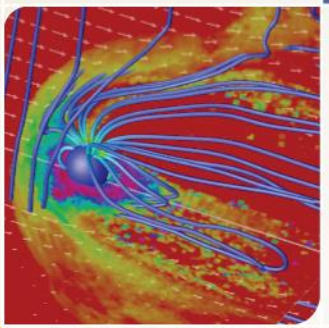
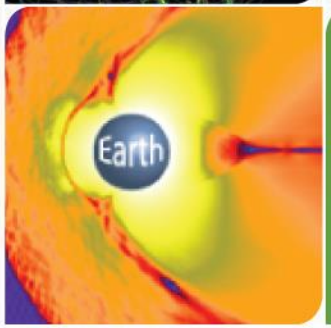
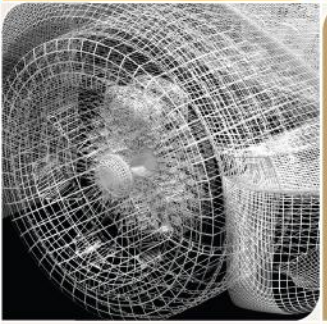




Hornet: An Efficient Data Structure for Dynamic Sparse Graphs and Matrices

Oded Green

Georgia Tech College of Computing
Computational Science and Engineering

Hornet

- A scalable and dynamic data structure for
  - Sparse data
  - Graph algorithms
  - Linear algebra based problems
- Formerly known as cuSTINGER
  - Hornet initialization is hundreds of times faster
  - Hornet updates are 4X-10X faster
  - The Hornet data structure offers is more robust and scalable than cuSTINGER.
- Essentially a dynamic CSR data structure
- Easy to use

“Separation of powers”

- Dynamic graph data structure and dynamic graph algorithms are in two different repositories
  - Easy to integrate with external library
  - Can also be used with matrices

- This talk focuses on the data structure

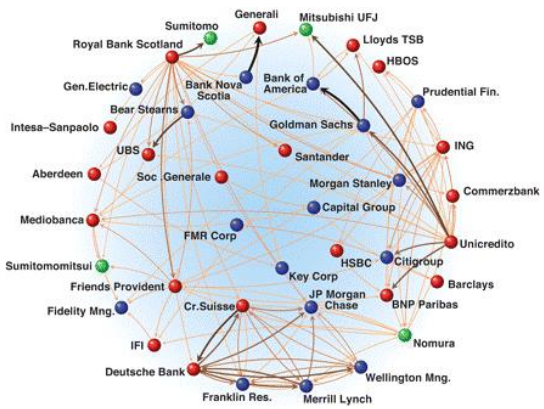
Graph Primitives – Upfront summary

- Great performance for static and dynamic graph algorithms
- Scalable
- Simple to use
- Will discuss algorithm framework later today
  - 1:00pm
  - Same room as this talk

Hornet – Upfront Summary

- Can support over 150 million updates per second
- Can easily scale to graphs with billions of vertices
- CSR comparison
  - Initializing is also relatively in-expensive – usually less than 3X slower
  - Hornet requires 30% more storage
  - Identical performance
- COO (edge-list) comparison
  - Hornet requires 20% less storage
  - Hornet has better locality

Big Data problems need Graph Analysis



Financial networks:

- Transactions between players.
- Different transactions types (property graph)

Communication networks:

- World-wide connectivity
- High velocity changes
- Different types of extracted data:
  - Physical communication network.
  - Person-to-person communication network.

Health-Care networks:

- Various players.
- Pattern matching and epidemic monitoring.
- Problem sizes have doubled in last 5 years.

Hornet Properties

- ✓ A Simple programming model
  - ✓ Enable algorithm designers to implement dynamic & streaming graph algorithms with ease.
- ✓ Can easily grows 1000X initial size (no restart needed)
- ✓ Millions of updates per second to graph
  - ✓ Updates are not bottlenecks for analytics.
- ✓ Automated data management
  - ✓ Transfers data between host and device automatically
  - ✓ Reduces fragmentation
  - ✓ Supports memory reclamation
- Scalable data structure

cuSTINGER paper: [Green&Bader; HPEC, 2016]:

cuSTINGER: Supporting dynamic graph algorithms for GPUs

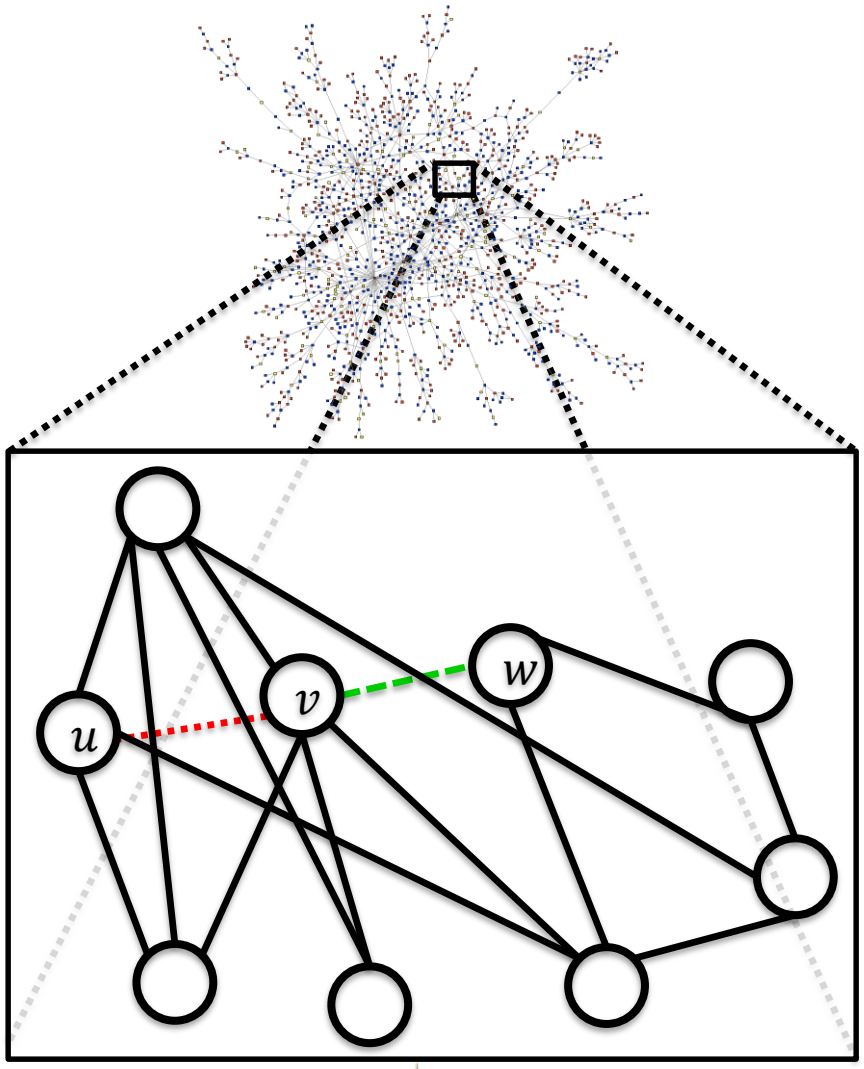
Definitions

- Dynamic graphs
  - Graph can change over time.
  - Changes can be to topology, edges, or vertices.
    - For example new edges between two vertices.
  - Changes to edge or vertex weights
- Streaming graphs:
  - Graphs changing at high rates.
  - 100s of thousands of updates per second.

- Dynamic matrices
  - Adding a perturbation to the matrix

Dynamic graph example

- Only a subset of the entire graph…
- Dynamic:
  - At time t:
    - v and w become friends.
    - $insert\_edge\ (v, w)$
  - At time $\hat{t}$:
    - $u\ and\ v\ no\ longer\ friends$
    - $\mathrm{delete}_{edge(u,v)}$
- Additional operations include vertex insertions & deletions



Widely used graph data structures

Names	Pros	Cons
Dense Adjacency Matrix	• Supports updates	• Poor locality • Massive storage requirements
Linked lists	• Flexible	• Poor locality • Limited parallelism • Allocation time is costly
COO (Edge list) - unsorted	• Has some flexibility • Updates are simple • Lots of parallelism	• Poor locality • Stores both the source and destination
CSR	• Uses exact amount of memory • Good locality • Lots of parallelism	• Inflexible

These data structures don't cut it

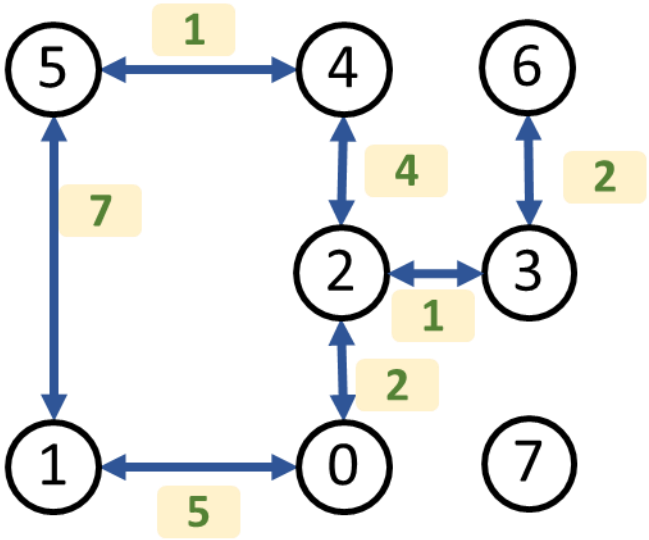
Compressed Sparse Row (CSR)

