Software and Algorithms for Graph Queries on Massively Multithreaded Architectures

Jonathan Berry (Sandia Labs)

Bruce Hendrickson (Sandia Labs)

Simon Kahan (Google)

Petr Konecny (Cray)

March 29, 2007

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed Martin Company, for the United States Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000.

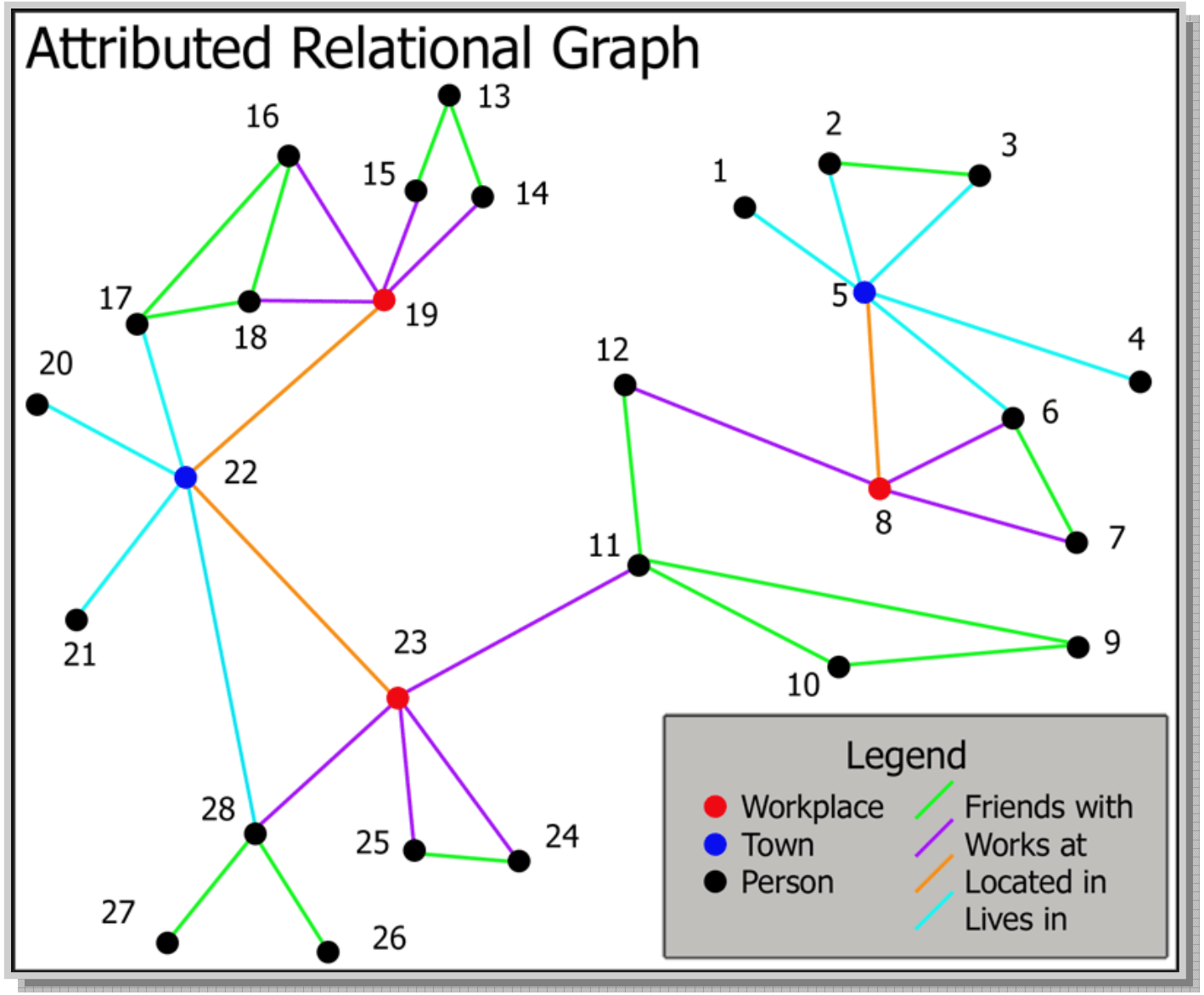
Sandia National Laboratories

Outline

- **Graph-based informatics**
- **Massively-multithreaded architectures**
- **Sandia's prototype Multithreaded Graph Library (MTGL)**
- **Algorithmic case studies on the Cray MTA-2**
- **Current and future directions, opportunities for collaboration**

Sandia National Laboratories

Graph-Based Informatics



Sandia National Laboratories

Massive Multithreading: e.g., The Cray MTA-2

- **Slow clock rate (220Mhz)**
- **128 “streams” per processor**
- **Global address space**
- **Fine-grain synchronization**
- **Simple, serial-like programming model**
- **Advanced parallelizing compilers**

No Data Cache

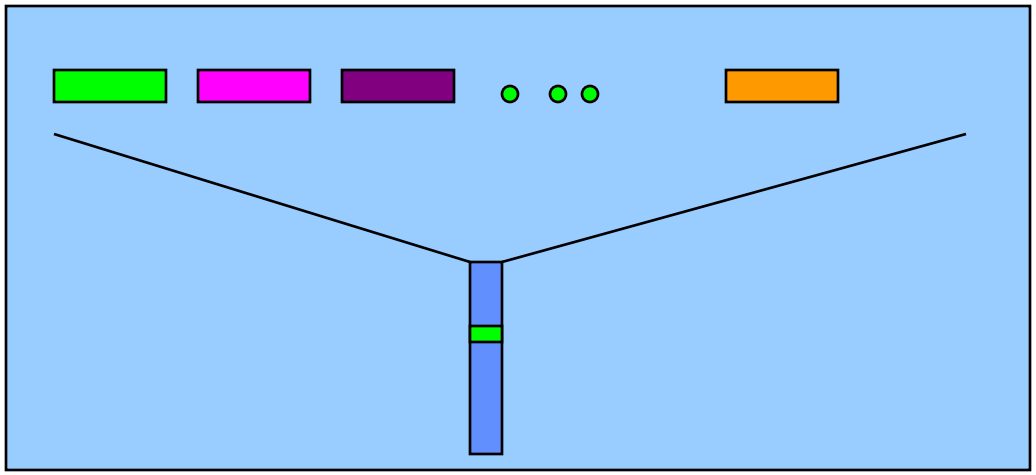
Latency Tolerant:

