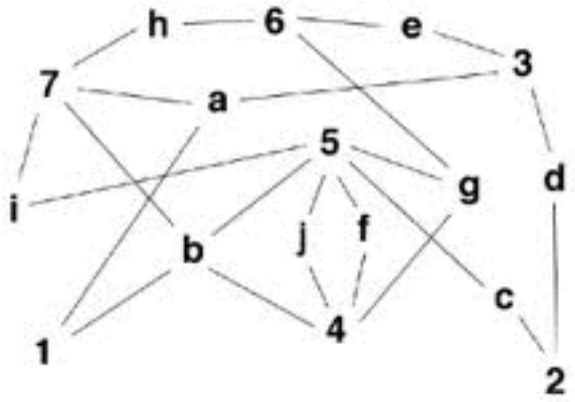
IO-efficient Computations

Benjamin Schiller, P2P Networks, TU Darmstadt

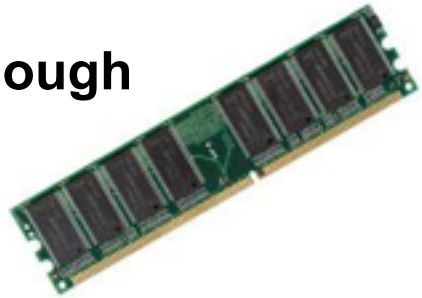
Research Meeting / 30.10.2013

Problem & Goals



Large

not enough



- Goal
  - Enabling analysis with small memory requirements
  - Current example: listing all triangles
    - Directed / undirected
    - Computing LCC / T / CC

Solution Space & Related Work

- Decrease problem (graph) size
  - Simplification, sampling, graph coarsening
  - Sampling-based approaches (…), DOULION (Tsourakakis et al.)

- Stream-based analysis
  - Process data as stream, forget afterwards
  - Various algorithms exist (single or multi) (…)

- Spatial division
  - Solve problem for sub-graphs (iteratively / in parallel)
  - In-memory triangle listing (Chu & Cheng)

- Computational division
  - MapReduce, Hadoop
  - Specialized algorithms & data structures (…)

Basic Triangle Listing Algorithm

- Listing local triangles (directed or undirected) of v

Algorithm 1 *In-Memory Triangle Listing*

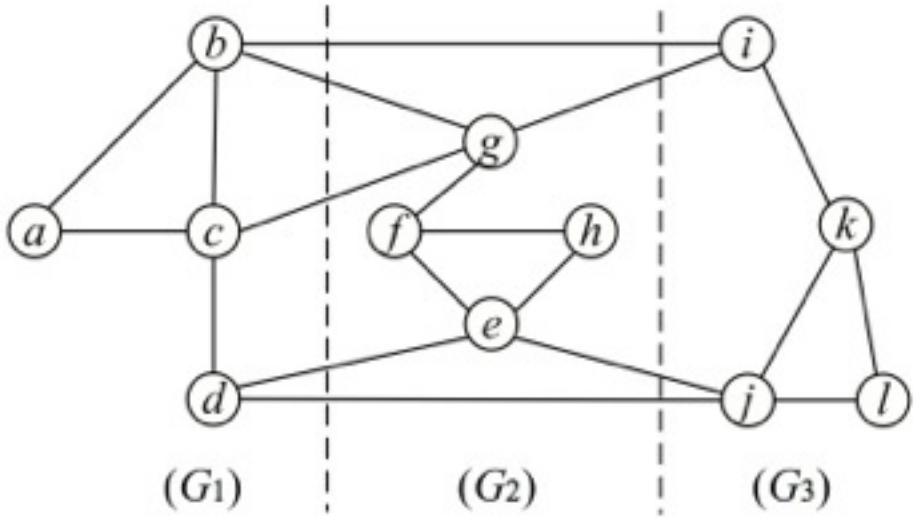
Input: A graph $G = (V_G, E_G)$
Output: $\triangle(G)$

1. $\triangle(G) \leftarrow \emptyset$;
2. **for each** $u \in V_G$ **do**
3. **for each** $v \in adj_G(u)$, where $v > u$, **do**
4. **for each** $w \in (adj_G(u) \cap adj_G(v))$, where $w > v$, **do**
5. $\triangle(G) \leftarrow (\triangle(G) \cup \{\triangle_{uvw}\})$;
6. **return** $\triangle(G)$;

