EVAN WEST, STONY BROOK UNIVERSITY

GRAPHZEPPELIN

PROCESSING ENORMOUS, CHANGING GRAPHS

GRADUATE RESEARCH DAY, 2022

EVAN WEST, STONY BROOK UNIVERSITY

GRAPHZEPPELIN

PROCESSING ENORMOUS, CHANGING GRAPHS WITH LINEAR-SKETCHING MADE USEFUL VIA ALGORITHMIC IMPROVEMENTS AND EXTERNAL MEMORY DATA-STRUCTURES

GRADUATE RESEARCH DAY, 2022

EVAN WEST, STONY BROOK UNIVERSITY

GRAPHZEPPELIN

PROCESSING ENORMOUS, CHANGING GRAPHS WITH LINEAR-SKETCHING MADE USEFUL VIA ALGORITHMIC IMPROVEMENTS AND EXTERNAL MEMORY DATA-STRUCTURES

GRADUATE RESEARCH DAY, 2022

GRAPHZEPPELIN AUTHORS

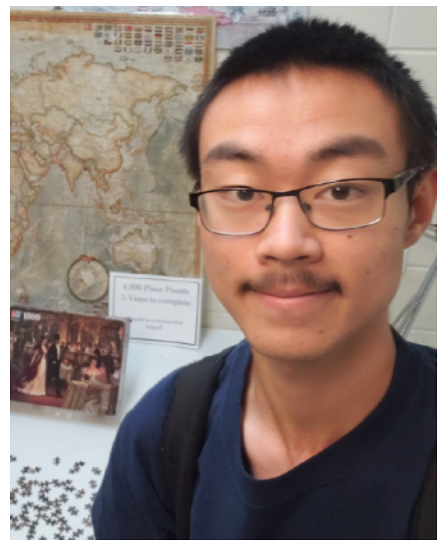
David Tench
Stony Brook University

Evan West
Stony Brook University

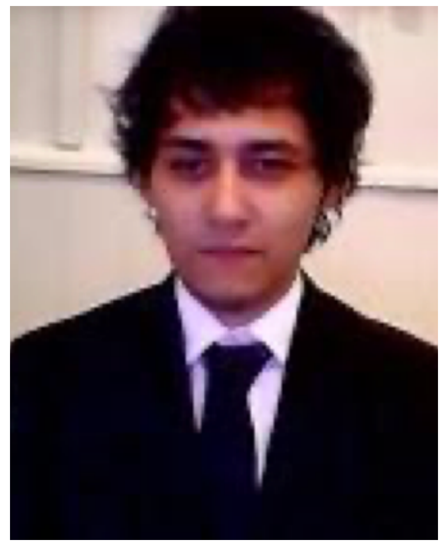
Victor Zhang
Rutgers University

Michael Bender
Stony Brook University

Martin Farach-Colton
Rutgers University

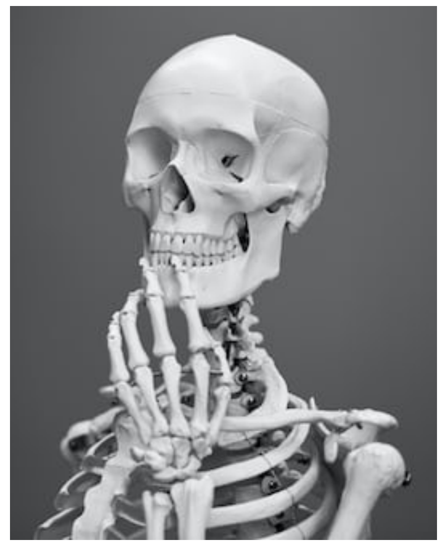


Kenny Zhang
Stony Brook University



Abiyaz Chowdhury
Stony Brook University

J. Ahmed Dellas
Rutgers University



Tyler Seip
MongoDB

TWO YEARS AGO . . .

Hi David! I'd like to do research
and use my coding skills

A group of us are implementing Ahn,
Guha, and McGregor's algorithm
[SODA12] for the dynamic streaming
connected components problem.
It should be an easy publication …

TWO YEARS AGO . . .

This algorithm is useful! It can analyze massive changing graphs even when they're bigger than RAM. *So weird* it hasn't already been implemented. It uses this really cool technique called linear ske-

TWO YEARS AGO . . .

This algorithm is useful! It can analyze massive changing graphs even when they're bigger than RAM. *So weird* it hasn't already been implemented. It uses this really cool technique called linear ske-

David. You had me at easy publication

Oh yes. This algorithm uses asymptotically smaller memory than storing the entire graph. A straightforward implementation will *definitely* outperform state of the art systems.

TWO YEARS AGO . . .

This algorithm is useful! It can analyze massive changing graphs even when they're bigger than RAM. *So weird* it hasn't already been implemented. It uses this really cool technique called linear ske-

David. You had me at easy publication

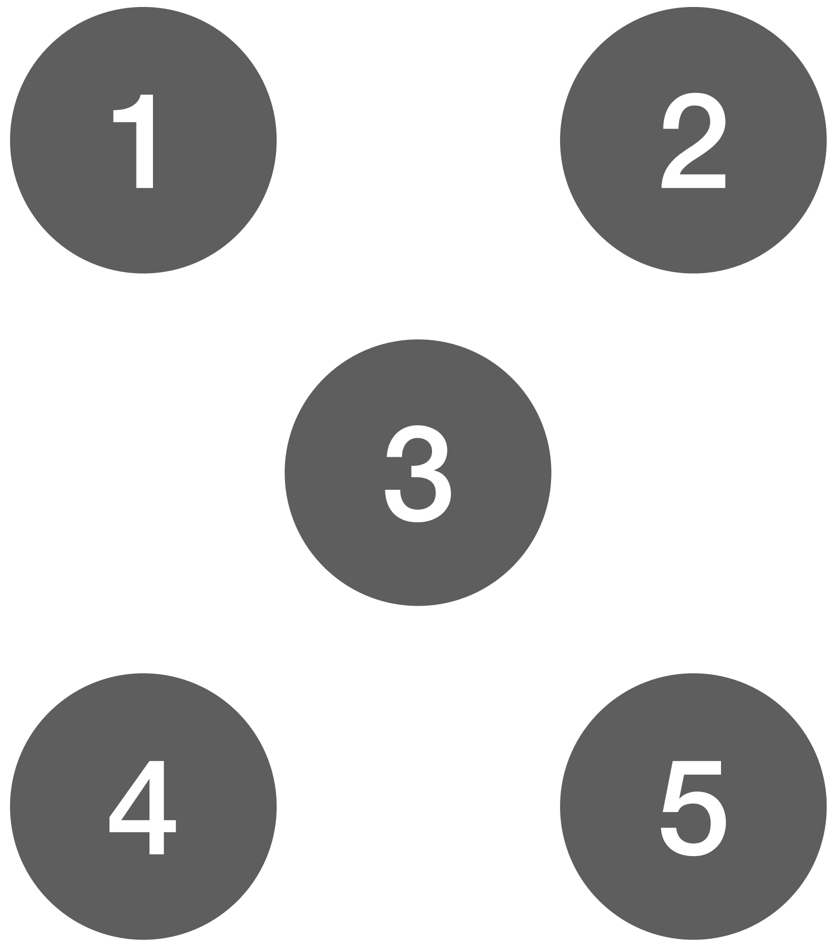
FORESHADOWING!

Oh yes. This algorithm uses asymptotically smaller memory than storing the entire graph. A straightforward implementation will *definitely* outperform state of the art systems.

PROCESSING ENORMOUS, CHANGING GRAPHS WITH LINEAR-SKETCHING MADE USEFUL VIA ALGORITHMIC IMPROVEMENTS AND EXTERNAL MEMORY DATA-STRUCTURES

DYNAMIC STREAMING CONNECTED COMPONENTS

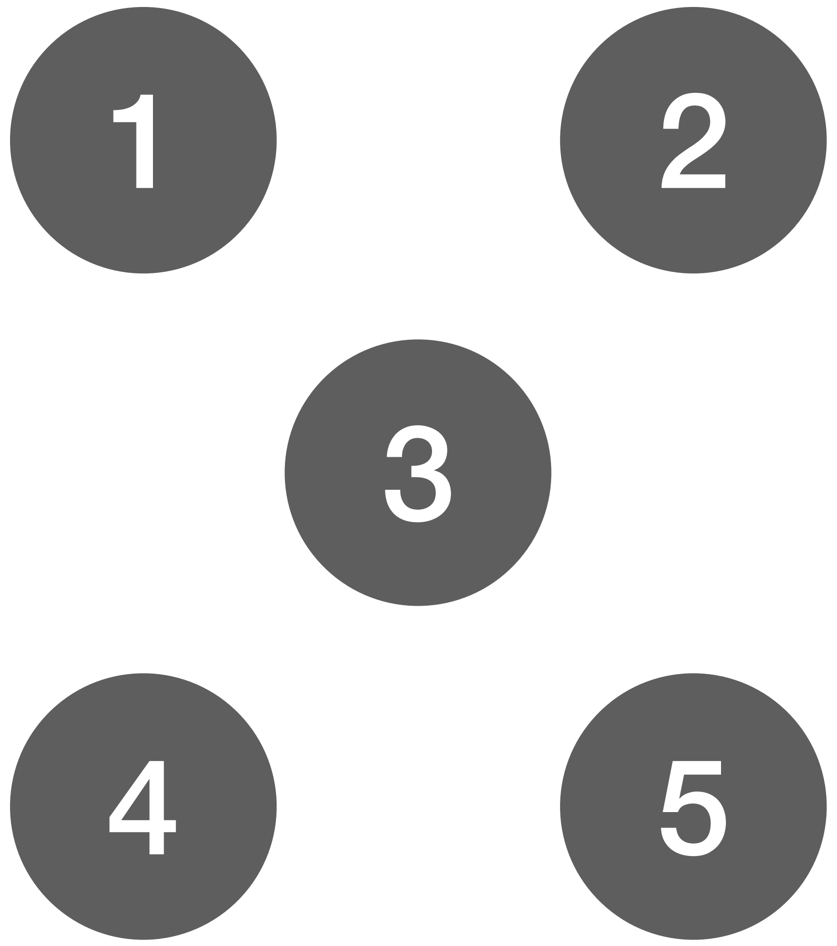
Goal: Find connected components of a graph with n nodes subject to stream of edge insertions and deletions.



