Fast Graph Pattern Mining Made Easy

Xuhao Chen

Massachusetts Institute of Technology

CSAIL

Outline

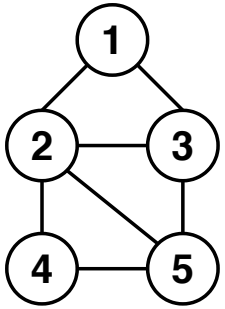
- Graph Pattern Mining (GPM) Use Cases
- Pangolin Overview 
- Pangolin System Internals
  - How to Achieve Easy Programming?
  - How to Achieve High Performance?
- Evaluation
- Sandslash (optional) 

Graph Pattern Mining (GPM)

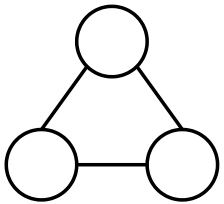
- **Inputs**: a graph G and a set of patterns $Q = \{P_1, P_2, \ldots, Pn\}$
- **Goal**: count or list all subgraphs in G that matches $P_i \in Q$

We only consider connected patterns

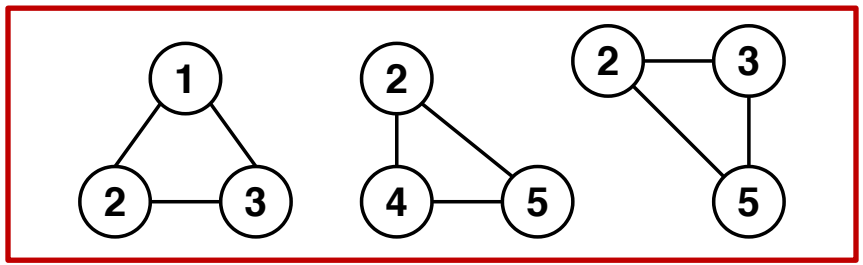
Input Graph G



Pattern P



Matched subgraphs

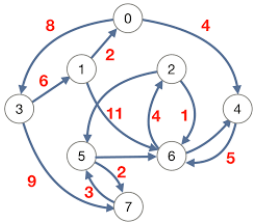


Graph Analytics vs. Graph Pattern Mining

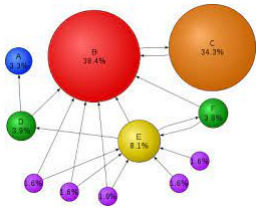
Graph Analytics

Updates vertex/edge labels of the input graph

Shortest Path



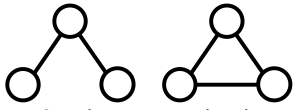
PageRank



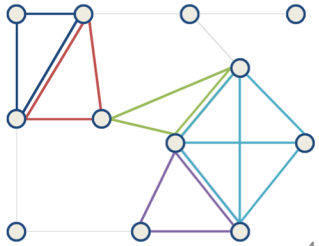
Graph Pattern Mining

Discovers structural patterns in the input graph

Motif Counting



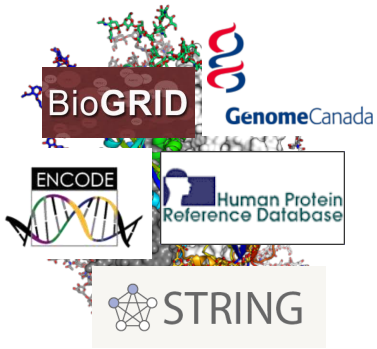
Clique Listing



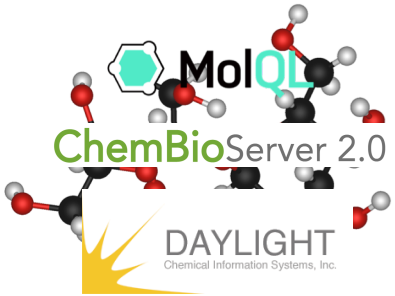
Why Graph Pattern Mining? (1/3)

Social Networks

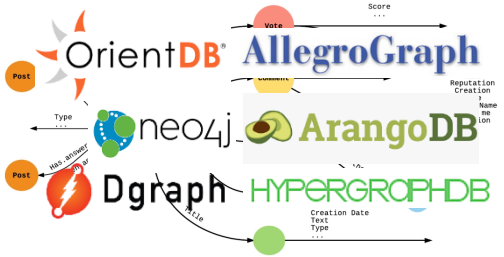
Recommender System



Bio-medicine



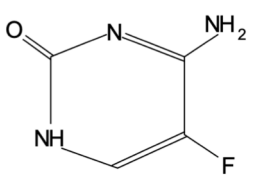
Chemical Engineering



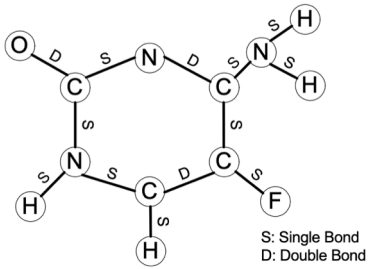
Graph Database

Why Graph Pattern Mining? (2/3)

- Drug discovery



(a) NSC 103025 Flucytosine



(b) Graph Representation

Chemical and Graphical representation of Flucytosine

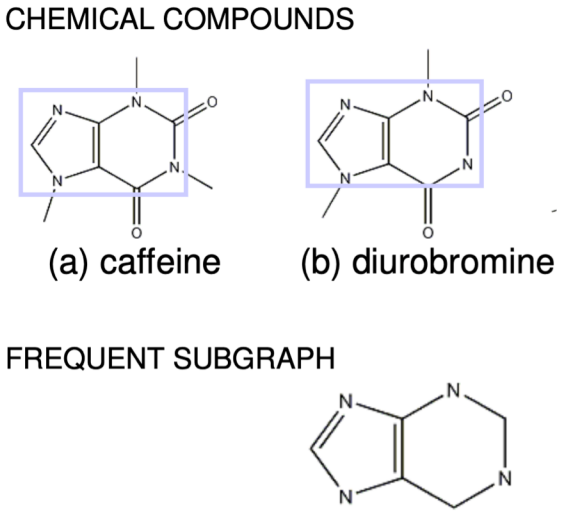
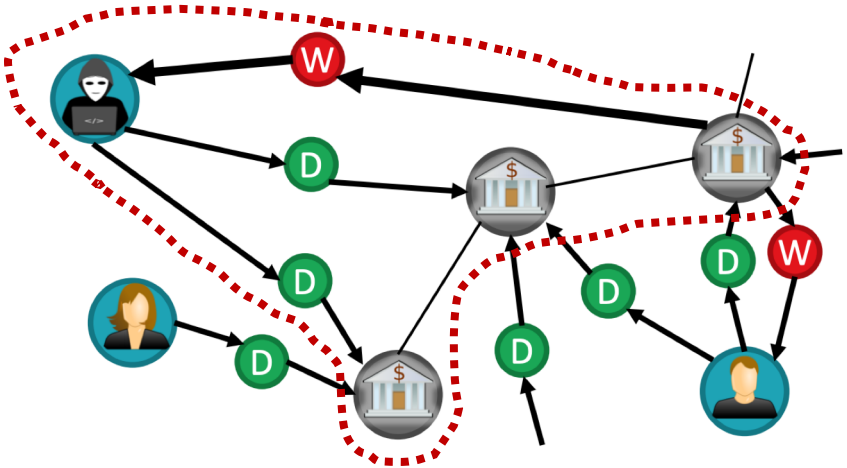


