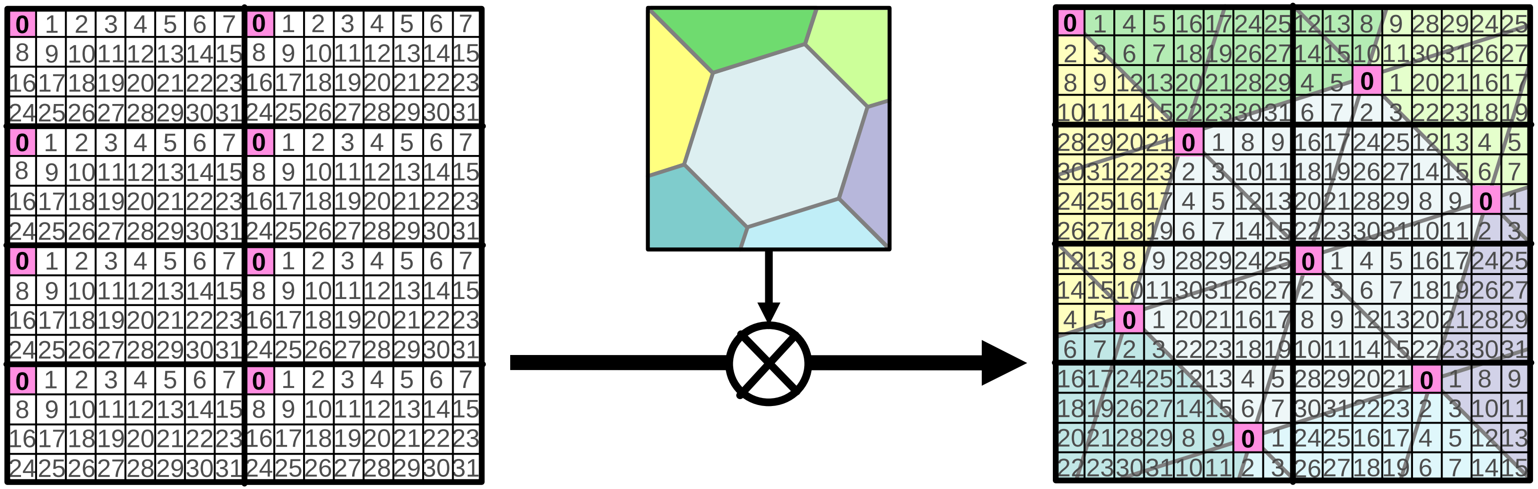
TOSHIBA

Hexagonal Storage Scheme for Interleaved Frame Buffers and Textures



0	1	2	3	4	5	6	7	**0**	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31	24	25	26	27	28	29	30	31
0	1	2	3	4	5	6	7	**0**	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31	24	25	26	27	28	29	30	31
0	1	2	3	4	5	6	7	**0**	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31	24	25	26	27	28	29	30	31
0	1	2	3	4	5	6	7	**0**	1	2	3	4	5	6	7
8	9	10	11	12	13	14	15	8	9	10	11	12	13	14	15
16	17	18	19	20	21	22	23	16	17	18	19	20	21	22	23
24	25	26	27	28	29	30	31	24	25	26	27	28	29	30	31

0	1	4	5	16	17	24	25	12	13	8	9	28	29	24	25
2	3	6	7	18	19	26	27	14	15	10	11	30	31	26	27
8	9	12	13	20	21	28	29	4	5	**0**	1	20	21	16	17
10	11	14	15	22	23	30	31	6	7	2	3	22	23	18	19
28	29	20	21	**0**	1	8	9	16	17	24	25	12	13	4	5
30	31	22	23	2	3	10	11	18	19	26	27	14	15	6	7
24	25	16	17	4	5	12	13	20	21	28	29	8	9	**0**	1
26	27	18	19	6	7	14	15	22	23	30	31	10	11	2	3
12	13	8	9	28	29	24	25	**0**	1	4	5	16	17	24	25
14	15	10	11	30	31	26	27	2	3	6	7	18	19	26	27
4	5	**0**	1	20	21	16	17	8	9	12	13	20	21	28	29
6	7	2	3	22	23	18	19	10	11	14	15	22	23	30	31
16	17	24	25	12	13	4	5	28	29	20	21	**0**	1	8	9
18	19	26	27	14	15	6	7	30	31	22	23	2	3	10	11
20	21	28	29	8	9	**0**	1	24	25	16	17	4	5	12	13
22	23	30	31	10	11	2	3	26	27	18	19	6	7	14	15

Yosuke Bando **Takahiro Saito** **Masahiro Fujita**

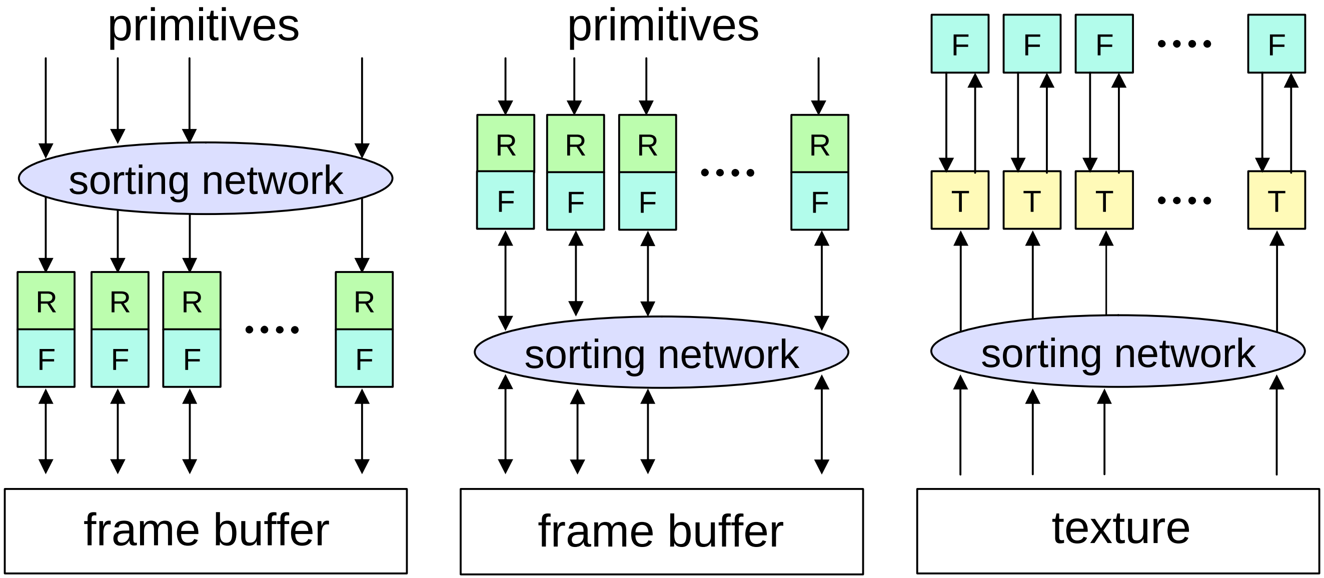
TOSHIBA Corporation

Outline

- Background
- Related work
- Hexagonal storage scheme
- Evaluation
- Results
- Conclusion

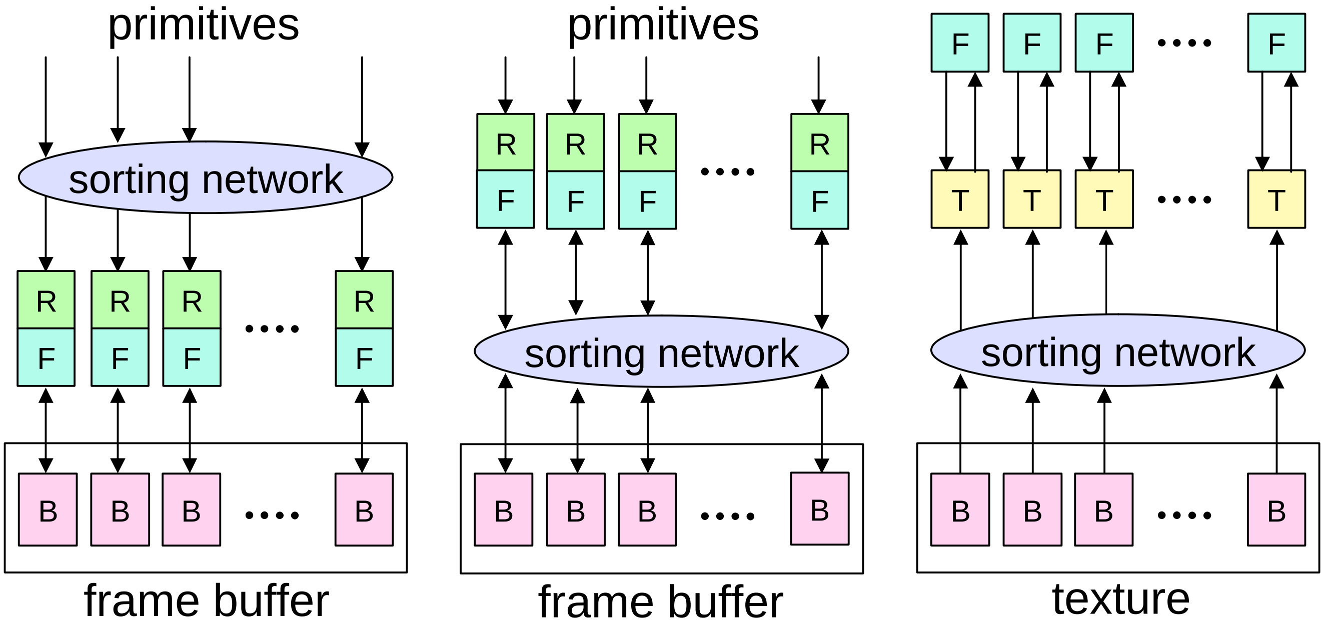
Background 1/2

- GPUs are highly parallelized
  - Multiple rasterizers, frag processors, tex units



Background 1/2

- GPUs are highly parallelized
  - Multiple rasterizers, frag processors, tex units
- Memory is also parallelized
  - Multiple memory banks



Background 2/2

- Problem: memory access load imbalance
  - Each bank stores disjoint portion of the buffer
  - Accesses cannot be dynamically distributed

