SFU

PEREGRINE

A Pattern-Aware Graph Mining System

Kasra Jamshidi Rakesh Mahadasa Keval Vora

Simon Fraser University

https://github.com/pdclab/peregrine

Why should you pay attention?

Peregrine executes 700x faster

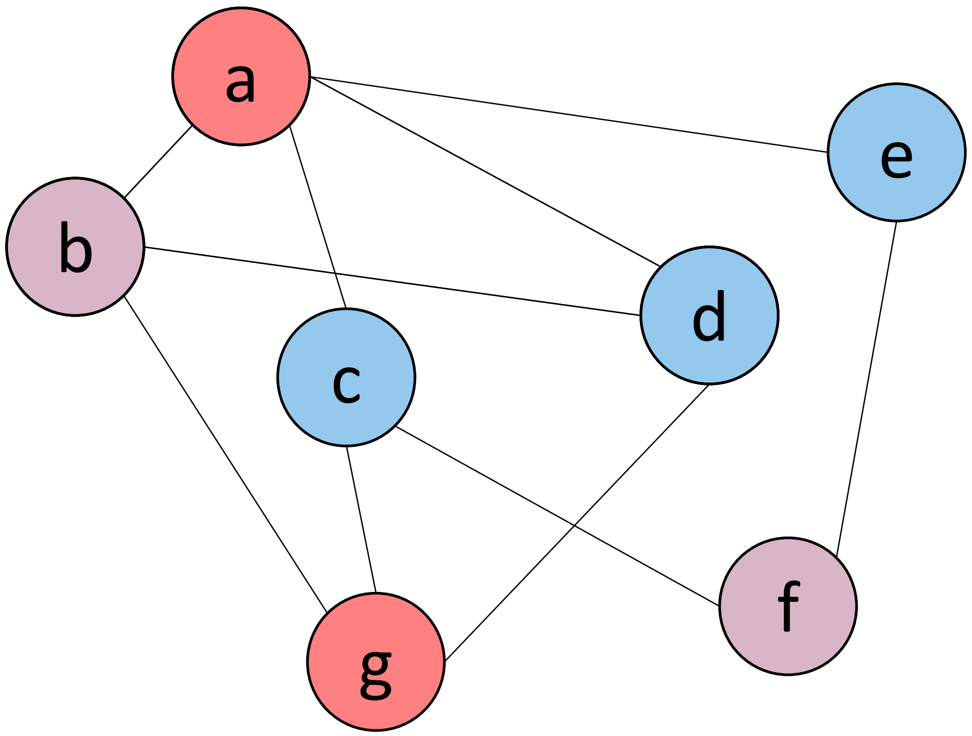
Peregrine consumes 100x less memory

Peregrine scales to 100x larger datasets

On 8x fewer machines

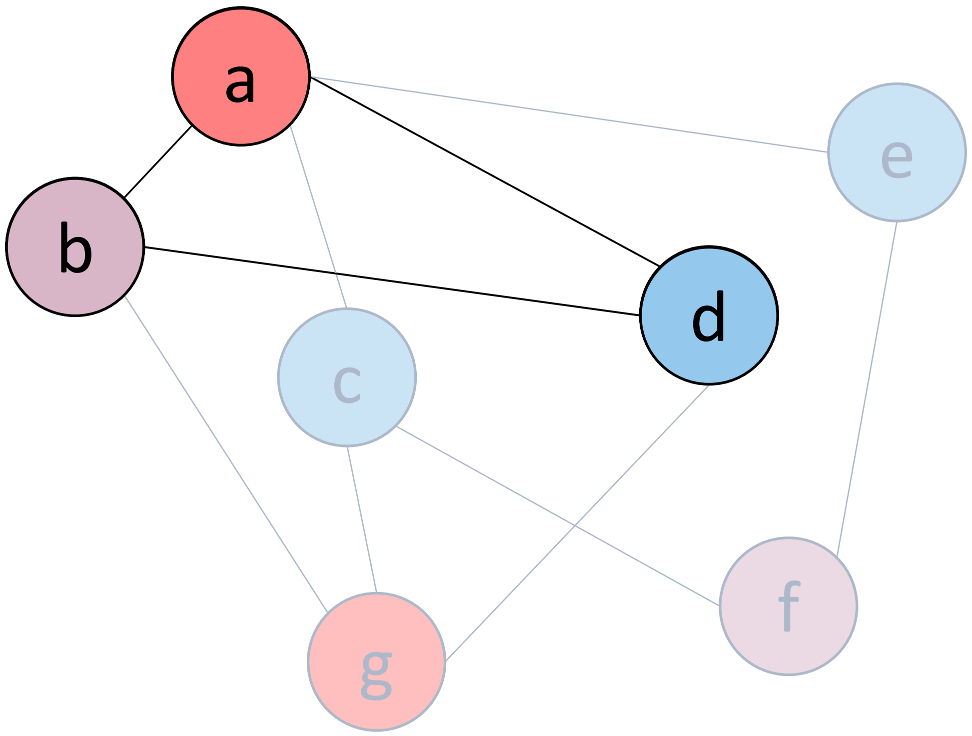
With a more expressive API

Graph Mining



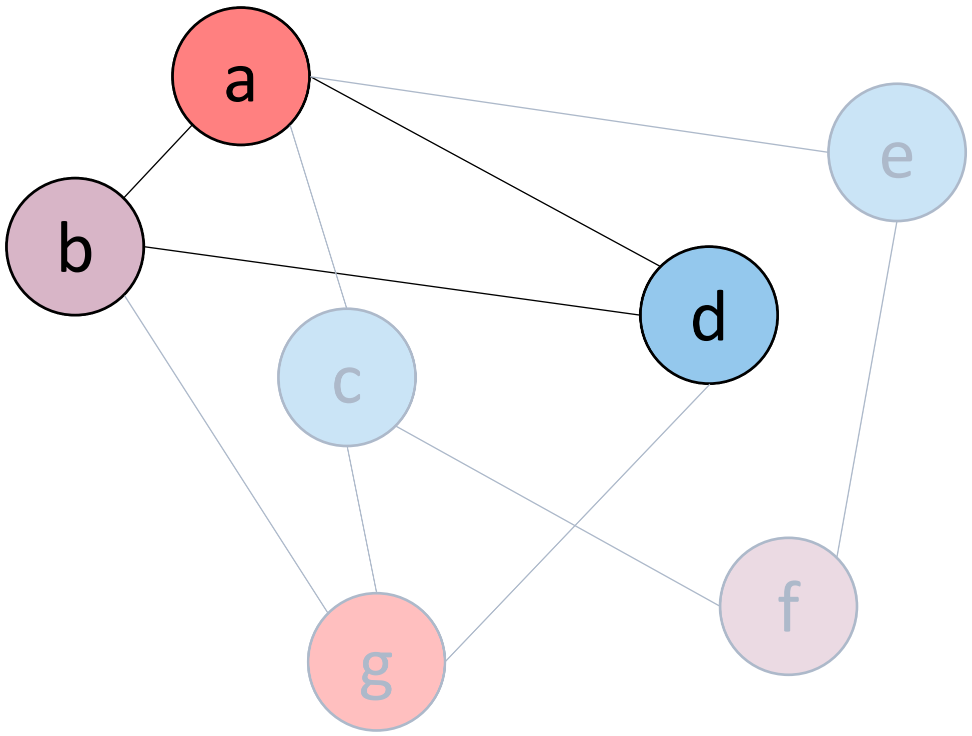
Data Graph

Graph Mining

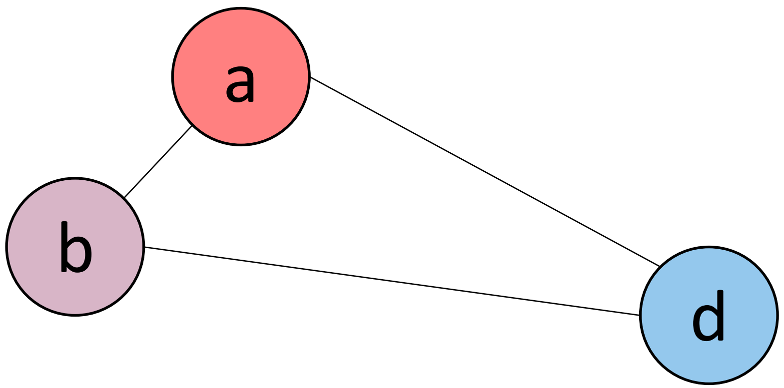


Data Graph

Graph Mining

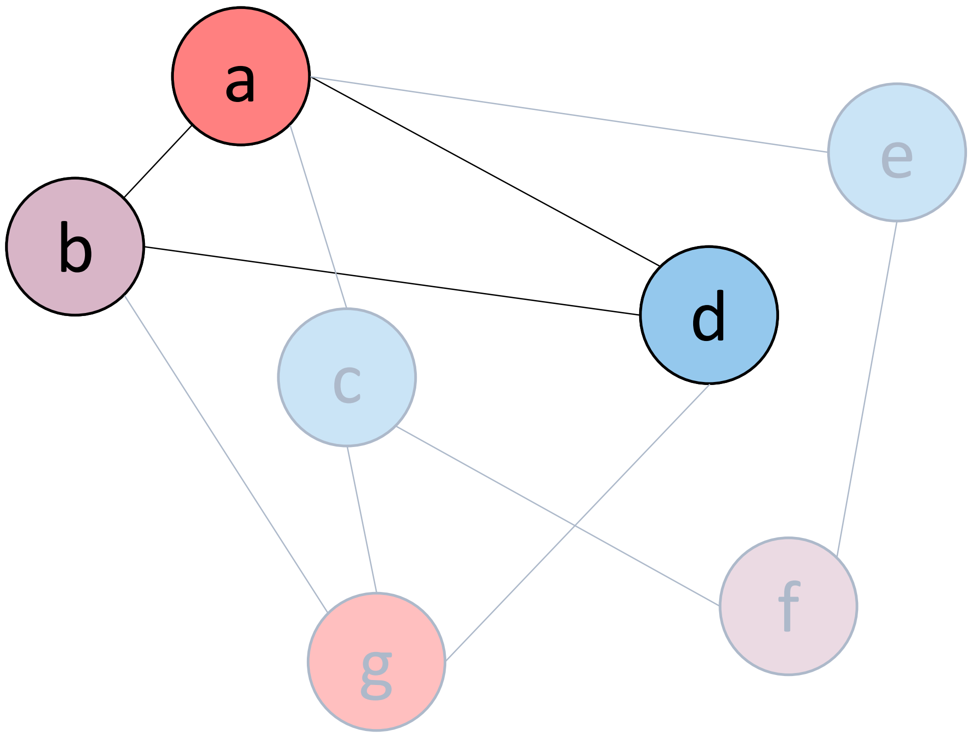


Data Graph

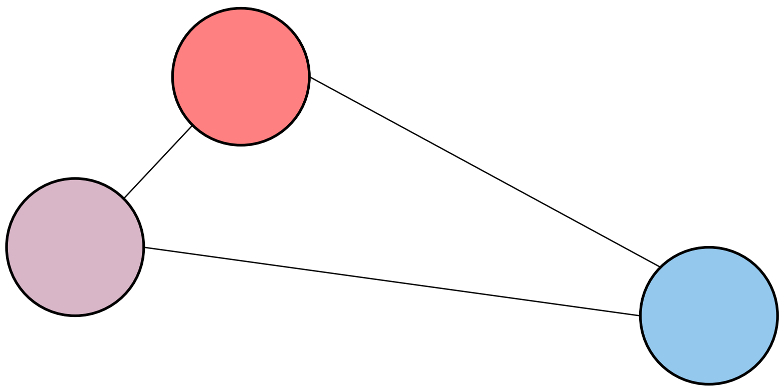


Subgraph

Graph Mining

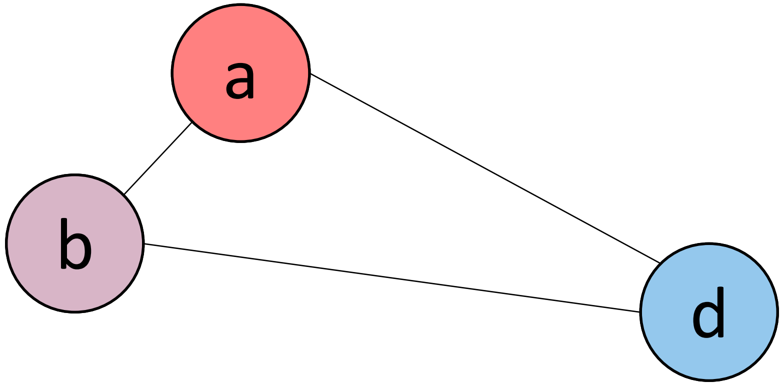


Data Graph



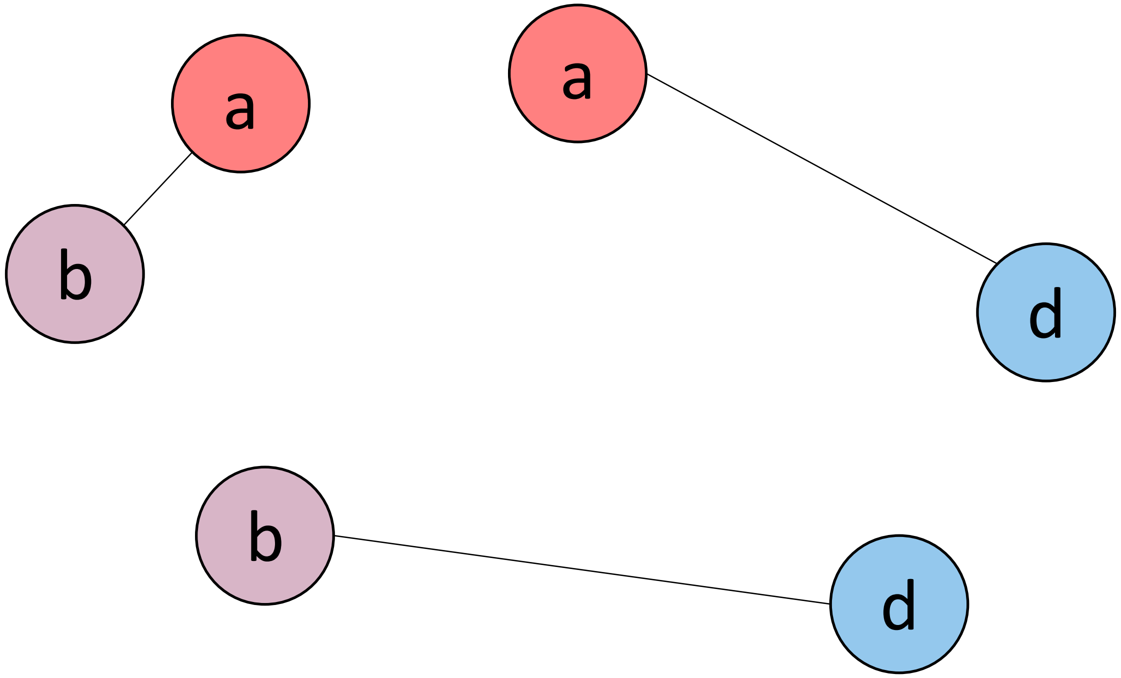
Pattern

Graph Mining



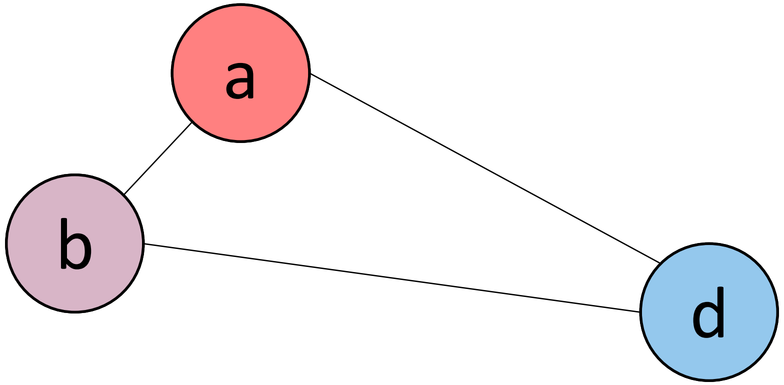
Edge-Induced

Graph Mining



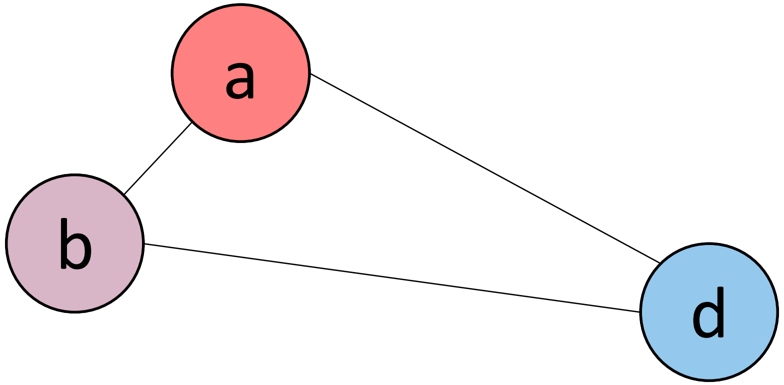
Edge-Induced

Graph Mining



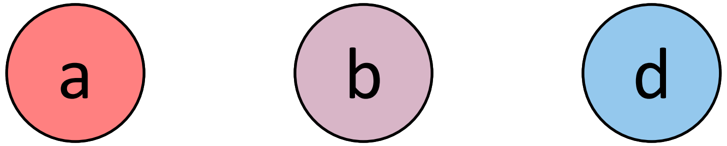
Edge-Induced

Graph Mining



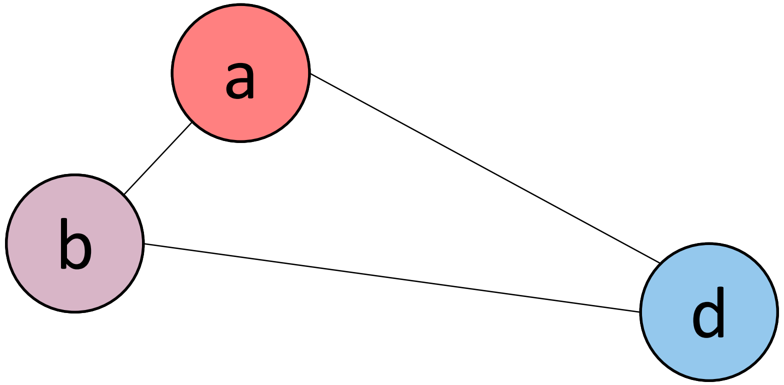