Figure modified from K. Borgwardt and X. Yan (KDD'08)

Why Graph Pattern Mining? (3/3)



Transactions and involved entities

Small deposits followed by large withdrawal

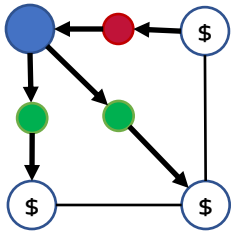


Figure modified from Anand Padmanabha Iyer, et.al., ASAP: Fast, Approximate Graph Pattern Mining at Scale (OSDI'18)

Outline

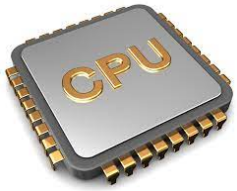
- Graph Pattern Mining (GPM) Use Cases
- **Pangolin Overview**
- Pangolin System Internals
  - How to Achieve Easy Programming?
  - How to Achieve High Performance?
- Evaluation

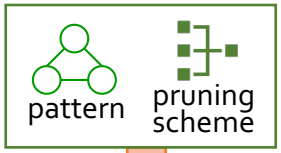
Pangolin: Efficient & Productive GPM Programming

OpenMP

NVIDIA CUDA

Too Difficult!





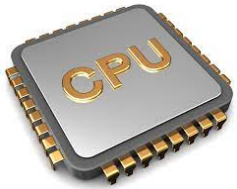
Pangolin API

Pangolin Runtime System
(transparent parallelism)

Sequential!

easy to code

run fast



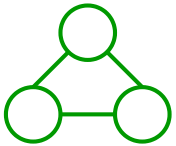
Easy Programming in Pangolin

- k-Clique Listing

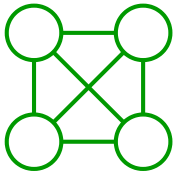
```
1  bool toExtend(Subgraph sg, Vertex v) {
2    return (sg.getLastVertex() == v);
3  }

4  bool toAdd(Subgraph sg, Vertex u) {
5    for v ∈ sg.getVertices()except last:
6      if (!isConnected(v,u)) return false;
7    return true;
8  }
```

Pangolin API
user defined functions



3-clique
i.e. triangle



4-clique

...

Hand Written	Pangolin
State-of-the-art: (KClist: WWW'18) **395 Lines of Code !!!**	**8 Lines of code !!!**

Outline

- Graph Pattern Mining (GPM) Use Cases
- Pangolin Overview
- **Pangolin System Internals**
  - **How to Achieve Easy Programming?**
  - **How to Achieve High Performance?**
- Evaluation

Graph Mining Challenges

Easy Programming

High Performance

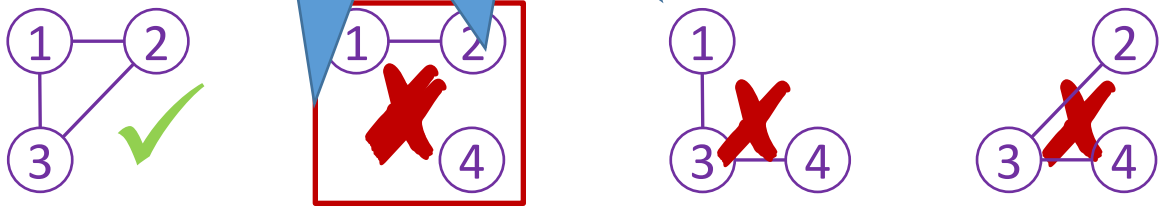
A "Naive" Abstraction

- Enumerate all the possible verte combinations!!!
- G has n vertic P has erti $\binom{n}{\cdot}$

Graphs are sparse!

