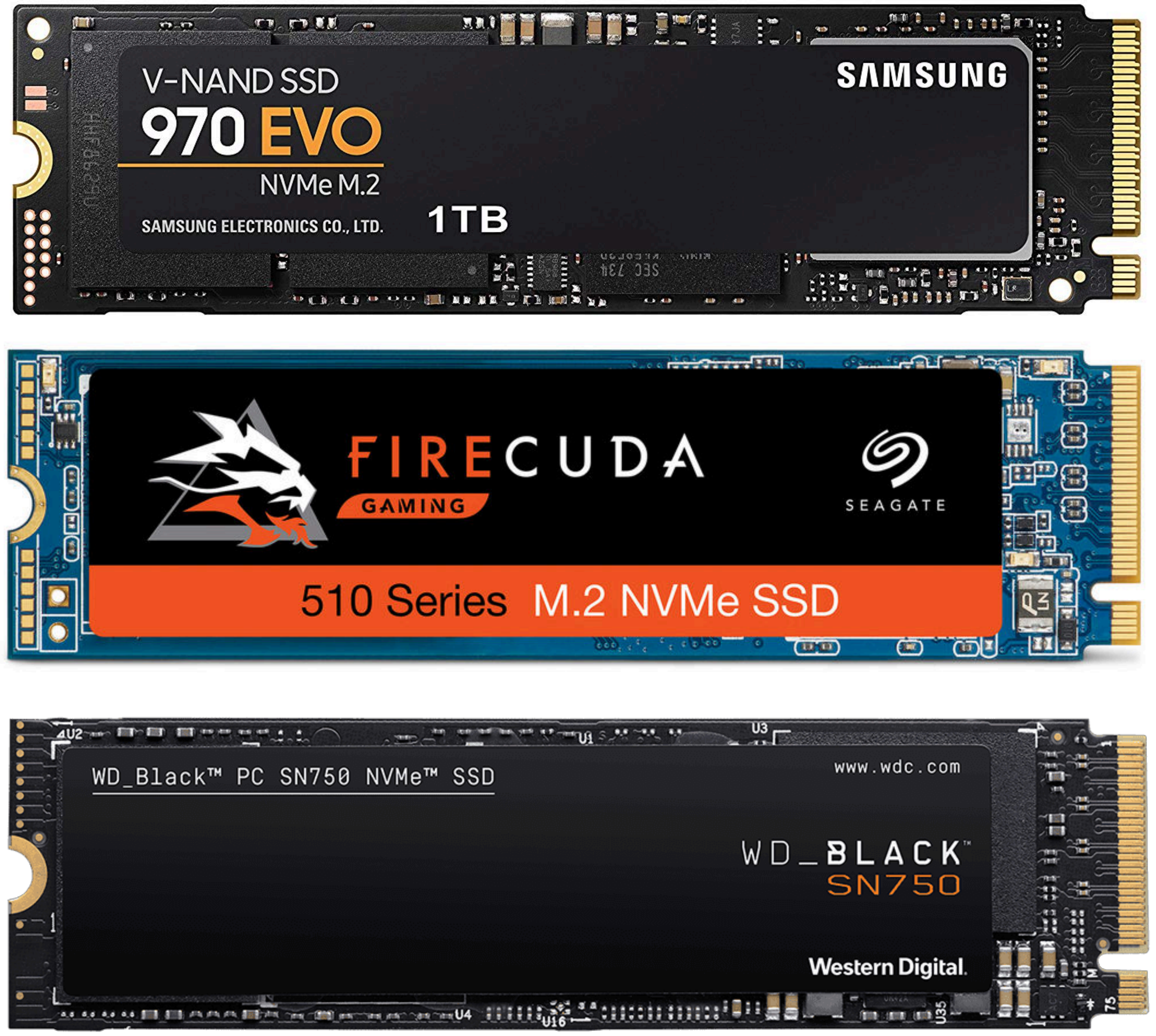
Read as Needed:
Building WiSER, a Flash-Optimized Search Engine

Jun He[1], Kan Wu, Sudarsun Kannan[#],
Andrea C. Arpaci-Dusseau, Remzi H. Arpaci-Dusseau

Department of Computer Sciences, University of Wisconsin - Madison
Department of Computer Science, Rutgers University

[1]*Jun He is now at Google.*

SSDs are Fast



...

- Sequential read: 3.5GB/s
- Random read: 500,000 IOPS
- 0.17 dollar/GB

Not an endorsement for any brand

Many applications/systems have been optimized for SSDs

Key-value stores

- RocksDb, Wisckey, …

Graph stores

- FlashGraph, Mosaic, …

File systems

- SFS, F2FS, …

…

Search engines are overlooked

Search engines are important and widely used

DB-Engines Ranking

Feb 2020	DBMS
1.	Oracle
2.	MySQL
3.	Microsoft SQL Server
4.	PostgreSQL
5.	MongoDB
6.	IBM Db2
7.	Elasticsearch
8.	Redis
9.	Microsoft Access
10.	SQLite

- Wikipedia
- Github
- Uber
- ...

Search engines are challenging for storage systems

Low data latency

- queries are interactive

High data throughput

- engines retrieve info from a large amount of data

High scalability

- data grows over time

Just use more RAM?

RAM is critical in existing search engines

- "your RAM will limit all other resources"

— an Elasticsearch user

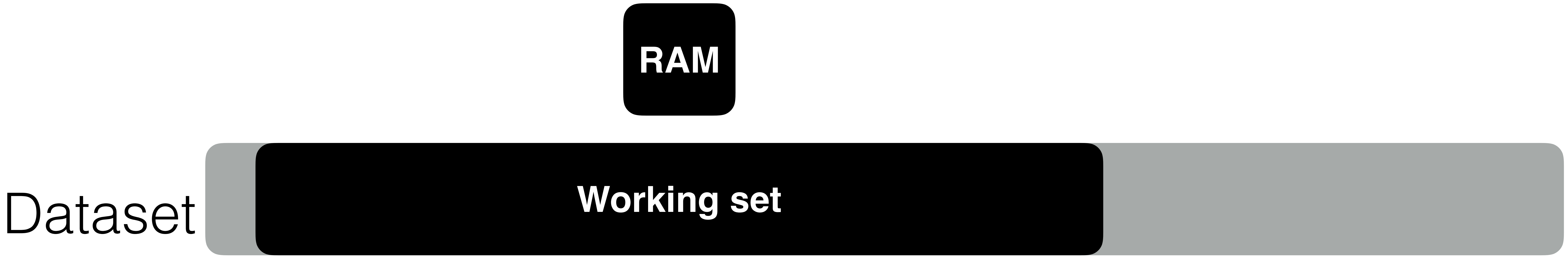
OK at small scale

- total cost is low

Cost prohibitive at large scale

- data grows fast
- may waste bandwidth: rarely read and process 100GB/s

Can search engines perform well with a small memory and a fast SSD?



