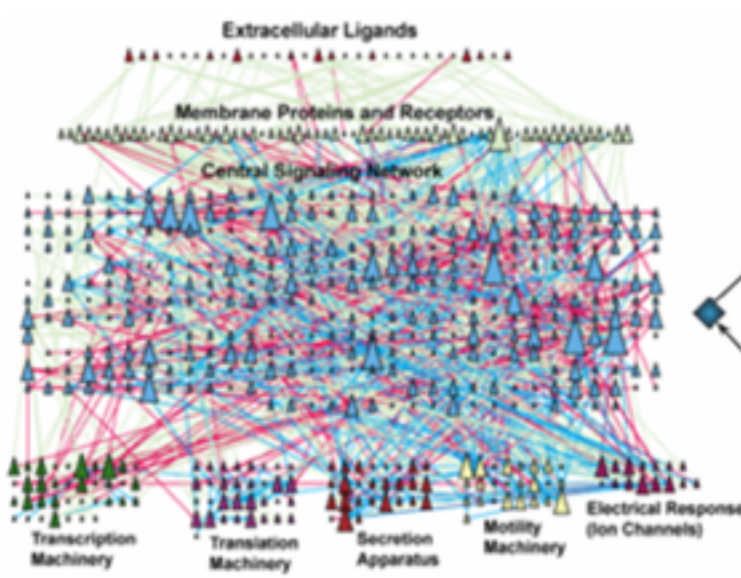
Tackling Large Graphs with Secondary Storage

Amitabha Roy
EPFL

Graphs

Social networks

Document networks



Biological networks

Humans, phones, bank accounts

Graph are Difficult

- Graph mining is challenging problem
- Traversal leads to data-dependent accesses
  - Little predictability
  - Hard to parallelize efficiently

Tackling Large Graphs

- Normal approach
- Throw resources at the problem
- What does it take to process a trillion edges ?

Big Iron

HPC/Graph500 benchmarks (June 2014)

Graph Edges	Hardware
1 trillion	Tsubame
1 trillion	Cray
1 trillion	Blue Gene
1 trillion	NEC

Large Clusters

Avery Ching,
Facebook
@Strata, 2/13/2014

Yes, using 3940 machines

Big Data

- Data is growing exponentially
  - 40 Zettabytes by 2020
- Unlikely you can put it all in DRAM
  - Need PM, SSD, Magnetic disks
  - Secondary storage != DRAM
- Also applicable to graphs

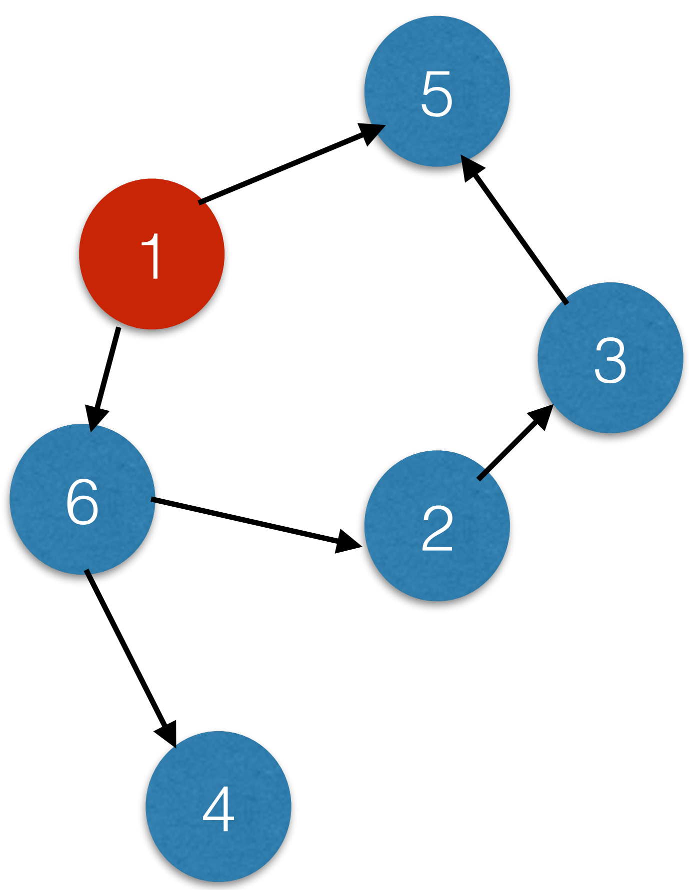
Motivation

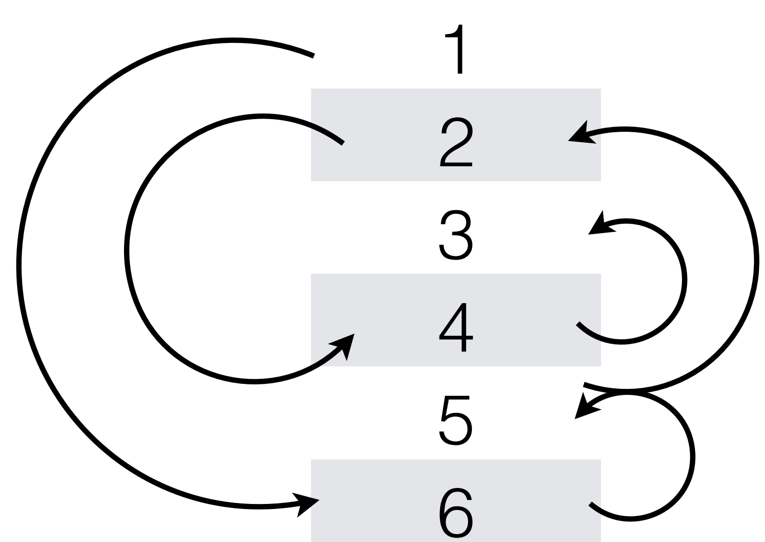
If I can store the graph then why can't I process it ?

- 32 machines x 2TB magnetic disk = 64 TB storage
- 1 trillion edges x 16 bytes per edge = 16 TB storage

Problem #1

- Irregular access patterns





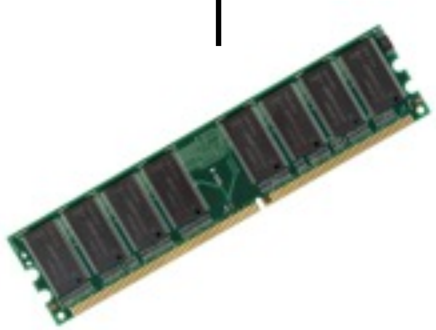
Problem #1

- Random access penalties

200X

20X

1.4X



RAM

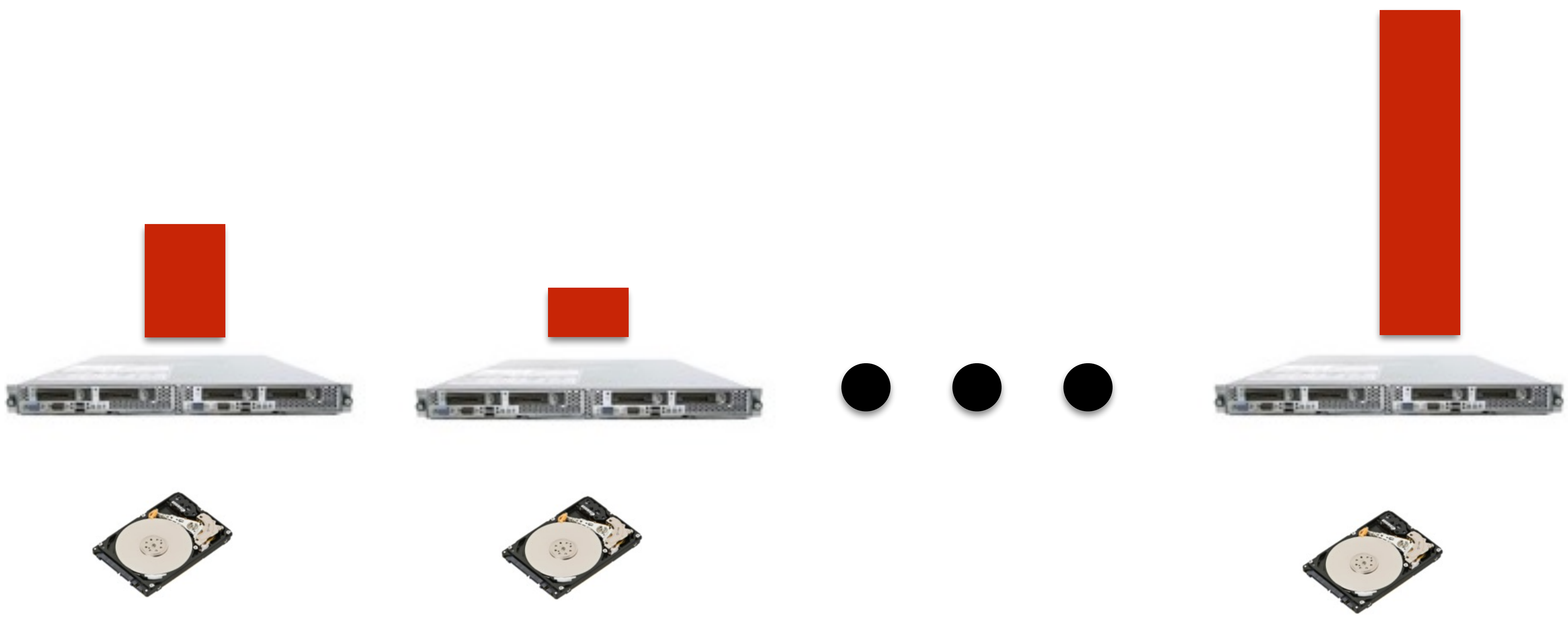
SSD

Disk

2ms seeks on a graph with a trillion edges ~ 1 year !

Problem #2

- Partitioning graphs across machines is hard
- Random partitions very poor for real-world graphs



Twitter graph: 20X difference with 32 machines !

Outline

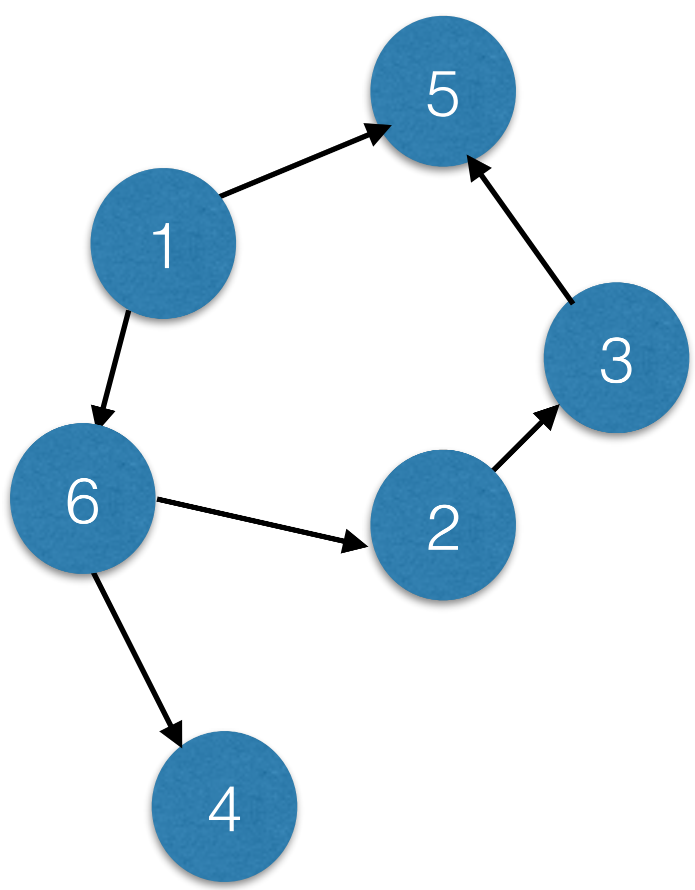
- X-Stream (address problem #1)
- SlipStream (address problem #2)

X-Stream

- Single machine graph processing system [SOSP'13]
- Turns graph processing into sequential access
  - Change computation model
  - Partitioning of graph