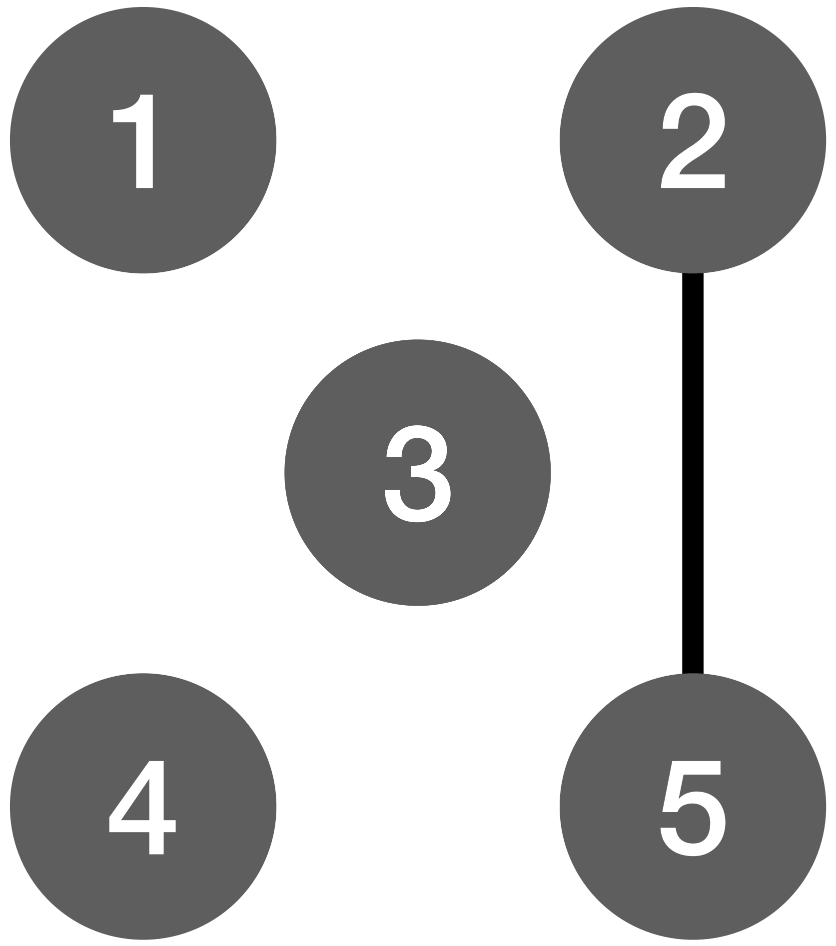
{{1}, {2}, {3}, {4}, {5}}
Initial State

DYNAMIC STREAMING CONNECTED COMPONENTS

Goal: Find connected components of a graph with n nodes subject to stream of edge insertions and deletions.



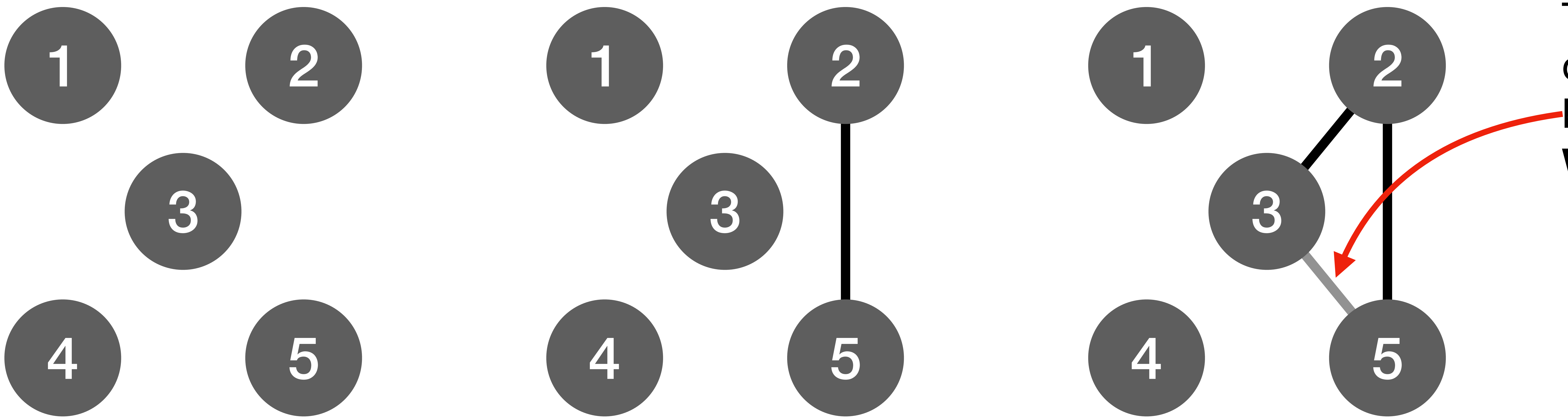
{{1}, {2}, {3}, {4}, {5}}
Initial State



{{1}, {2, 5}, {3}, {4}}
Insert Edge 2,5

DYNAMIC STREAMING CONNECTED COMPONENTS

Goal: Find connected components of a graph with n nodes subject to stream of edge insertions and deletions.



{{1}, {2}, {3}, {4}, {5}}
Initial State

{{1}, {2, 5}, {3}, {4}}
Insert Edge 2,5

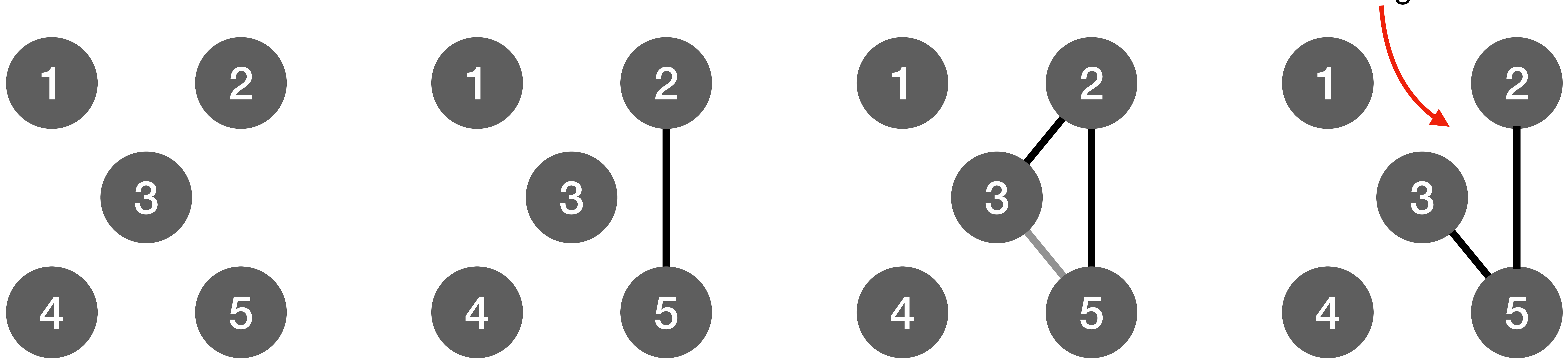
{{1}, {2, 3, 5}, {4}}
Insert Edges 2,3 3,5

This edge is redundant for defining the components but we still need to store it. **Why?**

DYNAMIC STREAMING CONNECTED COMPONENTS

Goal: Find connected components of a graph with n nodes subject to stream of edge insertions and deletions.

To return the correct answer we need to retain redundant edges in case other edges are deleted



{{1}, {2}, {3}, {4}, {5}}
Initial State

{{1}, {2, 5}, {3}, {4}}
Insert Edge 2,5

{{1}, {2, 3, 5}, {4}}
Insert Edges 2,3 3,5

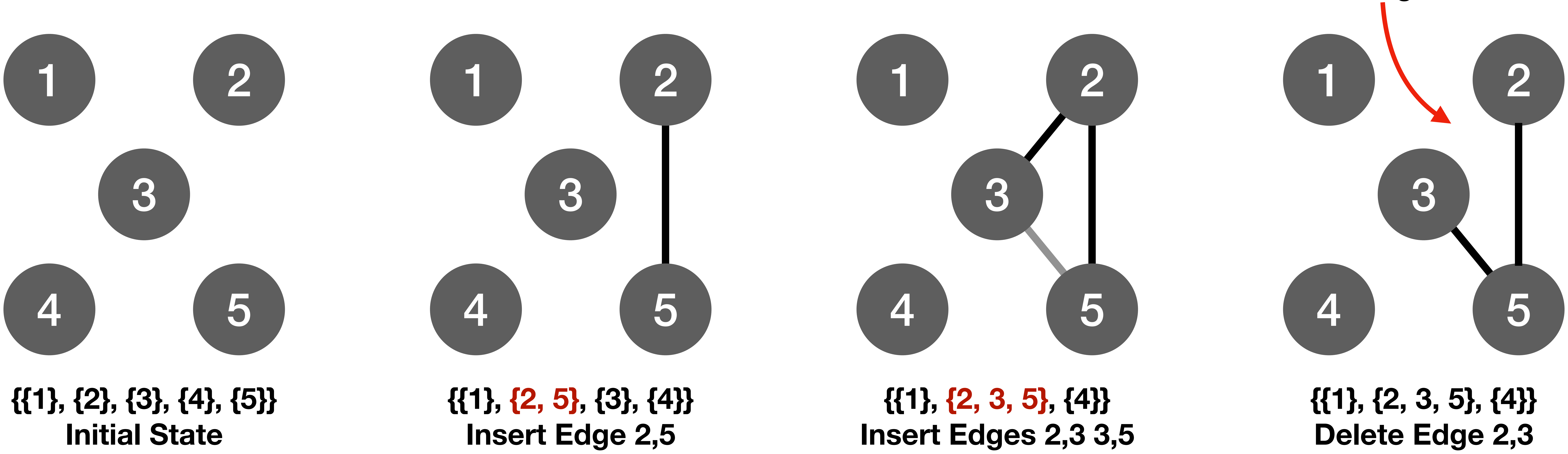
{{1}, {2, 3, 5}, {4}}
Delete Edge 2,3

DYNAMIC STREAMING CONNECTED COMPONENTS

Goal: Find connected components of a graph with n nodes subject to stream of edge insertions and deletions.

