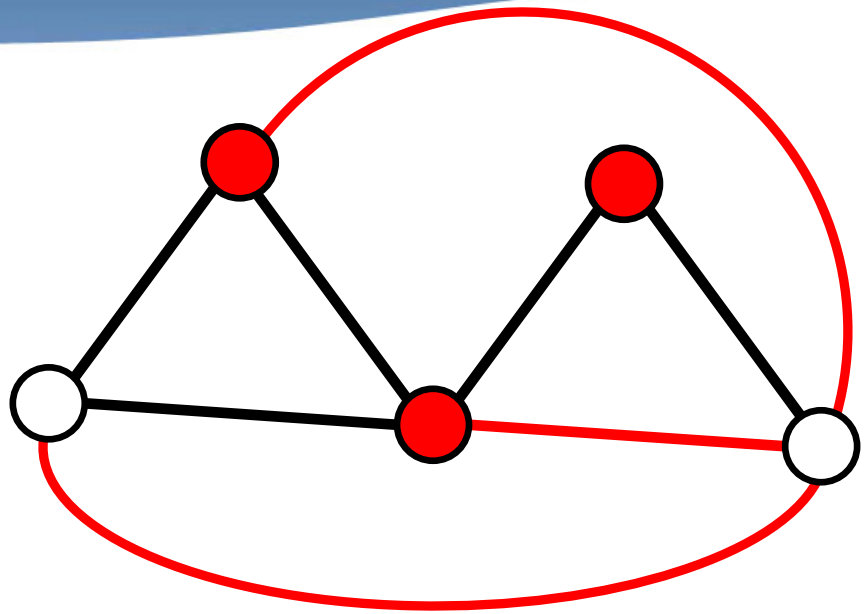
Prehistory: A 1-person game on graphs



- Mark a vertex.
- Connect its unmarked neighbors.
- Repeat.

UCSB

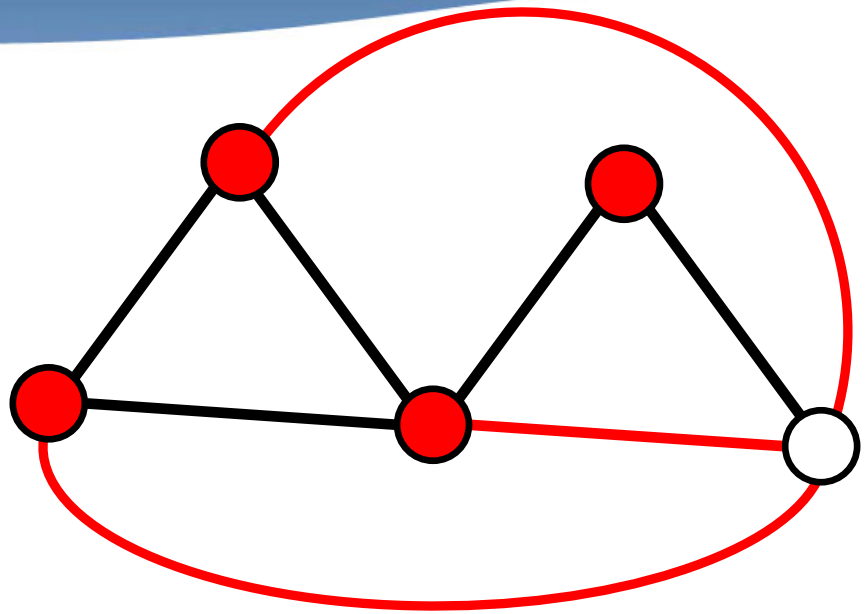
Prehistory: A 1-person game on graphs



- Mark a vertex.
- Connect its unmarked neighbors.
- Repeat.

UCSB

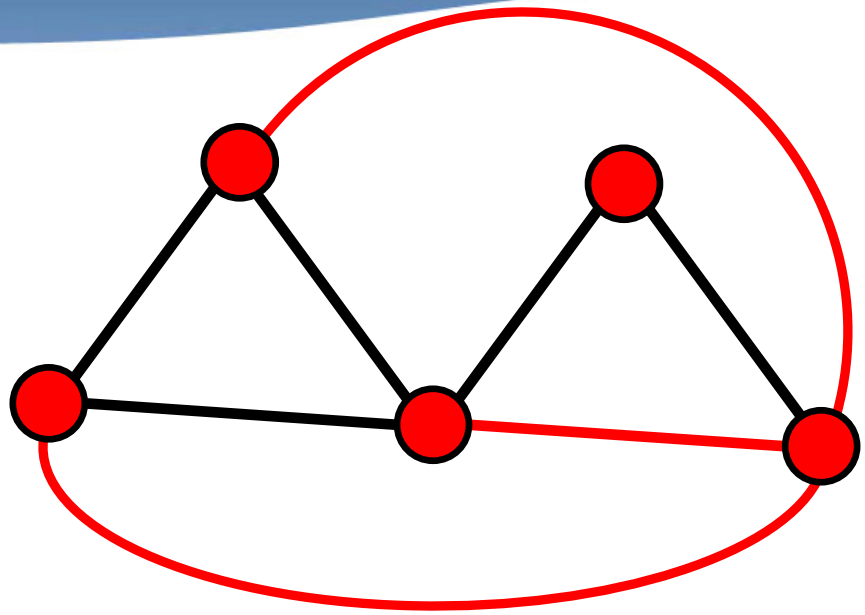
Prehistory: A 1-person game on graphs



- Mark a vertex.
- Connect its unmarked neighbors.
- Repeat.

UCSB

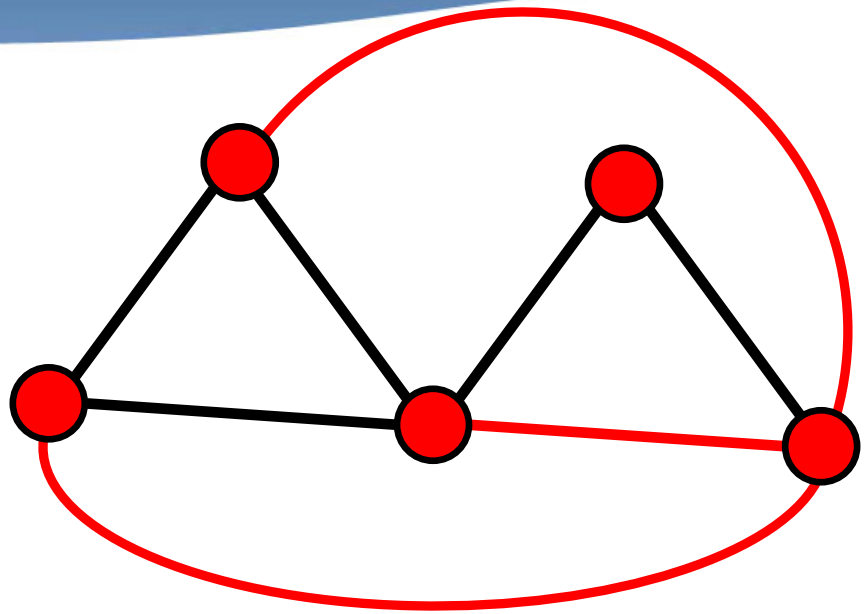
Prehistory: A 1-person game on graphs



- Mark a vertex.
- Connect its unmarked neighbors.
- Repeat.

UCSB

Prehistory: A 1-person game on graphs

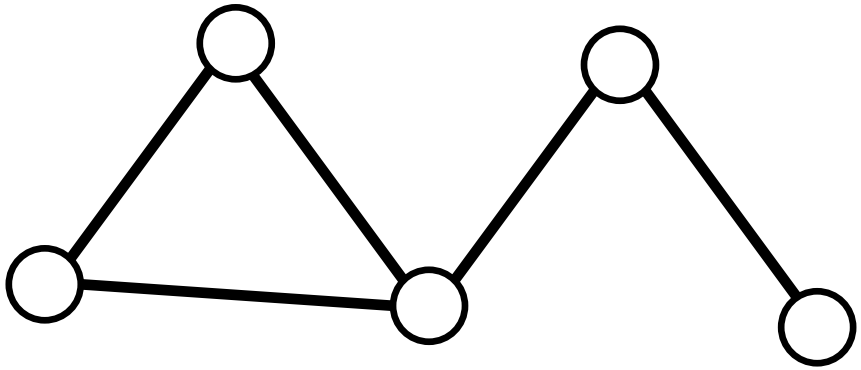


- Mark a vertex.
- Connect its unmarked neighbors.
- Repeat.

Goal: End up with as few edges as possible.

UCSB

Prehistory: A 1-person game on graphs

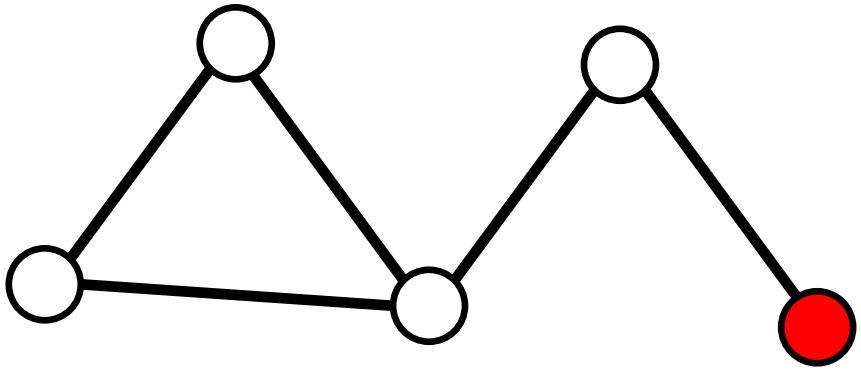


- Mark a vertex.
- Connect its unmarked neighbors.
- Repeat.

Goal: End up with as few edges as possible.

UCSB

Prehistory: A 1-person game on graphs

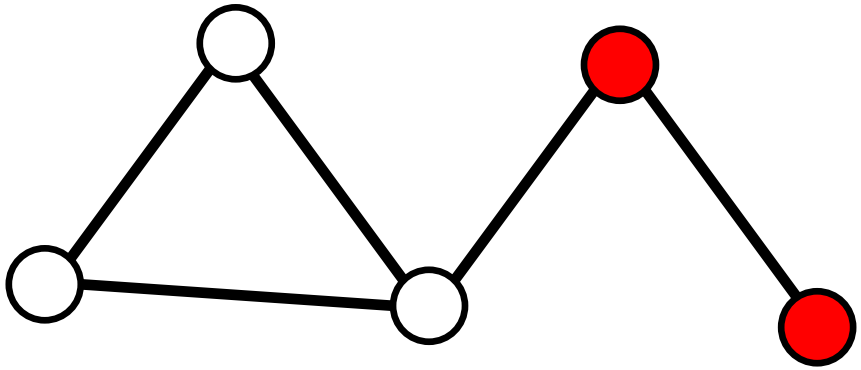


- Mark a vertex.
- Connect its unmarked neighbors.
- Repeat.

Goal: End up with as few edges as possible.

UCSB

Prehistory: A 1-person game on graphs

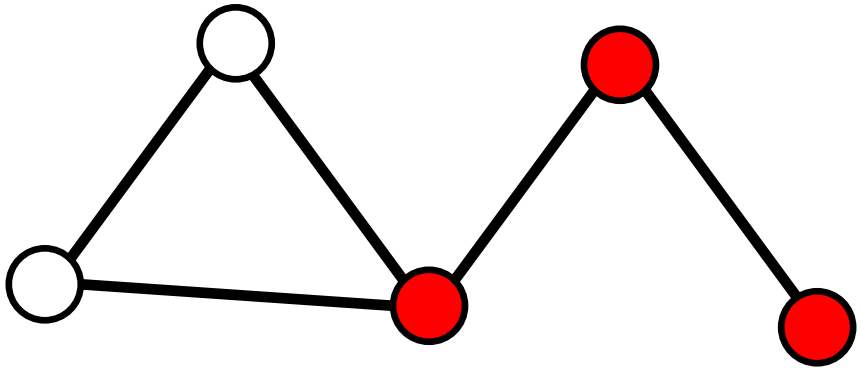


- Mark a vertex.
- Connect its unmarked neighbors.
- Repeat.

Goal: End up with as few edges as possible.

UCSB

Prehistory: A 1-person game on graphs

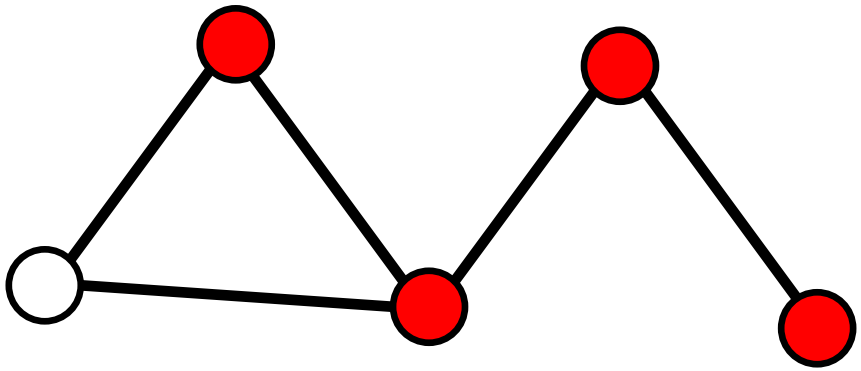


- Mark a vertex.
- Connect its unmarked neighbors.
- Repeat.

Goal: End up with as few edges as possible.

UCSB

Prehistory: A 1-person game on graphs

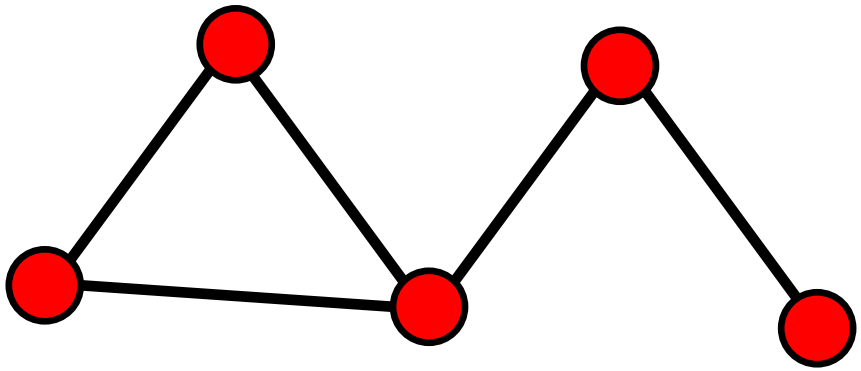


- Mark a vertex.
- Connect its unmarked neighbors.
- Repeat.

Goal: End up with as few edges as possible.

UCSB

Prehistory: A 1-person game on graphs



- Mark a vertex.
- Connect its unmarked neighbors.
- Repeat.

Goal: End up with as few edges as possible.

Vertex elimination game (or chordal completion)

[Parter, Rose]

Repeat:

Choose a vertex v and mark it;

Add edges between unmarked neighbors of v;

Until: Every vertex is marked

Goal: End up with as few edges as possible.

- Best play is NP-complete *[Yannakakis 1981]*
- The final graph is always *chordal* (every cycle has a shortcut edge).
- Perfect play is possible iff the initial graph is chordal.
- Changing "fewest edges" to "smallest complete subgraph" gives the graph's *treewidth*, which shows up in lots of graph algorithms.

