Graph Software Development and Performance on the MTA-2 and Eldorado

Jonathan Berry

Bruce Hendrickson

Sandia National Laboratories

Presentation at CUG 2006

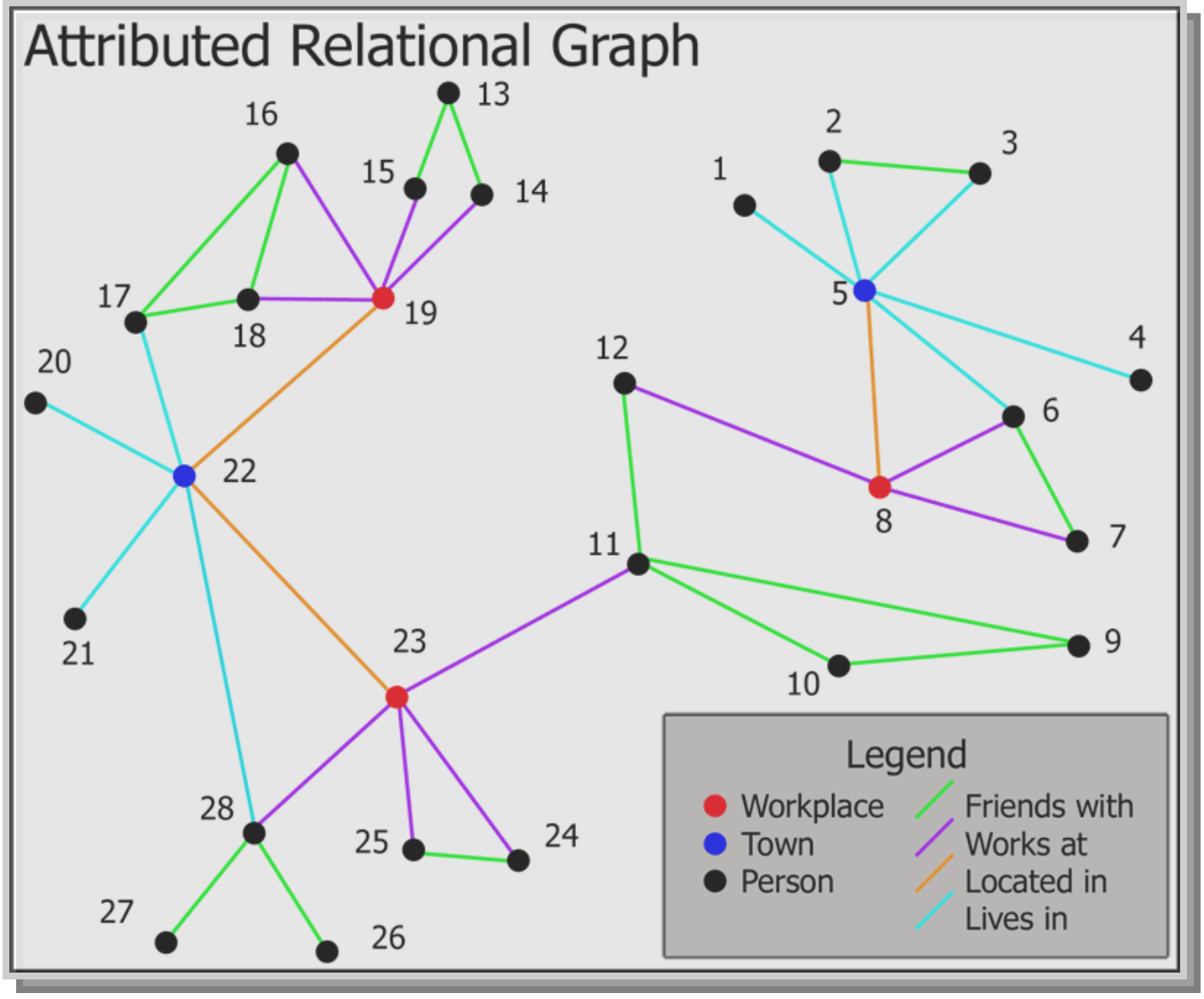
May 11, 2006

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.

Outline

- **Graph-based informatics**
- **Massively-multithreaded architectures**
- **Sandia's prototype graph infrastructure**
- **Algorithmic case studies on the Cray MTA-2**
- **Current and future directions**

Graph-Based Informatics



Graph-Based Informatics: Data

- **Graphs are giant**
- **Graphs are highly unstructured**
- **E.g.:**
  - 2^5 vertices of degree 2^20
  - 2^15 vertices of degree 2^10
  - 2^25 vertices of degree 5

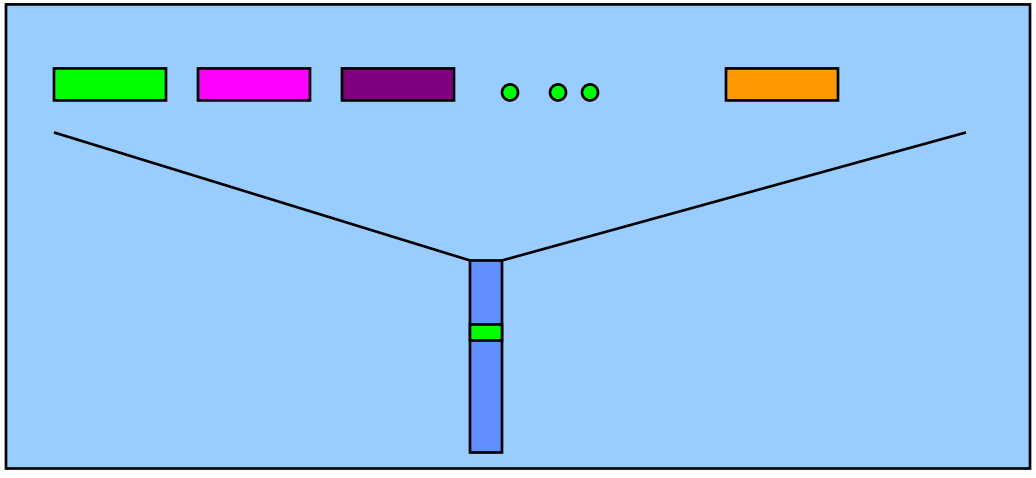
Massive Multithreading: The Cray MTA-2

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

Latency Tolerant:

important for Graph Algorithms

Cray MTA Processor



No Processor Cache

Hashed Memory

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

Take Home Messages

- **Multithreaded architectures**
  - Have huge performance advantages for sparse, unstructured discrete problems
  - Support a programming model on which generic software for unstructured problems can be written more effectively (no partitioning)

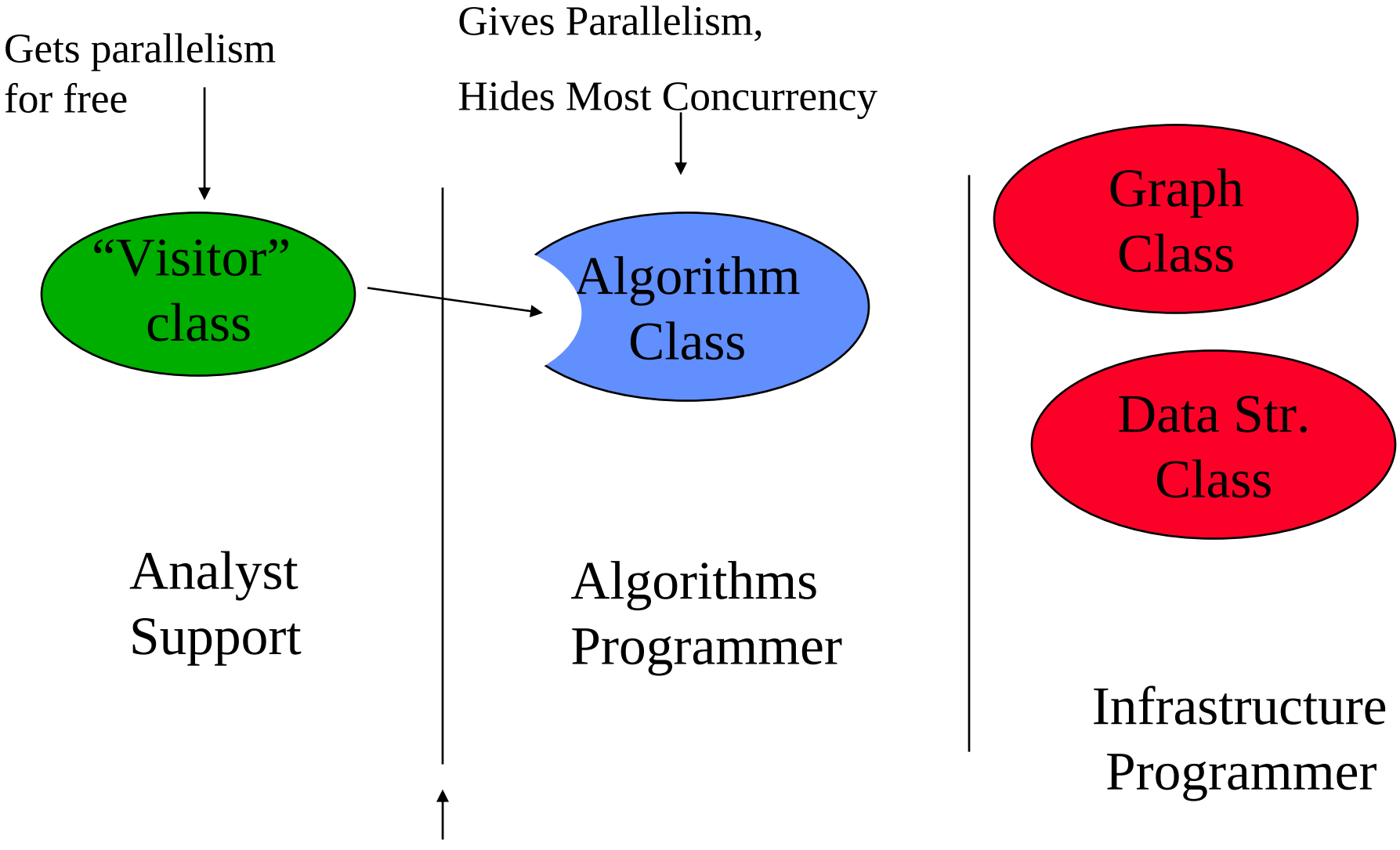
- **Sandia has developed a prototype graph infrastructure to support programming on these architectures**
  - Influenced by Boost GL, but not Boost GL
  - Nearly like serial code
  - Will be open-sourced