Yes

Built an engine that performs well on small memory

- could outperform other engine with entire dataset in memory

Our philosophy: read as needed

- use small memory
- read data from SSDs as needed
- do not attempt to cache data in memory
- attempt to read data from SSDs efficiently

SSDs have high read bandwidth and IOPS

- read an entire drive (1TB) in 5 minutes
- higher bandwidth is possible with RAID

Computation/network is more likely to be the bottleneck

To optimize a read-as-needed system, we must:

Reduce read amplification

- SSD bandwidth is still limited

Hide I/O latency

- SSD latency is still high

Use large requests to improve device efficiency

- SSDs favor large requests

We apply four techniques to build a read-as-needed search engine

1. **Grouping data by term**
   - reduce read amplification
   - use large requests
2. **Two-way Cost-aware Bloom Filters**
   - reduce read amplification
3. **Adaptive Prefetching**
   - hide I/O latency
4. **Trade Disk Space for I/O**
   - reduce read amplificationm

Our Contributions

Four techniques to improve search engine performance

- small memory, large dataset, fast SSDs

An open-source flash-optimized search engine from scratch (WiSER)

- 11,000 lines of C++

A benchmark for evaluating search engines

- based on Wikipedia
- a variety of queries for stressing the system

Better performance than the state-of-the-art Elasticsearch

- increase end-to-end query throughput by up to 2.7x
- reduce end-to-end latency by up to 16x

Outline

Overview

Techniques

Evaluation

Final Thoughts

Techniques

1. Cross-stage Data Grouping
2. Two-way Cost-aware Bloom Filters
3. Adaptive Prefetching
4. Trade Disk Space for I/O

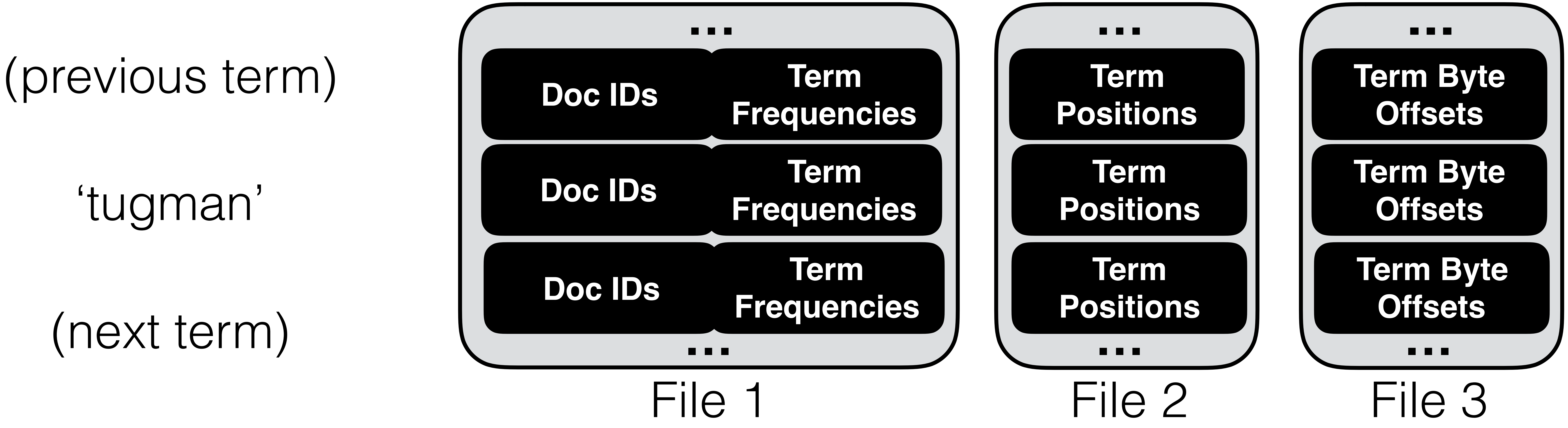
Guided by the Wikipedia dataset

Techniques

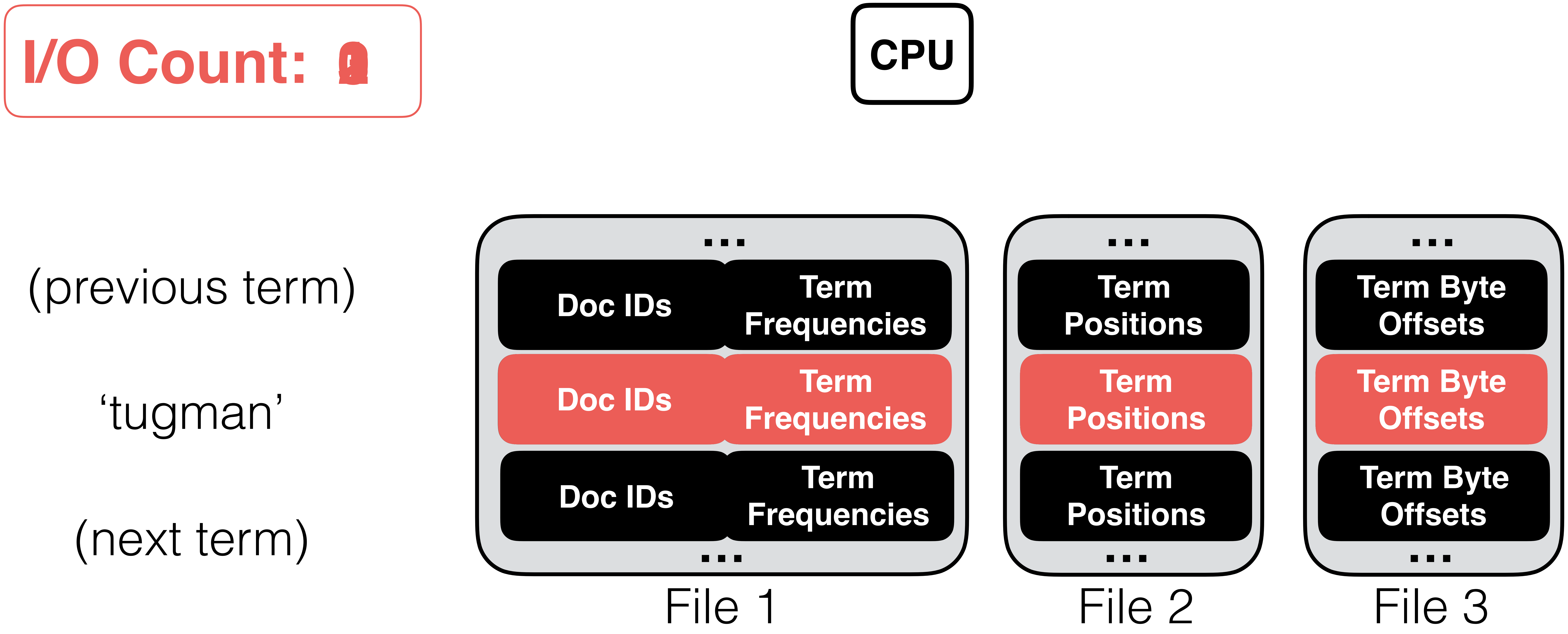
1. **Cross-stage Data Grouping**
2. Two-way Cost-aware Bloom Filters
3. Adaptive Prefetching
4. Trade Disk Space for I/O

Background: stages of a search query

Data in Elasticsearch is grouped by stage



How much I/O is needed for ‘tugman’?