Combinatorics in the service of linear algebra

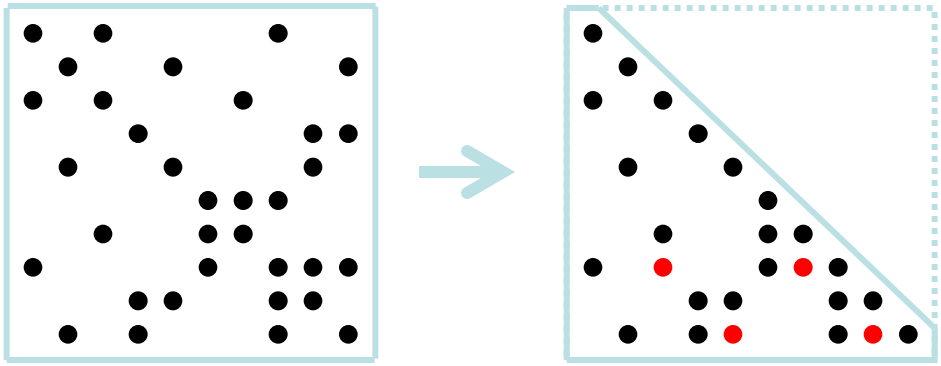
"I observed that most of the coefficients in our matrices were zero; i.e., the nonzeros were 'sparse' in the matrix, and that typically the triangular matrices associated with the forward and back solution provided by Gaussian elimination would remain sparse if pivot elements were chosen with care"

- Harry Markowitz, describing the 1950s work on portfolio theory that won the 1990 Nobel Prize for Economics

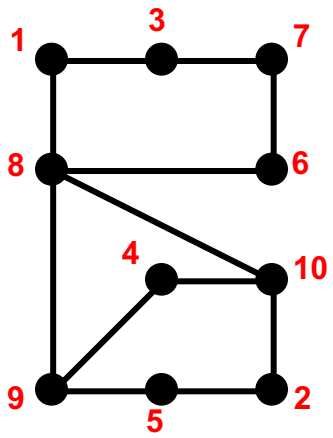
UCSB

Cholesky factorization: $A = LL^T$

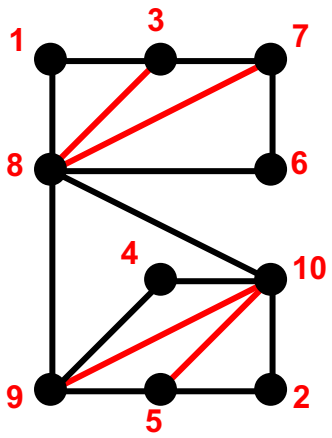
[Parter, Rose]



Fill: new nonzeros in factor



G(A)



$G^+(A)$
[chordal]

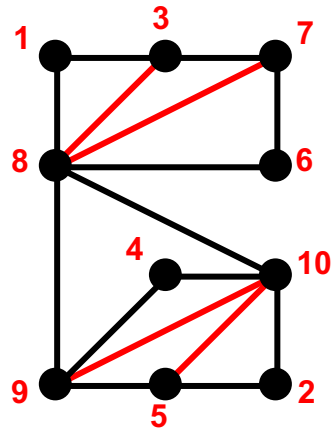
Symmetric Gaussian elimination:

for j = 1 to n
    add edges between j's higher-numbered neighbors

UCSB

Complexity measures for chordal completion

$G^+(A)$



Elimination degree:

d_j = # higher neighbors of j in G^+

d = (2, 2, 2, 2, 2, 2, 1, 2, 1, 0)

- Nonzeros = edges = $\sum_j d_j$ (moment 1)
- Work = flops = $\sum_j (d_j)^2$ (moment 2)
- Front size ~ fast memory = $\max_j d_j$ (moment ∞)

(minimum possible front size is the same as treewidth)

UCSB

Aside: Matrix structure prediction

- Computing the nonzero structure of Cholesky factor L is much cheaper than computing L itself.
- Cost to compute nnz(L) is almost linear in nnz(A). *[G, Ng, Peyton]*

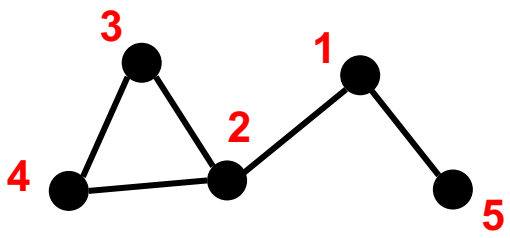
Aside: Matrix structure prediction

- Computing the nonzero structure of Cholesky factor L is much cheaper than computing L itself.
- Cost to compute nnz(L) is almost linear in nnz(A). *[G, Ng, Peyton]*
- Not so for sparse matrix product (SpGEMM); computing nnz(B*C) seems to be as hard as computing B*C.

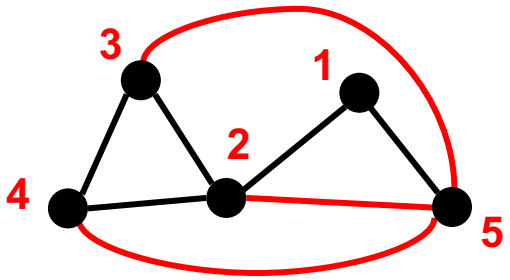
Aside: Matrix structure prediction

- Computing the nonzero structure of Cholesky factor L is much cheaper than computing L itself.
- Cost to compute nnz(L) is almost linear in nnz(A). *[G, Ng, Peyton]*
- Not so for sparse matrix product (SpGEMM); computing nnz(B*C) seems to be as hard as computing B*C.
- Can *estimate* nnz(B*C) accurately in time linear in nnz(B, C)! *[E. Cohen 1998]*
- Lots of cool recent work on sampling algorithms to estimate statistics of matrix functions.

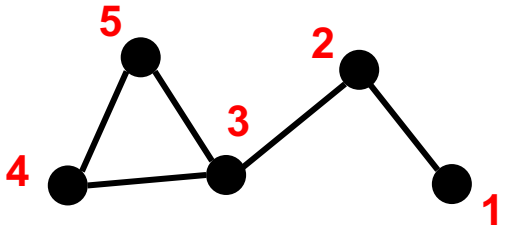
Orderings for sparse Gaussian elimination



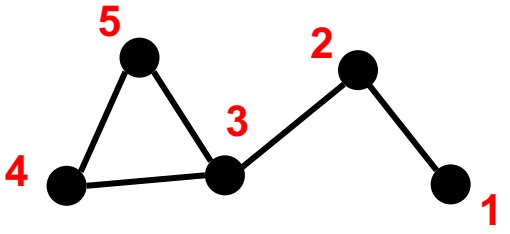
$$Ax = b$$



$$A = L_1 L_1^T$$



$$(PAP^T)\,(Px) = (Pb)$$

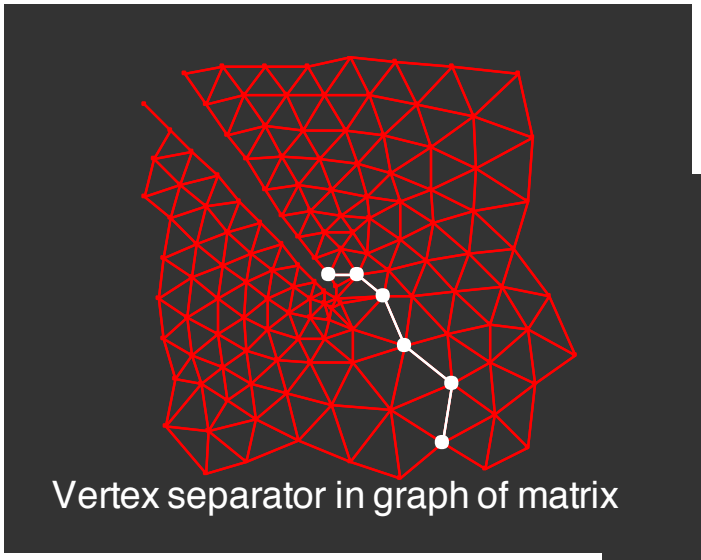


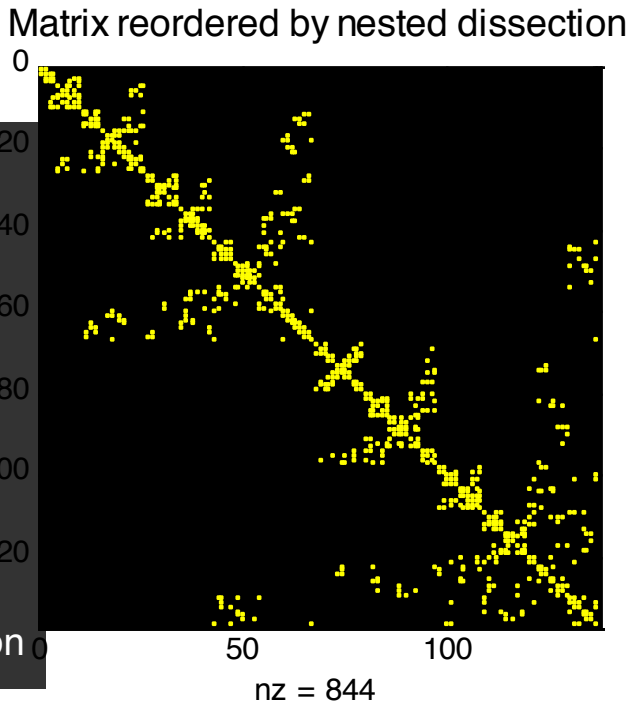
$$PAP^T = L_2 L_2^T$$

UCSB

Nested dissection and graph partitioning

[George 1973, many extensions]





- Heuristic: Find small vertex separator, put it last, recurse on subgraphs
- Theory: Approx optimal separators => approx optimal fill
- Practice: Lots of work on heuristics for graph partitioning!

UCSB

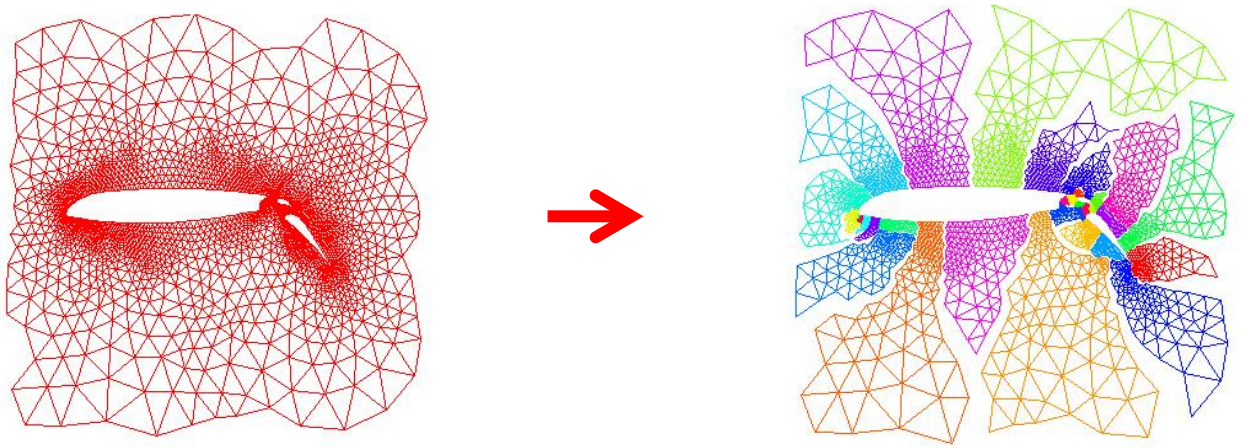
Prehistory: Graph algorithms for sparse matrices

Many, many graph algorithms have been used, invented, implemented at large scale for sparse matrix computation:

- Symmetric problems: elimination tree, nonzero structure prediction, sparse triangular solve, sparse matrix-matrix multiplication, min-height etree, …
- Nonsymmetric problems: sparse triangular solve, bipartite matching (weighted and unweighted), Dulmage-Mendelsohn decomposition / strong components, …
- Iterative methods: graph partitioning again, independent set, low-stretch spanning trees, …

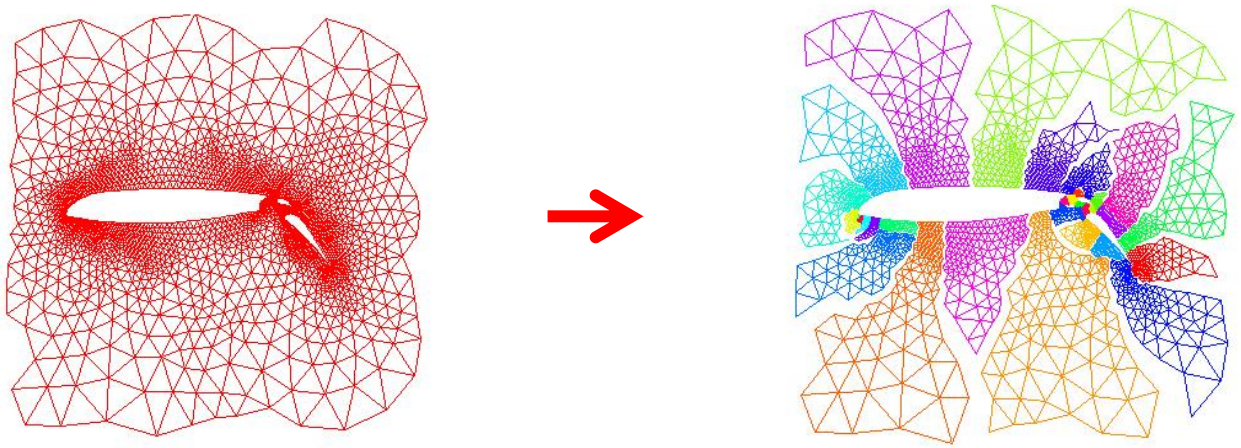
In the year 1992 ...

History: Mesh partitioning for scientific computing, circa 1992



- Both for nested dissection and for parallel sparse matvec
- Spectral partitioning: Laplacian eigenvectors
- Recursive coarsening: Chaco *[Hendrickson/Leland]*, Metis *[Karypis/Kumar]*
- ...

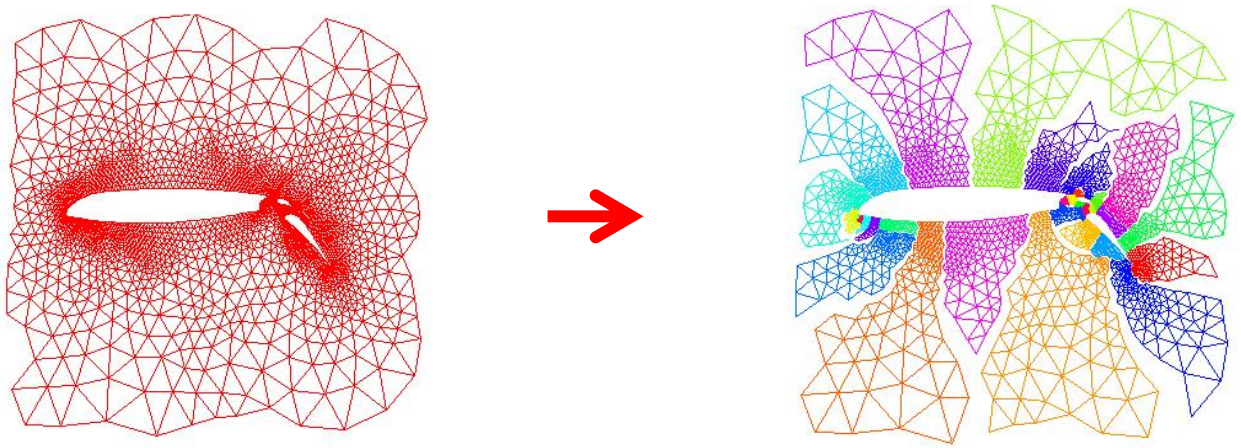
History: Mesh partitioning for scientific computing, circa 1992



- Both for nested dissection and for parallel sparse matvec
- Spectral partitioning: Laplacian eigenvectors
- Recursive coarsening: Chaco *[Hendrickson/Leland]*, Metis *[Karypis/Kumar]*
- Geometric partitioning: Shang-Hua Teng's PhD thesis **...**

UCSB

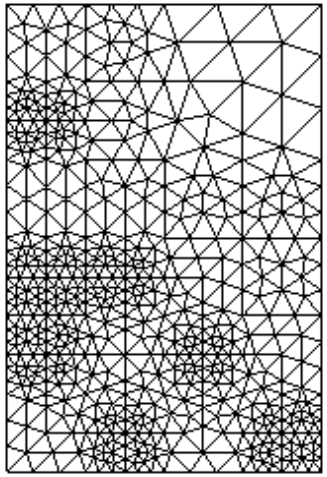
History: Mesh partitioning for scientific computing, circa 1992