Graph Infrastructure Status

- **Design:** Just enough C++ to be flexible, support general filtering. ***Runs on MTA, Linux, Mac.***

- **Algorithmic kernels implemented using infrastructure:**
  - Connected components (linear scaling)
  - Subgraph isomorphism (linear scaling)
  - S-T connectivity (near-linear scaling)
- **Coding paradigm**
  - Search primitives hide MT issues, visitors ease development

Eldorado Graph Infrastructure: C++ Design Levels



Infrastructure Primitives

- Wrapped MTA primitives
  - int mt_incr(int& value, int incr);
  - int mt_readfe(int& value);
  - int mt_readff(int& value);
  - int mt_write(int& target, int value);
- Pure MTA pragmas
  - #pragma mta assert nodep
  - #pragma mta assert parallel
  - #pragma mta loop future

These wrap int_fetch_add, readfe, readff, and writeef

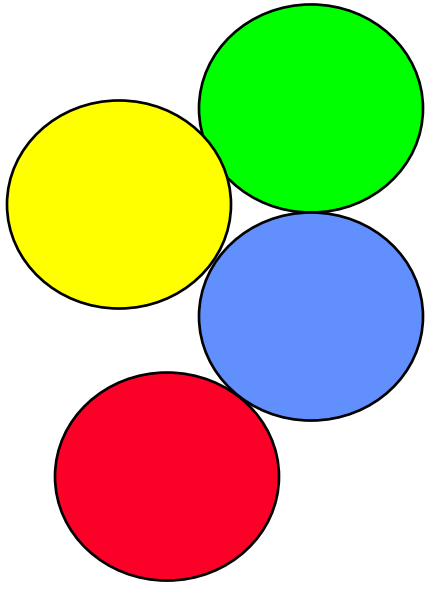
Allows efficient nested parallelism

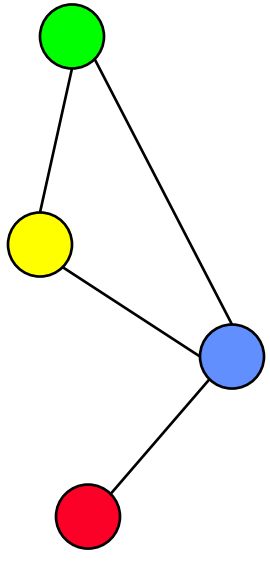
Case Studies: Algorithm Kernels

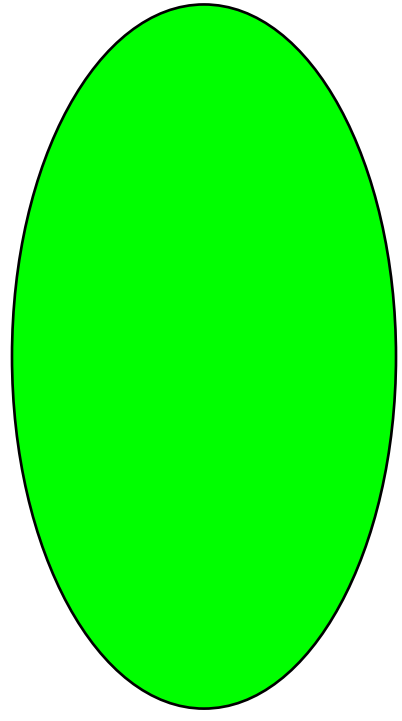
- **Connected Components**
- **S-T Connectivity (i.e., use of global queue)**
- **Subgraph Isomorphism (time permitting)**

Sandia National Laboratories

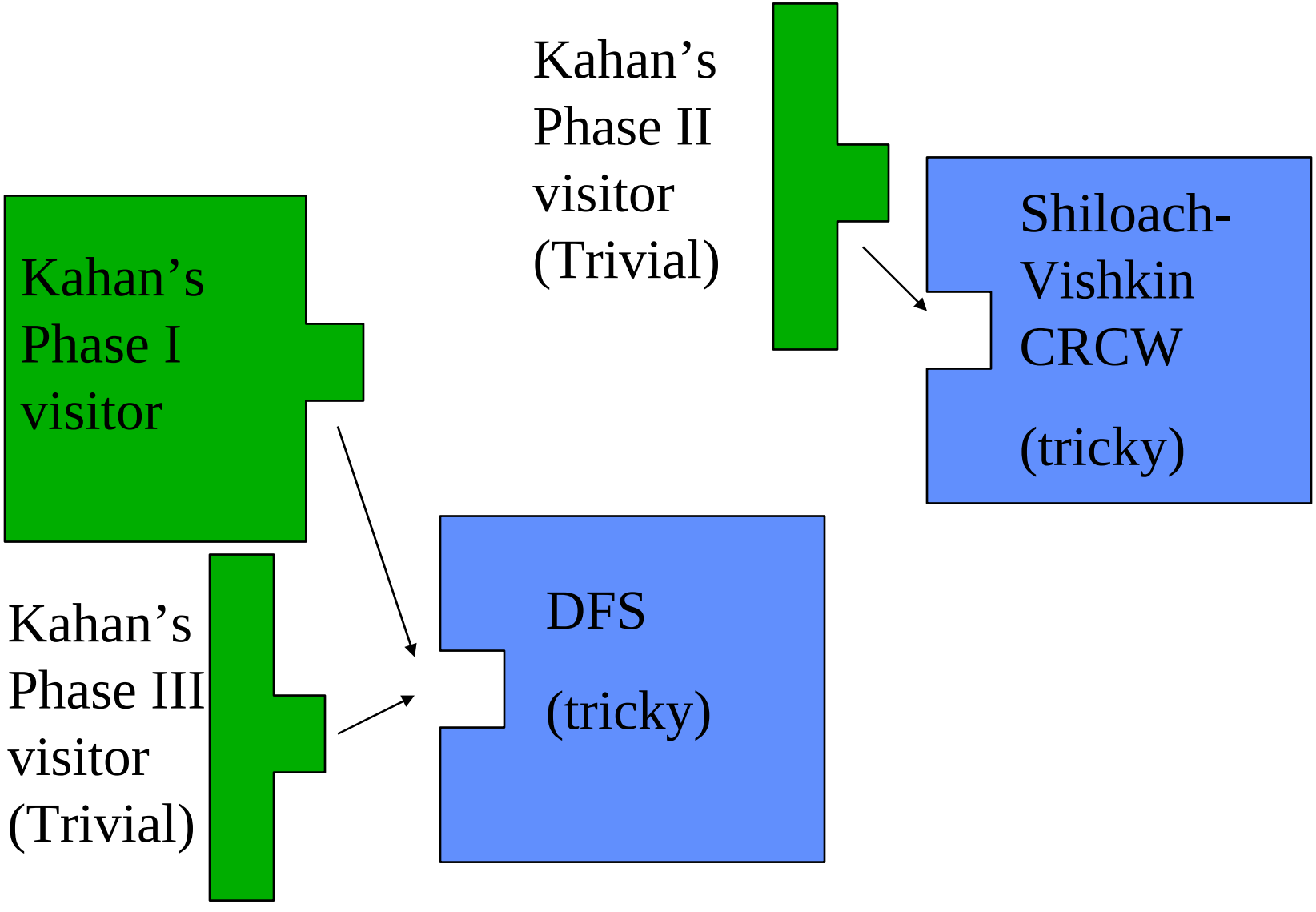
Kahan's Algorithm for Connected Components







