Metal без магии - когда GPU ускоряет, а не мешает

Mobius Spring 2026

Maxim Lomakin

План доклада

- Вступление.
- Метрики пользователя.
- Метрики ресурсов и оптимизации.
- Какие еще скрываются тонкости?
- Подведение итогов.

Вступление

Ожидание vs реальность

- Решили использовать Metal для решения задачи, но не знаем как организовать архитектуру.

Вступление

Ожидание vs реальность

- Решили использовать Metal для решения задачи, но не знаем как организовать архитектуру.
- Попробовали переписать существующее решение на Metal, но получили такой же результат или даже ухудшение производительности.

Вступление

Ожидание vs реальность

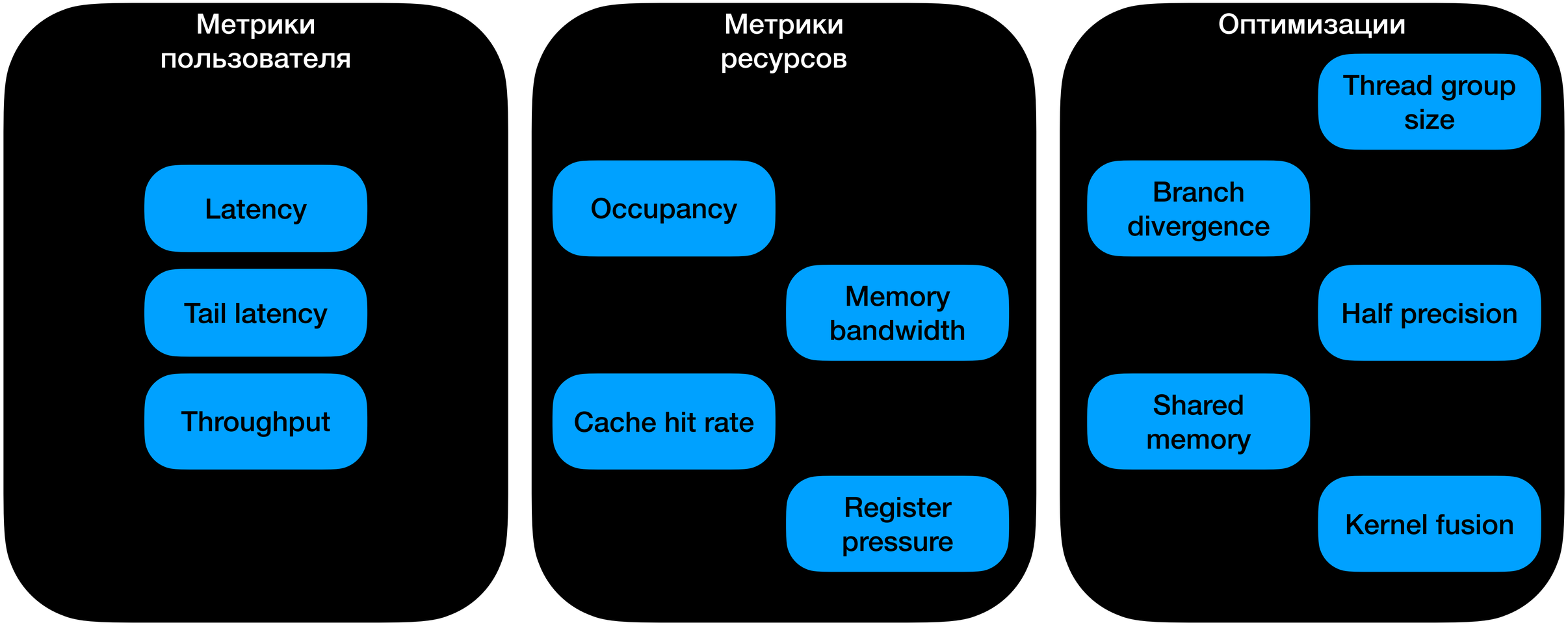
- Решили использовать Metal для решения задачи, но не знаем как организовать архитектуру.
- Попробовали переписать существующее решение на Metal, но получили такой же результат или даже ухудшение производительности.
- Ожидается, что Metal - это ускорение само по себе.

Вступление

Ожидание vs реальность

- Решили использовать Metal для решения задачи, но не знаем как организовать архитектуру.
- Попробовали переписать существующее решение на Metal, но получили такой же результат или даже ухудшение производительности.
- Ожидается, что Metal - это ускорение само по себе.
- Для **корректной работы с Metal** необходимо **понимать** как **измерить** его **производительность** и как **выстроить работу** в целом.

Как измерить «производительность»?



Как измерить «производительность»?

Метрики пользователя

Это то, что чувствует пользователь

- Latency
- Tail latency
- Throughput

Метрики ресурсов

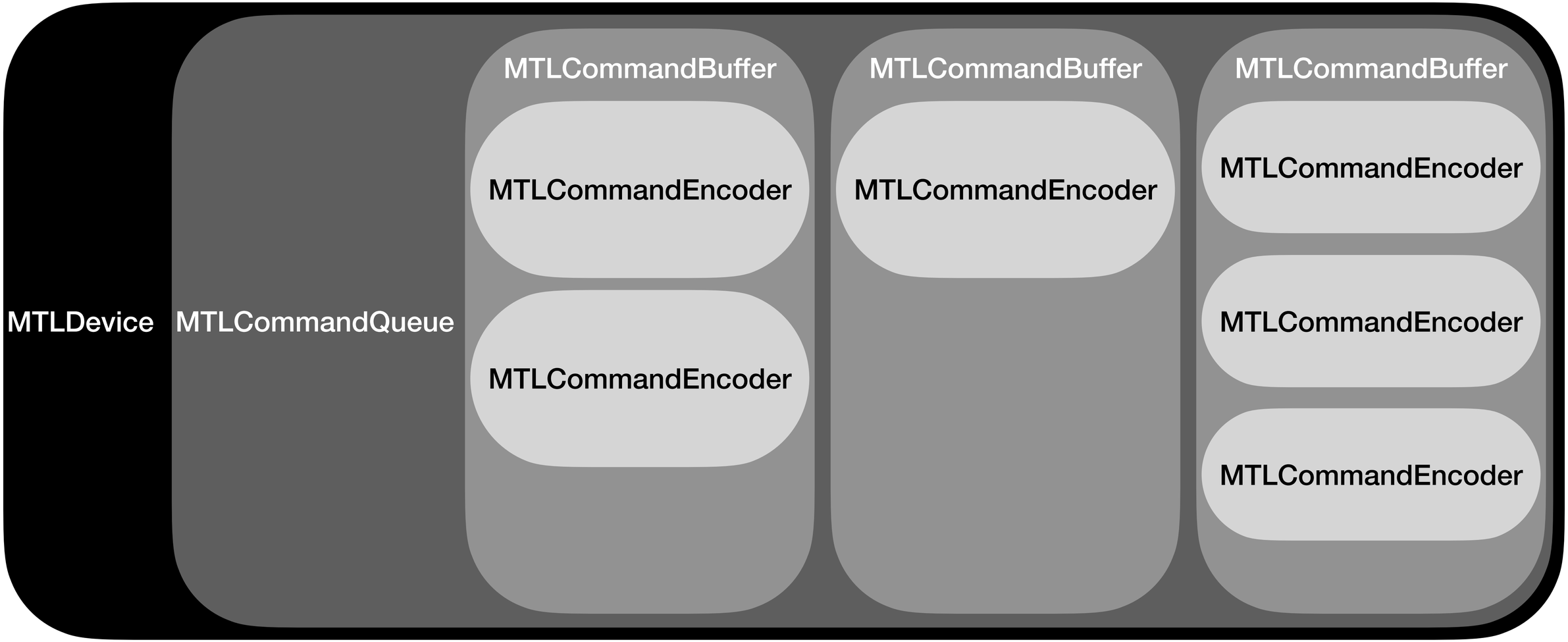
- Occupancy
- Memory bandwidth
- Cache hit rate
- Register pressure

Оптимизации

- Thread group size
- Branch divergence
- Half precision
- Shared memory
- Kernel fusion