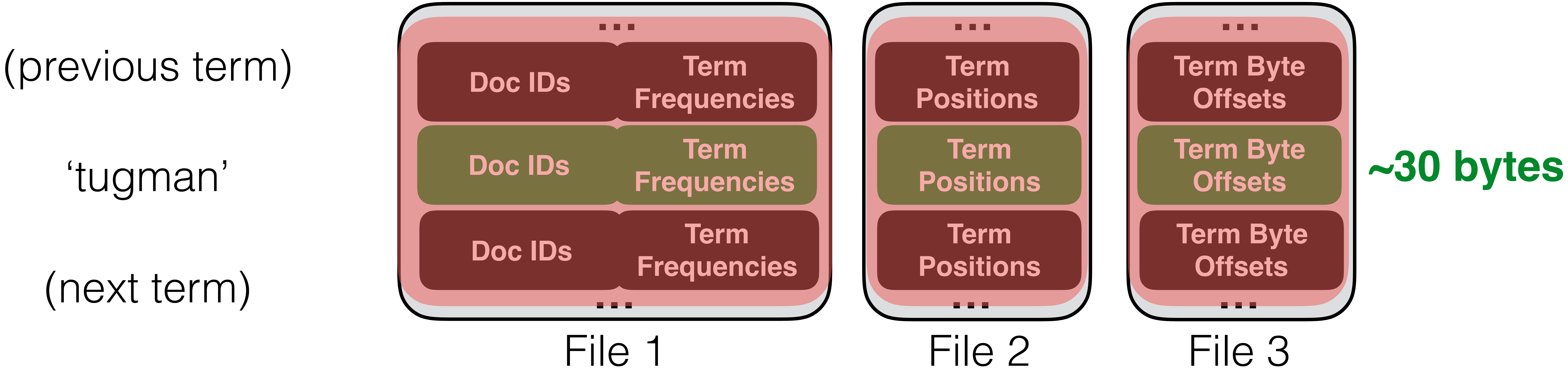
Transferred data is often wasted

I/O Count: 3, Size: 12 KB

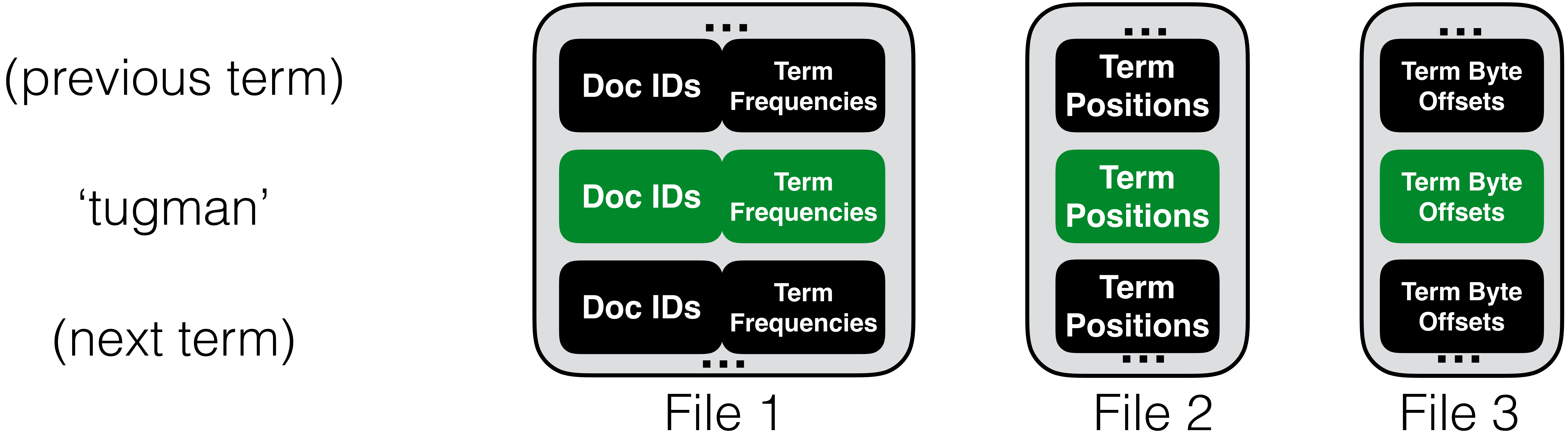
Data of ‘tugman’ can fit in 4KB

Small term is common

- 99% of Wikipedia terms can fit in 4 KB



WiSER groups data by term



WiSER groups data by term

How much I/O does WiSER need for ‘tugman’?

I/O Count: 0

Reduce I/O by up to 3x

CPU

Techniques

1. Cross-stage Data Grouping
2. **Two-way Cost-aware Bloom Filters**
3. Adaptive Prefetching
4. Trade Disk Space for I/O

Background: phrase queries offer high accuracy

Search “united AND states”

Search **phrase** “united states”

The **United States** president…

The spokesman of **united** airlines **states** that…

Background: phrase queries offer high accuracy

But demand a lot more data

Term	Position	Position
united	2,X,X,X,X,X,X…	4,X,X,X,X,X,X…
states	3,X,X,X,X,X,X…	6,X,X,X,X,X,X…

Our **United States** president…

The spokesman of **united** airlines **states** that…

Tried regular Bloom filters, it doesn't work…

WiSER adds two Bloom filters to enable "two-way" filtering

Term	bloom filter **before**	bloom filter **after**	**Position**
united	our	states	2
states	united	president	3
united	of	airline	4
states	airlines	that	6

Our **United States** president…

The spokesman of **united** airlines **states** that…

Check if "united states" is a phrase in a document by:

1. Checking if "states" is in "united".after
2. Or, checking if "united" is in "states".before.

Why adding two filters?

WiSER dynamically chooses the smaller filter for filtering (two-way)

WiSER dynamically chooses the smaller filter for filtering (two-way)

Check if 'states' is after 'united'

600 KB > 500 KB + 50 KB

	600 KB	600 KB	500 KB
united	Filters.before	Filters.after	Positions

	60KB	60KB	50KB
states	Filters before	Filters after	Positions

Check if 'united' is before 'states'

60 KB < 500 KB + 50 KB

WiSER only uses filters when it can reduce I/O (cost-aware)

WiSER only uses filters when it can reduce I/O (cost-aware)

Check if ‘states’ is after ‘united’

600 KB > 200 KB + 200 KB

Not worth it

Check if ‘united’ is before ‘states’

600 KB > 200 KB + 200 KB

Not worth it

Techniques

1. Cross-stage Data Grouping
2. Two-way Cost-aware Bloom Filters
3. **Adaptive Prefetching**
4. Trade Disk Space for I/O

Elasticsearch relies on the OS to prefetching…

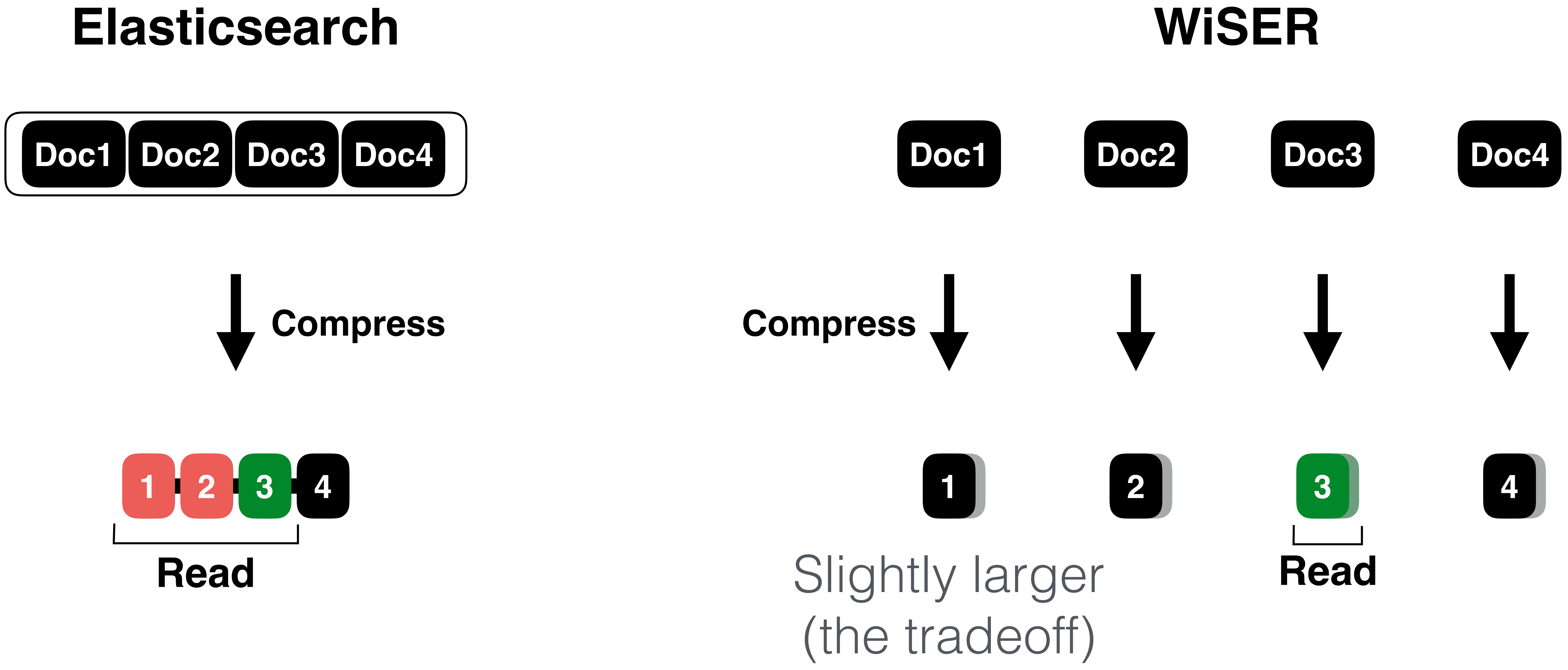
WiSER adaptively prefetches frequently-used data to hide I/O latency

