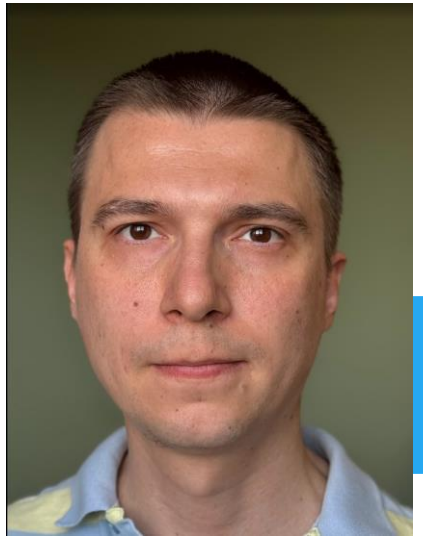
ALTENAR

Логирование в .NET с помощью Serilog производительность отказоустойчивость

Владимир Куропатка



Владимир Куропатка

10+ Лет опыта разработки на .net

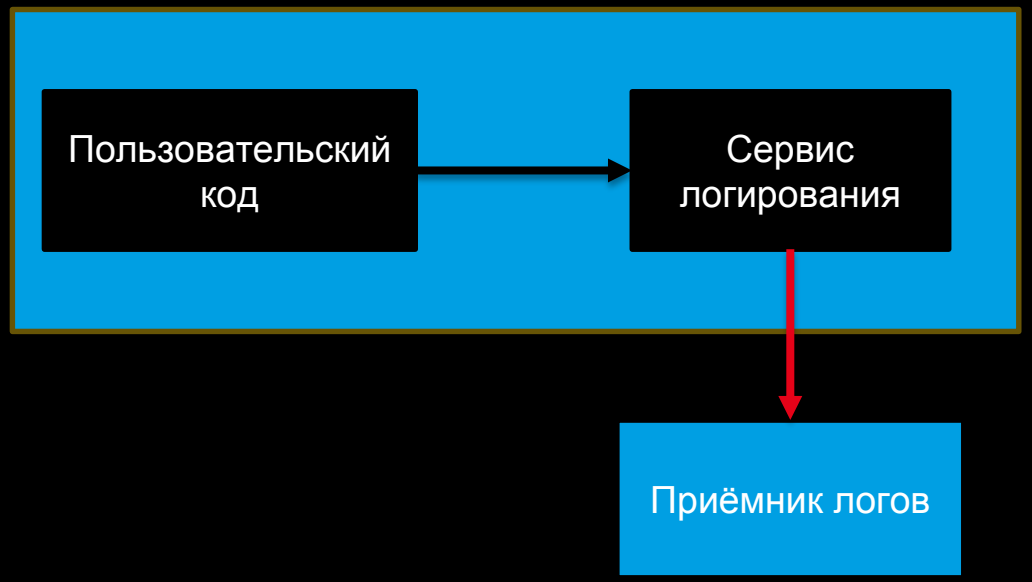
Старший BACKEND-разработчик в altenar

План доклада

- Базовые понятия
- Особенности работы Serilog с удалёнными приёмниками
- Способы повышения отказоустойчивости
- Возможности повышения производительности

Что такое логирование

Логирование — это процесс формирования и сохранения информации о ходе работы приложений и событий, связанных с выполнением приложения.



цель логирования

- Мониторинг параметров работы приложения
- Сбор метрик производительности
- Расследование инцидентов
- Отладка

Почему тема логов Актуальна?

- Неотъемлемая часть любого приложения
- Присутствует независимо от домена и архитектуры
- Влияет на производительность не самым очевидным образом
- Может стать причиной отказа при повышенной нагрузке

Чем интересен Serilog?

- Регулярно обновляется
- Активное комьюнити
- Множество опций форматирования
- Множество поддерживаемых приёмников
- Рекомендуется к использованию Microsoft

Что с логами в altenar

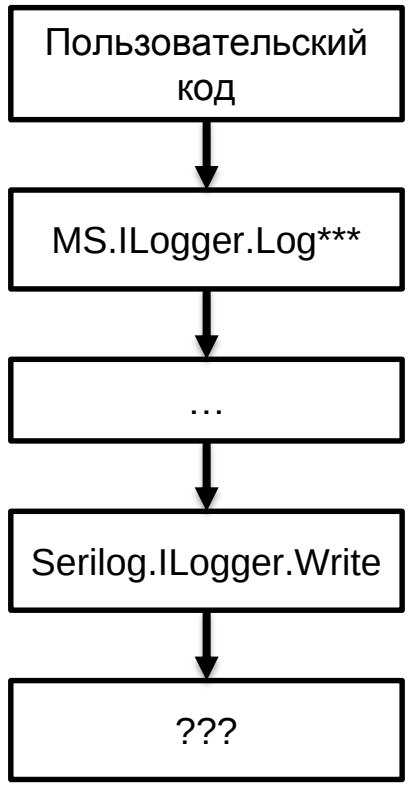
- Абстракции и расширения Microsoft
- Serilog под капотом
- Google Cloud Logging в качестве хранилища
- Очень много логов ☹

Почему мы за это взялись?