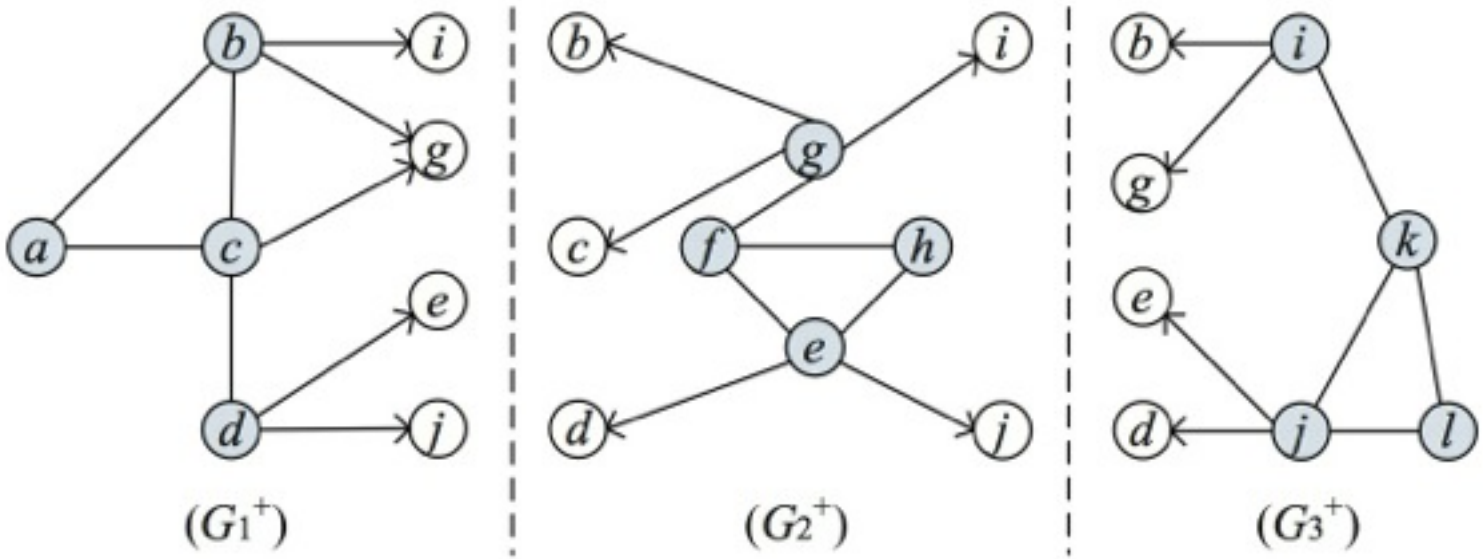
LCC

- Requires adjacency information of all neighbors N(v)