Semi-Streaming constraint: $O(n \times \text{polylog}(n))$ space

To return the correct answer we need to retain redundant edges in case other edges are deleted

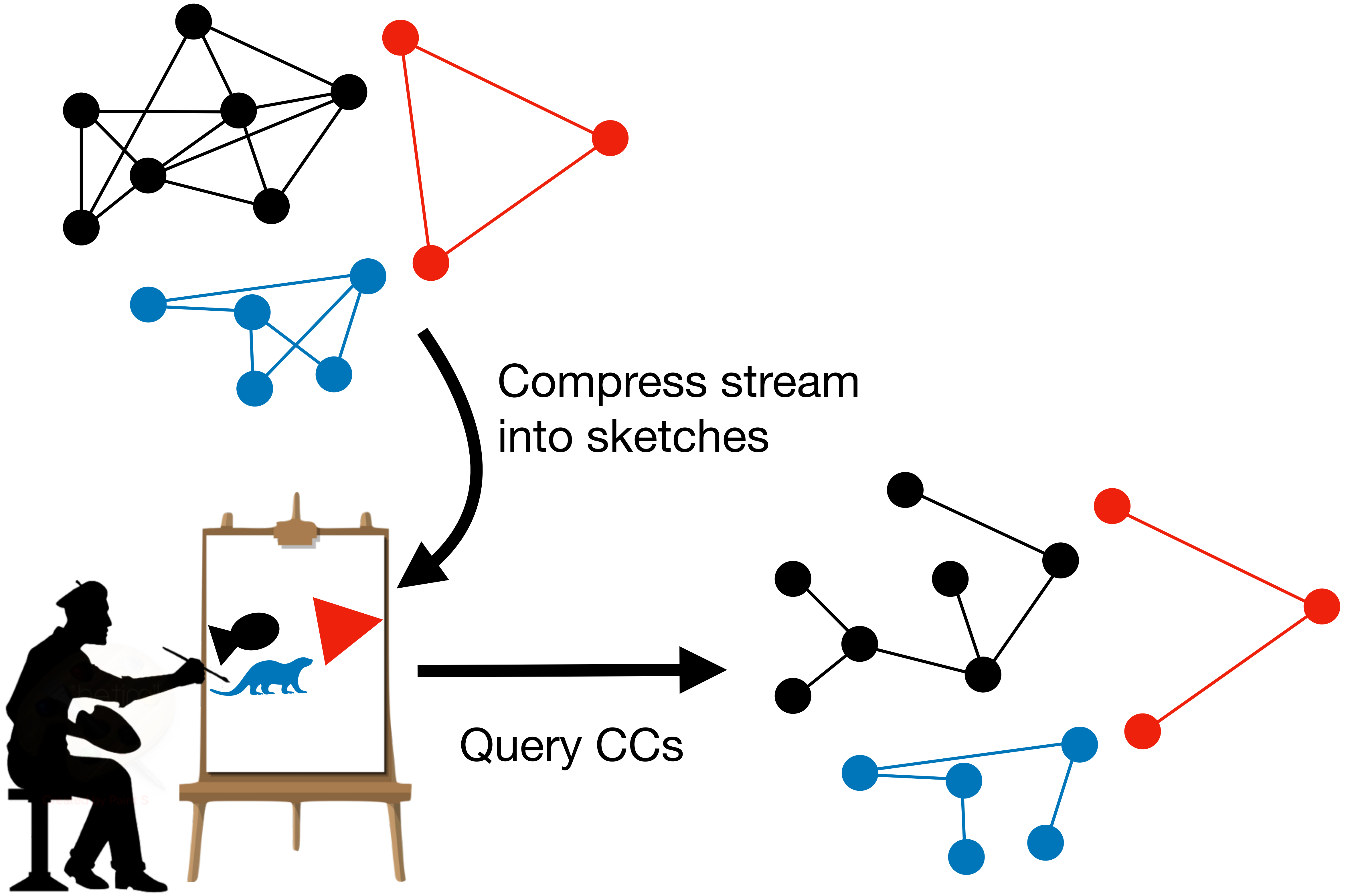


PROCESSING ENORMOUS, CHANGING GRAPHS WITH LINEAR-SKETCHING MADE USEFUL VIA ALGORITHMIC IMPROVEMENTS AND EXTERNAL MEMORY DATA-STRUCTURES

Hi David! I'd like to do research and use my coding skills

A group of us are implementing **Ahn, Guha, and McGregor's algorithm [SODA12]** for the dynamic streaming connected components problem.
It should be an easy publication …

AHN, GUHA, & MCGREGOR'S ALGORITHM: CONNECTIVITY IN SMALL SPACE



Even though edge insertion/deletion updates are received one by one in stream order.

Compresses graph stream via **linear sketching** to a size of only $O(n \log^3 n)$. The graph may be much larger than this, but the algorithm can still recover connected components **w.h.p.**

[Ahn, Guha, McGregor SODA 2012]

IMPLEMENTATION TIME!


TO BOLDLY GO . . .


TO BOLDLY IMPLEMENT . . .


It’s too big!
Back of the envelope math for graph on 1 billion nodes:
$10^9 \log^3(10^9) = 2.7 \times 10^{13}$
Before constants, roughly **25 TB** of RAM.
It’s too slow!
BOLDLY IMPLEMENTED . . . POORLY?

MAKE SKETCHES SMALLER?

Let's improve the asymptotic space cost!

MAKE SKETCHES SMALLER?

Let's improve the asymptotic space cost!

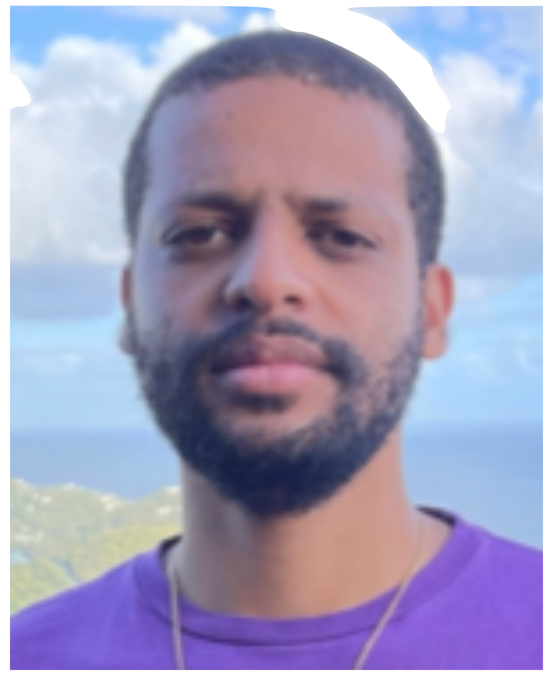
Great idea Evan, I always knew you'd finally contribute something to this project.

MAKE SKETCHES SMALLER?

Let's improve the asymptotic space cost!

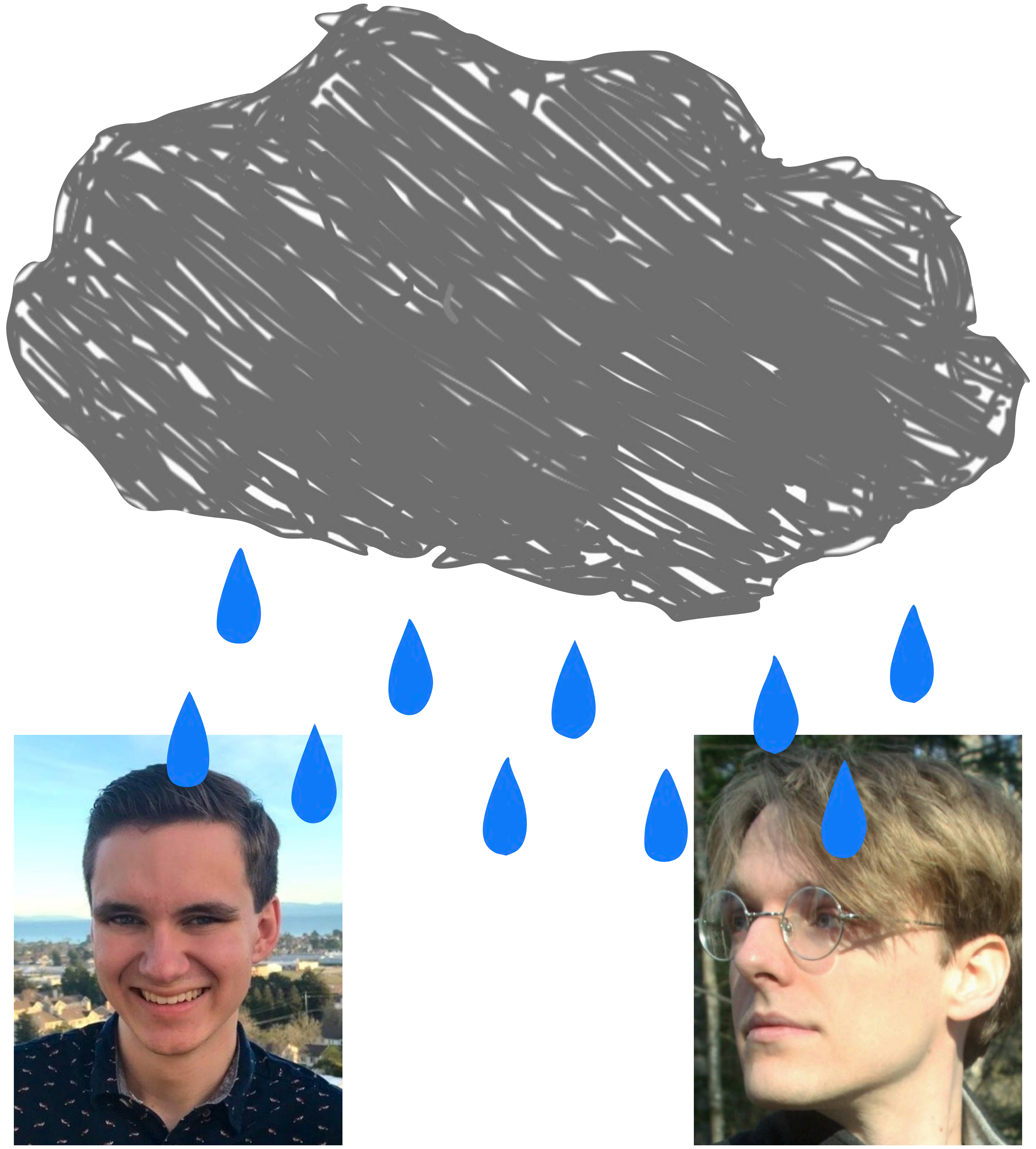
<u>You cannot</u>: Lower bound: $\Omega(n \log^3 n)$

[Nelson & Yu, SODA 2019]



