Construction of Small Diameter/ASPL Graph with GPU

Hajime Terao

The Graduate School of Informatics and Engineering
The University of Electro-Communications

Graph Golf (Order/Degree Problem)

Graph Golf as optimization problem

- Given:

  Order of graph: n

  Maximum degree of graph: d

- Minimize:

  Diameter of graph

  Average Shortest Path Length(ASPL)

- Note:

  Diameter has higher priority than ASPL.

  Smaller Diameter ≠ Better ASPL.

Difficulties in Graph Golf

- Vast search space

  At least $n!$ optimal solutions exist.
- Objective function is not convex.
- One edge can change many shortest paths.

  Every modification to the graph requires entire recalculation of ASPL/Diameter.
- The calculation time required for ASPL/Diameter is polynomial, but n is so large (up to **1e6**)

My Results

- I found 5 best solutions

Order n	Degree d	Diameter	ASPL	ASPL gap
50	4	4	2.64082	0.04898
1726	30	3	2.47921	0.01834
9344	6	7	5.48822	0.11436
65536	6	9	6.73615	0.18302
100,000	8	7	5.94733	0.20869

My Results

- I found 5 best solutions

Order n	Degree d	Diameter	ASPL	ASPL gap
50	4	4	2.6[illegible]	[illegible]
1726	30	3	2.47921	0.01834
9344	6	7	5.48822	0.11436
65536	6	9	6.73615	0.18302
100,000	8	7	5.94733	0.20869

Deepest Improvement

Approach

- Find better solution by **Simulated Annealing** from **multiple** initial solutions.
- Design **symmetric** and **memory-efficient** graph, in order to,
  1. Reduce theoretical calculation time
  2. Avoid memory bandwidth bottleneck
- Make use of **GPU** for ASPL calculation, achieved about **700x faster** than single thread naive CPU implementation

Design of Graph

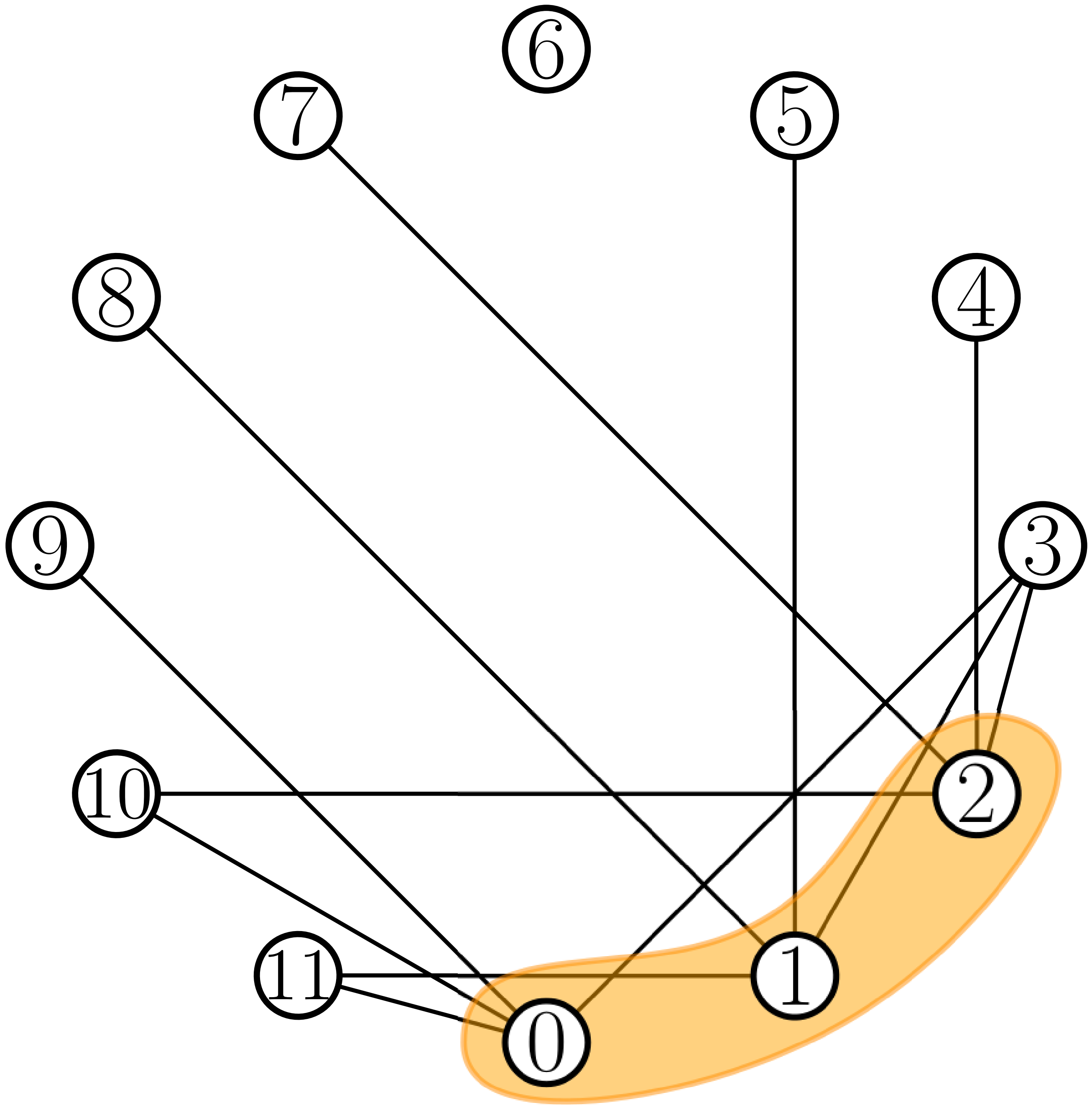
- I designed "part shift graph", similar to Cayley Graph
- Vertices have indices $0 \ldots n-1$

Indices are regarded as elements of cyclic group Z_n

- Show an example of $(n, d) = (12,4)$
- Choose size of "part" $m = 3$ from divisors of n
- Then construct a "part".

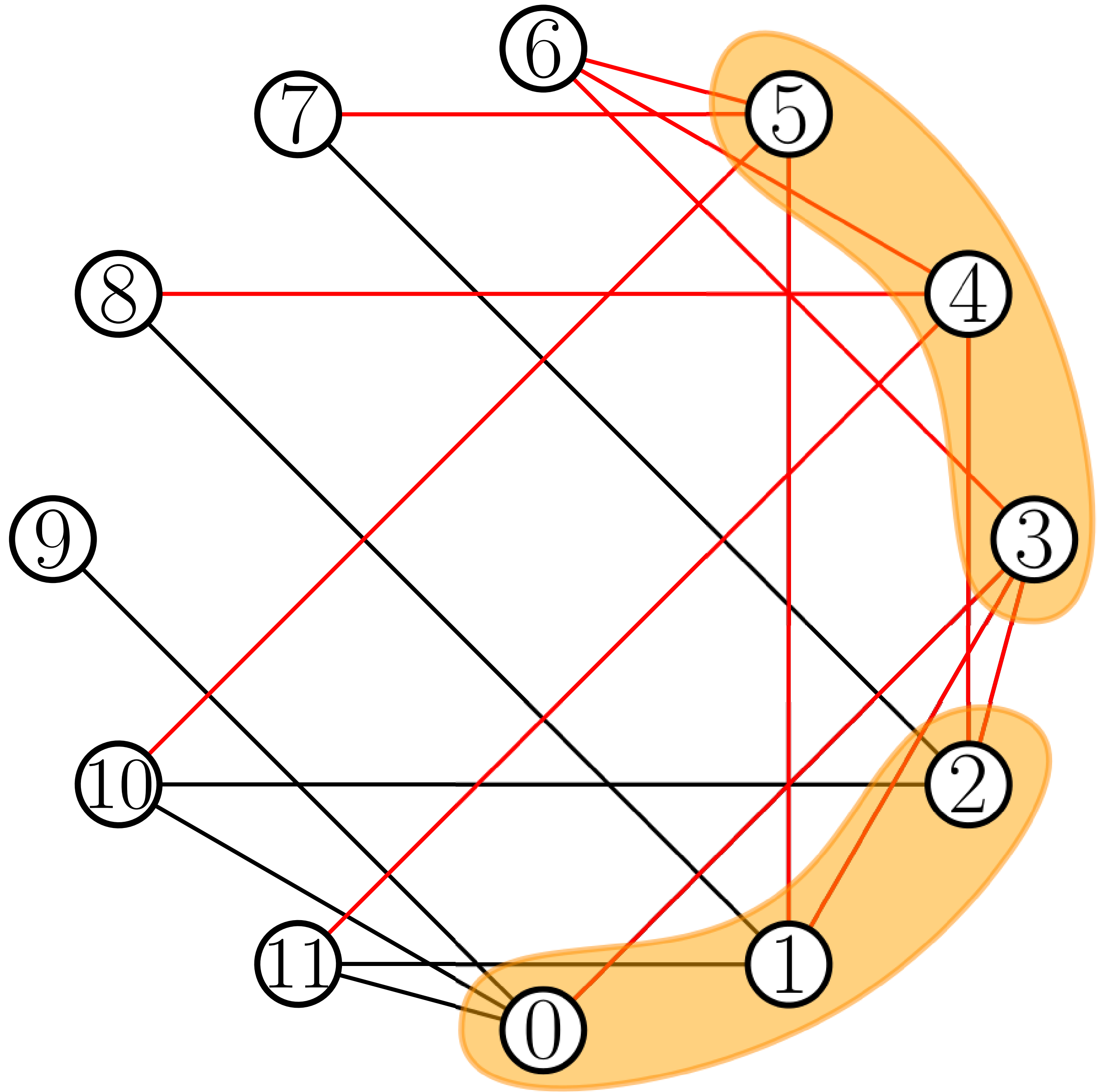
Design of Graph

- "part" is a subgraph, all edges join $0 \ldots m-1$ vertices.



