CS Space
Клуб технологий и науки

# Вычисления на видеокартах

Лекция 5

- Битовая арифметика
- Префиксная сумма (Scan)
- Как отлаживать
- OpenCL/CUDA самописный эмулятор


polarnick239@gmail.com
Николай Полярный

# Битовая арифметика

45 = как представить ввиде суммы степеней 2-ки?

# Битовая арифметика

$45 = 32 + \ldots$

$45_2 = \overset{32}{1} \ldots$

# Битовая арифметика

$45 = 32 + \cancel{16} + \ldots$

| | 32 | 16 | |
|---|---|---|---|
| $45_2 =$ | 1 | 0 | … |

# Битовая арифметика

$45 = 32 + \cancel{16} + 8 + \ldots$

| | 32 | 16 | 8 | |
|---|---|---|---|---|
| $45_2 =$ | 1 | 0 | 1 | ... |

# Битовая арифметика

$45 = 32 + \cancel{16} + 8 + 4 + \cancel{2} + 1$

| | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|
| $45_2 =$ | 1 | 0 | 1 | 1 | 0 | 1 |

Разложение однозначно?

# Битовая арифметика

$45 = 32 + \cancel{16} + 8 + 4 + \cancel{2} + 1$

| | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|
| $45_2 =$ | 1 | 0 | 1 | 1 | 0 | 1 |

Разложение однозначно?

Может ли какая-то степень встретиться дважды?

# Битовая арифметика

$45 = 32 + \sout{16} + 8 + 4 + \sout{2} + 1$

| | 32 | 16 | 8 | 4 | 2 | 1 | |
|---|---|---|---|---|---|---|---|
| $45_2 =$ | 1 | 0 | 1 | 1 | 0 | 1 | |
| $45_2 >> 1$ | ? | ? | ? | ? | ? | ? | Сдвиг вправо на 1 бит |

# Битовая арифметика

$45 = 32 + \sout{16} + 8 + 4 + \sout{2} + 1$

| | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|
| $45_2 =$ | 1 | 0 | 1 | 1 | 0 | 1 |
| $45_2 >> 1$ | 0 | 1 | 0 | 1 | 1 | 0 |

Сдвиг вправо на 1 бит

# Битовая арифметика

$45 = 32 + \sout{16} + 8 + 4 + \sout{2} + 1$

| | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|
| $45_2 =$ | 1 | 0 | 1 | 1 | 0 | 1 |
| $45_2 >> 1$ | 0 | 1 | 0 | 1 | 1 | 0 |

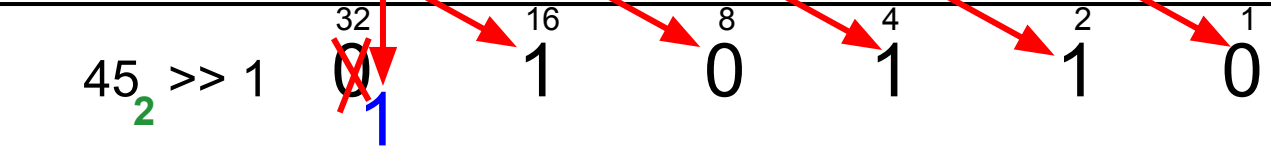
Сдвиг вправо на 1 бит

Какой мат. операции эквивалентен?

# Битовая арифметика

$45 = 32 + \cancel{16} + 8 + 4 + \cancel{2} + 1$

| | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|
| $45_2 =$ | 1 | 0 | 1 | 1 | 0 | 1 |
| $45_2 >> 1$ | ~~0~~ 1 | 1 | 0 | 1 | 1 | 0 |



Будьте осторожны если знаковый тип!

Сдвиг вправо на 1 бит

Какой мат. операции эквивалентен?

# Битовая арифметика

$45 = 32 + \sout{16} + 8 + 4 + \sout{2} + 1$

| | 32 | 16 | 8 | 4 | 2 | 1 | |
|---|---|---|---|---|---|---|---|
| $45_2 =$ | 1 | 0 | 1 | 1 | 0 | 1 | |
| $45_2 >> 1$ | 0 | 1 | 0 | 1 | 1 | 0 | Сдвиг вправо на 1 бит |
| $45_2 << 1$ | ? | ? | ? | ? | ? | ? | Сдвиг влево на 1 бит |

# Битовая арифметика

$45 = 32 + \cancel{16} + 8 + 4 + \cancel{2} + 1$

| | 32 | 16 | 8 | 4 | 2 | 1 | |
|---|---|---|---|---|---|---|---|
| $45_2 =$ | 1 | 0 | 1 | 1 | 0 | 1 | |
| $45_2 >> 1$ | 0 | 1 | 0 | 1 | 1 | 0 | Сдвиг вправо на 1 бит |
| $45_2 << 1$ | 0 | 1 | 1 | 0 | 1 | 0 | Сдвиг влево на 1 бит |

# Битовая арифметика

$45 = 32 + \sout{16} + 8 + 4 + \sout{2} + 1$

| | 32 | 16 | 8 | 4 | 2 | 1 | |
|---|---|---|---|---|---|---|---|
| $45_2 =$ | 1 | 0 | 1 | 1 | 0 | 1 | |
| $45_2 >> 1$ | 0 | 1 | 0 | 1 | 1 | 0 | Сдвиг вправо на 1 бит |
| $45_2 << 1$ | 0 | 1 | 1 | 0 | 1 | 0 | Сдвиг влево на 1 бит |

Какой мат. операции эквивалентен?

# Битовая арифметика

$45 = 32 + \sout{16} + 8 + 4 + \sout{2} + 1$

| | 32 | 16 | 8 | 4 | 2 | 1 | |
|---|---|---|---|---|---|---|---|
| $45_2 =$ | 1 | 0 | 1 | 1 | 0 | 1 | |
| $45_2 >> 1$ | 0 | 1 | 0 | 1 | 1 | 0 | Сдвиг вправо на 1 бит |
| $45_2 << 1$ | 0 | 1 | 1 | 0 | 1 | 0 | Сдвиг влево на 1 бит |

Как проверить
“единица ли в k-ом бите”?

# Битовая арифметика

$45 = 32 + \sout{16} + 8 + 4 + \sout{2} + 1$

| | 32 | 16 | 8 | 4 | 2 | 1 | |
|---|---|---|---|---|---|---|---|
| $45_2 =$ | 1 | 0 | 1 | 1 | 0 | 1 | |
| $45_2 >> 1$ | 0 | 1 | 0 | 1 | 1 | 0 | Сдвиг вправо на 1 бит |
| $45_2 << 1$ | 0 | 1 | 1 | 0 | 1 | 0 | Сдвиг влево на 1 бит |

(1 << k) ← k=4 — 1

Как проверить
“единица ли в k-ом бите”?

# Битовая арифметика

$45 = 32 + \sout{16} + 8 + 4 + \sout{2} + 1$

| | 32 | 16 | 8 | 4 | 2 | 1 | |
|---|---|---|---|---|---|---|---|
| $45_2 =$ | 1 | 0 | 1 | 1 | 0 | 1 | |
| $45_2 >> 1$ | 0 | 1 | 0 | 1 | 1 | 0 | Сдвиг вправо на 1 бит |
| $45_2 << 1$ | 0 | 1 | 1 | 0 | 1 | 0 | Сдвиг влево на 1 бит |

(1 << k) 1 ← k=4

Как проверить
“единица ли в k-ом бите”?

# Битовая арифметика

$45 = 32 + \sout{16} + 8 + 4 + \sout{2} + 1$

| | 32 | 16 | 8 | 4 | 2 | 1 | |
|---|---|---|---|---|---|---|---|
| $45_2 =$ | 1 | 0 | 1 | 1 | 0 | 1 | |
| $45_2 >> 1$ | 0 | 1 | 0 | 1 | 1 | 0 | Сдвиг вправо на 1 бит |
| $45_2 << 1$ | 0 | 1 | 1 | 0 | 1 | 0 | Сдвиг влево на 1 бит |

a & (1 << k)

1 ← k=4

Как проверить
“единица ли в k-ом бите”?

# Битовая арифметика

$45 = 32 + \sout{16} + 8 + 4 + \sout{2} + 1$

| | 32 | 16 | 8 | 4 | 2 | 1 | |
|---|---|---|---|---|---|---|---|
| $45_2 =$ | 1 | 0 | 1 | 1 | 0 | 1 | |
| $45_2 >> 1$ | 0 | 1 | 0 | 1 | 1 | 0 | Сдвиг вправо на 1 бит |
| $45_2 << 1$ | 0 | 1 | 1 | 0 | 1 | 0 | Сдвиг влево на 1 бит |

a & (1 << k)

1 ← k=4

Проверяем
“единица ли в k-ом бите”

Как выставить в числе
“единицу в k-ом бите”?

# Битовая арифметика

$45 = 32 + \sout{16} + 8 + 4 + \sout{2} + 1$

| | 32 | 16 | 8 | 4 | 2 | 1 | |
|---|---|---|---|---|---|---|---|
| $45_2 =$ | 1 | 0 | 1 | 1 | 0 | 1 | |
| $45_2 >> 1$ | 0 | 1 | 0 | 1 | 1 | 0 | Сдвиг вправо на 1 бит |
| $45_2 << 1$ | 0 | 1 | 1 | 0 | 1 | 0 | Сдвиг влево на 1 бит |

a & (1 << k) → 1 (k=4)

Проверяем “единица ли в k-ом бите”

a | (1 << k) → 1 (k=4)

Как выставить в числе “единицу в k-ом бите”?

# Битовая арифметика

$45 = 32 + \sout{16} + 8 + 4 + \sout{2} + 1$

| | 32 | 16 | 8 | 4 | 2 | 1 | |
|---|---|---|---|---|---|---|---|
| $45_2 =$ | 1 | 0 | 1 | 1 | 0 | 1 | |
| $45_2 >> 1$ | 0 | 1 | 0 | 1 | 1 | 0 | Сдвиг вправо на 1 бит |
| $45_2 << 1$ | 0 | 1 | 1 | 0 | 1 | 0 | Сдвиг влево на 1 бит |

a & (1 << k) → 1 (k=4) — Проверяем “единица ли в k-ом бите”

a | (1 << k) → 1 (k=4) — Выставляем в числе “единицу в k-ом бите”

Как узнать число единиц?

# Битовая арифметика

$45 = 32 + \sout{16} + 8 + 4 + \sout{2} + 1$

$45_2 =$ 32: 1, 16: 0, 8: 1, 4: 1, 2: 0, 1: 1

| | | |
|---|---|---|
| $45_2 >> 1$ | 32: 0, 16: 1, 8: 0, 4: 1, 2: 1, 1: 0 | Сдвиг вправо на 1 бит |
| $45_2 << 1$ | 32: 0, 16: 1, 8: 1, 4: 0, 2: 1, 1: 0 | Сдвиг влево на 1 бит |
| a & (1 << k) | 1 ← k=4 | Проверяем “единица ли в k-ом бите” |
| a \| (1 << k) | 1 ← k=4 | Выставляем в числе “единицу в k-ом бите” |
| **popcount32(a)** | | Узнаем число единичных бит |

Как посчитать сколько отличается бит между двумя числами A и B? (Hamming dist)

# Битовая арифметика

$45 = 32 + \cancel{16} + 8 + 4 + \cancel{2} + 1$

| | 32 | 16 | 8 | 4 | 2 | 1 | |
|---|---|---|---|---|---|---|---|
| $45_2 =$ | 1 | 0 | 1 | 1 | 0 | 1 | |
| $45_2 >> 1$ | 0 | 1 | 0 | 1 | 1 | 0 | Сдвиг вправо на 1 бит |
| $45_2 << 1$ | 0 | 1 | 1 | 0 | 1 | 0 | Сдвиг влево на 1 бит |

| | | |
|---|---|---|
| a & (1 << k) | 1 ← k=4 | Проверяем “единица ли в k-ом бите” |
| a \| (1 << k) | 1 ← k=4 | Выставляем в числе “единицу в k-ом бите” |
| **popcount32(a)** | | Узнаем число единичных бит |

Как посчитать сколько отличается бит между двумя числами A и B? (Hamming dist)

## Битовая арифметика

A ^ B (*xor*) = единички там где разное

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 45 = | 32 + | ~~16~~ + | 8 + | 4 + | ~~2~~ + | 1 | |
| $45_2$ = | 32: 1 | 16: 0 | 8: 1 | 4: 1 | 2: 0 | 1: 1 | |
| $45_2$ >> 1 | 32: 0 | 16: 1 | 8: 0 | 4: 1 | 2: 1 | 1: 0 | Сдвиг вправо на 1 бит |
| $45_2$ << 1 | 32: 0 | 16: 1 | 8: 1 | 4: 0 | 2: 1 | 1: 0 | Сдвиг влево на 1 бит |
| a & (1 << k) | 1 ← k=4 | | | | | | Проверяем “единица ли в k-ом бите” |
| a \| (1 << k) | 1 ← k=4 | | | | | | Выставляем в числе “единицу в k-ом бите” |
| **popcount32(a)** | | | | | | | Узнаем число единичных бит |

Как посчитать сколько отличается бит между двумя числами A и B? (Hamming dist)

## Битовая арифметика

A ^ B (*xor*) = единички там где разное

**popcount32**(A ^ B) = количество разных бит

45 = 32 + ~~16~~ + 8 + 4 + ~~2~~ + 1

| | 32 | 16 | 8 | 4 | 2 | 1 | |
|---|---|---|---|---|---|---|---|
| $45_2$ = | 1 | 0 | 1 | 1 | 0 | 1 | |
| $45_2$ >> 1 | 0 | 1 | 0 | 1 | 1 | 0 | Сдвиг вправо на 1 бит |
| $45_2$ << 1 | 0 | 1 | 1 | 0 | 1 | 0 | Сдвиг влево на 1 бит |

a & (1 << k) — 1 ← k=4 — Проверяем “единица ли в k-ом бите”

a | (1 << k) — 1 ← k=4 — Выставляем в числе “единицу в k-ом бите”

**popcount32(a)** — Узнаем число единичных бит

# Mr. 239

# Префиксная сумма: разложение числа

$$239 = 128 + 64 + 32 + \cancel{16} + 8 + 4 + 2 + 1$$

# Префиксная сумма: разложение числа

$$239 = 128 + 64 + 32 + \sout{16} + 8 + 4 + 2 + 1$$

$$239_2 = \overset{128}{1}\ \overset{64}{1}\ \overset{32}{1}\ \overset{16}{0}\ \overset{8}{1}\ \overset{4}{1}\ \overset{2}{1}\ \overset{1}{1}$$

# Префиксная сумма: разложение числа

$$239 = 128 + 64 + 32 + \cancel{16} + 8 + 4 + 2 + 1$$

| | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|
| $239_2 =$ | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |

# Префиксная сумма: разложение числа

$$239 = 128 + 64 + 32 + \cancel{16} + 8 + 4 + 2 + 1$$

$$239_2 = \overset{128}{1}\ \overset{64}{1}\ \overset{32}{1}\ \overset{16}{0}\ \overset{8}{1}\ \overset{4}{1}\ \overset{2}{1}\ \overset{1}{1}$$


Можно ли посчитать все префиксные суммы за один запуск одного кернела?

# Префиксная сумма: разложение числа

$$239 = 128 + 64 + 32 + \cancel{16} + 8 + 4 + 2 + 1$$

$$239_2 = \overset{128}{1}\ \overset{64}{1}\ \overset{32}{1}\ \overset{16}{0}\ \overset{8}{1}\ \overset{4}{1}\ \overset{2}{1}\ \overset{1}{1}$$


Что **предподсчитать** чтобы найти префиксную сумму?

# Префиксная сумма: разложение числа

$$239 = 128 + 64 + 32 + \cancel{16} + 8 + 4 + 2 + 1$$

$$239_2 = \overset{128}{1}\ \overset{64}{1}\ \overset{32}{1}\ \overset{16}{0}\ \overset{8}{1}\ \overset{4}{1}\ \overset{2}{1}\ \overset{1}{1}$$


Что **предподсчитать** чтобы найти префиксную сумму?

А любое разложение не подойдет?

# Префиксная сумма: разложение числа

$$239 = 128 + 64 + 32 + \cancel{16} + 8 + 4 + 2 + 1$$

$$239_2 = \overset{128}{1}\ \overset{64}{1}\ \overset{32}{1}\ \overset{16}{0}\ \overset{8}{1}\ \overset{4}{1}\ \overset{2}{1}\ \overset{1}{1}$$


Что **предподсчитать** чтобы найти префиксную сумму?