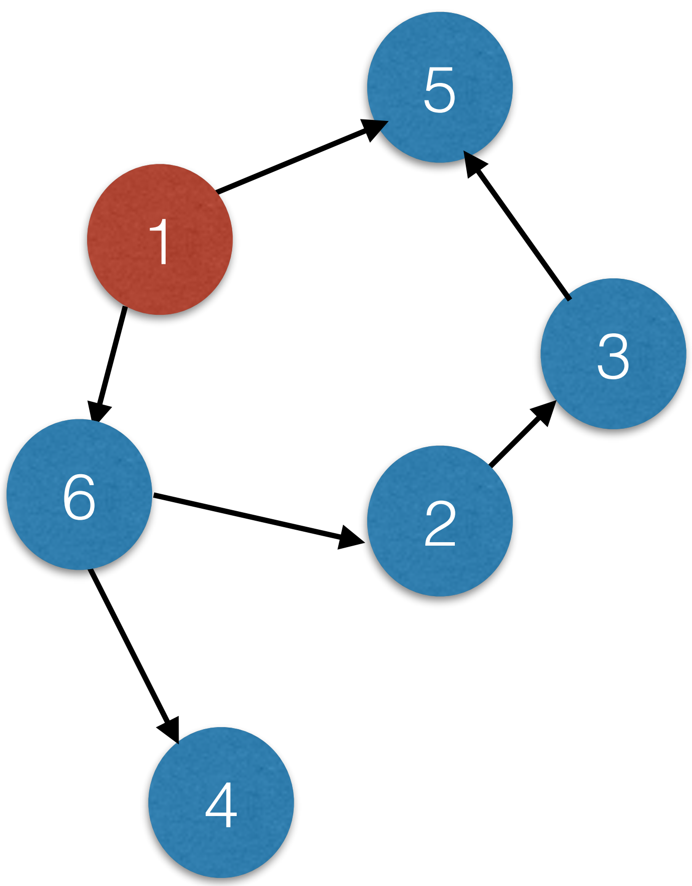
Scatter-Gather

Existing computational model



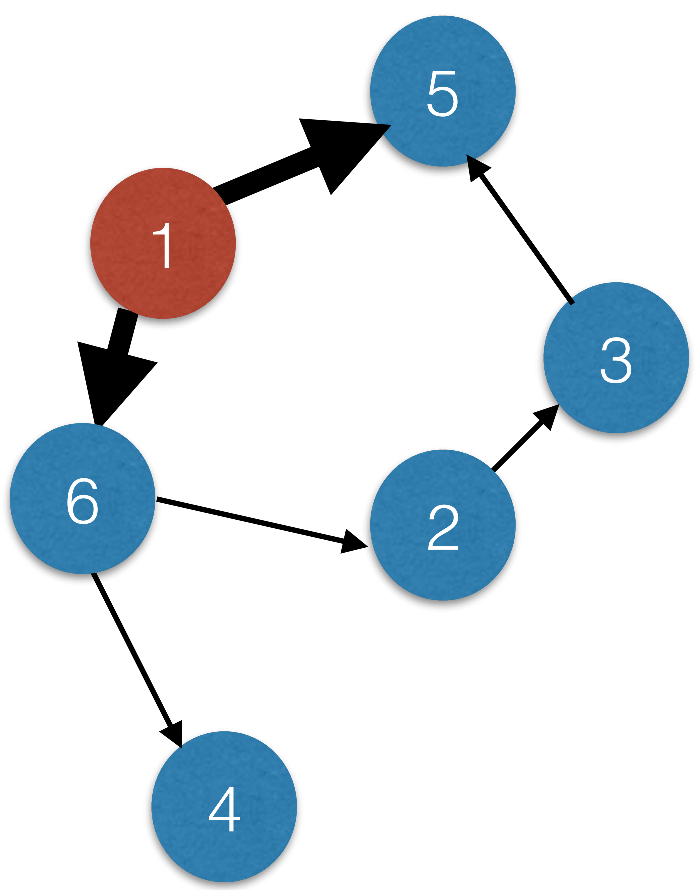
Scatter-Gather

Activate vertex



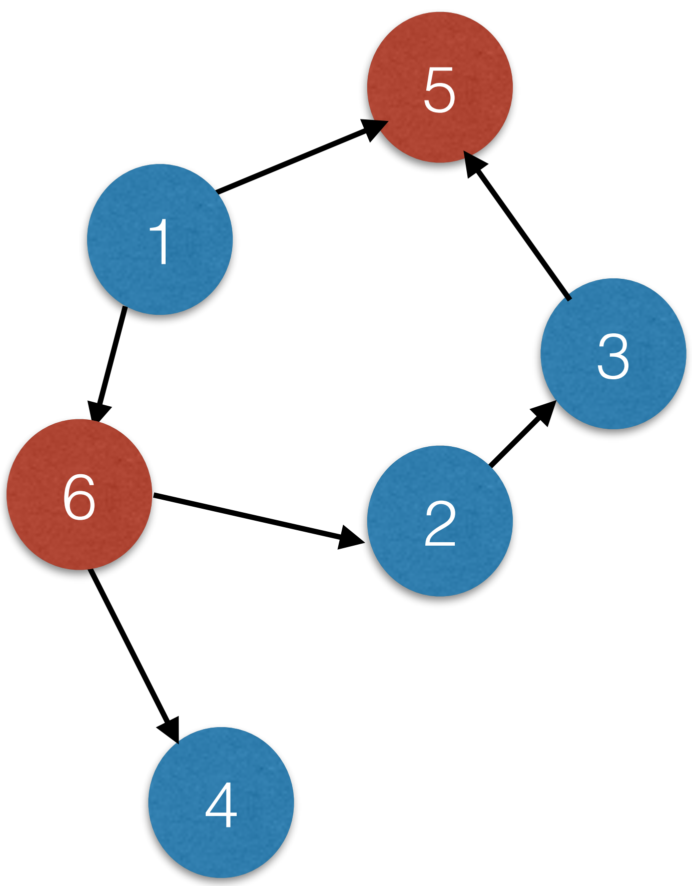
Scatter-Gather

Scatter Updates

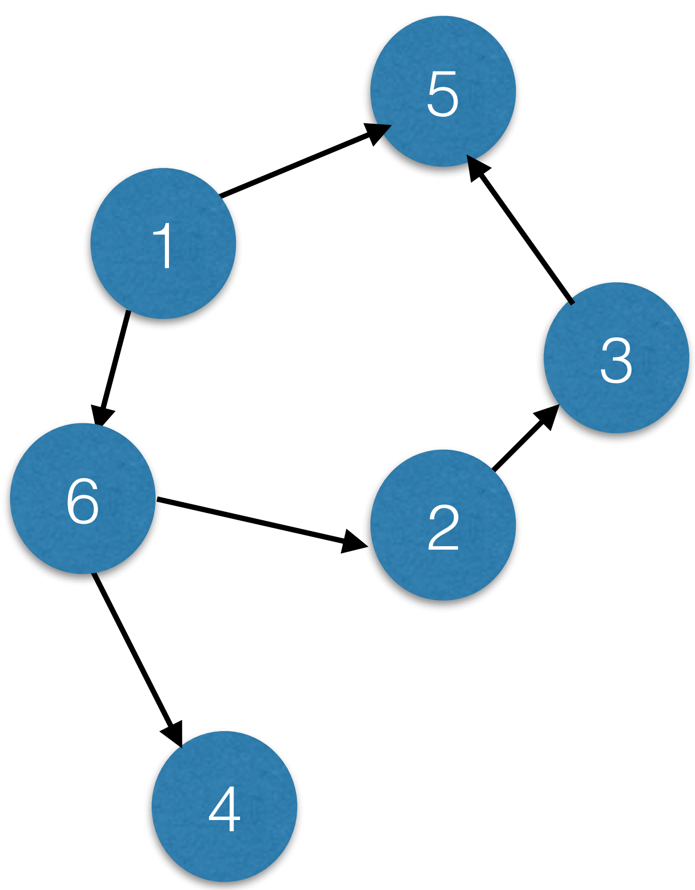


Scatter-Gather

Gather Updates



Storage

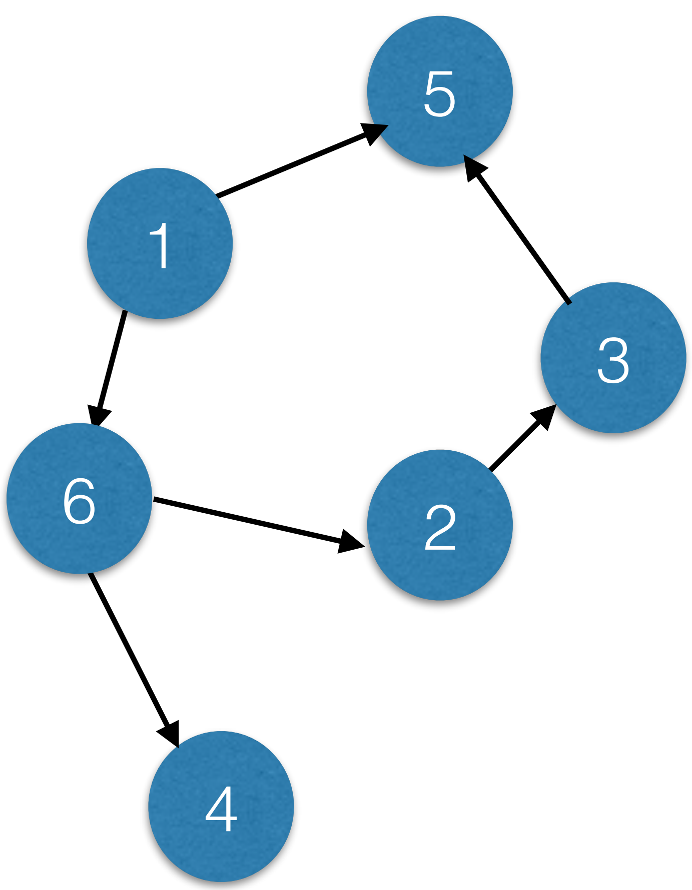


Edges

1 → 5
1 → 6
⋮
6 → 2
6 → 4

Vertices

Edge File

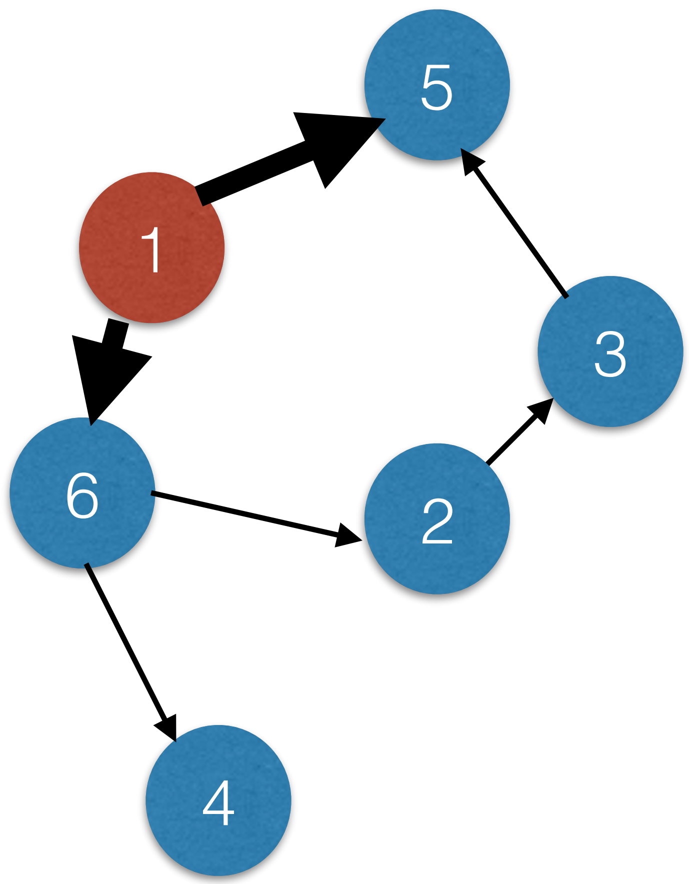


Edges

1 → 5
1 → 6
⋮
6 → 2
6 → 4

Vertices

Edge File



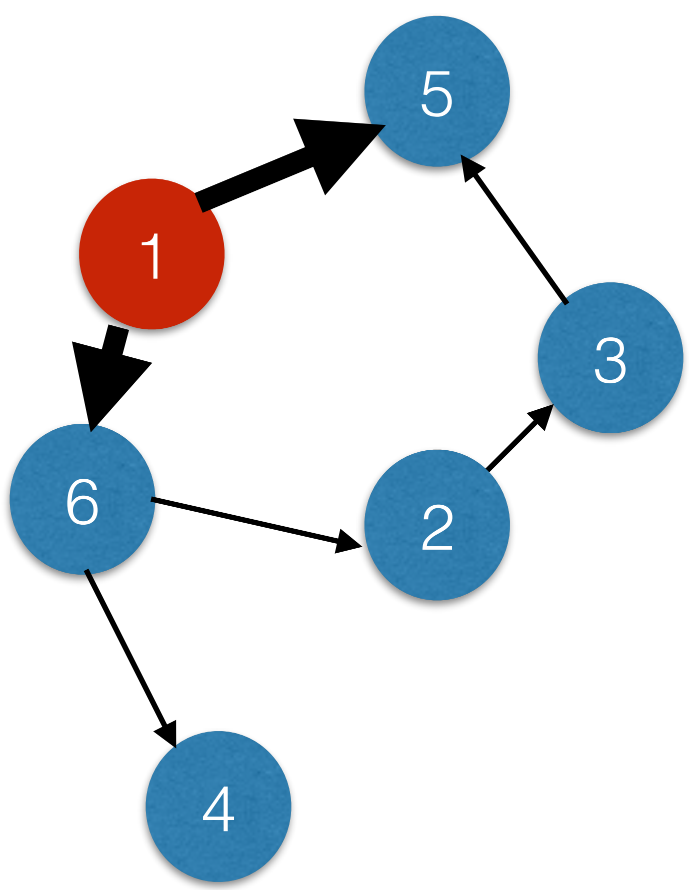
1 → 5
1 → 6

SEEK ⋮

6 → 2
6 → 4

Edge-centric Scatter-Gather

Scan entire edge list

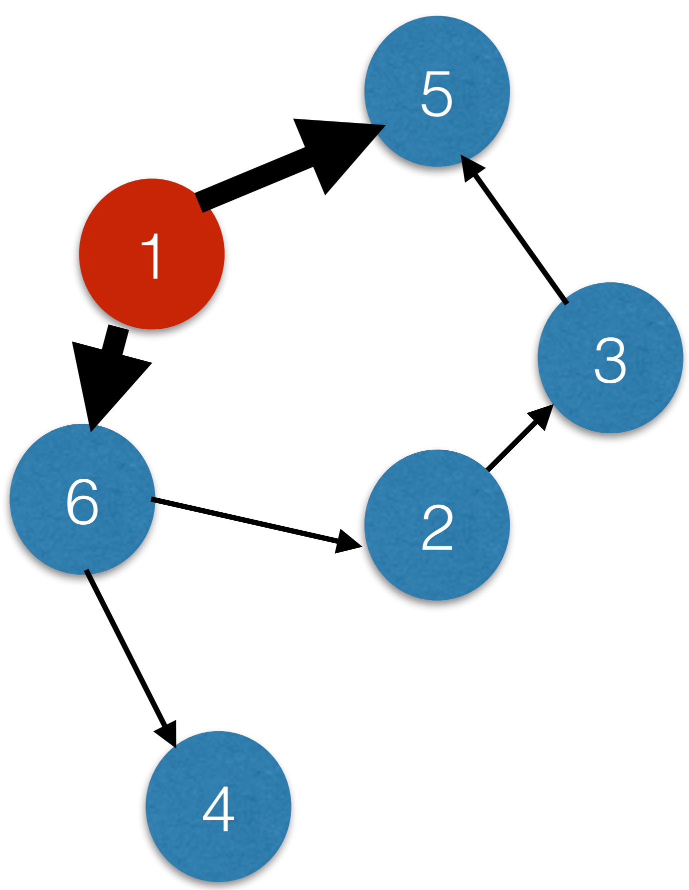


SCAN

1 → 5
1 → 6
⋮
6 → 2
6 → 4

Edge-centric Scatter-Gather

Use only necessary edges



SCAN

1 → 5
1 → 6

•
•
•

6 → 2
6 → 4

Tradeoff

✔ Achieve sequential bandwidth

✖ Need to scan entire edge list

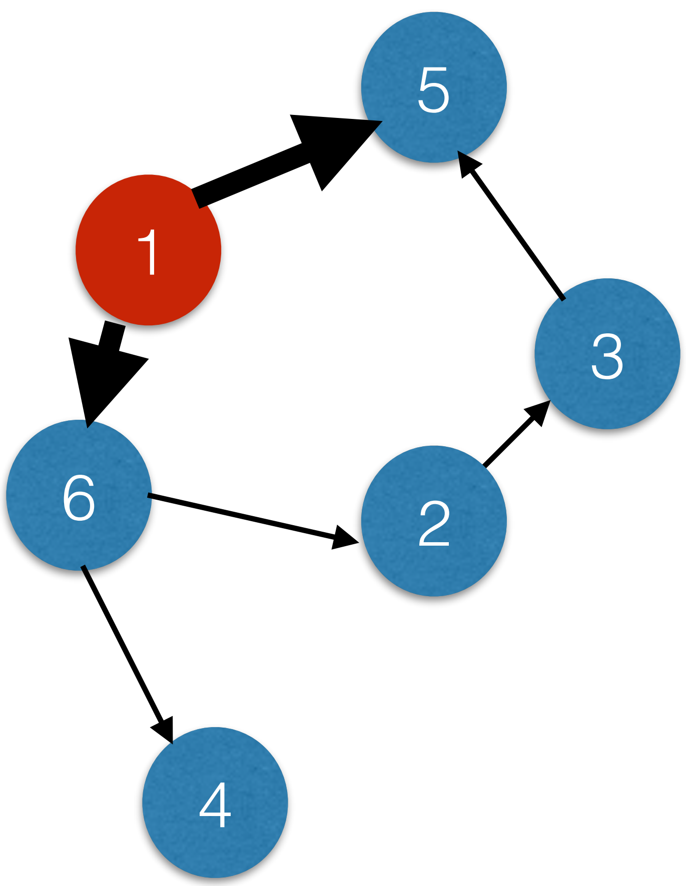
Winning Tradeoff !

Winning Tradeoff

- Real-world graphs have small diameter
- Traversals in just a few iterations of scatter-gather
- Large number of active vertices in most iterations

Benefit

Order oblivious



SCAN

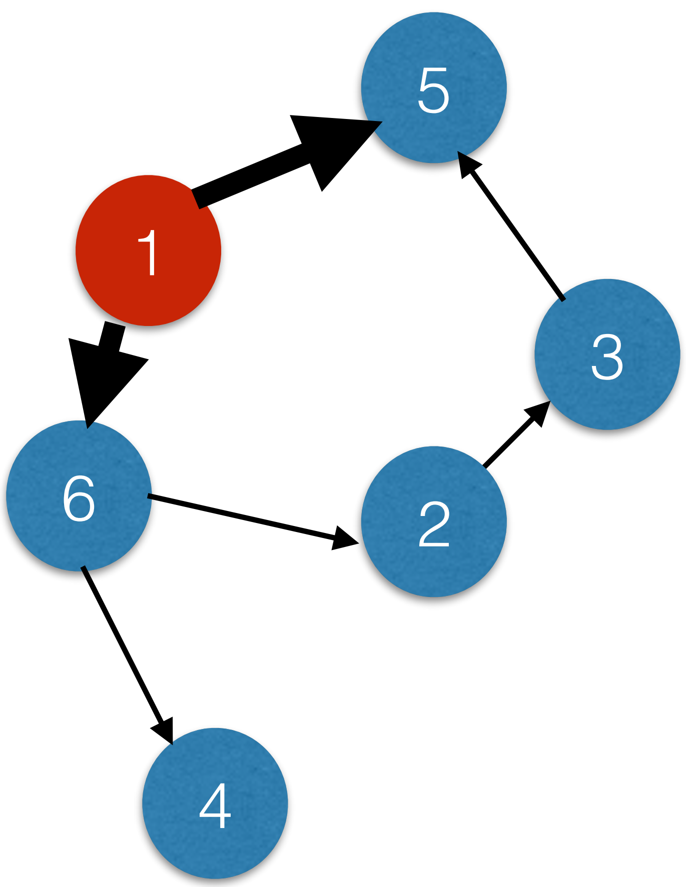
$\boxed{1 \rightarrow 5}$

$6 \rightarrow 4$

$\boxed{1 \rightarrow 6}$

$6 \rightarrow 2$

What about the vertices ?



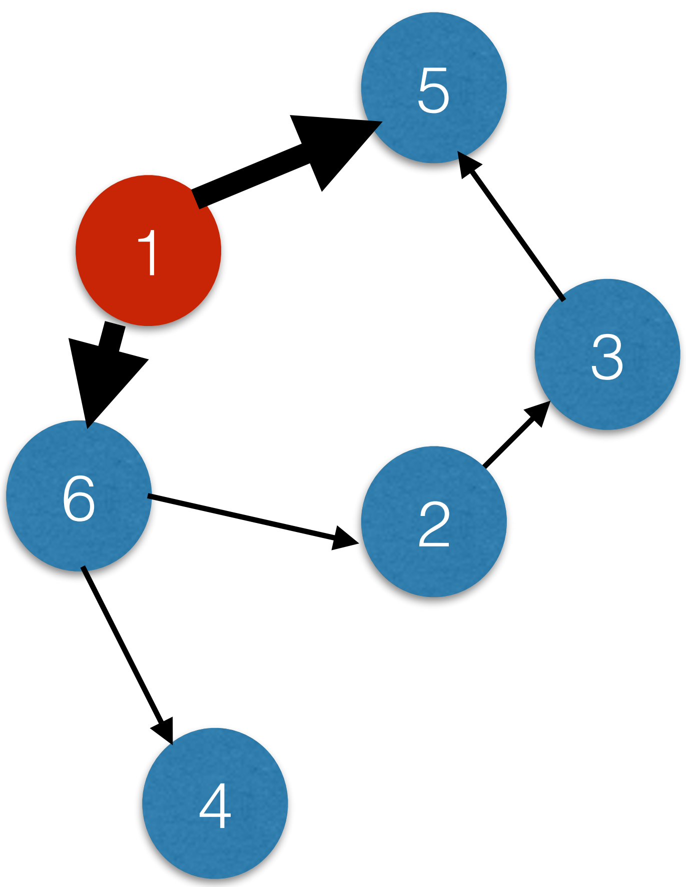
SCAN

1 → 5
1 → 6
•
•
•
6 → 2
6 → 4

SEEK

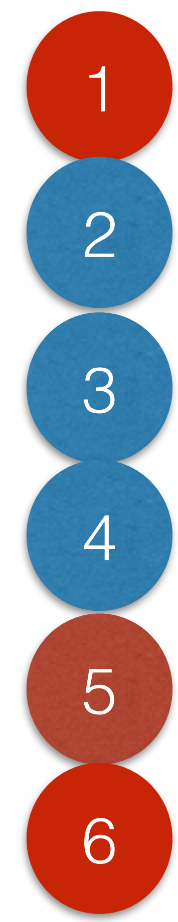
What about the vertices ?

Seeking in RAM is free !
How can we fit vertices in RAM ?