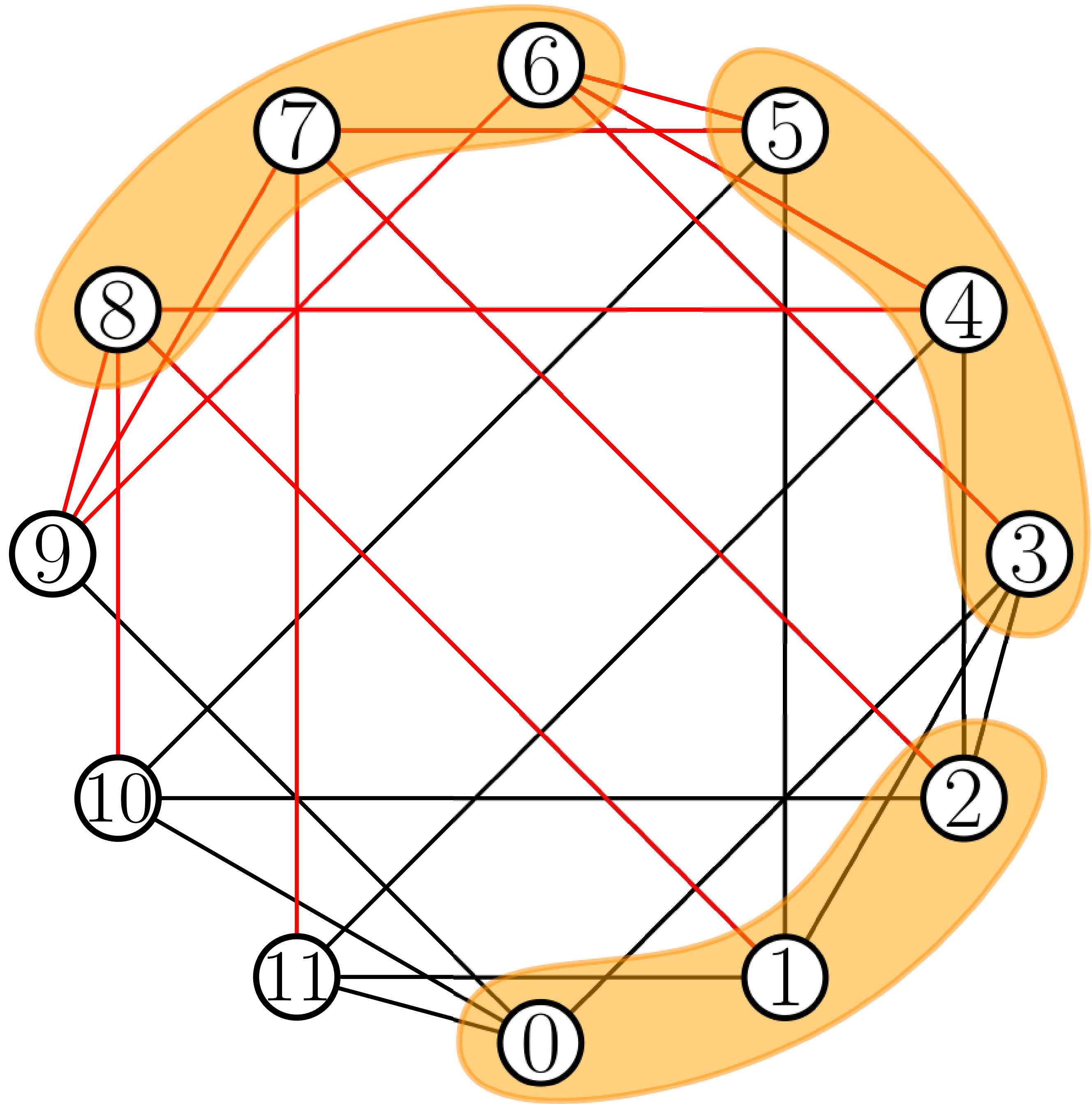
Design of Graph

- Copy & Shift the part by $m(=3)$



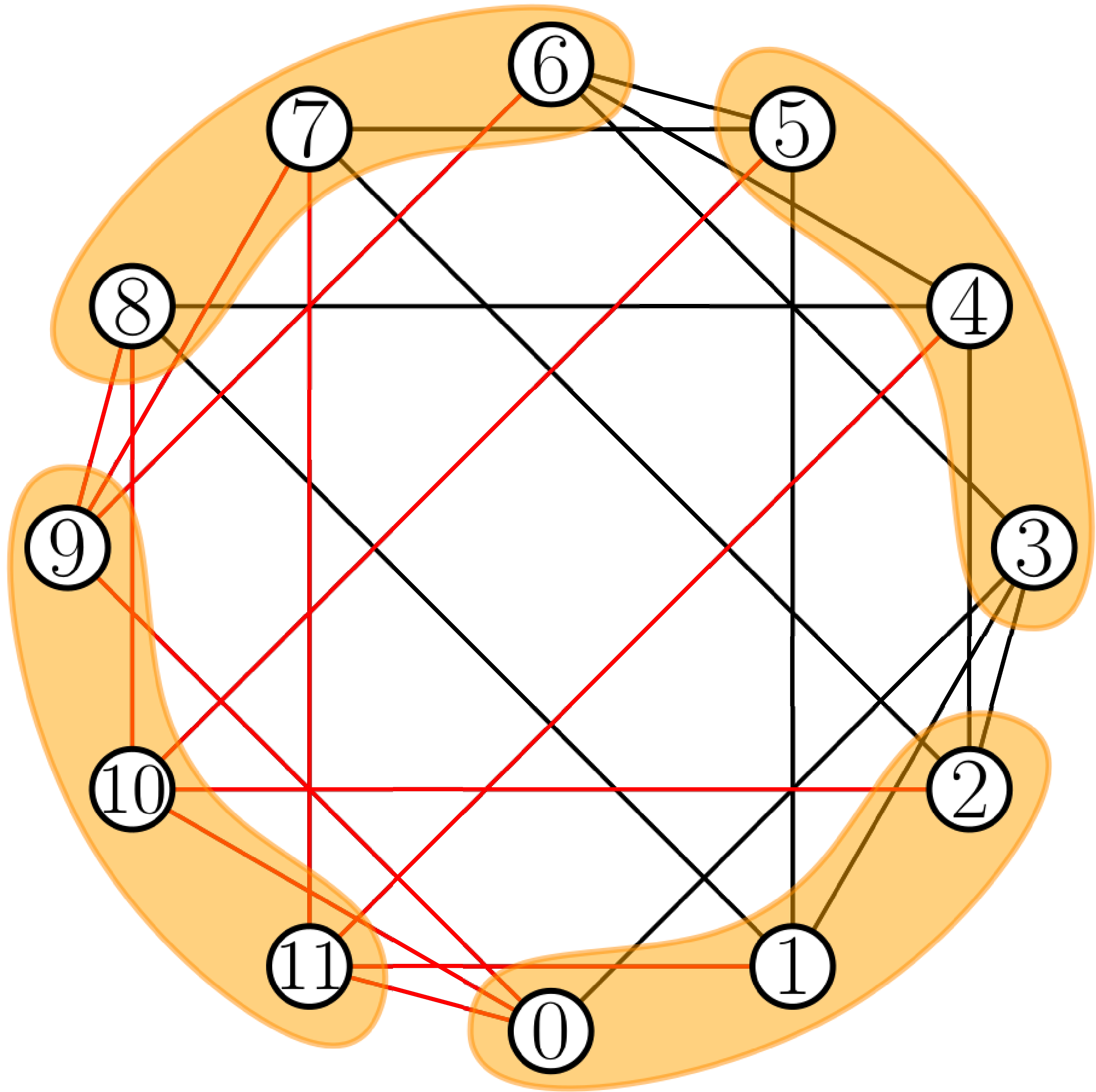
Design of Graph

- Copy & Shift the part by $m(=3)$



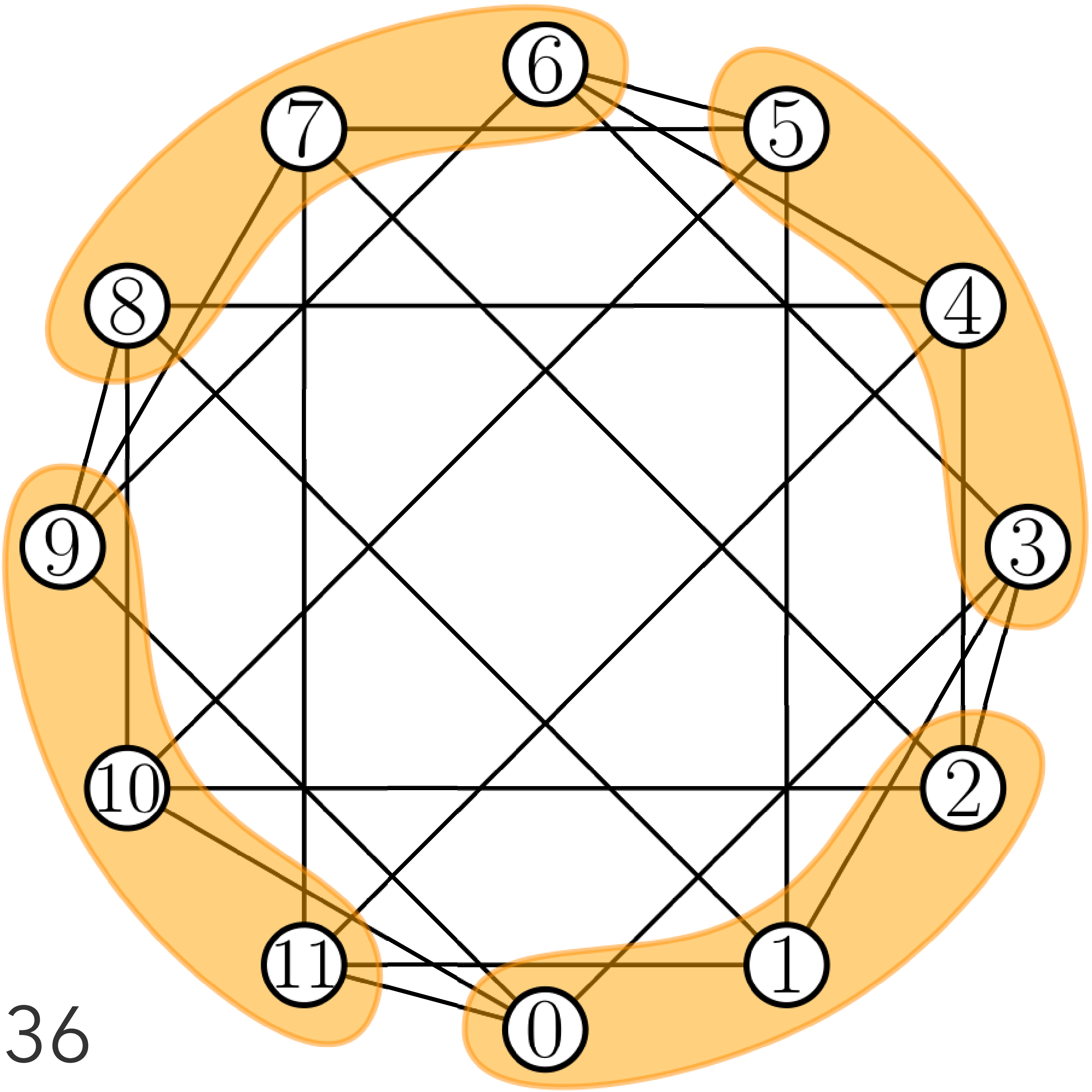
Design of Graph

- Copy & Shift the part by $m(=3)$



Design of Graph

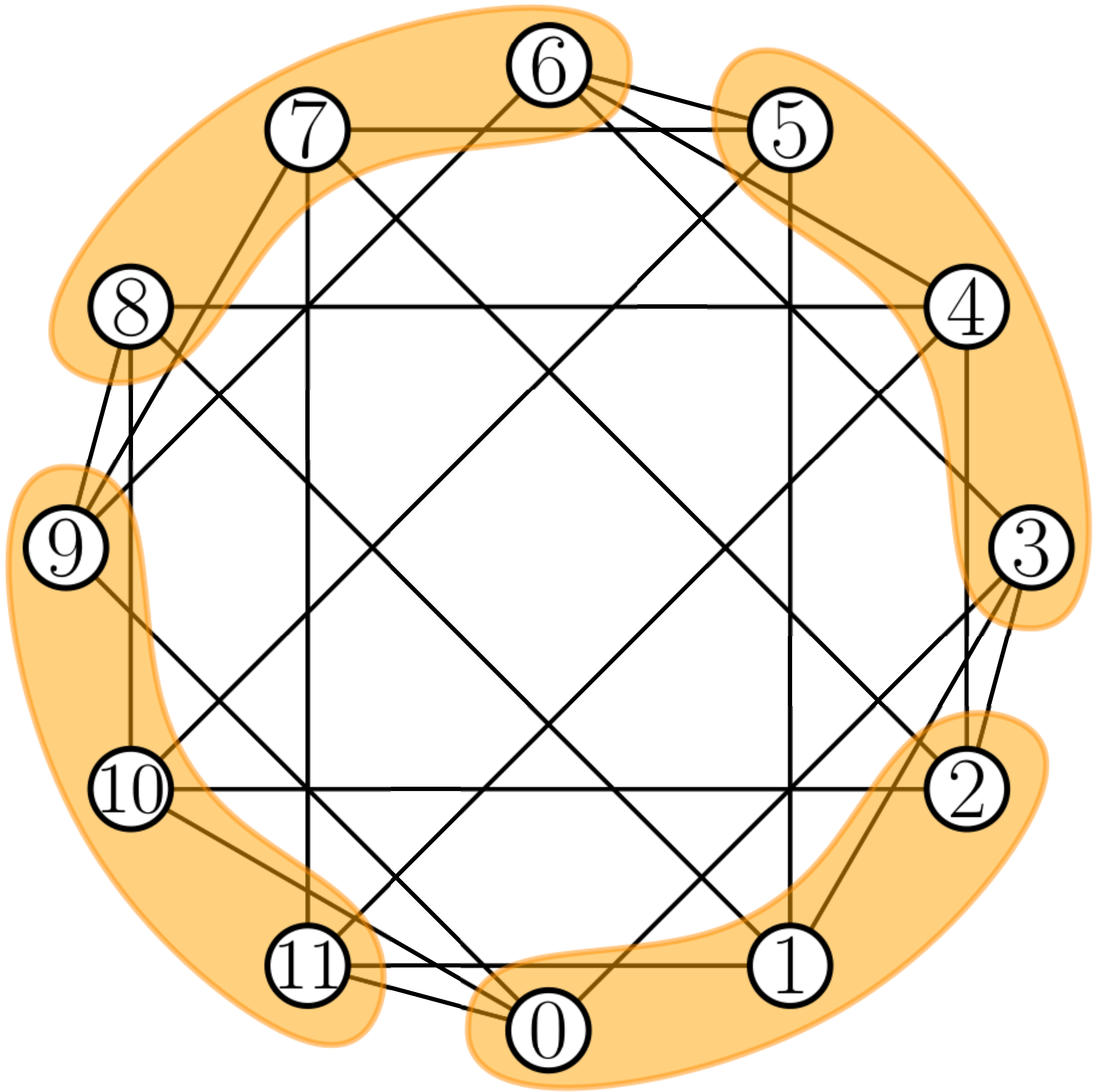
- Erase duplicated edges



Diameter: 2
ASPL: 1.63636

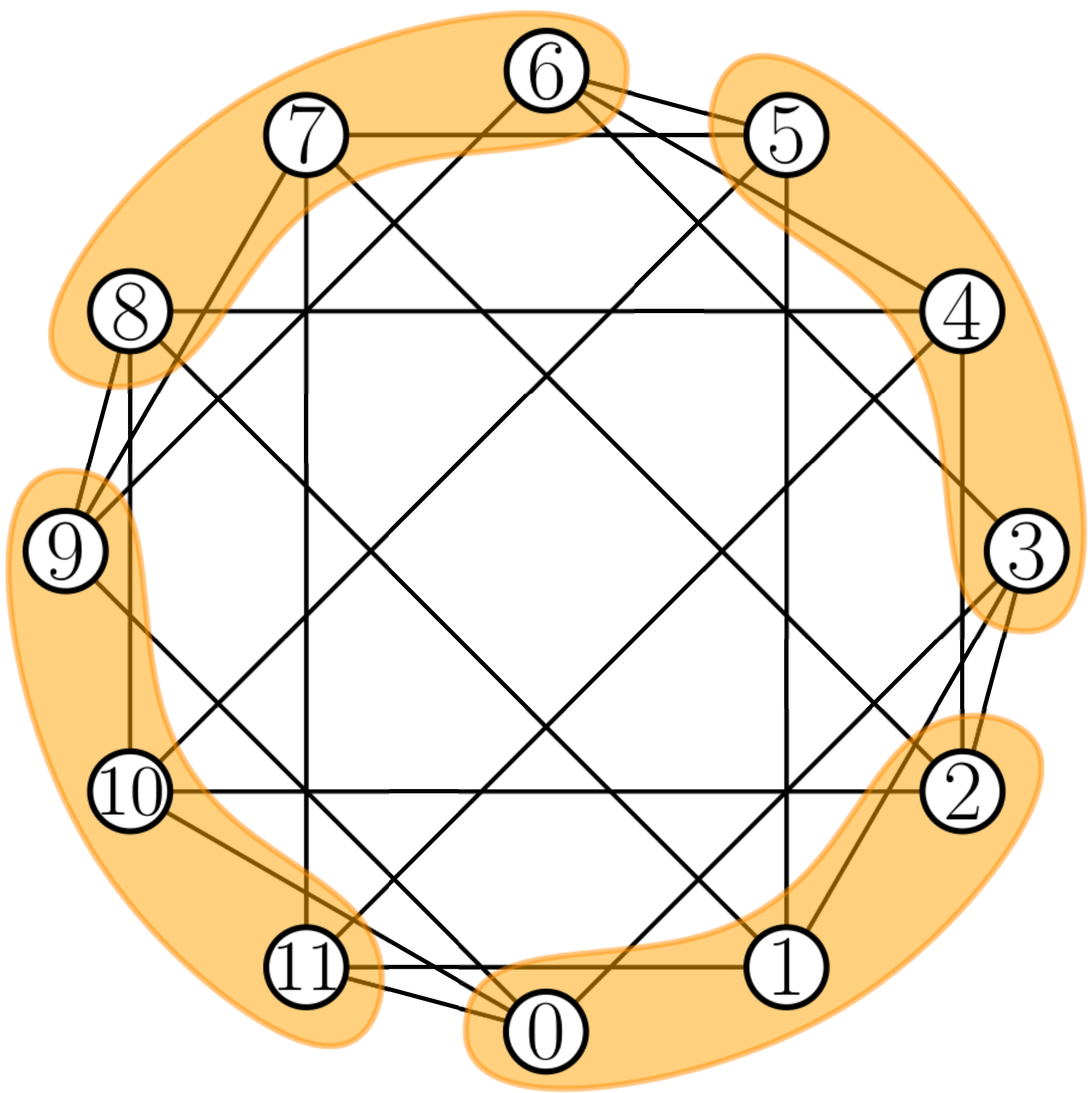
Design of Graph

- This graph is symmetric, thereby All Pairs Shortest Path problem can be solved by Single Source Shortest Path(SSSP) problem m times.



Design of Graph

- This graph is symmetric, thereby edge data requires only $O(md)$ space.



Design of Graph

- m is such a small number that $O(md)$ edge data can be stored in the cache.

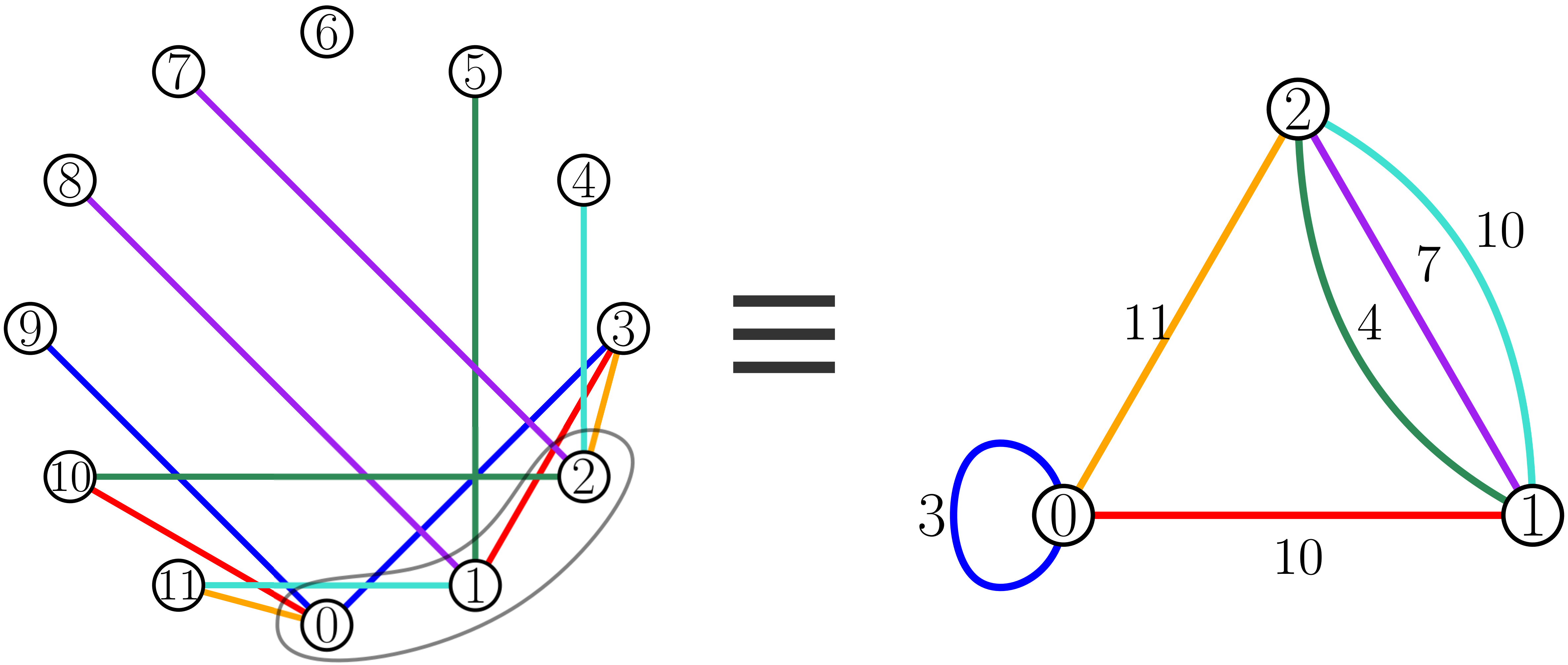
Order n	Degree d	Diameter	ASPL	ASPL gap	m
50	4	4	2.64082	0.04898	**50**
1726	30	3	2.47921	0.01834	**2**
9344	6	7	5.48822	0.11436	**16**
65536	6	9	6.73615	0.18302	**64**
100,000	8	7	5.94733	0.20869	**(4,5)**
4855	15	4	3.42917	0.13066	**5**
1,000,000	32	5	4.33066	0.34858	**64**

Simulated Annealing: Overview

- I'm a beginner of Simulated Annealing
- Representation of solution: "part"
