SC22

Dallas, TX | hpc accelerates.

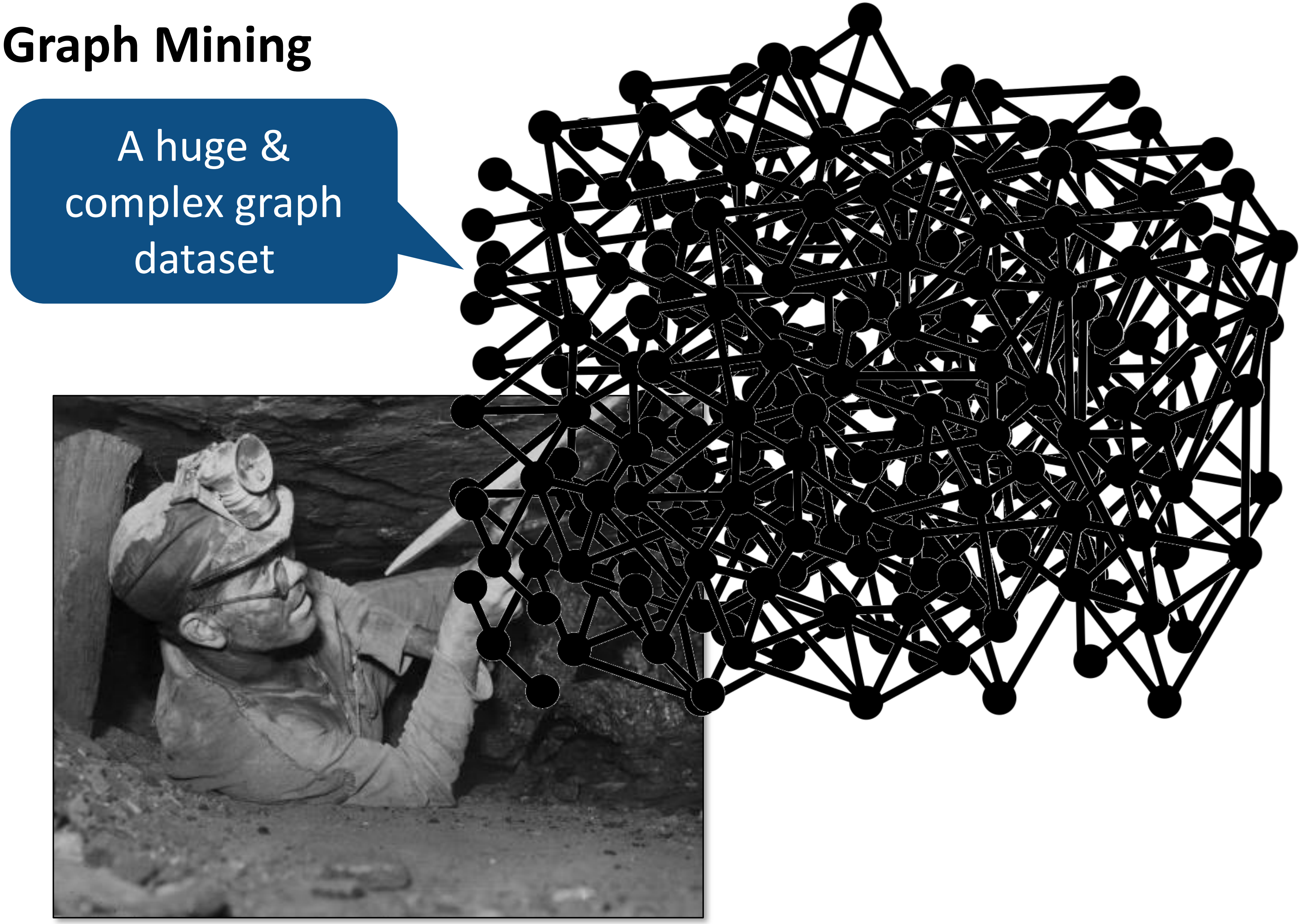
ProbGraph: High-Performance and High-Accuracy Graph Mining with Probabilistic Set Representations

M. BESTA, C. MIGLIOLI, P. S. LABINI, J. TĚTEK, P. IFF,
R. KANAKAGIRI, S. ASHKBOOS, K. JANDA, M. PODSTAWSKI,
G. KWASNIEWSKI, N. GLEINIG, F. VELLA, O. MUTLU, T. HOEFLER.

Graph Mining

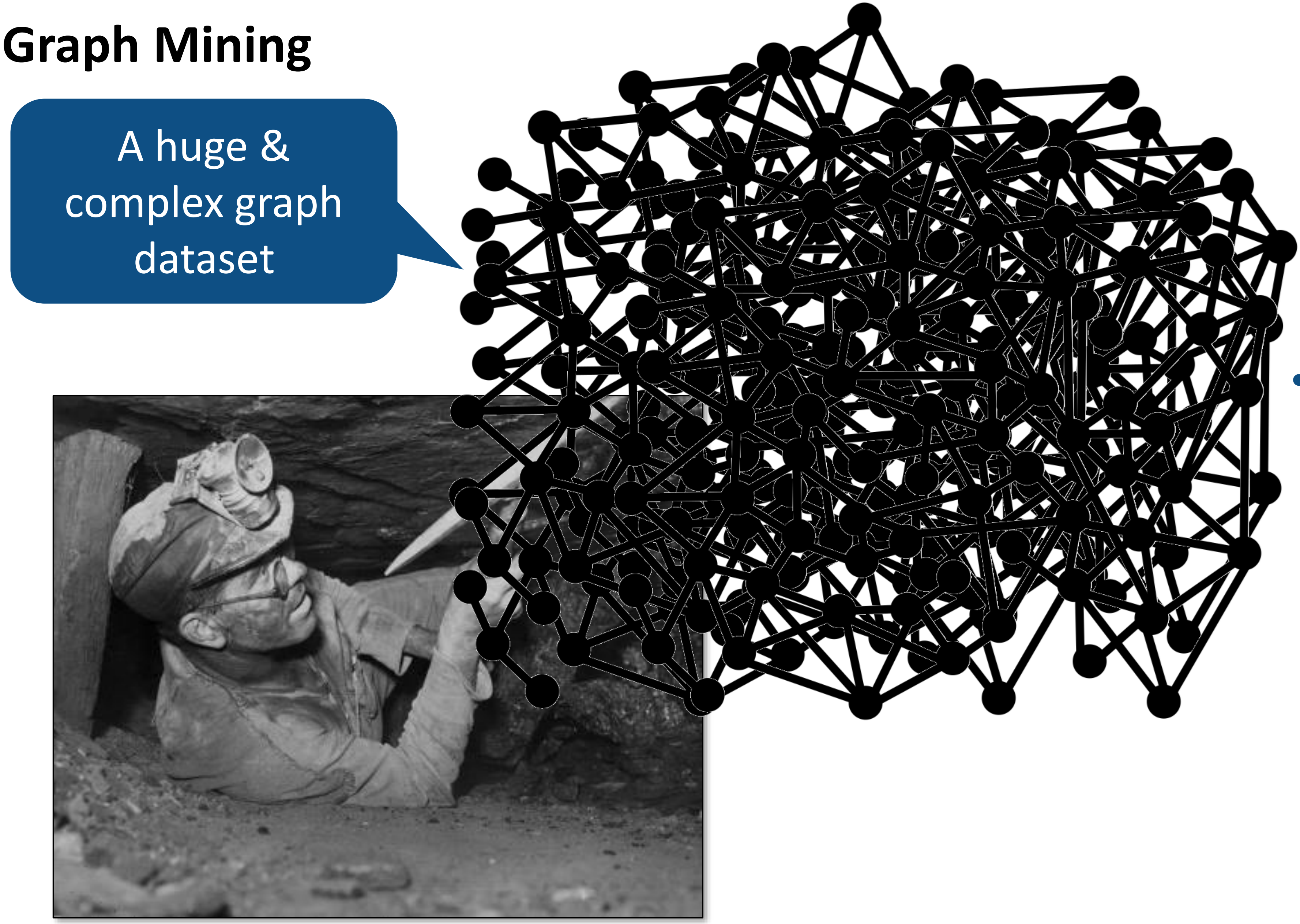
Graph Mining

A huge & complex graph dataset

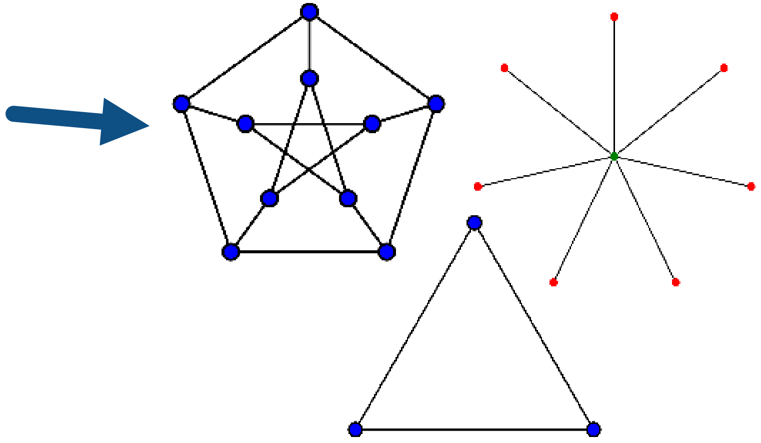


Graph Mining

A huge & complex graph dataset

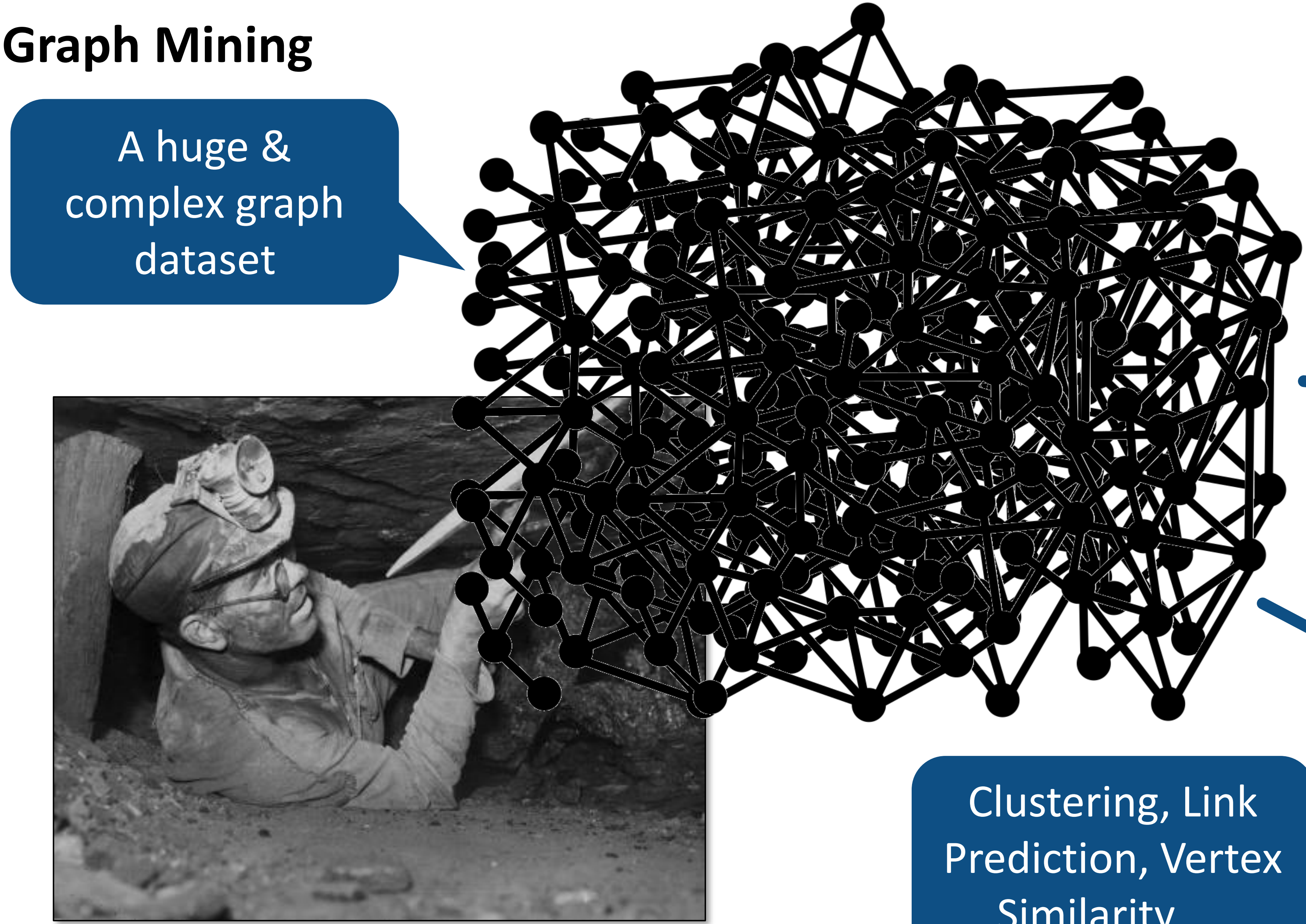


Pattern counting (triangles, higher-order cliques, dense subgraphs, ...)

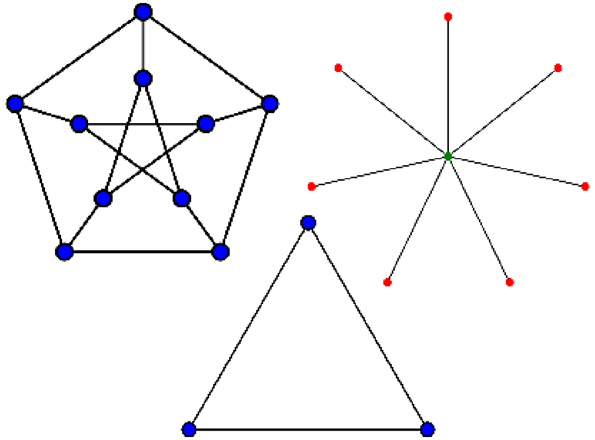


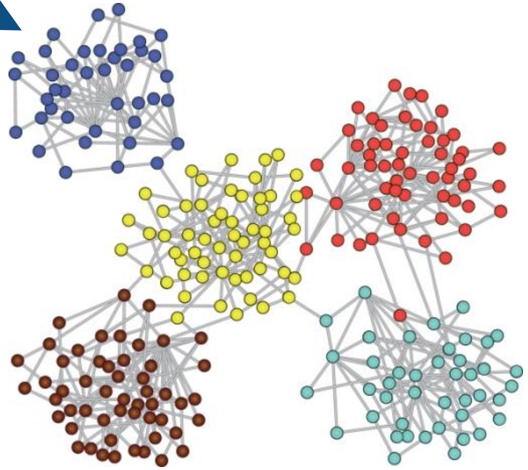
Graph Mining

A huge & complex graph dataset



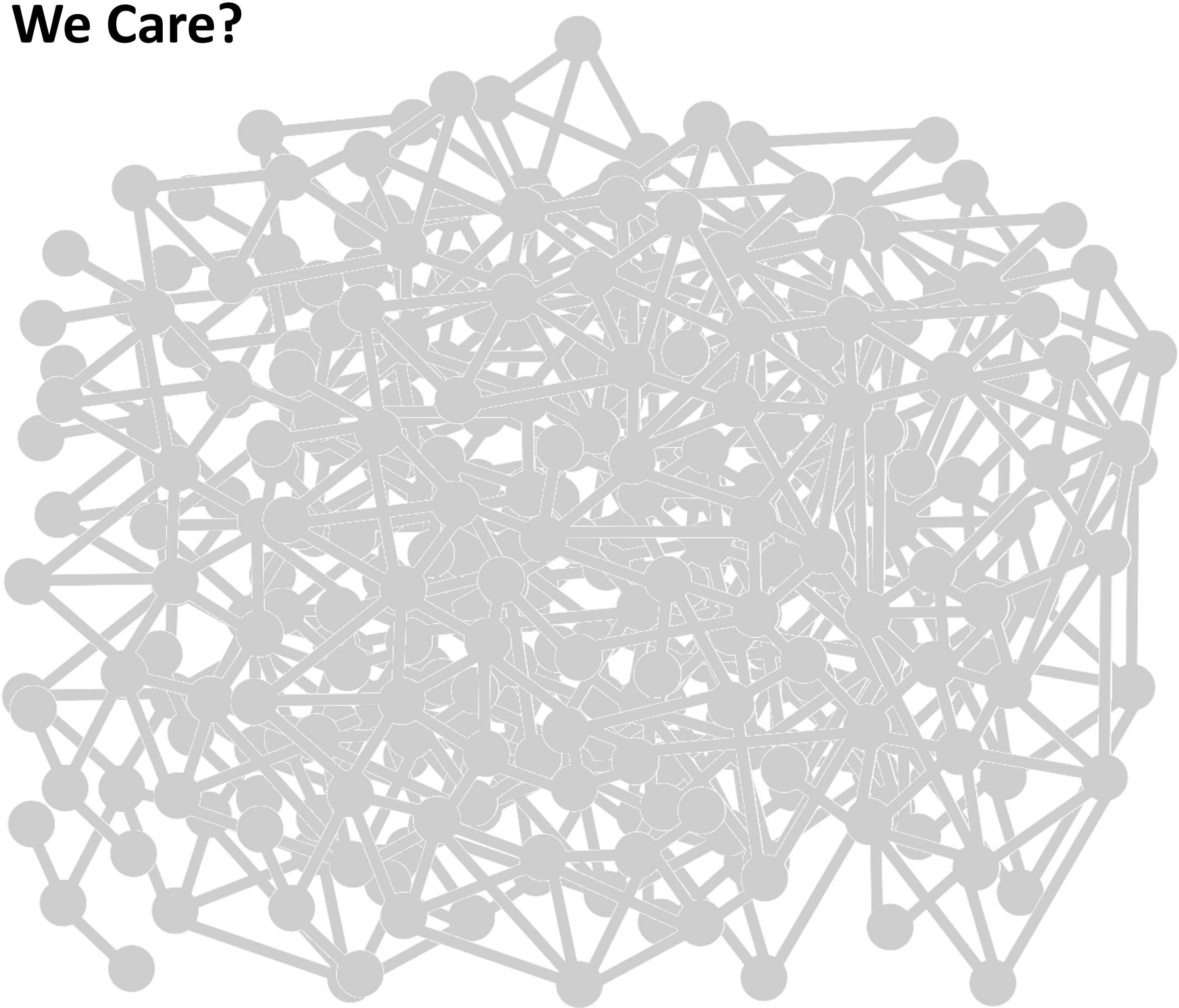
Pattern counting (triangles, higher-order cliques, dense subgraphs, ...)





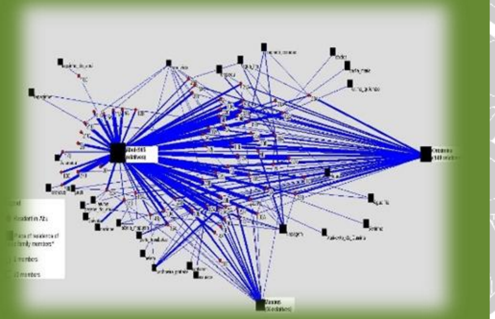
Clustering, Link Prediction, Vertex Similarity, ...

Graph Mining: Do We Care?

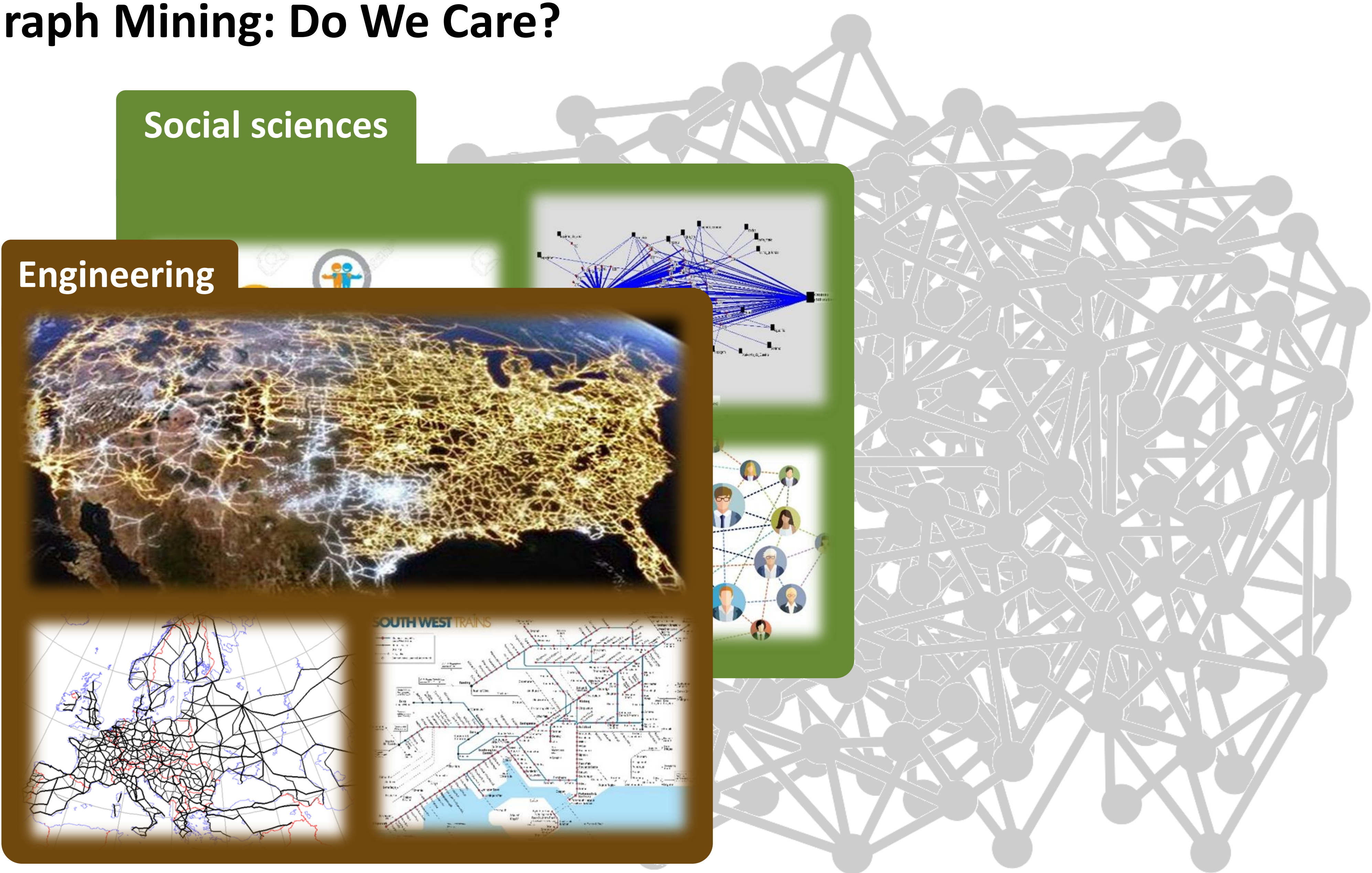


Graph Mining: Do We Care?

Social sciences



Graph Mining: Do We Care?



Social sciences

Engineering

Graph Mining: Do We Care?

Social sciences

Biology

Chemistry

Communication

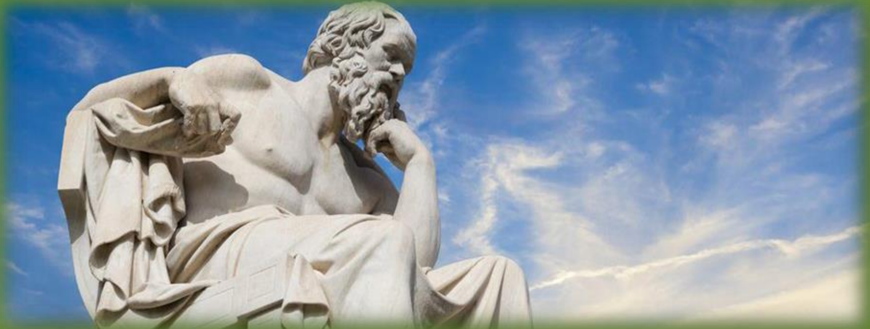
Engineering

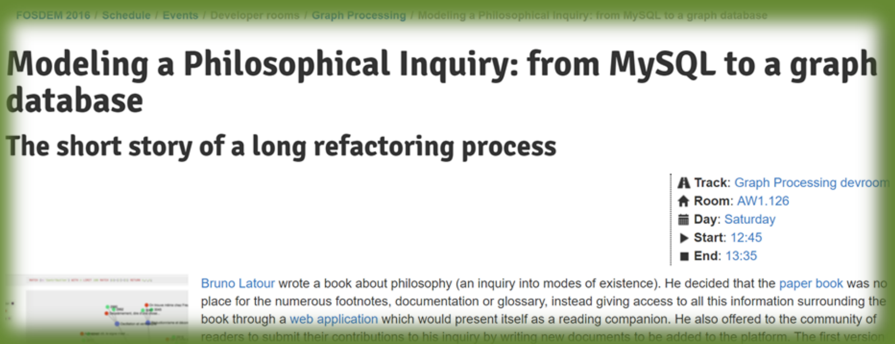
Medicine

Cybersecurity

Web graph analysis

...even philosophy ☺





FOSDEM 2016 / Schedule / Events / Developer rooms / Graph Processing / Modeling a Philosophical Inquiry: from MySQL to a graph database

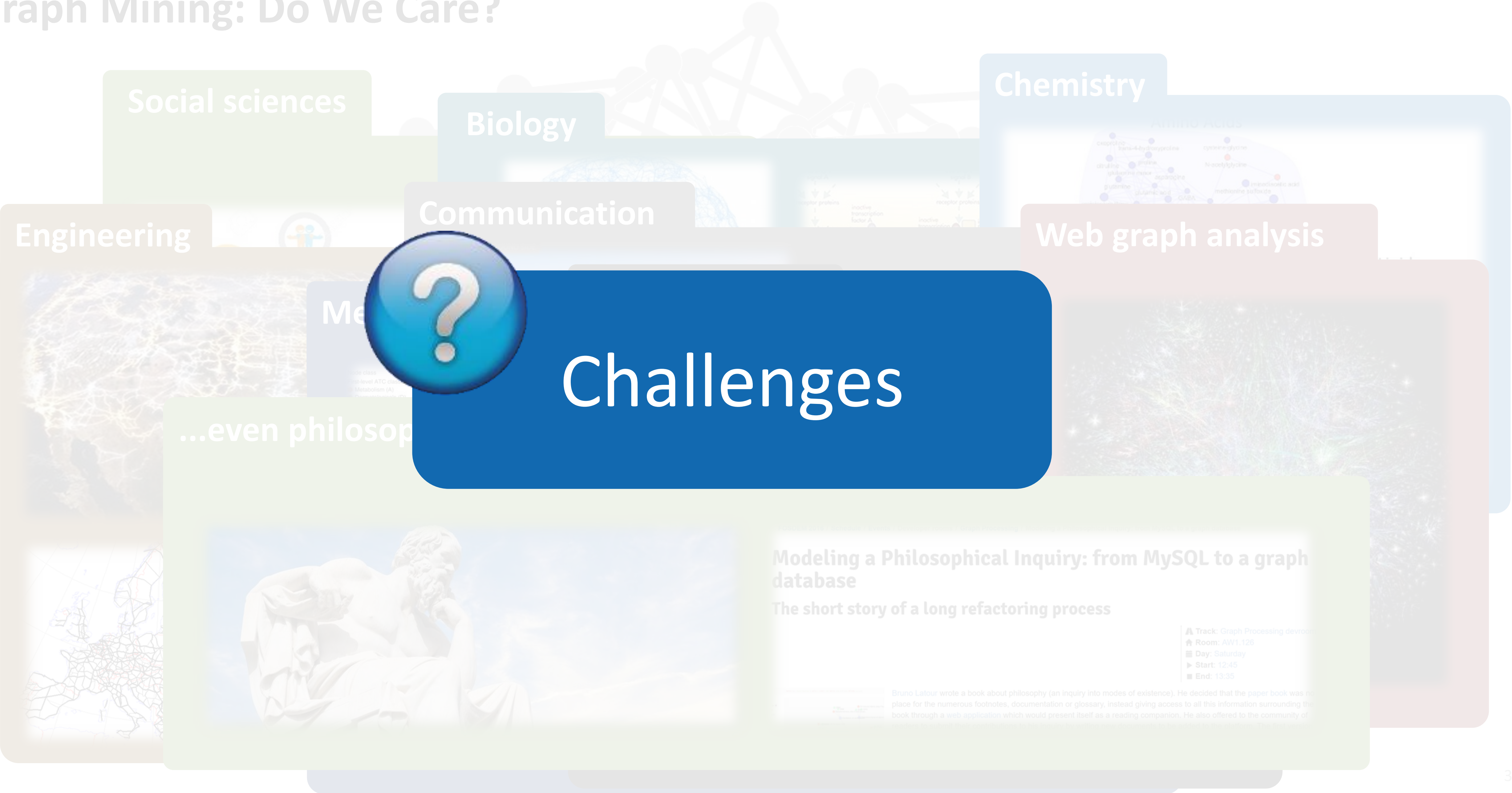
Modeling a Philosophical Inquiry: from MySQL to a graph database

The short story of a long refactoring process

- Track: Graph Processing devroom
- Room: AW1.126
- Day: Saturday
- Start: 12:45
- End: 13:35

Bruno Latour wrote a book about philosophy (an inquiry into modes of existence). He decided that the paper book was no place for the numerous footnotes, documentation or glossary, instead giving access to all this information surrounding the book through a web application which would present itself as a reading companion. He also offered to the community of readers to submit their contributions to his inquiry by writing new documents to be added to the platform. The first version

Graph Mining: Do We Care?



Modeling a Philosophical Inquiry: from MySQL to a graph database

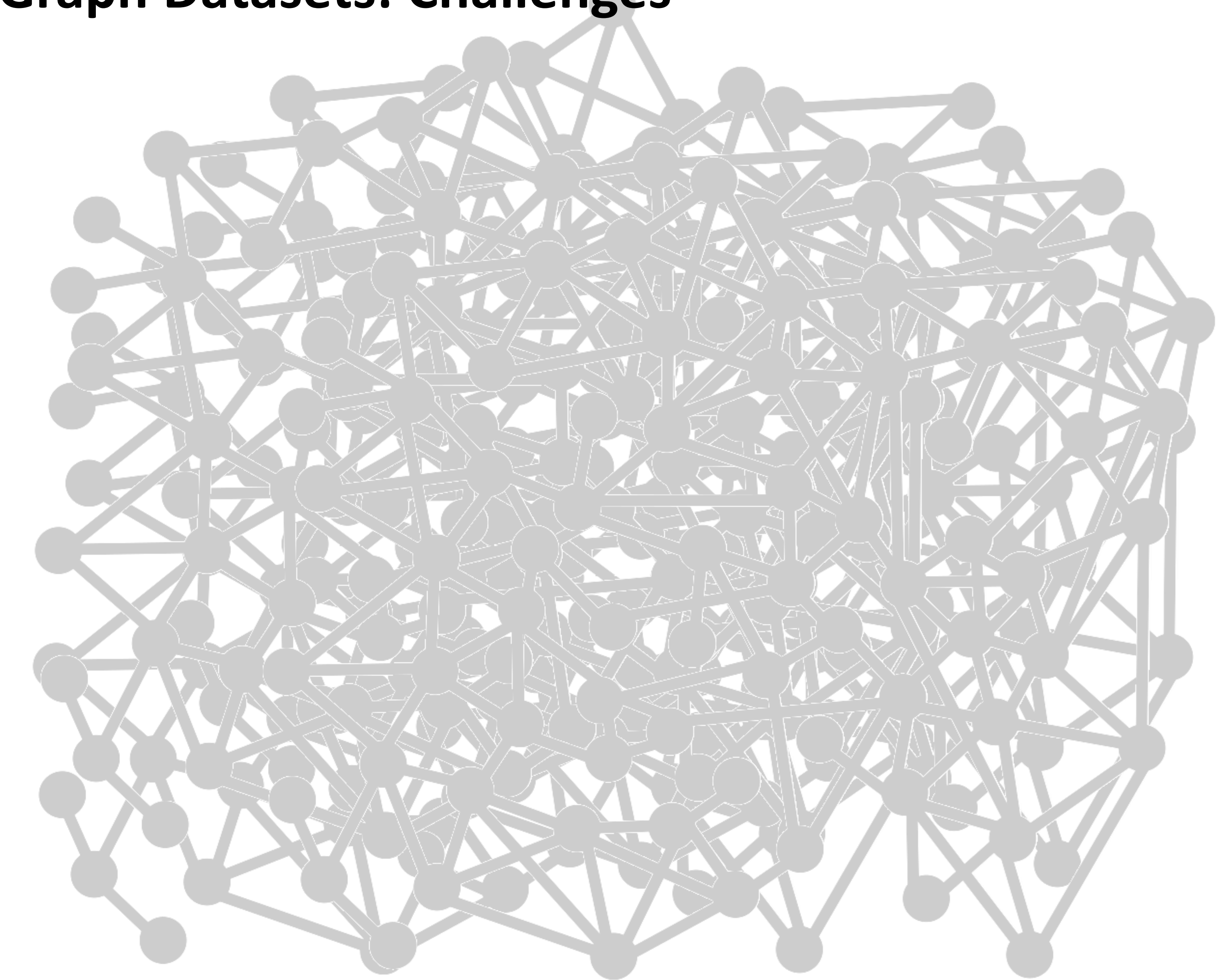
The short story of a long refactoring process

Track: Graph Processing devroom
Room: AW1.126
Day: Saturday
Start: 12:45
End: 13:35

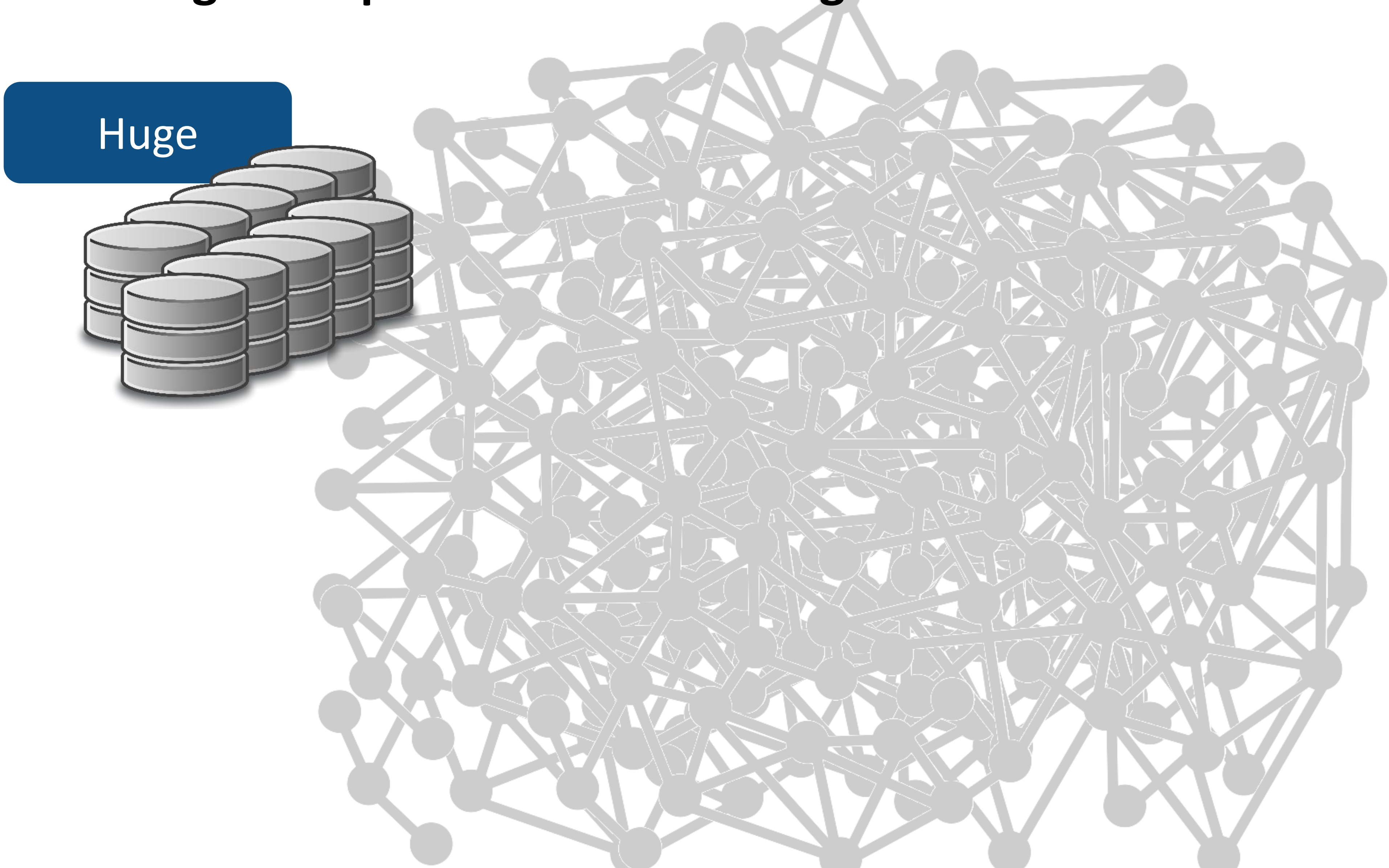
Bruno Latour wrote a book about philosophy (an inquiry into modes of existence). He decided that the paper book was no place for the numerous footnotes, documentation or glossary, instead giving access to all this information surrounding the book through a web application which would present itself as a reading companion. He also offered to the community of

Challenges

Graph Mining & Graph Datasets: Challenges



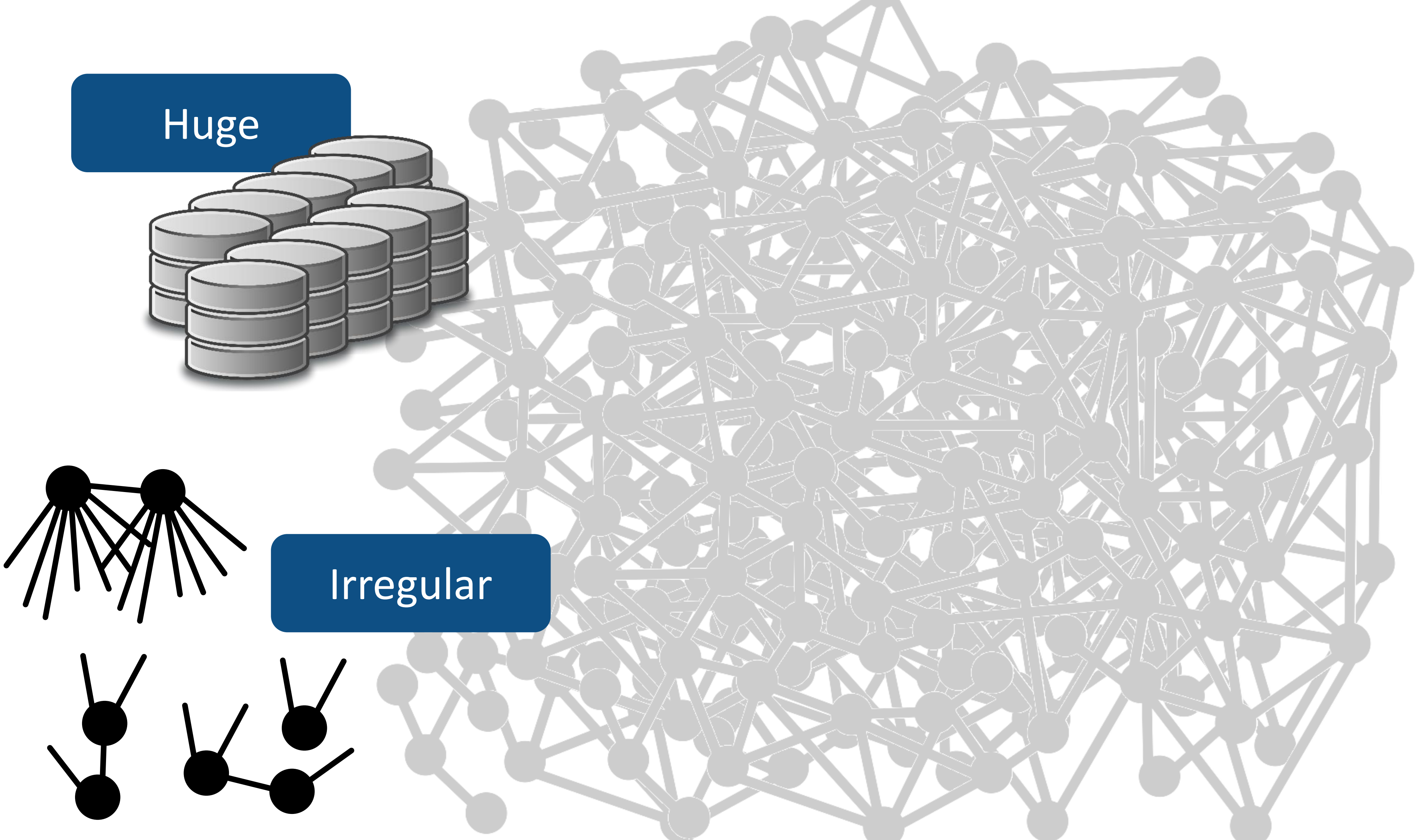
Graph Mining & Graph Datasets: Challenges



Huge

[1] Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

Graph Mining & Graph Datasets: Challenges

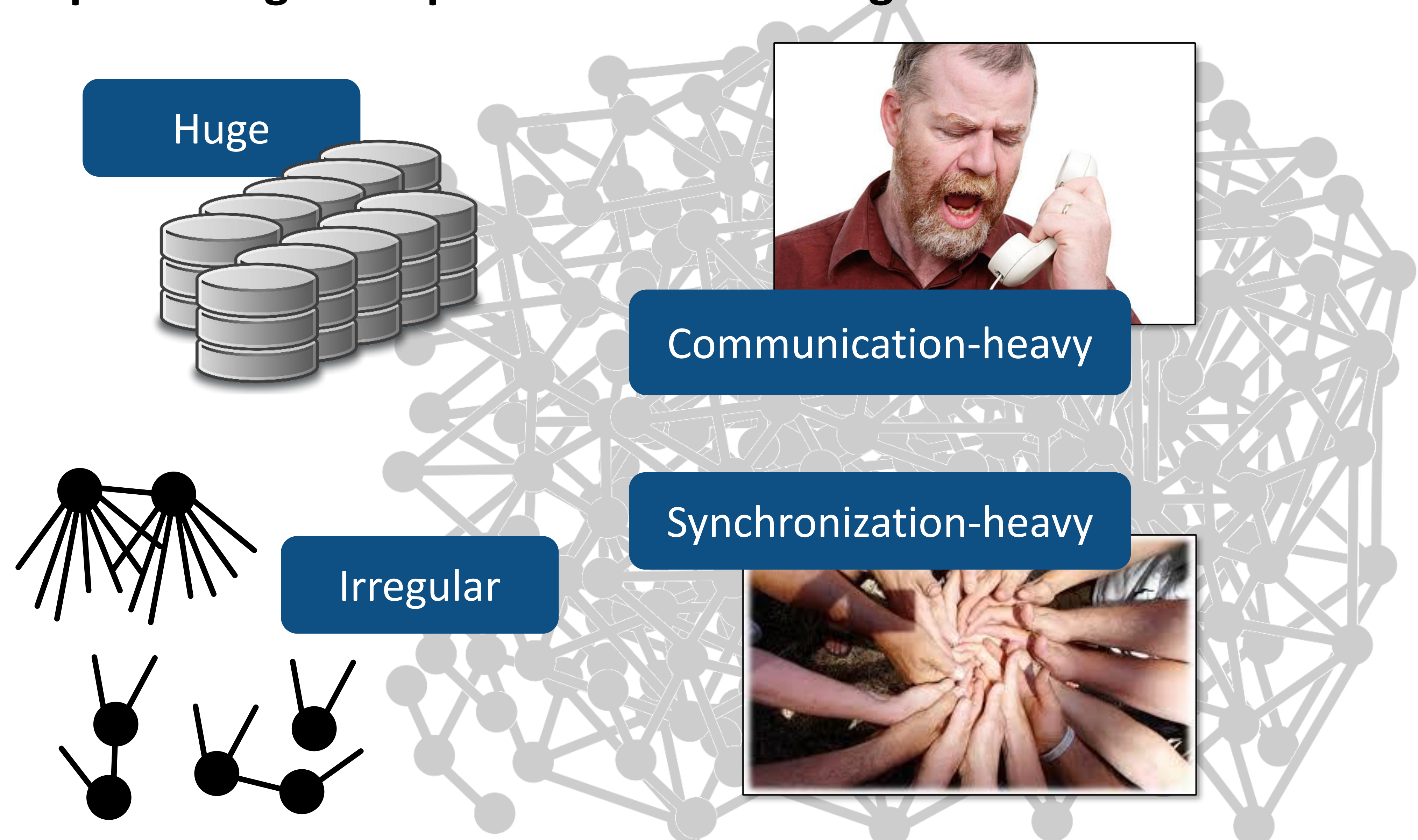


Huge

Irregular

[1] Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

Graph Mining & Graph Datasets: Challenges



Huge

Communication-heavy

Synchronization-heavy

Irregular

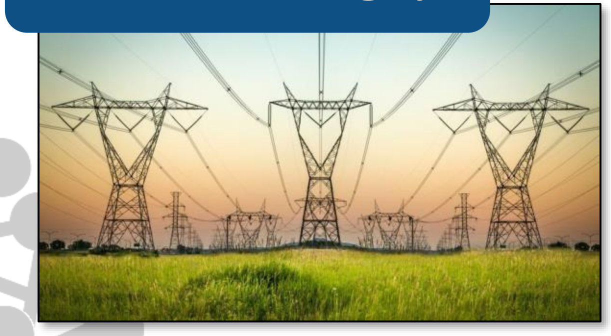
[1] Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

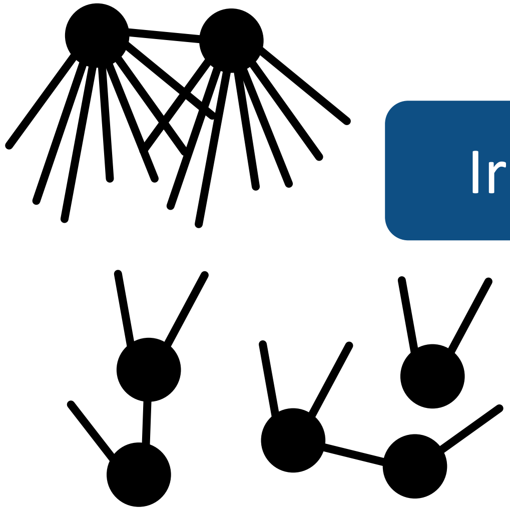
Graph Mining & Graph Datasets: Challenges

Huge

Communication-heavy

Power-hungry





Irregular

Synchronization-heavy

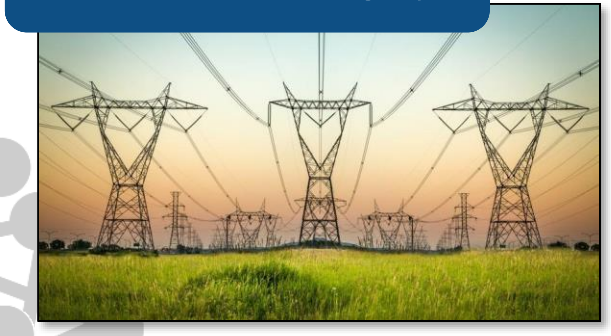
[1] Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

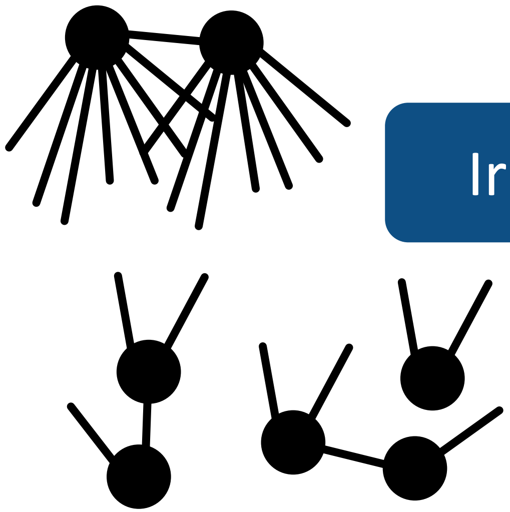
Graph Mining & Graph Datasets: Challenges

Huge

Communication-heavy

Power-hungry





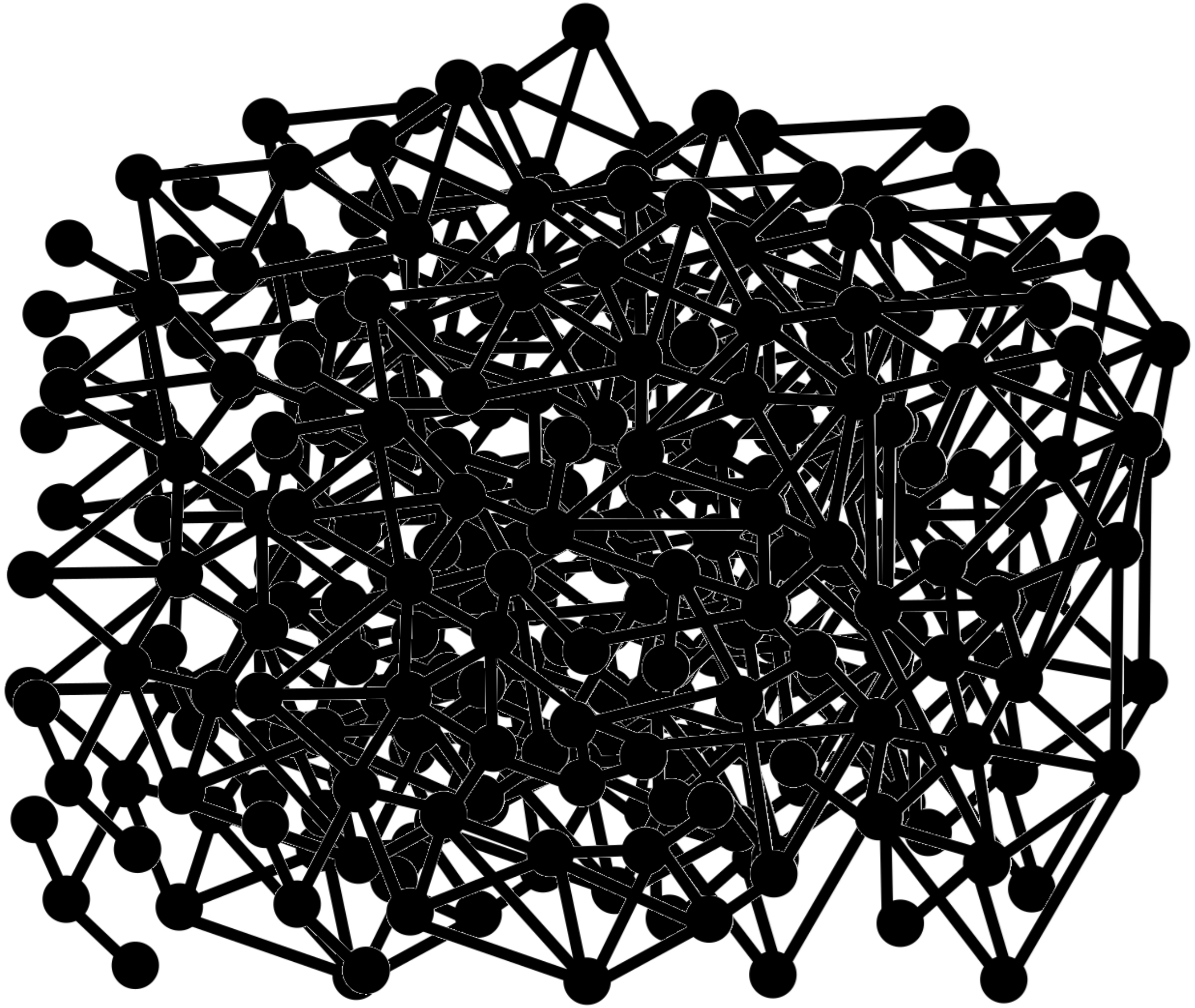
Irregular

Synchronization-heavy

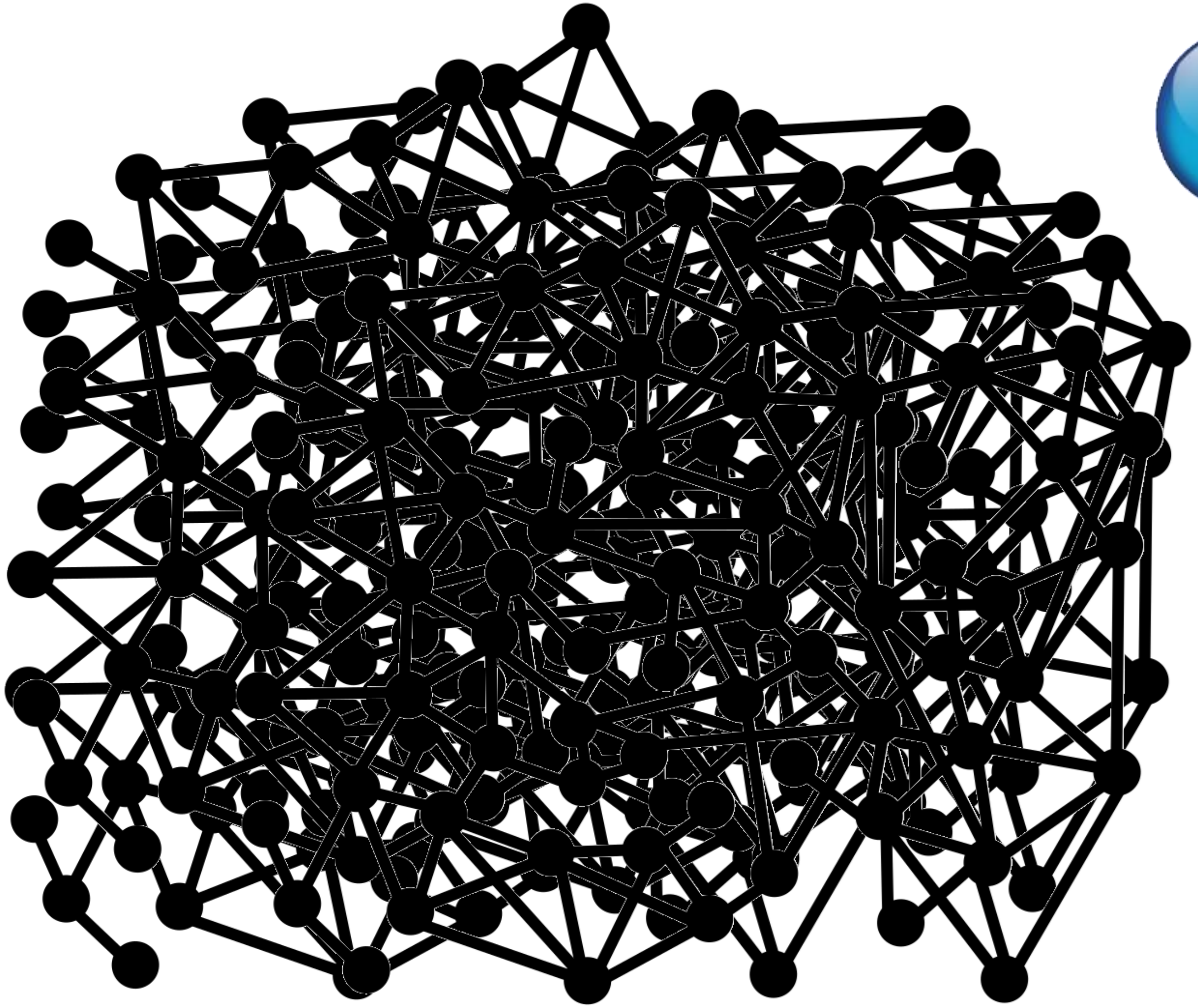
Time complexities often O(n^k) for $k \geq 2$

[1] Heng Lin et al.: ShenTu: Processing Multi-Trillion Edge Graphs on Millions of Cores in Seconds, SC18, Gordon Bell Finalist

Goal: Making Graph Mining Radically Faster

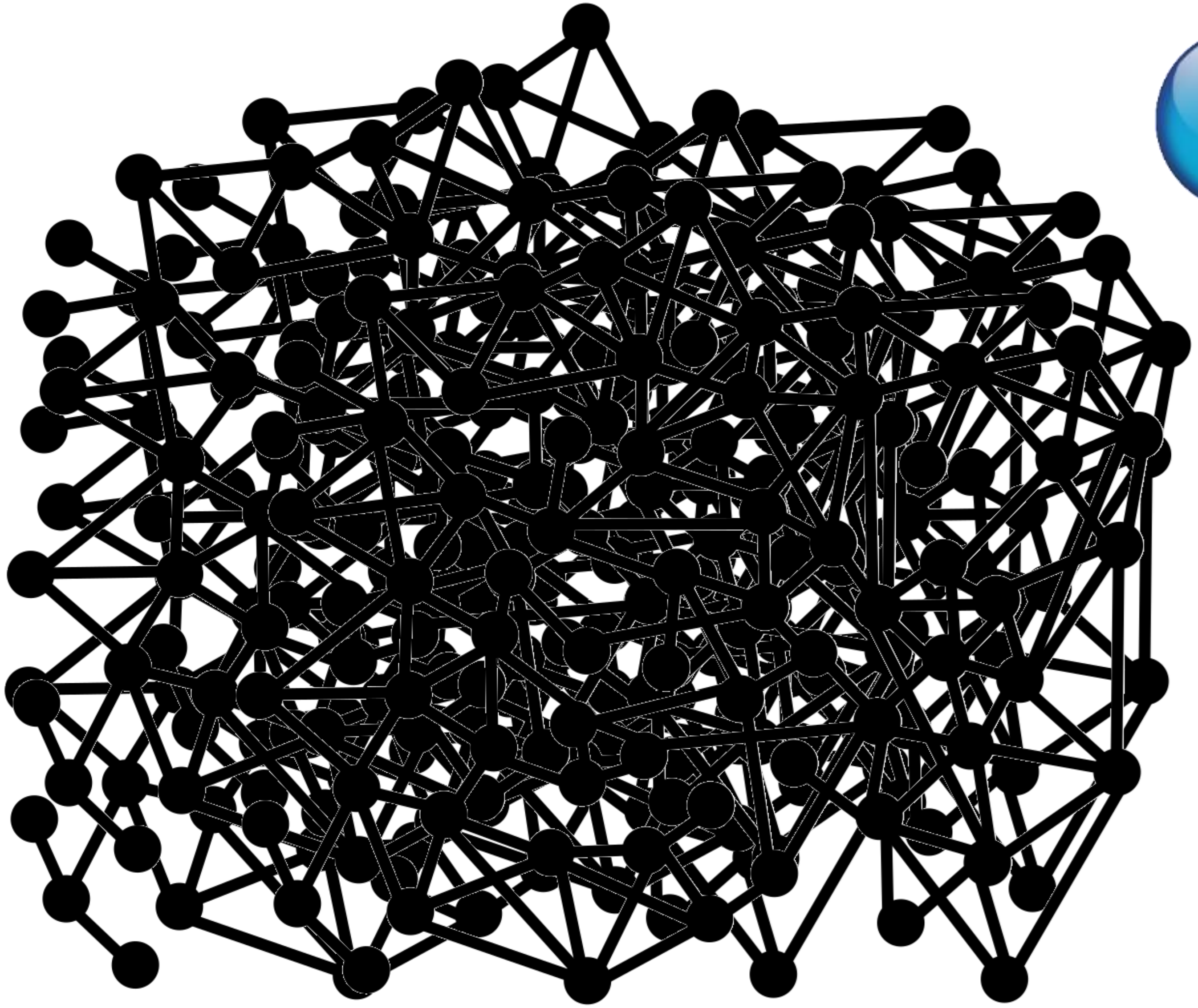


Goal: Making Graph Mining Radically Faster



Do we need 100% accurate results in all cases?

Goal: Making Graph Mining Radically Faster



Do we need 100% accurate results in all cases?

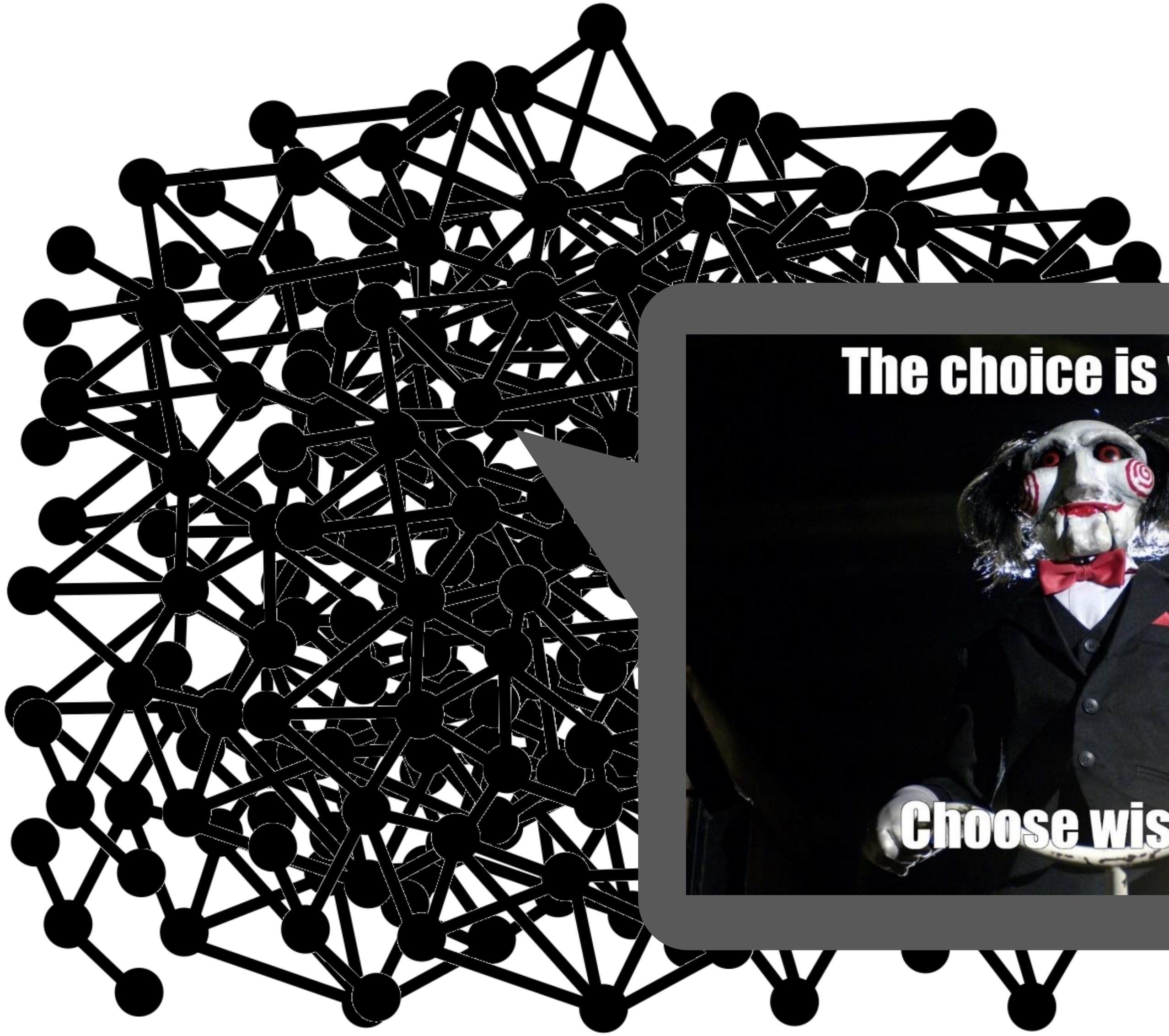
Let's say we can choose between...

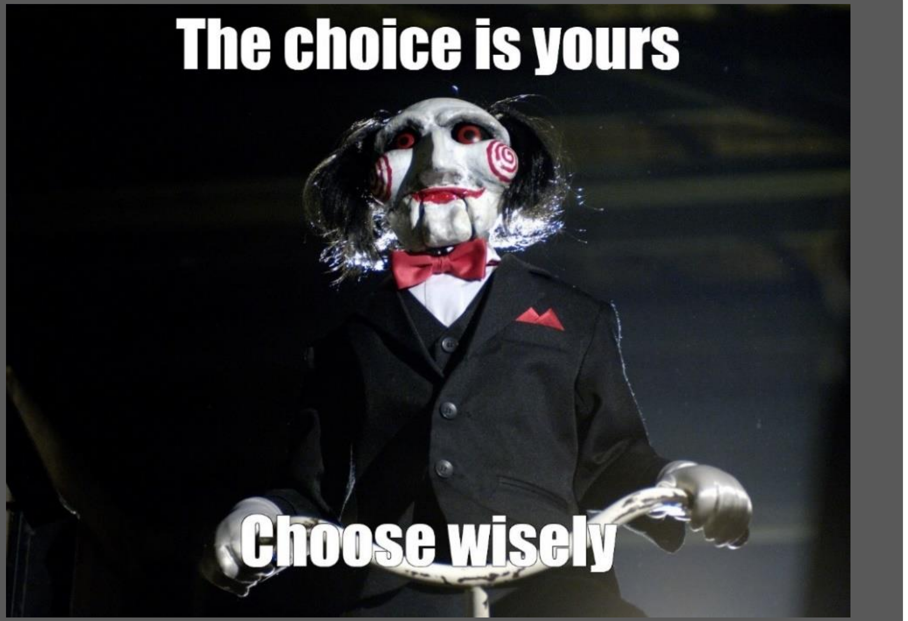
Find all the patterns (e.g., cliques) in 1 day

Find ≥ 90% of all the patterns in 30 minutes

Goal: Making Graph Mining Radically Faster

Do we need 100% accurate results in all cases?





Let’s say we can choose between...

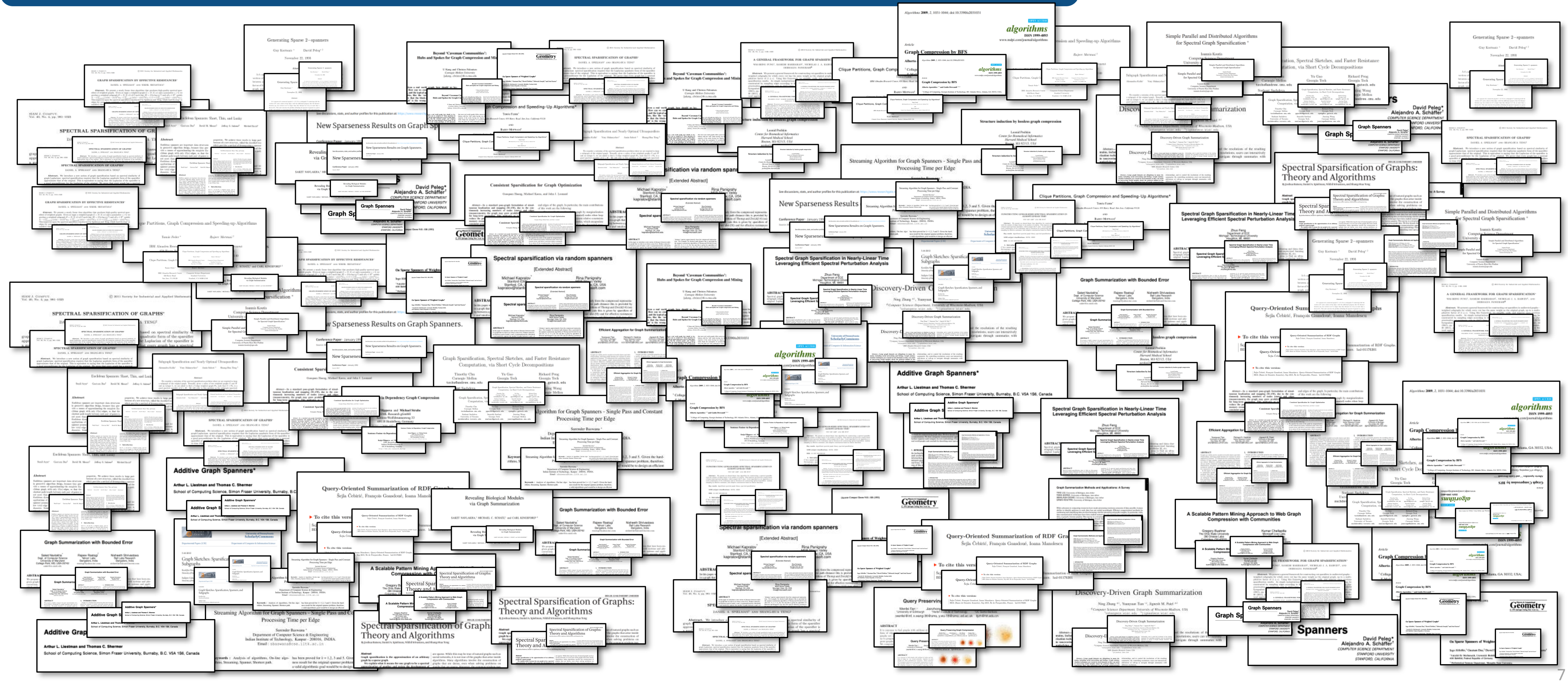
Find all the patterns (e.g., cliques) in 1 day

Find ≥ 90% of all the patterns in 30 minutes

Approximate Graph Processing: State & Challenges

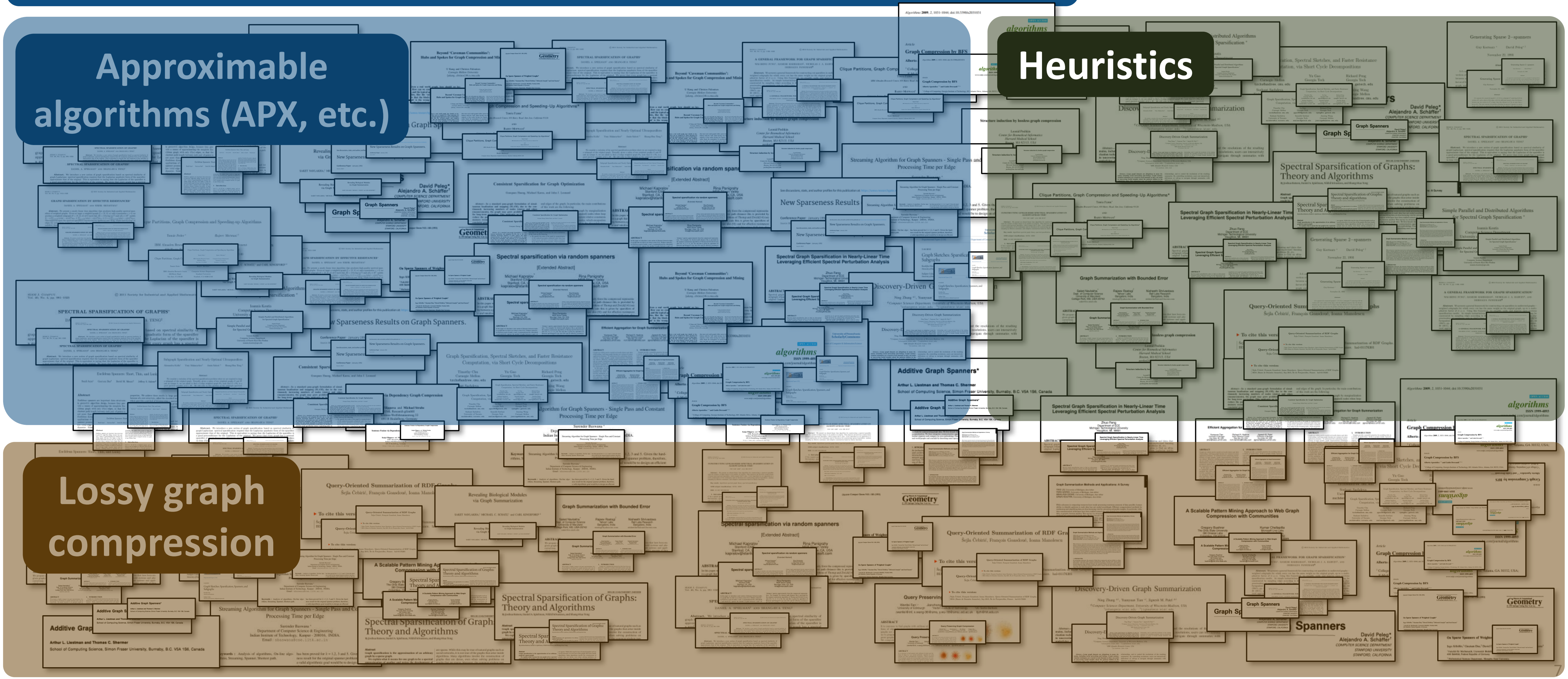
Approximate Graph Processing: State & Challenges

We analyzed > 500 works and identified three classes of schemes...


Generating Sparse 2-spanners
Spectral Sparsification of Graphs
Graph Sparsification by Effective Resistances
New Sparseness Results on Graph Spanners
Graph Spanners
Clique Partitions, Graph Compression and Speeding-Up Algorithms
Graph Compression by BFS
algorithms
Structure induction by lossless graph compression
Streaming Algorithm for Graph Spanners - Single Pass and Constant Processing Time per Edge
Spectral Sparsification of Graphs: Theory and Algorithms
Simple Parallel and Distributed Algorithms for Spectral Graph Sparsification
Spectral sparsification via random spanners
Consistent Sparsification for Graph Optimization
Spectral Graph Sparsification in Nearly-Linear Time Leveraging Efficient Spectral Perturbation Analysis
Graph Summarization with Bounded Error
Discovery-Driven Graph Summarization
Query-Oriented Summarization of RDF Graphs
Additive Graph Spanners
Graph Sketches, Sparsification, Spanners, and Subgraphs
Graph Sparsification, Spectral Sketches, and Faster Resistance Computation, via Short Cycle Decompositions
A Scalable Pattern Mining Approach to Web Graph Compression with Communities
Revealing Biological Modules via Graph Summarization
Efficient Aggregation for Graph Summarization
Geometry
Beyond 'Caveman Communities': Hubs and Spokes for Graph Compression and Mining
A General Framework for Graph Sparsification

Approximate Graph Processing: State & Challenges

We analyzed > 500 works and identified three classes of schemes...



Approximable algorithms (APX, etc.)

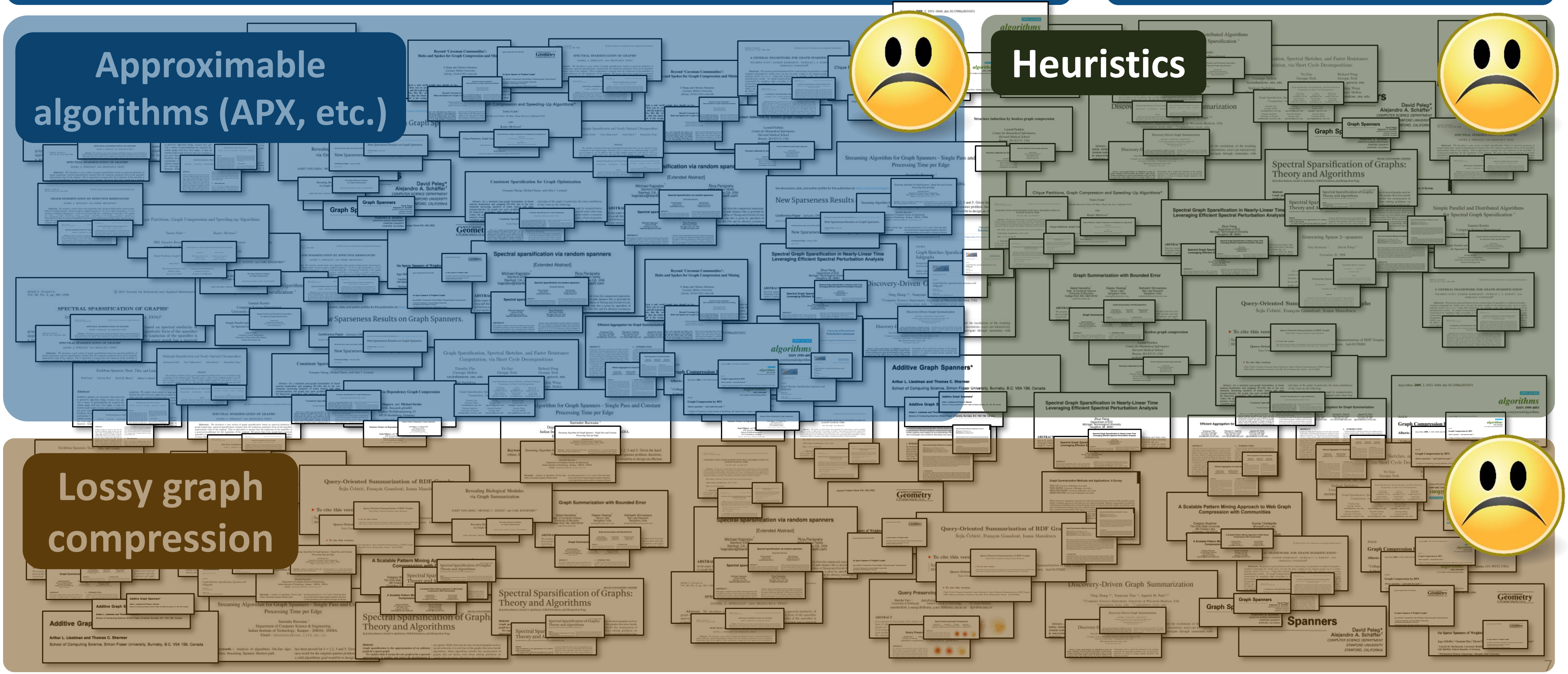
Heuristics

Lossy graph compression

Approximate Graph Processing: State & Challenges

We analyzed > 500 works and identified three classes of schemes...

...they all have problems



Approximable algorithms (APX, etc.)

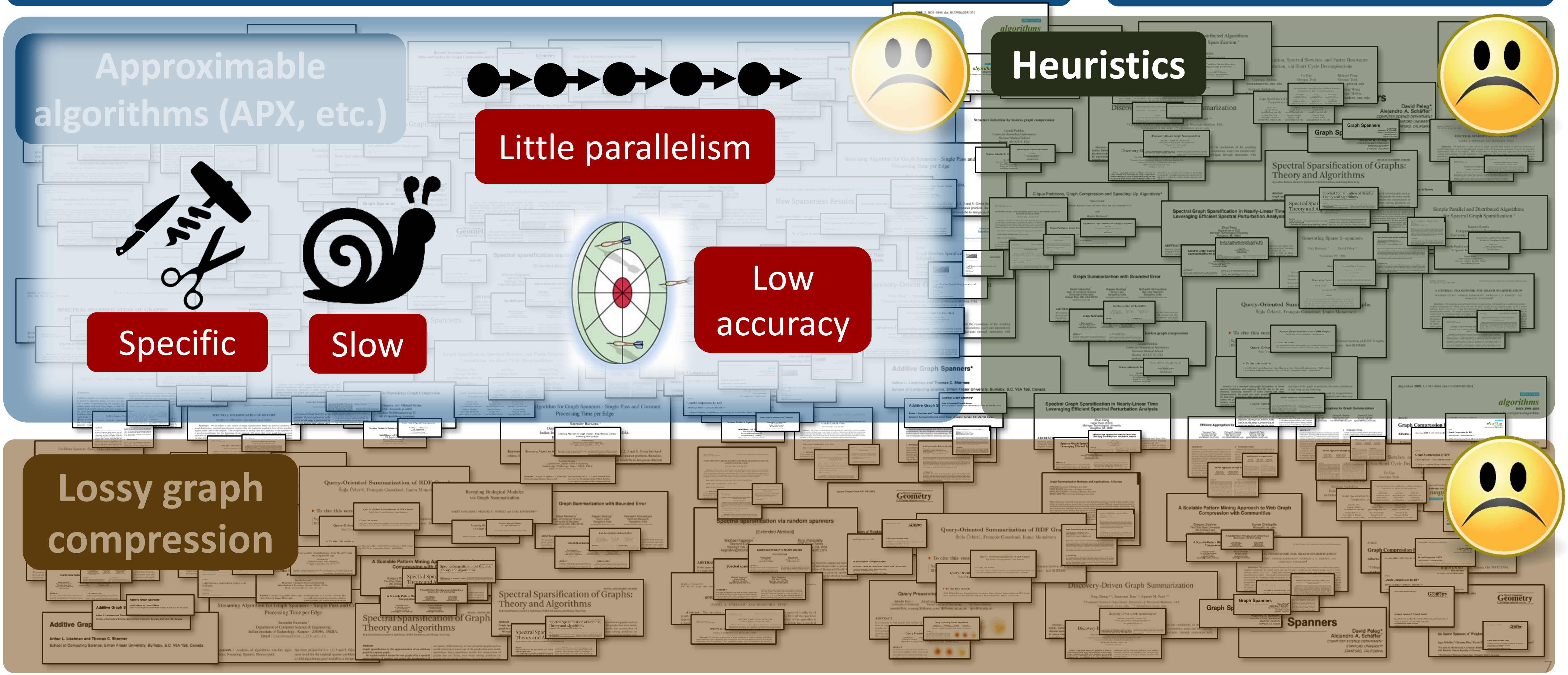
Heuristics

Lossy graph compression

Approximate Graph Processing: State & Challenges

We analyzed > 500 works and identified three classes of schemes...