Vertex-Induced

Graph Mining



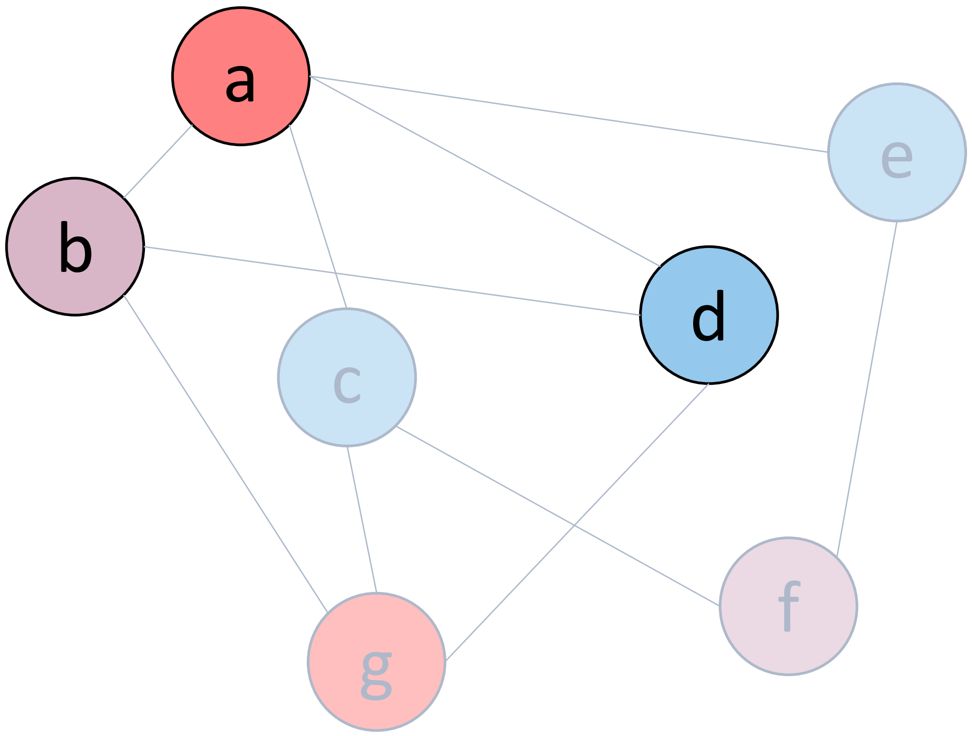
Vertex-Induced

Graph Mining



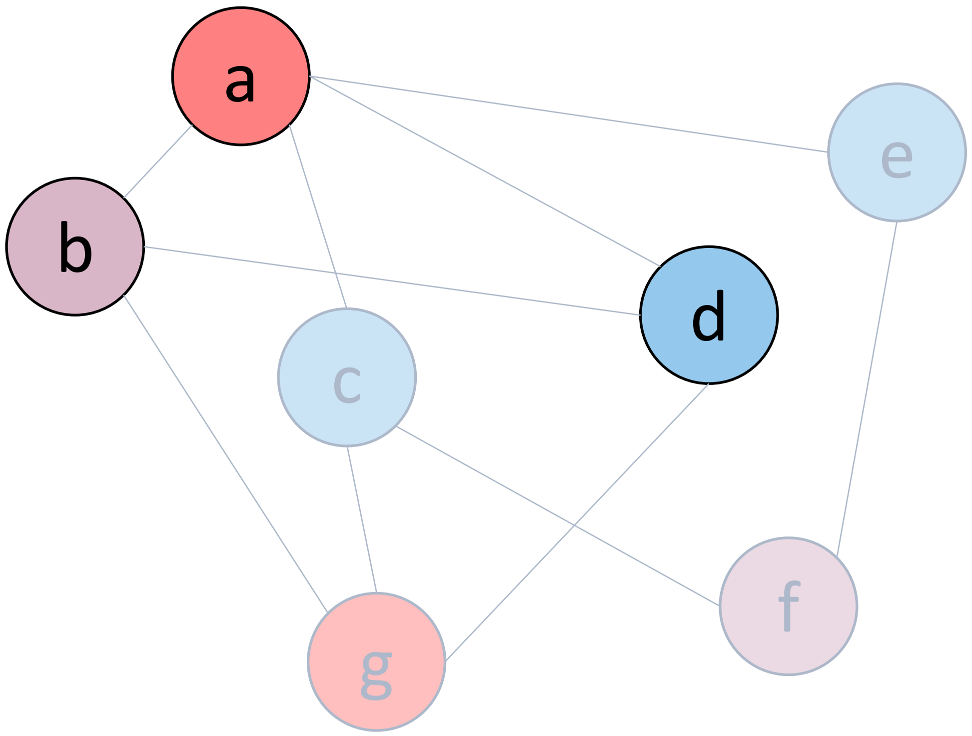
Vertex-Induced

Graph Mining

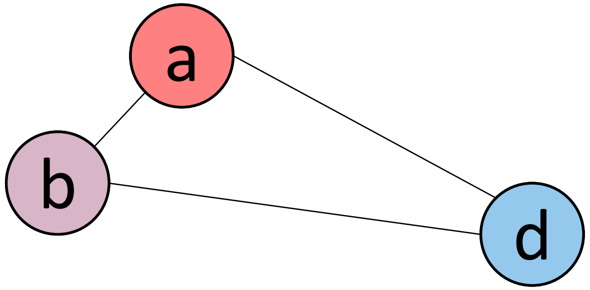


Data Graph

Graph Mining

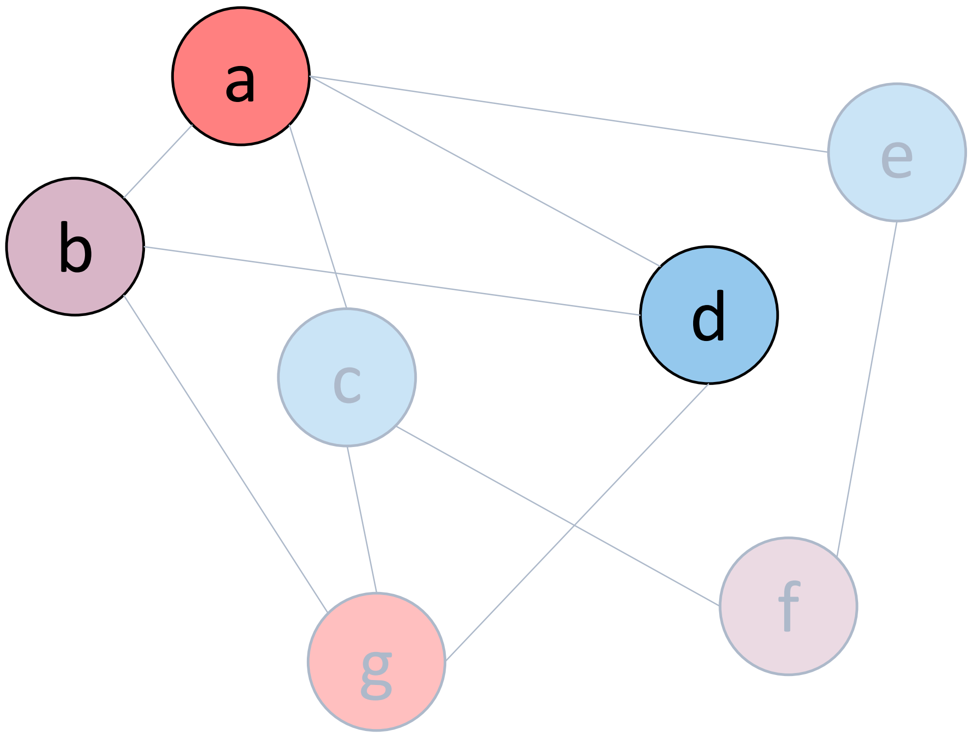


Data Graph

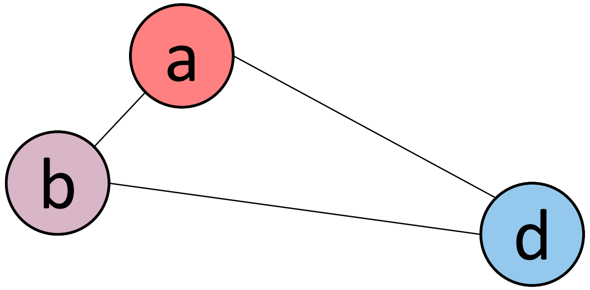


Vertex-Induced

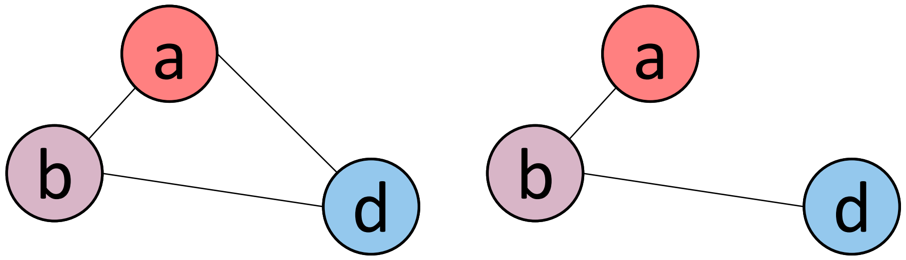
Graph Mining



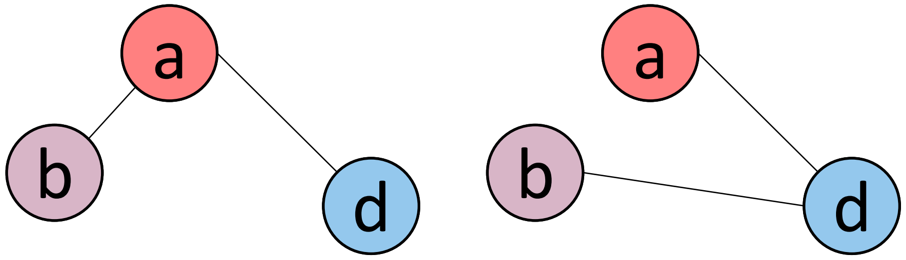
Data Graph



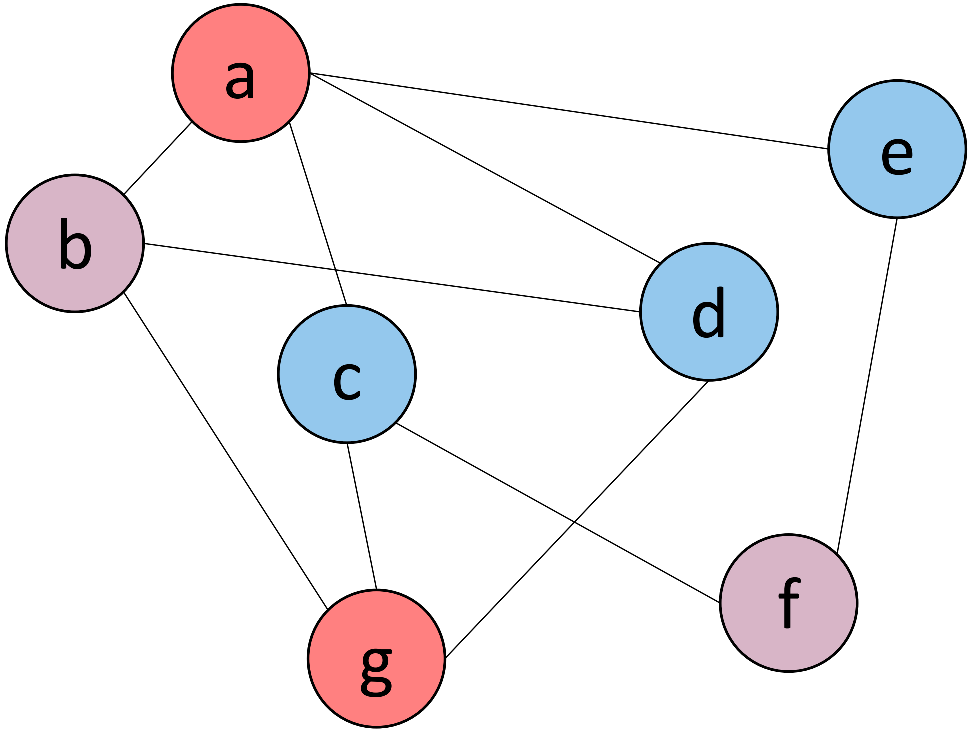
Vertex-Induced



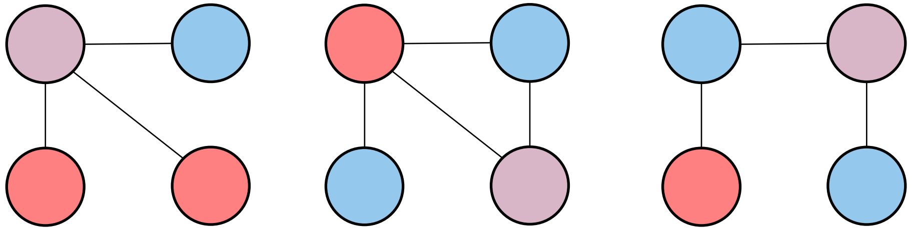
Edge-Induced



Graph Mining

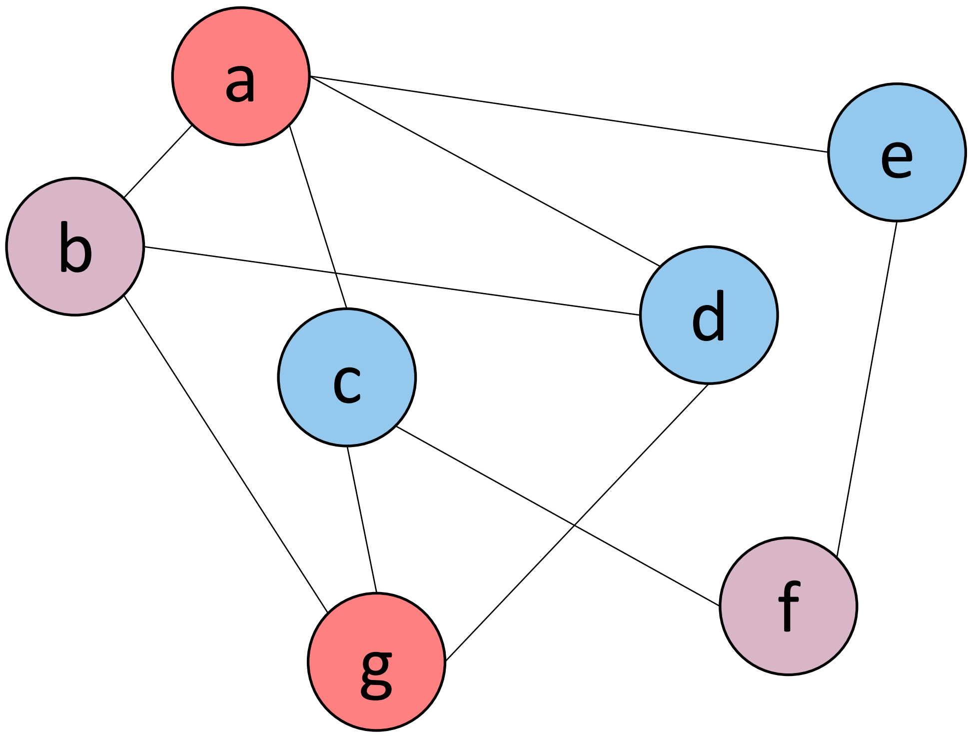


Data Graph

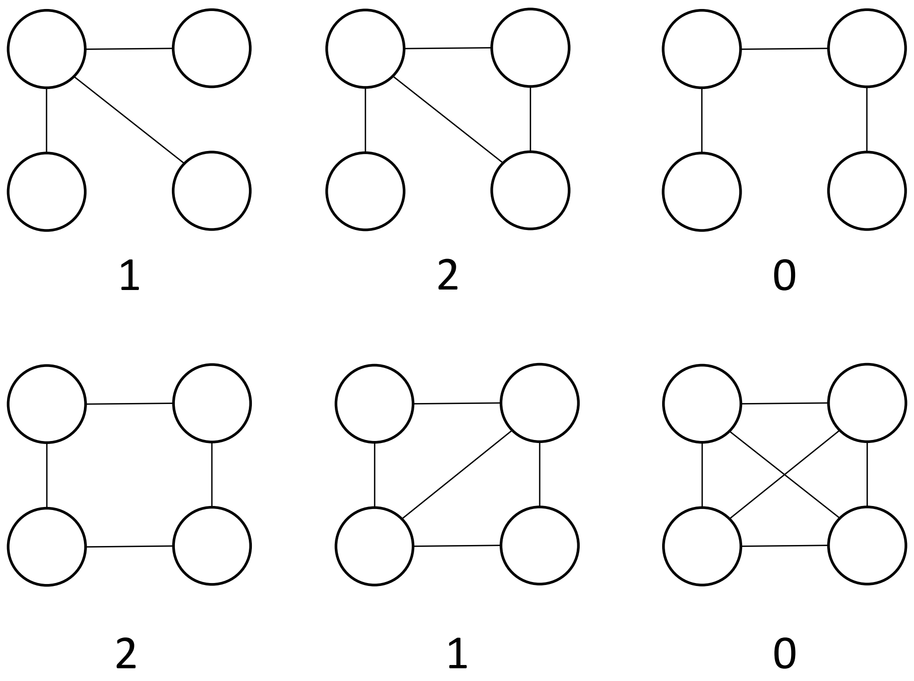


Frequent Patterns
(Edge-Induced)

Graph Mining



Data Graph

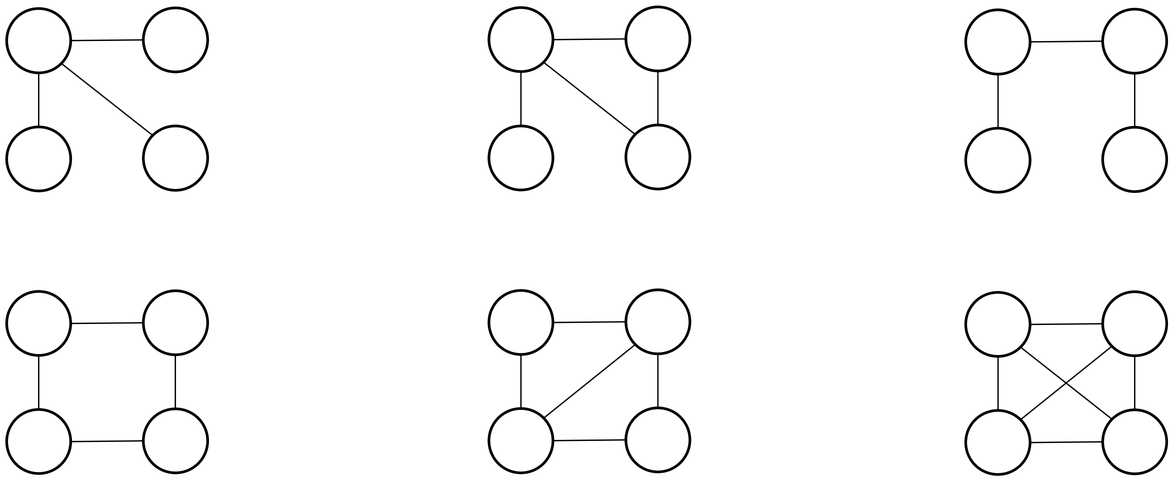


Unlabeled Pattern Distribution
(Vertex-Induced)

Scalability Challenge

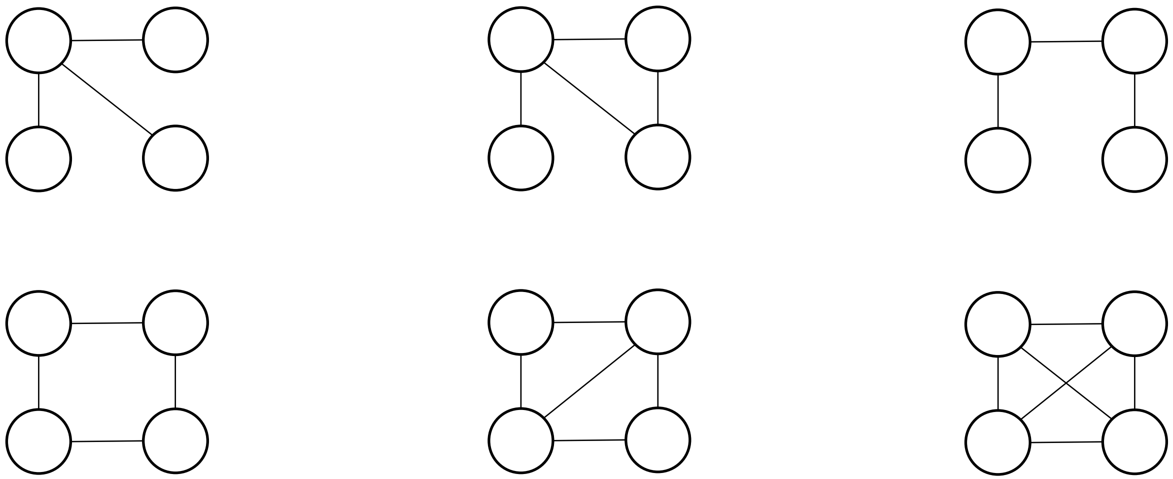
Scalability Challenge

- 4-motif counting on Orkut graph (|V| = 13M, |E| = 117M)



Scalability Challenge

- 4-motif counting on Orkut graph (|V| = 13M, |E| = 117M)