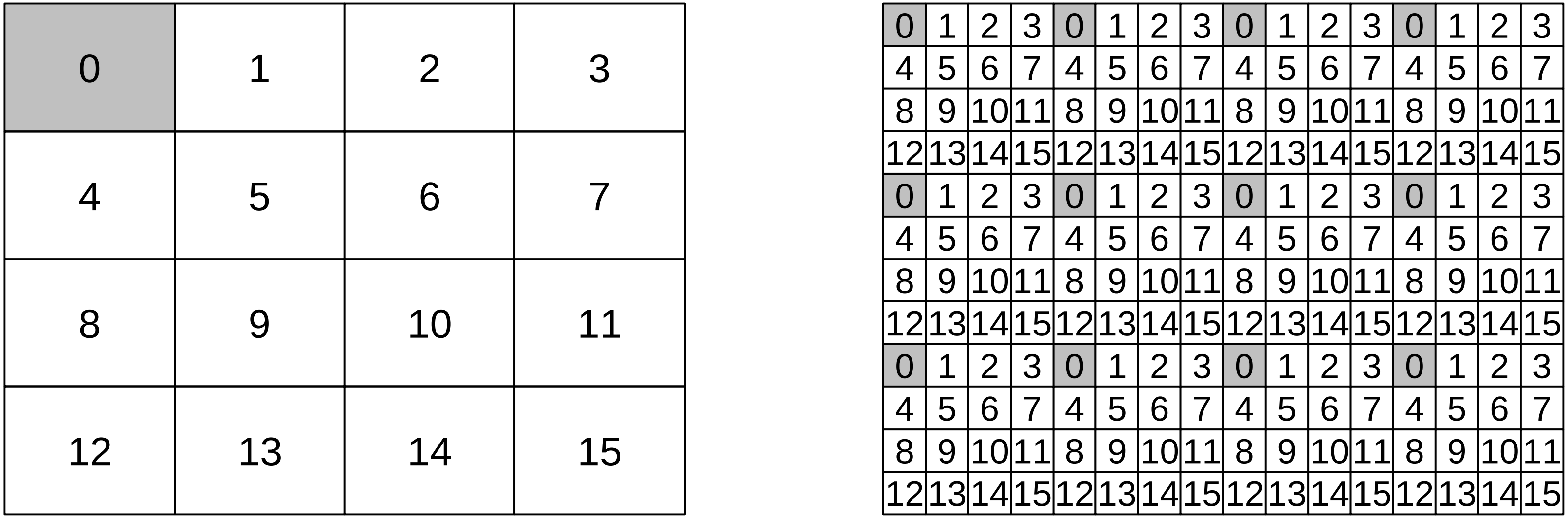
- Goal: to minimize load imbalance
  - Elaborate storage scheme
  - Static, passive load balancing

Outline

- Background
- Related work
- Hexagonal storage scheme
- Evaluation
- Results
- Conclusion

Related Work 1/2

- Divide the buffer into N regions
  - – Each of the N bank stores one region
    - [Parke 80]
- Divide the buffer into many small regions
  - – Interleaved assignment between regions and banks
    - [Fuchs 77, Potmesil 89, Deering 94, Mueller 95]

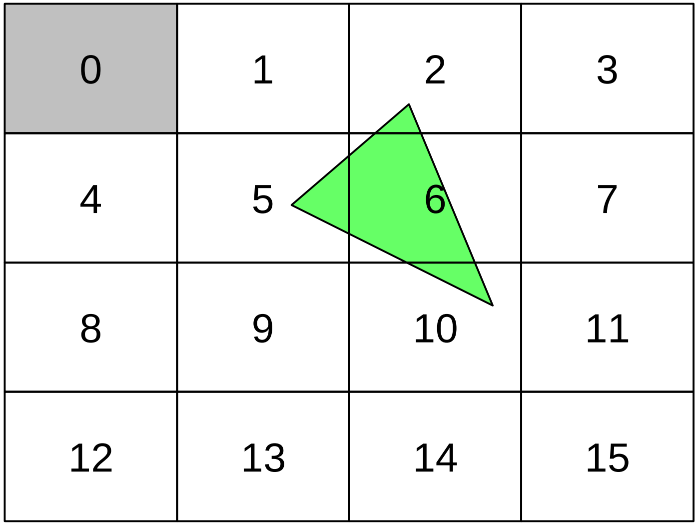


0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

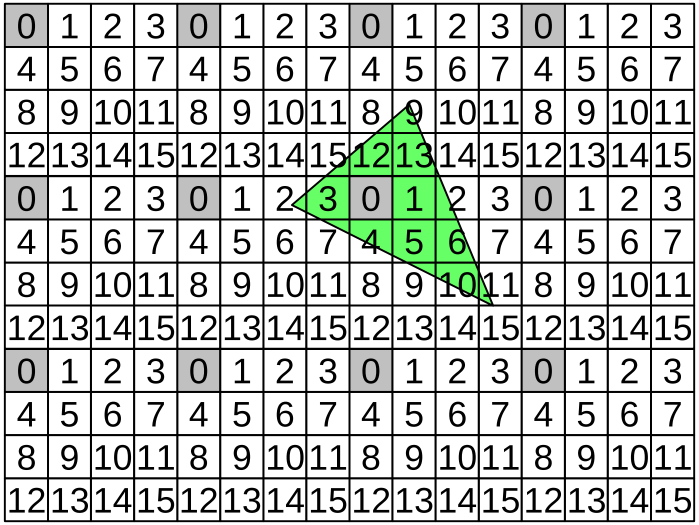
0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7	4	5	6	7	4	5	6	7
8	9	10	11	8	9	10	11	8	9	10	11	8	9	10	11
12	13	14	15	12	13	14	15	12	13	14	15	12	13	14	15
0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7	4	5	6	7	4	5	6	7
8	9	10	11	8	9	10	11	8	9	10	11	8	9	10	11
12	13	14	15	12	13	14	15	12	13	14	15	12	13	14	15
0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7	4	5	6	7	4	5	6	7
8	9	10	11	8	9	10	11	8	9	10	11	8	9	10	11
12	13	14	15	12	13	14	15	12	13	14	15	12	13	14	15

Related Work 2/2

- **Interleaved assignment is preferable**
  - **Primitives are often small compared to the screen size**



0	1	2	3
4	5	6	7
8	9	10	11
12	13	14	15

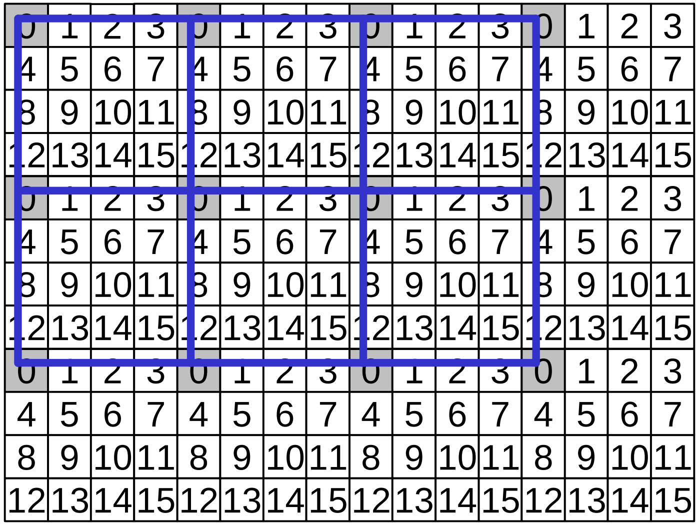


0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7	4	5	6	7	4	5	6	7
8	9	10	11	8	9	10	11	8	9	10	11	8	9	10	11
12	13	14	15	12	13	14	15	12	13	14	15	12	13	14	15
0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7	4	5	6	7	4	5	6	7
8	9	10	11	8	9	10	11	8	9	10	11	8	9	10	11
12	13	14	15	12	13	14	15	12	13	14	15	12	13	14	15
0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7	4	5	6	7	4	5	6	7
8	9	10	11	8	9	10	11	8	9	10	11	8	9	10	11
12	13	14	15	12	13	14	15	12	13	14	15	12	13	14	15

TOSHIBA

Related Work 2/2

- **Interleaved assignment is preferable**
  - **Primitives are often small compared to the screen size**
- **But rectangular patterns are not optimal**

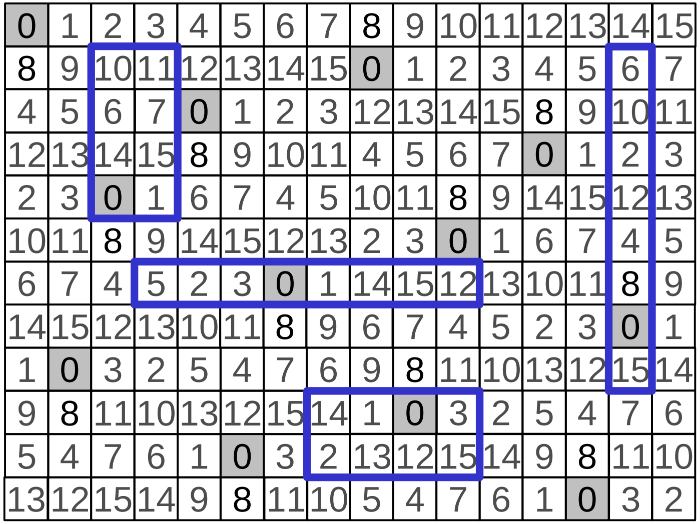


0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7	4	5	6	7	4	5	6	7
8	9	10	11	8	9	10	11	8	9	10	11	8	9	10	11
12	13	14	15	12	13	14	15	12	13	14	15	12	13	14	15
0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7	4	5	6	7	4	5	6	7
8	9	10	11	8	9	10	11	8	9	10	11	8	9	10	11
12	13	14	15	12	13	14	15	12	13	14	15	12	13	14	15
0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7	4	5	6	7	4	5	6	7
8	9	10	11	8	9	10	11	8	9	10	11	8	9	10	11
12	13	14	15	12	13	14	15	12	13	14	15	12	13	14	15

Related Work 2/2

- Interleaved assignment is preferable
  - Primitives are often small compared to the screen size
- But rectangular patterns are not optimal
- Multiaccess Frame Buffer [Harper 94]
  - An example of non-rectangular patterns

Regions in any rectangle whose area is N/2 are assigned to mutually different banks



0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
8	9	10	11	12	13	14	15	0	1	2	3	4	5	6	7
4	5	6	7	0	1	2	3	12	13	14	15	8	9	10	11
12	13	14	15	8	9	10	11	4	5	6	7	0	1	2	3
2	3	0	1	6	7	4	5	10	11	8	9	14	15	12	13
10	11	8	9	14	15	12	13	2	3	0	1	6	7	4	5
6	7	4	5	2	3	0	1	14	15	12	13	10	11	8	9
14	15	12	13	10	11	8	9	6	7	4	5	2	3	0	1
1	0	3	2	5	4	7	6	9	8	11	10	13	12	15	14
9	8	11	10	13	12	15	14	1	0	3	2	5	4	7	6
5	4	7	6	1	0	3	2	13	12	15	14	9	8	11	10
13	12	15	14	9	8	11	10	5	4	7	6	1	0	3	2

Related Work 2/2

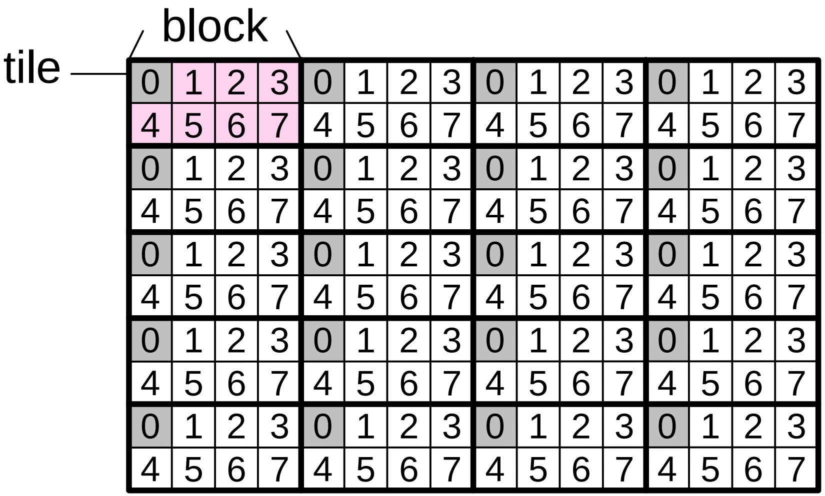
- Interleaved assignment is preferable
  - Primitives are often small compared to the screen size
- But rectangular patterns are not optimal
- Multiaccess Frame Buffer [Harper 94]
  - An example of non-rectangular patterns
- Are there better ones?