...they all have problems



Approximable algorithms (APX, etc.)

Little parallelism

Specific

Slow

Low accuracy

Heuristics

Lossy graph compression

Approximate Graph Processing: State & Challenges

We analyzed > 500 works and identified three classes of schemes...

...they all have problems

Approximable algorithms (APX, etc.)

Little parallelism

Specific

Slow

Low accuracy

Heuristics

Specific

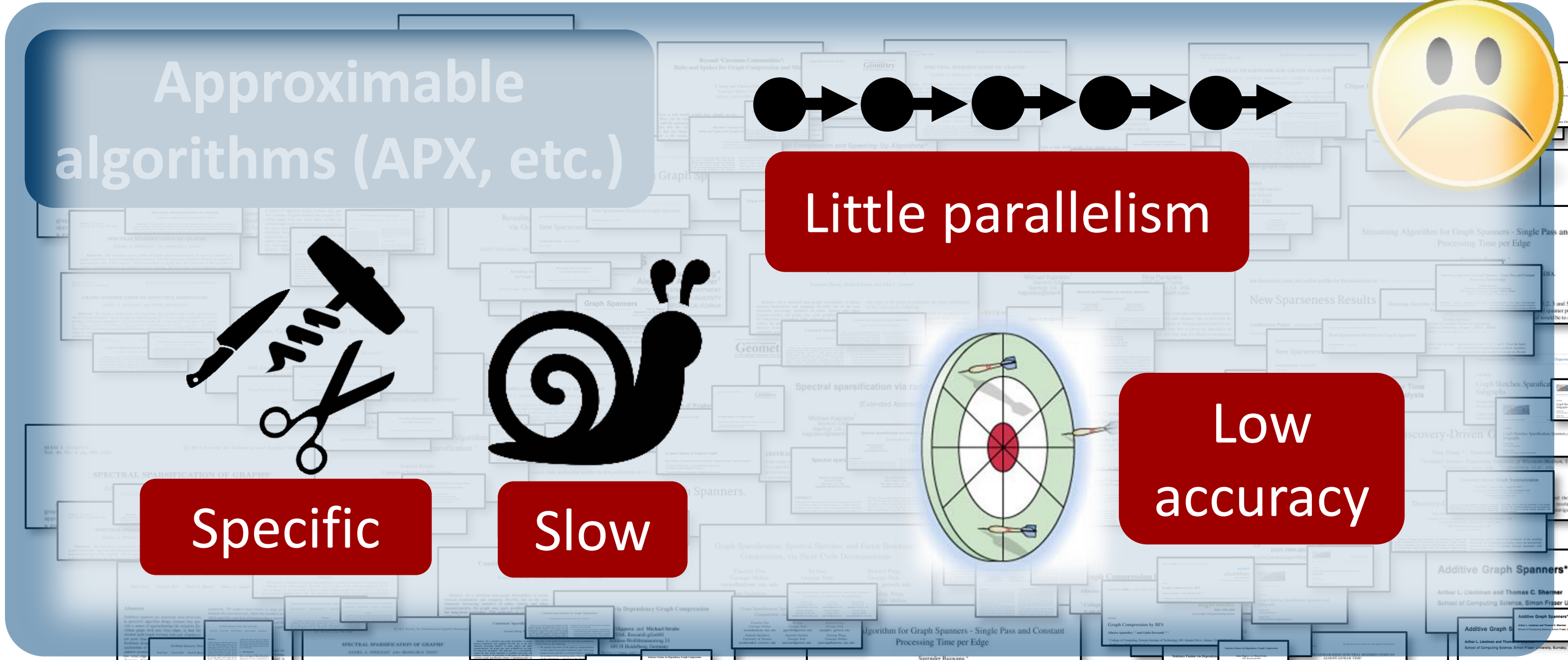
No/loose accuracy guarantees

Lossy graph compression

Approximate Graph Processing: State & Challenges

We analyzed > 500 works and identified three classes of schemes...

...they all have problems



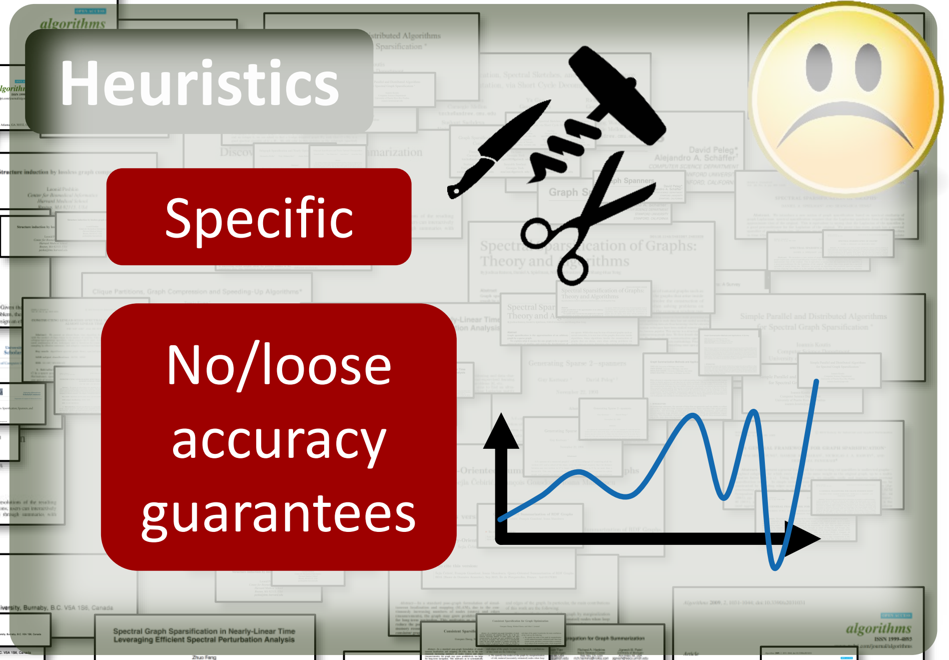
Approximable algorithms (APX, etc.)

Little parallelism

Specific

Slow

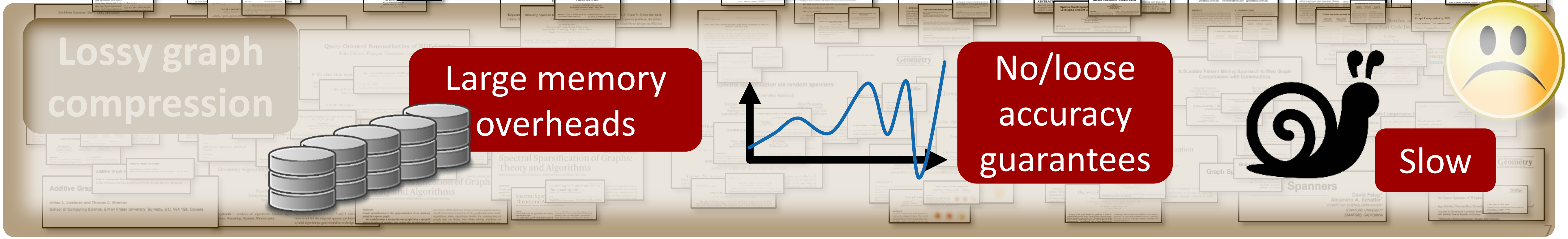
Low accuracy



Heuristics

Specific

No/loose accuracy guarantees



Lossy graph compression

Large memory overheads

No/loose accuracy guarantees

Slow

Approximate Graph Processing: Current Issues & Our Objectives

Approximate Graph Processing: Current Issues & Our Objectives

Little parallelism

Specific

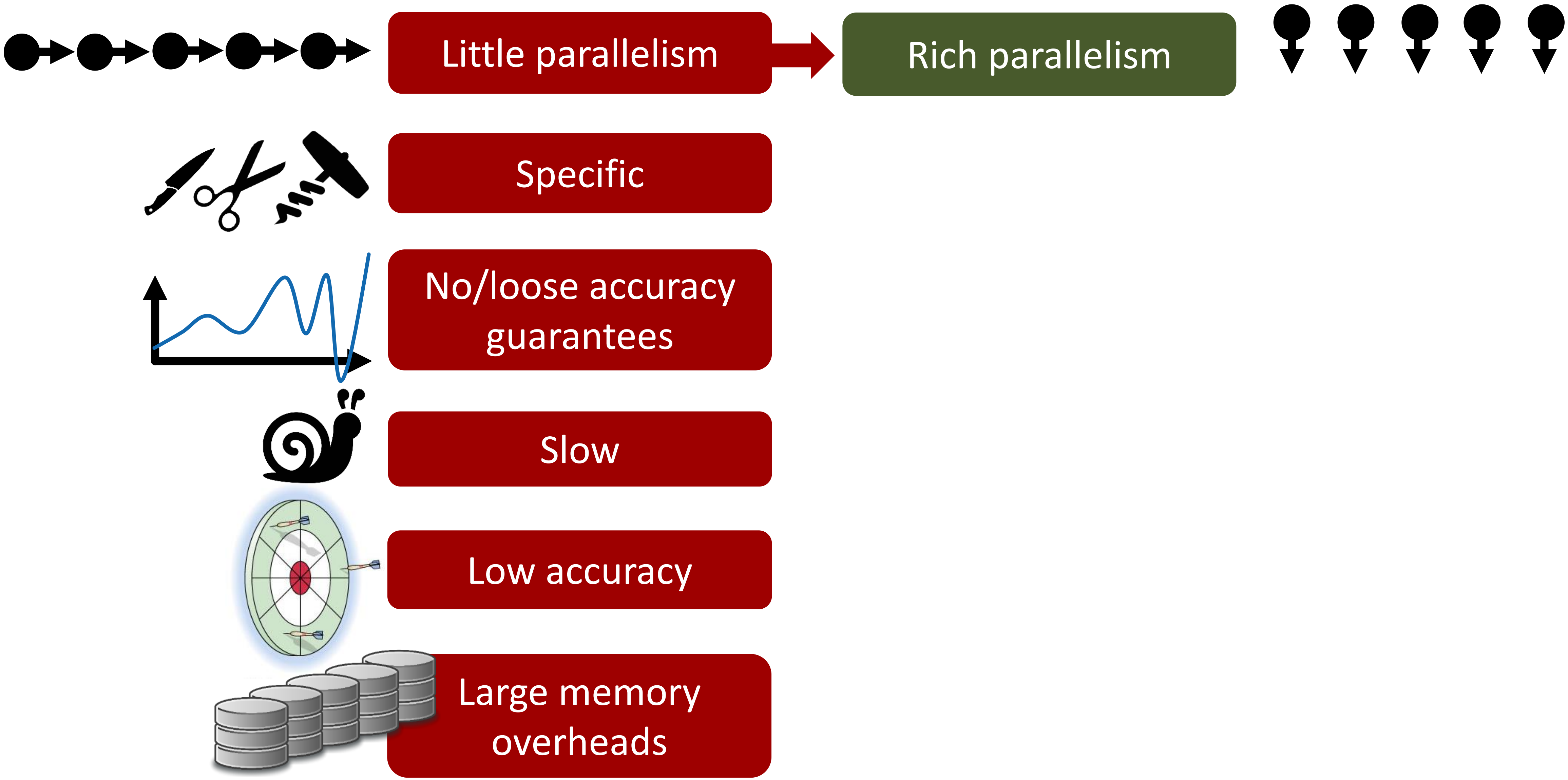
No/loose accuracy guarantees

Slow

Low accuracy

Large memory overheads

Approximate Graph Processing: Current Issues & Our Objectives



Little parallelism

Rich parallelism

Specific

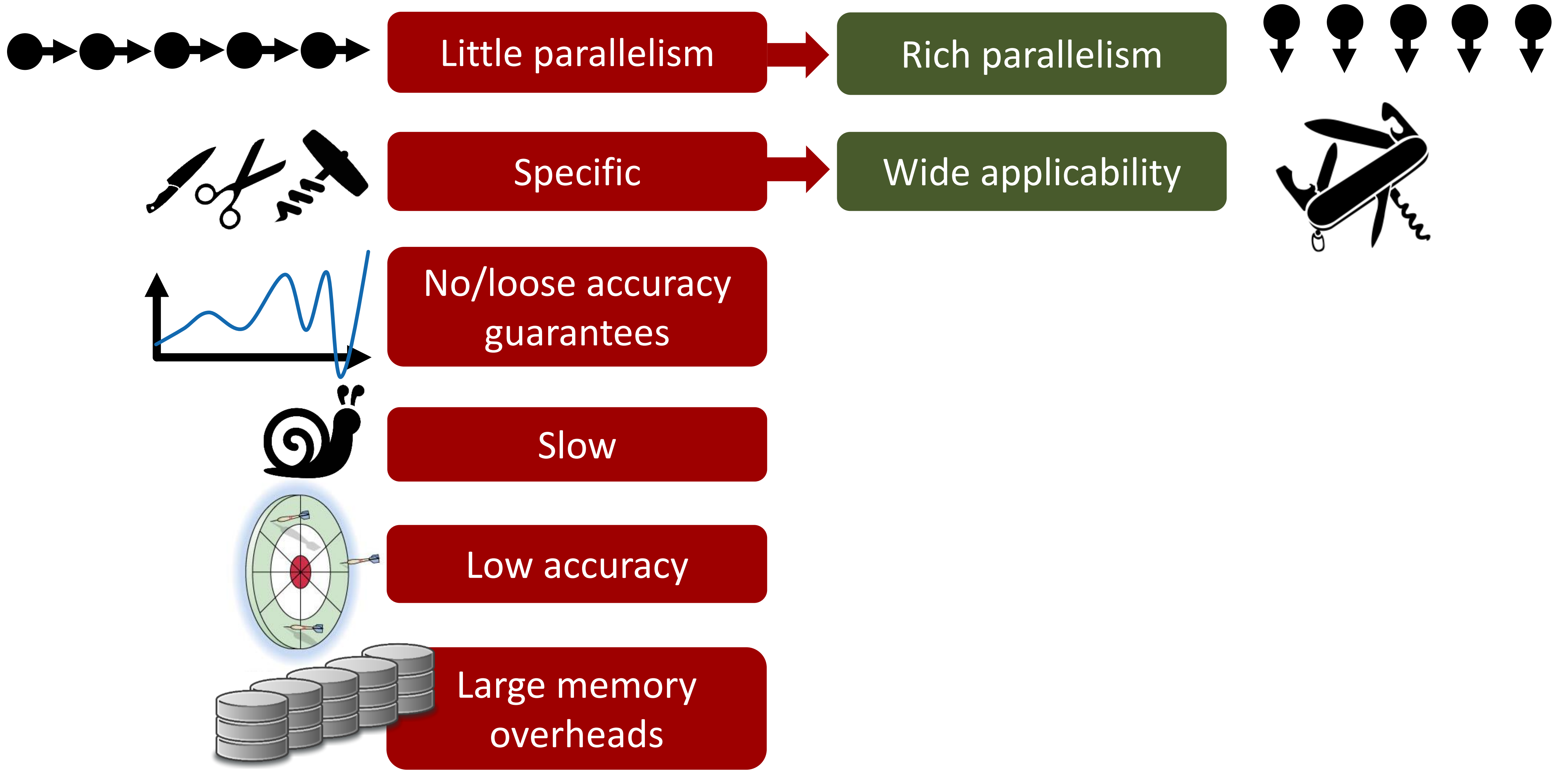
No/loose accuracy guarantees

Slow

Low accuracy

Large memory overheads

Approximate Graph Processing: Current Issues & Our Objectives



Little parallelism → Rich parallelism

Specific → Wide applicability

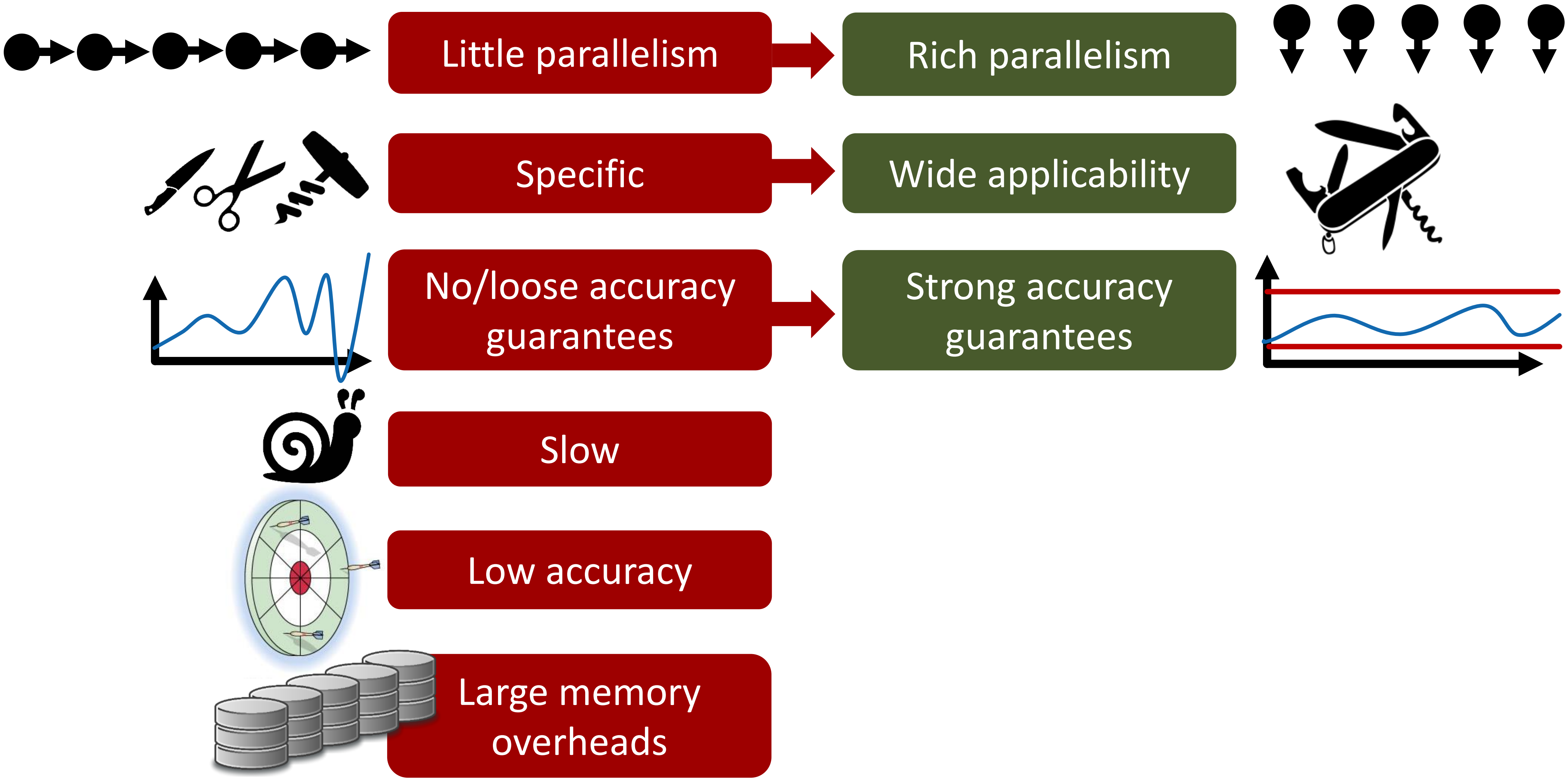
No/loose accuracy guarantees

Slow

Low accuracy

Large memory overheads

Approximate Graph Processing: Current Issues & Our Objectives



Little parallelism → Rich parallelism

Specific → Wide applicability

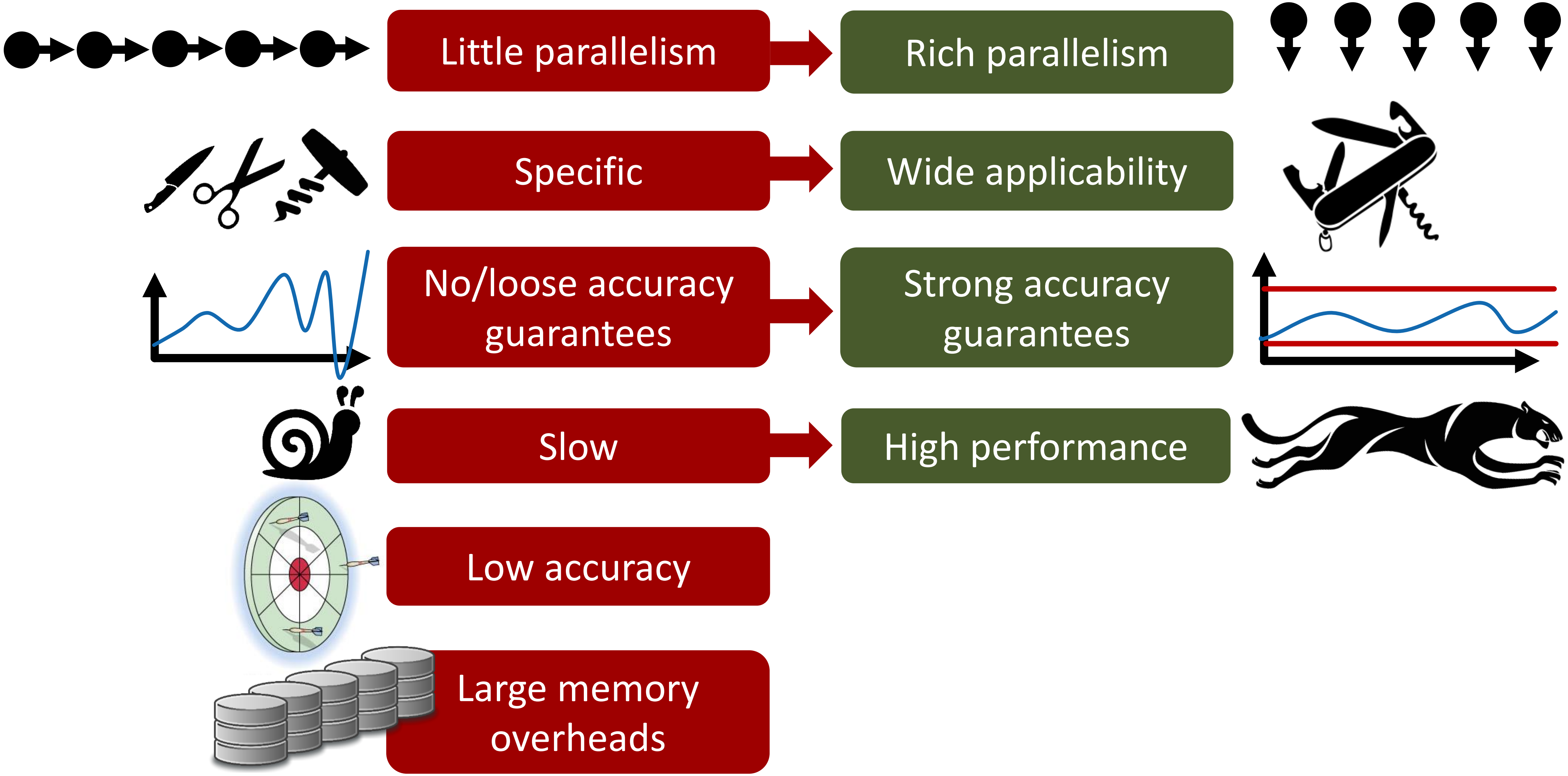
No/loose accuracy guarantees → Strong accuracy guarantees

Slow

Low accuracy

Large memory overheads

Approximate Graph Processing: Current Issues & Our Objectives



Little parallelism → Rich parallelism

Specific → Wide applicability

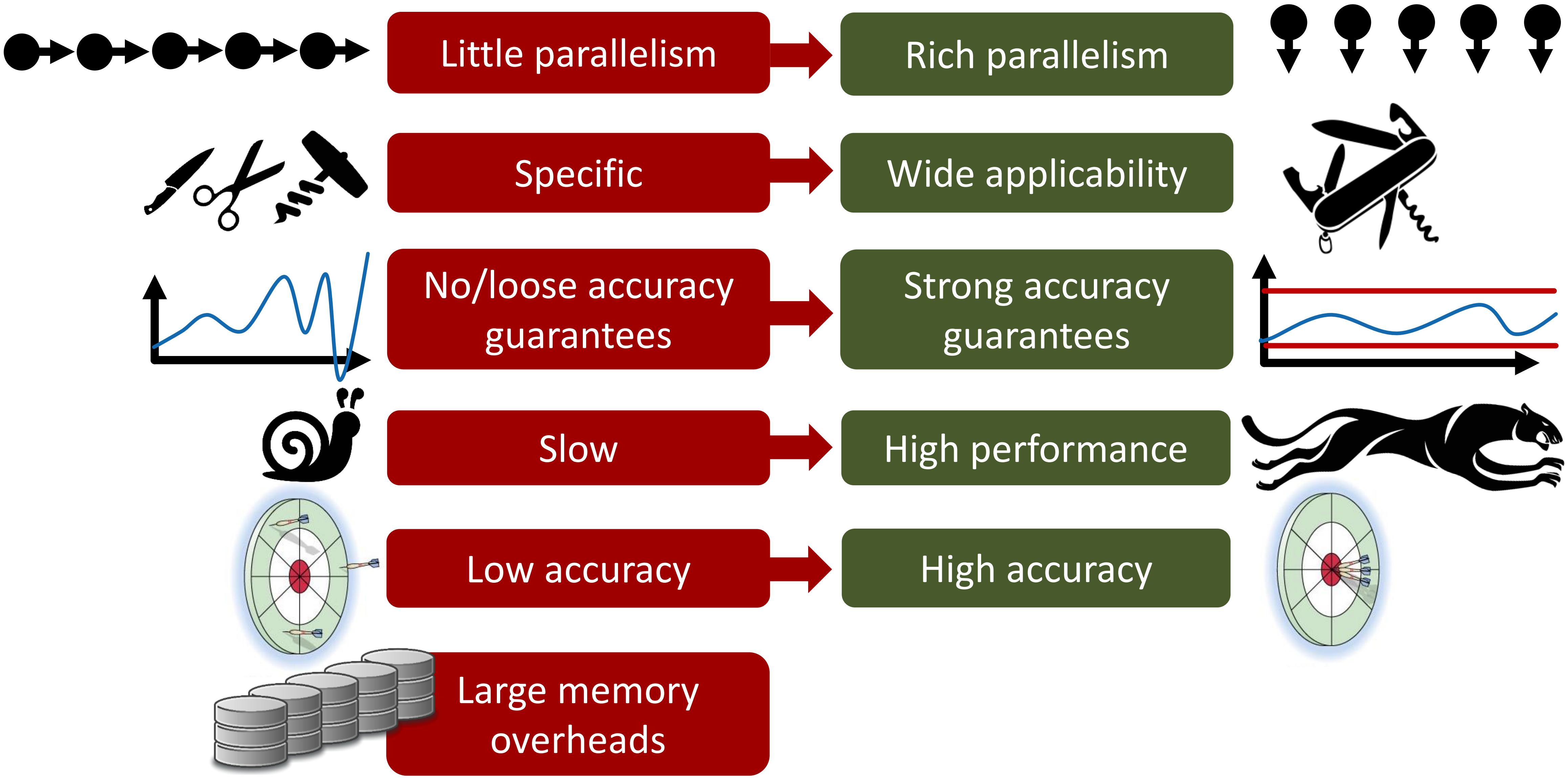
No/loose accuracy guarantees → Strong accuracy guarantees

Slow → High performance

Low accuracy

Large memory overheads

Approximate Graph Processing: Current Issues & Our Objectives



Little parallelism → Rich parallelism

Specific → Wide applicability

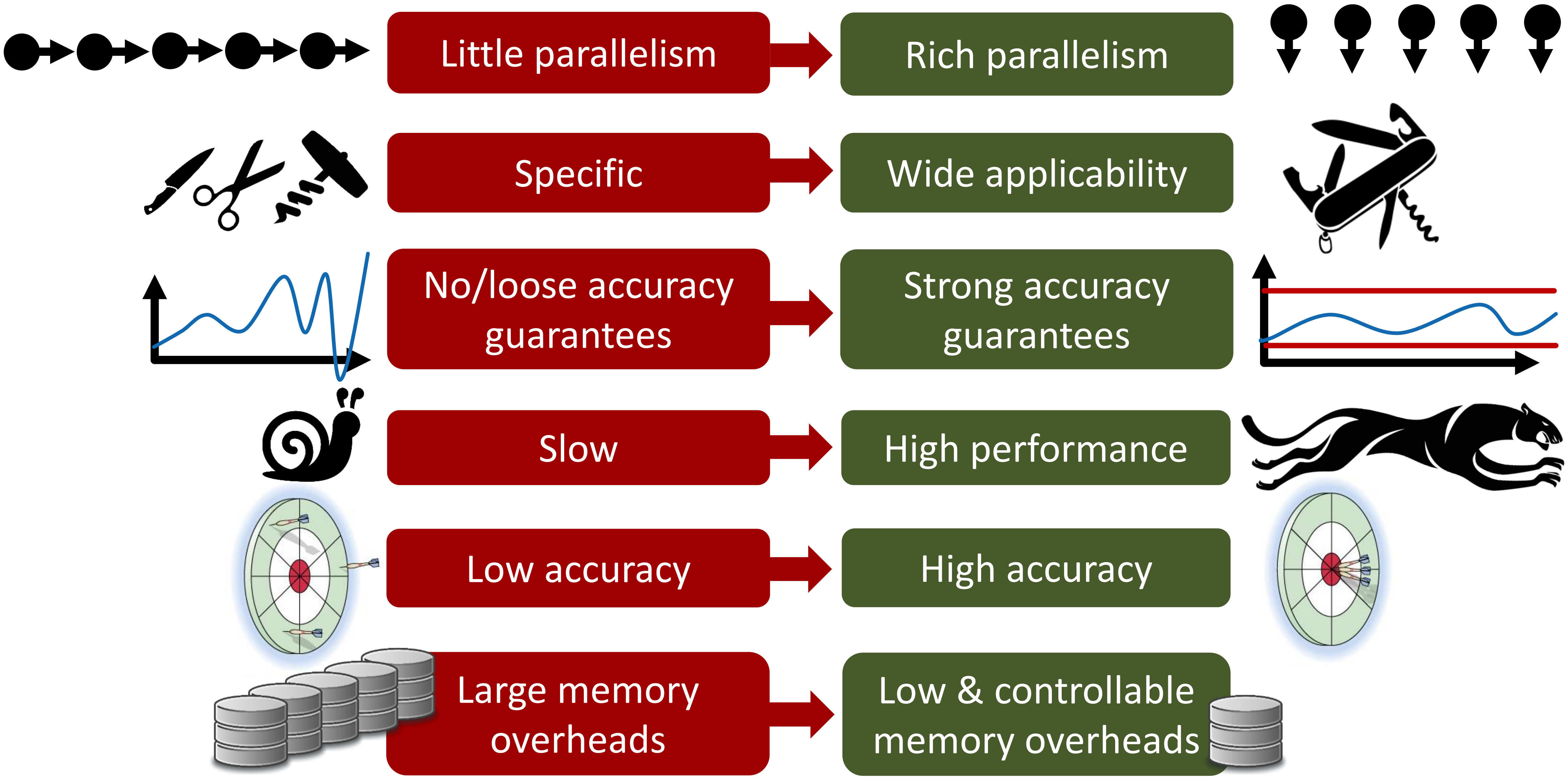
No/loose accuracy guarantees → Strong accuracy guarantees

Slow → High performance

Low accuracy → High accuracy

Large memory overheads

Approximate Graph Processing: Current Issues & Our Objectives



Little parallelism → Rich parallelism

Specific → Wide applicability

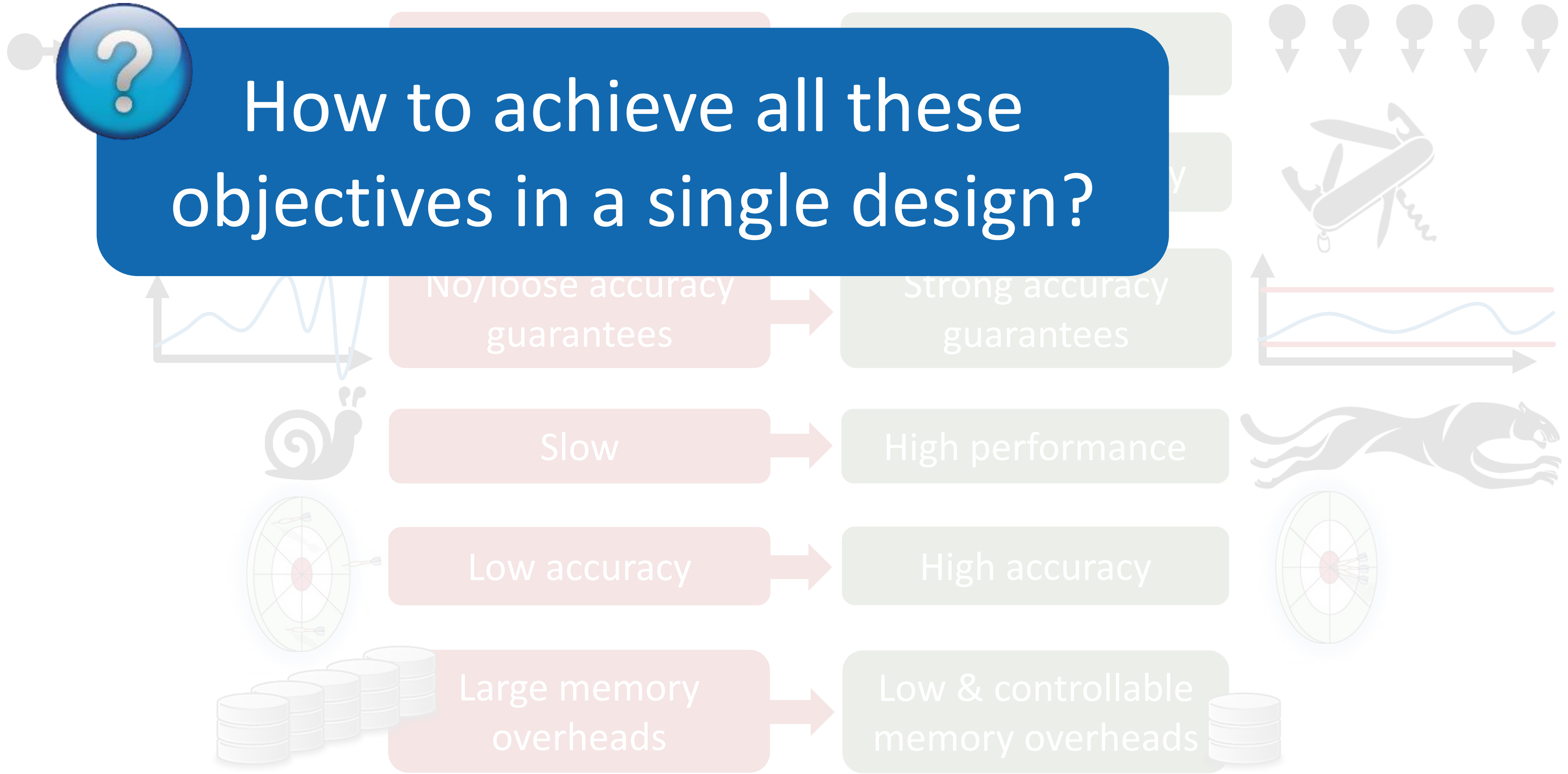
No/loose accuracy guarantees → Strong accuracy guarantees

Slow → High performance

Low accuracy → High accuracy

Large memory overheads → Low & controllable memory overheads

Approximate Graph Processing: Current Issues & Our Objectives



How to achieve all these objectives in a single design?

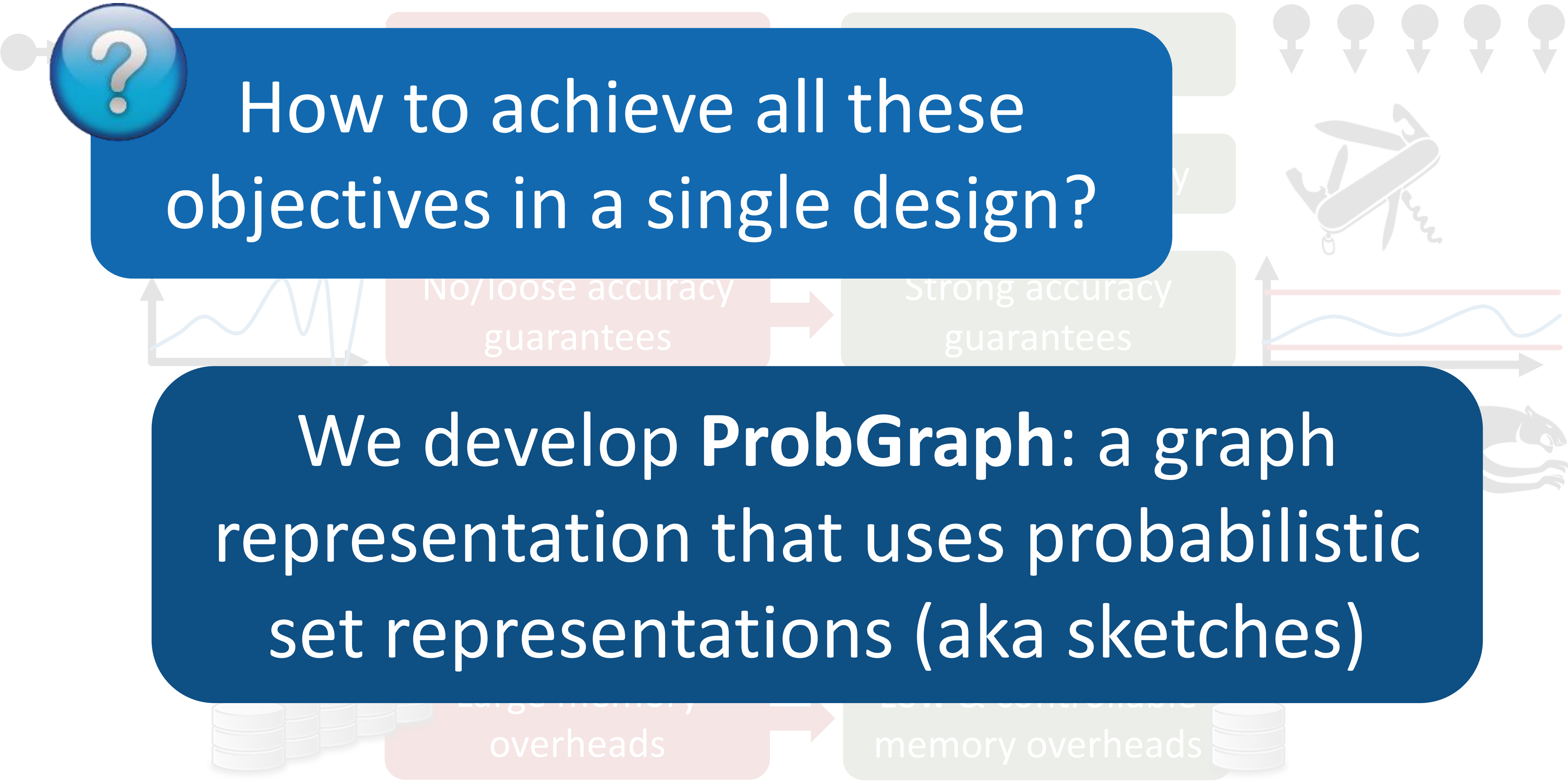
No/loose accuracy guarantees → Strong accuracy guarantees

Slow → High performance

Low accuracy → High accuracy

Large memory overheads → Low & controllable memory overheads

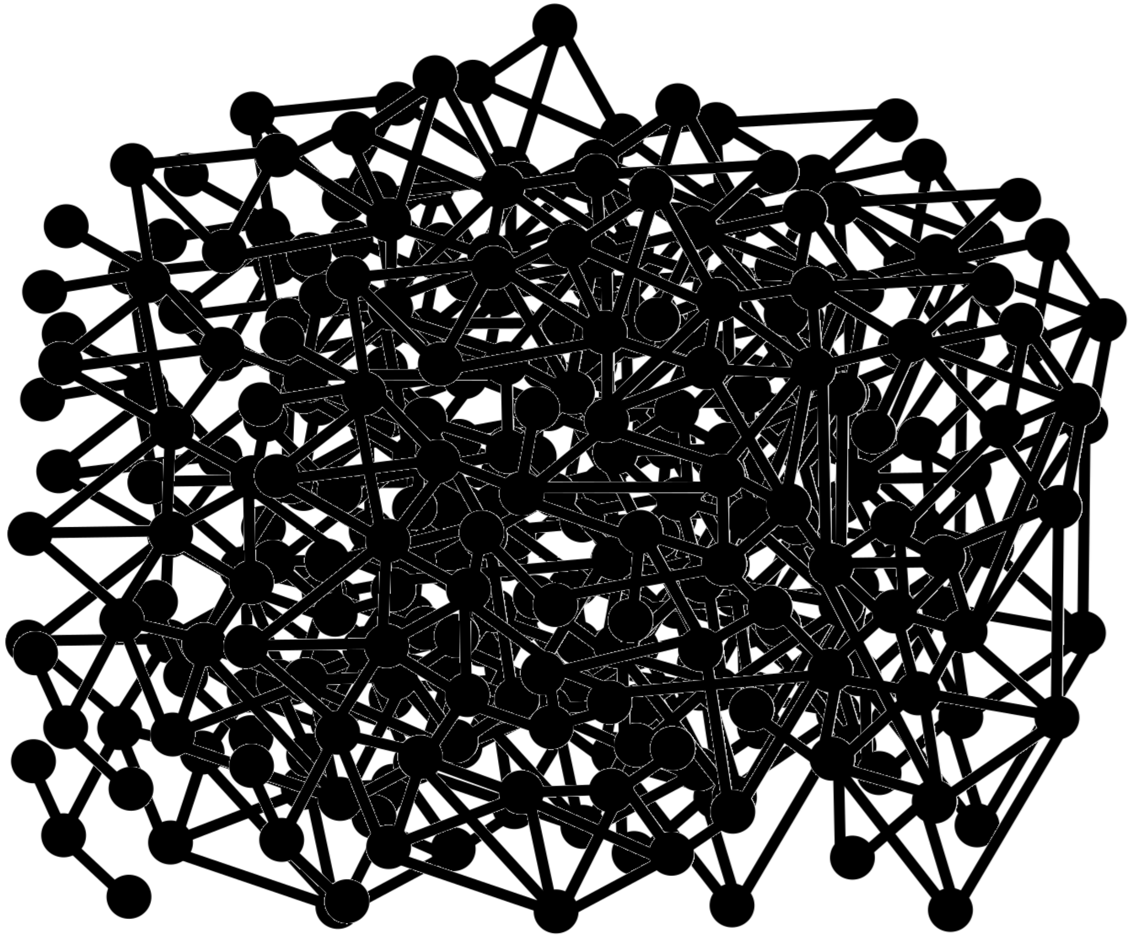
Approximate Graph Processing: Current Issues & Our Objectives



How to achieve all these objectives in a single design?

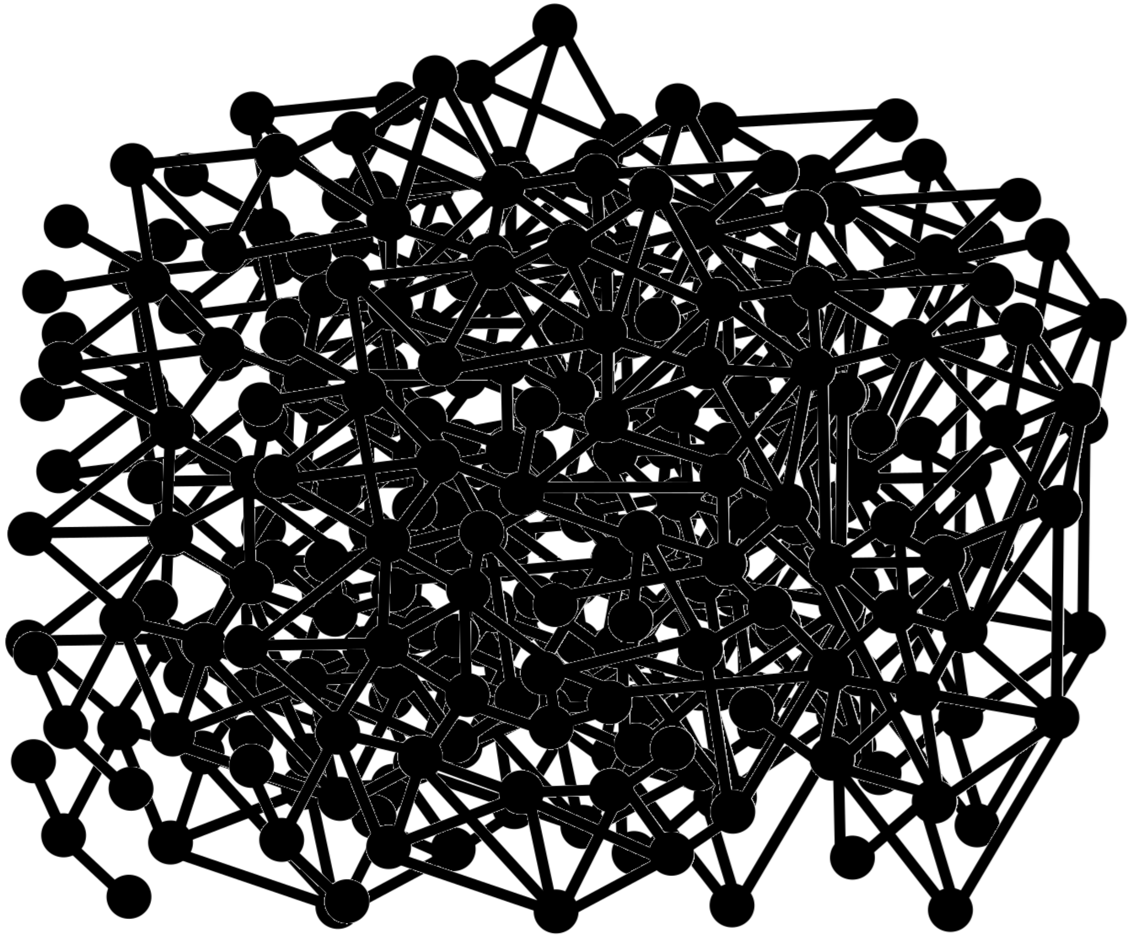
We develop **ProbGraph**: a graph representation that uses probabilistic set representations (aka sketches)

High-Level Approach Taken in ProbGraph



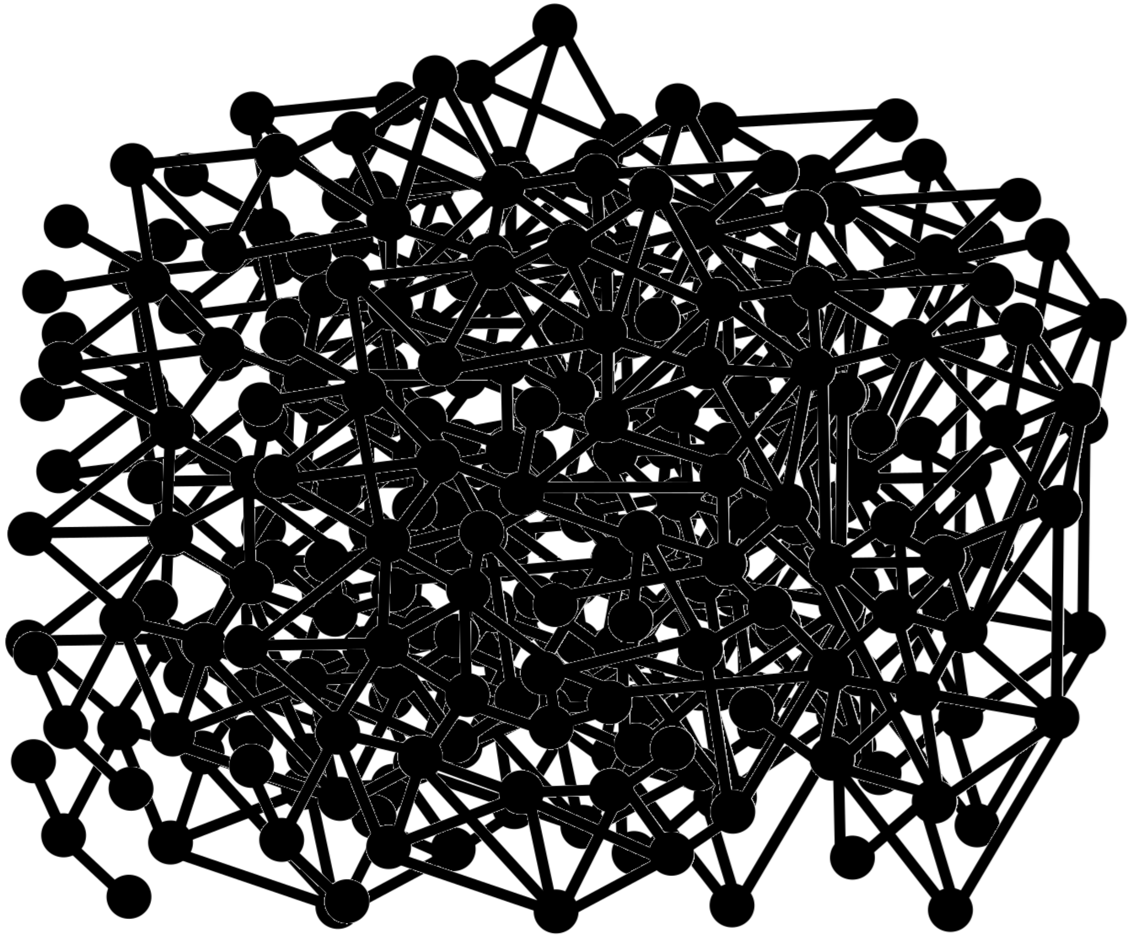
High-Level Approach Taken in ProbGraph

Keep the original graph



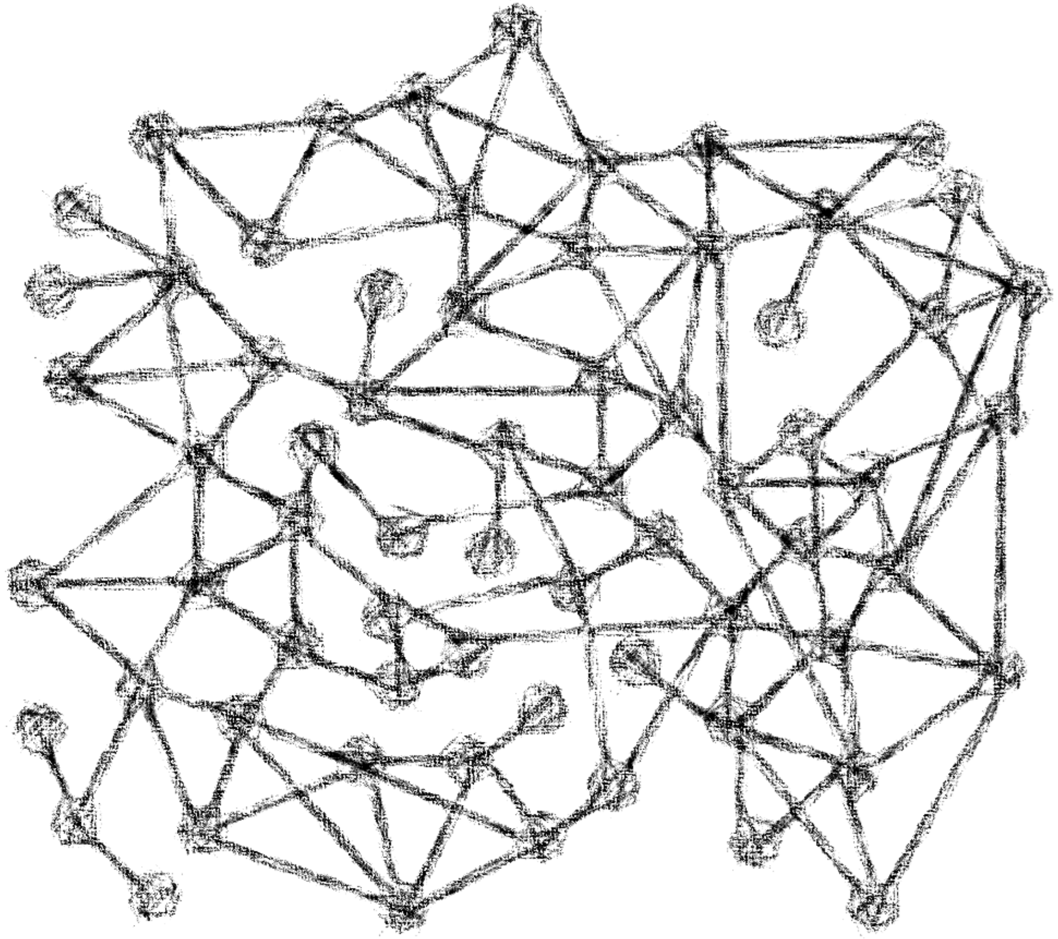
High-Level Approach Taken in ProbGraph

Keep the original graph



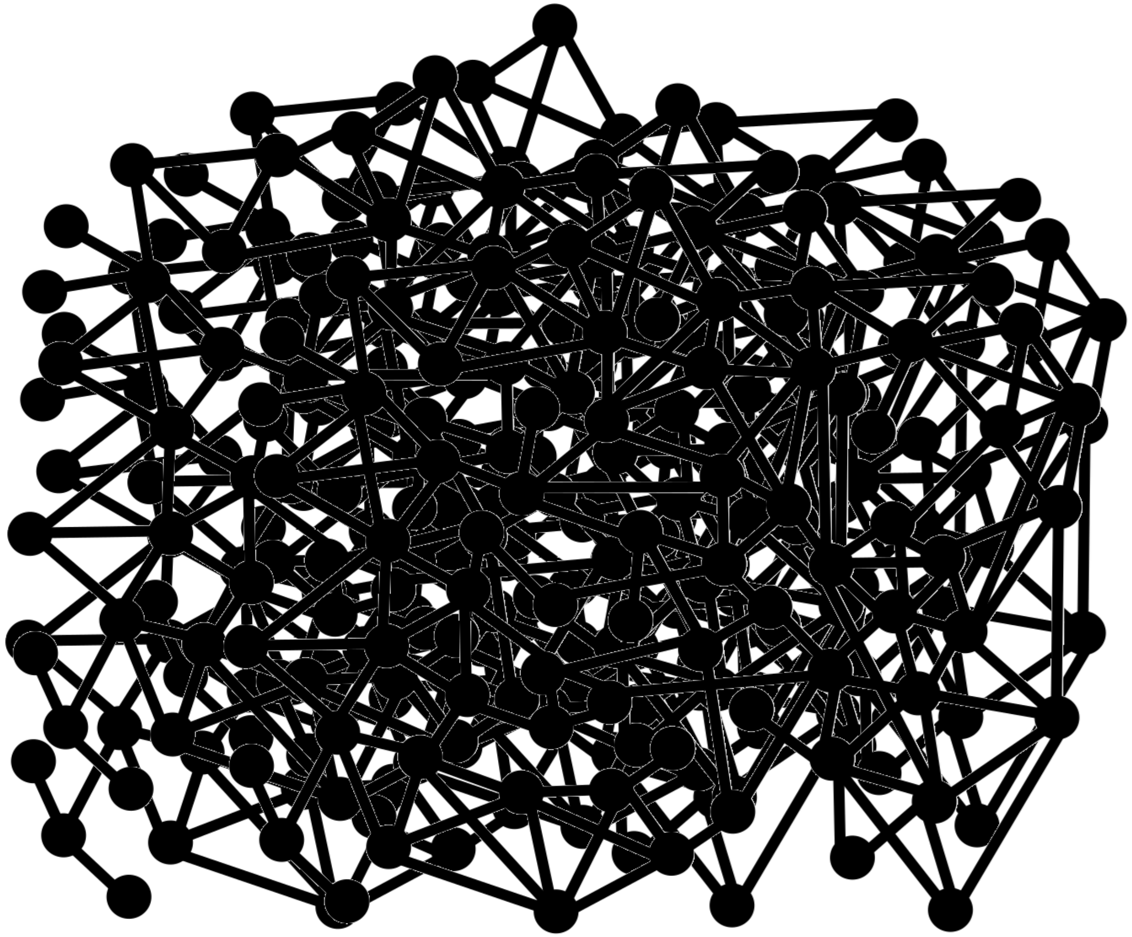
+

Maintain a very small “sketch” of a graph



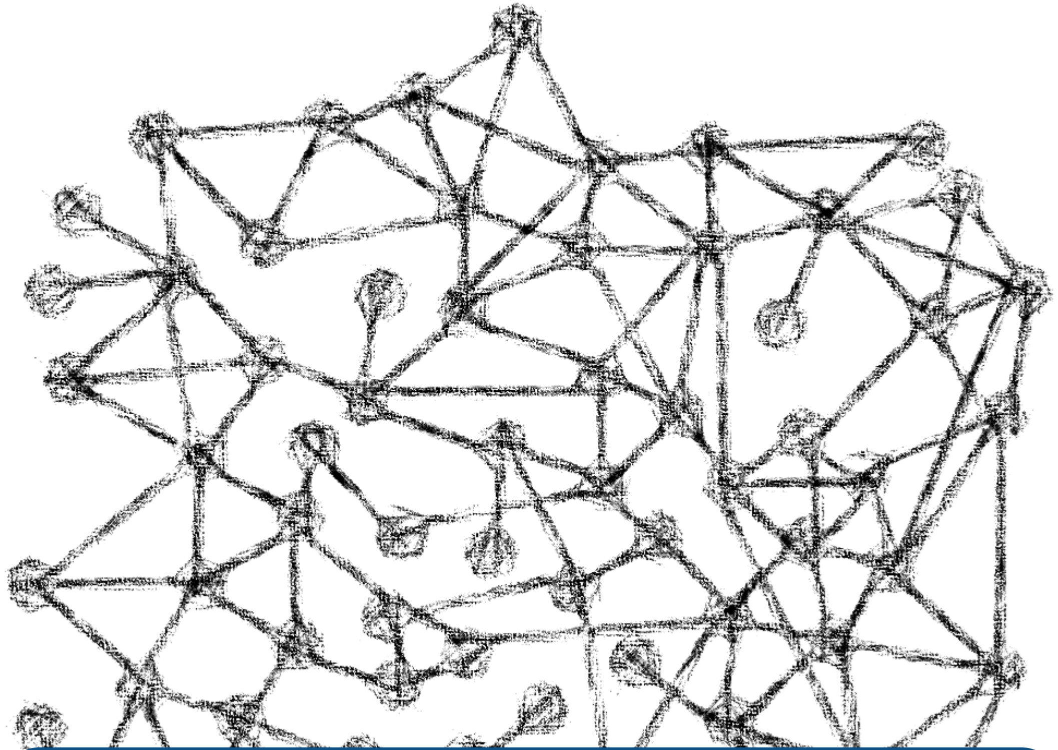
High-Level Approach Taken in ProbGraph

Keep the original graph



+

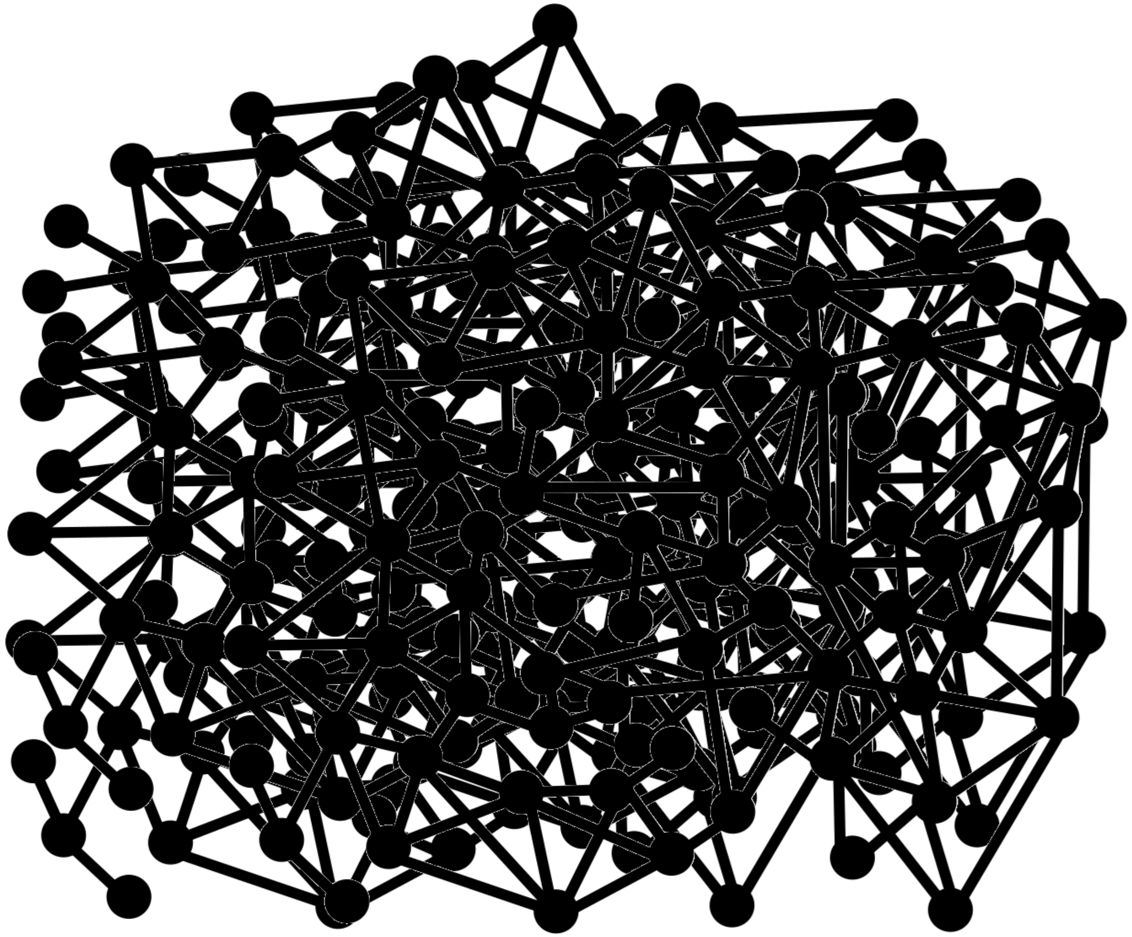
Maintain a very small "sketch" of a graph



Use the sketch to answer performance critical queries

High-Level Approach Taken in ProbGraph

Keep the original graph



+

Maintain a very small “sketch” of a graph

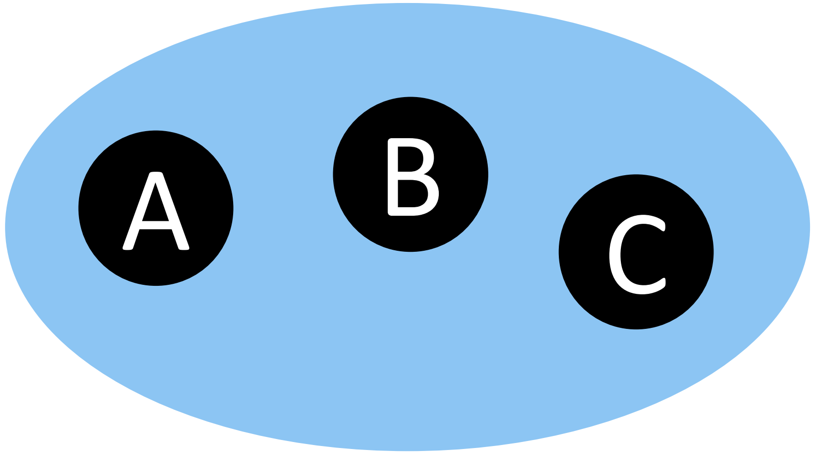
What design to use for the sketch, to satisfy all the goals?

Use the sketch to answer performance critical queries

ProbGraph key idea: Use probabilistic set representations (set sketches)

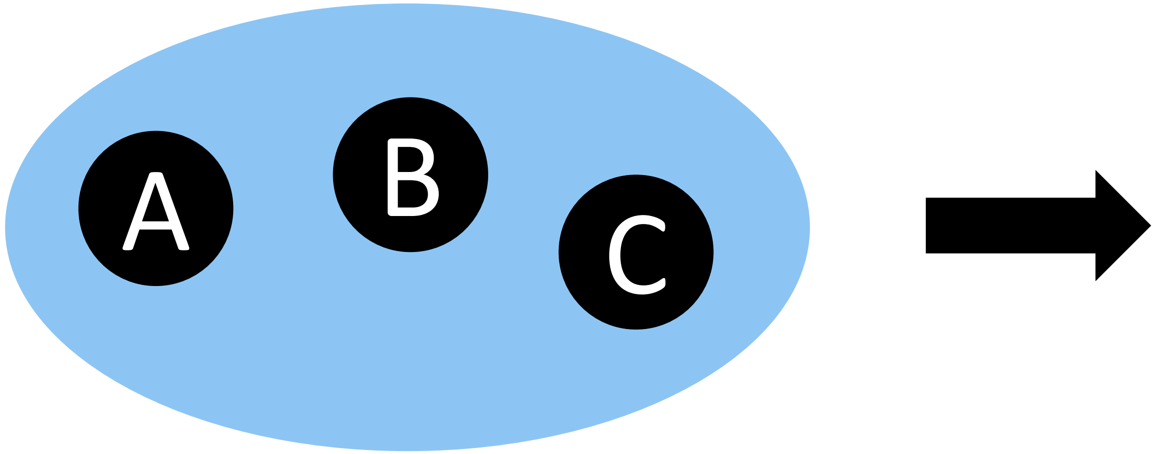
ProbGraph key idea: Use probabilistic set representations (set sketches)

A set = {A, B, C}



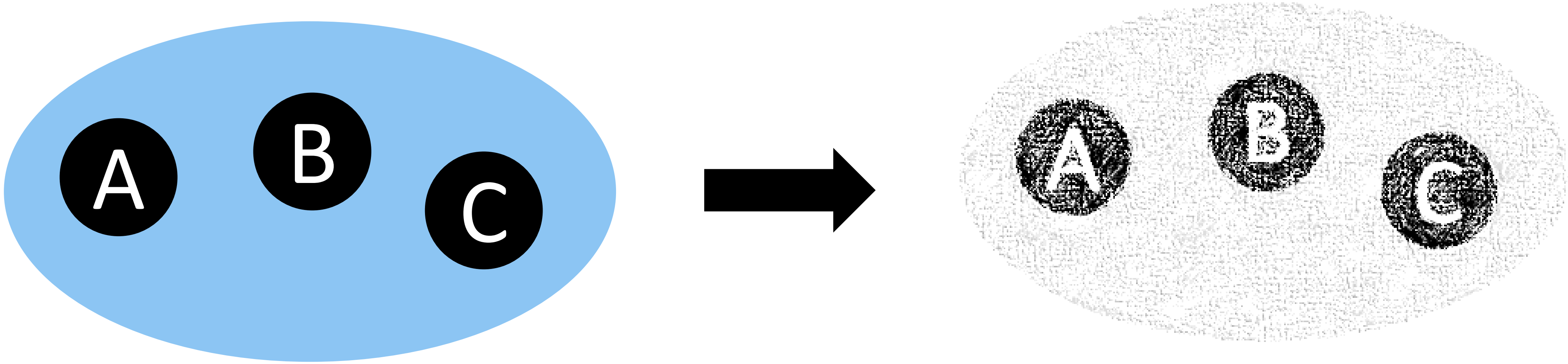
ProbGraph key idea: Use probabilistic set representations (set sketches)

A set = {A, B, C}



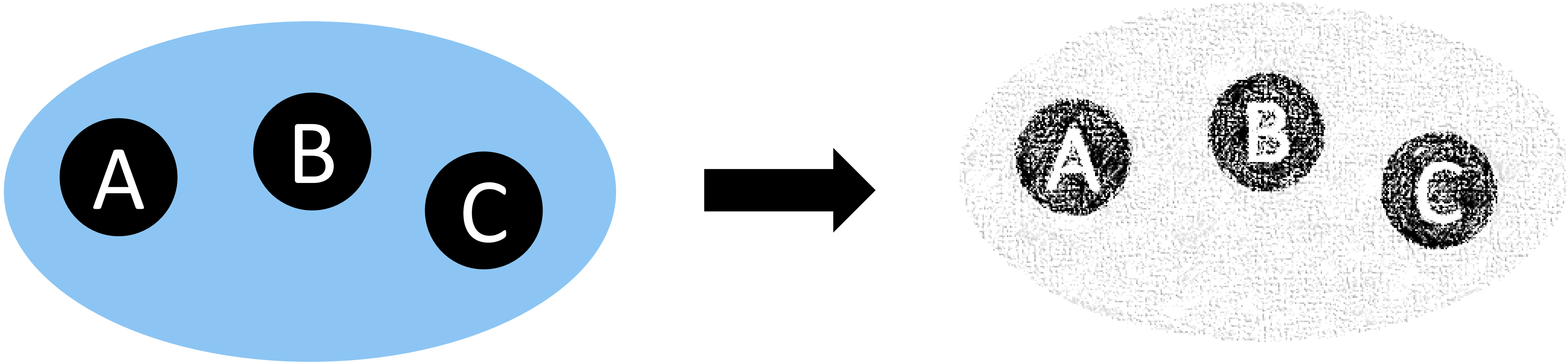
ProbGraph key idea: Use probabilistic set representations (set sketches)

A set = {A, B, C}



ProbGraph key idea: Use probabilistic set representations (set sketches)

A set = {A, B, C}



Bloom filters (BF) [1]

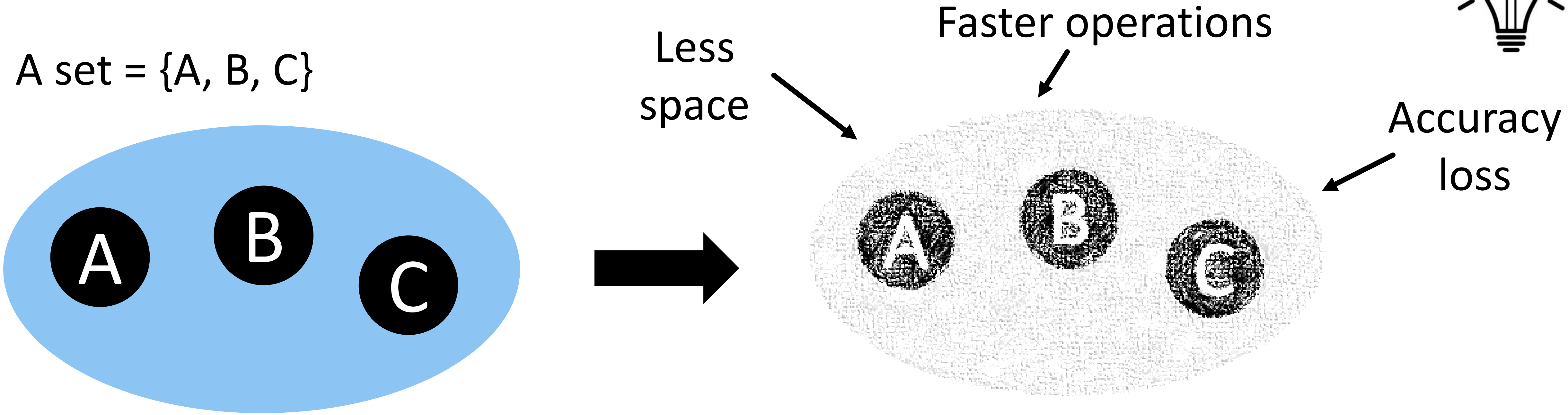
MinHash [2]

K Minimum Values [3]

[1] B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors”, CACM, 1970.
[2] A. Z. Broder, “On the resemblance and containment of documents”, IEEE SEQUENCES, 1997.
[3] Z. Bar-Yossef et al., “Counting distinct elements in a data stream”, in RANDOM, 2002.

ProbGraph key idea: Use probabilistic set representations (set sketches)

A set = {A, B, C}



Bloom filters (BF) [1]

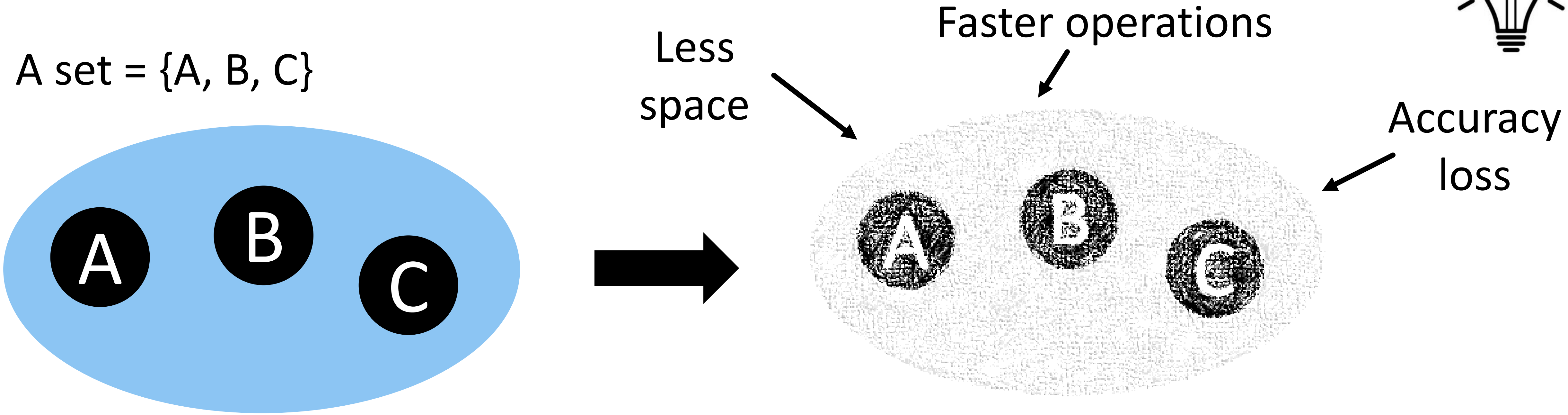
MinHash [2]

K Minimum Values [3]

[1] B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors”, CACM, 1970.
[2] A. Z. Broder, “On the resemblance and containment of documents”, IEEE SEQUENCES, 1997.
[3] Z. Bar-Yossef et al., “Counting distinct elements in a data stream”, in RANDOM, 2002.

ProbGraph key idea: Use probabilistic set representations (set sketches)

A set = {A, B, C}



Less space

Faster operations

Accuracy loss

Bloom filters (BF) [1]

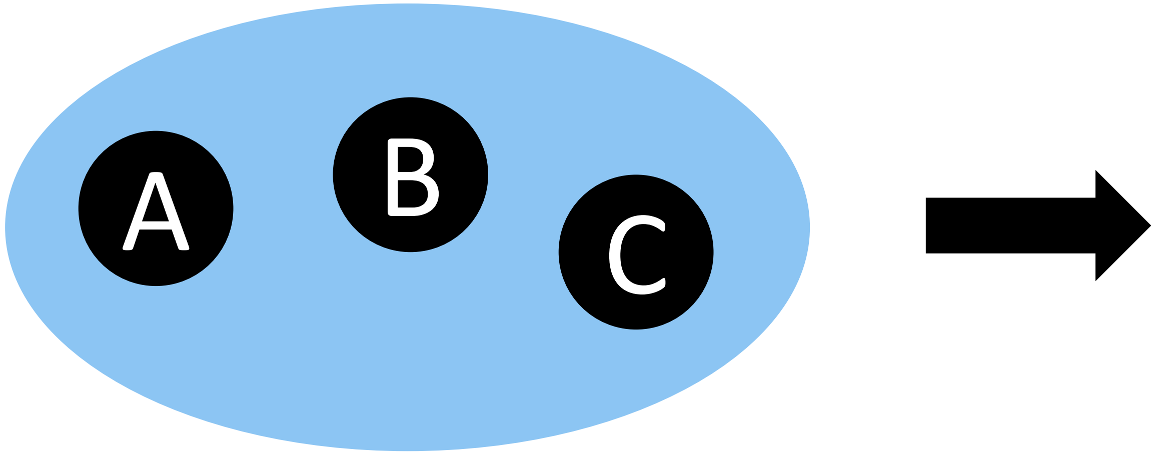
MinHash [2]

K Minimum Values [3]

[1] B. H. Bloom, “Space/time trade-offs in hash coding with allowable errors”, CACM, 1970.
[2] A. Z. Broder, “On the resemblance and containment of documents”, IEEE SEQUENCES, 1997.
[3] Z. Bar-Yossef et al., “Counting distinct elements in a data stream”, in RANDOM, 2002.

