X-Stream: A Case Study in Building a Graph Processing System

Amitabha Roy

(LABOS)

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

X-Stream

- Graph processing system
- Single Machine
- Works on graphs stored
  - Entirely in RAM
  - Entirely in SSD
  - Entirely on Magnetic Disk
- Generic
  - Can do all kinds of graph algorithms from BFS to triangle counting
- Paper, presentation slides and talk video from SOSP 2013 are online

This talk …

- A brief history of X-Stream
  - November 2012 to SOSP camera ready
- Cover the details not in the SOSP text
  - Including bad design decisions ☹

Preliminary Ideas (~ Nov 2012)

- Toying with graph processing from an algorithms perspective
- Observed graph processing as an instance of SpMV

$$Y = X^T A$$

- X,Y are vertex state vectors. A is the adjacency matrix
- Operators are algorithm specific
  - Numerical operations for pagerank
  - Reachability (and tree parent assignment) for BFS
- Do we know how to do sparse matrix multiplication **efficiently** ?

Preliminary Ideas (~ Nov 2012)

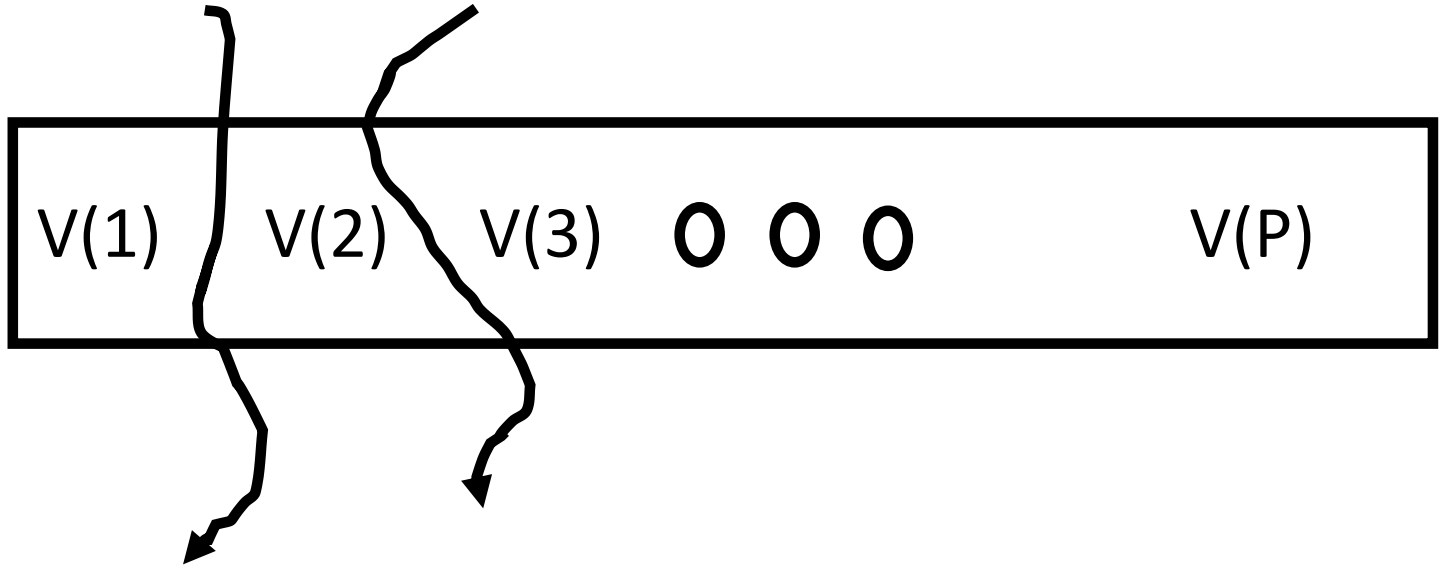
- Yes ! Algorithms community had beaten the problem to death ☺

Optimal Sparse Matrix Dense Vector Multiplication in the I/O-Model

-Bender et. al.

- Regretted not paying attention in grad school to complexity theory
- Isolated the good ideas from that paper
  - Cache the essentials (upper level of memory hierarchy, random access)
  - Stream everything else (lower level of memory hierarchy, block transfer)
  - Stream, don't sort the edge list

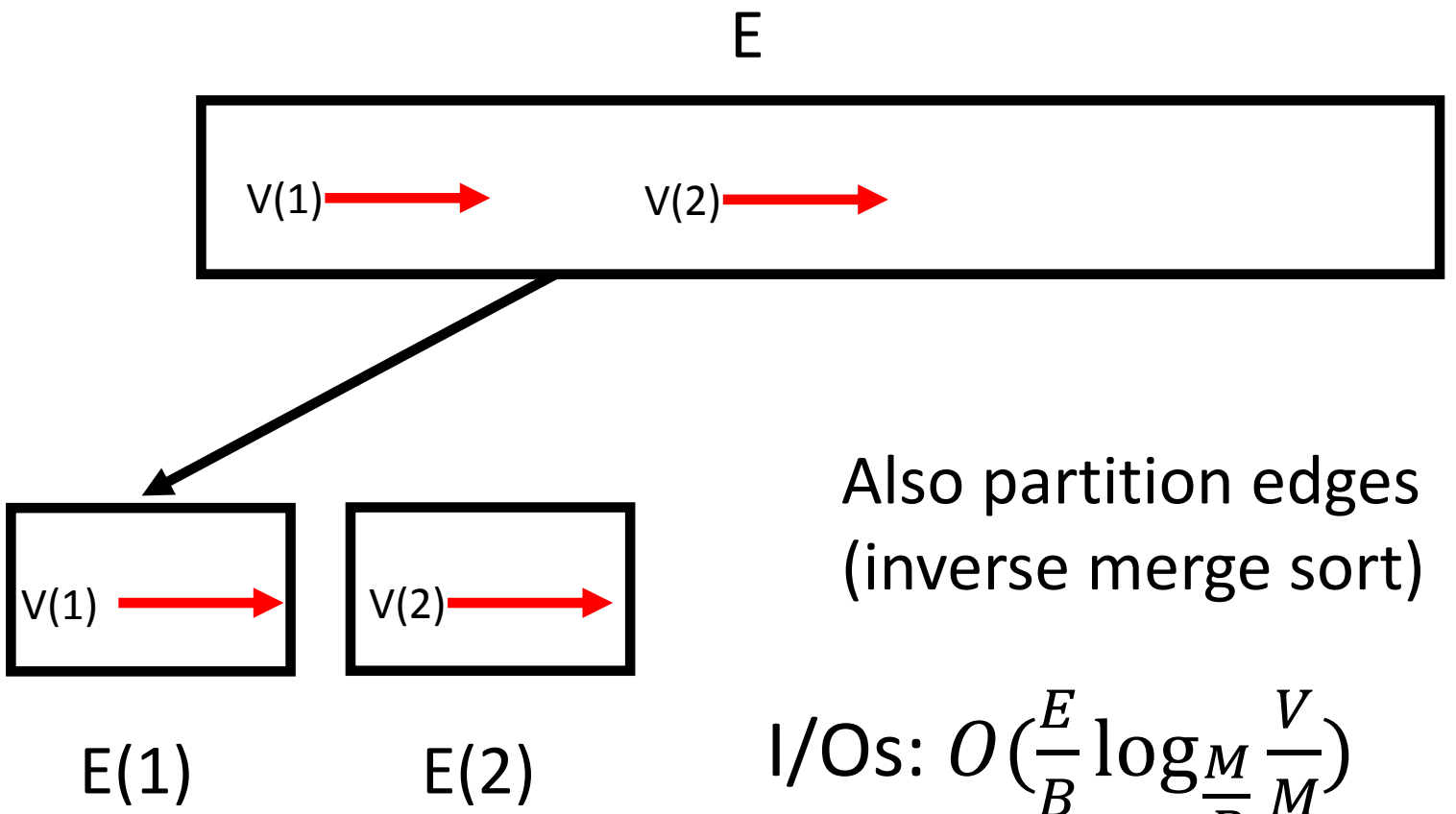
Preliminary Ideas (~ Nov 2012)



Shard vertices to fit each chunk in cache

I/Os to memory: $O(P) = O(\frac{V}{M})$

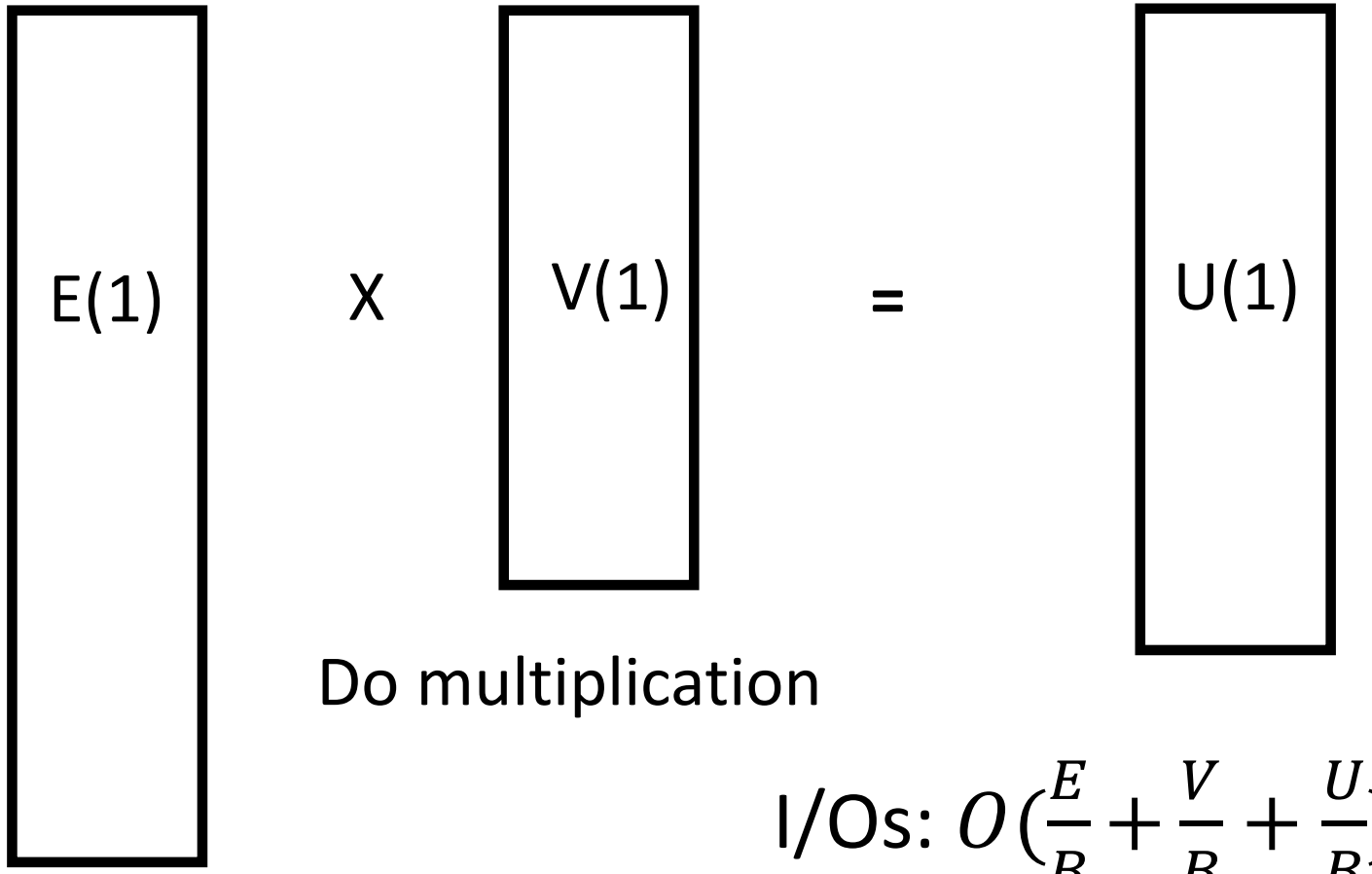
Preliminary Ideas (~ Nov 2012)