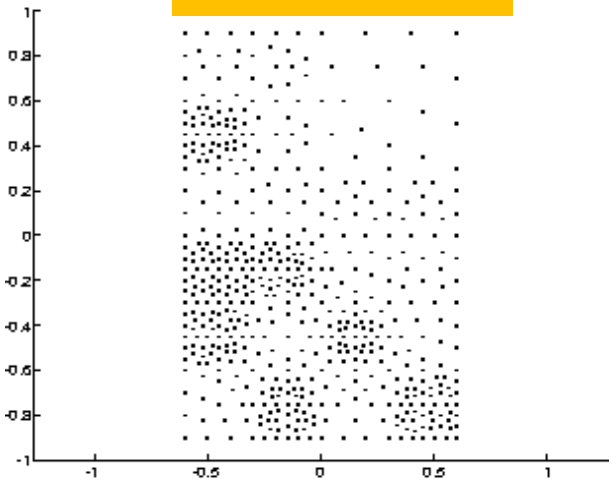
- Both for nested dissection and for parallel sparse matvec
- Spectral partitioning: Laplacian eigenvectors
- Recursive coarsening: Chaco *[Hendrickson/Leland]*, Metis *[Karypis/Kumar]*
- Geometric partitioning: Shang-Hua Teng's PhD thesis **...**
- **...** and sparse matrices had just been added to Matlab **...**

Geometric partitioning in Matlab *[G, Miller, Teng]*

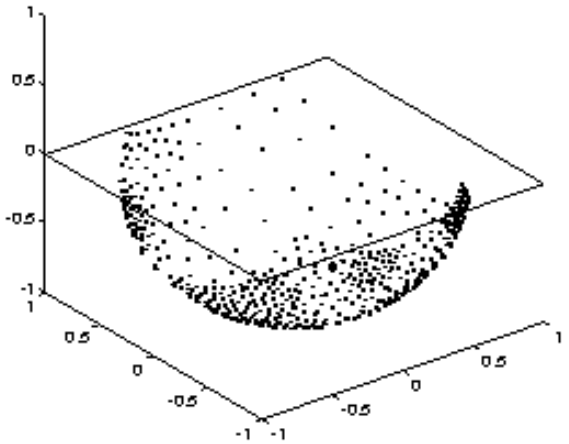
1. Original Mesh



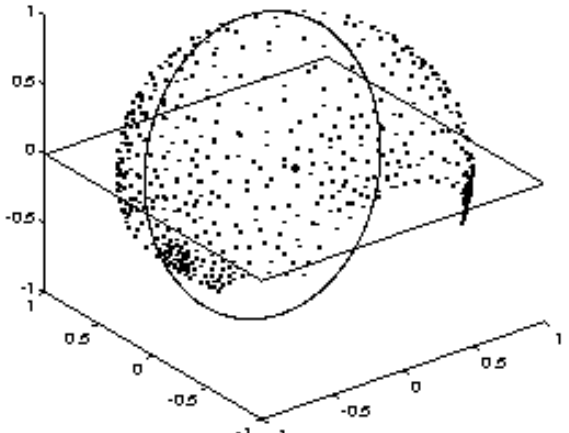
2. Mesh Points



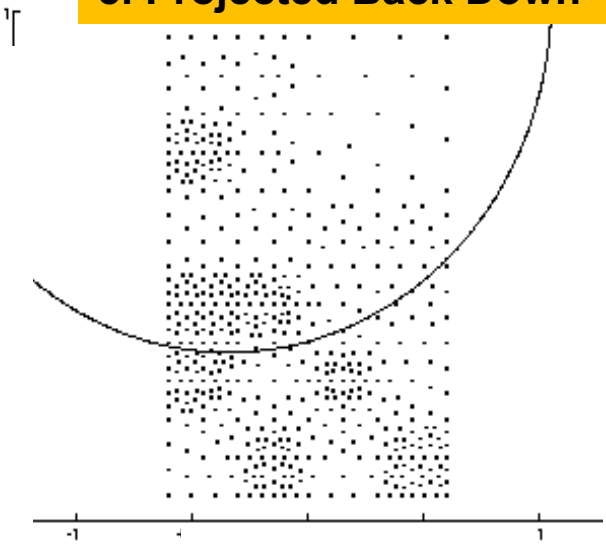
3.Stereographic Projection



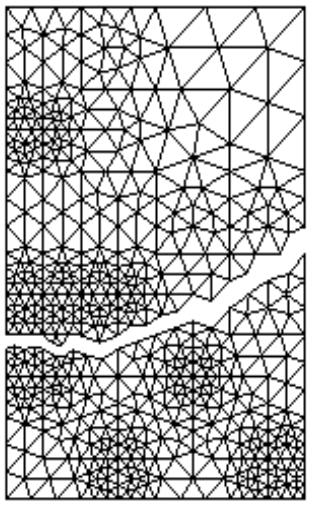
4. Conformal Mapping



5. Projected Back Down



6. Partitioned Mesh



In the year 2002

(and soon after) …

History

In the year 2002

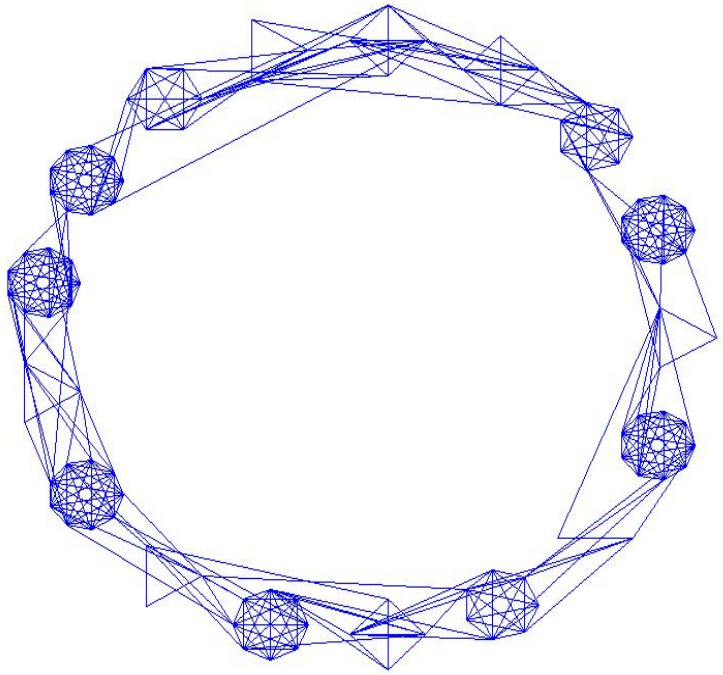
(and soon after) …

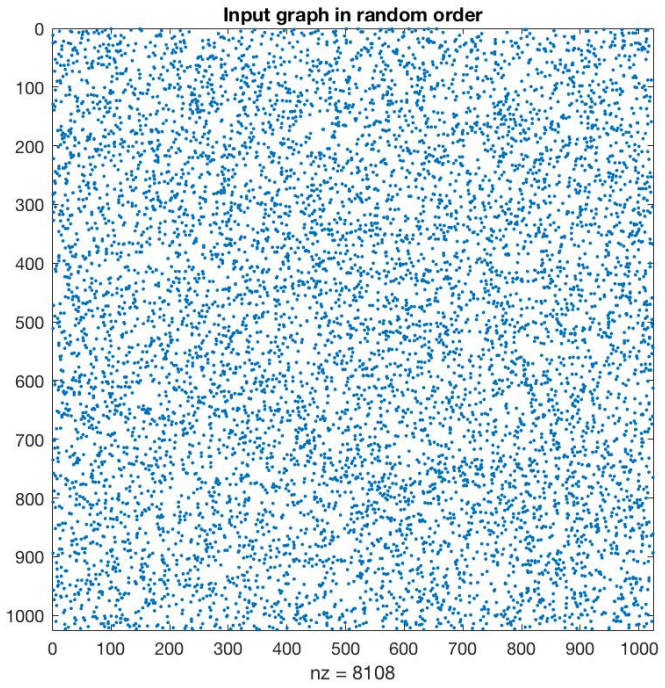
(In 2002, JRG shared an office with Jeremy Kepner at MIT.)

UCSB

First draft of HPCS graph analysis benchmark

[circa 2004]



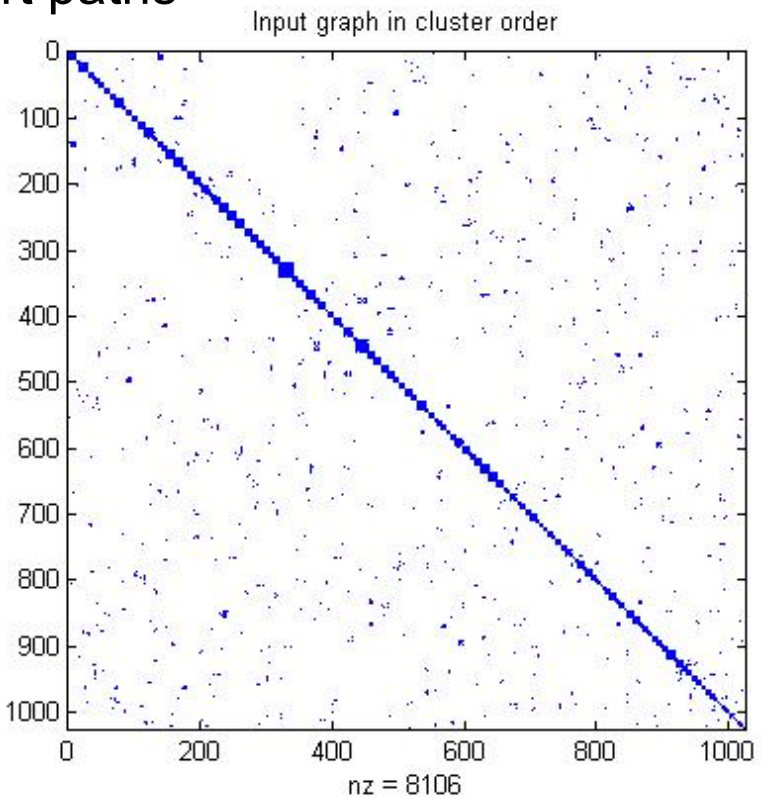


- Many tight clusters, loosely interconnected
- Input data is edge triples **<** i, j, a **>**
- Vertices and edges permuted randomly

Greedy clustering by breadth-first search

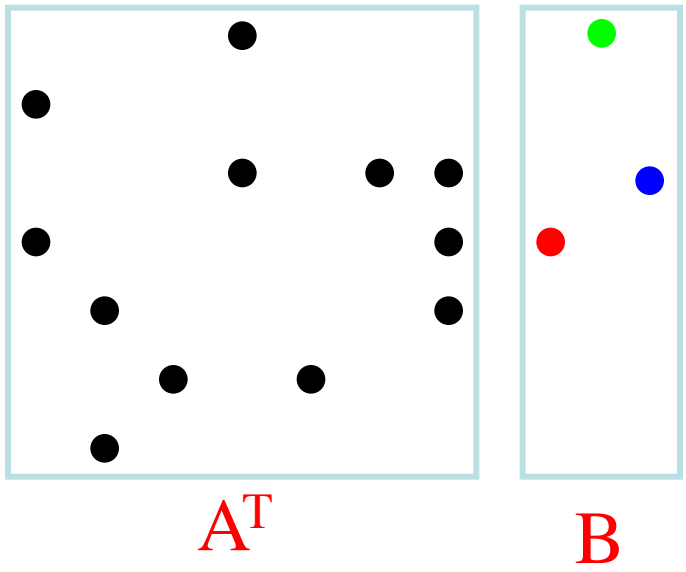
- Grow local clusters from many seeds in parallel
- Breadth-first search by sparse matrix * matrix
- Cluster vertices connected by many short paths

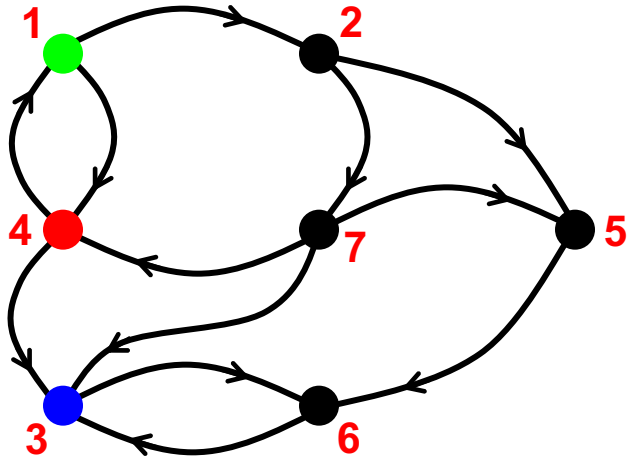
```
% Grow each seed to vertices
%    reached by at least k
%    paths of length 1 or 2

 C = sparse(seeds, 1:ns, 1, n, ns);
 C = A * C;
 C = C + A * C;
 C = C >= k;
```



UCSB

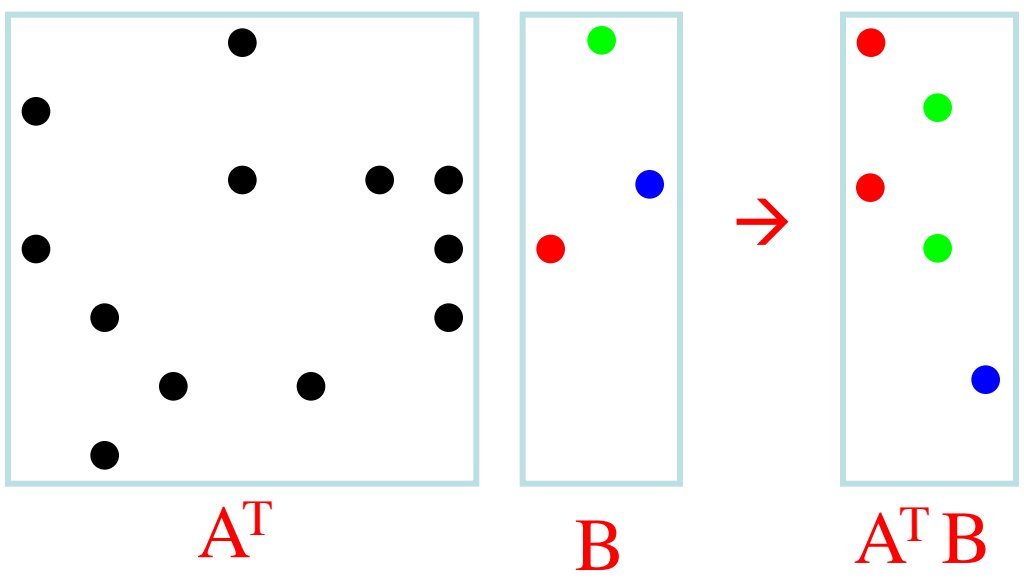
Multiple-source breadth-first search

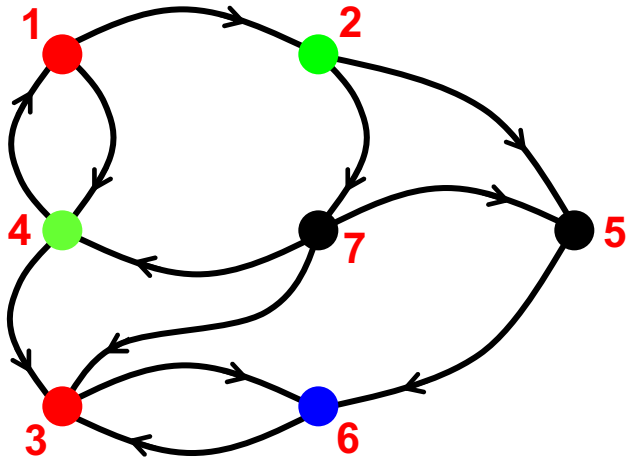




UCSB

Multiple-source breadth-first search

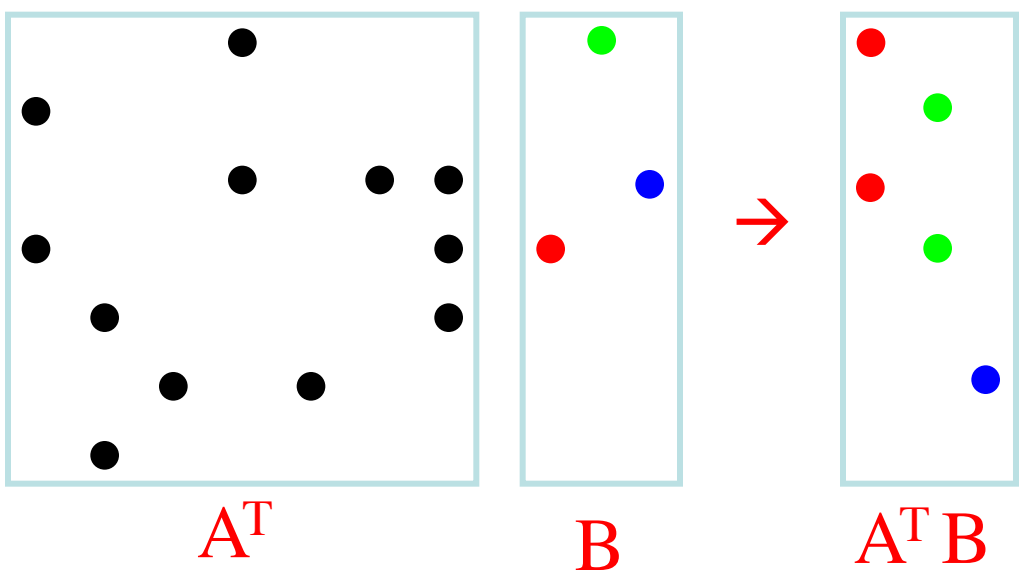


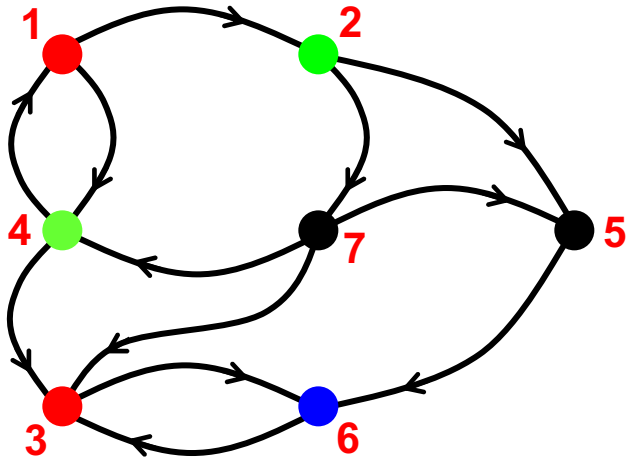


- Sparse array representation => space efficient
- Sparse matrix-matrix multiplication => work efficient
- Three possible levels of parallelism: searches, vertices, edges

UCSB

Multiple-source breadth-first search





The final HPCS graph analysis benchmark (SSCA2) was betweenness centrality, not clustering -- but the main primitive was still multiple-source breadth-first search!

