

# CACHE-GUIDED SCHEDULING

## EXPLOITING CACHES TO MAXIMIZE LOCALITY IN GRAPH PROCESSING

*Anurag Mukkara, Nathan Beckmann, Daniel Sanchez*



Carnegie Mellon University  
School of Computer Science

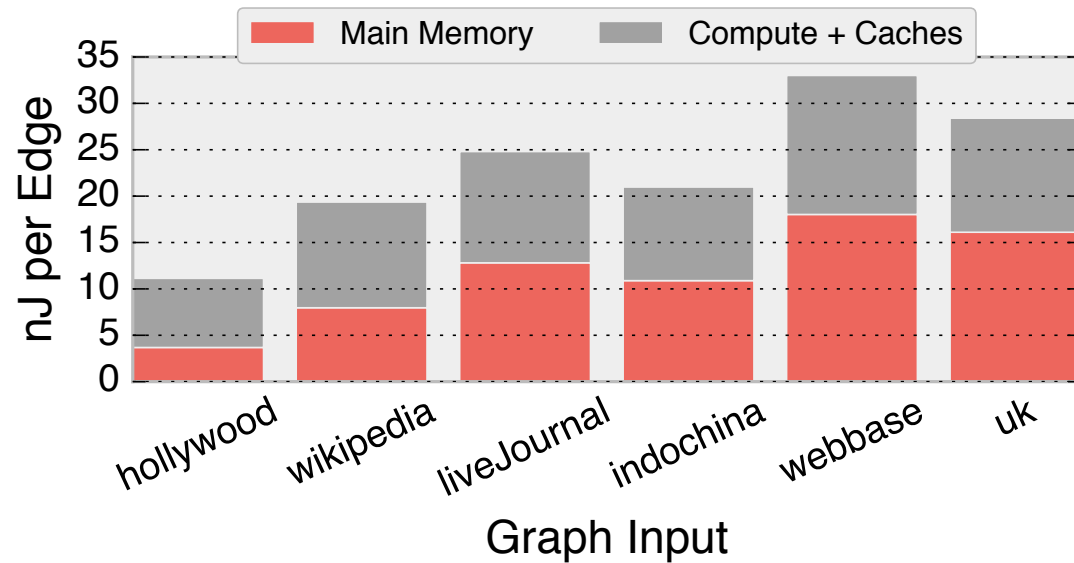
1<sup>st</sup> AGP – Toronto, Ontario – 24 June 2017

# Graph processing is memory-bound

- Irregular structure causes seemingly random memory references
- On-chip caches are too small to fit most real-world graphs

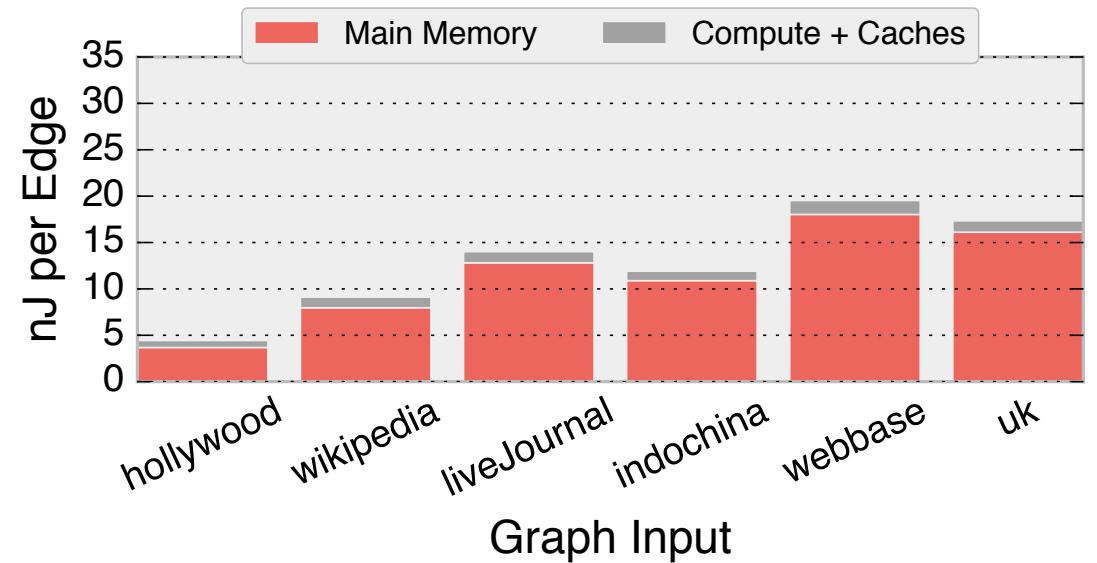
## PageRank

### General-purpose system



50% of system energy is due to main-memory

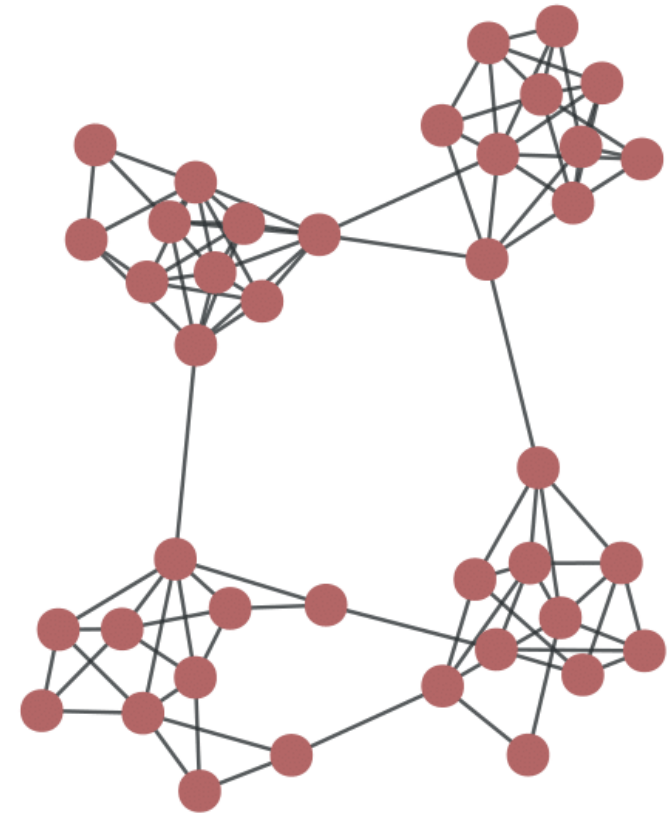
### Specialized accelerator



Memory bottleneck becomes more critical

# Exploiting graph structure through caches

- Real-world graphs have strong community structure
  - ▣ Significant **potential locality**
  - ▣ **Difficult to predict** ahead of time
- Idea: Let the **cache guide scheduling!**
  - ▣ Cache has information about the right vertices to process next – those which cause fewest misses
- This work: A limit study on the benefits of cache-guided scheduling (CGS)
  - ▣ CGS **reduces misses by up to 6x**