Also partition edges (inverse merge sort)

I/Os: $O\left(\frac{E}{B}\log_{\frac{M}{B}}\frac{V}{M}\right)$

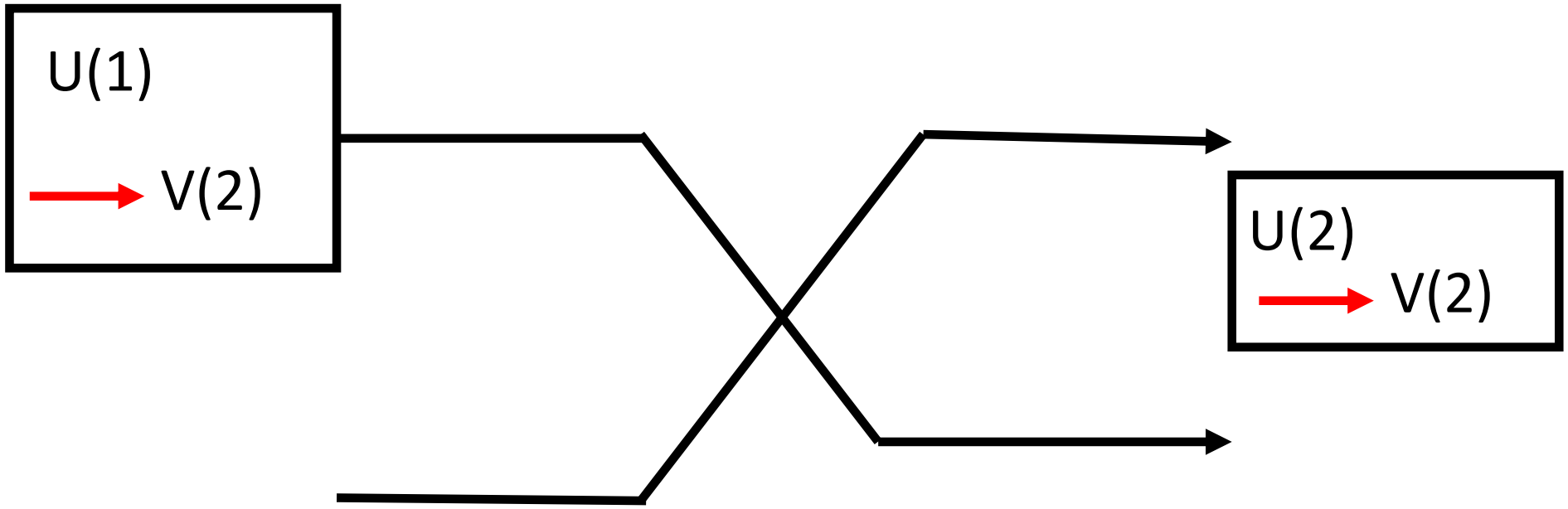
Preliminary Ideas (~ Nov 2012)



Do multiplication

I/Os: $O\left(\frac{E}{B}+\frac{V}{B}+\frac{U}{B}\right)$

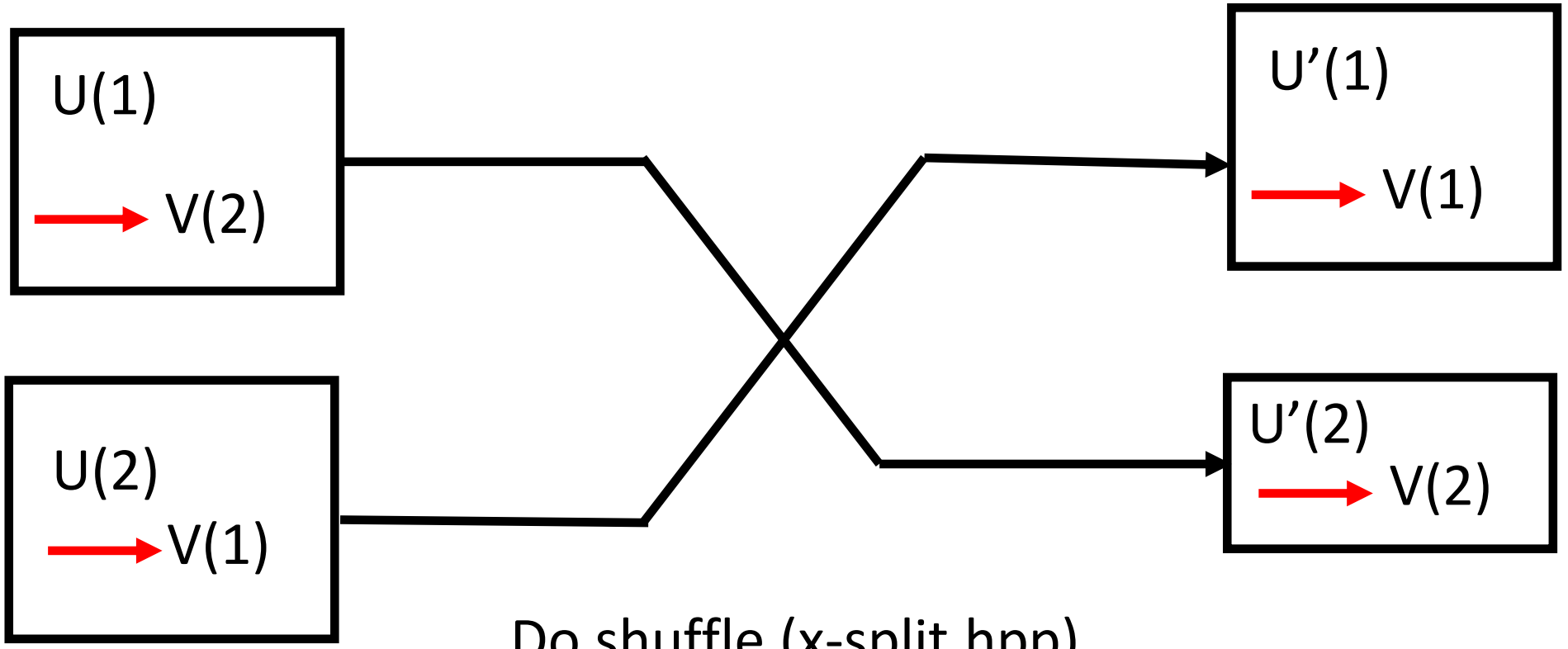
Preliminary Ideas (~ Nov 2012)



Do shuffle

I/Os: $O(\frac{U}{B}\log_{\frac{M}{B}}\frac{V}{M})$

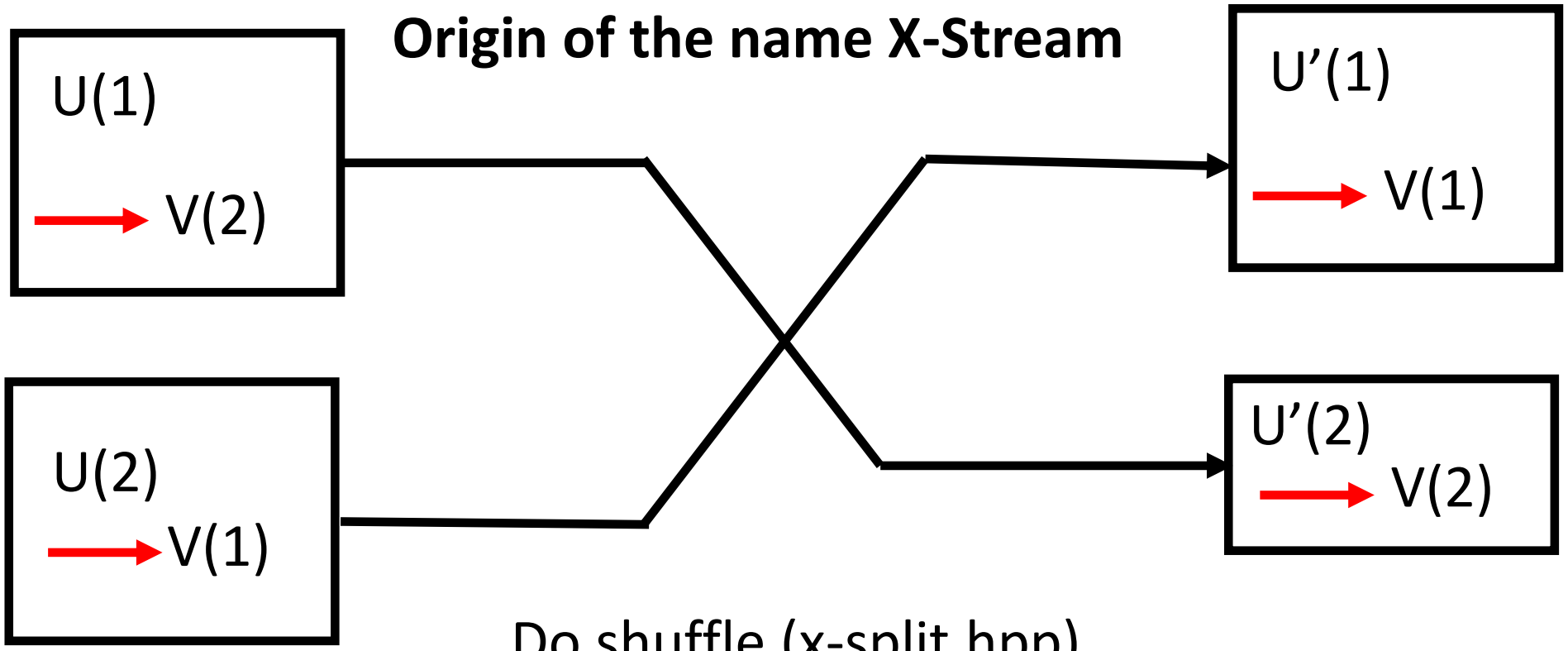
Preliminary Ideas (~ Nov 2012)



Do shuffle (x-split.hpp)

I/Os: $O(\frac{U}{B}\log_{\frac{M}{B}}\frac{V}{M})$

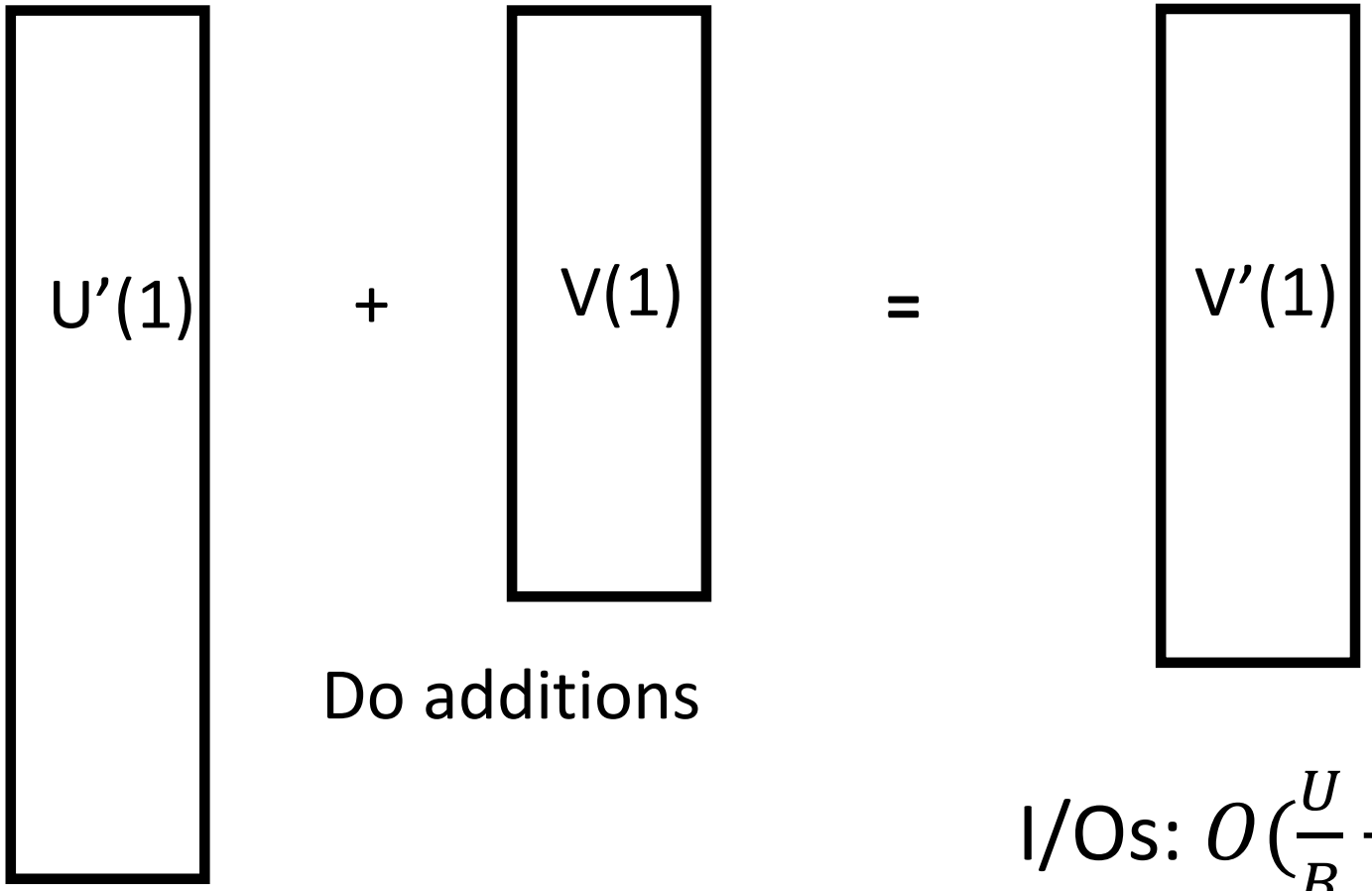
Preliminary Ideas (~ Nov 2012)



