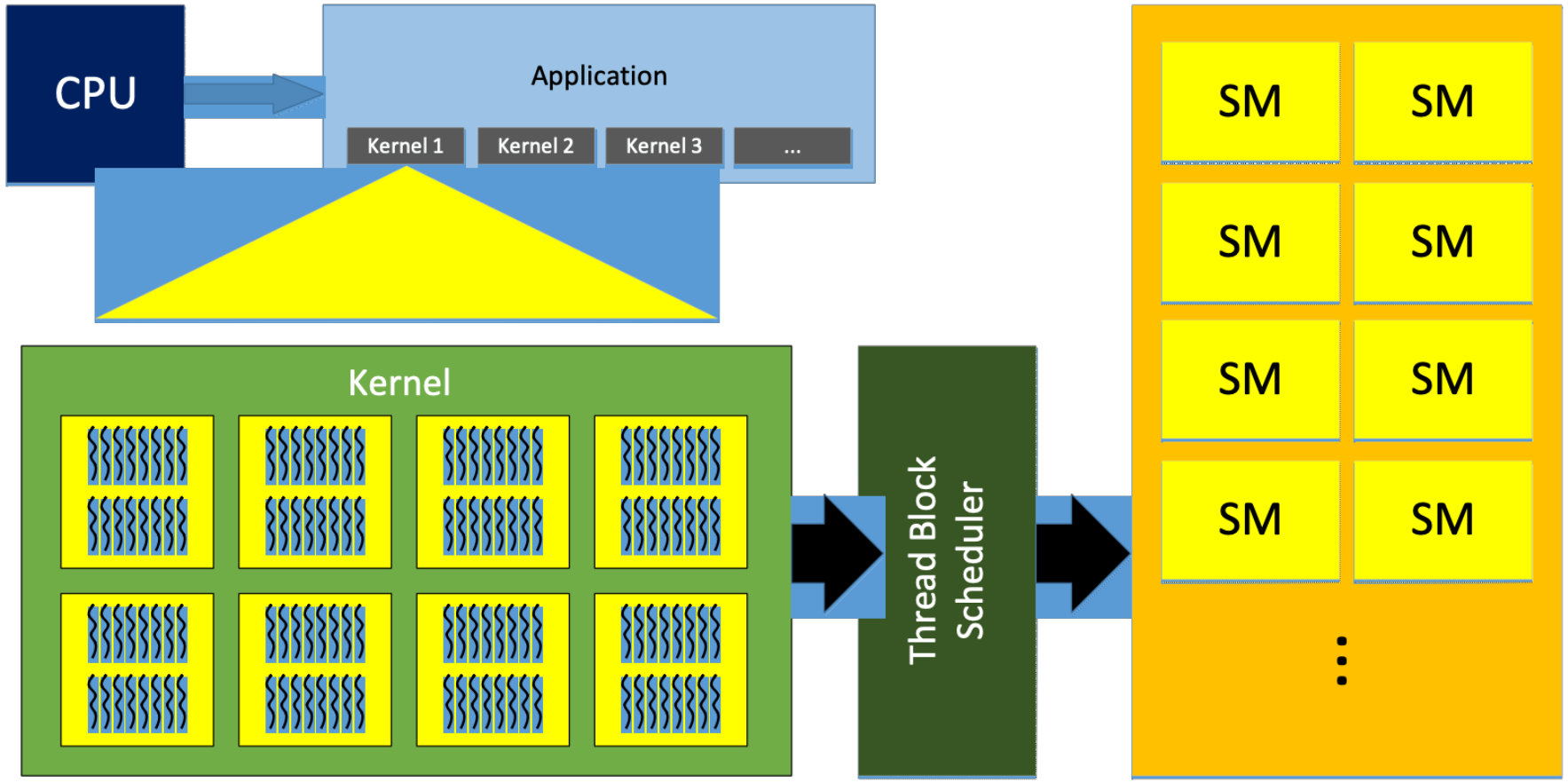
WIREFRAME: Supporting Data-dependent Parallelism through Dependency Graph Execution in GPUs

AmirAli Abdolrashidi†, Devashree Tripathy†, Mehmet E. Belviranli‡,

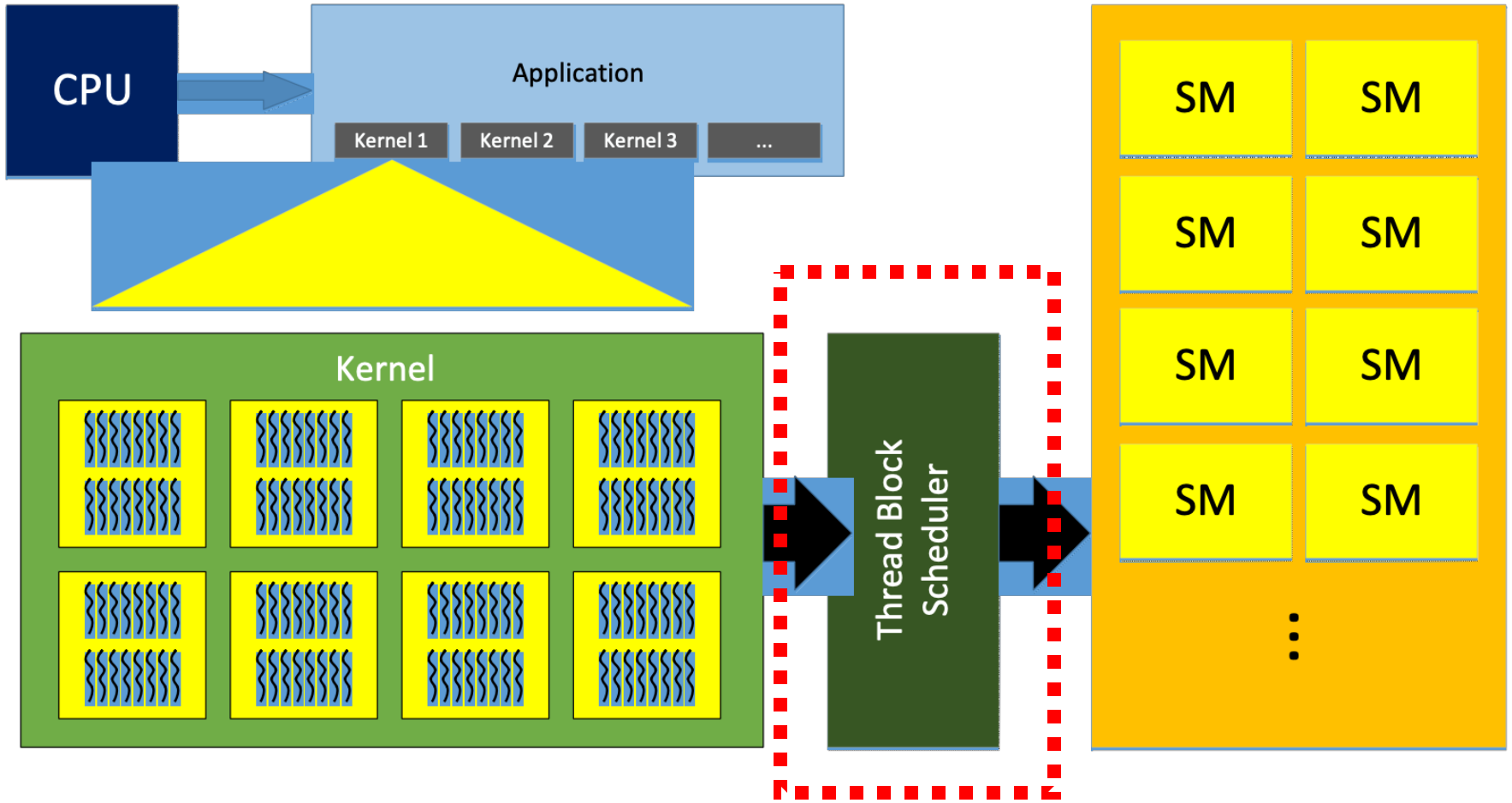
Laxmi N. Bhuyan†, Daniel Wong†

†*University of California Riverside*
‡*Oak Ridge National Laboratory*

Introduction



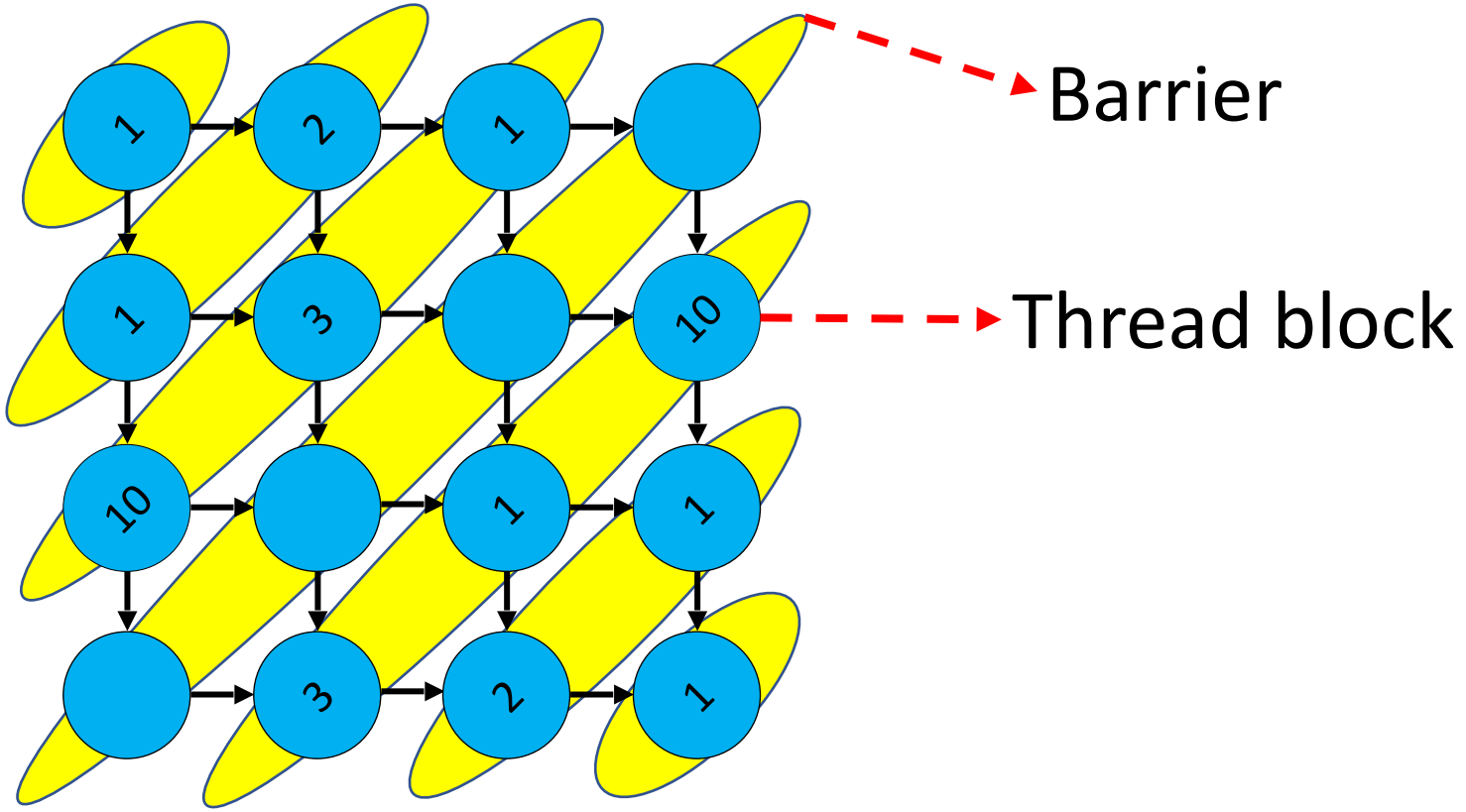
Introduction



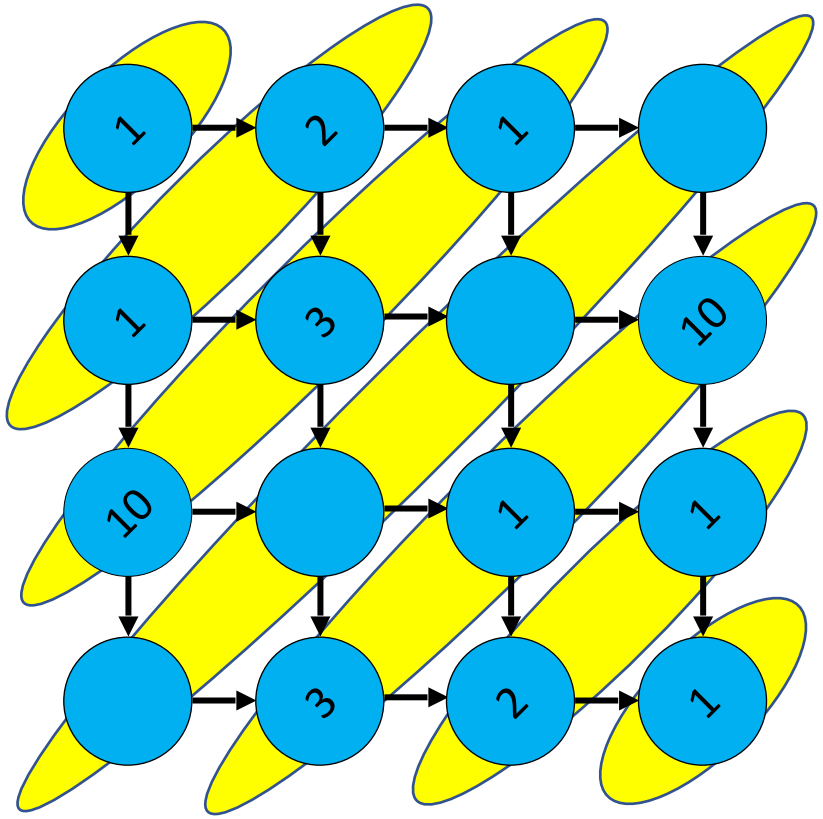
Motivation

- Despite the support for parallelism, GPUs lack support for data-dependent parallelism.

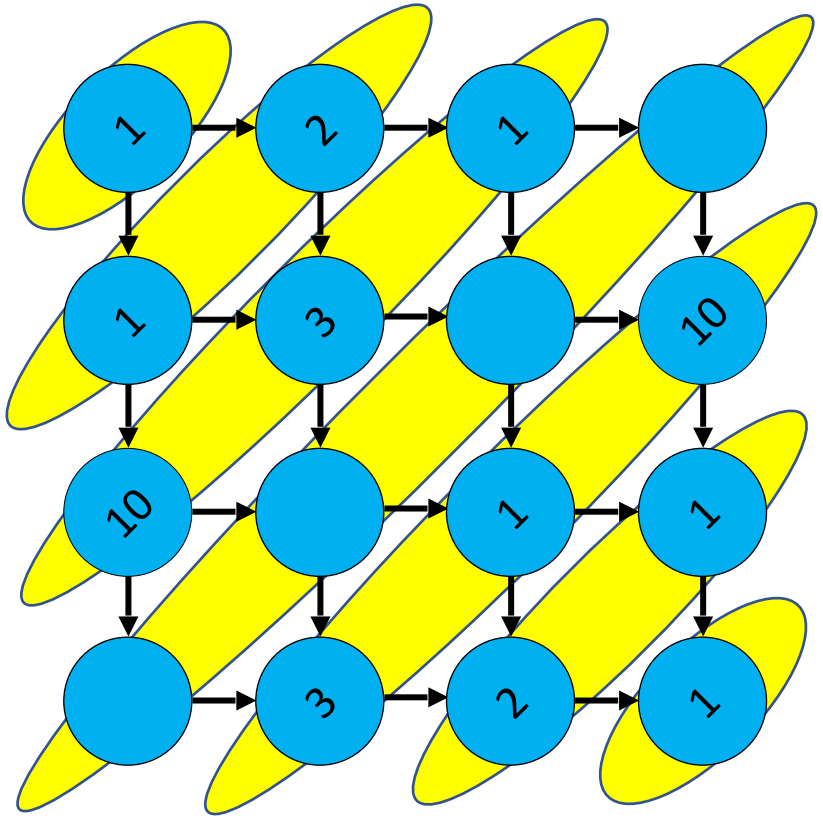
Example: Wavefront Pattern



Example: Wavefront Pattern



Example: Wavefront Pattern

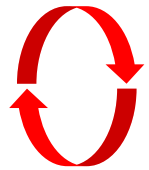


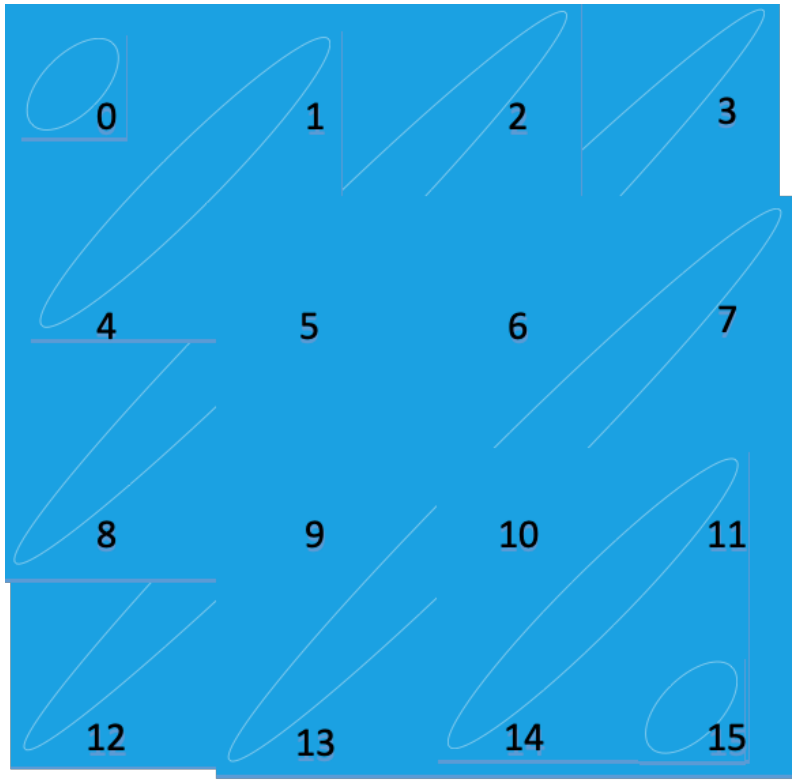
…until the application ends

Example

Global Barriers (Original)

for i = 1 to nWave:
-Kernel Launch
-Synchronize

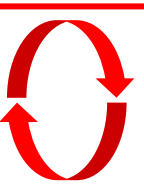




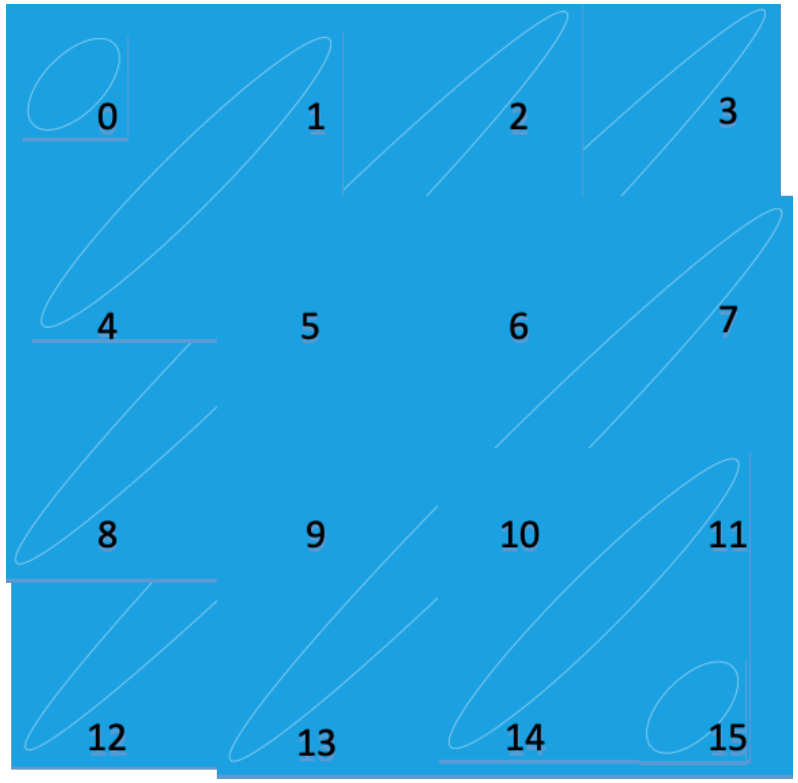
Example

Global Barriers (Original)

for i = 1 to nWave:
-Kernel Launch
-Synchronize



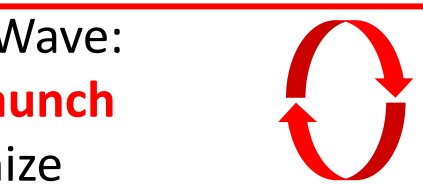
Enormous host-side kernel launch overhead!



Example

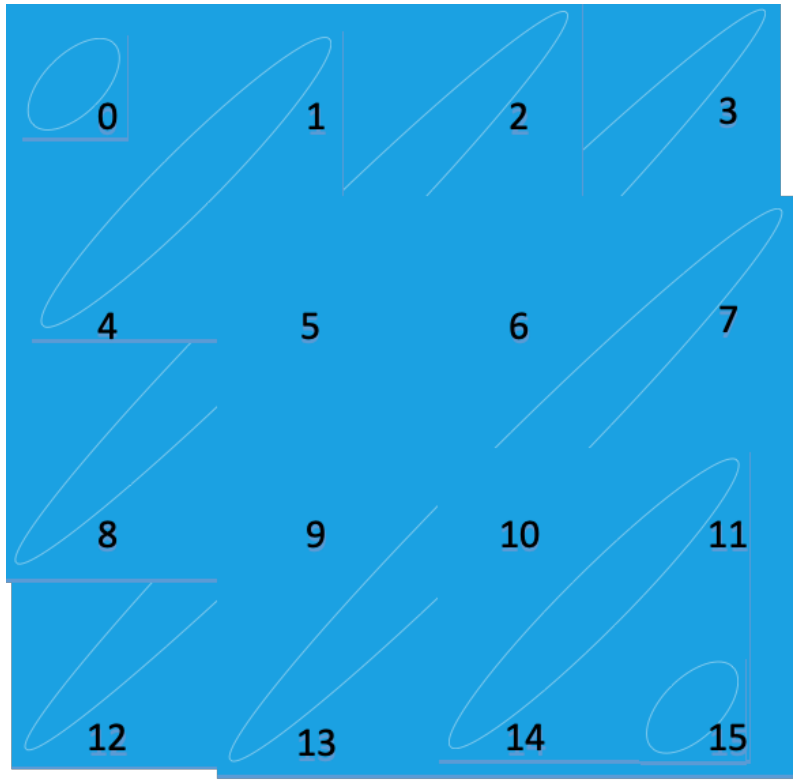
Global Barriers (Original)

for i = 1 to nWave:
-Kernel Launch
-Synchronize



Enormous host-side kernel launch overhead!

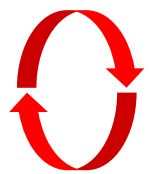
Waiting on non-parent thread blocks



Example

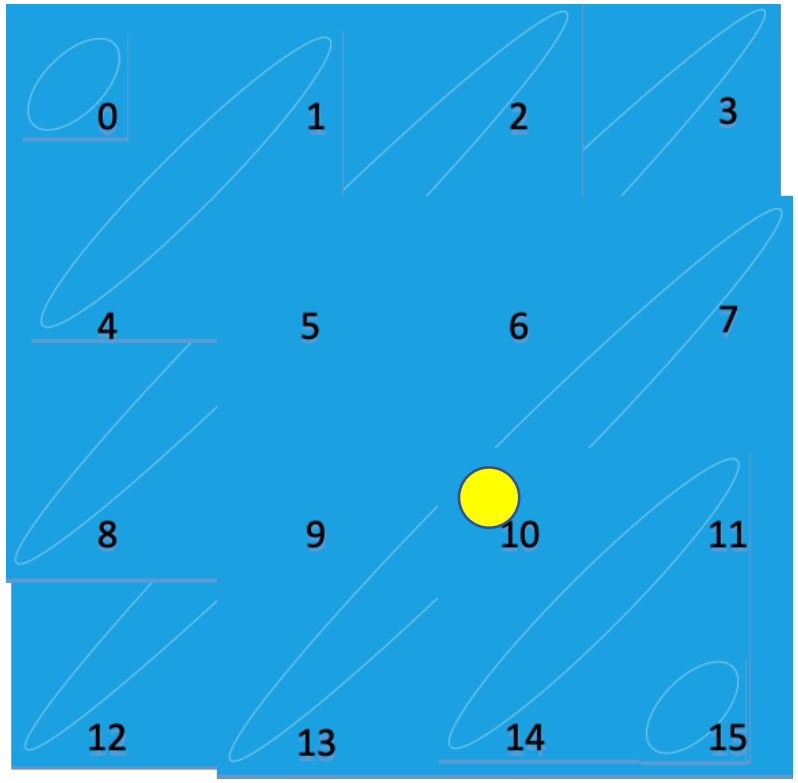
Global Barriers (Original)

for i = 1 to nWave:
-Kernel Launch
-Synchronize



Enormous host-side kernel launch overhead!

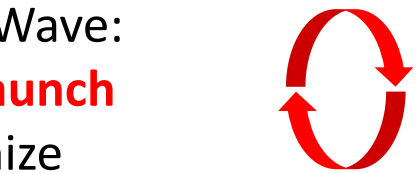
Waiting on non-parent thread blocks



Example

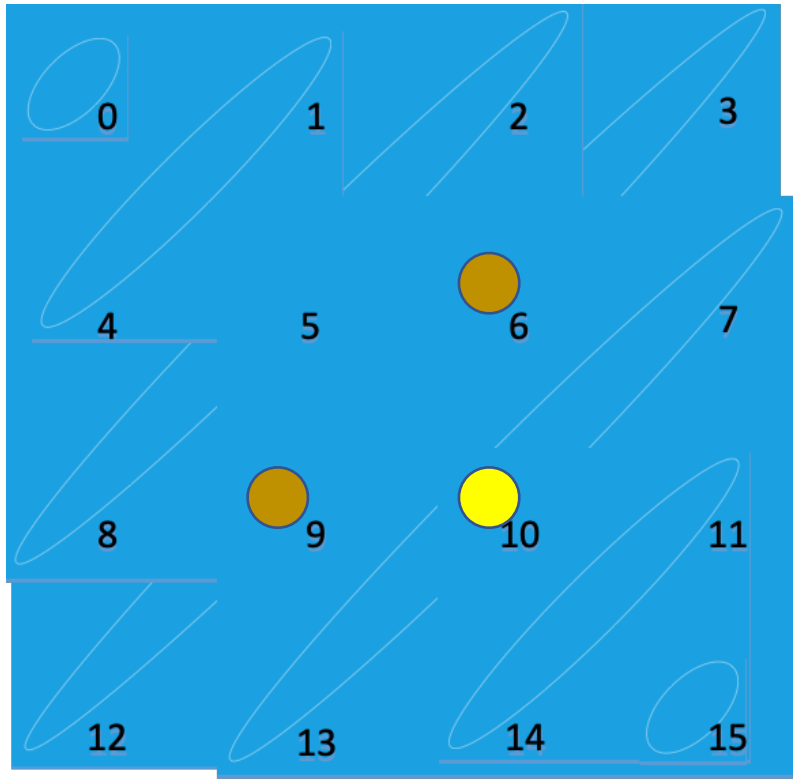
Global Barriers (Original)

for i = 1 to nWave:
-Kernel Launch
-Synchronize



Enormous host-side kernel launch overhead!

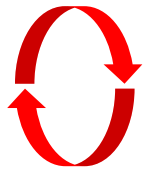
Waiting on non-parent thread blocks



Example

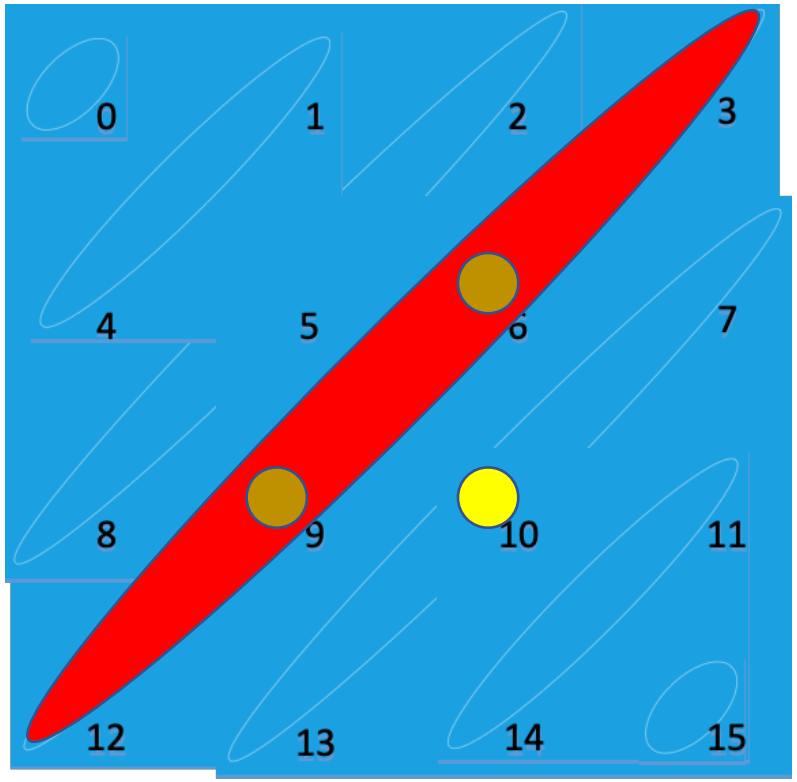
Global Barriers (Original)

for i = 1 to nWave:
-Kernel Launch
-Synchronize



Enormous host-side kernel launch overhead!

