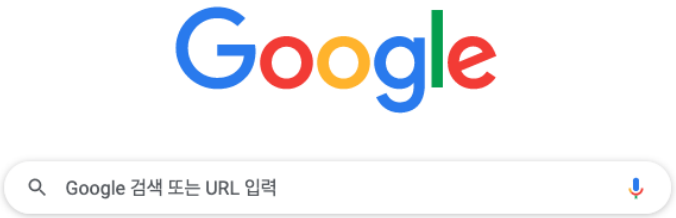
Decoupling Schedule, Topology Layout, and Algorithm to Easily Enlarge the Tuning Space of GPU Graph Processing

Shinnung Jeong[1], Yongwoo Lee[1], Jaeho Lee[1], Heelim Choi[1],
Seungbin Song[1], Jinho Lee[2], Yongsok Kim[1], Hanjun Kim[1]
Yonsei University[1] Seoul National University[2]

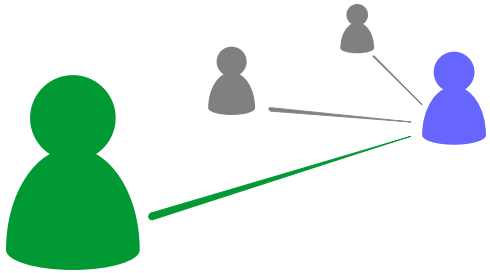
October 10-12, 2022

YONSEI UNIVERSITY

Graph Processing Is Important!

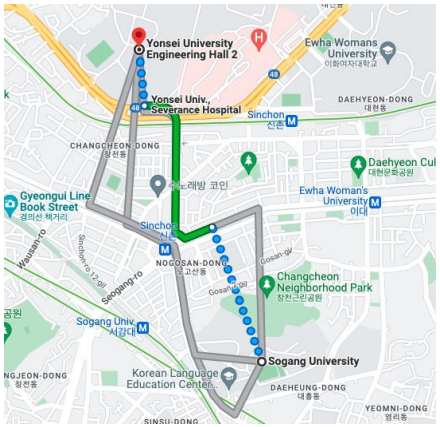


Web Search



Social Network Analysis

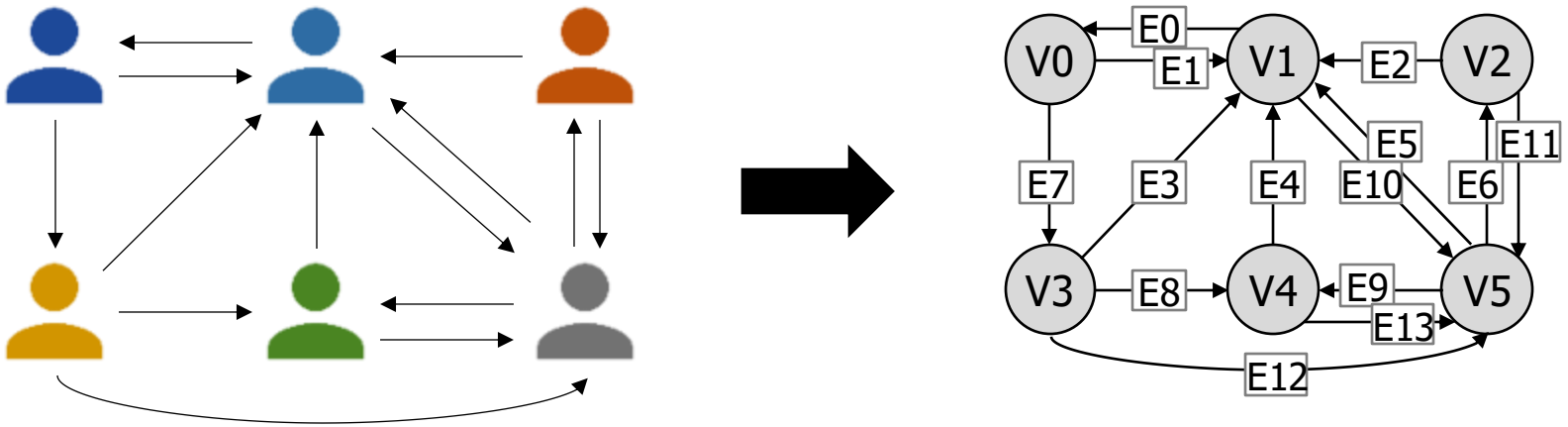
Neuroscience



Single Source Shortest Path in Map

Graph

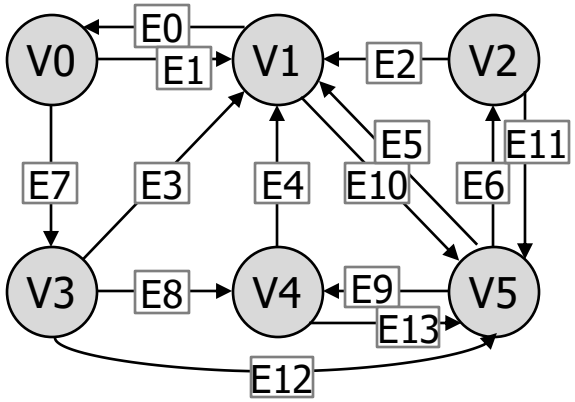
- An abstract data structure with vertices and their pairs (edges)
- Graph = (Vertex, Edge)



Friendship Relation **Graph**

Graph Processing

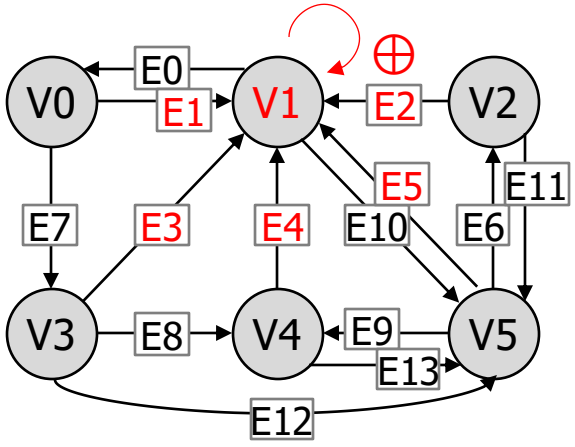
- Analyzing a graph with a given algorithm
- Computes each vertex value with its neighboring edges and vertices
- Graph processing = Algorithm + Schedule + Topology layout



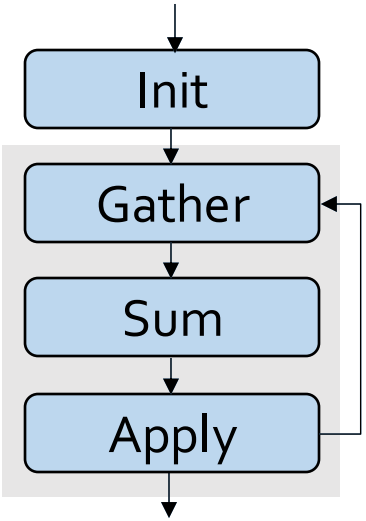
Graph

Graph Processing : Algorithm

- Graph processing = **Algorithm** + Schedule + Topology layout
- Algorithm: **how to process a graph**
- Gather data from neighbors, Accumulate the data, and Update vertex



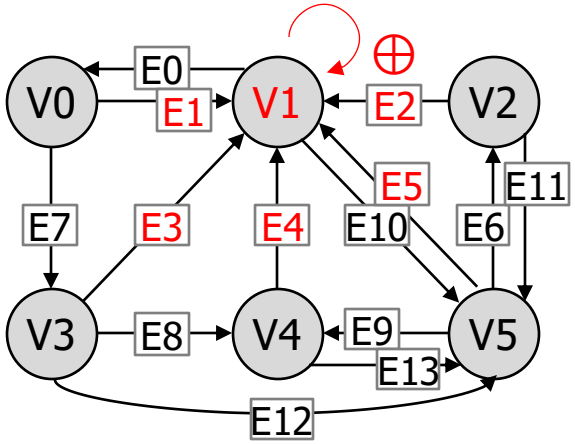
Graph



Algorithm

Graph Processing : Schedule

- Graph processing = Algorithm + **Schedule** + Topology layout
- Schedule: **How to execute the algorithm**
- Determine which thread processes which parts in what order