Do shuffle (x-split.hpp)

I/Os: $O(\frac{U}{B}\log_{\frac{M}{B}}\frac{V}{M})$

Preliminary Ideas (~ Nov 2012)



I/Os: $O\left(\frac{U}{B}+\frac{V}{B}\right)$

Preliminary Ideas (~ Nov 2012)

Total I/Os: $\frac{V+E}{B} + \frac{E}{B}\log_{\frac{M}{B}}\frac{V}{M}$

Bender et. al. tells us this is very close to the most efficient solution

Preliminary Ideas (~ Dec 2012)

- Yes, but….
  - Algorithmic complexity theory ignores constants
- Systems Research
  - Hypothesize
  - Build
  - Measure
- Quickly prototyped an SpMV implementation in C++
- Compared to Graphchi

Preliminary Ideas (~ Jan 2013)

- Results (BFS and pagerank) looked good
- Beat Graphchi by a huge margin ☺
- Often finished faster than Graphchi finished producing shards !
- Now what ?
- Write a “systems” paper from an “algorithms” idea

Preliminary Ideas (~ Jan 2012)

- HotOS submission (Jan 10, 2013, 6 page paper)
- “Pitch” ?
- Graph processing systems spend a lot of time indexing data before processing it
- Here is a system that produces results from unordered “big-data”
- It works from main memory and disk
- Sketch of the system (minimal “complexity theory”)
- Results: Beats Graphchi for graphs on disk
- Results: Beats sorting the edge list for graphs in memory

The next stage (~February 2013)

- X-Stream seems like a good idea
- Lets try to build and evaluate the full system
  - Only thought about SOSP very vaguely
- Loads of code written that month (month of code ☺)
- Made some arbitrary decisions that (we hoped) would not impact end result

Arbitrary Decision 1

- I/O path to disk

Option	Buffers controlled by	Overhead
read()/write()	OS (pagecache)	Copy
mmap	OS (pagecache)	Minor fault
Direct I/O	You	None

- Chose direct I/O. Great performance, controlled mem footprint ☺
- Nightmare to implement properly ☹ (look at core/disk_io.hpp)

Arbitrary Decision 2

- Shuffle entirely in memory
- Greatly simplifies implementation
  - However this means ….
- One buffer per partition should fit in memory (at least 16 MB)
- Number of partitions bounded
  - Below: Have to fit vertex data of a partition into memory
  - Above: Have to fit one buffer from each partition into memory
- Intersect covers large enough graphs (see sec 3.4 of SOSP paper)

Arbitrary Decision 3

- X-Stream targets any two level memory hierarchy

1. Disk/SSD + Main memory
2. Main memory + CPU cache

- Correct approach is to build two ‘X-Streams’ as independent pieces of software
- We instead decided to implicitly deal with a three level memory hierarchy in the code
- Disk/SSD + Main memory + CPU cache
- Does in-memory partitions of disk partitions !

Arbitrary Decision 3

- Why ?
  - Algorithmically elegant, same I/O complexity for any combination of two levels in the hierarchy
  - User does not need to worry about whether the graph fits in memory
  - In the distant future PCM cache connections would be handled gracefully
- Why not ?
  - HORRIBLY complex ☹ (look at x-lib.hpp)
  - Elegant complexity theory useless for a systems paper
  - PCM is yet to arrive

SOSP Submission ~ March 2013

- HotOS results arrived in March
  - Paper got rejected but …
- Review and PC explicitly said
  - Great set of ideas
  - Almost got in
  - Felt it was mature enough for a full conference rather than HotOS
- Decided to submit to SOSP at that point
  - Code base was stable, experiments were running, results were good

SOSP Submission ~ March 2013

- Reworked “pitch” for SOSP submission
- De-emphasized algorithmic contributions
- De-emphasized ability to process unordered data
- Emphasized difference between sequential and random access bandwidth
- Called the execution model “edge-centric”
- Justified saying that it results in more sequential access
- Paper became very evaluation heavy

SOSP Submission ~ March 2013

- Experimental evaluation critical to strength of a systems paper
- Carefully planned and executed experiments (~ 500 hours)
  - Figure placeholders in the paper with expected results
  - Tried to duplicate configurations in the cluster

    ~ 4 machines with 2x3TB drives each

    1 machine with SSD
  - 4x experimental throughput for the magnetic disk experiments
  - SSD experiments slower as only one SSD
    - Hence more magnetic disk results than SSD results

April 2013

- Vacation
  - Burnt out
  - Zero work ☺

May 2013

- Started thinking about more algorithms over X-Stream
- SOSP submission had
  - BFS, CC, SSSP, MIS, Pagerank, ALS.
  - Could all be cast as SpMV and therefore fitted our execution model
- Wanted to go further: show that X-Stream model not limited
  - SCC
  - Belief propagation
- Solution was to allow algorithm to generate new sparse matrices

May 2013

- X-Stream implemented $Y=X^TA$ efficiently
- A was static
  - for graph G=(V, E) A = E, X = V
- Allowed X-Stream to generate matrices instead of vectors

  B = X I A
- Very similar to SpMV
- Similar algorithmic complexity
- Equivalent to generating new edge list

May 2013

- Divided algorithms on top of X-Stream into two categories
  - Standard : BFS, CC, SSSP, Pagerank
  - Special: BP, Triangle counting, SCC, MCST

- Special algorithms use a lower level interface that lets them create, manage and manipulate sparse matrices of O(E) non-zeros.

- Had to completely rewrite core X-Stream to support this ☹

June 2013

- Started preparing for possible resubmission to ASPLOS (July deadline)
- Added in more ”systemsy” features
- Primarily compression
- Added zlib compression
- Bad idea in retrospect ☹
  - Zlib too slow to keep up with streaming speeds from RAIDED magnetic disks !!
  - Software decompression < 200 MB/s
  - RAID array, sequential access > 300 MB/s

July – August 2013

- SOSP paper accepted ☺
- SOSP camera ready deadline was September
- Diverted July and August to doing strategic extensions to X-Stream
- Worked with two summer interns
  - Intern 1: Added support to express algorithms in Python on X-Stream
  - Intern 2: Added more algorithms, Triangle counting, BC, K-Cores, HyperANF

August 2013 - September 2013

- SOSP camera ready
- Re-ran experiments
- Completely re-wrote paper, made it far clearer
- Interesting points:
  - Yahoo webgraph did not work well, left it as such
  - Kept in complexity analysis (hat-tip to X-Stream's roots)
- Camera ready deadline 15 Sep
- Conference presentation Nov 3 (video online)

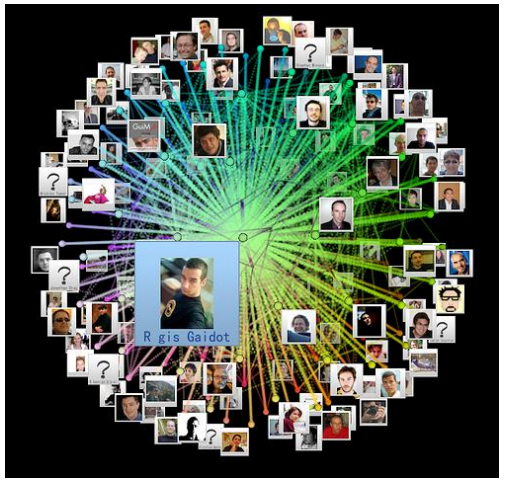
Conclusion

- Overview of a large systems project from concept to publication
- Many mistakes made, not apparent from finished paper
- Lots of people contributed
  - Willy Zwaenepoel, Ivo Mihailovic, Mia Primorac, Aida Amini
- What next ?
  - X-Stream could get us to a billion plus edges
  - How about a trillion edges ?
- **X-1: Scale out version**

BACKUP (SOSP slides)

X-Stream

Process large graphs on a single machine



1U server = 64 GB RAM + 2 x 200 GB SSD + 3 x 3TB drive

Approach

- Problem: Graph traversal = random access
- Random access is inefficient for storage
  - Disk (500X slower)
  - SSD (20X slower)
  - RAM (2X slower)

Solution: X-Stream makes graph accesses sequential

Contributions

- Edge-centric scatter gather model
- Streaming partitions

Standard Scatter Gather

- Edge-centric scatter gather based on Standard Scatter gather
- Popular graph processing model

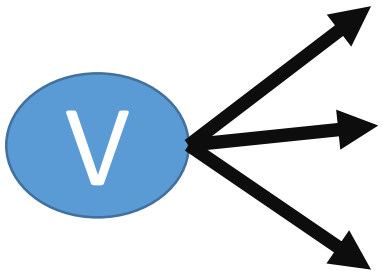
  Pregel [Google, SIGMOD 2010]

  …

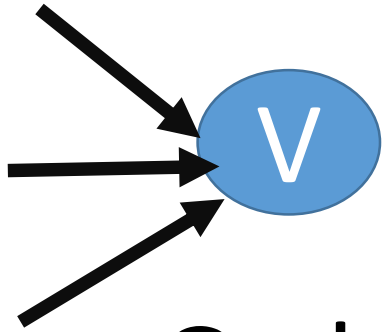
  Powergraph [OSDI 2012]

Standard Scatter Gather

- State stored in vertices
- Vertex operations
  - Scatter updates along outgoing edges
  - Gather updates from incoming edges