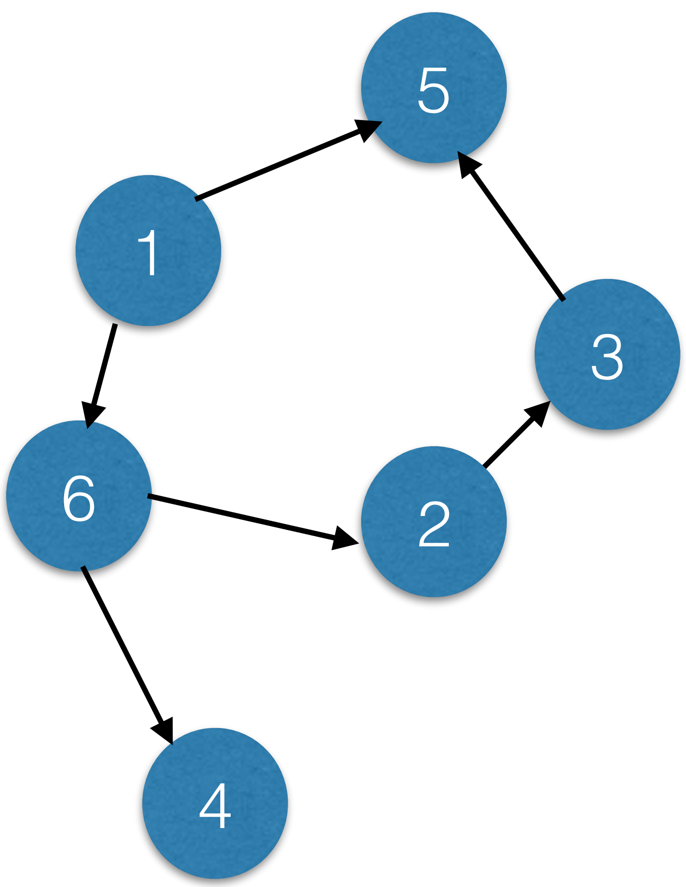
SCAN

1 → 5
1 → 6
⋮
6 → 2
6 → 4



SEEK

Streaming Partitions

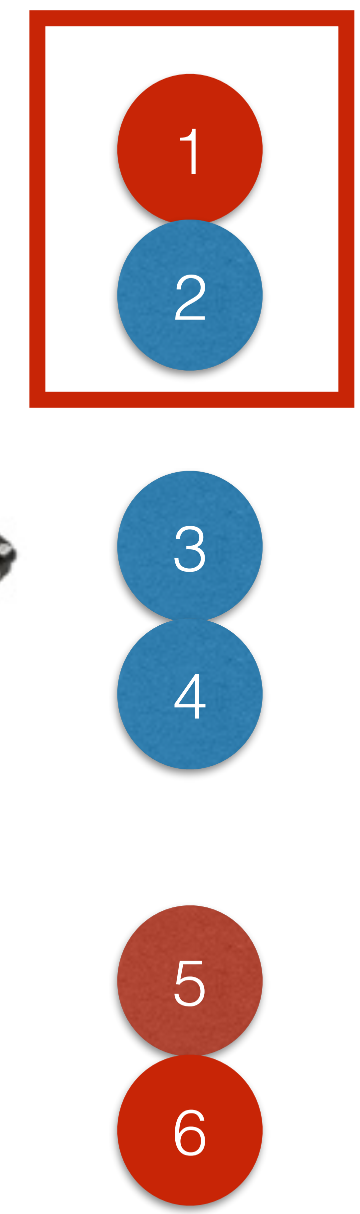


1 → 5
1 → 6
2 → 3

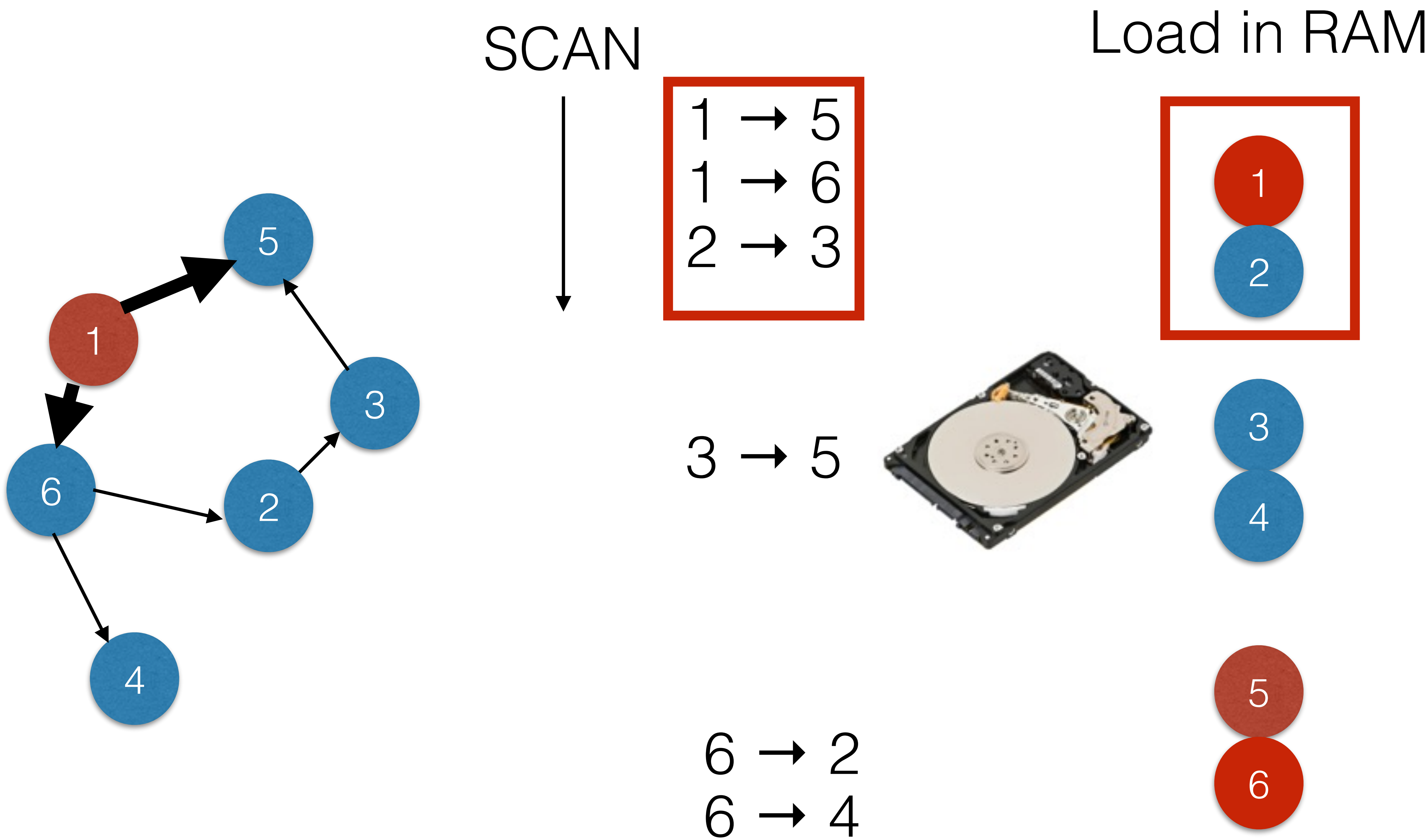
3 → 5

6 → 2
6 → 4

Fits in RAM



Streaming Partitions



Producing Partitions

- No requirement on quality (# of cross edges)
  - Need only fit into RAM
  - Random partitions are great
- Random partitions work great

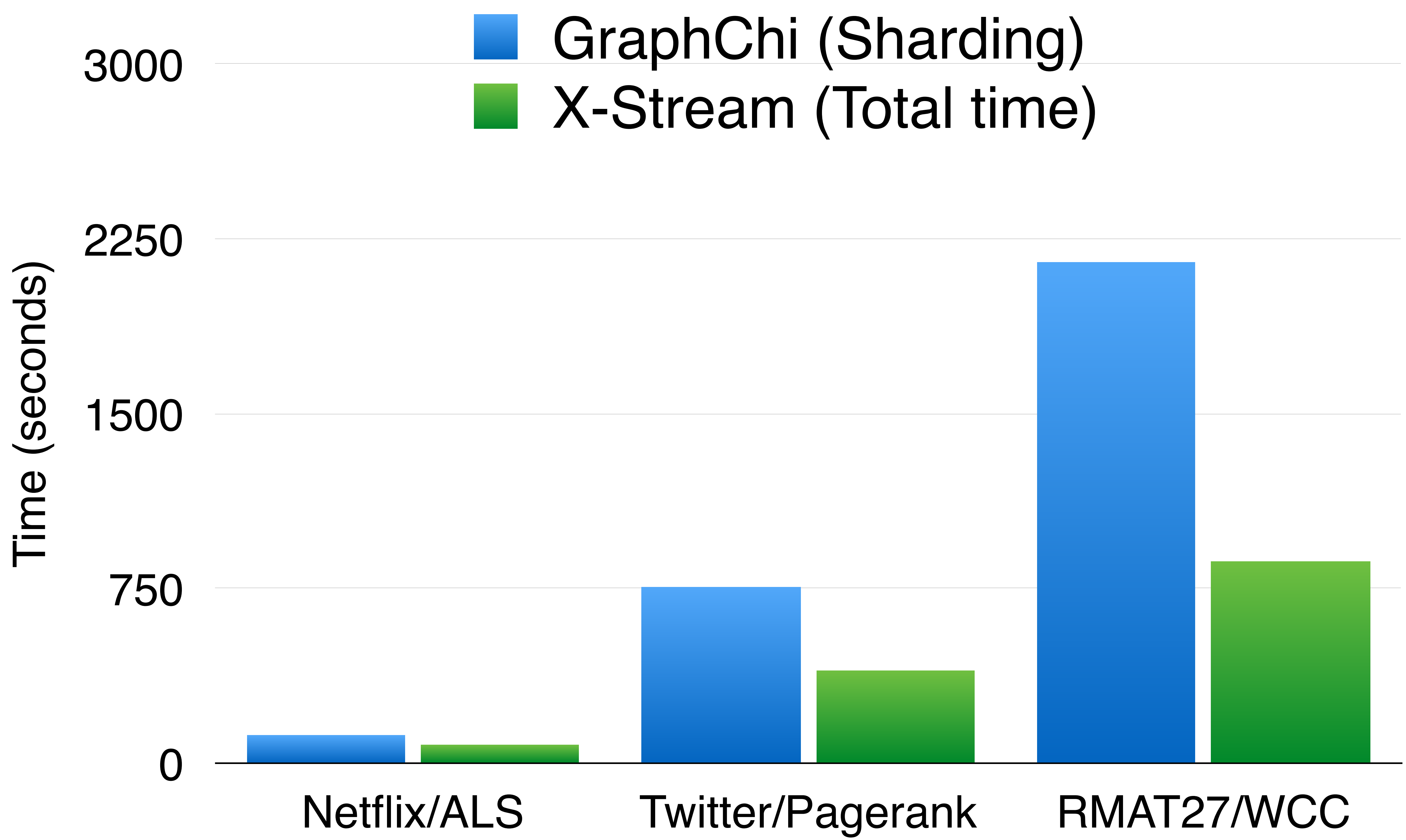
Algorithms Supported

- Supports traversal algorithms
  - BFS, WCC, MIS, SCC, K-Cores, SSSP, BC
- Supports algebraic operations on the graph
  - BP, ALS, SpMV, Pagerank
- <u>Good testbed for newer streaming algorithms</u>
  - HyperANF, Semi-streaming Triangle Counting

Competition

- Graphchi
  - Another on-disk graph processing system (OSDI'12)
  - Special on-disk data structure: shards
  - Makes accesses look sequential
- Producing shards requires sorting edges

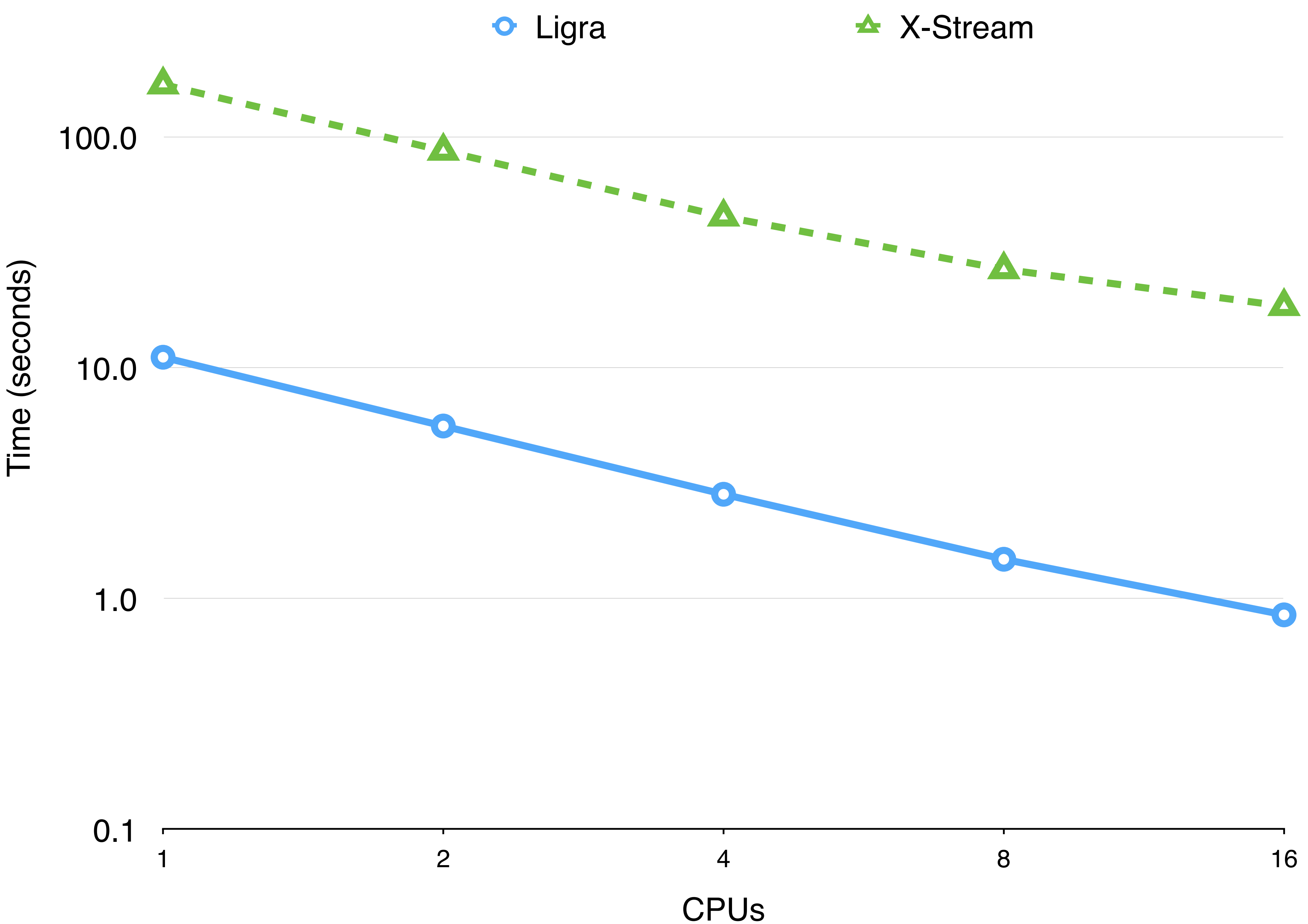
SSD


GraphChi (Sharding)
X-Stream (Total time)
Time (seconds)
3000
2250
1500
750
0
Netflix/ALS
Twitter/Pagerank
RMAT27/WCC

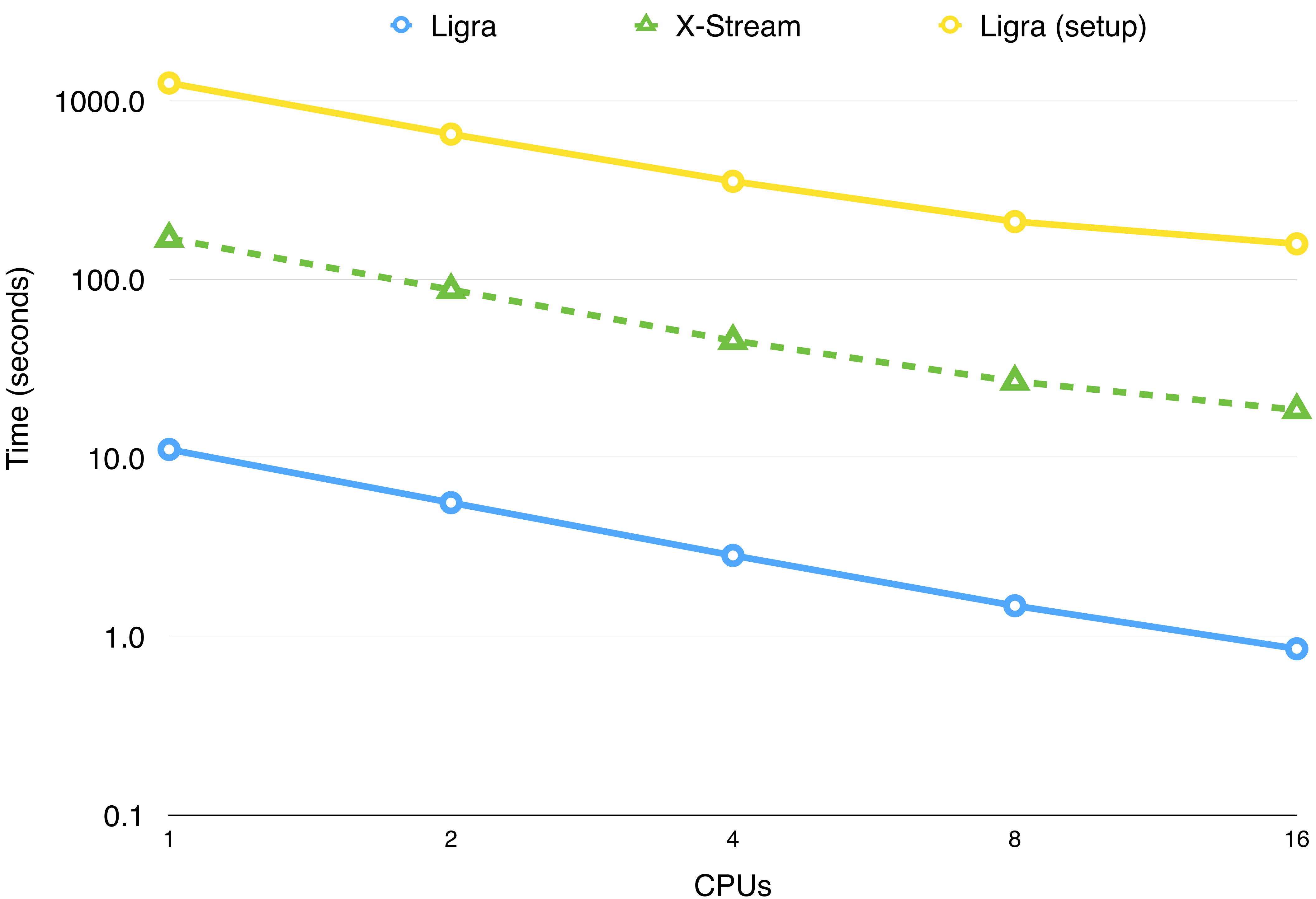
More Competition

- Applies to any two level memory
- Includes CPU cache and DRAM
- Main memory graph processing ?
- Looked at Ligra (PPoPP 2012)