Graph

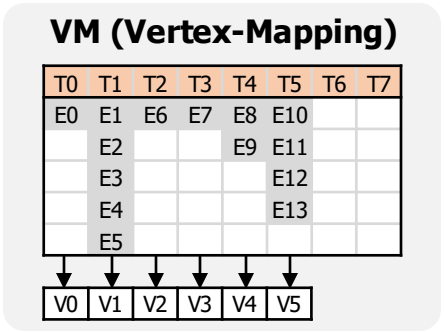
T0	T1	T2	T3	T4	T5
E0	E1	E6	E7	E8	E11
	E2			E9	E12
	E3			E10	E13
	E4				
	E5				

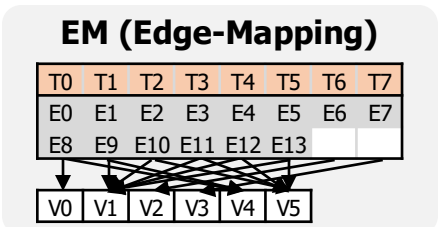
Time

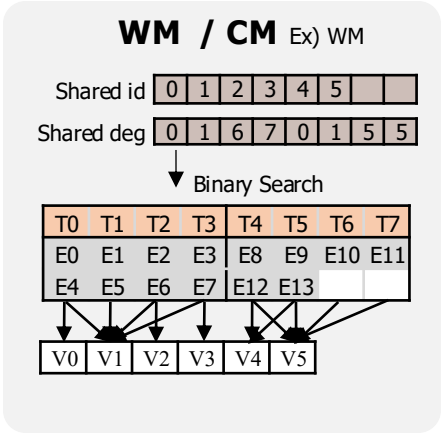
Schedule (Vertex-Mapping)

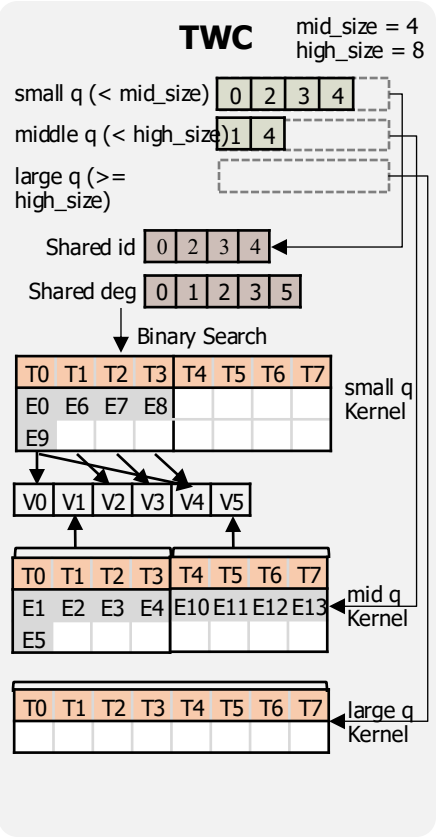
Graph Processing : Schedule

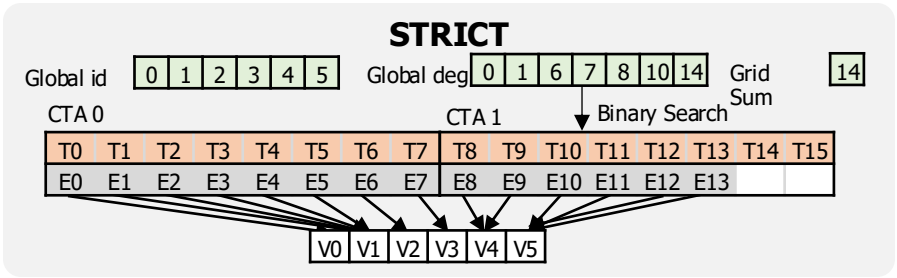
- Depending on how to fetch edges, there are various schedules

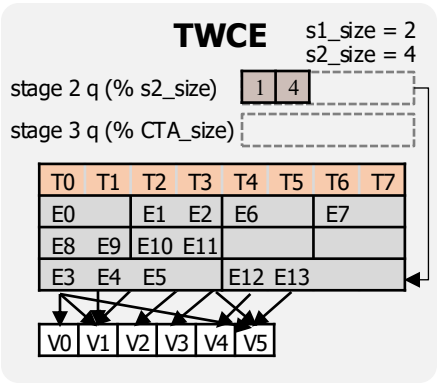






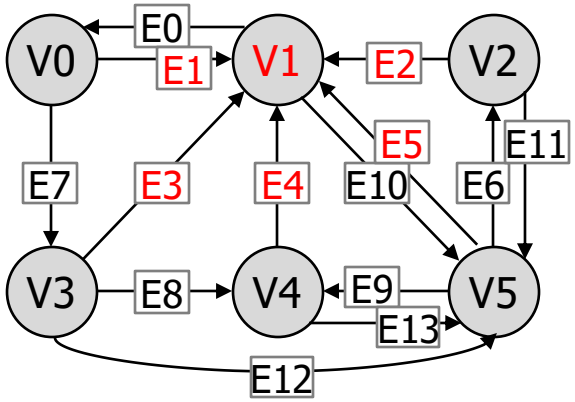




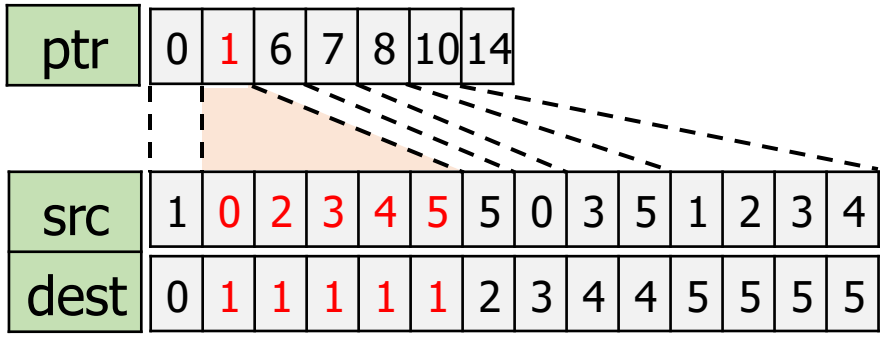


Graph Processing : Topology Layout

- Graph processing = Algorithm + Schedule + **Topology layout**
- Topology layout: **how to store a graph topology in GPU memory**
- Store edges (sources and destinations)



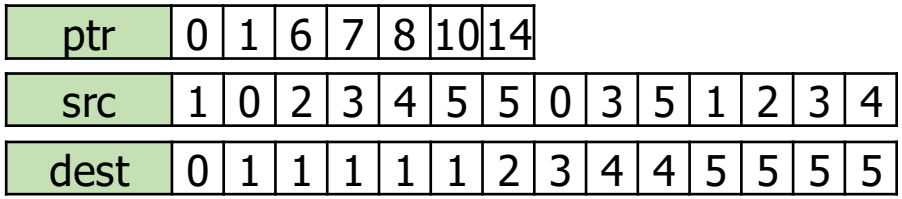
Graph



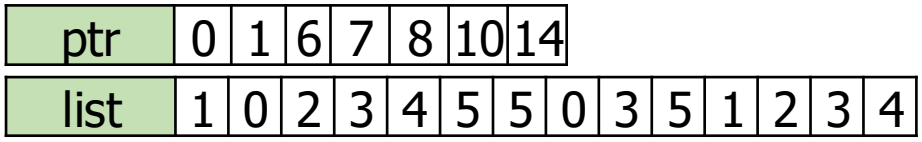
Topology layout (COO)

Graph Processing : Topology Layout

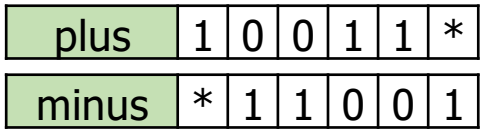
- Depending on how to store edges and how a vertex points its neighboring edges, there are various topology layouts