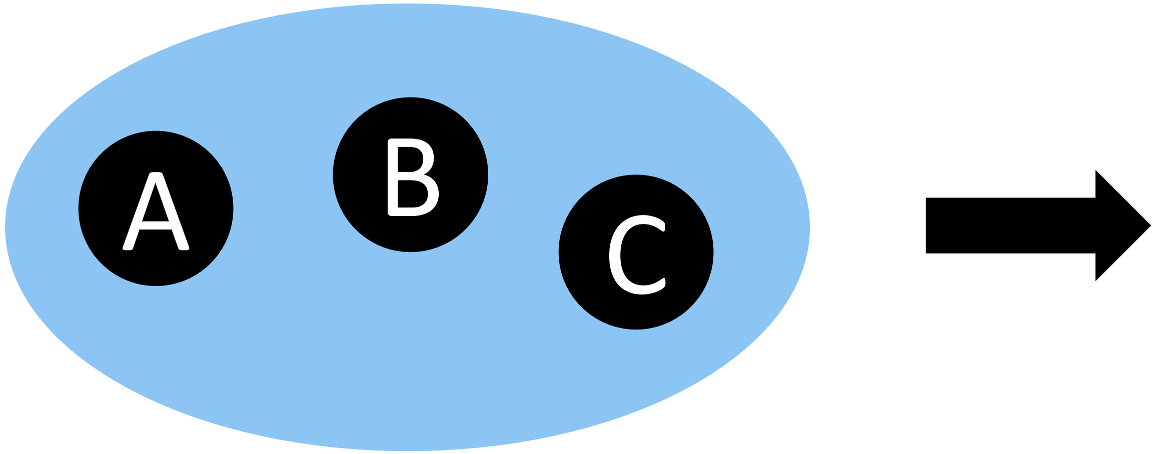
Bloom Filters for Graph Mining

A set = {A, B, C}



Bloom Filters for Graph Mining

A set = {A, B, C}

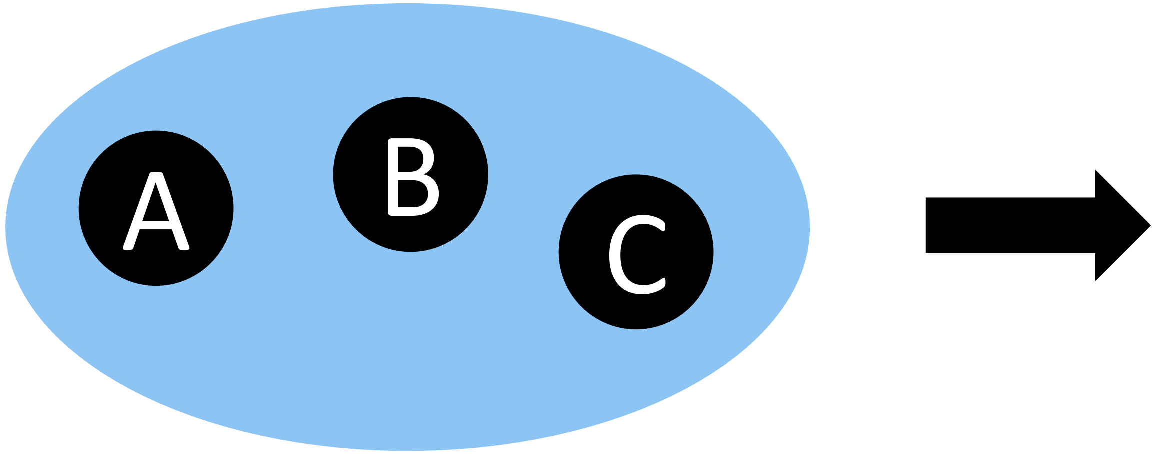


Bloom filter $\mathcal{B}_X$ of X

Bitvector of size B_X [bits]

Bloom Filters for Graph Mining

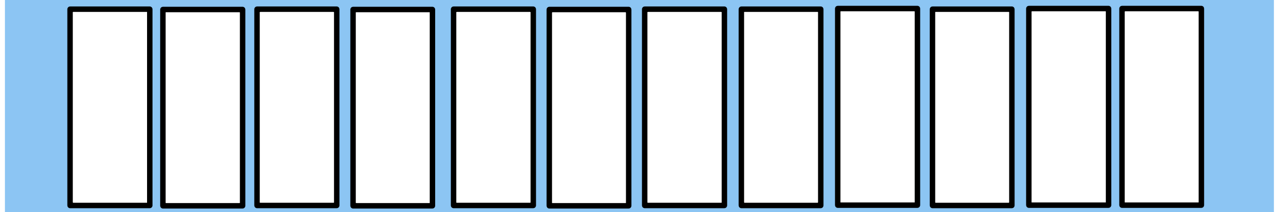
A set = {A, B, C}



Bloom filter $\mathcal{B}_X$ of X

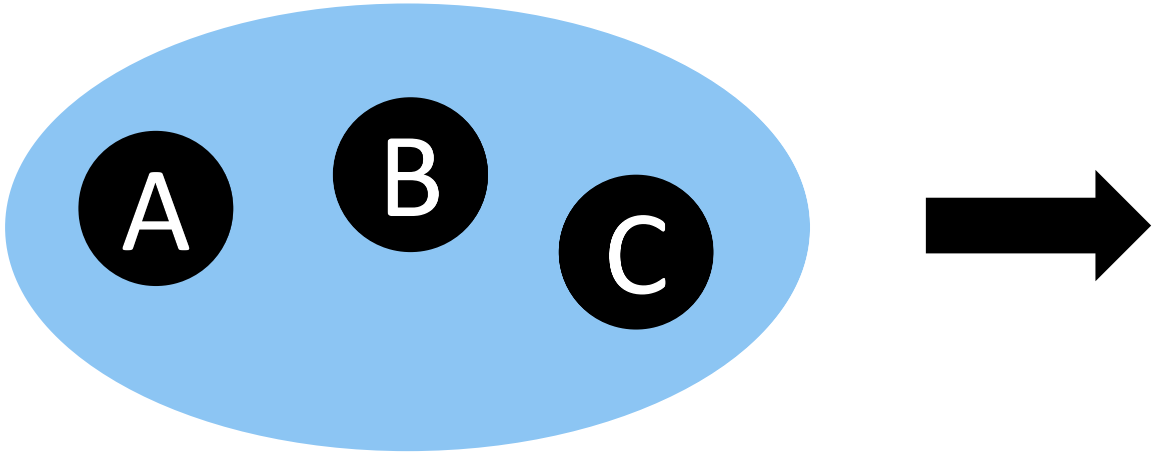
Bitvector of size B_X [bits]

$B_X = 12$



Bloom Filters for Graph Mining

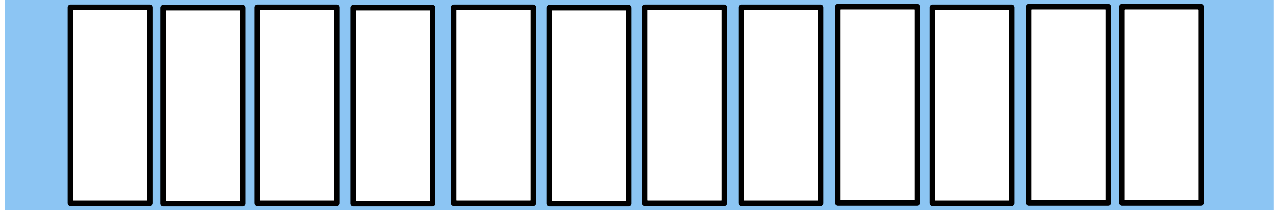
A set = {A, B, C}



Bloom filter $\mathcal{B}_X$ of X

Bitvector of size B_X [bits]

$B_X = 12$

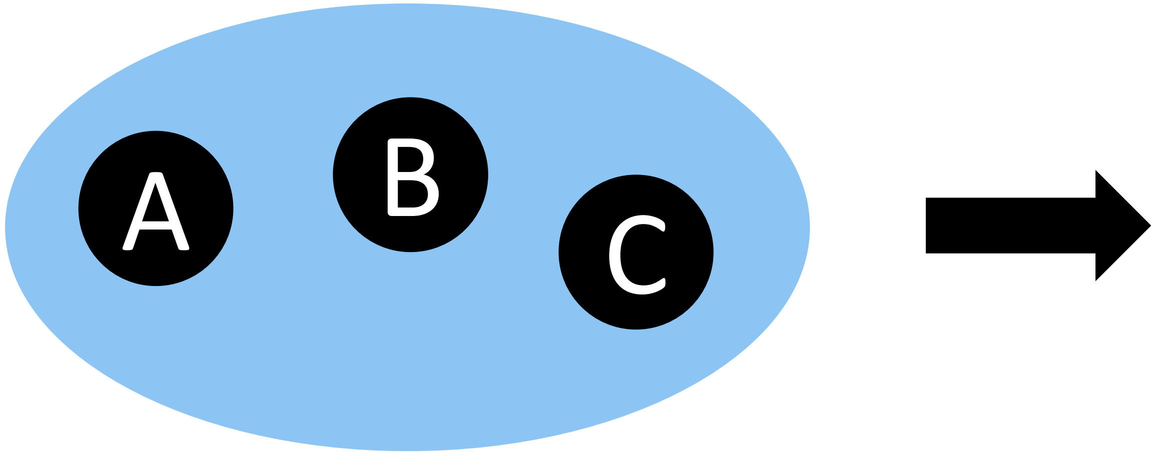


Hash functions $h_1, \ldots, h_b$

$h_i : X \rightarrow \{1, \ldots, B_X\}$

Bloom Filters for Graph Mining

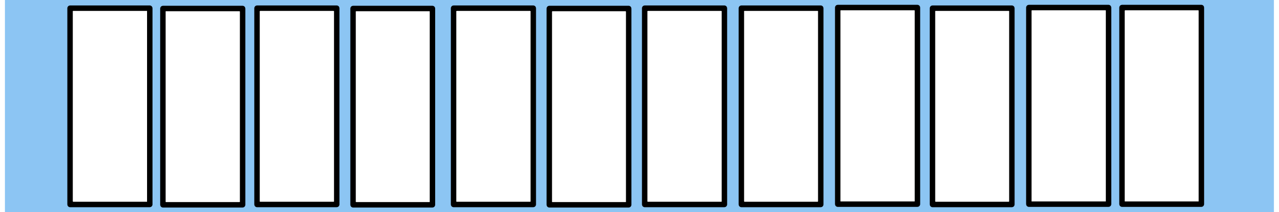
A set = {A, B, C}



Bloom filter $\mathcal{B}_X$ of X

Bitvector of size B_X [bits]

$B_X = 12$



Hash functions $h_1, ..., h_b$

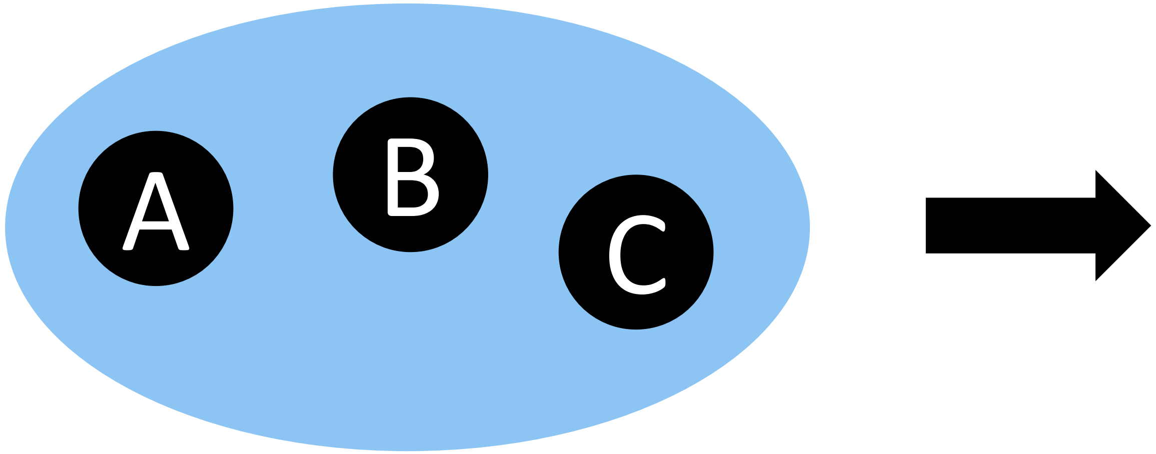
$h_i : X \rightarrow \{1, ..., B_X\}$

b = 2

$h_2, h_1 : X \rightarrow \{1, ..., 12\}$

Bloom Filters for Graph Mining

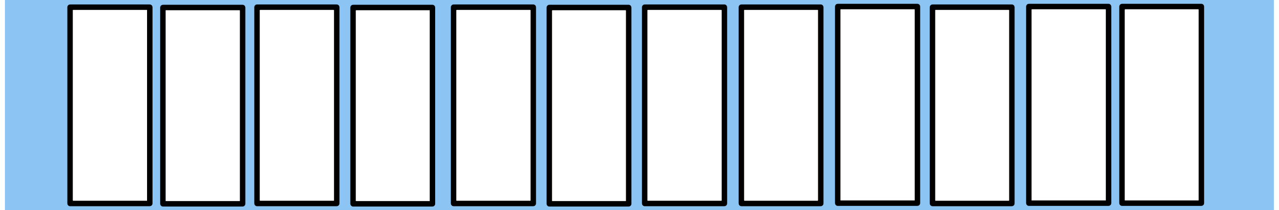
A set = {A, B, C}



Bloom filter $\mathcal{B}_X$ of X

Bitvector of size B_X [bits]

$B_X = 12$



Hash functions $h_1, \ldots, h_b$

$h_i : X \rightarrow \{1, \ldots, B_X\}$

$b = 2$

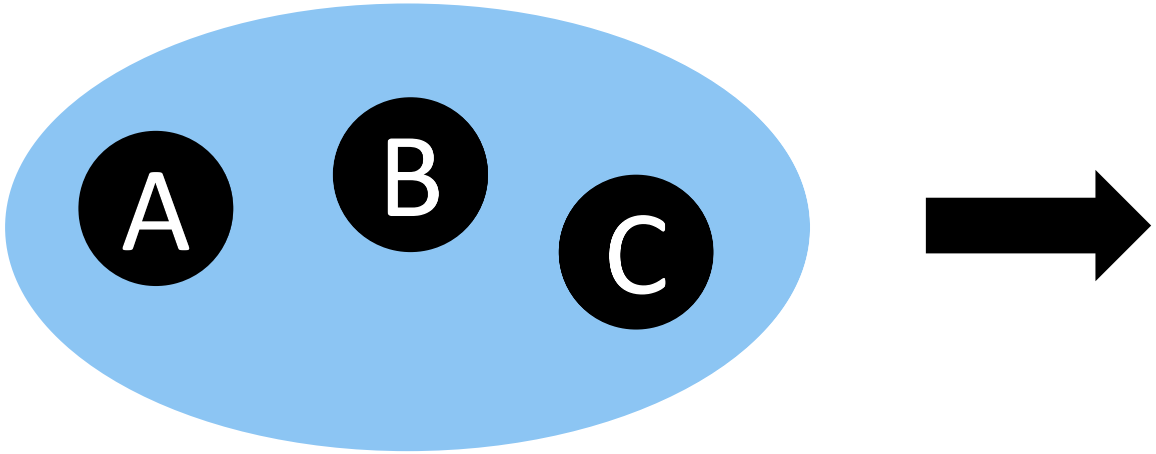
$h_2, h_1 : X \rightarrow \{1, \ldots, 12\}$

$h_1($A$) = 3$

$h_2($A$) = 5$

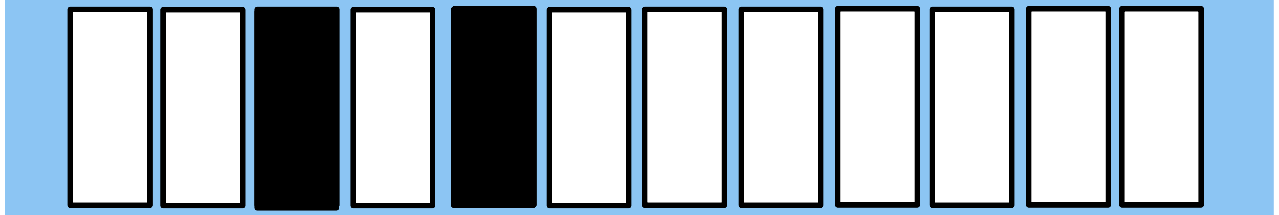
Bloom Filters for Graph Mining

A set = {A, B, C}



Bloom filter $\mathcal{B}_X$ of X

Bitvector of size B_X [bits] $B_X = 12$



Hash functions $h_1, \ldots, h_b$

$h_i : X \rightarrow \{1, \ldots, B_X\}$

$b = 2$

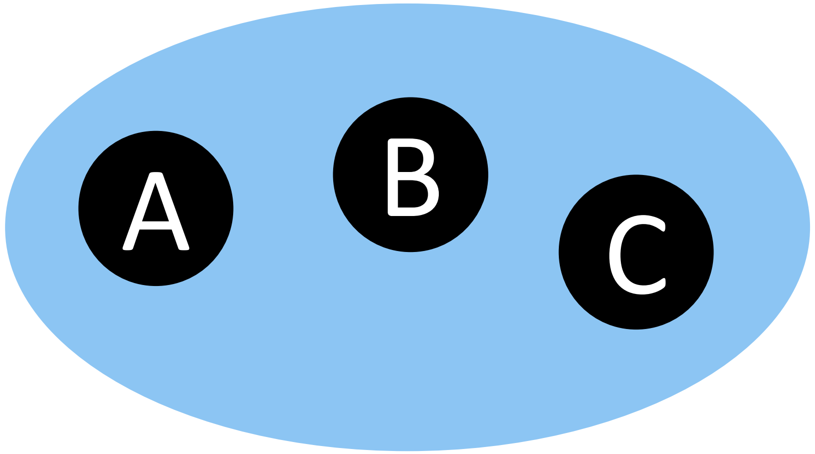
$h_2, h_1 : X \rightarrow \{1, \ldots, 12\}$

h_1(A) = 3

h_2(A) = 5

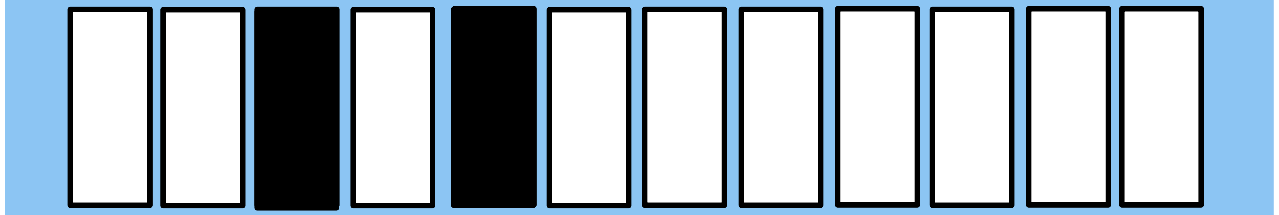
Bloom Filters for Graph Mining

A set = {A, B, C}



Bloom filter $\mathcal{B}_X$ of X

Bitvector of size B_X [bits] $B_X = 12$



Hash functions $h_1, \ldots, h_b$
$h_i : X \rightarrow \{1, \ldots, B_X\}$

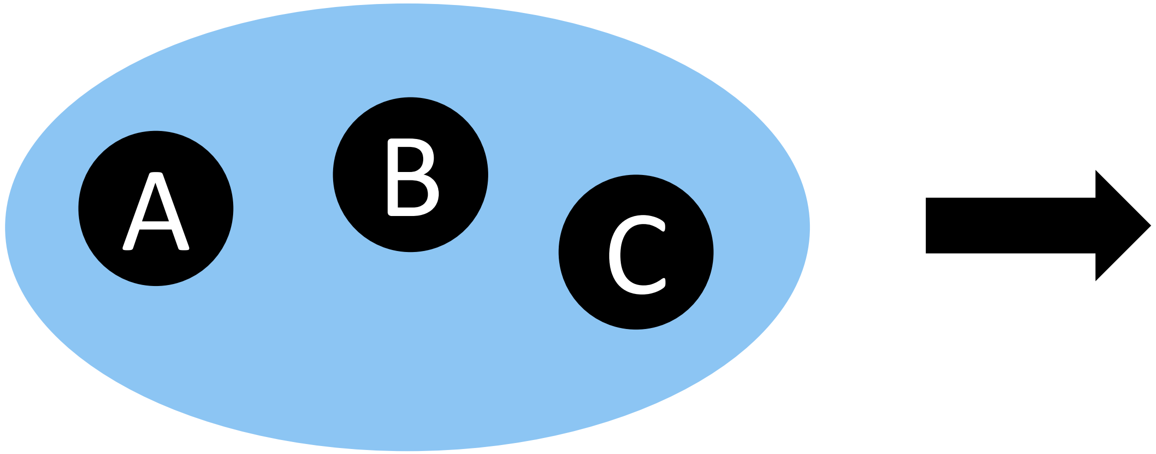
$b = 2$
$h_2, h_1 : X \rightarrow \{1, \ldots, 12\}$

h_1(A) = 3 h_1(B) = 1

h_2(A) = 5 h_2(B) = 8

Bloom Filters for Graph Mining

A set = {A, B, C}



Bloom filter $\mathcal{B}_X$ of X

Bitvector of size B_X [bits]

$B_X = 12$

■□■□■□□■□□□□

Hash functions $h_1, \ldots, h_b$

$h_i : X \rightarrow \{1, \ldots, B_X\}$

$b = 2$

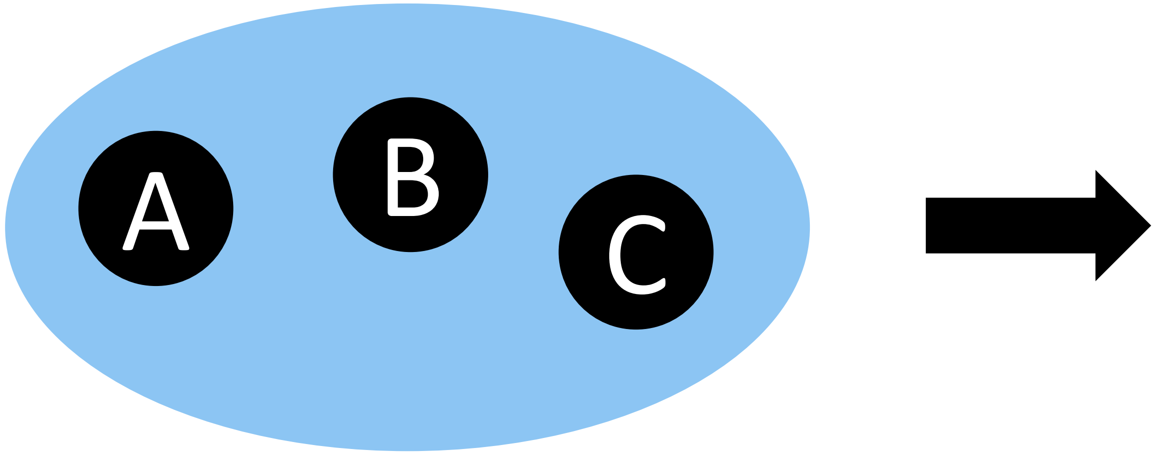
$h_2, h_1 : X \rightarrow \{1, \ldots, 12\}$

$h_1(A) = 3$ $h_1(B) = 1$

$h_2(A) = 5$ $h_2(B) = 8$

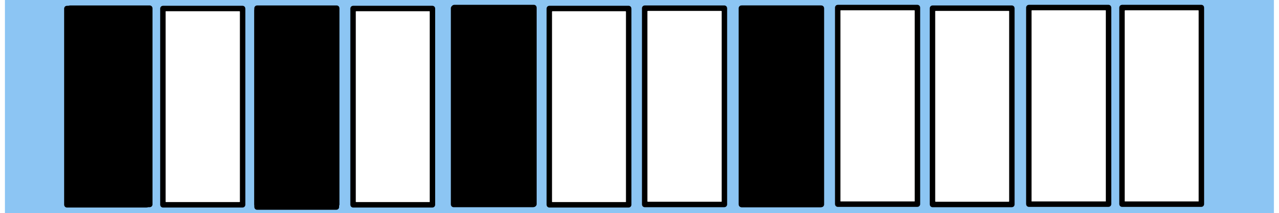
Bloom Filters for Graph Mining

A set = {A, B, C}



Bloom filter $\mathcal{B}_X$ of X

Bitvector of size B_X [bits] $B_X = 12$



Hash functions $h_1, \ldots, h_b$
$h_i : X \rightarrow \{1, \ldots, B_X\}$

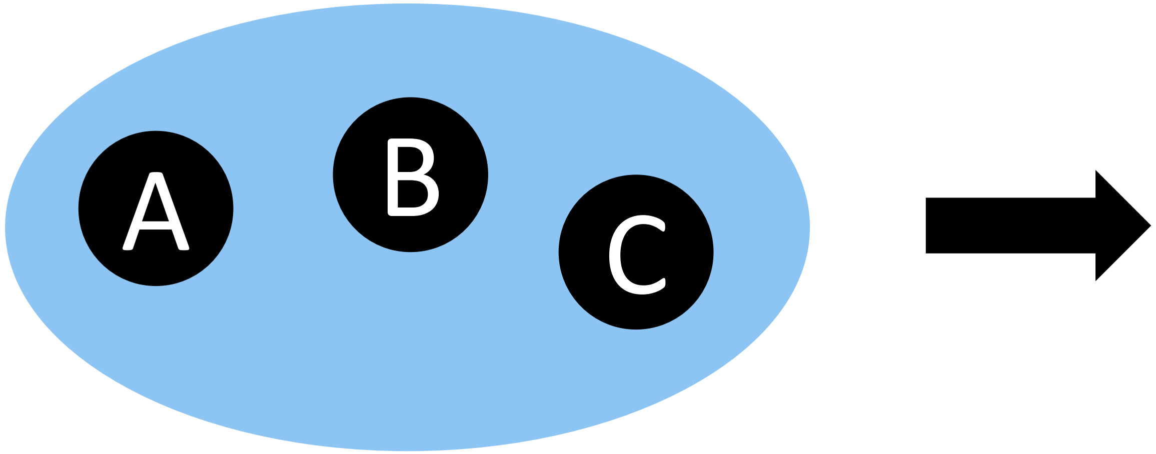
$b = 2$
$h_2, h_1 : X \rightarrow \{1, \ldots, 12\}$

h1(A) = 3   h1(B) = 1   h1(C) = 4

h2(A) = 5   h2(B) = 8   h2(C) = 11

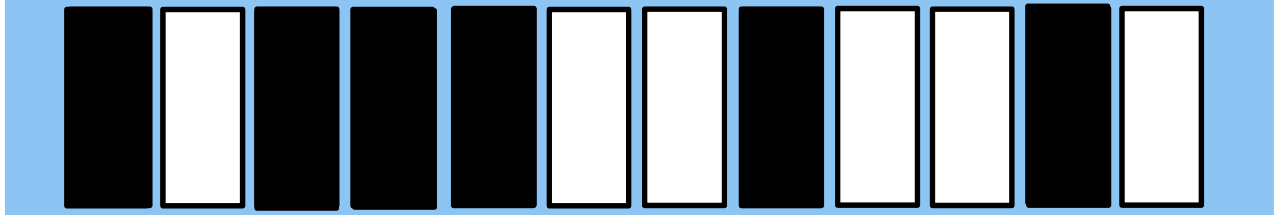
Bloom Filters for Graph Mining

A set = {A, B, C}



Bloom filter $\mathcal{B}_X$ of X

Bitvector of size B_X [bits] $B_X = 12$



Hash functions $h_1, \ldots, h_b$

$h_i : X \rightarrow \{1, \ldots, B_X\}$

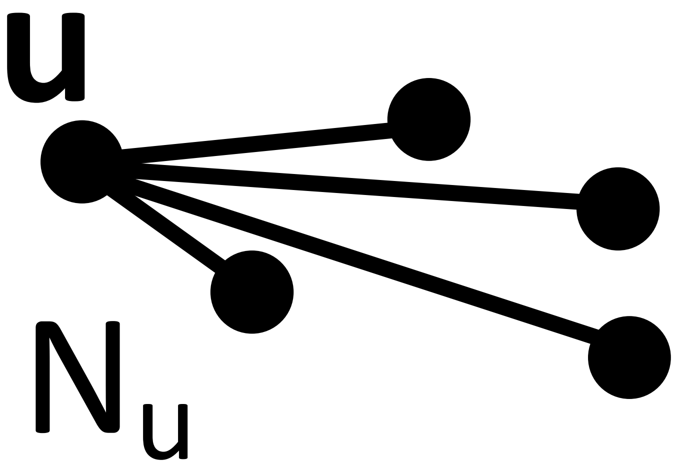
$b = 2$

$h_2, h_1 : X \rightarrow \{1, \ldots, 12\}$

h_1(A) = 3 h_1(B) = 1 h_1(C) = 4

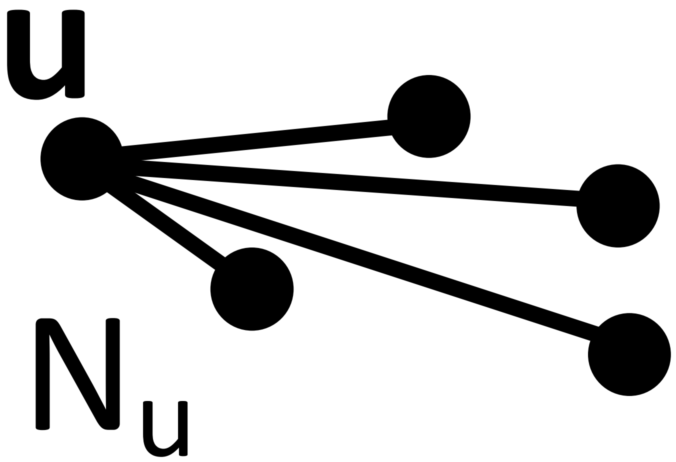
h_2(A) = 5 h_2(B) = 8 h_2(C) = 11

Bloom Filters for Graph Mining



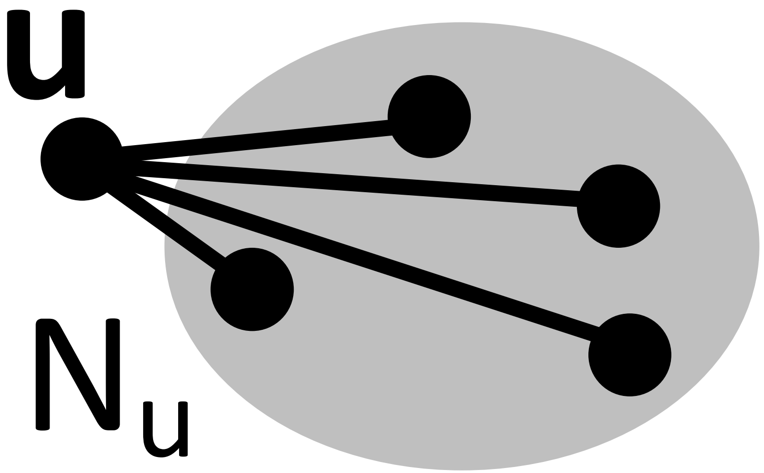
Bloom Filters for Graph Mining

Each neighborhood N_u is a set of vertices



Bloom Filters for Graph Mining

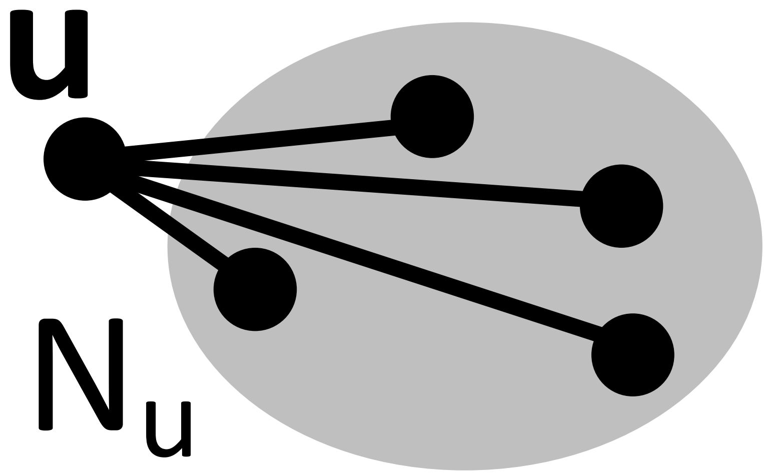
Each neighborhood N_u is a set of vertices



Bloom Filters for Graph Mining

Each neighborhood N_u is a set of vertices

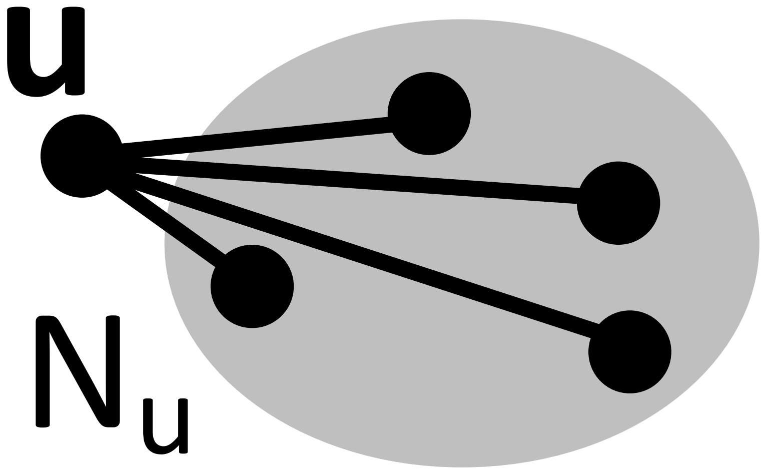
„Sketch” each N_u with a Bloom filter



Bloom Filters for Graph Mining

Each neighborhood N_u is a set of vertices

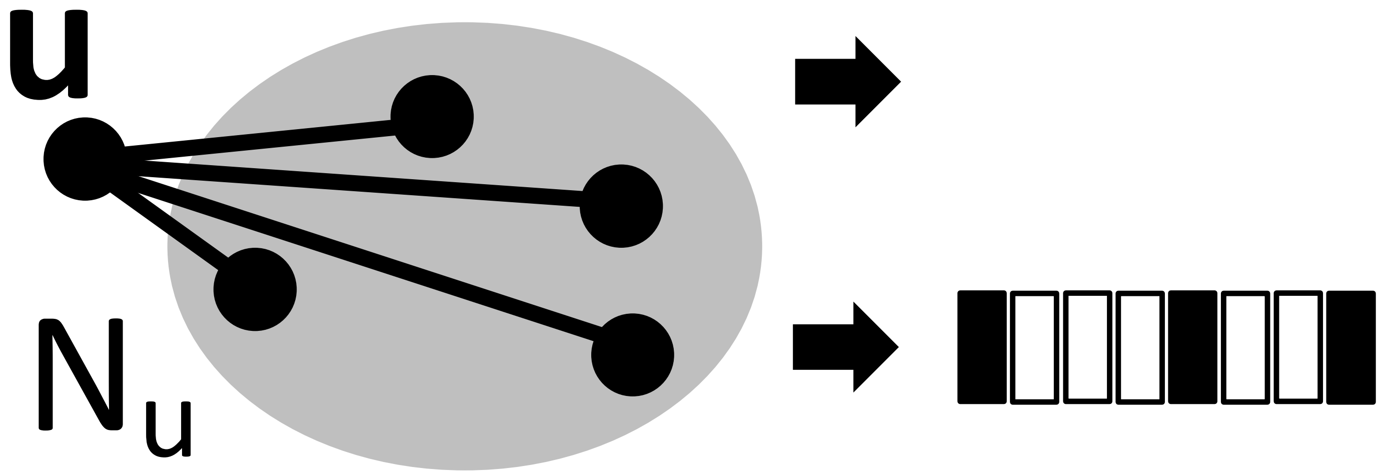
„Sketch" each N_u with a Bloom filter



Bloom Filters for Graph Mining

Each neighborhood N_u is a <u>set</u> of vertices

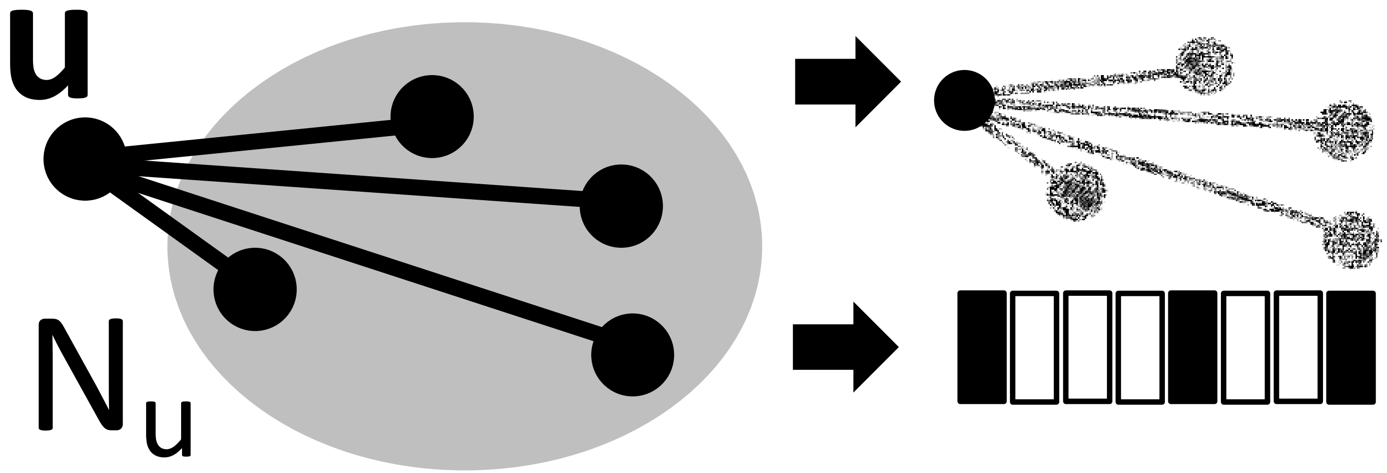
„Sketch” each N_u with a Bloom filter



Bloom Filters for Graph Mining

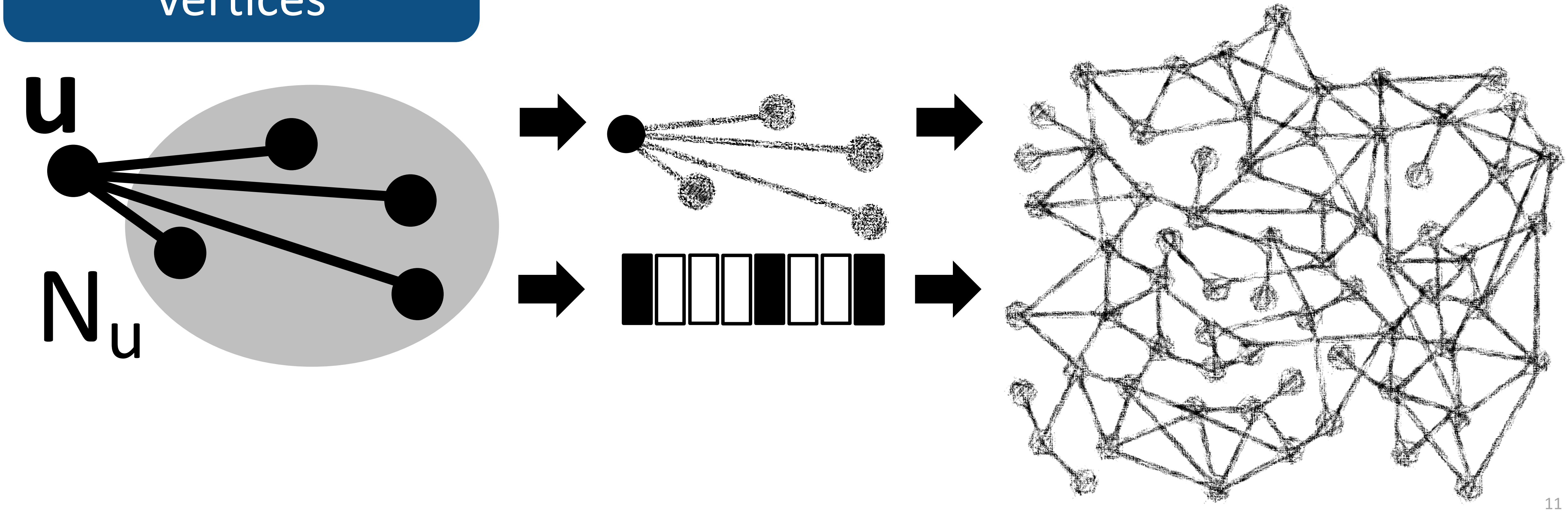
Each neighborhood N_u is a set of vertices

„Sketch” each N_u with a Bloom filter



Bloom Filters for Graph Mining

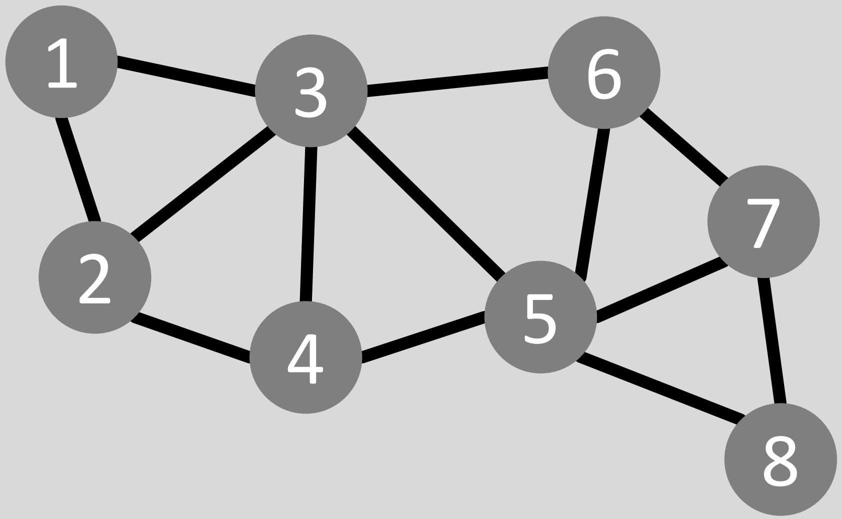
Each neighborhood N_u is a set of vertices → „Sketch” each N_u with a Bloom filter



ProbGraph: Summary of Design

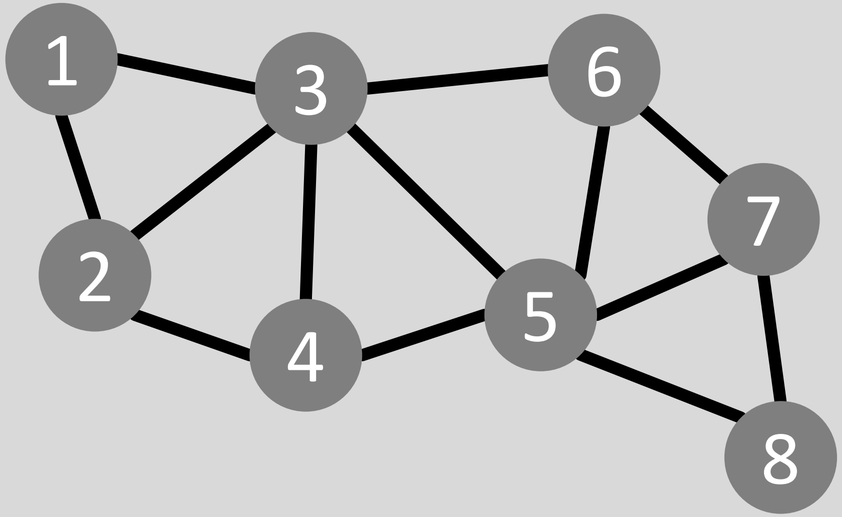
ProbGraph: Summary of Design

Input graph G



ProbGraph: Summary of Design

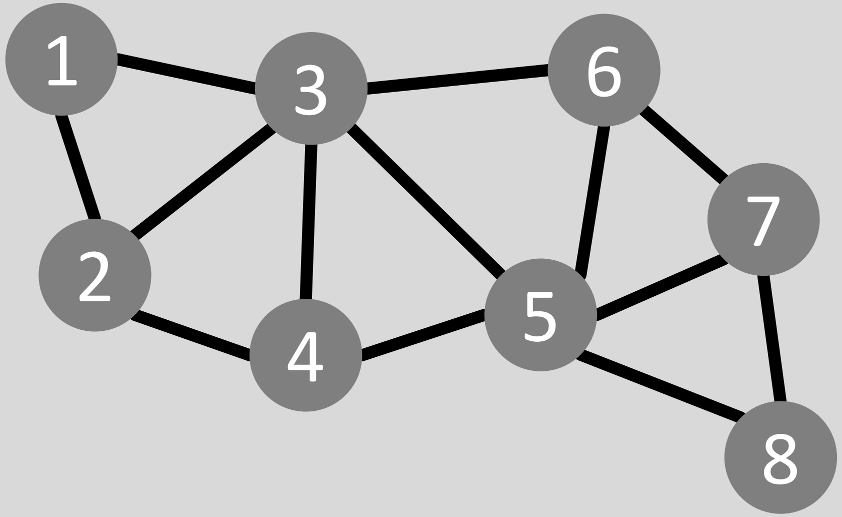
Input graph G



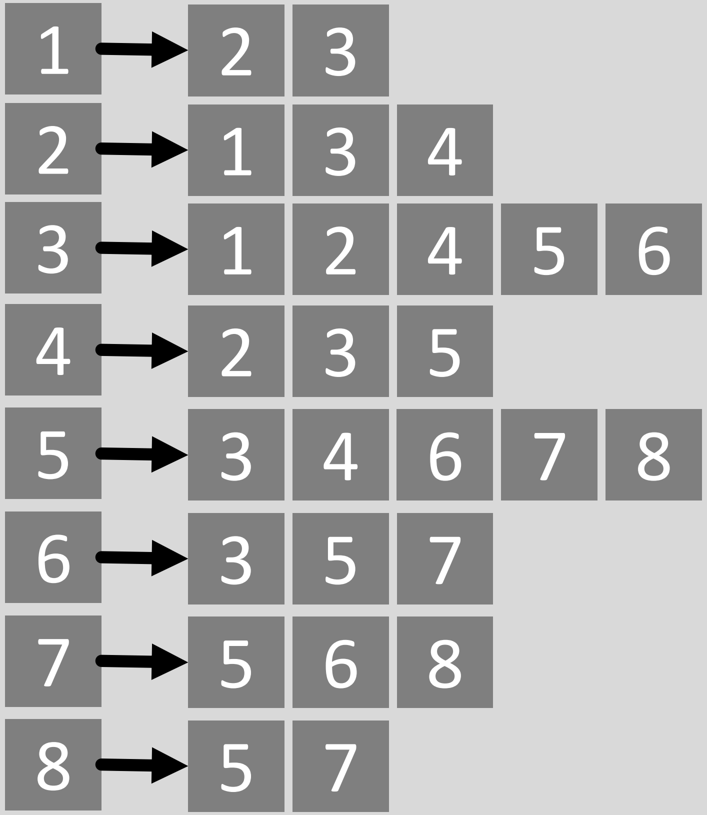
Standard graph representation (e.g., CSR)

ProbGraph: Summary of Design

Input graph G



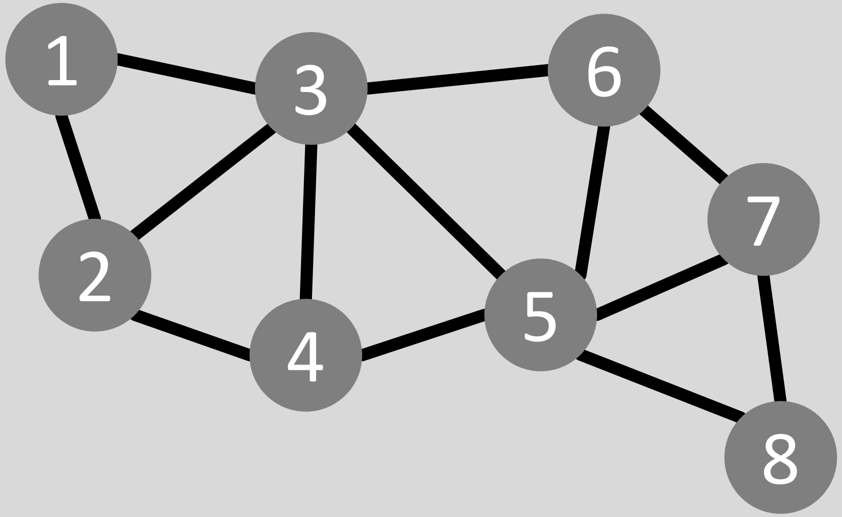
Standard graph representation (e.g., CSR)



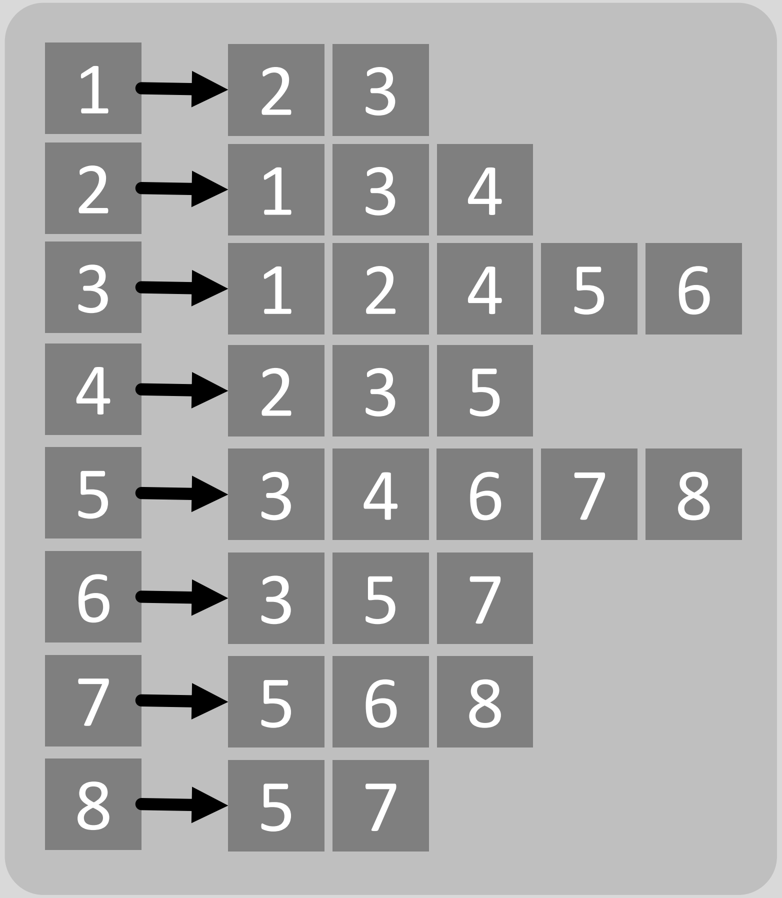
1 → 2 3
2 → 1 3 4
3 → 1 2 4 5 6
4 → 2 3 5
5 → 3 4 6 7 8
6 → 3 5 7
7 → 5 6 8
8 → 5 7

ProbGraph: Summary of Design

Input graph G



ProbGraph representation



1 → 2 3

2 → 1 3 4

3 → 1 2 4 5 6

4 → 2 3 5

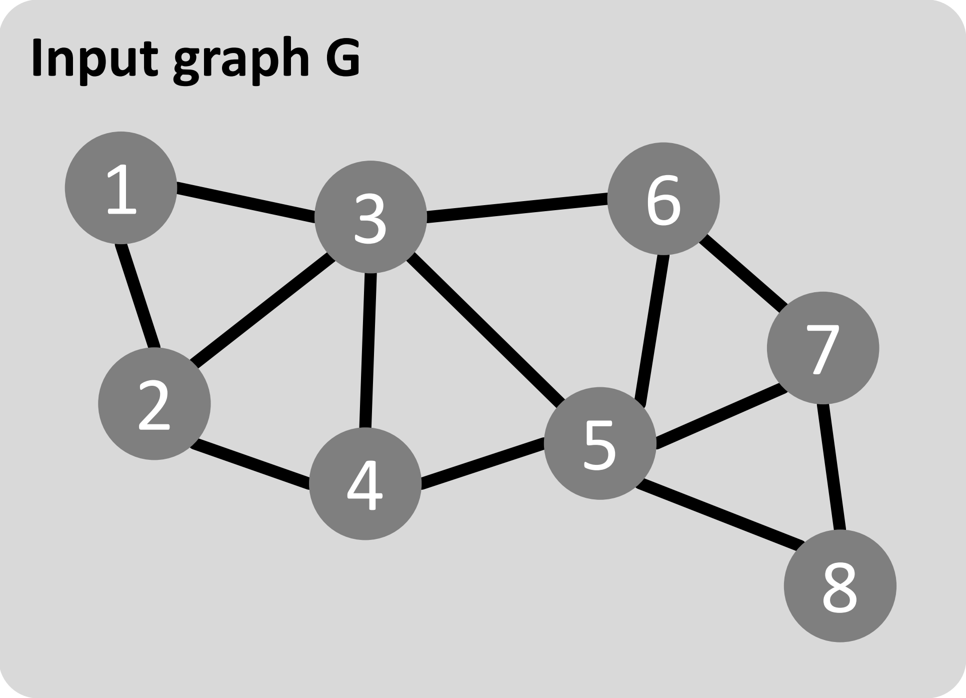
5 → 3 4 6 7 8

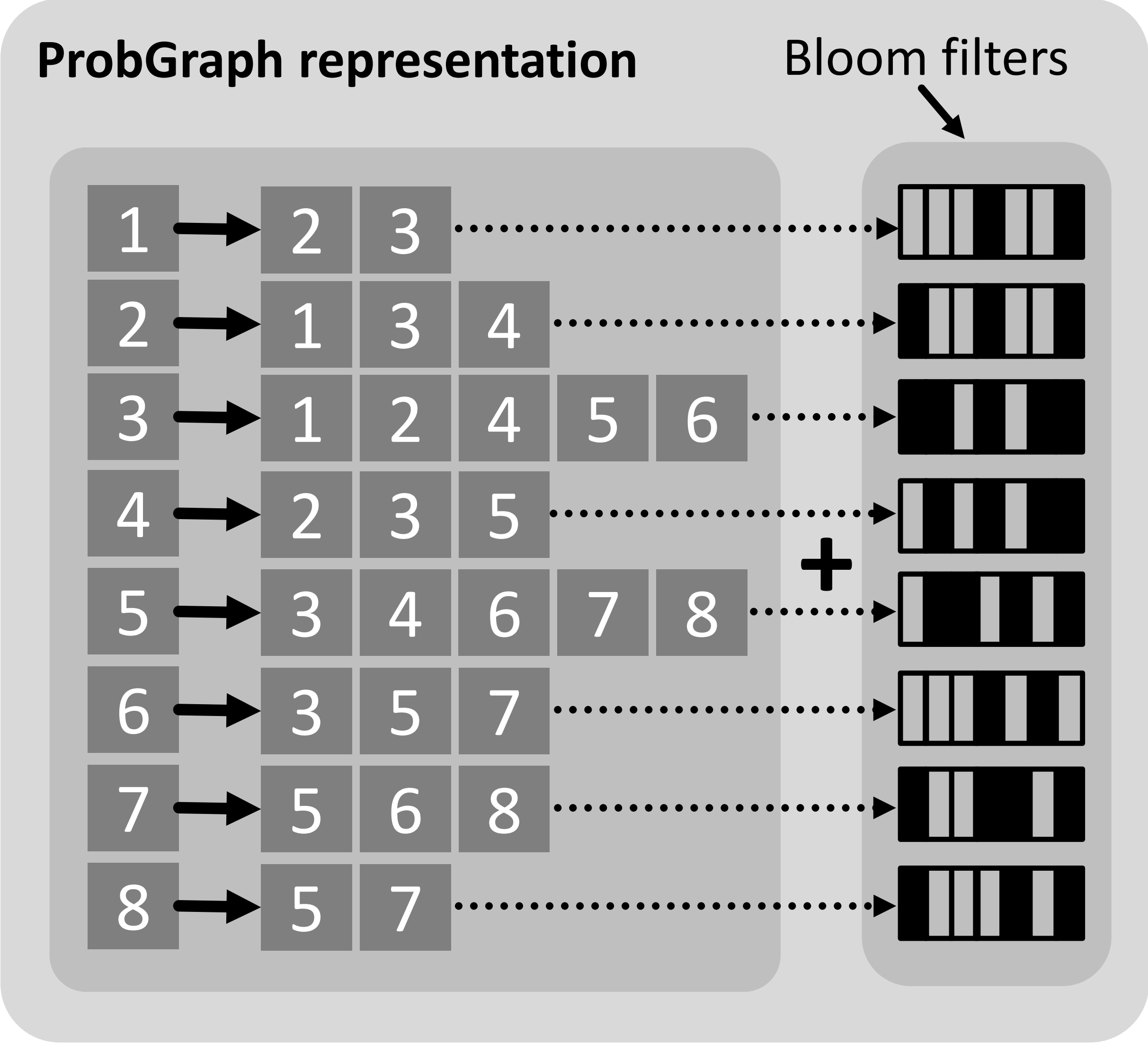
6 → 3 5 7

7 → 5 6 8

8 → 5 7

ProbGraph: Summary of Design

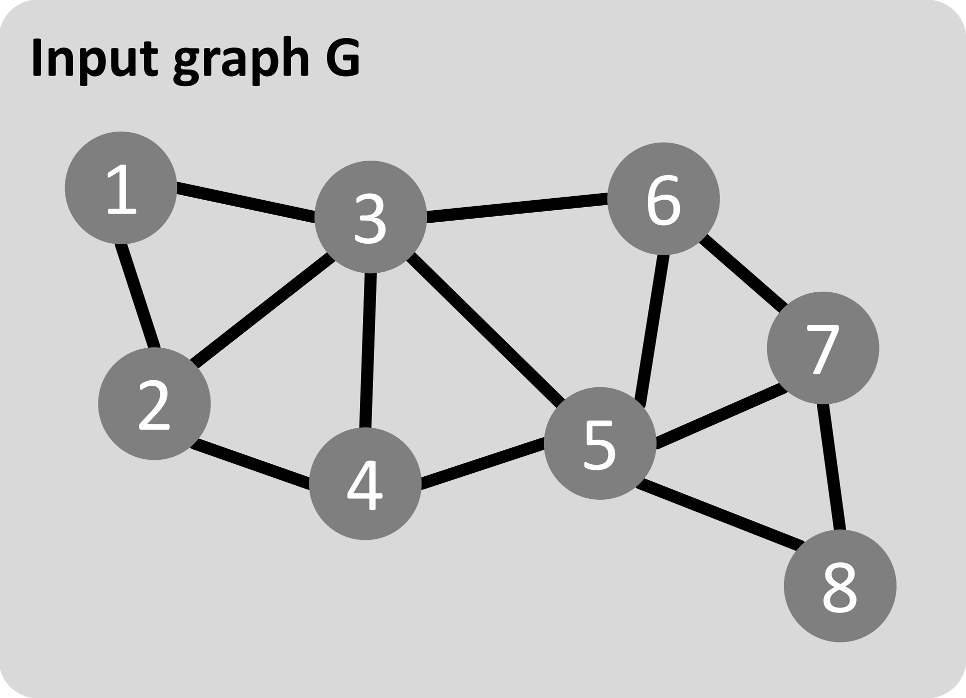

Input graph G

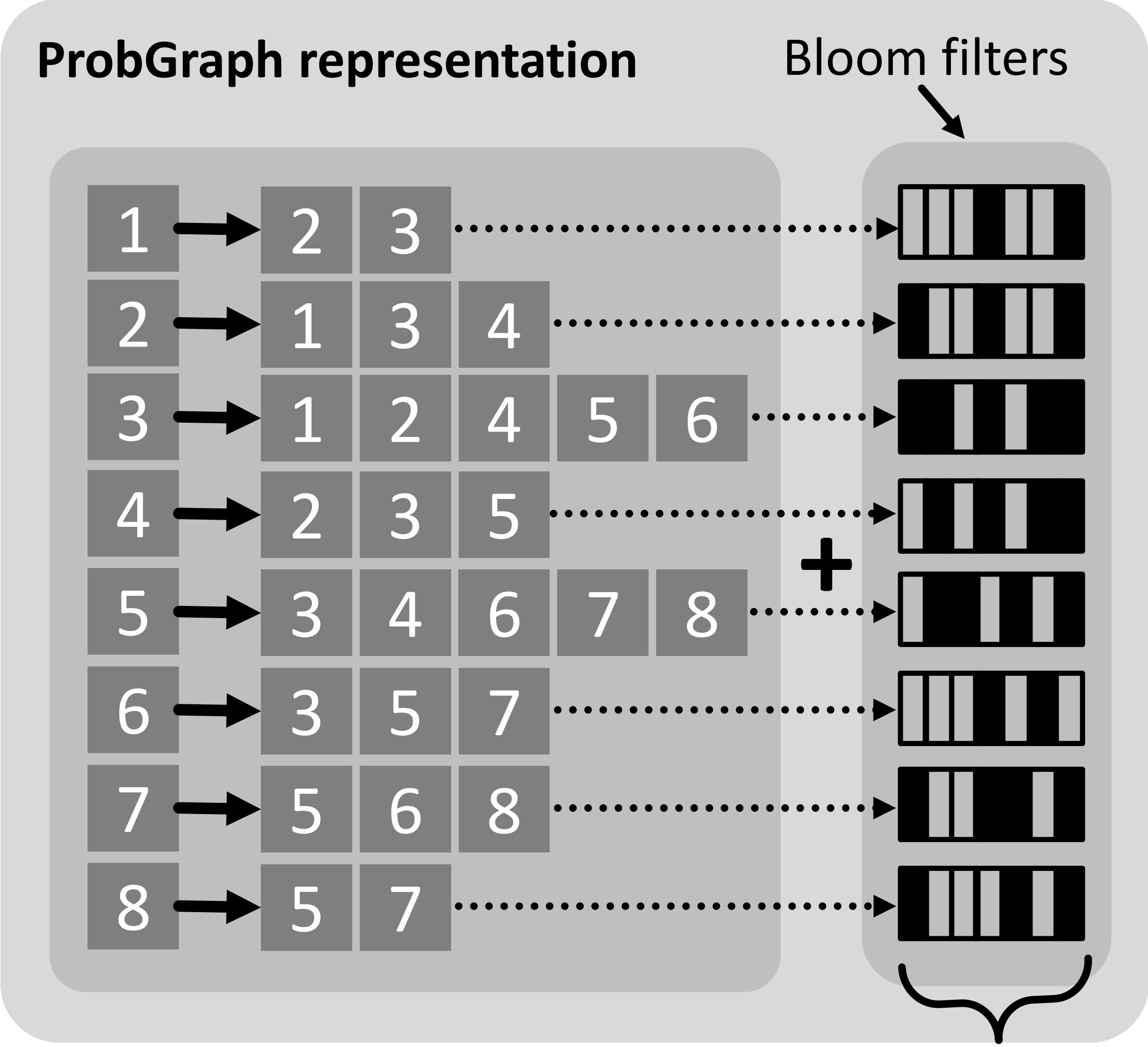

ProbGraph representation

Bloom filters

1 → 2 3
2 → 1 3 4
3 → 1 2 4 5 6
4 → 2 3 5
5 → 3 4 6 7 8
6 → 3 5 7
7 → 5 6 8
8 → 5 7

ProbGraph: Summary of Design





1 → 2 3

2 → 1 3 4

3 → 1 2 4 5 6

4 → 2 3 5

5 → 3 4 6 7 8

6 → 3 5 7

7 → 5 6 8

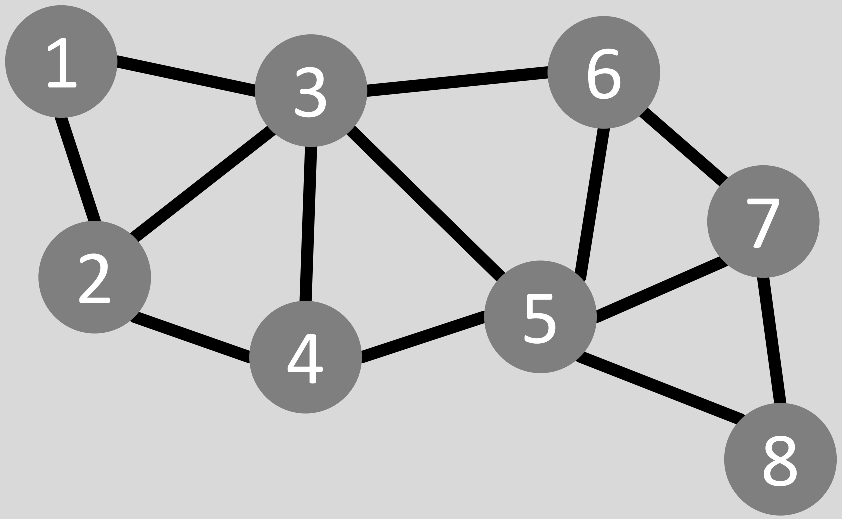
8 → 5 7

Larger B_X : more accuracy & more storage required. **Lower B_X** : vice versa.

B_X [bits]

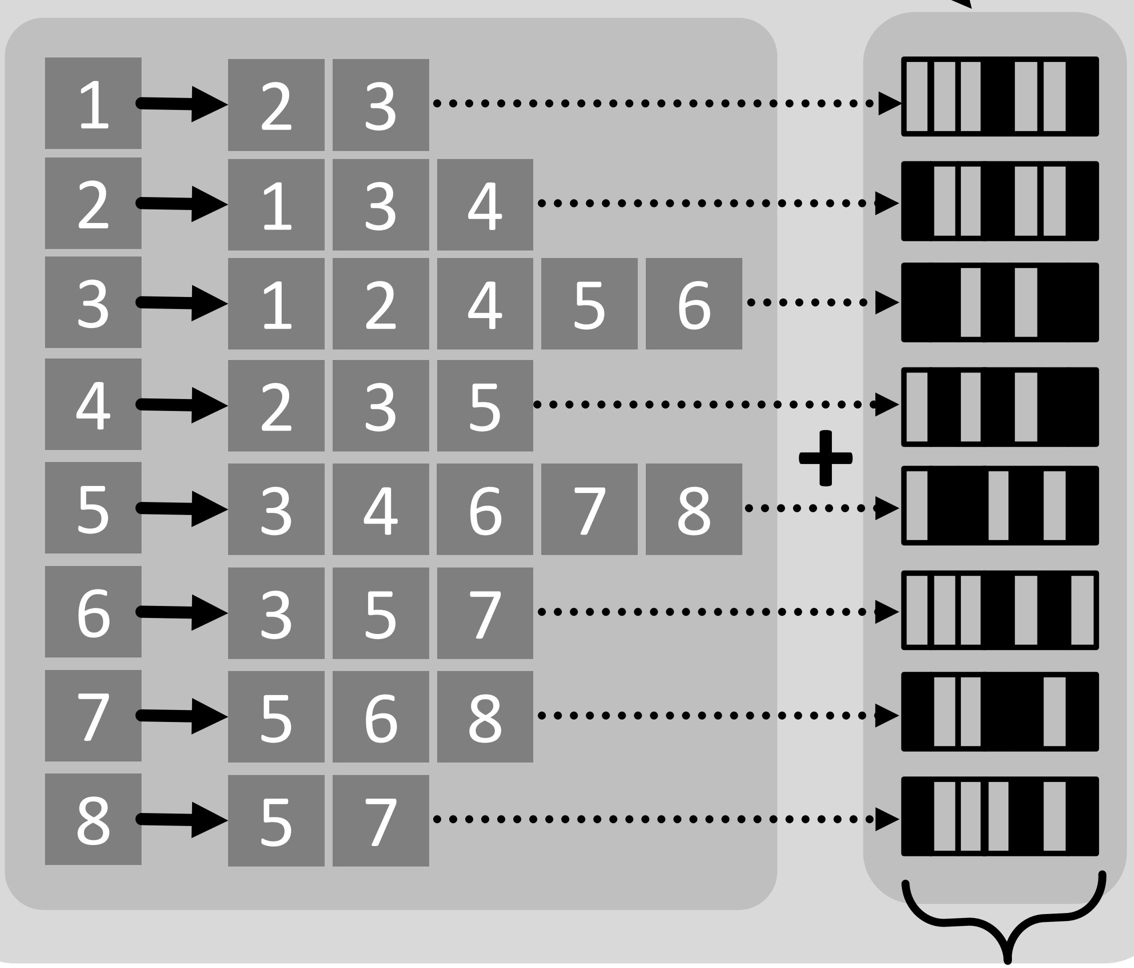
ProbGraph: Summary of Design

Input graph G



ProbGraph representation

Bloom filters



1 → 2 3

2 → 1 3 4

3 → 1 2 4 5 6

4 → 2 3 5

5 → 3 4 6 7 8

6 → 3 5 7

7 → 5 6 8

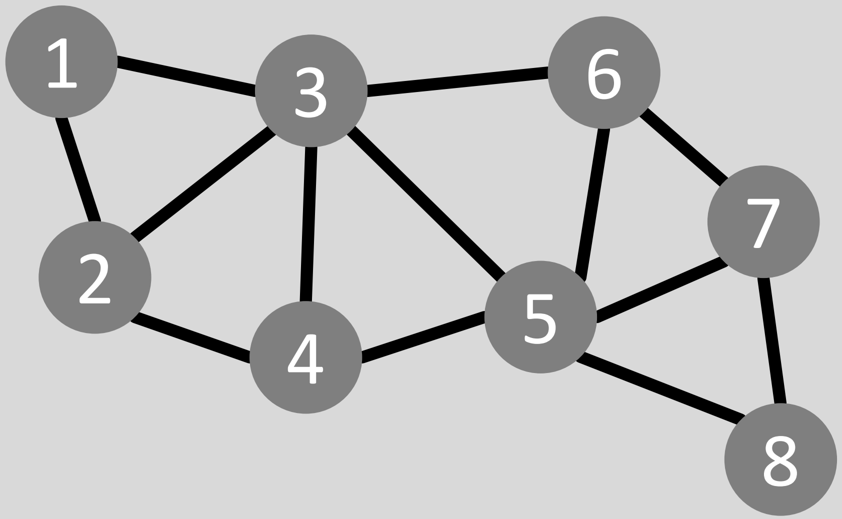
8 → 5 7

B_X is often small → <u>little storage</u>

Larger B_X : more accuracy & more storage required. **Lower B_X** : vice versa. → **B_X** [bits]

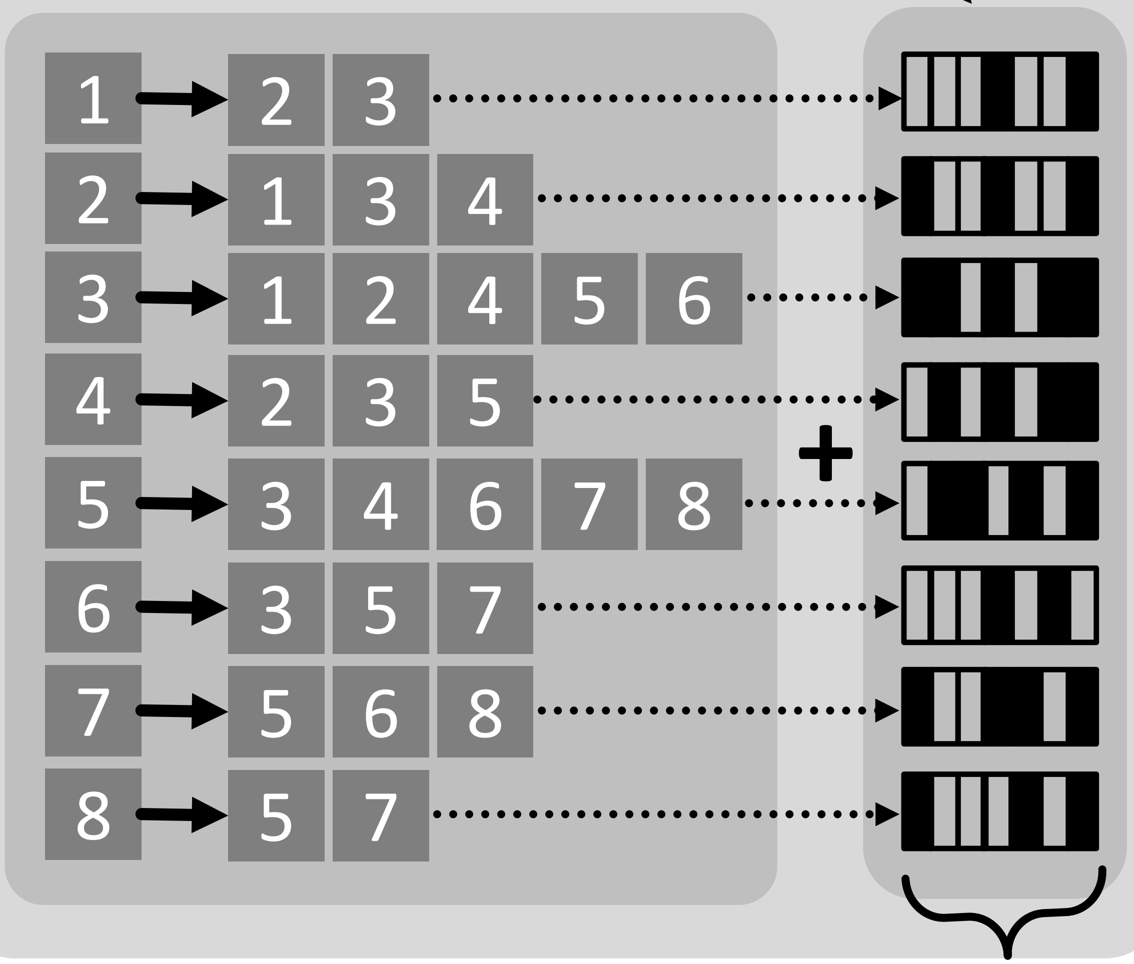
ProbGraph: Summary of Design

Input graph G



ProbGraph representation

Bloom filters



1 → 2 3
2 → 1 3 4
3 → 1 2 4 5 6
4 → 2 3 5
5 → 3 4 6 7 8
6 → 3 5 7
7 → 5 6 8
8 → 5 7

B_X is often small → little storage

BFs have the same size → great load balancing

Larger B_X : more accuracy & more storage required. **Lower B_X** : vice versa.

How does our idea compare to other Bloom filter use cases?

How does our idea compare to other Bloom filter use cases?

Traditional BF use case: presence tracking

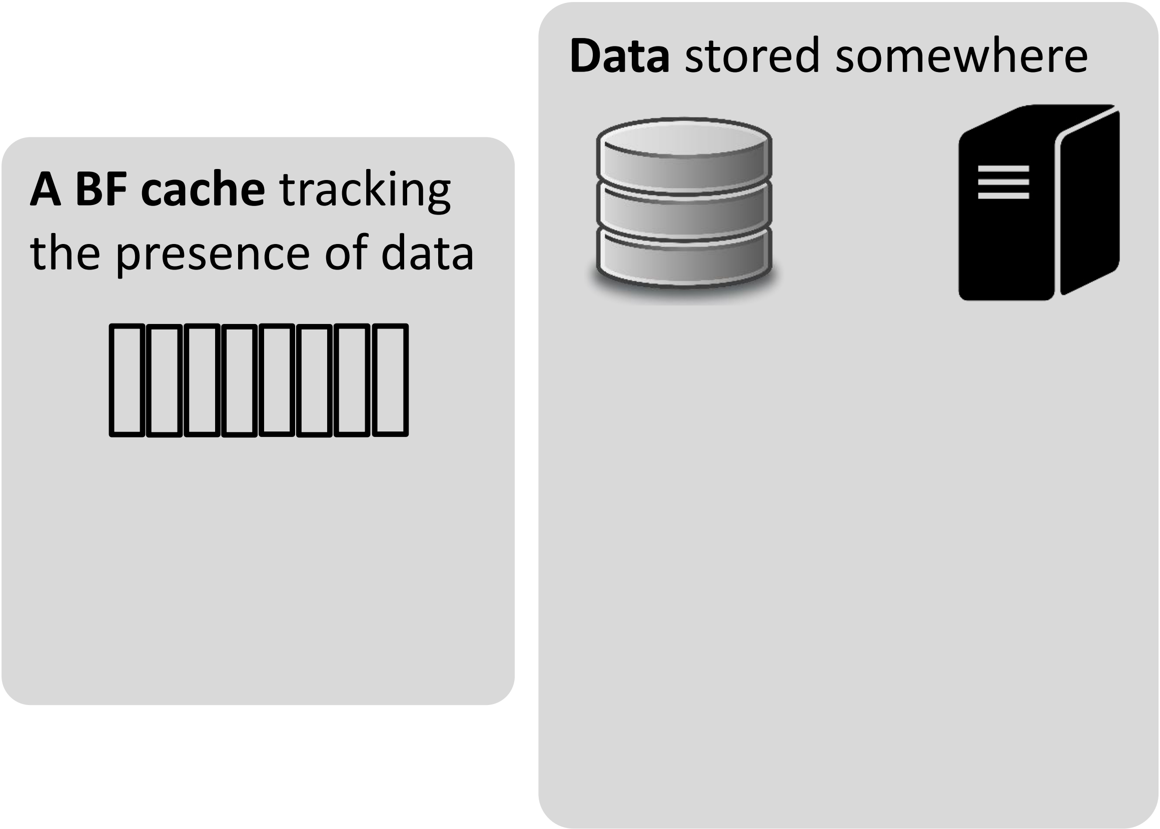
How does our idea compare to other Bloom filter use cases?

Traditional BF use case: presence tracking

Data stored somewhere

How does our idea compare to other Bloom filter use cases?

Traditional BF use case: presence tracking

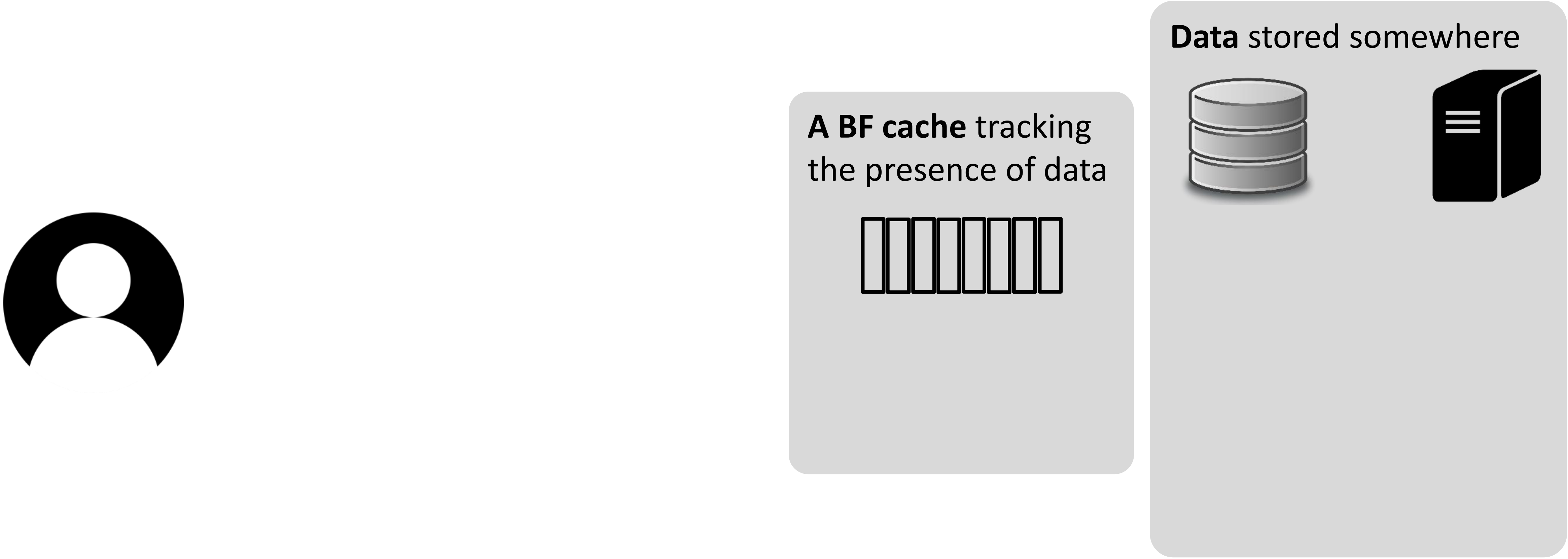


A BF cache tracking the presence of data

Data stored somewhere

How does our idea compare to other Bloom filter use cases?

