Graph Analytics in GraphBLAS

Vijay Gadepally, Jake Bolewski, Dan Hook, Dylan Hutchison, Ben Miller, Jeremy Kepner

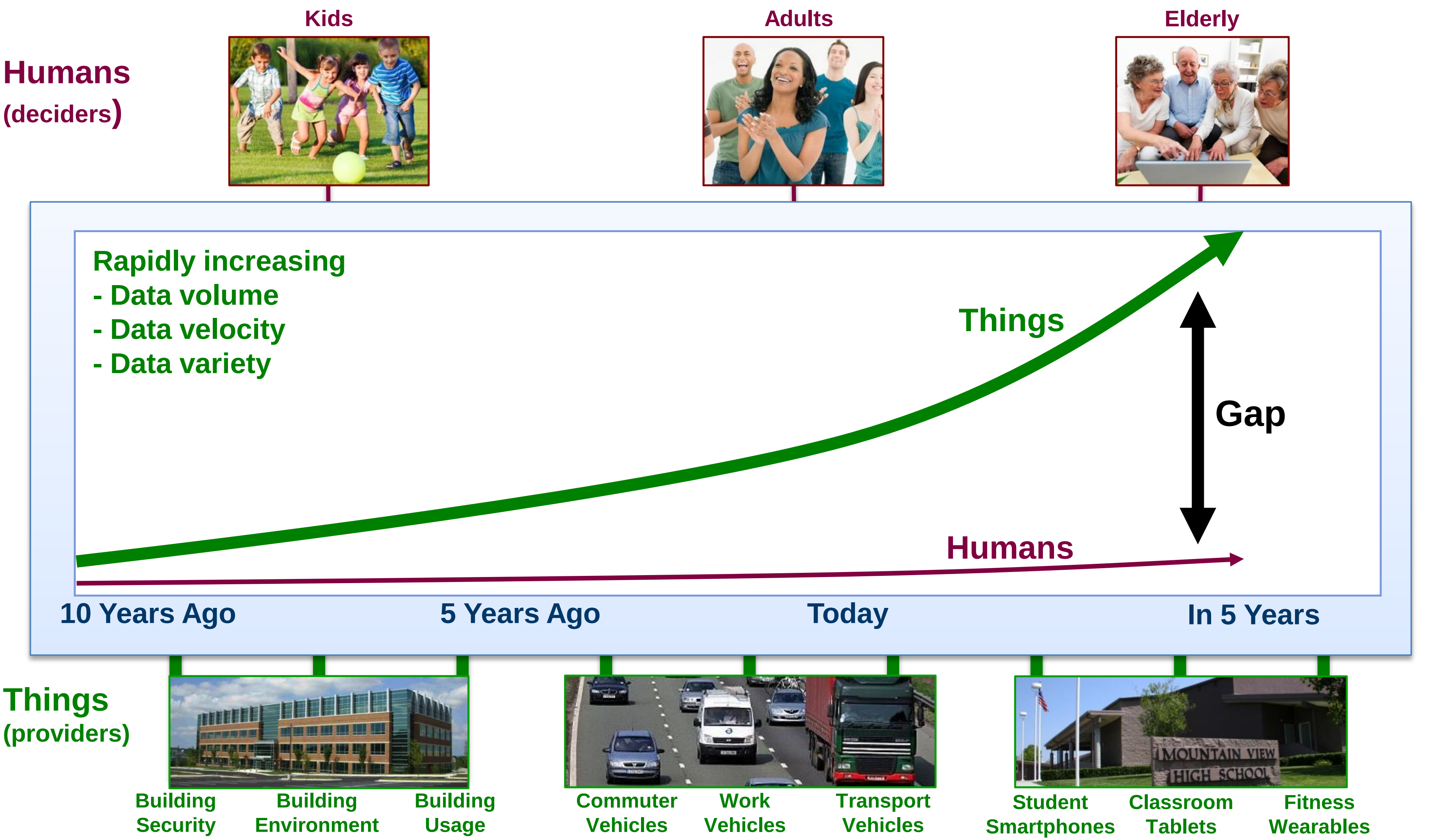
May 2015

This material is based upon work supported by the National Science Foundation under Grant No. DMS-1312831. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation.