Waiting on non-parent thread blocks



Example

CDP (Nested)

RUN:

-Parent Kernel Launch

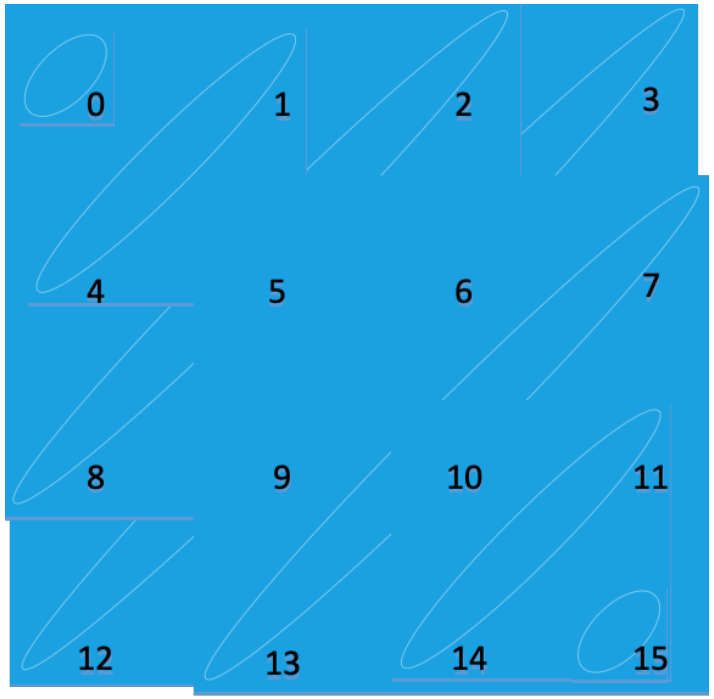
-Synchronize

Parent Kernel:

for i = 1 to nWaves:

-Child Kernel Launch

-Synchronize



Example

CDP (Nested)

RUN:

-Parent Kernel Launch

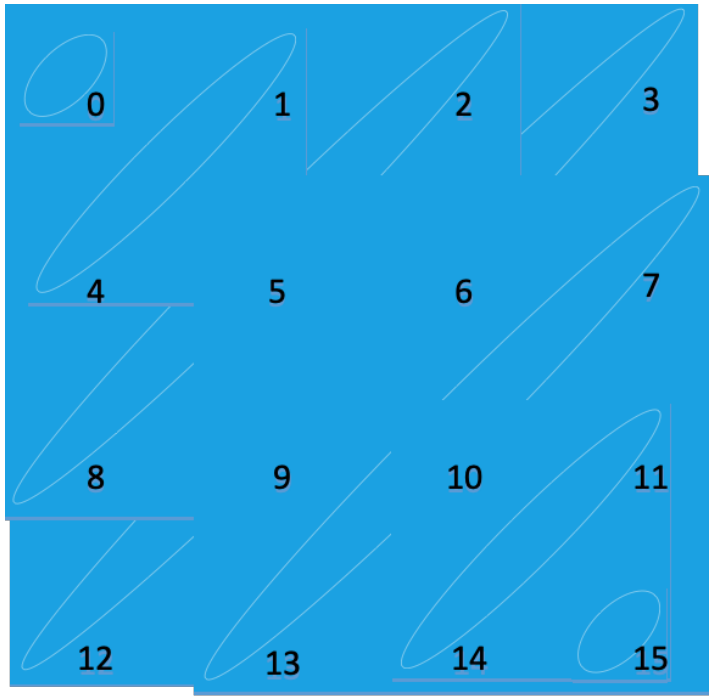
-Synchronize

Parent Kernel:

for i = 1 to nWaves:

-Child Kernel Launch

-Synchronize



Kernel Execution Pattern

Example

CDP (Nested)

RUN:

-Parent Kernel Launch

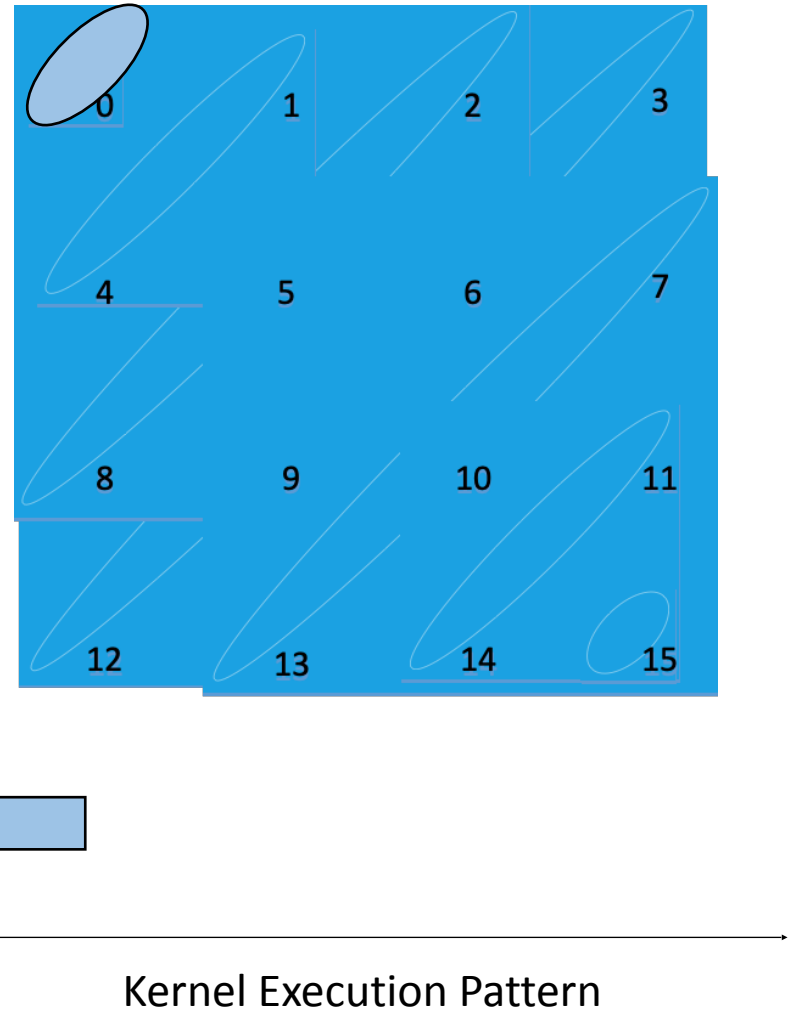
-Synchronize

Parent Kernel:

for i = 1 to nWaves:

-Child Kernel Launch

-Synchronize



Example

CDP (Nested)

RUN:

-Parent Kernel Launch

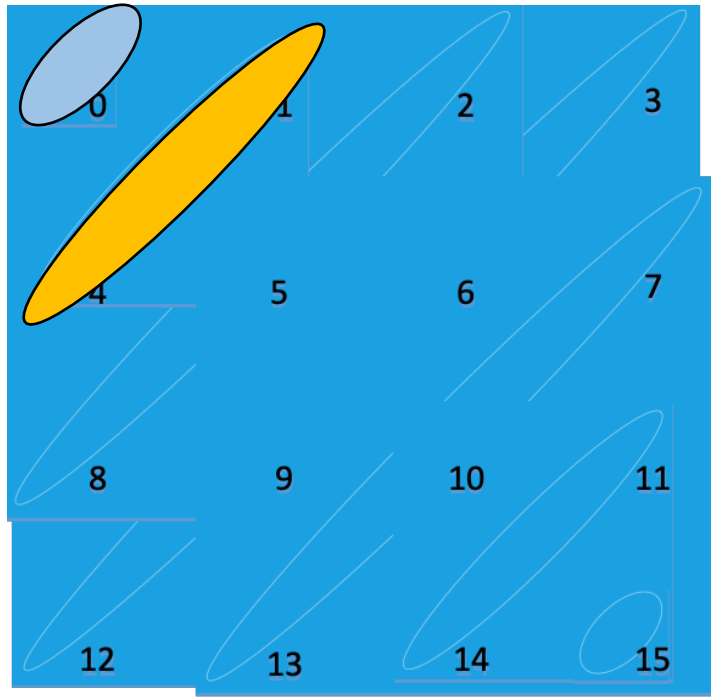
-Synchronize

Parent Kernel:

for i = 1 to nWaves:

-Child Kernel Launch

-Synchronize



Kernel Execution Pattern

Example

CDP (Nested)

RUN:

-Parent Kernel Launch

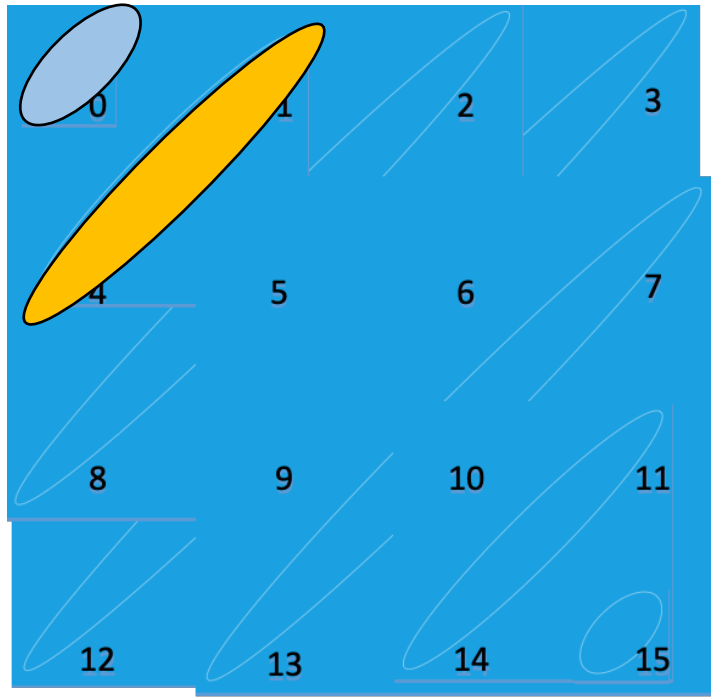
-Synchronize

Parent Kernel:

for i = 1 to nWaves:

-Child Kernel Launch

-Synchronize



Kernel Execution Pattern

Example

CDP (Nested)

RUN:

-Parent Kernel Launch

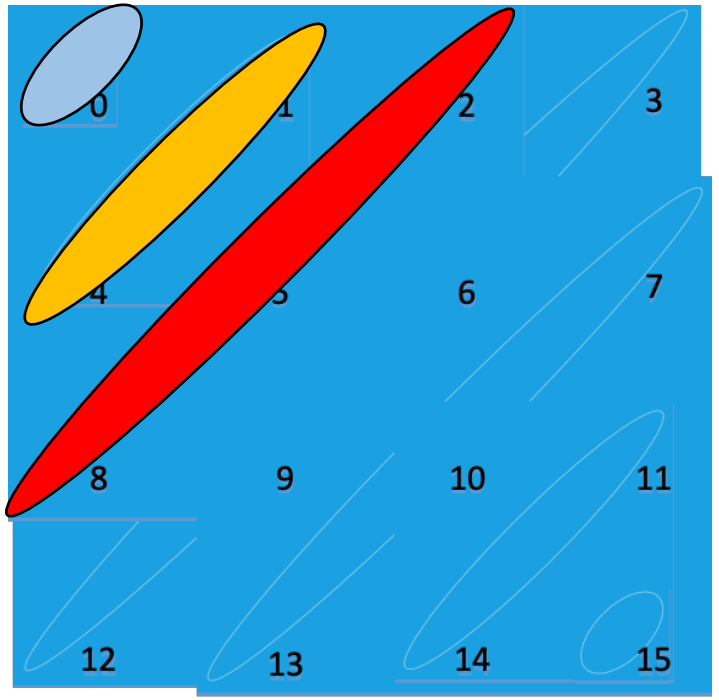
-Synchronize

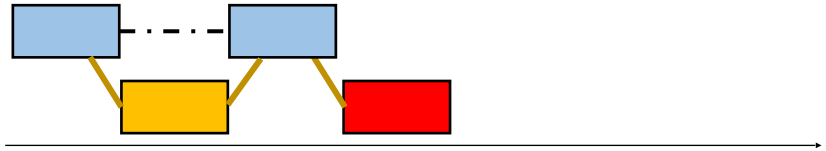
Parent Kernel:

for i = 1 to nWaves:

-Child Kernel Launch

-Synchronize





Kernel Execution Pattern

Example

CDP (Nested)

RUN:

-Parent Kernel Launch

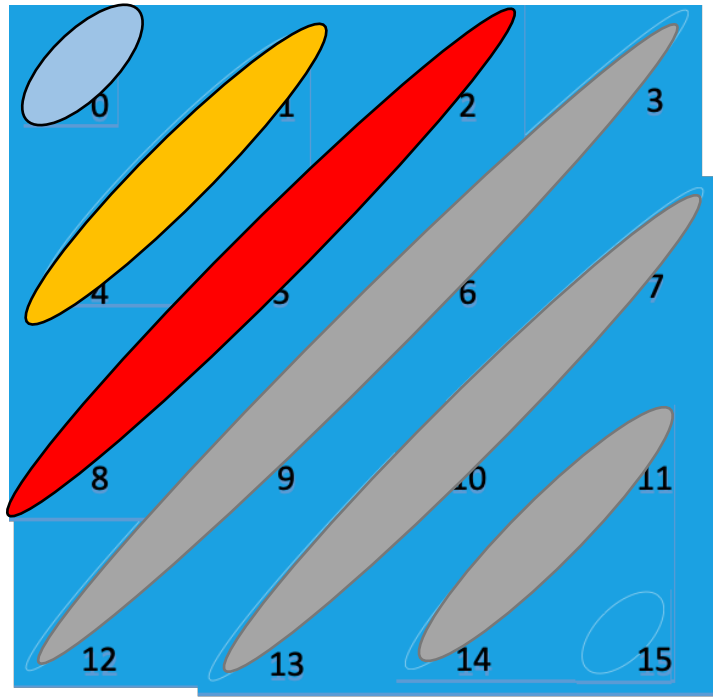
-Synchronize

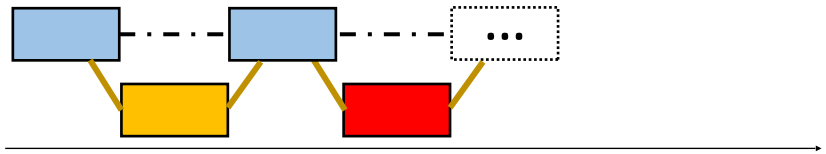
Parent Kernel:

for i = 1 to nWaves:

-Child Kernel Launch

-Synchronize





Kernel Execution Pattern

Example

CDP (Nested)

RUN:

-Parent Kernel Launch

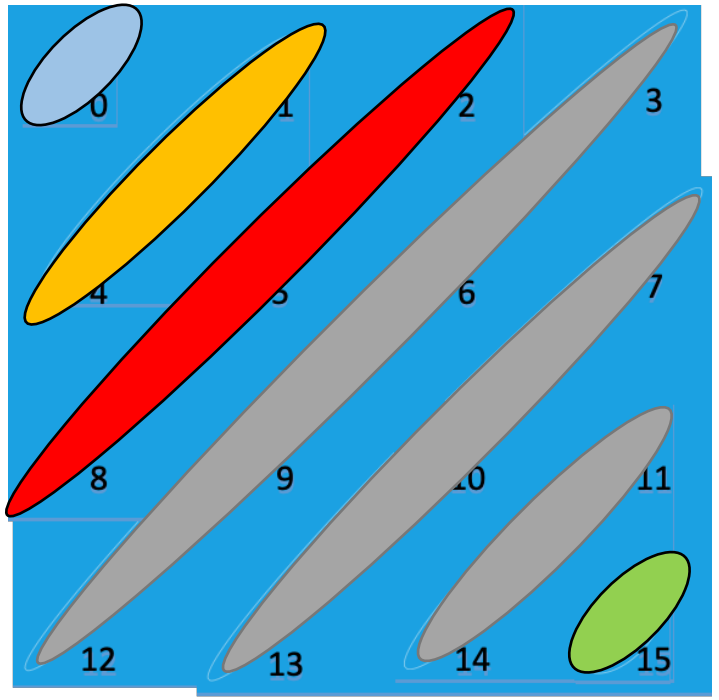
-Synchronize

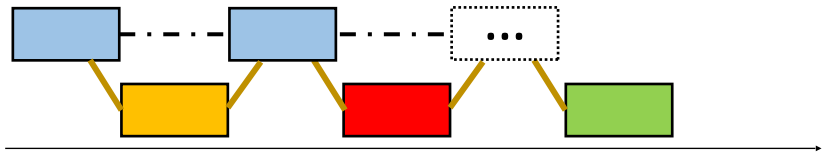
Parent Kernel:

for i = 1 to nWaves:

-Child Kernel Launch

-Synchronize





Kernel Execution Pattern

Example

CDP (Nested)

RUN:

-Parent Kernel Launch

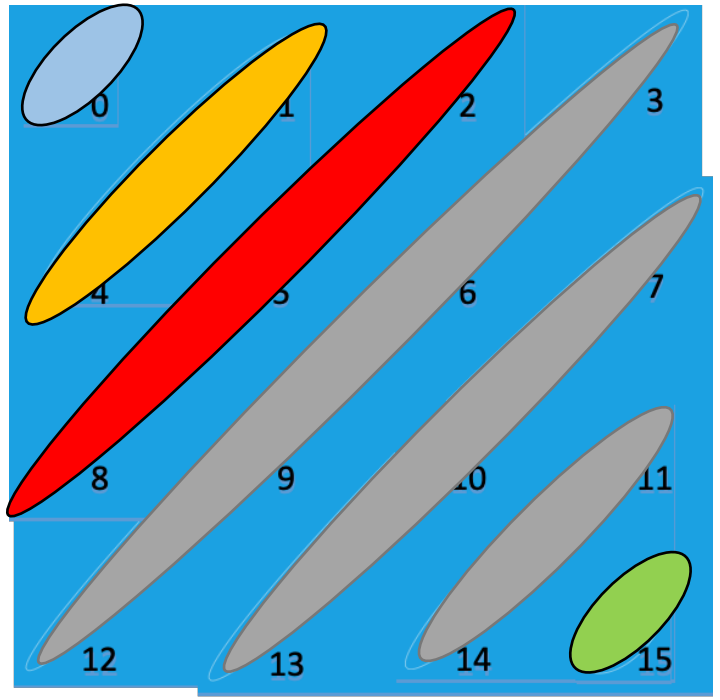
-Synchronize

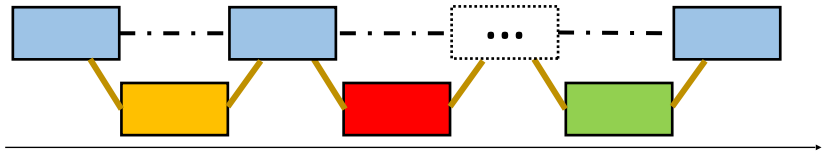
Parent Kernel:

for i = 1 to nWaves:

-Child Kernel Launch

-Synchronize





Kernel Execution Pattern

Example

CDP (Nested)

RUN:

-Parent Kernel Launch

-Synchronize

Parent Kernel:

for i = 1 to nWaves:

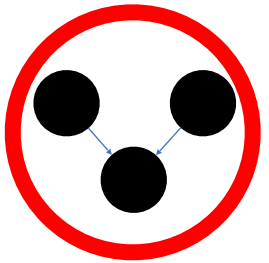
-Child Kernel Launch

-Synchronize

No more host-side kernel launch

Device-side kernel launch still has significant overhead

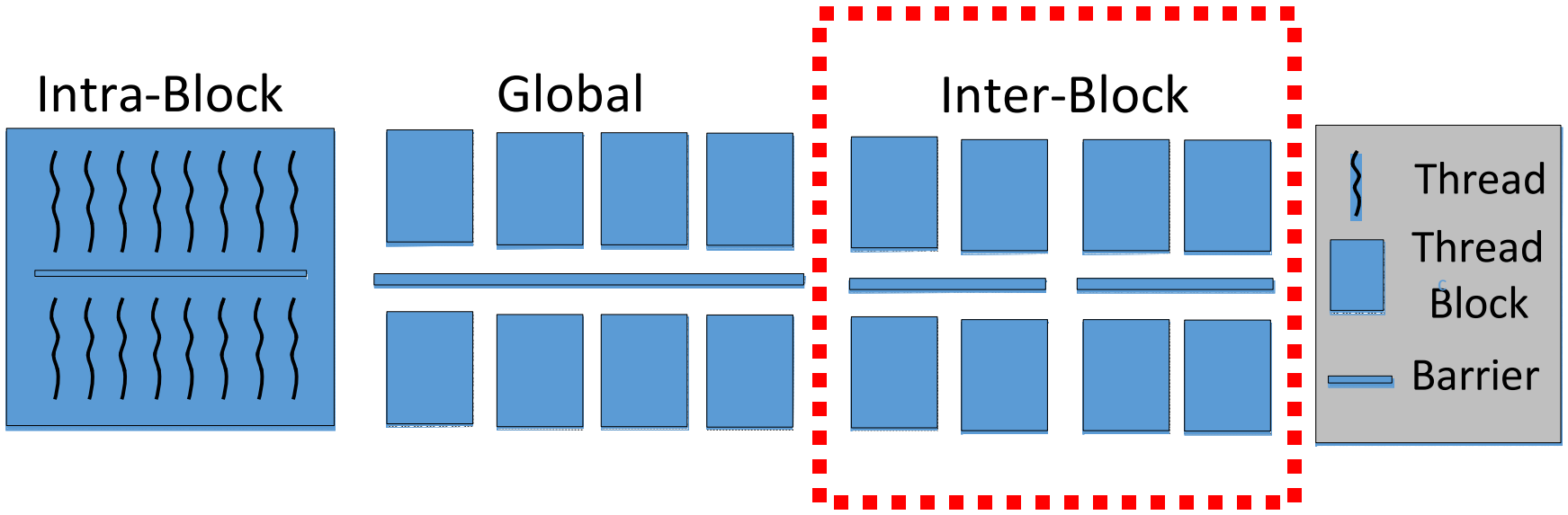
NO multi-parent dependency support



Still NO general dependency support!

Motivation

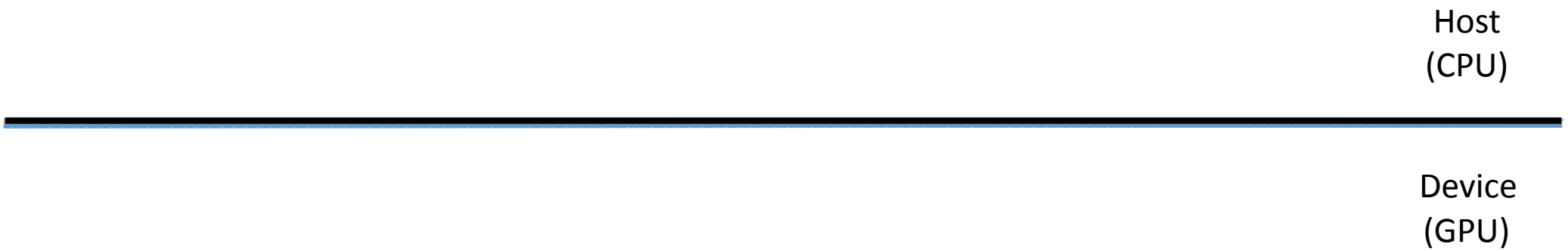
- There is a need for a generalized support for finer-grain inter-block data dependency for more performance and efficiency.



Motivation

- Current limitations
  - High device-side kernel launch overhead
  - No general inter-block data dependency support

Wireframe Overview



Wireframe Overview

Programming Model

```
#define parent1 dim3 (blockIdx.x-1,
blockIdx.y, blockIdx.z);
#define parent2 dim3 (blockIdx.x, blockIdx.y-
1, blockIdx.z);
void* DepLink() {
    if (blockIdx.x > 0)
WF::AddDependency(parent1);
    if (blockIdx.y > 0)
WF::AddDependency(parent2);
}
int main() {
    kernel<<<GridSize, BlockSize,
DepLink>>>(0, args);
}
__WF__ void kernel(args) {
    processWave();
}
```

Host
(CPU)

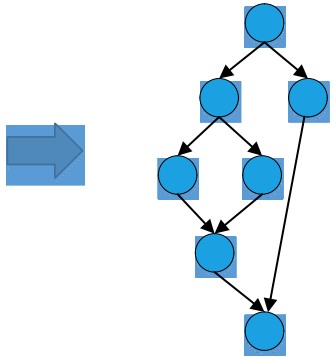
Device
(GPU)

Wireframe Overview

Programming Model

```
#define parent1 dim3 (blockIdx.x-1,
blockIdx.y, blockIdx.z);
#define parent2 dim3 (blockIdx.x, blockIdx.y-
1, blockIdx.z);
void* DepLink() {
    if (blockIdx.x > 0)
WF::AddDependency(parent1);
    if (blockIdx.y > 0)
WF::AddDependency(parent2);
}
int main() {
    kernel<<<GridSize, BlockSize,
DepLink>>>(0, args);
}
__WF__ void kernel(args) {
    processWave();
}
```

Dependency Graph



Host (CPU)

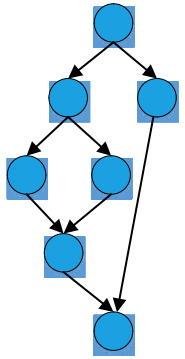
Device (GPU)

Wireframe Overview

Programming Model

```
#define parent1 dim3 (blockIdx.x-1,
blockIdx.y, blockIdx.z);
#define parent2 dim3 (blockIdx.x, blockIdx.y-
1, blockIdx.z);
void* DepLink() {
    if (blockIdx.x > 0)
WF::AddDependency(parent1);
    if (blockIdx.y > 0)
WF::AddDependency(parent2);
}
int main() {
    kernel<<<GridSize, BlockSize,
DepLink>>>(0, args);
}
__WF__ void kernel(args) {
    processWave();
}
```

Dependency Graph



Convert to CSR

Node Array

Edge Array

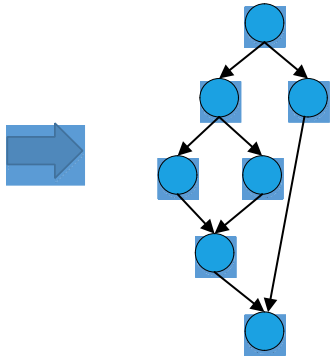
Host (CPU)

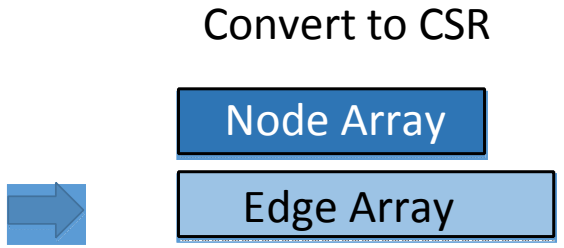
Device (GPU)