Traditional BF use case: presence tracking

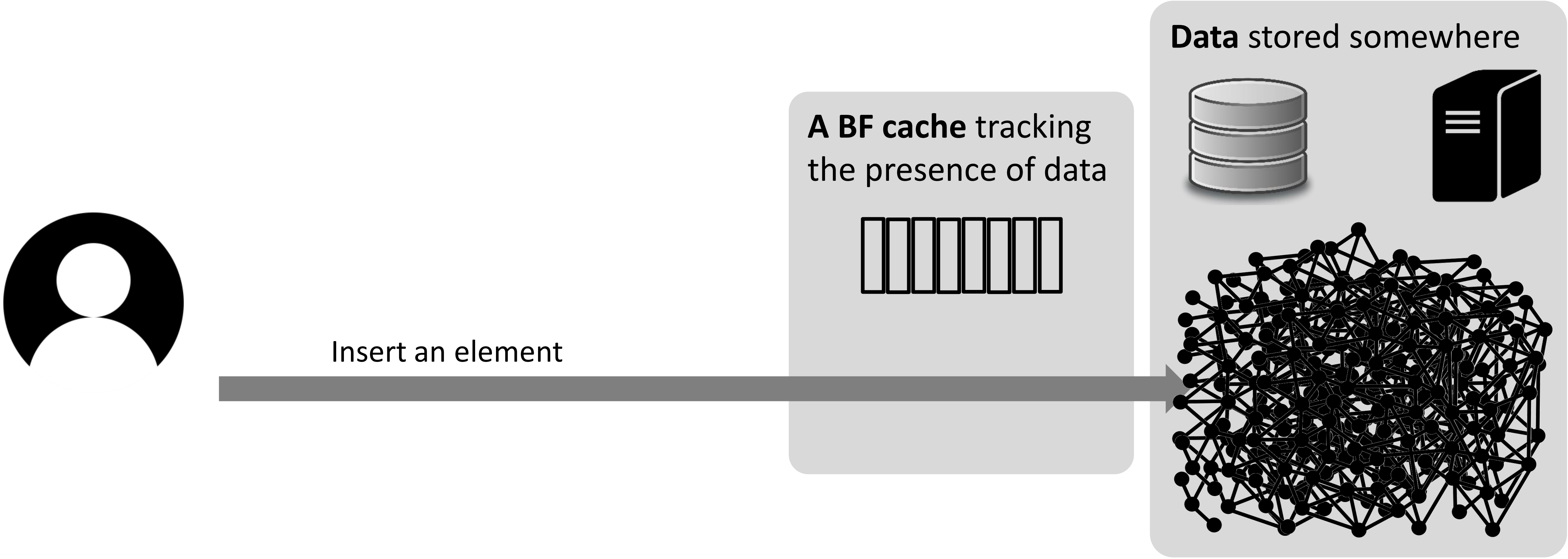


A BF cache tracking the presence of data

Data stored somewhere

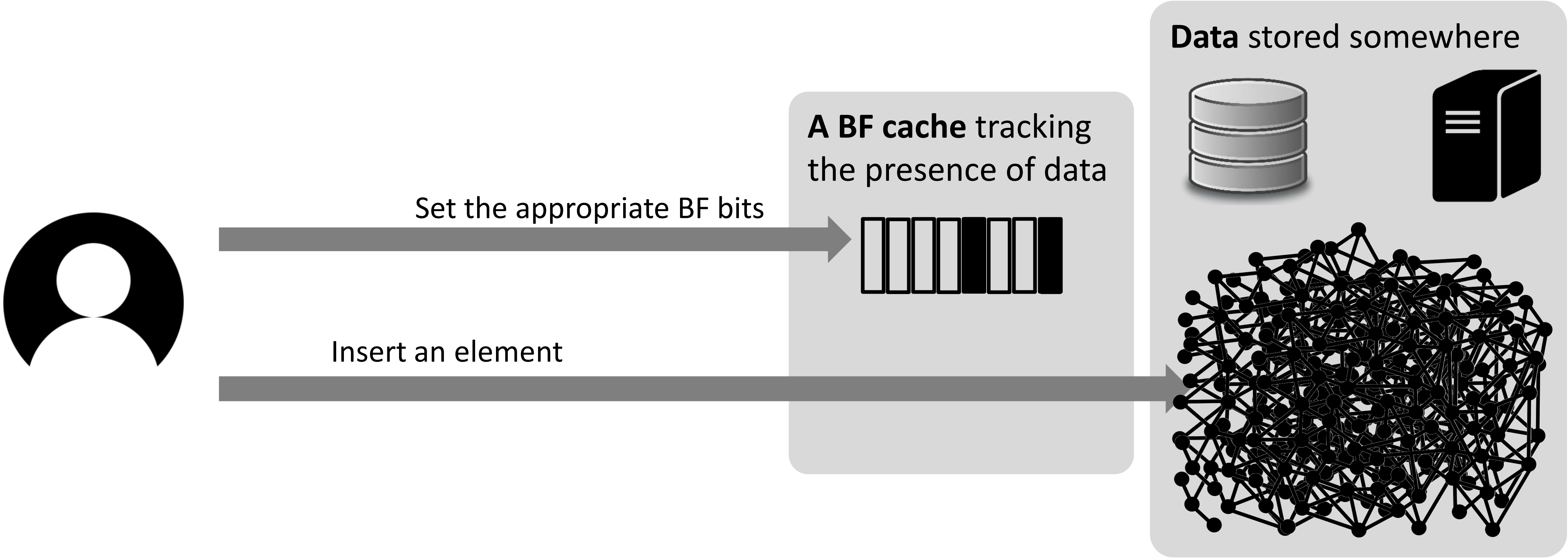
How does our idea compare to other Bloom filter use cases?

Traditional BF use case: presence tracking



How does our idea compare to other Bloom filter use cases?

Traditional BF use case: presence tracking



Data stored somewhere

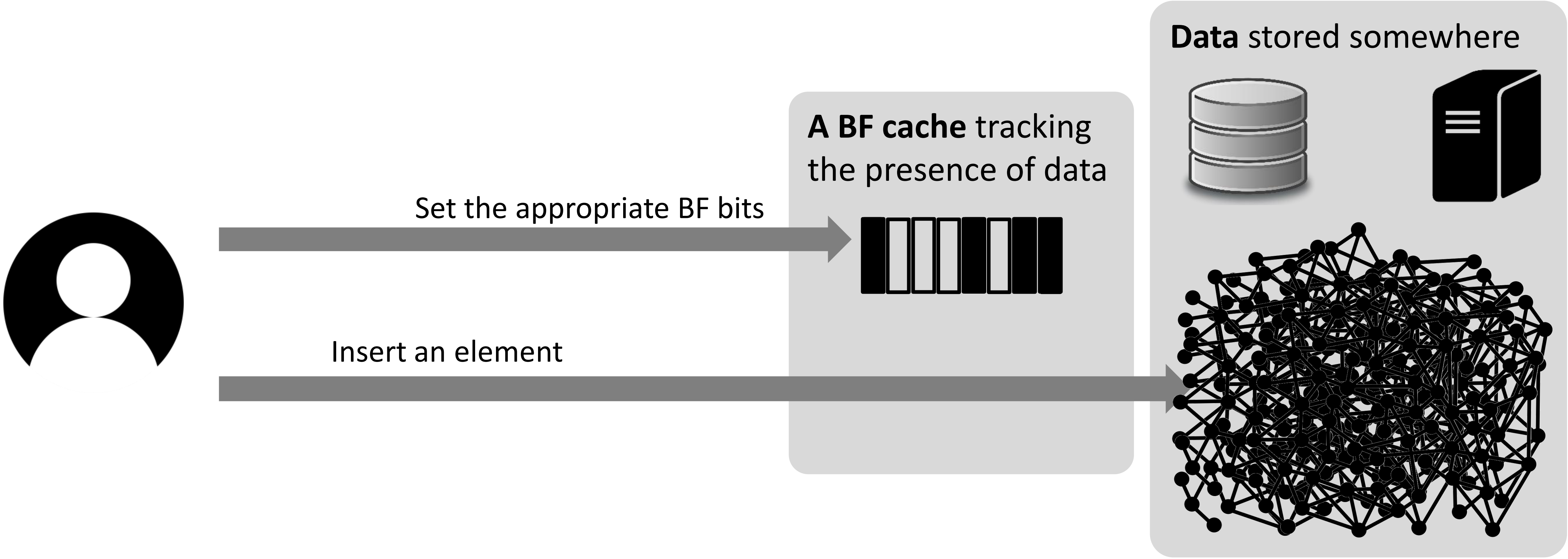
A BF cache tracking the presence of data

Set the appropriate BF bits

Insert an element

How does our idea compare to other Bloom filter use cases?

Traditional BF use case: presence tracking



Data stored somewhere

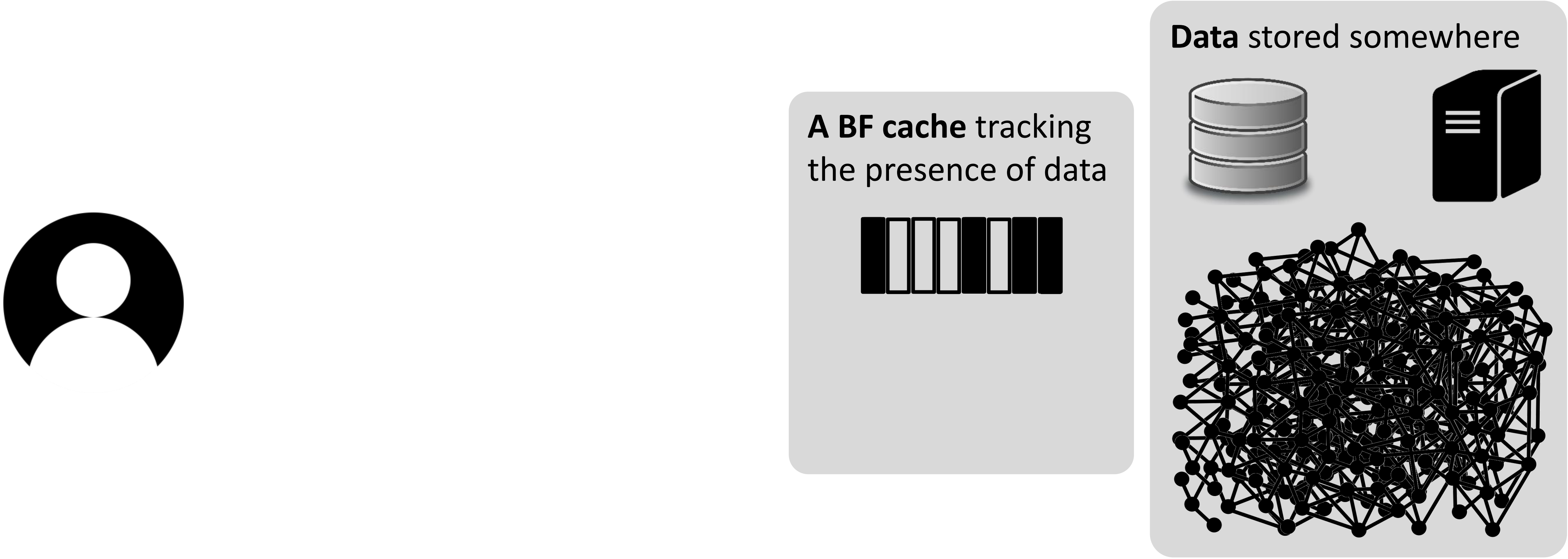
A BF cache tracking the presence of data

Set the appropriate BF bits

Insert an element

How does our idea compare to other Bloom filter use cases?

Traditional BF use case: presence tracking

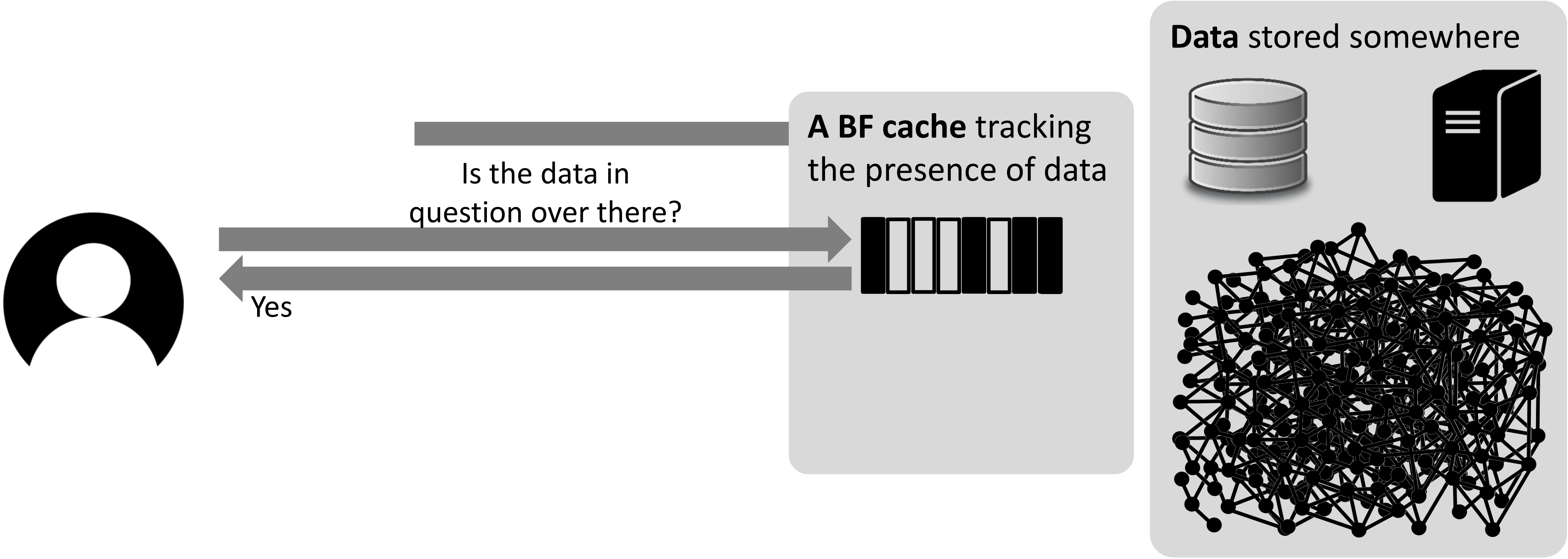


A BF cache tracking the presence of data

Data stored somewhere

How does our idea compare to other Bloom filter use cases?

Traditional BF use case: presence tracking



Data stored somewhere

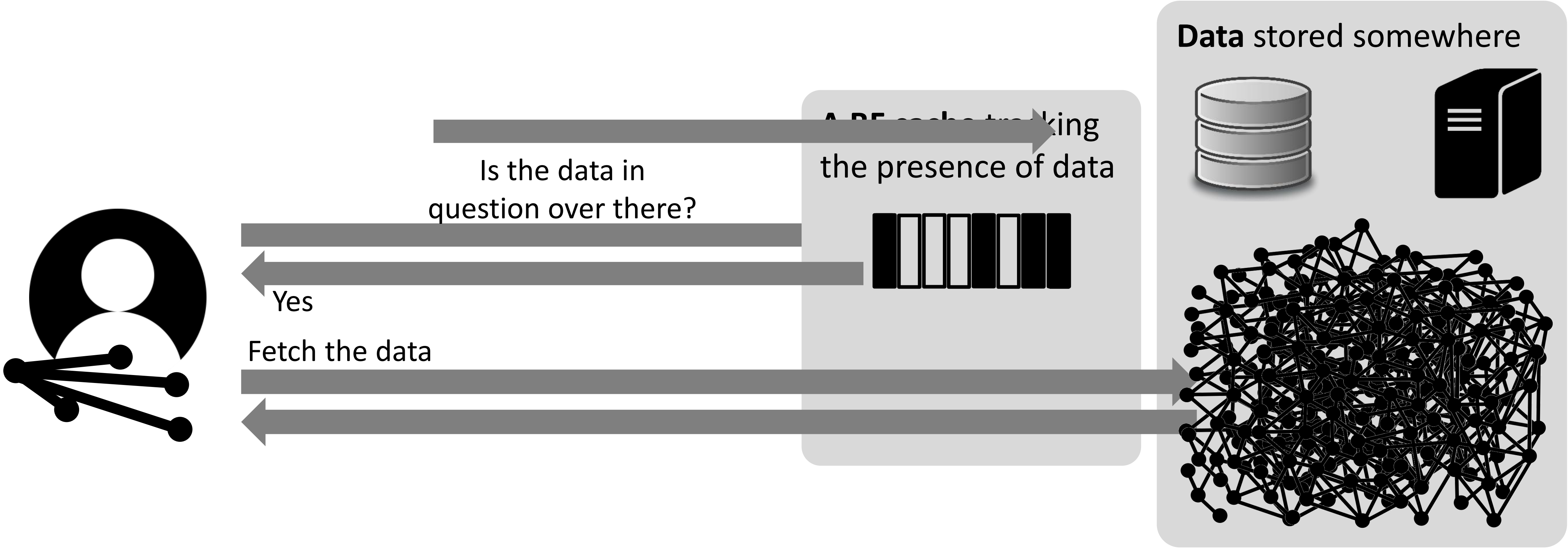
A BF cache tracking the presence of data

Is the data in question over there?

Yes

How does our idea compare to other Bloom filter use cases?

Traditional BF use case: presence tracking



Data stored somewhere

A BF cache tracking the presence of data

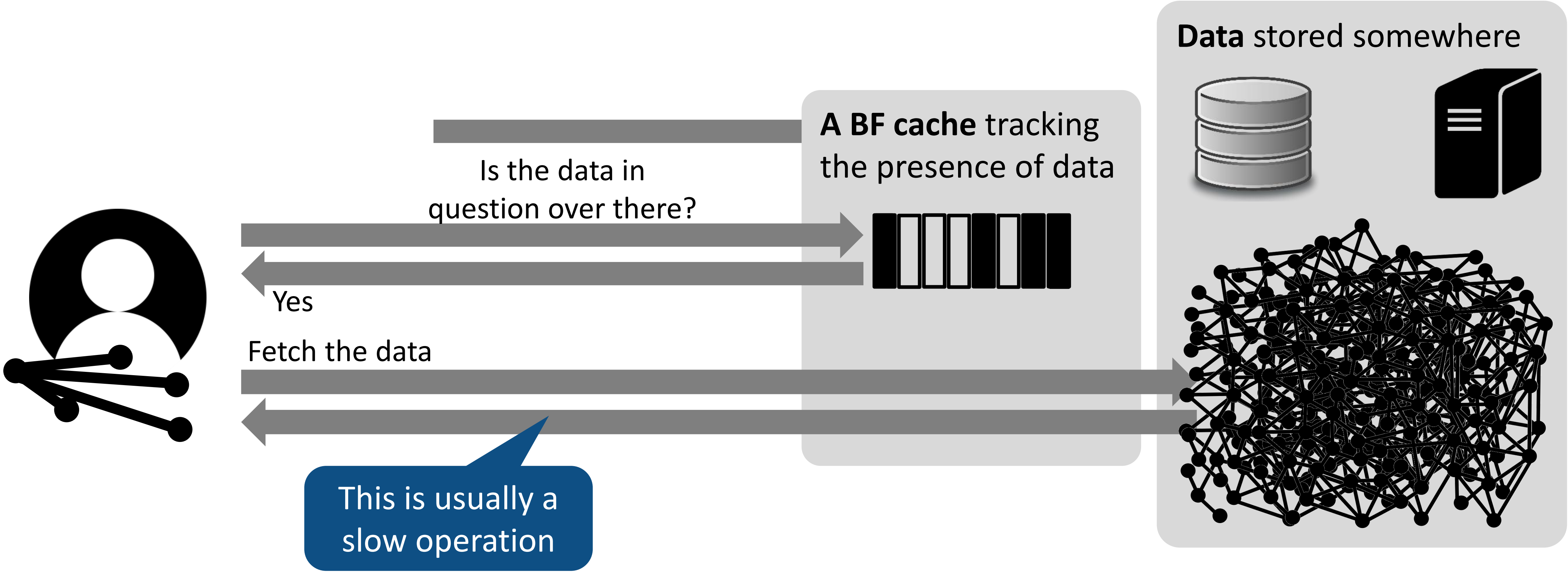
Is the data in question over there?

Yes

Fetch the data

How does our idea compare to other Bloom filter use cases?

Traditional BF use case: presence tracking



Data stored somewhere

A BF cache tracking the presence of data

Is the data in question over there?

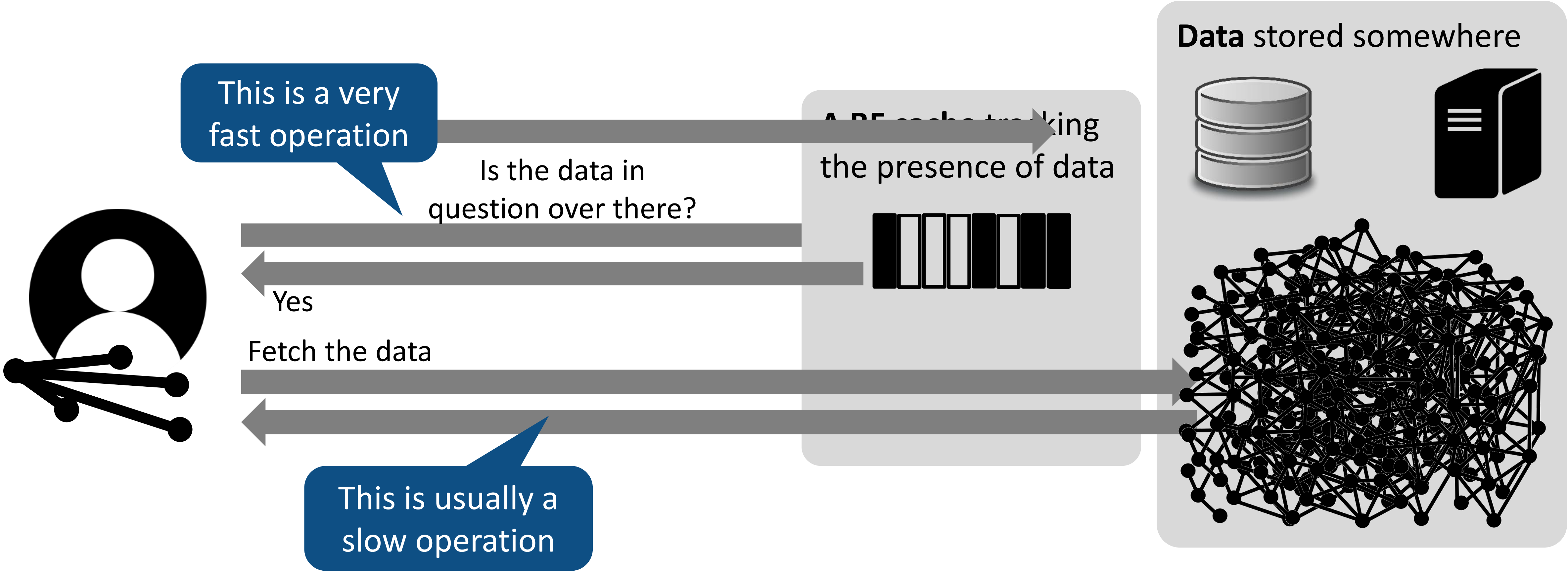
Yes

Fetch the data

This is usually a slow operation

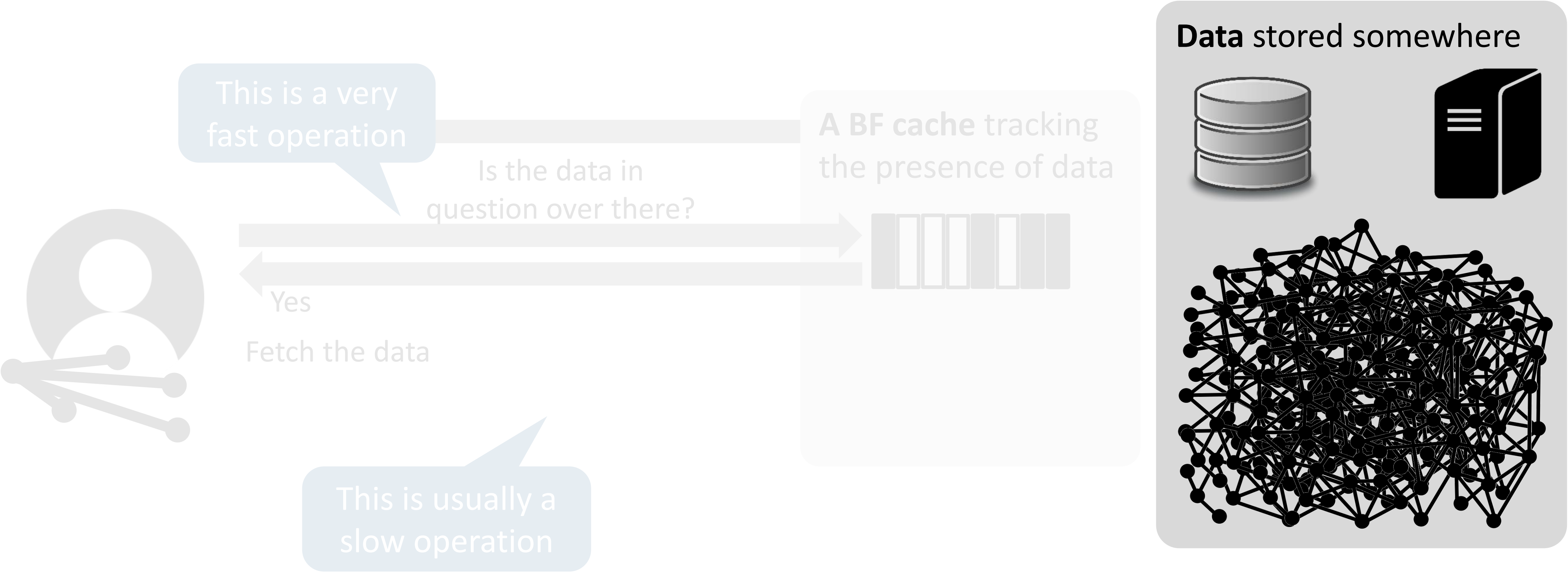
How does our idea compare to other Bloom filter use cases?

Traditional BF use case: presence tracking


Data stored somewhere
This is a very fast operation
A BF cache tracking the presence of data
Is the data in question over there?
Yes
Fetch the data
This is usually a slow operation

The novelty of ProbGraph

r Bloom filter use cases?


This is a very fast operation
Is the data in question over there?
A BF cache tracking the presence of data
Yes
Fetch the data
This is usually a slow operation
Data stored somewhere

The novelty of ProbGraph

...er Bloom filter use cases?

We use BFs as a sketch of the actual dataset

...ing

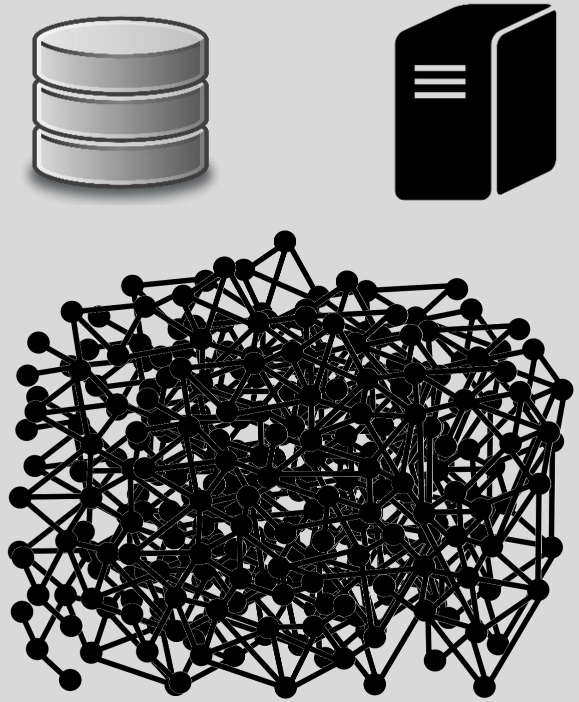
...e presence of data

Yes

Fetch the data

This is usually a slow operation

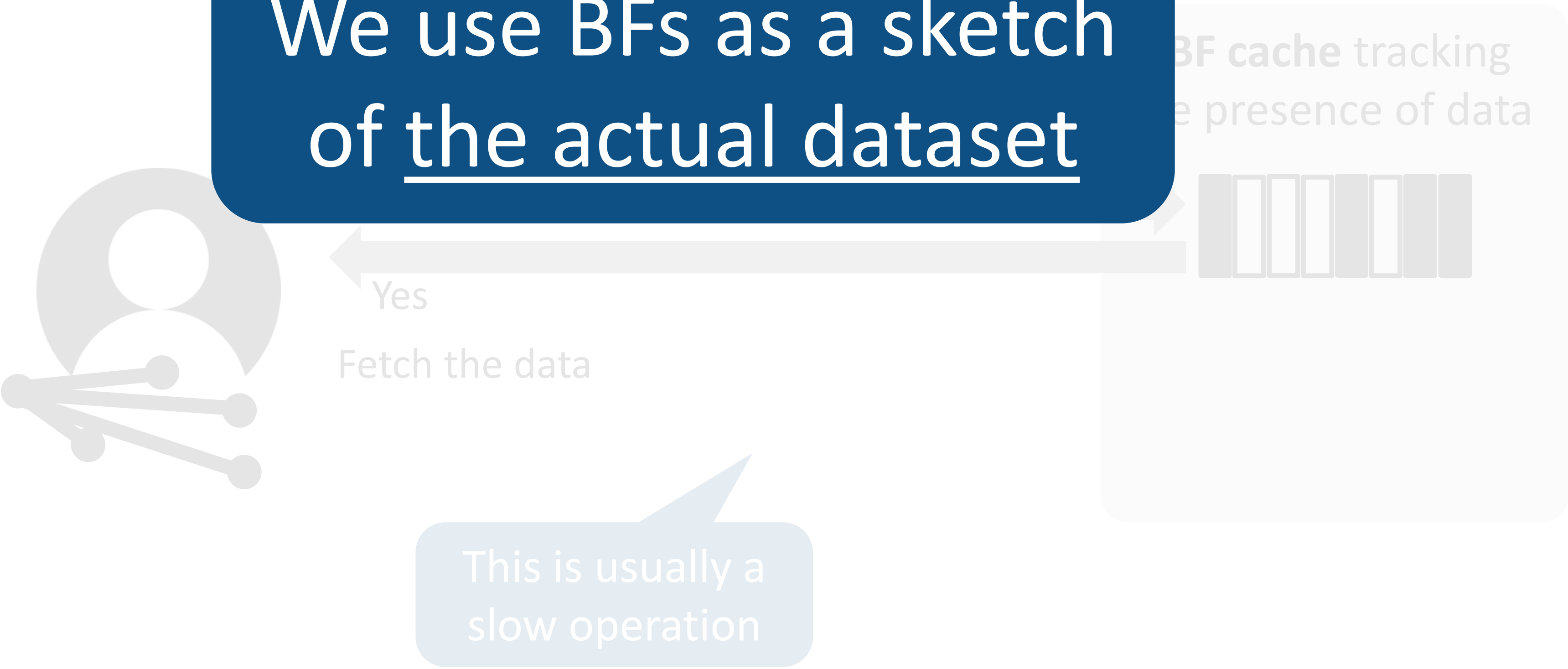
Data stored somewhere



The novelty of ProbGraph

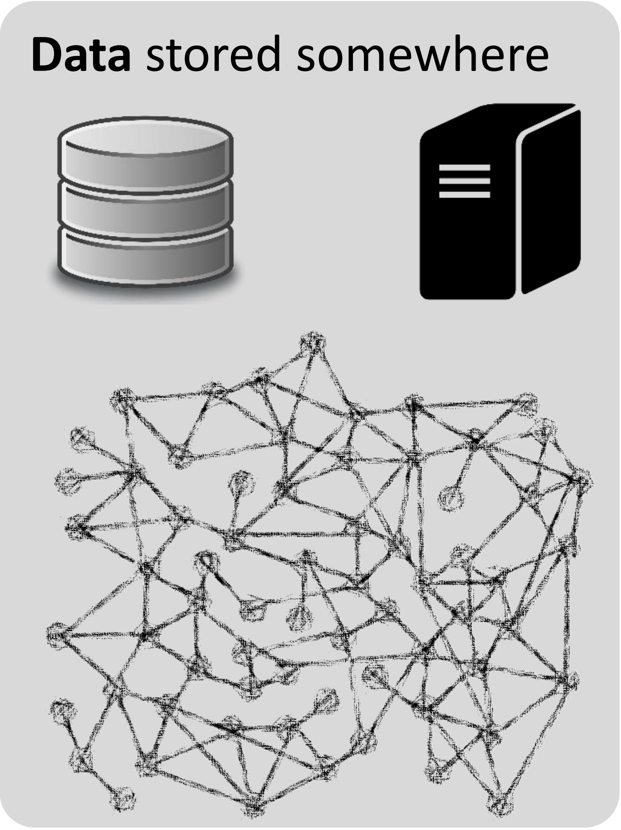
er Bloom filter use cases?

We use BFs as a sketch of <u>the actual dataset</u>



This is usually a slow operation

Data stored somewhere



The novelty of ProbGraph

er Bloom filter use cases?

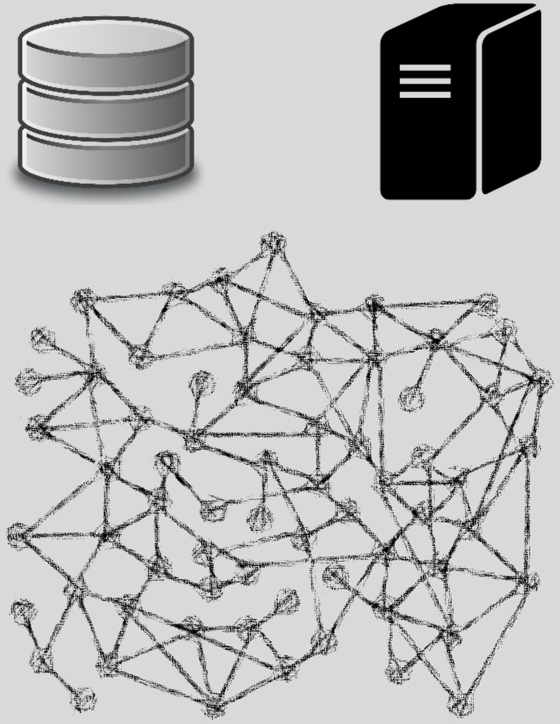
We use BFs as a sketch of the actual dataset

BF cache tracking
presence of data

Yes
Fetch the data

How do we exactly use these sketches to benefit graph mining?

Data stored somewhere

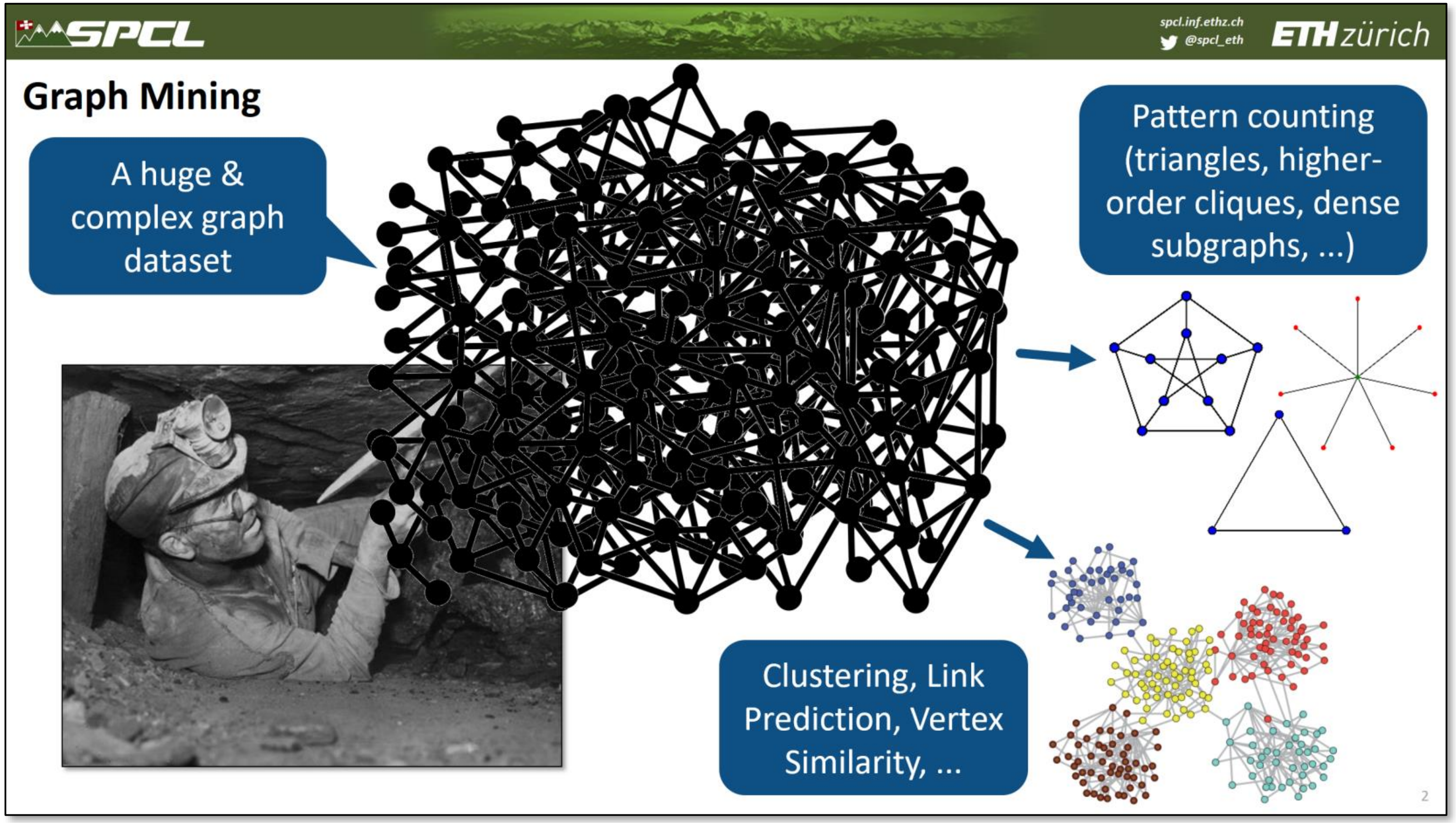


Observation: Set Intersection Cardinality is Prevalent in Graph Mining

Observation: Set Intersection Cardinality is Prevalent in Graph Mining

$$|X \cap Y|$$

Observation: Set Intersection Cardinality is Prevalent in Graph Mining



spcl.inf.ethz.ch
@spcl_eth
ETH zürich

Graph Mining

A huge & complex graph dataset

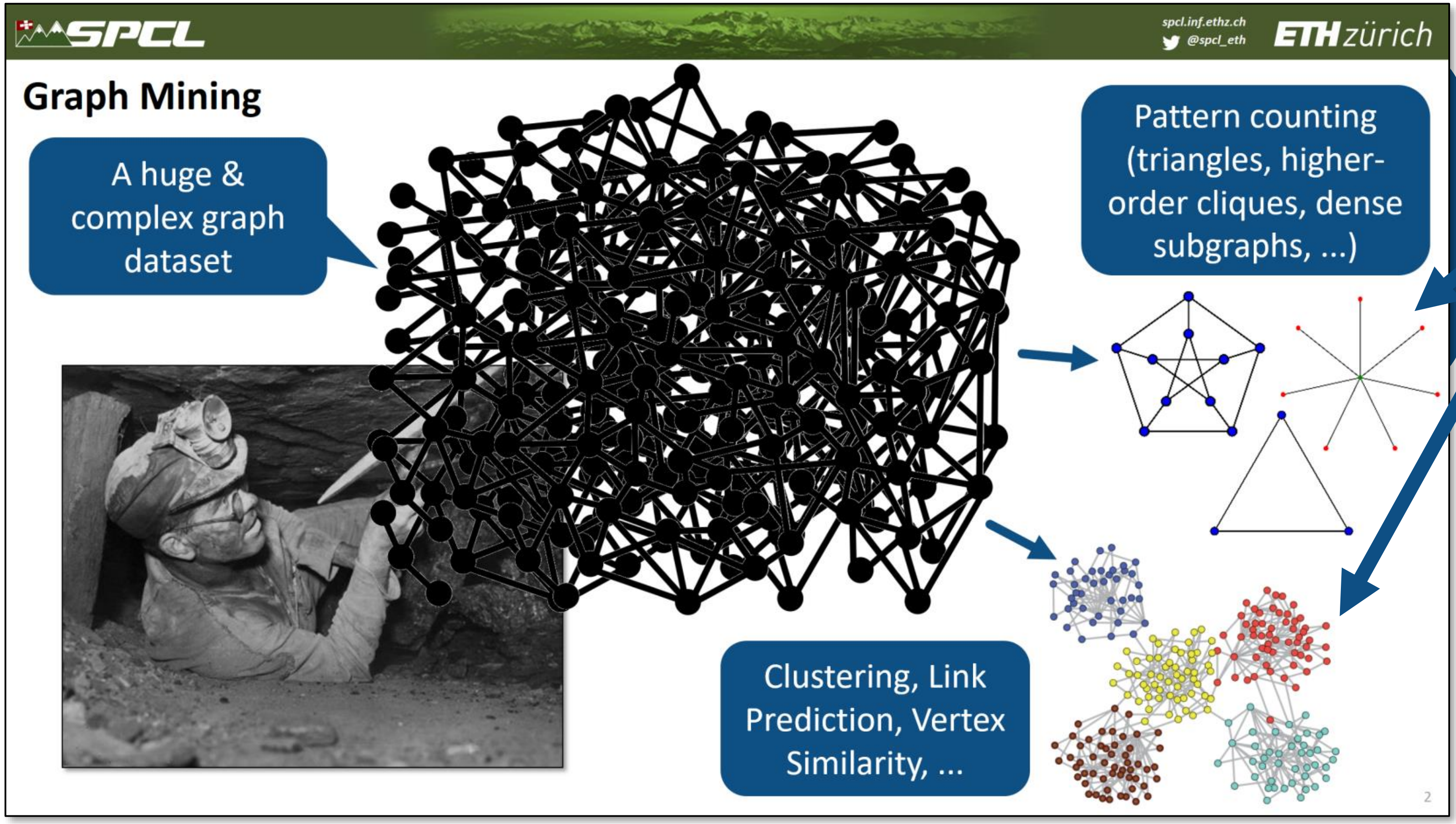
Pattern counting (triangles, higher-order cliques, dense subgraphs, ...)

Clustering, Link Prediction, Vertex Similarity, ...

2

$$|X \cap Y|$$

Observation: Set Intersection Cardinality is Prevalent in Graph Mining



spcl.inf.ethz.ch
@spcl_eth
ETH zürich

Graph Mining

A huge & complex graph dataset

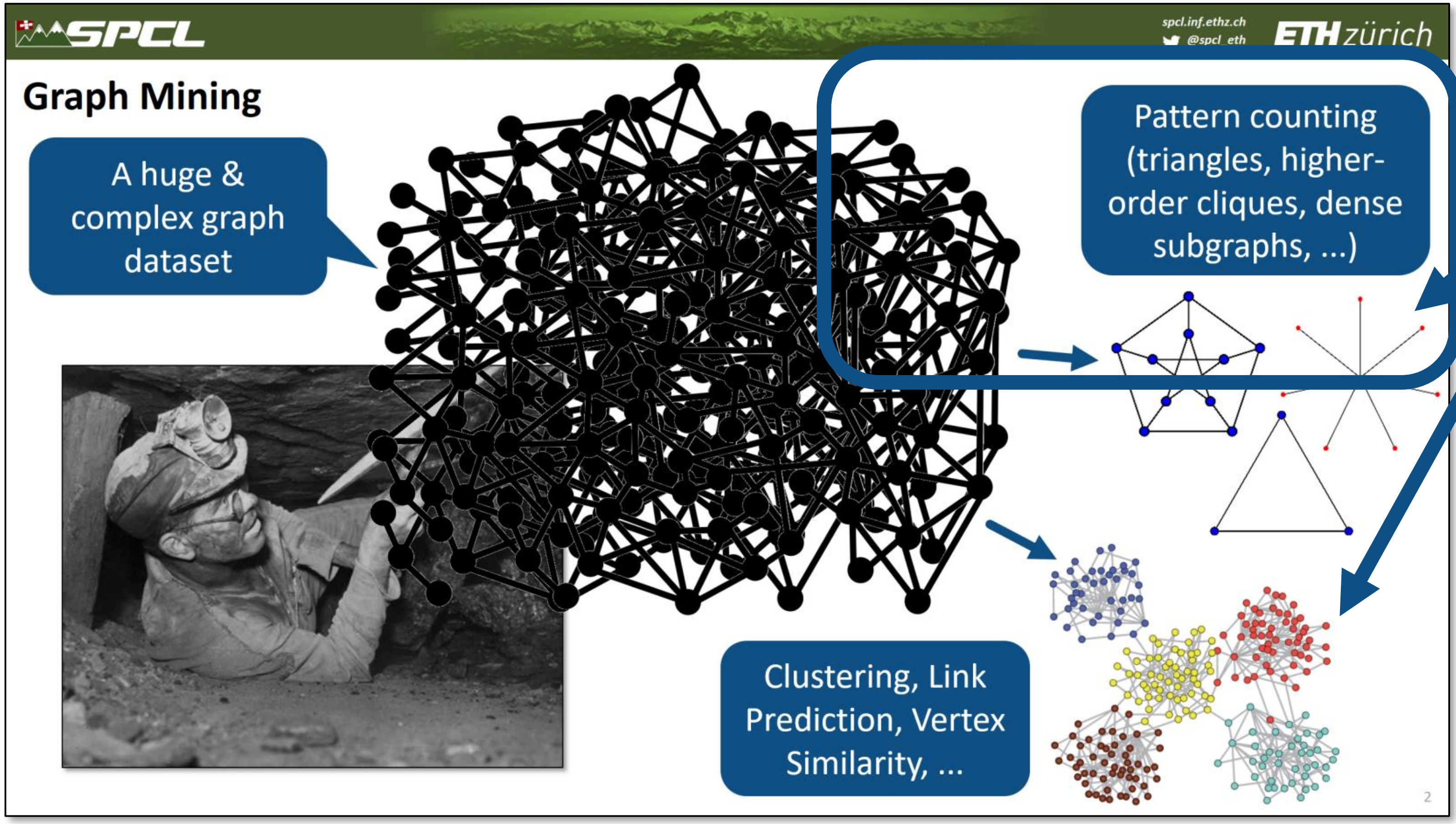
Pattern counting (triangles, higher-order cliques, dense subgraphs, ...)

Clustering, Link Prediction, Vertex Similarity, ...

2

$$|X \cap Y|$$

Observation: Set Intersection Cardinality is Prevalent in Graph Mining



spcl.inf.ethz.ch
@spcl_eth
ETH zürich

Graph Mining

A huge & complex graph dataset

Pattern counting (triangles, higher-order cliques, dense subgraphs, ...)

Clustering, Link Prediction, Vertex Similarity, ...

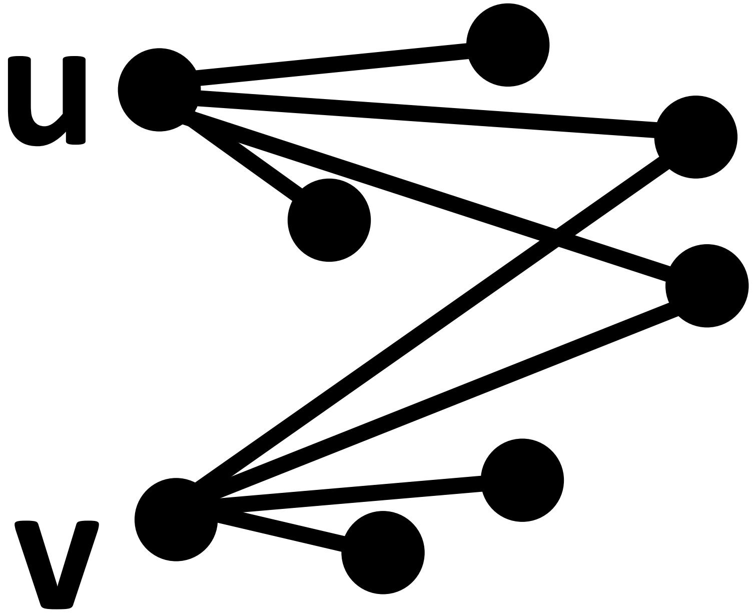
2

$|X \cap Y|$

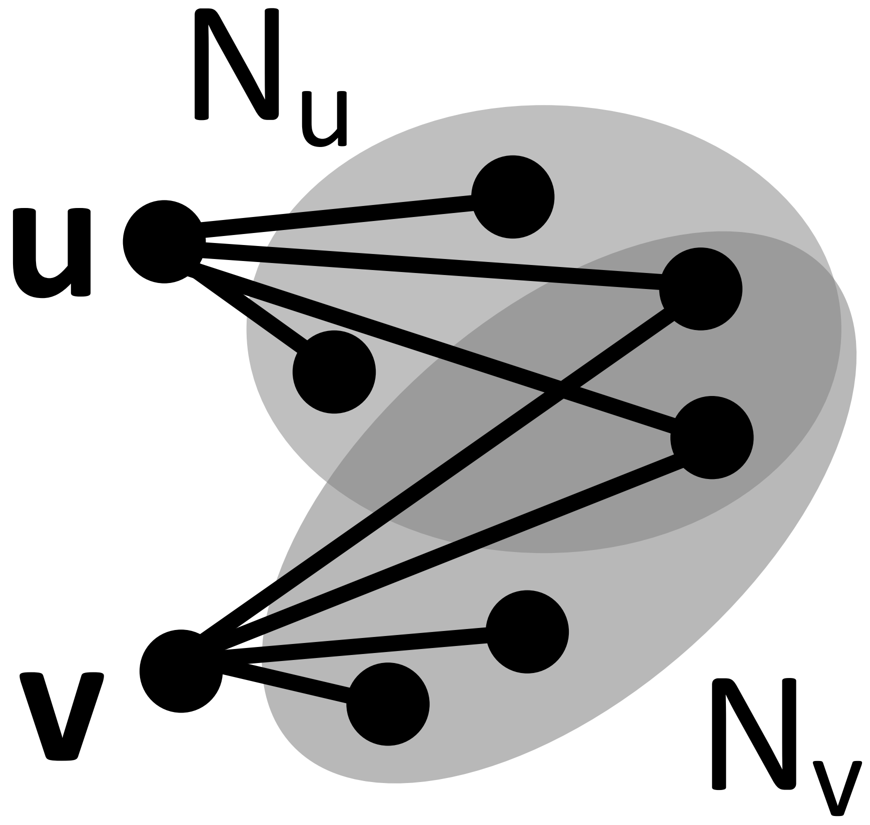
We greatly accelerate $|X \cap Y|$ with BFs

ProbGraph key idea, continued

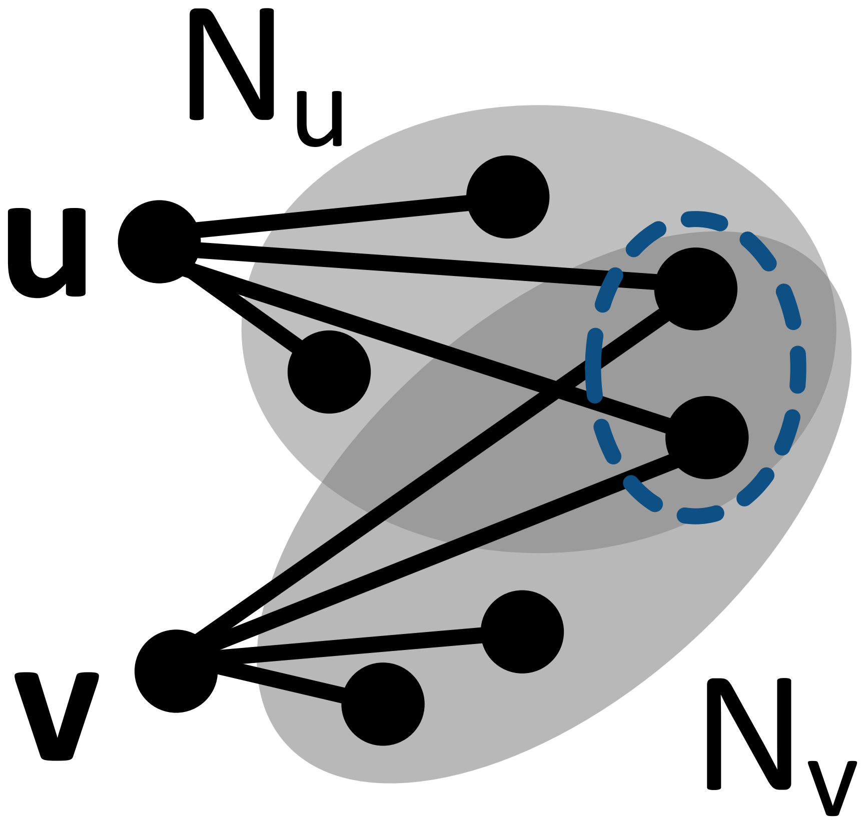
ProbGraph key idea, continued



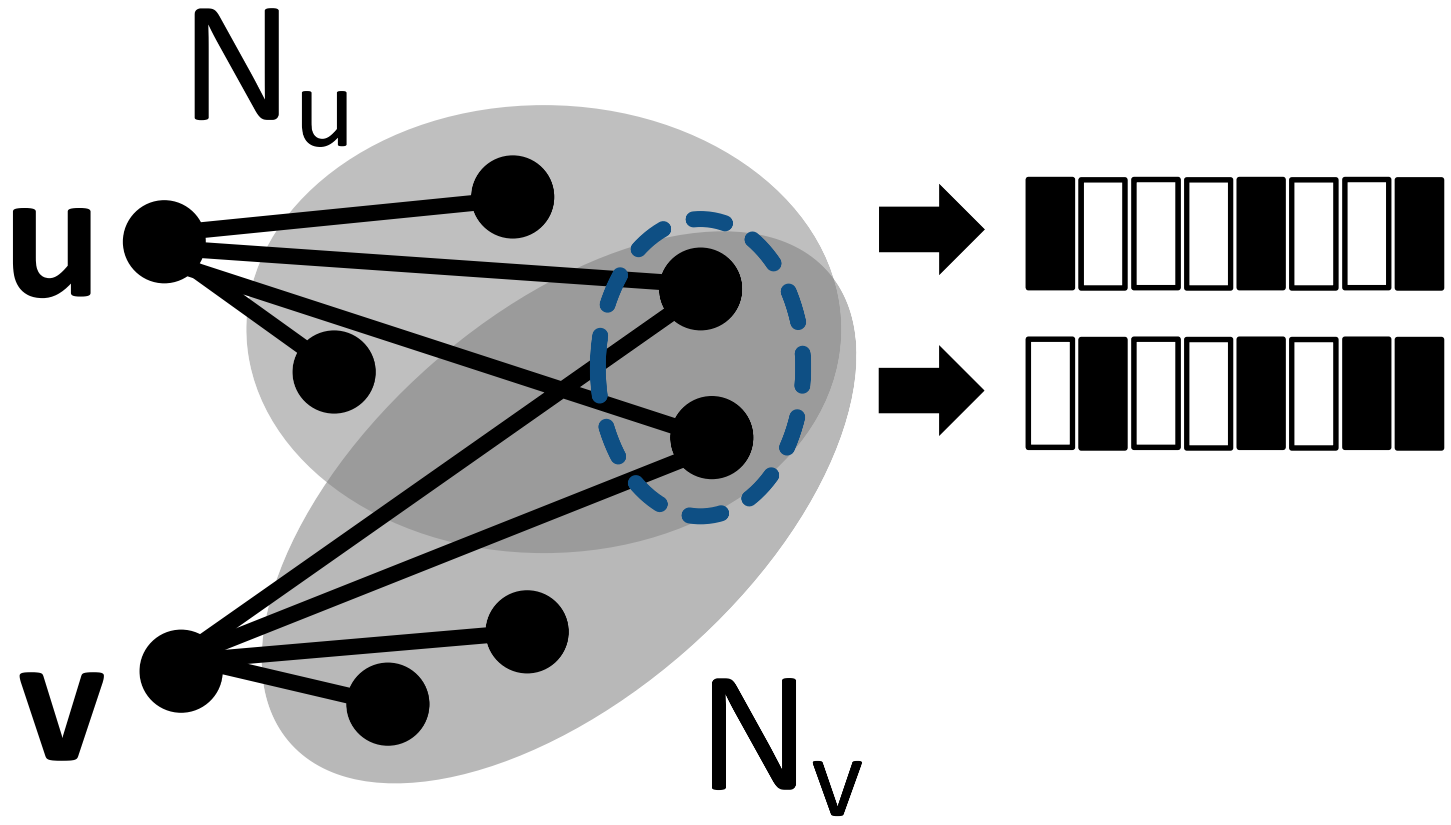
ProbGraph key idea, continued



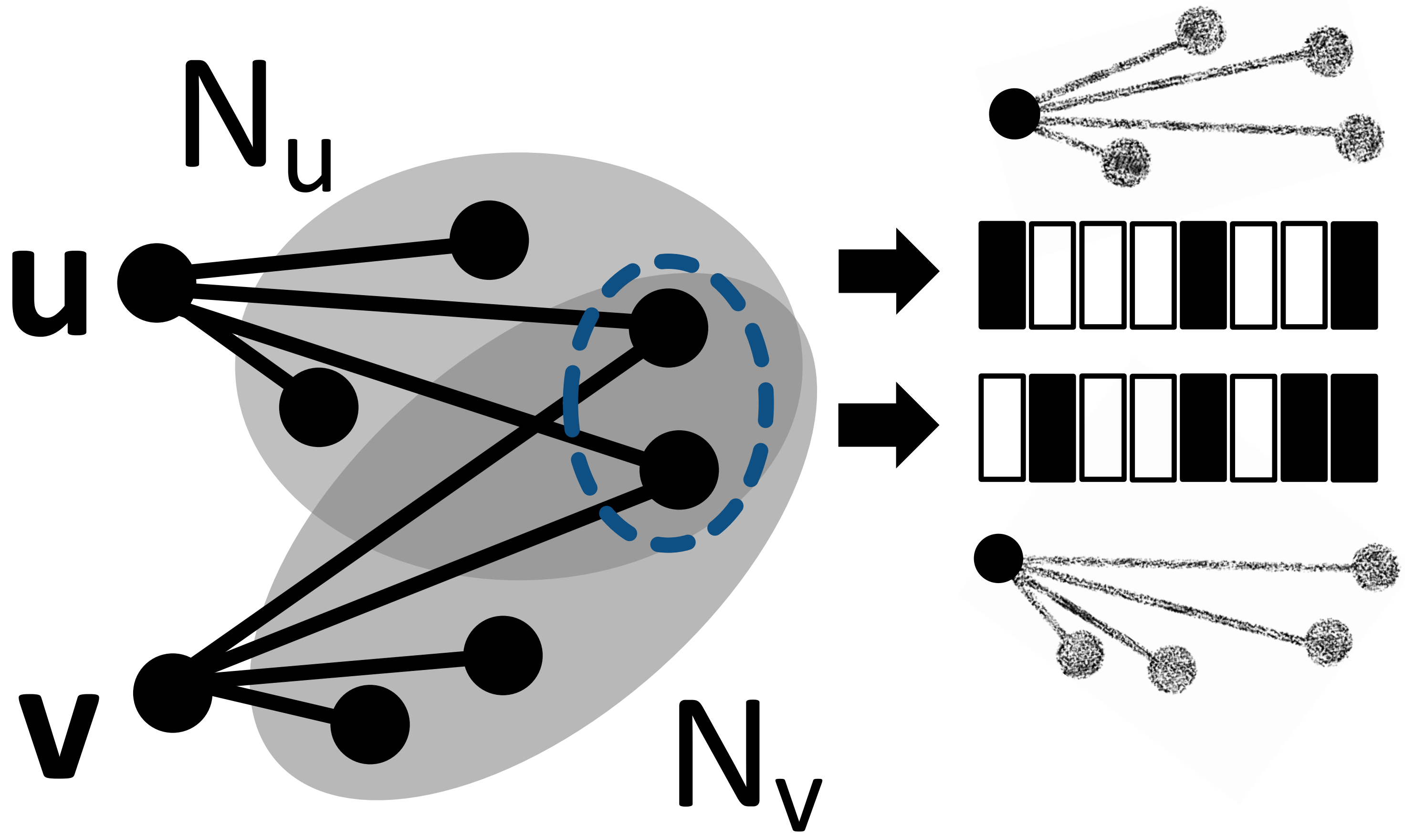
ProbGraph key idea, continued



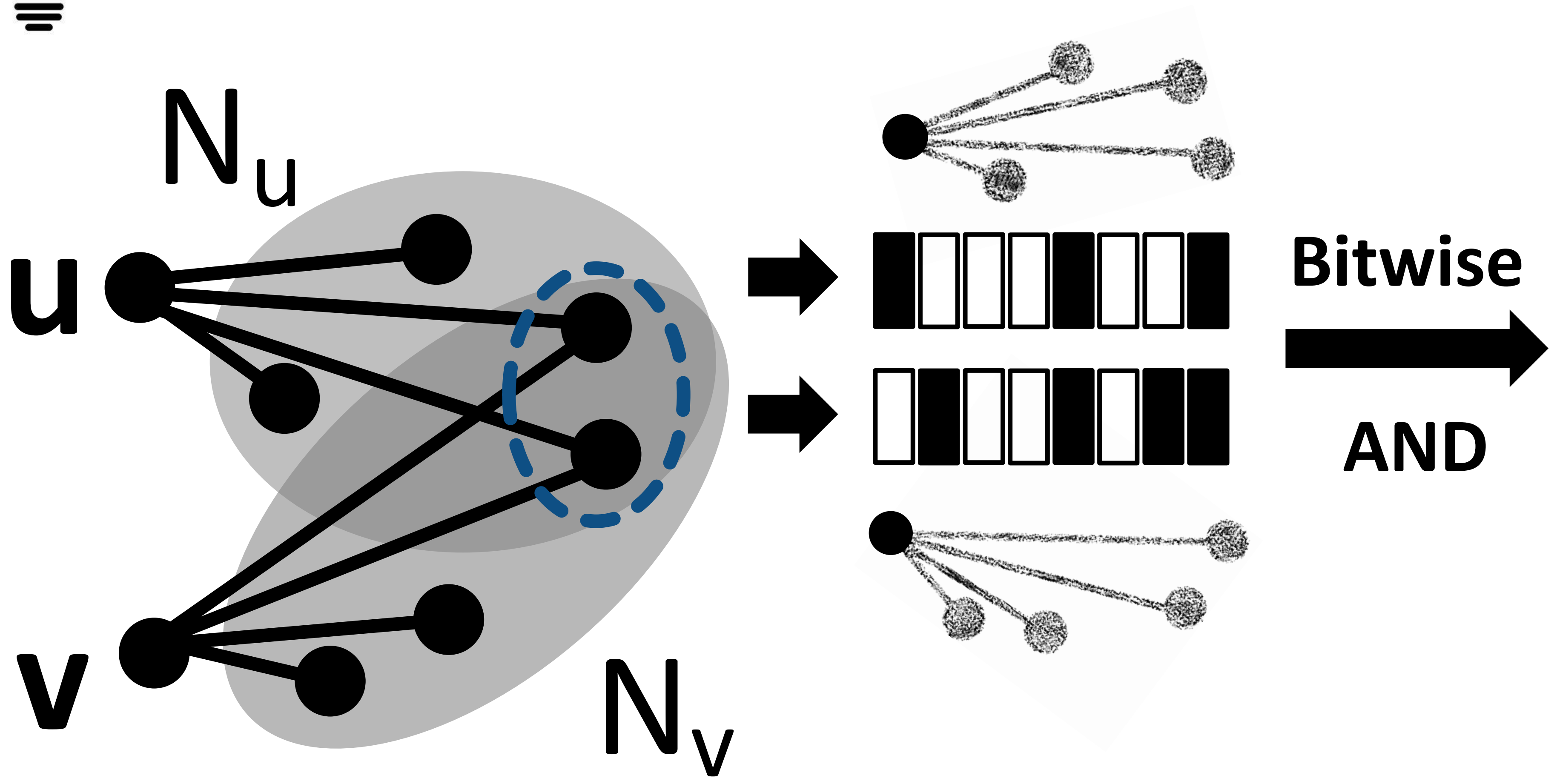
ProbGraph key idea, continued



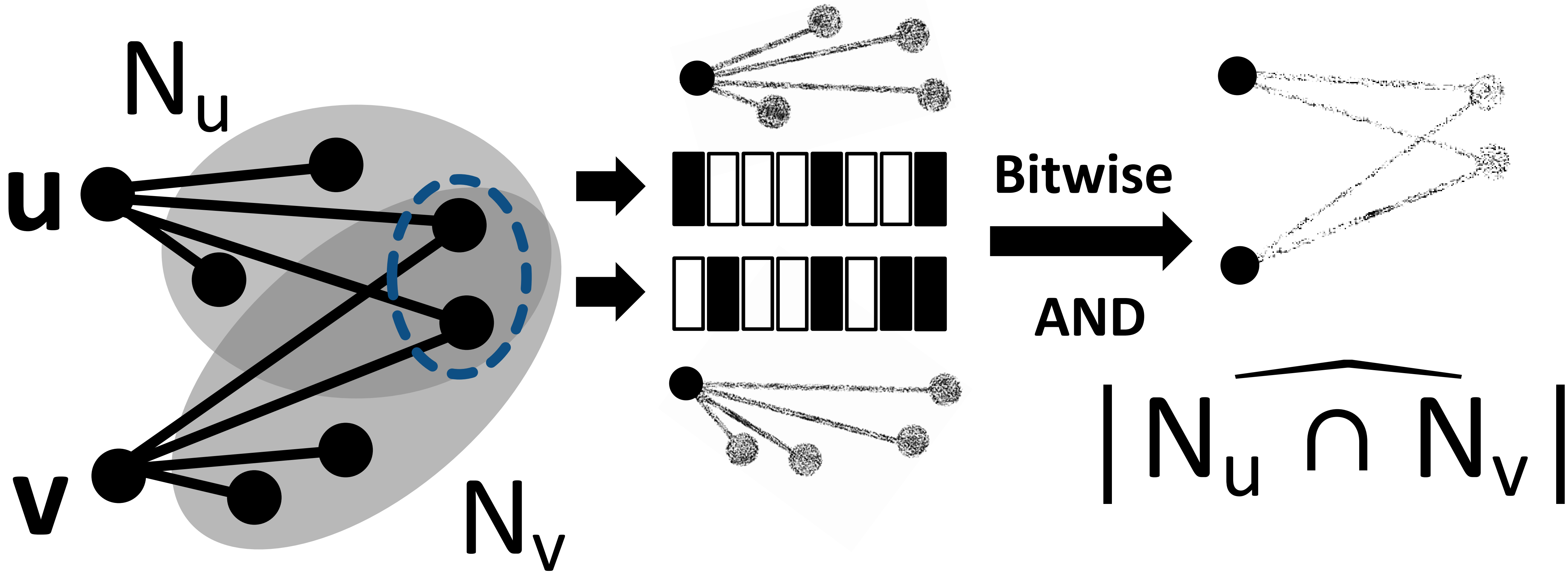
ProbGraph key idea, continued



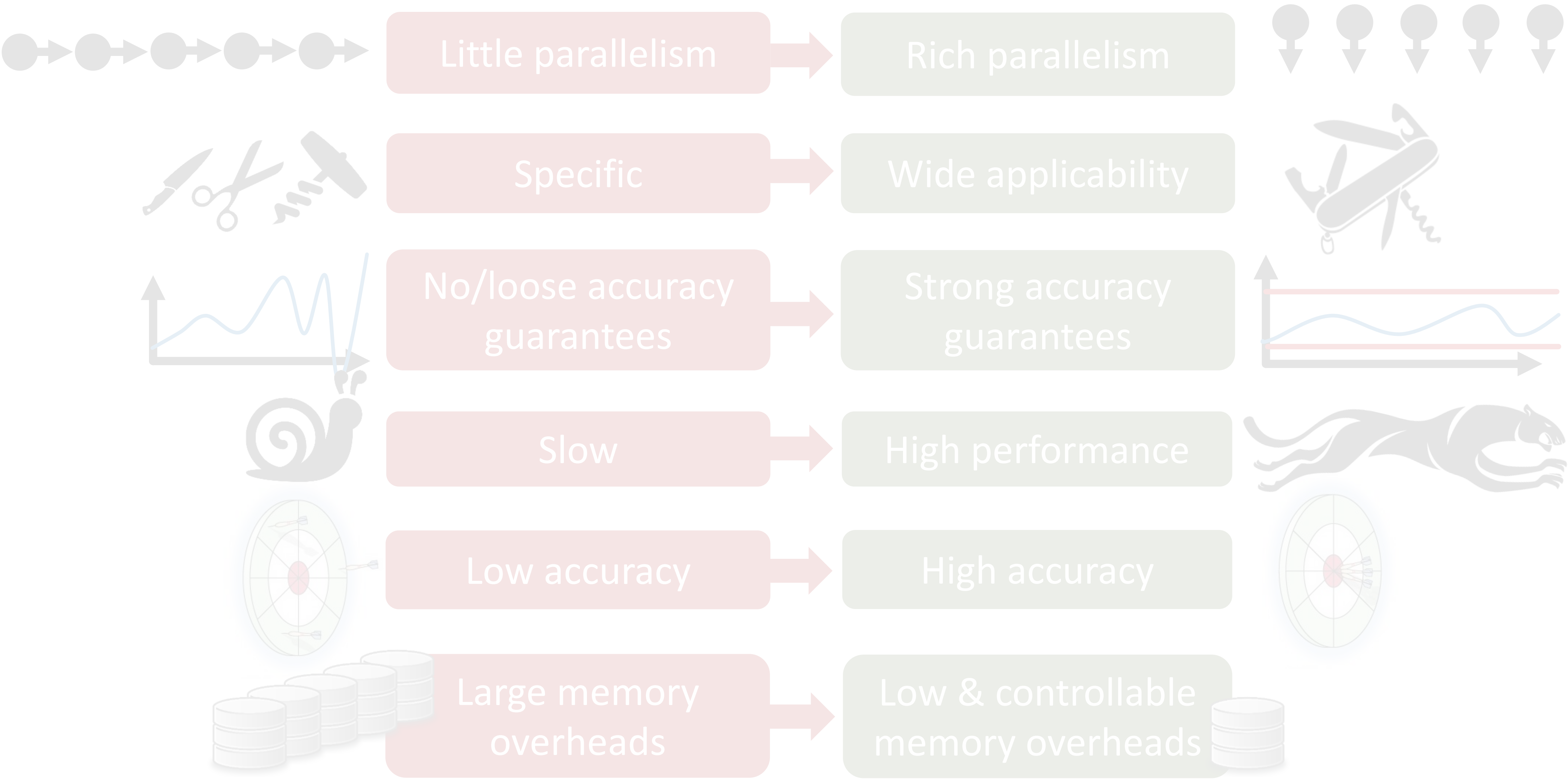
ProbGraph key idea, continued



ProbGraph key idea, continued



Approximate Graph Processing: Our Objectives



Little parallelism → Rich parallelism

Specific → Wide applicability

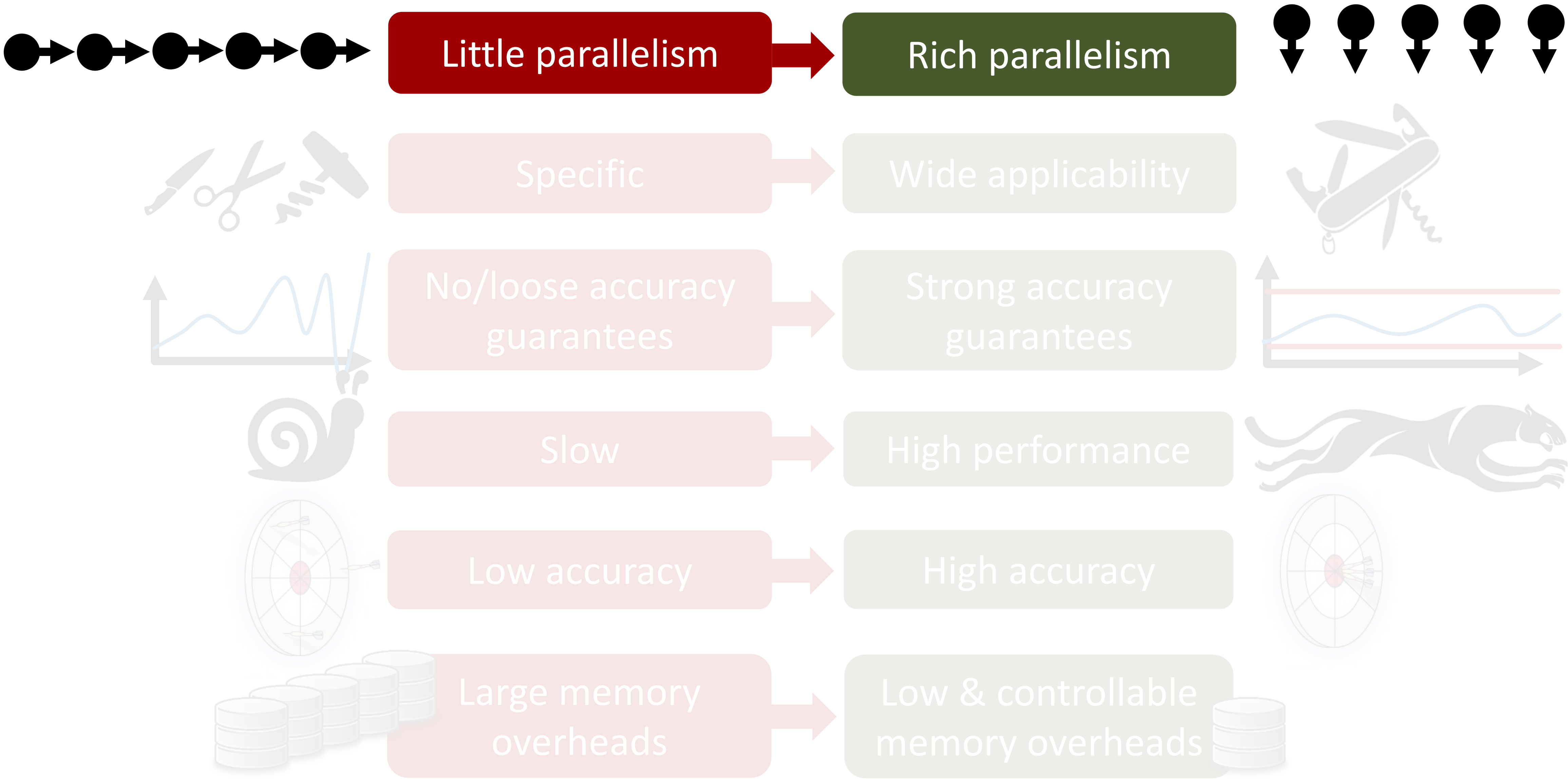
No/loose accuracy guarantees → Strong accuracy guarantees

Slow → High performance

Low accuracy → High accuracy

Large memory overheads → Low & controllable memory overheads

Approximate Graph Processing: Our Objectives



Little parallelism → Rich parallelism

Specific → Wide applicability

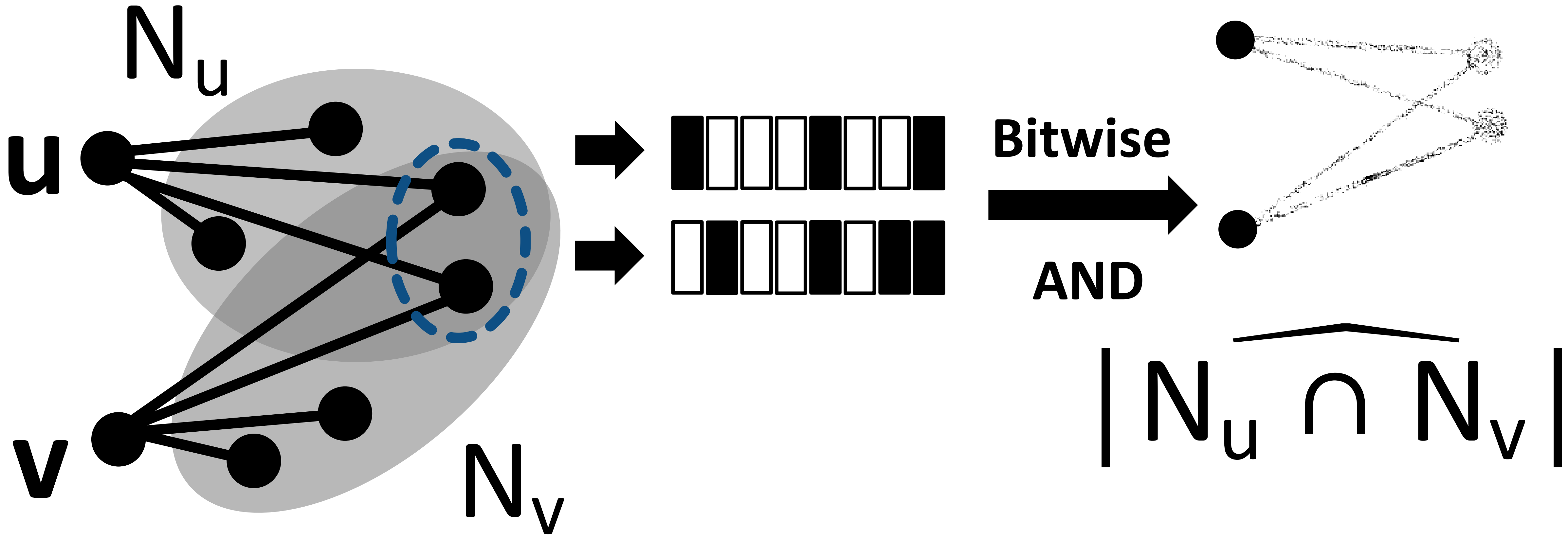
No/loose accuracy guarantees → Strong accuracy guarantees