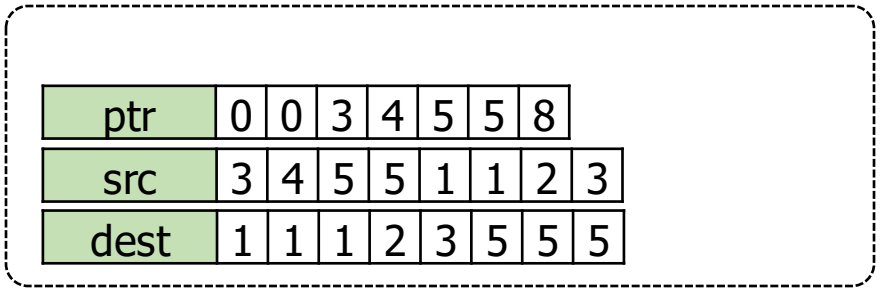


COO

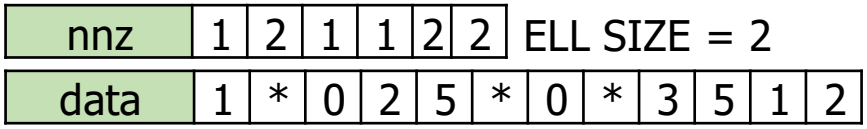


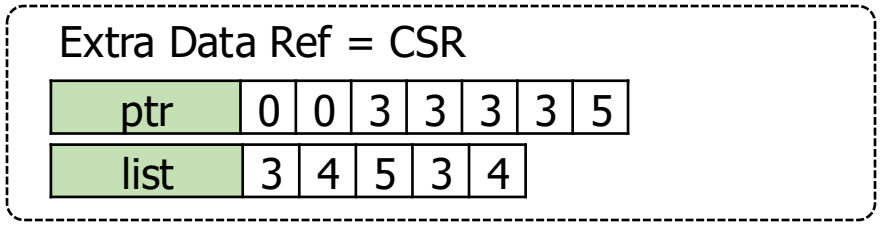
CSR/CSC





DIA

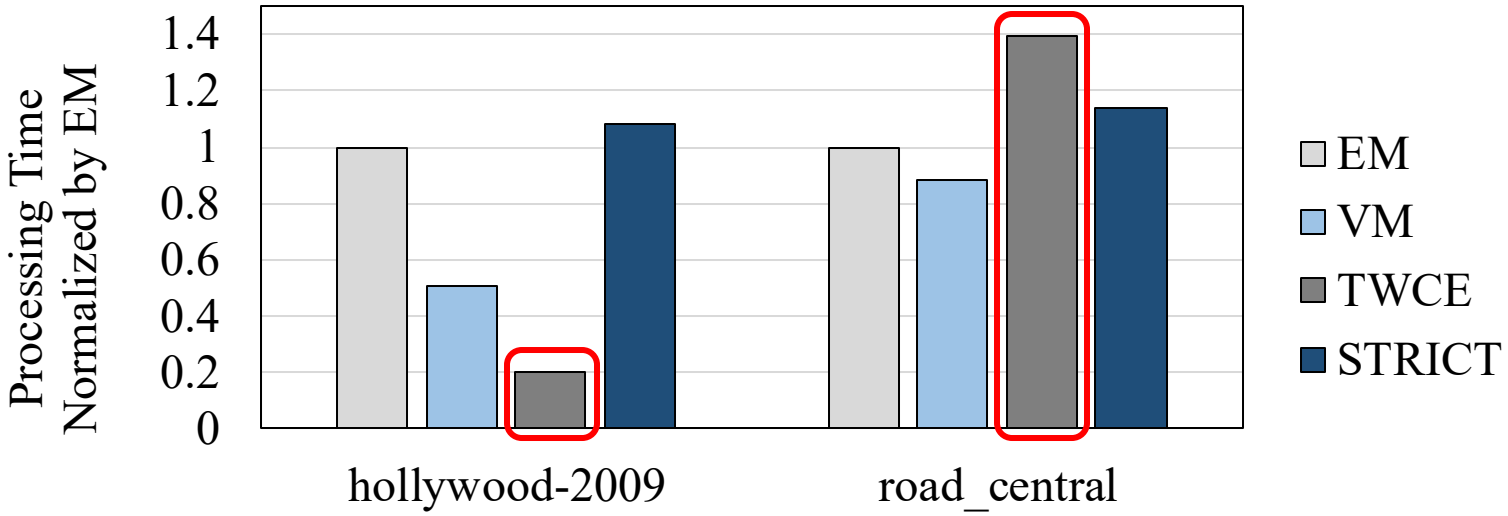




ELL

Tuning Schedule & Topology Layout

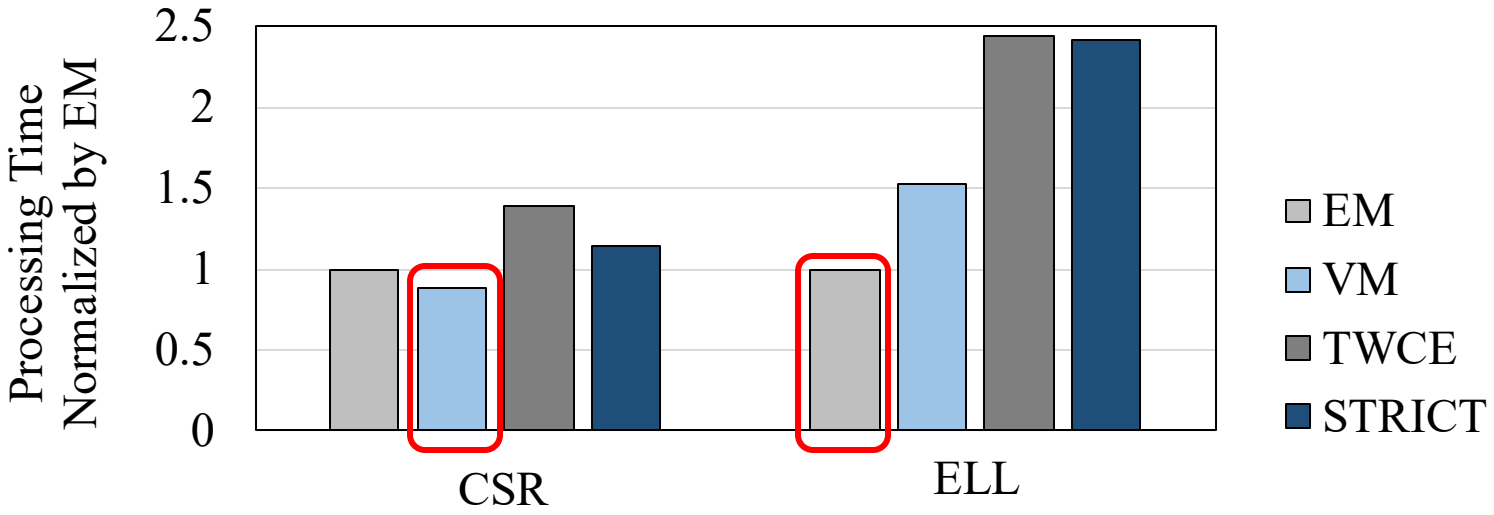
- Dataset affects the optimal schedule
- TWCE is best for Hollywood-2009, but worst for Road_central



Tuning a schedule is required for each dataset!

Tuning Schedule & Topology Layout

- Topology layout affects the optimal schedule
- VM is best for CSR, but EM is best for ELL



Schedule and topology layout should be considered together in tuning!