Pros:

- Uses precise storage requirements
- Great locality
  - Good for GPUs
- Handful of arrays
  - Simple to use and manage

Cons:

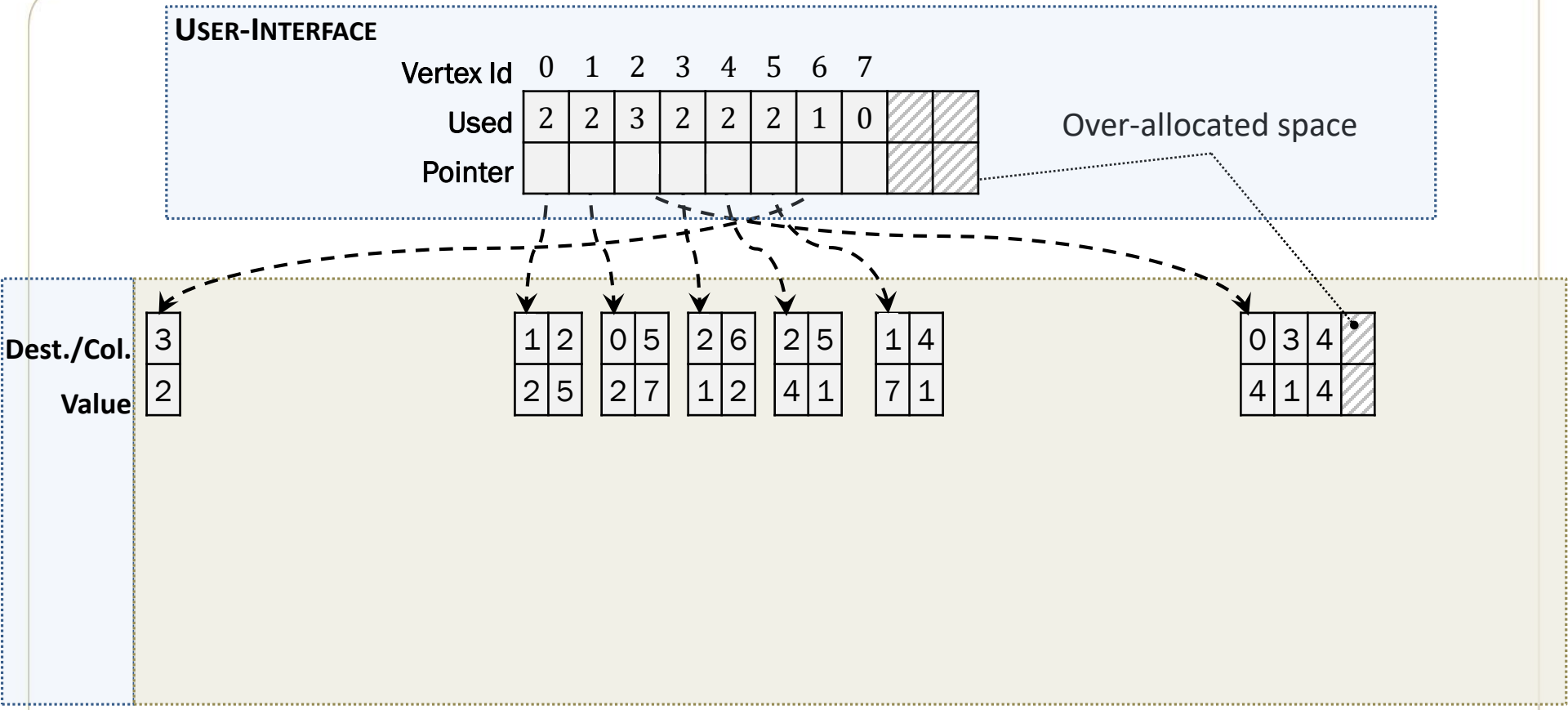
- Inflexible.
- Network growth unsupported
- Topology changes unsupported
- Property graphs not supported