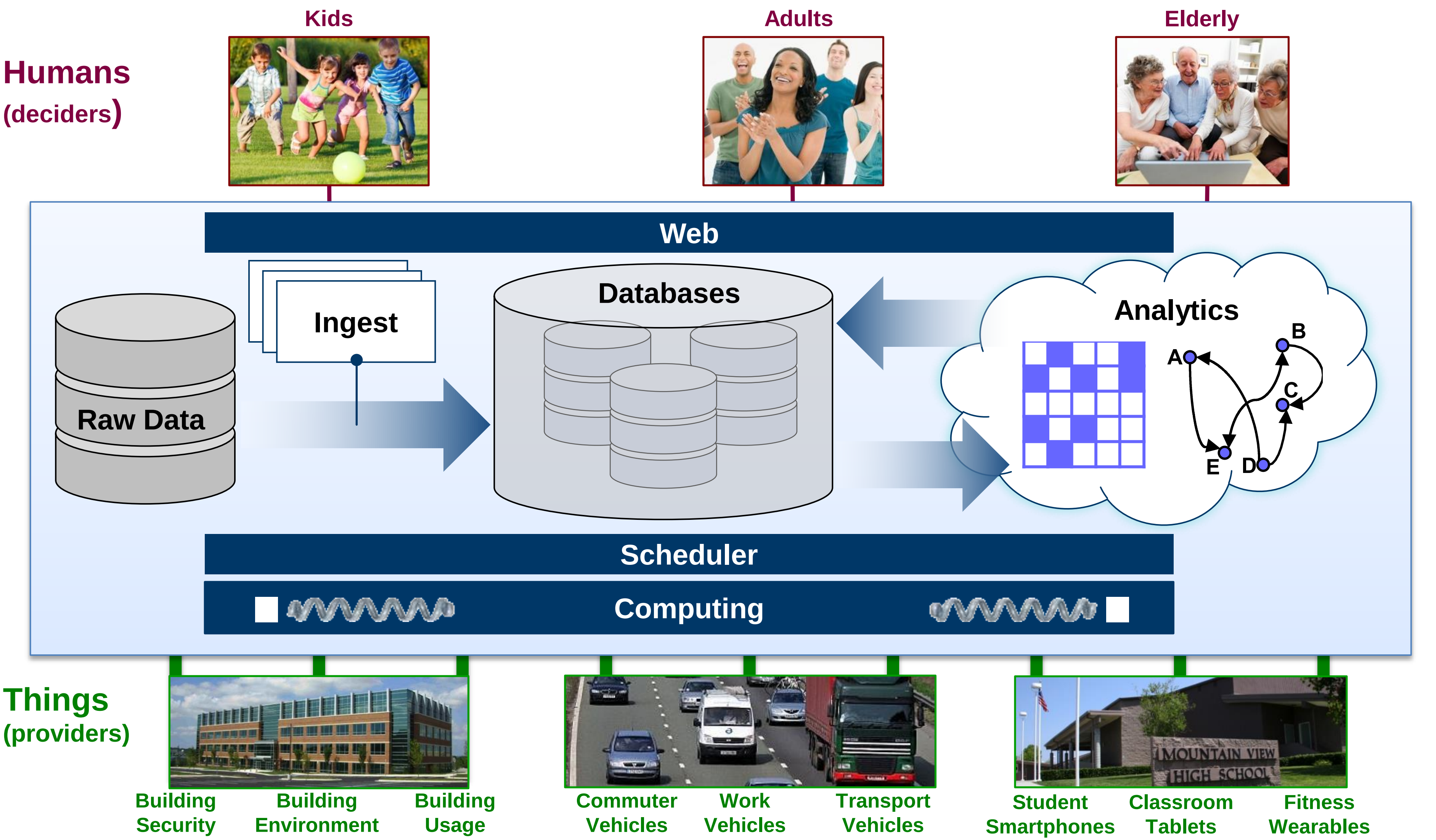
Outline

- Introduction
- Degree Filtered Breadth First Search
- K-Truss
- Jaccard Coefficient
- Non-Negative Matrix Factorization
- Summary

