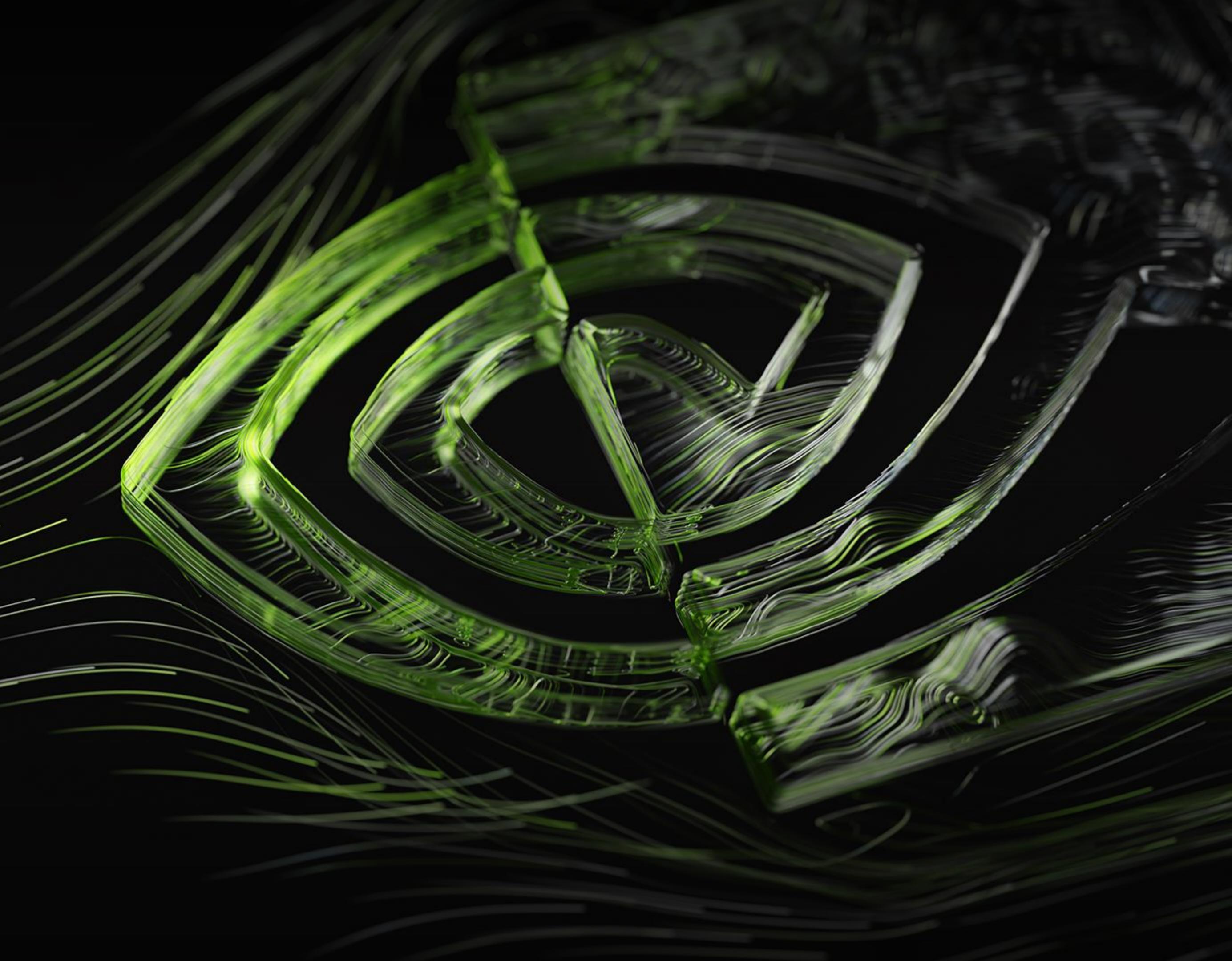


CSCI 5451 Jeremy Iverson | April 20, 2023



Your experience with CUDA





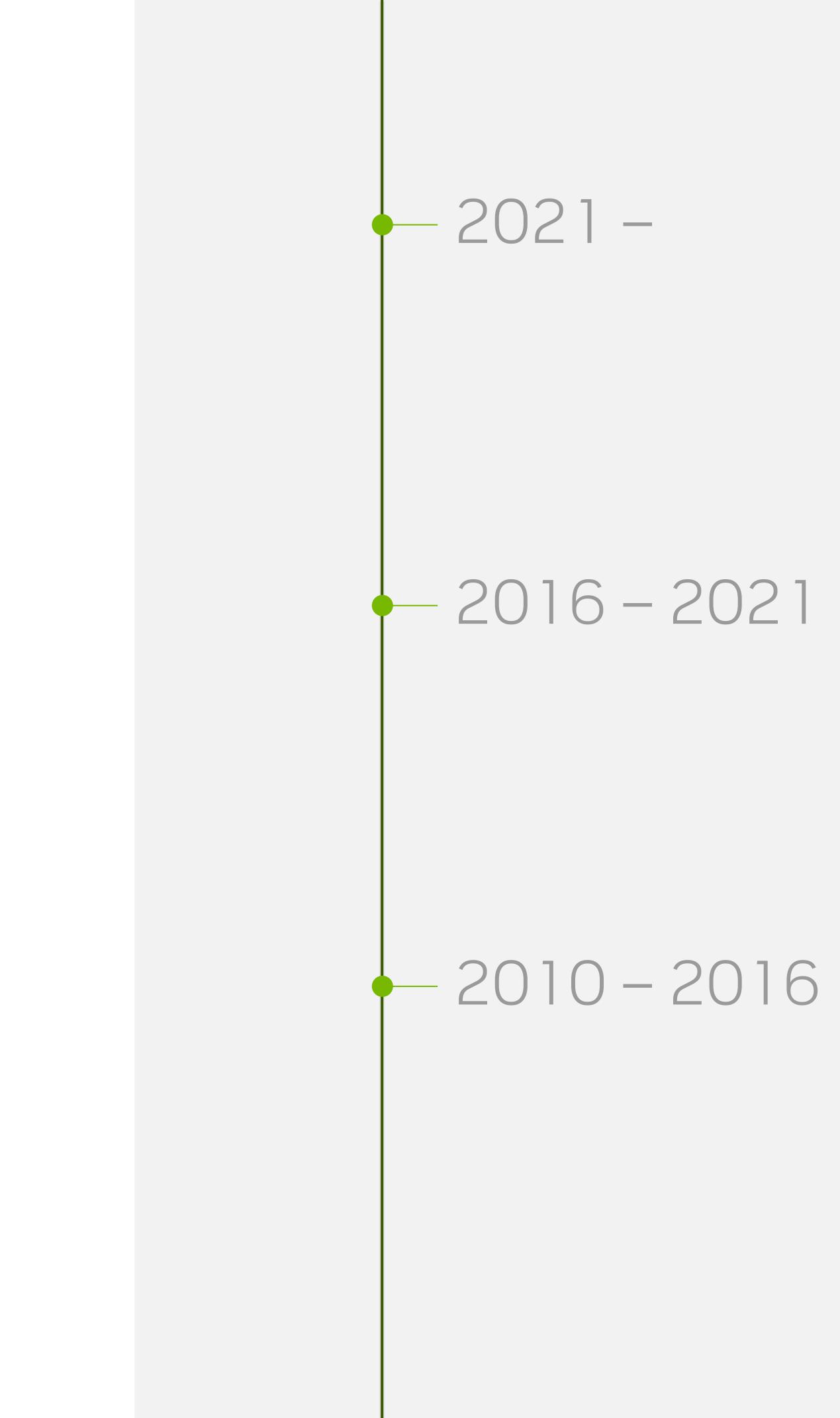


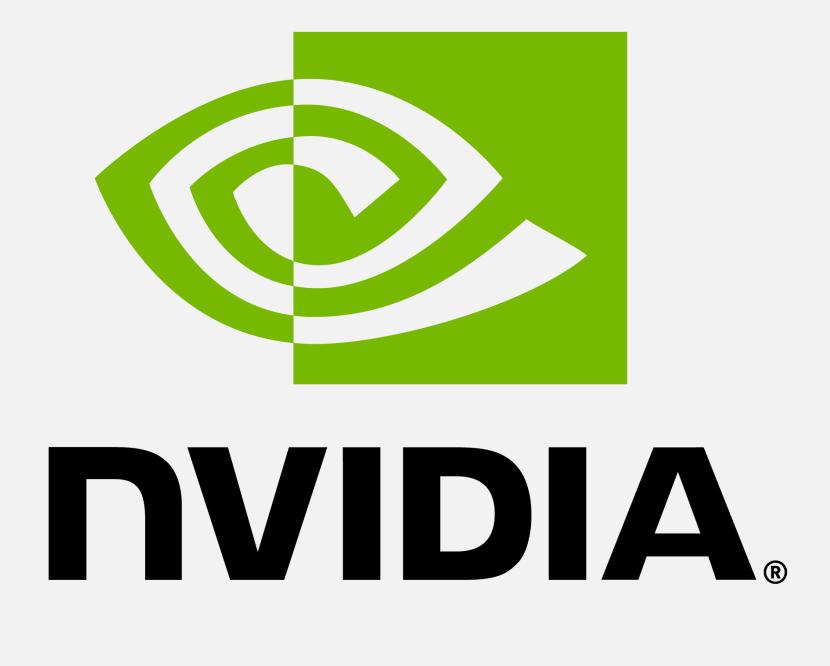
- About me
- GPU occupancy
- Thread block clusters