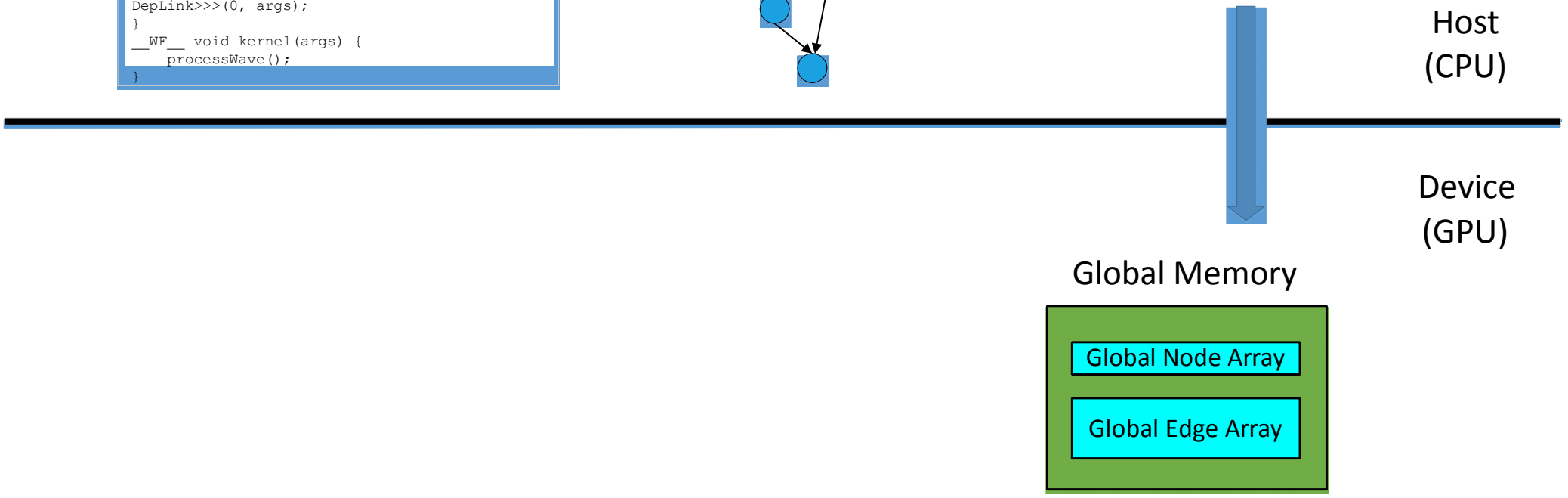
Wireframe Overview

Programming Model

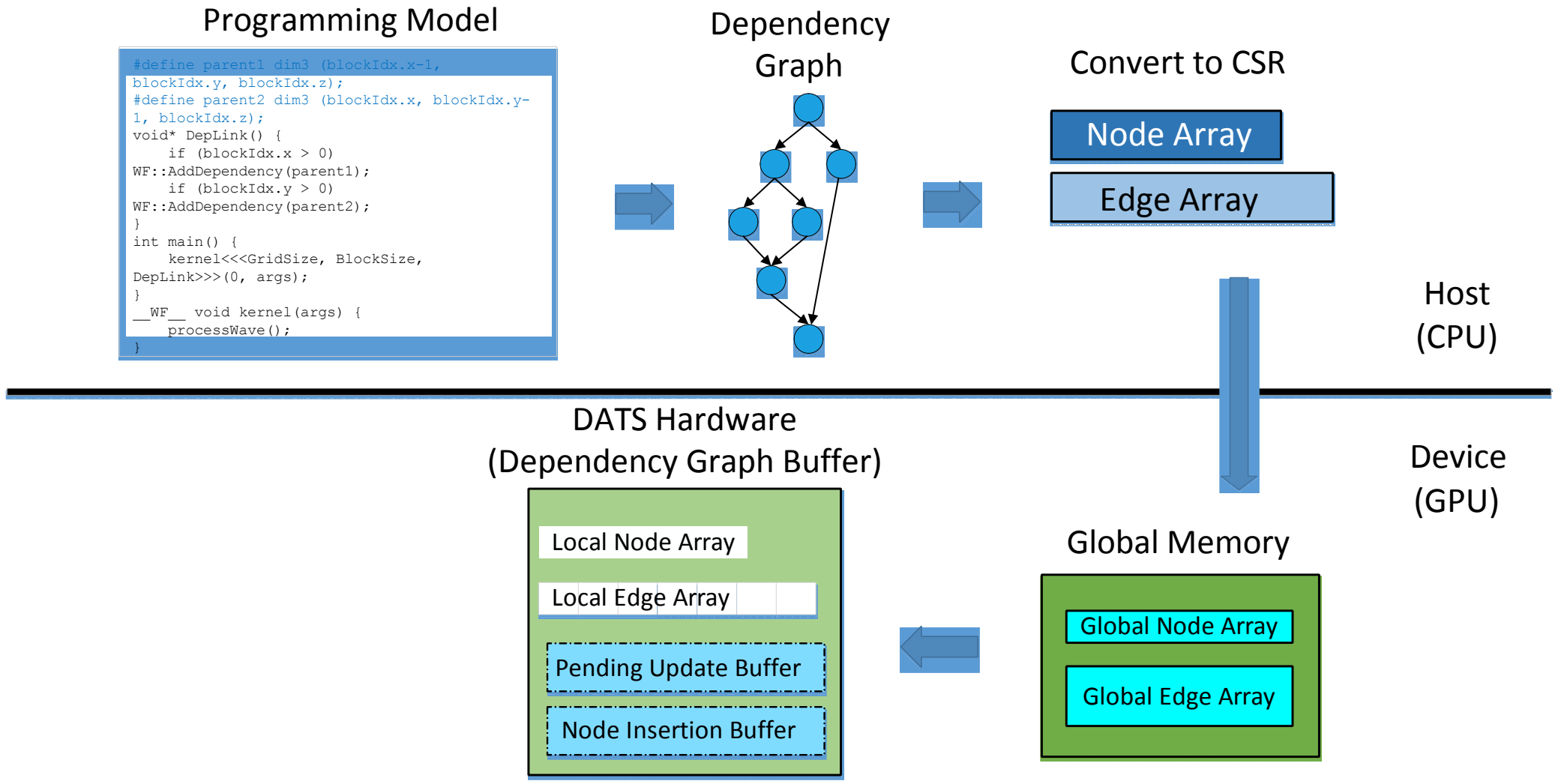
```
#define parent1 dim3 (blockIdx.x-1,
blockIdx.y, blockIdx.z);
#define parent2 dim3 (blockIdx.x, blockIdx.y-
1, blockIdx.z);
void* DepLink() {
    if (blockIdx.x > 0)
WF::AddDependency(parent1);
    if (blockIdx.y > 0)
WF::AddDependency(parent2);
}
int main() {
    kernel<<<GridSize, BlockSize,
DepLink>>>(0, args);
}
__WF__ void kernel(args) {
    processWave();
}
```



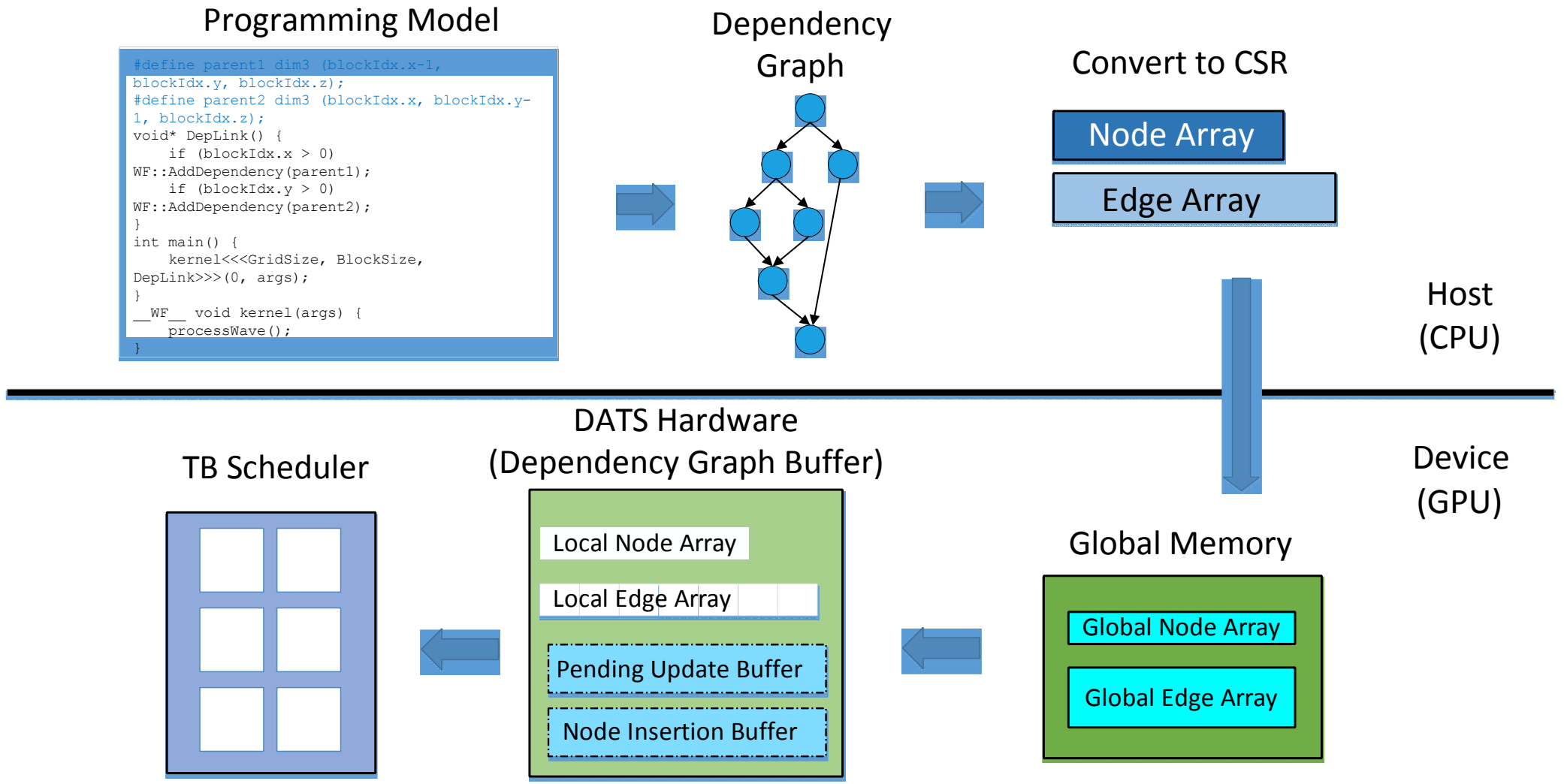




Wireframe Overview



Wireframe Overview



Programming Model

- New functions are needed to support dependency in CUDA
  - Add dependency
  - Policy settings
- Proposing DepLinks model
  - Would assign a dependency graph generation function to a kernel
  - Easy to learn and use

Wireframe Pseudo-code

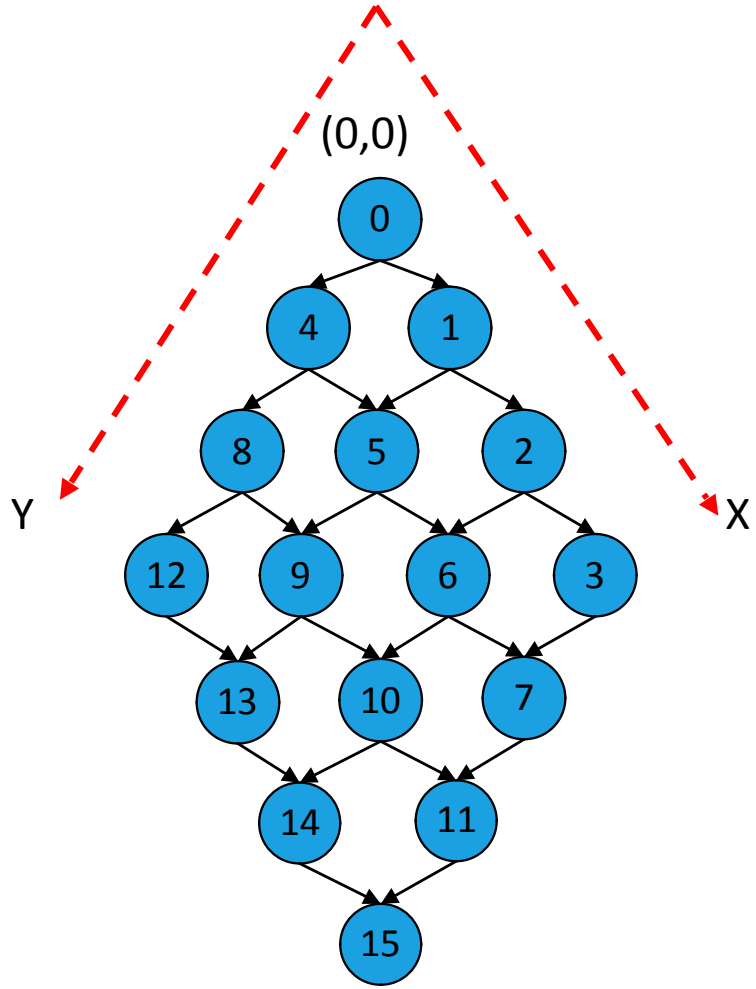
parent1 := (X-1, Y)
parent2 := (X, Y-1)

RUN:

-Kernel Launch (DepLinks)

DepLinks @BLOCK (X,Y):

-Add Dependency (parent1)
-Add Dependency (parent2)



Wireframe Pseudo-code

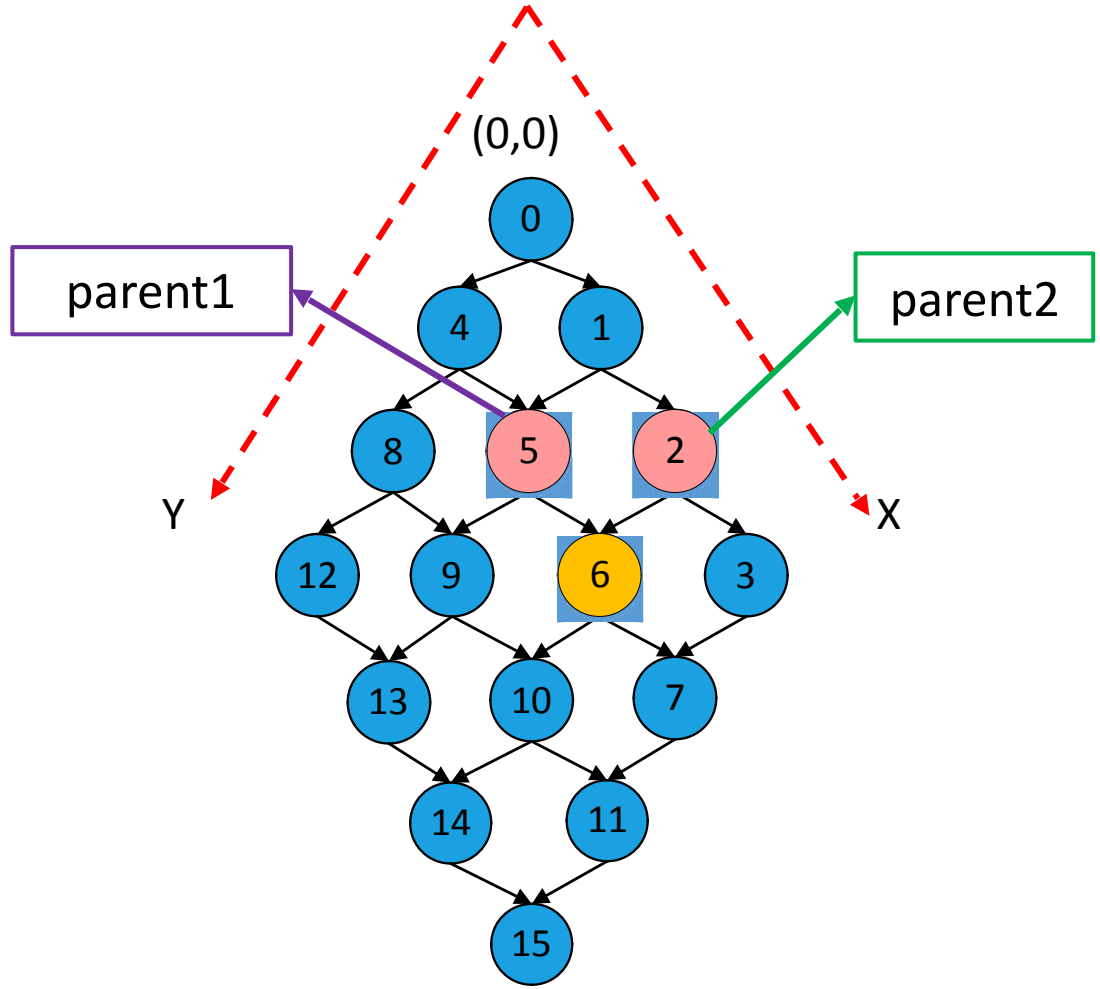
parent1 := (X-1, Y)
parent2 := (X, Y-1)

RUN:

-Kernel Launch (DepLinks)

DepLinks @BLOCK (X,Y):

-Add Dependency (parent1)
-Add Dependency (parent2)



Wireframe Pseudo-code

parent1 := (X-1, Y)
parent2 := (X, Y-1)

RUN:

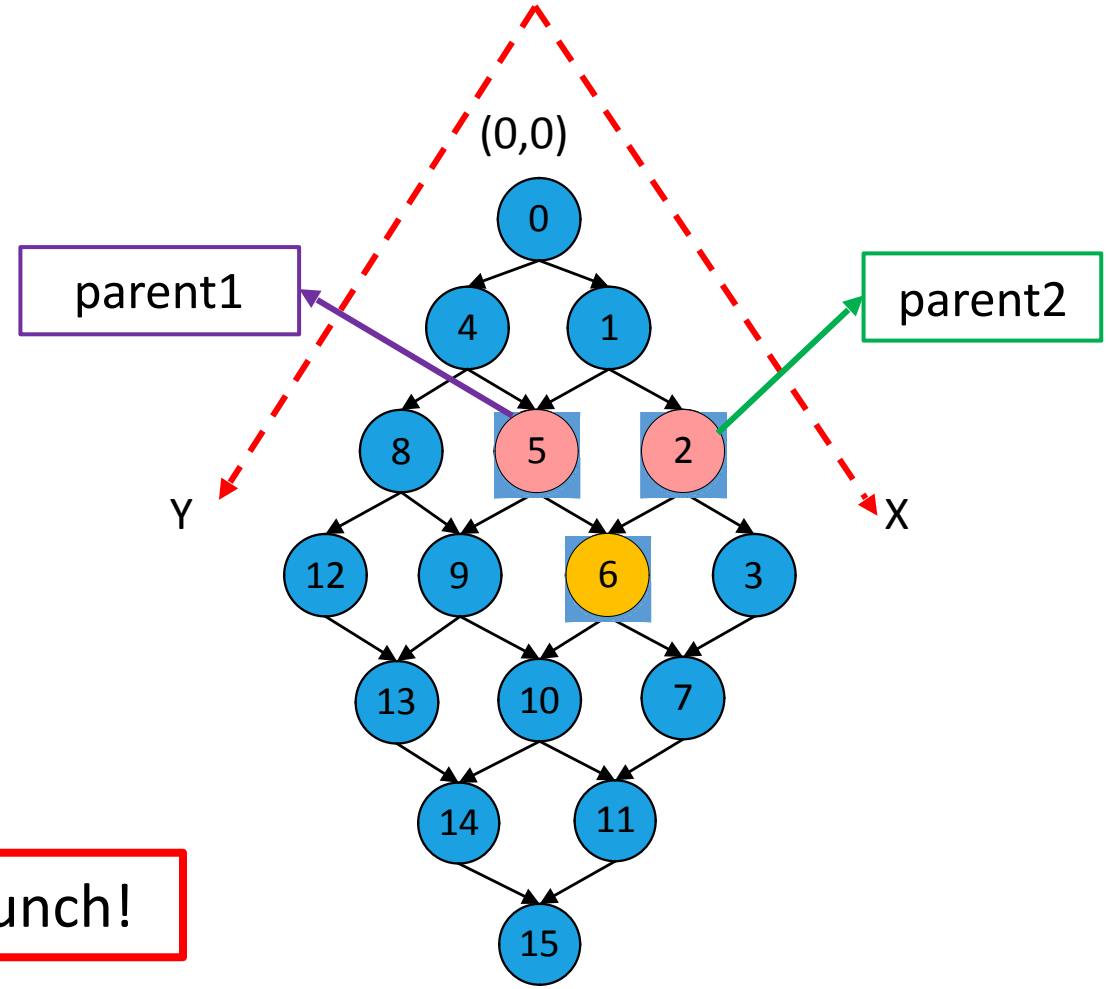
-Kernel Launch (DepLinks)

DepLinks @BLOCK (X,Y):

-Add Dependency (parent1)

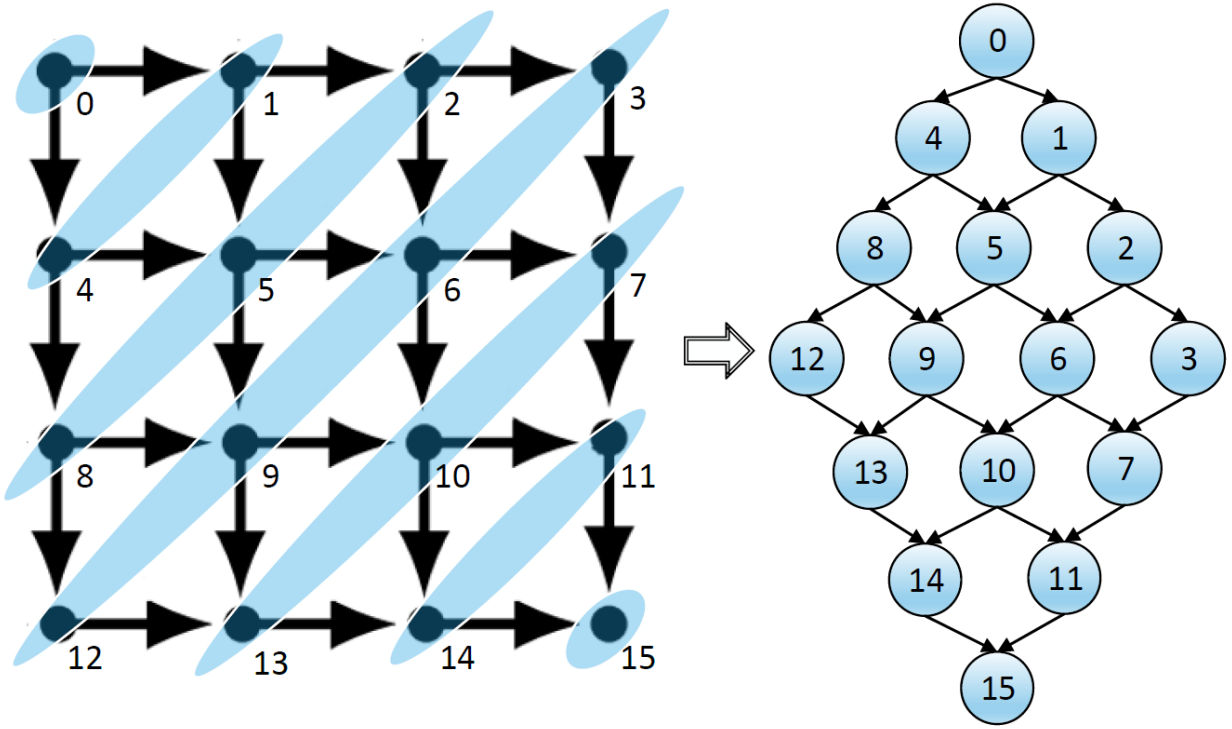
-Add Dependency (parent2)

One kernel launch!



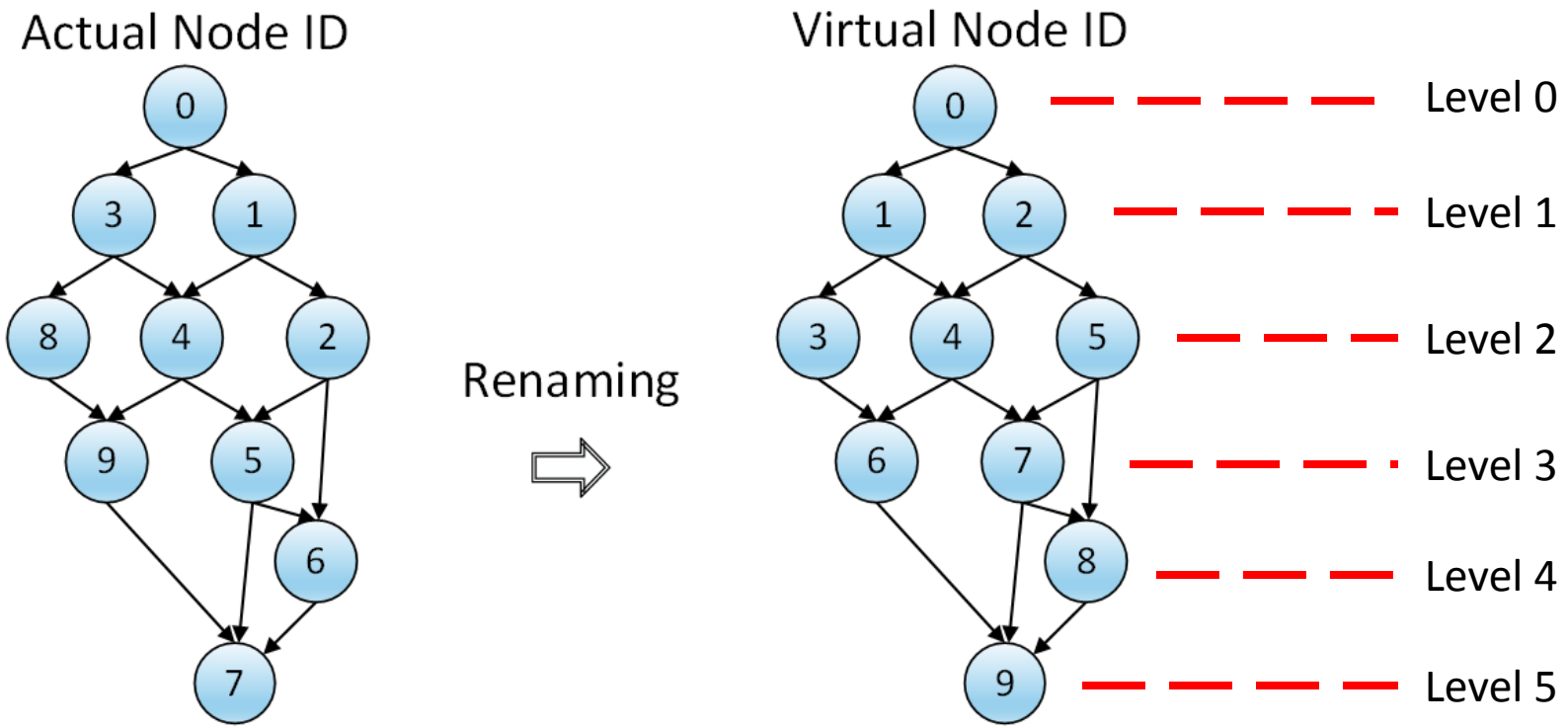
Dependency Graph

- Parent count and level of every node determined at runtime
- Sent to the GPU's global memory



Node Renaming

- To minimize data level range in the buffers



Dependency-Aware TB Scheduler (DATS)

- Thread block scheduler
  - Issues the relevant thread block at the time for execution based on the dependency graph

- Dependency Graph Buffer (DGB)
  - Cache data from global memory
  - Challenge: Efficient caching and data utilization

Dependency-Aware TB Scheduler (DATS)

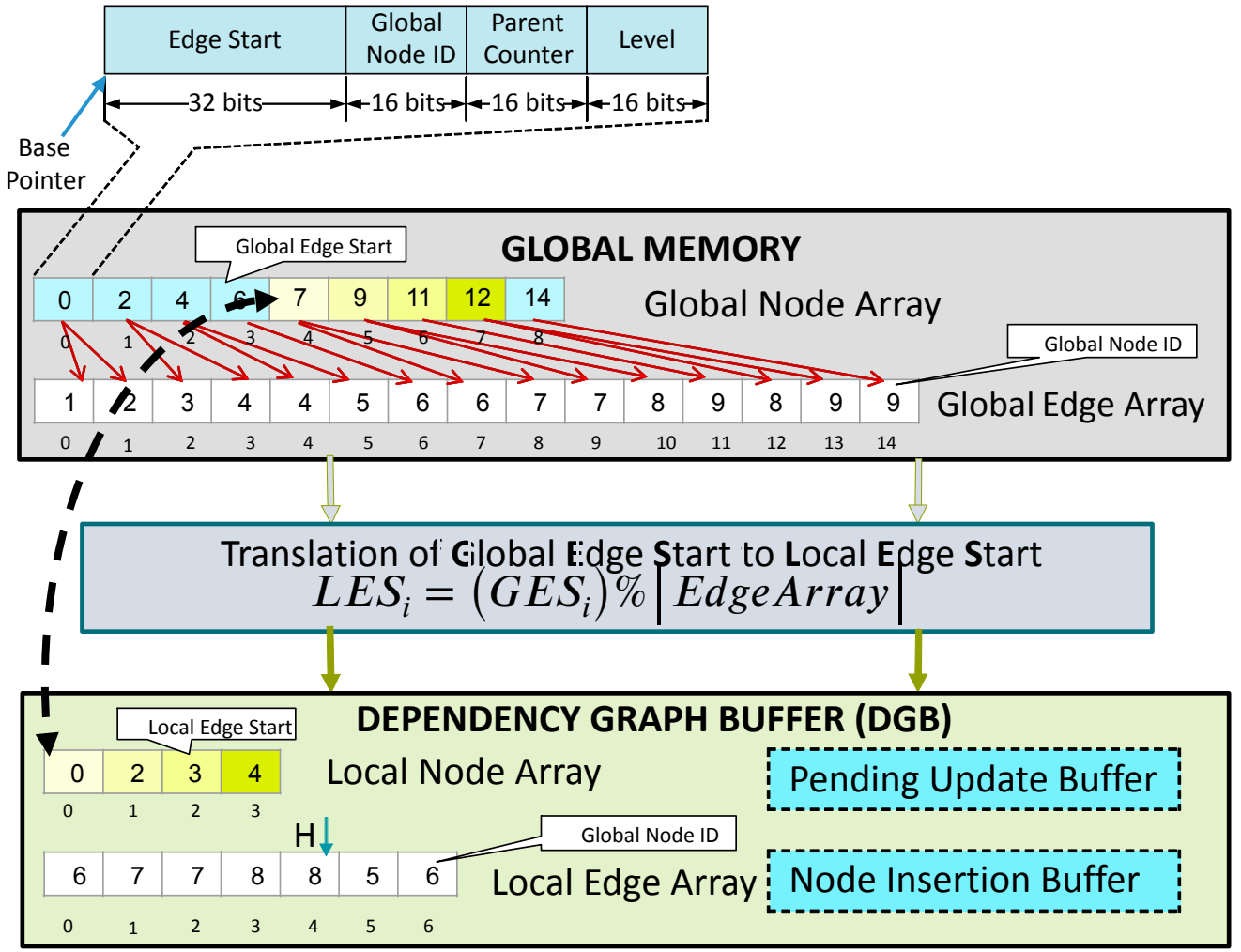
Data stored in compressed sparse (CSR) format