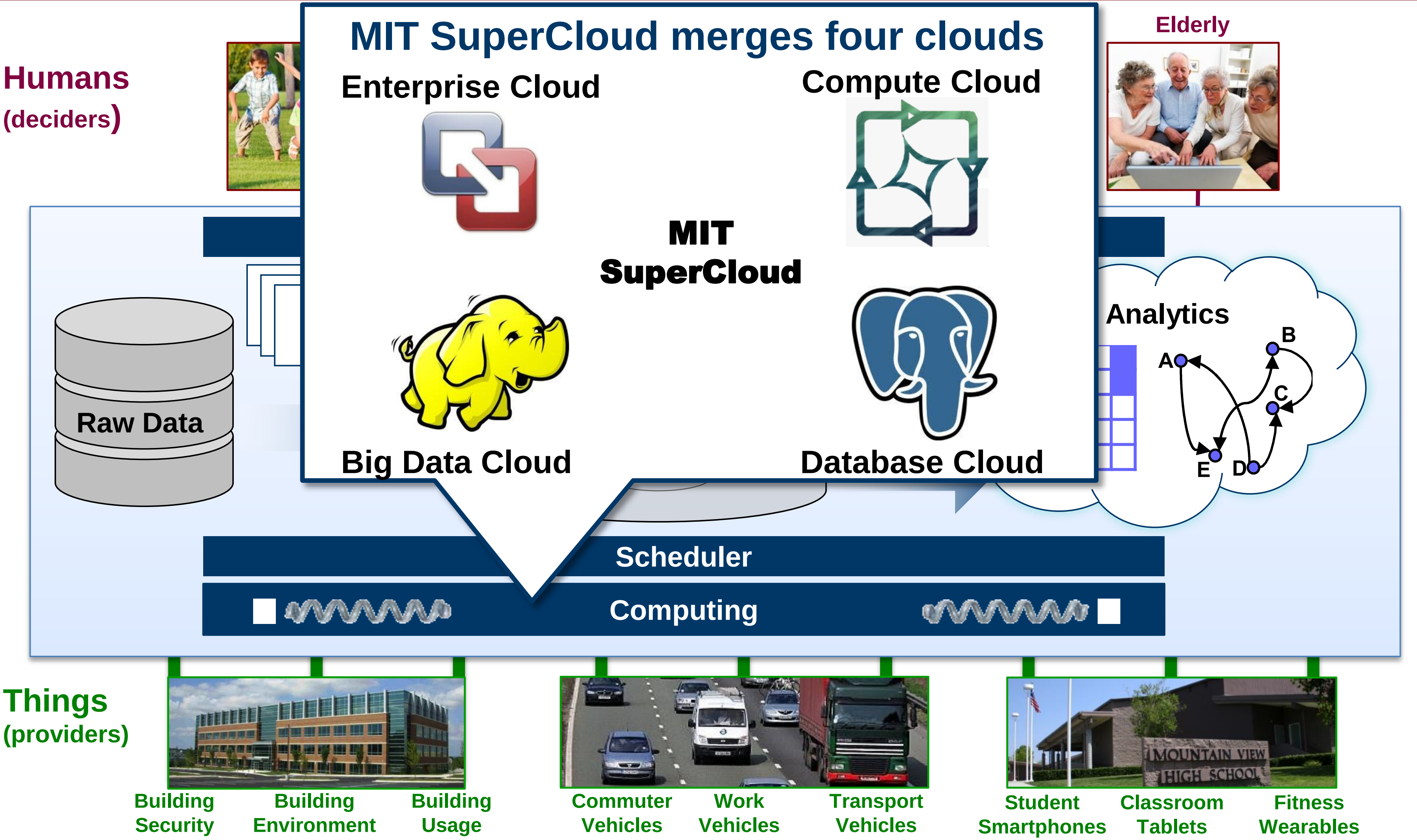
Big Data Challenge



Systems Architecture

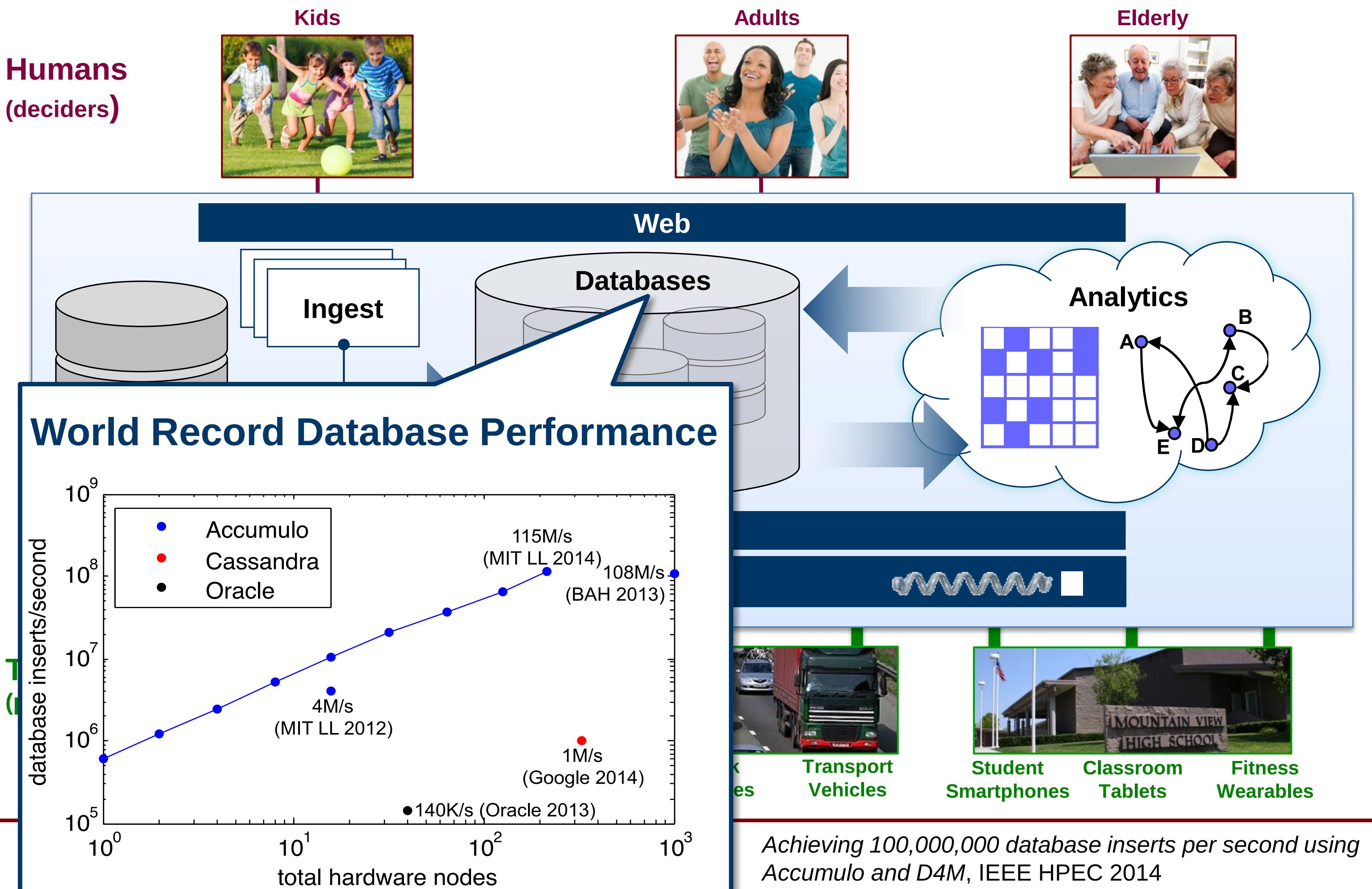


Systems Architecture