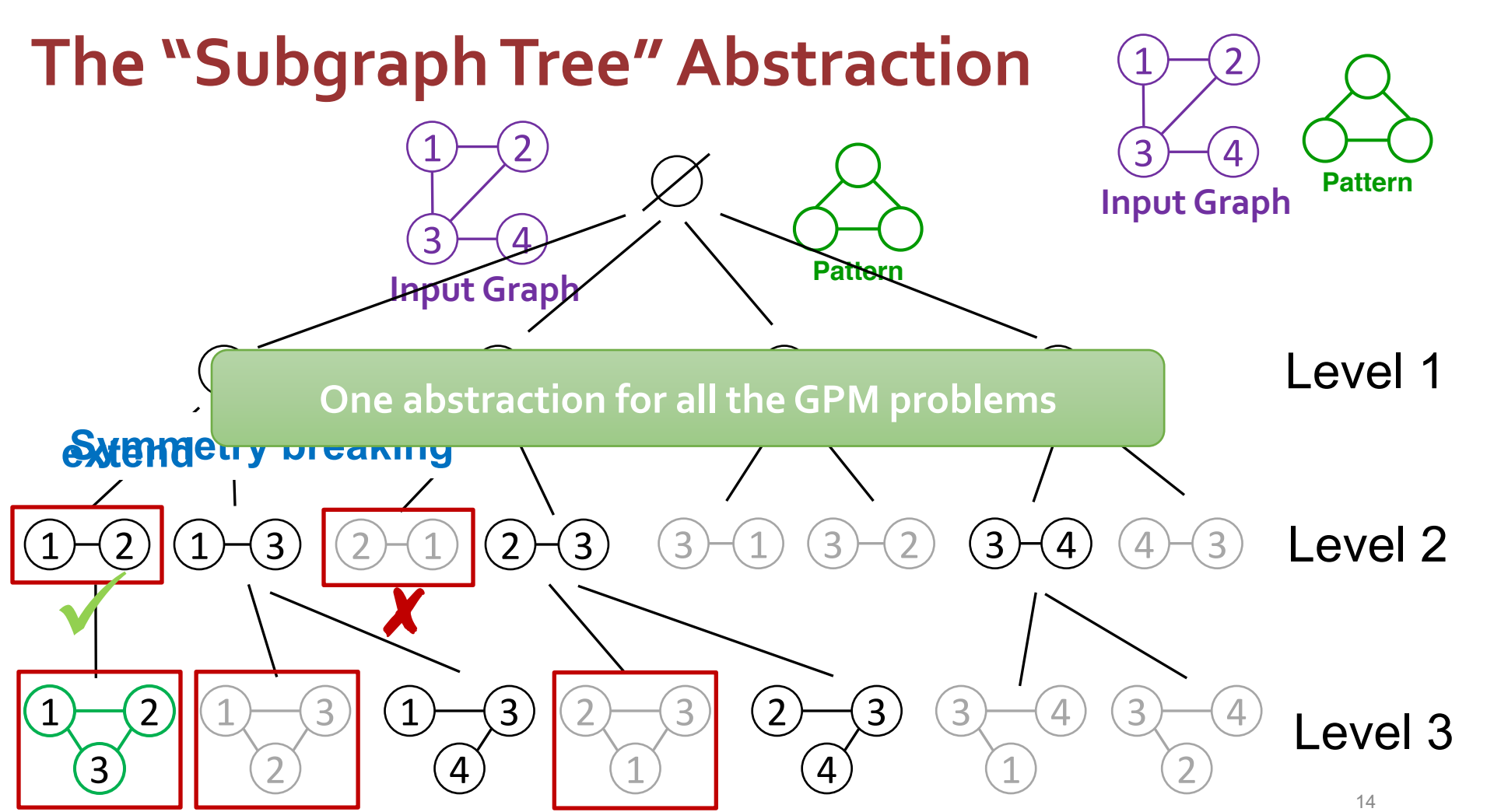
We only consider connected patterns

$\binom{3}{4}=4$: <1,2,3 1, 4> <1,3,4> <2,3,4>



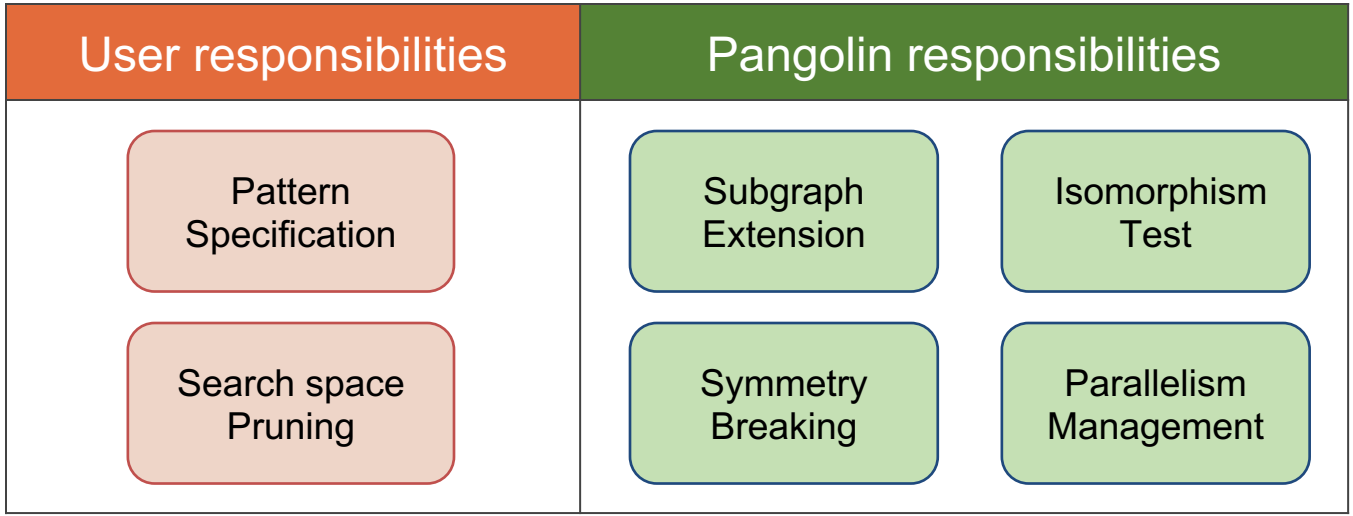
Isomorphism test

The "Subgraph Tree" Abstraction

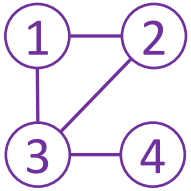


Work Division between User & System

- Pangolin does a lot for you!

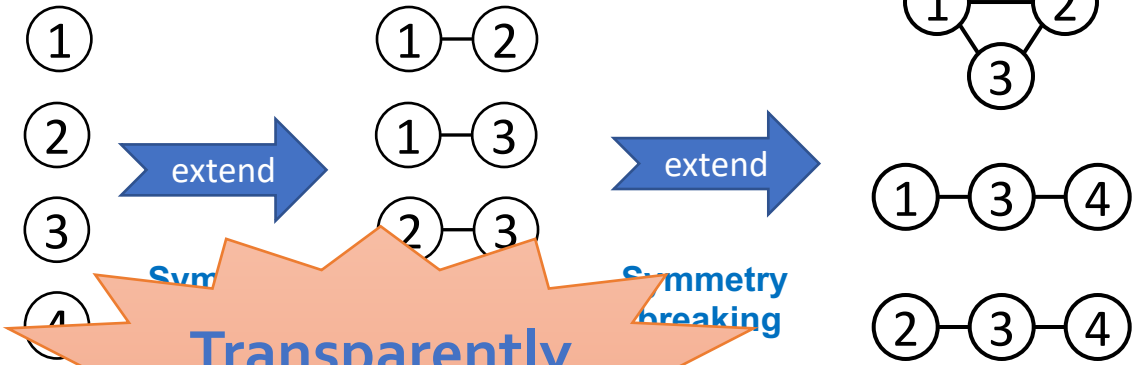


Work Done By Pangolin



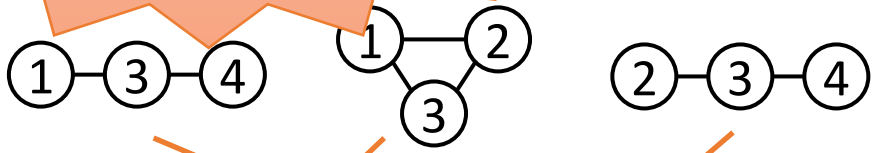
Input Graph

Extend Stage



Transparently parallel!

Subgraphs



Reduce Stage

Isomorphism test

Pattern Map

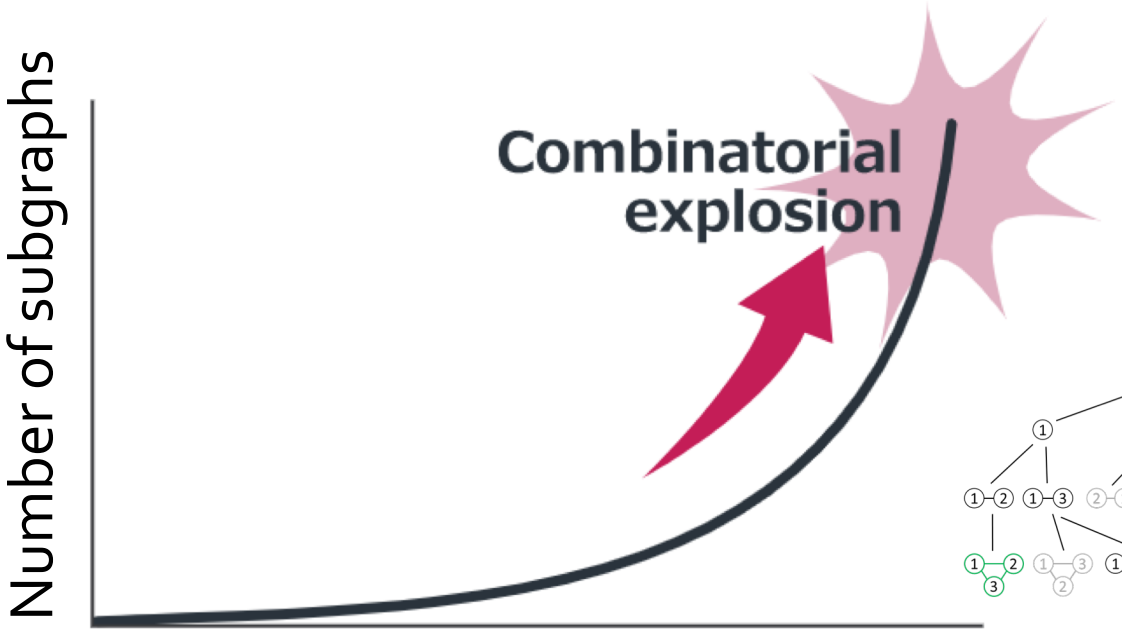
Graph Mining Challenges

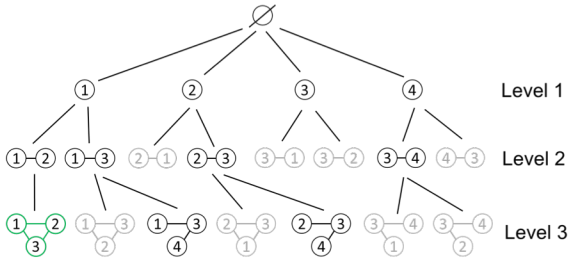
Easy Programming

- High level abstraction
  - subgraph search tree
- Work Division
  - Simple API
- Transparent parallelism
  - code in sequential, not in parallel

