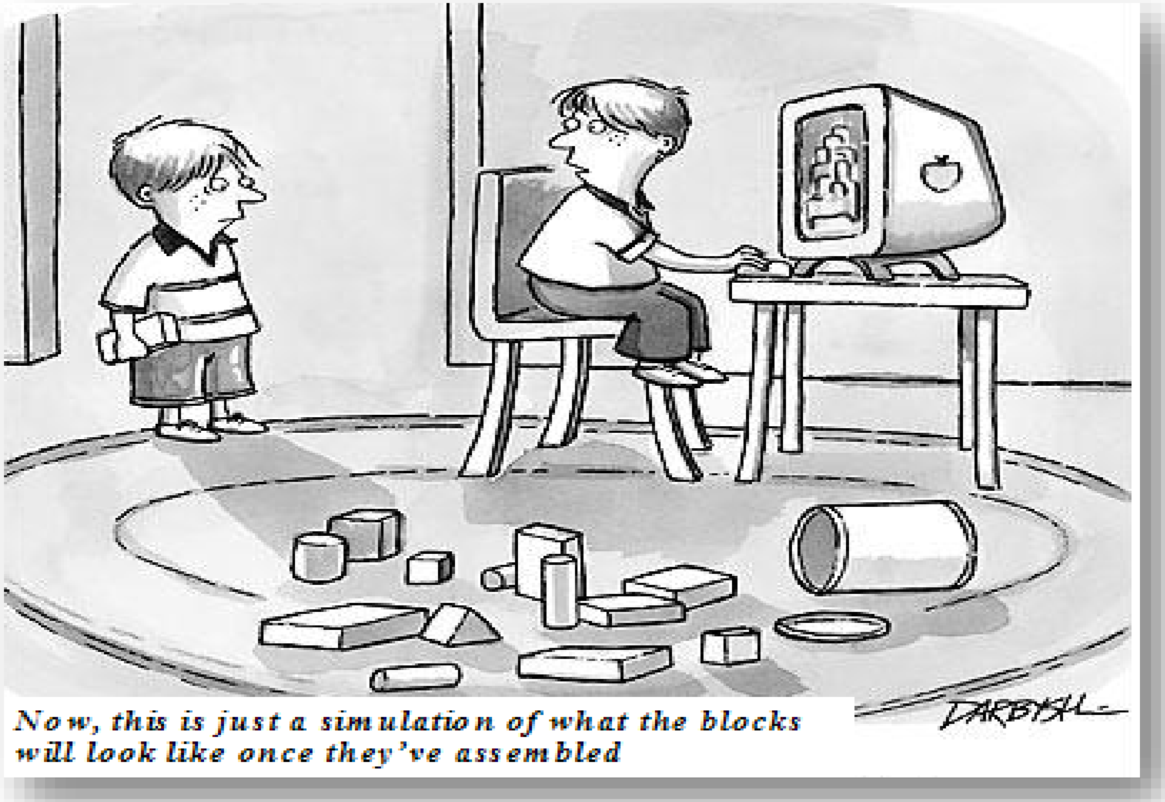
ECE569
Module 35



Now, this is just a simulation of what the blocks will look like once they've assembled

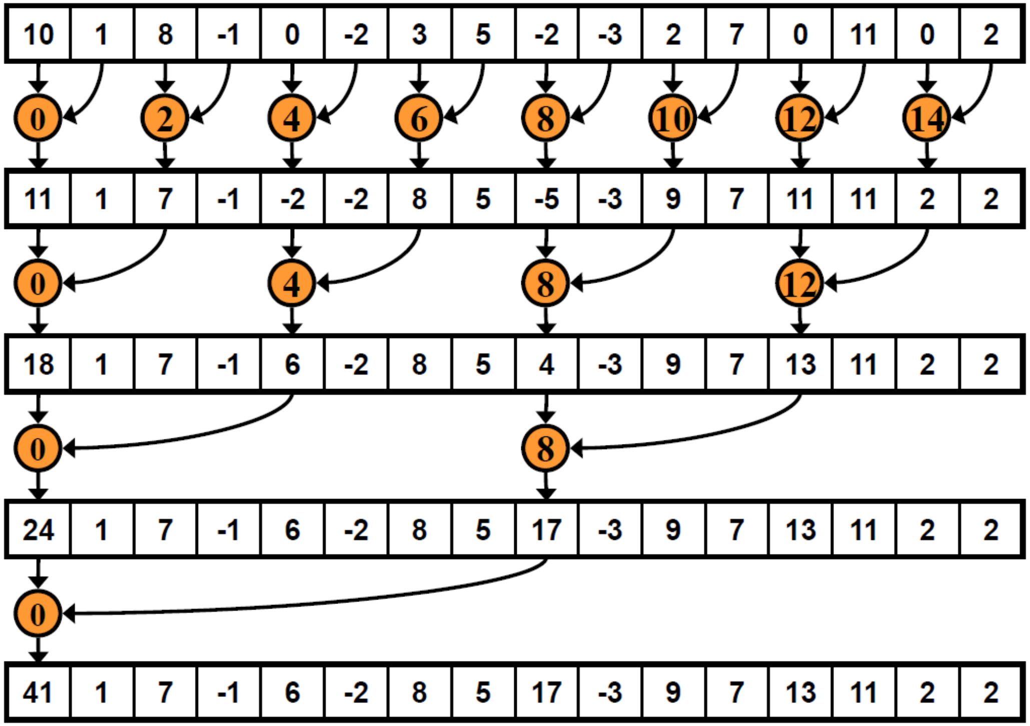
- Reduction – Stride Pattern – Shared Memory and Branch Divergence