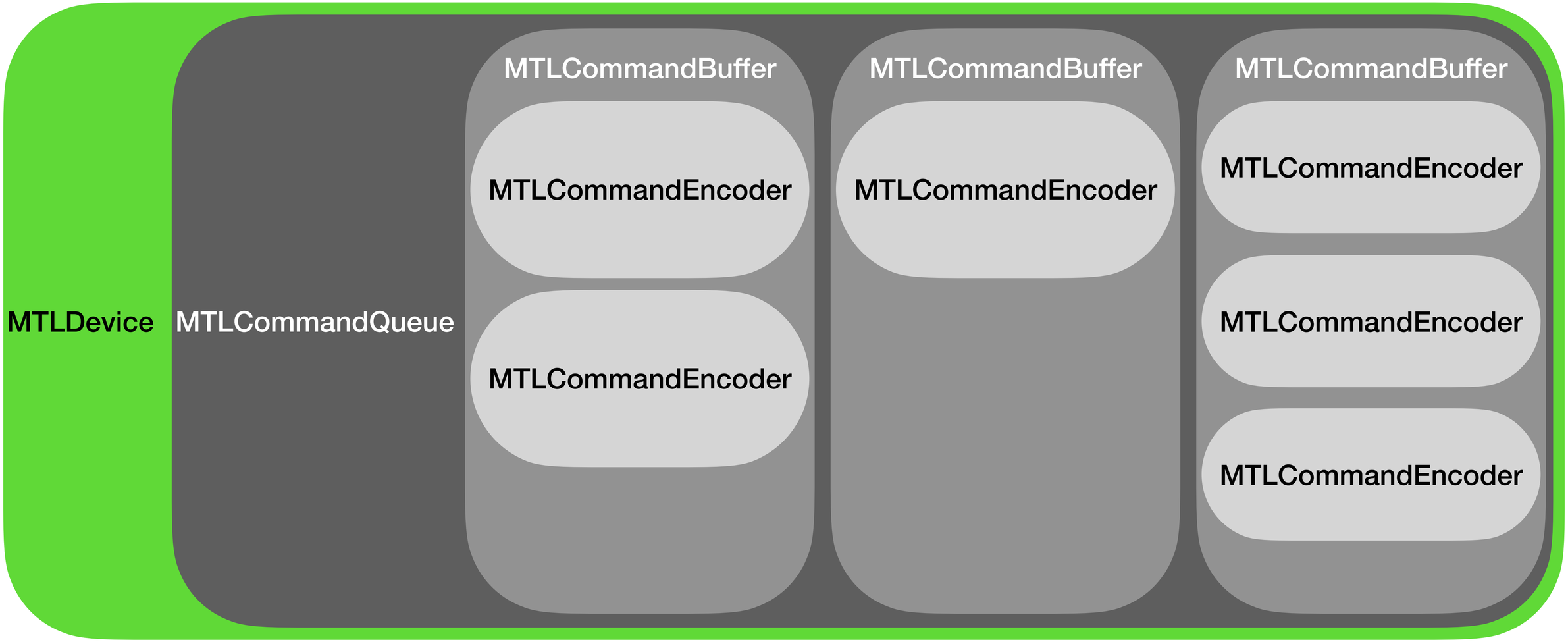
Пройдемся по понятиям

Основные объекты Metal



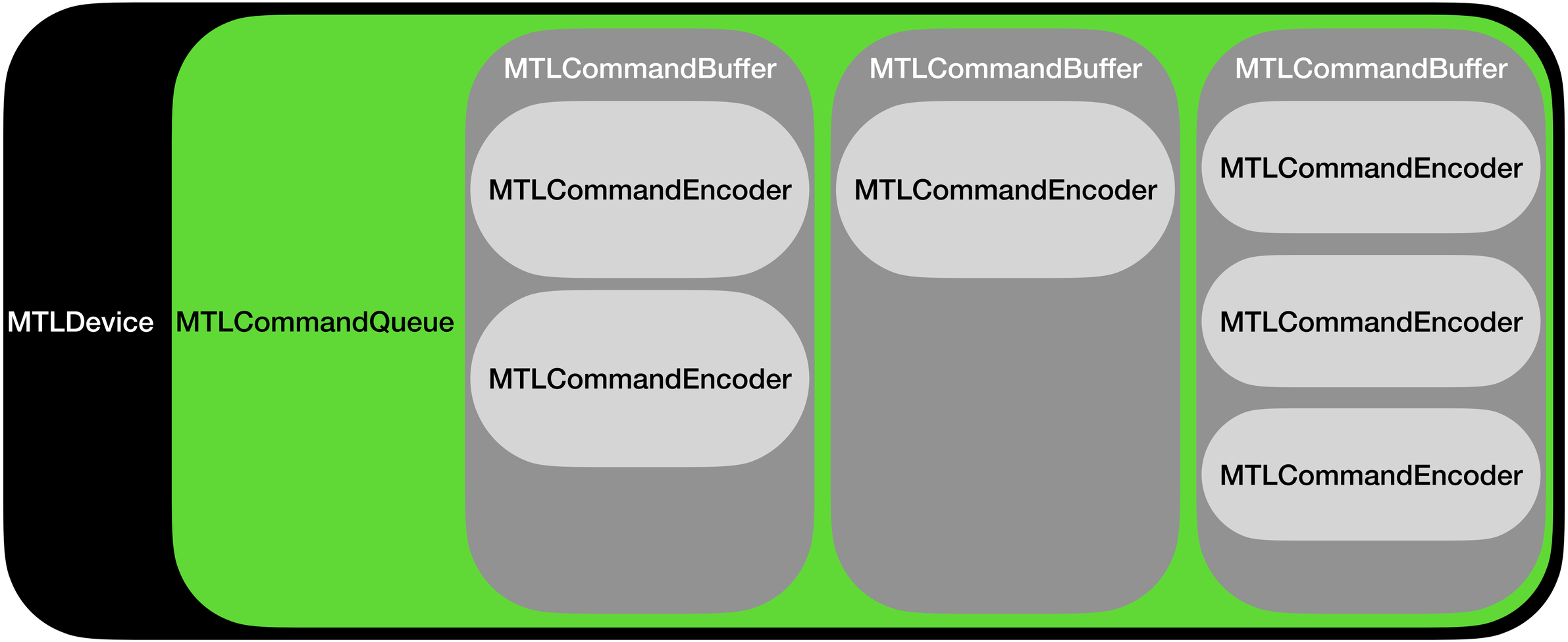
Пройдемся по понятиям

Основные объекты Metal



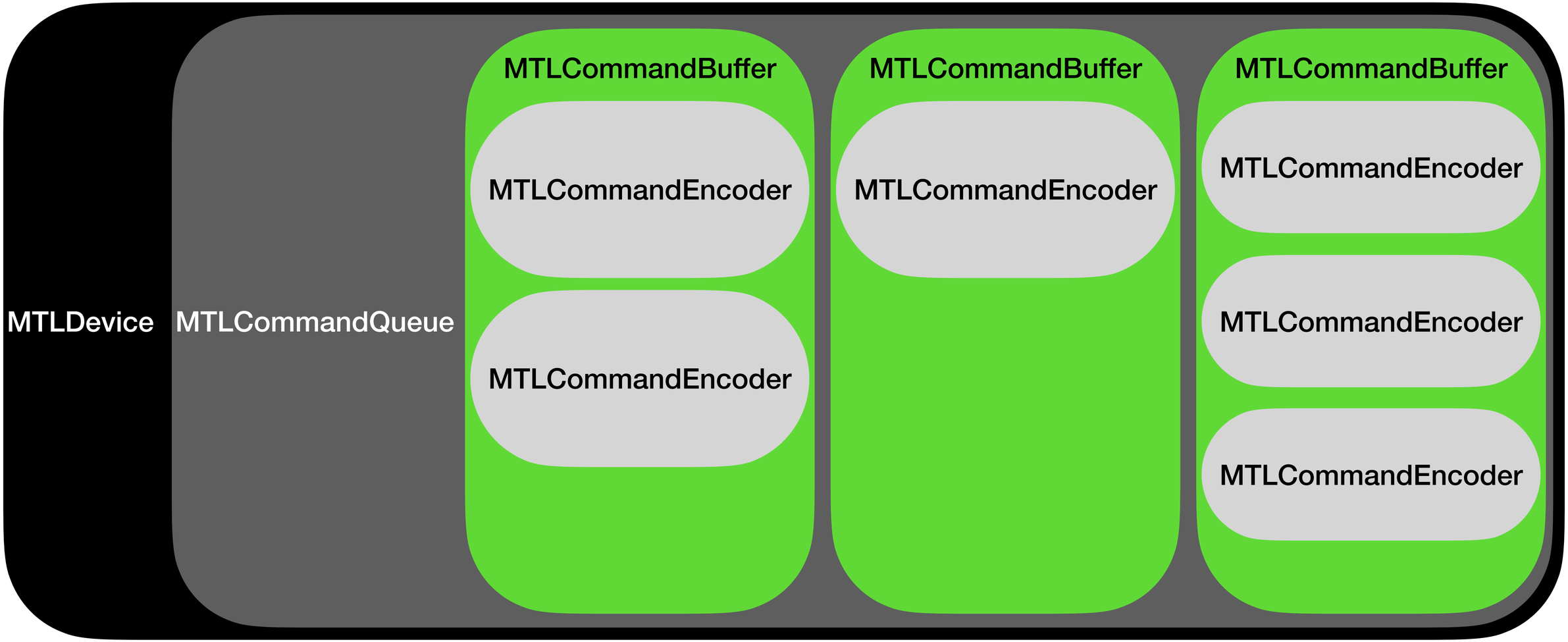
Пройдемся по понятиям

Основные объекты Metal



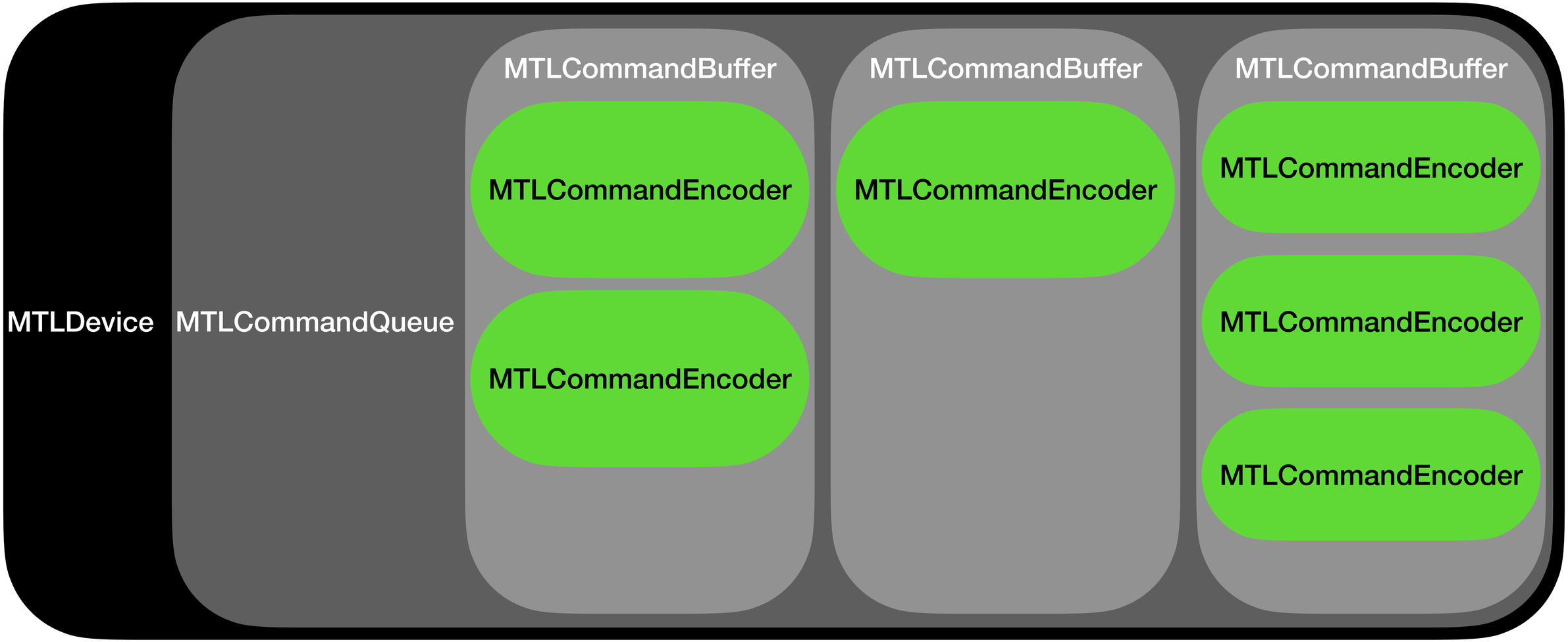
Пройдемся по понятиям

Основные объекты Metal



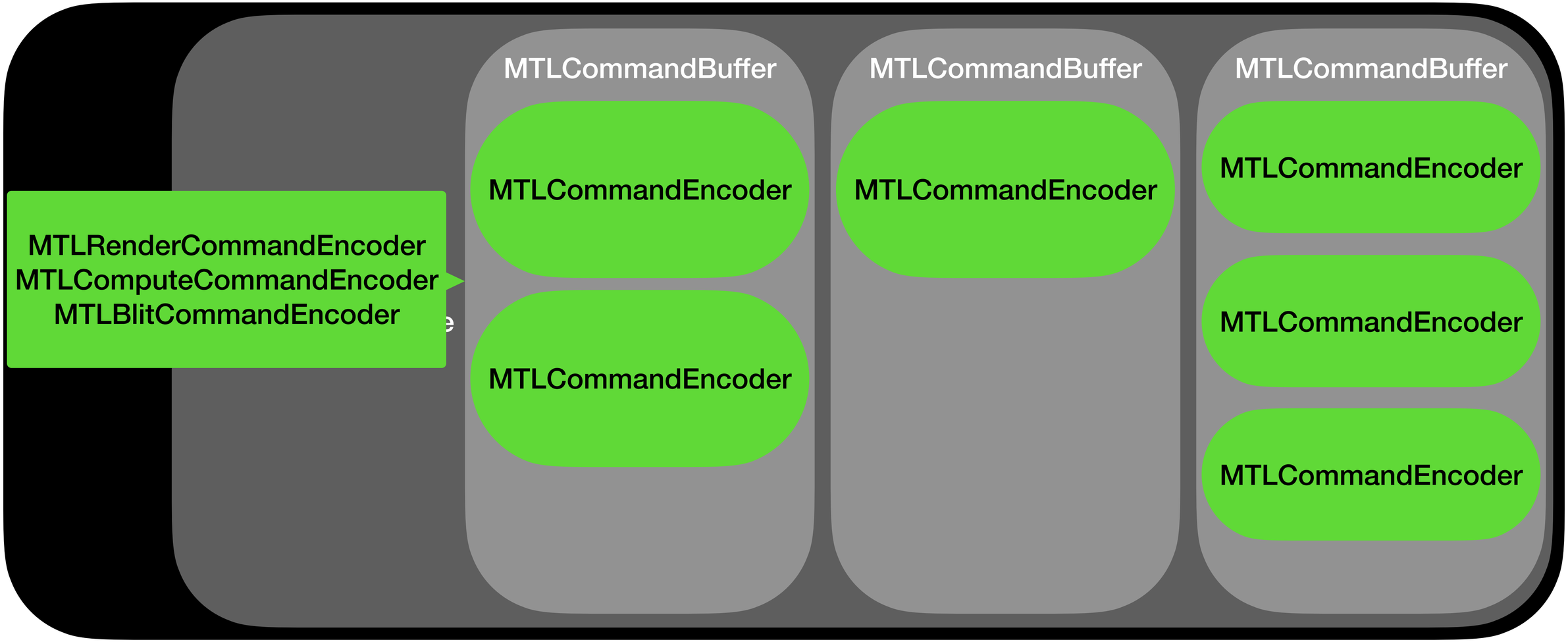
Пройдемся по понятиям

Основные объекты Metal



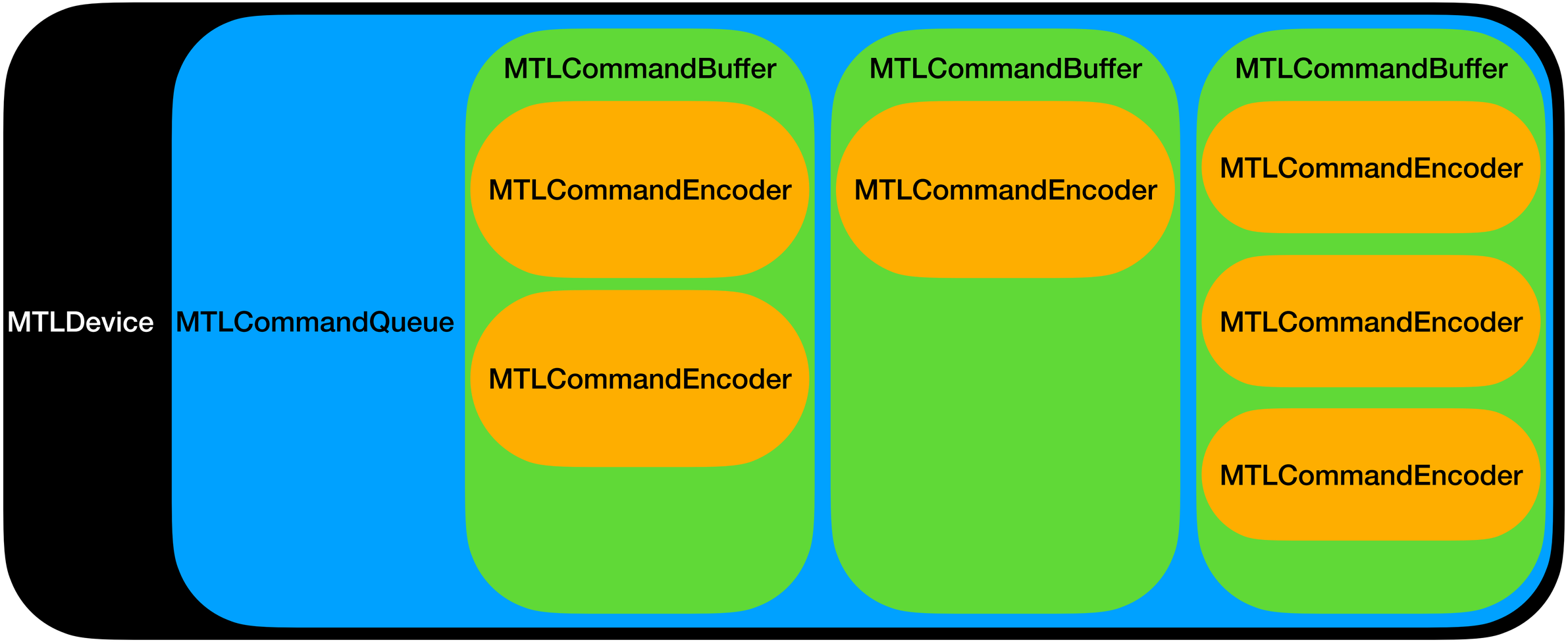
Пройдемся по понятиям

Основные объекты Metal



Пройдемся по понятиям

Основные объекты Metal



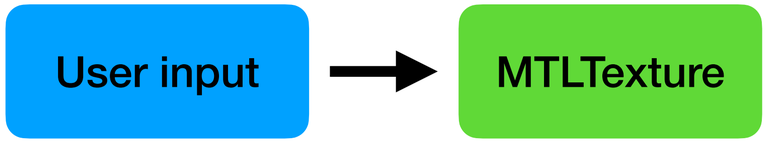
Пройдемся по понятиям

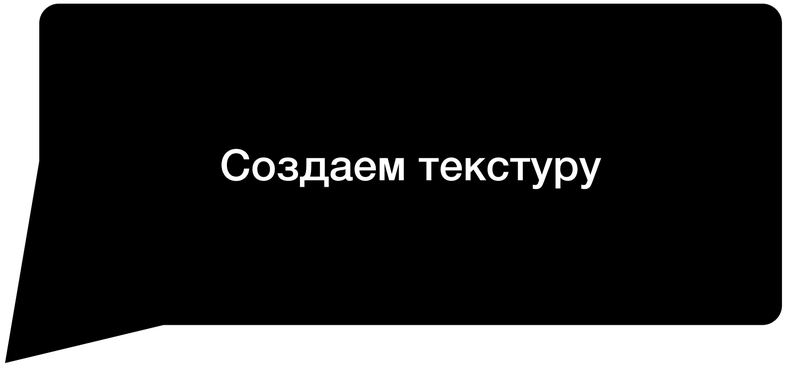
Устройство пайплайна

User input

Пройдемся по понятиям

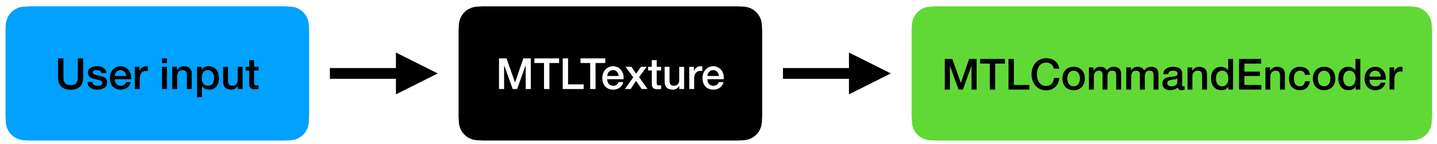
Устройство пайплайна





Пройдемся по понятиям

Устройство пайплайна



Отправляем текстуру в encoder

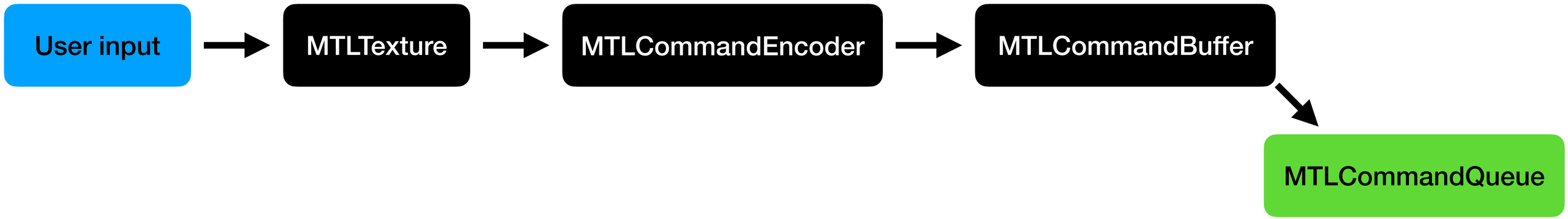
Пройдемся по понятиям

Устройство пайплайна

Вызываем commit() у буфера команд

Пройдемся по понятиям

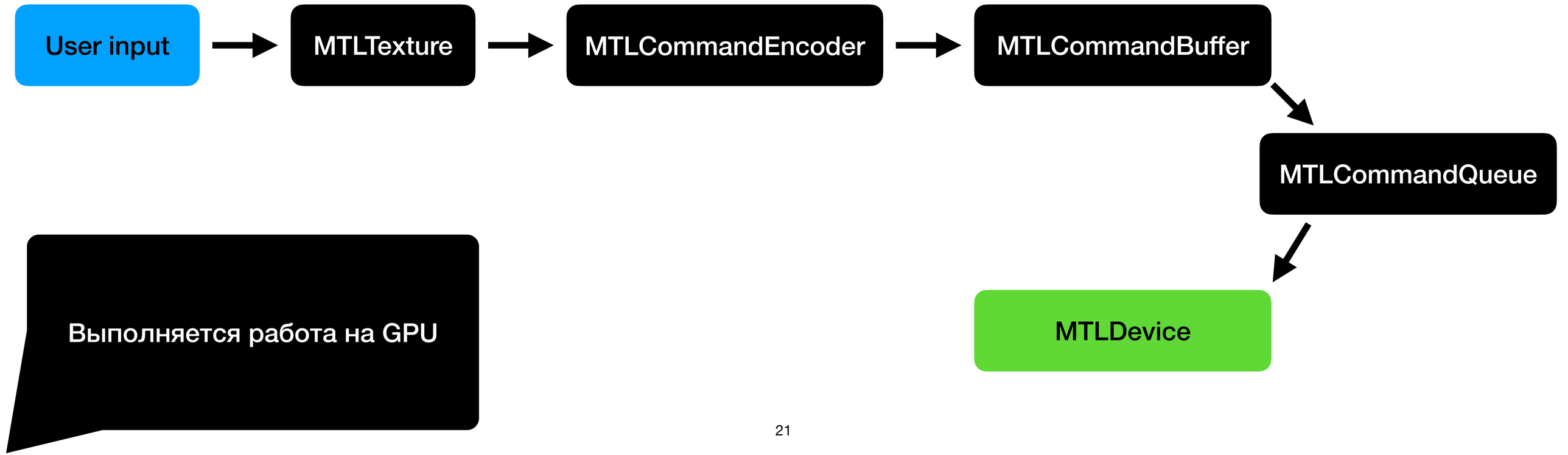
Устройство пайплайна



Буфер команд отправляется в очередь выполнения

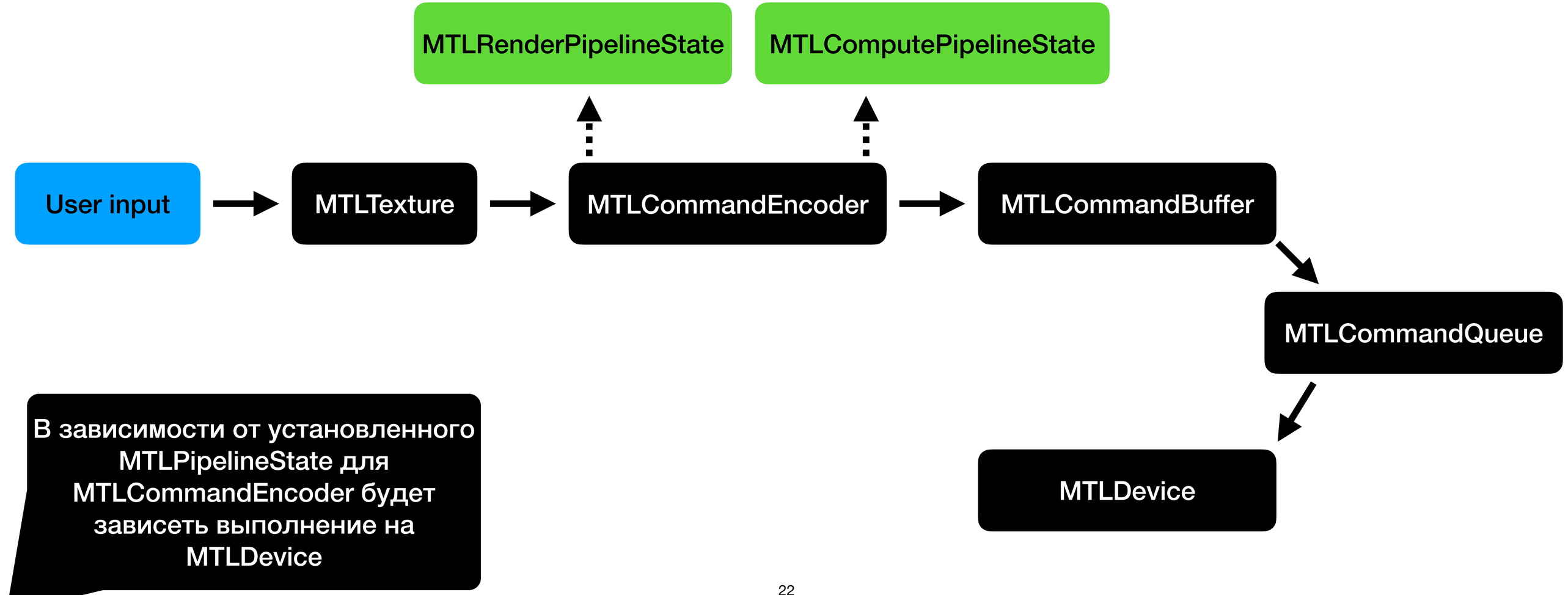
Пройдемся по понятиям

Устройство пайплайна



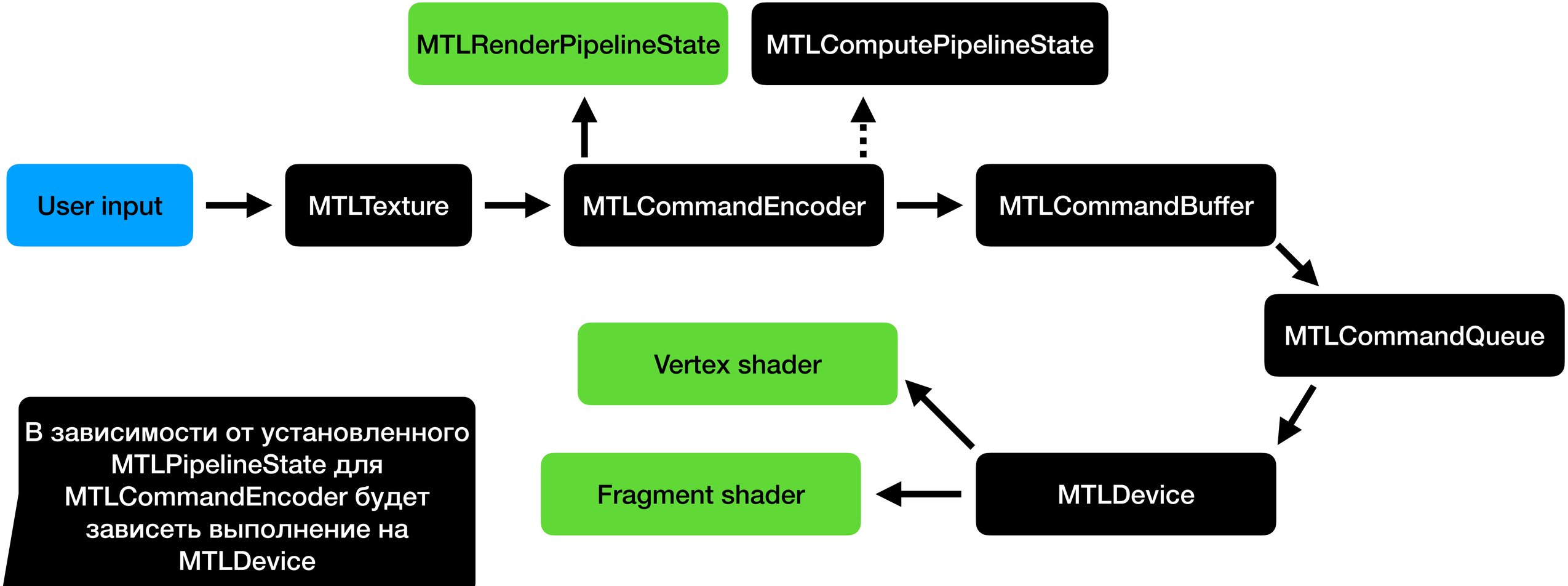
Пройдемся по понятиям

Устройство пайплайна



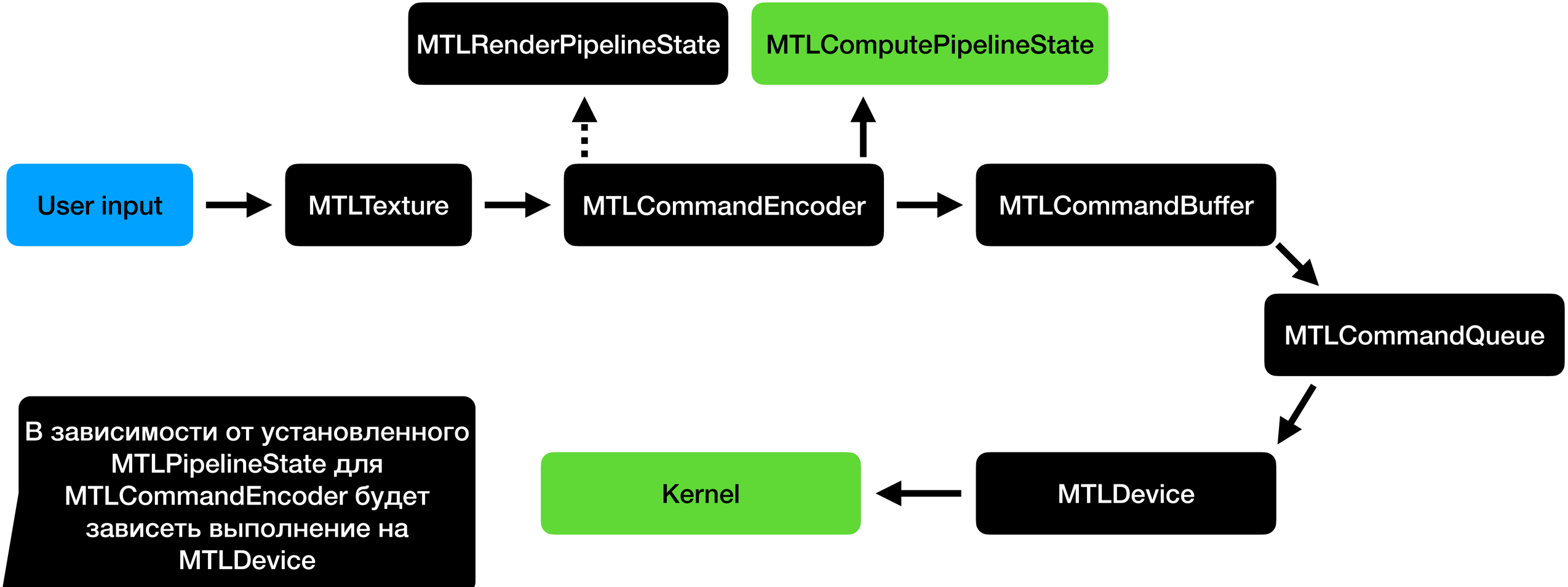
Пройдемся по понятиям

Устройство пайплайна



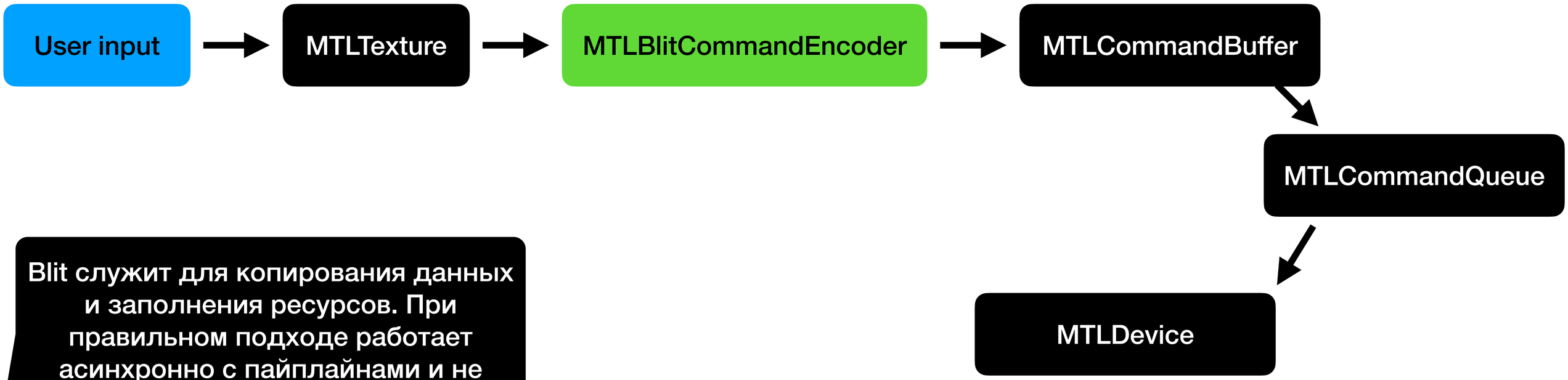
Пройдемся по понятиям

Устройство пайплайна



Пройдемся по понятиям

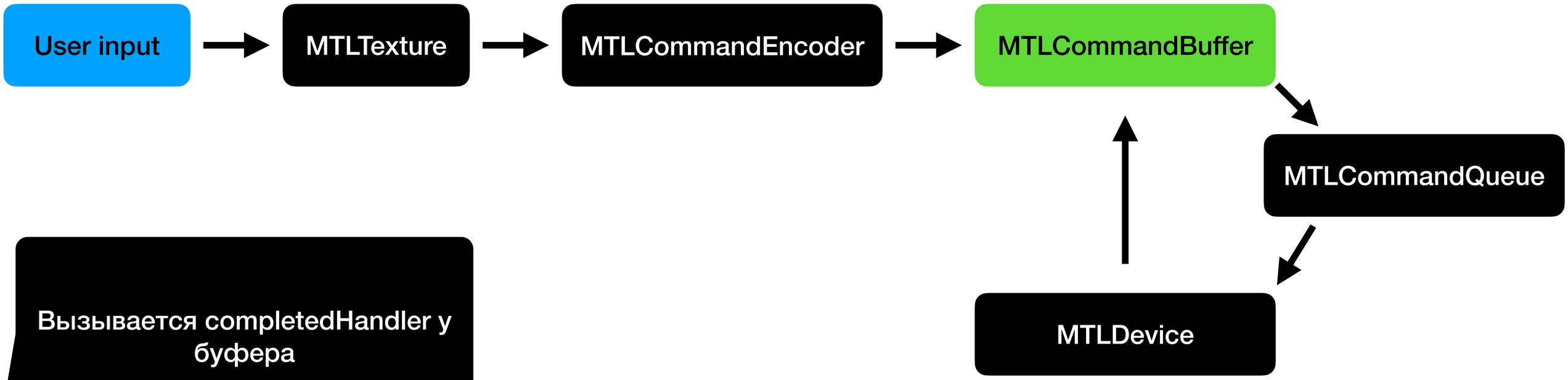
Устройство пайплайна



Blit служит для копирования данных и заполнения ресурсов. При правильном подходе работает асинхронно с пайплайнами и не блокирует работу

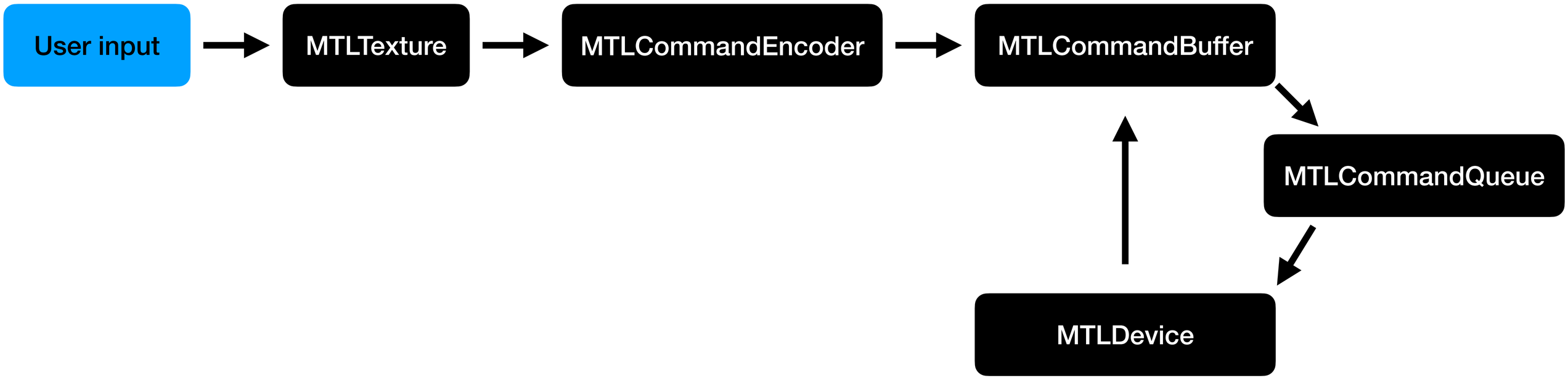
Пройдемся по понятиям

Устройство пайплайна



Пройдемся по понятиям

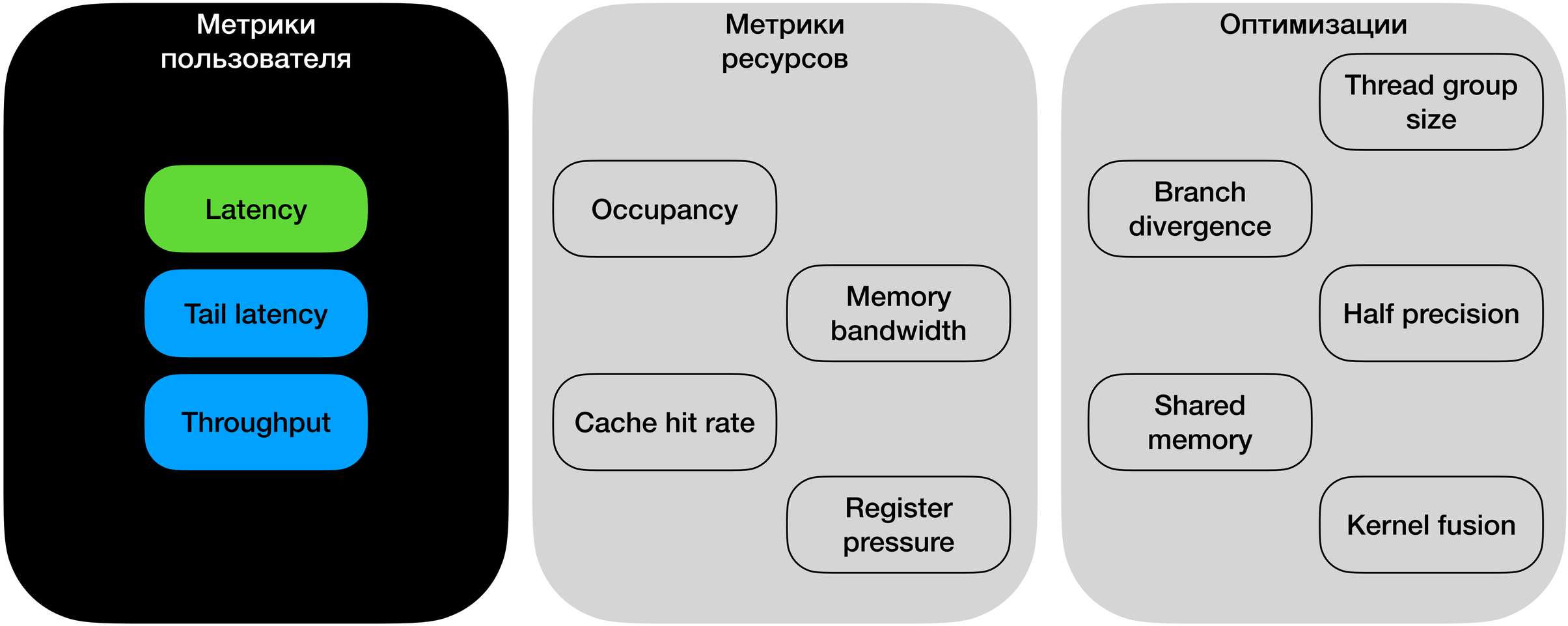
Устройство пайплайна



План доклада

- Вступление.
- Метрики пользователя.
- Метрики ресурсов и оптимизации.
- Какие еще скрываются тонкости?
- Подведение итогов.

Как измерить «производительность»?



Latency

Время от начала работы до получения результата

- Измерение способом **wall-clock**.

```
private func computeByCPU() {
    guard let cgImage = image?.data?.cgImage else {
        return
    }

    let startTime = CACurrentMediaTime()
    let resultImage = computeProcessor.processImage(
        cgImage,
        settings: RenderState.ComputeSettings(
            type: processingType,
            isOptimized: isOptimized
        )
    )
    self.image = ProcessingImage(
        id: UUID().uuidString,
        size: "CPU Processed Image",
        data: resultImage
    )
    let endTime = CACurrentMediaTime()
    let latency = endTime - startTime
}
```

Latency

Время от начала работы до получения результата

- Измерение способом **wall-clock**.


```
private func computeByCPU() {
    guard let cgImage = image?.data?.cgImage else {
        return
    }

    let startTime = CACurrentMediaTime()
    let resultImage = computeProcessor.processImage(
        cgImage,
        settings: RenderState.ComputeSettings(
            type: processingType,
            isOptimized: isOptimized
        )
    )
    self.image = ProcessingImage(
        id: UUID().uuidString,
        size: "CPU Processed Image",
        data: resultImage
    )
    let endTime = CACurrentMediaTime()
    let latency = endTime - startTime
}
```

Latency

Время от начала работы до получения результата

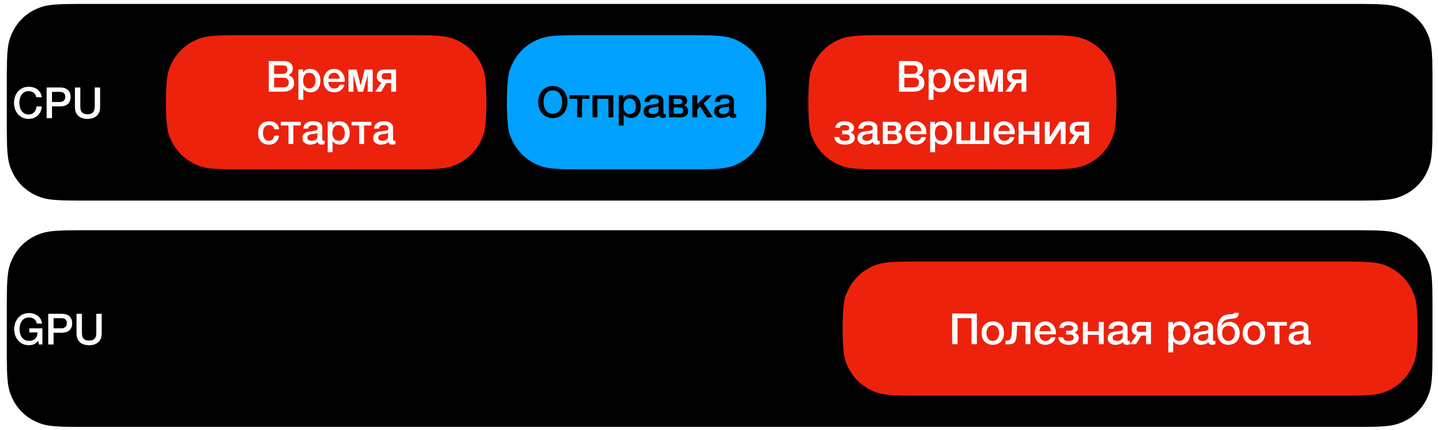
- Измерение способом **wall-clock**.

```
@MainActor func draw(texture: MTLTexture) {
    guard let drawable = layer?.nextDrawable(),
          let buffer = try? context.makeCommandBuffer(
            label: "Render Buffer"
          ) else {
        return
    }

    renderPipeline.encode(
        buffer: buffer,
        drawable: drawable,
        texture: texture
    )

    // Некорректное измерение
    let startTime = CACurrentMediaTime()
    buffer.present(drawable)
    buffer.commit()
    let endTime = CACurrentMediaTime()
    let incorrectLatency = endTime - startTime
}
```

Latency

Время от начала работы до получения результата

- Измерение способом **wall-clock**.




```
@MainActor func draw(texture: MTLTexture) {
    guard let drawable = layer?.nextDrawable(),
          let buffer = try? context.makeCommandBuffer(
            label: "Render Buffer"
          ) else {
        return
    }

    renderPipeline.encode(
        buffer: buffer,
        drawable: drawable,
        texture: texture
    )

    // Некорректное измерение
    let startTime = CACurrentMediaTime()
    buffer.present(drawable)
    buffer.commit()
    let endTime = CACurrentMediaTime()
    let incorrectLatency = endTime - startTime
}
```

Latency

Время от начала работы до получения результата

- Измерение способом wall-clock.
- Измерение **GPU latency** с помощью свойств **MTLCommandBuffer**.

```
@MainActor func draw(texture: MTLTexture) {
    guard let drawable = layer?.nextDrawable(),
          let buffer = try? context.makeCommandBuffer(
            label: "Render Buffer"
          ) else {
        return
    }

    renderPipeline.encode(
        buffer: buffer,
        drawable: drawable,
        texture: texture
    )

    buffer.present(drawable)
    buffer.addCompletedHandler { completedBuffer in
        // Измеряем latency выполнения кода на GPU
        let gpuLatency = completedBuffer.gpuEndTime - completedBuffer.gpuStartTime
    }
    buffer.commit()
    let endTime = CACurrentMediaTime()
}
```

Latency

Время от начала работы до получения результата

- Измерение способом wall-clock.
- Измерение **GPU latency** с помощью свойств **MTLCommandBuffer**.

```
@MainActor func draw(texture: MTLTexture) {
    guard let drawable = layer?.nextDrawable(),
          let buffer = try? context.makeCommandBuffer(
            label: "Render Buffer"
          ) else {
        return
    }

    renderPipeline.encode(
        buffer: buffer,
        drawable: drawable,
        texture: texture
    )

    buffer.present(drawable)
    buffer.addCompletedHandler { completedBuffer in
        // Измеряем latency выполнения кода на GPU
        let gpuLatency = completedBuffer.gpuEndTime - completedBuffer.gpuStartTime
    }
    buffer.commit()
    let endTime = CACurrentMediaTime()
}
```

Latency

Время от начала работы до получения результата

- Измерение способом wall-clock.
- Измерение **kernel latency** с помощью свойств **MTLCommandBuffer**.

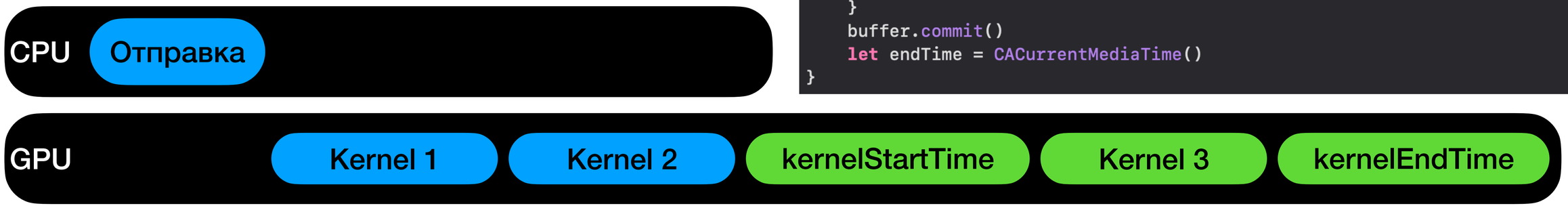
```
@MainActor func draw(texture: MTLTexture) {
    guard let drawable = layer?.nextDrawable(),
          let buffer = try? context.makeCommandBuffer(
            label: "Render Buffer"
          ) else {
        return
    }

    renderPipeline.encode(
        buffer: buffer,
        drawable: drawable,
        texture: texture
    )

    buffer.present(drawable)
    buffer.addCompletedHandler { completedBuffer in
        // Измеряем latency выполнения kernel-а
        // При этом учитывается, что нужно наличие MTLComputePipelineState
        let gpuLatency = completedBuffer.kernelEndTime - completedBuffer.kernelStartTime
    }
    buffer.commit()
    let endTime = CACurrentMediaTime()
}
```

Latency

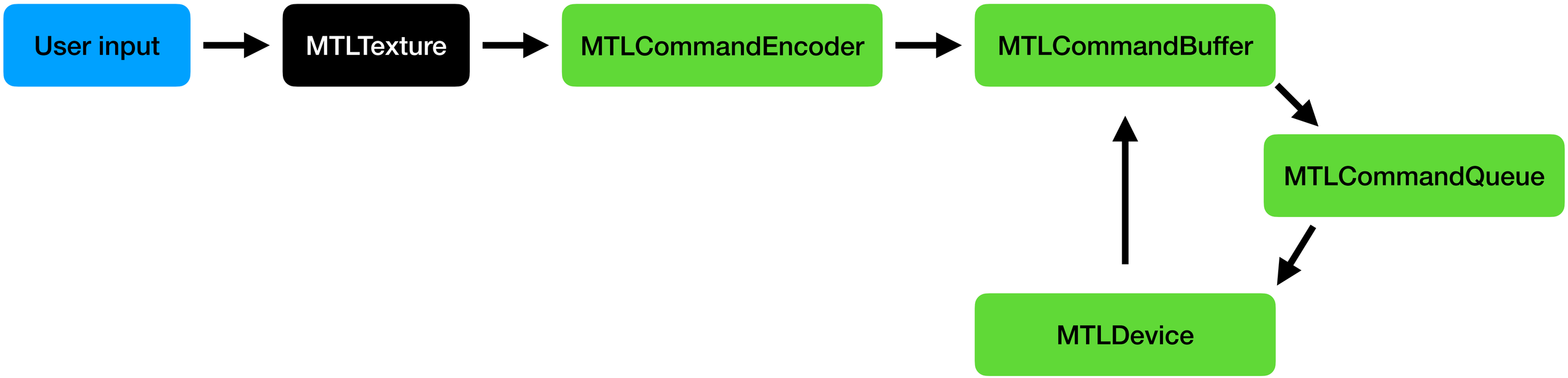
Время от начала работы до получения результата

- Измерение способом wall-clock.
- Измерение **kernel latency** с помощью свойств **MTLCommandBuffer**.

```
@MainActor func draw(texture: MTLTexture) {
    guard let drawable = layer?.nextDrawable(),
          let buffer = try? context.makeCommandBuffer(
            label: "Render Buffer"
          ) else {
        return
    }

    renderPipeline.encode(
        buffer: buffer,
        drawable: drawable,
        texture: texture
    )

    buffer.present(drawable)
    buffer.addCompletedHandler { completedBuffer in
        // Измеряем latency выполнения kernel-а
        // При этом учитывается, что нужно наличие MTLComputePipelineState
        let gpuLatency = completedBuffer.kernelEndTime - completedBuffer.kernelStartTime
    }
    buffer.commit()
    let endTime = CACurrentMediaTime()
}
```



Выбираем область измерения



Latency

Время от начала работы до получения результата

- Измерение способом wall-clock.
- Измерение latency с помощью свойств MTLCommandBuffer.
- Важен **контекст** того, что именно мы хотим посчитать.

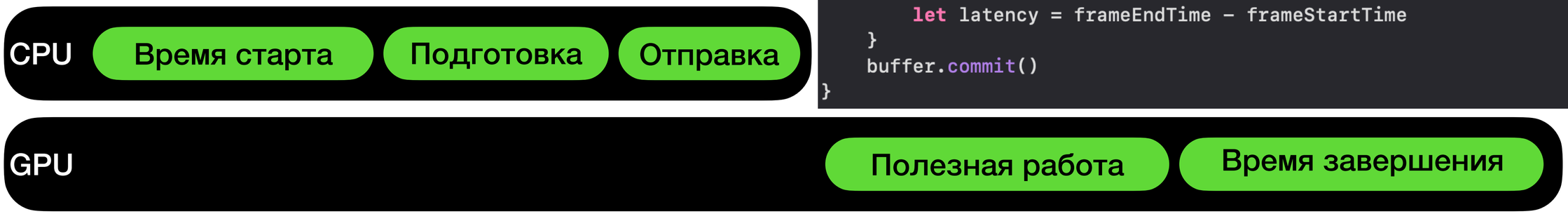
```
/// Draw texture by Metal
/// - Parameter texture: Texture to drawing
@MainActor func draw(texture: MTLTexture) {
    let frameStartTime = CACurrentMediaTime()
    guard let drawable = layer?.nextDrawable(),
          let buffer = try? context.makeCommandBuffer(label: "Render Buffer") else {
        return
    }

    renderPipeline.encode(
        buffer: buffer,
        drawable: drawable,
        texture: texture
    )

    buffer.present(drawable)
    buffer.addCompletedHandler { _ in
        let frameEndTime = CACurrentMediaTime()
        let latency = frameEndTime - frameStartTime
    }
    buffer.commit()
}
```

Latency

Время от начала работы до получения результата

- Измерение способом wall-clock.
- Измерение latency с помощью свойств MTLCommandBuffer.
- Важен **контекст** того, что именно мы хотим посчитать.

```swift
/// Draw texture by Metal
/// - Parameter texture: Texture to drawing
@MainActor func draw(texture: MTLTexture) {
    let frameStartTime = CACurrentMediaTime()
    guard let drawable = layer?.nextDrawable(),
          let buffer = try? context.makeCommandBuffer(label: "Render Buffer") else {
        return
    }

    renderPipeline.encode(
        buffer: buffer,
        drawable: drawable,
        texture: texture
    )

    buffer.present(drawable)
    buffer.addCompletedHandler { _ in
        let frameEndTime = CACurrentMediaTime()
        let latency = frameEndTime - frameStartTime
    }
    buffer.commit()
}
```

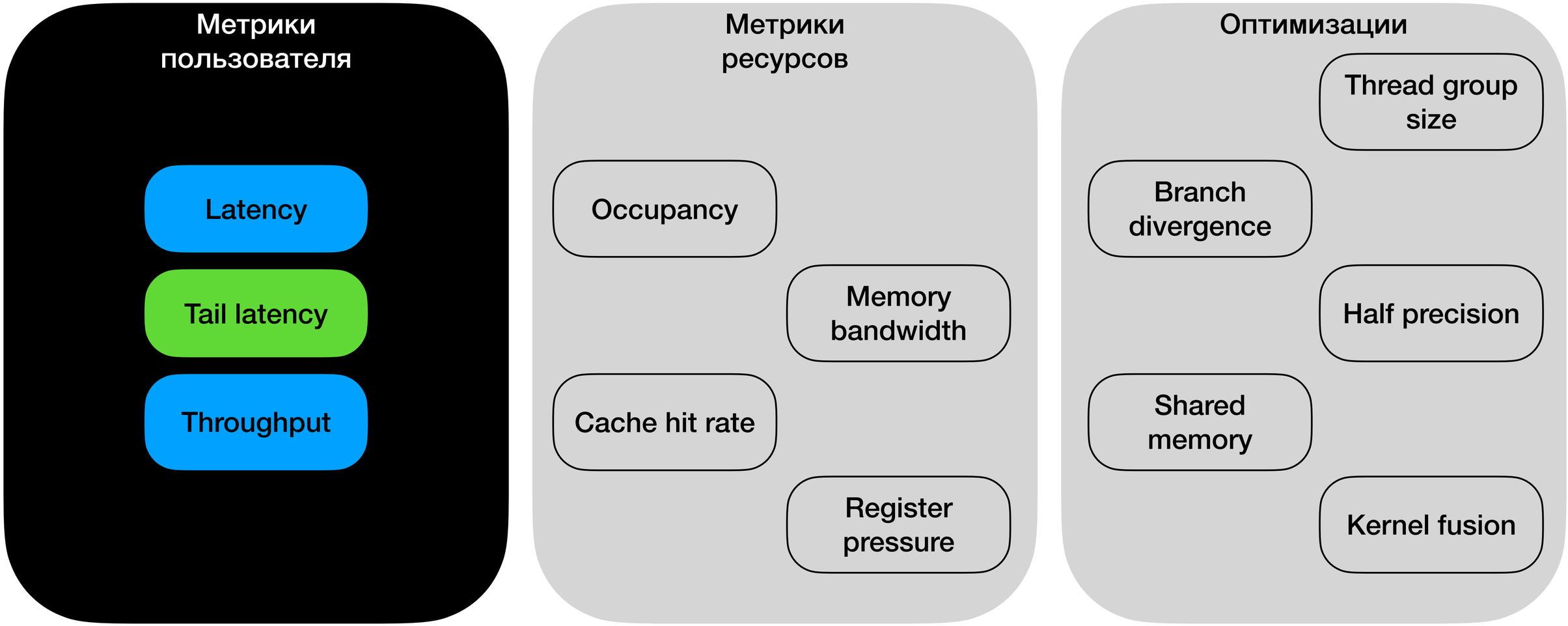


Latency

Время от начала работы до получения результата

- Измерение способом wall-clock.
- Измерение latency с помощью свойств MTLCommandBuffer.
- Важен контекст того, что именно мы хотим посчитать.
- Чем **ниже значение** данной метрики, тем быстрее выполняется **полезная работа**.

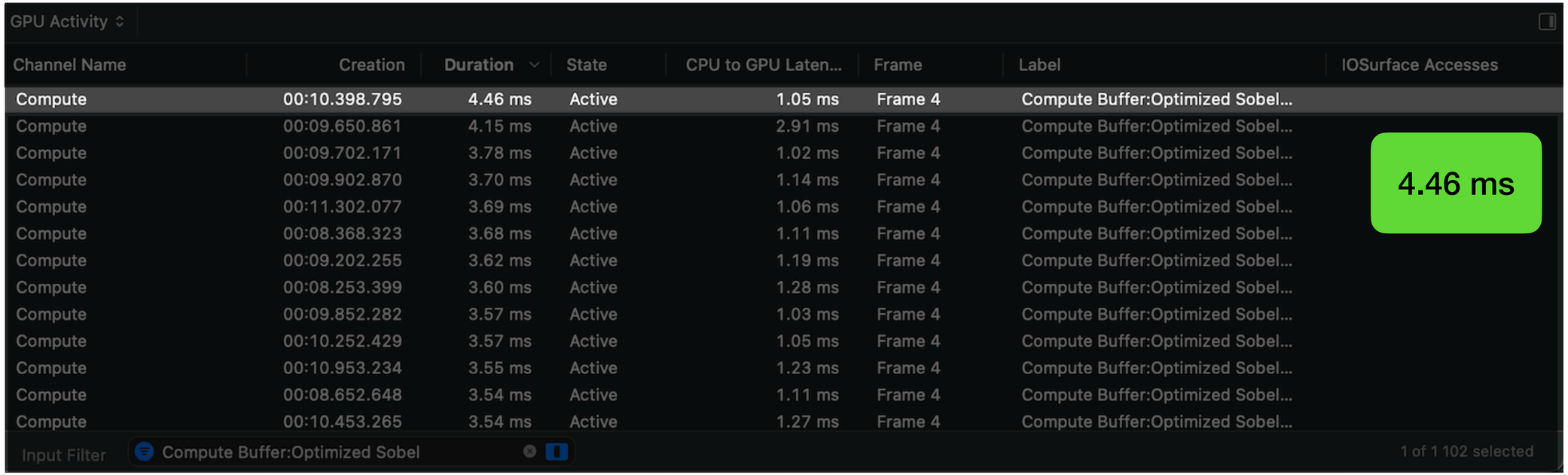
Как измерить «производительность»?



Tail latency

Насколько медленные худшие случаи выполнения

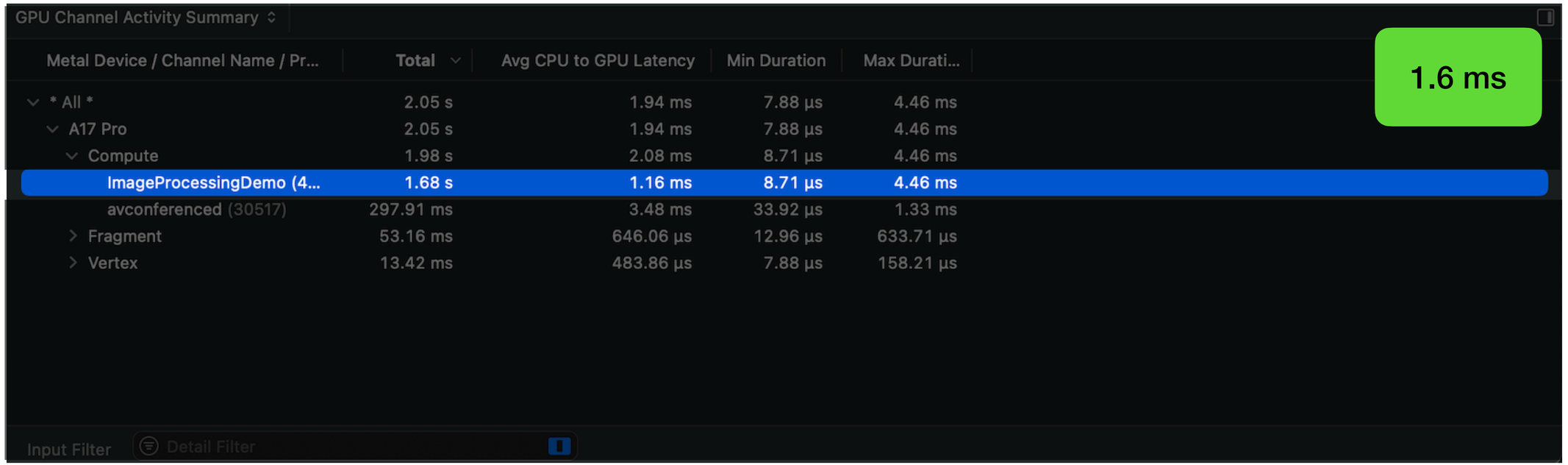
- **Максимальная** latency - **шумная** метрика.



Tail latency

Насколько медленные худшие случаи выполнения

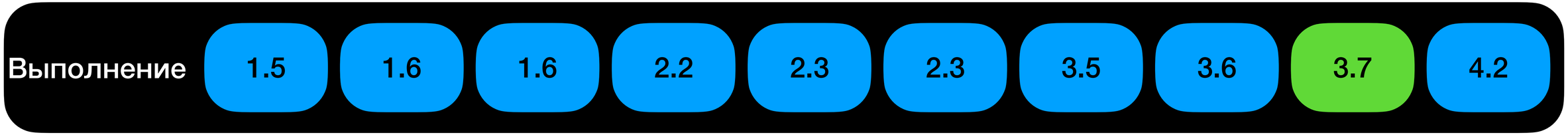
- Максимальная latency - шумная метрика.
- **Средняя** latency - не показывает проблемы.



Tail latency

Насколько медленные худшие случаи выполнения

- Максимальная latency - шумная метрика.
- Средняя latency - не показывает проблемы.
- Percentiles: **p50**, **p90**, **p95**, **p99**.



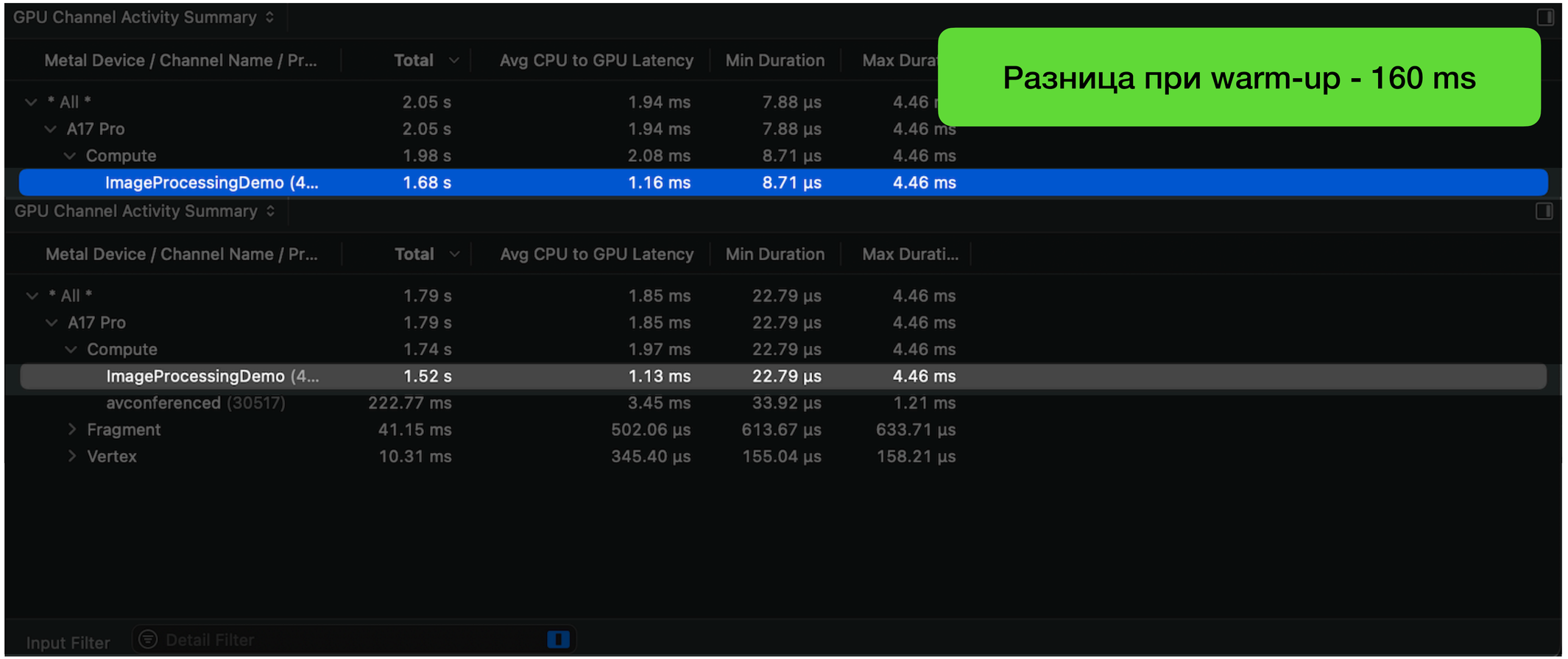
Tail latency

Насколько медленные худшие случаи выполнения

- Максимальная latency - шумная метрика.
- Средняя latency - не показывает проблемы.
- Percentiles: p50, p90, p95, p99
- **Warm-up** и реальная рабочая стабильность.

