CHAOS
Scale-out Graph Processing from Secondary Storage

Amitabha Roy — Intel, Santa Clara

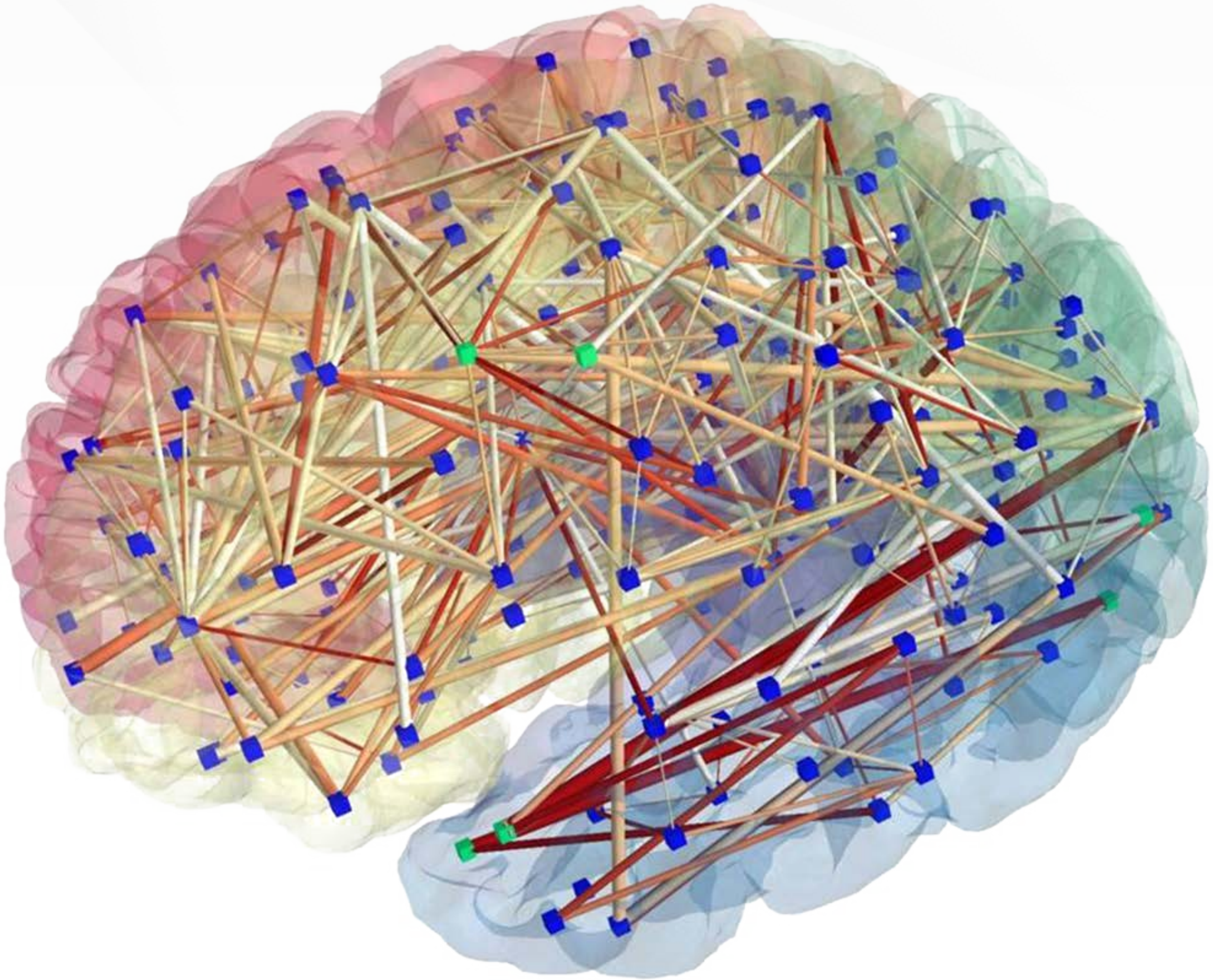
Laurent Bindschaedler

Jasmina Malicevic — EPFL, Switzerland

Willy Zwaenepoel

Large Graphs – Social Networks

Large Graphs – Brain



Large Graphs – Road Networks

How Large is a Large Graph?

“*A billion edges isn’t cool. You know what’s cool? A **TRILLION** edges!*”

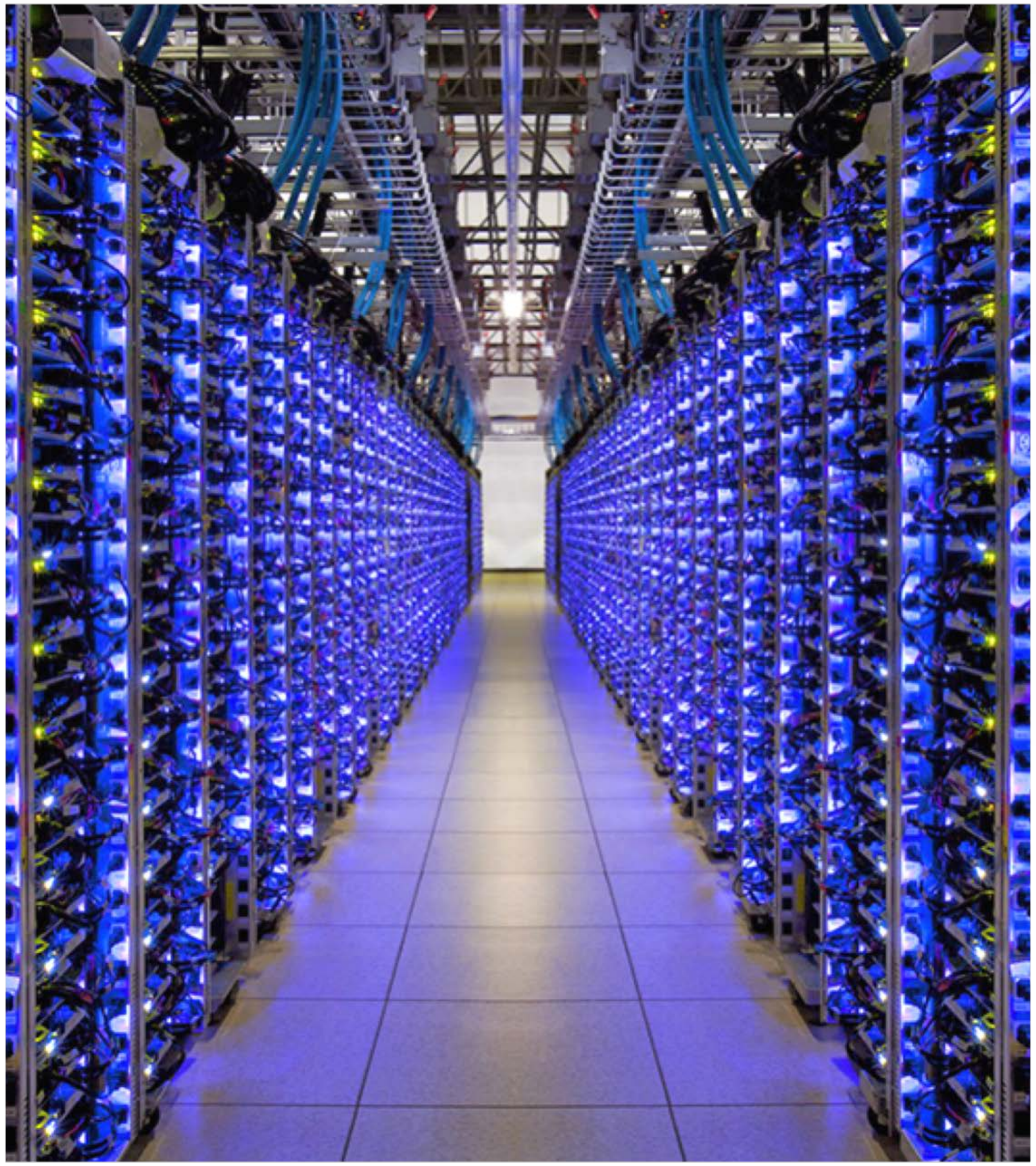
– Avery Ching, Facebook

Graph Processing – HPC Approach

Single machine

In memory

Graph Processing – Facebook Approach



Many machines

In memory

Graph Processing – Chaos Approach

+

A few machines

Out-of-core

Challenge for Out-of-core

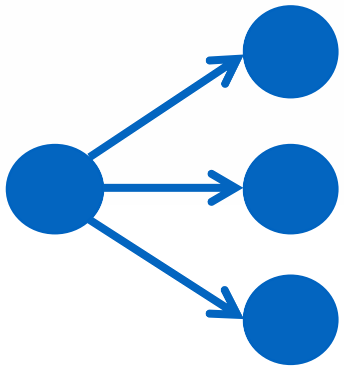
- Graph algorithms produce random accesses
- Performance requires sequential access
- A fortiori for secondary storage

X-Stream [SOSP'13]

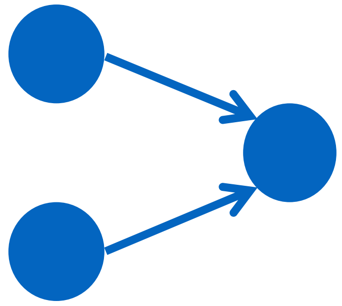
- Single-node (multi-core) graph processing
- Goal: make all access sequential!
- Two techniques:
  - Edge-centric graph processing
  - Streaming partitions

X-Stream – Programming Model

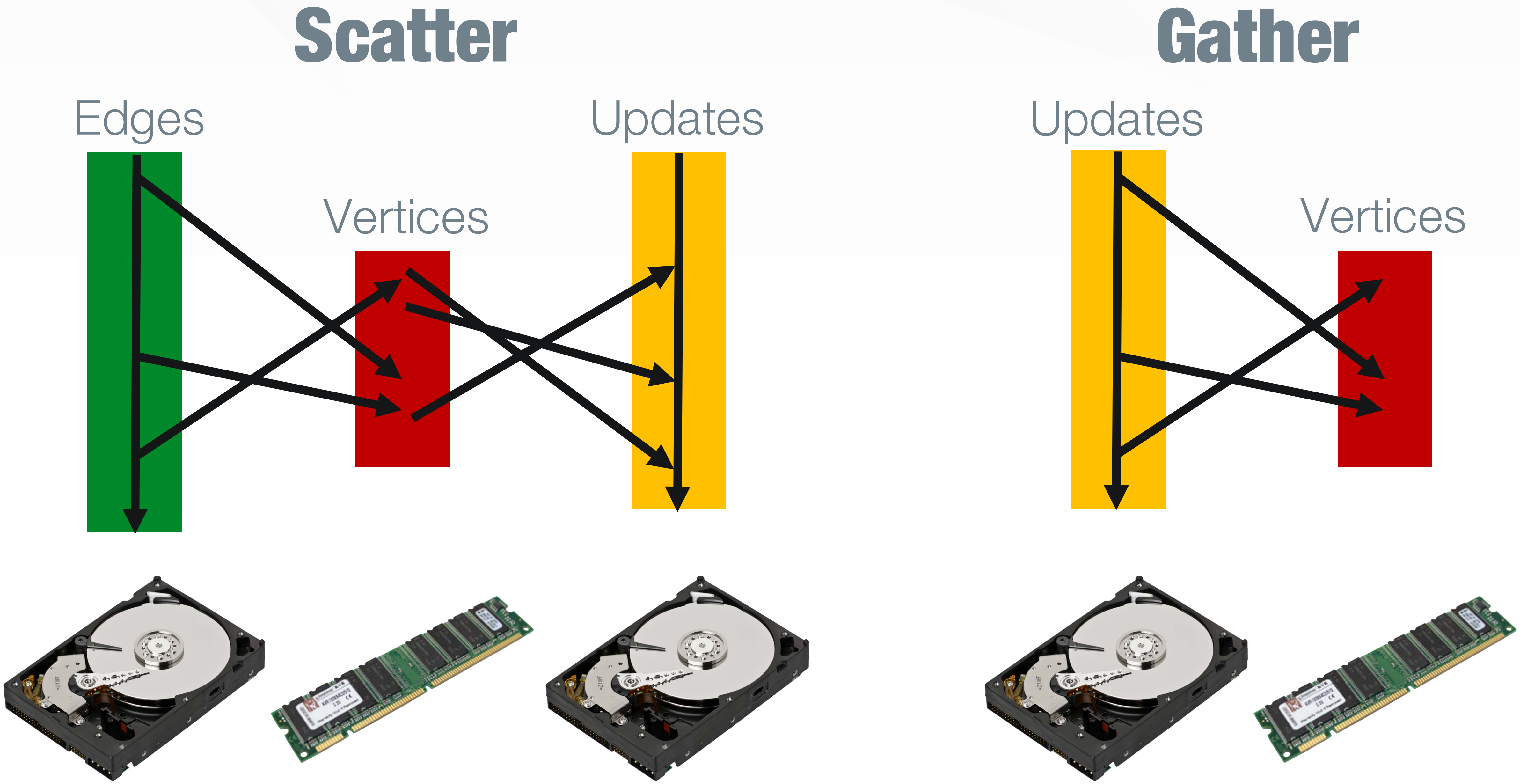
- Vertex-centric
  - Maintain state in vertex
  - Write a vertex program
- Vertex program has two methods
  - Scatter – For all outgoing edges:
    `new update = f(vertex value)`