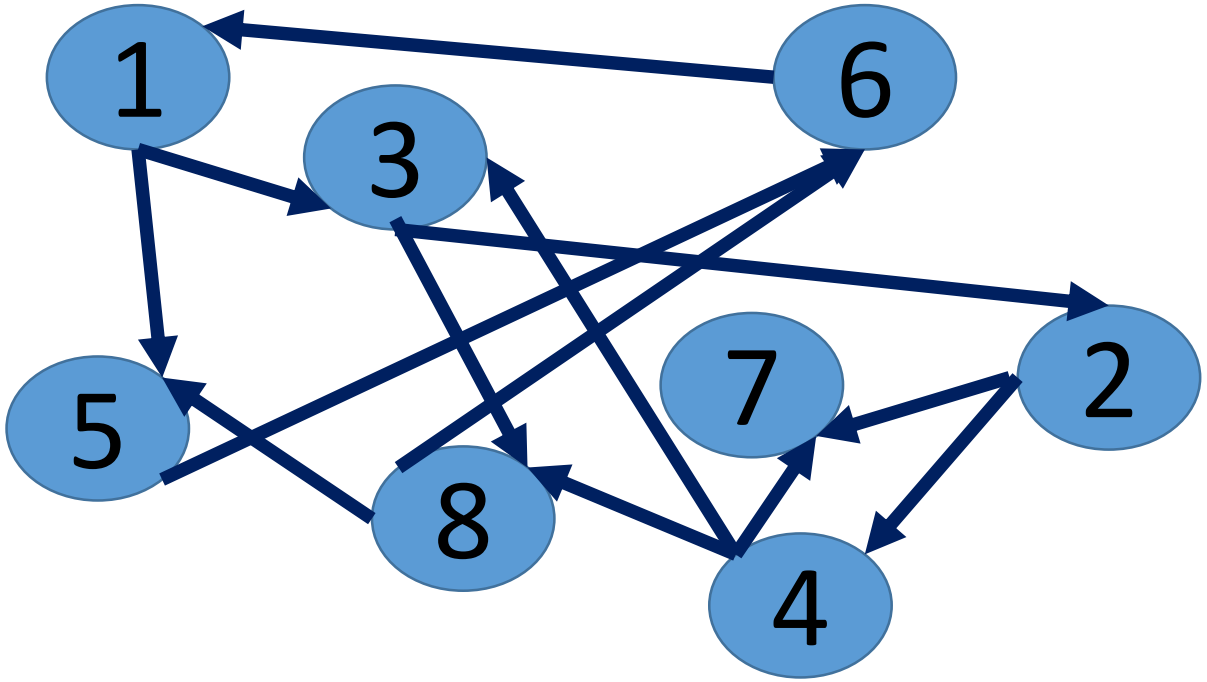


Scatter



Gather

Standard Scatter Gather



BFS

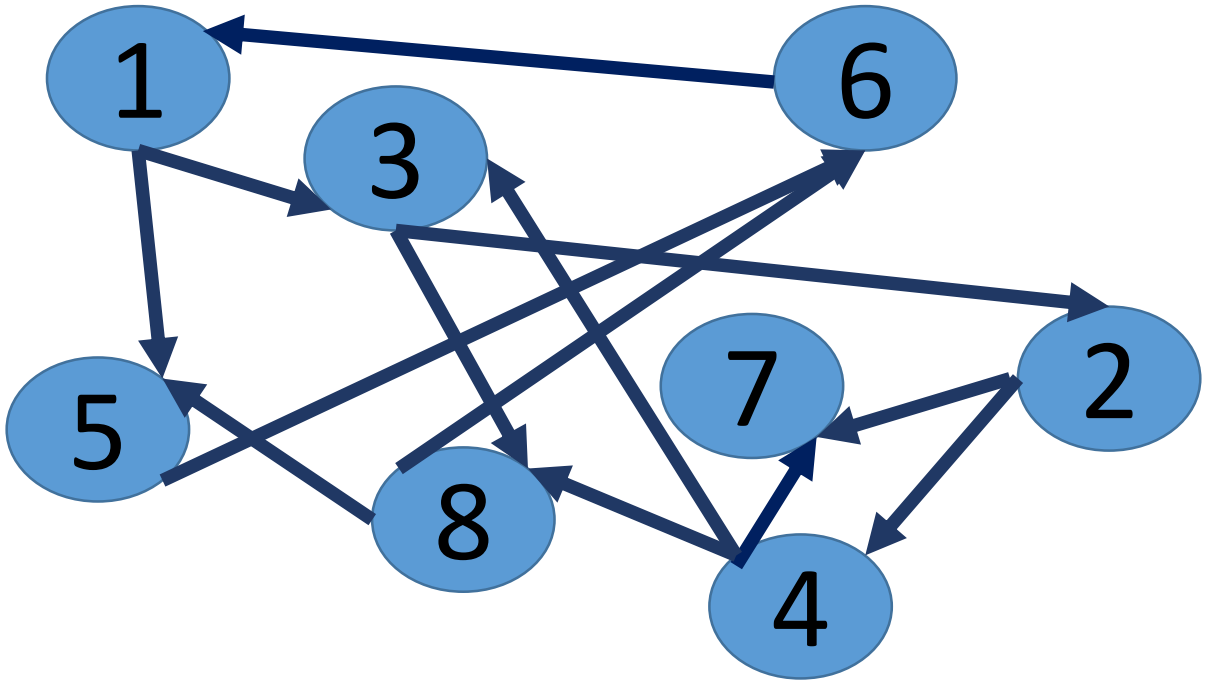
Vertex-Centric Scatter Gather

- Iterates over vertices

```
for each vertex v                    Scatter
  if v has update
    for each edge e from v
      scatter update along e
```

- Standard scatter gather is **vertex-centric**
- Does not work well with storage

Vertex-Centric Scatter Gather



BFS

Lookup Index

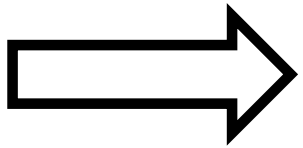
V
1
2
3
4
5
6
7
8

SOURCE	DEST
1	3
1	5
2	7
2	4
3	2
3	8
4	3
4	7
4	8
5	6
6	1
8	5
8	6

Transformation

```
for each vertex v                          Scatter
  if v has update
    for each edge e from v
        scatter update along e
```

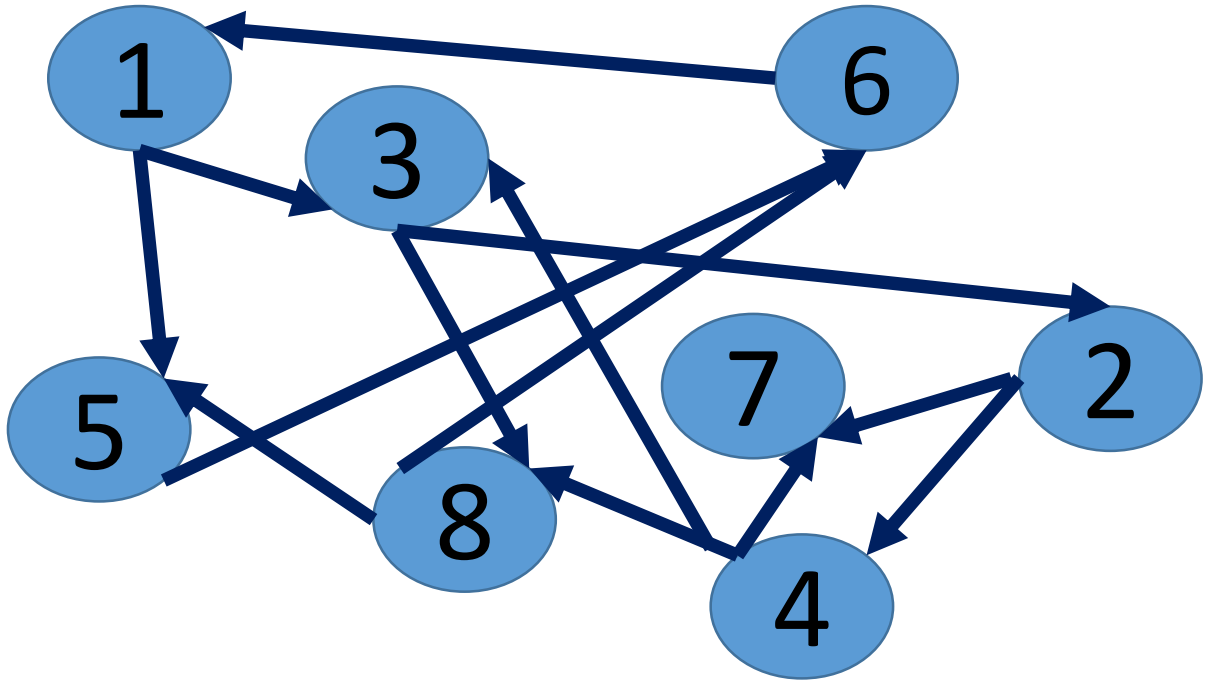
Vertex-Centric



```
for each edge e                            Scatter
    If e.src has update
        scatter update along e
```

Edge-Centric

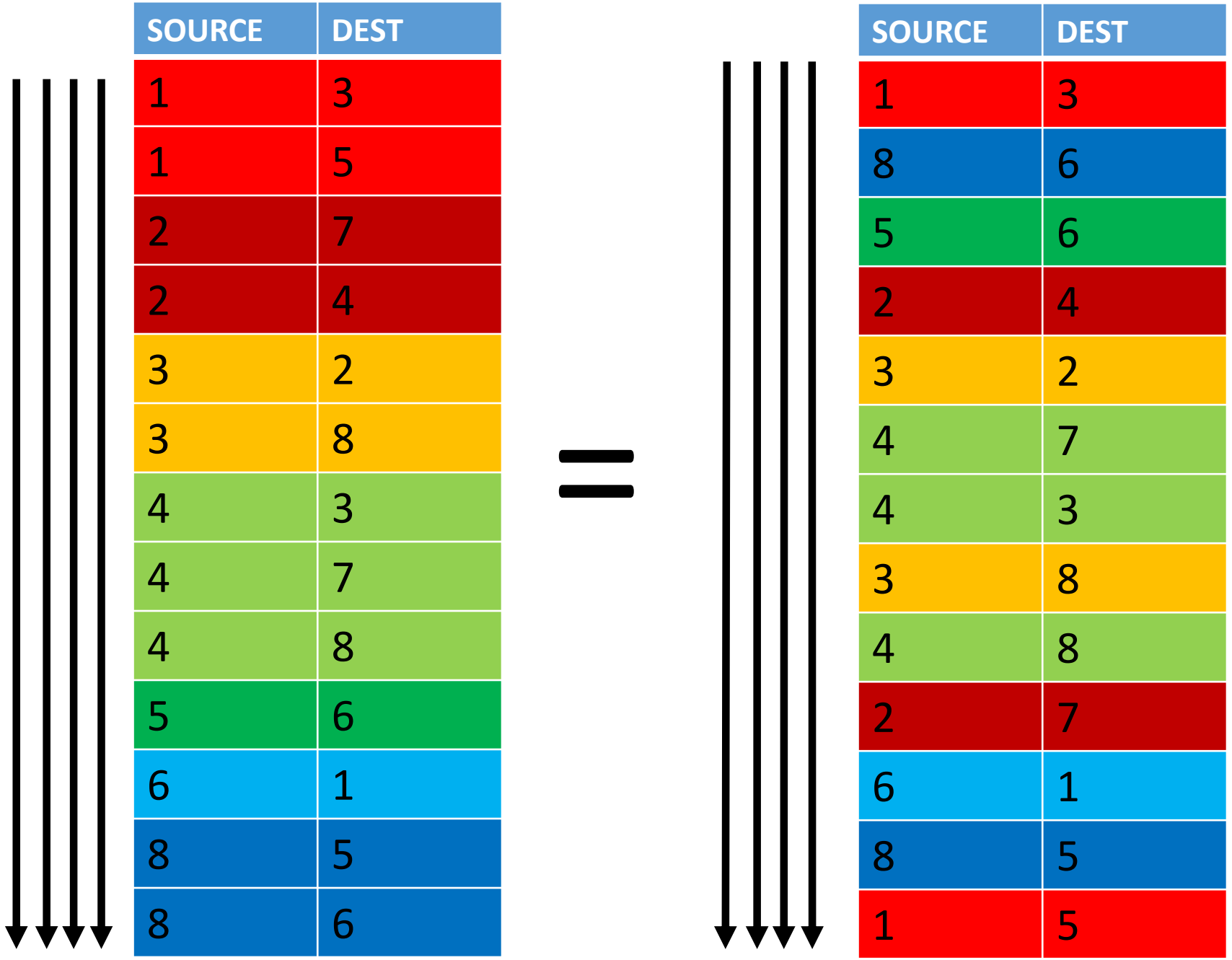
Edge-Centric Scatter Gather



BFS

V
1
2
3
4
5
6
7
8

SOURCE	DEST
1	3
1	5
2	7
2	4
3	2
3	8
4	3
4	7
4	8
5	6
6	1
8	5
8	6



SOURCE	DEST
1	3
1	5
2	7
2	4
3	2
3	8
4	3
4	7
4	8
5	6
6	1
8	5
8	6

=

SOURCE	DEST
1	3
8	6
5	6
2	4
3	2
4	7
4	3
3	8
4	8
2	7
6	1
8	5
1	5

No index

No clustering

No sorting

Tradeoff

Vertex-centric Scatter-Gather: $\frac{Edge\ Data}{Random\ Access\ Bandwidth}$

Edge-centric Scatter-Gather: $\frac{Scatters \times Edge\ Data}{Sequential\ Access\ Bandwidth}$

