CS Space
Клуб технологий и науки

Вычисления на видеокартах

Vulkan

OpenCL

NVIDIA CUDA

Лекция 7

- Merge sort
- Merge path
- Coarse to fine схема
- Patch Match (Depth Maps)
- Look Up Tables (LUT)

@UnicornGlade
@PolarNick239
polarnick239@gmail.com
Николай Полярный


ITMO
SPbU
MIPT
CUDA
1
2
3
OpenCL
KHRONOS
TESLA
NVIDIA

Task04: Префиксные суммы на **Tesla T4** (320 GB/s)

🥇 Роберт Смайт - команда СПбГУ - **108.15 GB/s**, CUDA

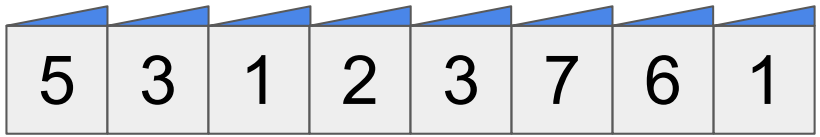
🥈 Илья Коннов - команда ИТМО / Яндекс - **92.98 GB/s**, OpenCL

🥉 Mikhail Stulov - команда МФТИ / ТБанк - **89.48 GB/s**, OpenCL

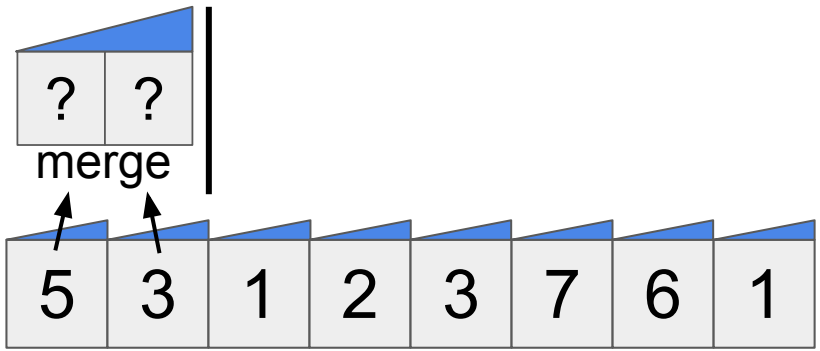
P.S. мое baseline решение - **10.68 GB/s**

Глава 1: Merge sort

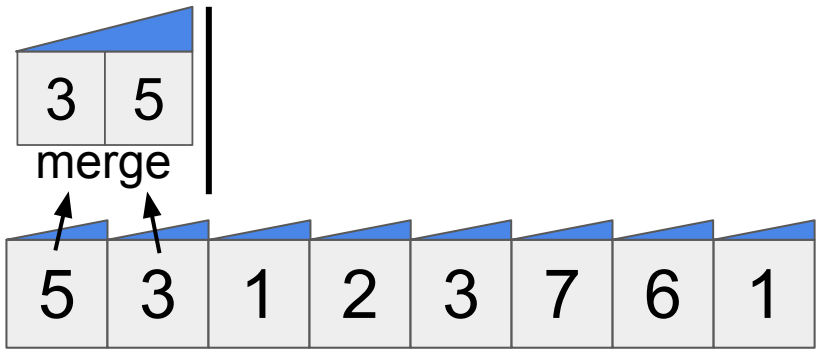
Merge sort



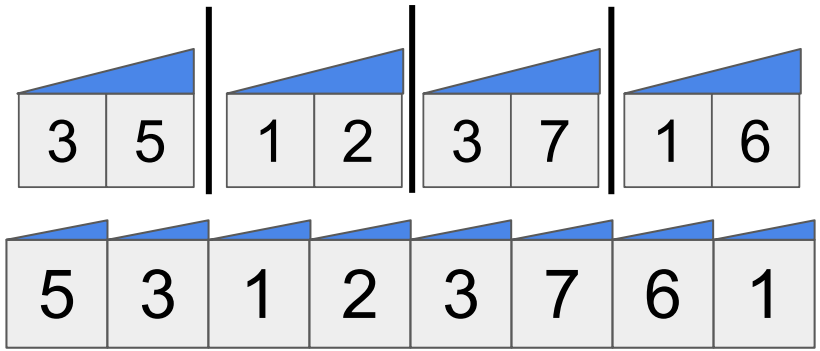
Merge sort



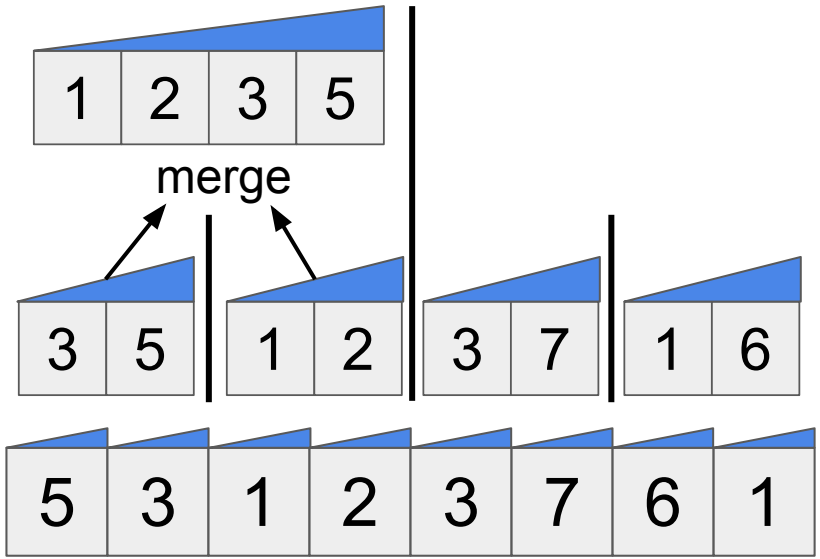
Merge sort



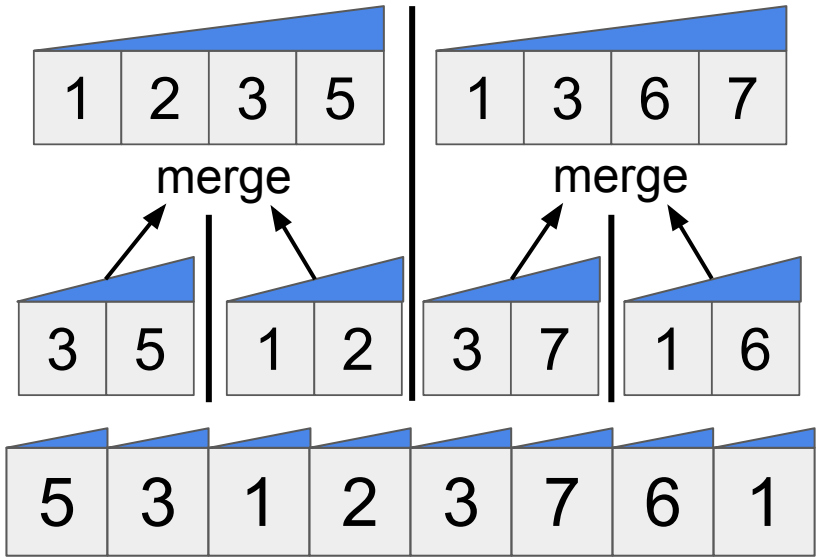
Merge sort



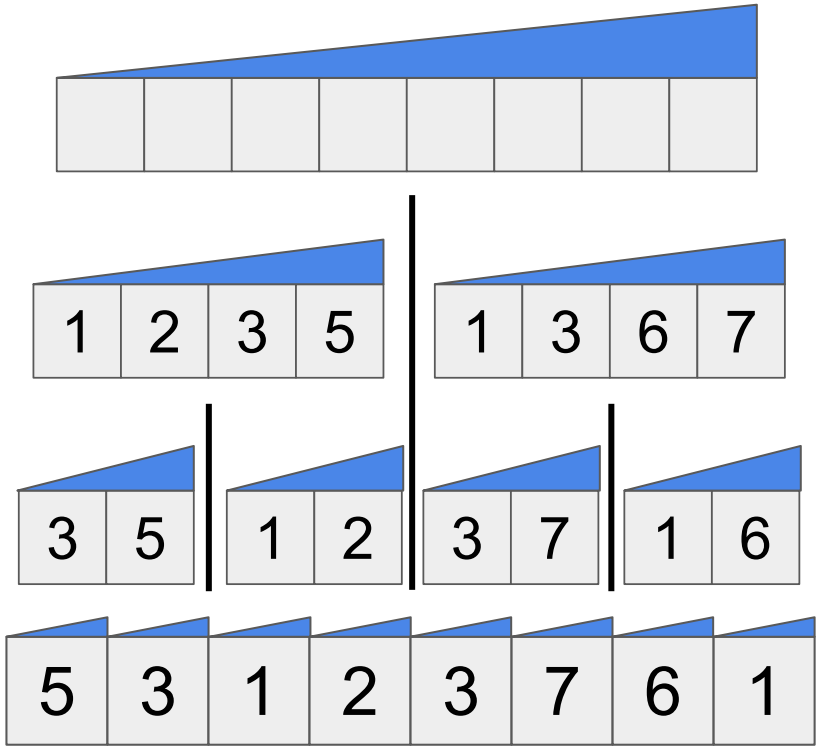
Merge sort



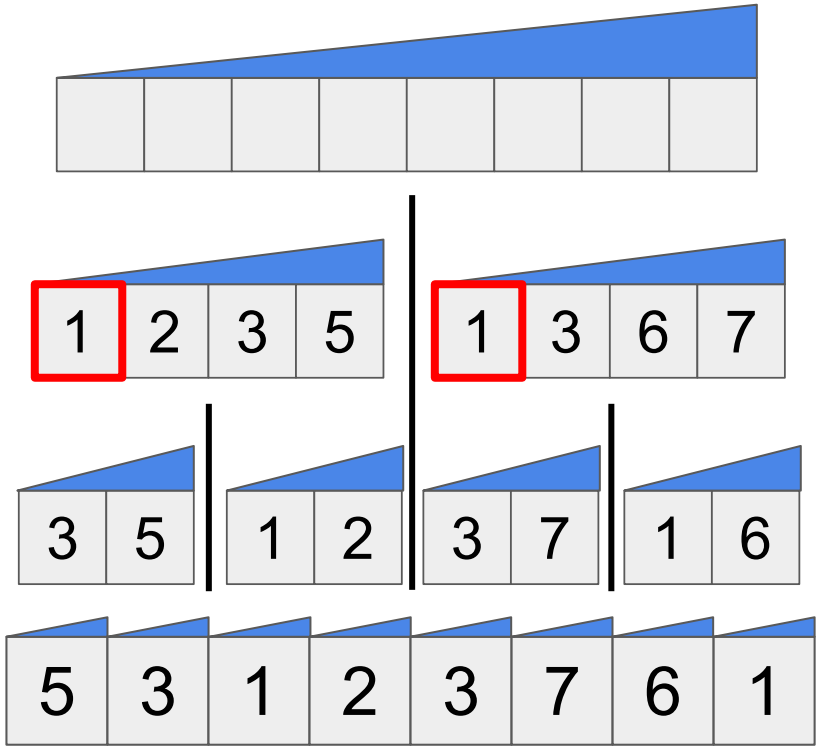
Merge sort



Merge sort: алгоритм слияния (merge)

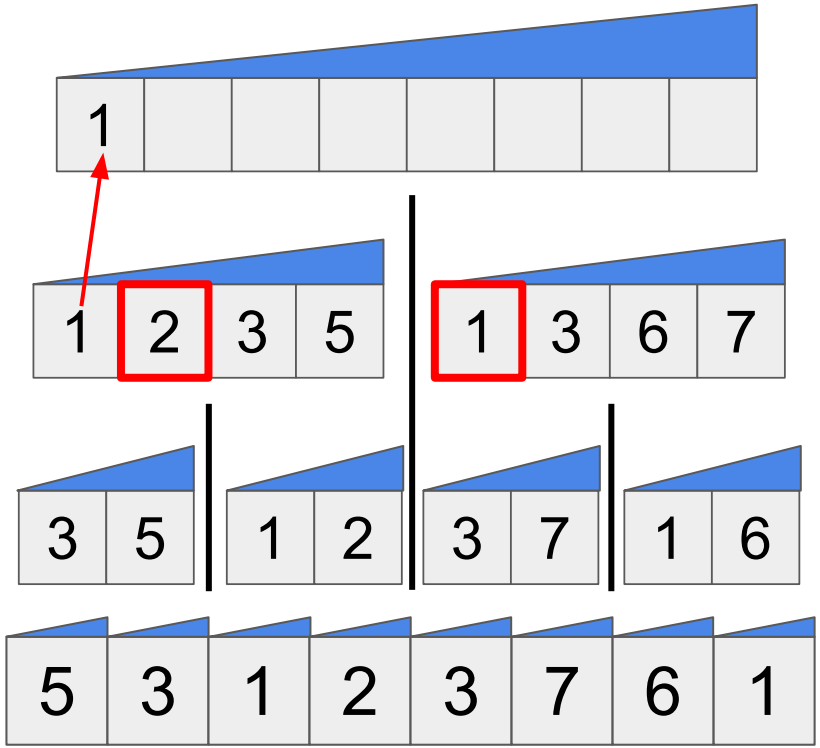


Merge sort: алгоритм слияния (merge)



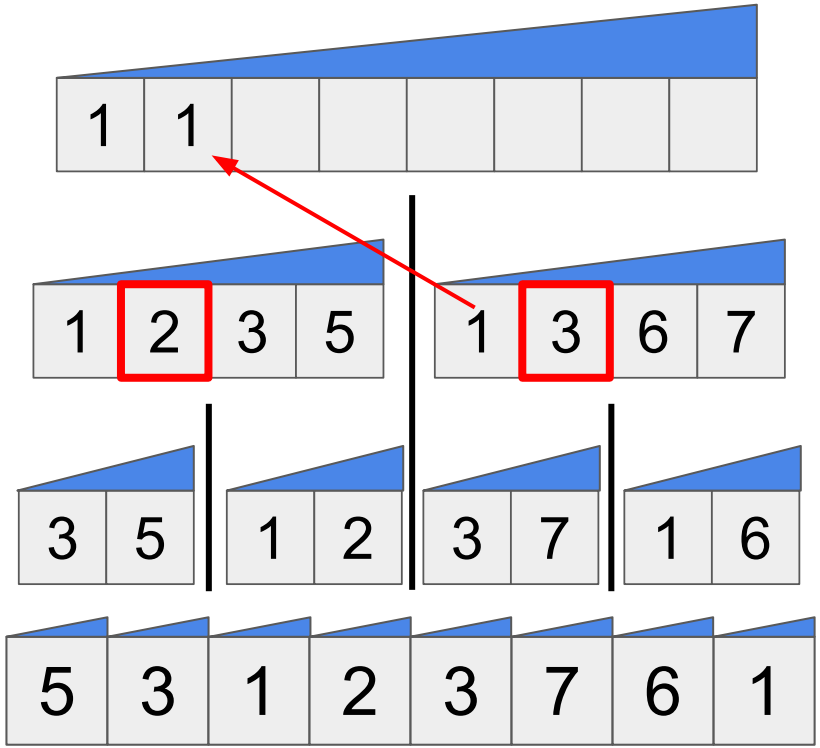
1) **Два указателя** - следующие кандидаты

Merge sort: алгоритм слияния (merge)



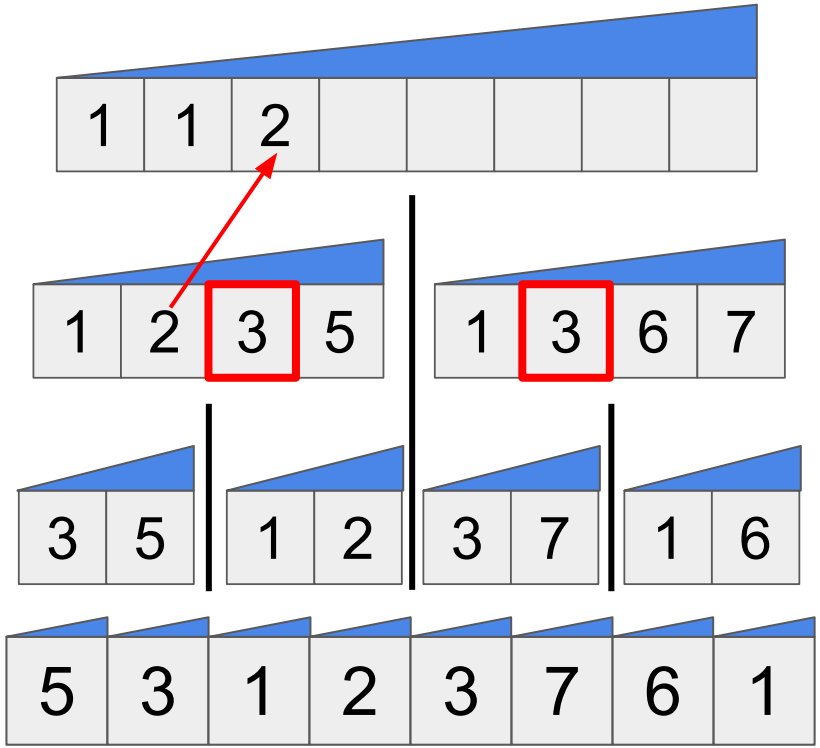
1) **Два указателя** - следующие кандидаты
2) Берем **минимум**

Merge sort: алгоритм слияния (merge)



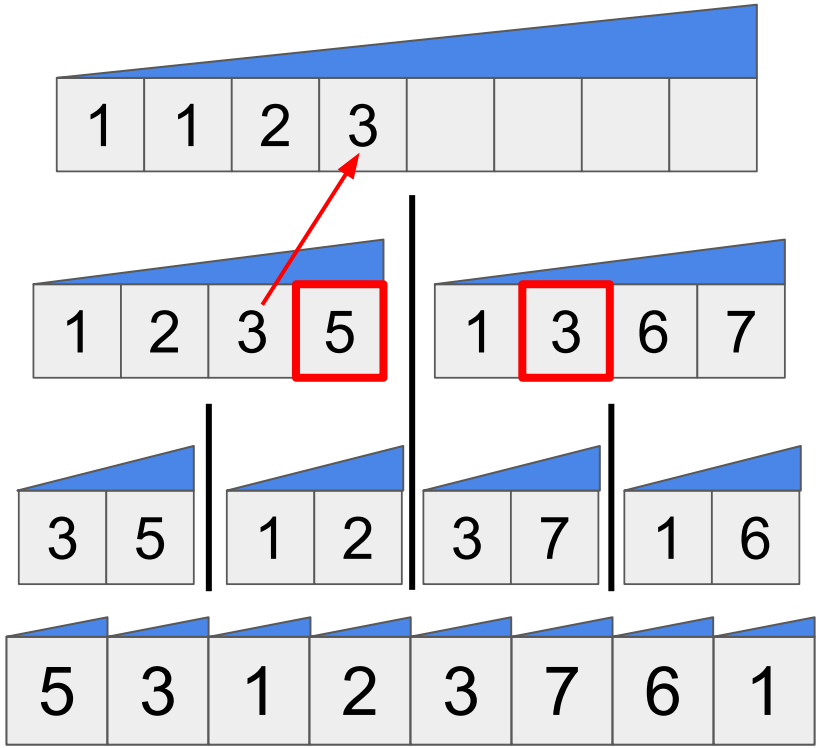
1) **Два указателя** - следующие кандидаты
2) Берем **минимум**

Merge sort: алгоритм слияния (merge)



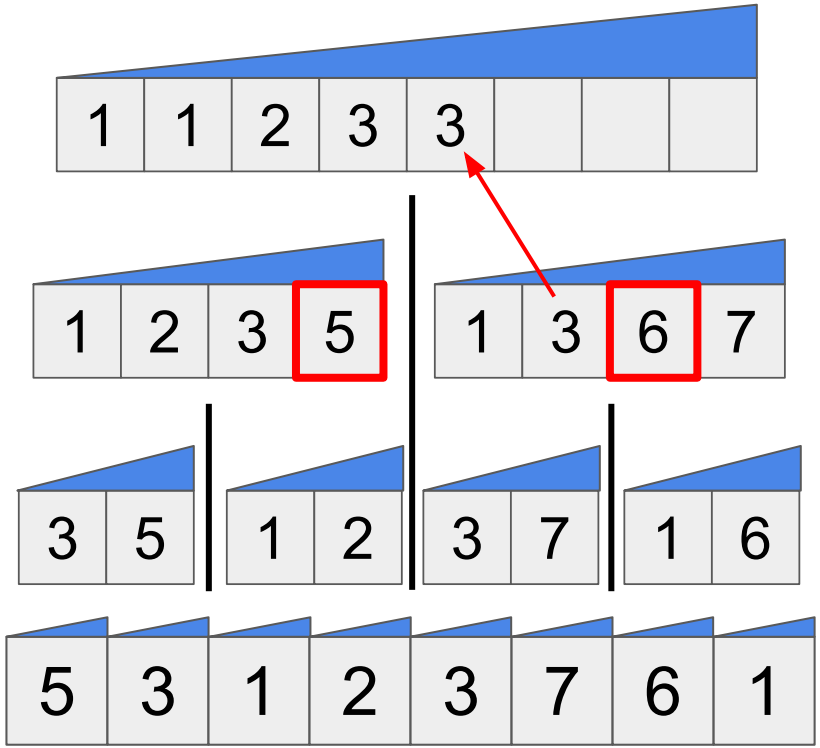
1) **Два указателя** - следующие кандидаты
2) Берем **минимум**

Merge sort: алгоритм слияния (merge)



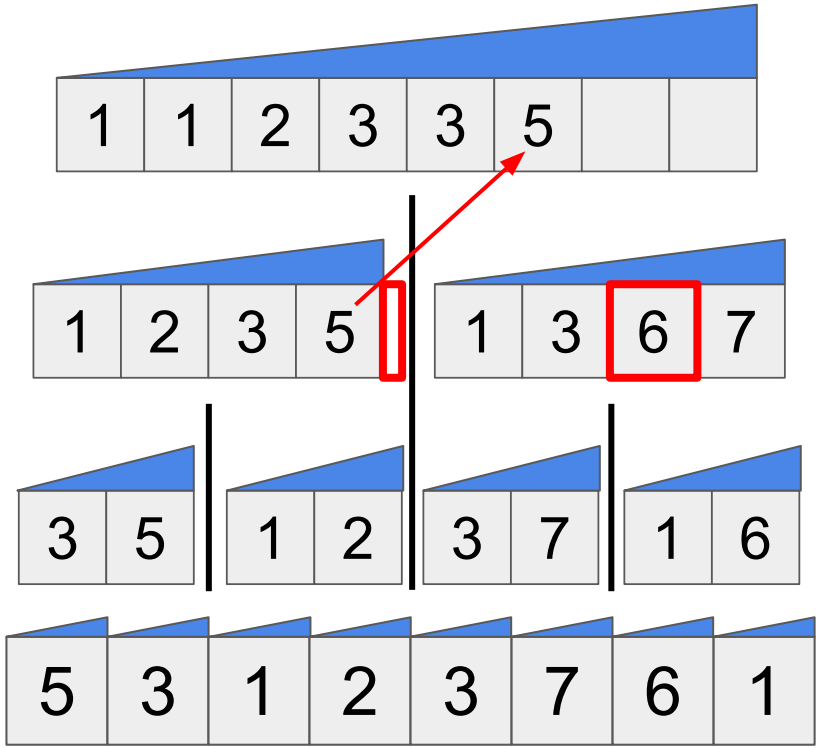
1) **Два указателя** - следующие кандидаты
2) Берем **минимум**

Merge sort: алгоритм слияния (merge)



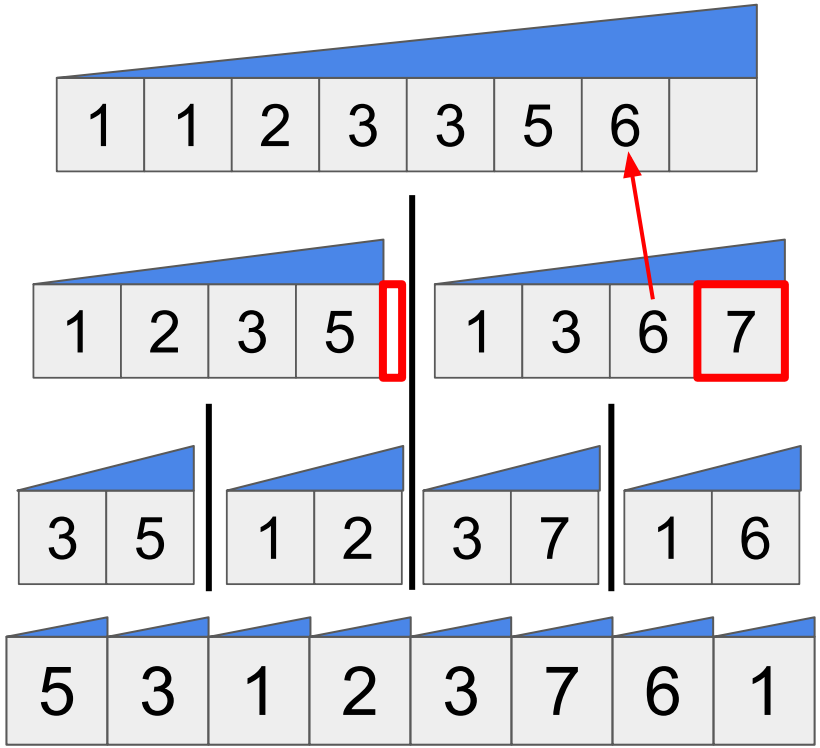
1) **Два указателя** - следующие кандидаты
2) Берем **минимум**

Merge sort: алгоритм слияния (merge)



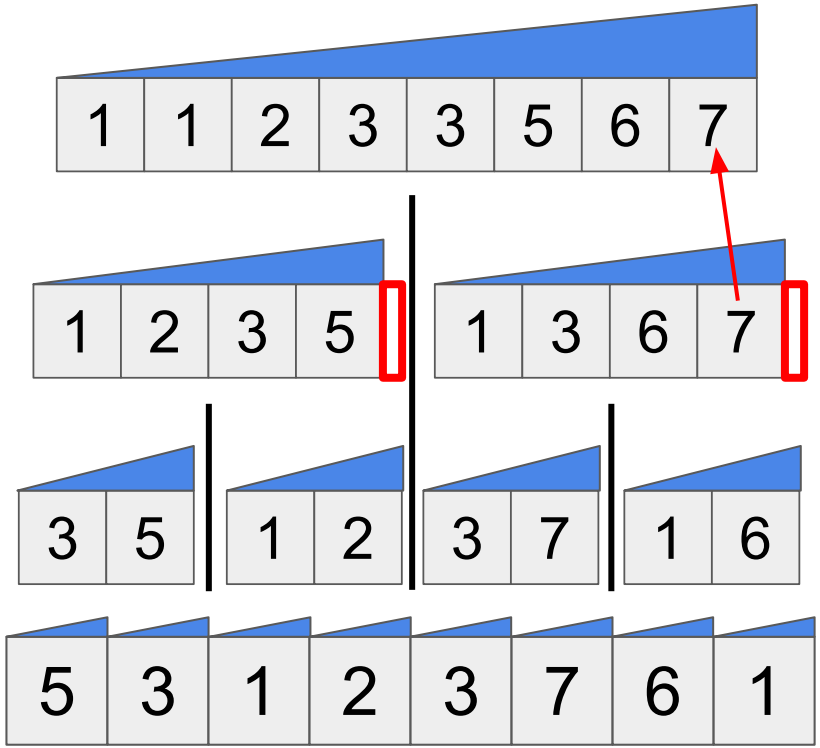
1) **Два указателя** - следующие кандидаты
2) Берем **минимум**

Merge sort: алгоритм слияния (merge)



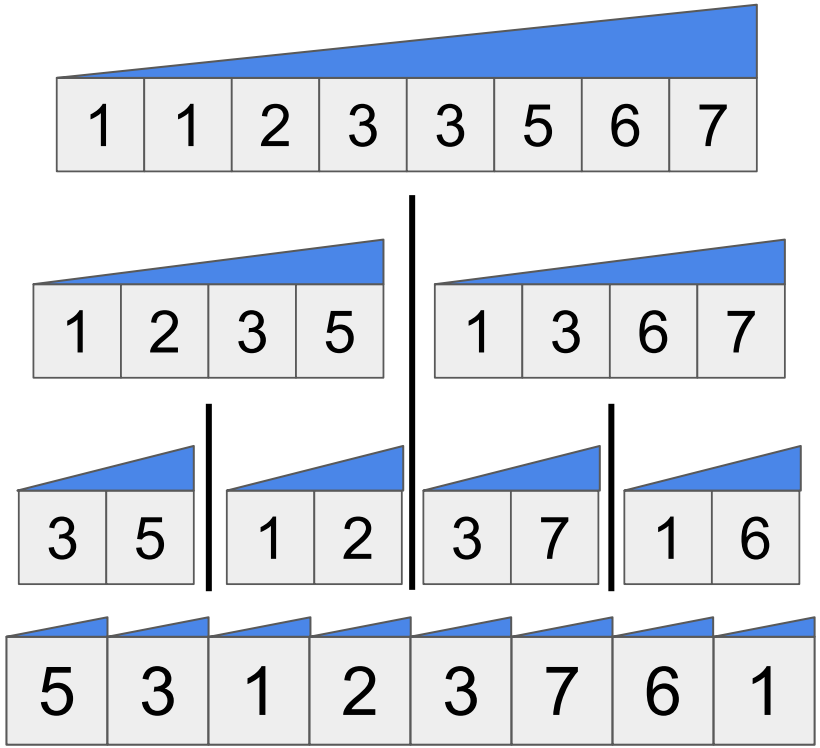
1) **Два указателя** - следующие кандидаты
2) Берем **минимум**

Merge sort: алгоритм слияния (merge)



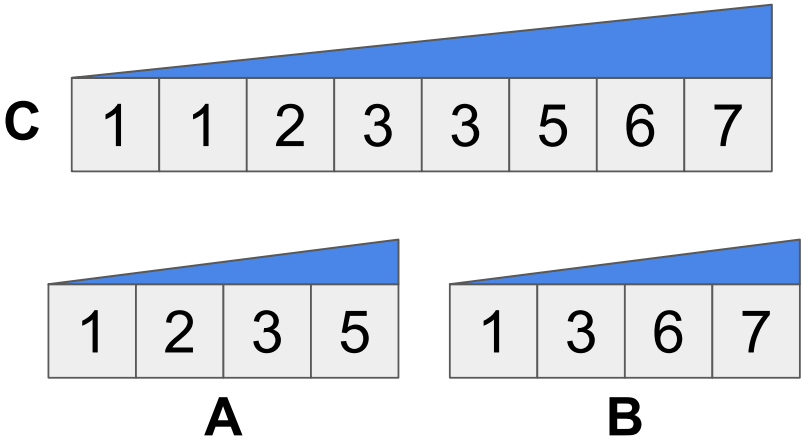
1) **Два указателя** - следующие кандидаты
2) Берем **минимум**

Merge sort: алгоритм слияния (merge)



Как отсортировать на видеокарте?
Т.е. как отсортировать супер многопоточно?

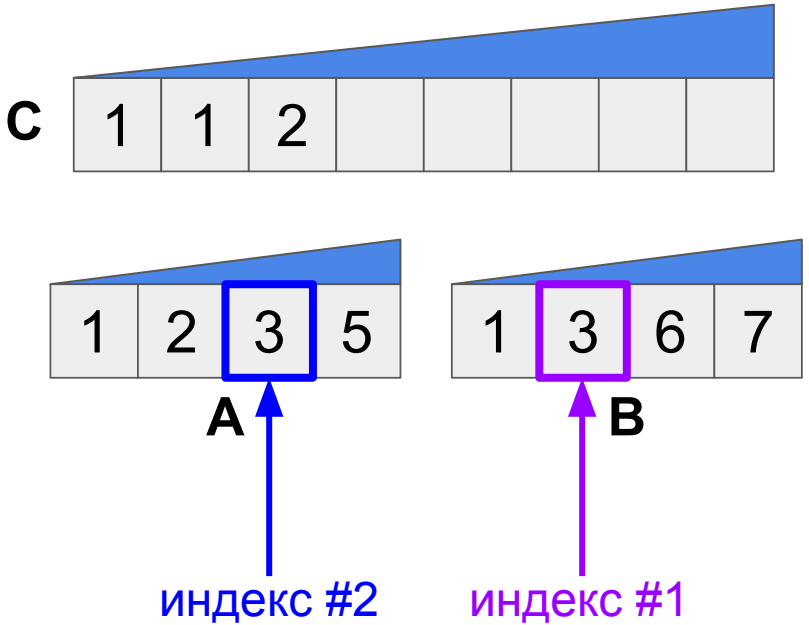
Merge sort: параллельный алгоритм слияния (merge)



Как отсортировать на видеокарте?
Т.е. как отсортировать супер многопоточно?

Достаточно научиться делать merge!

Merge sort: параллельный алгоритм слияния (merge)

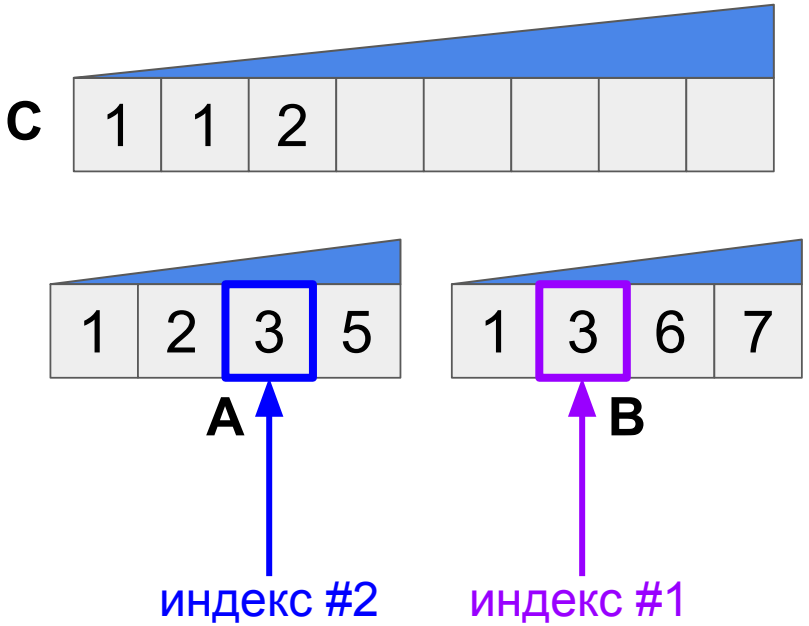


Как отсортировать на видеокарте?
Т.е. как отсортировать супер многопоточно?

Достаточно научиться делать merge!

Операция слияния тривиальна если
1) знаем **указатель** в A
2) знаем **указатель** в B

Merge sort: параллельный алгоритм слияния (merge)

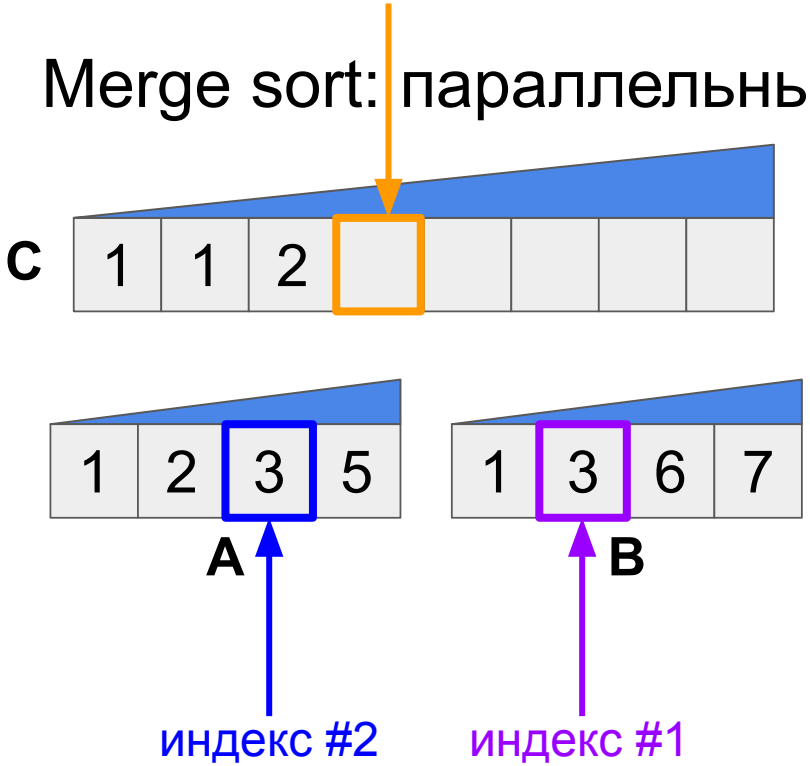


Как отсортировать на видеокарте?
Т.е. как отсортировать супер многопоточно?

Достаточно научиться делать merge!

Операция слияния тривиальна если
1) знаем **указатель** в A
2) знаем **указатель** в B
3) ???

Merge sort: параллельный алгоритм слияния (merge)

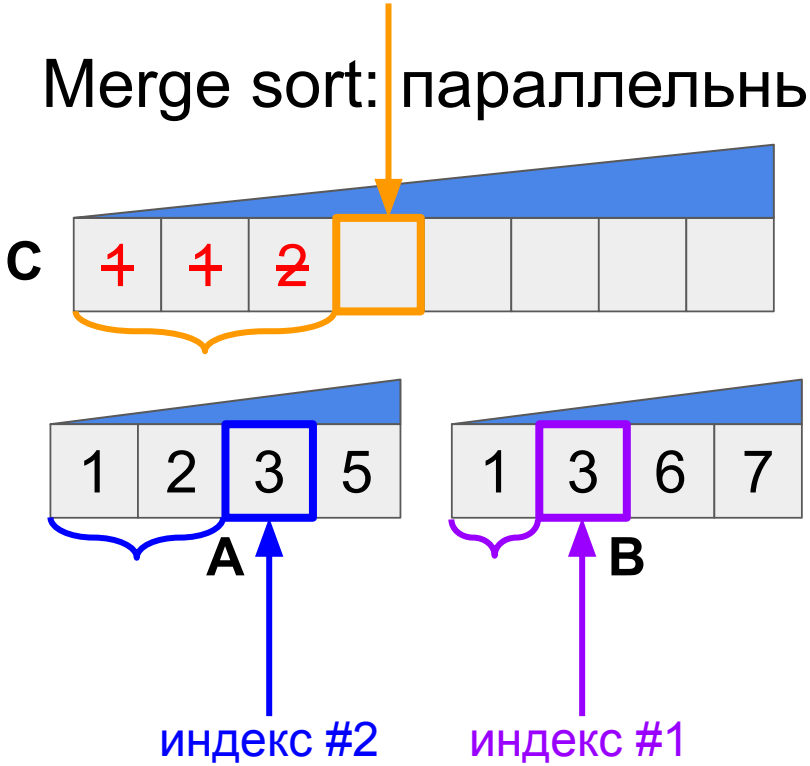


Как отсортировать на видеокарте?
Т.е. как отсортировать супер многопоточно?

Достаточно научиться делать merge!

Операция слияния тривиальна если
1) знаем **указатель** в A
2) знаем **указатель** в B
3) можем найти на какой **индекс** в C записать минимум?