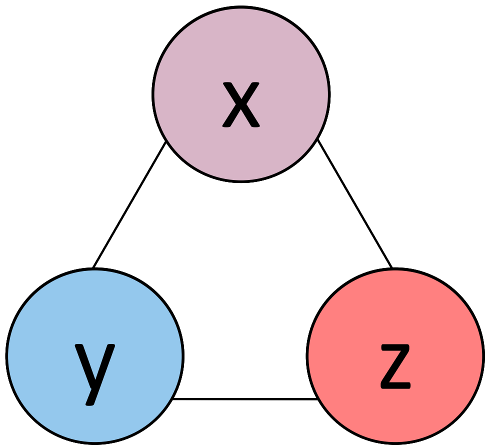


123,503,340,341,270 subgraphs

System Requirements

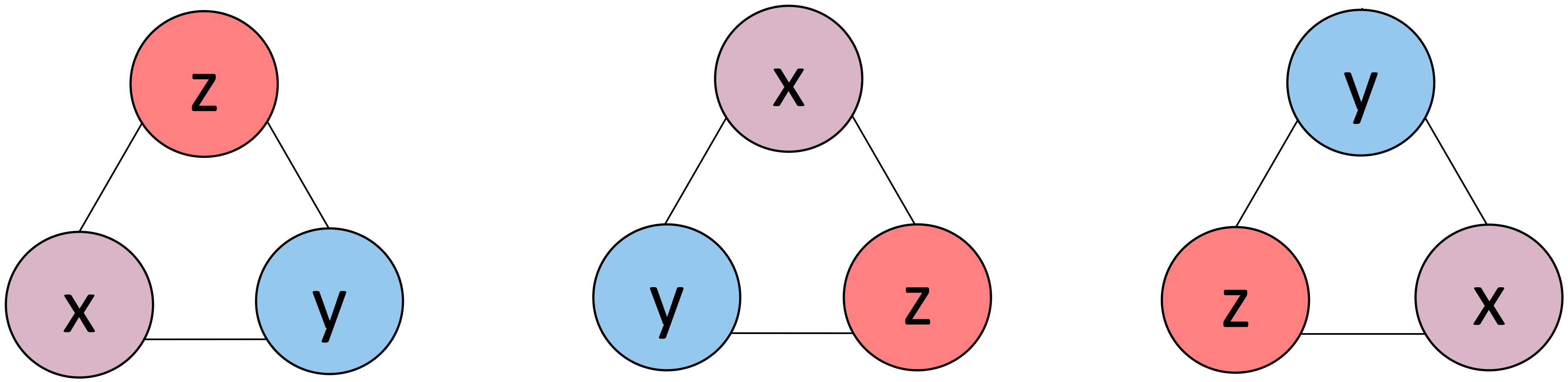
System Requirements

Uniqueness



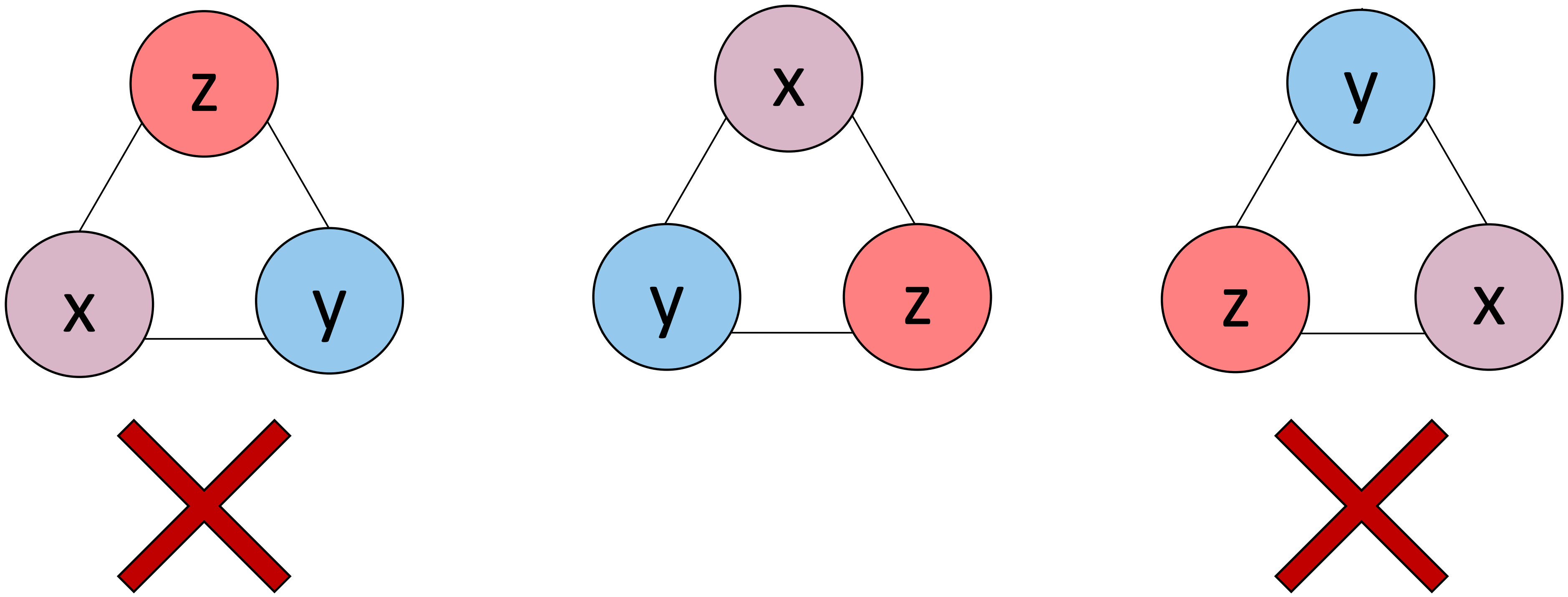
System Requirements

Uniqueness



System Requirements

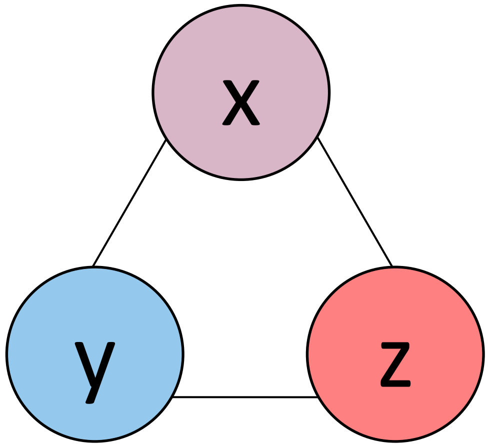
Uniqueness



System Requirements

Uniqueness

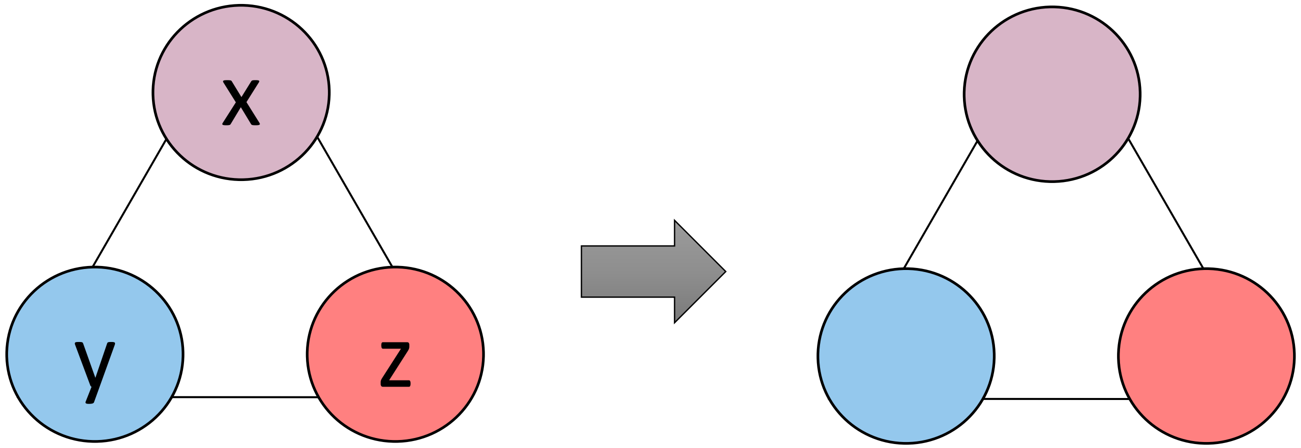
Structure



System Requirements

Uniqueness

Structure

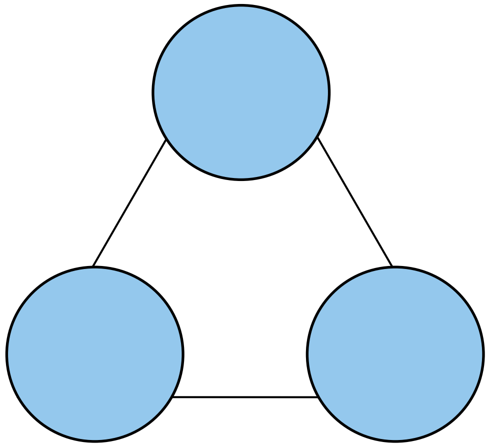


System Requirements

Uniqueness

Structure

Interestingness

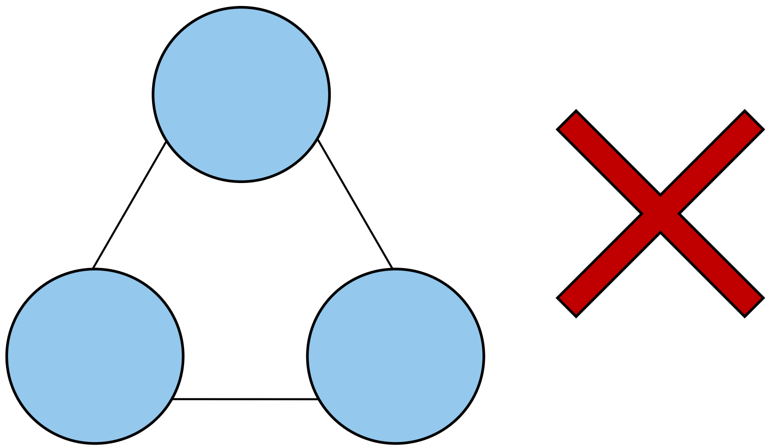


System Requirements

Uniqueness

Structure

Interestingness

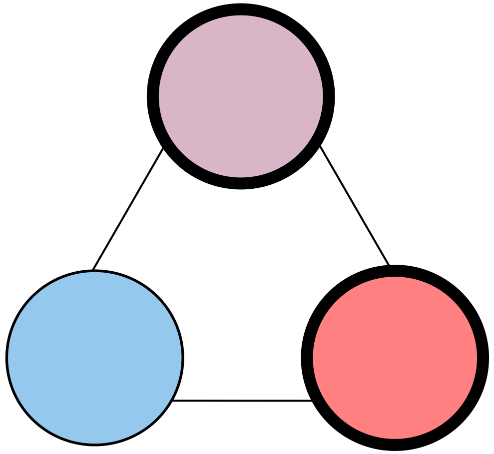


System Requirements

Uniqueness

Structure

Interestingness

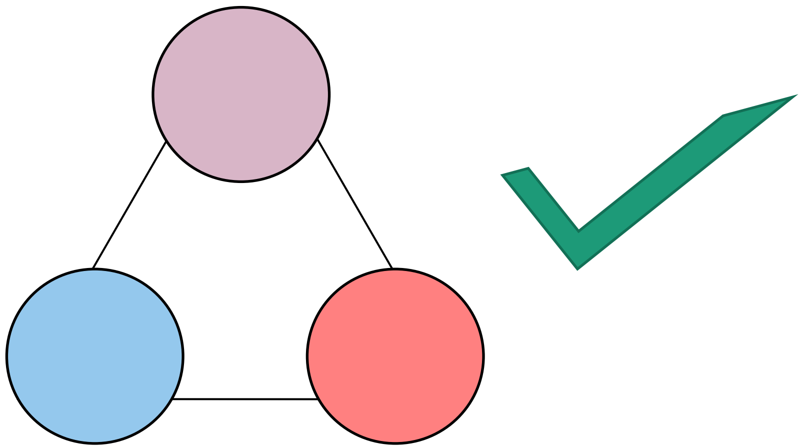


System Requirements

Uniqueness

Structure

Interestingness



Existing Work

Uniqueness

Structure

Interestingness

Arabesque (SOSP ‘15)

RStream (OSDI ‘18)

Fractal (SIGMOD ‘19)

AutoMine (SOSP ‘19)

Existing Work

Uniqueness

Structure

Interestingness

Arabesque (SOSP ‘15)

RStream (OSDI ‘18)

Fractal (SIGMOD ‘19)

AutoMine (SOSP ‘19)

Overlook user requirements

Existing Work

Uniqueness

Structure

Interestingness

Arabesque (SOSP ‘15)

RStream (OSDI ‘18)

Fractal (SIGMOD ‘19)

AutoMine (SOSP ‘19)

Overlook user requirements

Per-subgraph computations

PEREGRINE

Pattern Awareness

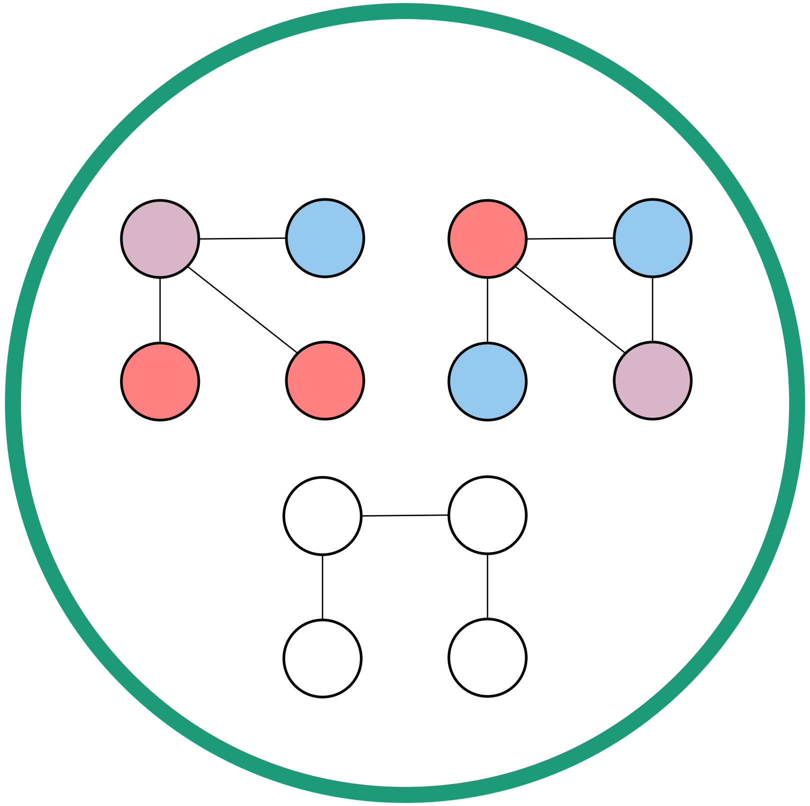
Pattern Selection

Pattern Matching

Pattern Awareness

Pattern Awareness

Pattern Selection



Pattern Programming

```
#include “Peregrine.hh”
using namespace Peregrine;

void motifCounting(int size)
{
    DataGraph G(“path/to/graph/”);
    auto patterns = PatternGenerator::all(size, VERTEX_INDUCED);
    auto counts = count(G, patterns);

    for (auto &[pattern, n] : counts)
      std::cout << pattern << “ ” << n << std::endl;
}
```

Pattern Programming

```
#include “Peregrine.hh”
using namespace Peregrine;

void motifCounting(int size)
{
    DataGraph G(“path/to/graph/”);
    auto patterns = PatternGenerator::all(size, VERTEX_INDUCED);
    auto counts = count(G, patterns);

    for (auto &[pattern, n] : counts)
      std::cout << pattern << “ ” << n << std::endl;
}
```

Pattern Programming

```
#include “Peregrine.hh”
using namespace Peregrine;

void motifCounting(int size)
{
    DataGraph G(“path/to/graph/”);
    auto patterns = PatternGenerator::all(size, VERTEX_INDUCED);
    auto counts = count(G, patterns);

    for (auto &[pattern, n] : counts)
      std::cout << pattern << “ ” << n << std::endl;
}
```

Pattern Programming

```
#include “Peregrine.hh”
using namespace Peregrine;

void motifCounting(int size)
{
    DataGraph G(“path/to/graph/”);
    auto patterns = PatternGenerator::all(size, VERTEX_INDUCED);
    auto counts = count(G, patterns);

    for (auto &[pattern, n] : counts)
      std::cout << pattern << “ ” << n << std::endl;
}
```

Pattern Programming

```
DataGraph G(“path/to/graph/”);
auto patterns = PatternGenerator::all(size, VERTEX_INDUCED);



auto counts = count(G, patterns);
```

Pattern Programming

```
DataGraph G(“path/to/graph/”);
auto patterns = PatternGenerator::all(size, VERTEX_INDUCED);
patterns[0].set_labels({‘a’, ‘b’, ‘c’, ‘d’});


auto counts = count(G, patterns);
```

Pattern Programming

```
DataGraph G("path/to/graph/");
auto patterns = PatternGenerator::all(size, VERTEX_INDUCED);
patterns[0].set_labels({'a', 'b', 'c', 'd'});
patterns[0].add_edge(1, 5);

auto counts = count(G, patterns);
```

Pattern Programming

```
DataGraph G(“path/to/graph/”);
auto patterns = PatternGenerator::all(size, VERTEX_INDUCED);
patterns[0].set_labels({‘a’, ‘b’, ‘c’, ‘d’});
patterns[0].add_edge(1, 5);
patterns.emplace_back(“path/to/pattern.txt”);
auto counts = count(G, patterns);
```

Pattern Programming

```
DataGraph G(“path/to/graph/”);
auto patterns = PatternGenerator::all(size, VERTEX_INDUCED);
auto pattern = Pattern().add_edge(1, 2)
                        .add_edge(1, 3)
                        .add_edge(2, 3);
auto counts = count(G, {pattern});
```

Pattern Programming

```
#include "Peregrine.hh"
using namespace Peregrine;

void motifCounting(int size)
{
    DataGraph G("path/to/graph/");
    auto patterns = PatternGenerator::all(size, VERTEX_INDUCED);
    auto counts = count(G, patterns);

    for (auto &[pattern, n] : counts)
      std::cout << pattern << " " << n << std::endl;
}
```

Pattern Programming

```
#include “Peregrine.hh”
using namespace Peregrine;

void motifCounting(int size)
{
    DataGraph G(“path/to/graph/”);
    auto patterns = PatternGenerator::all(size, VERTEX_INDUCED);
    auto counts = count(G, patterns);

    for (auto &[pattern, n] : counts)
      std::cout << pattern << “ ” << n << std::endl;
}
```

Pattern Programming

```
void frequentSubgraphMining()
{
   DataGraph G("path/to/graph/");
   auto patterns = PatternGenerator::all(2, EDGE_INDUCED);
   auto mapDomain = [](auto &&match, auto &&aggregator)
      { aggregator.map(match.pattern, match.mapping); };

   auto results = match<Pattern, Domain>(G, patterns, mapDomain);

   for (auto &[pattern, frequency] : results)
     std::cout << pattern << " " << frequency << std::endl;
}
```

Pattern Programming

```
void frequentSubgraphMining()
{
   DataGraph G("path/to/graph/");
   auto patterns = PatternGenerator::all(2, EDGE_INDUCED);
   auto mapDomain = [](auto &&match, auto &&aggregator)
      { aggregator.map(match.pattern, match.mapping); };

   auto results = match<Pattern, Domain>(G, patterns, mapDomain);

   for (auto &[pattern, frequency] : results)
     std::cout << pattern << " " << frequency << std::endl;
}
```

Pattern Programming

```
void frequentSubgraphMining()
{
   DataGraph G("path/to/graph/");
   auto patterns = PatternGenerator::all(2, EDGE_INDUCED);
   auto mapDomain = [](auto &&match, auto &&aggregator)
      { aggregator.map(match.pattern, match.mapping); };

   auto results = match<Pattern, Domain>(G, patterns, mapDomain);

   for (auto &[pattern, frequency] : results)
     std::cout << pattern << " " << frequency << std::endl;
}
```

Pattern Programming

```
void frequentSubgraphMining()
{
   DataGraph G("path/to/graph/");
   auto patterns = PatternGenerator::all(2, EDGE_INDUCED);
   auto mapDomain = [](auto &&match, auto &&aggregator)
      { aggregator.map(match.pattern, match.mapping); };

   auto results = match<Pattern, Domain>(G, patterns, mapDomain);

   for (auto &[pattern, frequency] : results)
     std::cout << pattern << " " << frequency << std::endl;
}
```

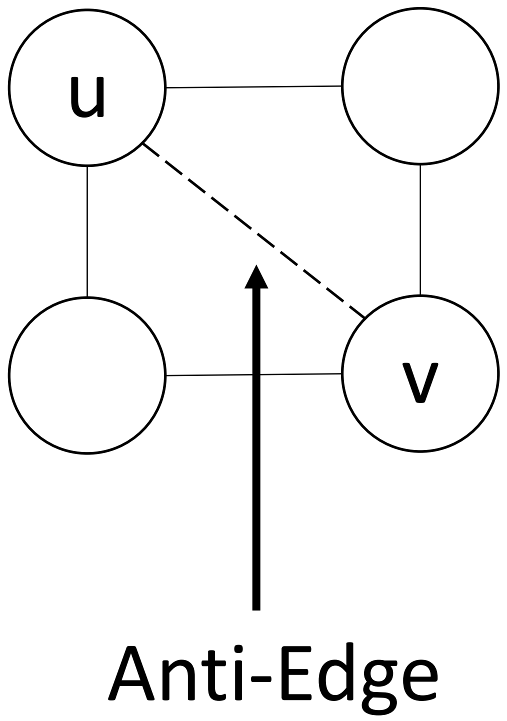
Pattern Awareness

Pattern Selection

Pattern Matching

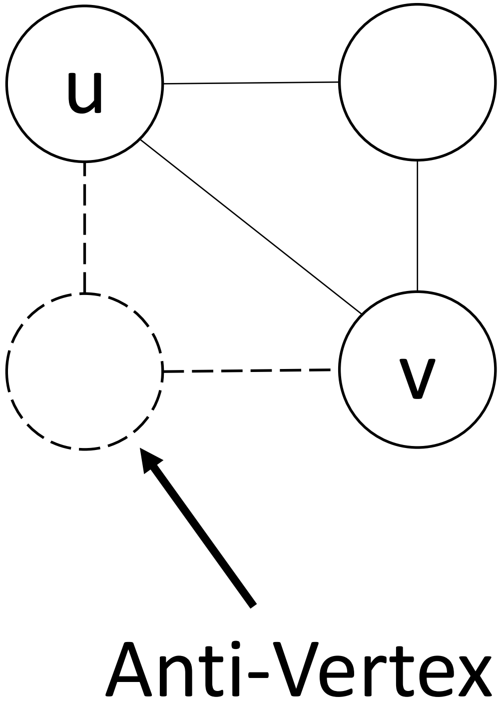
Pattern Awareness

Pattern Selection



Pattern Awareness

Pattern Selection



Pattern Awareness

Pattern Selection

Pattern Matching

Pattern Awareness

Pattern Selection

Pattern Matching

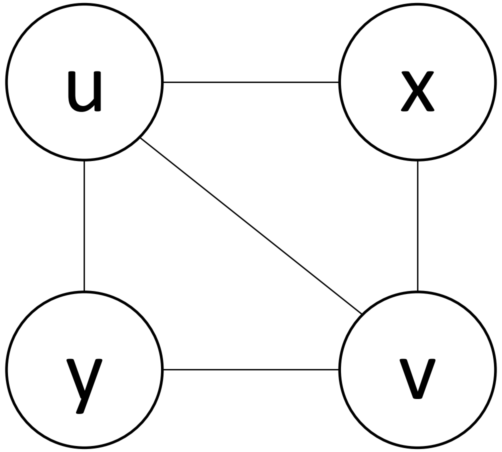
Pattern Awareness

Pattern Matching

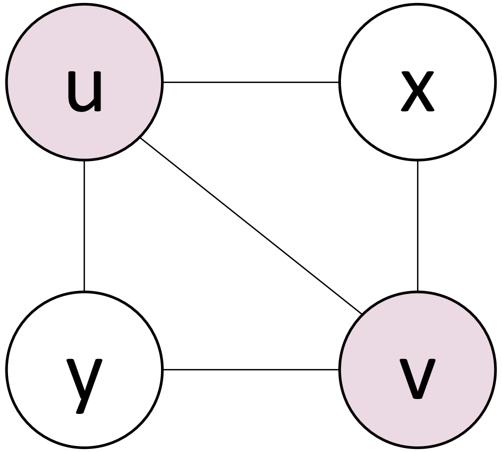
- Symmetry breaking (RECOMB '07)
- Core pattern reduction (SIGMOD '16)