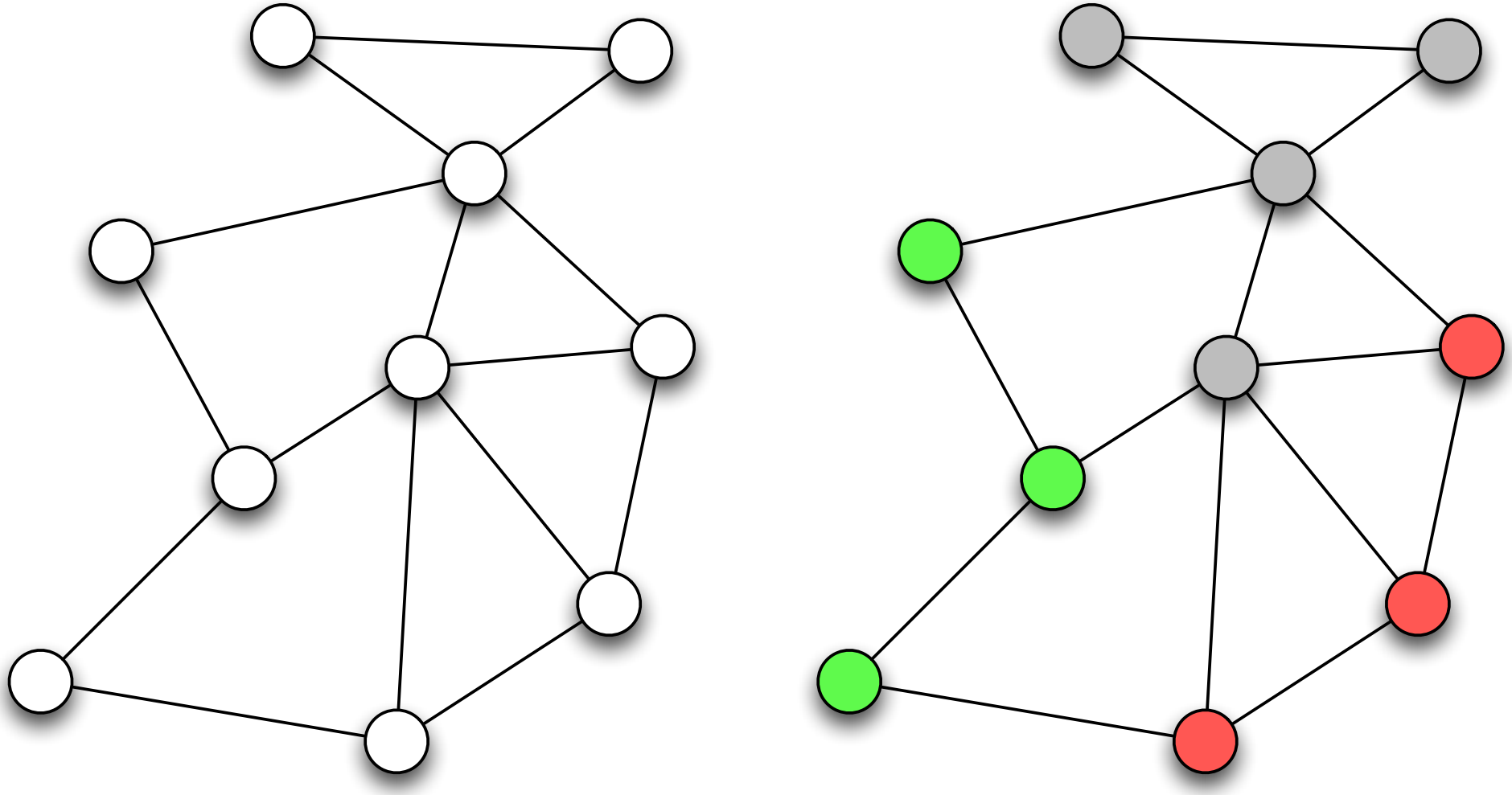
In-memory Triangle Listing (1/6)