HW/SW co-design at NVIDIA



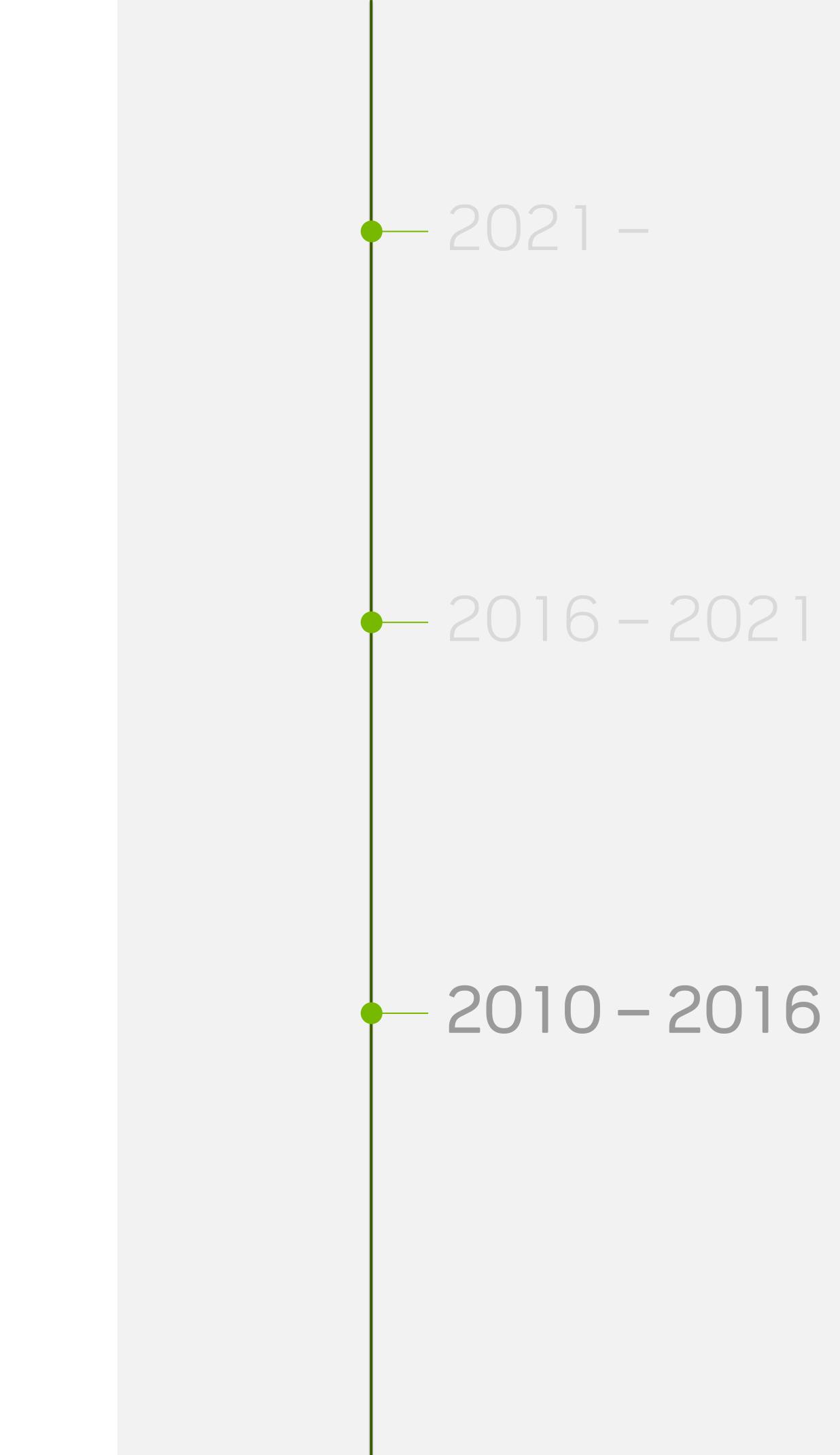


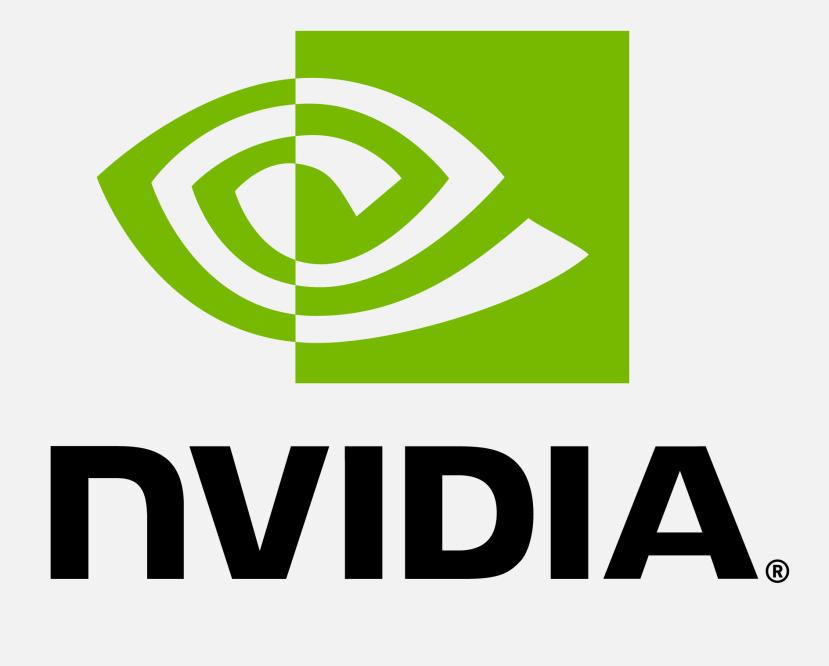






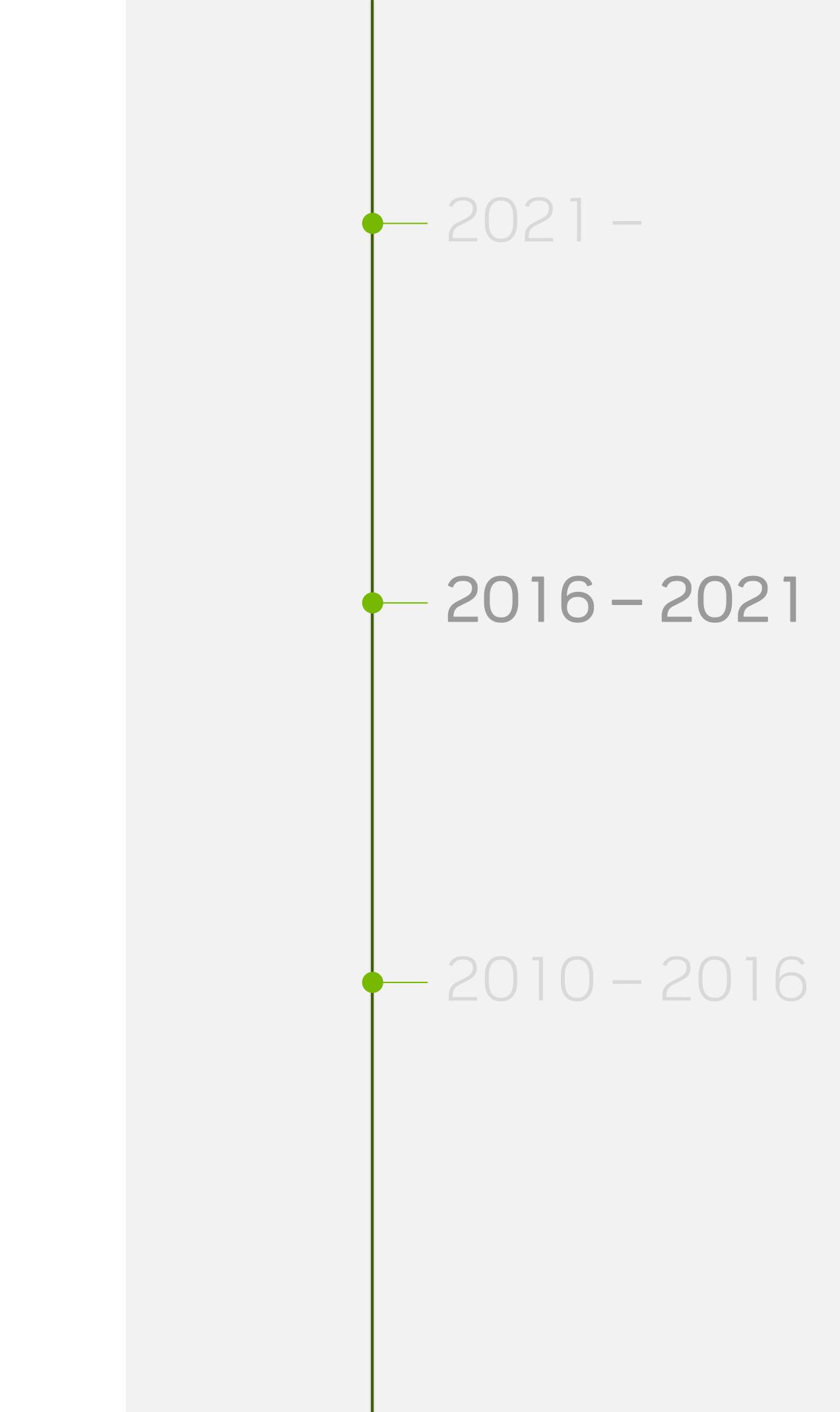








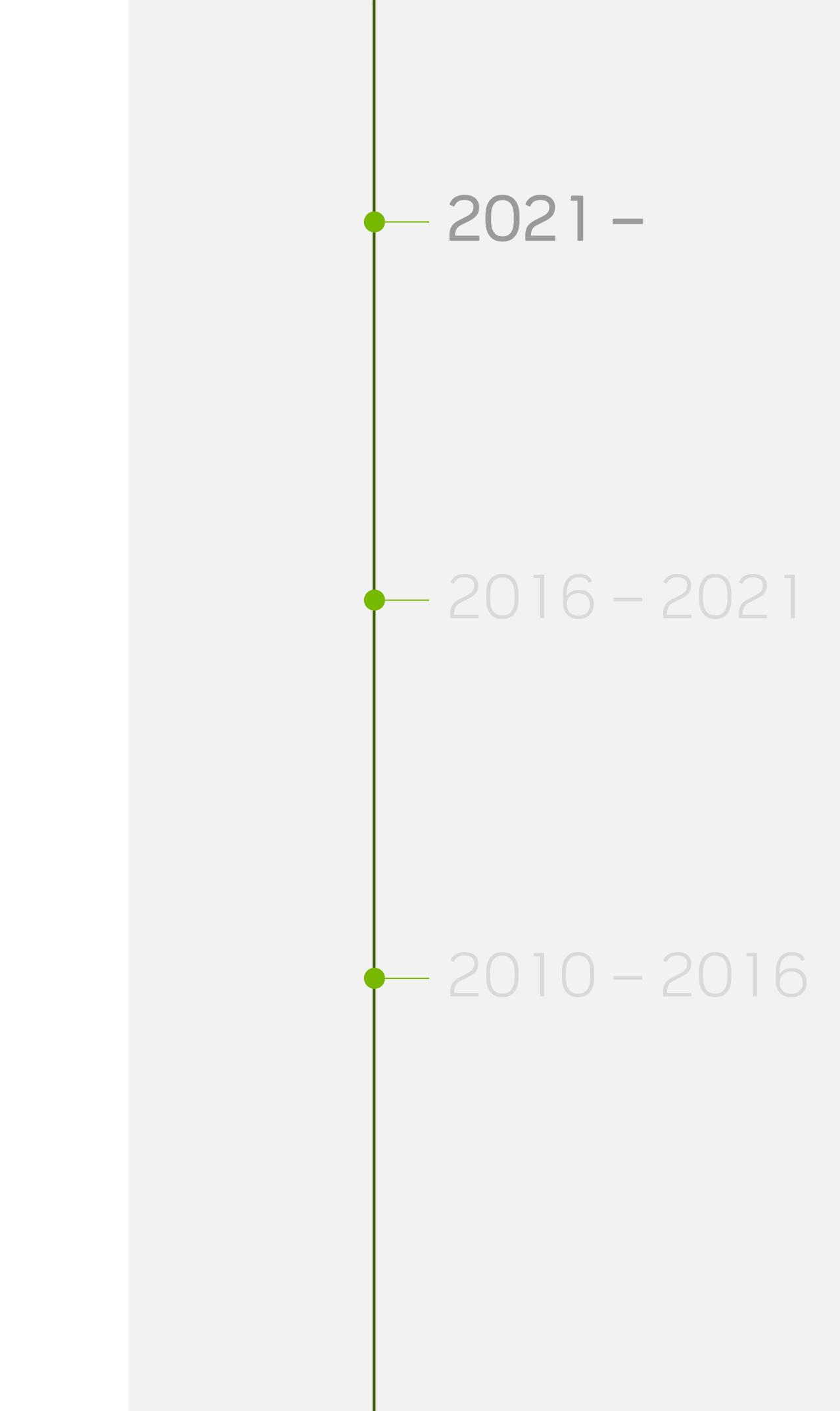
















- Different uses of GPUs: HPC, AI, Graphics, ProViz
- Different needs drive different requirements from HW
- HW innovates based on perceived needs of usages
- SW gathers feedback and characterizes customer needs
- SW considers how the innovation might address open problems
- SW considers how the innovate features will be exposed to users

HW/SW Co-design

8

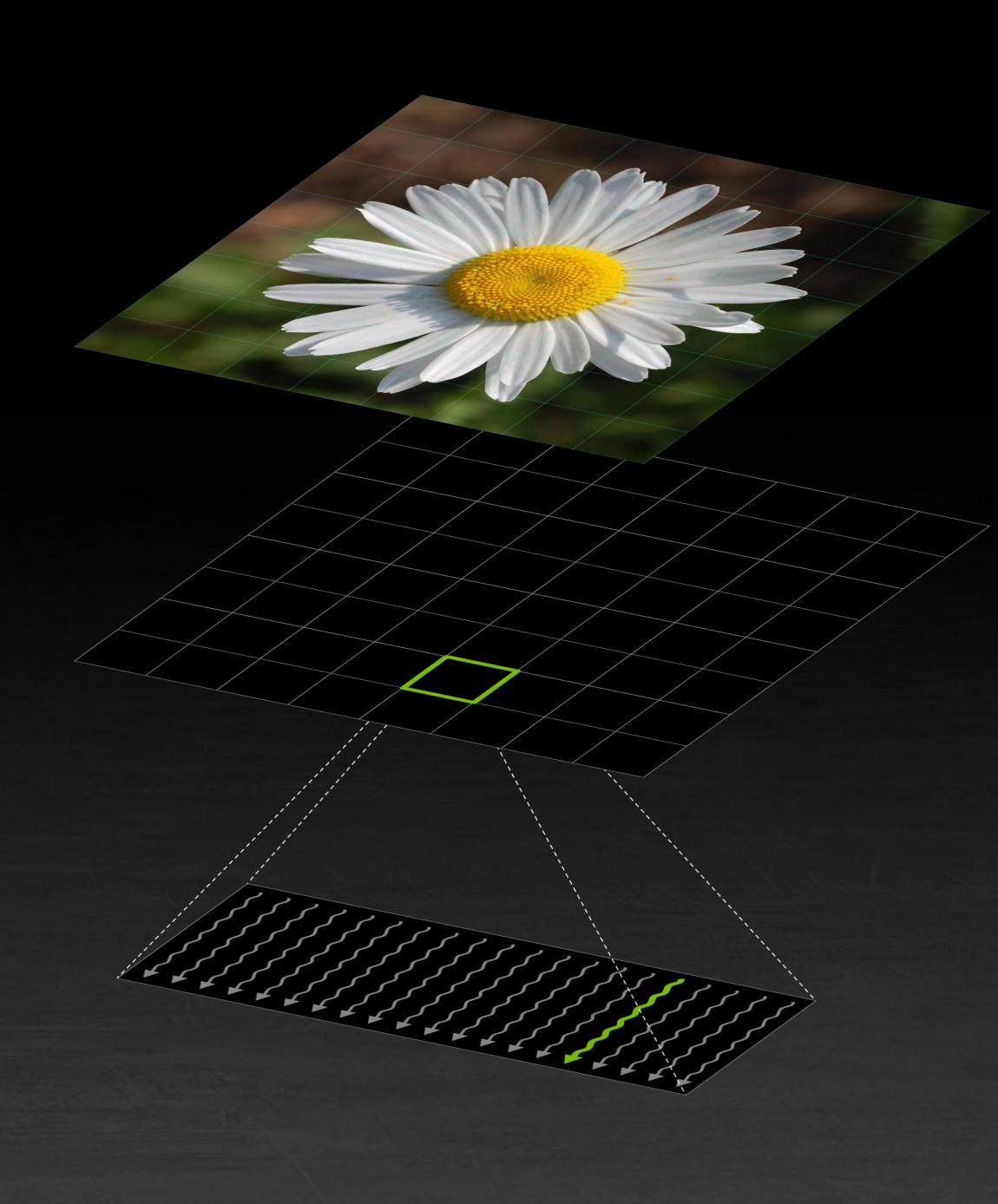


WHY IS THE WAY IT IS WOW CUDA PROGRAMMING WORKS STEPHEN JONES, GTC FALL 2022





CUDA'S GPU EXECUTION HIERARCHY



Grid of work

Divide into many blocks

Many threads in each block

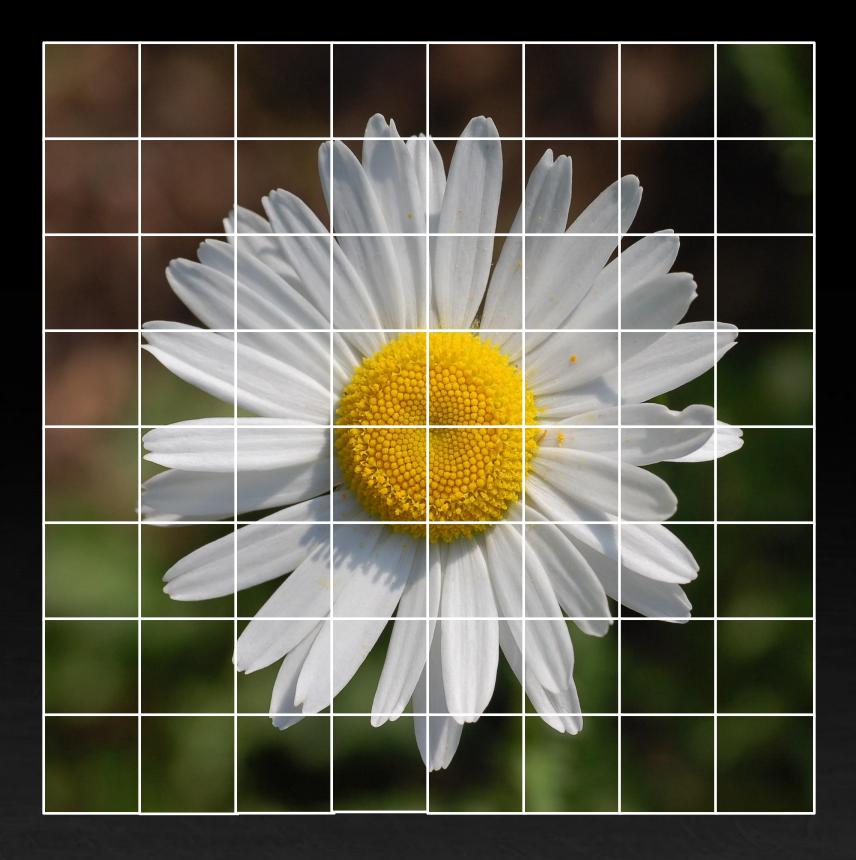


START WITH SOME WORK TO PROCESS



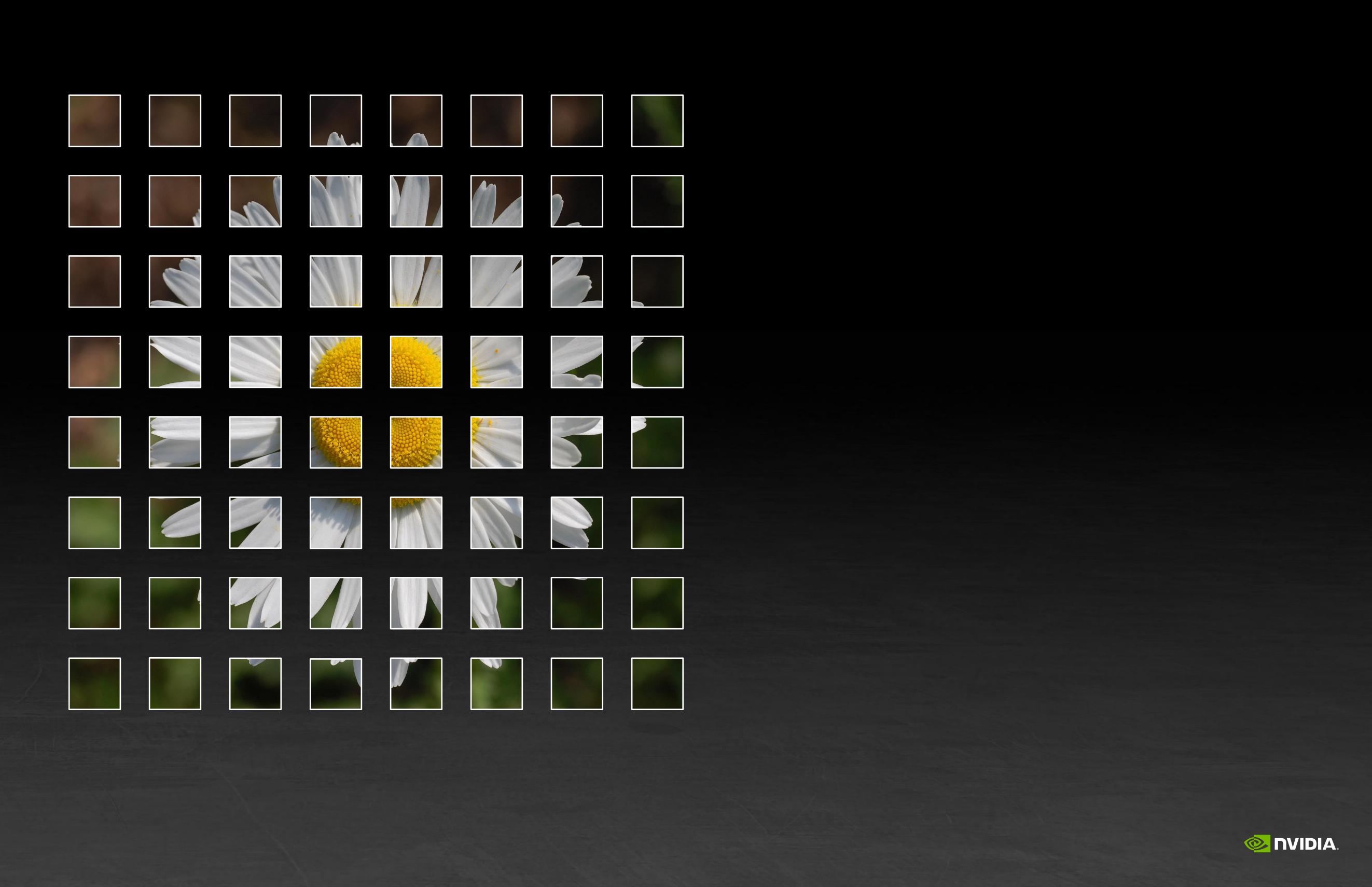


DIVIDE INTO A SET OF EQUAL-SIZED BLOCKS: THIS IS THE "GRID" OF WORK





EACH BLOCK WILL NOW BE PROCESSED INDEPENDENTLY CUDA does not guarantee the order of execution and you cannot exchange data between blocks



