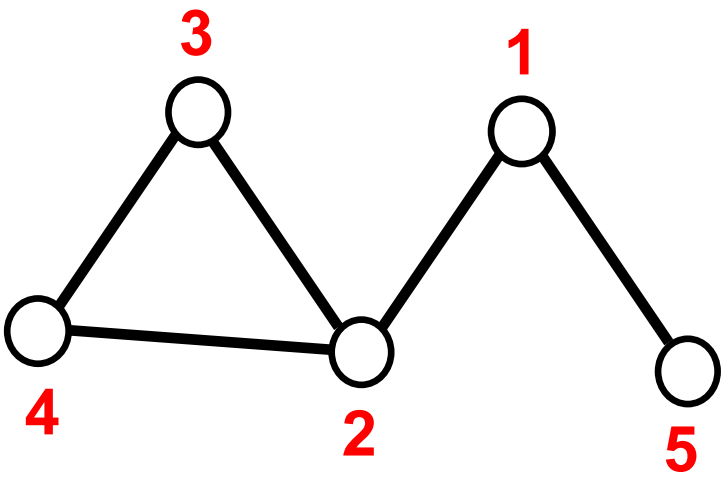
UCSB

Sparse Matrices and Graphs: There and Back Again

John Gilbert
UC Santa Barbara

University of New Mexico
April 22, 2026

Graphs and matrices

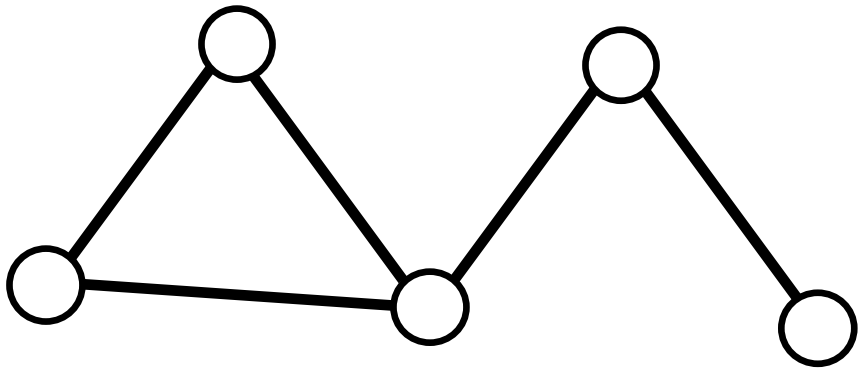


$$\begin{bmatrix} 4 & 2 & & & -4 \\ 2 & 2 & -1 & 2 & \\ & -1 & 5 & -4 & \\ & 2 & -4 & 14 & \\ -4 & & & & 11 \end{bmatrix}$$

UCSB

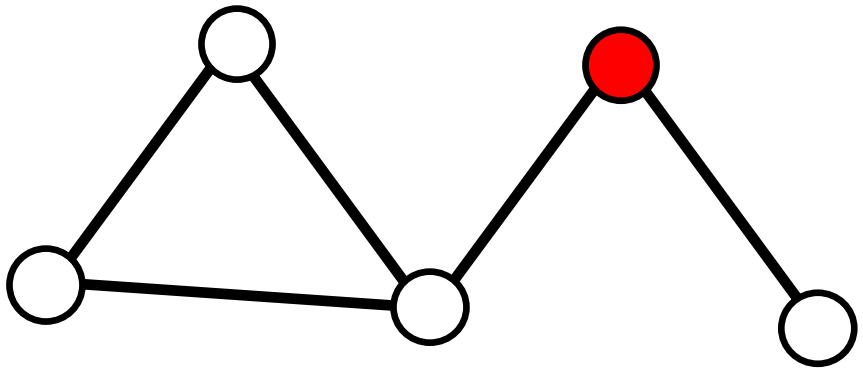
A one-person game on graphs

[S. Parter 1961]



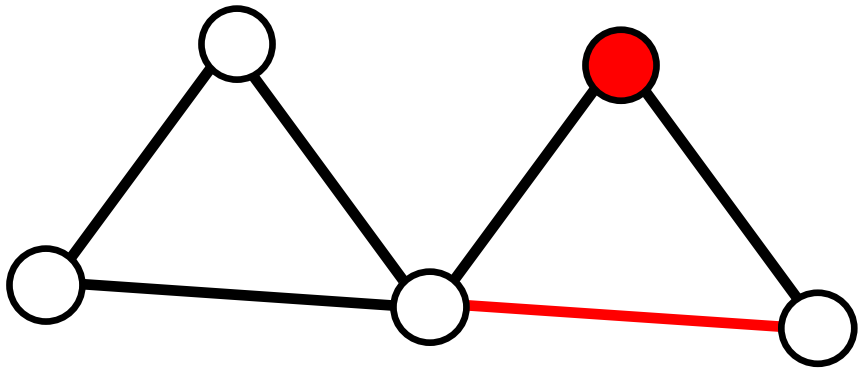
UCSB

A one-person game on graphs



- Mark a vertex.

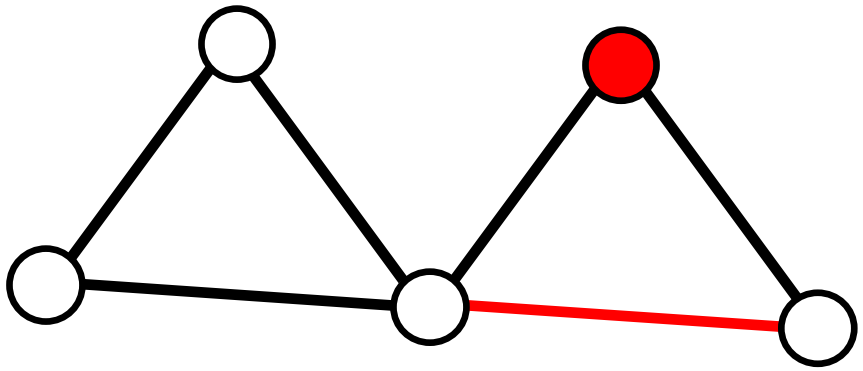
A one-person game on graphs



- Mark a vertex.
- Connect its unmarked neighbors.

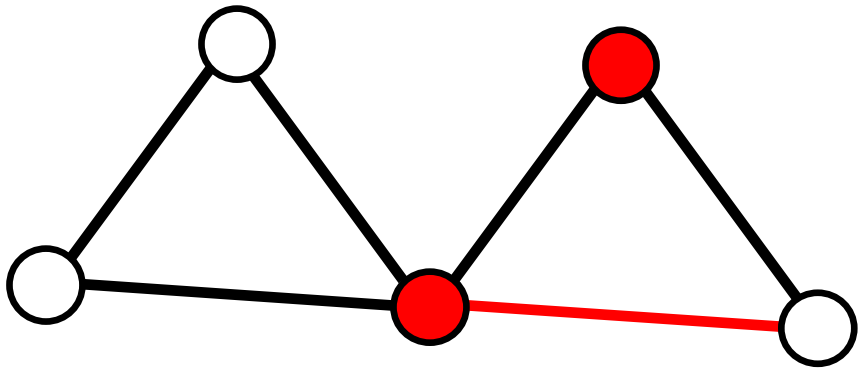
UCSB

A one-person game on graphs



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

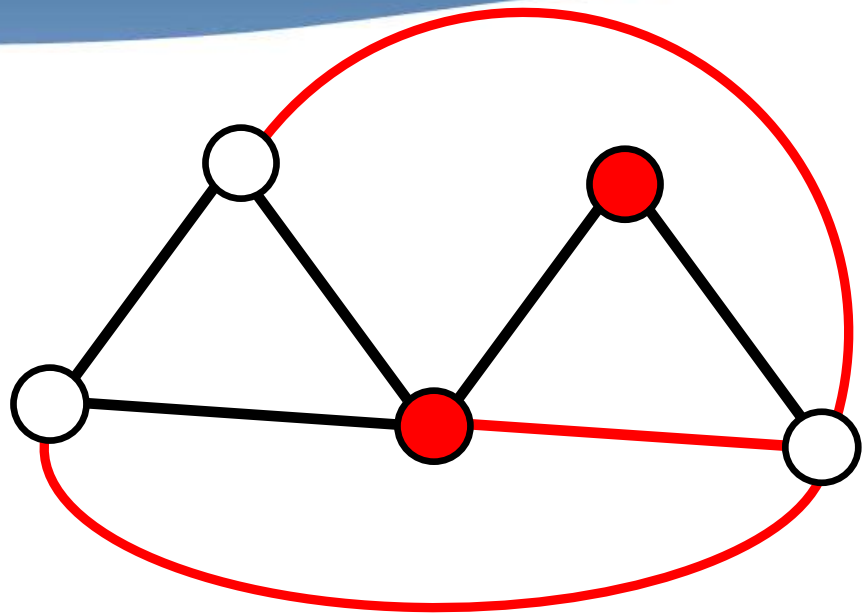
A one-person game on graphs



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

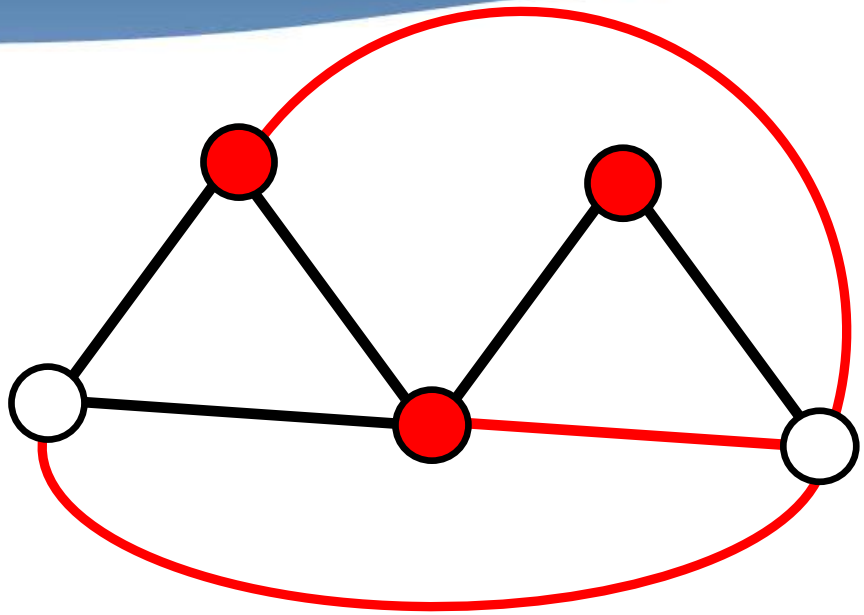
UCSB

A one-person game on graphs



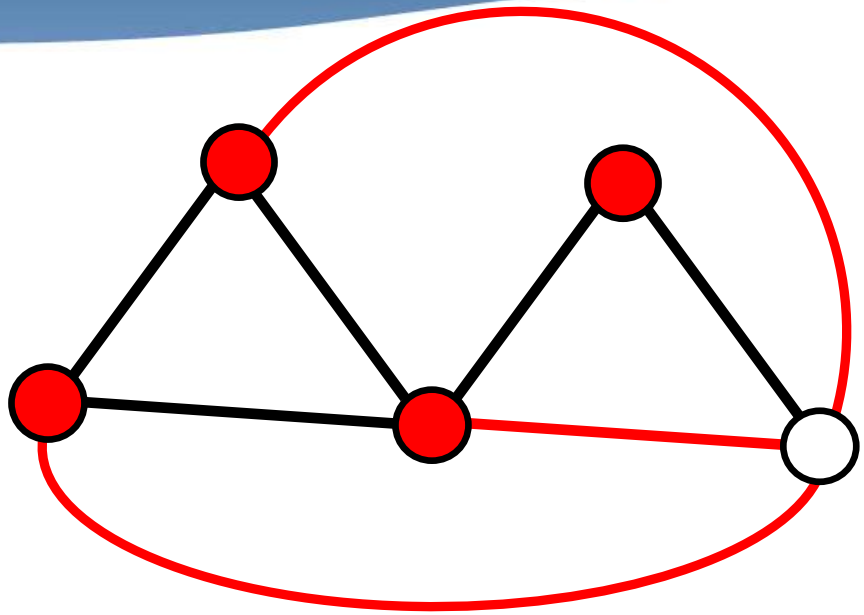
- Mark a vertex.
- Connect its unmarked neighbors.
- Repeat.

A one-person game on graphs



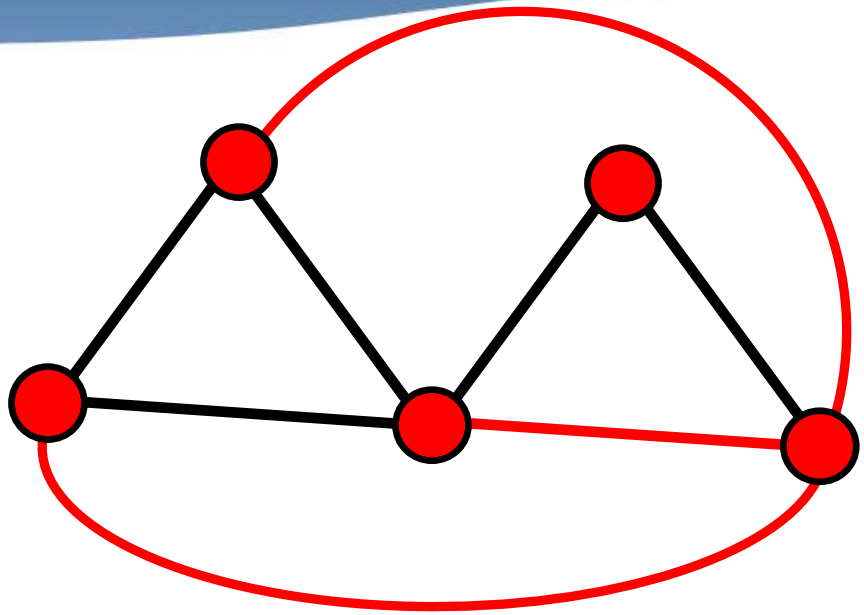
- Mark a vertex.
- Connect its unmarked neighbors.
- Repeat.

A one-person game on graphs



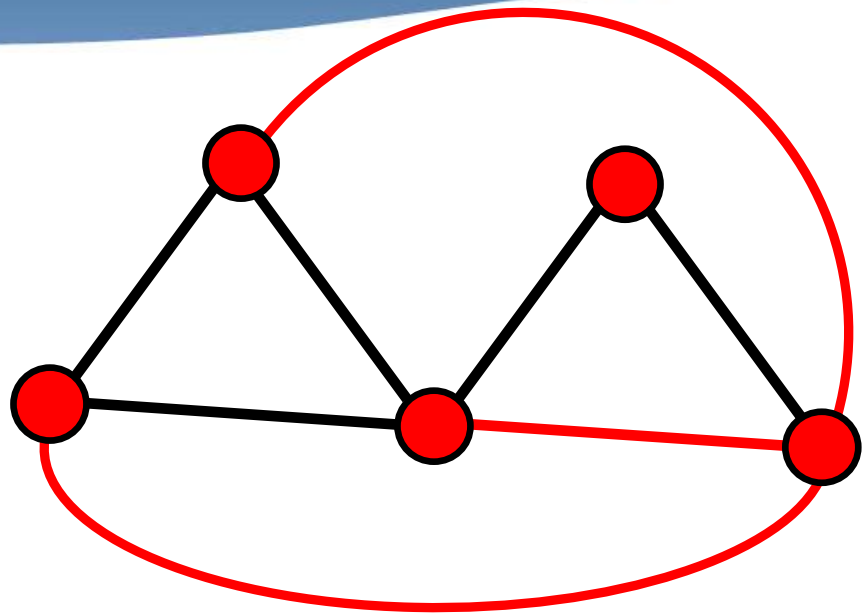
- Mark a vertex.
- Connect its unmarked neighbors.
- Repeat.