# Impact of Scheduling on Locality

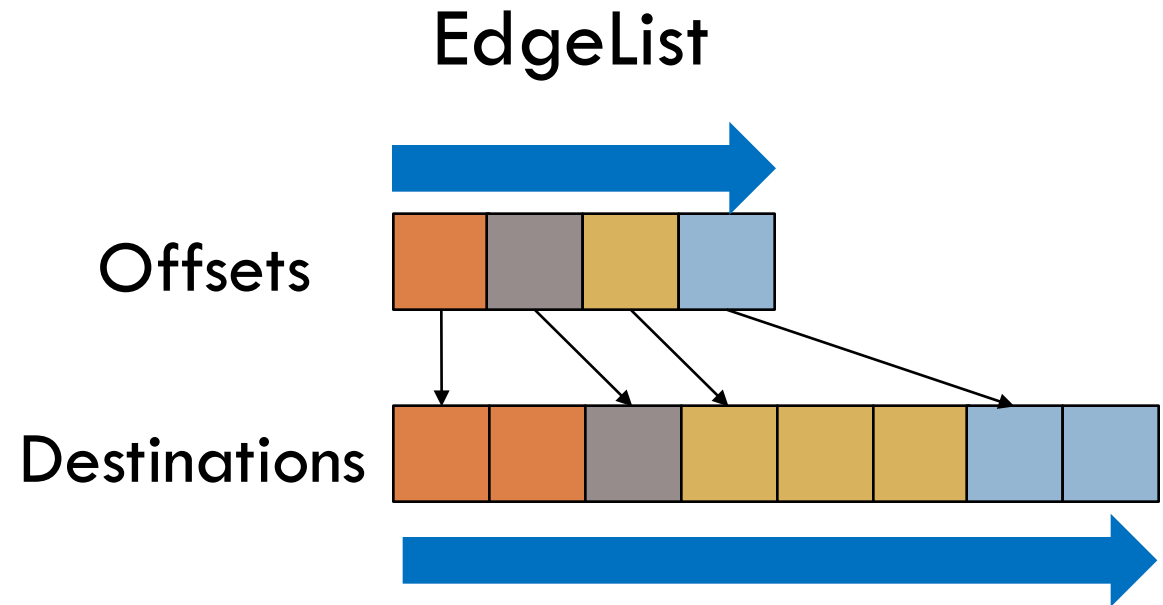
# Many graph algorithms allow flexibility in schedule 5

---

- **Schedule:** Order in which vertices of the graph are processed
  
- Many important algorithms are unordered – schedule does not affect correctness
  - ▣ Ex. PageRank, Collaborative Filtering, Label Propagation, Triangle Counting
  
- **Schedule impacts locality significantly**

# Vertex-ordered schedule follows layout order

- Vertices are processed in the order of their id
- All edges of a vertex are processed consecutively
- Used by state-of-the-art graph processing frameworks
  - ▣ Ligra, GraphMat, etc.
- Simplifies scheduling and parallelism
- Poor locality