Tail latency

Насколько медленные худшие случаи выполнения

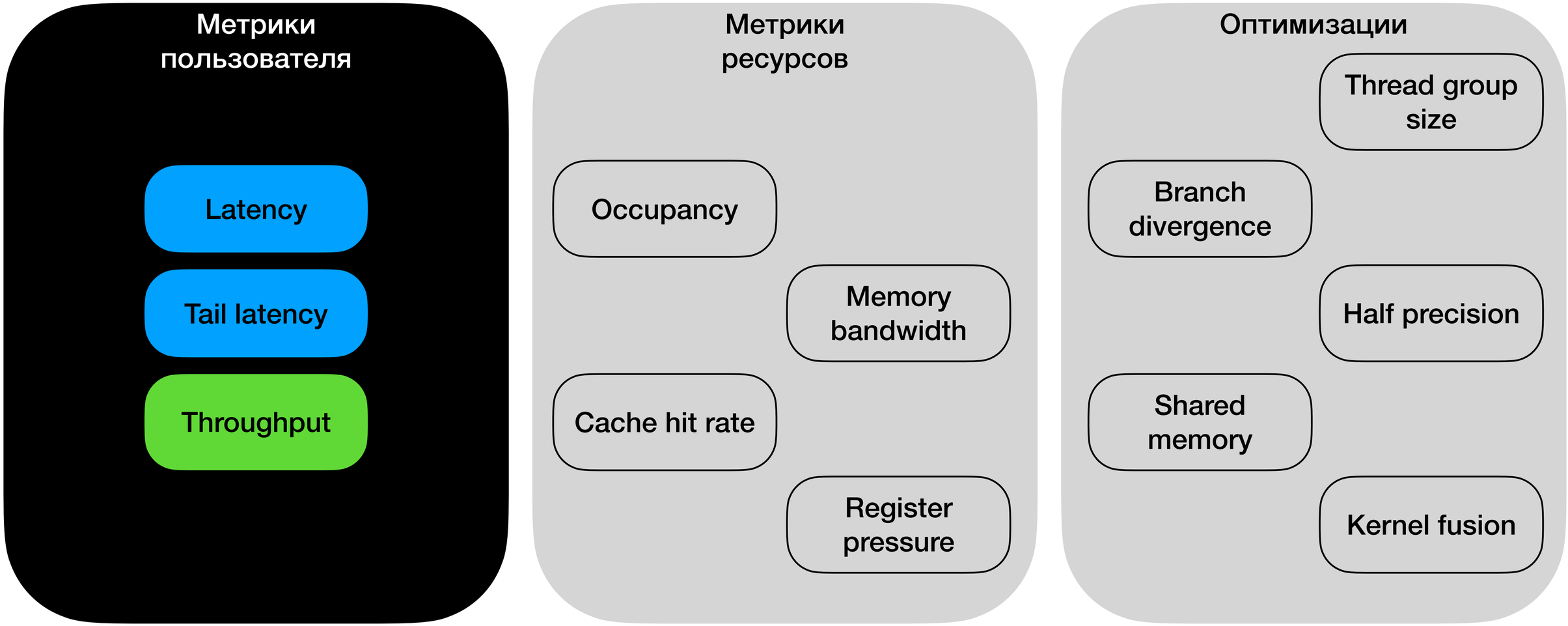


Tail latency

Насколько медленные худшие случаи выполнения

- Максимальная latency - шумная метрика.
- Средняя latency - не показывает проблемы.
- Percentiles: p50, p90, p95, p99
- Warm-up и реальная рабочая стабильность.
- Чем **ниже значение** данной метрики, тем **меньше** у нас **рывков** и еле заметных **торможений**.

Как измерить «производительность»?



Throughput

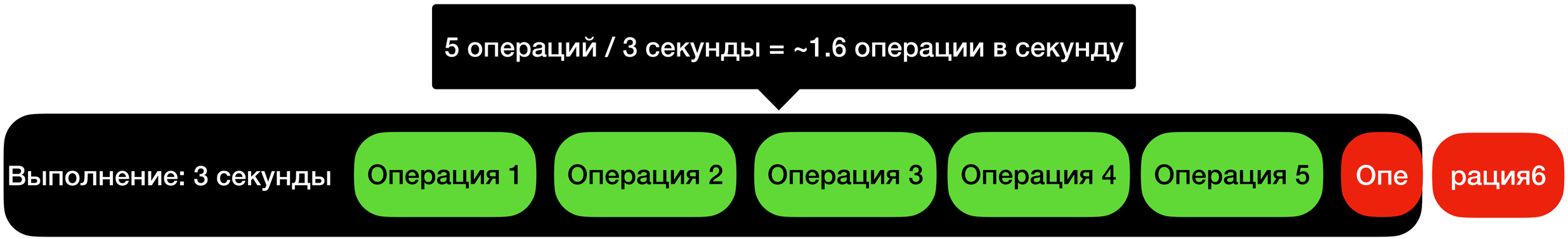
Сколько работы выполняется за единицу времени

- **Throughput** и **latency** говорят нам о разном.

Throughput

Сколько работы выполняется за единицу времени

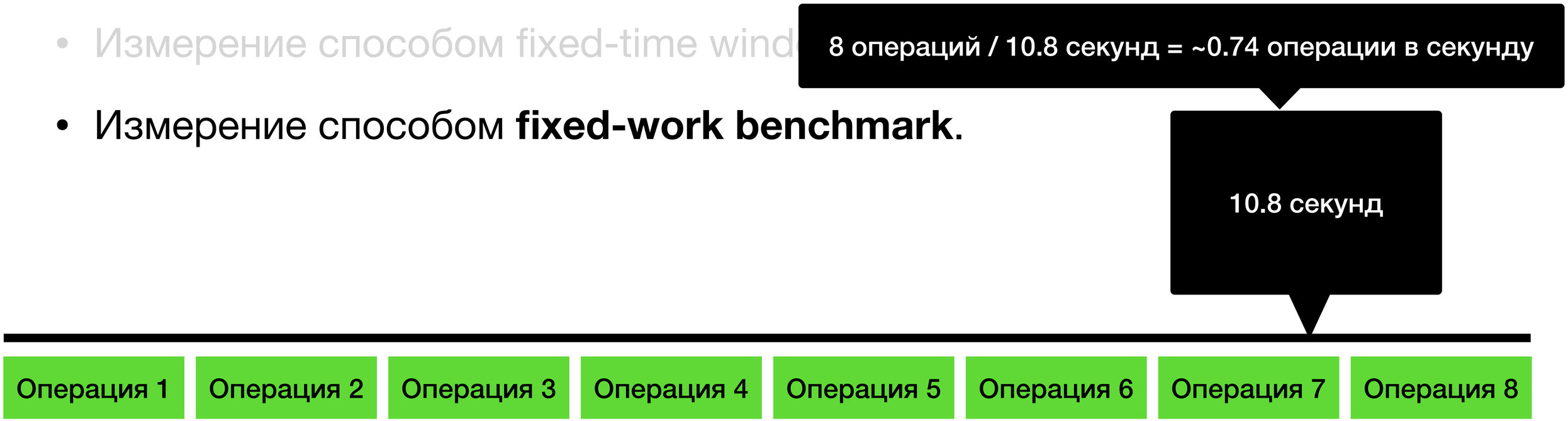
- Throughput и latency говорят нам о разном.
- Измерение способом **fixed-time window**.



Throughput

Сколько работы выполняется за единицу времени

- Throughput и latency говорят нам о разном.
- Измерение способом fixed-time wind
- Измерение способом **fixed-work benchmark**.



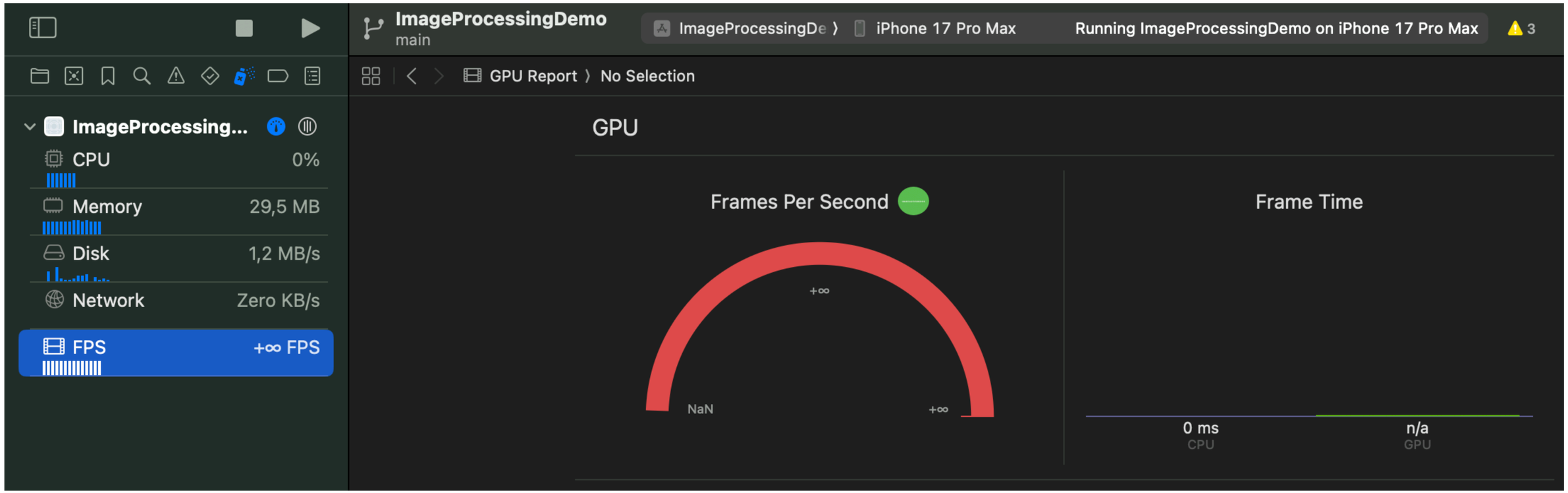
Throughput

Сколько работы выполняется за единицу времени

- Throughput и latency говорят нам о разном.
- Измерение способом fixed-time window.
- Измерение способом fixed-work benchmark.
- Первичная диагностика и profiling.

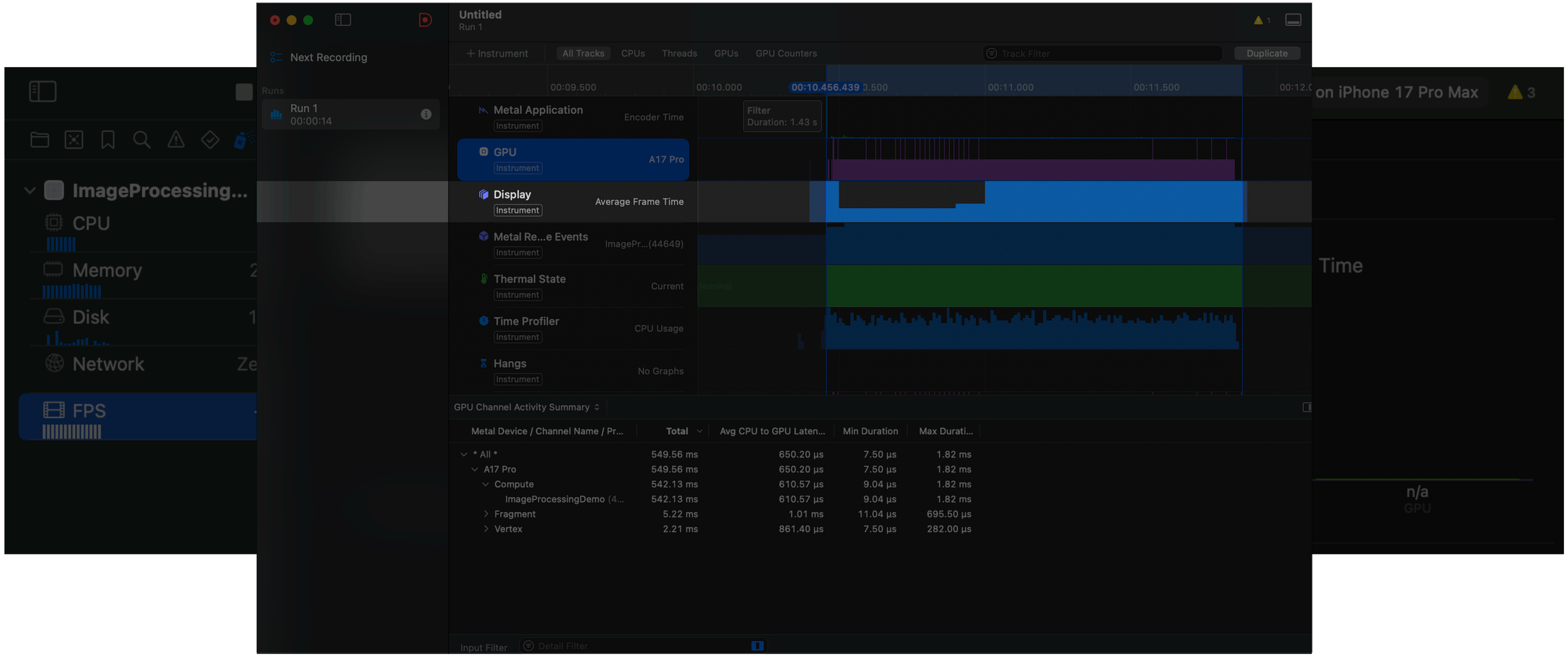
Throughput

Сколько работы выполняется за единицу времени



Throughput

Сколько работы выполняется за единицу времени



Throughput

Сколько работы выполняется за единицу времени

- Throughput и latency говорят нам о разном.
- Измерение способом fixed-time window.
- Измерение способом fixed-work benchmark.
- Первичная диагностика и profiling.
- Чем **выше значение** данной метрики, тем **быстрее** происходит **отображение** или **вычисления**.

План доклада

- Вступление.
- Метрики пользователя.
- Метрики ресурсов и оптимизации.
- Какие еще скрываются тонкости?
- Подведение итогов.

Как измерить «производительность»?

Метрики пользователя

- Latency
- Tail latency
- Throughput

Метрики ресурсов

Это то, что показывает нам железо

- Occupancy
- Memory bandwidth
- Cache hit rate
- Register pressure

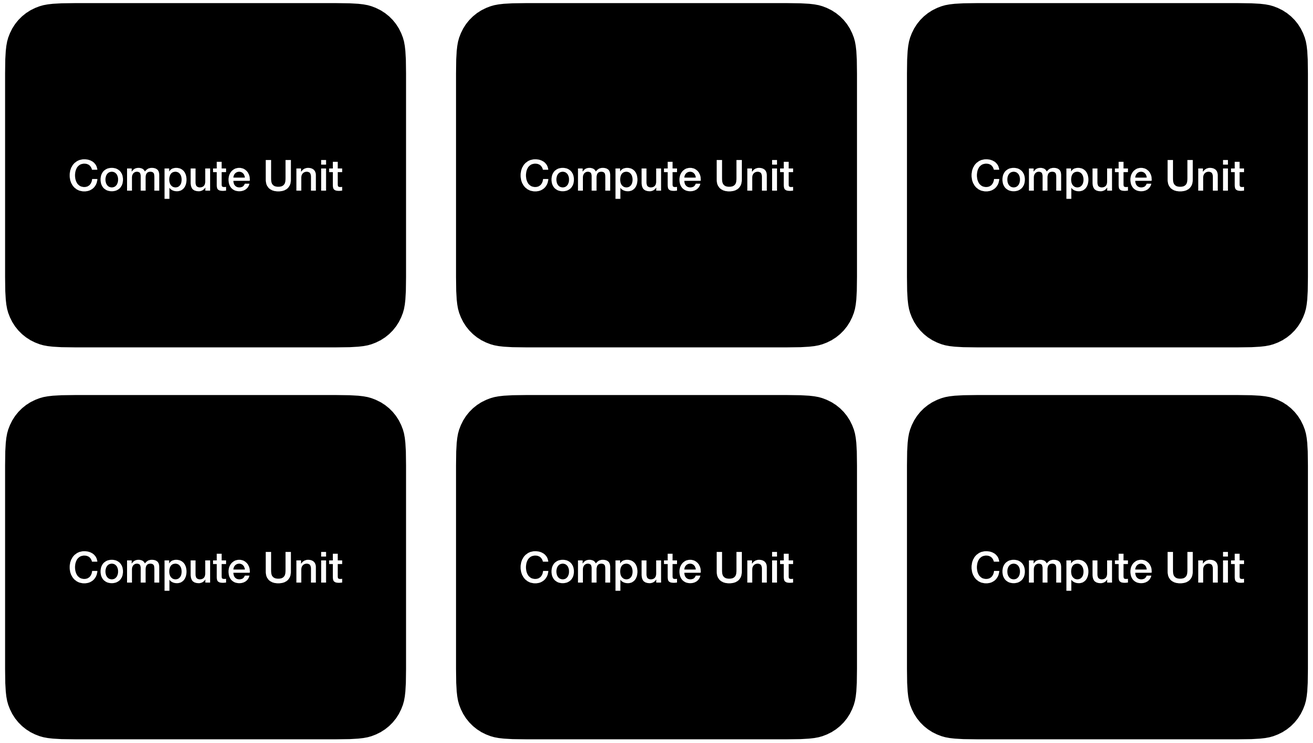
Оптимизации

- Thread group size
- Branch divergence
- Half precision
- Shared memory
- Kernel fusion

Пройдемся по понятиям

Основы работы GPU

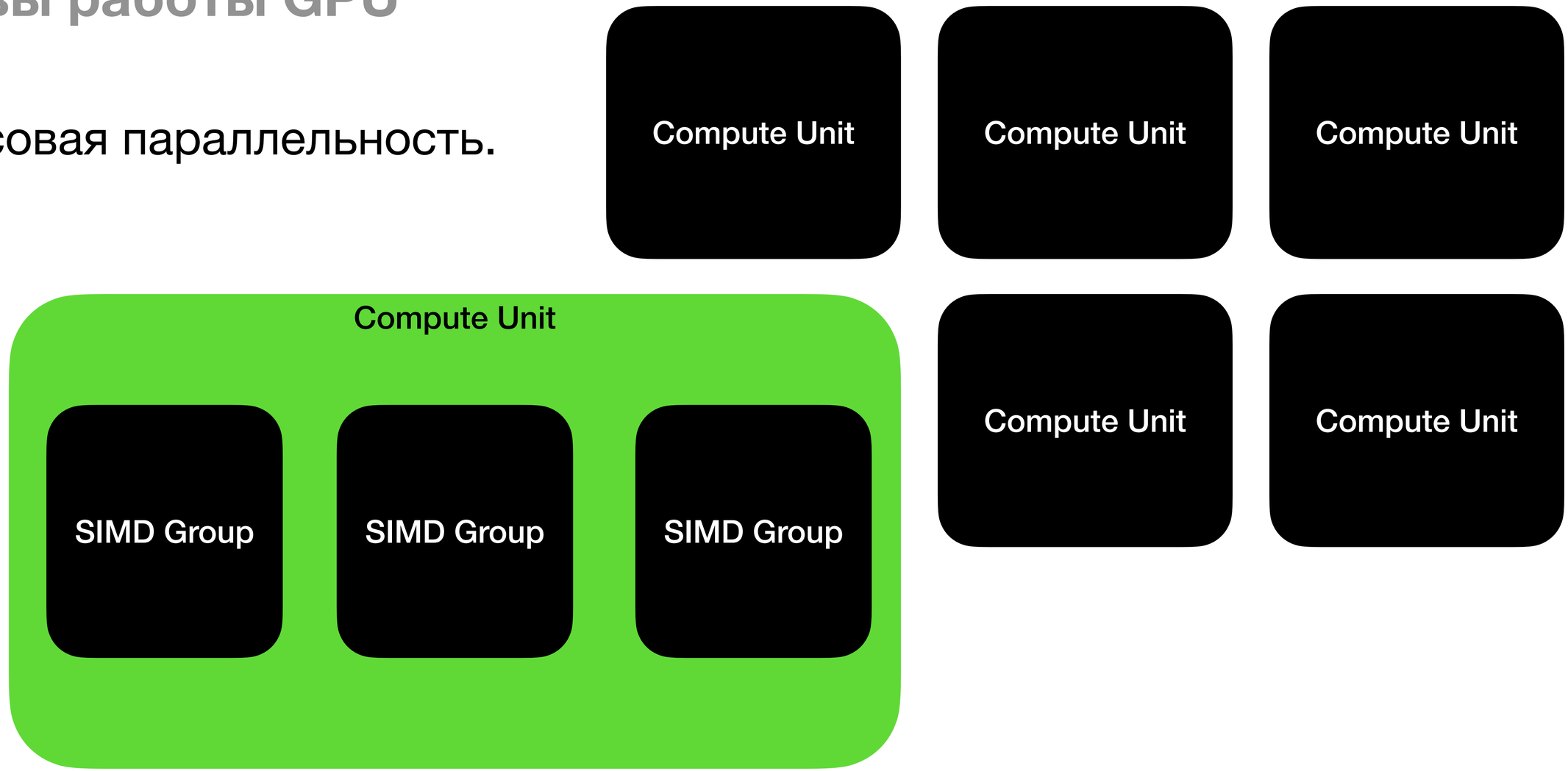
- Массовая параллельность.



Пройдемся по понятиям

Основы работы GPU

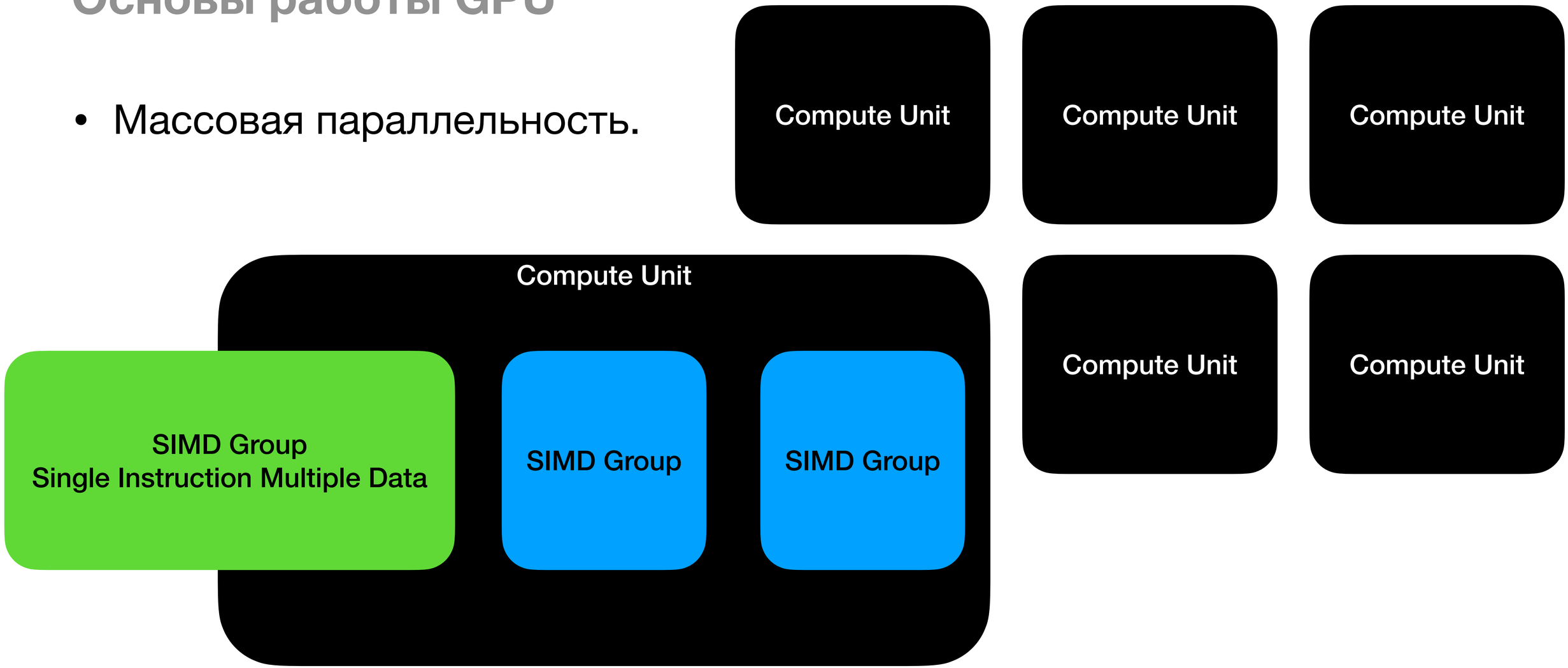
- Массовая параллельность.



Пройдемся по понятиям

Основы работы GPU

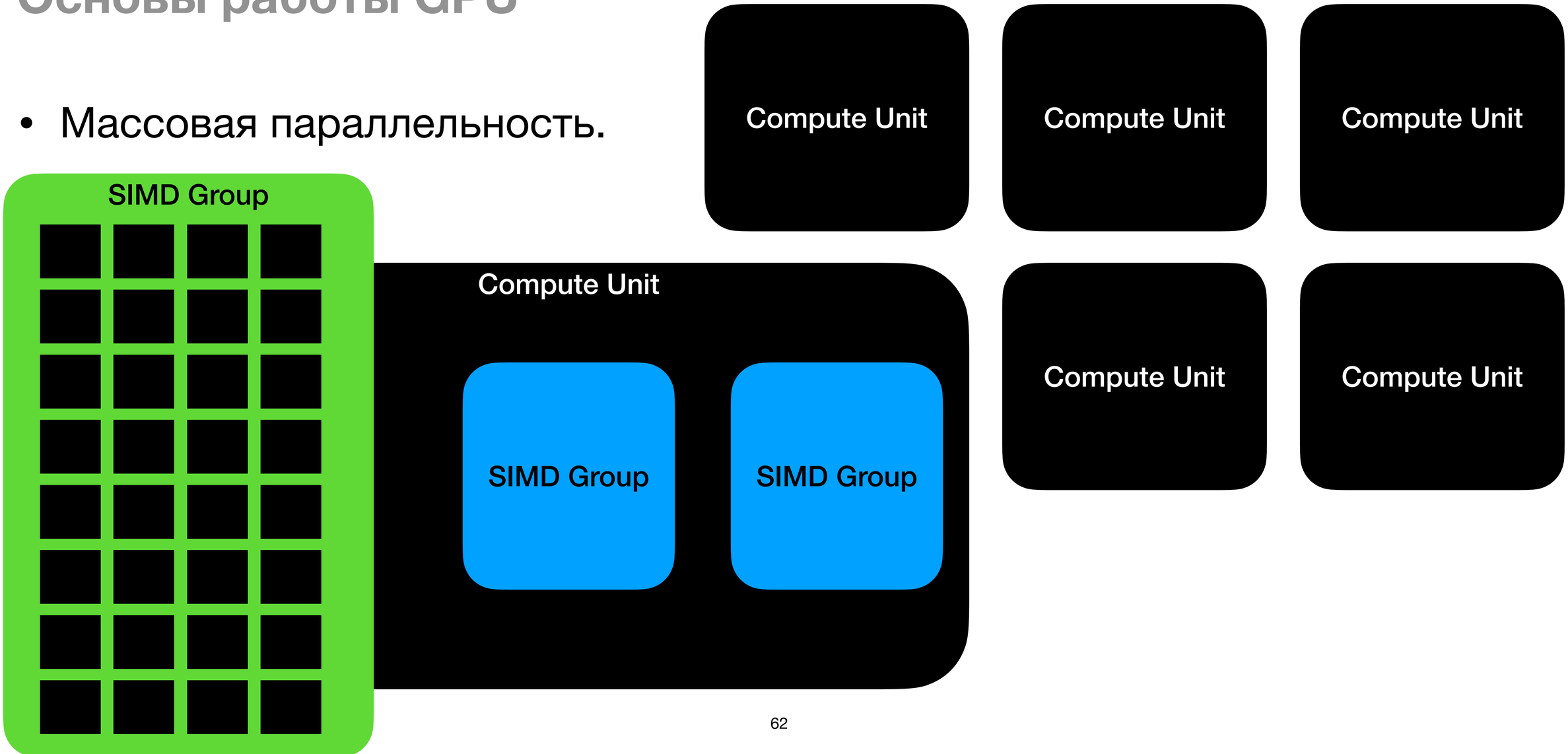
- Массовая параллельность.



Пройдемся по понятиям

Основы работы GPU

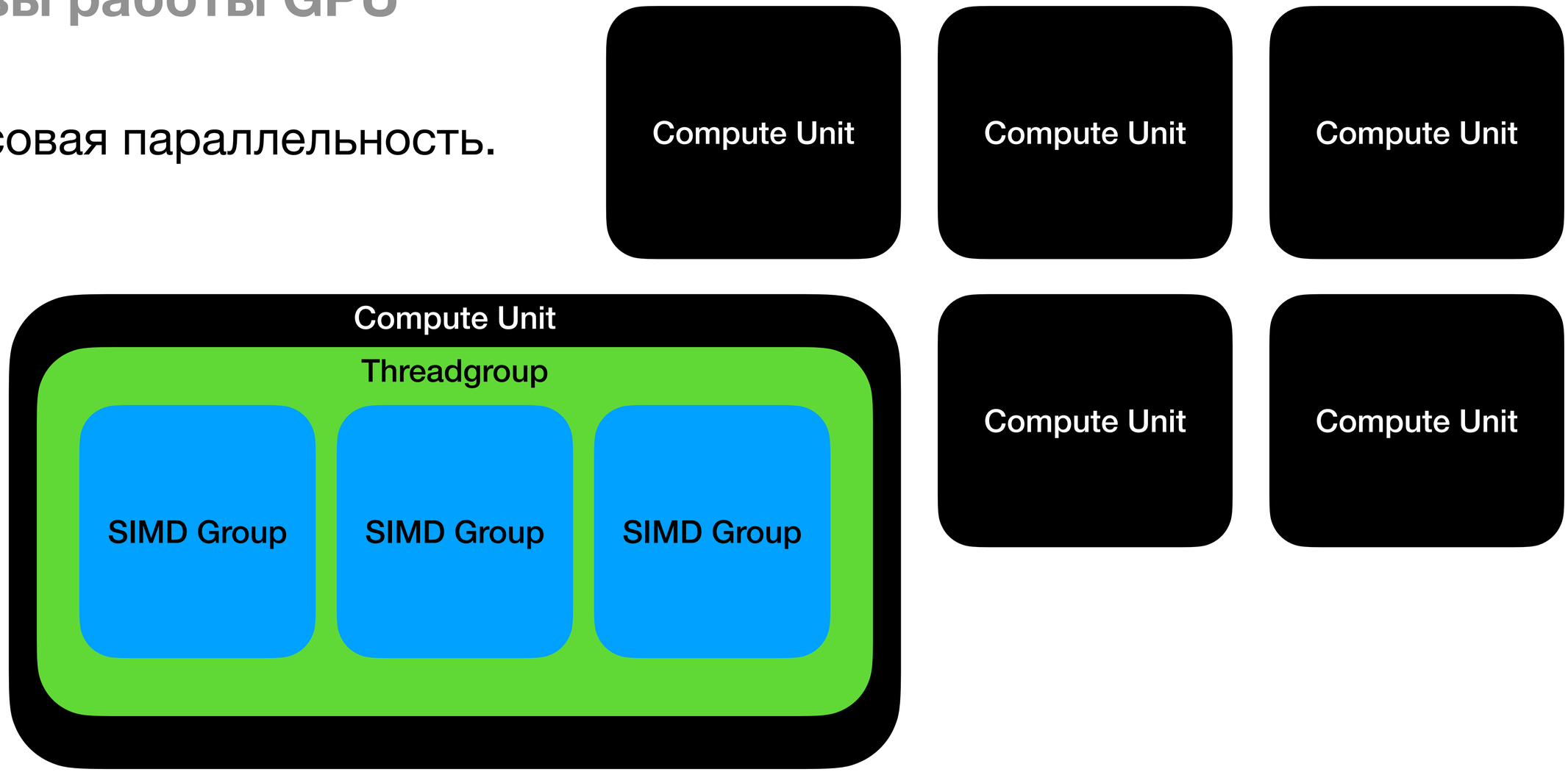
- Массовая параллельность.



Пройдемся по понятиям

Основы работы GPU

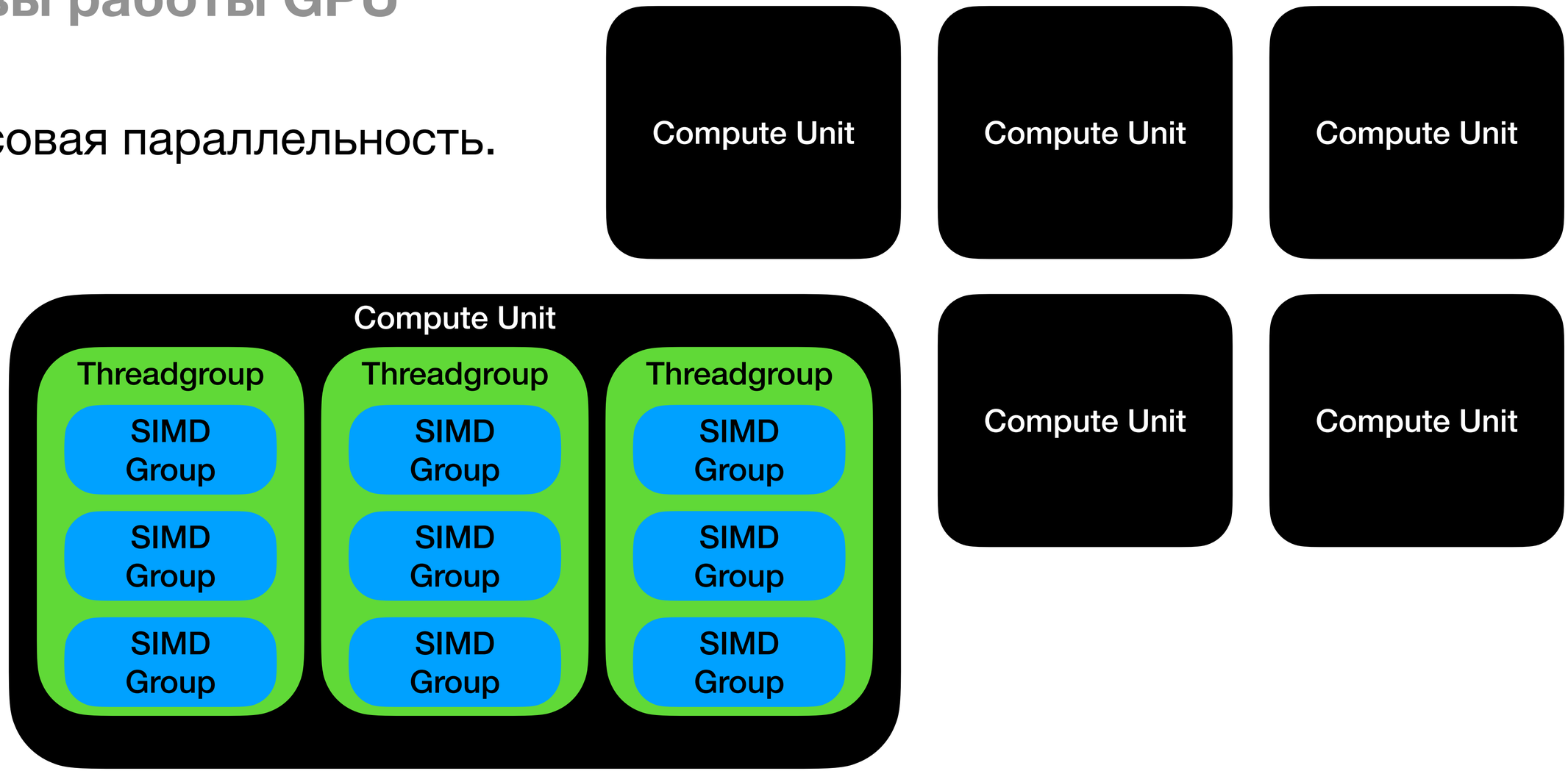
- Массовая параллельность.



Пройдемся по понятиям

Основы работы GPU

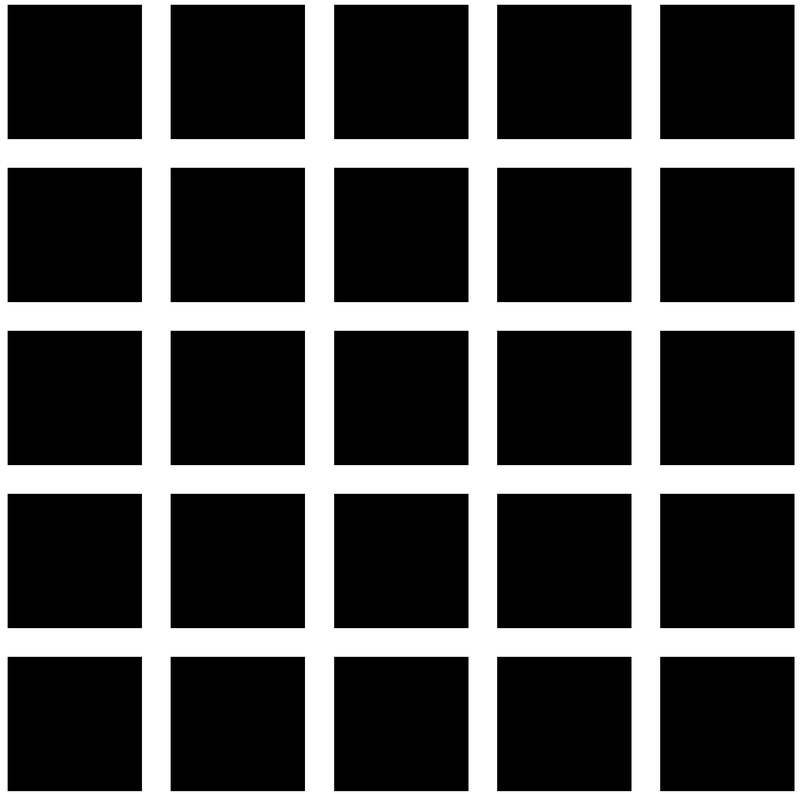
- Массовая параллельность.



Пройдемся по понятиям

Основы работы GPU

- Массовая параллельность.
- Собственная память.