UCSB

In the year 2010

(and soon after) …

Matrix-based graph processor design at MIT-LL

[Song, Kepner, et al. 2010]

3-D Graph Processor

William S. Song, Jeremy Kepner, Huy T. Nguyen, Joshua I. Kramer, Vitaliy Gleyzer, James R. Mann, Albert H. Horst, Larry L. Retherford, Robert A. Bond, Nadya T. Bliss, Eric I. Robinson, Sanjeev Mohindra, Julie Mullen
Lincoln Laboratory, Massachusetts Institute Technology, Lexington, MA 02420

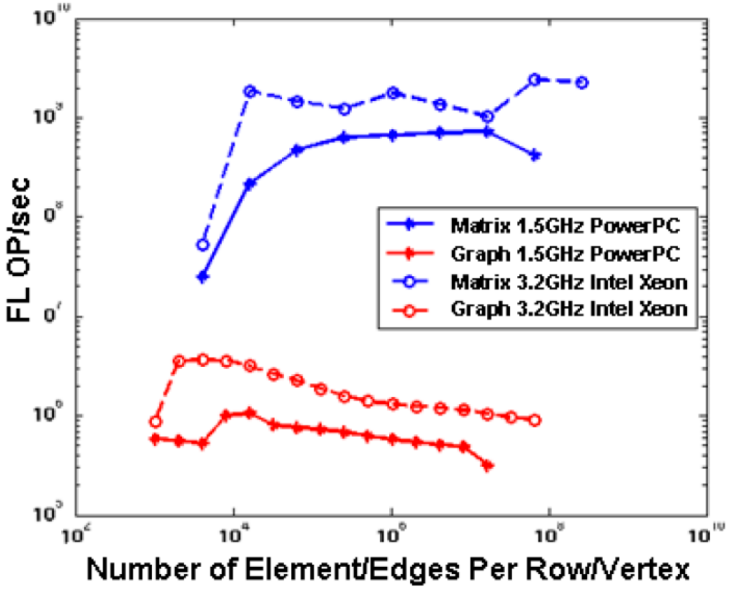


Figure 1: Computational Throughput Differences between Conventional and Graph Processing.

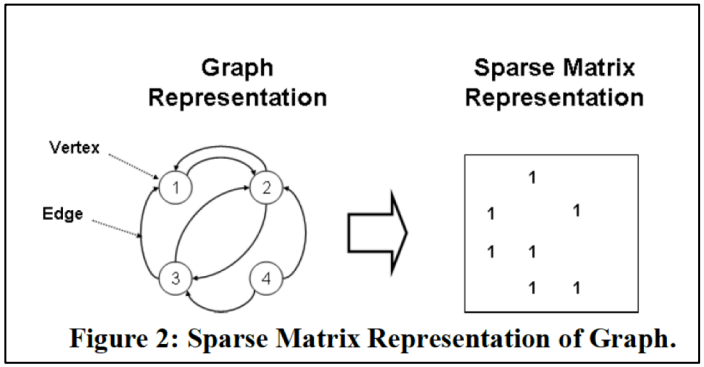


Figure 2: Sparse Matrix Representation of Graph.

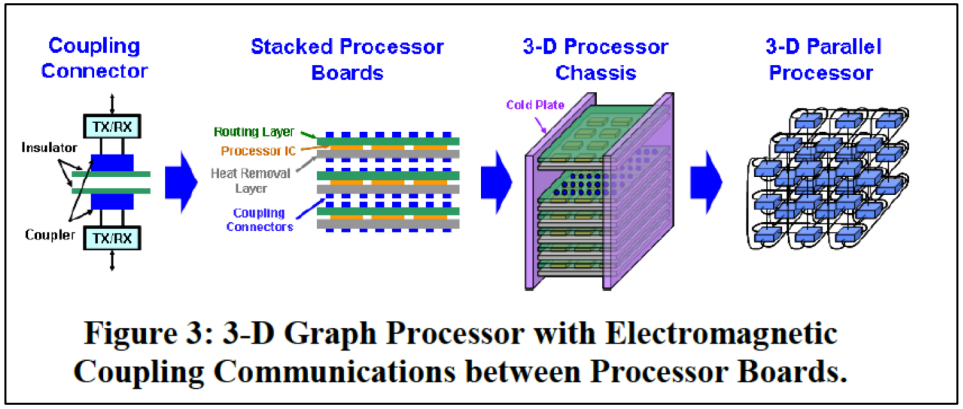
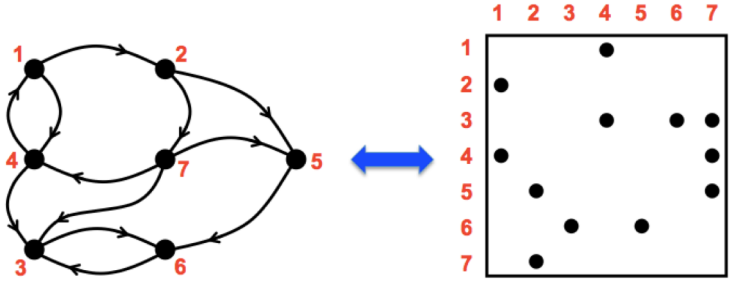


Figure 3: 3-D Graph Processor with Electromagnetic Coupling Communications between Processor Boards.

Combinatorial BLAS *[2010]*

gauss.cs.ucsb.edu/~aydin/CombBLAS

[Azad, Buluc, G, Lugowski, …]



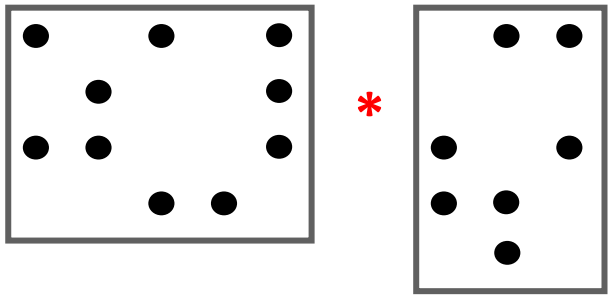
An extensible distributed-memory library offering a small but powerful set of linear algebraic operations specifically targeting graph analytics.

- Aimed at graph algorithm designers/programmers who are not expert in mapping algorithms to parallel hardware.
- Flexible templated C++ interface.
- Scalable performance from laptop to 100,000-processor HPC.

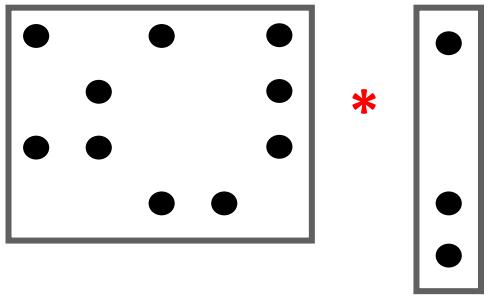
- Open source software.
- Version 1.6.2 released April 2018.

Sparse array primitives for graphs

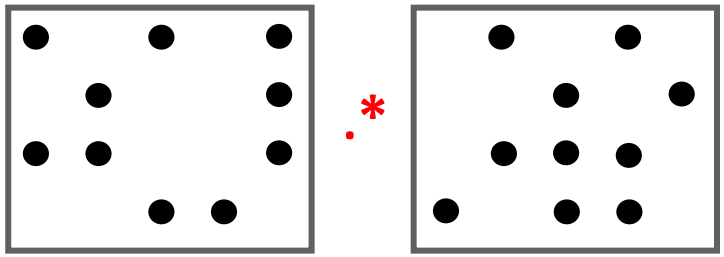
Sparse matrix-sparse matrix multiplication



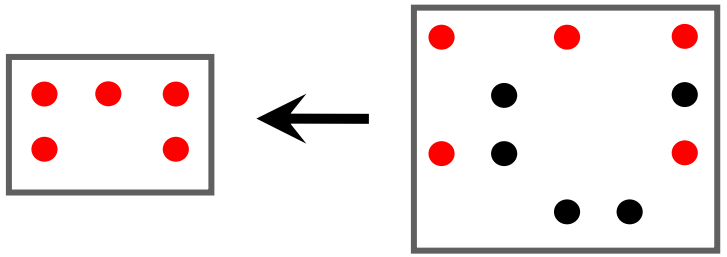
Sparse matrix-sparse vector multiplication



Element-wise operations



Sparse matrix indexing



Matrices over various semirings: (+, ·), (and, or), (min, +), …

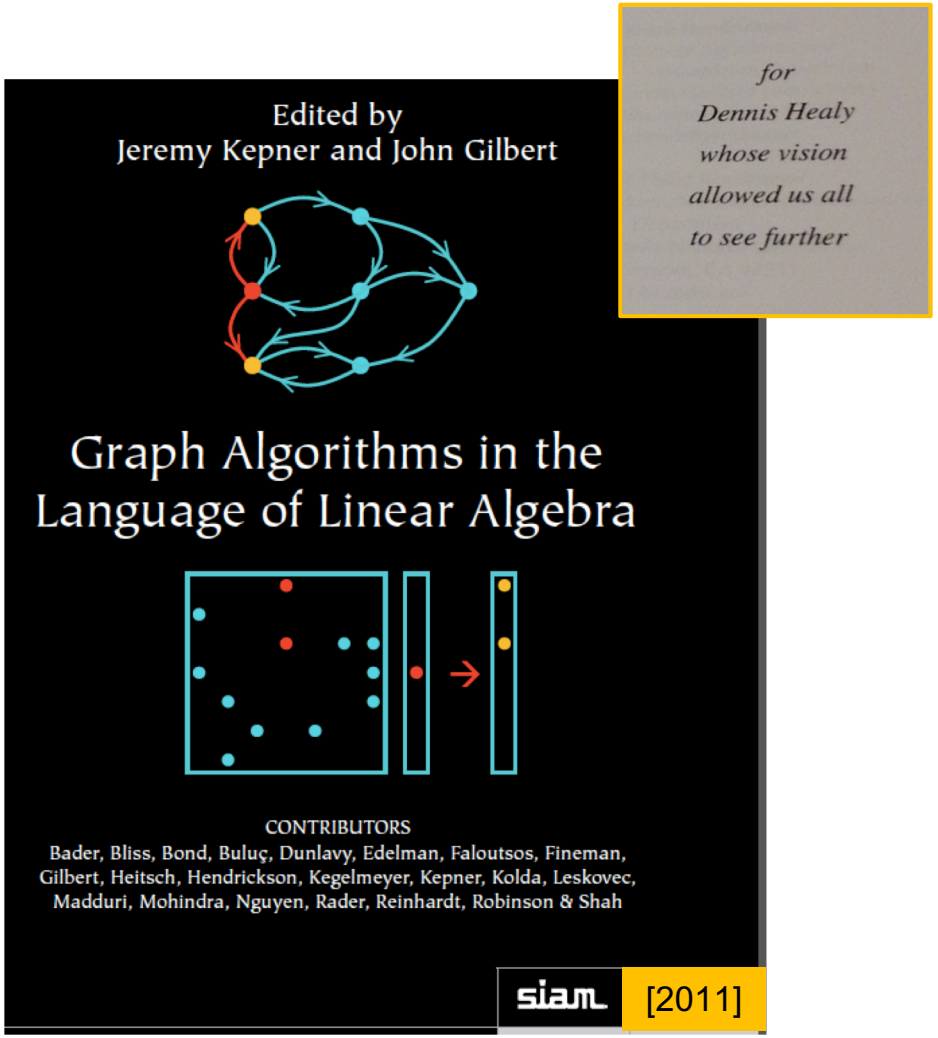
UCSB

Examples of semirings in graph algorithms

"values": edge/vertex attributes, "add": vertex data aggregation, "multiply": edge data processing	General schema for user-specified computation at vertices and edges
Real field: (R, +, *)	Numerical linear algebra
Boolean algebra: ({0 1}, |, &)	Graph traversal
Tropical semiring: (R U {∞}, min, +)	Shortest paths
(S, select, select)	Select subgraph, or contract nodes to form quotient graph

Graph algorithms in the language of linear algebra

- Kepner et al. study [2006]: fundamental graph algorithms including min spanning tree, shortest paths, independent set, max flow, clustering, …
- SSCA#2 / centrality [2008]
- Basic breadth-first search / Graph500 [2010]
- Combinatorial BLAS [2010]

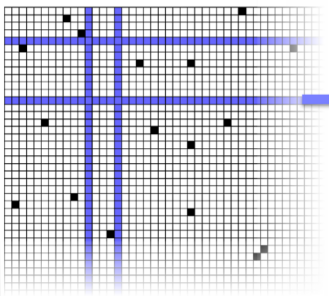


UCSB

History: D4M and Graphulo *[Kepner et al., MIT & UW 2011 - 2015]*

D4M: "Databases For Matlab"

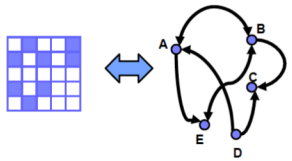
Triple Store
Distributed Database



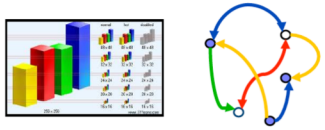
D4M
Dynamic
Distributed
Dimensional
Data
Model

Query:
Alice
Bob
Cathy
David
Earl

Associative Arrays
Numerical Computing Environment



A D4M query returns a sparse matrix or graph from Accumulo …



…for statistical signal processing or graph analysis in MATLAB

Triple store are high performance distributed databases for heterogeneous data

- D4M binds Associative Arrays to Triple Store, enabling rapid prototyping of data-intensive cloud analytics and visualization

D4M- 5

LINCOLN LABORATORY
MASSACHUSETTS INSTITUTE OF TECHNOLOGY

Computer Networks

Network Events Table: T
Associative Array: A

Row	Key (time)
1	2001-10-01 01 01 00
2	2001-10-01 01 02 00
3	2001-10-01 01 03 00
4	2001-10-01 01 04 00
5	2001-10-01 01 05 00
6	2001-10-01 01 06 00

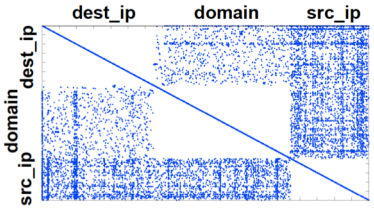
src_ip/a src_ip/b src_ip/c src_ip/d | domain/a domain/b domain/c domain/d | dest_ip/a dest_ip/b dest_ip/c dest_ip/d | Recv/a Recv/b Recv/c Recv/d Recv/e