  - Gather – For all incoming edges:
    `vertex value = g(vertex value, update)`



X-Stream – Overview



Expressive Power of Scatter Gather

Collaborative Filtering
ALS

Graph Statistics
HyperANF, Centralities

SpMV and variants

Scatter Gather

Traversal
BFS, SSSP

Structure Mining
(W)CC, MIS, MCST, Triangle Counting

Machine Learning
PageRank, BP, Cond

X-Stream Design – Summary

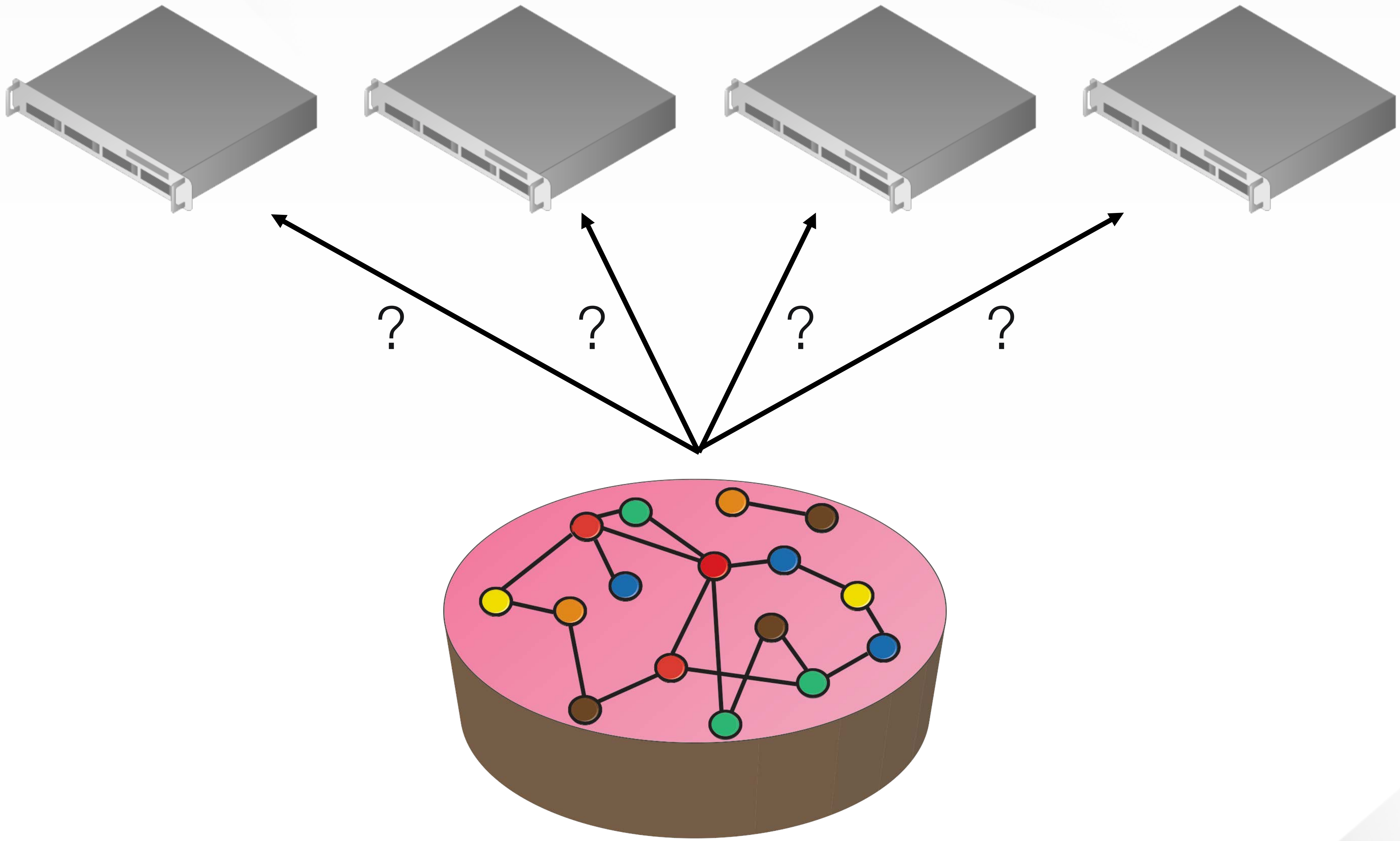
Edge Centric

Organize graph computation around edges to stream data from storage

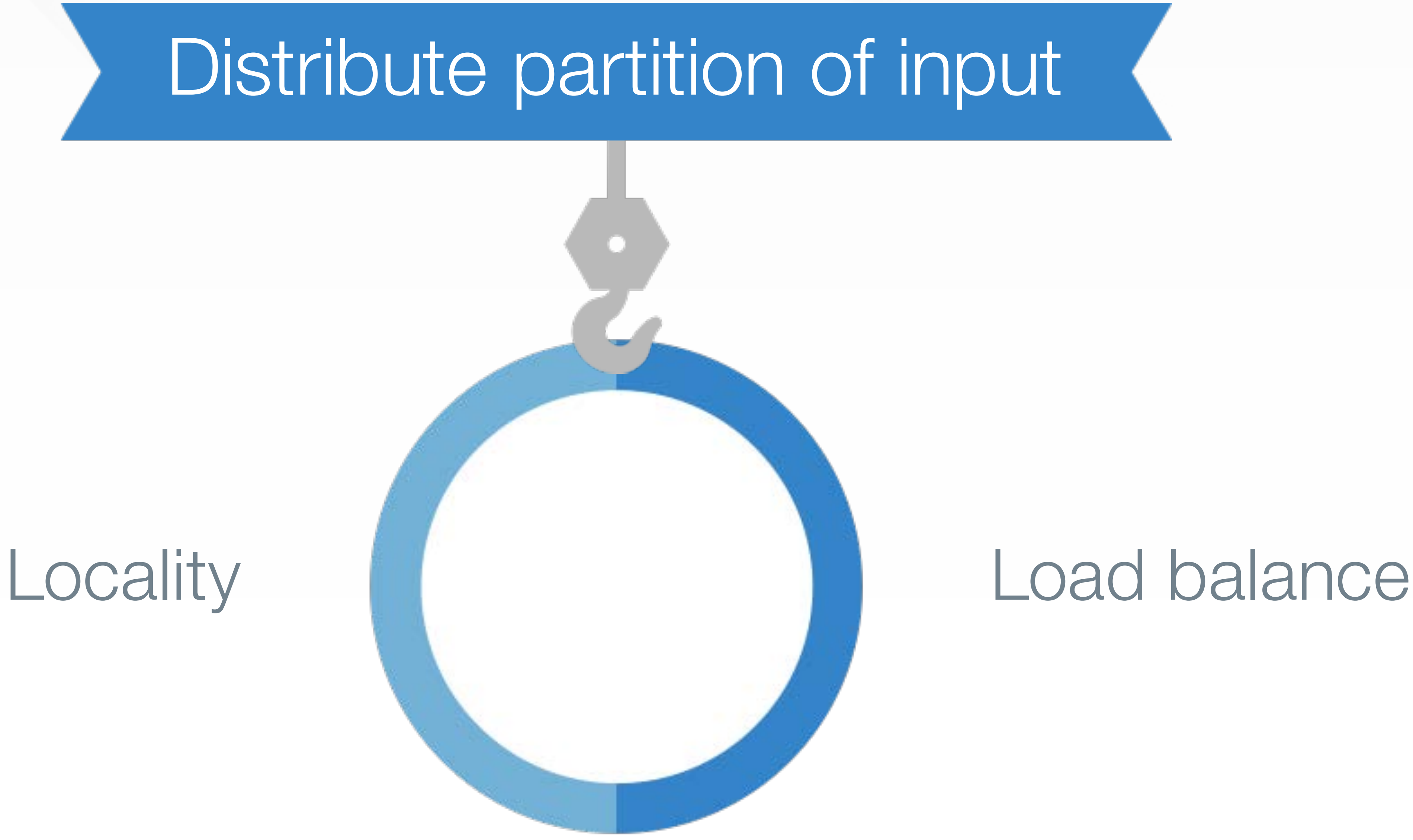
Streaming Partitions

Divide graph into partitions such that the vertex set for each partition fits in memory

Challenge for Distribution



Partitioning



Optimal partitioning is NP-hard

Partitioning – Existing Approaches

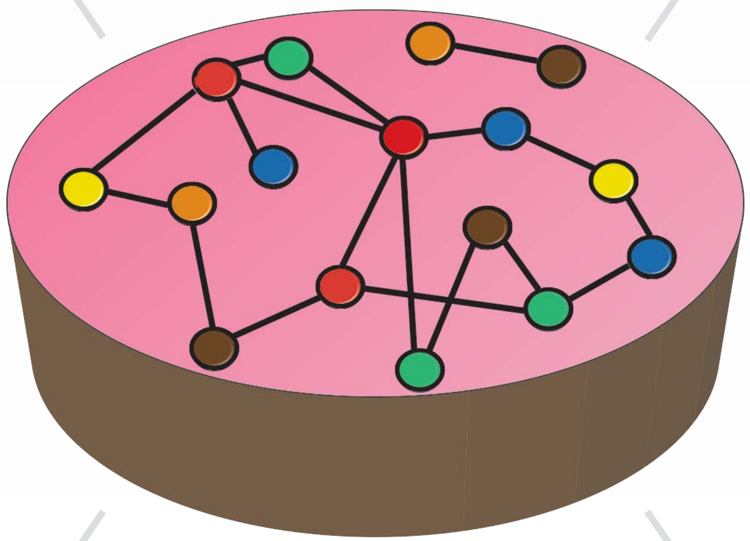
Edge cuts

Pregel [SIGMOD’10]

Giraph [HADOOP’11]

Vertex cuts

PowerGraph [OSDI’12]



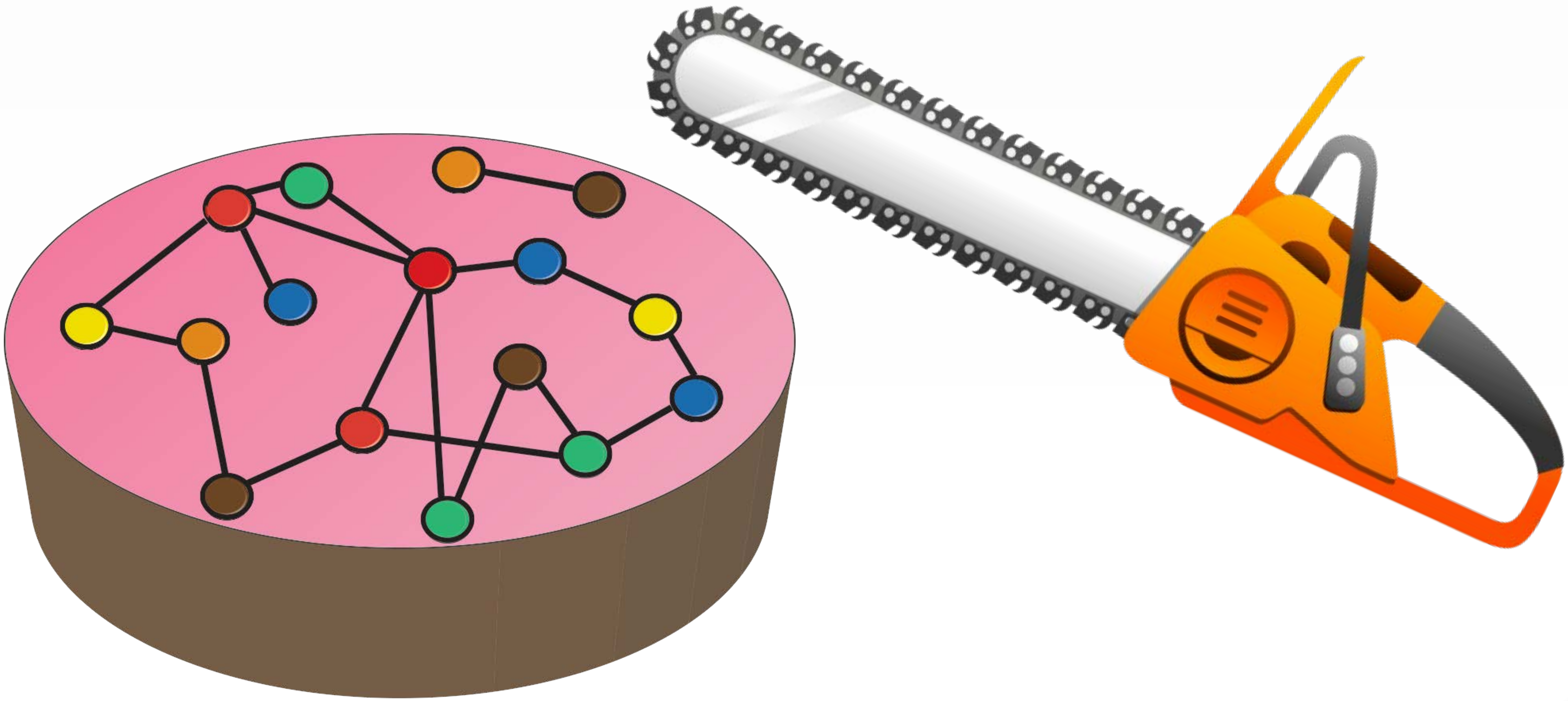
Sorting & sharding

GraphChi [OSDI’12]

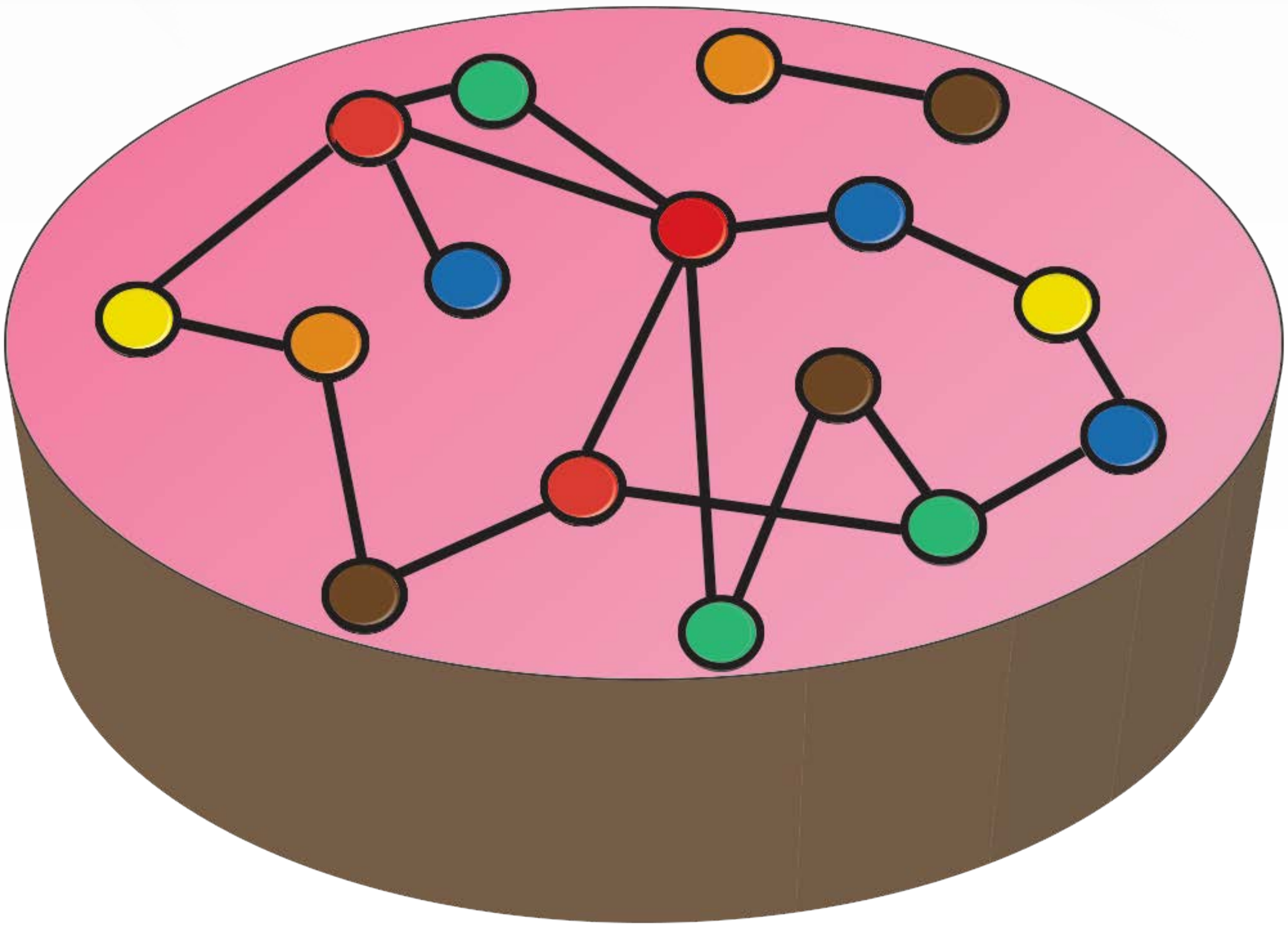
Hilbert curve

Naiad [SOSP’13]