Global memory

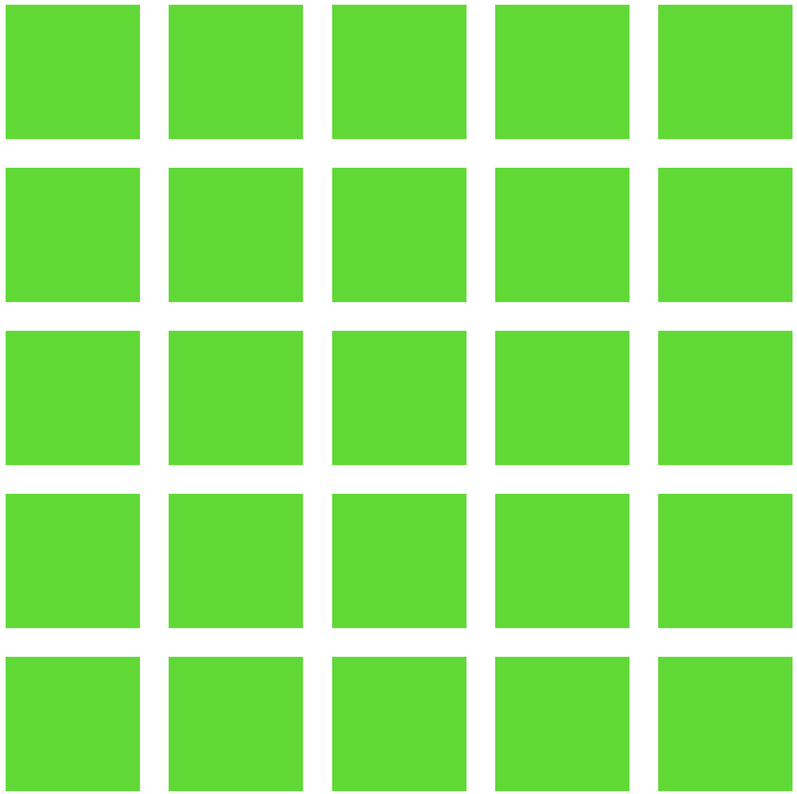
L1/L2 cache

Shared memory

Registers

Пройдемся по понятиям

Основы работы GPU



Global memory

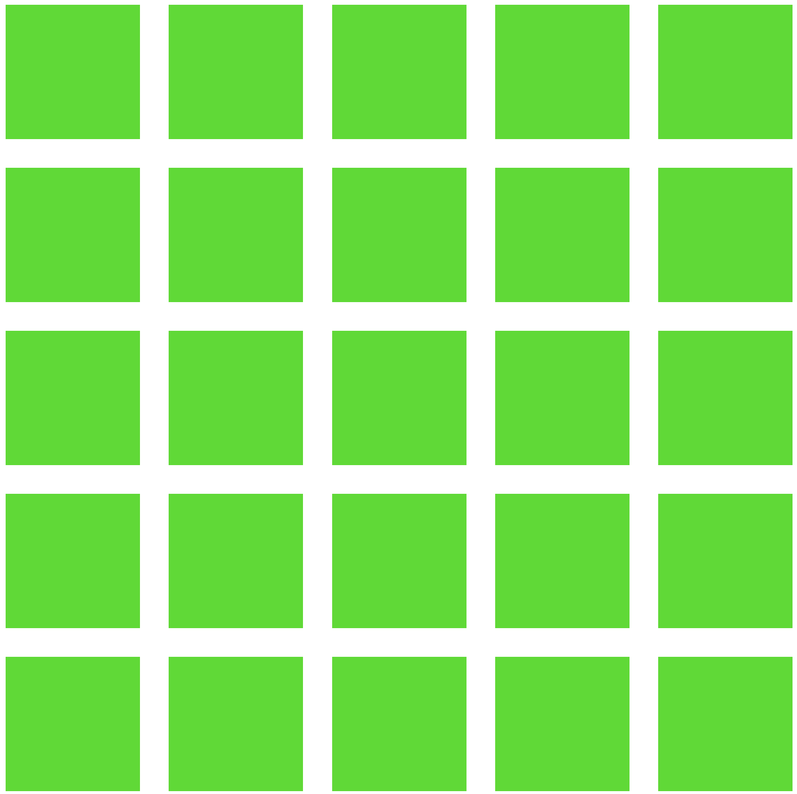
L1/L2 cache

Shared memory

Registers

Пройдемся по понятиям

Основы работы GPU



Global memory

Скорость доступа низкая

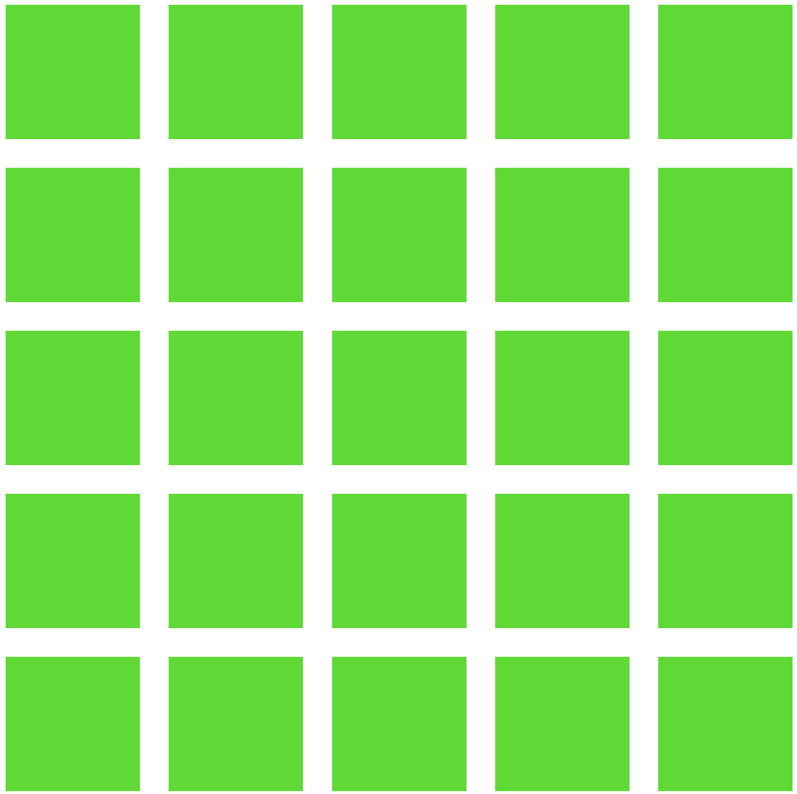
L1/L2 cache

Shared memory

Registers

Пройдемся по понятиям

Основы работы GPU



Global memory

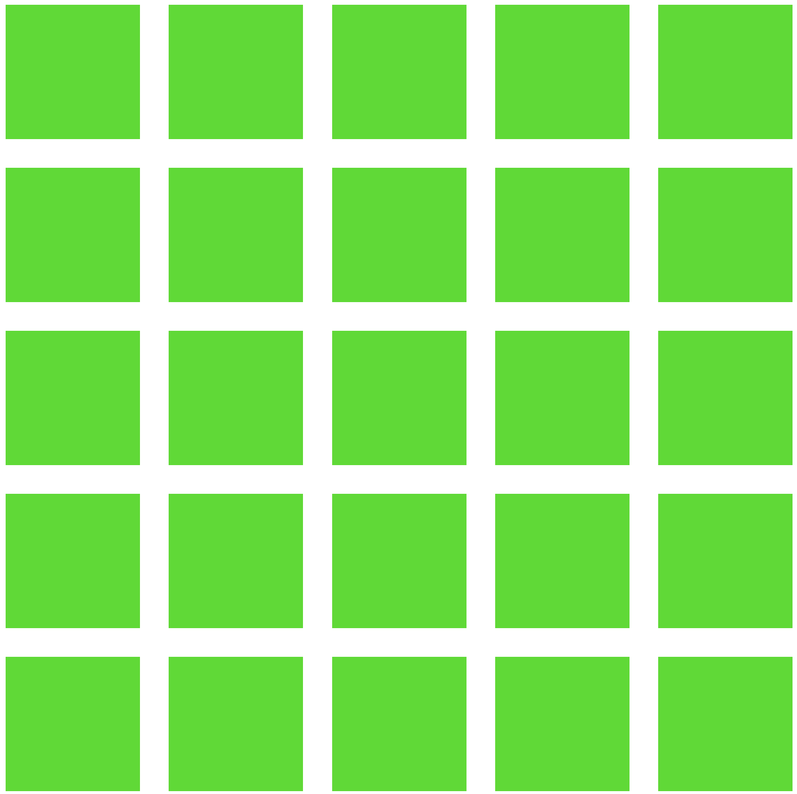
L1/L2 cache

Shared memory

Registers

Пройдемся по понятиям

Основы работы GPU



L1/L2 cache

Скорость доступа выше

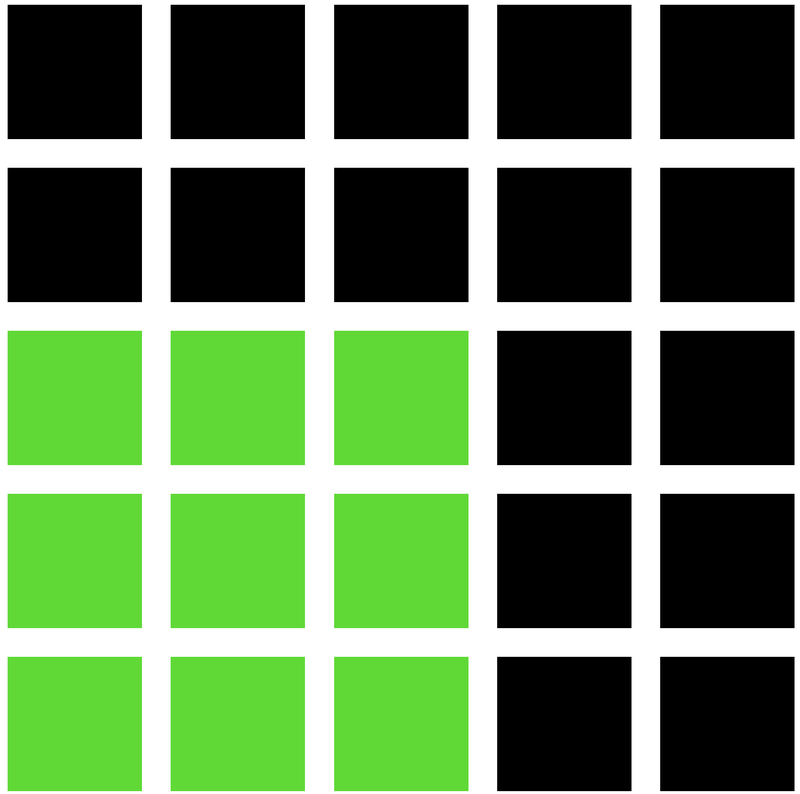
Global memory

Shared memory

Registers

Пройдемся по понятиям

Основы работы GPU



Размер thread group 3x3

Shared memory в Metal - Threadgroup memory

Global memory

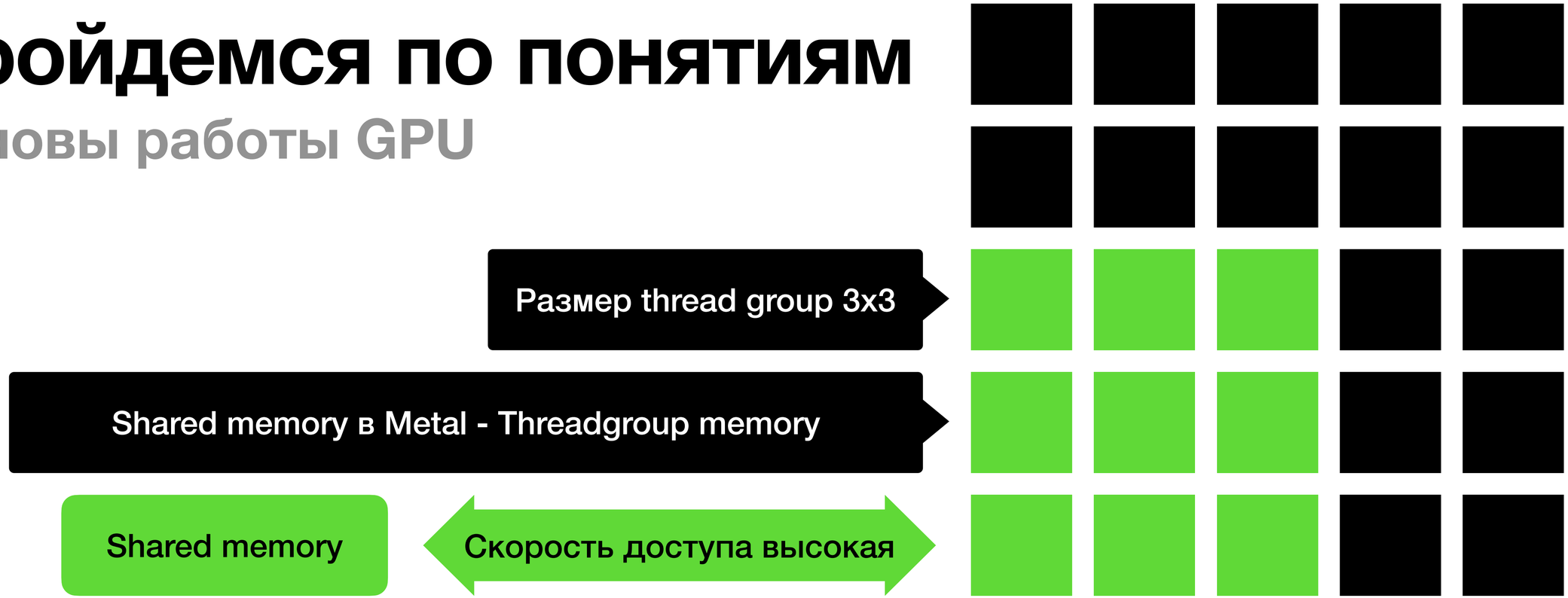
L1/L2 cache

Shared memory

Registers

Пройдемся по понятиям

Основы работы GPU



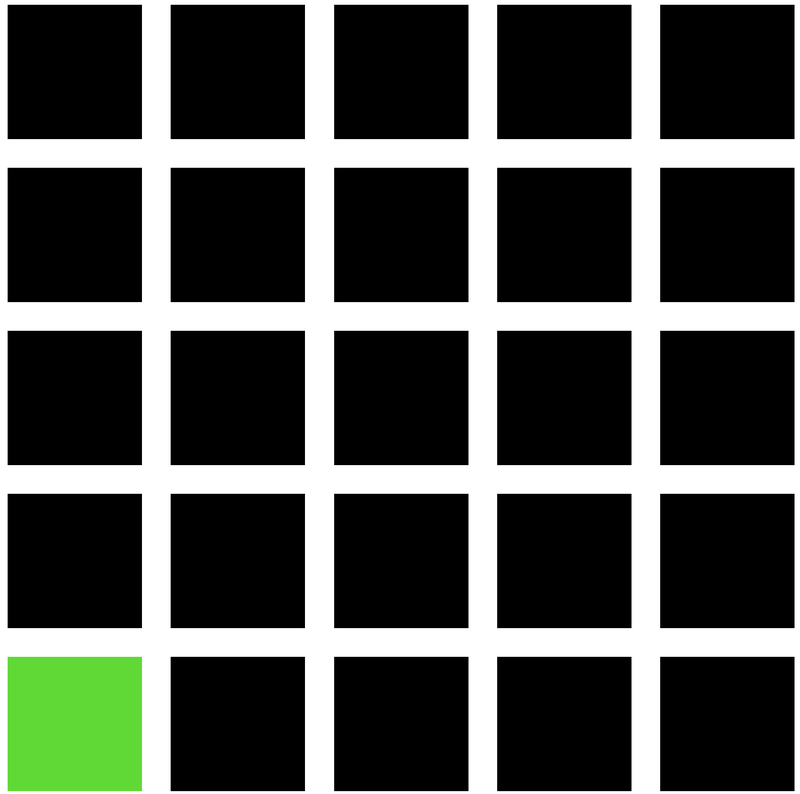
Global memory

L1/L2 cache

Registers

Пройдемся по понятиям

Основы работы GPU



Global memory

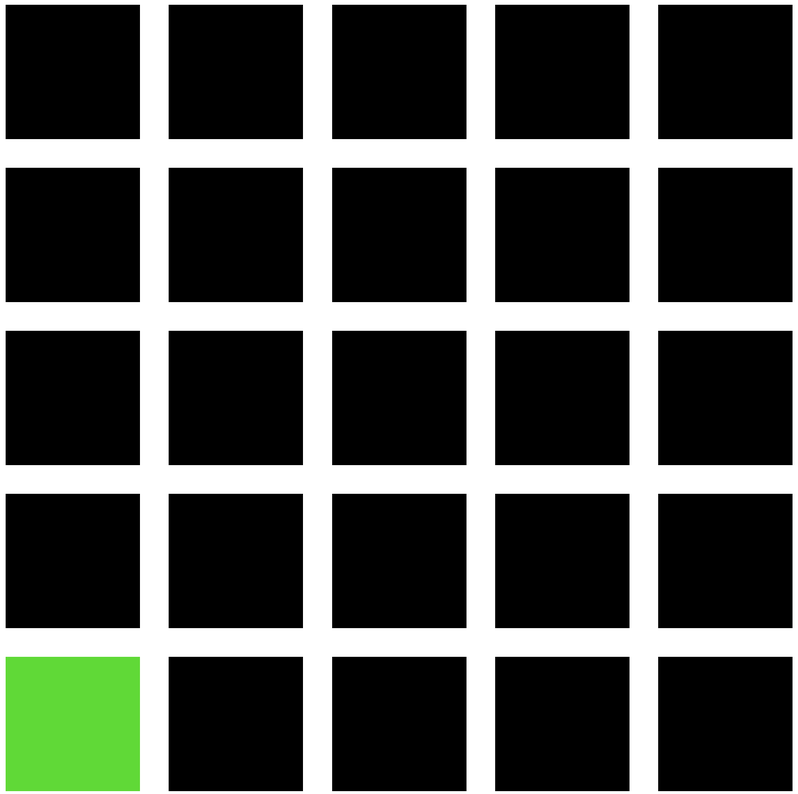
L1/L2 cache

Shared memory

Registers

Пройдемся по понятиям

Основы работы GPU



Registers

Скорость доступа самая высокая

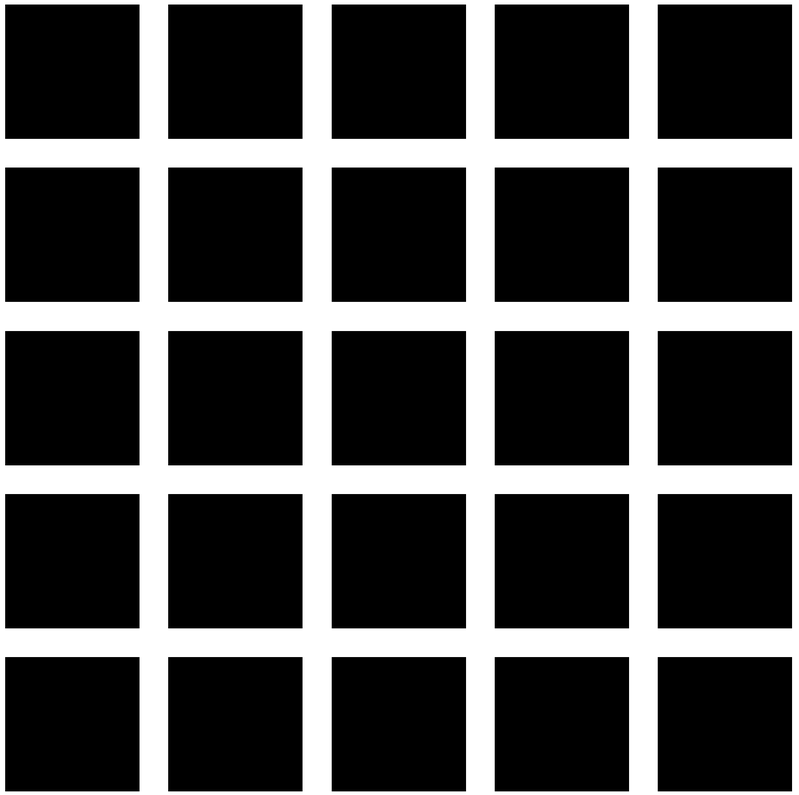
Global memory

L1/L2 cache

Shared memory

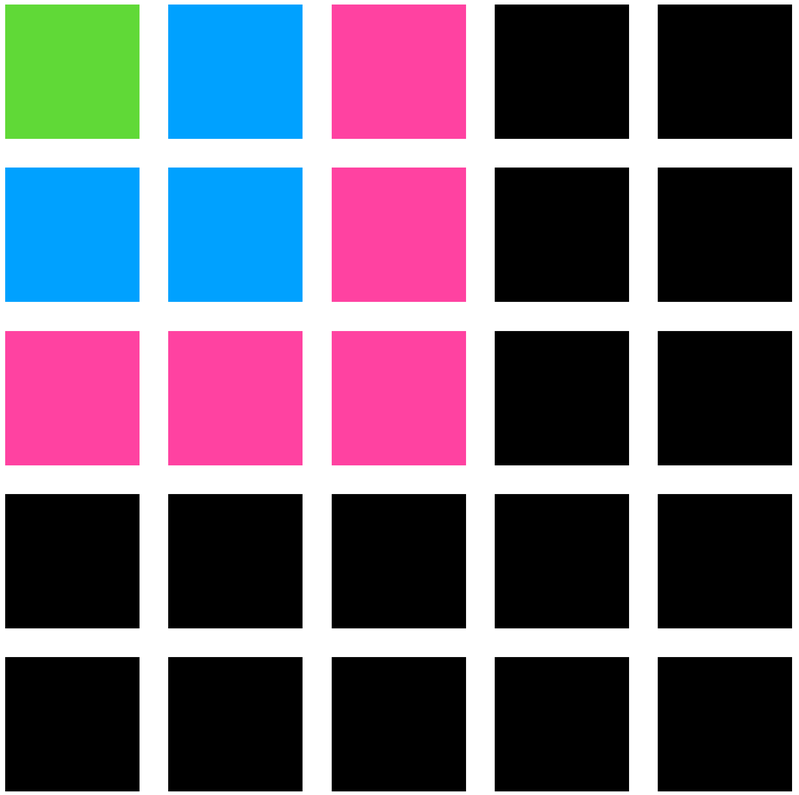
Пройдемся по понятиям

Gaussian blur



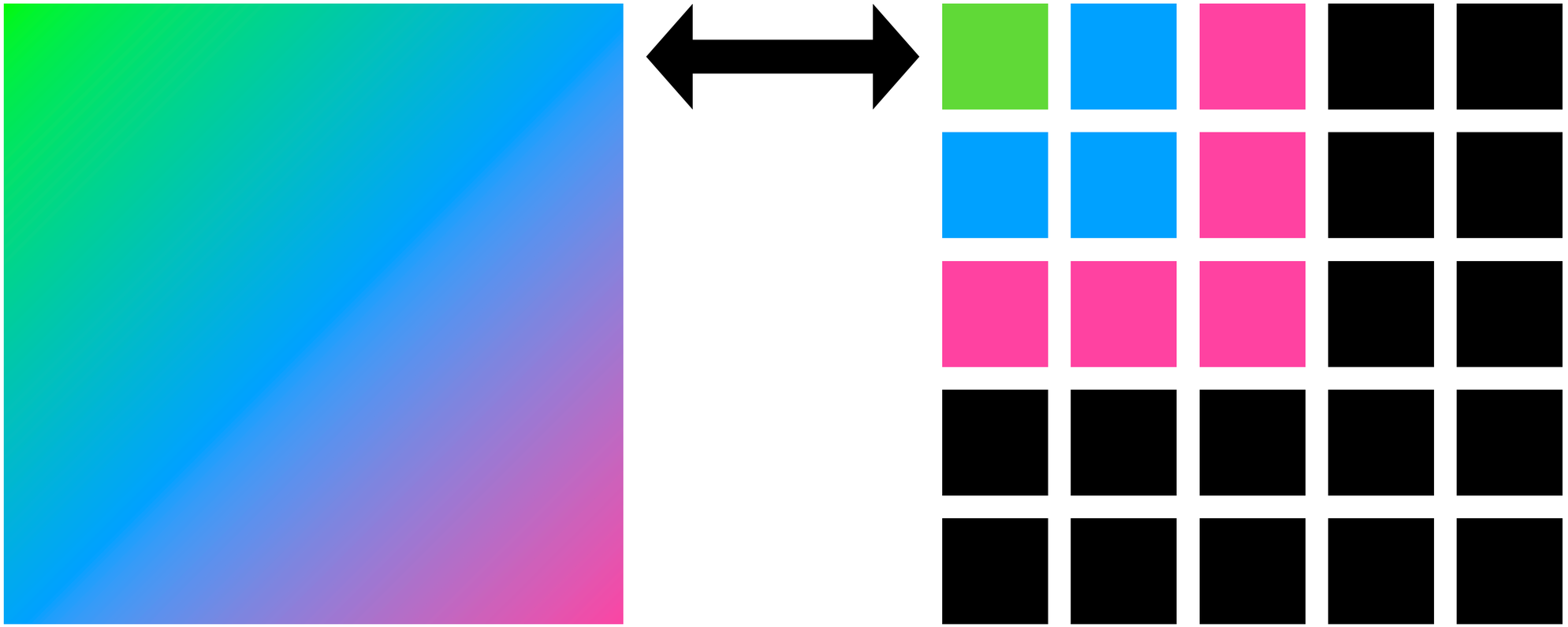
Пройдемся по понятиям

Gaussian blur



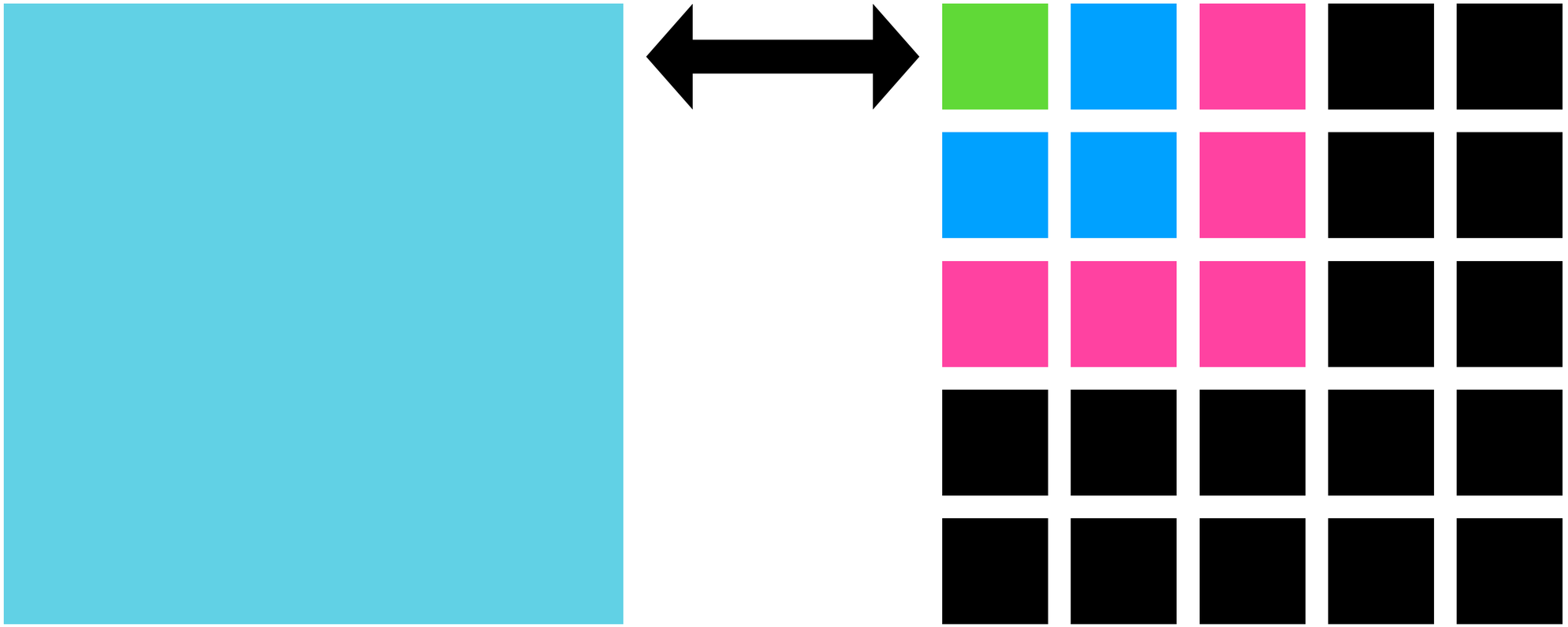
Пройдемся по понятиям

Gaussian blur



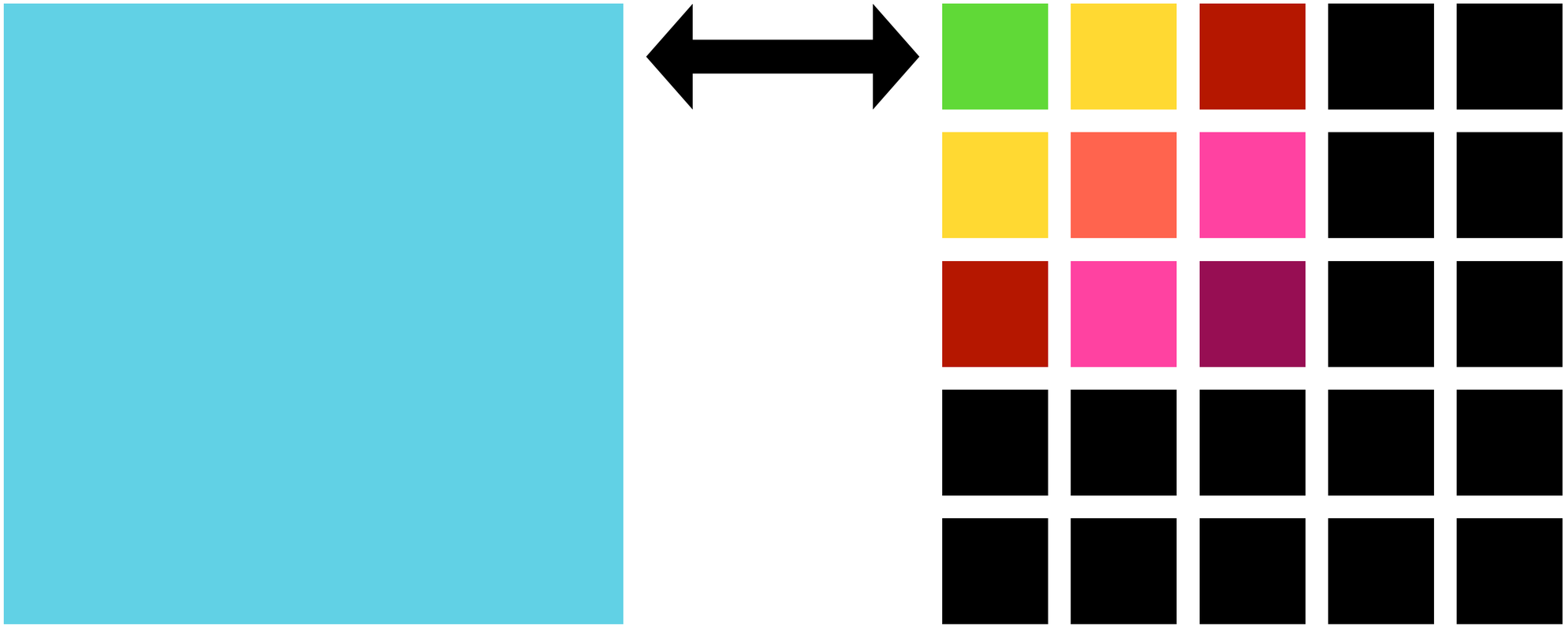
Пройдемся по понятиям

Gaussian blur

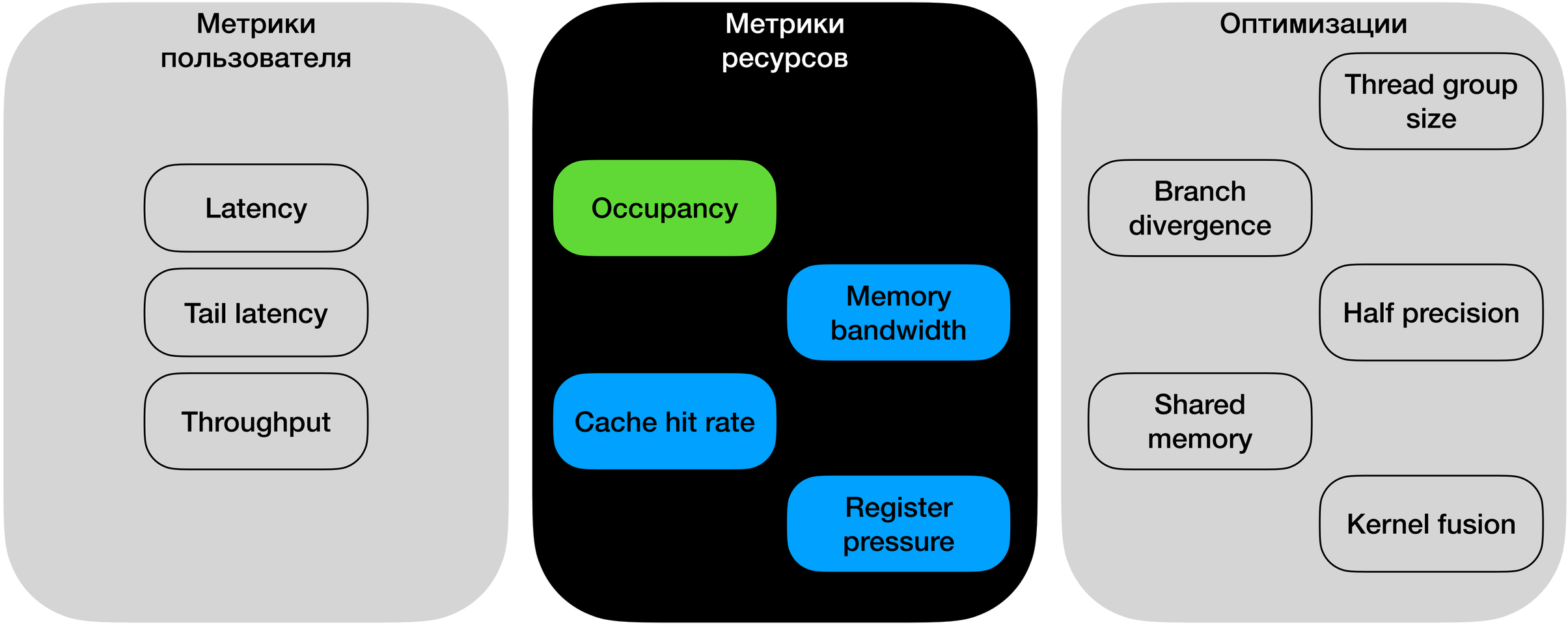


Пройдемся по понятиям

Gaussian blur



Как измерить «производительность»?



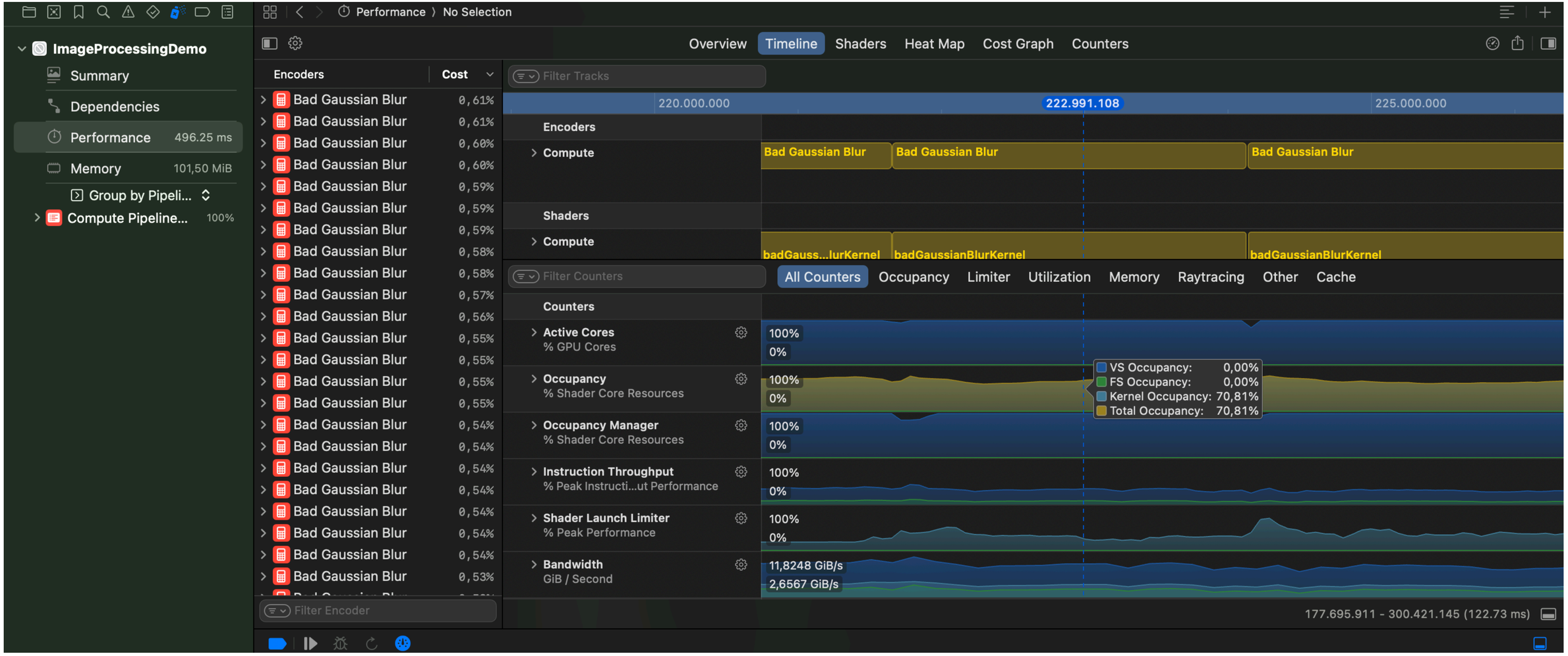
Occupancy

Степень использования ресурсов GPU на уровне потоков

- Profiling и поиск данных.