Reduction - Tesla P100;compute v6.0;

Version	Time (ms)
serial	3.27400
global reduce stride – naïve	0.16450
shared stride reduce	0.15835

n: 1<<20

20.7X



10 1 8 -1 0 -2 3 5 -2 -3 2 7 0 11 0 2

11 1 7 -1 -2 -2 8 5 -5 -3 9 7 11 11 2 2

18 1 7 -1 6 -2 8 5 4 -3 9 7 13 11 2 2

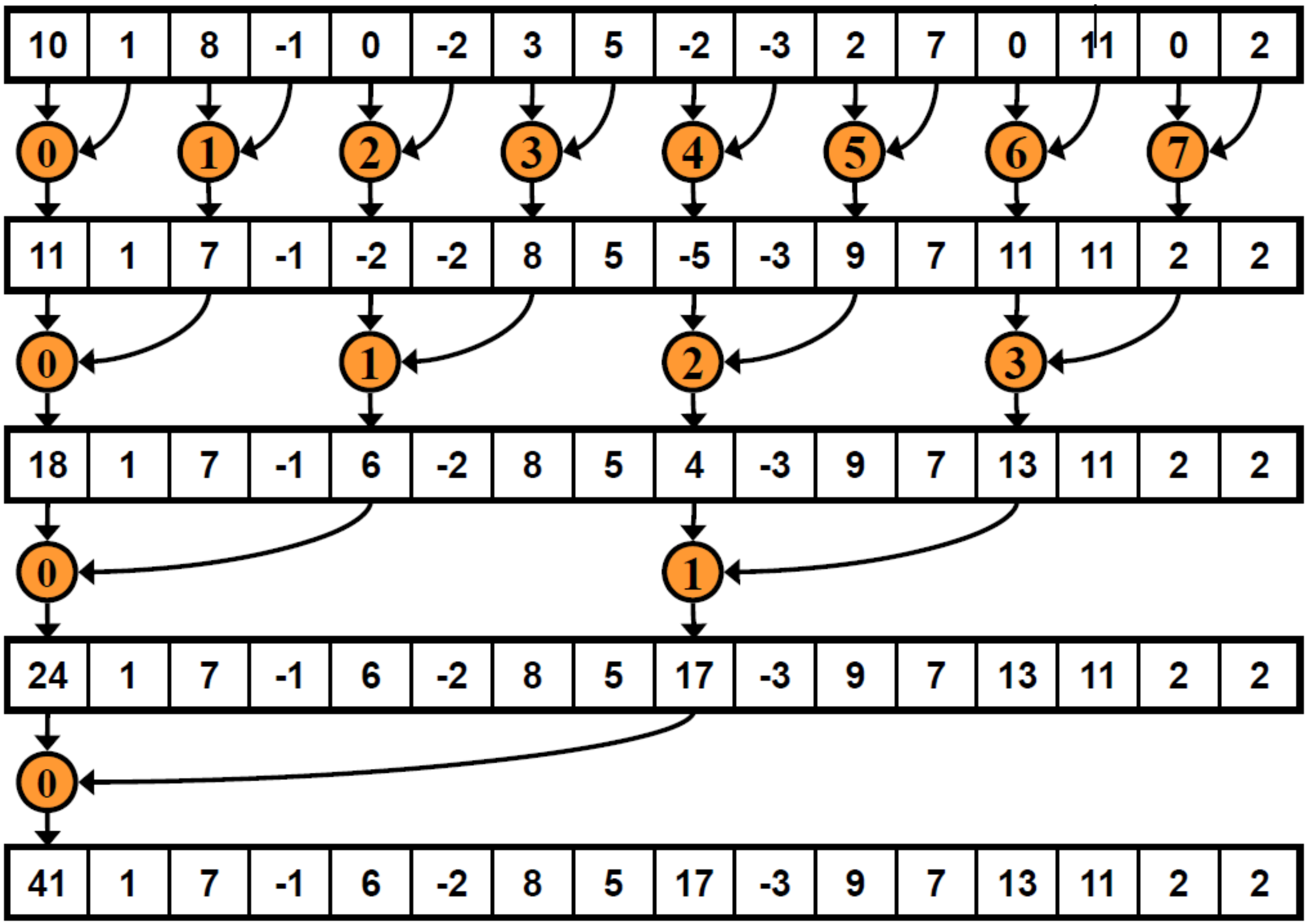
24 1 7 -1 6 -2 8 5 17 -3 9 7 13 11 2 2

41 1 7 -1 6 -2 8 5 17 -3 9 7 13 11 2 2

Unusual we should have had better performance!

How to reorganize workload assignment to avoid divergence?

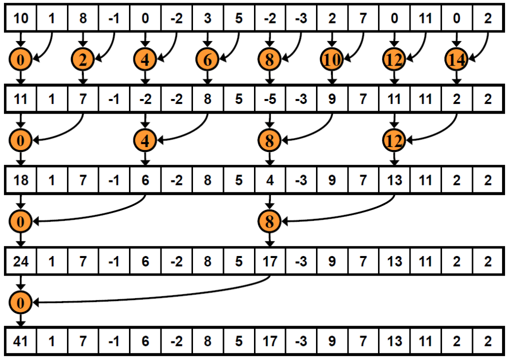
Kernel: Shared Memory – Stride Pattern



10	1	8	-1	0	-2	3	5	-2	-3	2	7	0	11	0	2
11	1	7	-1	-2	-2	8	5	-5	-3	9	7	11	11	2	2
18	1	7	-1	6	-2	8	5	4	-3	9	7	13	11	2	2
24	1	7	-1	6	-2	8	5	17	-3	9	7	13	11	2	2
41	1	7	-1	6	-2	8	5	17	-3	9	7	13	11	2	2