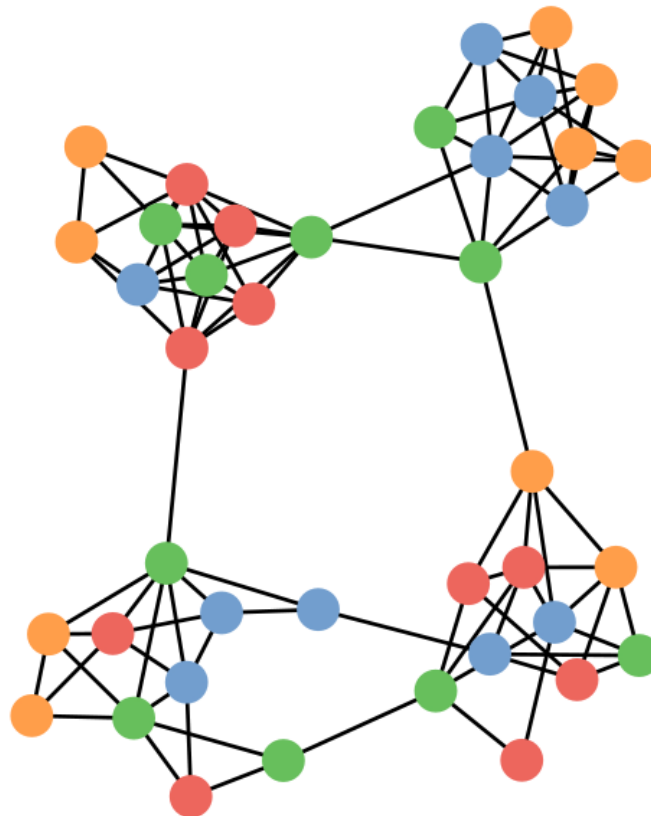


# Layout order might not match community structure

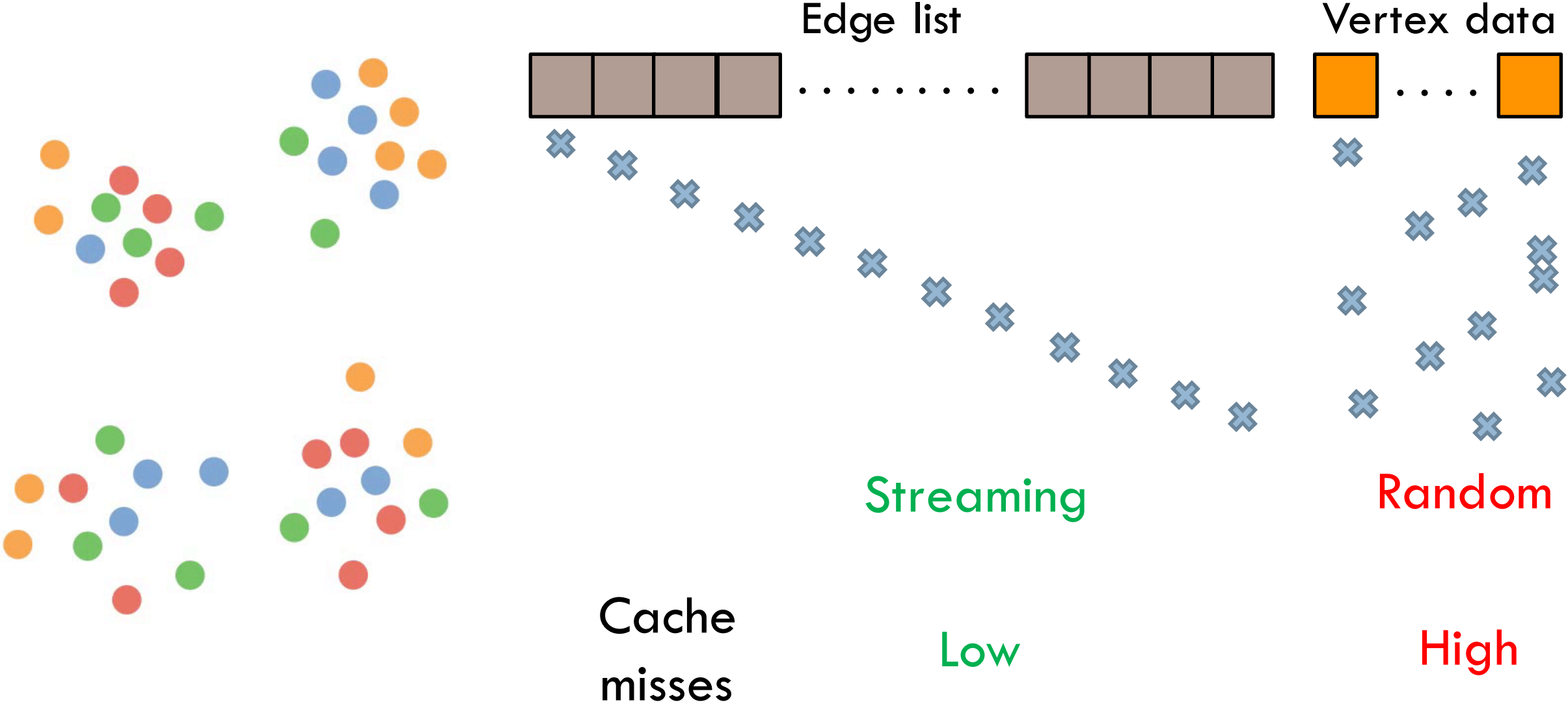
In-memory  
vertex layout



Consecutive vertices in layout are spread out across the graph

