АЛГОРИТМЫ МАТРИЧНОГО УМНОЖЕНИЯ.

АФАНАСЬЕВ ИЛЬЯ
AFANASIEV_ILYA@ICLOUD.COM

ПОСТАНОВКА ЗАДАЧИ

$$C = A \times B$$

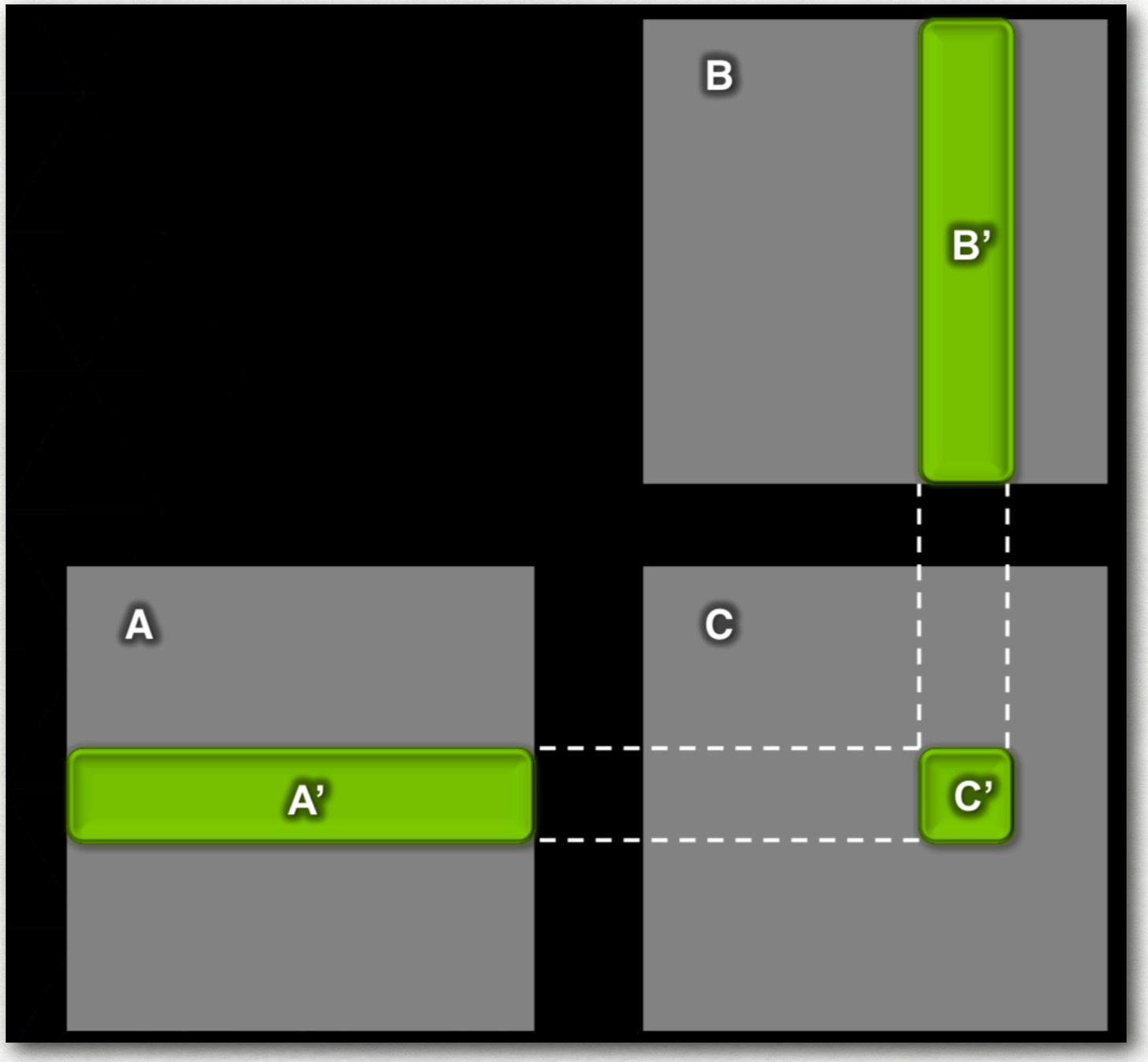
$$c_{i,j} = \sum_{k=0}^{N-1} a_{i,k} b_{k,j} \,|\, i, j = 0,1,...N-1$$