Symmetry Breaking

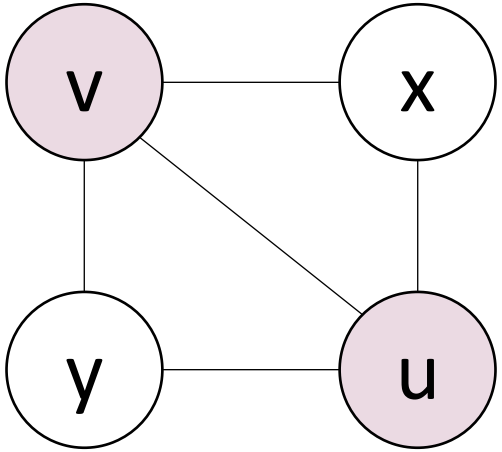
Symmetry Breaking



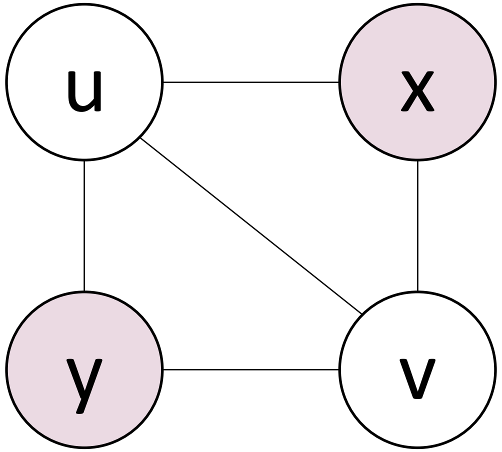
Symmetry Breaking



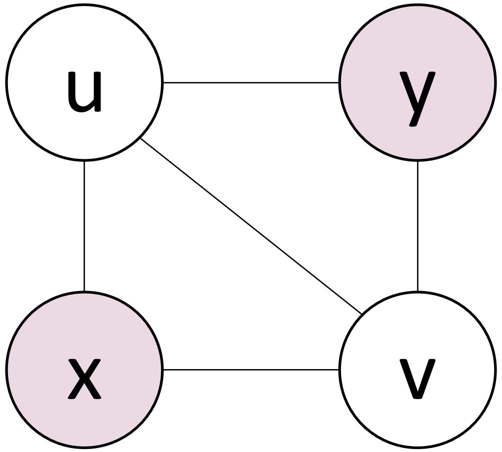
Symmetry Breaking



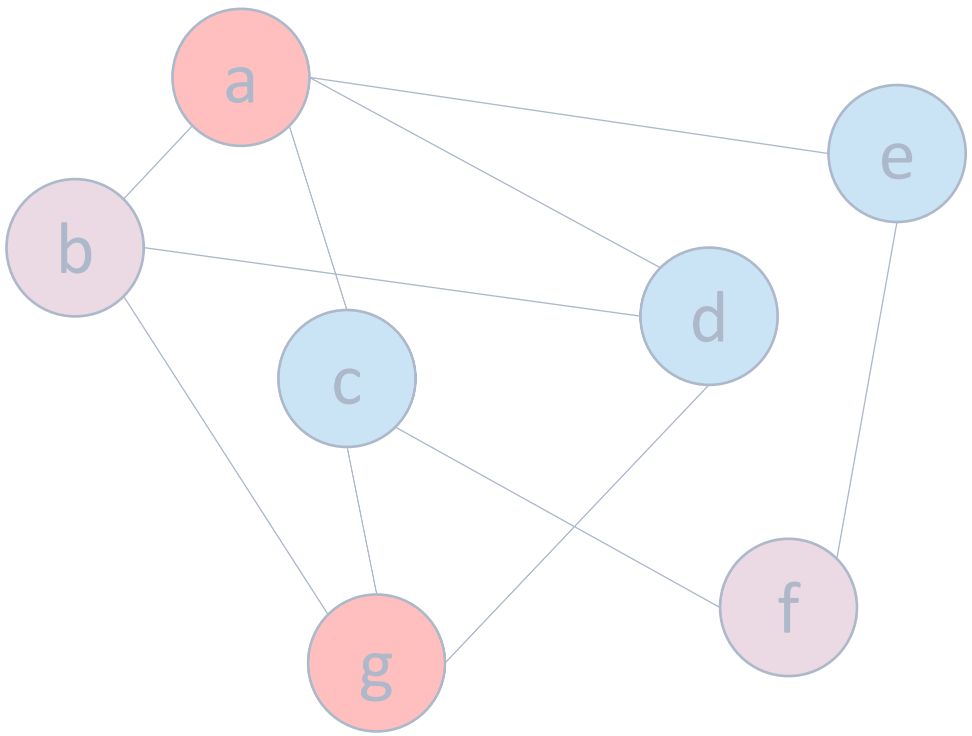
Symmetry Breaking



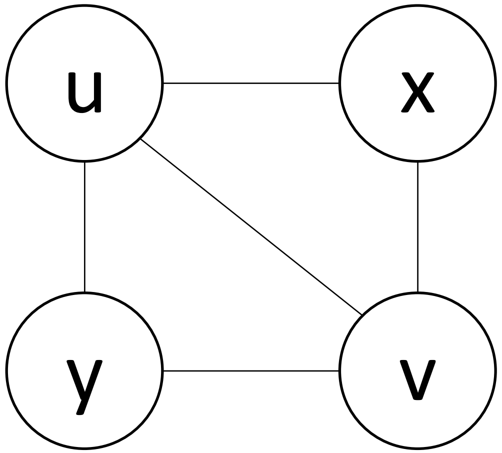
Symmetry Breaking



Symmetry Breaking

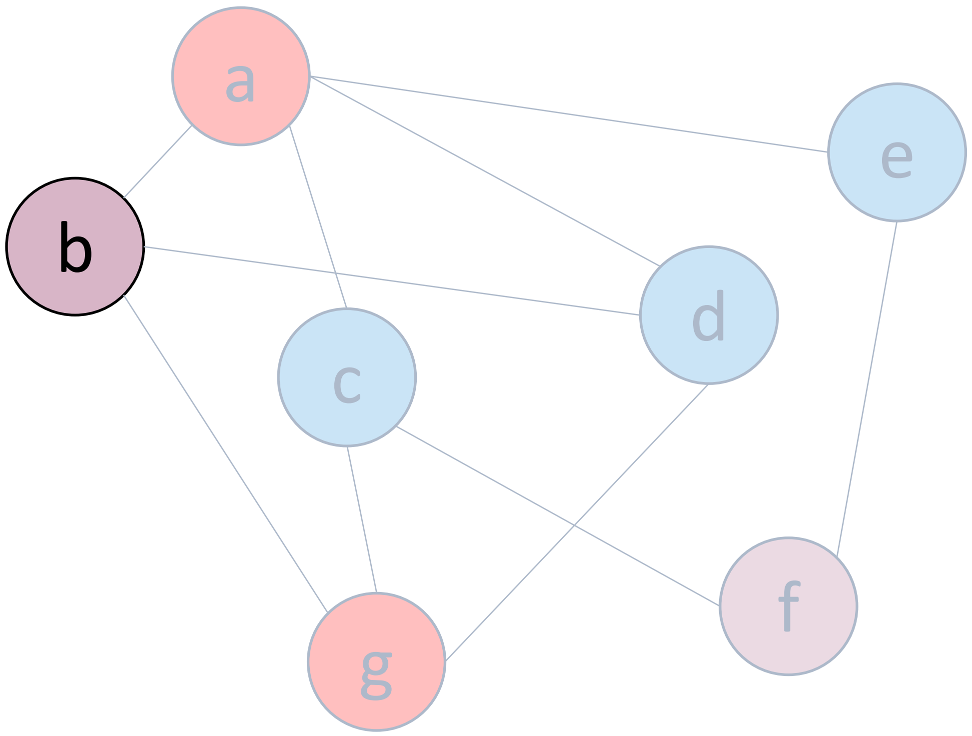


Data Graph

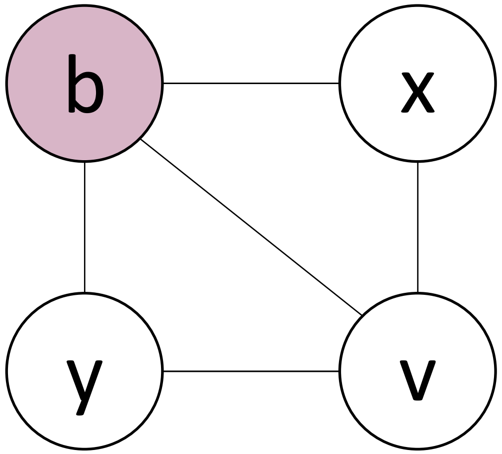


Worker 1

Symmetry Breaking

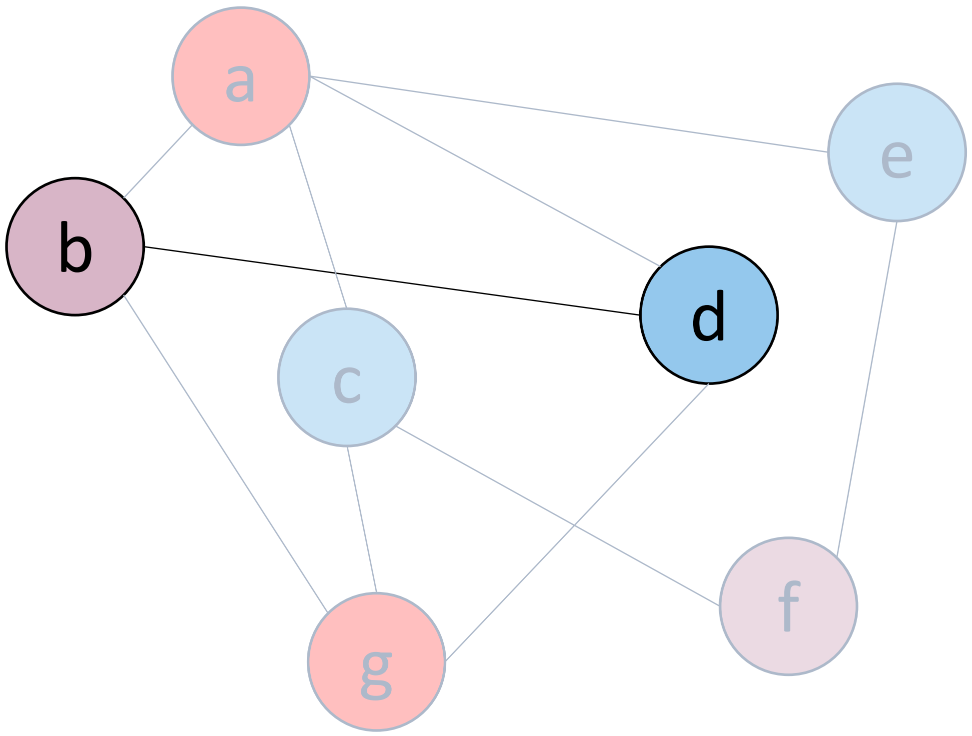


Data Graph

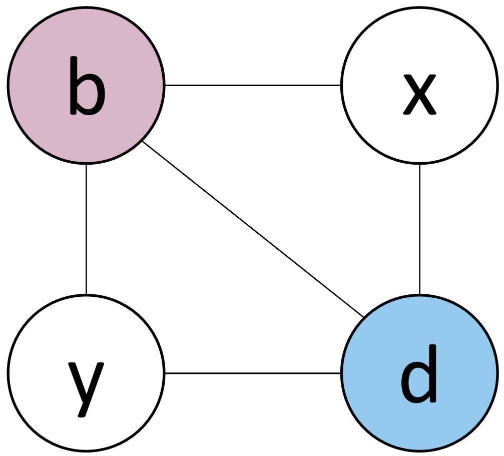


Worker 1

Symmetry Breaking

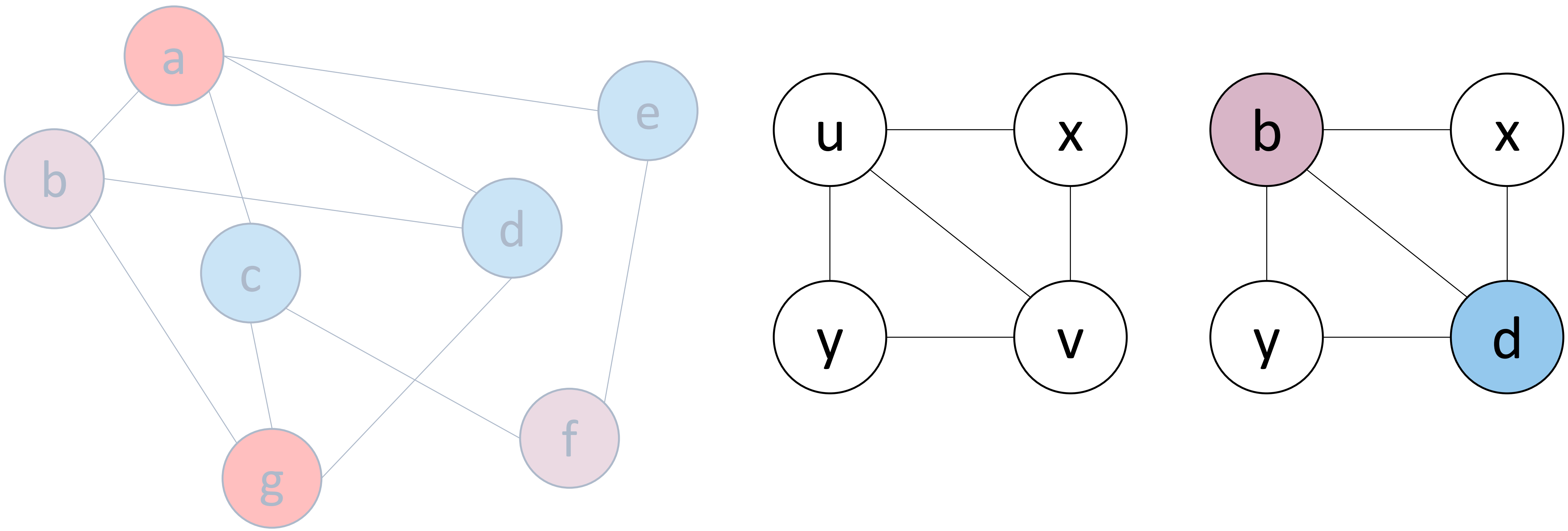


Data Graph



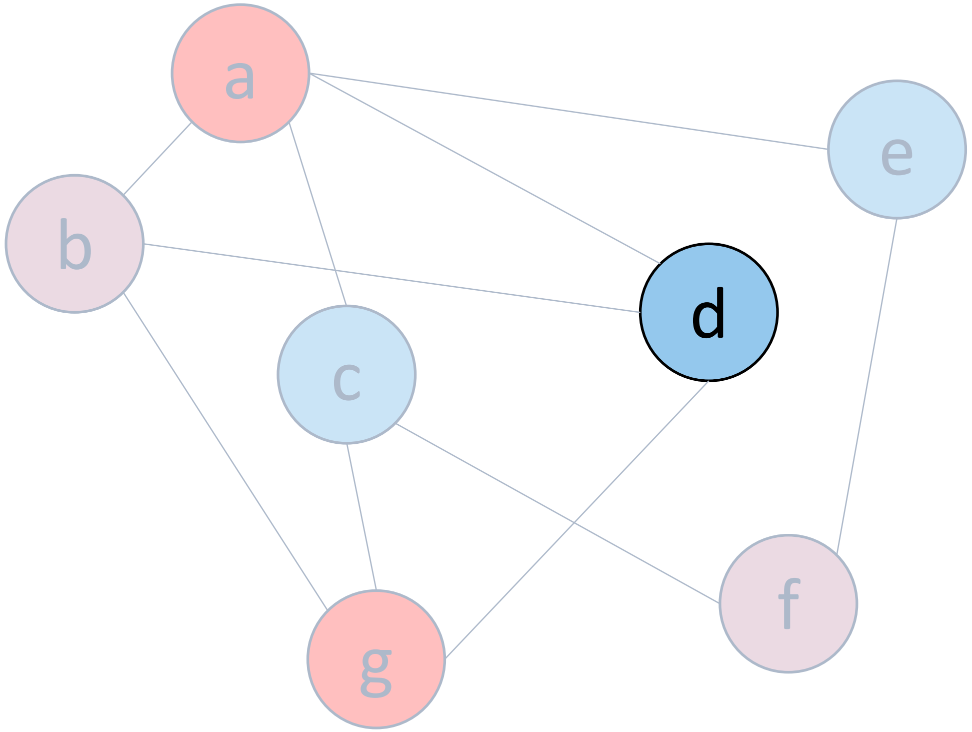
Worker 1

Symmetry Breaking

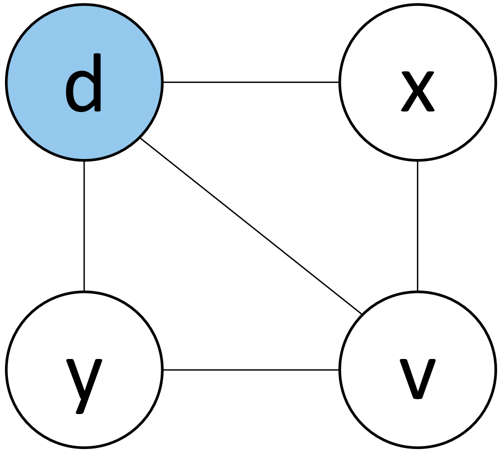


Data Graph Worker 2 Worker 1

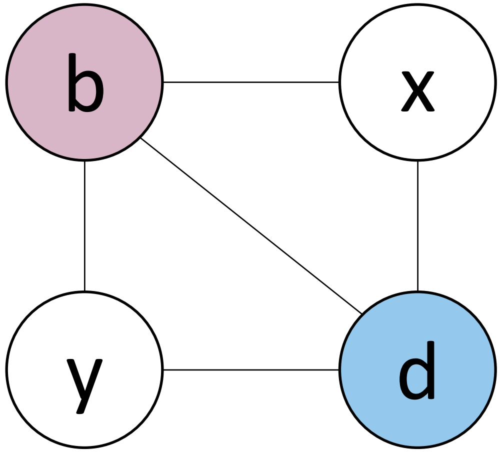
Symmetry Breaking



Data Graph

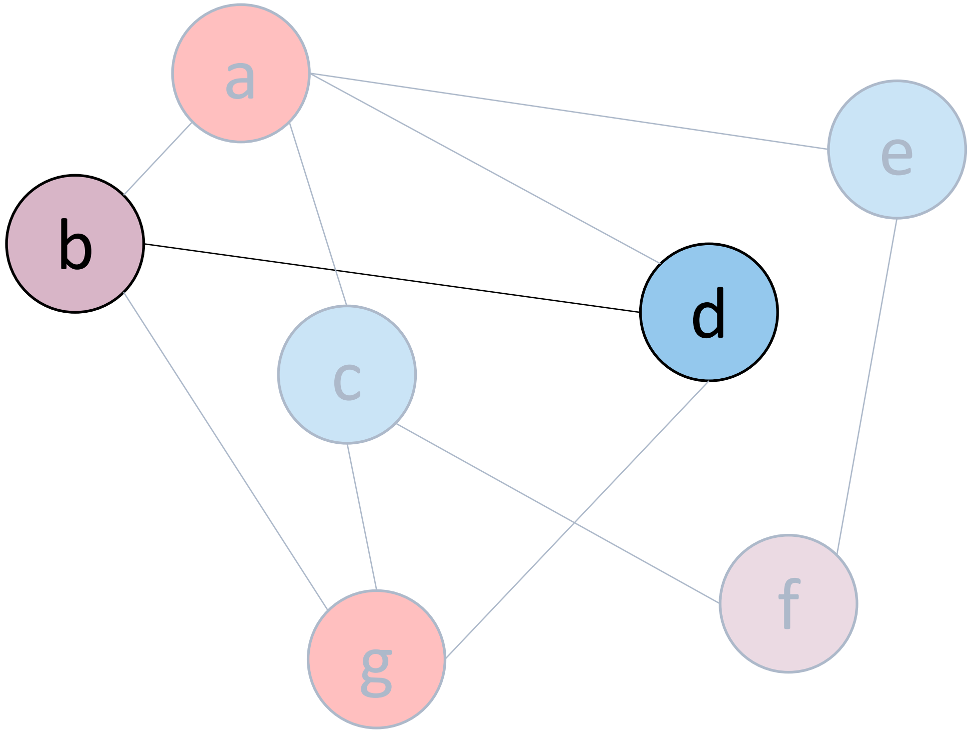


Worker 2

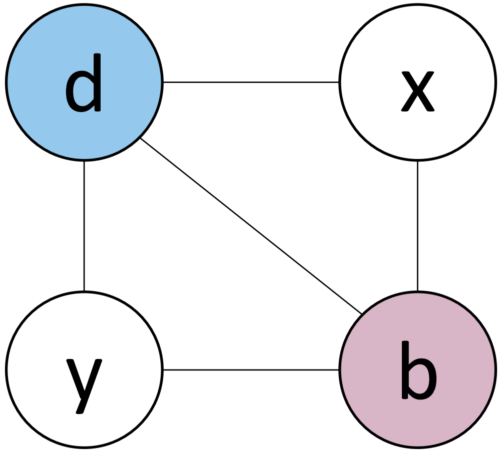


Worker 1

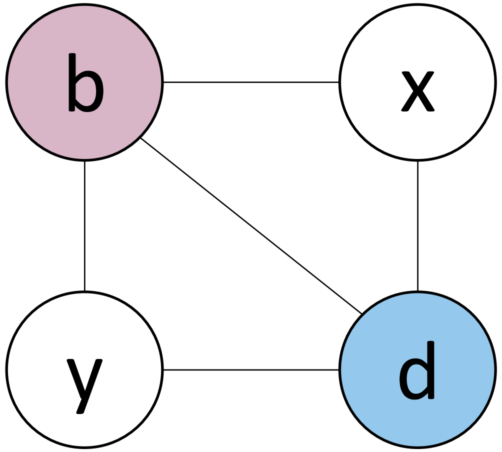
Symmetry Breaking



Data Graph

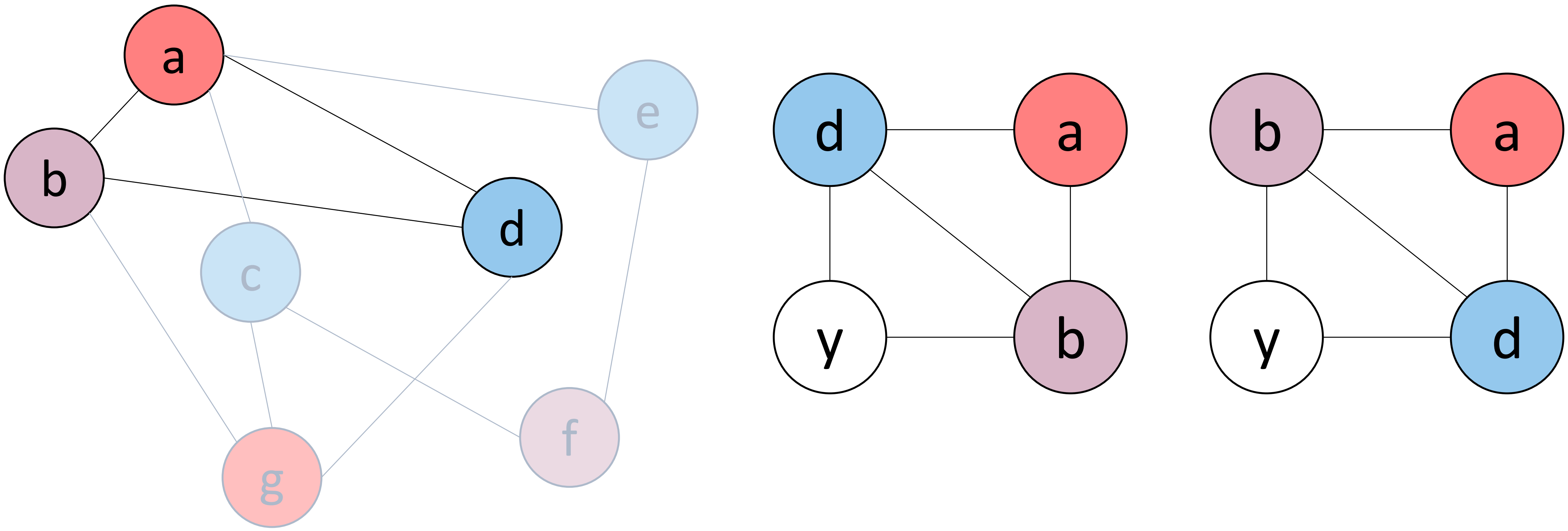


Worker 2



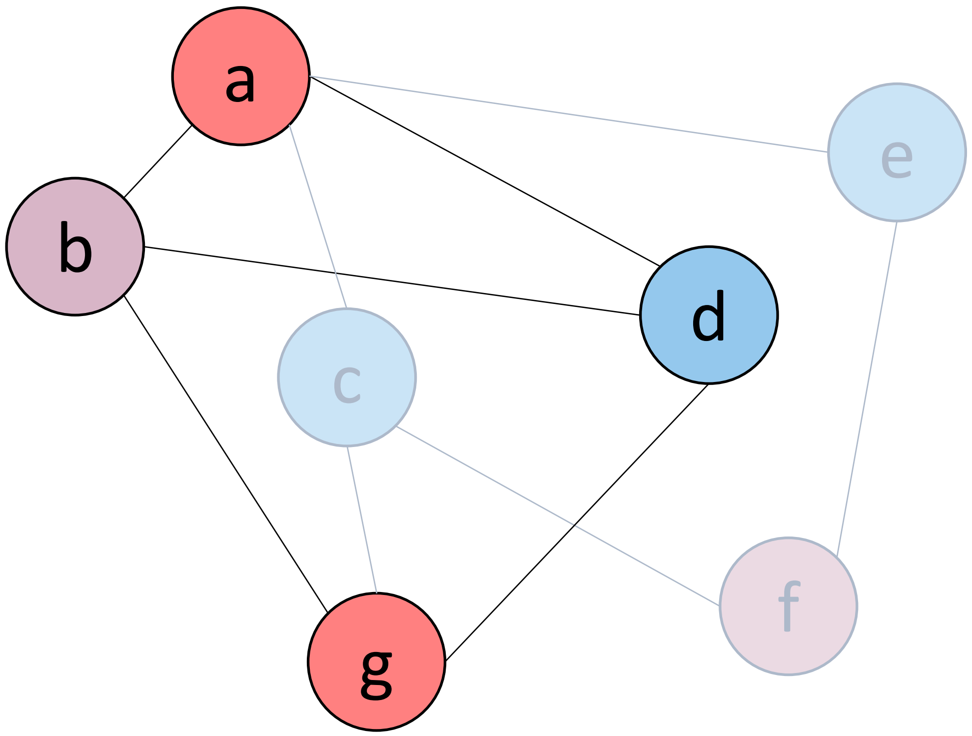
Worker 1

Symmetry Breaking

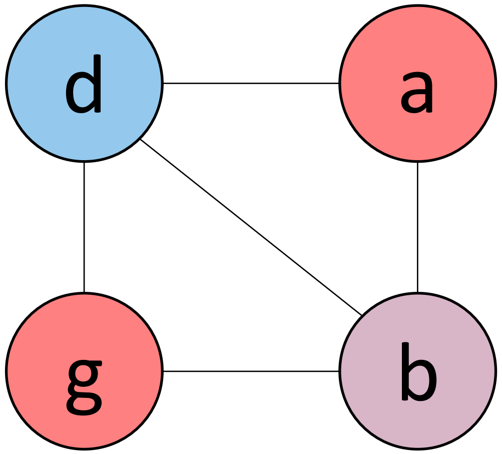


Data Graph Worker 2 Worker 1

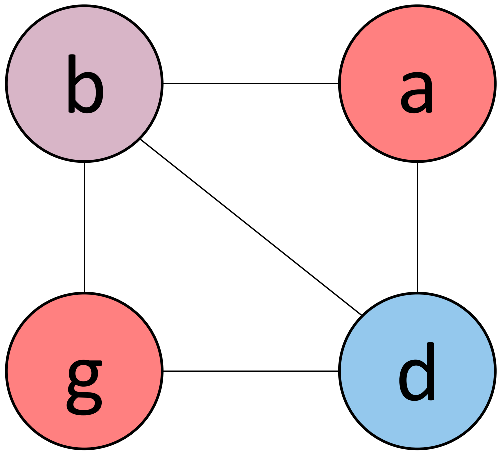
Symmetry Breaking



Data Graph

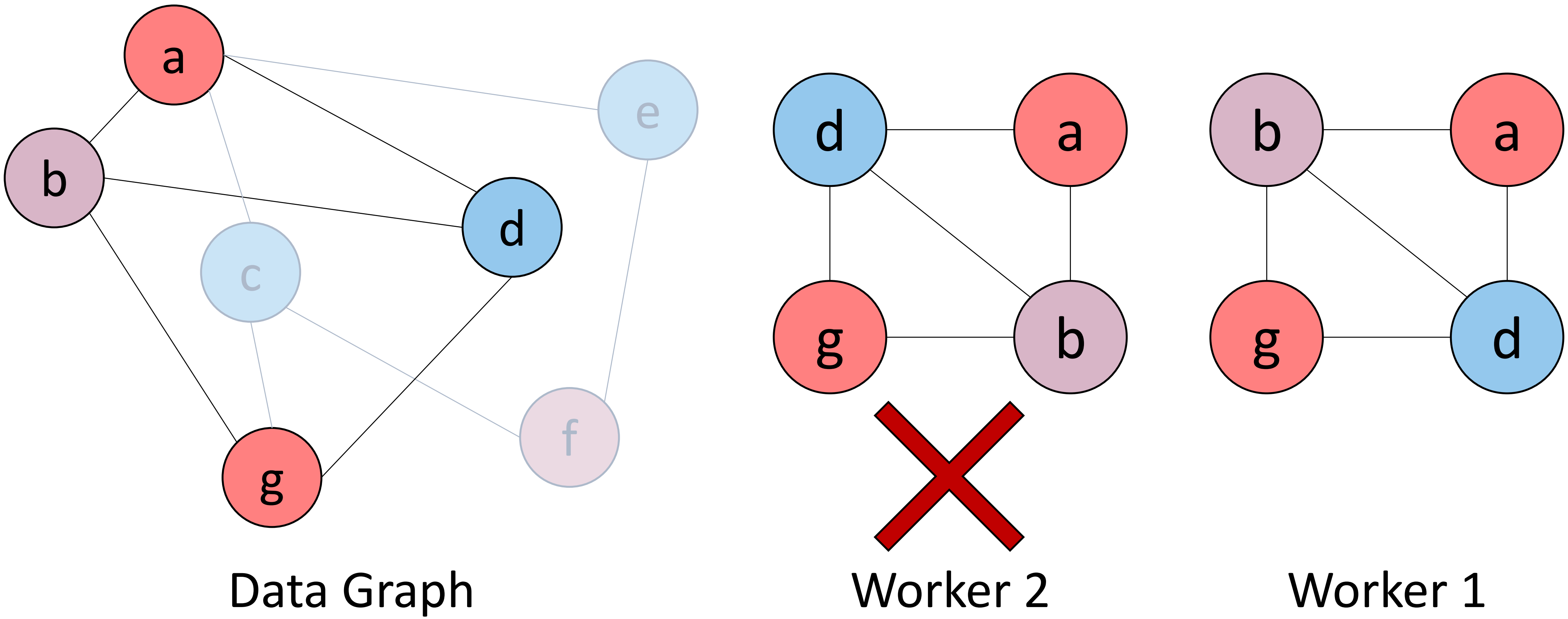