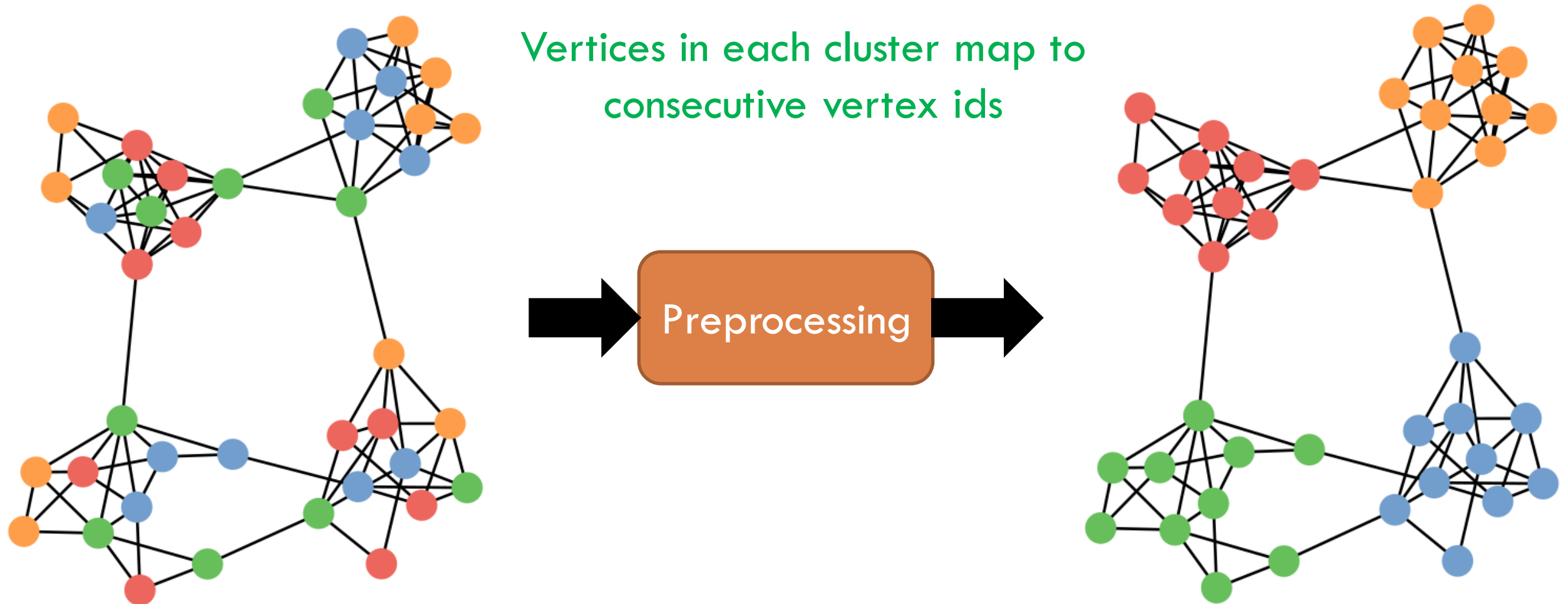


# Access pattern of vertex-ordered schedule

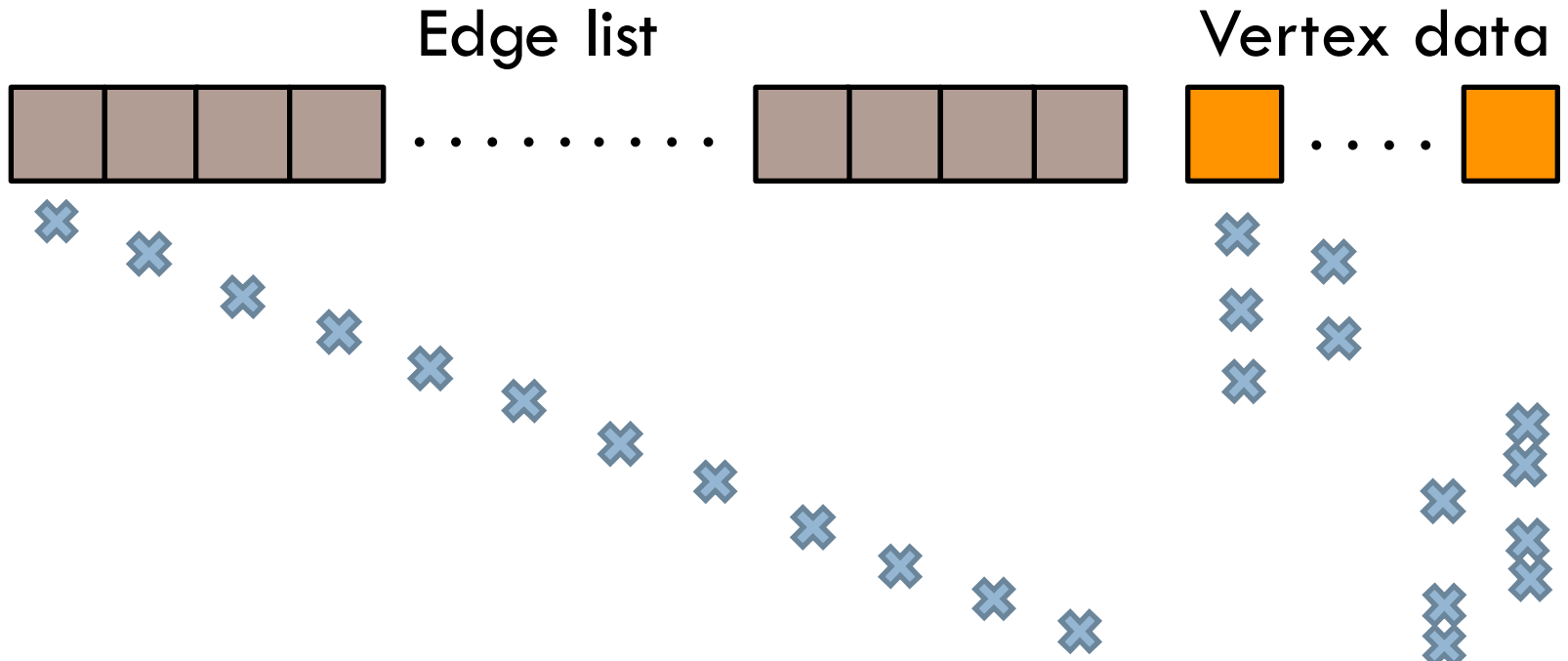
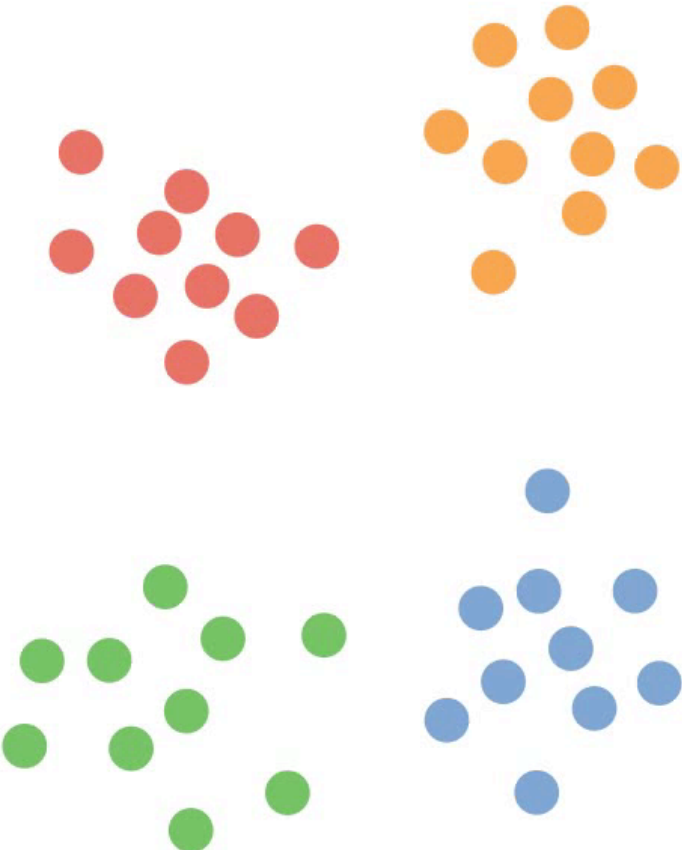