A one-person game on graphs



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

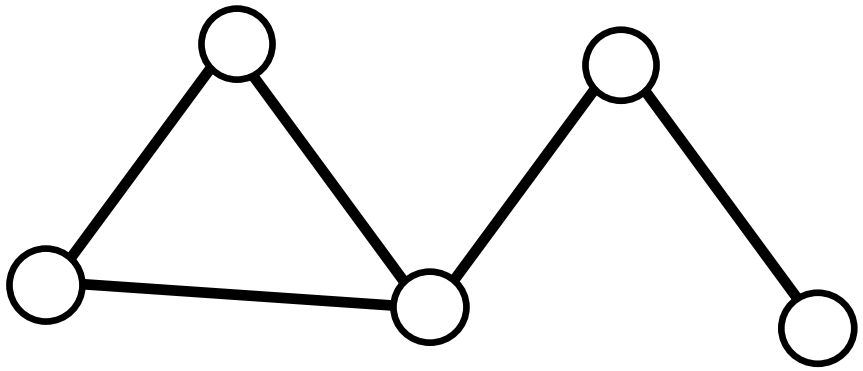
A one-person game on graphs



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

Goal: End up with as few edges as possible.

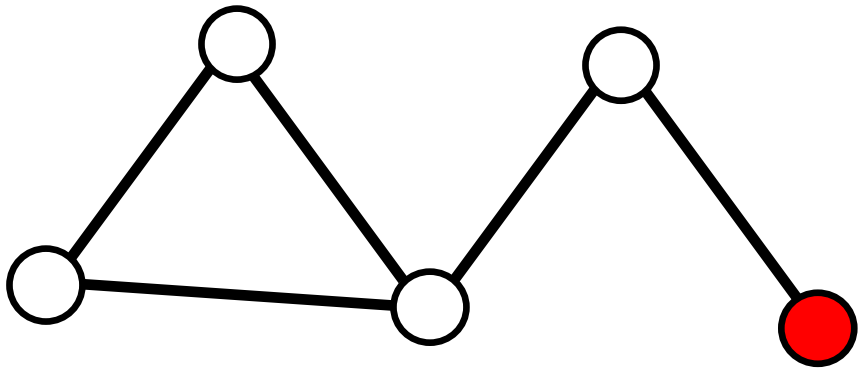
A one-person game on graphs



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

Goal: End up with as few edges as possible.

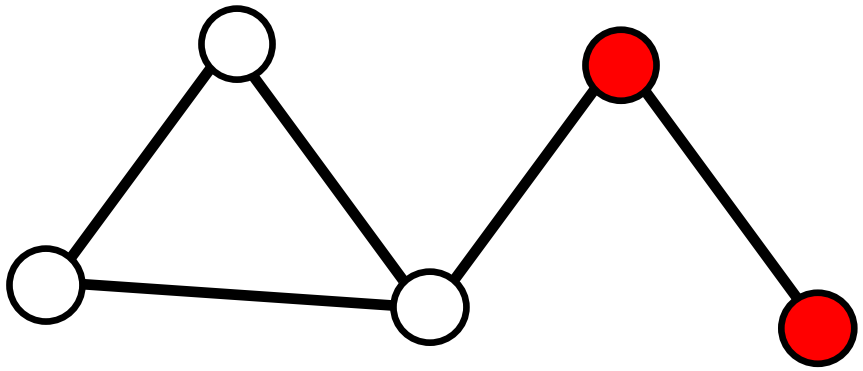
A one-person game on graphs



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

Goal: End up with as few edges as possible.

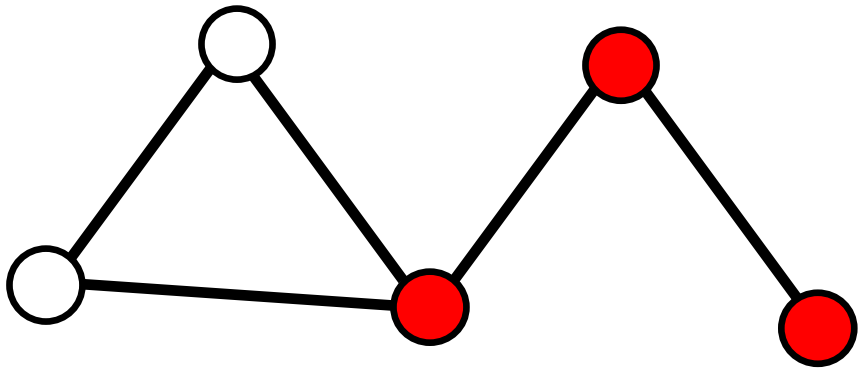
A one-person game on graphs



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

Goal: End up with as few edges as possible.

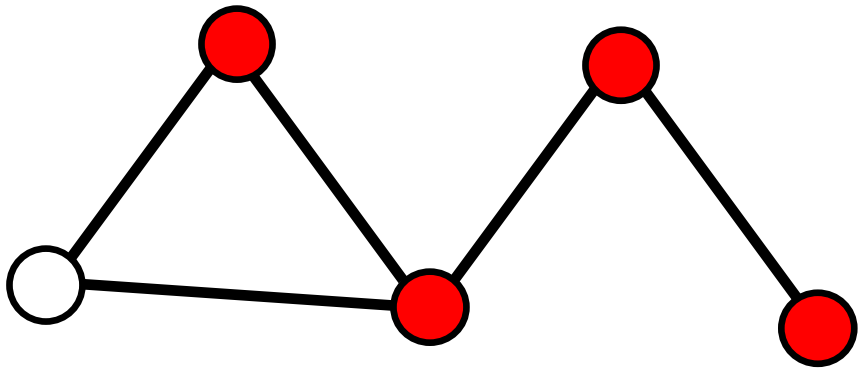
A one-person game on graphs



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

Goal: End up with as few edges as possible.

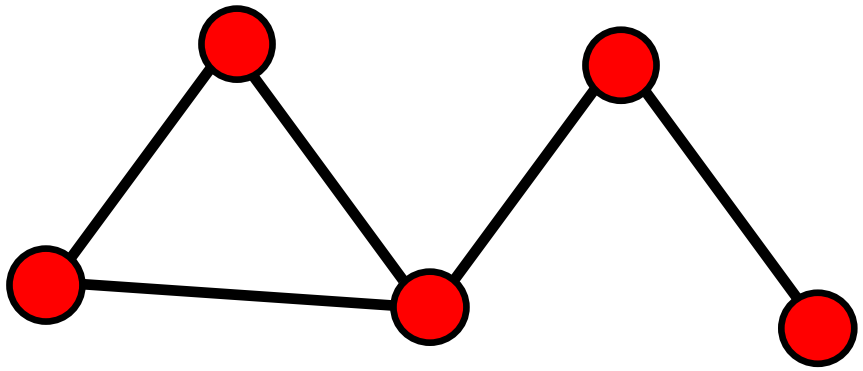
A one-person game on graphs



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

Goal: End up with as few edges as possible.

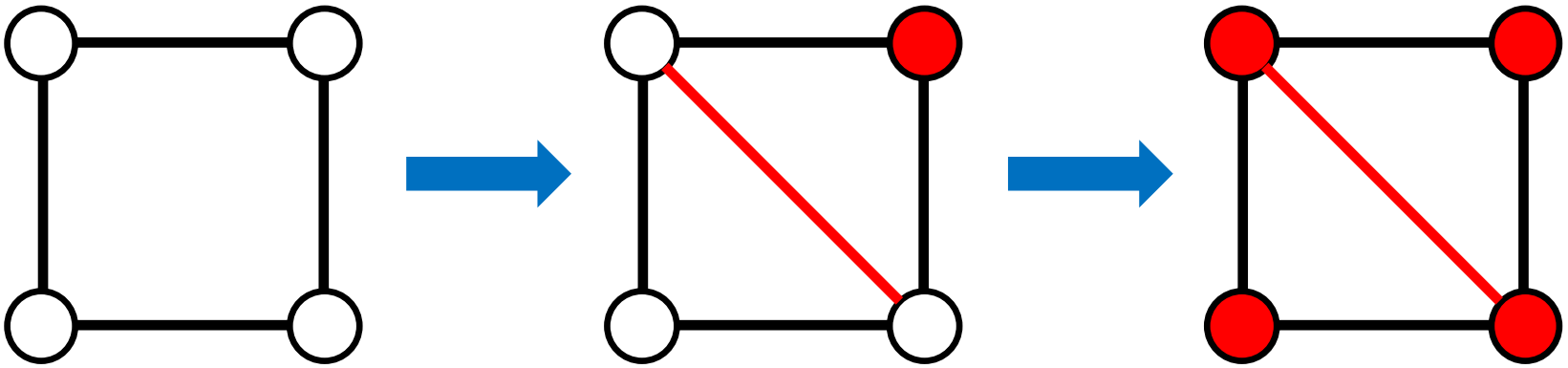
A one-person game on graphs



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

Goal: End up with as few edges as possible.

Sometimes (usually) no perfect game is possible



UCSB

Vertex elimination game (or chordal completion)

[Parter, Rose]

Repeat:

Choose a vertex v and mark it;

Add edges between unmarked neighbors of v;

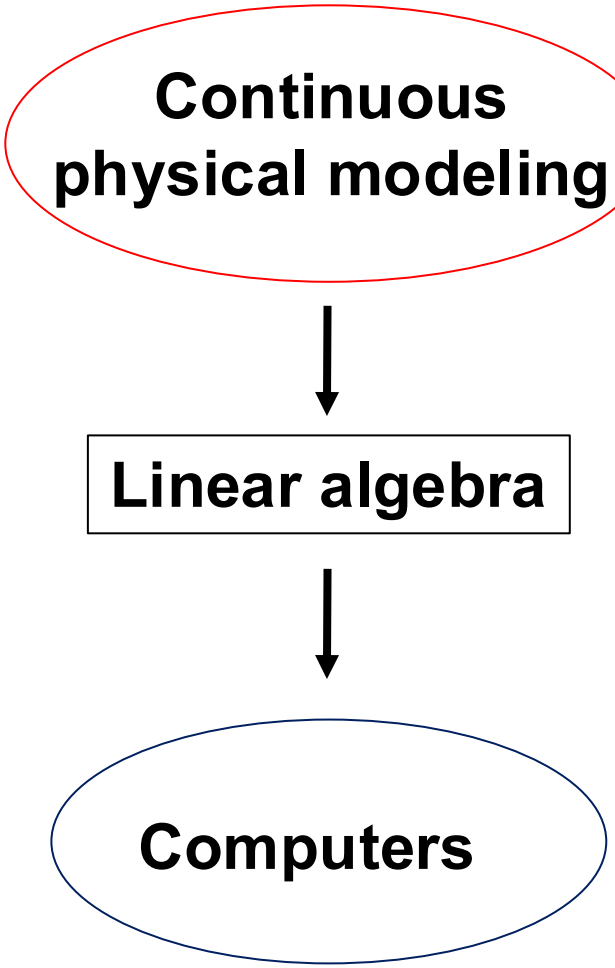
Until: Every vertex is marked

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.

$$A = \begin{bmatrix} 4 & 2 & & & -4 \\ 2 & 2 & -1 & 2 & \\ & -1 & 5 & -4 & \\ & 2 & -4 & 14 & \\ -4 & & & & 11 \end{bmatrix}$$

The middleware of scientific computing



$$Ax = b$$

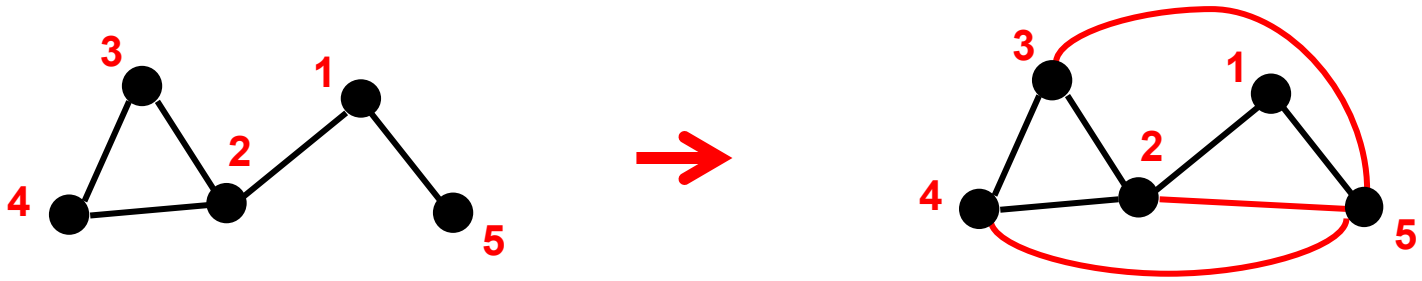
UCSB

Solving $Ax = b$ by Cholesky factorization

$$\begin{bmatrix} 4 & 2 & & & -4 \\ 2 & 2 & -1 & 2 & \\ & -1 & 5 & -4 & \\ & 2 & -4 & 14 & \\ -4 & & & & 11 \end{bmatrix} = \begin{bmatrix} 2 & & & & \\ 1 & 1 & & & \\ & -1 & 2 & & \\ & 2 & -1 & 3 & \\ -2 & 2 & 1 & -1 & 1 \end{bmatrix} \times \begin{bmatrix} 2 & 1 & & & -2 \\ & 1 & -1 & 2 & 2 \\ & & 2 & -1 & 1 \\ & & & 3 & -1 \\ & & & & 1 \end{bmatrix}$$

- Factor $A = LL^T$ by Gaussian elimination.
- Solve lower triangular $Ly = b$ for y by forward substitution.
- Solve upper triangular $L^T x = y$ for x by backward substitution.