Existing Graph Processing Model

- Only **algorithm** is decoupled (**gather**, **sum**, **apply**)
- **Schedule** and **Topology layout** are tightly **coupled** to the model

```
1  foreach tid in (0, #Thread){
2    for eid in (tid, #Edge, #Thread){
3      src = coo.srclist[eid]
4      dest = coo.destlist[eid]
5      if sum(dest, gather(src, dest, eid))
6        activate(dest)
7    }}
8    foreach vid in (0, #Vertex){
9      apply(vid)
10   }
```

T0	T1	T2	T3	T4	T5	T6	T7
E0	E1	E2	E3	E4	E5	E6	E7
E8	E9	E10	E11	E12	E13		

Time

Edge Mapping

ptr	0	1	6	7	8	10	14
src	1	0	2	3	4	5	5	0	3	5	1	2	3	4
dest	0	1	1	1	1	1	2	3	4	4	5	5	5	5

COO

Schedule : Edge Mapping
Topology Layout : COO
Algorithm : Abstract API (gather,sum,apply)

Existing Work : When to change Topology Layout

```
1  foreach tid in (0, #Thread){
2    for eid in (tid, #Edge, #Thread){
3       src = coo.srclist[eid]
4       dest = coo.destlist[eid]
5       if sum(dest, gather(src, dest, eid))
6          activate(dest)
7    }}
8    foreach vid in (0, #Vertex){
9       apply(vid) }
```

```
1  foreach tid in (0, #Thread){
2    for eid in (tid, #Edge, #Thread){
3       vid = binarySearch(csr.ptr, eid)
4       if(vid == -1) continue
5       dest = vid
6       src = csr.list[eid]
7       if sum(dest, gather(src, dest, eid))
8          activate(dest)
9    }}
10   foreach vid in (0, #Vertex){
11      apply(vid)}
```

Schedule : Edge Mapping
Topology Layout : COO

Schedule : Edge Mapping
Topology Layout : CSR

ptr	0	1	6	7	8	10	14
src	1	0	2	3	4	5	5	0	3	5	1	2	3	4
dest	0	1	1	1	1	1	2	3	4	4	5	5	5	5

dest id 0 1 2 3 4 5

ptr	0	1	6	7	8	10	14
list	1	0	2	3	4	5	5	0	3	5	1	2	3	4

Existing Work : When to change Schedule

```
1  foreach tid in (0, #Thread){
2    for eid in (tid, #Edge, #Thread){
3      vid = binarySearch(csr.ptr, eid)
4      if(vid == -1) continue
5      dest = vid
6      src = csr.list[eid]
7      if sum(dest, gather(src, dest, eid))
8        activate(dest)
9    }}
10 foreach vid in (0, #Vertex){
11 apply(vid)
12 }
```

Schedule : Edge Mapping
Topology Layout : CSR

```
1   foreach tid in (0, #Thread){
2     for vid in (tid, #Vertex, #Thread){
3       begin = csr.offset[vid]
4       end = csr.offset[vid + 1]
5       for eid in(begin, end):
6         dest = vid
7         src = csr.list[eid]
8         if sum(dest, gather(src, dest, eid)):
9           activate(dest)
10  }}
11  foreach vid in (0, #Vertex){
12    apply(vid)
13  }
```

Schedule : Vertex Mapping
Topology Layout : CSR

Changing a schedule and layout requires to change the entire program!

Observation 1: Edge Scheduling

- Processing models **schedule edge**, and execute gather and sum

```
1  foreach tid in (0, #Thread){
2    for eid in (tid, #Edge, #Thread){
3      vid = binarySearch(csr.ptr, eid)
4      if(vid == -1) continue
5      dest = vid
6      src = csr.list[eid]
7      if sum(dest, gather(src, dest, eid))
8        activate(dest)
9    }}
10 foreach vid in (0, #Vertex){
11 apply(vid)
12 }
```

Schedule : Edge Mapping
Topology Layout : CSR

```
1    foreach tid in (0, #Thread){
2      for vid in (tid, #Vertex, #Thread){
3        begin = csr.offset[vid]
4        end = csr.offset[vid + 1]
5        for eid in(begin, end):
6          dest = vid
7          src = csr.list[eid]
8          if sum(dest, gather(src, dest, eid)):
9            activate(dest)
10   }}
11   foreach vid in (0, #Vertex){
12     apply(vid)
13   }
```

Schedule : Vertex Mapping
Topology Layout : CSR

Observation 1: Edge Scheduling

- Decouple the edge schedule step from the processing model

```
1  foreach tid in (0, #Thread){
2
3
4
5     dest = vid
6     src = csr.list[eid]
7     if sum(dest, gather(src, dest, eid))
8        activate(dest)
9  }}
10 foreach vid in (0, #Vertex){
11 apply(vid)
12 }
```

for edge (edge schedule)

Schedule : Edge Mapping
Topology Layout : CSR

```
1    foreach tid in (0, #Thread){
2
3
4
5
6        dest = vid
7        src = csr.list[eid]
8        if sum(dest, gather(src, dest, eid)):
9           activate(dest)
10   }}
11   foreach vid in (0, #Vertex){
12      apply(vid)
13   }
```

for edge (edge schedule)

Schedule : Vertex Mapping
Topology Layout : CSR

Observation 2: Topology Layout Access

- After scheduling edge, access source and destination for each edge

```
1  foreach tid in (0, #Thread){
2
3        for edge (edge schedule)
4
5     dest = vid
6     src = csr.list[eid]
7     if sum(dest, gather(src, dest, eid))
8        activate(dest)
9  }}
10 foreach vid in (0, #Vertex){
11 apply(vid)
12 }
```

Schedule : Edge Mapping
Topology Layout : CSR

```
1    foreach tid in (0, #Thread){
2
3
4          for edge (edge schedule)
5
6          dest = vid
7          src = csr.list[eid]
8          if sum(dest, gather(src, dest, eid)):
9             activate(dest)
10   }}
11   foreach vid in (0, #Vertex){
12      apply(vid)
13   }
```

Schedule : Vertex Mapping
Topology Layout : CSR

Observation 2: Topology Layout Access

- Decouple edge data access from the model

```
1  foreach tid in (0, #Thread){
2
3      for edge (edge schedule)
4
5      Get Edge Info.
6      (Topology Layout Access)
7      if sum(dest, gather(src, dest, eid))
8          activate(dest)
9  }}
10 foreach vid in (0, #Vertex){
11 apply(vid)
12 }
```

Schedule : Edge Mapping
Topology Layout : CSR

```
1    foreach tid in (0, #Thread){
2
3
4        for edge (edge schedule)
5
6        Get Edge Info.
7        (Topology Layout Access)
8            if sum(dest, gather(src, dest, eid)):
9                activate(dest)
10   }}
11   foreach vid in (0, #Vertex){
12       apply(vid)
13   }
```

Schedule : Vertex Mapping
Topology Layout : CSR

Observation 3: Vertex Scheduling

- Apply is executed for each vertex (Vertex Scheduling)

```
1  foreach tid in (0, #Thread){
2
3      for edge (edge schedule)
4
5      Get Edge Info.
6      (Topology Layout Access)
7      if sum(dest, gather(src, dest, eid))
8          activate(dest)
9  }}
10 foreach vid in (0, #Vertex){
11 apply(vid)
12 }
```

Schedule : Edge Mapping
Topology Layout : CSR

```
1  foreach tid in (0, #Thread){
2
3      for edge (edge schedule)
4
5
6      Get Edge Info.
7      (Topology Layout Access)
8          if sum(dest, gather(src, dest, eid)):
9              activate(dest)
10 }}
11 foreach vid in (0, #Vertex){
12     apply(vid)
13 }
```

Schedule : Vertex Mapping
Topology Layout : CSR

Observation 3: Vertex Scheduling

- Decouple the vertex schedule step from the processing model

```
1  foreach tid in (0, #Thread){
2
3      for edge (edge schedule)
4
5      Get Edge Info.
6      (Topology Layout Access)
7      if sum(dest, gather(src, dest, eid))
8          activate(dest)
9  }}
10 foreach vertex (vertex schedule)
11 apply(vid)
12 }
```

Schedule : Edge Mapping
Topology Layout : CSR

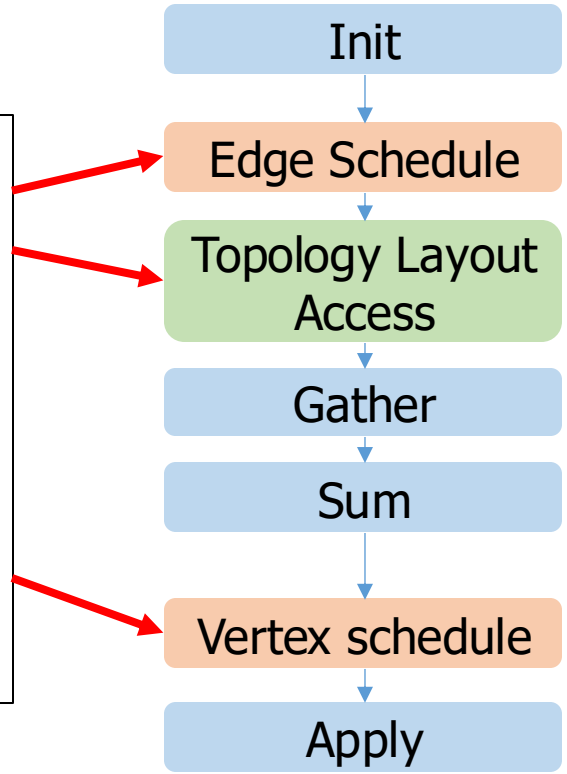
```
1  foreach tid in (0, #Thread){
2
3      for edge (edge schedule)
4
5
6      Get Edge Info.
7      (Topology Layout Access)
8          if sum(dest, gather(src, dest, eid)):
9              activate(dest)
10 }}
11 foreach vertex (vertex schedule)
12     apply(vid)
13 }
```