Slow → High performance

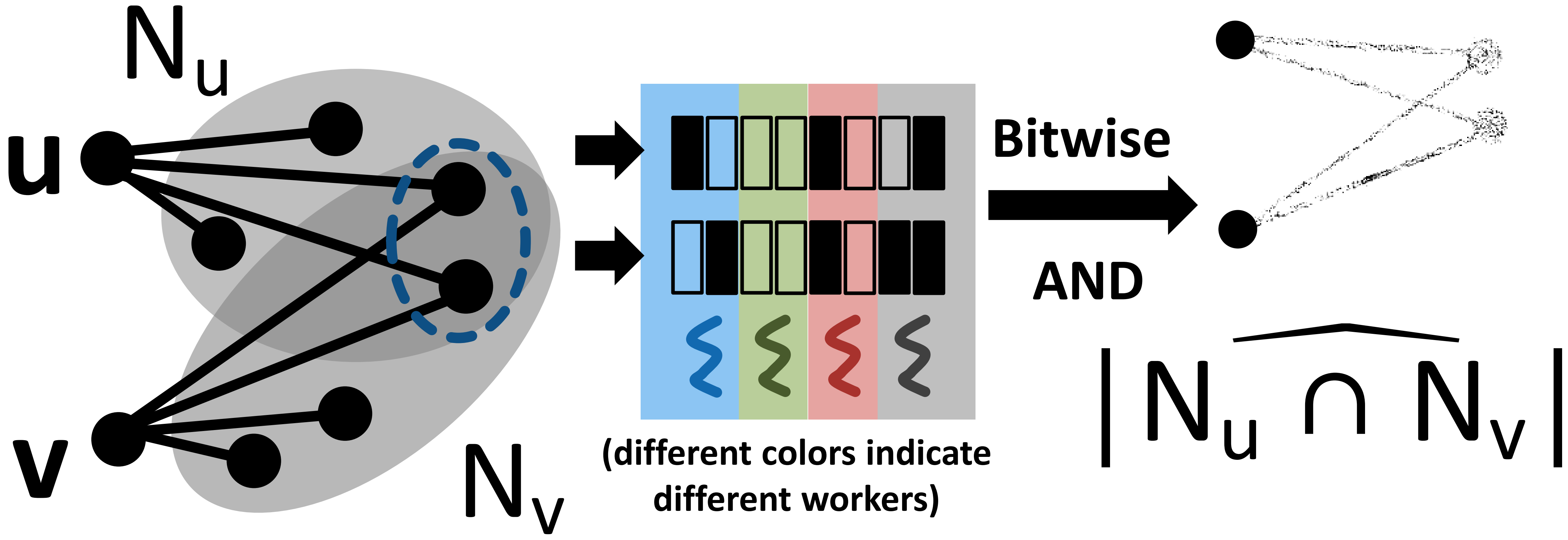
Low accuracy → High accuracy

Large memory overheads → Low & controllable memory overheads

ProbGraph: Fast & Parallel Execution

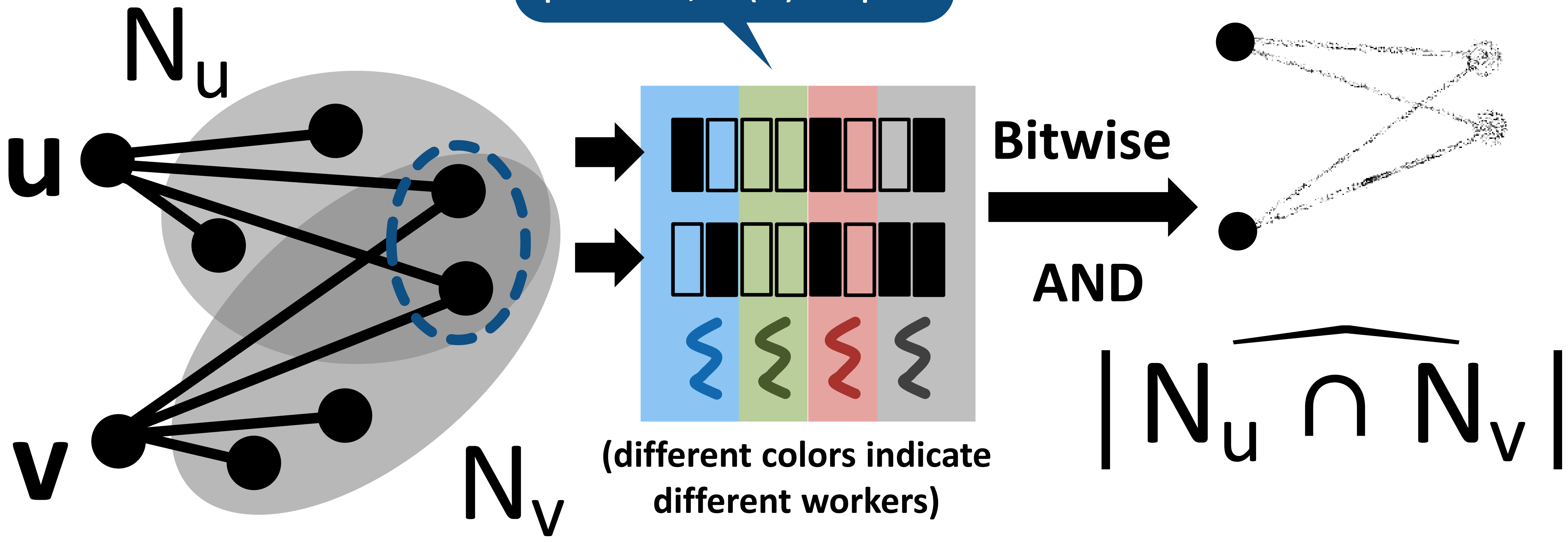


ProbGraph: Fast & Parallel Execution



(different colors indicate different workers)

ProbGraph: Fast & Parallel Execution



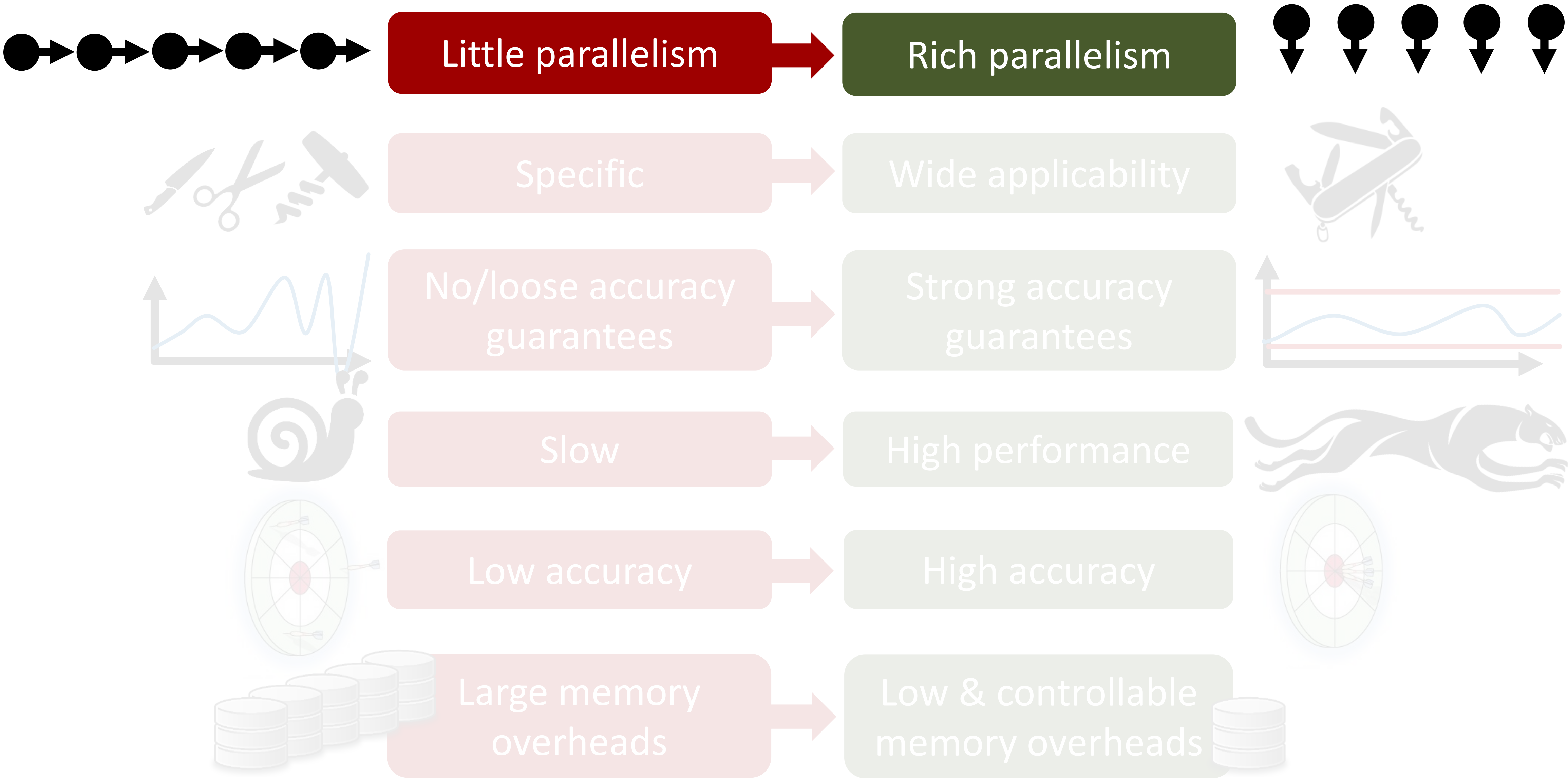
Embarrassingly parallel, O(1) depth

Bitwise AND

(different colors indicate different workers)

$$|\widehat{N_u \cap N_v}|$$

Approximate Graph Processing: Our Objectives



Little parallelism → Rich parallelism

Specific → Wide applicability

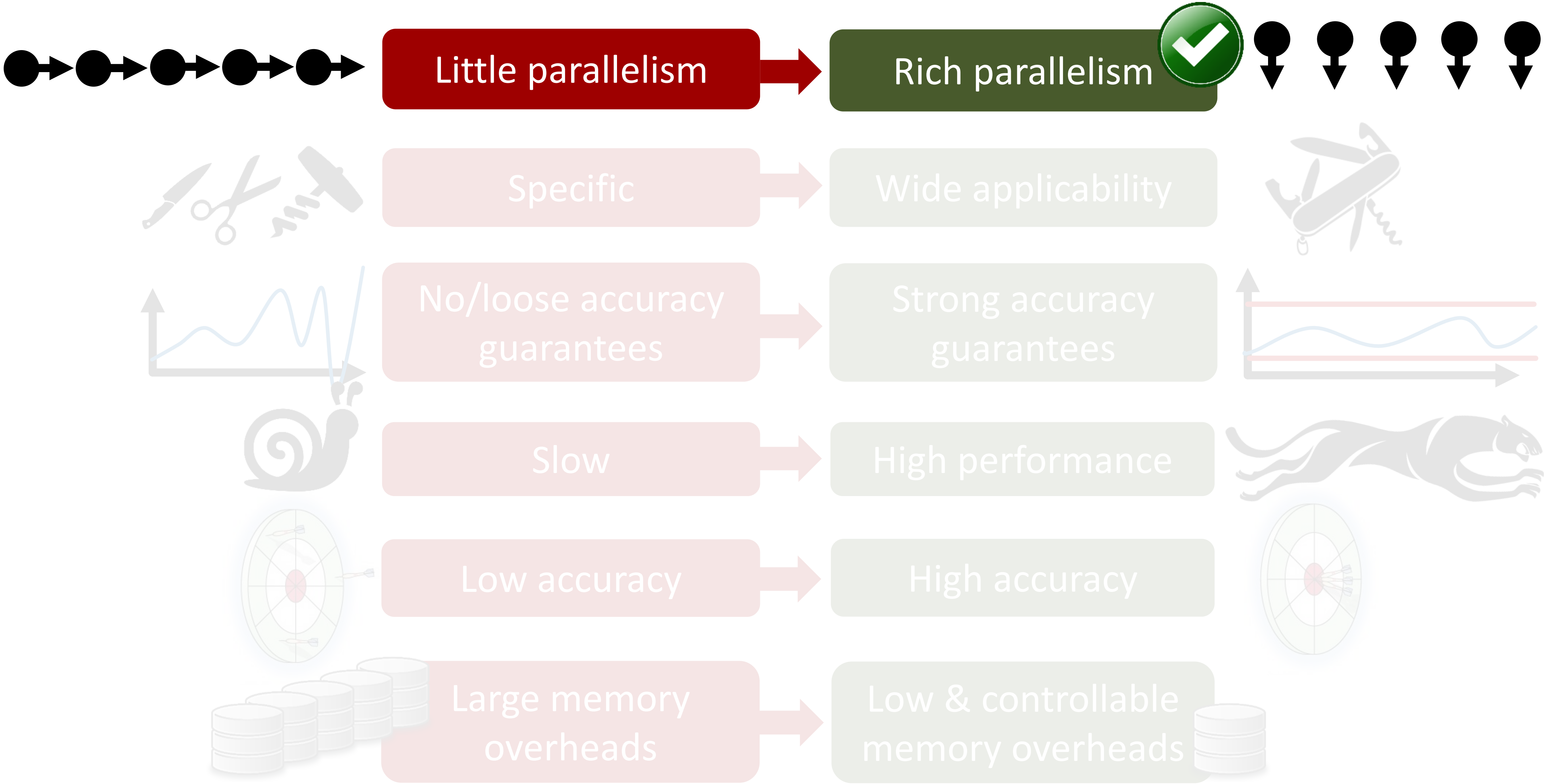
No/loose accuracy guarantees → Strong accuracy guarantees

Slow → High performance

Low accuracy → High accuracy

Large memory overheads → Low & controllable memory overheads

Approximate Graph Processing: Our Objectives



Little parallelism → Rich parallelism

Specific → Wide applicability

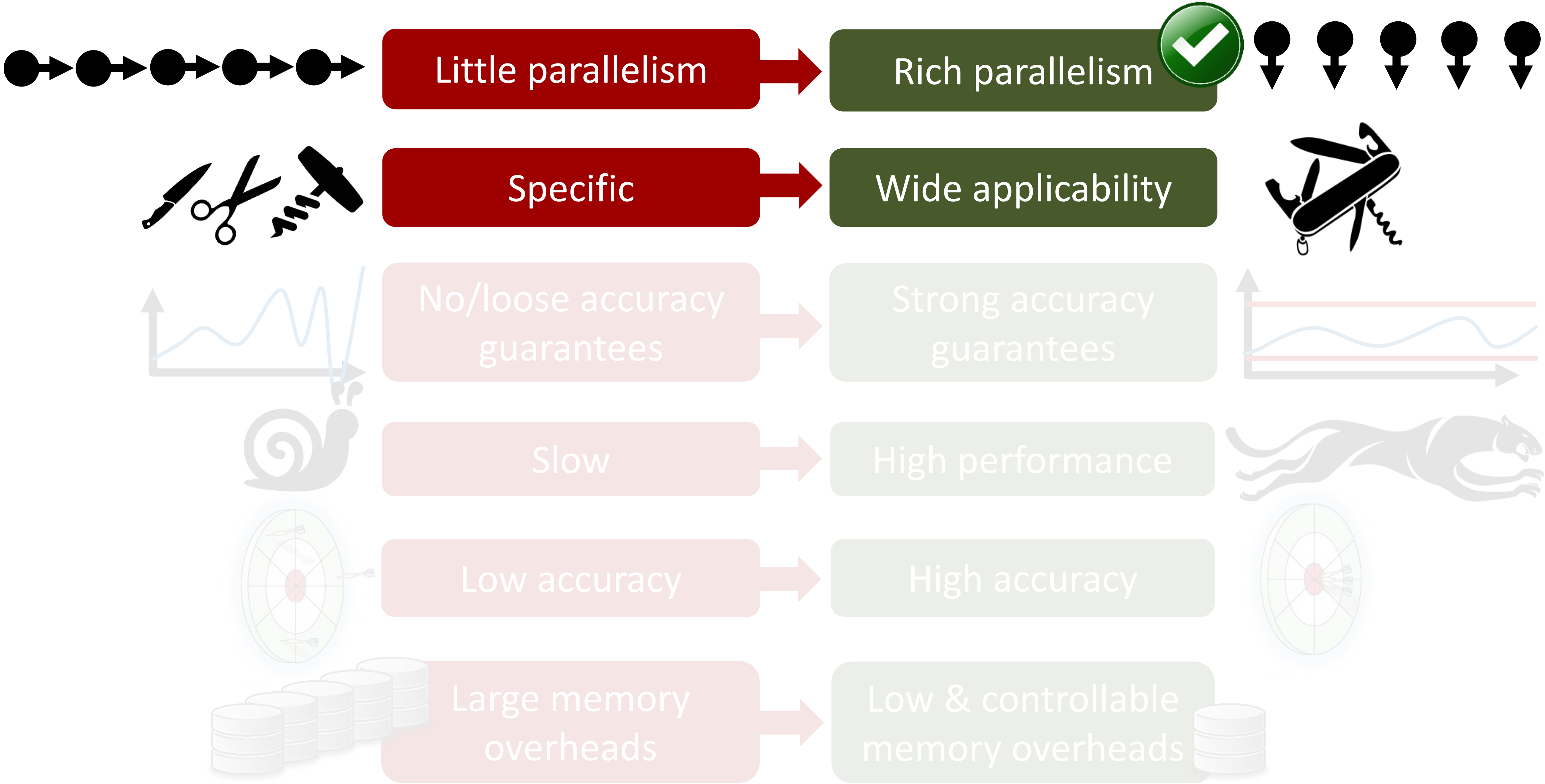
No/loose accuracy guarantees → Strong accuracy guarantees

Slow → High performance

Low accuracy → High accuracy

Large memory overheads → Low & controllable memory overheads

Approximate Graph Processing: Our Objectives



Little parallelism → Rich parallelism

Specific → Wide applicability

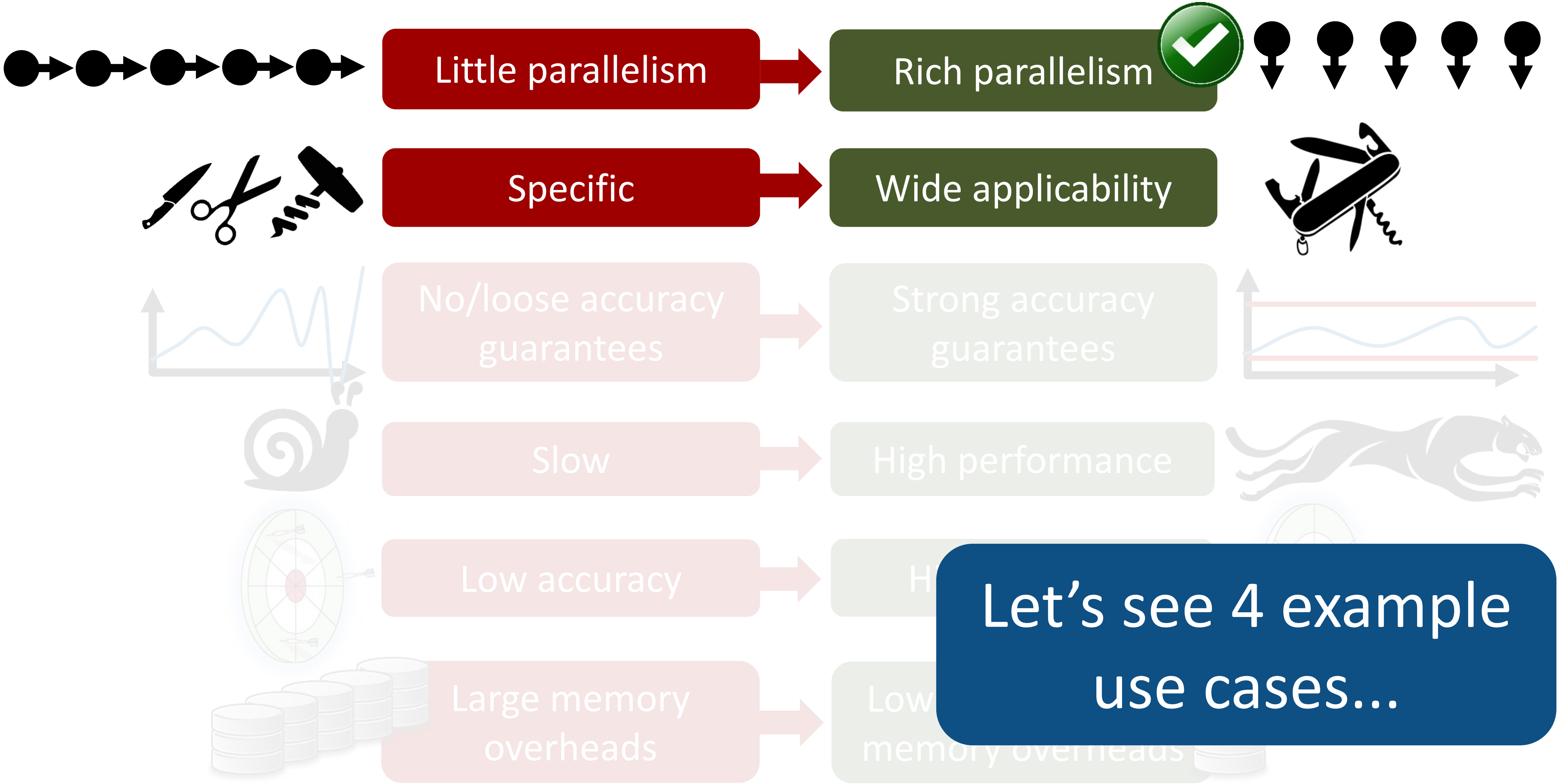
No/loose accuracy guarantees → Strong accuracy guarantees

Slow → High performance

Low accuracy → High accuracy

Large memory overheads → Low & controllable memory overheads

Approximate Graph Processing: Our Objectives



Little parallelism → Rich parallelism ✓

Specific → Wide applicability

No/loose accuracy guarantees → Strong accuracy guarantees

Slow → High performance

Low accuracy → H

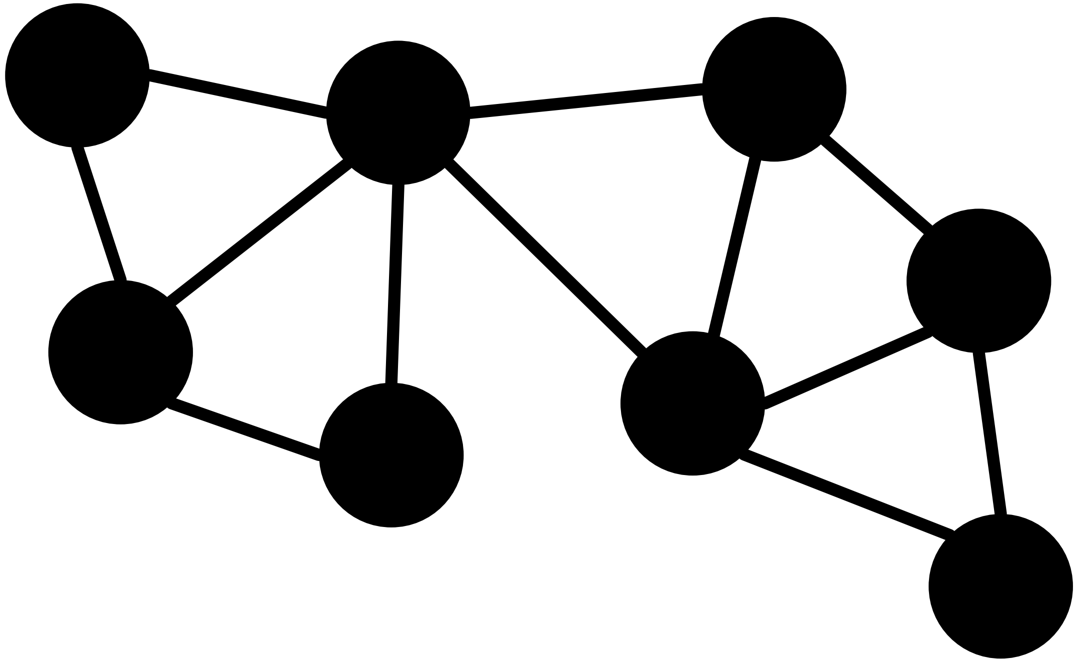
Large memory overheads → Low memory overheads

Let’s see 4 example use cases...

Use Case 1: Link Prediction

Which links will appear?

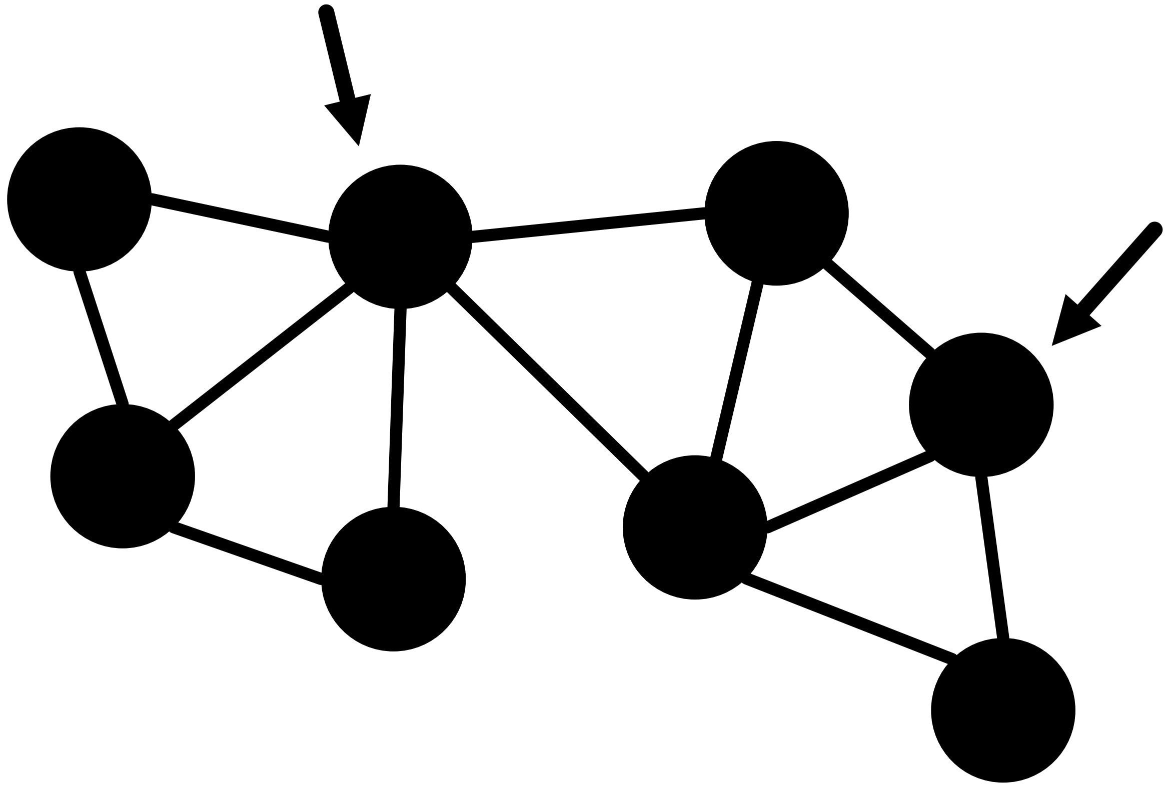
Which links are missing?



Use Case 1: Link Prediction

Which links will appear?

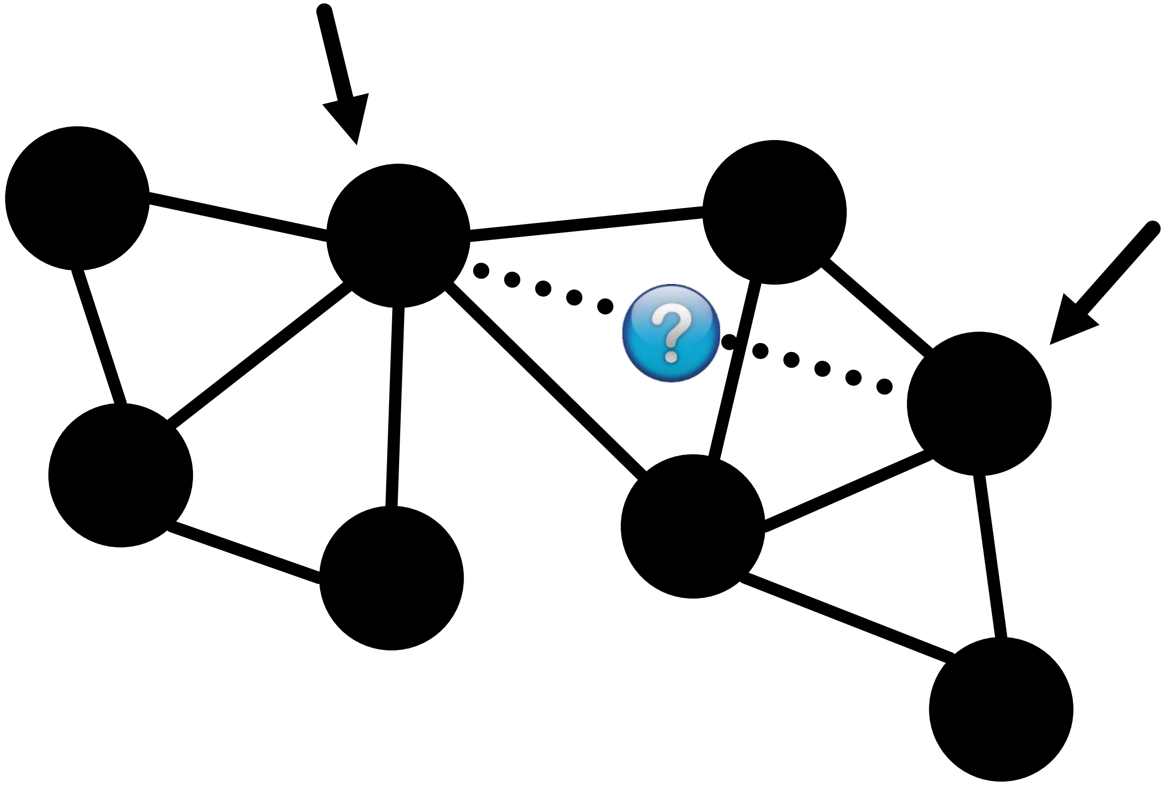
Which links are missing?



Use Case 1: Link Prediction

Which links will appear?

Which links are missing?



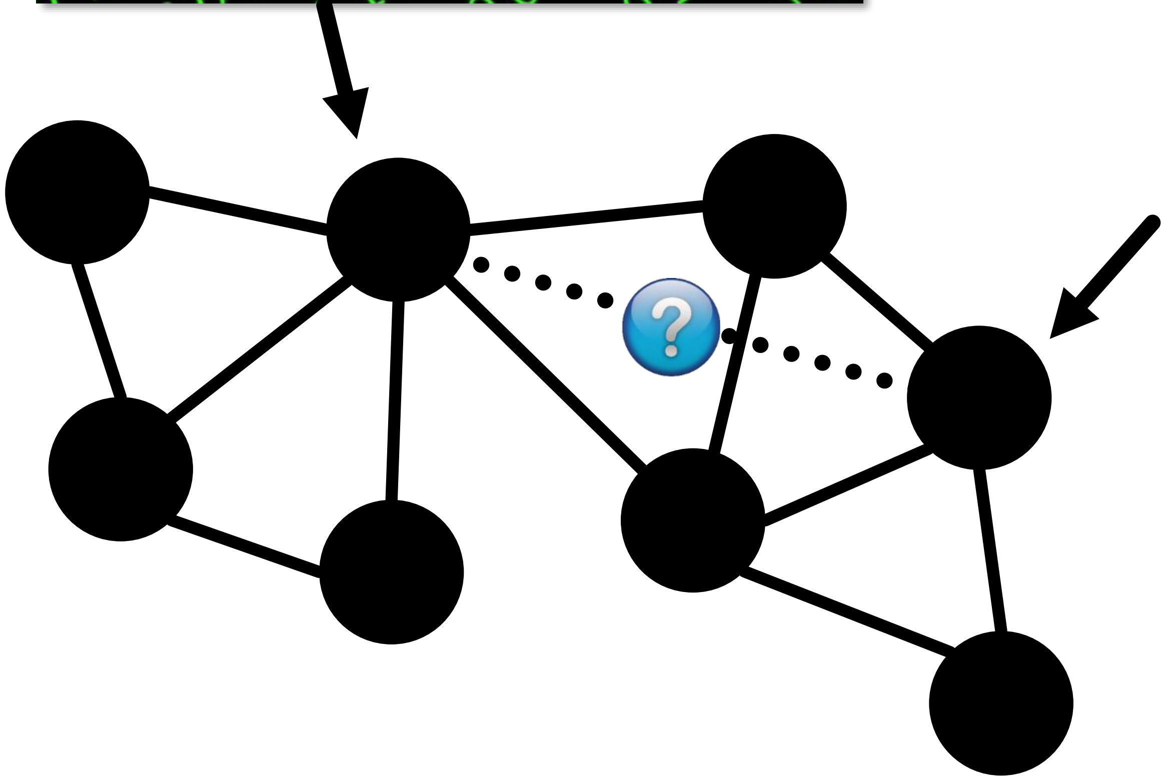
Use Case 1: Link Prediction

Which links will appear?

Which links are missing?

Fixing missing data

Predict future data



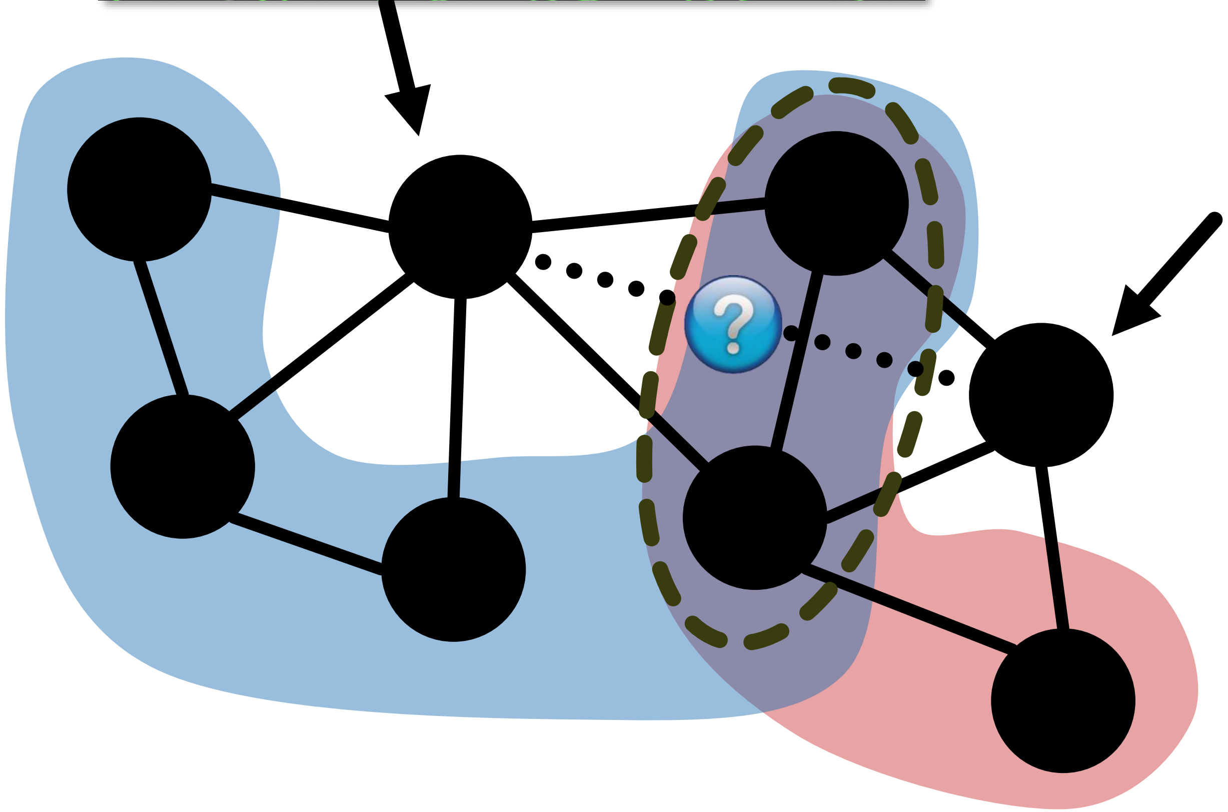
Use Case 1: Link Prediction

Which links will appear?

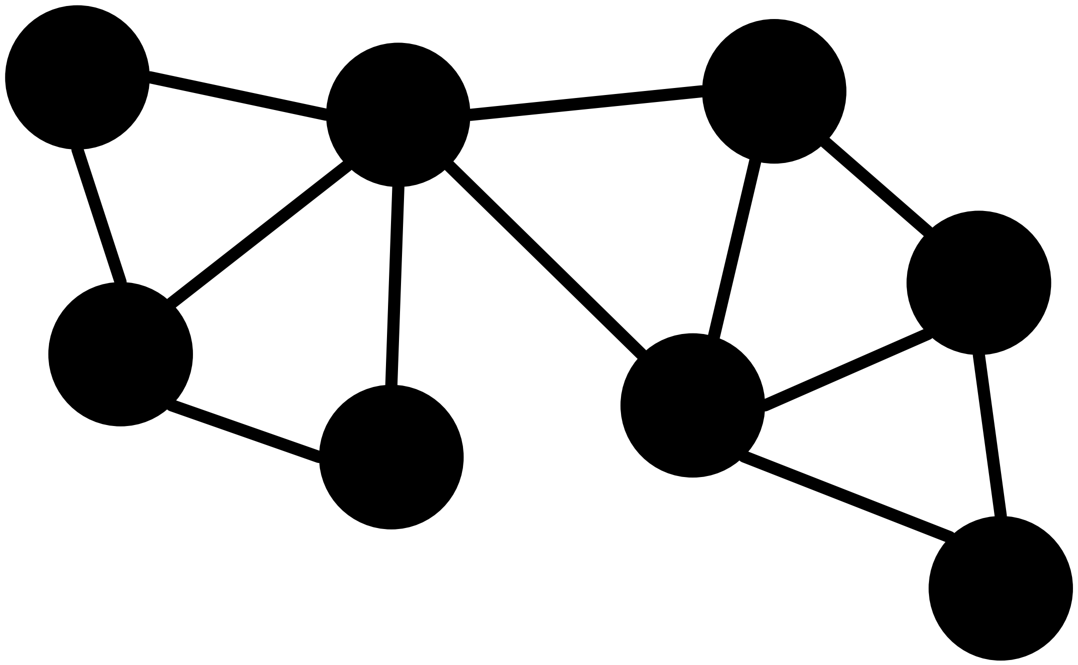
Which links are missing?

Fixing missing data

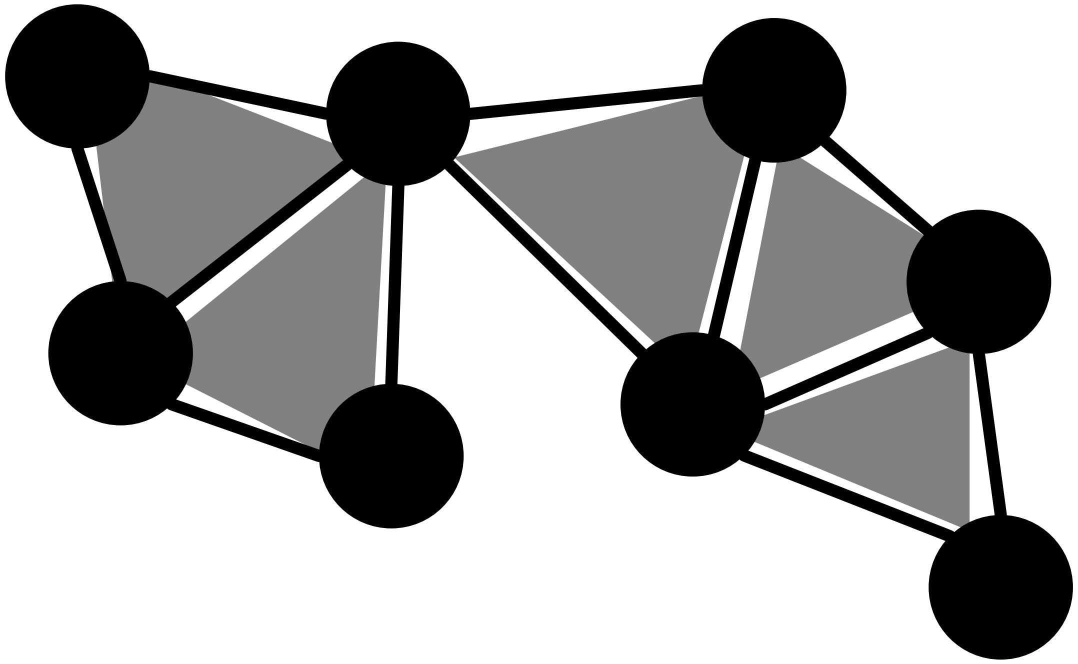
Predict future data



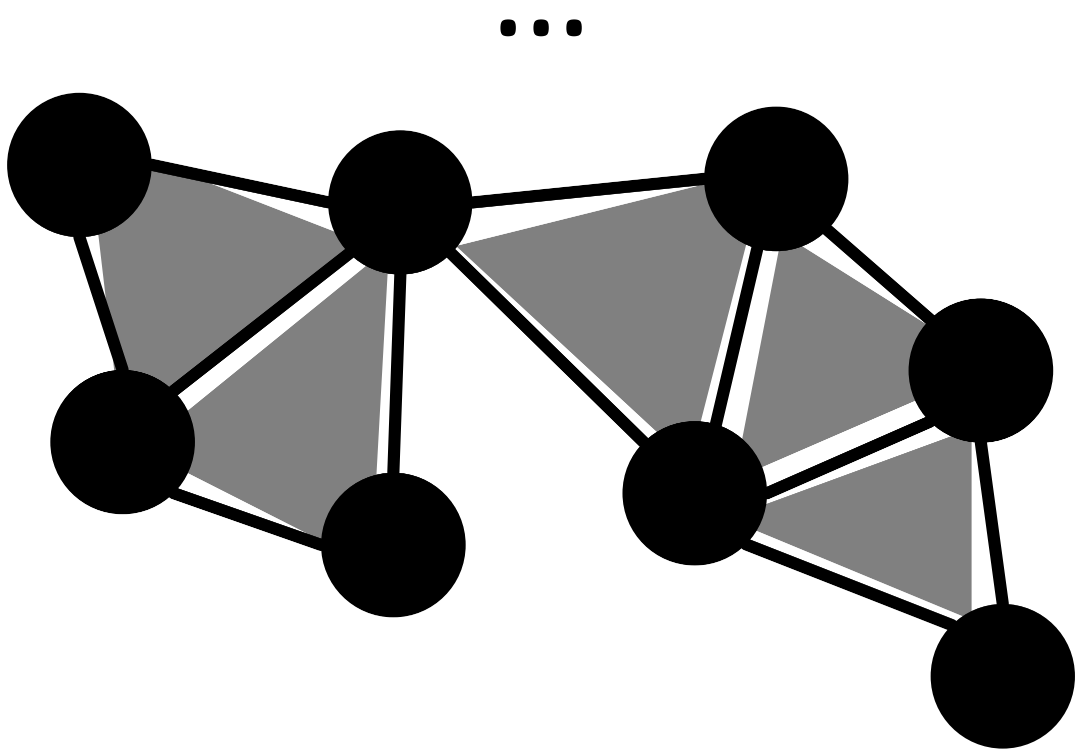
Use Case 2: Clique Counting



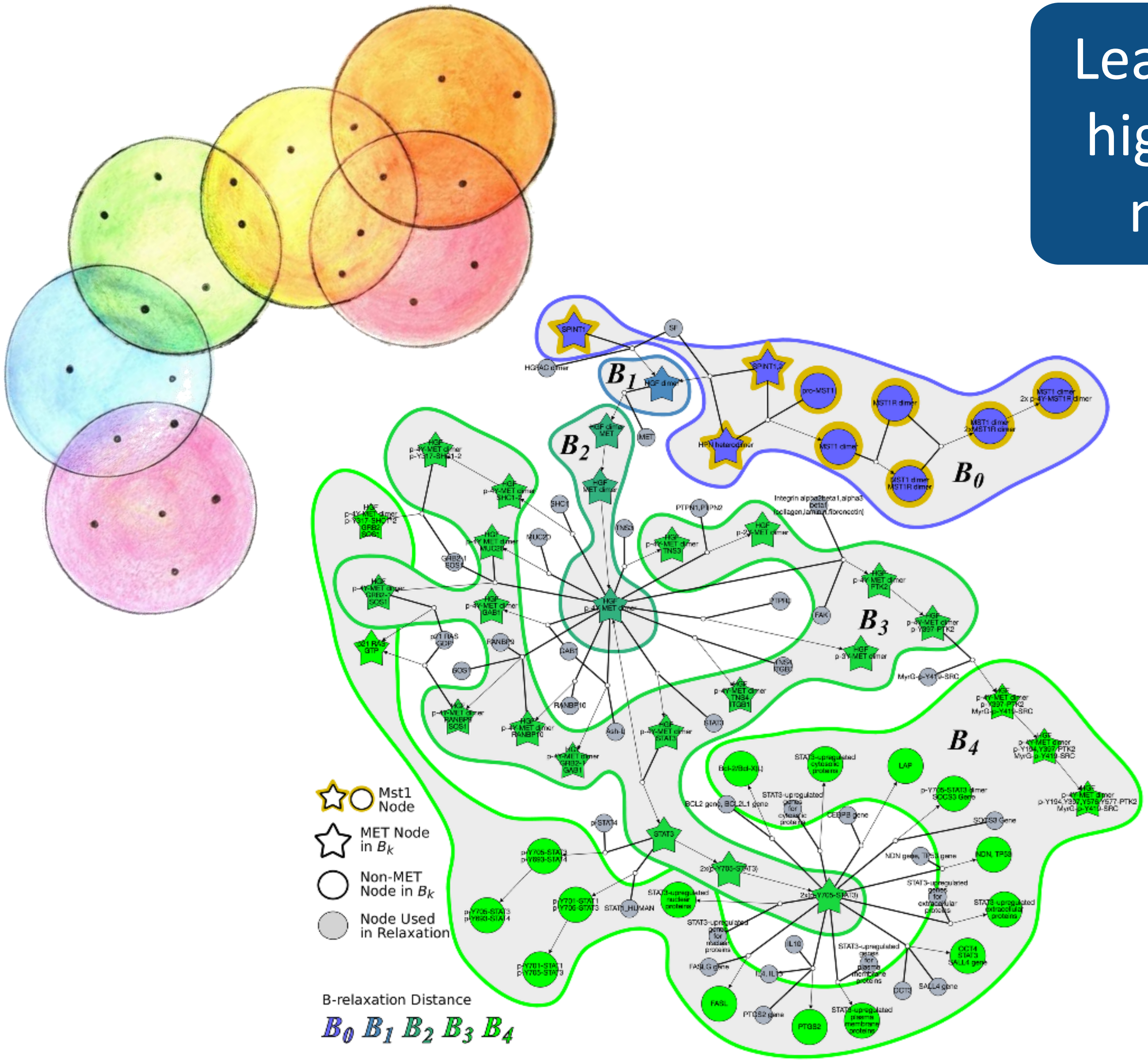
Use Case 2: Clique Counting



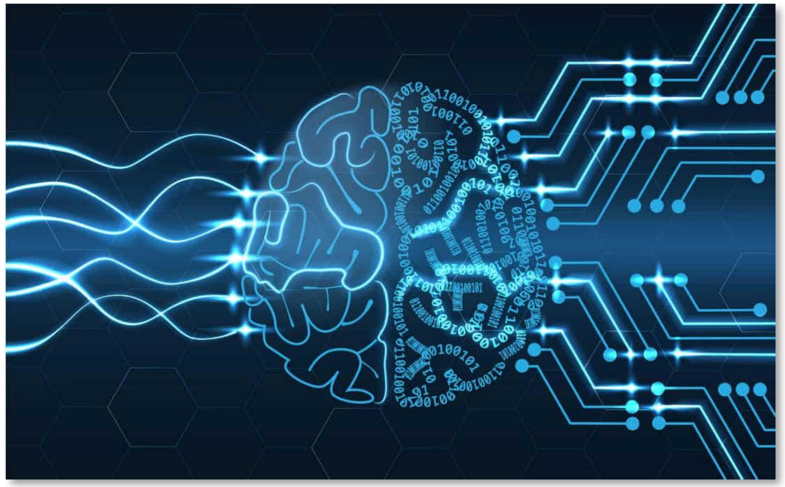
Use Case 2: Clique Counting



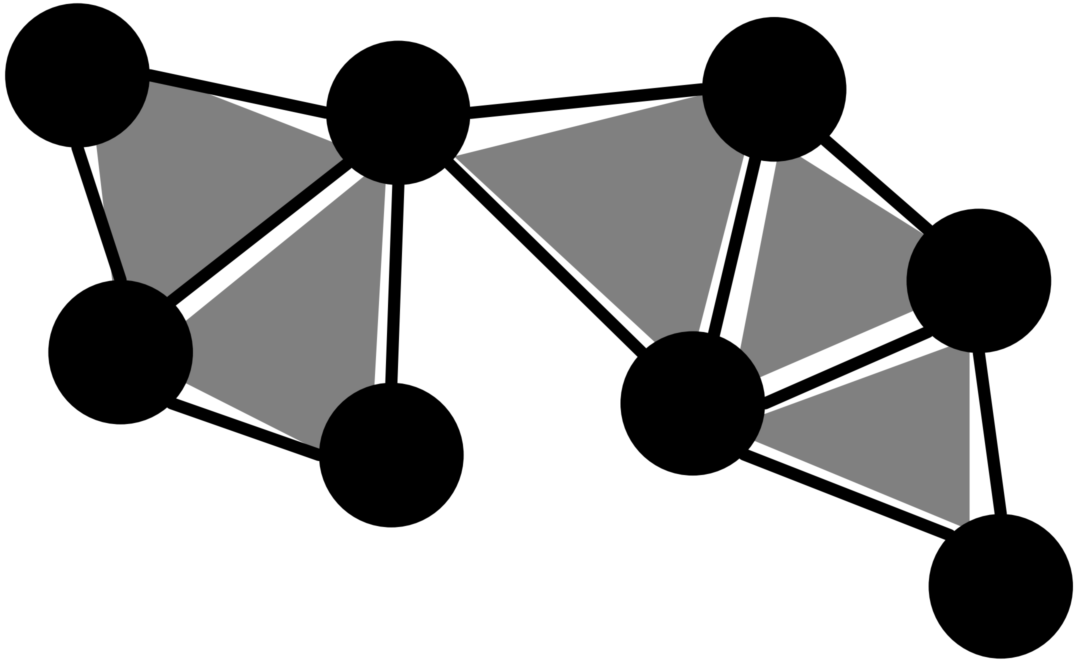
Use Case 2: Clique Counting



Learning over higher-order networks



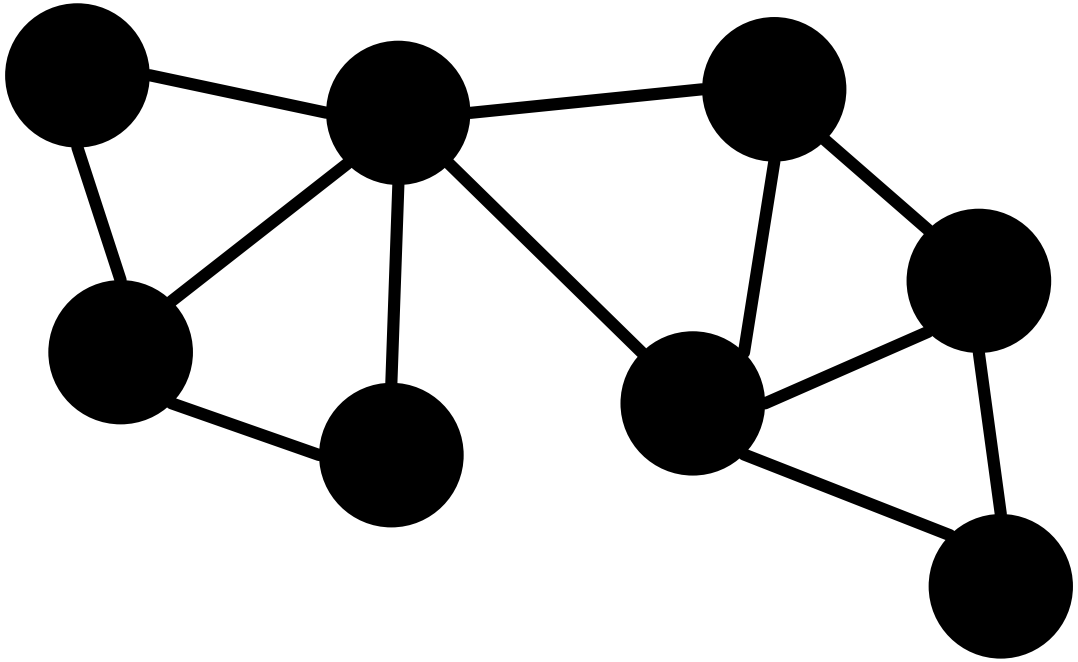
...



Use Case 3: Clustering

# Clusters?

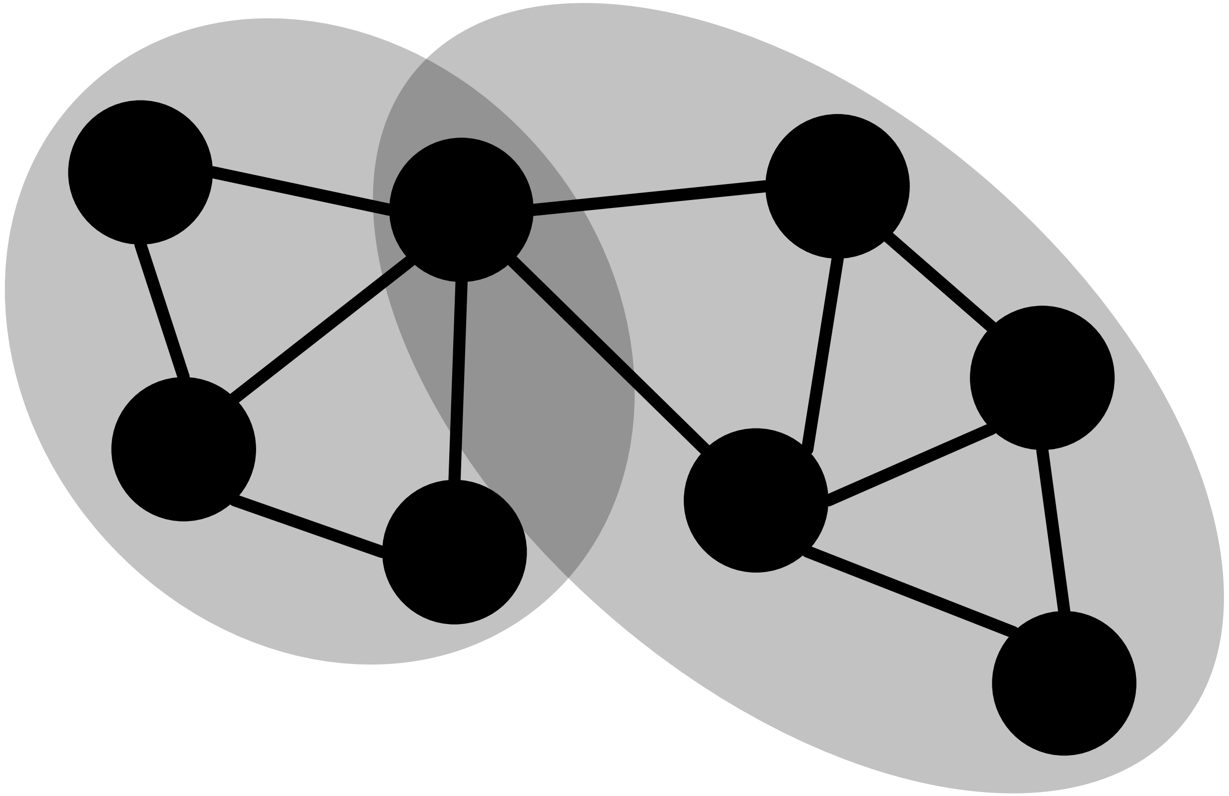
Structure of clusters?



Use Case 3: Clustering

# Clusters?

Structure of clusters?

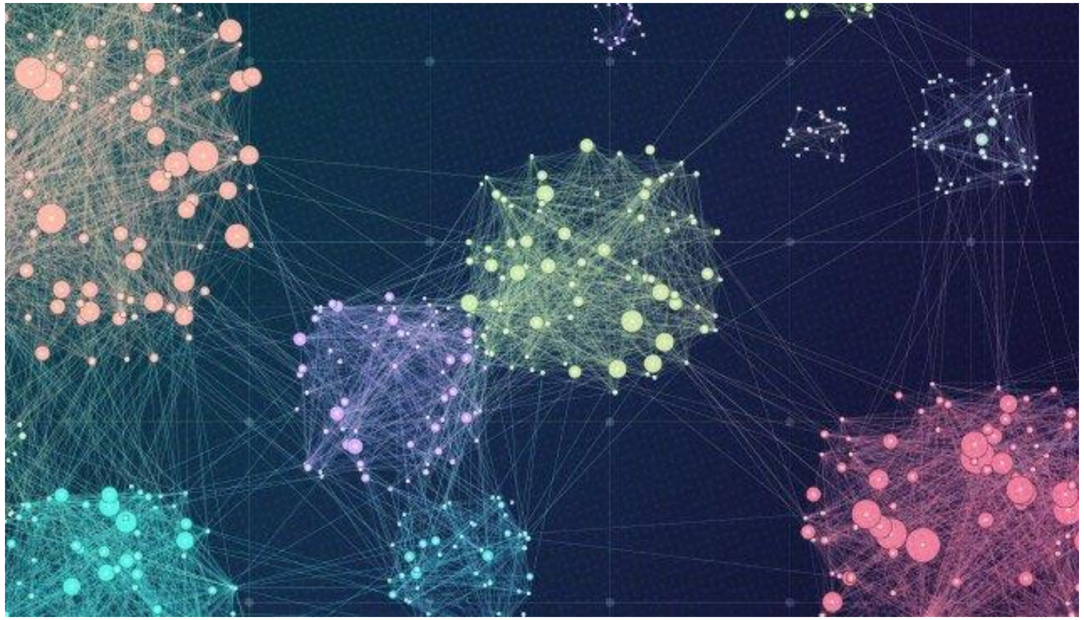


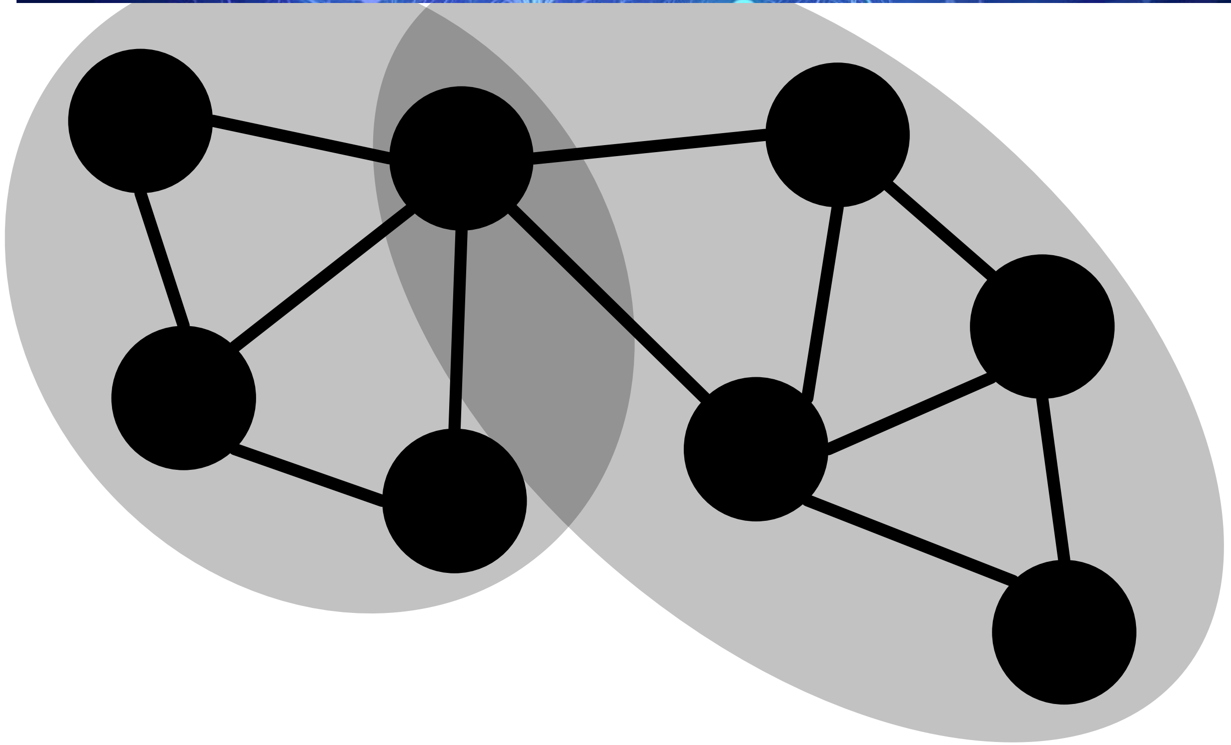
Use Case 3: Clustering

# Clusters?

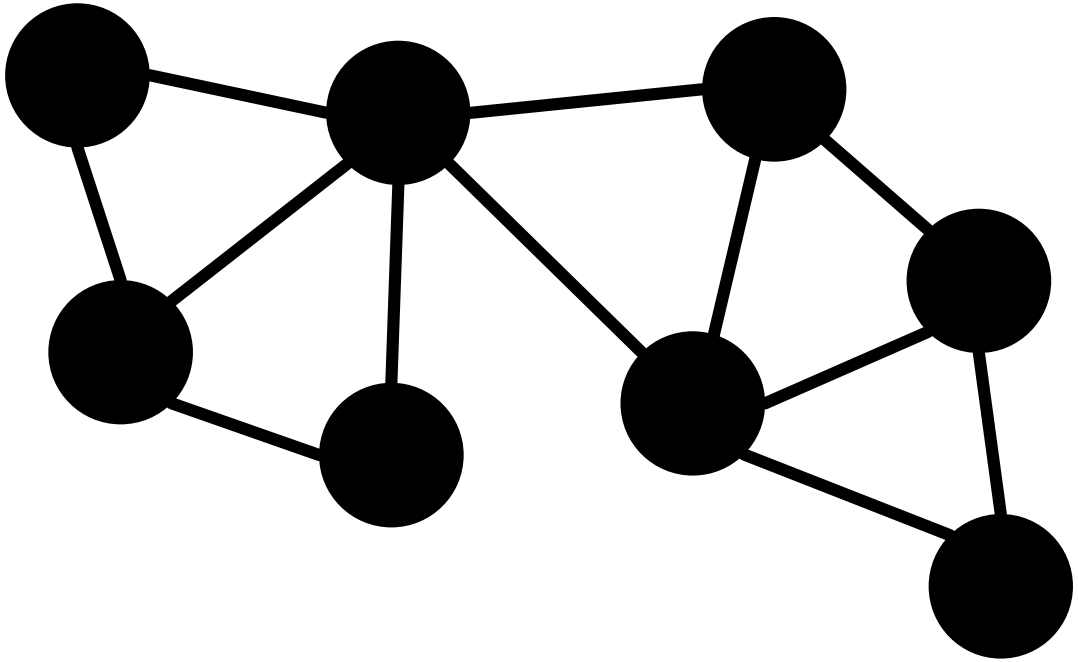
Structure of clusters?

Minibatch selection in Graph Neural Networks

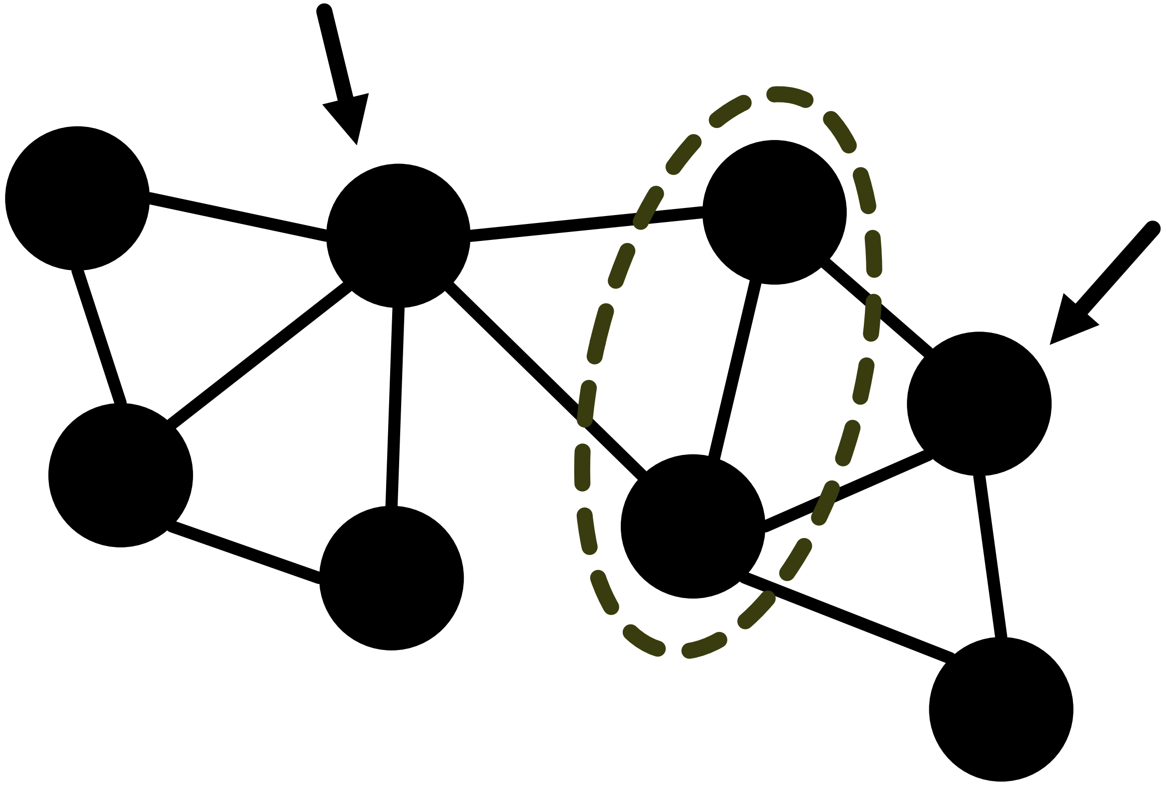




Use Case 4: Vertex Similarity

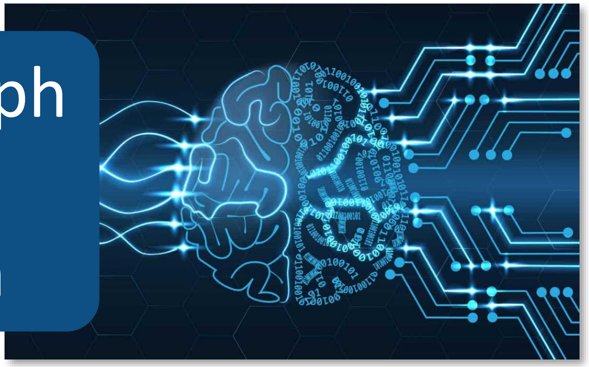


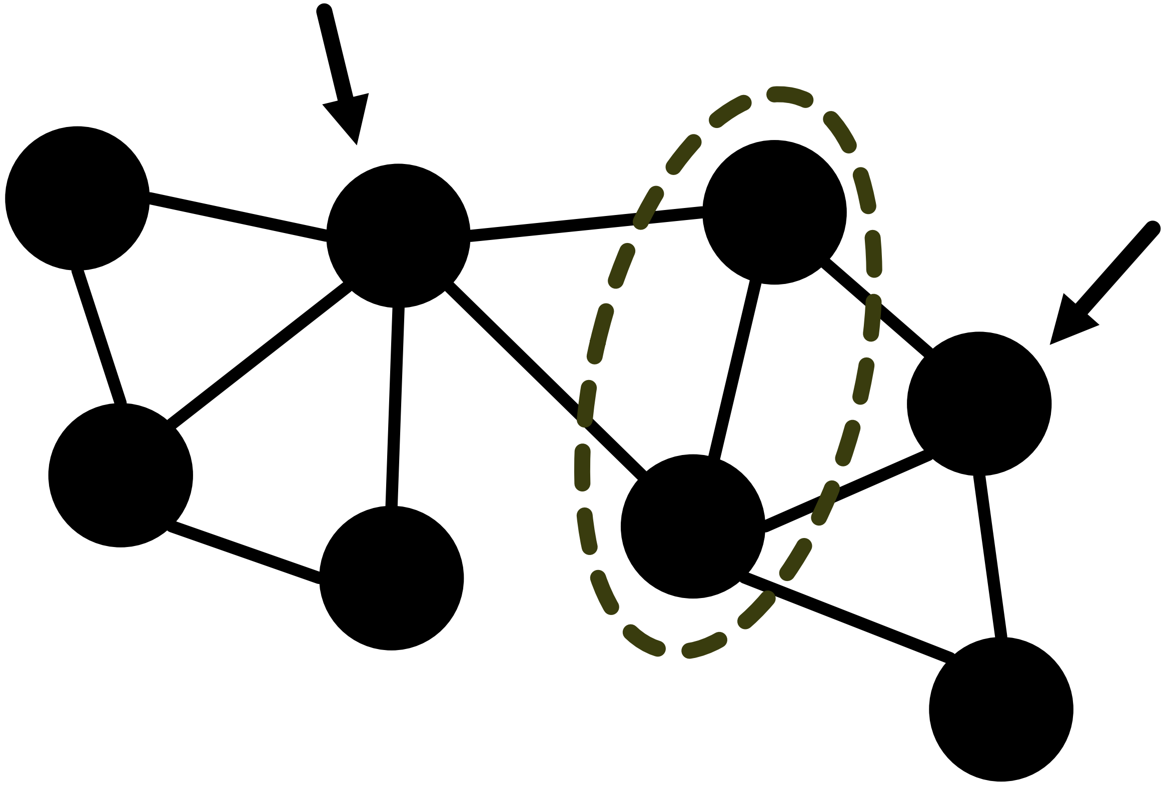
Use Case 4: Vertex Similarity



Use Case 4: Vertex Similarity

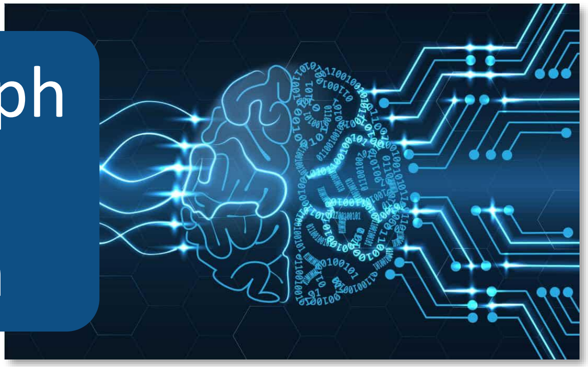
Enhancing graph embedding construction

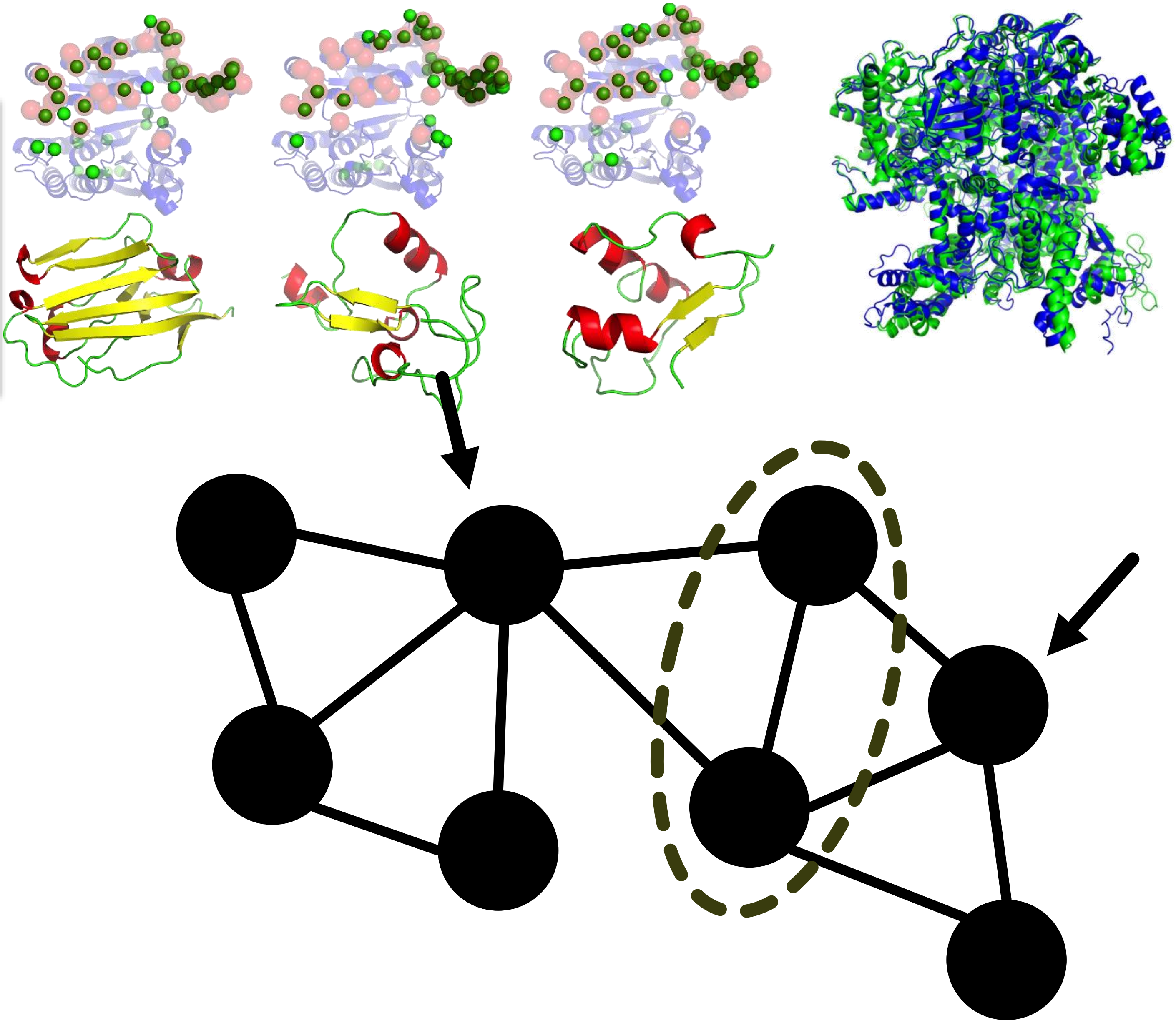




Use Case 4: Vertex Similarity

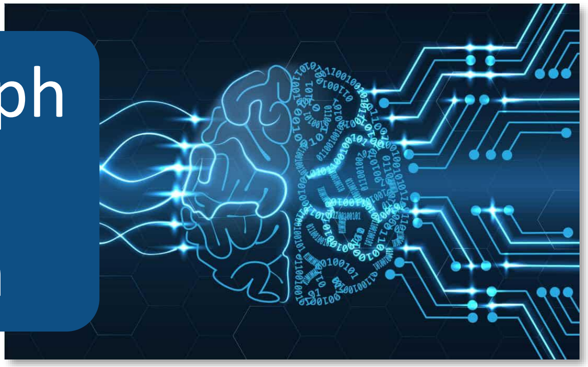
Enhancing graph embedding construction

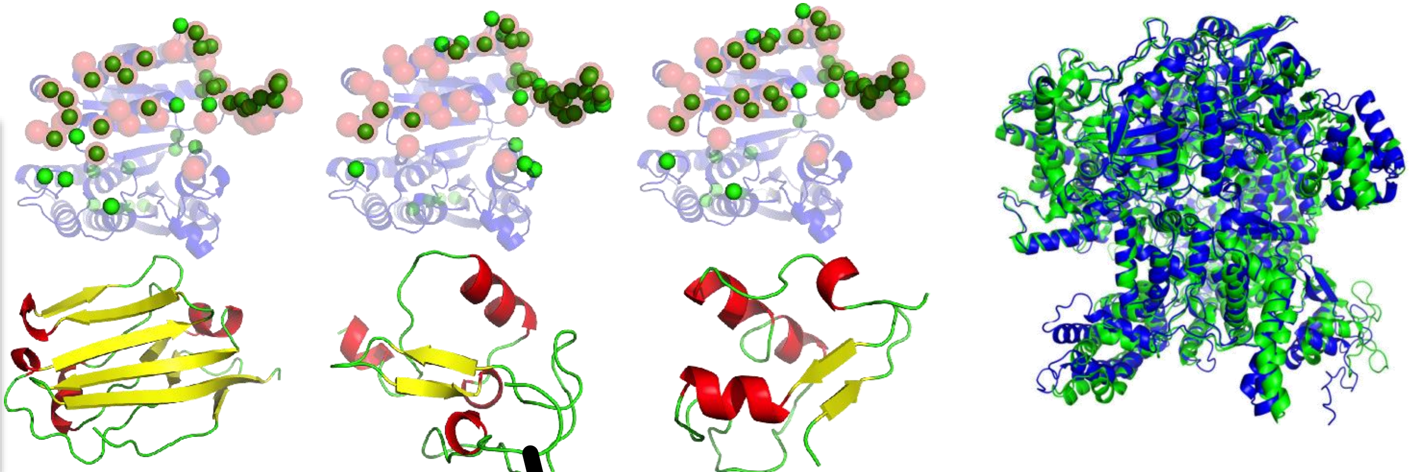


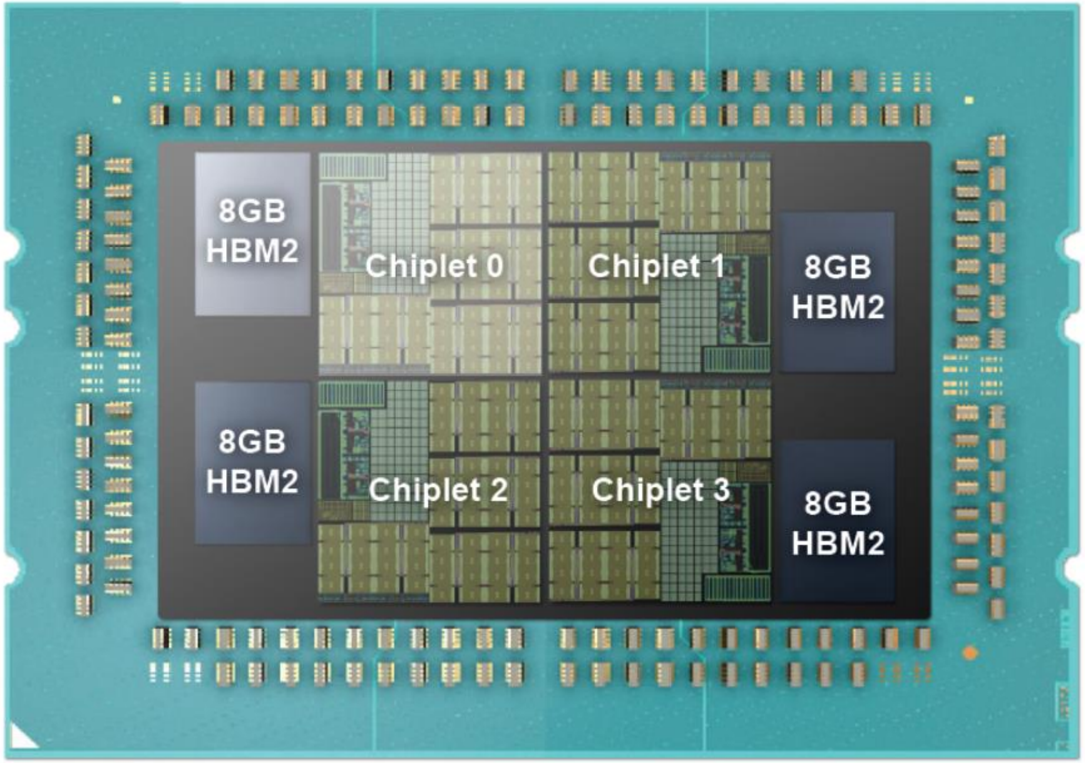


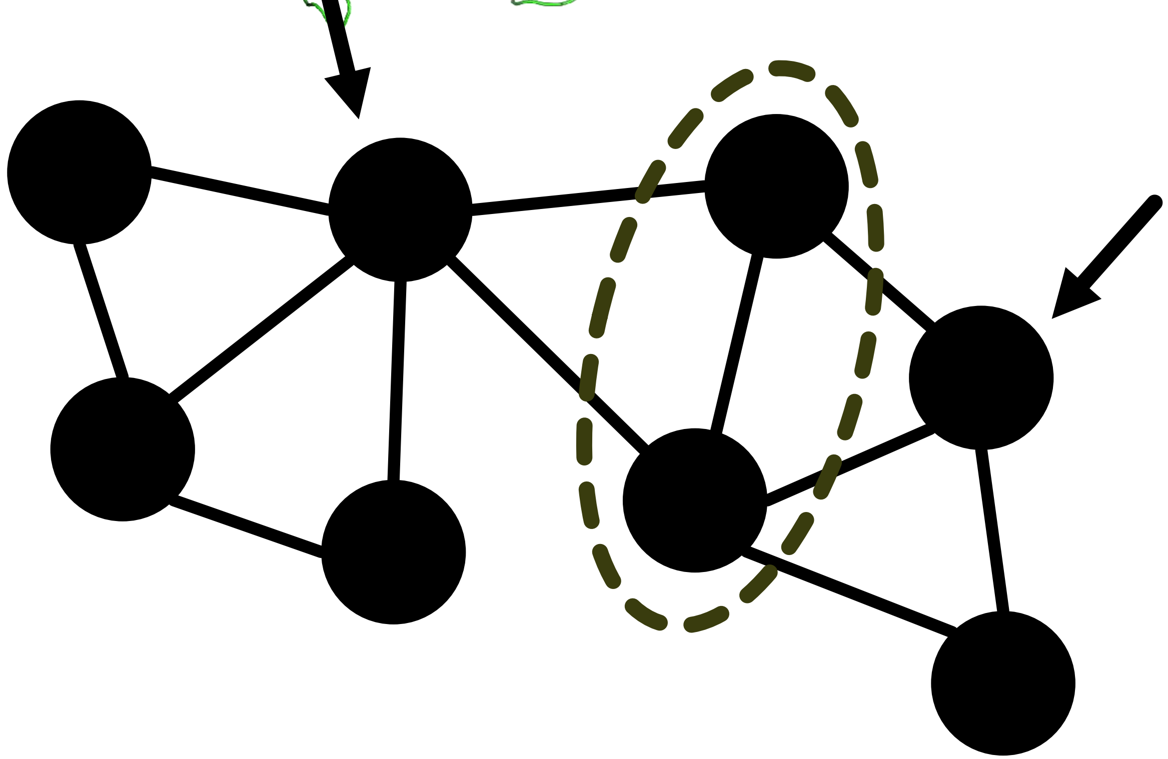
Use Case 4: Vertex Similarity

Enhancing graph embedding construction

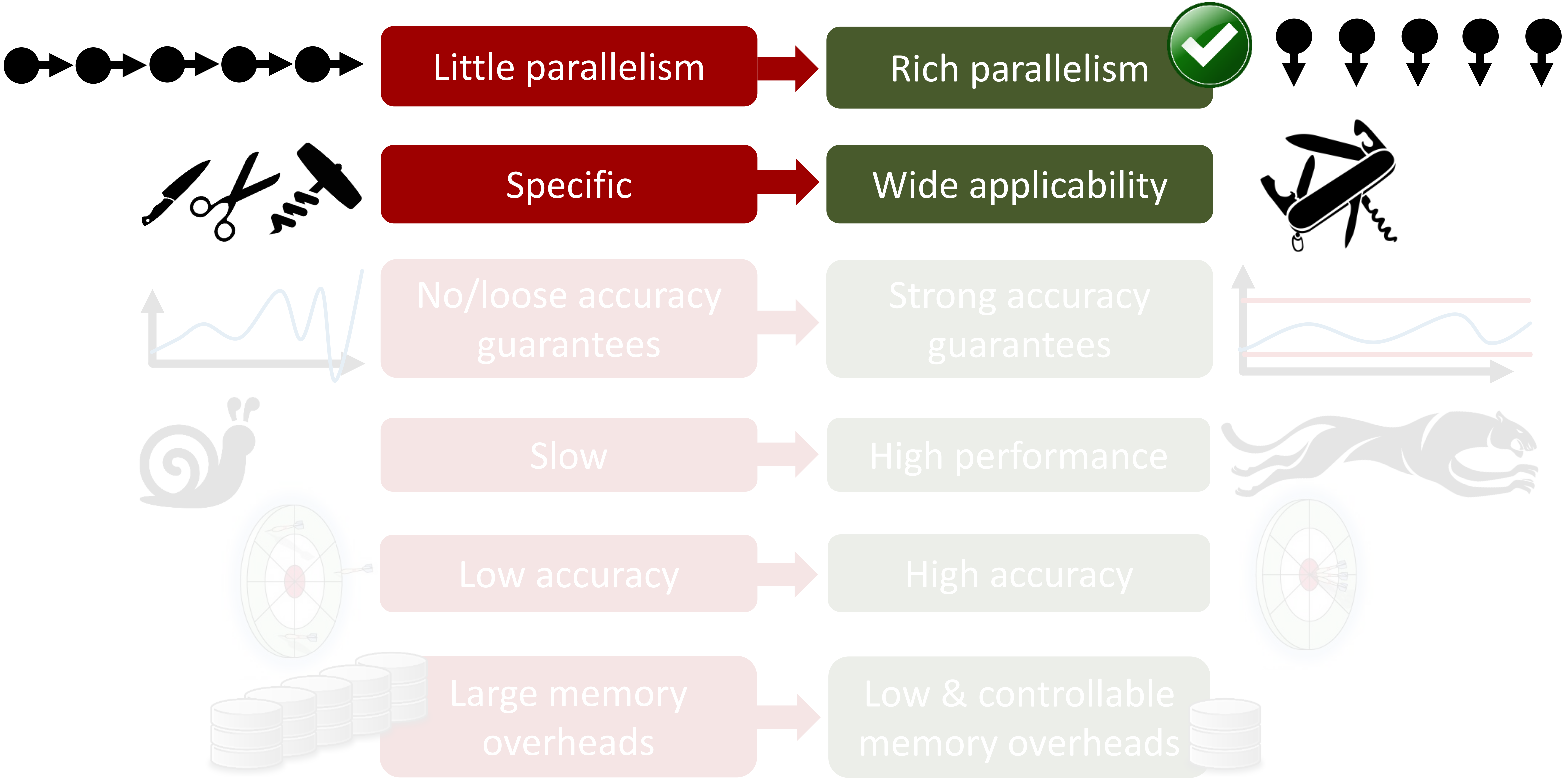








Approximate Graph Processing: Our Objectives



Little parallelism → Rich parallelism

Specific → Wide applicability

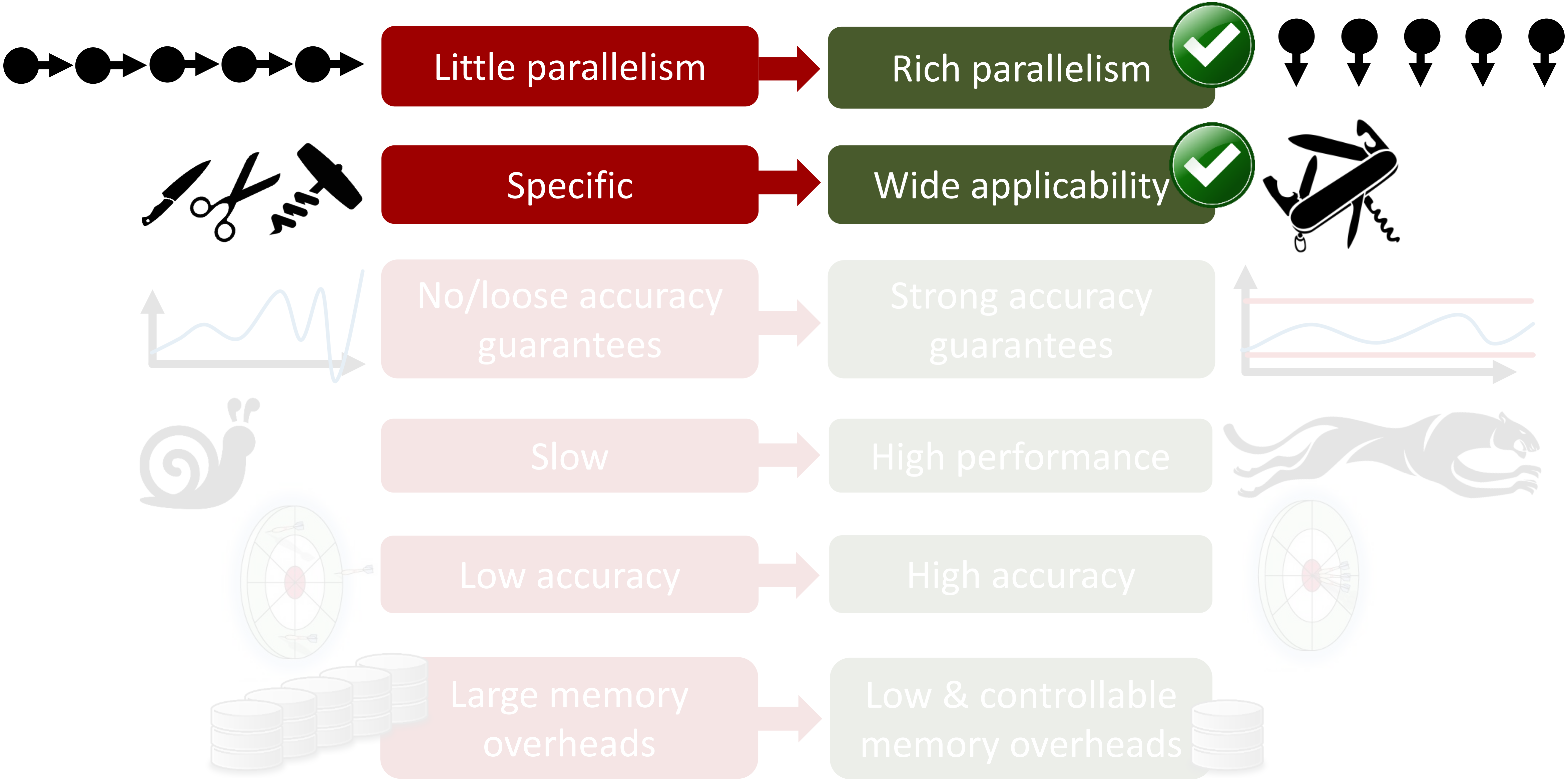
No/loose accuracy guarantees → Strong accuracy guarantees