Merge sort: параллельный алгоритм слияния (merge)

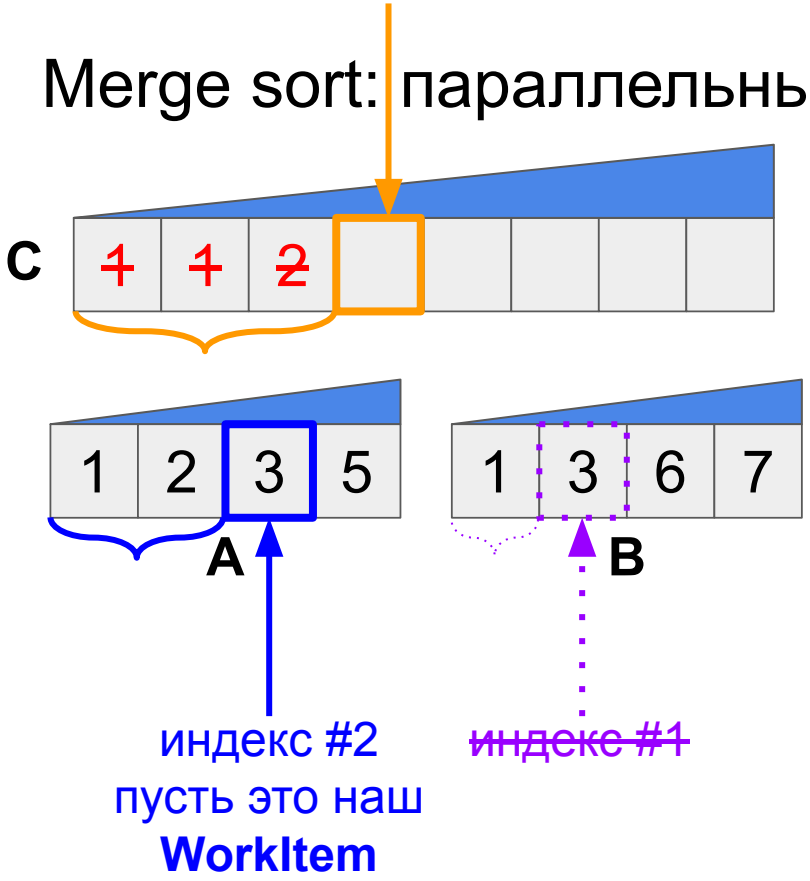


Как отсортировать на видеокарте? Т.е. как отсортировать супер многопоточно?

Достаточно научиться делать merge!

Операция слияния тривиальна если
1) знаем **указатель** в A
2) знаем **указатель** в B
3) можем найти на какой **индекс** в C записать минимум?

Merge sort: параллельный алгоритм слияния (merge)

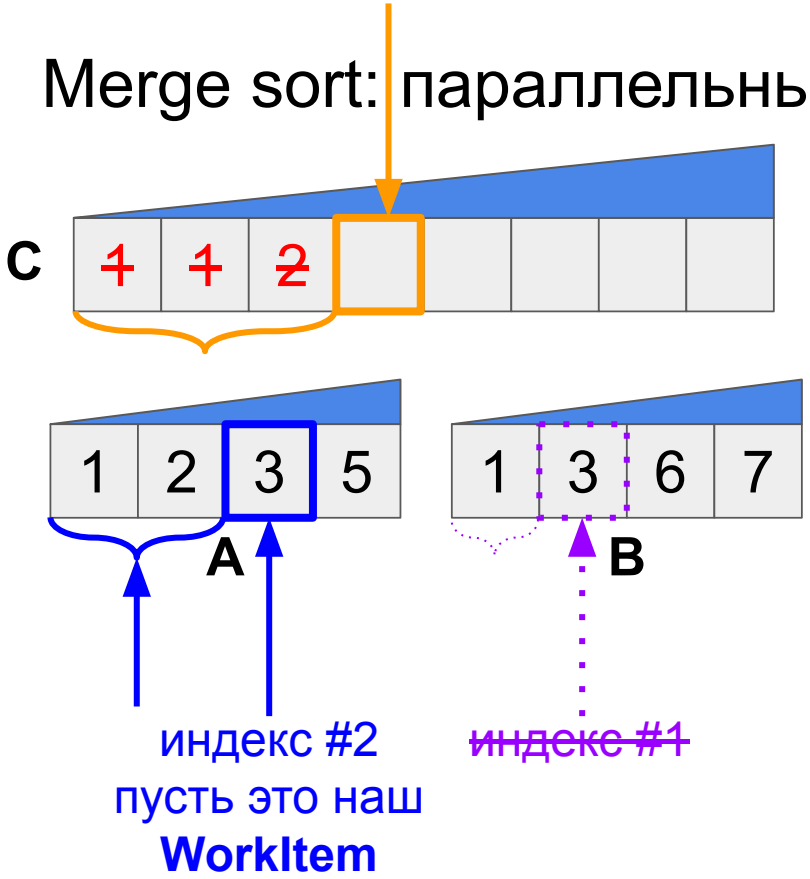


Как отсортировать на видеокарте?
Т.е. как отсортировать супер многопоточно?

Достаточно научиться делать merge!

Операция слияния тривиальна если
1) знаем **указатель** в A
2) знаем **указатель** в B
3) можем найти на какой **индекс** в C
записать минимум?

Merge sort: параллельный алгоритм слияния (merge)



Знаем ли мы сколько чисел до нас в A?

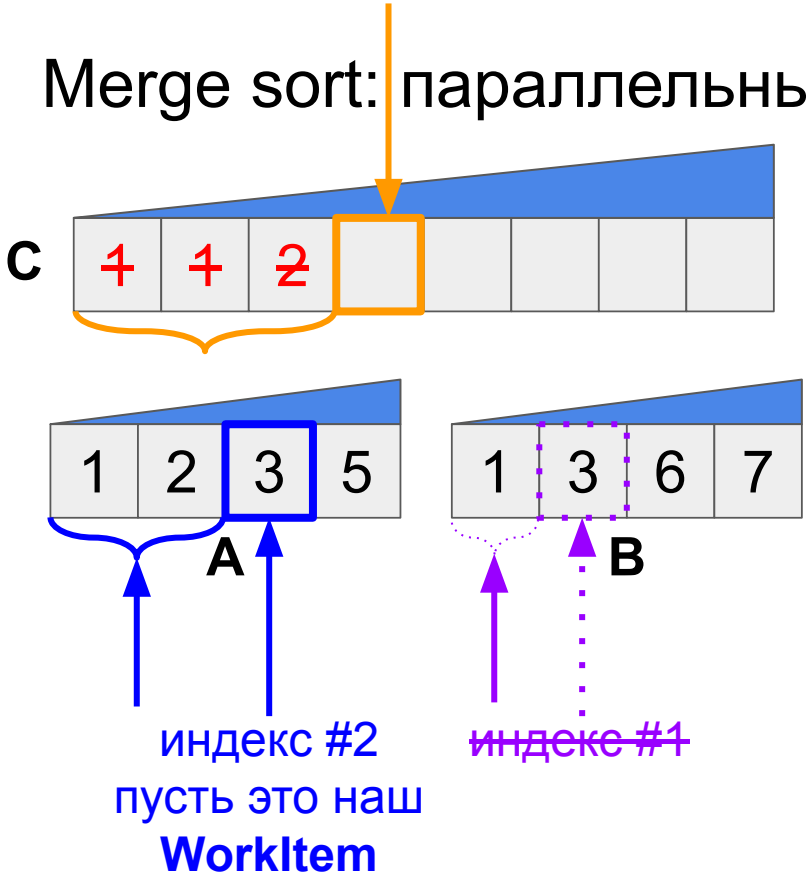
Как отсортировать на видеокарте? Т.е. как отсортировать супер многопоточно?

Достаточно научиться делать merge!

Операция слияния тривиальна если
1) знаем **указатель** в A
2) знаем **указатель** в B
3) можем найти на какой **индекс** в C записать минимум?

neerc.ifmo.ru: двоичный поиск на вики ИТМО

Merge sort: параллельный алгоритм слияния (merge)



Знаем ли мы сколько чисел до нас в A?

Как отсортировать на видеокарте?
Т.е. как отсортировать супер многопоточно?

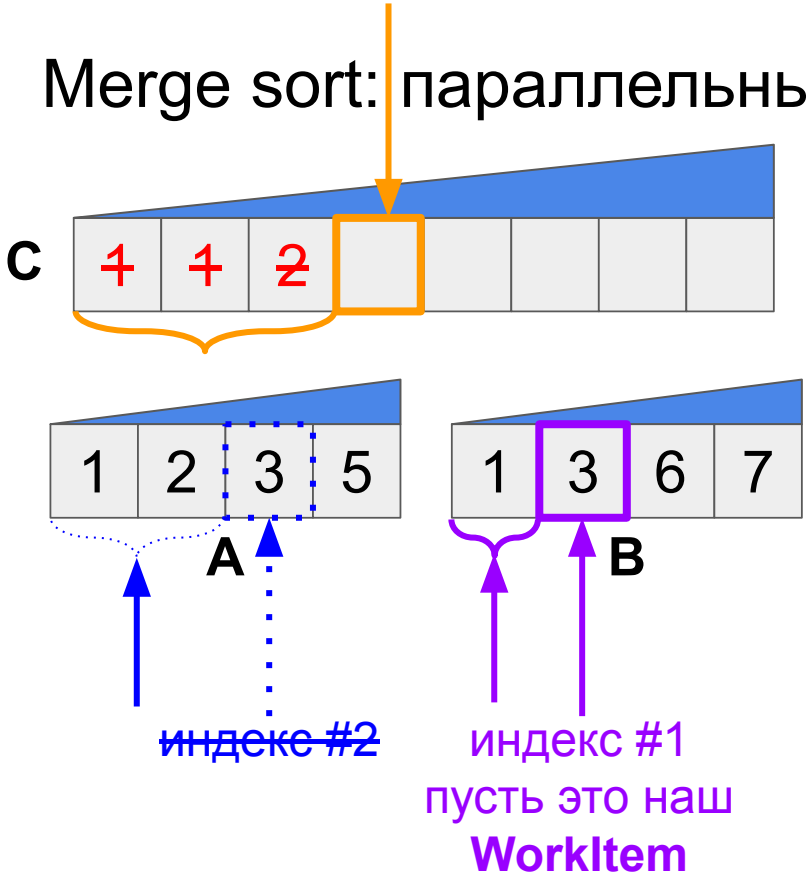
Достаточно научиться делать merge!

Операция слияния тривиальна если
1) знаем **указатель** в A
2) знаем **указатель** в B
3) можем найти на какой **индекс** в C
записать минимум?

Как найти сколько чисел до нас в B?
Бинарный поиск! O(log N)

neerc.ifmo.ru: двоичный поиск на вики ИТМО

Merge sort: параллельный алгоритм слияния (merge)



Как отсортировать на видеокарте?
Т.е. как отсортировать супер многопоточно?

Достаточно научиться делать merge!

Операция слияния тривиальна если
1) знаем **указатель** в A
2) знаем **указатель** в B
3) можем найти на какой **индекс** в C
записать минимум?

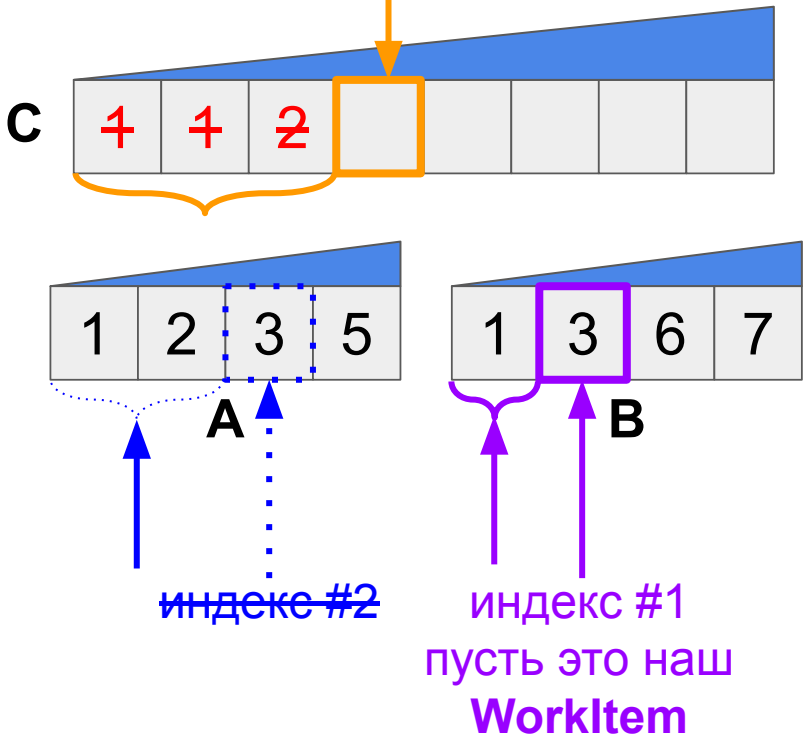
Как найти сколько чисел до нас в B?

Знаем ли мы сколько чисел до нас в A?
Бинарный поиск! O(log N)

Есть ли разница в случае когда наш WorkItem в A или в B?

neerc.ifmo.ru: двоичный поиск на вики ИТМО

Merge sort: параллельный алгоритм слияния (merge)



Как отсортировать на видеокарте?
Т.е. как отсортировать супер многопоточно?

Достаточно научиться делать merge!

Операция слияния тривиальна если
1) знаем **указатель** в A
2) знаем **указатель** в B
3) можем найти на какой **индекс** в C
записать минимум?

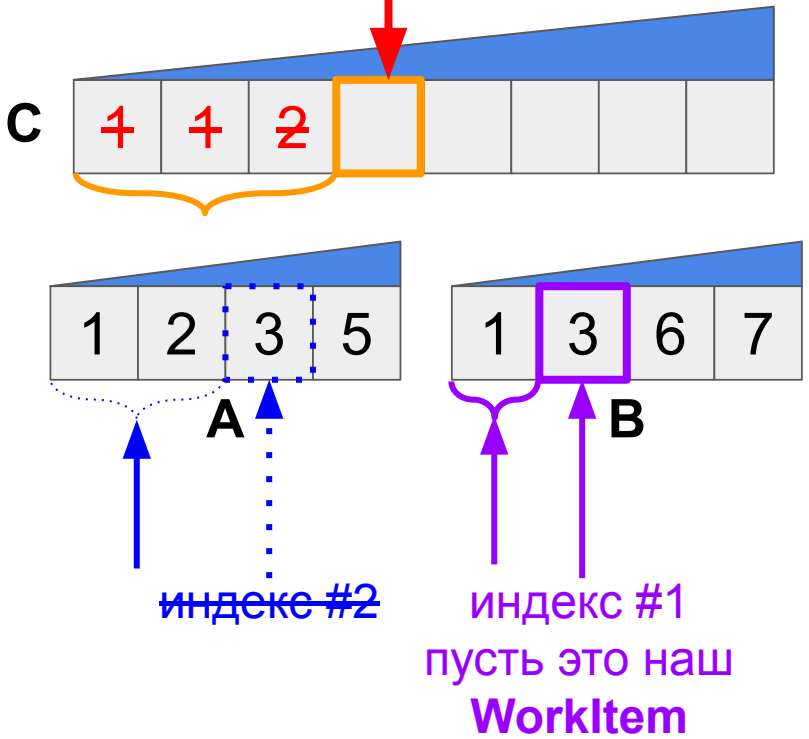
Как найти сколько чисел до нас в B?

Знаем ли мы сколько чисел до нас в A?
Бинарный поиск! O(log N)

Есть ли разница в случае когда наш WorkItem в A или в B?
Кто окажется тут?

neerc.ifmo.ru: двоичный поиск на вики ИТМО

Merge sort: параллельный алгоритм слияния (merge)



Как отсортировать на видеокарте?
Т.е. как отсортировать супер многопоточно?

Достаточно научиться делать merge!

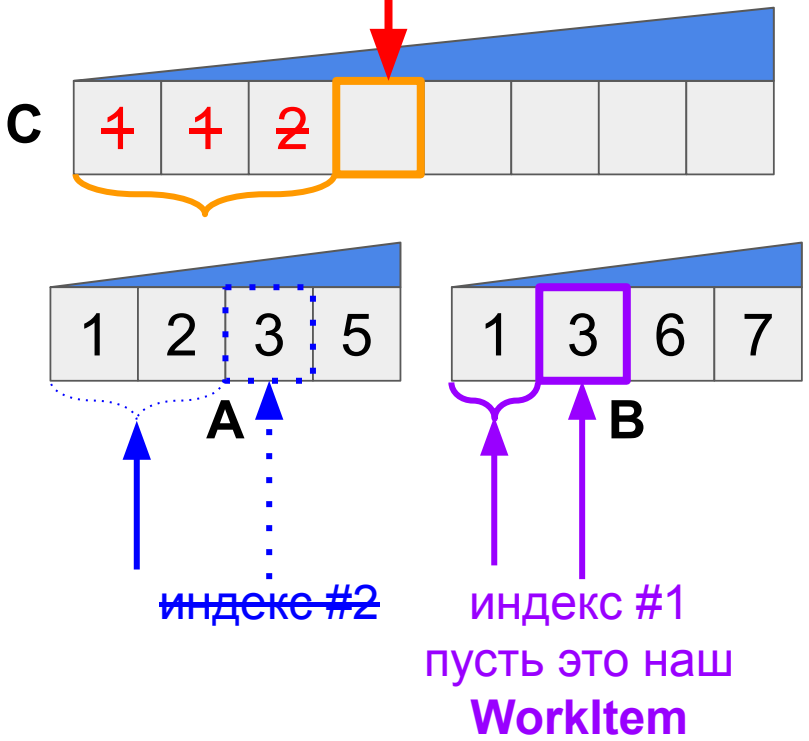
Операция слияния тривиальна если
1) знаем **указатель** в A
2) знаем **указатель** в B
3) можем найти на какой **индекс** в C
записать минимум?

Как найти сколько чисел до нас в B?

Знаем ли мы сколько чисел до нас в A?
Бинарный поиск! O(log N)

Есть ли разница в случае когда наш WorkItem в A или в B?
Кто окажется тут? Как реализовать два разных бин. поиска без *code divergence*?

Merge sort: параллельный алгоритм слияния (merge)



Как отсортировать на видеокарте?
Т.е. как отсортировать супер многопоточно?

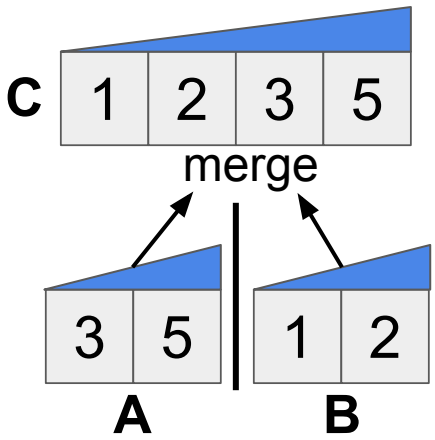
Достаточно научиться делать merge!

Операция слияния тривиальна если
1) знаем **указатель** в A
2) знаем **указатель** в B
3) можем найти на какой **индекс** в C
записать минимум?

Как найти сколько чисел до нас в B?

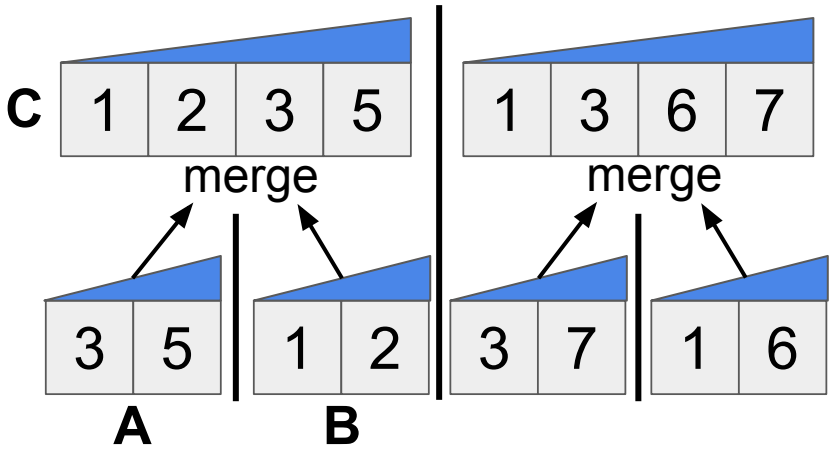
Знаем ли мы сколько чисел до нас в A?
Бинарный поиск! O(log N)

Merge sort: параллельный алгоритм слияния (merge)



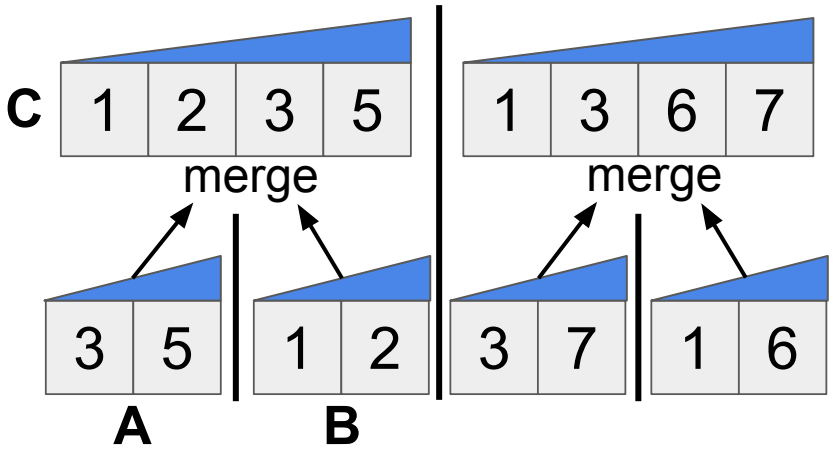
Есть ли здесь отличия в реализации?

Merge sort: параллельный алгоритм слияния (merge)



Есть ли здесь отличия в реализации?

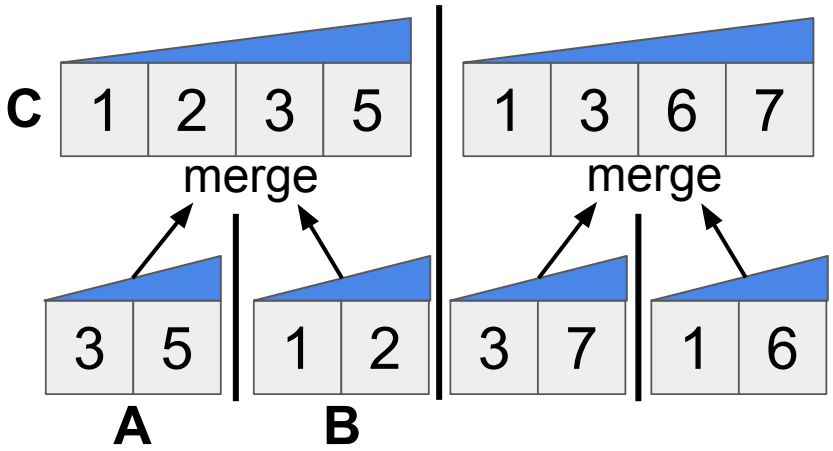
Merge sort: параллельный алгоритм слияния (merge)



Есть ли здесь отличия в реализации?

Сколько WorkItems запускаем?

Merge sort: параллельный алгоритм слияния (merge)

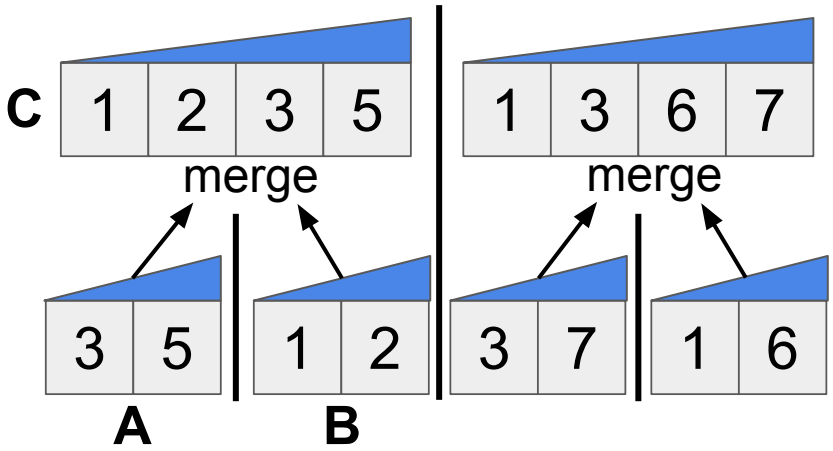


Есть ли здесь отличия в реализации?

Сколько WorkItems запускаем?

Что делает каждый WorkItem?

Merge sort: параллельный алгоритм слияния (merge)



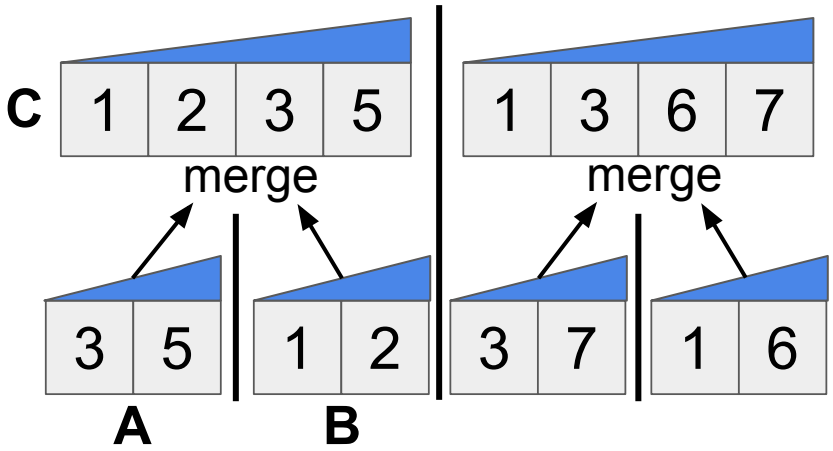
Есть ли здесь отличия в реализации?

Сколько WorkItems запускаем?

Что делает каждый WorkItem?

Какая асимптотика на CPU?

Merge sort: параллельный алгоритм слияния (merge)



Есть ли здесь отличия в реализации?

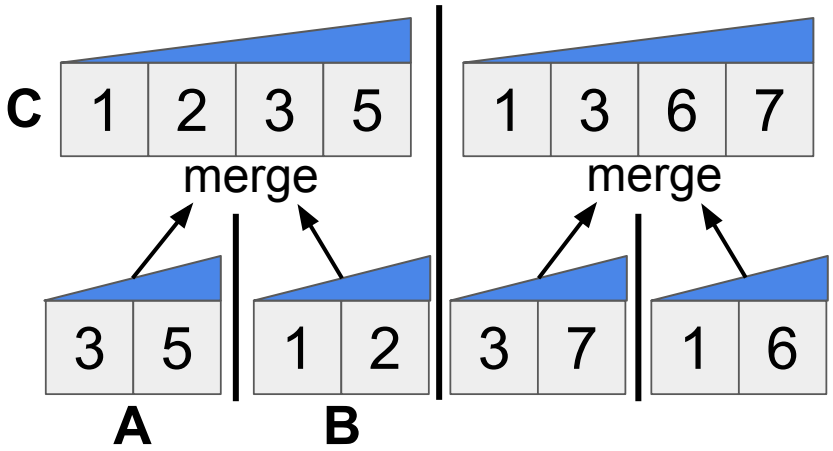
Сколько WorkItems запускаем?

Что делает каждый WorkItem?

Какая асимптотика на CPU?

O(N * log N)

Merge sort: параллельный алгоритм слияния (merge)



Есть ли здесь отличия в реализации?

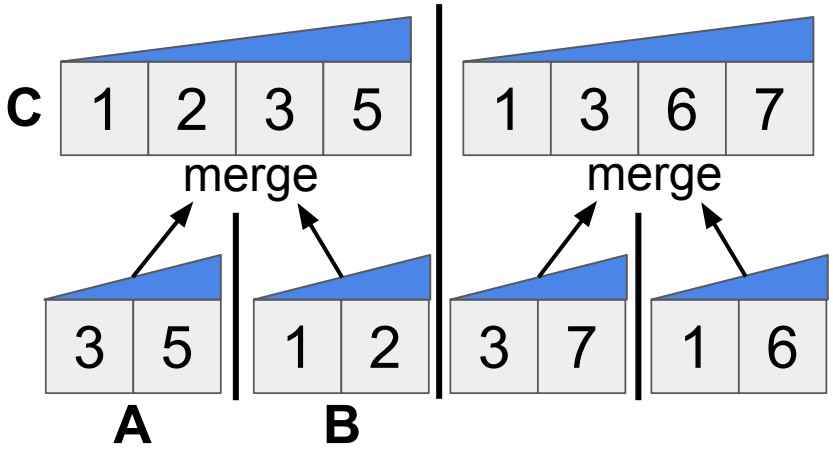
Сколько WorkItems запускаем?

Что делает каждый WorkItem?

Какая асимптотика на CPU?
O(N * log N)

Сколько запусков кернелов?
Сколько итого сравнений на GPU?
O(N * log N * log N)

Merge sort: параллельный алгоритм слияния (merge)



Есть ли здесь отличия в реализации?

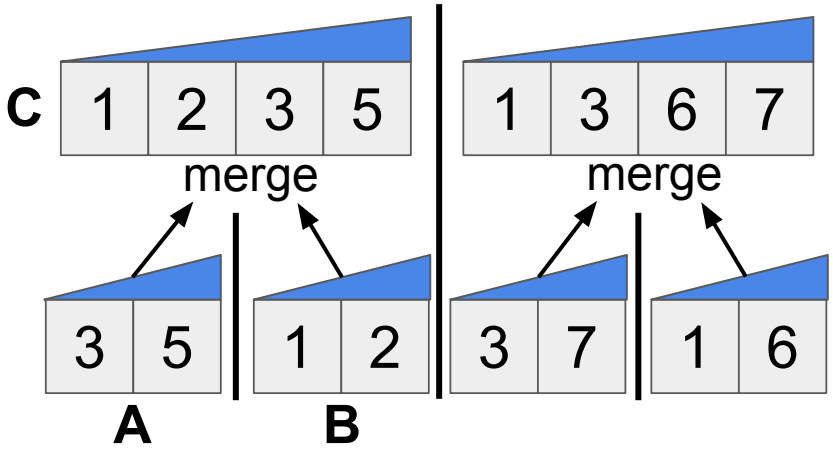
Сколько WorkItems запускаем?

Что делает каждый WorkItem?

Какая асимптотика на CPU?
O(N * log N)

Сколько запусков кернелов?
Сколько итого сравнений на GPU?
Какая итого скорость на GPU?
O(N * log N * log N / K)

Merge sort: параллельный алгоритм слияния (merge)



Стабильная ли это сортировка?

Есть ли здесь отличия в реализации?

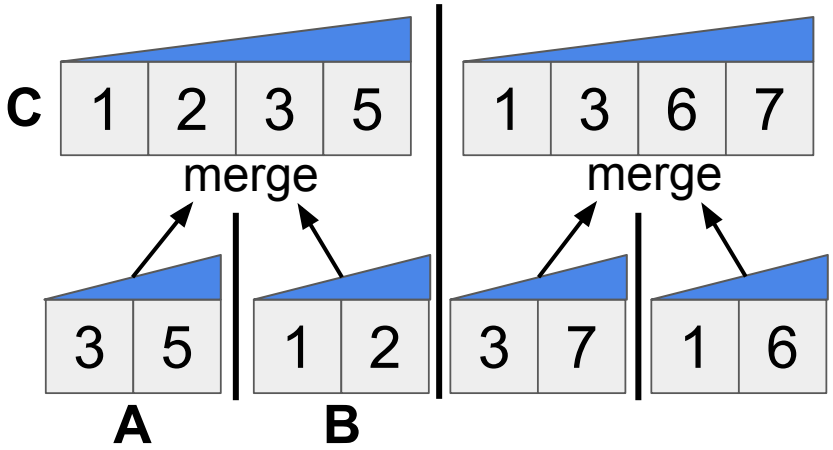
Сколько WorkItems запускаем?

Что делает каждый WorkItem?

Какая асимптотика на CPU?
O(N * log N)

Сколько запусков кернелов?
Сколько итого сравнений на GPU?
Какая итого скорость на GPU?
O(N * log N * log N / K)

Merge sort: параллельный алгоритм слияния (merge)



Стабильная ли это сортировка?

Какие отличия в свойствах от radix?

Есть ли здесь отличия в реализации?

Сколько WorkItems запускаем?

Что делает каждый WorkItem?

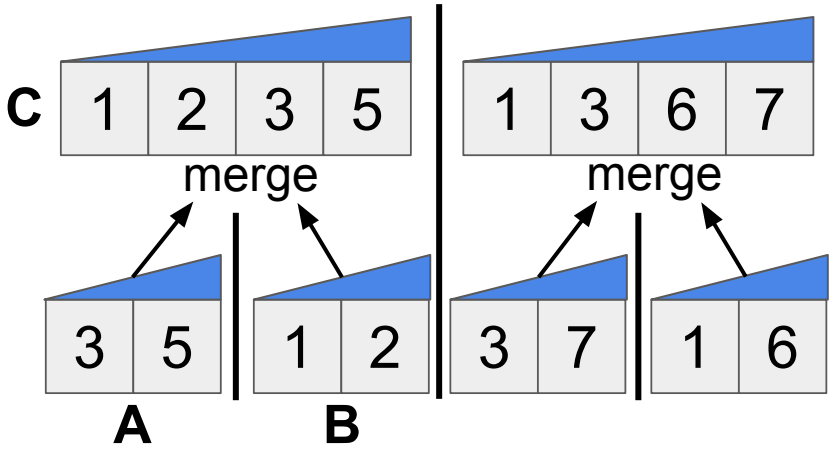
Какая асимптотика на CPU?
O(N * log N)

Сколько запусков кернелов?
Сколько итого сравнений на GPU?
Какая итого скорость на GPU?
O(N * log N * log N / K)

Merge sort: параллельный алгоритм слияния (merge)

Как ускорить?

Двухуровневая иерархия!



Стабильная ли это сортировка?

Какие отличия в свойствах от radix?

Есть ли здесь отличия в реализации?

Сколько WorkItems запускаем?

Что делает каждый WorkItem?

Какая асимптотика на CPU?

O(N * log N)

Сколько запусков кернелов?

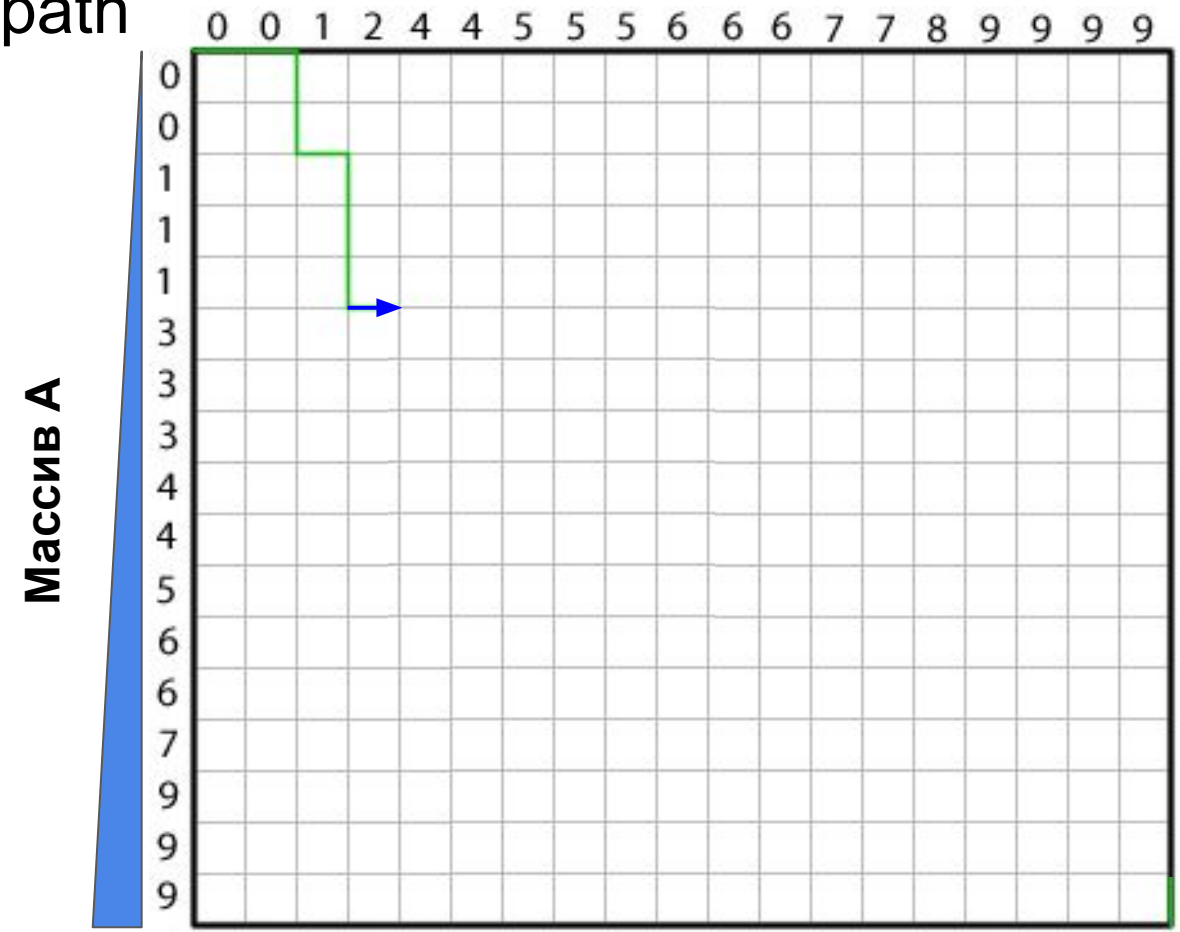
Сколько итого сравнений на GPU?

Какая итого скорость на GPU?