Outline

- Background
- Related work
- Hexagonal storage scheme
- Evaluation
- Results
- Conclusion

Our Storage Scheme

- Divide the buffer into square *tiles*
- To simplify bank ID calculation, assume
  - The number of banks, N, is a power of two
  - N tiles are clustered into a *block*
  - Tiles in a block are stored in mutually different banks
- Place the tiles stored in each bank uniformly

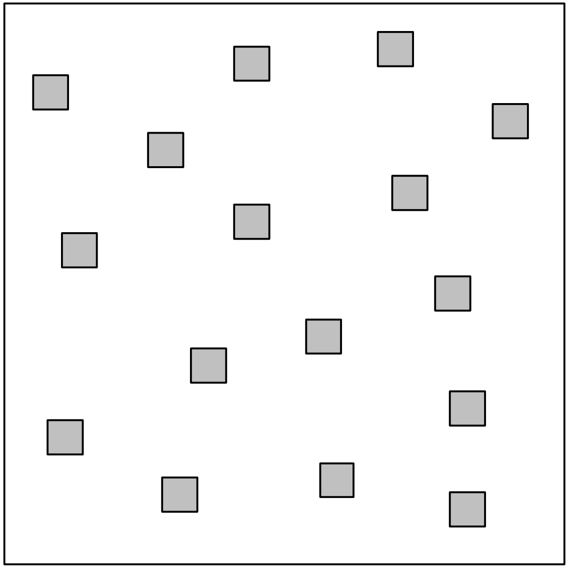


0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7	4	5	6	7	4	5	6	7
0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7	4	5	6	7	4	5	6	7
0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7	4	5	6	7	4	5	6	7
0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7	4	5	6	7	4	5	6	7
0	1	2	3	0	1	2	3	0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7	4	5	6	7	4	5	6	7

# of memory banks: 8
block size: 4 x 2

What Is "Uniform?"

- Ideally, regular hexagonal placement
  - Equidistant to the neighboring tiles
    - Maximizes the minimum distance
  - Isotropic



16 tiles are distributed.

buffer size: 64 x 64

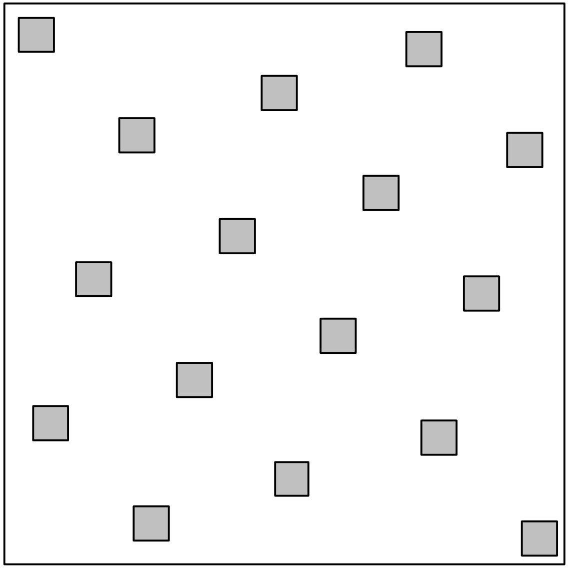
tile size: 4 x 4

# of memory banks: 16

# of tiles stored in one bank: 16

What Is “Uniform?”

- **Ideally, regular hexagonal placement**
  - **Equidistant to the neighboring tiles**
    - **Maximizes the minimum distance**
  - **Isotropic**



If you move them so that they are equidistant to their neighbors…

buffer size: 64 x 64

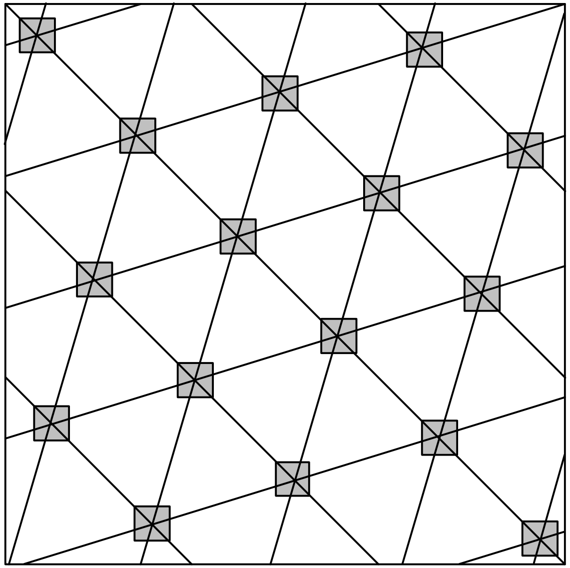
tile size: 4 x 4

# of memory banks: 16

# of tiles stored in one bank: 16

What Is "Uniform?"

- Ideally, regular hexagonal placement
  - Equidistant to the neighboring tiles
    - Maximizes the minimum distance
  - Isotropic



The placement will become hexagonal!

buffer size: 64 x 64

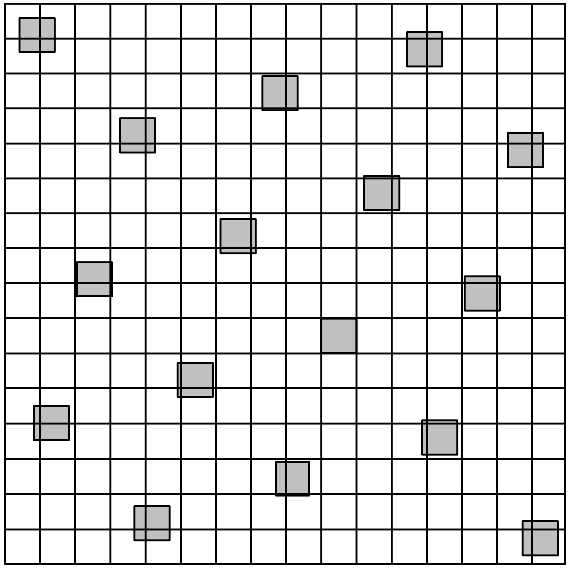
tile size: 4 x 4

# of memory banks: 16

# of tiles stored in one bank: 16

What Is "Uniform?"

- Ideally, regular hexagonal placement
- Actually, this cannot be achieved
  - Tiles must be on a grid of squares



buffer size: 64 x 64

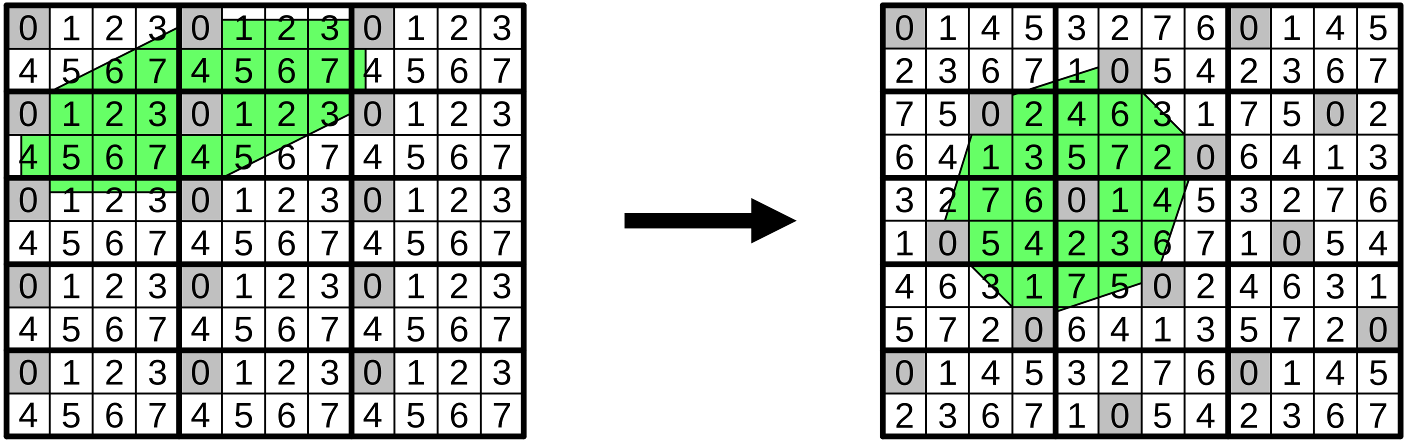
tile size: 4 x 4

# of memory banks: 16

# of tiles stored in one bank: 16

“Semi-regular” Hexagons

- **Make hexagons as regular as possible**
  - **Minimize the chance of frequent accesses to the same bank**

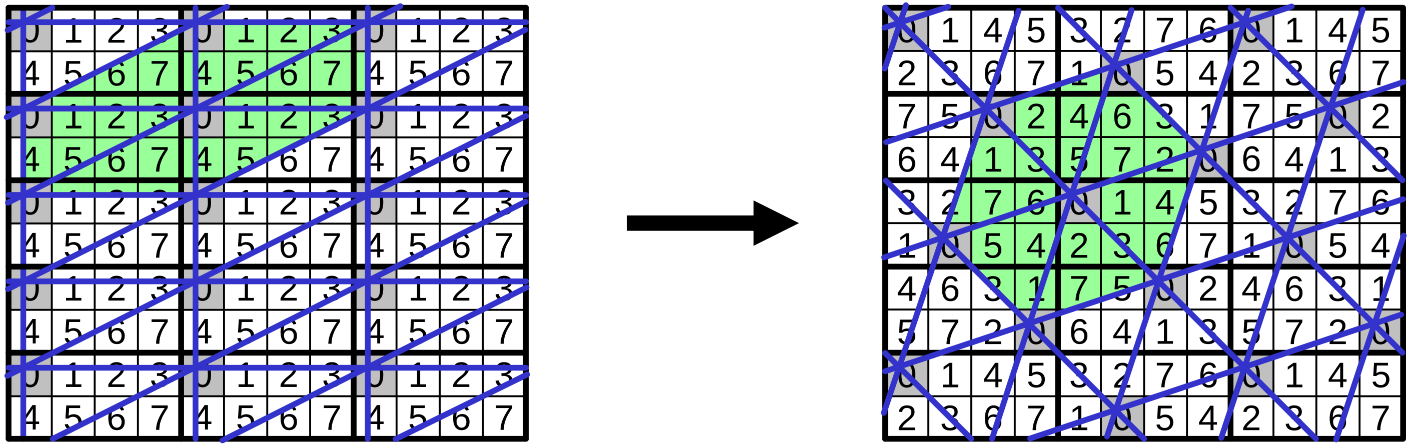


0	1	2	3	0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7	4	5	6	7
0	1	2	3	0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7	4	5	6	7
0	1	2	3	0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7	4	5	6	7
0	1	2	3	0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7	4	5	6	7
0	1	2	3	0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7	4	5	6	7

