Глобальная память

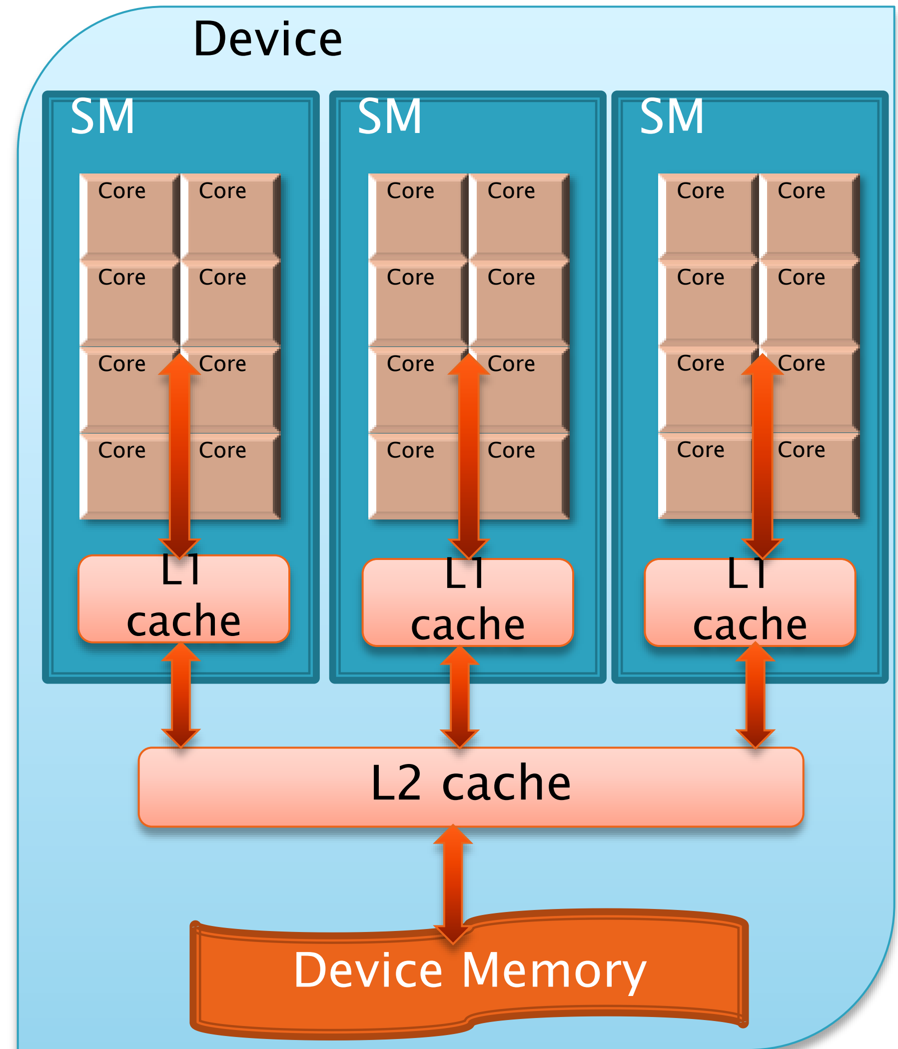
- Расположена в **DRAM GPU**
- Объём до 6Gb
  - Параметр устройства totalGlobalMem
- **Кешируется** в:
  - L2 – на устройстве

    максимальный размер 1536 KB

    Параметр устройства l2CacheSize
  - L1– на каждом мультипроцессоре

    максимальный размер 48KB

    минимальный размер 16KB



Режимы работы кеша L1

- Кеш может работать в двух режимах: 48KB и 16KB
- Переключение режимов:
  - `cudaDeviceSetCacheConfig(cudaFuncCache cacheConfig)`

    Устанавливает режим работы кеша cacheConfig для всего устройства

  - `cudaFuncSetCacheConfig ( const void* func, cudaFuncCache cacheConfig )`

    Устанавливает режим работы кеша cacheConfig для отдельного ядра

Режимы работы кеша L1

- cudaDeviceSetCacheConfig(cudaFuncCache cacheConfig)
  - Возможные режимы:
    - cudaFuncCachePreferNone – без предпочтений(по умолчанию). Выбирается последняя использованная конфигурация. Начальная конфигурация – 16KB L1
    - cudaFuncCachePreferShared: 16KB L1
    - cudaFuncCachePreferL1: 48KB L1

```
cudaFuncSetCacheConfig ( const void* func,
                         cudaFuncCache  cacheConfig )
```