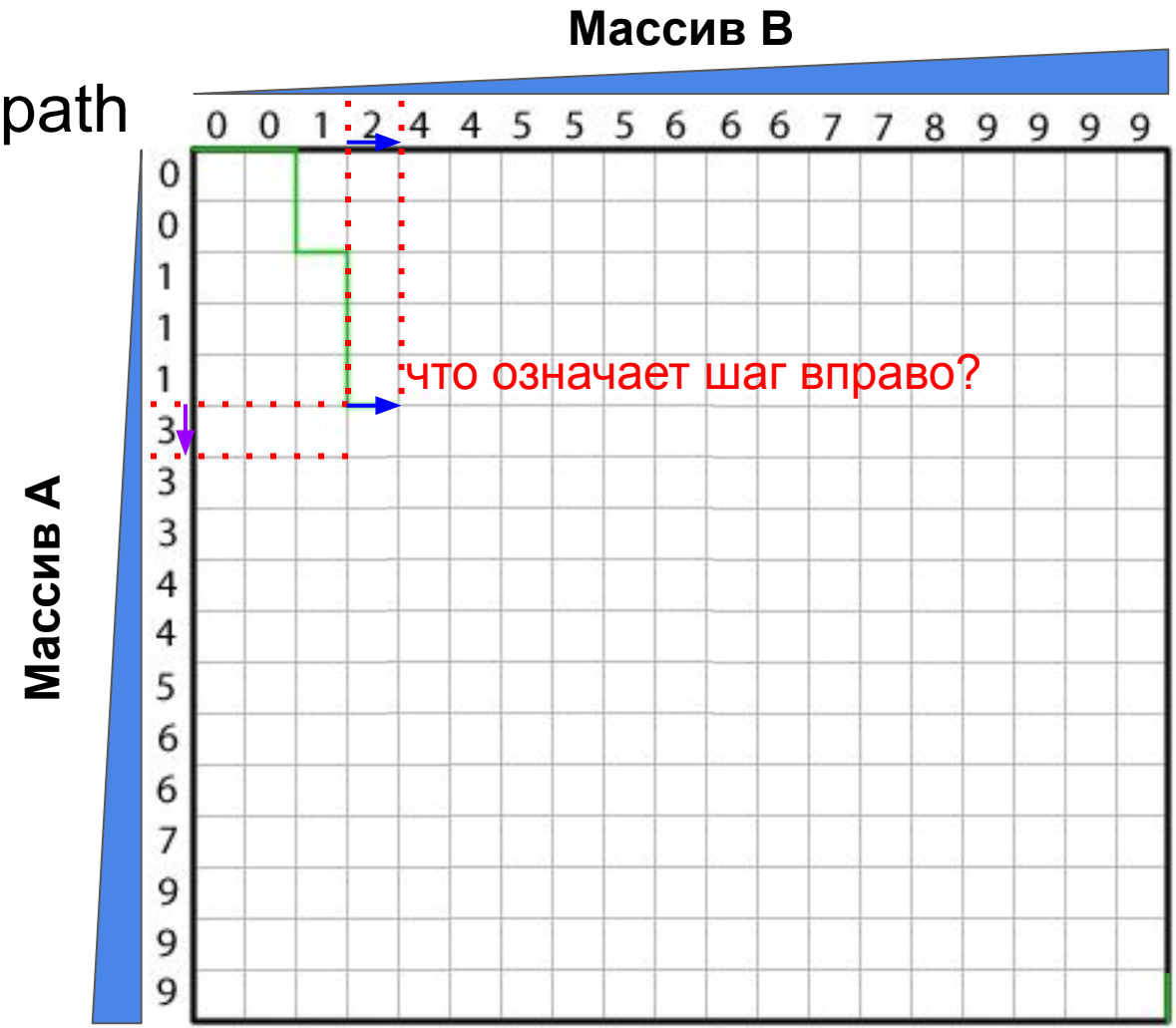
O(N * log N * log N / K)

Глава 2: Merge path

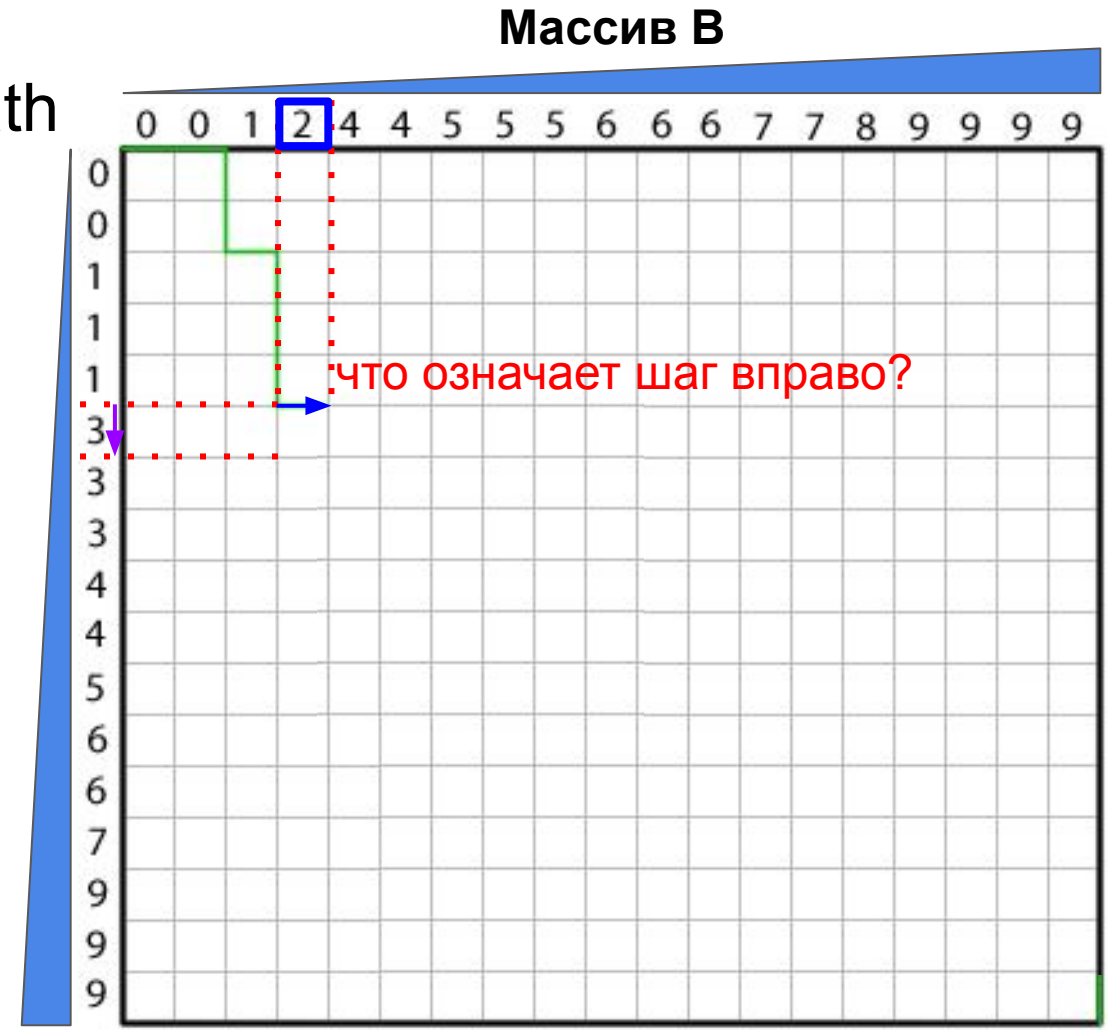
Merge path



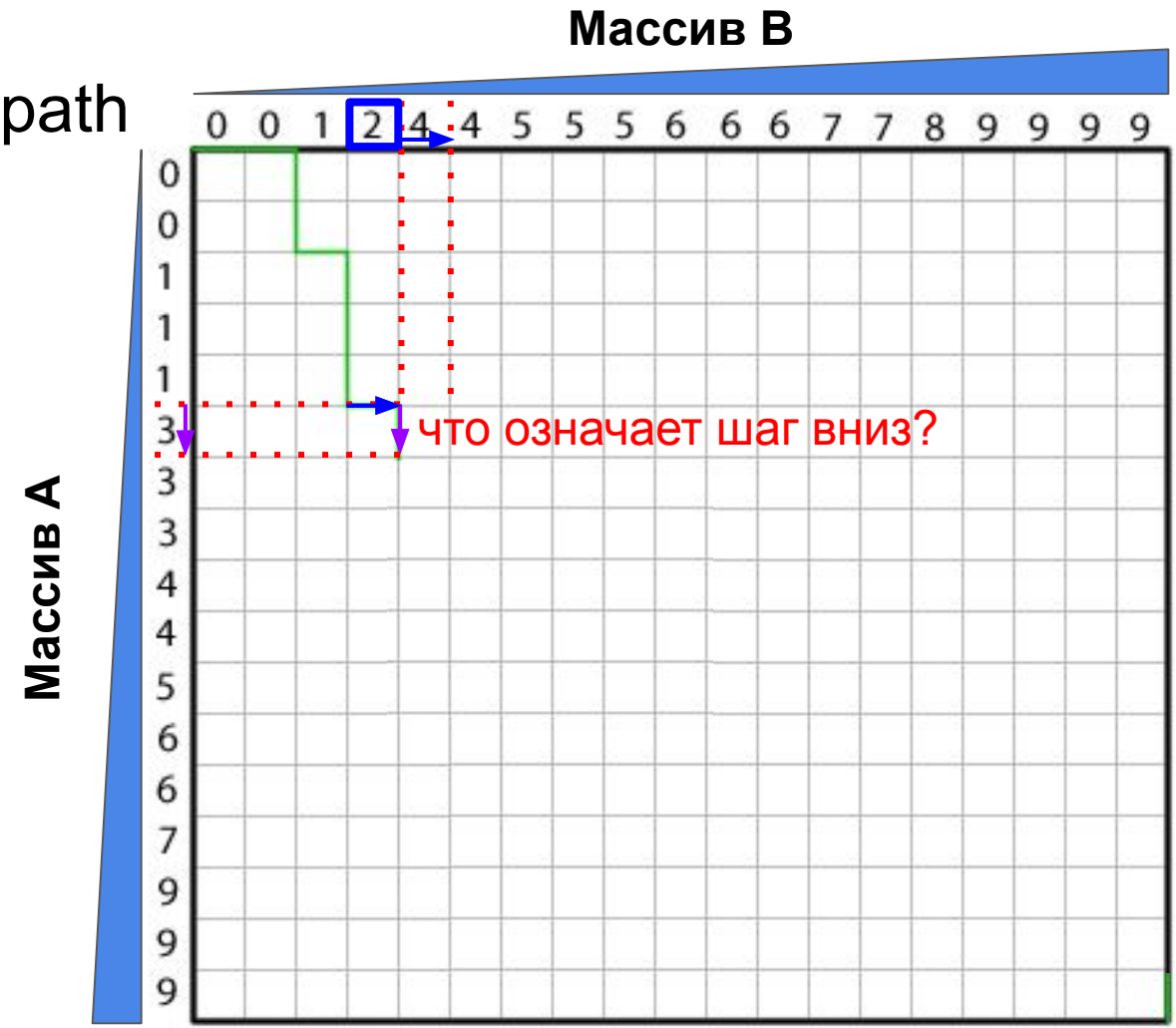
Merge path


Массив B
0 0 1 2 4 4 5 5 5 6 6 6 7 7 8 9 9 9 9
Массив A
0 0 1 1 1 3 3 3 4 4 5 6 6 7 9 9 9
что означает шаг вправо?

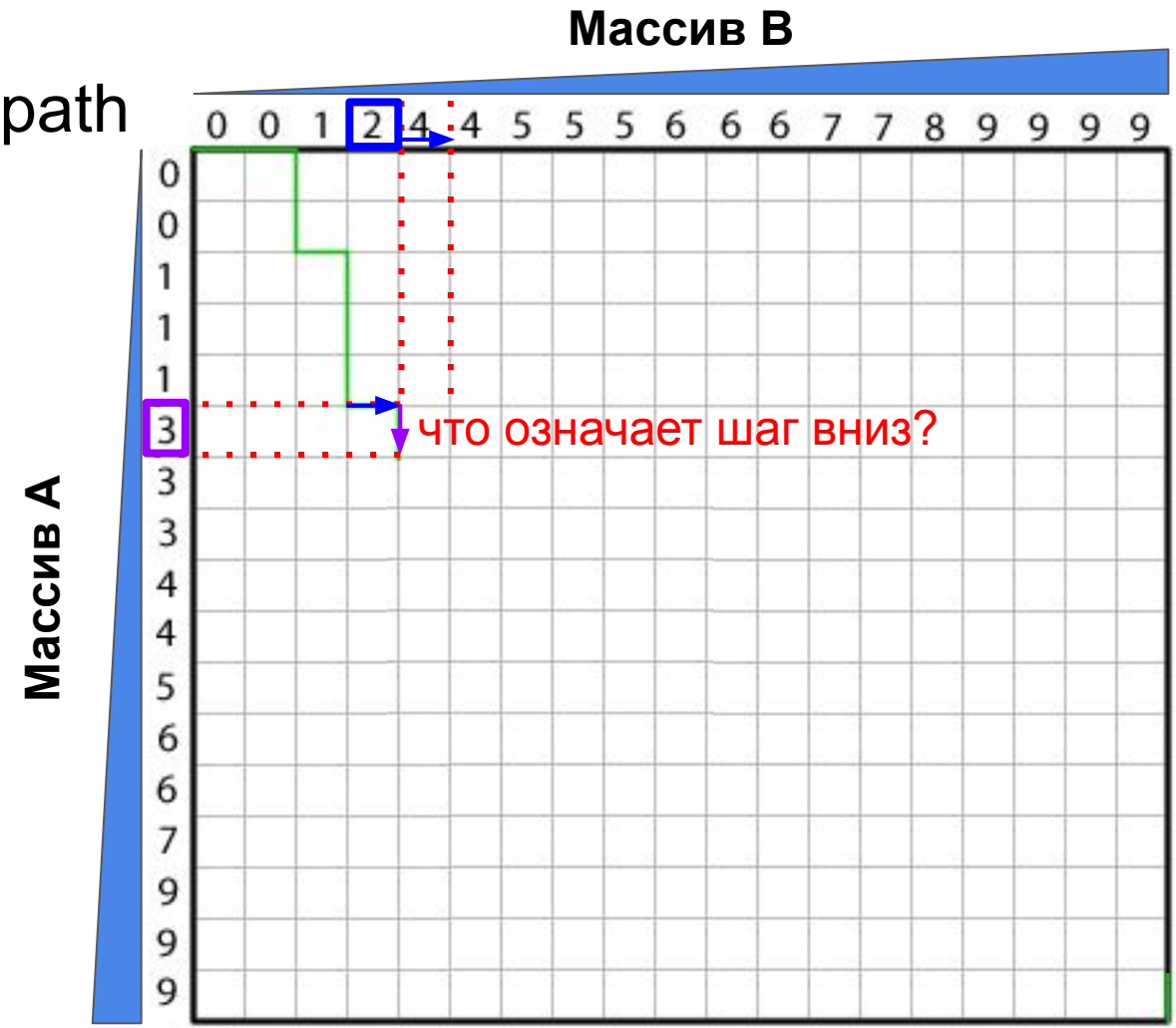
Merge path


Массив B
0 0 1 2 4 4 5 5 5 6 6 6 7 7 8 9 9 9 9
Массив A
0 0 1 1 1 3 3 3 4 4 5 6 6 7 9 9 9
что означает шаг вправо?

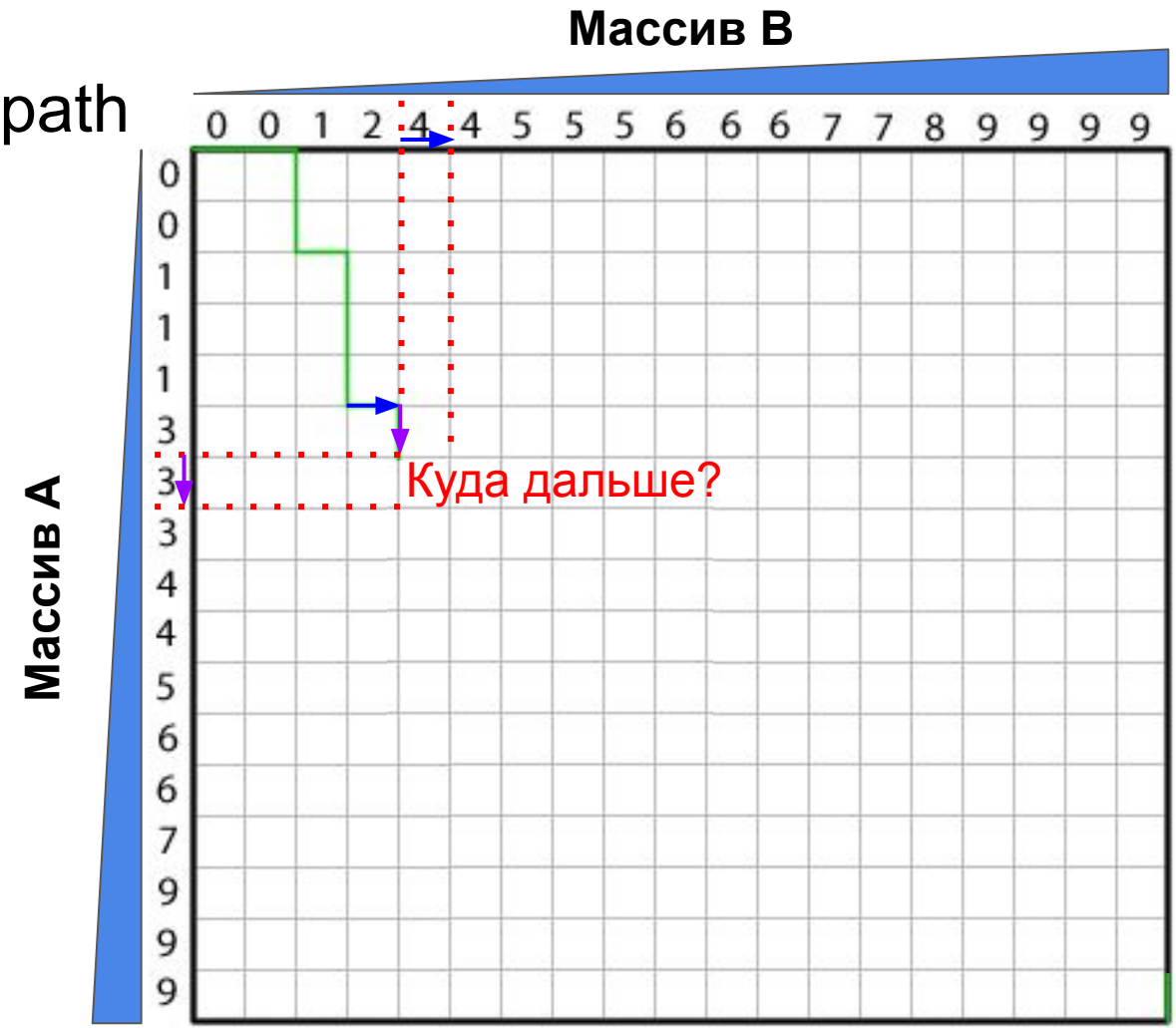
Merge path


Массив B
0 0 1 2 4 4 5 5 5 6 6 6 7 7 8 9 9 9 9
Массив A
0 0 1 1 1 3 3 3 4 4 5 6 6 7 9 9 9
что означает шаг вниз?

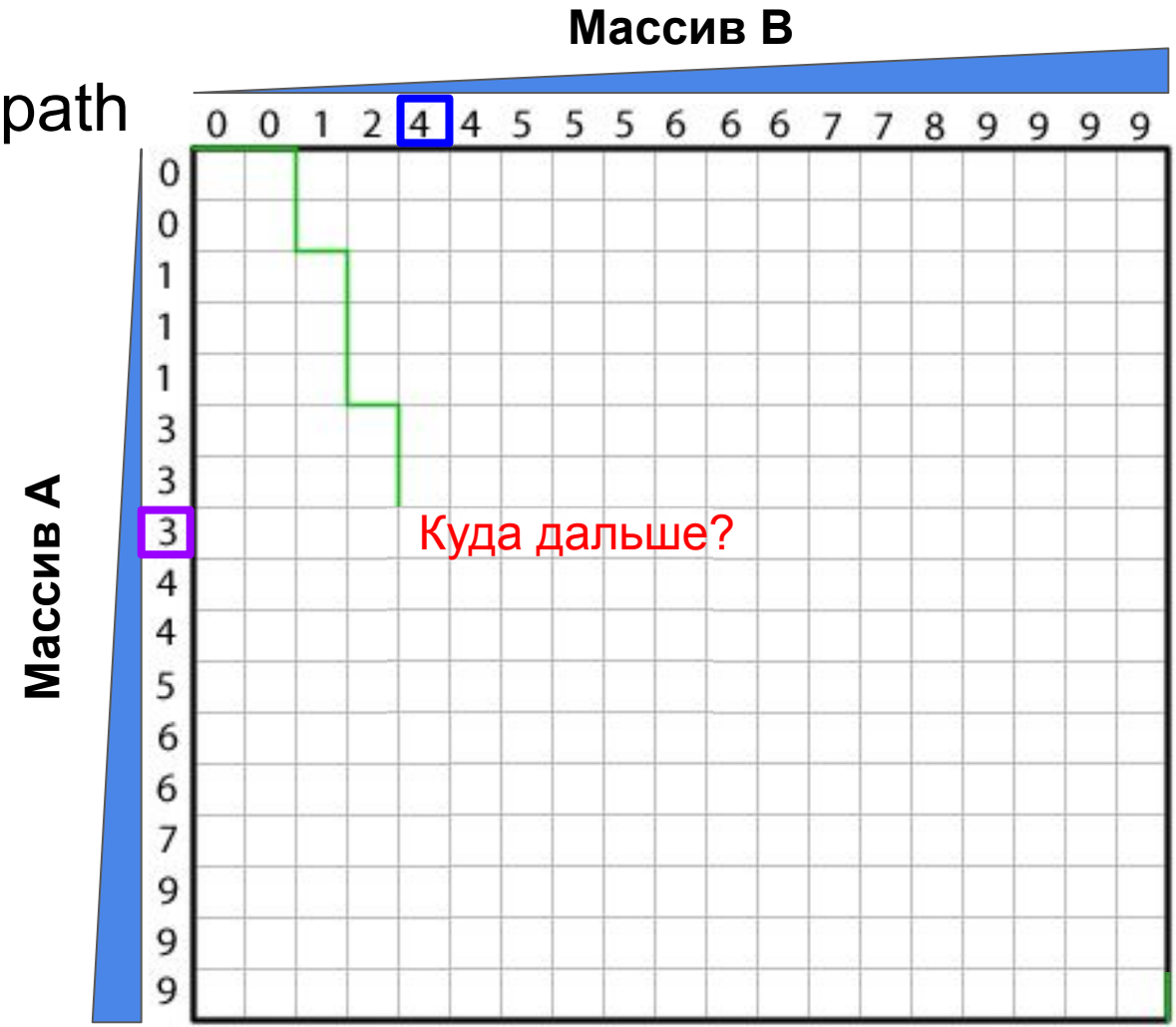
Merge path


Массив B
0 0 1 2 4 4 5 5 5 6 6 6 7 7 8 9 9 9 9
Массив A
0 0 1 1 1 3 3 3 4 4 5 6 6 7 9 9 9
что означает шаг вниз?

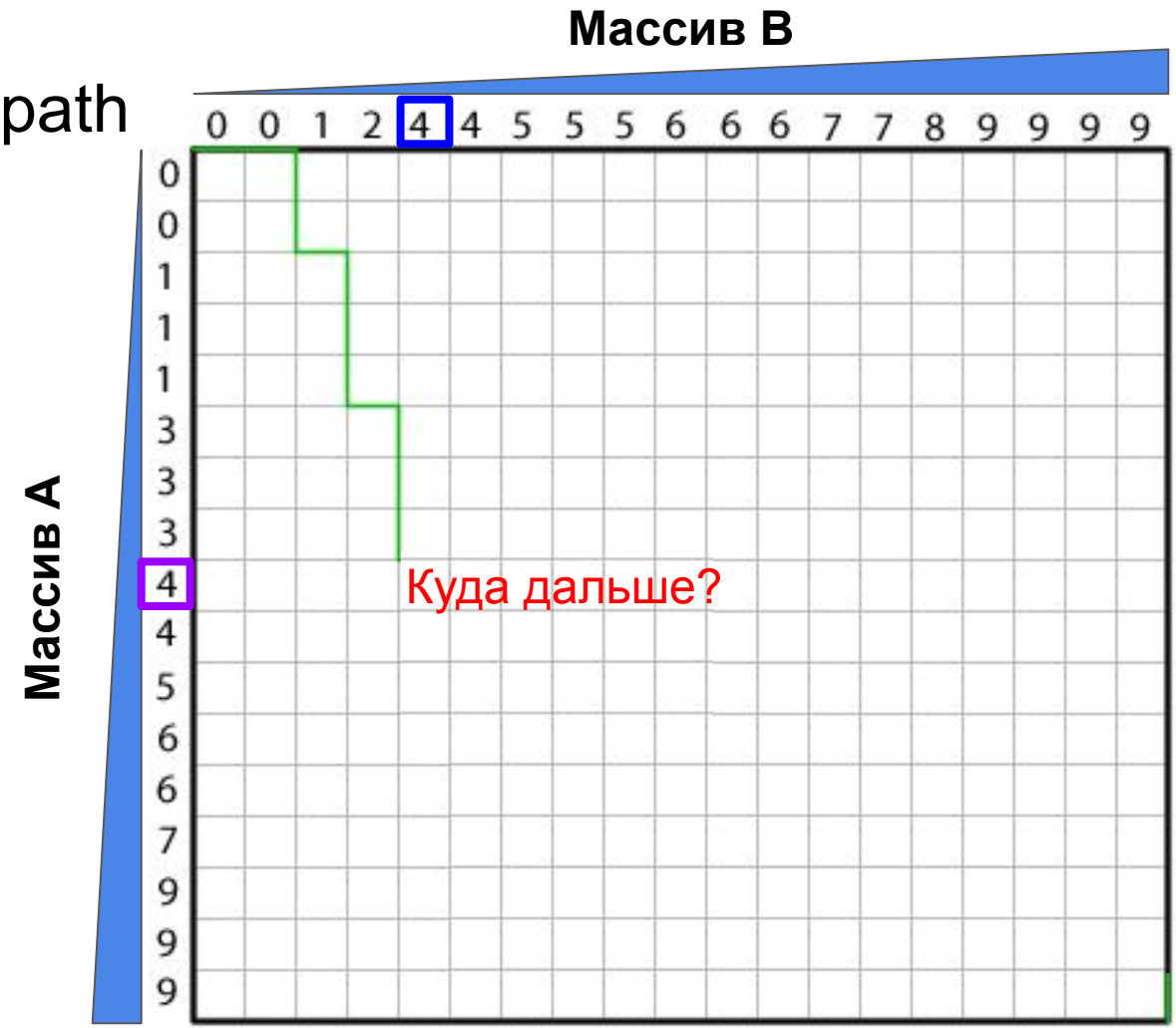
Merge path


Массив B
0 0 1 2 4 4 5 5 5 6 6 6 7 7 8 9 9 9 9
Массив A
0 0 1 1 1 3 3 3 4 4 5 6 6 7 9 9 9
Куда дальше?

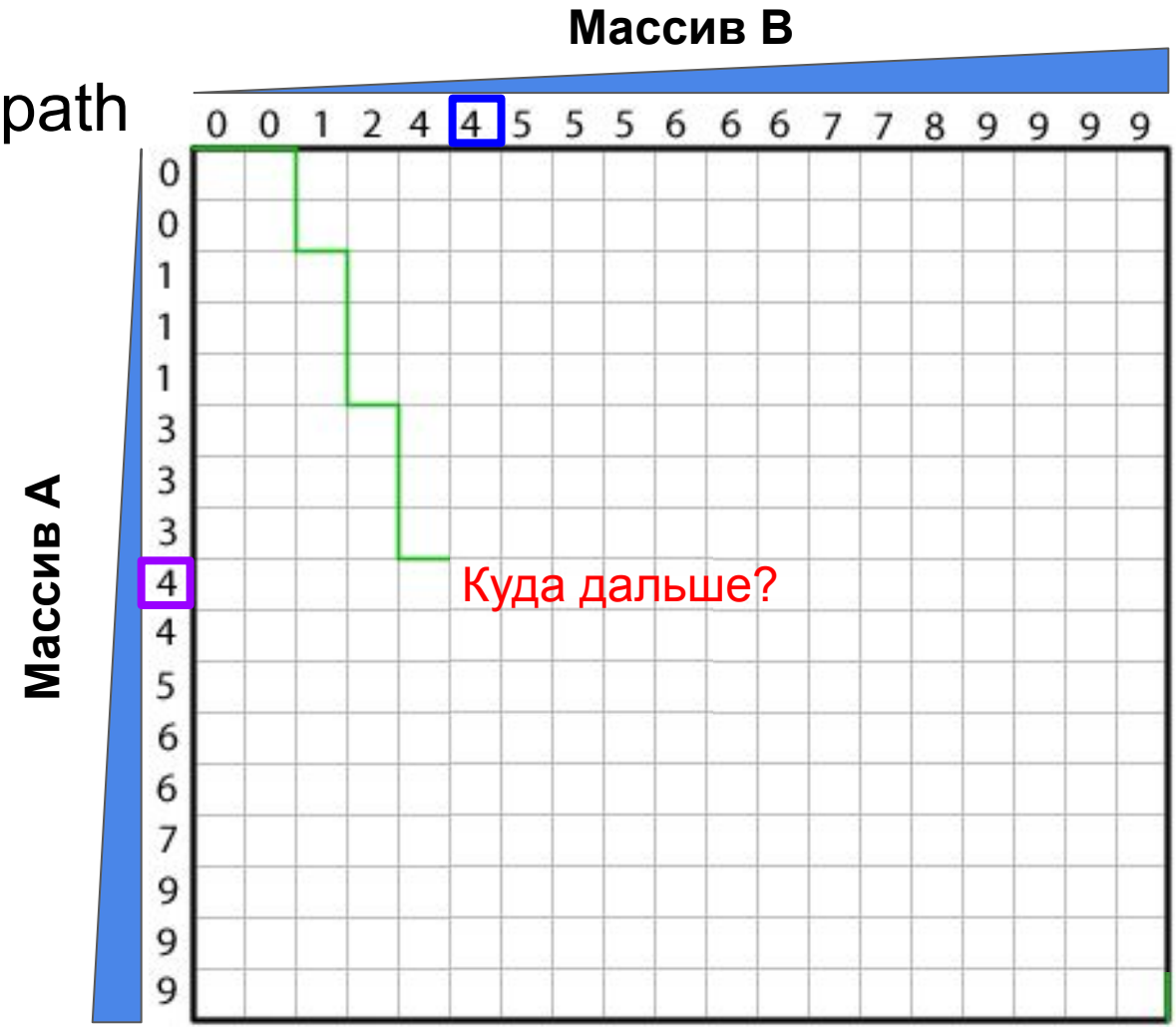
Merge path


Массив B
0 0 1 2 4 4 5 5 5 6 6 6 7 7 8 9 9 9 9
Массив A
0 0 1 1 1 3 3 3 4 4 5 6 6 7 9 9 9
Куда дальше?

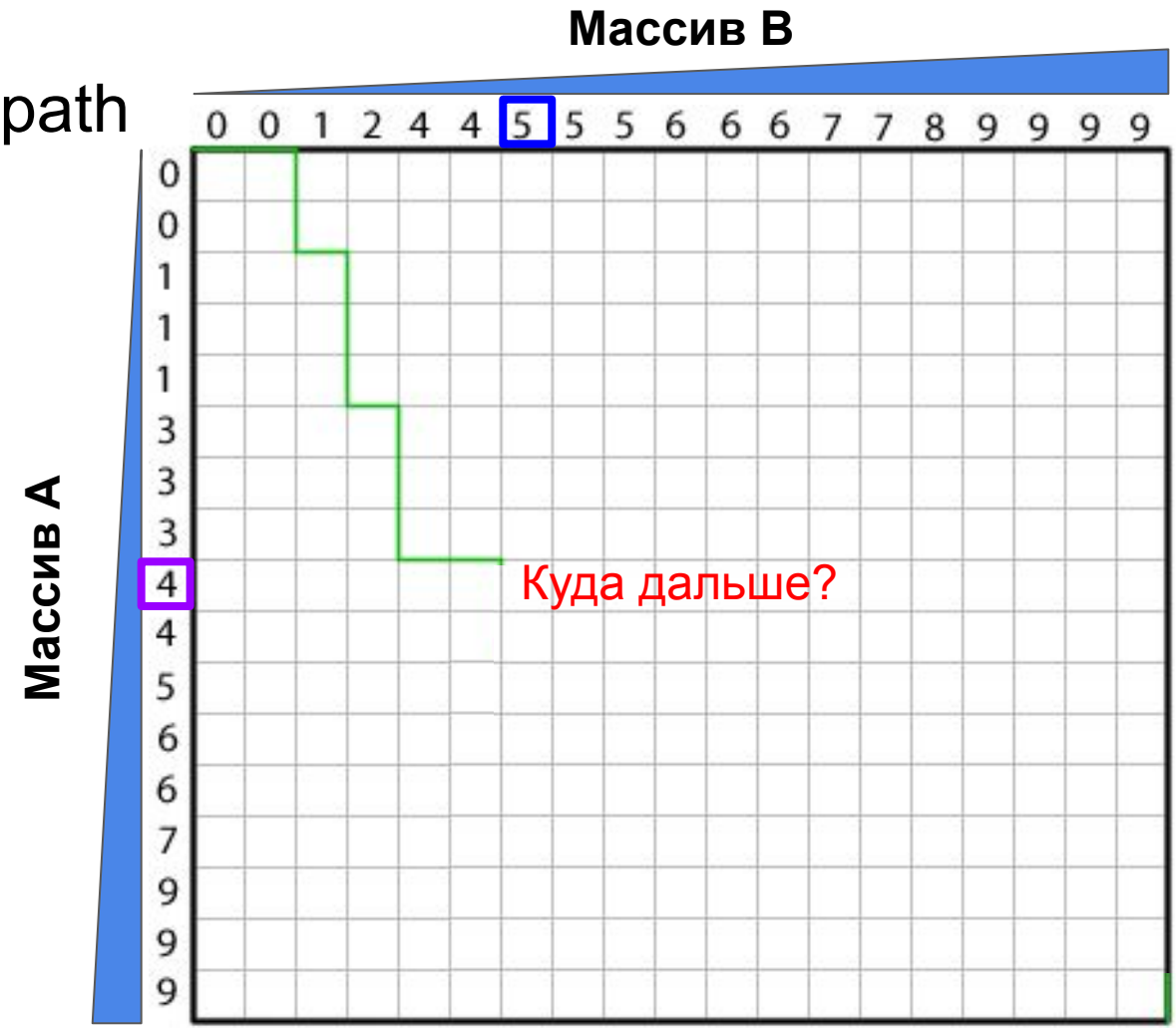
Merge path


Массив B
0 0 1 2 4 4 5 5 5 6 6 6 7 7 8 9 9 9 9
Массив A
0 0 1 1 1 3 3 3 4 4 5 6 6 7 9 9 9
Куда дальше?

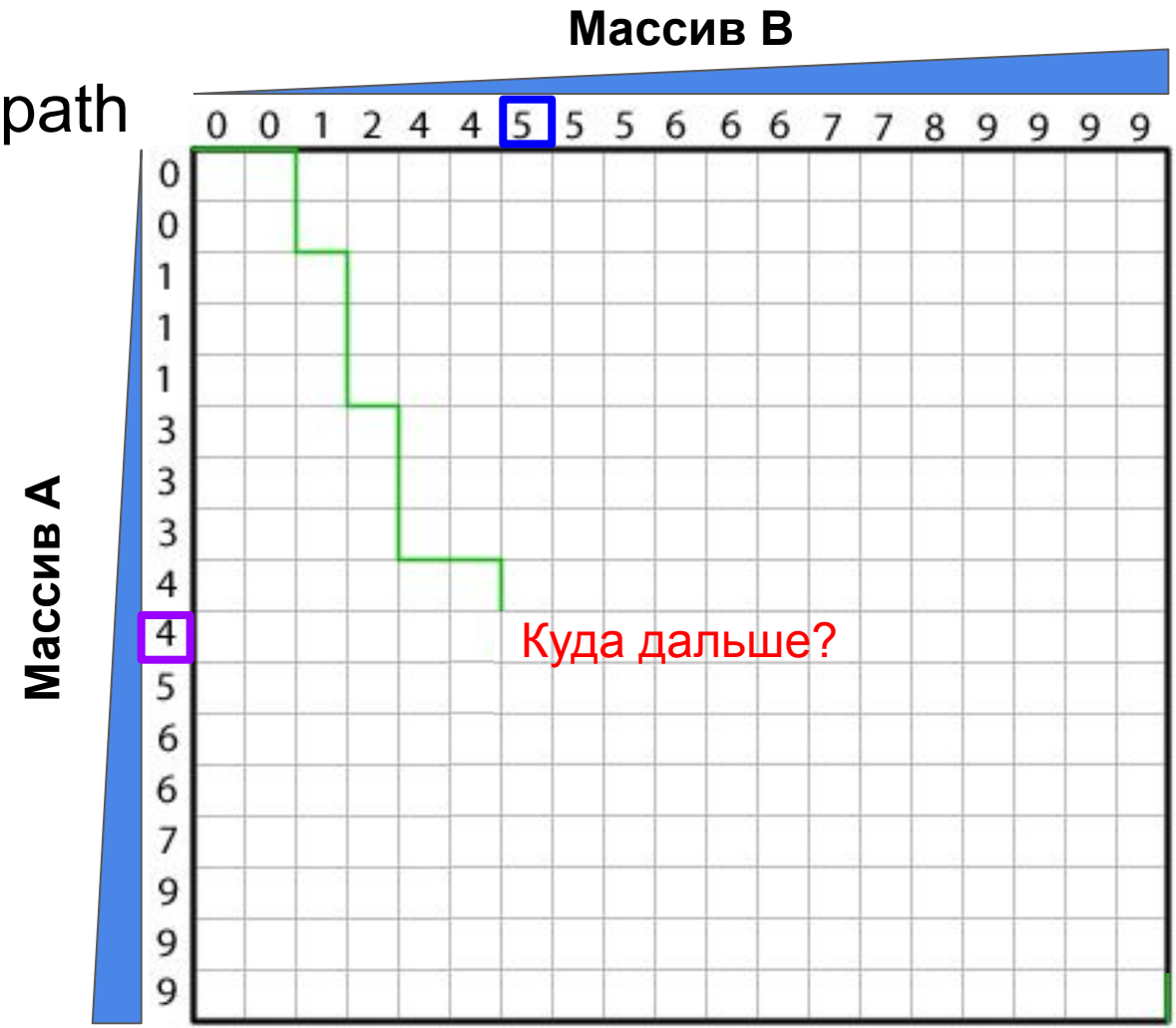
Merge path


Массив B
0 0 1 2 4 4 5 5 5 6 6 6 7 7 8 9 9 9 9
Массив A
0 0 1 1 1 3 3 3 4 4 5 6 6 7 9 9 9
Куда дальше?

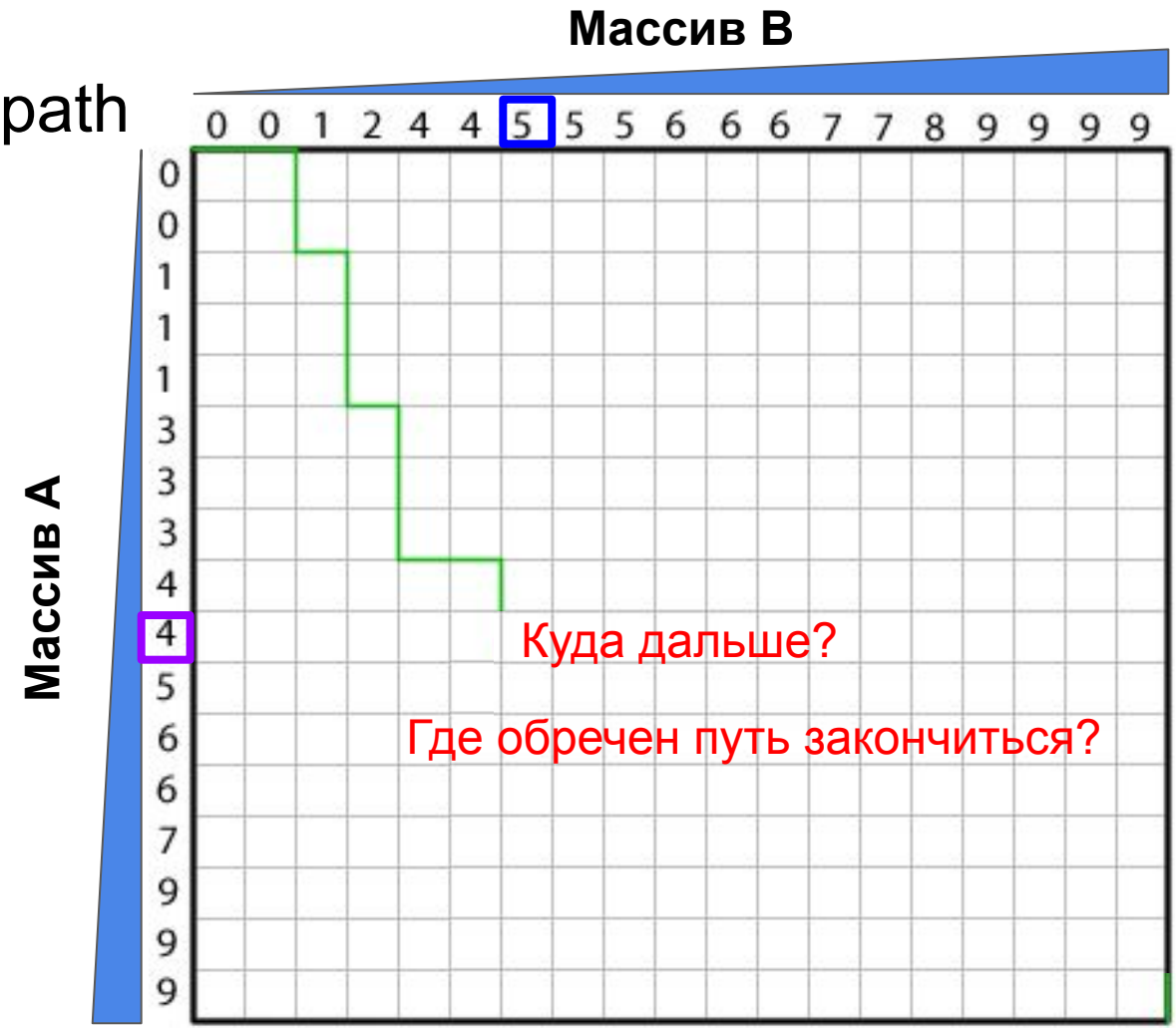
Merge path


Массив B
0 0 1 2 4 4 5 5 5 6 6 6 7 7 8 9 9 9 9
Массив A
0 0 1 1 1 3 3 3 4 4 5 6 6 7 9 9 9
Куда дальше?