CANNOT MAKE SKETCHES SMALLER

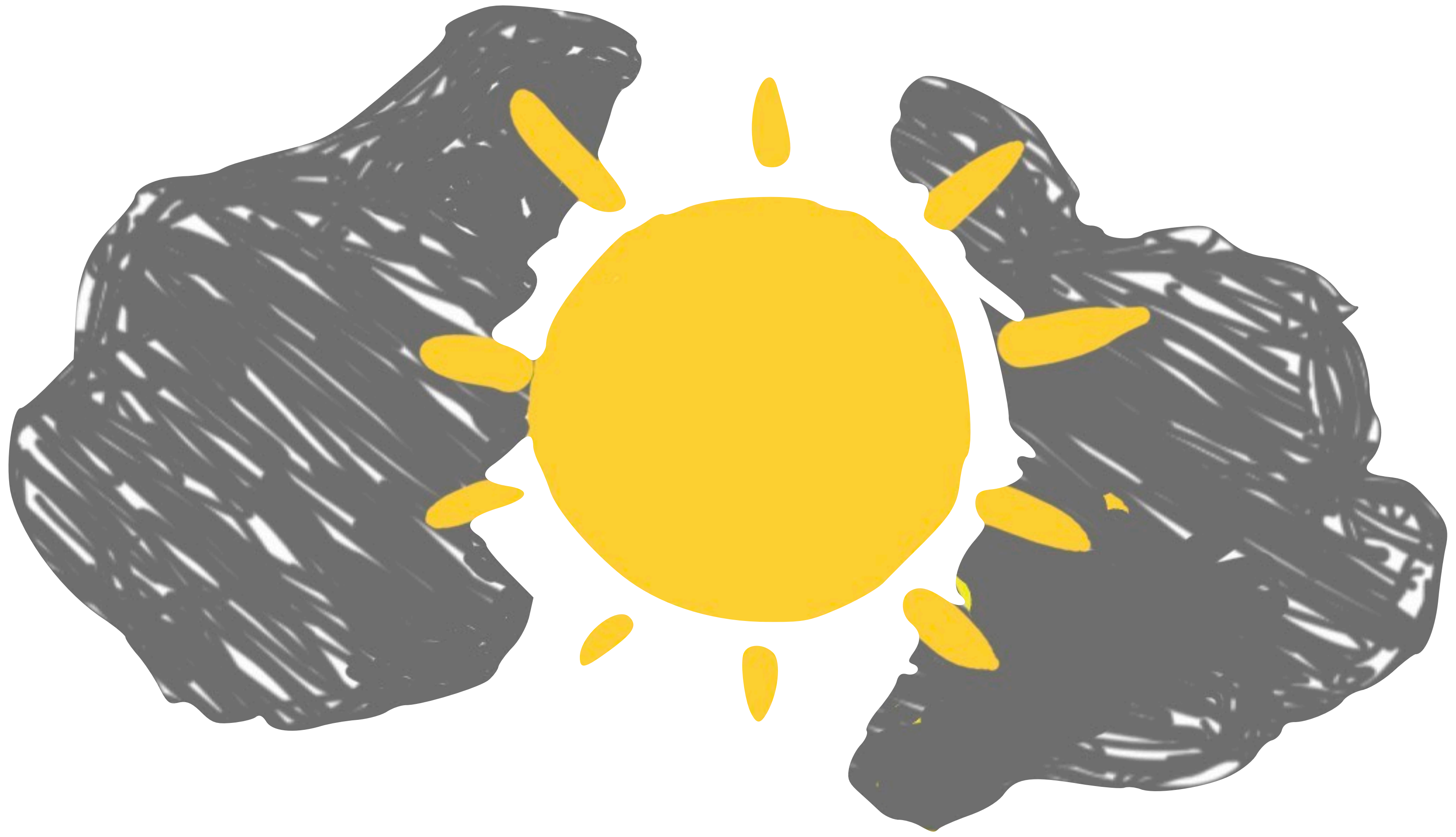
<u>You cannot</u>: Lower bound: $\Omega(n \log^3 n)$
[Nelson & Yu, SODA 2019]



For our purposes:
This might as well be infinite RAM

How can we overcome this lower bound?

CANNOT MAKE SKETCHES SMALLER: MAKE THEM USEFUL ANYWAY



GraphZeppelin:
We built a system that solves dynamic streaming connected components for a critical use case.

To do this we designed an algorithm which works well **despite the space lower bound.**

PROCESSING ENORMOUS, CHANGING GRAPHS WITH LINEAR-SKETCHING MADE USEFUL VIA ALGORITHMIC IMPROVEMENTS AND EXTERNAL MEMORY DATA-STRUCTURES

WHAT MAKES A STREAMING ALGORITHM "USEFUL"?

1. Can be run on
today's hardware.

How can we overcome the
space lower bound?

WHAT MAKES A STREAMING ALGORITHM "USEFUL"?

1. Can be run on **today's hardware.**

How can we overcome the space lower bound?

2. Keeps pace with **high-speed streams.**

How can we achieve our other goals while remaining fast?

WHAT MAKES A STREAMING ALGORITHM "USEFUL"?

1. Can be run on **today's hardware.**

How can we overcome the space lower bound?

2. Keeps pace with **high-speed streams.**

How can we achieve our other goals while remaining fast?

3. Meets an **unmet need**

When are existing systems for processing graph streams unable to find CCs?

WHAT MAKES A STREAMING ALGORITHM "USEFUL"?

1. Can be run on **today's hardware.**

How can we overcome the space lower bound?

SPACE

2. Keeps pace with **high-speed streams.**

How can we achieve our other goals while remaining fast?

SPEED

3. Meets an **unmet need**

When are existing systems for processing graph streams unable to find CCs?

DENSITY

UNMET NEED: PROCESSING DENSE GRAPHS

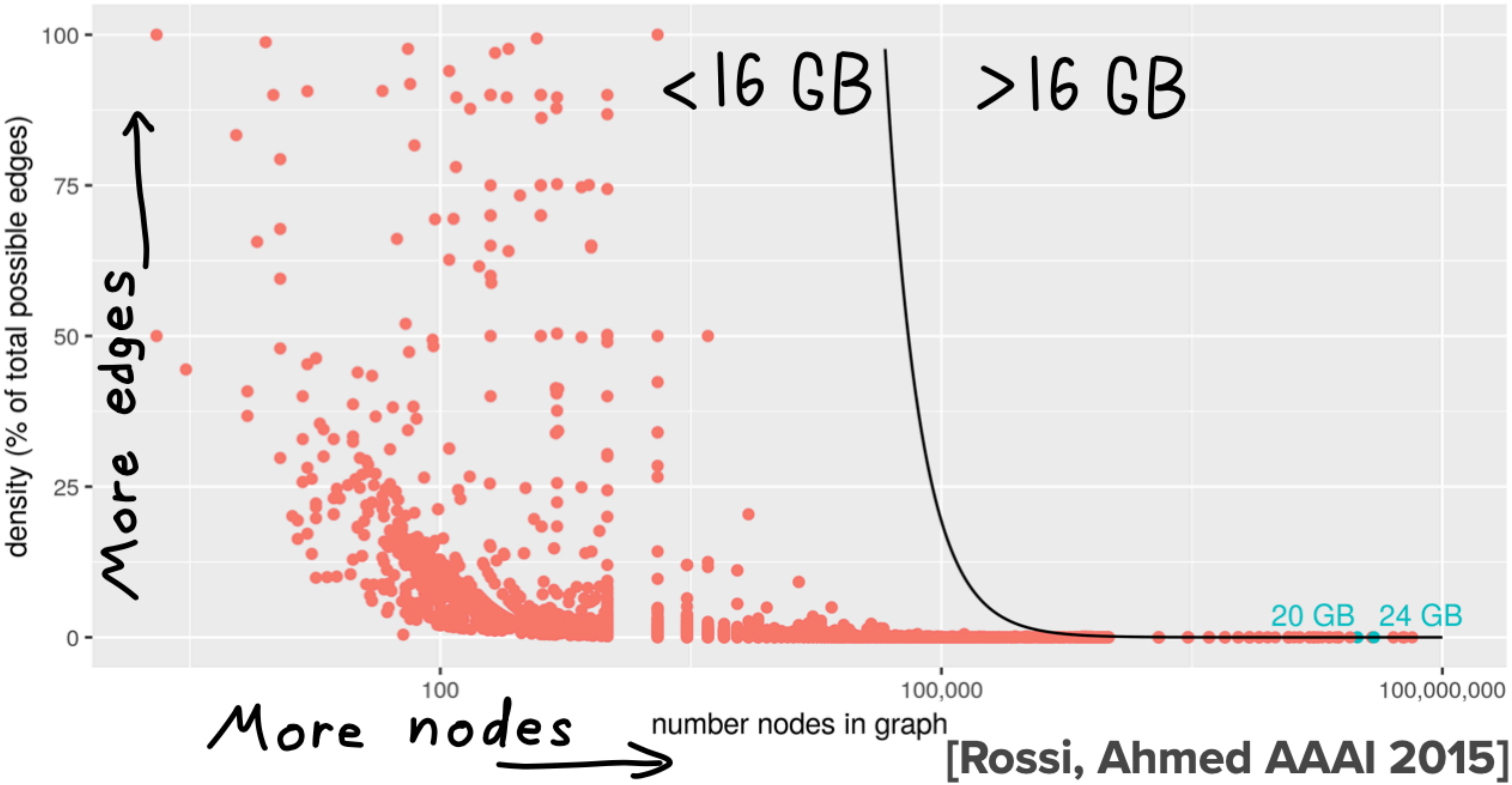
Memory usage of semi-streaming algorithms scales with n the number of nodes, **but not the number of edges E.**

Get the greatest gains when E is large, i.e., **graph is dense.**

Folk wisdom: "**Large dense graphs don't exist** in practice. Real-world graphs are **sparse**."