Kernel: Shared Memory – Stride Pattern

```
__global__ void shared_reduce_stride(float* d_out, float* d_in){
  extern __shared__ float sdata[];
  // shared_reduce<<<blocks,threads,threads*sizeof(float)>>>
  int myId = threadIdx.x + blockDim.x * blockIdx.x;
  int tid  = threadIdx.x;
  // load shared mem from global mem
  sdata[tid] = d_in[myId];
  // make sure entire block is loaded!
  // do reduction in shared memory
  for(int stride = 1; stride < blockDim.x; stride *= 2)  {
    __syncthreads();
    if(myId % (2*stride) == 0) {
      sdata[tid] += sdata[tid+stride]; }
   }
// thread 0 writes result for this block back to global mem
 if (tid == 0) {
        d_out[blockIdx.x] = sdata[tid]; }
}
```



10 1 8 -1 0 -2 3 5 -2 -3 2 7 0 11 0 2

11 1 7 -1 -2 -2 8 5 -5 -3 9 7 11 11 2 2

18 1 7 -1 6 -2 8 5 4 -3 9 7 13 11 2 2

24 1 7 -1 6 -2 8 5 17 -3 9 7 13 11 2 2

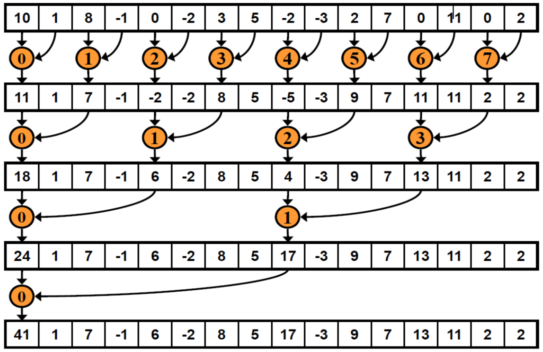
41 1 7 -1 6 -2 8 5 17 -3 9 7 13 11 2 2

Kernel: Shared Memory – No Diverge – Stride Pattern

```
__global__ void shared_reduce_stride_nodiverge (float* d_out,
float* d_in){
  extern __shared__ float sdata[];
  // shared_reduce<<<blocks,threads,threads*sizeof(float)>>>
  int myId = threadIdx.x + blockDim.x * blockIdx.x;
  int tid  = threadIdx.x;
  // load shared mem from global mem
  sdata[tid] = d_in[myId];
  // make sure entire block is loaded!
  // do reduction in shared memory
  for(int stride = 1; stride < blockDim.x; stride *= 2)  {
    __syncthreads();



 }
 // thread 0 writes result for this block back to global mem
 if (tid == 0) {
        d_out[blockIdx.x] = sdata[tid]; }
}
```



10 1 8 -1 0 -2 3 5 -2 -3 2 7 0 11 0 2

11 1 7 -1 -2 -2 8 5 -5 -3 9 7 11 11 2 2

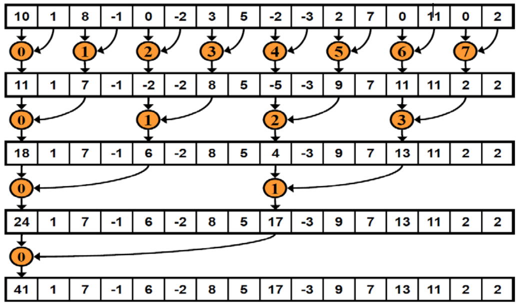
18 1 7 -1 6 -2 8 5 4 -3 9 7 13 11 2 2

24 1 7 -1 6 -2 8 5 17 -3 9 7 13 11 2 2

41 1 7 -1 6 -2 8 5 17 -3 9 7 13 11 2 2

Kernel: Shared Memory – No Diverge – Stride Pattern

```
__global__ void shared_reduce_stride_nodiverge (float* d_out,
float* d_in){
  extern __shared__ float sdata[];
  // shared_reduce<<<blocks,threads,threads*sizeof(float)>>>
  int myId = threadIdx.x + blockDim.x * blockIdx.x;
  int tid  = threadIdx.x;
  // load shared mem from global mem
  sdata[tid] = d_in[myId];
  // make sure entire block is loaded!
  // do reduction in shared memory
  for(int stride = 1; stride < blockDim.x; stride *= 2)  {
    __syncthreads();
    int index = 2*stride*tid;
    if( index < blockDim.x ) {
      sdata[index] += sdata[stride+index]; }
   }
// thread 0 writes result for this block back to global mem
 if (tid == 0) {
        d_out[blockIdx.x] = sdata[tid]; }
}
```