Data Volume ⇒ Many Clouds



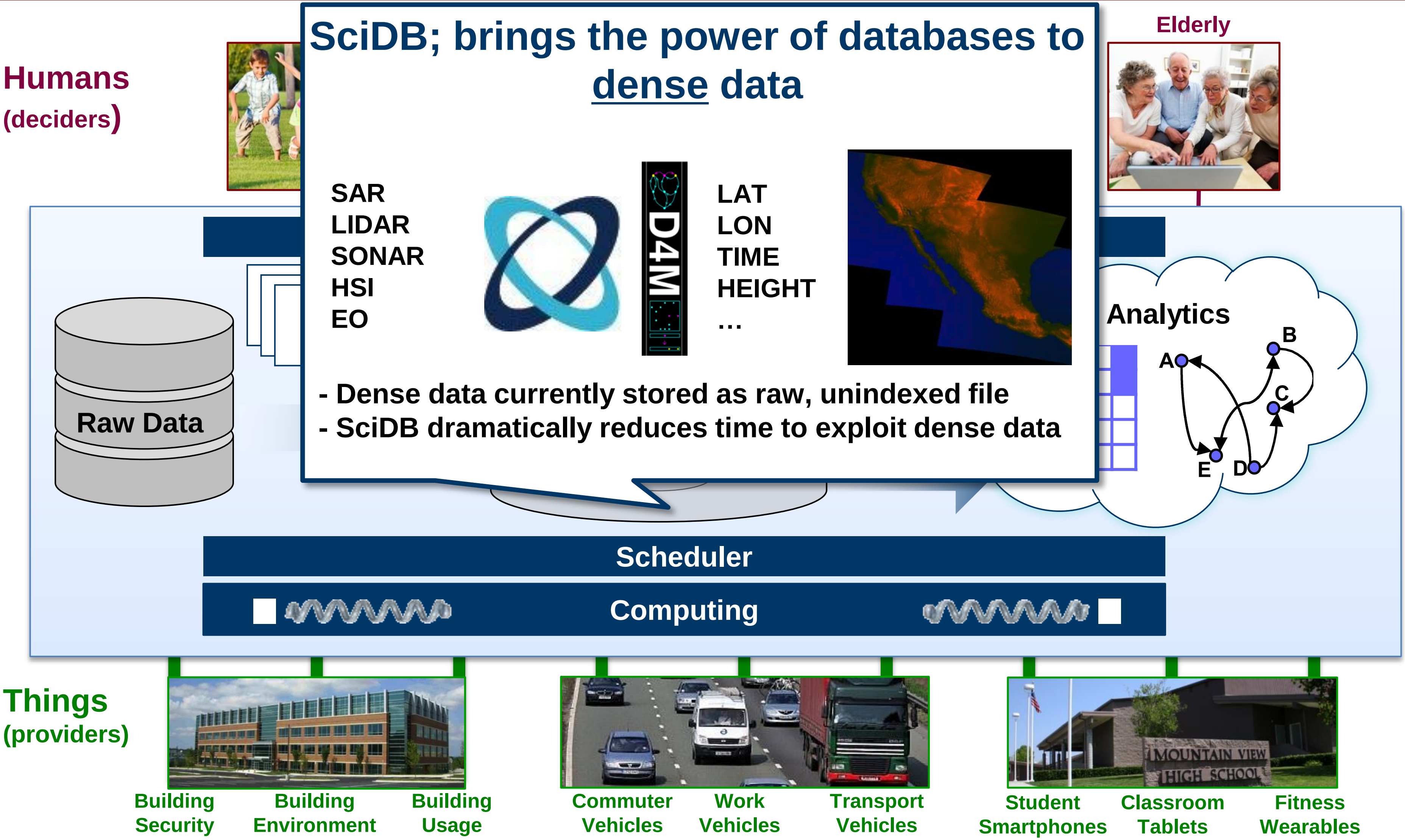
LLSuperCloud: Sharing HPC Systems for Diverse Rapid Prototyping, IEEE HPEC 2013