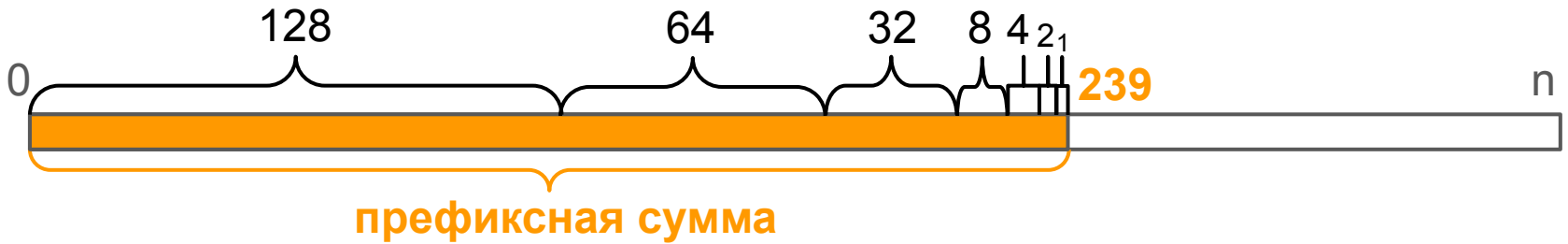
А любое разложение не подойдет?

Нам обязательно нужно чтобы **предподсчет** ложился на массовый параллелизм!

# Префиксная сумма: разложение числа

$$239 = 128 + 64 + 32 + \cancel{16} + 8 + 4 + 2 + 1$$

$$239_2 = \overset{128}{1}\ \overset{64}{1}\ \overset{32}{1}\ \overset{16}{0}\ \overset{8}{1}\ \overset{4}{1}\ \overset{2}{1}\ \overset{1}{1}$$




Что **предподсчитать** чтобы найти префиксную сумму?

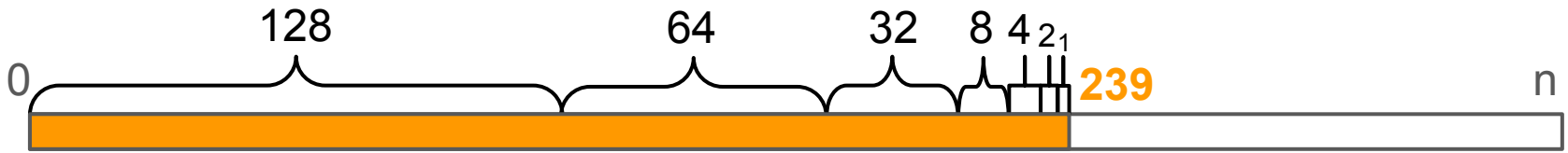
А любое разложение не подойдет?

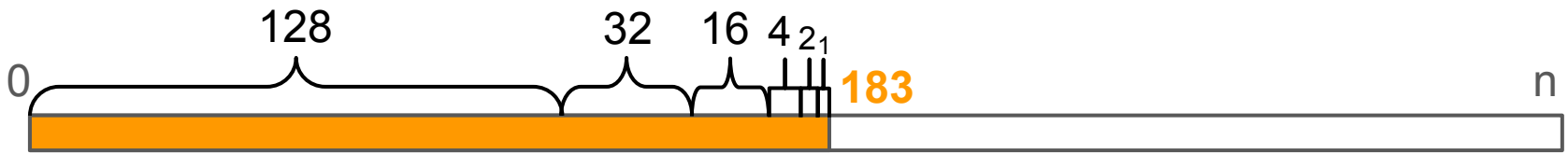
Нам обязательно нужно чтобы **предподсчет** ложился на массовый параллелизм!

# Префиксная сумма: разложение числа

$$239 = 128 + 64 + 32 + \sout{16} + 8 + 4 + 2 + 1$$

$$239_2 = \overset{128}{1}\ \overset{64}{1}\ \overset{32}{1}\ \overset{16}{0}\ \overset{8}{1}\ \overset{4}{1}\ \overset{2}{1}\ \overset{1}{1}$$



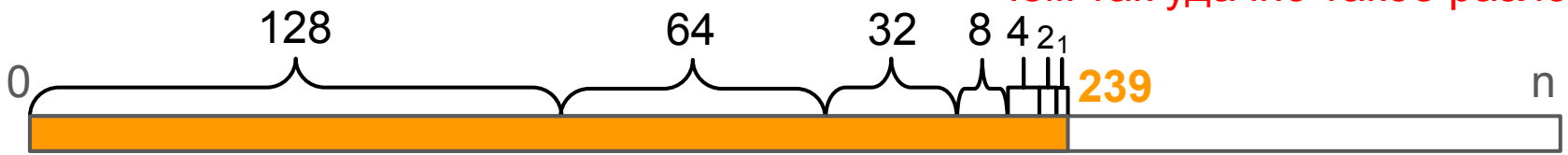

$$183 = 128 + \sout{64} + 32 + 16 + \sout{8} + 4 + 2 + 1$$

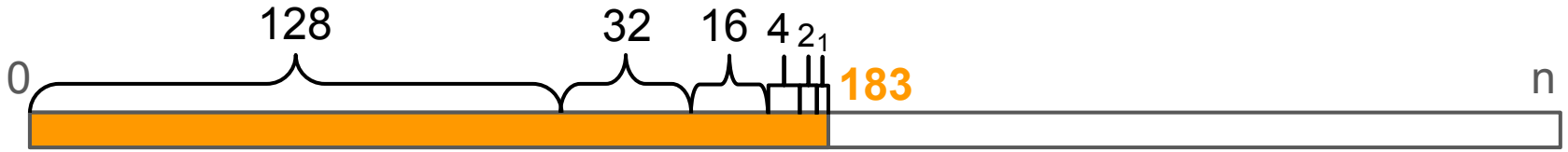
# Префиксная сумма: разложение числа

$$239 = 128 + 64 + 32 + \sout{16} + 8 + 4 + 2 + 1$$

$$239_2 = \overset{128}{1}\ \overset{64}{1}\ \overset{32}{1}\ \overset{16}{0}\ \overset{8}{1}\ \overset{4}{1}\ \overset{2}{1}\ \overset{1}{1}$$

Чем так удачно такое разложение?



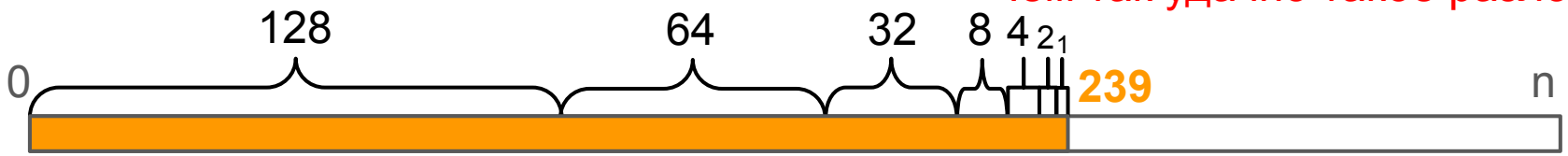

$$183 = 128 + \sout{64} + 32 + 16 + \sout{8} + 4 + 2 + 1$$

# Префиксная сумма: разложение числа

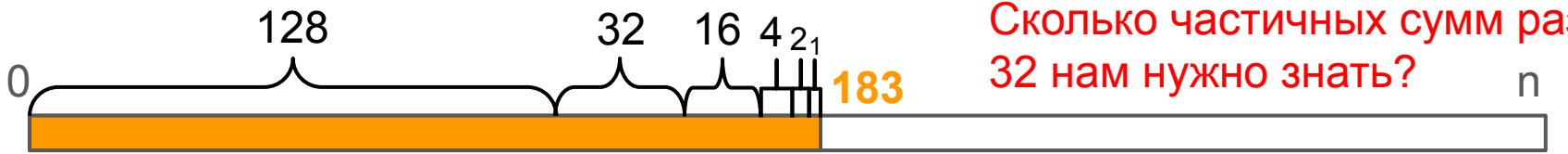
$$239 = 128 + 64 + 32 + \sout{16} + 8 + 4 + 2 + 1$$

| | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|
| $239_2 =$ | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |

Чем так удачно такое разложение?




Сколько частичных сумм размера 32 нам нужно знать?



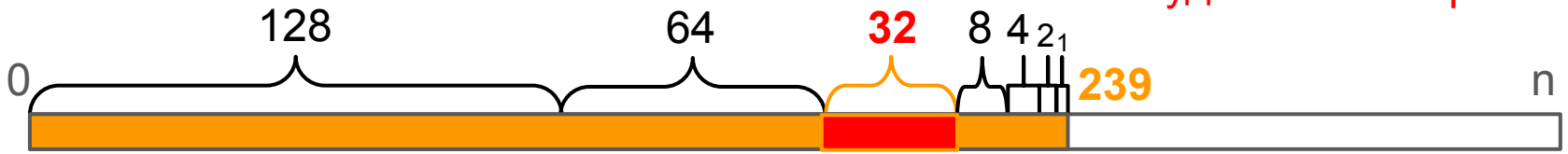

$$183 = 128 + \sout{64} + 32 + 16 + \sout{8} + 4 + 2 + 1$$

# Префиксная сумма: разложение числа

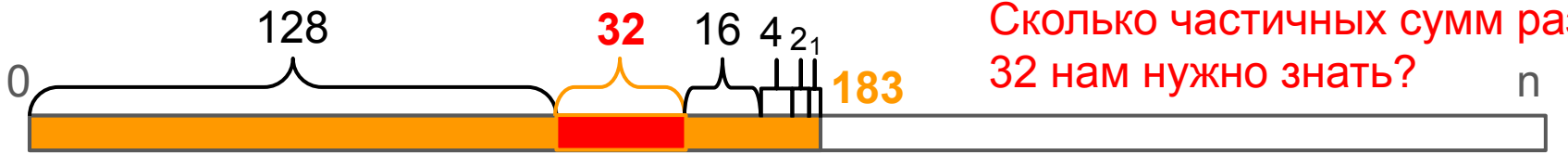
$239 = 128 + 64 + 32 + \sout{16} + 8 + 4 + 2 + 1$

| | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|
| $239_2 =$ | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |

Чем так удачно такое разложение?




Сколько частичных сумм размера 32 нам нужно знать?



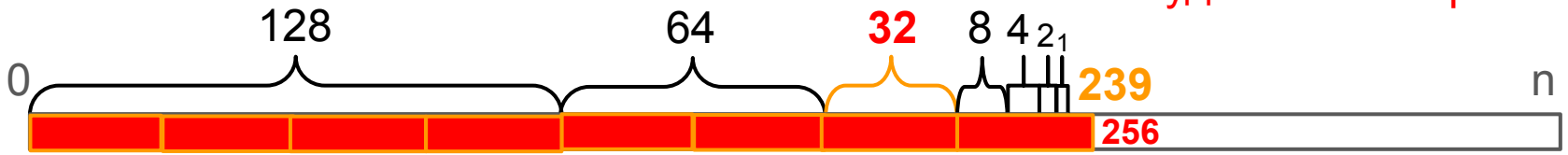

$183 = 128 + \sout{64} + 32 + 16 + \sout{8} + 4 + 2 + 1$

# Префиксная сумма: разложение числа

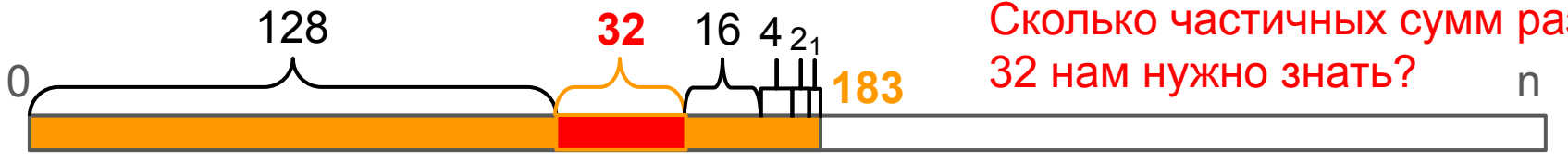
$239 = 128 + 64 + 32 + \sout{16} + 8 + 4 + 2 + 1$

| | 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|---|
| $239_2 =$ | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |

Чем так удачно такое разложение?




Сколько частичных сумм размера 32 нам нужно знать?

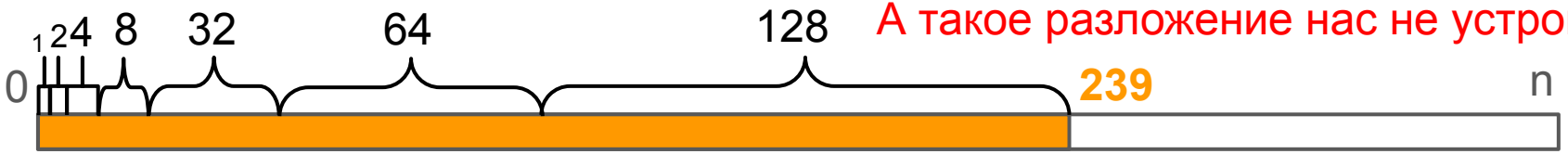



$183 = 128 + \sout{64} + 32 + 16 + \sout{8} + 4 + 2 + 1$

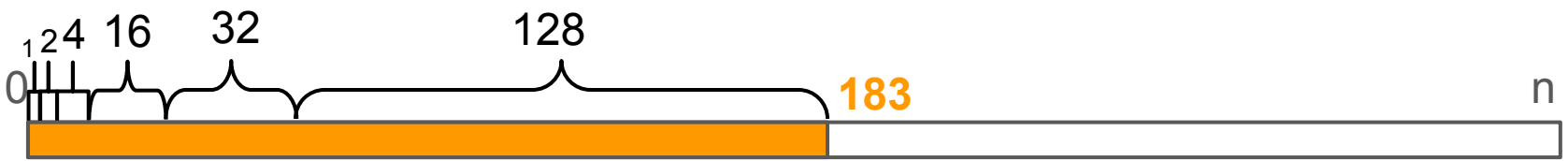
# Префиксная сумма: разложение числа

$$239 = 128 + 64 + 32 + \sout{16} + 8 + 4 + 2 + 1$$

$$239_2 = \overset{128}{1}\ \overset{64}{1}\ \overset{32}{1}\ \overset{16}{0}\ \overset{8}{1}\ \overset{4}{1}\ \overset{2}{1}\ \overset{1}{1}$$




А такое разложение нас не устроит?

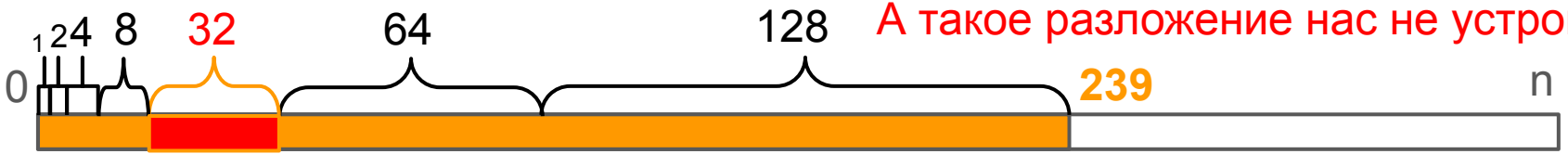



$$183 = 128 + \sout{64} + 32 + 16 + \sout{8} + 4 + 2 + 1$$

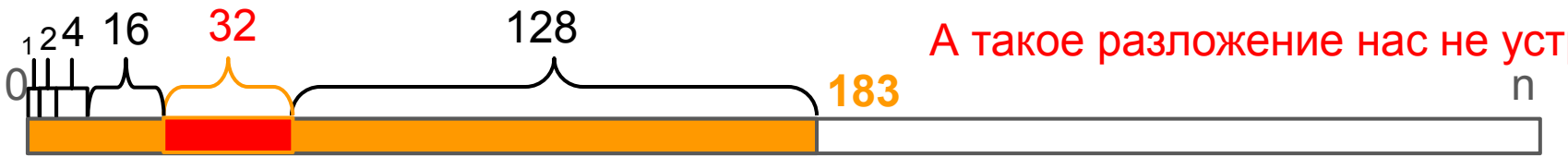
# Префиксная сумма: разложение числа

$239 = 128 + 64 + 32 + \sout{16} + 8 + 4 + 2 + 1$

$239_2 = \overset{128}{1}\ \overset{64}{1}\ \overset{32}{1}\ \overset{16}{0}\ \overset{8}{1}\ \overset{4}{1}\ \overset{2}{1}\ \overset{1}{1}$




А такое разложение нас не устроит?