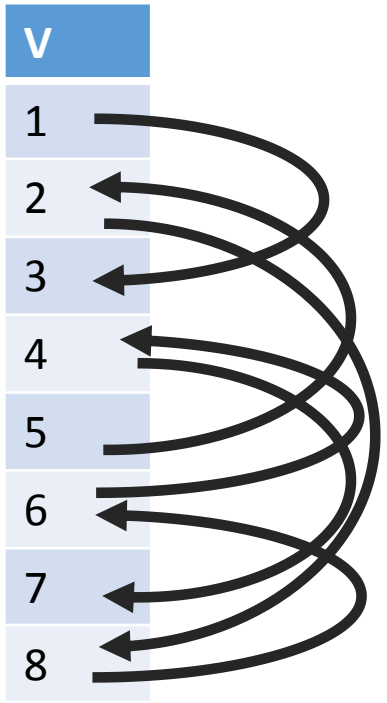
- Sequential Access Bandwidth >> Random Access Bandwidth
- Few scatter gather iterations for real world graphs
  - Well connected, variety of datasets covered in the paper

Contributions

- Edge-centric scatter gather model
- **Streaming partitions**

Streaming Partitions

- Problem: still have random access to vertex set



- Solution: partition the graph into streaming partitions

Streaming Partitions

- A streaming partition is
  - A subset of the vertices that fits in RAM
  - All edges whose source vertex is in that subset
  - No requirement on quality of the partition

Partitioning the Graph

Subset of vertices

V1
1
2
3
4

SOURCE	DEST
1	5
4	7
2	7
4	3
4	8
3	8
2	4
1	3
3	2

V2
5
6
7
8

SOURCE	DEST
5	6
8	6
8	5
6	1

Random Accesses for Free

V1
1
2
3
4

SOURCE	DEST
1	5
4	7
2	7
4	3
4	8
3	8
2	4
1	3
3	2

Generalization

V1
1
2
3
4

Fast storage

SOURCE	DEST
1	5
4	7
2	7
4	3
4	8
3	8
2	4
1	3
3	2

Slow storage

Applies to any two level memory hierarchy

Generally Applicable

RAM

Disk

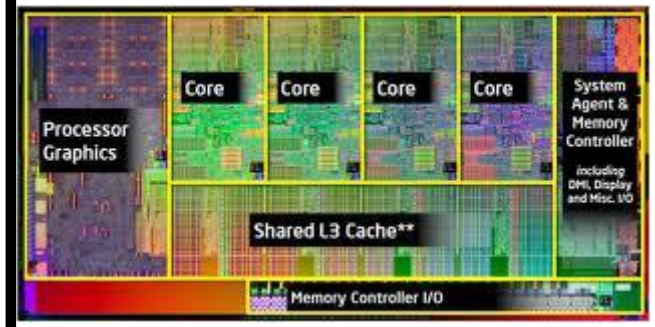
OR

RAM

SSD

OR

CPU Cache

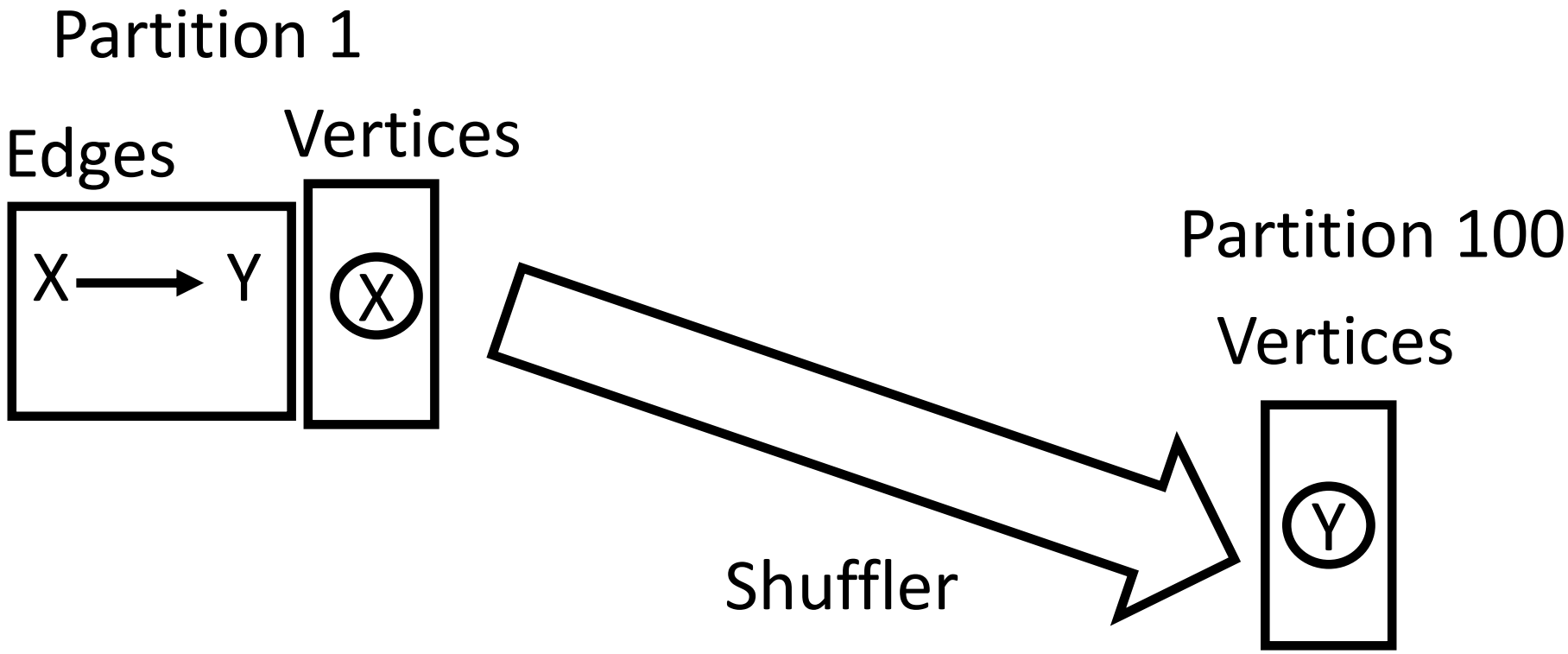


RAM

Parallelism

- Simple Parallelism
- State is stored in vertex
- Streaming partitions have disjoint vertices
- Can process streaming partitions in parallel

Gathering Updates



Minimize random access for large number of partitions
Multi-round copying akin to merge sort but cheaper

Performance

- Focus on SSD results in this talk
  - Similar results with in-memory graphs

Baseline

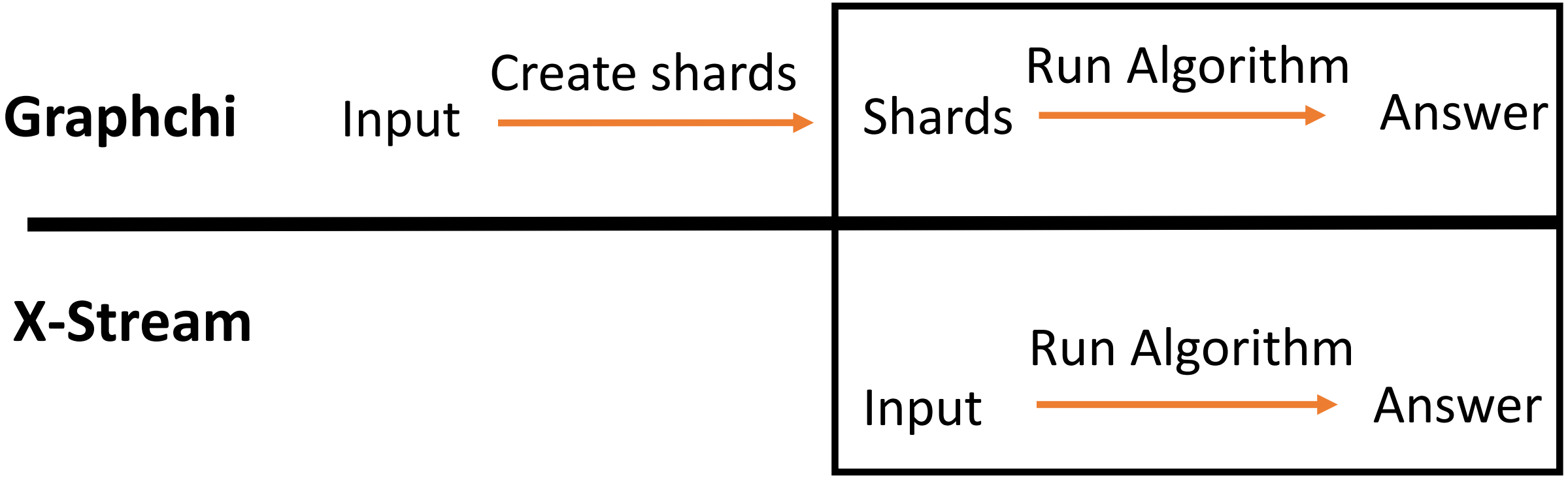
- Graphchi [OSDI 2012]
- First to show that graph processing on a single machine
  - Is viable
  - Is competitive
- Also targets larger sequential bandwidth of SSD and Disk

Different Approaches

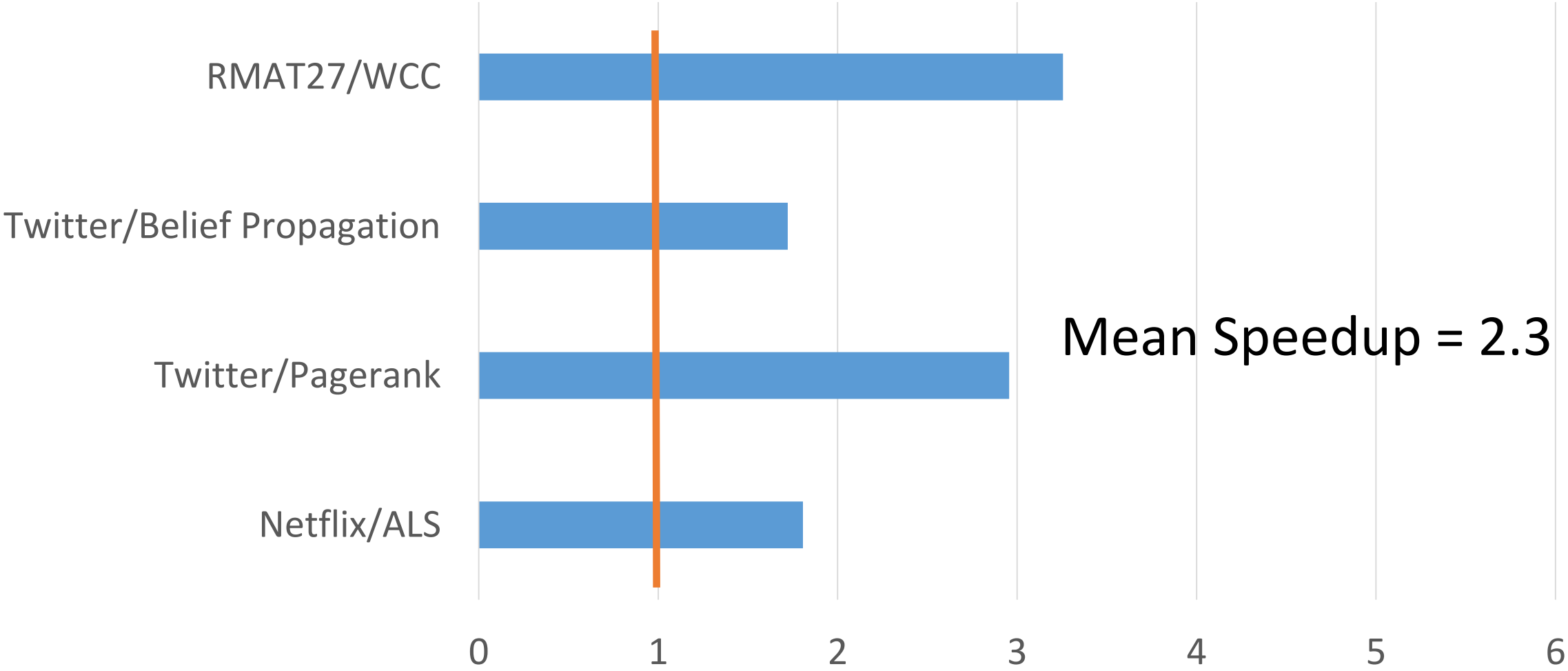
- **Fundamentally different approaches to same goal**
- Graphchi uses “shards”
  - Partitions edges into **sorted** shards
- X-Stream uses sequential scans
  - Partitions edges into **unsorted** streaming partitions