Schedule : Vertex Mapping
Topology Layout : CSR

New processing model

- Schedule and topology layout are decoupled from processing model

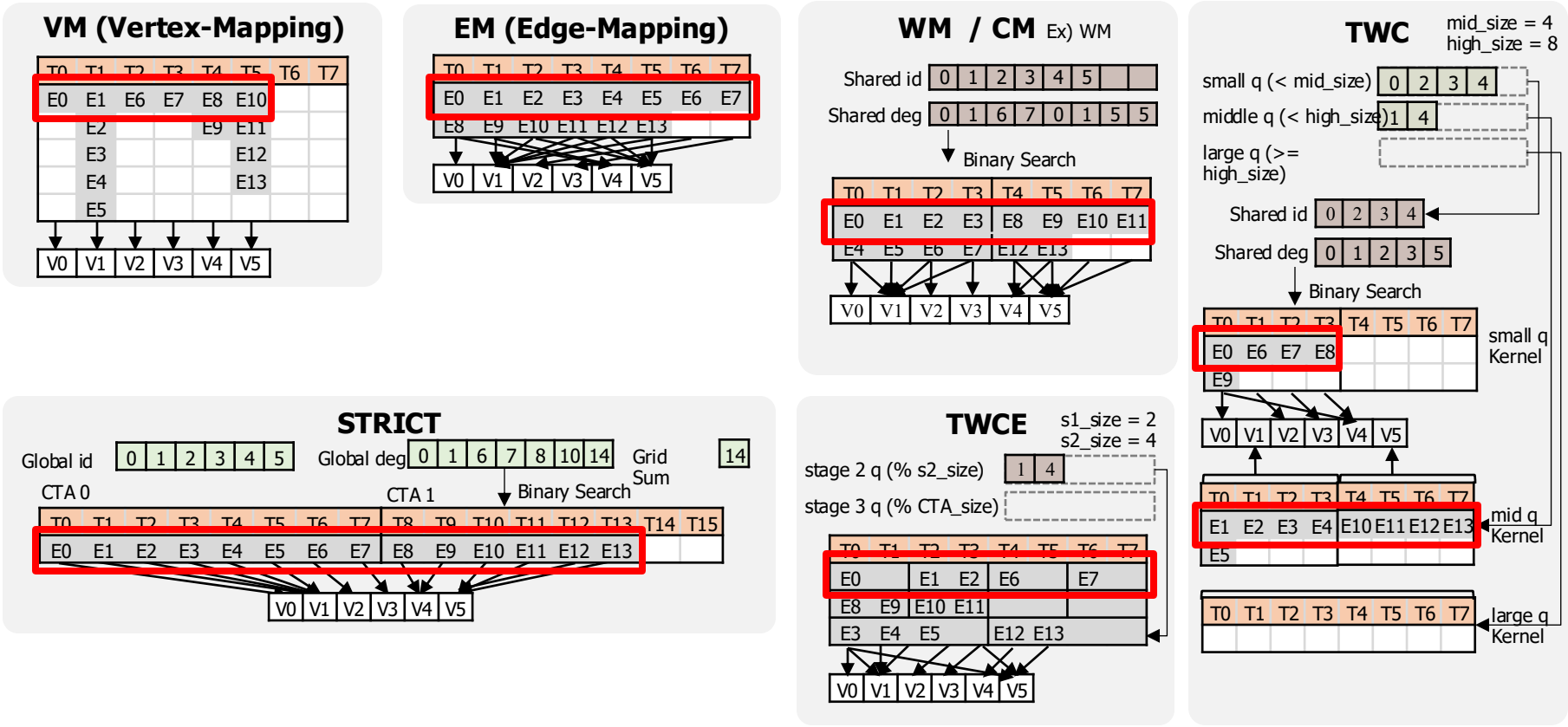
```
1   foreach tid in (0, #Workload){
2    while(getNextEdgeID(tid, data, vid, eid)){
3          getEdge(vid, eid, src, dest)
5          if sum(dest, gather(src, dest, eid))
6             activate(dest)
7          }
8      }
9    }
10   foreach tid in (0, #Vertex){
11     apply(tid)
12  }
```



New Processing Model **New Processing Steps**

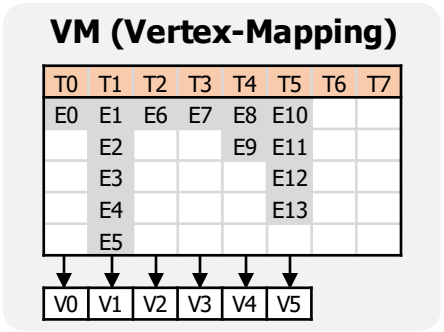
Schedule Abstraction

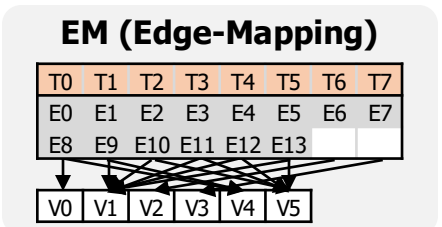
- Edge scheduling Abstraction: ***getNextEdgeID***
  - Return edge id for each iteration for each thread

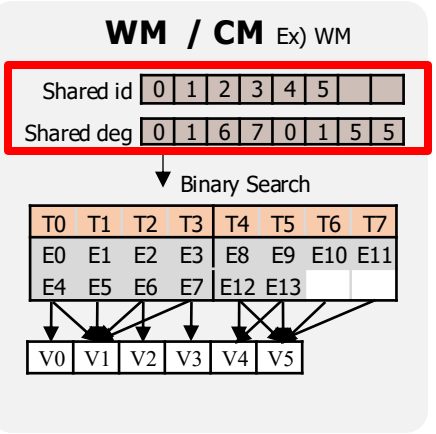


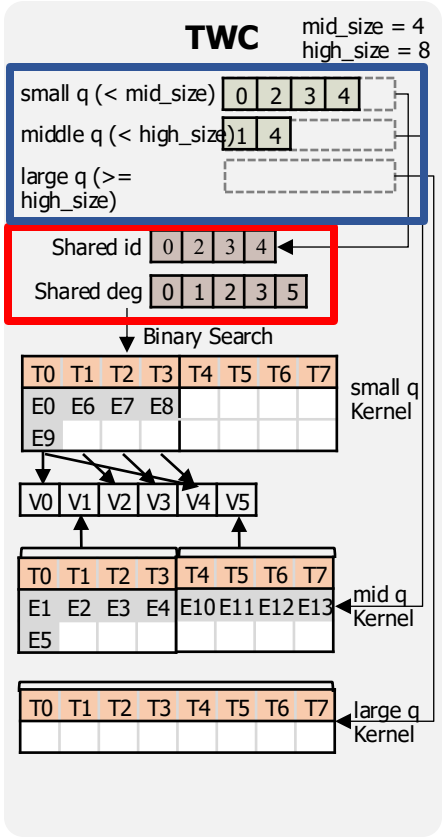
Schedule Abstraction

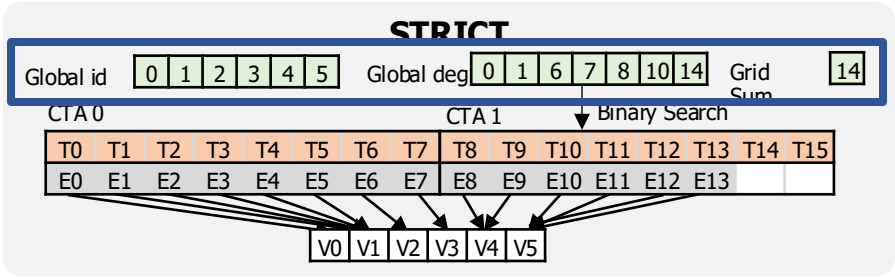
- Some schemes support load balancing: ***initShared***, ***initGlobal***
  - Setting **shared** or **global** **memory** before edge scheduling