EACH BLOCK WILL NOW BE PROCESSED INDEPENDENTLY CUDA does not guarantee the order of execution and you cannot exchange data between blocks





EVERY BLOCK GET PLACED ONTO AN SM CUDA does not guarantee the order of execution and you cannot exchange data between blocks





EVERY BLOCK GET PLACED ONTO AN SM CUDA does not guarantee the order of execution and you cannot exchange data between blocks











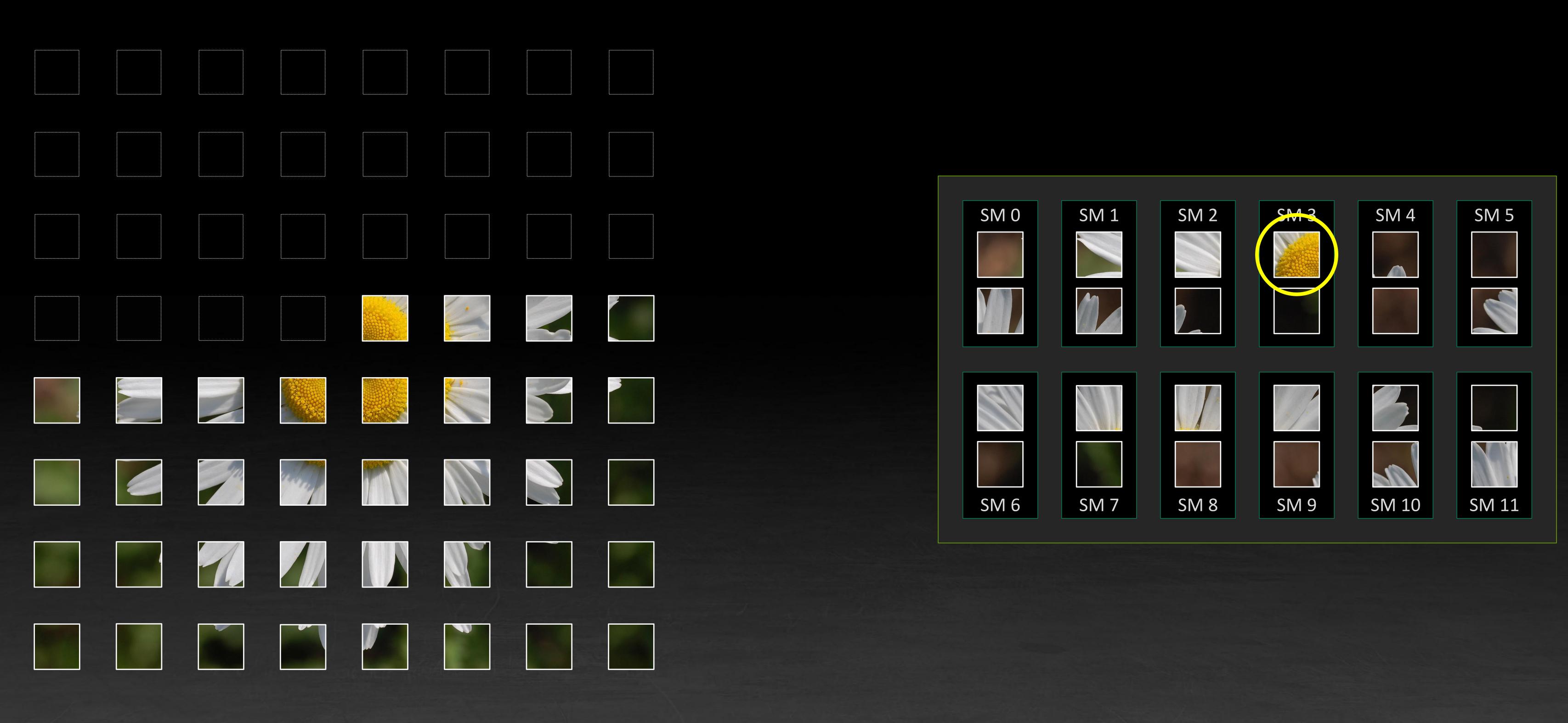
















		SM			L1 In:	struction Cache
			LO) Instruction (
			Contract Street, Stree	cheduler (32 t tch Unit (32 th	STATEMENT AND A	
			Registe	er File (16,38	4 x 32-bit)	
	L	FP64	INT IN	T FP32 FP32		FP64
		FP64	INT IN	T FP32 FP32		FP64
		FP64	INT IN	T FP32 FP32		FP64
		FP64	INT IN	T FP32 FP32	TENSOR CORE	FP64
		FP64	INT IN	T FP32 FP32	and the second se	FP64
		FP64	INT IN			FP64
		FP64	INT IN	T FP32 FP32		FP64
		FP64	INT IN			FP64
		LDV LDV ST ST	LDY LC ST S	DY LDY LDY T ST ST	ST ST SFU	
) Instruction (No. (NO.CH	
				cheduler (32 t tch Unit (32 th	and a second	
				er File (16,38		
		FP64	INT IN	T FP32 FP32		FP64
		FP64	INT IN	T FP32 FP32		FP64
		FP64	INT IN	T FP32 FP32		FP64
		FP64	INT IN	T FP32 FP32	TENSOR CORE	FP64
		FP64	INT IN	T FP32 FP32	ILIIOON CONL	FP64
		FP64	INT IN	T FP32 FP32		FP64
		FP64	INT IN	T FP32 FP32		FP64
		FP64	INT IN			FP64
		LDY LDY ST ST	LDY LO ST ST	DY LDY LDY T ST ST	LDY LDY ST ST SFU	LD/ LC/ ST S
					192KB L1 Data	Cache / Shared
			Тех		Tex	
						1500

WHAT DOES IT MEAN FOR AN SM TO BE "FULL"?





LOOKING INSIDE A STREAMING MULTIPROCESSOR

SM .											
			L1 Instruct	ion Cache							
-	LO I	nstruction C	ache	1		L0 Ir	nstruct	ion C	ache		
-	Warp Sch	neduler (32 1	thread/cik)		War	p Sch	eduler	(32 1	hread/o	ik)	
2	Dispate	h Unit (32 th	read/clk)	1	Di	spatcl	h Unit ((32 th	read/cl	k)	
	Register	File (16,38	4 x 32-bit)		Reg	ister	File (1	6,384	4 x 32-	bit)	
FP64	INT INT	FP32 FP32		FP64	INT	INT	FP32	FP32			
FP64	INT INT	FP32 FP32		FP64	INT	INT	FP32	FP32			
FP64	INT INT	FP32 FP32		FP64	INT	INT	FP32	FP32			
FP64	INT INT	FP32 FP32		FP64	INT	INT	FP32	FP32			
FP64	INT INT	FP32 FP32	TENSOR CORE	FP64	INT	INT	FP32	FP32	TE	NSOF	RCORE
FP64	INT INT	FP32 FP32		FP64	INT	INT	FP32	FP32			
FP64	INT INT	FP32 FP32		FP64	INT	INT	FP32	FP32			
FP64	INT INT	FP32 FP32		FP64	INT	INT	FP32				
					LD	LD	LD/	LDY	LDY	LDY	SFU
ST ST	ST ST	ST ST	ST ST SPU	ST ST	ST	ST	ST	ST	ST	ST	GIU
-	LO In	nstruction C	ache	-		L0 Ir	nstruct	tion C	ache		
	Warp Sch	neduler (32 1	thread/cik)	1	-		NAMES OF TAXABLE PARTY.	-	hread/o		
1	Dispate	h Unit (32 th	read/clk)		Di	spate	h Unit ((32 th	read/cl	k)	
	Register	File (16,38	4 x 32-bit)		Reg	ister	File (1	6,384	4 x 32-	bit)	
FP64	INT INT	FP32 FP32		FP64	INT	INT	FP32	FP32			
FP64	INT INT	FP32 FP32		FP64	INT	INT	FP32	FP32			
FP64	INT INT	FP32 FP32		FP64	INT	INT	FP32	FP32			
FP64	INT INT	FP32 FP32	the second se	FP64	INT	INT	FP32	FP32	2550		
FP64	INT INT	FP32 FP32	TENSOR CORE	FP64	INT	INT	FP32	FP32	TE	NSOF	RCORE
FP64	INT INT	FP32 FP32		FP64	INT	INT	FP32	FP32			
FP64	INT INT	FP32 FP32		FP64	INT	INT	FP32	FP32			
FP64	INT INT	FP32 FP32		FP64	INT	INT	FP32	FP32			
LDY LDY ST ST	LDY LDY ST ST	LDY LDY ST ST	ST ST SFU	LD/ LD/ ST ST	LD' ST	LDV ST	LD/ ST	LD/ ST	LDV ST	LDY ST	SFU
			192KB L1 Data Cach	ie / Shared M	emory						
8	Tex		Tex		Tex					Tex	
2 (A)	1989-05		18/201		AT SKADS		-	-		001200	

A 1	
2048	
32	
65,536	
160 kB	
32	
4	
64	
32	
192 kB	
90 GB/sec	
1410 MHz	

100 SM Resources

Max threads per SM

Max blocks per SM

Total registers per SM

Total shared memory in SM

Threads per warp

Concurrent warps active

FP32 cores per SM

FP64 cores per SM

Max L1 cache size

Load bandwidth per SM

GPU Boost Clock



LOOKING INSIDE A STREAMING MULTIPROCESSOR

1					L1 instruc	tion Cach:
	1	0 instruct	ion C:	ache		
_	Constant Property in such	Scheduler	or a real of a	CONTRACTOR OF STREET		
	Dispa	atch Unit (32 01	read/clk)		
	Regist	ter File (1	6,384	i x 32-bit)		
FP64	INT IN	T FP32	FP32			FP64
FP64	INT IN	IT FP32	FP32			FP64
FP64	INT IN	IT FP32	FP32			FP64
FP64	INT IN	IT FP32	FP32			FP64
FP64	INT IN	IT FP32	FP32	TENSO	RCORE	FP64
FP64	INT IN	IT FP32	FP32			FP64
FP64	INT IN	IT FP32	FP32			FP64
FP64	INT IN	T FP32	FP32			FP64
LDV LDV ST ST		DY LDY ST ST	LD/ ST	LDY LDY ST ST	SFU	LDY ST
_		0 Instruct	ion C	ache		
	Warp 8	Scheduler	(32 1	hread/cik)		
	Dispa	atch Unit ((32 th	read/clk)		
	Regist	ter File (1	6,384	x 32-bit)		
FP64	INT IN	T FP32	FP32			FP64
FP64	INT IN	IT FP32	FP32			FP64
FP64	INT IN	IT FP32	FP32			FP64
FP64	INT IN	T FP32	FP32			FP64
FP64	INT IN	T FP32	FP32	TENSO	RCORE	FP64
FP64	INT IN	T FP32	FP32			FP64
FP64	INT IN	T FP32	FP32			FP64
FP64	INT IN	T FP32	FP32			FP64
LDY LDY ST ST	LDY L ST S	DY LDY ST ST	LDY ST	LDY LDY ST ST	SFU	LD/ I ST
				192K	B L1 Data Ca	he / Share



A1
2048
32
65,536
160 kB
32
4
64
32
192 kB
90 GB/sec
1410 MHz

OO SM Resources

Max threads per SM

Max blocks per SM

Total registers per SM

Total shared memory in SM

Threads per warp

Concurrent warps active

FP32 cores per SM

FP64 cores per SM

Max L1 cache size

Load bandwidth per SM

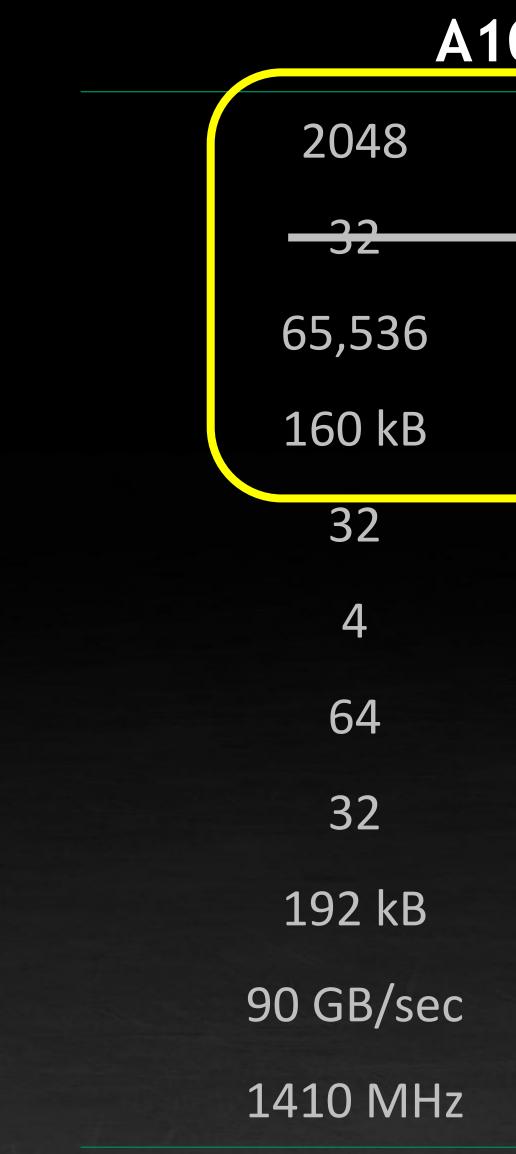
GPU Boost Clock



LOOKING INSIDE A STREAMING MULTIPROCESSOR

1					L1 instruc	tion Cach:
	1	0 instruct	ion C:	ache		
_	Constant Property in such	Scheduler	or a real of a	CONTRACTOR OF STREET		
	Dispa	atch Unit (32 01	read/clk)		
	Regist	ter File (1	6,384	i x 32-bit)		
FP64	INT IN	T FP32	FP32			FP64
FP64	INT IN	IT FP32	FP32			FP64
FP64	INT IN	IT FP32	FP32			FP64
FP64	INT IN	IT FP32	FP32			FP64
FP64	INT IN	IT FP32	FP32	TENSO	RCORE	FP64
FP64	INT IN	IT FP32	FP32			FP64
FP64	INT IN	IT FP32	FP32			FP64
FP64	INT IN	T FP32	FP32			FP64
LDV LDV ST ST		DY LDY ST ST	LD/ ST	LDY LDY ST ST	SFU	LDY ST
_		0 Instruct	ion C	ache		
	Warp 8	Scheduler	(32 1	hread/cik)		
	Dispa	atch Unit ((32 th	read/clk)		
	Regist	ter File (1	6,384	x 32-bit)		
FP64	INT IN	T FP32	FP32			FP64
FP64	INT IN	IT FP32	FP32			FP64
FP64	INT IN	IT FP32	FP32			FP64
FP64	INT IN	T FP32	FP32	-		FP64
FP64	INT IN	T FP32	FP32	TENSO	RCORE	FP64
FP64	INT IN	T FP32	FP32			FP64
FP64	INT IN	T FP32	FP32			FP64
FP64	INT IN	T FP32	FP32			FP64
LDY LDY ST ST	LDY L ST S	DY LDY ST ST	LDY ST	LDY LDY ST ST	SFU	LD/ I ST
				192K	B L1 Data Ca	he / Share





