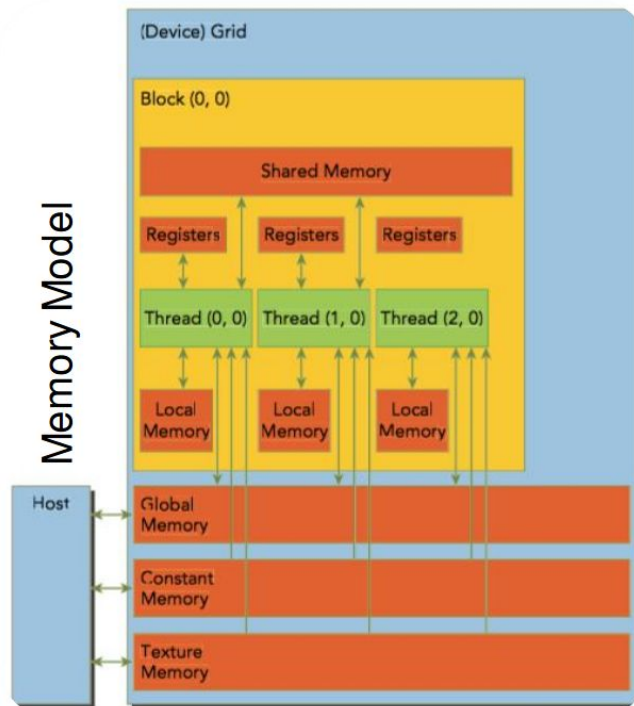


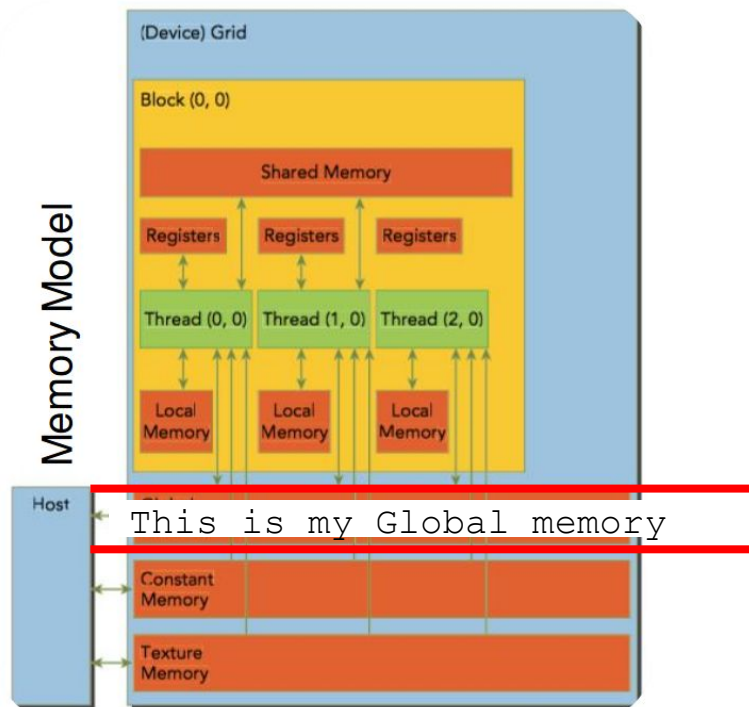
# Global memory and Coalescing

A click through example  
By Nicolas Bohm Agostini

# Memory Model



# Memory Model



---

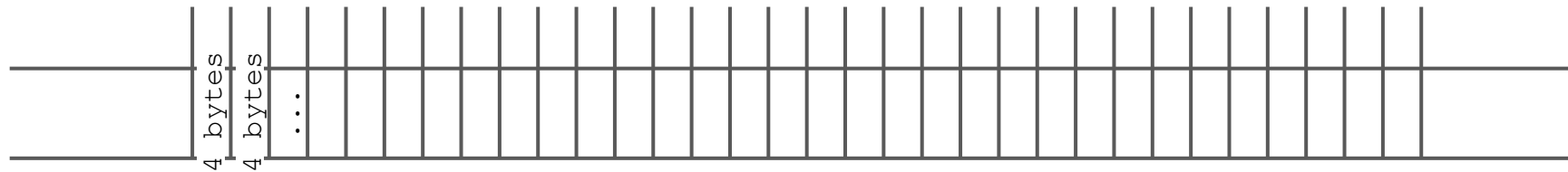
This is my Global memory

---

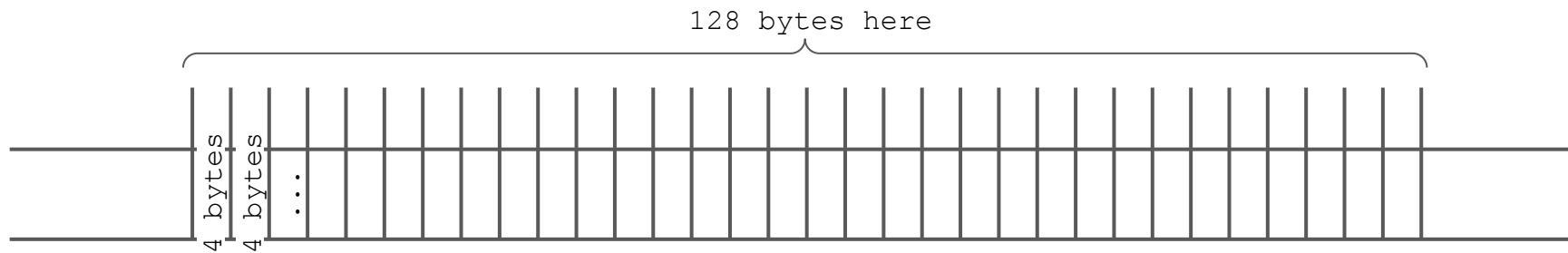
---

This is my Global memory

---

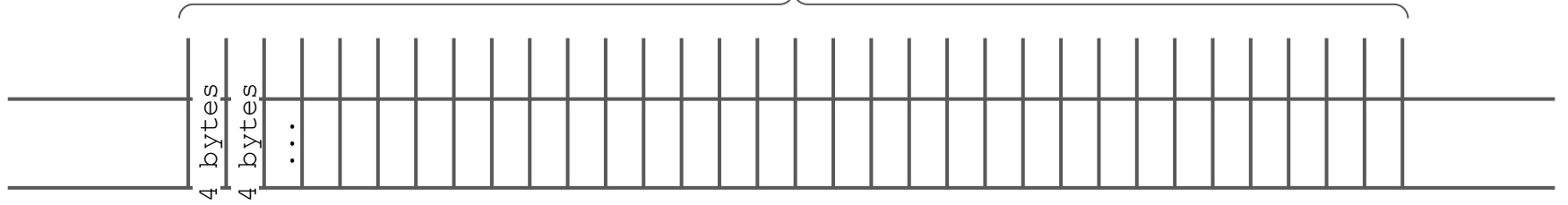


These are groups of  
4 bytes



These are groups of  
4 bytes

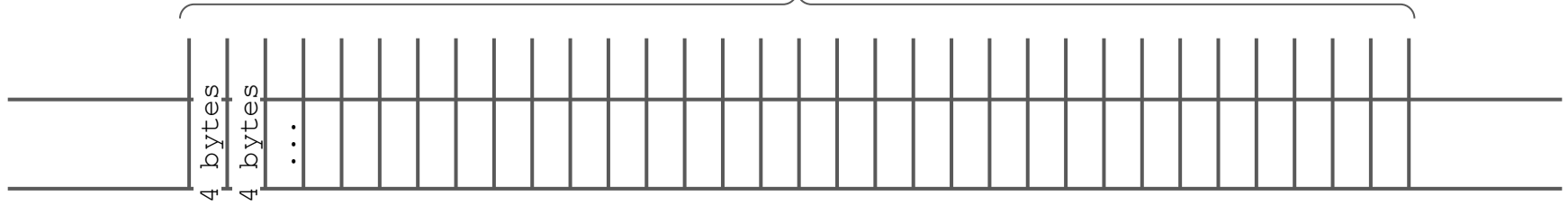
128 bytes here. 32 Groups of 4 bytes



These are groups of  
4 bytes

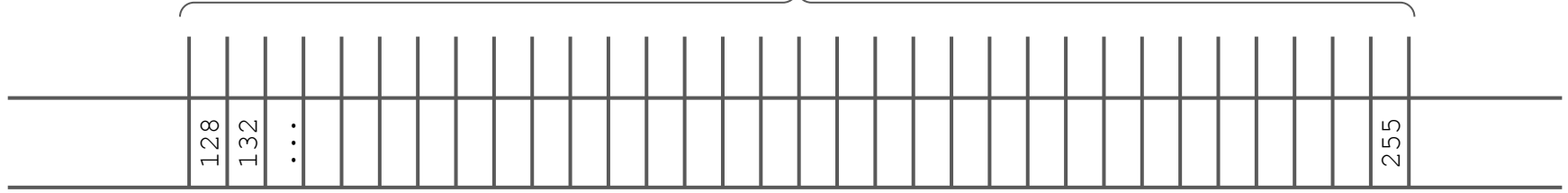


