Parallel Computing with GPUs

Shared Memory
Part 2 - Bank Conflicts

Dr Paul Richmond
http://paulrichmond.shef.ac.uk/teaching/COM4521/

This Lecture (learning objectives)

- ❑Shared Memory Bank Conflicts
  - ❑Explain the concepts of memory banks and how conflicts can occur
  - ❑Give examples of access strides and the impact on bank conflicts
  - ❑Present solutions for avoiding bank conflicts

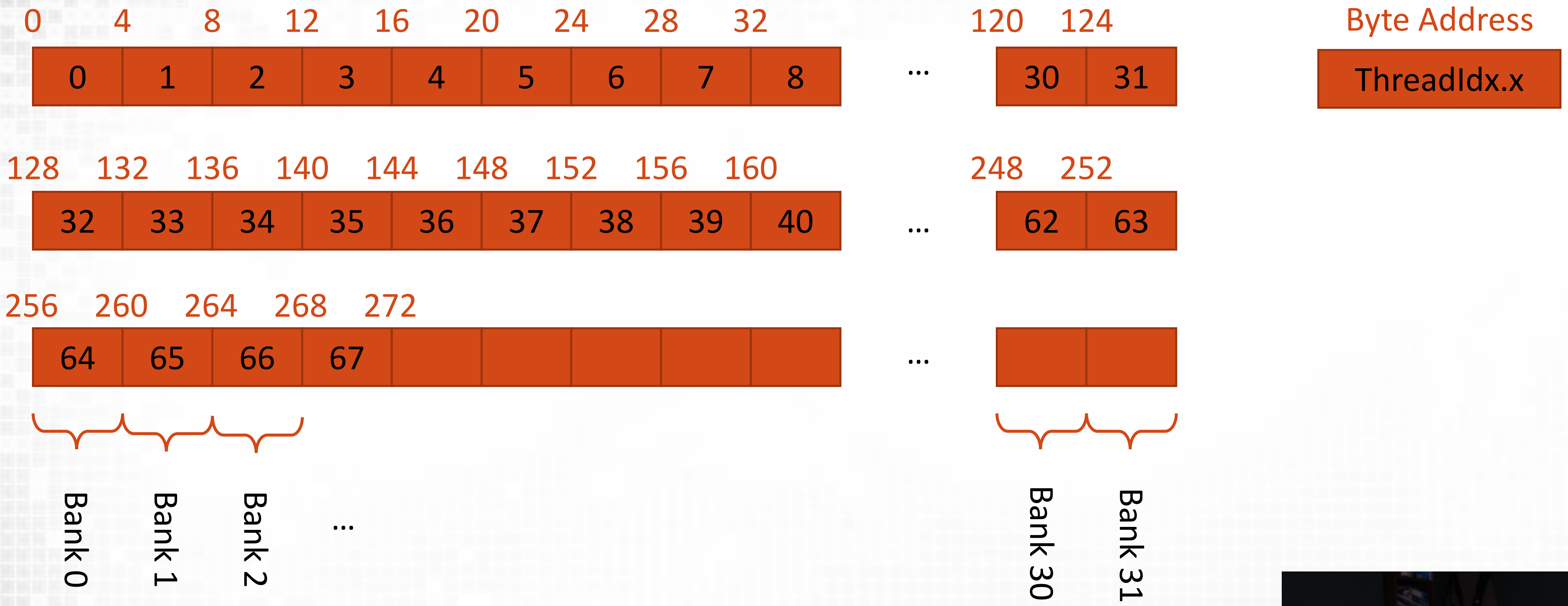
Shared Memory Bank Conflicts

- Shared memory is arranged into 4byte (32bit banks)
  - A load or store of N addresses spanning N distinct banks can be serviced simultaneously
    - Overall bandwidth of $\times N$ a single module
    - Kepler+ can also serve broadcast accesses simultaneously
- A bank conflict occurs when two threads request addresses from the same bank (without reading in broadcast pattern)
  - Results in serialisation of the access
- Bank conflicts only occur between threads in a warp
  - There are 32 banks and 32 threads per warp
  - If two threads in a warp access the same bank this is said to be a 2-way bank conflict