Baseline to Graphchi

- Replicated OSDI 2012 experiments on our SSD

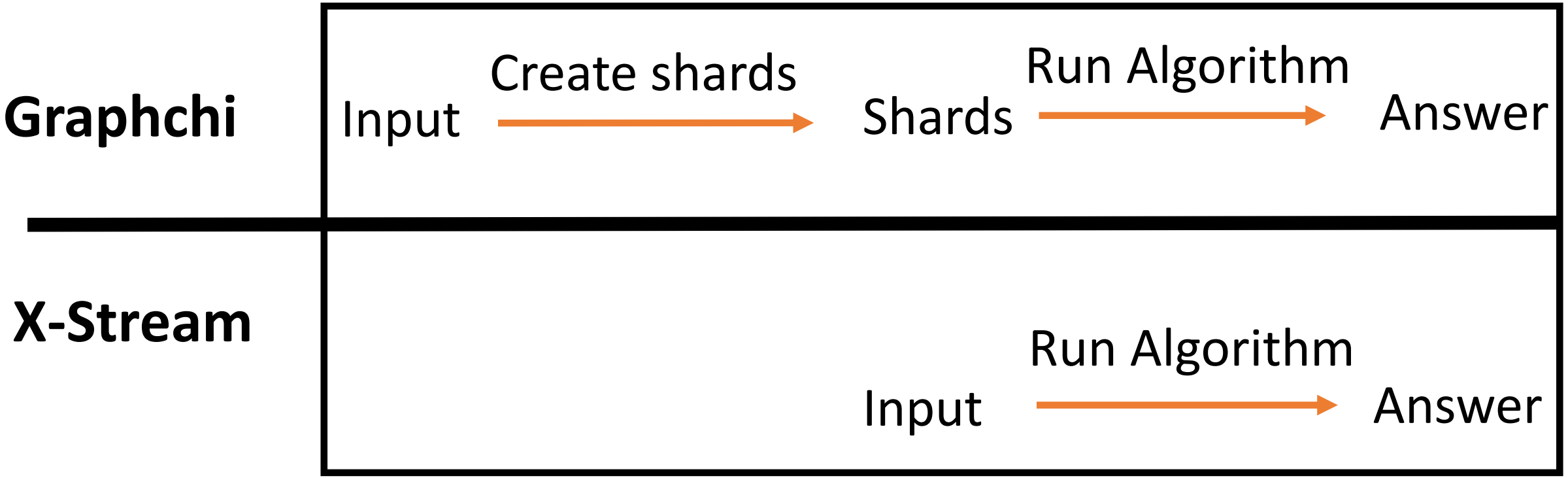


X-Stream Speedup over Graphchi

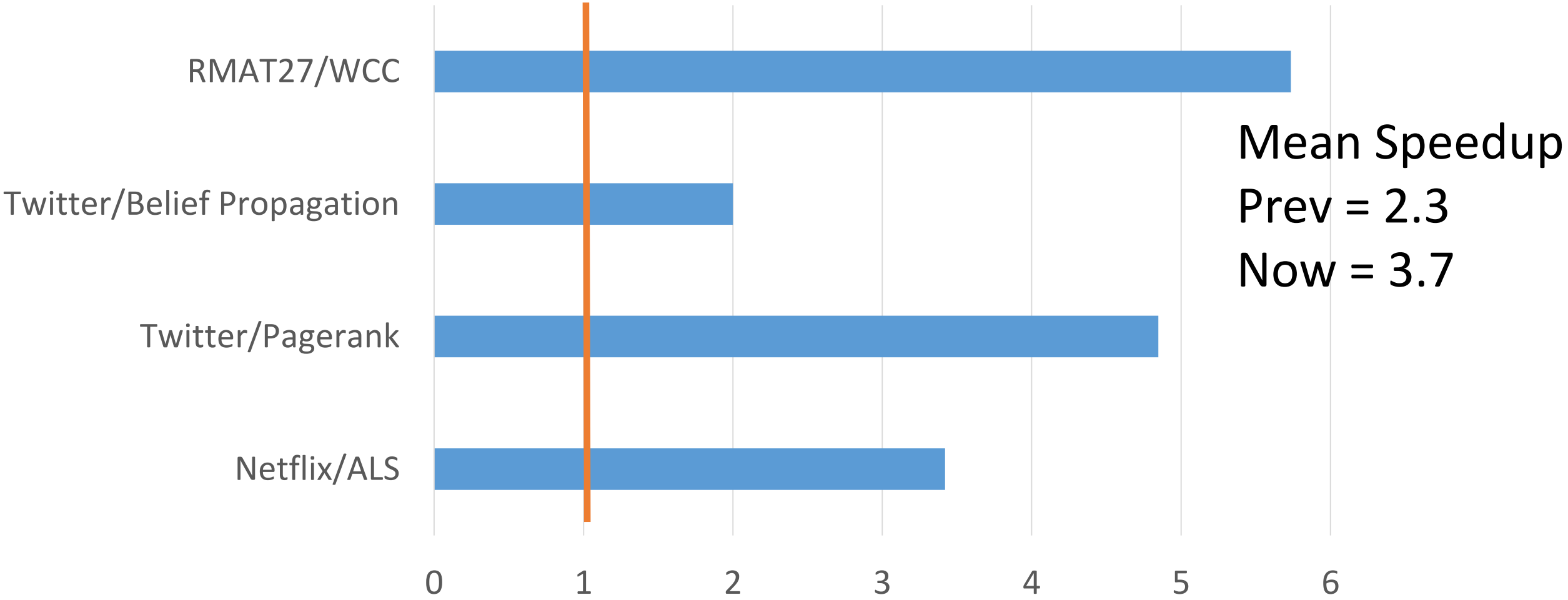


Baseline to Graphchi

- Replicated OSDI 2012 experiments on our SSD

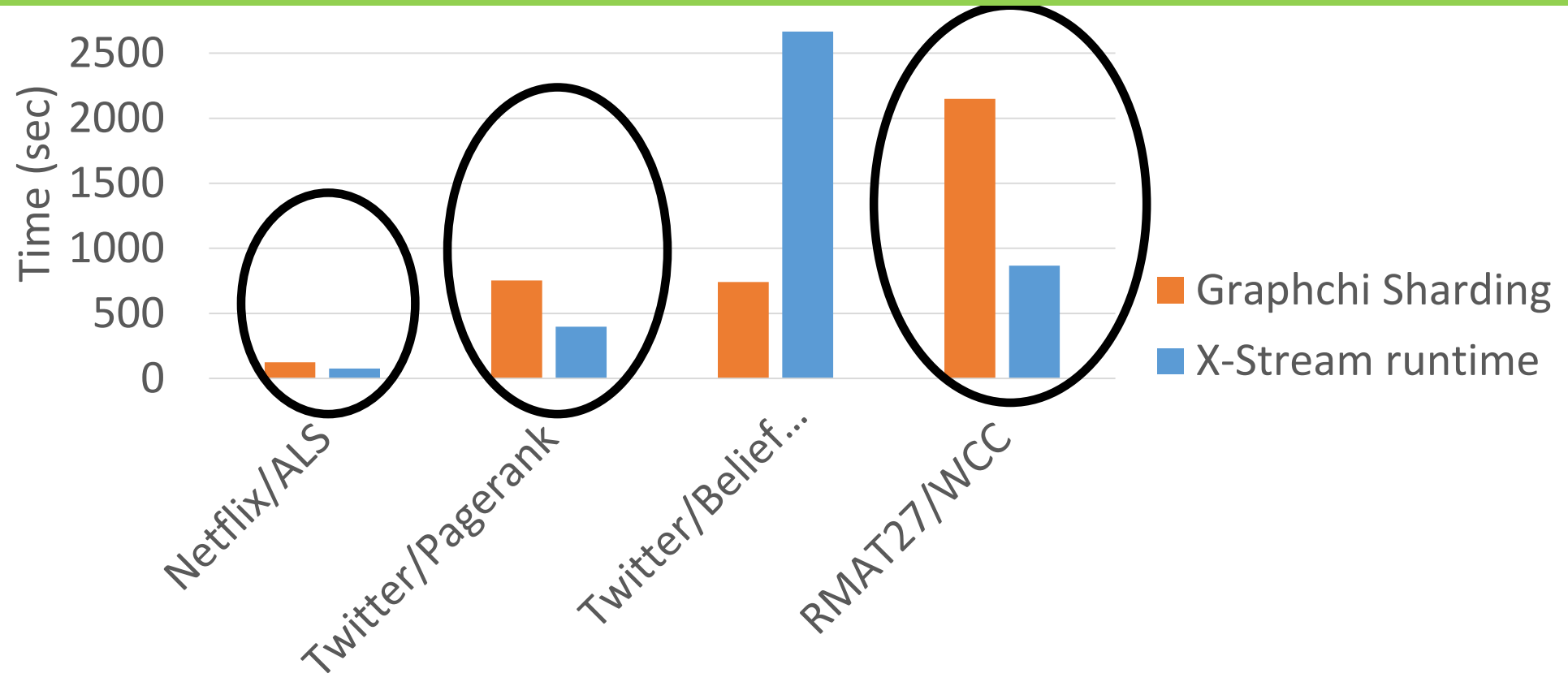


X-Stream Speedup over Graphchi (+ sharding)