Systems Architecture
Data Velocity ⇒ Databases

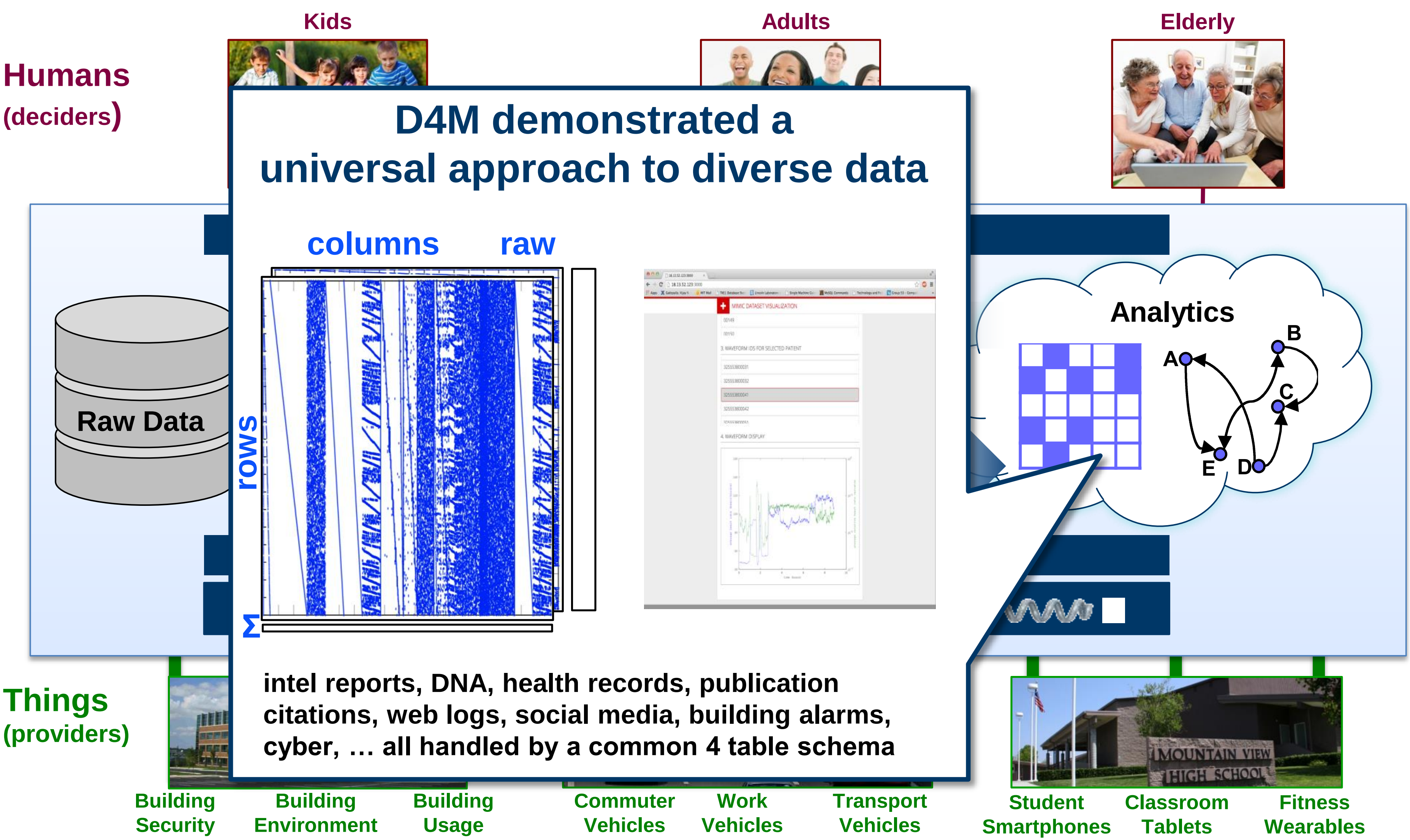


Achieving 100,000,000 database inserts per second using Accumulo and D4M, IEEE HPEC 2014

Systems Architecture
Diversity ⇒ Databases



Systems Architecture
Variety ⇒ D4M Schema



Graphulo Goals

- **Primary Goal**
  - Open source Apache Accumulo Java library that enables many graph algorithms in Accumulo using Accumulo server-side constructs
- **Additional Goals**
  - Enable a wide range of graph algorithms with a small number of functions on a range of graph schemas
  - Efficient and predictable performance; minimize maximum run time
  - Instructive and useful example programs; well written spec
  - Minimal external dependencies
  - Fully documented at graphulo.mit.edu
  - Drive Accumulo features (e.g., temporary tables, split API, user defined functions, …)
  - Focus on localized analytics within a neighborhood, as opposed to whole table analytics

Plan

- **Phase 1: Graph Mathematics Specification**
  – **Define library mathematics**
  – **Define example applications and data sets**

- **Phase 2: Graph Mathematics Prototype**
  – **Implement example applications in Accumulo prototyping environment**
  – **Verify that example applications can be effectively implemented**

- **Phase 3: Java Implementation**
  – **Implement in Java and test at scale**

GraphBLAS

- **The GraphBLAS is an effort to define standard building blocks for graph algorithms in the language of linear algebra**
  - **More information about the group: http://istc-bigdata.org/GraphBlas/**
- **Background material in book by J. Kepner and J. Gilbert: Graph Algorithms in the Language of Linear Algebra. SIAM, 2011**
- **Draft GraphBLAS functions:**
  - **SpGEMM, SpM{Sp}V, SpEWiseX, Reduce, SpRef, SpAsgn, Scale, Apply**
- **Goal: show that these functions can perform the types of analytics that are often applied to data represented in graphs**

GraphBLAS is a natural starting point Graphulo Mathematics

Examples of Graph Problems

Algorithm Class	Description	Algorithm Examples
Exploration & Traversal	Algorithms to traverse or search vertices	Depth First Search, Breadth First Search
Centrality & Vertex Nomination	Finding important vertices or components within a graph	Betweenness Centrality, K-Truss sub graph detection
Similarity	Finding parts of a graph which are similar in terms of vertices or edges	Graph Isomorphism, Jaccard Index, Neighbor matching
Community Detection	Look for communities (areas of high connectedness or similarity) within a graph	Topic Modeling, Non-negative matrix factorization, Principle Component Analysis
Prediction	Predicting new or missing edges	Link Prediction
Shortest Path	Finding the shorted distance between two vertices	Floyd Warshall, Bellman Ford, A* algorithm, Johnson's algorithm

Accumulo Graph Schema Variants

- Adjacency Matrix (directed/undirected/weighted graphs)
  - row = start vertex; column = vertex; value = edge weight
- Incidence Matrix (multi-hyper-graphs)
  - row = edge; column = vertices associated with edge; value = weight
- D4M Schema
  - Standard: main table, transpose table, column degree table, row degree table, raw data table
  - Multi-Family: use 1 table with multiple column families
  - Many-Table: use different tables for different classes of data
- Single-Table
  - use concatenated v1|v2 as a row key, and isolated v1 or v2 row key implies a degree

Graphulo should work with as many of Accumulo graph schemas as is possible

Algorithms of Interest

- **Degree Filtered Breadth First Search**
  - **Very common graph analytic**
- **K-Truss**
  - **Finds the clique-*iness* of a graph**
- **Jaccard Coefficient**
  - **Finds areas of similarity in a graph**
- **Topic Modeling through Non-negative matrix factorization**
  - **Provides a quick topic model of a graph**

Outline

- Introduction
- Degree Filtered Breadth First Search
- K-Truss
- Jaccard Coefficient
- Non-Negative Matrix Factorization
- Summary

Degree Filtered Breadth First Search

- Used for searching in a graph starting from a root node
  - Very often, popular nodes can significantly slow down the search process and may not lead to results of interest
- A degree filtered breadth first search, first filters out high degree nodes and then performs a BFS on the remaining graph
- A graph G=(V,E) can be represented by an adjacency matrix A where A(i,j)=1 if there is an edge between v_i and v_j
- Alternately, one can represent a graph G using an incidence matrix representation E where rows are edges, columns are nodes, and E(i,j) = 1 if e_i goes into v_j and E(i,j) = -1 if e_i leaves v_j
- The Degree Filtered BFS can be computed using either representation

Adjacency Matrix based Degree Filtered BFS

- **Uses the adjacency matrix representation of a graph G to perform the BFS.**
- **Algorithm Inputs:**

  v_0: Starting vertex set

  k: number of hops to go

  T: Accumulo table of graph adjacency matrix

  T_{in} = sum(T,1).'; % Accumulo table in-degree

  T_{out} = sum(T,2); % Accumulo table out-degree

  d_{min}: minimum allowable degree

  d_{max}: maximum allowable degree

- **Algorithm Output:**

  A_k: adjacency matrix of sub-graph

Adjacency Matrix based Degree Filtered BFS

- **The algorithm begins by retaining vertices whose degree are between d_{min} and d_{max}**