Infrastructure Implementation of Kahan's Algorithm



Infrastructure Implementation of Kahan's Algorithm

Phase I:

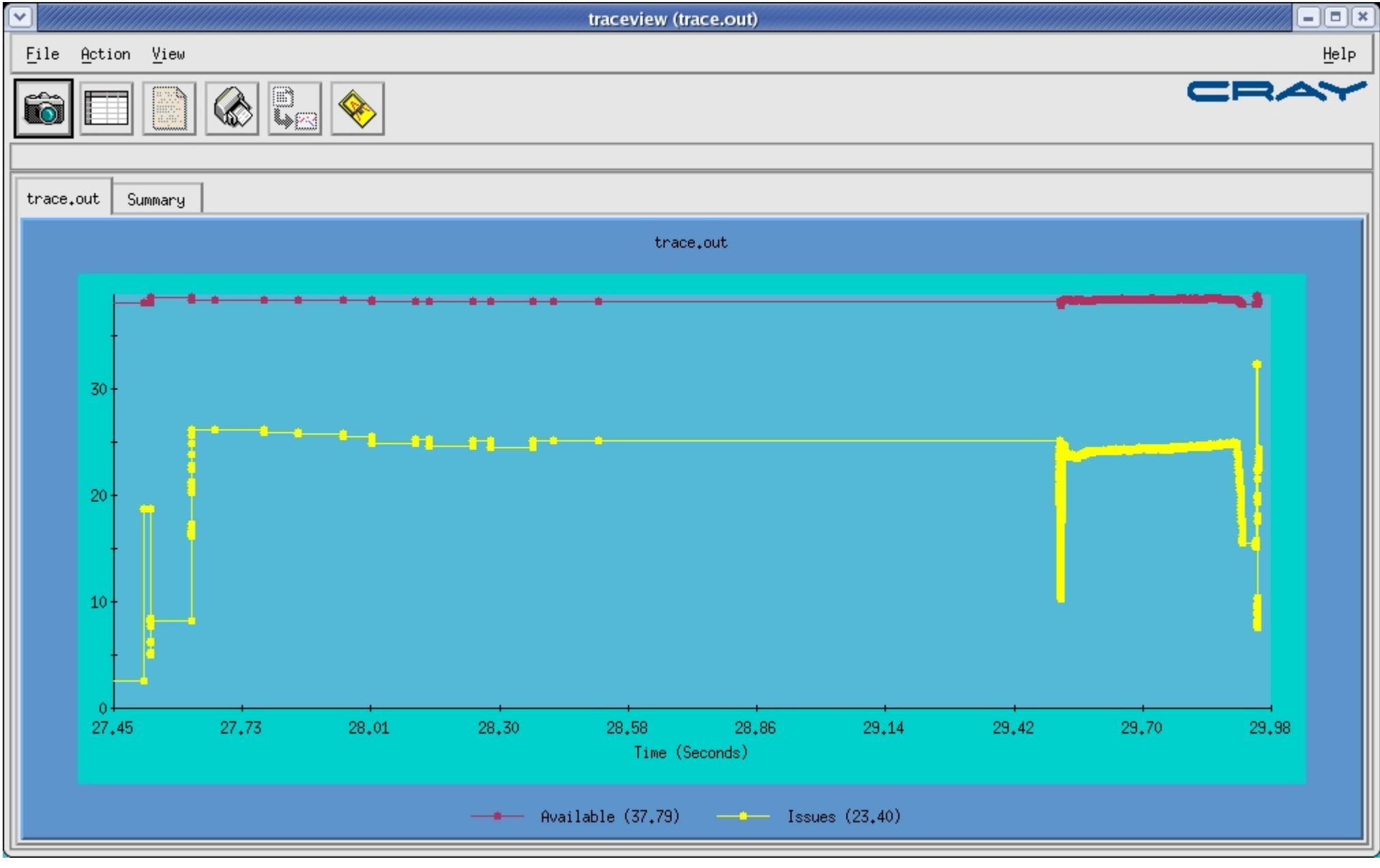
```
void tree_edge(Edge *e, Vertex *src) const
{
        #pragma mta trace "cc_dfs:tree_edge"
        Vertex *dest = e->other(src);
        mt_write(component[dest->id], component[src->id]);
        (void) mt_incr(count,1);
}
void back_edge(Edge *e, Vertex *src) const
{
        Vertex *dest = e->other(src);
        int v1=src->id, v2=dest->id;
        int c1 = mt_readff(component[v1]);
        int c2 = mt_readff(component[v2]);
        if (c1 != c2) {
                int mn = min(c1,c2);
                int mx = max(c1,c2);
                small_graph.insert(mn*g_order+mx,
                                   QuadSV(e->id,mn,mx,NULL),true);
        }
}
```

"component" values start "empty;" Make them "full."

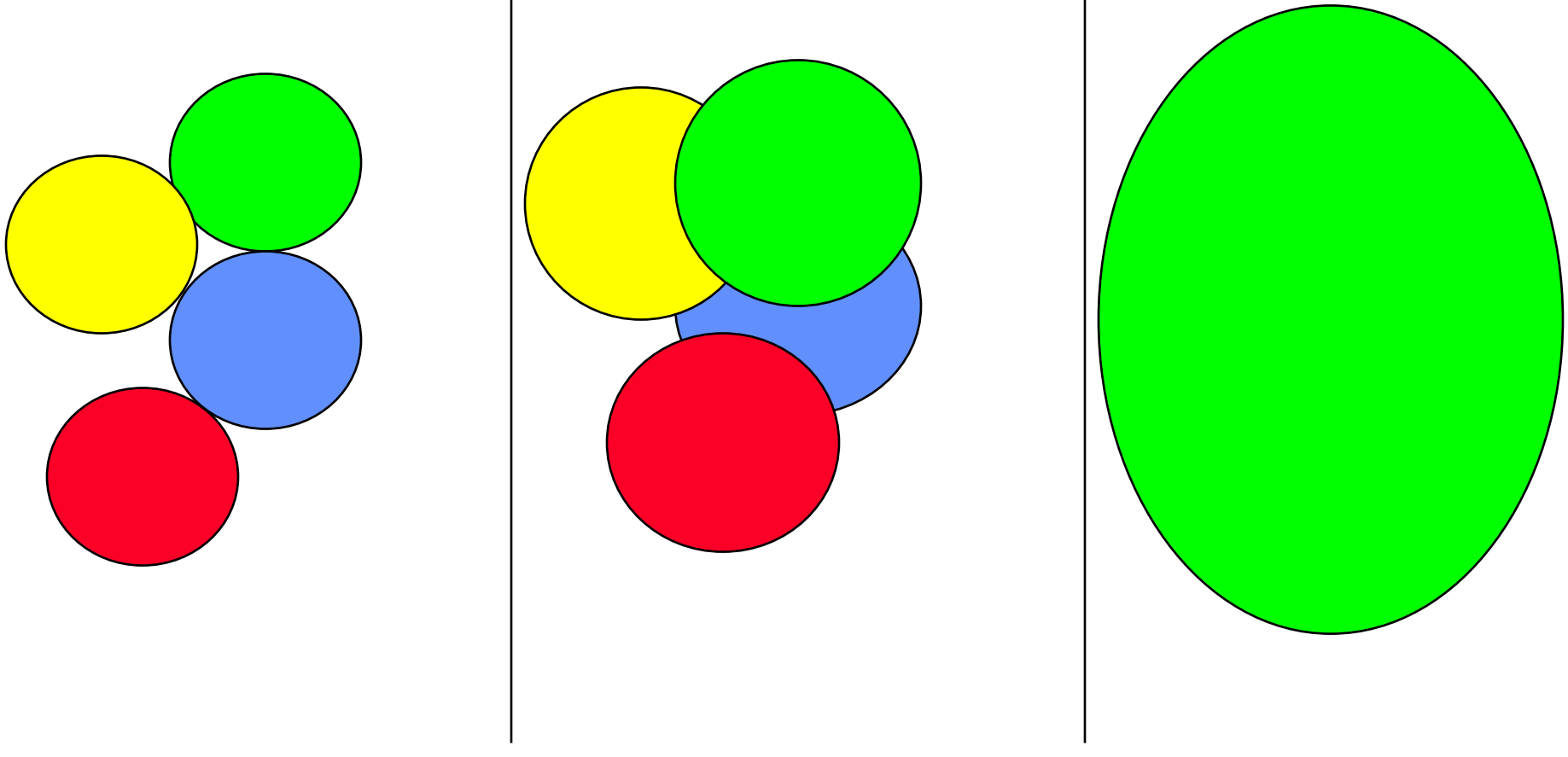
Wait until both "full,"