- **To reduce memory usage**

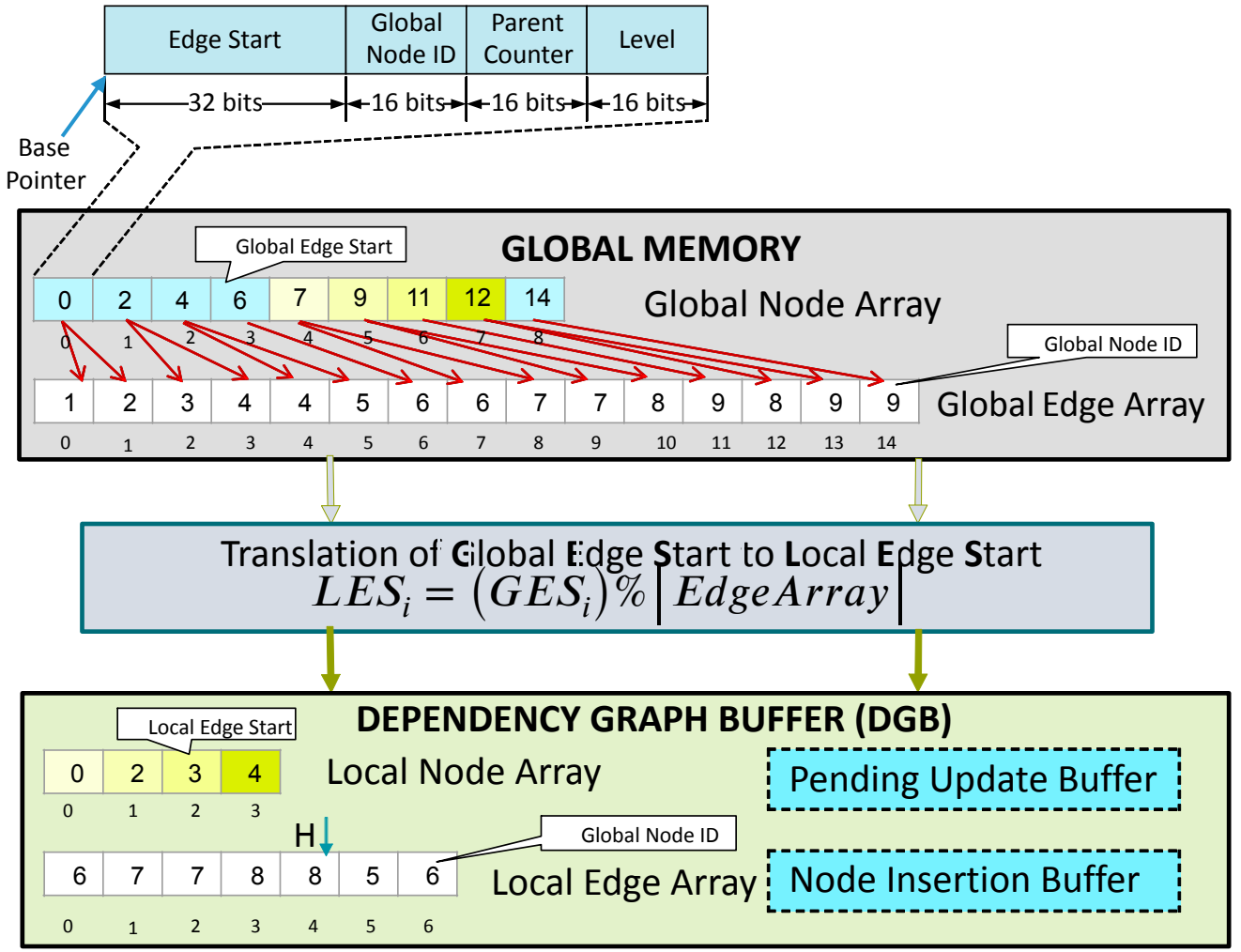
- **Thread blocks → Node Array**
- **Dependencies → Edge Array**

- space complexity

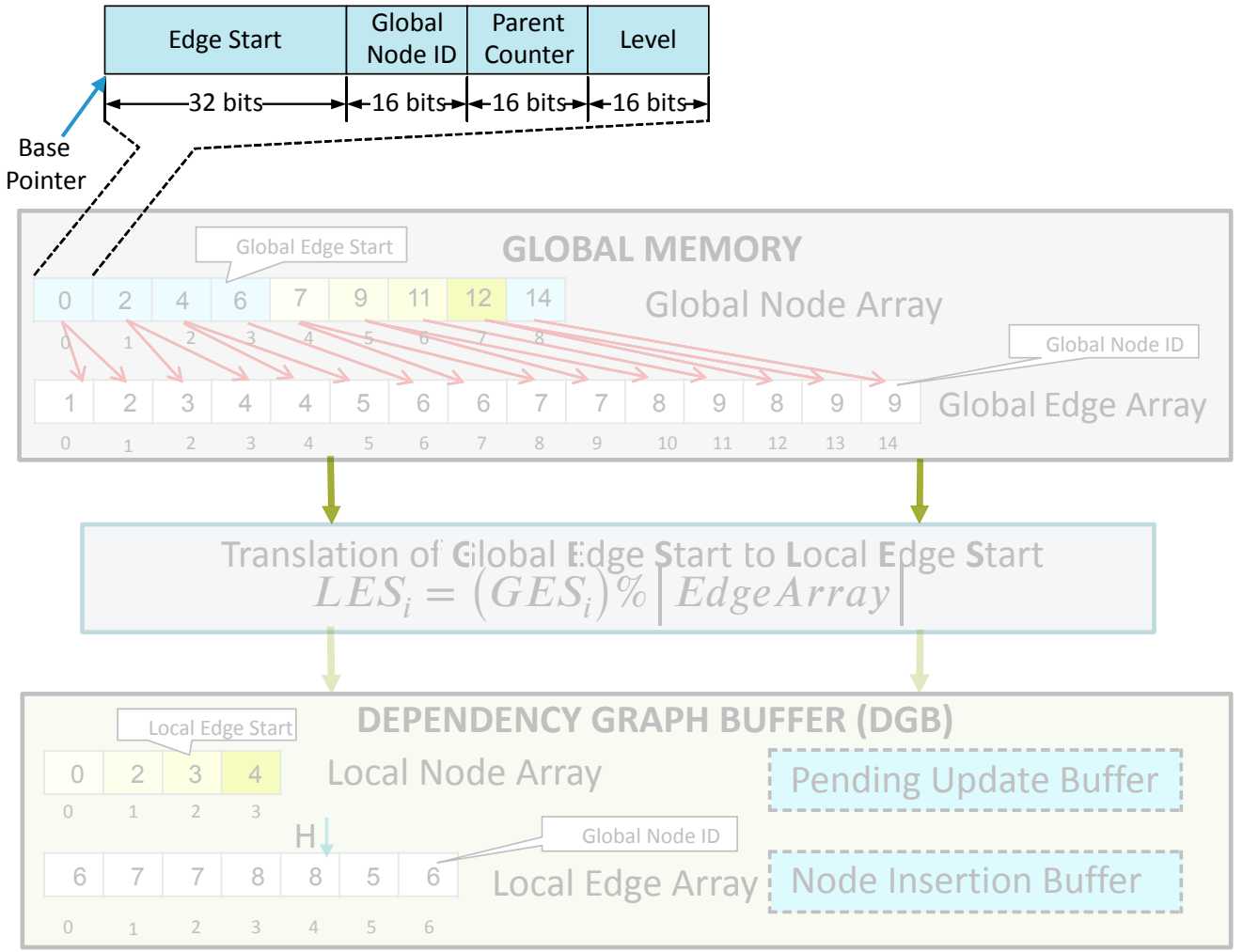
DATS Overview



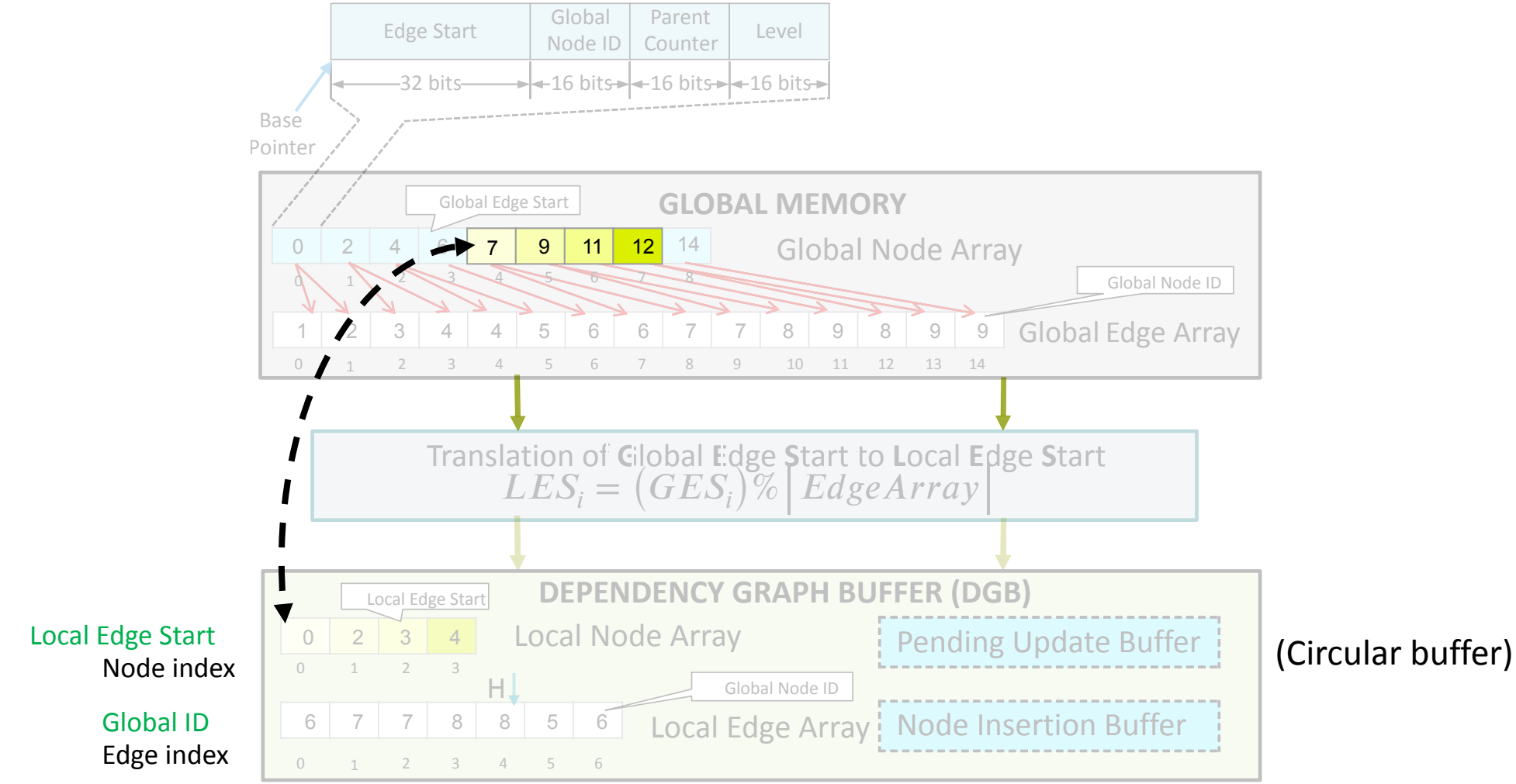
DATS Overview



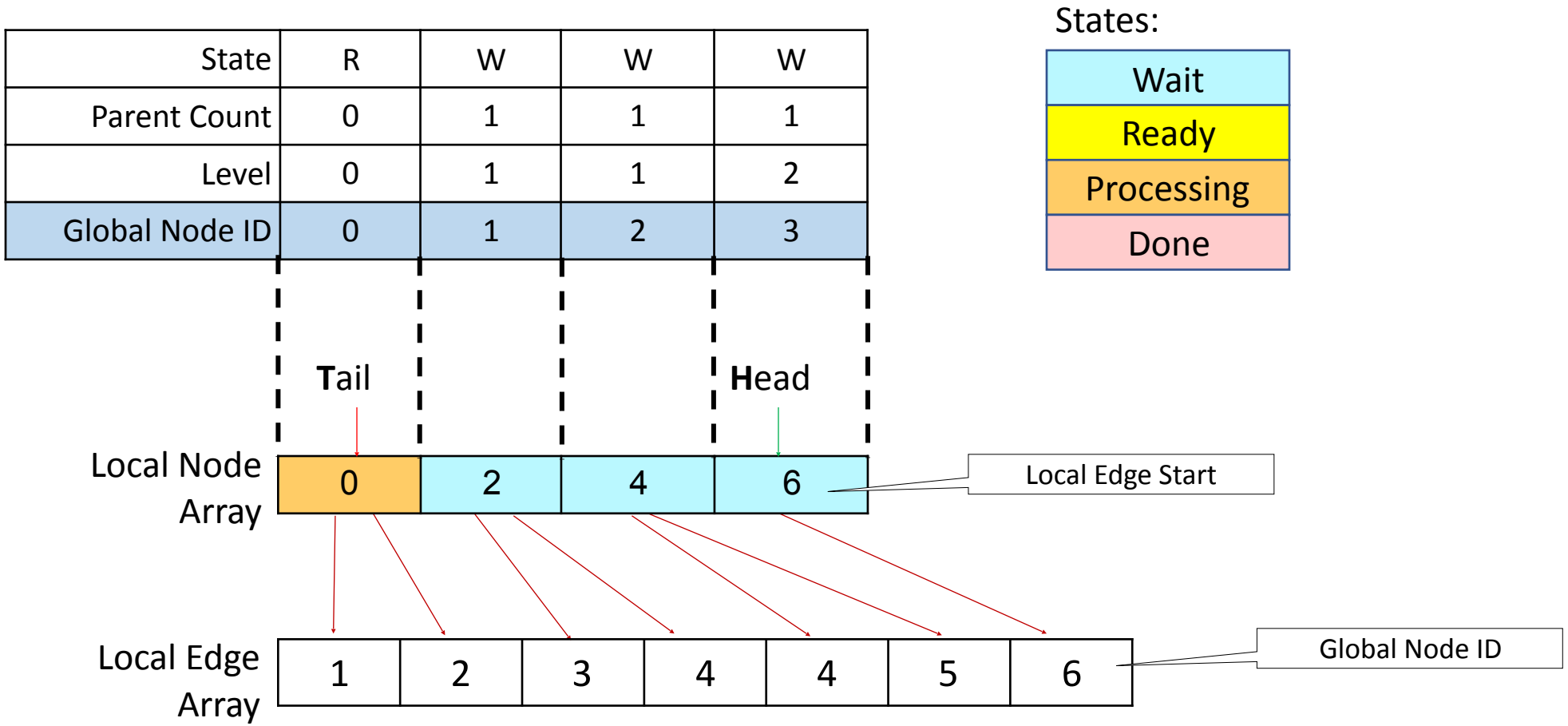
DATS Overview



DATS Overview



Node State Table

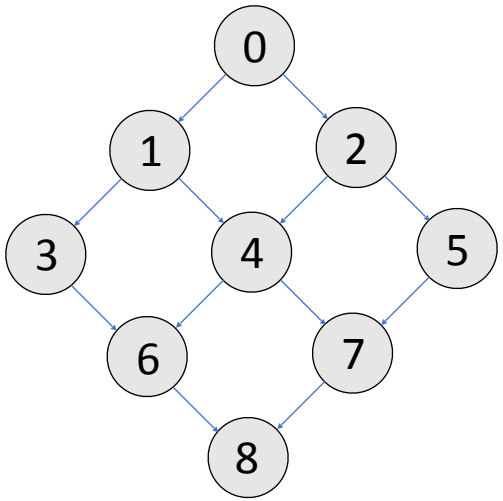


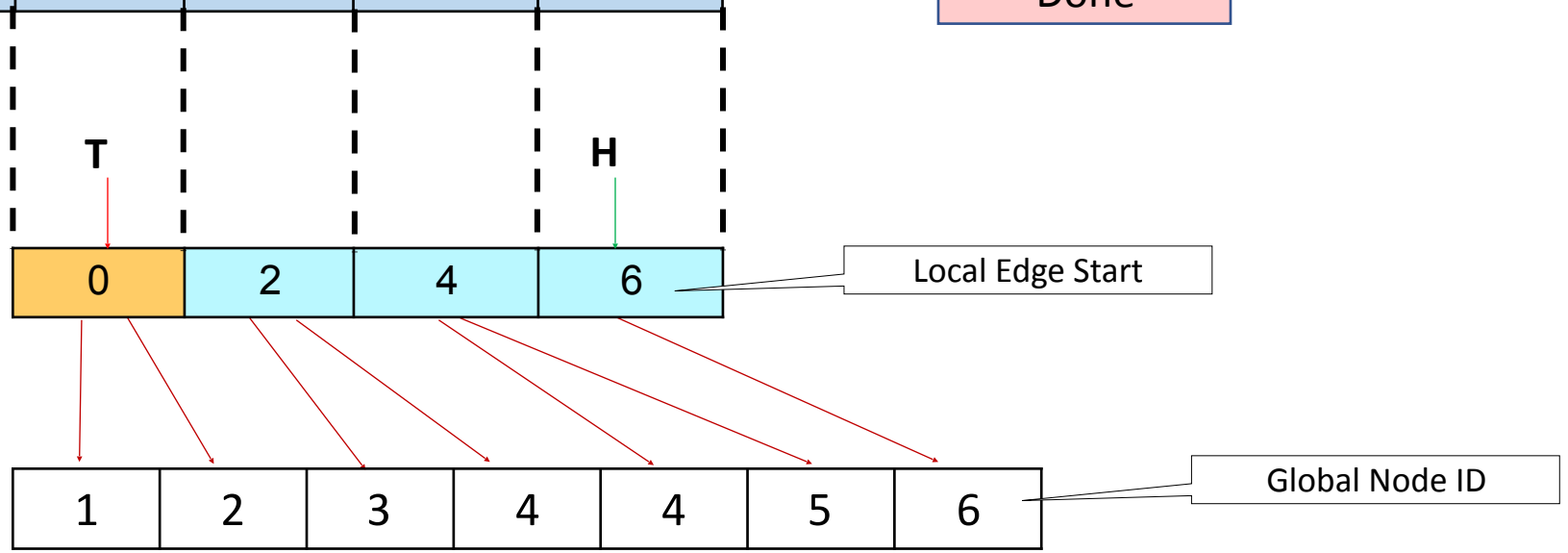
Node State Table

State	R	W	W	W
Parent Count	0	1	1	1
Level	0	1	1	2
Global Node ID	0	1	2	3

States:

- Wait
- Ready
- Processing
- Done



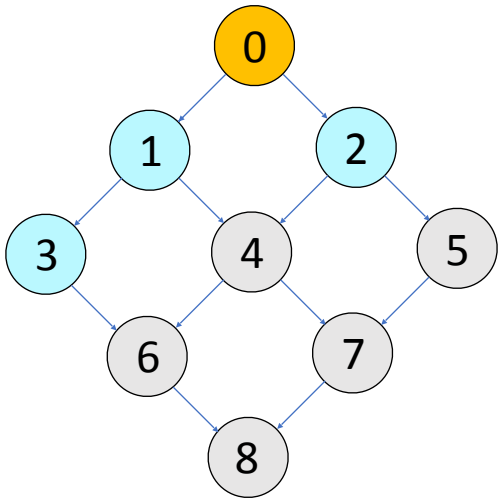


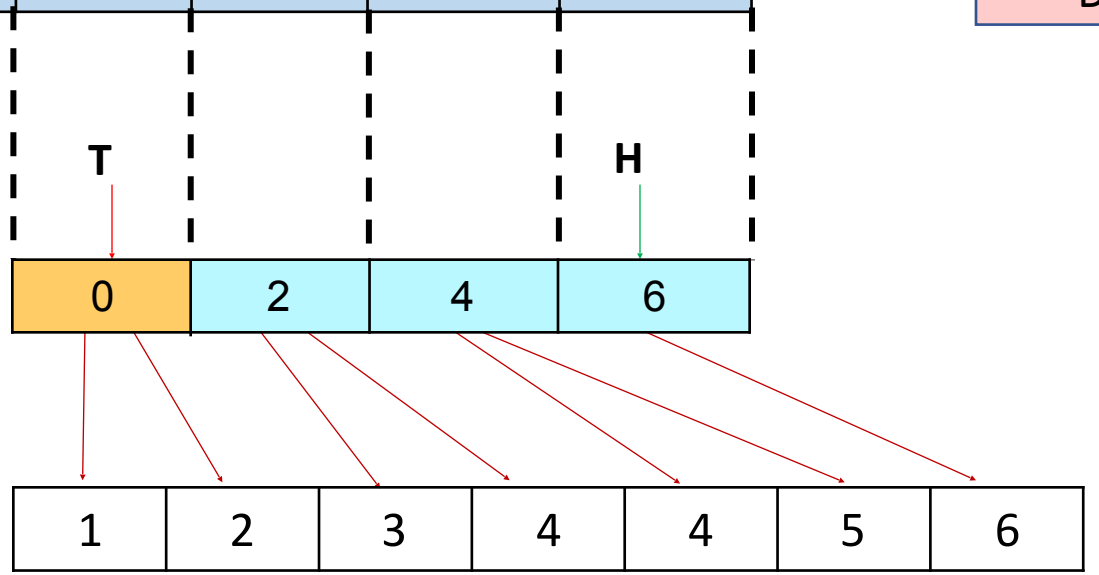
Example: Child Node Execution

State	R	W	W	W
Parent Count	0	1	1	1
Level	0	1	1	2
Global Node ID	0	1	2	3

States:

- Wait
- Ready
- Processing
- Done



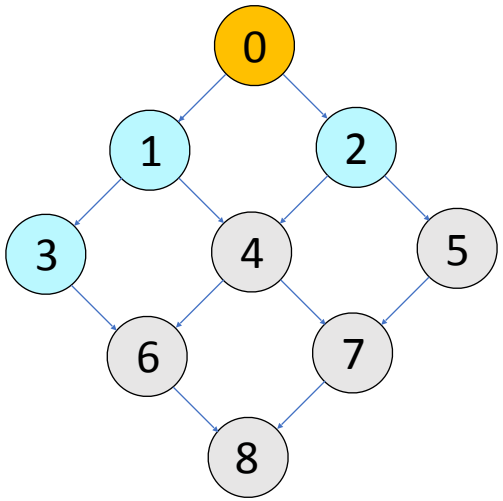


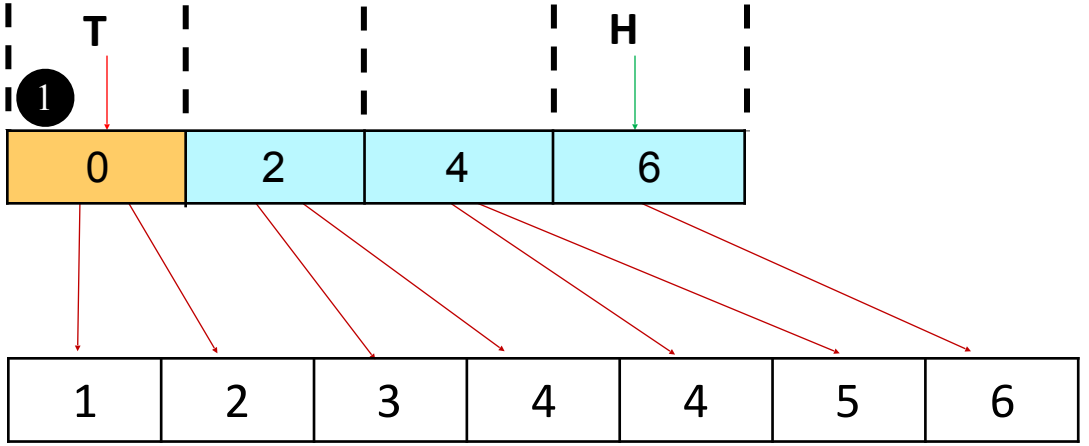
Example: Child Node Execution

State	1 D	W	W	W
Parent Count	0	1	1	1
Level	0	1	1	2
Global Node ID	0	1	2	3

States:

- Wait
- Ready
- Processing
- Done



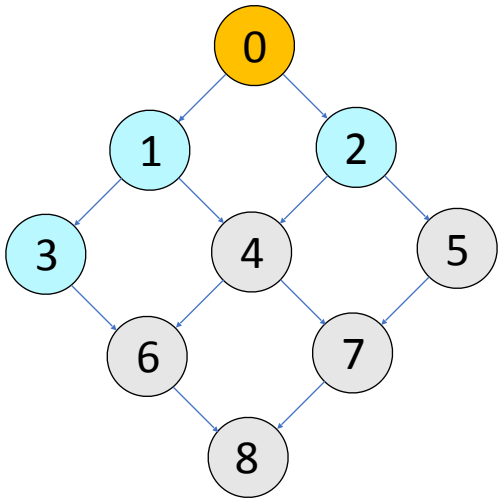


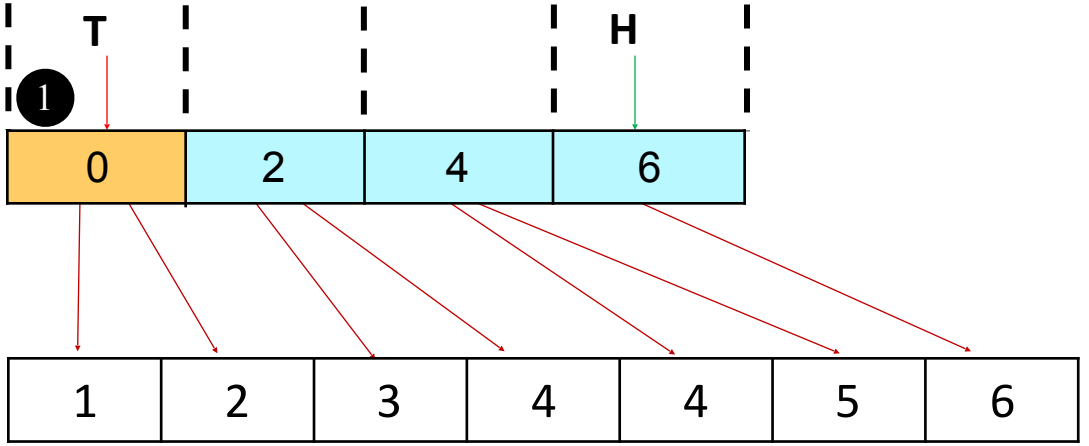
Example: Child Node Execution

State	1 D	W	W	W
Parent Count	0	2 0	2 0	1
Level	0	1	1	2
Global Node ID	0	1	2	3

States:

- Wait
- Ready
- Processing
- Done



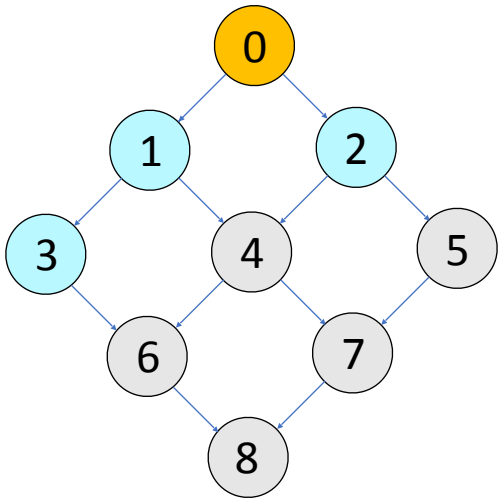


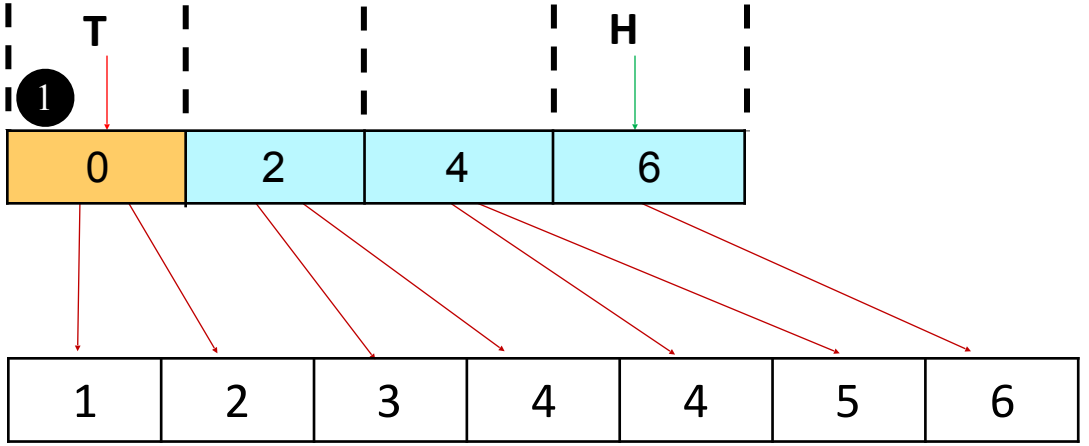
Example: Child Node Execution

State	1 D	3 R	3 R	W
Parent Count	0	2 0	2 0	1
Level	0	1	1	2
Global Node ID	0	1	2	3

States:

- Wait
- Ready
- Processing
- Done



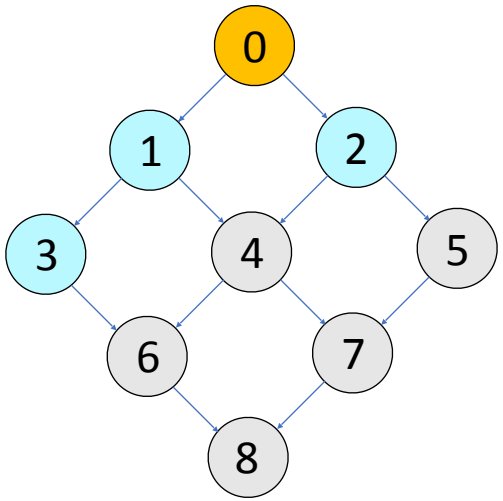


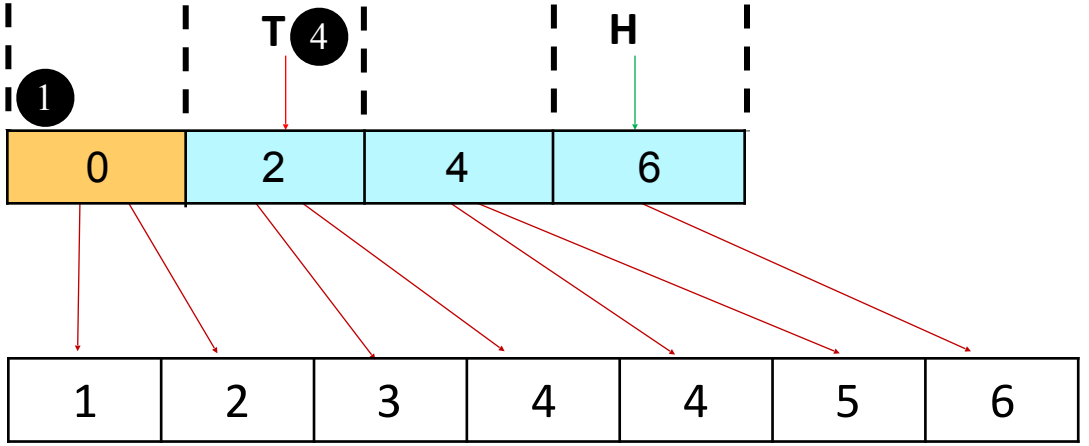
Example: Child Node Execution

State	1 D	3 R	3 R	W
Parent Count	0	2 0	2 0	1
Level	0	1	1	2
Global Node ID	0	1	2	3

States:

- Wait
- Ready
- Processing
- Done



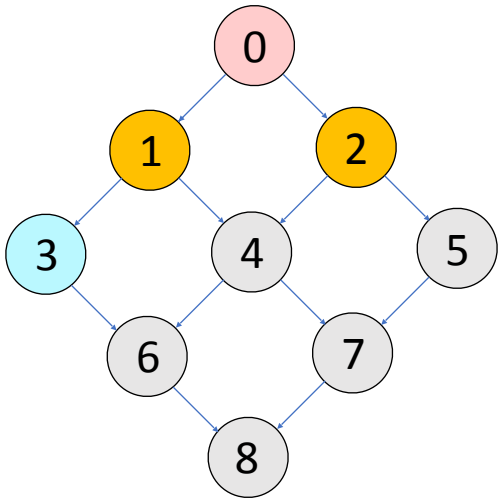


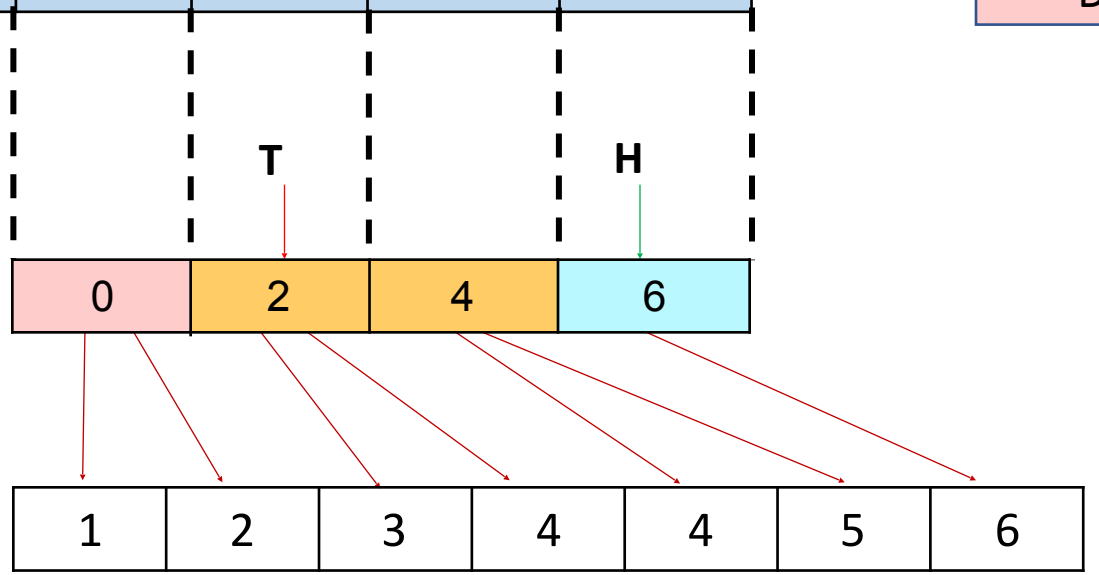
Example: Update Buffer Store

State	D	P	P	W
Parent Count	0	0	0	1
Level	0	1	1	2
Global Node ID	0	1	2	3

States:

- Wait
- Ready
- Processing
- Done



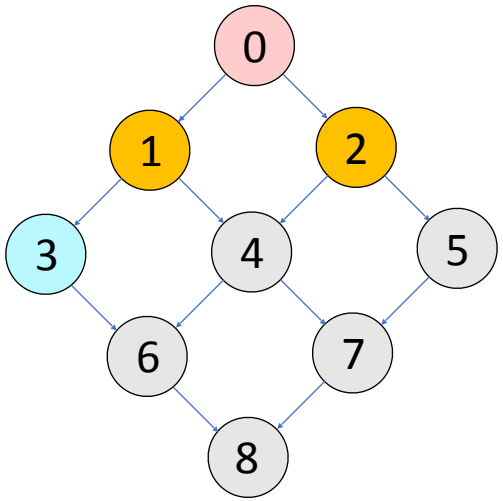


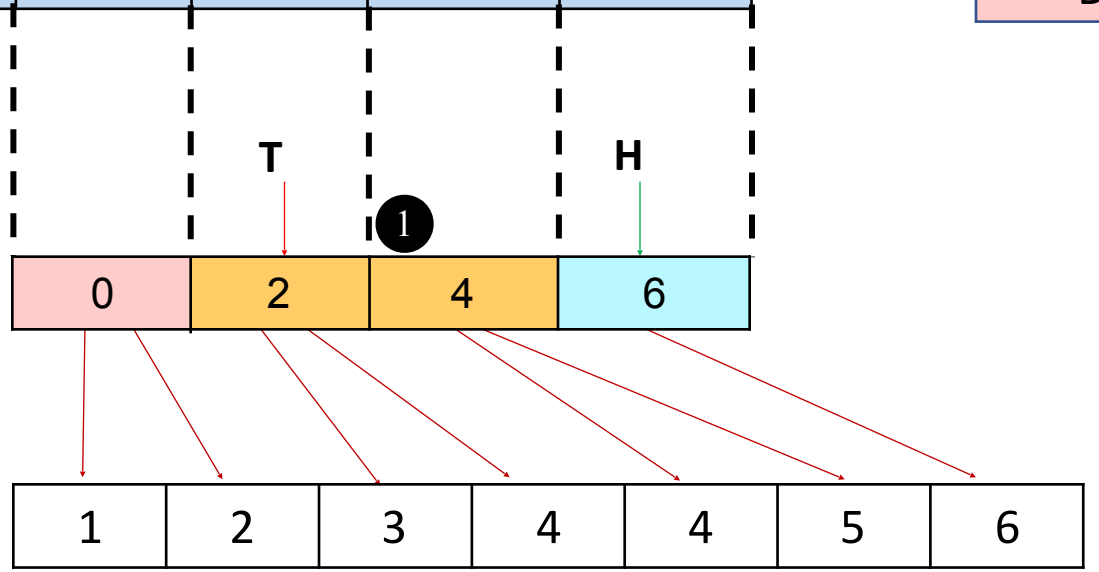
Example: Update Buffer Store

State	D	P	1 D	W
Parent Count	0	0	0	1
Level	0	1	1	2
Global Node ID	0	1	2	3

States:

- Wait
- Ready
- Processing
- Done



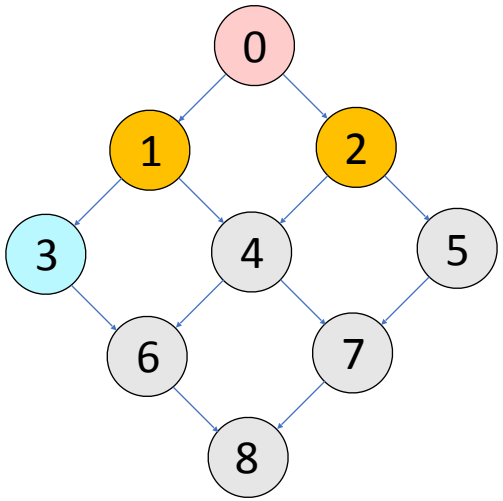


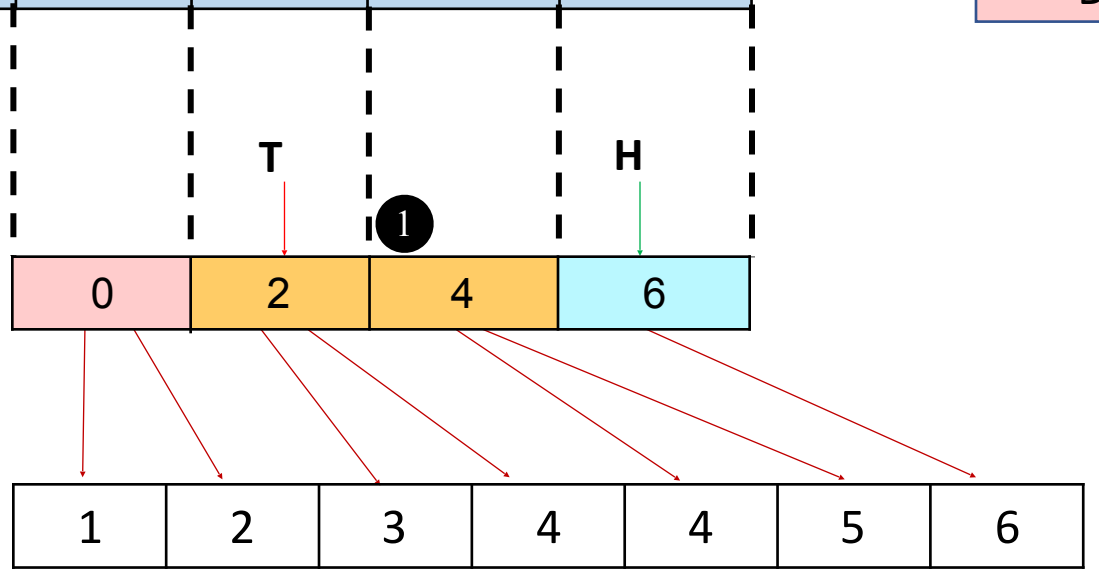
Example: Update Buffer Store

State	D	P	1 D	W
Parent Count	0	0	0	1
Level	0	1	1	2
Global Node ID	0	1	2	3

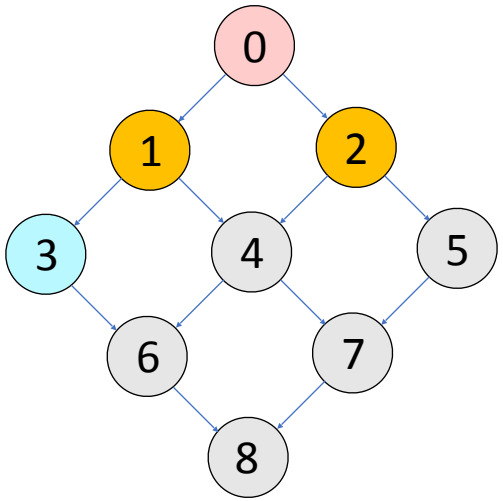
States:

- Wait
- Ready
- Processing
- Done





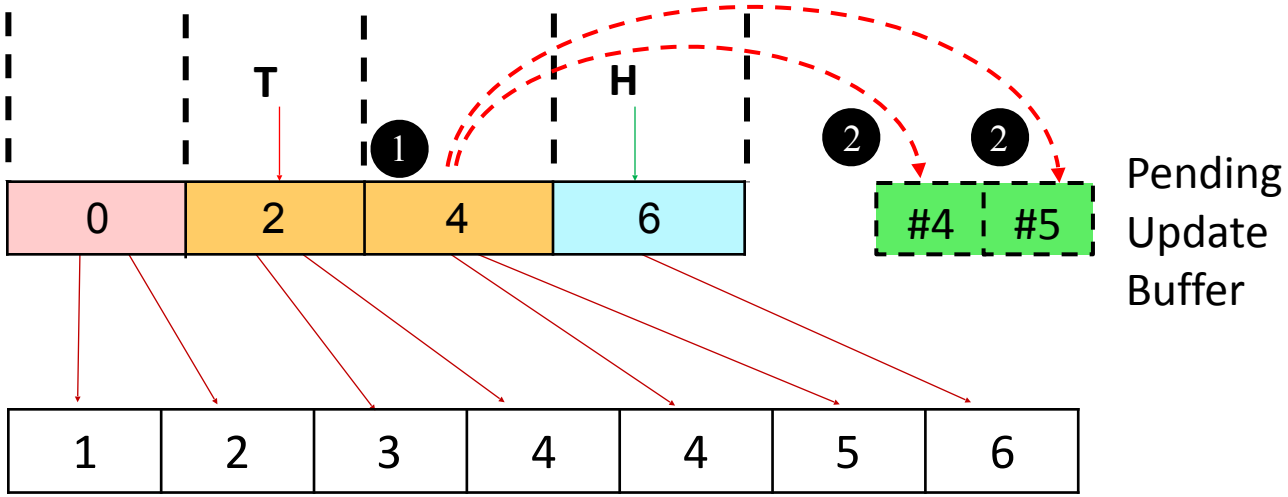
Example: Update Buffer Store



State	D	P	1 D	W
Parent Count	0	0	0	1
Level	0	1	1	2
Global Node ID	0	1	2	3

States:

- Wait
- Ready
- Processing
- Done



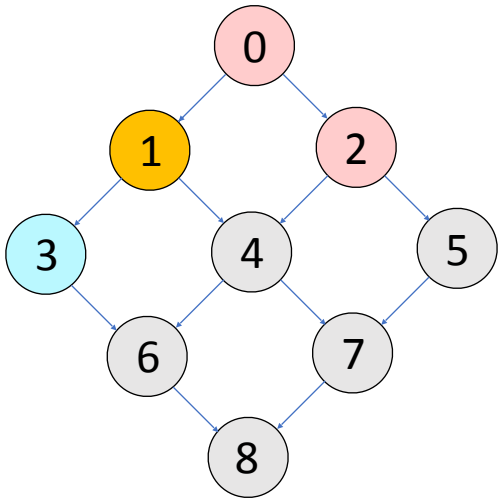
Pending Update Buffer

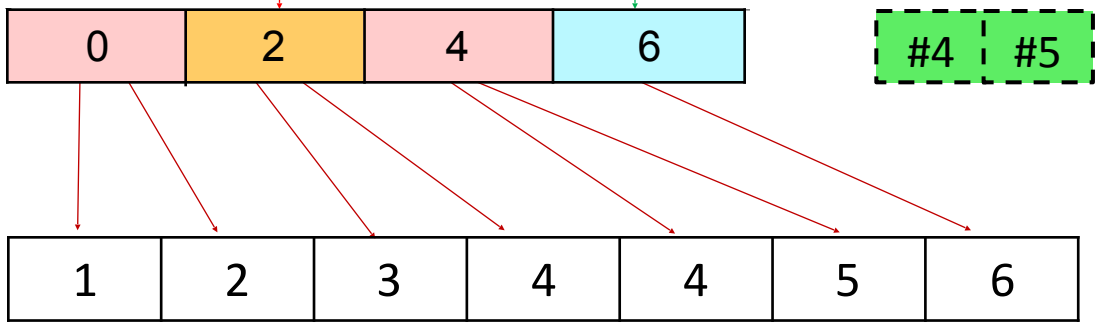
Example: Invalidation…

State	D	P	D	W
Parent Count	0	0	0	1
Level	0	1	1	2
Global Node ID	0	1	2	3

States:

- Wait
- Ready
- Processing
- Done





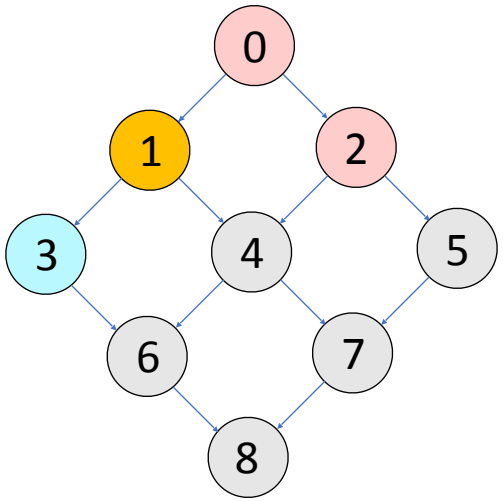
#4 #5

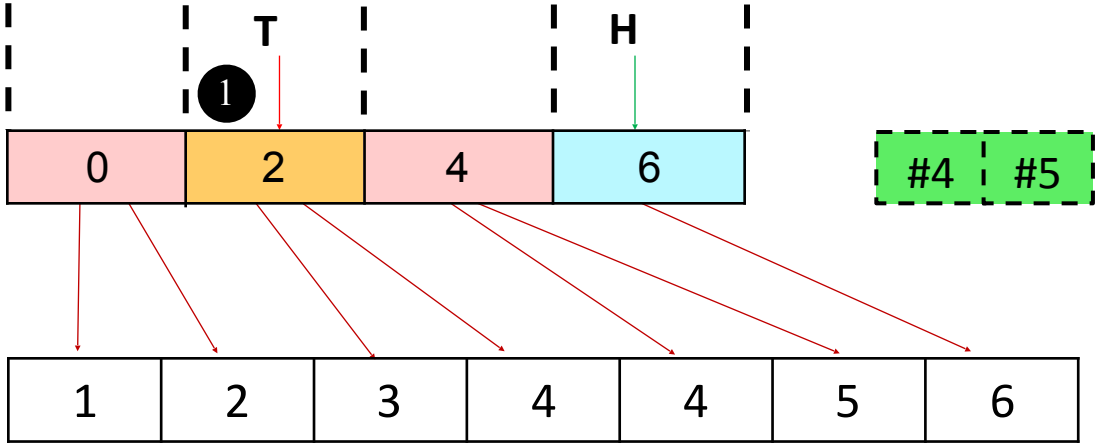
Example: Invalidation…

State	D	D	D	W
Parent Count	0	0	0	1
Level	0	1	1	2
Global Node ID	0	1	2	3

States:

- Wait
- Ready
- Processing
- Done





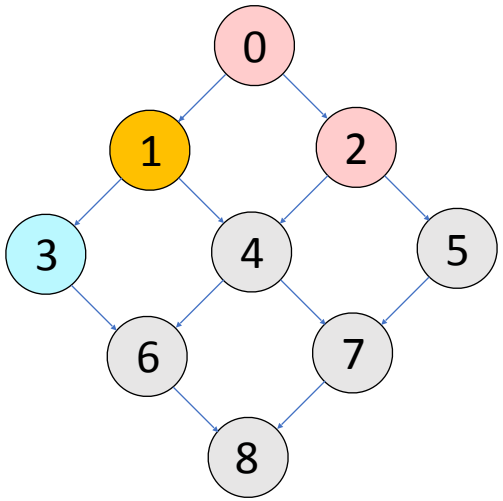
#4 #5

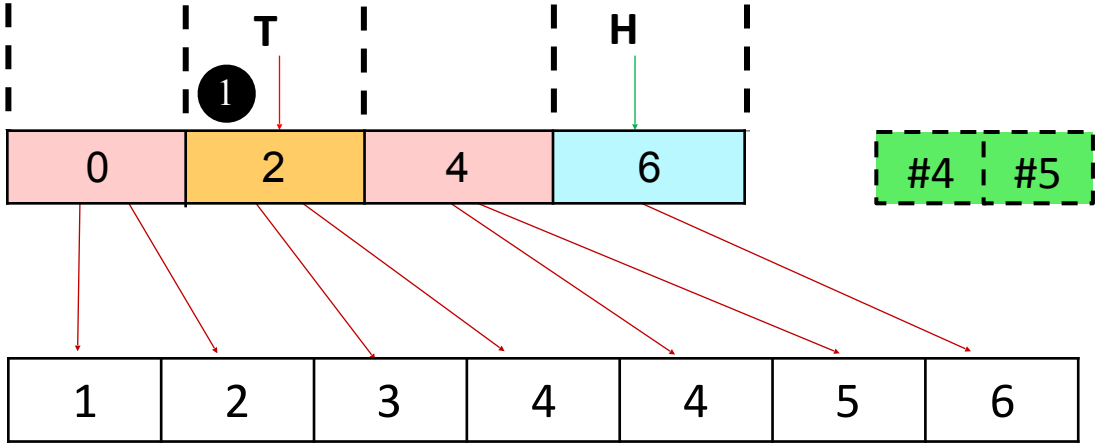
Example: Invalidation…

State	D	D	D	3 R
Parent Count	0	0	0	2 0
Level	0	1	1	2
Global Node ID	0	1	2	3

States:

- Wait
- Ready
- Processing
- Done





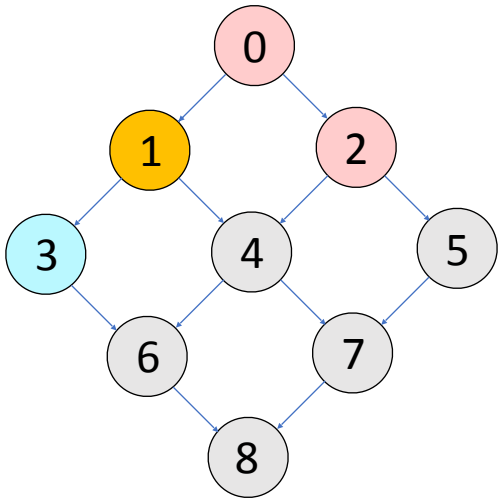
#4 #5

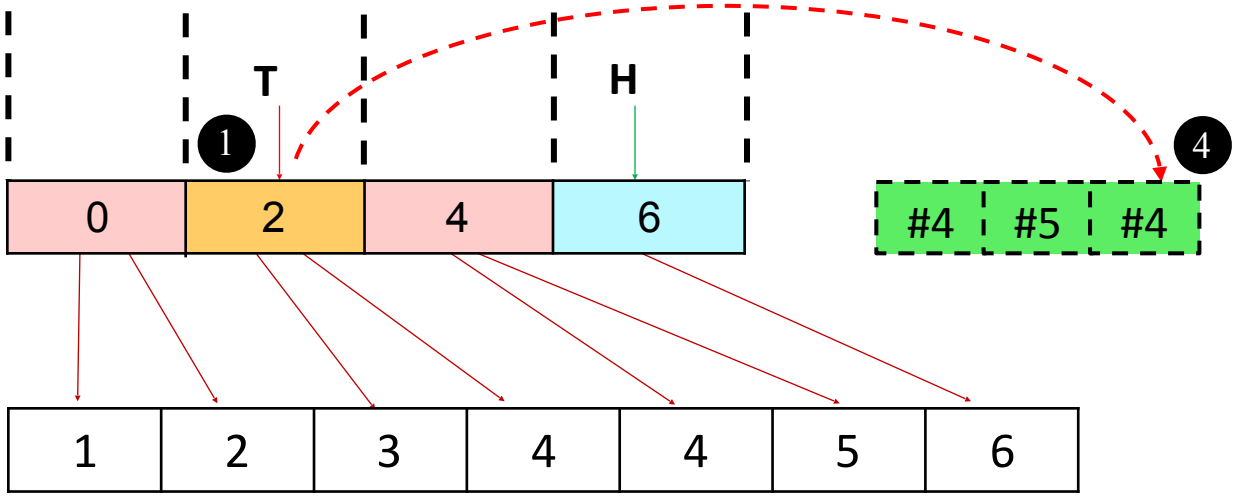
Example: Invalidation…

State	D	D	D	3 R
Parent Count	0	0	0	2 0
Level	0	1	1	2
Global Node ID	0	1	2	3

States:

- Wait
- Ready
- Processing
- Done



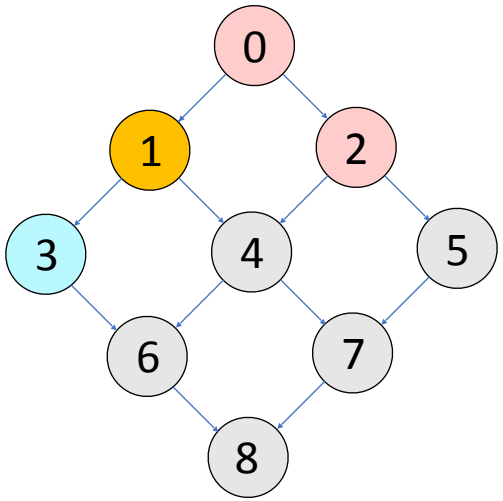


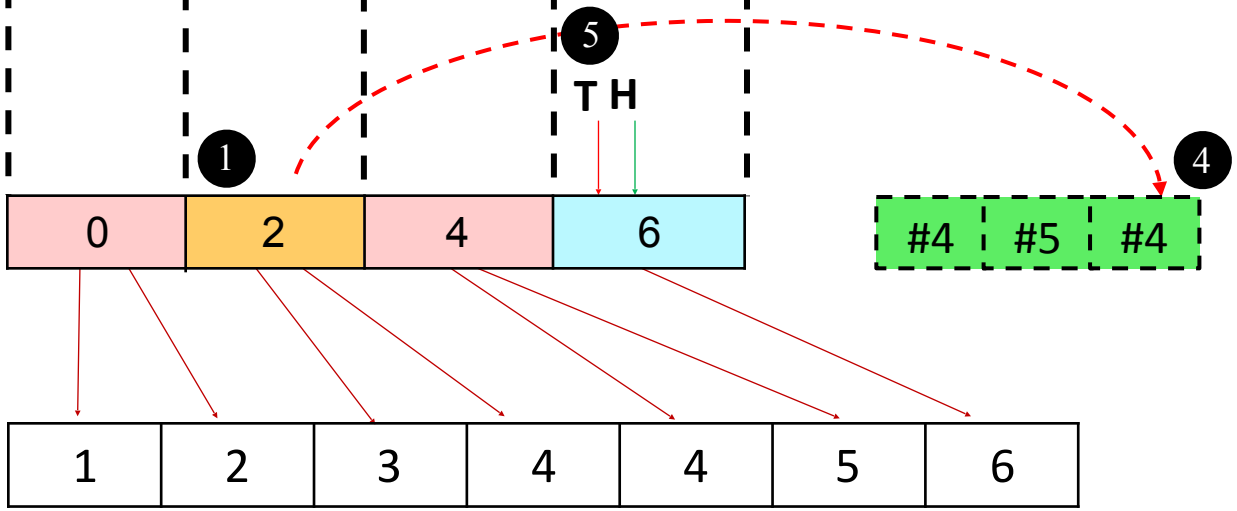
Example: Invalidation…

State	D	D	D	3 R
Parent Count	0	0	0	2 0
Level	0	1	1	2
Global Node ID	0	1	2	3

States:

- Wait
- Ready
- Processing
- Done





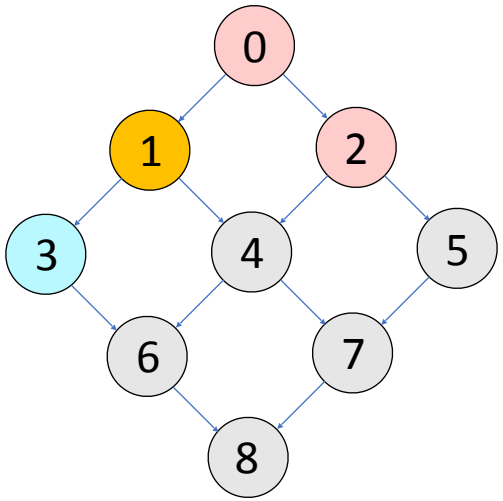
Example: Invalidation…

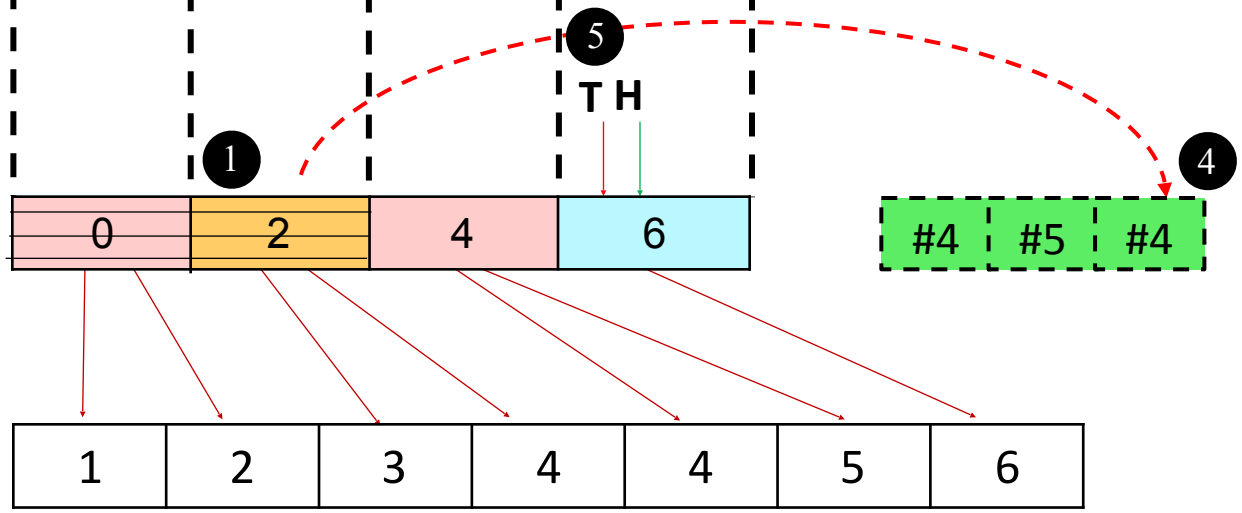
Enough spaces to load to DGB

State	D	D	D	3 R
Parent Count	0	0	0	2 0
Level	0	1	1	2
Global Node ID	0	1	2	3

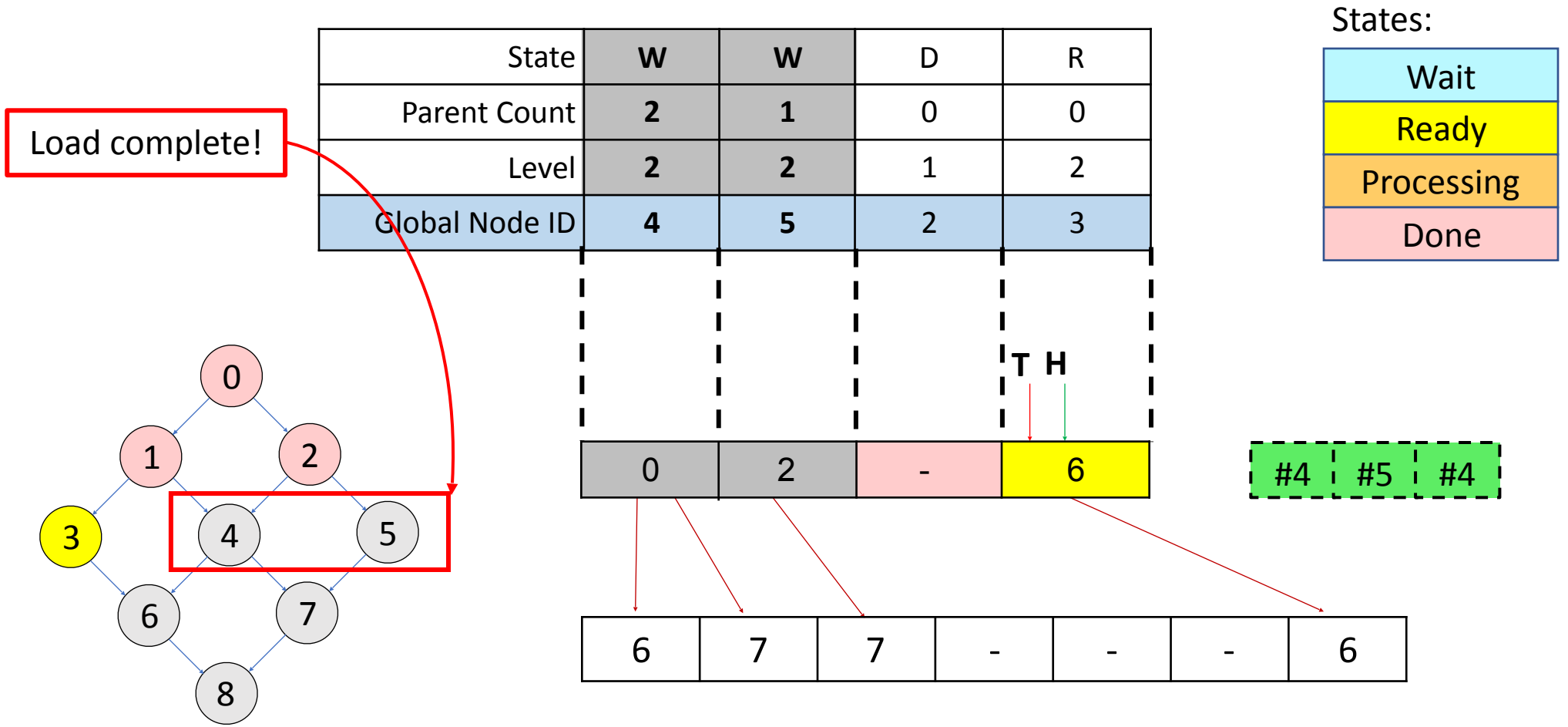
States:

- Wait
- Ready
- Processing
- Done





Example: …Reloading data



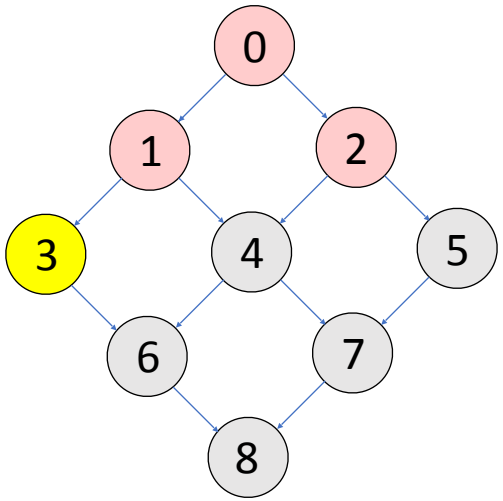
Example: …Reloading data

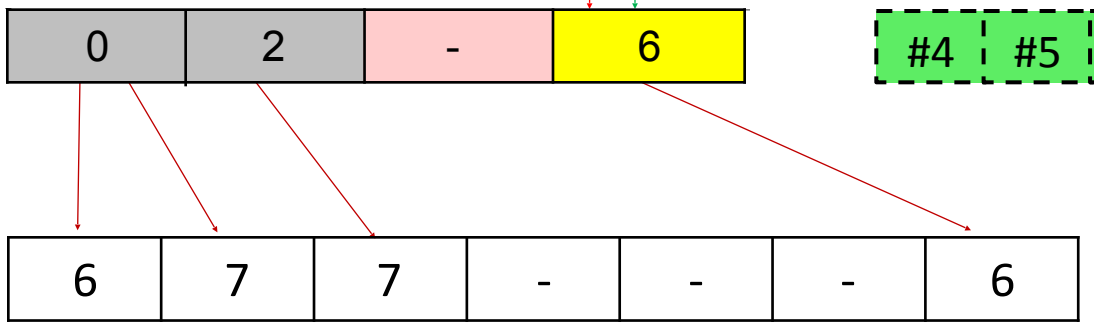
Load complete!

State	W	W	D	R
Parent Count	2	1	0	0
Level	2	2	1	2
Global Node ID	4	5	2	3

States:

- Wait
- Ready
- Processing
- Done





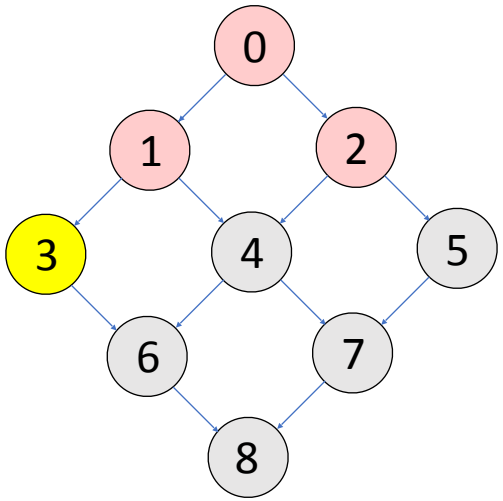
Example: …Reloading data

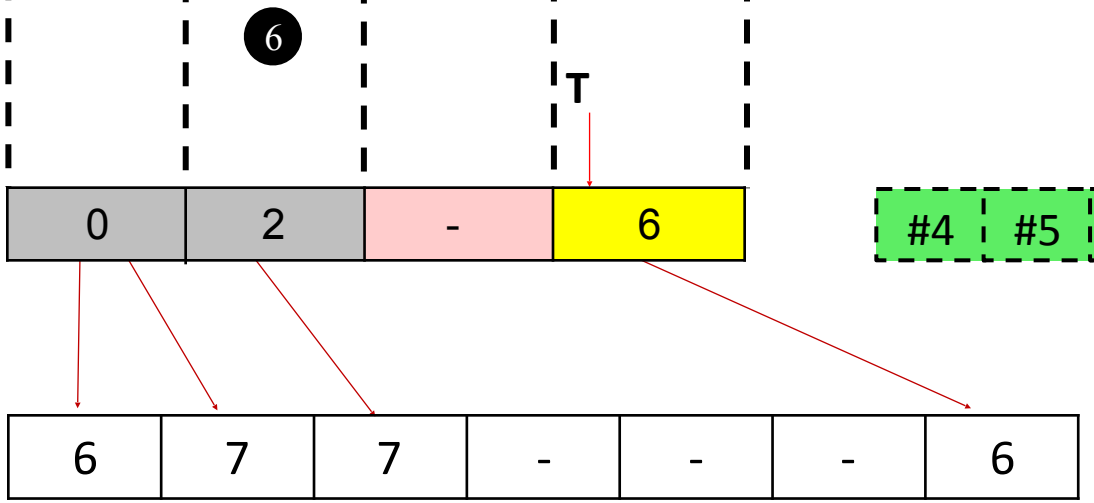
Load complete!

State	W	W	D	R
Parent Count	2	1	0	0
Level	2	2	1	2
Global Node ID	4	5	2	3

States:

- Wait
- Ready
- Processing
- Done



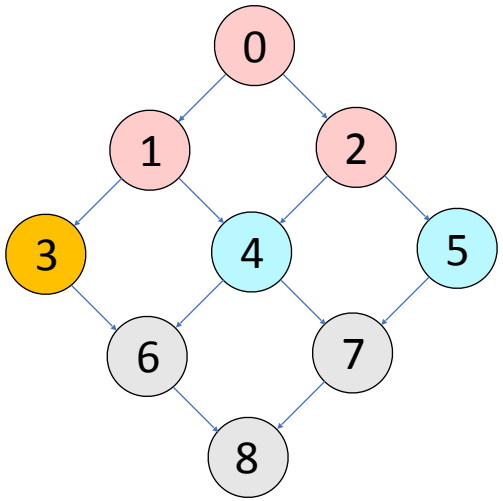


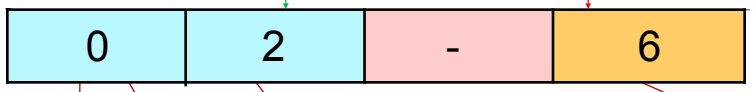
Example: Update Buffer Load

State	W	W	D	P
Parent Count	2	1	0	0
Level	2	2	1	2
Global Node ID	4	5	2	3

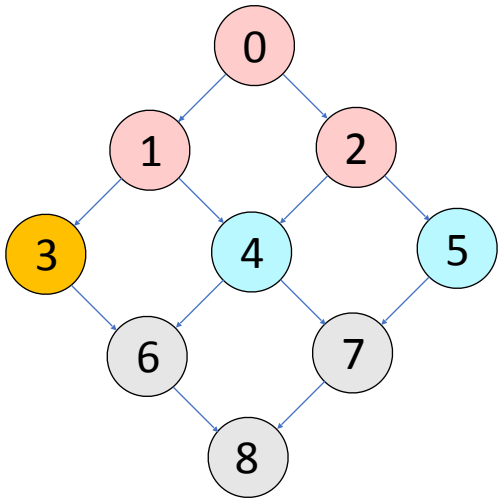
States:

- Wait
- Ready
- Processing
- Done

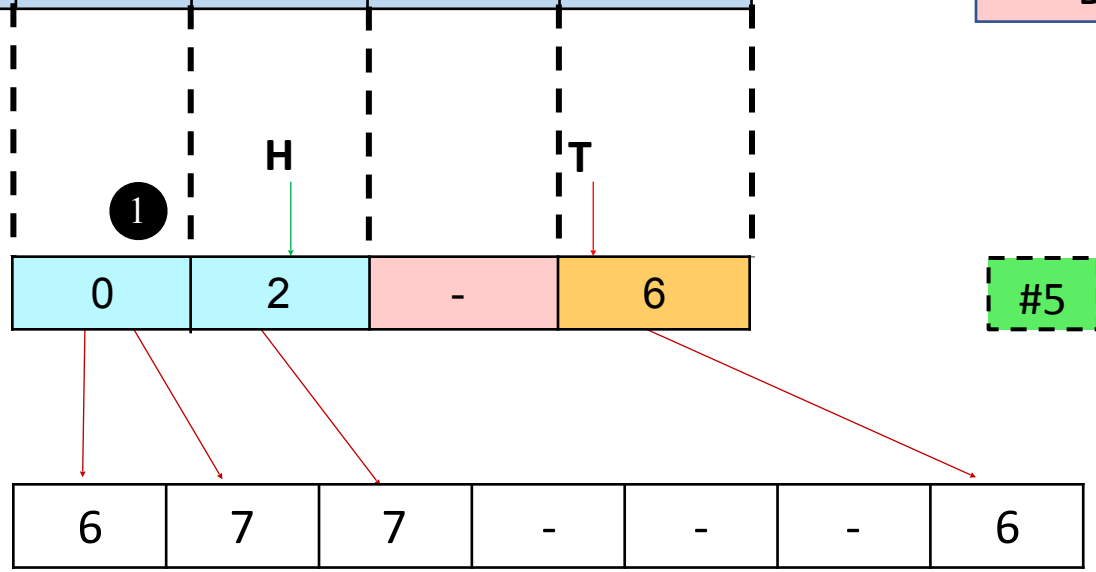




Example: Update Buffer Load



State	R 3	W	D	P
Parent Count	0 2	1	0	0
Level	2	2	1	2
Global Node ID	4	5	2	3



States:

Wait
Ready
Processing
Done

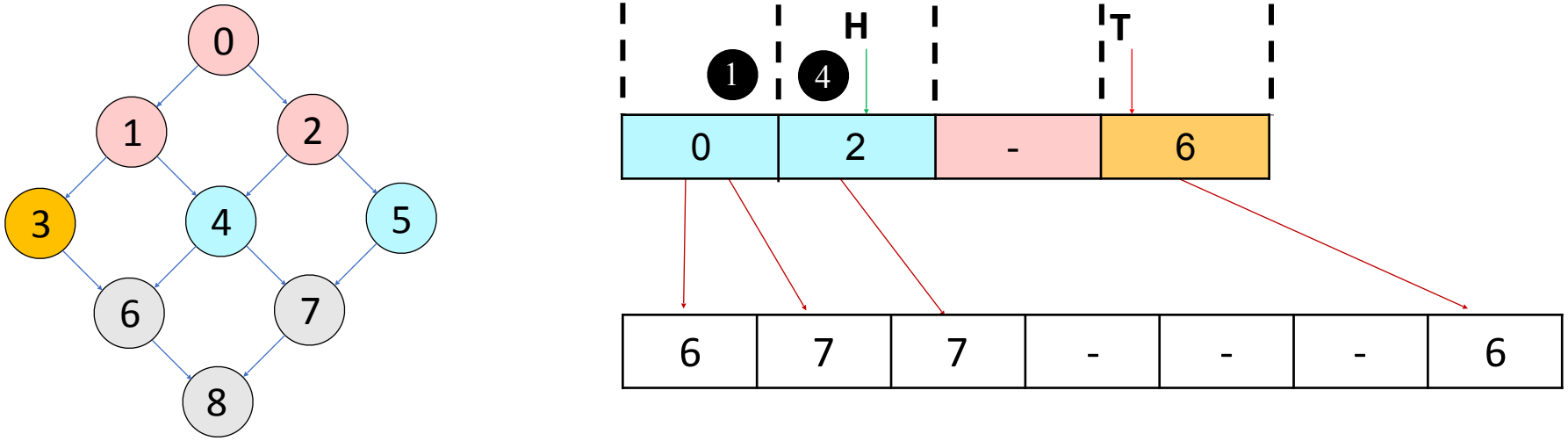
#5

Example: Update Buffer Load

State	R 3	R	D	P
Parent Count	0 2	0	0	0
Level	2	2	1	2
Global Node ID	4	5	2	3

States:

- Wait
- Ready
- Processing
- Done

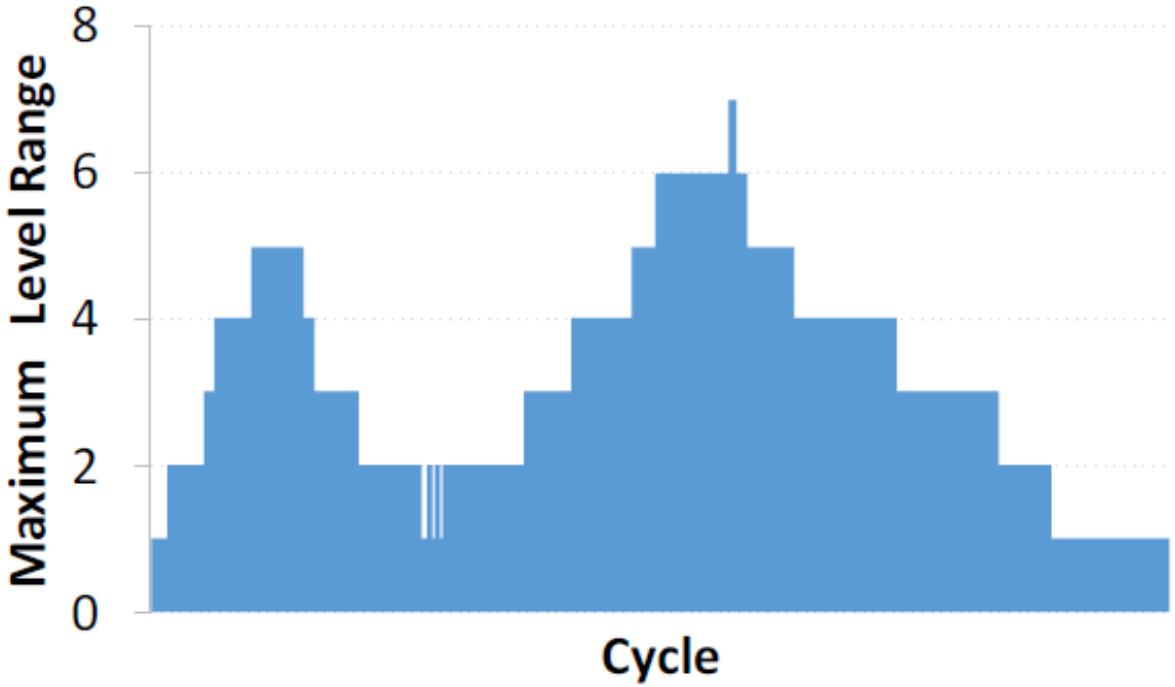


Challenges

- Minimizing global memory usage
  - Used CSR format
- Minimizing the buffer size
  - Limit Level Range
- Local Node/Edge Array size

Level Range

- Unbalanced execution may entail using the baseline TB scheduling policy (LRR).

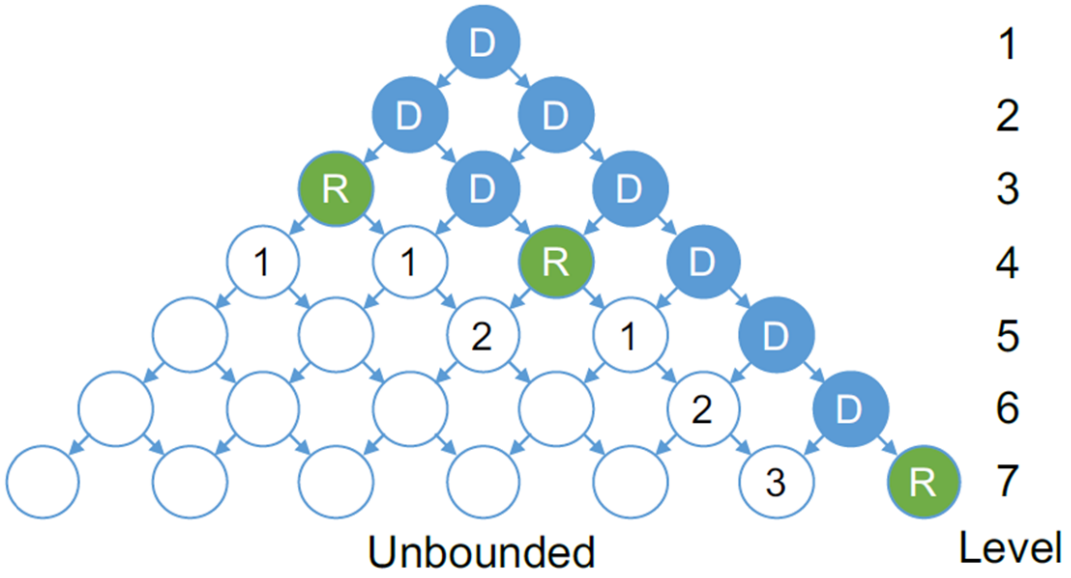


Sample benchmark (HEAT2D) w/ LRR scheduler

Level Range

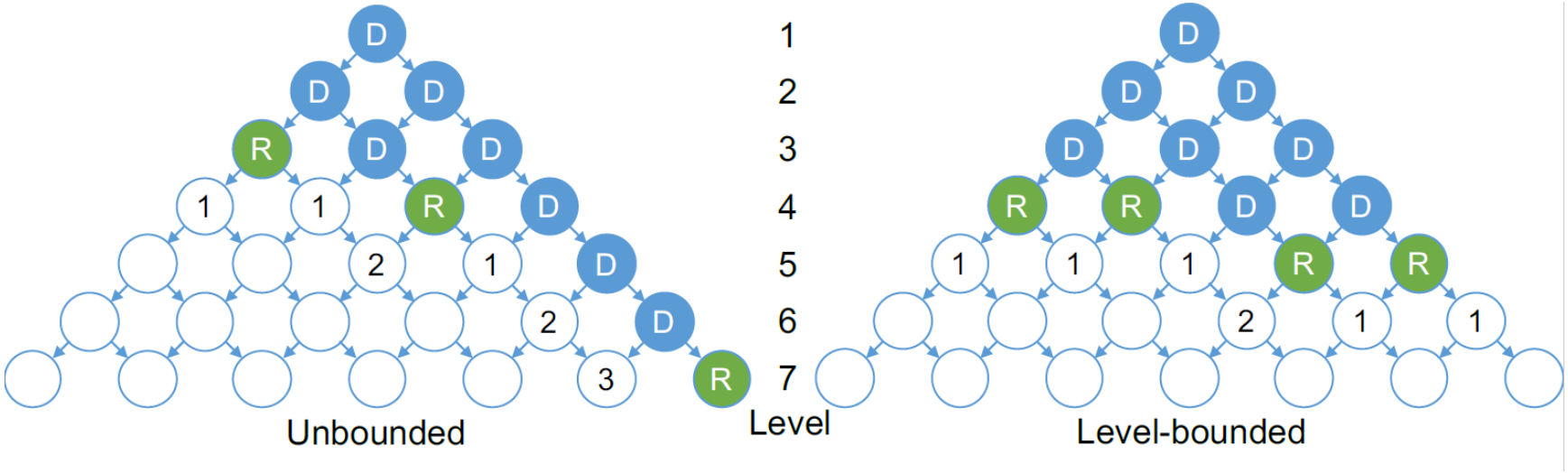
- Unbounded level range means:
  - Larger DGB is required
  - Limiting TB execution

Key challenge:
Efficient scheduling



Level-bound Scheduling (LVL)

- Prioritizing lower-level thread blocks in the graph
- More ready nodes → More parallelism
- Minimizing the buffering operation
- Limiting the level range to avoid serialization