- **Algorithm:**

```
v_k = v_0;          % Initialize seed set
for i=1:k
    u_k = Row(d_min ≤ str2num(T_out(v_k,:)) ≤ d_max); % Check d_min and d_max
    A_k = T(u_k,:);                                    % Get graph of u_k
    v_k = Col(A_k);                                    % Neighbors of u_k
end
```

Incidence Matrix based Degree Filtered BFS

- **Uses the incidence matrix representation of a graph G to perform the BFS.**
- **Algorithm Inputs**

  v_0: starting vertex set

  k: number of hops to go

  T: Accumulo table of graph incidence matrix

  T_{col} = sum(logical(T==-1),1).'; % Node out-degrees

  d_{min}: minimum allowable degree

  d_{max}: maximum allowable degree
- **Algorithm Output**

  E_k: adjacency matrix of sub-graph

Incidence Matrix based Degree Filtered BFS

- **The algorithm begins by retaining vertices whose degree are between d_{min} and d_{max}**

- **Algorithm:**

$v_k = v_0$; % Initialize seed set

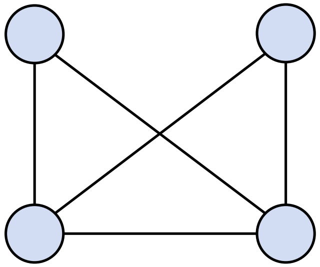
for i=1:k
    u_k = Row($d_{min} \leq$ str2num(T_{col}(v_k,:)) $\leq d_{max}$); % Check d_{min} and d_{max}
    E_k = T(Row(T(:,u_k)),:); % Get graph of u_k
    v_k = Col(E_k==1); % Get neighbors of u_k
end

Outline

- Introduction
- Degree Filtered Breadth First Search
- K-Truss
- Jaccard Coefficient
- Non-Negative Matrix Factorization
- Summary

K-Truss

- **A graph is a k-truss if each edge is part of at least k-2 triangles**
- **A generalization of a clique (a k-clique is a k-truss), ensuring a minimum level of connectivity within the graph**
- **Traditional technique to find a k-truss subgraph:**
  - **Compute the support for every edge**
  - **Remove any edges with support less than k-2 and update the list of edges**
  - **When all edges have support of at least k-2, we have a k-truss**



Example 3-truss

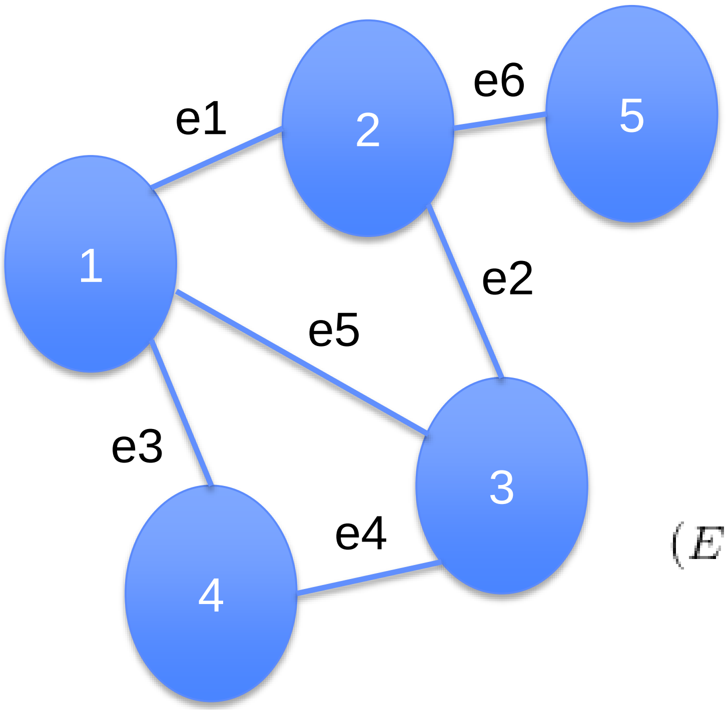
K Truss in Terms of Matrices

- **If E is the unoriented incidence matrix (rows are edges and columns are vertices) of graph G, and A is the associated adjacency matrix**
- **If G is a k-truss, the following must be satisfied:**
  - AND((E*A == 2) * 1 > k – 2)
  - where AND is the logical and operation
- **Why?**
  - E*A: each row of the result is the sum of rows in A associated with the two vertices of an edge in G
  - E*A == 2: Result is 1 where vertex pair of edge have a common neighbor
  - (E*A ==2) * 1 : Result is the sum of number of common neighbors for vertices of each edge
  - (E*A ==2) * 1 > k – 2: Result is 1 if more common neighbors than k-2

As an iterative algorithm

- **Strategy: start with the whole graph and iteratively remove edges that don't find the k-truss criteria**
- Adjacency Matrix (A) = $E^T E - \mathrm{diag}(E^T E)$
- **Algorithm:**
  - $R \leftarrow E*A$
  - $x \leftarrow \mathrm{find}((R = 2)*\mathbf{1} < k - 2)$ % x is edges preventing a k-truss
  - While x is not empty, do:
    - $E_x \leftarrow E(x, :)$ % get the edges to remove
    - $E \leftarrow E(x_c, :)$ % keep only the complementary edges
    - $R \leftarrow E(x_c, :)*A$ % remove the rows associated with non-truss edges
    - $R \leftarrow R - E * [E_x E_x^T - (\mathrm{diag}(E_x E_x^T))]$ %update R
    - $x \leftarrow \mathrm{find}((R==2)*\mathbf{1} < k-2)$ %update x
- **GraphBLAS kernels required: SpGEMM, SpMV**