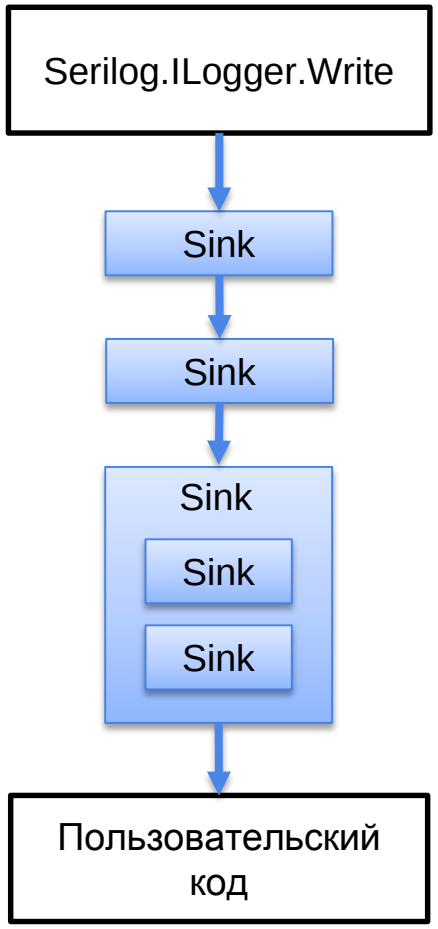
- В часы пиковой нагрузки сервисы падали с OOM
- В дампах памяти обнаружилось много записей логов
- Нужно было понять, почему забивается память и как этого избежать

Как мы логируем

```
ILogger<LoginController> _logger;

_logger.LogInformation(
    "User {userId} logged in successfully.",
    userId);
// И всё. Или нет?
```



Что внутри serilog



- Конечная точка обработки лога – приёмник (Sink)
- Приёмников может быть несколько
- Один приёмник может включать в себя несколько приёмников
- Приёмники можно разделить на синхронные и асинхронные

Синхронные приёмники

`ILogEventSink` – интерфейс приёмника, синхронно фиксирующего одиночные записи

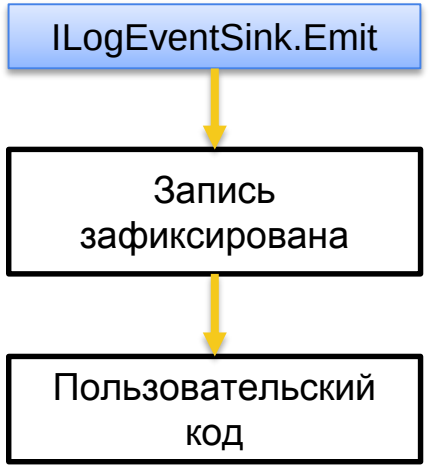
```
public interface ILogEventSink
{
    void Emit(LogEvent logEvent);
}
```

АСинхронные приёмники

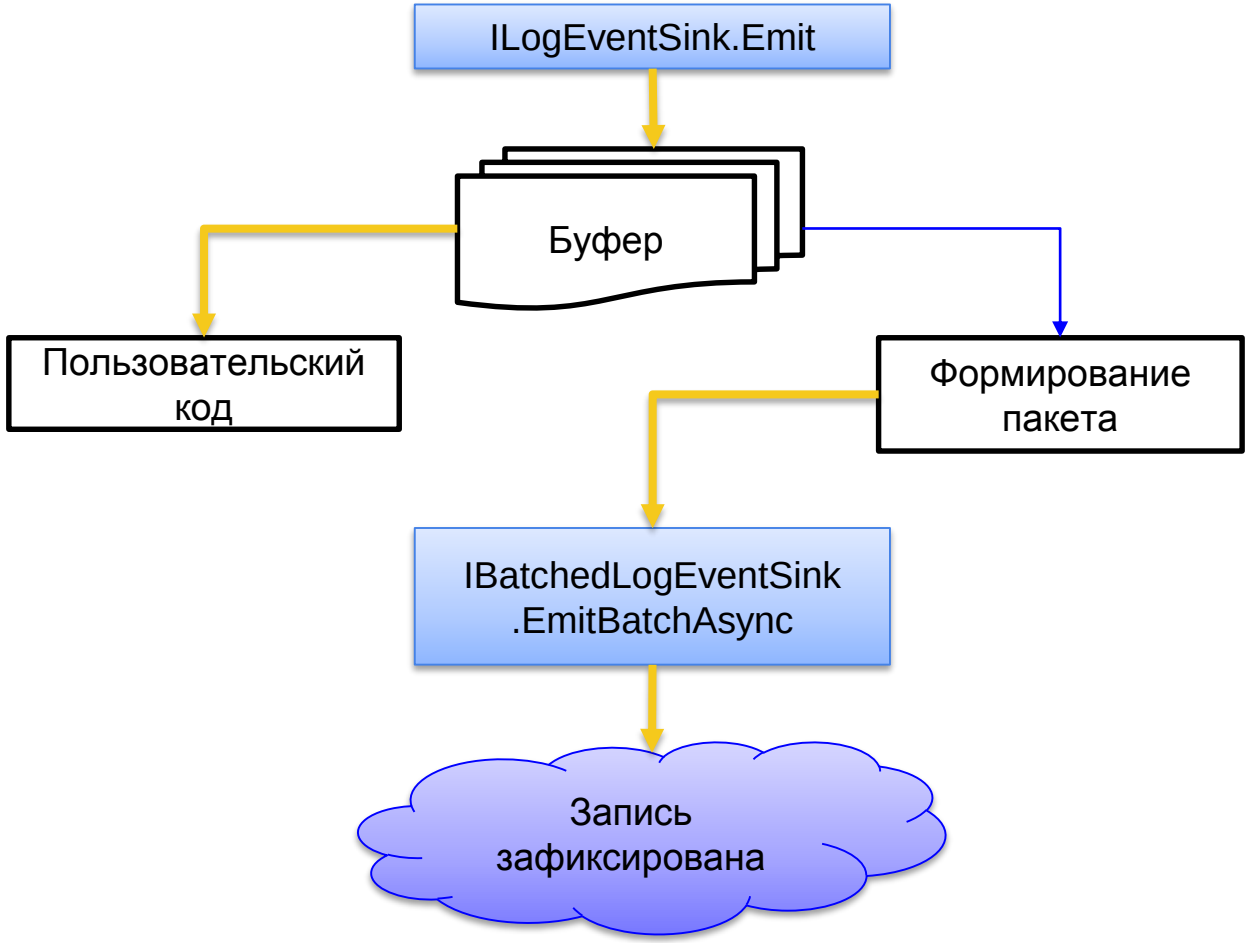
`IBatchedLogEventSink` – интерфейс приёмника асинхронной фиксации пакетов записей

```
public interface IBatchedLogEventSink
{
    Task EmitBatchAsync(
        IReadOnlyCollection<LogEvent> batch);
}
```