128 bytes here. 32 Groups of 4 bytes



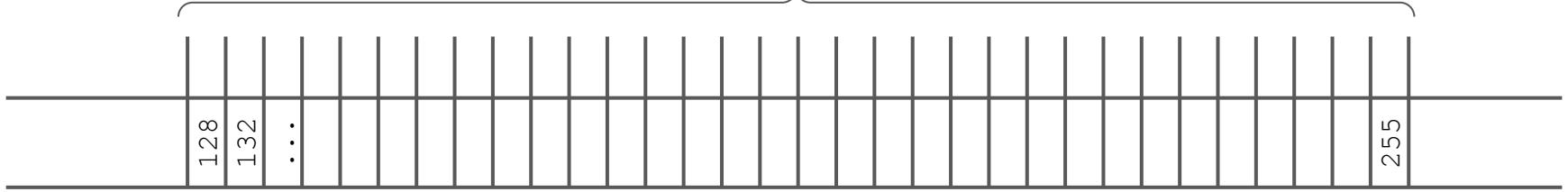
These are groups of  
4 bytes

128 bytes here. 32 Groups of 4 bytes

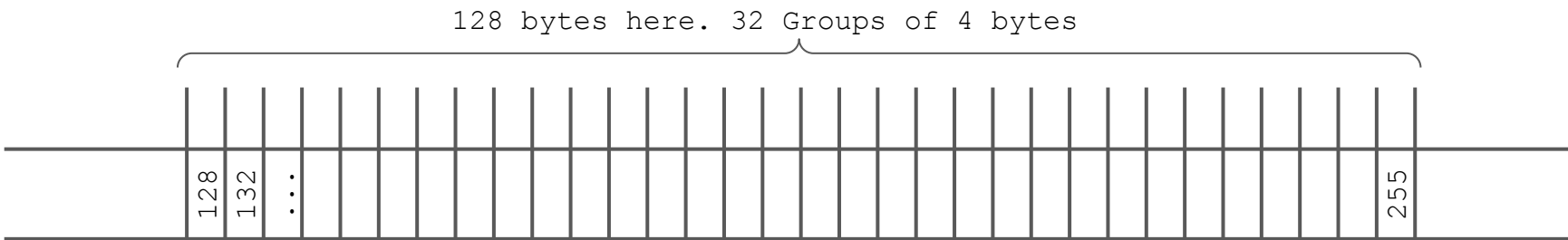


This chunk of data got allocated starting on byte 128 of my  
Global Memory

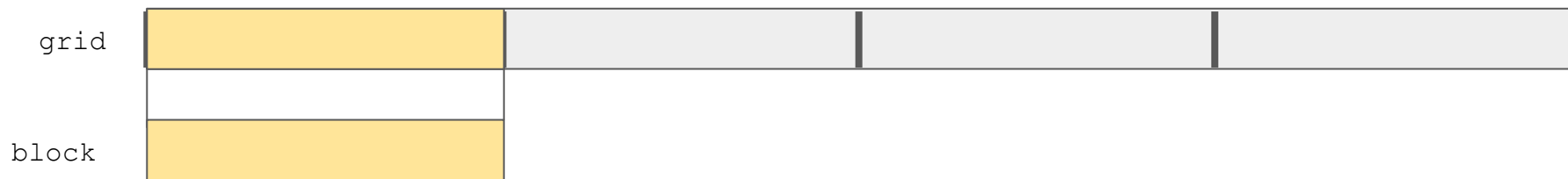
128 bytes here. 32 Groups of 4 bytes



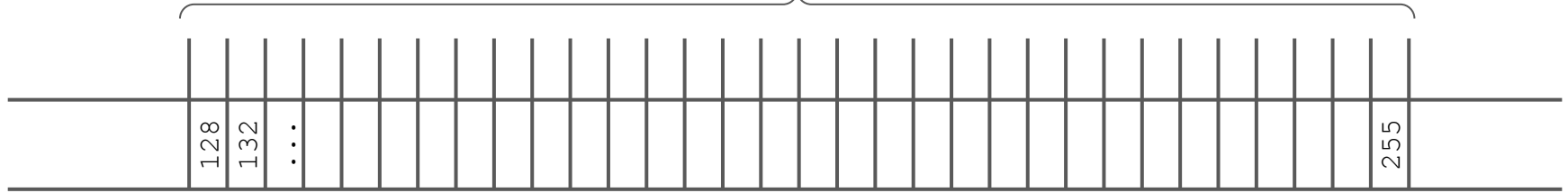
```
__device__ void add1(int* a, int vector_size) {  
    id = blockIdx.x*blockDim + threadIdx.x;  
  
    if (id<vector_size)  
        a[id]=a[id]+1;  
}  
  
int main() {  
    // do setup  
  