Add to hash table

Traceview Output for Infrastructure Impl. of Kahan's CC algorithm



More General Filtering: The “Bully” Algorithm



“Bully” Algorithm Implementation

```
bool visit_test(Edge *e, Vertex *src) const
{
        Vertex *v2 = e->other(src);
        if (post_dfs_visitor<ComponentMap>::component[src->id] <
            post_dfs_visitor<ComponentMap>::component[v2->id])
                return true;
        else    return false;
}
void tree_edge(Edge *e, Vertex *src) const
{
        Vertex *dest = e->other(src);
        int c = mt_readfe(post_dfs_visitor<
                          component[dest->i
        if (c==-1 || c > post_dfs_visitor<ComponentMap>::
                         component[src->id]) {
                post_dfs_visitor<ComponentMap>::component[dest->id] =
                post_dfs_visitor<ComponentMap>::component[src->id];
        } else {
                mt_write(post_dfs_visitor<ComponentMap>::
                         component[dest->id], c);
        }
}
```

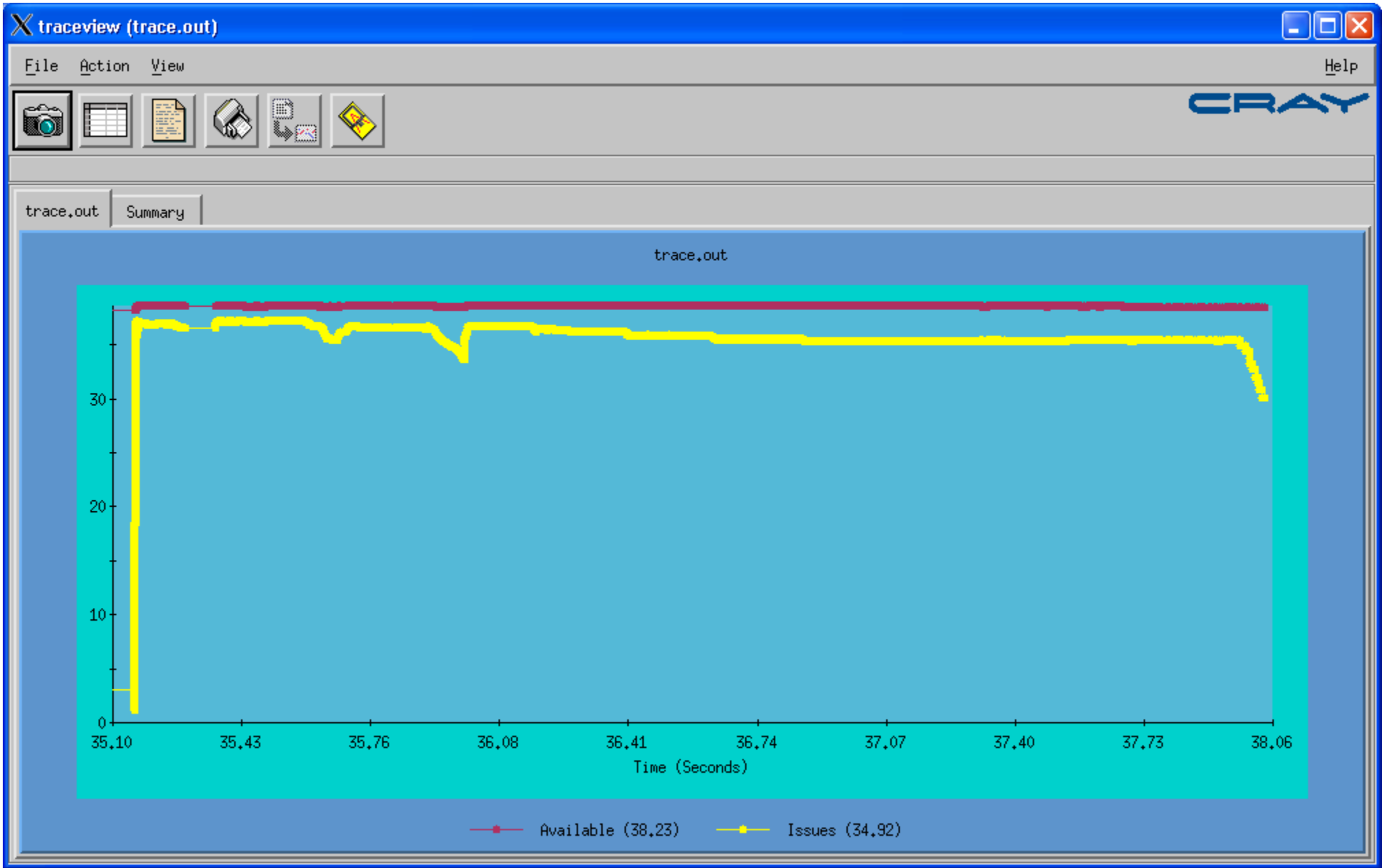
Traverse “e” if we would anyway, ***or*** if this test returns true

[***or,and,replace***]