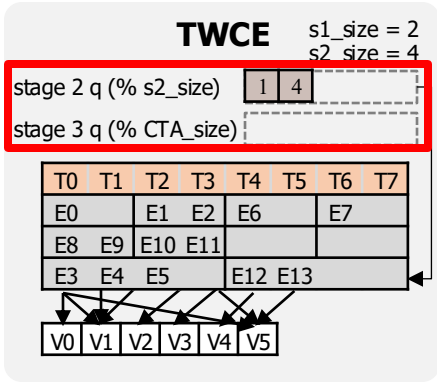












Schedule Abstraction

- Support different schedules w/o changing the processing model

```
1   foreach tid in (0, #Workload){
2      initShared(tid, data)
3      while(getNextEdgeID(tid, data, vid, eid)){
4         getEdge(vid, eid, src, dest)
5         if sum(dest, gather(src, dest, eid))
6            activate(dest)
7         }
8      }
9   }
10  foreach tid in (0, #Vertex){
11     apply(tid)
12  }
```

Proposed Processing Model

Vertex Mapping

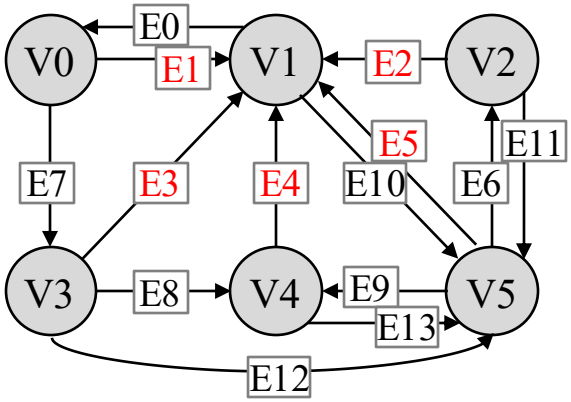
```
1   initShared (tid, data){
2      getNeighbor(tid, start, end)
3      data[0] = start
4      data[1] = end}
5   getNextEdgeID (tid, data, &vid, &eid){
6      if(data[0] >= data[1]) return false
7      vid = tid
8      eid = data[0]++
9      return true}
```

Edge Mapping

```
1   initShared (tid, data){
2      data[0] = 0}
2   getNextEdgeID (tid, data, &vid, &eid){
3      if((data[0]++) = 1) return false
4      if(searchVID(tid, vid)) return false
5      eid = tid
6      return true}
```

Topology Layout Abstraction

- Topology layout abstraction interfaces
  - ***getEdge*** : Return source and destination vertex for a given edge
  - ***getNeighbor*** : Return incoming and outgoing edges for a given vertex



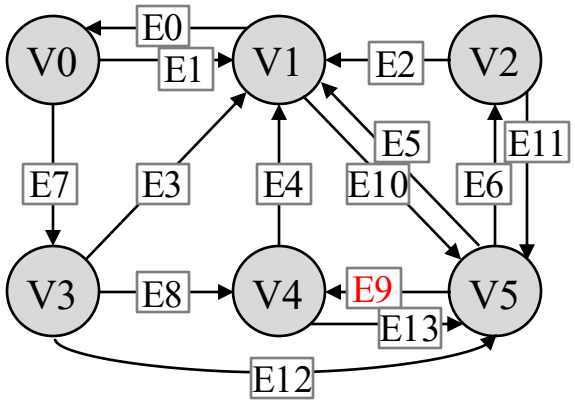
Graph

getNeighbor(v1) → e1, e2, e3, e4, e5

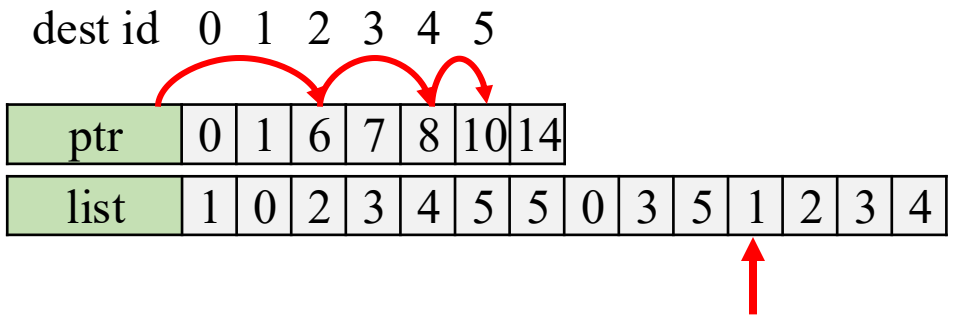
getEdge(e1) → (v0, v1)

Topology Layout Abstraction

- Some schemes (CSR) store only one of source or destination ID
- For fast data access
  - Split ***getEdge*** into two steps: ***searchVID*** , ***getEdge***



Graph



CSR

Topology Layout Abstraction

- Support different schedules w/o changing the processing model

```
1   foreach tid in (0, #Workload){
2      initShared(tid, data)
3      while(getNextEdgeID(tid, data, vid, eid)){
4         getEdge(vid, eid, src, dest)
5         if sum(dest, gather(src, dest, eid))
6            activate(dest)
7         }
8      }
9   }
10  foreach tid in (0, #Vertex){
11     apply(tid)
12  }
```

Proposed Processing Model

COO

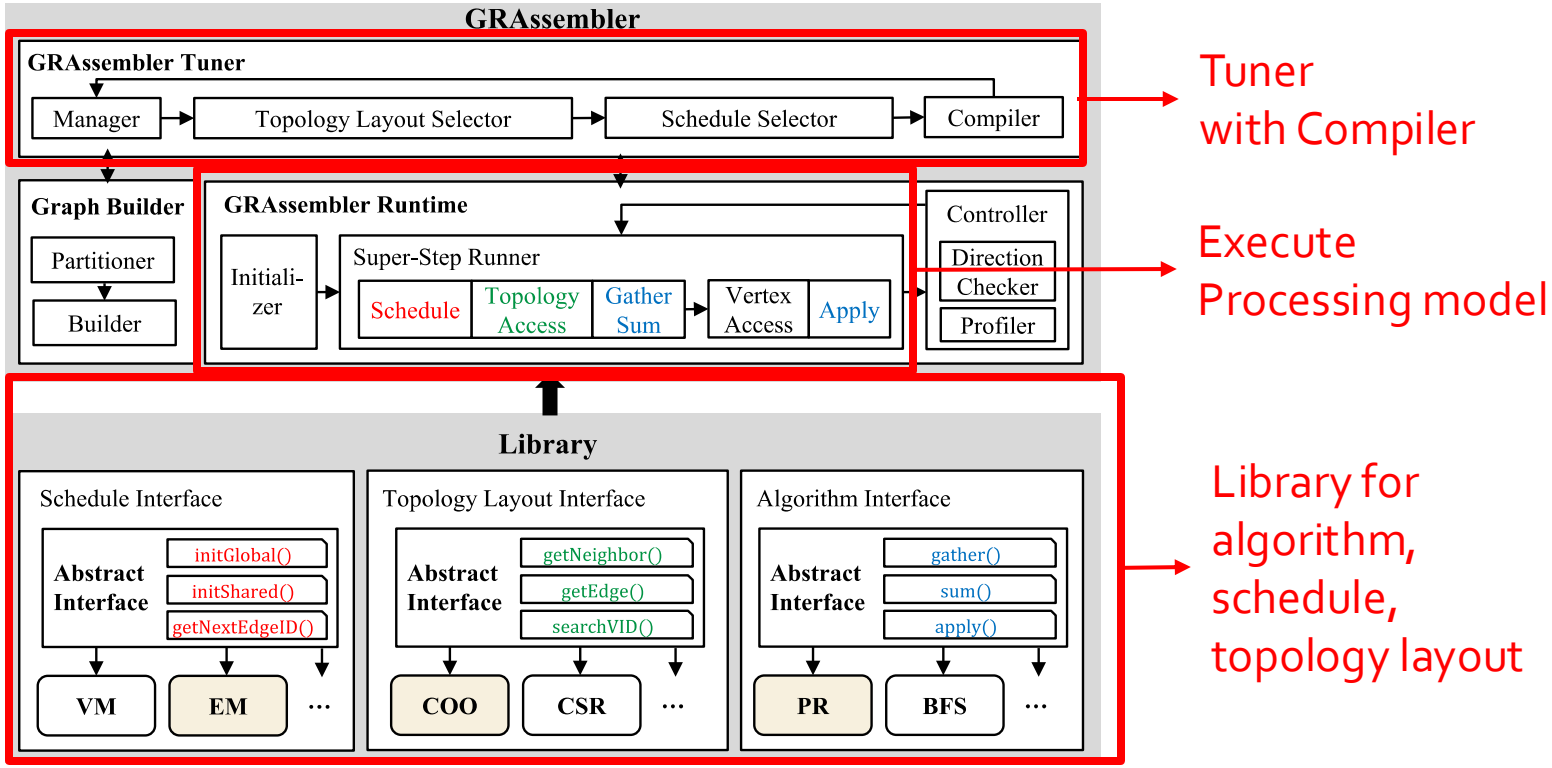
```
1  getNeighbor (vid, &start, &end){
2     src = ptr[vid]
3     dest = ptr[vid + 1]}
4  searchVID (eid, &vid, &cond){
5     vid = srclist[eid]}
6  getEdge (vid, eid, &src, &dest){
7    src = srclist[eid]
8    dest = destlist[eid]}
```