Think about you block sized array of floats

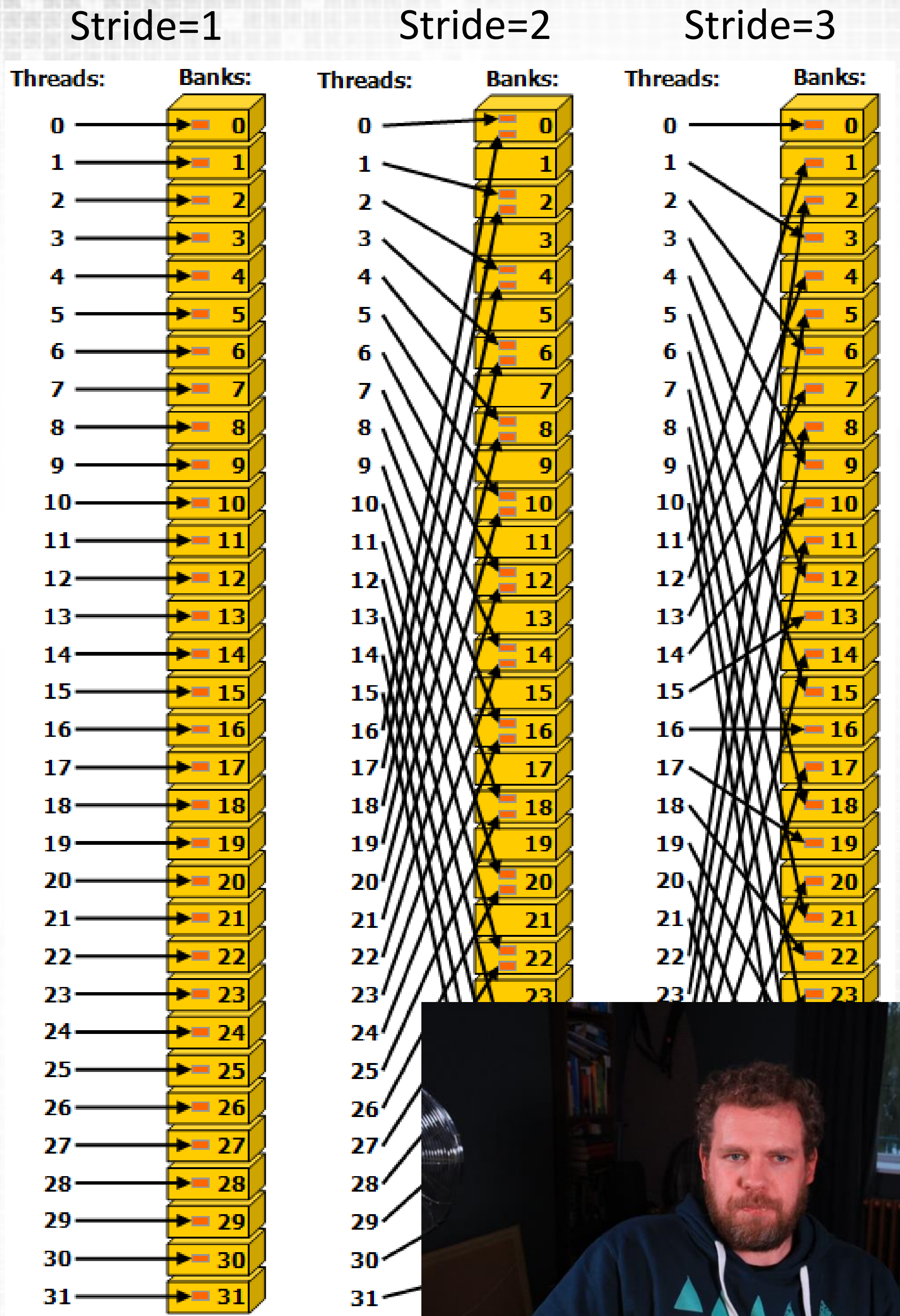
```
bank = (index * stride) % 32
```

Logical View of Shared Memory Banks



Access Strides

- ❑Stride refers to the size (in increments of the bank size) between each threads memory access pattern
  - ❑If threads access consecutive 4 byte values (e.g. `int` or `float`) then the stride is 1.
    - ❑No conflicts