Slow → High performance

Low accuracy → High accuracy

Large memory overheads → Low & controllable memory overheads

Approximate Graph Processing: Our Objectives



Little parallelism → Rich parallelism ✓

Specific → Wide applicability ✓

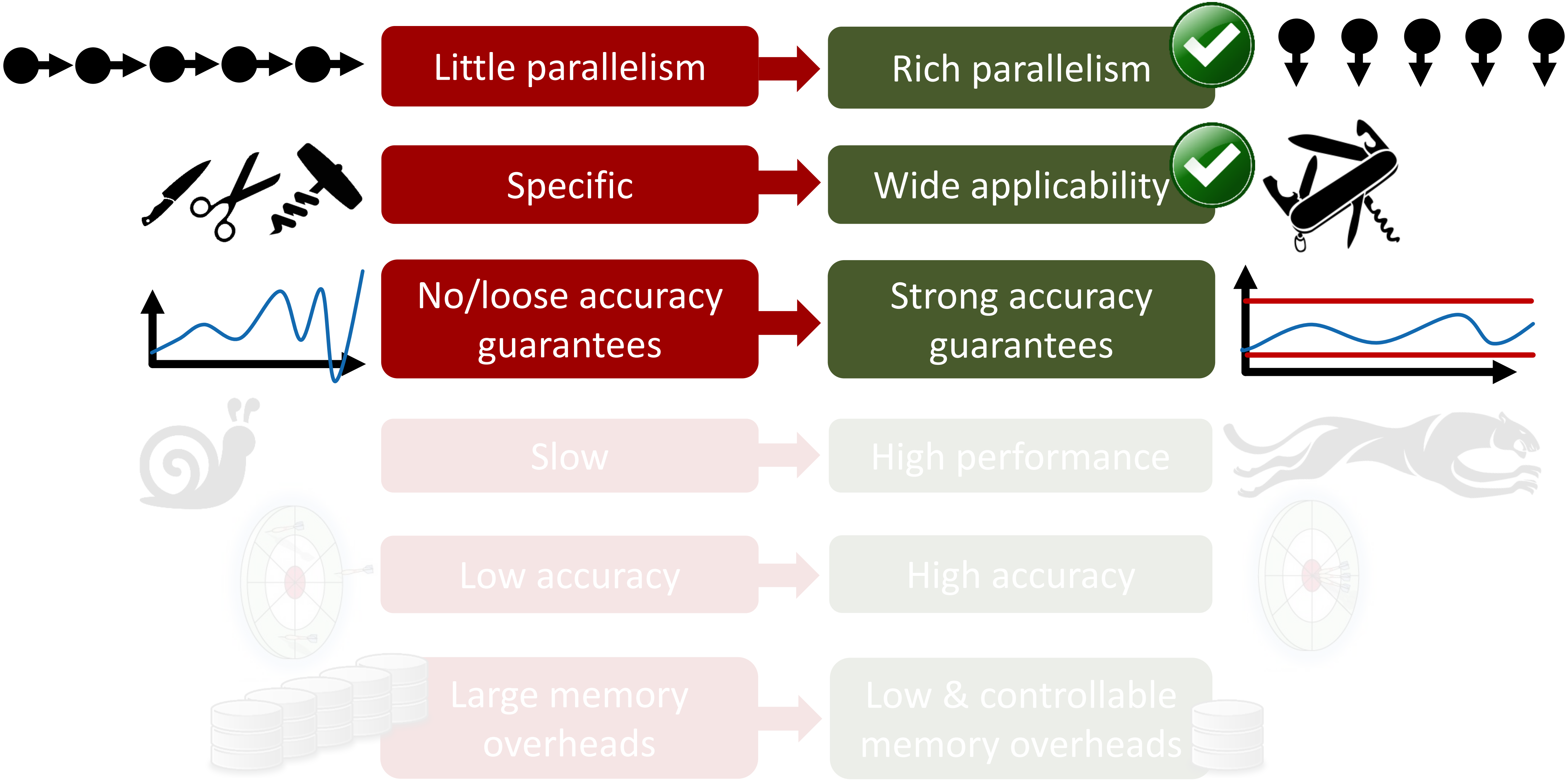
No/loose accuracy guarantees → Strong accuracy guarantees

Slow → High performance

Low accuracy → High accuracy

Large memory overheads → Low & controllable memory overheads

Approximate Graph Processing: Our Objectives



Little parallelism → Rich parallelism ✓

Specific → Wide applicability ✓

No/loose accuracy guarantees → Strong accuracy guarantees

Slow → High performance

Low accuracy → High accuracy

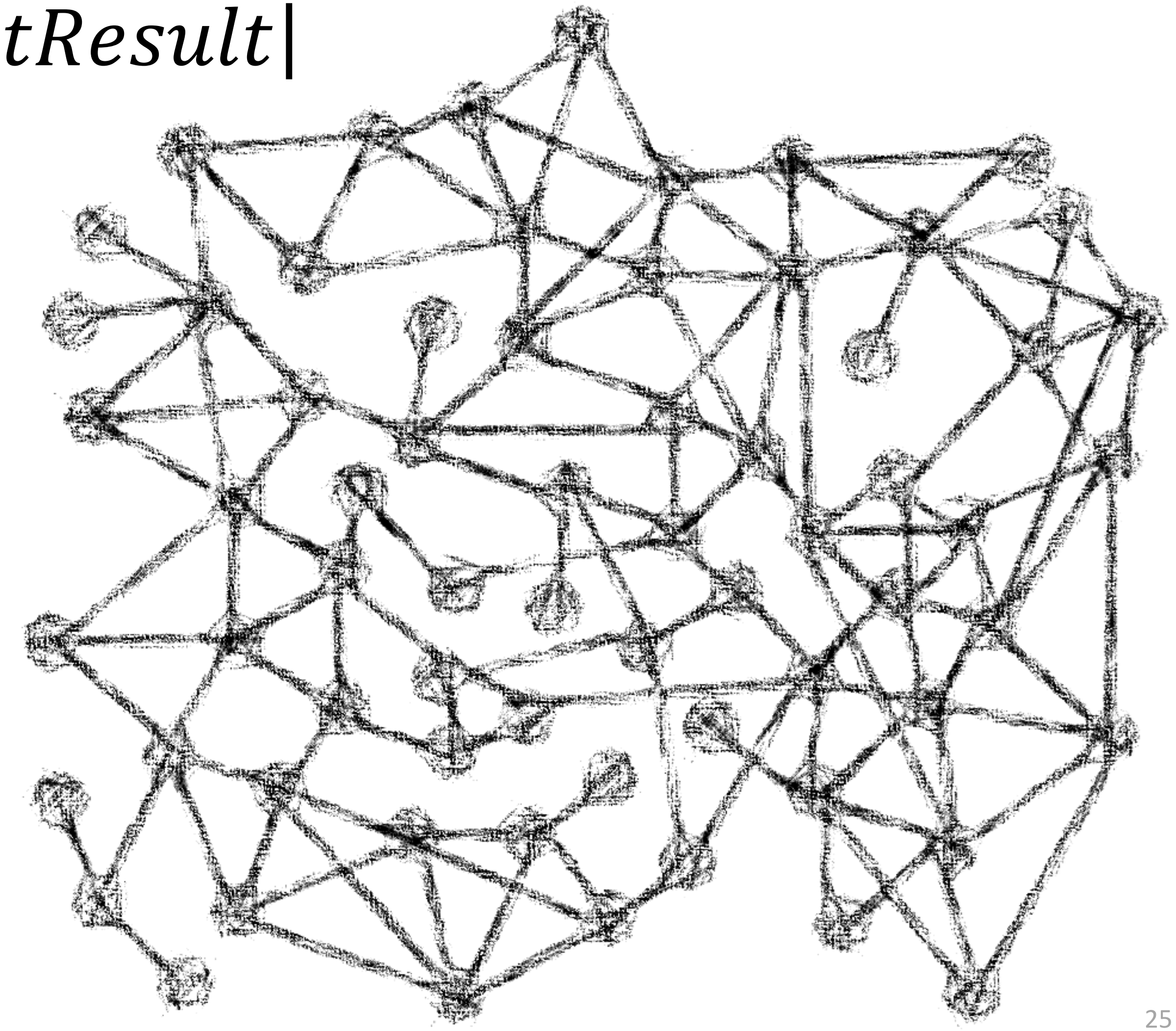
Large memory overheads → Low & controllable memory overheads

ProbGraph: Summary of Theoretical Results

ProbGraph: Summary of Theoretical Results

We want guarantees for

$$|ProbGraphEstimate - exactResult|$$

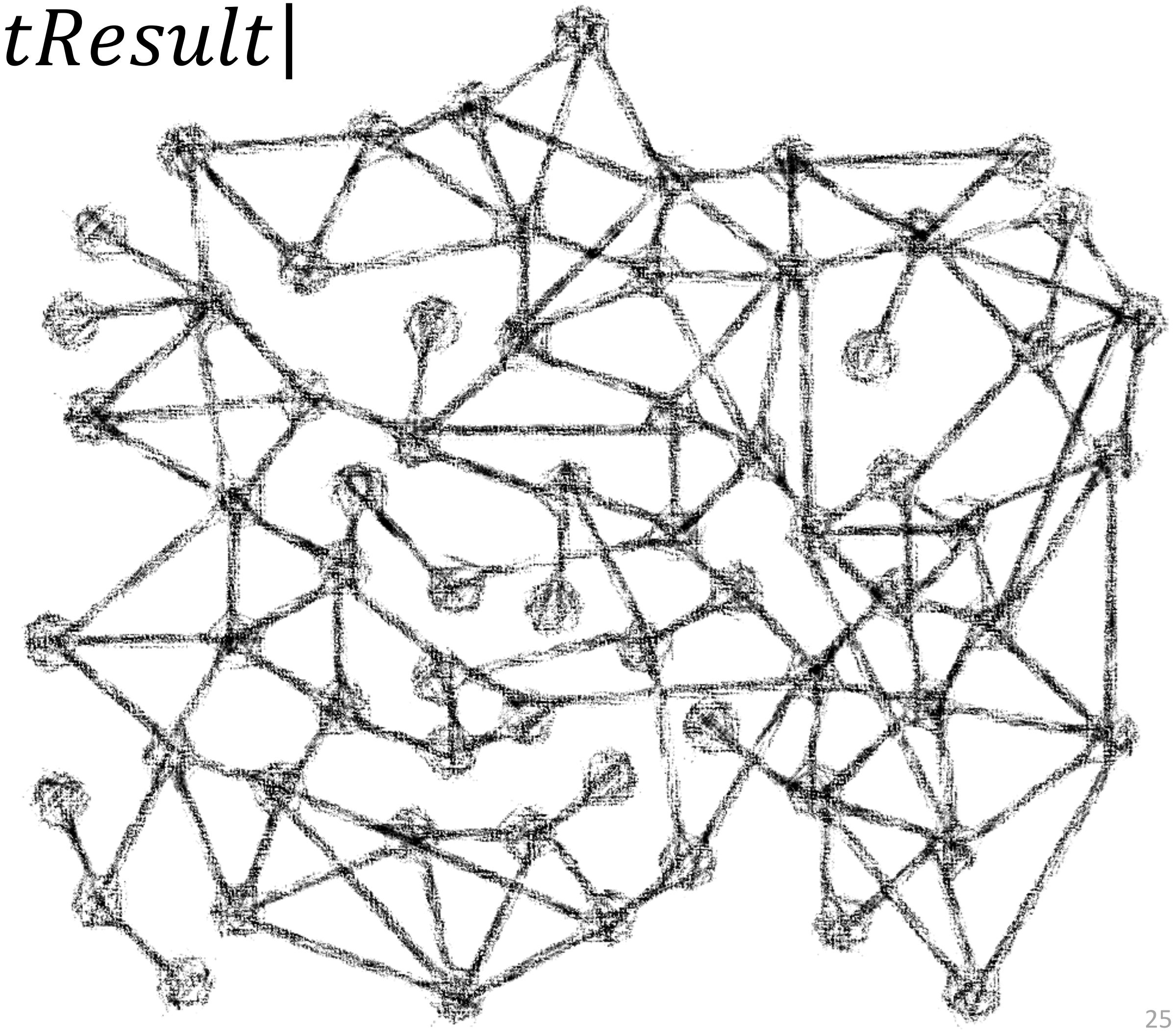


ProbGraph: Summary of Theoretical Results

We want guarantees for

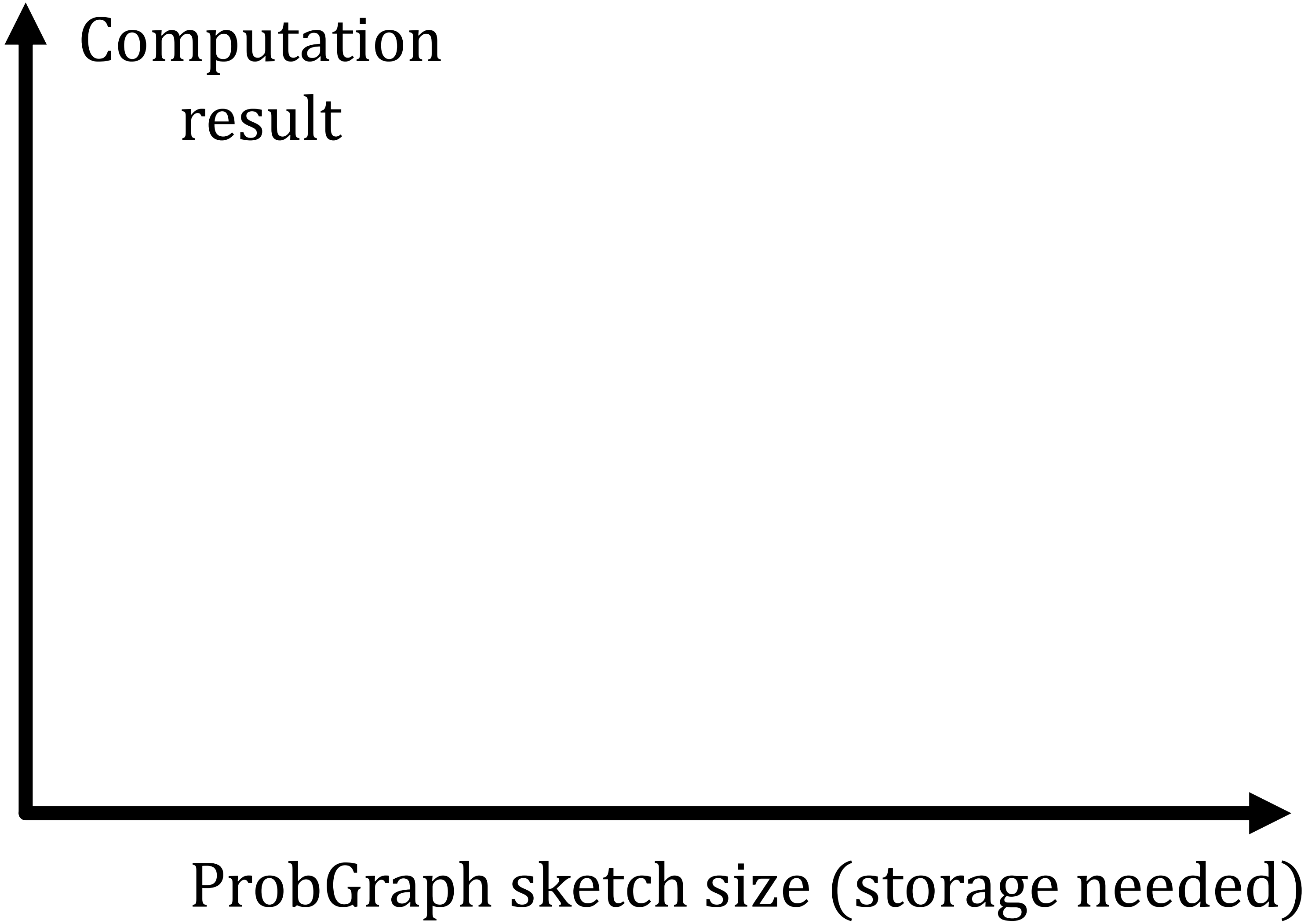
$$|ProbGraphEstimate - exactResult|$$

We incorporate statistical theory of estimators

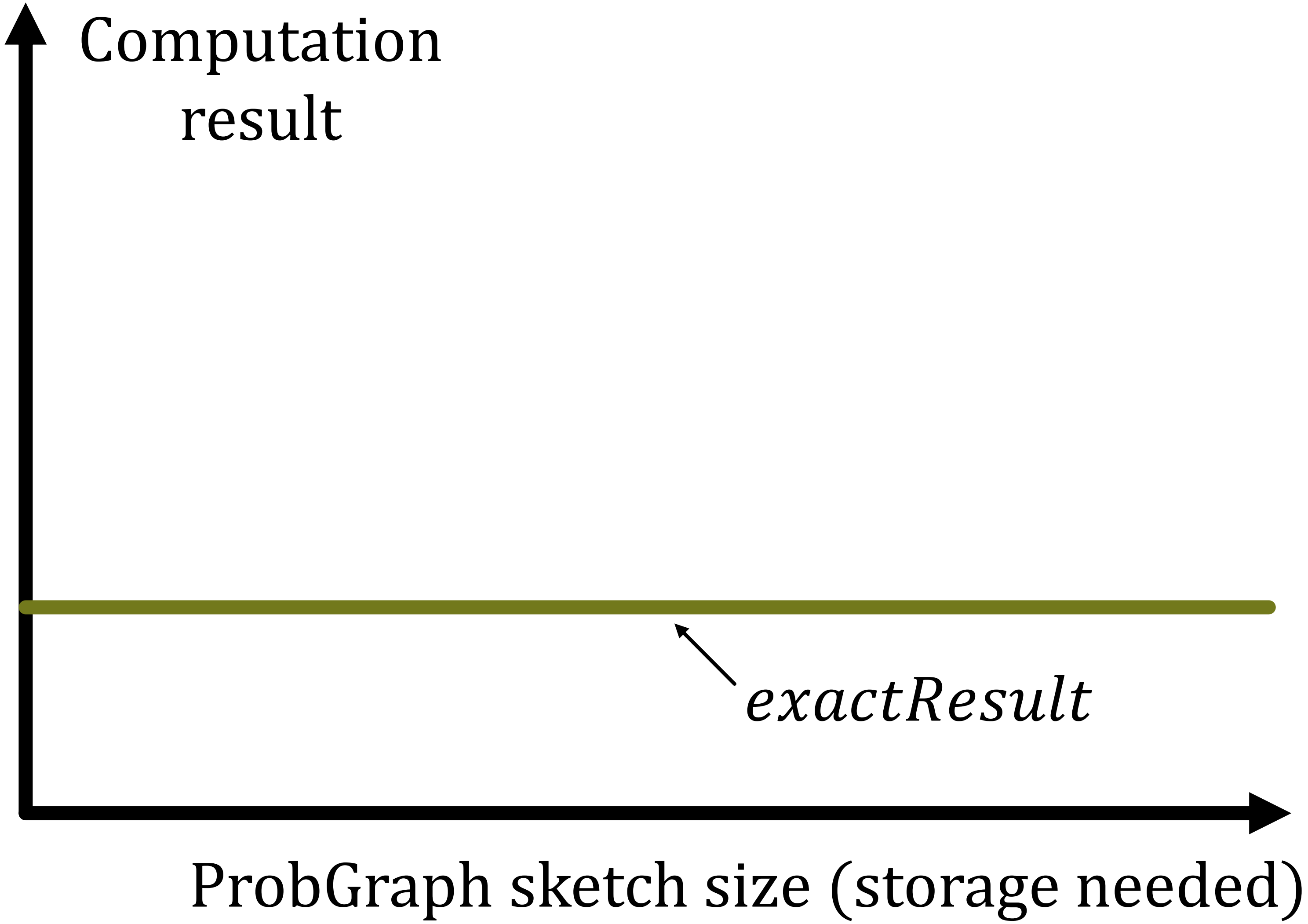


ProbGraph is asymptotically unbiased

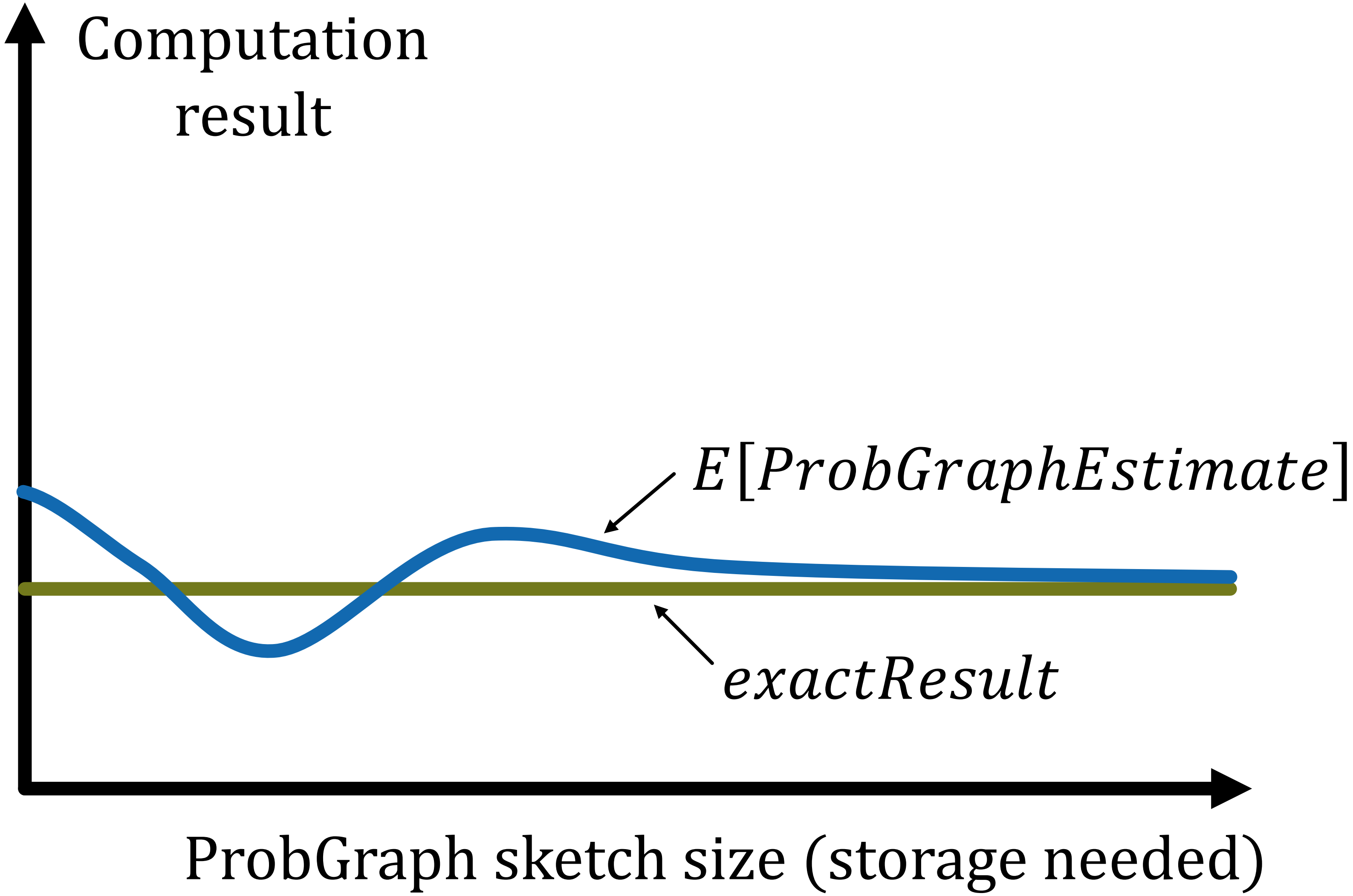
ProbGraph is asymptotically unbiased



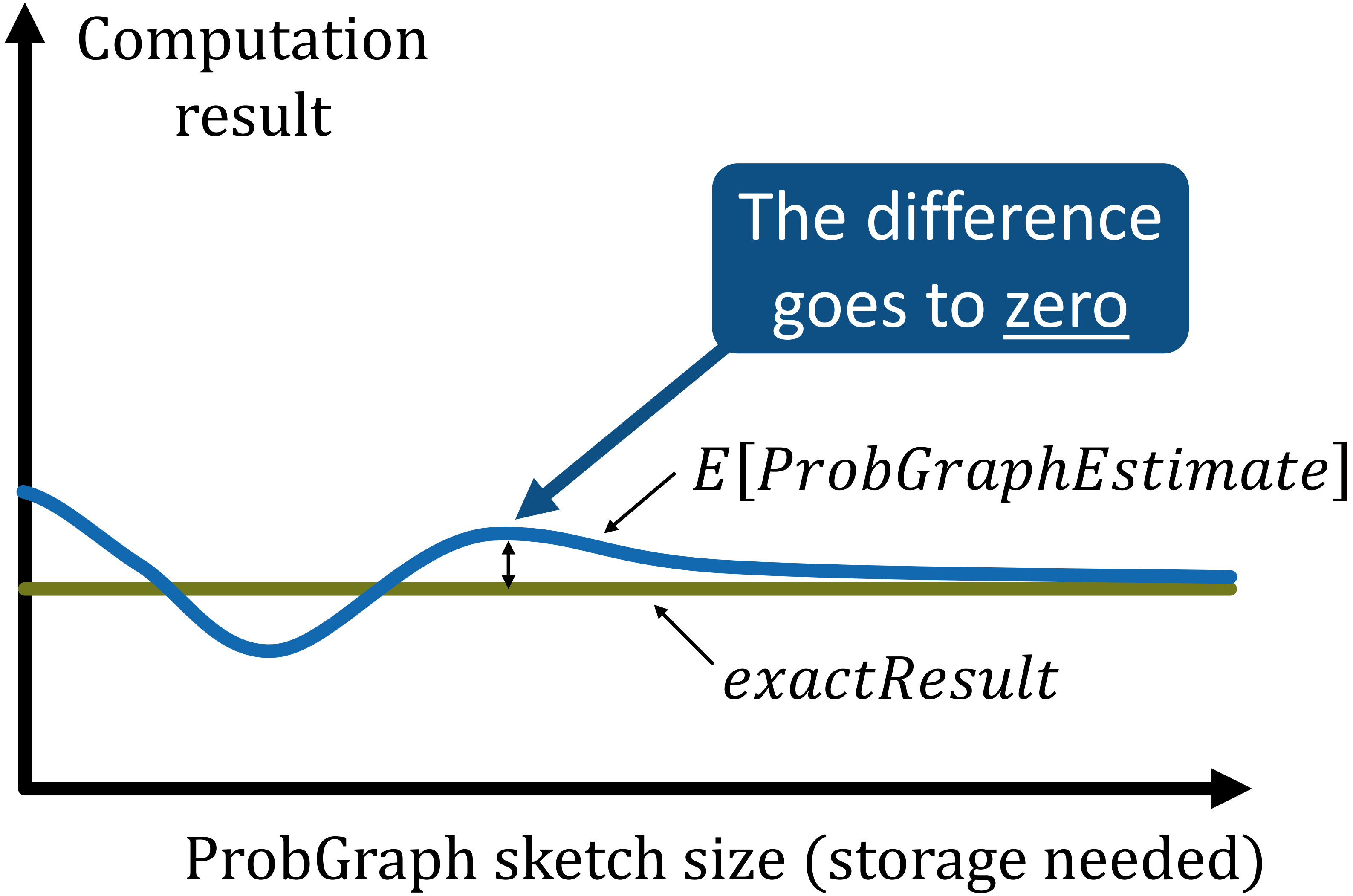
ProbGraph is asymptotically unbiased



ProbGraph is asymptotically unbiased

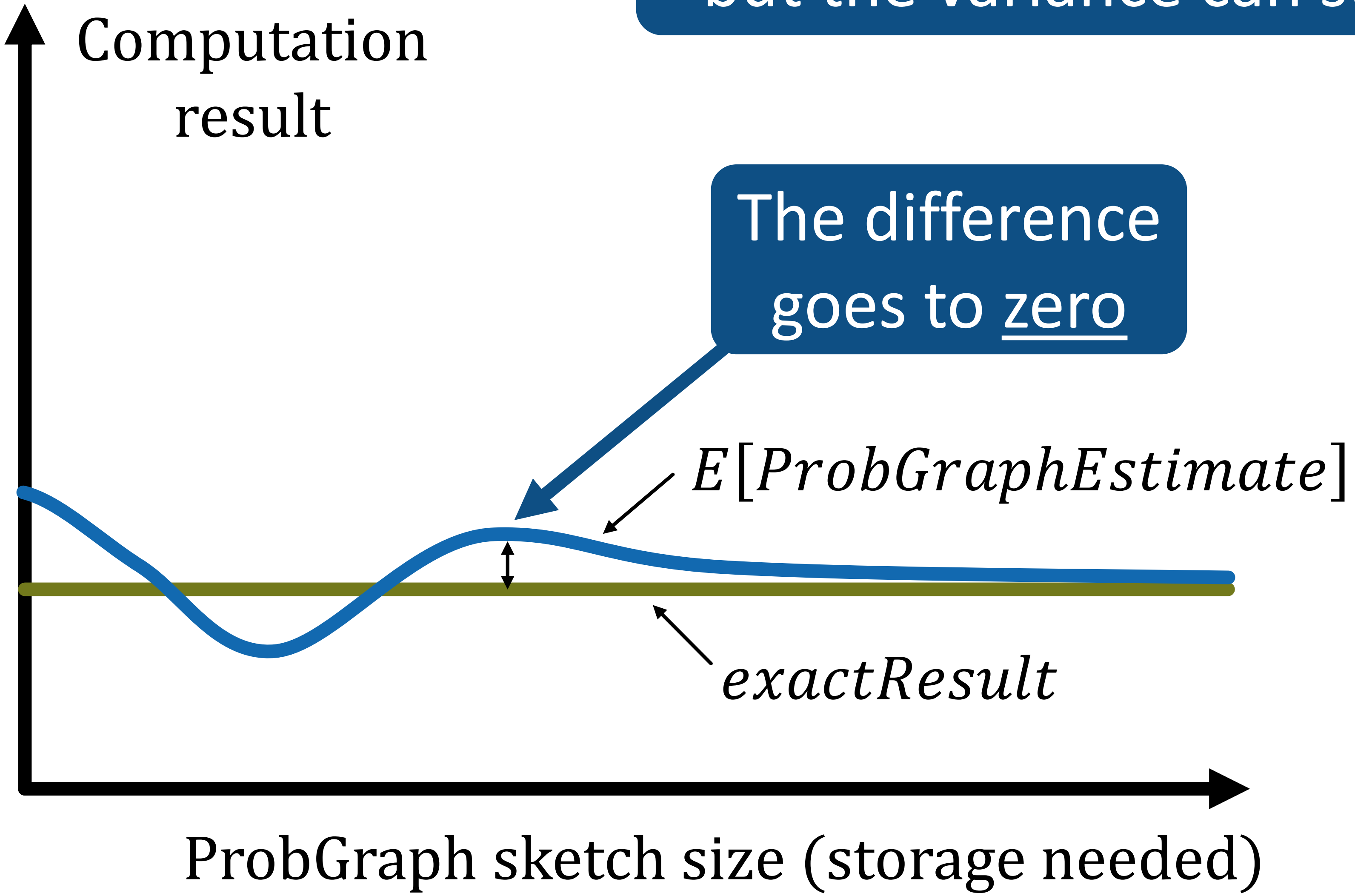


ProbGraph is asymptotically unbiased

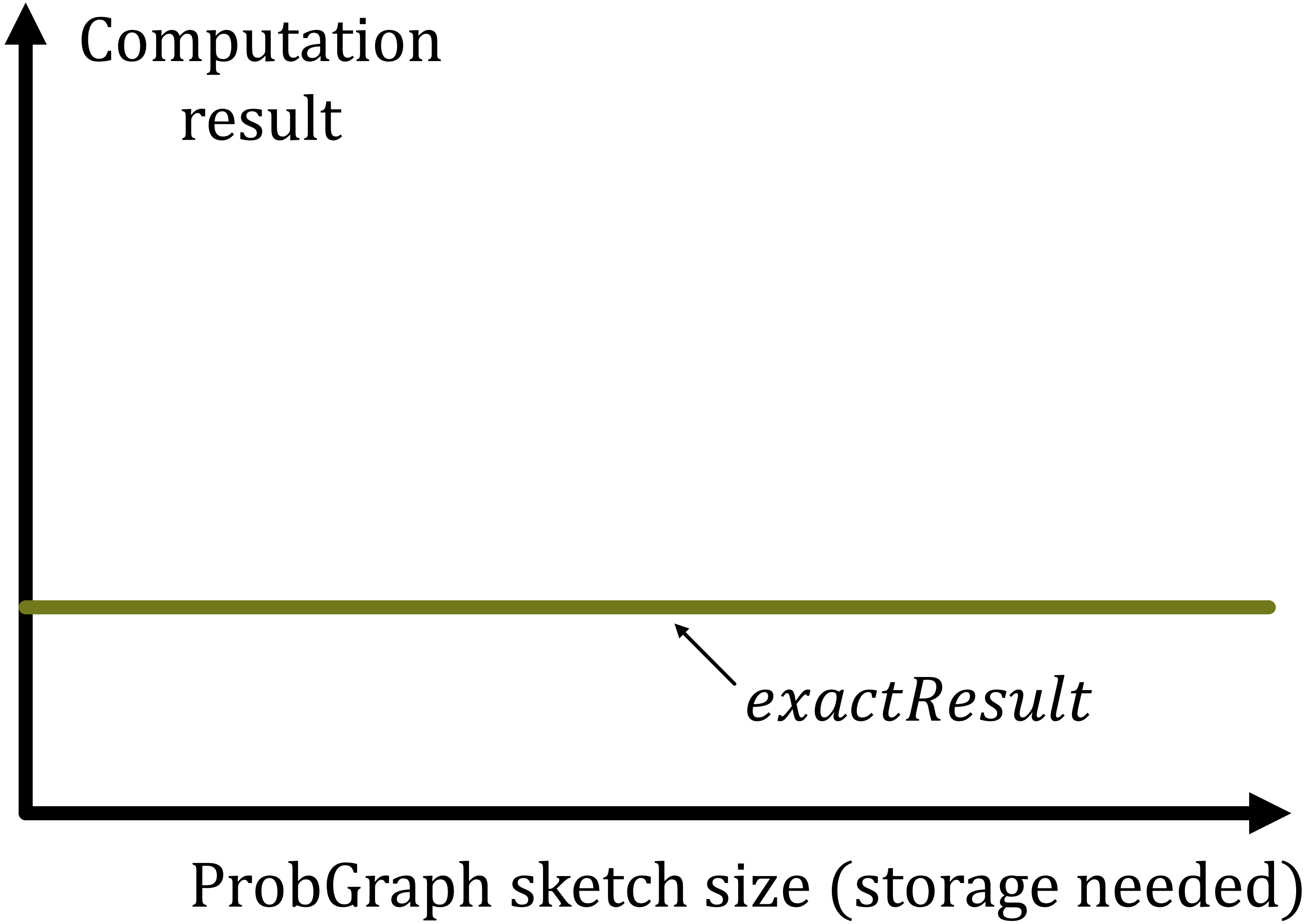


ProbGraph is asymptotically unbiased

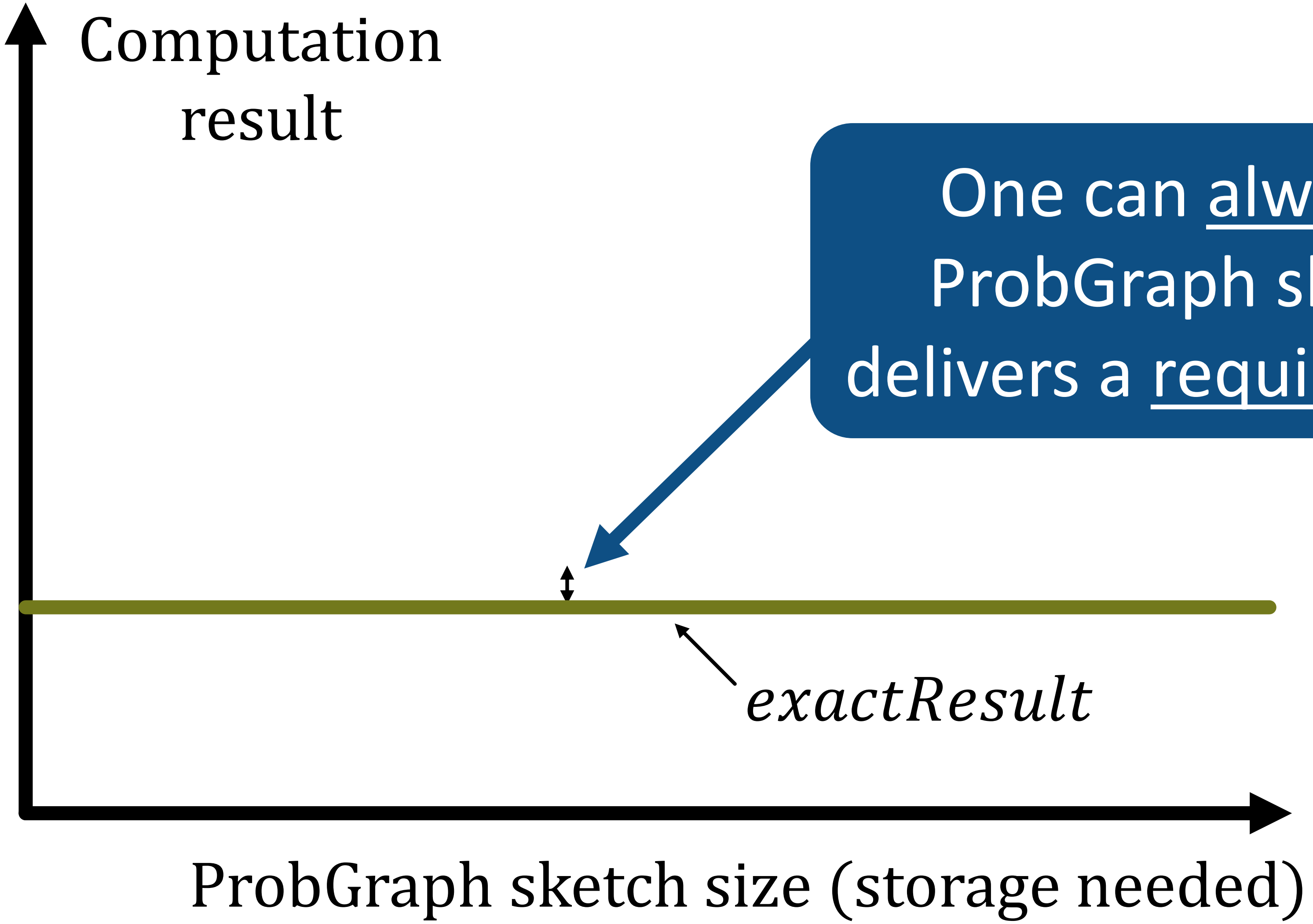
Zero average error at some point… but the variance can still go wild



ProbGraph is consistent

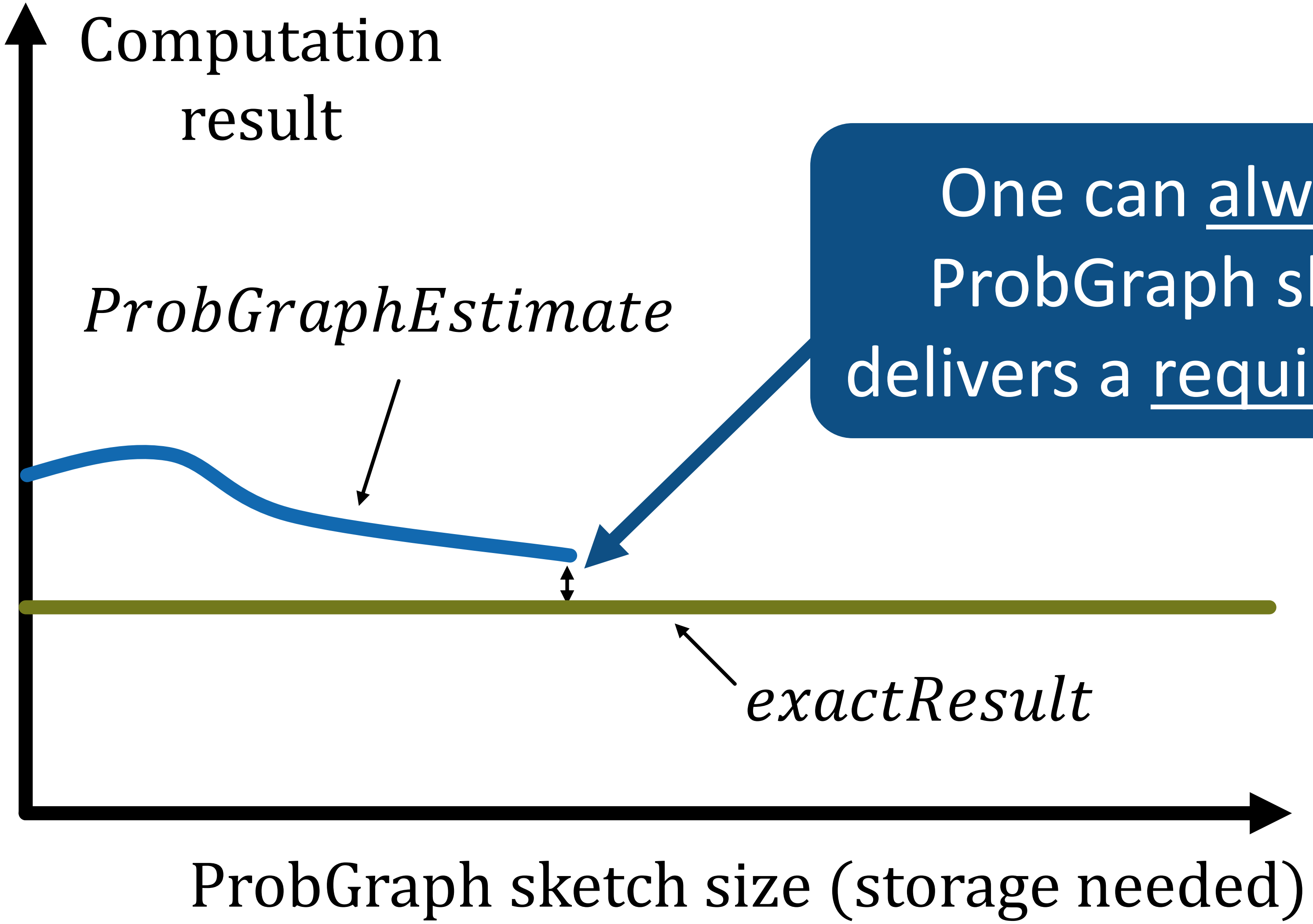


ProbGraph is consistent



One can always find a ProbGraph sketch that delivers a required accuracy

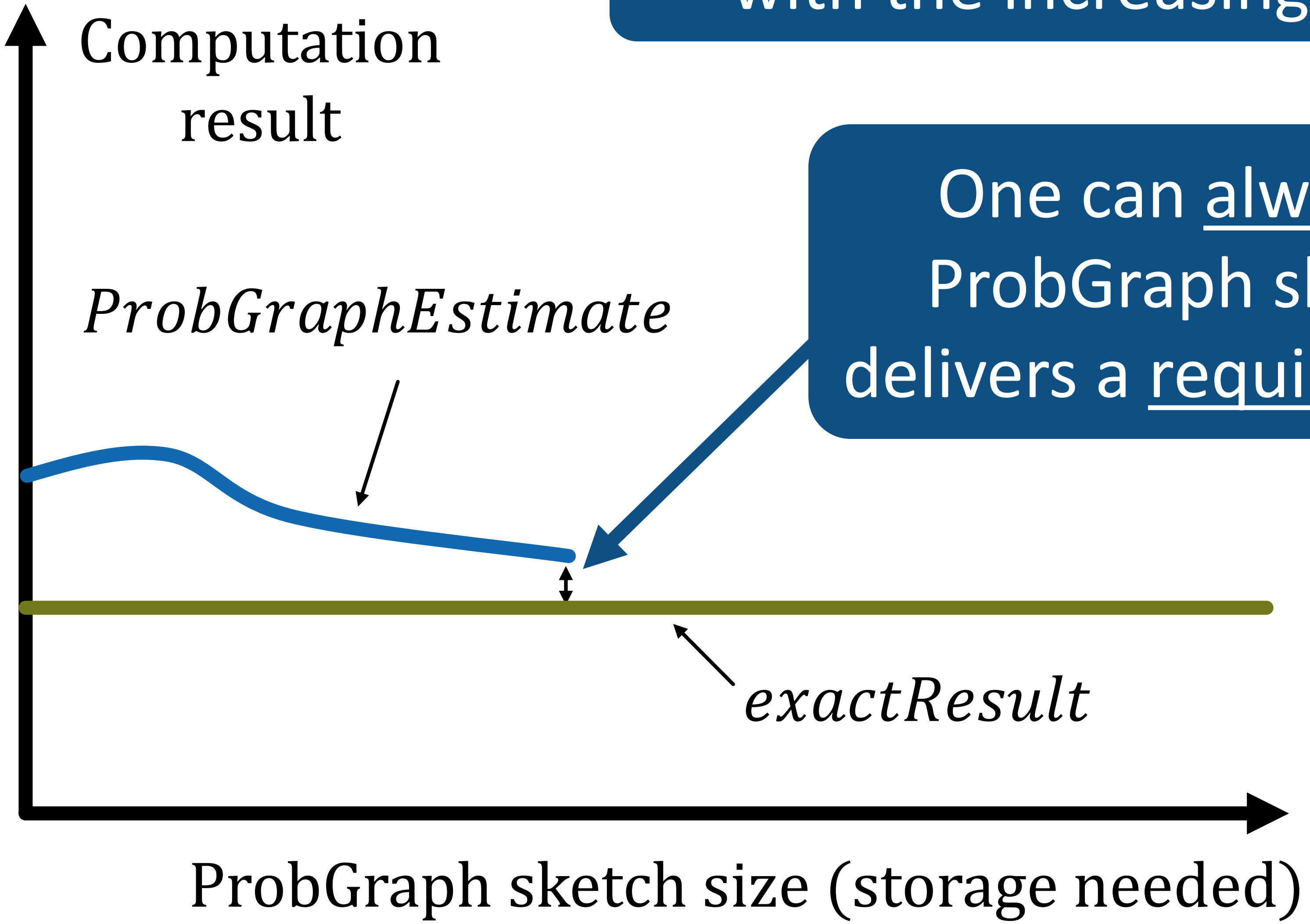
ProbGraph is consistent



One can always find a ProbGraph sketch that delivers a required accuracy

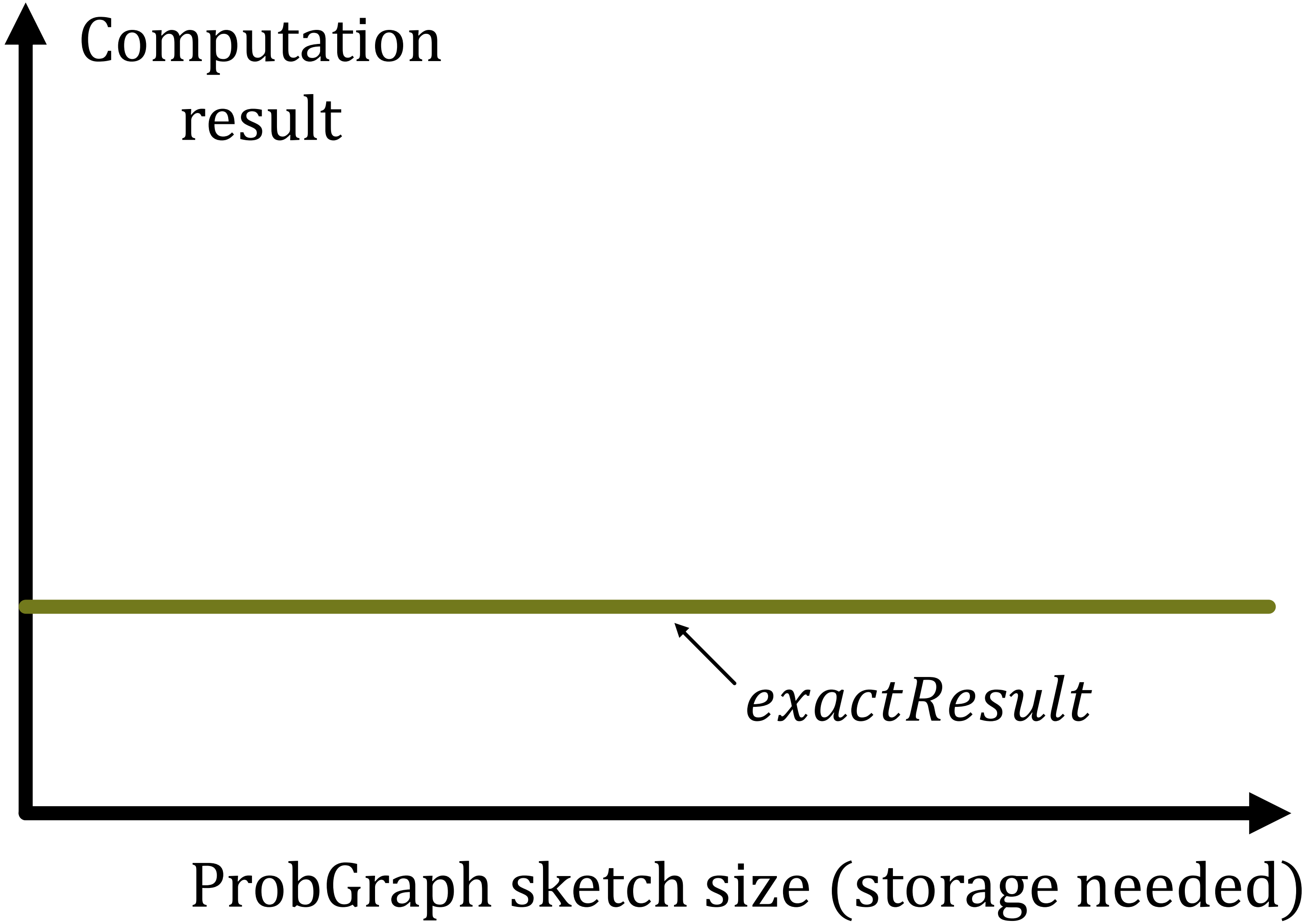
ProbGraph is consistent

The variance also converges to zero with the increasing sketch size

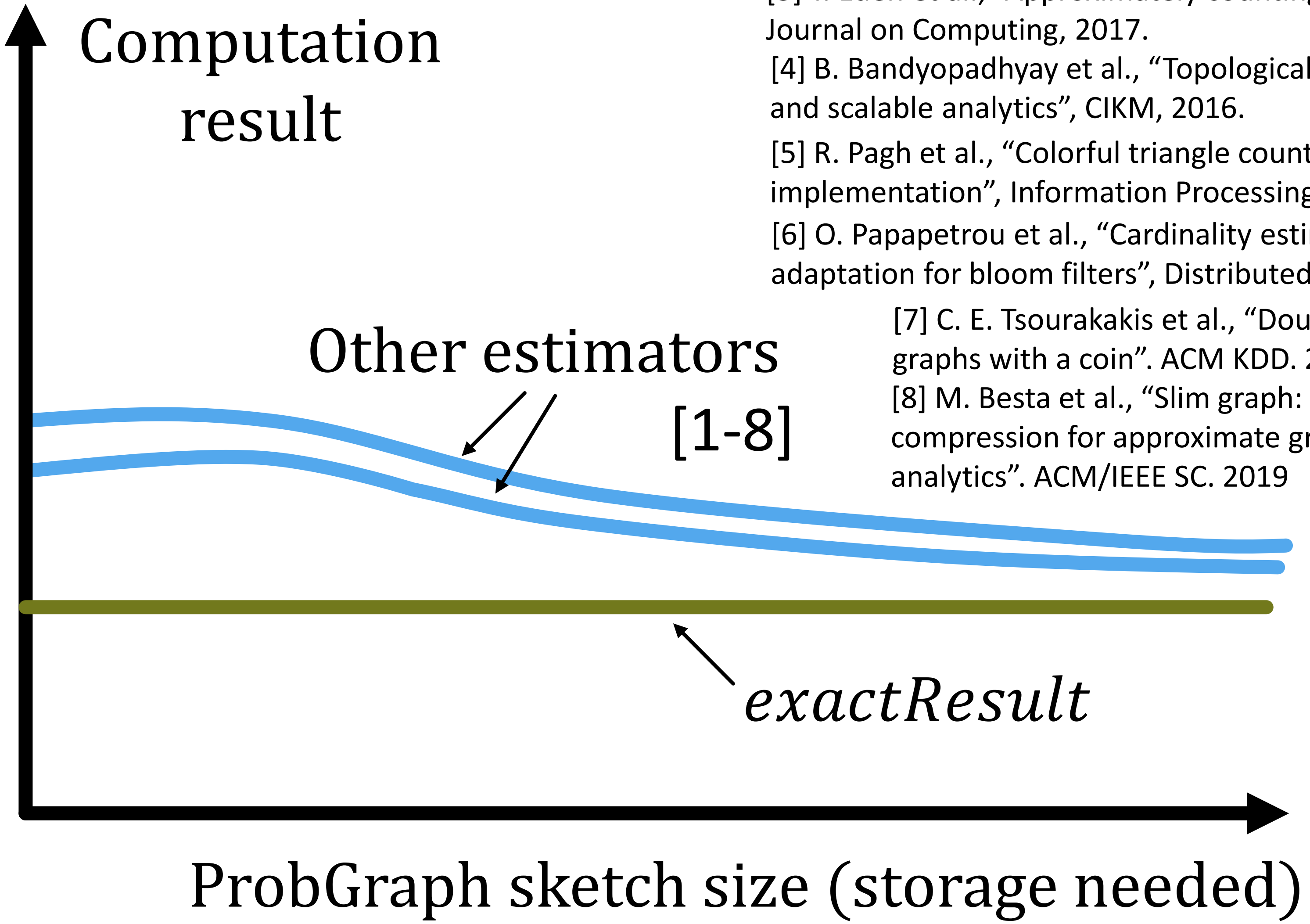


One can always find a ProbGraph sketch that delivers a required accuracy

ProbGraph is asymptotically efficient



ProbGraph is asymptotically efficient



[1] J. Tetek, “Approximate triangle counting via sampling and fast matrix ˇ multiplication”, arXiv 2021.

[2] S. Assadi et al., “A simple sublinear-time algorithm for counting arbitrary subgraphs via edge sampling”, arXiv 2018.

[3] T. Eden et al., “Approximately counting triangles in sublinear time”, SIAM Journal on Computing, 2017.

[4] B. Bandyopadhyay et al., “Topological graph sketching for incremental and scalable analytics”, CIKM, 2016.

[5] R. Pagh et al., “Colorful triangle counting and a MapReduce implementation”, Information Processing Letters, 2012.

[6] O. Papapetrou et al., “Cardinality estimation and dynamic length adaptation for bloom filters”, Distributed and Parallel Databases, 2010.

[7] C. E. Tsourakakis et al., “Doulion: counting triangles in massive graphs with a coin”. ACM KDD. 2009.

[8] M. Besta et al., “Slim graph: Practical lossy graph compression for approximate graph processing, storage, and analytics”. ACM/IEEE SC. 2019

ProbGraph is asymptotically efficient

[1] J. Tetek, “Approximate triangle counting via sampling and fast matrix ˇ multiplication”, arXiv 2021.

[2] S. Assadi et al., “A simple sublinear-time algorithm for counting arbitrary subgraphs via edge sampling”, arXiv 2018.

[3] T. Eden et al., “Approximately counting triangles in sublinear time”, SIAM Journal on Computing, 2017.

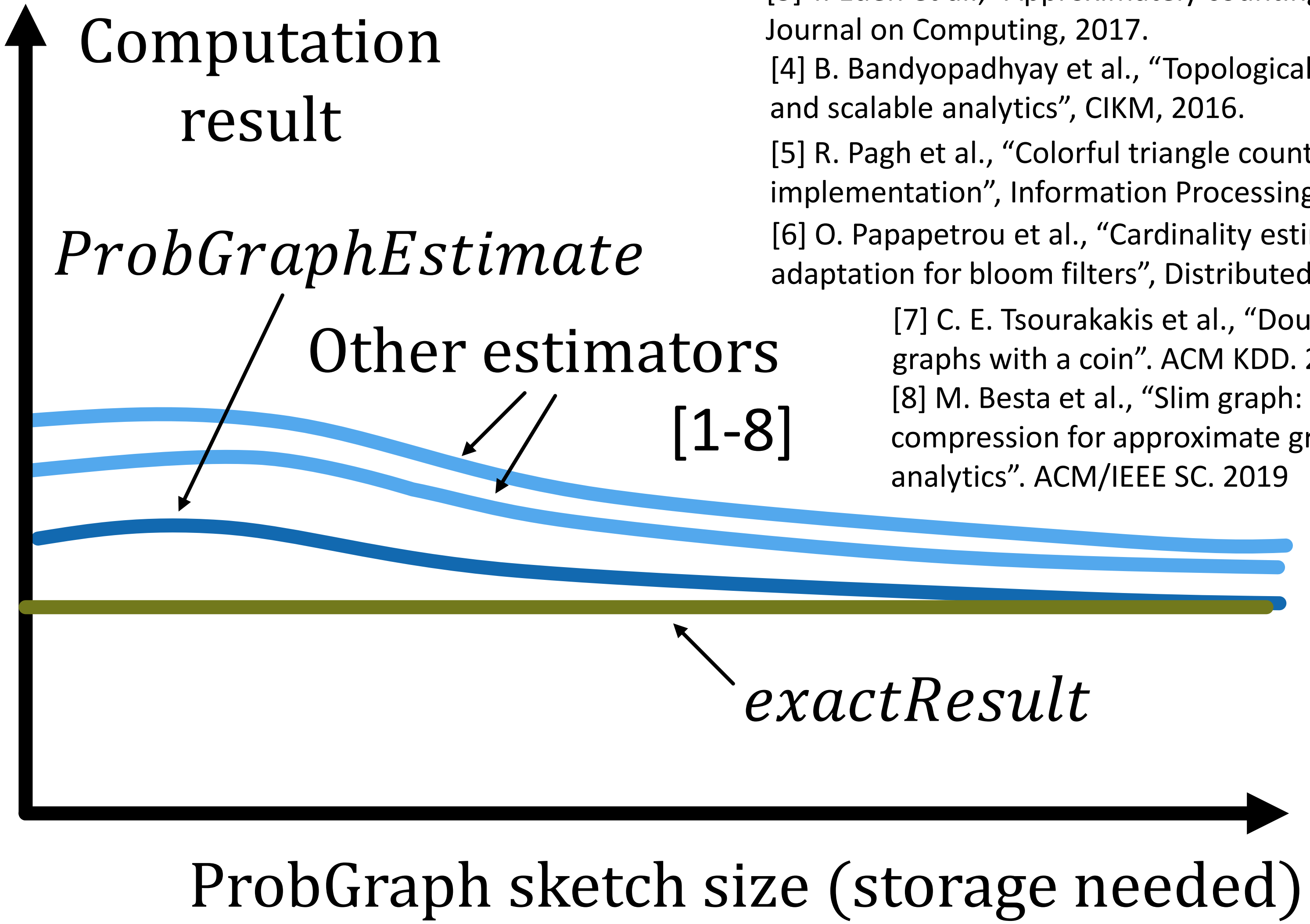
[4] B. Bandyopadhyay et al., “Topological graph sketching for incremental and scalable analytics”, CIKM, 2016.

[5] R. Pagh et al., “Colorful triangle counting and a MapReduce implementation”, Information Processing Letters, 2012.

[6] O. Papapetrou et al., “Cardinality estimation and dynamic length adaptation for bloom filters”, Distributed and Parallel Databases, 2010.

[7] C. E. Tsourakakis et al., “Doulion: counting triangles in massive graphs with a coin”. ACM KDD. 2009.

[8] M. Besta et al., “Slim graph: Practical lossy graph compression for approximate graph processing, storage, and analytics”. ACM/IEEE SC. 2019



ProbGraph is asymptotically efficient

[1] J. Tetek, “Approximate triangle counting via sampling and fast matrix ˇ multiplication”, arXiv 2021.

[2] S. Assadi et al., “A simple sublinear-time algorithm for counting arbitrary subgraphs via edge sampling”, arXiv 2018.

[3] T. Eden et al., “Approximately counting triangles in sublinear time”, SIAM Journal on Computing, 2017.

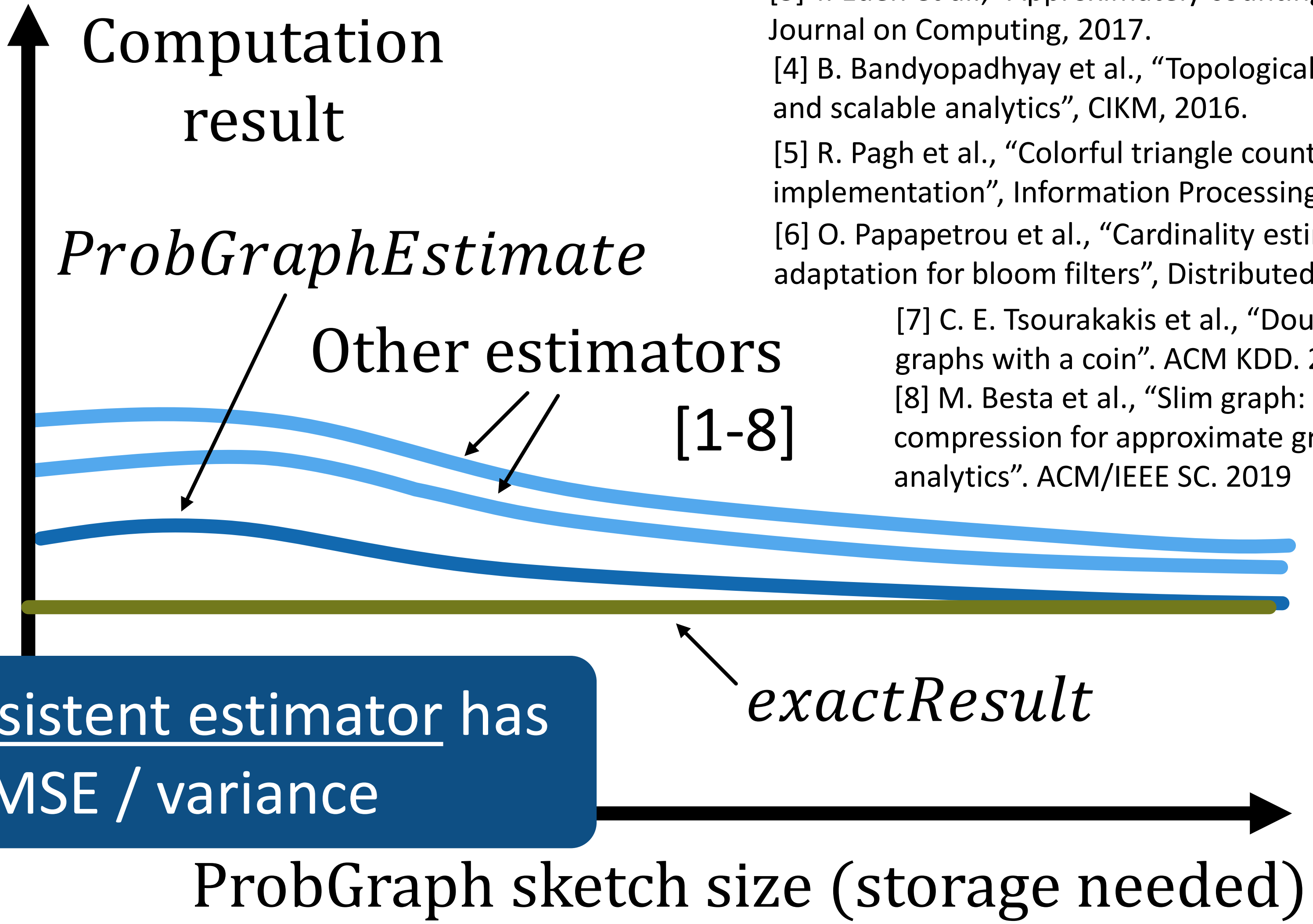
[4] B. Bandyopadhyay et al., “Topological graph sketching for incremental and scalable analytics”, CIKM, 2016.

[5] R. Pagh et al., “Colorful triangle counting and a MapReduce implementation”, Information Processing Letters, 2012.

[6] O. Papapetrou et al., “Cardinality estimation and dynamic length adaptation for bloom filters”, Distributed and Parallel Databases, 2010.

[7] C. E. Tsourakakis et al., “Doulion: counting triangles in massive graphs with a coin”. ACM KDD. 2009.