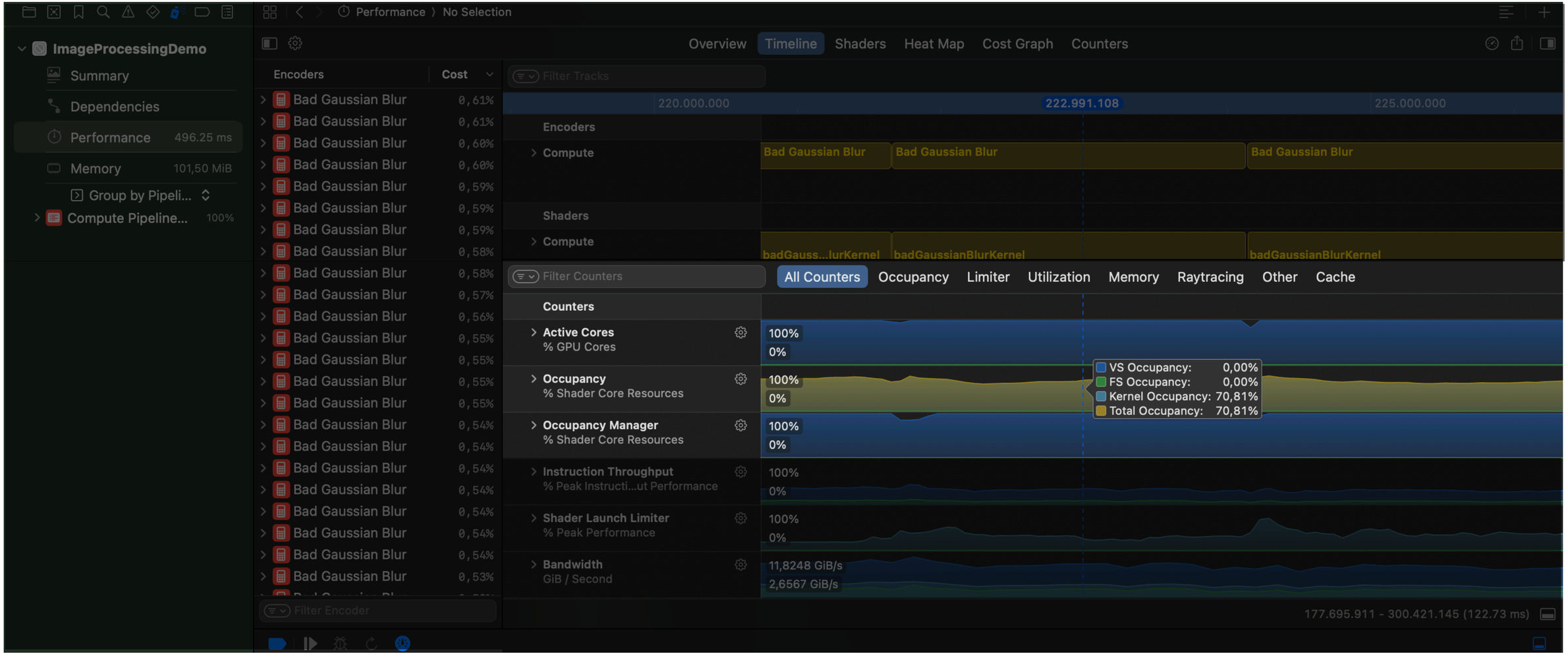
Occupancy

Степень использования ресурсов GPU на уровне потоков



Occupancy

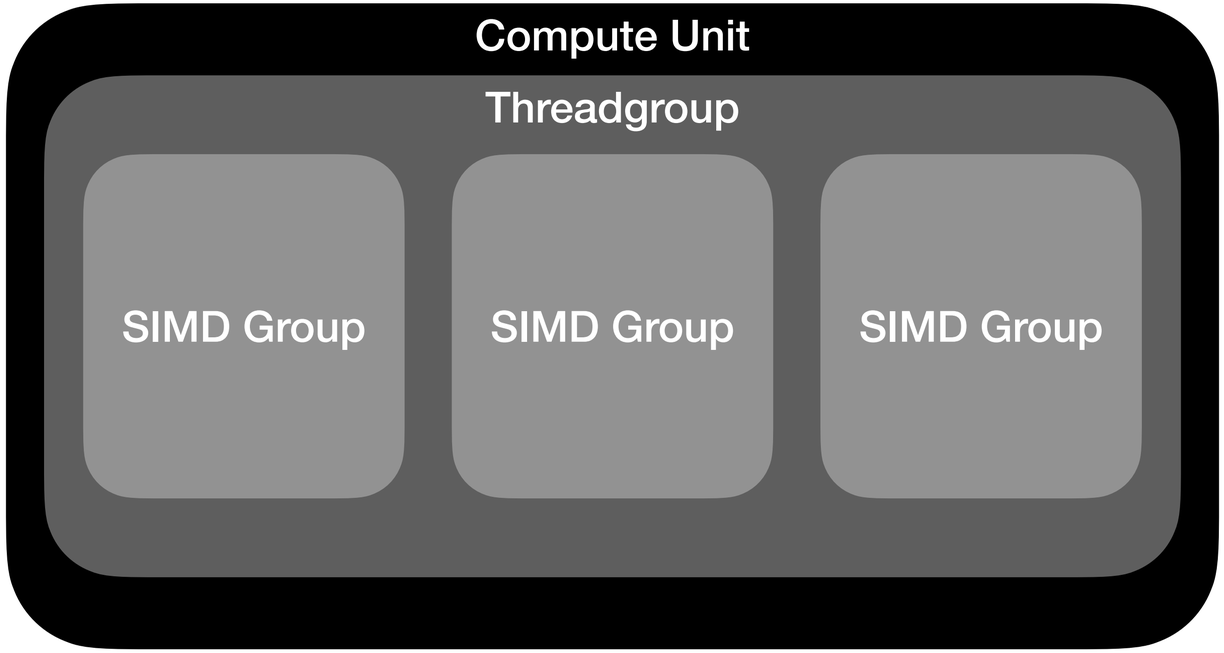
Степень использования ресурсов GPU на уровне потоков



Occupancy

Степень использования ресурсов GPU на уровне потоков

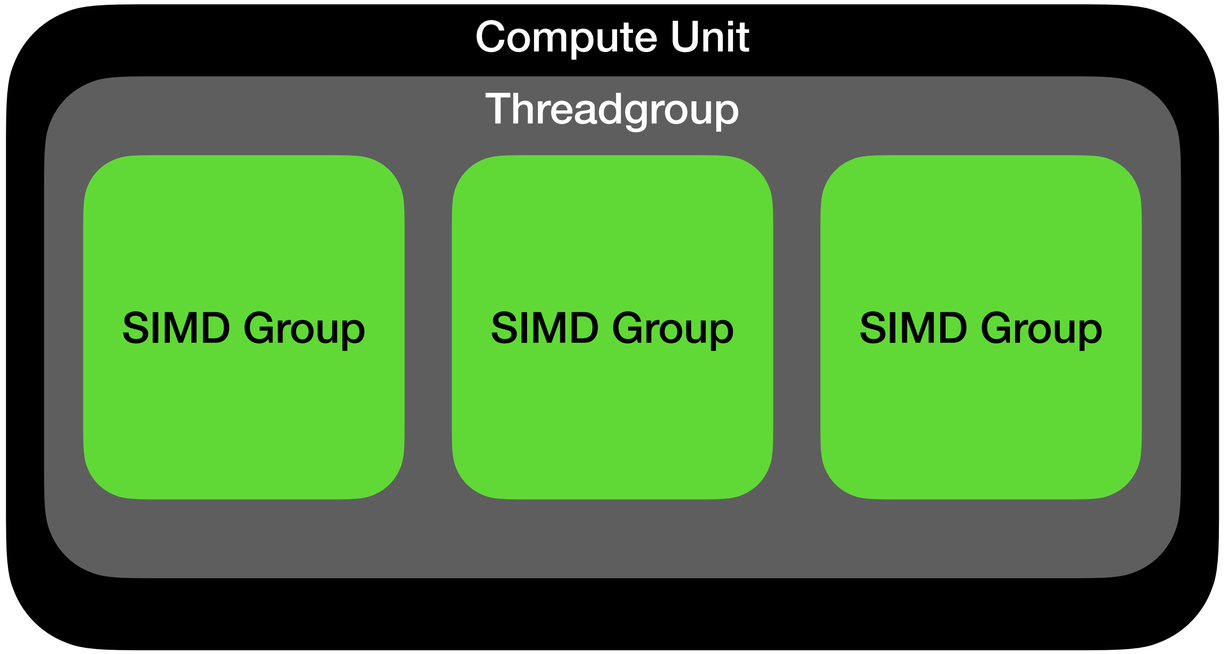
- Profiling и поиск данных.
- Зависимость от **регистров** и **общей памяти**.



Occupancy

Степень использования ресурсов GPU на уровне потоков

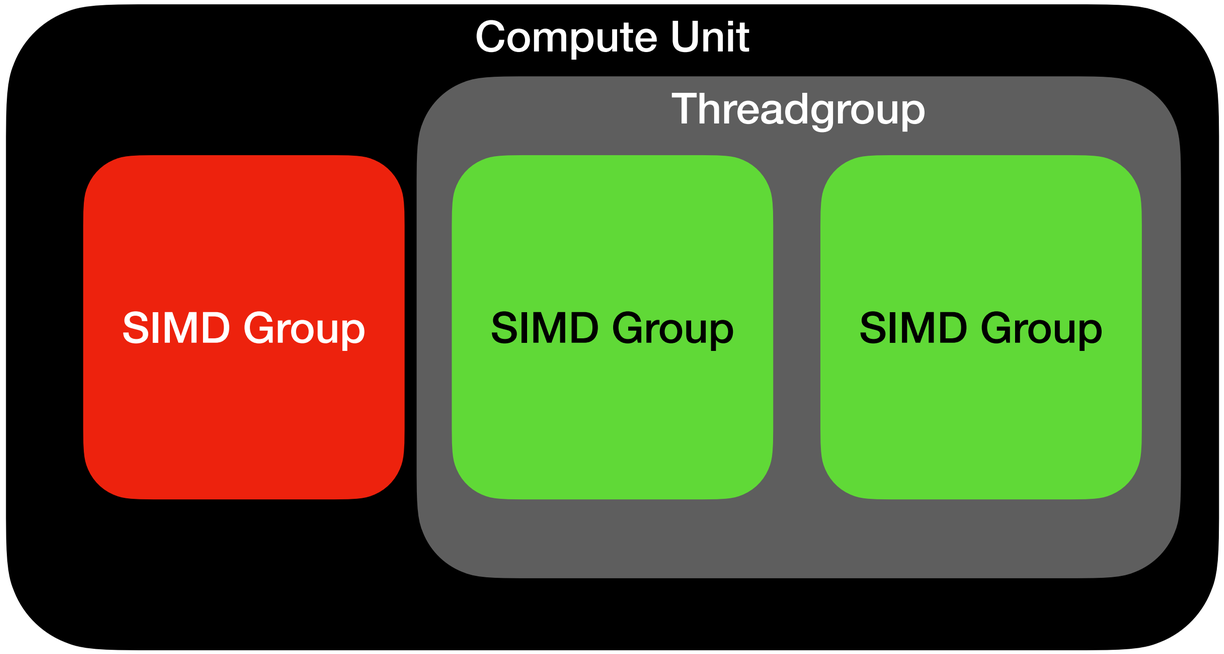
- Profiling и поиск данных.
- Зависимость от **регистров** и **общей памяти.**



Occupancy

Степень использования ресурсов GPU на уровне потоков

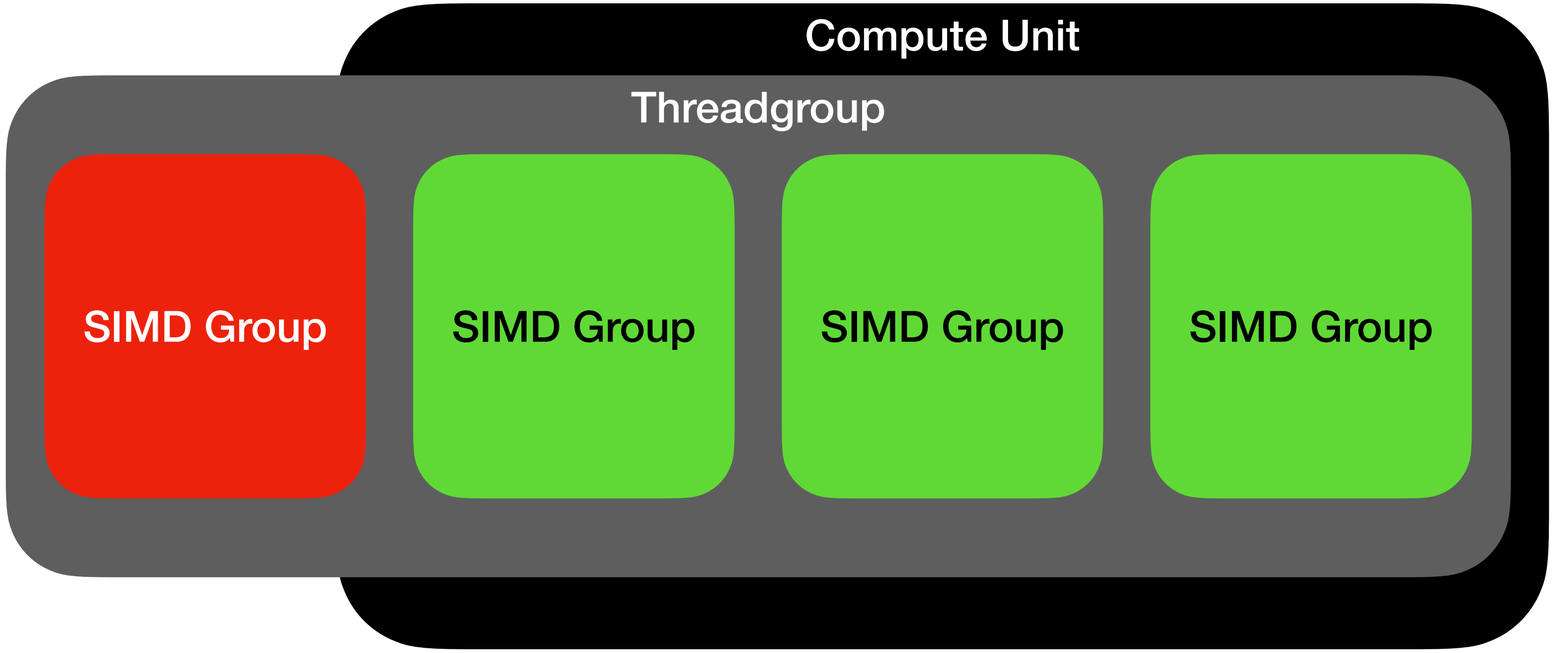
- Profiling и поиск данных.
- Зависимость от **регистров** и **общей памяти.**



Occupancy

Степень использования ресурсов GPU на уровне потоков

- Profiling и поиск данных.
- Зависимость от **регистров** и **общей памяти.**

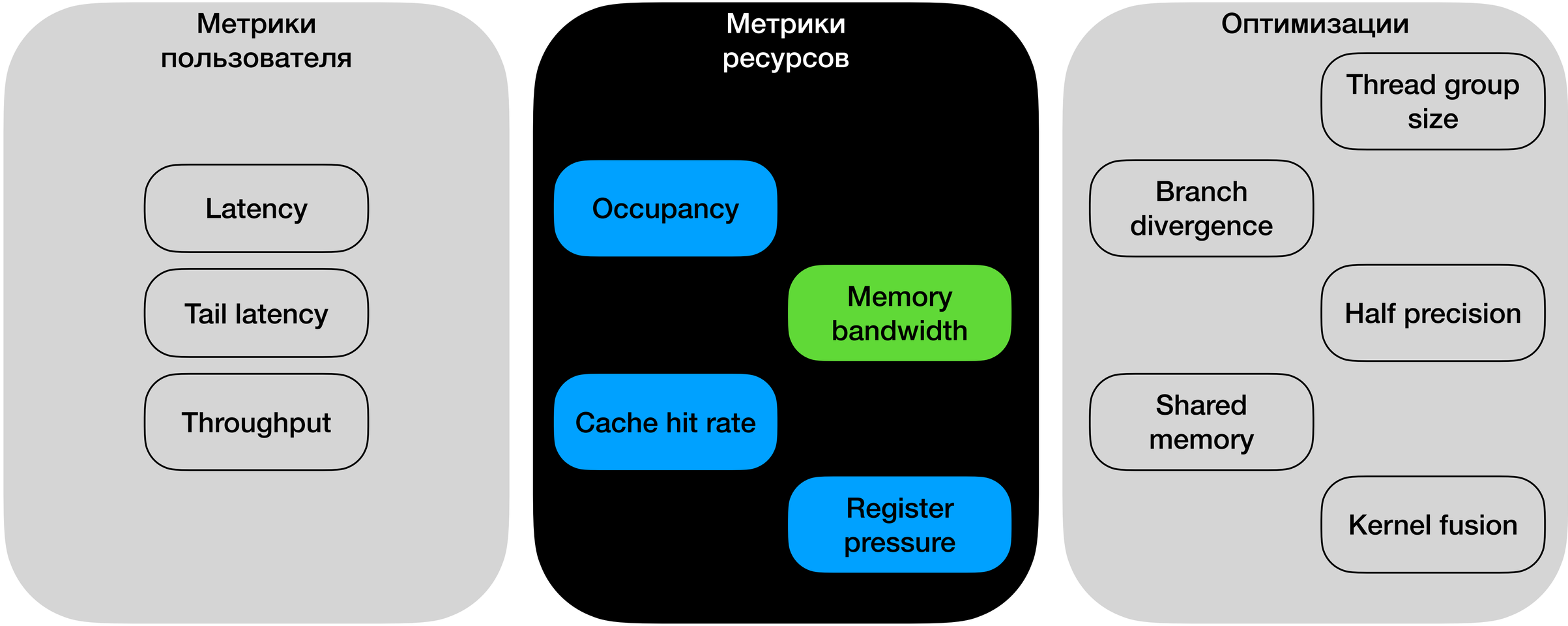


Occupancy

Степень использования ресурсов GPU на уровне потоков

- Profiling и поиск данных.
- Зависимость от регистров и общей памяти.
- Чем **выше** значение данной метрики, тем **больше потоков могут быть** задействованы в работе.

Как измерить «производительность»?



Memory bandwidth

Сколько данных GPU может передать за единицу времени

- Поиск по **формуле**: общее число байт / время выполнения.

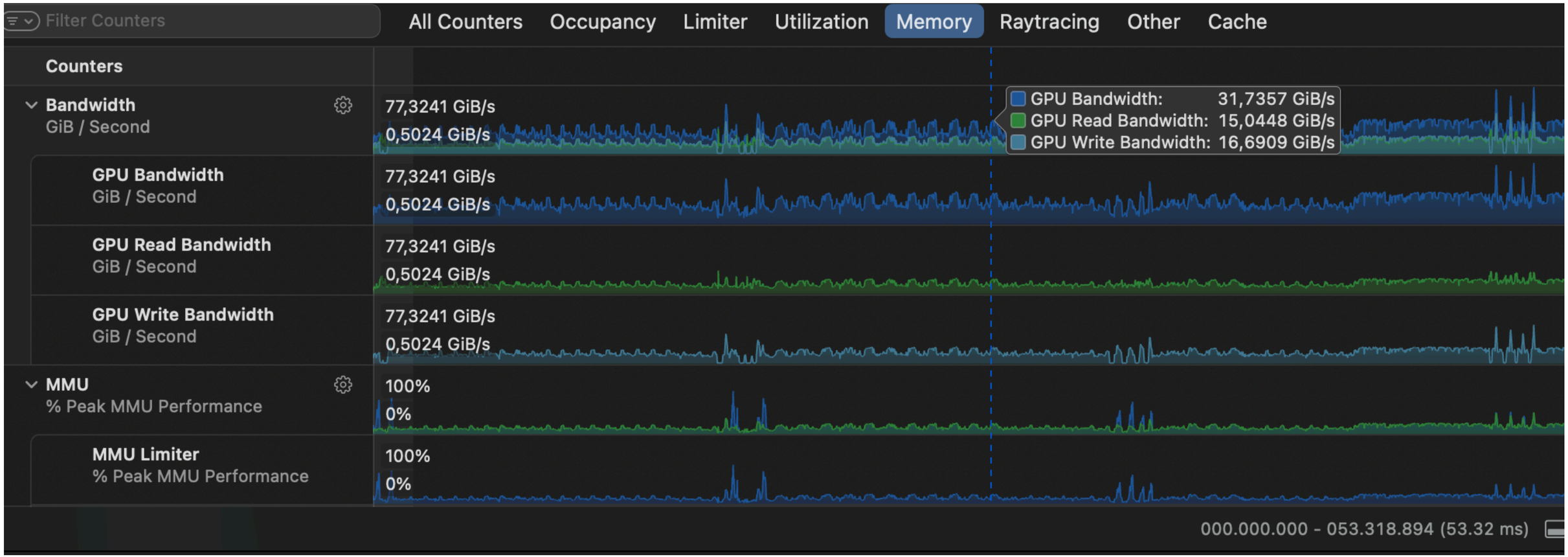
Memory bandwidth

Сколько данных GPU может передать за единицу времени

- Поиск по формуле: общее число байт / время выполнения.
- Profiling и поиск данных.

Memory bandwidth

Сколько данных GPU может передать за единицу времени



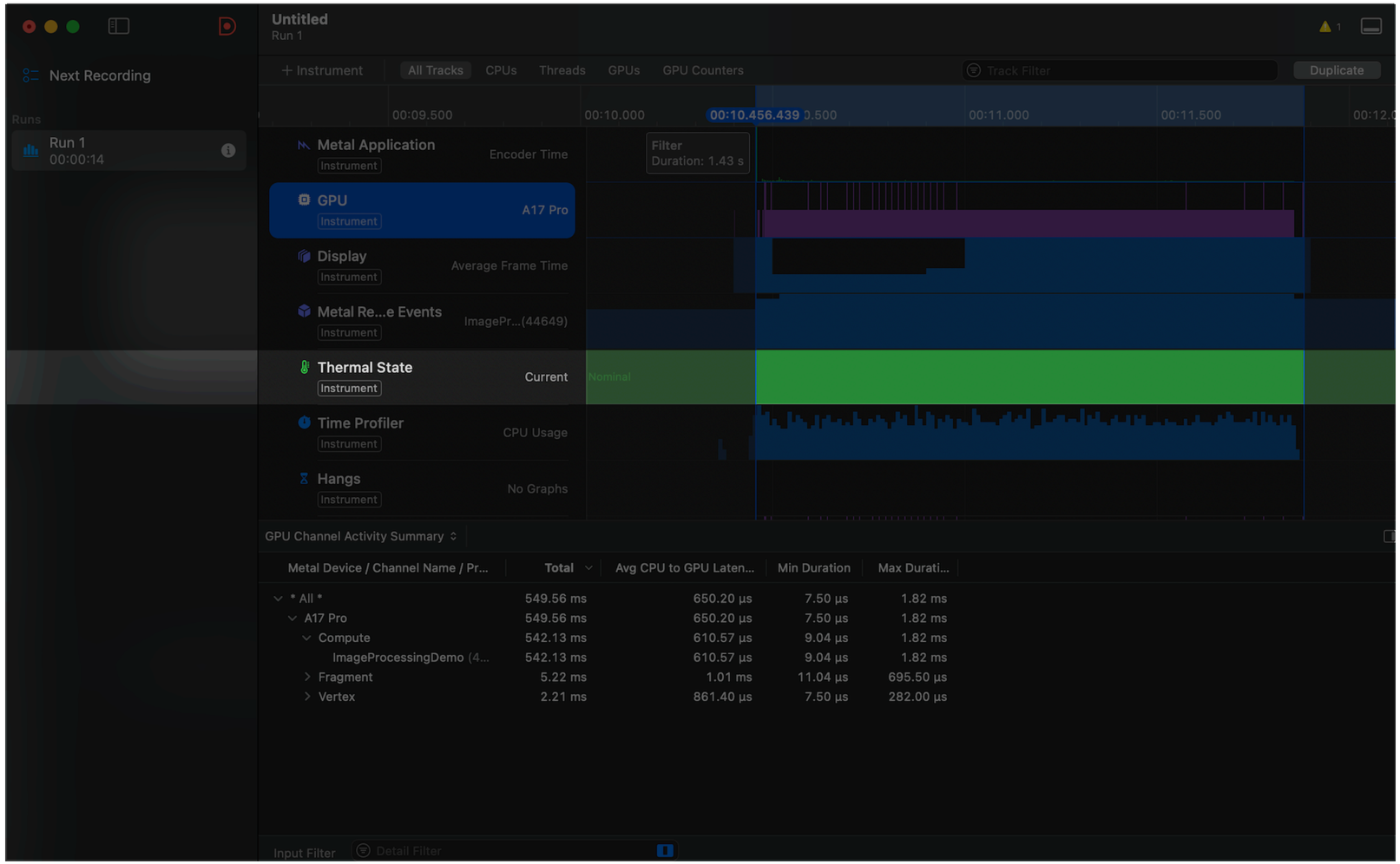
Memory bandwidth

Сколько данных GPU может передать за единицу времени

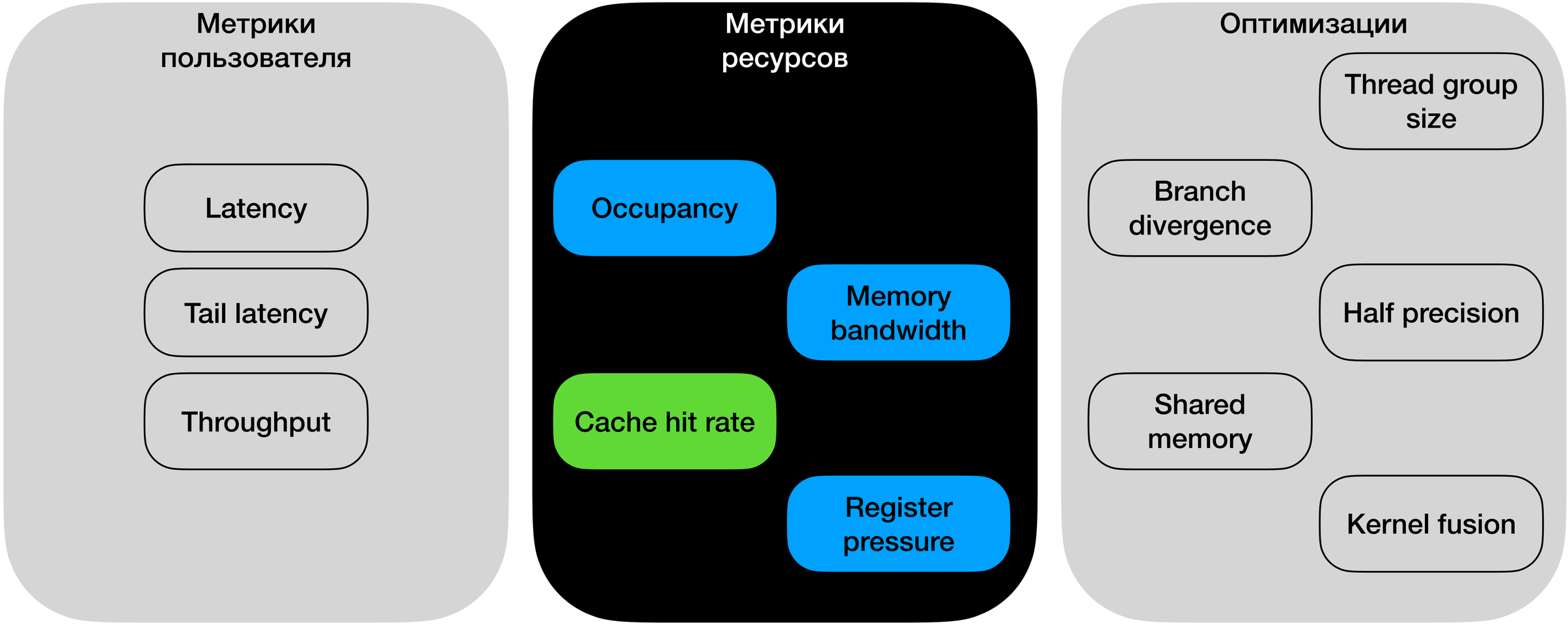
- Поиск по формуле: общее число байт / время выполнения.
- Profiling и поиск данных.
- Зависимость **thermal state** от показателей данной метрики. Чем **выше значение** метрики, тем **выше** вероятность **thermal throttling**.

Memory bandwidth

Сколько данных GPU может передать за единицу времени



Как измерить «производительность»?



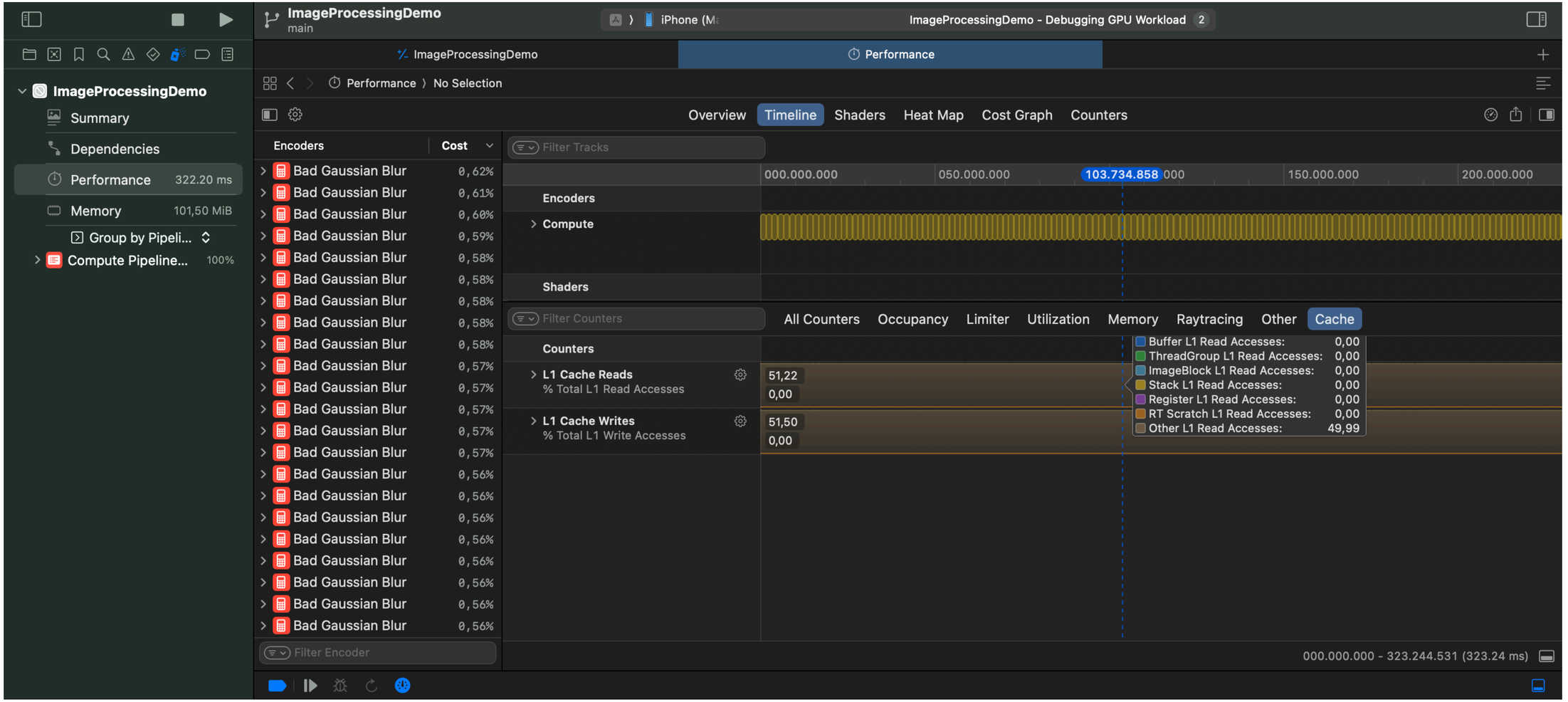
Cache hit rate

Процент успешных обращений к кэш памяти

- Profiling и поиск данных.

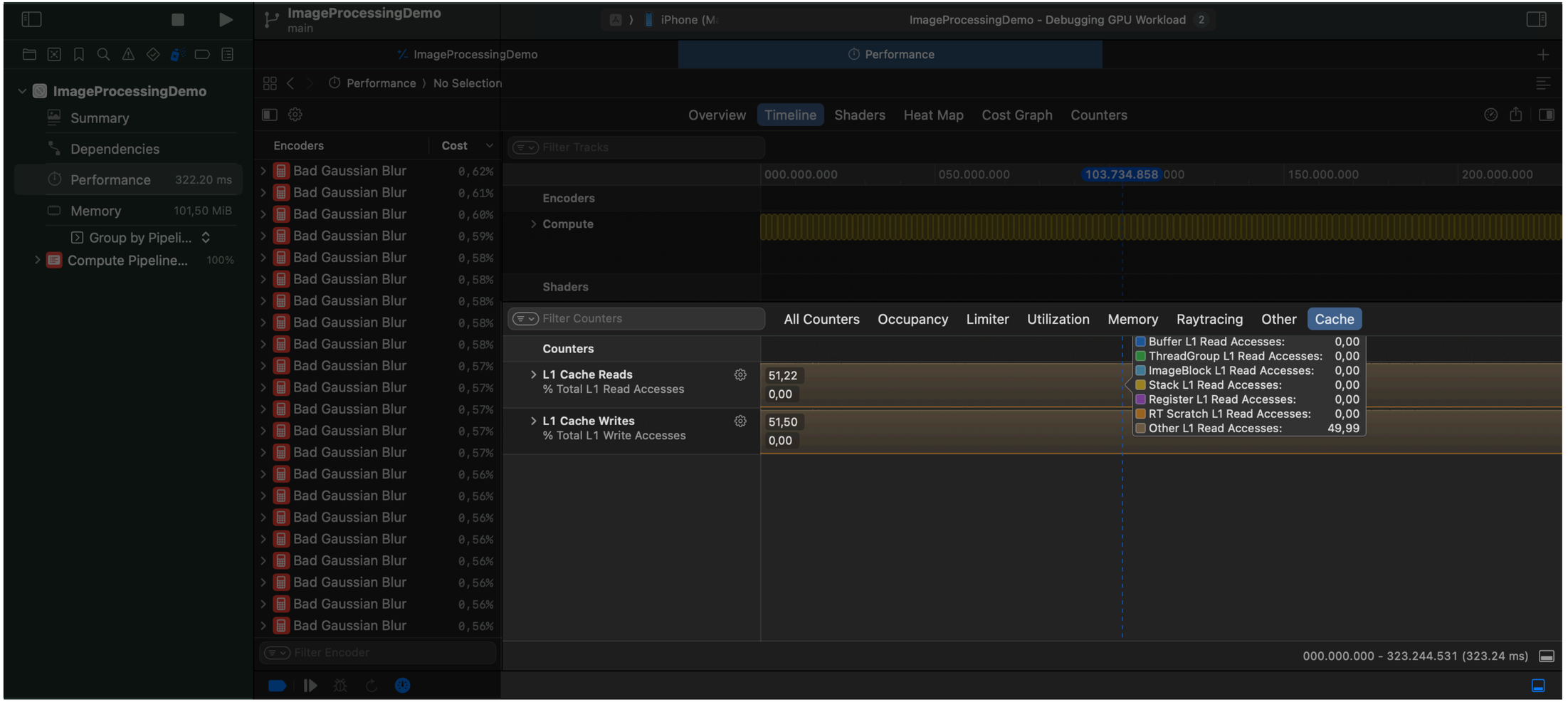
Cache hit rate

Процент успешных обращений к кэш памяти



Cache hit rate

Процент успешных обращений к кэш памяти



Cache hit rate

Процент успешных обращений к кэш памяти

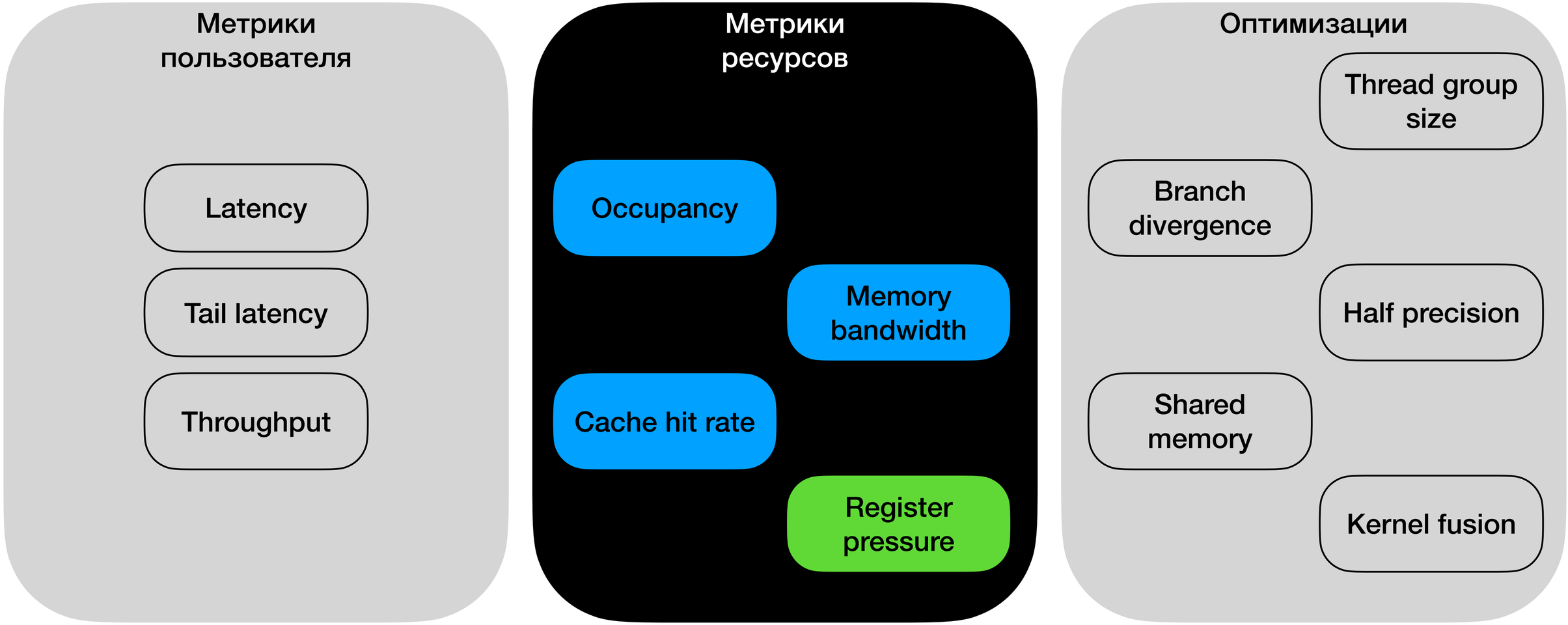
- Profiling и поиск данных.
- Зависимость **throughput** от показателей данной метрики.

Cache hit rate

Процент успешных обращений к кэш памяти

- Profiling и поиск данных.
- Зависимость throughput от показателей данной метрики.
- Чем **выше** данная метрика, тем **быстрее** выполняется работа по **отображению** или **вычислению**.

Как измерить «производительность»?



Register pressure

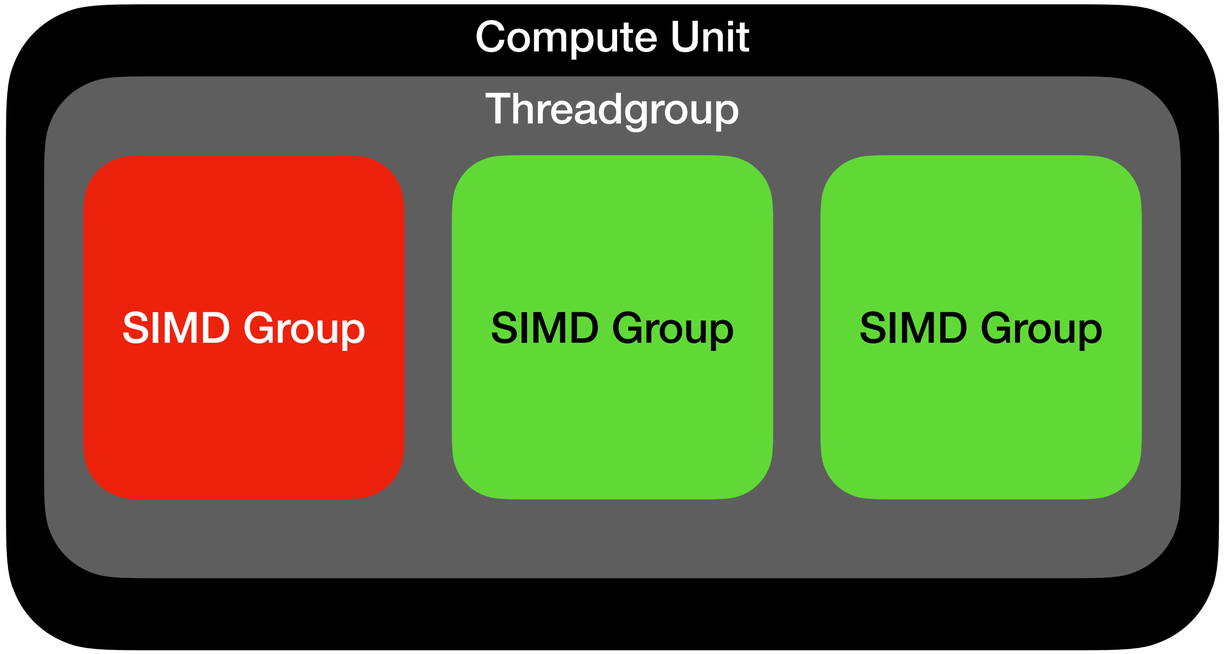
Сколько регистров использует каждый поток внутри GPU

- Profiling и поиск данных.