# Preprocessing changes layout for better order



# Access pattern with preprocessed graph



Cache misses

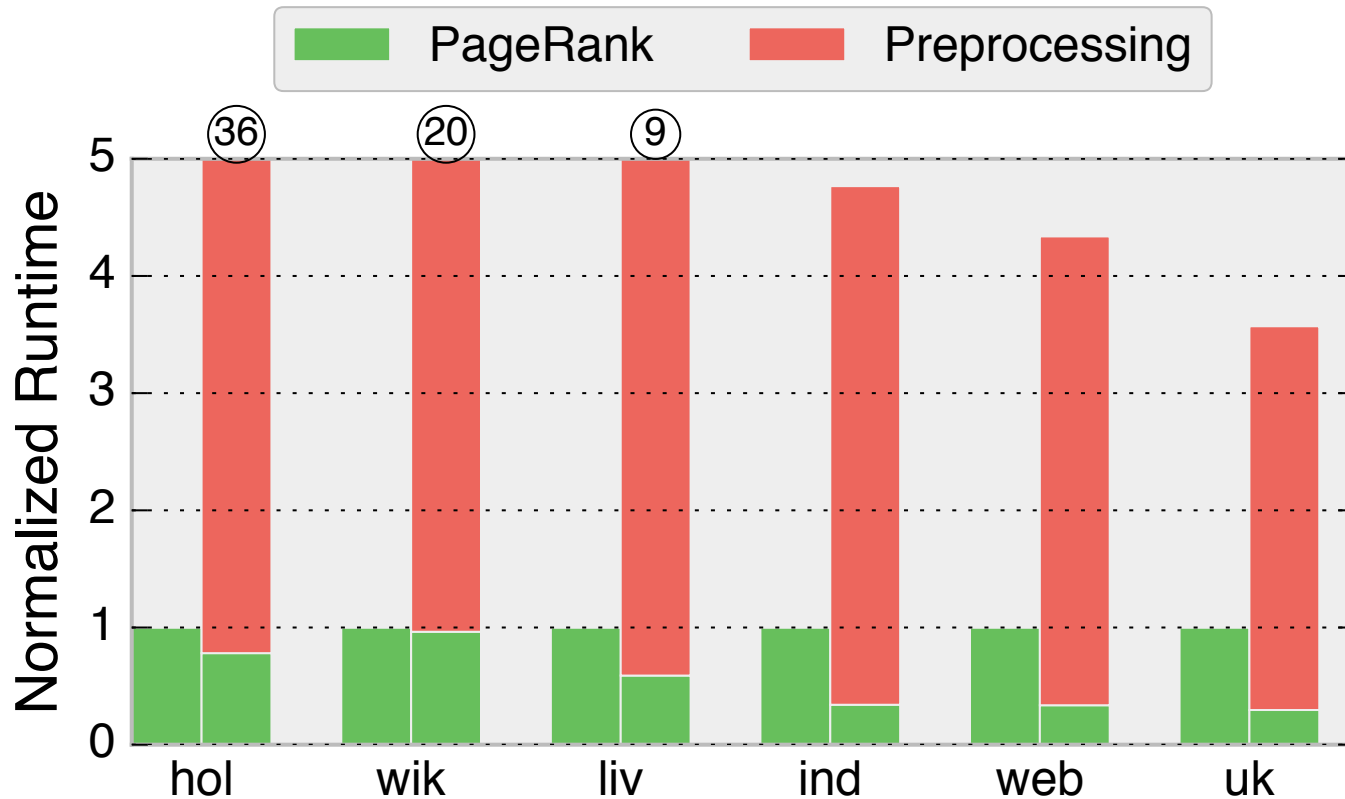
Streaming

Good locality

Low

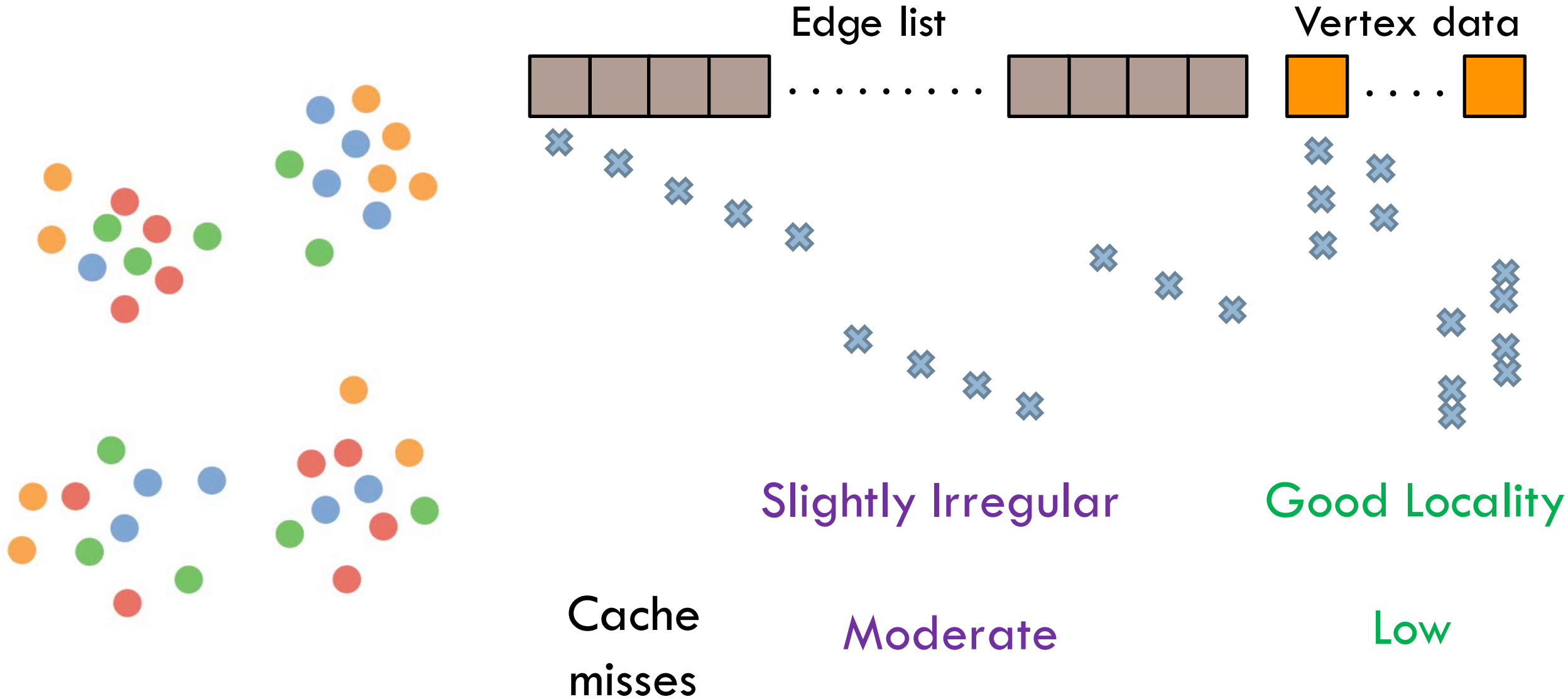
Low

# Preprocessing is often impractical



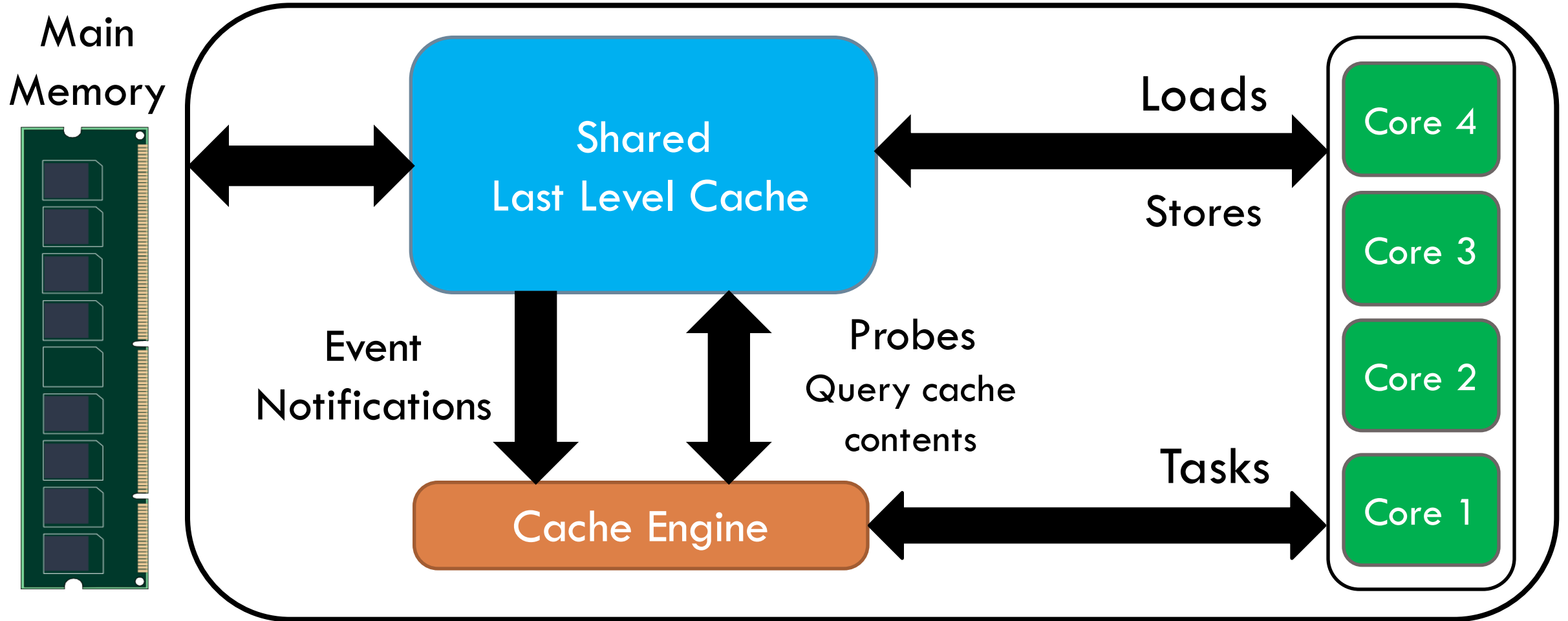
- Preprocessing is more expensive than algorithm itself
- Impractical for many important use cases

# Cache-guided scheduling finds good order at runtime



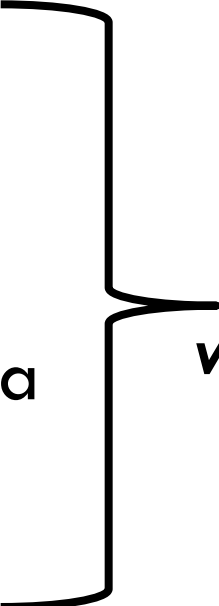