important for Graph Algorithms

Hashed Memory

Sandia National Laboratories

Cray MTA Processor



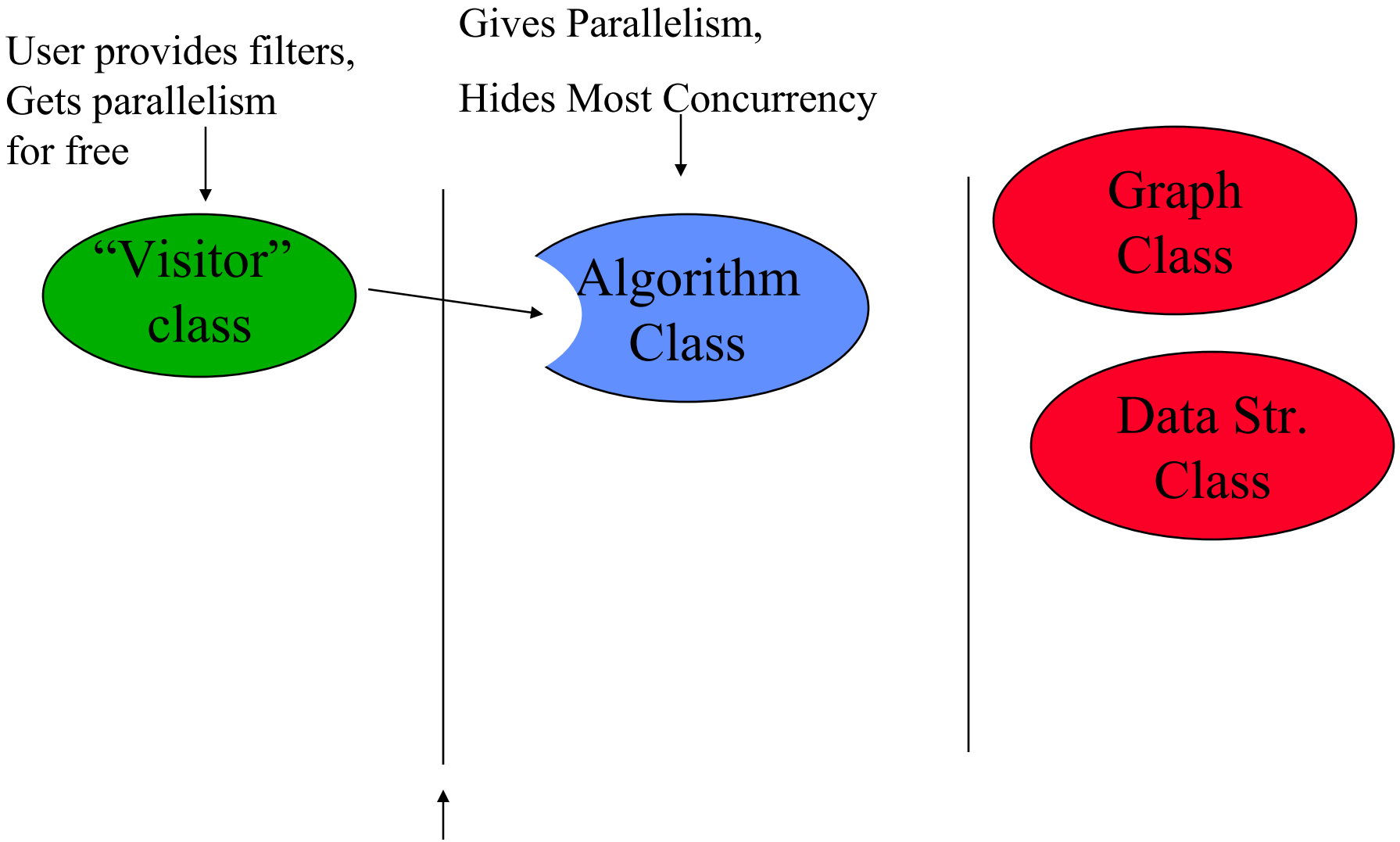
- Each thread can have up to 8 memory refs in flight
- Round trip to memory ~150 cycles

The MultiThreaded Graph Library (MTGL)

- **Recent work**
  - Design & implement the Multithreaded Graph Library (MTGL)
    - Boost GL-like, but not Boost-based
    - In the process of open-sourcing
    - Develop, run, debug codes on workstations and/or MTA/XMT
  - Design multithreaded algorithms
    - Simon Kahan's connected components algorithm
    - "Bully" connected components algorithm
    - Subgraph isomorphism for semantic graphs

- **Current work**
  - Adapt the MTGL to run on SMP's
  - Flesh out the MTGL with community finding, etc.
  - Integrate the MTGL with visualization frameworks

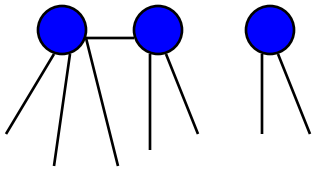
MTGL: C++ Design Levels



Four Modes of MTGL Graph Exploration

for v in V: visit_adj

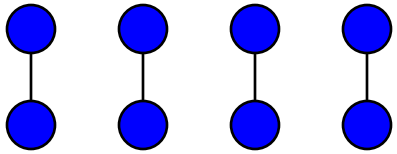
visit v's neighbors



~10 memref/edge

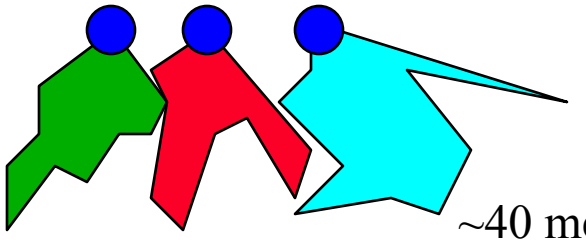
for e in E: visit_edges

visit e's endpoints



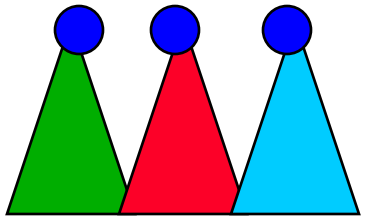
~10 memref/edge

recursive parallel search psearch



~40 memref/edge

breadth-first search bfs



~20-40 memref/edge

Sandia National Laboratories

Current MTGL Algorithms

- Connected components (psearch, visit_edges, visit_adj)
- Strongly-connected components (psearch)
- Maximal independent set (visit_edges)
- Typed subgraph isomorphism (psearch, visit_edges)
- S-t connectivity (bfs)
- Single-source shortest paths (psearch)
- Betweenness centrality (bfs-like)
- Connection subgraphs (bfs, sparse matrix, mt-quicksort)
- Find triangles (psearch)
- Find assortativity (psearch)
- Find modularity (psearch)

Under development:

- Optimize modularity via simulated annealing (visit_adj, visit_edges)
- Count edges in 1, 2 neighborhoods (various)
- more

MTGL PSearch Primitive: visitors

```
DefaultVisitor {
   sr(v) {}       # search root
   d(v) {}        # discover
   vt(v) {}       # visit test
   te(v,w) {}  # tree edge
   oe(v,w) {} # "other" edge
   …
}
```

The search primitives give the programmer these opportunities to react to search events

Pseudocode for this talk uses these shortened method names

Abuse of notation: multiple edges *are* allowed

Sandia National Laboratories

MTGL Search Primitives (e.g)

```
PSearch<OR,vis>(v)
{
   vis.d(v)
   for (v,w) in E(v):
        if (v,w) unvisited OR vis.vt(v,v')
             vis.te(v,w)
             PSearch<OR,vis>(w)
        else
             vis.oe(vw)
}
```

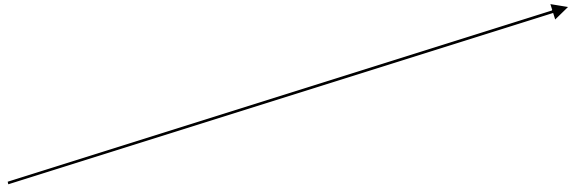
```
PSearch<AND,vis>(v)
{
   vis.d(v)
   for (v,w) in E(v):
      if vis.vt(v,v')
        if (v,w) unvisited
             vis.te(v,w)
             PSearch<AND,vis>(w)
        else
             vis.oe(vw)
}
```

More general, but details omitted here…

MTGL Synchronization Primitives

MTA Primitive	Meaning	Pseudocode	MTGL
b = int_fetch_add(a,i)	Atomic read, increment of *a*	$b \xleftarrow{\text{ifa}} a, i$	mt_incr(a,i)
b = readfe(&a)	Wait for *a* to be "full," read *a,* leave empty	$b \xleftarrow{\text{fe}} a$	mt_readfe(a)
b = readff(&a)	Wait for *a* to be "full," Read *a,* leave full	$b \xleftarrow{\text{ff}} a$	mt_readff(a)
writeef(&a, v)	Wait for *a* to be "empty," Write *a,* leave full	$b \xleftarrow{\text{ef}} a$	mt_write(a,v)