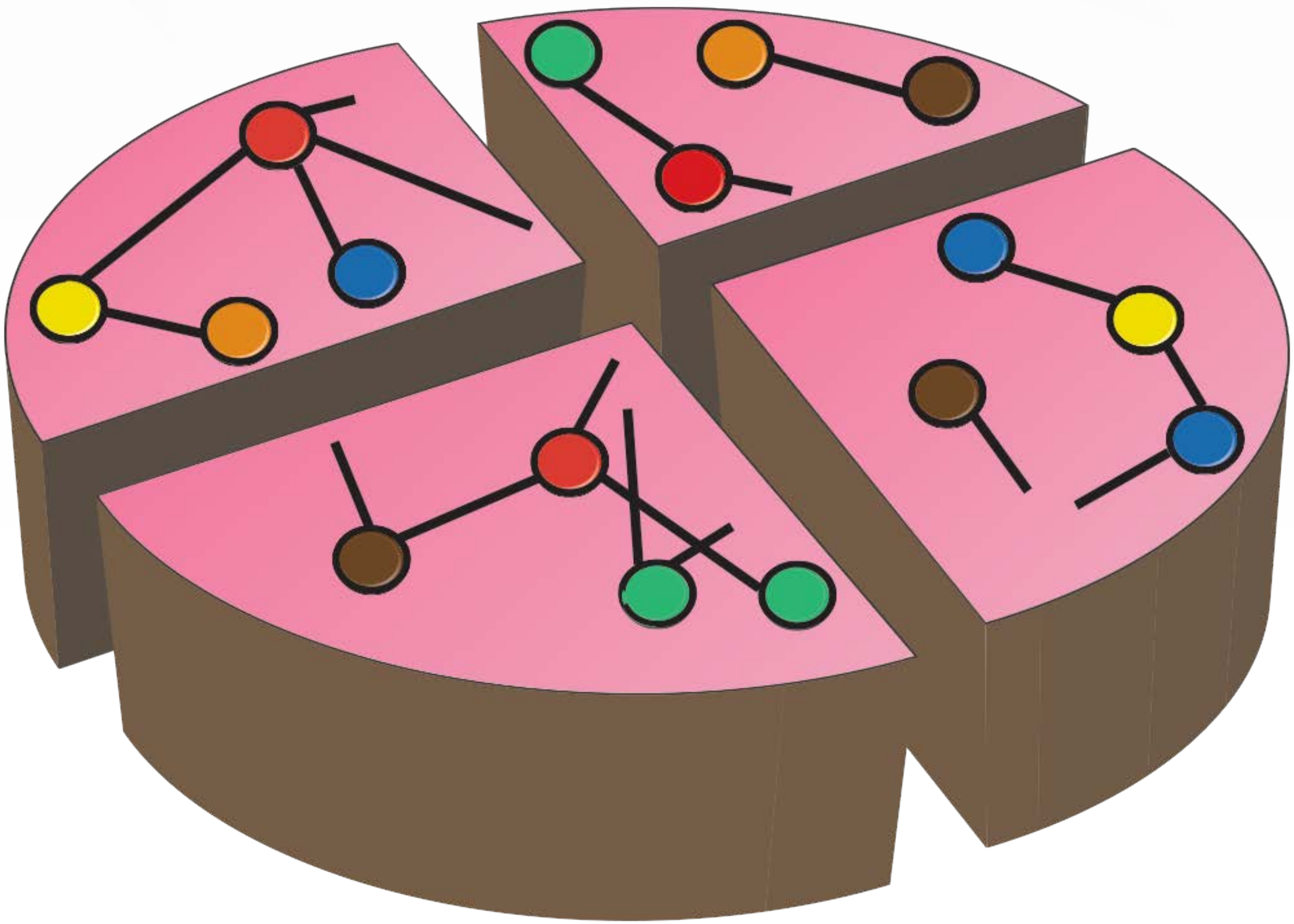
Partitioning – Chaos Approach



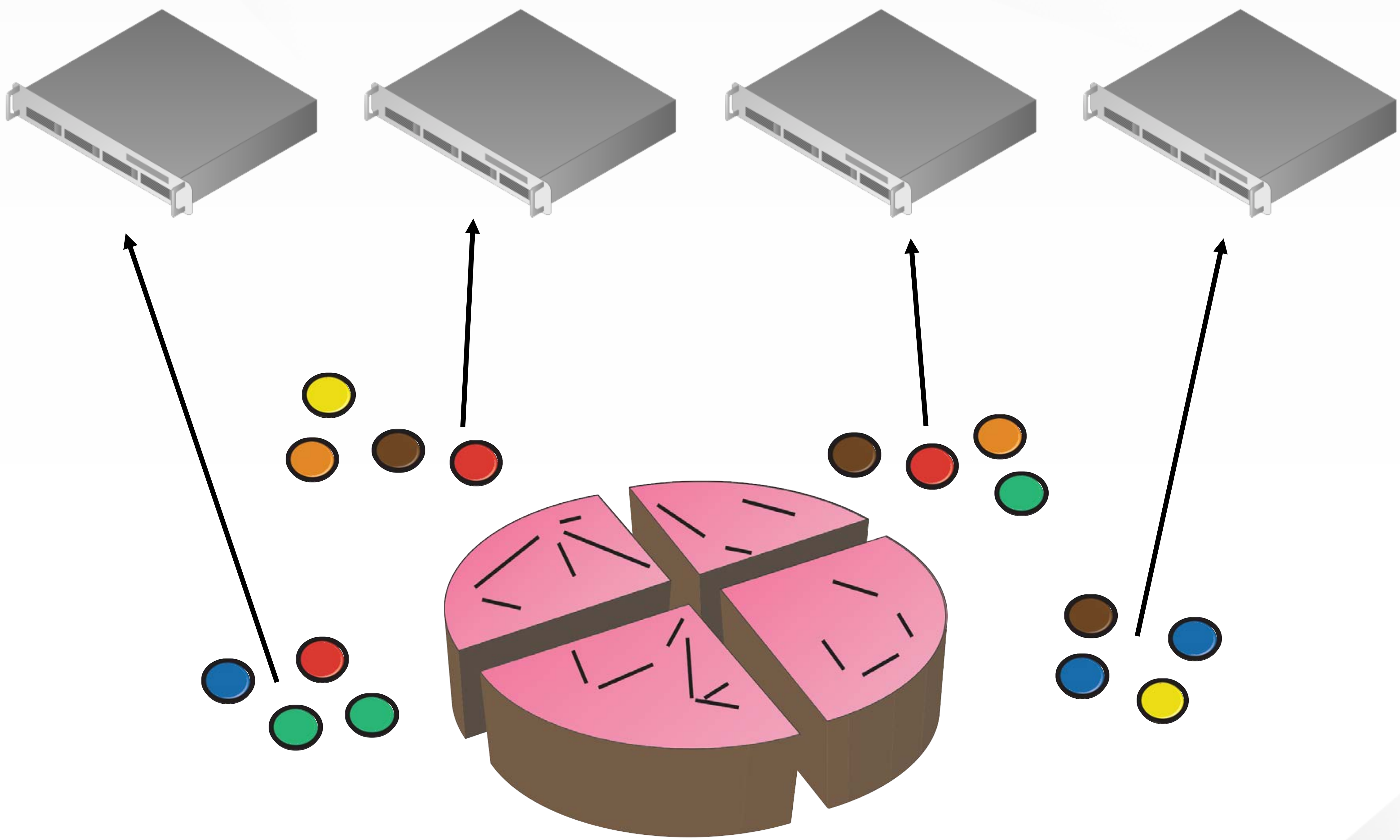
How do we split the input graph?



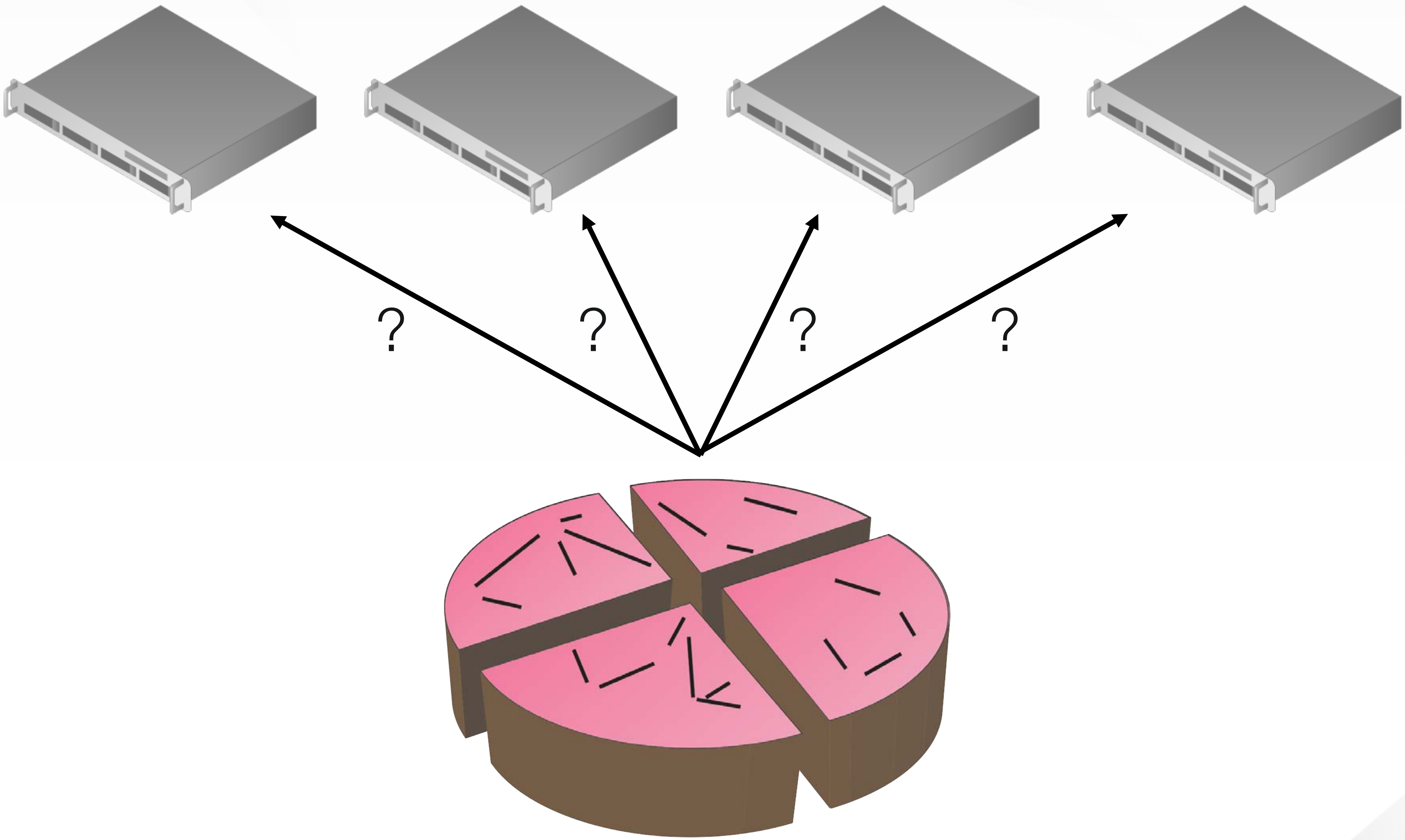
Answer: Split vertices in equal partitions!



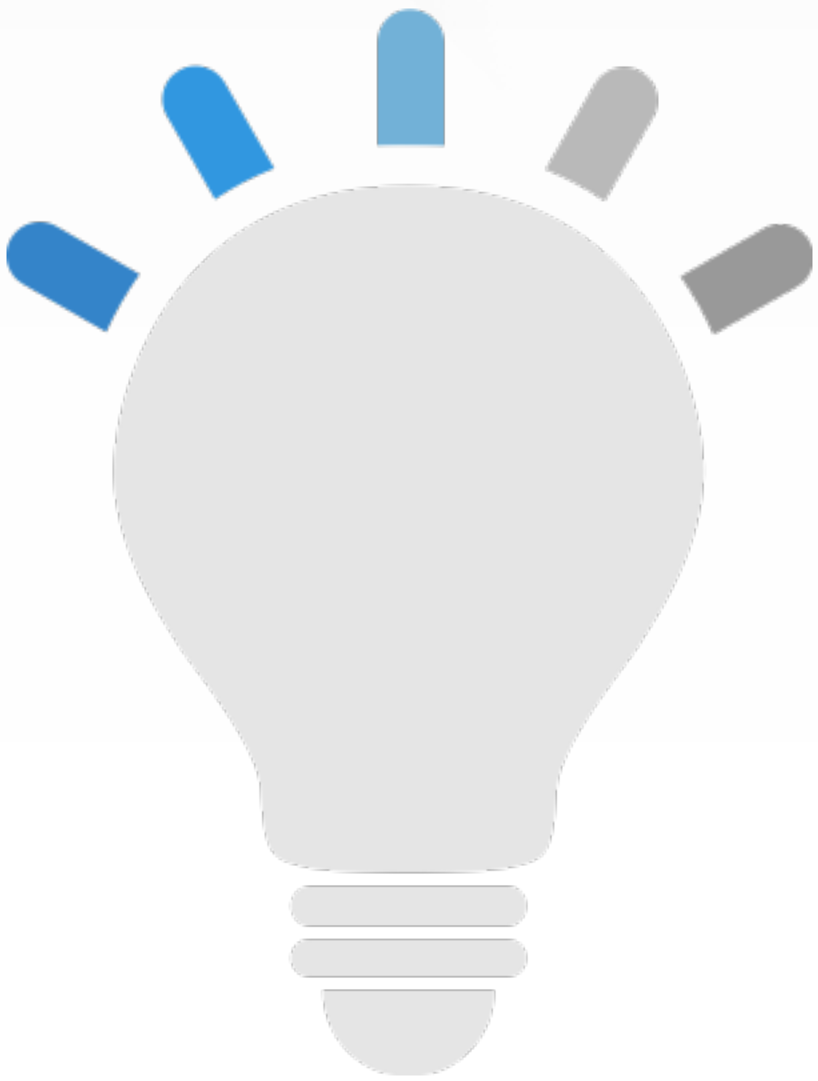
Distribute partitions equally



Where do we put the edges?



