Сказки о преждевременной оптимизации

Дмитрий Иванов, JetBrains

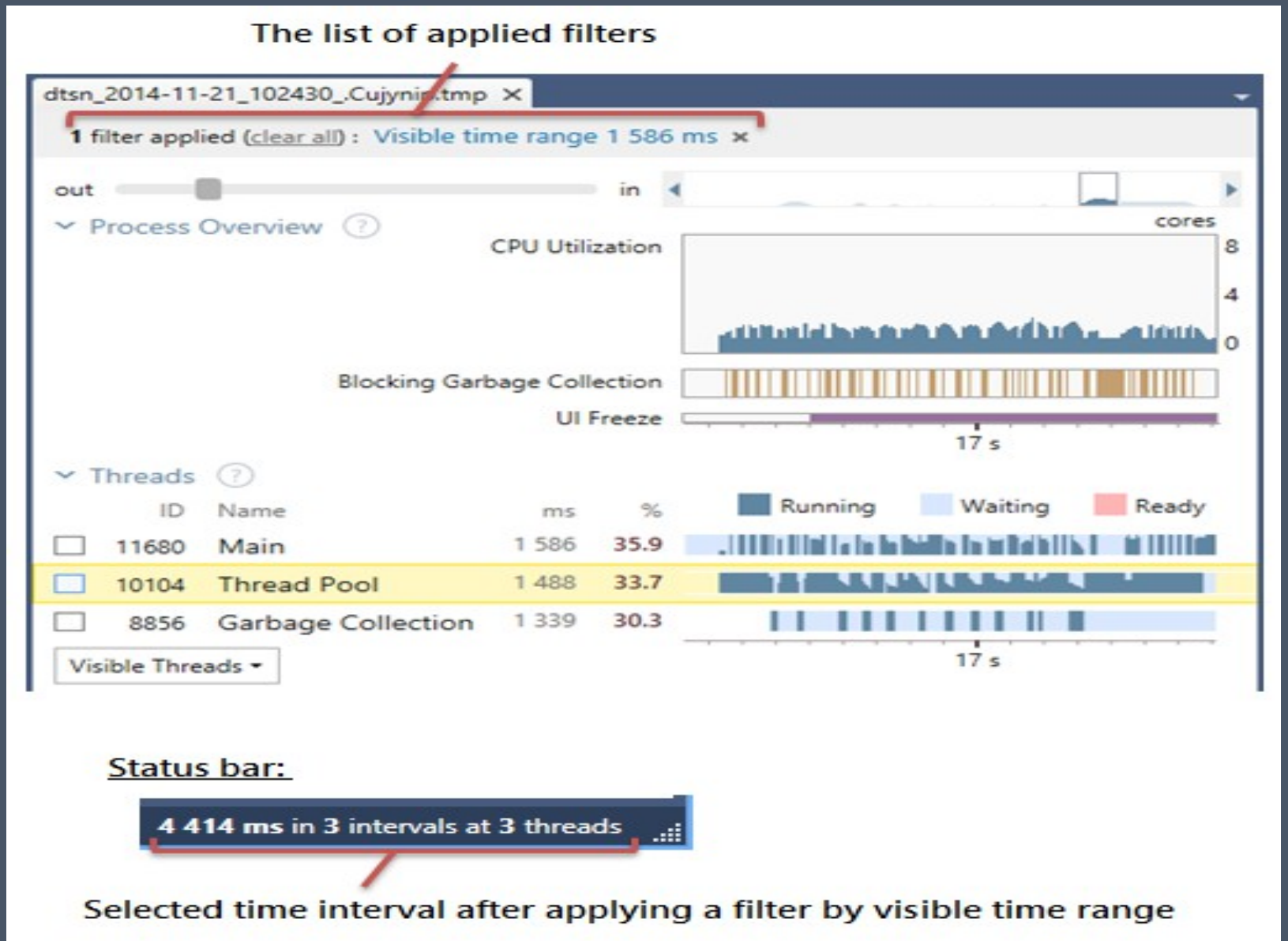
Мантра