A100 SM Resources

Max threads per SM

Max blocks per SM

Total registers per SM

Total shared memory in SM

Threads per warp

Concurrent warps active

FP32 cores per SM

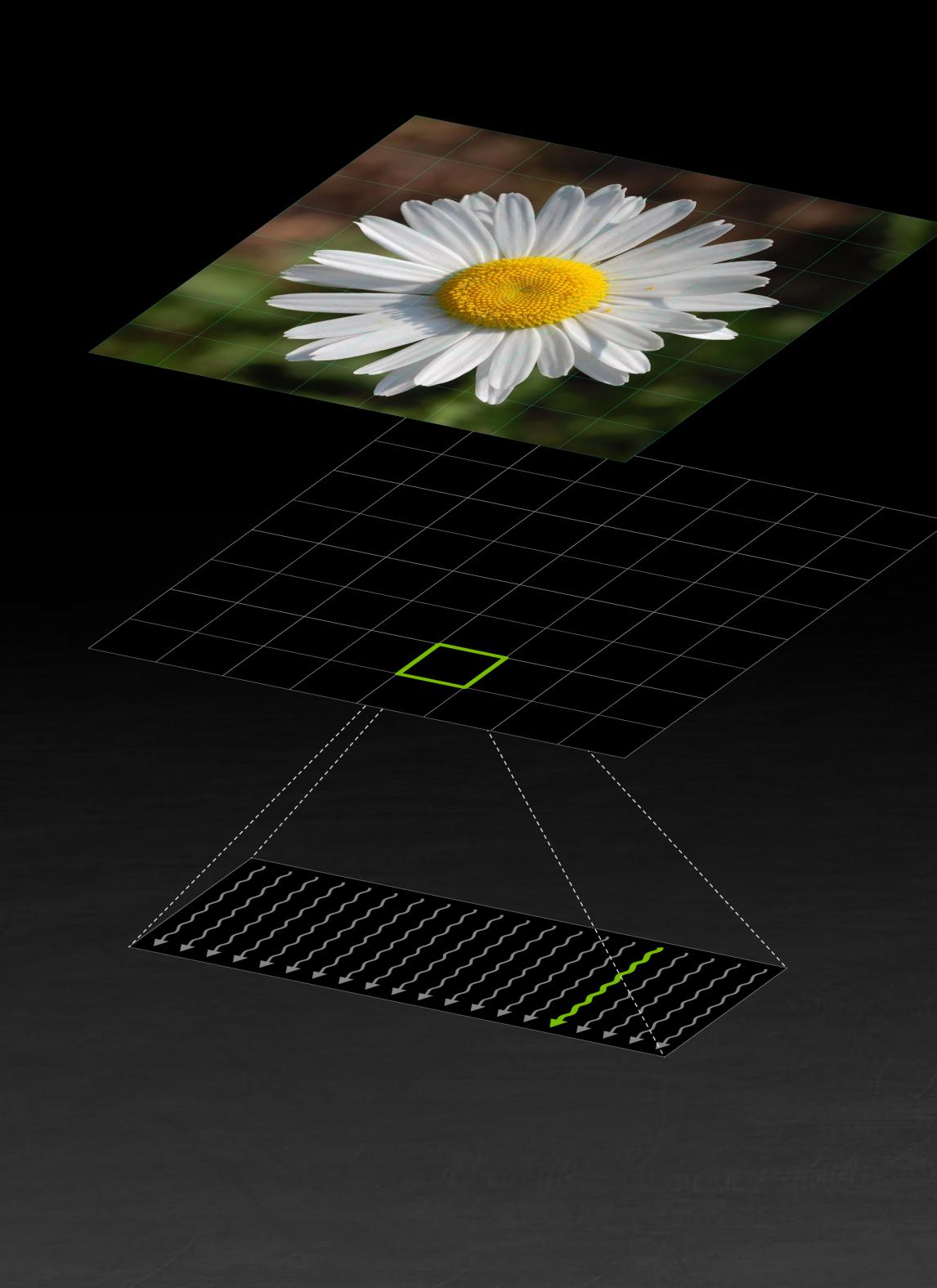
FP64 cores per SM

Max L1 cache size

Load bandwidth per SM

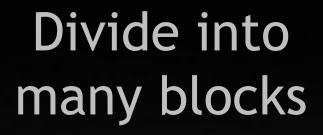
GPU Boost Clock



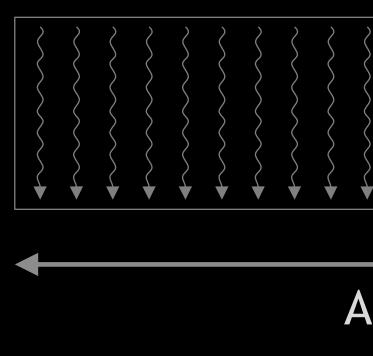


THE CUDA PROGRAMMING MODEL

Grid of work



Many threads in each block



_____shared____float mean = 0.0f;

return dist;

Every thread runs exactly the same program (this is the "SIMT" model)

Thread block

A block has a fixed number of threads

```
__device__ float mean_euclidian_distance(float2 *p1, float2 *p2) {
   // Compute the Euclidian distance between two points
   float2 dp = p2[threadIdx.x] - p1[threadIdx.x];
   float dist = sqrtf(dp.x * dp.x + dp.y * dp.y);
```

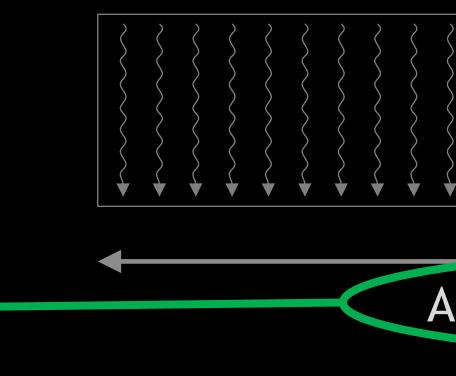
```
// Accumulate the mean distance atomically and return distance
atomicAdd(&mean, dist / blockDim.x);
```

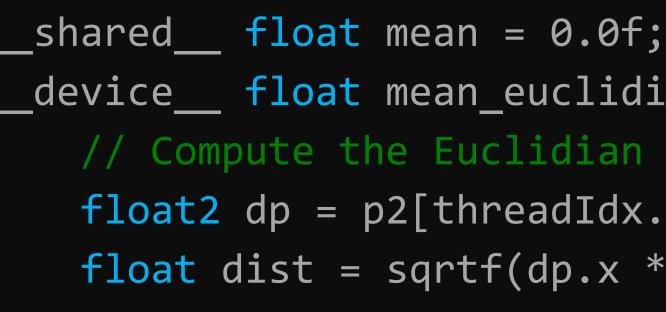


All blocks in a grid run the same program using the same number of threads, leading to 3 resource requirements

1. Block size - the number of threads which must be concurrent

ANATOMY OF A THREAD BLOCK





return dist;

Every thread runs exactly the same program (this is the "SIMT" model)

Thread block

A block has a fixed number of threads

```
__device__ float mean_euclidian_distance(float2 *p1, float2 *p2) {
   // Compute the Euclidian distance between two points
   float2 dp = p2[threadIdx.x] - p1[threadIdx.x];
   float dist = sqrtf(dp.x * dp.x + dp.y * dp.y);
```

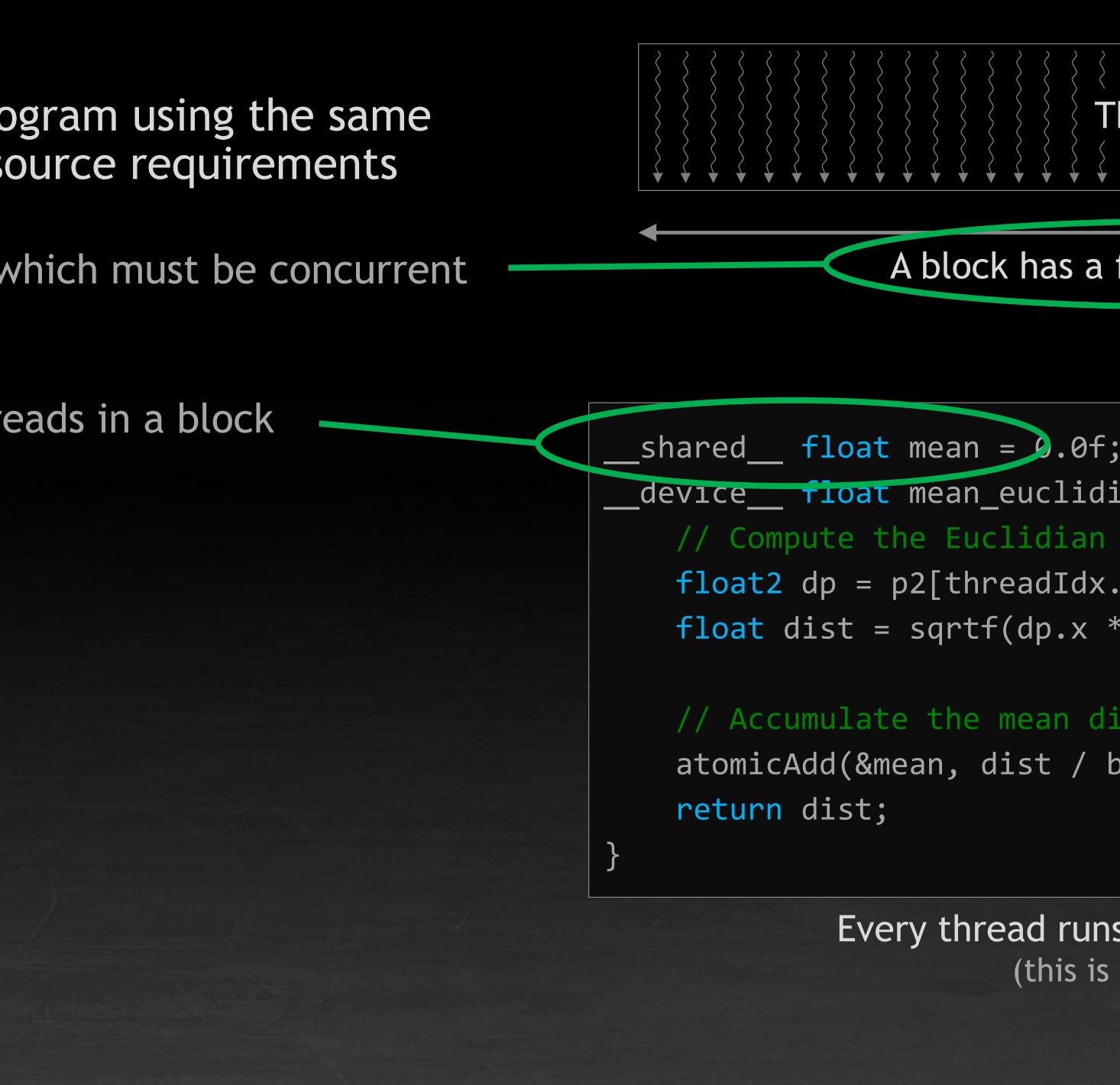
```
// Accumulate the mean distance atomically and return distance
atomicAdd(&mean, dist / blockDim.x);
```



All blocks in a grid run the same program using the same number of threads, leading to 3 resource requirements

- 1. Block size the number of threads which must be concurrent
- 2. Shared memory common to all threads in a block

ANATOMY OF A THREAD BLOCK



A block has a fixed number of threads

__shared__ float mean = 0.0f; __device__ float mean_euclidian_distance(float2 *p1, float2 *p2) { // Compute the Euclidian distance between two points float2 dp = p2[threadIdx.x] - p1[threadIdx.x]; float dist = sqrtf(dp.x * dp.x + dp.y * dp.y);

```
// Accumulate the mean distance atomically and return distance
atomicAdd(&mean, dist / blockDim.x);
```

Every thread runs exactly the same program (this is the "SIMT" model)

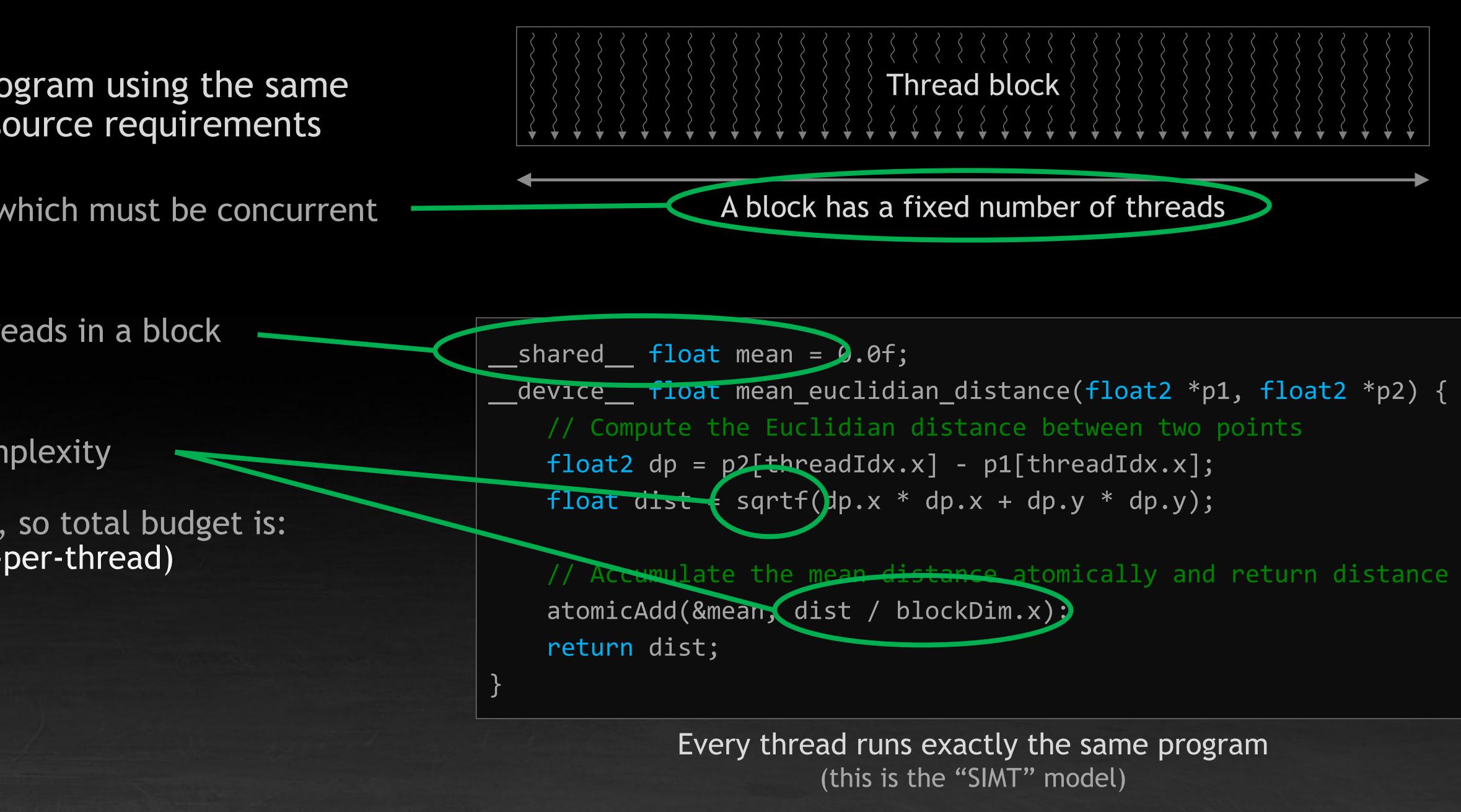


All blocks in a grid run the same program using the same number of threads, leading to 3 resource requirements

- Block size the number of threads which must be concurrent
- 2. Shared memory common to all threads in a block
- 3. Registers depends on program complexity