Local Node Array Size

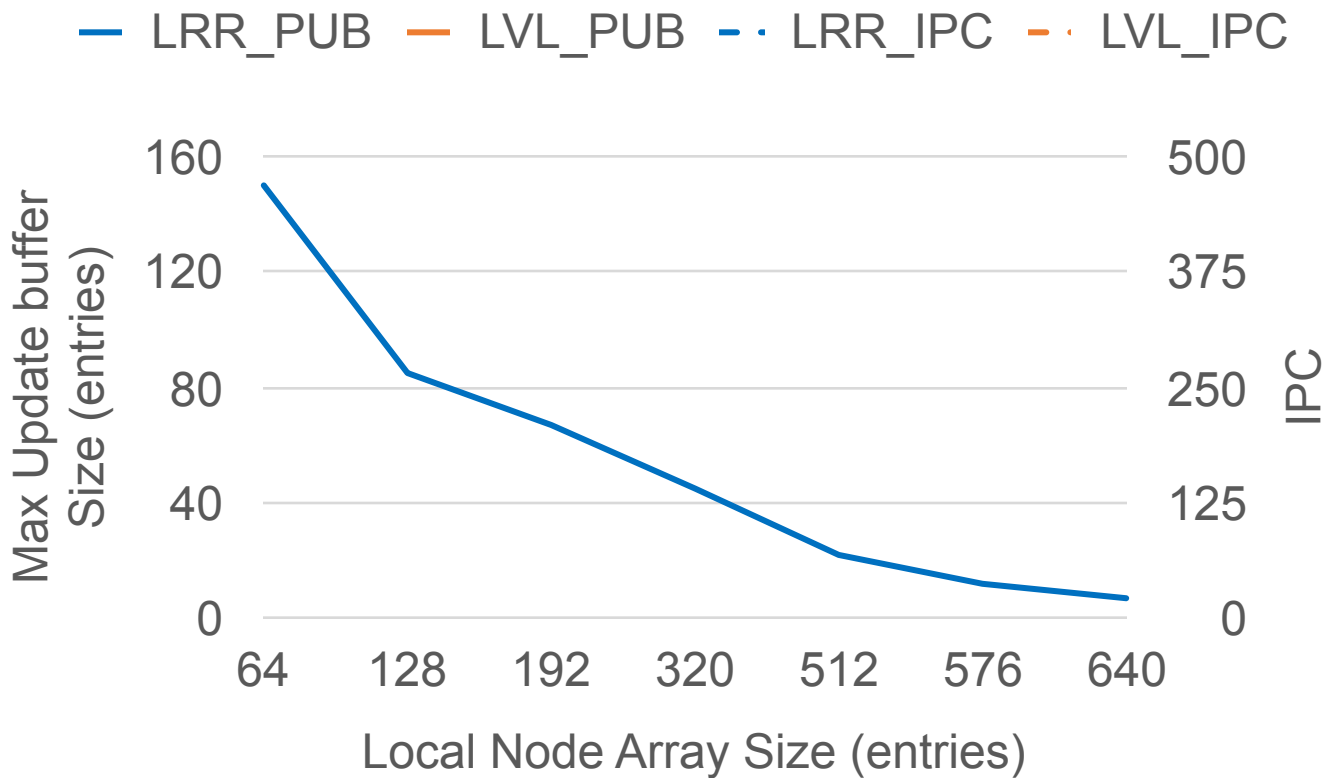
- Empirical estimation used

Local Node Array Size

- Empirical estimation used
- Reduce size
  - Until performance suffers

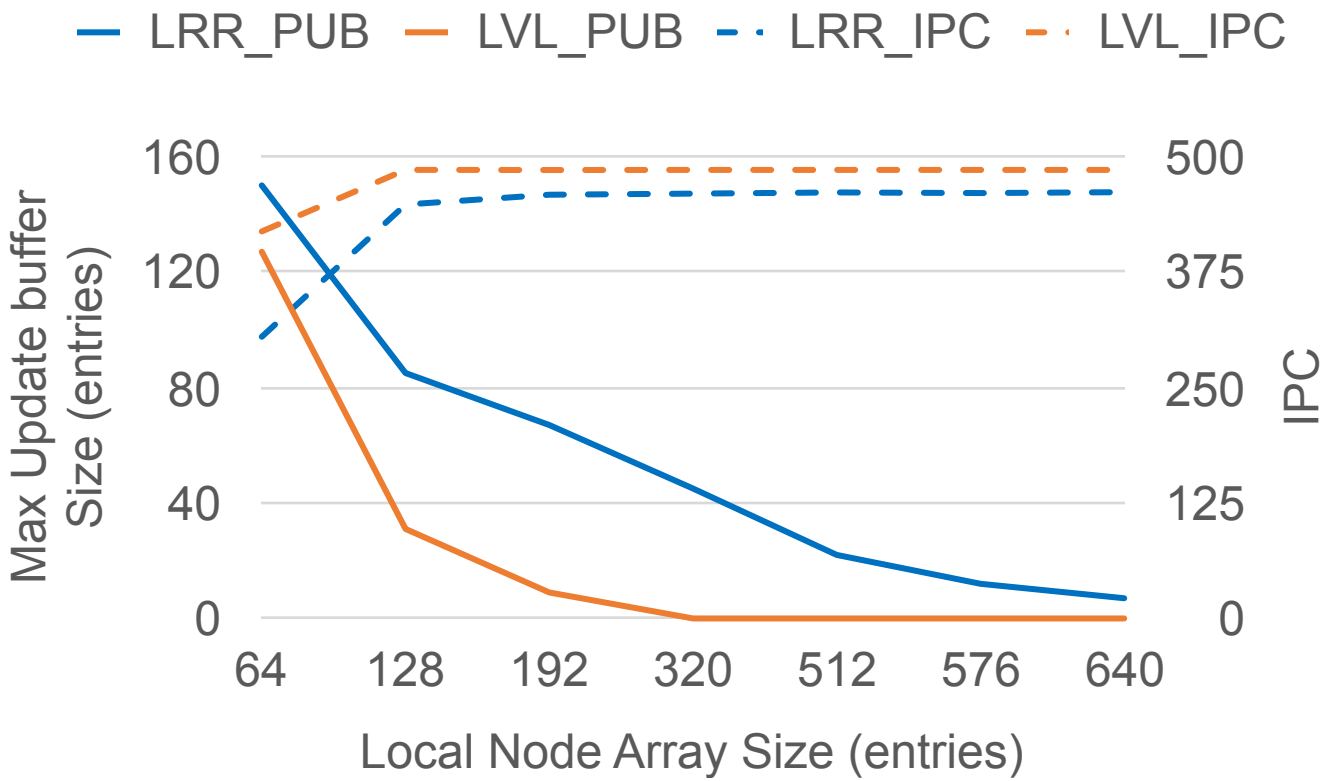
Local Node Array Size

- Empirical estimation used
- Reduce size
  - Until performance suffers



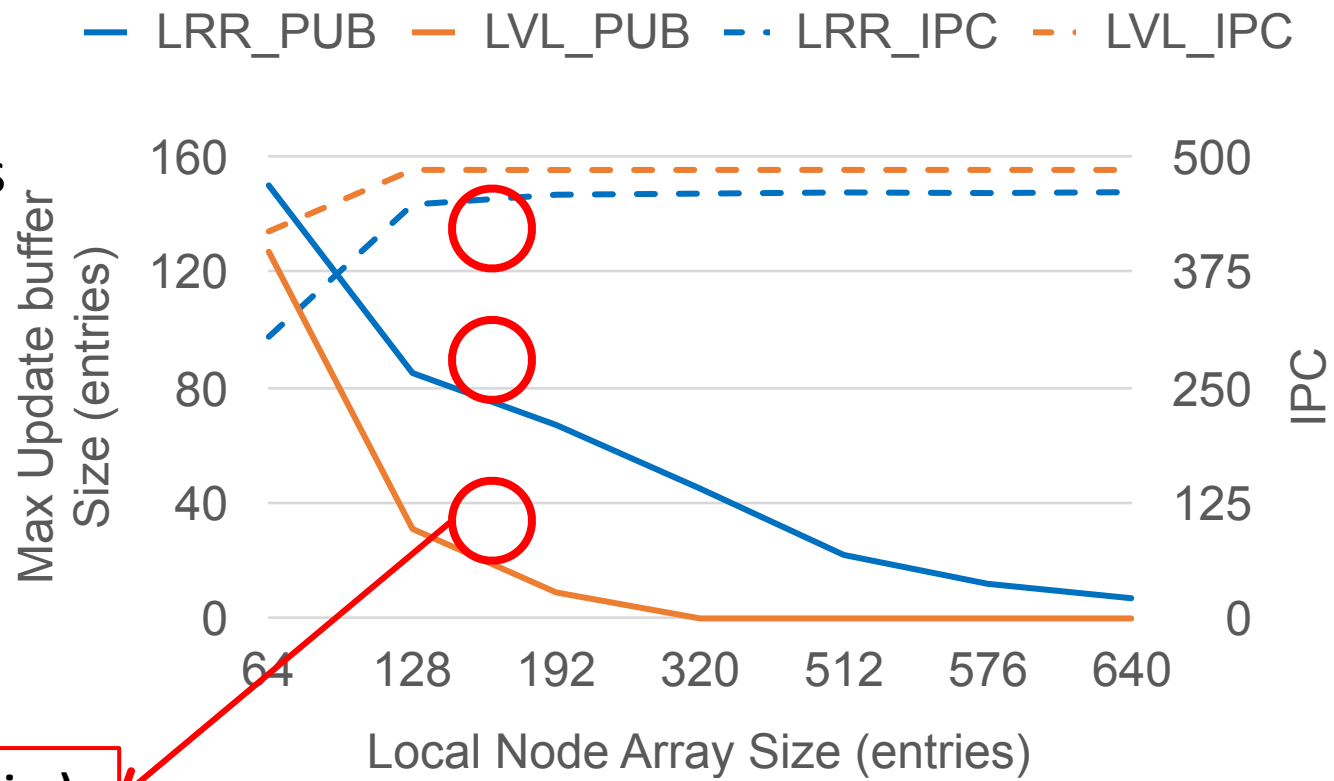
Local Node Array Size

- Empirical estimation used
- Reduce size
  - Until performance suffers



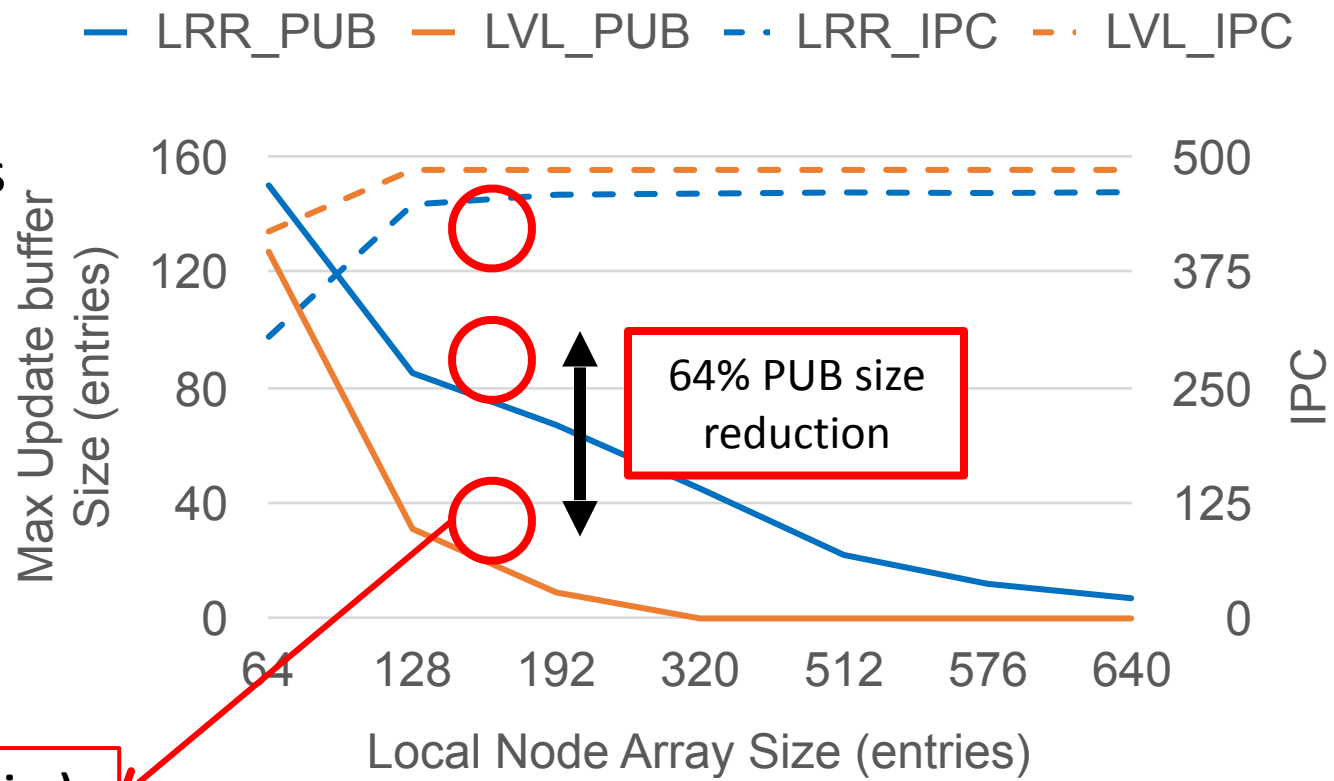
Local Node Array Size

- Empirical estimation used
- Reduce size
  - Until performance suffers



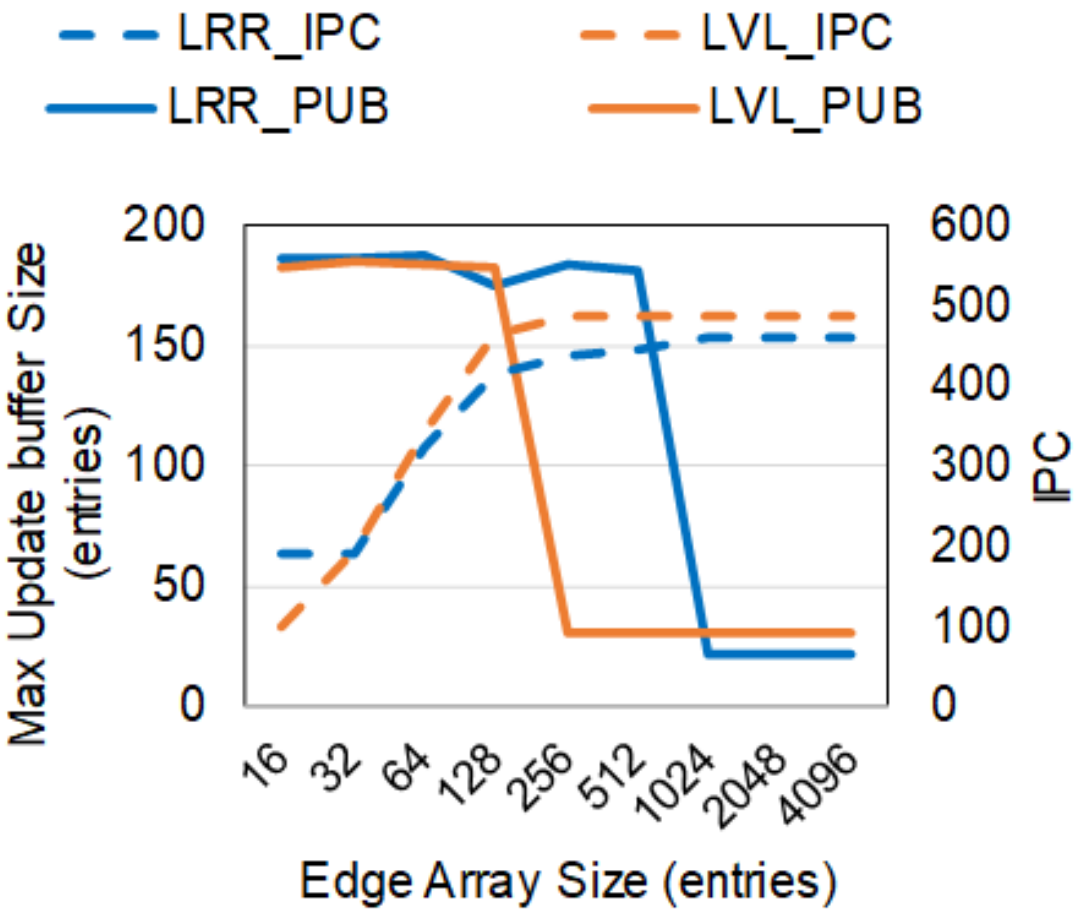
Local Node Array Size

- Empirical estimation used
- Reduce size
  - Until performance suffers
- LVL saves 64% PUB size



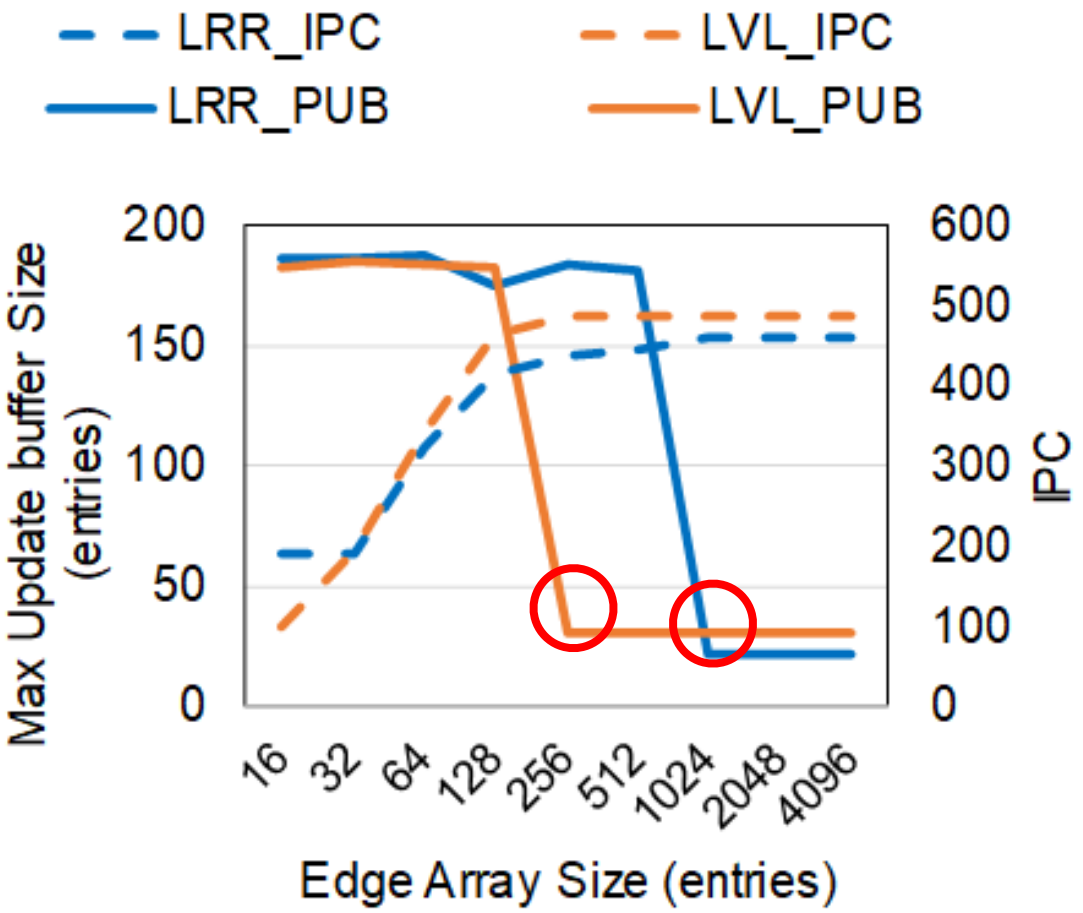
Local Edge Array Size

- Empirical estimation used



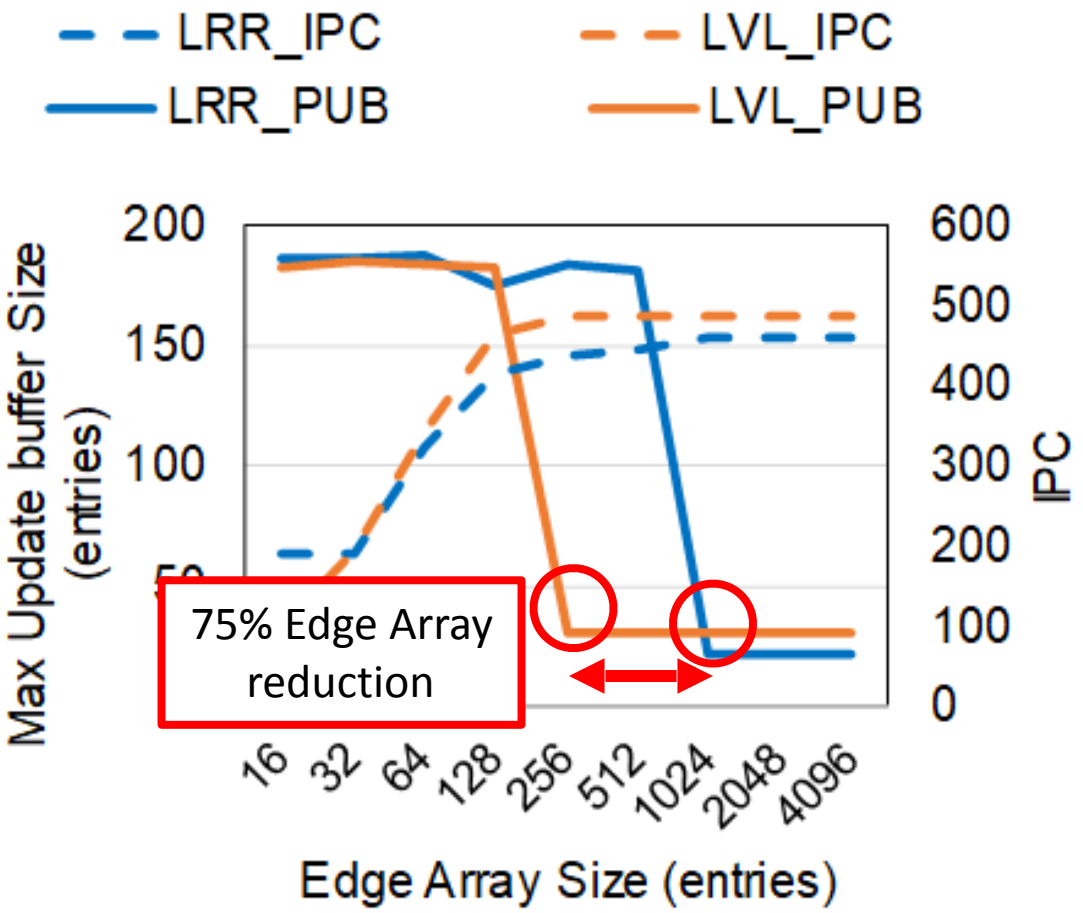
Local Edge Array Size

- Empirical estimation used



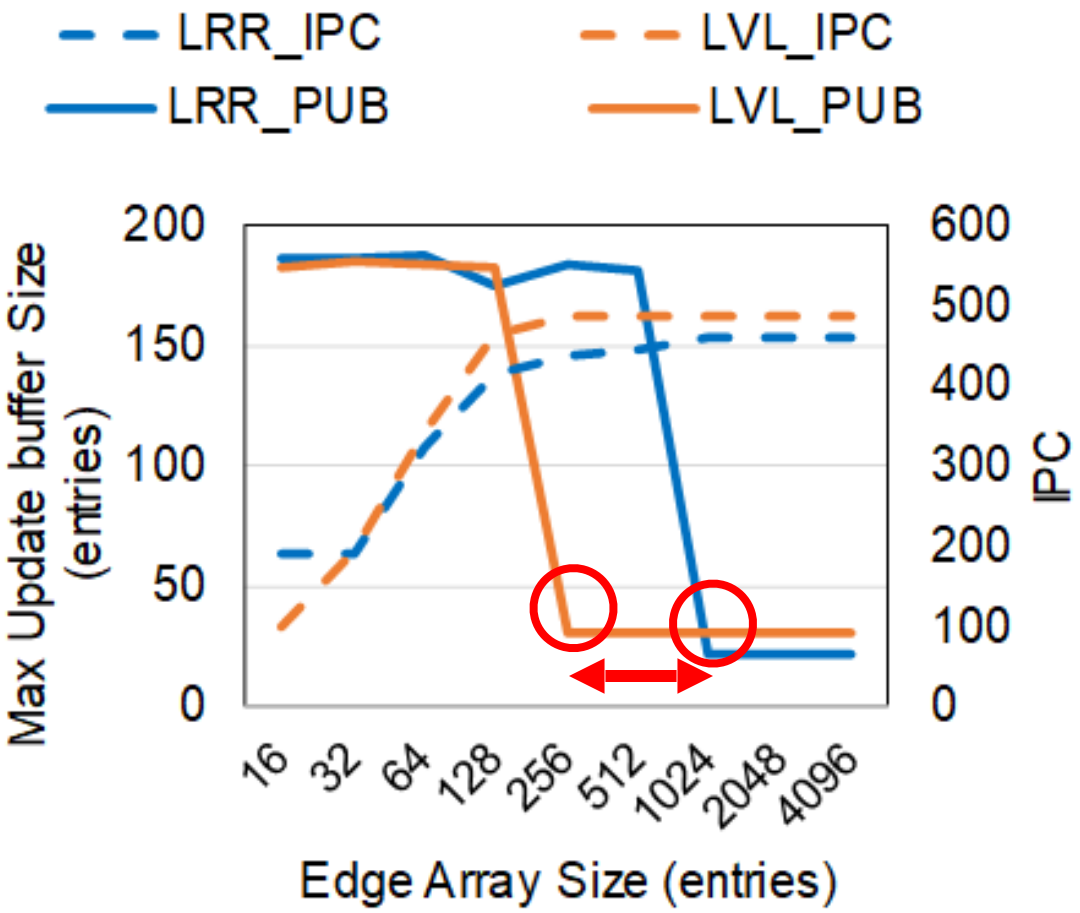
Local Edge Array Size

- Empirical estimation used
  - LVL requires 75% less storage



Local Edge Array Size

- Empirical estimation used
  - LVL requires 75% less storage
  - 256 entries



Evaluation

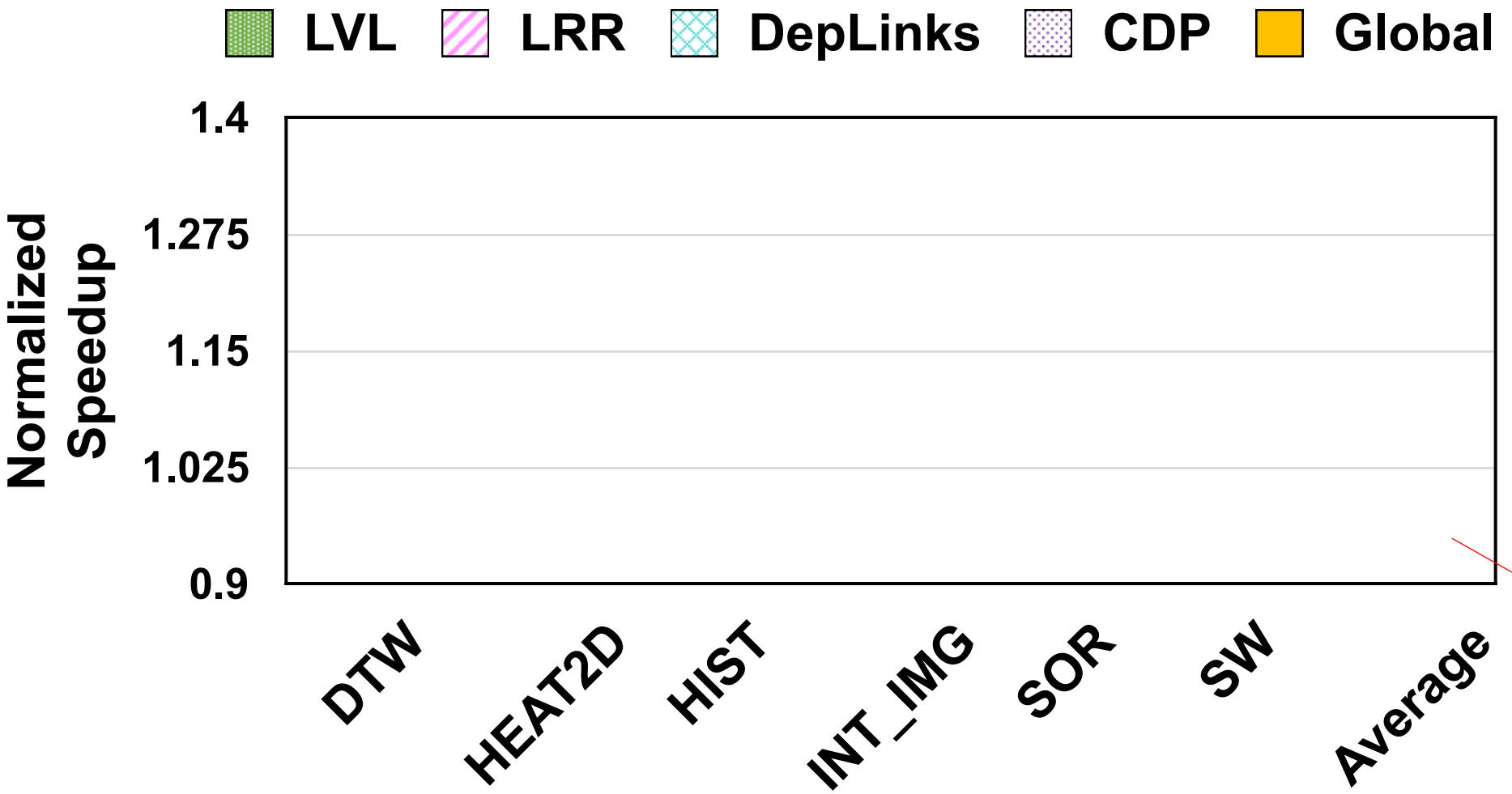
- Evaluation platform
  - GPGPU-Sim v3.2.2 (GTX480)
  - Six data dependency-heavy benchmarks
- Cases
  - Global, CDP
  - DepLinks primitives
  - LRR and LVL
    - LVL=3