  - По умолчанию – cudaFuncCachePreferNone – запускать с режимом устройства

Транзакции

- Глобальная память оптимизирована с целью увеличения полосы пропускания
  - Отдать максимум данных за одно обращение

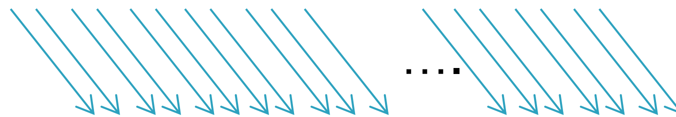
Транзакции

- Транзакция – выполнение загрузки из глобальной памяти сплошного отрезка в 128 байт, с началом кратным 128 (**naturally aligned**)

- Инструкция обращения в память выполняется одновременно для всех нитей варпа (SIMT)
  - Выполняется столько транзакций, сколько нужно для покрытия обращений всех нитей варпа
  - Если нужен один байт – все равно загрузится 128

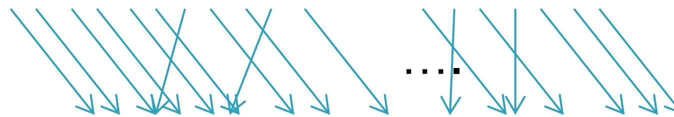
Шаблоны доступа

Обращения нитей варпа



Все обращения умещаются в одну транзакцию

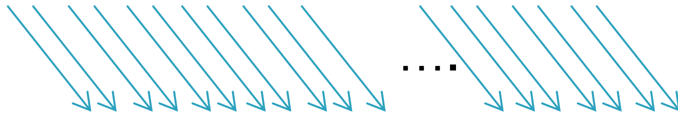
Обращения нитей варпа



Порядок не важен, главное, чтобы попадали в одну кеш-линию

Шаблоны доступа

Обращения нитей варпа



Запрашивается 128 байт, но с не выровненного адреса
2 транзакции – 256 байт

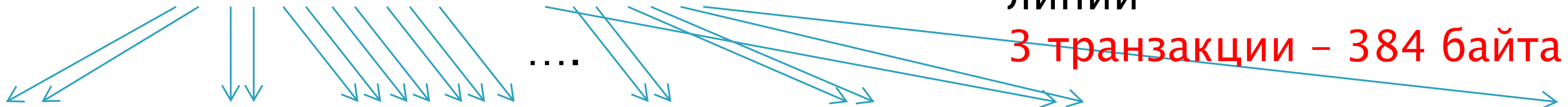
51
2

64
0

76
8

Запрашивается 128 байт, но обращения разбросаны в пределах трёх кеш-линий –
3 транзакции – 384 байта

Обращения нитей варпа



51
2

64
0

76
8

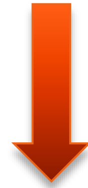
Кеш-линии

- Ядра взаимодействуют не с памятью напрямую, а с кешами
- Транзакция – выполнение загрузки кеш-линии
  - У кеша **L1** кеш-линии **128** байт, у **L2 – 32** байта, **naturally aligned**
  - Кеш грузит из памяти всю кеш-линию, даже если нужен один байт