Overloaded for different platforms



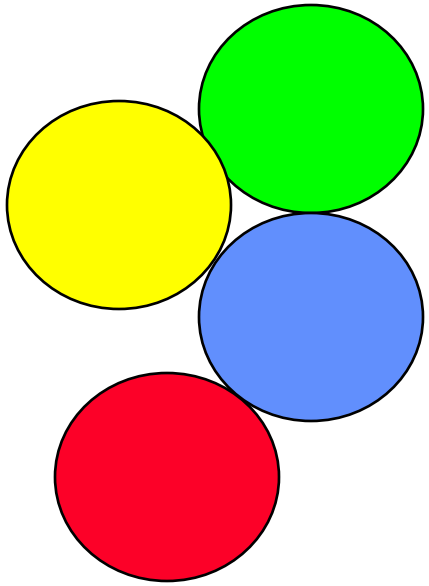
Sandia National Laboratories

Case Studies: Algorithm Kernels

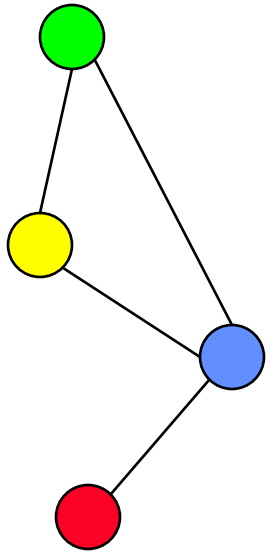
- **Connected Components**
- **Subgraph Isomorphism**
- **S-T Connectivity (i.e., use of BFS)**

Sandia National Laboratories

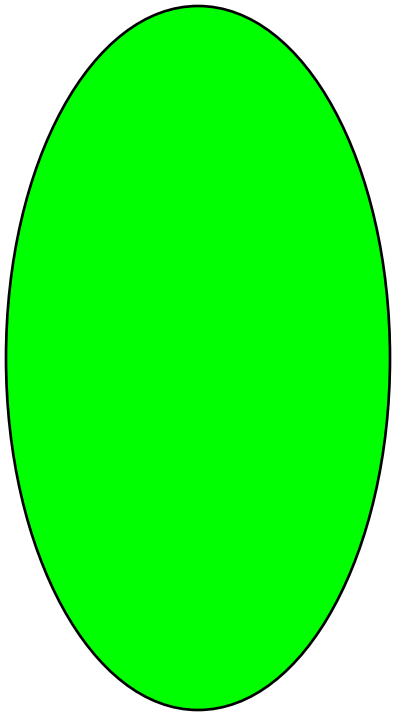
Kahan's Algorithm for Connected Components



Phase 1:
concurrent
searches

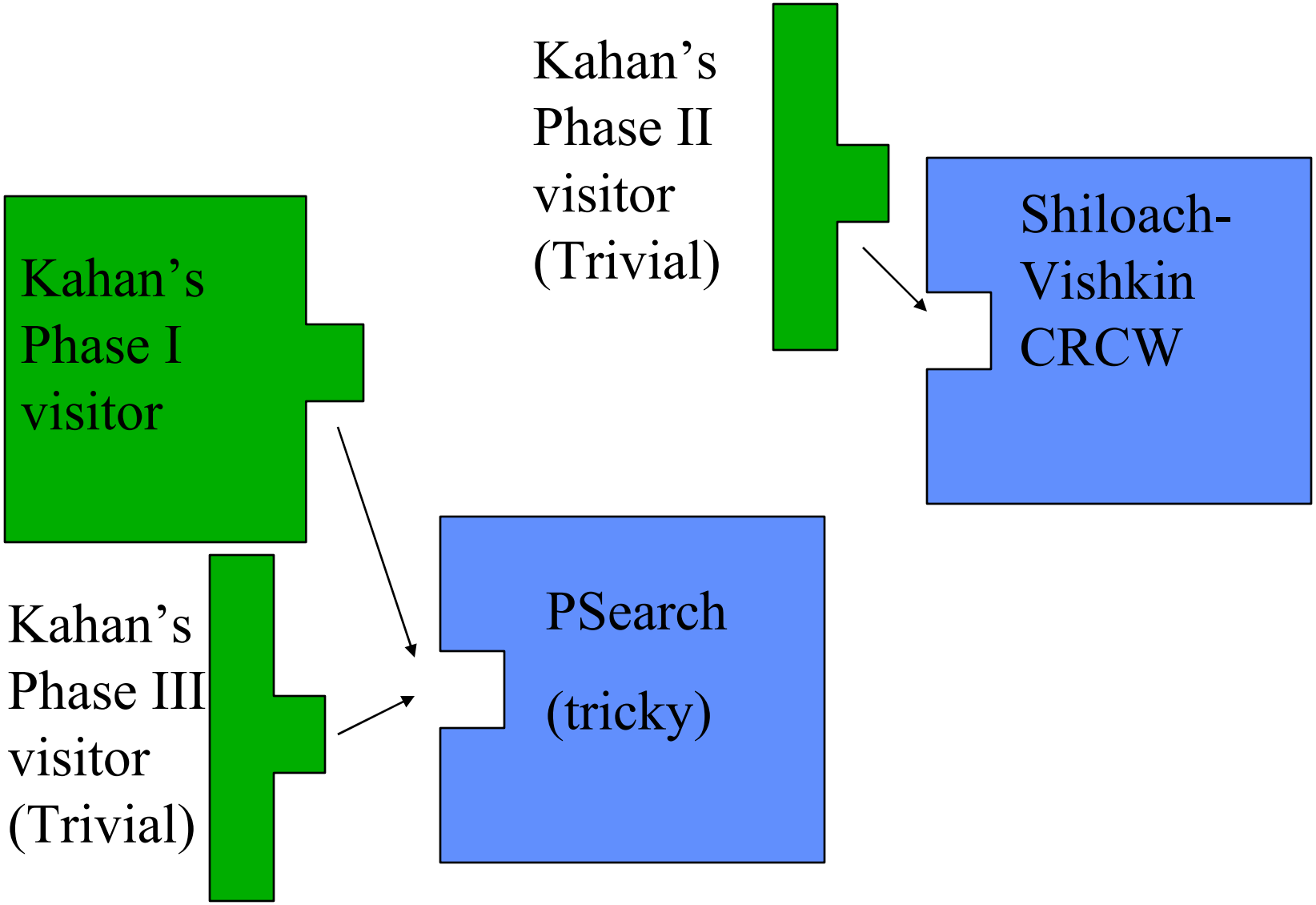


Phase 2: CRCW
(Shiloach-Vishkin)



Phase 3:
relabel

MTGL Implementation of Kahan's Algorithm



MTGL Implementation of Kahan's Algorithm

Phase I visitor:

$V_1 = \{$

$C \leftarrow$ user array of $|V(G)|$ ints

$T \leftarrow$ hash table of (int, int) pairs

sr(v) $\{ C[v] \xleftarrow{ef} v \}$

te(v, w) $\{ C[w] \xleftarrow{ef} C[v] \}$

oe(v, w) $\{ cv \xleftarrow{ff} C[v];$

$cw \xleftarrow{ff} C[w];$

$T \xleftarrow{ts} T \cup \{(cv, cw)\}$

$\}$

$\}$

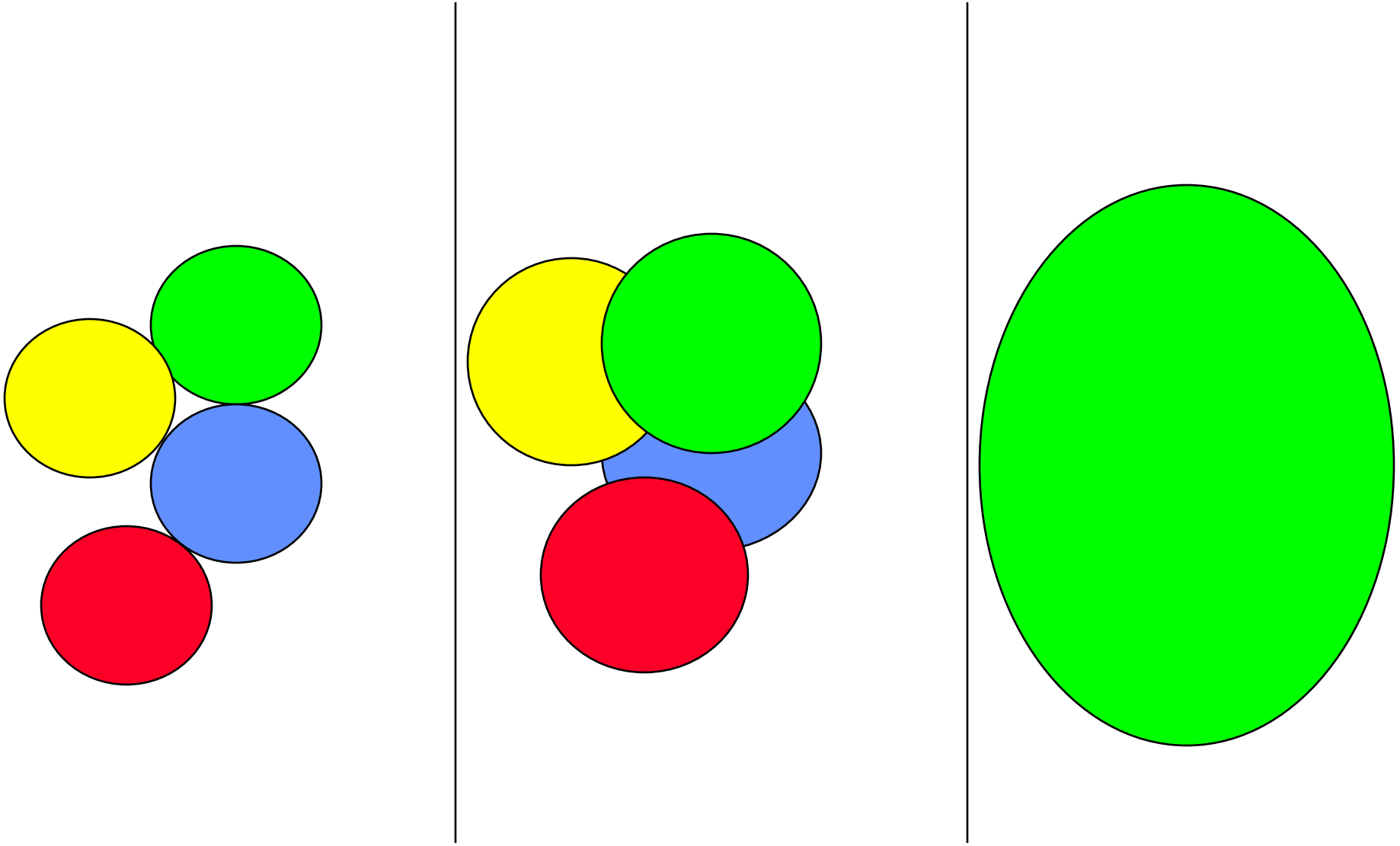
"component" values were defined by user to start "empty"

Wait until both full

Thread-safe Add to hash table

Sandia National Laboratories

More General Filtering: The “Bully” Algorithm



Sandia National Laboratories

MTGL Implementation of the Bully Algorithm

$V_b = \{$

.....

vt(v, w) $\{cv \xleftarrow{ff} C[v];\quad cw \xleftarrow{ff} C[w];$

return $(cv < cw)$;

$\}$

te(v, w) $\{cv \xleftarrow{ff} C[v];\quad cw \xleftarrow{fe} C[w];$

if w unvisited OR $(cv < cw)$

$C[w] \xleftarrow{ef} C[v]$

else

$C[w] \xleftarrow{ef} cw$

$\}$

$\}$

Sandia National Laboratories

Parallelized Shiloach-Vishkin (Bader, Feo, Cong'06)

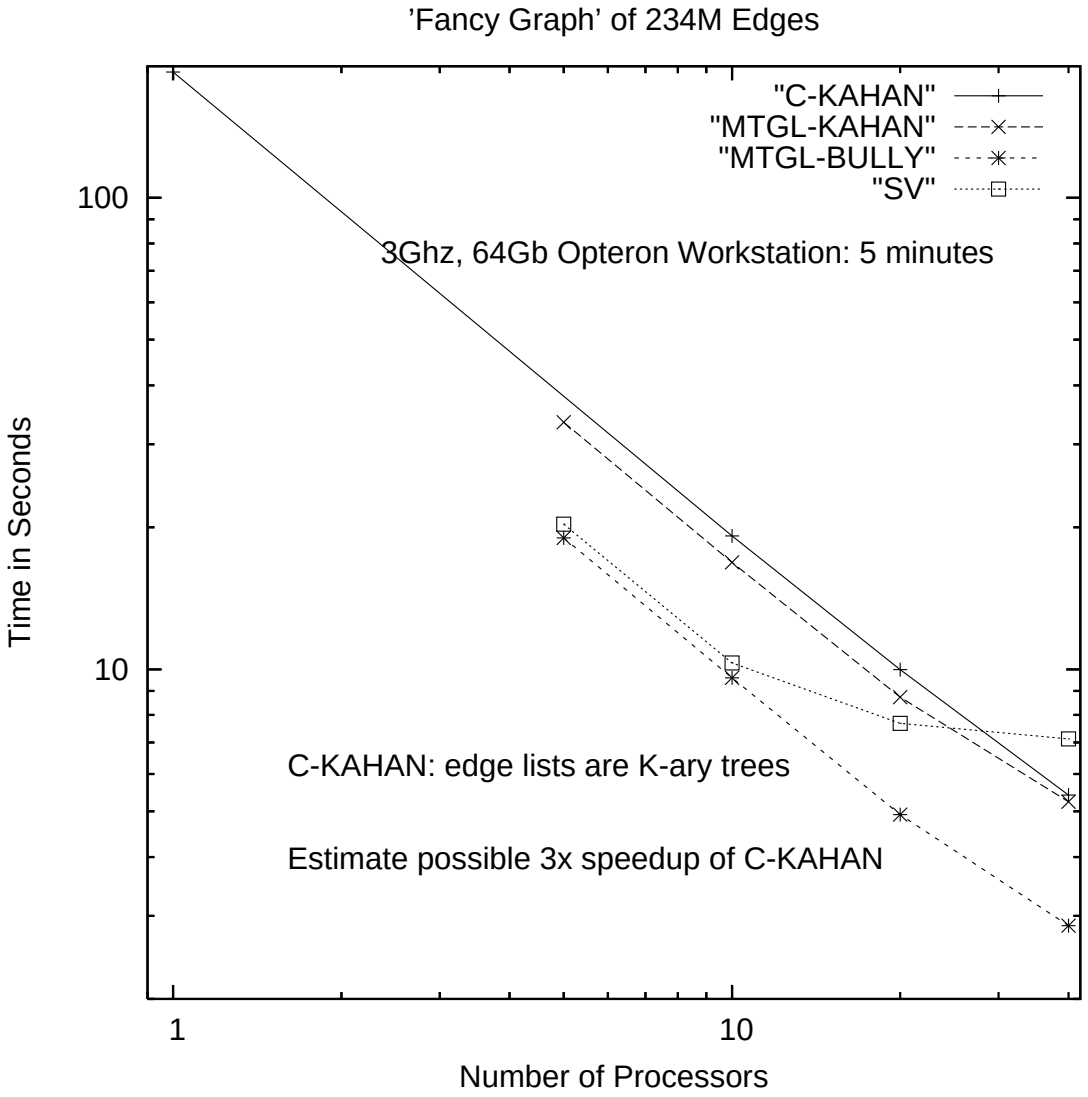
```
edge   **edges    = g->edges.getStore();
   while (graft) {
        numiter++;
        graft = 0;
        #pragma mta assert parallel
        for (int i=0; i<m; i++) {
            int u = edges[i]->vert1->id;
            int v = edges[i]->vert2->id;
            if (D[u] < D[v] && D[v] == D[D[v]]) {
                D[D[v]] = D[u];
                graft = 1;
            }
            if (D[v] < D[u] && D[u] == D[D[u]]) {
                D[D[u]] = D[v];
                graft = 1;
            }
        }
        #pragma mta assert parallel
        for (int i=0; i<n; i++) {
            while (D[i] != D[D[i]]) D[i] = D[D[i]];
        }
   }
```

"Graft"

"Hook"

Sandia National Laboratories

Connected Components Performance (1)