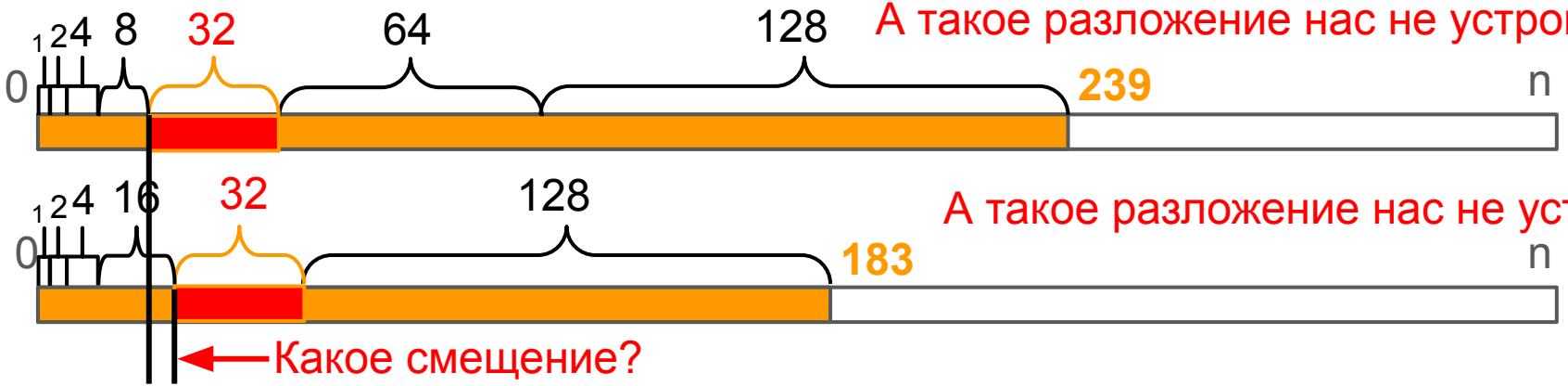

А такое разложение нас не устроит?

$183 = 128 + \sout{64} + 32 + 16 + \sout{8} + 4 + 2 + 1$

# Префиксная сумма: разложение числа

$$239 = 128 + 64 + 32 + \sout{16} + 8 + 4 + 2 + 1$$

$$239_2 = \overset{128}{1}\ \overset{64}{1}\ \overset{32}{1}\ \overset{16}{0}\ \overset{8}{1}\ \overset{4}{1}\ \overset{2}{1}\ \overset{1}{1}$$

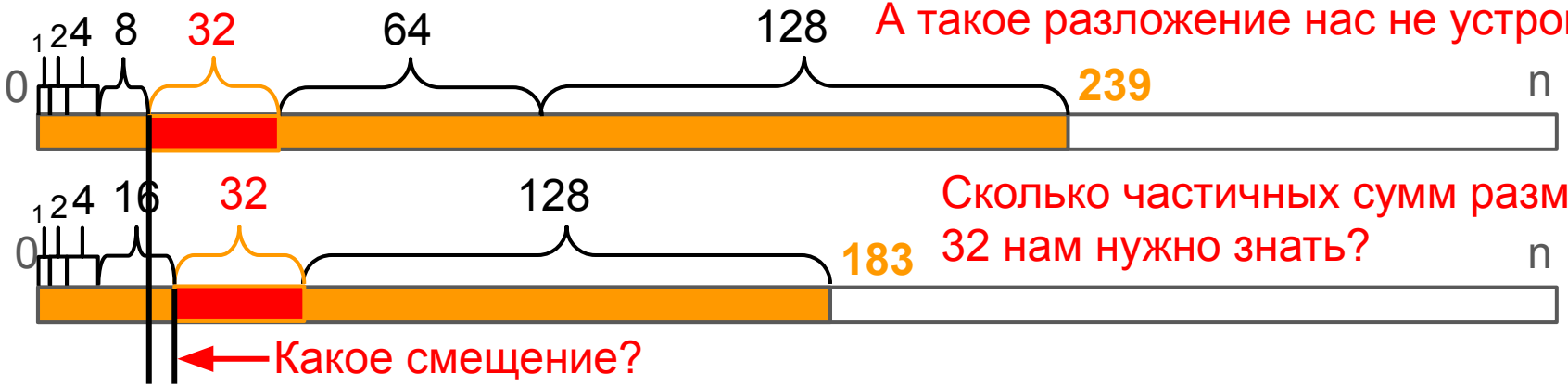



$$183 = 128 + \sout{64} + 32 + 16 + \sout{8} + 4 + 2 + 1$$

# Префиксная сумма: разложение числа

$$239 = 128 + 64 + 32 + \sout{16} + 8 + 4 + 2 + 1$$

$$239_2 = \overset{128}{1}\ \overset{64}{1}\ \overset{32}{1}\ \overset{16}{0}\ \overset{8}{1}\ \overset{4}{1}\ \overset{2}{1}\ \overset{1}{1}$$



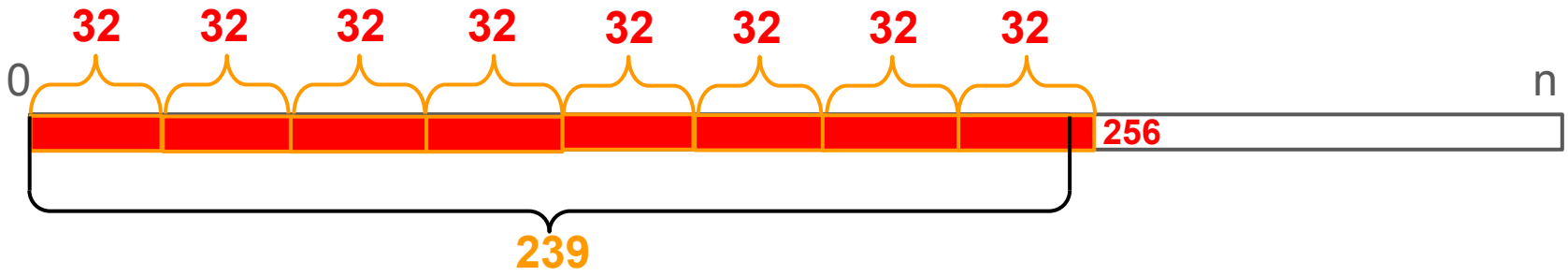

$$183 = 128 + \sout{64} + 32 + 16 + \sout{8} + 4 + 2 + 1$$

# Префиксная сумма: есть идеи?

# Префиксная сумма: **сборка**

$$239 = 128 + 64 + 32 + \cancel{16} + 8 + 4 + 2 + 1$$

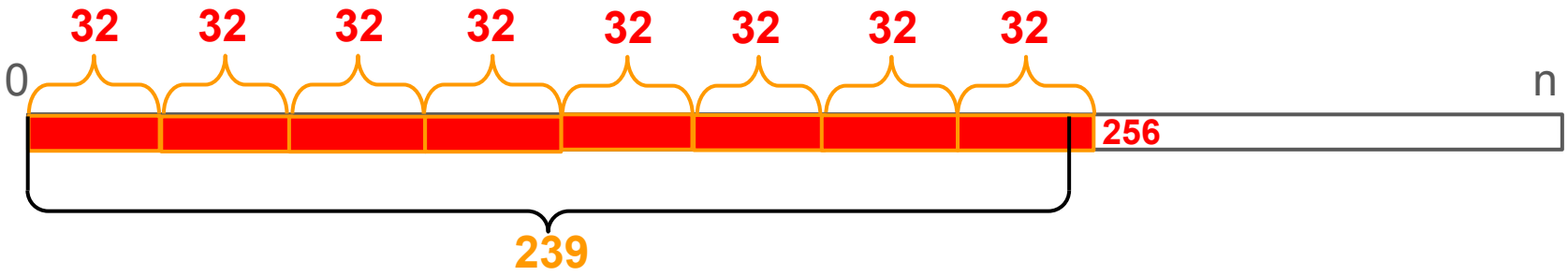
$$239_2 = \underset{128}{1}\ \underset{64}{1}\ \underset{32}{1}\ \underset{16}{0}\ \underset{8}{1}\ \underset{4}{1}\ \underset{2}{1}\ \underset{1}{1}$$



# Префиксная сумма: **сборка**

$$239 = 128 + 64 + 32 + \sout{16} + 8 + 4 + 2 + 1$$

$$239_2 = \overset{128}{1}\ \overset{64}{1}\ \overset{32}{1}\ \overset{16}{0}\ \overset{8}{1}\ \overset{4}{1}\ \overset{2}{1}\ \overset{1}{1}$$

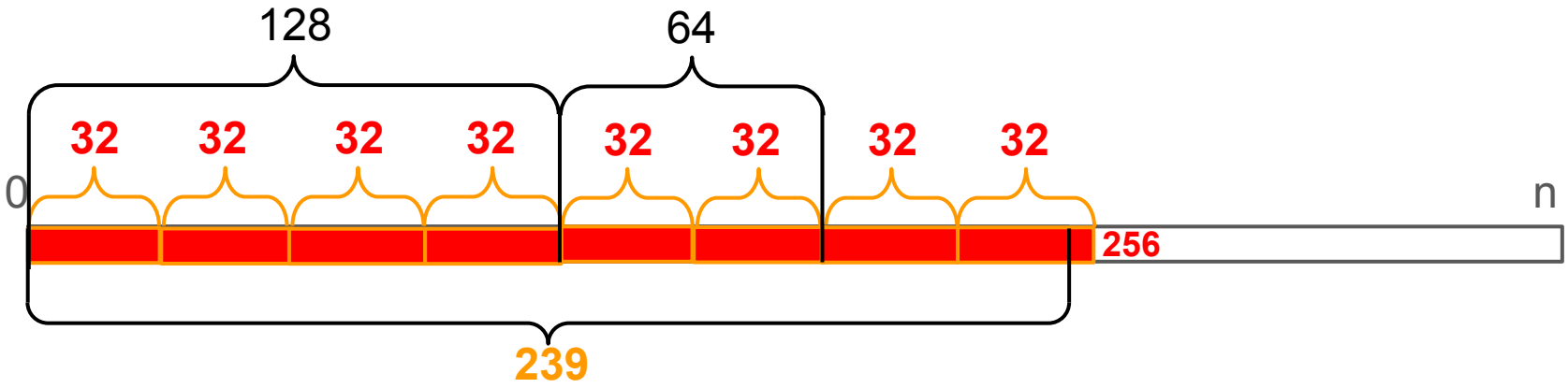



Из чего составить ответ для WorkItem на позиции 239?

# Префиксная сумма: **сборка**

$$239 = 128 + 64 + 32 + \cancel{16} + 8 + 4 + 2 + 1$$

$$239_2 = \overset{128}{1}\ \overset{64}{1}\ \overset{32}{1}\ \overset{16}{0}\ \overset{8}{1}\ \overset{4}{1}\ \overset{2}{1}\ \overset{1}{1}$$

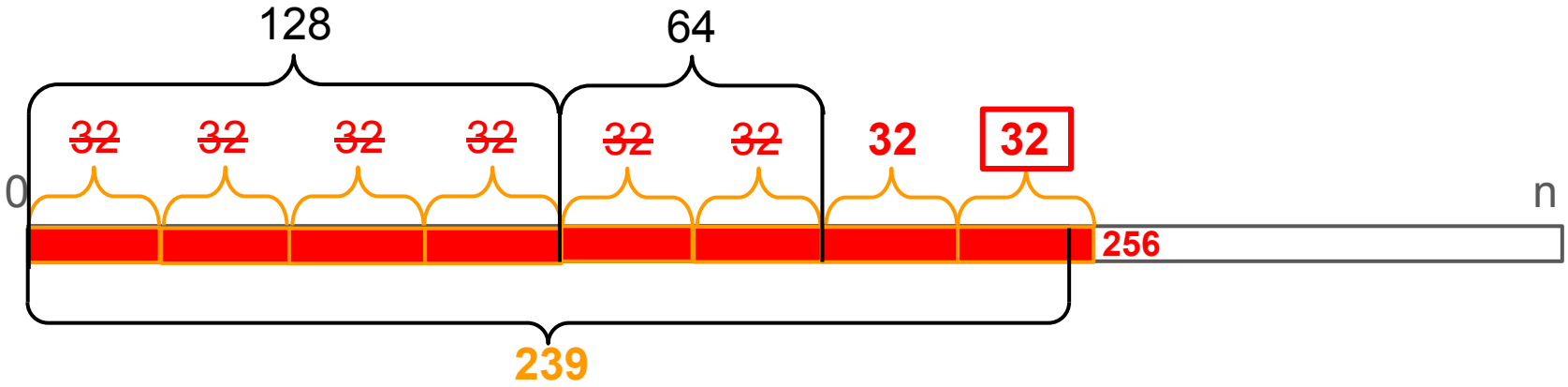



Из чего составить ответ для WorkItem на позиции 239?

# Префиксная сумма: **сборка**

$$239 = 128 + 64 + 32 + \cancel{16} + 8 + 4 + 2 + 1$$

$$239_2 = \underset{128}{1}\ \underset{64}{1}\ \underset{32}{1}\ \underset{16}{0}\ \underset{8}{1}\ \underset{4}{1}\ \underset{2}{1}\ \underset{1}{1}$$

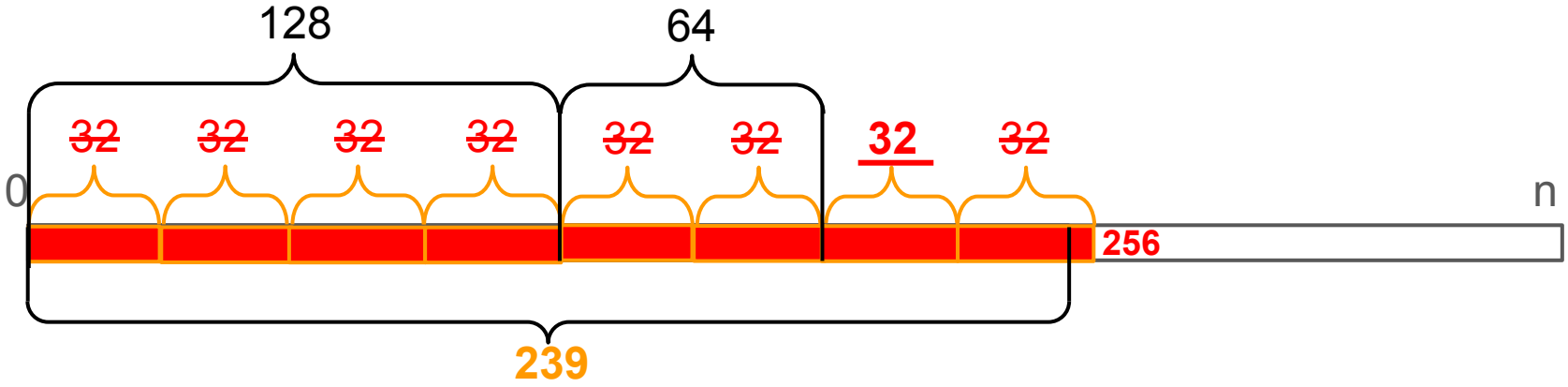



Из чего составить ответ для WorkItem на позиции 239?

# Префиксная сумма: **сборка**

$$239 = 128 + 64 + 32 + \cancel{16} + 8 + 4 + 2 + 1$$

$$239_2 = \underset{128}{1}\ \underset{64}{1}\ \underset{32}{1}\ \underset{16}{0}\ \underset{8}{1}\ \underset{4}{1}\ \underset{2}{1}\ \underset{1}{1}$$

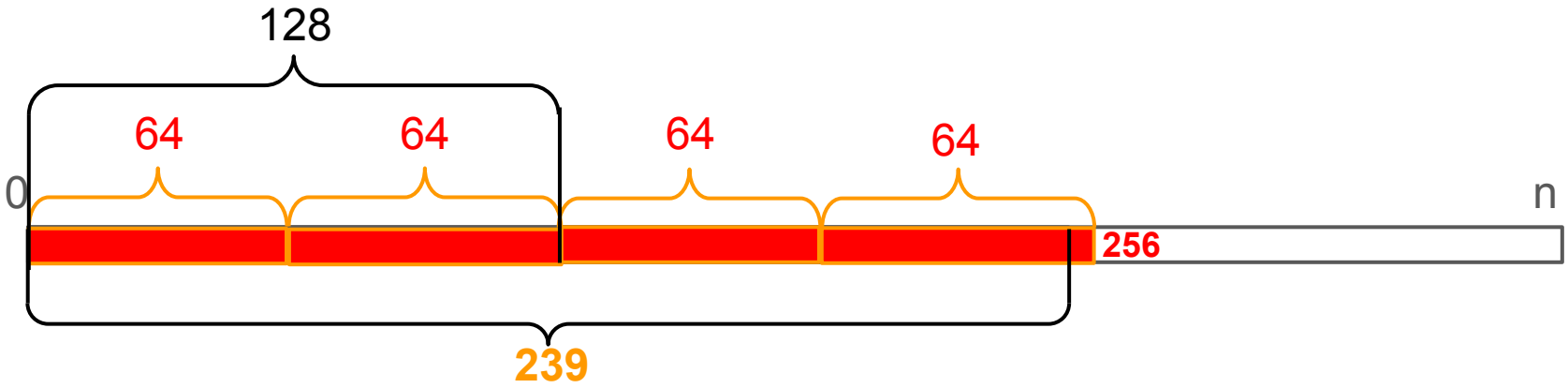



Из чего составить ответ для WorkItem на позиции 239?