  - ❑If a threads accesses consecutive 8 bytes values (e.g. `double`) then the stride is 2.
    - ❑2 way conflicts
  - ❑In general odd strides result in no conflicts



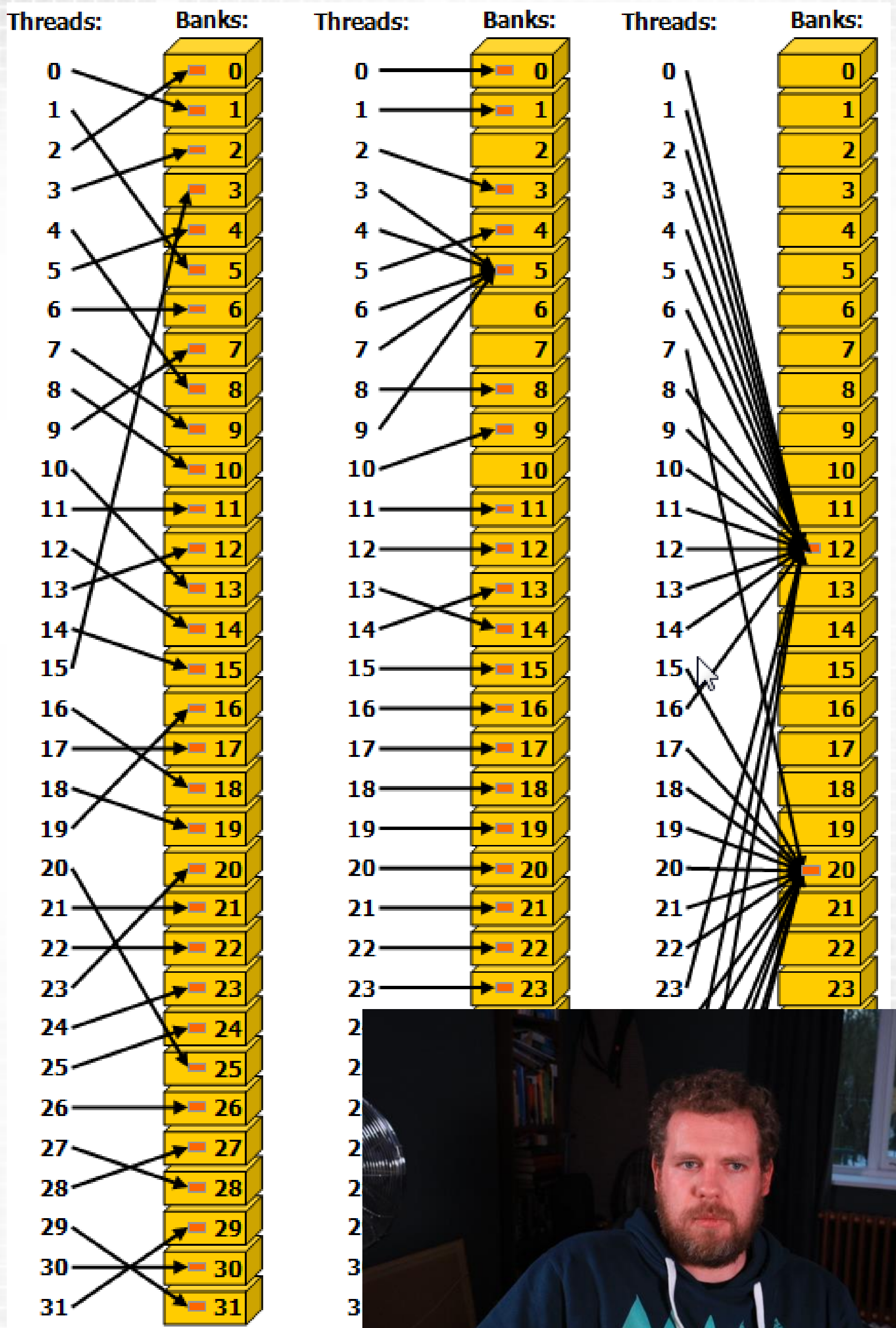
Stride (4 byte)	1			
TPB	128			
threadIdx.x			**index**	**bank**
0			0	1
1			1	2
2			2	3
3			3	4
4			4	5
5			5	6
6			6	7
7			7	8
8			8	9
9			9	10
10			10	11
31			31	12
			Banks Used	32
			Max Conflicts	1

```
bank = (index*stride) % 32
```