WiSER adaptively prefetches frequently-used data to hide I/O latency

Techniques

1. Cross-stage Data Grouping
2. Two-way Cost-aware Bloom Filters
3. Adaptive Prefetching
4. **Trade Disk Space for I/O**

WiSER compresses documents individually to reduce read amplification



Techniques

1. Cross-stage Data Grouping
2. Two-way Cost-aware Bloom Filters
3. Adaptive Prefetching
4. Trade Disk Space for I/O

Outline

Overview

Techniques

Evaluation

Final Thoughts

Evaluate with *WSBench*

Dataset: Wikipedia

- 6 million documents, 6 million terms, 18GB

Queries

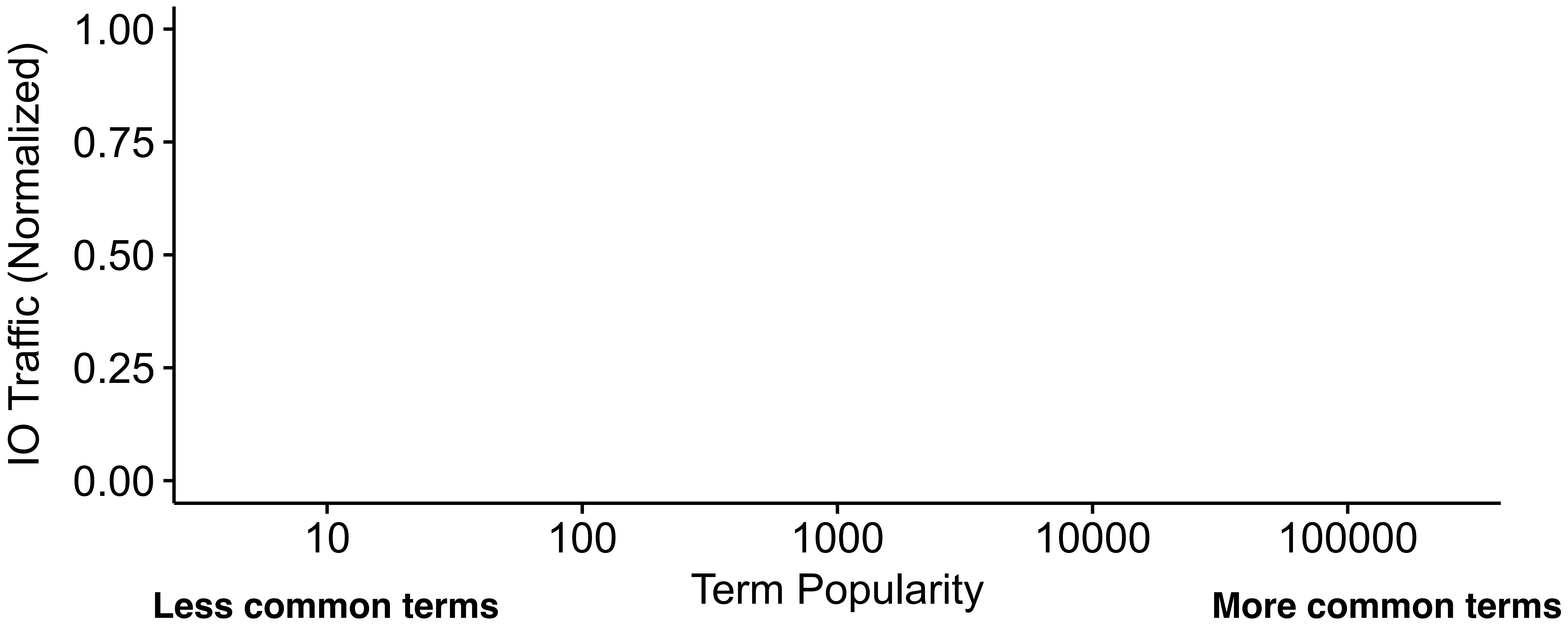
- single term queries, “and” queries, “phrase” queries, real queries
- vary term popularities in wikipedia

Machine

- 16 cores
- NVMe SSD: peak read 2 GB/s, 200,000 IOPS
- Linux container with 512-MB RAM
  - how well each search engine can scale up to large working sets that do not fit in main memory?

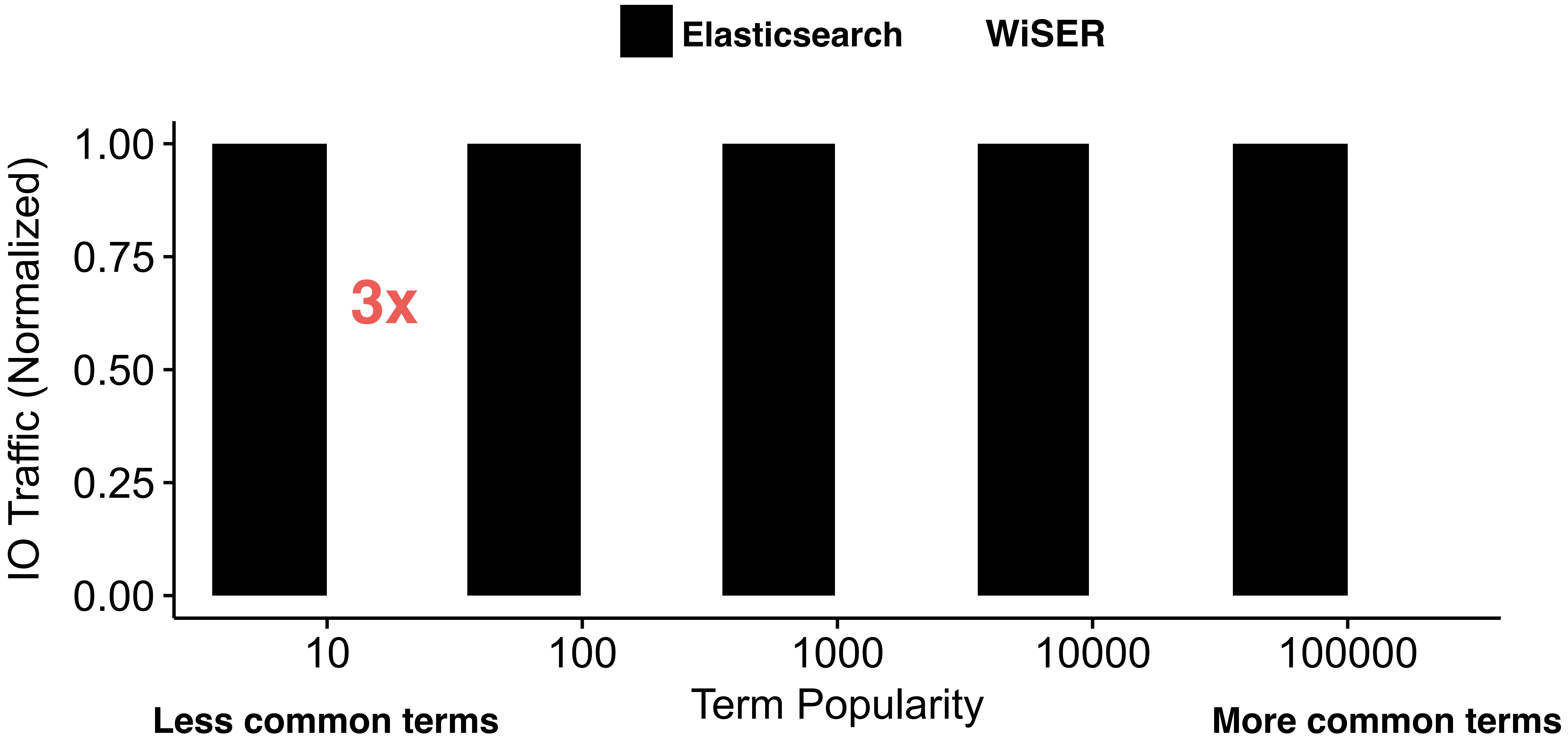
How much read traffic can “grouping by term” (technique 1) reduce?