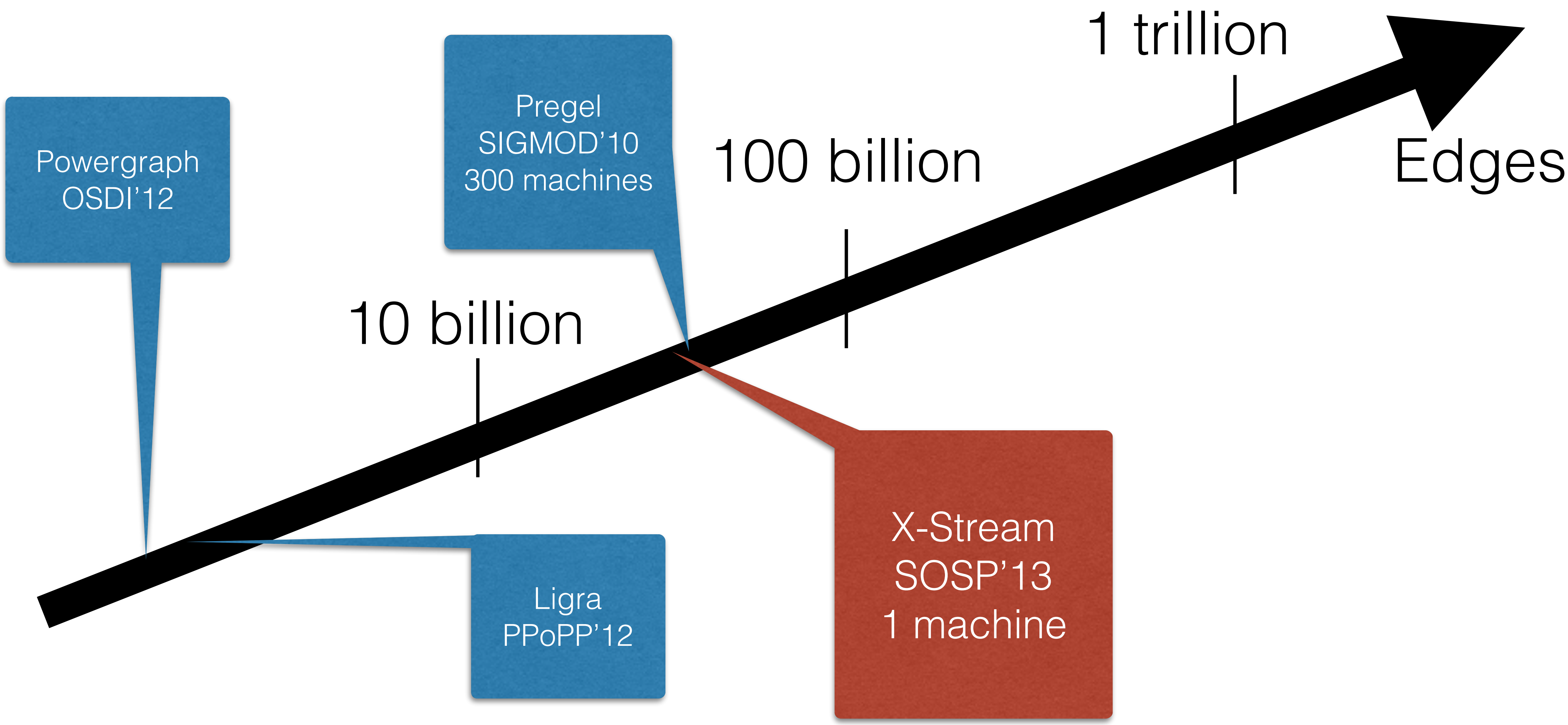
BFS



BFS


Ligra
X-Stream
Ligra (setup)
1000.0
100.0
10.0
1.0
0.1
Time (seconds)
1
2
4
8
16
CPUs

Where we stand



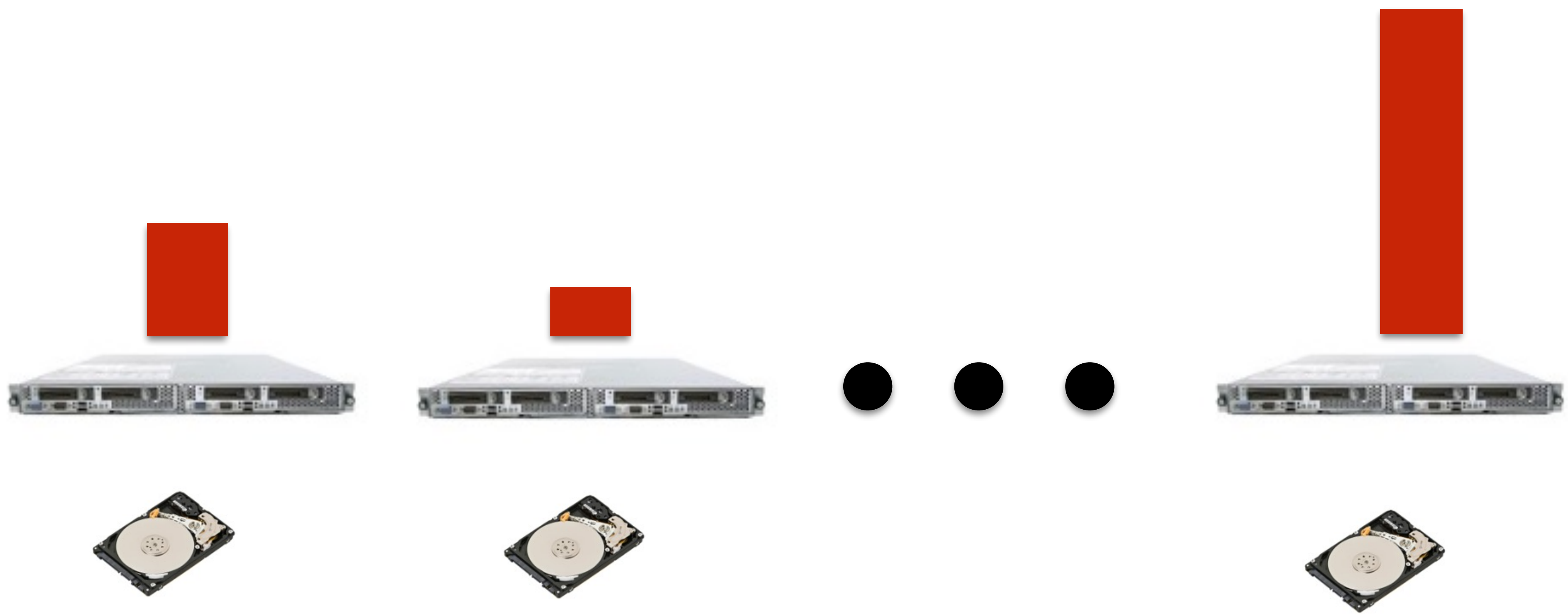
How do we get further ? Scale out

SlipStream

- Aggregate bandwidth and storage of a cluster
- Solves the graph partitioning problem
  - Rethinking storage access
  - Rethinking streaming partition execution
- We know how to do it right for one machine

Scaling Out

- Assign different streaming partitions to machines



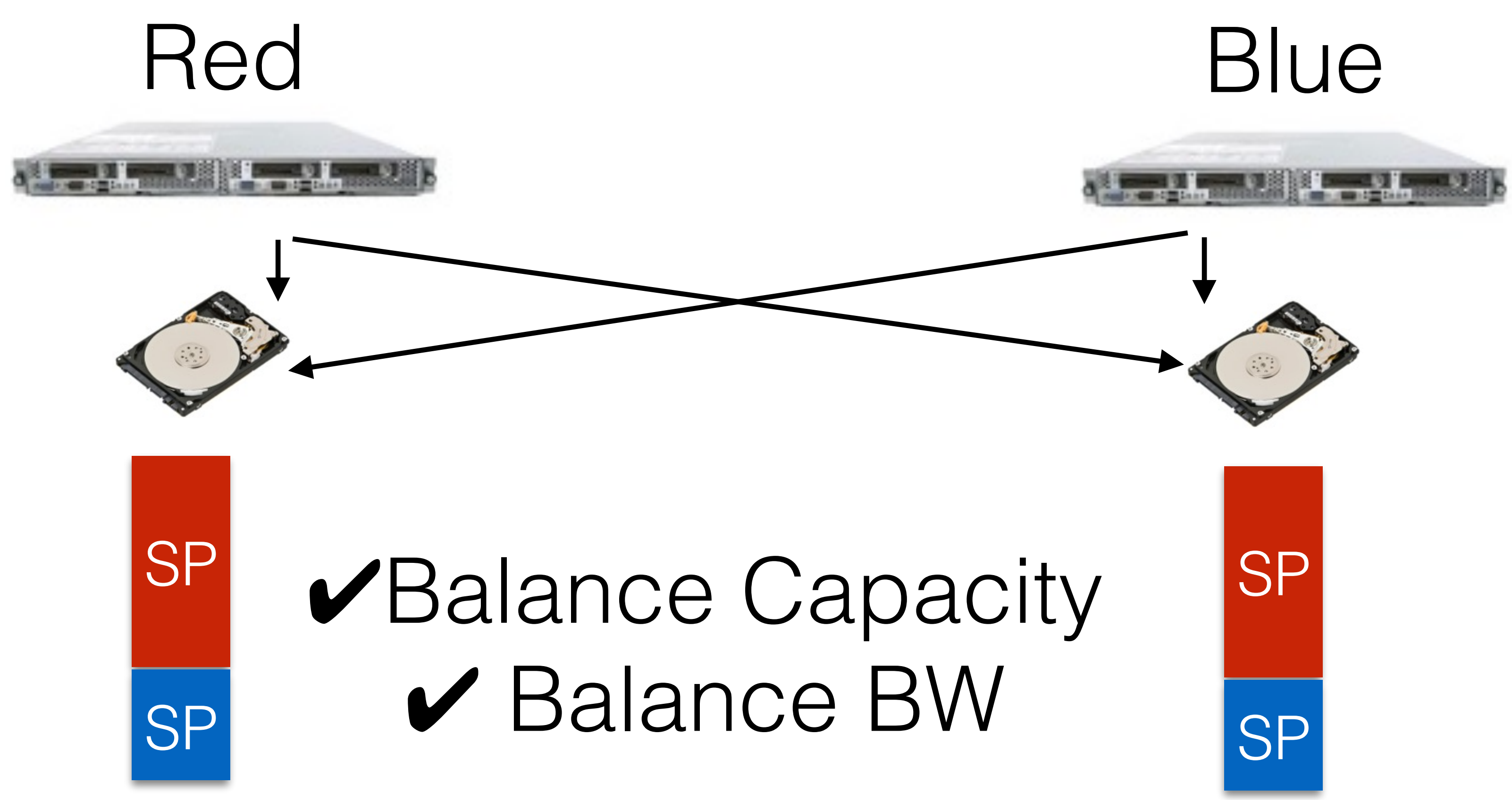
Graph partitioning is hard to get right

Load Imbalance

Load Imbalance

Flat Storage

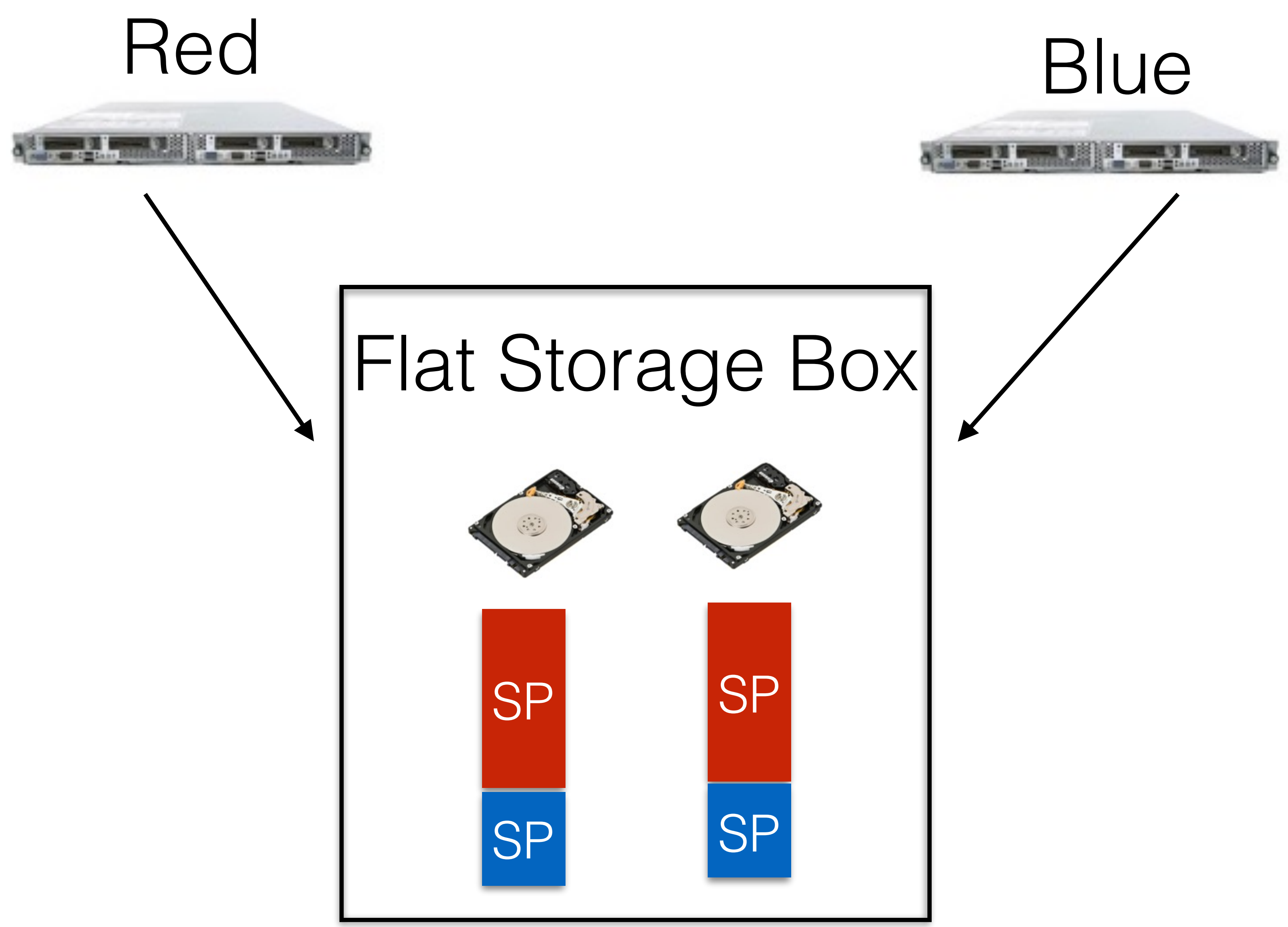
Stripe data across all disks
Allow any machine to access any disk



✔Balance Capacity
✔ Balance BW

Flat Storage

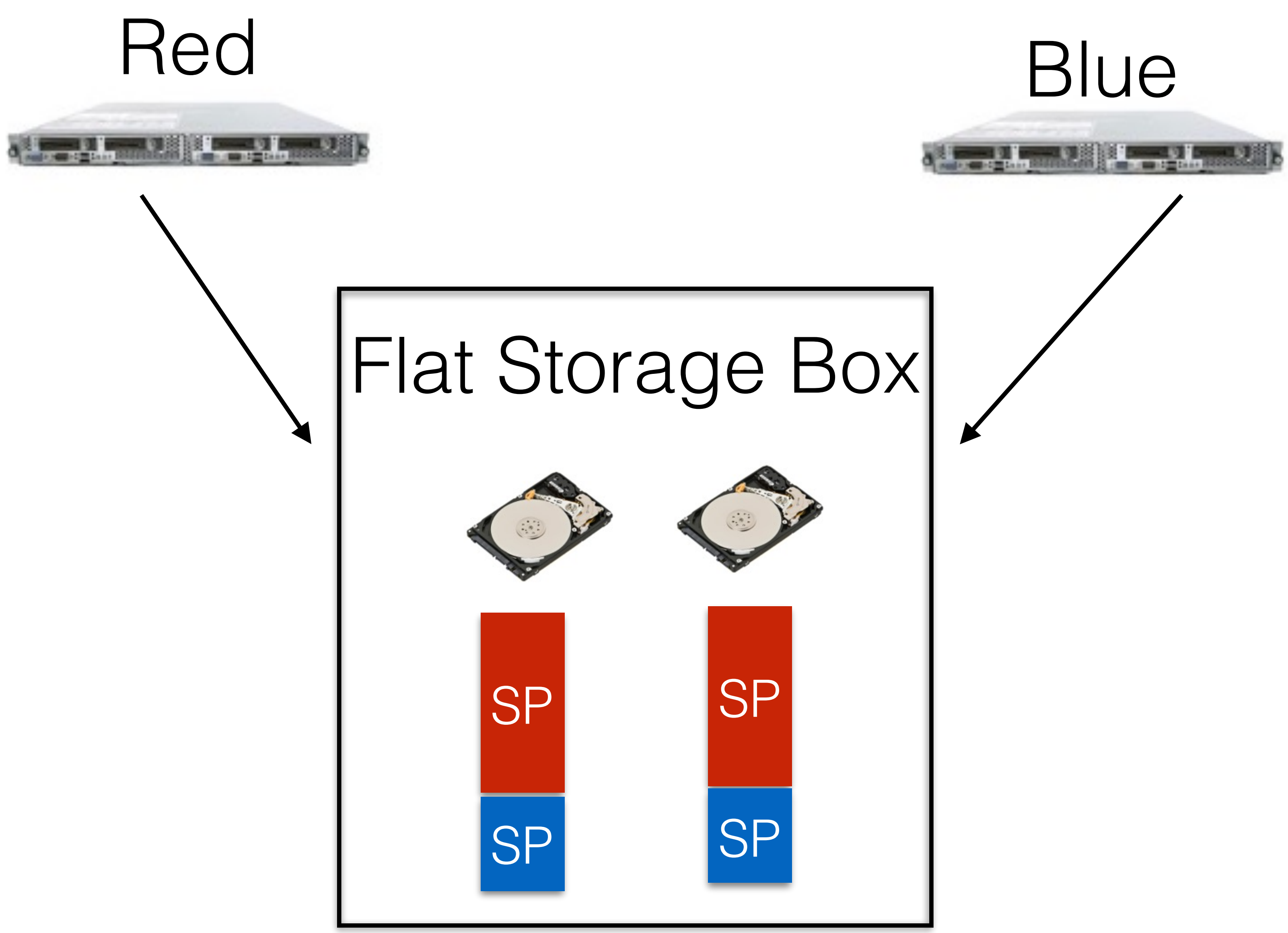
Stripe data across all disks
Allow any machine to access any disk