Worker 2

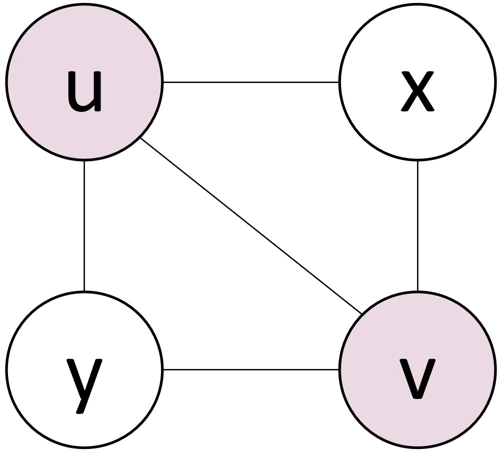


Worker 1

Symmetry Breaking

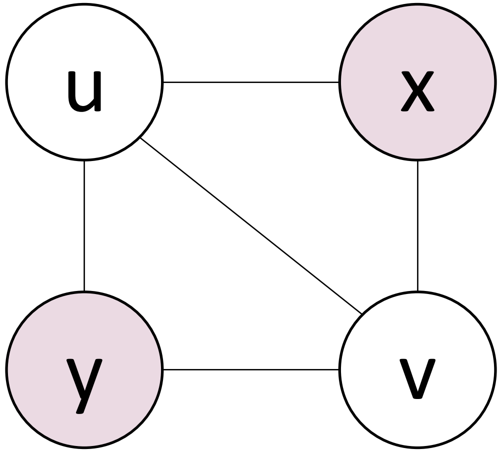


Symmetry Breaking



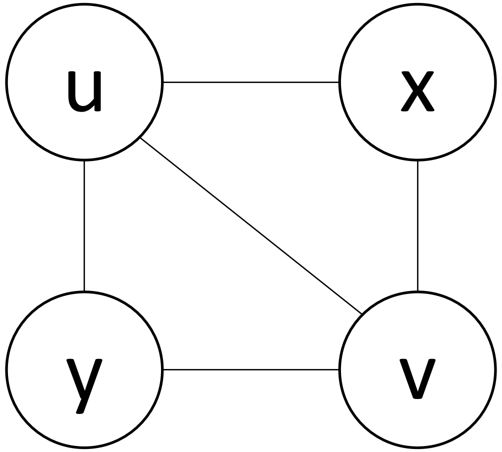
u < v

Symmetry Breaking



x < y

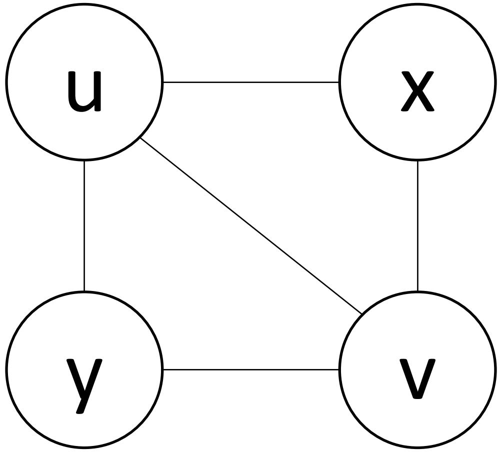
Symmetry Breaking



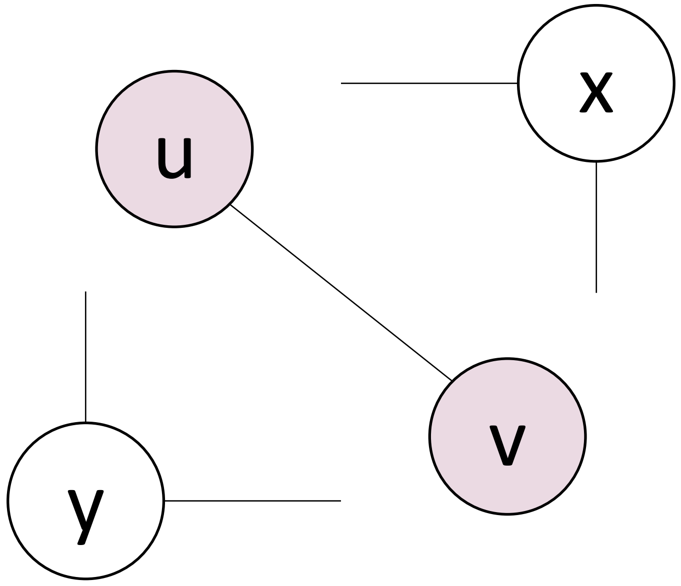
u < v

x < y

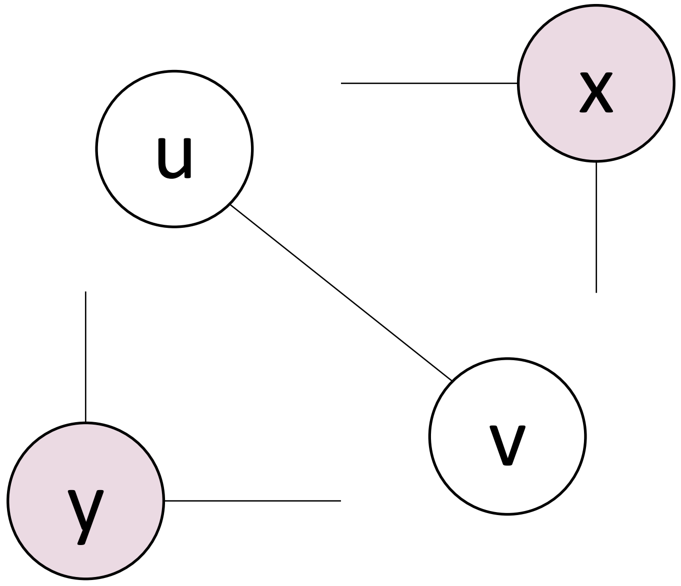
Core Pattern



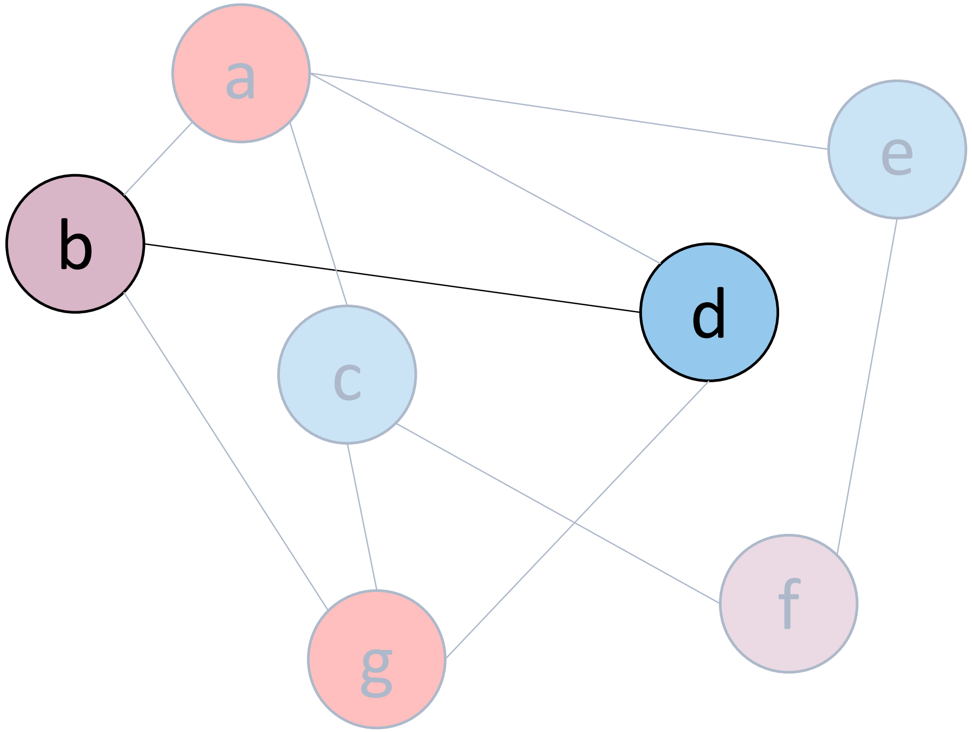
Core Pattern



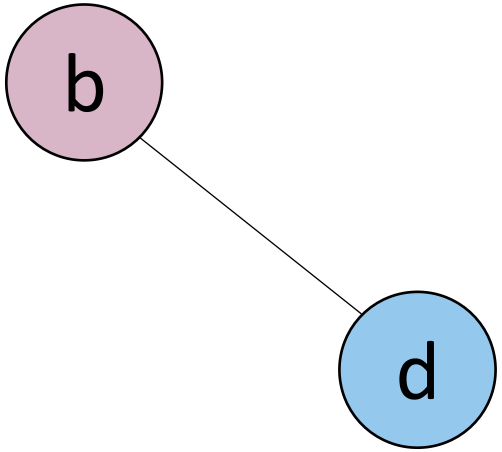
Core Pattern



Core Pattern

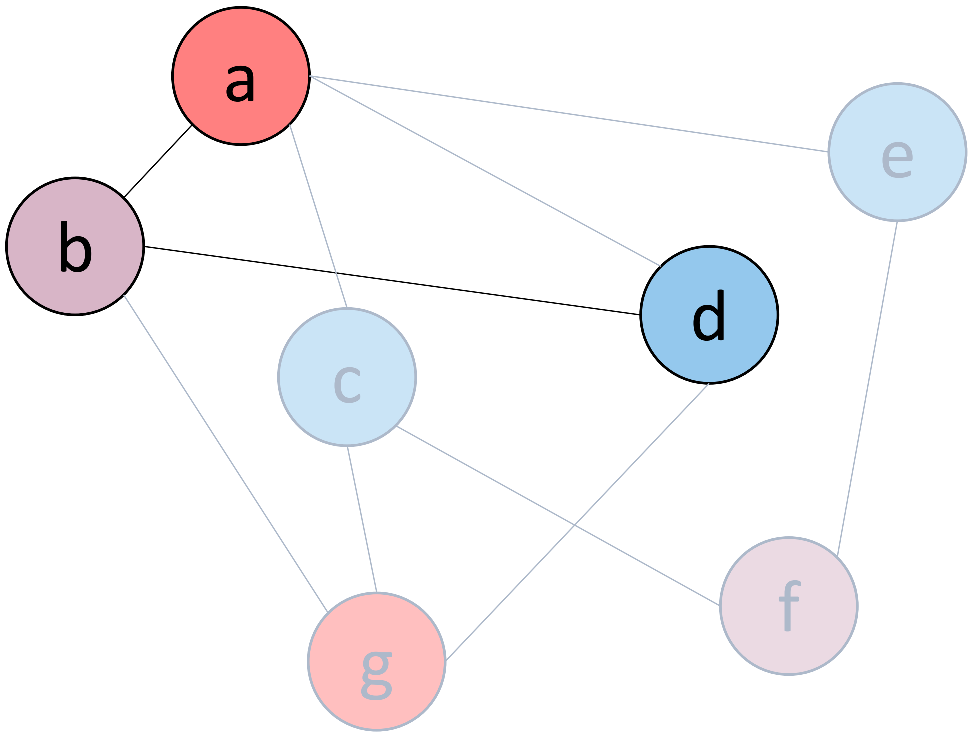


Data Graph

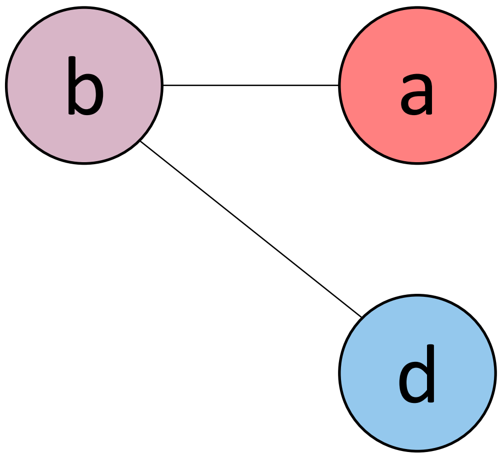


Pattern-Unaware

Core Pattern

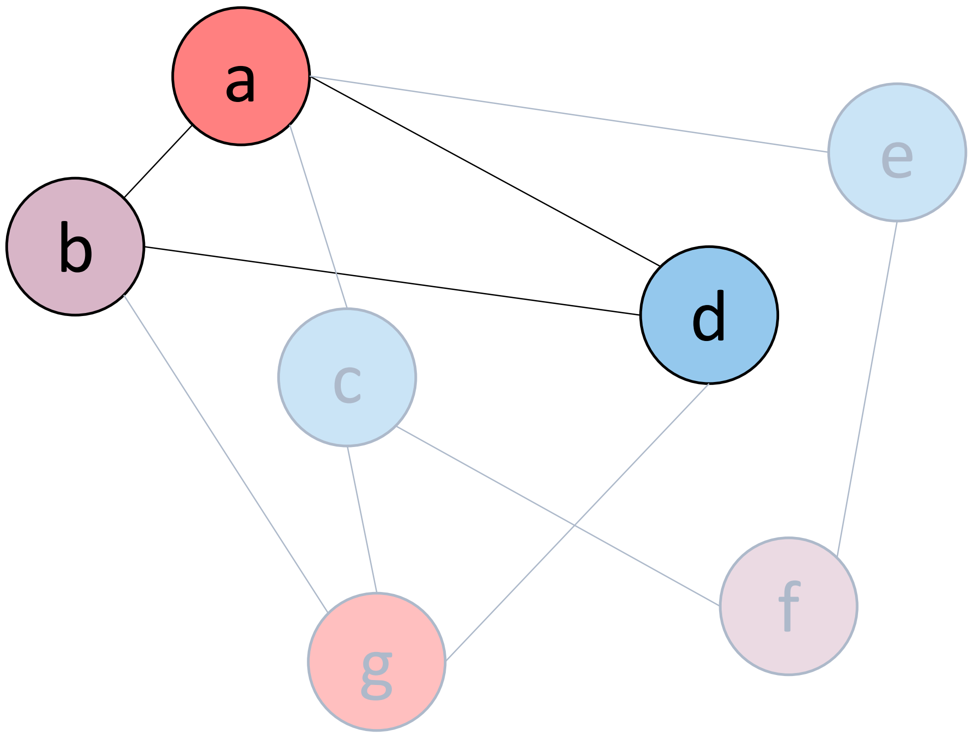


Data Graph

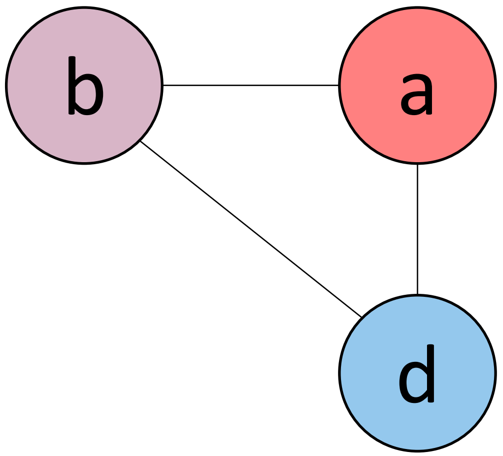


Pattern-Unaware

Core Pattern

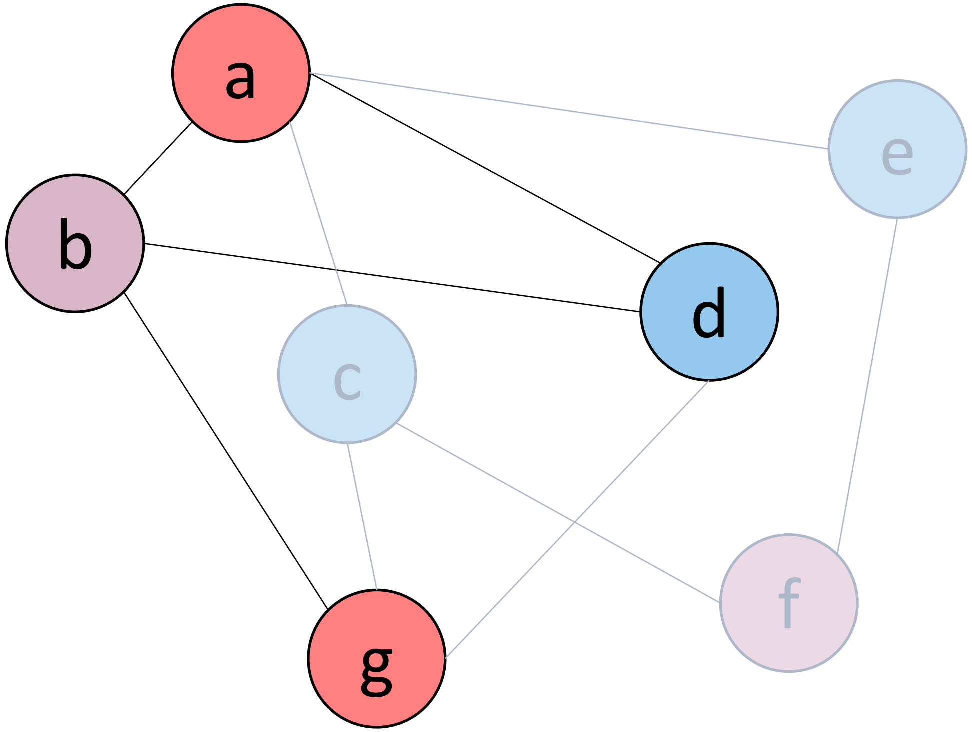


Data Graph

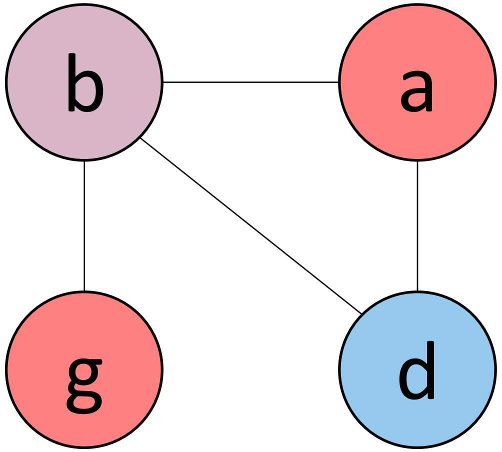


Pattern-Unaware

Core Pattern

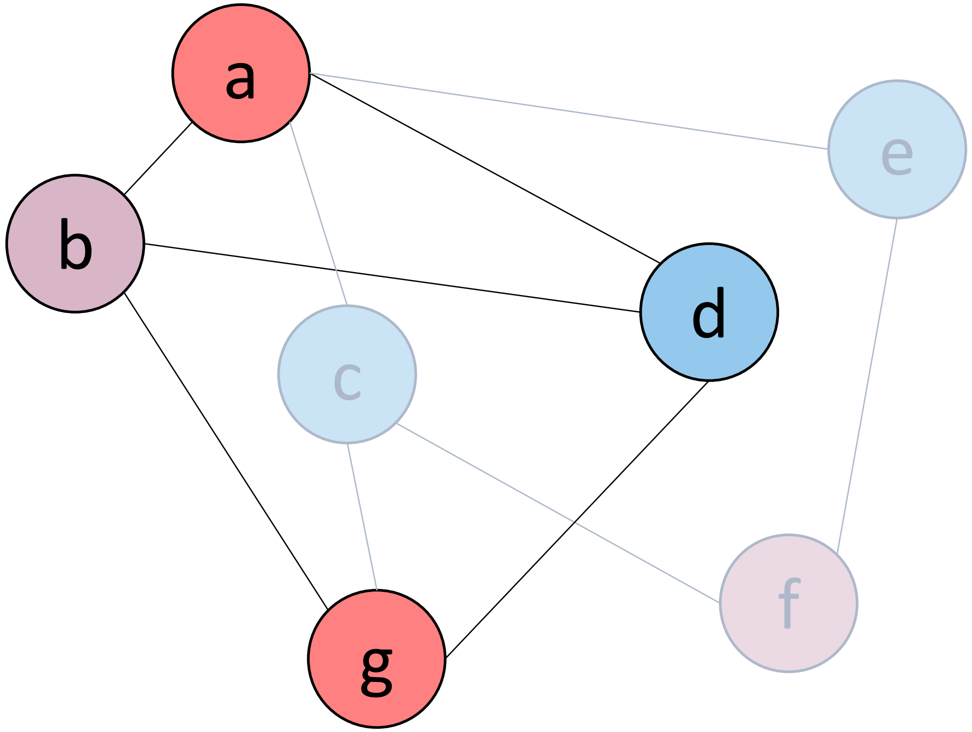


Data Graph

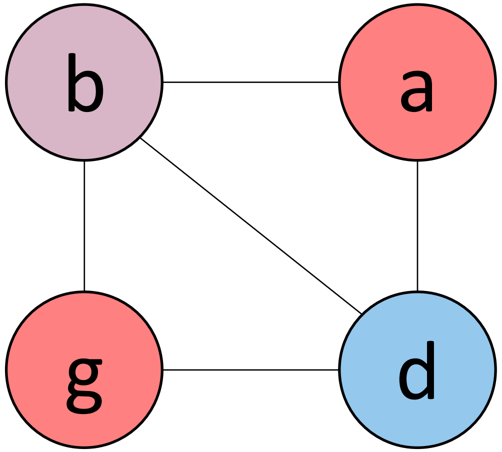


Pattern-Unaware

Core Pattern

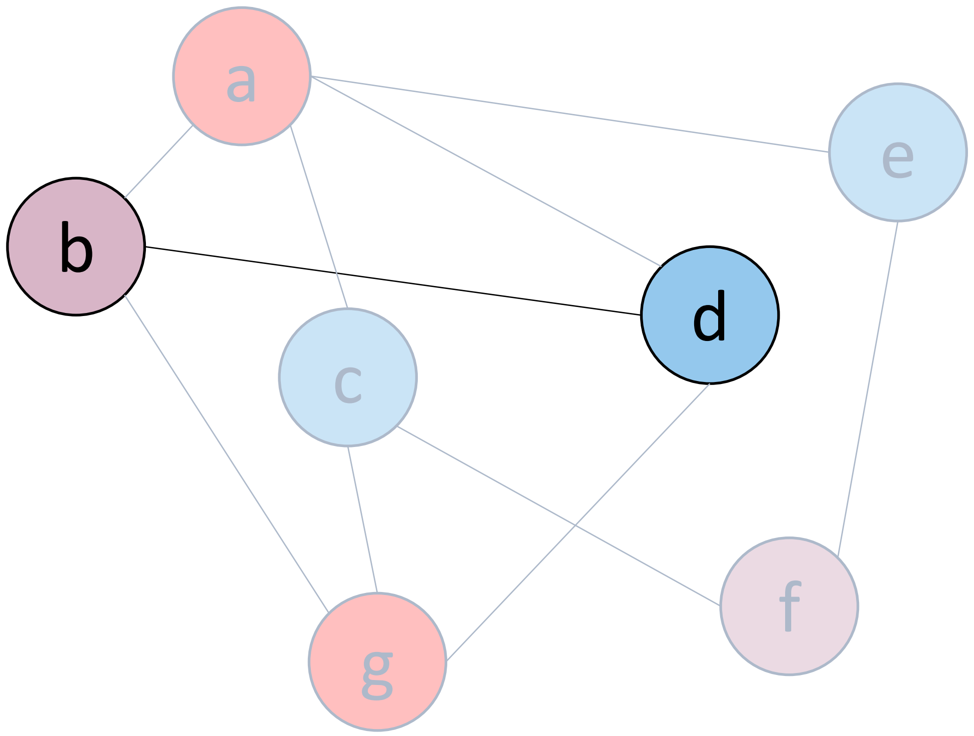


Data Graph

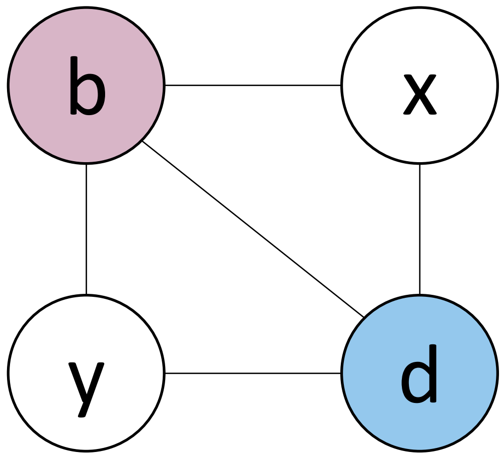


Pattern-Unaware

Core Pattern

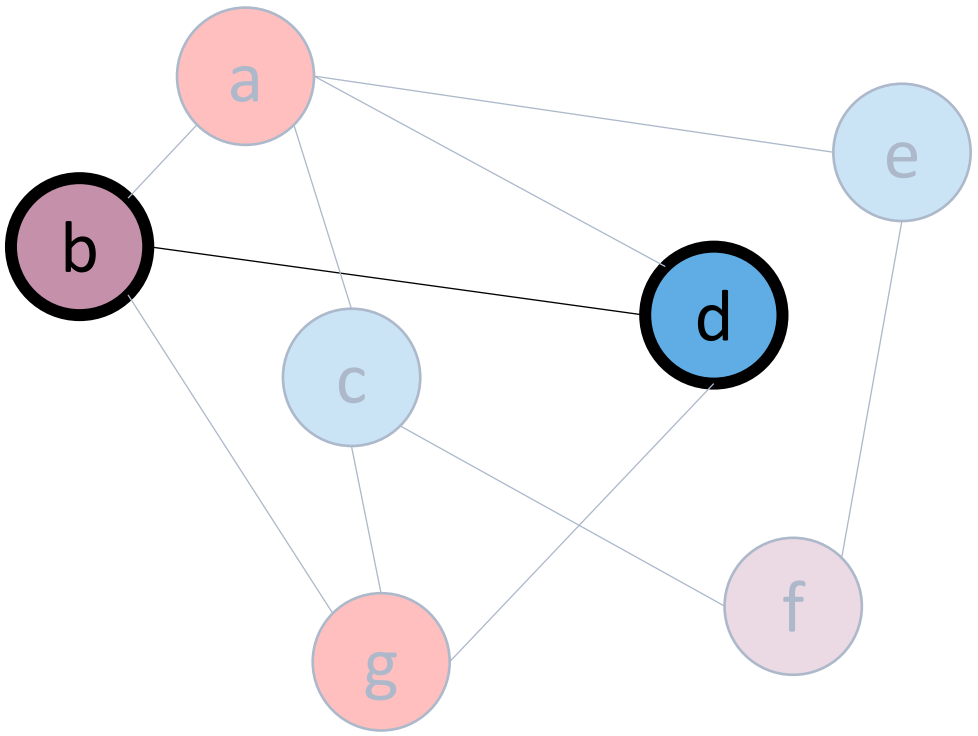


Data Graph

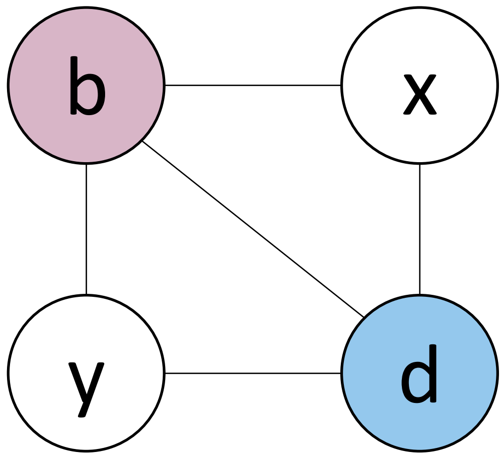


Pattern-Aware

Core Pattern

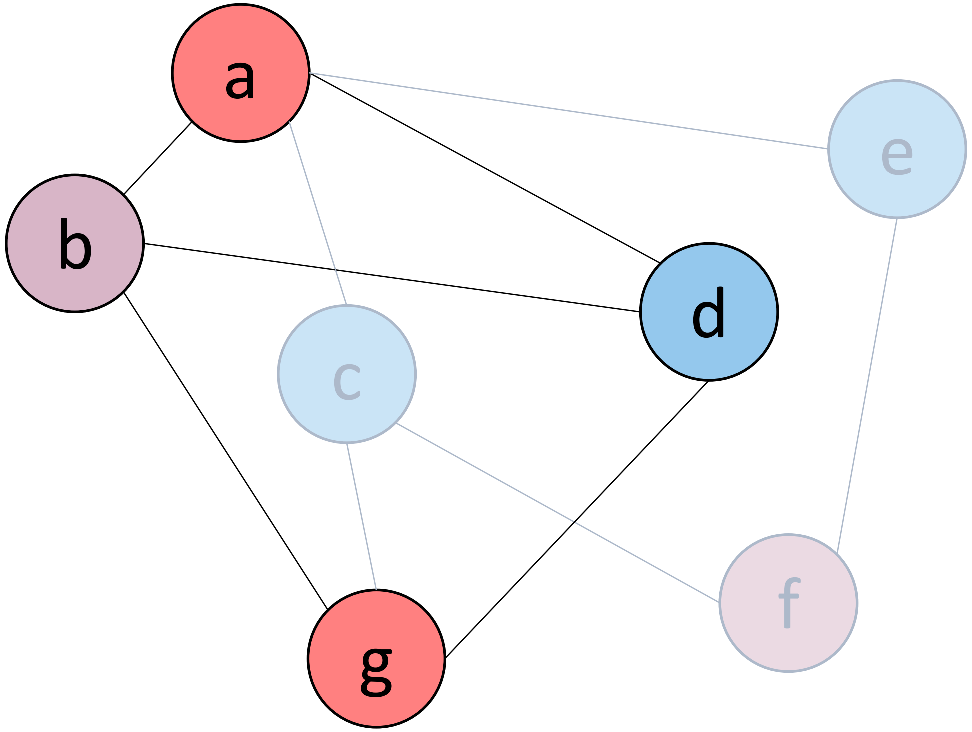


Data Graph