The 1-person graph game takes A to L and L^T

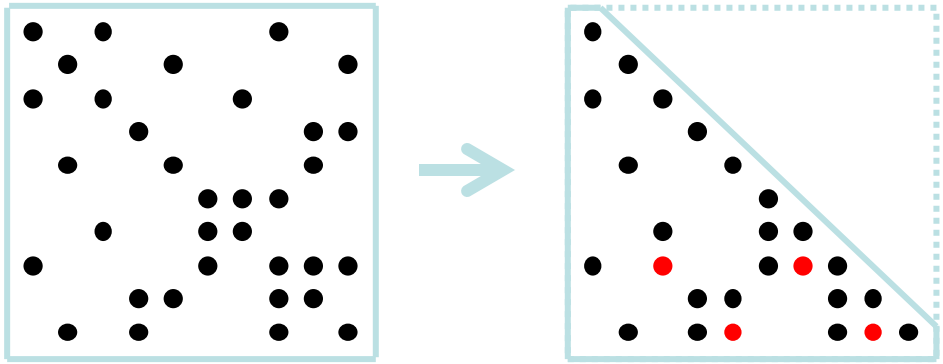


$$\begin{bmatrix} 4 & 2 & & & -4 \\ 2 & 2 & -1 & 2 & \\ & -1 & 5 & -4 & \\ & 2 & -4 & 14 & \\ -4 & & & & 11 \end{bmatrix} = \begin{bmatrix} 2 & & & & \\ 1 & 1 & & & \\ & -1 & 2 & & \\ & 2 & -1 & 3 & \\ -2 & 2 & 1 & -1 & 1 \end{bmatrix} \times \begin{bmatrix} 2 & 1 & & & -2 \\ & 1 & -1 & 2 & 2 \\ & & 2 & -1 & 1 \\ & & & 3 & -1 \\ & & & & 1 \end{bmatrix}$$

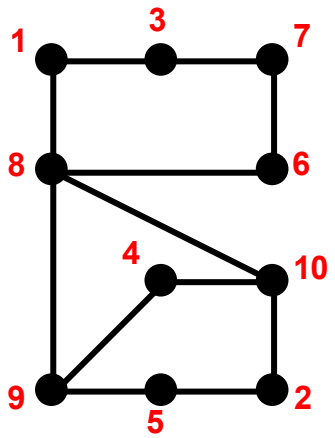
UCSB

Cholesky factorization: $A = LL^T$

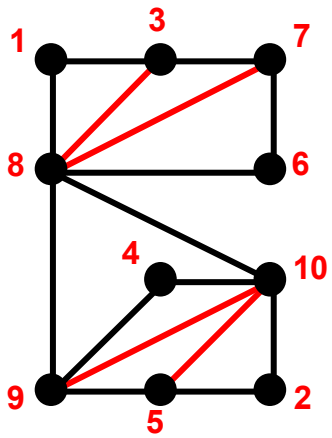
[Parter, Rose]



Fill: new nonzeros in factor



G(A)



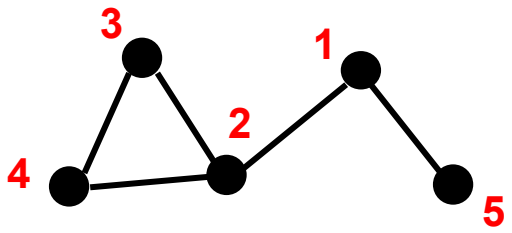
$G^+(A)$
[chordal]

Symmetric Gaussian elimination:

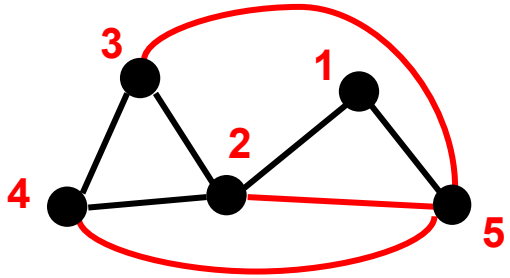
for j = 1 to n
    add edges between j's higher-numbered neighbors

UCSB

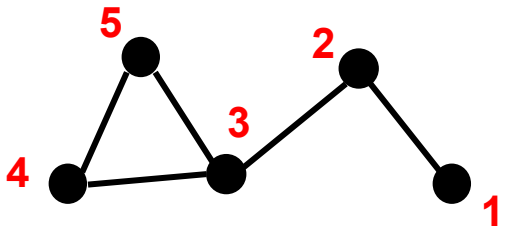
Orderings for sparse Gaussian elimination



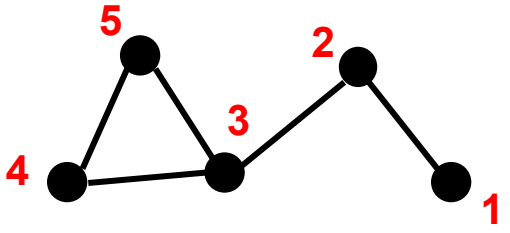
$$Ax = b$$



$$A = L_1 L_1^T$$



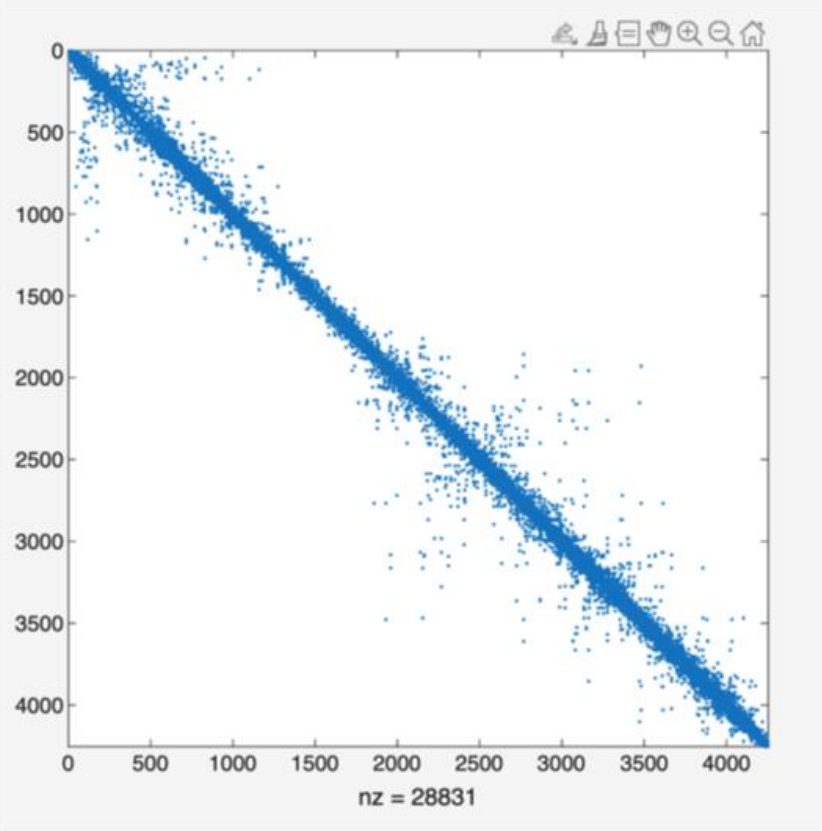
$$(PAP^T)\,(Px) = (Pb)$$

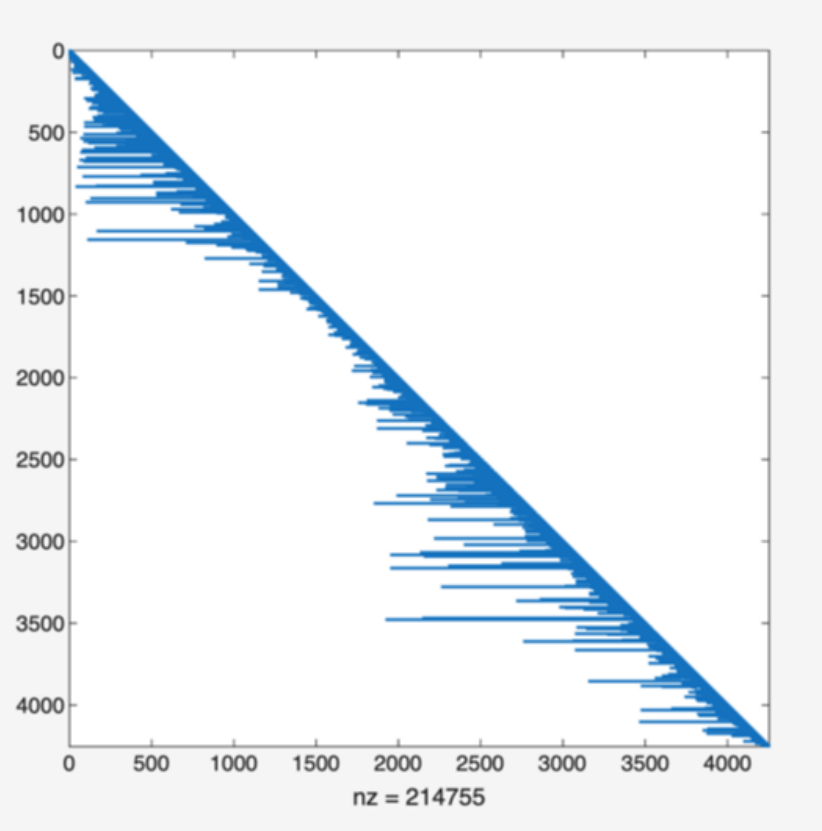


$$PAP^T = L_2 L_2^T$$

UCSB

Cholesky factor of a 4253-by-4253 matrix



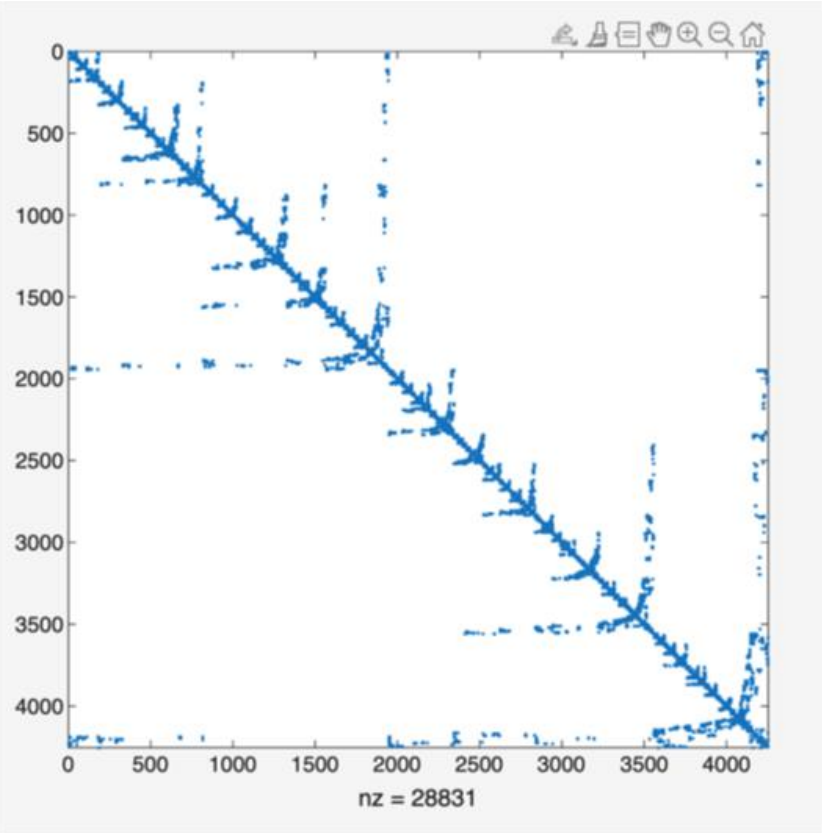


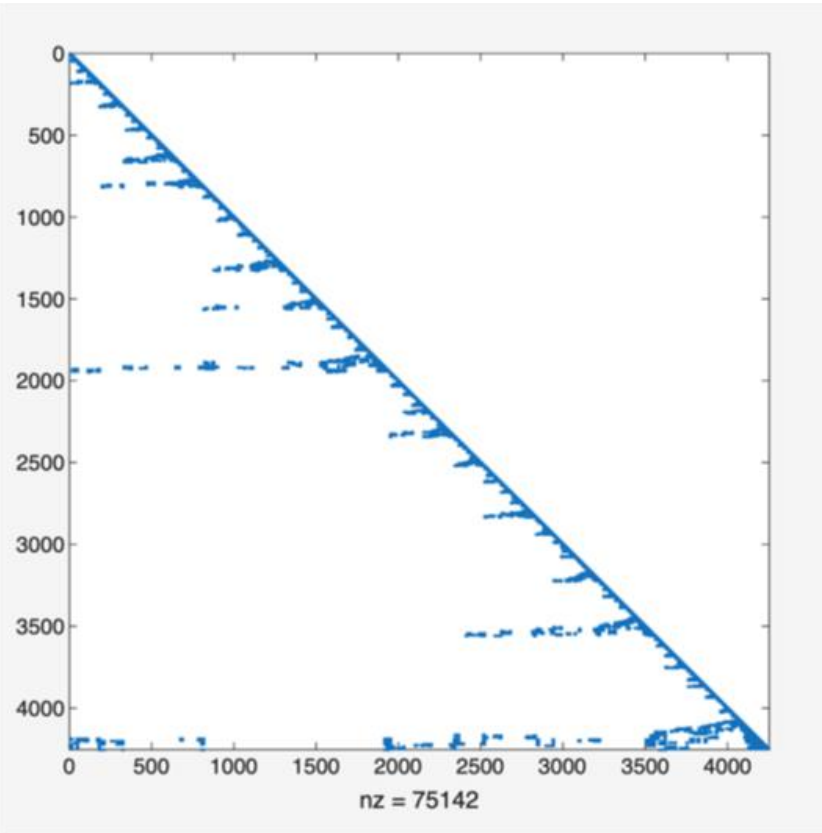
A : 28,831 nonzeros

L : 214,755 nonzeros

UCSB

Cholesky factor of the permuted matrix





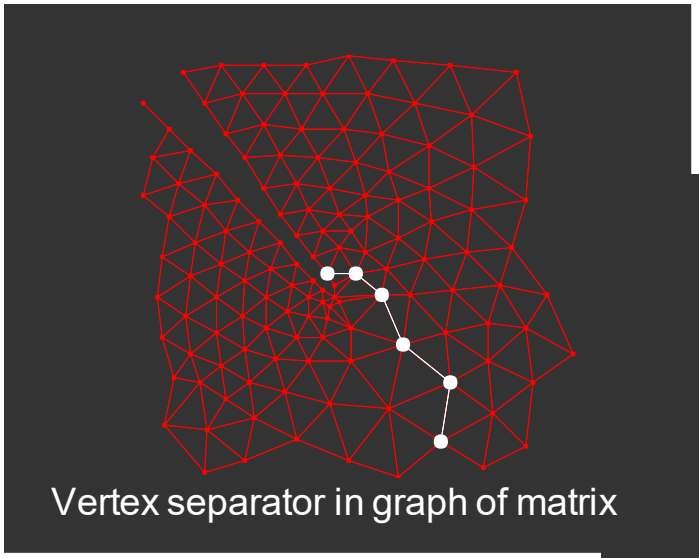
A : 28,831 nonzeros

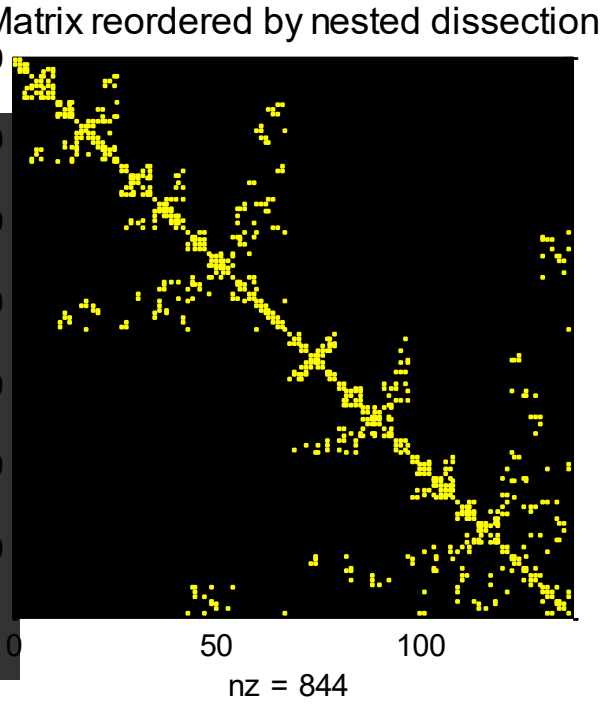
L : 75,142 nonzeros

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

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

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), DulmageMendelsohn decomposition / strong components, …
- Iterative methods: graph partitioning again, independent set, low-stretch spanning trees, …

1970's and 1980's: Sparse direct methods

- Preordering and symbolic factorization to make a big numerical computation less expensive
- Efficient sequential graph algorithms, often based on path search (depth-first, breadth-first, alternating paths, Dijkstra, …)
- Careful "classical" data structure design
- Efficiency mostly follows asymptotic complexity analysis, with some attention to memory hierarchy
- E.g. Parter's theorem (1961), nested dissection, supernodal methods

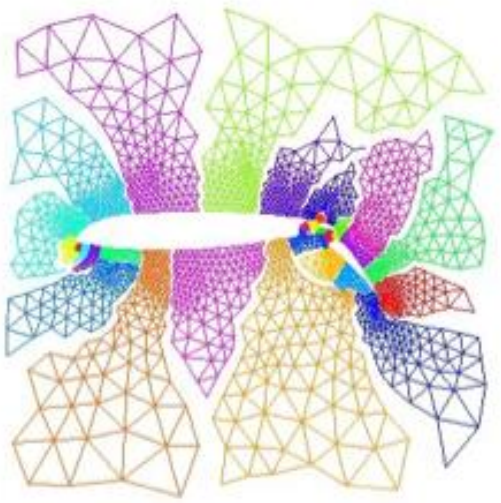
1990's and 2000's: Parallel scientific simulation

- Graph algorithms for partitioning and scheduling
- The graph algorithms now have to be parallel too!
- Multilevel graph algorithms for partitioning etc.