10 | 1 | 8 | -1 | 0 | -2 | 3 | 5 | -2 | -3 | 2 | 7 | 0 | 11 | 0 | 2

11 | 1 | 7 | -1 | -2 | -2 | 8 | 5 | -5 | -3 | 9 | 7 | 11 | 11 | 2 | 2

18 | 1 | 7 | -1 | 6 | -2 | 8 | 5 | 4 | -3 | 9 | 7 | 13 | 11 | 2 | 2

24 | 1 | 7 | -1 | 6 | -2 | 8 | 5 | 17 | -3 | 9 | 7 | 13 | 11 | 2 | 2

41 | 1 | 7 | -1 | 6 | -2 | 8 | 5 | 17 | -3 | 9 | 7 | 13 | 11 | 2 | 2

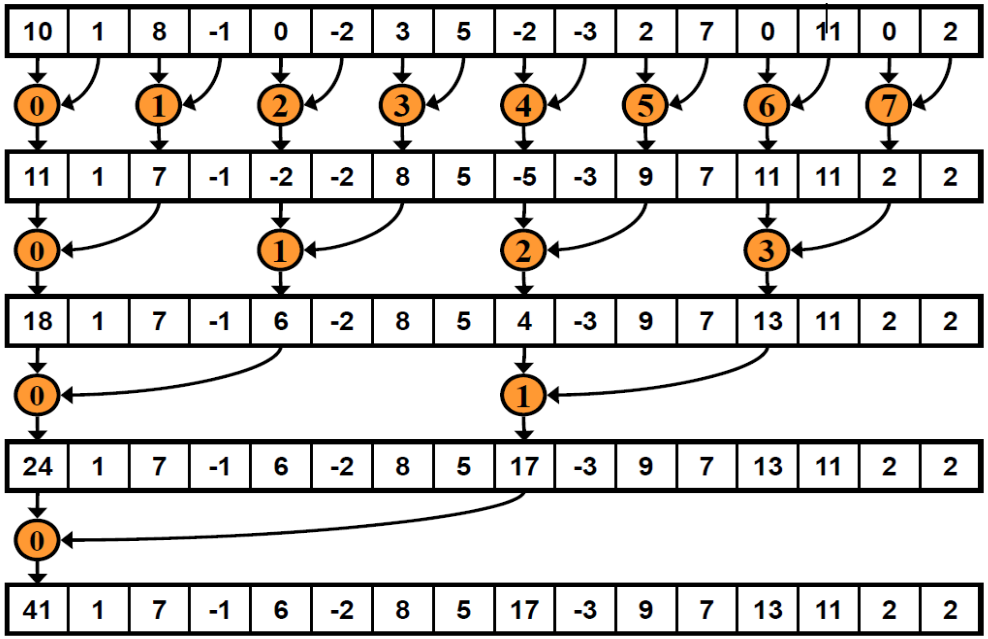
Bonus: No more expensive % operator

Reduction - Tesla P100;compute v6.0;

Version	Time (ms)
serial	3.27400
global reduce stride – naïve	0.16450
shared stride reduce	0.15835
shared_reduce_stride_nodiverge	0.09081

n: 1<<20

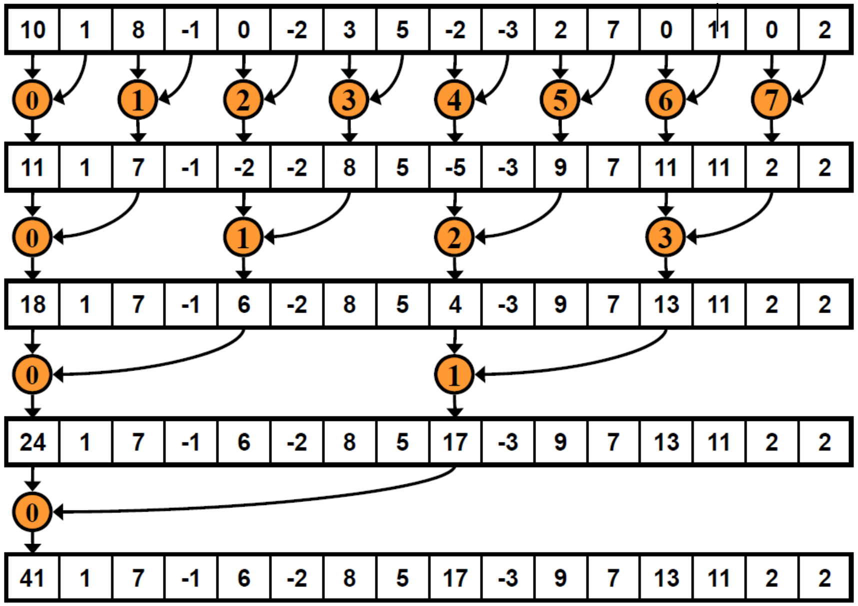
36.1X



1.7X!

Observation on Stride Pattern-No divergence

- Divergence free
- New problem:
  - Shared memory bank conflicts
  - Will come back to reduction, first bank conflicts!