Preprocessing Impact

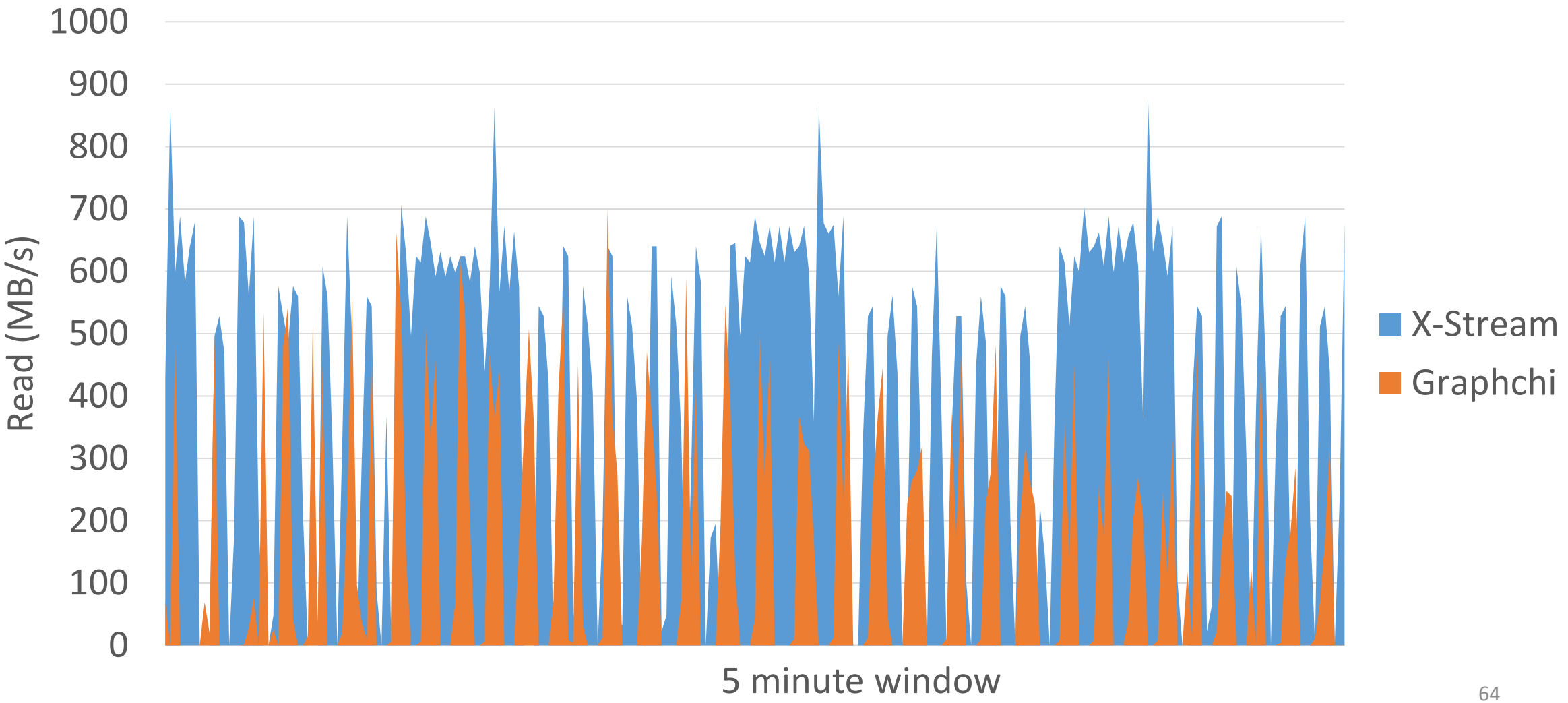
X-Stream returns answers before Graphchi finishes sharding



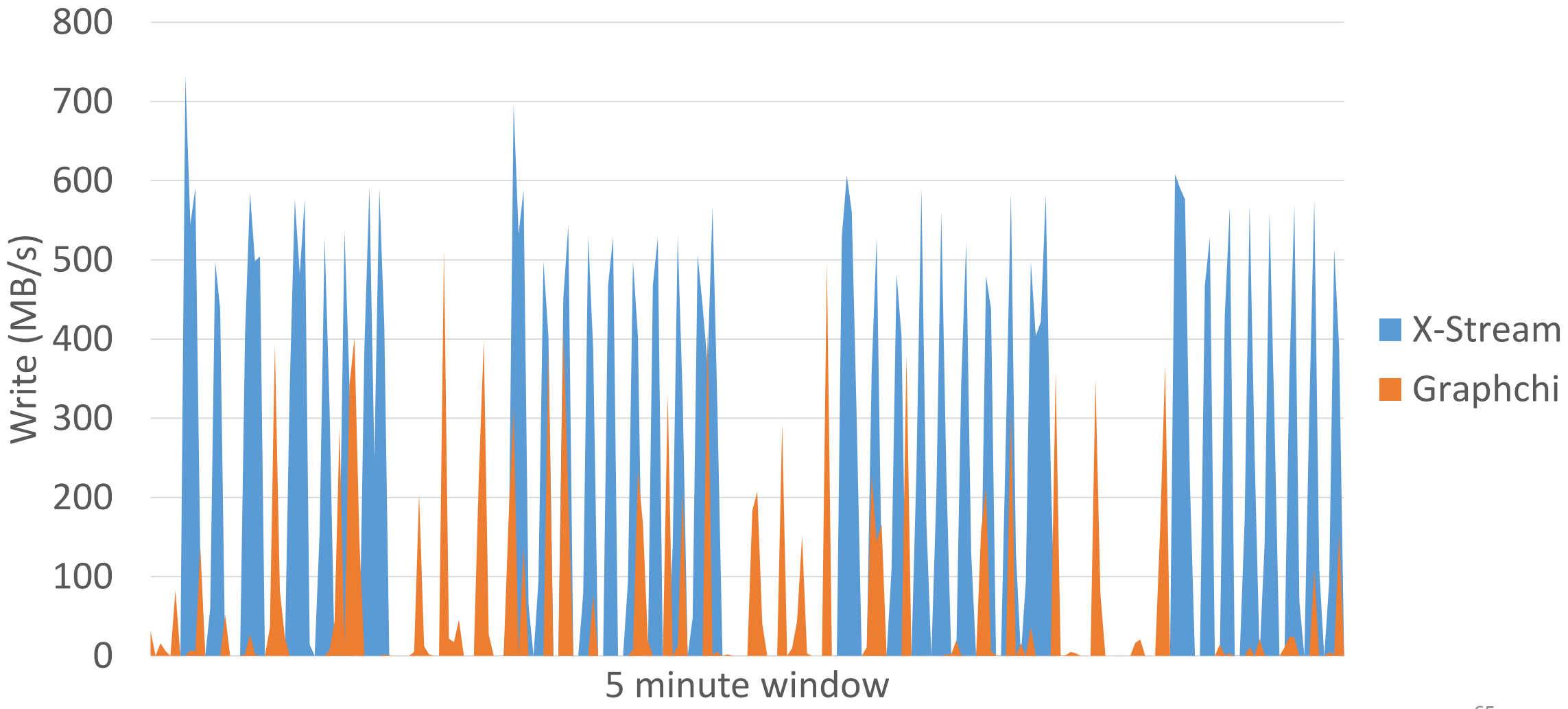
Sequential Access Bandwidth

- Graphchi shard
  - All vertices and edges must fit in memory
- X-Stream partition
  - Only vertices must fit in memory
- More Graphchi shards than X-Stream partitions
- Makes access more random for Graphchi

SSD Read Bandwidth (Pagerank on Twitter)



SSD Write Bandwidth (Pagerank on Twitter)



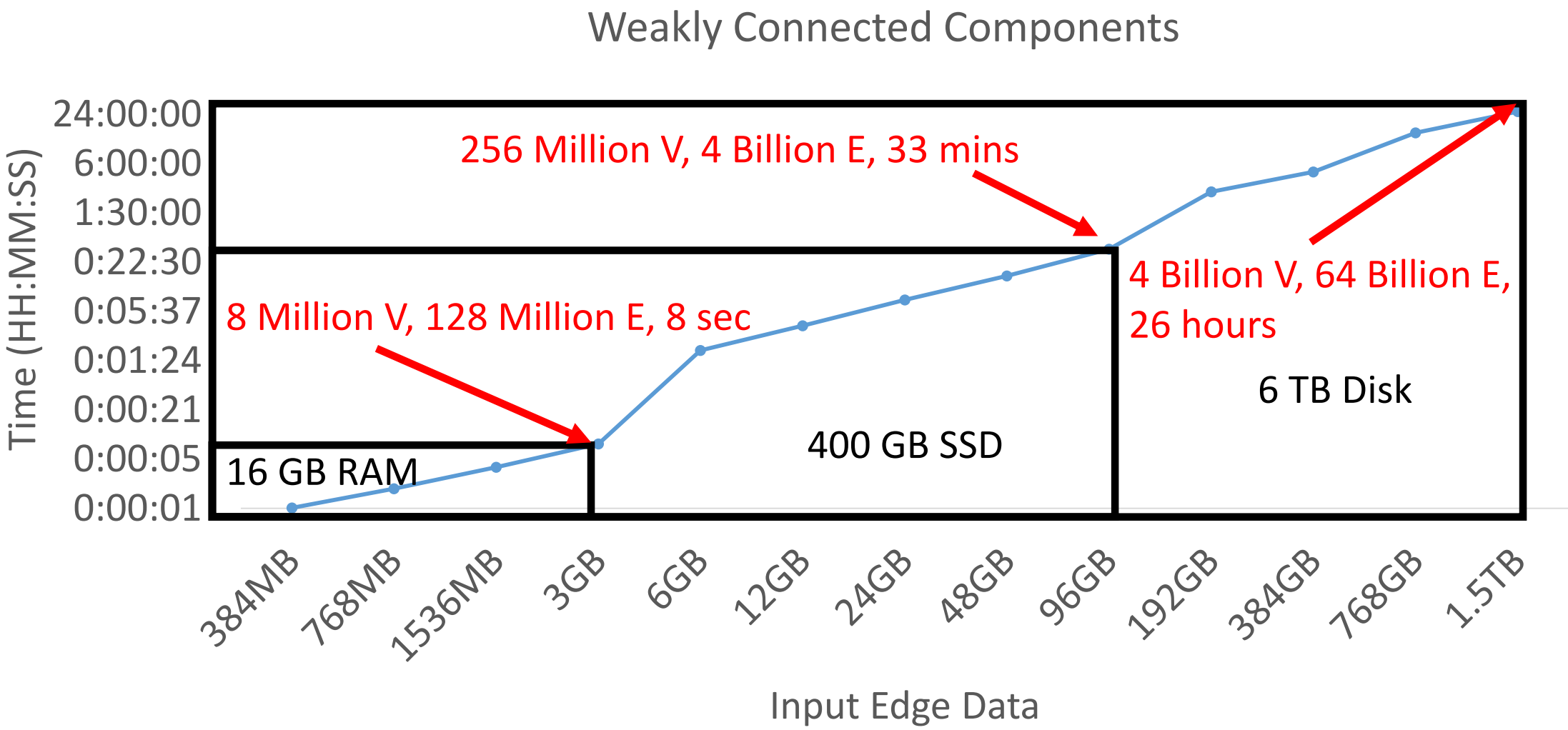
Disk Transfers (Pagerank on Twitter)

Metric	X-Stream	Graphchi
Data moved	224 GB	322 GB
Time taken	398 seconds	2613 seconds
Transfer rate	578 MB/s	126 MB/s

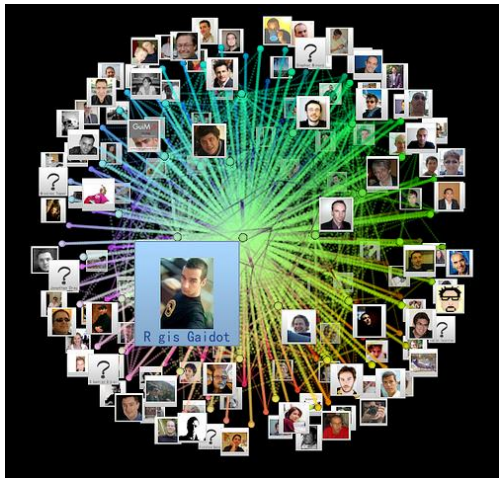
SSD can sustain reads = 667 MB/s, writes = 576 MB/s

X-Stream uses all available bandwidth from the storage device

Scaling up


Weakly Connected Components
Time (HH:MM:SS)
24:00:00
6:00:00
1:30:00
0:22:30
0:05:37
0:01:24
0:00:21
0:00:05
0:00:01
256 Million V, 4 Billion E, 33 mins
8 Million V, 128 Million E, 8 sec
4 Billion V, 64 Billion E, 26 hours
16 GB RAM
400 GB SSD
6 TB Disk
384MB
768MB
1536MB
3GB
6GB
12GB
24GB
48GB
96GB
192GB
384GB
768GB
1.5TB
Input Edge Data

Conclusion



X-Stream

Edge-centric processing
+
Streaming Partitions
=
Sequential Access

Big graphs

Good Performance
RAM, SSD, Disk

Download from http://labos.epfl.ch/xstream

BACKUP

API Restrictions

- Updates must be commutative
- Cannot access all edges from a vertex in single step