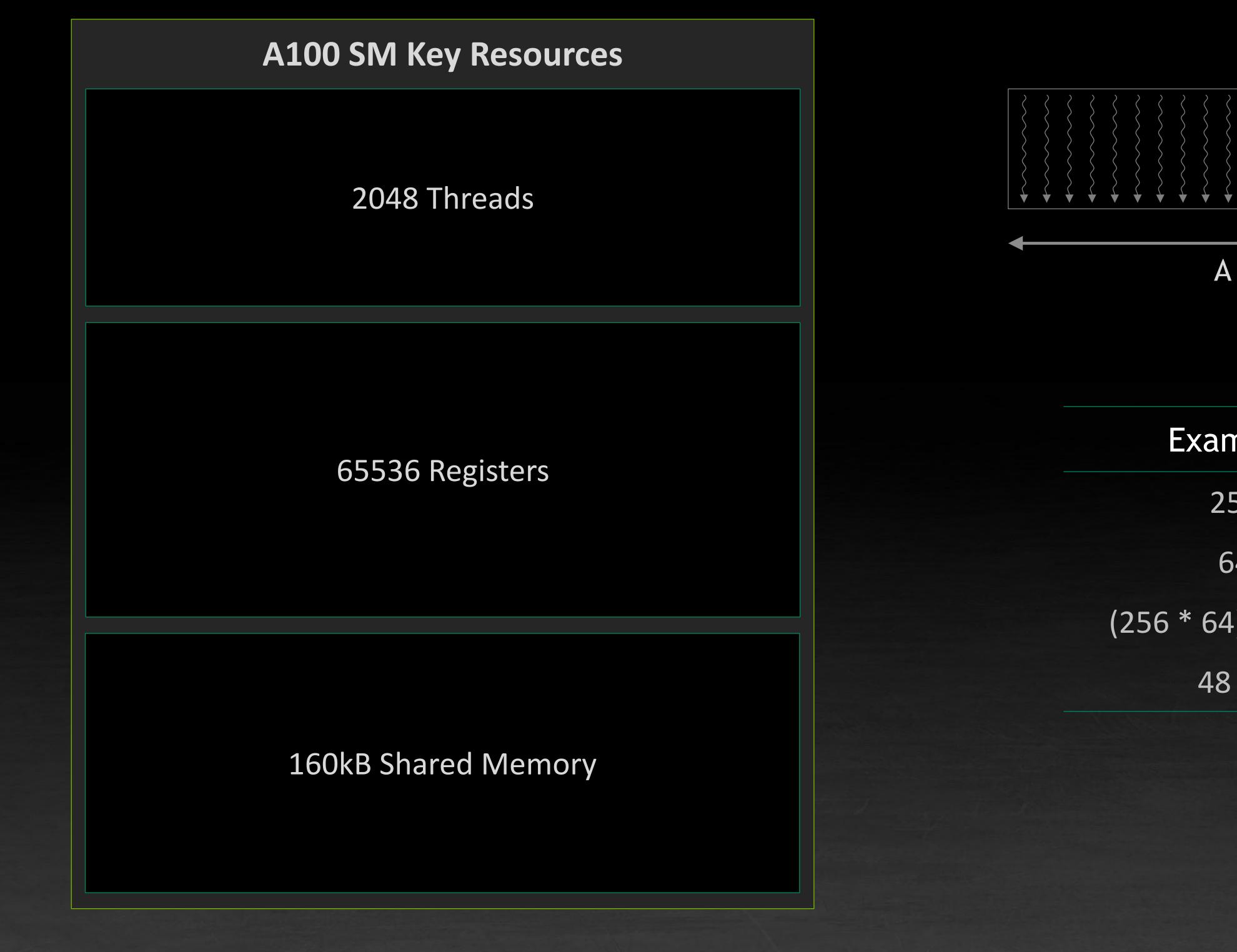
Registers are a per-thread resource, so total budget is: (threads-per-block x registers-per-thread)

ANATOMY OF A THREAD BLOCK









Thread block

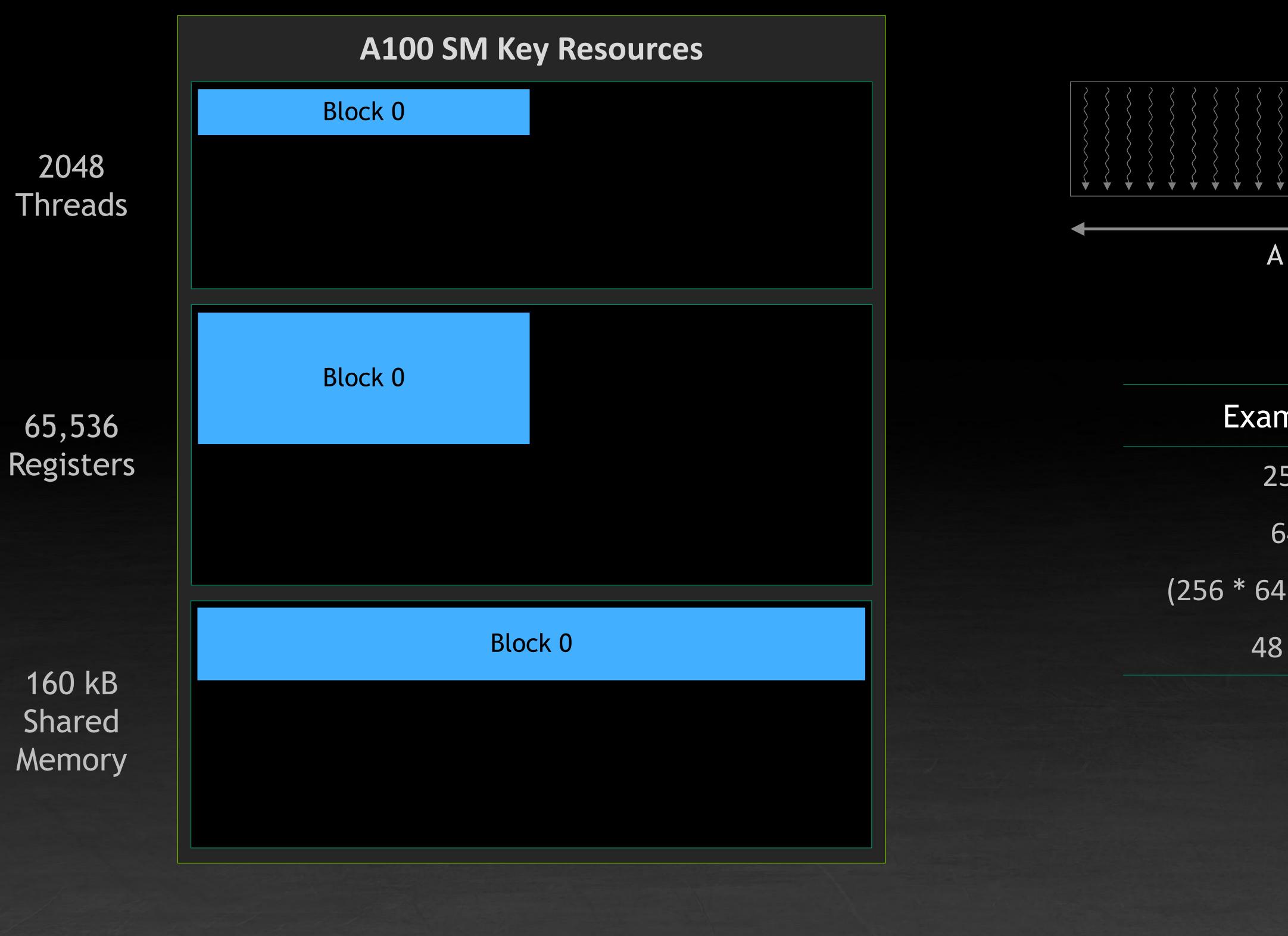
A block has a fixed number of threads always running on a single SM