# Префиксная сумма: **сборка**

$$239 = 128 + 64 + 32 + \sout{16} + 8 + 4 + 2 + 1$$

$$239_2 = \overset{128}{1}\ \overset{64}{1}\ \overset{32}{1}\ \overset{16}{0}\ \overset{8}{1}\ \overset{4}{1}\ \overset{2}{1}\ \overset{1}{1}$$

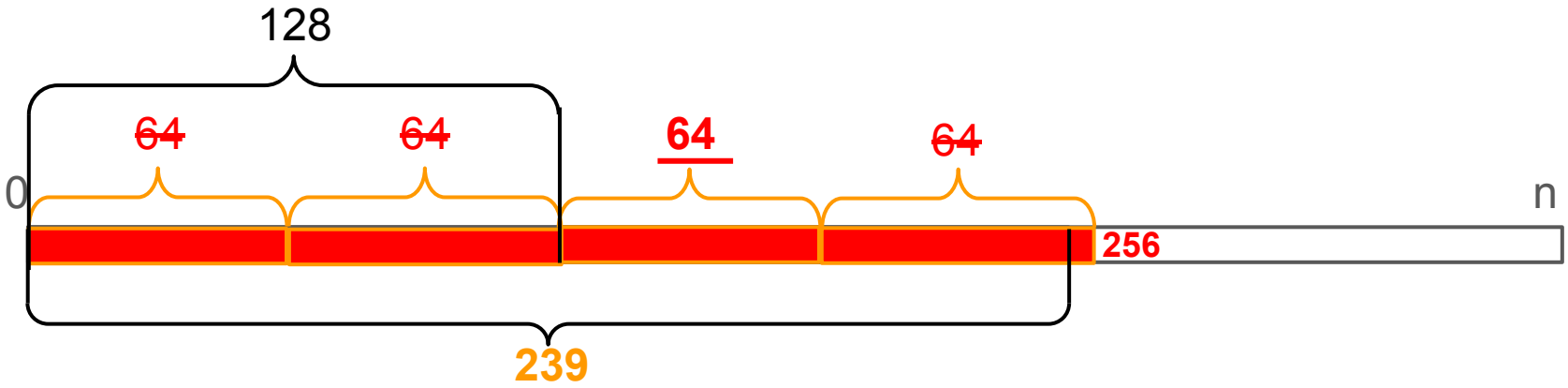



Из чего составить ответ для WorkItem на позиции 239?

# Префиксная сумма: **сборка**

$$239 = 128 + 64 + 32 + \sout{16} + 8 + 4 + 2 + 1$$

$$239_2 = \underset{128}{1}\ \underset{64}{1}\ \underset{32}{1}\ \underset{16}{0}\ \underset{8}{1}\ \underset{4}{1}\ \underset{2}{1}\ \underset{1}{1}$$




Из чего составить ответ для WorkItem на позиции 239?

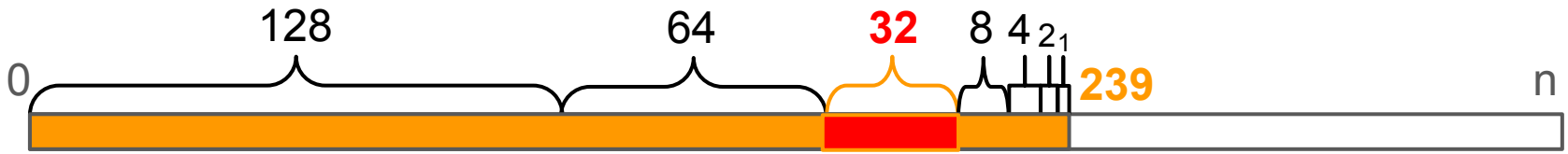
# Префиксная сумма: **есть идеи?**

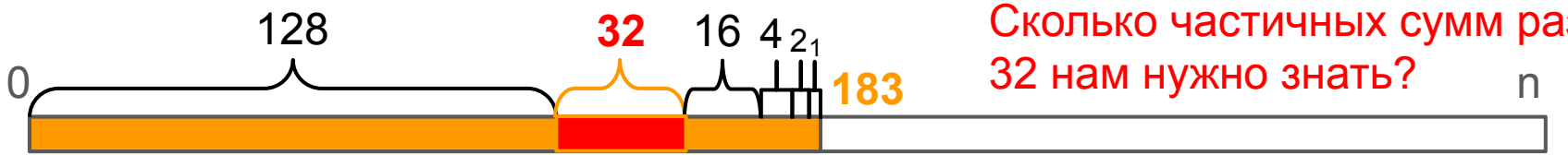
Нам нужно массовым параллелизмом сразу для всех префиксов!
Сразу для всех WorkItems!

# Префиксная сумма: **частичные суммы**

$239 = 128 + 64 + 32 + \sout{16} + 8 + 4 + 2 + 1$

$239_2 =$ 128: 1, 64: 1, 32: 1, 16: 0, 8: 1, 4: 1, 2: 1, 1: 1




Сколько частичных сумм размера 32 нам нужно знать?

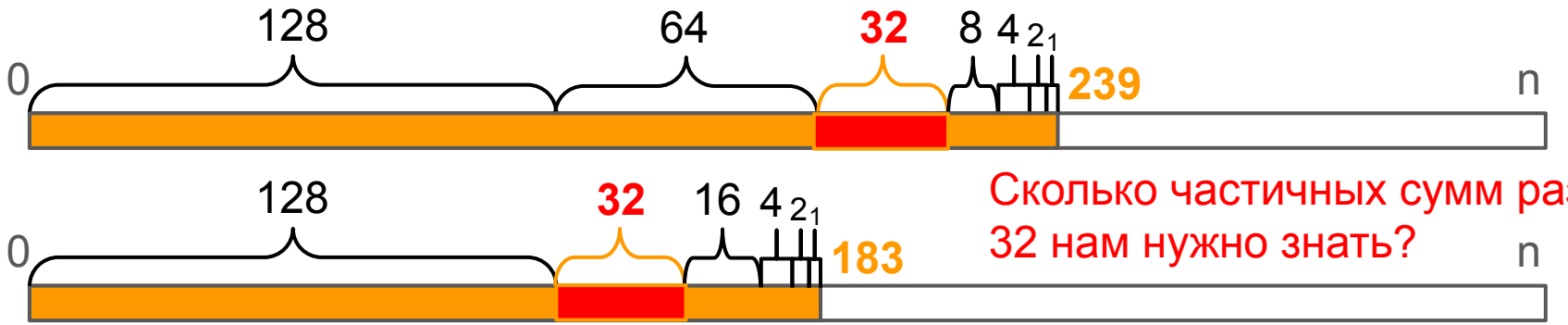
$183 = 128 + \sout{64} + 32 + 16 + \sout{8} + 4 + 2 + 1$

# Префиксная сумма: **частичные суммы**

0) Входной массив - суммы размера 1 *(n штук)*

..) …

5) **Reduction** чтобы построить суммы размера 32 ***(сколько штук?)***



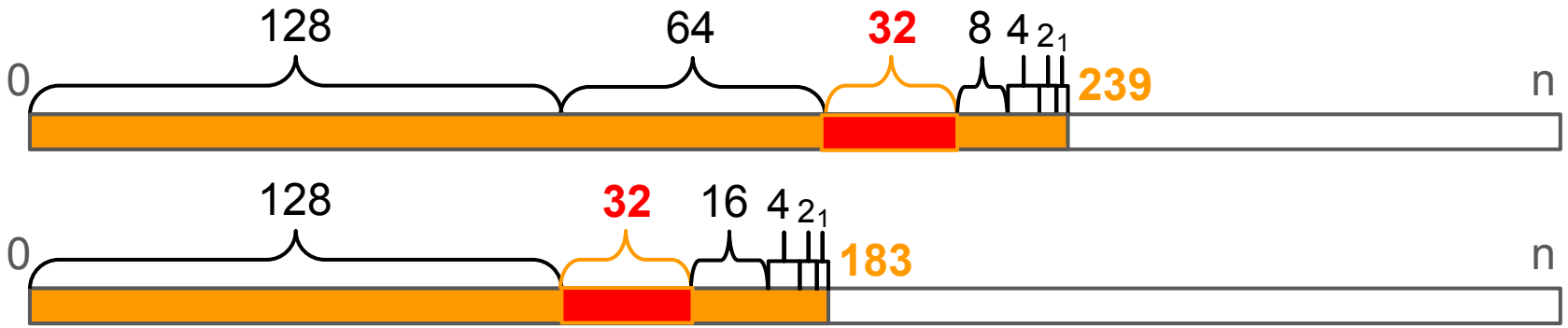

183 = 128 + ~~64~~ + 32 + 16 + ~~8~~ + 4 + 2 + 1

# Префиксная сумма: **частичные суммы**

0) Входной массив - суммы размера 1 *(n штук)*

..) …

5) **Reduction** чтобы построить суммы размера 32 ***на каждой 32-ой позиции (n/32)***




183 = 128 + ~~64~~ + 32 + 16 + ~~8~~ + 4 + 2 + 1

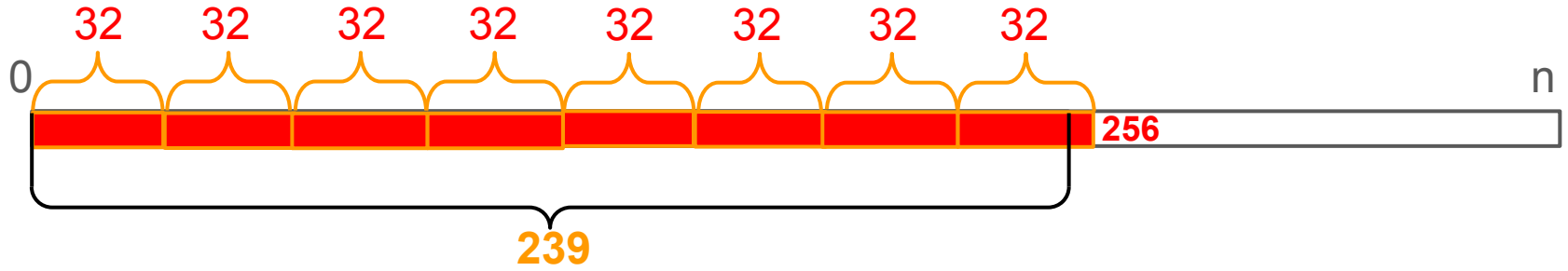
# Префиксная сумма: **частичные суммы**

0) Входной массив - суммы размера 1 *(n штук)*

..) …

5) **Reduction** чтобы построить суммы размера 32 ***на каждой 32-ой позиции (n/32)***

**Какая из этих частичных сумм пригодится для префиксной суммы 239?**



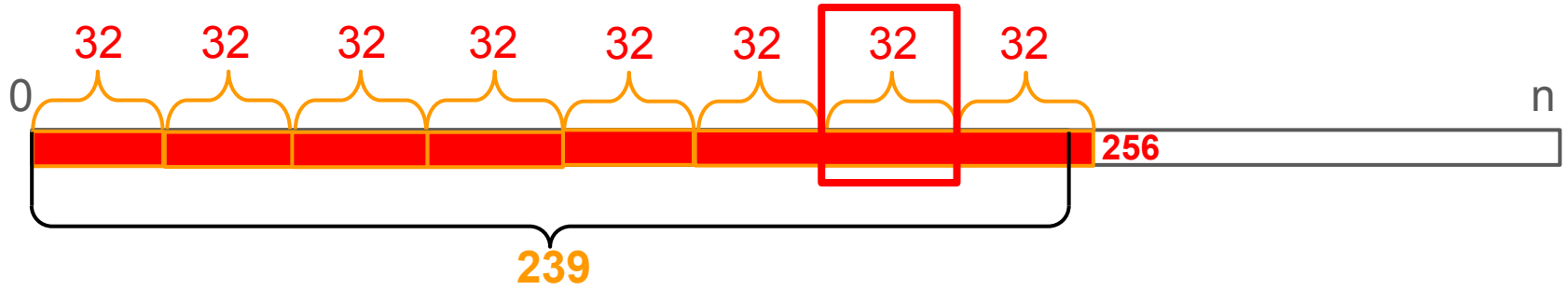
# Префиксная сумма: **частичные суммы**

0) Входной массив - суммы размера 1 *(n штук)*

..) …

5) **Reduction** чтобы построить суммы размера 32 ***на каждой 32-ой позиции (n/32)***

**Какая из этих частичных сумм пригодится для префиксной суммы 239?**



# Префиксная сумма: **частичные суммы**

0) Входной массив - суммы размера 1 *(n штук)*

..) **Как обобщить шаги?**

5) **Reduction** чтобы построить суммы размера 32 на каждой 32-ой позиции *(n/32)*

# Префиксная сумма: **частичные суммы**

0) Входной массив - суммы размера 1 *(n штук)*

1) **Reduction** чтобы построить суммы размера 2 *(сколько штук?)*

…

5) **Reduction** чтобы построить суммы размера 32 на каждой 32-ой позиции *(n/32)*

# Префиксная сумма: **частичные суммы**

0) Входной массив - суммы размера 1 *(n штук)*
1) **Reduction** чтобы построить суммы размера 2 на четных позициях *(n/2 штук)*
…
5) **Reduction** чтобы построить суммы размера 32 на каждой 32-ой позиции *(n/32)*

# Префиксная сумма: **частичные суммы**

0) Входной массив - суммы размера 1 *(n штук)*

1) **Reduction** чтобы построить суммы размера 2 на четных позициях *(n/2 штук)*

**k**) Как выглядит **k**-ый шаг?

5) **Reduction** чтобы построить суммы размера 32 на каждой 32-ой позиции *(n/32)*

# Префиксная сумма: **частичные суммы**

0) Входной массив - суммы размера 1 *(n штук)*
1) **Reduction** чтобы построить суммы размера 2 на четных позициях *(n/2 штук)*
**k**) **Reduction** чтобы построить суммы размера **2^k** на каждой **2^k** позиции ***(n/k)***
5) **Reduction** чтобы построить суммы размера 32 на каждой 32-ой позиции *(n/32)*

$239_2 =$ 128: 1, 64: 1, 32: 1, 16: 0, 8: 1, 4: 1, 2: 1, 1: 1

# Префиксная сумма: **частичные суммы + сборка**

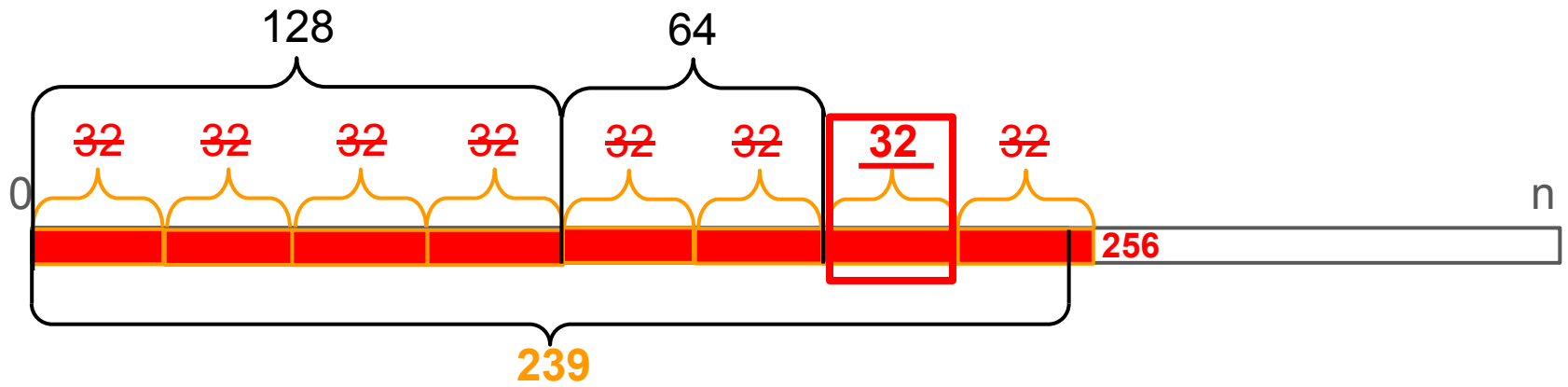
0) Входной массив - суммы размера 1 *(n штук)*

1) **Reduction** чтобы построить суммы размера 2 на четных позициях *(n/2 штук)*

**k**) **Reduction** чтобы построить суммы размера **2^k** на каждой **2^k** позиции ***(n/k)***

5) **Reduction** чтобы построить суммы размера 32 на каждой 32-ой позиции *(n/32)*