CSR

```
1  getNeighbor (vid, &start, &end){
2    src = offset[vid]
3    dest = offset[vid + 1]}
4  searchVID (eid, &vid, &cond){
5     vid = binarySearch(offset, eid)}
6  getEdge (vid, eid, &src, &dest){
7     src = vid
8     dest = list[eid]}
```

GRAssembler : Graph Processing Framework

- Execute the proposed processing model
  - Integrating algorithm, schedule, and topology layout
- Optimize integrated graph processing program



Optimization: Dead Code Elimination

- Some algorithm and schedule skip some processing steps
  - Vertex-Mapping: Do nothing for initGlobal
  - BFS algorithm: Do nothing for apply
- We can eliminate processing step functions if not used

```
1  initGlobal()
2  Foreach Thread tid in (0, #workload){
3       initShared(tid, data)
4       while(getNextEdgeID(tid, data, vid, eid){
5            getEdge(vid, eid, src, dest)
6.           if sum(dest, gather(src, dest, eid))
7                activate(dest)
8   }  }
9   Foreach Thread tid in (0, #Vertex){
10       apply(tid)
11  }
```

Optimization: Atomic Operation Reduction

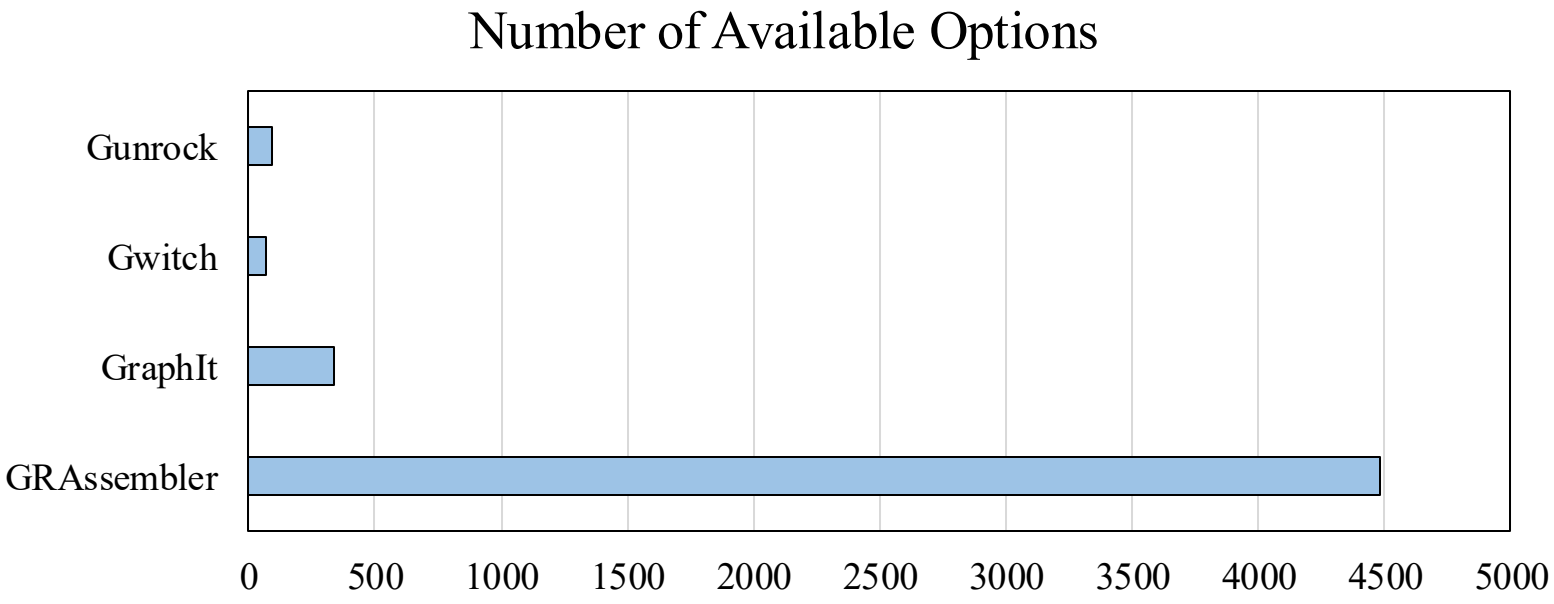
- If each vertex value is updated by only a thread, the atomic operation is not necessary
- Remove the unnecessary synchronization

```
1   sum(dest, data){
2       CAS(new_rank[dest], -1, data);
3   }
```

```
1   sum(dest, data){
2       if(new_rank[dest] != -1)
3           new_rank[dest] = data;
4   }
```