Register pressure

Сколько регистров использует каждый поток внутри GPU

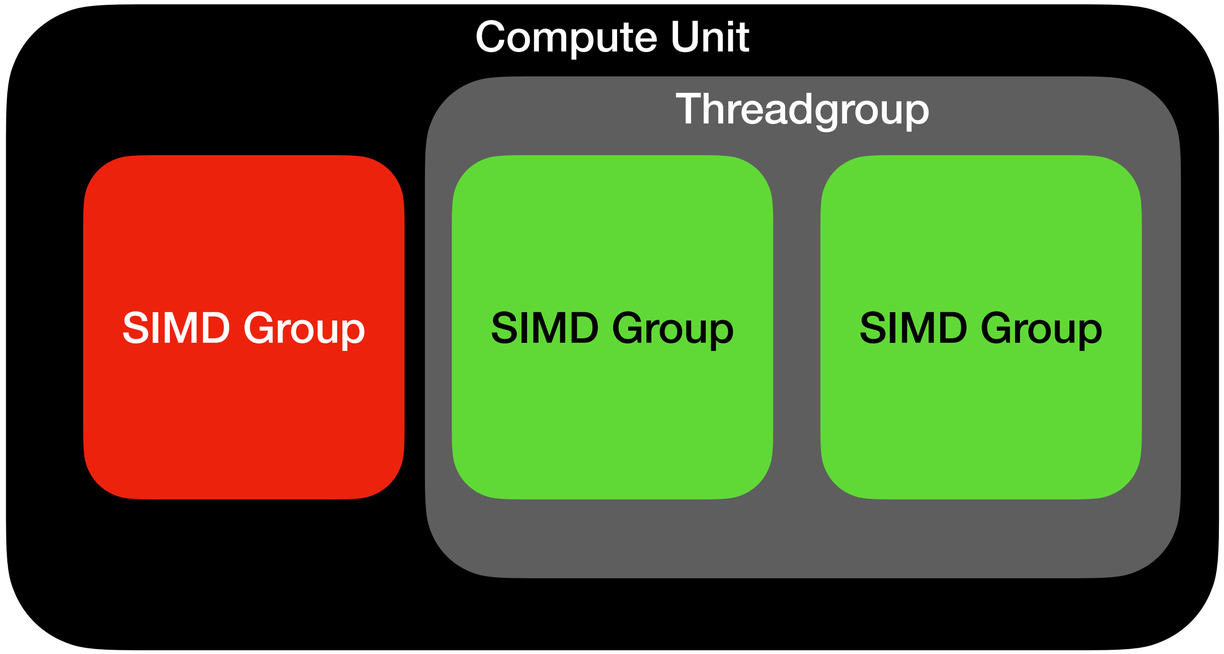
- Profiling и поиск данных.
- Данная метрика напрямую **ограничивает occupancy**.



Register pressure

Сколько регистров использует каждый поток внутри GPU

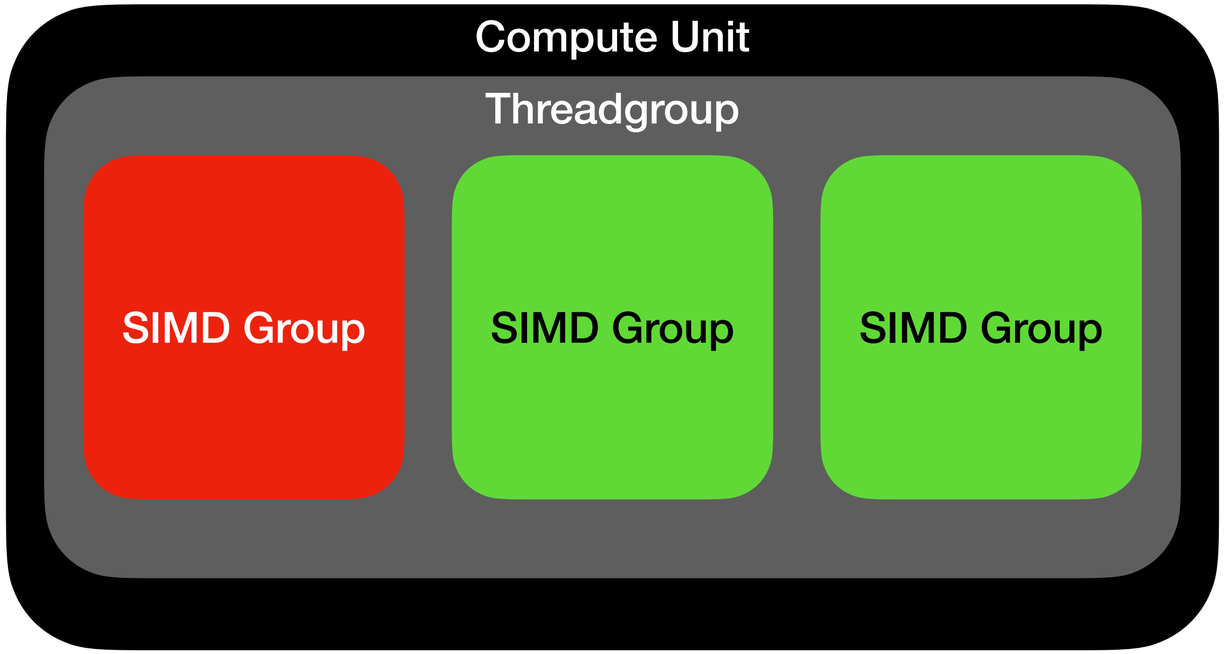
- Profiling и поиск данных.
- Данная метрика напрямую ограничивает occupancy.
- **Увеличение thread group size** с **отсутствием роста occupancy** также является показателем **проблем** с данной **метрикой**.



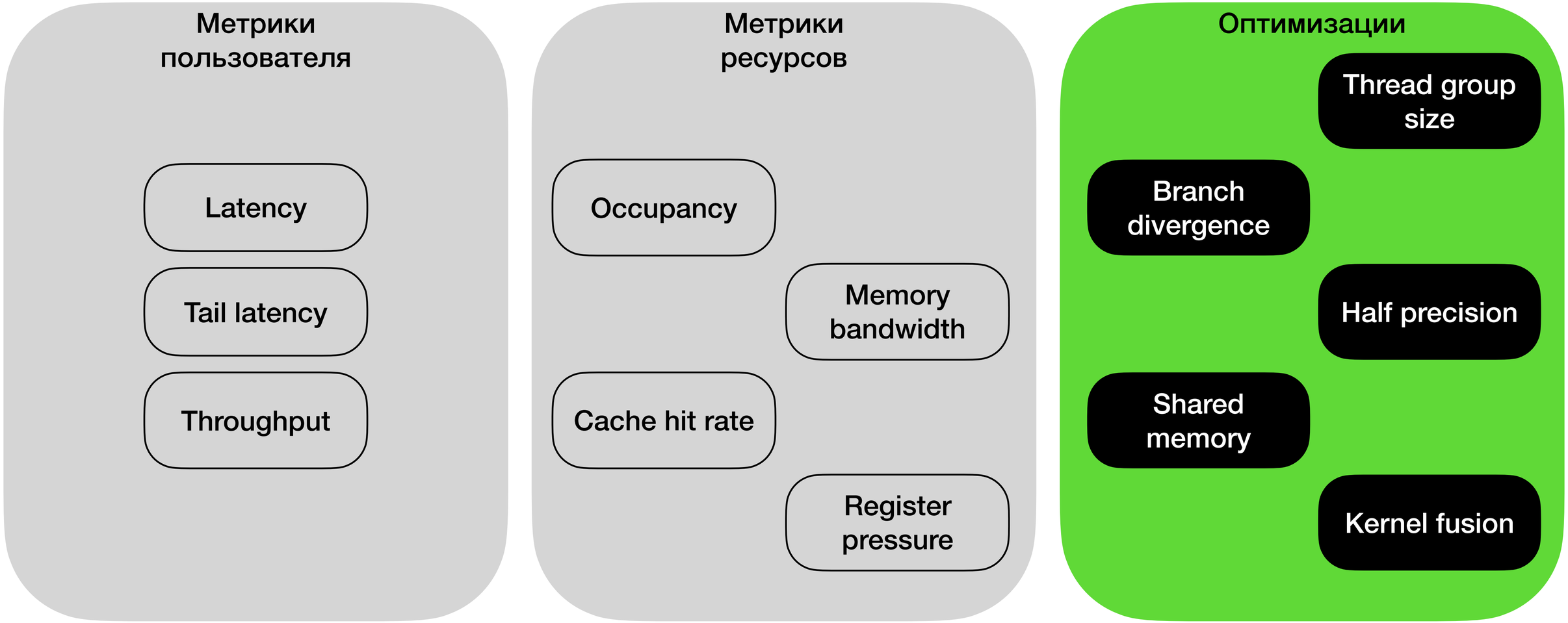
Register pressure

Сколько регистров использует каждый поток внутри GPU

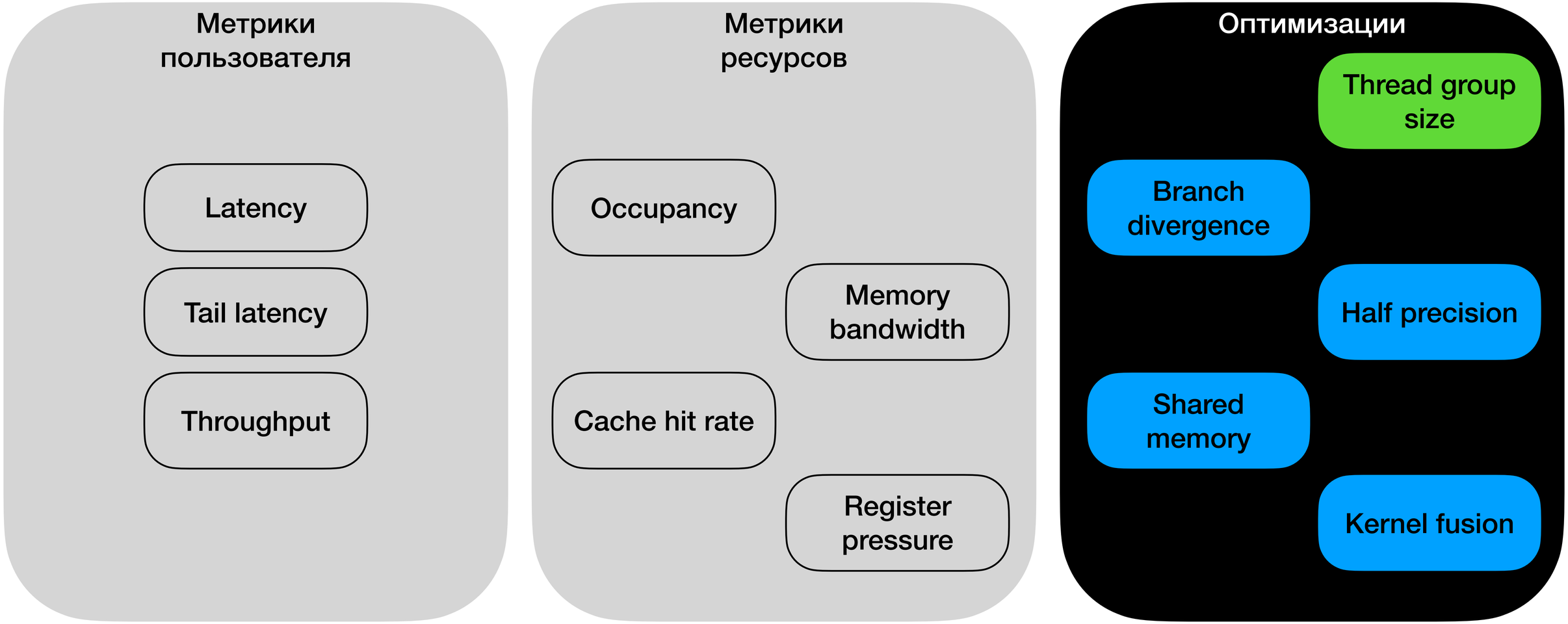
- Profiling и поиск данных.
- Данная метрика напрямую ограничивает occupancy.
- **Увеличение thread group size** с **отсутствием роста occupancy** также является показателем **проблем** с данной **метрикой**.



Как измерить «производительность»?

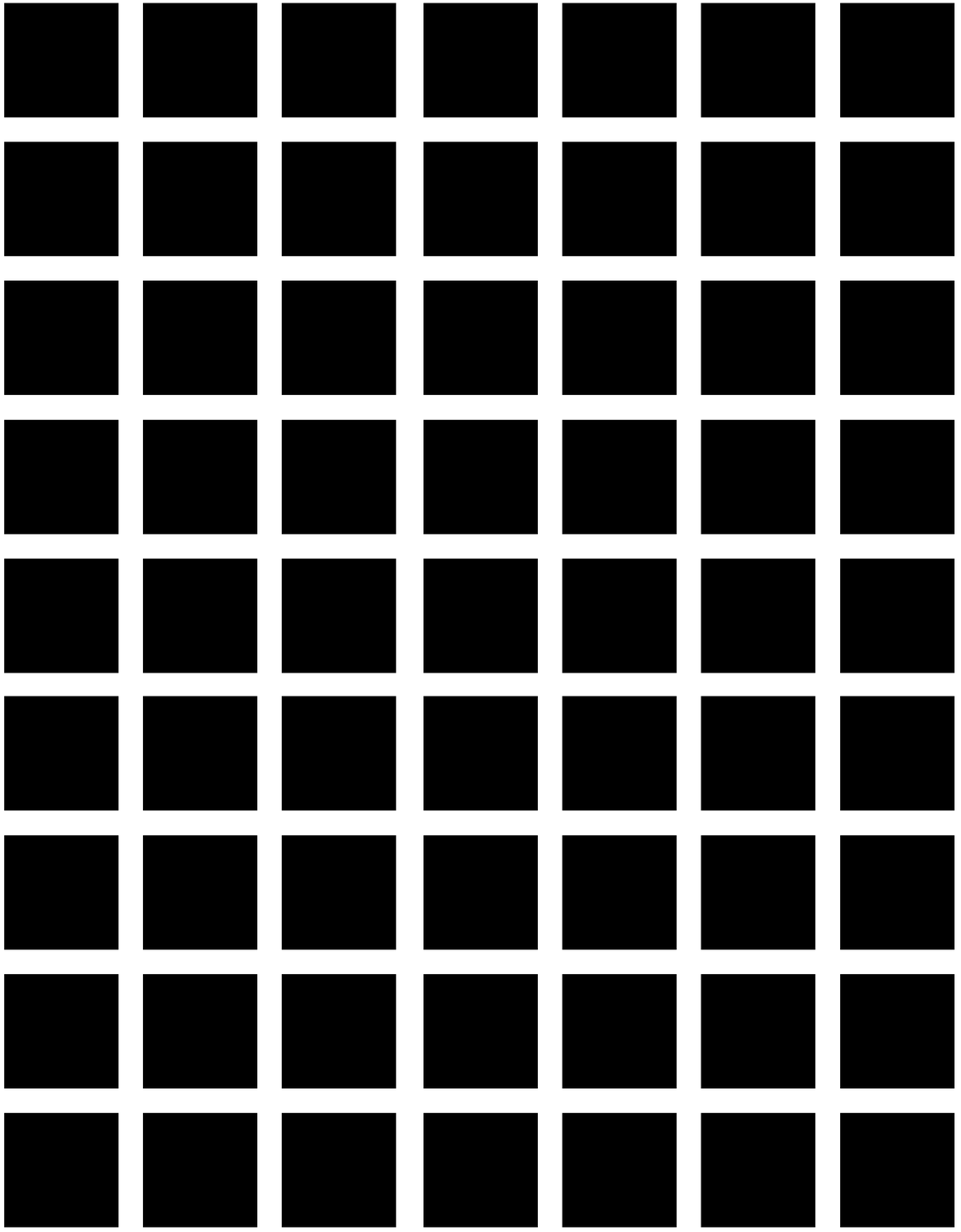


Как измерить «производительность»?



Threadgroup size

Размер группы потоков



Threadgroup size

Размер группы потоков

```
func encode(
    _ encoder: MTLComputeCommandEncoder,
    inputTexture: MTLTexture,
    outputTexture: MTLTexture,
    subtype: GaussComputePass.Subtype
) {
    encoder.label = makeEncoderLabel(by: subtype)
    encoder.setComputePipelineState(makePipelineState(by: subtype))

    encoder.setTexture(inputTexture, index: 0)
    encoder.setTexture(outputTexture, index: 1)

    let threadGroupSize = MTLSize(width: 4, height: 4, depth: 1)
    let threadGroupCount = MTLSize(
        width: (inputTexture.width  + threadGroupSize.width  - 1) / threadGroupSize.width,
        height: (inputTexture.height + threadGroupSize.height - 1) / threadGroupSize.height,
        depth: 1
    )

    encoder.dispatchThreadgroups(
        threadGroupCount,
        threadsPerThreadgroup: threadGroupSize
    )
}
```

Threadgroup size

Размер группы потоков

```
func encode(
    _ encoder: MTLComputeCommandEncoder,
    inputTexture: MTLTexture,
    outputTexture: MTLTexture,
    subtype: GaussComputePass.Subtype
) {
    encoder.label = makeEncoderLabel(by: subtype)
    encoder.setComputePipelineState(makePipelineState(by: subtype))

    encoder.setTexture(inputTexture, index: 0)
    encoder.setTexture(outputTexture, index: 1)

    let threadGroupSize = MTLSize(width: 4, height: 4, depth: 1)
    let threadGroupCount = MTLSize(
        width: (inputTexture.width  + threadGroupSize.width  - 1) / threadGroupSize.width,
        height: (inputTexture.height + threadGroupSize.height - 1) / threadGroupSize.height,
        depth: 1
    )

    encoder.dispatchThreadgroups(
        threadGroupCount,
        threadsPerThreadgroup: threadGroupSize
    )
}
```

Threadgroup size

Размер группы потоков

```
func encode(
    _ encoder: MTLComputeCommandEncoder,
    inputTexture: MTLTexture,
    outputTexture: MTLTexture,
    subtype: GaussComputePass.Subtype
) {
    encoder.label = makeEncoderLabel(by: subtype)
    encoder.setComputePipelineState(makePipelineState(by: subtype))

    encoder.setTexture(inputTexture, index: 0)
    encoder.setTexture(outputTexture, index: 1)

    let threadGroupSize = MTLSize(width: 16, height: 16, depth: 1)
    let threadGroupCount = MTLSize(
        width: (inputTexture.width  + threadGroupSize.width  - 1) / threadGroupSize.width,
        height: (inputTexture.height + threadGroupSize.height - 1) / threadGroupSize.height,
        depth: 1
    )

    encoder.dispatchThreadgroups(
        threadGroupCount,
        threadsPerThreadgroup: threadGroupSize
    )
}
```

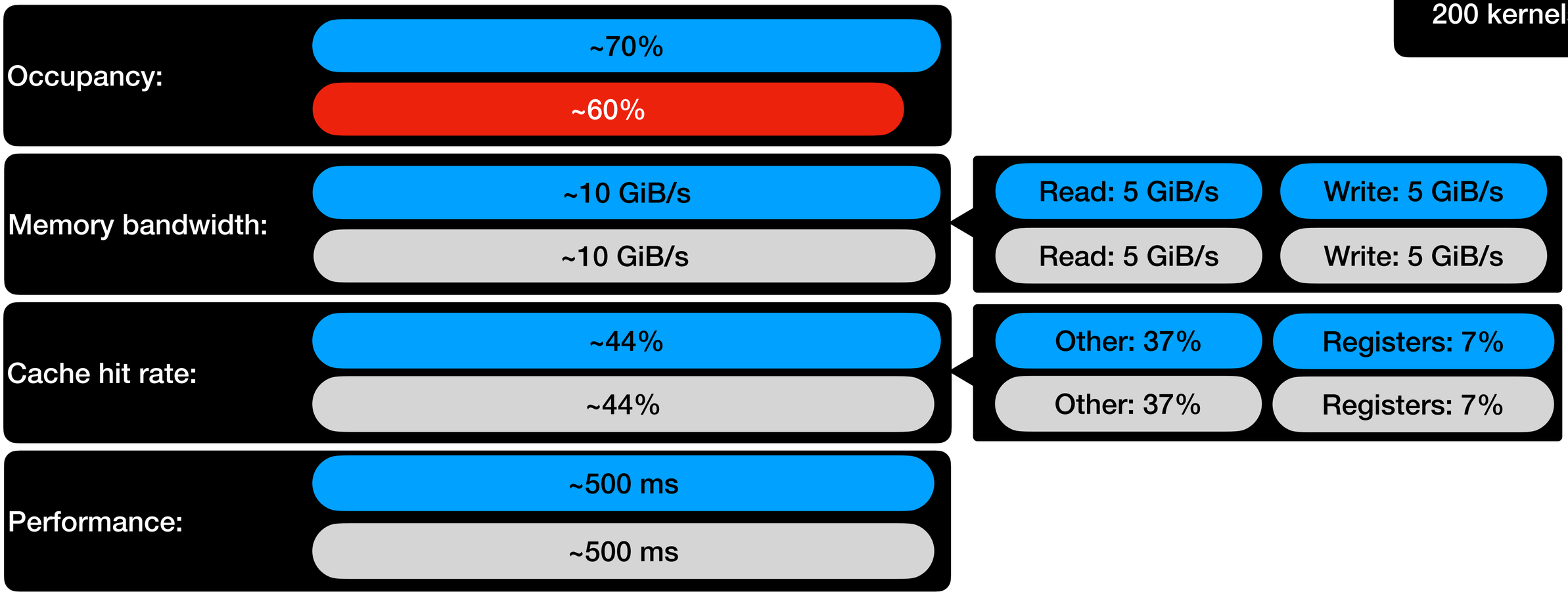
Threadgroup size

Размер группы потоков

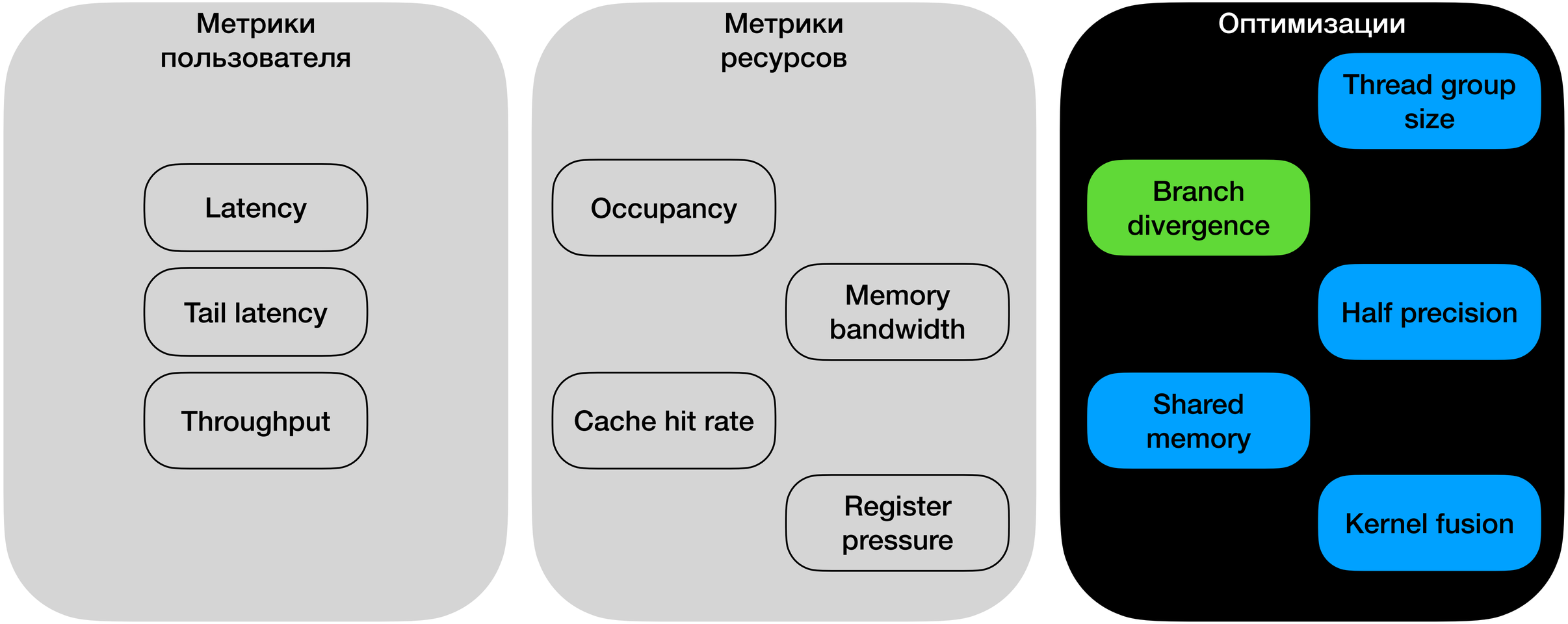
Без оптимизации

После оптимизации

200 kernels



Как измерить «производительность»?



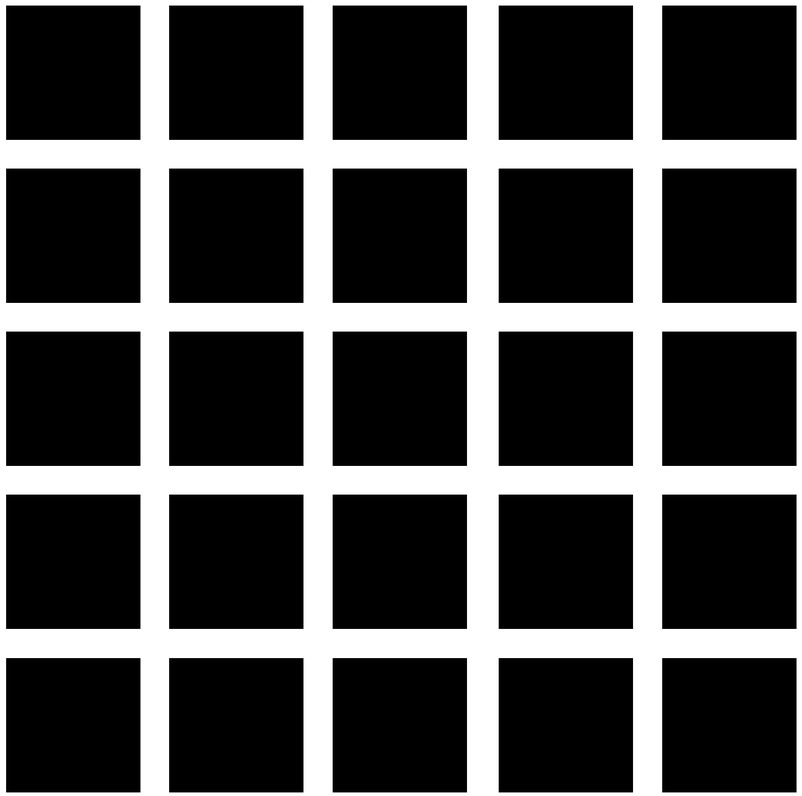
Branch divergence

Ветвление потоков

```
float gaussEdge[5][5] = {
    {1.0,  4.0,  7.0,  4.0, 1.0},
    {4.0, 16.0, 26.0, 16.0, 4.0},
    {7.0, 26.0, 41.0, 26.0, 7.0},
    {4.0, 16.0, 26.0, 16.0, 4.0},
    {1.0,  4.0,  7.0,  4.0, 1.0}
};

float4 sum = float4(0.0);
float weightSum = 0.0;

for (int ky = -radius; ky <= radius; ++ky) {
    for (int kx = -radius; kx <= radius; ++kx) {
        int sx = int(gid.x) + kx;
        int sy = int(gid.y) + ky;

        if (sx >= 0 &&
            sx < int(width) &&
            sy >= 0 &&
            sy < int(height)) {

            float4 color = input.read(uint2(sx, sy));
            float w = gaussEdge[ky + radius][kx + radius];

            sum += color * w;
            weightSum += w;
        }
    }
}

float4 result = sum / weightSum;

output.write(result, gid);
```

Branch divergence

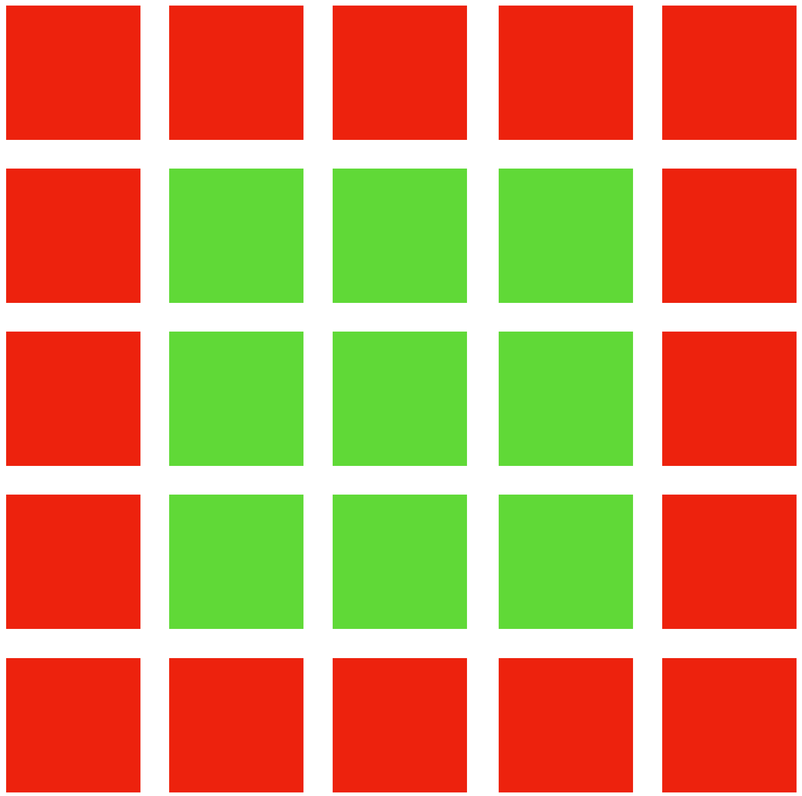
Ветвление потоков



```
float gaussEdge[5][5] = {
    {1.0,  4.0,  7.0,  4.0, 1.0},
    {4.0, 16.0, 26.0, 16.0, 4.0},
    {7.0, 26.0, 41.0, 26.0, 7.0},
    {4.0, 16.0, 26.0, 16.0, 4.0},
    {1.0,  4.0,  7.0,  4.0, 1.0}
};

float4 sum = float4(0.0);
float weightSum = 0.0;

for (int ky = -radius; ky <= radius; ++ky) {
    for (int kx = -radius; kx <= radius; ++kx) {
        int sx = int(gid.x) + kx;
        int sy = int(gid.y) + ky;

        if (sx >= 0 &&
            sx < int(width) &&
            sy >= 0 &&
            sy < int(height)) {

            float4 color = input.read(uint2(sx, sy));
            float w = gaussEdge[ky + radius][kx + radius];

            sum += color * w;
            weightSum += w;
        }
    }
}

float4 result = sum / weightSum;

output.write(result, gid);
```

Branch divergence

Ветвление потоков



```
float gaussEdge[5][5] = {
    {1.0,  4.0,  7.0,  4.0, 1.0},
    {4.0, 16.0, 26.0, 16.0, 4.0},
    {7.0, 26.0, 41.0, 26.0, 7.0},
    {4.0, 16.0, 26.0, 16.0, 4.0},
    {1.0,  4.0,  7.0,  4.0, 1.0}
};

float4 sum = float4(0.0);
float weightSum = 0.0;

for (int ky = -radius; ky <= radius; ++ky) {
    for (int kx = -radius; kx <= radius; ++kx) {
        int sx = int(gid.x) + kx;
        int sy = int(gid.y) + ky;

        if (sx >= 0 &&
            sx < int(width) &&
            sy >= 0 &&
            sy < int(height)) {

            float4 color = input.read(uint2(sx, sy));
            float w = gaussEdge[ky + radius][kx + radius];

            sum += color * w;
            weightSum += w;
        }
    }
}

float4 result = sum / weightSum;

output.write(result, gid);
```

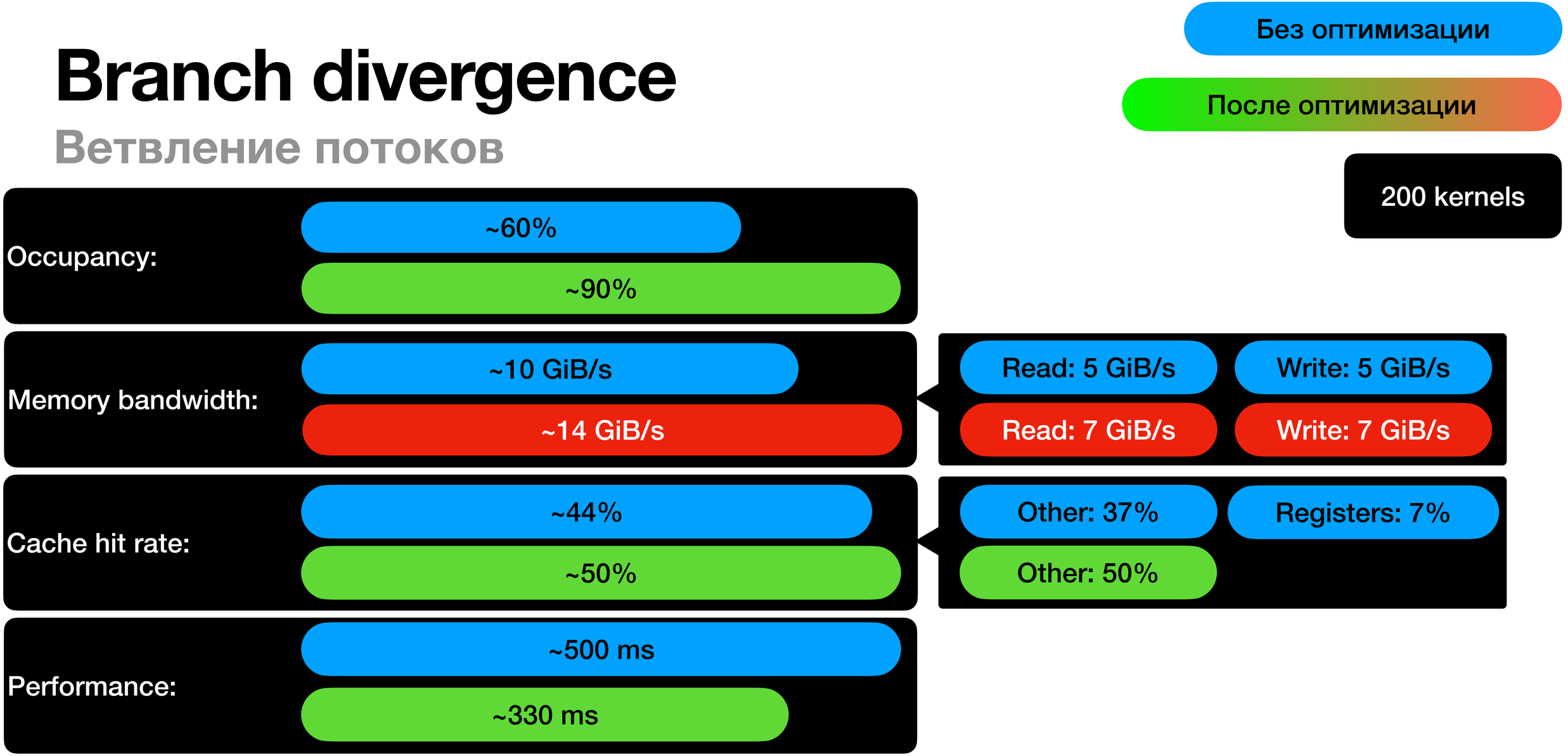
Branch divergence

Ветвление потоков

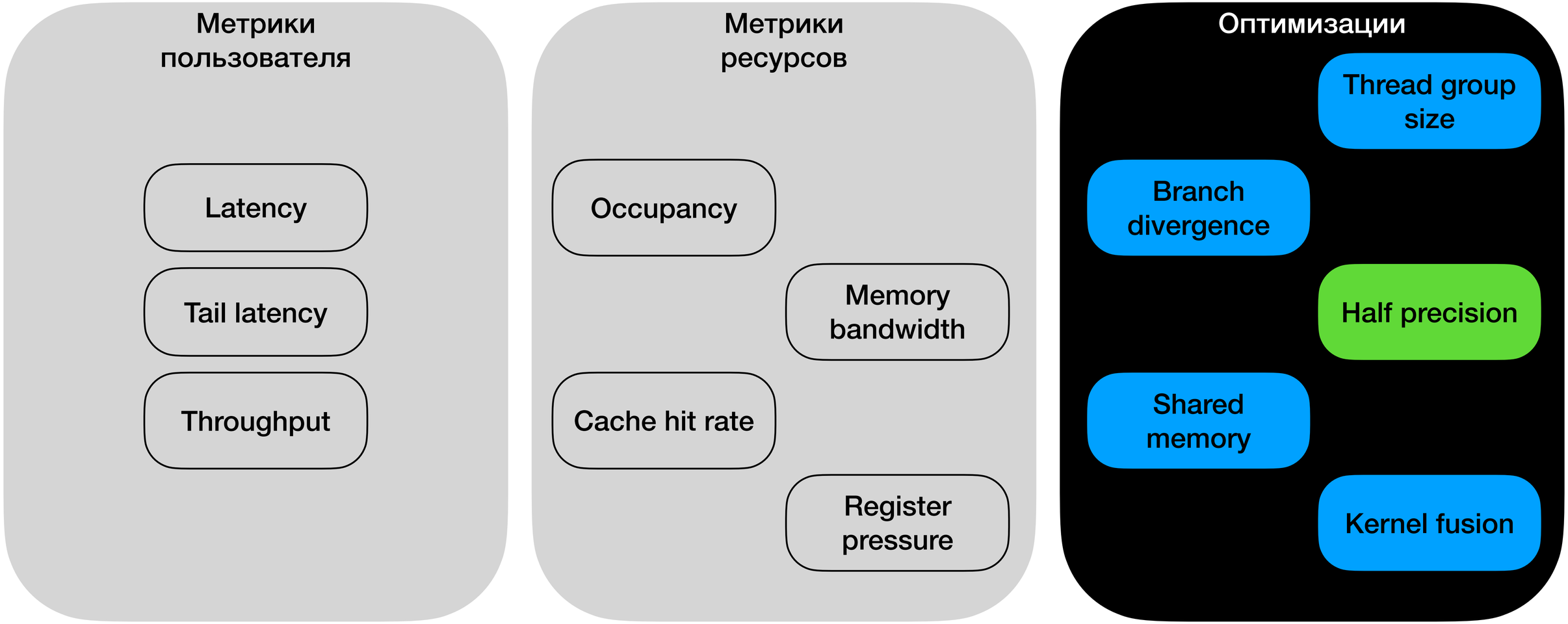
```
float4 sum = float4(0.0);
float weightSum = 0.0;

for (int ky = -radius; ky <= radius; ++ky) {
    for (int kx = -radius; kx <= radius; ++kx) {
        uint2 coord;
        coord.x = clamp(baseX + kx, 0, maxX - 1);
        coord.y = clamp(baseY + ky, 0, maxY - 1);

        float4 color = input.read(coord);
        float w = gaussEdge[ky + radius][kx + radius];

        sum += color * w;
        weightSum += w;
    }
}

float4 result = sum / weightSum;

output.write(result, gid);
```

Branch divergence

Ветвление потоков



Как измерить «производительность»?