  (graph contraction, heuristic matching,

  not based on global path search)
- Challenges of distributed memory:

  locality, small messages, etc.
- E.g. Chaco, MUMPS, ParMetis, Trilinos



2010's and 2020's: Network and graph analytics

- Social networks, biology, infrastructure, finance, cybersecurity
- The graph becomes the main event!

  (not just preprocessing for efficient numerical computation)
- Streaming and sampling algorithms, dynamic graphs, pattern matching,

  metadata, graph databases, spectral analysis & signal processing
- Explosion of tools and libraries, from Pregel to NetworkX to GraphBLAS

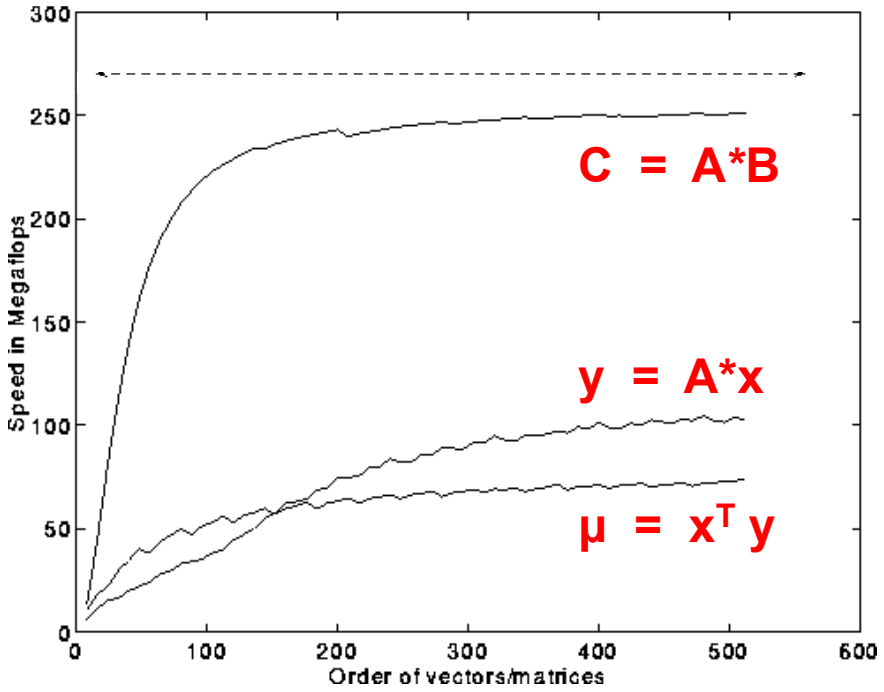
The middleware challenge for graph analysis

The BLAS had a revolutionary impact on computational linear algebra.

Separation of concerns:

- Experts in mapping algorithms onto hardware tuned BLAS to specific platforms.
- Experts in linear algebra built software on top of the BLAS to obtain high performance "for free".

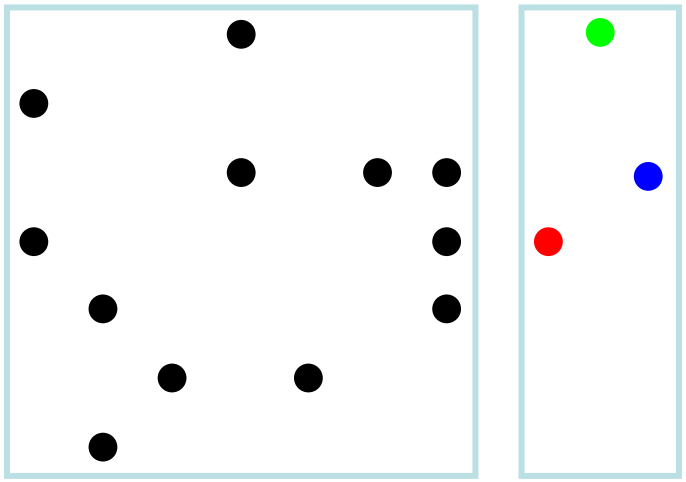
Basic Linear Algebra Subroutines (BLAS):
Ops/Sec vs. Matrix Size



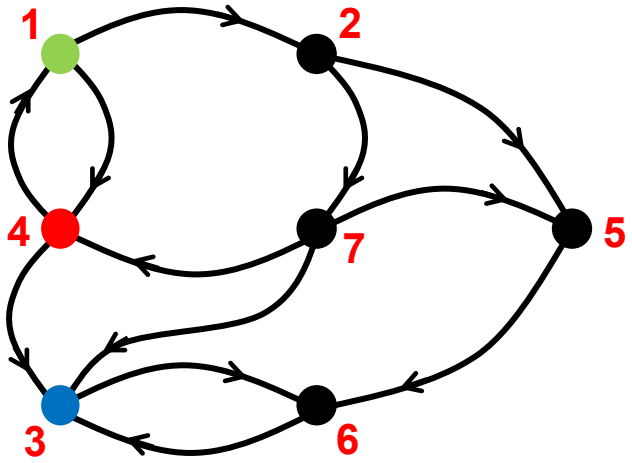
What should the combinatorial BLAS look like?

UCSB

Multiple-source breadth-first search



A^T X

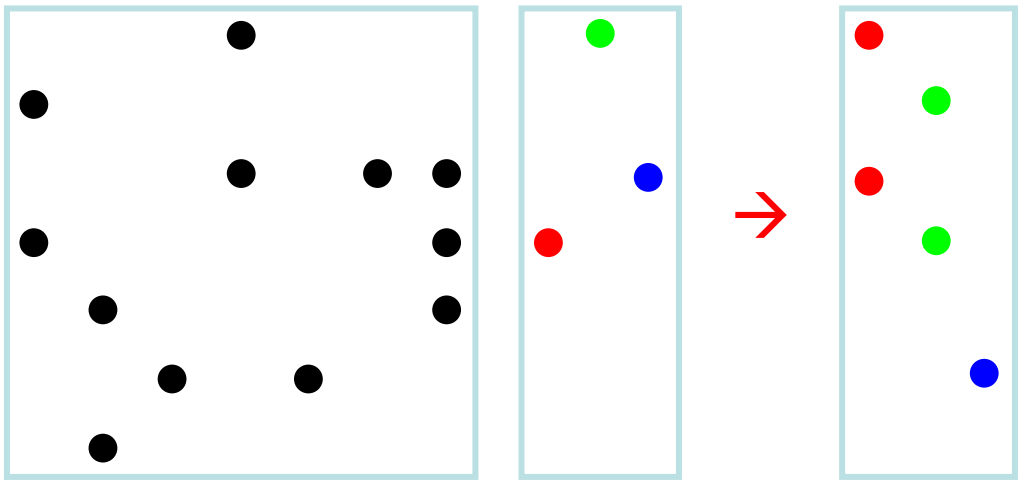


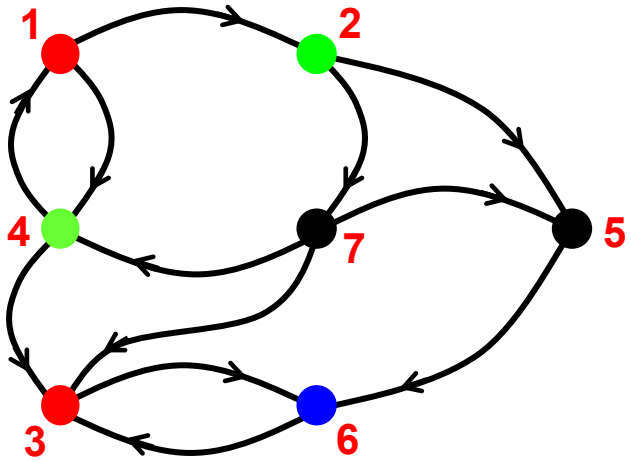
A^T is the transpose of the graph's adjacency matrix.

Each column of X picks out one source vertex for the search.

UCSB

Multiple-source breadth-first search



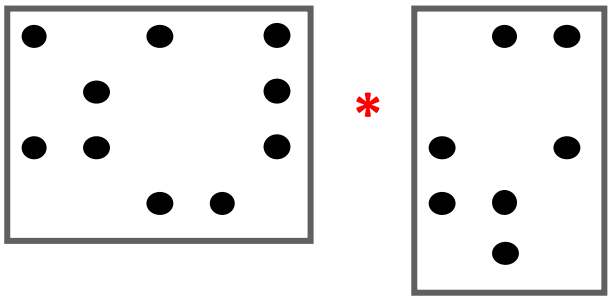


- Sparse array representation => space efficient
- Sparse matrix-matrix multiplication => work efficient
- Three possible levels of parallelism: searches, vertices, edges

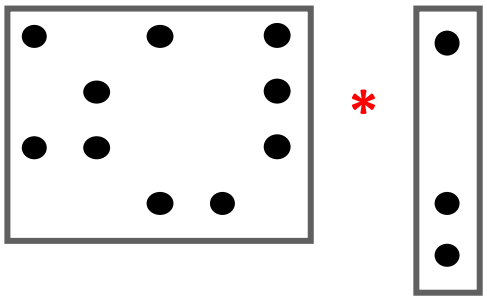
UCSB

Sparse array primitives for graphs

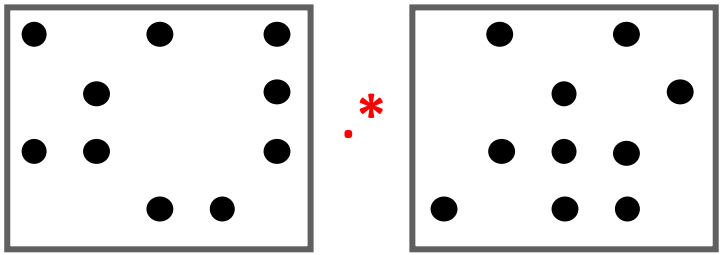
Sparse matrix-sparse matrix multiplication



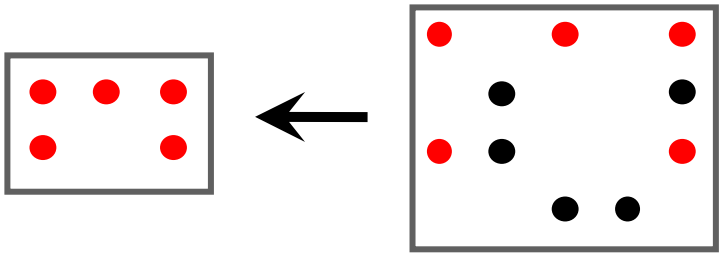
Sparse matrix-sparse vector multiplication



Element-wise operations



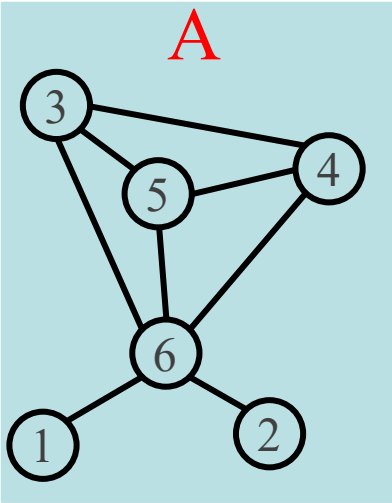
Sparse matrix indexing



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

UCSB

Counting triangles (clustering coefficient)

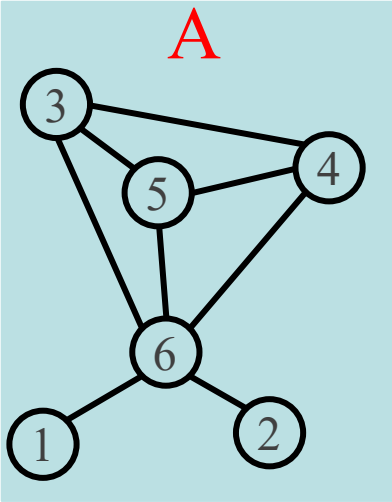


Clustering coefficient:

- 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

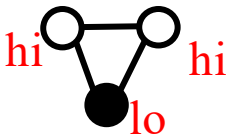
Counting triangles (clustering coefficient)



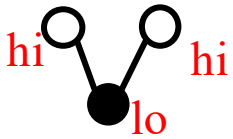
Clustering coefficient:

- 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

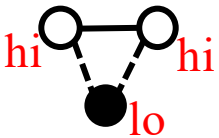
One algorithm to count triangles:



- Count triangles by lowest-degree vertex.



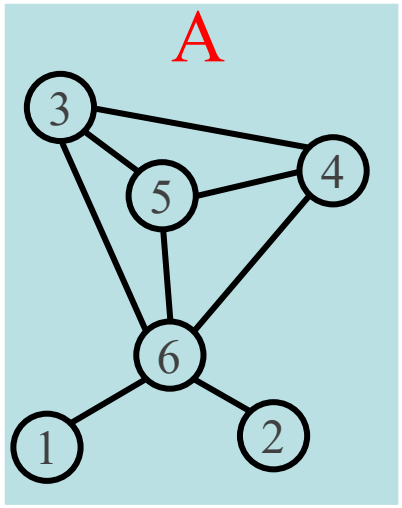
- Enumerate "low-hinged" wedges.



- Keep wedges that close.