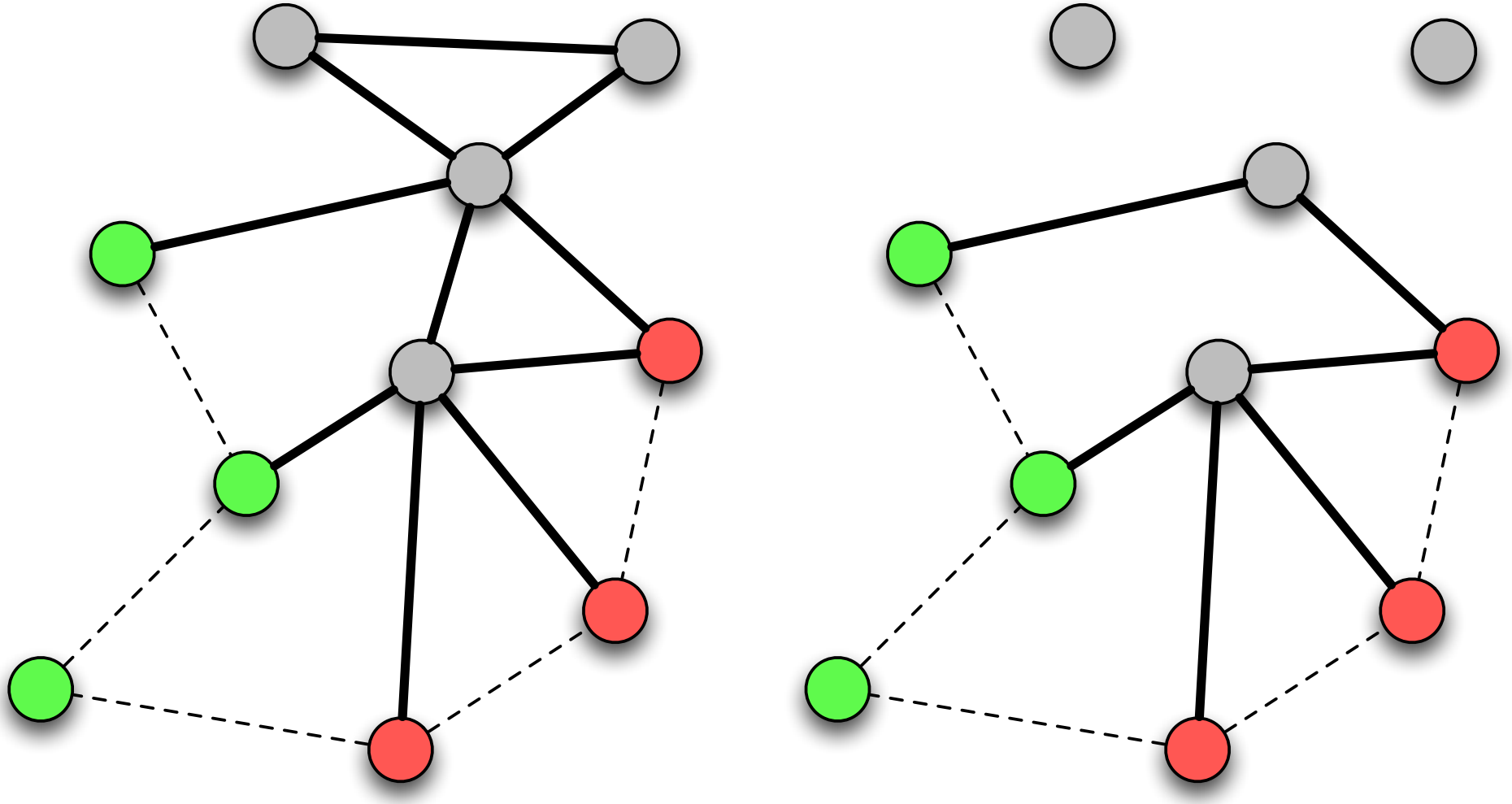
Related Work



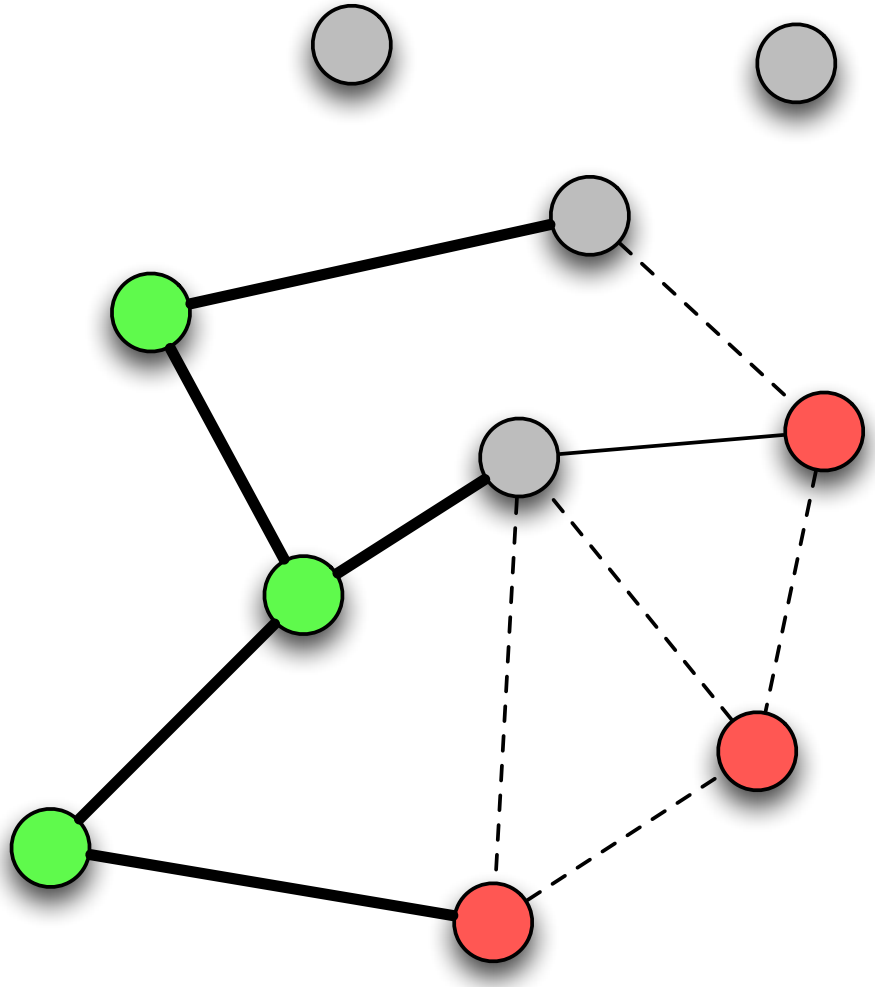
In-memory Triangle Listing (2/6)