Precision optimization

Оптимизация формата представления чисел

```
kernel void badGaussianBlurKernel(
    texture2d<float, access::read>  input  [[texture(0)]],
    texture2d<float, access::write> output [[texture(1)]],
    uint2 gid [[thread_position_in_grid]]
) {
    uint width = input.get_width();
    uint height = input.get_height();

    if (gid.x >= width ||
        gid.y >= height) {
        return;
    }

    int baseX = gid.x;
    int baseY = gid.y;
    int maxX = width;
    int maxY = height;

    constexpr int radius = 2;

    float gaussEdge[5][5] = {
        {1.0,  4.0,  7.0,  4.0, 1.0},
        {4.0, 16.0, 26.0, 16.0, 4.0},
        {7.0, 26.0, 41.0, 26.0, 7.0},
        {4.0, 16.0, 26.0, 16.0, 4.0},
        {1.0,  4.0,  7.0,  4.0, 1.0}
    };
```

```
float4 sum = float4(0.0);
float weightSum = 0.0;

for (int ky = -radius; ky <= radius; ++ky) {
    for (int kx = -radius; kx <= radius; ++kx) {
        uint2 coord;
        coord.x = clamp(baseX + kx, 0, maxX - 1);
        coord.y = clamp(baseY + ky, 0, maxY - 1);

        float4 color = input.read(coord);
        float w = gaussEdge[ky + radius][kx + radius];

        sum += color * w;
        weightSum += w;
    }
}

float4 result = sum / weightSum;

output.write(result, gid);
```

Precision optimization

Оптимизация формата представления чисел

```
kernel void badGaussianBlurKernel(
    texture2d<half, access::read>  input  [[texture(0)]],
    texture2d<float, access::write> output [[texture(1)]],
    uint2 gid [[thread_position_in_grid]]
) {
    uint width = input.get_width();
    uint height = input.get_height();

    if (gid.x >= width ||
        gid.y >= height) {
        return;
    }

    int baseX = gid.x;
    int baseY = gid.y;
    int maxX = width;
    int maxY = height;

    constexpr int radius = 2;

    constexpr half gaussEdge[5][5] = {
        {1.0,  4.0,  7.0,  4.0, 1.0},
        {4.0, 16.0, 26.0, 16.0, 4.0},
        {7.0, 26.0, 41.0, 26.0, 7.0},
        {4.0, 16.0, 26.0, 16.0, 4.0},
        {1.0,  4.0,  7.0,  4.0, 1.0}
    };
```

```
float4 sum = float4(0.0);
float weightSum = 0.0;

for (int ky = -radius; ky <= radius; ++ky) {
    for (int kx = -radius; kx <= radius; ++kx) {
        uint2 coord;
        coord.x = clamp(baseX + kx, 0, maxX - 1);
        coord.y = clamp(baseY + ky, 0, maxY - 1);

        half4 colorH = input.read(coord);
        half wH = gaussEdge[ky + radius][kx + radius];

        sum += float4(colorH) * float(wH);
        weightSum += float(wH);
    }
}

float4 result = sum / weightSum;

output.write(result, gid);
```

Precision optimization

Оптимизация формата представления чисел

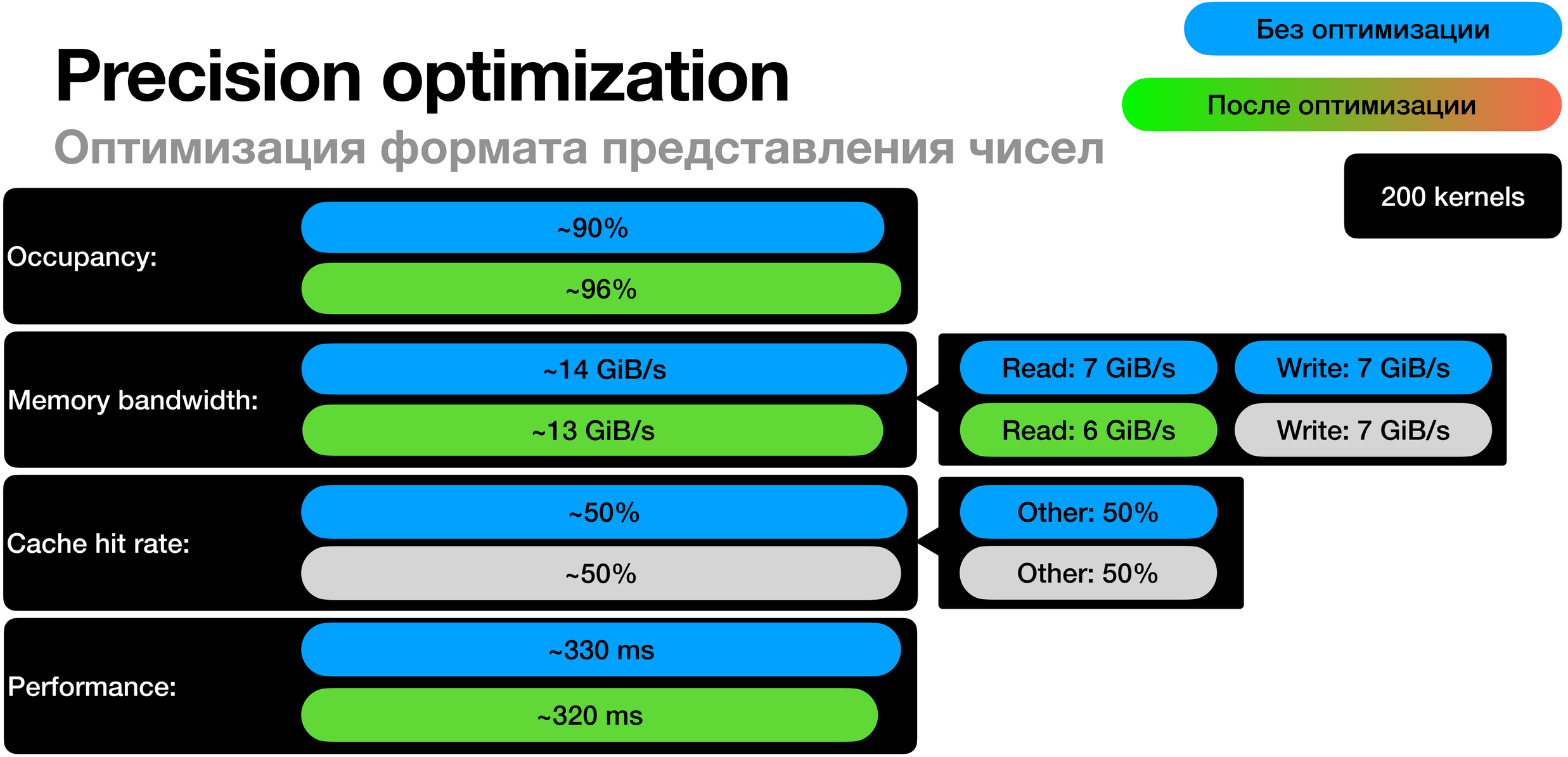
```
kernel void badGaussianBlurKernel(
    texture2d<half, access::read>  input  [[texture(0)]],
    texture2d<float, access::write> output [[texture(1)]],
    uint2 gid [[thread_position_in_grid]]
) {
    uint width = input.get_width();
    uint height = input.get_height();

    if (gid.x >= width ||
        gid.y >= height) {
        return;
    }

    int baseX = gid.x;
    int baseY = gid.y;
    int maxX = width;
    int maxY = height;

    constexpr int radius = 2;

    constexpr half gaussEdge[5][5] = {
        {1.0,  4.0,  7.0,  4.0, 1.0},
        {4.0, 16.0, 26.0, 16.0, 4.0},
        {7.0, 26.0, 41.0, 26.0, 7.0},
        {4.0, 16.0, 26.0, 16.0, 4.0},
        {1.0,  4.0,  7.0,  4.0, 1.0}
    };
```

```
float4 sum = float4(0.0);
float weightSum = 0.0;

for (int ky = -radius; ky <= radius; ++ky) {
    for (int kx = -radius; kx <= radius; ++kx) {
        uint2 coord;
        coord.x = clamp(baseX + kx, 0, maxX - 1);
        coord.y = clamp(baseY + ky, 0, maxY - 1);

        half4 colorH = input.read(coord);
        half wH = gaussEdge[ky + radius][kx + radius];

        sum += float4(colorH) * float(wH);
        weightSum += float(wH);
    }
}
```

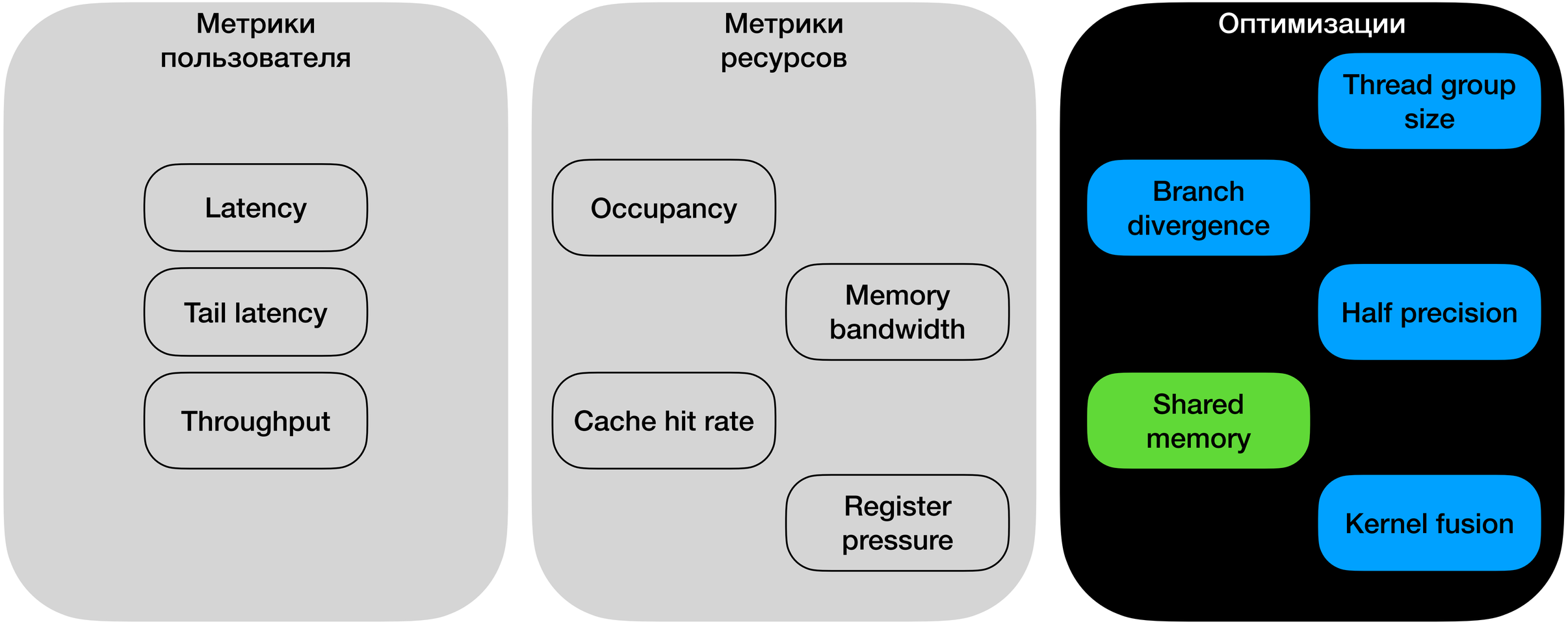
```
float4 result = sum / weightSum;

output.write(result, gid);
```

Precision optimization

Оптимизация формата представления чисел



Как измерить «производительность»?



Shared memory optimization

Работа с памятью группы потоков

```
kernel void badGaussianBlurKernel(
    texture2d<half, access::read>  input  [[texture(0)]],
    texture2d<float, access::write> output [[texture(1)]],
    uint2 gid [[thread_position_in_grid]]
) {
    uint width = input.get_width();
    uint height = input.get_height();

    if (gid.x >= width ||
        gid.y >= height) {
        return;
    }

    int baseX = gid.x;
    int baseY = gid.y;
    int maxX = width;
    int maxY = height;

    constexpr int radius = 2;

    constexpr half gaussEdge[5][5] = {
        {1.0,  4.0,  7.0,  4.0, 1.0},
        {4.0, 16.0, 26.0, 16.0, 4.0},
        {7.0, 26.0, 41.0, 26.0, 7.0},
        {4.0, 16.0, 26.0, 16.0, 4.0},
        {1.0,  4.0,  7.0,  4.0, 1.0}
    };
```

```
float4 sum = float4(0.0);
float weightSum = 0.0;

for (int ky = -radius; ky <= radius; ++ky) {
    for (int kx = -radius; kx <= radius; ++kx) {
        uint2 coord;
        coord.x = clamp(baseX + kx, 0, maxX - 1);
        coord.y = clamp(baseY + ky, 0, maxY - 1);

        half4 colorH = input.read(coord);
        half wH = gaussEdge[ky + radius][kx + radius];

        sum += float4(colorH) * float(wH);
        weightSum += float(wH);
    }
}

float4 result = sum / weightSum;

output.write(result, gid);
```

Shared memory optimization

Работа с памятью группы потоков

```
constant half kGaussian[5][5] = {
    { 1.0h,  4.0h,  7.0h,  4.0h, 1.0h },
    { 4.0h, 16.0h, 26.0h, 16.0h, 4.0h },
    { 7.0h, 26.0h, 41.0h, 26.0h, 7.0h },
    { 4.0h, 16.0h, 26.0h, 16.0h, 4.0h },
    { 1.0h,  4.0h,  7.0h,  4.0h, 1.0h }
};

constant float kWeightSum = 273.0f;
```

```
kernel void badGaussianBlurKernel(
    texture2d<half, access::read> input  [[texture(0)]],
    texture2d<float, access::write> output [[texture(1)]],

    uint2 gid [[thread_position_in_grid]],
    uint2 tid [[thread_position_in_threadgroup]],
    uint2 tgid [[threadgroup_position_in_grid]]
)
{
    constexpr uint RADIUS = 2;
    constexpr uint TG_SIZE_X = 16;
    constexpr uint TG_SIZE_Y = 16;

    constexpr uint TILE_SIZE_X = TG_SIZE_X + RADIUS * 2;
    constexpr uint TILE_SIZE_Y = TG_SIZE_Y + RADIUS * 2;

    const uint width  = input.get_width();
    const uint height = input.get_height();

    threadgroup half4 tile[TILE_SIZE_Y][TILE_SIZE_X];

    int baseX = int(tgid.x * TG_SIZE_X);
    int baseY = int(tgid.y * TG_SIZE_Y);
```

```
for (uint y = tid.y; y < TILE_SIZE_Y; y += TG_SIZE_Y) {
    for (uint x = tid.x; x < TILE_SIZE_X; x += TG_SIZE_X) {
        int globalX = baseX + int(x) - int(RADIUS);
        int globalY = baseY + int(y) - int(RADIUS);

        globalX = clamp(globalX, 0, int(width) - 1);
        globalY = clamp(globalY, 0, int(height) - 1);

        tile[y][x] = input.read(uint2(globalX, globalY));
    }
}

threadgroup_barrier(mem_flags::mem_threadgroup);

if (gid.x >= width || gid.y >= height)
    return;

uint localX = tid.x + RADIUS;
uint localY = tid.y + RADIUS;

float4 sum = float4(0.0);

for (int ky = -int(RADIUS); ky <= int(RADIUS); ++ky) {
    for (int kx = -int(RADIUS); kx <= int(RADIUS); ++kx) {
        half4 pixel = tile[localY + ky][localX + kx];
        half weight = kGaussian[ky + int(RADIUS)][kx + int(RADIUS)];

        sum += float4(pixel) * float(weight);
    }
}

float4 result = sum / kWeightSum;

output.write(result, gid);
```

Shared memory optimization

Работа с памятью группы потоков

```
constant half kGaussian[5][5] = {
    { 1.0h,  4.0h,  7.0h,  4.0h, 1.0h },
    { 4.0h, 16.0h, 26.0h, 16.0h, 4.0h },
    { 7.0h, 26.0h, 41.0h, 26.0h, 7.0h },
    { 4.0h, 16.0h, 26.0h, 16.0h, 4.0h },
    { 1.0h,  4.0h,  7.0h,  4.0h, 1.0h }
};

constant float kWeightSum = 273.0f;

kernel void badGaussianBlurKernel(
    texture2d<half, access::read> input  [[texture(0)]],
    texture2d<float, access::write> output [[texture(1)]],

    uint2 gid [[thread_position_in_grid]],
    uint2 tid [[thread_position_in_threadgroup]],
    uint2 tgid [[threadgroup_position_in_grid]]
)
{
    constexpr uint RADIUS = 2;
    constexpr uint TG_SIZE_X = 16;
    constexpr uint TG_SIZE_Y = 16;

    constexpr uint TILE_SIZE_X = TG_SIZE_X + RADIUS * 2;
    constexpr uint TILE_SIZE_Y = TG_SIZE_Y + RADIUS * 2;

    const uint width  = input.get_width();
    const uint height = input.get_height();

    threadgroup half4 tile[TILE_SIZE_Y][TILE_SIZE_X];

    int baseX = int(tgid.x * TG_SIZE_X);
    int baseY = int(tgid.y * TG_SIZE_Y);
```

```
for (uint y = tid.y; y < TILE_SIZE_Y; y += TG_SIZE_Y) {
    for (uint x = tid.x; x < TILE_SIZE_X; x += TG_SIZE_X) {
        int globalX = baseX + int(x) - int(RADIUS);
        int globalY = baseY + int(y) - int(RADIUS);

        globalX = clamp(globalX, 0, int(width) - 1);
        globalY = clamp(globalY, 0, int(height) - 1);

        tile[y][x] = input.read(uint2(globalX, globalY));
    }
}

threadgroup_barrier(mem_flags::mem_threadgroup);

if (gid.x >= width || gid.y >= height)
    return;

uint localX = tid.x + RADIUS;
uint localY = tid.y + RADIUS;

float4 sum = float4(0.0);

for (int ky = -int(RADIUS); ky <= int(RADIUS); ++ky) {
    for (int kx = -int(RADIUS); kx <= int(RADIUS); ++kx) {
        half4 pixel = tile[localY + ky][localX + kx];
        half weight = kGaussian[ky + int(RADIUS)][kx + int(RADIUS)];

        sum += float4(pixel) * float(weight);
    }
}

float4 result = sum / kWeightSum;

output.write(result, gid);
```

Shared memory optimization

Работа с памятью группы потоков

```
constant half kGaussian[5][5] = {
    { 1.0h,  4.0h,  7.0h,  4.0h, 1.0h },
    { 4.0h, 16.0h, 26.0h, 16.0h, 4.0h },
    { 7.0h, 26.0h, 41.0h, 26.0h, 7.0h },
    { 4.0h, 16.0h, 26.0h, 16.0h, 4.0h },
    { 1.0h,  4.0h,  7.0h,  4.0h, 1.0h }
};

constant float kWeightSum = 273.0f;

kernel void badGaussianBlurKernel(
    texture2d<half, access::read> input  [[texture(0)]],
    texture2d<float, access::write> output [[texture(1)]],

    uint2 gid [[thread_position_in_grid]],
    uint2 tid [[thread_position_in_threadgroup]],
    uint2 tgid [[threadgroup_position_in_grid]]
)
{
    constexpr uint RADIUS = 2;
    constexpr uint TG_SIZE_X = 16;
    constexpr uint TG_SIZE_Y = 16;

    constexpr uint TILE_SIZE_X = TG_SIZE_X + RADIUS * 2;
    constexpr uint TILE_SIZE_Y = TG_SIZE_Y + RADIUS * 2;

    const uint width  = input.get_width();
    const uint height = input.get_height();

    threadgroup half4 tile[TILE_SIZE_Y][TILE_SIZE_X];

    int baseX = int(tgid.x * TG_SIZE_X);
    int baseY = int(tgid.y * TG_SIZE_Y);
```

```
for (uint y = tid.y; y < TILE_SIZE_Y; y += TG_SIZE_Y) {
    for (uint x = tid.x; x < TILE_SIZE_X; x += TG_SIZE_X) {
        int globalX = baseX + int(x) - int(RADIUS);
        int globalY = baseY + int(y) - int(RADIUS);

        globalX = clamp(globalX, 0, int(width) - 1);
        globalY = clamp(globalY, 0, int(height) - 1);

        tile[y][x] = input.read(uint2(globalX, globalY));
    }
}

threadgroup_barrier(mem_flags::mem_threadgroup);

if (gid.x >= width || gid.y >= height)
    return;

uint localX = tid.x + RADIUS;
uint localY = tid.y + RADIUS;

float4 sum = float4(0.0);

for (int ky = -int(RADIUS); ky <= int(RADIUS); ++ky) {
    for (int kx = -int(RADIUS); kx <= int(RADIUS); ++kx) {
        half4 pixel = tile[localY + ky][localX + kx];
        half weight = kGaussian[ky + int(RADIUS)][kx + int(RADIUS)];

        sum += float4(pixel) * float(weight);
    }
}

float4 result = sum / kWeightSum;

output.write(result, gid);
```

Shared memory optimization

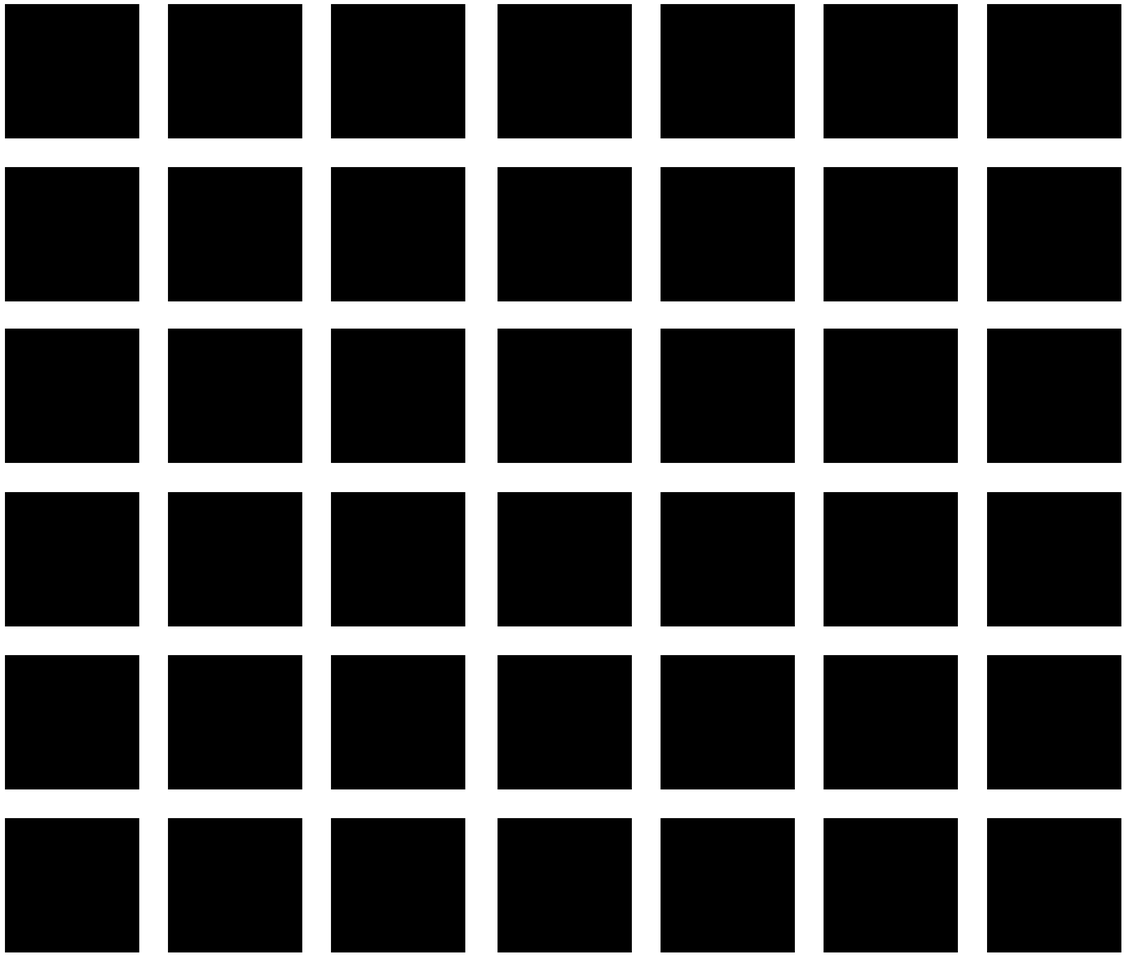
Работа с памятью группы потоков

```
constant half kGaussian[5][5] = {
    { 1.0h,  4.0h,  7.0h,  4.0h, 1.0h },
    { 4.0h, 16.0h, 26.0h, 16.0h, 4.0h },
    { 7.0h, 26.0h, 41.0h, 26.0h, 7.0h },
    { 4.0h, 16.0h, 26.0h, 16.0h, 4.0h },
    { 1.0h,  4.0h,  7.0h,  4.0h, 1.0h }
};

constant float kWeightSum = 273.0f;

kernel void badGaussianBlurKernel(
    texture2d<half, access::read> input  [[texture(0)]],
    texture2d<float, access::write> output [[texture(1)]],

    uint2 gid [[thread_position_in_grid]],
    uint2 tid [[thread_position_in_threadgroup]],
    uint2 tgid [[threadgroup_position_in_grid]]
)
{
    constexpr uint RADIUS = 2;
    constexpr uint TG_SIZE_X = 16;
    constexpr uint TG_SIZE_Y = 16;

    constexpr uint TILE_SIZE_X = TG_SIZE_X + RADIUS * 2;
    constexpr uint TILE_SIZE_Y = TG_SIZE_Y + RADIUS * 2;

    const uint width  = input.get_width();
    const uint height = input.get_height();

    threadgroup half4 tile[TILE_SIZE_Y][TILE_SIZE_X];

    int baseX = int(tgid.x * TG_SIZE_X);
    int baseY = int(tgid.y * TG_SIZE_Y);
```

```
for (uint y = tid.y; y < TILE_SIZE_Y; y += TG_SIZE_Y) {
    for (uint x = tid.x; x < TILE_SIZE_X; x += TG_SIZE_X) {
        int globalX = baseX + int(x) - int(RADIUS);
        int globalY = baseY + int(y) - int(RADIUS);

        globalX = clamp(globalX, 0, int(width) - 1);
        globalY = clamp(globalY, 0, int(height) - 1);

        tile[y][x] = input.read(uint2(globalX, globalY));
    }
}

threadgroup_barrier(mem_flags::mem_threadgroup);

if (gid.x >= width || gid.y >= height)
    return;

uint localX = tid.x + RADIUS;
uint localY = tid.y + RADIUS;

float4 sum = float4(0.0);

for (int ky = -int(RADIUS); ky <= int(RADIUS); ++ky) {
    for (int kx = -int(RADIUS); kx <= int(RADIUS); ++kx) {
        half4 pixel = tile[localY + ky][localX + kx];
        half weight = kGaussian[ky + int(RADIUS)][kx + int(RADIUS)];

        sum += float4(pixel) * float(weight);
    }
}

float4 result = sum / kWeightSum;

output.write(result, gid);
```