More on SM banks

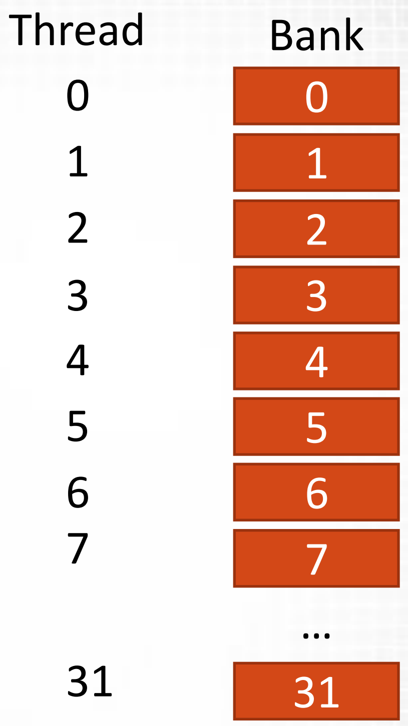
- ❑Irregular access is fine as long as no bank conflicts
- ❑Multiple threads can access the same bank conflict free **if** they access the same addresses (e.g. a broadcast access pattern)
- ❑Broadcast can be to any number of threads in a warp

```
__shared__ float s_data[??];
//read from shared memory using broadcast

some_thread_value = s_data[0] ;
```



Strided access example

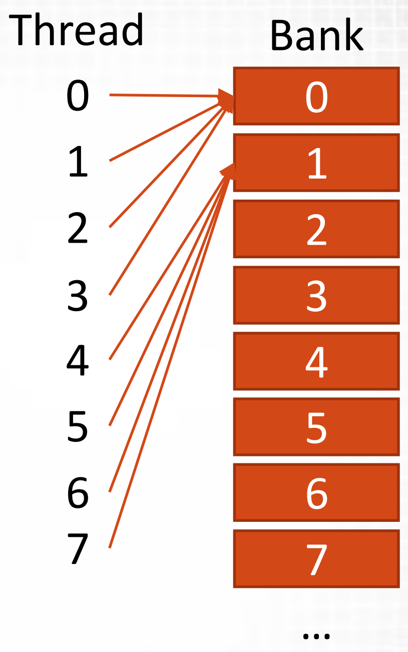
```
__shared__ float s_data[BLOCK_SIZE];

//load or calculate some_thread_value

s_data[threadIdx.x*2] = some_thread_value;
__syncthreads();
```



- ❑What is the stride?
- ❑What is the level of conflict?
- ❑How can this be improved?