10 1 8 -1 0 -2 3 5 -2 -3 2 7 0 11 0 2

11 1 7 -1 -2 -2 8 5 -5 -3 9 7 11 11 2 2

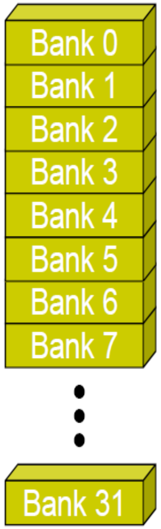
18 1 7 -1 6 -2 8 5 4 -3 9 7 13 11 2 2

24 1 7 -1 6 -2 8 5 17 -3 9 7 13 11 2 2

41 1 7 -1 6 -2 8 5 17 -3 9 7 13 11 2 2

Shared Memory and Bank Conflicts

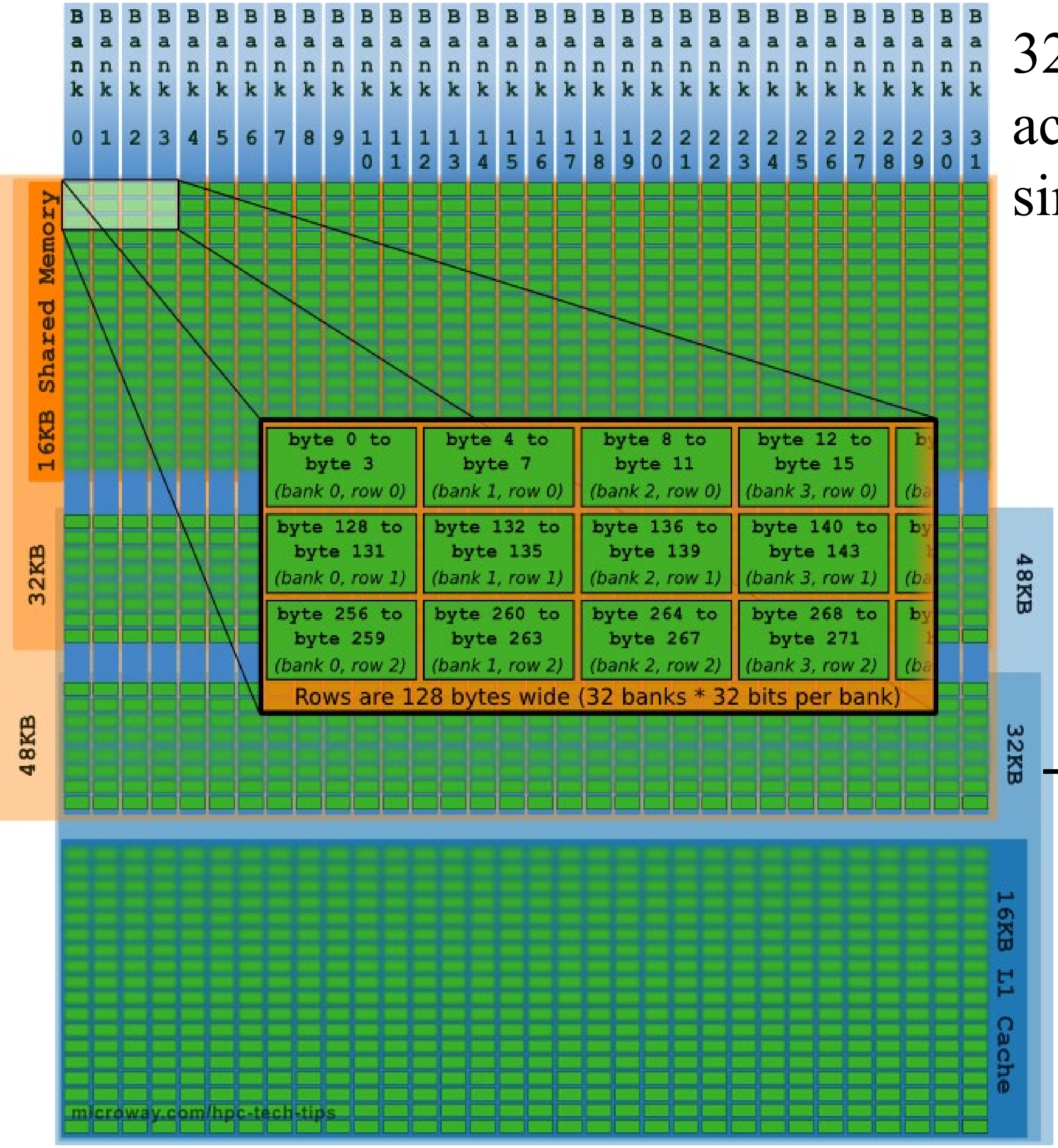
- Many threads access memory at the same time
  - To service more than one thread, memory is divided into independent banks
  - This layout essential to achieve high bandwidth
- Each SM has Shared Memory organized in 32 Memory banks



Shared Memory Architecture

- The 32 banks of the Shared Memory are organized like benches in a movie theater
  - You have multiple rows of benches
  - Each row has 32 benches which are separated and grouped in long columns
  - In each bench you can "seat" a family of four bytes (32bits total)
  - Note that a bank represents a column of benches in the movie theater, which is perpendicular to the screen

Organization of Memory Banks



byte 0 to byte 3 (bank 0, row 0)	byte 4 to byte 7 (bank 1, row 0)	byte 8 to byte 11 (bank 2, row 0)	byte 12 to byte 15 (bank 3, row 0)	by (ba
byte 128 to byte 131 (bank 0, row 1)	byte 132 to byte 135 (bank 1, row 1)	byte 136 to byte 139 (bank 2, row 1)	byte 140 to byte 143 (bank 3, row 1)	by (ba
byte 256 to byte 259 (bank 0, row 2)	byte 260 to byte 263 (bank 1, row 2)	byte 264 to byte 267 (bank 2, row 2)	byte 268 to byte 271 (bank 3, row 2)	by (ba

Rows are 128 bytes wide (32 banks * 32 bits per bank)

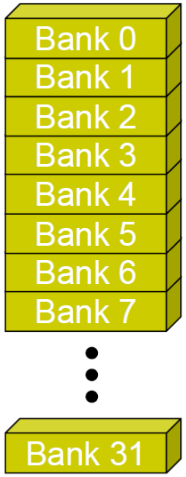
32 threads in a warp attempt to access shared memory simultaneously

Bank conflict: the scenario where two different threads access *different* words in the same bank