N x N = 2048 x 2048, BLOCK_SIZE = 32

(tx, ty) – (32, 32) – нити внутри блока

(bx, by) – (64, 64) – число блоков

ПЕРЕМНОЖЕНИЕ МАТРИЦ С ИСПОЛЬЗОВАНИЕМ ГЛОБАЛЬНОЙ ПАМЯТИ



ПЕРЕМНОЖЕНИЕ МАТРИЦ С ИСПОЛЬЗОВАНИЕМ ГЛОБАЛЬНОЙ ПАМЯТИ

```
template <typename dataT>
__global__ void kernel(dataT *a, dataT *b, int n, dataT *c)
{
    int bx = blockIdx.x; // номер блока по x
    int by = blockIdx.y; // номер блока по y
    int tx = threadIdx.x; // номер нити в блоке по x
    int ty = threadIdx.y; // номер нити в блоке по y
    dataT sum = 0;
    int ia = n * (BLOCK_SIZE * by + ty); // номер строки из A'
    int ib = BLOCK_SIZE * bx + tx; // номер столбца из B'
    int ic = ia + ib; // номер элемента из C'
    // вычисление элемента матрицы C
    for (int k = 0; k < n; k++)
        sum += a[ia + k] * b[ib + k * n];
    c[ic] = sum;
}
```