Strided access example

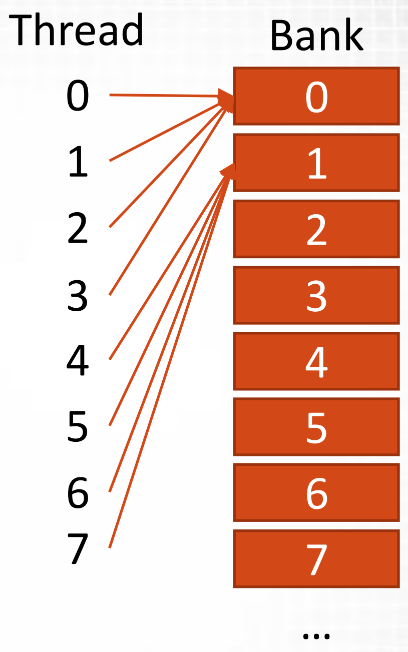
```
__shared__ float s_data[BLOCK_SIZE*2];

//load or calculate some_thread_value

s_data[threadIdx.x*2] = some_thread_value;
__syncthreads();
```



- ❑What is the stride? 2
- ❑What is the level of conflict? 2 way
- ❑How can this be improved? Remove the stride (use a stride of 1)

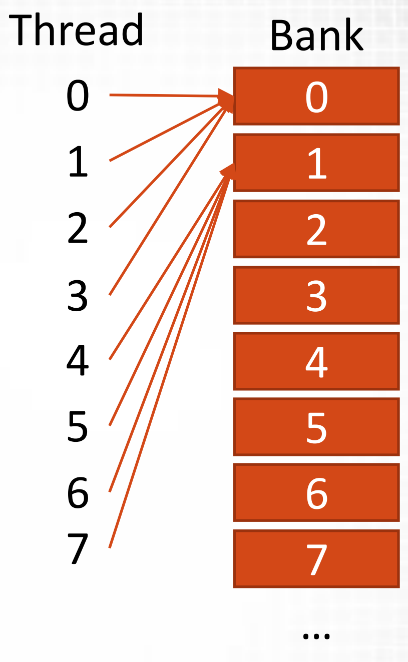
Edge case for broadcast access

```
__shared__ char s_data[BLOCK_SIZE];

//load or calculate some_thread_value

s_data[threadIdx.x] = some_thread_value;
__syncthreads();
```



Edge case for broadcast access

```
__shared__ char s_data[BLOCK_SIZE];

//load or calculate some_thread_value

s_data[threadIdx.x] = some_thread_value;
__syncthreads();
```



- Stride: 0.25 BUT no conflict (for compute capability > 2.0)
  - Why?
  - "A shared memory request for a warp does not generate a bank conflict between two threads that access any sub-word within the same 32-bit word or within two 32-bit words whose indices *i* and *j* are in the same 64-word aligned segment (i.e., a segment whose first index is a multiple of 64) and such that *j=i+32* (even though the addresses of the two sub-words fall in the same bank): In that case, for read accesses, the 32-bit words are broadcast to the requesting threads and for write accesses, each sub-word is written by only one of the threads (which thread performs the write is undefined)". - https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html#shared-memory-3-0
  - E.g. Adjacent threads accessing the same bank falls under the broadcast pattern

Large Data Types and Phasing

- For larger data types Volta+ Hardware allows scheduling of address accesses in phases
  - Bank conflicts only occur between threads in the same phase