Merge path


Массив B
0 0 1 2 4 4 5 5 5 6 6 6 7 7 8 9 9 9 9
Массив A
0 0 1 1 1 3 3 3 4 4 5 6 6 7 9 9 9
Куда дальше?

Merge path


Массив B
0 0 1 2 4 4 5 5 5 6 6 6 7 7 8 9 9 9 9
Массив A
0 0 1 1 1 3 3 3 4 4 5 6 6 7 9 9 9
Куда дальше?
Где обречен путь закончиться?

Merge path


Массив B
0 0 1 2 4 4 5 5 5 6 6 6 7 7 8 9 9 9 9
Массив A
0 0 1 1 1 3 3 3 4 4 5 6 6 7 9 9 9
Как на базе этого концепта реализовать слияние в модели массового параллелизма?

Merge path


Массив B
0 0 1 2 4 4 5 5 5 6 6 6 7 7 8 9 9 9 9
Массив A
0 0 1 1 1 3 3 3 4 4 5 6 6 7 9 9 9
Как на базе этого концепта реализовать слияние в модели массового параллелизма?
?
Слева больше чем сверху?

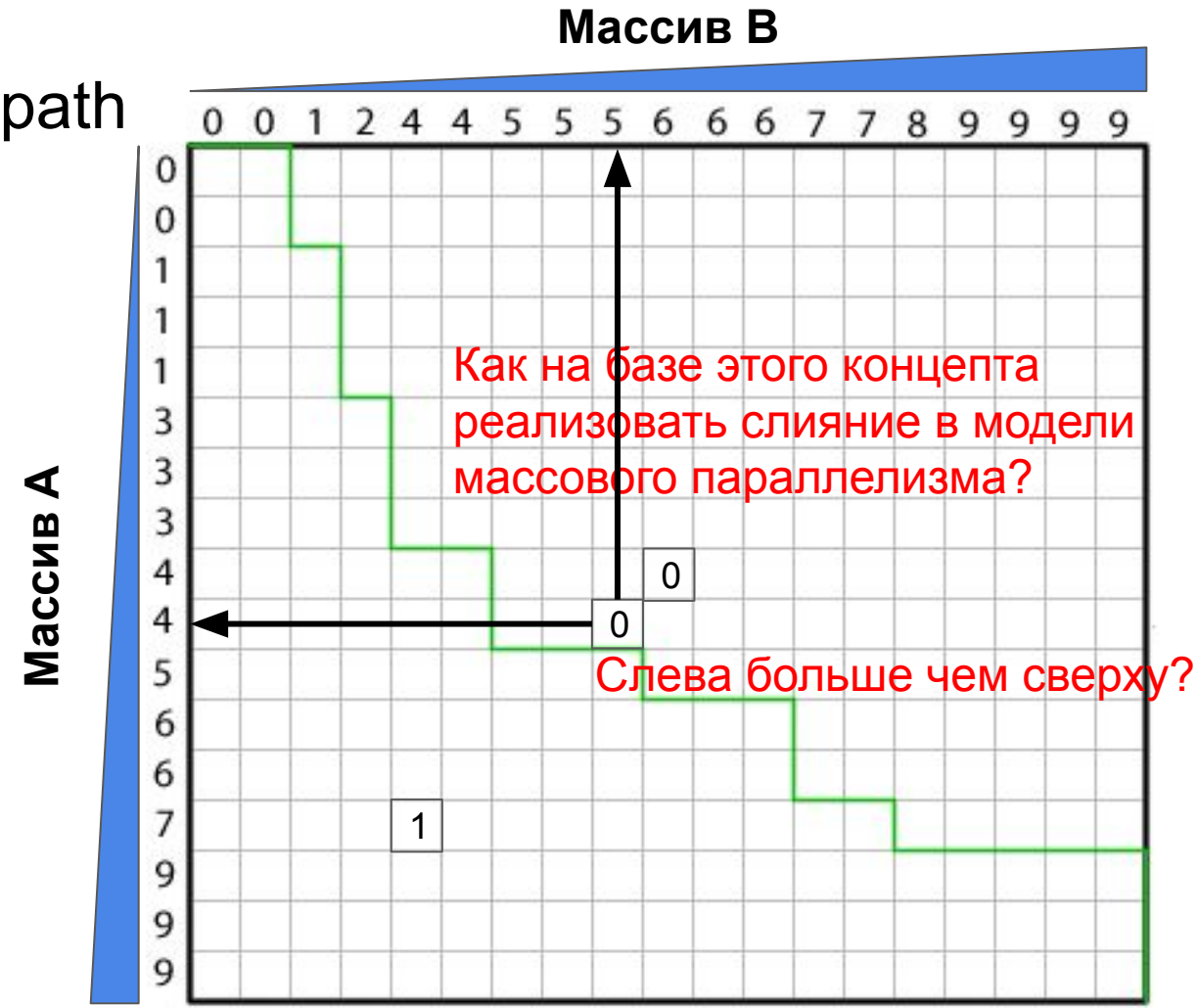
Merge path


Массив B
0 0 1 2 4 4 5 5 5 6 6 6 7 7 8 9 9 9 9
Массив A
0 0 1 1 1 3 3 3 4 4 5 6 6 7 9 9 9
Как на базе этого концепта реализовать слияние в модели массового параллелизма?
1
Слева больше чем сверху?

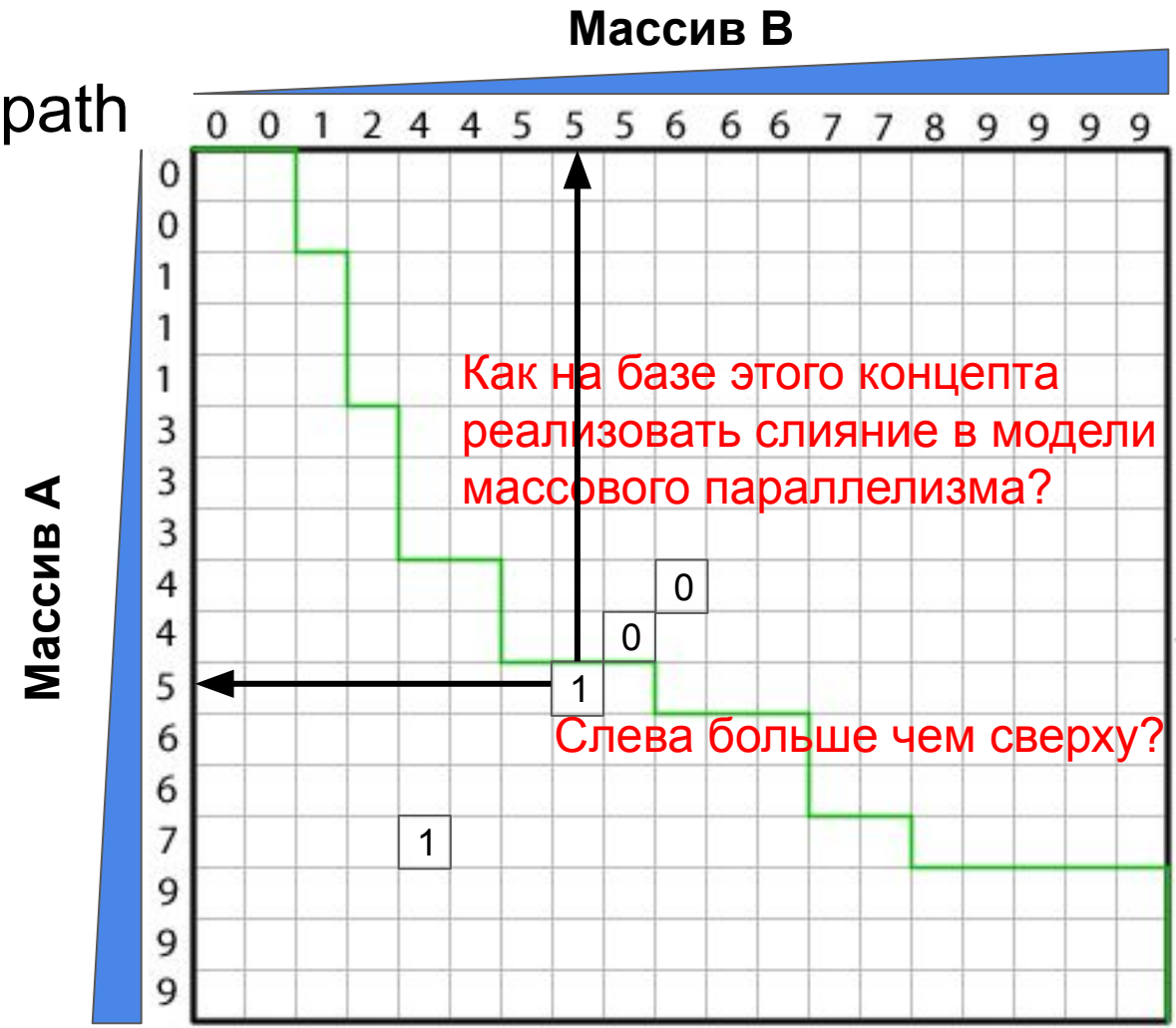
Merge path


Массив B
0 0 1 2 4 4 5 5 5 6 6 6 7 7 8 9 9 9 9
Массив A
0 0 1 1 1 3 3 3 4 4 5 6 6 7 9 9 9
Как на базе этого концепта реализовать слияние в модели массового параллелизма?
0
Слева больше чем сверху?
1

Merge path


Массив B
0 0 1 2 4 4 5 5 5 6 6 6 7 7 8 9 9 9 9
Массив A
0 0 1 1 1 3 3 3 4 4 5 6 6 7 9 9 9
Как на базе этого концепта реализовать слияние в модели массового параллелизма?
0
0
Слева больше чем сверху?
1

Merge path


Массив B
0 0 1 2 4 4 5 5 5 6 6 6 7 7 8 9 9 9 9
Массив A
0 0 1 1 1 3 3 3 4 4 5 6 6 7 9 9 9
Как на базе этого концепта реализовать слияние в модели массового параллелизма?
0
0
1
Слева больше чем сверху?
1

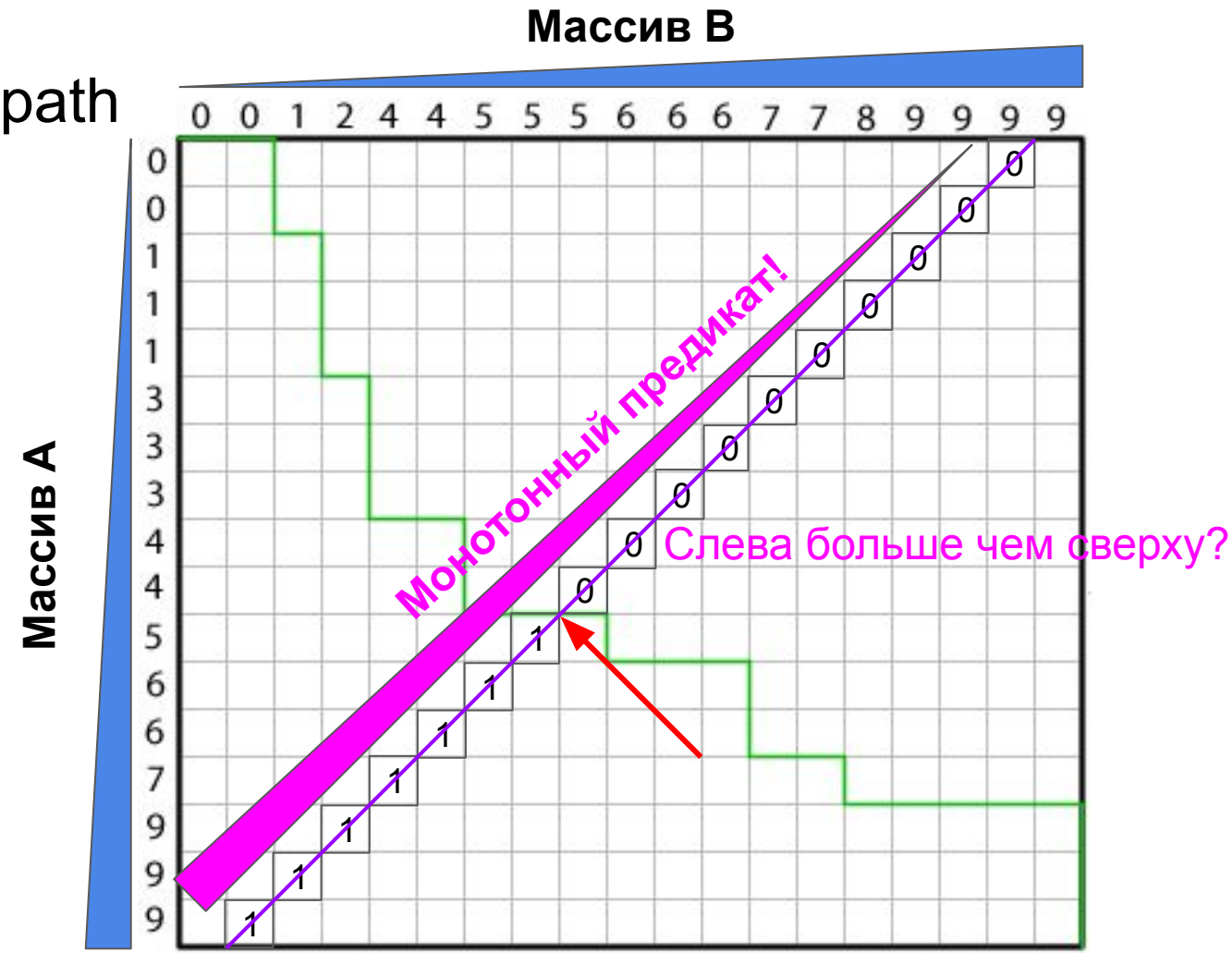
Merge path


Массив B
0 0 1 2 4 4 5 5 5 6 6 6 7 7 8 9 9 9 9
Массив A
0 0 1 1 1 3 3 3 4 4 5 6 6 7 9 9 9
Как на базе этого концепта реализовать слияние в модели массового параллелизма?
0
0
1
Слева больше чем сверху?
1
Как выглядит все поле?

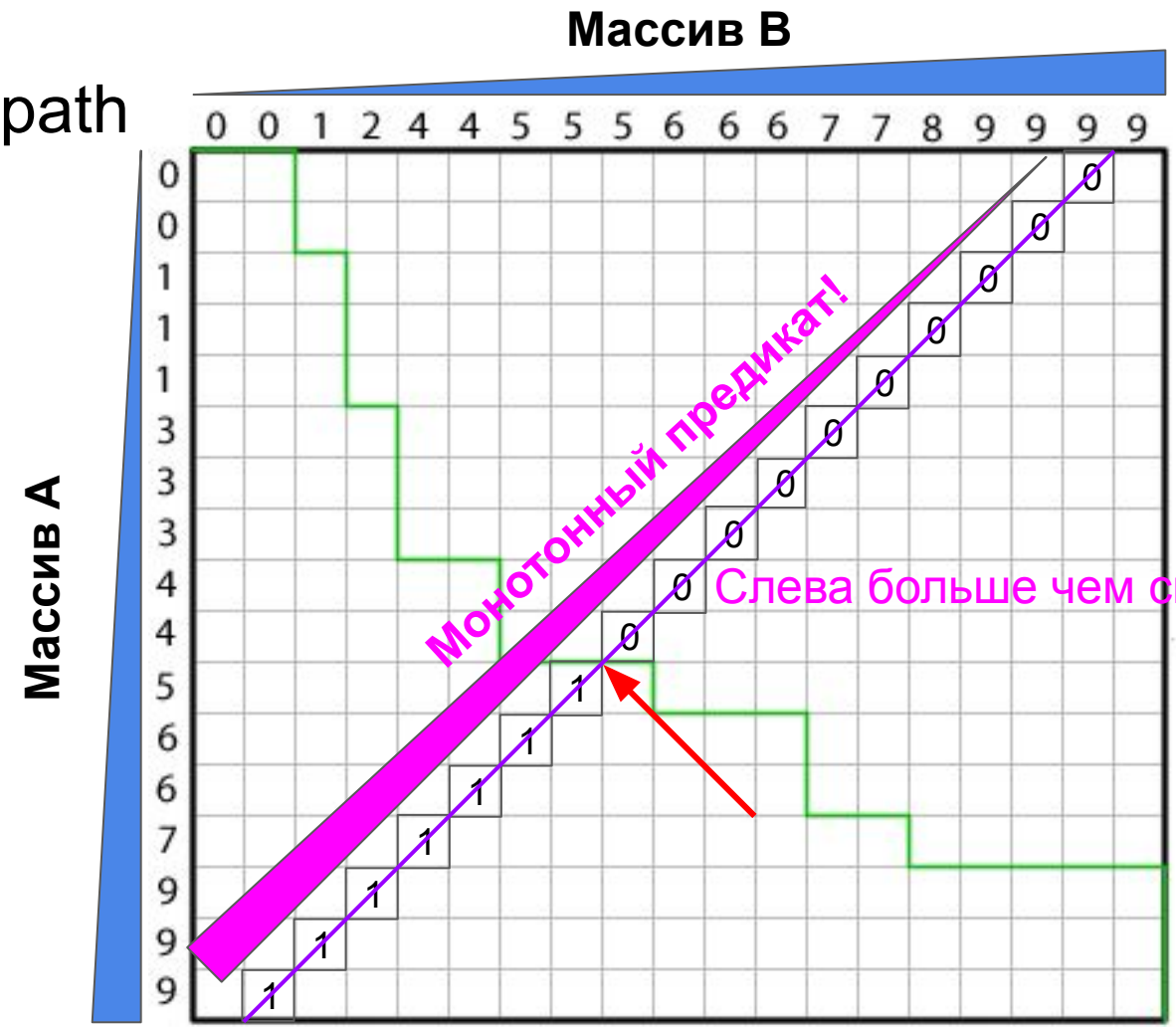
Merge path


Массив B
0 0 1 2 4 4 5 5 5 6 6 6 7 7 8 9 9 9 9
Массив A
0 0 1 1 1 3 3 4 4 5 6 6 7 9 9 9
0
1
Как на базе этого концепта реализовать слияние в модели массового параллелизма?
Слева больше чем сверху?
Как выглядит все поле?

Merge path

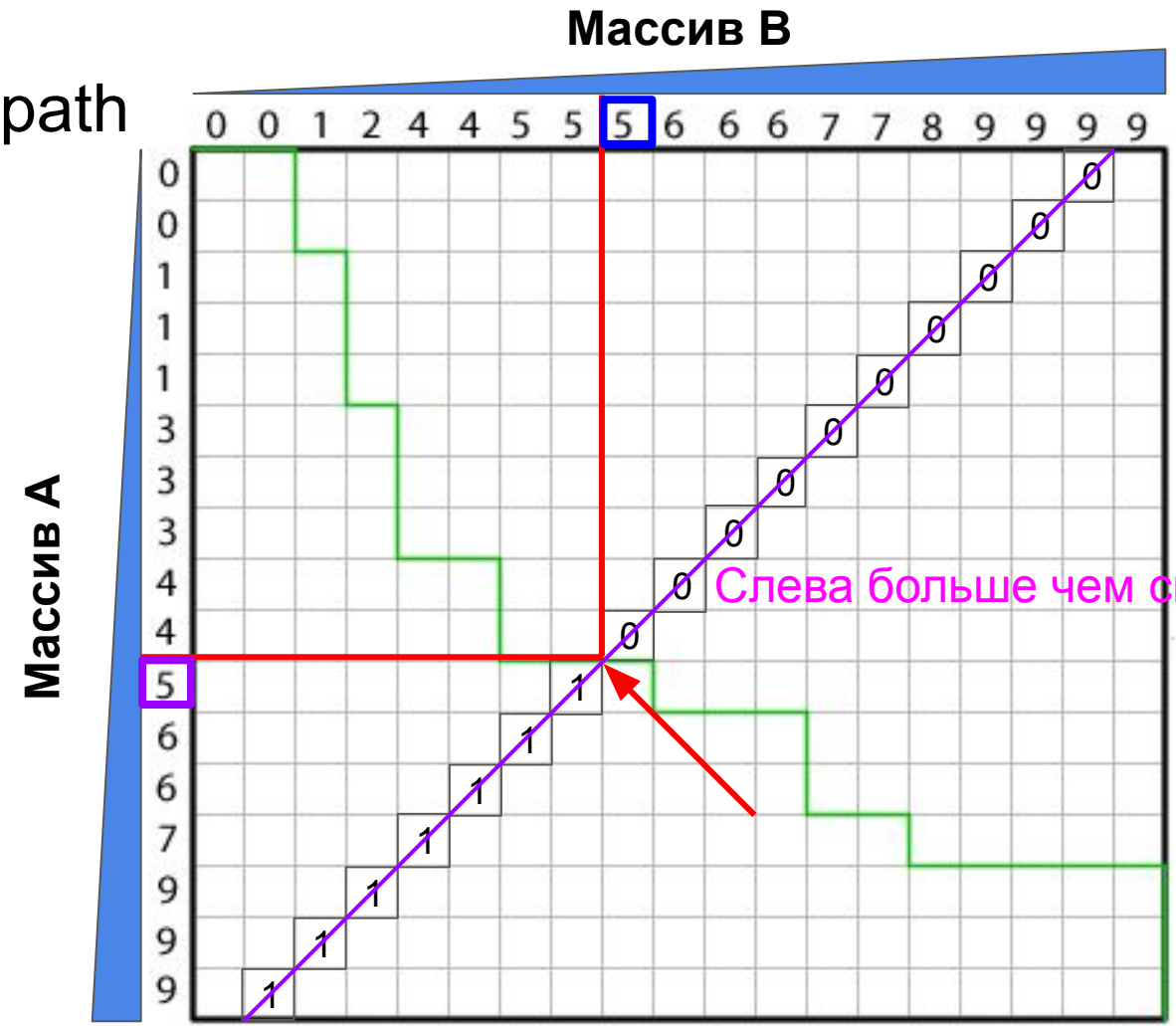

Массив B
0 0 1 2 4 4 5 5 5 6 6 6 7 7 8 9 9 9 9
Массив A
0 0 1 1 1 3 3 3 4 4 5 6 6 7 9 9 9
Монотонный предикат!
Слева больше чем сверху?

Merge path



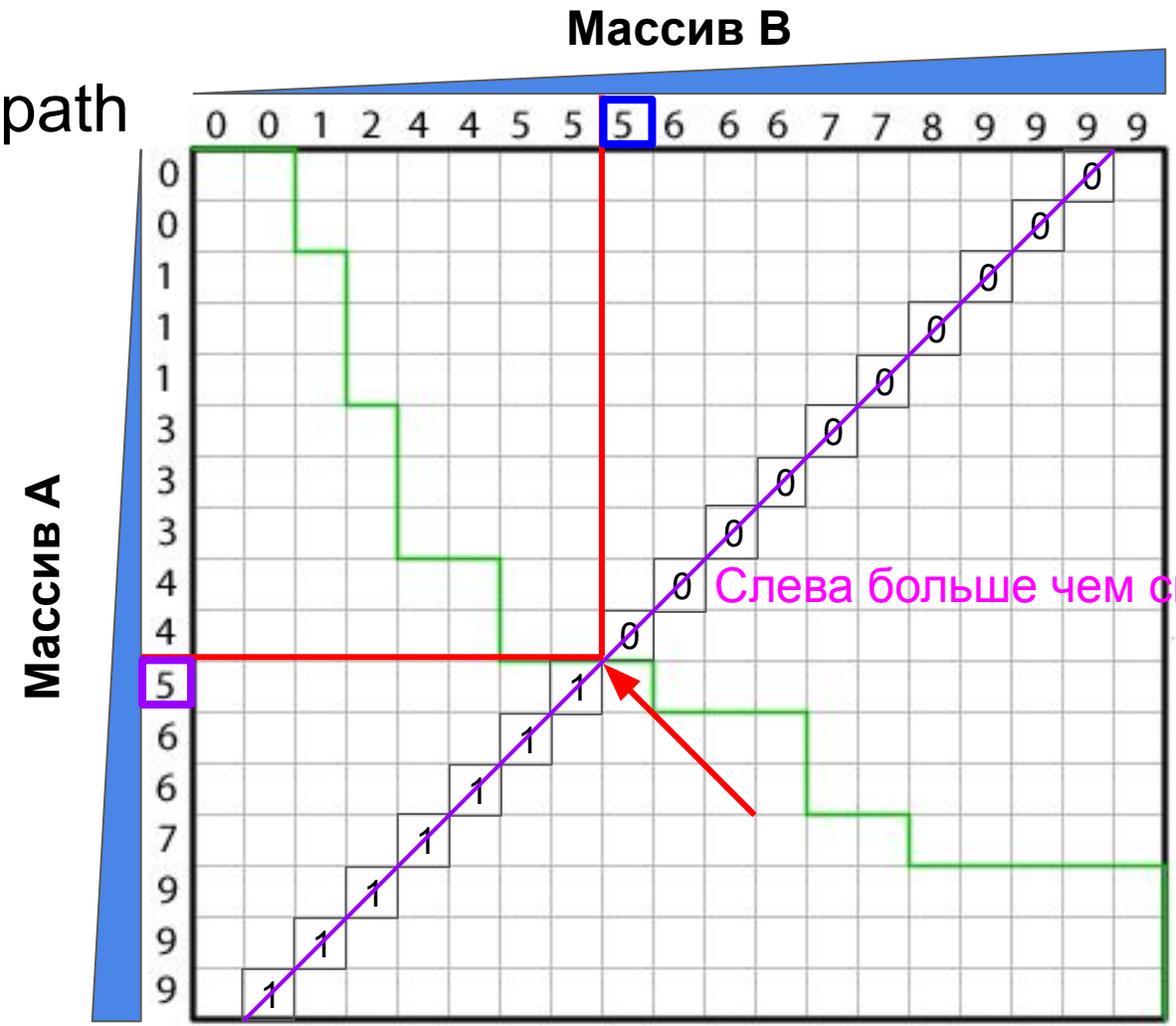
Как реализовать слияние двух отсортированныхм ассивов на GPU?

Merge path



Как реализовать слияние двух отсортированныхм ассивов на GPU?

Merge path

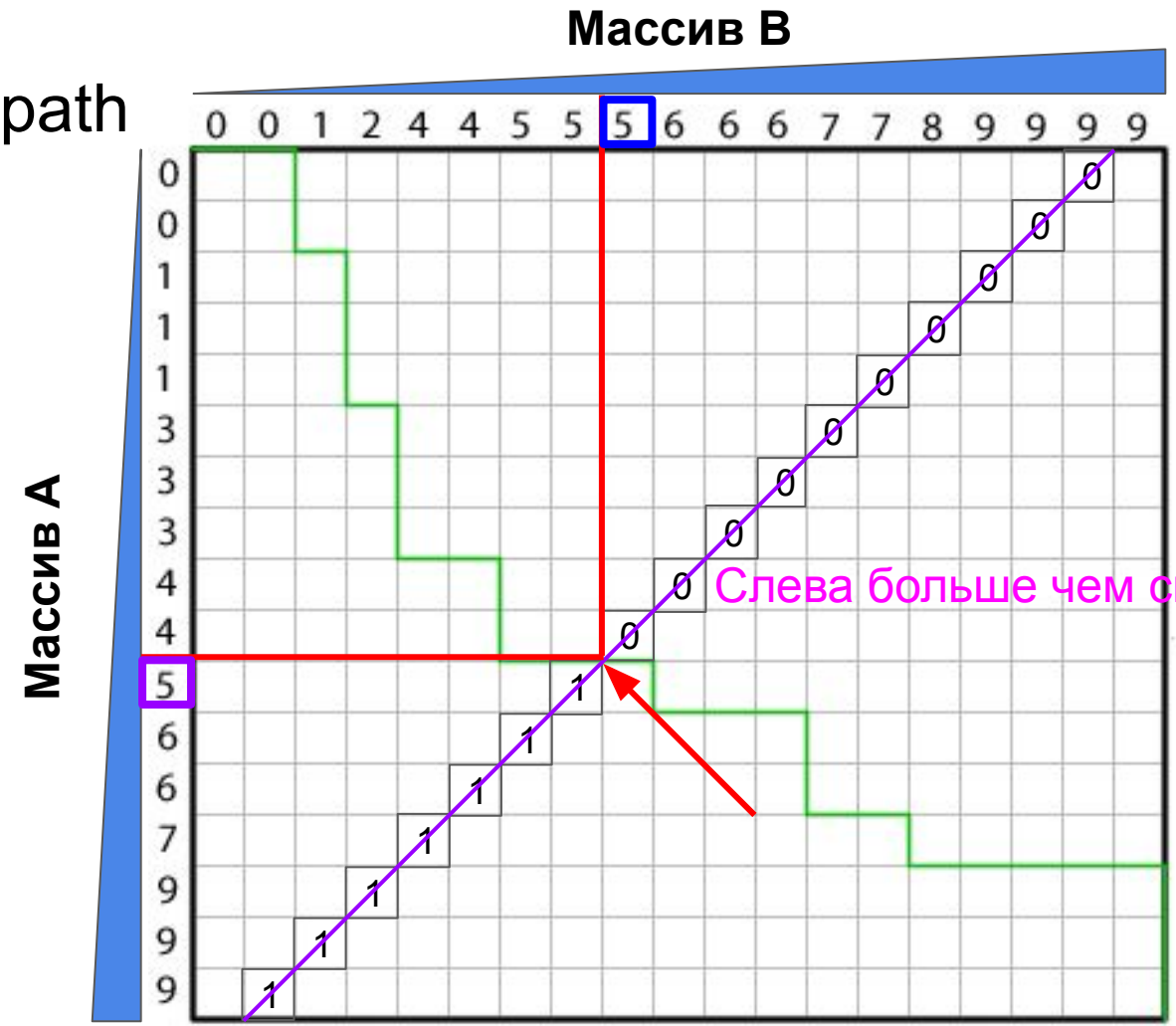


Как реализовать слияние двух отсортированныхм ассивов на GPU?

Слева больше чем сверху?

На какой индекс в **C** записать?

Merge path

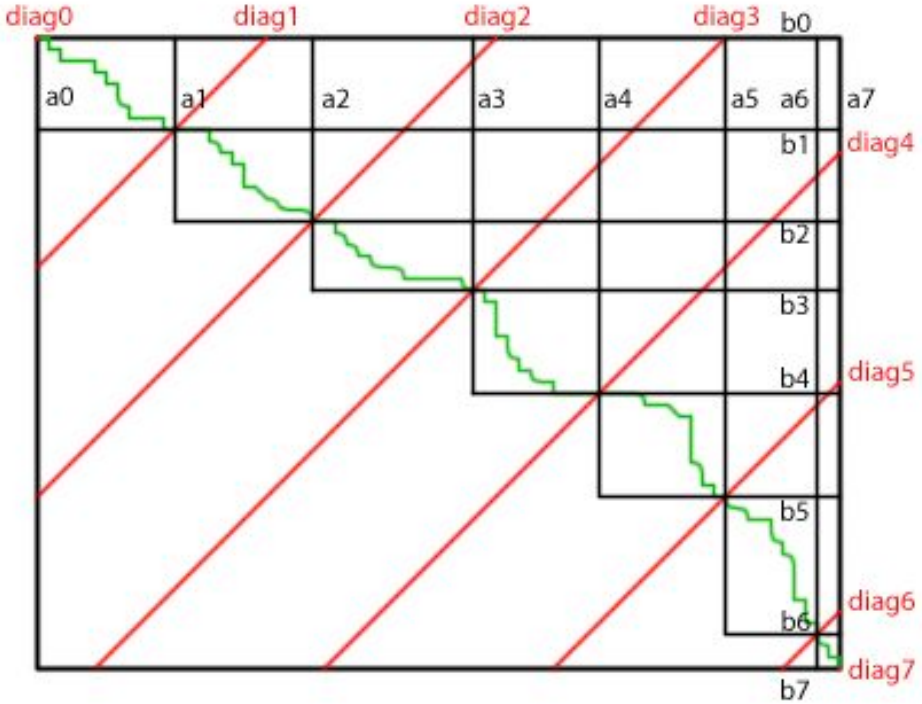


Как реализовать слияние двух отсортированныхм ассивов на GPU?

На какой индекс в **C** записать?

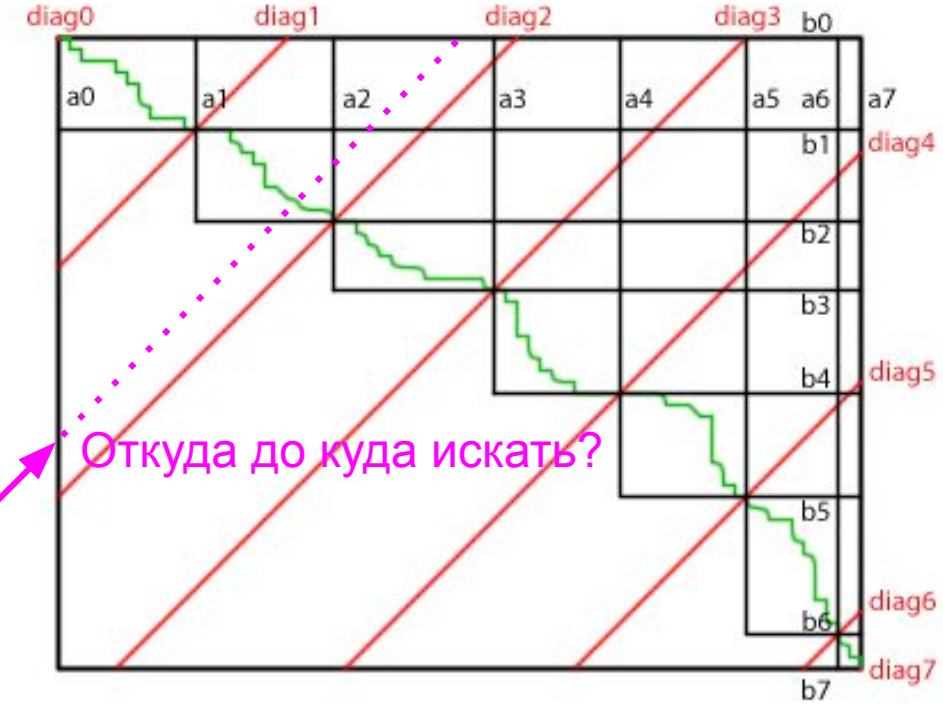
Как ускорить?