- Можно обращаться в память минуя кеш L1
  - Транзакции будут по 32 байта

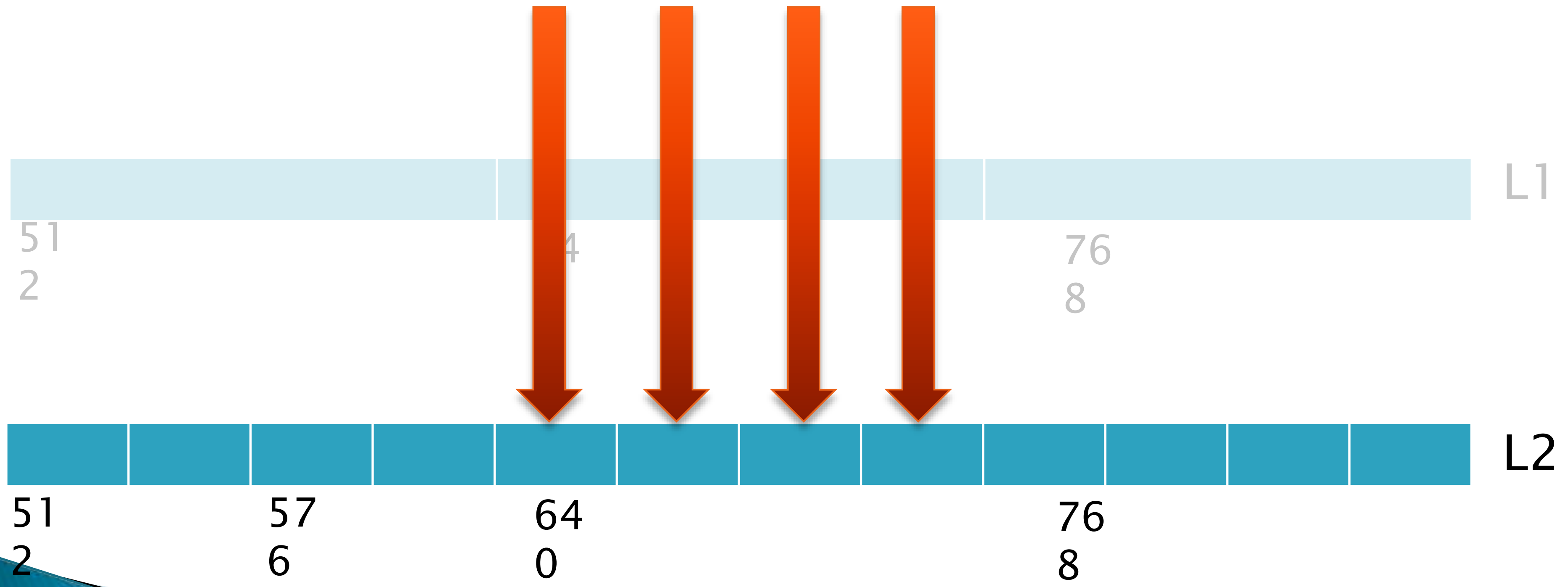
Транзакции: L1 включен

Варп



Транзакции: L1 выключен

Варп



Включение \ отключение L1

- Кеширование в L1 можно отключить **при компиляции**
  - nvcc –Xptxas –dlcm=ca

    с кешированием в L1 (по умолчанию)
  - nvcc –Xptxas –dlcm=cg

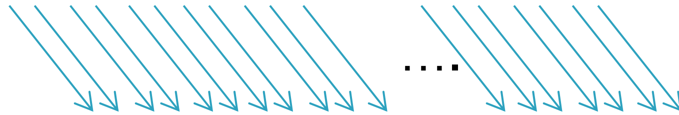
    В бинарном коде обращения в глобальную память будут транслированы в инструкции, не использующие кеш L1 при выполнении

- Различия именно на уровне бинарного кода – другие инструкции ассемблера

Шаблоны доступа: L1 выключен

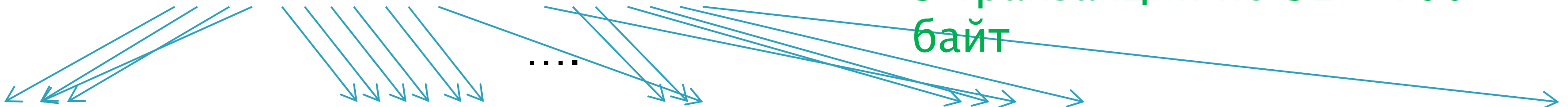
Запрашивается 128 байт, но с невыровненного адреса.
Умещаются в четыре кеш-линии L2
4 транзакции по 32 байта – 128

Обращения нитей варпа



Запрашивается 128 байт,
но обращения разбросаны
по пяти кеш-линиям L2
5 транзакций по 32 – 160 байт

Обращения нитей варпа



Вывод

- Если в ядре не используется общая память (см. далее), то заведомо стоит включить cudaFuncCachePreferL1
- Если разреженный доступ – кеширование в L1 отключаем
- В общем случае, стоит проверить производительность работы всех 4-х вариантов:
  - (-dlcm=ca, -clcm=cg)x(16KB, 48KB) !