Src/Row	0	1	2	3	4	5	6	7	
Offset	0	2	4	7	9	11	13	14	14

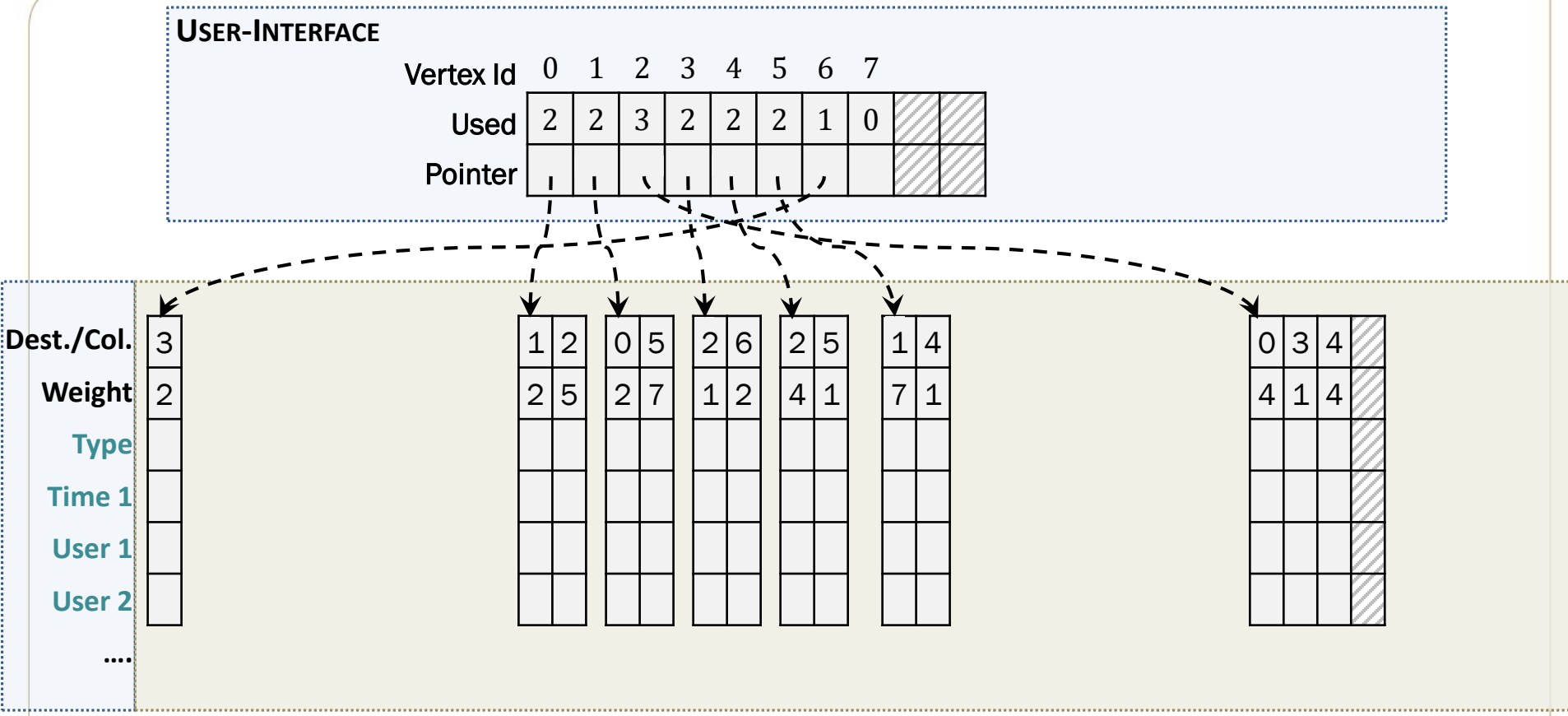
Dest./Col.	1	2	0	5	0	3	4	2	6	2	5	1	4	3
Value	2	5	2	7	4	1	4	1	2	4	1	7	1	2