Applications

- X-Stream can solve a variety of problems

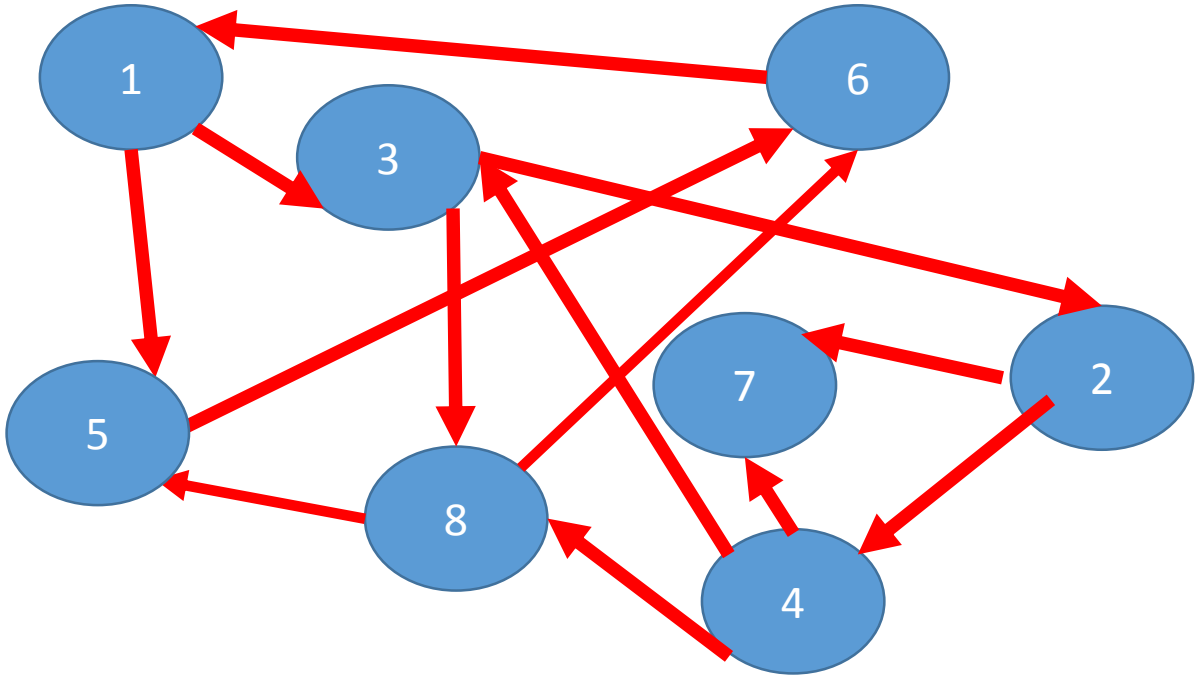
BFS, SSSP, Weakly connected components, Strongly connected components, Maximal independent sets, Minimum cost spanning trees, Belief propagation, Alternating least squares, Pagerank, Betweenness centrality, Triangle counting, Approximate neighborhood function, Conductance, K-Cores

Q. Average distance between people on a social network ?

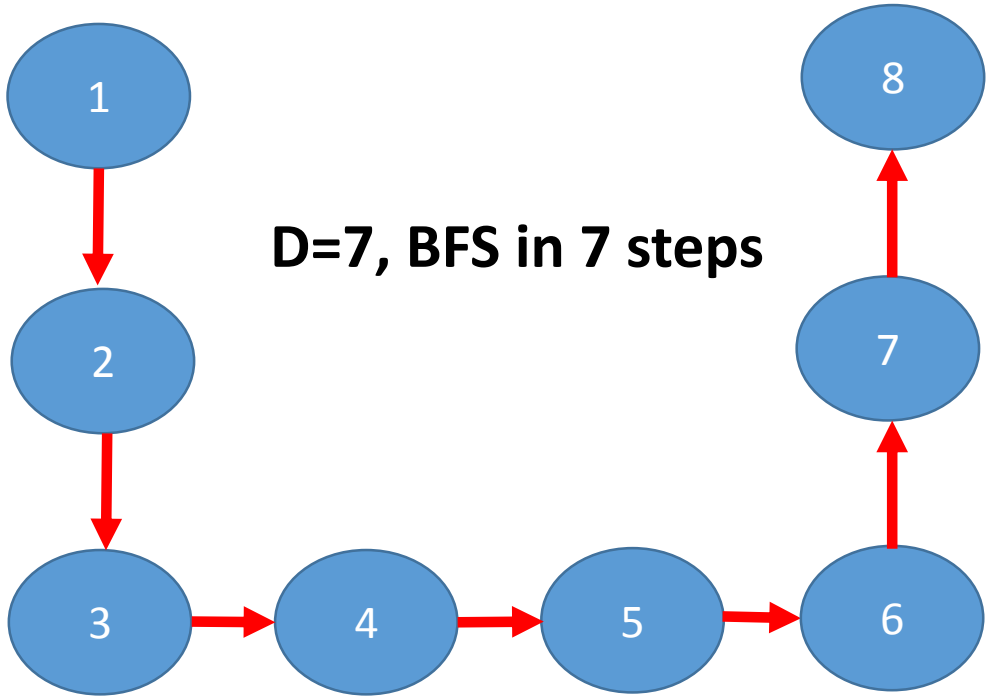
A. Use approximate neighborhood function.

Edge-centric Scatter Gather

- **Real world graphs have low diameter**

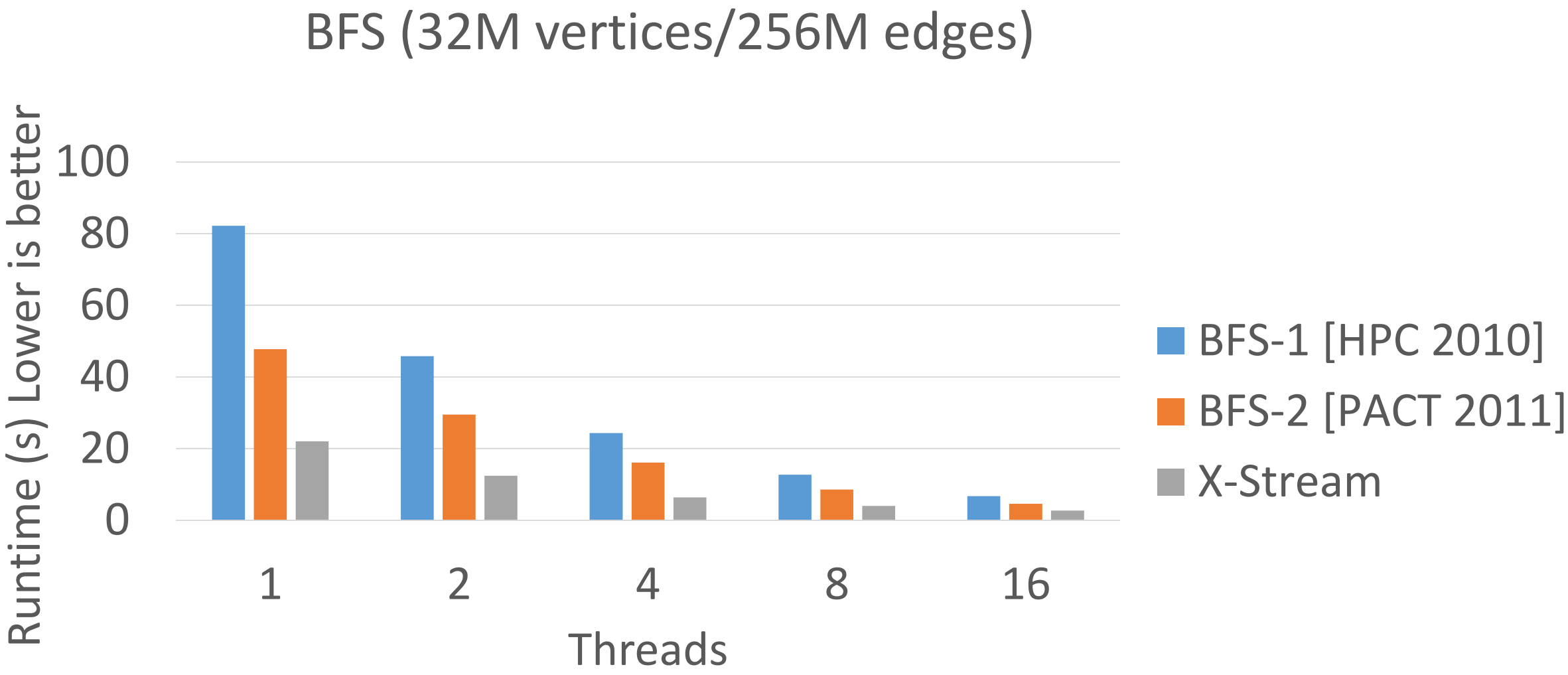


D=3, BFS in 3 steps, Most real-world graphs

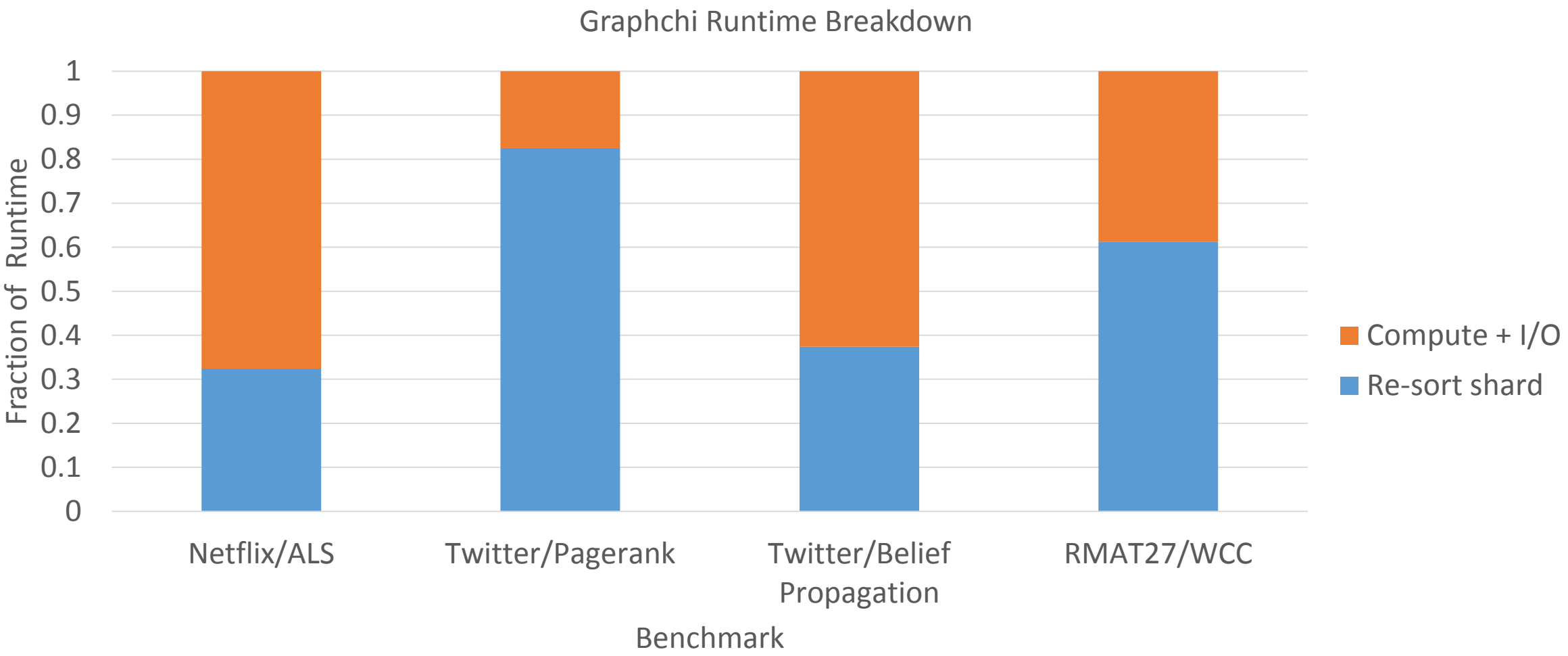


D=7, BFS in 7 steps

X-Stream Main Memory Performance



Runtime impact of Graphchi Sharding



Pre-processing Overhead

- Low overhead for producing streaming partition
- **Strictly cheaper than sorting edges by source vertex**