**Как массово для всех WorkItem собрать ответ из частичных сумм?**



$239_2$ = 128: 1, 64: 1, 32: 1, 16: 0, 8: 1, 4: 1, 2: 1, 1: 1

# Префиксная сумма: **частичные суммы + сборка**

0) Входной массив - суммы размера 1 *(n штук)*

1) **Reduction** чтобы построить суммы размера 2 на четных позициях *(n/2 штук)*

**k**) **Reduction** чтобы построить суммы размера **2^k** на каждой **2^k** позиции ***(n/k)***

5) **Reduction** чтобы построить суммы размера 32 на каждой 32-ой позиции *(n/32)*

**Пусть мы WorkItem=239, как понять, нужно ли добавить себе сумму размера 1?**
**Сколько штук и на каких позициях?**

$239_2$ = 128: 1, 64: 1, 32: 1, 16: 0, 8: 1, 4: 1, 2: 1, 1: 1

# Префиксная сумма: **частичные суммы + сборка**

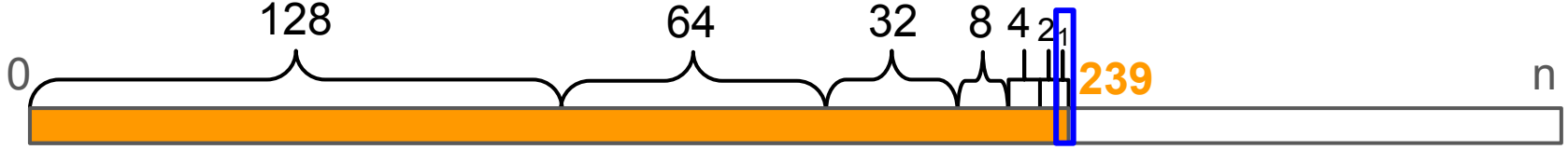
0) Входной массив - суммы размера 1 *(n штук)*

1) **Reduction** чтобы построить суммы размера 2 на четных позициях *(n/2 штук)*

**k**) **Reduction** чтобы построить суммы размера **2^k** на каждой **2^k** позиции ***(n/k)***

5) **Reduction** чтобы построить суммы размера 32 на каждой 32-ой позиции *(n/32)*

**Пусть мы WorkItem=239, как понять, нужно ли добавить себе сумму размера 1?**
**Сколько штук и на каких позициях?**



$239_2$ =

| 128 | 64 | 32 | 16 | 8 | 4 | 2 | 1 |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |

# Префиксная сумма: **частичные суммы + сборка**

0) Входной массив - суммы размера 1 *(n штук)*

1) **Reduction** чтобы построить суммы размера 2 на четных позициях *(n/2 штук)*

**k**) **Reduction** чтобы построить суммы размера **2^k** на каждой **2^k** позиции ***(n/k)***

5) **Reduction** чтобы построить суммы размера 32 на каждой 32-ой позиции *(n/32)*

**Пусть мы WorkItem=239, как понять, нужно ли добавить себе сумму размера 1?**
**Сколько штук и на каких позициях?**

$239_2$ = 128: 1, 64: 1, 32: 1, 16: 0, 8: 1, 4: 1, 2: 1, 1: 1

# Префиксная сумма: **частичные суммы + сборка**

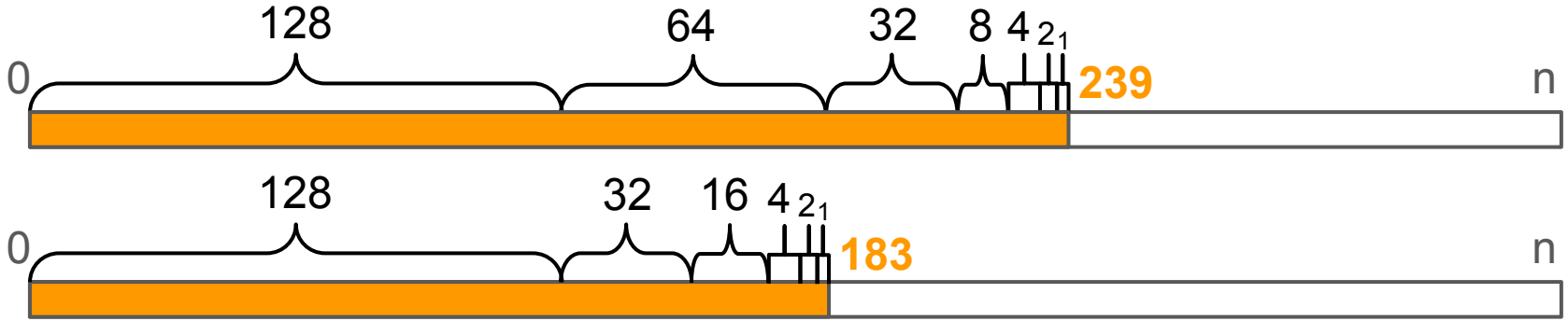
0) Входной массив - суммы размера 1 *(n штук)*

1) **Reduction** чтобы построить суммы размера 2 на четных позициях *(n/2 штук)*

**k**) **Reduction** чтобы построить суммы размера **2^k** на каждой **2^k** позиции ***(n/k)***

5) **Reduction** чтобы построить суммы размера 32 на каждой 32-ой позиции *(n/32)*

**Пусть мы WorkItem=183, как понять, нужно ли добавить себе сумму размера 1?**
**Сколько штук и на каких позициях?**



$239_2$ = 128: 1, 64: 1, 32: 1, 16: 0, 8: 1, 4: 1, 2: 1, 1: 1

# Префиксная сумма: **частичные суммы + сборка**

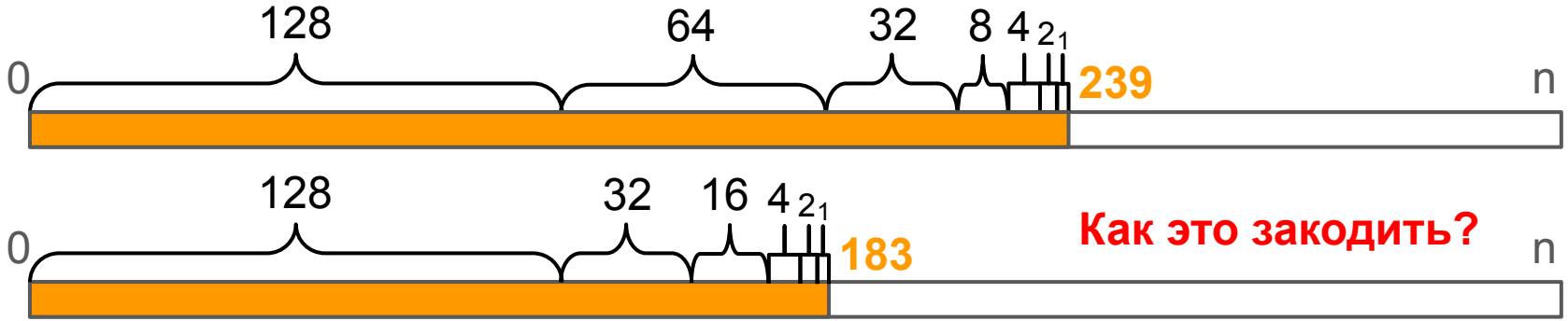
0) Входной массив - суммы размера 1 *(n штук)*

1) **Reduction** чтобы построить суммы размера 2 на четных позициях *(n/2 штук)*

**k**) **Reduction** чтобы построить суммы размера **2^k** на каждой **2^k** позиции ***(n/k)***

5) **Reduction** чтобы построить суммы размера 32 на каждой 32-ой позиции *(n/32)*

**Пусть мы WorkItem=183, как понять, нужно ли добавить себе сумму размера 1?**
**Сколько штук и на каких позициях?**



$239_2$ = 128: 1, 64: 1, 32: 1, 16: 0, 8: 1, 4: 1, 2: 1, 1: 1

# Префиксная сумма: **частичные суммы + сборка**

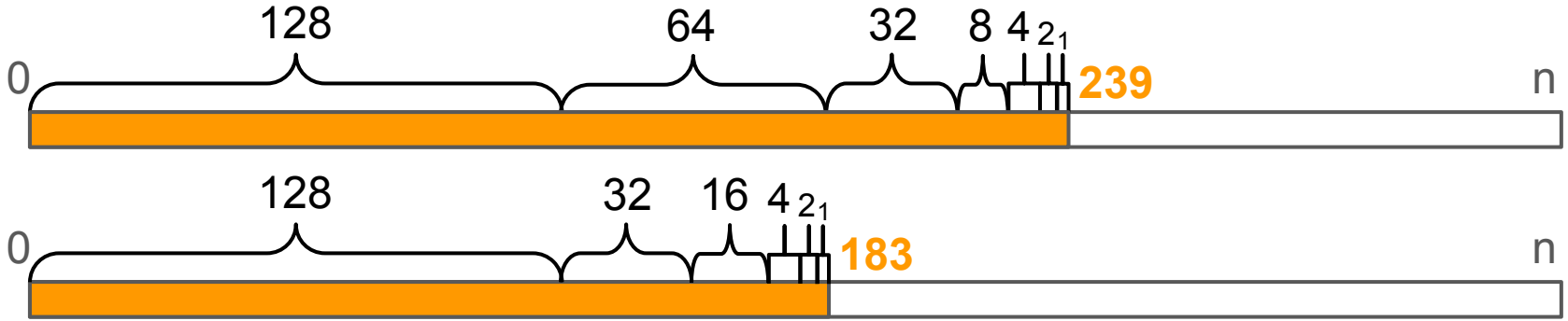
0) Входной массив - суммы размера 1 *(n штук)*

1) **Reduction** чтобы построить суммы размера 2 на четных позициях *(n/2 штук)*

**k**) **Reduction** чтобы построить суммы размера **2^k** на каждой **2^k** позиции ***(n/k)***

5) **Reduction** чтобы построить суммы размера 32 на каждой 32-ой позиции *(n/32)*

**Пусть мы WorkItem, как понять, нужно ли добавить себе сумму размера 2?**
**Одну или ноль штук на какой позиции?**



$239_2$ = 128: 1, 64: 1, 32: 1, 16: 0, 8: 1, 4: 1, 2: 1, 1: 1

# Префиксная сумма: **частичные суммы + сборка**

0) Входной массив - суммы размера 1 *(n штук)*

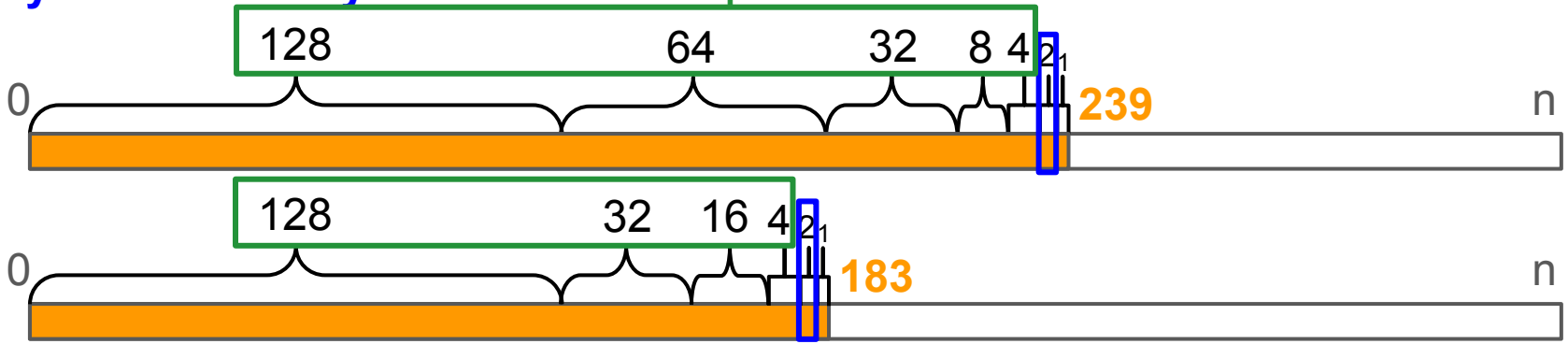
1) **Reduction** чтобы построить суммы размера 2 на четных позициях *(n/2 штук)*

**k**) **Reduction** чтобы построить суммы размера **2^k** на каждой **2^k** позиции ***(n/k)***

5) **Reduction** чтобы построить суммы размера 32 на каждой 32-ой позиции *(n/32)*

**Пусть мы WorkItem, как понять, нужно ли добавить себе сумму размера 2?**

**Одну или ноль штук на какой позиции?**



$239_2 =$ 128: 1, 64: 1, 32: 1, 16: 0, 8: 1, 4: 1, 2: 1, 1: 1

# Префиксная сумма: **частичные суммы + сборка**

0) Входной массив - суммы размера 1 *(n штук)*

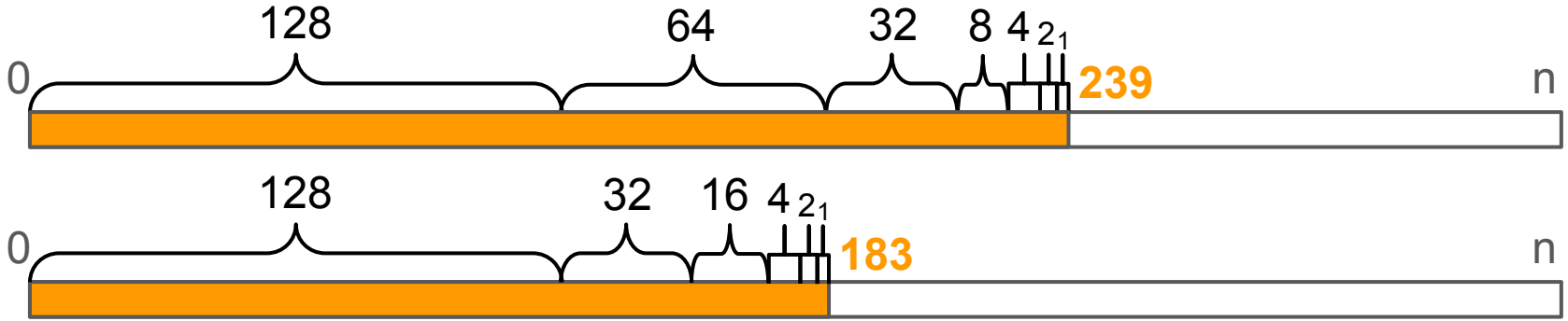
1) **Reduction** чтобы построить суммы размера 2 на четных позициях *(n/2 штук)*

**k**) **Reduction** чтобы построить суммы размера **2^k** на каждой **2^k** позиции ***(n/k)***

5) **Reduction** чтобы построить суммы размера 32 на каждой 32-ой позиции *(n/32)*

Пусть мы WorkItem, как понять, нужно ли добавить себе **сумму размера 16?**

**Одну или ноль штук на какой позиции?**



$239_2$ = 128: 1, 64: 1, 32: 1, 16: 0, 8: 1, 4: 1, 2: 1, 1: 1

# Префиксная сумма: **частичные суммы + сборка**

0) Входной массив - суммы размера 1 *(n штук)*

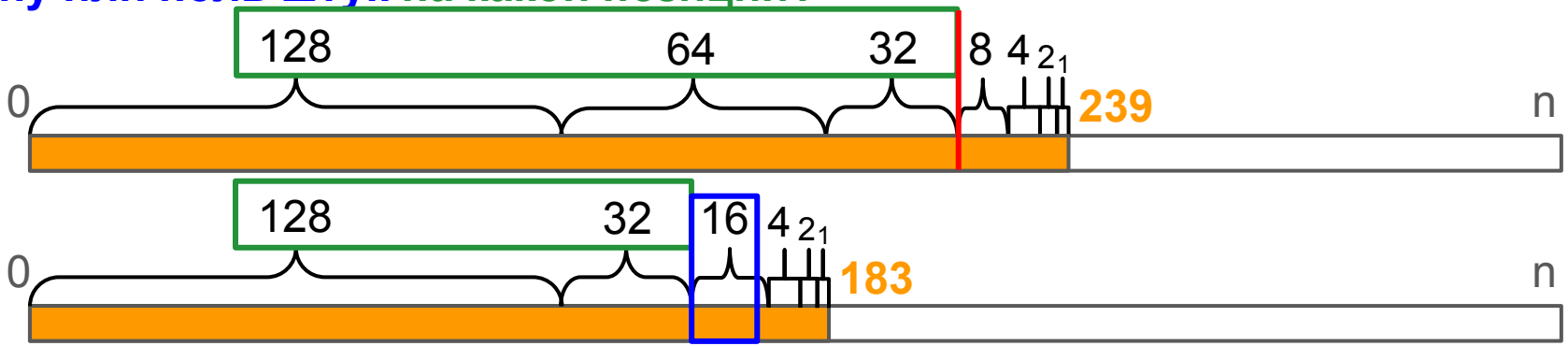
1) **Reduction** чтобы построить суммы размера 2 на четных позициях *(n/2 штук)*