How much read traffic can “grouping by term” (technique 1) reduce?



(single term query)

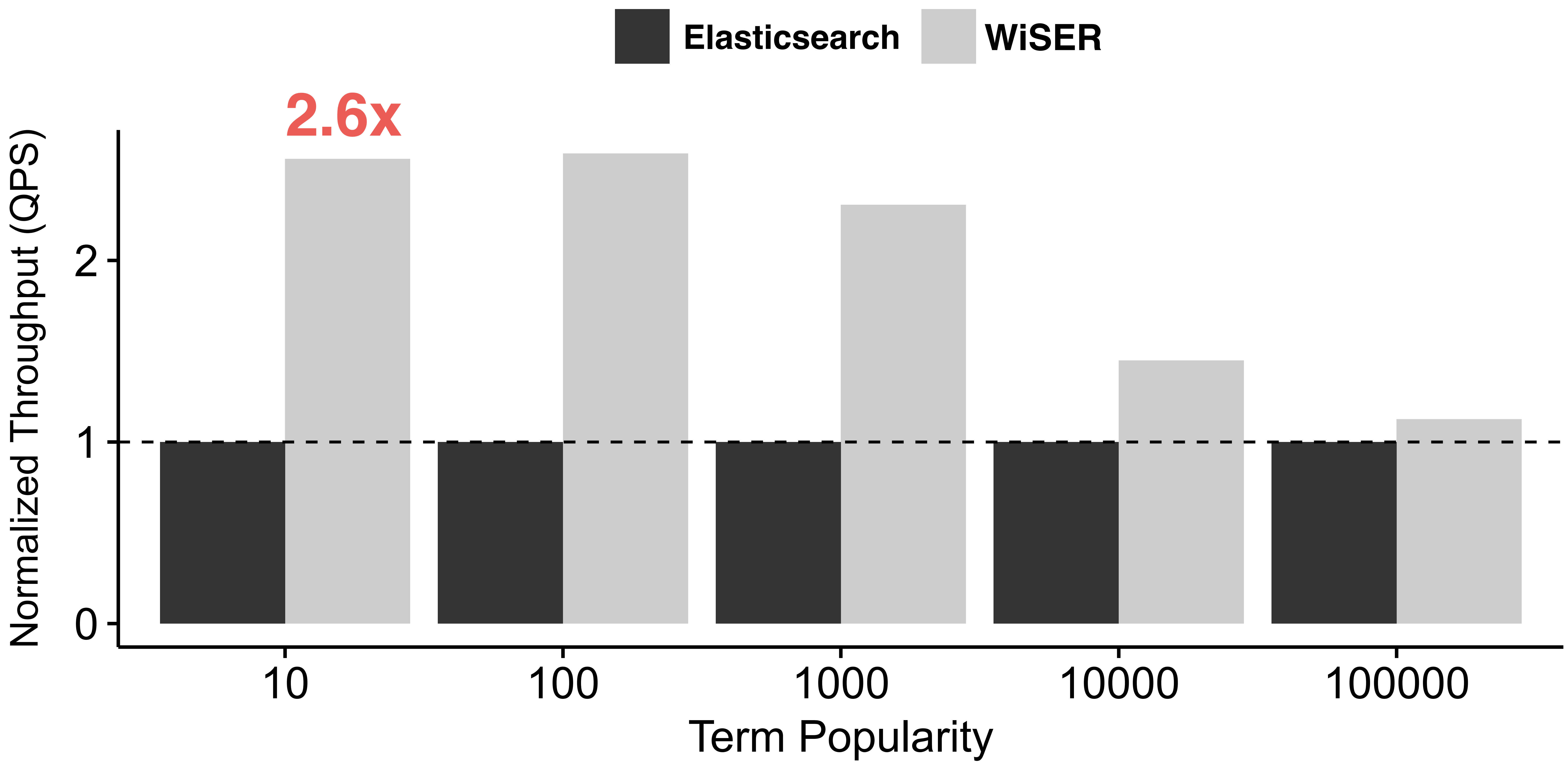
How much read traffic can "grouping by term" (technique 1) reduce?



(single term query)

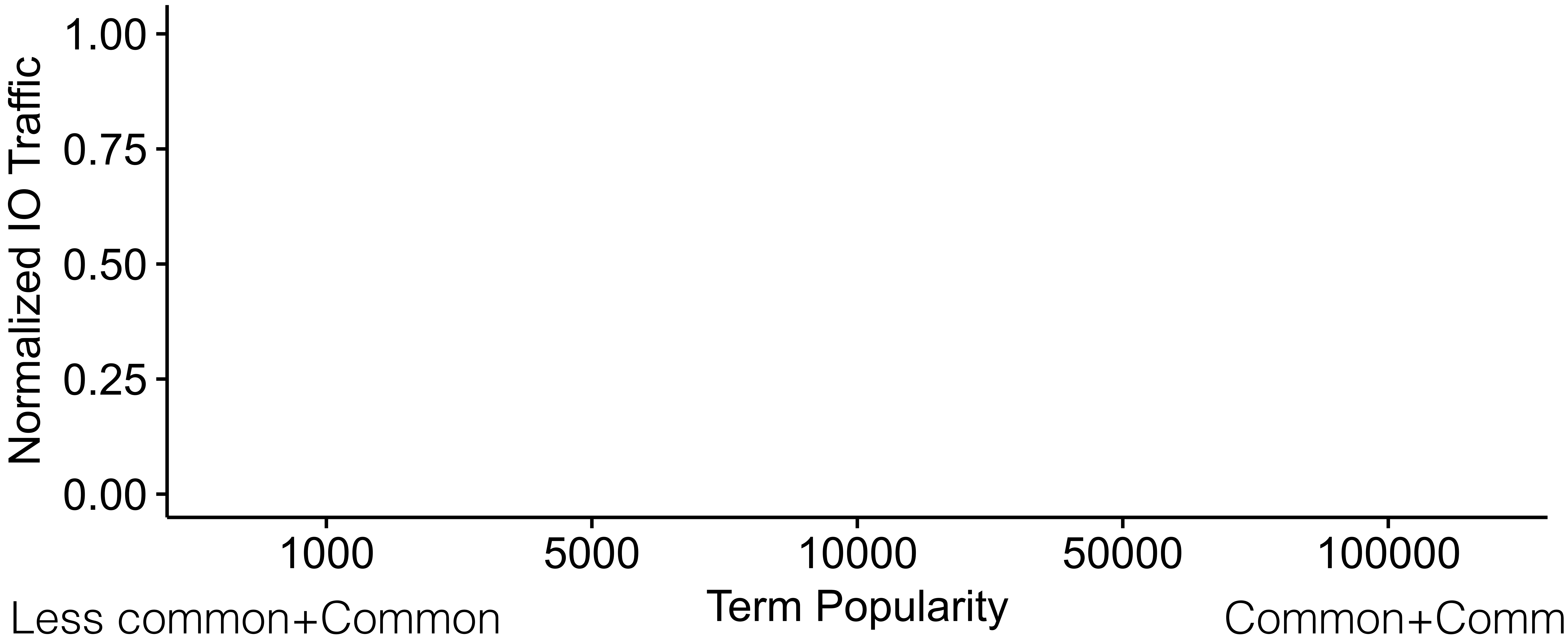
How much query throughput can "grouping by term"(technique 1) increase?