Синхронный приёмник



АСинхронный приёмник



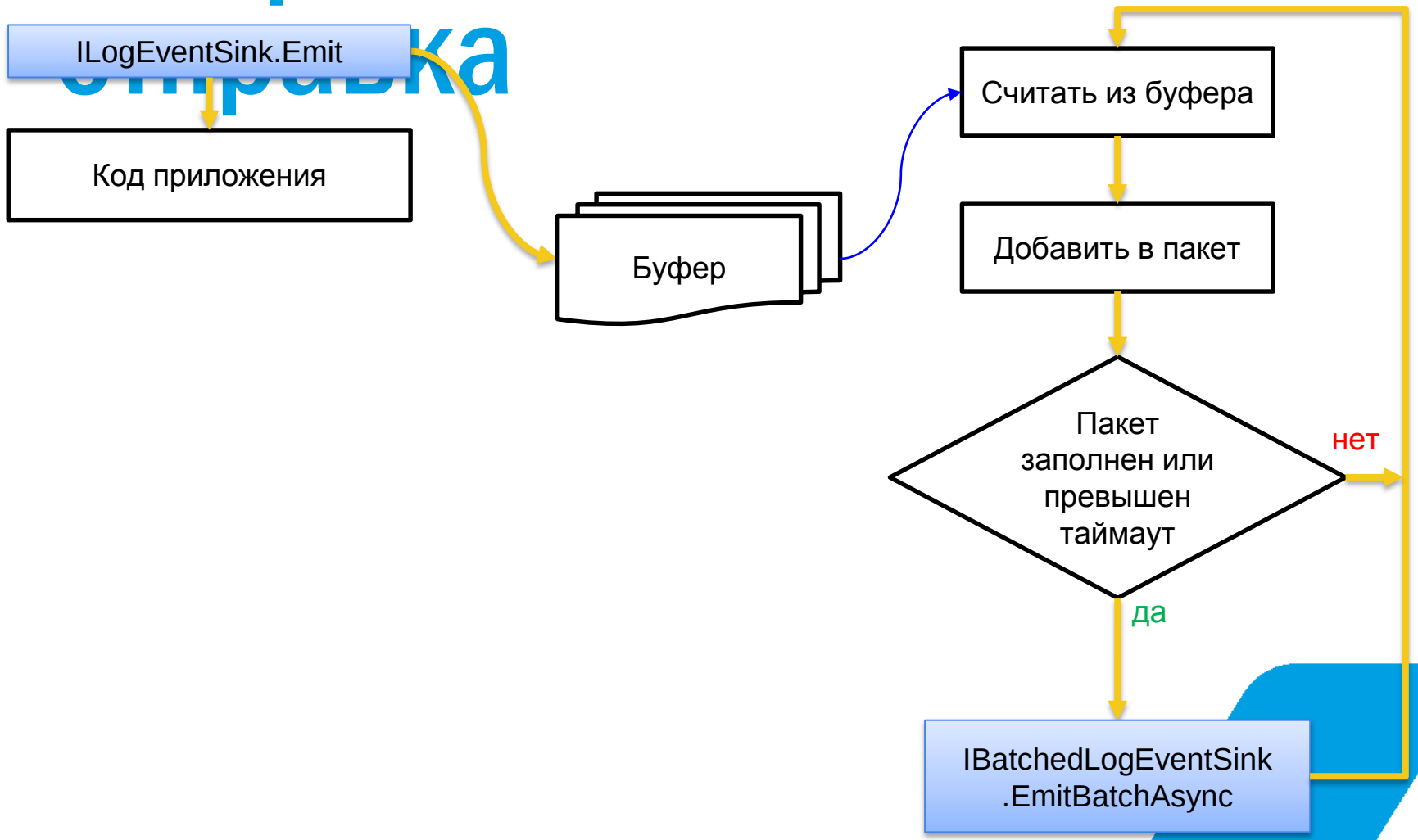
Почему нельзя обойтись отправкой одиночных записей?

- Возможный троттлинг на стороне приёмника
- Повышенная нагрузка на сеть
- Сохранение порядка записи
- Повышенная стоимость обслуживания
- Необходимость ожидания завершения операций или использования подхода «выстрелил и убежал»

Преимущества пакетной отправки

- Быстрая запись в буфер, нет нужды ожидать
- Сглаживание пиков нагрузки
- Компромисс между задержкой отправки и частотой операций I/O