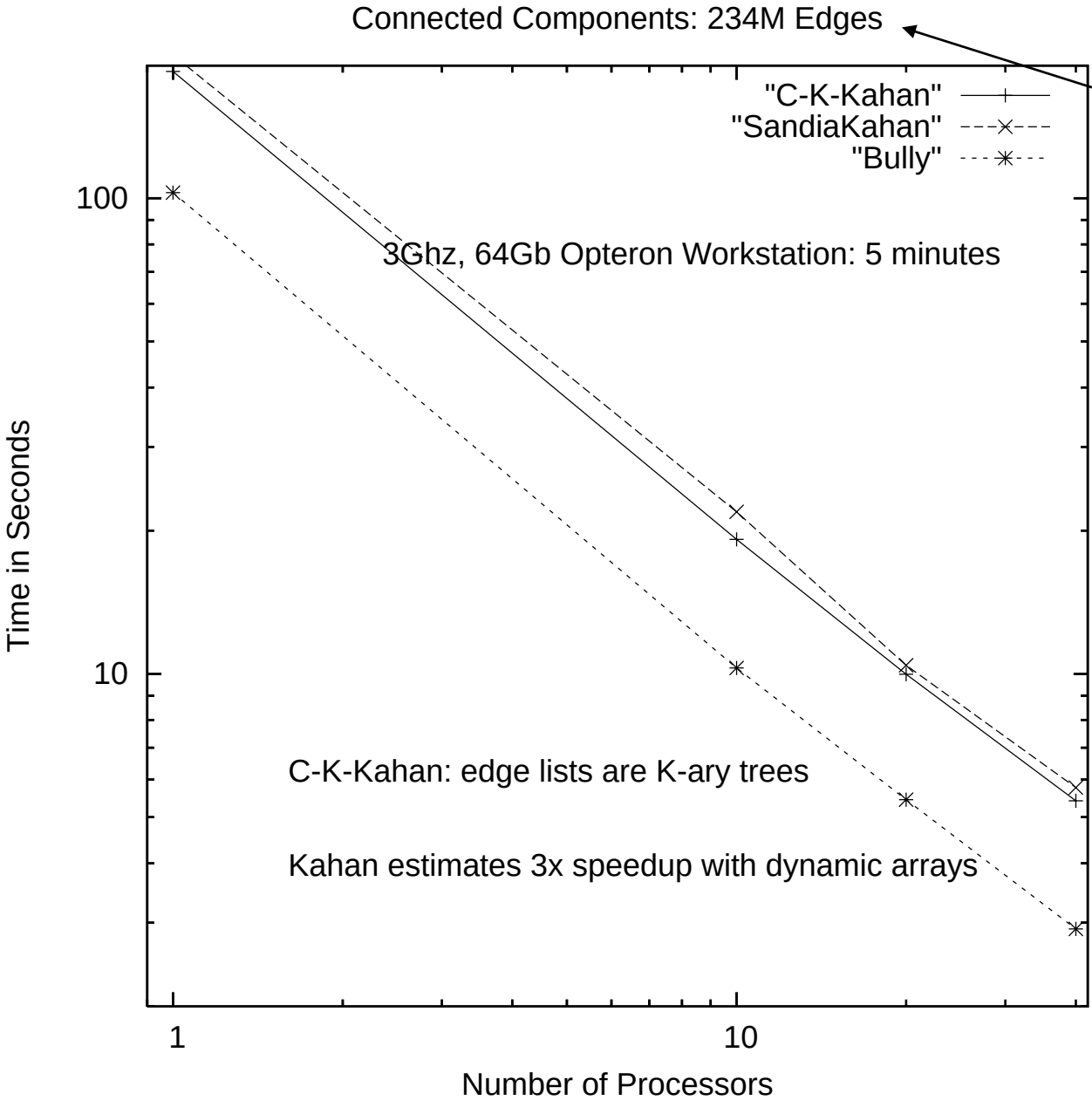
Lock dest while testing

Sandia National Laboratories

Traceview Output for the Bully Algorithm



MTA-2 Scaling of Connected Components



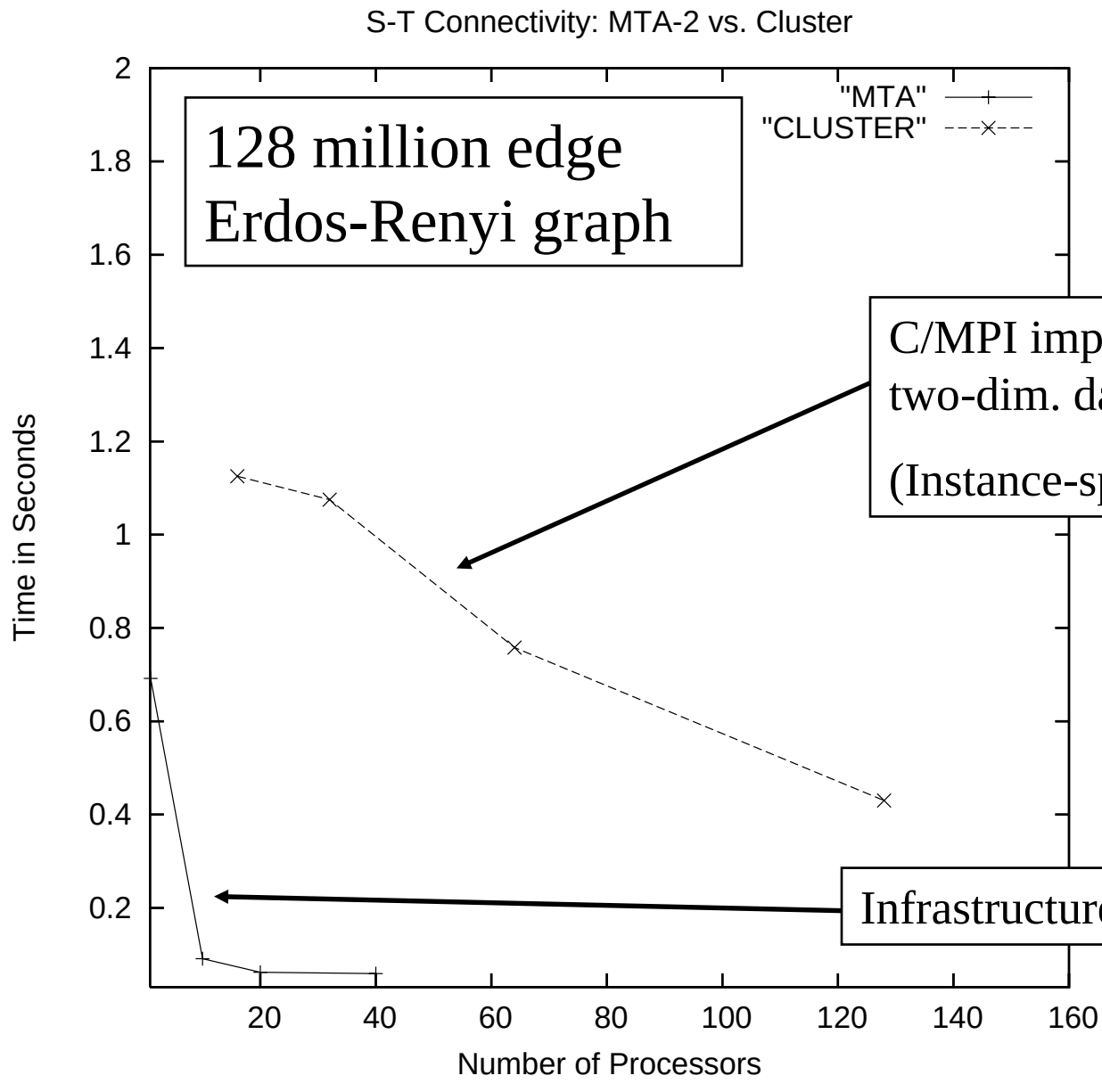
Power Law Graph (highly unstructured)

5.41s

2.91s

Sandia National Laboratories

Case Study: S-T Connectivity



128 million edge Erdos-Renyi graph

C/MPI implementation with two-dim. data decomposition

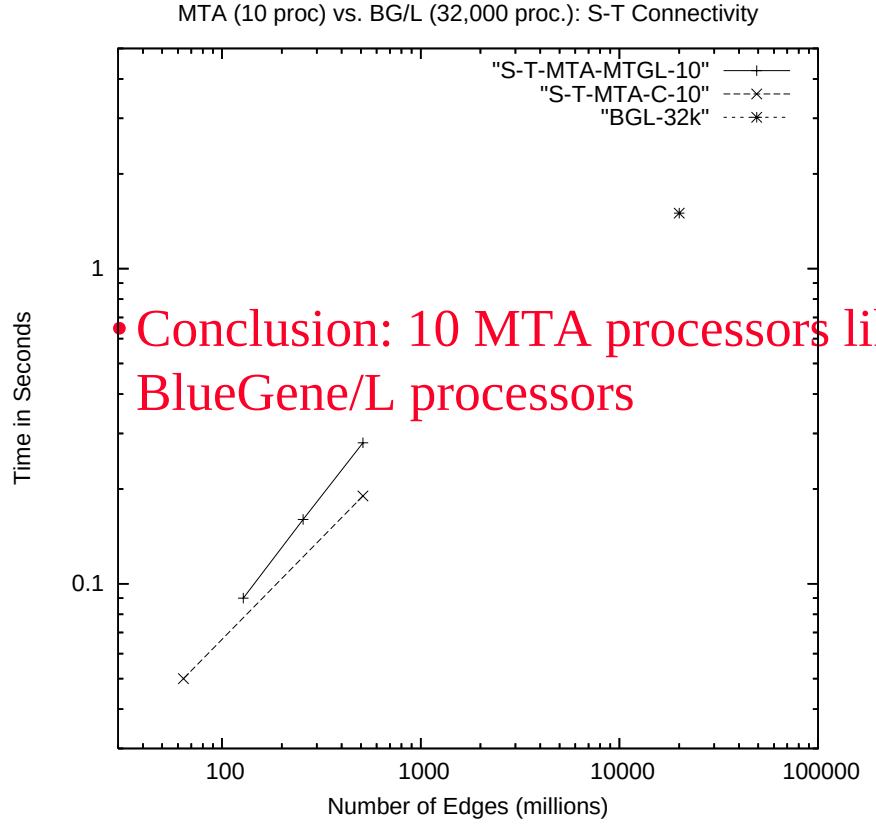
(Instance-specific)

Infrastructure impl. (general)

Sandia National Laboratories

S-T Connectivity, The MTA-2, and BG/L

- **IBM/LLNL BlueGene/L is considered fastest computer in the world**
- **With researchers at LLNL, Sandia implemented s-t shortest paths in MPI (same implementation of the previous slide)**
- **Finalist for 2005 Gordon Bell Prize**



- Conclusion: 10 MTA processors likely as fast as 32K BlueGene/L processors

Sandia National Laboratories

Successor to the MTA-2: “Eldorado” (2006)

- **Faster CPU clock rate**
- **Slower network**
- **Slower memory**
- **Locality matters**

Sandia work by Keith Underwood suggests that our codes are likely to scale on Eldorado as if it were a larger MTA-2 (up to ~500 processors).

What is next for our infrastructure?