# Cache-Guided Scheduling Design

# High-level design



Maintains a list of tasks ranked based on a locality metric

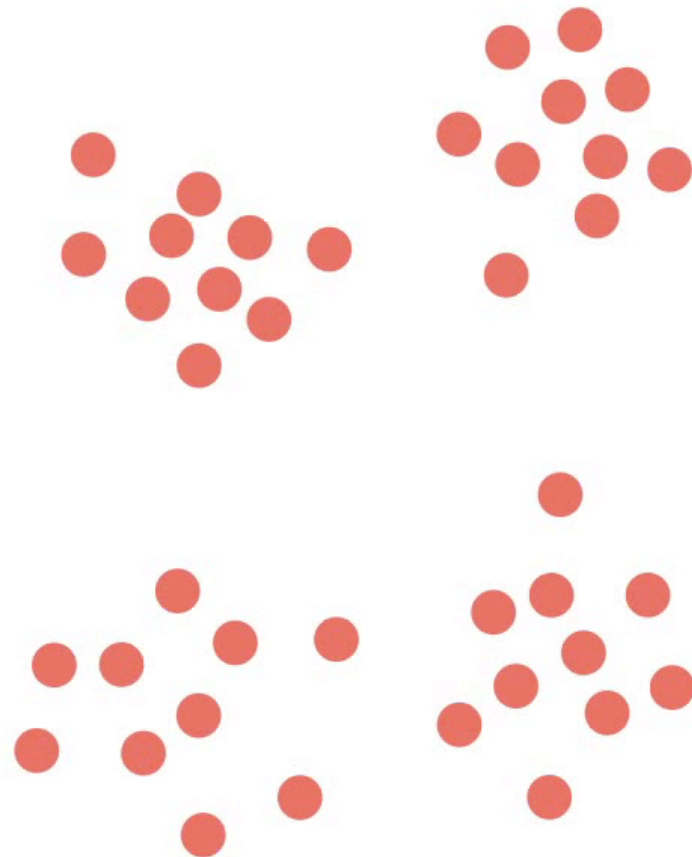
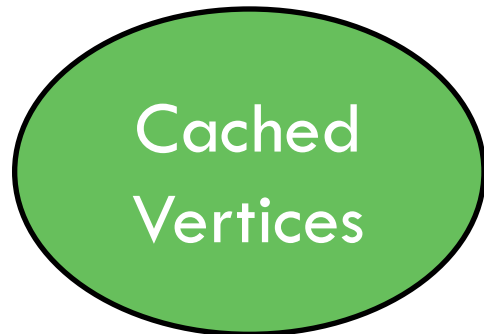
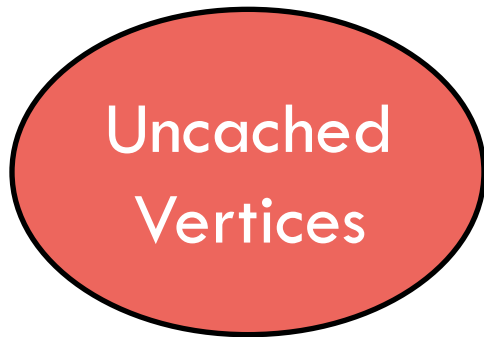
- **Extra memory accesses** to edge list
  - ▣ Filling worklist with tasks
  - ▣ Keeping task scores up to date
- **Space overheads** of worklist and auxiliary metadata
  - ▣ Takes away some of the available cache capacity
- Large **reduction in memory accesses**
  - ▣ Better energy efficiency and performance