Прикладные проблемы

Матрицы и глобальная память

Пусть транзакция – 8*4=32 байта, адрес транзакции выровнен по 32 байта

Если ширина матрицы не кратна 32 байтам – большая часть строк не выровнена

0 7 13

Матрицы и глобальная память

2 транзакции!

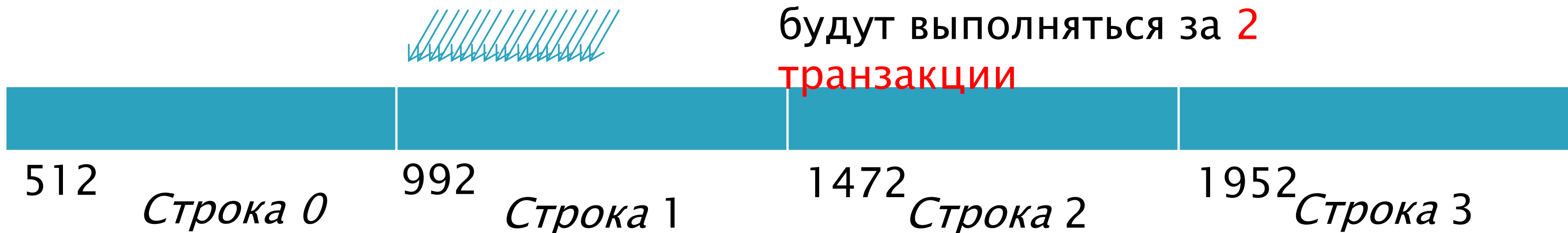
Обращения варпа

0 7 13

Матрицы и глобальная память

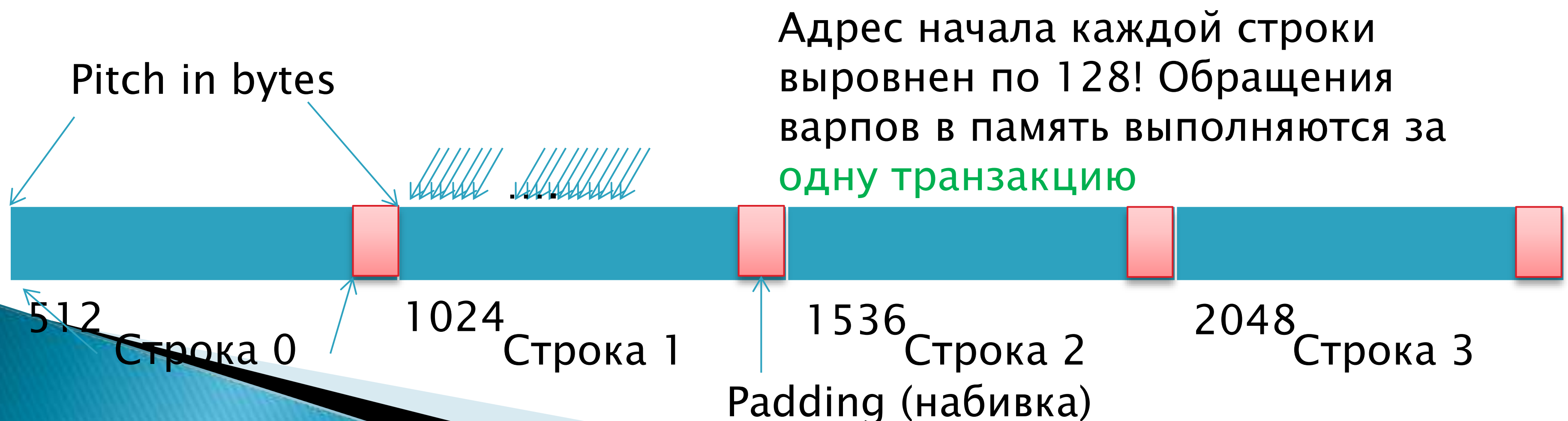
- Матрицы хранятся в линейном виде, по строкам
- Пусть длина строки матрицы – 480 байт (120 float)
  - обращение – matrix[idy*120 + idx]

Адрес начала каждой строки, кроме первой, не выровнен по 128 – обращения варпов в память будут выполняться за 2 транзакции



Матрицы и глобальная память

- Дополним каждую строку до размера, кратного 128 байтам – в нашем случае, 480 + 32 = 512, это наш pitch – фактическая ширина в байтах
- Эти байты никак не будут использоваться, т.е. 32/512=6% лишней памяти будет выделено (Но для больших матриц эта доля будет существенно меньше)
- Зато каждая строка будет выровнена по 128 байт
  - Обращение matrix[idy*128+ idx]



Матрицы и глобальная память

1 транзакция!