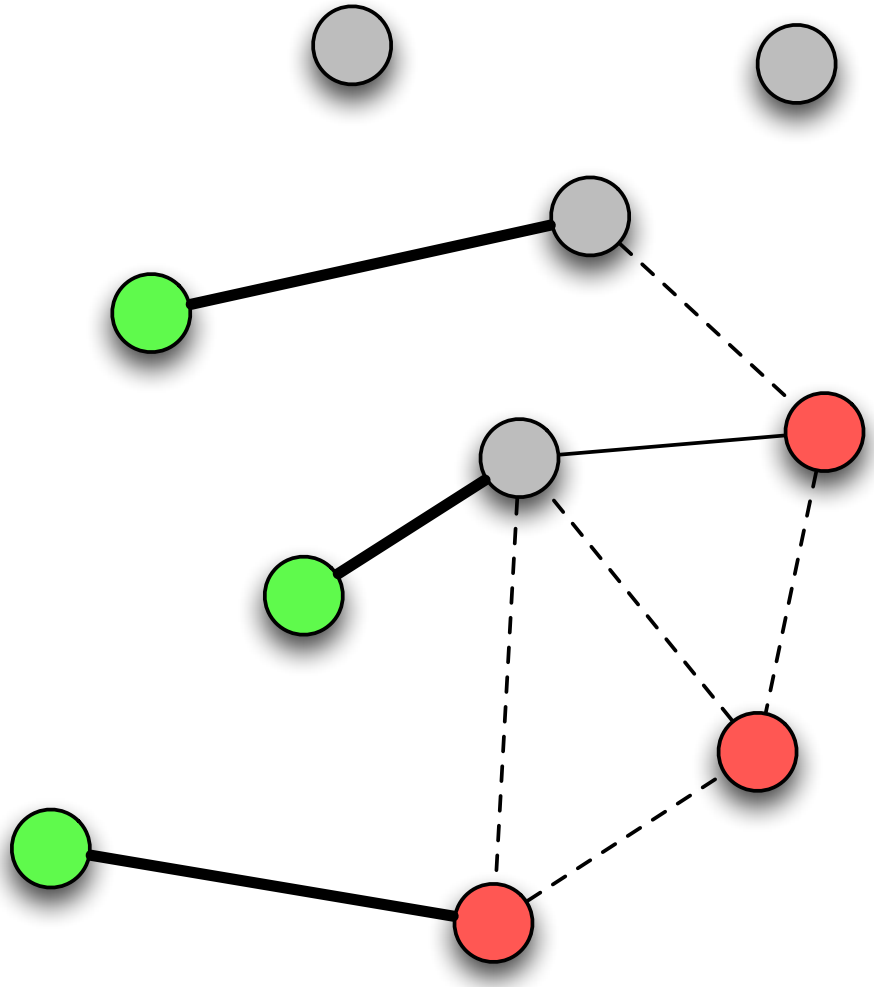


In-memory Triangle Listing (3/6)