Other dynamic graph processing systems optimize for sparse graphs.
Aspen [Dhulipala, Blelloch, Shun 2019]
Terrace [Pandey, Wheatman, Xu, Buluç 2021]

UNMET NEED: PROCESSING DENSE GRAPHS



[Rossi, Ahmed AAAI 2015]

UNMET NEED: PROCESSING DENSE GRAPHS

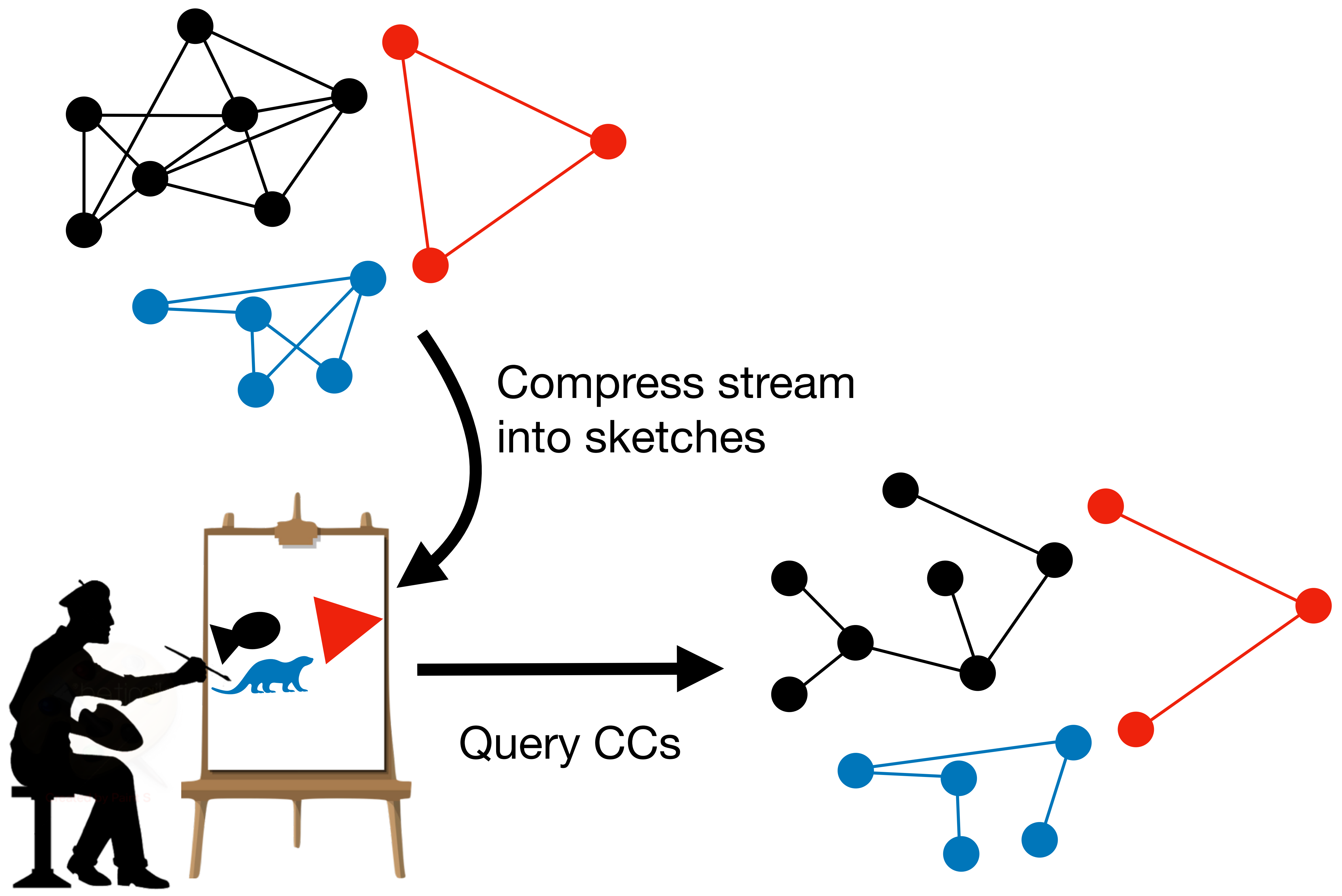
Facebook works with large, dense graphs (40 million nodes and larger) since at least 2015.
They do so at great cost on supercomputing clusters.
[Ching, Edunov, Kabiljo, Logothetic, Muthukrishnan VLDB 2015]

The folk wisdom is in fact observing a **selection effect**
We lack the tools to process large, dense graph streams so they are rarely studied.

DENSITY

PROCESSING ENORMOUS, CHANGING GRAPHS WITH LINEAR-SKETCHING MADE USEFUL VIA ALGORITHMIC IMPROVEMENTS AND EXTERNAL MEMORY DATA-STRUCTURES

SKETCHES ARE TOO LARGE FOR MODERN HARDWARE: OR ARE THEY?

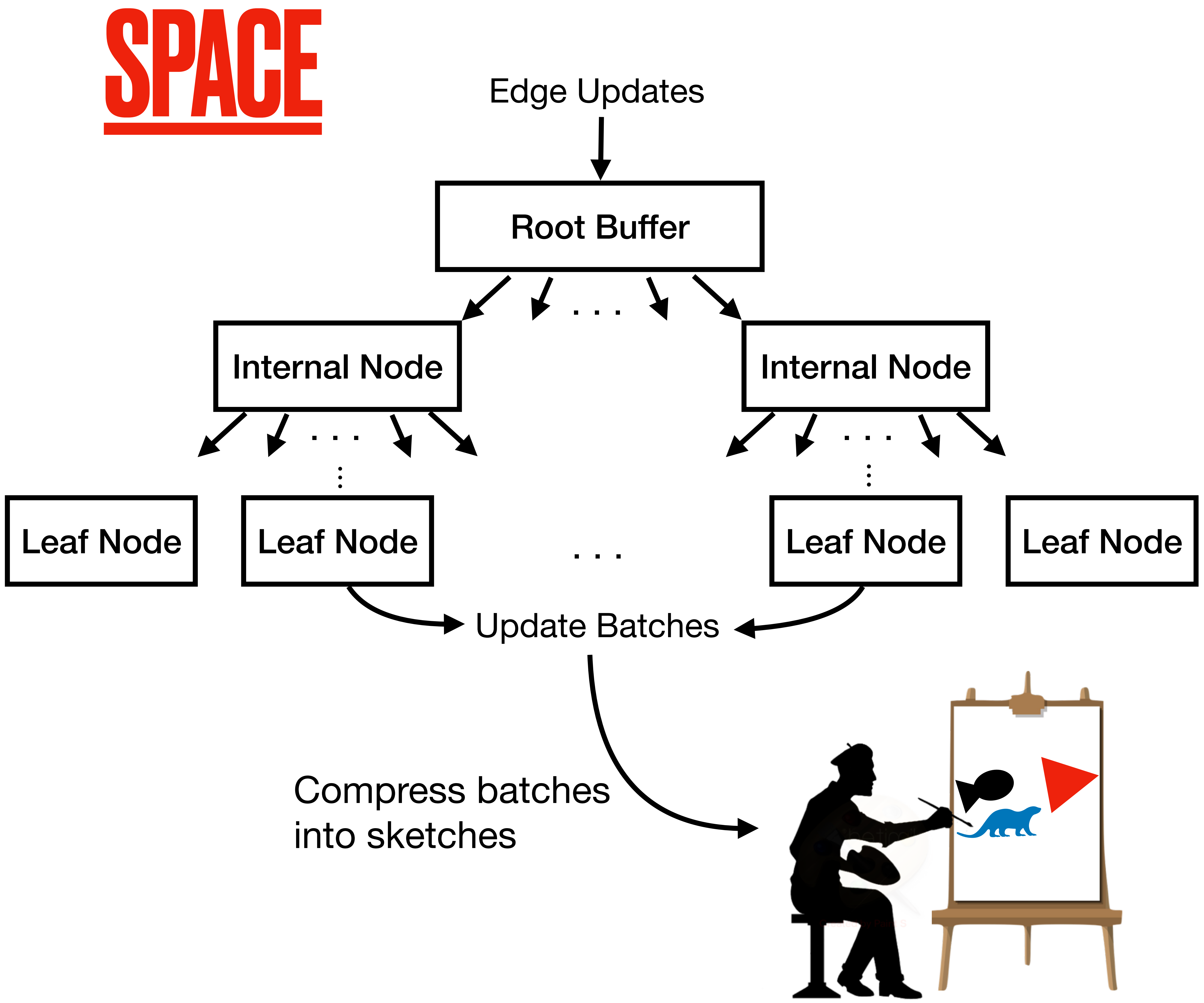


The problem:

- $O(n \log^3 n)$ space is big actually. On top of that sketches have huge constants. Easily overflows RAM.

- Streams and sketches are random so data out-of-core is EXTREMELY slow

SKETCHES ARE TOO LARGE FOR MODERN HARDWARE: OR ARE THEY?



The problem:

- $O(n \log^3 n)$ space is big actually. On top of that sketches have huge constants. Easily overflows RAM.
- Streams and sketches are random, out-of-core is EXTREMELY slow

Solve the space problem:

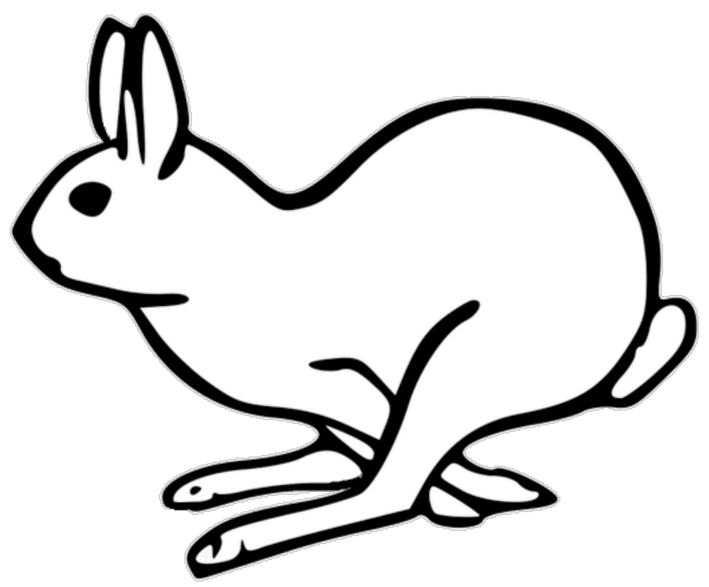
- GraphZeppelin uses a GutterTree to efficiently buffer updates on disk.
- Buffering updates **amortizes the cost of accessing the disk.**
- Still a **space optimal** connected components

EXTERNAL MEMORY ALGORITHMS

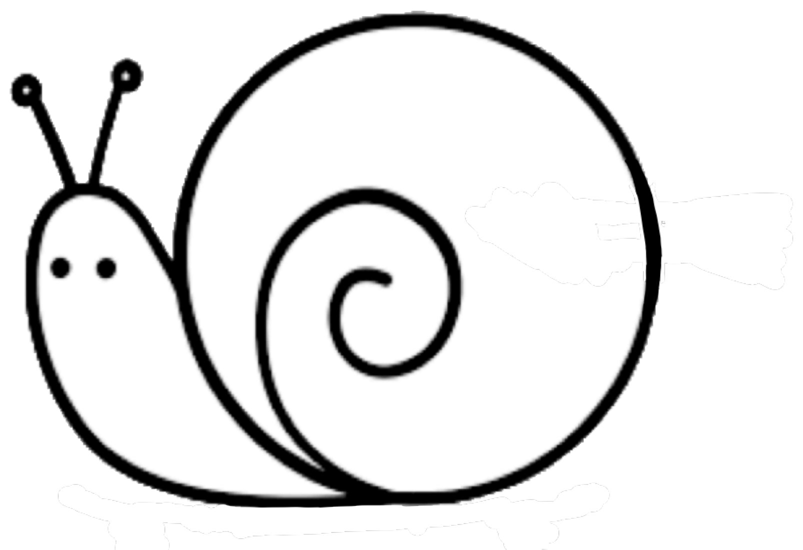
Fetching a block from disk into RAM is expensive. Latency of disk is much higher than that of RAM.