- **OpenSource**
- **Add abstractions for partitioned global address space**
  - Run on SMPs, multi-core workstations
  - Codes developed will be closer to Eldorado and beyond
  - Distributed memory? Probably not -- Recall BG/L comparison
- **Support Applications**
  - Agent-Based Modeling
  - Graph Query
  - Branch and Bound
  - Heuristics

Conclusions

- **Massive multithreading with latency tolerance very attractive for unstructured graph applications**
  - Demonstrated potiential for high productivity
  - Excellent MTA-2 performance, scalability
  - Reason to be optimistic about Eldorado scalability

- **Graph infrastructure development promising; will continue**
  - Run same code on range of architectures – workstation to Eldorado
  - Extend for Important Applications

Acknowledgements

- **Bruce Hendrickson (Project lead)**
- **Simon Kahan, Petr Konecny (Cray): help in all aspects of this project**
- **David Bader, Kamesh Madduri (Ga. Tech) (MTA s-t connectivity)**
- **Will McClendon (MPI s-t connectivity)**

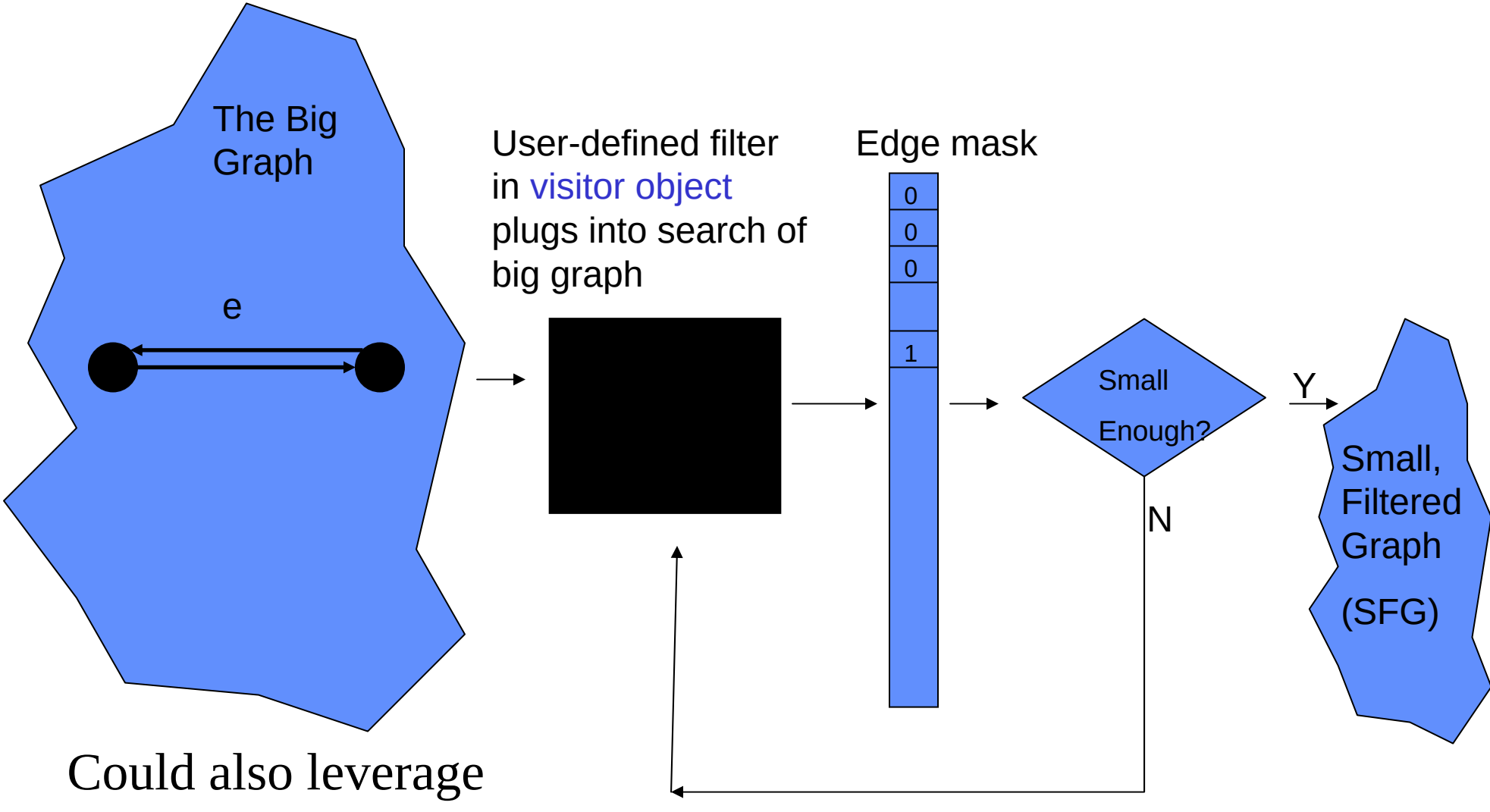
Sandia National Laboratories

Extra Slides

Case Study: Subgraph Isomorphism Kernel

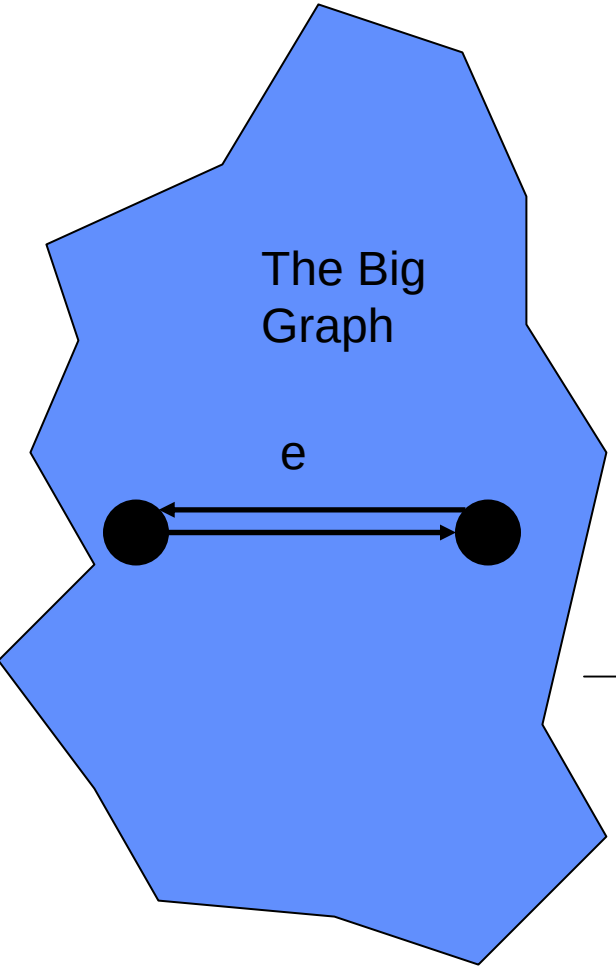
- **Objective: find exact or inexact matches of a small pattern graph within a large semantic graph**
- **Potentially useful for finding instances of interesting activities in a large dataset**

Preprocessing with “Black Box” Filtering

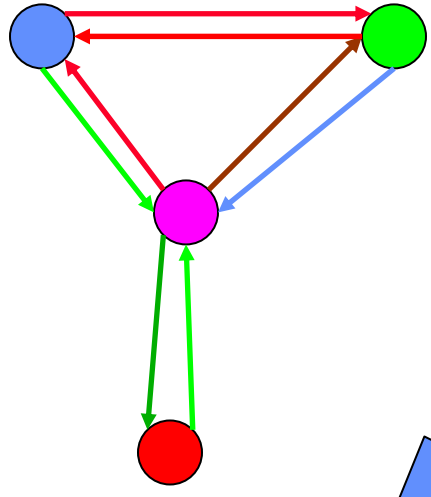


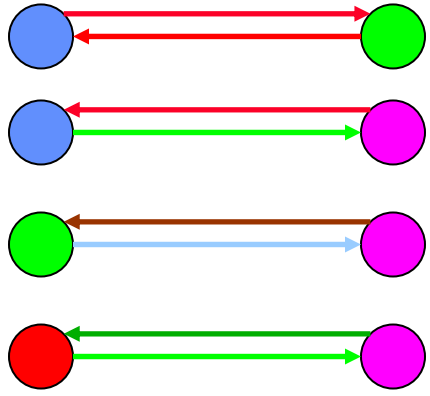
Could also leverage pre-existing filters

Instance-Specific Type Filtering for Subgraph Iso.



Pattern graph →





For each edge e in Big Graph:

Do the 4 types associated with e match those of any of the target graph edges?

Small, Filtered Graph (SFG)

Sandia National Laboratories

Subgraph Isomorphism: Input

The Target Graph:

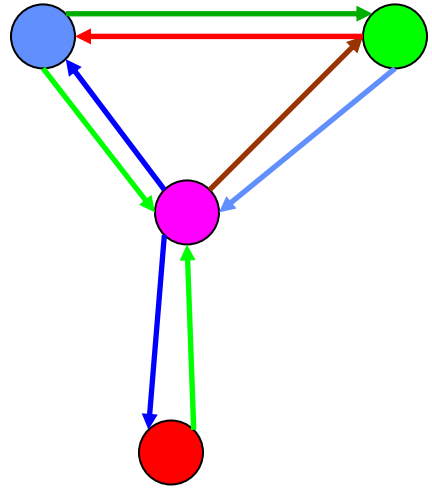


Table of Type and Auxiliary Information:

T																
V	2		2		3		2		3		1		3		2		2

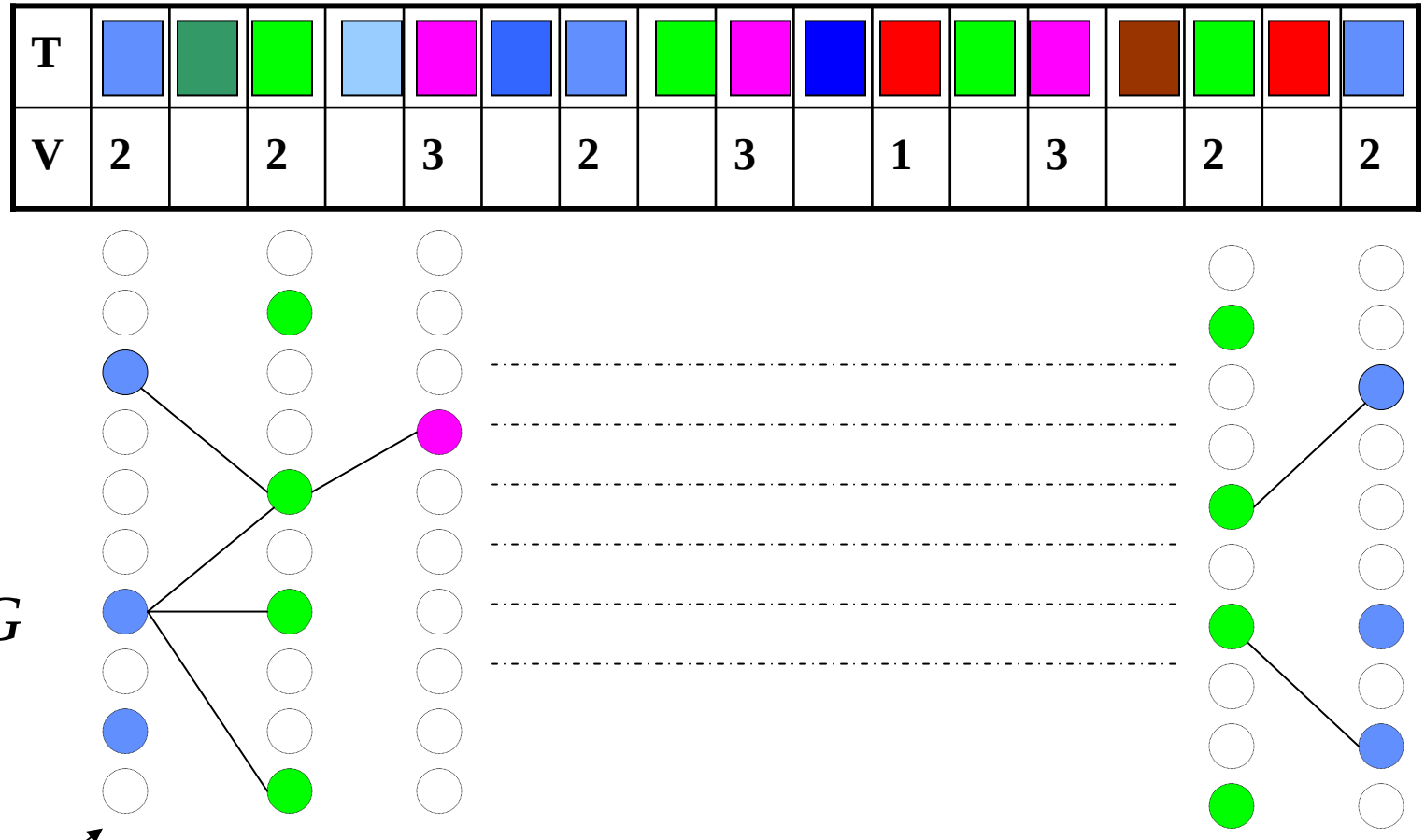
Ideal: Euler Tour

Our Experiments: Random Walk

Subgraph Isomorphism: Creating a Bipartite Graph

Small, Filtered Graph (SFG)

k times:
visit each edge of SFG

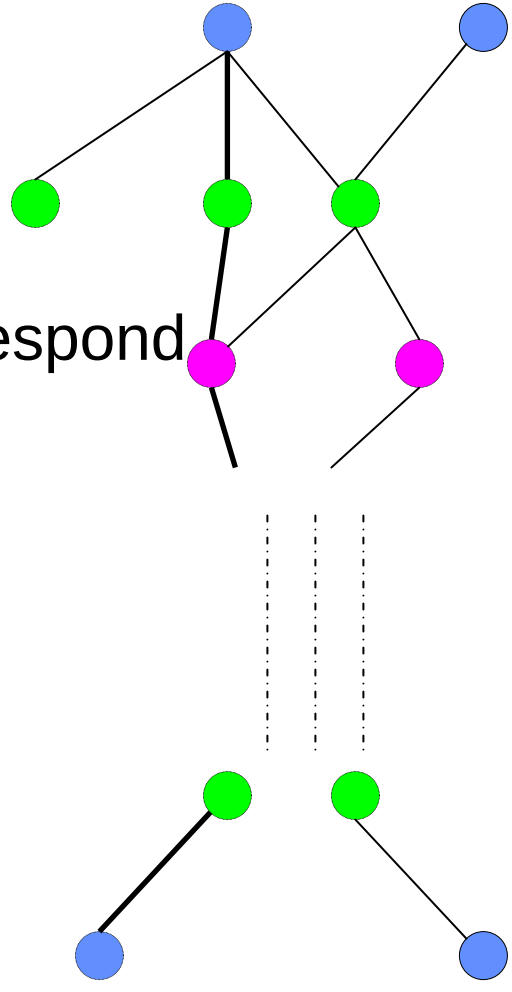


Logical placeholders for vertices in the SFG.

Subgraph Isomorphism: Creating a Bipartite Graph

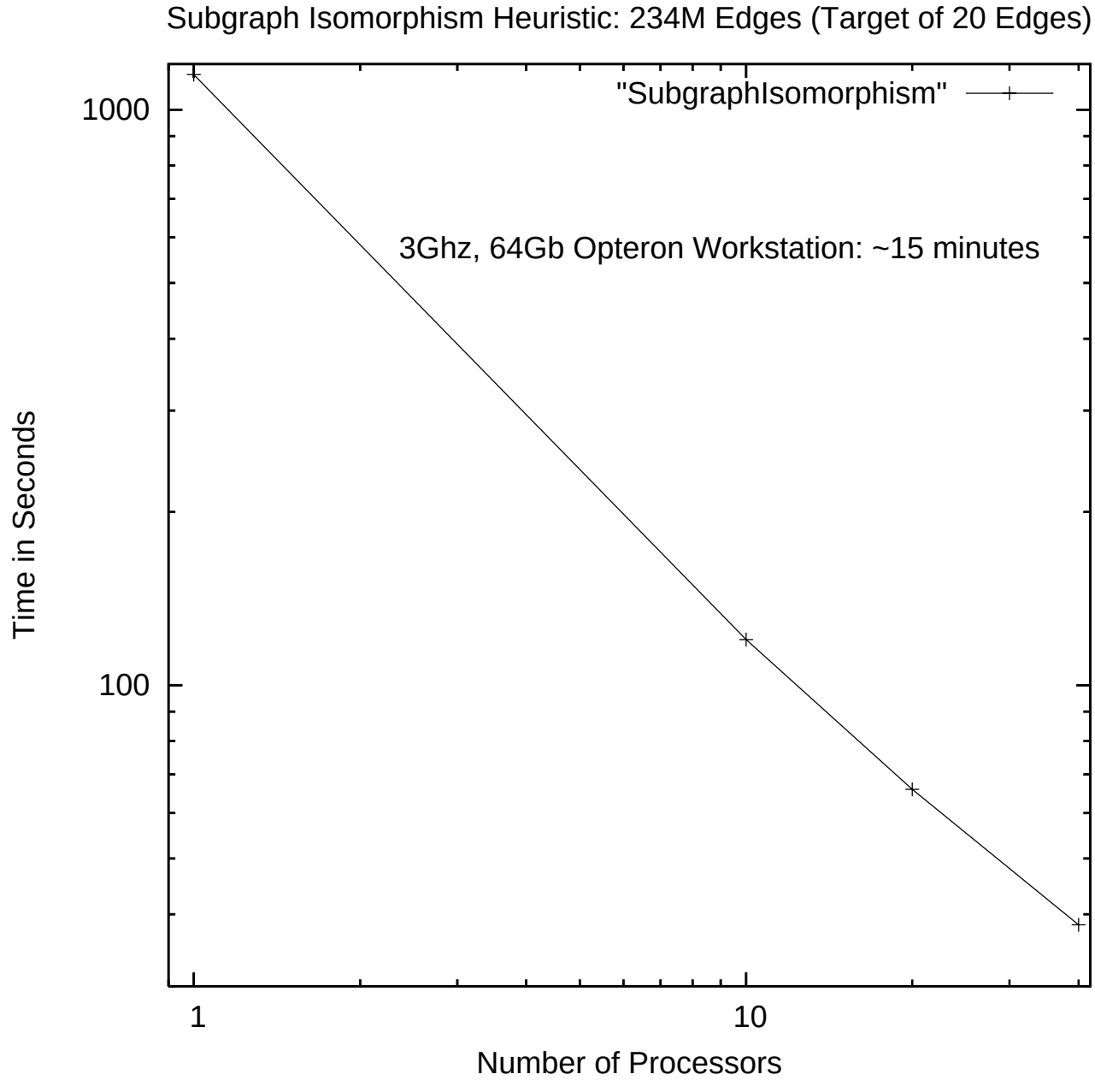
S-T shortest paths (top to bottom) correspond to candidate matches.