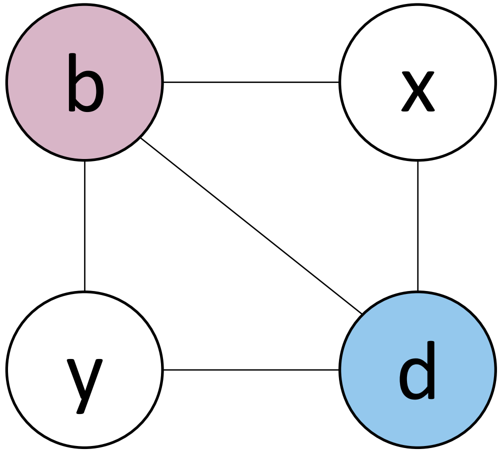


Pattern-Aware

Core Pattern

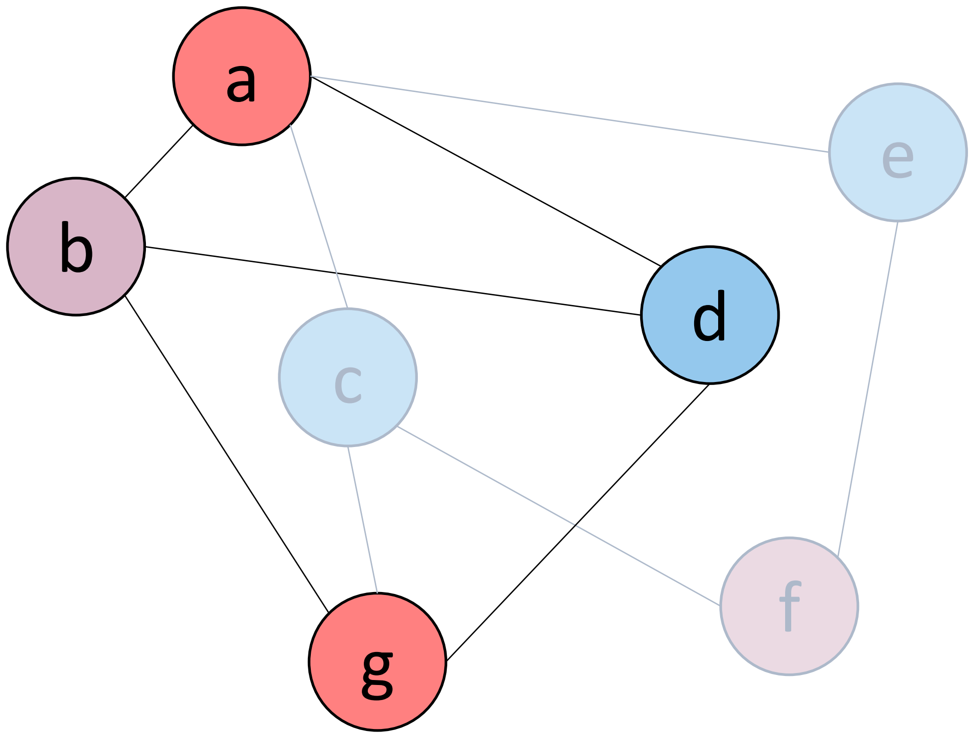


Data Graph

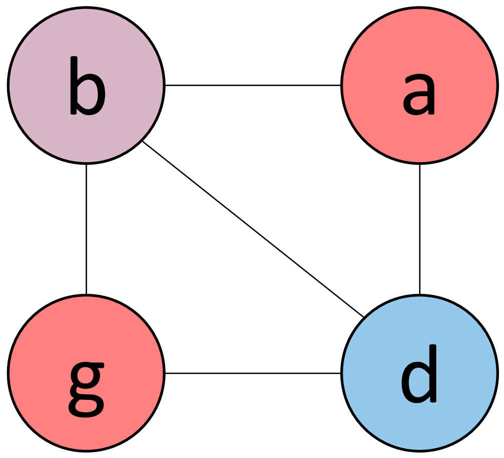


Pattern-Aware

Core Pattern

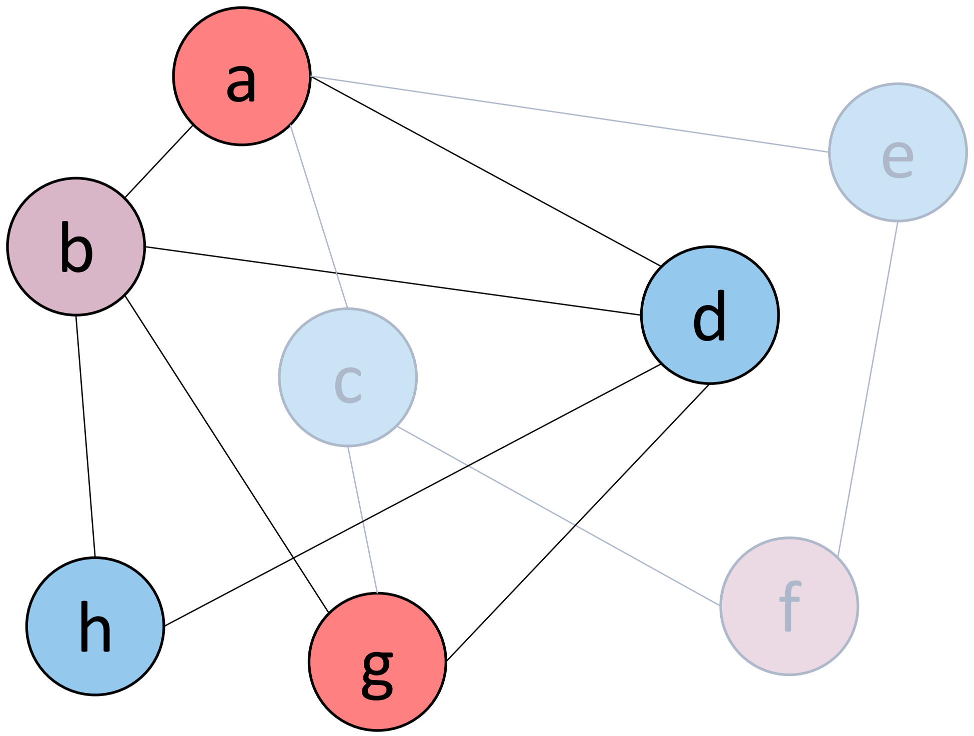


Data Graph

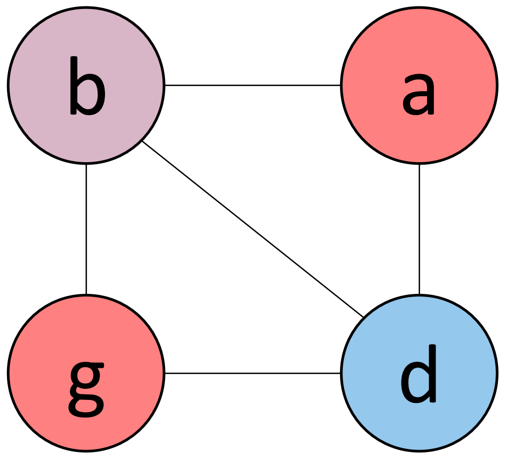


Pattern-Aware

Core Pattern

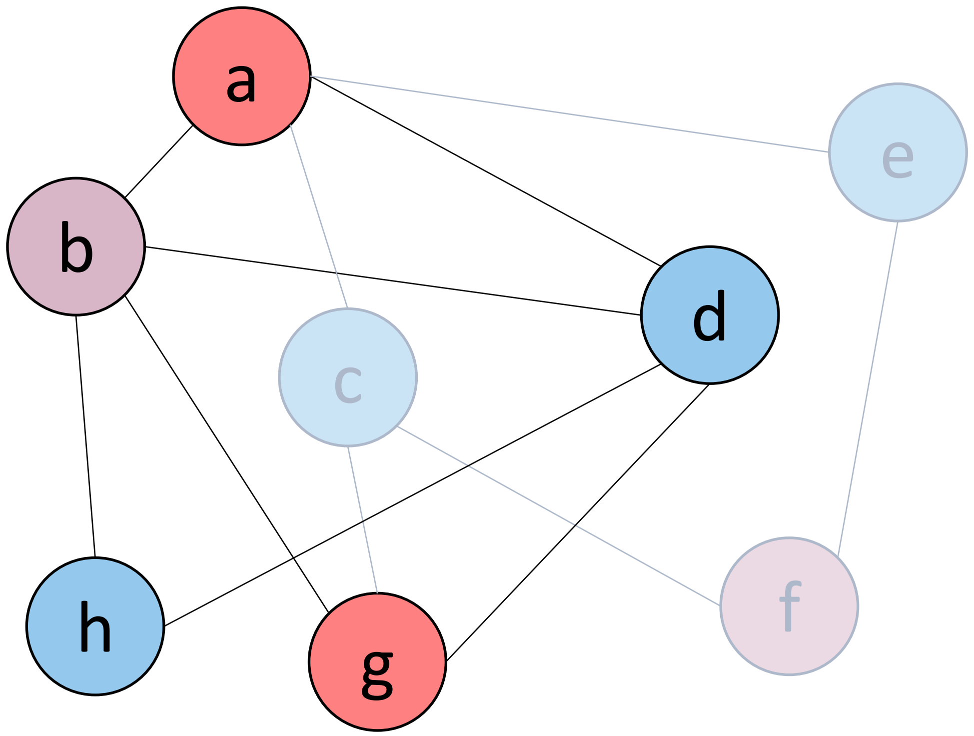


Data Graph

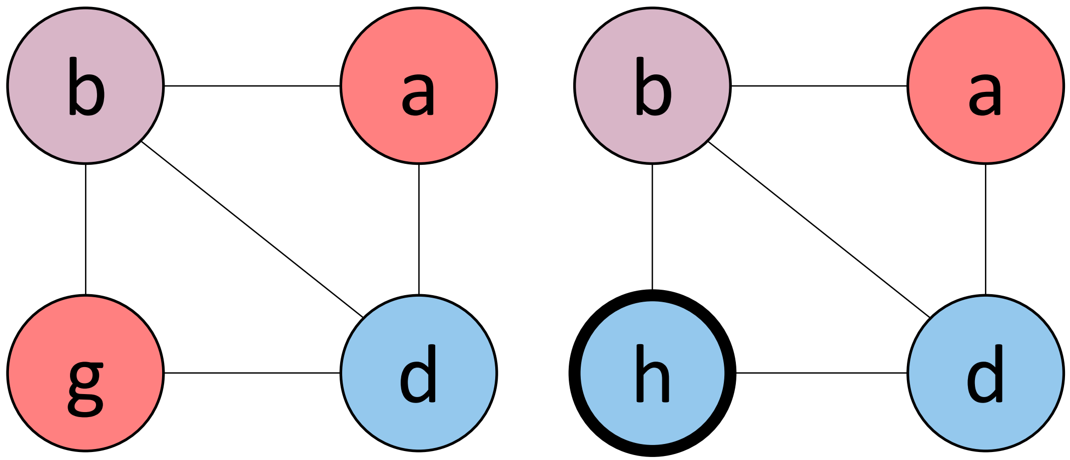


Pattern-Aware

Core Pattern

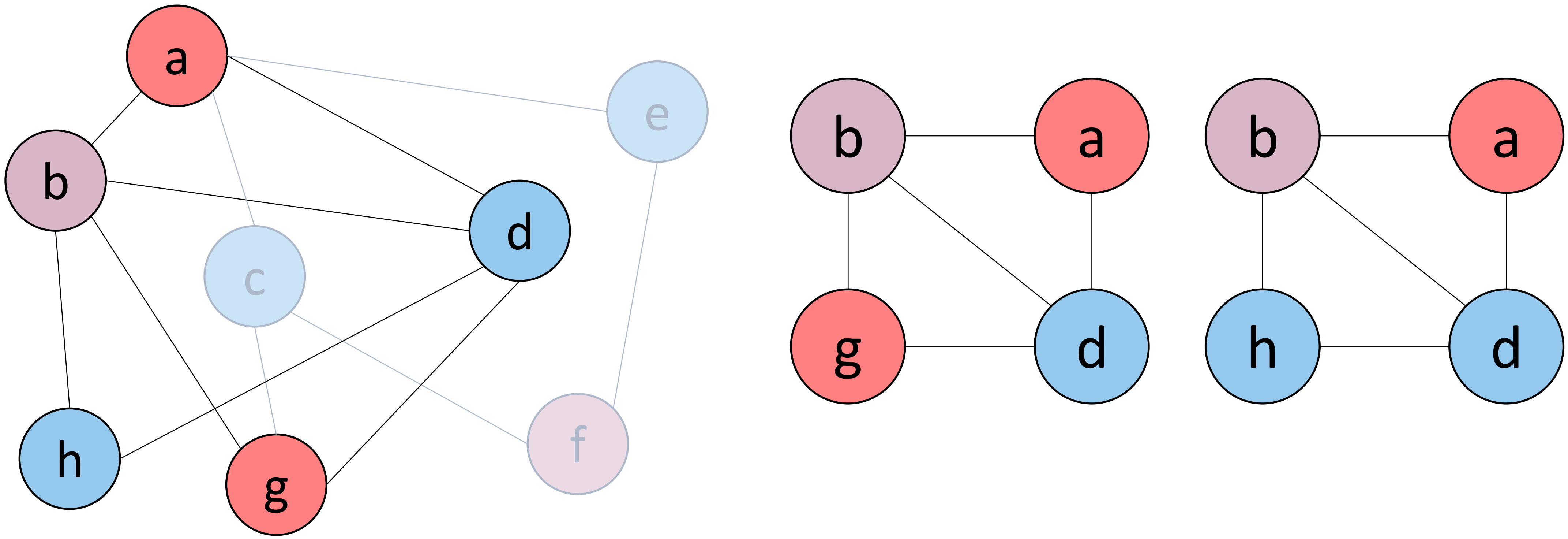


Data Graph



Pattern-Aware

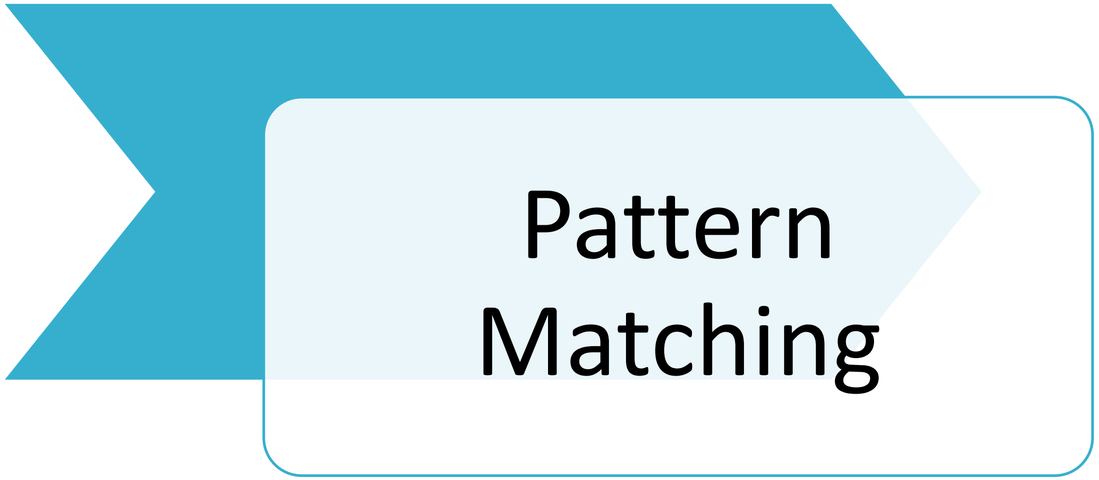
Core Pattern



Data Graph

Pattern-Aware

Pattern Awareness



- Symmetry breaking (RECOMB ‘07)
- Core pattern reduction (SIGMOD ‘16)

Pattern Awareness

- Early termination

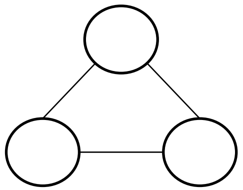
Early Termination

```
bool globalClusteringCoefficient(int bound)
{
  DataGraph G("path/to/graph/");
  auto triplet = PatternGenerator::star(3);
  int numTriplets = count(G, {triplet});
  auto countAndCheck = [=](auto &&match, auto &&aggregator)
  {
    int numTriangles = aggregator.readValue(match.pattern);
    if (3*numTriangles/numTriplets > bound) aggregator.stop();
    else aggregator.map(match.pattern, 1);
  }
  auto triangle = PatternGenerator::clique(3);
  auto result = match<Pattern, int>(G, triangle, countAndCheck);
  return 3*result[triangle]/numTriplets > bound;
}
```