Merge path + двухуровневая иерархия



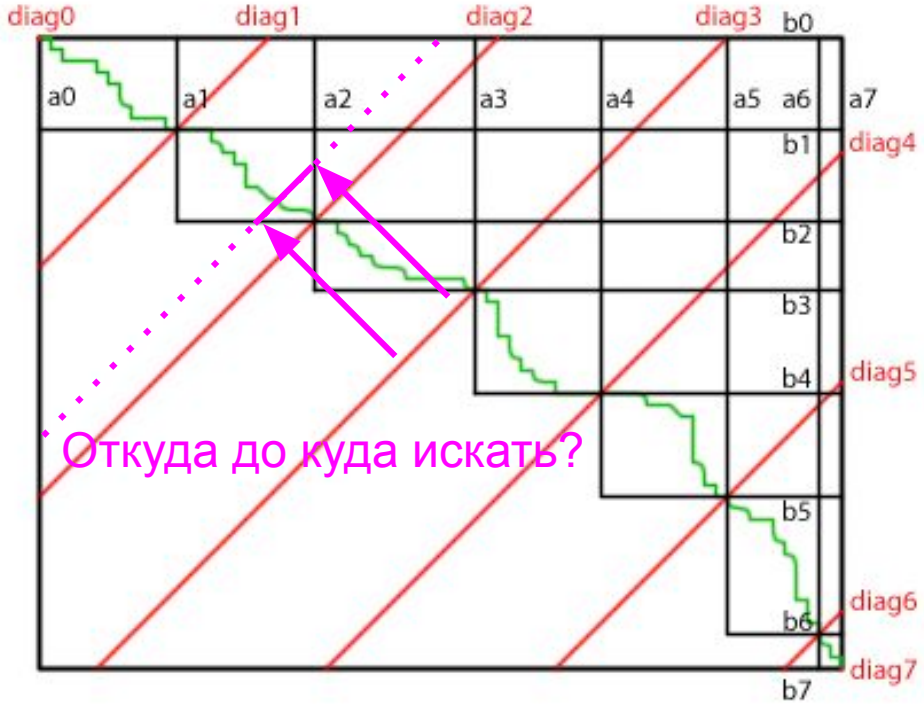
- https://moderngpu.github.io/bulkinsert.html
- https://moderngpu.github.io/merge.html
- Merge Path - Parallel Merging Made Simple, Odeh et al., 2012

Merge path + двухуровневая иерархия



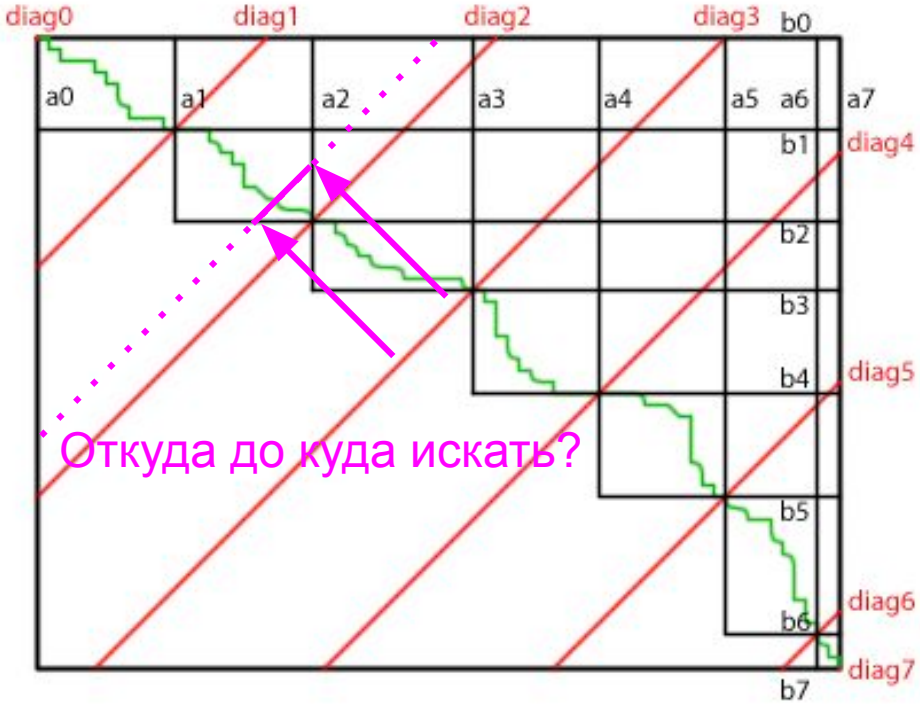
- https://moderngpu.github.io/bulkinsert.html
- https://moderngpu.github.io/merge.html
- Merge Path - Parallel Merging Made Simple, Odeh et al., 2012

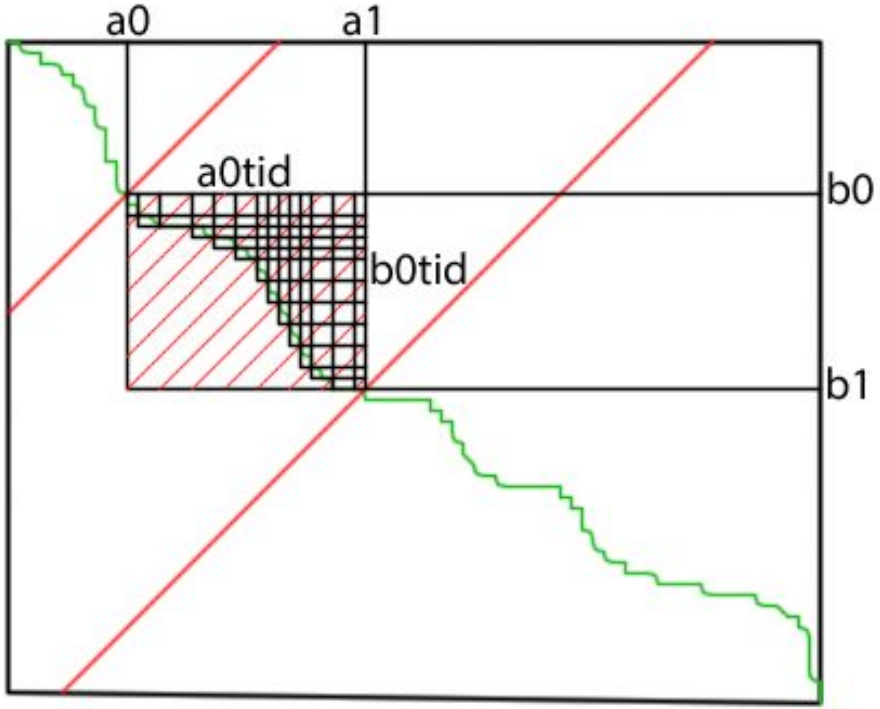
Merge path + двухуровневая иерархия



- https://moderngpu.github.io/bulkinsert.html
- https://moderngpu.github.io/merge.html
- Merge Path - Parallel Merging Made Simple, Odeh et al., 2012

Merge path + двухуровневая иерархия

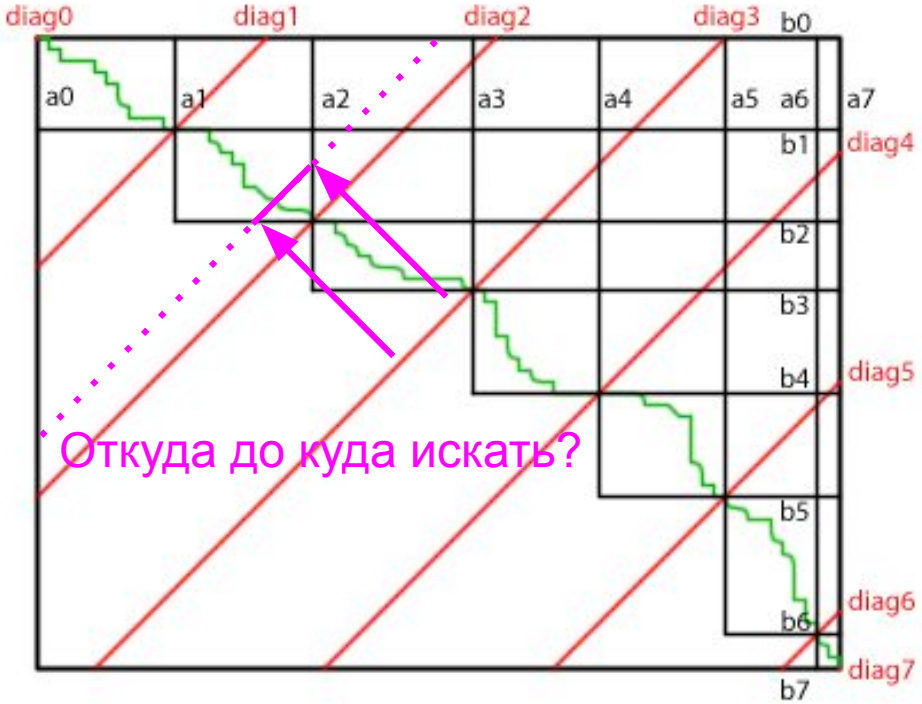


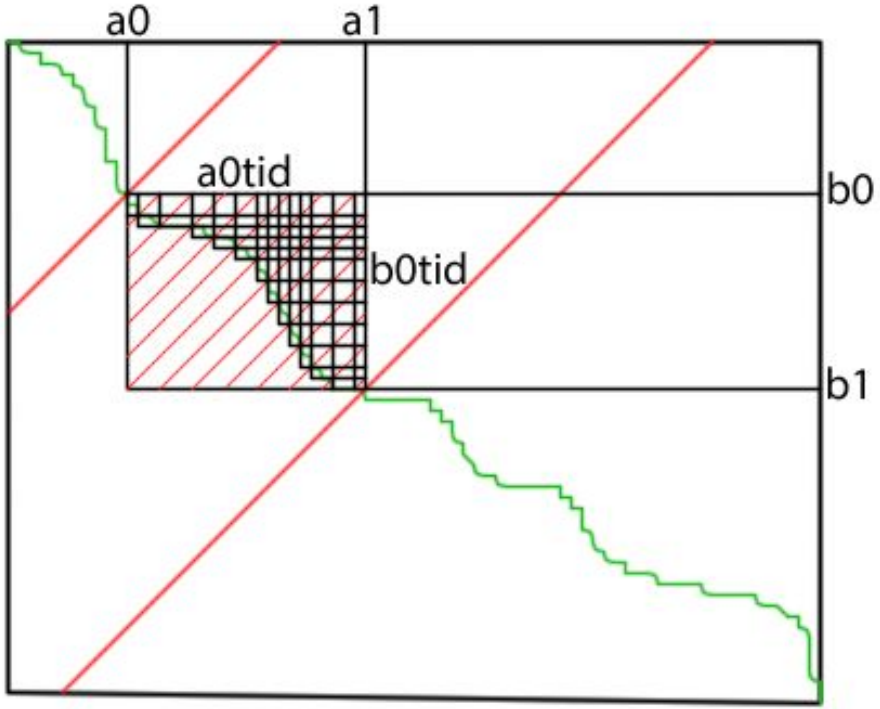


- https://moderngpu.github.io/bulkinsert.html
- https://moderngpu.github.io/merge.html
- Merge Path - Parallel Merging Made Simple, Odeh et al., 2012

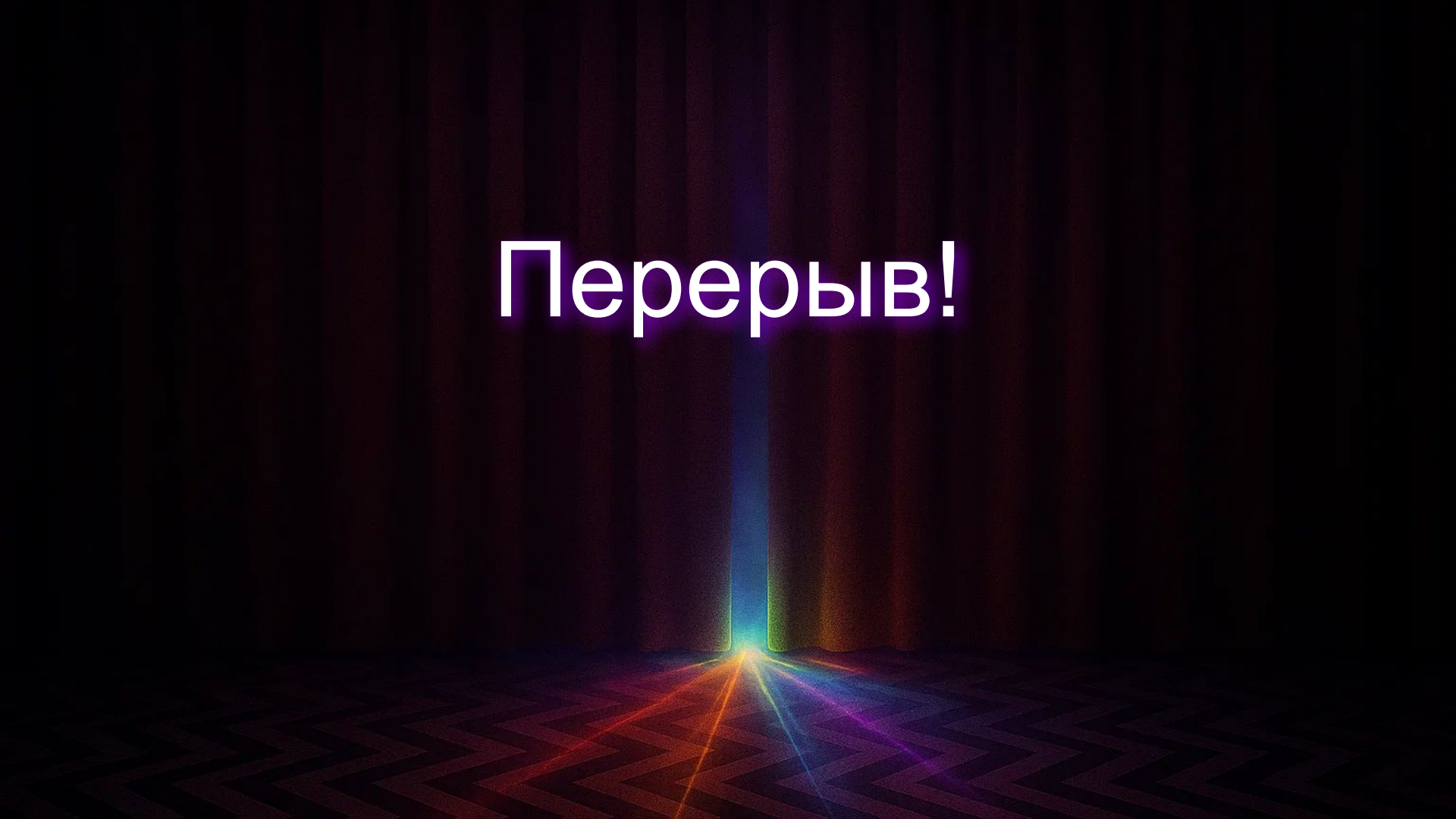
Ничего не напоминает?

Merge path + двухуровневая иерархия





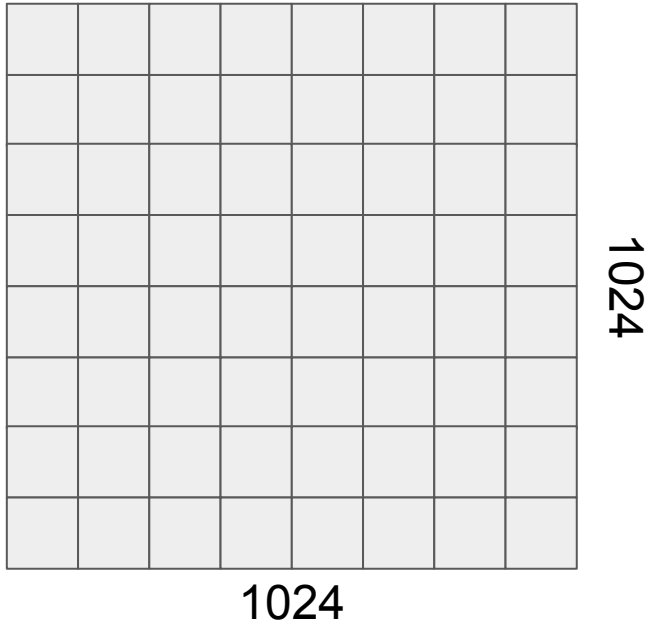
- https://moderngpu.github.io/bulkinsert.html
- https://moderngpu.github.io/merge.html
- Merge Path - Parallel Merging Made Simple, Odeh et al., 2012


Перерыв!

Глава 3:
Coarse to fine схема

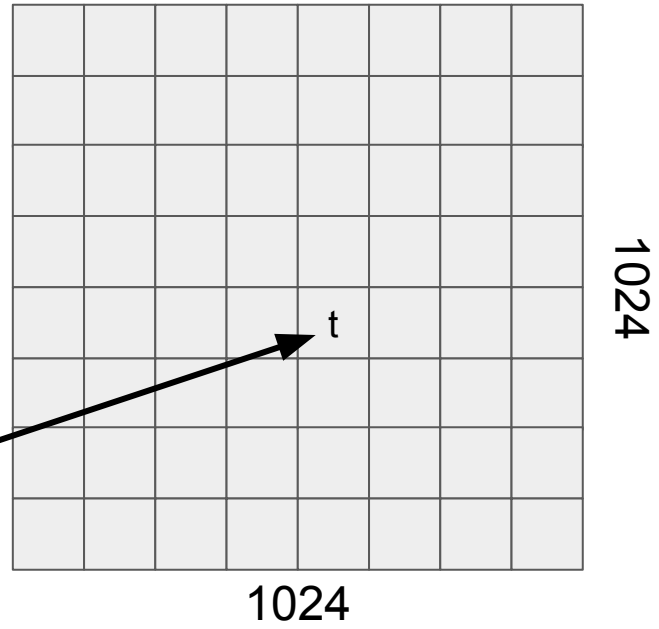
Coarse to fine схема: задачи на регулярной решетке

Есть дискретизация 2D пространства



Coarse to fine схема: задачи на регулярной решетке

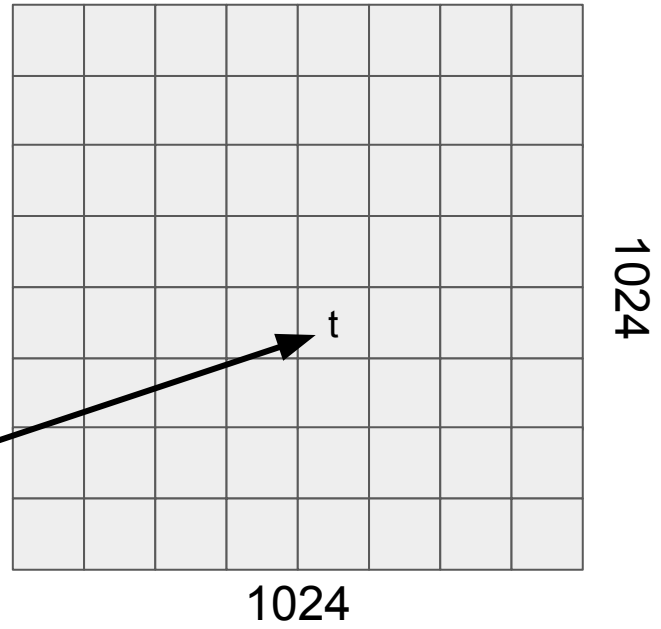
Есть дискретизация 2D пространства



в каждой точке/ячейке
хотим найти индикаторное значение какими-
то численными методами
(**много итераций**
с частичными производными)

Coarse to fine схема: задачи на регулярной решетке

Есть дискретизация 2D пространства

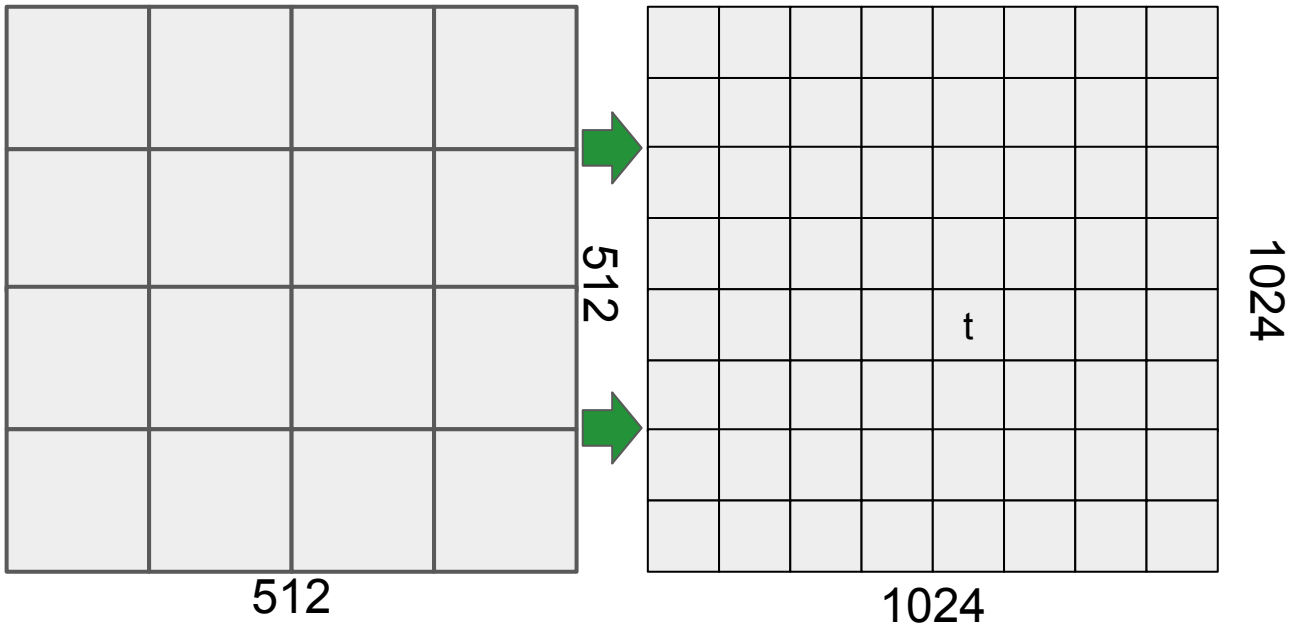


как ускорить рассчеты?
как потратить меньше GFlops но так же дойти до сходимости?

в каждой точке/ячейке хотим найти индикаторное значение какими-то численными методами (**много итераций** с частичными производными)

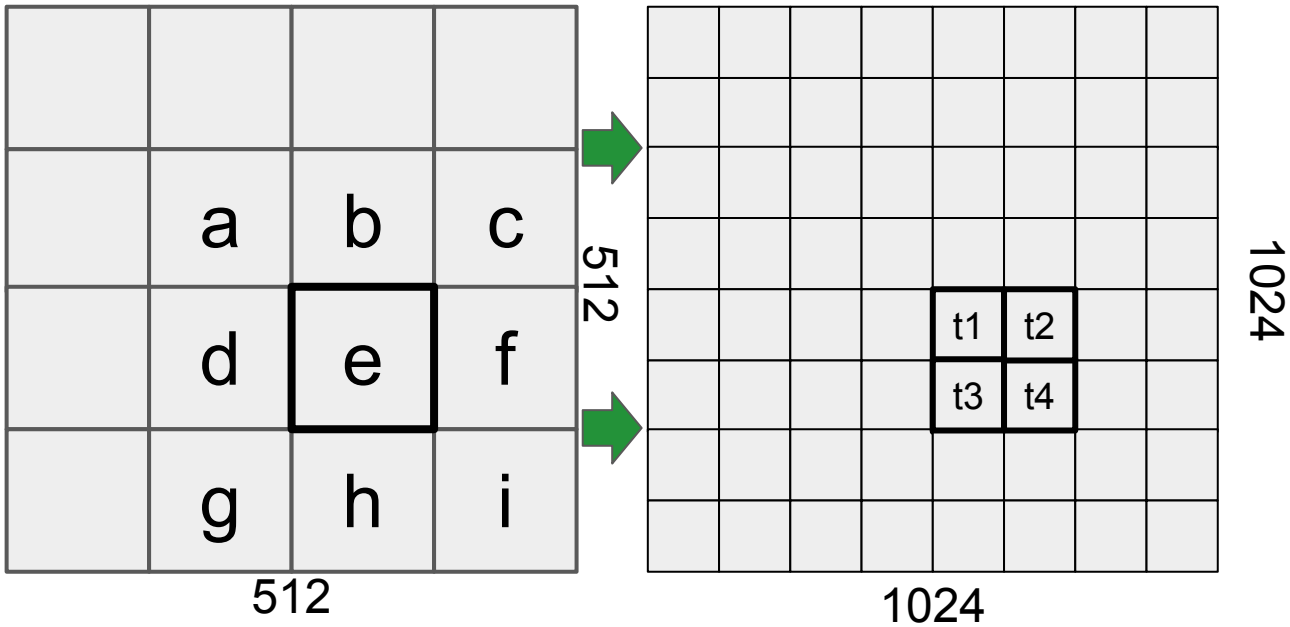
Coarse to fine схема: задачи на регулярной решетке

Есть дискретизация 2D пространства



Coarse to fine схема: задачи на регулярной решетке

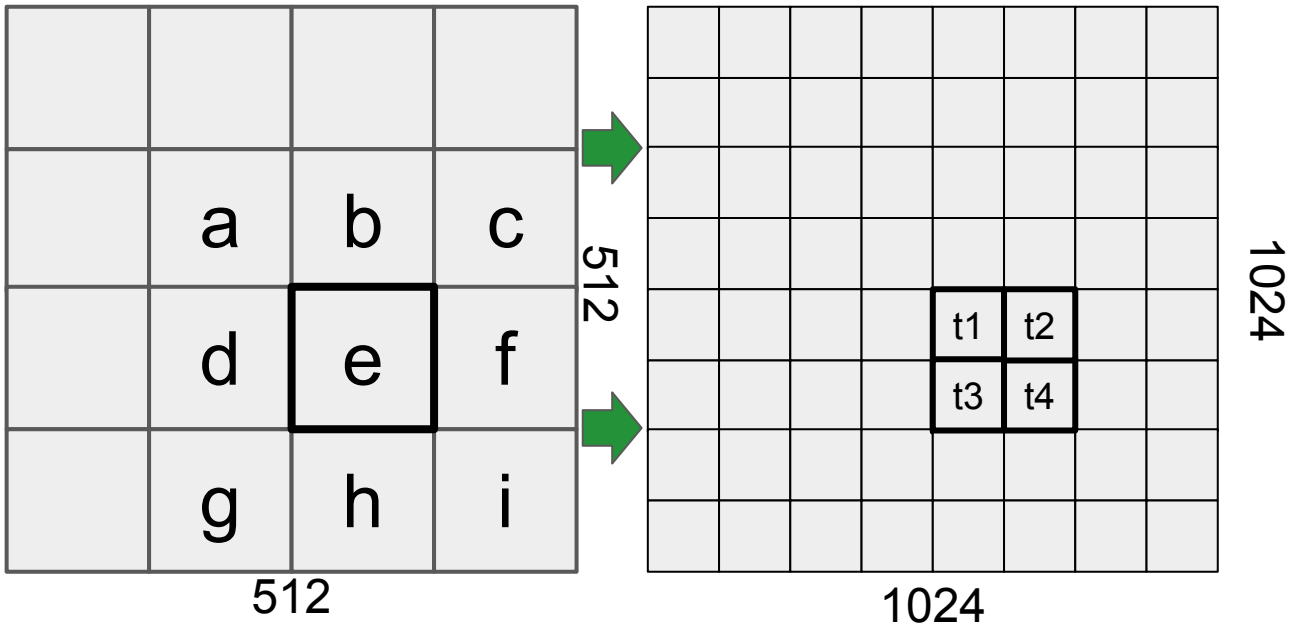
Есть дискретизация 2D пространства



С чего начинать итерацию после перехода детализации?

Coarse to fine схема: задачи на регулярной решетке

Есть дискретизация 2D пространства



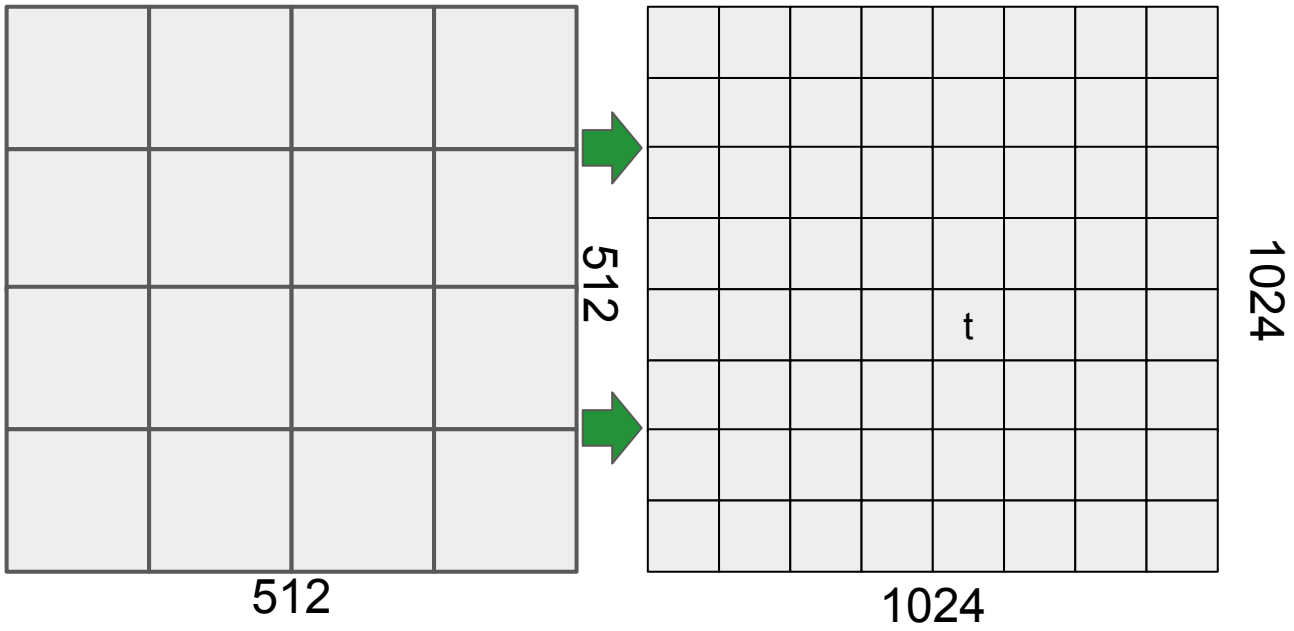
С чего начинать итерацию после перехода детализации?

Например билинейная интерполяция!

Coarse to fine схема: задачи на регулярной решетке

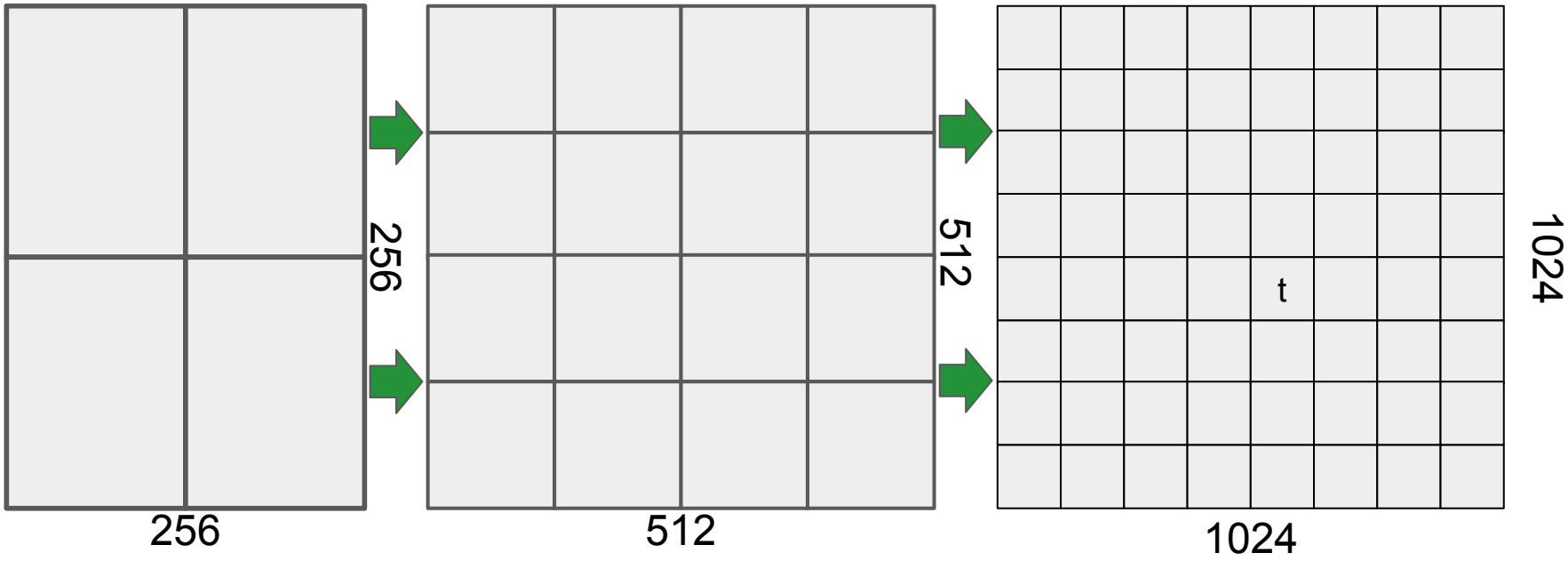
Есть дискретизация 2D пространства

как еще ускорить?



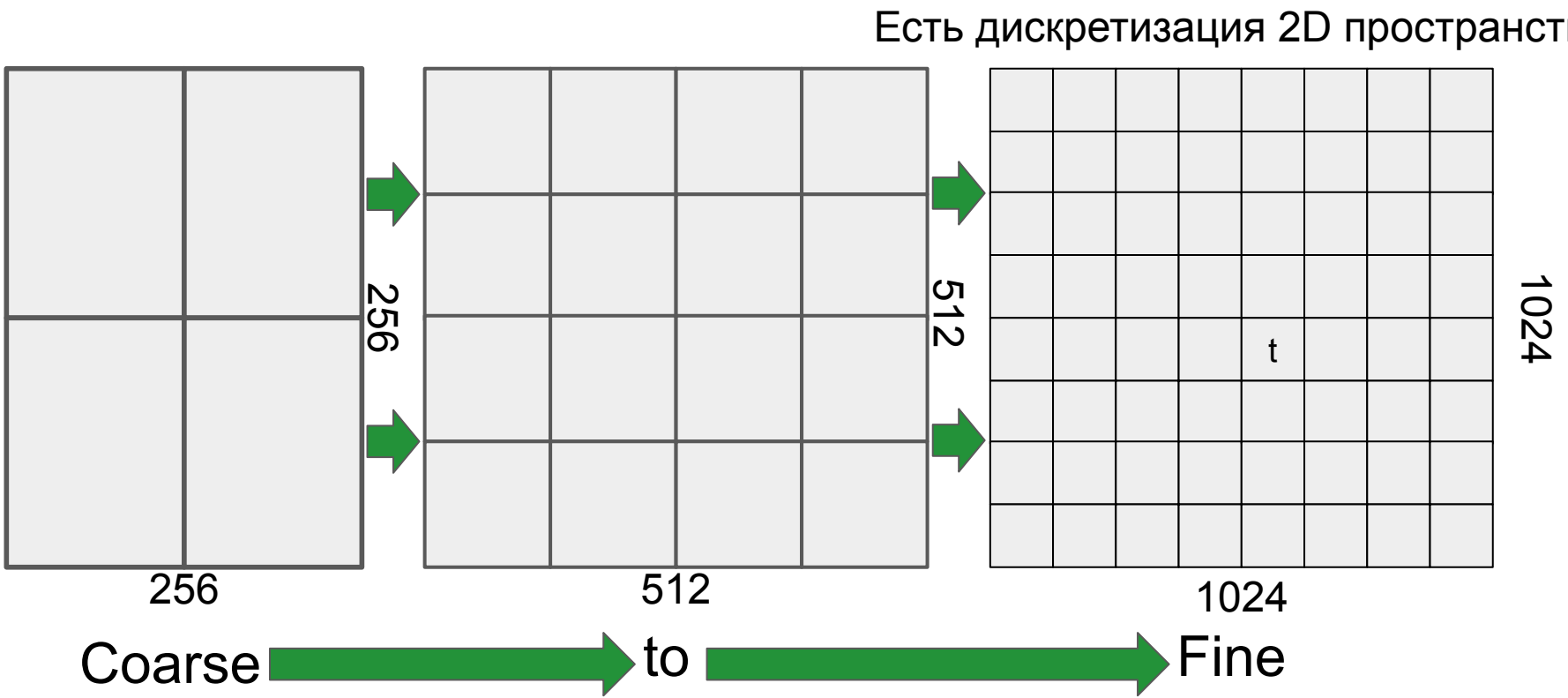
Coarse to fine схема: задачи на регулярной решетке

Есть дискретизация 2D пространства



Почему так названо?

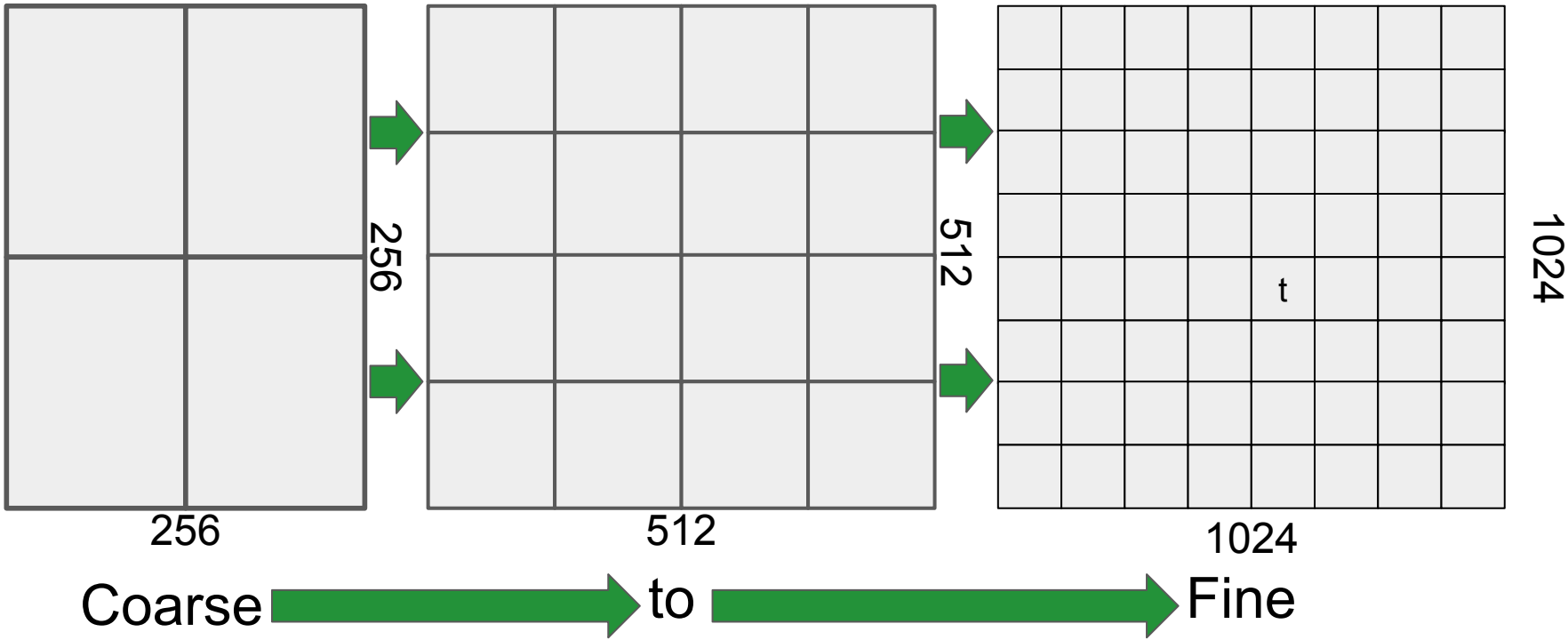
Coarse to fine схема: задачи на регулярной решетке



Coarse to fine схема: задачи на регулярной решетке

Обязательно 2D?

Есть дискретизация 2D пространства

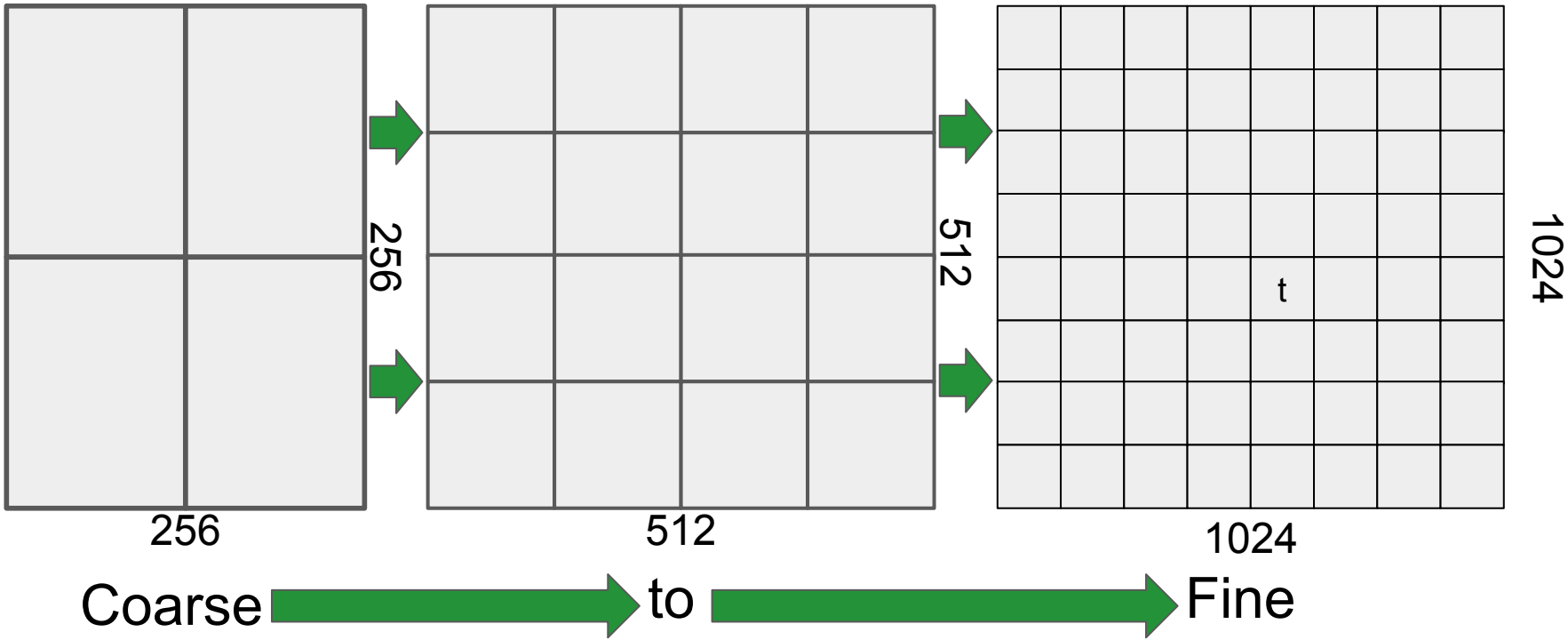


Обязательно регулярная решетка?