Insight



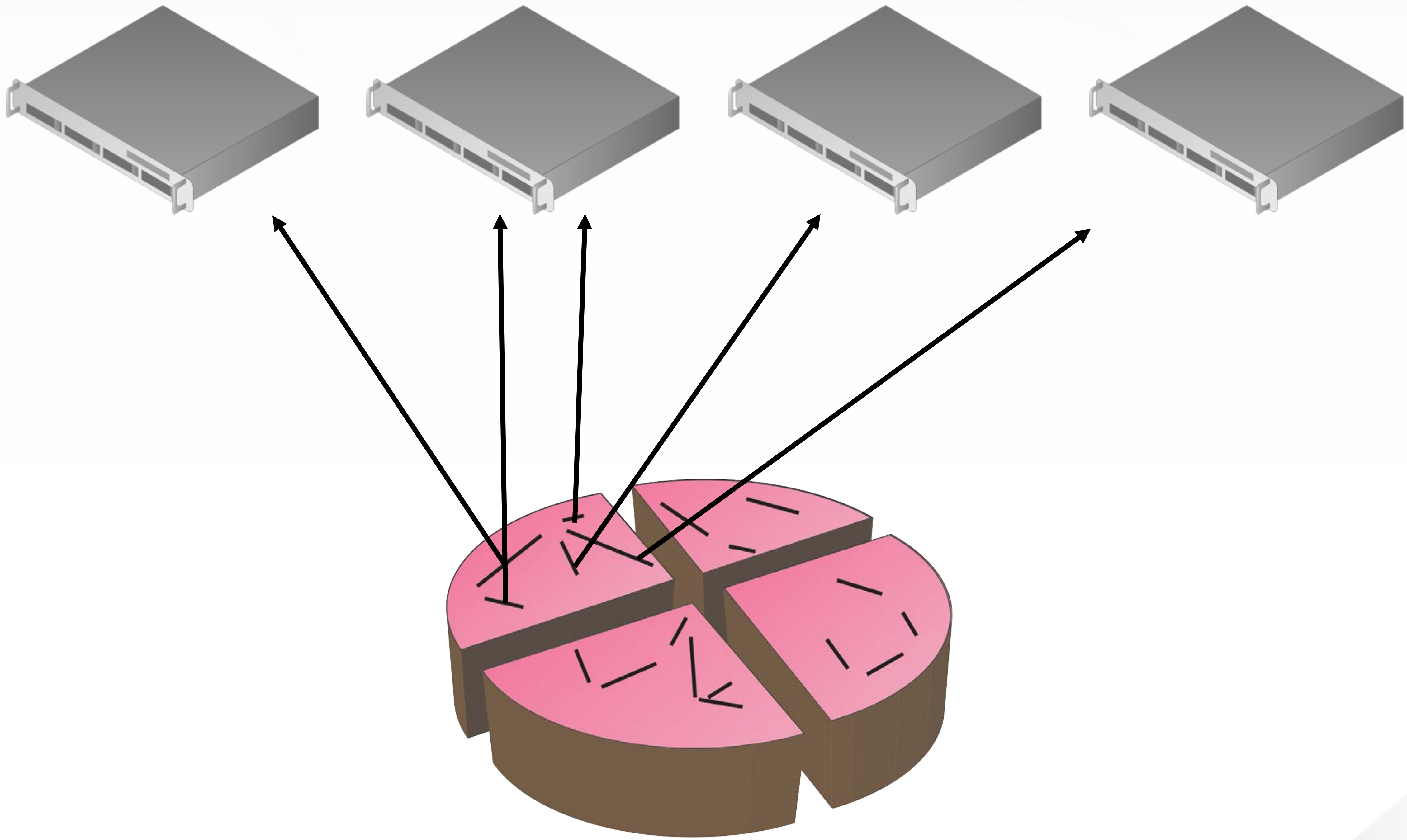
For secondary storage in a cluster:

Locality hardly matters

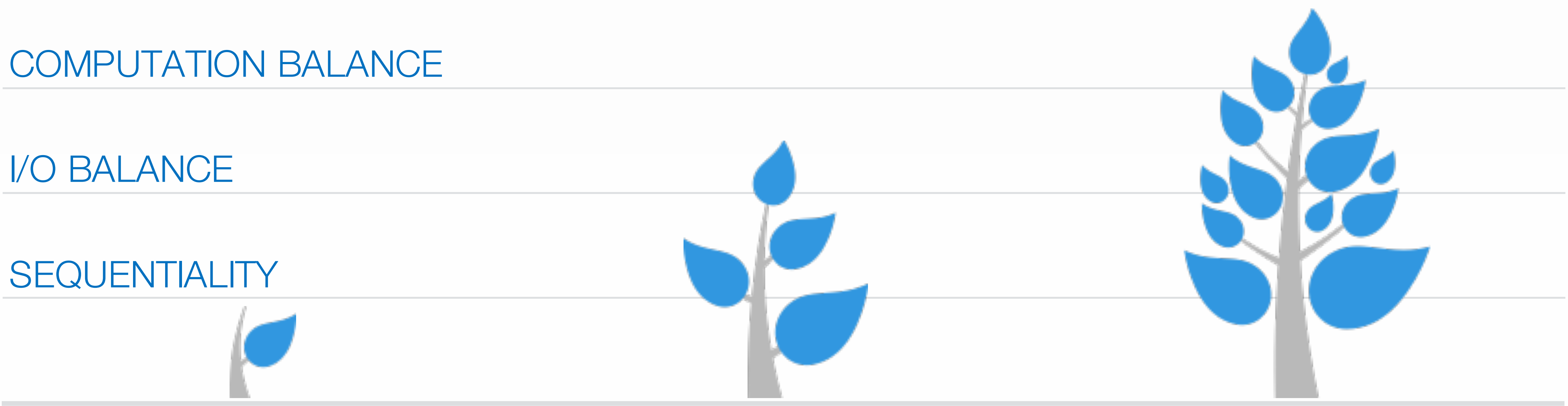
⇒ Remote bandwidth ~ local bandwidth

⇒ Edges need not be stored with their streaming partition

Answer: Randomly distribute them!



Chaos Design Approach



1. X-Stream

Edge-centric

Streaming partitions

2. Scale-out

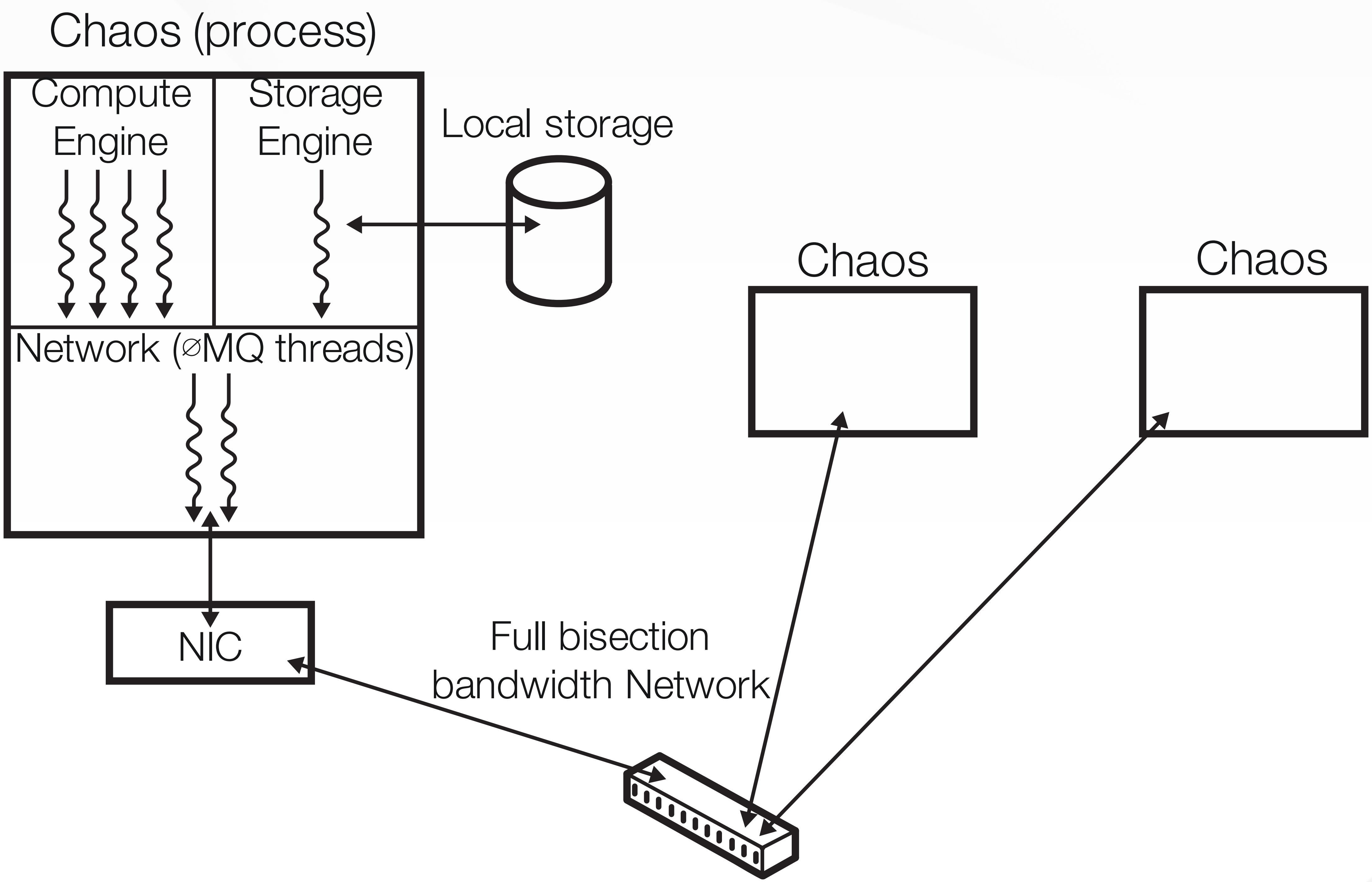
Flat storage

Distribute partitions

3. Chaos

Work stealing

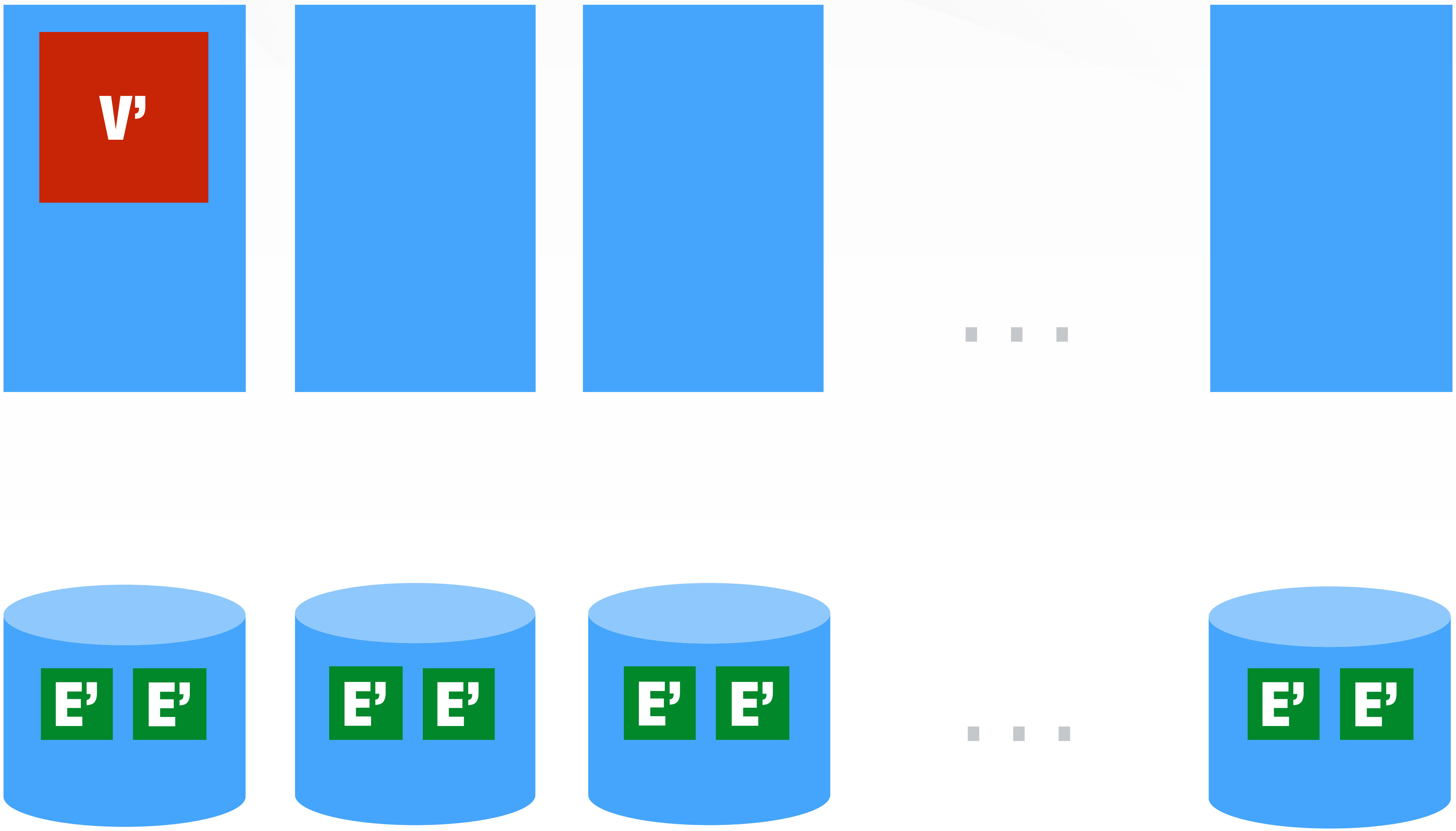
Chaos Architecture



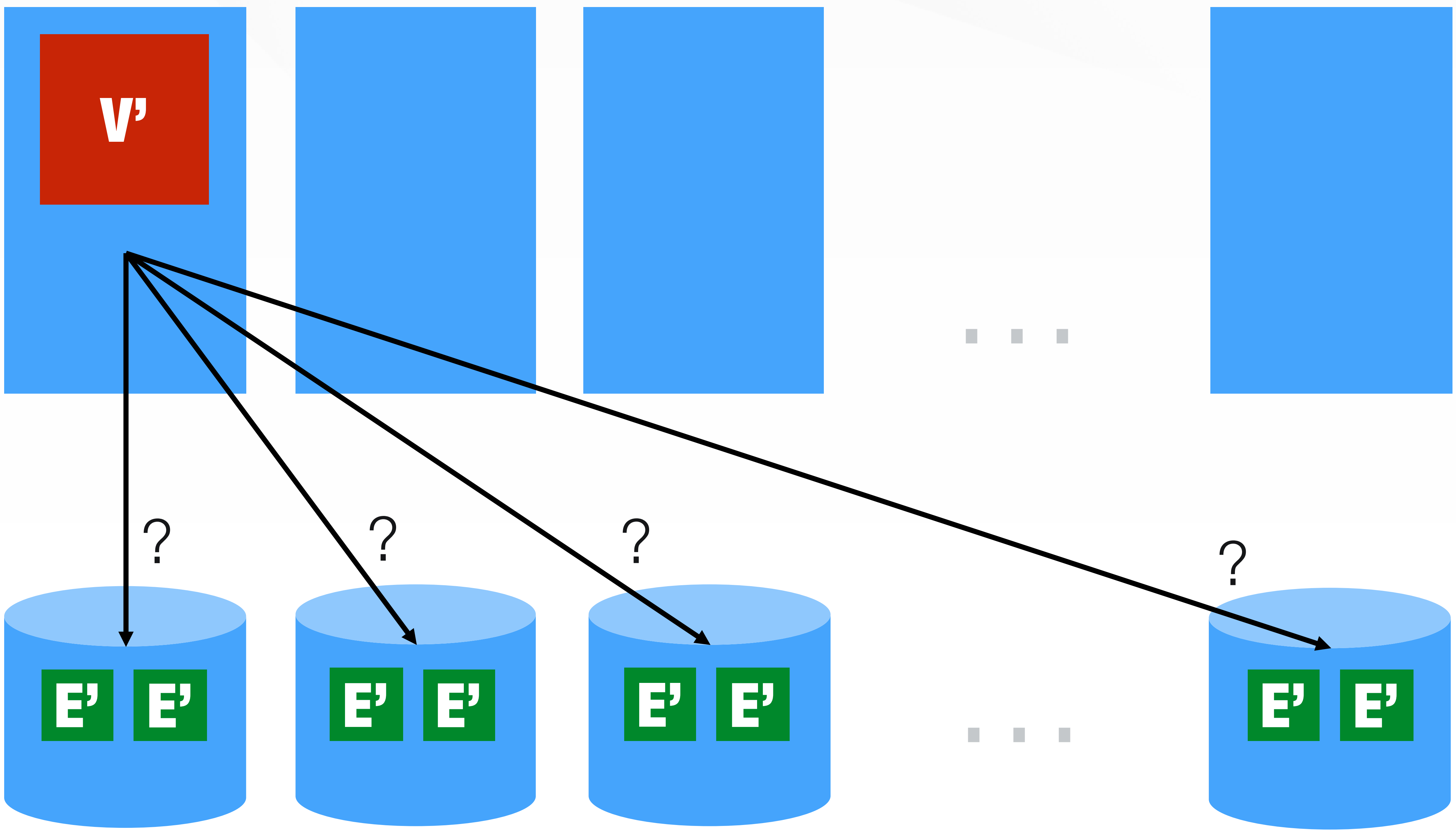
I/O Design

- Principle: Do not worry about locality
- Stripe graph data across nodes
  - Edge lists
  - Update lists