High Performance

Search Space Pruning

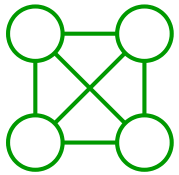


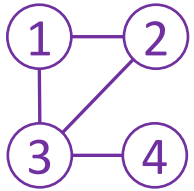


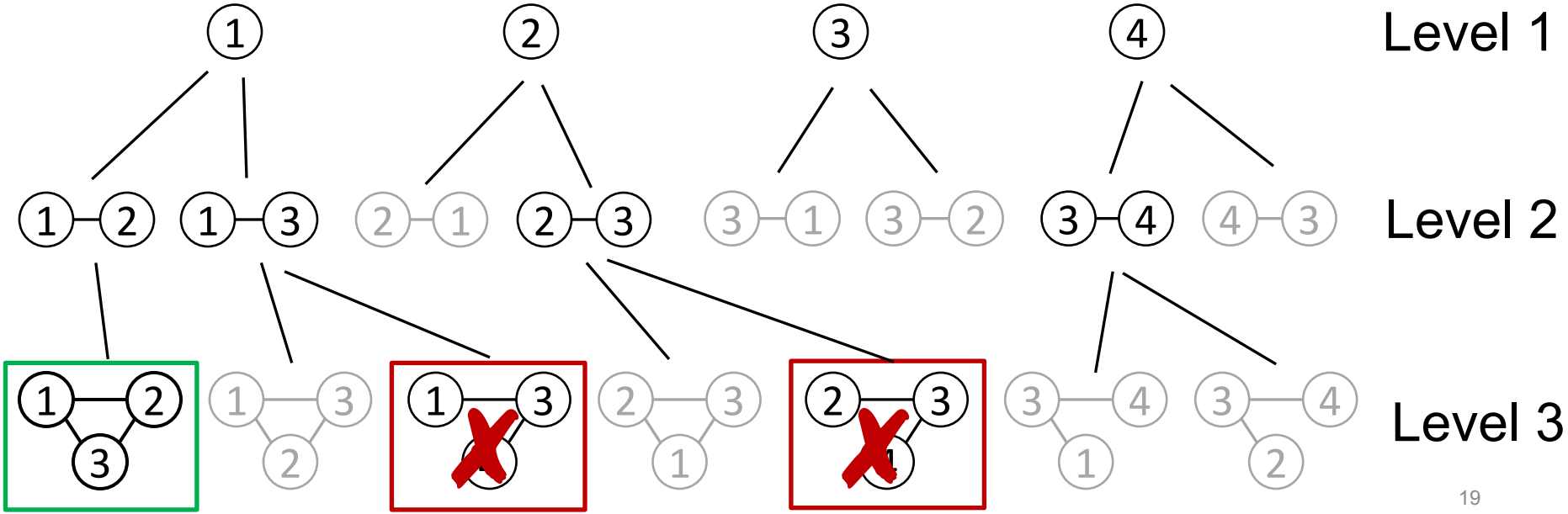
search space pruning is the key for GPM performance

Pattern-aware Search Space Pruning

Problem: 4-Clique Listing

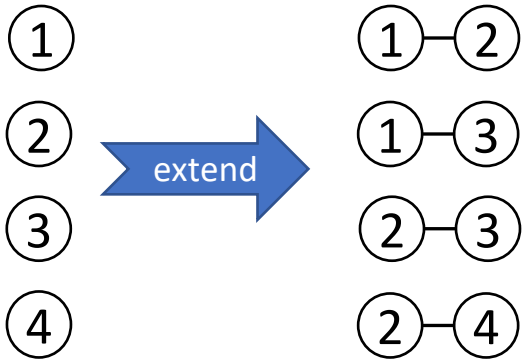






Exposing Interface for User Defined Pruning

Extend Stage



Input subgraph list

Output subgraph list

```
procedure Extend(SubgraphList in, SubgraphList out) {
  for each subgraph sg ∈ in
    for each vertex v ∈ sg
      if (toExtend(sg,v) == true)
        for each vertex u ∈ adj(v)
          if (toAdd(sg,u) == true)
            out.insert(sg ∪ u)
}
```

Two of the user defined functions*:

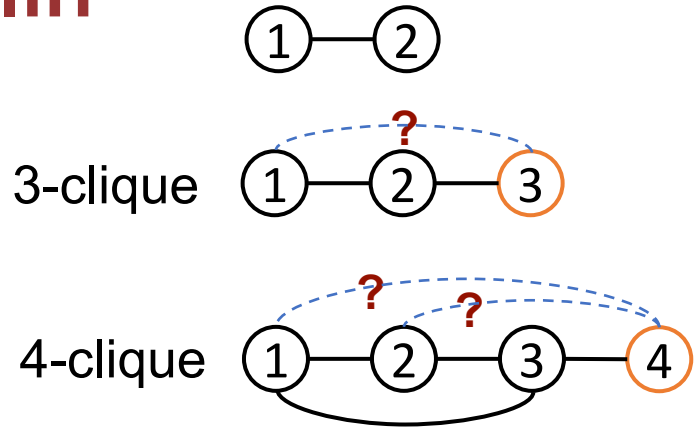
```
bool toExtend(Subgraph sg, Vertex v) {
    return true;
}

bool toAdd(Subgraph sg, Vertex u) {
    return true;
}
```

* The full list of API functions can be found in the Pangolin paper: Chen et. al, VLDB 2020

Easy Programming in Pangolin

- k-Clique Listing
  - Extend only the last vertex v
  - Check if new vertex u is connected to all previous vertices




```
bool toExtend(Subgraph sg, Vertex v) {
  return (sg.getLastVertex() == v);
}
```

```
bool toAdd(Subgraph sg, Vertex u) {
  for v ∈ sg.getVertices()except last:
    if (!isConnected(v,u)) return false;
  return true;
}
```

Hand Written	Pangolin
State-of-the-art: (KClist: WWW'18) 395 Lines of Code !!!	8 Lines of code !!!