Обращения варпа

0 7 13 15

набивка

Выделение памяти с «паддингом»

- `cudaError_t cudaMallocPitch (void ** devPtr, size_t * pitch, size_t width, size_t height)`
  - width – логическая ширина матрицы в байтах
  - Выделяет не менее width * height байтов , может добавить в конец строк набивку, с целью соблюдения выравнивания начала строк
  - сохраняет указатель на память в (*devPtr)
  - сохраняет фактическую ширину строк в байтах в (*pitch)

Выделение памяти с «паддингом»

- ```
  cudaError_t cudaMallocPitch (void ** devPtr, size_t * pitch,
                   size_t width, size_t height)
  ```
- Адрес элемента (Row, Column) матрицы, выделенной при помощи cudaMallocPitch:

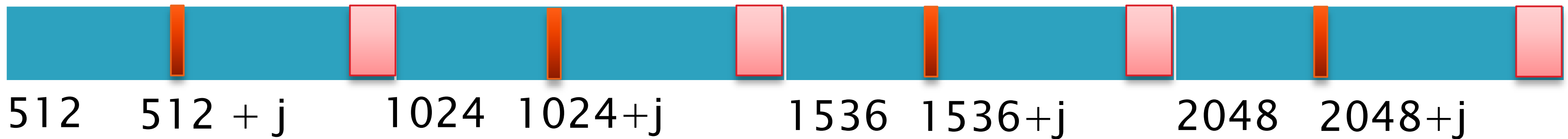
```
T* pElement = (T*)((char*) devPtr + Row * pitch) + Column
```
```

Копирование в матрицу с padding-ом

- `cudaError_t cudaMemcpy2D ( void* dst, size_t dpitch, const void* src, size_t spitch, size_t width, size_t height, cudaMemcpyKind kind )`
  - dst – указатель на матрицу, *в которую* нужно копировать , dpitch – ***фактическая*** ширина её строк в байтах
  - src – указатель на матрицу *из которой* нужно копировать, spitch – ***фактическая*** ширина её строк в байтах
  - width – сколько ***байтов*** каждой строки нужно копировать
  - height – число строк
  - kind – направление копирования (как в обычном cudaMemcpy)

Копирование в матрицу с padding-ом

- ```
  cudaError_t cudaMemcpy2D ( void* dst, size_t dpitch, const
                             void* src, size_t spitch,
                             size_t width, size_t height,
                                 cudaMemcpyKind kind )
  ```
- Из начала каждой строки исходной матрицы копируется по width байтов. Всего копируется width*height байтов, при этом
  - Адрес строки с индексом Row определяется по фактической ширине:

    `(char*)src + Row* spitch` – *в матрице-источнике*

    `(char*)dst + Row* dpitch` – *в матрице-получателе*
  ```

Обращение к матрице по столбцам?

- Матрица расположена по строкам, а обращение идёт по столбцам

Обращения нитей варпа

Каждая нить варпа обращается в свою строку к элементу в столбце j



Если матрица имеет размер больше 128 байт, то эти обращения ни за что не «влезут» в одну транзакцию!