External memory algorithms: basic idea is to delay accessing the disk until we have a bunch of operations that touch the same block.

Therefore, latency of disk is shared (amortized) among all these operations



RAM



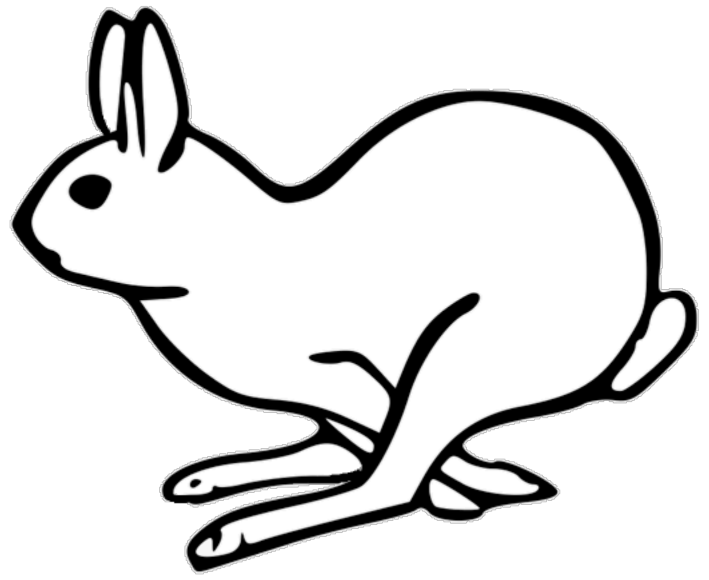
SSD with standard algorithms

EXTERNAL MEMORY ALGORITHMS

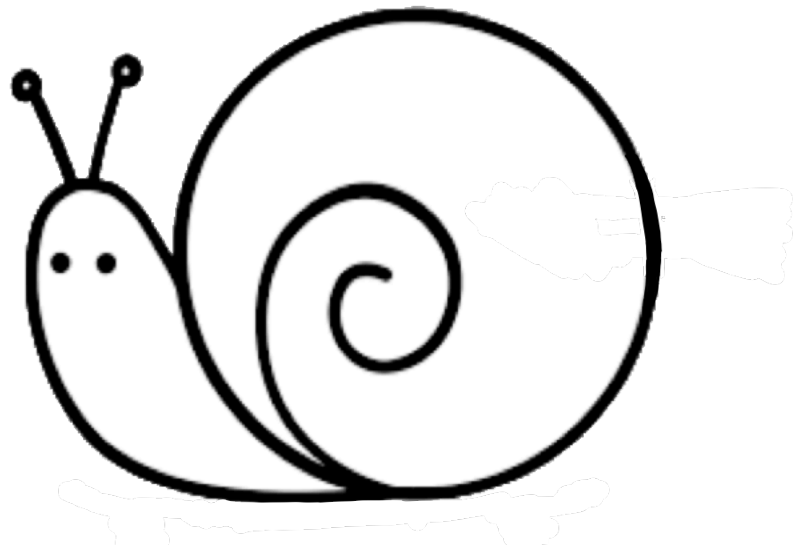
Fetching a block from disk into RAM is expensive. Latency of disk is much higher than that of RAM.

External memory algorithms: basic idea is to delay accessing the disk until we have a bunch of operations that touch the same block.

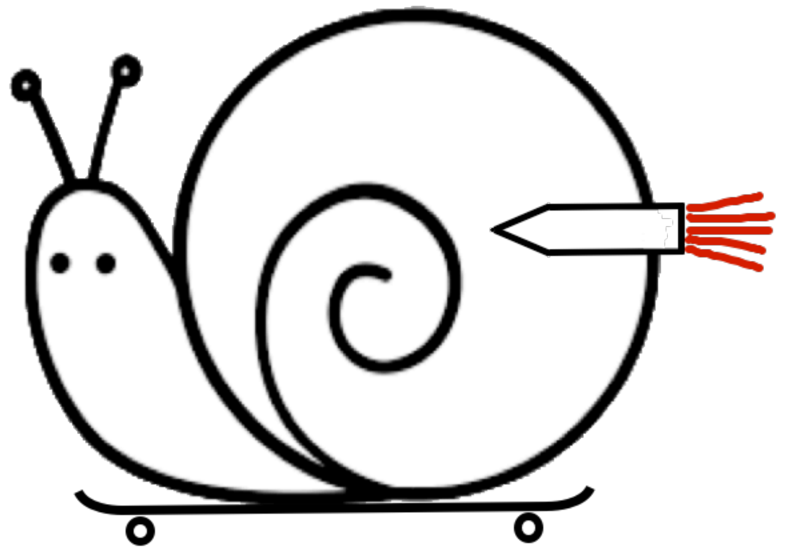
Therefore, latency of disk is shared (amortized) among all these operations