0	1	4	5	3	2	7	6	0	1	4	5
2	3	6	7	1	0	5	4	2	3	6	7
7	5	0	2	4	6	3	1	7	5	0	2
6	4	1	3	5	7	2	0	6	4	1	3
3	2	7	6	0	1	4	5	3	2	7	6
1	0	5	4	2	3	6	7	1	0	5	4
4	6	3	1	7	5	0	2	4	6	3	1
5	7	2	0	6	4	1	3	5	7	2	0
0	1	4	5	3	2	7	6	0	1	4	5
2	3	6	7	1	0	5	4	2	3	6	7

“Semi-regular” Hexagons

- Make hexagons as regular as possible
  - Minimize the chance of frequent accesses to the same bank
- Measure uniformity by the side lengths of the Delauney triangles



0	1	2	3	0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7	4	5	6	7
0	1	2	3	0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7	4	5	6	7
0	1	2	3	0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7	4	5	6	7
0	1	2	3	0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7	4	5	6	7
0	1	2	3	0	1	2	3	0	1	2	3
4	5	6	7	4	5	6	7	4	5	6	7

0	1	4	5	3	2	7	6	0	1	4	5
2	3	6	7	1	0	5	4	2	3	6	7
7	5	0	2	4	6	3	1	7	5	0	2
6	4	1	3	5	7	2	0	6	4	1	3
3	2	7	6	0	1	4	5	3	2	7	6
1	0	5	4	2	3	6	7	1	0	5	4
4	6	3	1	7	5	0	2	4	6	3	1
5	7	2	0	6	4	1	3	5	7	2	0
0	1	4	5	3	2	7	6	0	1	4	5
2	3	6	7	1	0	5	4	2	3	6	7

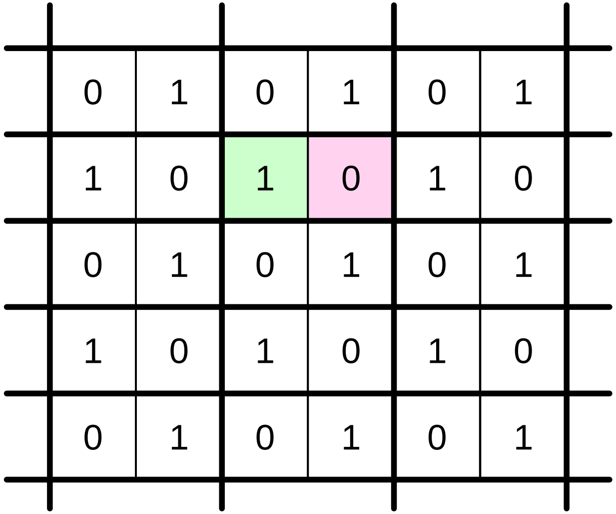
Pattern Search 1/2

- Unfortunately, brute-force
- Reduced search space
  - Avoids combinatorial explosion
  - Time complexity is reduced to O(logN)
  - No guarantee of optimality

Pattern Search 2/2

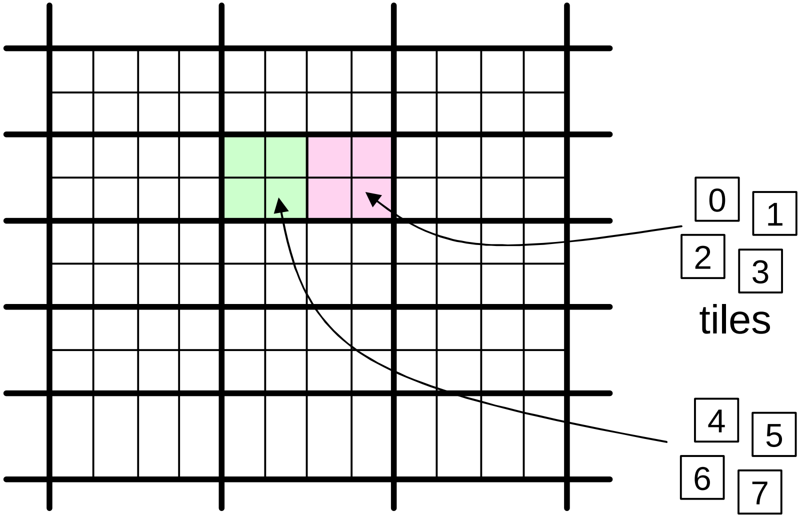
- **Search based on the solution for smaller N**

Based on the assignment for N = 2,



0	1	0	1	0	1
1	0	1	0	1	0
0	1	0	1	0	1
1	0	1	0	1	0
0	1	0	1	0	1

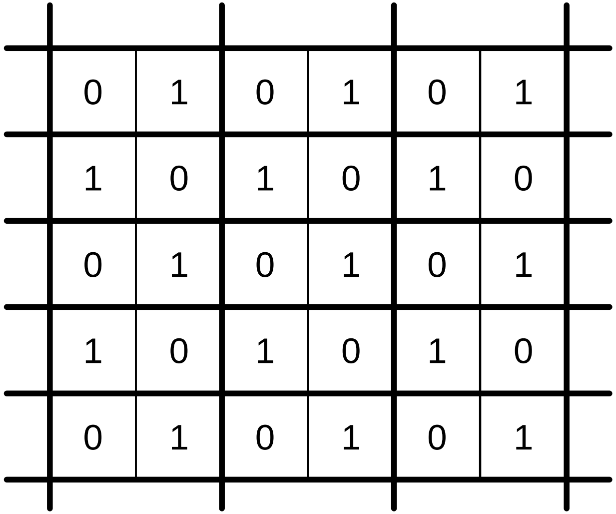
Consider assignments for N = 8



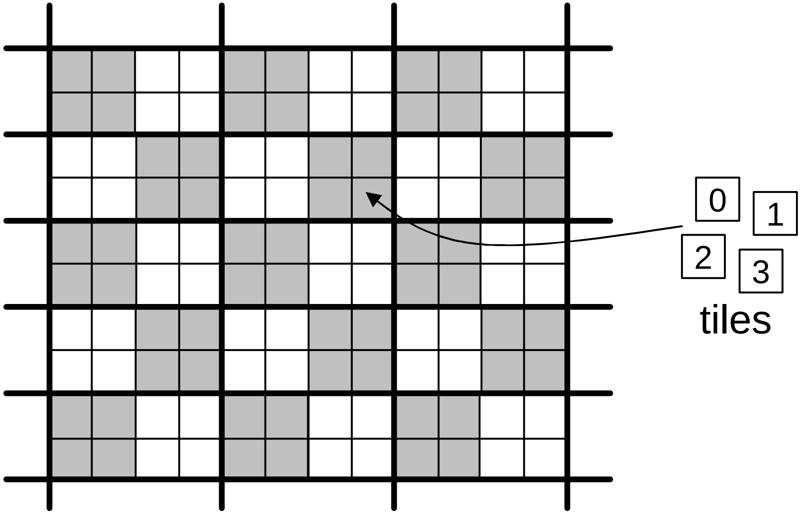
Pattern Search 2/2

- Search based on the solution for smaller N
- Force the same pattern for all the banks

The pattern of tiles 0 must be congruent to that of tiles 1, 2, or 3

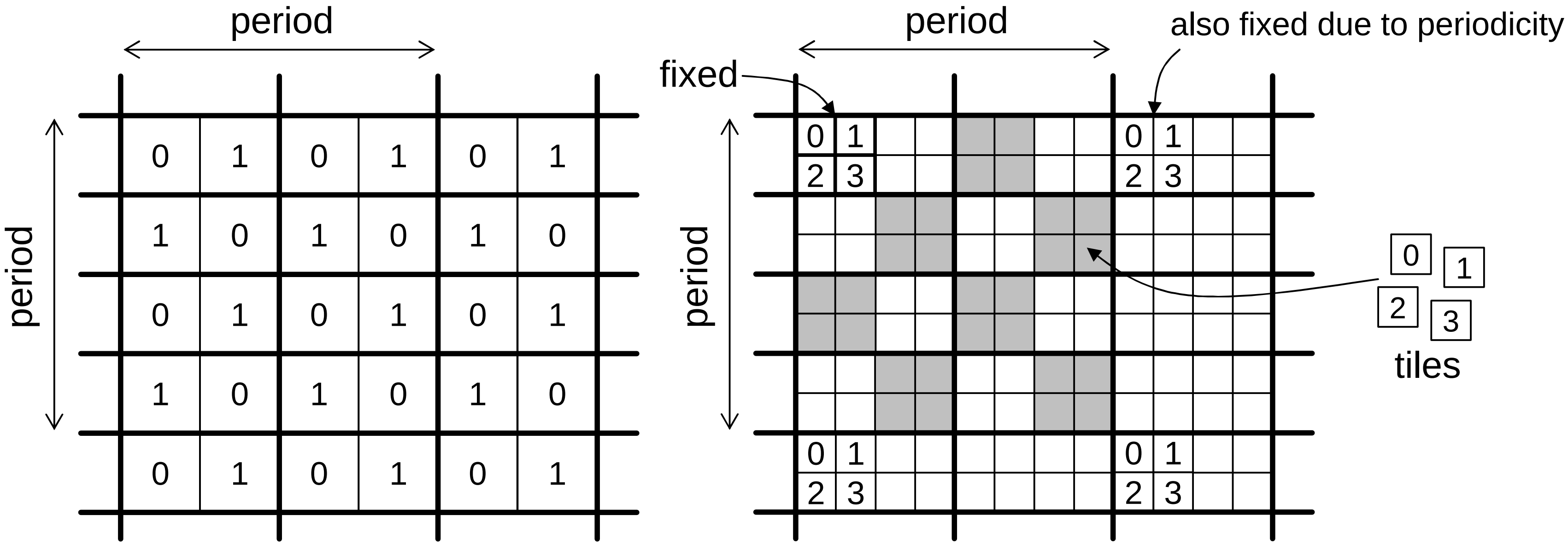


0	1	0	1	0	1
1	0	1	0	1	0
0	1	0	1	0	1
1	0	1	0	1	0
0	1	0	1	0	1

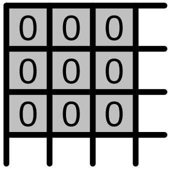


Pattern Search 2/2

- Search based on the solution for smaller N
- Force the same pattern for all the banks
- Assume periodicity