Performance Breakdown

4K Graph Size

Performance Breakdown

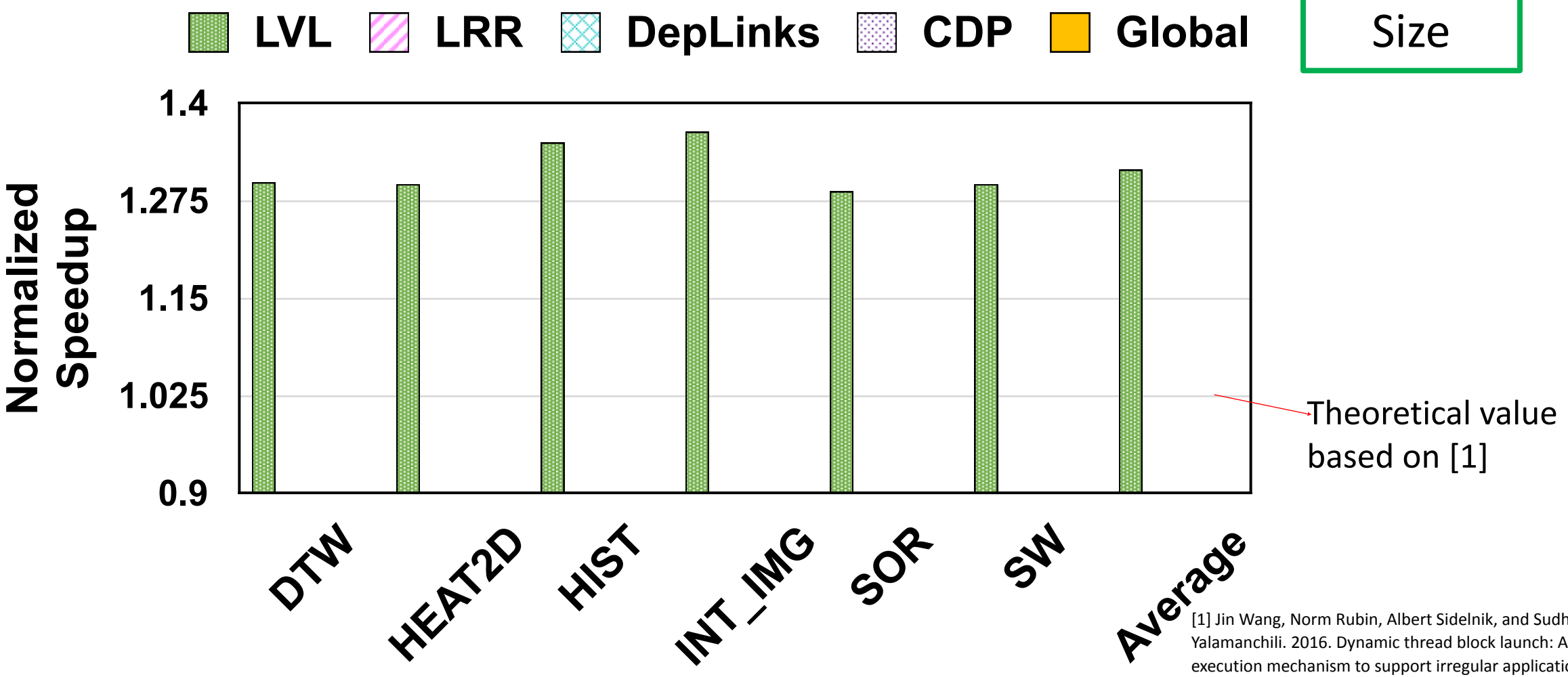
4K Graph Size



NOT accounting for data transfer time

Performance Breakdown

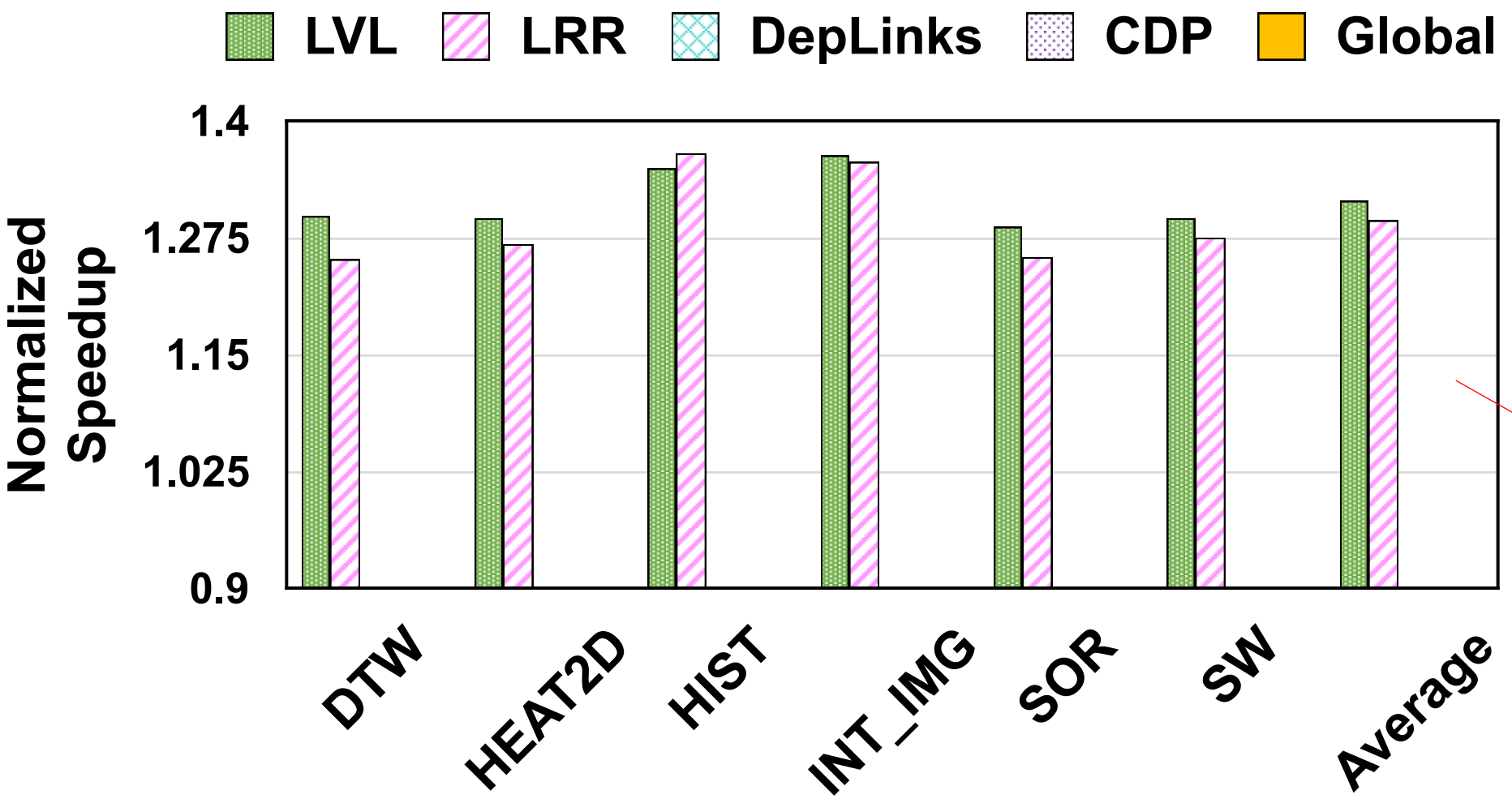
4K Graph Size



[1] Jin Wang, Norm Rubin, Albert Sidelnik, and Sudhakar Yalamanchili. 2016. Dynamic thread block launch: A lightweight execution mechanism to support irregular applications on GPUs.

Performance Breakdown

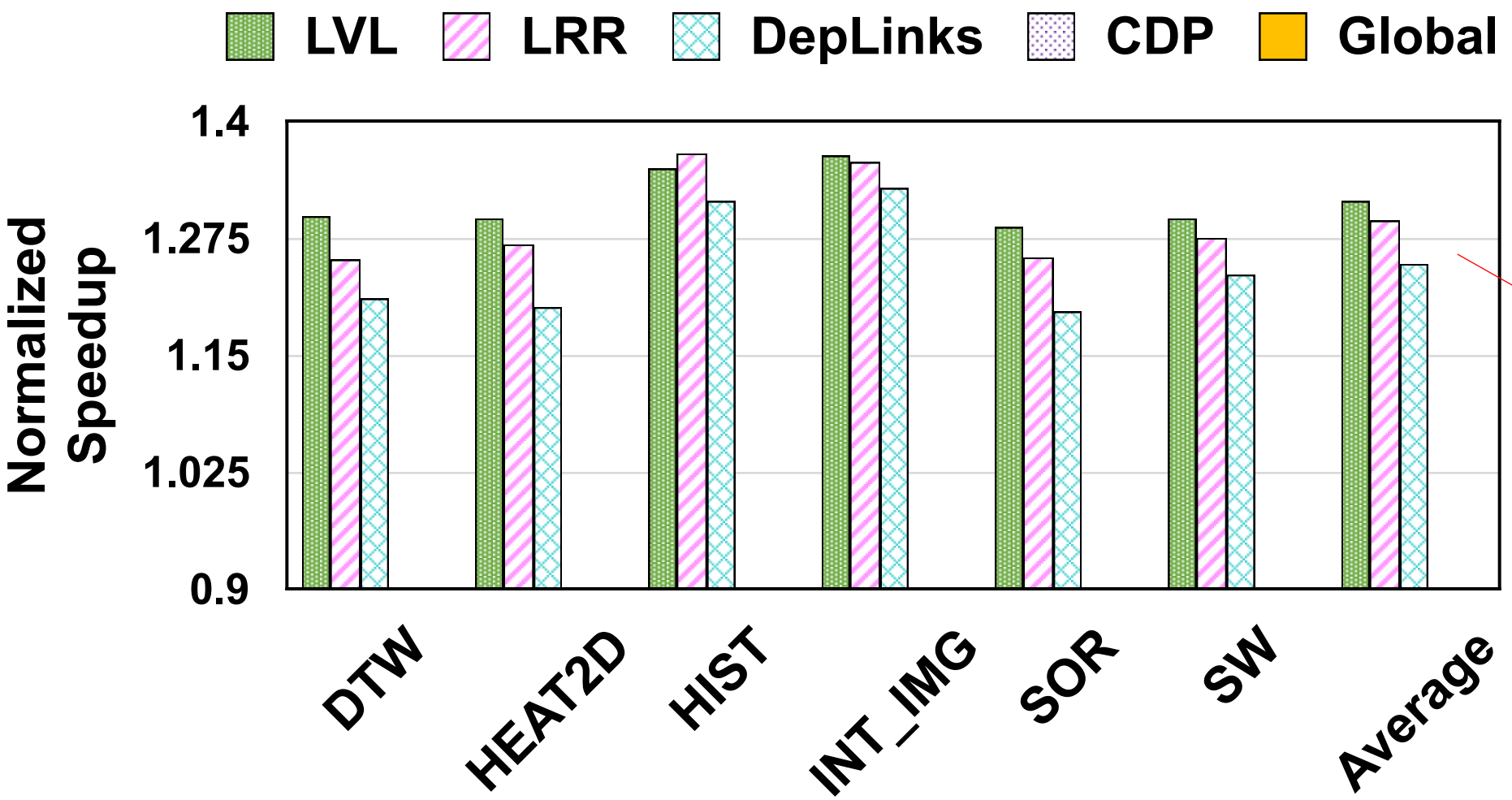
4K Graph Size



Barriers enforced by DepLinks instead; kernel launch overhead removed

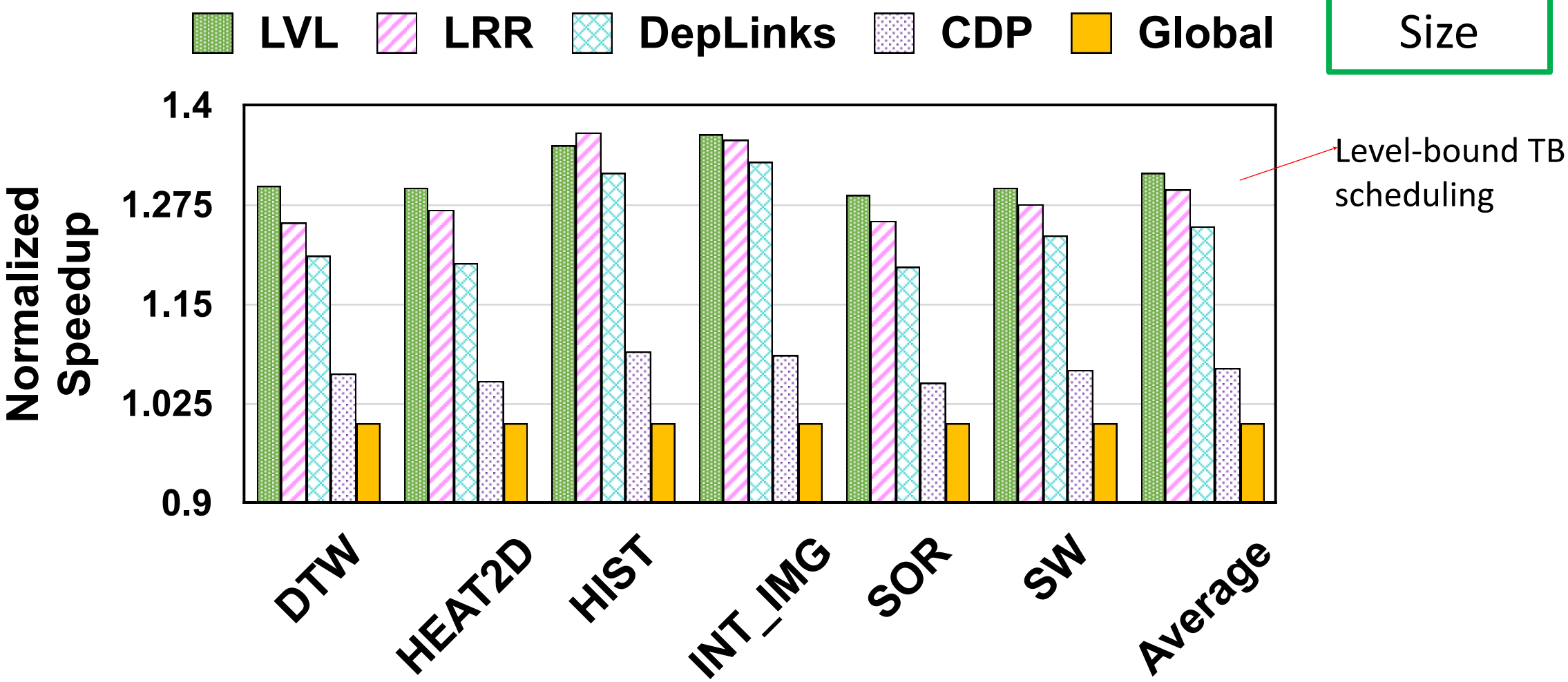
Performance Breakdown

4K Graph Size



Barriers removed. Nodes can now run ahead.

Performance Breakdown

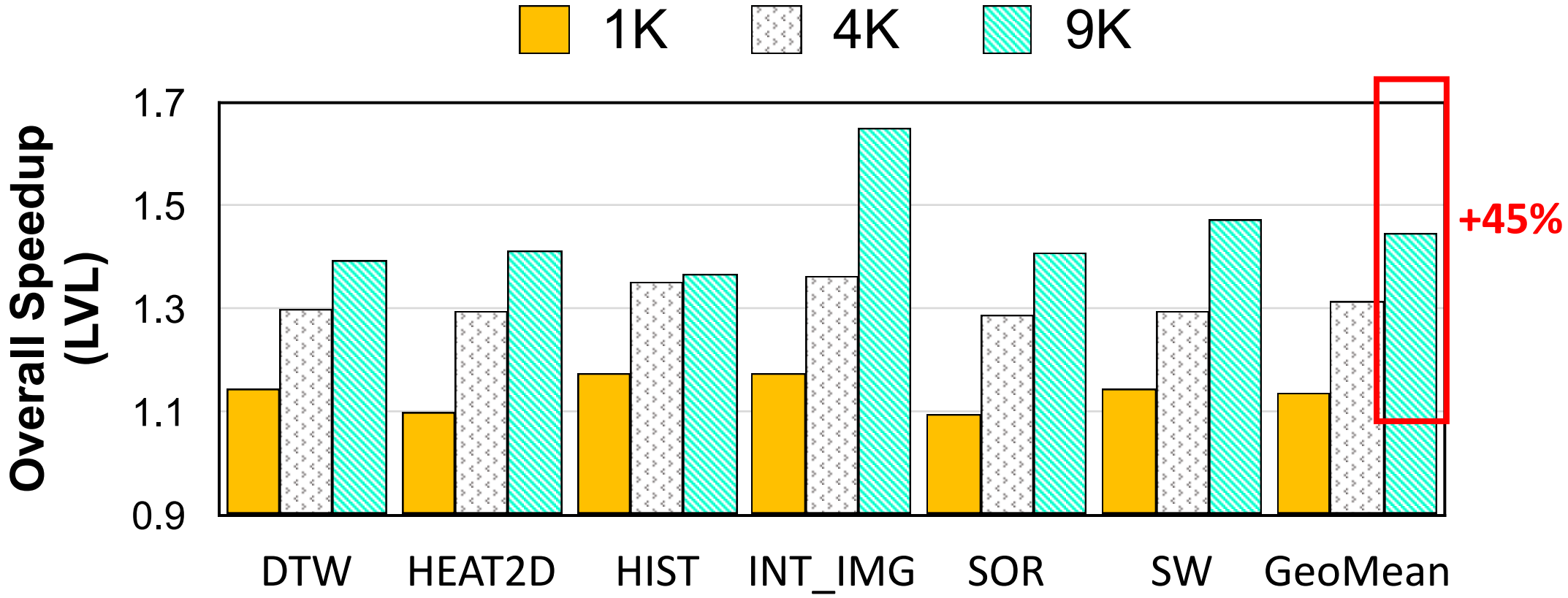


4K Graph Size

Level-bound TB scheduling

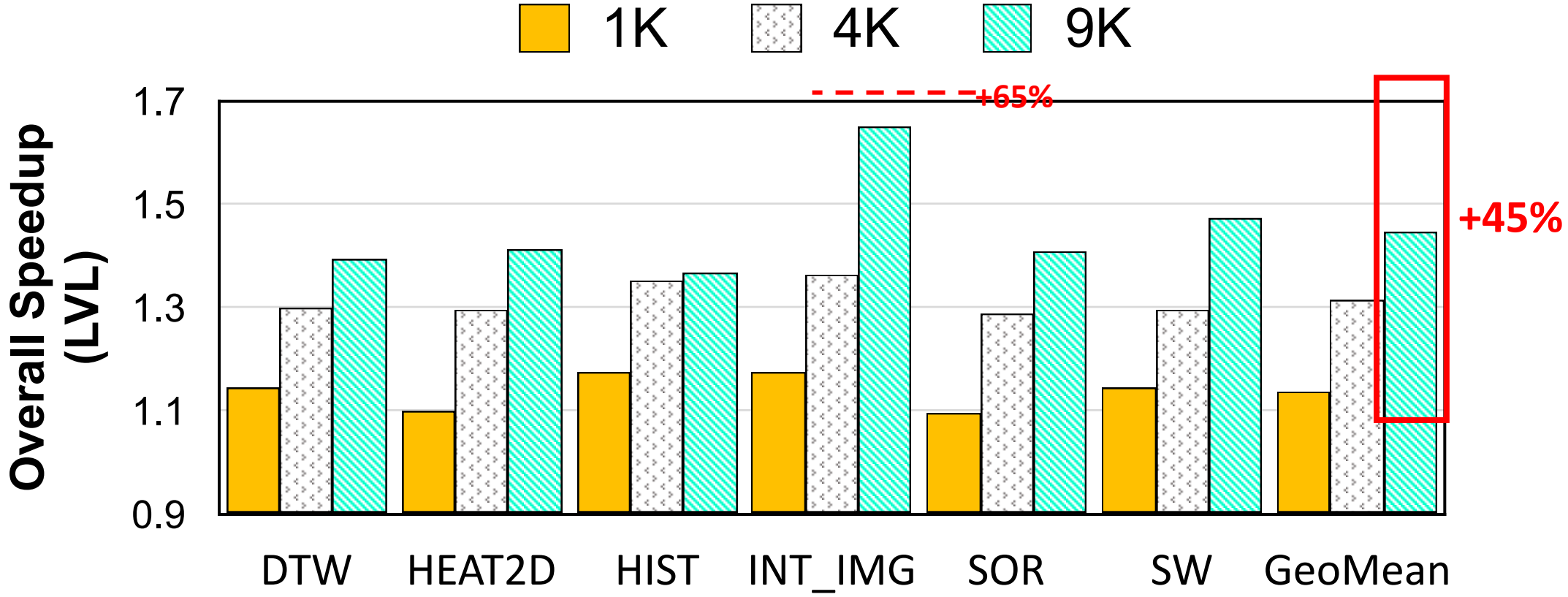
Performance

- Speedup across different graph sizes



Performance

- Speedup across different graph sizes



Evaluation Summary

- 2KB area overhead
- No significant impact on L2 miss rate
- Low global memory request overhead
  - 0.13% Average

Conclusion

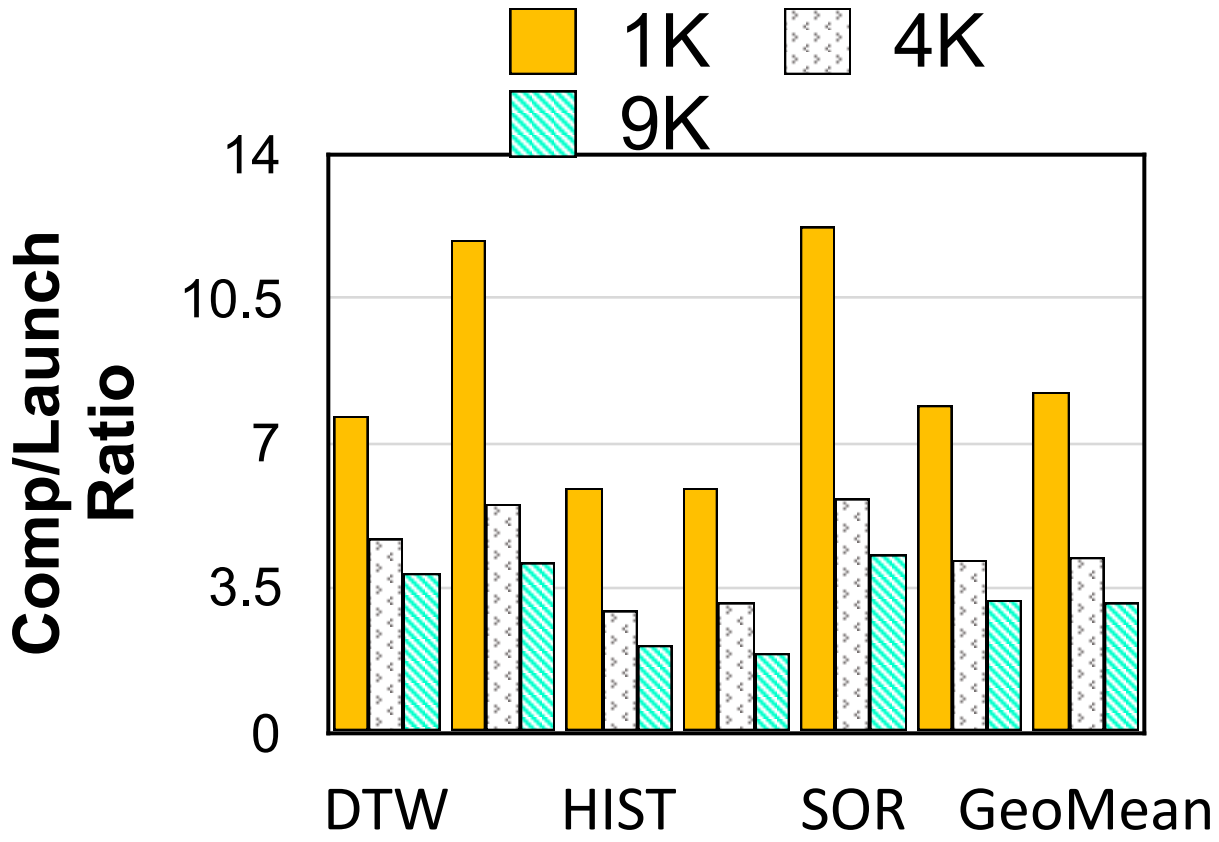
- Presenting *Wireframe*, hardware support for GPU data dependency
- Supporting generalized inter-block dependencies through hardware
- Minimizing buffering through level-bound TB scheduling
- 45% average speedup improvement over the baseline

Thank you!

Questions?

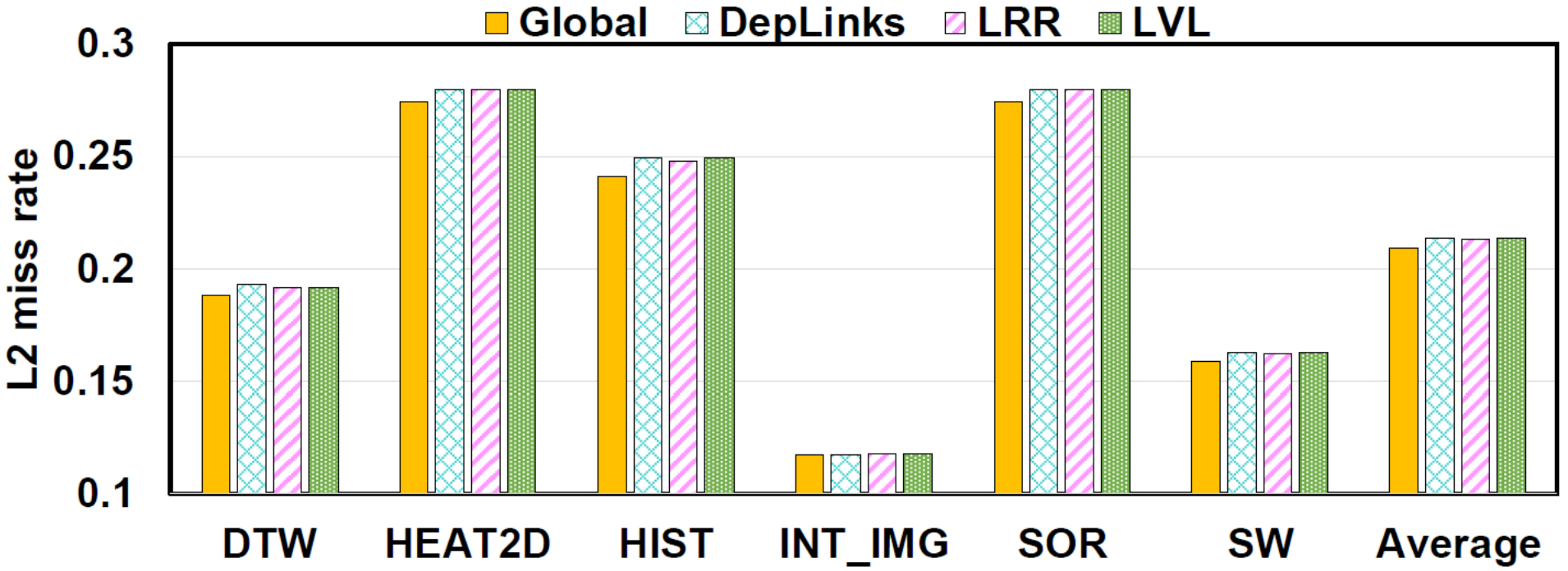
Computations vs Launch Overhead

- With a constant data size
  - Kernel launches increase with graph size
- is still sizable at 9K nodes.
  - times on average

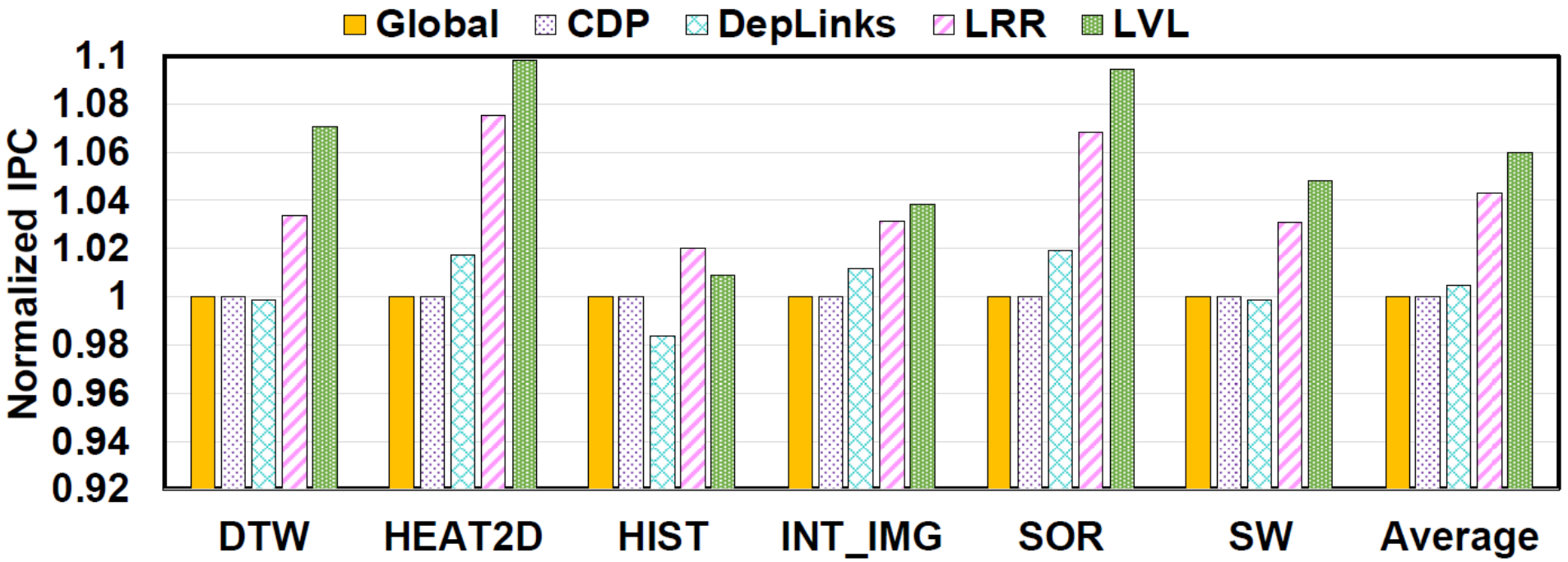


Performance

- Impact on L2 ~ 0.5%



Performance (IPC)



Thank you!

Questions?