Hornet – A High Level View



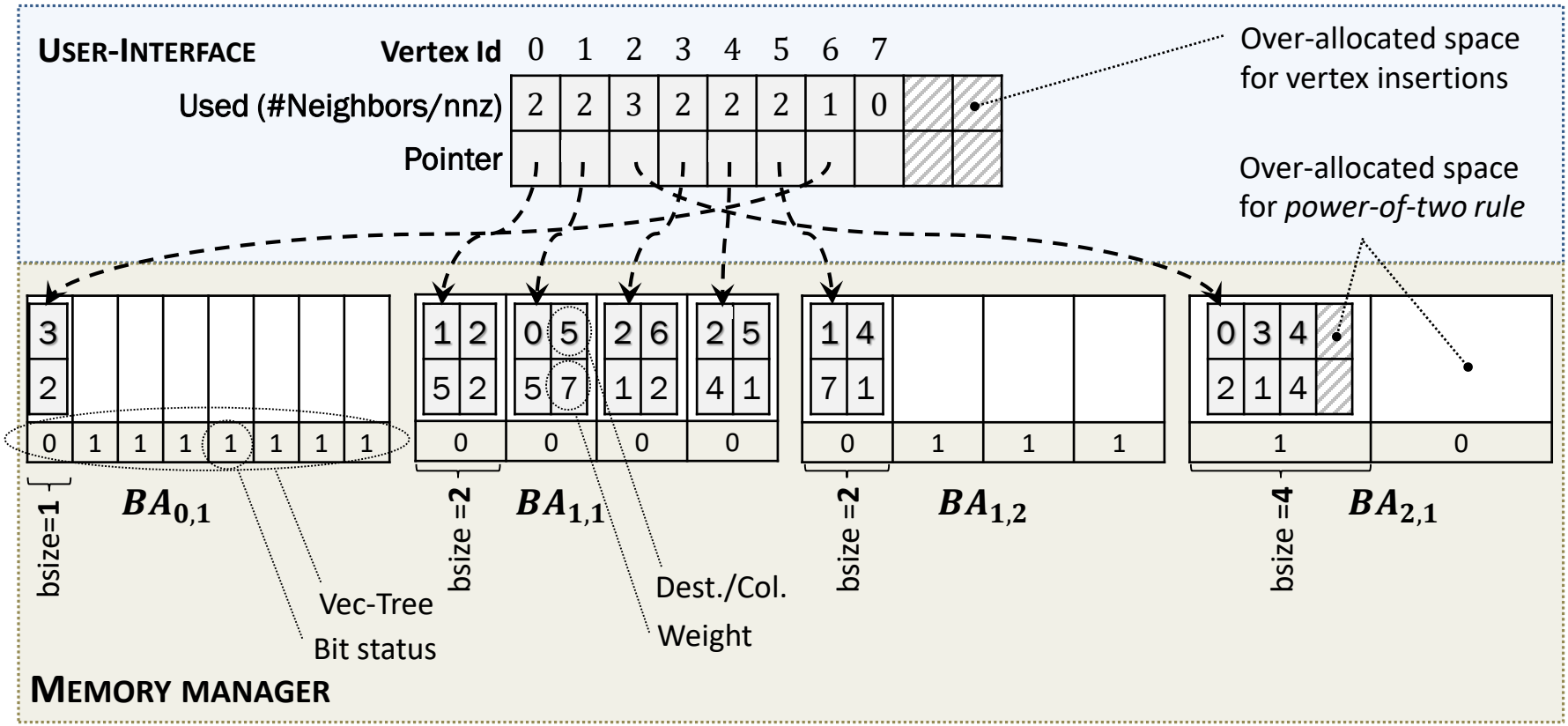
- Every vertex points at its own array
- Many edges array (blocks)
- Block size is determined by the number of neighbors (always powers of 2)
- Extra space left at the end of the block

Hornet – Property Graph Support



- Programmers can add fields per edge
- Easy to mange for static graph data structures
- Hornet manages the data movement

Hornet in Detail



Hornet Performance

- Memory Utilization
  - Independent of the GPU being used
- Initialization overhead
- Update rate

Hornet Performance Analysis

- All performance analysis is for the P100
  - 56 SMs
  - 3584 SPs
  - 16GB HBM2 memory