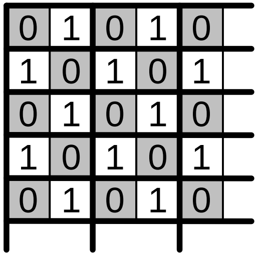


0	1	0	1	0	1
1	0	1	0	1	0
0	1	0	1	0	1
1	0	1	0	1	0
0	1	0	1	0	1



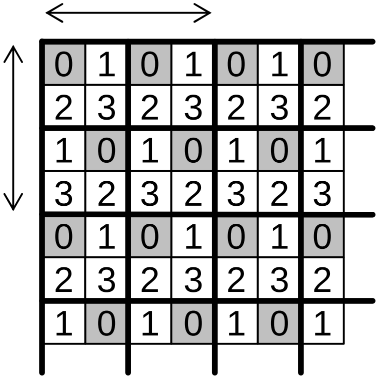
0	0	0
0	0	0
0	0	0

N = 1



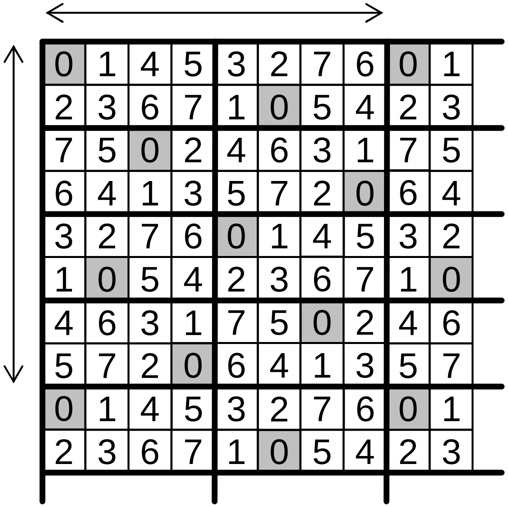
0	1	0	1	0
1	0	1	0	1
0	1	0	1	0
1	0	1	0	1
0	1	0	1	0

N = 2



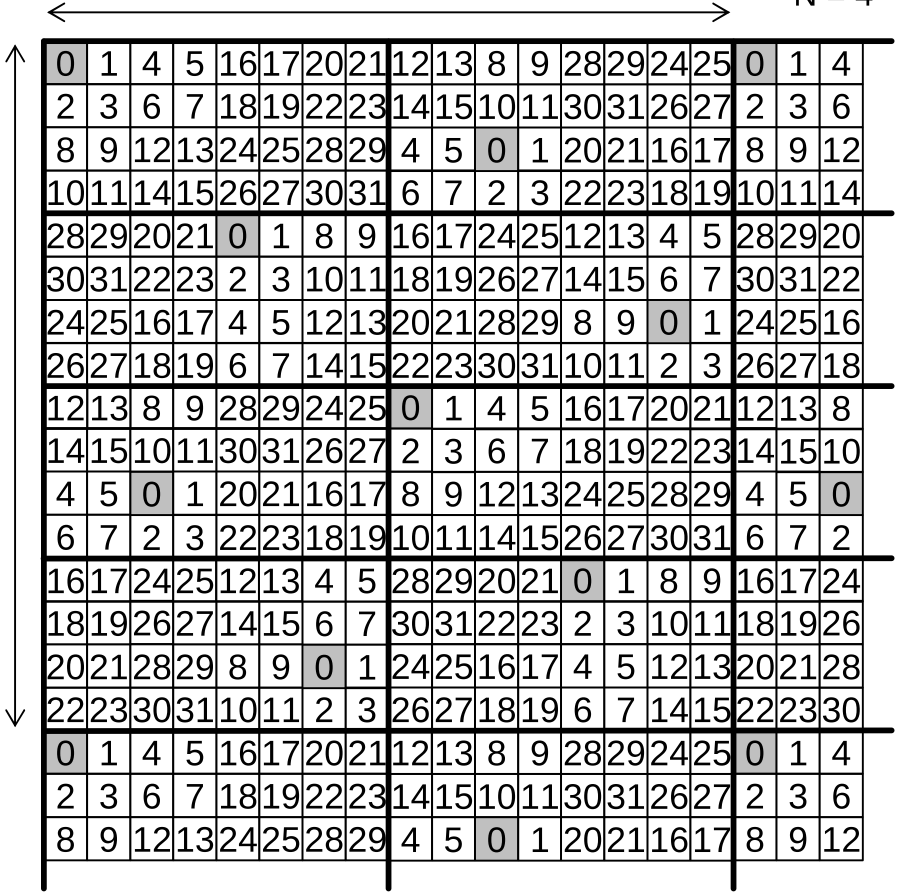
0	1	0	1	0	1	0
2	3	2	3	2	3	2
1	0	1	0	1	0	1
3	2	3	2	3	2	3
0	1	0	1	0	1	0
2	3	2	3	2	3	2
1	0	1	0	1	0	1

N = 4



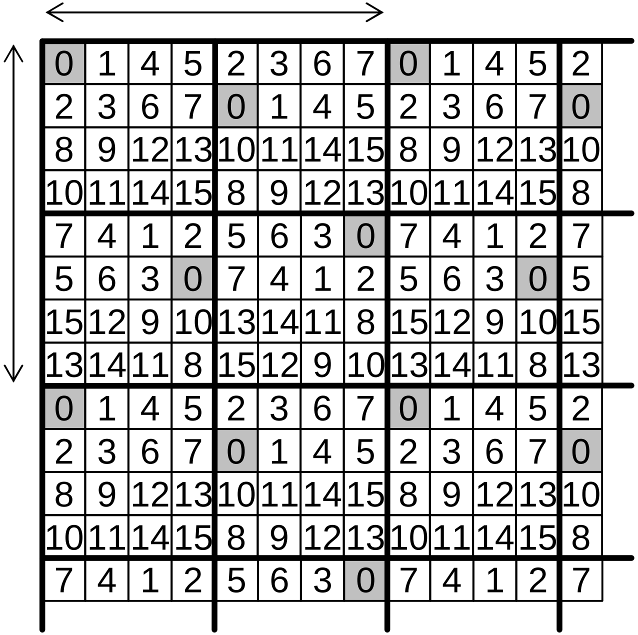
0	1	4	5	3	2	7	6	0	1
2	3	6	7	1	0	5	4	2	3
7	5	0	2	4	6	3	1	7	5
6	4	1	3	5	7	2	0	6	4
3	2	7	6	0	1	4	5	3	2
1	0	5	4	2	3	6	7	1	0
4	6	3	1	7	5	0	2	4	6
5	7	2	0	6	4	1	3	5	7
0	1	4	5	3	2	7	6	0	1
2	3	6	7	1	0	5	4	2	3

N = 8



0	1	4	5	16	17	20	21	12	13	8	9	28	29	24	25	0	1	4
2	3	6	7	18	19	22	23	14	15	10	11	30	31	26	27	2	3	6
8	9	12	13	24	25	28	29	4	5	0	1	20	21	16	17	8	9	12
10	11	14	15	26	27	30	31	6	7	2	3	22	23	18	19	10	11	14
28	29	20	21	0	1	8	9	16	17	24	25	12	13	4	5	28	29	20
30	31	22	23	2	3	10	11	18	19	26	27	14	15	6	7	30	31	22
24	25	16	17	4	5	12	13	20	21	28	29	8	9	0	1	24	25	16
26	27	18	19	6	7	14	15	22	23	30	31	10	11	2	3	26	27	18
12	13	8	9	28	29	24	25	0	1	4	5	16	17	20	21	12	13	8
14	15	10	11	30	31	26	27	2	3	6	7	18	19	22	23	14	15	10
4	5	0	1	20	21	16	17	8	9	12	13	24	25	28	29	4	5	0
6	7	2	3	22	23	18	19	10	11	14	15	26	27	30	31	6	7	2
16	17	24	25	12	13	4	5	28	29	20	21	0	1	8	9	16	17	24
18	19	26	27	14	15	6	7	30	31	22	23	2	3	10	11	18	19	26
20	21	28	29	8	9	0	1	24	25	16	17	4	5	12	13	20	21	28
22	23	30	31	10	11	2	3	26	27	18	19	6	7	14	15	22	23	30
0	1	4	5	16	17	20	21	12	13	8	9	28	29	24	25	0	1	4
2	3	6	7	18	19	22	23	14	15	10	11	30	31	26	27	2	3	6
8	9	12	13	24	25	28	29	4	5	0	1	20	21	16	17	8	9	12

N = 32



0	1	4	5	2	3	6	7	0	1	4	5	2
2	3	6	7	0	1	4	5	2	3	6	7	0
8	9	12	13	10	11	14	15	8	9	12	13	10
10	11	14	15	8	9	12	13	10	11	14	15	8
7	4	1	2	5	6	3	0	7	4	1	2	7
5	6	3	0	7	4	1	2	5	6	3	0	5
15	12	9	10	13	14	11	8	15	12	9	10	15
13	14	11	8	15	12	9	10	13	14	11	8	13
0	1	4	5	2	3	6	7	0	1	4	5	2
2	3	6	7	0	1	4	5	2	3	6	7	0
8	9	12	13	10	11	14	15	8	9	12	13	10
10	11	14	15	8	9	12	13	10	11	14	15	8
7	4	1	2	5	6	3	0	7	4	1	2	7

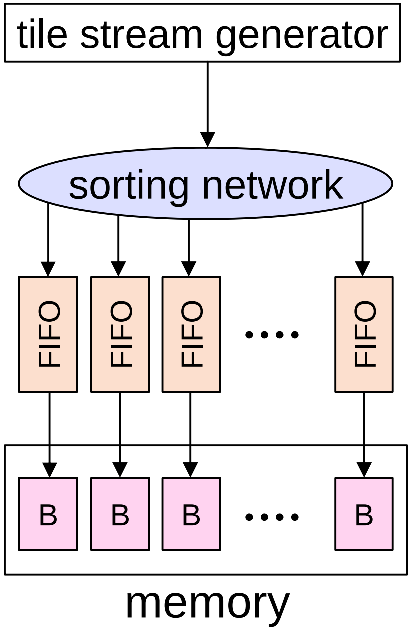
N = 16

Outline

- Background
- Related work
- Hexagonal storage scheme
- Evaluation
- Results
- Conclusion