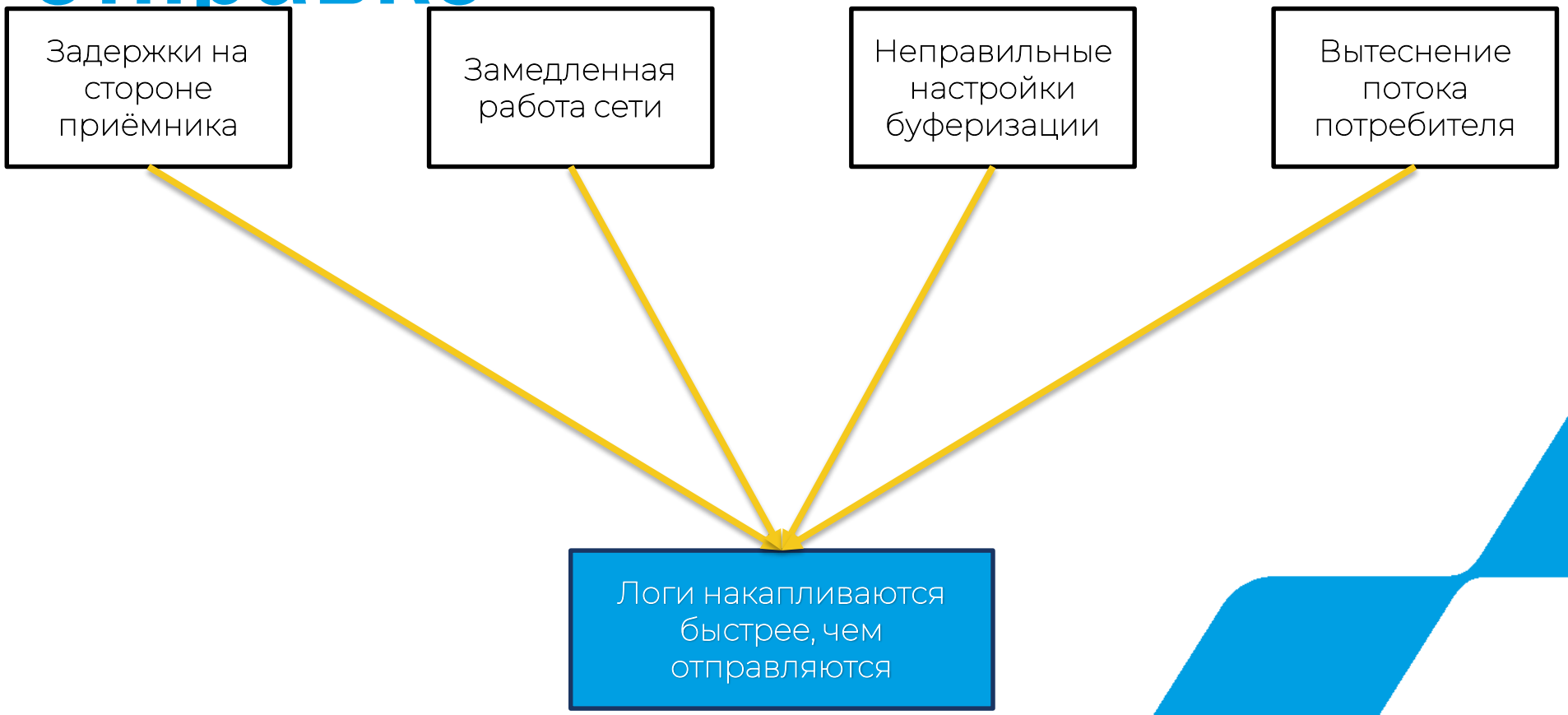
Как работает пакетная отправка



Недостатки пакетной отправки

- Задержка доставки логов в приёмник
- Потеря содержимого буфера при сбое
- Необходимость хранения буфера в оперативной памяти

Проблемы при пакетной отправке



Возможности настройки

Размер пакета - `BatchSizeLimit`

- Требует правильного выбора в зависимости от конкретного приёмника
- Значение по умолчанию - 1000

Возможности настройки

Максимальное время формирования пакета - `BufferingTimeLimit`

- Требует правильного выбора в зависимости от конкретного приёмника
- Значение по умолчанию – 1 секунда

Возможности настройки

Максимальное время попыток повторной отправки при ошибке - `RetryTimeLimit`

- В зависимости от количества ошибок может быть удалён текущий пакет или весь буфер
- Не работает в случае повышенных задержек при отправке
- Значение по умолчанию – 10 секунд

Возможности настройки

Максимальное количество записей в буфере - `QueueLimit`

- Если буфер заполнен на момент попытки записи, запись будет потеряна
- При потере записей нет телеметрии
- Наиболее эффективная настройка с точки зрения избегания переполнения кучи