RAM

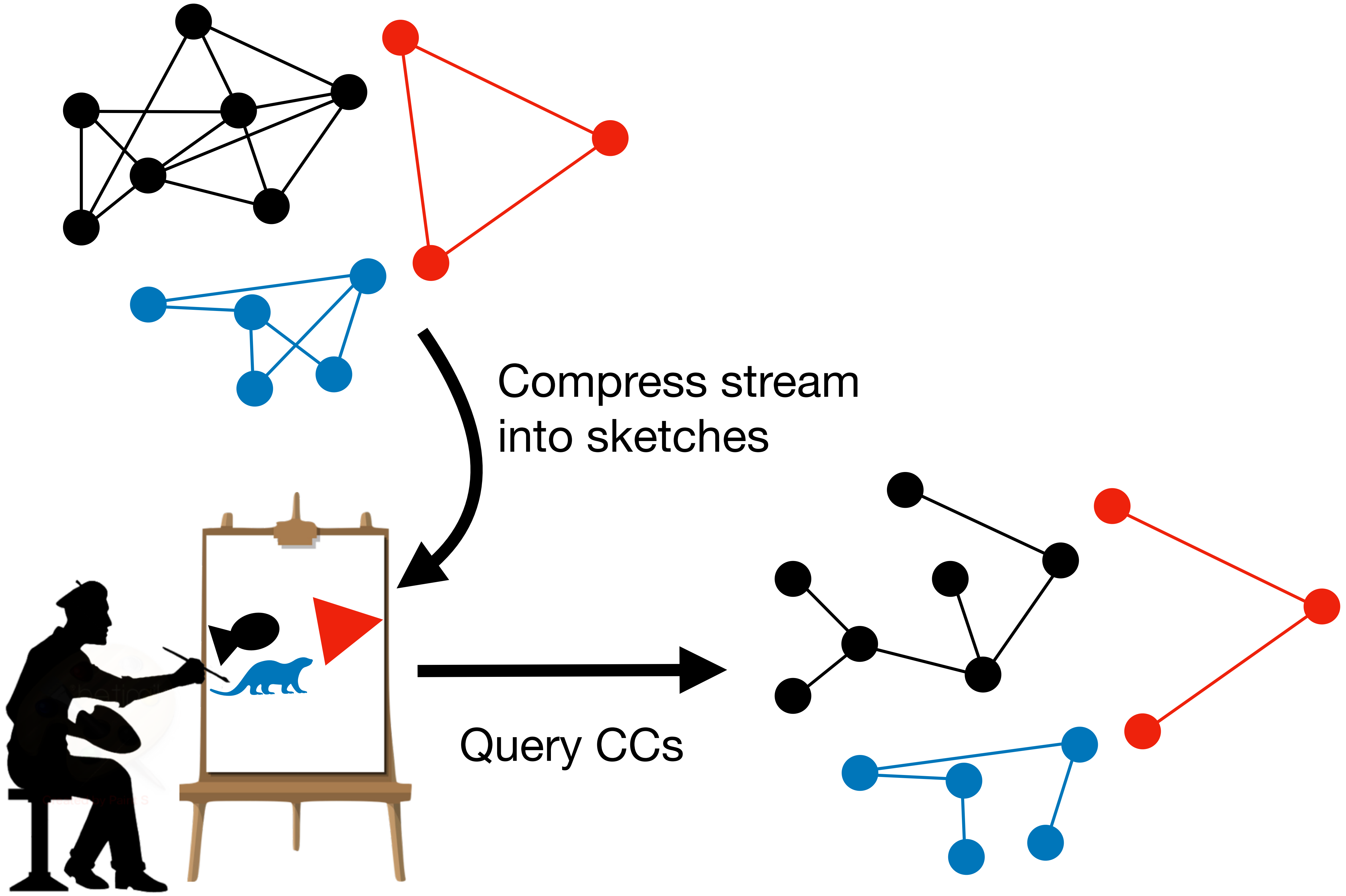


SSD with standard algorithms



SSD with external memory algorithms

SKETCHES ARE TOO SLOW FOR MODERN HARDWARE

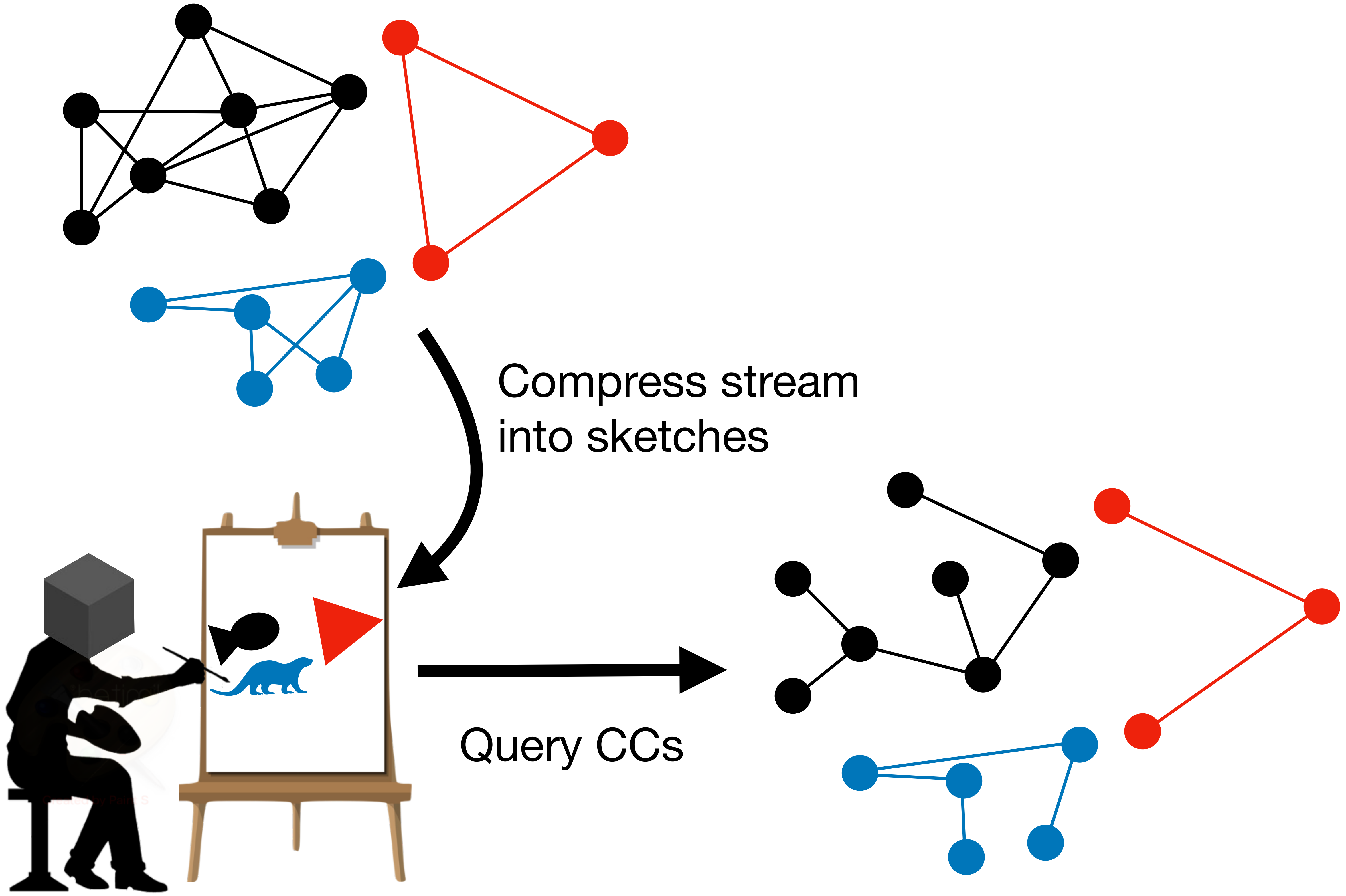


The problem:

- Linear-sketching algorithm by AGM uses modular exponentiation
- Also requires 128 bit integers
- This is very, very slow
  - For a graph on 10^6 nodes update rate is only 833 per second

SKETCHES ARE TOO SLOW FOR MODERN HARDWARE: IMPROVE THEM!

SPEED



The problem:

- Linear-sketching algorithm by AGM uses modular exponentiation
- Also requires 128 bit integers
- This is very, very slow
  - For a graph on 10^6 nodes update rate is only 833 per second

CubeSketch:

- Linear-sketching algorithm for connected components
- Updates $> 10^3$ times faster
- Uses 8 times less space

GRAPHZEPPELIN: C++ LINEAR-SKETCHING LIBRARY

GRAPHZEPPELIN: AVOIDING DATA EXPLOSION IN GRAPH STREAMS

Graf-Zeppelin, **NOT** the Hindenburg, did not explode

GraphZeppelin: Solves streaming connected components using CubeSketch.

Fast:

- 3-5 million updates/sec in RAM
- >2.5 million updates/sec on consumer SSD

Compact:

- Compresses >200 GB stream into 45 GB sketch (2^{18} node graph)

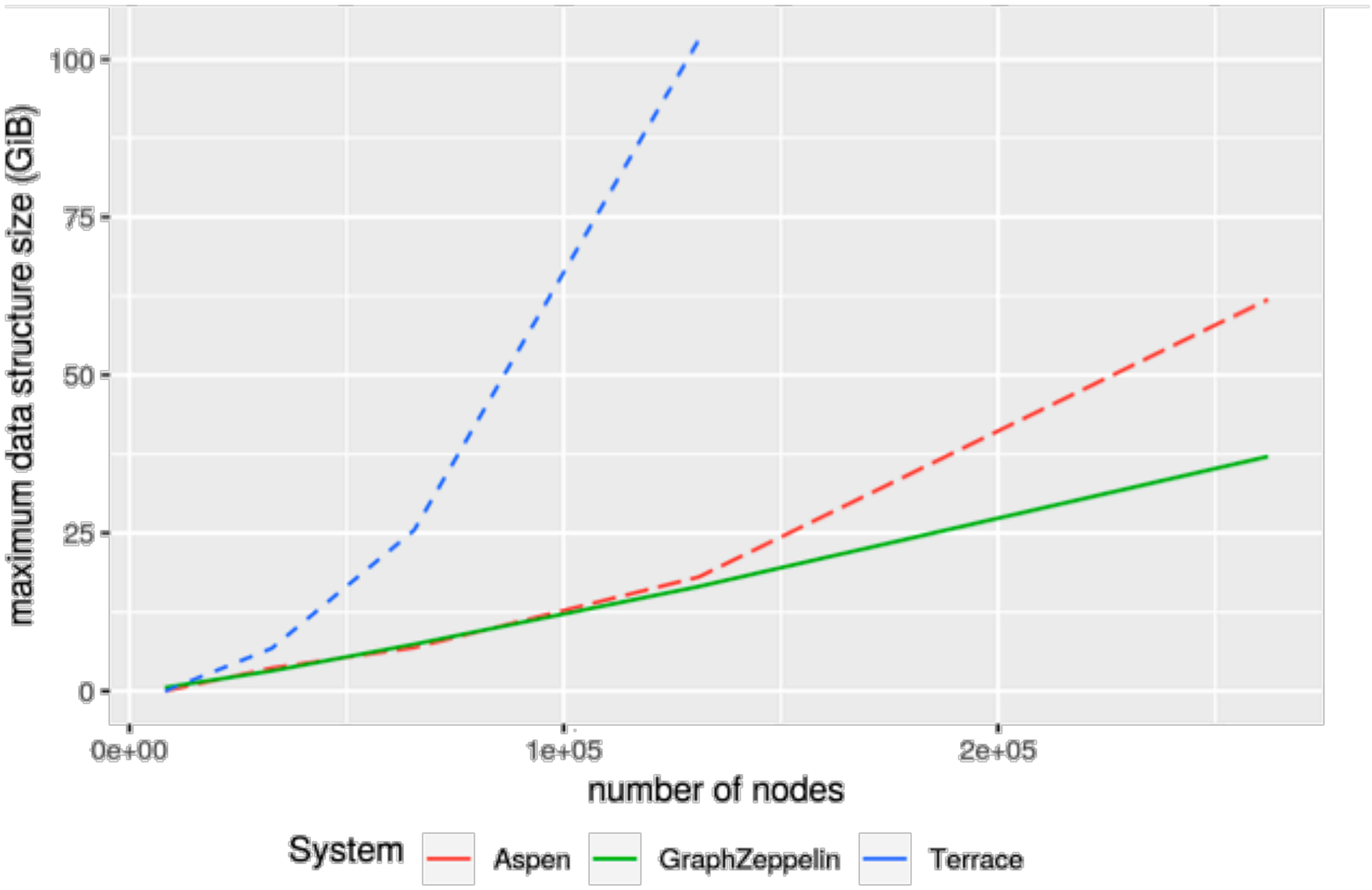
SMALLER ON DENSE GRAPHS

State-of-the-art:
Aspen **[Dhulipala, Blelloch, Shun 2019]**
Terrace **[Pandey, Wheatman, Xu, Buluç 2021]**

More compact:

- Aspen is 2x larger than GraphZeppelin
- Terrace is 10x larger than GraphZeppelin

Trend will continue - GraphZeppelin is asymptotically smaller: $O(n \log^3 n)$ vs $O(n^2)$