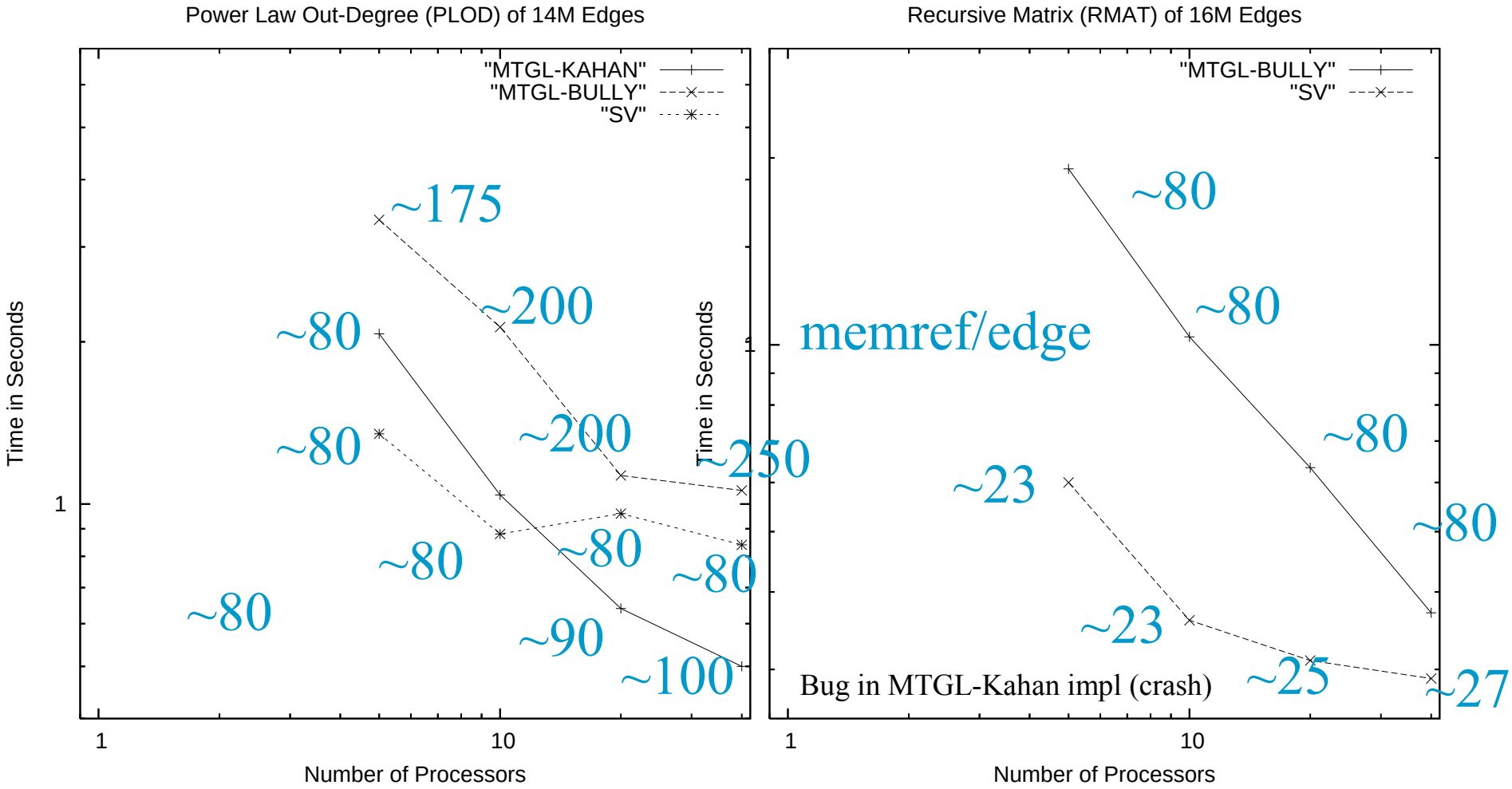


~2.9s

~40 memref/edge

Sandia National Laboratories

Connected Components Performance (2)

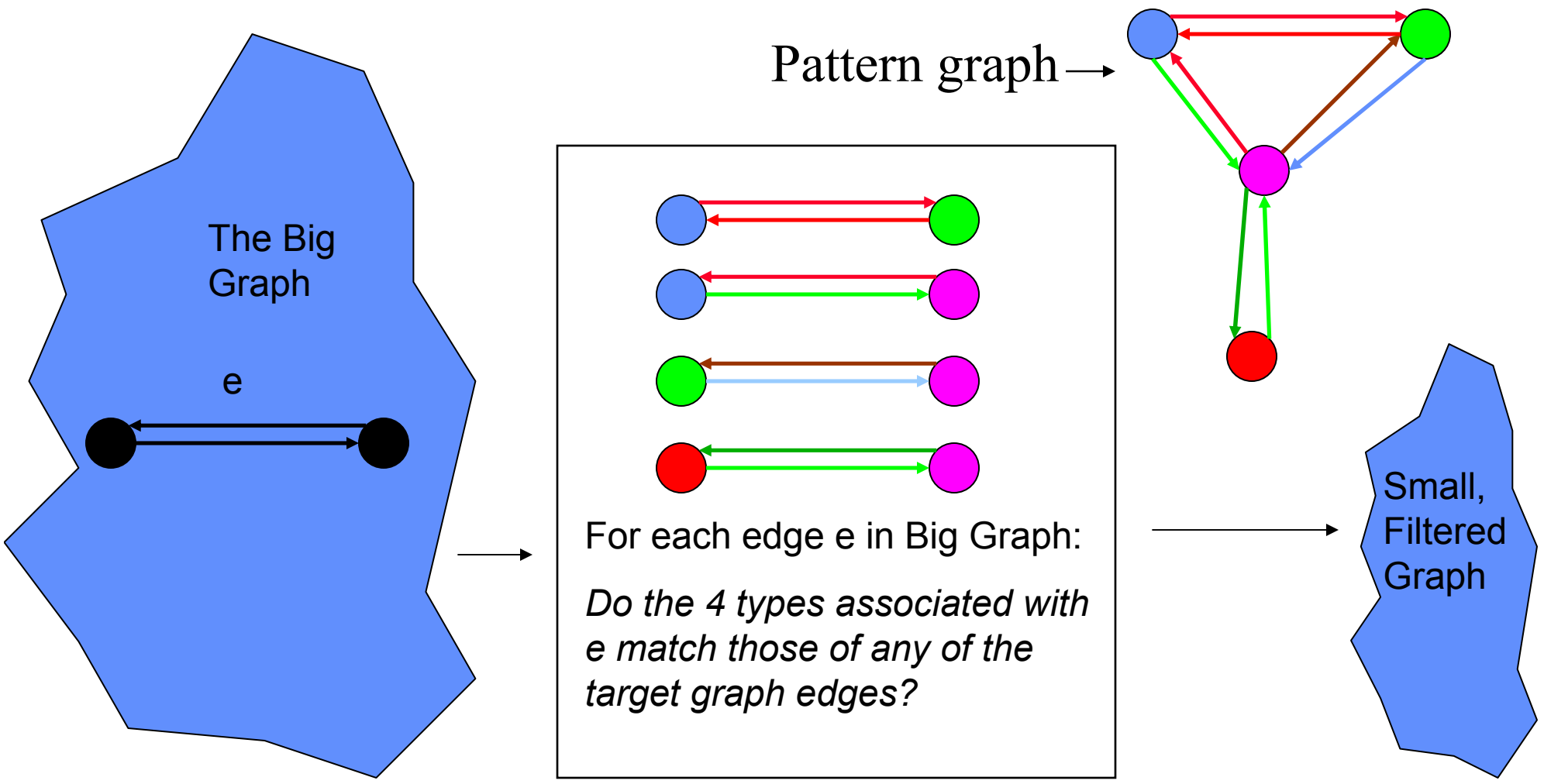


Sandia National Laboratories

Case Study: Subgraph Isomorphism Kernel

- **Objective: find exact or inexact matches of a small pattern graph within a large semantic graph**

Instance-Specific Type Filtering for Subgraph Iso.