TOSHIBA

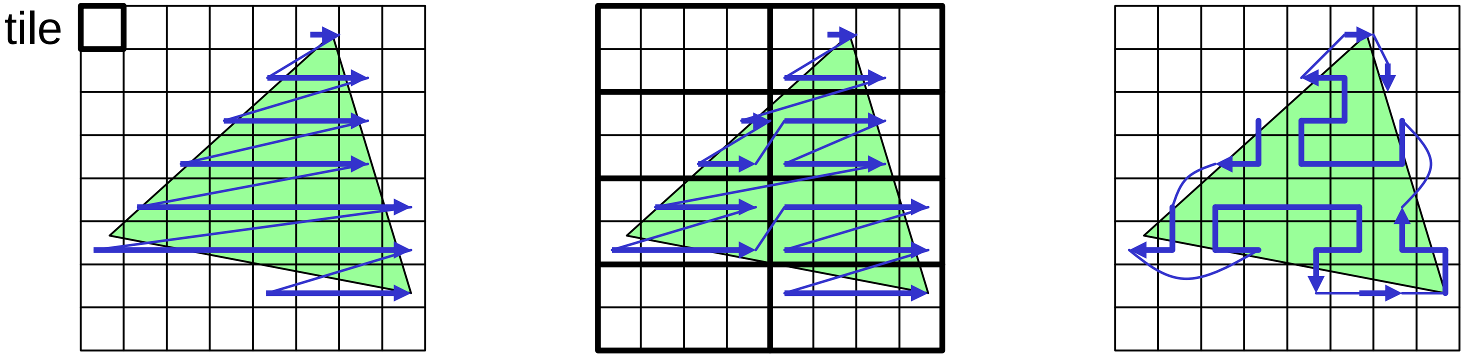
Evaluation Model



- Generic Model
  - Tile stream generator (TSG)
    - Outputs one tile per cycle
    - Is the rasterizer for frame buffers
    - Is the texture unit for textures
    - May have a cache
  - Each tile is sent to the appropriate bank
  - Each bank may have a FIFO
- TSG's parallelism is not considered
  - Only the tile order matters
  - We change this by rasterization order

Rasterization Order

- Row-major
- Blocked
- Hilbert
  - Rasterization granularity: tile



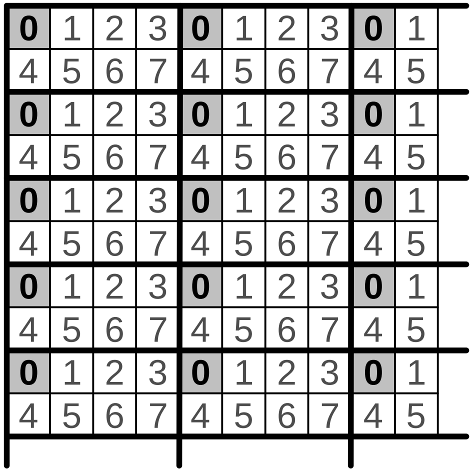
Row-major

Blocked

Hilbert

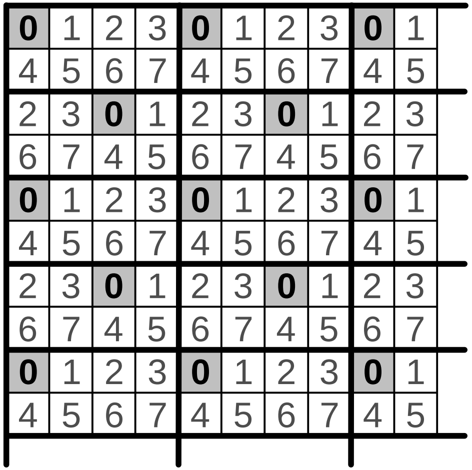
Counterpart Schemes

- Rectangular
- Flipped
- Multiaccess Frame Buffer (MFB)



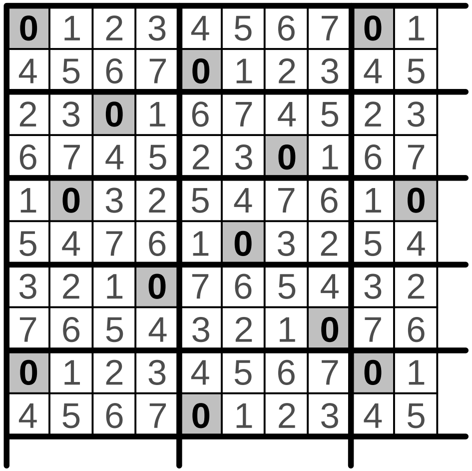
0	1	2	3	**0**	1	2	3	**0**	1
4	5	6	7	4	5	6	7	4	5
0	1	2	3	**0**	1	2	3	**0**	1
4	5	6	7	4	5	6	7	4	5
0	1	2	3	**0**	1	2	3	**0**	1
4	5	6	7	4	5	6	7	4	5
0	1	2	3	**0**	1	2	3	**0**	1
4	5	6	7	4	5	6	7	4	5
0	1	2	3	**0**	1	2	3	**0**	1
4	5	6	7	4	5	6	7	4	5

Rectangular



0	1	2	3	**0**	1	2	3	**0**	1
4	5	6	7	4	5	6	7	4	5
2	3	**0**	1	2	3	**0**	1	2	3
6	7	4	5	6	7	4	5	6	7
0	1	2	3	**0**	1	2	3	**0**	1
4	5	6	7	4	5	6	7	4	5
2	3	**0**	1	2	3	**0**	1	2	3
6	7	4	5	6	7	4	5	6	7
0	1	2	3	**0**	1	2	3	**0**	1
4	5	6	7	4	5	6	7	4	5

Flipped



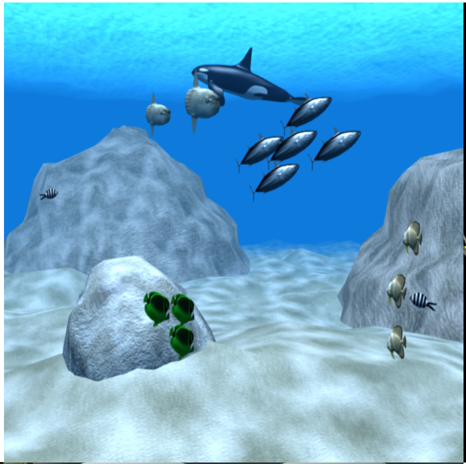
0	1	2	3	4	5	6	7	**0**	1
4	5	6	7	**0**	1	2	3	4	5
2	3	**0**	1	6	7	4	5	2	3
6	7	4	5	2	3	**0**	1	6	7
1	**0**	3	2	5	4	7	6	1	**0**
5	4	7	6	1	**0**	3	2	5	4
3	2	1	**0**	7	6	5	4	3	2
7	6	5	4	3	2	1	**0**	7	6
0	1	2	3	4	5	6	7	**0**	1
4	5	6	7	**0**	1	2	3	4	5

MFB

Test Scenes

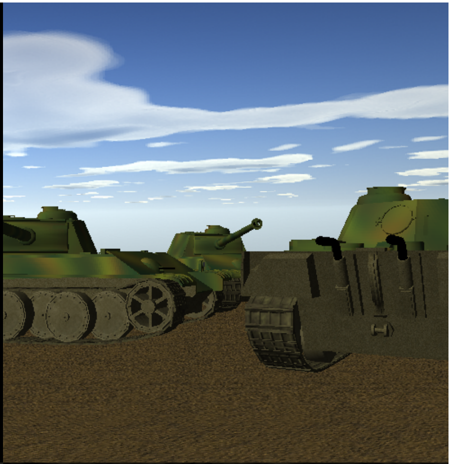
- **Frame buffer: 512 x 512**
- **Mipmaps**

Sea

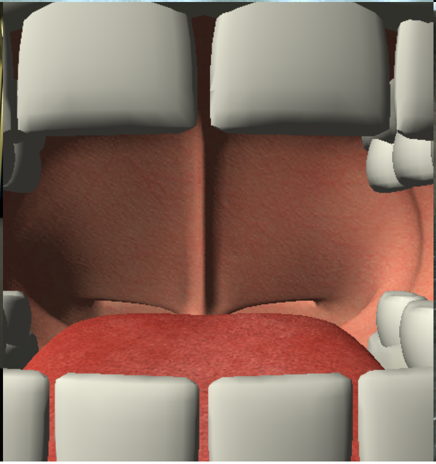


Stegosaurus

Tank

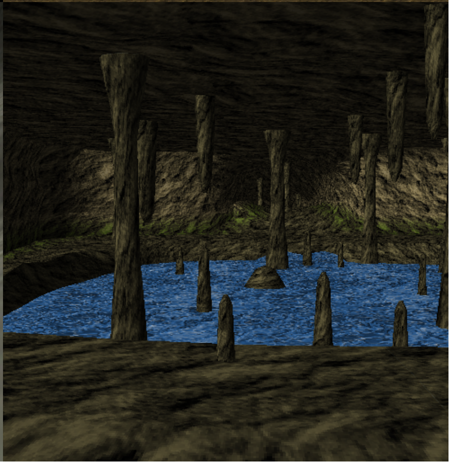


Station



Mouth

Waterfall



Cave

Outline

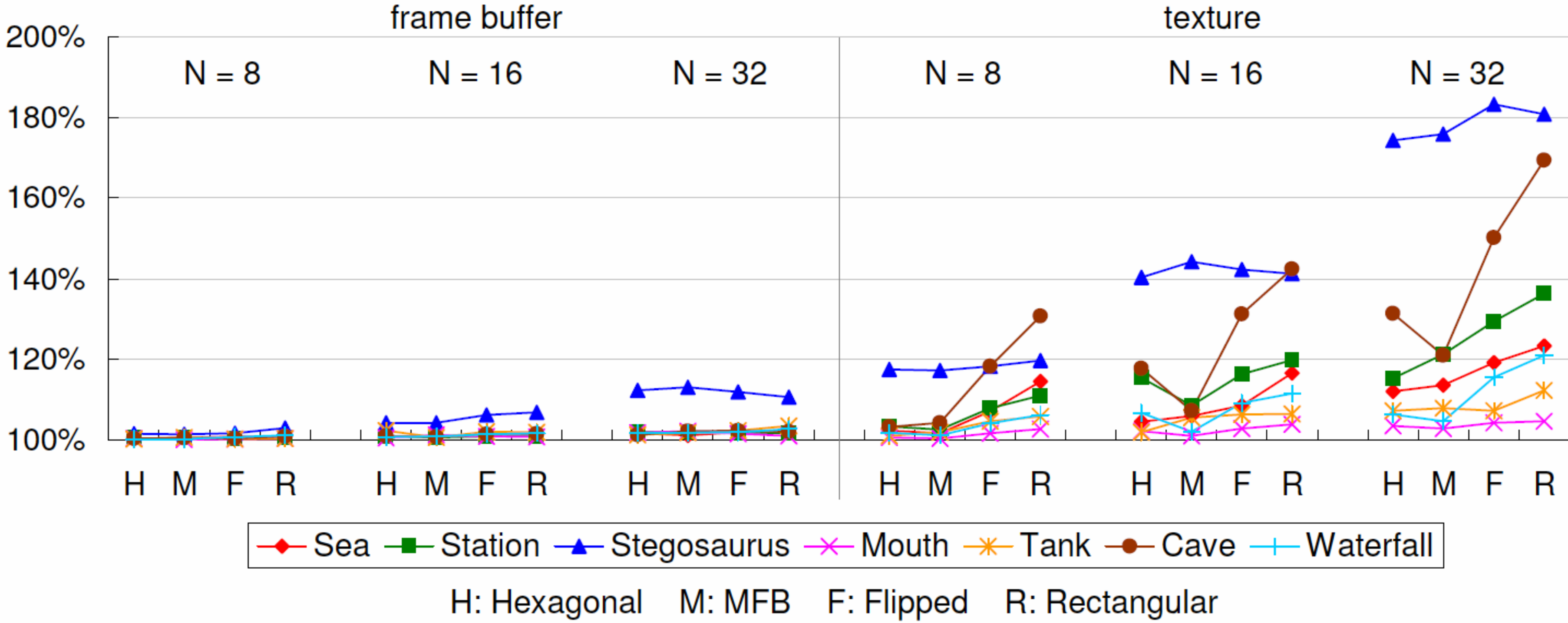
- Background
- Related work
- Hexagonal storage scheme
- Evaluation
- Results
- Conclusion