    add1<<<4,256>>>(dev_a,vector_size);  
  
    // do cleanup  
}
```



```
int main() {  
    // do setup  
  
    add1<<<4,256>>>(dev_a,vector_size);  
  
    // do cleanup  
}
```



128 bytes here. 32 Groups of 4 bytes



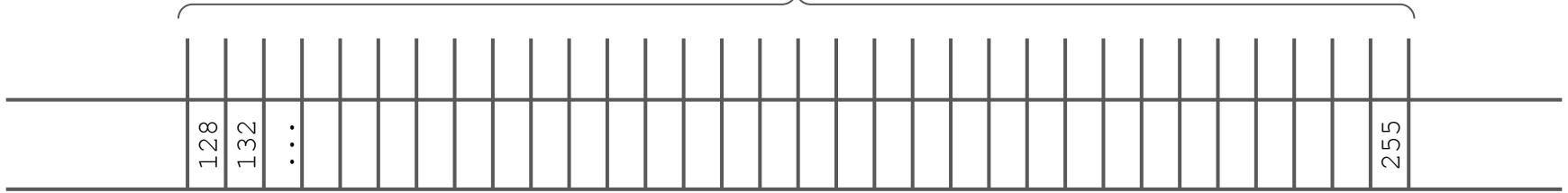
```
int main() {  
    // do setup  
  
    add1<<<4,256>>>(dev_a,vector_size);  
  
    // do cleanup  
}
```

...

block

...

128 bytes here. 32 Groups of 4 bytes



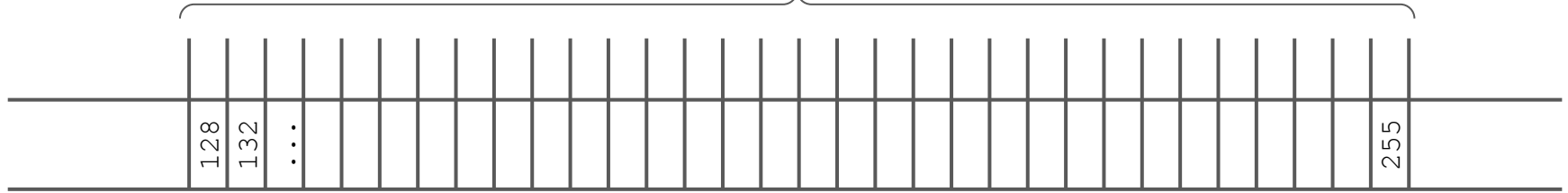
```
int main() {  
    // do setup  
  
    add1<<<4,256>>>(dev_a,vector_size);  
  
    // do cleanup  
}
```

...

block with 256/32 warps

...