For this limit study  
***we ignore these costs***

# Cache-Guided Scheduling of Vertices (CGS-V)

- Ranks and schedules each **vertex** of the graph
- Vertices ranked by fraction of neighbors that are cached





- Large locality benefits
- Track vertices only (not edges)
- *Pitfall: Real-world graphs have skewed degree distributions*
  - ▣ Many high-degree vertices that are connected to most of the graph
- Processing high-degree vertices
  - ▣ Flushes the cache and kills locality
  - ▣ Misses opportunities to process other beneficial regions

- Ranks and schedules **edges** instead of vertices
- **Better locality** due to finer-grained scheduling
- Each edge causes exactly two cache accesses
  - **Simpler ranking algorithm** - Number of endpoints that are cached
- $\#Edges \gg \#Vertices \rightarrow$  **Higher tracking overheads**

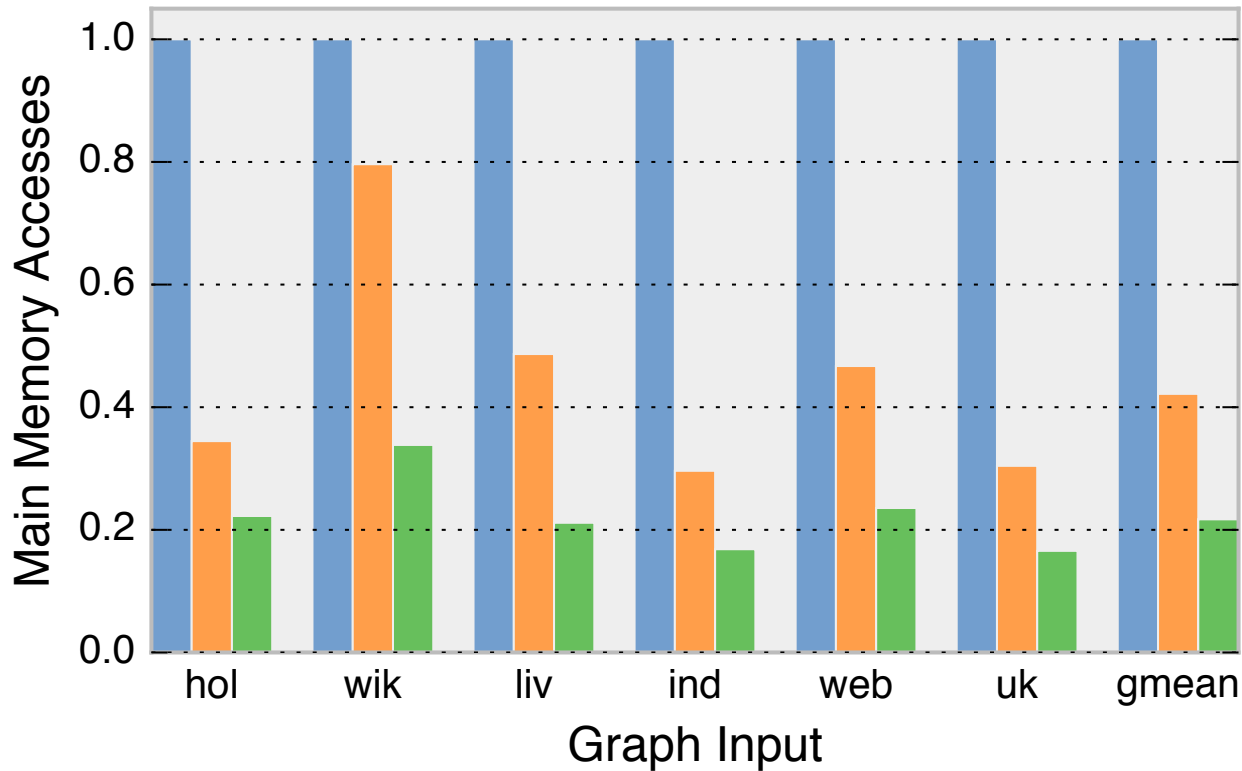
# Limit Study on Benefits of CGS

- Large real-world graphs with up to 100 million vertices, 1 billion edges

Graph	hol	wik	liv	ind	uk	web		nfl	yms
Vertices (Millions)	1.1	3.5	4.8	7.4	19	118		0.5	0.5
Edges (Millions)	113	45	69	194	298	1020		100	61

- Graph algorithms
  - PageRank – 16-byte vertex objects
  - Collaborative Filtering – 256-byte vertex objects
- Custom cache simulator to compute main-memory accesses
  - Single core system
  - 2-level cache hierarchy with 32KB L1, 8MB L2
- See paper for details