Example block resource requirements

56	Threads per block
54	(Registers per thread)
I) = 16384	Registers per block
kB	Shared memory per block









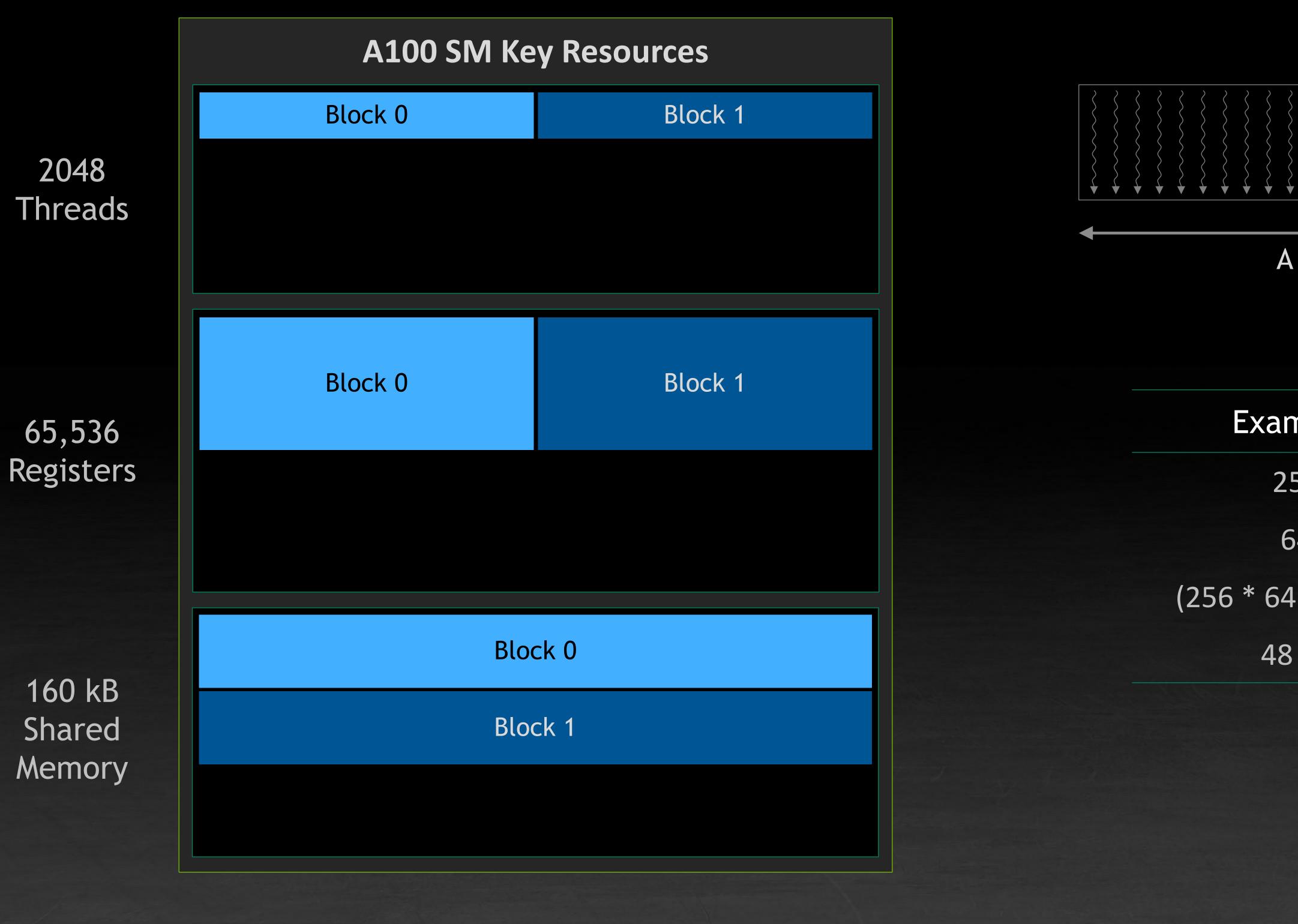
Thread block

				•	
nnla	a h		resource	requirement	C
		UUUN	I C SOUI C C	i cquii cincii c	

56	Threads per block
54	(Registers per thread)
) = 16384	Registers per block
kB	Shared memory per block









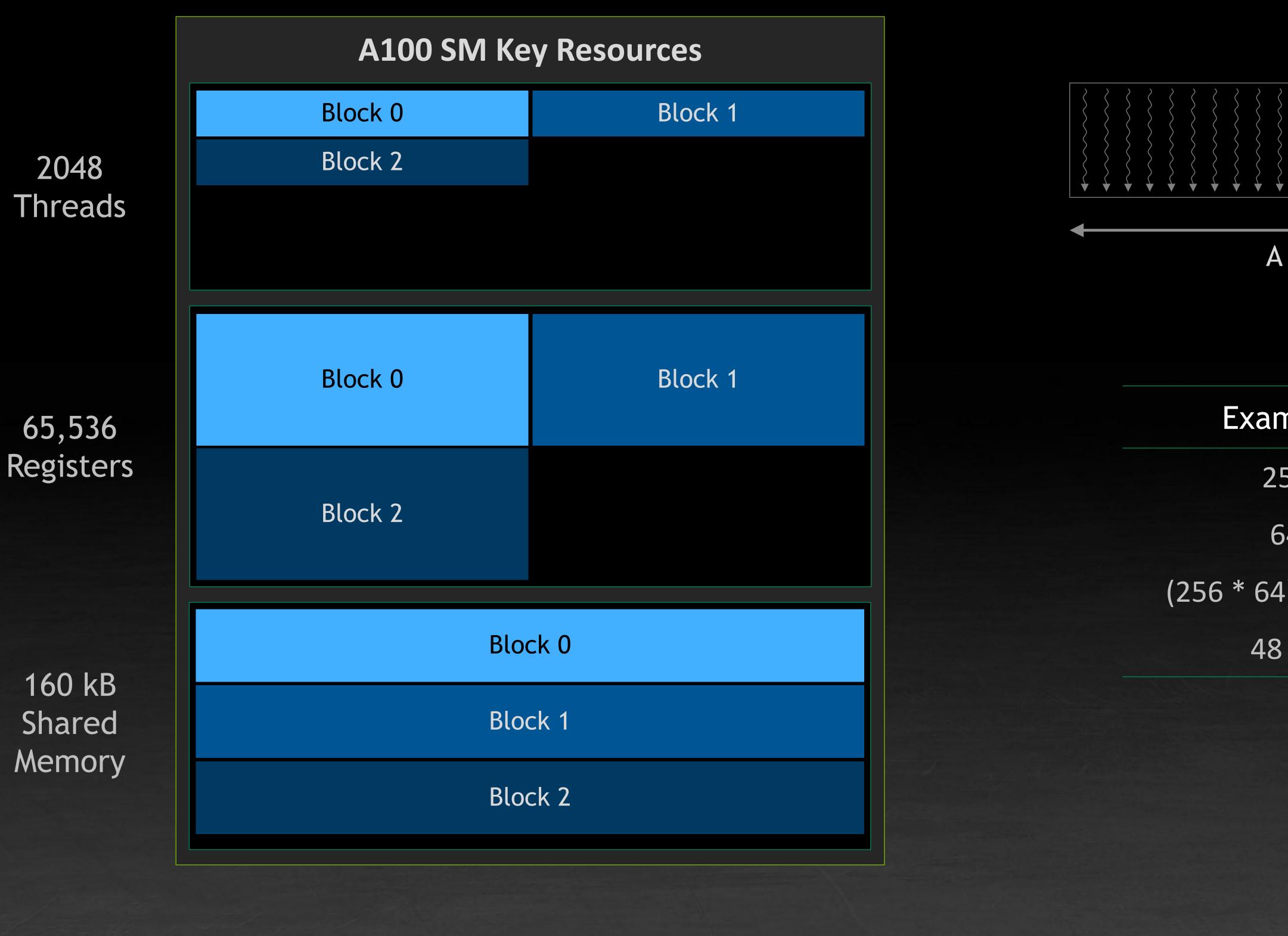
Thread block

				•	
nnla	a h		resource	requirement	C
		UUUN	I C SOUI C C	i cquii cincii c	

56	Threads per block
54	(Registers per thread)
) = 16384	Registers per block
kB	Shared memory per block







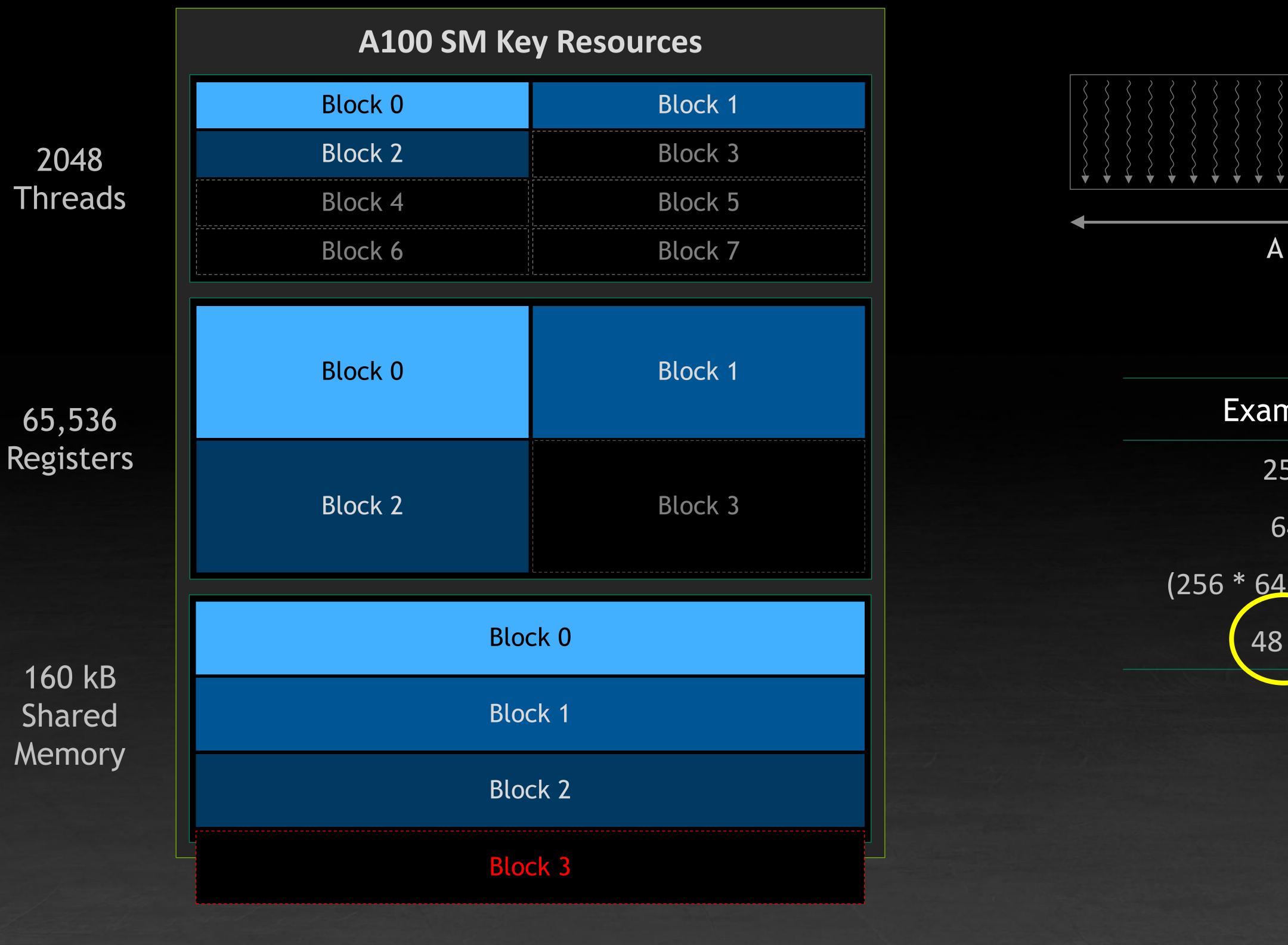


Thread block

				•	
nnla	a h		resource	requirement	C
		UUUN	I C SOUI C C	i cquii chicht	

56	Threads per block
54	(Registers per thread)
) = 16384	Registers per block
kB	Shared memory per block





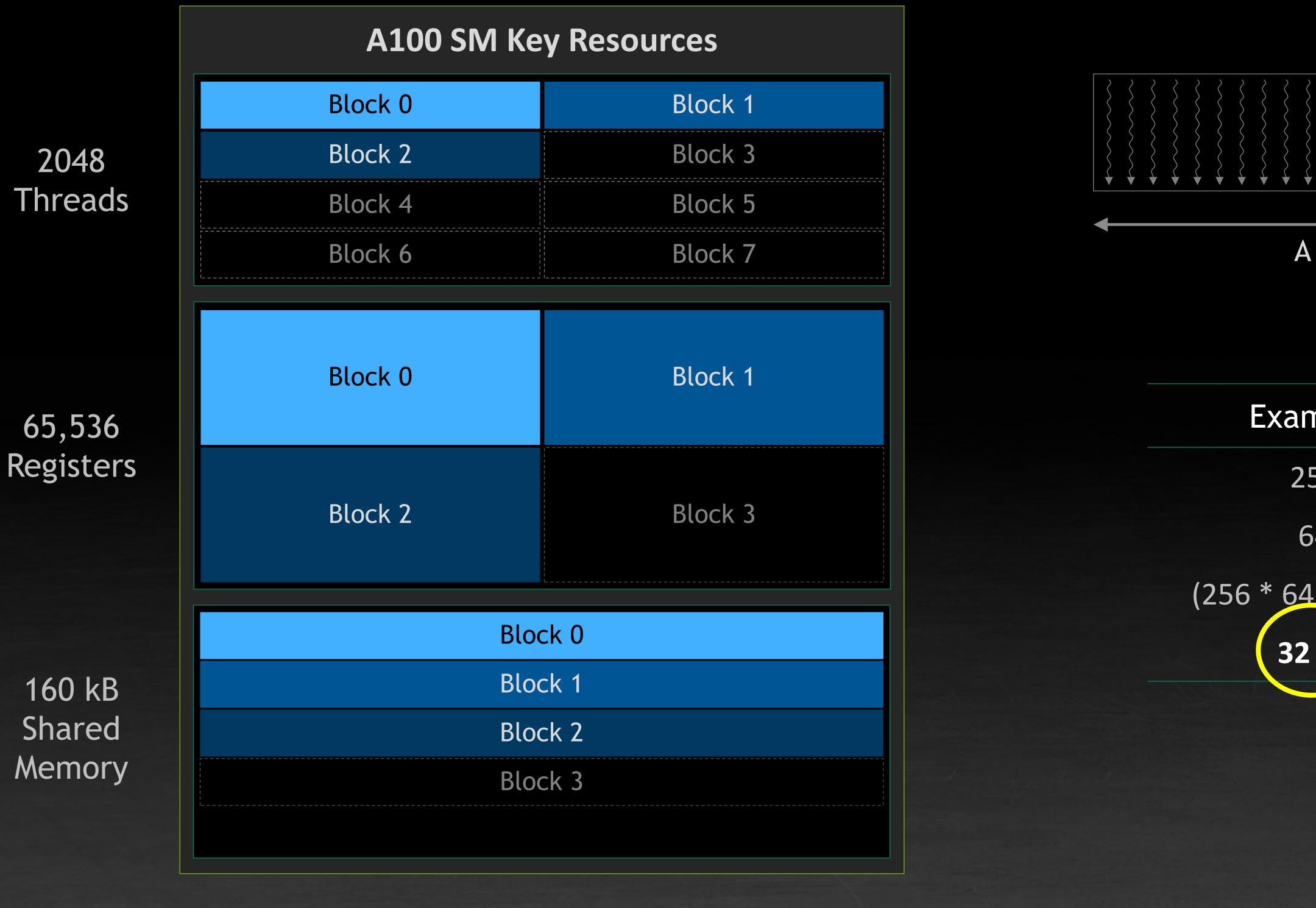


Thread block

							•		1	
nn	ρ	NC	(re	1029	irce	ren		rem	ents	
				500						

56	Threads per block
54	(Registers per thread)
1) = 16384	Registers per block
s kB	Shared memory per block







Thread block

						•	
nn	ρ	ninc	k reso	NIRCA	reall	iremer	ntc
пр			N I C S C		ICYU		

56	Threads per block
54	(Registers per thread)
1) = 16384	Registers per block
kB	Shared memory per block









Thread block

						•	
nn	ρ	ninc	k reso	NIRCA	reall	iremer	ntc
пр			N I C S C		ICYU		

56	Threads per block