128 bytes here. 32 Groups of 4 bytes



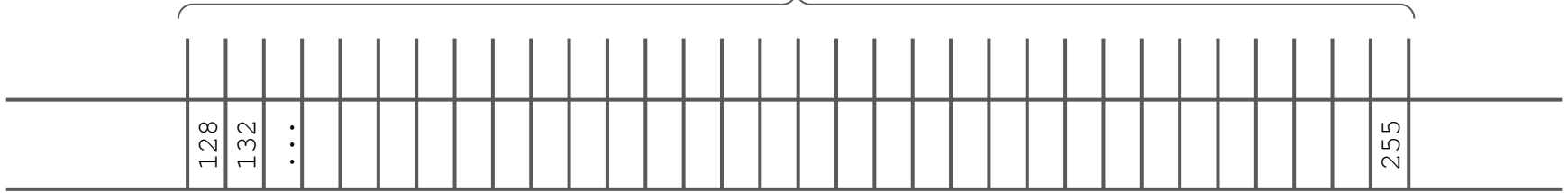
```
int main() {  
    // do setup  
  
    add1<<<4,256>>>(dev_a,vector_size);  
  
    // do cleanup  
}
```

...

block with 8 warps

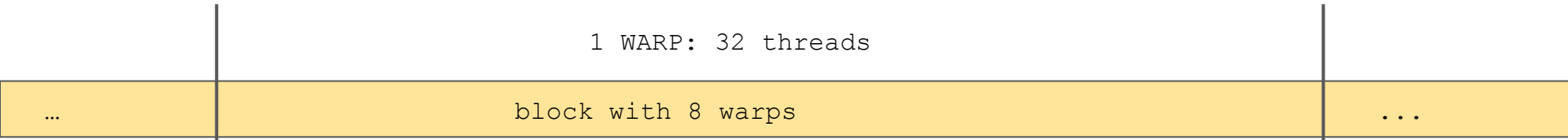
...

128 bytes here. 32 Groups of 4 bytes



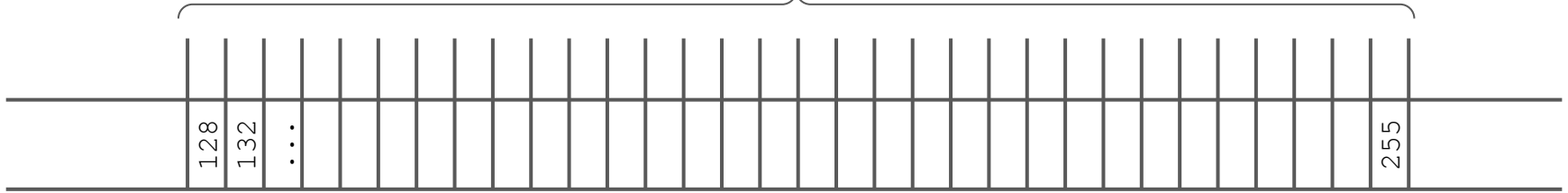
```
int main() {  
    // do setup  
  
    add1<<<4,256>>>(dev_a,vector_size);  
  
    // do cleanup  
}
```

1 WARP: 32 threads

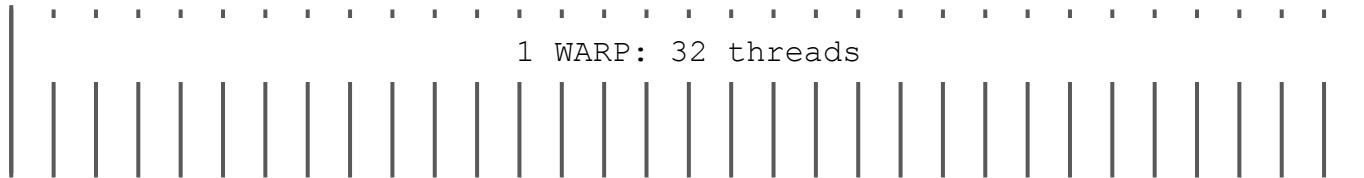




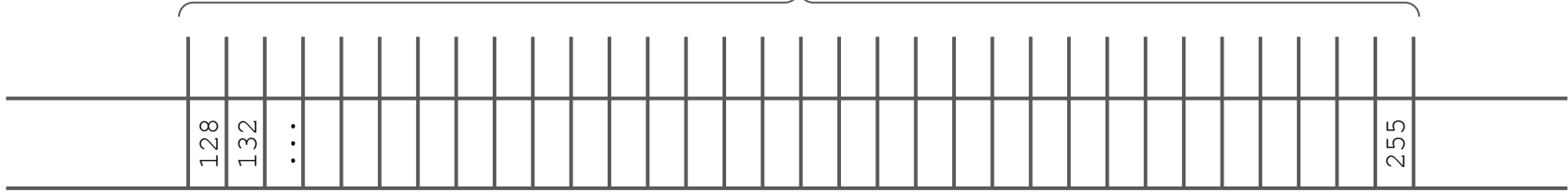
128 bytes here. 32 Groups of 4 bytes



```
int main() {  
    // do setup  
  
    add1<<<4,256>>>(dev_a,vector_size);  
  
    // do cleanup  
}
```