Graph Mining Challenges

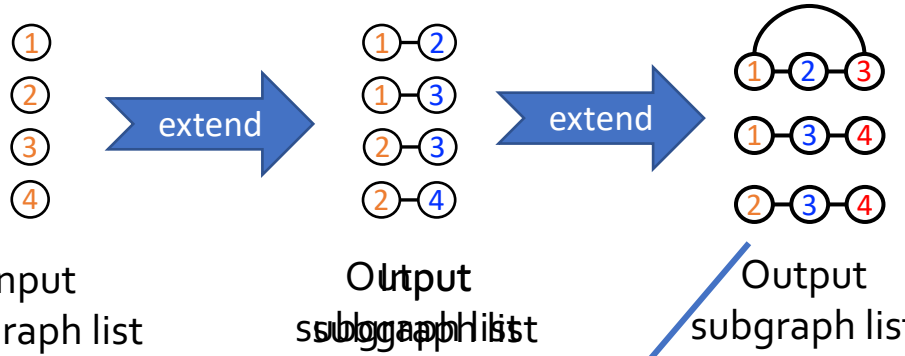
Easy Programming

High Performance

- Search space pruning
- Architectural Optimizations

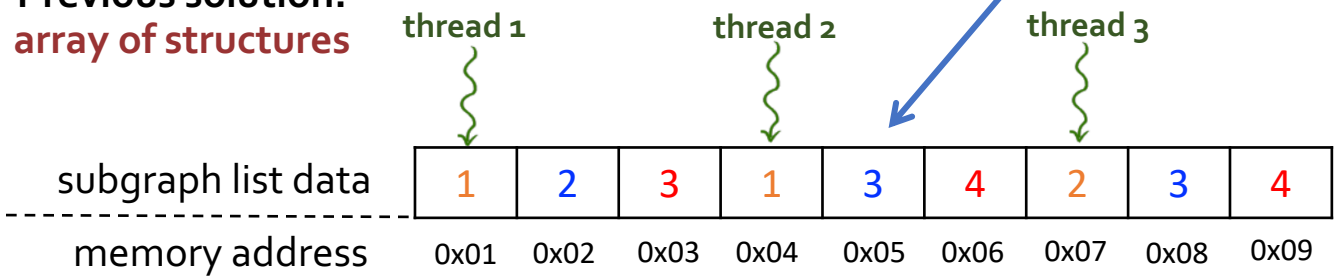
Unlocking GPU Horsepower for GPM (1/3)

Extend Stage



subgraph list is the key data structure

Previous solution:
array of structures

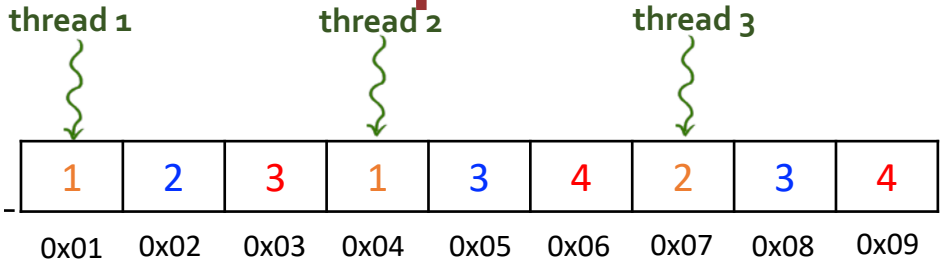


Uncoalesced memory accesses!!!

Not GPU friendly!

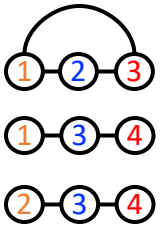
Unlocking GPU Horsepower for GPM (2/3)

Previous solution:
array of structures



→ **Not GPU friendly!**

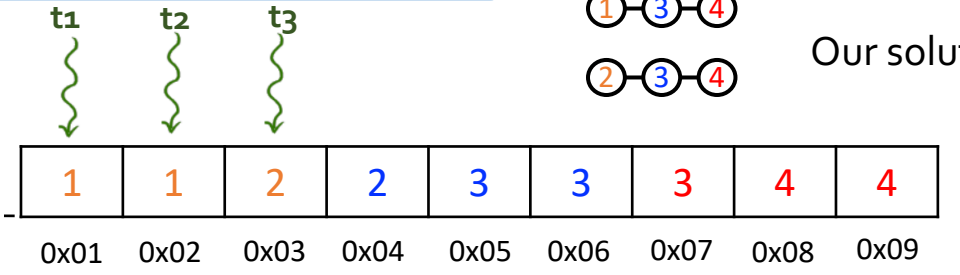
Key idea: Re-structure data layout



Previous solution: **one by one** ✗

Our solution**: level by level** ✓

Our solution:
structures of arrays



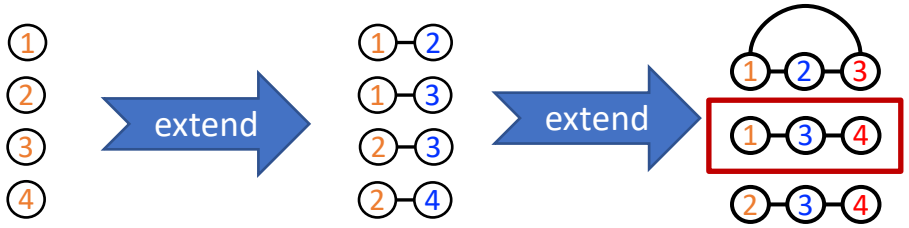
GPU friendly!

Coalesced memory accesses!!!

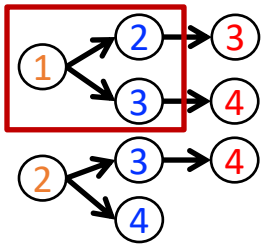
→

Unlocking GPU Horsepower for GPM (3/3)

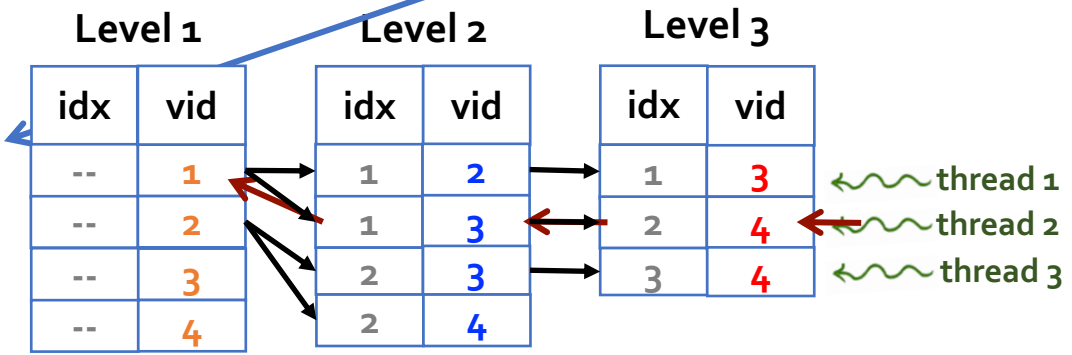
Extend Stage



Real implementation:
Build a **prefix tree** level by level



Prefix tree



GPU friendly!

Coalesced memory accesses!!!