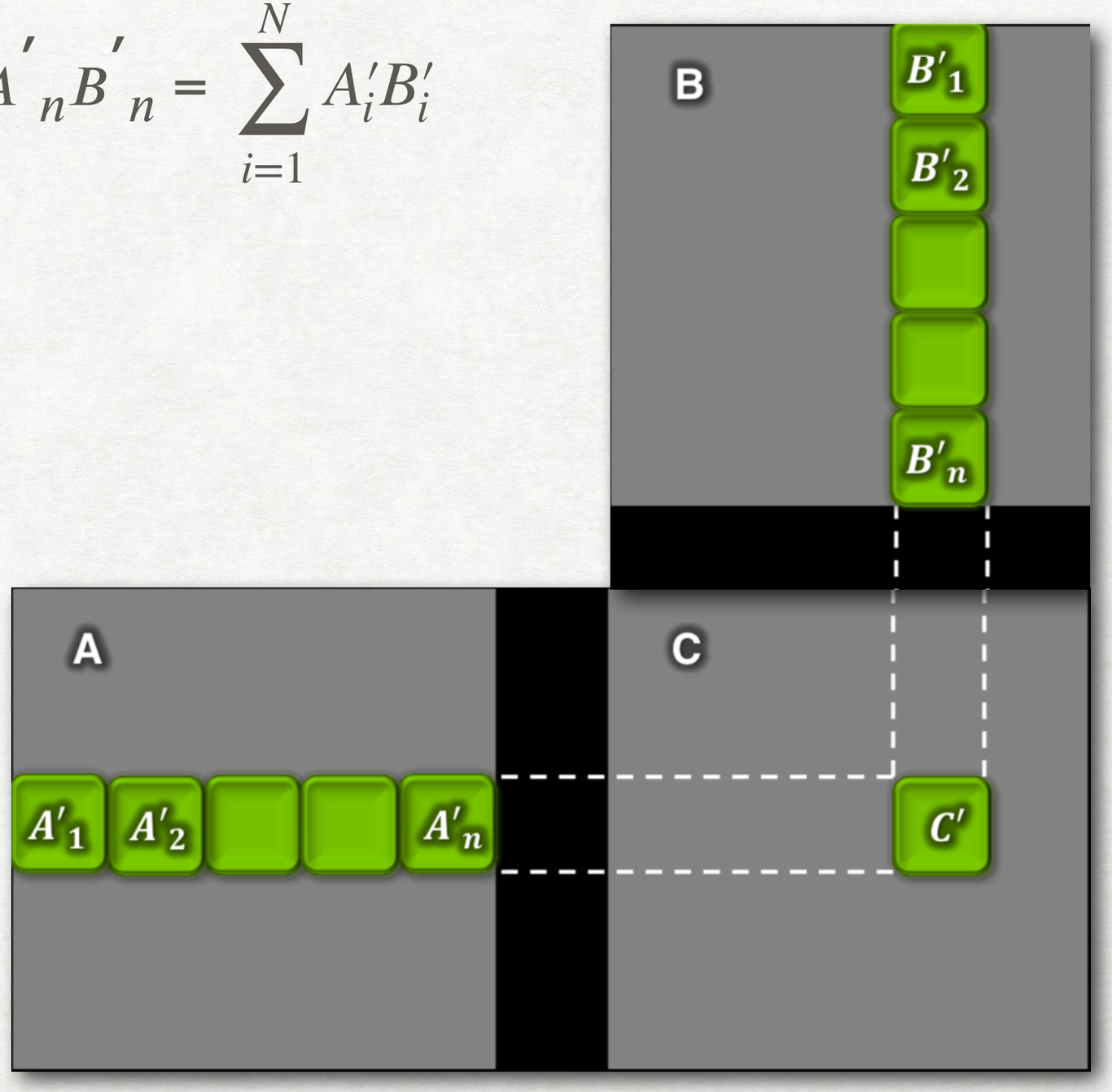
ПЕРЕМНОЖЕНИЕ МАТРИЦ С ИСПОЛЬЗОВАНИЕМ ГЛОБАЛЬНОЙ ПАМЯТИ

- Проблема: данные не переиспользуются
- При вычислении $ci,0$, $ci,1$, ... , $ci,N-1$ N-раз считывается строка $ai,0$, $ai,1$, ... , $ai,N-1$