For example: find a 3-truss of G



$$E = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 \end{bmatrix} \quad A = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix}$$

$$(E * A == 2) * \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 2 \\ 0 \end{bmatrix}$$

For 3 truss, k=3

$$x = 6$$

$$x^c = \begin{bmatrix} 1 & 2 & 3 & 4 & 5 \end{bmatrix}^T$$

$$R = R - E(x^c, :) * [E(x, :) * E(x, :)^T - diag(E(x, :) * E(x, :)^T)] = \begin{bmatrix} 1 & 1 & 2 & 1 & 1 \\ 2 & 1 & 1 & 1 & 1 \\ 1 & 1 & 2 & 1 & 0 \\ 2 & 1 & 1 & 1 & 0 \\ 1 & 2 & 1 & 2 & 0 \end{bmatrix}$$

3 truss SubGraph given by $E(x^c, :) = \begin{bmatrix} 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \end{bmatrix}$

Outline

- Introduction
- Degree Filtered Breadth First Search
- K-Truss
- Jaccard Coefficient
- Non-Negative Matrix Factorization
- Summary

Jaccard Index

- **The Jaccard coefficient measures the neighborhood overlap of two vertices in an unweighted, undirected graph**
- **Expressed as (for vertices v_i and v_j), where *N* is the neighbors:**

$$J_{ij} = \frac{|N(v_i) \cap N(v_j)|}{|N(v_i) \cup N(v_j)|}$$

- **Given the connection vectors (a column or row in the adjacency matrix A) for vertices v_i and v_j (denoted as a_i and a_j) the numerator and denominator can be expressed as $a_i^T a_j$ where the we replace multiplication with the AND operator in the numerator and the OR operator in the denominator**
- **This gives us:**

$$J_{ij} = (a_i^T \wedge a_j)./(a_i^T \vee a_j) = A^2_{AND}./A^2_{OR}$$

  – Where ./ represents the element by element division

Algorithm to Find Jaccard Index

- **Using the standard operations, A^2_{AND} is the same as A^2**
- **Also, the inclusion-exclusion principle gives us a way to compute** A^2_{OR} **when we have the degrees of the vertex neighbors d_i and d_j:** $A^2_{OR} = \Sigma d_i + \Sigma d_j - A^2_{AND}$
- **So, an algorithm to compute the Jaccard in linear algebraic terms would be:**
  - Initialize J to $A^{2:}$ J = triu(A^2) %Take upper triangular portion
  - Remove diagonal of J: J = J-diag(J)
  - For each non zero entry in J given by index i and j that correspond to vertices v_i and v_j:

$$J_{ij} = J_{ij}/(d_i + d_j - J_{ij})$$

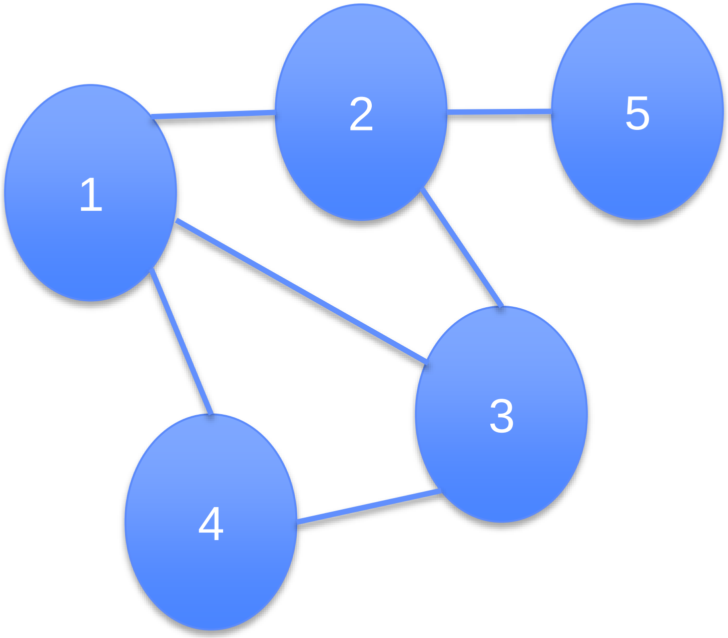
Example Jaccard Calculation

$$A = \begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 0 & 1 & 0 & 1 \\ 1 & 1 & 0 & 1 & 0 \\ 1 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \end{bmatrix} \quad A^2 = \begin{bmatrix} 3 & 1 & 2 & 1 & 1 \\ 1 & 3 & 1 & 2 & 0 \\ 2 & 1 & 3 & 1 & 1 \\ 1 & 2 & 1 & 2 & 0 \\ 1 & 0 & 1 & 0 & 1 \end{bmatrix} \quad J = \begin{bmatrix} 0 & 1 & 2 & 1 & 1 \\ 0 & 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$$J./(D_i + D_j - J) = \begin{bmatrix} 0 & 1 & 2 & 1 & 1 \\ 0 & 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} ./ \left(\begin{bmatrix} 0 & 3 & 3 & 3 & 3 \\ 0 & 0 & 3 & 3 & 3 \\ 0 & 0 & 0 & 3 & 3 \\ 0 & 0 & 0 & 0 & 2 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} + \begin{bmatrix} 0 & 3 & 3 & 2 & 1 \\ 0 & 0 & 3 & 2 & 1 \\ 0 & 0 & 0 & 2 & 1 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} - \begin{bmatrix} 0 & 1 & 2 & 1 & 1 \\ 0 & 0 & 1 & 2 & 0 \\ 0 & 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix} \right)$$

$$= \begin{bmatrix} 0 & 1/5 & 1/2 & 1/4 & 1/3 \\ 0 & 0 & 1/5 & 2/3 & 0 \\ 0 & 0 & 0 & 1/4 & 1/3 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