- Cooling Schedule: Exponential Cooling
- Initial Temperature: Determined by experiments
- # of iteration: Determined by experiments (3M~10M)
- Energy: Difference of ASPLs

Simulated Annealing: Initial Solution

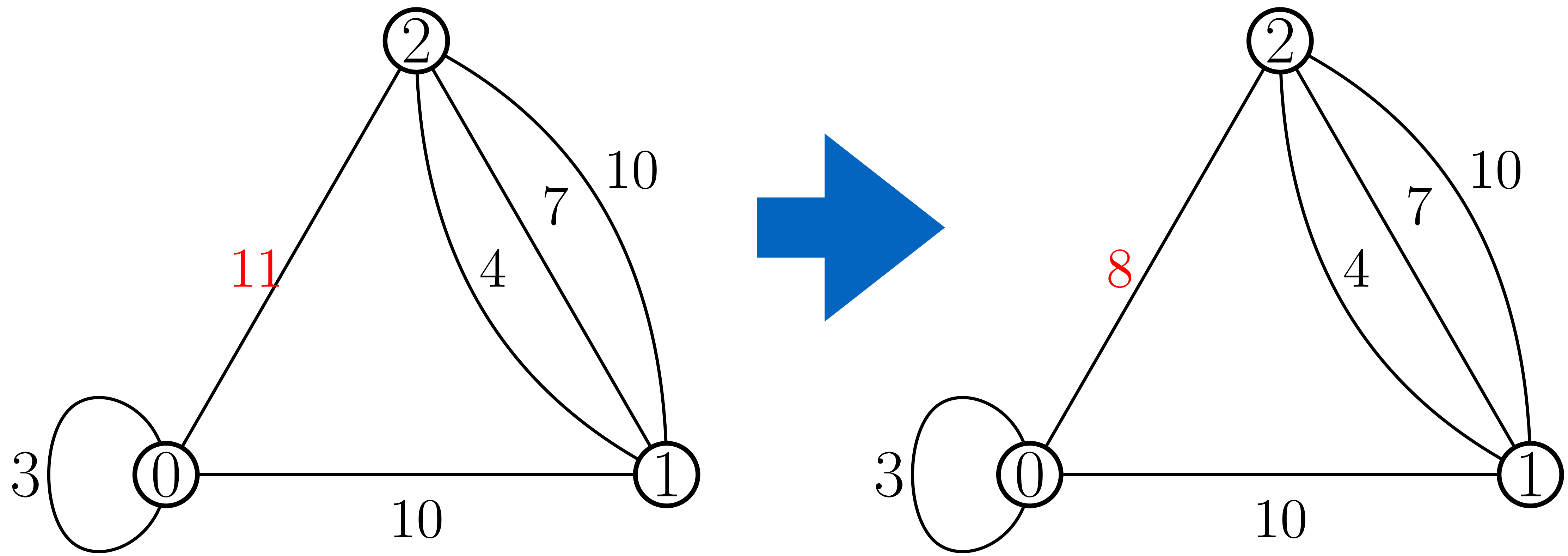
- "part" can be represented as weighted order m graph



- Generate random graph and convert it into a "part"
- Weight is difference of indices.

Simulated Annealing: The Neighbors of State

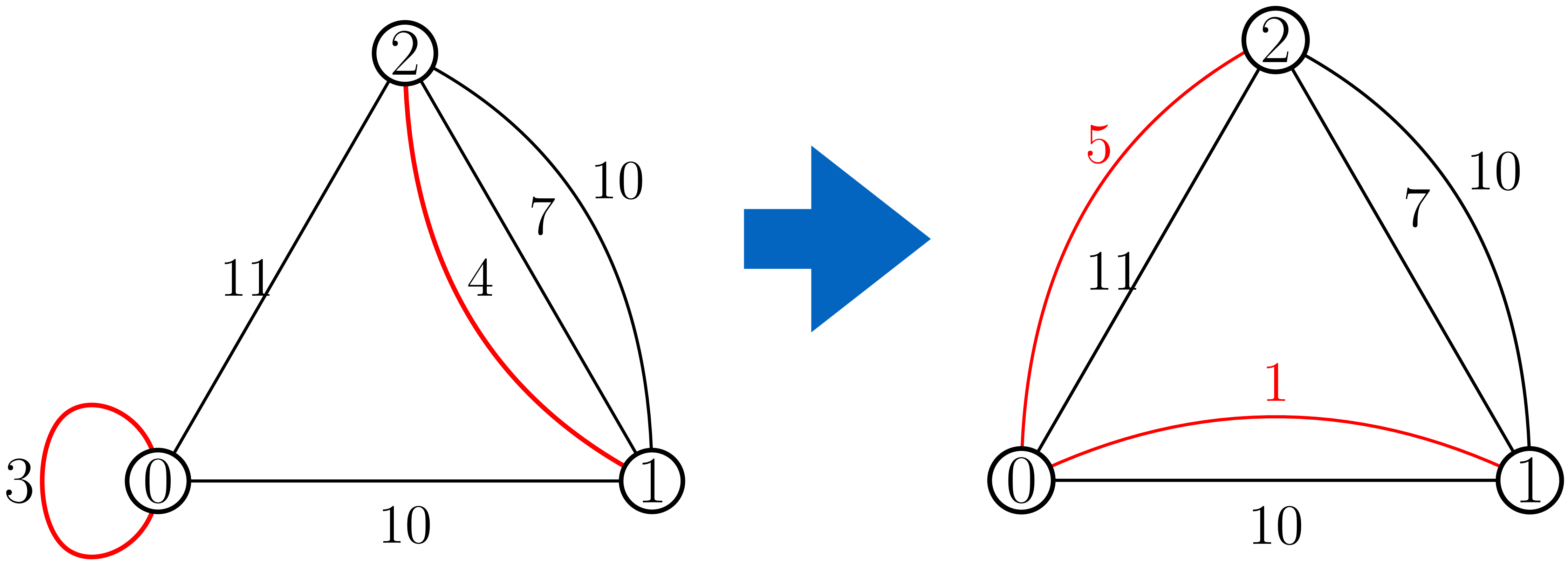
- 2 types of the neighbor
  1. Modify a weight of single edge



(edge weight)%m = (difference of indices)%m

Simulated Annealing: The Neighbors of State

- 2 types of the neighbor
  2. Cut 2 edges and reconstruct 2 edges



(edge weight)%m = (difference of indices)%m

ASPL Calculation

- Average Shortest Path Length(ASPL) of Graph (V, E) is defined as

$$\mathrm{ASPL}(V, E) = \frac{\sum_{u \in V} \sum_{v \in V} d(u, v)}{n(n-1)}$$

- The calculation of All Pairs Shortest Path is needed.
- Calculation time is enormous
  - Floyd–Warshall algorithm: $\mathrm{O}(n^3)$
  - Solving SSSP for all vertices: $\mathrm{O}(n^2 d)$
  - Solving SSSP for all vertices of the “part”: $\mathrm{O}(nmd)$
- **Parallelization** is required

Why GPU Acceleration?