Early Termination

```
bool globalClusteringCoefficient(int bound)
{
  DataGraph G("path/to/graph/");
  auto triplet = PatternGenerator::star(3);
  int numTriplets = count(G, {triplet});
  auto countAndCheck = [=](auto &&match, auto &&aggregator)
  {
    int numTriangles = aggregator.readValue(match.pattern);
    if (3*numTriangles/numTriplets > bound) aggregator.stop();
    else aggregator.map(match.pattern, 1);
  }
  auto triangle = PatternGenerator::clique(3);
  auto result = match<Pattern, int>(G, triangle, countAndCheck);
  return 3*result[triangle]/numTriplets > bound;
}
```

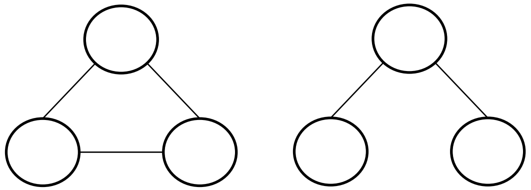
Early Termination



Triangle

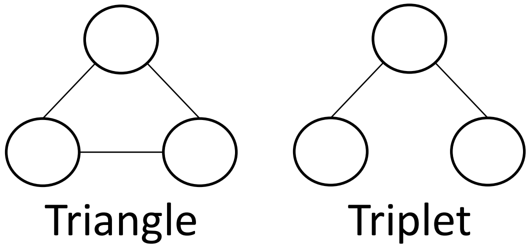
```
bool globalClusteringCoefficient(int bound)
{
  DataGraph G("path/to/graph/");
  auto triplet = PatternGenerator::star(3);
  int numTriplets = count(G, {triplet});
  auto countAndCheck = [=](auto &&match, auto &&aggregator)
  {
    int numTriangles = aggregator.readValue(match.pattern);
    if (3*numTriangles/numTriplets > bound) aggregator.stop();
    else aggregator.map(match.pattern, 1);
  }
  auto triangle = PatternGenerator::clique(3);
  auto result = match<Pattern, int>(G, triangle, countAndCheck);
  return 3*result[triangle]/numTriplets > bound;
}
```

Early Termination



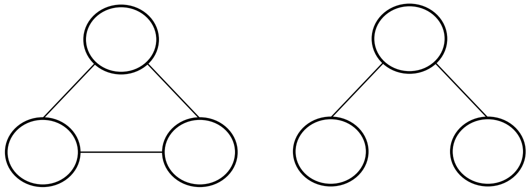

```
bool globalClusteringCoefficient(int bound)
{
  DataGraph G("path/to/graph/");
  auto triplet = PatternGenerator::star(3);
  int numTriplets = count(G, {triplet});
  auto countAndCheck = [=](auto &&match, auto &&aggregator)
  {
    int numTriangles = aggregator.readValue(match.pattern);
    if (3*numTriangles/numTriplets > bound) aggregator.stop();
    else aggregator.map(match.pattern, 1);
  }
  auto triangle = PatternGenerator::clique(3);
  auto result = match<Pattern, int>(G, triangle, countAndCheck);
  return 3*result[triangle]/numTriplets > bound;
}
```

Early Termination



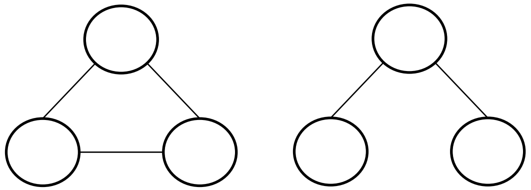

```
bool globalClusteringCoefficient(int bound)
{
  DataGraph G("path/to/graph/");
  auto triplet = PatternGenerator::star(3);
  int numTriplets = count(G, {triplet});
  auto countAndCheck = [=](auto &&match, auto &&aggregator)
  {
    int numTriangles = aggregator.readValue(match.pattern);
    if (3*numTriangles/numTriplets > bound) aggregator.stop();
    else aggregator.map(match.pattern, 1);
  }
  auto triangle = PatternGenerator::clique(3);
  auto result = match<Pattern, int>(G, triangle, countAndCheck);
  return 3*result[triangle]/numTriplets > bound;
}
```

Early Termination




```
bool globalClusteringCoefficient(int bound)
{
  DataGraph G("path/to/graph/");
  auto triplet = PatternGenerator::star(3);
  int numTriplets = count(G, {triplet});
  auto countAndCheck = [=](auto &&match, auto &&aggregator)
  {
    int numTriangles = aggregator.readValue(match.pattern);
    if (3*numTriangles/numTriplets > bound) aggregator.stop();
    else aggregator.map(match.pattern, 1);
  }
  auto triangle = PatternGenerator::clique(3);
  auto result = match<Pattern, int>(G, triangle, countAndCheck);
  return 3*result[triangle]/numTriplets > bound;
}
```

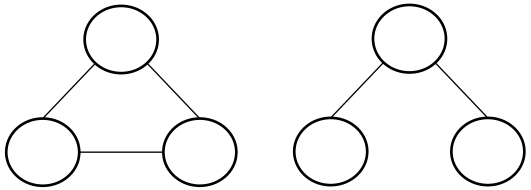
Early Termination



Triangle Triplet

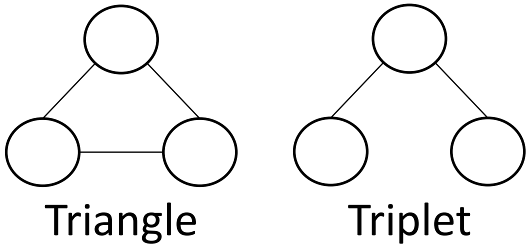
```
bool globalClusteringCoefficient(int bound)
{
  DataGraph G("path/to/graph/");
  auto triplet = PatternGenerator::star(3);
  int numTriplets = count(G, {triplet});
  auto countAndCheck = [=](auto &&match, auto &&aggregator)
  {
    int numTriangles = aggregator.readValue(match.pattern);
    if (3*numTriangles/numTriplets > bound) aggregator.stop();
    else aggregator.map(match.pattern, 1);
  }
  auto triangle = PatternGenerator::clique(3);
  auto result = match<Pattern, int>(G, triangle, countAndCheck);
  return 3*result[triangle]/numTriplets > bound;
}
```

Early Termination



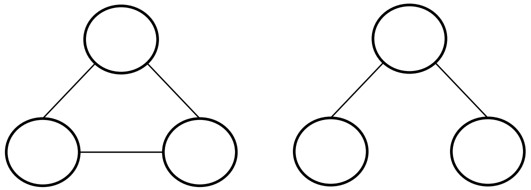

```
bool globalClusteringCoefficient(int bound)
{
  DataGraph G(“path/to/graph/”);
  auto triplet = PatternGenerator::star(3);
  int numTriplets = count(G, {triplet});
  auto countAndCheck = [=](auto &&match, auto &&aggregator)
  {
    int numTriangles = aggregator.readValue(match.pattern);
    if (3*numTriangles/numTriplets > bound) aggregator.stop();
    else aggregator.map(match.pattern, 1);
  }
  auto triangle = PatternGenerator::clique(3);
  auto result = match<Pattern, int>(G, triangle, countAndCheck);
  return 3*result[triangle]/numTriplets > bound;
}
```

Early Termination




```
bool globalClusteringCoefficient(int bound)
{
  DataGraph G("path/to/graph/");
  auto triplet = PatternGenerator::star(3);
  int numTriplets = count(G, {triplet});
  auto countAndCheck = [=](auto &&match, auto &&aggregator)
  {
    int numTriangles = aggregator.readValue(match.pattern);
    if (3*numTriangles/numTriplets > bound) aggregator.stop();
    else aggregator.map(match.pattern, 1);
  }
  auto triangle = PatternGenerator::clique(3);
  auto result = match<Pattern, int>(G, triangle, countAndCheck);
  return 3*result[triangle]/numTriplets > bound;
}
```

Early Termination




```
bool globalClusteringCoefficient(int bound)
{
  DataGraph G("path/to/graph/");
  auto triplet = PatternGenerator::star(3);
  int numTriplets = count(G, {triplet});
  auto countAndCheck = [=](auto &&match, auto &&aggregator)
  {
    int numTriangles = aggregator.readValue(match.pattern);
    if (3*numTriangles/numTriplets > bound) aggregator.stop();
    else aggregator.map(match.pattern, 1);
  }
  auto triangle = PatternGenerator::clique(3);
  auto result = match<Pattern, int>(G, triangle, countAndCheck);
  return 3*result[triangle]/numTriplets > bound;
}
```

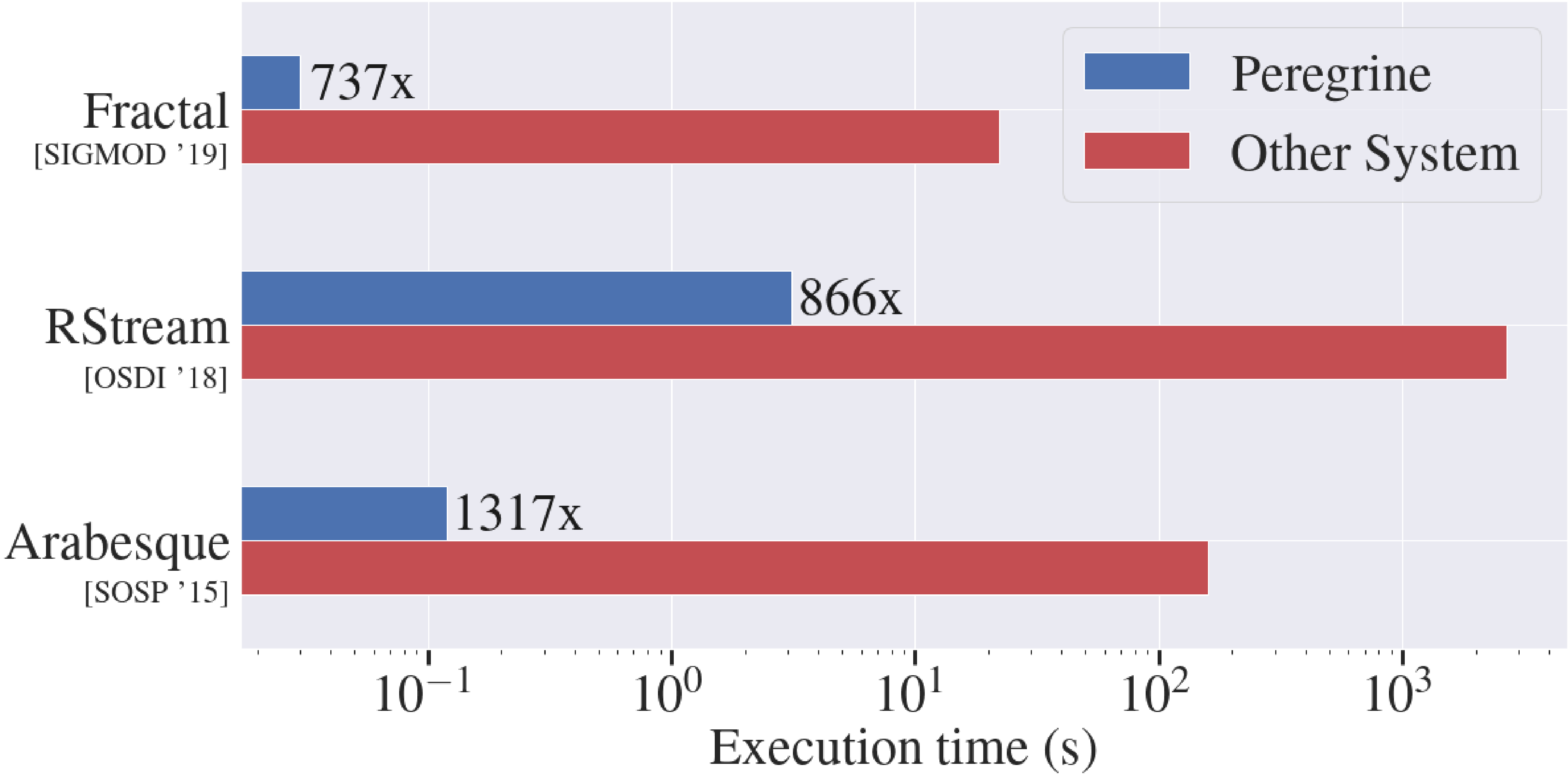
Early Termination

```
bool globalClusteringCoefficient(int bound)
{
  DataGraph G("path/to/graph/");
  auto triplet = PatternGenerator::star(3);
  int numTriplets = count(G, {triplet});
  auto countAndCheck = [=](auto &&match, auto &&aggregator)
  {
    int numTriangles = aggregator.readValue(match.pattern);
    if (3*numTriangles/numTriplets > bound) aggregator.stop();
    else aggregator.map(match.pattern, 1);
  }
  auto triangle = PatternGenerator::clique(3);
  auto result = match<Pattern, int>(G, triangle, countAndCheck);
  return 3*result[triangle]/numTriplets > bound;
}
```

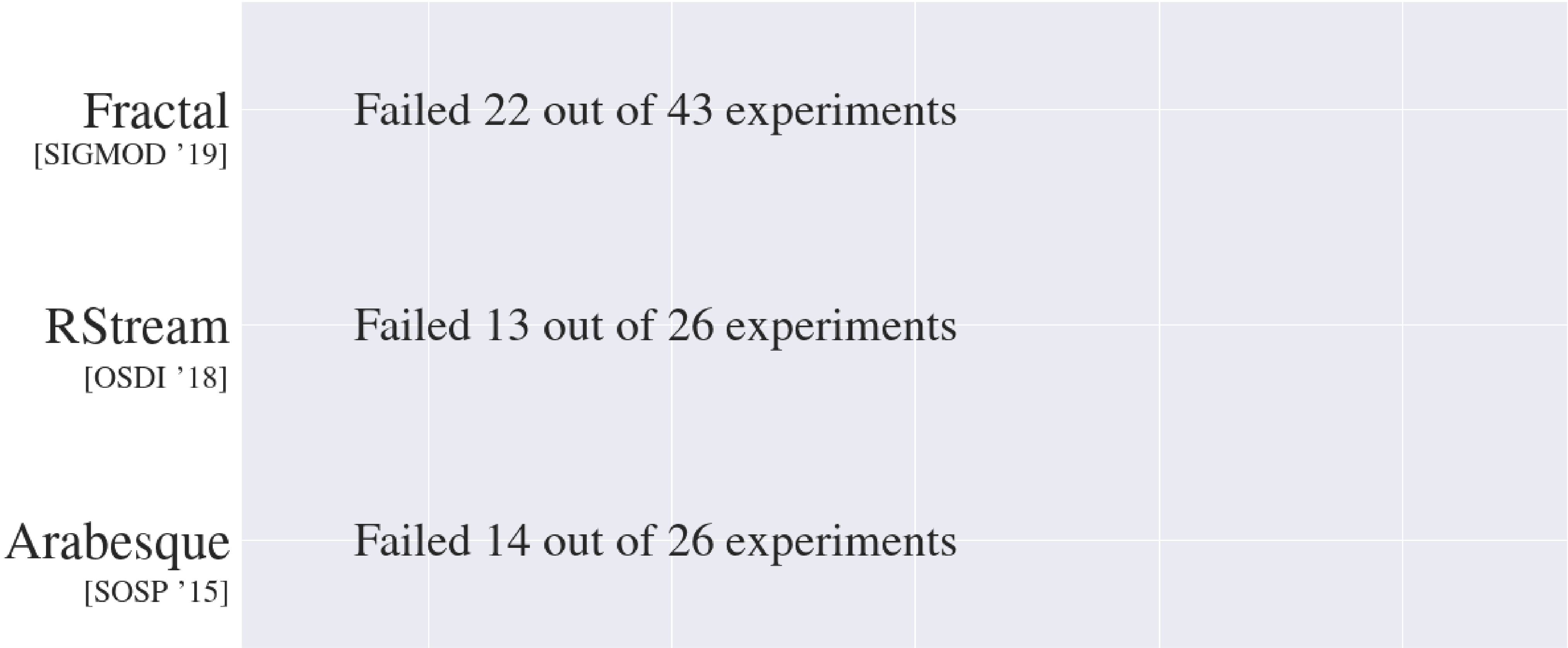
Comparison with Existing Work

- Peregrine with 16 logical cores and 32GB RAM
- Arabesque & Fractal with 8x16 logical cores and 8x32GB RAM
- RStream with 96 logical cores and 192GB RAM