Graph Mining Challenges

Easy Programming

High Performance

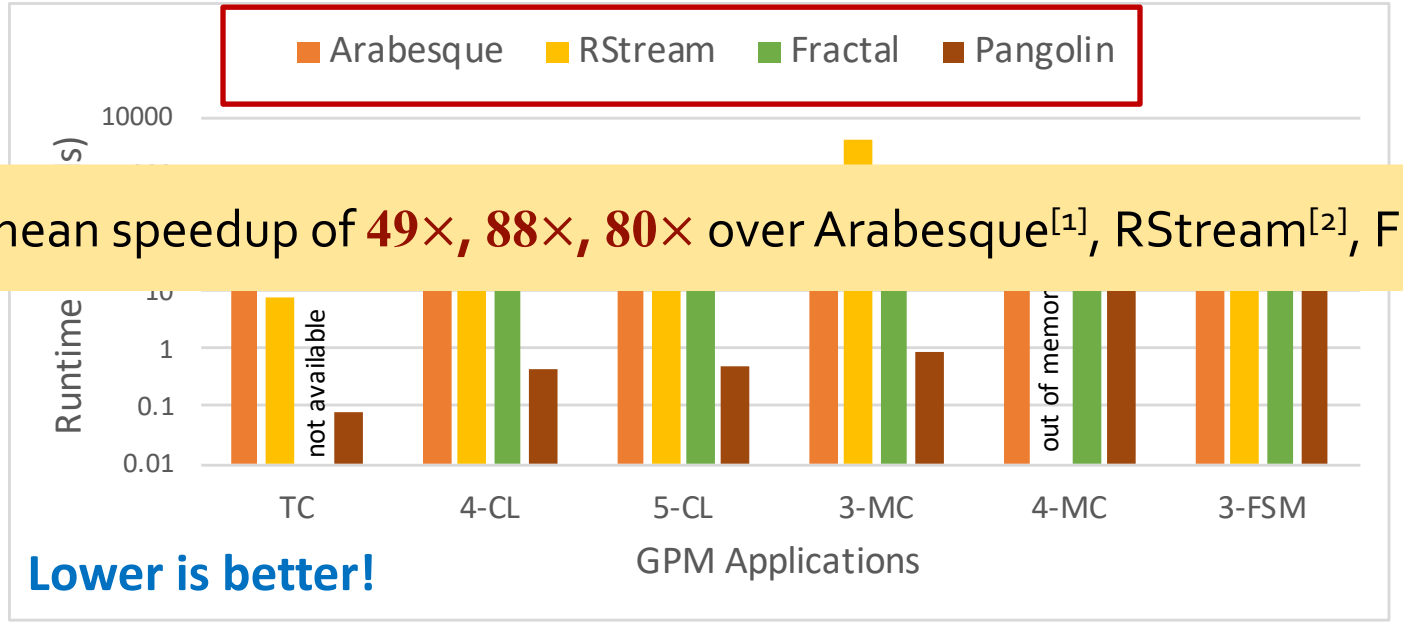
Search space pruning

Architectural Optimizations

Outline

- Graph Pattern Mining (GPM) Use Cases
- Pangolin Overview
- Pangolin System Internals
  - How to Achieve Easy Programming?
  - How to Achieve High Performance?
- **Evaluation**

Comparing with Existing Systems on CPU



Geomean speedup of **49×, 88×, 80×** over Arabesque[1], RStream[2], Fractal[3]

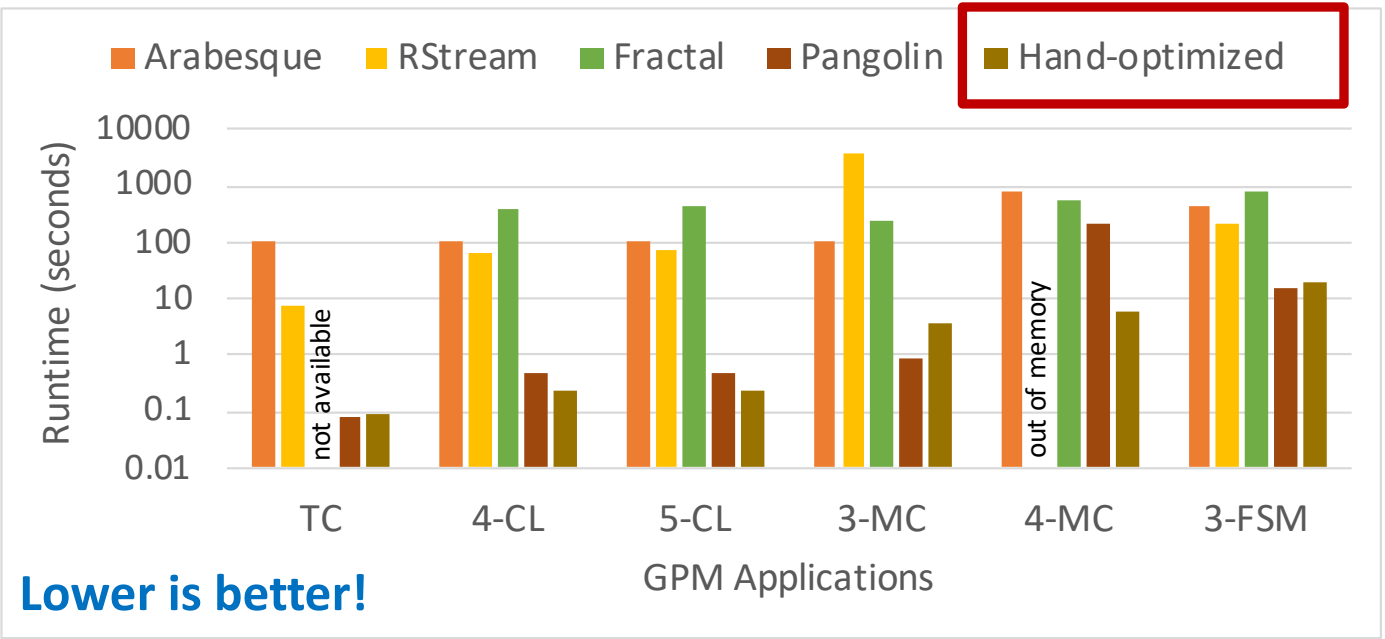
Input graph: `Patent-citations`

TC: Triangle Counting
CL: Clique Listing
MC: Motif Counting
FSM: Frequent Subgraph Mining

Intel Xeon Gold 5120 CPU (2.2GHz) 2 sockets (14 cores each), 190GB memory, and 3TB SSD

[1] Arabesque SOSP'15 [2] RStream OSDI'18 [3] Fractal SIGMOD'19

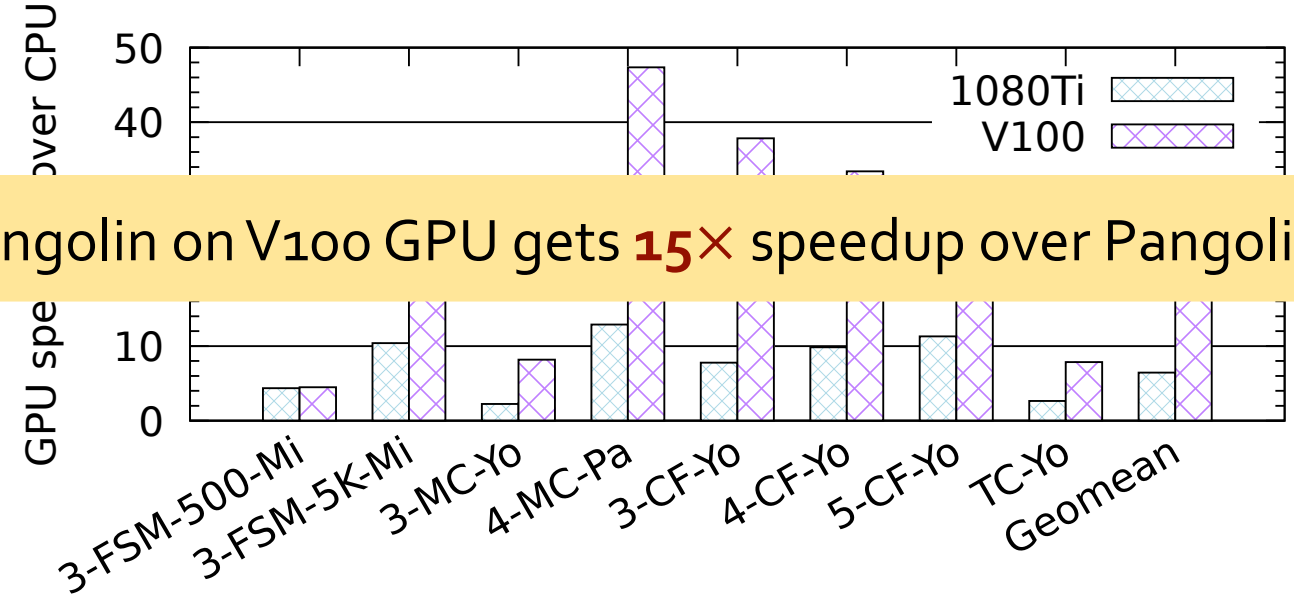
Comparing with Hand-written Code on CPU



Input graph: `Patent-citations`

TC: Triangle Counting
CL: Clique Listing
MC: Motif Counting
FSM: Frequent Subgraph Mining