(computing on non-zero entries)



Efficiently Computing triu(A^2)

- **Since only the upper triangular part of** A^2 **is needed, we can exploit the symmetry of the matrix** A**, and its lack of nonzero values on the diagonal, to avoid some unnecessary computation**
- **Let** $A=(L+U)$**, where** L **and** U **are strictly lower and upper triangular, respectively**
  - **Note that** $L = U^T$**, since** A **is symmetric**
- **Then** $A^2 = (U^T)^2+U^TU+UU^T+U^2$
  - **Note that** $(U^T)^2$ **is lower triangular and** U^2 **is upper triangular**
- **Then** triu(A^2) **can be efficiently computed as follows:**
  - $U \leftarrow \text{triu}(A)$
  - $X \leftarrow U*U^T$
  - $Y \leftarrow U^T*U$
  - $X \leftarrow \text{triu}(X) + \text{triu}(Y) + U*U$
- **Now** triu(X) **is the same as** triu(A^2)

triu, tril, diag **as element-wise products**

- **A Hadamard (entrywise) matrix product can be used to implement functions that extract the upper- and lower-triangular parts of a matrix in the GraphBLAS framework**
- **To implement** triu**,** tril**, and** diag **on a matrix A, we perform** $A \otimes 1$
- **Where** $\otimes$ = f(i,j) **is a user defined multiply function that operates on indices of the non-zero element of A**
  – **For** triu(A) = $A \otimes 1$**, the upper triangle,** f(i,j) = {A(i,j): $i \leq j$, 0 otherwise}
  – **For** tril(A) = $A \otimes 1$**, the lower triangle,** f(i,j) = {A(i,j): $i \geq j$, 0 otherwise}
  – **For** diag(A) = $A \otimes 1$**, the diagonal,** f(i,j) = {A(i,j): i==j, 0 otherwise}

- triu**,** tril**, and** diag **all represent GraphBLAS utility functions than can be built with user defined multiplication capabilities found in the GraphBLAS**

Outline

- Introduction
- Degree Filtered Breadth First Search
- K-Truss
- Jaccard Coefficient
- Non-Negative Matrix Factorization
- Summary

Topic Modeling

- **Common tool for individuals working with big data**
  - **Quick summarization**
  - **Understanding of common themes in dataset**
  - **Used extensively in recommender systems and similar systems**
- **Common techniques: Latent dirichlet allocation, Latent semantic analysis, Non-negative matrix factorization (NMF)**
- **Non-negative matrix factorization is a (relatively) recent algorithm for matrix factorization that has the property that the results will be positive**
- **NMF applied on a matrix A_{mxn}:**

$$A_{mxn} = W_{mxk} * H_{kxn}$$

  - where W, H are the resultant matrices and k is the number of desired topics
- **Columns of W can be considered as basis for matrix A and rows of H being the associated weights needed to reconstruct A (or vice versa)**

NMF through Iteration

- **One way to compute the NMF is through an iterative technique known as alternating least squares given below:**

Data: Adjacency Matrix A (size mxn), number of topics k
Result: W and H
initialization;
W = random m x k matrix
while $\|A - W * H\|_F > threshold$ **do**
  Solve $W^T * W * H = W * A$ for H
  Set elements in $H < 0$ to 0
  Solve $H * H^T * W^T = H * A^T$ for W
  Set elements in $W < 0$ to 0
end

- **A challenge implementing the above is in determining the matrix inverse (essentially the solution of a least squares problem for alternating W and H)**

Matrix Inversion through Iteration

- **A (not too common) way to solve a least squares problem is to use the relation that** $x_{k+1} = x_k * (2 - X_n)$
- **In matrix notation,** $X_{k+1} = X_k * (2I - AX_n)$
- **Thus, to compute the least squares solution, we can use an algorithm as below:**

Data: Matrix A to invert
Result: $X = A^{-1}$
initialization;
$\|A_{row}\| = \max_i(\sum_j A_{ij})$
$\|A_{col}\| = \max_j(\sum_i A_{ij})$
$X_1 = A^T/(\|A_{row}\| * \|A_{col}\|)$
while *for some time* **do**
| $X_{t+1} = X_t * (2 * I_{nxn}) - A * X_t$
end

Combining NMF and matrix inversion

- **The previous two slides can be combined to provide an algorithm that uses only GraphBLAS kernels to determine the factorization of a matrix A (which can be a matrix representation of a graph)**

Data: Adjacency Matrix A (size mxn), number of topics k
Result: W and H
W = random m x k matrix
while *Frobenius norm of* $A - W * H > threshold$ **do**
  Solve $H = (W^T * W)^{-1} * W * A$ for H
  Set elements in $H < 0$ to 0
  Solve $W^T = (H * H^T)^{-1} * H * A^T$ for W
  Set elements in $W < 0$ to 0
end

Mapping to GraphBLAS

- **In order to implement the NMF using the formulation, the functions necessary are:**
  - **SpRef/SpAsgn**
  - **SpGEMM**
  - **SpEWiseX**
  - **Scale**
  - **Reduce**
  - **Addition/Subtraction (can be realized over (min,+) semiring with scale operator)**
- **Challenges:**
  - **Major challenge is making sure pieces are sparse. The matrix inversion process may lead to dense matrices. Looking at other ways to solve the least squares problem through QR factorization (however same challenge applies)**
  - **Complexity of the proposed algorithm is quite high**

Summary

- The GraphBLAS effort aims to standardize the kernels used to express graph algorithms in terms of linear algebraic operations
- One of the important aspects in standardizing these kernels is in the ability to perform common graph algorithms
- This presentation highlights the applicability of the current GraphBLAS kernels applied to four popular analytics:
  - Degree Filtered Breadth First Search
  - K-Truss
  - Jaccard Index
  - Non-negative matrix factorization