БЛОЧНОЕ УМНОЖЕНИЕ С ИСПОЛЬЗОВАНИЕМ РАЗДЕЛЯЕМОЙ ПАМЯТИ

$$C' = A'_1 B'_1 + A'_2 B'_2 + \cdots + A'_n B'_n = \sum_{i=1}^{N} A'_i B'_i$$

$$n = \frac{N}{block\_size}$$

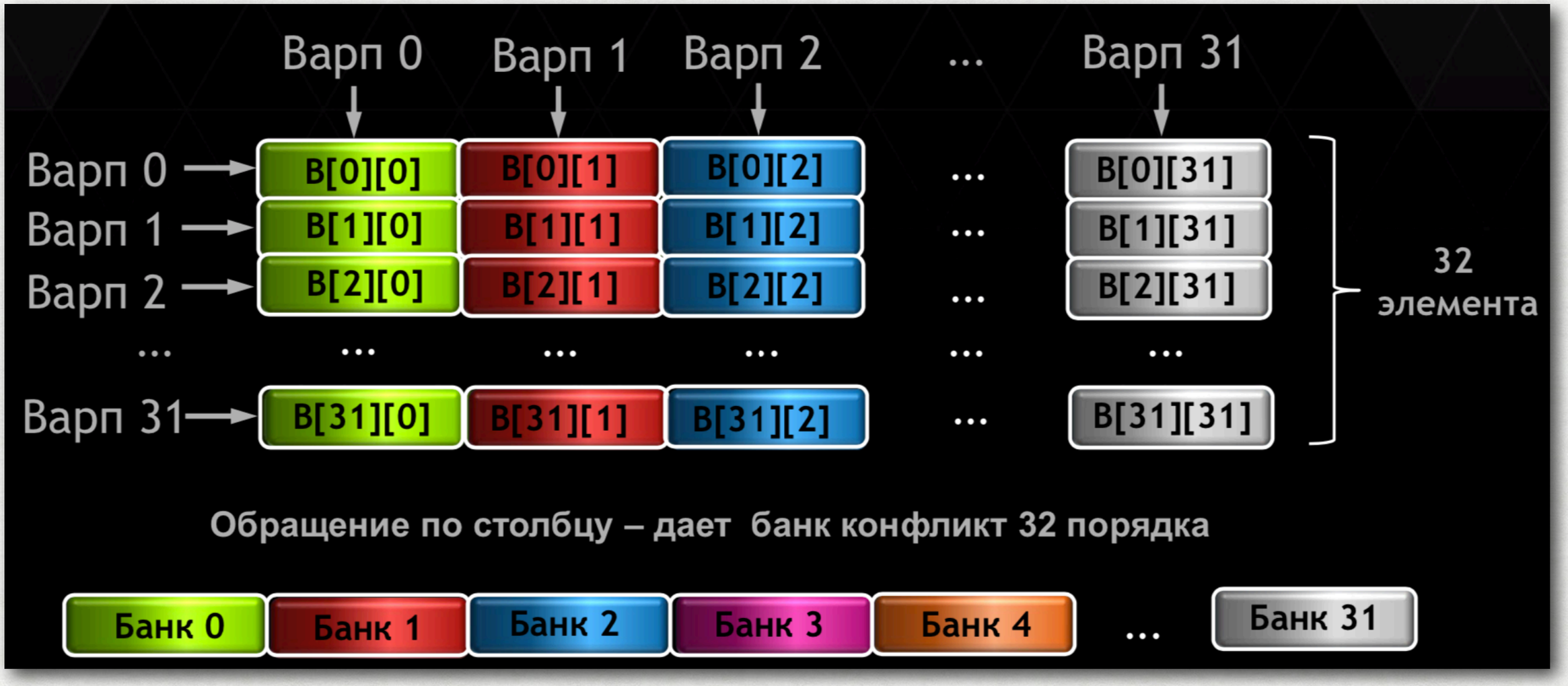


БЛОЧНОЕ УМНОЖЕНИЕ С ИСПОЛЬЗОВАНИЕМ РАЗДЕЛЯЕМОЙ ПАМЯТИ

```
template <typename dataT>
__global__ void kernel(dataT *a, dataT *b, int n, dataT *c)
{
    int bx = blockIdx.x, by = blockIdx.y; // номер блока по x и y
    int tx = threadIdx.x, ty = threadIdx.y; // номер нити в блоке по x и y
    int aBegin = n * BLOCK_SIZE * by;
    int aEnd = aBegin + n - 1, bBegin = BLOCK_SIZE * bx;
    int aStep = BLOCK_SIZE, bStep = BLOCK_SIZE * n;
    dataT sum = 0;
    __shared__ dataT as[BLOCK_SIZE][BLOCK_SIZE];
    __shared__ dataT bs[BLOCK_SIZE][BLOCK_SIZE];
    for (int ia = aBegin, ib = bBegin; ia <= aEnd; ia += aStep, ib += bStep)
    {
        as[tx][ty] = a[ia + n * ty + tx];
        bs[tx][ty] = b[ib + n * ty + tx];
        __syncthreads();
        for (int k = 0; k < BLOCK_SIZE; k++)
            sum += as[k][ty] * bs[tx][k];
        __syncthreads();
    }
    c[aBegin + bBegin + ty * n + tx] = sum;
}
```