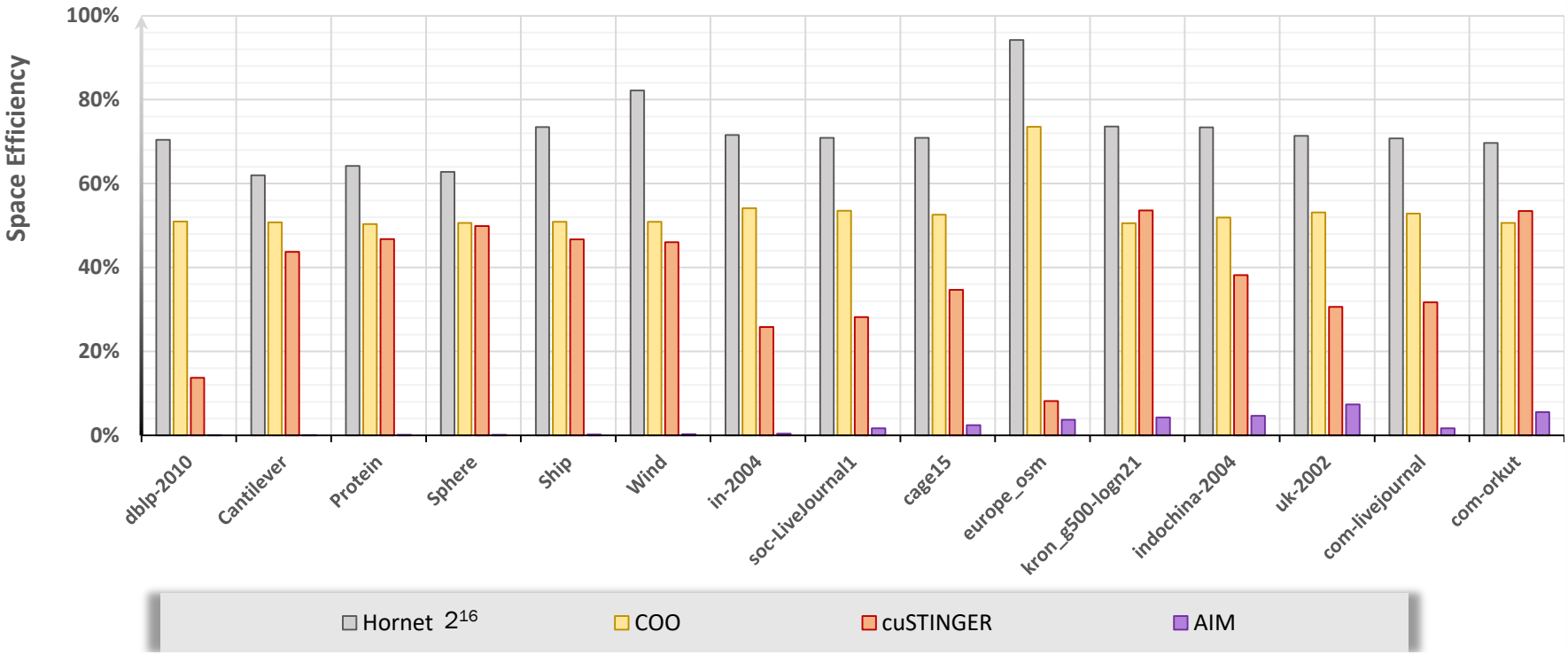
Inputs Graphs

- DIMACS 10 Graph Implementation Challenge
- SNAP – Stanford Network Analysis Project
- Florida Matrix Collection

The following is only a subset of these graphs:

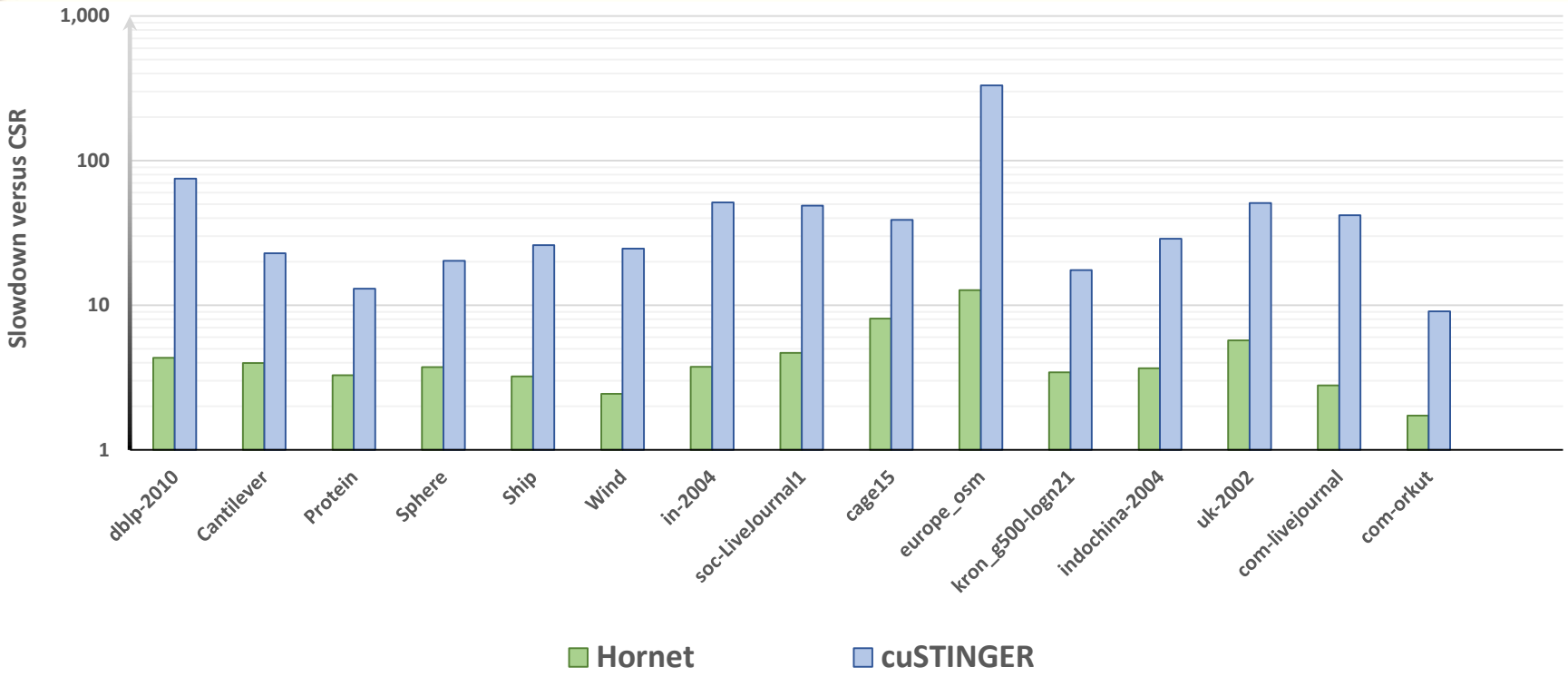
Name	Type	$\|V\|$	$\|E\|$*	Source
$coAuthorsDBLP$	Collaboration	$299k$	$1.95M$	DIMACS
$as-skitter$	Trace route	$1.69M$	$11.1M$	SNAP
$kron\_21$	Random	$2M$	$201M$	DIMACS
$cit-patents$	Citation	$3.77M$	$16.5M$	SNAP
$cage15$	Matrix	$5.15M$	$94M$	DIMACS
$uk-2002$	Webcrawl	$18.52M$	$523M$	DIMACS