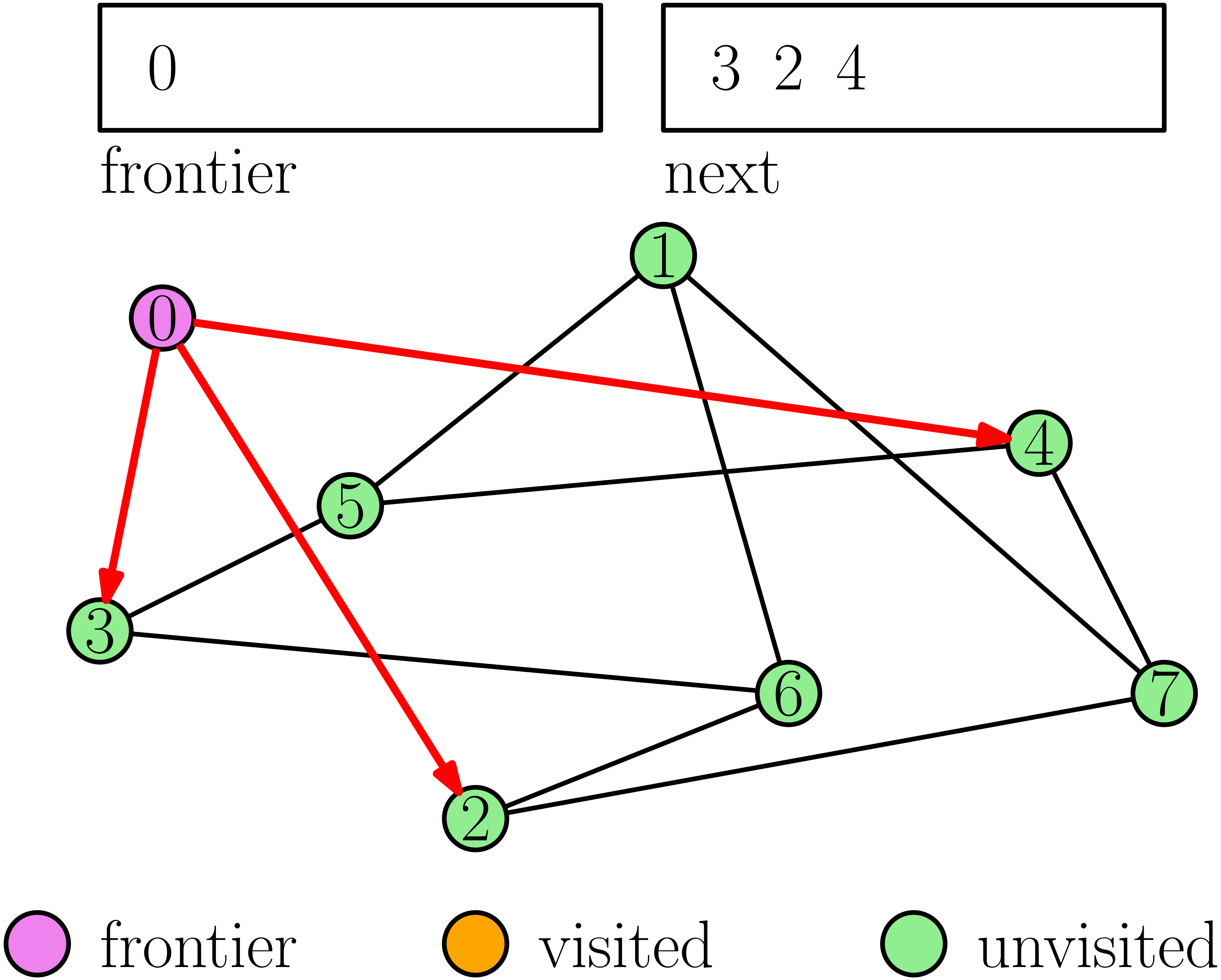
- **Parallelization** is required
- Somehow, my laboratory PC has **<u>two</u>** Geforce GTX 780
- I got **<u>four</u>** PCs equipping Geforce GTX745