**k**) **Reduction** чтобы построить суммы размера **2^k** на каждой **2^k** позиции ***(n/k)***

5) **Reduction** чтобы построить суммы размера 32 на каждой 32-ой позиции *(n/32)*

Пусть мы WorkItem, как понять, нужно ли добавить себе **сумму размера 16?**

**Одну или ноль штук на какой позиции?**



$239_2$ = 128: 1, 64: 1, 32: 1, 16: 0, 8: 1, 4: 1, 2: 1, 1: 1

# Префиксная сумма: **частичные суммы + сборка**

0) Входной массив - суммы размера 1 *(n штук)*

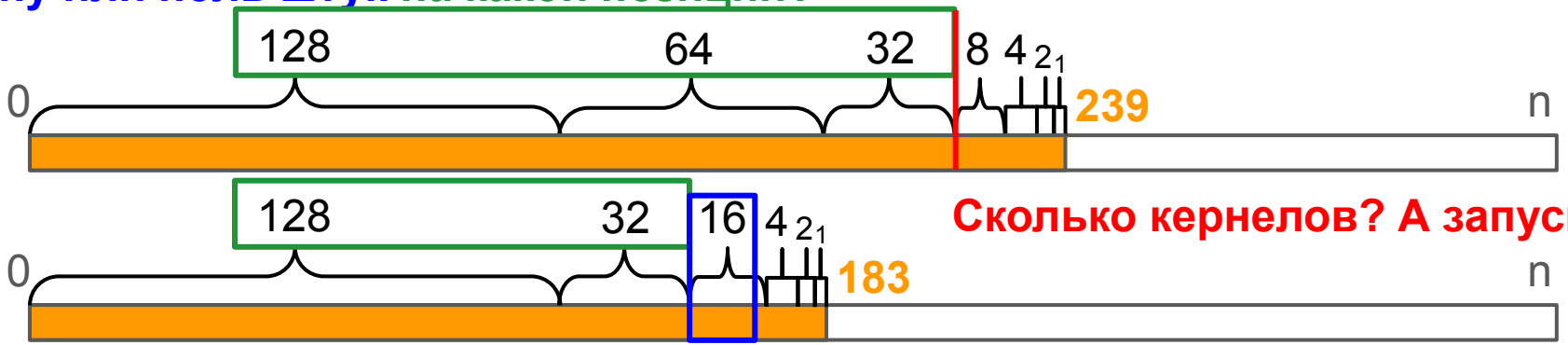
1) **Reduction** чтобы построить суммы размера 2 на четных позициях *(n/2 штук)*

**k**) **Reduction** чтобы построить суммы размера **2^k** на каждой **2^k** позиции ***(n/k)***

5) **Reduction** чтобы построить суммы размера 32 на каждой 32-ой позиции *(n/32)*

Пусть мы WorkItem, как понять, нужно ли добавить себе **сумму размера 16?**

**Одну или ноль штук на какой позиции?**




**Сколько кернелов? А запусков?**

$239_2$ = 128: 1, 64: 1, 32: 1, 16: 0, 8: 1, 4: 1, 2: 1, 1: 1

# Префиксная сумма: **частичные суммы + сборка**

0) Входной массив - суммы размера 1 *(n штук)*

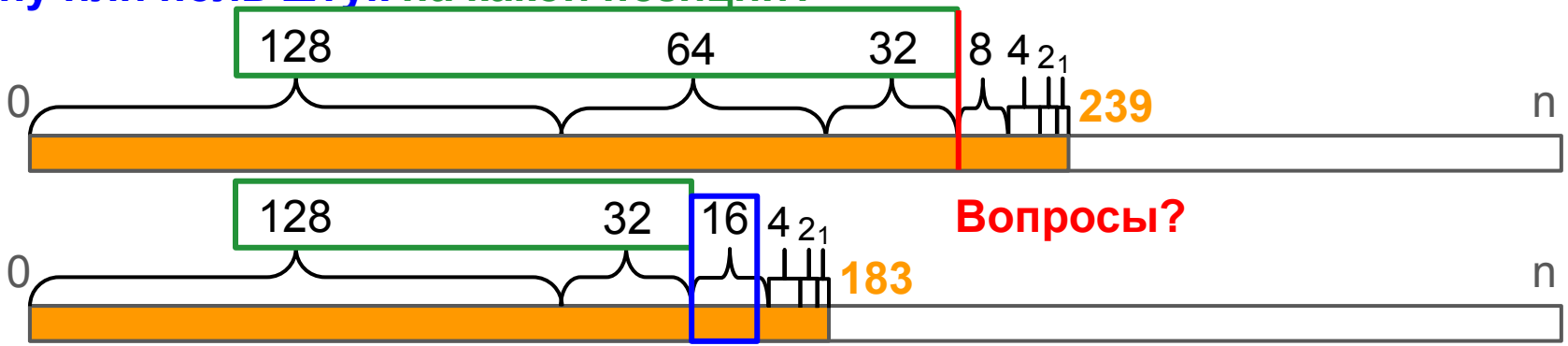
1) **Reduction** чтобы построить суммы размера 2 на четных позициях *(n/2 штук)*

**k**) **Reduction** чтобы построить суммы размера **2^k** на каждой **2^k** позиции ***(n/k)***

5) **Reduction** чтобы построить суммы размера 32 на каждой 32-ой позиции *(n/32)*

Пусть мы WorkItem, как понять, нужно ли добавить себе **сумму размера 16?**

**Одну или ноль штук на какой позиции?**




**Вопросы?**

# Как ускорить?

Какие идеи?

# Как ускорить?

Какие идеи?

Где зарыт главный потенциал ускорения?

Какой ресурс в дефиците?

# Да начнется префиксная гонка!


Перерыв!
Streaming
Pubiprocestor
VIDIA

# Префиксные суммы: применения

- Scan

# Префиксные суммы: применения

- Scan
- Что если каждый WorkItem детектирует ключевую точку в пикселе?
  - кто-то мог решить что пиксель плохой (без текстуры)
  - кто-то другой мог решить что тут есть ключевая точка - по какому индексу в выходному массиву ее записать?

# Префиксные суммы: применения

- Scan
- Что если каждый WorkItem детектирует ключевую точку в пикселе?
  - кто-то мог решить что пиксель плохой (без текстуры)
  - кто-то другой мог решить что тут есть ключевая точка - по какому индексу в выходному массиву ее записать?
  - а чем плохо просто сохранять результат с пузырями (“здесь нет точки”)?

# Префиксные суммы: применения

- Scan
- Генерация результата в три прохода:
  - **map** - каждый WorkItem записал 1 если результат есть, иначе 0

# Префиксные суммы: применения

- Scan
- Генерация результата в три прохода:
  - **map** - каждый WorkItem записал 1 если результат есть, иначе 0
  - **scan** - посчитали префиксные суммы PrefixSum[i]

# Префиксные суммы: применения

- Scan
- Генерация результата в три прохода:
  - **map** - каждый WorkItem записал 1 если результат есть, иначе 0
  - **scan** - посчитали префиксные суммы PrefixSum[i]
  - куда теперь должен писать свой результат WorkItem i?

# Префиксные суммы: применения

- Scan
- Генерация результата в три прохода:
  - **map** - каждый WorkItem записал 1 если результат есть, иначе 0
  - **scan** - посчитали префиксные суммы PrefixSum[i]
  - **scatter** - WorkItem i пишет свой *результат* в *Result*[**PrefixSum[i]**]

# Префиксные суммы: применения

- Scan
- Генерация результата в три прохода:
  - **map** - каждый WorkItem записал 1 если результат есть, иначе 0
  - **scan** - посчитали префиксные суммы PrefixSum[i]
  - **scatter** - WorkItem i пишет свой *результат* в *Result*[**PrefixSum[i]**]
- Radix-sort

# Префиксные суммы: применения

- Scan
- Генерация результата в три прохода:
  - **map** - каждый WorkItem записал 1 если результат есть, иначе 0
  - **scan** - посчитали префиксные суммы PrefixSum[i]
  - **scatter** - WorkItem i пишет свой *результат* в *Result*[**PrefixSum[i]**]
- Radix-sort
- CSR-форматы разреженных матриц
- …

# Отладка

**Как отладить и отдебажить В-С-Ё?**

# Отладка

1) Фиксируйте **минимальный репродюсер** (минимальное **n**)

# Отладка

1) Фиксируйте **минимальный репродюсер** (минимальное **n**)
   - проще держать в голове

# Отладка

1) Фиксируйте **минимальный репродюсер** (минимальное **n**)
   - проще держать в голове
   - легче посмотреть на все происходящее (можно данные заменить на 111111)

# Отладка

1) Фиксируйте **минимальный репродюсер** (минимальное **n**)
   - проще держать в голове
   - легче посмотреть на все происходящее (можно данные заменить на 111111)
   - быстрее каждый запуск

# Отладка

1) Фиксируйте **минимальный репродюсер** (минимальное **n**)
   - проще держать в голове
   - легче посмотреть на все происходящее (можно данные заменить на 111111)
   - быстрее каждый запуск
2) Убедитесь что он **стабилен** - запустите например 5 или 10 раз

# Отладка

1) Фиксируйте **минимальный репродюсер** (минимальное **n**)
   - проще держать в голове
   - легче посмотреть на все происходящее (можно данные заменить на 111111)
   - быстрее каждый запуск
2) Убедитесь что он **стабилен** - запустите например 5 или 10 раз
3) **Локализуйте проблему** до минимального контекста:

# Отладка

1) Фиксируйте **минимальный репродюсер** (минимальное **n**)
   - проще держать в голове
   - легче посмотреть на все происходящее (можно данные заменить на 111111)
   - быстрее каждый запуск
2) Убедитесь что он **стабилен** - запустите например 5 или 10 раз
3) **Локализуйте проблему** до минимального контекста:
   - то есть чтобы под подозрением было минимум кода и переменных

# Отладка

1) Фиксируйте **минимальный репродюсер** (минимальное **n**)
   - проще держать в голове
   - легче посмотреть на все происходящее (можно данные заменить на 111111)
   - быстрее каждый запуск
2) Убедитесь что он **стабилен** - запустите например 5 или 10 раз
3) **Локализуйте проблему** до минимального контекста:
   - то есть чтобы под подозрением было минимум кода и переменных
   - посмотрите на вывод массивов после каждого кернела

# Отладка

1) Фиксируйте **минимальный репродюсер** (минимальное **n**)
   - проще держать в голове
   - легче посмотреть на все происходящее (можно данные заменить на 111111)
   - быстрее каждый запуск
2) Убедитесь что он **стабилен** - запустите например 5 или 10 раз
3) **Локализуйте проблему** до минимального контекста:
   - то есть чтобы под подозрением было минимум кода и переменных
   - посмотрите на вывод массивов после каждого кернела
   - найдите первый шаг когда появились неверные результаты

# Отладка

1) Фиксируйте **минимальный репродюсер** (минимальное **n**)
   - проще держать в голове
   - легче посмотреть на все происходящее (можно данные заменить на 111111)
   - быстрее каждый запуск
2) Убедитесь что он **стабилен** - запустите например 5 или 10 раз
3) **Локализуйте проблему** до минимального контекста:
   - то есть чтобы под подозрением было минимум кода и переменных
   - посмотрите на вывод массивов после каждого кернела
   - найдите первый шаг когда появились неверные результаты
4) Когда зажали конкретный запуск кернела - **локализация/отладка** в кернеле:

# Отладка

1) Фиксируйте **минимальный репродюсер** (минимальное **n**)
   - проще держать в голове
   - легче посмотреть на все происходящее (можно данные заменить на 111111)
   - быстрее каждый запуск
2) Убедитесь что он **стабилен** - запустите например 5 или 10 раз
3) **Локализуйте проблему** до минимального контекста:
   - то есть чтобы под подозрением было минимум кода и переменных
   - посмотрите на вывод массивов после каждого кернела
   - найдите первый шаг когда появились неверные результаты
4) Когда зажали конкретный запуск кернела - **локализация/отладка** в кернеле:
   - compute-sanitizer (входит в CUDA SDK) или Oclgrind

# Отладка

1) Фиксируйте **минимальный репродюсер** (минимальное **n**)
   - проще держать в голове
   - легче посмотреть на все происходящее (можно данные заменить на 111111)
   - быстрее каждый запуск
2) Убедитесь что он **стабилен** - запустите например 5 или 10 раз
3) **Локализуйте проблему** до минимального контекста:
   - то есть чтобы под подозрением было минимум кода и переменных
   - посмотрите на вывод массивов после каждого кернела
   - найдите первый шаг когда появились неверные результаты
4) Когда зажали конкретный запуск кернела - **локализация/отладка** в кернеле:
   - compute-sanitizer (входит в CUDA SDK) или Oclgrind
   - rassert-ы везде где вы можете проверить хоть какой-то инвариант

# Отладка

1) Фиксируйте **минимальный репродюсер** (минимальное **n**)
   - проще держать в голове
   - легче посмотреть на все происходящее (можно данные заменить на 111111)
   - быстрее каждый запуск
2) Убедитесь что он **стабилен** - запустите например 5 или 10 раз
3) **Локализуйте проблему** до минимального контекста:
   - то есть чтобы под подозрением было минимум кода и переменных
   - посмотрите на вывод массивов после каждого кернела
   - найдите первый шаг когда появились неверные результаты
4) Когда зажали конкретный запуск кернела - **локализация/отладка** в кернеле:
   - compute-sanitizer (входит в CUDA SDK) или Oclgrind
   - rassert-ы везде где вы можете проверить хоть какой-то инвариант
   - if (WorkItem.x == <проблемный индекс>) printf(все важные переменные);

# Отладка

**Что отловит неинициализированный глобальный буфер?**

1) Фиксируйте **минимальный репродюсер** (минимальное **n**)
   - проще держать в голове
   - легче посмотреть на все происходящее (можно данные заменить на 111111)
   - быстрее каждый запуск
2) Убедитесь что он **стабилен** - запустите например 5 или 10 раз
3) **Локализуйте проблему** до минимального контекста:
   - то есть чтобы под подозрением было минимум кода и переменных
   - посмотрите на вывод массивов после каждого кернела
   - найдите первый шаг когда появились неверные результаты
4) Когда зажали конкретный запуск кернела - **локализация/отладка** в кернеле:
   - compute-sanitizer (входит в CUDA SDK) или Oclgrind
   - rassert-ы везде где вы можете проверить хоть какой-то инвариант
   - if (WorkItem.x == <проблемный индекс>) printf(все важные переменные);

# Отладка

**Что отловит неинициализированный глобальный буфер?**

1) Фиксируйте **минимальный репродюсер** (минимальное **n**)
   - проще держать в голове
   - легче посмотреть на все происходящее (можно данные заменить на 111111)
   - быстрее каждый запуск
2) Убедитесь что он **стабилен** - запустите например 5 или 10 раз
3) **Локализуйте проблему** до минимального контекста:
   - то есть чтобы под подозрением было минимум кода и переменных
   - посмотрите на вывод массивов после каждого кернела
   - найдите первый шаг когда появились неверные результаты
4) Когда зажали конкретный запуск кернела - **локализация/отладка** в кернеле:
   - compute-sanitizer (входит в CUDA SDK) или Oclgrind
   - rassert-ы везде где вы можете проверить хоть какой-то инвариант
   - if (WorkItem.x == <проблемный индекс>) printf(все важные переменные);

# Отладка

## Как отладить сегфолт в кернеле? (out-of-bounds)

1) Фиксируйте **минимальный репродюсер** (минимальное **n**)
   - проще держать в голове
   - легче посмотреть на все происходящее (можно данные заменить на 111111)
   - быстрее каждый запуск
2) Убедитесь что он **стабилен** - запустите например 5 или 10 раз
3) **Локализуйте проблему** до минимального контекста:
   - то есть чтобы под подозрением было минимум кода и переменных
   - посмотрите на вывод массивов после каждого кернела
   - найдите первый шаг когда появились неверные результаты
4) Когда зажали конкретный запуск кернела - **локализация/отладка** в кернеле:
   - compute-sanitizer (входит в CUDA SDK) или Oclgrind
   - rassert-ы везде где вы можете проверить хоть какой-то инвариант
   - if (WorkItem.x == <проблемный индекс>) printf(все важные переменные);