Memory Utilization - Overall



- BlockArrays of size 2^{16}
- 70% average utilization of CSR
- Better utilization then: COO, cuSTINGER, AIM
  - AIM allocates all GPU memory

Initialization overhead

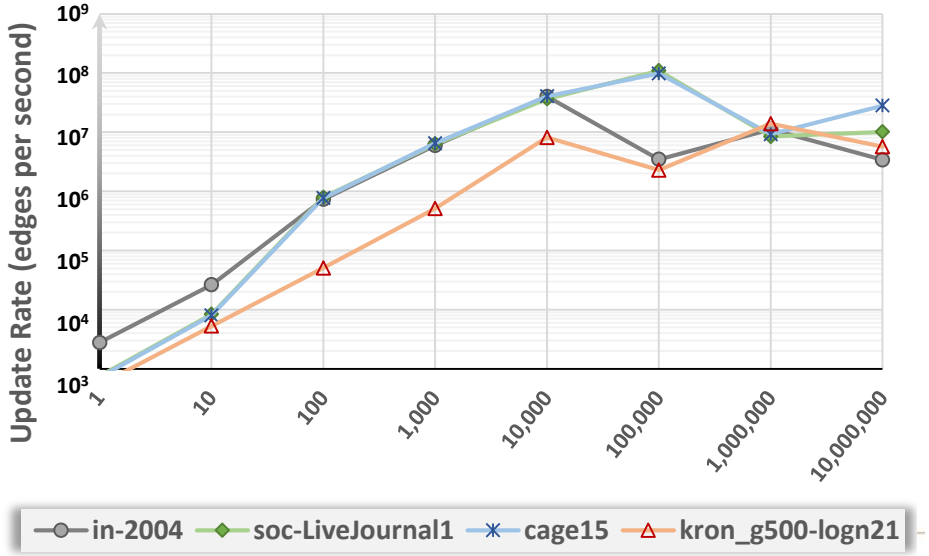


- Time to initialize data structure in comparison to CSR
- In most cases 2X-3X slower
  - One time penalty
- Much faster than cuSTINGER

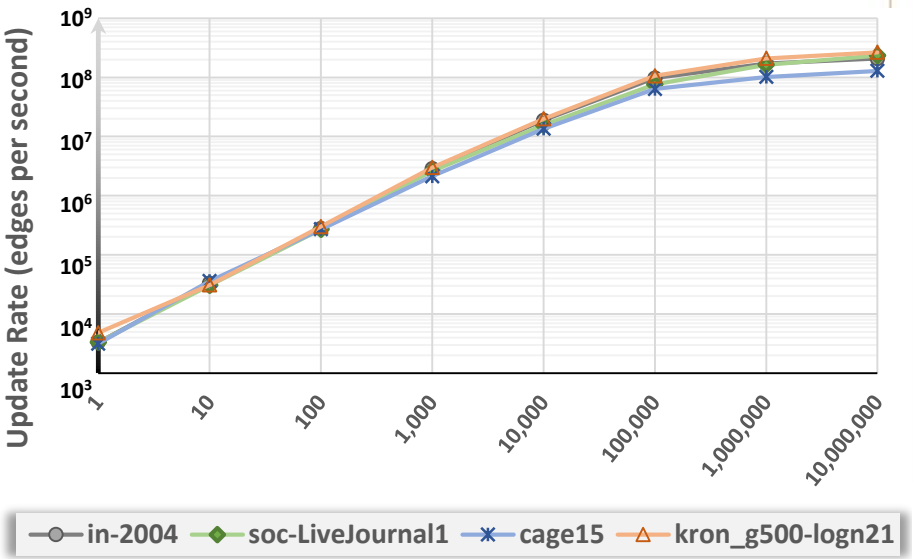
Insertion Rates

- Supports over 150M updates per second
- Hornet
  - $4X - 10X$ faster than cuSTINGER
  - Does not have $performance\ dip$ like cuSTINGER
- Scalable growth in update rate

cuSTINGER



Hornet



Take away

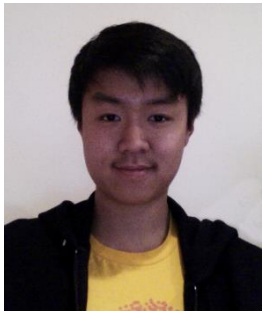
- Anything you can do with CSR you can also do with Hornet (other way is not true)
- Supports high update rates