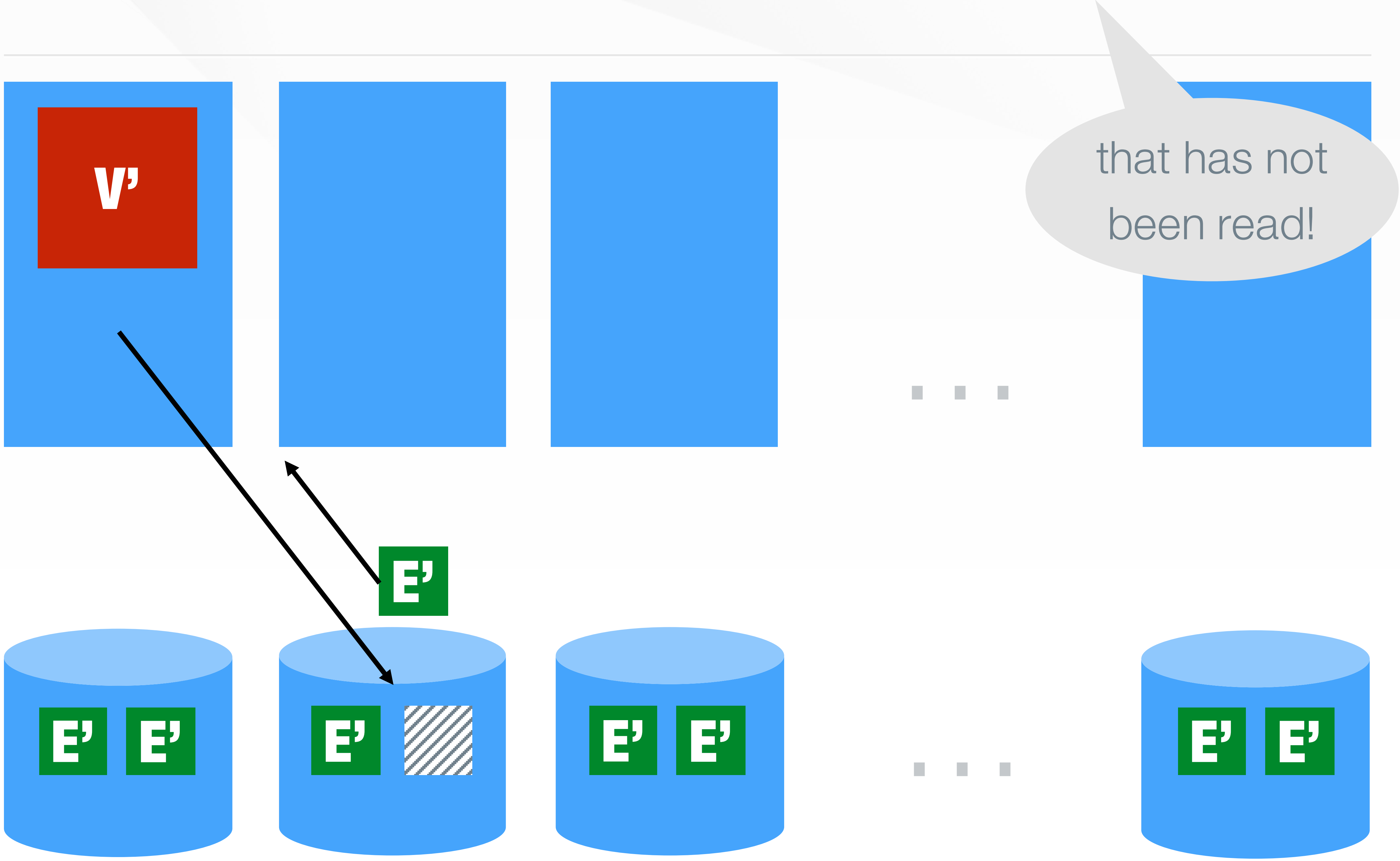
Vertex and Edge Distribution



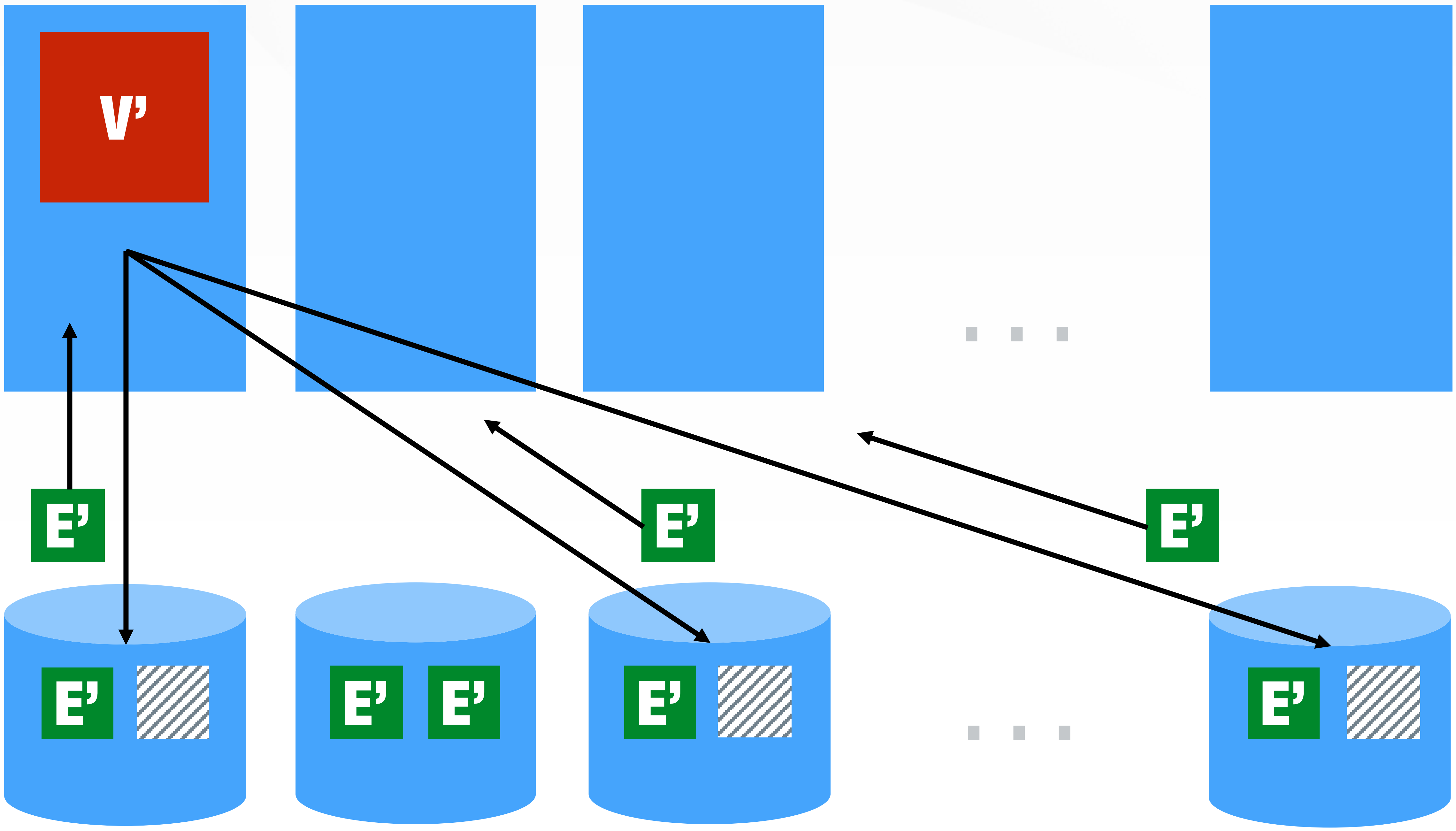
Where do we read the next edge stripe?



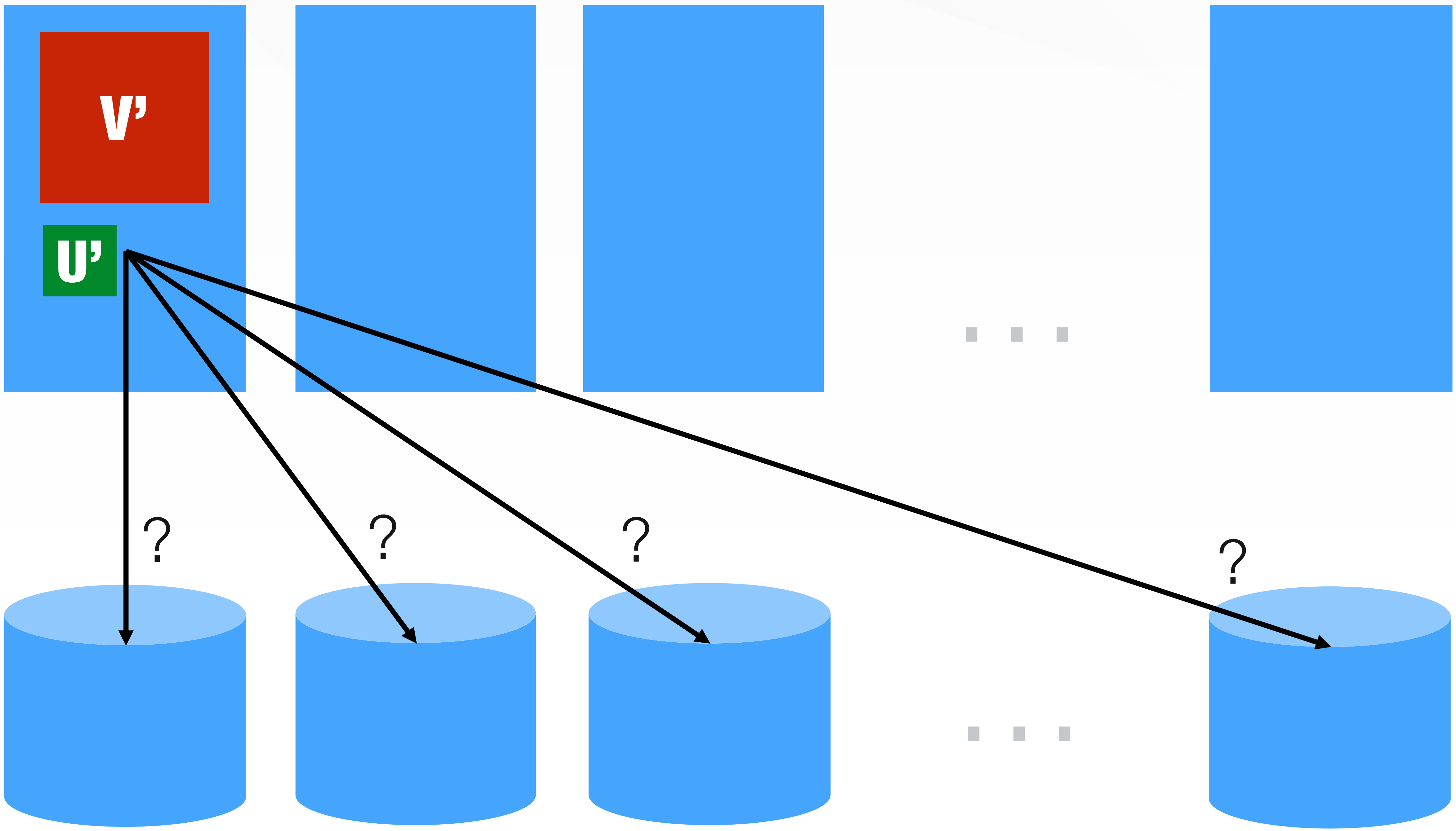
Answer: From any random stripe!



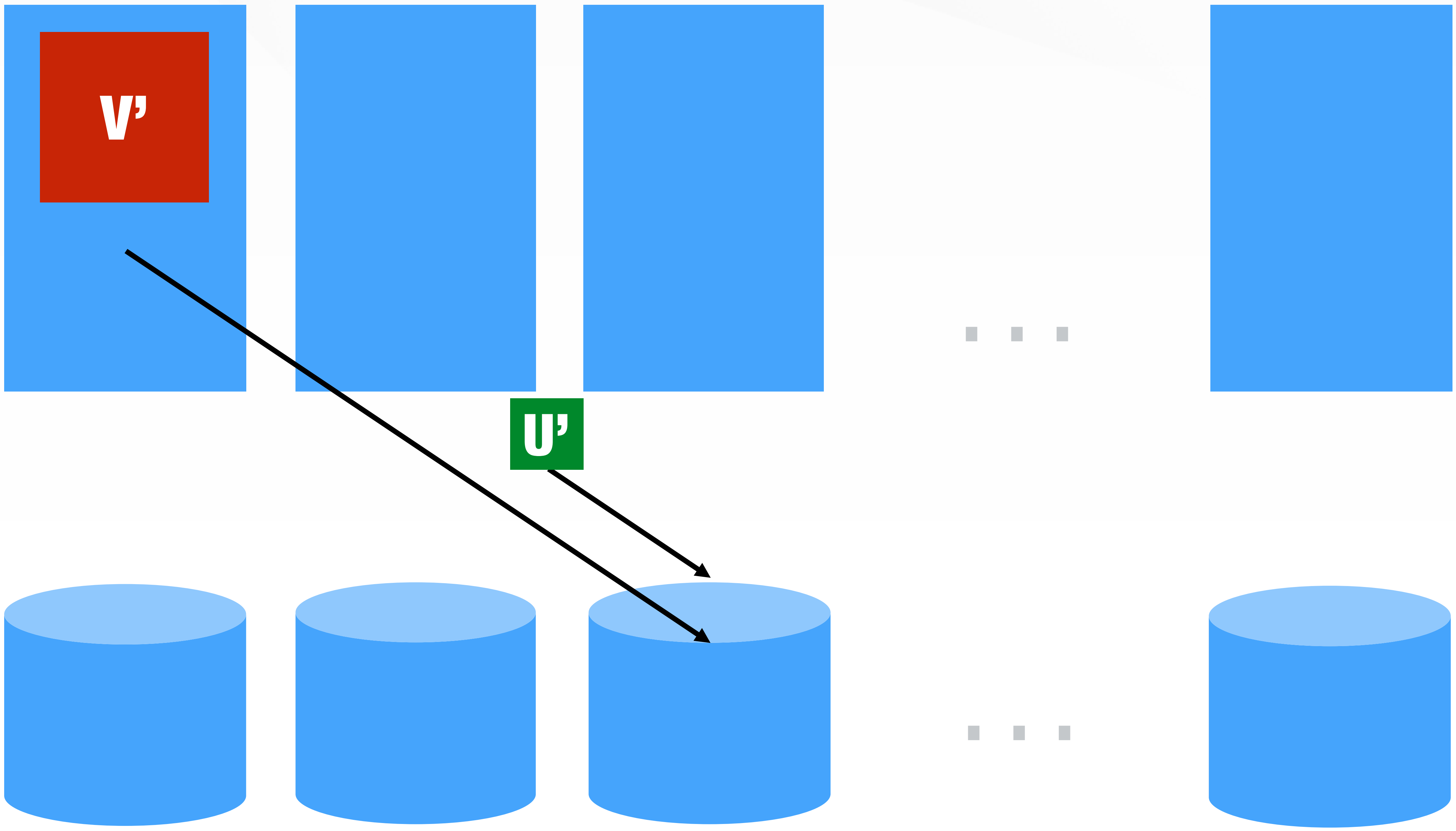
In fact, we read several stripes…



Where do we write the update stripe?