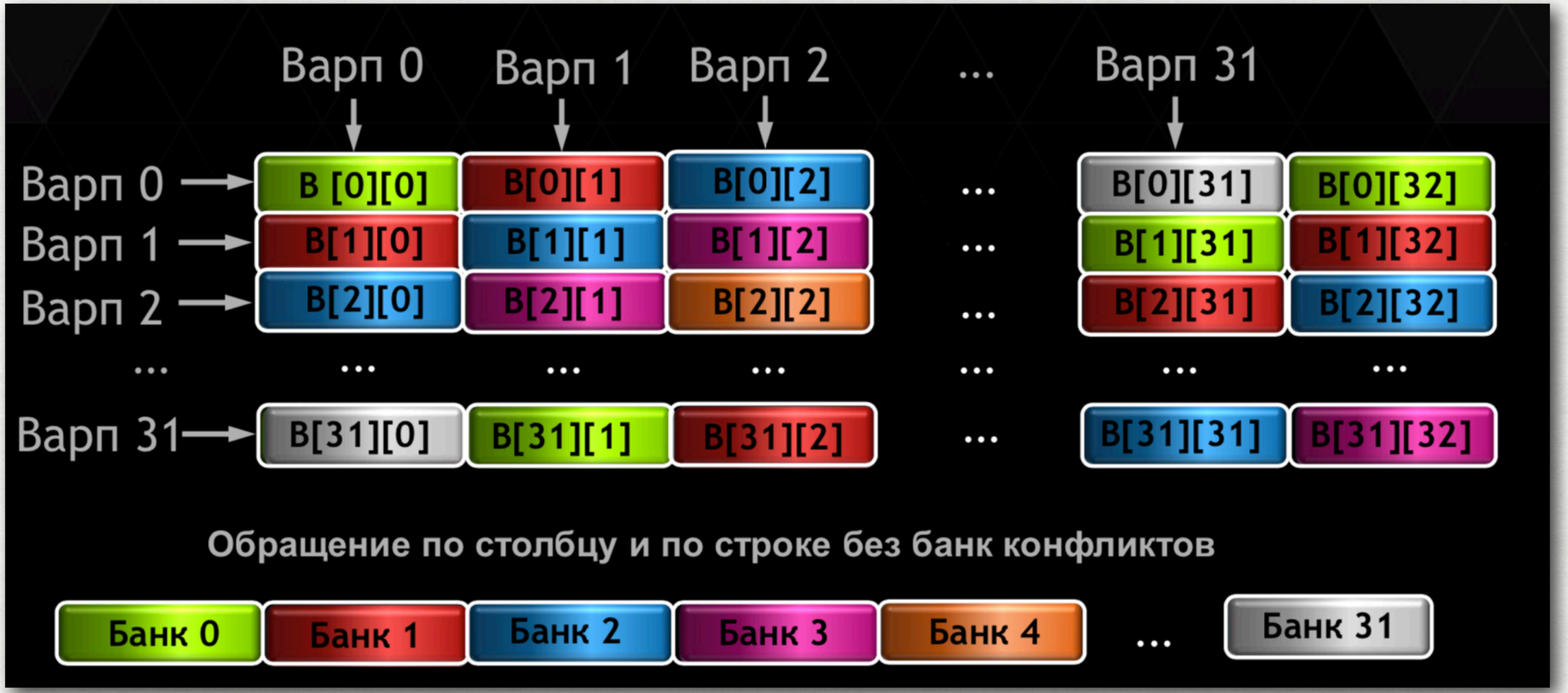
БАНК КОНФЛИКТЫ

Проблема при работе с массивом __shared__ double B [32][32]:



УСТРАНЕНИЕ БАНК КОНФЛИКТОВ

__shared__ double B [32][**33**];



ПРАКТИЧЕСКОЕ ЗАДАНИЕ

- Реализовать два варианта матричного умножения (с использованием глобальной и разделяемой памяти)
- Для реализации с использованием разделяемой памяти использовать версию с конфликтами доступа к банкам памяти и без, измерить различия во времени выполнения
- Версия CPU-кода с генерацией матриц, копированиями данных, шаблонами ядре и проверки корректности доступна в файле matrix_multiplications_template.cu
- Сравнить время выполнения и производительность для типов float и double, реализовав шаблонные (c++) ядра
- Производительность определяется как:

$$performance(GFLOP/s) = \frac{FLOP}{T * 10^9} = \frac{2 * N^3}{T * 10^9}$$

ВОПРОСЫ?