Subgraph Isomorphism: Input

The Target Graph:

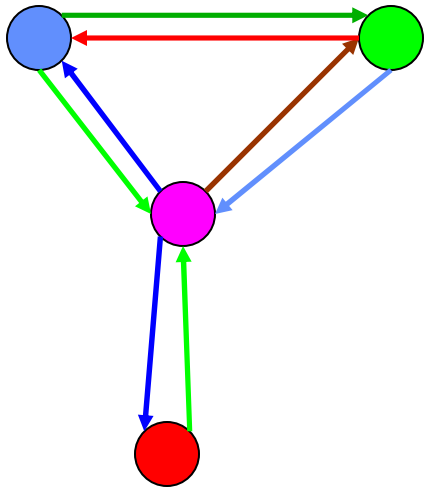


Table of Type and Auxiliary Information:

T																	
V	**2**		**2**		**3**		**2**		**3**		**1**		**3**		**2**		**2**

Ideal: Euler Tour

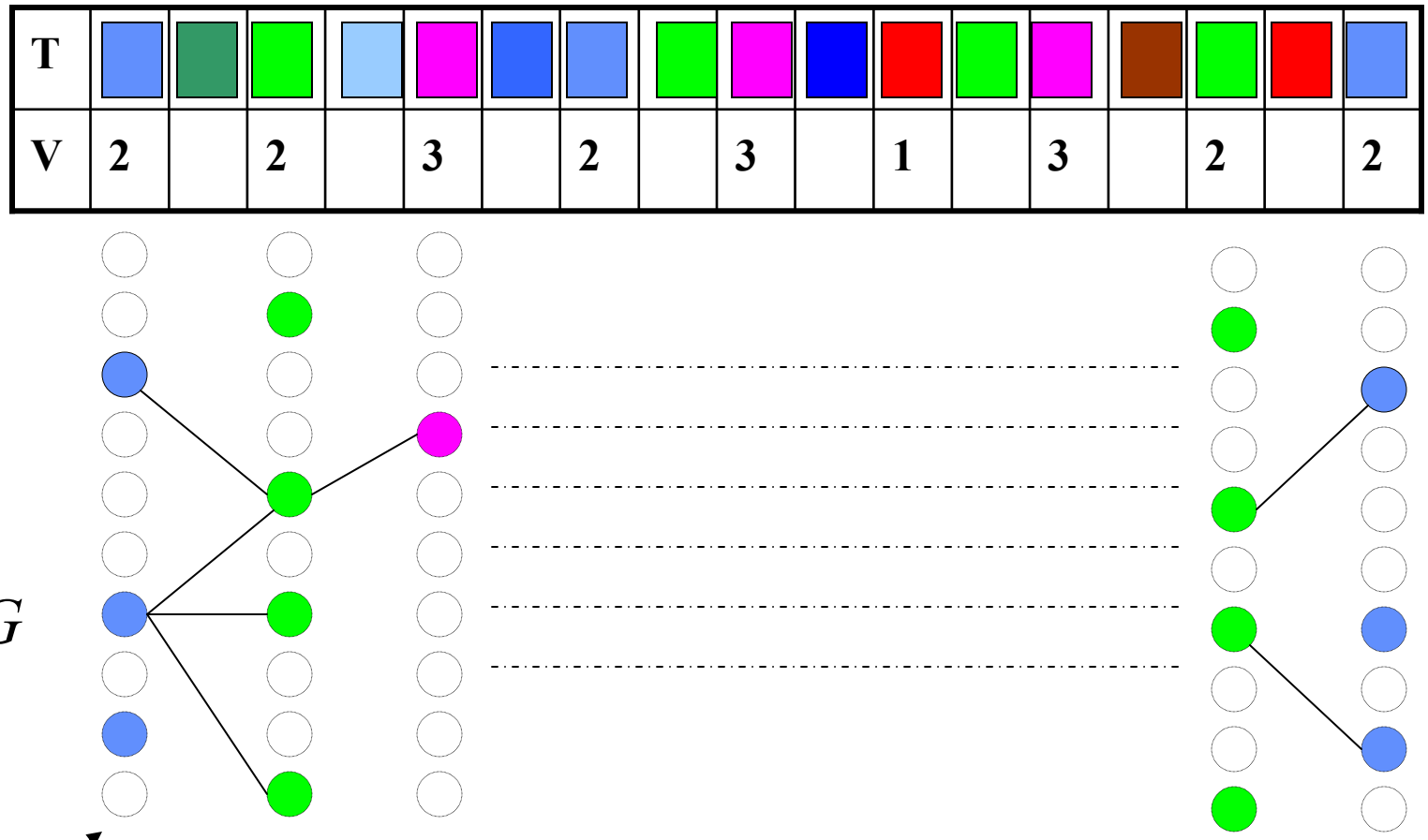
Our Experiments: Random Walk

Sandia National Laboratories

Subgraph Isomorphism: Creating a Bipartite Graph

Small, Filtered Graph (SFG)

k times:
visit each edge of SFG

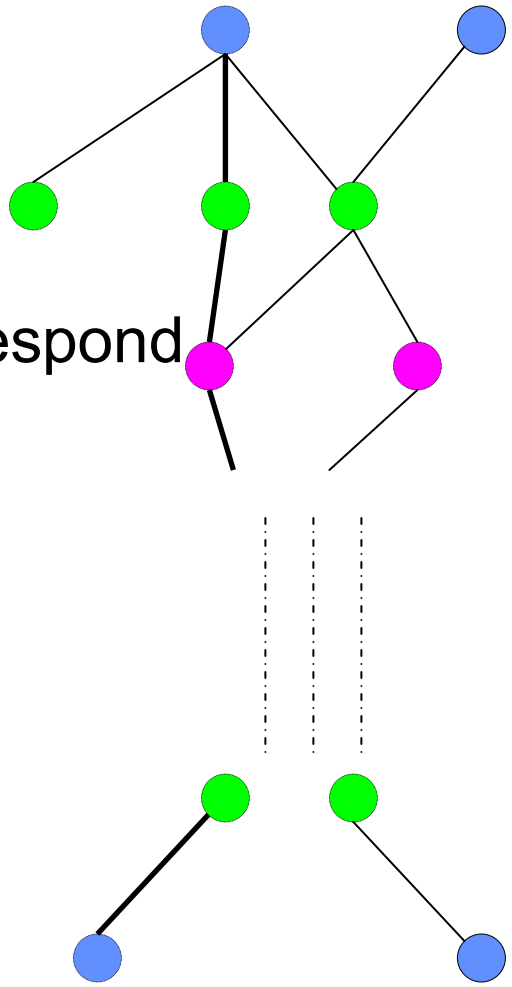


Logical placeholders for vertices in the SFG.