Branch and bound to Find better matches.



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

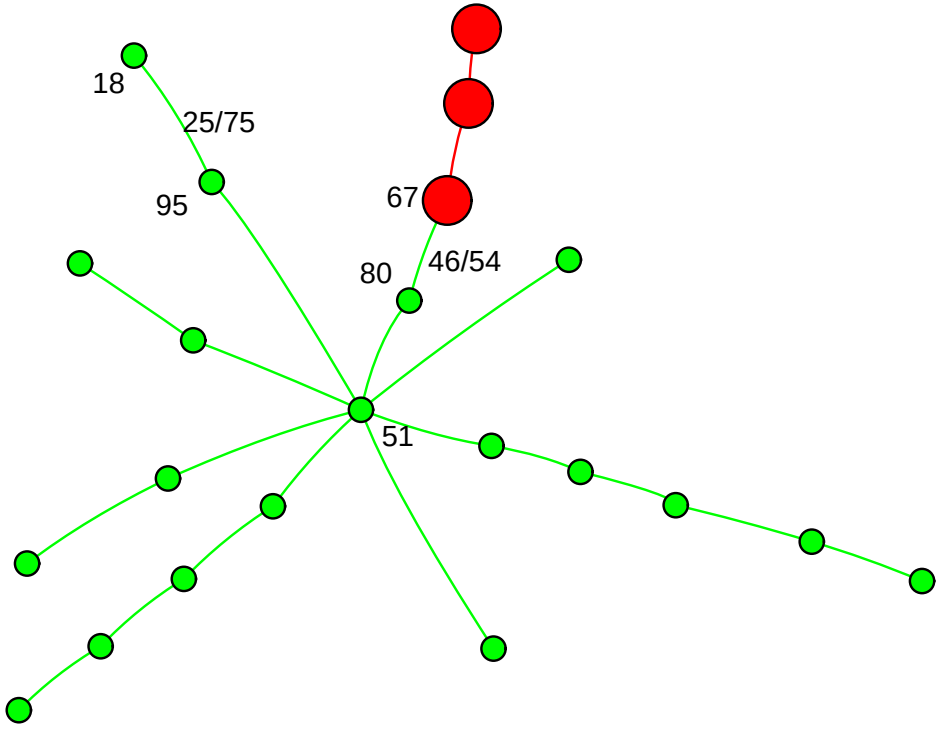
Computational Results: Subgraph Isomorphism



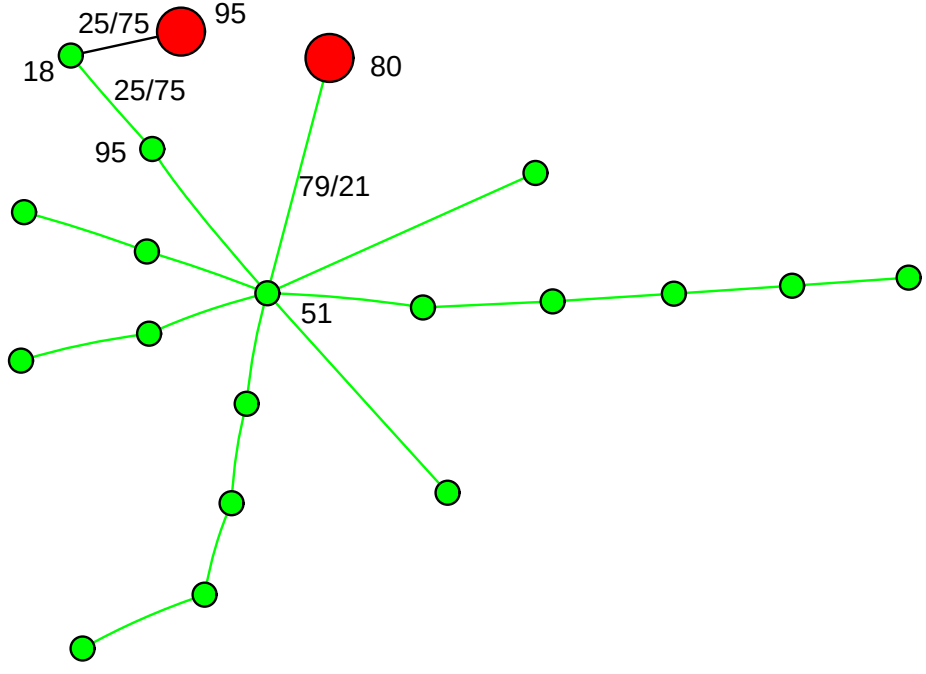
Sandia National Laboratories

Computational Results: Subgraph Isomorphism

Type & topological isomorphism exists between green vertices



Target



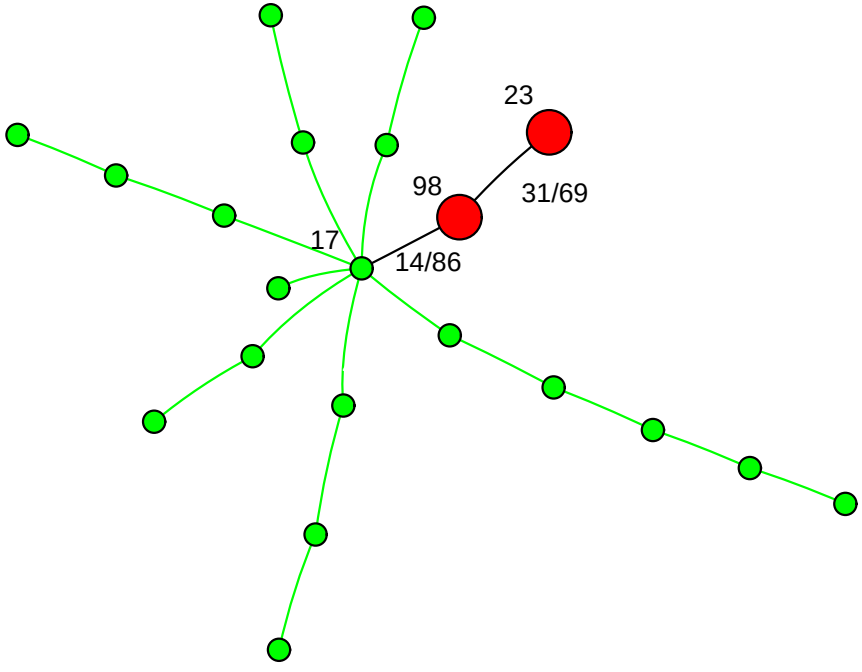
Found

Actual graphs from a 234M edge instance

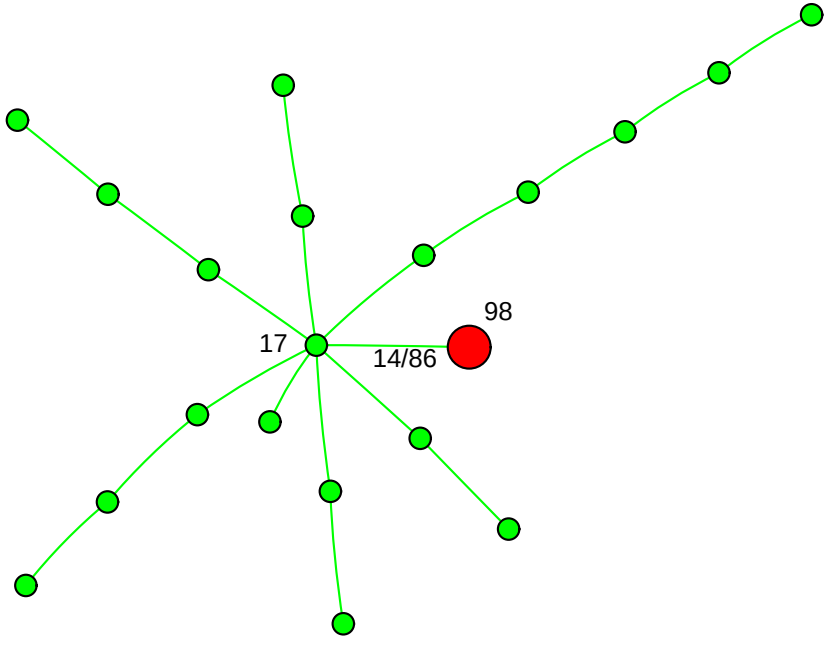
Sandia National Laboratories

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