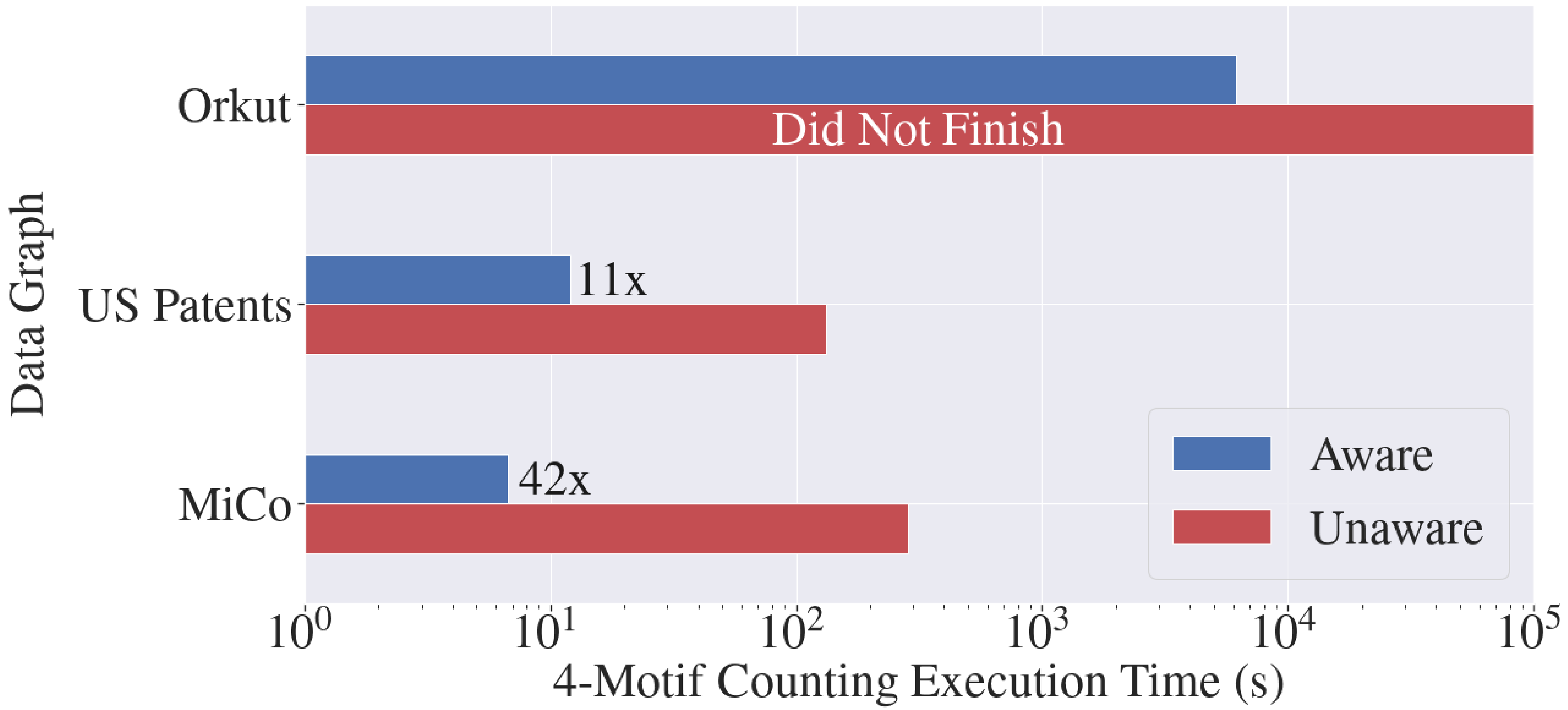
Comparison with Existing Work



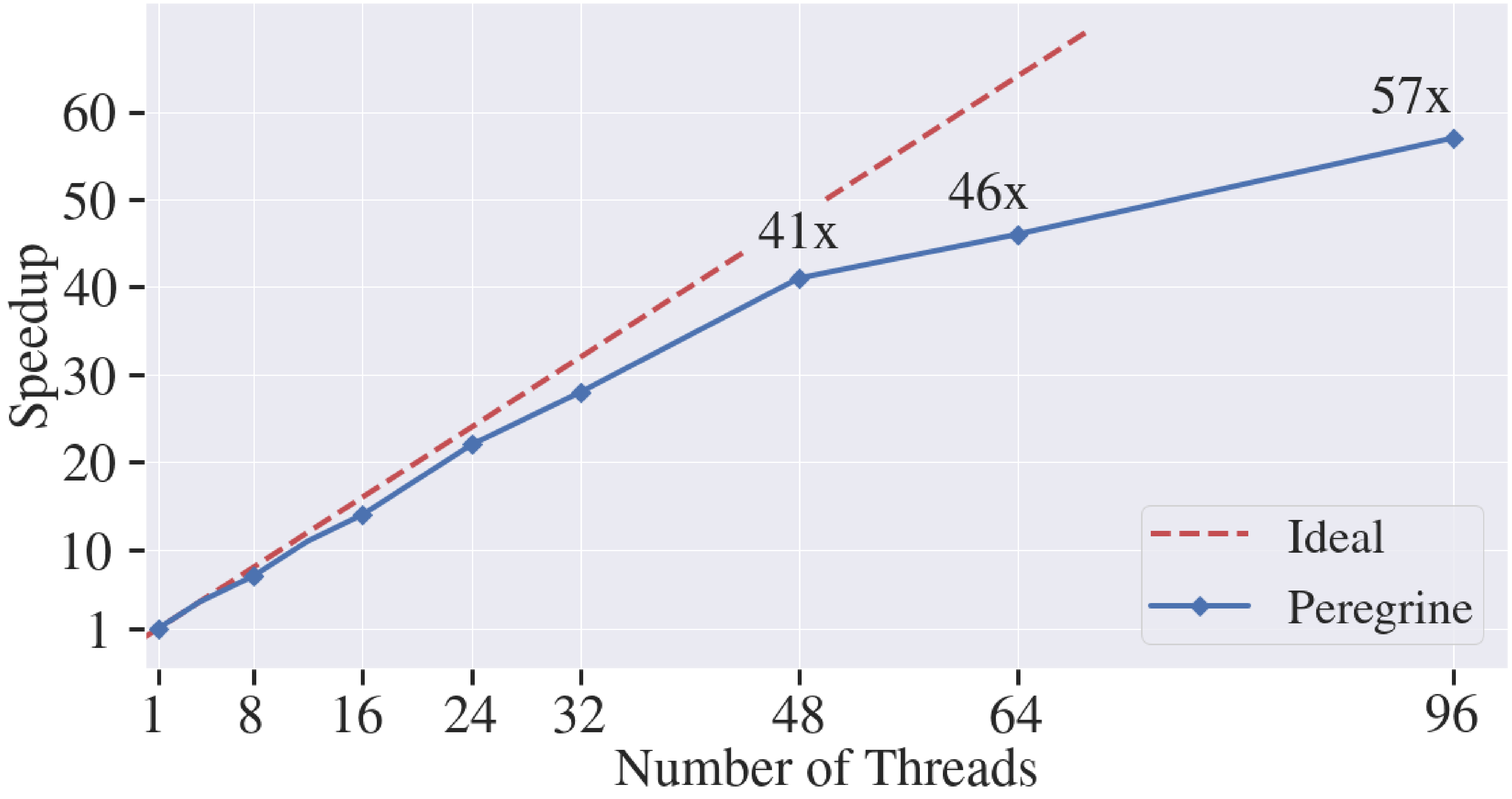
Comparison with Existing Work



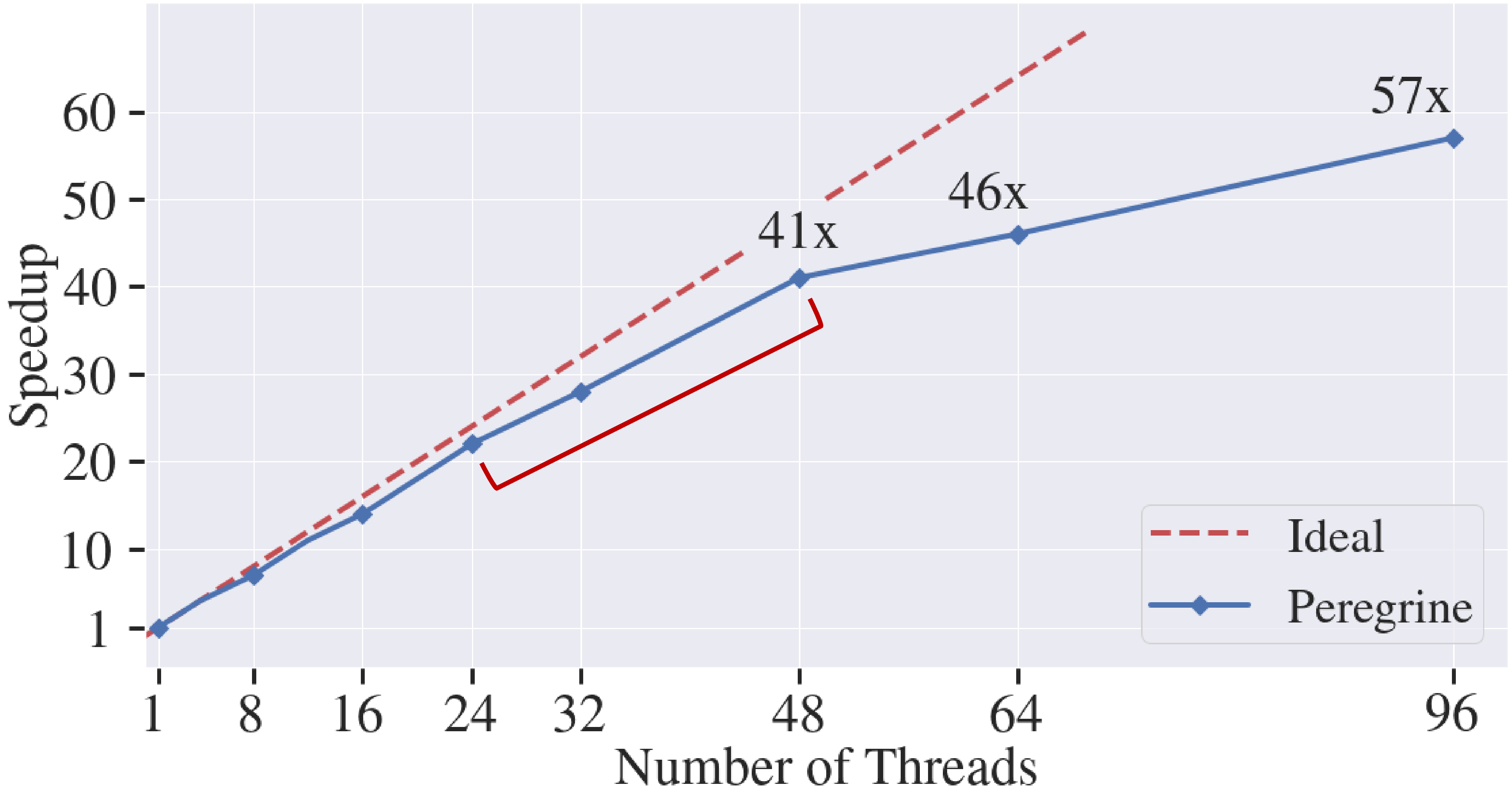
Effects of Pattern Awareness



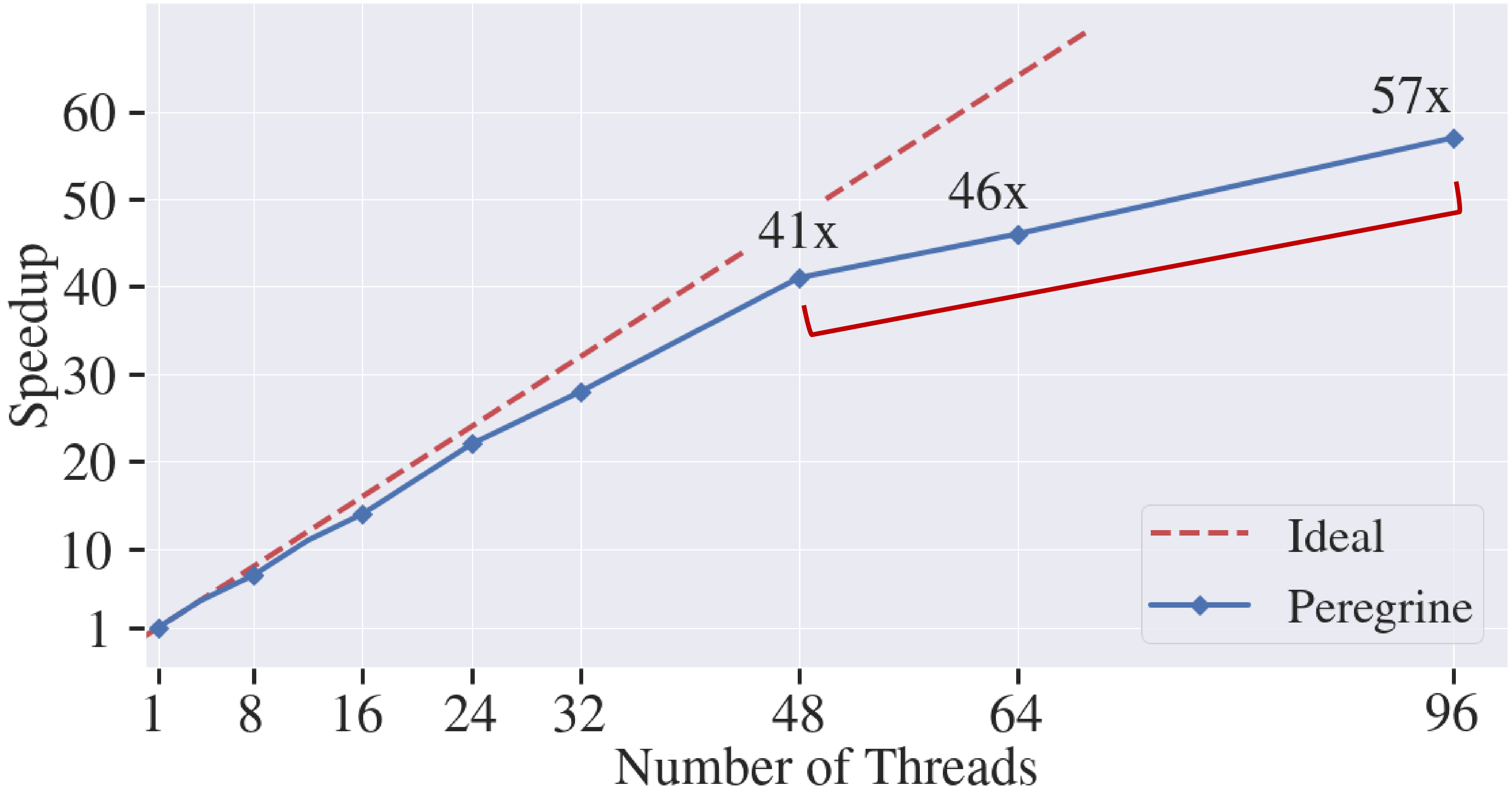
Scalability



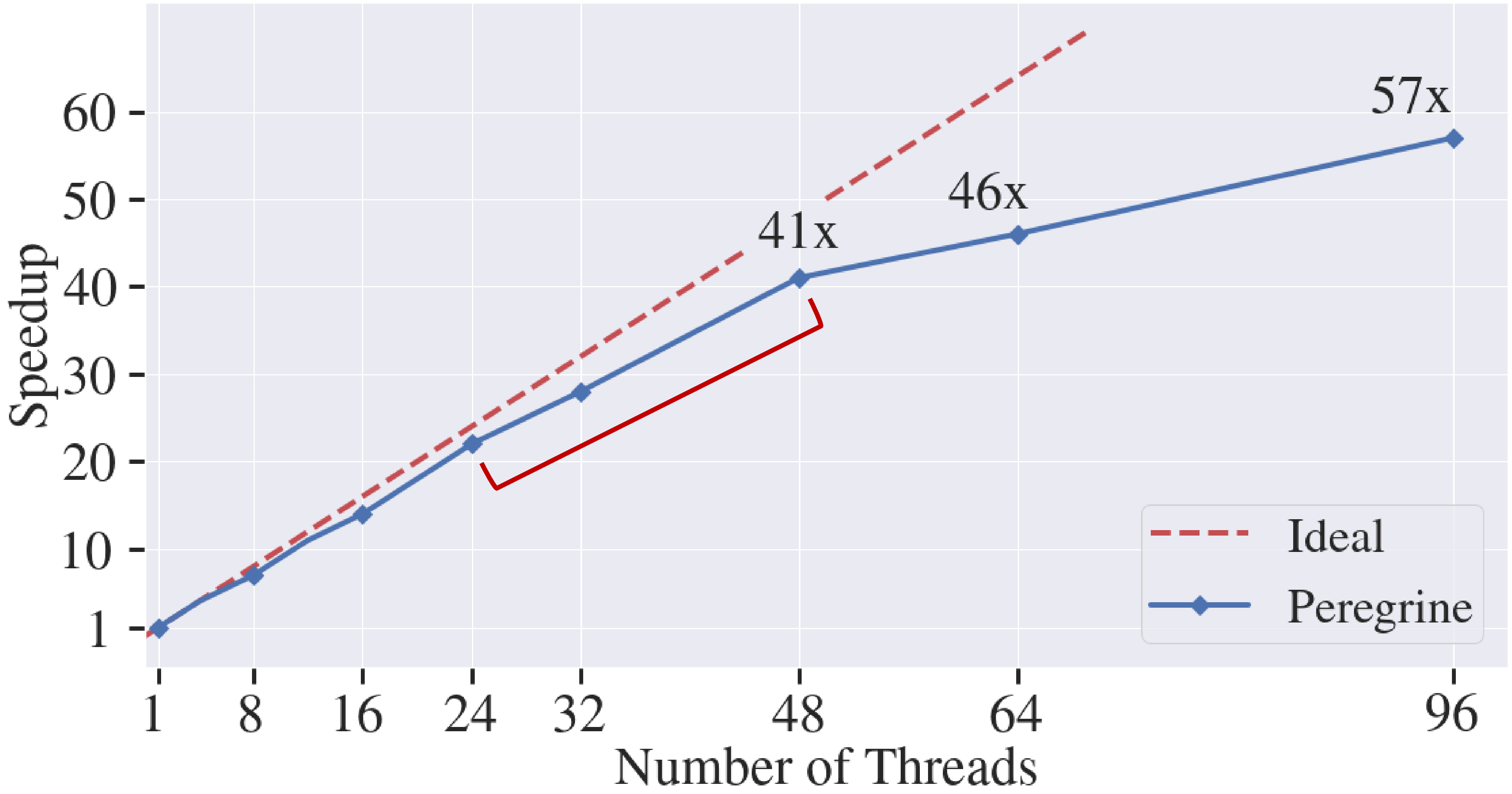
Scalability



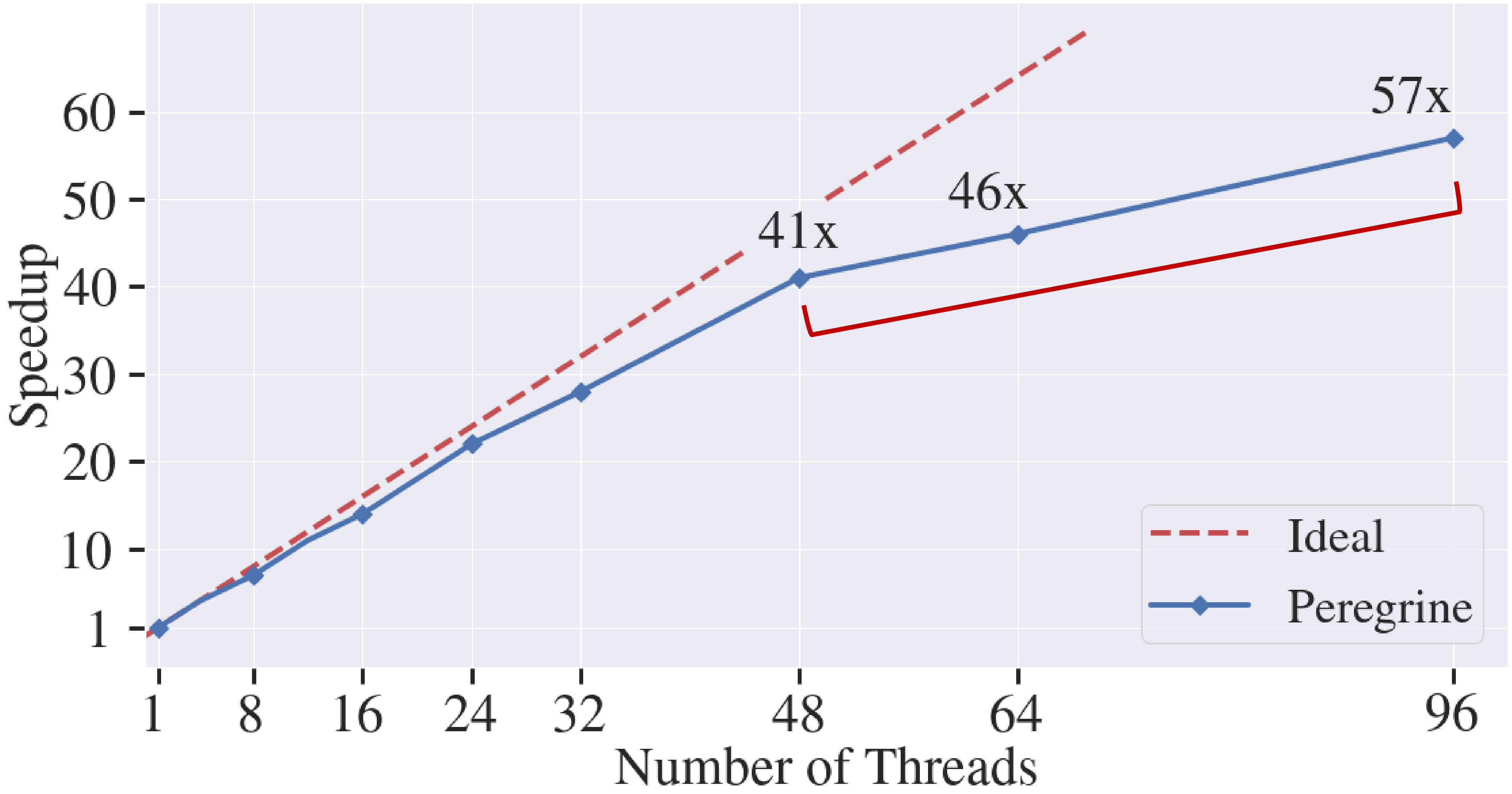
Scalability



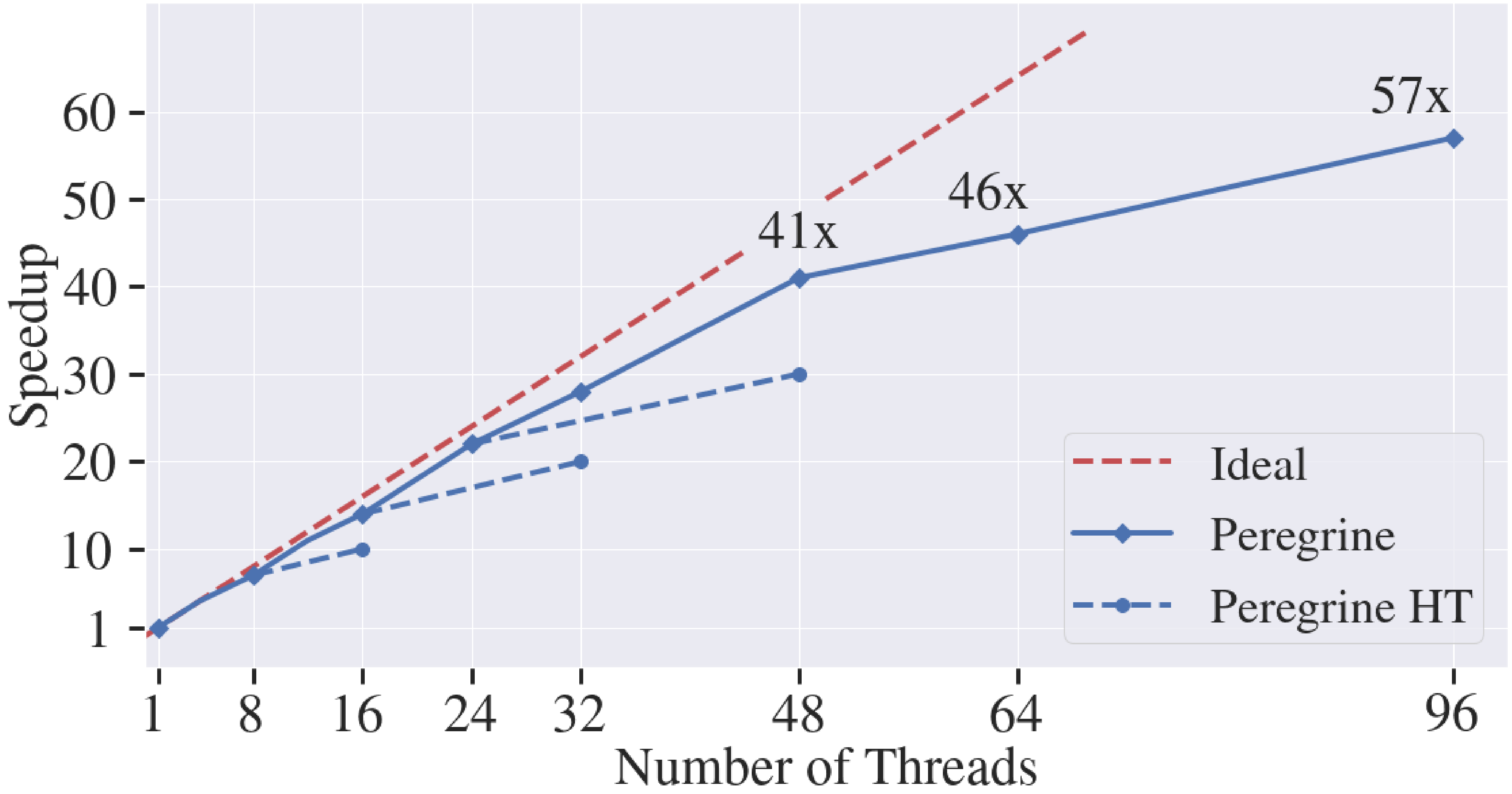
Scalability



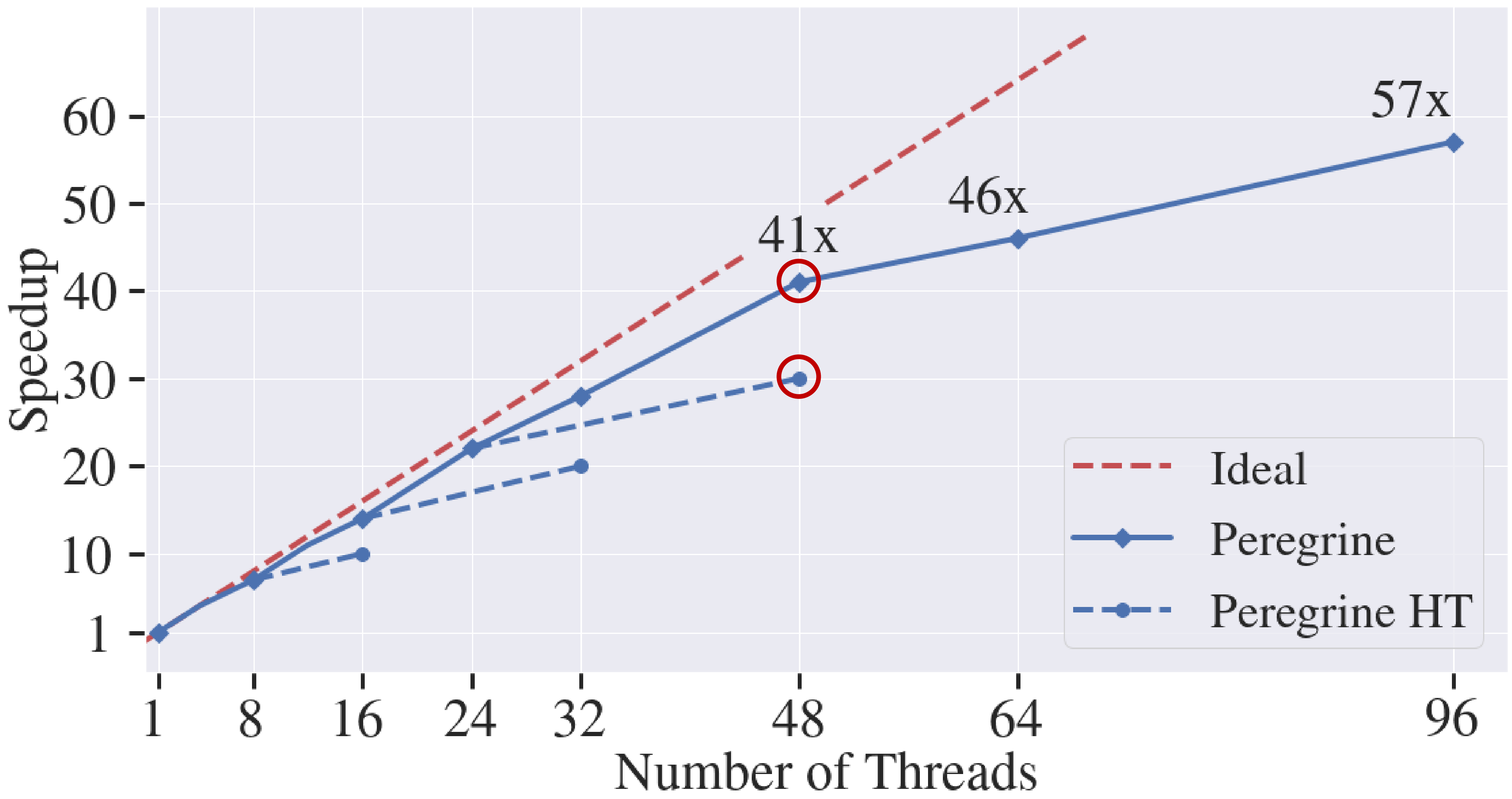
Scalability



Scalability



Scalability



PEREGRINE

Pattern-aware programming and processing models

PEREGRINE

Pattern-aware programming and processing models

- Shift abstraction from subgraph to pattern

PEREGRINE

Pattern-aware programming and processing models

- Shift abstraction from subgraph to pattern
- User program is transparent to the system

PEREGRINE

Pattern-aware programming and processing models

- Shift abstraction from subgraph to pattern
- User program is transparent to the system
- Up to 42x faster than pattern-unaware
- Up to 737x faster than state-of-the-art

PEREGRINE

Pattern-aware programming and processing models

- Shift abstraction from subgraph to pattern
- User program is transparent to the system
- Up to 42x faster than pattern-unaware
- Up to 737x faster than state-of-the-art

https://github.com/pdclab/peregrine