Flat Storage

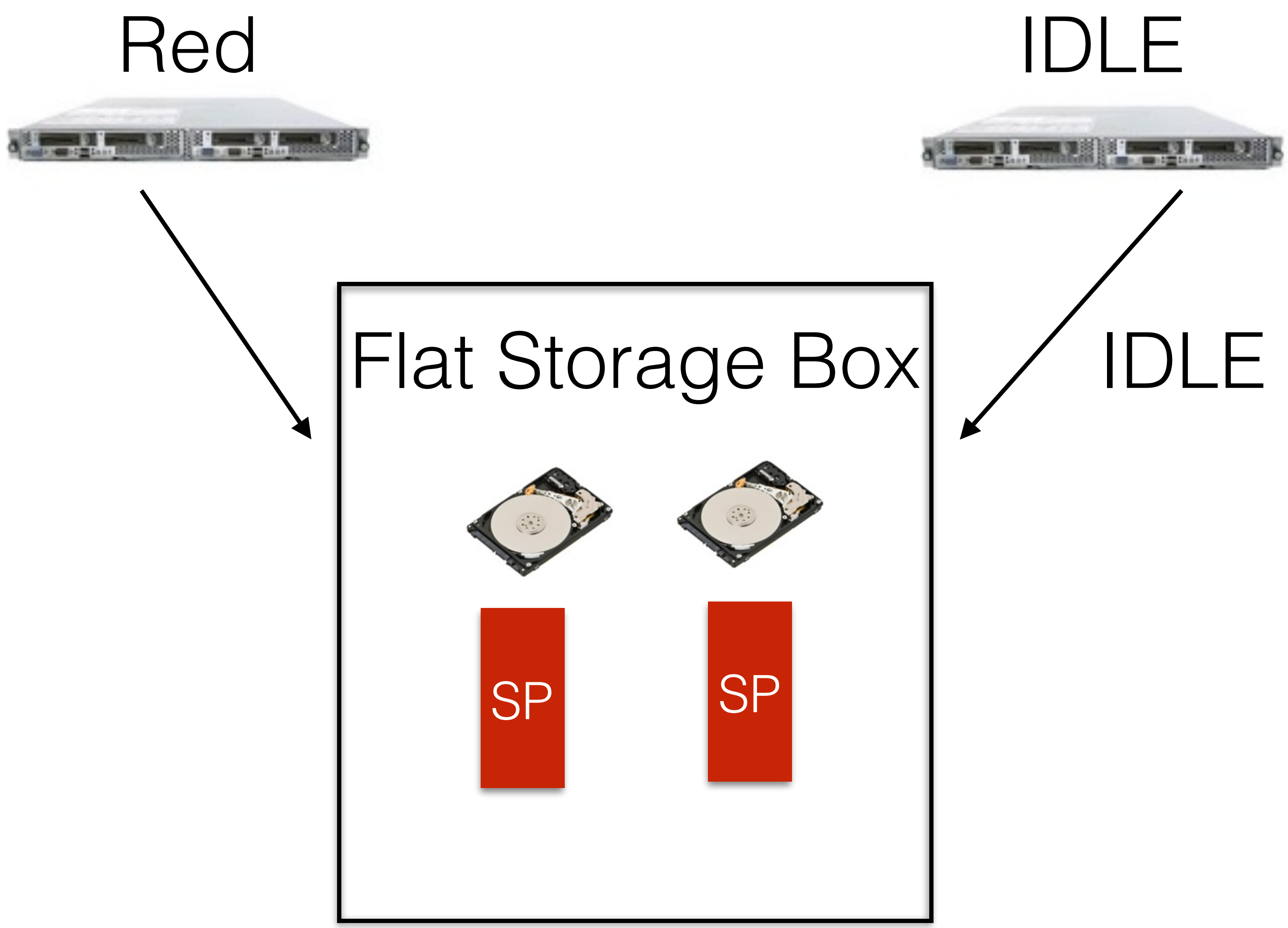
- Assumes full bisection bandwidth network
- Can be done at data-center scales
- Nightingale et. al. OSDI 2012 using CLOS switches
- Already true at rack scale
    - Like in our cluster

Flat Storage



Flat Storage

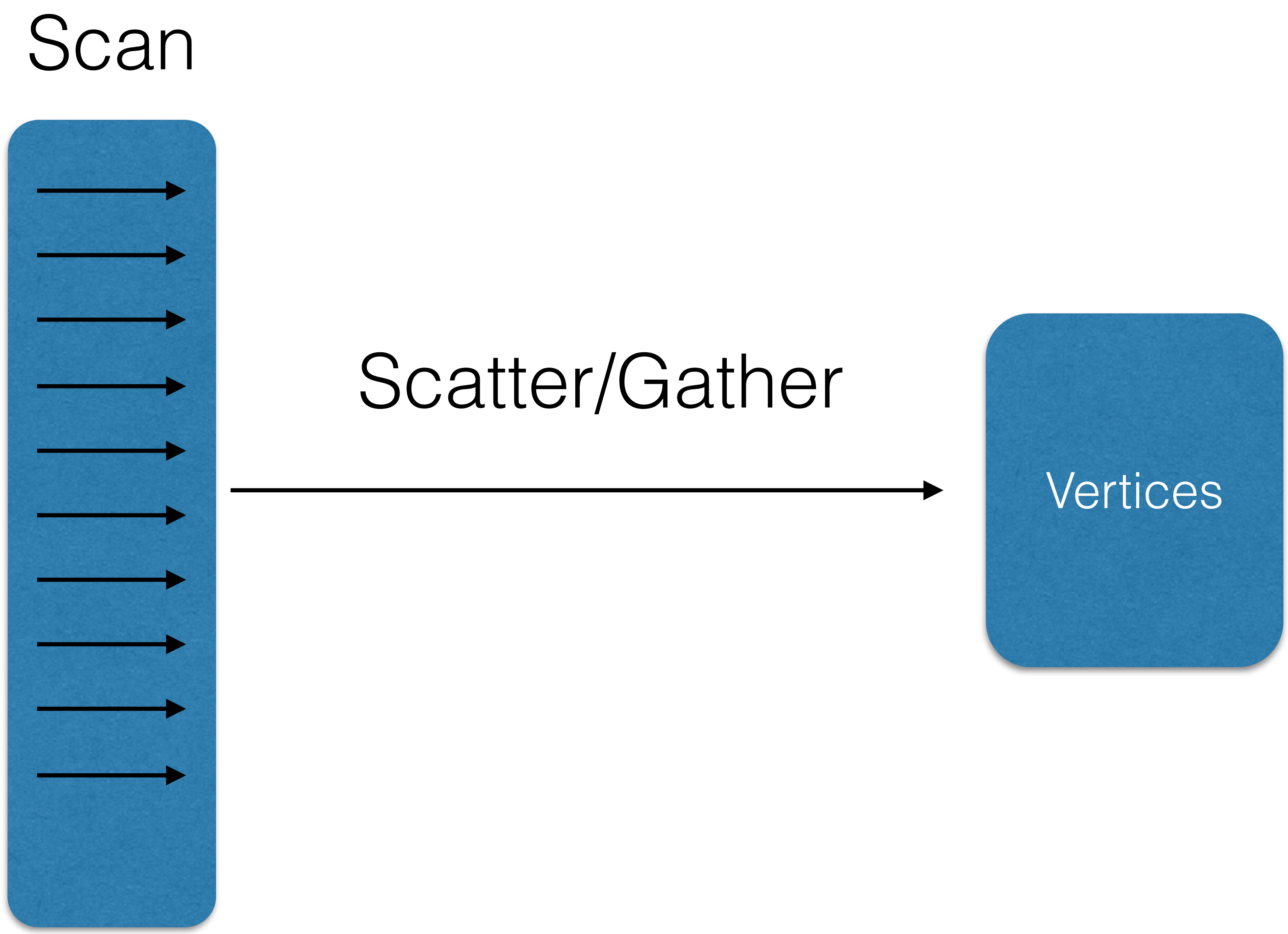
Using only half the available bandwidth



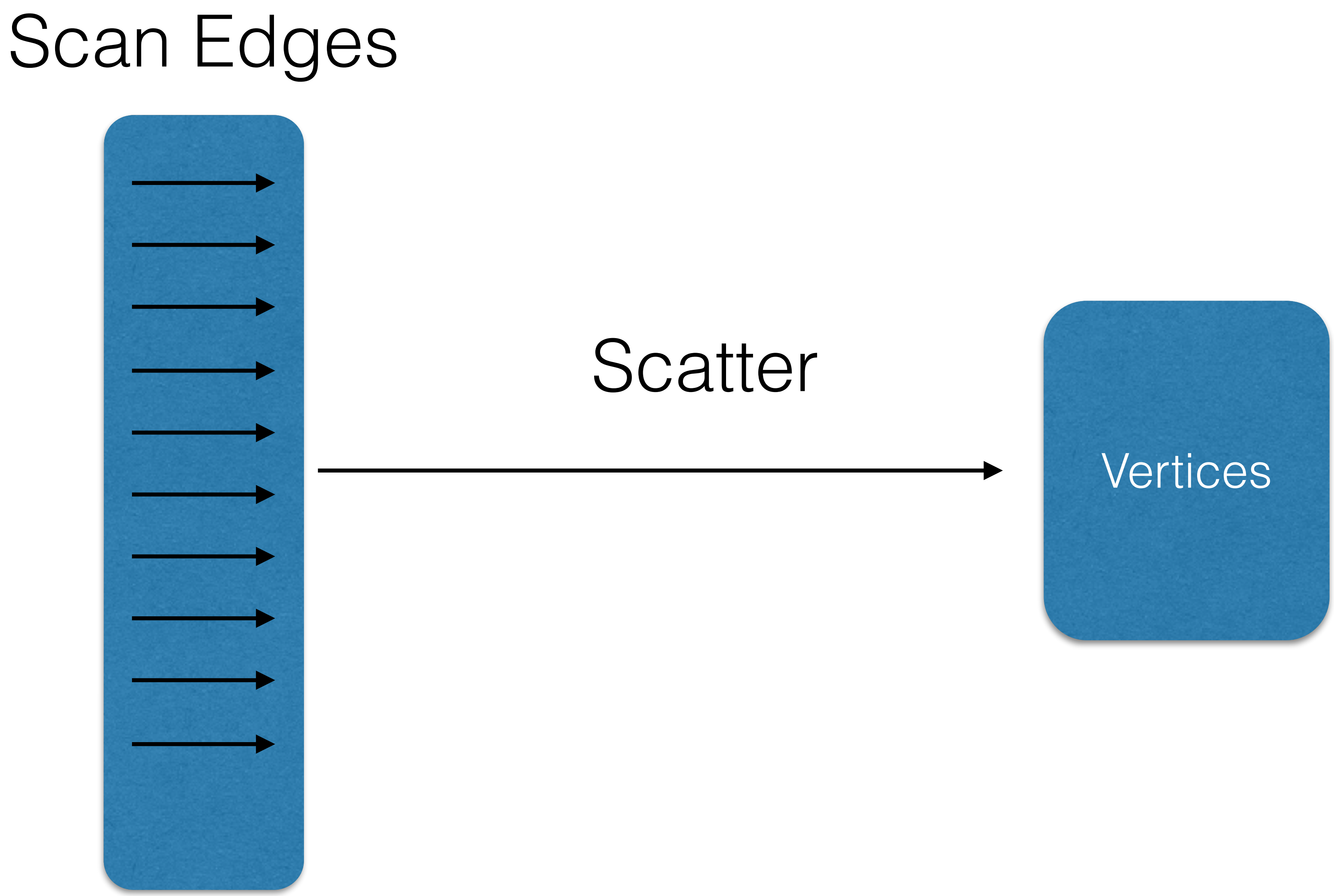
Extracting Parallelism

- Edge-centric loop
  - Stream in edges/updates
  - Access vertices
- What if…
- Independent copies of vertices on machines