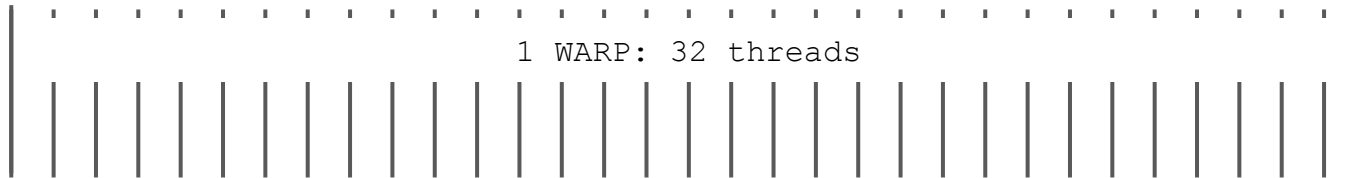


128 bytes here. 32 Groups of 4 bytes

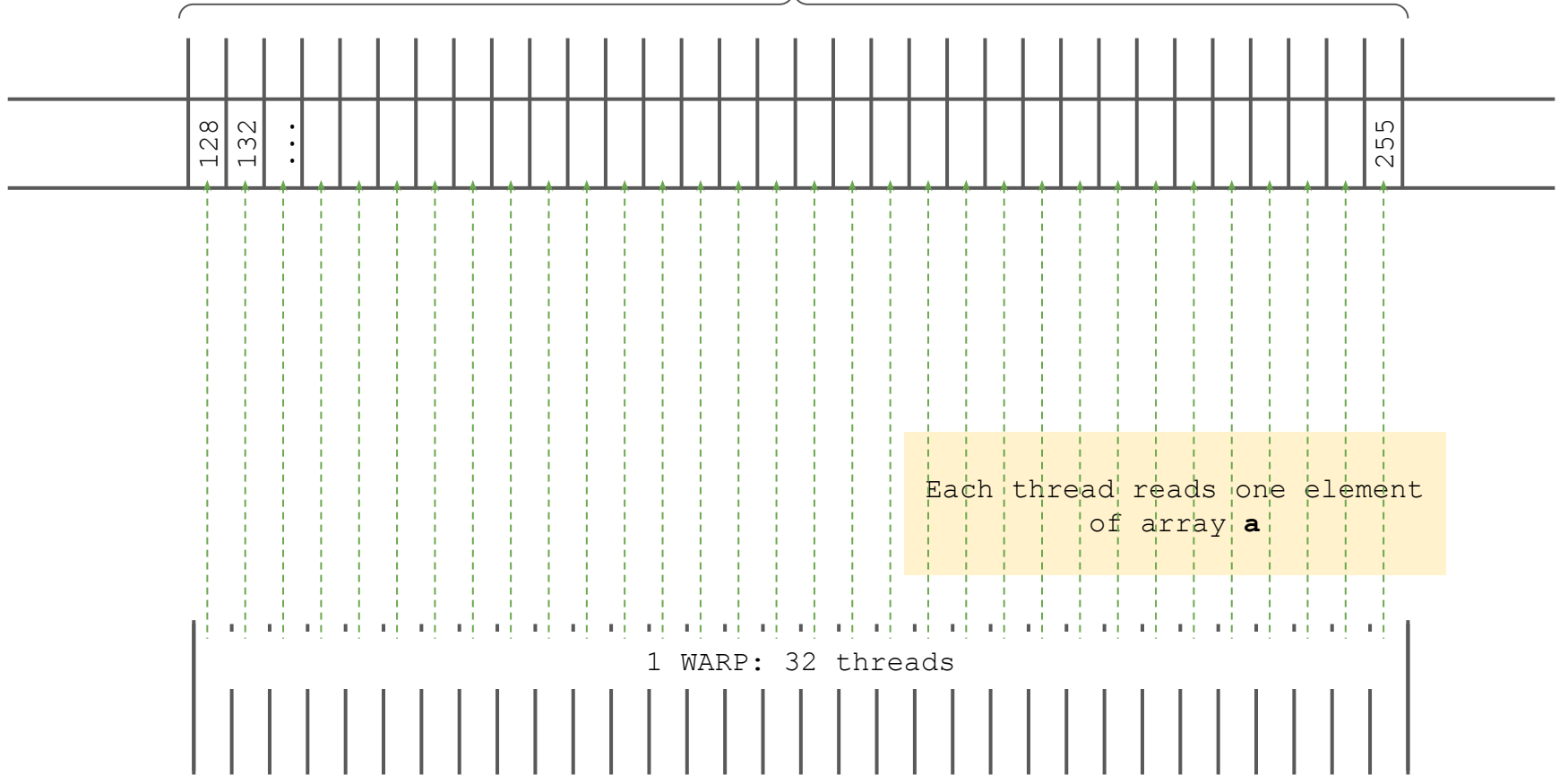


```
__device__ void add1(int* a, int vector_size) {  
    id = blockIdx.x*blockDim + threadIdx.x;  
  
    if (id<vector_size)  
        a[id]=a[id]+1;  
}
```