Coarse to fine схема: задачи на регулярной решетке

Обязательно 2D?

Есть дискретизация 2D пространства

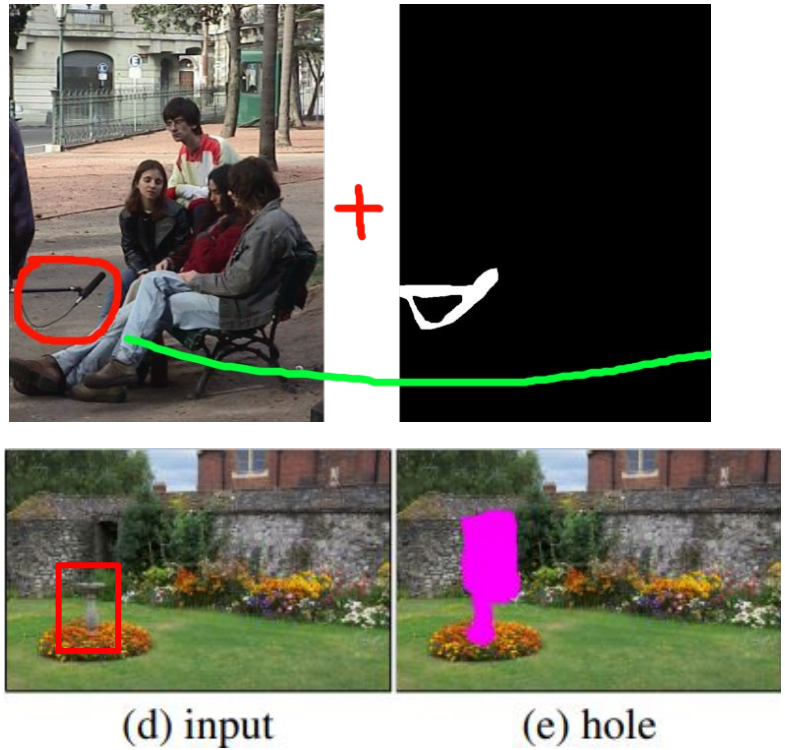


Глава 4:
Карты глубины - Patch Match

PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing *(Barnes et. al., 2009)*

(d) input

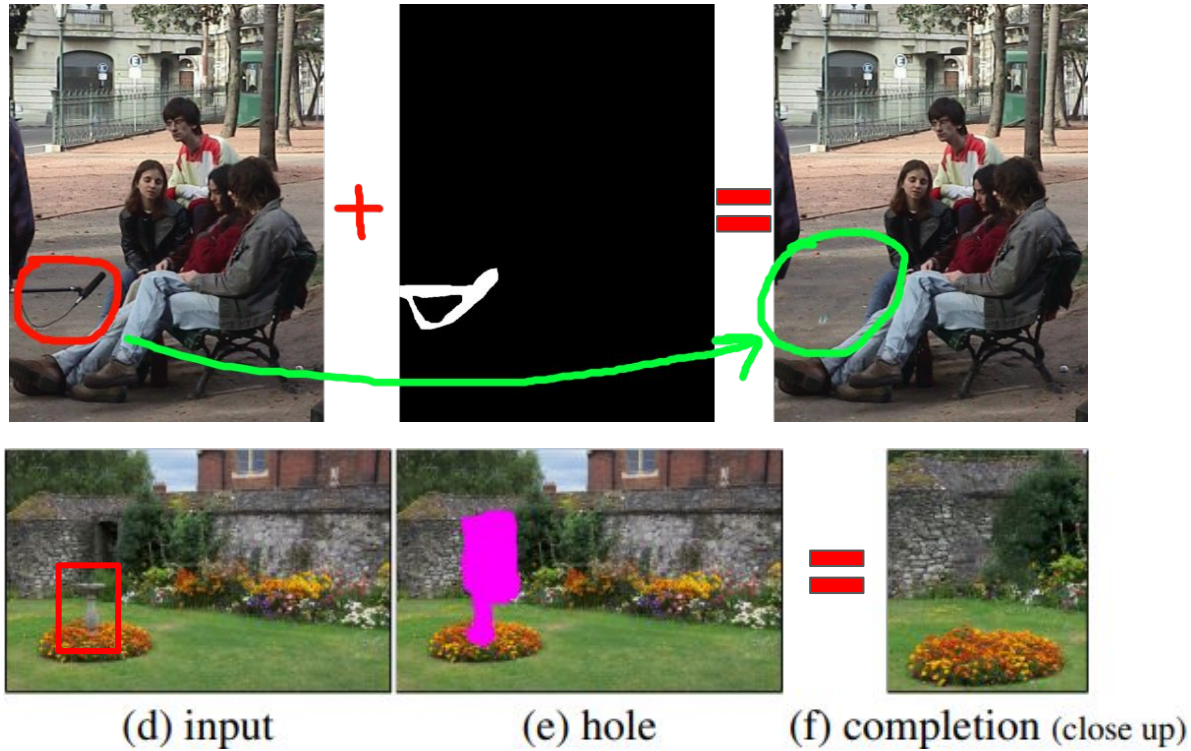
PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing *(Barnes et. al., 2009)*



(d) input (e) hole

PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing, Barnes et. al., 2009

PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing *(Barnes et. al., 2009)*



(d) input (e) hole (f) completion (close up)

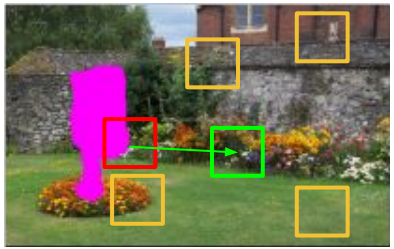
PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing, Barnes et. al., 2009

PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing *(Barnes et. al., 2009)*

Как выбрать какой цвет у пикселя на краю удаленной зоны?

PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing, Barnes et. al., 2009

PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing *(Barnes et. al., 2009)*

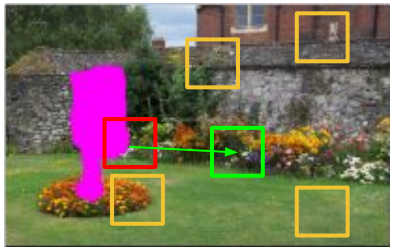


Найдем в другой части картинки патч, который больше всего похож на нас.

Как ускорить чтобы не перебирать все варианты патчей?

PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing, Barnes et. al., 2009

PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing *(Barnes et. al., 2009)*



Найдем в другой части картинки патч, который больше всего похож на нас.

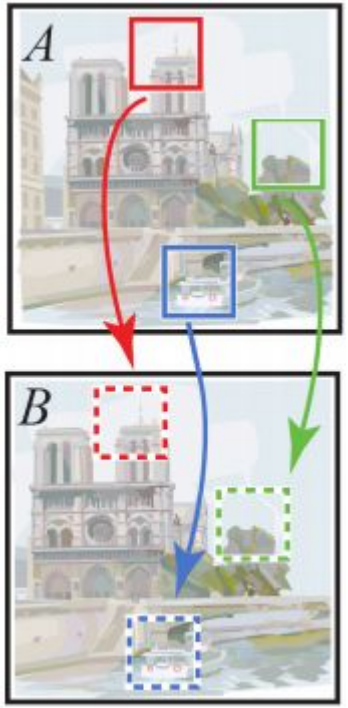
Как ускорить чтобы не перебирать все варианты патчей?

(кроме конечно же **Coarse-to-Fine** схемы)

PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing, Barnes et. al., 2009

PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing *(Barnes et. al., 2009)*

Хотим для каждого патча в A найти самый похожий патч в B.

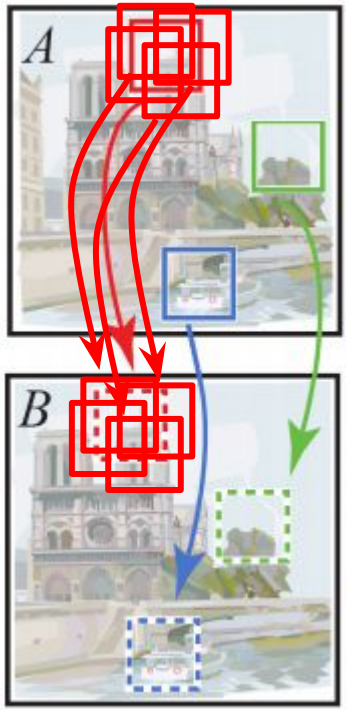


PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing, Barnes et. al., 2009

PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing *(Barnes et. al., 2009)*

Хотим для каждого патча в A найти самый похожий патч в B.

Natural structure of images: правильный ответ обычно содержит большие связные регионы сопоставления. Можем увеличить эффективность выполняя поиск зависимо от соседей. Т.е. вдоховляясь их успехами.



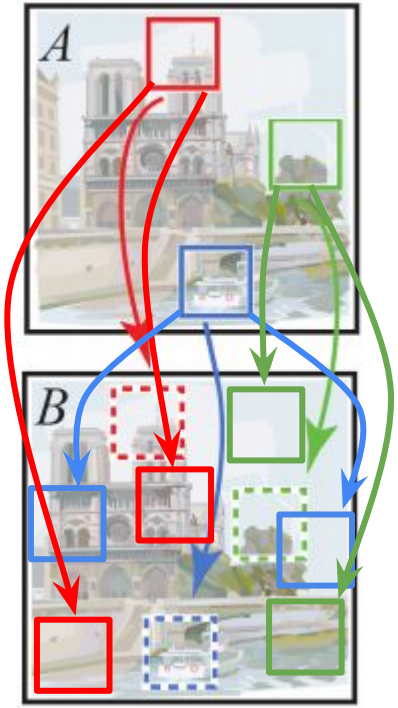
PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing, Barnes et. al., 2009

PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing *(Barnes et. al., 2009)*

Хотим для каждого патча в A найти самый похожий патч в B.

Natural structure of images: правильный ответ обычно содержит большие связные регионы сопоставления. Можем увеличить эффективность выполняя поиск зависимо от соседей. Т.е. вдоховляясь их успехами.

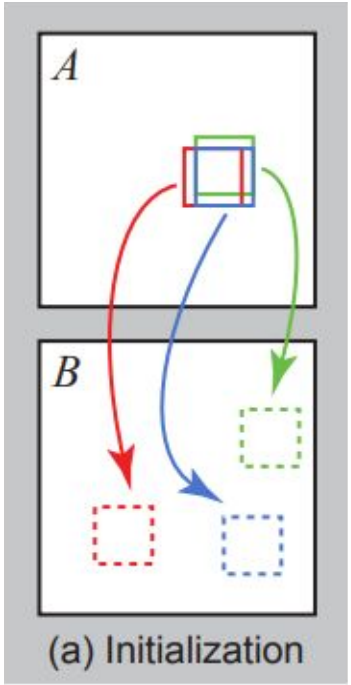
The law of large numbers: одна случайная гипотеза не угадает никогда. Но если каждый пиксель посмотрит в случайное место - кто-то да угадает! Дополнительные итерации увеличивают шансы еще больше.



PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing, Barnes et. al., 2009

PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing *(Barnes et. al., 2009)*

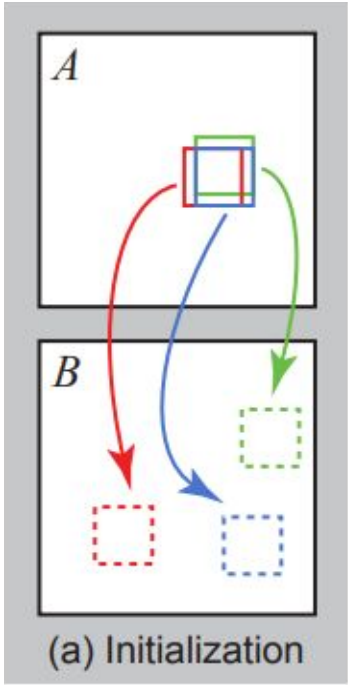
1) В каждом пикселе случайная гипотеза (dx, dy)



PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing, Barnes et. al., 2009

PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing *(Barnes et. al., 2009)*

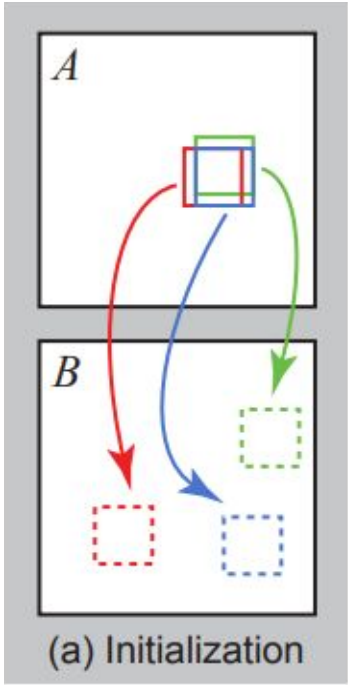
1) В каждом пикселе случайная гипотеза (dx, dy)
2) Несколько итераций делаем:



PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing, Barnes et. al., 2009

PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing *(Barnes et. al., 2009)*

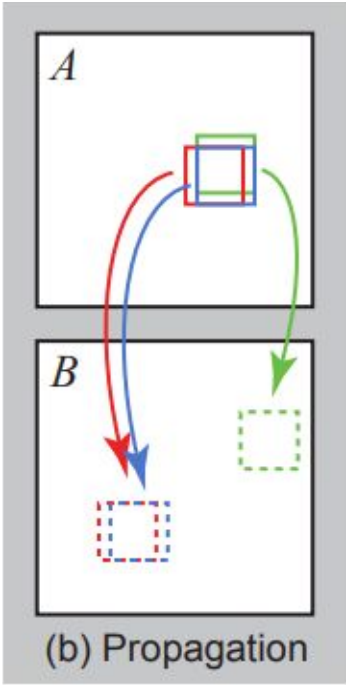
1) В каждом пикселе случайная гипотеза (dx, dy)
2) Несколько итераций делаем:
   2.1) **Propagation:** примеряем на себя гипотезы соседей.
   Т.е. используем их как источник хорошей идеи.



PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing, Barnes et. al., 2009

PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing *(Barnes et. al., 2009)*

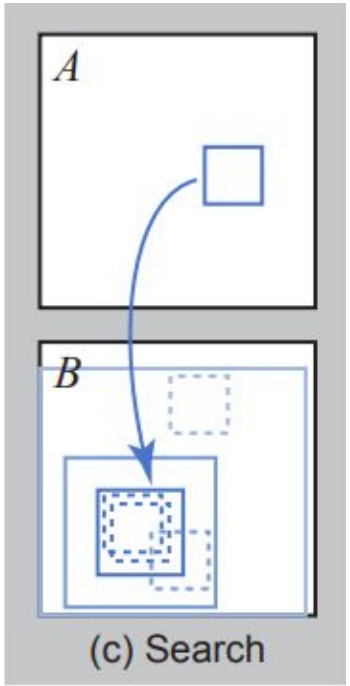
1) В каждом пикселе случайная гипотеза (dx, dy)
2) Несколько итераций делаем:
   2.1) **Propagation:** примеряем на себя гипотезы соседей.
   Т.е. используем их как источник хорошей идеи.



PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing, Barnes et. al., 2009

PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing *(Barnes et. al., 2009)*

1) В каждом пикселе случайная гипотеза (dx, dy)
2) Несколько итераций делаем:
   2.1) **Propagation:** примеряем на себя гипотезы соседей. Т.е. используем их как источник хорошей идеи.
   2.2) **Refinement:** пробуем улучшить/уточнить свою гипотезу случайными сдвигами.

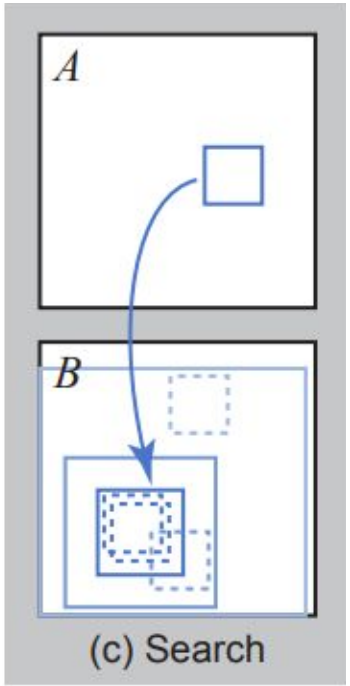


PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing, Barnes et. al., 2009

PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing *(Barnes et. al., 2009)*

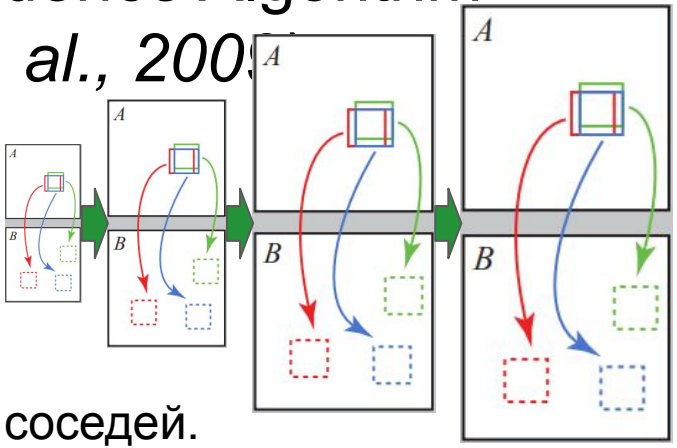
Как ускорить сходимость?

1) В каждом пикселе случайная гипотеза (dx, dy)
2) Несколько итераций делаем:
   2.1) **Propagation:** примеряем на себя гипотезы соседей. Т.е. используем их как источник хорошей идеи.
   2.2) **Refinement:** пробуем улучшить/уточнить свою гипотезу случайными сдвигами.



PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing, Barnes et. al., 2009

PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing *(Barnes et. al., 200*



for **level** = 0 … N (**Coarse-to-Fine**)

1) В каждом пикселе **upscale** / случайное (dx, dy)
2) Несколько итераций делаем:
   2.1) **Propagation:** примеряем на себя гипотезы соседей. Т.е. используем их как источник хорошей идеи.
   2.2) **Refinement:** пробуем улучшить/уточнить свою гипотезу случайными сдвигами.

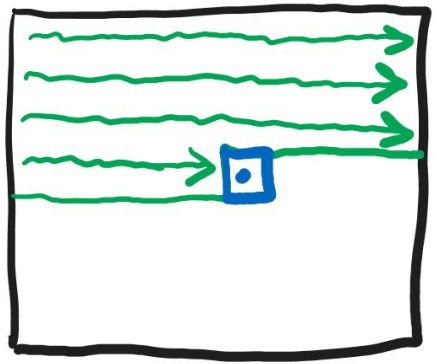
PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing, Barnes et. al., 2009

PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing *(Barnes et. al., 2009)*

for **level** = 0 … N (**Coarse-to-Fine**)

1) В каждом пикселе **upscale** / случайное (dx, dy)
2) Несколько итераций делаем:
   2.1) **Propagation:** примеряем на себя гипотезы соседей.
   Т.е. используем их как источник хорошей идеи.
   При прямом проходе - **???**

   2.2) **Refinement:** пробуем улучшить/уточнить свою гипотезу случайными сдвигами.



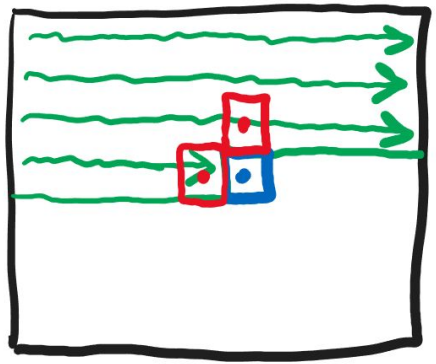
PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing, Barnes et. al., 2009

PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing *(Barnes et. al., 2009)*

for **level** = 0 … N (**Coarse-to-Fine**)

1) В каждом пикселе **upscale** / случайное (dx, dy)
2) Несколько итераций делаем:
   2.1) **Propagation:** примеряем на себя гипотезы соседей.
   Т.е. используем их как источник хорошей идеи.
   При прямом проходе - соседи слева/сверху.

   2.2) **Refinement:** пробуем улучшить/уточнить свою гипотезу случайными сдвигами.

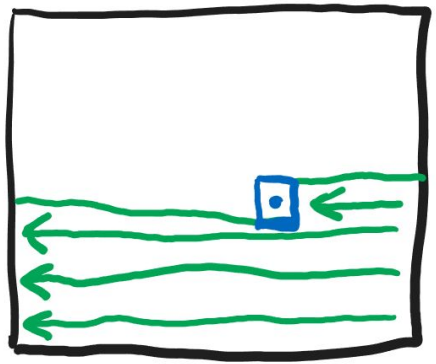


PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing, Barnes et. al., 2009

PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing *(Barnes et. al., 2009)*

for **level** = 0 … N (**Coarse-to-Fine**)

1) В каждом пикселе **upscale** / случайное (dx, dy)
2) Несколько итераций делаем:
   2.1) **Propagation:** примеряем на себя гипотезы соседей.
   Т.е. используем их как источник хорошей идеи.
   При прямом проходе - соседи слева/сверху.
   При обратном проходе - **???**
   2.2) **Refinement:** пробуем улучшить/уточнить свою гипотезу случайными сдвигами.

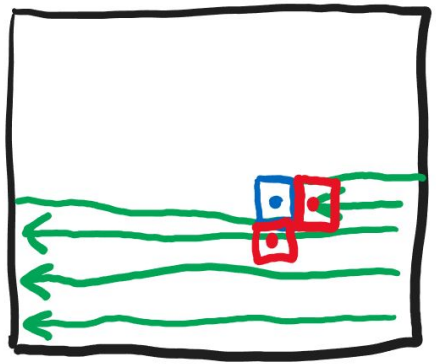


PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing, Barnes et. al., 2009

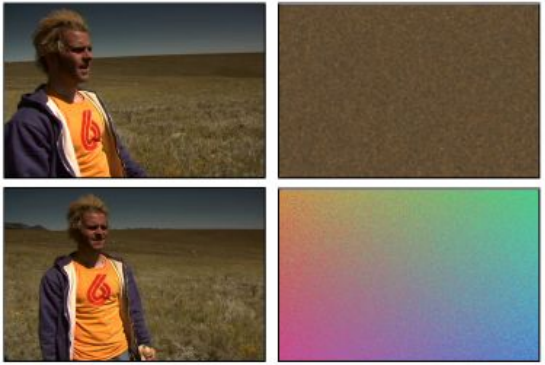
PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing *(Barnes et. al., 2009)*

for **level** = 0 … N (**Coarse-to-Fine**)

1) В каждом пикселе **upscale** / случайное (dx, dy)
2) Несколько итераций делаем:
   2.1) **Propagation:** примеряем на себя гипотезы соседей.
   Т.е. используем их как источник хорошей идеи.
   При прямом проходе - соседи слева/сверху.
   При обратном проходе - соседи справа/снизу.
   2.2) **Refinement:** пробуем улучшить/уточнить свою гипотезу случайными сдвигами.

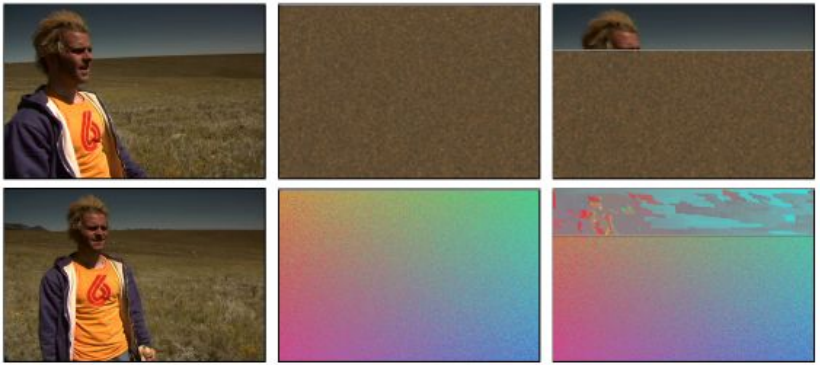


PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing, Barnes et. al., 2009



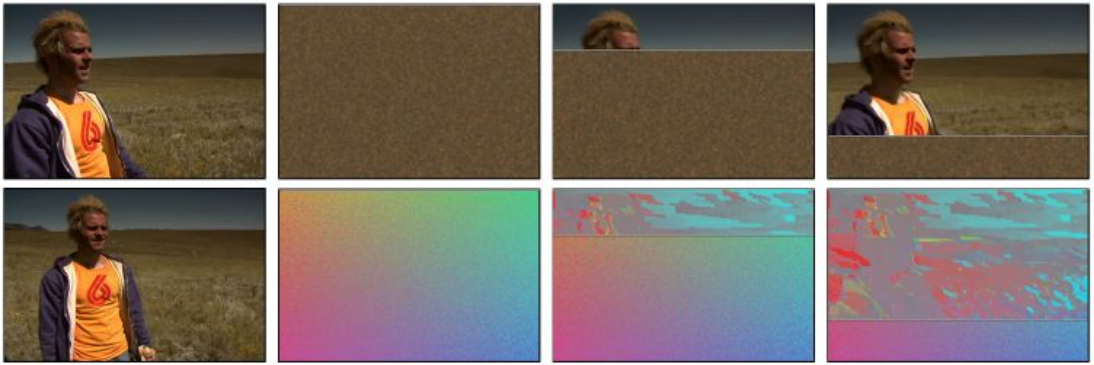
(a) originals (b) random

PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing, Barnes et. al., 2009



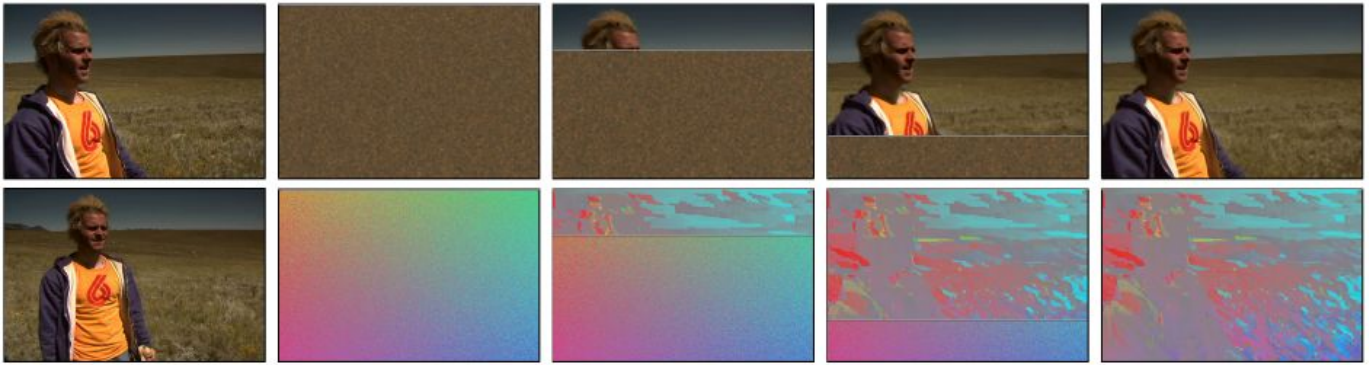
(a) originals (b) random (c) $\frac{1}{4}$ iteration

PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing, Barnes et. al., 2009



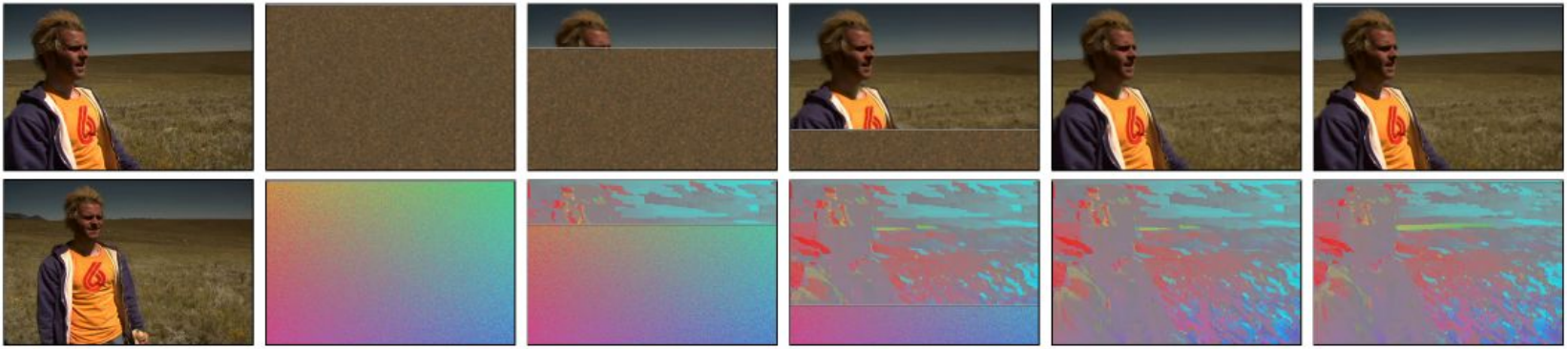
(a) originals (b) random (c) $\frac{1}{4}$ iteration (d) $\frac{3}{4}$ iteration

PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing, Barnes et. al., 2009



(a) originals (b) random (c) $\frac{1}{4}$ iteration (d) $\frac{3}{4}$ iteration (e) 1 iteration

PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing, Barnes et. al., 2009



(a) originals (b) random (c) $\frac{1}{4}$ iteration (d) $\frac{3}{4}$ iteration (e) 1 iteration (f) 2 iterations

PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing, Barnes et. al., 2009

(a) originals (b) random (c) $\frac{1}{4}$ iteration (d) $\frac{3}{4}$ iteration (e) 1 iteration (f) 2 iterations (g) 5 iterations

Figure 3: *Illustration of convergence. (a) The top image is reconstructed using only patches from the bottom image. (b) above: the reconstruction by the patch "voting" described in Section 4, below: a random initial offset field, with magnitude visualized as saturation and angle visualized as hue. (c) 1/4 of the way through the first iteration, high-quality offsets have been propagated in the region above the current scan line (denoted with the horizontal bar). (d) 3/4 of the way through the first iteration. (e) First iteration complete. (f) Two iterations. (g) After 5 iterations, almost all patches have stopped changing. The tiny orange flowers only find good correspondences in the later iterations.*

(d) input (e) hole (f) completion (close up)

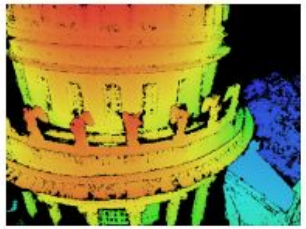
PatchMatch: A Randomized Correspondence Algorithm for Structural Image Editing, Barnes et. al., 2009

Построение карты глубины методом Patch Match

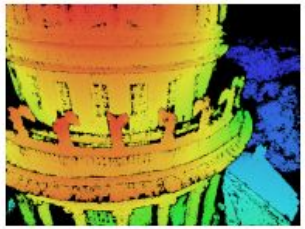
is_copter-20180821-0-4041

is_copter-20180821-0-4042

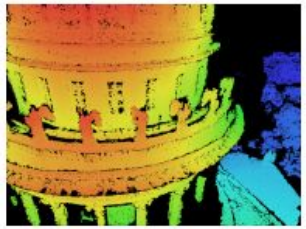
is_copter-20180821-0-4043



is_copter-20180821-0-4044



is_copter-20180821-0-4045

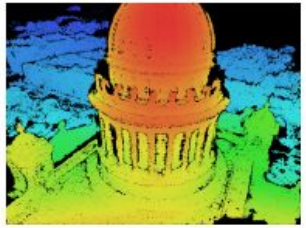


is_copter-20180821-0-4046

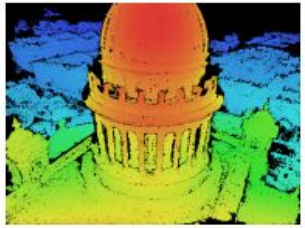
is_copter-20180823-0-0811

is_copter-20180823-0-0812

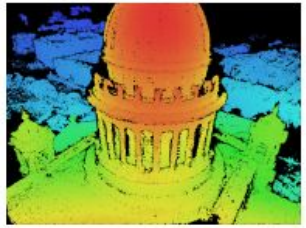
is_copter-20180823-0-0813



is_copter-20180823-0-0814



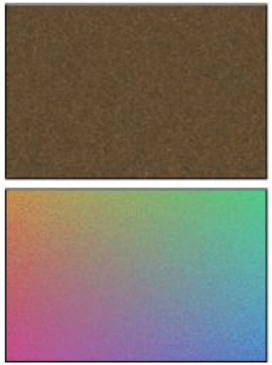
is_copter-20180823-0-0815



is_copter-20180823-0-0816

Построение карты глубины методом Patch Match

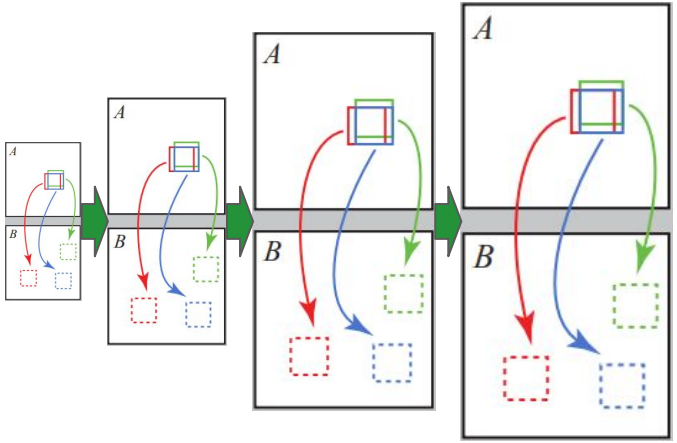
depth_map, normal_map = random()



Построение карты глубины методом Patch Match

depth_map, normal_map = random()

for level = 0 … N: (Coarse to Fine)

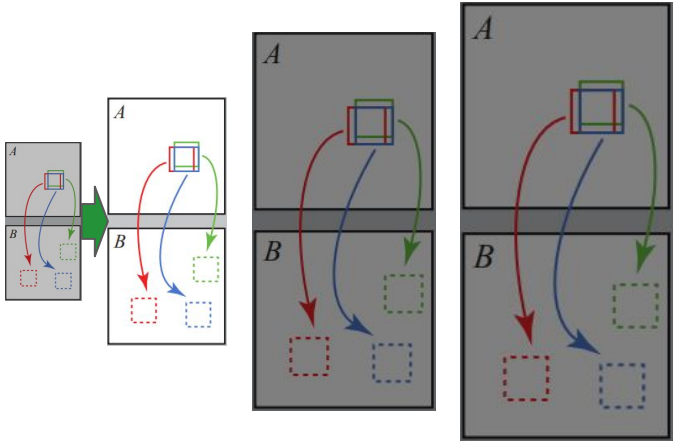


Построение карты глубины методом Patch Match

depth_map, normal_map = random()

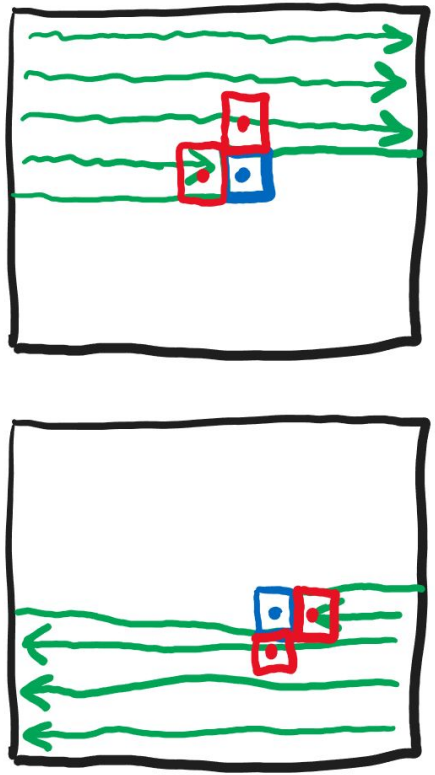
for level = 0 … N:

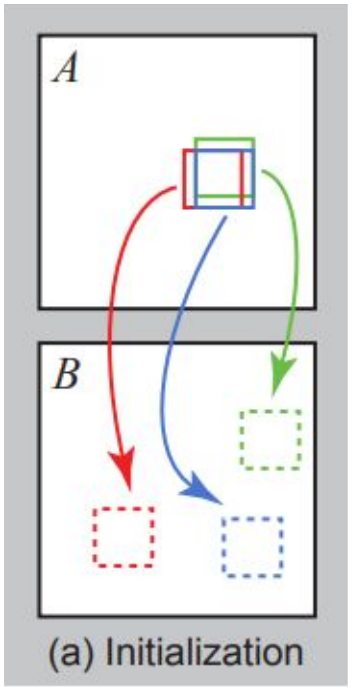
    upscale(depth_map, normal_map)



Построение карты глубины методом Patch Match

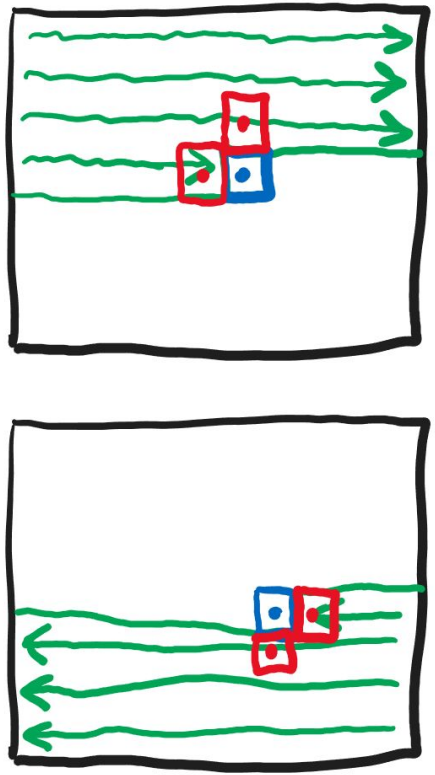
```
depth_map, normal_map = random()

for level = 0 … N:
    upscale(depth_map, normal_map)
    for iteration = 0 … 100:
        propagation()
```

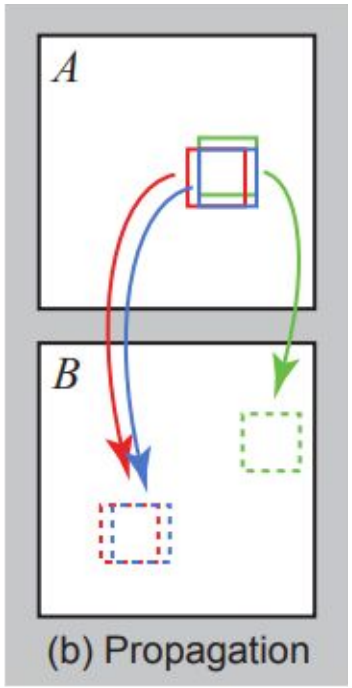




Построение карты глубины методом Patch Match

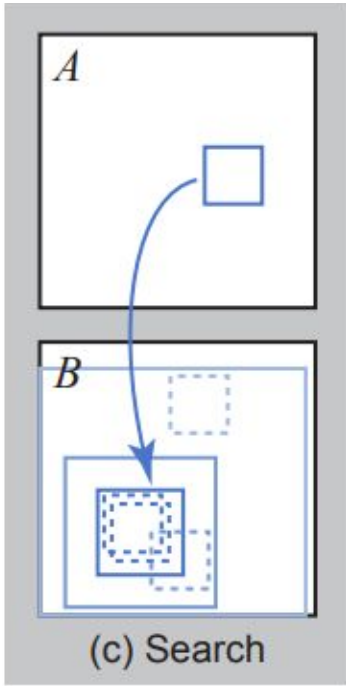
```
depth_map, normal_map = random()

for level = 0 … N:
    upscale(depth_map, normal_map)
    for iteration = 0 … 100:
        propagation()
```





Построение карты глубины методом Patch Match

```
depth_map, normal_map = random()

for level = 0 … N:
    upscale(depth_map, normal_map)
    for iteration = 0 … 100:
        propagation()
        refinement()
```

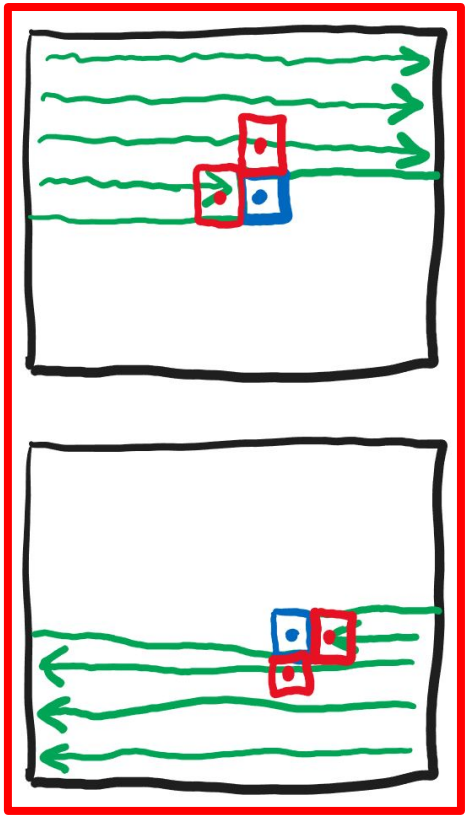


Построение карты глубины методом Patch Match

```
depth_map, normal_map = random()

for level = 0 … N:
    upscale(depth_map, normal_map)
    for iteration = 0 … 100:
        propagation()
        refinement()

return depth_map, normal_map
```

Построение карты глубины методом Patch Match

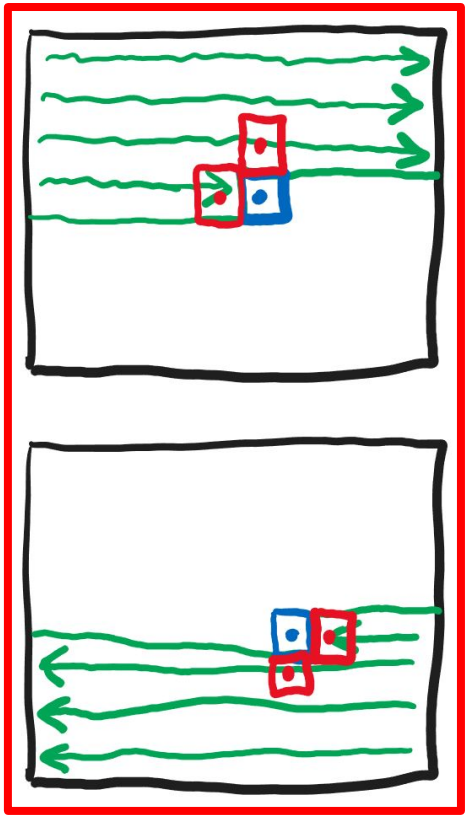
```
depth_map, normal_map = random()

for level = 0 … N:
    upscale(depth_map, normal_map)
    for iteration = 0 … 100:
        propagation()
        refinement()

return depth_map, normal_map
```



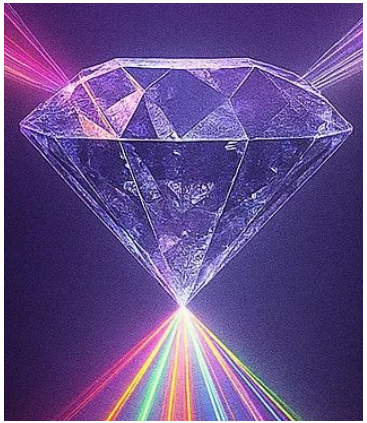
Массовый параллелизм?