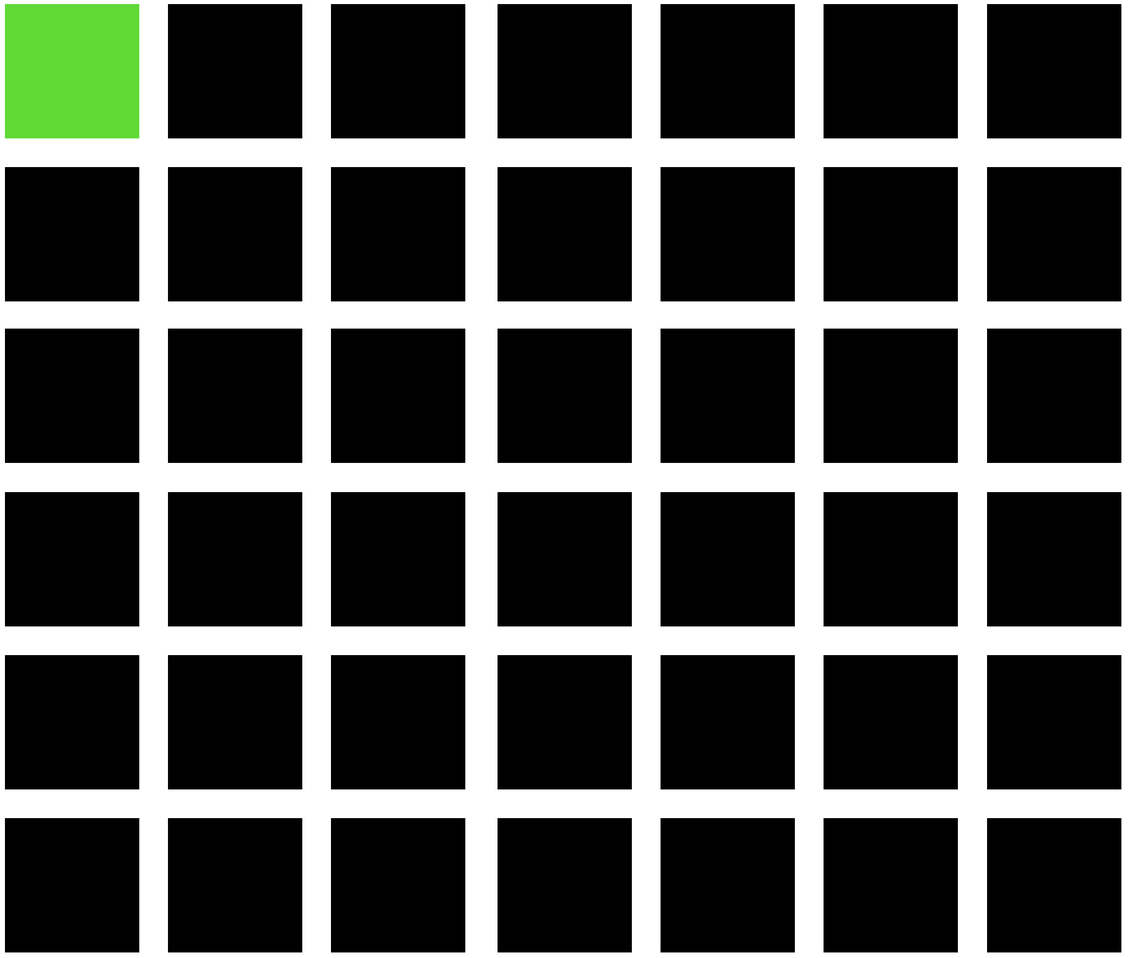
Shared memory optimization

Работа с памятью группы потоков



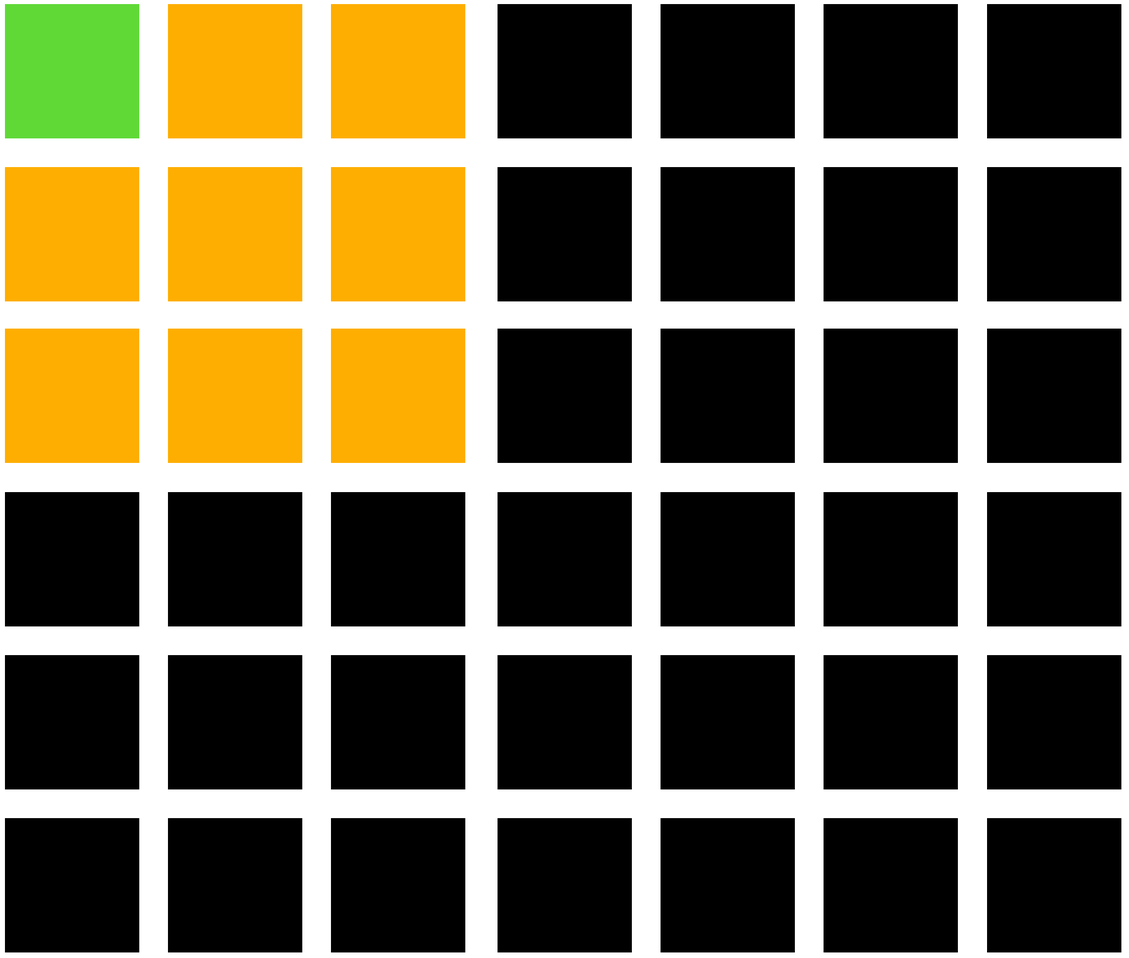
Shared memory optimization

Работа с памятью группы потоков



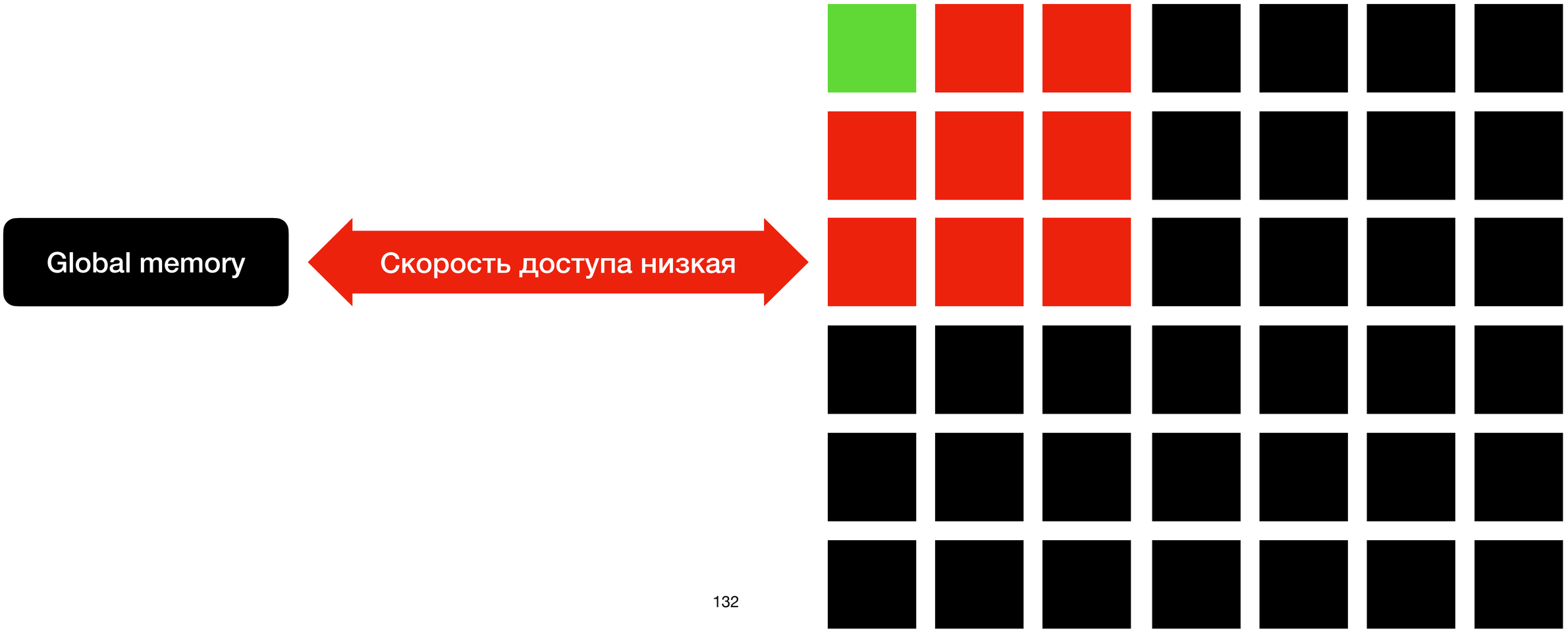
Shared memory optimization

Работа с памятью группы потоков



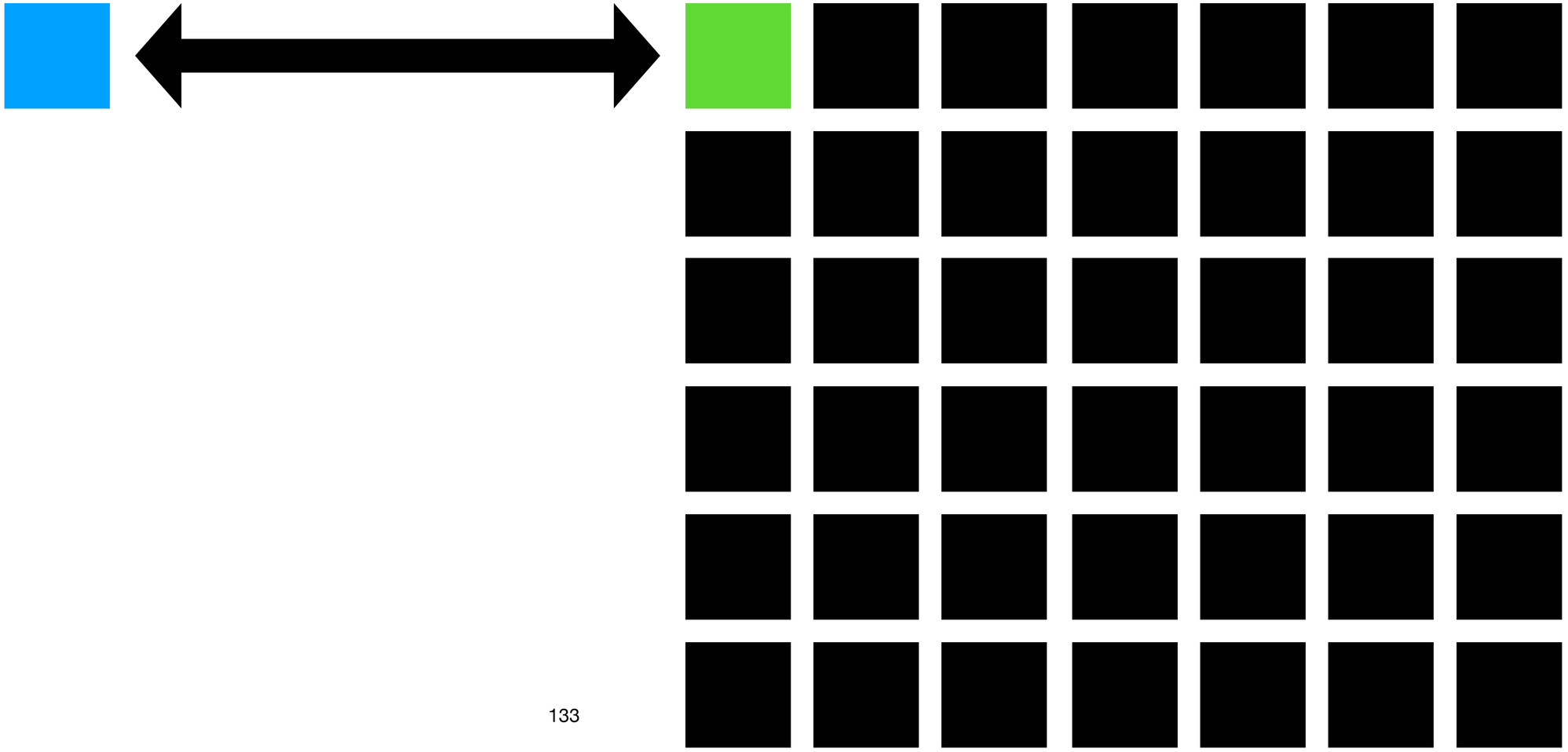
Shared memory optimization

Работа с памятью группы потоков



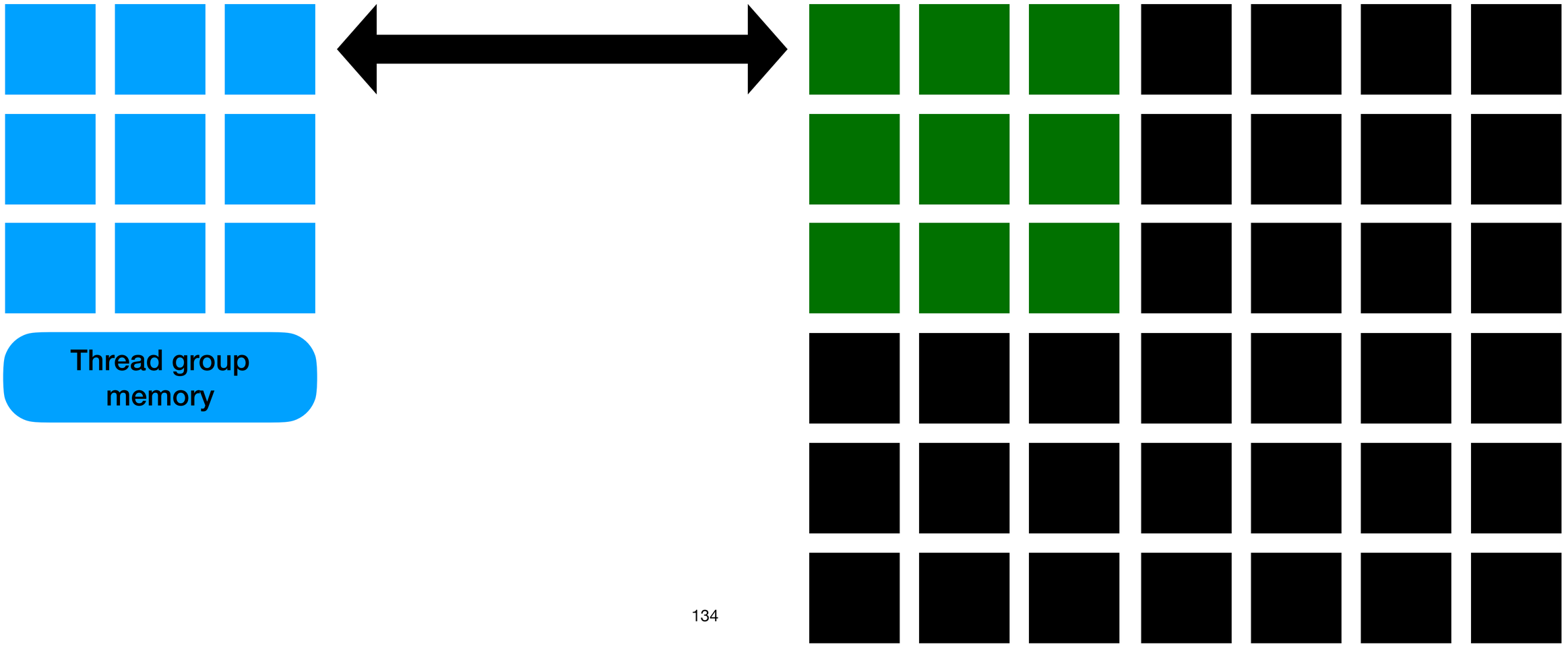
Shared memory optimization

Работа с памятью группы потоков



Shared memory optimization

Работа с памятью группы потоков



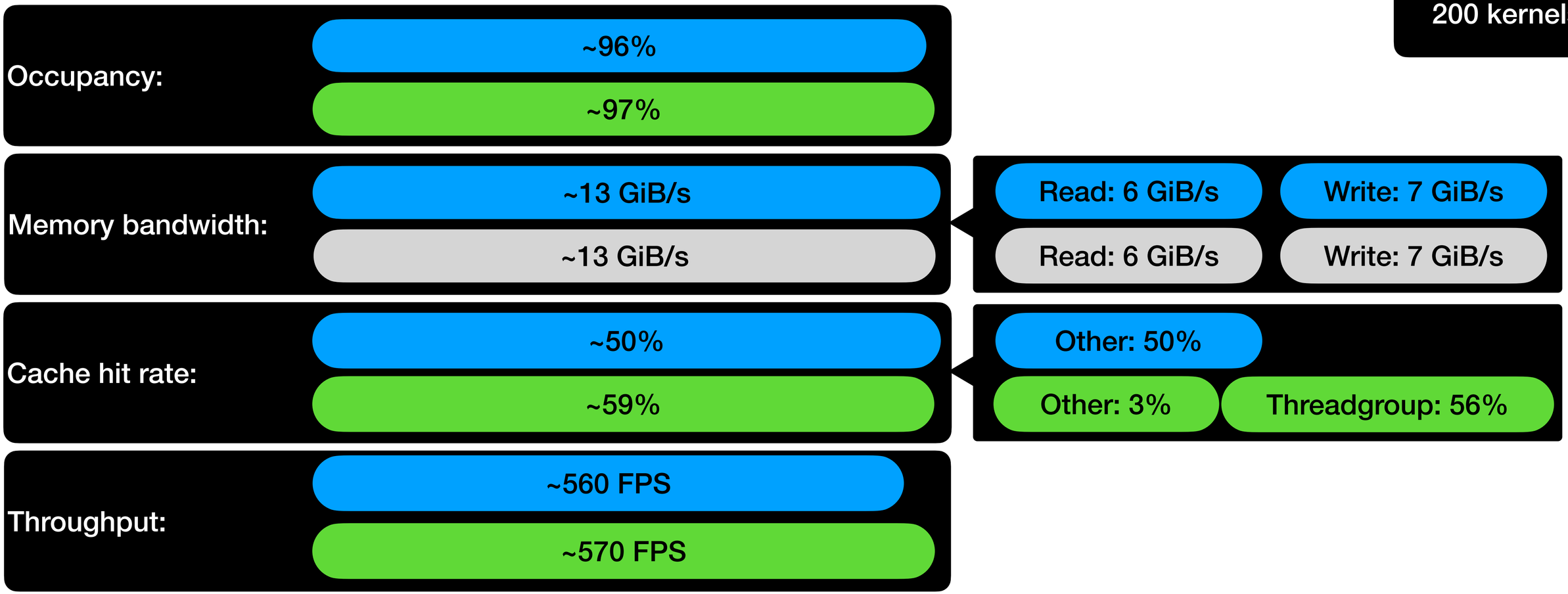
Shared memory optimization

Работа с памятью группы потоков

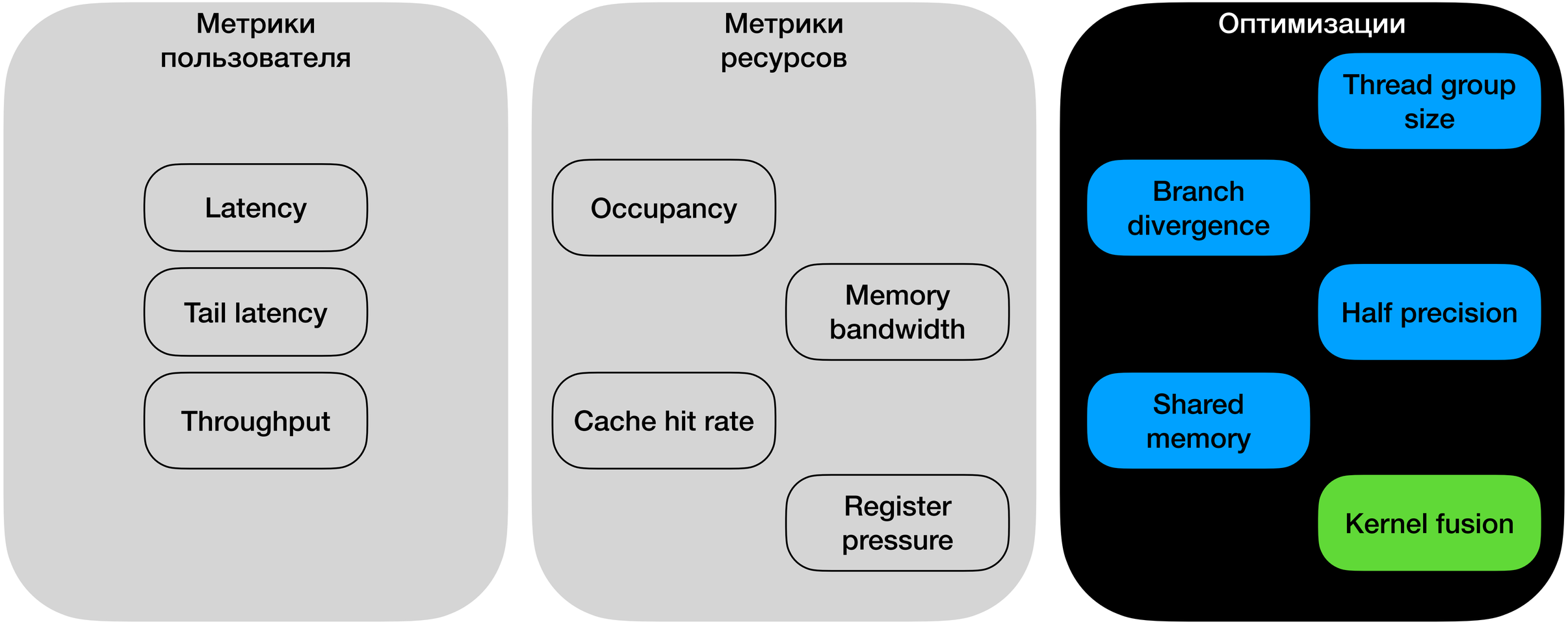
Без оптимизации

После оптимизации

200 kernels



Как измерить «производительность»?



Kernel fusion

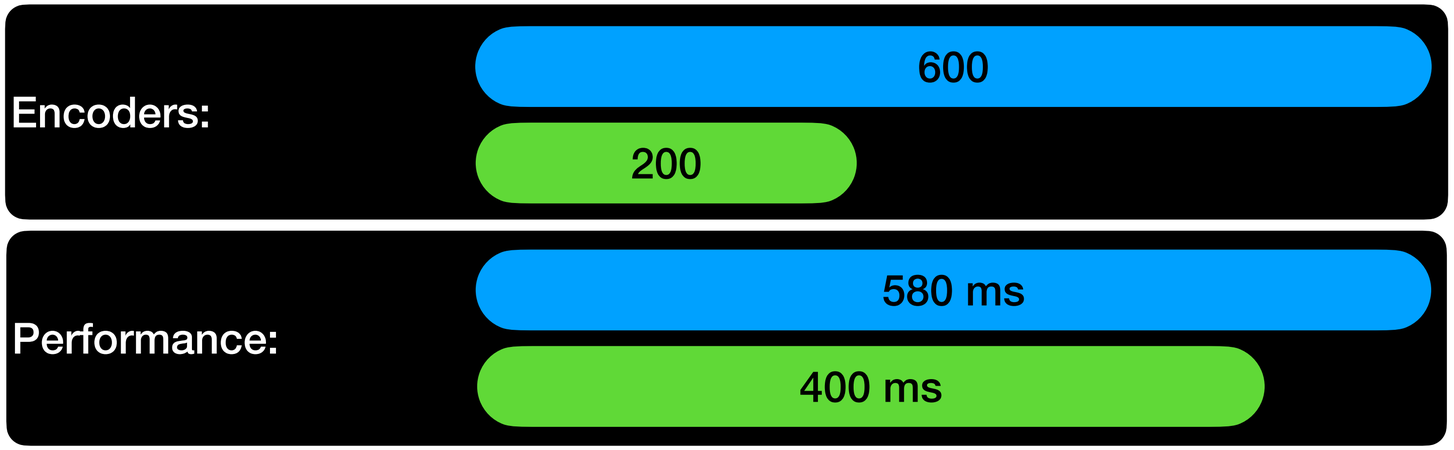
Объединение нескольких kernels

Kernel fusion

Объединение нескольких kernels

Kernel fusion

Объединение нескольких kernels



Без оптимизации

После оптимизации

600 vs 200 kernels

План доклада

- Вступление.
- Метрики пользователя.
- Метрики ресурсов и оптимизации.
- Какие еще скрываются тонкости?
- Подведение итогов.

Причины «плохого» результата

Основные проблемы

- Непонимание производительности.

Причины «плохого» результата

Основные проблемы

- Непонимание производительности.
- Неподходящие задачи.

Неподходящие задачи

Возможные пути решения проблем

- Одиночные или редкие операции.

Неподходящие задачи

Возможные пути решения проблем

- Одиночные или редкие операции.
- Небольшая потеря производительности, которую можно решить оптимизацией CPU кода.

Изображение 1500x1000 pixels

CPU Gaussian Blur без оптимизации: Latency 0.8s, P95 0.82s, FPS 1.24

CPU Gaussian Blur c vImage (Accelerate): Latency 0.15s, P95 0.15s, FPS 6.34

Неподходящие задачи

Возможные пути решения проблем

- Одиночные или редкие операции.
- Небольшая потеря производительности, которую можно решить оптимизацией CPU кода.
- Маленькие изображения.

Изображение 128x128 pixels

CPU Gaussian Blur без оптимизации: Latency 0.008s, P95 0.014s, FPS 121.4

Причины «плохого» результата

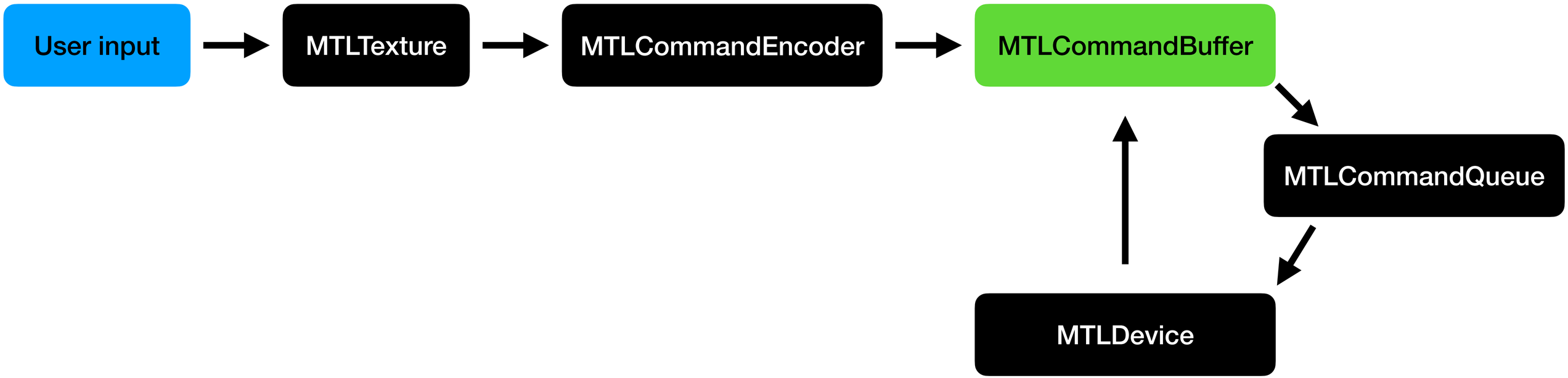
Основные проблемы

- Непонимание производительности.
- Неподходящие задачи.
- Синхронизация CPU/GPU.

Синхронизация CPU/GPU

Решение проблем связанных с синхронизацией

- Сведение к минимуму синхронизацию между CPU и GPU.



Синхронизация CPU/GPU

Решение проблем связанных с синхронизацией

- Сведение к минимуму синхронизацию между CPU и GPU.
- Настройка текстур с точки зрения расположения в памяти.

Синхронизация CPU/GPU

Решение проблем связанных с синхронизацией

```
func makeTexture(
    width: Int,
    height: Int,
    pixelFormat: MTLPixelFormat = .rgba8Unorm,
    usage: MTLTextureUsage = [.shaderRead, .shaderWrite]
) -> MTLTexture? {
    let descriptor = MTLTextureDescriptor.texture2DDescriptor(
        pixelFormat: pixelFormat,
        width: width,
        height: height,
        mipmapped: false
    )

    descriptor.usage = usage
    descriptor.storageMode = .shared

    return device.makeTexture(descriptor: descriptor)
}
```

Синхронизация CPU/GPU

Решение проблем связанных с синхронизацией

```
func makeTexture(
    width: Int,
    height: Int,
    pixelFormat: MTLPixelFormat = .rgba8Unorm,
    usage: MTLTextureUsage = [.shaderRead, .shaderWrite]
) -> MTLTexture? {
    let descriptor = MTLTextureDescriptor.texture2DDescriptor(
        pixelFormat: pixelFormat,
        width: width,
        height: height,
        mipmapped: false
    )

    descriptor.usage = usage
    descriptor.storageMode = .shared

    return device.makeTexture(descriptor: descriptor)
}
```

Синхронизация CPU/GPU

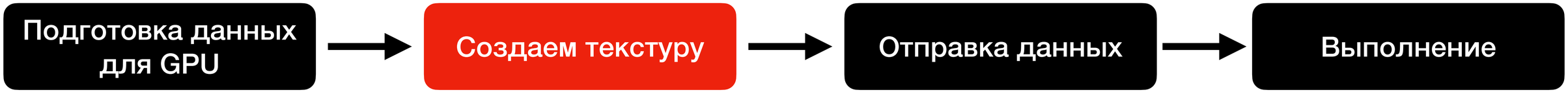
Решение проблем связанных с синхронизацией

```
func makeTexture(
    width: Int,
    height: Int,
    pixelFormat: MTLPixelFormat = .rgba8Unorm,
    usage: MTLTextureUsage = [.shaderRead, .shaderWrite]
) -> MTLTexture? {
    let descriptor = MTLTextureDescriptor.texture2DDescriptor(
        pixelFormat: pixelFormat,
        width: width,
        height: height,
        mipmapped: false
    )

    descriptor.usage = usage
    descriptor.storageMode = .shared

    return device.makeTexture(descriptor: descriptor)
}
```

```
func makeTexture(
    width: Int,
    height: Int,
    pixelFormat: MTLPixelFormat = .rgba8Unorm,
    usage: MTLTextureUsage = [.shaderRead, .shaderWrite]
) -> MTLTexture? {
    let descriptor = MTLTextureDescriptor.texture2DDescriptor(
        pixelFormat: pixelFormat,
        width: width,
        height: height,
        mipmapped: false
    )

    descriptor.usage = usage
    descriptor.storageMode = .private

    return device.makeTexture(descriptor: descriptor)
}
```

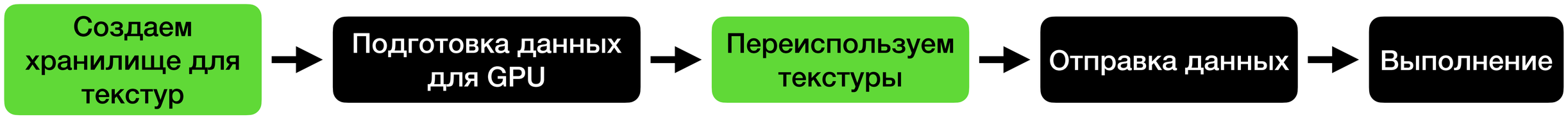
Синхронизация CPU/GPU

Решение проблем связанных с синхронизацией



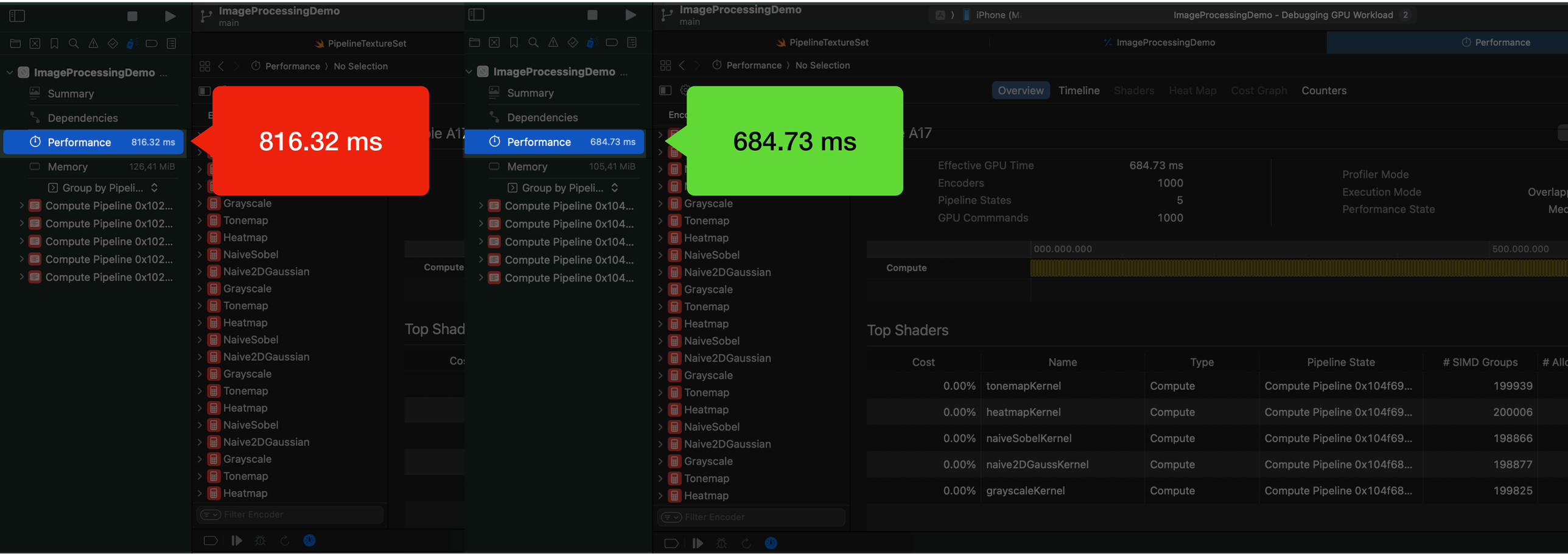
Синхронизация CPU/GPU

Решение проблем связанных с синхронизацией



Синхронизация CPU/GPU

Решение проблем связанных с синхронизацией



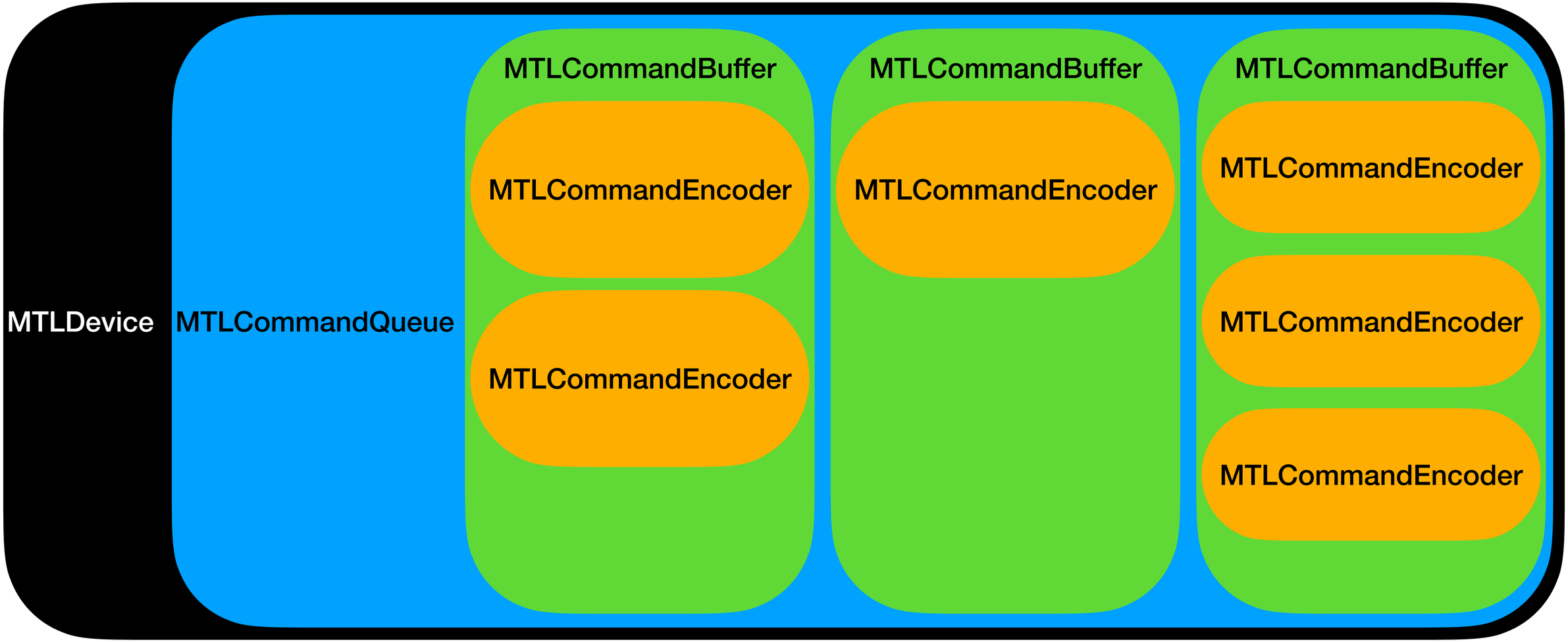
Причины «плохого» результата

Основные проблемы

- Непонимание производительности.
- Неподходящие задачи.
- Синхронизация CPU/GPU.
- Неправильная организация pipeline.

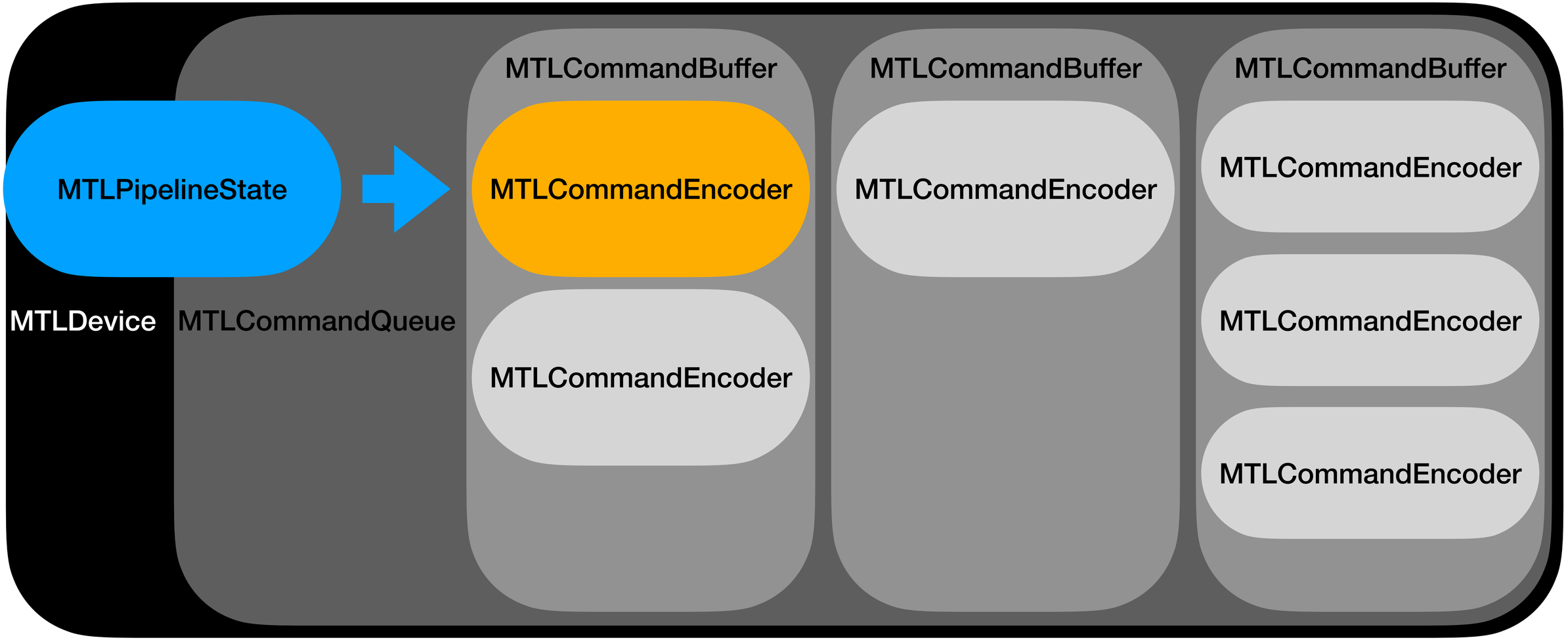
Пройдемся по понятиям

Основные объекты Metal



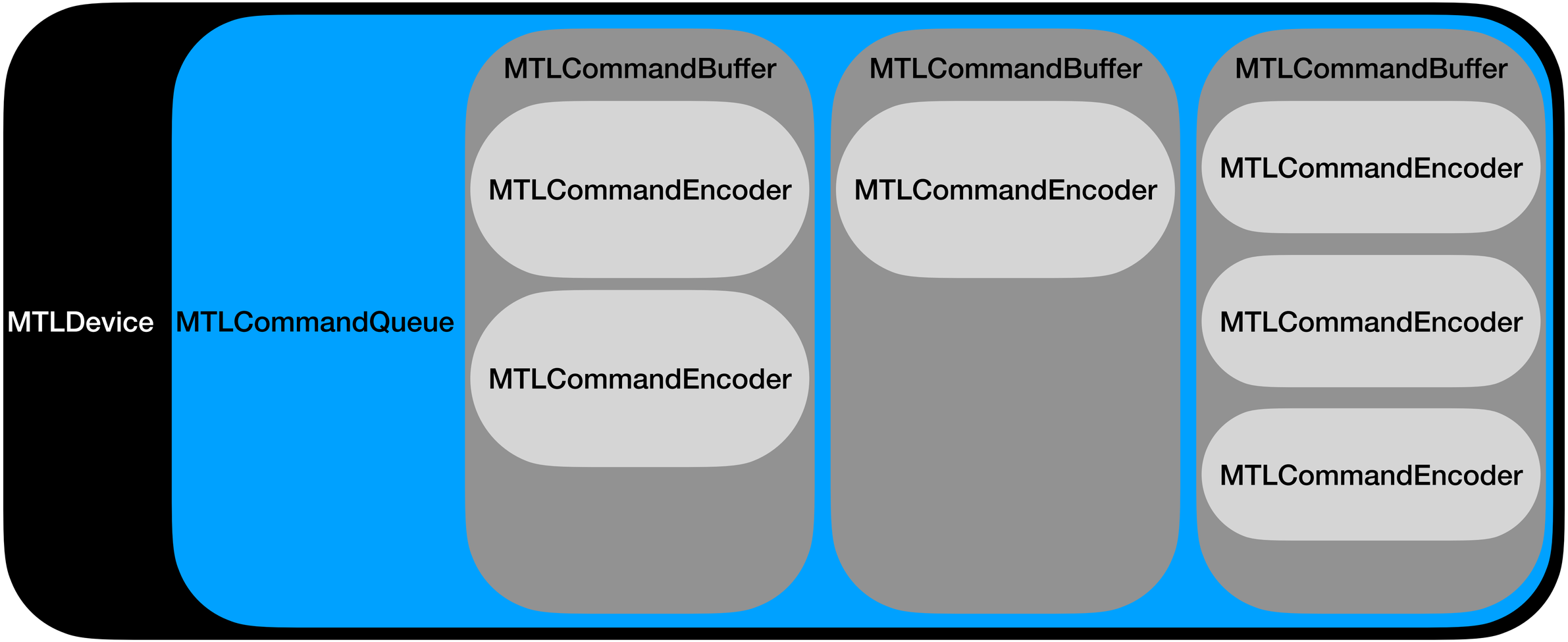
Пройдемся по понятиям

Основные объекты Metal



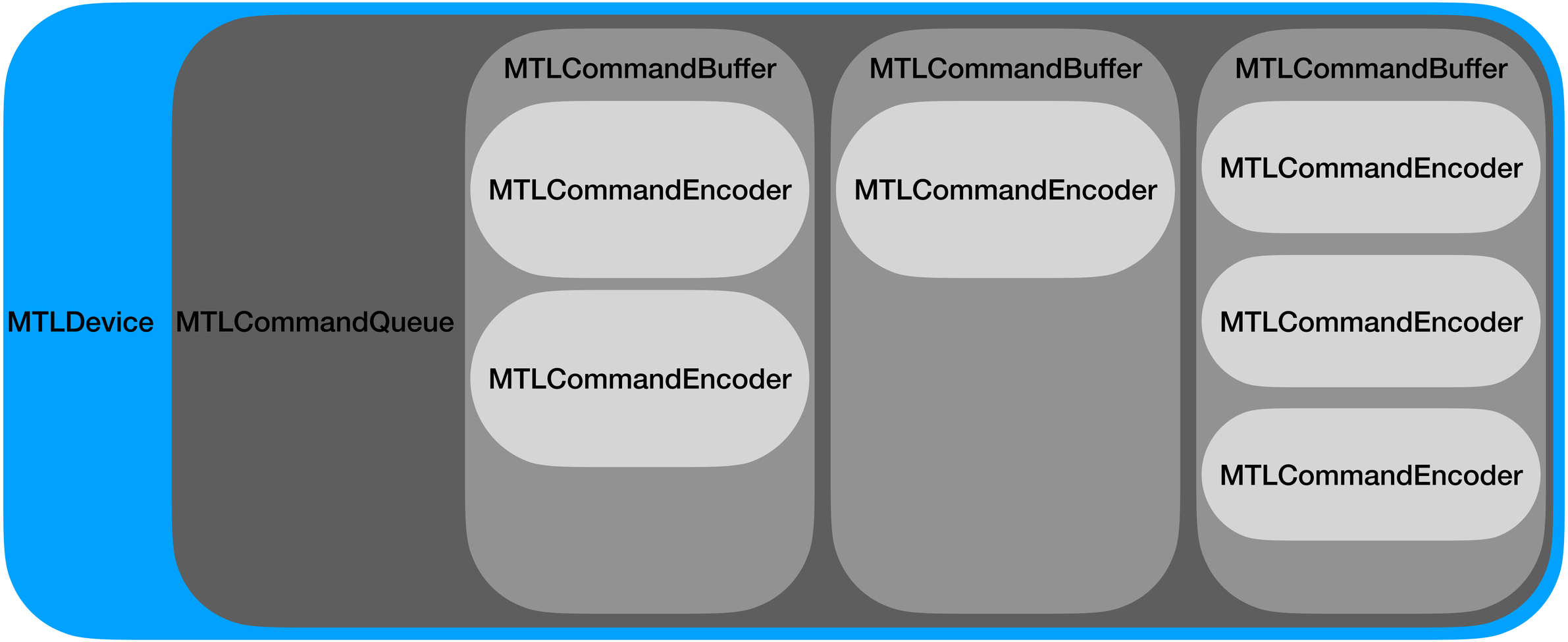
Пройдемся по понятиям

Основные объекты Metal



Пройдемся по понятиям

Основные объекты Metal



Синхронизация CPU/GPU

Решение проблем связанных с синхронизацией

План доклада

- Вступление.
- Метрики пользователя.
- Метрики ресурсов и оптимизации.
- Какие еще скрываются тонкости?
- Подведение итогов.

Подводим итоги

- **Производительность** - это четкий набор инструментов и действий.

Подводим итоги

- **Производительность** - это четкий набор инструментов и действий.
- **Metal** сам по себе **не равно ускорение**, а скорее **ресурс с большим потенциалом** и возможностями при правильном подходе.

Подводим итоги

- **Производительность** - это четкий набор инструментов и действий.
- **Metal** сам по себе **не равно ускорение**, а скорее **ресурс с большим потенциалом** и возможностями при правильном подходе.
- **Подход** и **действия** при построении архитектуры **Metal** напрямую зависят от **задач**, которые вы хотите решить.

Ресурсы

Metal by Tutorials - Kodeco Team

Metal Programming Guide - Janie Clayton

Демо проект

@MATRIPLEXIM