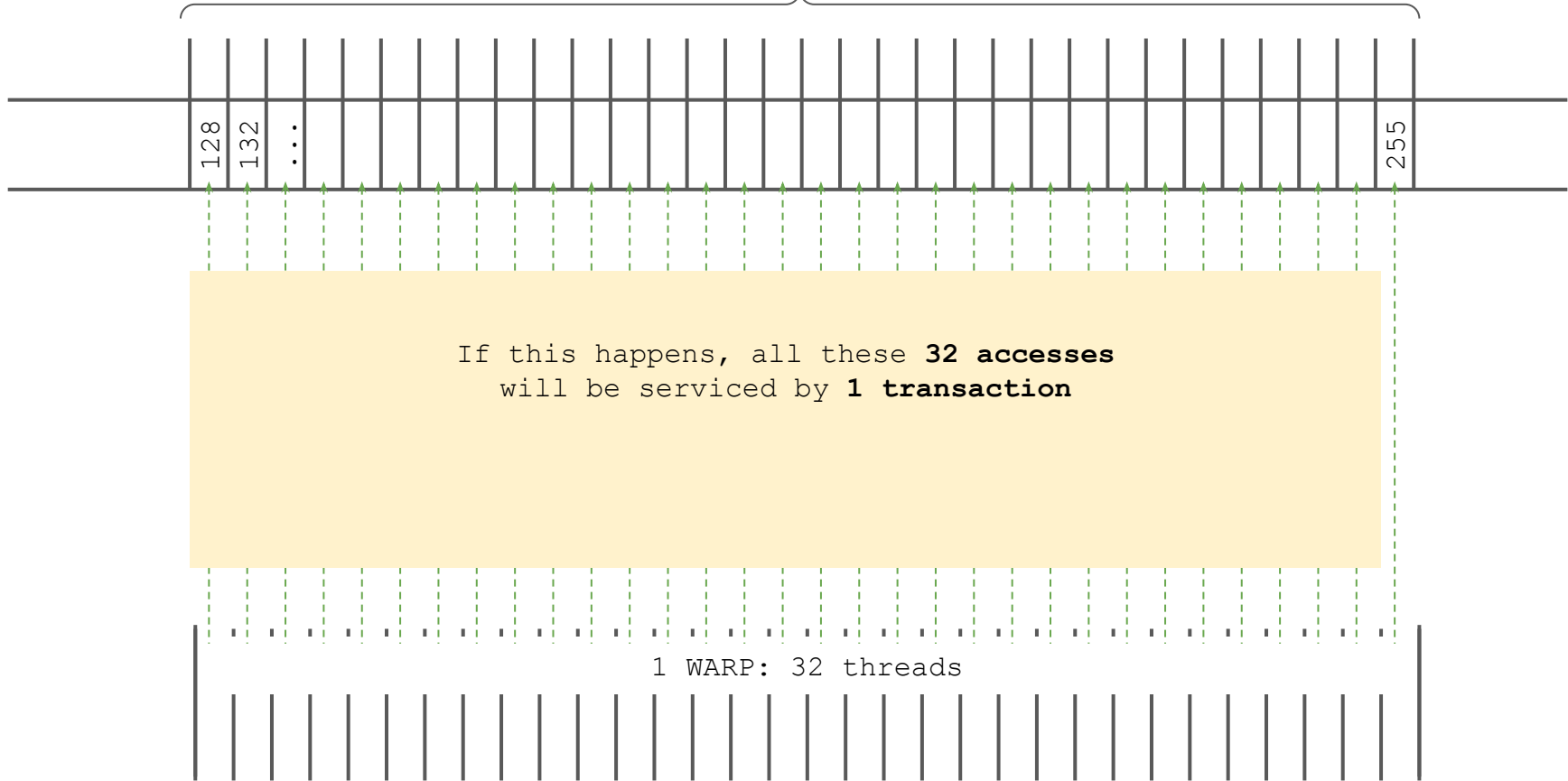
Each thread reads one element  
of array **a**



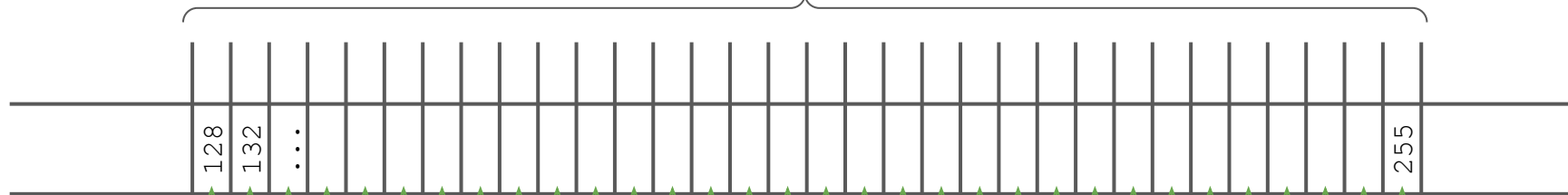
128 bytes here. 32 Groups of 4 bytes



128 bytes here. 32 Groups of 4 bytes



128 bytes here. 32 Groups of 4 bytes

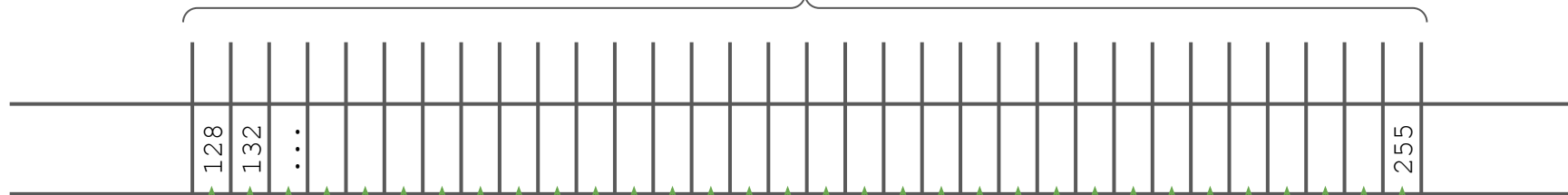


If this happens, all these **32 accesses**  
will be serviced by **1 transaction**