Построение карты глубины методом Patch Match

```
depth_map, normal_map = random()

for level = 0 … N:
    upscale(depth_map, normal_map)
    for iteration = 0 … 100:
        propagation()
        refinement()

return depth_map, normal_map
```

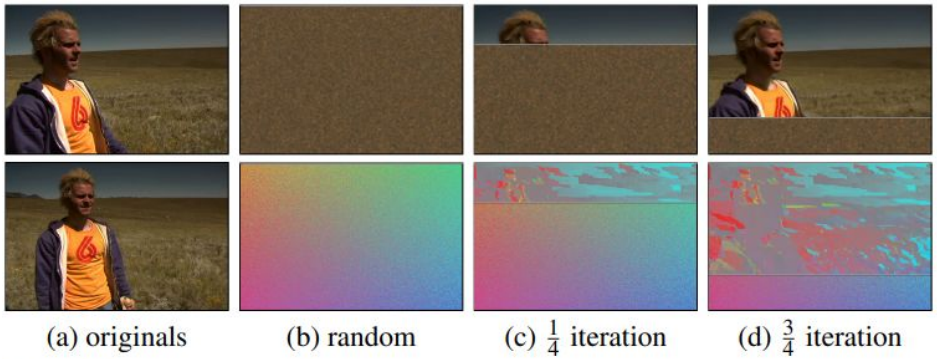


Массовый параллелизм?

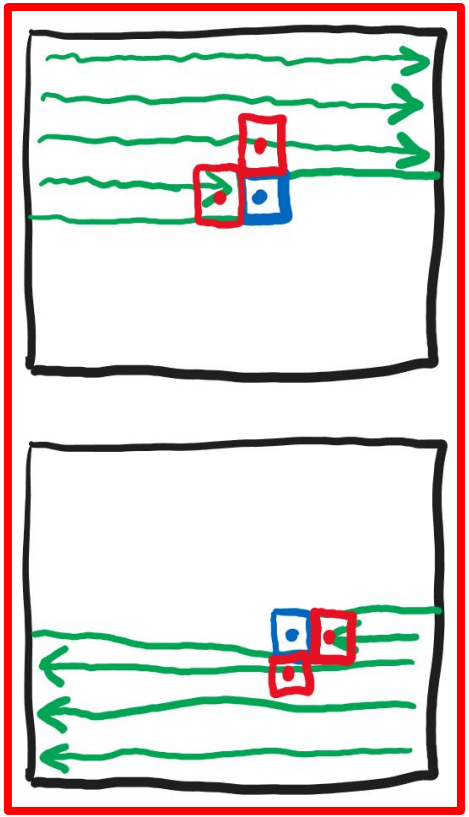


Построение карты глубины методом Patch Match

```
depth_map, normal_map = random()

for level = 0 … N:
    upscale(depth_map, normal_map)
    for iteration = 0 … 100:
        propagation()
        refinement()

return depth_map, normal_map
```



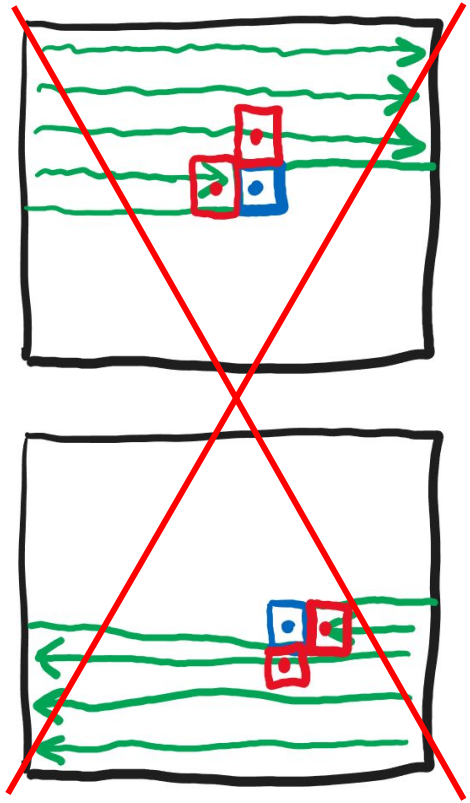
(a) originals (b) random (c) $\frac{1}{4}$ iteration (d) $\frac{3}{4}$ iteration



~~Массовый параллелизм?~~

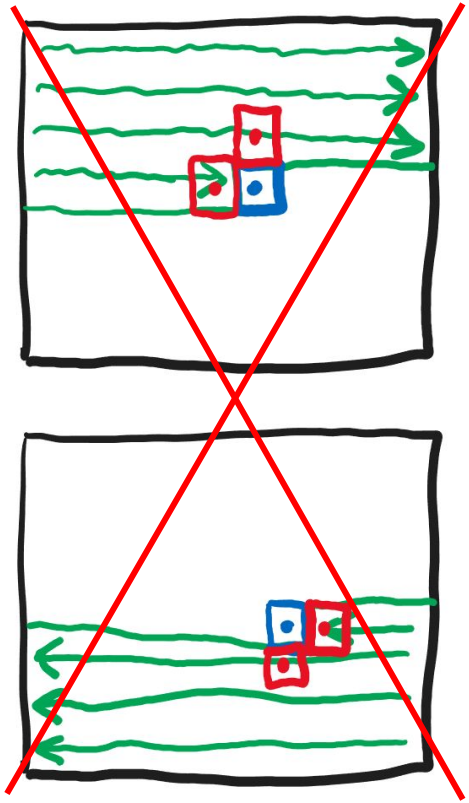
Gipuma

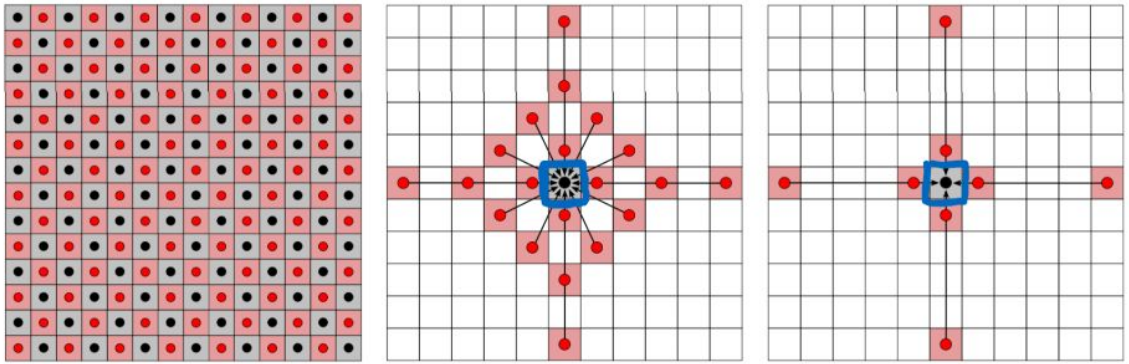
1) **propagation**: мы хотим массовый параллелизм (многоядерные **CPU/GPU**)



Gipuma

1) **propagation**: мы хотим массовый параллелизм (многоядерные **CPU/GPU**)





(a) (b) (c)

Figure 2: The propagation scheme: (a) Depth and normal are updated in parallel for all red pixels, using black pixels as candidates, and vice versa. (b) Planes from a local neighborhood (red points) serve as candidates to update a given pixel (black). (c) Modified scheme for speed setting, using only inner and outermost pixels of the pattern.

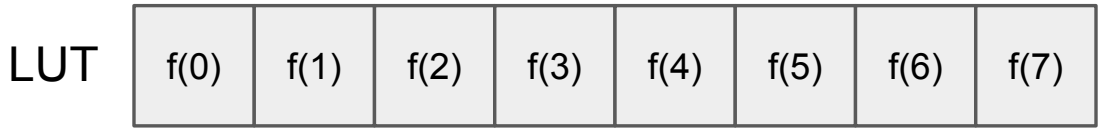
Глава 5:
Look Up Table (LUT)

Примитивная Look Up Table (LUT)

Что если я хочу быстро (пусть и приближенно) считать сложную функцию f(x)?

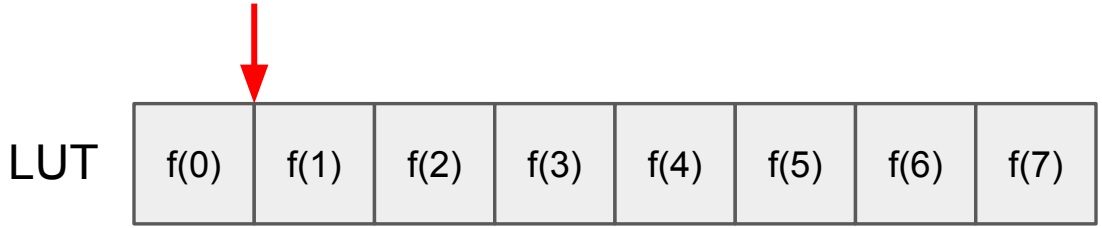
Примитивная Look Up Table (LUT)

Что если я хочу быстро (пусть и приближенно) считать сложную функцию f(x)?



Примитивная Look Up Table (LUT)

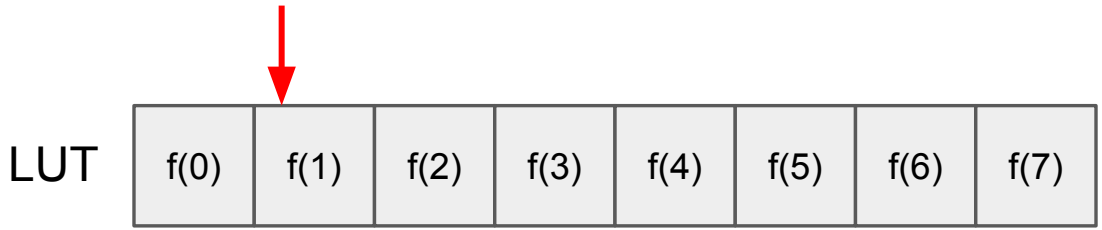
Что если я хочу быстро (пусть и приближенно) считать сложную функцию f(x)?



Что если потребуется f(0.5)?

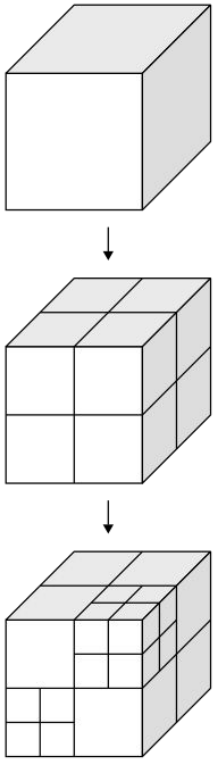
Примитивная Look Up Table (LUT)

Что если я хочу быстро (пусть и приближенно) считать сложную функцию f(x)?

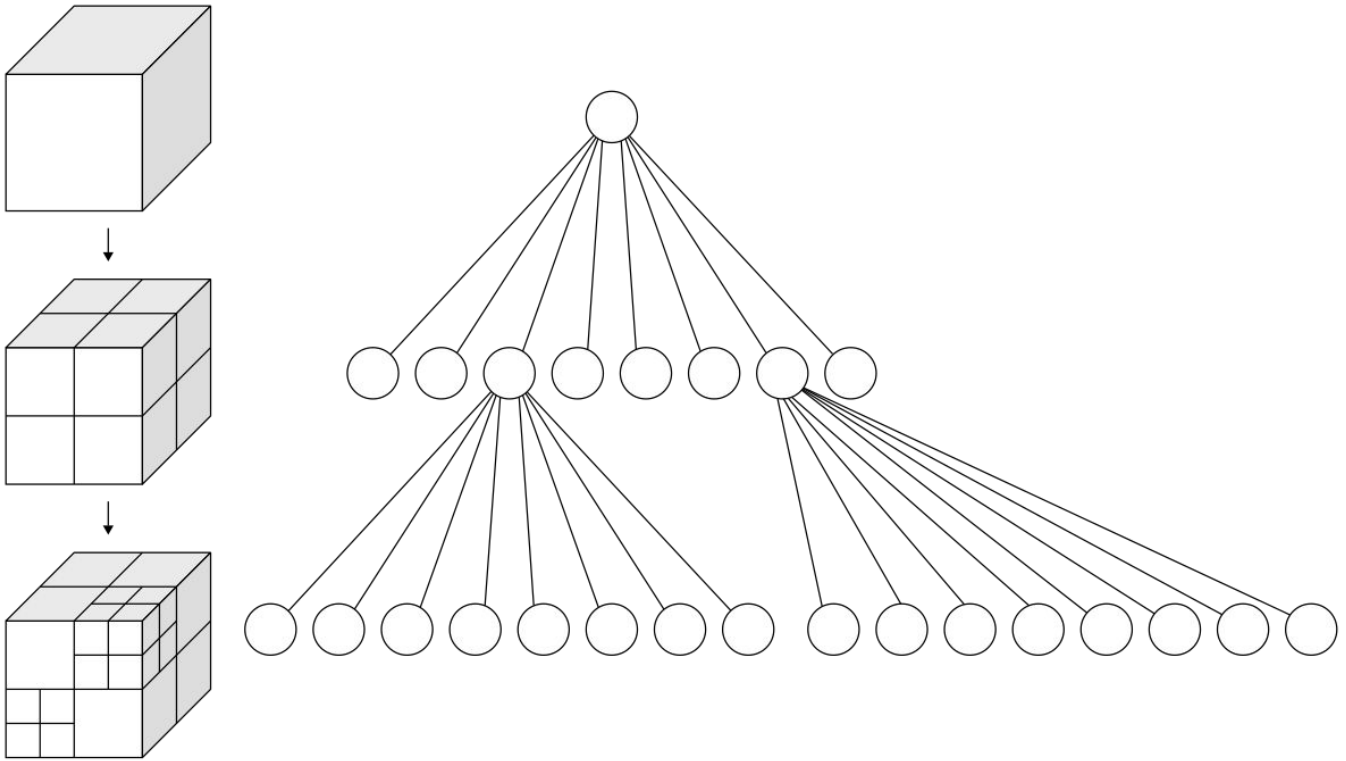


Что если потребуется f(0.75)?

Октодеревья

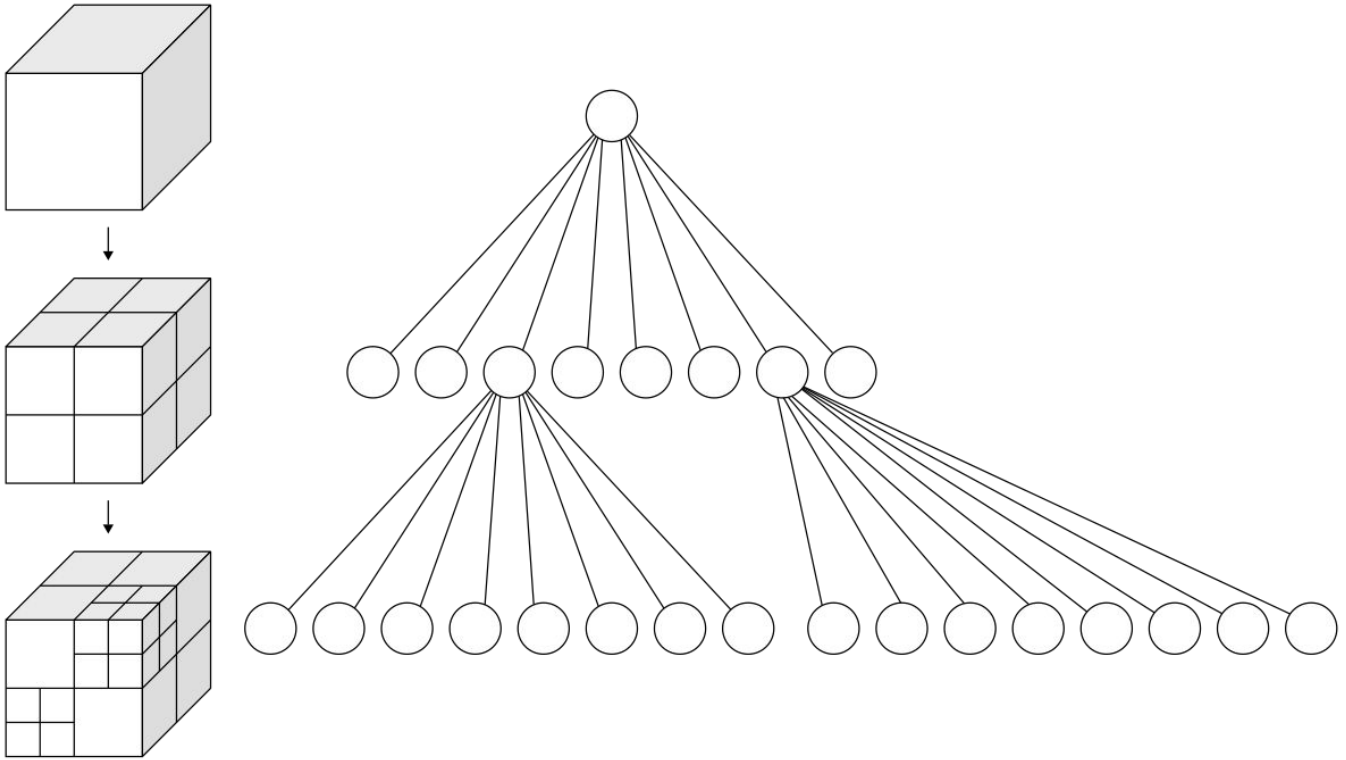


Октодеревья



Октодеревья

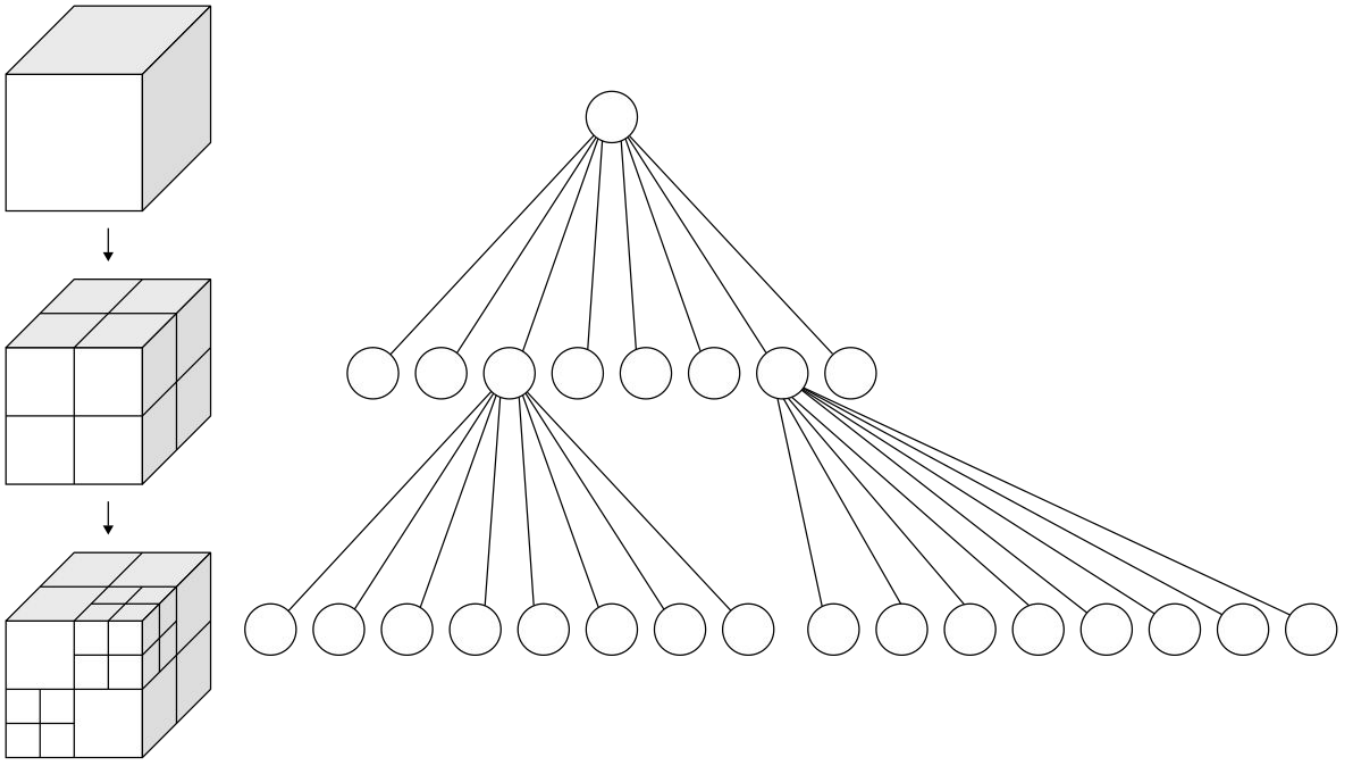
Что нам нужно уметь делать чтобы численную схему на октодереве исполнять?



Октодеревья

Что нам нужно уметь делать чтобы численную схему на октодереве исполнять?

Что нужно чтобы считать частичные производные?

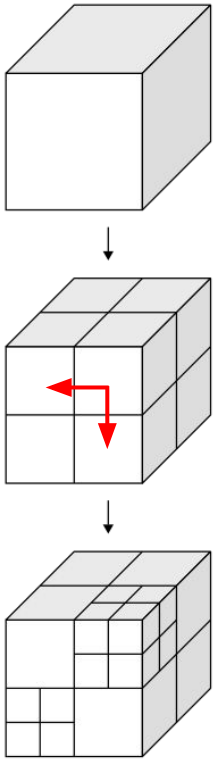


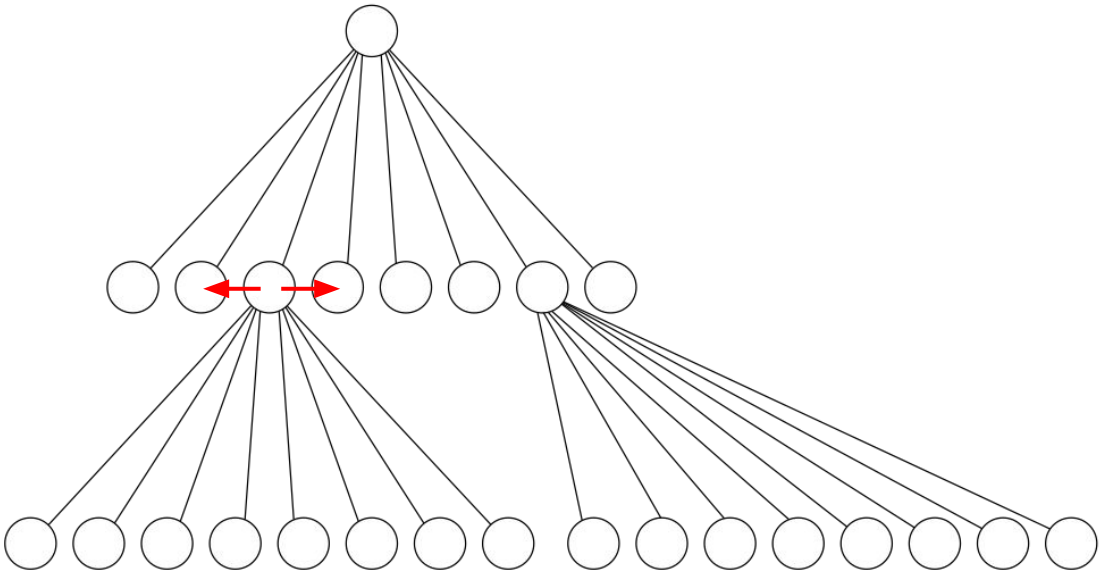
Октодеревья

Что нам нужно уметь делать чтобы численную схему на октодереве исполнять?

Что нужно чтобы считать частичные производные?

Как для каждого узла записать **26 соседей**?



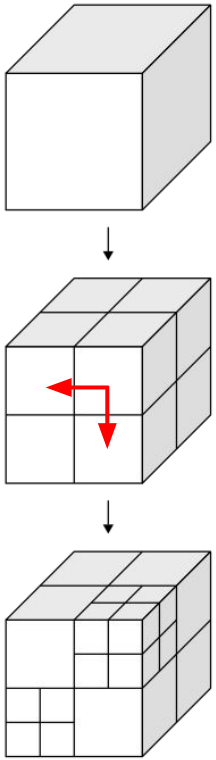


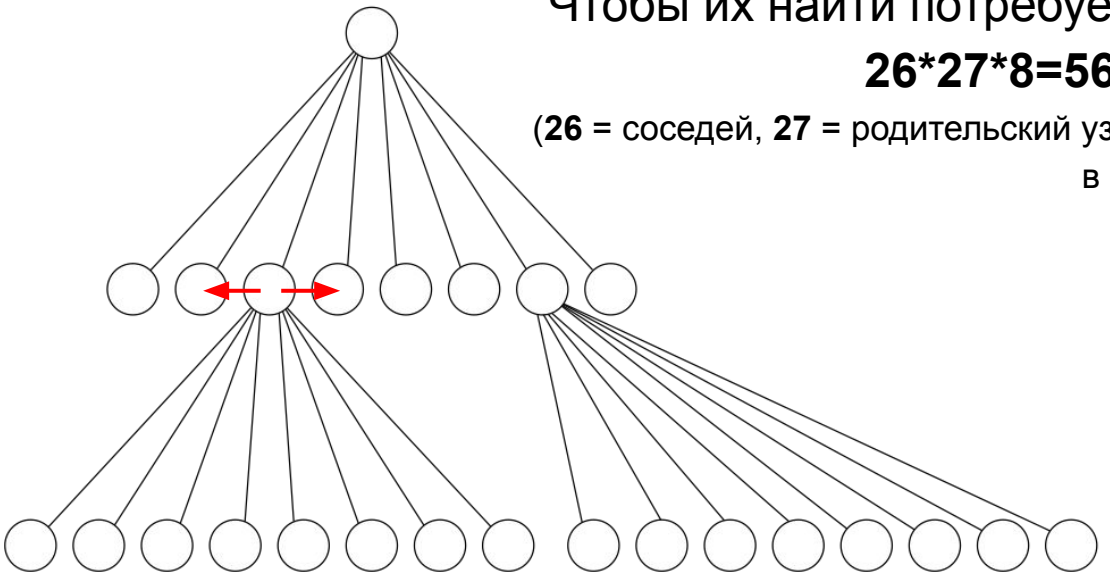
Октодеревья

Что нам нужно уметь делать чтобы численную схему на октодереве исполнять?

Что нужно чтобы считать частичные производные?

Как для каждого узла записать **26 соседей**?

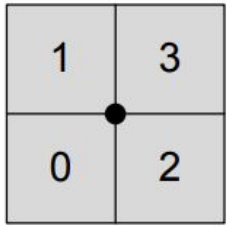




Чтобы их найти потребуется сделать **26*27*8=5616** поисков.

(**26** = соседей, **27** = родительский узел + его соседи, в каждом **8** детей)

Октодеревья: соседи через **Look Up Tables (LUT)**

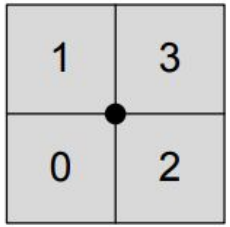


Индекс ребенка

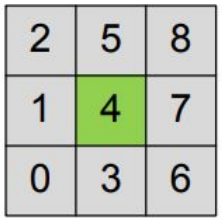
LUTparent[4]

LUTchild[4]

Октодеревья: соседи через **Look Up Tables (LUT)**



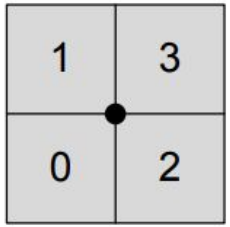
LUTparent[4][9]

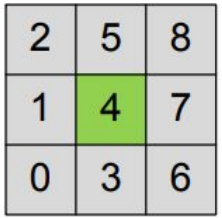


LUTchild[4][9]

Индекс Соседа

Октодеревья: соседи через **Look Up Tables (LUT)**

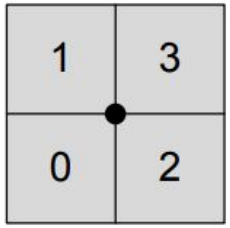


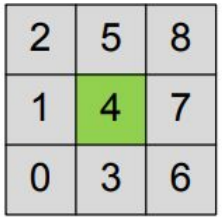


LUTparent[4][9] = {
{0, 1, 1, 3, 4, 4, 3, 4, 4},
{1, 1, 2, 4, 4, 5, 4, 4, 5},
{3, 4, 4, 3, 4, 4, 6, 7, 7},
{4, 4, 5, 4, 4, 5, 7, 7, 8} };

LUTchild[4][9] = {
{3, 2, 3, 1, 0, 1, 3, 2, 3},
{2, 3, 2, 0, 1, 0, 2, 3, 2},
{1, 0, 1, 3, 2, 3, 1, 0, 1},
{0, 1, 0, 2, 3, 2, 0, 1, 0} };

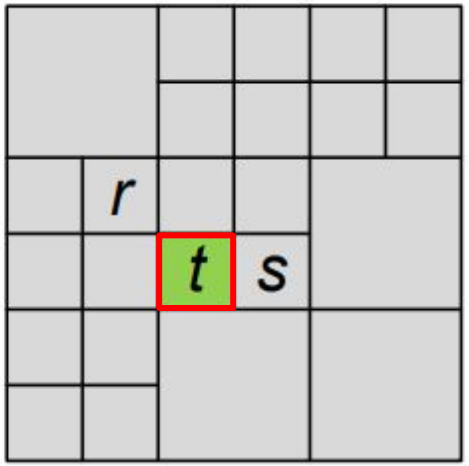
Октодеревья: соседи через **Look Up Tables (LUT)**

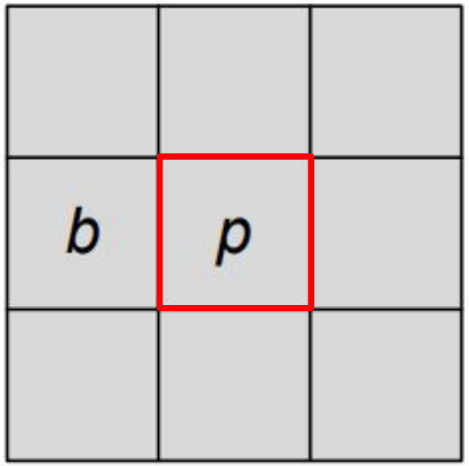




LUTparent[4][9] = {
{0, 1, 1, 3, 4, 4, 3, 4, 4},
{1, 1, 2, 4, 4, 5, 4, 4, 5},
{3, 4, 4, 3, 4, 4, 6, 7, 7},
{4, 4, 5, 4, 4, 5, 7, 7, 8} };

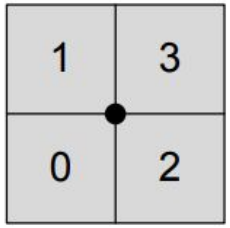
LUTchild[4][9] = {
{3, 2, 3, 1, 0, 1, 3, 2, 3},
{2, 3, 2, 0, 1, 0, 2, 3, 2},
{1, 0, 1, 3, 2, 3, 1, 0, 1},
{0, 1, 0, 2, 3, 2, 0, 1, 0} };

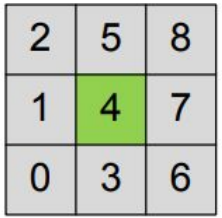




1) LUTparent: Пусть есть узел **t** чей родитель **p**.

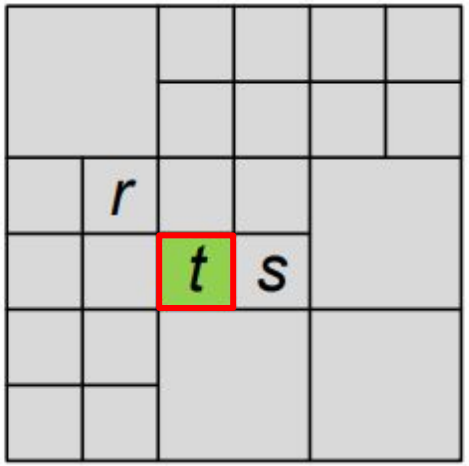
Октодеревья: соседи через **Look Up Tables (LUT)**

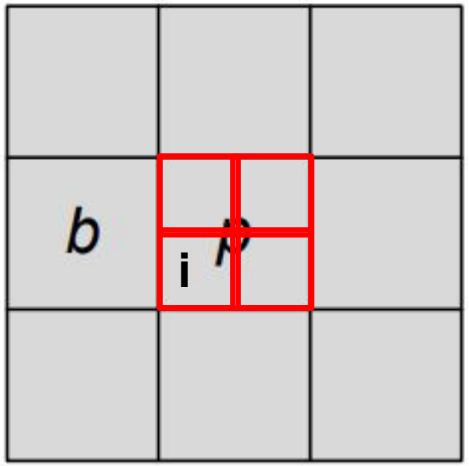




LUTparent[4][9] = {
{0, 1, 1, 3, 4, 4, 3, 4, 4},
{1, 1, 2, 4, 4, 5, 4, 4, 5},
{3, 4, 4, 3, 4, 4, 6, 7, 7},
{4, 4, 5, 4, 4, 5, 7, 7, 8} };

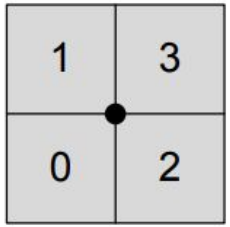
LUTchild[4][9] = {
{3, 2, 3, 1, 0, 1, 3, 2, 3},
{2, 3, 2, 0, 1, 0, 2, 3, 2},
{1, 0, 1, 3, 2, 3, 1, 0, 1},
{0, 1, 0, 2, 3, 2, 0, 1, 0} };

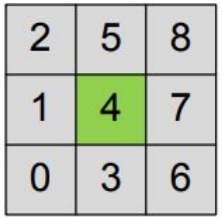




1) LUTparent: Пусть есть узел **t** чей родитель **p**.
Пусть **p.children[i] = t**,

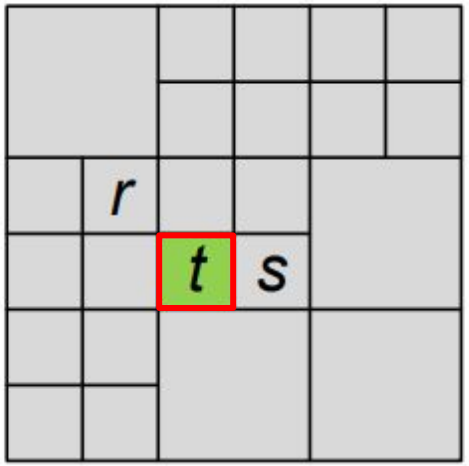
Октодеревья: соседи через **Look Up Tables (LUT)**

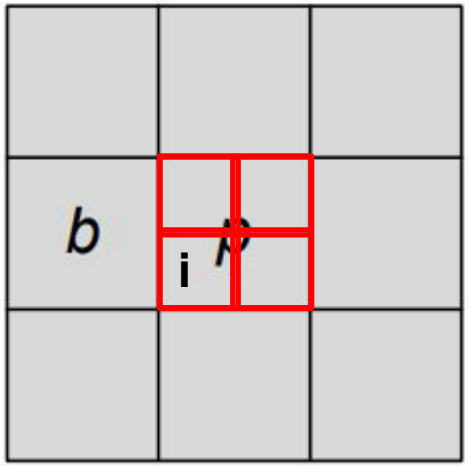




LUTparent[4][9] = {
{0, 1, 1, 3, 4, 4, 3, 4, 4},
{1, 1, 2, 4, 4, 5, 4, 4, 5},
{3, 4, 4, 3, 4, 4, 6, 7, 7},
{4, 4, 5, 4, 4, 5, 7, 7, 8} };

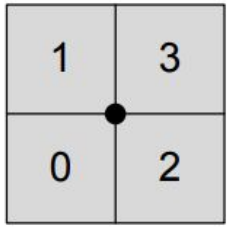
LUTchild[4][9] = {
{3, 2, 3, 1, 0, 1, 3, 2, 3},
{2, 3, 2, 0, 1, 0, 2, 3, 2},
{1, 0, 1, 3, 2, 3, 1, 0, 1},
{0, 1, 0, 2, 3, 2, 0, 1, 0} };

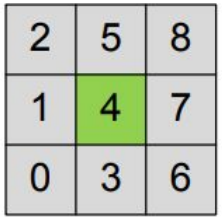




1) LUTparent: Пусть есть узел **t** чей родитель **p**.
Пусть **p.children[i] = t**,
Тогда **parent(t.neighs[j]) = LUTparent[i][j].**

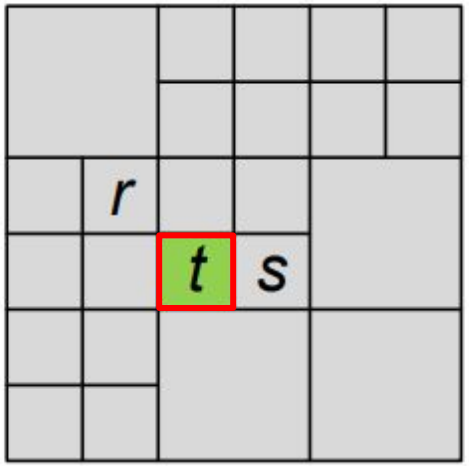
Октодеревья: соседи через **Look Up Tables (LUT)**

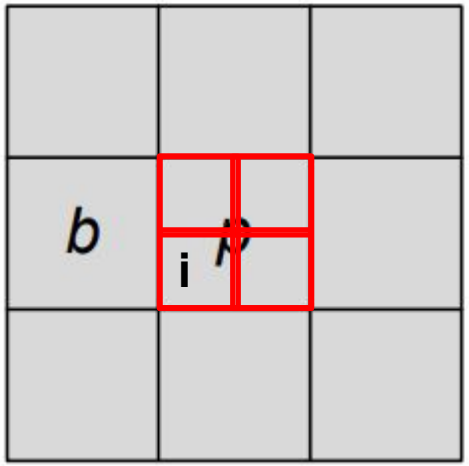




LUTparent[4][9] = {
{0, 1, 1, 3, 4, 4, 3, 4, 4},
{1, 1, 2, 4, 4, 5, 4, 4, 5},
{3, 4, 4, 3, 4, 4, 6, 7, 7},
{4, 4, 5, 4, 4, 5, 7, 7, 8} };

LUTchild[4][9] = {
{3, 2, 3, 1, 0, 1, 3, 2, 3},
{2, 3, 2, 0, 1, 0, 2, 3, 2},
{1, 0, 1, 3, 2, 3, 1, 0, 1},
{0, 1, 0, 2, 3, 2, 0, 1, 0} };

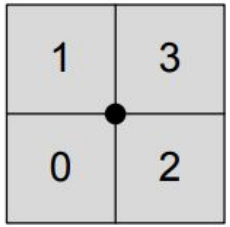


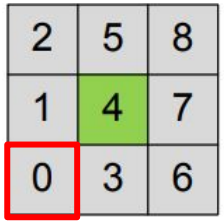


1) LUTparent: Пусть есть узел **t** чей родитель **p**.
Пусть **p.children[i] = t**,
Тогда **parent(t.neighs[j]) = LUTparent[i][j].**

2) LUTchild: Пусть есть узел **t** чей родитель **p**. Пусть **p.children[i] = t**.