How much query throughput can “grouping by term”(technique 1) increase?



How much read traffic can two-way cost-aware Bloom filters (technique 2) reduce?

How much read traffic can two-way cost-aware Bloom filters (technique 2) reduce?

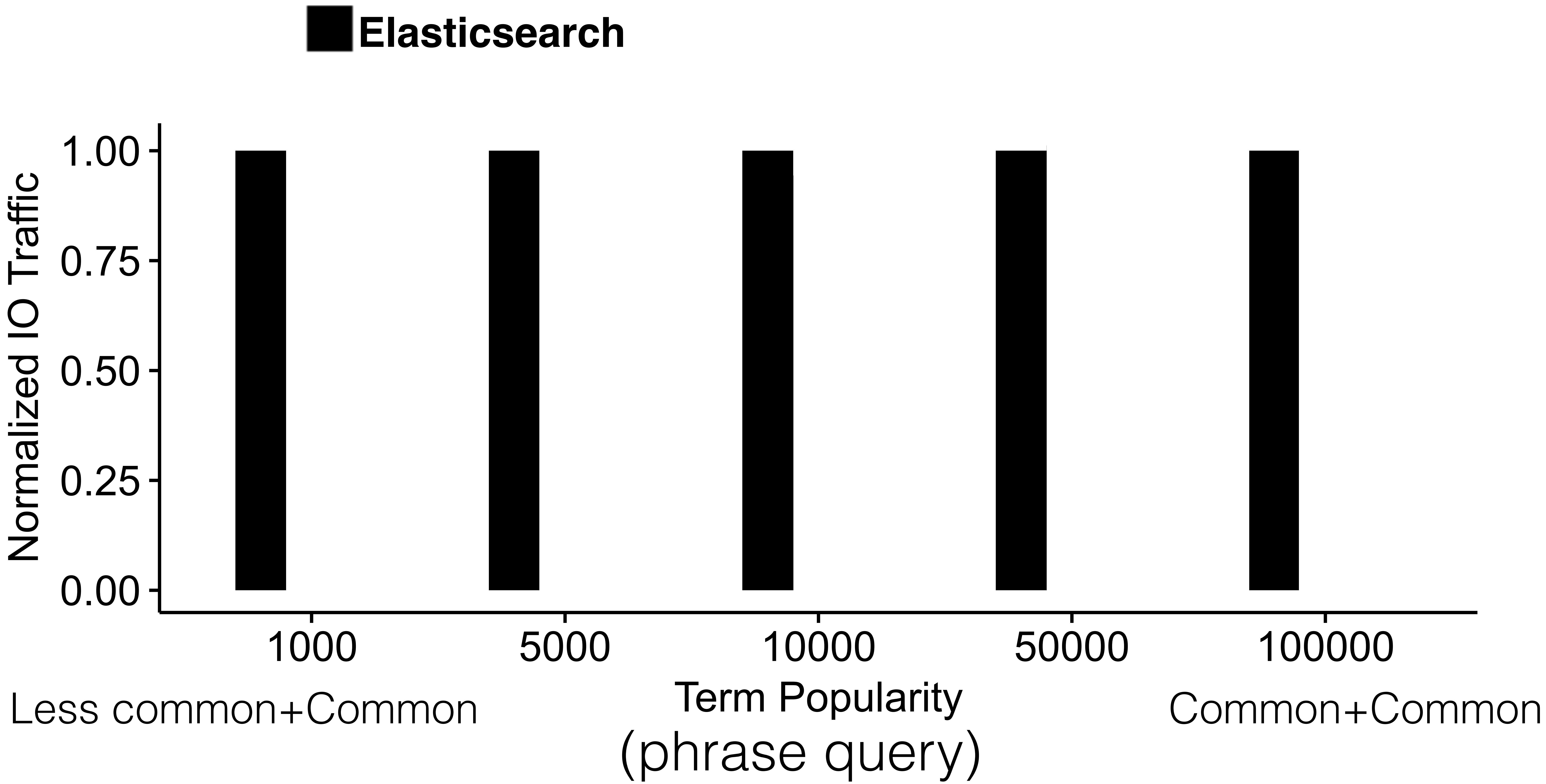


Less common+Common

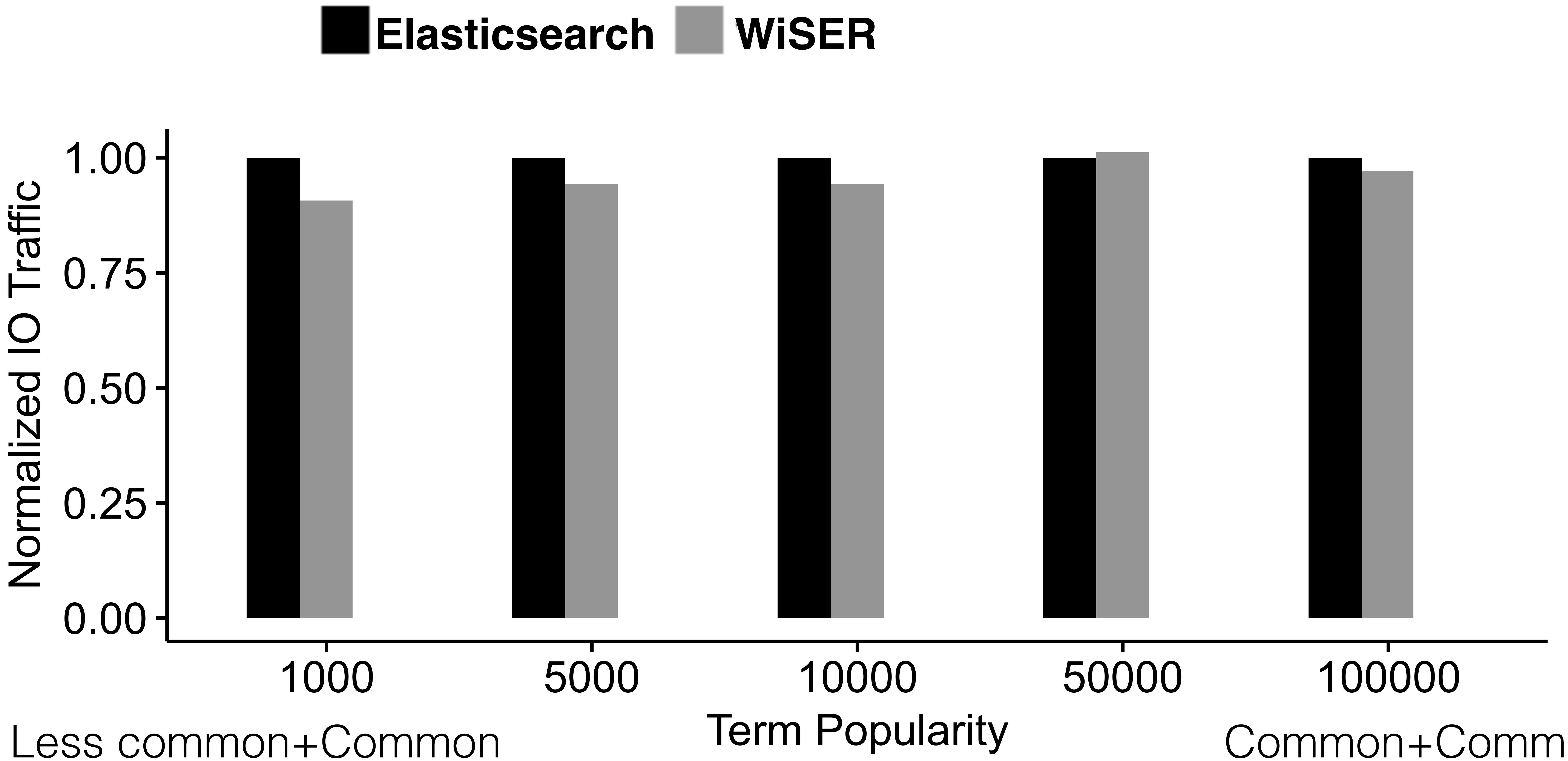
Common+Common

(phrase query)

How much read traffic can two-way cost-aware Bloom filters (technique 2) reduce?