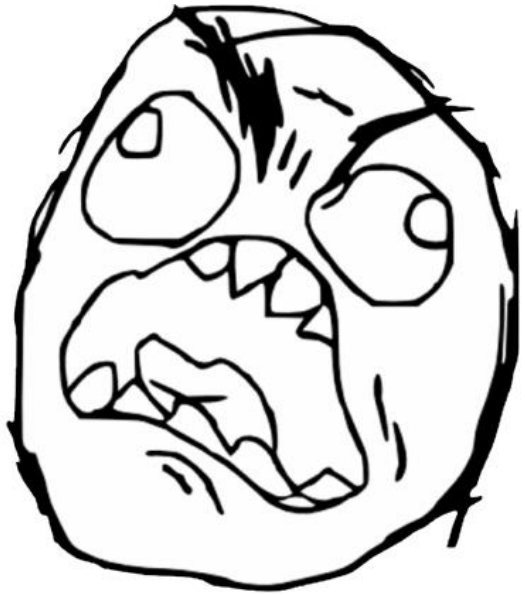
Критерии оптимизации

Максимальное количество записей в буфере - `QueueLimit`

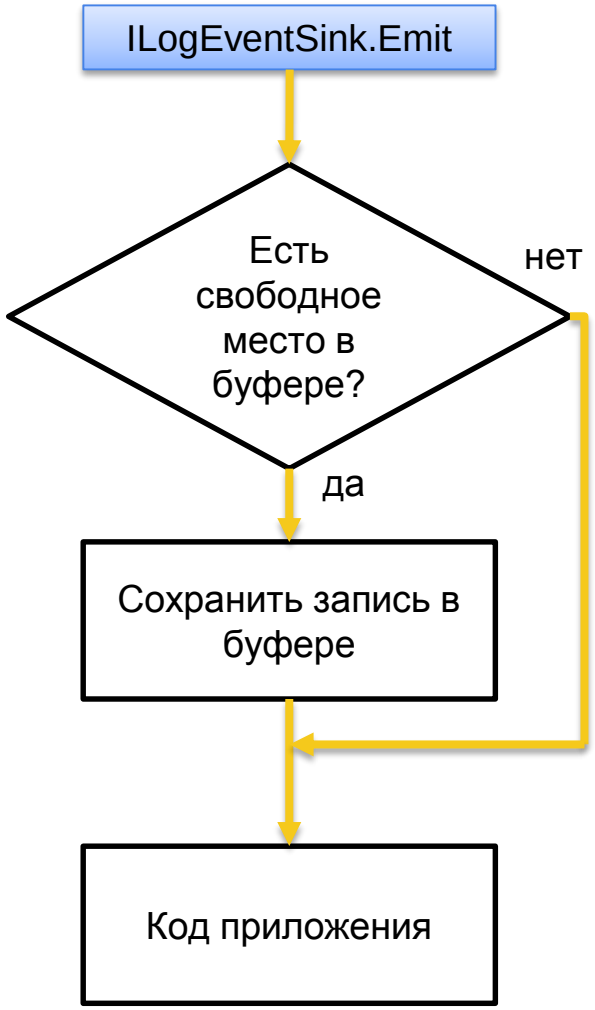
- Снижение вероятности отказа приложения по вине подсистемы логирования
- Минимизация потерь логов
- Снижение влияния подсистемы логирования на производительность приложения

Может, писать меньше логов?

```
[HttpGet]
public IEnumerable<WeatherForecast> Get()
{
    _logger.LogTrace("О, мы оказались в методе
{MethodName}!",
        nameof(Get));

    _logger.LogTrace("А сейчас мы пойдем в
{MethodName}!",
        nameof(_weatherService.GetForecast));

    var result = _weatherService.GetForecast();

    _logger.LogTrace("Успешно вернулись из
{MethodName}!",
        nameof(_weatherService.GetForecast));

    _logger.LogTrace("Всё, метод {MethodName}
успешно выполнился!",
        nameof(Get));
}
```



Ограничение размера буфера



Преимущества

- Решает проблему заполнения оперативной памяти логами
- Требует только изменения конфигурации

Недостатки

- Безвозвратная потеря логов
- Нет возможности узнать о потерянных логах

Проблемы удалённых приёмников

Ошибки

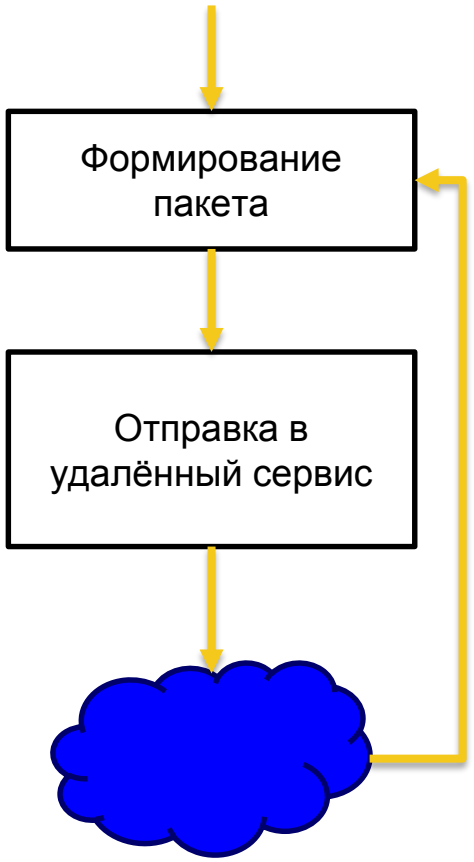
- Обрабатываются логикой Serilog с использованием настройки RetryTimeLimit
- Есть возможность отследить на стороне кода приложения

Задержки

- Приводят к накоплению логов в буфере
- Нет возможности отследить и обработать в коде приложения