Subgraph Isomorphism: Creating a Bipartite Graph

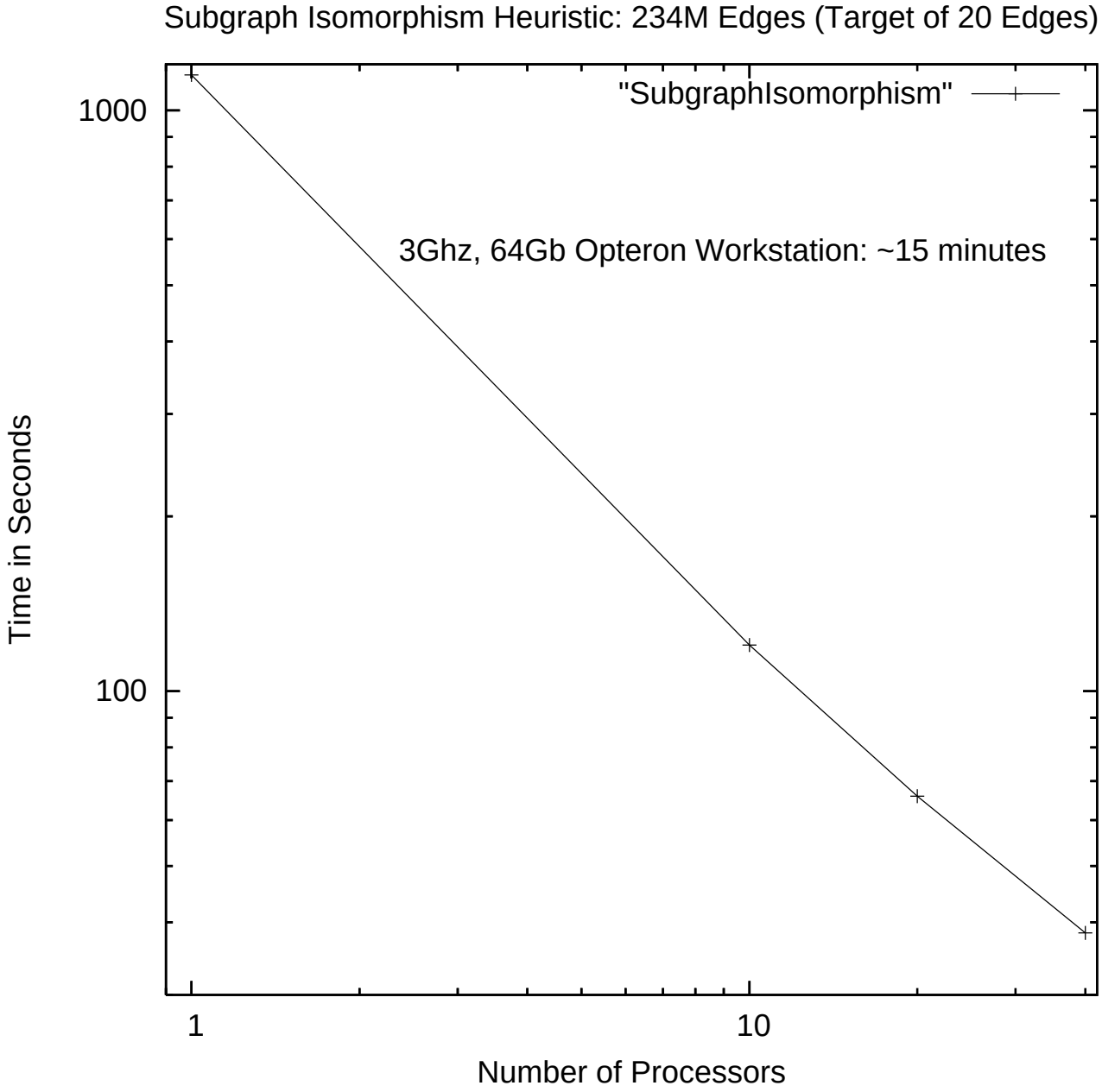
S-T shortest paths (top to bottom) correspond to candidate matches.

Branch and bound to Find better matches.



Visitor object tailors Search so that it never goes up (similar to "Bully" algorithm).

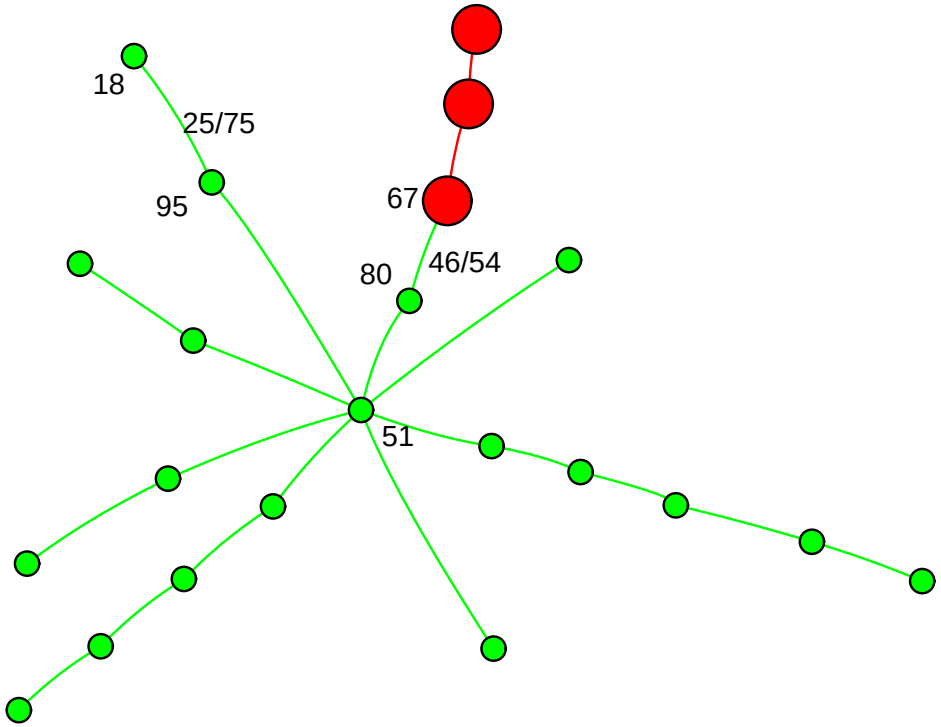
Computational Results: Subgraph Isomorphism