How much read traffic can two-way cost-aware Bloom filters (technique 2) reduce?

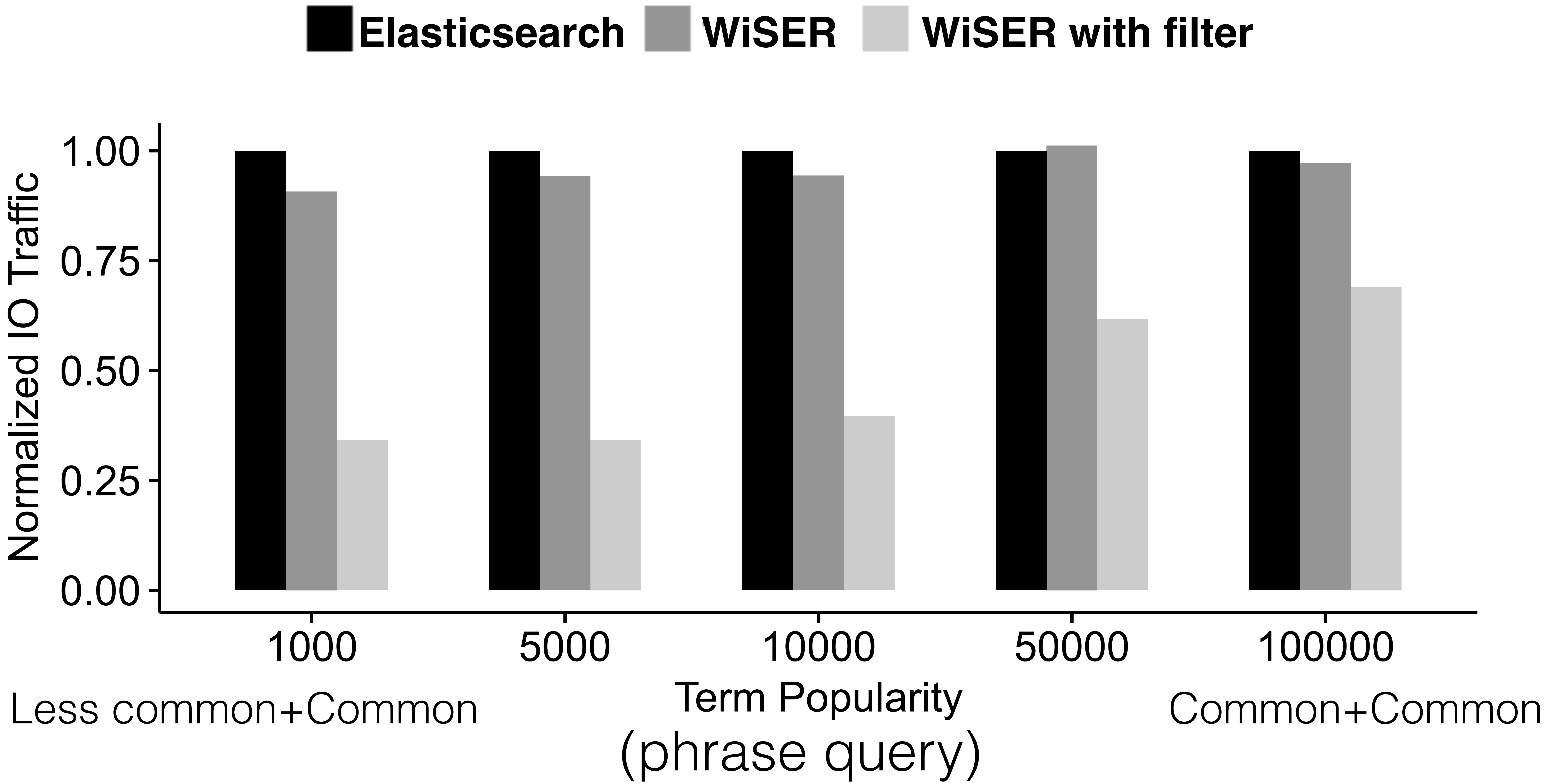


Less common+Common

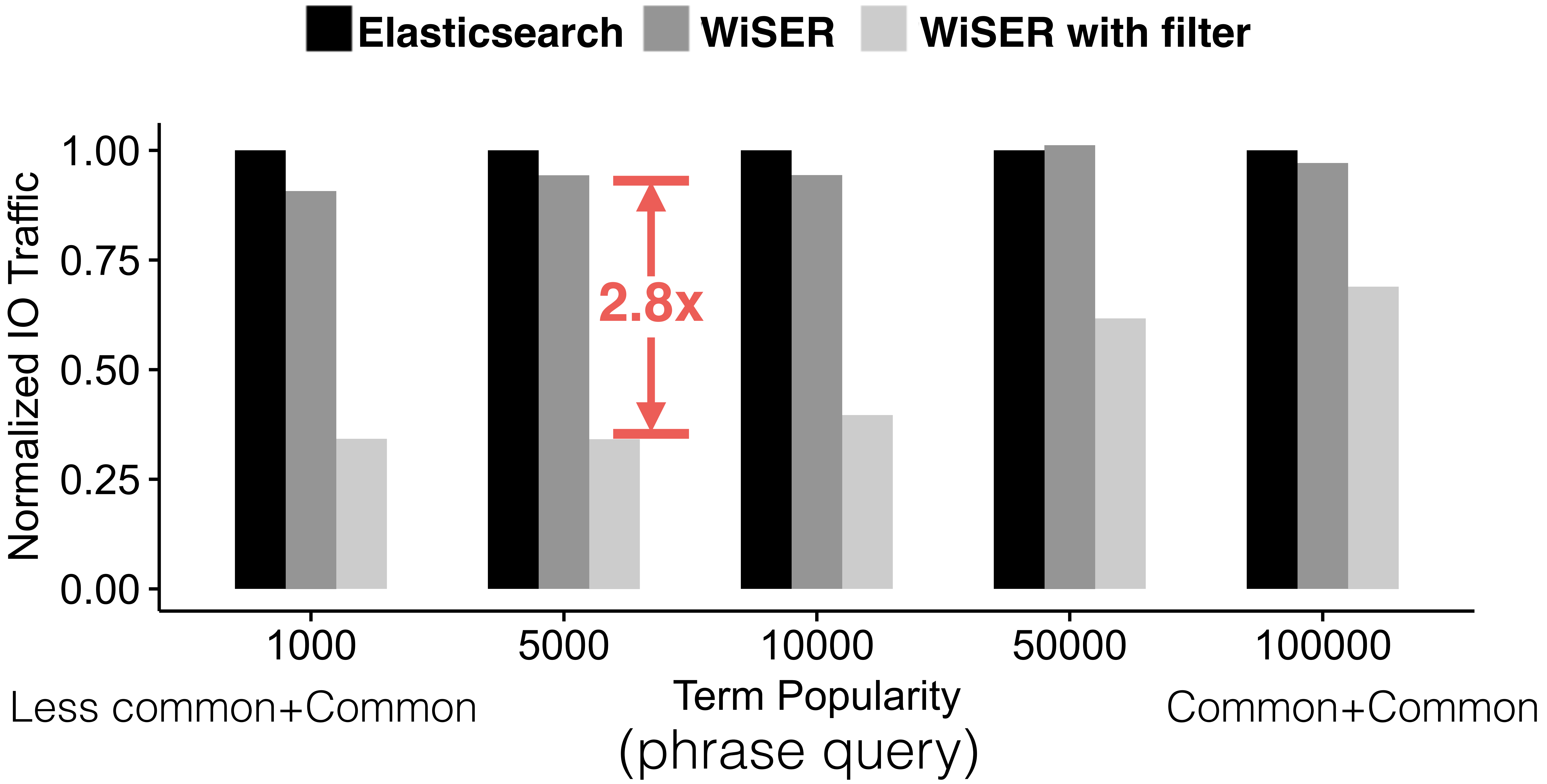
(phrase query)