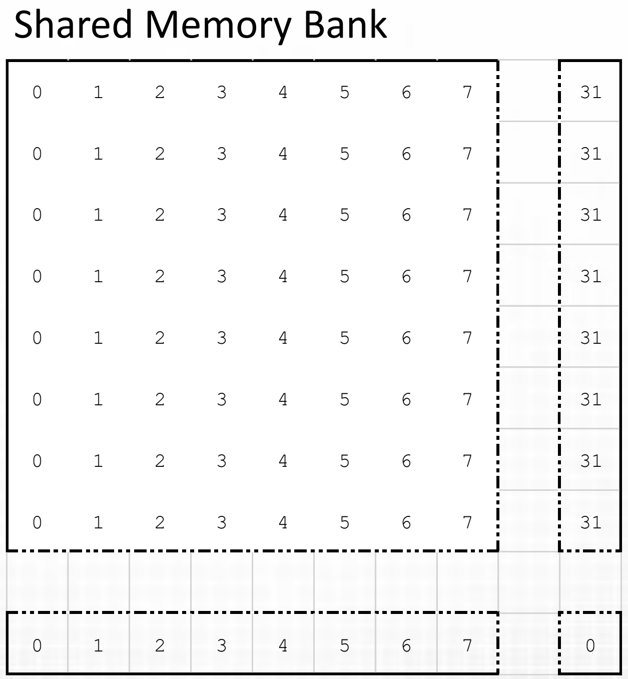
- Double and 8B data types
  - 2 Phases
    - Process addresses in first 16 threads of warp
    - Process addresses in the last 16 of the warp
- 16B data types (E.g. float4)
  - 4 Phases
    - Each phase processes addresses in a quarter of the warp

Bank conflicts with 2D tiles

```
__global__ void image_kernel(float *image)
{
  __shared__ float s_data[BLOCK_DIM][BLOCK_DIM];

  for (int i = 0; i < BLOCK_DIM; i++){
    some_thread_value += f(s_data[threadIdx.x][i]);
}
```

- ❑ Example where each thread (of 2D block) operates on a row
  - ❑ Loads values by column

```
bank = threadIdx.x * stride % 32
```



Bank conflicts with 2D tiles

```
__global__ void image_kernel(float *image)
{
  __shared__ float s_data[BLOCK_DIM][BLOCK_DIM];

  for (int i = 0; i < BLOCK_DIM; i++){
    some_thread_value += f(s_data[threadIdx.x][i]);
}
```

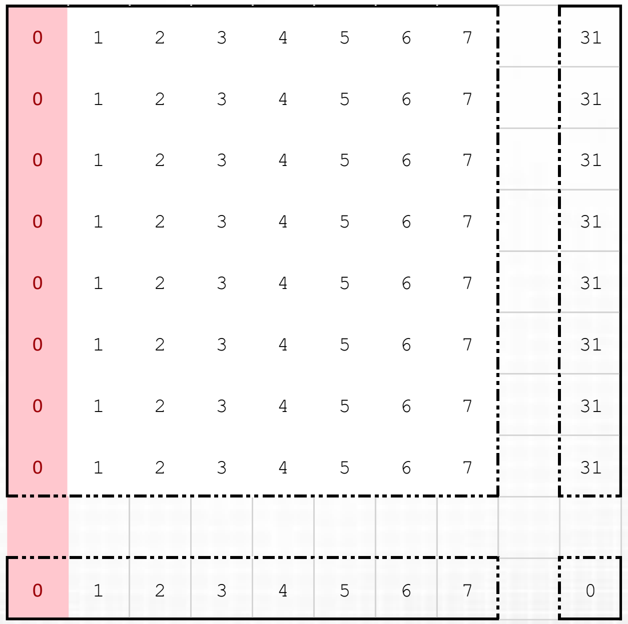
- ❑Example where each thread (of 2D block) operates on a row
  - ❑Loads values by column

```
bank = threadIdx.x * stride % 32
```