Results

- Simulated in C
  - N = 8, 16, 32
  - Tile size: 4x4

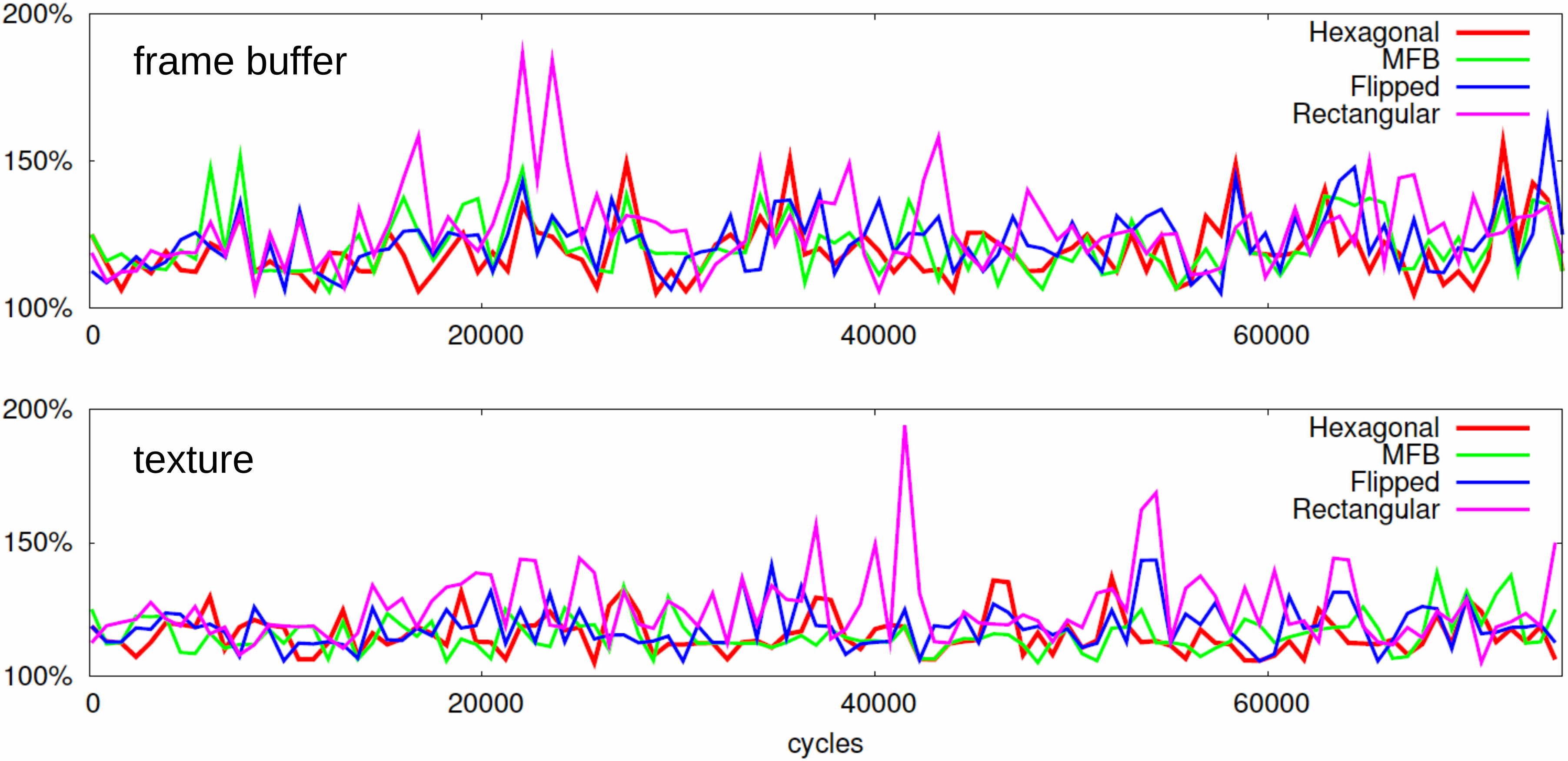
Overall Imbalance

- Max # of accesses to a bank / average
  - FB: Hex ~ MFB ~ Flip ~ Rect
  - Tex: Hex ~ MFB < Flip < Rect



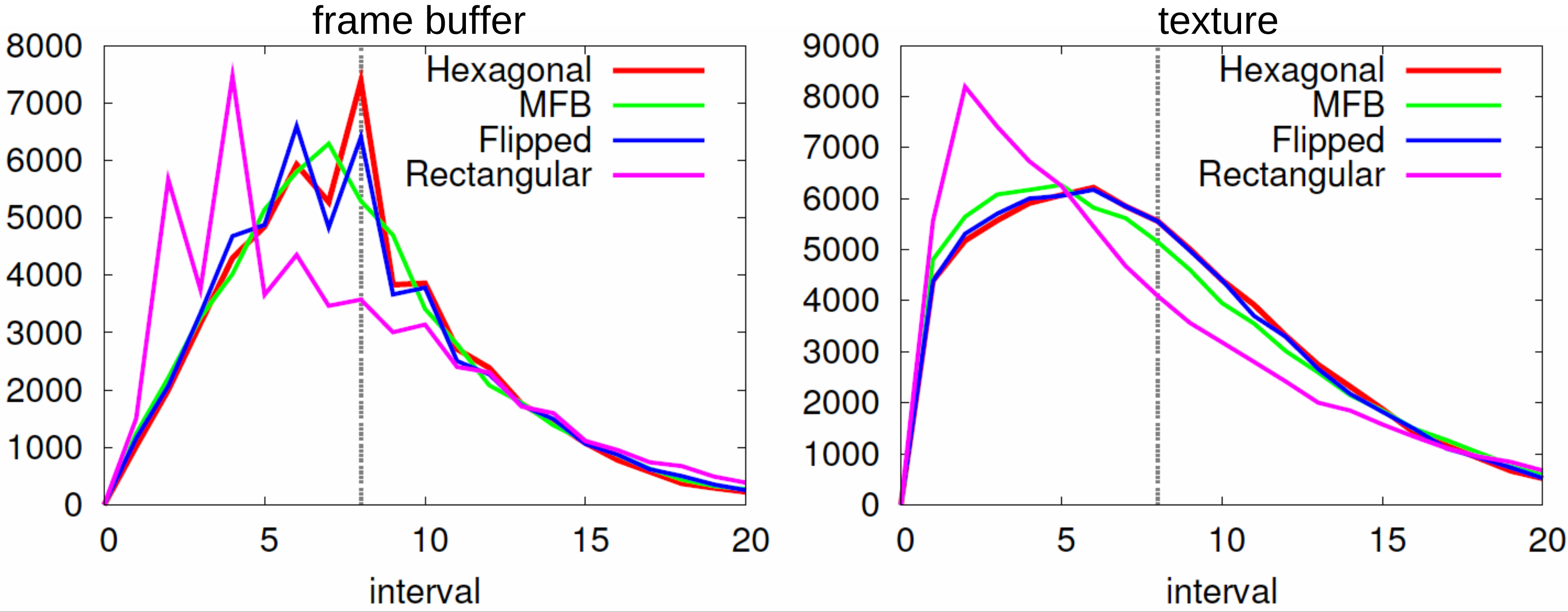
Temporal Imbalance

- **Time seq. with a window of 128 cycles**
  - **Rectangular has higher peaks**



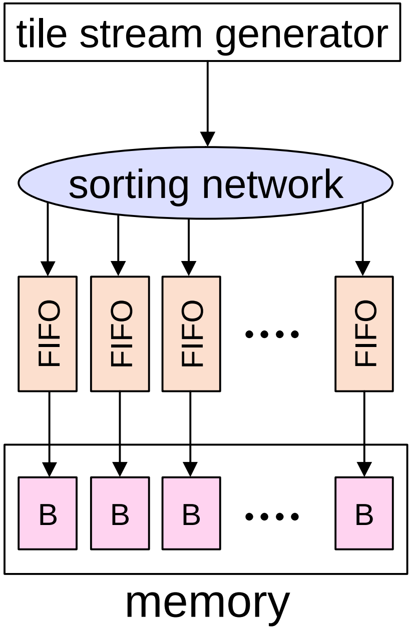
Tile Intervals

- Intervals between two tiles sent to a bank
  - Rectangular is strongly shifted to the left
  - MFB and Flipped are slightly shifted to the left