Common+Common

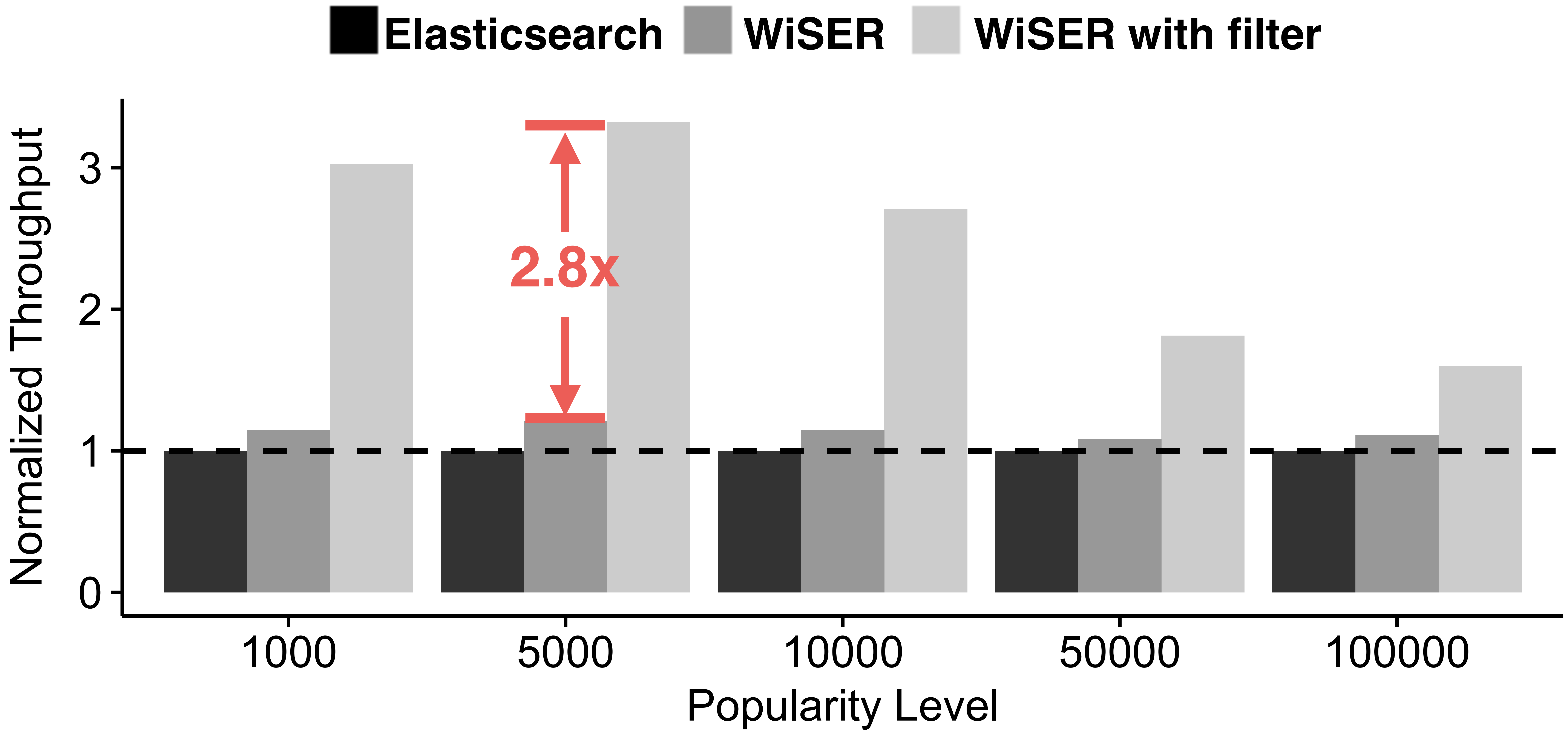
How much read traffic can two-way cost-aware Bloom filters (technique 2) reduce?



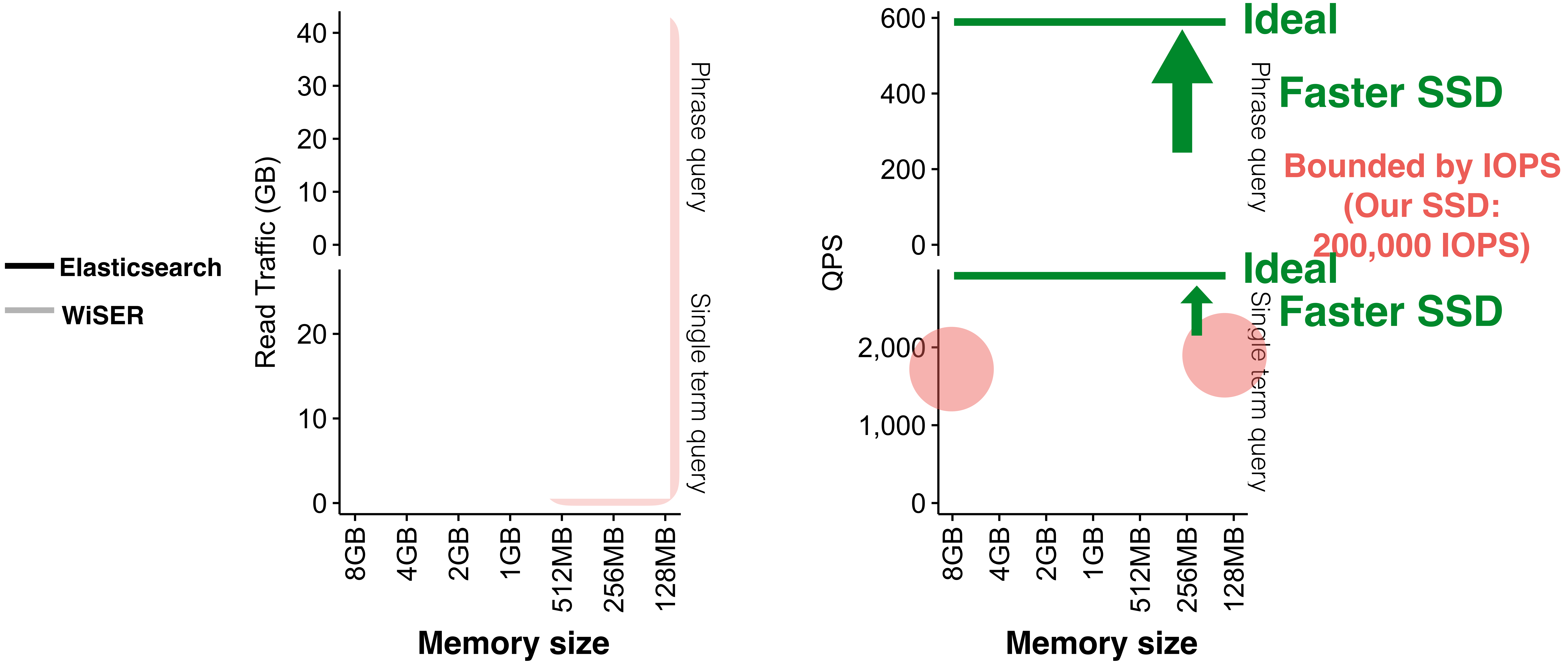
How much read traffic can two-way cost-aware Bloom filters (technique 2) reduce?



How much query throughput can two-way cost-aware filtering (technique 2) increase?



How does small-memory performance compare with larger-memory performance?



More results are in the paper…

Final Thoughts

Does your application really need large cache/RAM?

Will your application works just fine if it reads data as needed from fast SSDs?

Q & A

Ask as Needed