BLOCK_DIM=32

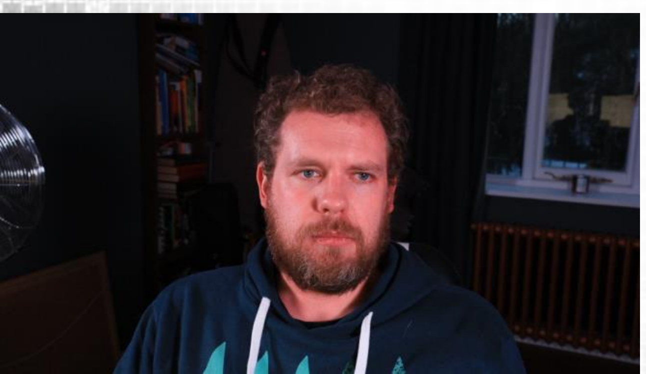
i=0

Shared Memory Bank



0	1	2	3	4	5	6	7		31
0	1	2	3	4	5	6	7		31
0	1	2	3	4	5	6	7		31
0	1	2	3	4	5	6	7		31
0	1	2	3	4	5	6	7		31
0	1	2	3	4	5	6	7		31
0	1	2	3	4	5	6	7		31
0	1	2	3	4	5	6	7		31
0	1	2	3	4	5	6	7		0

- ❑32 way bank conflicts!
  - ❑Very bad
- ❑Stride = 32



Bank conflicts with 2D tiles

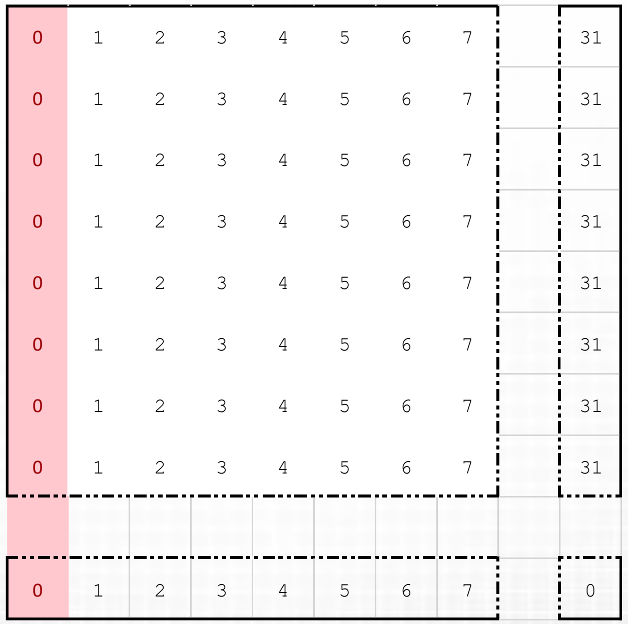
```
__global__ void image_kernel(float *image)
{
  __shared__ float s_data[BLOCK_DIM][BLOCK_DIM];

  for (int i = 0; i < BLOCK_DIM; i++){
    some_thread_value += f(s_data[threadIdx.x][i]);
}
```

```
bank = threadIdx.x * stride % 32
```

```
BLOCK_DIM=32
```

i=0

Shared Memory Bank



- ❑ Example where each thread (of 2D block) operates on a row
  - ❑ Loads values by column

- ❑ **How to fix**
  - ❑ **Memory padding**
  - ❑ **Transpose the matrix**
    - ❑ **Or operate on columns (loading by row) if possible**

Bank conflicts with 2D tiles

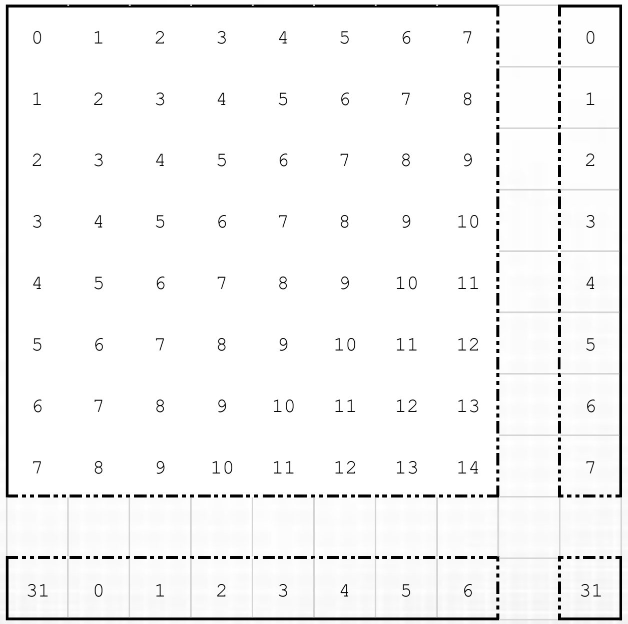
```
__global__ void image_kernel(float *image)
{
  __shared__ float s_data[BLOCK_DIM][BLOCK_DIM+1];

  for (int i = 0; i < BLOCK_DIM; i++){
    some_thread_value += f(d_data[threadIdx.x][i]);
}
```