-

-No “bank conflicts” between threads of different warps

Bank Conflict Example

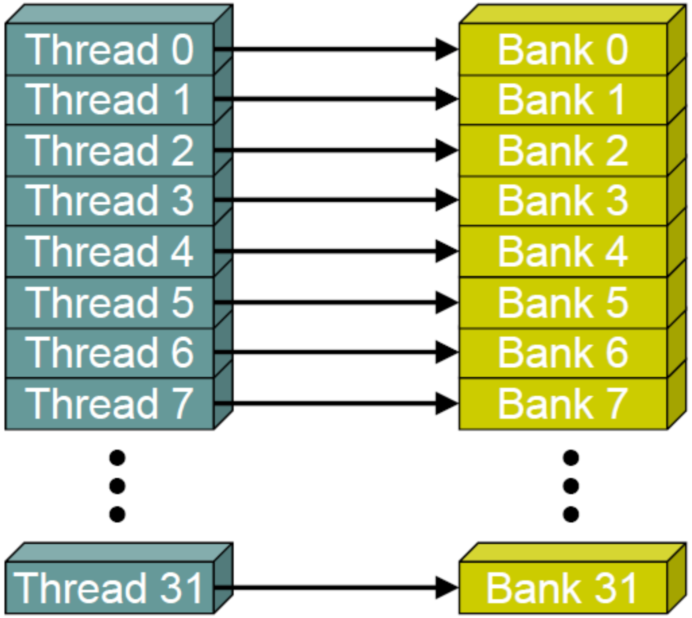


- Bank = (address of offset) % 32
- Example:
- 1D shared mem array, myShMemVar, of 1024 floats
  - myShMemVar[4]: accesses bank #4 (the fifth one – first row)
  - myShMemVar[31]: accesses bank #31 (the last one – first row)
  - **myShMemVar[50]:** access bank #18 (the 19th one – second row)
  - myShMemVar[128]: access bank #0 (the first one – fifth row)
  - **myShMemVar[178]:** access bank #18 (the 19th one – sixth row)
- NOTE: If, for instance, the third thread in a warp accesses myShMemVar[50] and the eight thread in the warp access myShMemVar[178], then you have a two-way bank conflict and the two transactions get serialized

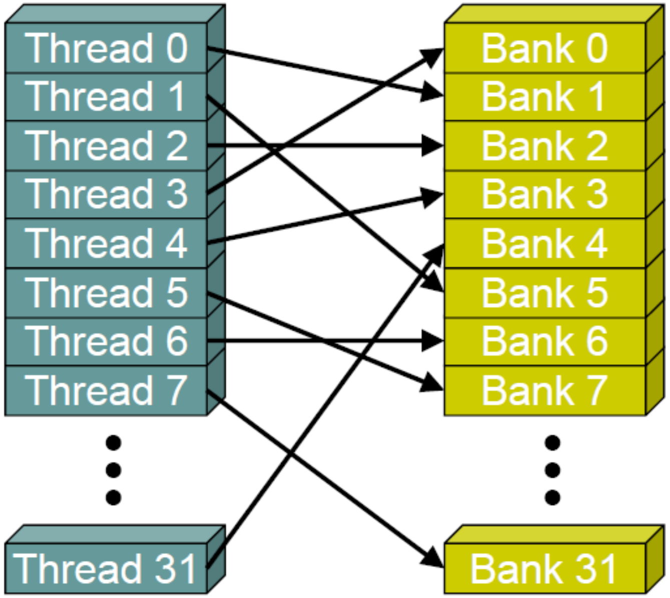
Bank Conflict Examples(1): 4 Byte Words

Bank = (address of offset) % 32

- No Bank Conflicts
  - Linear addressing stride == 1



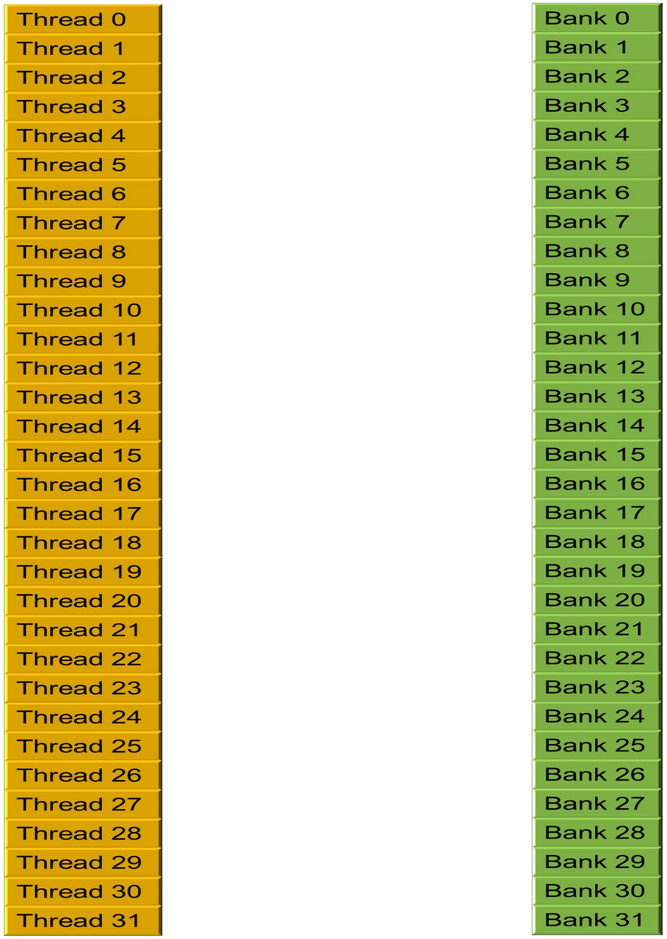
- No Bank Conflicts
  - Random 1:1 Permutation