- Define ranges of rows and columns

```
r = '2001-01-01 01 02 00,:,2001-01-01 01 04 00,'
c = StartsWith('src_ip/,domain/,dest_ip/')
```

- Query table and find popular pairs

```
A = T(r,c)
A' * A > 200
```



GRAPHULO

http://graphulo.mit.edu

- ***How*** **to do matrix math in Accumulo?**
  - **The TwoTable iterator pipeline**
- **Jaccard & k-Truss**
- > ***When*** **to do matrix math in Accumulo?**
  - **Memory requirements**
  - **Compare in-database I/O vs. alternatives**
- > **Future Work: *Multi-Node*, expand to *Relational Algebra*, use an *Optimizer* to choose the best plan**

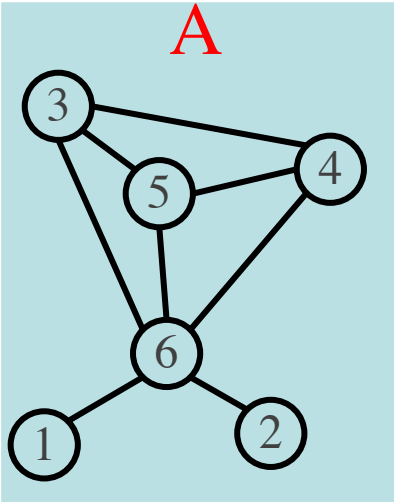
Linear algebra on associative arrays for heterogeneous distributed databases & graphs

Contact: Dylan Hutchison
dhutchis@cs.washington.edu

History: Jon Berry challenge problems *[2013]*

- Clustering coefficient (triangle counting)
- Connected components (bully algorithm)
- Maximum independent set (NP-hard)
- Maximal independent set (Luby algorithm)
- Single-source shortest paths
- Special betweenness (for subgraph isomorphism)

Counting triangles (clustering coefficient)

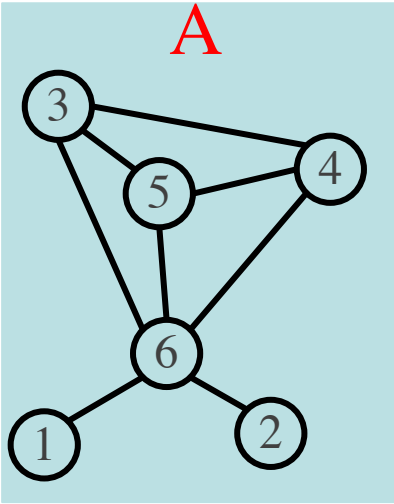


Clustering coefficient:

- Pr (wedge i-j-k makes a triangle with edge i-k)
- 3 * # triangles / # wedges
- 3 * **4** / **19** = **0.63** in example
- may want to compute for each vertex j

UCSB

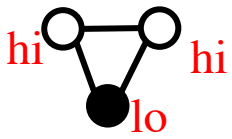
Counting triangles (clustering coefficient)



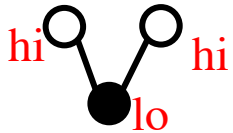
Clustering coefficient:

- Pr (wedge i-j-k makes a triangle with edge i-k)
- 3 * # triangles / # wedges
- 3 * **4** / **19** = **0.63** in example
- may want to compute for each vertex j

"Cohen's" algorithm to count triangles:



- Count triangles by lowest-degree vertex.

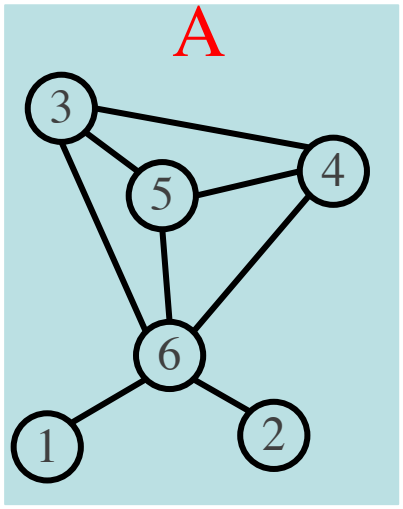


- Enumerate "low-hinged" wedges.

- Keep wedges that close.

UCSB

Counting triangles (clustering coefficient)

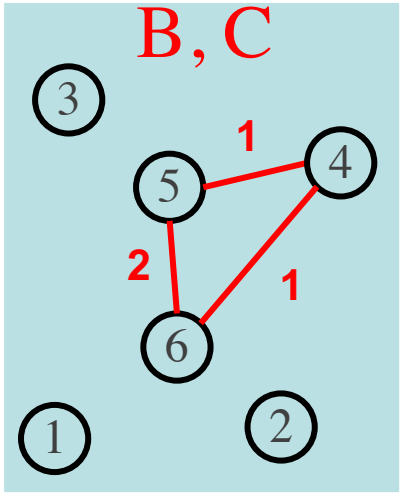


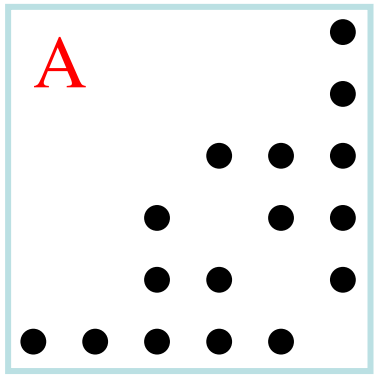
$A = L + U$ (hi->lo + lo->hi)

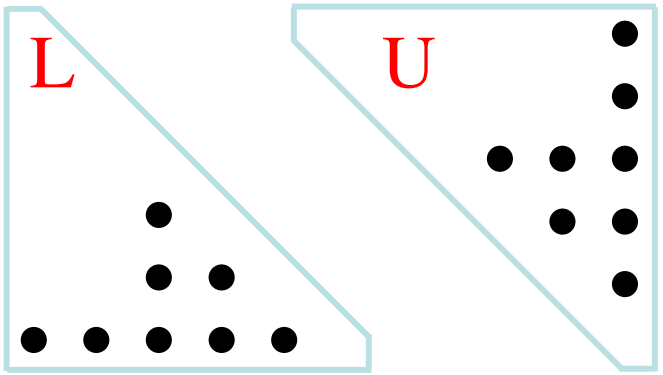
$L \times U = B$ (wedge, low hinge)

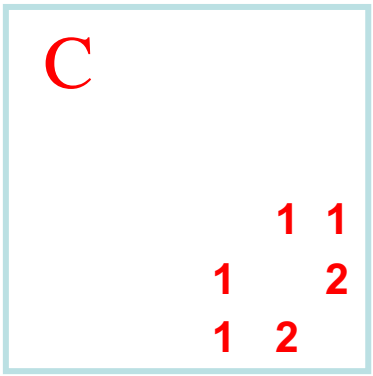
$A \wedge B = C$ (closed wedge)

$sum(C)/2 =$ **4 triangles**



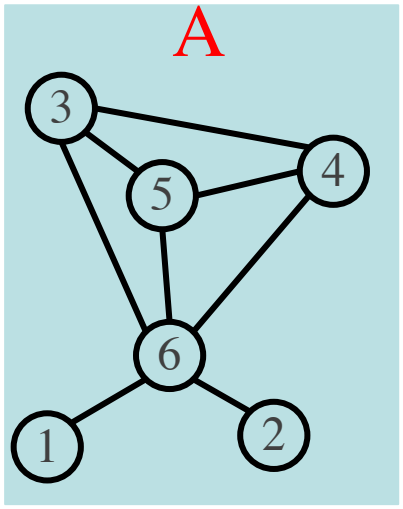






UCSB

Counting triangles (clustering coefficient)

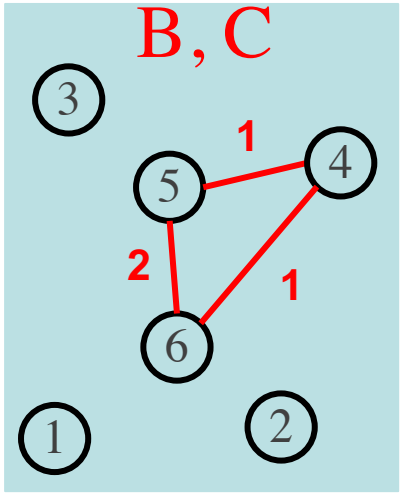


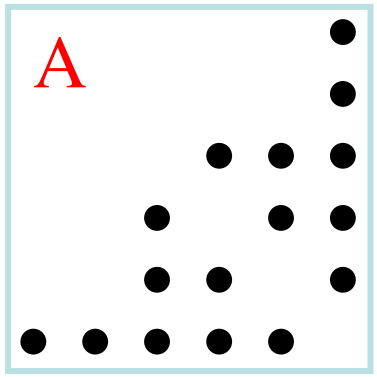
$A = L + U$ (hi->lo + lo->hi)

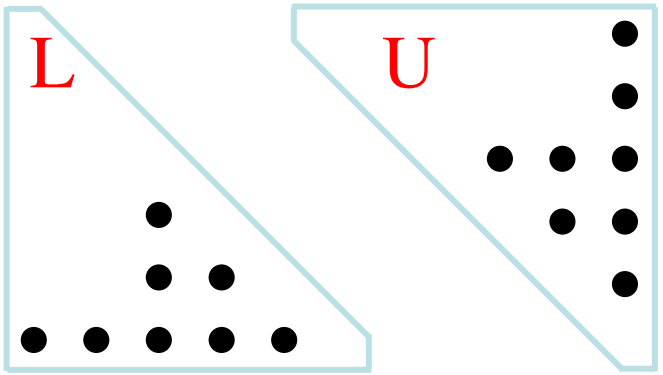
$L \times U = B$ (wedge, low hinge)

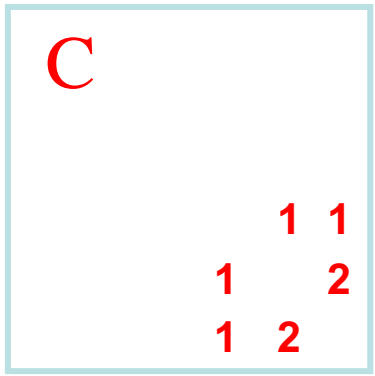
$A \wedge B = C$ (closed wedge)

$sum(C)/2 =$ **4 triangles**









Spoiler: $(L \times L) \wedge L$ works better in practice *[Wolf et al. 2017]*

UCSB

History: The Graph BLAS Forum

http://graphblas.org

- Manifesto, HPEC 2013:

Standards for Graph Algorithm Primitives

Tim Mattson (Intel Corporation), David Bader (Georgia Institute of Technology), Jon Berry (Sandia National Laboratory), Aydin Buluc (Lawrence Berkeley National Laboratory), Jack Dongarra (University of Tennessee), Christos Faloutsos (Carnegie Melon University), John Feo (Pacific Northwest National Laboratory), John Gilbert (University of California at Santa Barbara), Joseph Gonzalez (University of California at Berkeley), Bruce Hendrickson (Sandia National Laboratory), Jeremy Kepner (Massachusetts Institute of Technology), Charles Leiserson (Massachusetts Institute of Technology), Andrew Lumsdaine (Indiana University), David Padua (University of Illinois at Urbana-Champaign), Stephen Poole (Oak Ridge National Laboratory), Steve Reinhardt (Cray Corporation), Mike Stonebraker (Massachusetts Institute of Technology), Steve Wallach (Convey Corporation), Andrew Yoo (Lawrence Livermore National Laboratory)

Abstract-- It is our view that the state of the art in constructing a large collection of graph algorithms in terms of linear algebraic operations is mature enough to support the emergence of a standard set of primitive building blocks. This paper is a position paper defining the problem and announcing our intention to launch an open effort to define this standard.

- Foundations, HPEC 2016:

Mathematical Foundations of the GraphBLAS

Jeremy Kepner (MIT Lincoln Laboratory Supercomputing Center), Peter Aaltonen (Indiana University), David Bader (Georgia Institute of Technology), Aydın Buluç (Lawrence Berkeley National Laboratory), Franz Franchetti (Carnegie Mellon University), John Gilbert (University of California, Santa Barbara), Dylan Hutchison (University of Washington), Manoj Kumar (IBM), Andrew Lumsdaine (Indiana University), Henning Meyerhenke (Karlsruhe Institute of Technology), Scott McMillan (CMU Software Engineering Institute), Jose Moreira (IBM), John D. Owens (University of California, Davis), Carl Yang (University of California, Davis), Marcin Zalewski (Indiana University), Timothy Mattson (Intel)

The Present

In the year 2018 ....

UCSB

The Present

GABB 2018 Talks

(8:30am-10am)	Spectral Graph Drawing: Building Blocks and Performance Analysis	Shad Kirmani and Kamesh Madduri
	Morning Break (10-10:30am)	
1 (10:30am-12) Generating Graphs with known properties	Parallel generation of large-scale random graphs	Anil Vullikanti
	Design, Generation, and Validation of Extreme Scale Power-Law Graphs	Jeremy Kepner and Sid Samsi
	On Large-Scale Graph Generation with Validation of Diverse Triangle Statistics at Edges and Vertices	Geoffrey Sanders, Roger Pearce, Timothy La Fond and Jeremy Kepner
	Lunch (12-1:30pm)	
2 (1:30pm-3pm) GraphBLAS implementations	Patterns of GraphBLAS Algorithms: Tales from the Trenches	Scott McMillian
	Implementing the GraphBLAS C API	Jose Moreira, Manoj Kumar a William Horn
	PyGB: GraphBLAS DSL in Python with Dynamic Compilation into Efficient C++	Jesse Chamberlin, Marcin Zalews Scott McMillan and Andr Lumsdaine
	Afternoon Break (3-3:30pm)	
3 (3:30pm-5pm) Graph Building Blocks	A Survey of Modern Analysis on Graphs: Open Problems	Chris Long
	Panel: *Graph Building Blocks in Graph*	**Panelists:** Jose Moreira, Chris Lo and Marcin Zalewski.

```
GrB_Info bfs5m              // BFS of a graph (using vector assign & reduce)
(
    GrB_Vector *v_output,   // v [i] is the BFS level of node i in the graph
    const GrB_Matrix A,     // input graph, treated as if boolean in semiring
    GrB_Index s             // starting node of the BFS
)
{
    GrB_Info info ;
    GrB_Index n ;                       // # of nodes in the graph
    GrB_Vector q = NULL ;               // nodes visited at each level
    GrB_Vector v = NULL ;               // result vector
    GrB_Monoid Lor = NULL ;             // Logical-or monoid
    GrB_Semiring Boolean = NULL ;       // Boolean semiring
    GrB_Descriptor desc = NULL ;        // Descriptor for mxv

    GrB_Matrix_nrows (&n, A) ;          // n = # of rows of A
    GrB_Vector_new (&v, GrB_INT32, n) ; // Vector<int32_t> v(n) = 0
    GrB_Vector_new (&q, GrB_BOOL, n) ;  // Vector<bool> q(n) = false
    for (int32_t i = 0 ; i < n ; i++) GrB_Vector_setElement (v, 0, i) ;
    GrB_Vector_setElement (q, true, s) ;  // q[s] = true, false elsewhere

    GrB_Monoid_new (&Lor, GrB_LOR, (bool) false) ;
    GrB_Semiring_new (&Boolean, Lor, GrB_LAND) ;
    GrB_Descriptor_new (&desc) ;
    GrB_Descriptor_set (desc, GrB_MASK, GrB_SCMP) ;     // invert the mask
    GrB_Descriptor_set (desc, GrB_OUTP, GrB_REPLACE) ;  // clear q first

    bool successor = true ; // true when some successor found
```

The GraphBLAS C API Specification †:

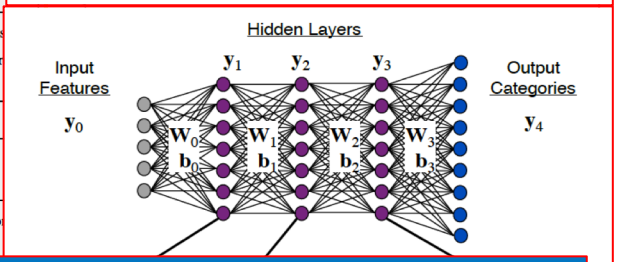
Version 1.2.0

Aydın Buluç, Timothy Mattson, Scott McMillan, José Moreira, Carl Yang

Operation Name	Mathematical Notation
mxm	$\mathbf{C}\langle\mathbf{M}, z\rangle = \mathbf{C} \odot \mathbf{A} \oplus . \otimes \mathbf{B}$
mxv	$\mathbf{w}\langle\mathbf{m}, z\rangle = \mathbf{w} \odot \mathbf{A} \oplus . \otimes \mathbf{u}$
vxm	$\mathbf{w}^T\langle\mathbf{m}^T, z\rangle = \mathbf{w}^T \odot \mathbf{u}^T \oplus . \otimes \mathbf{A}$
eWiseMult	$\mathbf{C}\langle\mathbf{M}, z\rangle = \mathbf{C} \odot \mathbf{A} \otimes \mathbf{B}$
	$\mathbf{w}\langle\mathbf{m}, z\rangle = \mathbf{w} \odot \mathbf{u} \otimes \mathbf{v}$
eWiseAdd	$\mathbf{C}\langle\mathbf{M}, z\rangle = \mathbf{C} \odot \mathbf{A} \oplus \mathbf{B}$
	$\mathbf{w}\langle\mathbf{m}, z\rangle = \mathbf{w} \odot \mathbf{u} \oplus \mathbf{v}$
reduce (row)	$\mathbf{w}\langle\mathbf{m}, z\rangle = \mathbf{w} \odot [\oplus_j \mathbf{A}(:, j)]$
reduce (scalar)	$s = s \odot [\oplus_{i,j} \mathbf{A}(i, j)]$
	$s = s \odot [\oplus_i \mathbf{u}(i)]$
apply	$\mathbf{C}\langle\mathbf{M}, z\rangle = \mathbf{C} \odot f_u(\mathbf{A})$
	$\mathbf{w}\langle\mathbf{m}, z\rangle = \mathbf{w} \odot f_u(\mathbf{u})$
transpose	$\mathbf{C}\langle\mathbf{M}, z\rangle = \mathbf{C} \odot \mathbf{A}^T$
extract	$\mathbf{C}\langle\mathbf{M}, z\rangle = \mathbf{C} \odot \mathbf{A}(i, j)$
	$\mathbf{w}\langle\mathbf{m}, z\rangle = \mathbf{w} \odot \mathbf{u}(i)$
assig	

Enabling Massive Deep Neural Networks with the GraphBLAS

Jeremy Kepner[1], Manoj Kumar[2], José Moreira[2], Pratap Pattnaik[2], Mauricio Serrano[2], Henry Tufo[2]



User Guide for SuiteSparse:GraphBLAS

Timothy A. Davis

davis@tamu.edu, Texas A&M University.

http://www.suitesparse.com and http://aldenmath.com

VERSION 2.0.1, Mar 15, 2018

Abstract

SuiteSparse:GraphBLAS is a full implementation of the GraphBLAS standard, which defines a set of sparse matrix operations on an extended algebra of semirings using an almost unlimited variety of operators and types. When applied to sparse adjacency matrices, these algebraic operations are equivalent to computations on graphs. GraphBLAS provides a powerful and expressive framework for creating graph algorithms based on the elegant mathematics of sparse matrix operations on a semiring.

GraphChallenge

Motivation | Challenges | Data Sets | Scenarios | Submit

Home

Graph Challenge Champions

2017 Analysis of all Triangle Counting Submissions

Champions

- *Fast Linear Algebra-Based Triangle Counting with Kok*
  Hammond, Sivasankaran Rajamanickam (Sandia)
- *Triangle Counting for Scale-Free Graphs at Scale in Di*

Fast Linear Algebra-Based Triangle Counting with KokkosKernels

Michael M. Wolf, Mehmet Deveci, Jonathan W. Berry, Simon D. Hammond, Sivasankaran Rajamanickam
Center for Computing Research, Sandia National Laboratories
Albuquerque, NM 87185
{mmwolf,mndevec,jberry,sdhammo,srajama}@sandia.gov

***Abstract*—Triangle counting serves as a key building block for a set of important graph algorithms in network science. In this paper, we address the IEEE HPEC Static Graph Challenge problem of triangle counting, focusing on obtaining the best parallel performance on a single multicore node. Our implementation uses a linear algebra-based approach to triangle counting that has grown out of work related to our miniTri data analytics miniapplication [1] and our efforts to pose graph algorithms in the language of linear algebra. We leverage KokkosKernels to implement this approach efficiently on multicore architectures. Our performance results are competitive with the fastest known graph traversal-based approaches and are significantly faster than the Graph Challenge reference implementations, up to 670,000 times faster than the C++ reference and 10,000 times faster than the Python reference on a single Intel Haswell node.**

to pose graph algorithms in the language of linear algebra. We focus on triangle counting on a single compute node, leveraging KokkosKernels [14] to implement this approach efficiently. We obtain results that are competitive with the fastest known graph traversal-based approaches.

B. Linear Algebra Primitives for Graph Algorithms

The Graph BLAS [15], [16] community has been working to standardize a set of building blocks to solve graph problems in the language of sparse linear algebra. Many graph computations can be efficiently written in terms of linear algebra [17], including breadth-first search, betweenness centrality, and triangle counting/enumeration [1], [18] (discussed further

The Future

In the years 2019 —

UCSB

What do we hope for in the future?

- More basic capabilities
  - Streaming and dynamic-graph algorithms
  - “Priority queue” algorithms: strong components, top k vertices, etc.
  - Not materializing intermediate results (eg, incidence matrix methods)
  - Laplacian paradigm for graph algorithms

Question: Not materializing big matrix products

In sparse Gaussian elimination, for nonsymmetric A, one can find . . .

- column nested dissection or min degree permutation
- column elimination tree $T(A^TA)$
- row and column counts for $G^+(A^TA)$
- supernodes of $G^+(A^TA)$
- nonzero structure of $G^+(A^TA)$

. . . efficiently, without ever forming A^TA explicitly.

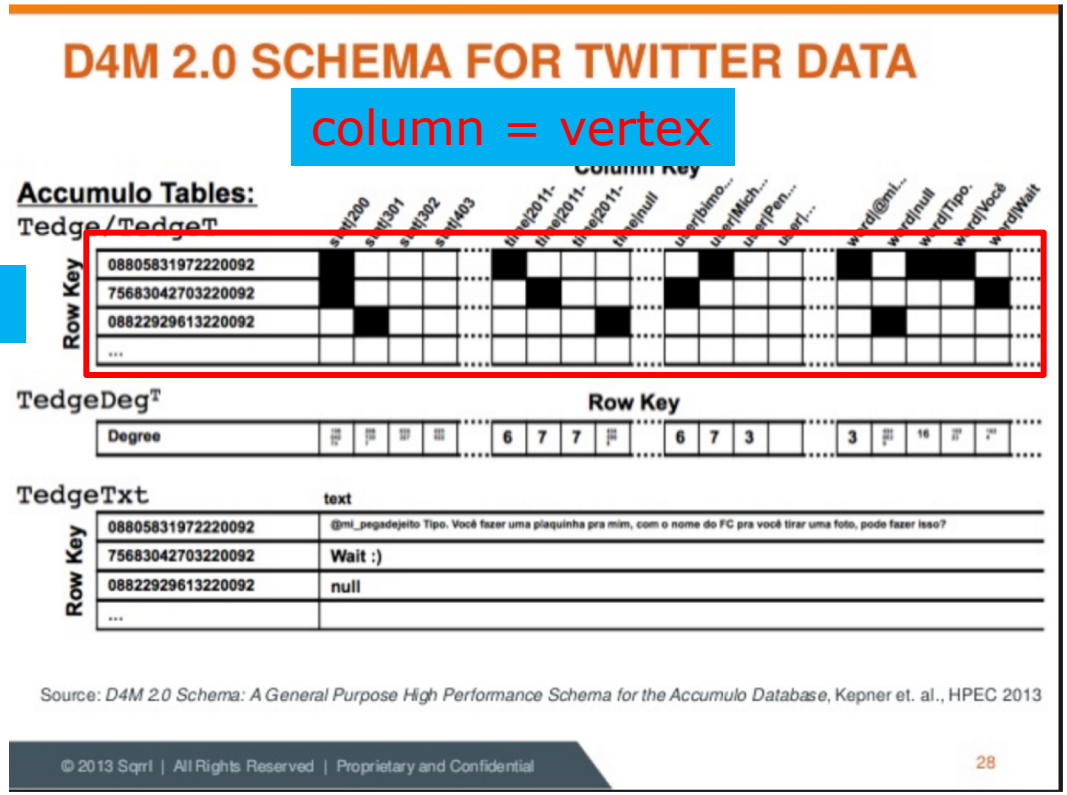
- How generally can we do graph algorithms in linear algebra without storing intermediate results?
- Can we do fine-grained scheduling of vertex and edge operations to break out of bulk synchronous execution?
- Can we reason directly about products of sparse matrices?

Storing A, operating implicitly on A^TA

- CombBLAS represents graphs as adjacency matrices.
- D4M represents graphs as incidence matrices; matrix A represents $G(A^TA)$:

A

row = hyperedge



UCSB

Storing A, operating implicitly on A^TA

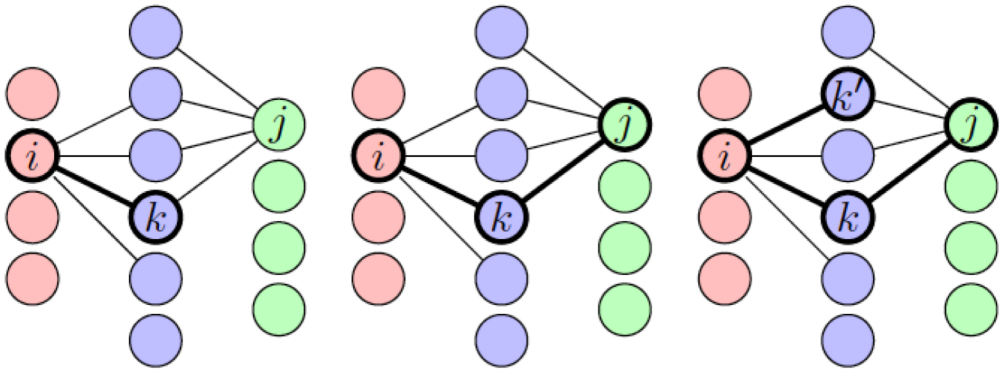
- Many other cases:
  - Optimization: KKT systems, interior point methods.
  - Automatic differentiation: distance-2 coloring.
  - Linear equations: QR factorization, structure prediction for LU factorization with partial pivoting.

- Question: What can you do fast on $G(A^TA)$ just from $G(A)$?

Statistics for A^TA itself are harder!

- nnz(A^TA) seems to be as hard as computing A^TA.
  - but randomized estimate is possible *[Cohen 1998]*

- Sampling algorithms are possible too, e.g. diamond sampling for k largest elements of A^TA (or B*C in general) *[Ballard/Kolda/Pinar/Seshadri 2015]*



(a) Sample $(k, i) \propto w_{ki}$ (b) Sample $j \in \mathcal{N}_k^B$ (c) Sample $k' \in \mathcal{N}_i^A$

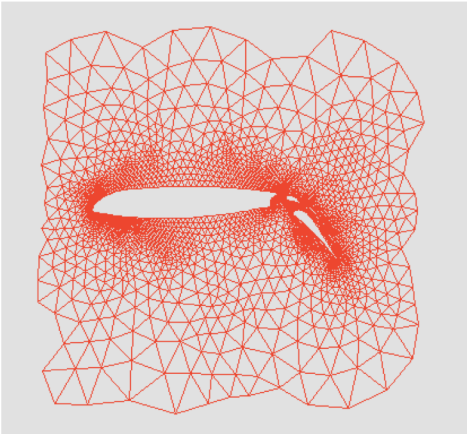
Ballard et al. ICDM 2015

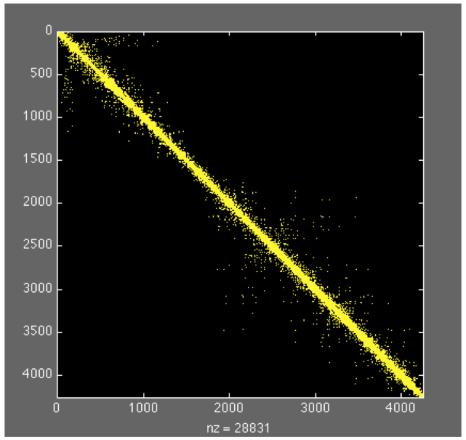
UCSB

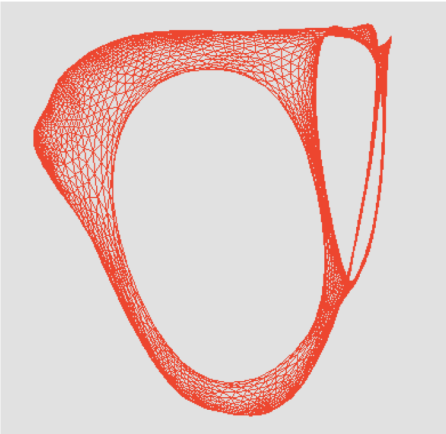
What do we hope for in the future?

- More basic capabilities
  - Streaming and dynamic algorithms
  - “Priority queue” algorithms: strong components, top k vertices, etc.
  - Not materializing intermediate results (eg, incidence matrix methods)
  - Laplacian paradigm for graph algorithms

Laplacian matrix of a graph







- Graph Laplacian: Symmetric, positive semidefinite, weighted.
- Laplacian paradigm: Use Ax = b as a subroutine in graph algorithms
  [Kelner, Teng, many others]
- Laplacian eigenvectors for partitioning, embedding, and clustering
  [Fiedler, Pothen/Simon, Spielman/Teng, many others]
- Interesting new ideas coming from theoretical computer science.

UCSB

What do we hope for in the future?

- More basic capabilities
  - Streaming and dynamic algorithms
  - “Priority queue” algorithms: strong components, top k vertices, etc.
  - Not materializing intermediate results (eg, incidence matrix methods)
  - Laplacian paradigm for graph algorithms

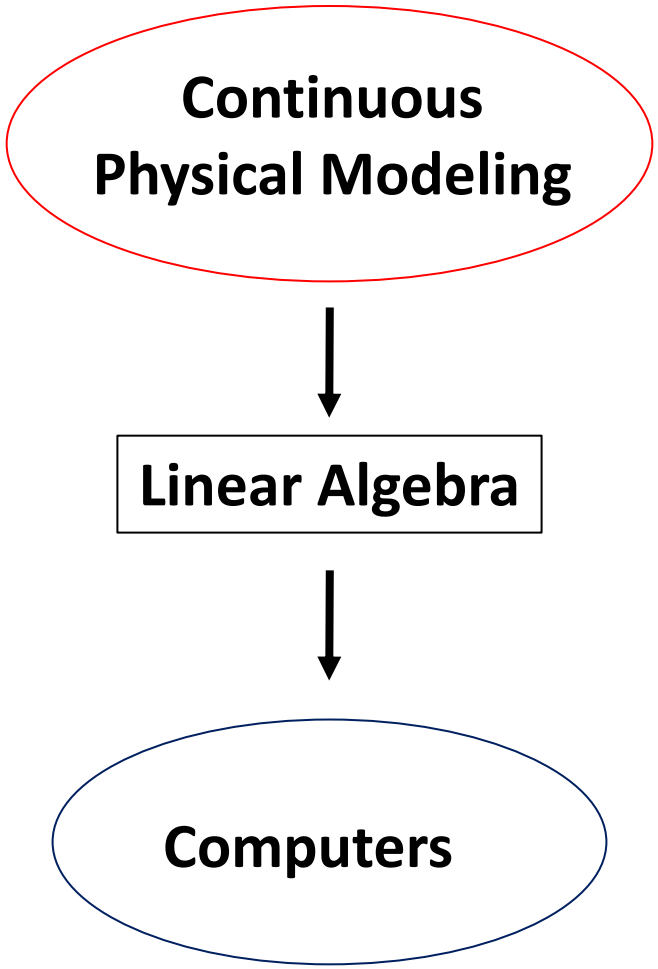
What do we hope for in the future?

- More basic capabilities
  - Streaming and dynamic algorithms
  - “Priority queue” algorithms: strong components, top k vertices, etc.
  - Not materializing intermediate results (eg, incidence matrix methods)
  - Laplacian paradigm for graph algorithms
- More directions
  - Integration with numerical matrix libraries
  - Statistical perspective: random objects, stochastic graphs, etc.
  - Deep neural networks (more)
  - Signal processing on graphs

What do we hope for in the future?