UCSB

Counting triangles (clustering coefficient)

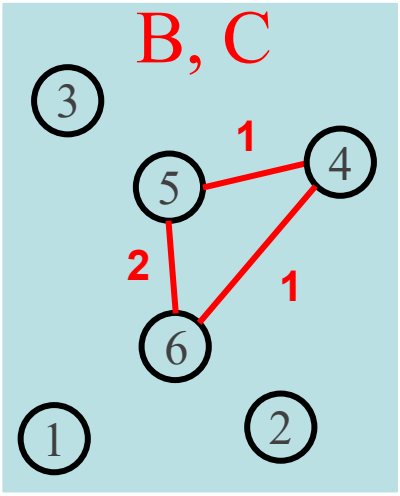


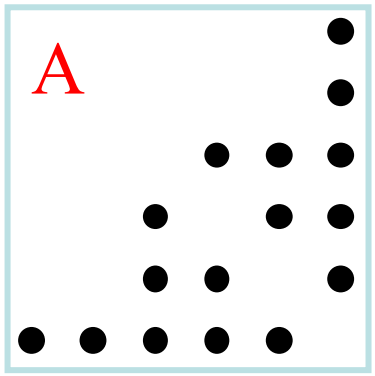
$A = L + U$ (hi->lo + lo->hi)

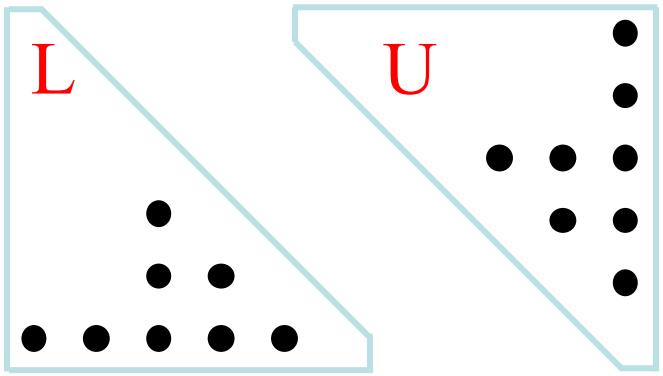
$L \times U = B$ (wedge, low hinge)

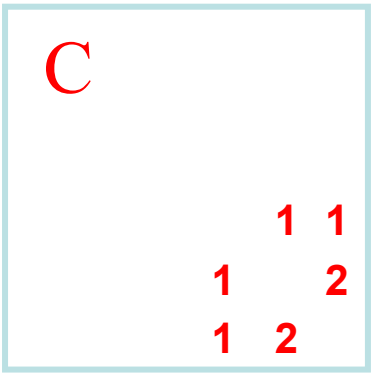
$A \wedge B = C$ (closed wedge)

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





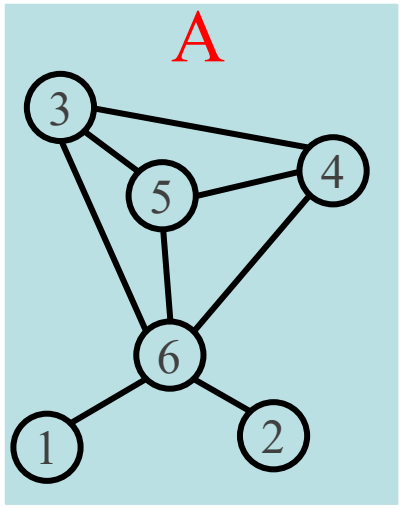




A. Azad, A. Buluç, JG. Parallel triangle counting and enumeration using matrix algebra. *IPDPSW 2015*.

UCSB

Counting triangles (clustering coefficient)

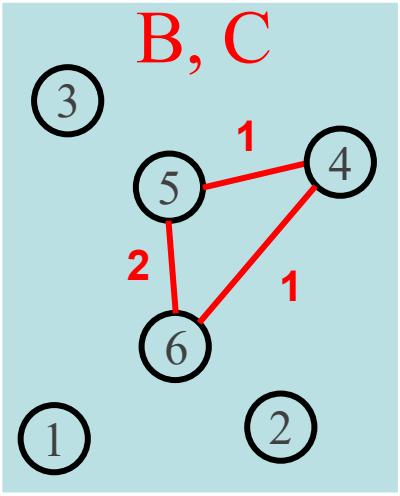


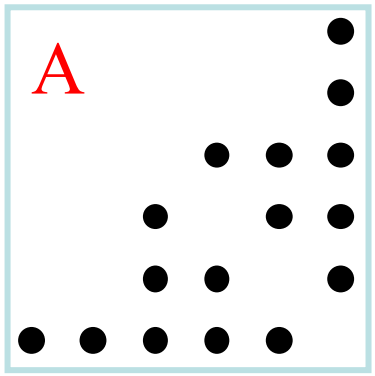
$A = L + U$ (hi->lo + lo->hi)

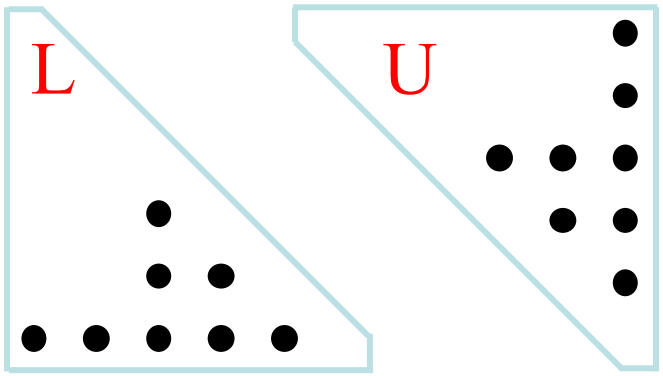
$L \times U = B$ (wedge, low hinge)

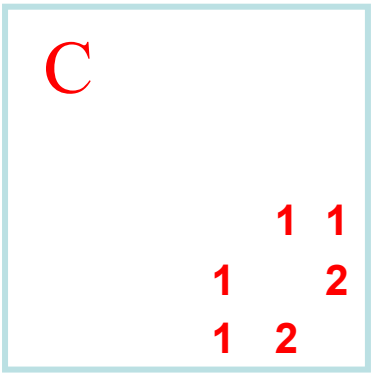
$A \wedge B = C$ (closed wedge)

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









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

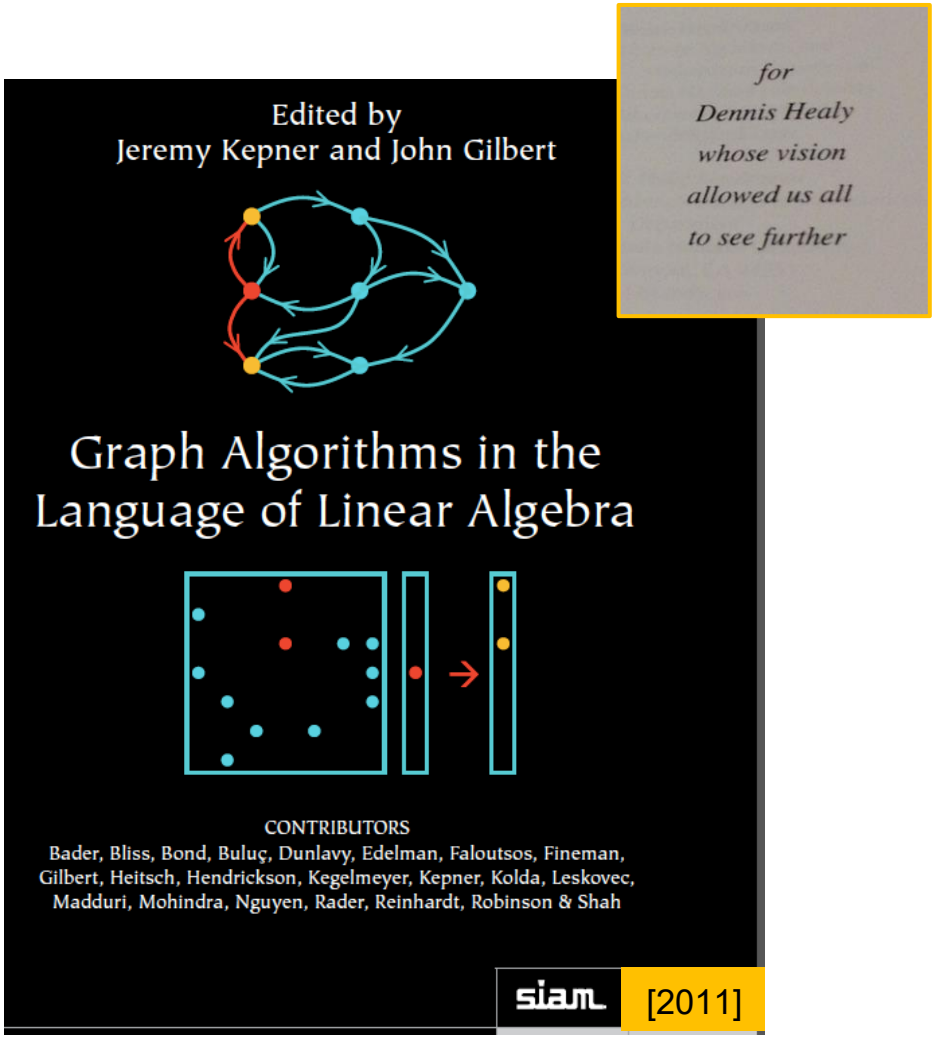
UCSB

Examples of semirings in graph algorithms

Real field: **$(\mathbb{R}, +, \times)$**	Numerical linear algebra
Boolean algebra: **({0 1}, or, and)**	Connectivity & traversal
Tropical semiring: **$(\mathbb{R} \cup \{\infty\}, \min, +)$**	Shortest paths
(S, select, select)	Select subgraph, or contract nodes to form quotient graph
$(\mathbb{R}, \max, +)$	Graph matching & network alignment
$(\mathbb{R}, \max, \times)$	Maximal independent set
"values": edge/vertex attributes, "add": vertex data aggregation, "multiply": edge data processing	General schema for user-specified computation at vertices and edges

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

But why do it this way in practice?

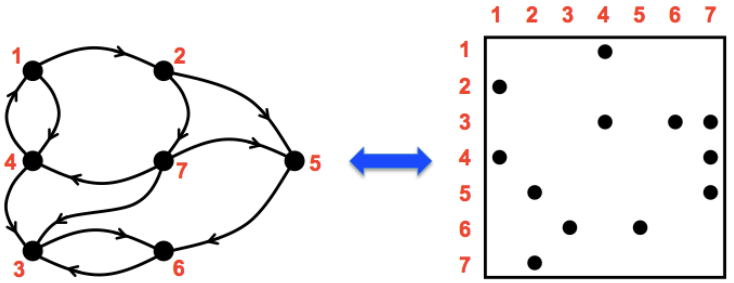
Picking the right level of abstraction:

High enough to optimize,

Low enough to be broadly useful

Vertex/edge graph computations	Graphs in the language of linear algebra
Unpredictable, data-driven communication patterns	Fixed communication patterns
Irregular data accesses, with poor locality	Matrix block operations exploit memory hierarchy
Fine grained data accesses, dominated by latency	Coarse grained parallelism, limited by bandwidth not latency

Combinatorial BLAS (2010)



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 parallel supercomputer.

- Open source software: people.eecs.berkeley.edu/~aydin/CombBLAS/html/
- Version 2.0 released July 2022.

A. Buluç, JG. The Combinatorial BLAS: Design, implementation, and applications. *Int. J. HPC Appl. 2011*.

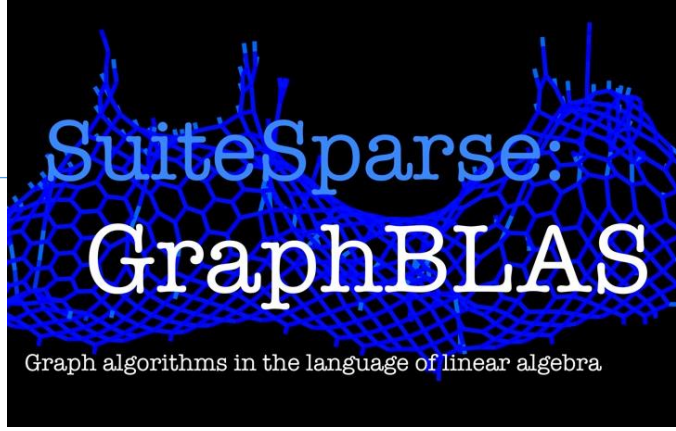
GraphBLAS:

Standard building blocks for graphs as linear algebra

Operation name	Mathematical description
mxm	$\mathbf{C} \odot= \mathbf{A} \oplus . \otimes \mathbf{B}$
mxv	$\mathbf{w} \odot= \mathbf{A} \oplus . \otimes \mathbf{v}$
vxm	$\mathbf{w}^T \odot= \mathbf{v}^T \oplus . \otimes \mathbf{A}$
eWiseMult	$\mathbf{C} \odot= \mathbf{A} \otimes \mathbf{B}$
	$\mathbf{w} \odot= \mathbf{u} \otimes \mathbf{v}$
eWiseAdd	$\mathbf{C} \odot= \mathbf{A} \oplus \mathbf{B}$
	$\mathbf{w} \odot= \mathbf{u} \oplus \mathbf{v}$
reduce (row)	$\mathbf{w} \odot= \bigoplus_j \mathbf{A}(:, j)$
apply	$\mathbf{C} \odot= F_u(\mathbf{A})$
	$\mathbf{w} \odot= F_u(\mathbf{u})$
transpose	$\mathbf{C} \odot= \mathbf{A}^T$
extract	$\mathbf{C} \odot= \mathbf{A}(\mathbf{i}, \mathbf{j})$
	$\mathbf{w} \odot= \mathbf{u}(\mathbf{i})$
assign	$\mathbf{C}(\mathbf{i}, \mathbf{j}) \odot= \mathbf{A}$
	$\mathbf{w}(\mathbf{i}) \odot= \mathbf{u}$

- Operators: $\oplus$, $\otimes$: semiring “add” and “multiply”, $\odot$: “accumulate”
- Objects: matrix, vector, monoid, semiring, ...

SuiteSparse:GraphBLAS



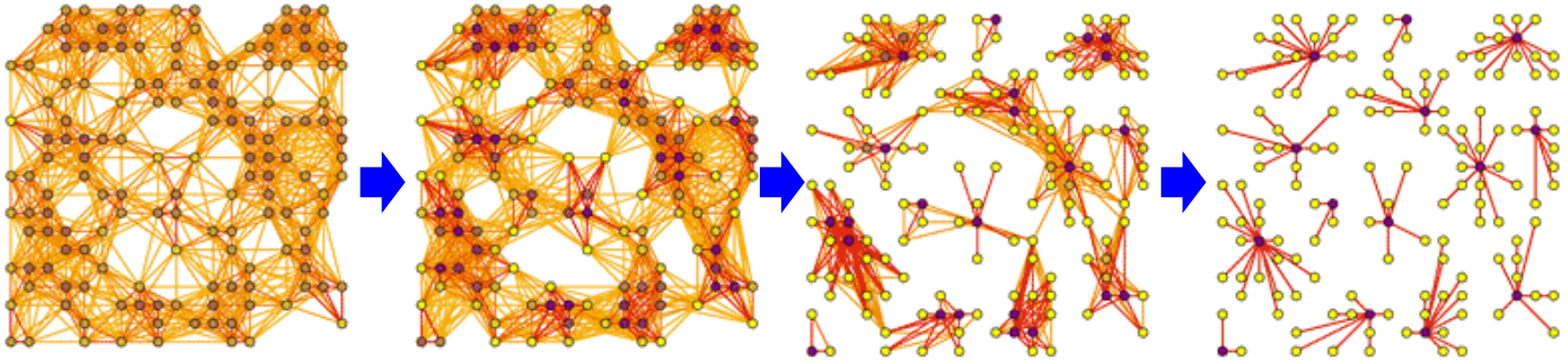
- Tim Davis group (Texas A&M)
- Conforming implementation of C API
- Features:
  - >1000 semirings built in; also user-defined semirings
  - Fast updates using non-blocking mode and “zombies”
  - Several sparse and “hypersparse” data structures
- Multithreaded (via OpenMP), GPU (via CUDA JIT)
- Performance on graph benchmarks (e.g. triangles, k-truss) comparable to highly-tuned custom C code
- Included in Debian and Ubuntu Linux distributions
- Used as computational engine in various commercial products
- Version 10.3.1 released January 2026

T. Davis. Algorithm 1000: SuiteSparse:GraphBLAS. *ACM Trans. Math. Software,* 2019.

Markov clustering algorithm

Aydin Buluç et al. (Berkeley)

Popular and successful algorithm for discovering clusters in protein interaction and protein similarity networks



At each iteration:

Step 1 (Expansion): Square the matrix

Step 2 (Pruning): Remove small entries and dense columns

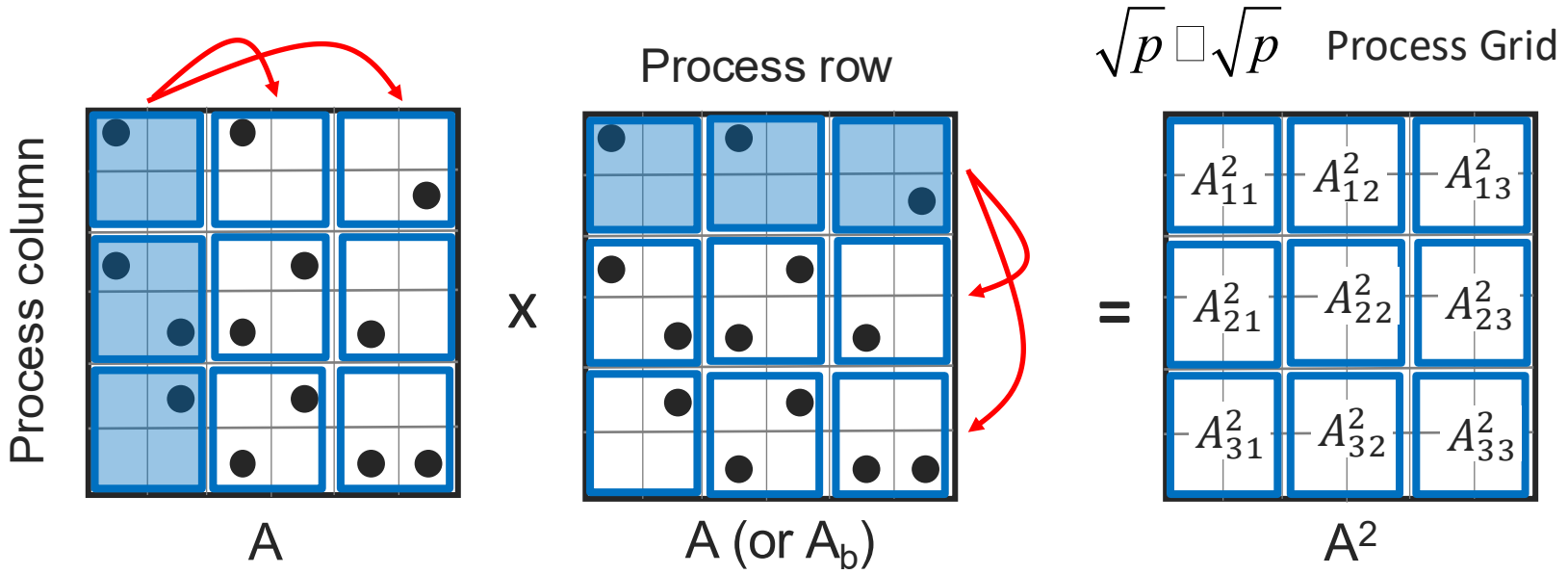
Step 3 (Inflation): Take powers entry-wise

Naïve implementation: sparse matrix-matrix product (SpGEMM), followed by column-wise top-k selection and column-wise pruning

HipMCL: High-performance Markov clustering

Buluç et al.

- MCL is both **computationally expensive** and **memory hungry**, limiting the sizes of networks that can be clustered.
- HipMCL overcomes these limitations via **sparse parallel algorithms**, with a combined expansion and pruning step.
- **Up to 1000X times faster** than original MCL with same accuracy.



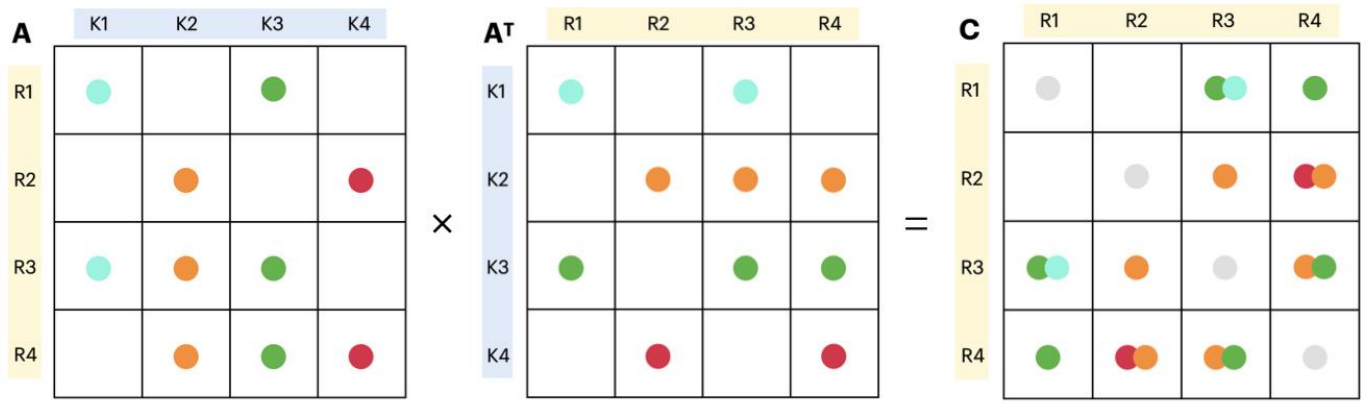
A. Azad, G. Pavlopoulos, C. Ouzounis, N. Kyrpides, A. Buluç. HipMCL: A high-performance parallel implementation of the Markov clustering algorithm for large-scale networks. *Nucleic Acids Research, 2018*

Giulia Guidi et al. (Cornell)

De novo genome assembly via sparse matrices

In de novo genome assembly, a complete DNA sequence is created from scratch from fragmented pieces of DNA **without a reference genome**

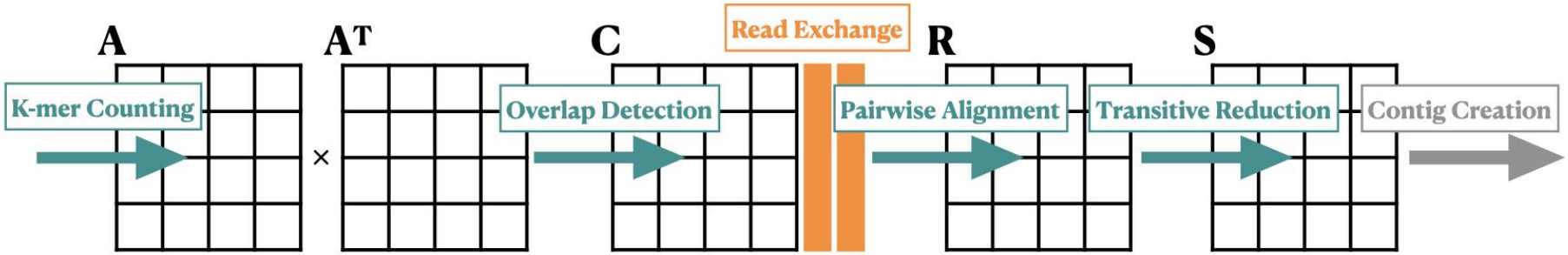
If we multiply* **A** by its transpose, we obtain a **|sequences|-by-|sequences|** matrix **C**:



Giulia Guidi et al. (Cornell)

Genome assembly via sparse matrix abstraction

In **ELBA,** we address the challenges of unstructured irregular computation **by using sparse matrices and linear algebra** as core data structures and computation

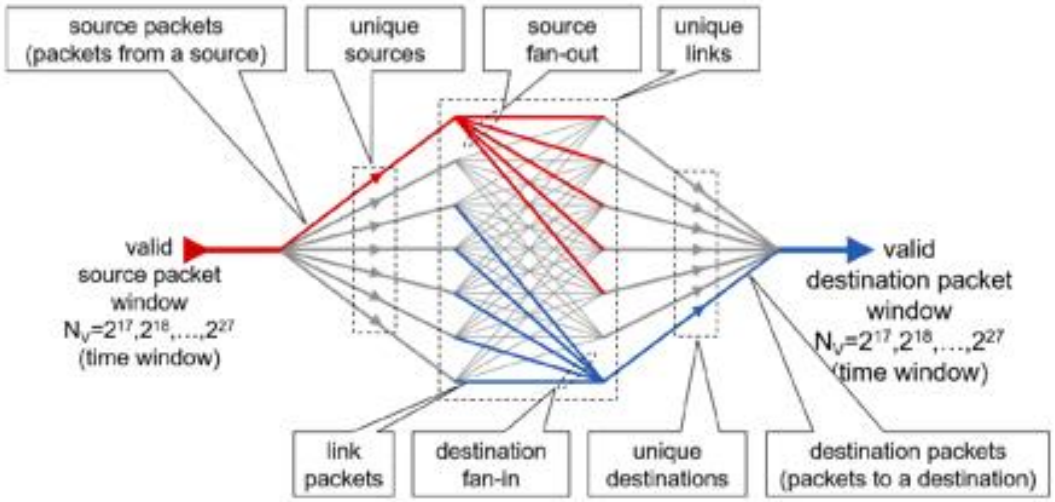


By using sparse matrices, we were able to achieve:

- **high scalability**,
- **high productivity**,
- and **significant speedup** on distributed memory architectures

Guidi, Giulia, et al. "Parallel string graph construction and transitive reduction for de novo genome assembly." IEEE IPDPS 2021.
Guidi, Giulia, et al. "BELLA: Berkeley efficient long-read to long-read aligner and overlapper." SIAM ACDA 2021.

GraphBLAS Hypersparse Math



Aggregate Property	Summation Notation	Matrix Notation
Valid packets N_V	$\sum_i \sum_j \mathbf{A}_t(i,j)$	$\mathbf{1}^\mathsf{T}\mathbf{A}_t\mathbf{1}$
Unique links	$\sum_i \sum_j \lvert\mathbf{A}_t(i,j)\rvert_0$	$\mathbf{1}^\mathsf{T}\lvert\mathbf{A}_t\rvert_0\mathbf{1}$
Link packets from i to j	$\mathbf{A}_t(i,j)$	$\mathbf{A}_t$
Max link packets ($d_{\max}$)	$\max_{ij} \mathbf{A}_t(i,j)$	$\max(\mathbf{A}_t)$
Unique sources	$\sum_i \lvert\sum_j \mathbf{A}_t(i,j)\rvert_0$	$\mathbf{1}^\mathsf{T}\lvert\mathbf{A}_t\mathbf{1}\rvert_0$
Packets from source i	$\sum_j \mathbf{A}_t(i,j)$	$\mathbf{A}_t\mathbf{1}$
Max source packets ($d_{\max}$)	$\max_i \sum_j \mathbf{A}_t(i,j)$	$\max(\mathbf{A}_t\mathbf{1})$
Source fan-out from i	$\sum_j \lvert\mathbf{A}_t(i,j)\rvert_0$	$\lvert\mathbf{A}_t\rvert_0\mathbf{1}$
Max source fan-out ($d_{\max}$)	$\max_i \sum_j \lvert\mathbf{A}_t(i,j)\rvert_0$	$\max(\lvert\mathbf{A}_t\rvert_0\mathbf{1})$
Unique destinations	$\sum_j \lvert\sum_i \mathbf{A}_t(i,j)\rvert_0$	$\lvert\mathbf{1}^\mathsf{T}\mathbf{A}_t\rvert_0\mathbf{1}$
Destination packets to j	$\sum_i \mathbf{A}_t(i,j)$	$\mathbf{1}^\mathsf{T}\lvert\mathbf{A}_t\rvert_0$
Max destination packets ($d_{\max}$)	$\max_j \sum_i \mathbf{A}_t(i,j)$	$\max(\mathbf{1}^\mathsf{T}\lvert\mathbf{A}_t\rvert_0)$
Destination fan-in to j	$\sum_i \lvert\mathbf{A}_t(i,j)\rvert_0$	$\mathbf{1}^\mathsf{T}\mathbf{A}_t$
Max destination fan-in ($d_{\max}$)	$\max_j \sum_i \lvert\mathbf{A}_t(i,j)\rvert_0$	$\max(\mathbf{1}^\mathsf{T}\mathbf{A}_t)$

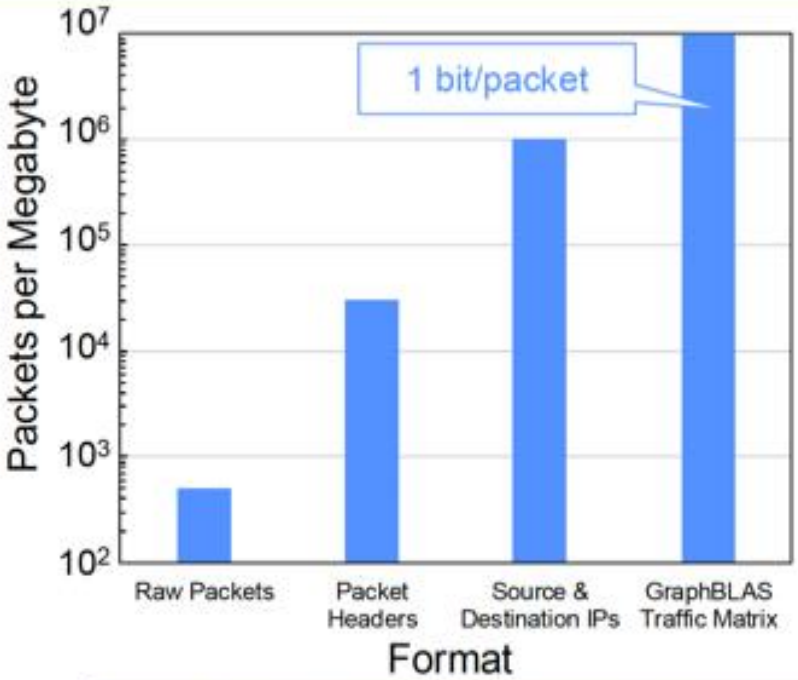
GraphBLAS hypersparse traffic images enable efficient computation of network quantities in C/C++/Python/Julia/Matlab/Octave

Jeremy Kepner et al. (MIT Lincoln Lab)