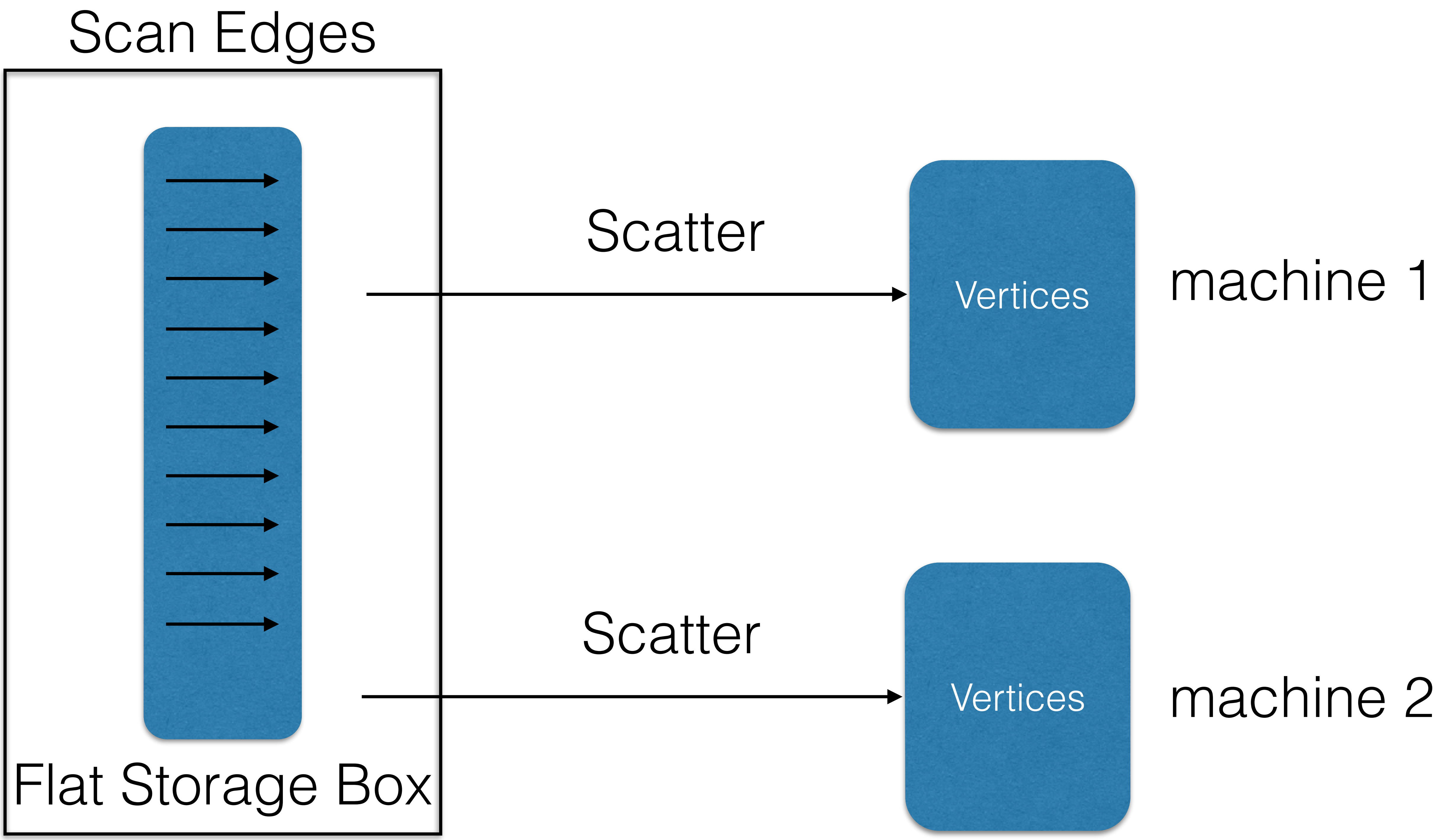
Extracting Parallelism



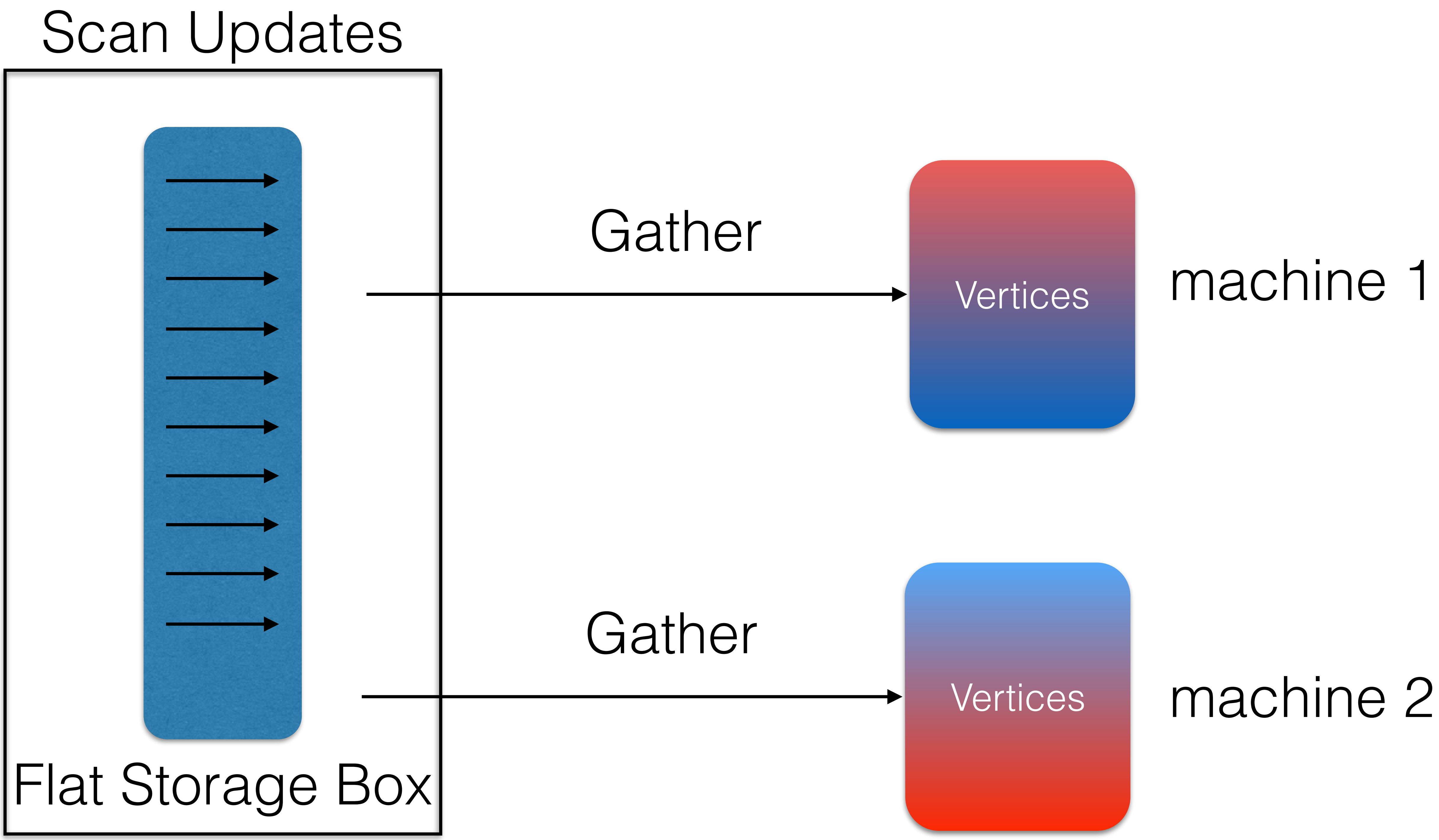
Scatter Step



Scatter Step



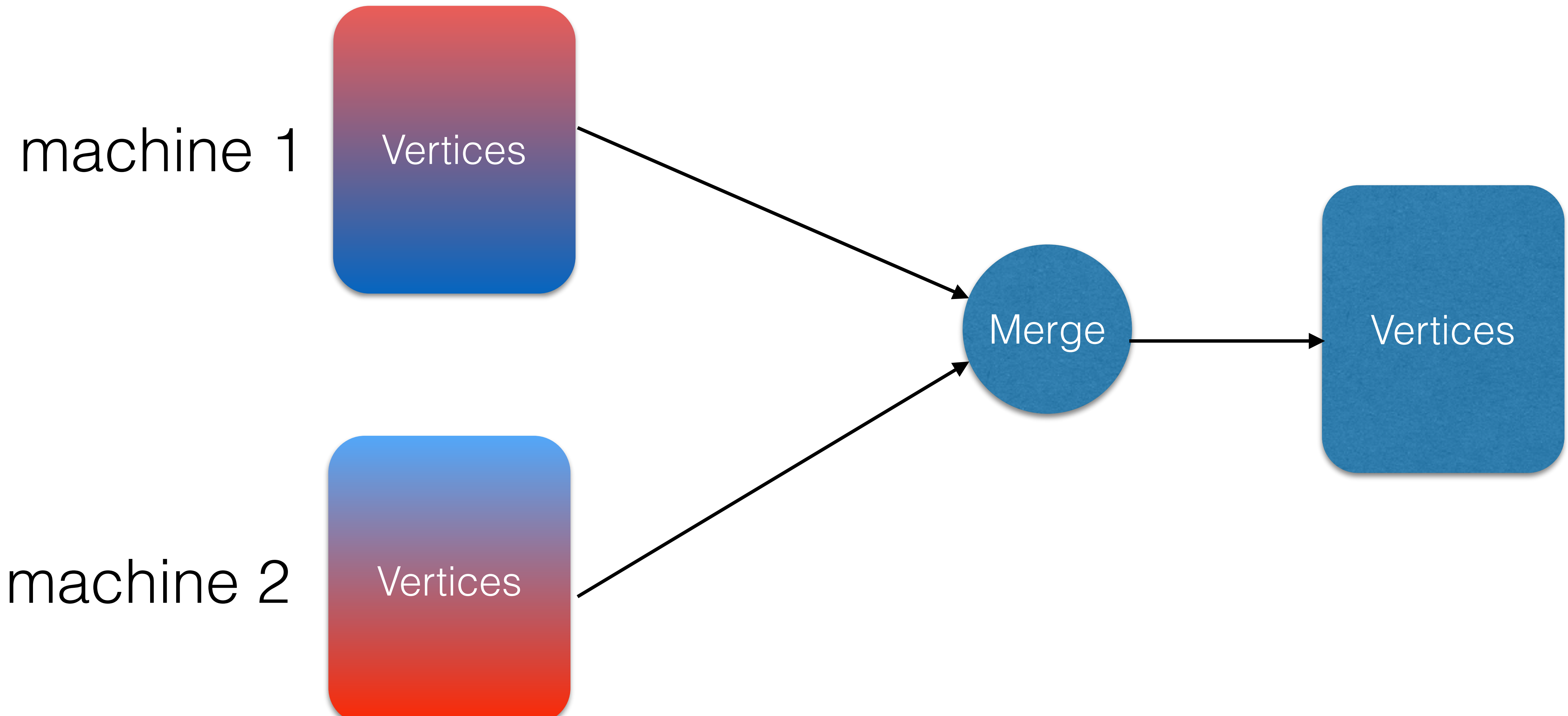
Gather Step



Scan Updates
Flat Storage Box
Gather
Vertices
machine 1
Gather
Vertices
machine 2

Merge Step

Application of updates is commutative



No need to go to disk

X-Stream to SlipStream

SlipStream graph algorithms

=

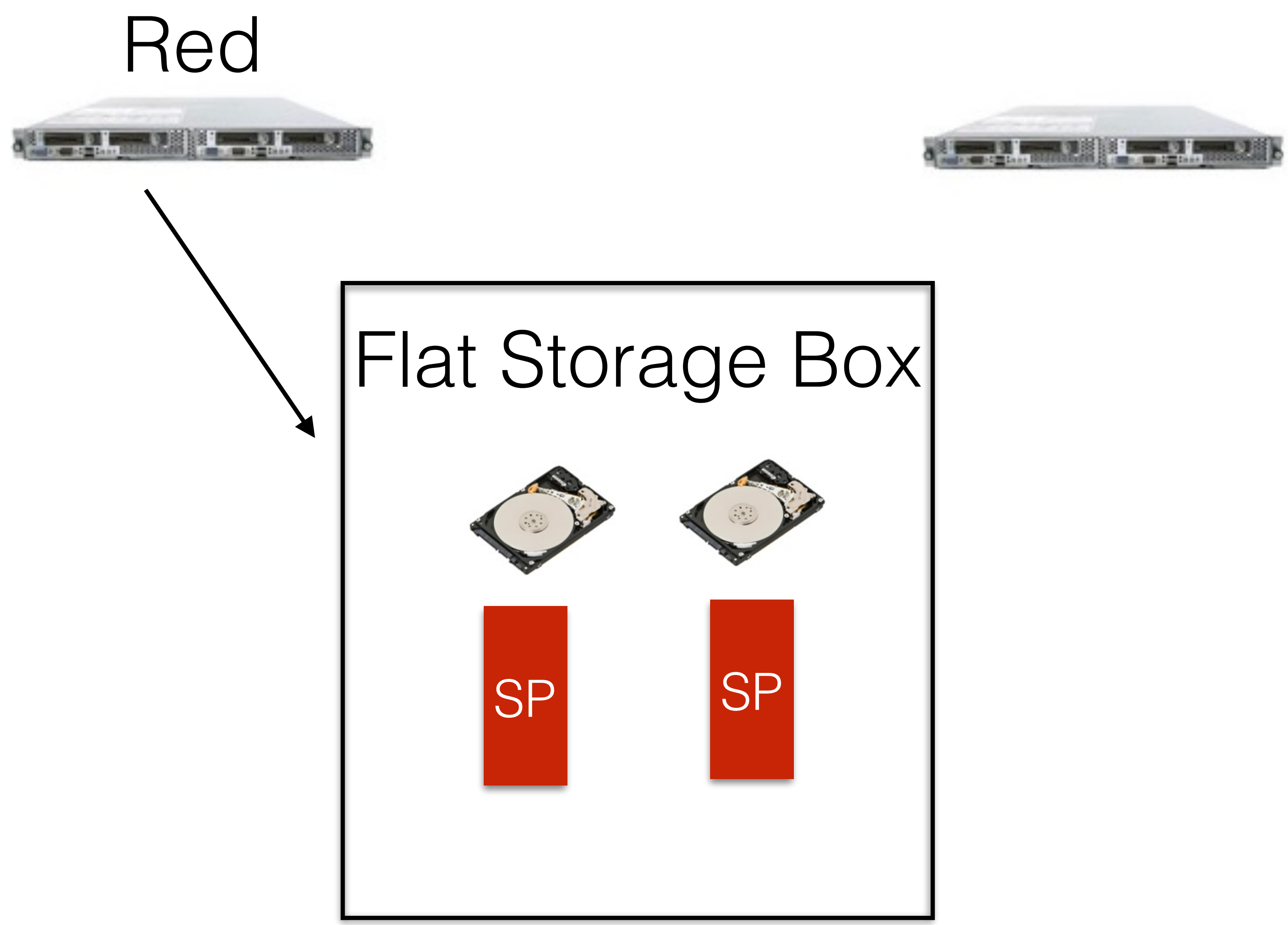
X-Stream graph algorithms

+

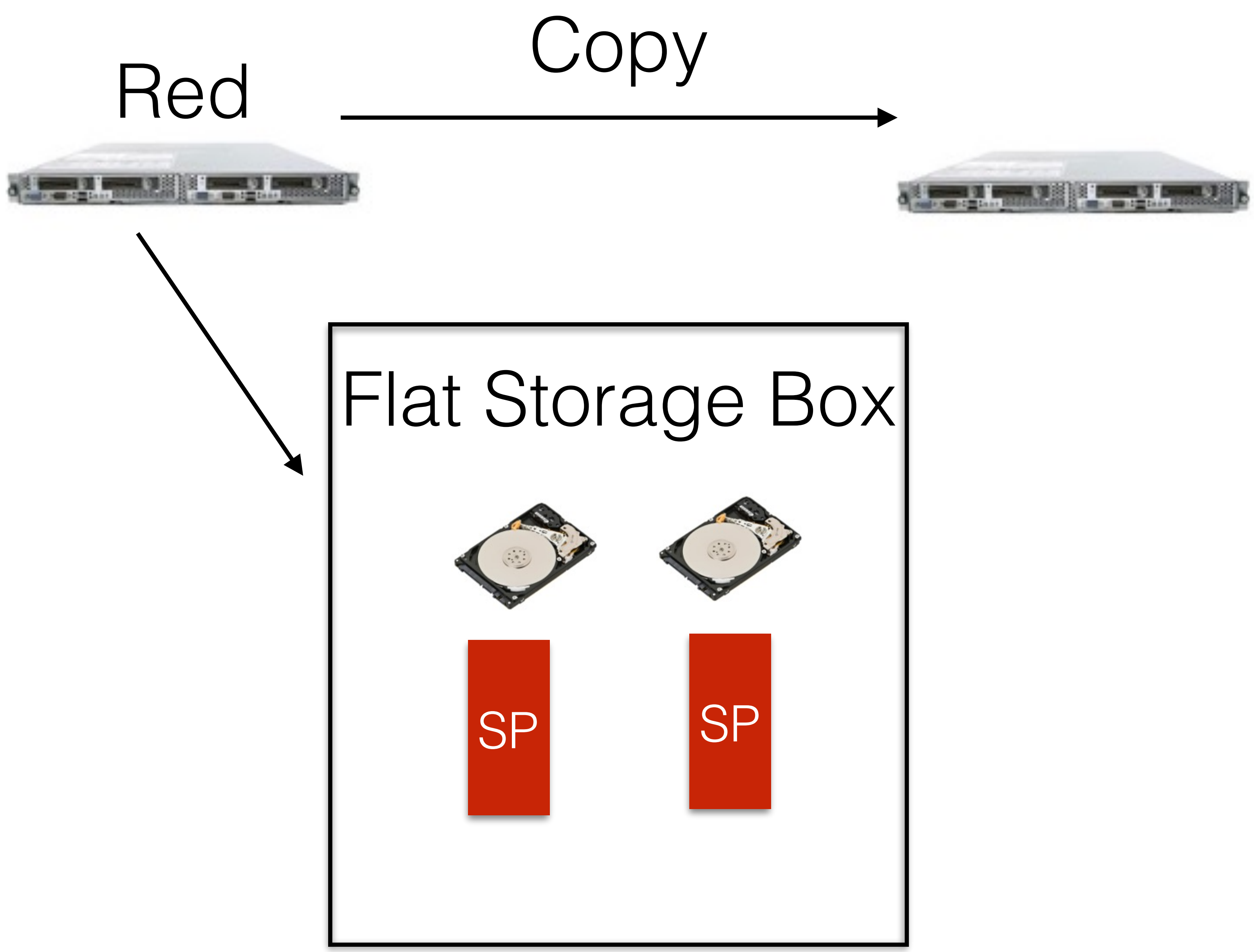
Merge function

- Easy to write merge function (looks like gather)