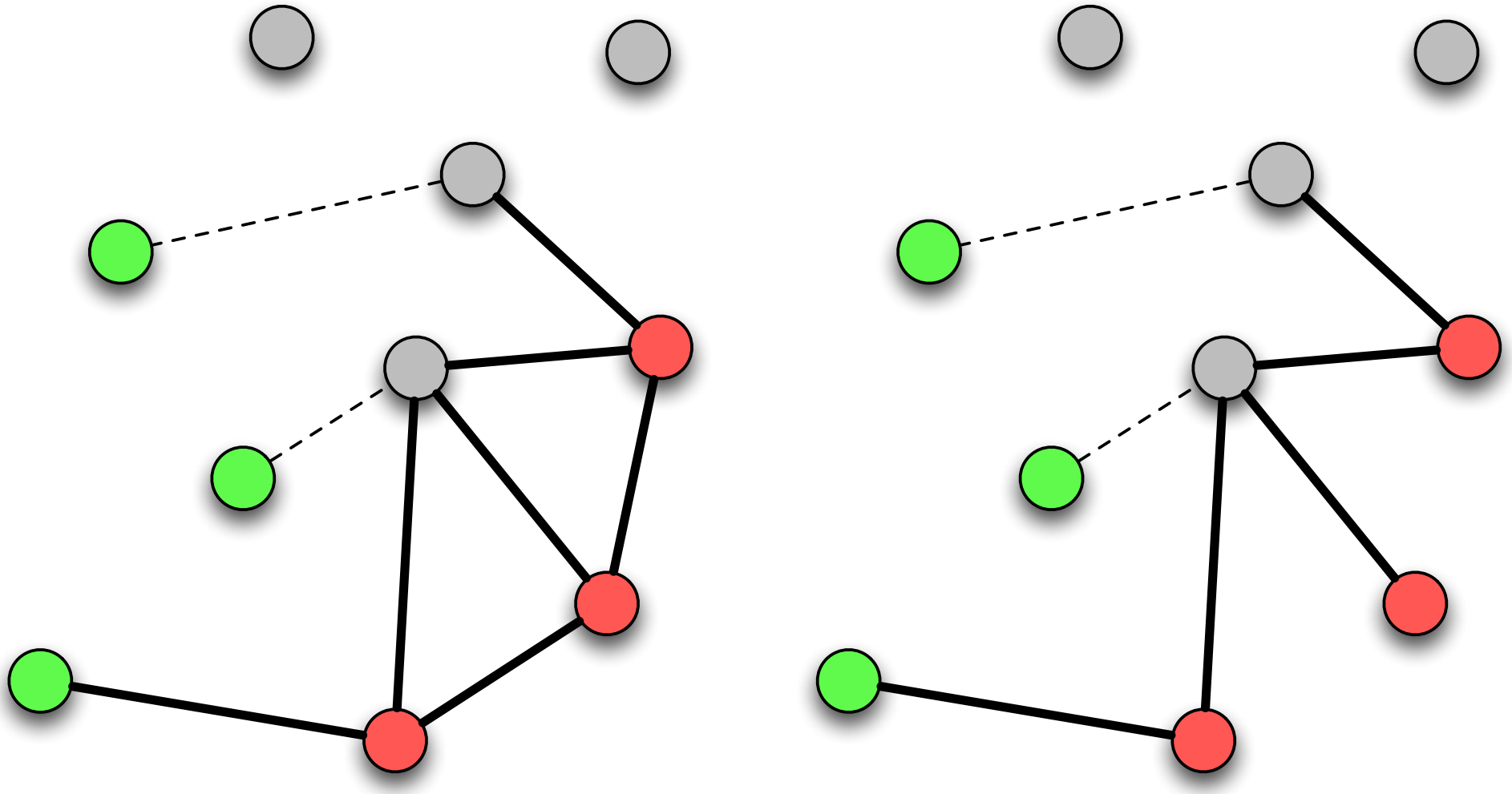


In-memory Triangle Listing (4/6)