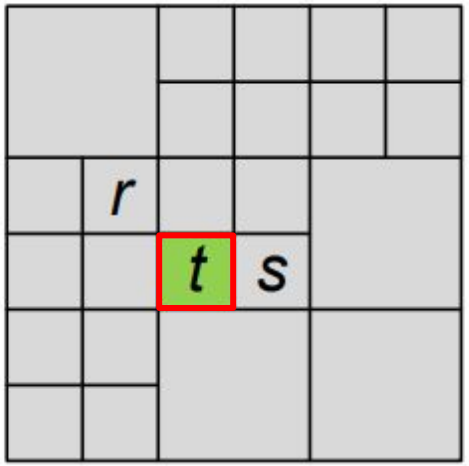
Октодеревья: соседи через **Look Up Tables (LUT)**

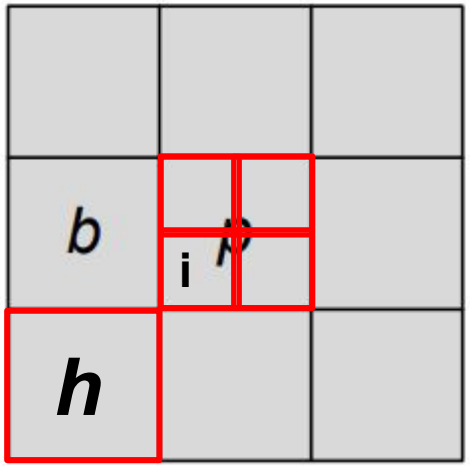




LUTparent[4][9] = {
{0, 1, 1, 3, 4, 4, 3, 4, 4},
{1, 1, 2, 4, 4, 5, 4, 4, 5},
{3, 4, 4, 3, 4, 4, 6, 7, 7},
{4, 4, 5, 4, 4, 5, 7, 7, 8} };

LUTchild[4][9] = {
{3, 2, 3, 1, 0, 1, 3, 2, 3},
{2, 3, 2, 0, 1, 0, 2, 3, 2},
{1, 0, 1, 3, 2, 3, 1, 0, 1},
{0, 1, 0, 2, 3, 2, 0, 1, 0} };

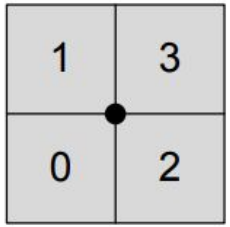


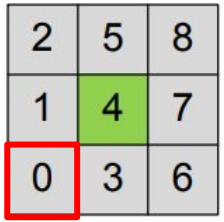


1) LUTparent: Пусть есть узел **t** чей родитель **p**.
Пусть **p.children[i] = t**,
Тогда **parent(t.neighs[j]) = LUTparent[i][j].**

2) LUTchild: Пусть есть узел **t** чей родитель **p**. Пусть **p.children[i] = t**.
Пусть **parent(t.neighs[j]) = h,**

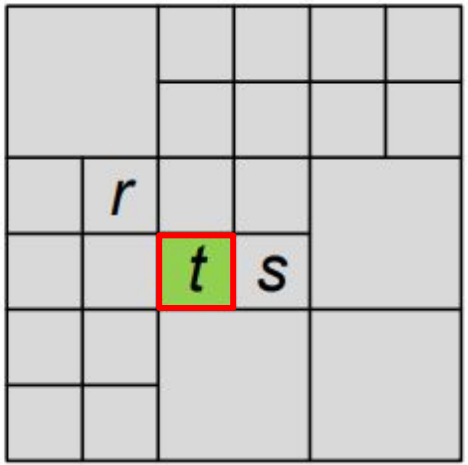
Октодеревья: соседи через **Look Up Tables (LUT)**

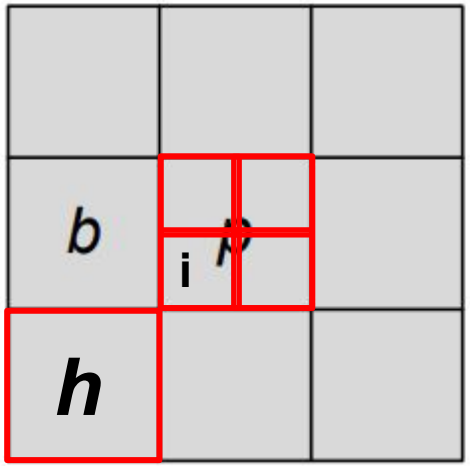




LUTparent[4][9] = {
{0, 1, 1, 3, 4, 4, 3, 4, 4},
{1, 1, 2, 4, 4, 5, 4, 4, 5},
{3, 4, 4, 3, 4, 4, 6, 7, 7},
{4, 4, 5, 4, 4, 5, 7, 7, 8} };

LUTchild[4][9] = {
{3, 2, 3, 1, 0, 1, 3, 2, 3},
{2, 3, 2, 0, 1, 0, 2, 3, 2},
{1, 0, 1, 3, 2, 3, 1, 0, 1},
{0, 1, 0, 2, 3, 2, 0, 1, 0} };



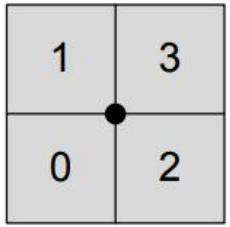


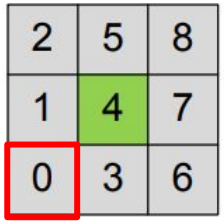
Где тогда наш сосед?

1) LUTparent: Пусть есть узел **t** чей родитель **p**.
Пусть **p.children[i] = t**,
Тогда **parent(t.neighs[j]) = LUTparent[i][j].**

2) LUTchild: Пусть есть узел **t** чей родитель **p**. Пусть **p.children[i] = t**.
Пусть **parent(t.neighs[j]) = h,**

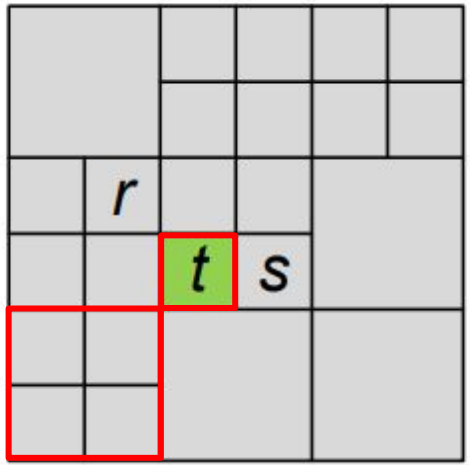
Октодеревья: соседи через **Look Up Tables (LUT)**

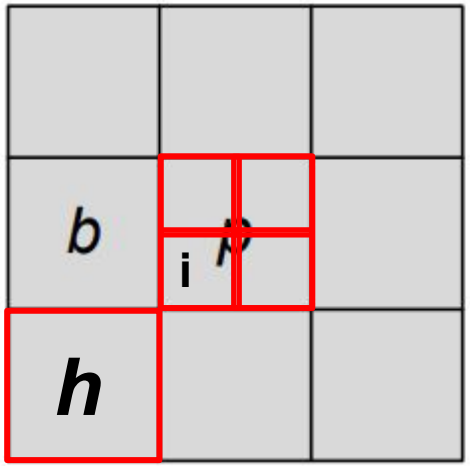




LUTparent[4][9] = {
{0, 1, 1, 3, 4, 4, 3, 4, 4},
{1, 1, 2, 4, 4, 5, 4, 4, 5},
{3, 4, 4, 3, 4, 4, 6, 7, 7},
{4, 4, 5, 4, 4, 5, 7, 7, 8} };

LUTchild[4][9] = {
{3, 2, 3, 1, 0, 1, 3, 2, 3},
{2, 3, 2, 0, 1, 0, 2, 3, 2},
{1, 0, 1, 3, 2, 3, 1, 0, 1},
{0, 1, 0, 2, 3, 2, 0, 1, 0} };



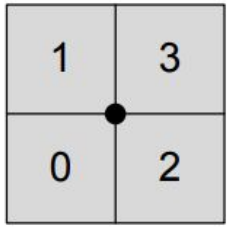


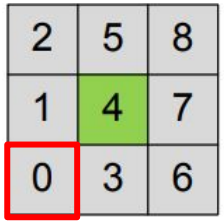
Чем однозначно определяется кто именно?

1) LUTparent: Пусть есть узел **t** чей родитель **p**.
Пусть **p.children[i] = t**,
Тогда **parent(t.neighs[j]) = LUTparent[i][j].**

2) LUTchild: Пусть есть узел **t** чей родитель **p**. Пусть **p.children[i] = t**.
Пусть **parent(t.neighs[j]) = h,**

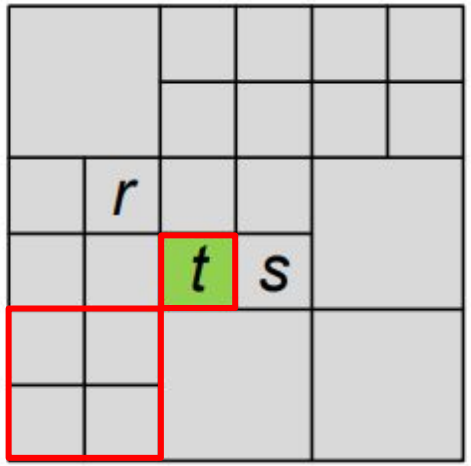
Октодеревья: соседи через **Look Up Tables (LUT)**

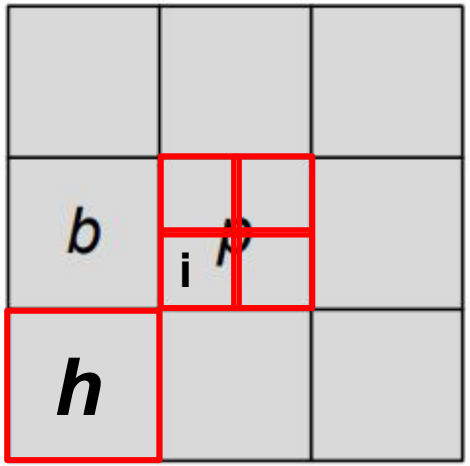




LUTparent[4][9] = {
{0, 1, 1, 3, 4, 4, 3, 4, 4},
{1, 1, 2, 4, 4, 5, 4, 4, 5},
{3, 4, 4, 3, 4, 4, 6, 7, 7},
{4, 4, 5, 4, 4, 5, 7, 7, 8} };

LUTchild[4][9] = {
{3, 2, 3, 1, 0, 1, 3, 2, 3},
{2, 3, 2, 0, 1, 0, 2, 3, 2},
{1, 0, 1, 3, 2, 3, 1, 0, 1},
{0, 1, 0, 2, 3, 2, 0, 1, 0} };

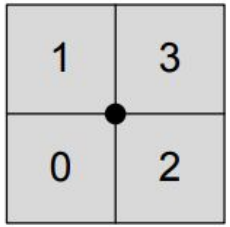


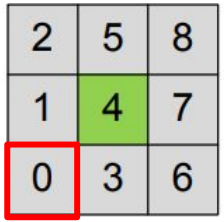


1) LUTparent: Пусть есть узел **t** чей родитель **p**.
Пусть **p.children[i] = t**,
Тогда **parent(t.neighs[j]) = LUTparent[i][j].**

2) LUTchild: Пусть есть узел **t** чей родитель **p**. Пусть **p.children[i] = t**.
Пусть **parent(t.neighs[j]) = h,**
Тогда **t.neigh[j] = h.children[LUTchild[i][j]].**

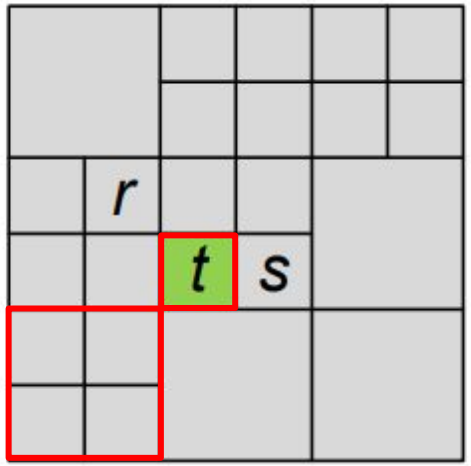
Октодеревья: соседи через **Look Up Tables (LUT)**

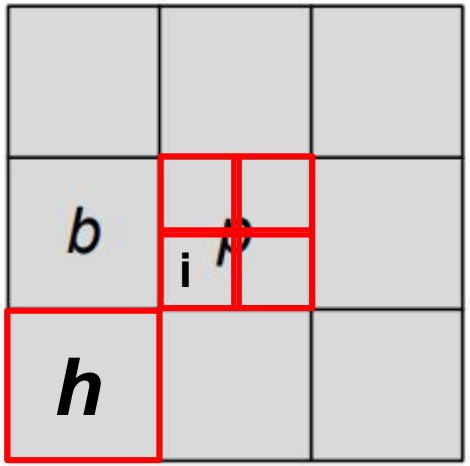




LUTparent[4][9] = {
{0, 1, 1, 3, 4, 4, 3, 4, 4},
{1, 1, 2, 4, 4, 5, 4, 4, 5},
{3, 4, 4, 3, 4, 4, 6, 7, 7},
{4, 4, 5, 4, 4, 5, 7, 7, 8} };

LUTchild[4][9] = {
{3, 2, 3, 1, 0, 1, 3, 2, 3},
{2, 3, 2, 0, 1, 0, 2, 3, 2},
{1, 0, 1, 3, 2, 3, 1, 0, 1},
{0, 1, 0, 2, 3, 2, 0, 1, 0} };



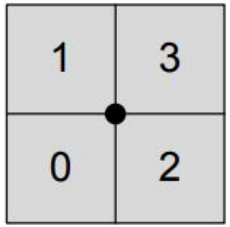


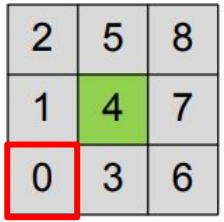
1) LUTparent: Пусть есть узел **t** чей родитель **p**.
Пусть **p.children[i] = t**,
Тогда **parent(t.neighs[j]) = LUTparent[i][j].**

2) LUTchild: Пусть есть узел **t** чей родитель **p**.
Пусть **parent(t.neighs[j]) = h,**
Тогда **t.neigh[j] = h.children[LUTchild[i][j]].**

Есть ли способ проще?

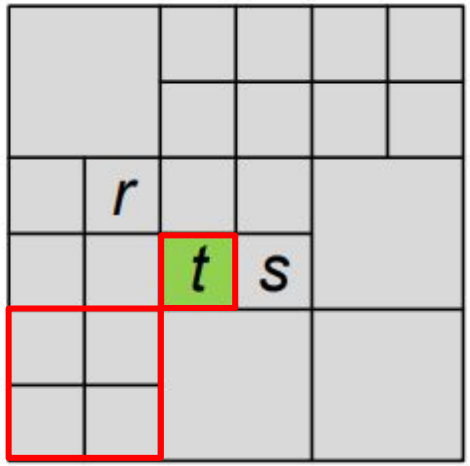
Октодеревья: соседи через **Look Up Tables (LUT)**

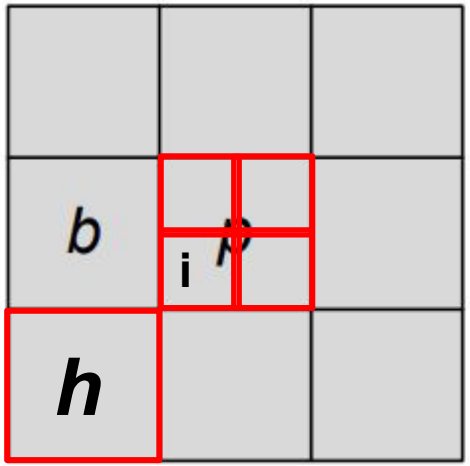




LUTparent[4][9] = {
{0, 1, 1, 3, 4, 4, 3, 4, 4},
{1, 1, 2, 4, 4, 5, 4, 4, 5},
{3, 4, 4, 3, 4, 4, 6, 7, 7},
{4, 4, 5, 4, 4, 5, 7, 7, 8} };

LUTchild[4][9] = {
{3, 2, 3, 1, 0, 1, 3, 2, 3},
{2, 3, 2, 0, 1, 0, 2, 3, 2},
{1, 0, 1, 3, 2, 3, 1, 0, 1},
{0, 1, 0, 2, 3, 2, 0, 1, 0} };





1) LUTparent: Пусть есть узел **t** чей родитель **p**.
Пусть **p.children[i] = t**,
Тогда **parent(t.neighs[j]) = LUTparent[i][j].**

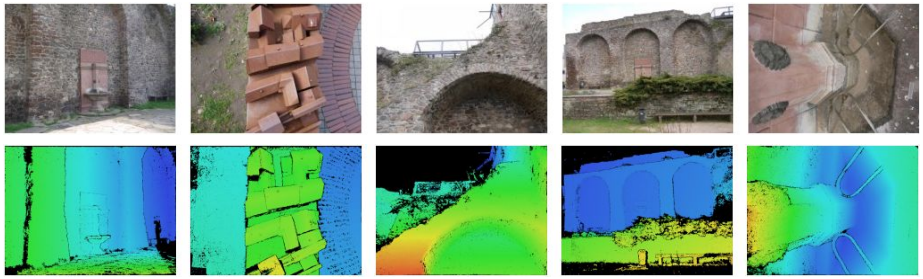
2) LUTchild: Пусть есть узел **t** чей родитель **p**.
Пусть **parent(t.neighs[j]) = h,**
Тогда **t.neigh[j] = h.children[LUTchild[i][j]].**

Можно **бин. поиском** по всем узлам найти… Нужно только ввести хоть какой-то **порядок** по узлам дерева, даже не обязательно **Z-curve/Huber/Morton Codes.**

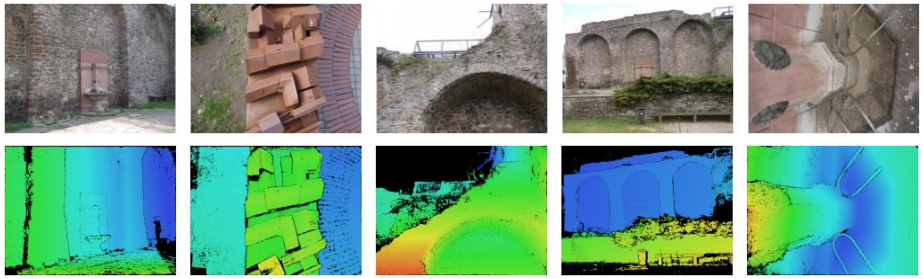
Октодеревья: соседи через **Look Up Tables (LUT)**

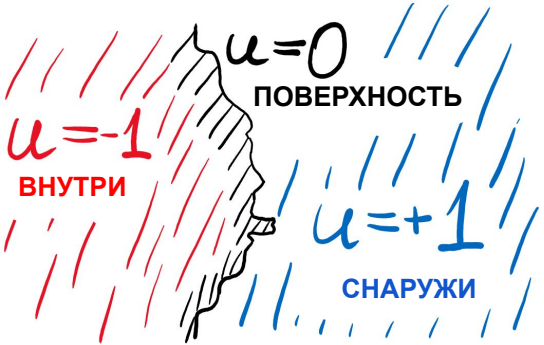
```
LUTparents = []
for iy in range(2):
    for ix in range(2):
        i = iy * 2 + ix

        parents = []
        for jy in range(3):
            for jx in range(3):
                j = jy * 3 + jx
                globalx, globaly = 2 + ix + (jx - 1), 2 + iy + (jy - 1)
                px, py = int(globalx // 2), int(globaly // 2)
                parents.append(py * 3 + px)

        LUTparents.append(parents)

def LUTparent(i, j):
    ix, iy, iz = i % 2, (i // 2) % 2, (i // 4)
    jx, jy, jz = i % 3, (i // 3) % 3, (i // 9)
    globalx, globaly, globalz = ...
    px, py, pz = ...
    return pz * 3 * 3 + py * 3 + px
```

Look Up Tables (LUT)

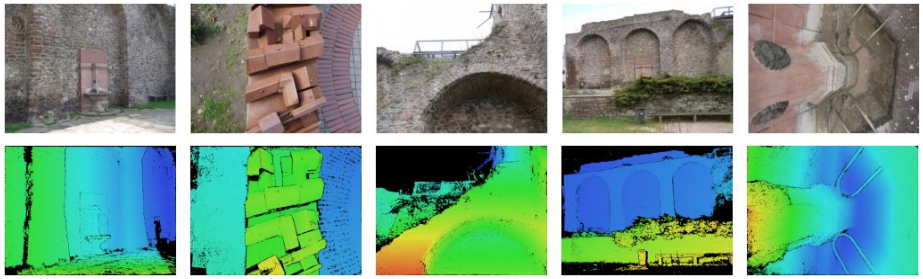


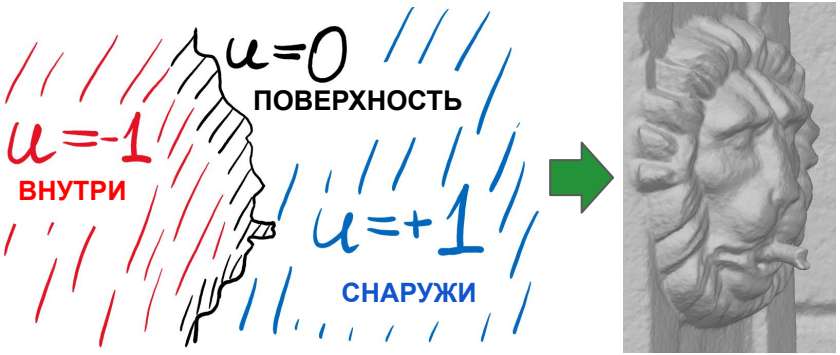
Look Up Tables (LUT)



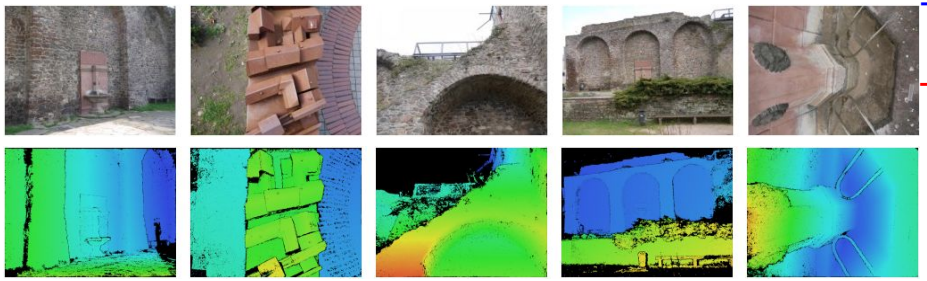


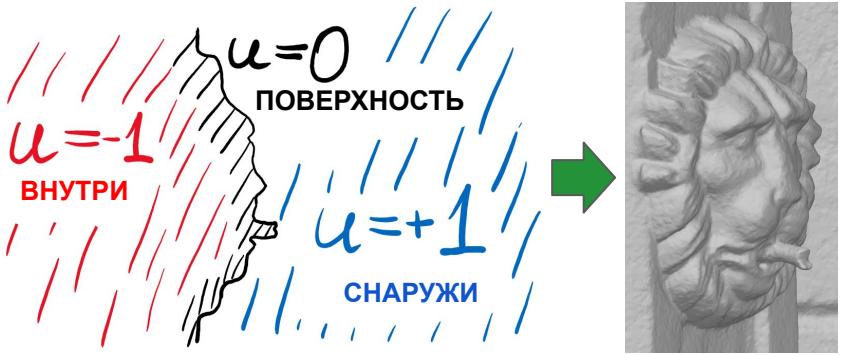
Look Up Tables (LUT)



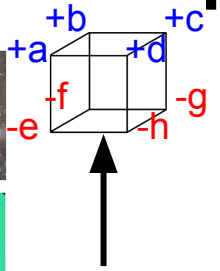


Look Up Tables (LUT)





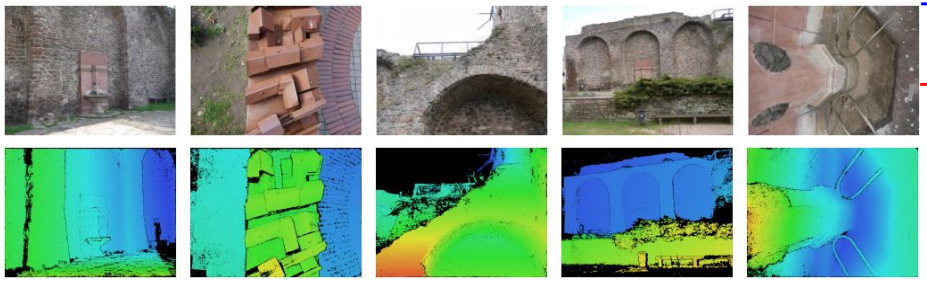
Маршировка кубов

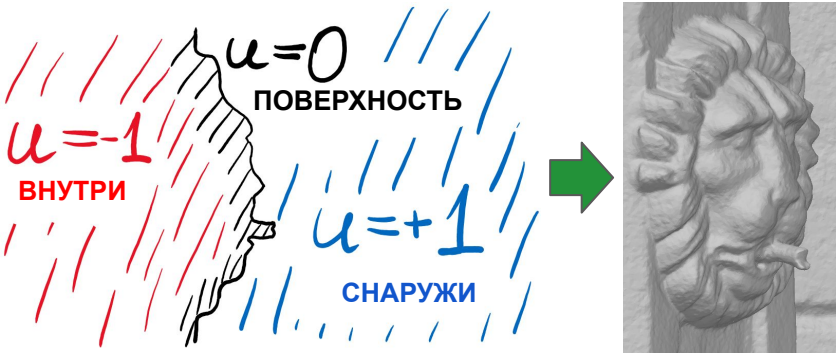


рассмотрим воксель нашего пространства

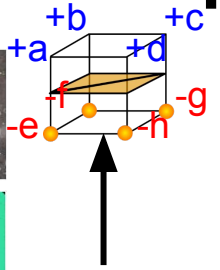
как зная индикаторное поле на его 8 углах понять где проходит поверхность?

Look Up Tables (LUT)





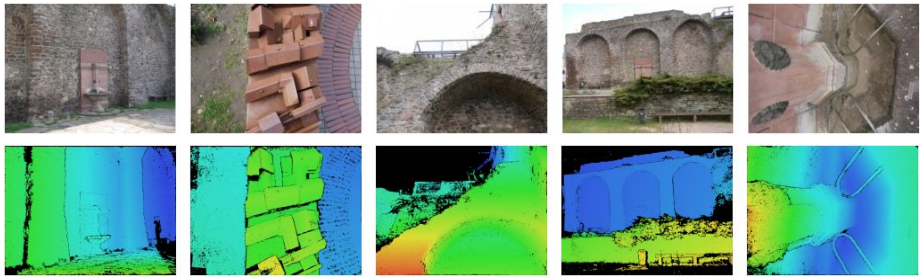
Маршировка кубов

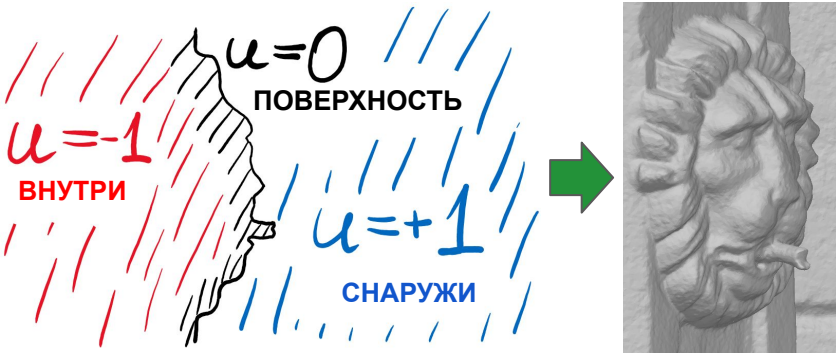


рассмотрим воксель нашего пространства

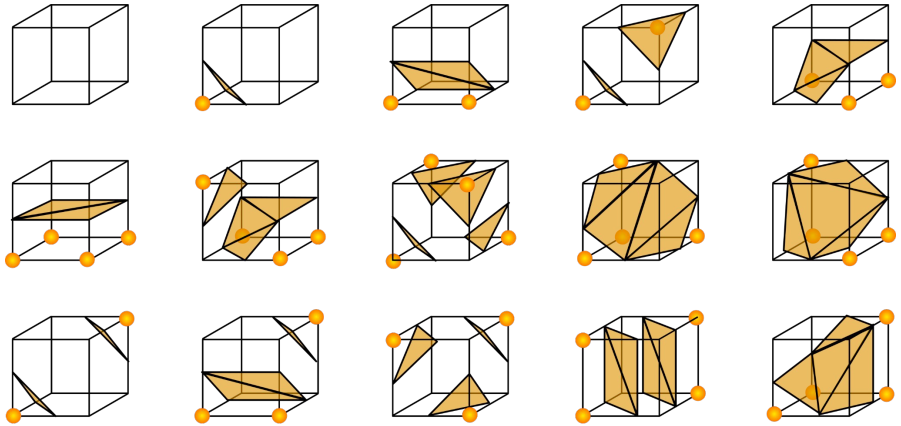
как зная индикаторное поле на его 8 углах понять где проходит поверхность?

Look Up Tables (LUT)

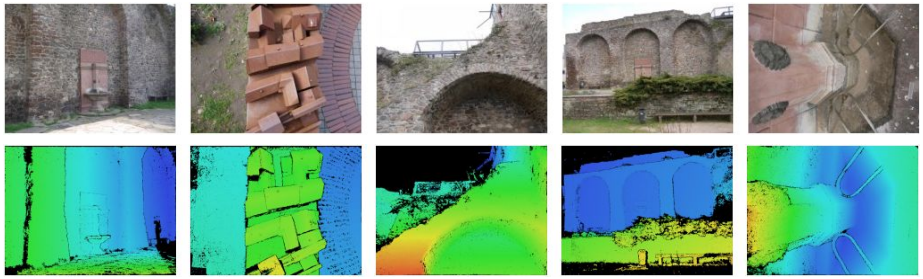


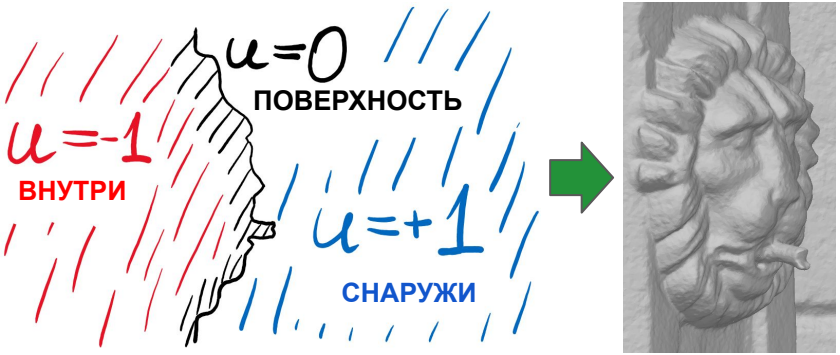


Маршировка кубов

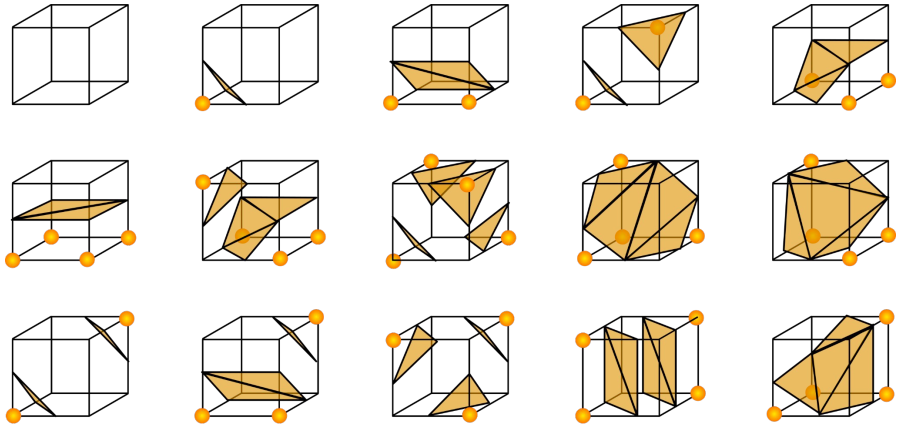


Look Up Tables (LUT)



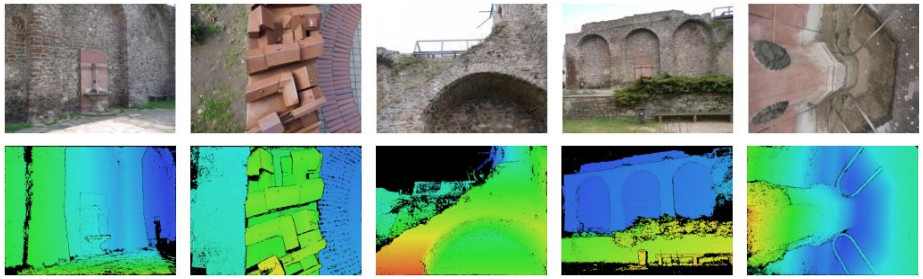


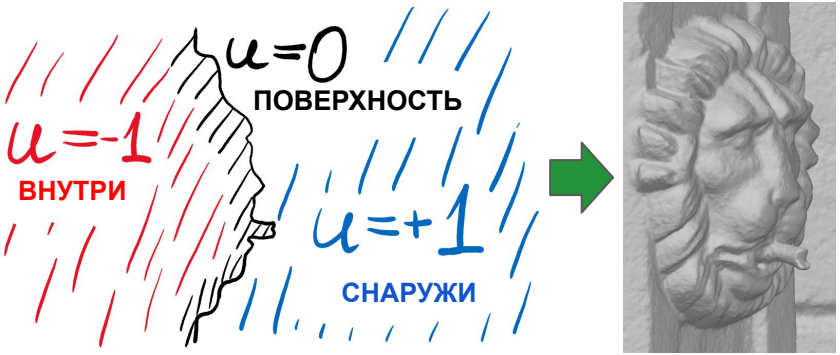
Маршировка кубов



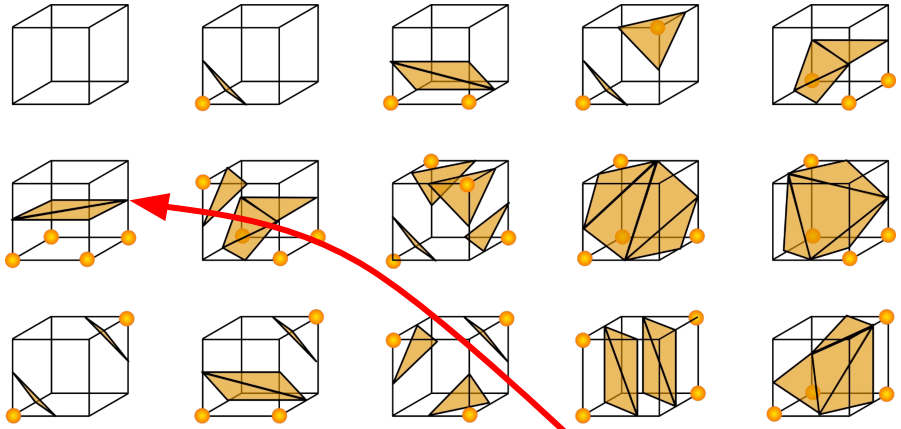
А сколько всего случаев?

Look Up Tables (LUT)





Маршировка кубов



А сколько всего случаев?

А где между углами вокселя ставить вершину?

CS Space
Клуб технологий и науки

Вопросы?

@UnicornGlade
@PolarNick239
polarnick239@gmail.com
Николай Полярный