OS: Ubuntu18.04
CPU: i7-4790 3.6GHz
RAM: 8GB
GPU: Geforce GTX745

- I implemented CUDA code inspired by Beamer's Bottom-up Algorithm

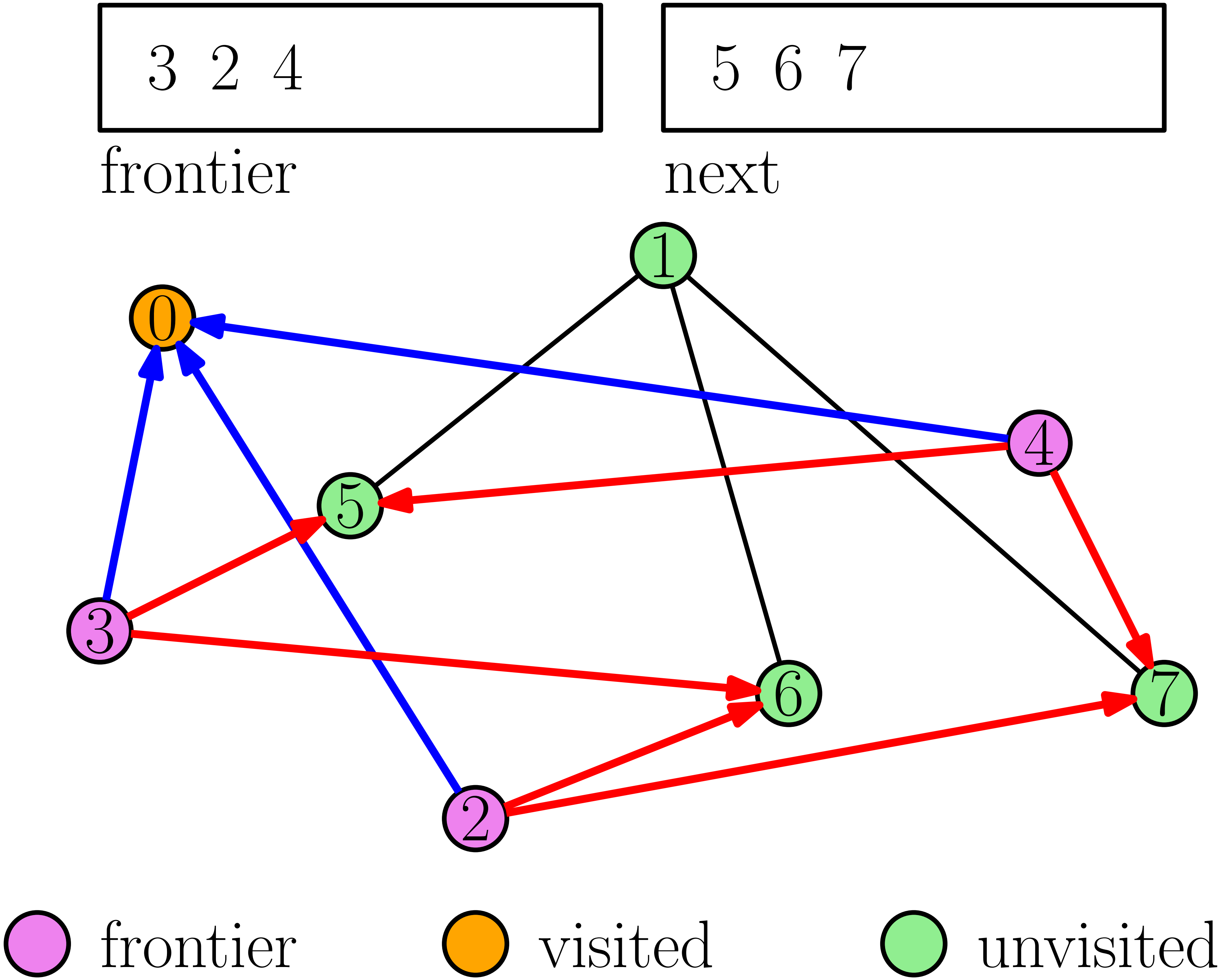
Parallel Top-Down BFS

Find **unvisited** vertices from each vertex in the **frontier**



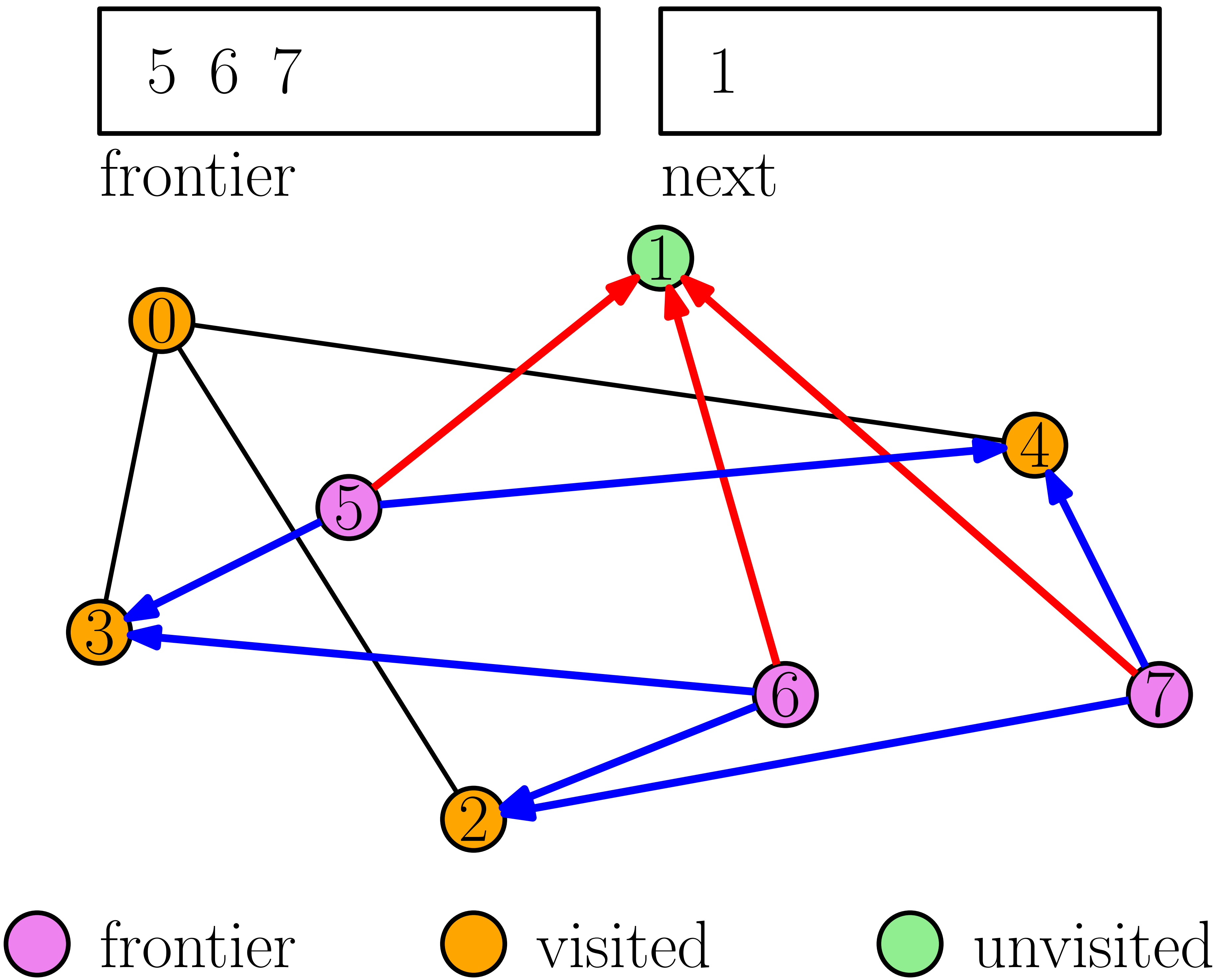
Parallel Top-Down BFS

Find **unvisited** vertices from each vertex in the **frontier**



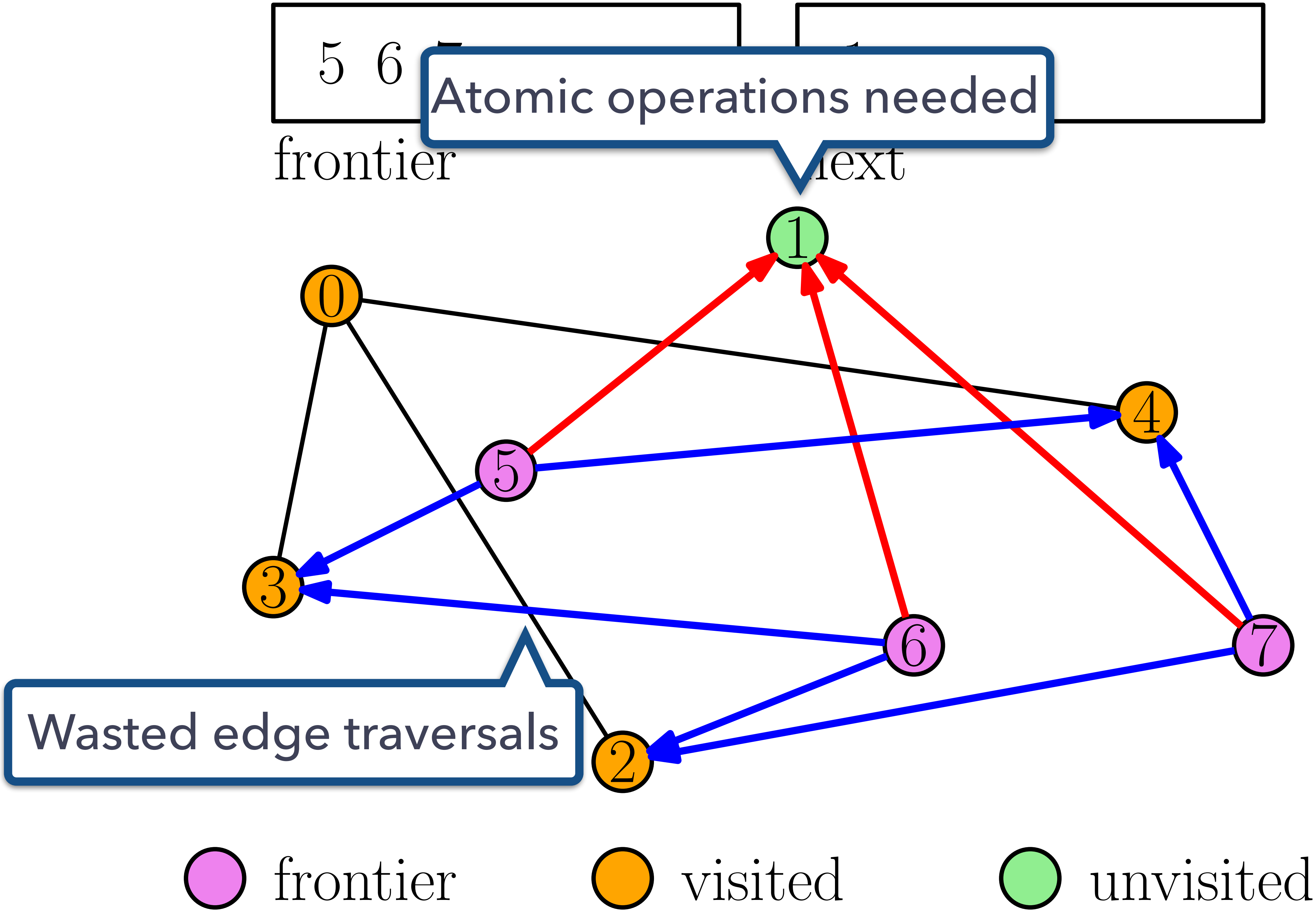
Parallel Top-Down BFS

Find **unvisited** vertices from each vertex in the **frontier**



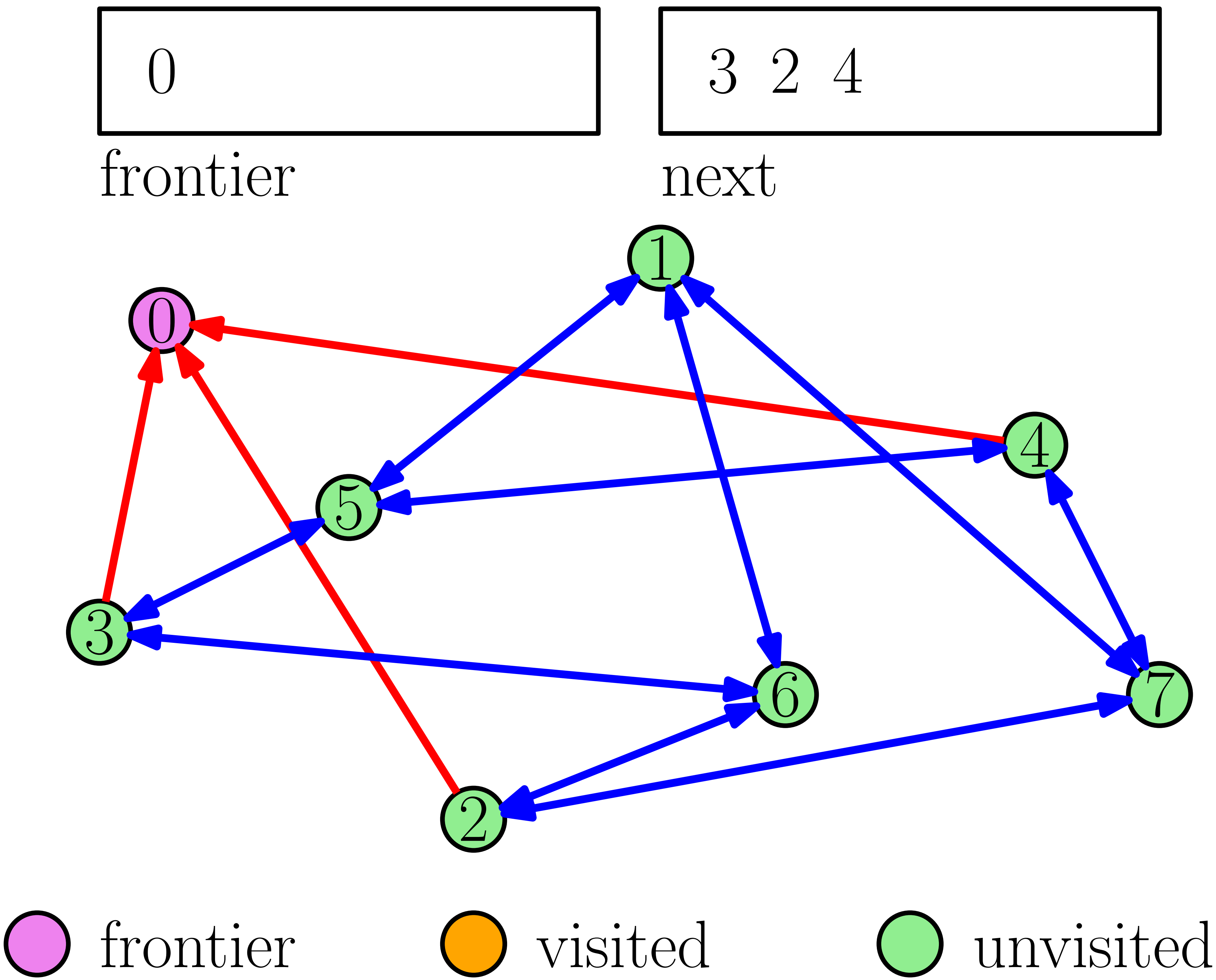
Parallel Top-Down BFS

Find **unvisited** vertices from each vertex in the **frontier**



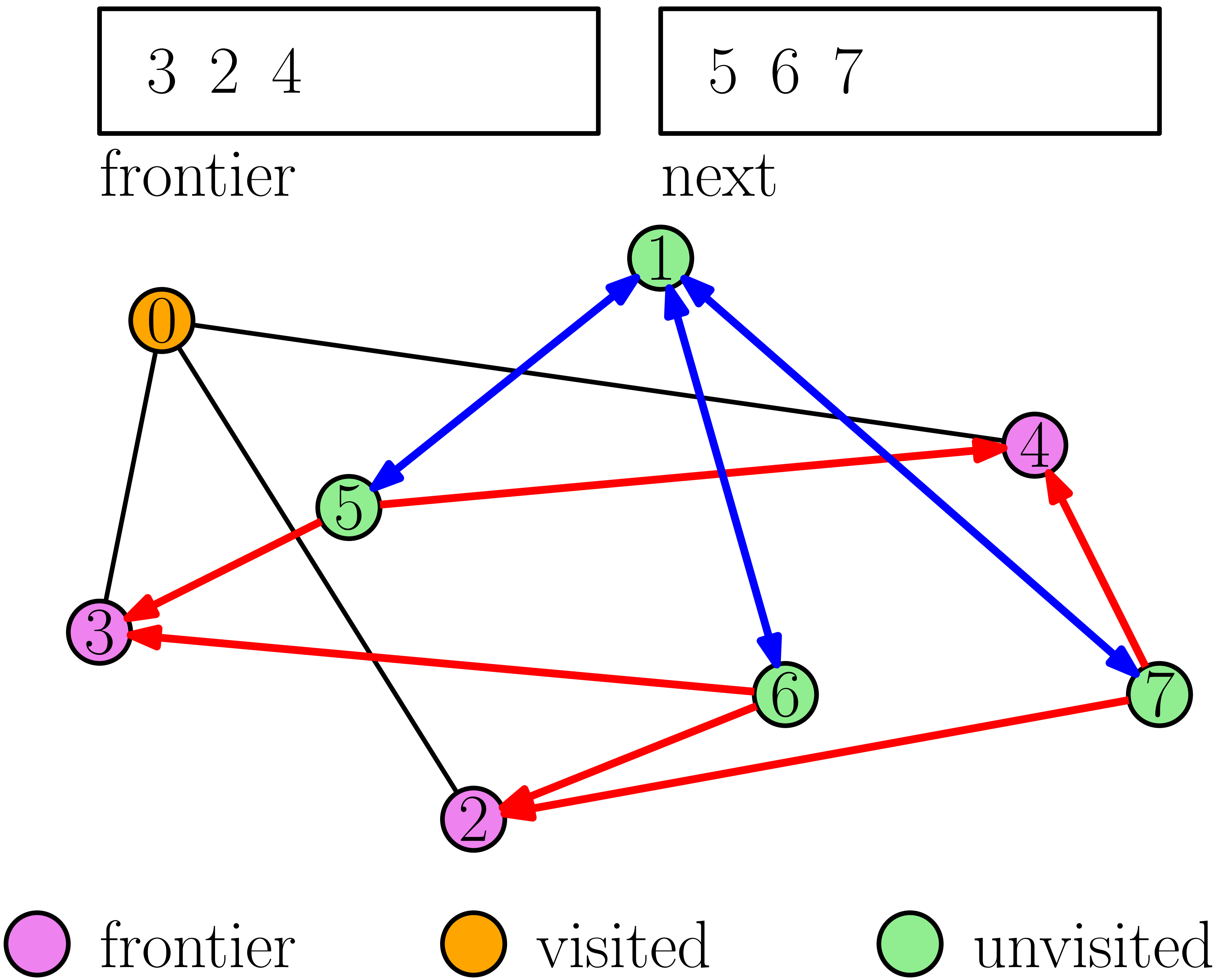
Beamer’s Algorithm (Bottom-up BFS)

Find **frontier** vertices from each **unvisited** vertex



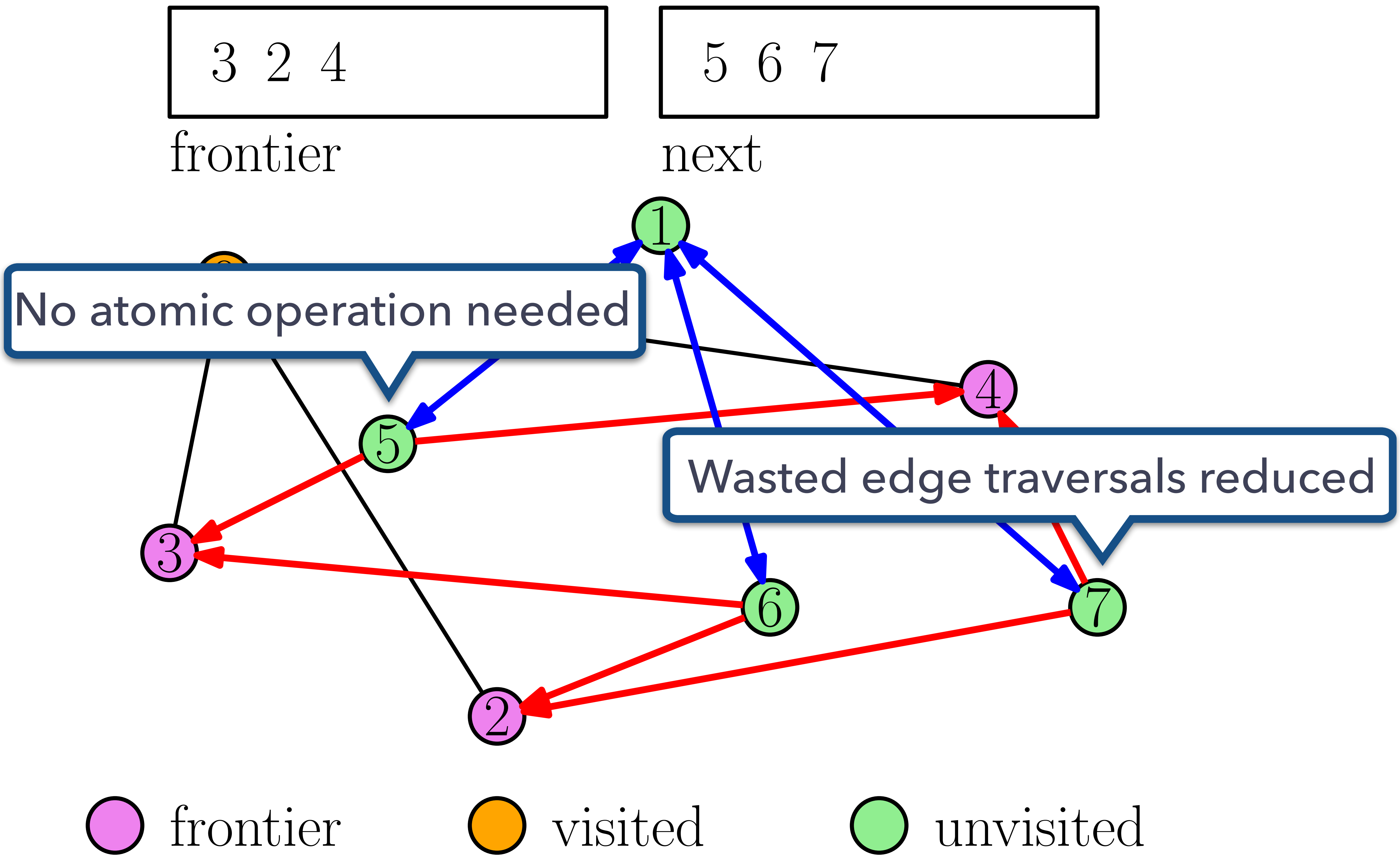
Beamer's Algorithm (Bottom-up BFS)