Pangolin GPU Speedup over Pangolin CPU



Pangolin on V100 GPU gets **15×** speedup over Pangolin CPU

- **Speedup of Pangolin on GPU over Pangolin on 28-thread CPU**

TC: Triangle Counting
CL: Clique Listing
MC: Motif Counting
FSM: Frequent Subgraph Mining

Mi: Mico
Pa: Patent-citations
Yo: Youtube

Summary: GPM System Design

Easy Programming

Easy coding!

- High level abstraction
- Work Division
- Transparent parallelism

100× ~ 1000× speedup!!!

High Performance

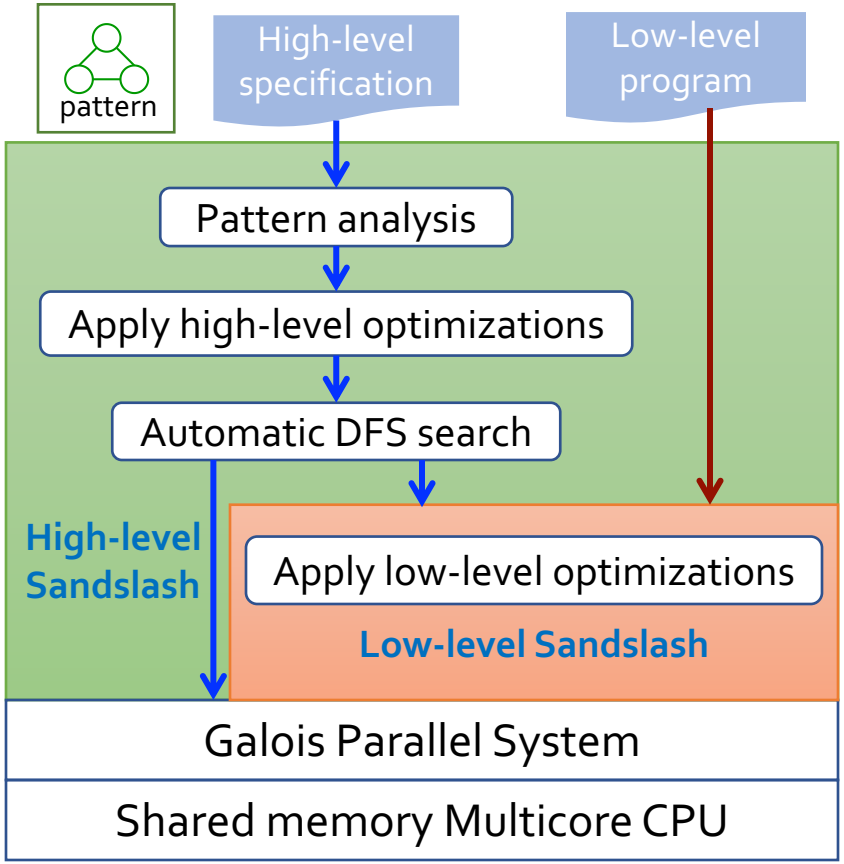
- Search space pruning
- Architectural optimizations

Outline

- Graph Pattern Mining (GPM) Use Cases
- Pangolin Overview 
- Pangolin System Internals
  - How to Achieve Easy Programming?
  - How to Achieve High Performance?
- Evaluation
- **Sandslash (optional)** 

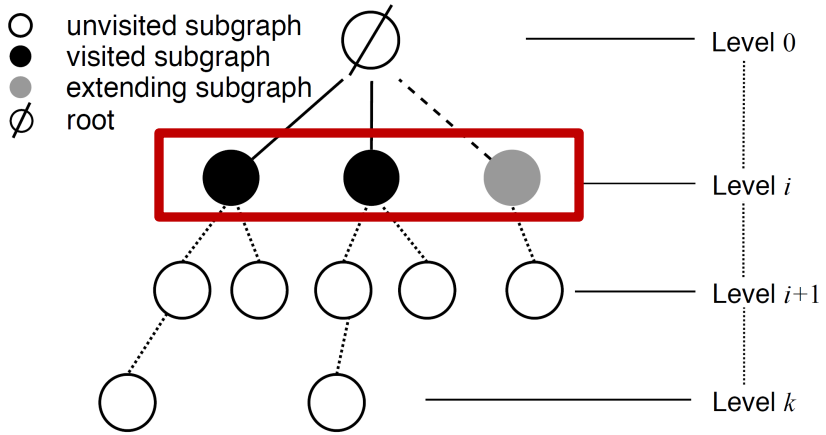
Sandslash*: A Two-level GPM System on CPU

- **Easier programming**
  - High-level: pattern specification
  - Low-level (**optional**): user defined pruning
- **Higher performance**
  - DFS exploration instead of BFS
  - High-level optimizations: automated
  - Low-level optimizations (**optional**)

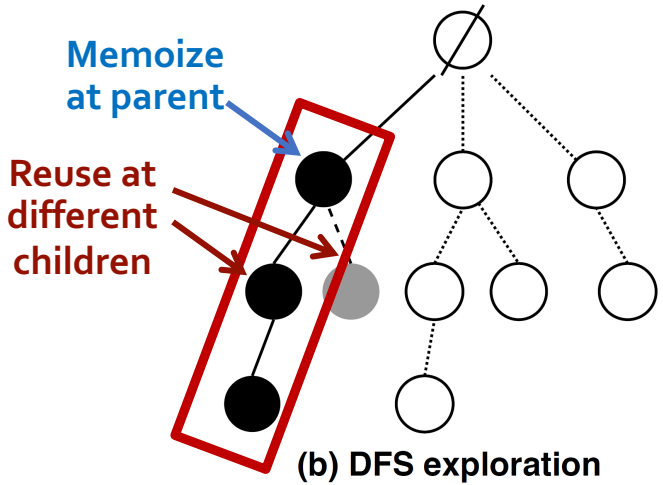


* Sandslash: A Two-Level Framework for Efficient Graph Pattern Mining, to appear in ICS'21