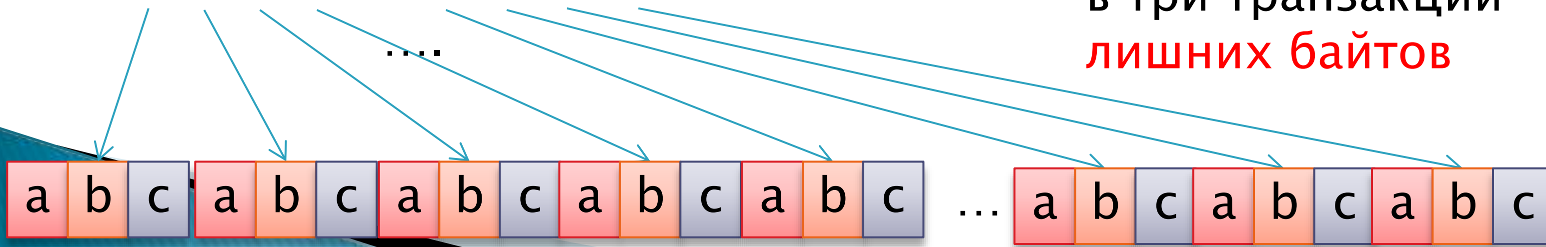
Транспонировать!

- Решение – хранить матрицу в транспонированном виде!
  - В этом случае обращения по столбцам превратятся в обращения к последовательным адресам
  - Выделять память под транспонированную матрицу также через `cudaMallocPitch`

Массивы структур?

```
struct example {
   int a;
   int b;
   int c;
}
__global__ void kernel(example * arrayOfExamples) {
    int idx = threadIdx.x + blockIdx.x * blockDim.x;
    arrayOfExamples[idx].c =
         arrayOfExamples[idx].b + arrayOfExamples[idx].a;
}
```

Обращение варпа в память будет выполняться в три транзакции – 256 лишних байтов

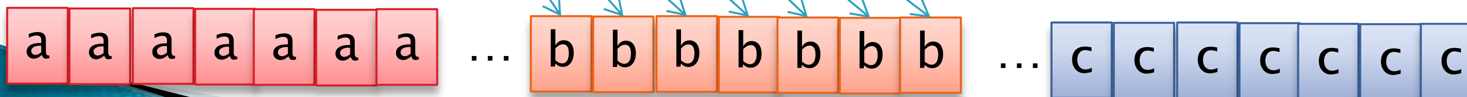


Структура с массивами!

```
struct example {
   int *a;
   int *b;
   int *c;
}
__global__ void kernel(example arrayOfExamples) {
    int idx = threadIdx.x + blockIdx.x * blockDim.x;
    arrayOfExamples.c[idx] =
         arrayOfExamples.b[idx] + arrayOfExamples.a[idx];
}
```

Обращения нитей варпа

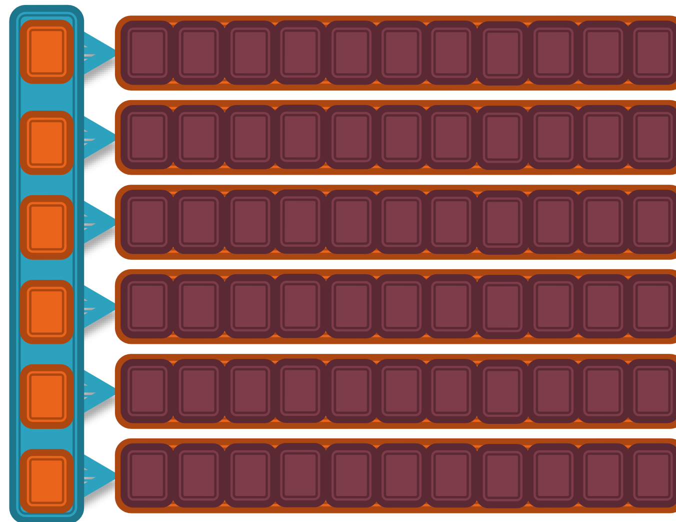
Обращение варпа в память будет выполняться за одну транзакцию



Косвенная адресация

- Требует двух чтений из памяти
  - сначала A[i], потом A[i][j]
- При первом чтении варпу нужно всего 4 байта, а скачается 128

```
float **A;
A[i][j] = 1;
```



Выводы

- Обращения нитей варпа в память должны быть пространственно-локальными
- Начала строк матрицы должны быть выровнены
- Массивы структур -> структура с массивами
- 16KB vs 48KB L1

- Избегаем косвенной адресации
- Избегаем обращений нитей варпа к столбцу матрицы, если матрица хранится по строкам (threadIdx.x должен быть без множителя!!!!)

- В случае сильно разреженного доступа проверяем работу с отключенным кешем