Find **frontier** vertices from each **unvisited** vertex



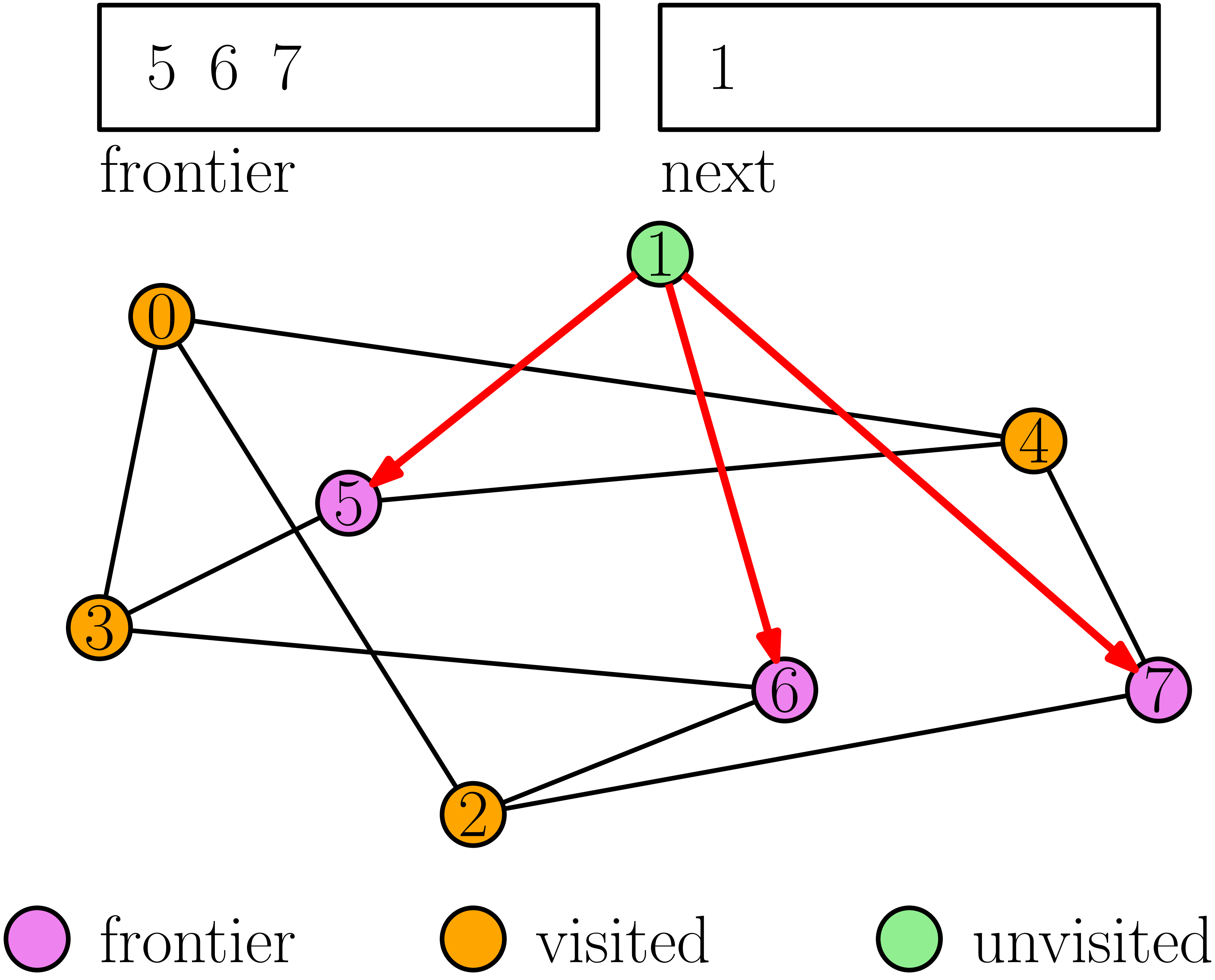
Beamer's Algorithm (Bottom-up BFS)

Find **frontier** vertices from each **unvisited** vertex



Beamer's Algorithm (Bottom-up BFS)

Find **frontier** vertices from each **unvisited** vertex

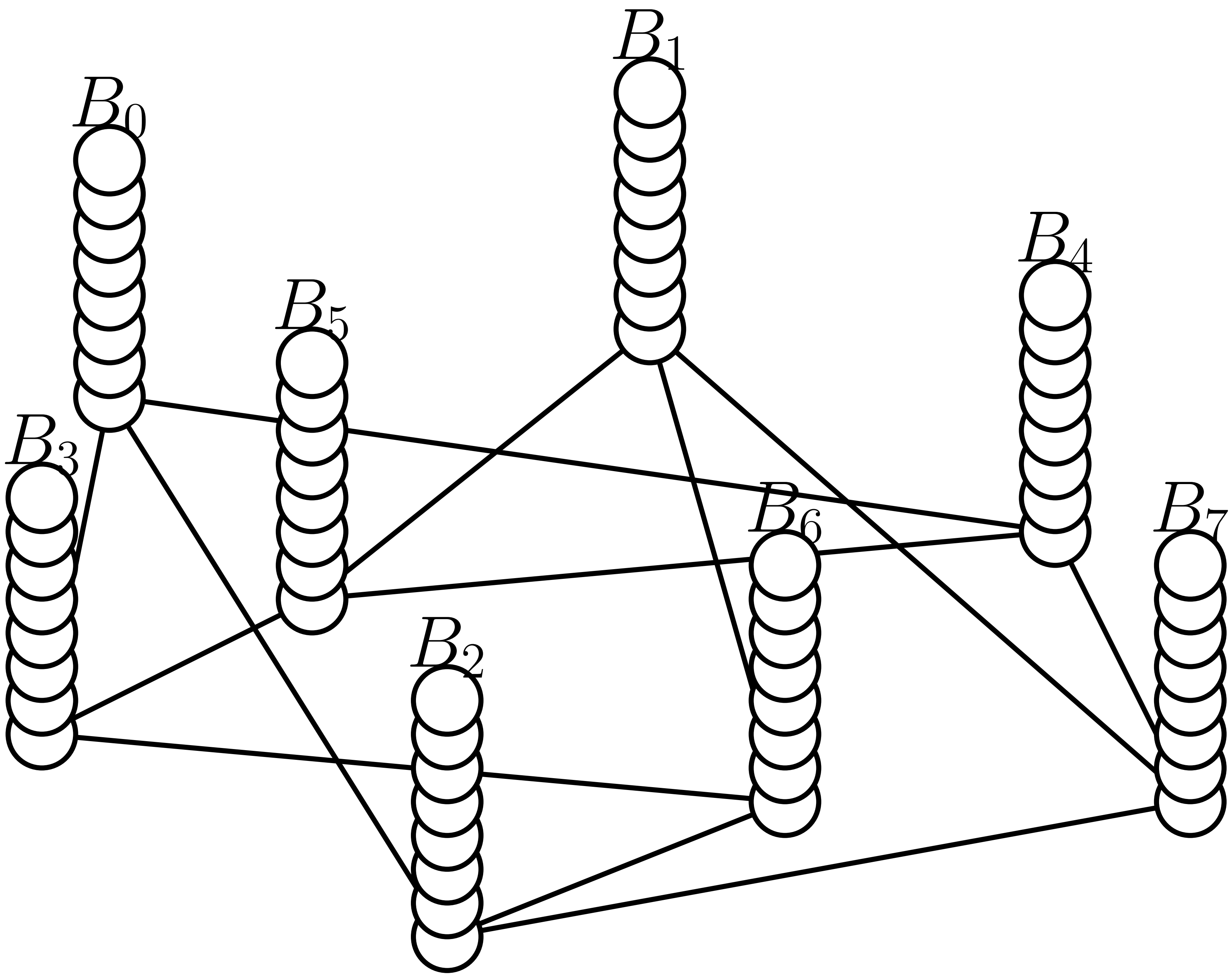


Optimization for CUDA

- Beamer's Bottom-up Algorithm is less efficient for ASPL calculation with CUDA because
  1. visited/unvisited flag is 1 bit information, but each edge traversal cause 32 byte memory access. Memory bandwidth limits performance.
  2. branch divergence gets most of the CUDA cores assigned to vertices idle.
- To maximize efficiency, perform multiple SSSP at once.