[8] M. Besta et al., “Slim graph: Practical lossy graph compression for approximate graph processing, storage, and analytics”. ACM/IEEE SC. 2019



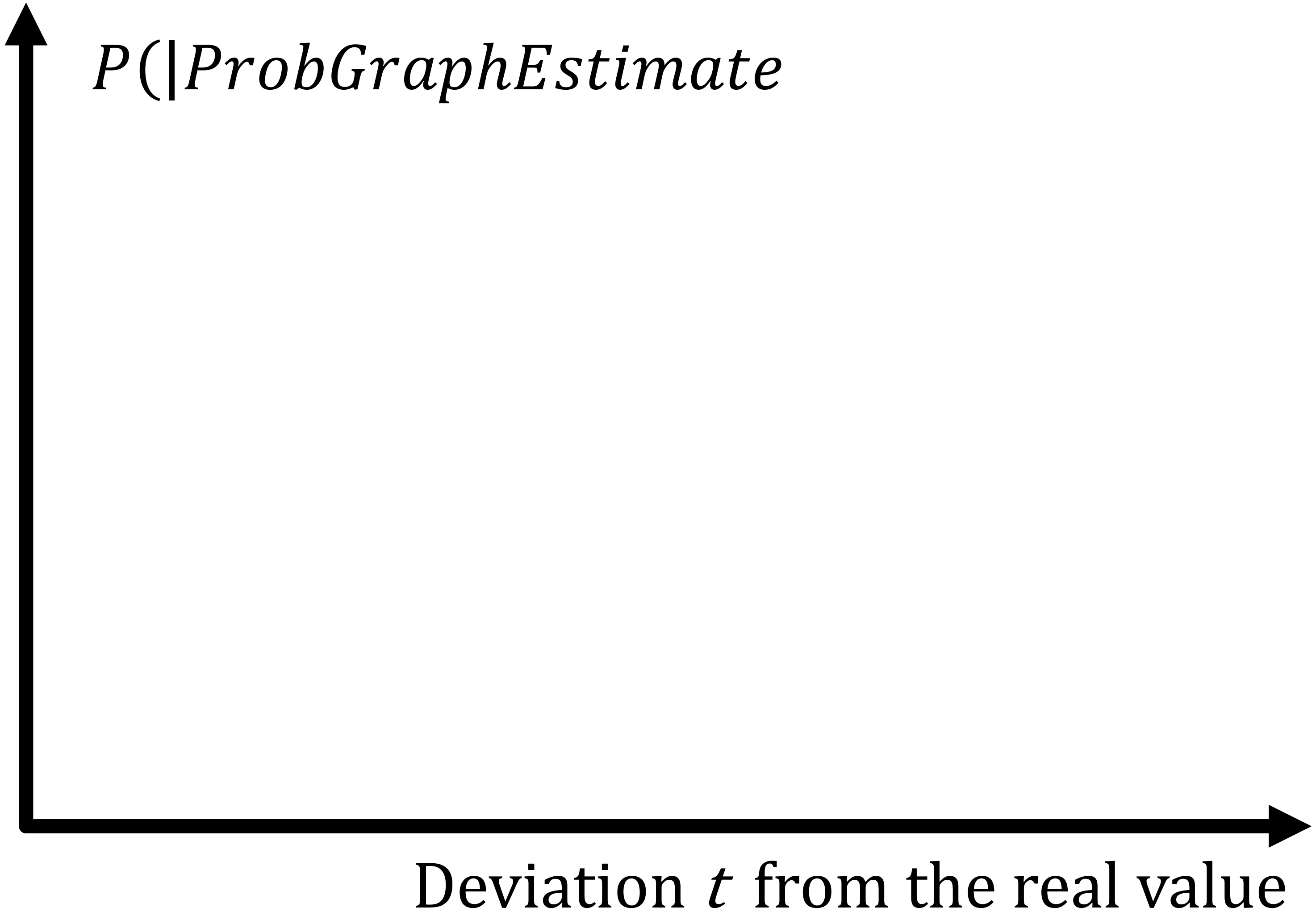
No other consistent estimator has lower MSE / variance

ProbGraph has strong concentration bounds

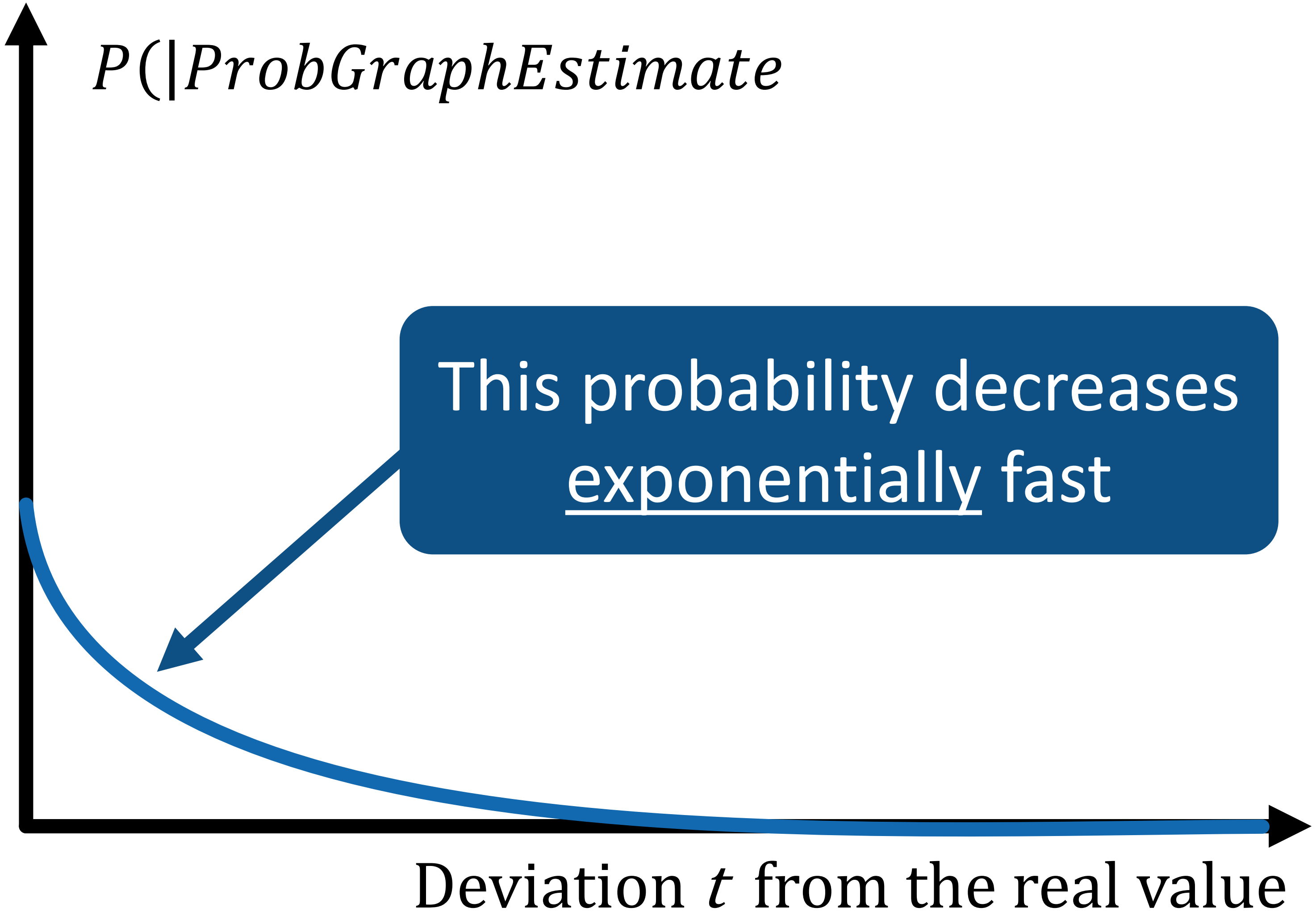
ProbGraph has strong concentration bounds

Deviation t from the real value

ProbGraph has <u>strong concentration bounds</u>



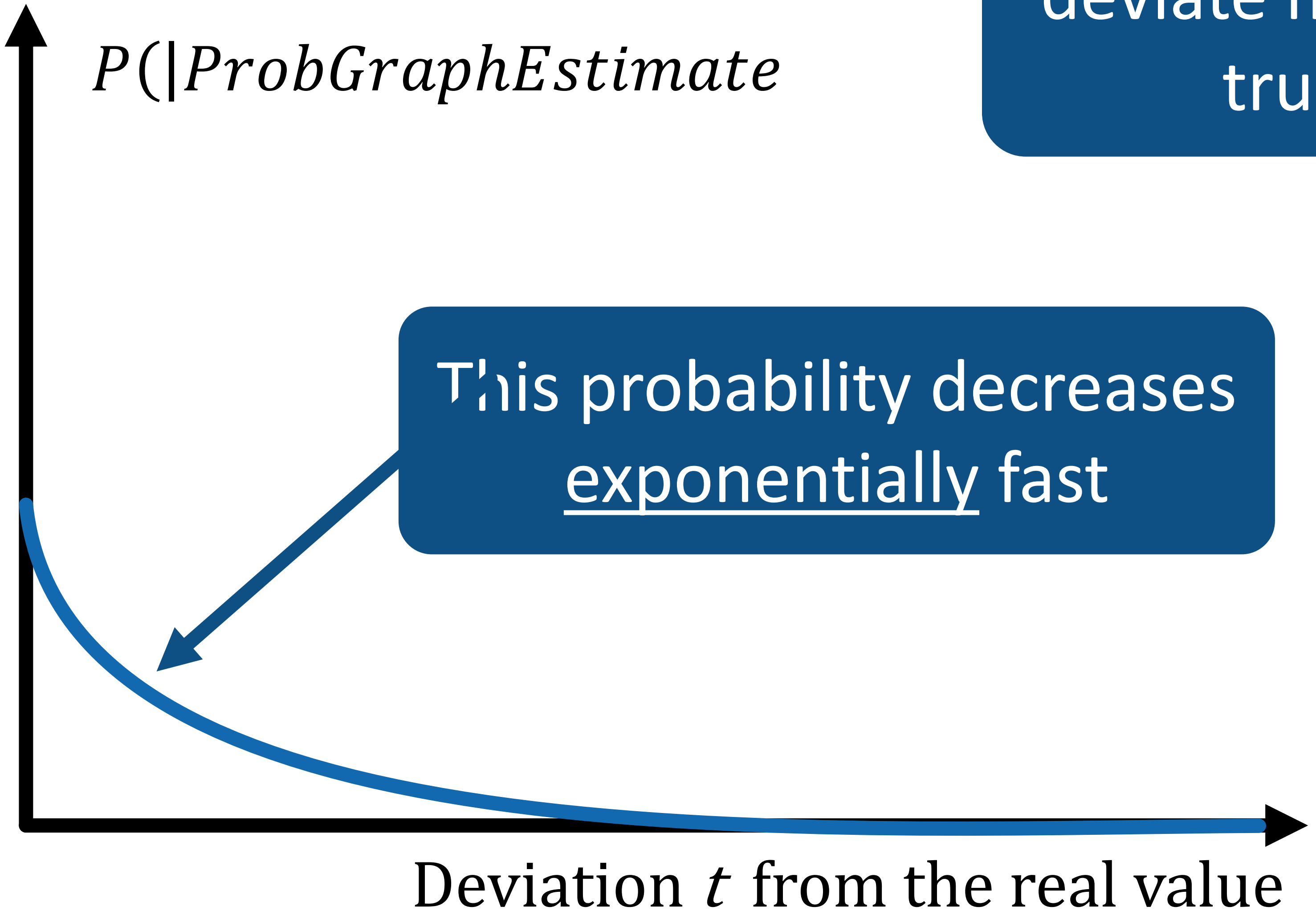
ProbGraph has strong concentration bounds



This probability decreases exponentially fast

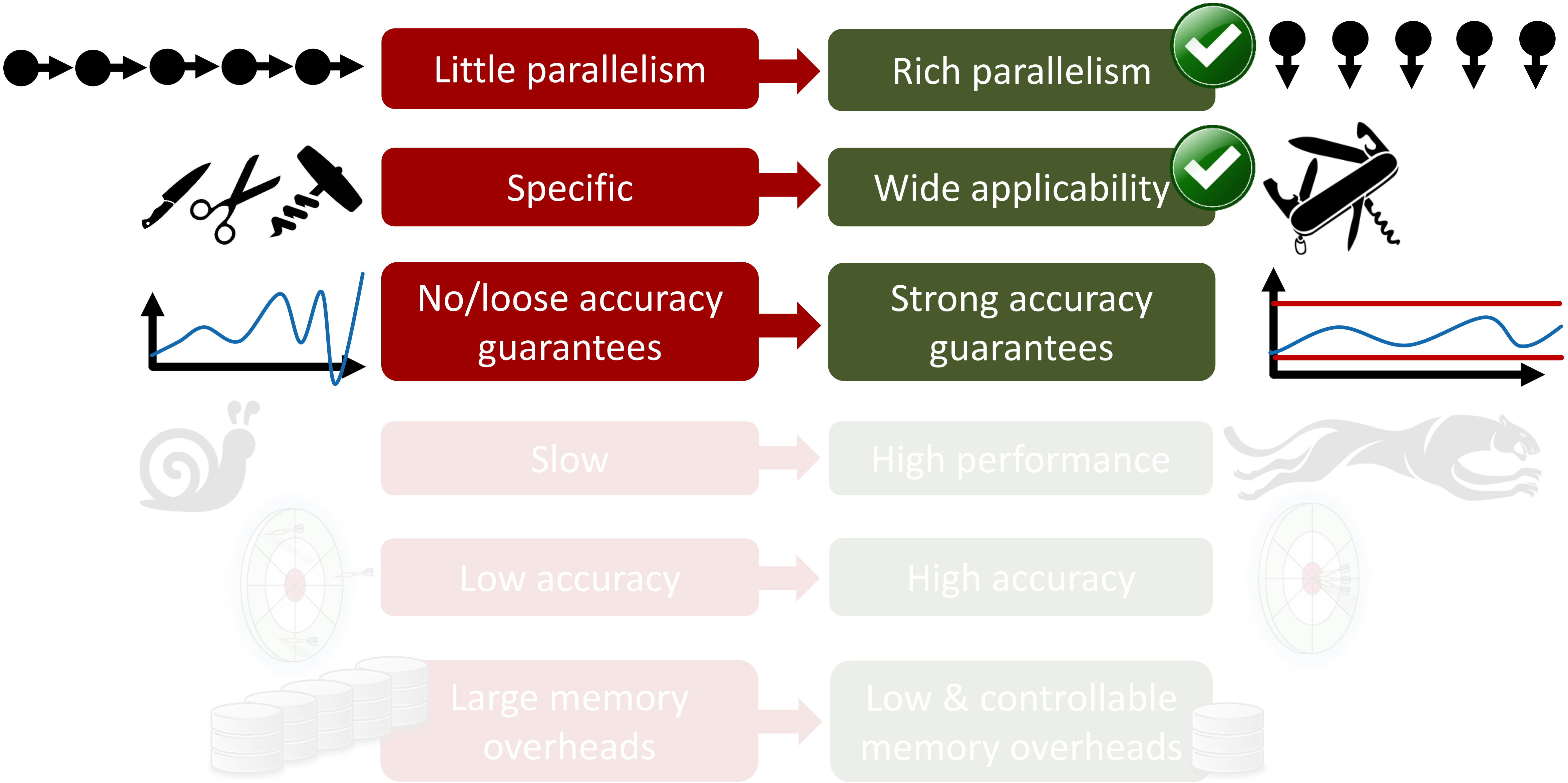
ProbGraph has strong concentration bounds

ProbGraph is unlikely to deviate much from the true values



This probability decreases exponentially fast

Approximate Graph Processing: Our Objectives



Little parallelism → Rich parallelism ✓

Specific → Wide applicability ✓

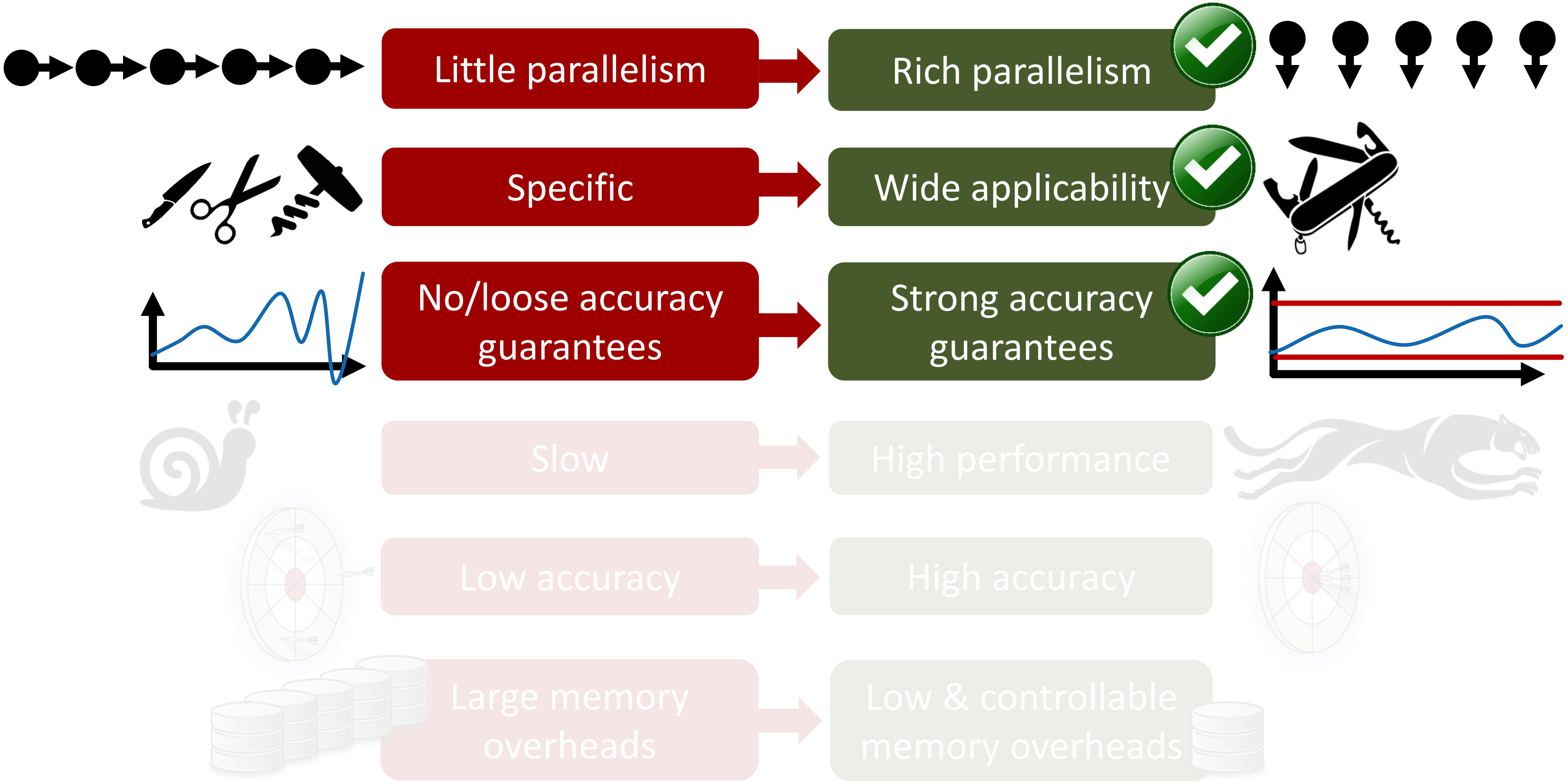
No/loose accuracy guarantees → Strong accuracy guarantees

Slow → High performance

Low accuracy → High accuracy

Large memory overheads → Low & controllable memory overheads

Approximate Graph Processing: Our Objectives



Little parallelism → Rich parallelism ✓

Specific → Wide applicability ✓

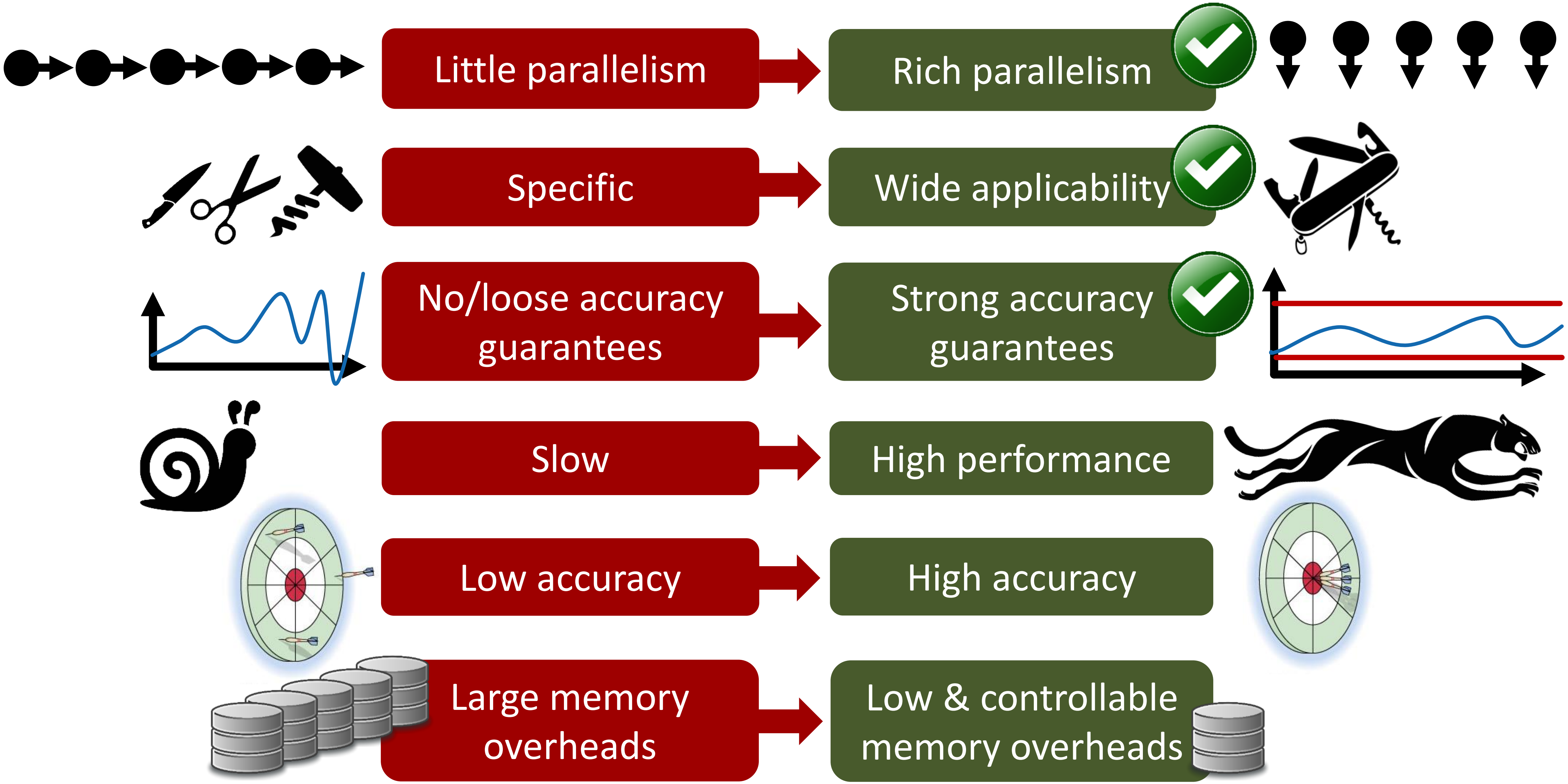
No/loose accuracy guarantees → Strong accuracy guarantees ✓

Slow → High performance

Low accuracy → High accuracy

Large memory overheads → Low & controllable memory overheads

Approximate Graph Processing: Our Objectives



Little parallelism → Rich parallelism ✓

Specific → Wide applicability ✓

No/loose accuracy guarantees → Strong accuracy guarantees ✓

Slow → High performance

Low accuracy → High accuracy

Large memory overheads → Low & controllable memory overheads

Evaluation: Used Machines & Objectives

Evaluation: Used Machines & Objectives

CSCS Cray Piz Daint,
64 GB per compute node

Evaluation: Used Machines & Objectives

Dell PowerEdge R910 server

CSCS Cray Piz Daint,
64 GB per compute node

Evaluation: Used Machines & Objectives

Goal: One design with…

large speedups +

small & controlled accuracy loss +

small & controlled memory requirements

Dell PowerEdge R910 server

CSCS Cray Piz Daint,
64 GB per compute node

Considered Graph Datasets

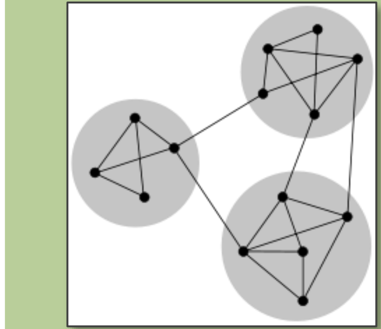
Considered Graph Datasets

67 graph datasets,
15 areas,
5 major graph dataset repositories

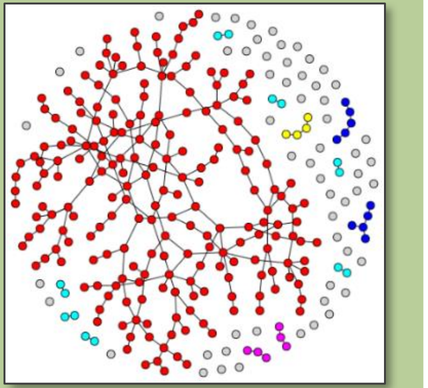
Considered Graph Datasets

67 graph datasets,
15 areas,
5 major graph
dataset repositories

Synthetic graphs



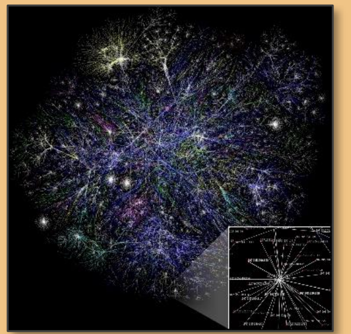
Kronecker [1]



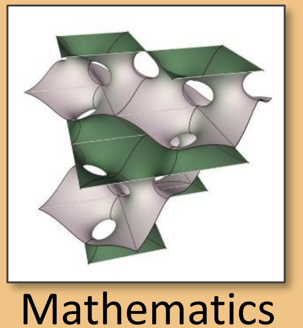
Erdös-Rényi [2]

Real-world graphs

Social networks

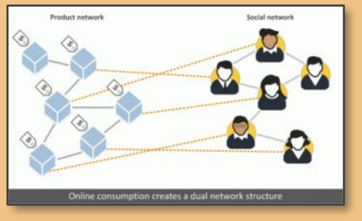


Web graphs

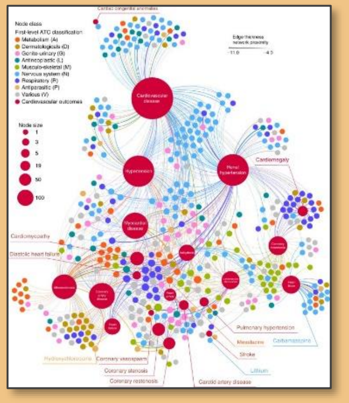


Mathematics

Purchases

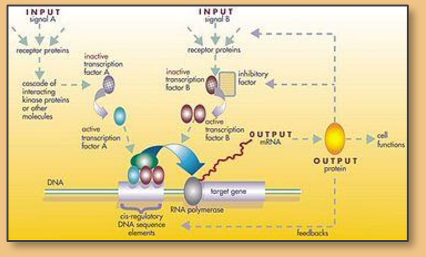


Road nets

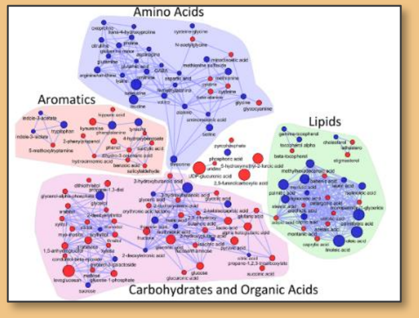


Medicine

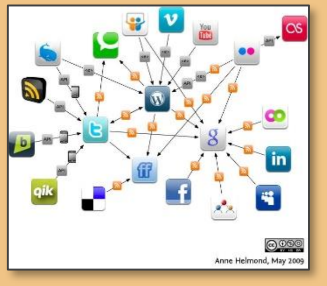
Gene functions



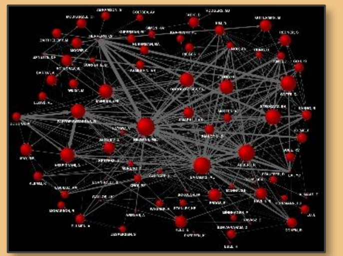
Brain structure



Chemistry



Communication



Citation graphs

Compute graphs

Economic nets

[1] J. Leskovec et al. Kronecker Graphs: An Approach to Modeling Networks. J. Mach. Learn. Research. 2010.
[2] P. Erdos and A. Renyi. On the evolution of random graphs. Pub. Math. Inst. Hun. A. Science. 1960.

Considered Graph Datasets

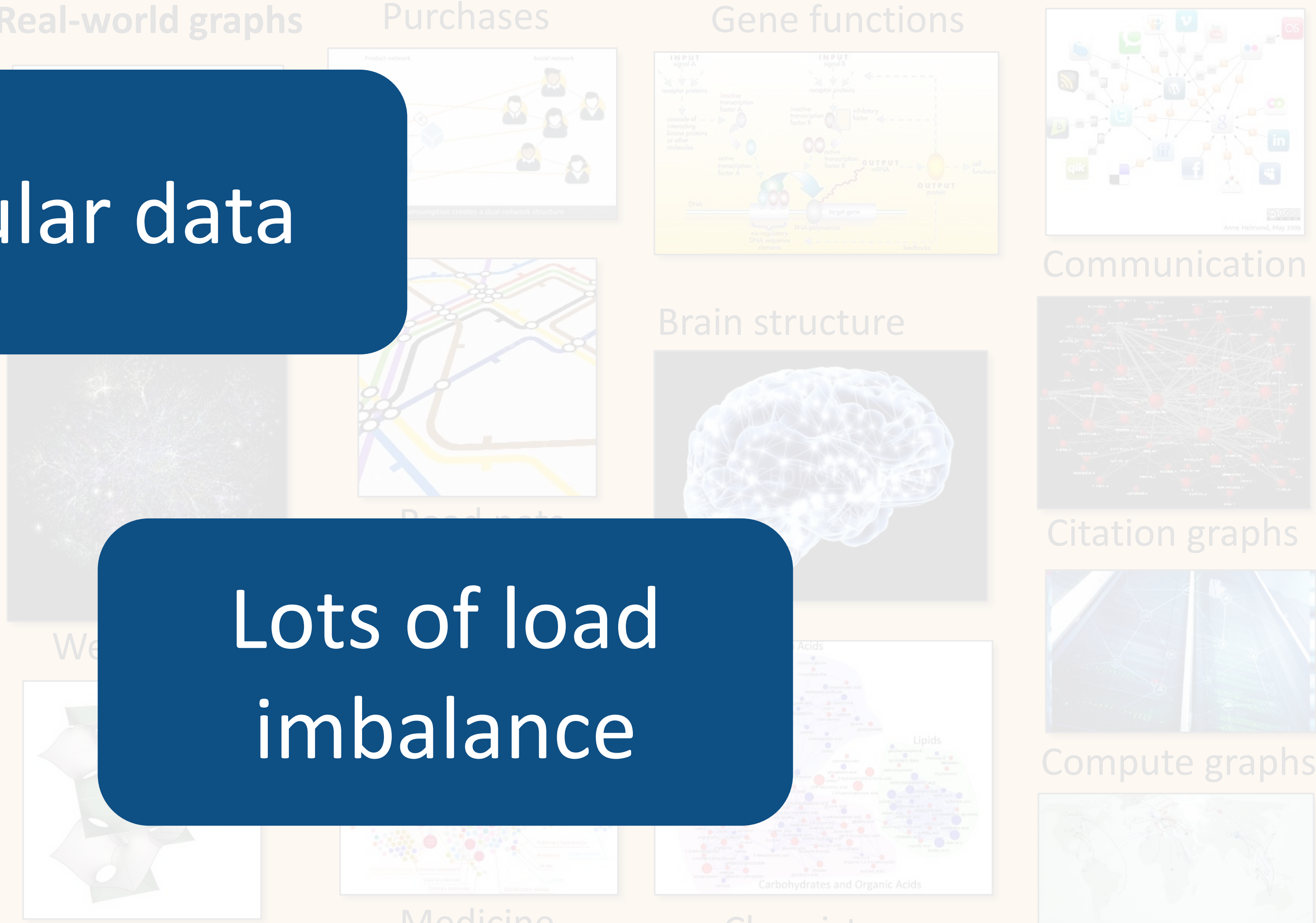
67

5

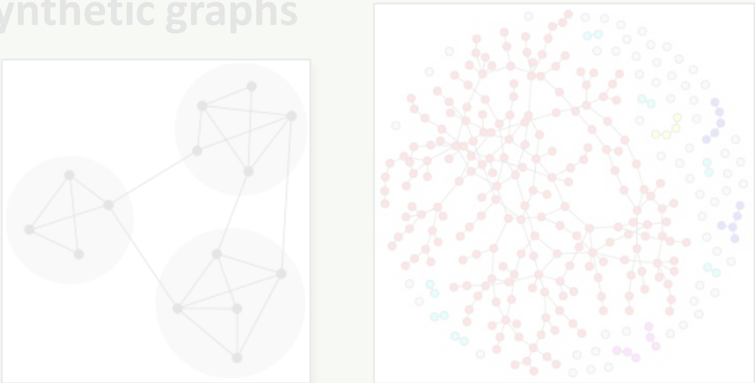
dataset repositories

Highly irregular data

Real-world graphs



Synthetic graphs



Lots of load imbalance

[1] J. Leskovec et al. Kronecker Graphs: An Approach to Modeling Networks. J. Mach. Learn. Research. 2010.
[2] P. Erdos and A. Renyi. On the evolution of random graphs. Pub. Math. Inst. Hun. A. Science. 1960.

Triangle Counting

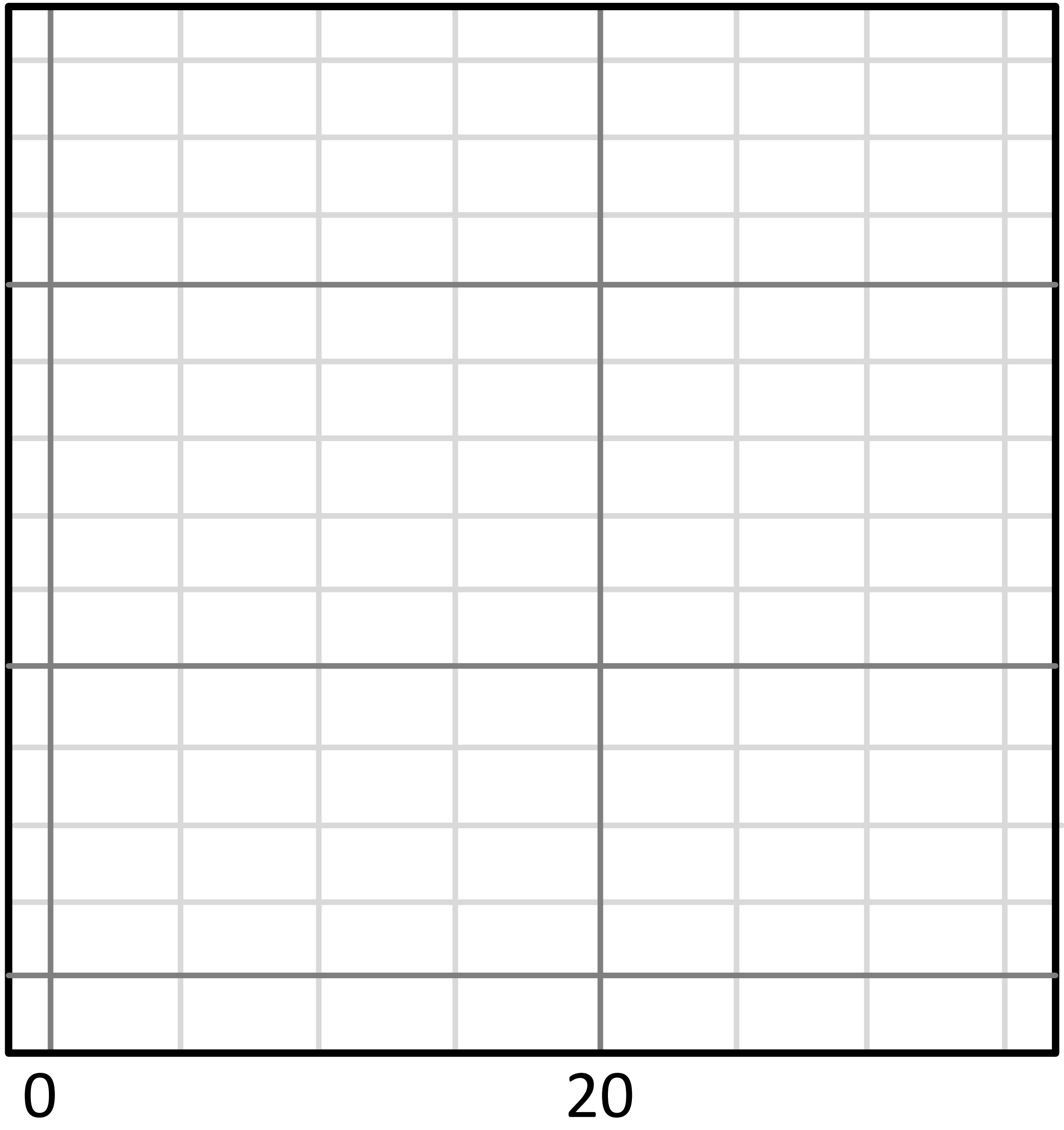
Cores/threads: 32
Max memory overhead: 20%

Triangle Counting

Cores/threads: 32
Max memory overhead: 20%

Triangle Counting

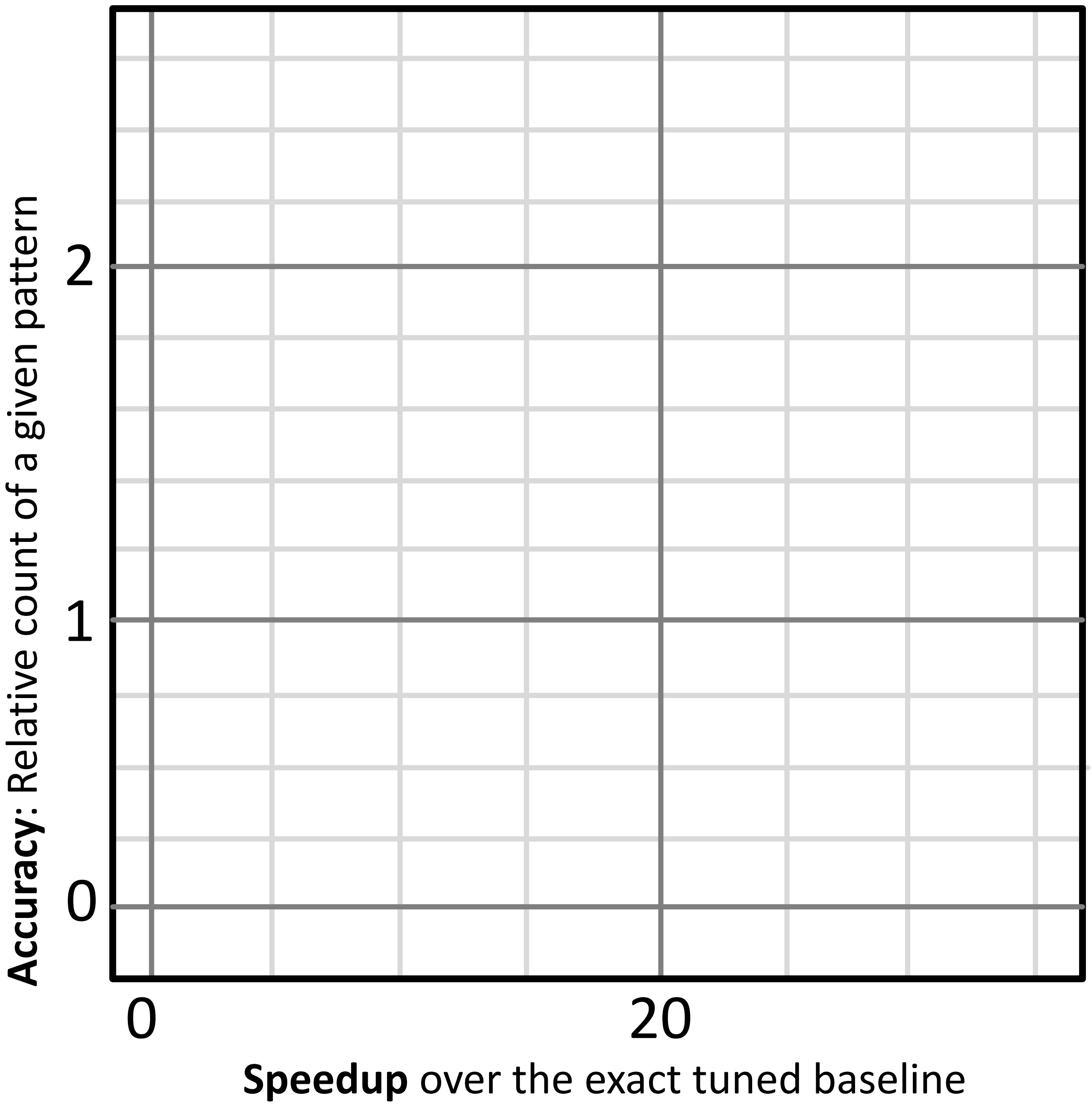
Cores/threads: 32
Max memory overhead: 20%



Speedup over the exact tuned baseline

Triangle Counting

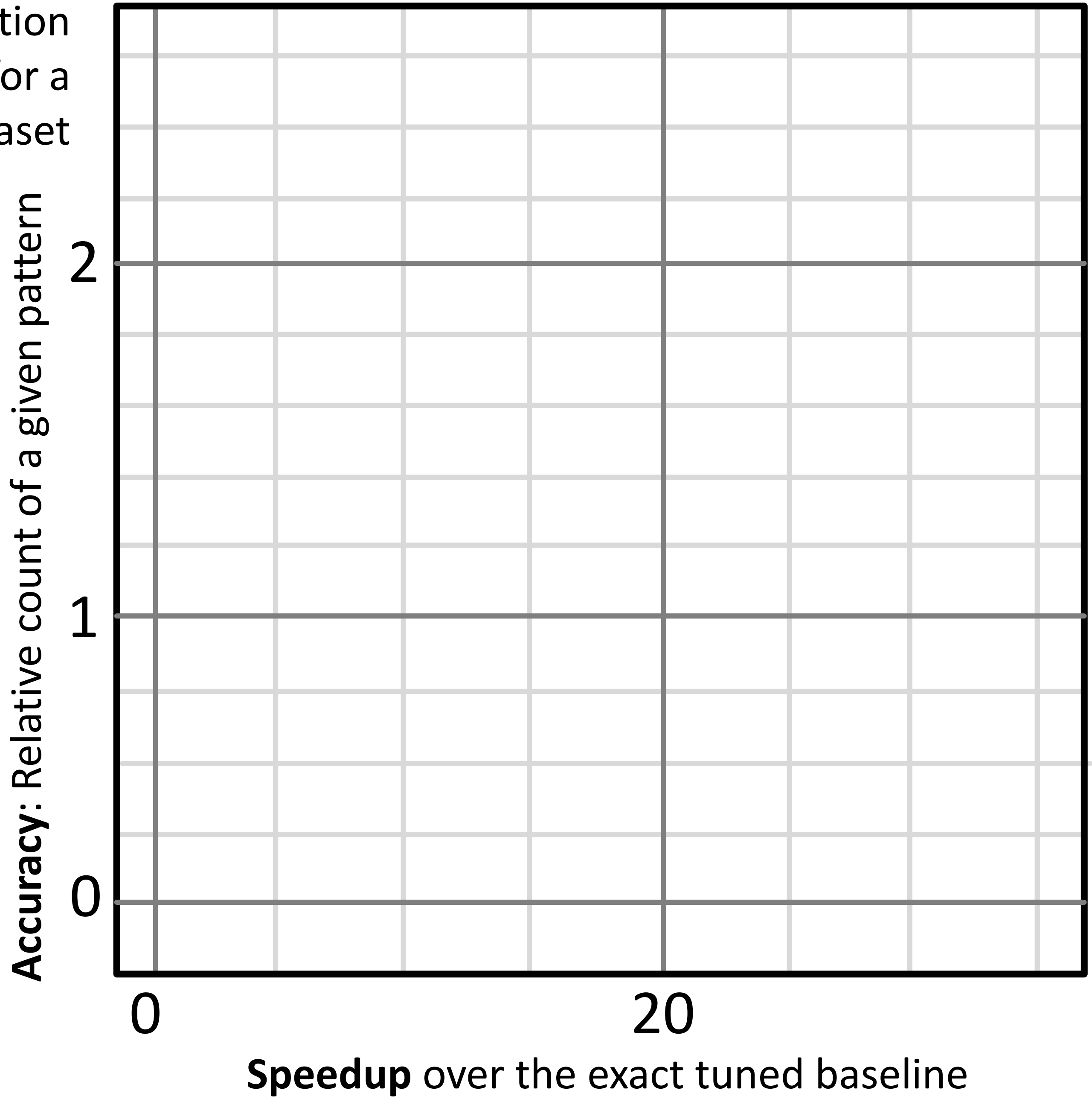
Cores/threads: 32
Max memory overhead: 20%



Triangle Counting

Cores/threads: 32
Max memory overhead: 20%

Each data point: the execution of a given scheme for a specific graph dataset

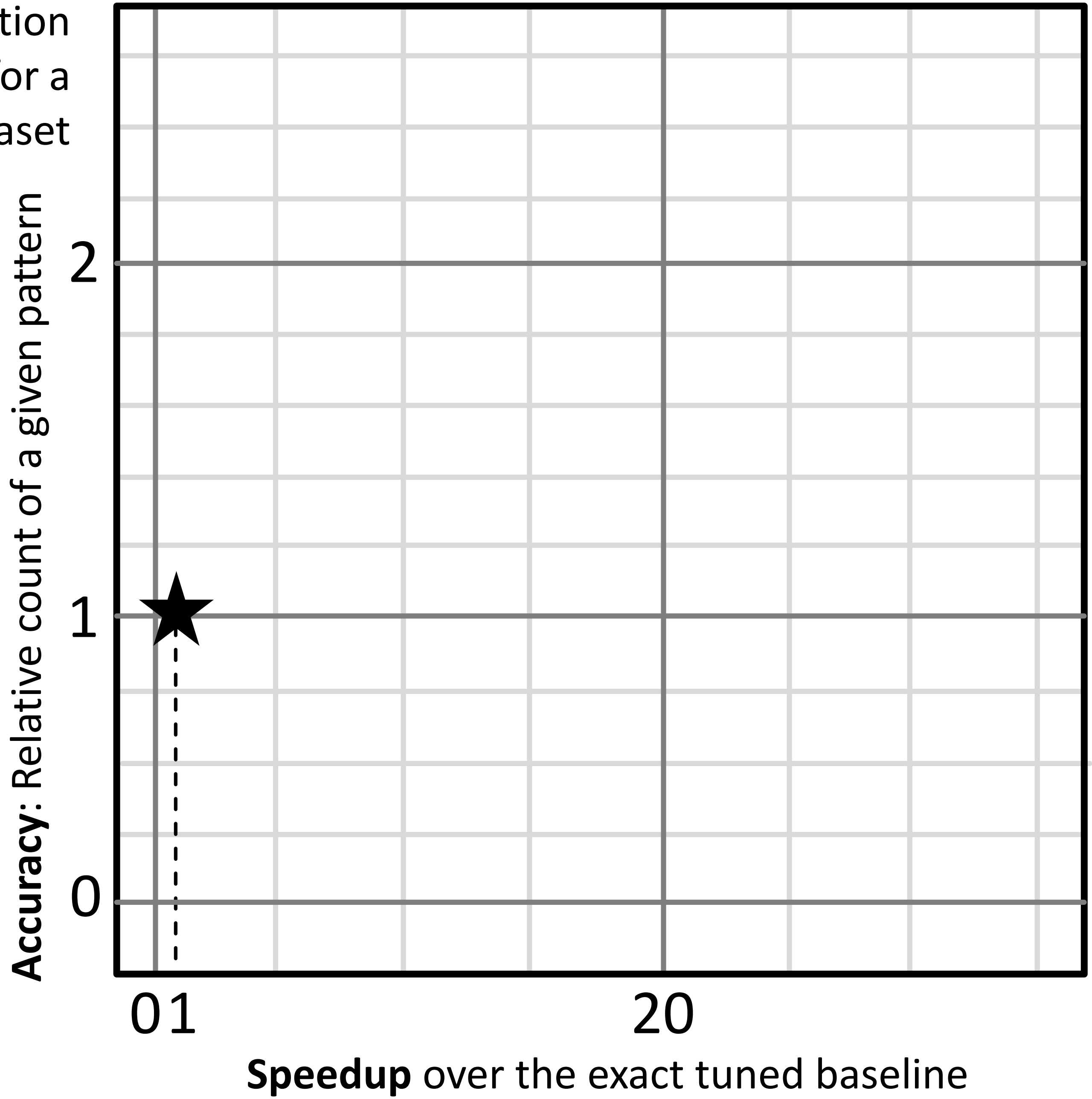


Triangle Counting

Cores/threads: 32
Max memory overhead: 20%

Each data point: the execution of a given scheme for a specific graph dataset

★ Exact baseline [1]



[1] S. Beamer et al., „The GAP Benchmark Suite”. 2015

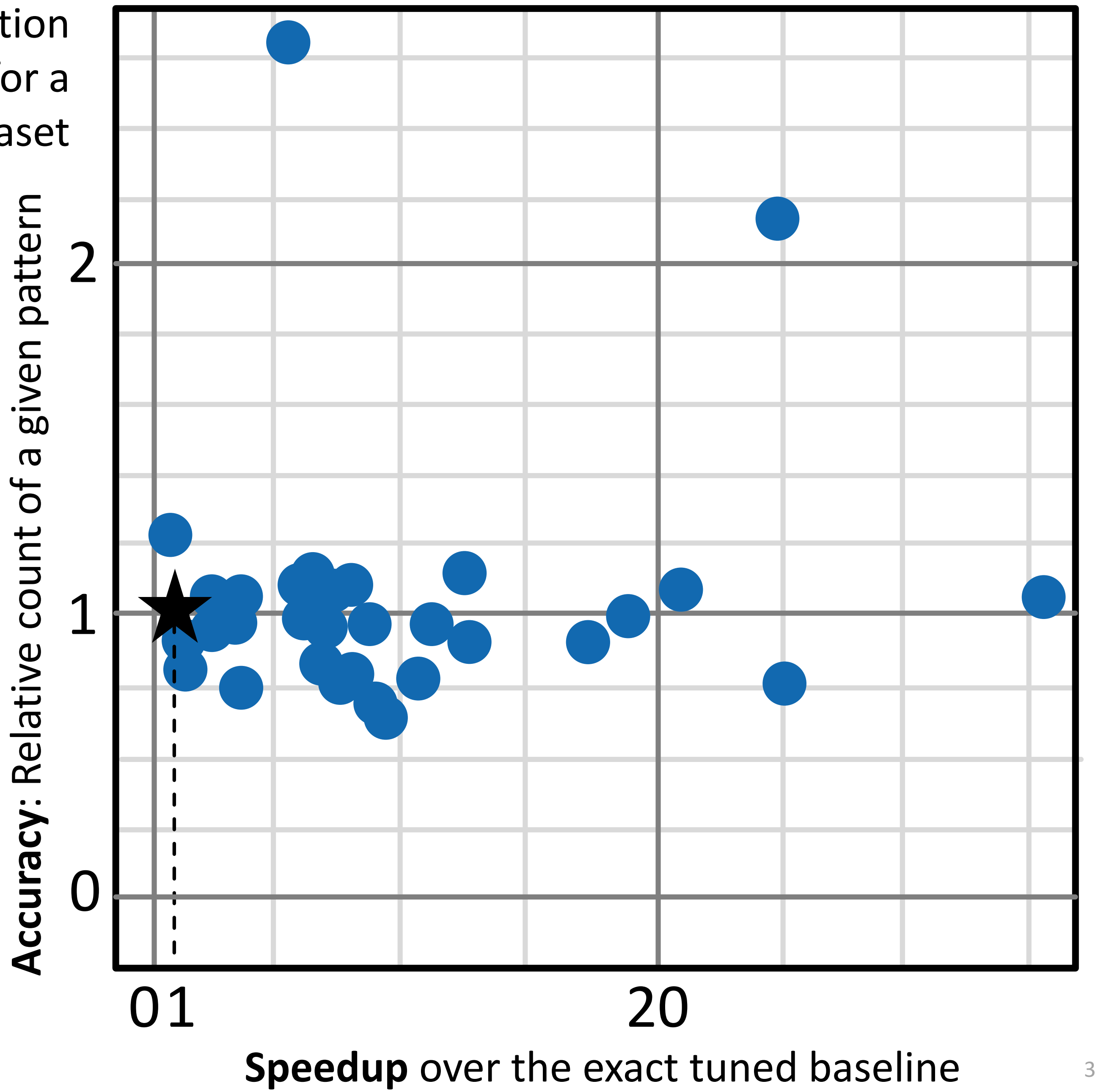
Triangle Counting

Cores/threads: 32
Max memory overhead: 20%

Each data point: the execution of a given scheme for a specific graph dataset

● ProbGraph

★ Exact baseline [1]



[1] S. Beamer et al., „The GAP Benchmark Suite". 2015

Triangle Counting

Cores/threads: 32

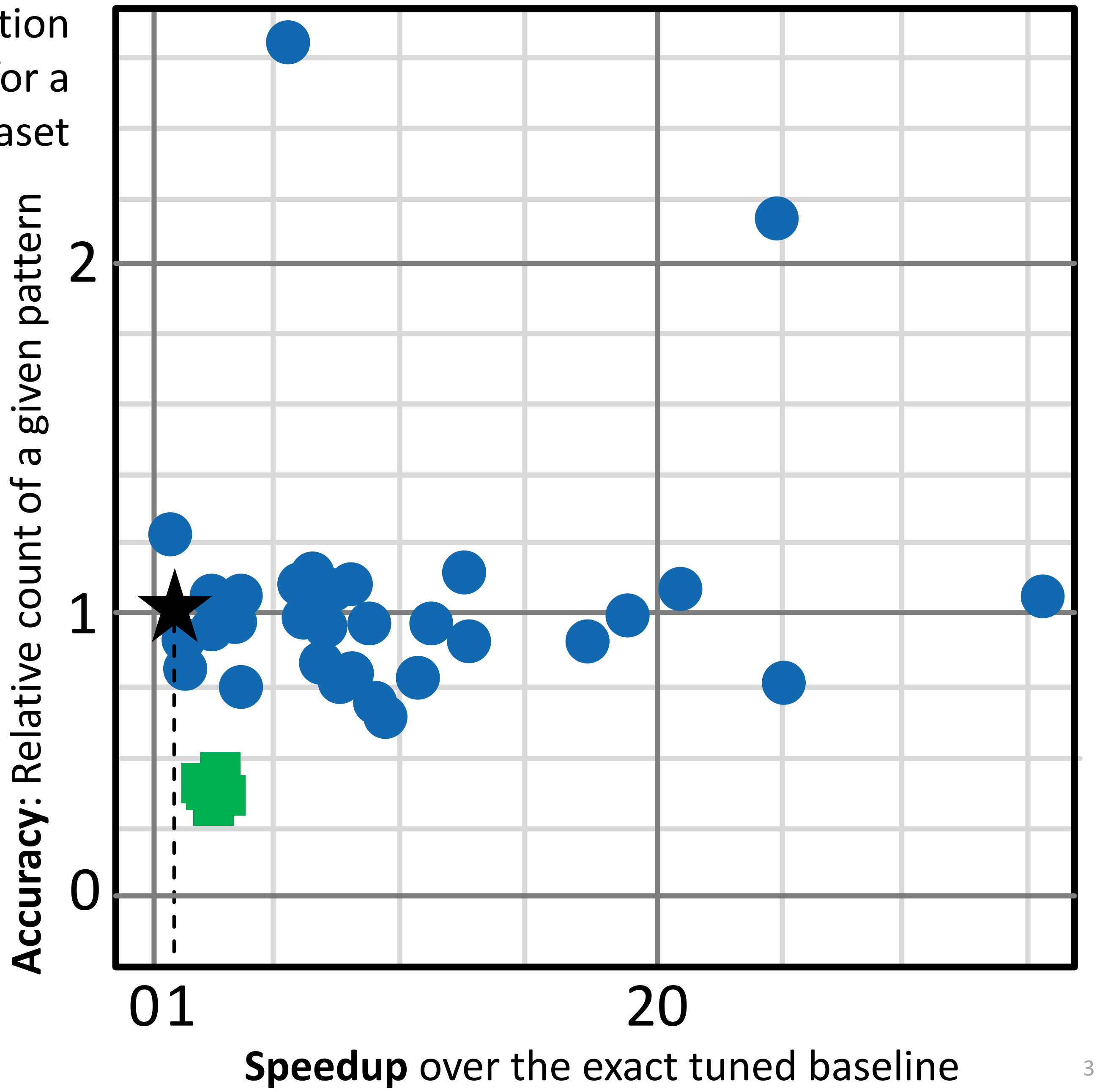
Max memory overhead: 20%

Each data point: the execution of a given scheme for a specific graph dataset

● ProbGraph

★ Exact baseline [1]

🟩 Heuristics, no formal guarantees [2]



[1] S. Beamer et al., „The GAP Benchmark Suite". 2015

[2] S. Singh et al., "Scalable and performant graph processing on GPUs using approximate computing". IEEE TMSCS. 2018

Triangle Counting

Cores/threads: 32

Max memory overhead: 20%

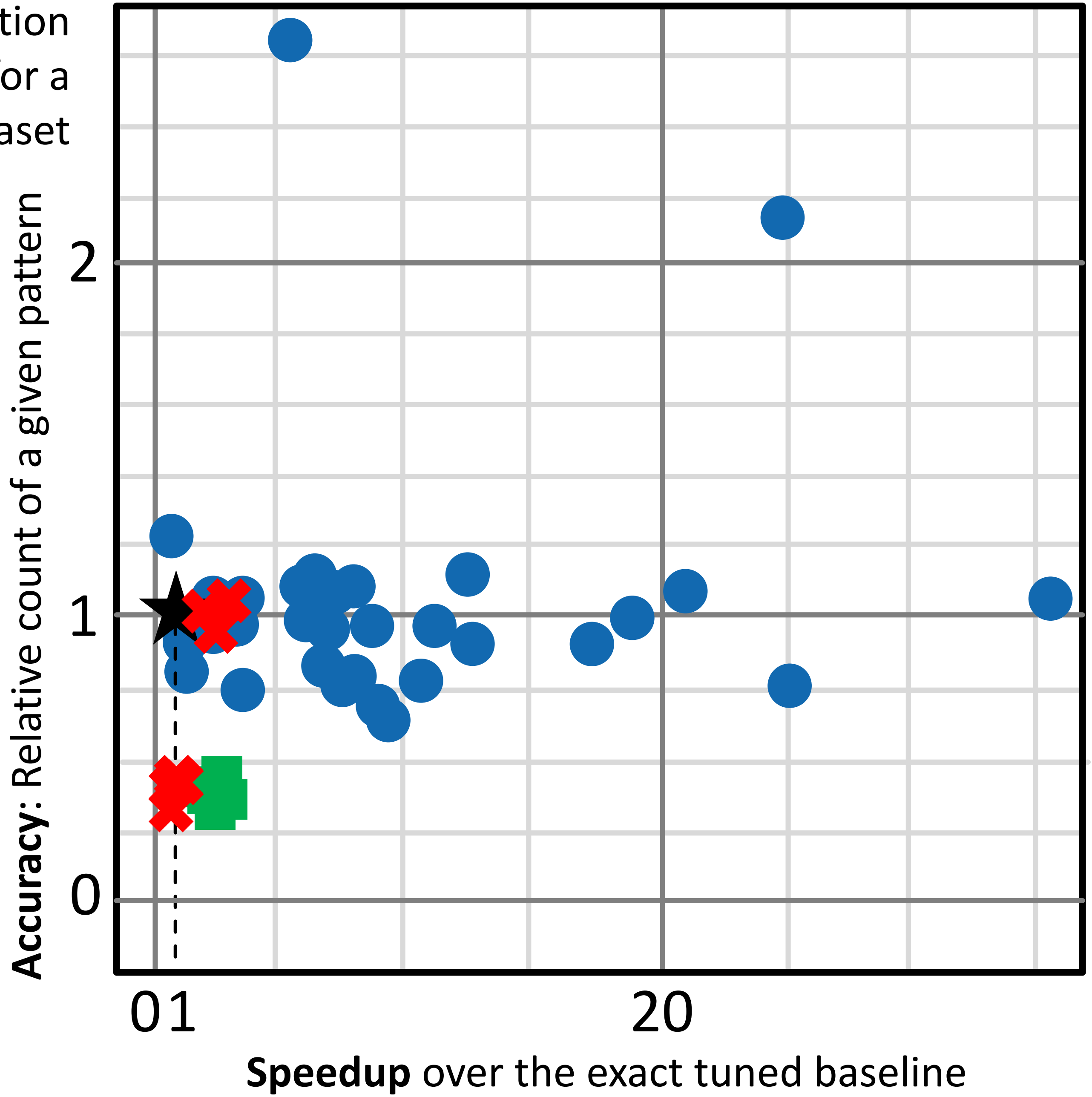
Each data point: the execution of a given scheme for a specific graph dataset

● ProbGraph

★ Exact baseline [1]

🟩 Heuristics, no formal guarantees [2]

✖ Heuristics, formal guarantees [3-4]



[1] S. Beamer et al., „The GAP Benchmark Suite". 2015

[2] S. Singh et al., "Scalable and performant graph processing on GPUs using approximate computing". IEEE TMSCS. 2018

[3] R. Pagh et al., "Colorful triangle counting and a mapreduce implementation". Information Processing Letters. 2012

[4] Z. Shang et al., "Auto-approximation of graph computing". VLDB. 2014

Triangle Counting

Cores/threads: 32

Max memory overhead: 20%

Each data point: the execution of a given scheme for a specific graph dataset

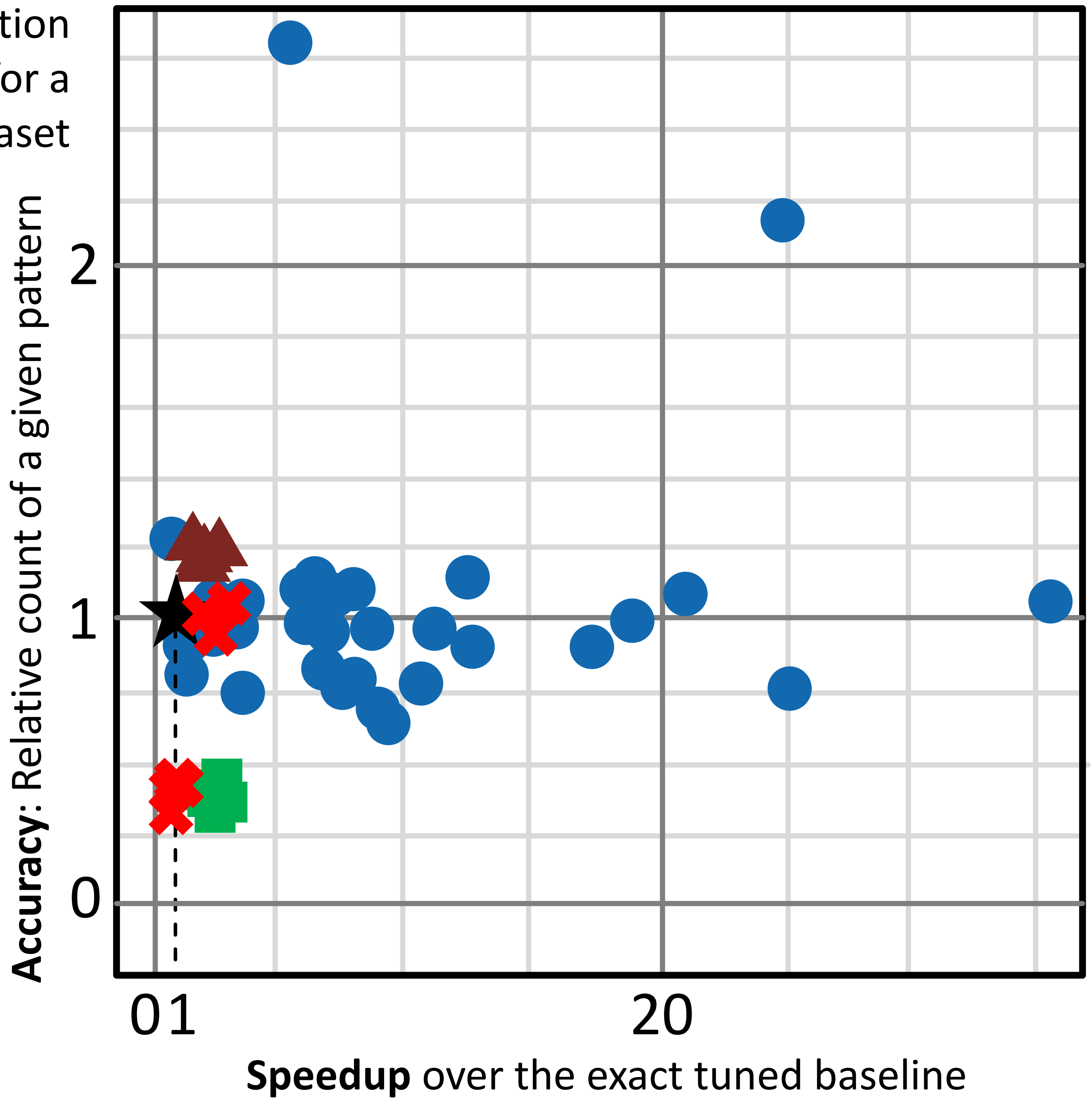
● ProbGraph

★ Exact baseline [1]

■ Heuristics, no formal guarantees [2]

✖ Heuristics, formal guarantees [3-4]

▲ Lossy graph compression [5-6]



[1] S. Beamer et al., „The GAP Benchmark Suite”. 2015

[2] S. Singh et al., “Scalable and performant graph processing on GPUs using approximate computing”. IEEE TMSCS. 2018

[3] R. Pagh et al., “Colorful triangle counting and a mapreduce implementation”. Information Processing Letters. 2012

[4] Z. Shang et al., “Auto-approximation of graph computing”. VLDB. 2014

[5] C. E. Tsourakakis et al., “Doulion: counting triangles in massive graphs with a coin”. ACM KDD. 2009.

[6] M. Besta et al., “Slim graph: Practical lossy graph compression for approximate graph processing, storage, and analytics”. ACM/IEEE SC. 2019

Triangle Counting

Cores/threads: 32

Max memory overhead: 20%

Each data point: the execution of a given scheme for a specific graph dataset

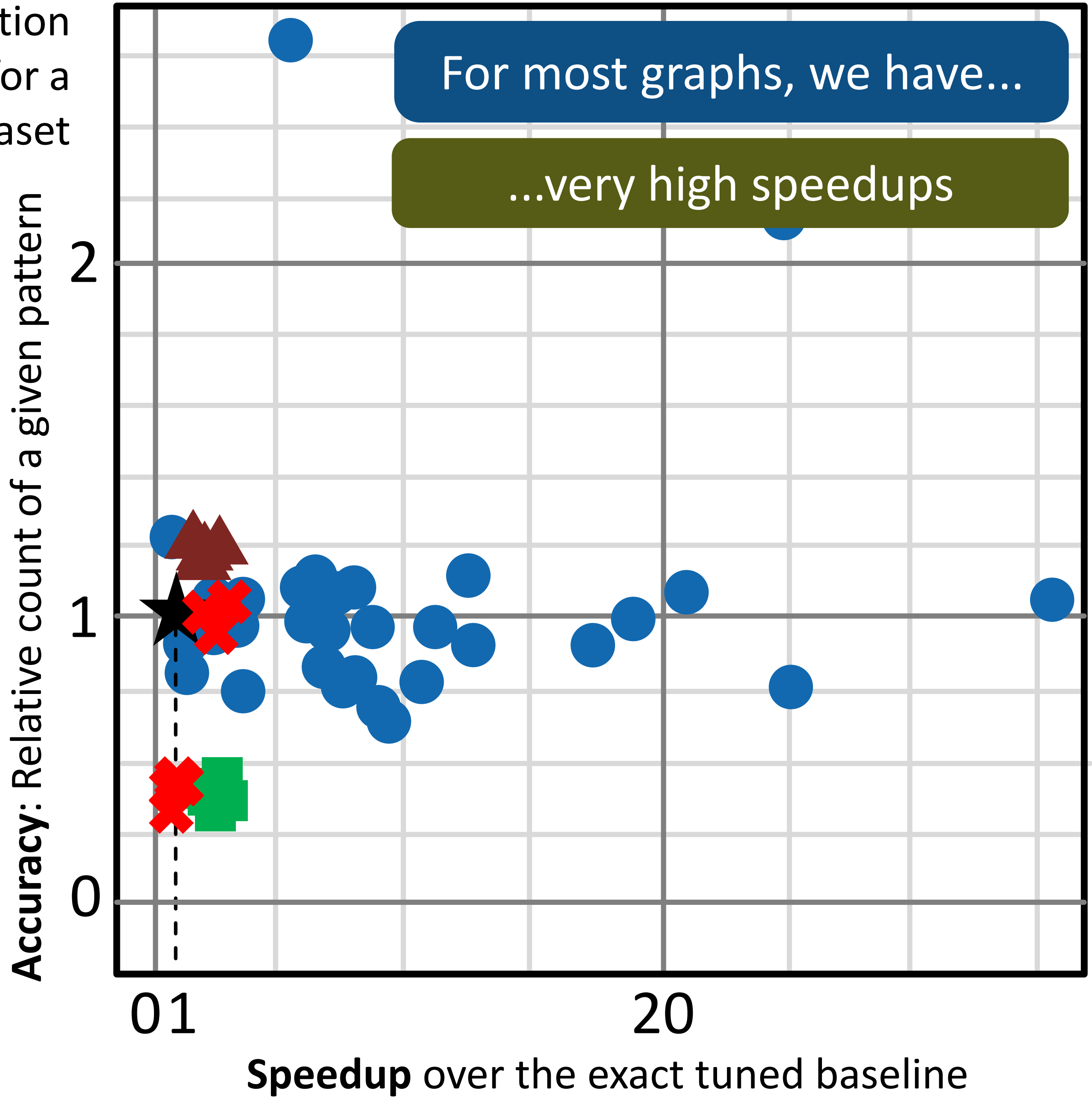
● ProbGraph

★ Exact baseline [1]

■ Heuristics, no formal guarantees [2]

✖ Heuristics, formal guarantees [3-4]

▲ Lossy graph compression [5-6]



For most graphs, we have...

...very high speedups

[1] S. Beamer et al., „The GAP Benchmark Suite". 2015

[2] S. Singh et al., "Scalable and performant graph processing on GPUs using approximate computing". IEEE TMSCS. 2018

[3] R. Pagh et al., "Colorful triangle counting and a mapreduce implementation". Information Processing Letters. 2012

[4] Z. Shang et al., "Auto-approximation of graph computing". VLDB. 2014

[5] C. E. Tsourakakis et al., "Doulion: counting triangles in massive graphs with a coin". ACM KDD. 2009.

[6] M. Besta et al., "Slim graph: Practical lossy graph compression for approximate graph processing, storage, and analytics". ACM/IEEE SC. 2019

Triangle Counting

Cores/threads: 32

Max memory overhead: 20%

Each data point: the execution of a given scheme for a specific graph dataset

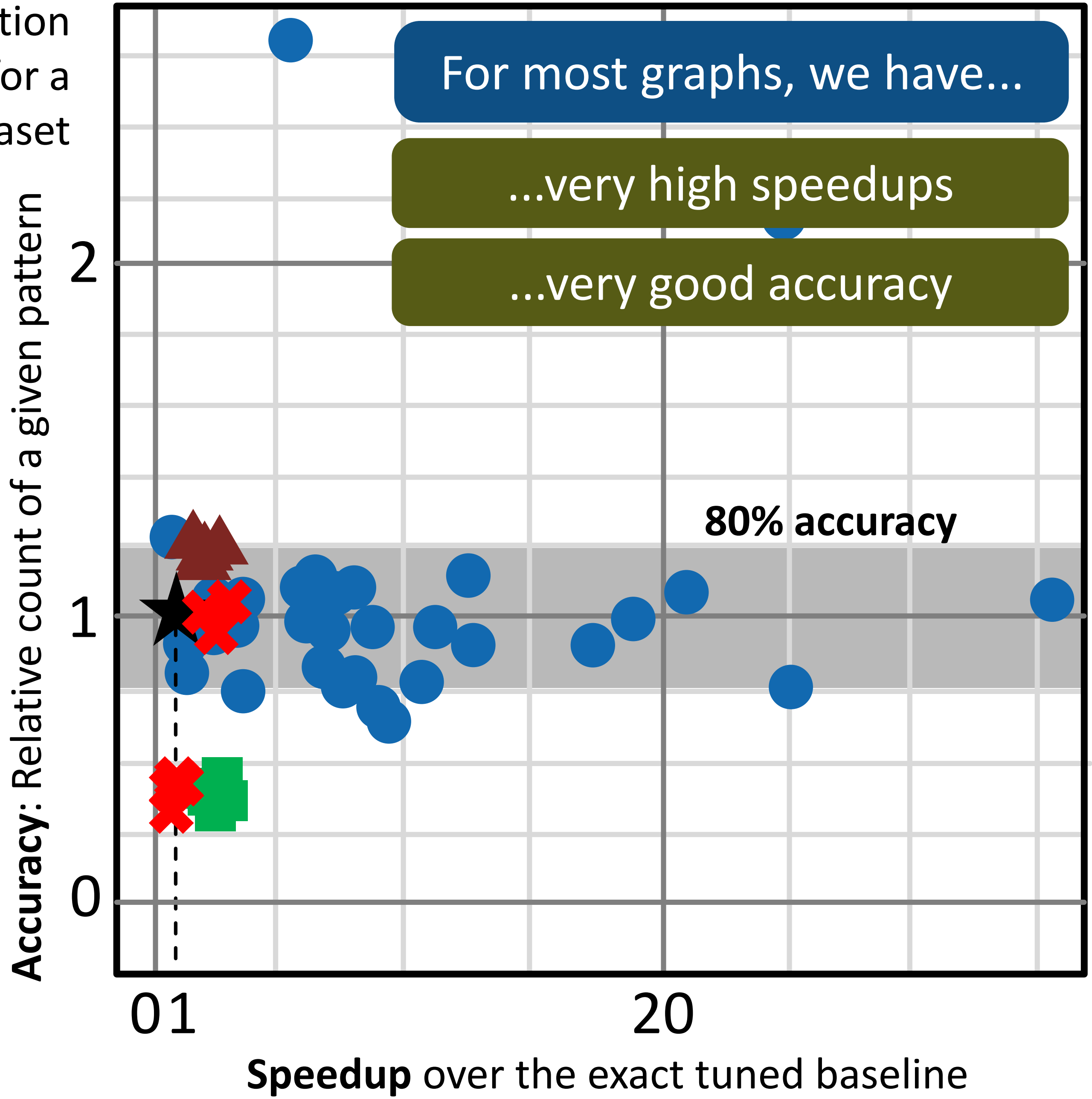
● ProbGraph

★ Exact baseline [1]

■ Heuristics, no formal guarantees [2]

✖ Heuristics, formal guarantees [3-4]

▲ Lossy graph compression [5-6]



For most graphs, we have...

...very high speedups

...very good accuracy

[1] S. Beamer et al., „The GAP Benchmark Suite". 2015

[2] S. Singh et al., "Scalable and performant graph processing on GPUs using approximate computing". IEEE TMSCS. 2018

[3] R. Pagh et al., "Colorful triangle counting and a mapreduce implementation". Information Processing Letters. 2012

[4] Z. Shang et al., "Auto-approximation of graph computing". VLDB. 2014

[5] C. E. Tsourakakis et al., "Doulion: counting triangles in massive graphs with a coin". ACM KDD. 2009.

[6] M. Besta et al., "Slim graph: Practical lossy graph compression for approximate graph processing, storage, and analytics". ACM/IEEE SC. 2019

Triangle Counting

Cores/threads: 32
Max memory overhead: 20%

Each data point: the execution of a given scheme for a specific graph dataset

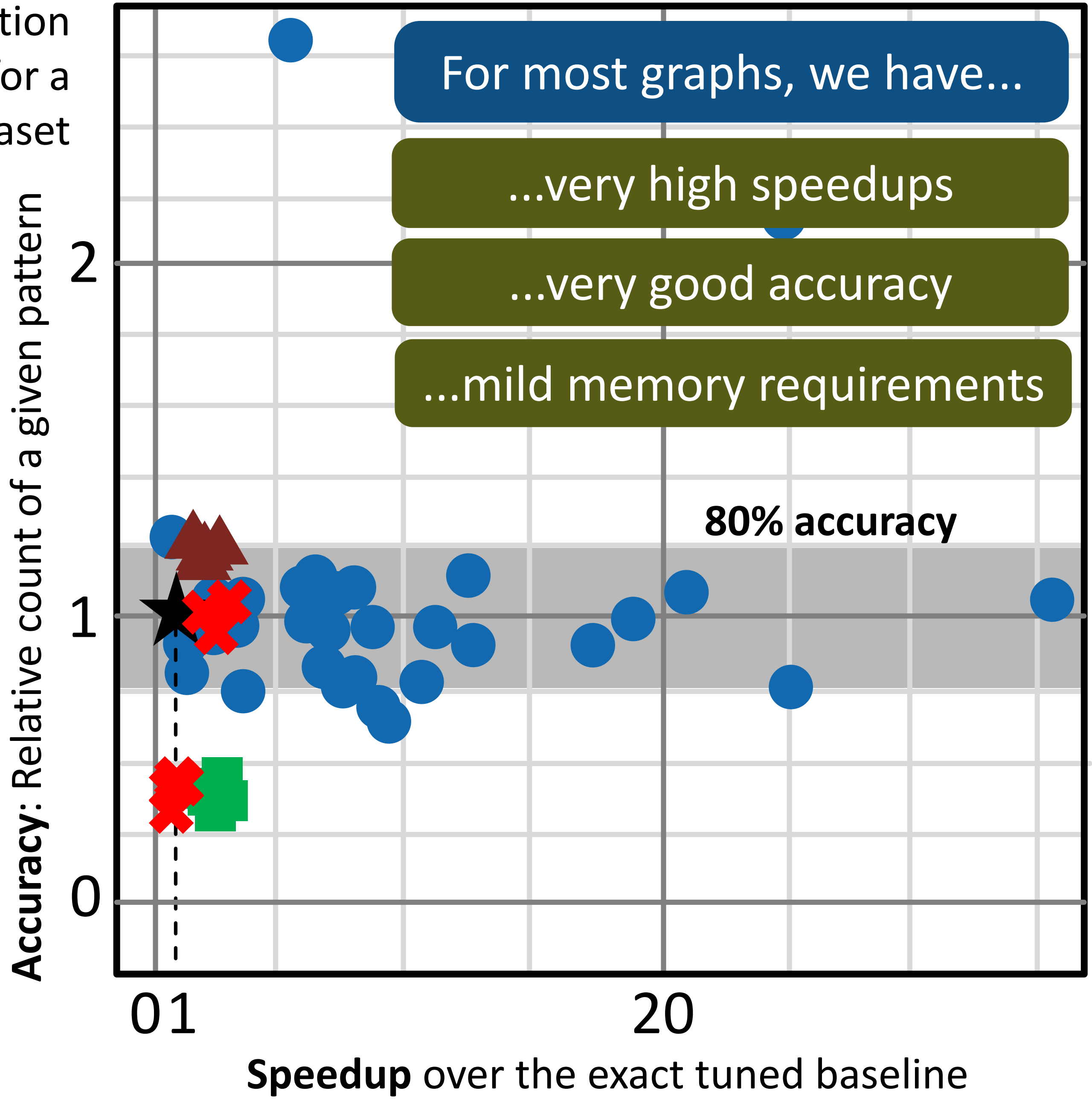
● ProbGraph

★ Exact baseline [1]

■ Heuristics, no formal guarantees [2]

✖ Heuristics, formal guarantees [3-4]

▲ Lossy graph compression [5-6]



For most graphs, we have...

...very high speedups

...very good accuracy

...mild memory requirements

[1] S. Beamer et al., „The GAP Benchmark Suite”. 2015

[2] S. Singh et al., “Scalable and performant graph processing on GPUs using approximate computing”. IEEE TMSCS. 2018

[3] R. Pagh et al., “Colorful triangle counting and a mapreduce implementation”. Information Processing Letters. 2012

[4] Z. Shang et al., “Auto-approximation of graph computing”. VLDB. 2014

[5] C. E. Tsourakakis et al., “Doulion: counting triangles in massive graphs with a coin”. ACM KDD. 2009.

[6] M. Besta et al., “Slim graph: Practical lossy graph compression for approximate graph processing, storage, and analytics”. ACM/IEEE SC. 2019

4-Clique Counting

Cores/threads: 32
Max memory overhead: 20%

Each data point: the execution of a given scheme for a specific graph dataset

★ Exact baseline [1]

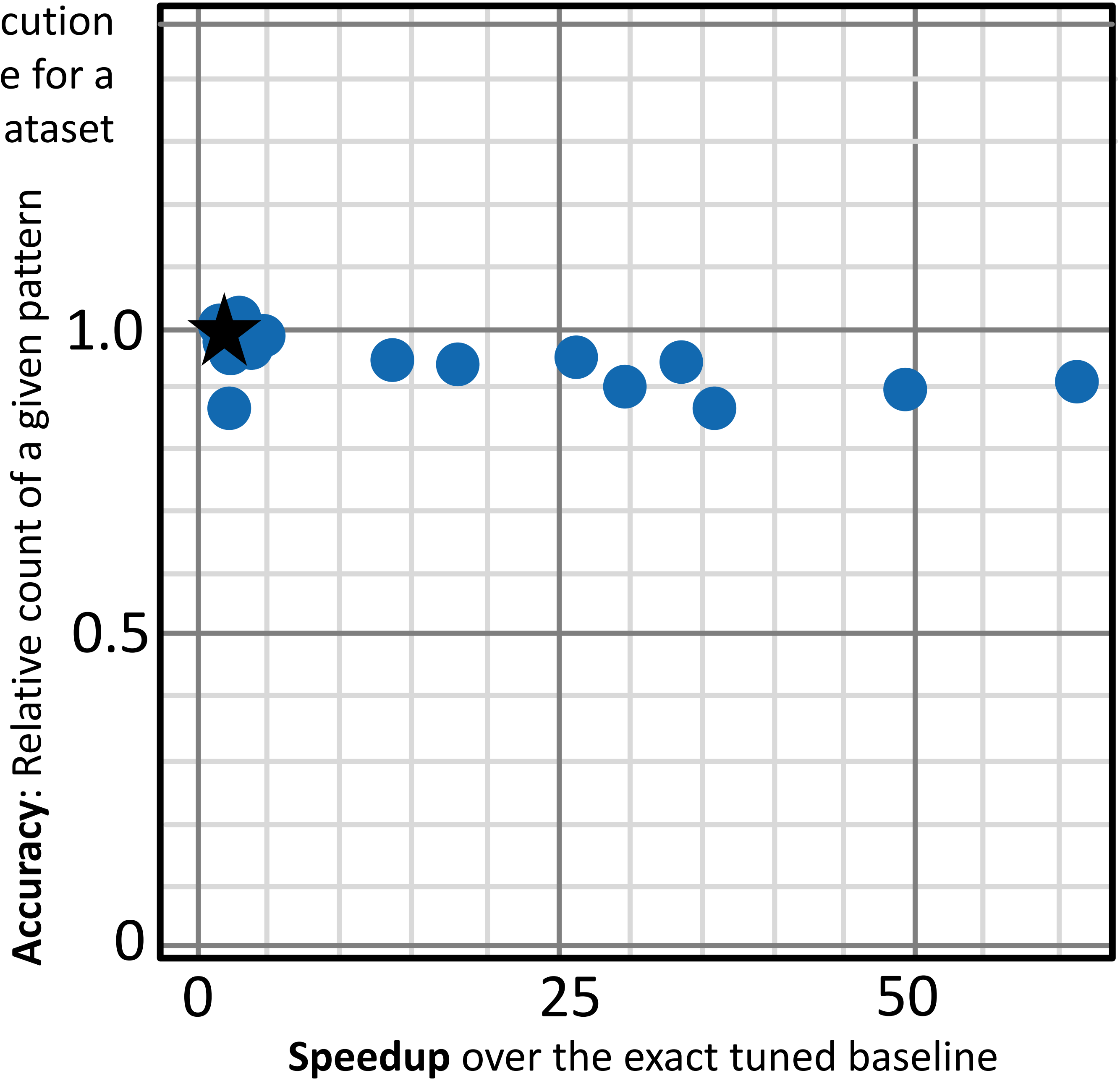
● ProbGraph

For most graphs, we have...