Putting it Together

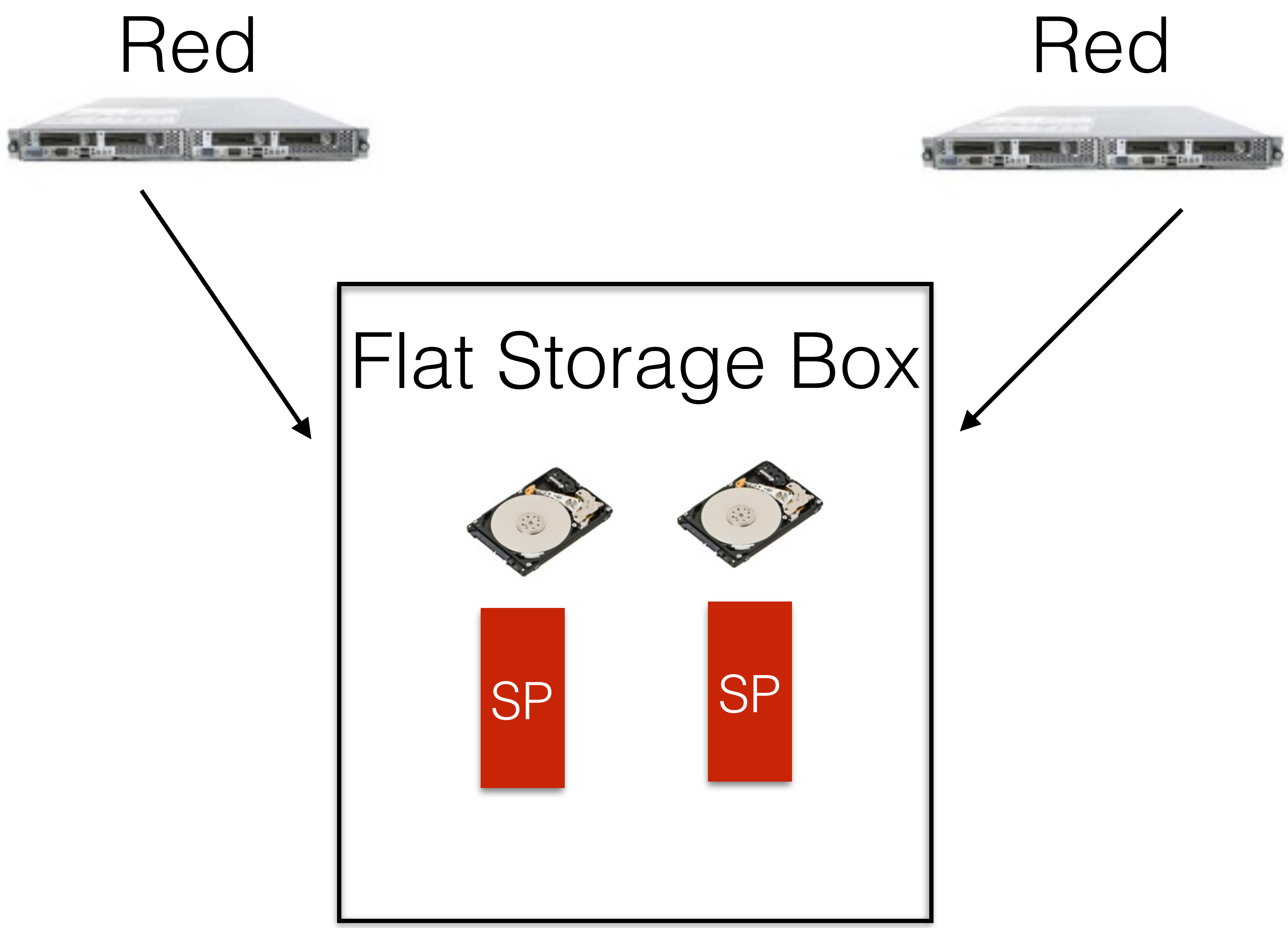


Putting it Together

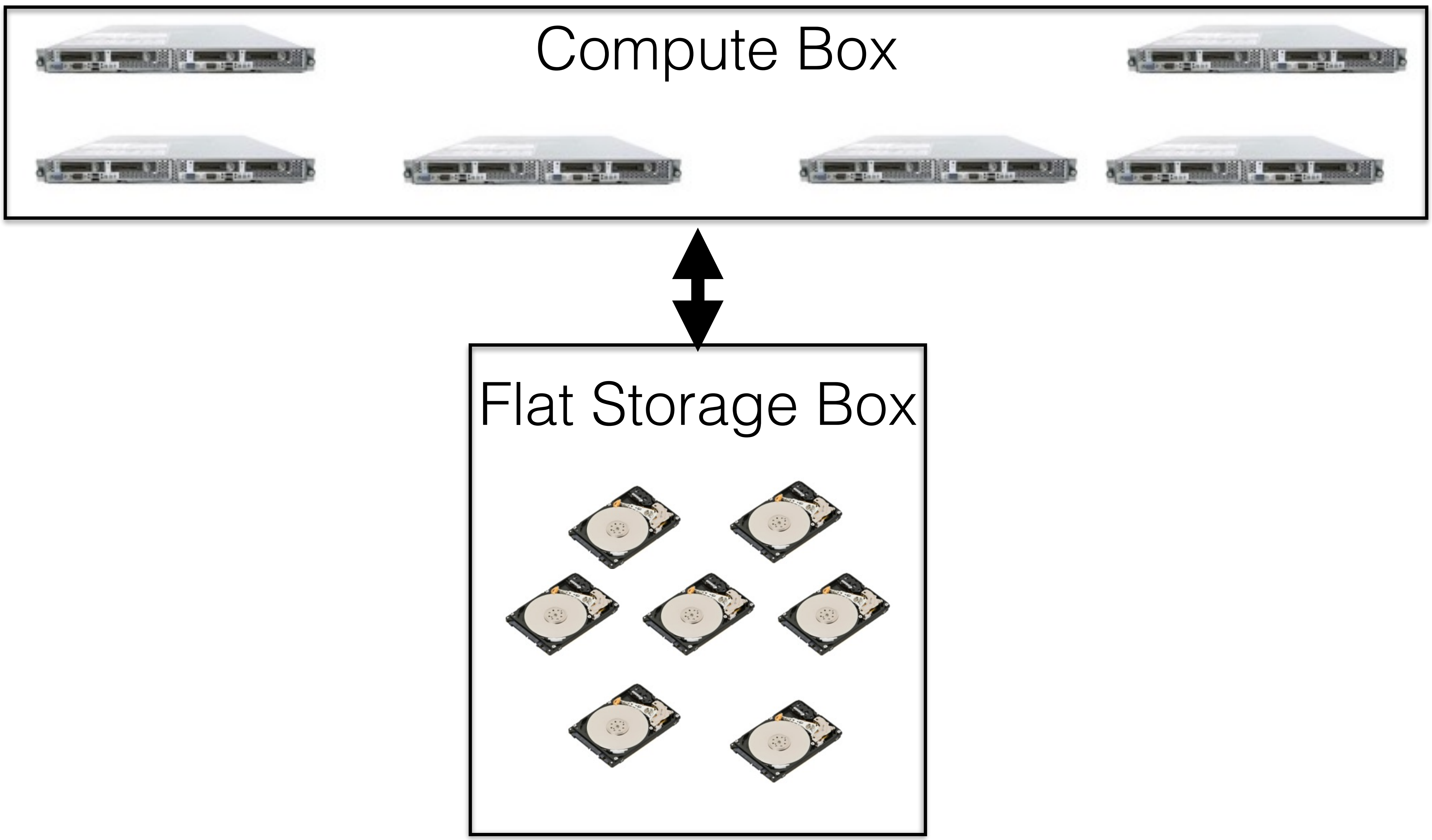


Putting it Together

✔ Back to Full Bandwidth



Automatic Load Balancing



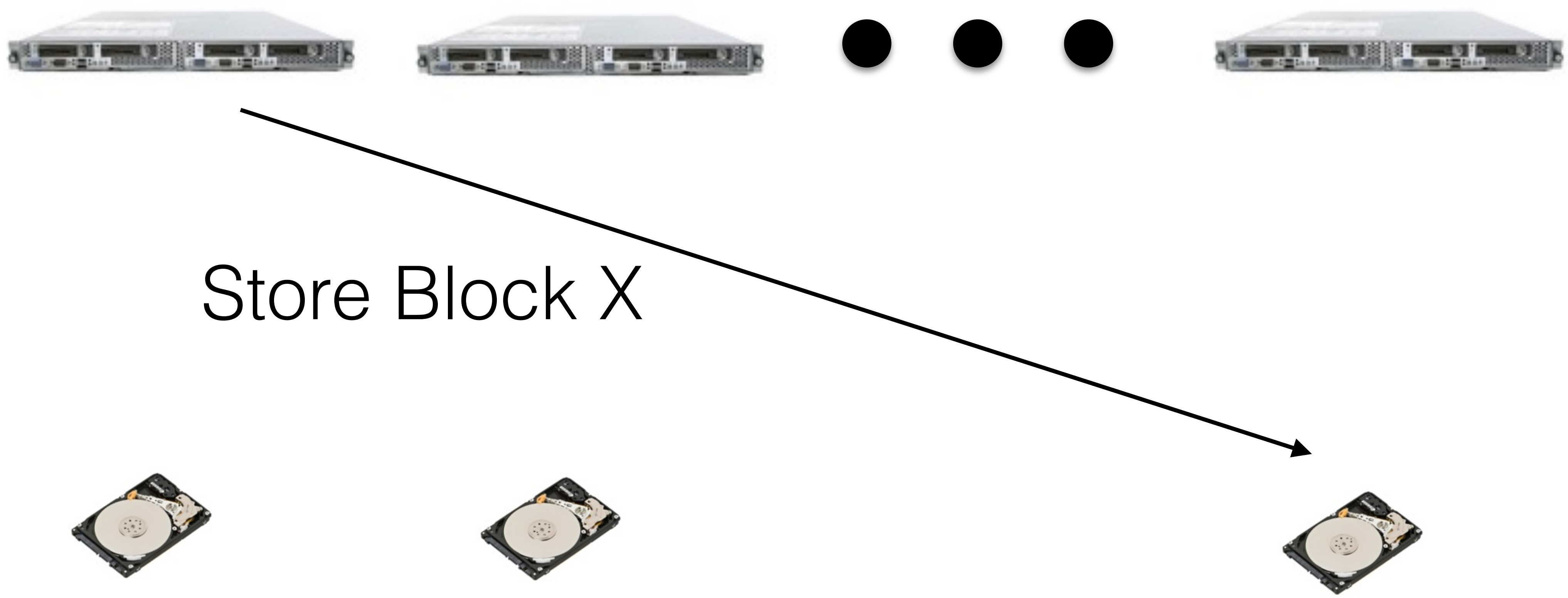
Recap

- Graph Partitioning across machines is hard
    - Drop locality using flat storage
        - Make it one disk
    - Same streaming partition on multiple nodes
        - Extract full bandwidth from the aggregated disk
- Systems approach to solving algorithms problem

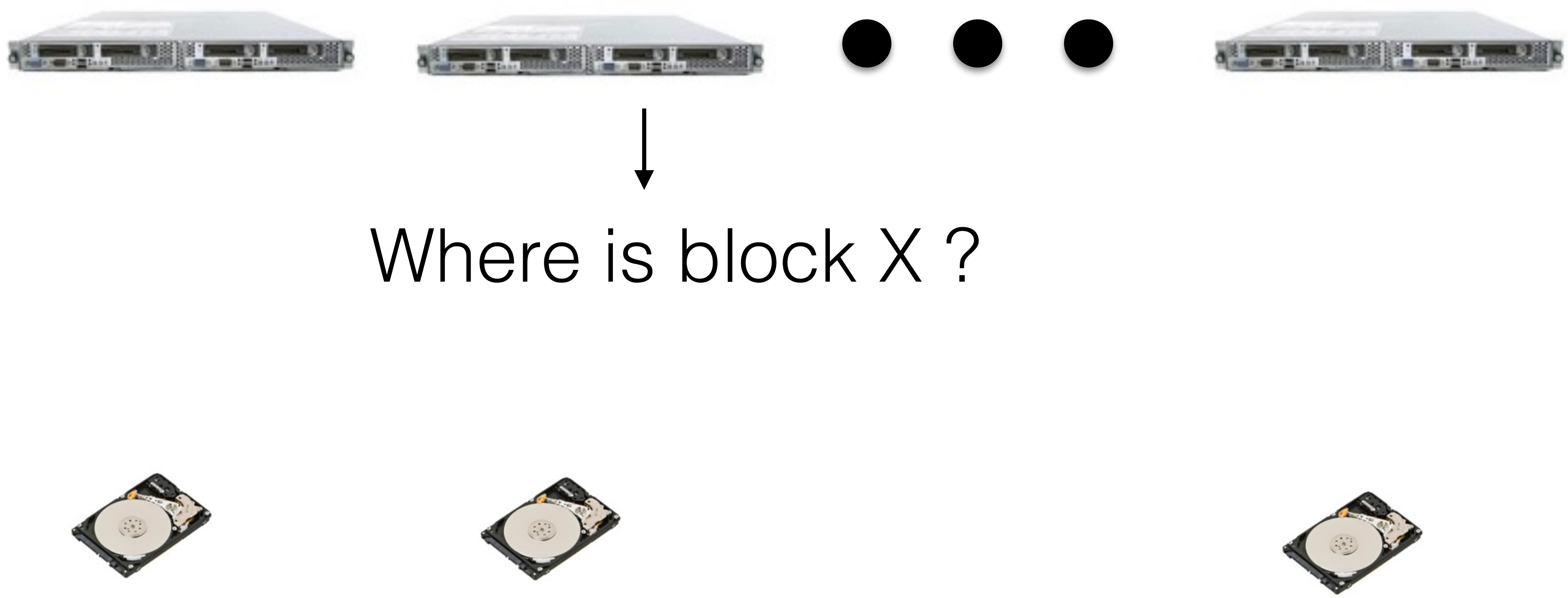
Flat Storage

- Distributed Storage layer for SlipStream
- Looked at other designs
  - FDS (OSDI 2012)
  - GFS (SOSP 2003)
  - …
- Implementing distributed storage is hard ☹

The Hard Bit

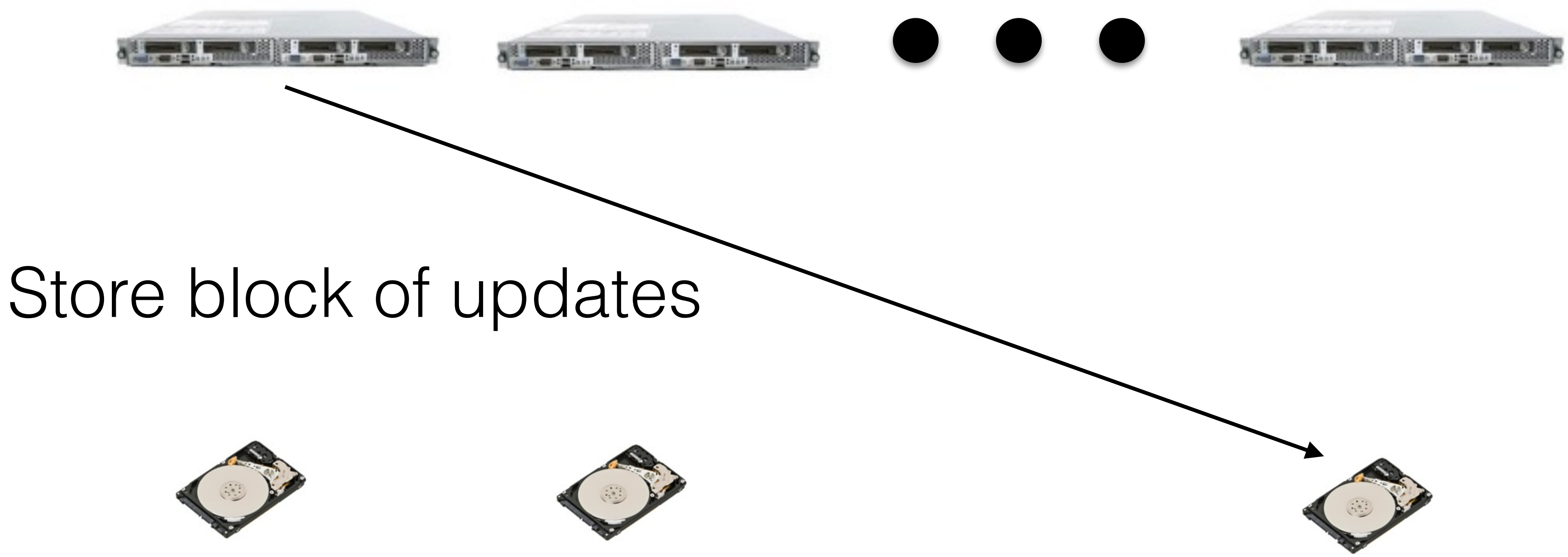


The Hard Bit

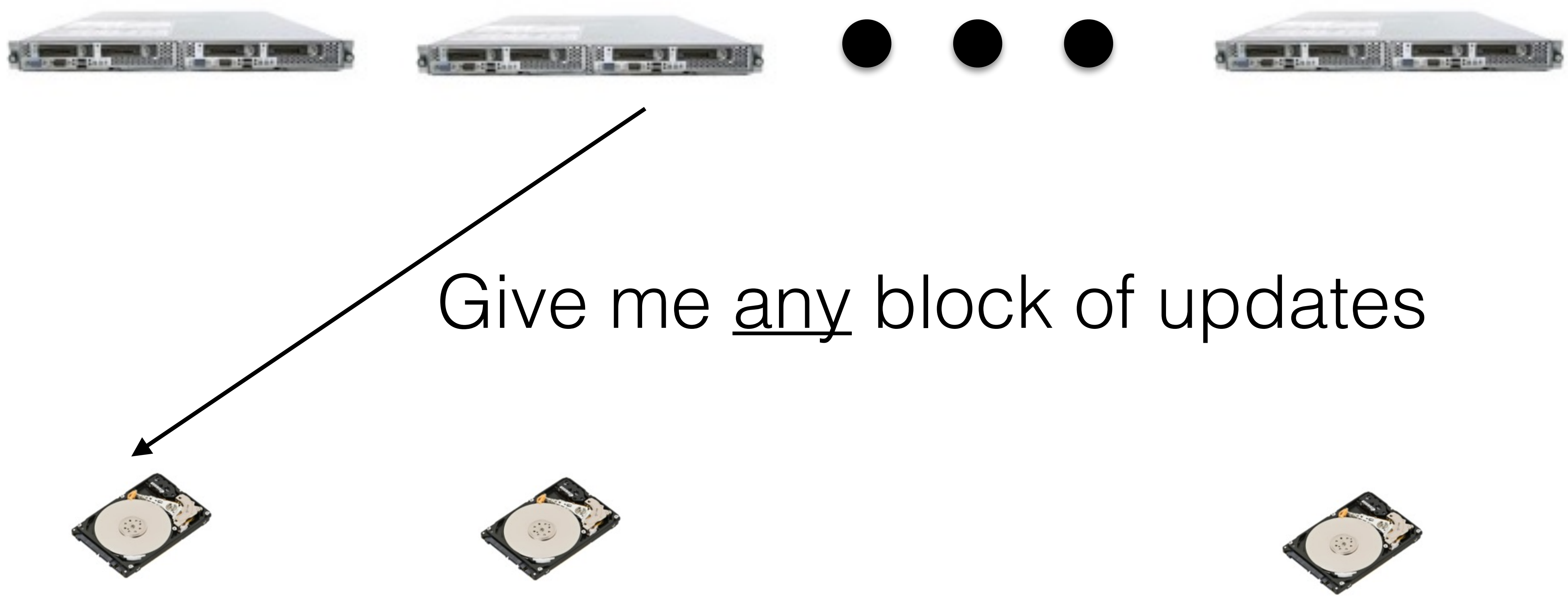


Need a location service
f: file, block → machine, offset

Block Location



Block Location is Irrelevant



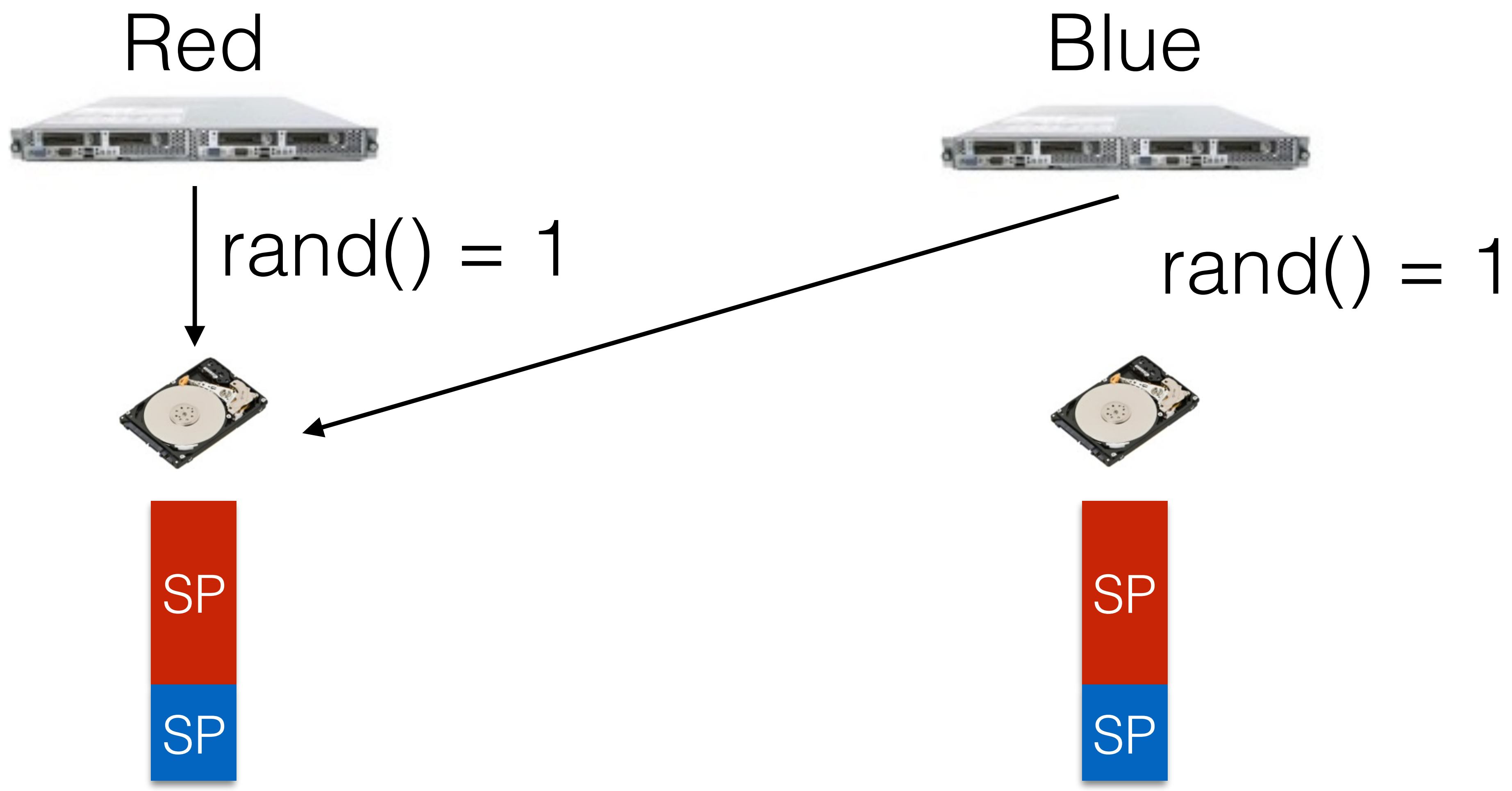
Streaming is order oblivious !