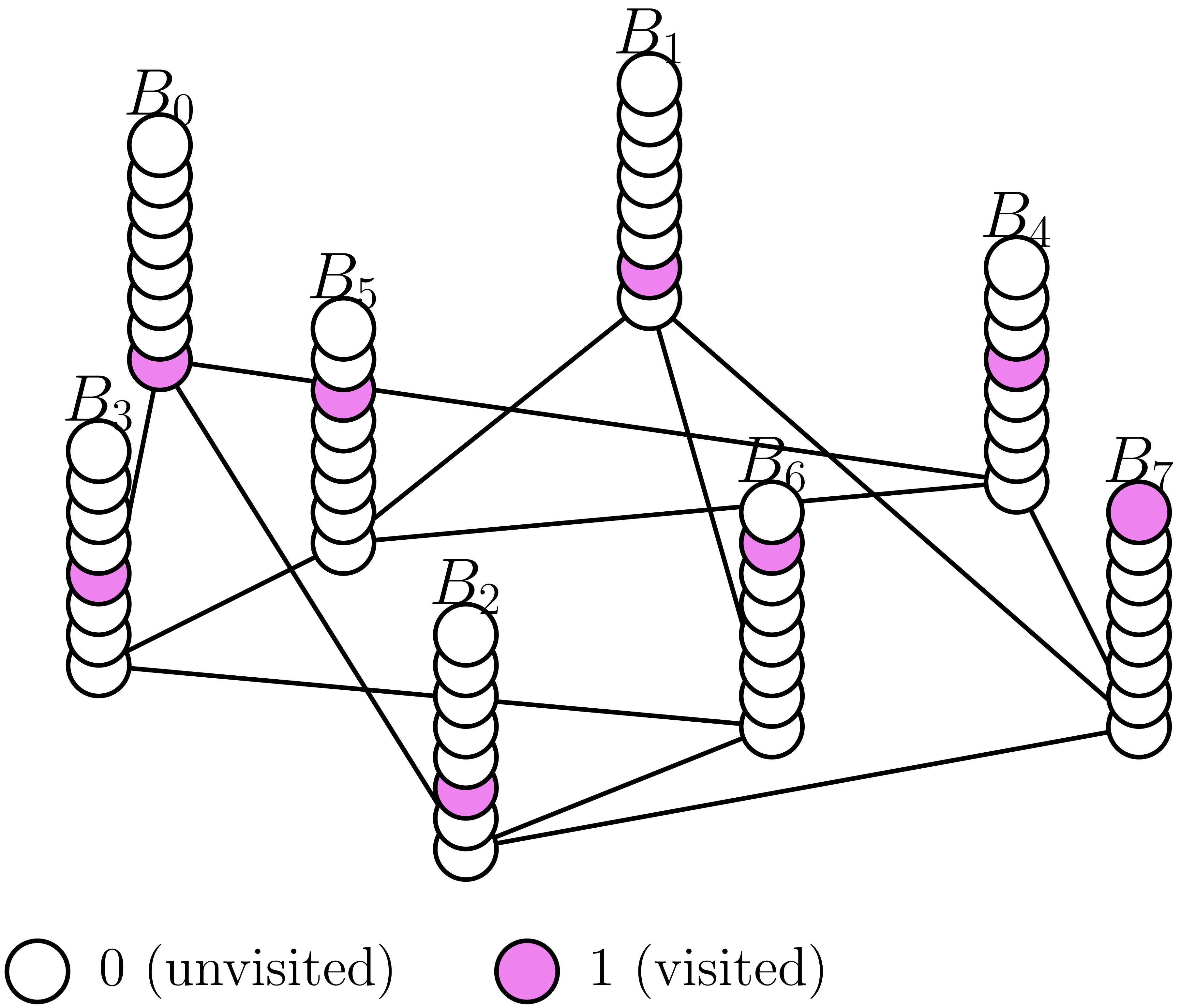
Optimization for CUDA

Assign bit vectors for all vertices



Optimization for CUDA

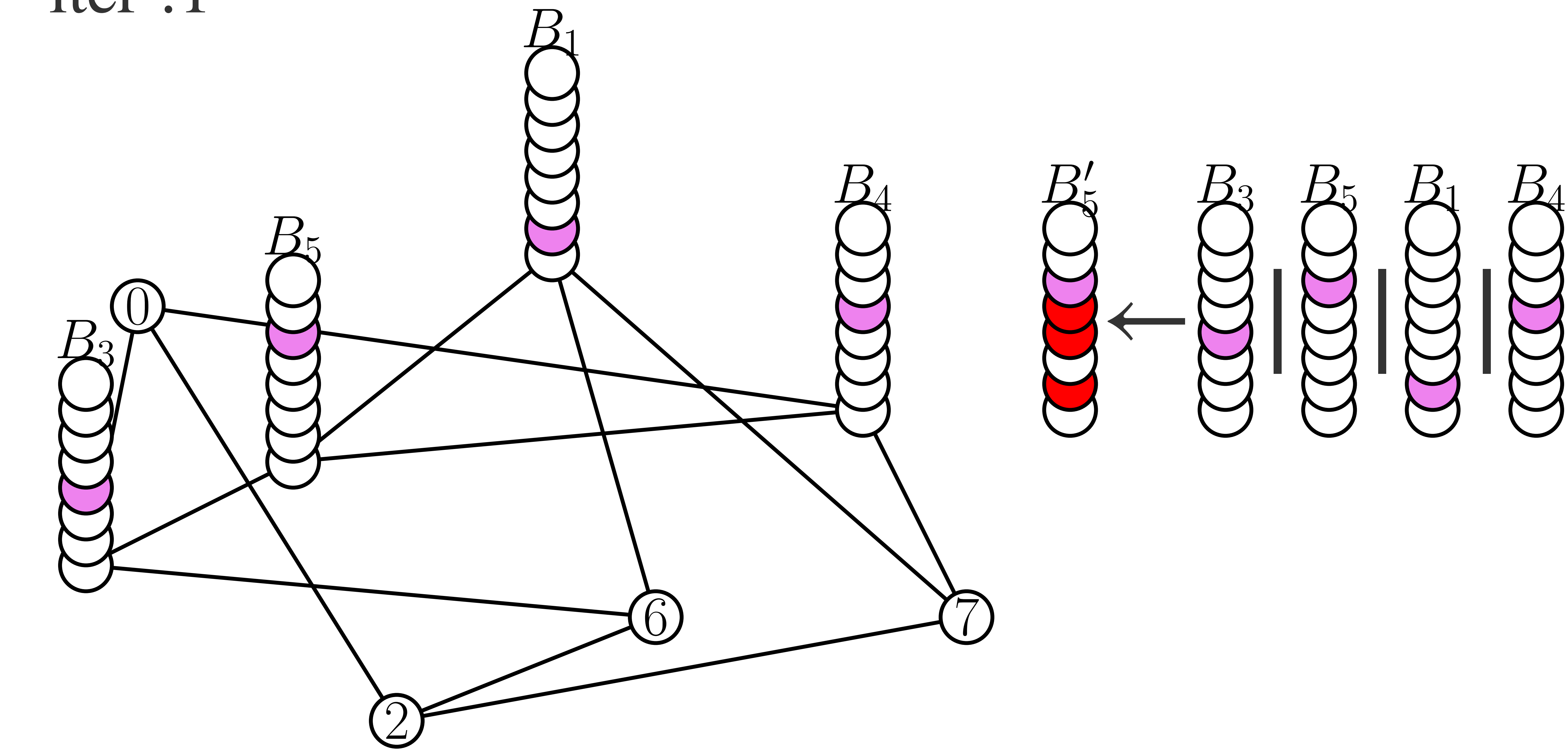
Set i th bit of i th vector "1"



Optimization for CUDA

Update vectors with bitwise OR of neighbors

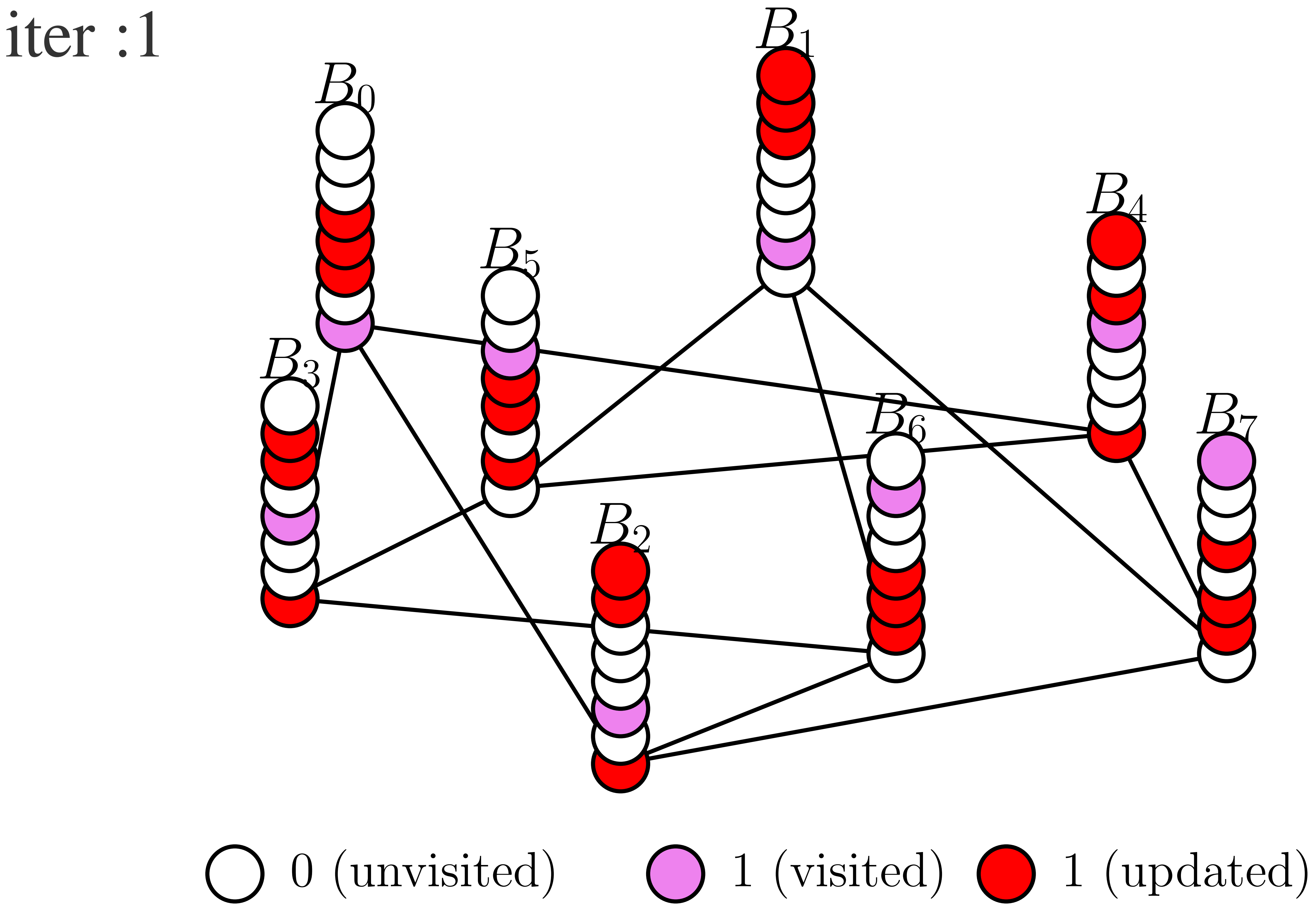
iter :1



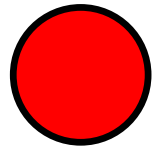
0 (unvisited) 1 (visited) 1 (updated)