Answer: choose any device at random



I/O Design – Summary

FLAT STORAGE

Without any access ordering

Without any central entity

Computation Design

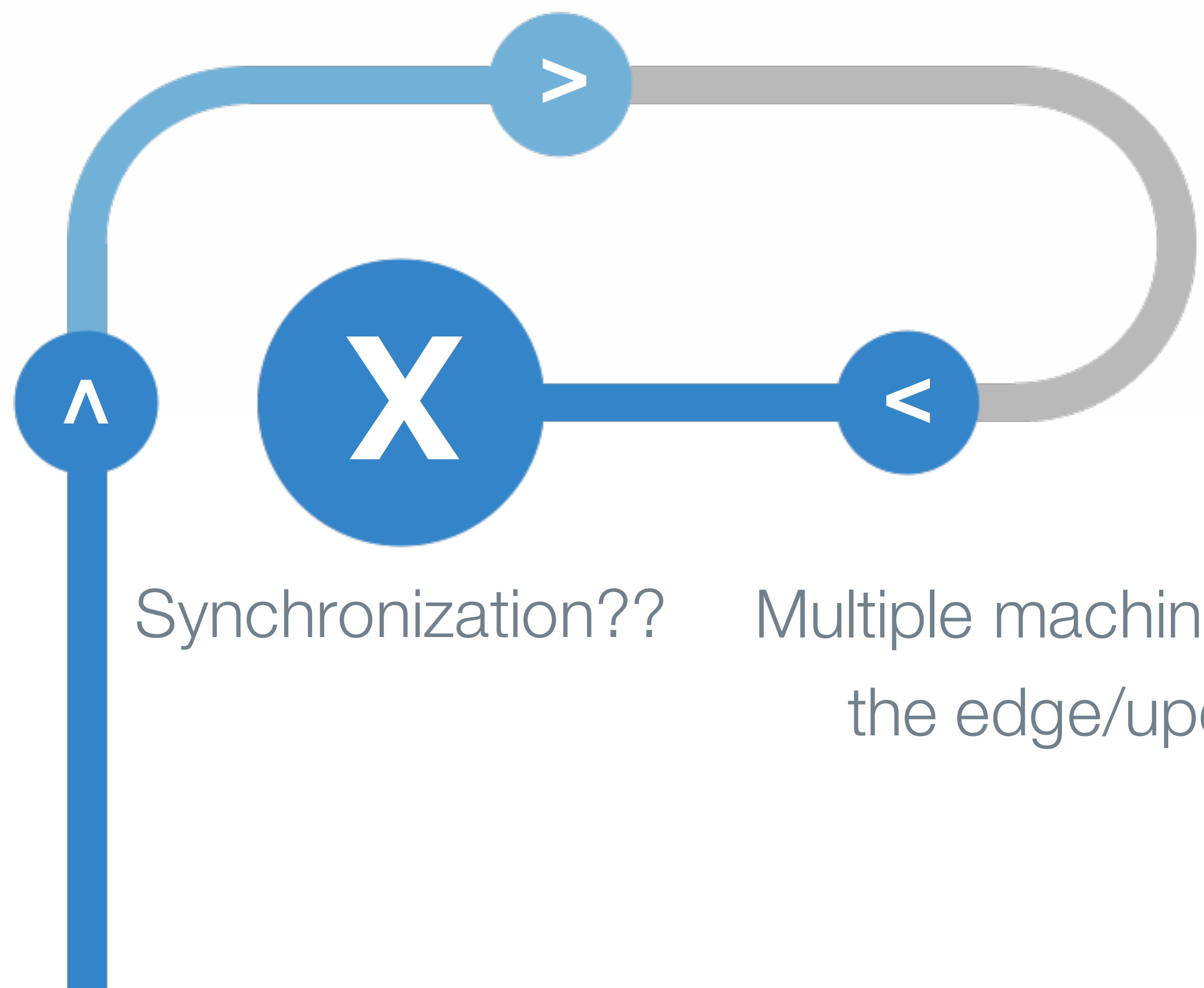
- Principle: work stealing

- Start: one streaming partition per node
- Work stealing deals with load imbalance

Work Stealing Issue?

Multiple machines can work on
the same streaming partition

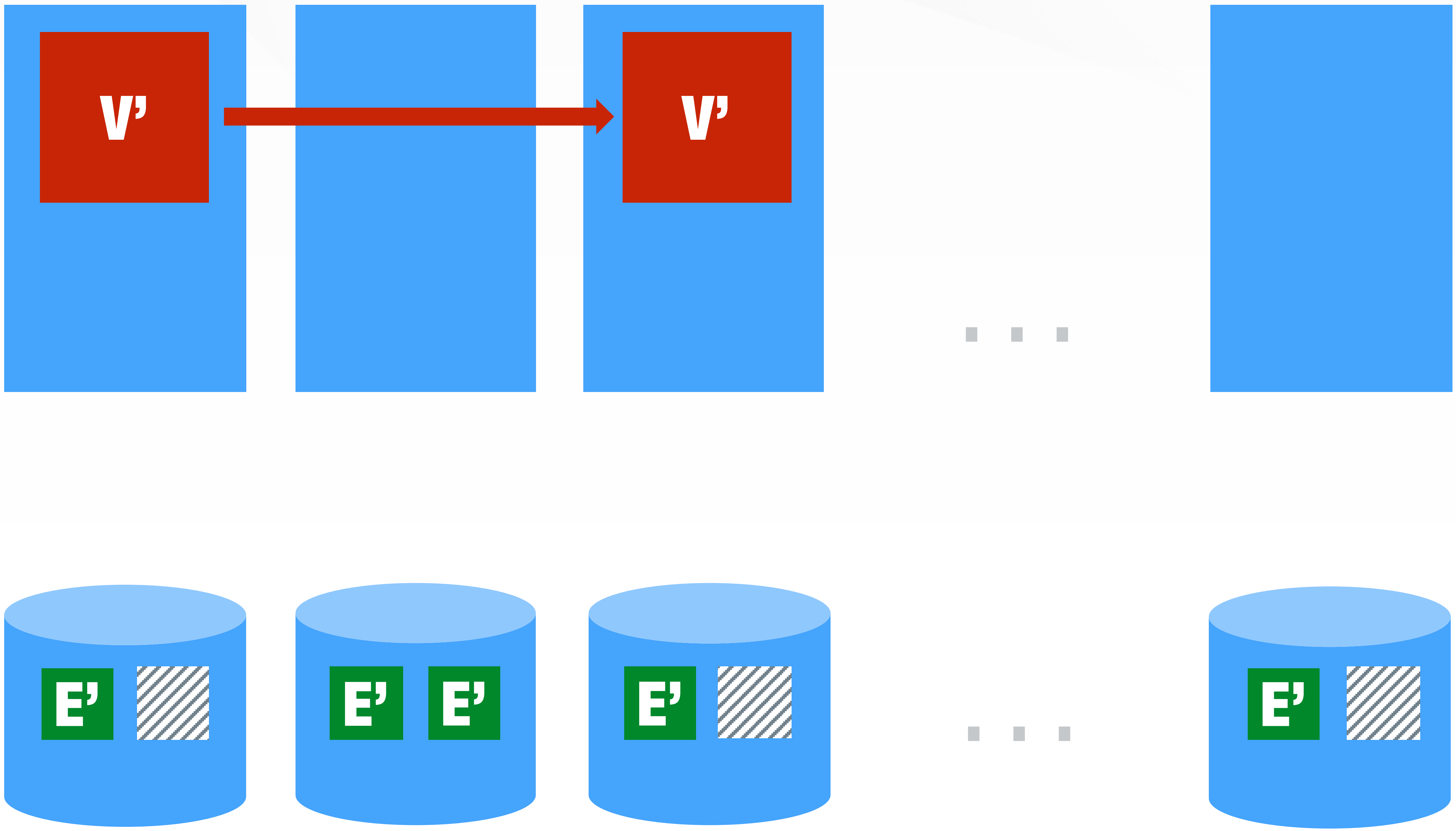
Computation
Imbalance



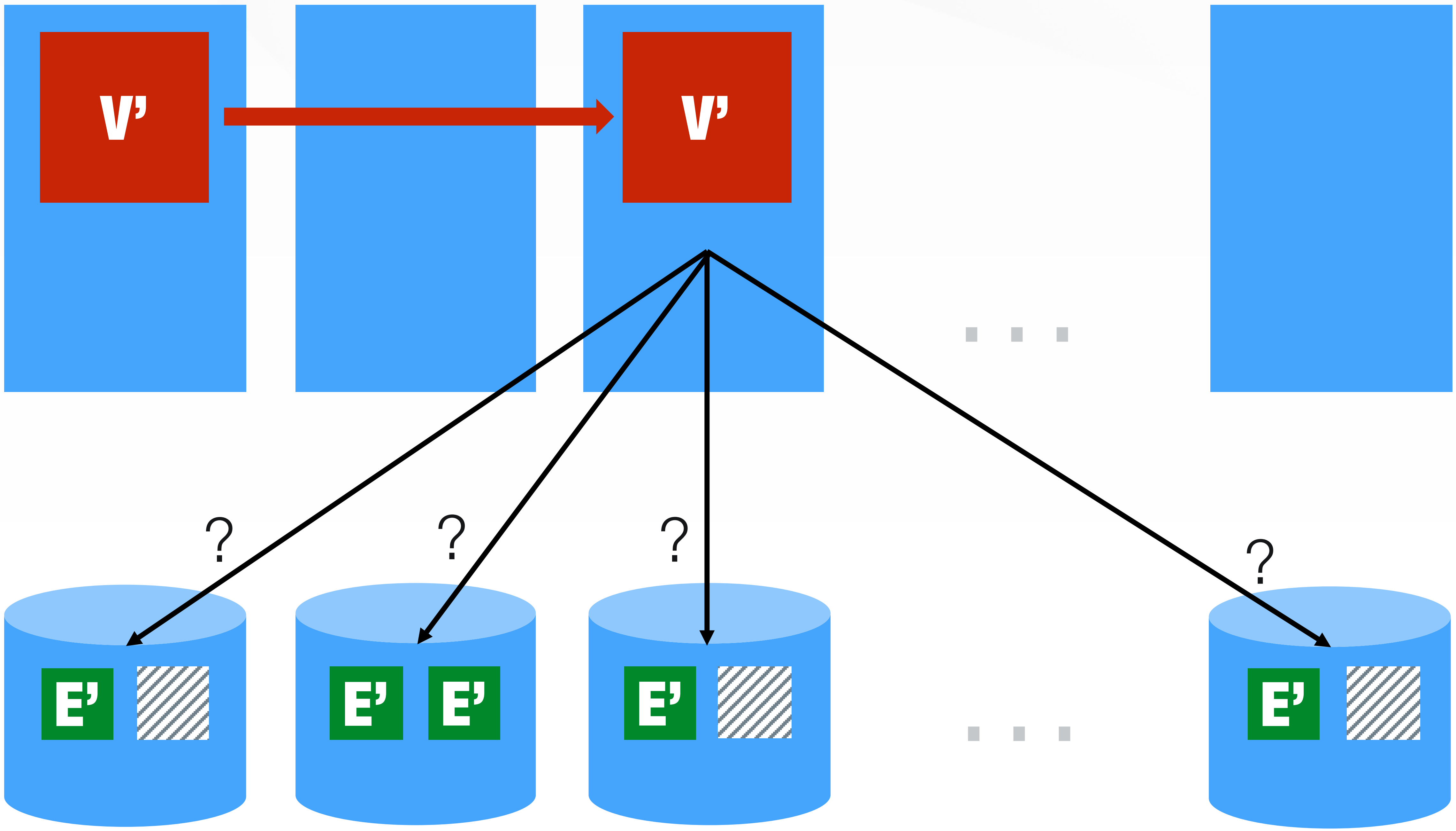
Synchronization??

Multiple machines access
the edge/update list

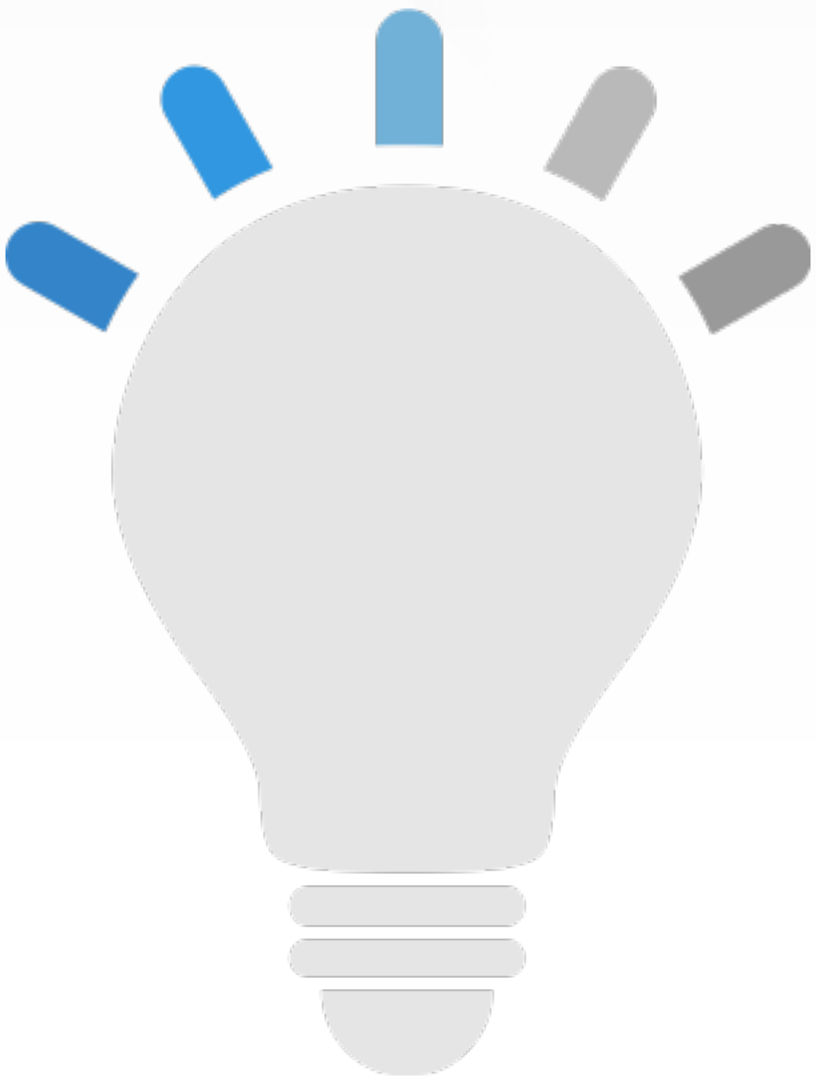
Stealing: Copy Vertex Accumulators



Stealing: Which edge stripe do we read?