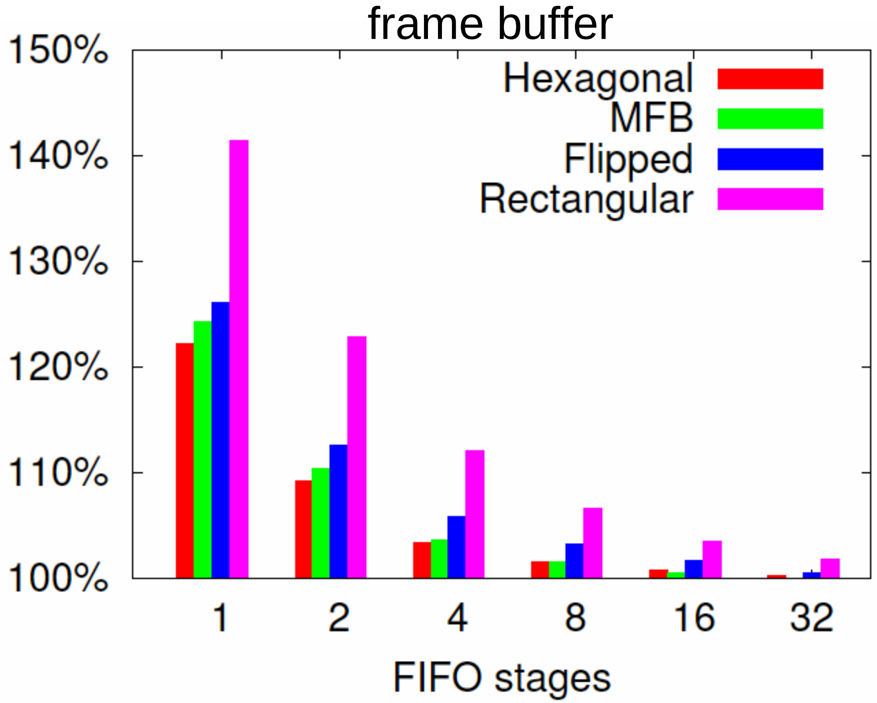
Performance Degradation

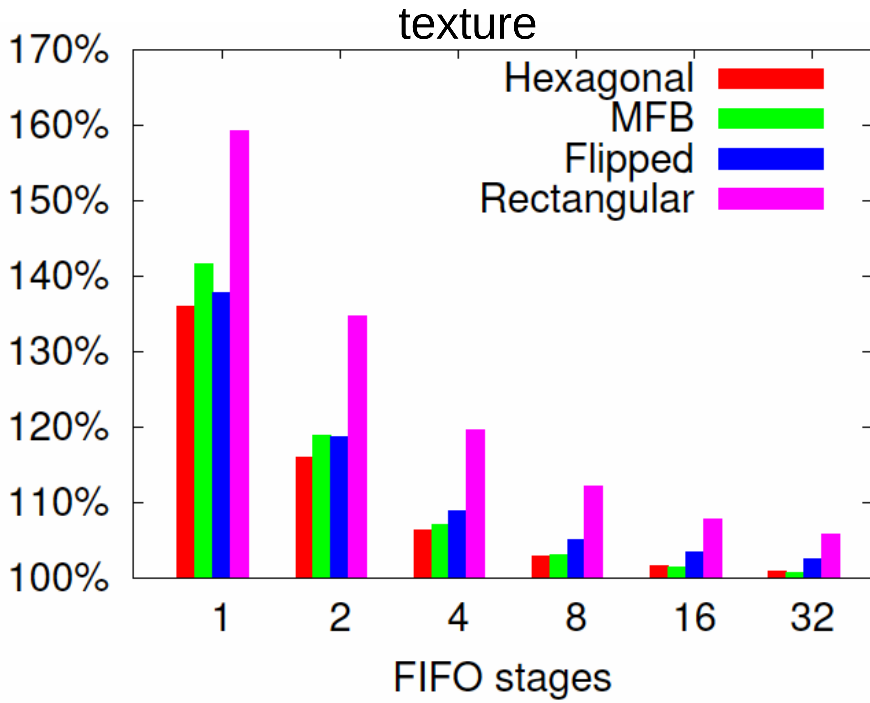
- Balanced system
  - TSG outputs one tile per cycle
  - Bank is busy for N cycles after receiving a tile
  - Tiles are queued in the FIFO of a busy bank
  - TSG stalls if the FIFO is full



Performance Degradation

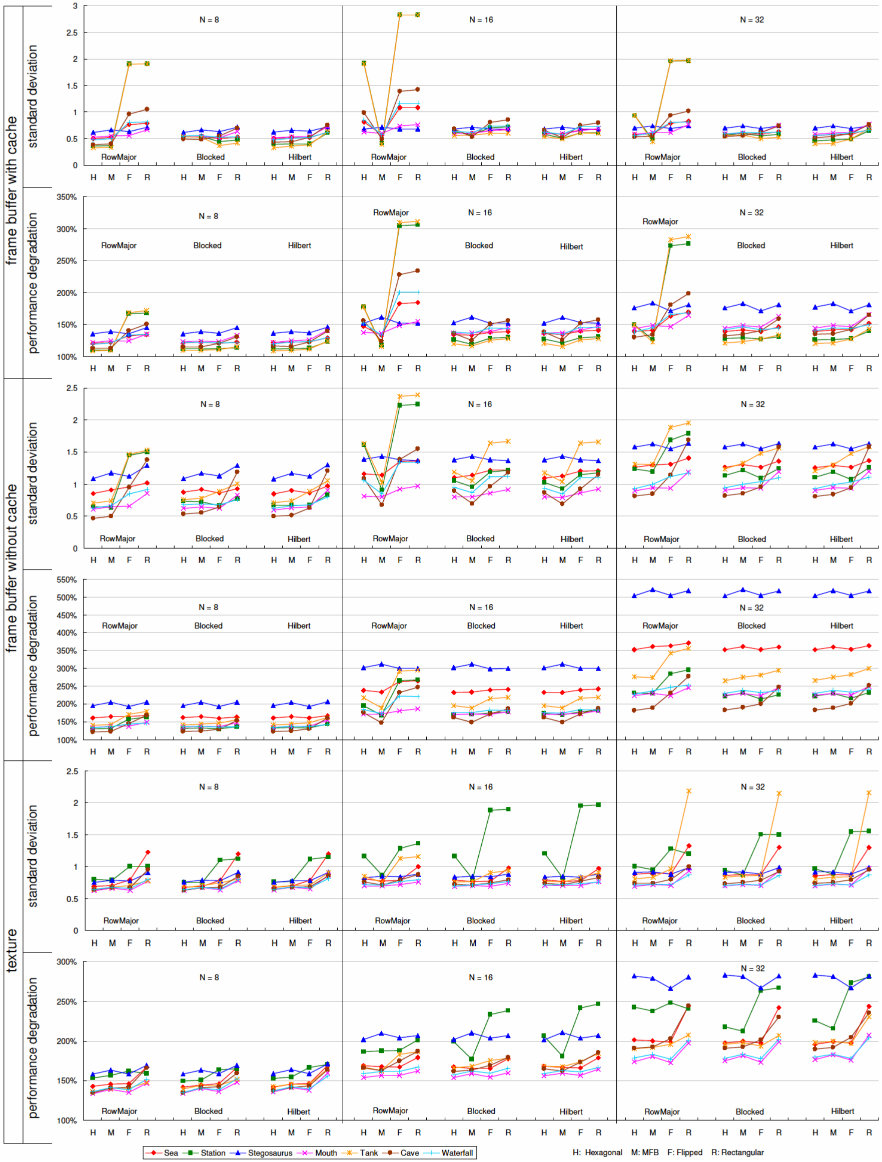
- **# of cycles to render scene / total # of tiles**
  - **Ratio between two schemes is preserved**
  - **Hexagonal has the least**


frame buffer


texture

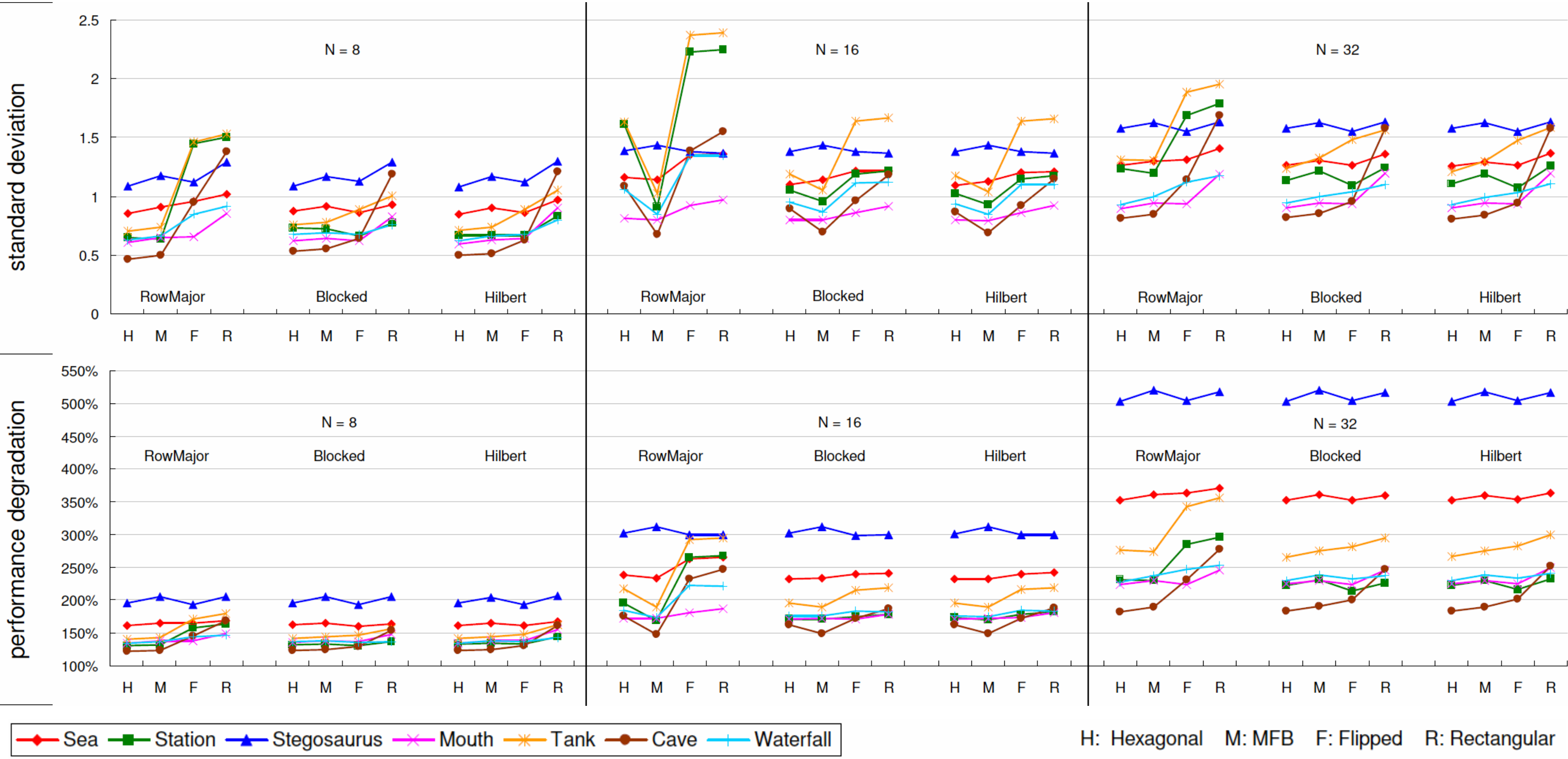
Summary Graphs

- For all combinations
  - N = 8, 16, 32
  - 3 rasterization orders
  - 4 storage schemes
  - 7 scenes
  - 3 types of buffers
    - FB with cache
    - FB without cache
    - Texture
  - 2 graphs
    - Deviation of tile intervals
    - Performance degradation



Summary Graphs

- **Overall tendency: slope upward**
  - **Major exception: Hex > MFB when N = 16**



Performance Gain

	Compared to	N = 8	N = 16	N = 32
Frame buffer	MFB	1.1%	-8.6%	0.2%
	Flipped	6.3%	10.4%	8.2%
	Rectangular	11.5%	11.7%	14.4%
Texture	MFB	3.0%	-0.3%	0.6%
	Flipped	3.1%	3.3%	1.6%
	Rectangular	11.2%	7.2%	11.9%

🟪 < 0%　　🟦 > 0%　　🟨 > 10%

Outline

- **Background**
- **Related work**
- **Hexagonal storage scheme**
- **Evaluation**
- **Conclusion**

Conclusion

- Hexagonal storage scheme
  - Evenly distributes memory accesses
  - Up to 10% performance gain on an average
    - Not so drastic as we expected…
    - At least supports the validity of our strategy
  - Minimal impact on silicon area
    - Only a few additional logical operations if any