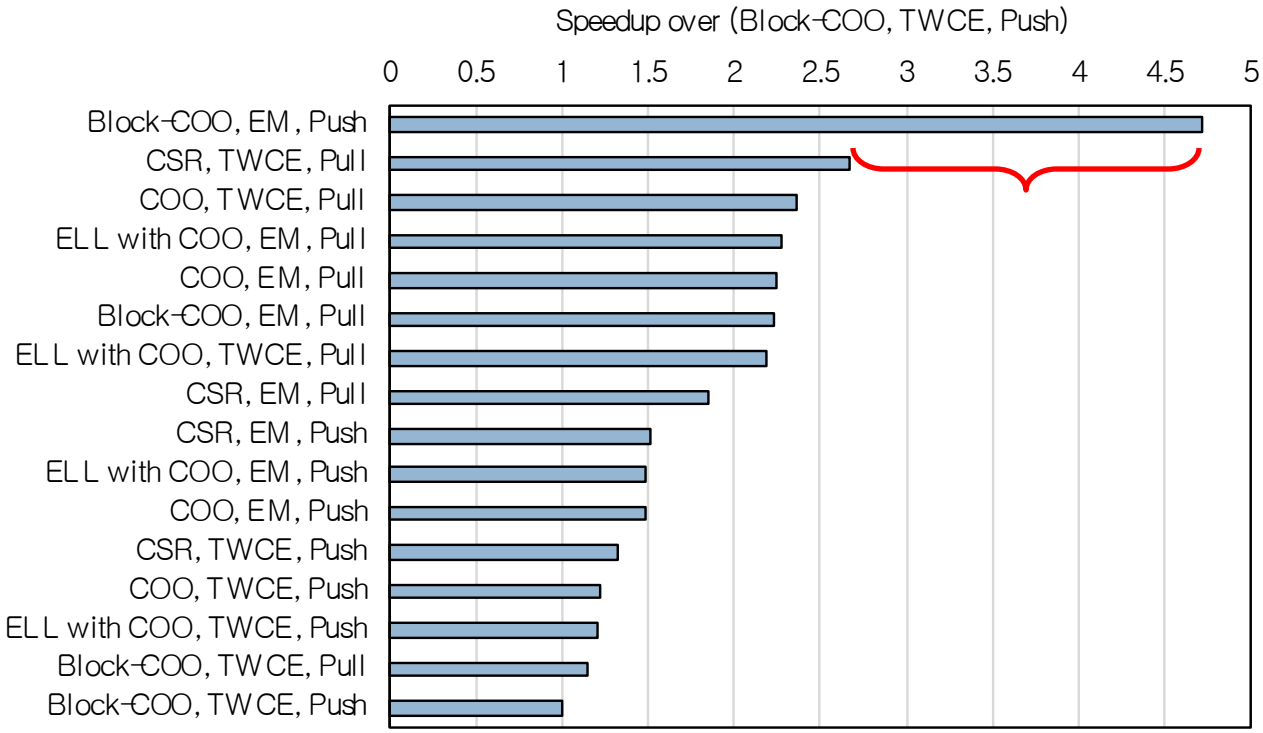
Evaluation: Tuning Space

- Decoupling schedule and topology layout increases tuning space from 336 (GraphIt) to 4480 (GRAssembler)
- GRAssembler supports additional tuning options
  - CTA Size, Blocking, Active Dataset Structure



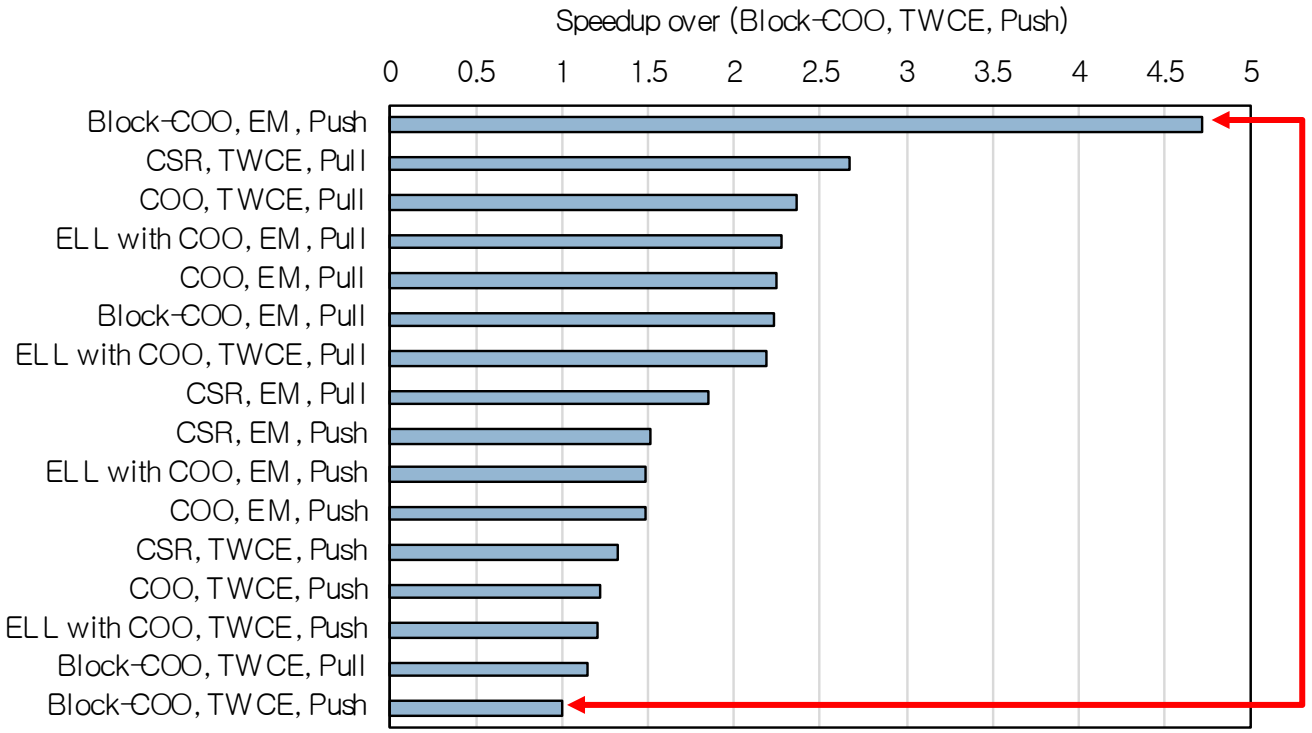
Extending Tuning Space is Critical

- Almost two times better performance than second optimal solution
- The second optimal solution use different topology layout and schedule



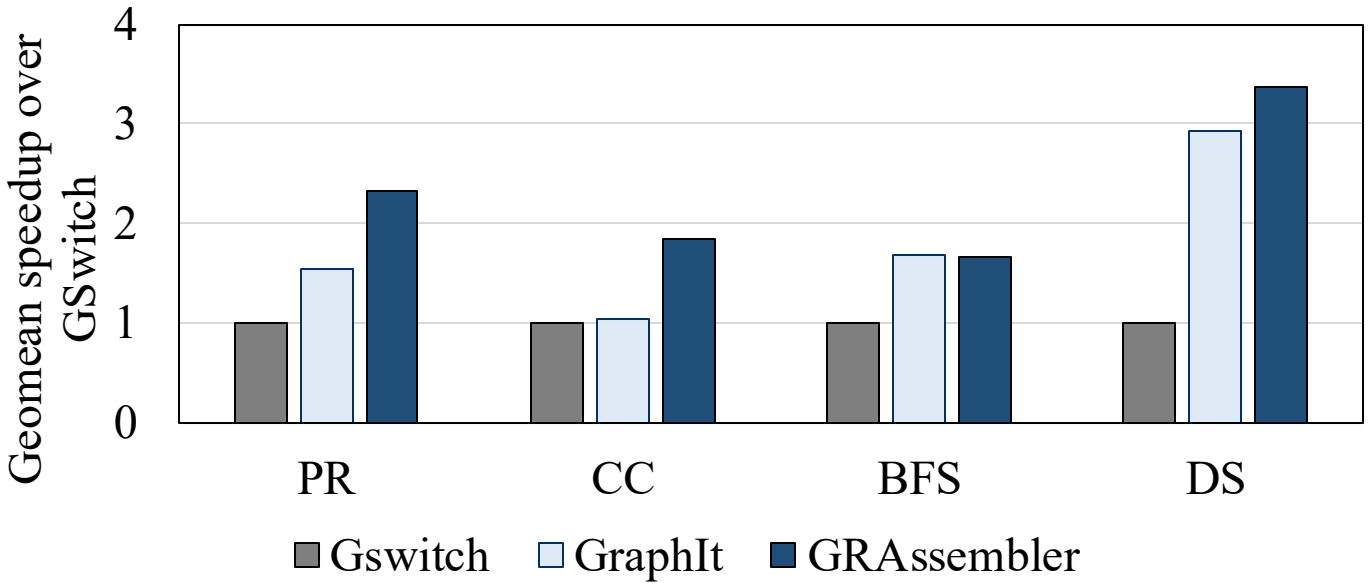
Synergistic Performance Effect

- There exist synergistic performance effects between tuning options.
- Both the worst and the best use same topology layout.



GRAssembler Performance Evaluation

- GRAssembler achieves 2.21x and 1.30x geomean speedup respectively over compared to Gswitch and GraphIt
  - GPU : NVIDIA GeForce RTX 3090



Conclusion

- Decoupling schedule and topology layout from processing model
  - Makes changing a new schedule and topology layout easier
  - Allows new combinations of existing schedules and topology layouts
  - Increases tuning spaces : 336 (GraphIt) to 4480 (GRAssembler)

- GRAssembler: a new prototype graph processing framework
  - Supports the new processing model that decouples schedule and topology layout
  - Optimizes integrated graph processing program

- Increased tuning space yields 30% speedup compared to state-of-the-art framework

Thank You!

Shinnung Jeong, shin0403@yonsei.ac.kr