54	(Registers per thread)
1) = 16384	Registers per block
kB	Shared memory per block





HOW THE GPU PLACES BLOCKS ON AN SM



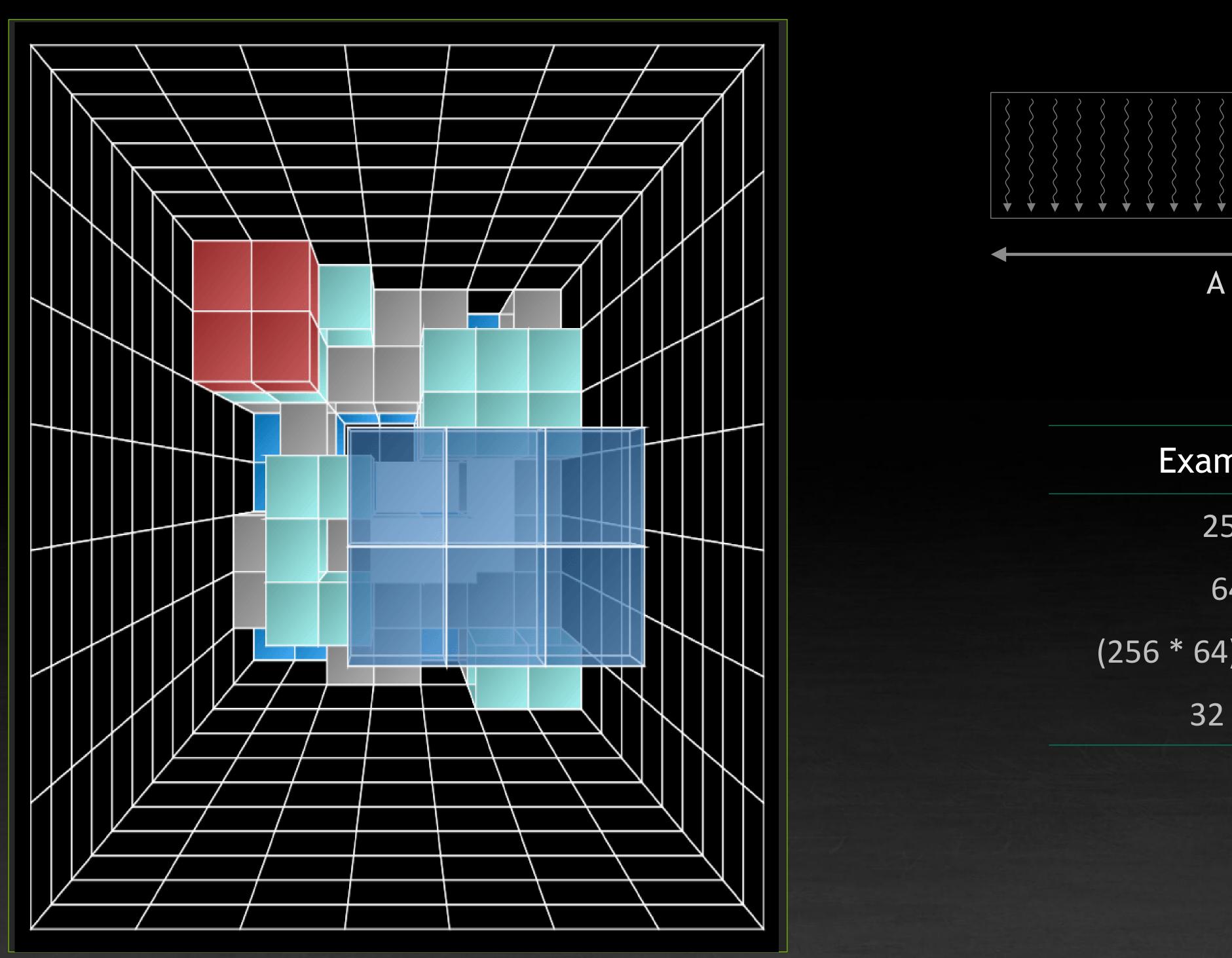
Thread block

A block has a fixed number of threads always running on a single SM

			•
nnie hi	ock reso	lirce real	lirements
inple bl		urce requ	

56	Threads per block
54	(Registers per thread)
l) = 16384	Registers per block
kB	Shared memory per block





HOW THE GPU PLACES BLOCKS ON AN SM



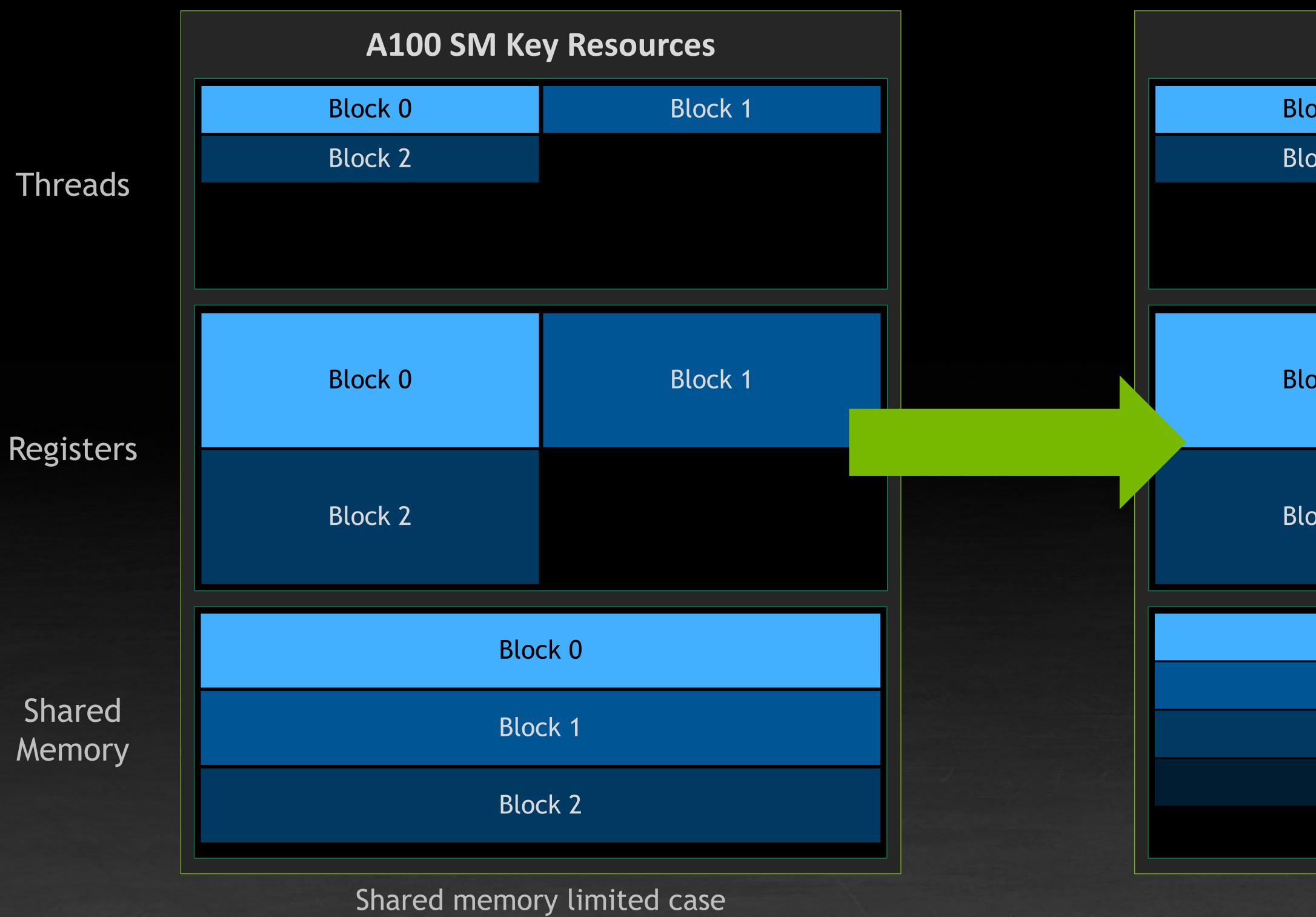
Thread block

A block has a fixed number of threads always running on a single SM

			•
nnie hi	ock reso	lirce real	lirements
inple bl		urce requ	

56	Threads per block
54	(Registers per thread)
l) = 16384	Registers per block
kB	Shared memory per block



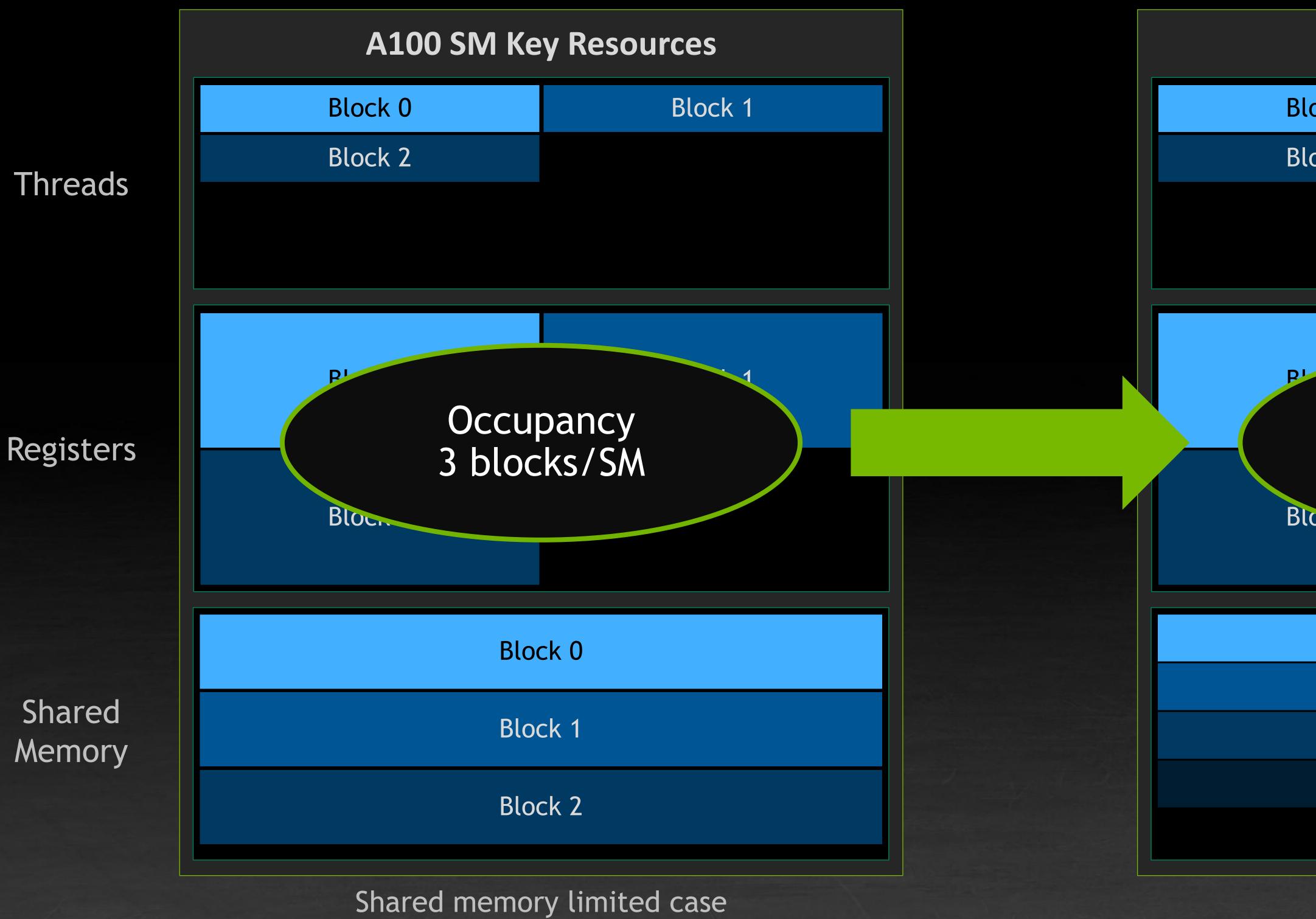


OCCUPANCY

A100 SM Key Resources		
ock 0	Block 1	
ock 2	Block 3	
ock 0	Block 1	
ock 2	Block 3	
Bloc	ck O	
Block 1		
Block 2		
Blog	ck 3	

Register limited case





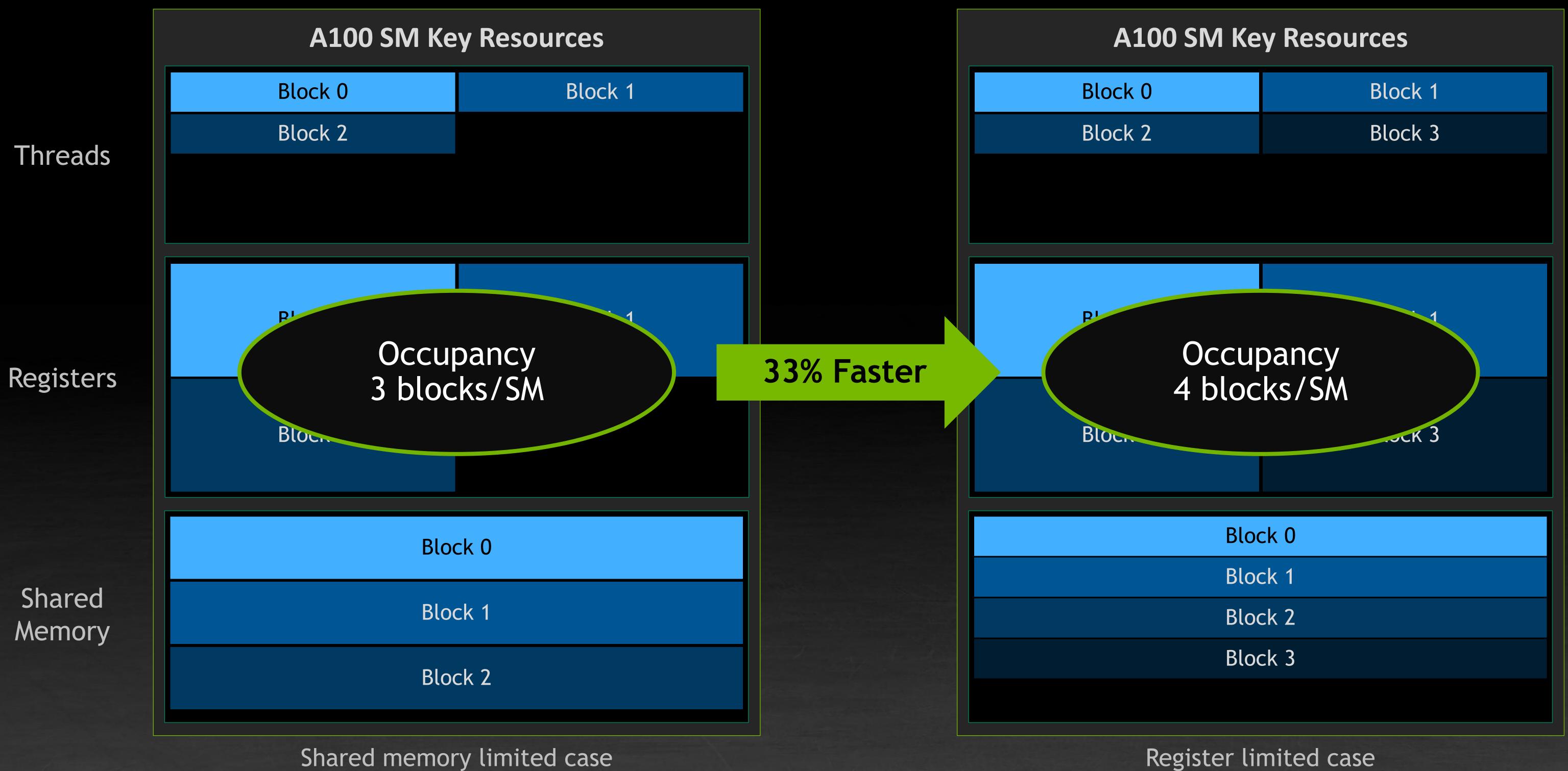
OCCUPANCY

A100 SM Key Resources		
Block 1		
Block 3		
Dancy		
ks/SM		
JOCK 3		
ck 0		
ck 1		
ck 2		
ck 3		

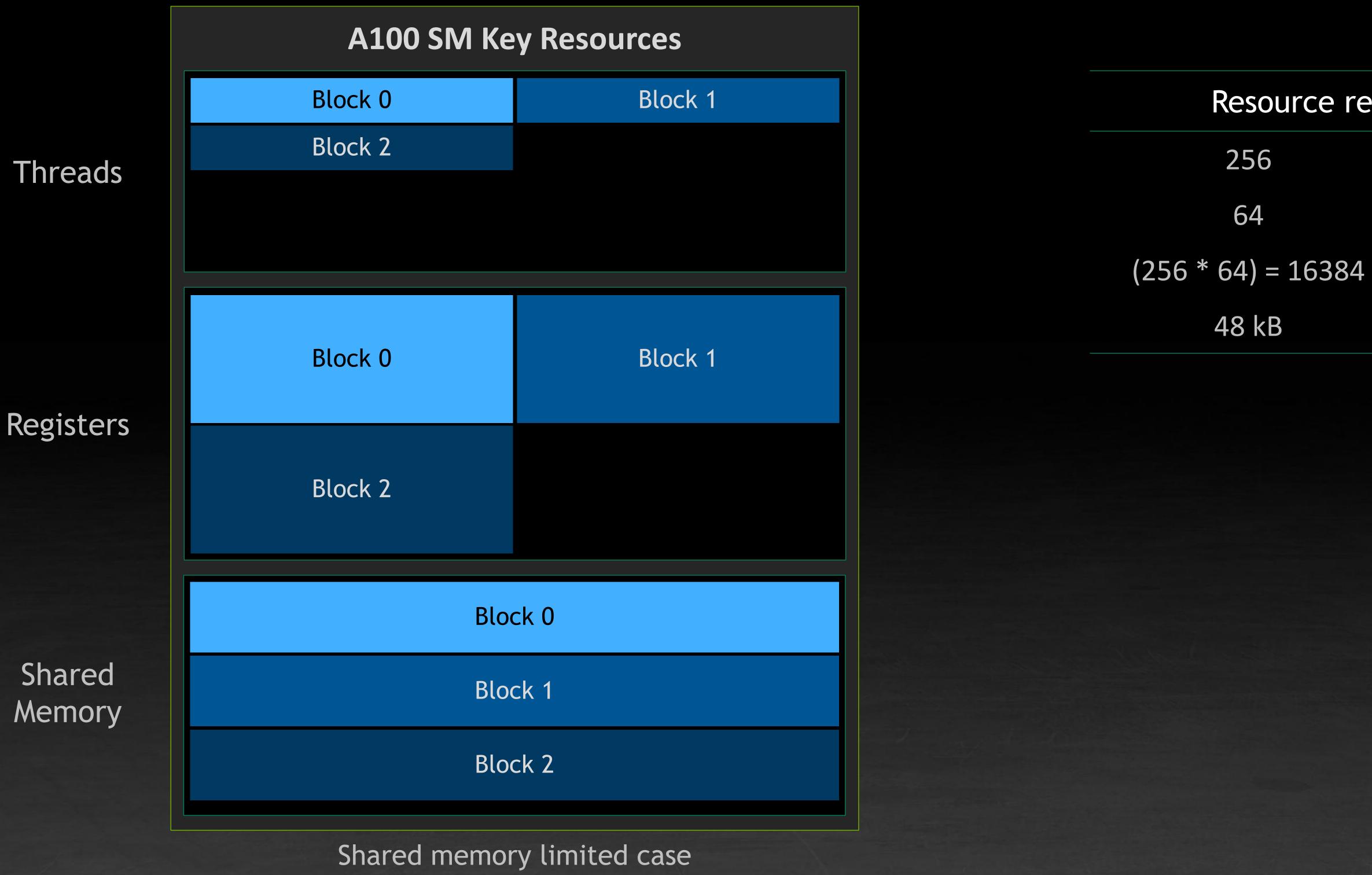
Register limited case



OCCUPANCY IS THE MOST POWERFUL TOOL FOR TUNING A PROGRAM





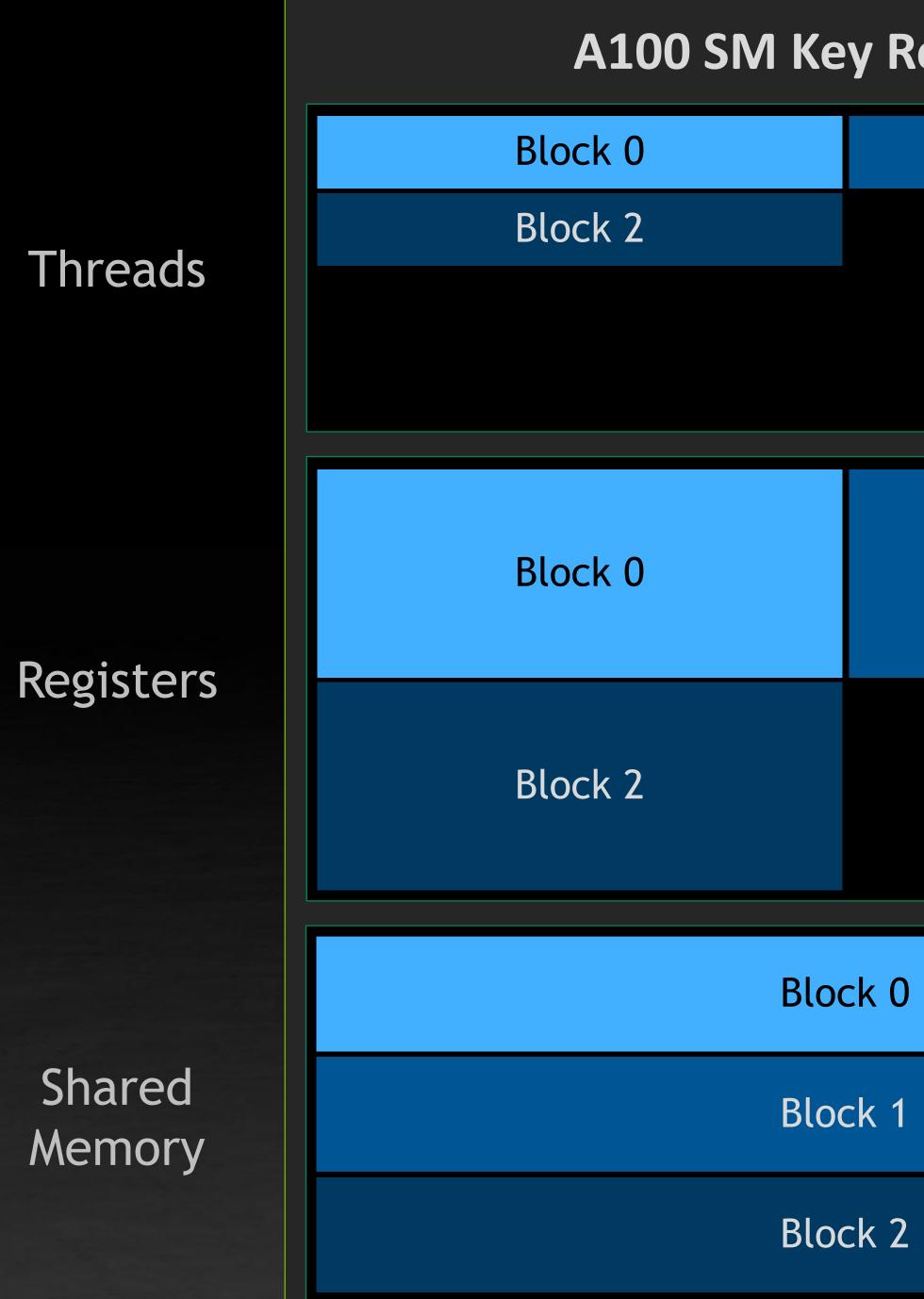


FILLING IN THE GAPS

Resource requirements (blue grid)

- Threads per block 256
- (Registers per thread) 64
 - **Registers per block**
 - Shared memory per block