- More basic capabilities
  - Streaming and dynamic algorithms
  - “Priority queue” algorithms: strong components, top k vertices, etc.
  - Not materializing intermediate results (eg, incidence matrix methods)
  - Laplacian paradigm for graph algorithms
- More directions
  - Integration with numerical matrix libraries
  - Statistical perspective: random objects, stochastic graphs, etc.
  - Deep neural networks (more)
  - Signal processing on graphs
- More uptake
  - By hardware vendors
  - By software vendors

Summary: Past 60 Years

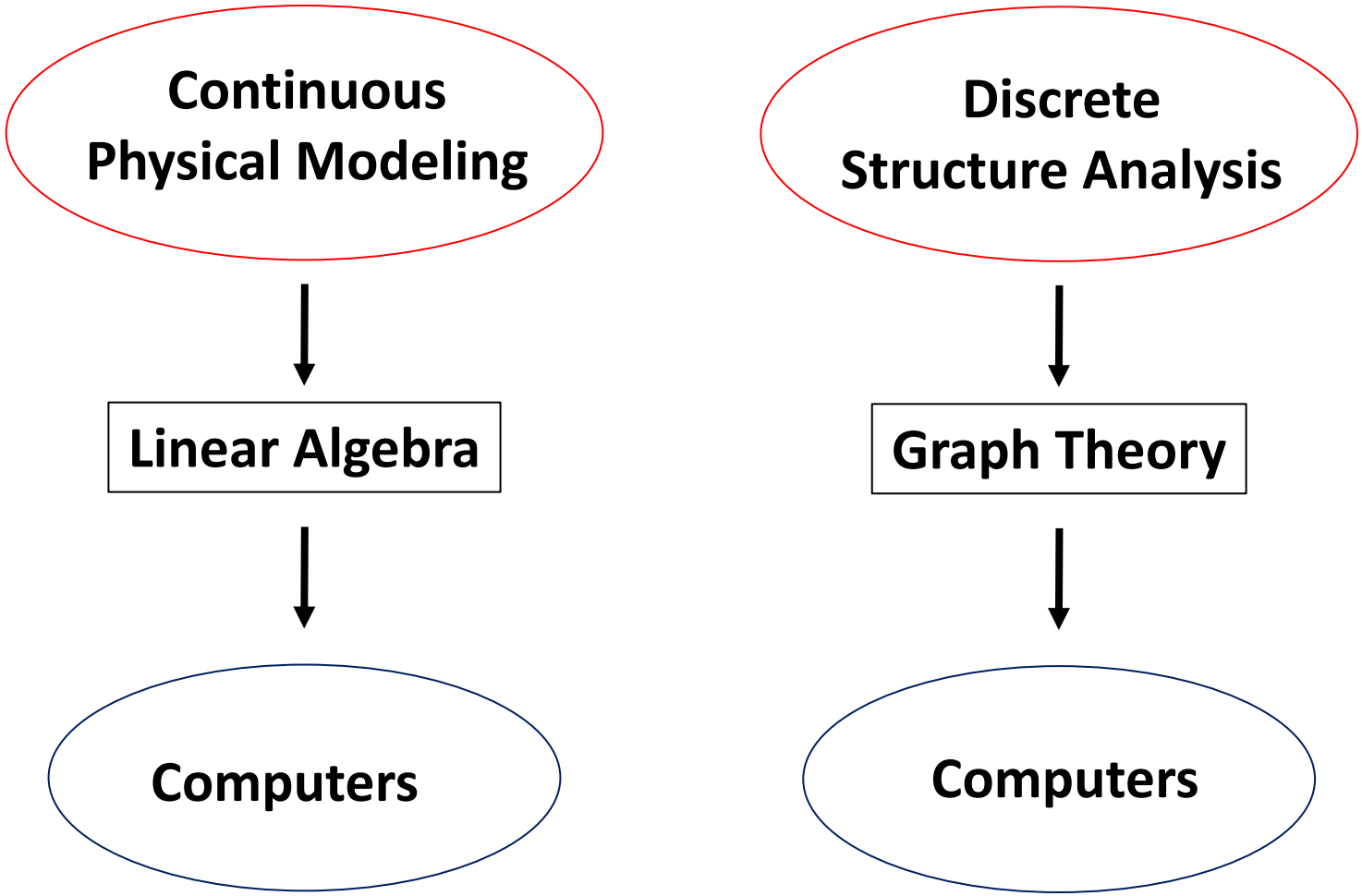


As the "middleware"
of scientific computing,
linear algebra has given us:

- Mathematical tools
- High-level primitives
- High-quality software libraries
- High-performance kernels for computer architectures
- Interactive environments

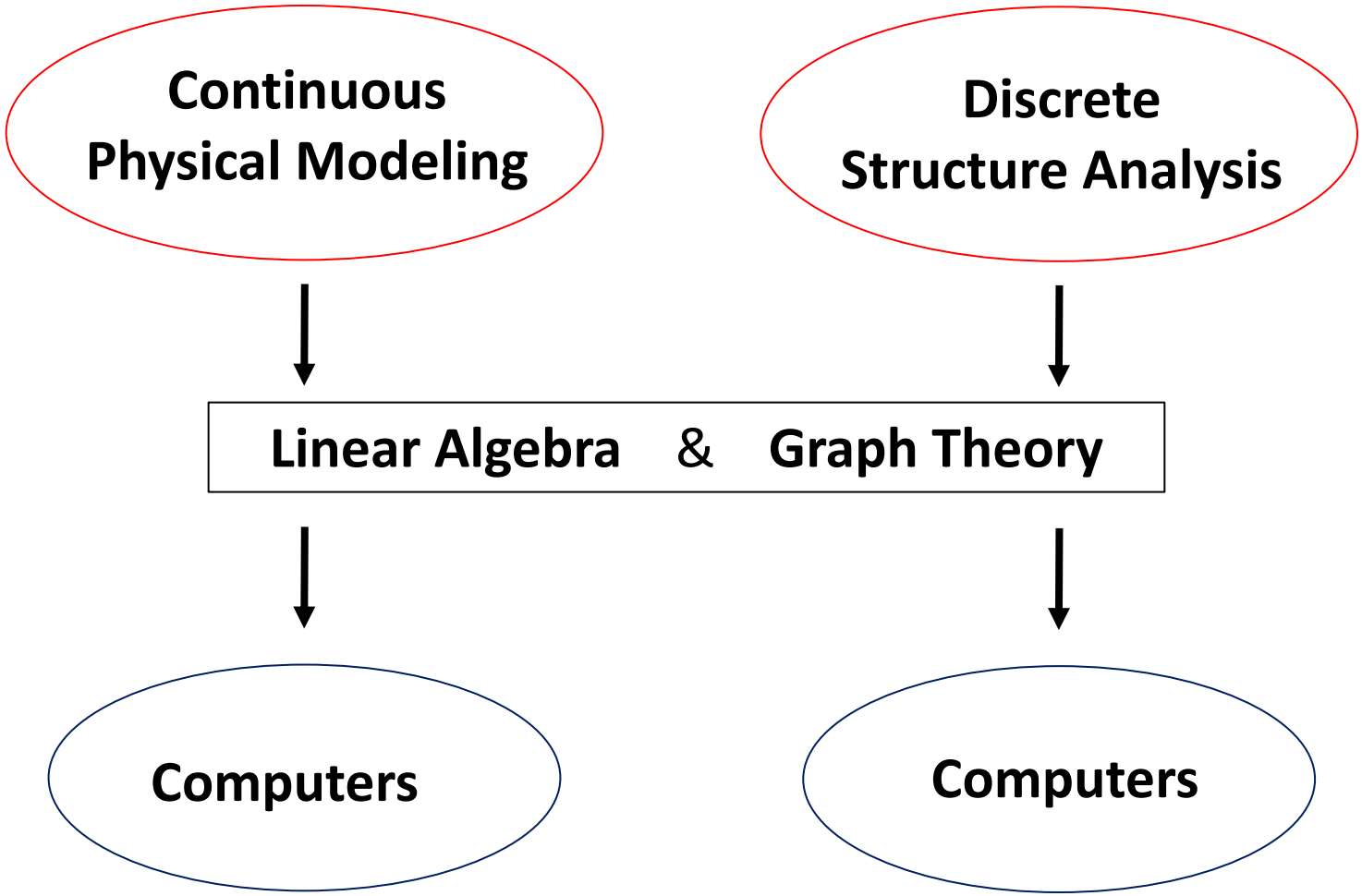
UCSB

Today


Continuous
Physical Modeling
Linear Algebra
Computers
Discrete
Structure Analysis
Graph Theory
Computers

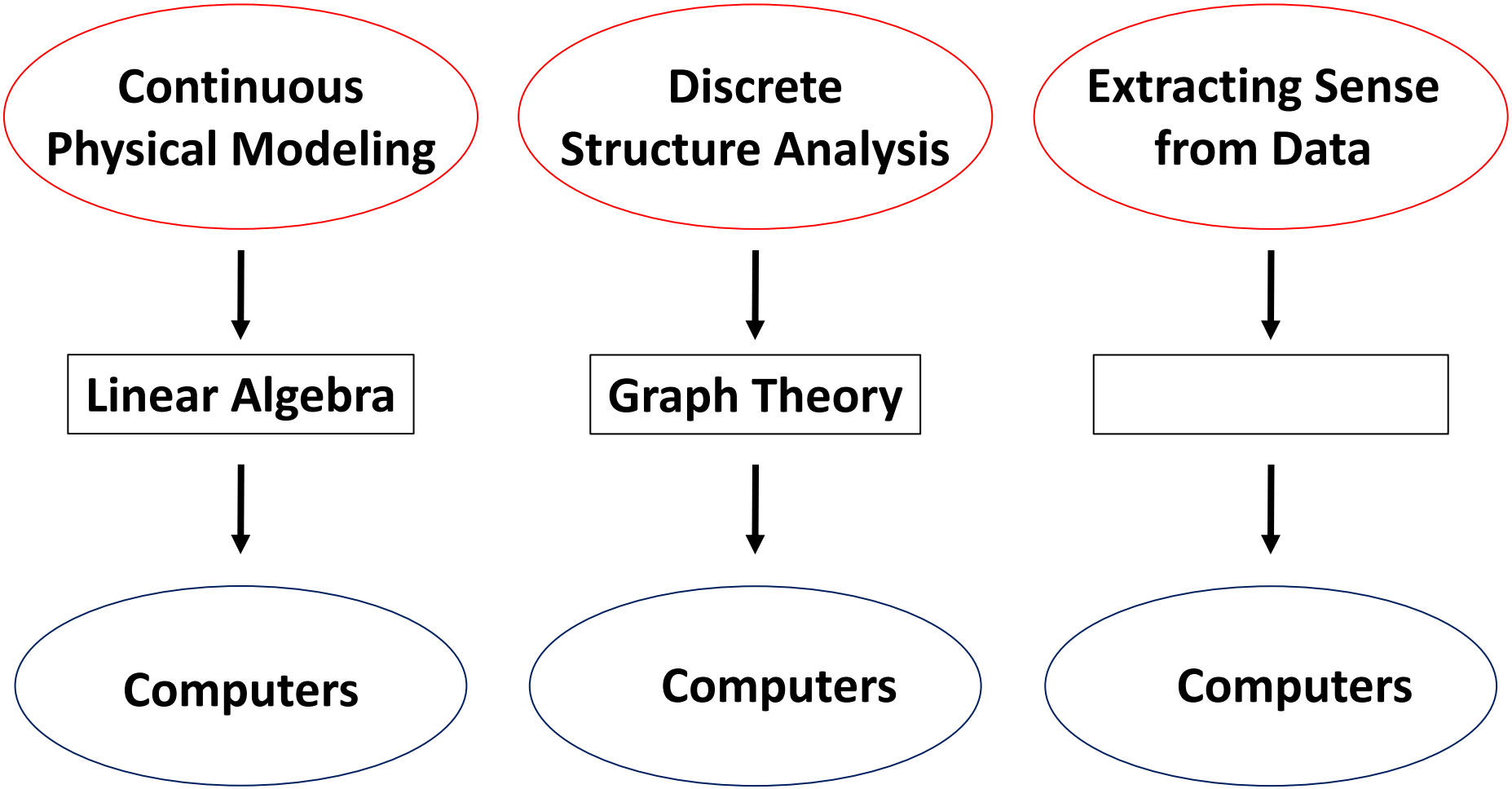
UCSB

Today


Continuous
Physical Modeling
Discrete
Structure Analysis
Linear Algebra & Graph Theory
Computers
Computers