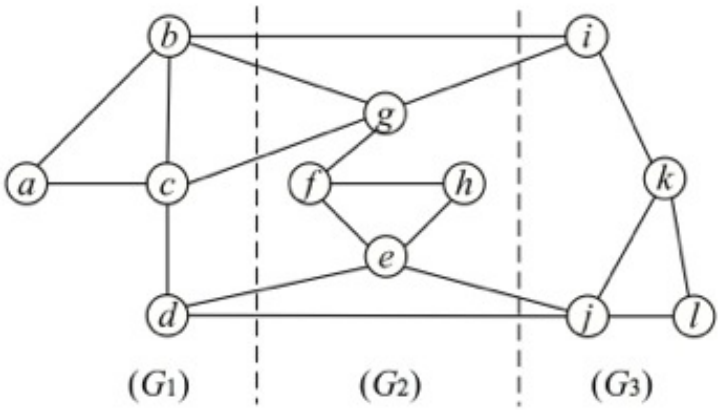


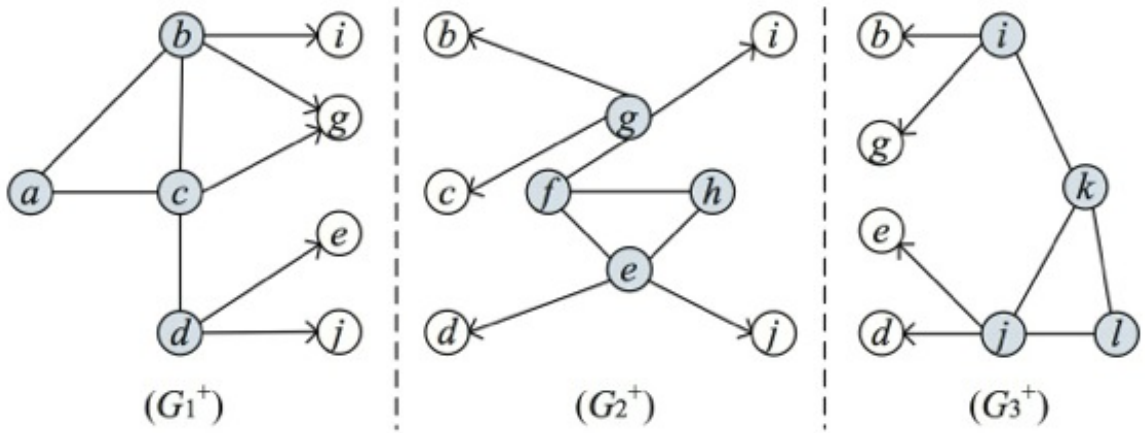
In-memory Triangle Listing (5/6)



In-memory Triangle Listing (6/6)

- Partitioning strategies
  - SEQ - Sequential - pre-defined order
  - DOM - Dominating set - attempt to get highly connected components
- Problems
  - Read complete graph for computation of dominating sets
  - Write each node after each round to FS (delete edges)