Bank Conflict Examples(2): 4 Byte Words

Bank = (address of offset) % 32

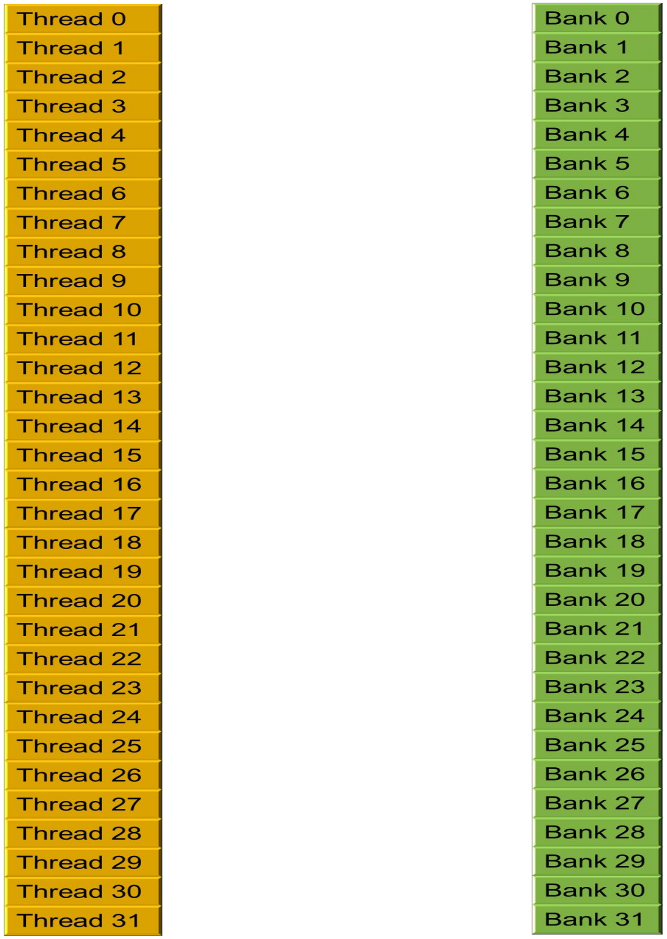
- sdata[threadIdx.x*2]++;



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

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

- sdata[threadIdx.x*8]++;



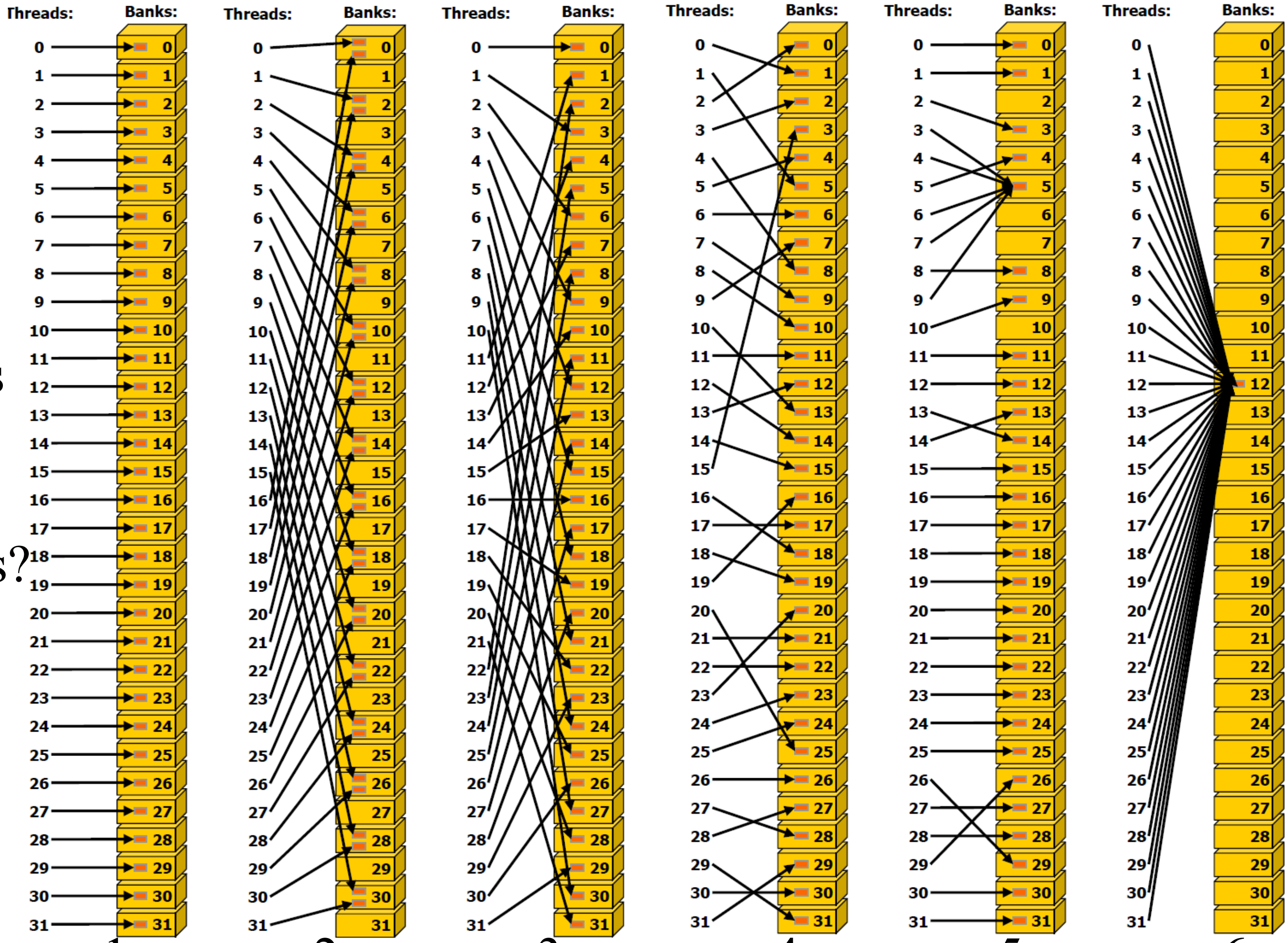
Thread 0
Thread 1
Thread 2
Thread 3
Thread 4
Thread 5
Thread 6
Thread 7
Thread 8
Thread 9
Thread 10
Thread 11
Thread 12
Thread 13
Thread 14
Thread 15
Thread 16
Thread 17
Thread 18
Thread 19
Thread 20
Thread 21
Thread 22
Thread 23
Thread 24
Thread 25
Thread 26
Thread 27
Thread 28
Thread 29
Thread 30
Thread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

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

Which stride amount results with bank conflicts for all threads in a warp?