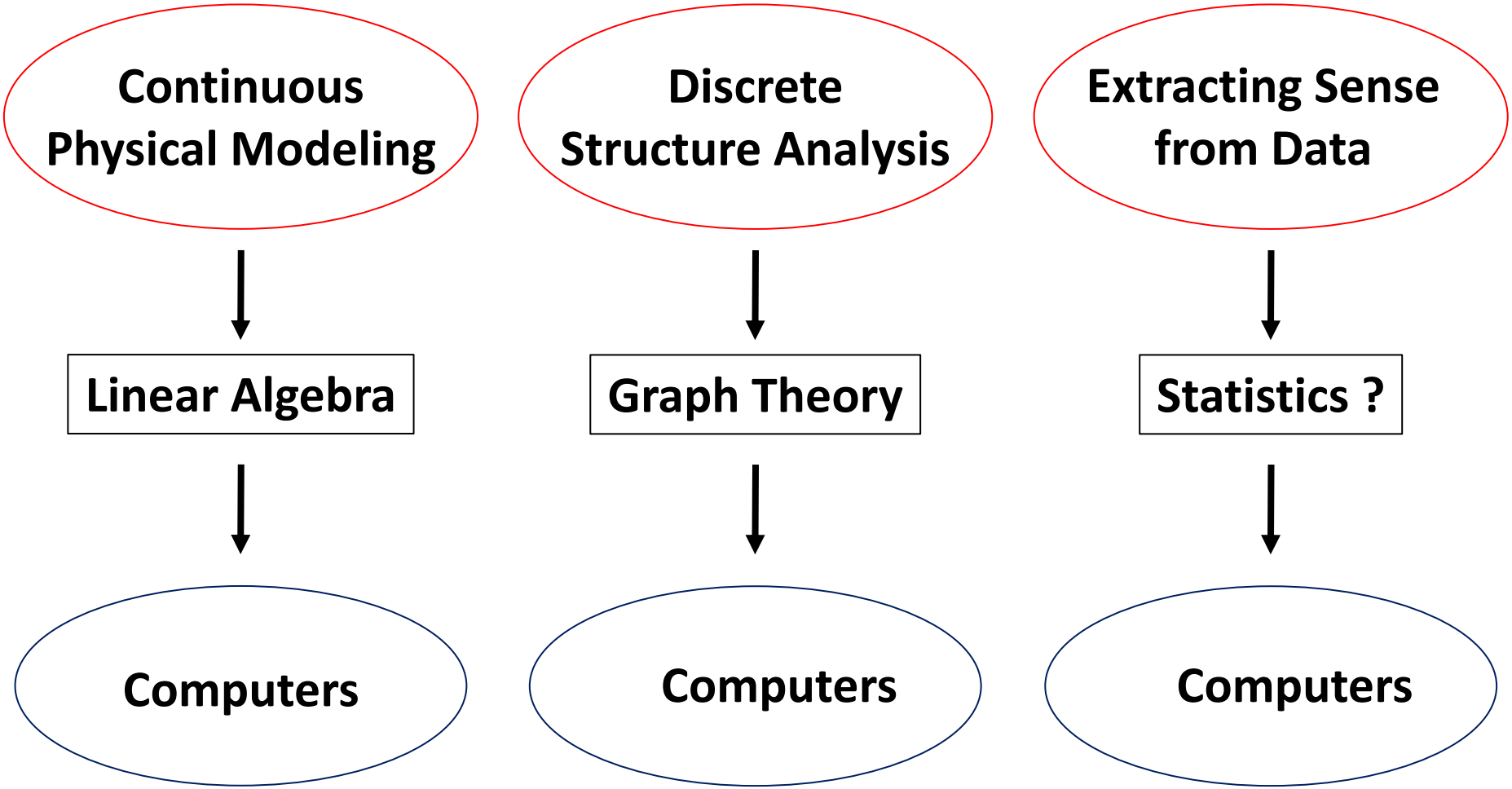
UCSB

Tomorrow



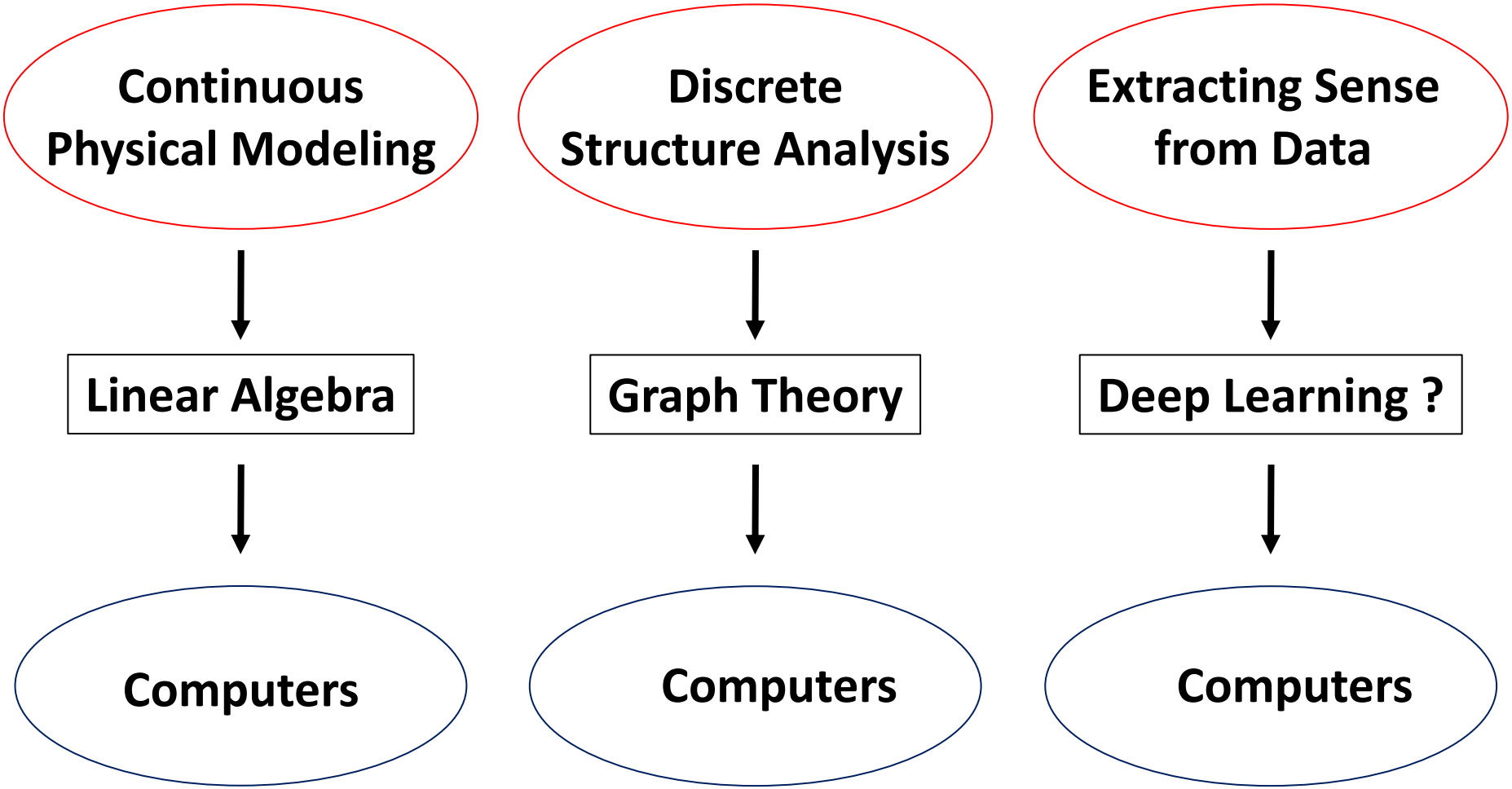
UCSB

Tomorrow



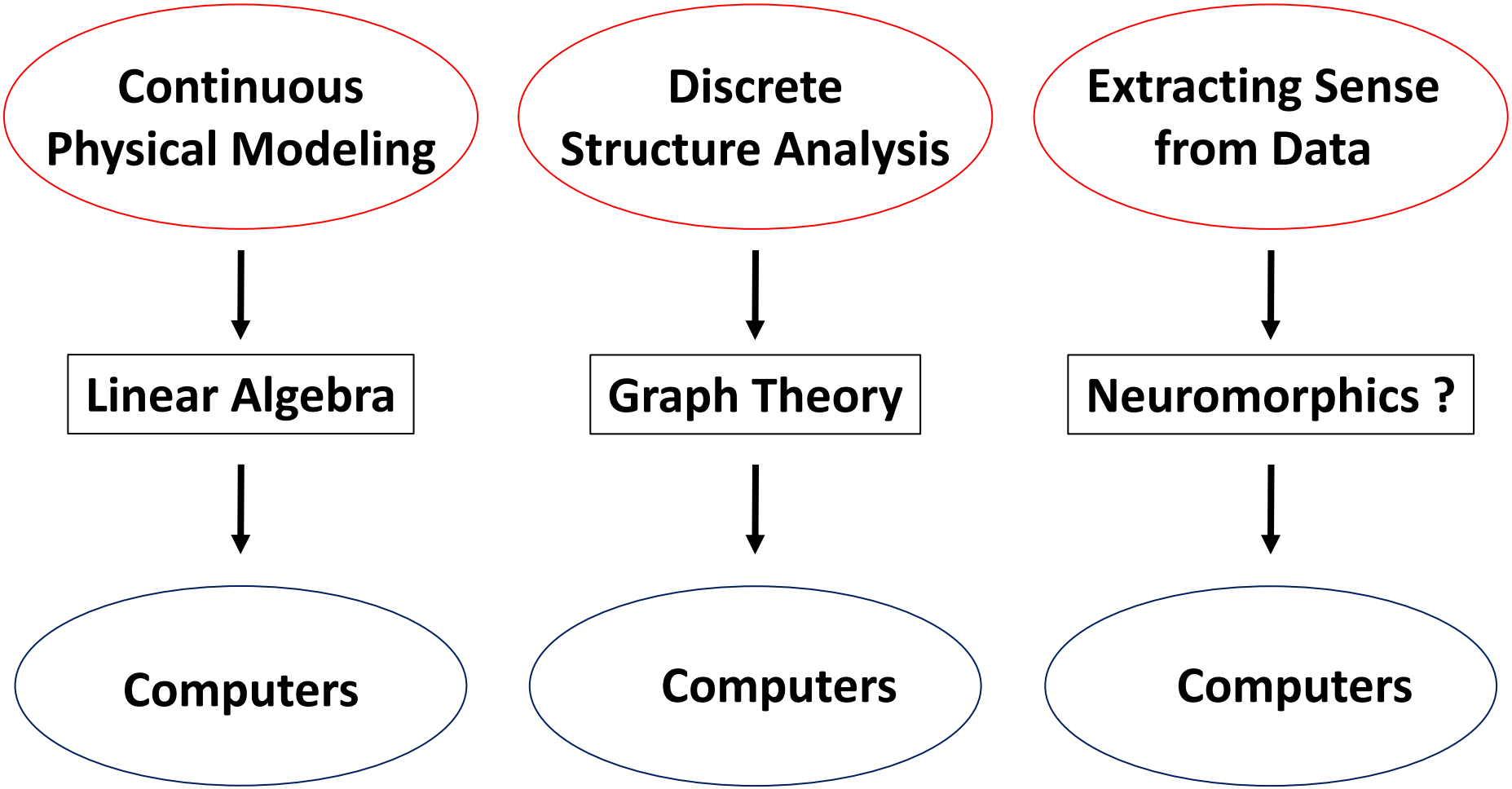
UCSB

Tomorrow



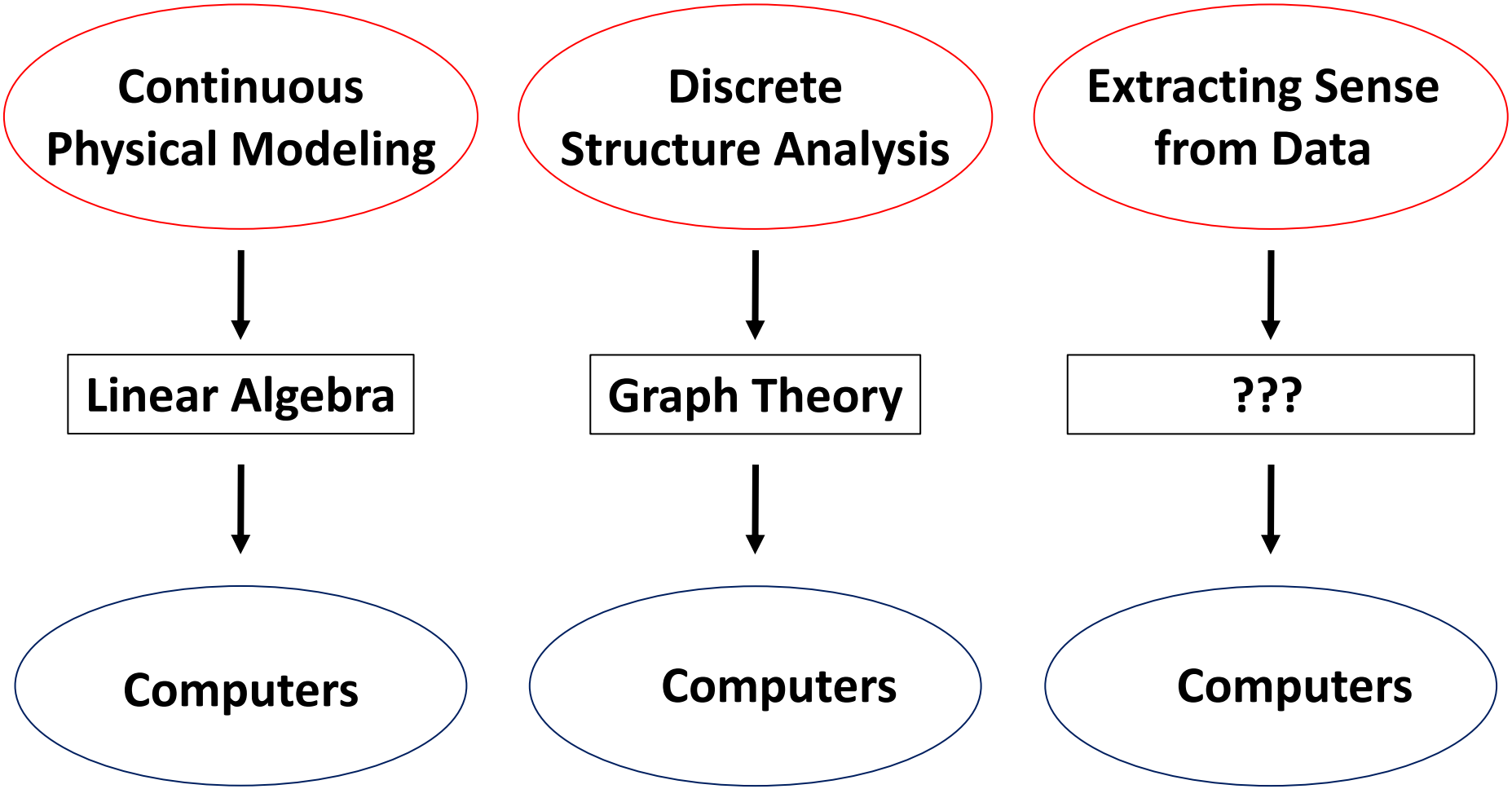
UCSB

Tomorrow



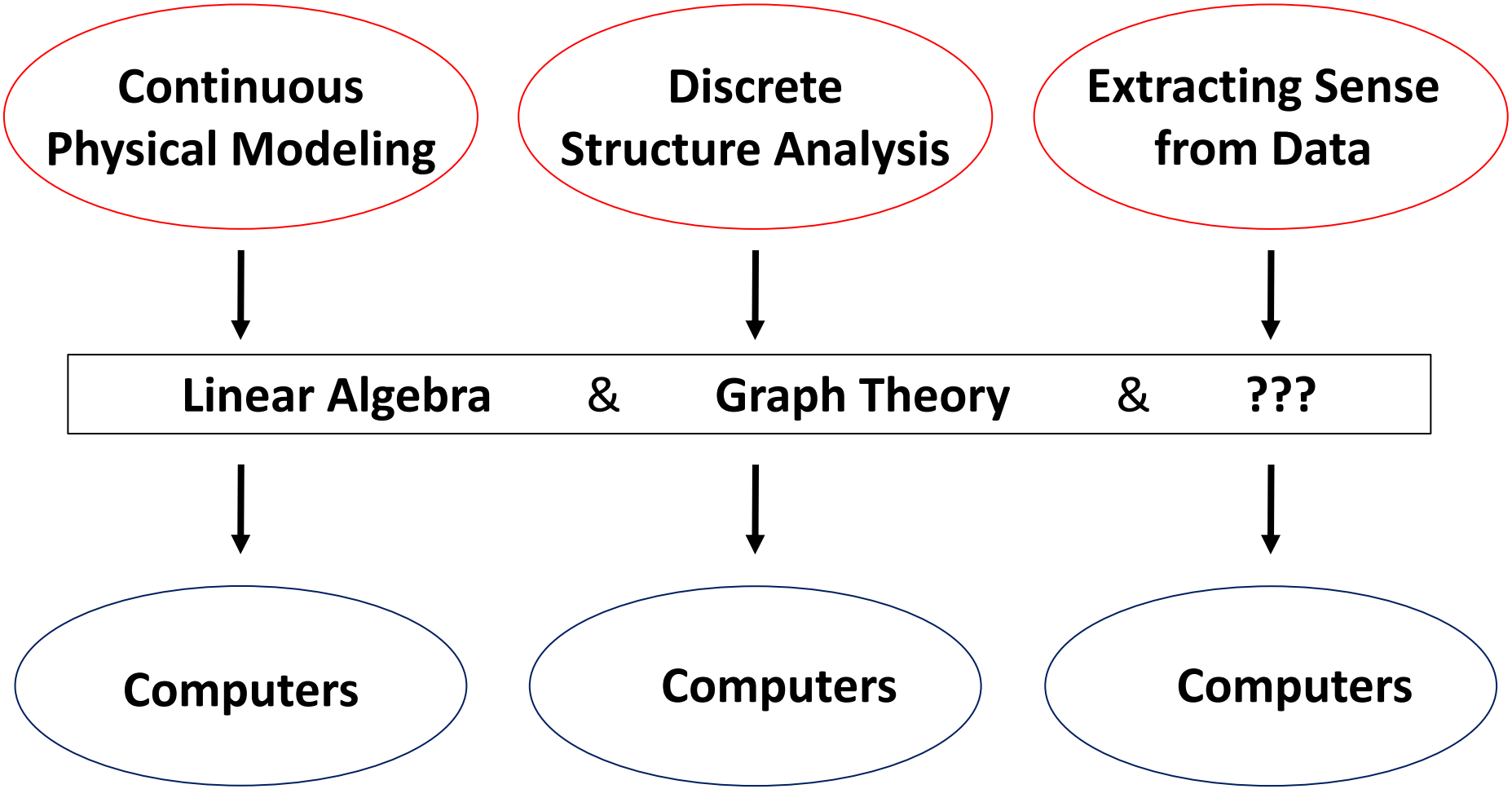
UCSB

Tomorrow



UCSB

Tomorrow



UCSB

Thanks …

Ariful Azad, David Bader, Jon Berry, Eric Boman, Aydin Buluc, Ben Chang, John Conroy, Tim Davis, Kevin Deweese, Erika Duriakova, Assefaw Gebremedhin, Shoaib Kamil, Jeremy Kepner, Tammy Kolda, Tristan Konolige, Manoj Kumar, Adam Lugowski, Tim Mattson, Scott McMillan, Henning Meyerhenke, Jose Moreira, Esmond Ng, Lenny Oliker, Weimin Ouyang, Ali Pinar, Alex Pothen, Carey Priebe, Steve Reinhardt, Lijie Ren, Eric Robinson, Viral Shah, Veronika Strnadova-Neeley, Blair Sullivan, Shang-Hua Teng, Yun Teng, Sam Williams

… and Intel, Microsoft, NSF, DOE Office of Science

UCSB