- Scalable in both data size and in performance
- Simple and high-level programming model
  - See you at 1:00pm
- Also, look for James Fox's talk on a cool algorithm for finding the maximal K-Truss in a graph
  - Uses dynamic triangle counting and the Hornet's deletion…

Hornet Team (Current & Alumni)

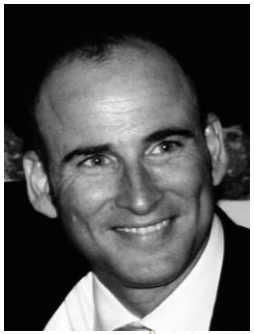








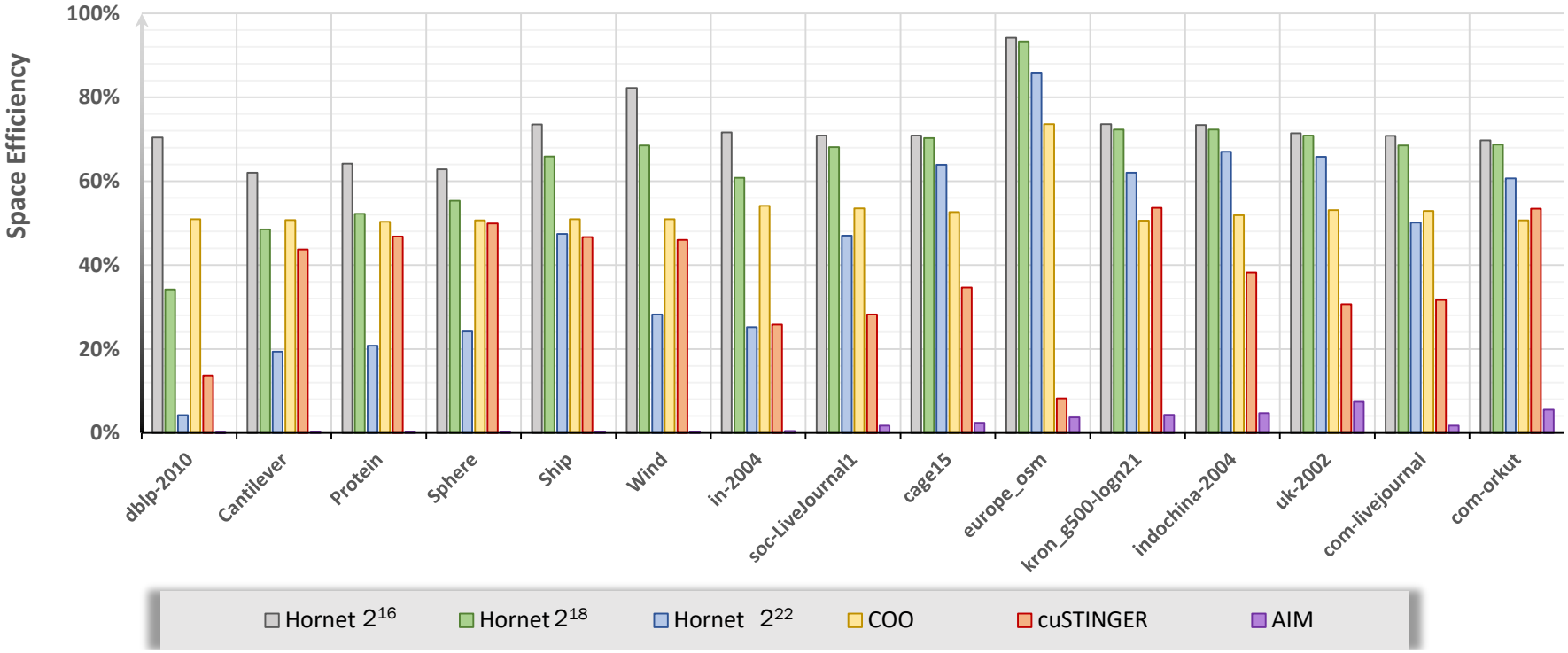
Thank you

- Email: ogreen@gatech.edu

- Hornet:
  - https://github.com/hornet-gt/hornet
- HornetsNest:
  - https://github.com/hornet-gt/hornetsnest

Backup slides

Memory Utilization - Overall



- 70% average utilization of CSR
- Better utilization in comparison to: COO, cuSTINGER, AIMS

Part 2: HornetsNest

- Algorithm framework for Hornet data structure
  - We support CSR as well
- All algorithms are implemented using a small set of operations
  - We show that these operators are efficient for static graph algorithms and can be used for dynamic graph algorithms
- Uses features from C++11 and C++14

Sparse Matrix Vector Multiplication

- In comparison to DCSR [King et al; 2016; ISC]
  - DCSR requires customized SpMV
- Hornet uses identical algorithm code as CSR.