Performance Breakthroughs

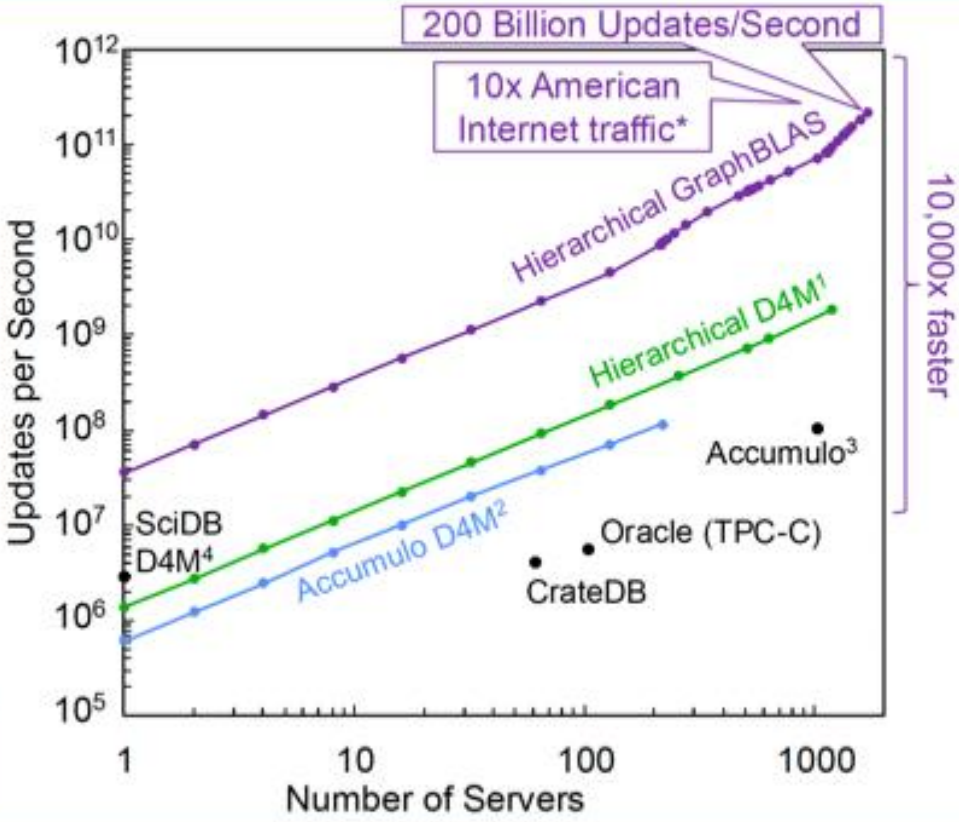
GRAPHBLAS

Compression



GraphBLAS Traffic Matrices
3,000x compression & 10,000x faster

Speed

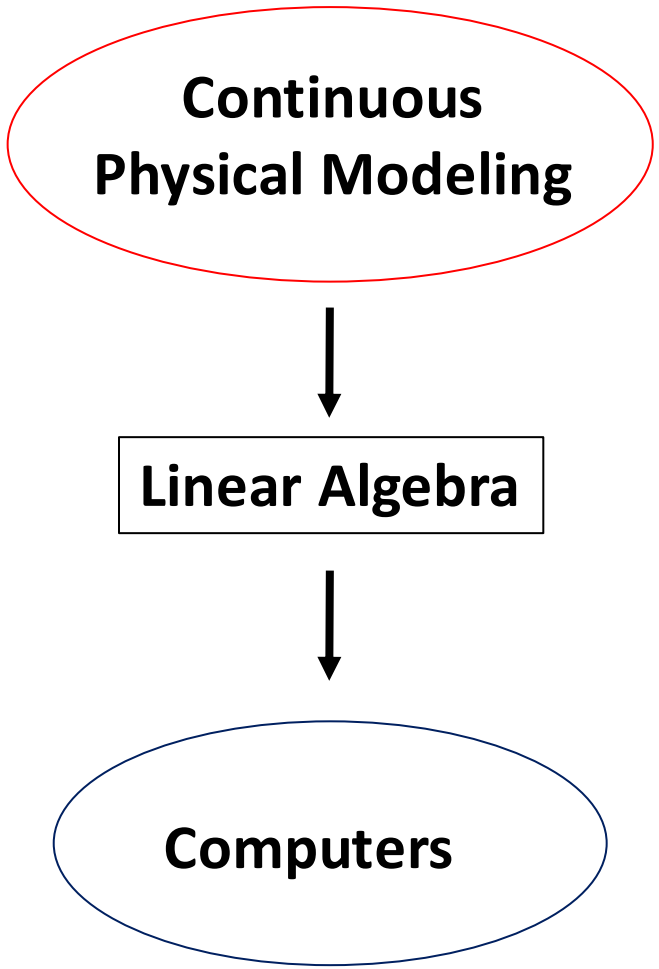


*Cisco Visual Networking Index: Forecast and Trends, 2018–2023; [1]*Vertical, Temporal, and Horizontal Scaling of Hierarchical Hypersparse GraphBLAS Matrices*, Kepner et al, IEEE HPEC 2021; [2]*Streaming 1.9 Billion Hypersparse Network Updates per Second with D4M*, Kepner et al, IEEE HPEC 2019; [3]*Achieving 100,000,000 database inserts per second using Accumulo and D4M*, Kepner et al, IEEE HPEC 2014; [4]*Benchmarking Apache Accumulo BigData Distributed Table Store*, Sen et al, IEEE Big Data Congress 2013; [5]*Benchmarking SciDB Data Import on HPC Systems*, Samsi et al, IEEE HPEC 2016

13.33 x 7.50 in

(Note: Kepner will speak about this at MIT tomorrow 23 April; see me for zoom info.)

Past 60 Years

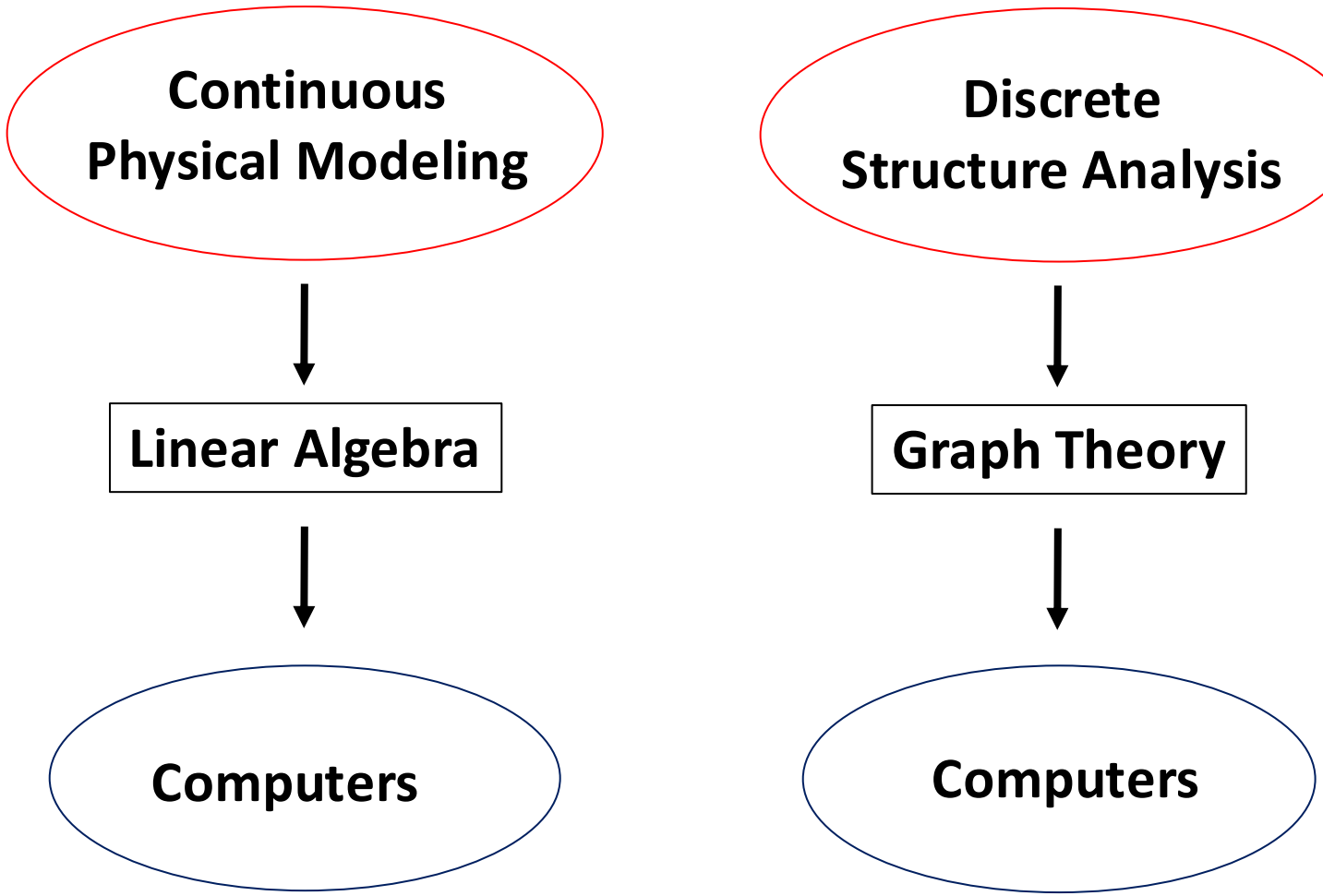


As the "middleware"
of scientific computing,
linear algebra has given us:

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

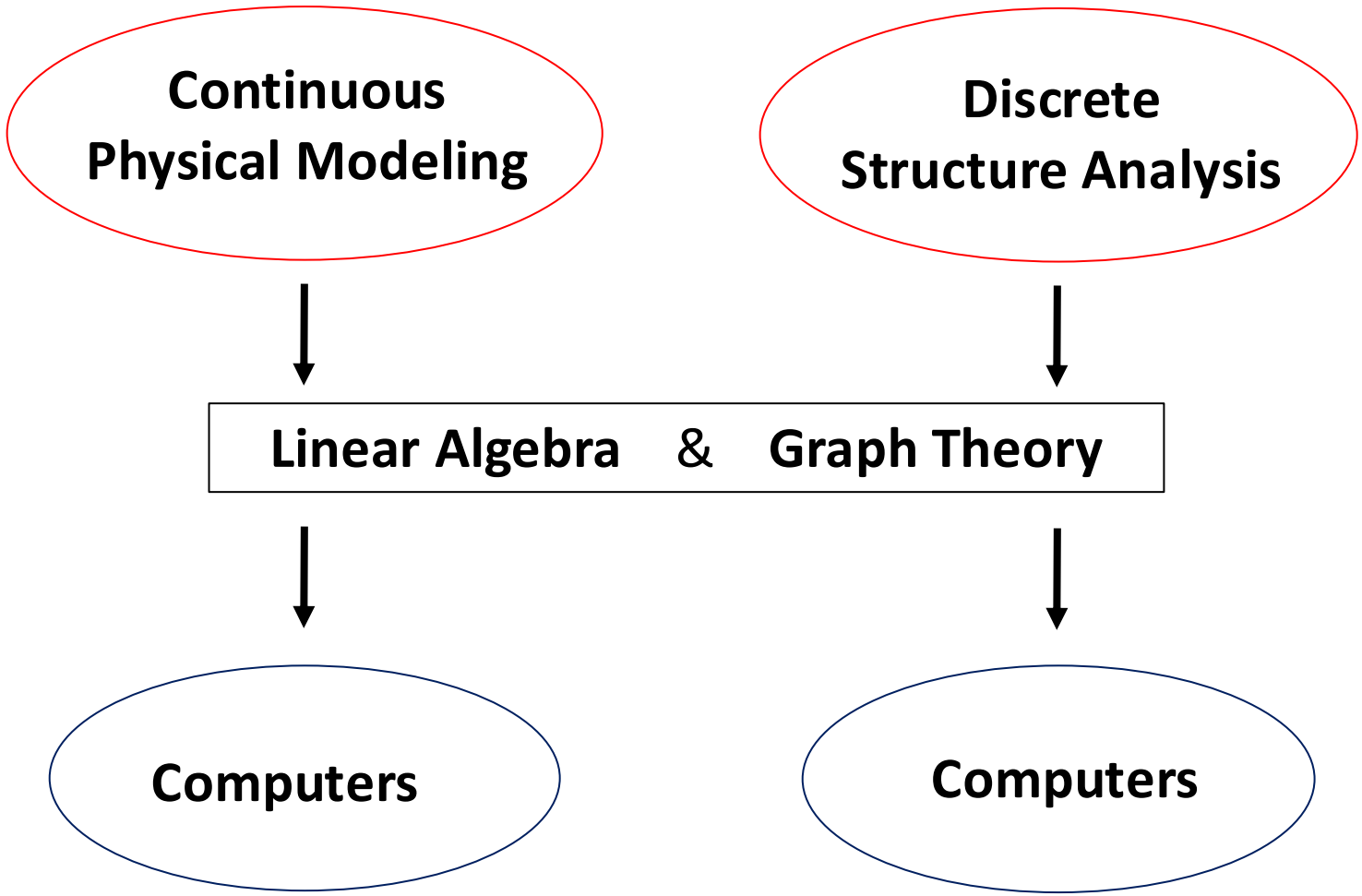
UCSB

Today


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

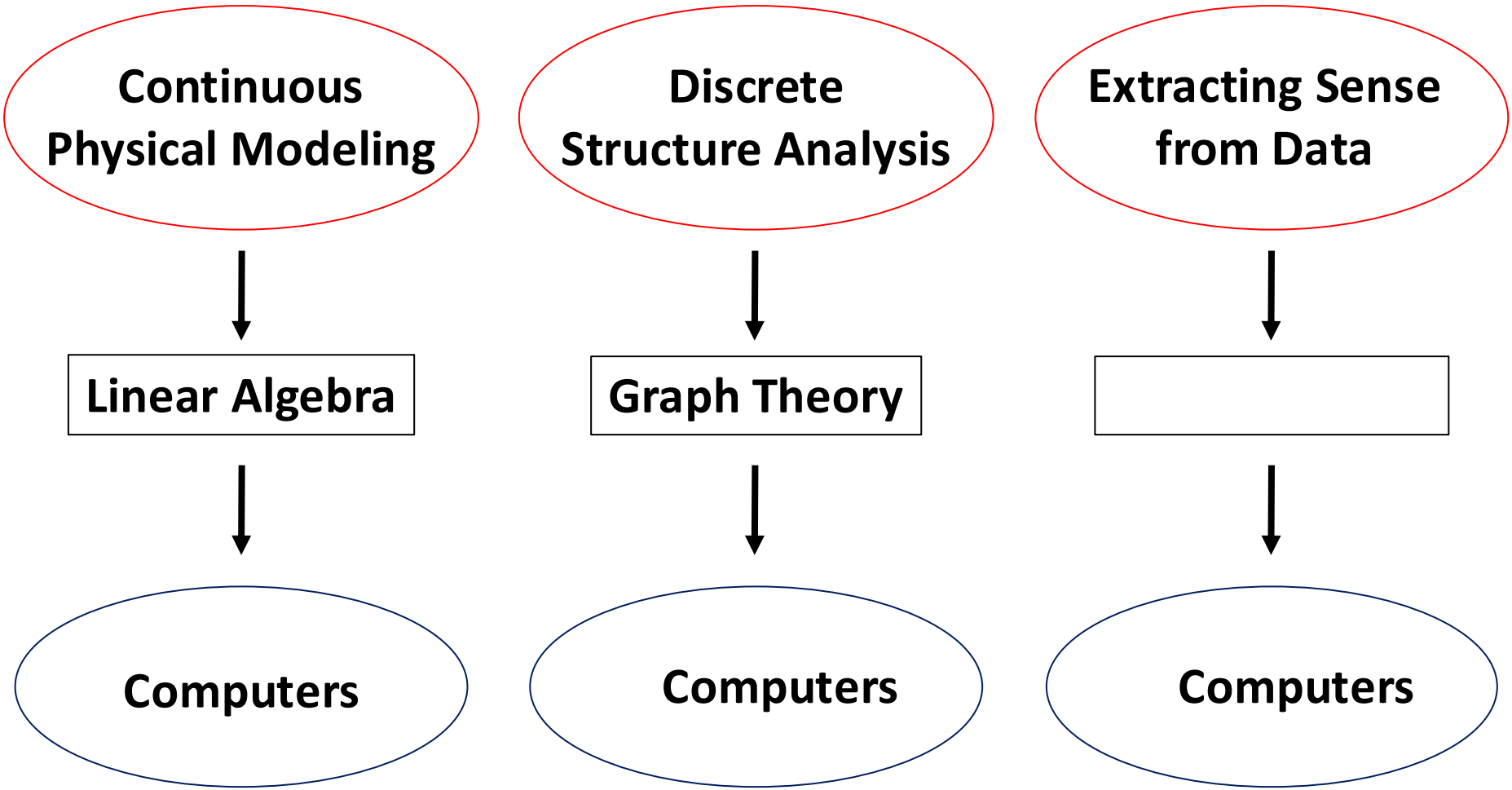
UCSB

Today


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

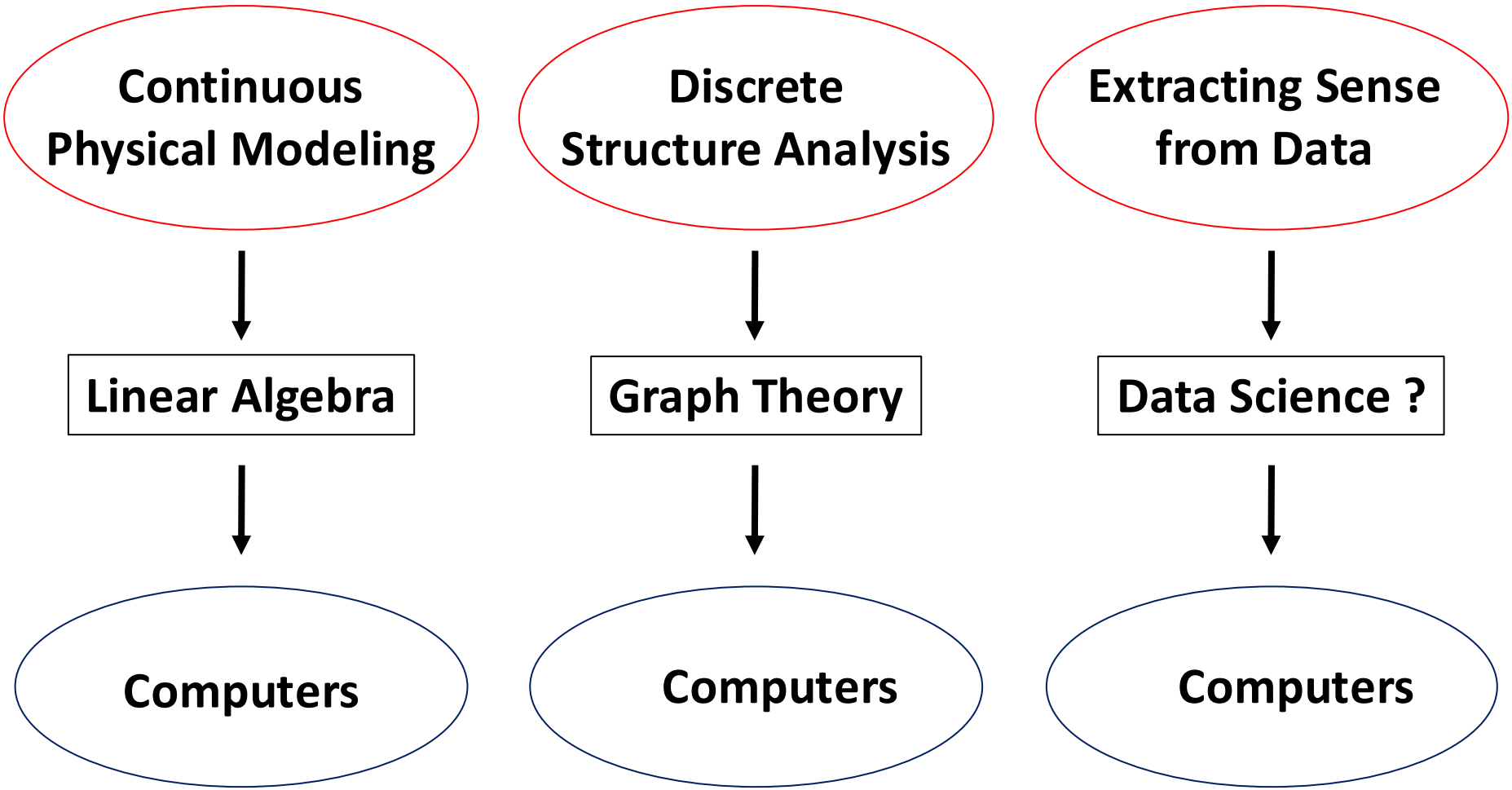
UCSB

Tomorrow



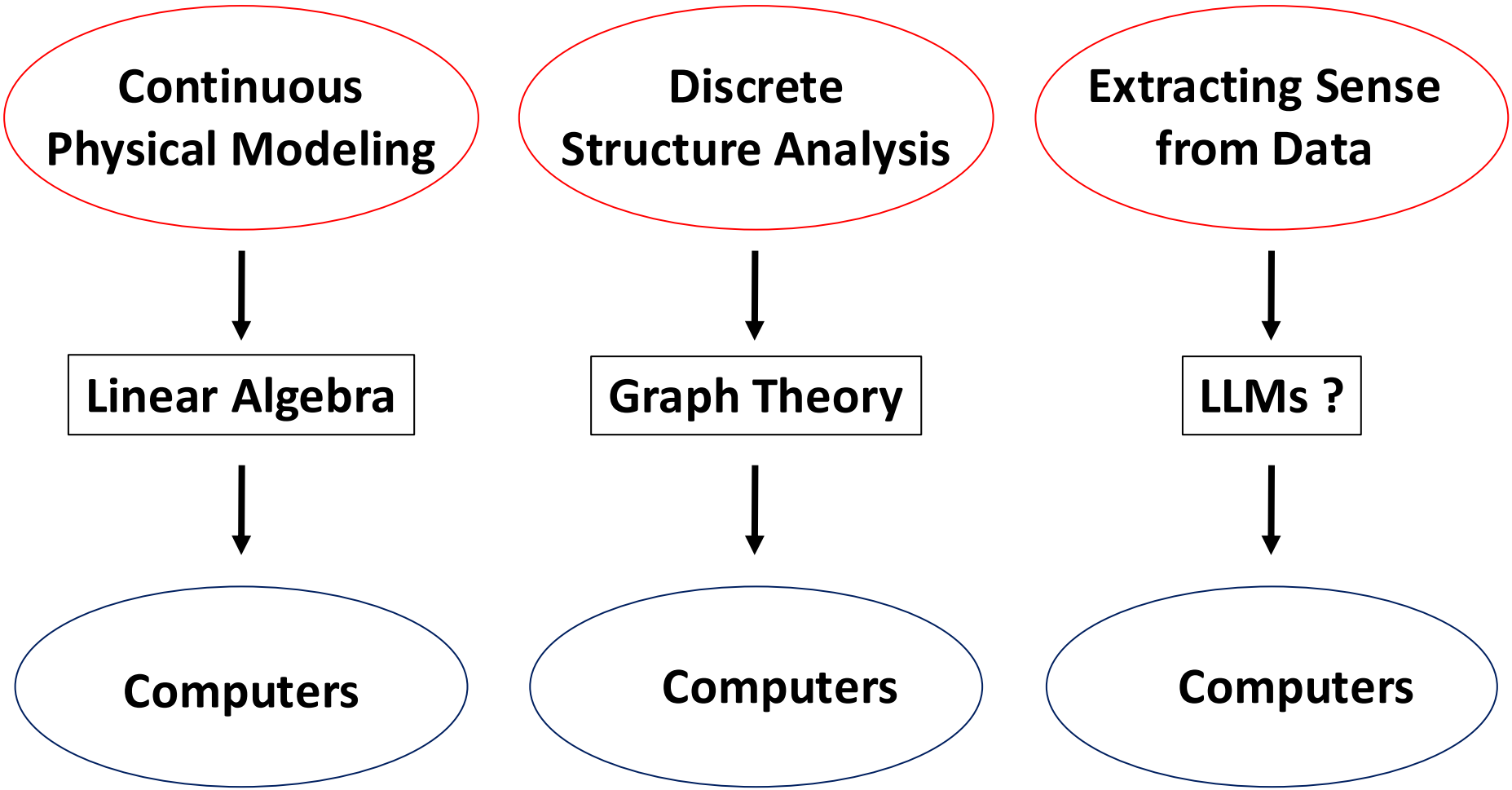
UCSB

Tomorrow



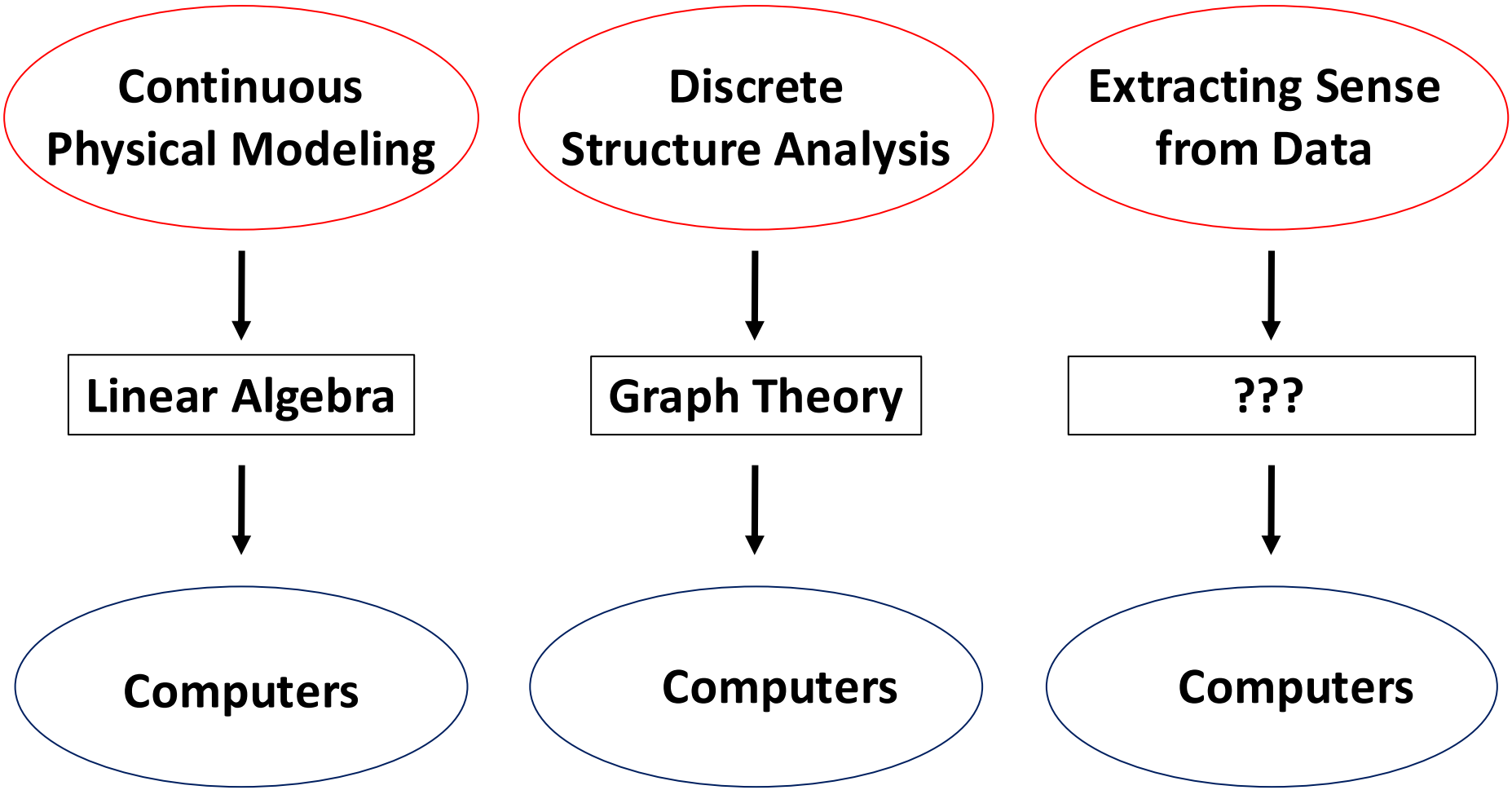
UCSB

Tomorrow



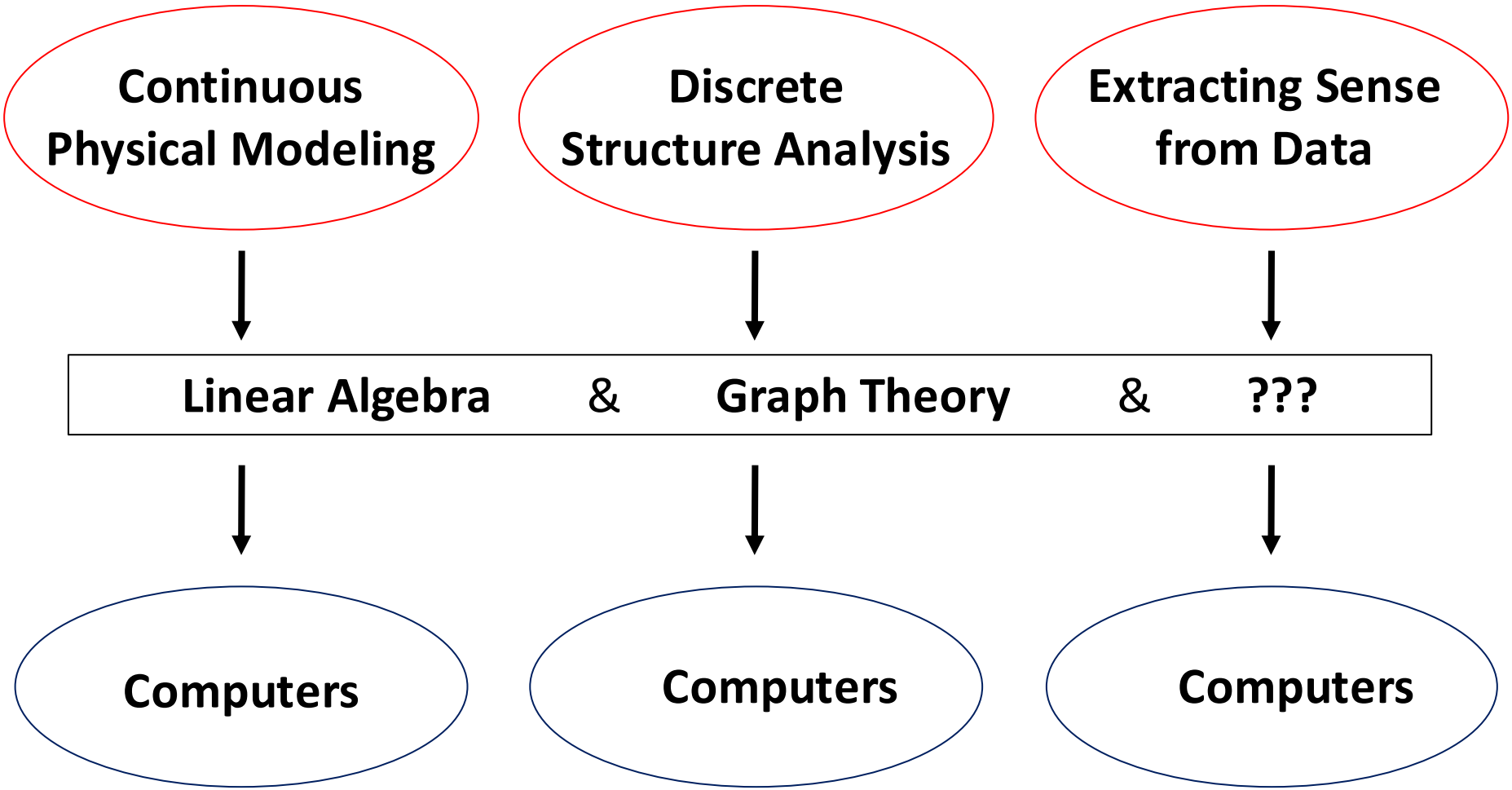
UCSB

Tomorrow



UCSB

Tomorrow



UCSB

Morals

- Matrix computation is beginning to repay a 60-year debt to graph algorithms.
- Discrete algorithms are pervasive in large-scale applied computation, and will become more so.
- Computational resources are in a rare period of flux, with opportunities for influence going both ways.
- It helps to look at things from two directions.