- Memory Padding Solution

```
bank = threadIdx.x * stride %32
```

BLOCK_DIM+1=33

Shared Memory Bank



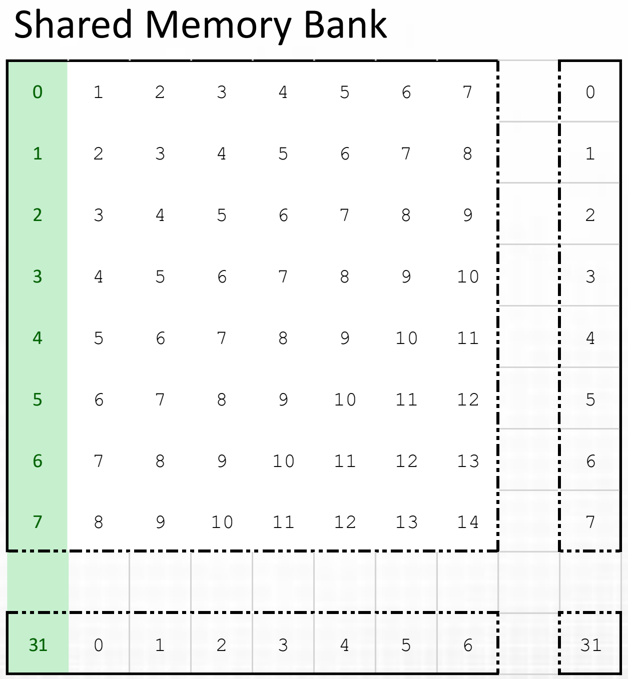
Bank conflicts with 2D tiles

```
__global__ void image_kernel(float *image)
{
  __shared__ float s_data[BLOCK_DIM][BLOCK_DIM+1];

  for (int i = 0; i < BLOCK_DIM; i++){
    some_thread_value += f(d_data[threadIdx.x][i]);
}
```

```
bank = threadIdx.x * stride % 32
```

```
BLOCK_DIM+1=33
```

i=0


Shared Memory Bank

0	1	2	3	4	5	6	7		0
1	2	3	4	5	6	7	8		1
2	3	4	5	6	7	8	9		2
3	4	5	6	7	8	9	10		3
4	5	6	7	8	9	10	11		4
5	6	7	8	9	10	11	12		5
6	7	8	9	10	11	12	13		6
7	8	9	10	11	12	13	14		7
31	0	1	2	3	4	5	6		31

- ❑ **Memory Padding Solution**

- ❑ Every thread in warp reads from different bank
- ❑ *Alternative: Transpose solution left to you!*

Summary

- ❑Shared Memory Bank Conflicts
  - ❑Explain the concepts of memory banks and how conflicts can occur
  - ❑Give examples of access strides and the impact on bank conflicts
  - ❑Present solutions for avoiding bank conflicts

- ❑Next Lecture: Boundary Conditions

Kepler Shared Memory Bank Size

- In Kepler it is possible to specify the size of shared memory banks
  - This changes the bank conflicts depending on your access pattern
- `cudaDeviceSharedMemConfig`
  - Options are;
    - `cudaSharedMemBankSizeDefault`: default size
    - `cudaSharedMemBankSizeFourByte`: 32 bit size banks
    - `cudaSharedMemBankSizeEightByte`: 64 bit size banks
  - Can improve performance of access for even strides.
- Not available in Maxwell +.

The University Of Sheffield.