# Large reduction in memory accesses for PageRank

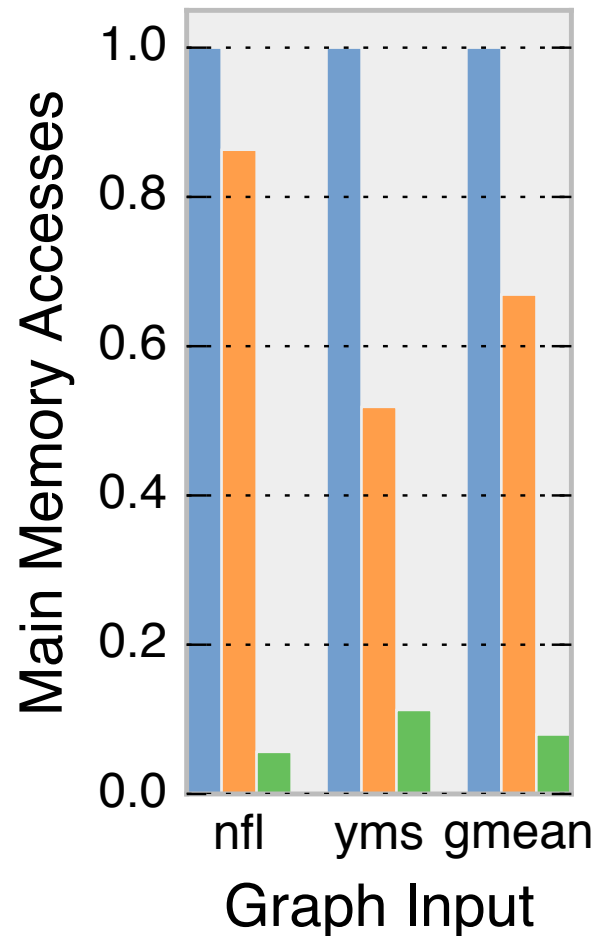


Memory Access Reduction

CGS-V - 2.4x gmean

CGS-E - 4.6x gmean

# Much larger benefits with Collaborative Filtering



## Memory Access Reduction

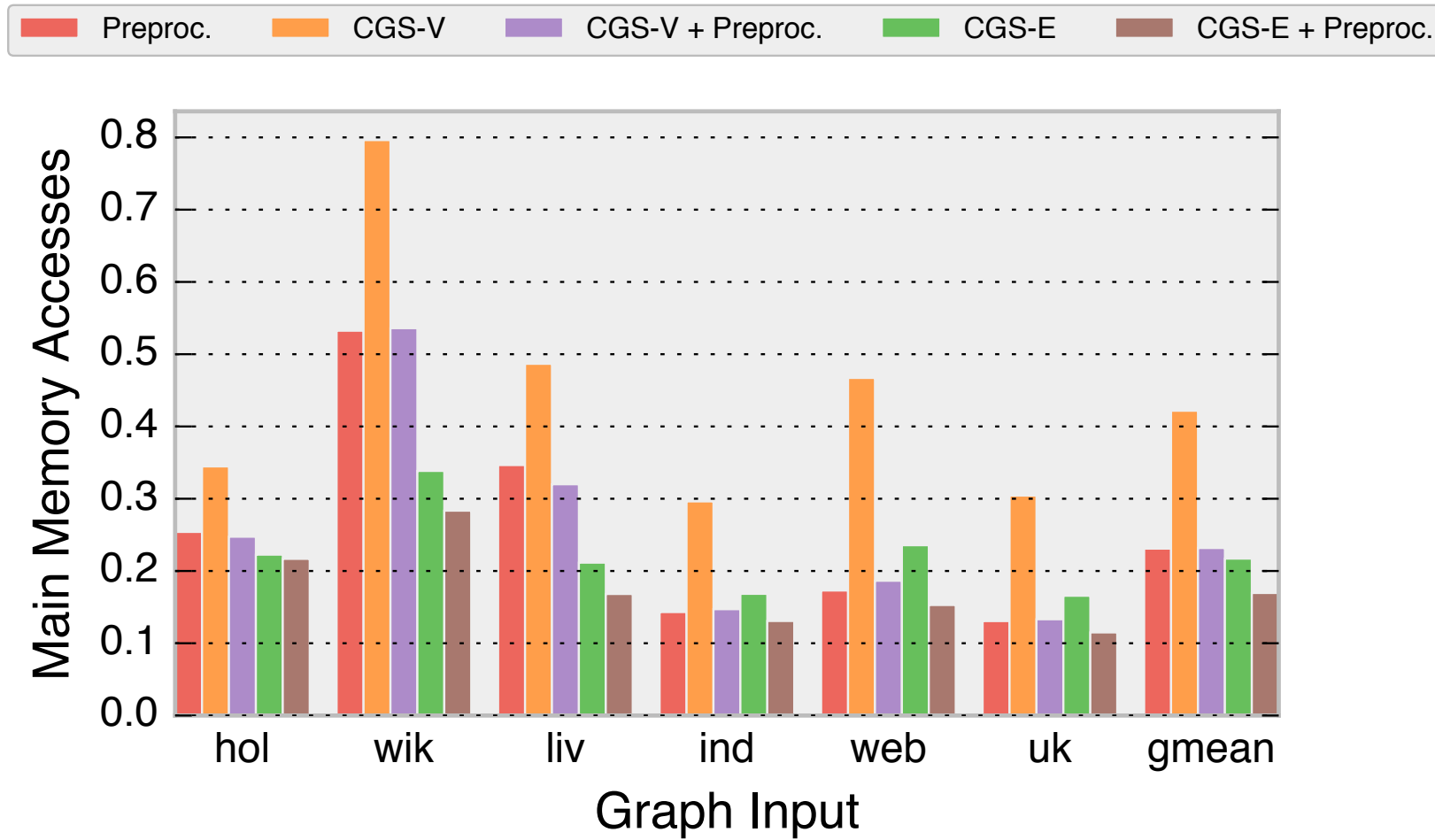
CGS-V - 1.5x gmean

CGS-E - 1.2x gmean

Larger vertex data – 256 bytes per vertex

- Edge list accesses are negligible (3% only)
- Finer-granularity scheduling of CGS-E becomes more important

# CGS benefits from better graph layout



# Ongoing Work

## CGS Hardware Implementation



- Maintaining all vertices in the worklist is prohibitively expensive
- Can a small worklist capture most of the benefits?
  - ▣ Order in which the worklist is filled is crucial
- Adding vertices in order of their id is bad
  - ▣ Explores multiple disjoint regions of the graph simultaneously
- **Insight: Explore the graph in depth-first fashion to fill the worklist**
  - ▣ 100 element worklist gives 50% of the benefits of CGS-E

- Processing each edge takes only a few instructions
  - ▣ Ex. PageRank: One floating point addition per edge
  - ▣ Task scheduling logic must be cheap
- CGS-E gives much better locality than CGS-V, but has higher overheads
- **Practical middle ground: Each task processes a cache line of edges**
  - ▣ Minimizes loss of spatial locality in edge list accesses
  - ▣ Sidesteps the issue of high-degree nodes

- Real-world graphs have abundant locality, but hard to predict
- Cache has rich information about which regions are best to process
- Cache-Guided Scheduling gives large reduction in memory accesses

**THANKS FOR YOUR ATTENTION!**

**QUESTIONS ARE WELCOME!**