Random Schedule

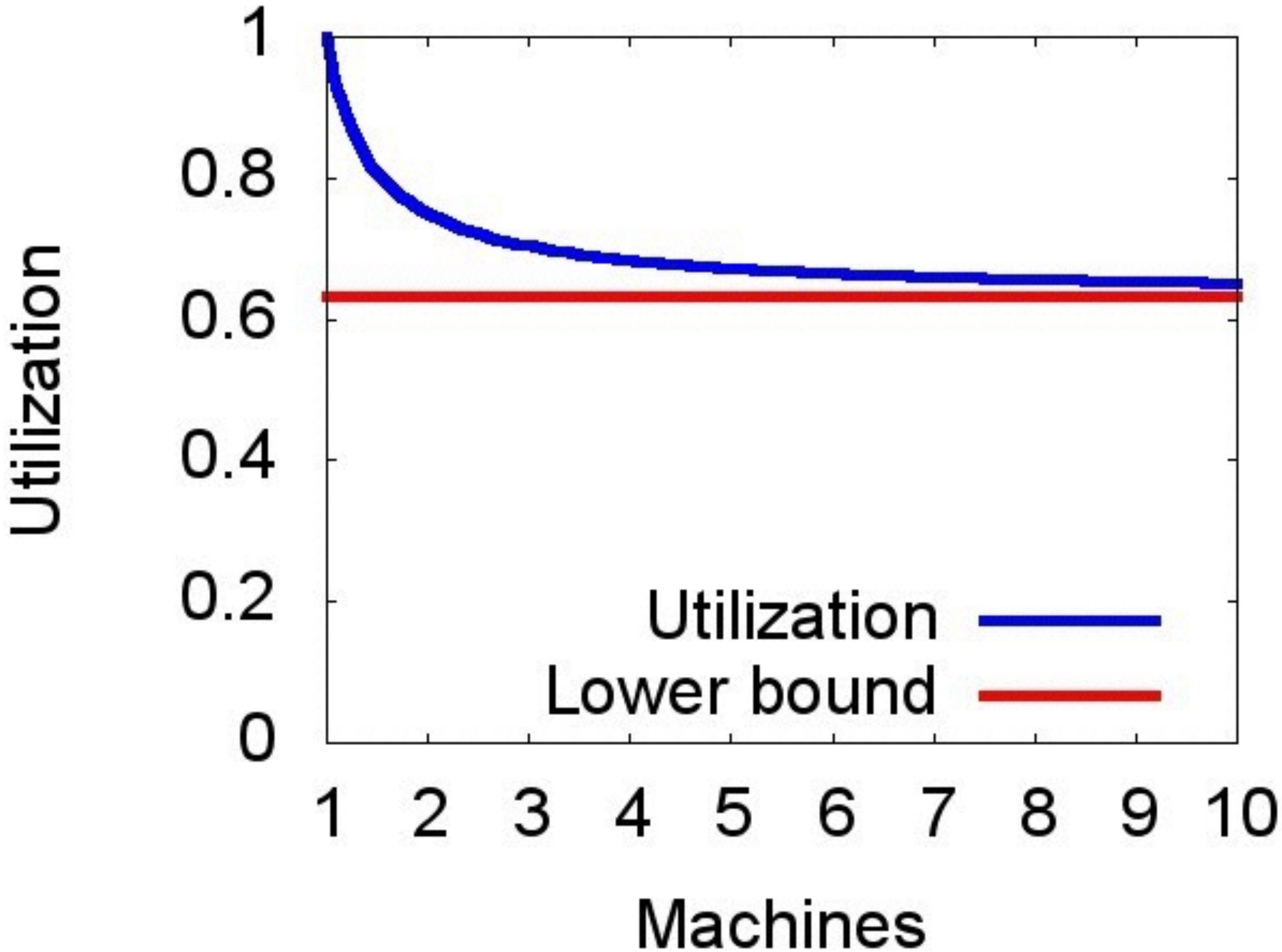
- ~~Centralized metadata service~~ ⇒ randomization
- Connect to a random machine for load/store
- Extremely simple implementation

Downside ?

- Can lead to collisions
- Collisions reduce utilization



No Downside

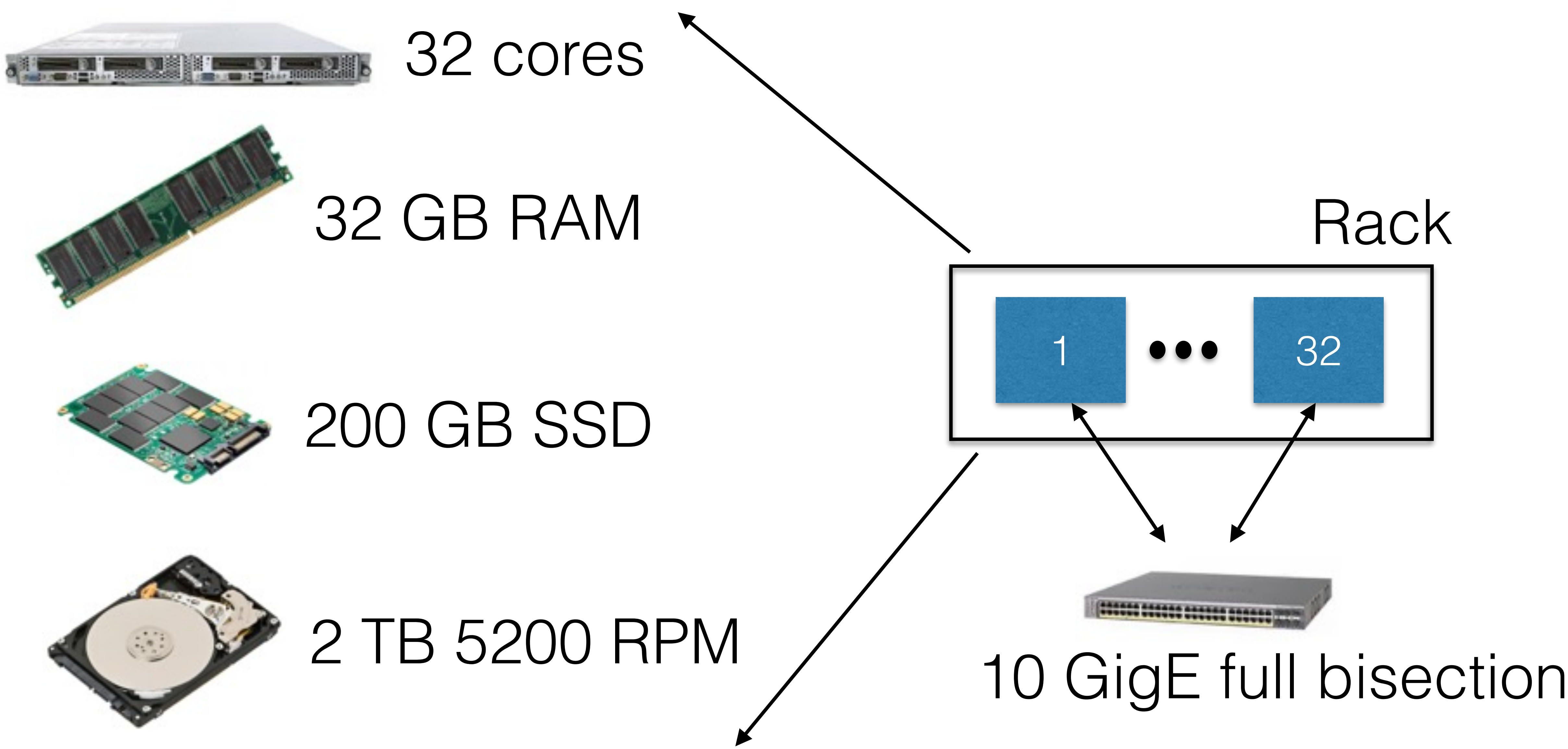


- Utilization lower bound at (1 - 1/e) ~ 62%

Recap

- Building distributed storage is hard
- Algorithms approach to solving systems problem
  - Streaming algorithms are order oblivious
  - Randomized schedule

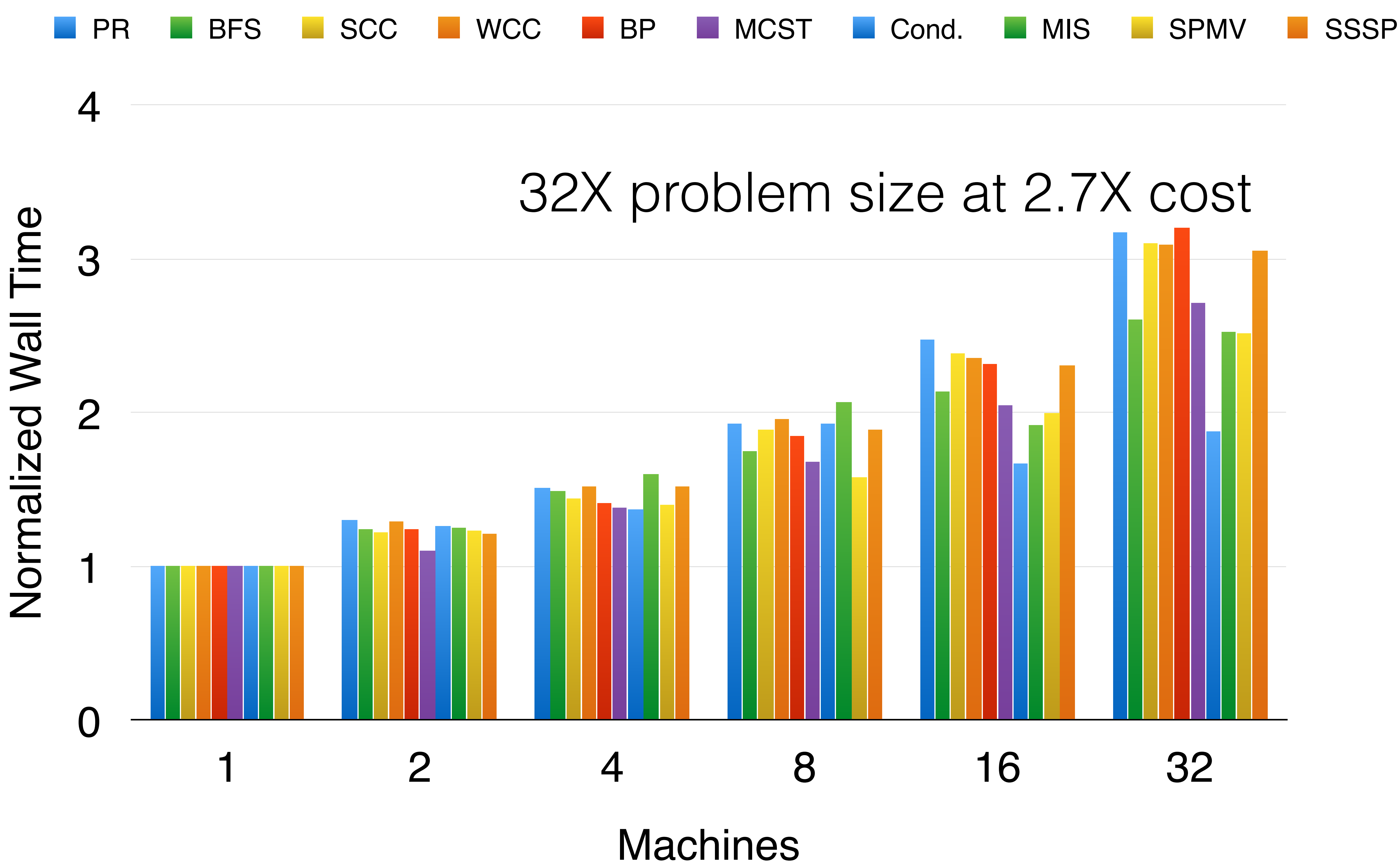
Evaluation Results



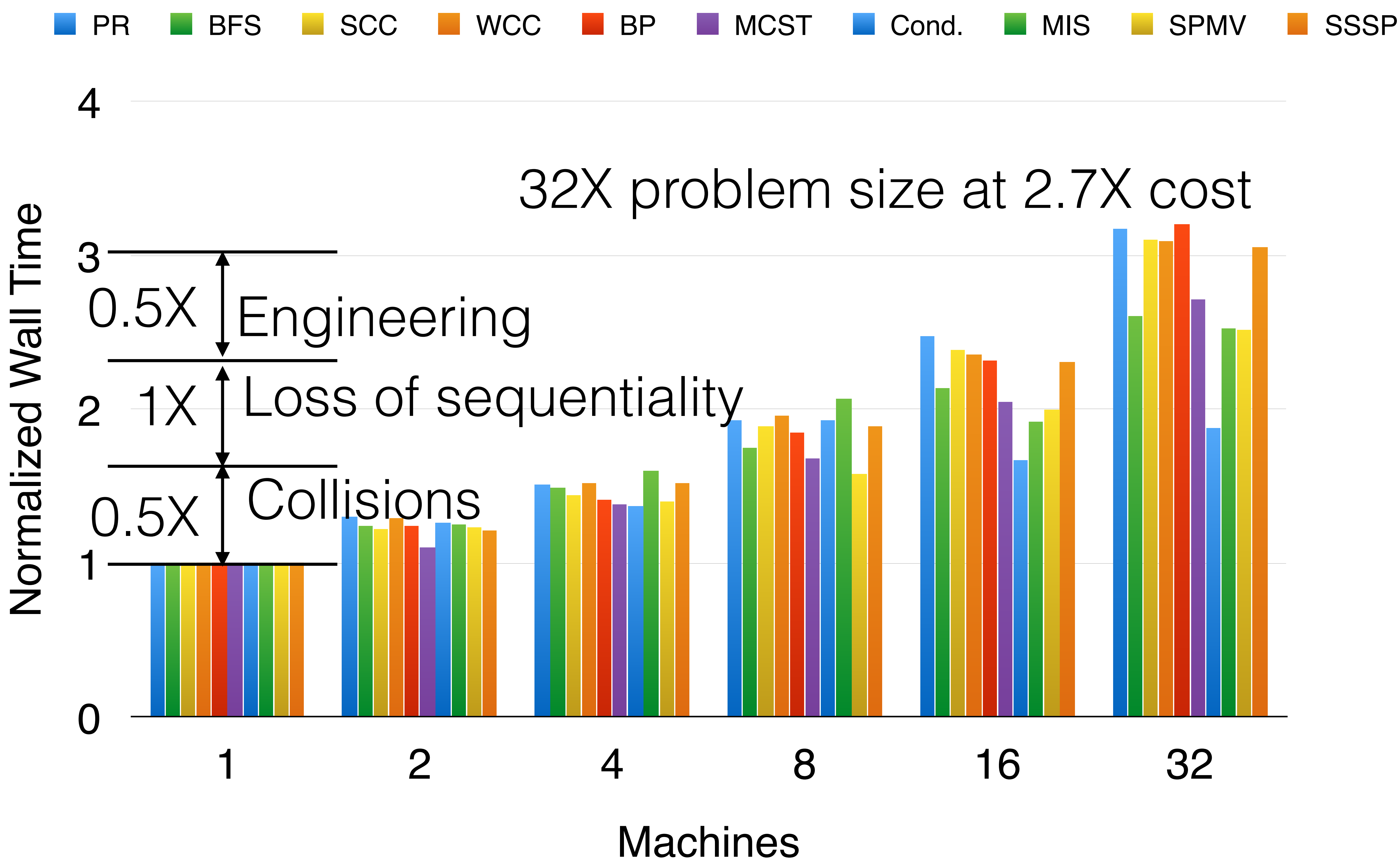
Scalability

- Solve larger problems using more machines
- Used synthetic scale-free graphs
  - Double problem size (vertices and edges)
  - Double machine count
- Till 32 machines, 4 billion vertices, 64 billion edges

Scaling RMAT (SSD)



Scaling RMAT (SSD)



Capacity

- Largest graph we can fit in our cluster
  - 32 billion vertices, 1 trillion edges
  - Magnetic disks
  - BFS
- Projected seeks were 1 year

Terascale

Metric	Value
<u>Wall Time</u>	<u>2d 9h</u>
MTEPS	<u>5</u>
I/O	282 TB
BW	1.53 GB/s

Don’t need supercomputers or very large clusters

Terascale

Metric	Value
<u>Wall Time</u>	<u>2d 9h</u>
MTEPS	<u>5</u>
I/O	282 TB
BW	1.53 GB/s

Direct results from unordered edge list

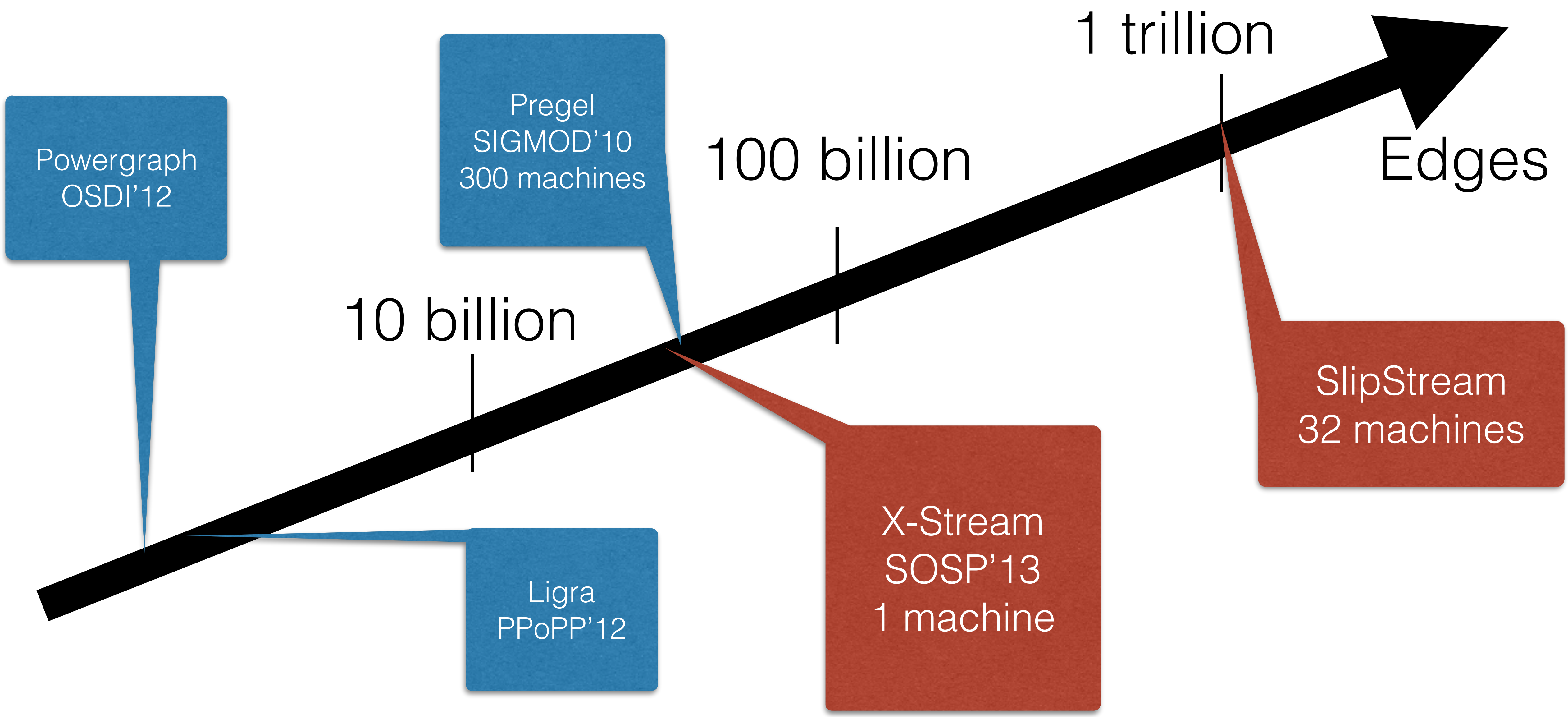
SlipStream vs. Competition

System	RAM	Pre-process	Run
Powergraph	128 GB	1271s	103s
SlipStream	32 GB	X	1854s

WCC/RMAT/128M vertices 2B edges/2 machines

Preprocessing your data for locality can take a lot of time !

Where we stand



How do we get further ? Buy more disks :)

Conclusion

- Process large graphs using secondary storage
  - Match algorithm to systems: streaming
  - Match system to algorithms: order obliviousness
- If you can store it, you can process it