# Отладка

**Как отладить сегфолт в кернеле? (out-of-bounds)**

1) Фиксируйте **минимальный репродюсер** (минимальное **n**)
   - проще держать в голове
   - легче посмотреть на все происходящее (можно данные заменить на 111111)
   - быстрее каждый запуск
2) Убедитесь что он **стабилен** - запустите например 5 или 10 раз
3) **Локализуйте проблему** до минимального контекста:
   - то есть чтобы под подозрением было минимум кода и переменных
   - посмотрите на вывод массивов после каждого кернела
   - найдите первый шаг когда появились неверные результаты
4) Когда зажали конкретный запуск кернела - **локализация/отладка** в кернеле:
   - compute-sanitizer (входит в CUDA SDK) или Oclgrind
   - rassert-ы везде где вы можете проверить хоть какой-то инвариант
   - if (WorkItem.x == <проблемный индекс>) printf(все важные переменные);

# Отладка

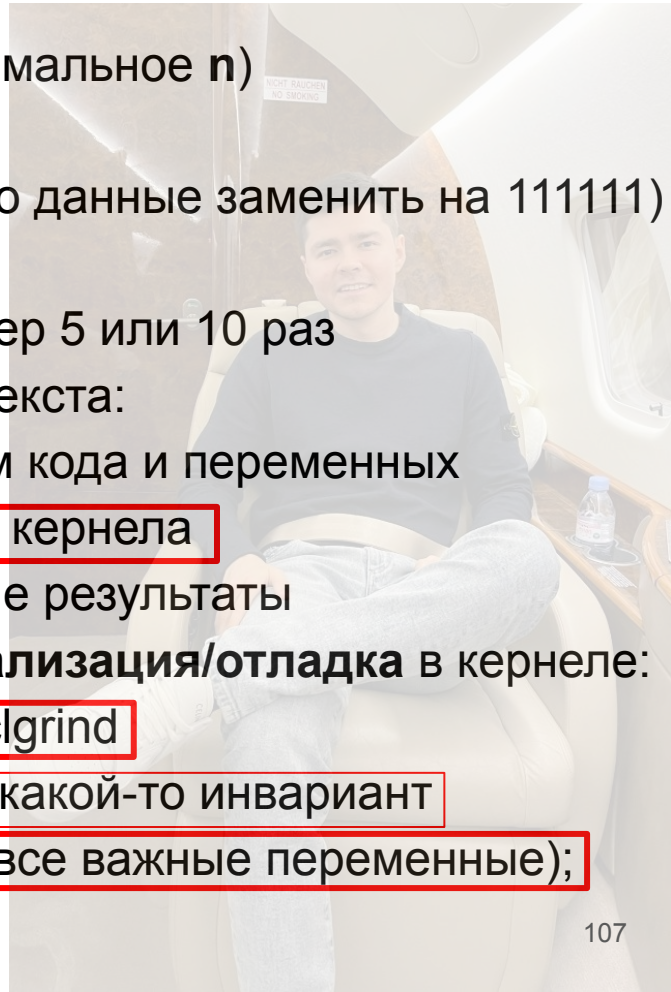
**Что отловит забытый барьер?**

1) Фиксируйте **минимальный репродюсер** (минимальное **n**)
   - проще держать в голове
   - легче посмотреть на все происходящее (можно данные заменить на 111111)
   - быстрее каждый запуск
2) Убедитесь что он **стабилен** - запустите например 5 или 10 раз
3) **Локализуйте проблему** до минимального контекста:
   - то есть чтобы под подозрением было минимум кода и переменных
   - посмотрите на вывод массивов после каждого кернела
   - найдите первый шаг когда появились неверные результаты
4) Когда зажали конкретный запуск кернела - **локализация/отладка** в кернеле:
   - compute-sanitizer (входит в CUDA SDK) или Oclgrind
   - rassert-ы везде где вы можете проверить хоть какой-то инвариант
   - if (WorkItem.x == <проблемный индекс>) printf(все важные переменные);

# Отладка

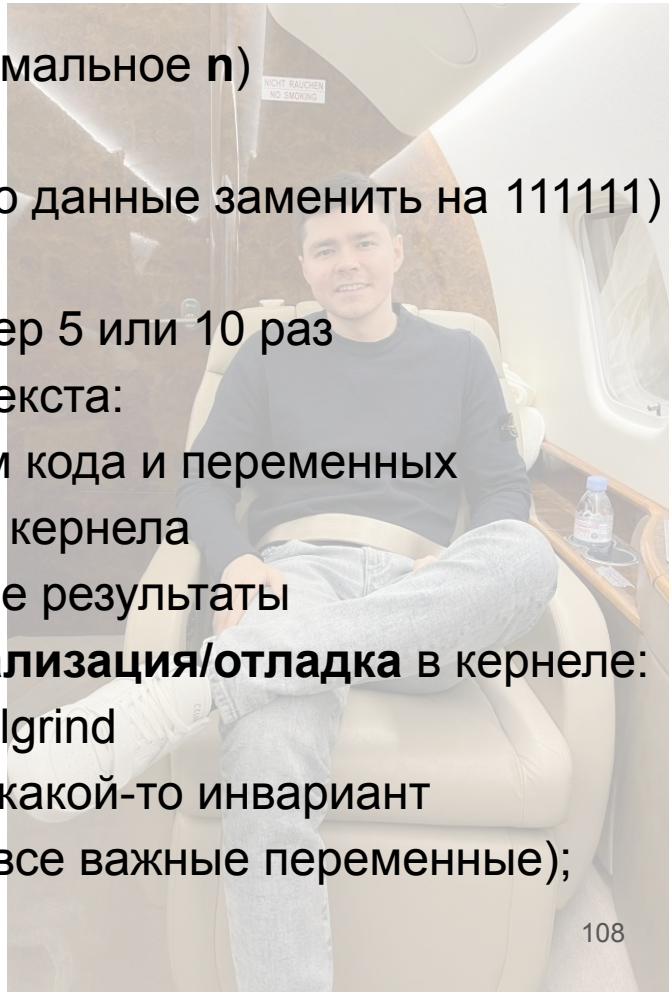
**Что отловит забытый барьер?**

1) Фиксируйте **минимальный репродюсер** (минимальное **n**)
   - проще держать в голове
   - легче посмотреть на все происходящее (можно данные заменить на 111111)
   - быстрее каждый запуск
2) Убедитесь что он **стабилен** - запустите например 5 или 10 раз
3) **Локализуйте проблему** до минимального контекста:
   - то есть чтобы под подозрением было минимум кода и переменных
   - посмотрите на вывод массивов после каждого кернела
   - найдите первый шаг когда появились неверные результаты
4) Когда зажали конкретный запуск кернела - **локализация/отладка** в кернеле:
   - compute-sanitizer (входит в CUDA SDK) или Oclgrind
   - rassert-ы везде где вы можете проверить хоть какой-то инвариант
   - if (WorkItem.x == <проблемный индекс>) printf(все важные переменные);



# Отладка

**Какая у вас была бага? Как можно ее технично отладить?**




1) Фиксируйте **минимальный репродюсер** (минимальное **n**)
   - проще держать в голове
   - легче посмотреть на все происходящее (можно данные заменить на 111111)
   - быстрее каждый запуск
2) Убедитесь что он **стабилен** - запустите например 5 или 10 раз
3) **Локализуйте проблему** до минимального контекста:
   - то есть чтобы под подозрением было минимум кода и переменных
   - посмотрите на вывод массивов после каждого кернела
   - найдите первый шаг когда появились неверные результаты
4) Когда зажали конкретный запуск кернела - **локализация/отладка** в кернеле:
   - compute-sanitizer (входит в CUDA SDK) или Oclgrind
   - rassert-ы везде где вы можете проверить хоть какой-то инвариант
   - if (WorkItem.x == <проблемный индекс>) printf(все важные переменные);

# OpenCL/CUDA software симулятор

```
__global__ void sum_03_local_memory_atomic_per_workgroup(
    const unsigned int* values,
    unsigned int* global_acc,
    unsigned int  n)
{
```

# OpenCL/CUDA software симулятор

Сложно ли было делать задание про Фрактал Мандельброта?

# OpenCL/CUDA software симулятор

Сложно ли было делать задание про Фрактал Мандельброта?

Сложно ли было делать задание про суммирование чисел массива?

# OpenCL/CUDA software симулятор

```
__global__ void sum_03_local_memory_atomic_per_workgroup(
    const unsigned int* values,
    unsigned int* global_acc,
    unsigned int  n)
{
    const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;
    __shared__ unsigned int shared_data[GROUP_SIZE];

    shared_data[threadIdx.x] = (index < n) ? values[index] : 0;
```

# OpenCL/CUDA software симулятор

```
__global__ void sum_03_local_memory_atomic_per_workgroup(
    const unsigned int* values,
    unsigned int* global_acc,
    unsigned int  n)
{
    const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;
    __shared__ unsigned int shared_data[GROUP_SIZE];

    shared_data[threadIdx.x] = (index < n) ? values[index] : 0;

    __syncthreads();
```

зачем?

# OpenCL/CUDA software симулятор

```
__global__ void sum_03_local_memory_atomic_per_workgroup(
    const unsigned int* values,
    unsigned int* global_acc,
    unsigned int  n)
{
    const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;
    __shared__ unsigned int shared_data[GROUP_SIZE];

    shared_data[threadIdx.x] = (index < n) ? values[index] : 0;

    __syncthreads();

    if (threadIdx.x == 0) { // master-thread
        unsigned int wg_acc = 0;
        for (int i = 0; i < GROUP_SIZE; ++i) {
            wg_acc += shared_data[i];
        }
```

# OpenCL/CUDA software симулятор

```
__global__ void sum_03_local_memory_atomic_per_workgroup(
    const unsigned int* values,
    unsigned int* global_acc,
    unsigned int  n)
{
    const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;
    __shared__ unsigned int shared_data[GROUP_SIZE];

    shared_data[threadIdx.x] = (index < n) ? values[index] : 0;

    __syncthreads();

    if (threadIdx.x == 0) { // master-thread
        unsigned int wg_acc = 0;
        for (int i = 0; i < GROUP_SIZE; ++i) {
            wg_acc += shared_data[i];
        }

        atomicAdd(global_acc, wg_acc);
    }
}
```

# OpenCL/CUDA software симулятор

```
__global__ void sum_03_local_memory_atomic_per_workgroup(
    const unsigned int* values,
    unsigned int* global_acc,
    unsigned int  n)
{
    const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;
    __shared__ unsigned int shared_data[GROUP_SIZE];

    shared_data[threadIdx.x] = (index < n) ? values[index] : 0;

    __syncthreads();

    if (threadIdx.x == 0) { // master-thread
        unsigned int wg_acc = 0;
        for (int i = 0; i < GROUP_SIZE; ++i) {
            wg_acc += shared_data[i];
        }

        atomicAdd(global_acc, wg_acc);
    }
}
```

А как можно локально суммировать?

# OpenCL/CUDA software симулятор

```
__global__ void sum_03_local_memory_atomic_per_workgroup(
    const unsigned int* values,
    unsigned int* global_acc,
    unsigned int  n)
{
    const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;
    __shared__ unsigned int shared_data[GROUP_SIZE];

    shared_data[threadIdx.x] = (index < n) ? values[index] : 0;

    __syncthreads();

    if (threadIdx.x == 0) { // master-thread
        unsigned int wg_acc = 0;
        for (int i = 0; i < GROUP_SIZE; ++i) {
            wg_acc += shared_data[i];
        }

        atomicAdd(global_acc, wg_acc);
    }
}
```

**atomicAdd(local_acc, values[index]);**

А как можно локально суммировать?

# OpenCL/CUDA software симулятор

```
__global__ void sum_03_local_memory_atomic_per_workgroup(
    const unsigned int* values,
    unsigned int* global_acc,
    unsigned int  n)
{
    const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;
    __shared__ unsigned int shared_data[GROUP_SIZE];

    shared_data[threadIdx.x] = (index < n) ? values[index] : 0;

    __syncthreads();

    if (threadIdx.x == 0) { // master-thread
        unsigned int wg_acc = 0;
        for (int i = 0; i < GROUP_SIZE; ++i) {
            wg_acc += shared_data[i];
        }

        atomicAdd(global_acc, wg_acc);
    }
}
```

**atomicAdd(local_acc, values[index]);**

А как найти минимальное значение?
(не везде есть atomicMin, но есть atomicCAS - Compare And Set)

# OpenCL/CUDA software симулятор

```
__global__ void sum_03_local_memory_atomic_per_workgroup(
    const unsigned int* values,
    unsigned int* global_acc,
    unsigned int  n)
{
    const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;
    __shared__ unsigned int shared_data[GROUP_SIZE];

    shared_data[threadIdx.x] = (index < n) ? values[index] : 0;

    __syncthreads();

    if (threadIdx.x == 0) { // master-thread
        unsigned int wg_acc = 0;
        for (int i = 0; i < GROUP_SIZE; ++i) {
            wg_acc += shared_data[i];
        }

        atomicAdd(global_acc, wg_acc);
    }
}
```

atomicAdd(l

```
void atomic_min_f32(volatile __global float *p, const float val) {
    float cmp = *p;
    while (val < cmp) {
        float old = atomic_cmpxchg_float(p, cmp, val);
        if (old == cmp) {
            break;
        } else {
            cmp = old;
        }
    }
}
```

А как найти минимальное значение?
(не везде есть atomicMin, но есть atomicCAS - Compare And Set)

# OpenCL/CUDA software симулятор

```
__global__ void sum_03_local_memory_atomic_per_workgroup(
    const unsigned int* values,
    unsigned int* global_acc,
    unsigned int  n)
{
    const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;
    __shared__ unsigned int shared_data[GROUP_SIZE];

    shared_data[threadIdx.x] = (index < n) ? values[index] : 0;

    __syncthreads();

    if (threadIdx.x == 0) { // master-thread
        unsigned int wg_acc = 0;
        for (int i = 0; i < GROUP_SIZE; ++i) {
            wg_acc += shared_data[i];
        }

        atomicAdd(global_acc, wg_acc);
    }
}
```

atomicAdd(

```
void atomic_min_f32(volatile __global float *p, const float val) {
    float cmp = *p;
    while (val < cmp) {
        float old = atomic_cmpxchg_float(p, cmp, val);
        if (old == cmp) {
            break;
        } else {
            cmp = old;
        }
    }
}
```

правда CAS на float нет в OpenCL 1.2, но есть для uint32

А как найти минимальное значение?
(не везде есть atomicMin, но есть atomicCAS - Compare And Set)

# OpenCL/CUDA software симулятор

```
__global__ void sum_03_local_memory_atomic_per_workgroup(
    const unsigned int* values,
    unsigned int* global_acc,
    unsigned int  n)
{
    const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;
    __shared__ unsigned int shared_data[GROUP_SIZE];

    shared_data[threadIdx.x] = (index < n) ? values[index] : 0;

    __syncthreads();

    if (threadIdx.x == 0) { // master-thread
        unsigned int wg_acc = 0;
        for (int i = 0; i < GROUP_SIZE; ++i) {
            wg_acc += shared_data[i];
        }

        atomicAdd(global_acc, wg_acc);
    }
}
```

atomicAdd(

```
void atomic_min_f32(volatile __global float *p, const float val) {
    float cmp = *p;
    while (val < cmp) {
        float old = atomic_cmpxchg_float(p, cmp, val);
        if (old == cmp) {
            break;
        } else {
            cmp = old;
        }
    }
}
```

правда CAS на float нет в OpenCL 1.2, но есть для uint32

А как найти минимальное значение?
(не везде есть atomicMin, но есть atomicCAS - Compare And

# OpenCL/CUDA software симулятор

```
float atomic_cmpxchg_f32(volatile __global float *p, float cmp, float val) {
    union {
        unsigned int    u32;
        float           f32;
    } cmp_union, val_union, old_union;

    cmp_union.f32 = cmp;
    val_union.f32 = val;
    old_union.u32 = atomic_cmpxchg((volatile __global unsigned int *) p, cmp_union.u32, val_union.u32);
    return old_union.f32;
}
```

правда CAS на float нет в OpenCL 1.2, но есть для uint32

А как найти минимальное значение?
(не везде есть atomicMin, но есть atomicCAS - Compare And

# OpenCL/CUDA software симулятор

```
__global__ void sum_03_local_memory_atomic_per_workgroup(
    const unsigned int* values,
    unsigned int* global_acc,
    unsigned int  n)
{
    const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;
    __shared__ unsigned int shared_data[GROUP_SIZE];

    shared_data[threadIdx.x] = (index < n) ? values[index] : 0;

    __syncthreads();

    if (threadIdx.x == 0) { // master-thread
        unsigned int wg_acc = 0;
        for (int i = 0; i < GROUP_SIZE; ++i) {
            wg_acc += shared_data[i];
        }

        atomicAdd(global_acc, wg_acc);
    }
}
```

```
unsigned int cpu::sumMimicsCUDA(const unsigned int* values, unsigned int n)
{
    unsigned int global_acc = 0;
```

# OpenCL/CUDA software симулятор

```
__global__ void sum_03_local_memory_atomic_per_workgroup(
    const unsigned int* values,
    unsigned int* global_acc,
    unsigned int  n)
{
    const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;
    __shared__ unsigned int shared_data[GROUP_SIZE];

    shared_data[threadIdx.x] = (index < n) ? values[index] : 0;

    __syncthreads();

    if (threadIdx.x == 0) { // master-thread
        unsigned int wg_acc = 0;
        for (int i = 0; i < GROUP_SIZE; ++i) {
            wg_acc += shared_data[i];
        }

        atomicAdd(global_acc, wg_acc);
    }
}
```

```
unsigned int cpu::sumMimicsCUDA(const unsigned int* values, unsigned int n)
{
    unsigned int global_acc = 0;

    blockDim_t blockDim = {GROUP_SIZE, 1, 1}; // WorkGroup size
    blockIdx_t blockIdx = {0,          0, 0}; // WorkGroup index
    for (blockIdx.x = 0; blockIdx.x * GROUP_SIZE < n; ++blockIdx.x) {
        unsigned int shared_data[GROUP_SIZE];
```

# OpenCL/CUDA software симулятор

```
__global__ void sum_03_local_memory_atomic_per_workgroup(
    const unsigned int* values,
    unsigned int* global_acc,
    unsigned int  n)
{
    const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;
    __shared__ unsigned int shared_data[GROUP_SIZE];

    shared_data[threadIdx.x] = (index < n) ? values[index] : 0;

    __syncthreads();

    if (threadIdx.x == 0) { // master-thread
        unsigned int wg_acc = 0;
        for (int i = 0; i < GROUP_SIZE; ++i) {
            wg_acc += shared_data[i];
        }

        atomicAdd(global_acc, wg_acc);
    }
}
```

```
unsigned int cpu::sumMimicsCUDA(const unsigned int* values, unsigned int n)
{
    unsigned int global_acc = 0;

    blockDim_t blockDim = {GROUP_SIZE, 1, 1}; // WorkGroup size
    blockIdx_t blockIdx = {0,          0, 0}; // WorkGroup index
    for (blockIdx.x = 0; blockIdx.x * GROUP_SIZE < n; ++blockIdx.x) {
        unsigned int shared_data[GROUP_SIZE];


        #pragma omp parallel for schedule(dynamic, 1) num_threads(GROUP_SIZE)
        for (int threadIdx_x = 0; threadIdx_x < GROUP_SIZE; ++threadIdx_x) {
            threadIdx_t threadIdx = {threadIdx_x, 0, 0}; // local index in WorkGroup
```

Зачем?