Bank Conflict Examples(3): 4 Byte Words

Which accesses have bank conflicts?



1 2 3 4 5 6

Bank Conflicts – Demo

```
__global__ void mykernel1(unsigned long long* time){
__shared__ float shared[1024];
// clock returns clock ticks  unsigned long long
startTime = clock();
//all threads accessing the same location (broadcast)
shared[0]++;
unsigned long long finishTime = clock();  *time =
(finishTime-startTime);
}
```

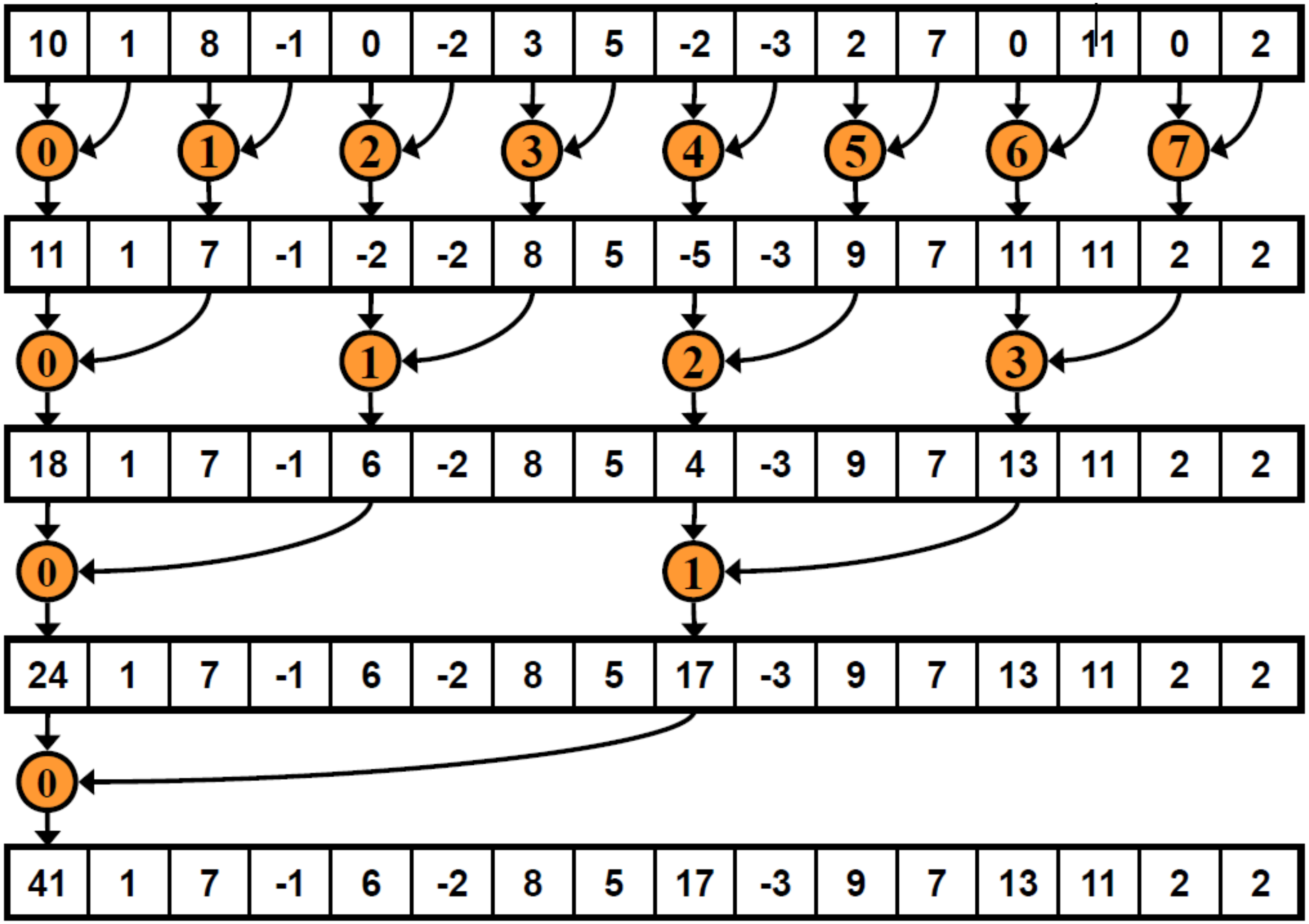
Replace **shared[0]++ with**

```
Kernel 2: shared[threadIdx.x]++;
Kernel 3: shared[threadIdx.x*4]++;
Kernel 4: shared[threadIdx.x*8]++;
Kernel 5: shared[threadIdx.x*32]++;
```

Refer to D2L→Demo→8.BankConflict

Kernel: Shared Memory – Stride Pattern

Next: How to eliminate bank conflicts?



10	1	8	-1	0	-2	3	5	-2	-3	2	7	0	11	0	2
11	1	7	-1	-2	-2	8	5	-5	-3	9	7	11	11	2	2
18	1	7	-1	6	-2	8	5	4	-3	9	7	13	11	2	2
24	1	7	-1	6	-2	8	5	17	-3	9	7	13	11	2	2
41	1	7	-1	6	-2	8	5	17	-3	9	7	13	11	2	2