Insight



For `scatter` and `gather`:

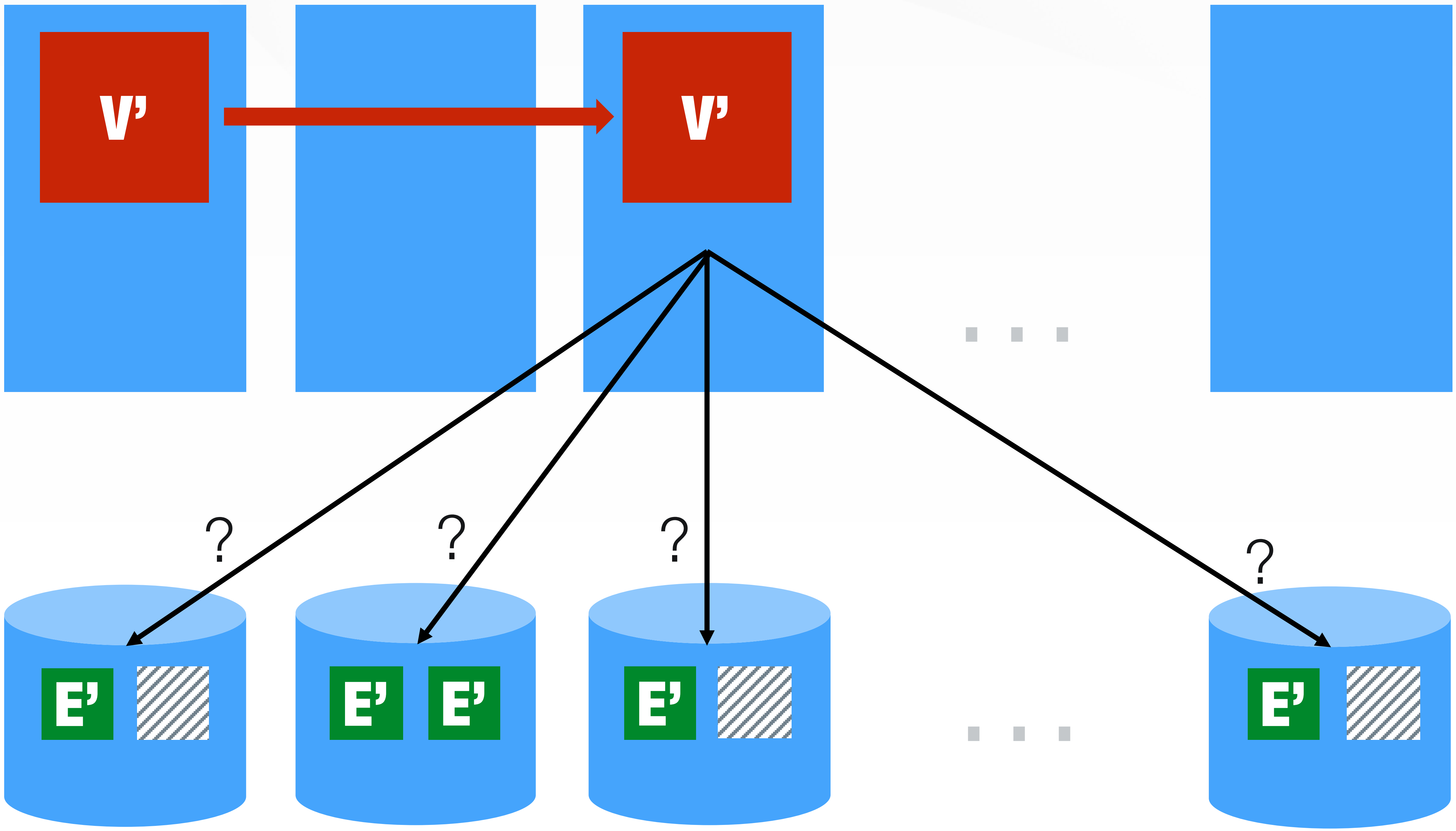
Order does not matters

⇒ No need for explicit division of work

⇒ Multiple machines can work on the same partition simultaneously

Answer: Any random stripe!

that has not been read!



Computation Design – Summary

WORK STEALING

Without synchronization

Without a centralized entity

Consequence of work stealing:

Gather phase requires a **`merge`** function

Chaos – Summary

Striping	→	**I/O balance**
Work stealing	→	**Computation balance**
Streaming partitions	→	**Sequentiality**

We achieve all of this

- without expensive partitioning
- without I/O synchronization

Evaluation

- 32 nodes (one rack)
- 32GB RAM, 480GB SSD, 2x6TB HDD
- Full-bisection bandwidth 40GigE switch

Results Overview

1.61X

Average Scaling Factor from 1 machine to 32 (weak scaling)

13X

Average Speedup with 32 machines (strong scaling)

9 hours

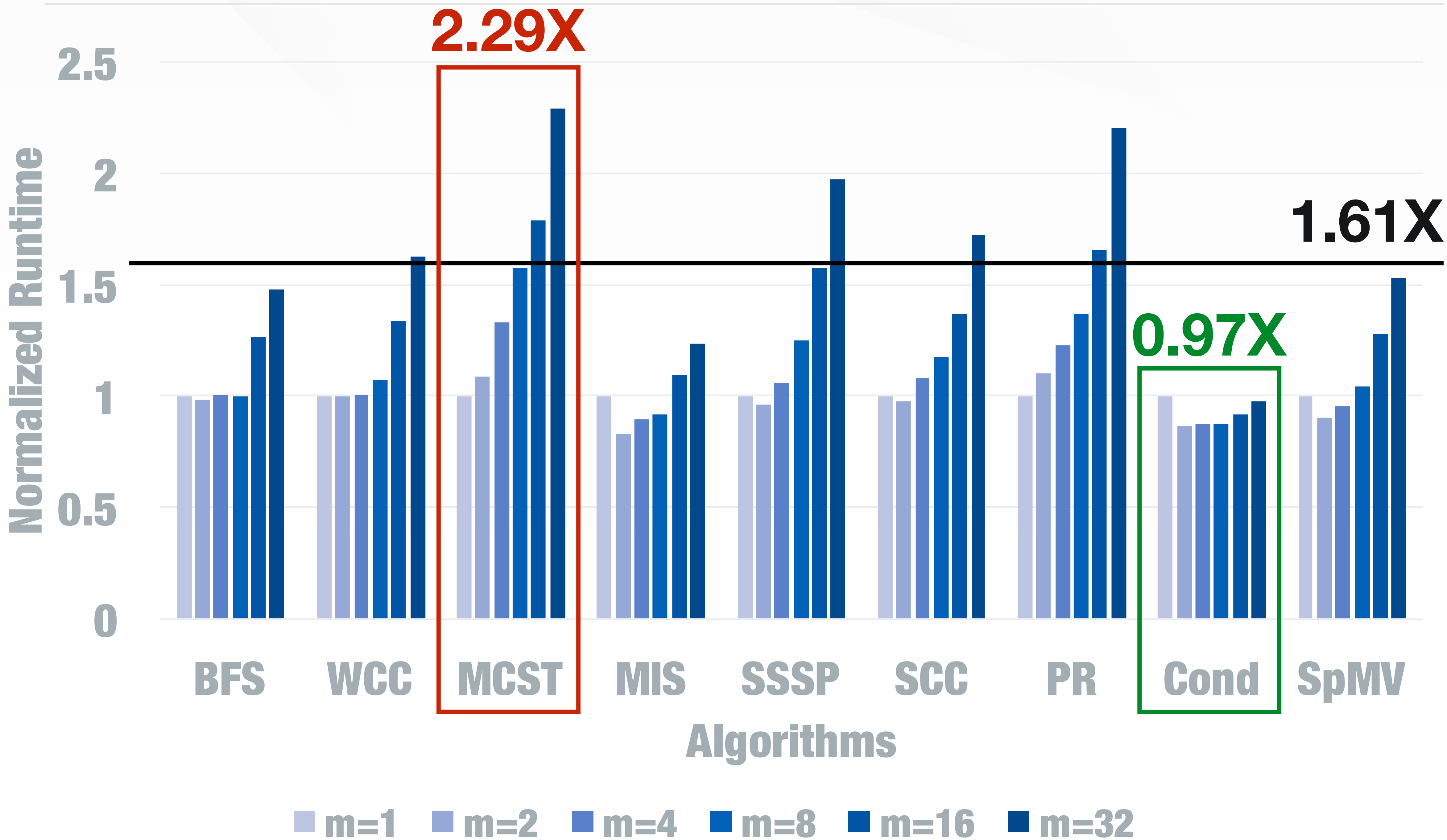
to solve BFS on a graph with 1T edges

19 hours

to run 5 iterations of PageRank on a graph with 1T edges

Weak Scaling Results

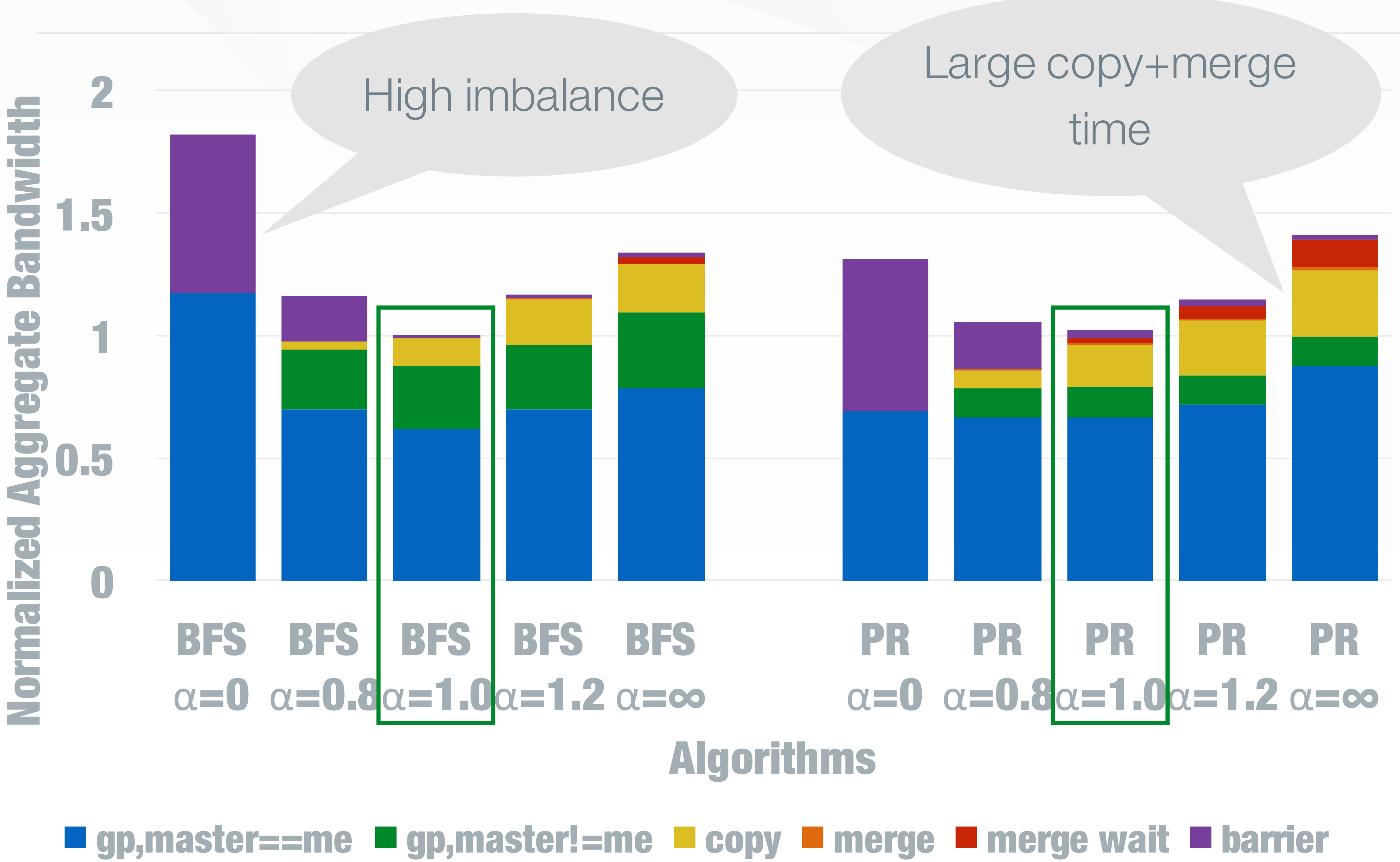
Input size doubles when m doubles



Why does it not stay at 1?

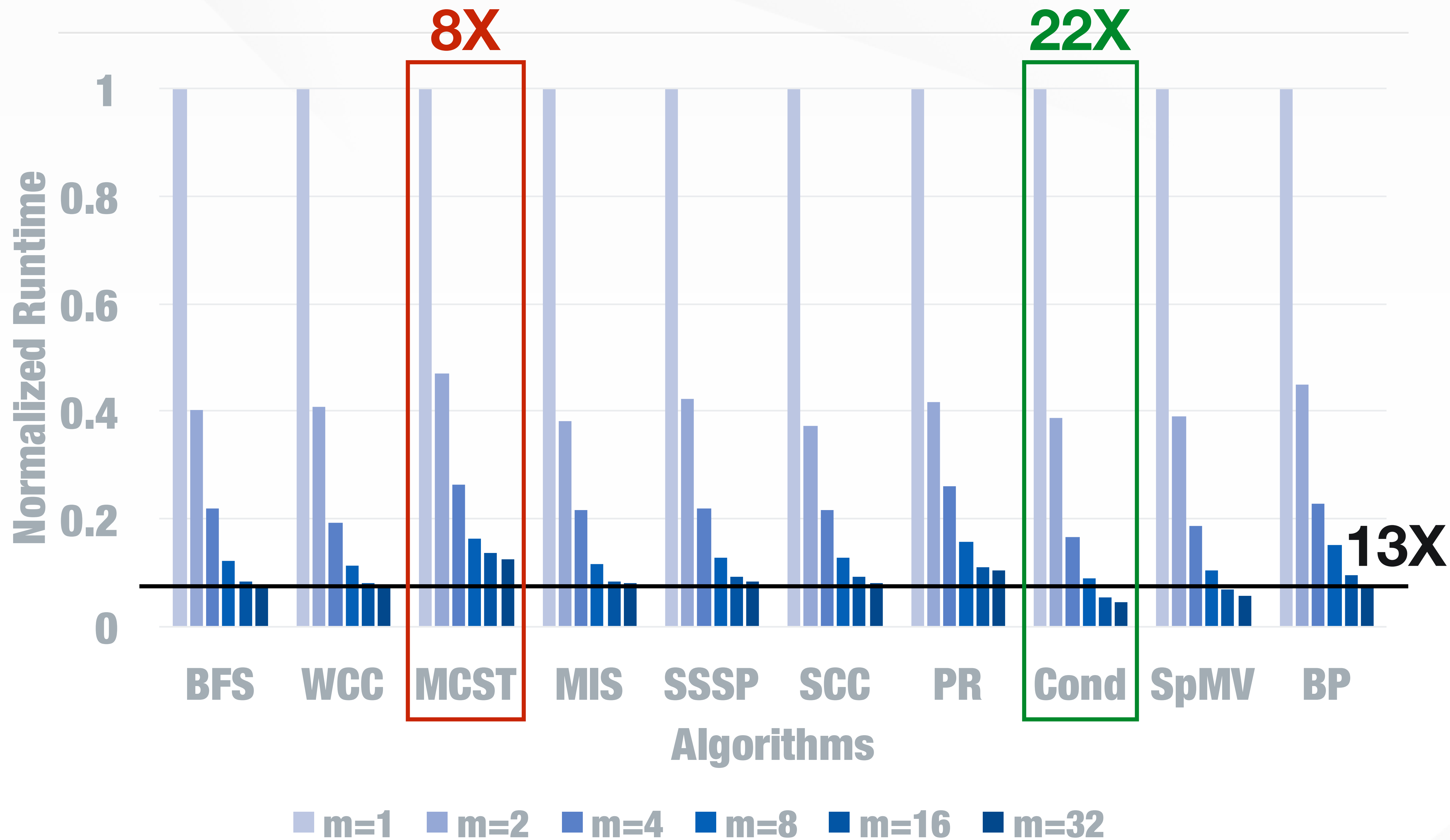
- Load balance isn't perfect
- Work stealing introduces overheads
- Some algorithms generate more updates

Is work stealing optimal?