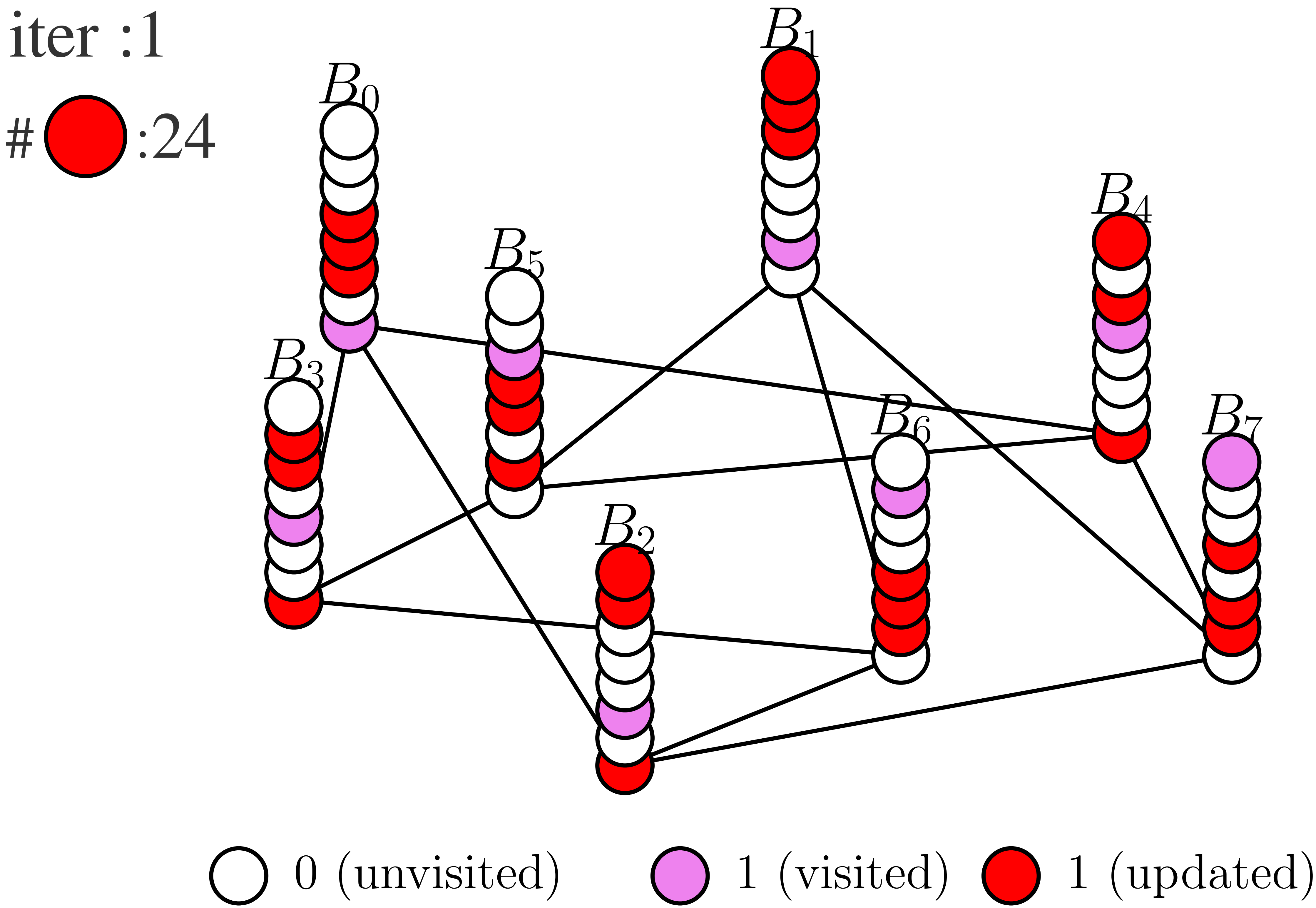
Optimization for CUDA

Update vectors in parallel



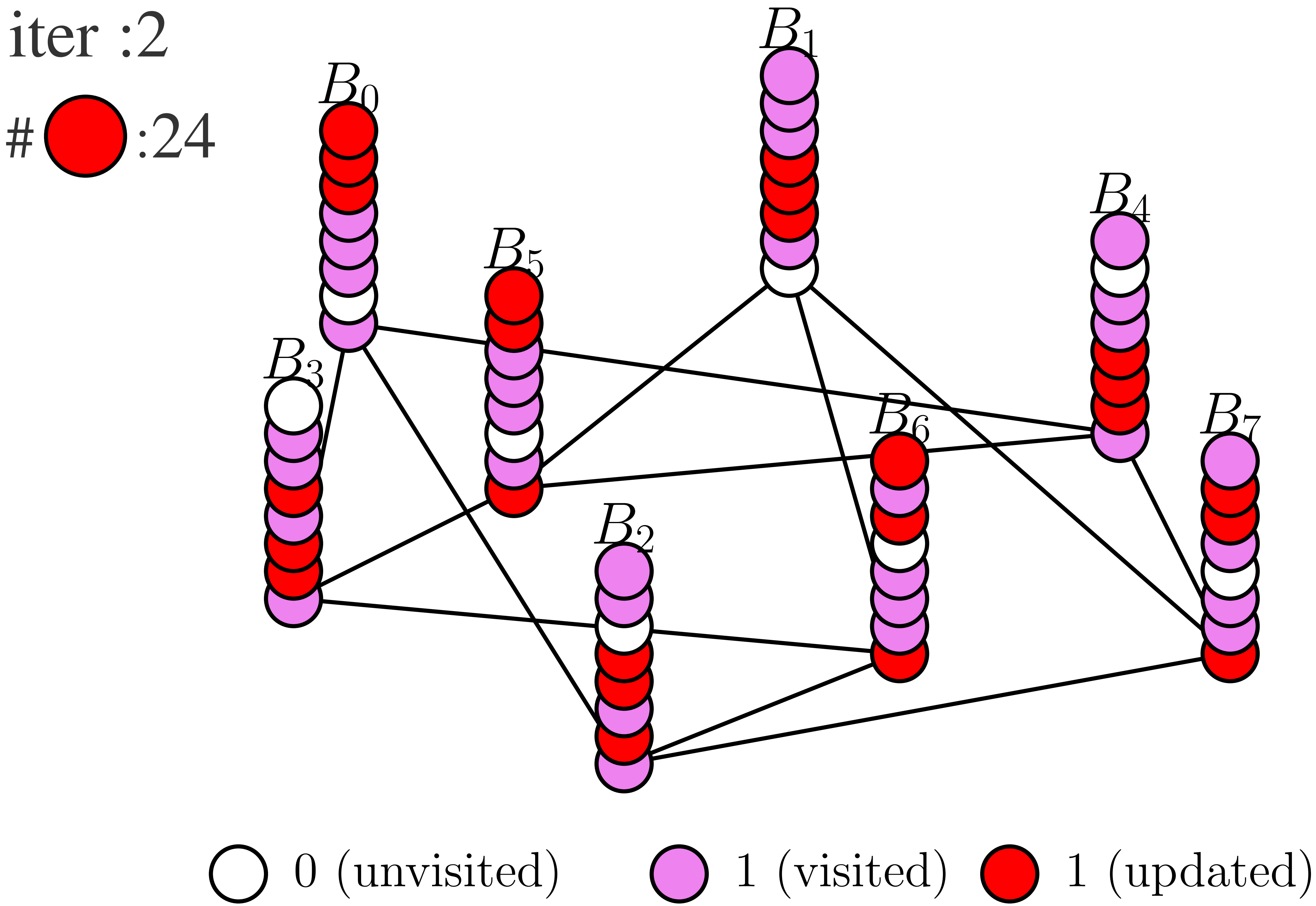
Optimization for CUDA

In t th iteration, # of  corresponds # of distance t pairs



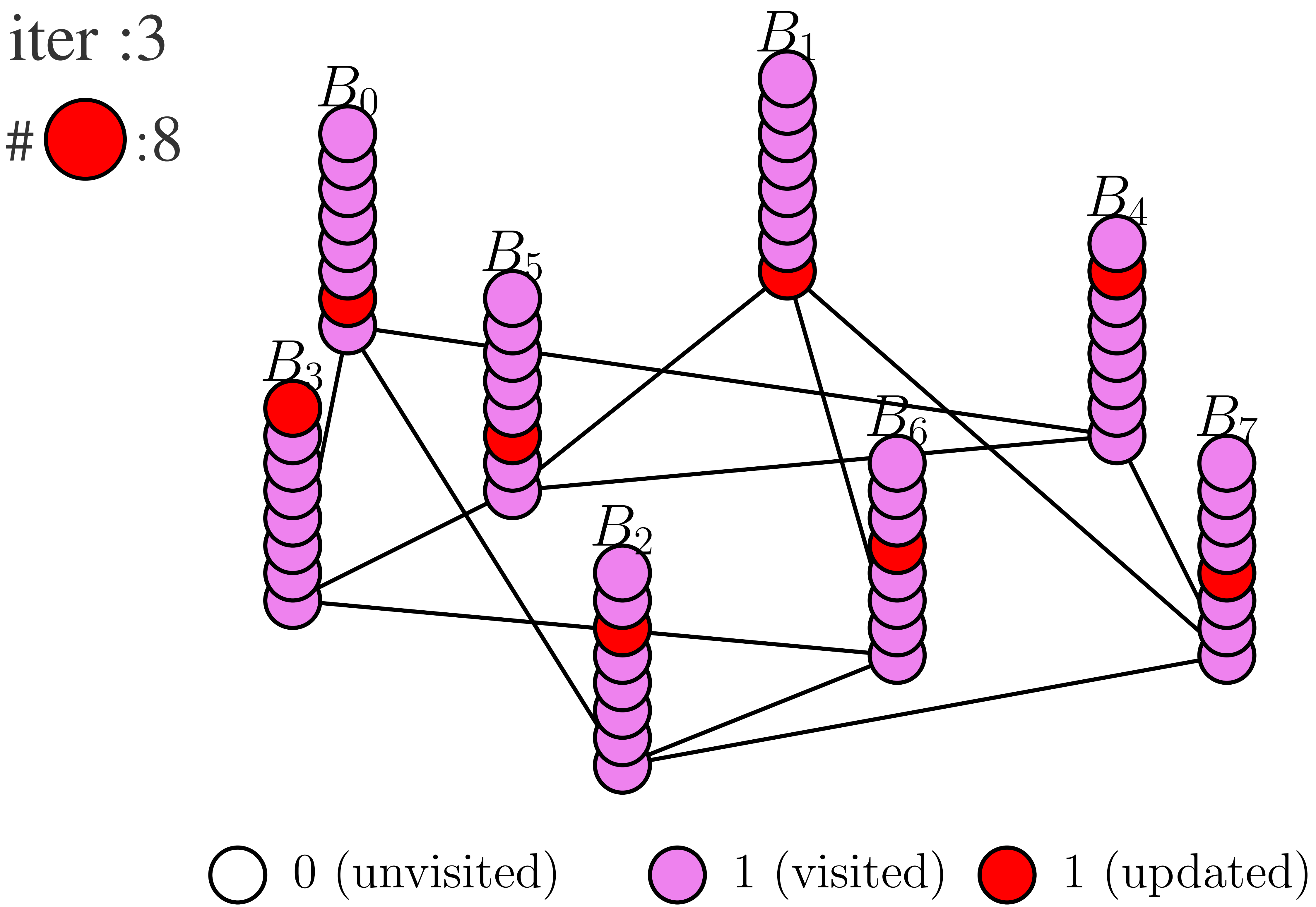
Optimization for CUDA

Update with bitwise OR of neighbors



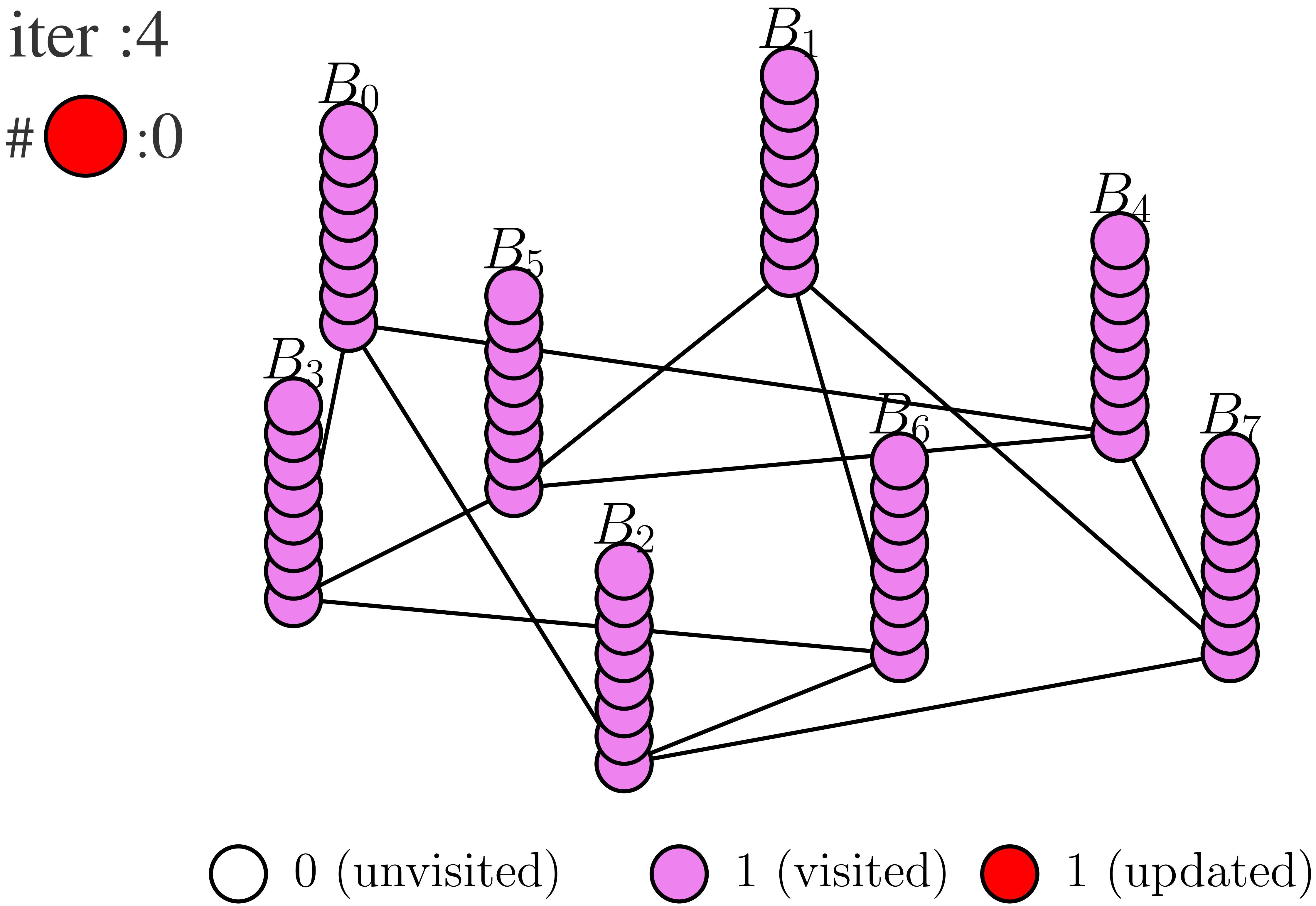
Optimization for CUDA

Update with bitwise OR of neighbors



Optimization for CUDA

Terminate iteration when all bit get "1"



Optimization for CUDA

iter :1

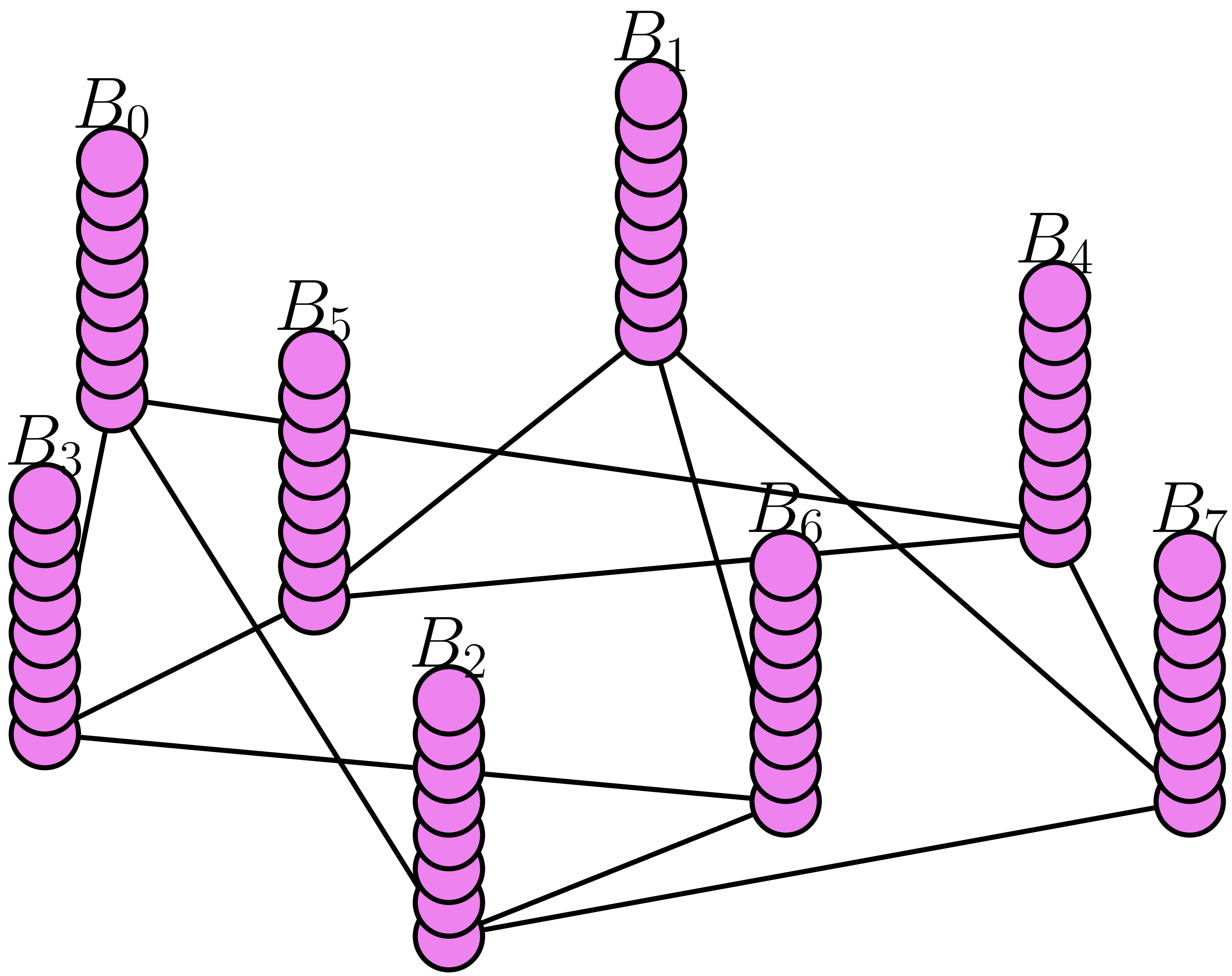
:24

iter :2

:24

iter :3

:8



0 (unvisited) 1 (visited) 1 (updated)

$$\text{ASPL} : \frac{24 \cdot 1 + 24 \cdot 2 + 8 \cdot 3}{8 \cdot 7} = 1.714285\ldots$$

Performance of the implementation

- This implementation reduces memory access drastically.
- ASPL of part shift graph with $(n, d, m) = (1e6, 32, 64)$ can be calculated in 113ms with Geforce GTX780
- ASPL of entire graph with $(n, d) = (1e6, 32)$ is calculated in 160s, 710x faster than native serial BFS implementation with i7-8700.

Conclusion

- "part shift graph" can achieve small Diameter/ASPL.
- GPU acceleration is powerful tool for ASPL calculation of large graphs.

Source Code & References

- Source Code

  https://github.com/confused-uec/graphgolf-cuda

- References

Scott Beamer, Krste Asanović, and David Patterson.
Searching for a parent instead of fighting over children: A fast breadth-first search implementation for graph500. Technical Report UCB/EECS-2011-117, EECS Department, University of California, Berkeley, 2011.

Scott Beamer, Understanding and Improving Graph Algorithm Performance, Technical Report UCB/EECS-2016-153 EECS Department, University of California, Berkeley 2016.