Search Tree Exploration: BFS vs. DFS



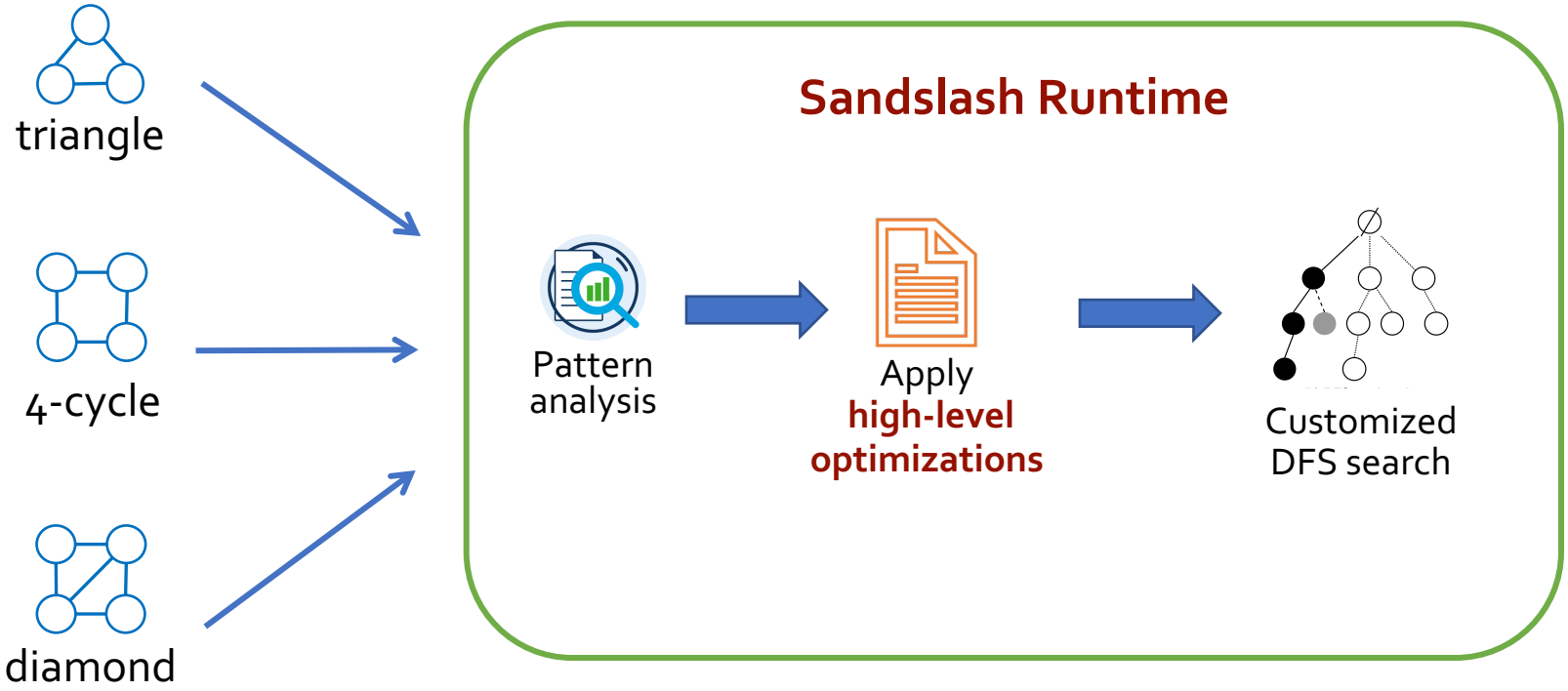
(a) BFS exploration



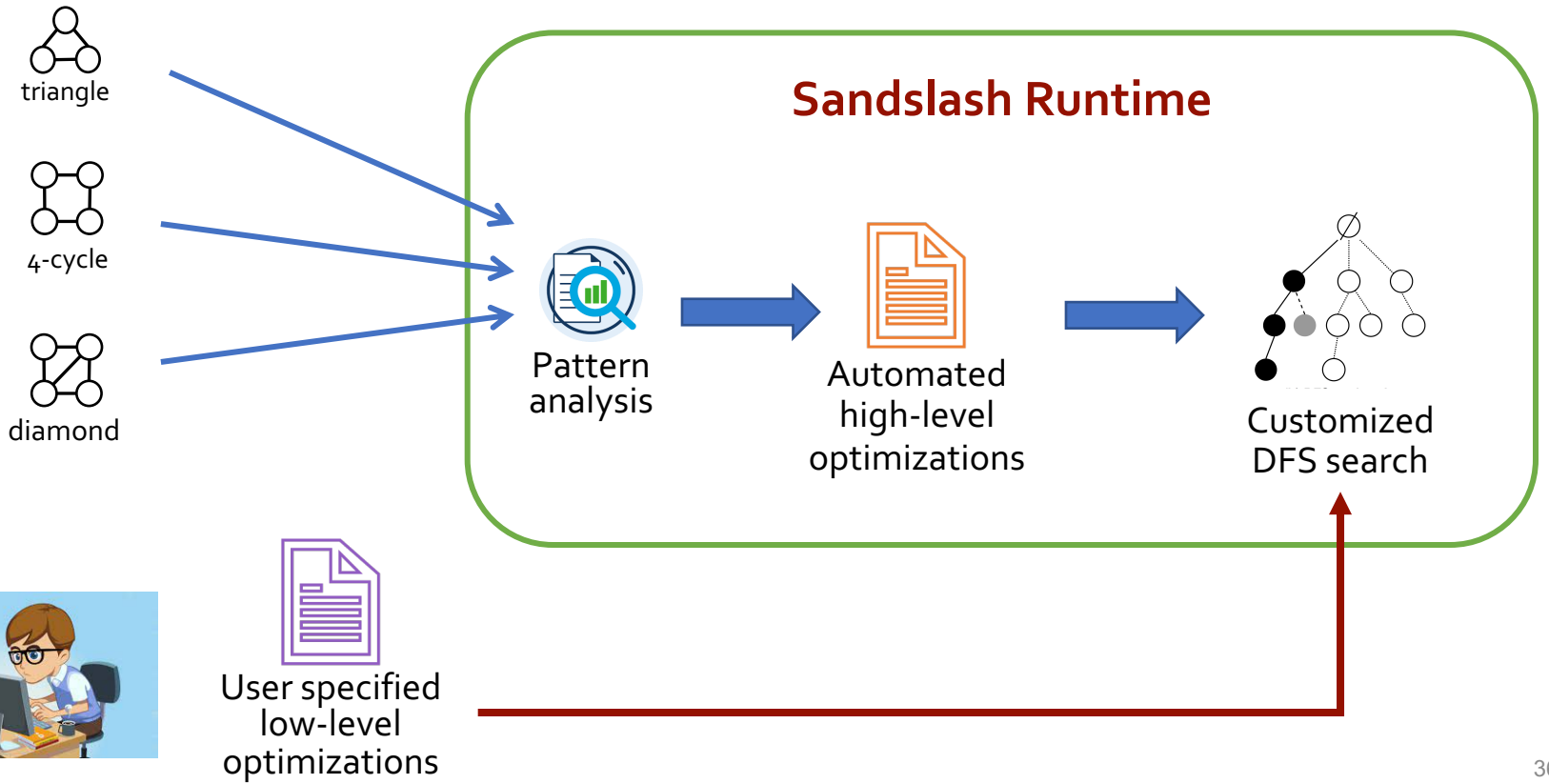
(b) DFS exploration

- Pangolin (both CPU and GPU) uses BFS
  - Good for massive parallelism on GPU
  - Large memory footprint for intermediate data
  - Lots of redundant computation
- Sandslash (CPU-only) uses DFS
  - Small and confined memory footprint
  - Avoid redundancy by **memoization**

Automated High-level Optimizations



User Defined Low-level Optimizations



Summary: GPM System on CPU

Easier Programming

Easier coding!

- API separation
- Automated high-level optimizations

Higher Performance

On CPU **8×** and **14×** speedup over Pangolin and AutoMine*

- Avoid redundancy with **memoization**
- Low-level user defined pruning

* D Mawhirter et. al., AutoMine: Harmonizing High-Level Abstraction and High Performance for Graph Mining, SOSP'19

Thank You!
Q&A