As opposed to 32 transactions

1 WARP: 32 threads

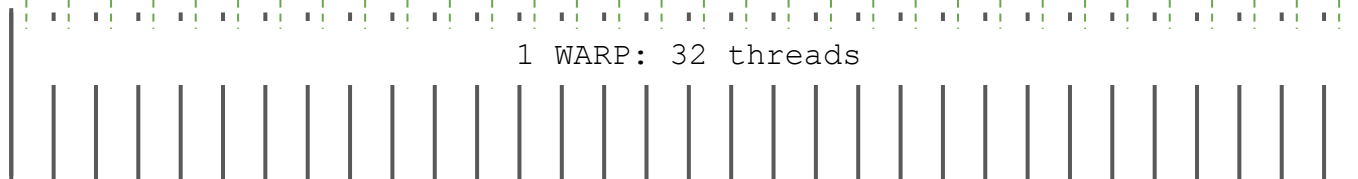
128 bytes here. 32 Groups of 4 bytes



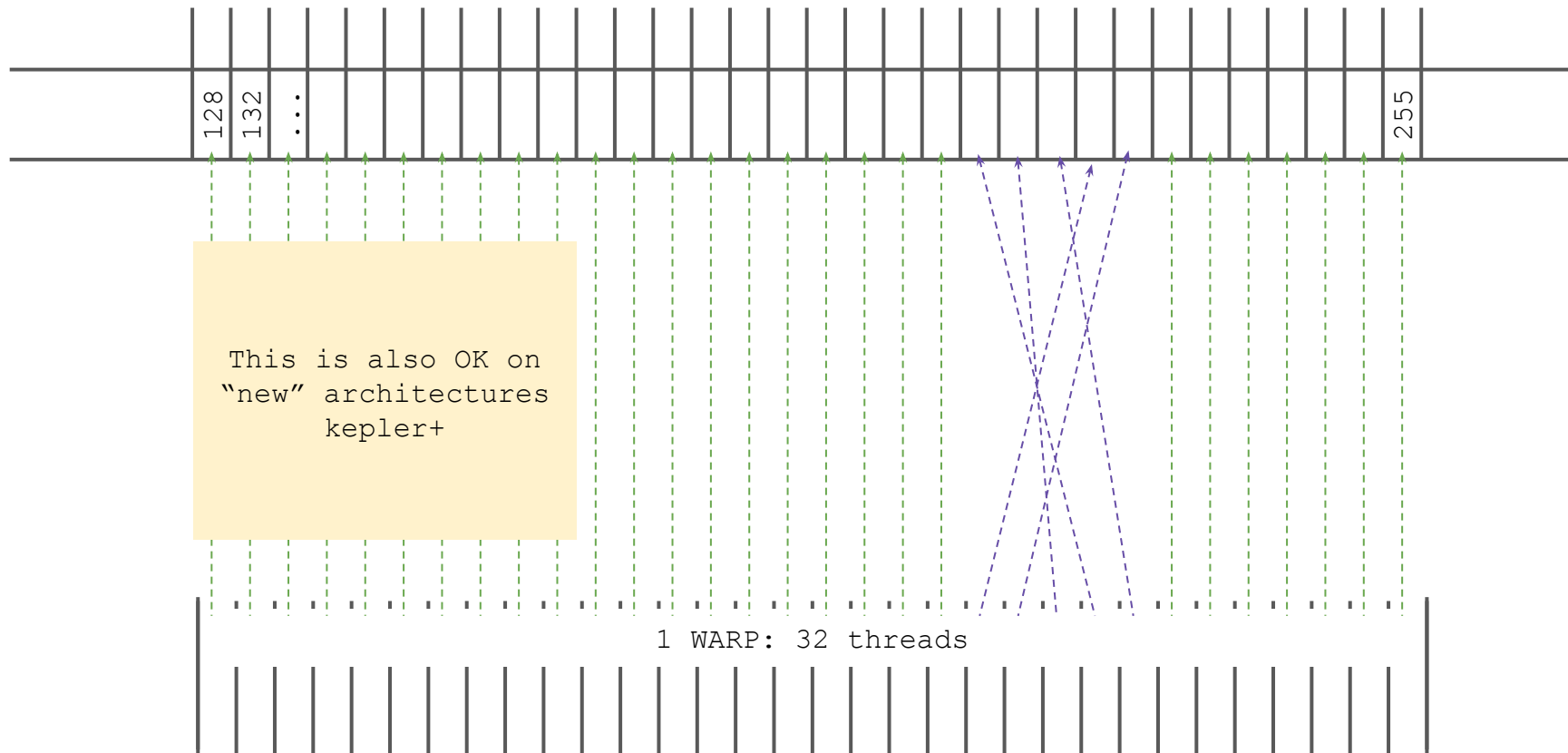
If this happens, all these **32 accesses**  
will be serviced by **1 transaction**