...very high speedups

...very good accuracy

...mild memory requirements



[1] based on S. Beamer et al., „The GAP Benchmark Suite". 2015

4-Clique Counting

Cores/threads: 32
Max memory overhead: 20%

Each data point: the execution of a given scheme for a specific graph dataset

★ Exact baseline [1]

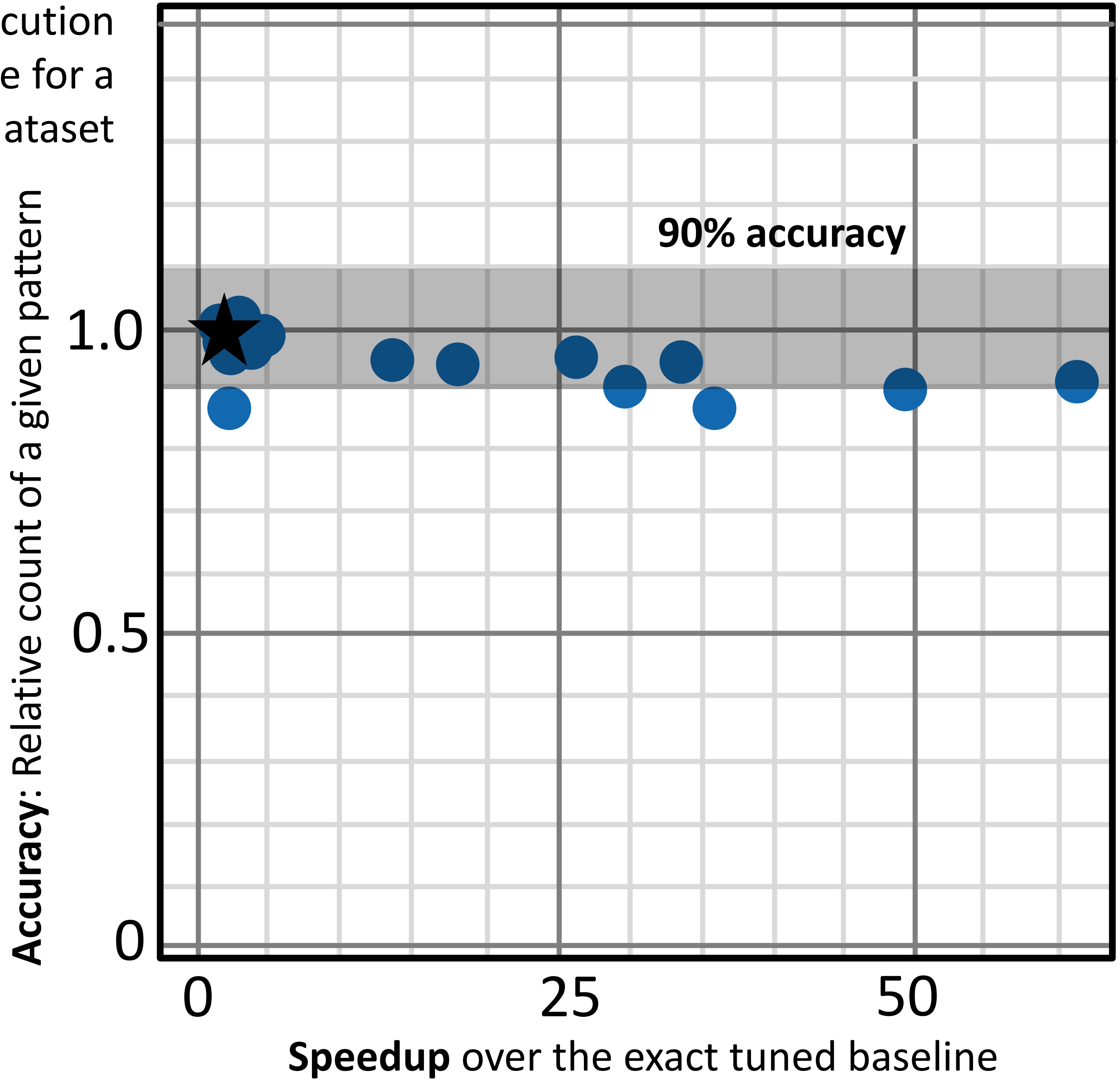
● ProbGraph

For most graphs, we have...

...very high speedups

...very good accuracy

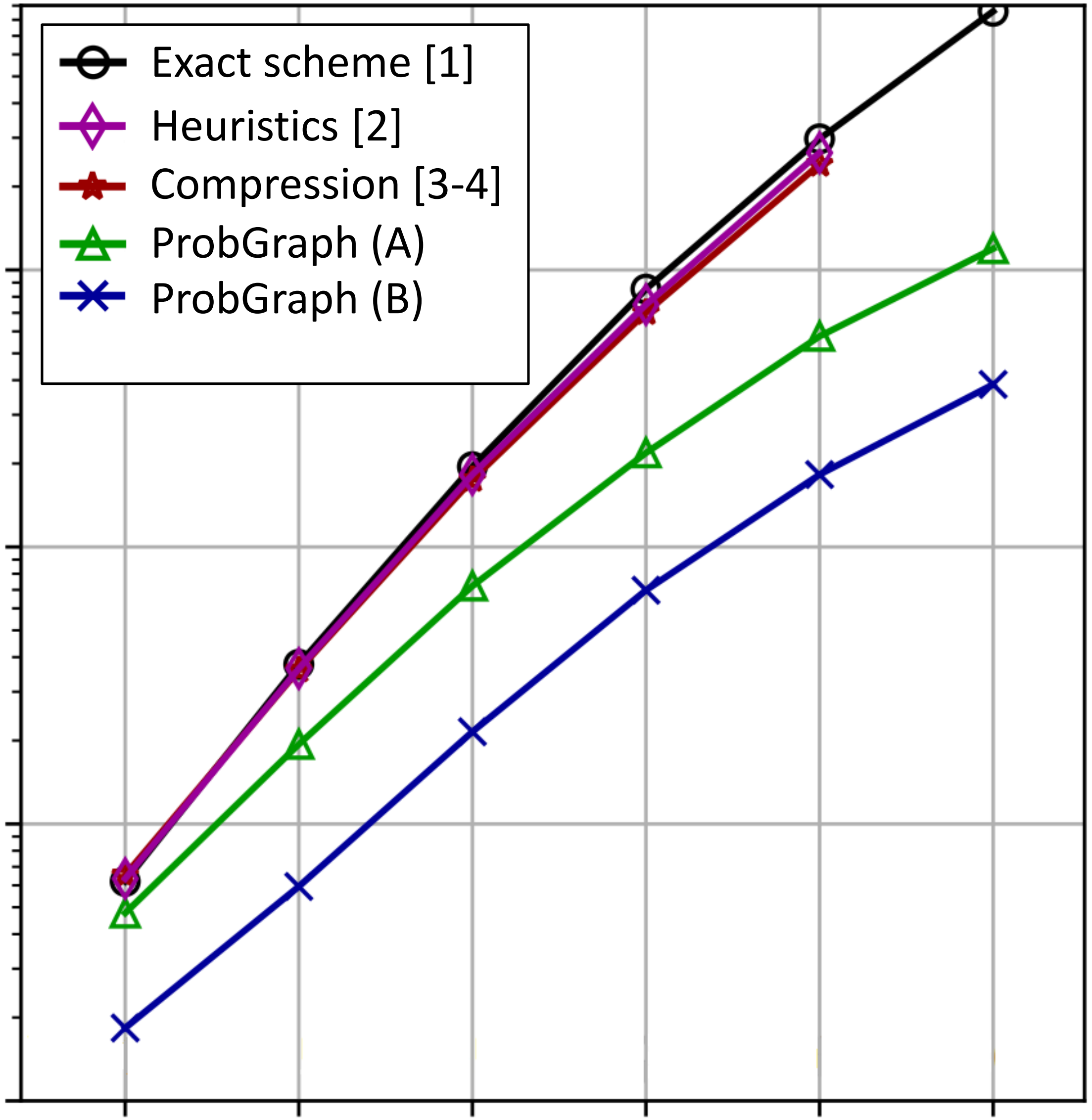
...mild memory requirements



[1] based on S. Beamer et al., „The GAP Benchmark Suite". 2015

Clustering (Scaling)

Max memory overhead: 20%



[1] S. Beamer et al., „The GAP Benchmark Suite”. 2015

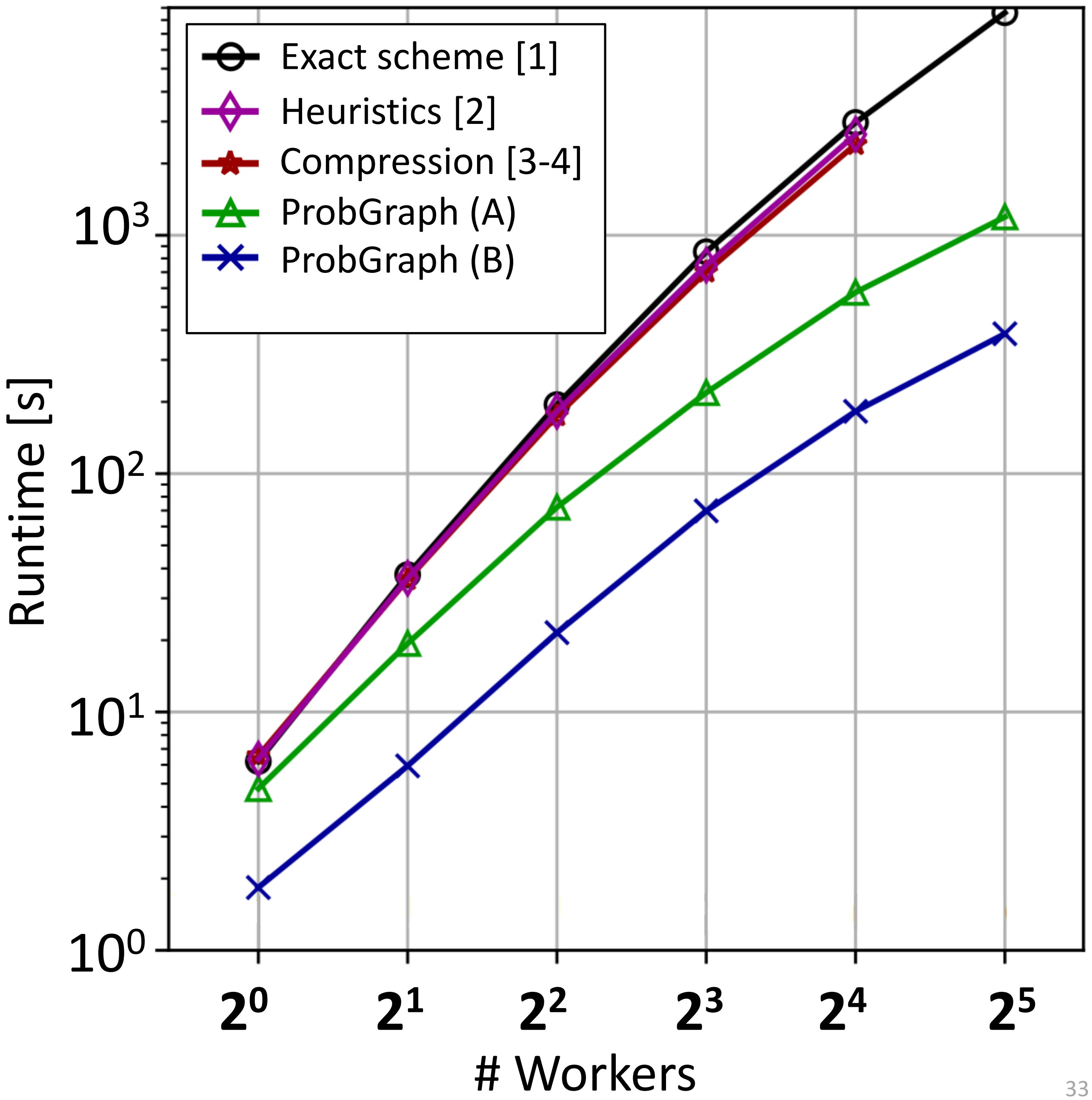
[2] R. Pagh et al., “Colorful triangle counting and a mapreduce implementation”. Information Processing Letters. 2012

[3] C. E. Tsourakakis et al., “Doulion: counting triangles in massive graphs with a coin”. ACM KDD. 2009.

[4] M. Besta et al., “Slim graph: Practical lossy graph compression for approximate graph processing, storage, and analytics”. ACM/IEEE SC. 2019

Clustering (Scaling)

Max memory overhead: 20%



[1] S. Beamer et al., „The GAP Benchmark Suite". 2015

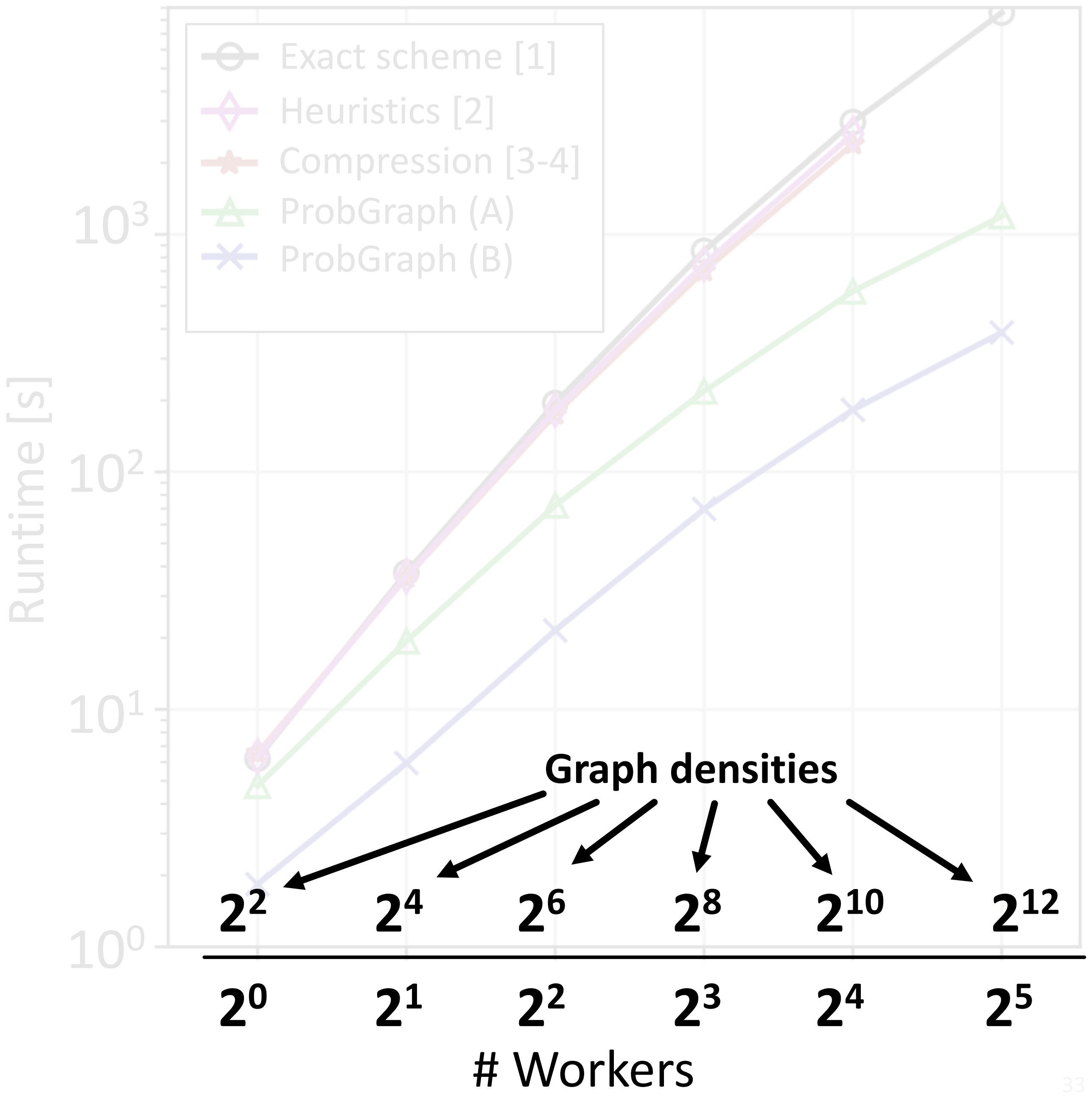
[2] R. Pagh et al., "Colorful triangle counting and a mapreduce implementation". Information Processing Letters. 2012

[3] C. E. Tsourakakis et al., "Doulion: counting triangles in massive graphs with a coin". ACM KDD. 2009.

[4] M. Besta et al., "Slim graph: Practical lossy graph compression for approximate graph processing, storage, and analytics". ACM/IEEE SC. 2019

Clustering (Scaling)

Max memory overhead: 20%



[1] S. Beamer et al., „The GAP Benchmark Suite". 2015

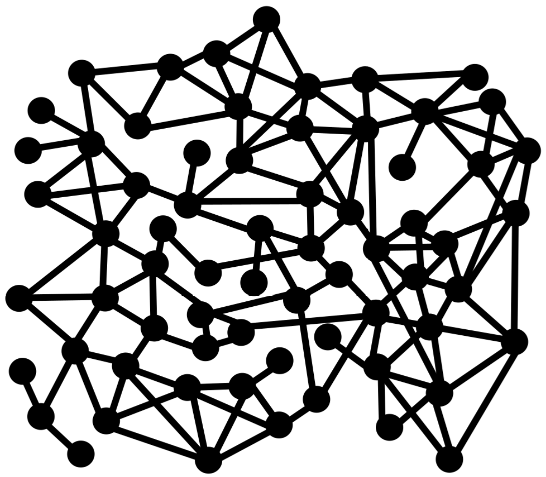
[2] R. Pagh et al., "Colorful triangle counting and a mapreduce implementation". Information Processing Letters. 2012

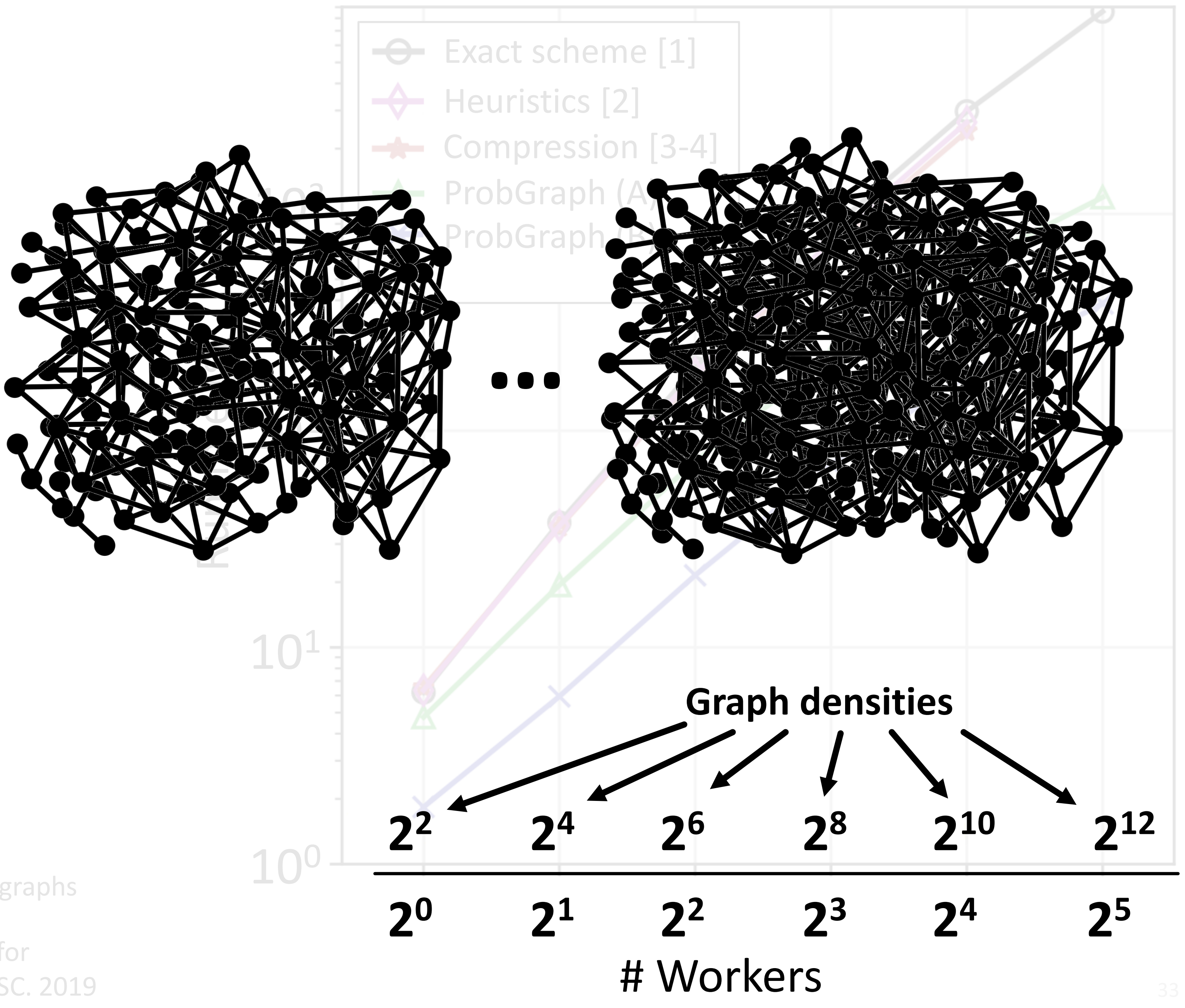
[3] C. E. Tsourakakis et al., "Doulion: counting triangles in massive graphs with a coin". ACM KDD. 2009.

[4] M. Besta et al., "Slim graph: Practical lossy graph compression for approximate graph processing, storage, and analytics". ACM/IEEE SC. 2019

Clustering (Scaling)

Max memory overhead: 20%





[1] S. Beamer et al., „The GAP Benchmark Suite”. 2015

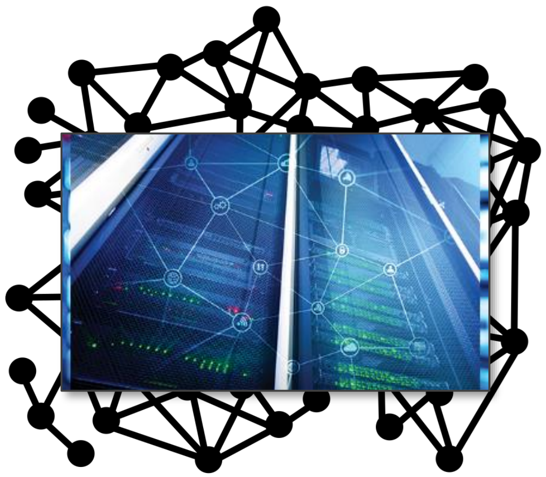
[2] R. Pagh et al., “Colorful triangle counting and a mapreduce implementation”. Information Processing Letters. 2012

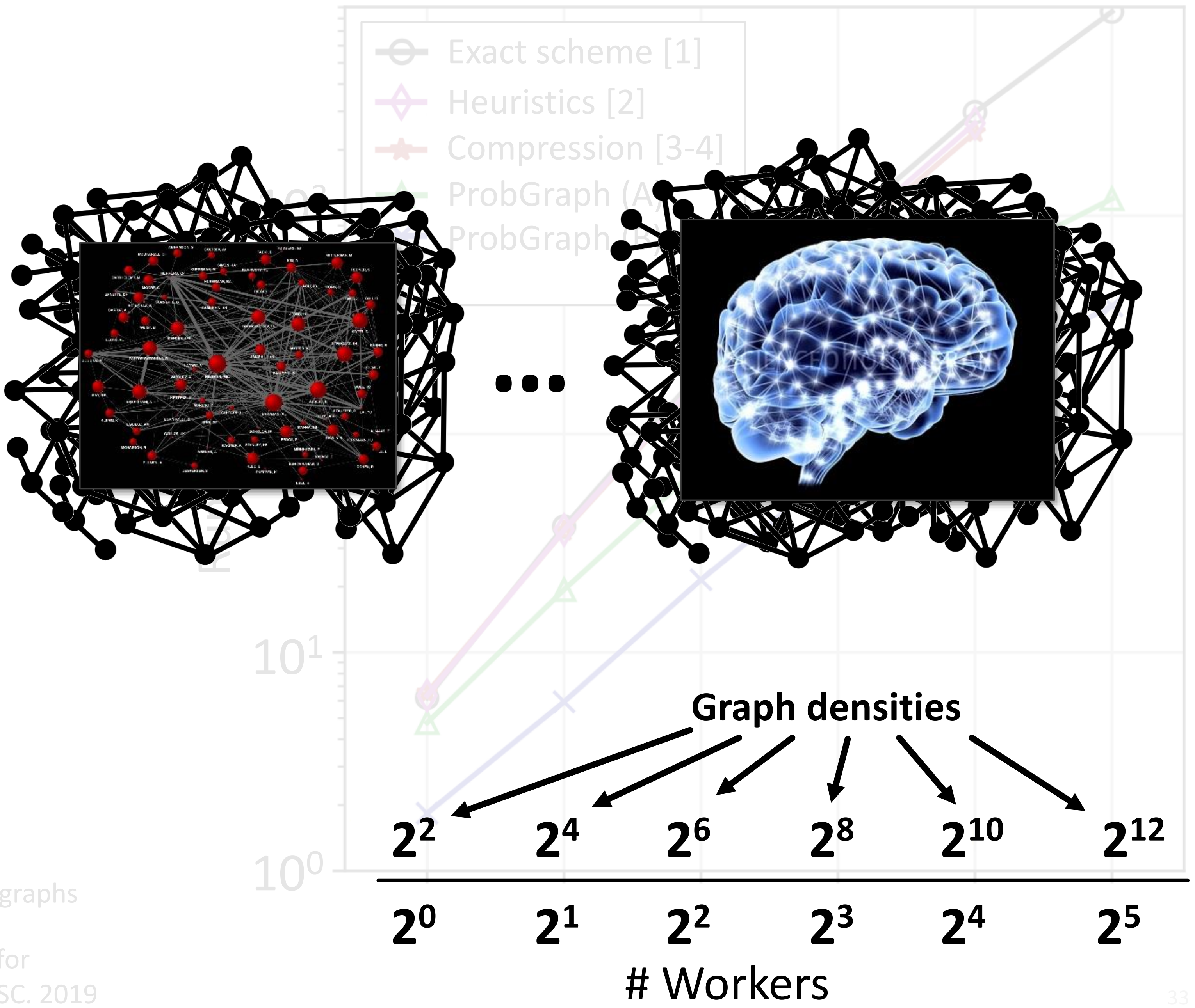
[3] C. E. Tsourakakis et al., “Doulion: counting triangles in massive graphs with a coin”. ACM KDD. 2009.

[4] M. Besta et al., “Slim graph: Practical lossy graph compression for approximate graph processing, storage, and analytics”. ACM/IEEE SC. 2019

Clustering (Scaling)

Max memory overhead: 20%

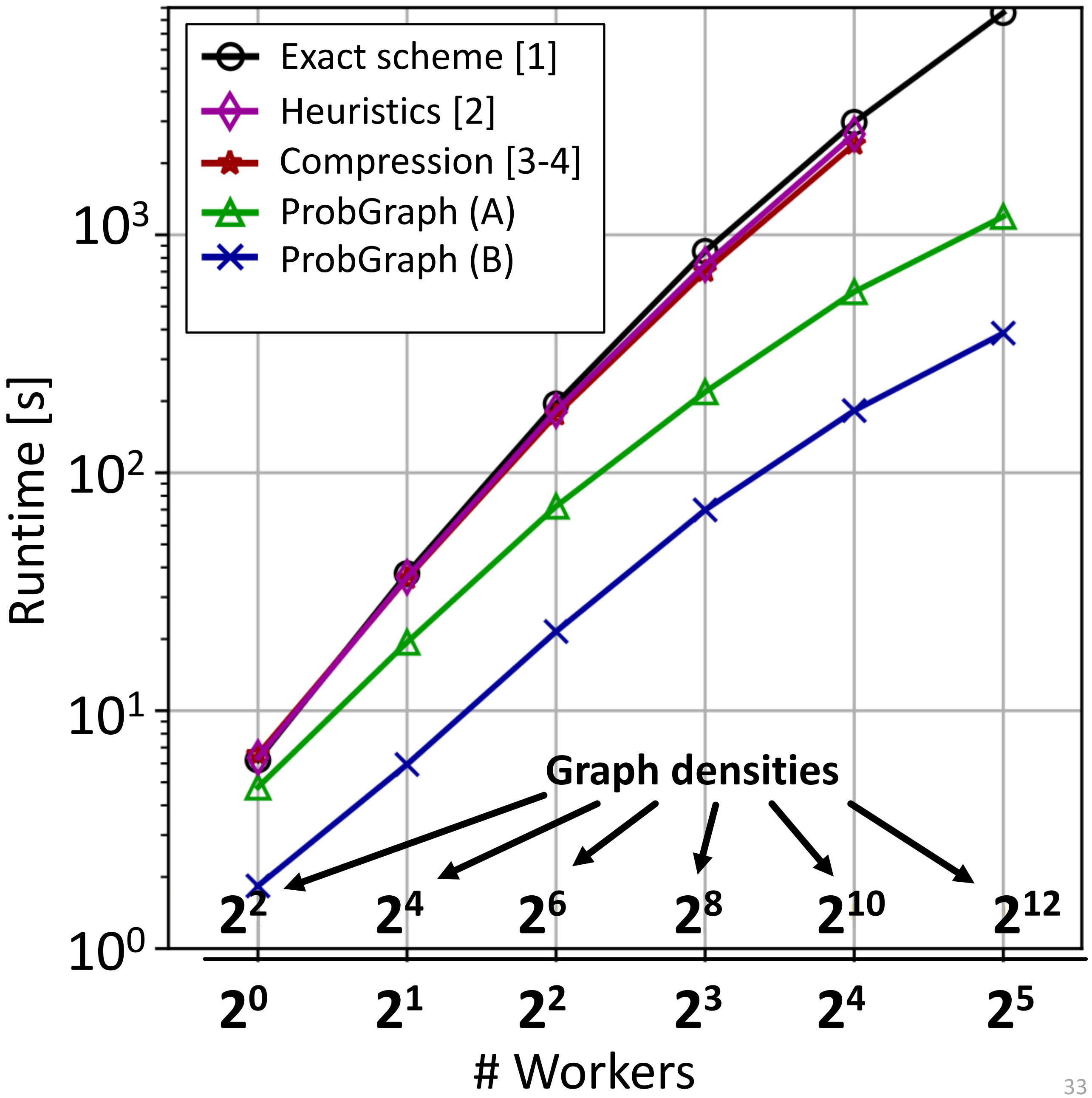




[1] S. Beamer et al., „The GAP Benchmark Suite". 2015
[2] R. Pagh et al., "Colorful triangle counting and a mapreduce implementation". Information Processing Letters. 2012
[3] C. E. Tsourakakis et al., "Doulion: counting triangles in massive graphs with a coin". ACM KDD. 2009.
[4] M. Besta et al., "Slim graph: Practical lossy graph compression for approximate graph processing, storage, and analytics". ACM/IEEE SC. 2019

Clustering (Scaling)

Max memory overhead: 20%



[1] S. Beamer et al., „The GAP Benchmark Suite”. 2015

[2] R. Pagh et al., “Colorful triangle counting and a mapreduce implementation”. Information Processing Letters. 2012

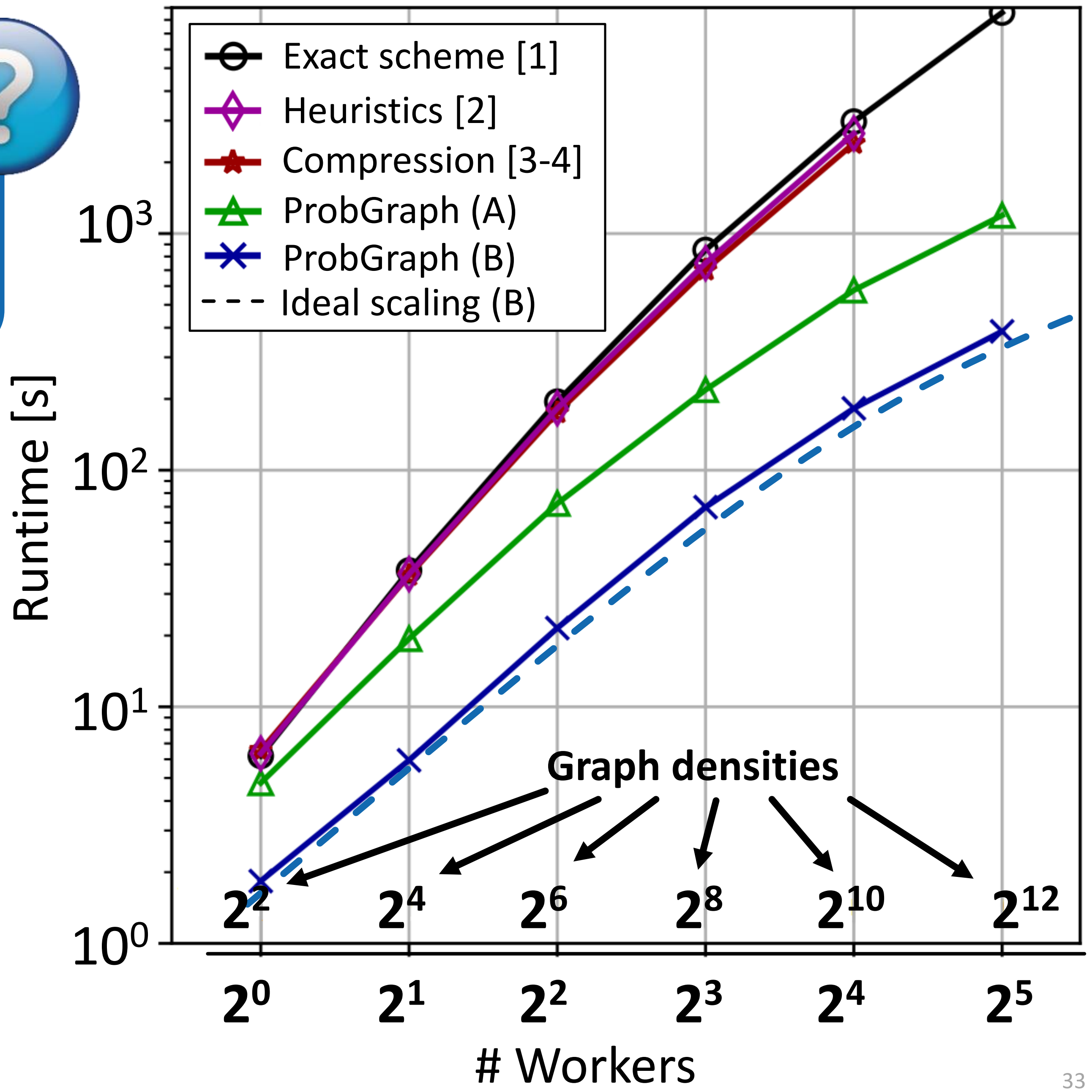
[3] C. E. Tsourakakis et al., “Doulion: counting triangles in massive graphs with a coin”. ACM KDD. 2009.

[4] M. Besta et al., “Slim graph: Practical lossy graph compression for approximate graph processing, storage, and analytics”. ACM/IEEE SC. 2019

Clustering (Scaling)

Max memory overhead: 20%

Why do we scale so well?



[1] S. Beamer et al., „The GAP Benchmark Suite". 2015

[2] R. Pagh et al., "Colorful triangle counting and a mapreduce implementation". Information Processing Letters. 2012

[3] C. E. Tsourakakis et al., "Doulion: counting triangles in massive graphs with a coin". ACM KDD. 2009.

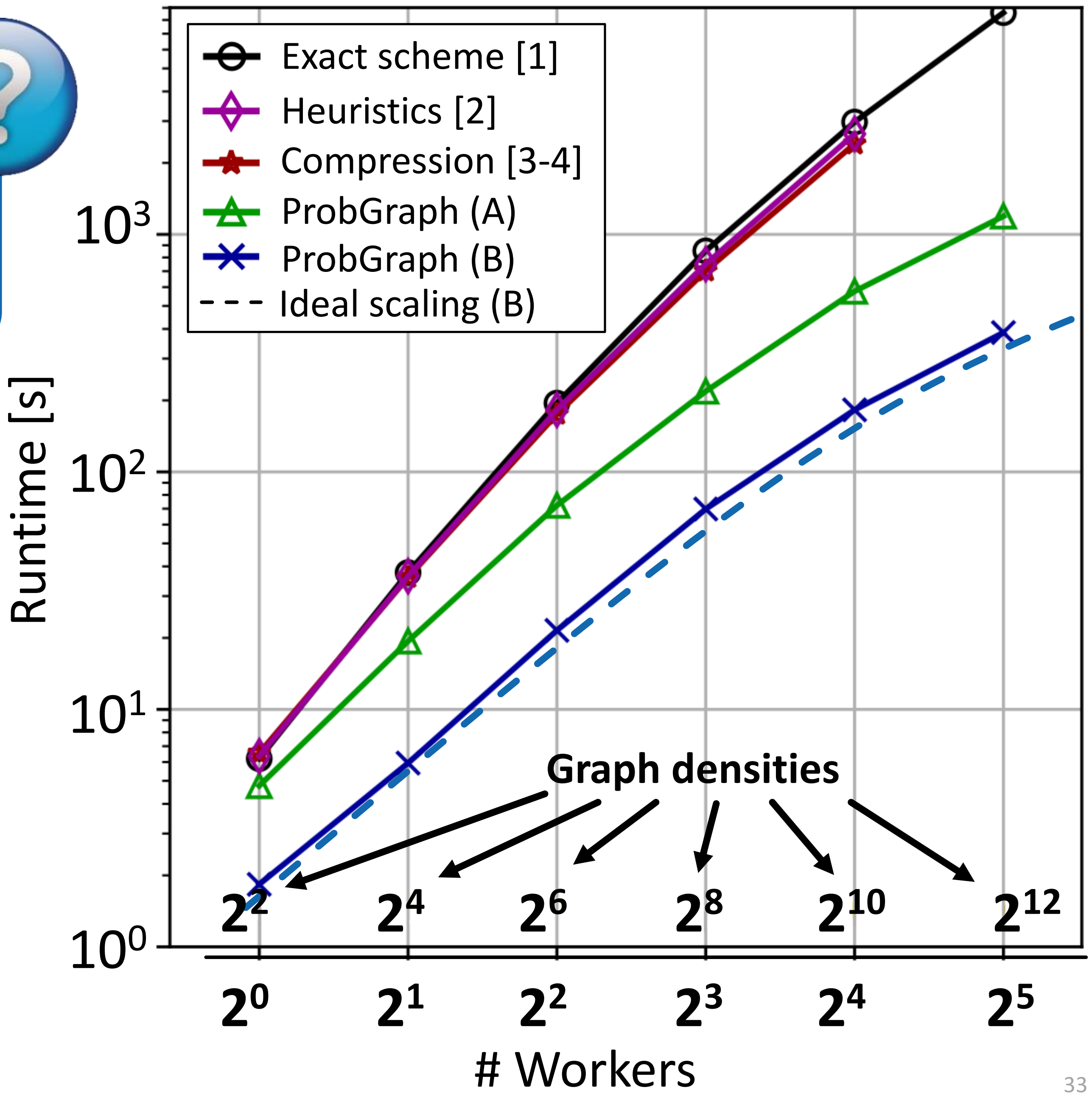
[4] M. Besta et al., "Slim graph: Practical lossy graph compression for approximate graph processing, storage, and analytics". ACM/IEEE SC. 2019

Clustering (Scaling)

Max memory overhead: 20%

Why do we scale so well?

Great load balancing properties

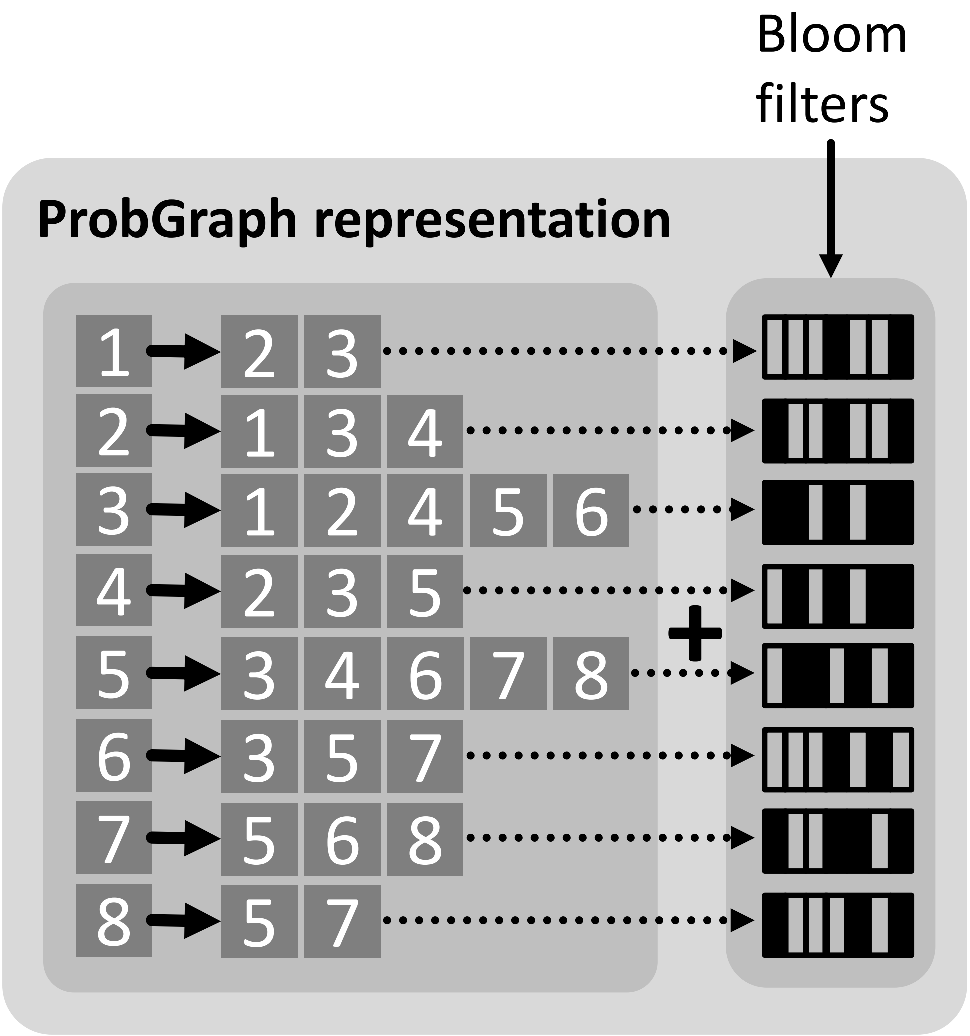


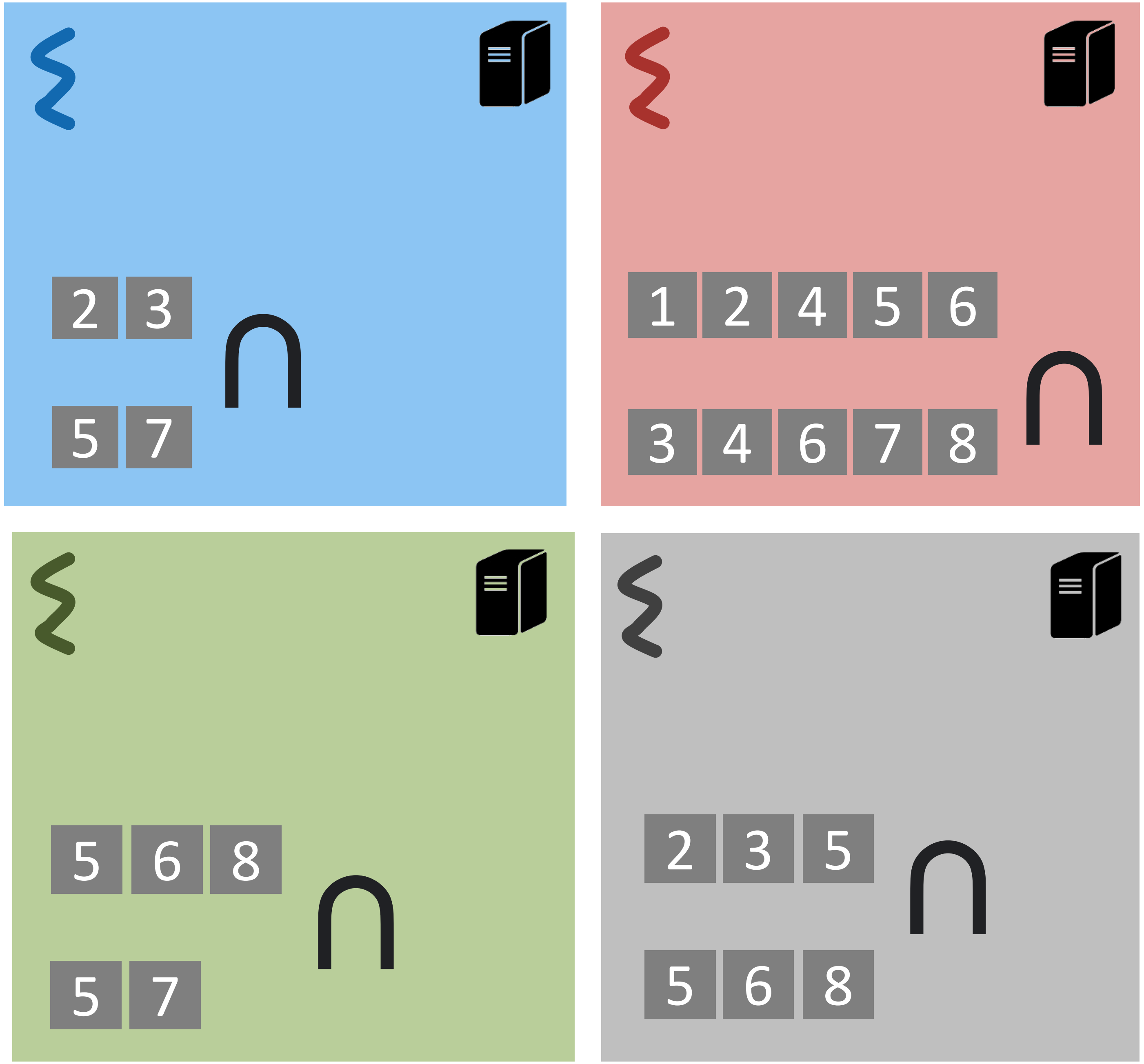
[1] S. Beamer et al., „The GAP Benchmark Suite". 2015

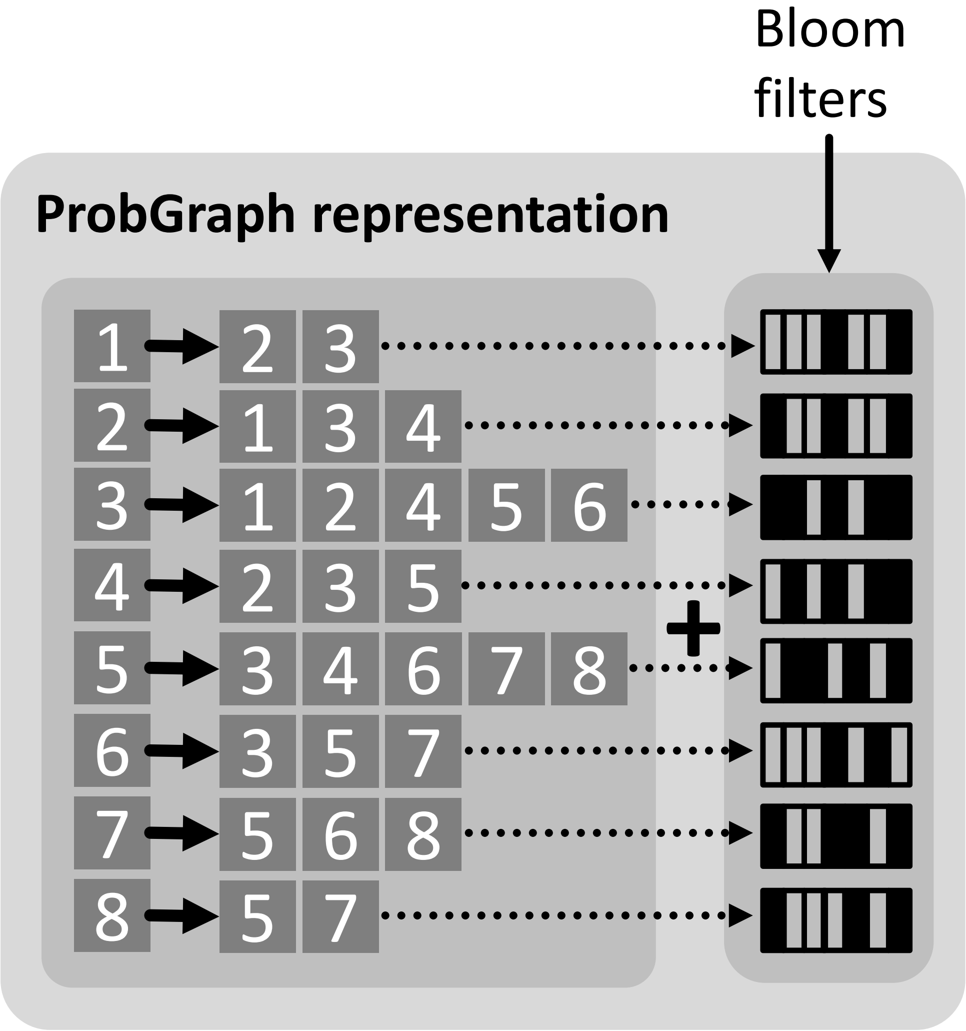
[2] R. Pagh et al., "Colorful triangle counting and a mapreduce implementation". Information Processing Letters. 2012

[3] C. E. Tsourakakis et al., "Doulion: counting triangles in massive graphs with a coin". ACM KDD. 2009.

[4] M. Besta et al., "Slim graph: Practical lossy graph compression for approximate graph processing, storage, and analytics". ACM/IEEE SC. 2019

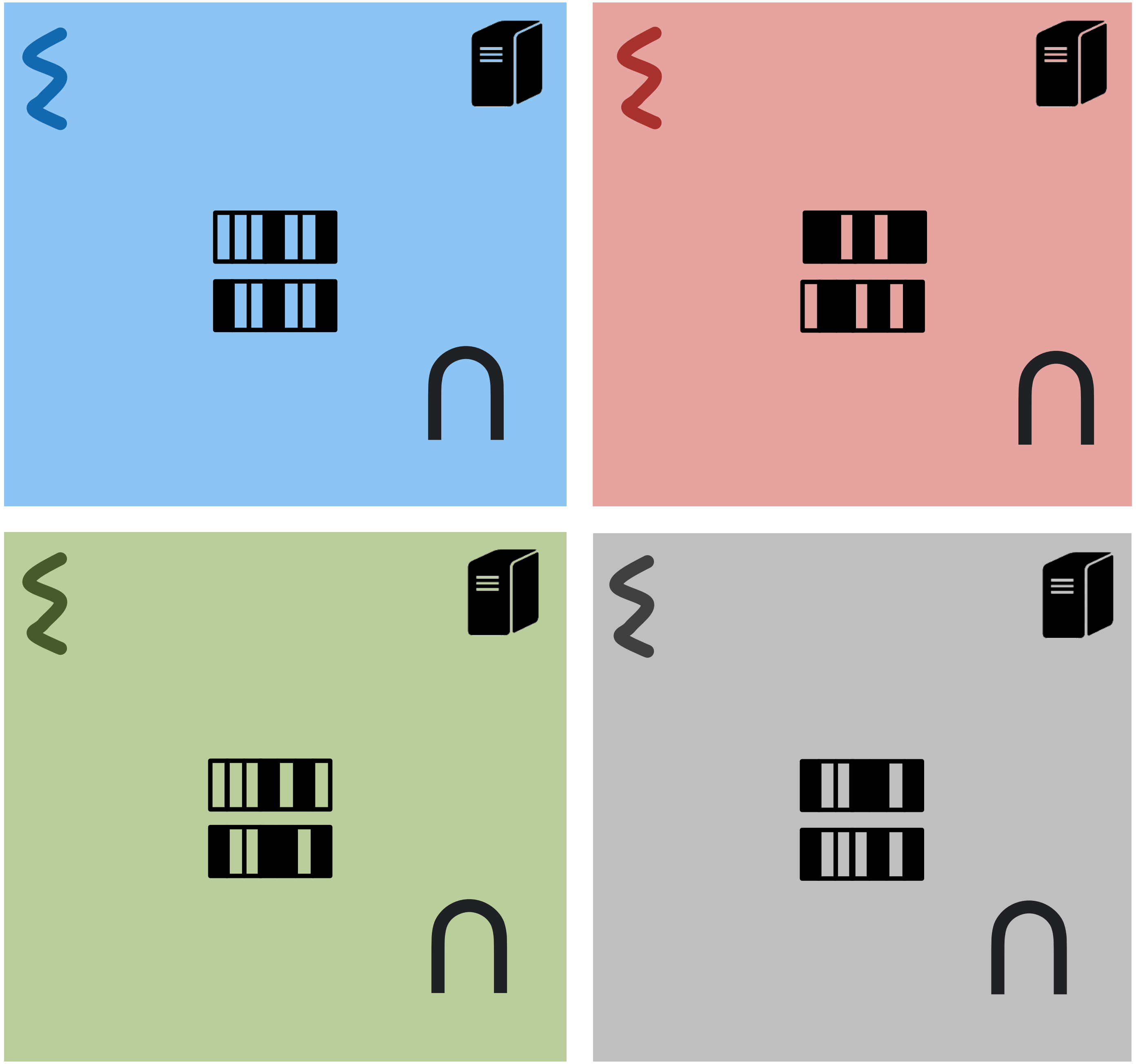

Bloom filters
ProbGraph representation
1 → 2 3
2 → 1 3 4
3 → 1 2 4 5 6
4 → 2 3 5
5 → 3 4 6 7 8
6 → 3 5 7
7 → 5 6 8
8 → 5 7
+

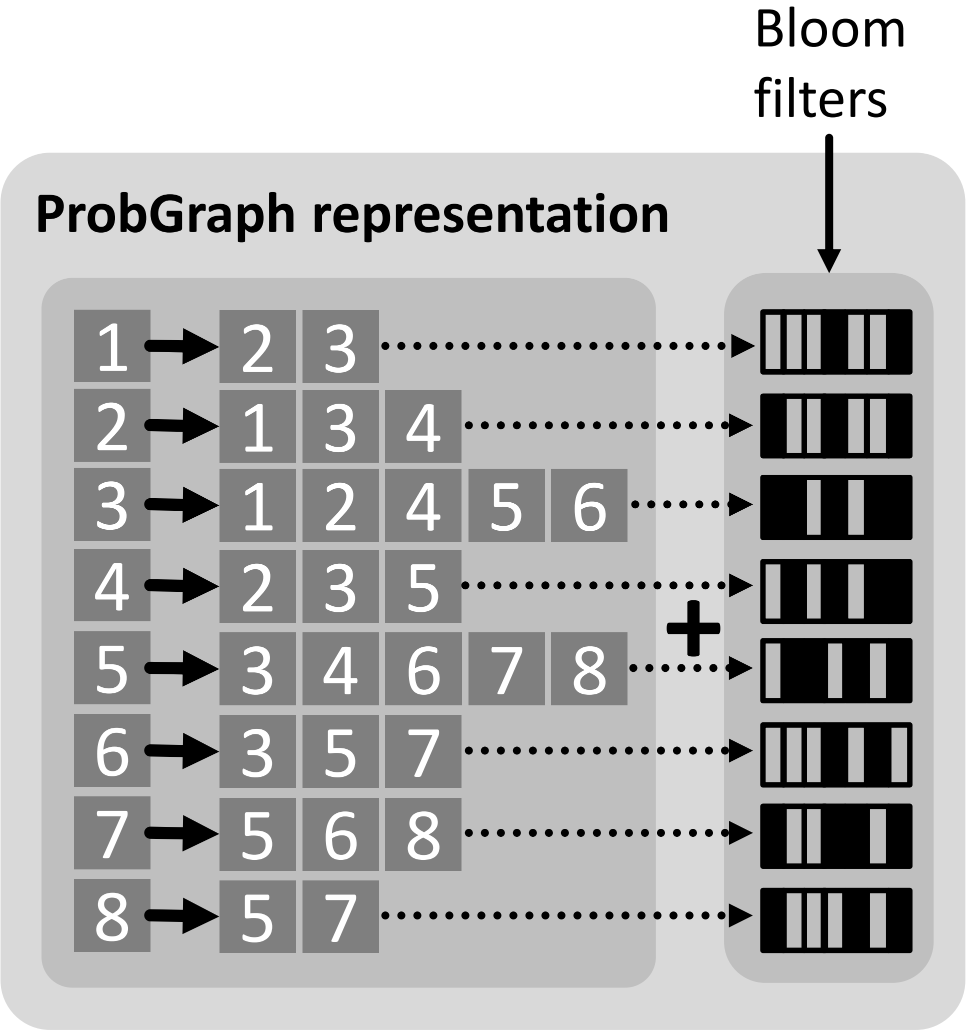



Bloom filters

ProbGraph representation

1 → 2 3
2 → 1 3 4
3 → 1 2 4 5 6
4 → 2 3 5
5 → 3 4 6 7 8
6 → 3 5 7
7 → 5 6 8
8 → 5 7



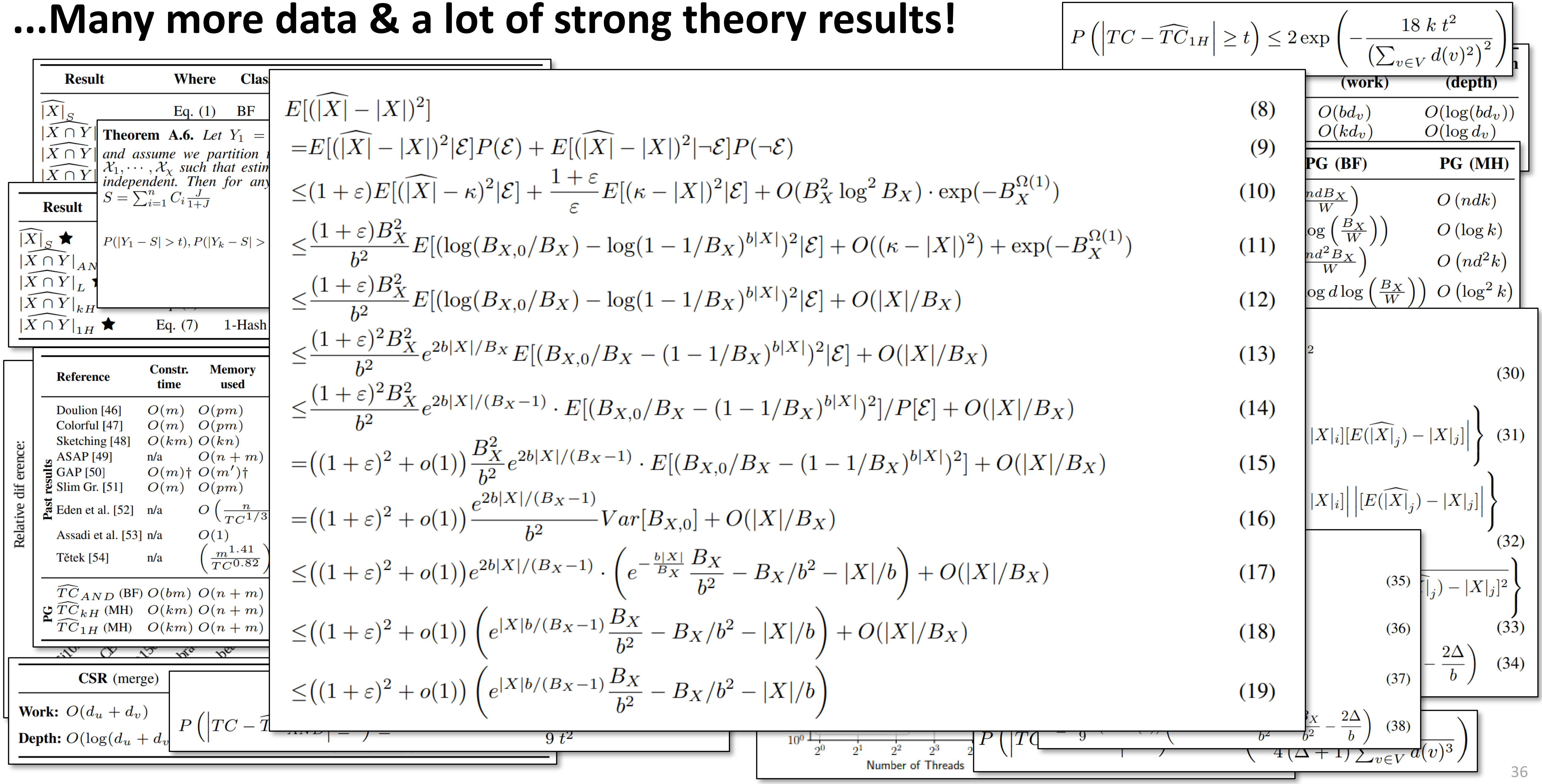


Bloom filters

ProbGraph representation

1 → 2 3
2 → 1 3 4
3 → 1 2 4 5 6
4 → 2 3 5
5 → 3 4 6 7 8
6 → 3 5 7
7 → 5 6 8
8 → 5 7

...Many more data & a lot of strong theory results!



$$P\left(\left|TC-\widehat{TC}_{1H}\right|\ge t\right)\le 2\exp\left(-\frac{18\,k\,t^2}{\left(\sum_{v\in V}d(v)^2\right)^2}\right)$$

$$E[(\widehat{|X|}-|X|)^2] \quad (8)$$

$$=E[(\widehat{|X|}-|X|)^2|\mathcal{E}]P(\mathcal{E})+E[(\widehat{|X|}-|X|)^2|\neg\mathcal{E}]P(\neg\mathcal{E}) \quad (9)$$

$$\le(1+\varepsilon)E[(\widehat{|X|}-\kappa)^2|\mathcal{E}]+\frac{1+\varepsilon}{\varepsilon}E[(\kappa-|X|)^2|\mathcal{E}]+O(B_X^2\log^2 B_X)\cdot\exp(-B_X^{\Omega(1)}) \quad (10)$$

$$\le\frac{(1+\varepsilon)B_X^2}{b^2}E[(\log(B_{X,0}/B_X)-\log(1-1/B_X)^{b|X|})^2|\mathcal{E}]+O((\kappa-|X|)^2)+\exp(-B_X^{\Omega(1)}) \quad (11)$$

$$\le\frac{(1+\varepsilon)B_X^2}{b^2}E[(\log(B_{X,0}/B_X)-\log(1-1/B_X)^{b|X|})^2|\mathcal{E}]+O(|X|/B_X) \quad (12)$$

$$\le\frac{(1+\varepsilon)^2B_X^2}{b^2}e^{2b|X|/B_X}E[(B_{X,0}/B_X-(1-1/B_X)^{b|X|})^2|\mathcal{E}]+O(|X|/B_X) \quad (13)$$

$$\le\frac{(1+\varepsilon)^2B_X^2}{b^2}e^{2b|X|/(B_X-1)}\cdot E[(B_{X,0}/B_X-(1-1/B_X)^{b|X|})^2]/P[\mathcal{E}]+O(|X|/B_X) \quad (14)$$

$$=((1+\varepsilon)^2+o(1))\frac{B_X^2}{b^2}e^{2b|X|/(B_X-1)}\cdot E[(B_{X,0}/B_X-(1-1/B_X)^{b|X|})^2]+O(|X|/B_X) \quad (15)$$

$$=((1+\varepsilon)^2+o(1))\frac{e^{2b|X|/(B_X-1)}}{b^2}Var[B_{X,0}]+O(|X|/B_X) \quad (16)$$

$$\le((1+\varepsilon)^2+o(1))e^{2b|X|/(B_X-1)}\cdot\left(e^{-\frac{b|X|}{B_X}}\frac{B_X}{b^2}-B_X/b^2-|X|/b\right)+O(|X|/B_X) \quad (17)$$

$$\le((1+\varepsilon)^2+o(1))\left(e^{|X|b/(B_X-1)}\frac{B_X}{b^2}-B_X/b^2-|X|/b\right)+O(|X|/B_X) \quad (18)$$

$$\le((1+\varepsilon)^2+o(1))\left(e^{|X|b/(B_X-1)}\frac{B_X}{b^2}-B_X/b^2-|X|/b\right) \quad (19)$$

Result	Where	Clas
$\widehat{\lvert X\rvert}_S$	Eq. (1)	BF
$\widehat{\lvert X\cap Y\rvert}$		
$\widehat{\lvert X\cap Y\rvert}$		
$\widehat{\lvert X\cap Y\rvert}$		

Theorem A.6. *Let* Y_1 = *and assume we partition* $\mathcal{X}_1,\cdots,\mathcal{X}_X$ *such that estin independent. Then for any* $S=\sum_{i=1}^{n}C_i\frac{J}{1+J}$

$P(|Y_1-S|>t), P(|Y_k-S|>$

Result	Where	
$\widehat{\lvert X\rvert}_S$ ★		
$\widehat{\lvert X\cap Y\rvert}_{AN}$		
$\widehat{\lvert X\cap Y\rvert}_{L}$		
$\widehat{\lvert X\cap Y\rvert}_{kH}$		
$\widehat{\lvert X\cap Y\rvert}_{1H}$ ★	Eq. (7)	1-Hash

	Reference	Constr. time	Memory used
Past results	Doulion [46]	$O(m)$	$O(pm)$
	Colorful [47]	$O(m)$	$O(pm)$
	Sketching [48]	$O(km)$	$O(kn)$
	ASAP [49]	n/a	$O(n+m)$
	GAP [50]	$O(m)$†	$O(m')$†
	Slim Gr. [51]	$O(m)$	$O(pm)$
	Eden et al. [52]	n/a	$O\left(\frac{n}{TC^{1/3}}\right.$
	Assadi et al. [53]	n/a	$O(1)$
	Tětek [54]	n/a	$\left(\frac{m^{1.41}}{TC^{0.82}}\right)$
PG	$\widehat{TC}_{AND}$ (BF)	$O(bm)$	$O(n+m)$
	$\widehat{TC}_{kH}$ (MH)	$O(km)$	$O(n+m)$
	$\widehat{TC}_{1H}$ (MH)	$O(km)$	$O(n+m)$

Relative dif erence:

CSR (merge)

Work: $O(d_u+d_v)$

Depth: $O(\log(d_u+d_v$

$P\left(\left|TC-\widehat{T}\right.\right.$ $9\,t^2$

(work)	(depth)
$O(bd_v)$	$O(\log(bd_v))$
$O(kd_v)$	$O(\log d_v)$

PG (BF)	PG (MH)
$\frac{ndB_X}{W})$	$O(ndk)$
$\log\left(\frac{B_X}{W}\right))$	$O(\log k)$
$\frac{nd^2B_X}{W})$	$O(nd^2k)$
$\log d\log\left(\frac{B_X}{W}\right))$	$O(\log^2 k)$

(30)

$|X|_i][E(\widehat{|X|}_j)-|X|_j]\}$ (31)

$|X|_i|\,\left|[E(\widehat{|X|}_j)-|X|_j]\right|\}$ (32)

$|_j)-|X|_j]^2\}$ (33)

$-\frac{2\Delta}{b})$ (34)

(35)

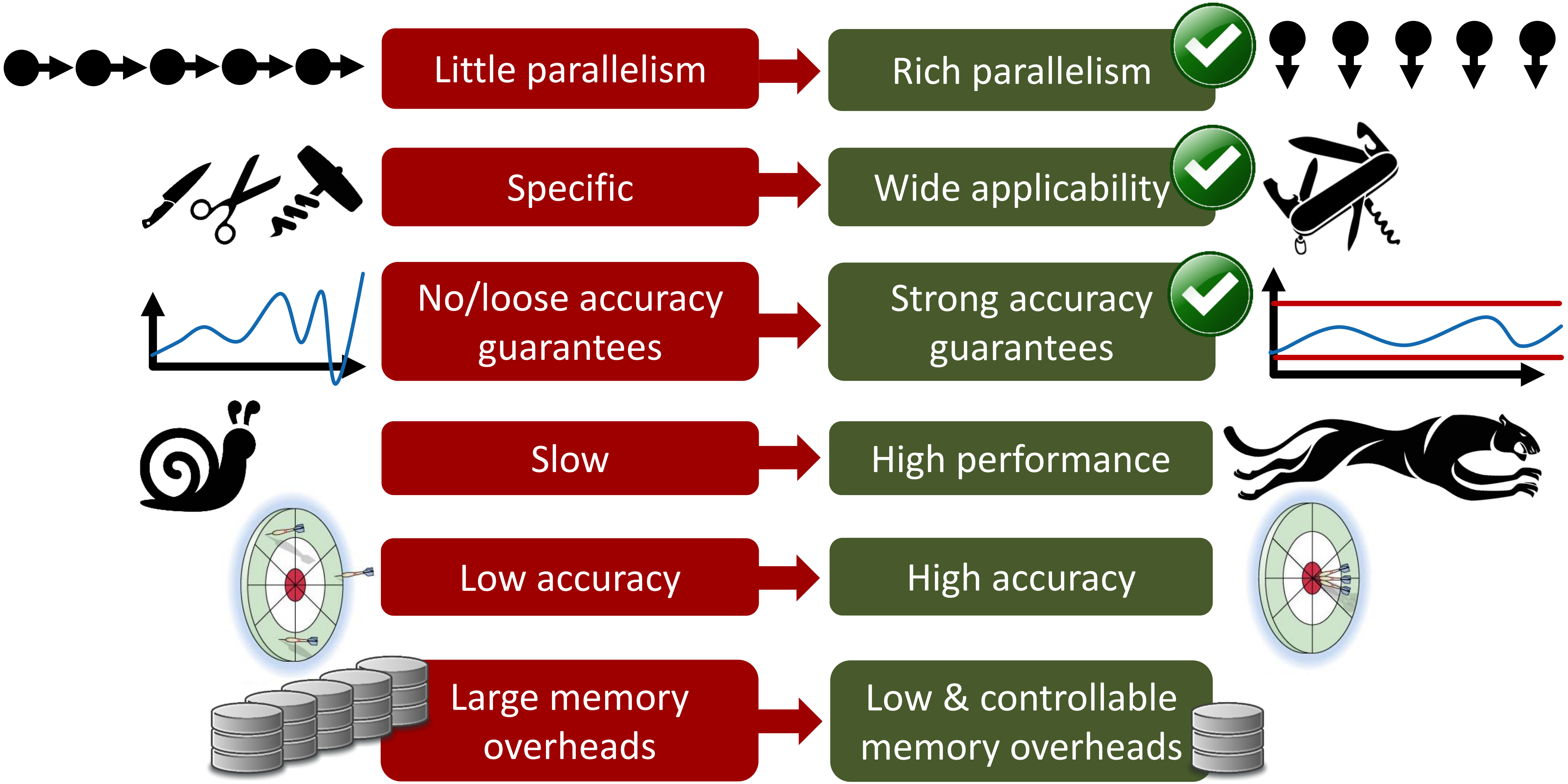
(36)

(37)

$\frac{B_X}{b^2}-\frac{2\Delta}{b})$ (38)

$P\left(\left|TC\right.\right.$ $4(\Delta+1)\sum_{v\in V}d(v)^3$

Approximate Graph Processing: Our Objectives



Little parallelism → Rich parallelism ✓

Specific → Wide applicability ✓

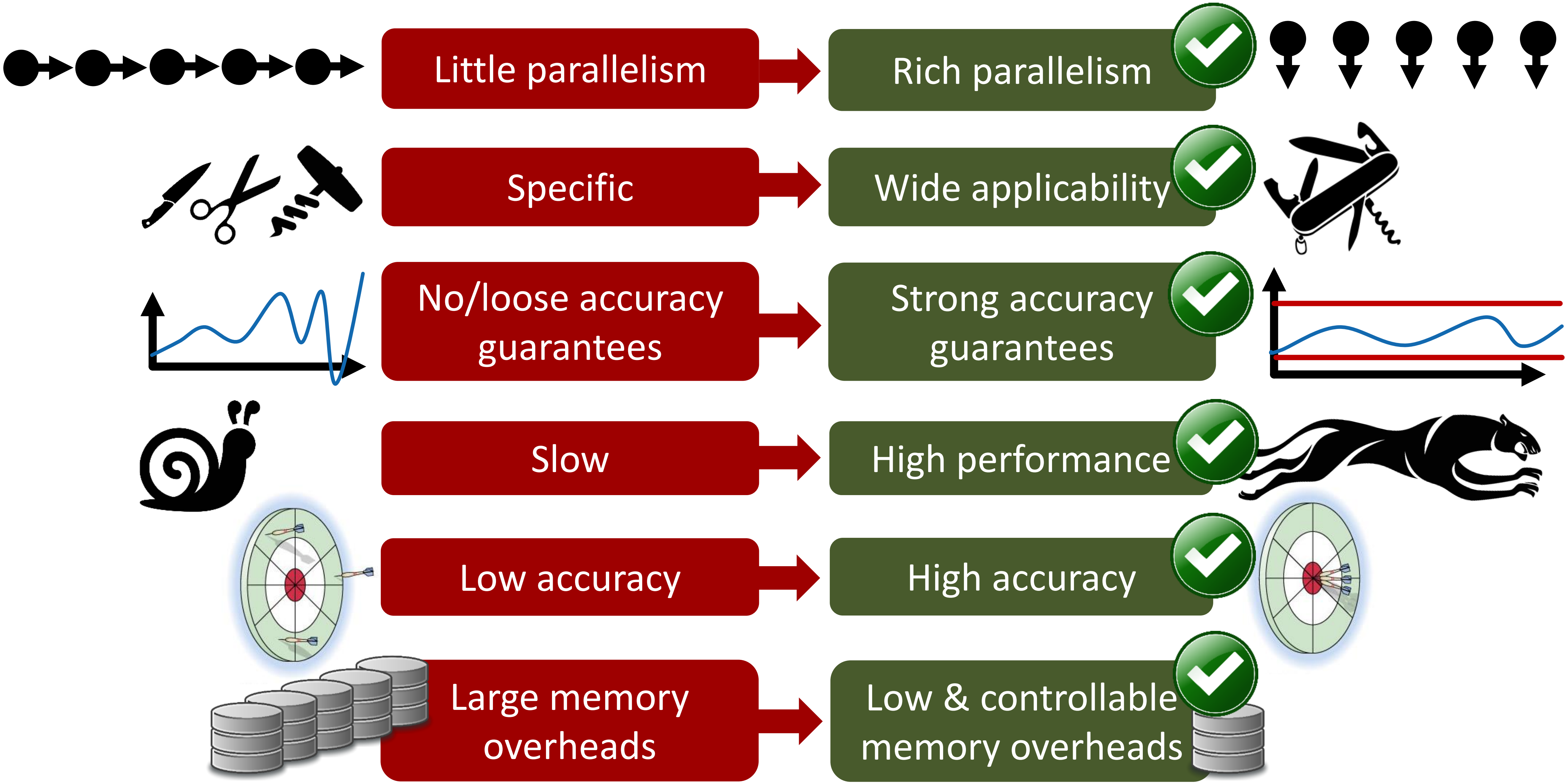
No/loose accuracy guarantees → Strong accuracy guarantees ✓

Slow → High performance

Low accuracy → High accuracy

Large memory overheads → Low & controllable memory overheads

Approximate Graph Processing: Our Objectives



Little parallelism → Rich parallelism ✓

Specific → Wide applicability ✓

No/loose accuracy guarantees → Strong accuracy guarantees ✓

Slow → High performance ✓

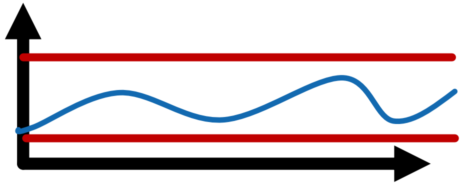
Low accuracy → High accuracy ✓

Large memory overheads → Low & controllable memory overheads ✓

Conclusion: ProbGraph Enables Approximate Graph Mining with...

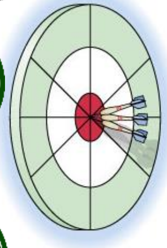
- Rich parallelism ✓
- Wide applicability ✓

- Strong accuracy guarantees ✓



- High performance ✓

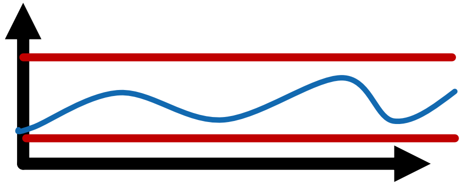
- High accuracy ✓



- Low & controllable memory overheads ✓

Conclusion: ProbGraph Enables Approximate Graph Mining with...

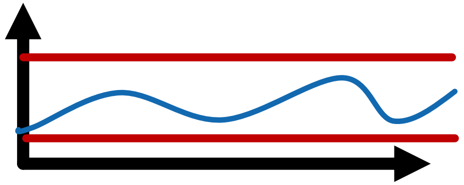
- Rich parallelism ✓
- Wide applicability ✓
- Strong accuracy guarantees ✓



- High performance ✓
- High accuracy ✓
- Low & controllable memory overheads ✓

Conclusion: ProbGraph Enables Approximate Graph Mining with...

- Rich parallelism ✓
- Wide applicability ✓
- Strong accuracy guarantees ✓

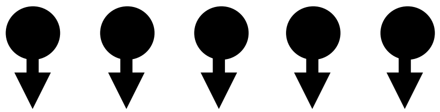


- High performance ✓
- High accuracy ✓
- Low & controllable memory overheads ✓

Thank you

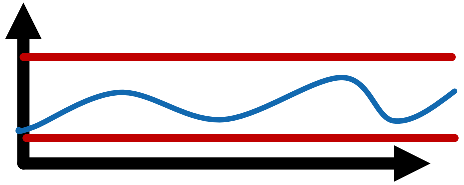
Conclusion: ProbGraph Enables Approximate Graph Mining with...

- Rich parallelism ✓



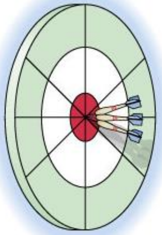
- Wide applicability ✓

- Strong accuracy guarantees ✓



- High performance ✓

- High accuracy ✓



- Low & controllable memory overheads ✓

Thank you

Want to know more?

youtube.com/@spcl

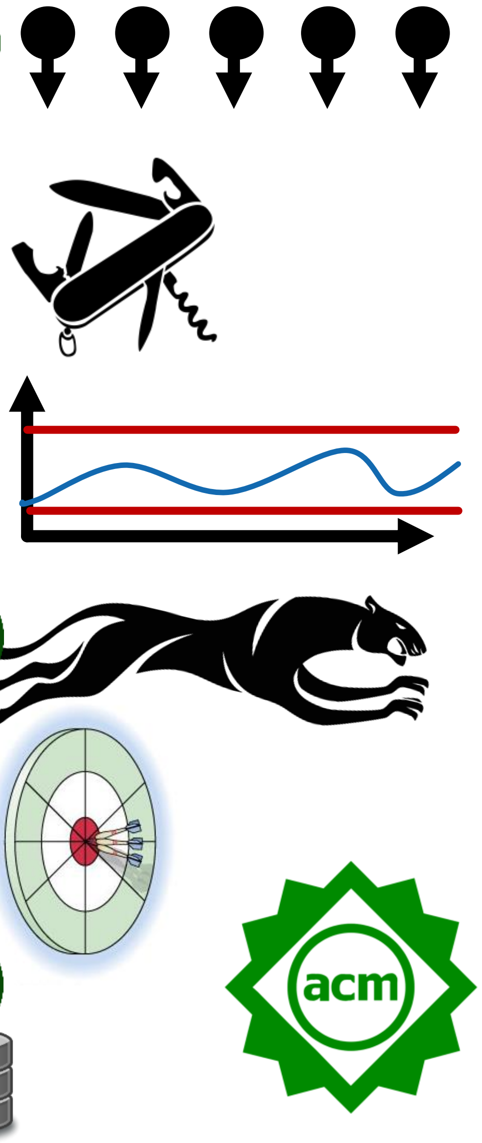
twitter.com/spcl_eth

spcl.inf.ethz.ch

github.com/spcl

Conclusion: ProbGraph Enables Approximate Graph Mining with...

- Rich parallelism ✓
- Wide applicability ✓
- Strong accuracy guarantees ✓
- High performance ✓
- High accuracy ✓
- Low & controllable memory overheads ✓



Backup slides

Want to know more?

- youtube.com/@spcl
- twitter.com/spcl_eth
- spcl.inf.ethz.ch
- github.com/spcl