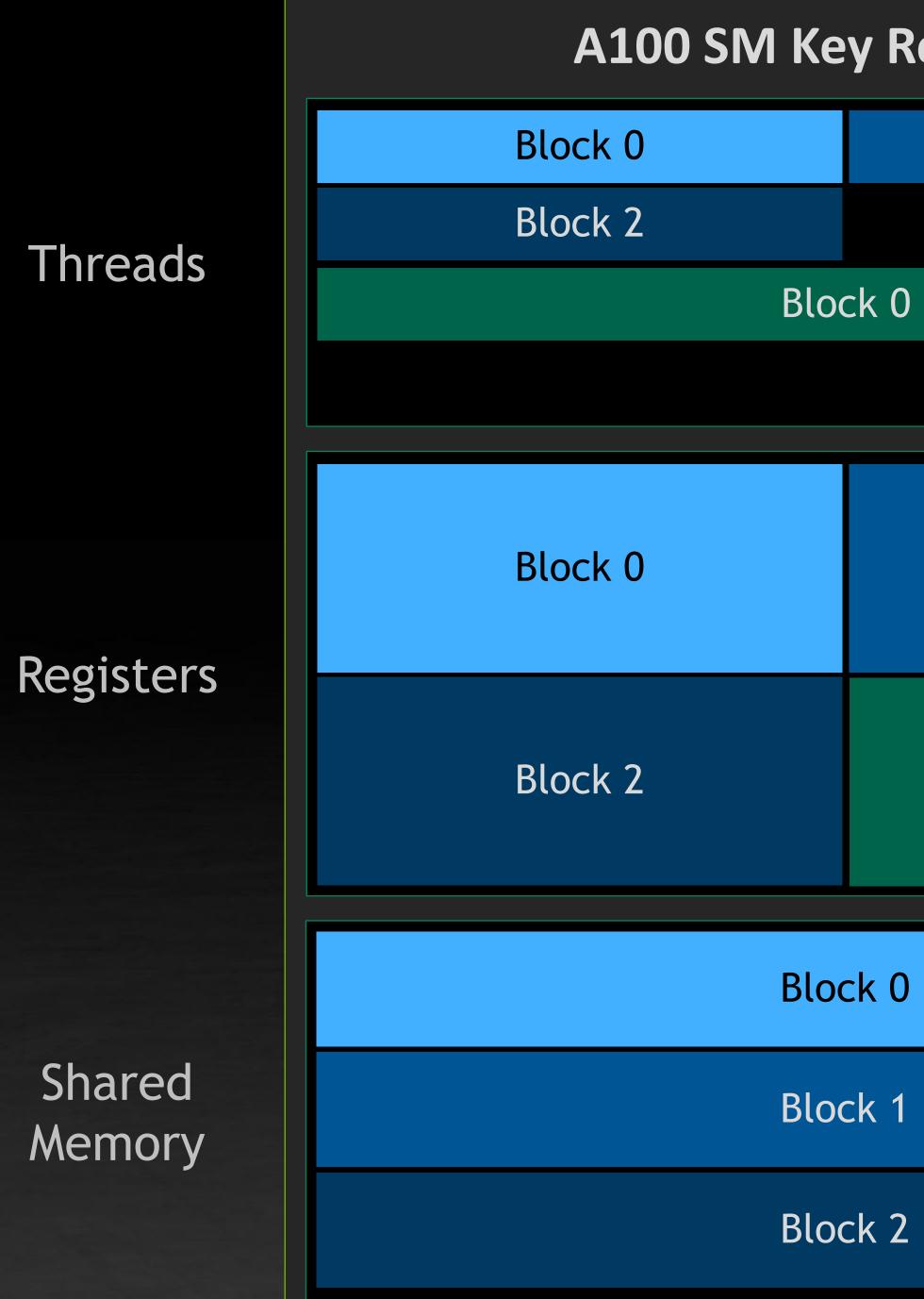




FILLING IN THE GAPS

Resource req	uirements (blue grid)
256	Threads per block
64	(Registers per thread)
(256 * 64) = 16384	Registers per block
48 kB	Shared memory per block
Resource requ	uirements (green grid)
Resource real	lirements (green grid)
512	Threads per block
32	(Registers per thread)
(512 * 32) = 16384	Registers per block

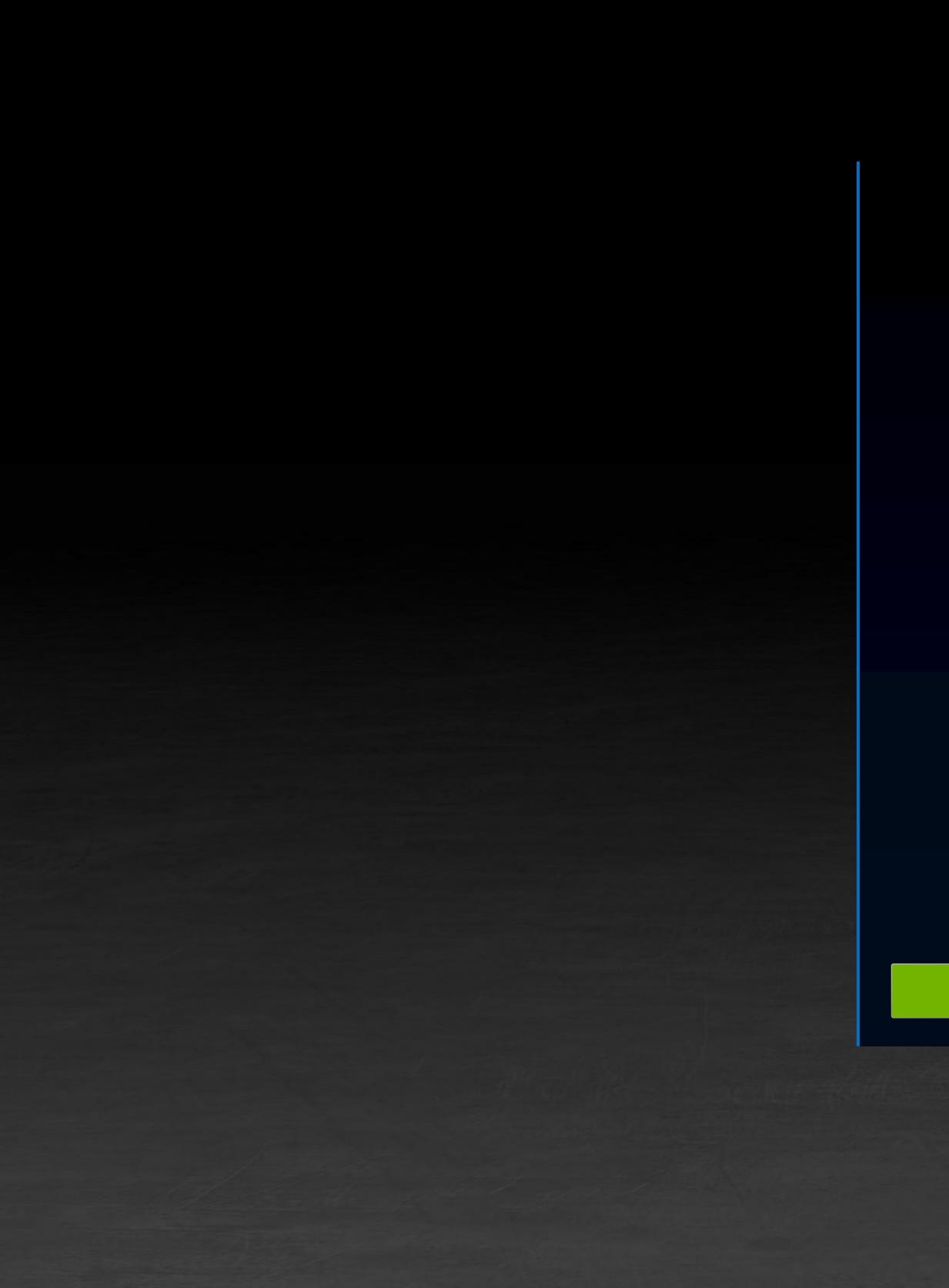




FILLING IN THE GAPS

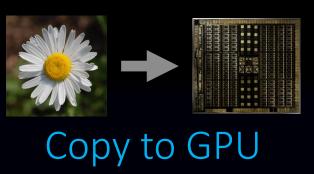
Resource req	uirements (blue grid)
256	Threads per block
64	(Registers per thread)
(256 * 64) = 16384	Registers per block
48 kB	Shared memory per block
Resource requ	uirements (green grid)
512	Threads per block
32	(Registers per thread)
(512 * 32) = 16384	Registers per block
0 kB	Shared memory per block
	256 64 (256 * 64) = 16384 48 kB





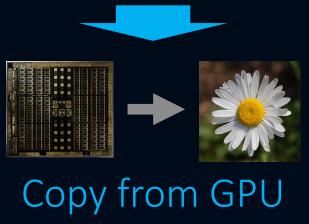
KEEPING THE GPU FULL





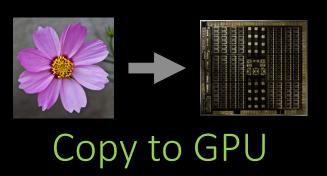


Process Flower



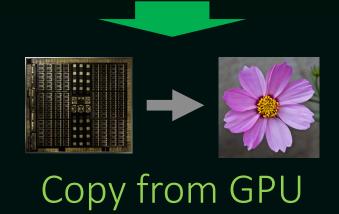
Synchronize

Stream 2





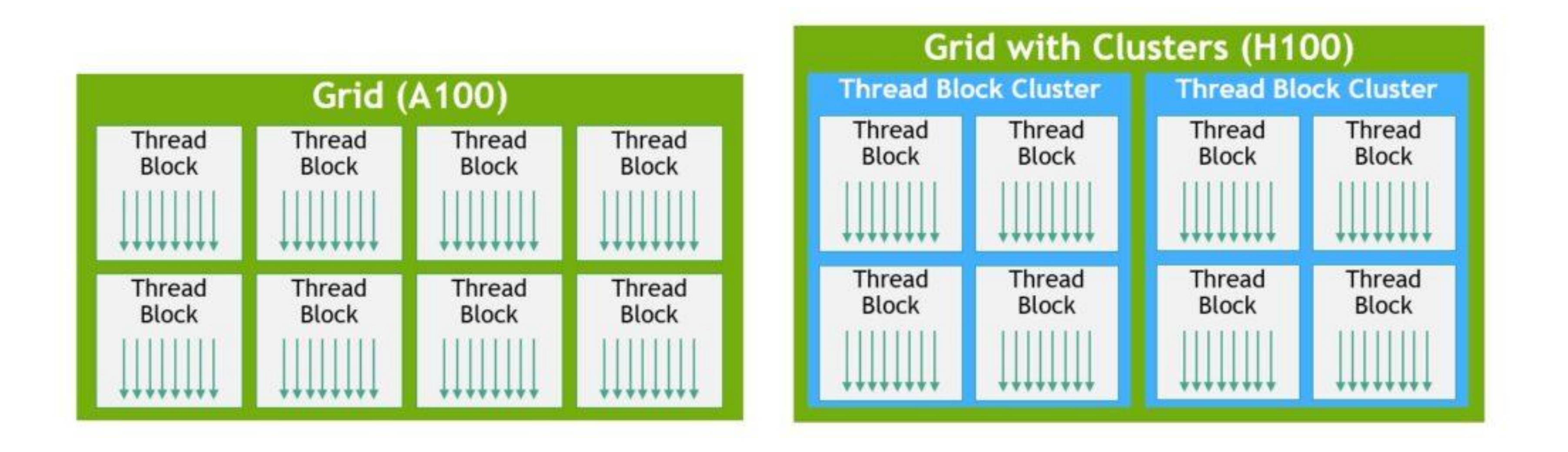
Process Flower



Synchronize

A100 SM Key Resources





Thread Block Clusters



<<< Dg, Db, Ns, S >>>

- Dg: specifies the dimension and
- Db: specifies the dimension and
- Ns: specifies the number of byte is dynamically allocated per block argument which defaults to 0
- S: specifies the associated stream argument which defaults to 0



	dim3 dim3
size of the grid	<i>, ,</i> , , ,
size of each block	// ł
es in shared memory that k; Ns is an optional	kerr auto
m; S is an optional	
	asse

- 3 gridDim = { gX, gY, gZ }; 3 blockDim = { bX, bY, bZ };
- how to specify cluster dimensions??
- nel<<< gridDim, blockDim >>>(...);
 o err = cudaPeekAtLastError();
- ert(cudaSuccess == err);

s??);

cudaLaunchKernelEx(cfg, kern, args) <<< Dg, Db, Ns, S >>> cfg: specifies the launch configuration, including the dimension and size of the grid, the dimension and size of each block, the number of bytes in shared memory, and the associated stream kern: specifies the kernel function to launch args: specifies the arguments to the kernel function



- dim3 gridDim = $\{gX, gY, gZ\};$ dim3 blockDim = $\{ bX, bY, bZ \};$
- kernel<<< gridDim, blockDim >>>(...); auto err = cudaPeekAtLastError();
- assert(cudaSuccess == err);

cudaLaunchKernelEx(cfg, kern, args) <<< Dg, Db, Ns, S >>> cfg: specifies the launch configuration, including the dimension and size of the grid, the dimension and size of each block, the number of bytes in shared memory, and the associated stream kern: specifies the kernel function to launch args: specifies the arguments to the kernel function



cudaLaunchConfig_t cfg = { }; dim3 gridDim = { gX, gY, gZ }; dim3 blockDim = $\{ bX, bY, bZ \};$

kernel<<< gridDim, blockDim >>>(...); auto err = cudaPeekAtLastError();

cudaLaunchKernelEx(cfg, kern, args) $\leftarrow Dg, Db, Ns, S >>>$ cfg: specifies the launch configuration, including the dimension and size of the grid, the dimension and size of each block, the number of bytes in shared memory, and the associated stream kern: specifies the kernel function to launch args: specifies the arguments to the kernel function



assert(cudaSuccess == err);

cudaLaunchConfig_t cfg = { }; dim3 cfg.gridDim = { gX, gY, gZ }; dim3 cfg.blockDim = { bX, bY, bZ };

kernel<<< gridDim, blockDim >>>(...); auto err = cudaPeekAtLastError();

cudaLaunchKernelEx(cfg, kern, args) <<< Dg, Db, Ns, S >>>

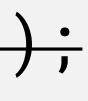
- cfg: specifies the launch configuration, including the dimension and size of the grid, the dimension and size of each block, the number of bytes in shared memory, and the associated stream
- kern: specifies the kernel function to launch
- args: specifies the arguments to the kernel function



cudaLaunchConfig_t cfg = { }; dim3 cfg.gridDim = { gX, gY, gZ }; dim3 cfg.blockDim = { bX, bY, bZ };

// what about cluster dimensions??

kernel<<< gridDim, blockDim >>>(...); auto err = cudaPeekAtLastError(); auto err = cudaLaunchKernelEx(&cfg, kernel, ...



cudaLaunchAttribute

- id: specifies the type of launch attribute
- val: specifies the value of the launch attribute; interpreted differently based on the launch attribute type



```
cudaLaunchAttribute attr;
attr.id =
  cudaLaunchAttributeClusterDimension;
attr.val.clusterDim = { cX, cY, cZ };
```

```
cudaLaunchConfig_t cfg = { };
cfg.gridDim = \{ gX, gY, gZ \};
cfg.blockDim = \{ bX, bY, bZ \};
cfg.attrs = &attr;
cfg.numAttrs = 1;
```

auto err = cudaLaunchKernelEx(&cfg, kernel, ...



cudaLaunchAttribute

- accessPolicyWindow
- cooperative
- clusterDim
- clusterSchedulingPolicyPreference
- priority
- syncPolicy



```
cudaLaunchAttribute attr;
attr.id =
  cudaLaunchAttributeClusterDimension;
attr.val.clusterDim = { cX, cY, cZ };
```

```
cudaLaunchConfig_t cfg = { };
cfg.gridDim = \{ gX, gY, gZ \};
cfg.blockDim = \{ bX, bY, bZ \};
cfg.attrs = &attr;
cfg.numAttrs = 1;
```

```
auto err = cudaLaunchKernelEx(
    &cfg, kernel, ...
```