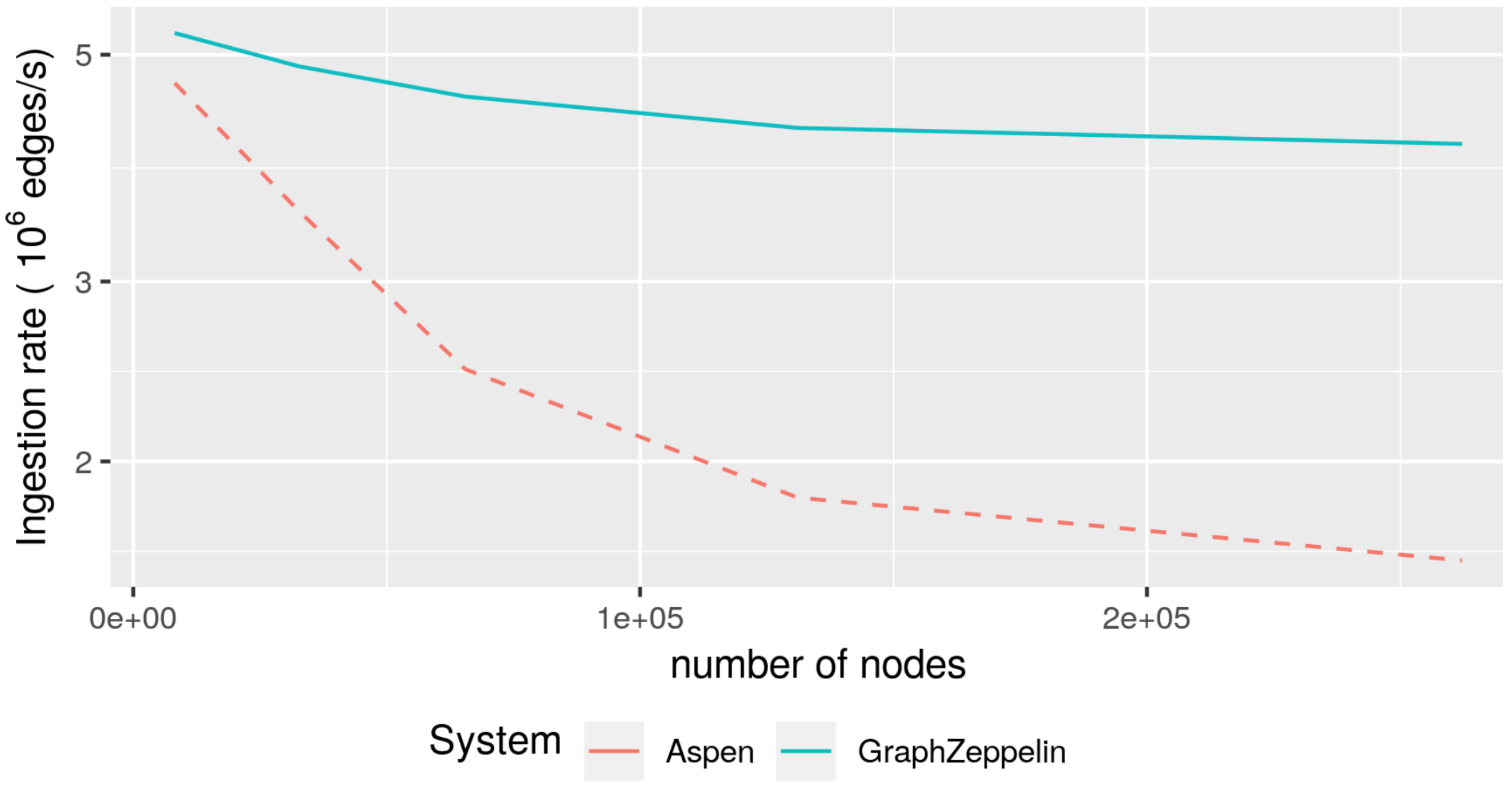
FASTER ON DENSE GRAPHS

Faster:

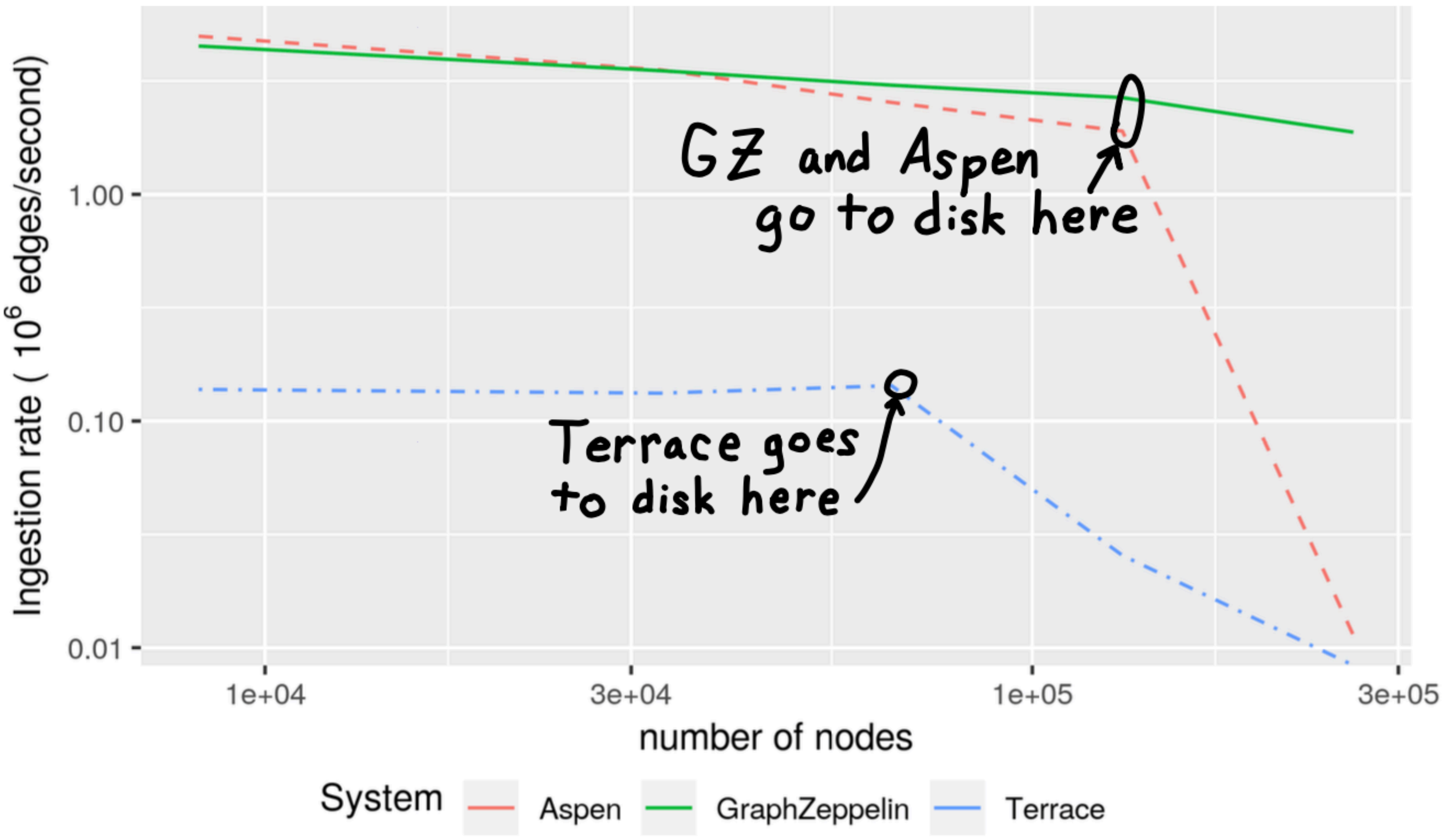
On dense graphs* in RAM,

- GraphZeppelin is 2x faster than Aspen
- GraphZeppelin is 30x faster than Terrace

*Aspen and Terrace are very fast on sparse graphs in RAM (10-50 million edges/sec)



FAST ON SSD

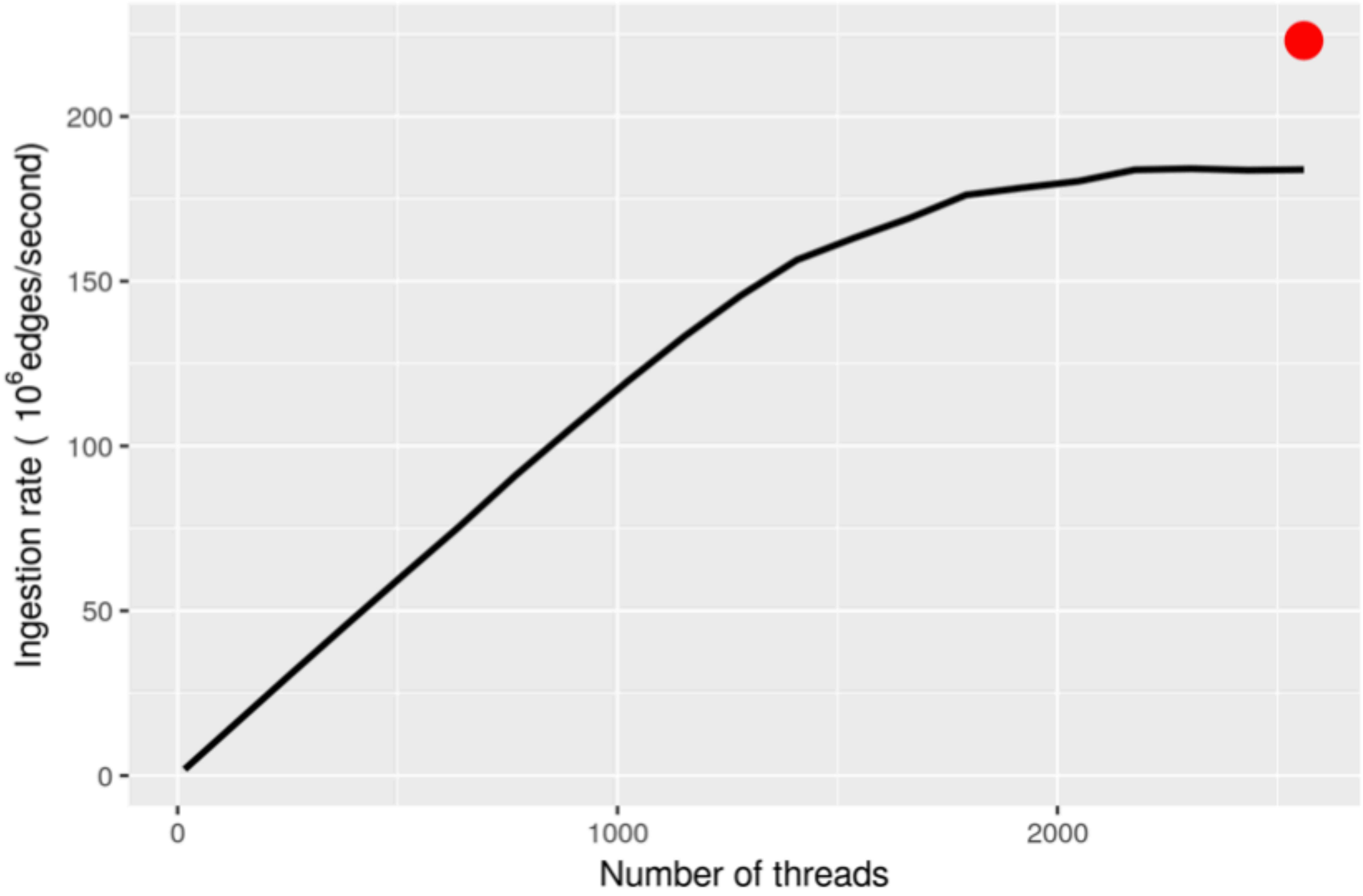

Ingestion rate (10^6 edges/second)
1.00
0.10
0.01
1e+04
3e+04
1e+05
3e+05
number of nodes
GZ and Aspen go to disk here
Terrace goes to disk here
System
Aspen
GraphZeppelin
Terrace

QUESTIONS? PLEASE ASK!

MORE DETAILS . . .

DISTRIBUTED SKETCHING

- Overcome GraphZeppelin's CPU bottleneck by distributing update work
- Sketching lets us avoid communication bottleneck of distributed graph systems
- Scale near-linearly until system bottlenecked by sequential RAM bandwidth



FUTURE WORK

- **Improve query performance**

  Current query times are comparable to Aspen/Terrace, but likely can be improved with a better algorithm.

- **Support more graph algorithms**

  E.g., k-connectivity, correlation clustering, triangle counting, spectral sparsification, minimum cut.