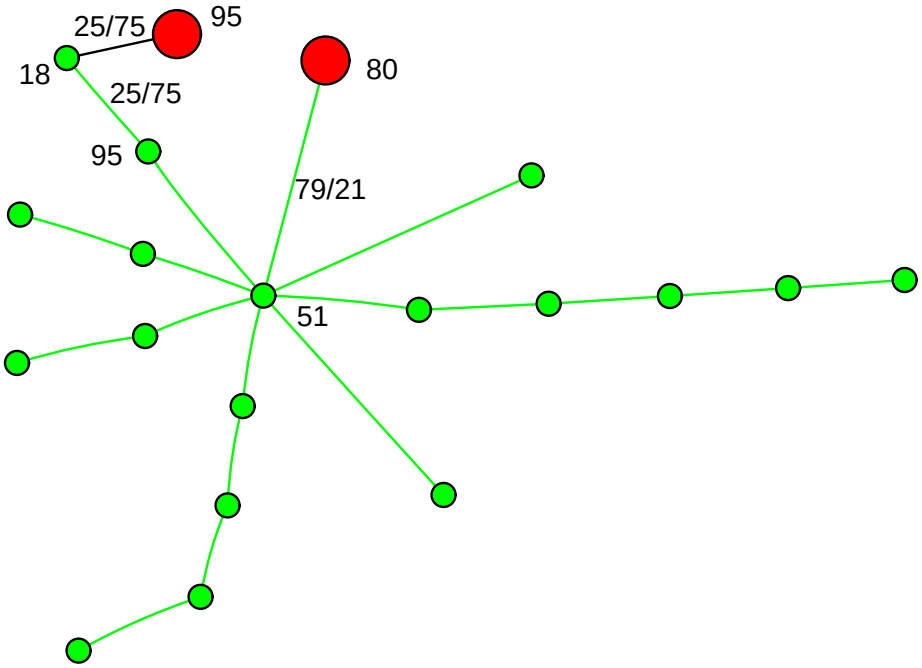
Sandia National Laboratories

Computational Results: Subgraph Isomorphism

Type & topological isomorphism exists between green vertices



Target



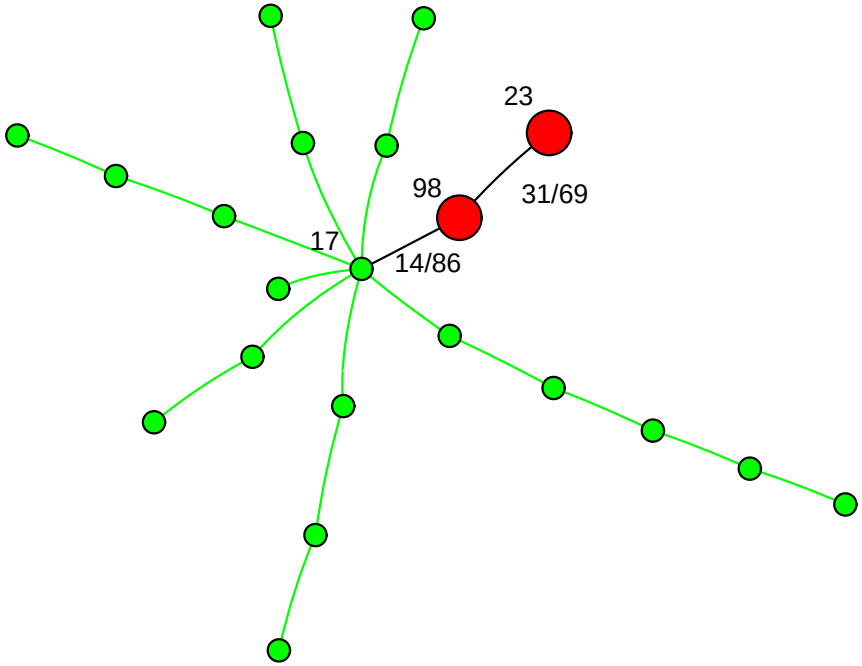
Found

Actual graphs from a 234M edge instance

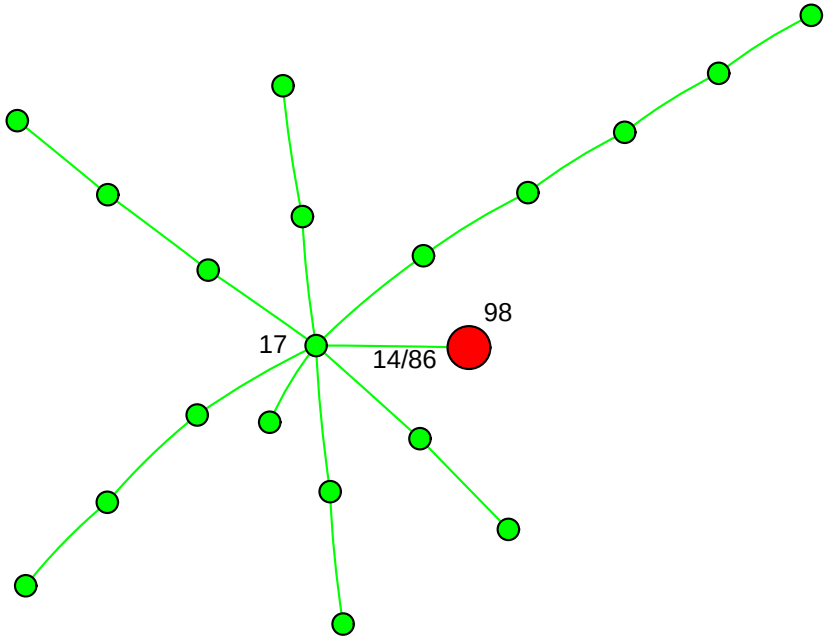
Sandia National Laboratories

Can try harder if we want a closer match

Type & topological isomorphism exists between green vertices



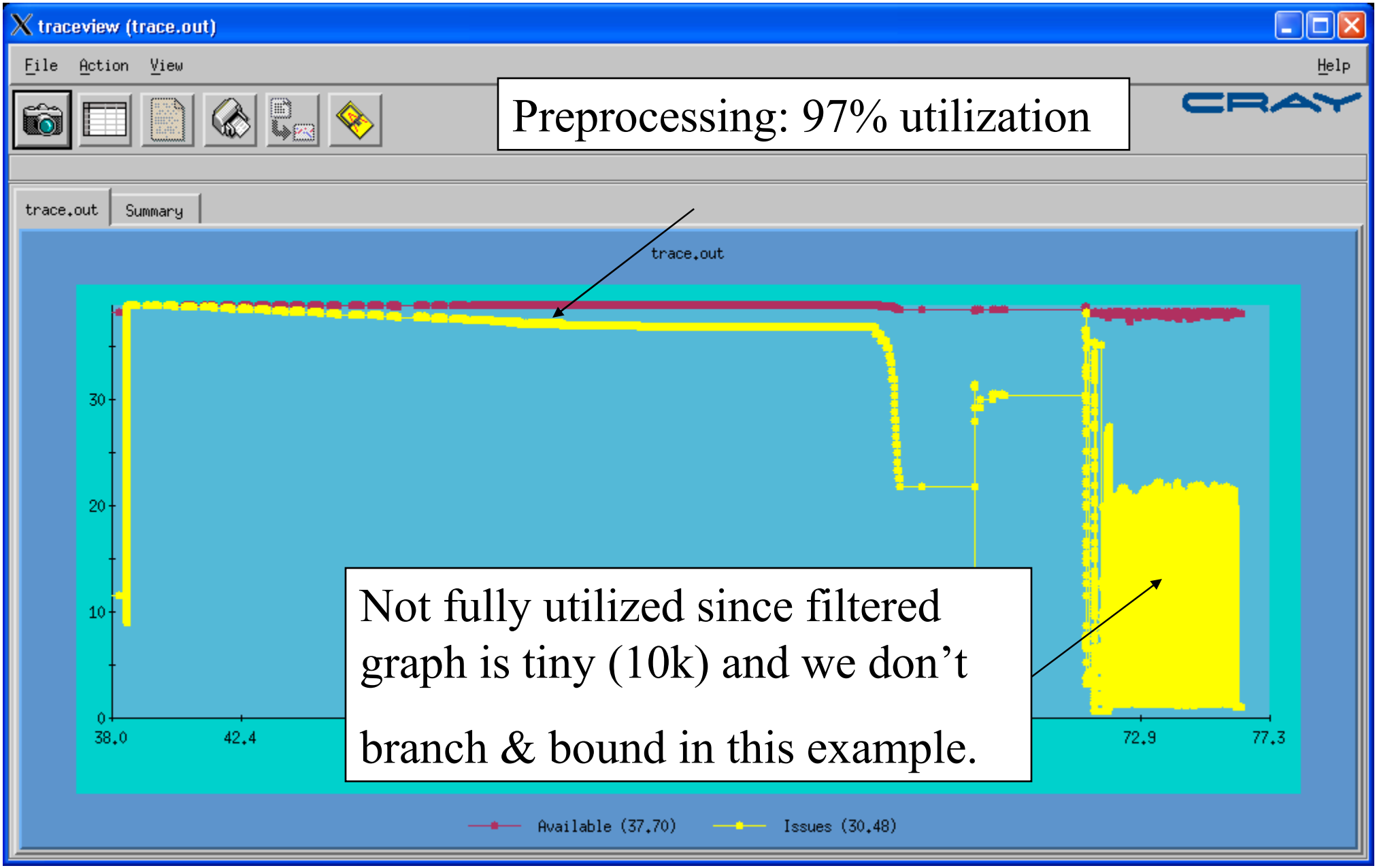
Target



Found

Actual graphs from a 234M edge instance

Traceview Output for Subgraph Isomorphism



Future

- **Massively-multithreaded Graph Algorithm R&D**
- **Open-Sourcing of the MTGL**
- **Run on SMP**
- **Query system with shared, mmapped graphs**

Sandia National Laboratories

Acknowledgements

- Bruce Hendrickson (Graph Informatics lead)
- Keith Underwood (Eldorado performance prediction)

- Simon Kahan, Petr Konecny (Cray): help in all aspects of this project
- Wayne Wong (Cray): Eldorado simulations

- Curt Janssen (usage model)
- Bob Heaphy (matrix-vector kernel)
- David Bader, Kamesh Madduri (Ga. Tech) (s-t connectivity)
- Cindy Phillips (help with algorithms)
- John Cieslewicz (Columbia) (database operations)
- Joe Crobak (Lafayette) (single-source shortest paths)
- Andrew Lumsdaine (Indiana U.) (Parallel Boost Graph Library)
- Douglas Gregor (Indiana U.) (Parallel Boost Graph Library)
- Others!

Sandia National Laboratories