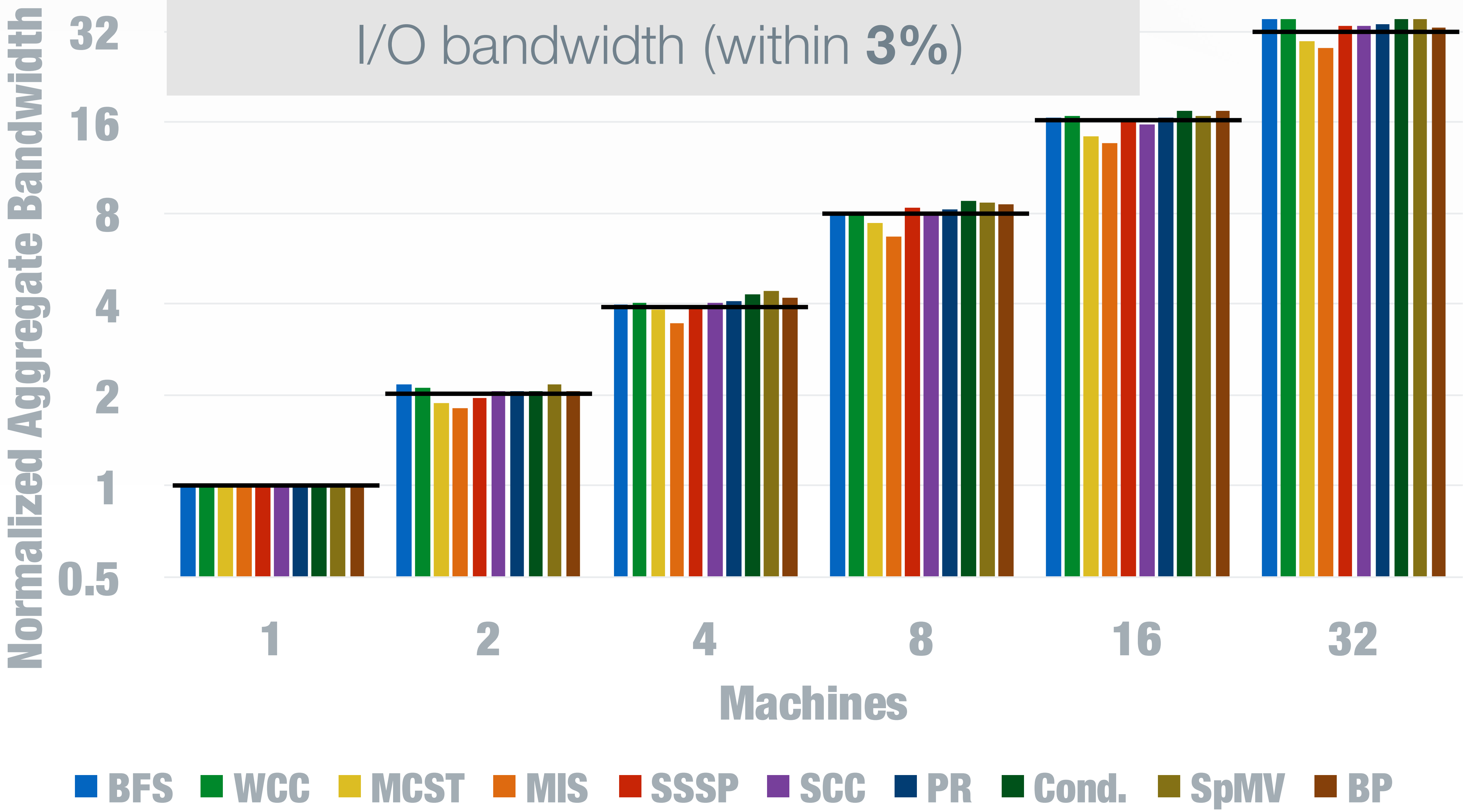
Strong Scaling Results

Input size remains fixed



Why does it scale well?

Chaos runs constantly at near-maximum I/O bandwidth (within **3%**)



Are there any scaling limitations?

Network

Bottleneck resource

Performance suffers if network < storage

Full-bisection bandwidth necessary

Storage

Bottleneck resource

Unrealized performance (waste)

if storage < network

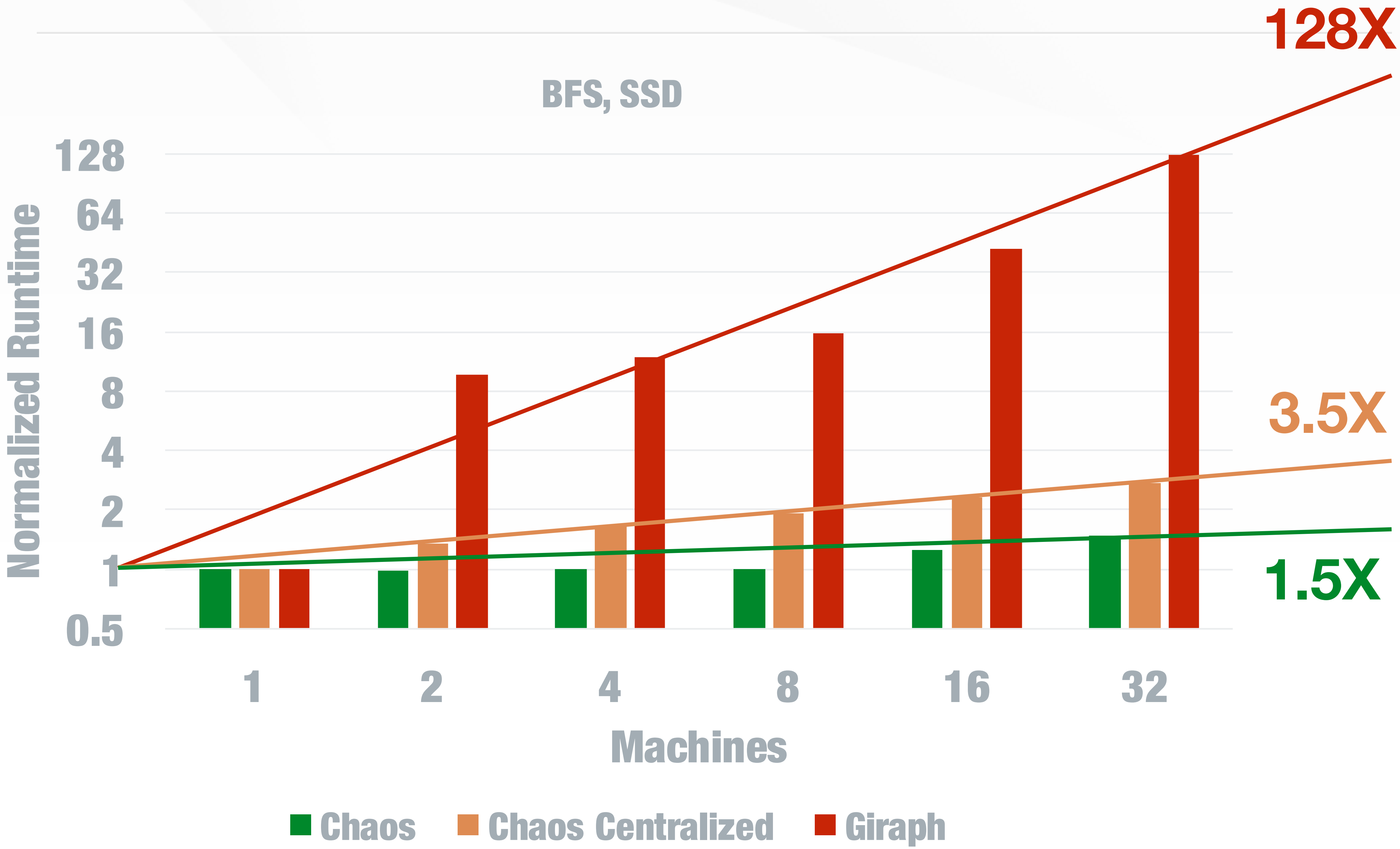
Compute

Low impact

Possible minor impact on performance

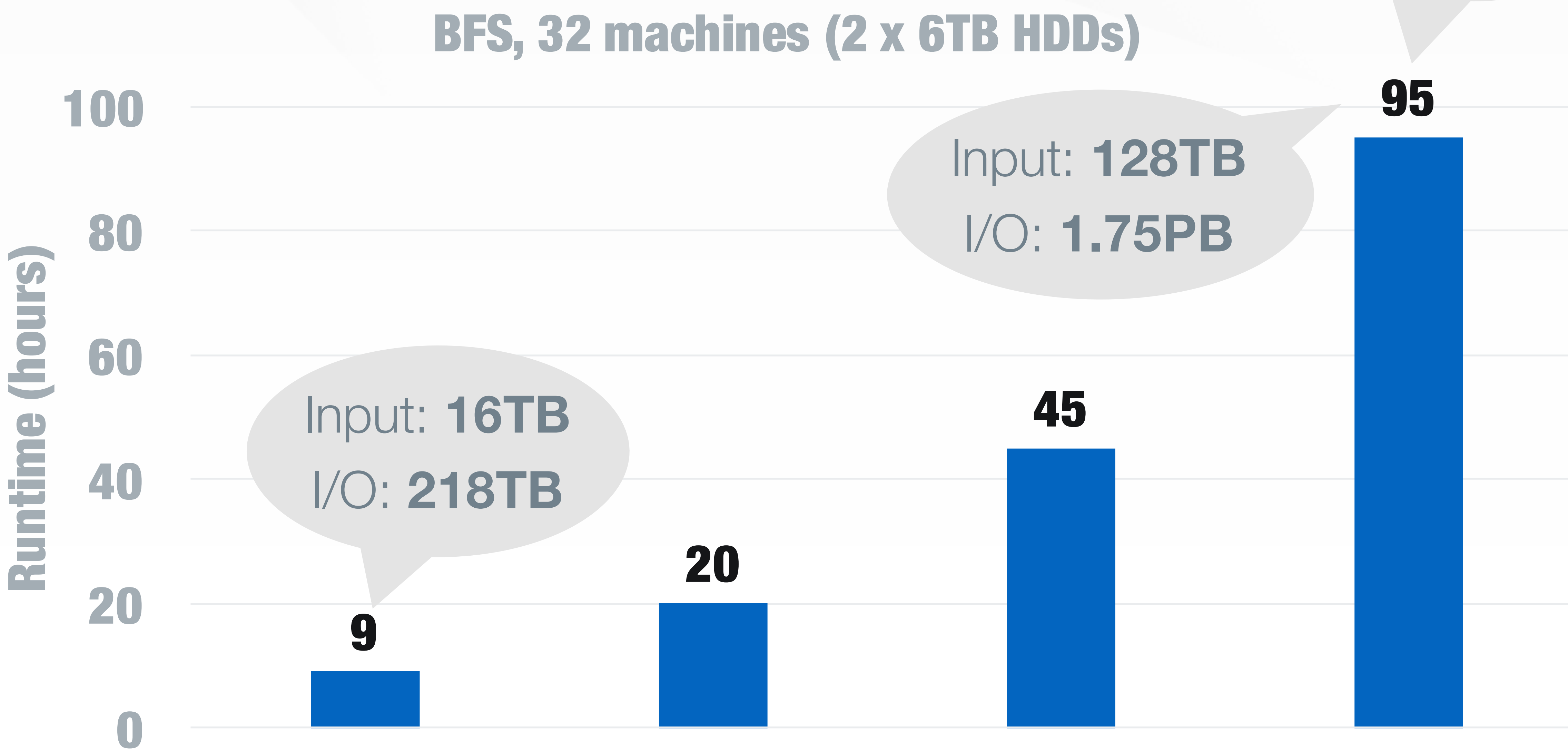
in some applications (e.g., FPU intensive)

Comparison to Other Systems



How large can we go?

Only **8X** smaller than the largest graph ever processed

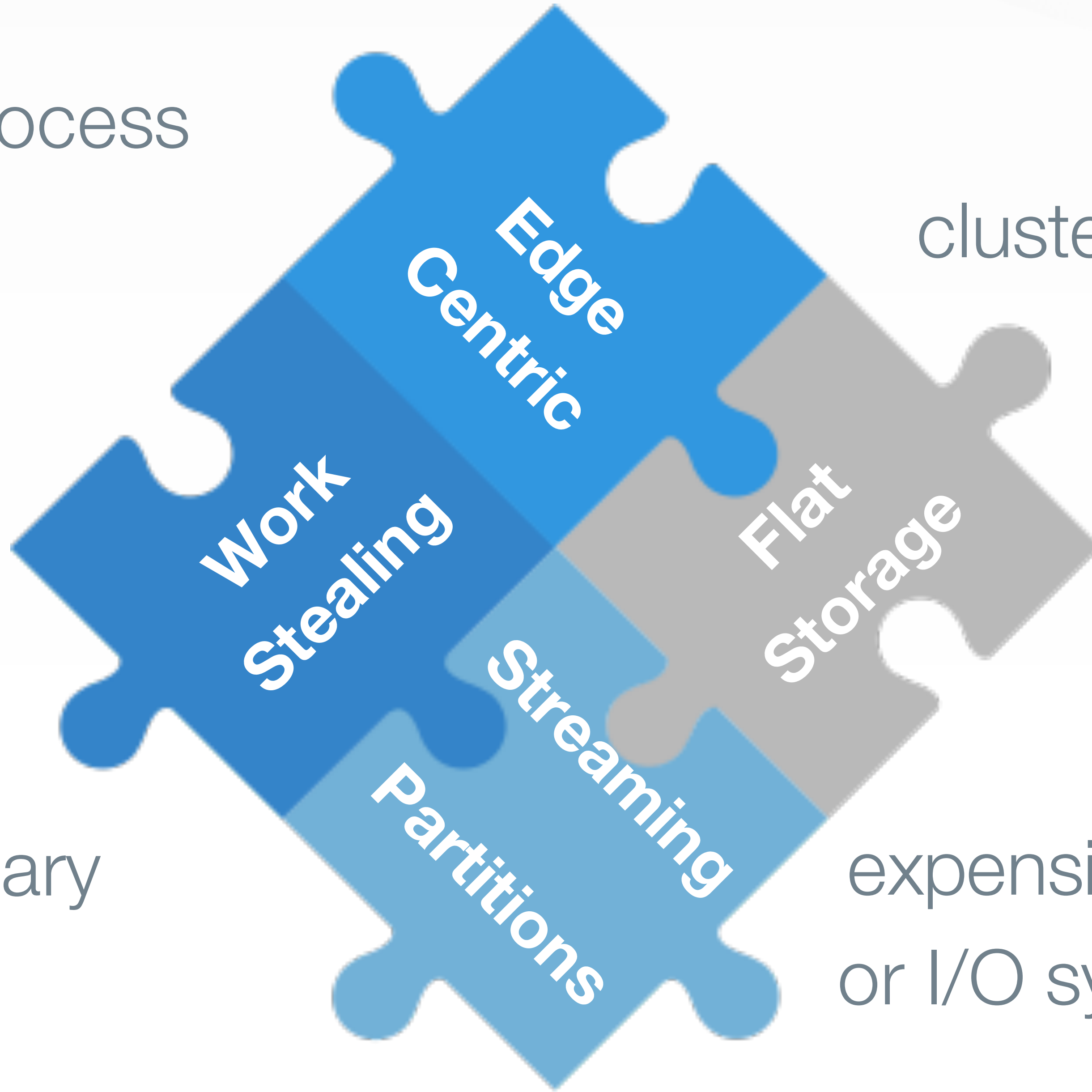


If you can store the graph, you can compute on it!

Conclusion

Chaos can process large graphs

On a small cluster of machines



Using secondary storage

Without expensive partitioning or I/O synchronization

CHAOS

Scale-out Graph Processing from Secondary Storage

https://github.com/labos-epfl/chaos

http://labos.epfl.ch/

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Fork me on GitHub