# OpenCL/CUDA software симулятор

```
__global__ void sum_03_local_memory_atomic_per_workgroup(
    const unsigned int* values,
    unsigned int* global_acc,
    unsigned int  n)
{
    const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;
    __shared__ unsigned int shared_data[GROUP_SIZE];

    shared_data[threadIdx.x] = (index < n) ? values[index] : 0;

    __syncthreads();

    if (threadIdx.x == 0) { // master-thread
        unsigned int wg_acc = 0;
        for (int i = 0; i < GROUP_SIZE; ++i) {
            wg_acc += shared_data[i];
        }

        atomicAdd(global_acc, wg_acc);
    }
}
```

```
unsigned int cpu::sumMimicsCUDA(const unsigned int* values, unsigned int n)
{
    unsigned int global_acc = 0;

    blockDim_t blockDim = {GROUP_SIZE, 1, 1}; // WorkGroup size
    blockIdx_t blockIdx = {0,          0, 0}; // WorkGroup index
    for (blockIdx.x = 0; blockIdx.x * GROUP_SIZE < n; ++blockIdx.x) {
        unsigned int shared_data[GROUP_SIZE];


        #pragma omp parallel for schedule(dynamic, 1) num_threads(GROUP_SIZE)
        for (int threadIdx_x = 0; threadIdx_x < GROUP_SIZE; ++threadIdx_x) {
            threadIdx_t threadIdx = {threadIdx_x, 0, 0}; // local index in WorkGroup

            // global WorkItem index
            const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;

            shared_data[threadIdx.x] = (index < n) ? values[index] : 0;
```

# OpenCL/CUDA software симулятор

```
__global__ void sum_03_local_memory_atomic_per_workgroup(
    const unsigned int* values,
    unsigned int* global_acc,
    unsigned int  n)
{
    const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;
    __shared__ unsigned int shared_data[GROUP_SIZE];

    shared_data[threadIdx.x] = (index < n) ? values[index] : 0;

    __syncthreads();

    if (threadIdx.x == 0) { // master-thread
        unsigned int wg_acc = 0;
        for (int i = 0; i < GROUP_SIZE; ++i) {
            wg_acc += shared_data[i];
        }

        atomicAdd(global_acc, wg_acc);
    }
}
```

```
unsigned int cpu::sumMimicsCUDA(const unsigned int* values, unsigned int n)
{
    unsigned int global_acc = 0;

    blockDim_t blockDim = {GROUP_SIZE, 1, 1}; // WorkGroup size
    blockIdx_t blockIdx = {0,          0, 0}; // WorkGroup index
    for (blockIdx.x = 0; blockIdx.x * GROUP_SIZE < n; ++blockIdx.x) {
        unsigned int shared_data[GROUP_SIZE];


        #pragma omp parallel for schedule(dynamic, 1) num_threads(GROUP_SIZE)
        for (int threadIdx_x = 0; threadIdx_x < GROUP_SIZE; ++threadIdx_x) {
            threadIdx_t threadIdx = {threadIdx_x, 0, 0}; // local index in WorkGroup

            // global WorkItem index
            const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;

            shared_data[threadIdx.x] = (index < n) ? values[index] : 0;



            if (threadIdx.x == 0) { // master-thread
                unsigned int wg_acc = 0;
                for (int i = 0; i < GROUP_SIZE; ++i) {
                    wg_acc += shared_data[i];
                }
```

# OpenCL/CUDA software симулятор

```
__global__ void sum_03_local_memory_atomic_per_workgroup(
    const unsigned int* values,
    unsigned int* global_acc,
    unsigned int  n)
{
    const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;
    __shared__ unsigned int shared_data[GROUP_SIZE];

    shared_data[threadIdx.x] = (index < n) ? values[index] : 0;

    __syncthreads();

    if (threadIdx.x == 0) { // master-thread
        unsigned int wg_acc = 0;
        for (int i = 0; i < GROUP_SIZE; ++i) {
            wg_acc += shared_data[i];
        }

        atomicAdd(global_acc, wg_acc);
    }
}
```

```
unsigned int cpu::sumMimicsCUDA(const unsigned int* values, unsigned int n)
{
    unsigned int global_acc = 0;

    blockDim_t blockDim = {GROUP_SIZE, 1, 1}; // WorkGroup size
    blockIdx_t blockIdx = {0,          0, 0}; // WorkGroup index
    for (blockIdx.x = 0; blockIdx.x * GROUP_SIZE < n; ++blockIdx.x) {
        unsigned int shared_data[GROUP_SIZE];


        #pragma omp parallel for schedule(dynamic, 1) num_threads(GROUP_SIZE)
        for (int threadIdx_x = 0; threadIdx_x < GROUP_SIZE; ++threadIdx_x) {
            threadIdx_t threadIdx = {threadIdx_x, 0, 0}; // local index in WorkGroup

            // global WorkItem index
            const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;

            shared_data[threadIdx.x] = (index < n) ? values[index] : 0;



            if (threadIdx.x == 0) { // master-thread
                unsigned int wg_acc = 0;
                for (int i = 0; i < GROUP_SIZE; ++i) {
                    wg_acc += shared_data[i];
                }

                // atomicAdd
                #pragma omp critical
                {
                    global_acc += wg_acc;
                }
            }
        }
    }

    return global_acc;
}
```

# OpenCL/CUDA software симулятор

```
__global__ void sum_03_local_memory_atomic_per_workgroup(
    const unsigned int* values,
    unsigned int* global_acc,
    unsigned int  n)
{
    const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;
    __shared__ unsigned int shared_data[GROUP_SIZE];

    shared_data[threadIdx.x] = (index < n) ? values[index] : 0;

    __syncthreads();

    if (threadIdx.x == 0) { // master-thread
        unsigned int wg_acc = 0;
        for (int i = 0; i < GROUP_SIZE; ++i) {
            wg_acc += shared_data[i];
        }

        atomicAdd(global_acc, wg_acc);
    }
}
```

```
unsigned int cpu::sumMimicsCUDA(const unsigned int* values, unsigned int n)
{
    unsigned int global_acc = 0;

    blockDim_t blockDim = {GROUP_SIZE, 1, 1}; // WorkGroup size
    blockIdx_t blockIdx = {0,          0, 0}; // WorkGroup index
    for (blockIdx.x = 0; blockIdx.x * GROUP_SIZE < n; ++blockIdx.x) {
        unsigned int shared_data[GROUP_SIZE];


        #pragma omp parallel for schedule(dynamic, 1) num_threads(GROUP_SIZE)
        for (int threadIdx_x = 0; threadIdx_x < GROUP_SIZE; ++threadIdx_x) {
            threadIdx_t threadIdx = {threadIdx_x, 0, 0}; // local index in WorkGroup

            // global WorkItem index
            const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;

            shared_data[threadIdx.x] = (index < n) ? values[index] : 0;



            if (threadIdx.x == 0) { // master-thread
                unsigned int wg_acc = 0;
                for (int i = 0; i < GROUP_SIZE; ++i) {
                    wg_acc += shared_data[i];
                }

                // atomicAdd
                #pragma omp critical
                {
                    global_acc += wg_acc;
                }
            }
        }
    }

    return global_acc;
}
```

Все ли мы эмулировали?

# OpenCL/CUDA software симулятор

```
__global__ void sum_03_local_memory_atomic_per_workgroup(
    const unsigned int* values,
    unsigned int* global_acc,
    unsigned int  n)
{
    const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;
    __shared__ unsigned int shared_data[GROUP_SIZE];

    shared_data[threadIdx.x] = (index < n) ? values[index] : 0;

    __syncthreads();

    if (threadIdx.x == 0) { // master-thread
        unsigned int wg_acc = 0;
        for (int i = 0; i < GROUP_SIZE; ++i) {
            wg_acc += shared_data[i];
        }

        atomicAdd(global_acc, wg_acc);
    }
}
```

```
unsigned int cpu::sumMimicsCUDA(const unsigned int* values, unsigned int n)
{
    unsigned int global_acc = 0;

    blockDim_t blockDim = {GROUP_SIZE, 1, 1}; // WorkGroup size
    blockIdx_t blockIdx = {0,          0, 0}; // WorkGroup index
    for (blockIdx.x = 0; blockIdx.x * GROUP_SIZE < n; ++blockIdx.x) {
        unsigned int shared_data[GROUP_SIZE];


        #pragma omp parallel for schedule(dynamic, 1) num_threads(GROUP_SIZE)
        for (int threadIdx_x = 0; threadIdx_x < GROUP_SIZE; ++threadIdx_x) {
            threadIdx_t threadIdx = {threadIdx_x, 0, 0}; // local index in WorkGroup

            // global WorkItem index
            const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;

            shared_data[threadIdx.x] = (index < n) ? values[index] : 0;



            if (threadIdx.x == 0) { // master-thread
                unsigned int wg_acc = 0;
                for (int i = 0; i < GROUP_SIZE; ++i) {
                    wg_acc += shared_data[i];
                }

                // atomicAdd
                #pragma omp critical
                {
                    global_acc += wg_acc;
                }
            }
        }
    }

    return global_acc;
}
```

Все ли мы эмулировали?

# OpenCL/CUDA software симулятор

```cpp
__global__ void sum_03_local_memory_atomic_per_workgroup(
    const unsigned int* values,
    unsigned int* global_acc,
    unsigned int  n)
{
    const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;
    __shared__ unsigned int shared_data[GROUP_SIZE];

    shared_data[threadIdx.x] = (index < n) ? values[index] : 0;

    __syncthreads();

    if (threadIdx.x == 0) { // master-thread
        unsigned int wg_acc = 0;
        for (int i = 0; i < GROUP_SIZE; ++i) {
            wg_acc += shared_data[i];
        }

        atomicAdd(global_acc, wg_acc);
    }
}
```

```cpp
void syncthreads(std::atomic<int> &barrier_arrivals_counter, unsigned int group_size)
{
    int v = barrier_arrivals_counter.fetch_add(1, std::memory_order_acq_rel) + 1;
    while (v % group_size != 0) {
        v = barrier_arrivals_counter.load();
        std::this_thread::sleep_for(std::chrono::nanoseconds(1));
    }
}
```

```cpp
unsigned int cpu::sumMimicsCUDA(const unsigned int* values, unsigned int n)
{
    unsigned int global_acc = 0;

    blockDim_t blockDim = {GROUP_SIZE, 1, 1}; // WorkGroup size
    blockIdx_t blockIdx = {0,          0, 0}; // WorkGroup index
    for (blockIdx.x = 0; blockIdx.x * GROUP_SIZE < n; ++blockIdx.x) {
        unsigned int shared_data[GROUP_SIZE];
        std::atomic<int> barrier_arrivals_counter{0};

        #pragma omp parallel for schedule(dynamic, 1) num_threads(GROUP_SIZE)
        for (int threadIdx_x = 0; threadIdx_x < GROUP_SIZE; ++threadIdx_x) {
            threadIdx_t threadIdx = {threadIdx_x, 0, 0}; // local index in WorkGroup

            // global WorkItem index
            const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;

            shared_data[threadIdx.x] = (index < n) ? values[index] : 0;

            syncthreads(barrier_arrivals_counter, GROUP_SIZE);

            if (threadIdx.x == 0) { // master-thread
                unsigned int wg_acc = 0;
                for (int i = 0; i < GROUP_SIZE; ++i) {
                    wg_acc += shared_data[i];
                }

                // atomicAdd
                #pragma omp critical
                {
                    global_acc += wg_acc;
                }
            }
        }
    }

    return global_acc;
}
```

# OpenCL/CUDA software симулятор

```cpp
__global__ void sum_03_local_memory_atomic_per_workgroup(
    const unsigned int* values,
    unsigned int* global_acc,
    unsigned int  n)
{
    const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;
    __shared__ unsigned int shared_data[GROUP_SIZE];

    shared_data[threadIdx.x] = (index < n) ? values[index] : 0;

    __syncthreads();

    if (threadIdx.x == 0) { // master-thread
        unsigned int wg_acc = 0;
        for (int i = 0; i < GROUP_SIZE; ++i) {
            wg_acc += shared_data[i];
        }

        atomicAdd(global_acc, wg_acc);
    }
}
```

```cpp
void syncthreads(std::atomic<int> &barrier_arrivals_counter, unsigned int group_size)
{
    int v = barrier_arrivals_counter.fetch_add(1, std::memory_order_acq_rel) + 1;
    while (v % group_size != 0) {
        v = barrier_arrivals_counter.load();
        std::this_thread::sleep_for(std::chrono::nanoseconds(1));
    }
}
```

```cpp
unsigned int cpu::sumMimicsCUDA(const unsigned int* values, unsigned int n)
{
    unsigned int global_acc = 0;

    blockDim_t blockDim = {GROUP_SIZE, 1, 1}; // WorkGroup size
    blockIdx_t blockIdx = {0,          0, 0}; // WorkGroup index
    for (blockIdx.x = 0; blockIdx.x * GROUP_SIZE < n; ++blockIdx.x) {
        unsigned int shared_data[GROUP_SIZE];
        std::atomic<int> barrier_arrivals_counter{0};

        #pragma omp parallel for schedule(dynamic, 1) num_threads(GROUP_SIZE)
        for (int threadIdx_x = 0; threadIdx_x < GROUP_SIZE; ++threadIdx_x) {
            threadIdx_t threadIdx = {threadIdx_x, 0, 0}; // local index in WorkGroup

            // global WorkItem index
            const unsigned int index = blockIdx.x * blockDim.x + threadIdx.x;

            shared_data[threadIdx.x] = (index < n) ? values[index] : 0;

            syncthreads(barrier_arrivals_counter, GROUP_SIZE);

            if (threadIdx.x == 0) { // master-thread
                unsigned int wg_acc = 0;
                for (int i = 0; i < GROUP_SIZE; ++i) {
                    wg_acc += shared_data[i];
                }

                // atomicAdd
                #pragma omp critical
                {
                    global_acc += wg_acc;
                }
            }
        }
    }

    return global_acc;
}
```

```cpp
#define threadIdx_t uint3
#define blockIdx_t uint3
#define blockDim_t dim3
```

Если вы не знаете как что-то софтварно сэмулировать,
то вы не знаете как оно работает!

*Лао Цзы, Искусство войны, § 69*

CS Space
Клуб технологий и науки

Vulkan

NVIDIA
CUDA

OpenCL™

# Вопросы?


@UnicornGlade
@PolarNick239
polarnick239@gmail.com
Николай Полярный