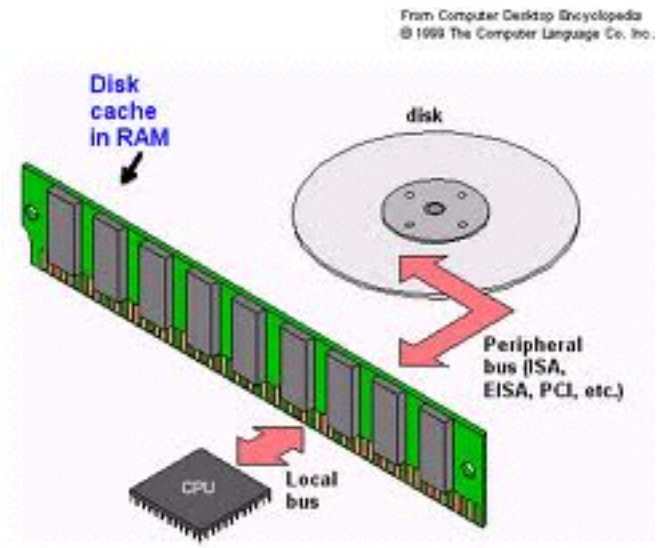


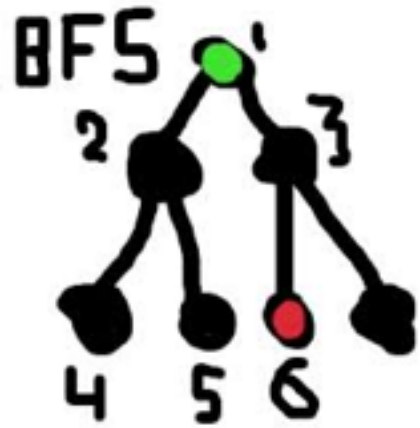


Our Approach

- Process each node separately (LCC)
- Cache read nodes for future computations
- Two decisions to make:

1. Which caching strategy?
   - LRU, FIFO, LFU
   - Weighted, Random
2. Order to process the nodes?
   - BFS, BFS*, DFS, DFS*
   - BFS-Bubble, BFS-Bubble*
   - Degree, Random

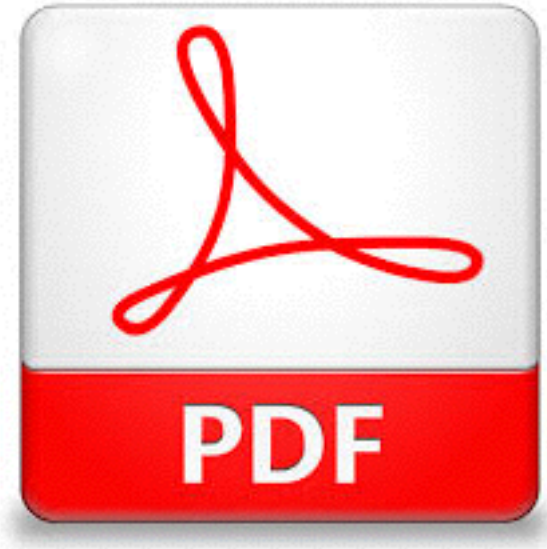




Evaluation Setup

- Per snapshot
  - Varying memory size (<<, <, >)
  - Repeating 1-5 times
  - 8 to 40 combinations + SEQ + DOM
- Environment
  - crawler10, crawler11, crawler12
  - p2pram (with ramdisk)
- Statistics
  - Computation time
  - Our approach: hit rate
    - Pre-processing time
  - Related work: iterations & subgraphs
    - Partition time, counting time, delete time

Results



Related Work - Results

Table 2: Datasets

	LJ	USRD	WebUK	BTC
$\|V_G\|$	4.8M	24M	106M	165M
$\|E_G\|$	69M	58M	1,877M	773M
Disk size	809.1MB	969.6MB	20.3GB	10.0GB

Table 4: Running time (wall-clock time in seconds)

	LJ	USRD	BTC	WebUK
TL-Seq	29.63	6.24	350	2411
TL-DS	29.43	12.46	412	2503
In-mem	32.98	6.68	N.A.	N.A.
Semi-stream ($\overline{\sigma}(N_{\triangle}(v)) \approx 0.8$)	306	321	3402	7032
Semi-stream ($\overline{\sigma}(N_{\triangle}(v)) \approx 0.5$)	1275	1683	13711	34722

Summary

- LRU best caching strategy
  - Independent of processing order
- Faster than related work
  - All G+ snapshots
  - Most models
- Published runtimes of related work
  - Appear very strange
- p2pram?
  - Same differences between approaches
  - Can be used for further analysis!

Paper Outline

- Abstract, Introduction
- Background
  - graphs, d/u triangles, cc/t/lcc, basic algorithm(s) (contains vs. cut)
- Related work
  - Describe other approach / list similar works
- Our approach
  - Main idea & rationale behind it
  - Caching strategies (CS), processing orders (PO)
- Evaluation
  - Setup (datasets, runs, what measured)
  - Results (comparison of CS & PO, comparison with related work)
- Summary, conclusion, & outlook
  - Further CS & PO, other datasets