Промежуточный буфер

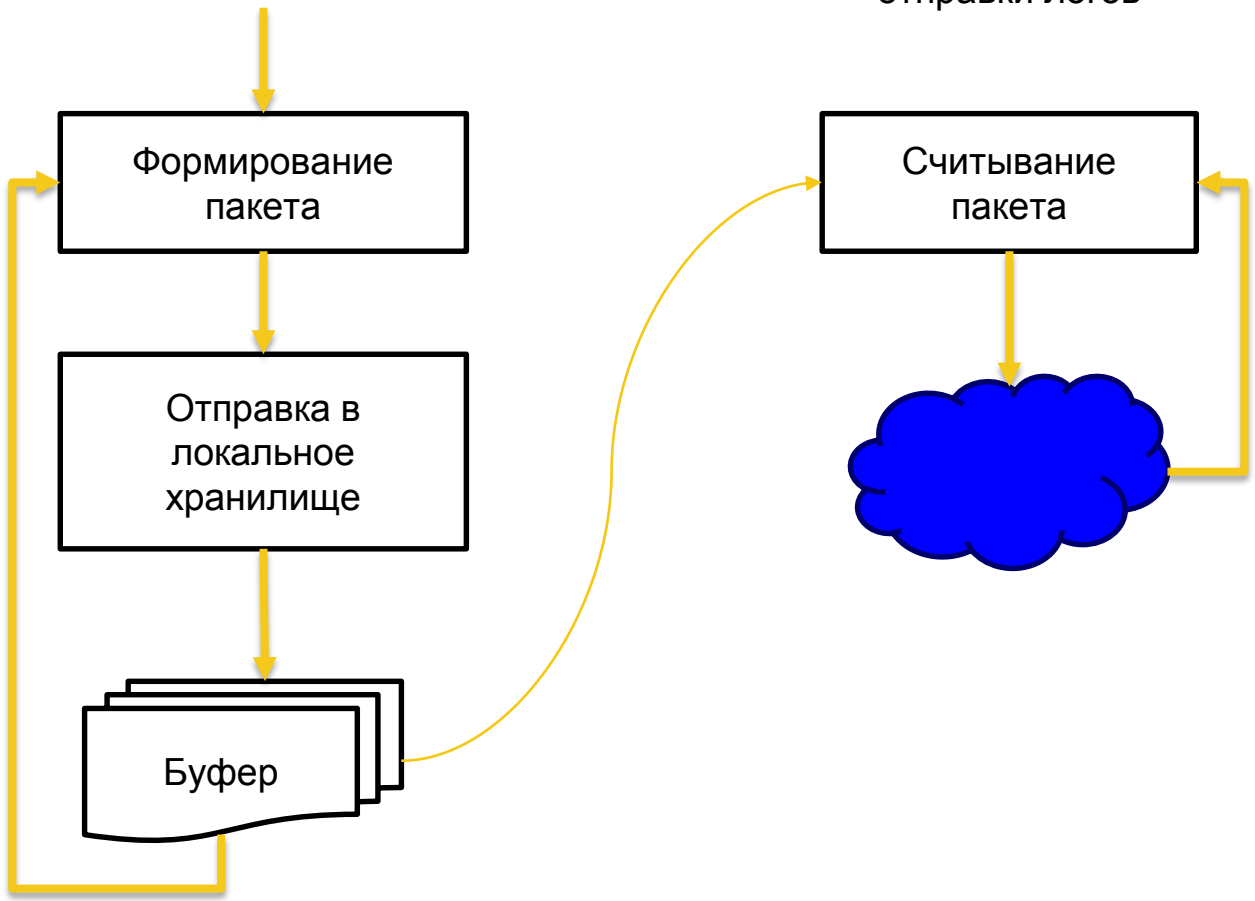
Поток потребителя логов



Промежуточный буфер

Поток потребителя логов

Отдельное приложение для отправки логов



Промежуточный буфер

Преимущества

- Решает проблему задержек на стороне приёмника
- Нивелирует влияние ошибок на стороне приёмника
- Нивелирует влияние задержек сети

Промежуточный буфер

Недостатки

- Дополнительные затраты на разработку
- Усложнение архитектуры
- Требование дополнительных аппаратных ресурсов

Разгрузка потребителя логов

Обязанности потребителя по умолчанию

- Преобразование логов из формата Serilog в формат приёмника
- Сериализация
- Передача

Как улучшить?

- Отказ от преобразования
- Сериализация логов «как есть» с упором на производительность

Разгрузка потребителя логов: тест

Вариант «как есть»:

- Логика преобразования и сериализации Google с github
- Код отправки в облака удалён

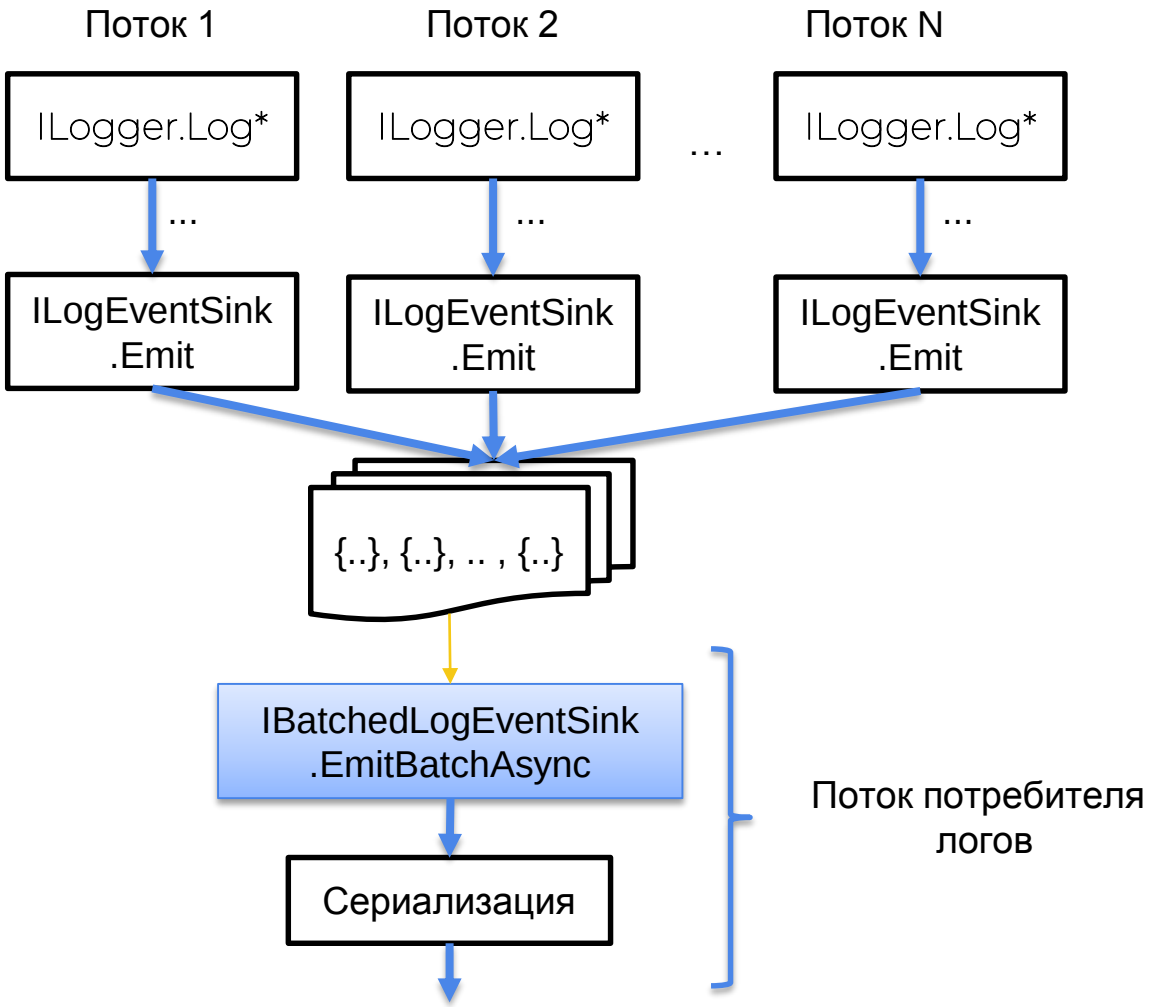
Вариант «с разгрузкой»:

- Serilog.Formatting.Compact в качестве форматера

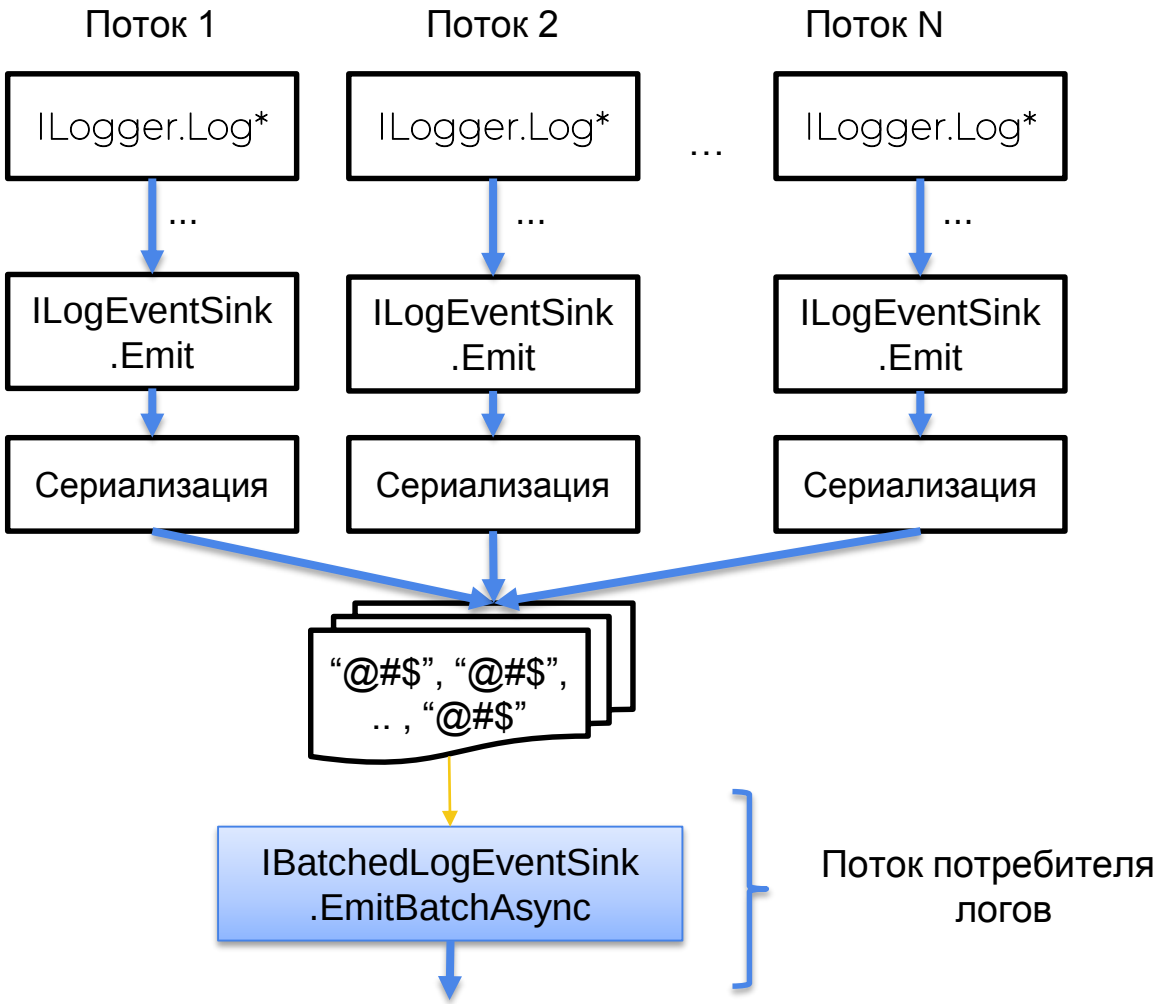
Результат для 1000 объектов LogEvent

Method	Mean	Allocated
SerializeCompact	422.7 us	832.61 KB
SerializeGoogle	1,089.3 us	2260.17 KB

Разгрузка потребителя vol2



Разгрузка потребителя vol2



Разгрузка потребителя VOL2

Преимущества

- Снижение накладных расходов при отправке в удалённый приёмник

Недостатки

- Замедление потоков поставщиков
- Требуются дополнительные усилия по разработке

Тестирование работы приёмника

Простейший макет удалённого приёмника

```
public class RemoteLoggingSink : IBatchedLogEventSink
{
    public Task EmitBatchAsync(IReadOnlyCollection<LogEvent> batch)
        => Task.Delay(50);
}
```

Регистрация:

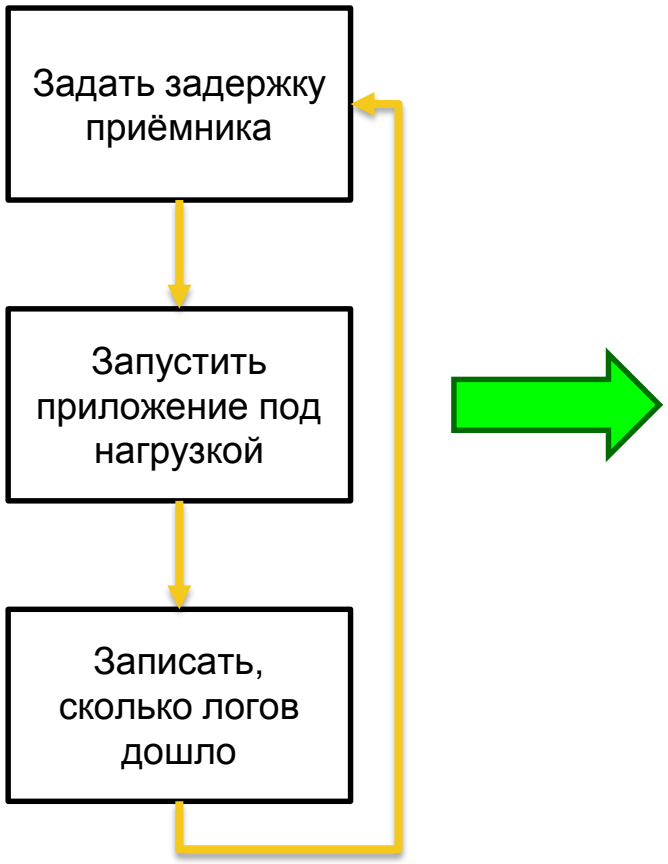
```
services.AddLogging(x => x
    .AddSerilog(
        new LoggerConfiguration()
        .WriteTo.Sink(
            new RemoteLoggingSink(),
            new BatchingOptions())
        .CreateLogger()));
```

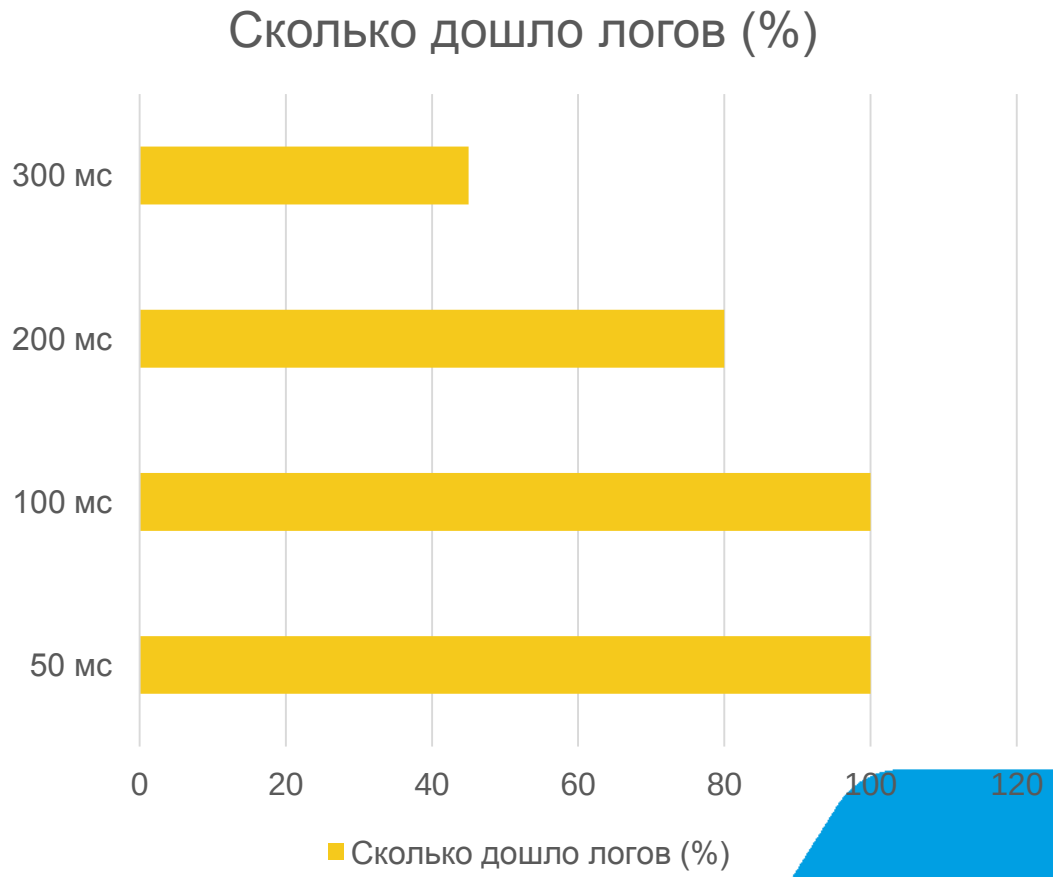
Оценка потерь логов

- Изменяемая величина – задержка приёмника
- Постоянная величина – максимальный размер буфера (`QueueLimit`)
- Измеряемая величина – количество логов, зафиксированных приёмником

```
public class RemoteLoggingSink : IBatchedLogEventSink
{
    private int LogCount = 0;

    public Task EmitBatchAsync(IReadOnlyCollection<LogEvent> batch)
    {
        LogCount += batch.Count;
        return Task.Delay(200);
    }
}
```

Оценка потерь логов - результат





Выводы по удалённым приёмникам

- Основная проблема – переполнение буфера
- Возможные решения:
    - Ограничение размера буфера - надежно, но это компромисс
    - Переход к локальному приёмнику с низкими задержками – эффективно, но требует затрат
    - Оптимизация и разгрузка потребителя – эффективно, но требует затрат

Выводы

Основные проблемы подсистемы логирования с использованием удалённых приёмников:

- Потеря логов при ограничении размера буфера
- Заполнение памяти при отсутствии ограничения буфера

Выводы

Пути решения проблем:

Кастомизация приёмников с целью снижения задержек

- Усложнение архитектуры
- Повышенные трудозатраты
- Нет 100% защиты от отказа

Выводы

Пути решения проблем:

Ограничение размера буфера

- Минимальные трудозатраты
- Чревато потерей логов
- Высокая надежность решения

Выводы

Чем же хорош Serilog?

- Простота использования
- Обилие опций форматирования и вывода
- Широкие возможности кастомизации

Спасибо за внимание!

Владимир Куропатка

SCAN ME

vladimir.kuropatka@altenar.com

Владимир Куропатка