This is called **Memory Coalescing!**

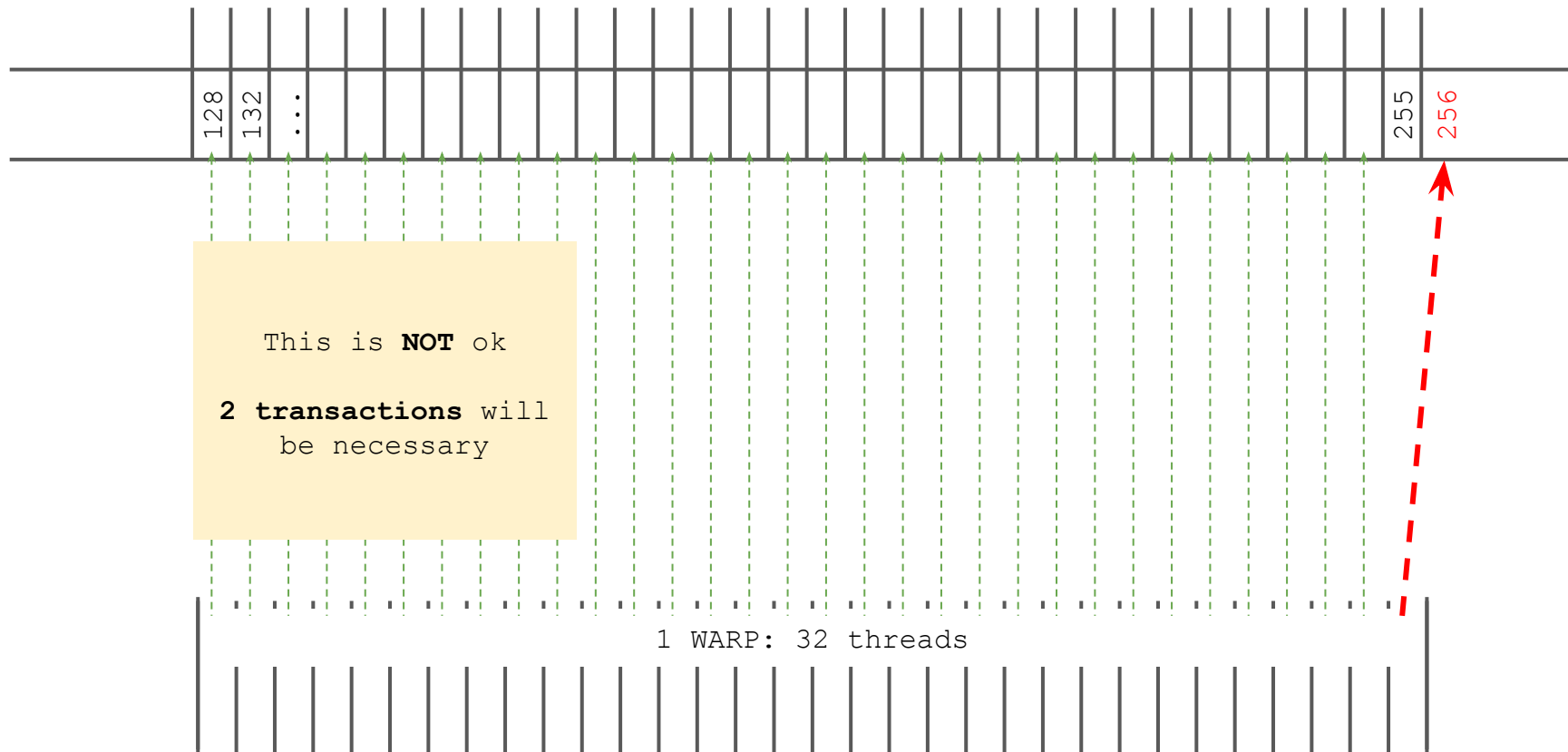
1 WARP: 32 threads



Another kernel... that has some non-contiguous access based on tid



Yet Another kernel... that has some non-contiguous access based on tid





**In your lab today!**

```
Typedef struct myCoordinates {  
    int x;  
    int y;  
    int z;  
} Coordinates;
```

```
int main () {  
    // do stuff  
  
    coordinates * a = malloc(sizeof(Coordinates)*vector_size);  
  
    // do more stuff  
}
```

**In your lab today!**

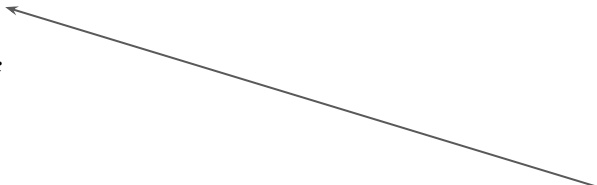
```
typedef struct myCoordinates {  
    int x;  
    int y;  
    int z;  
} Coordinates;
```

```
int main () {  
    // do stuff
```

```
    coordinates * a = malloc(sizeof(Coordinates)*vector_size);
```

```
    // do more stuff
```

```
}
```



Size of 3 integers:  
12 bytes

**In your lab today!**

```
Typedef struct myCoordinates {  
    int x;  
    int y;  
    int z;  
} Coordinates;
```

```
int main () {  
    // do stuff  
  
    coordinates * a = malloc(sizeof(Coordinates)*vector_size);  
  
    // do more stuff  
}
```

## In your lab today!

```
typedef struct myCoordinates {  
    int x;  
    int y;  
    int z;  
} Coordinates;
```

In memory 1 Coordinates  
object would look like this

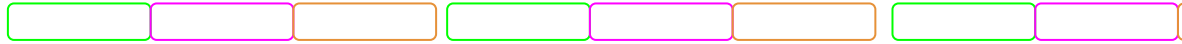


```
int main () {  
    // do stuff  
  
    coordinates * a = malloc(sizeof(Coordinates)*vector_size);  
  
    // do more stuff  
}
```

## In your lab today!

```
typedef struct myCoordinates {  
    int x;  
    int y;  
    int z;  
} Coordinates;
```

In memory Several  
Coordinates object would  
look like this

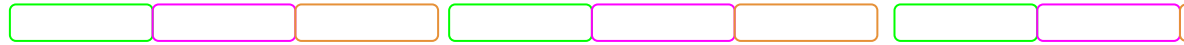


```
int main () {  
    // do stuff  
  
    coordinates * a = malloc(sizeof(Coordinates)*vector_size);  
  
    // do more stuff  
}
```

## In your lab today!

```
typedef struct myCoordinates {  
    int x;  
    int y;  
    int z;  
} Coordinates;
```

So, accessing only **x (the greens)** would be a strided access



```
int main () {  
    // do stuff  
  
    coordinates * a = malloc(sizeof(Coordinates)*vector_size);  
  
    // do more stuff  
}
```

## In your lab today!

```
typedef struct myCoordinates {  
    int x;  
    int y;  
    int z;  
} Coordinates;
```

What we want is:



```
int main () {  
    // do stuff  
  
    coordinates * a = malloc(sizeof(Coordinates)*vector_size);  
  
    // do more stuff  
}
```

Good luck! \\_(ツ)\_/