Контрмантра

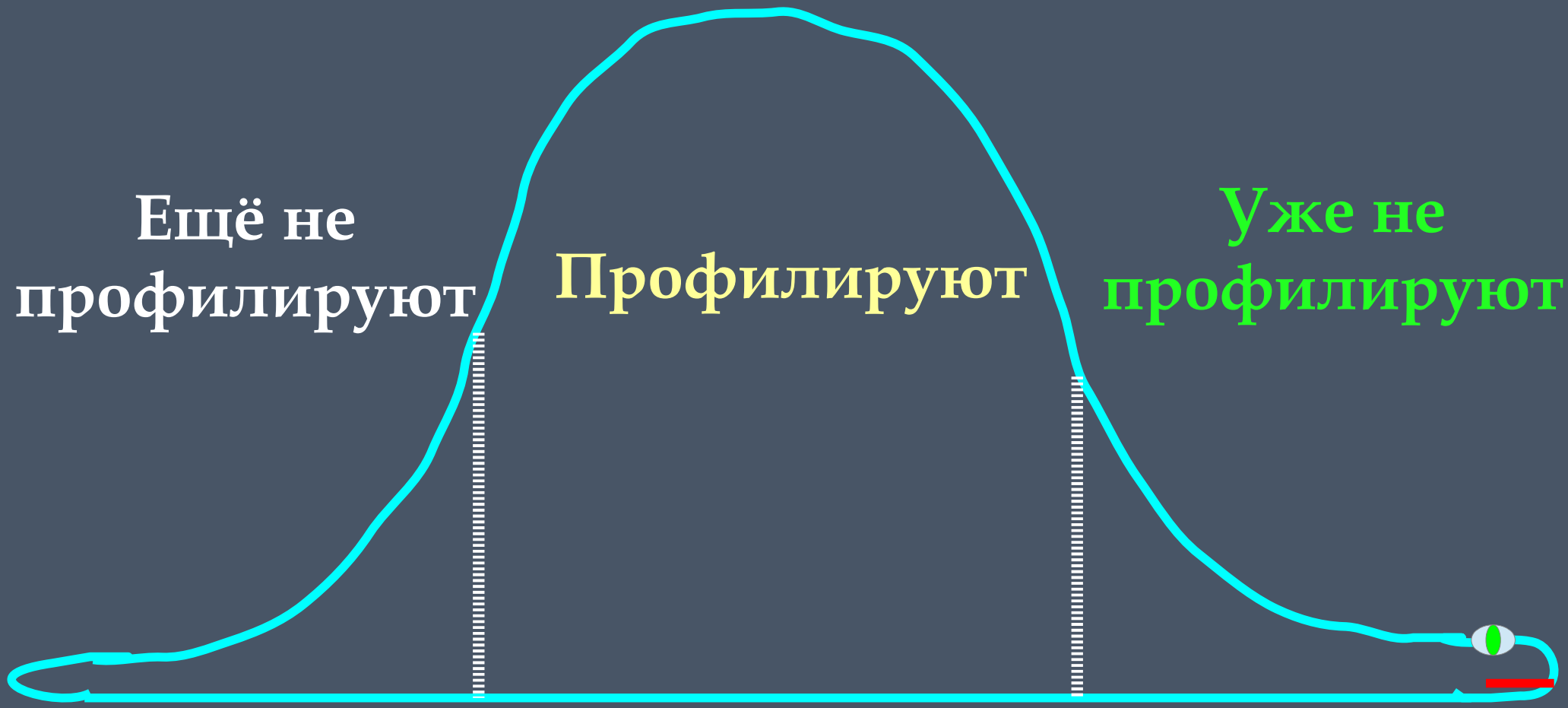
*“Преждевременная **пессимизация** — sqrt(всех зол)”*

Порочный круг

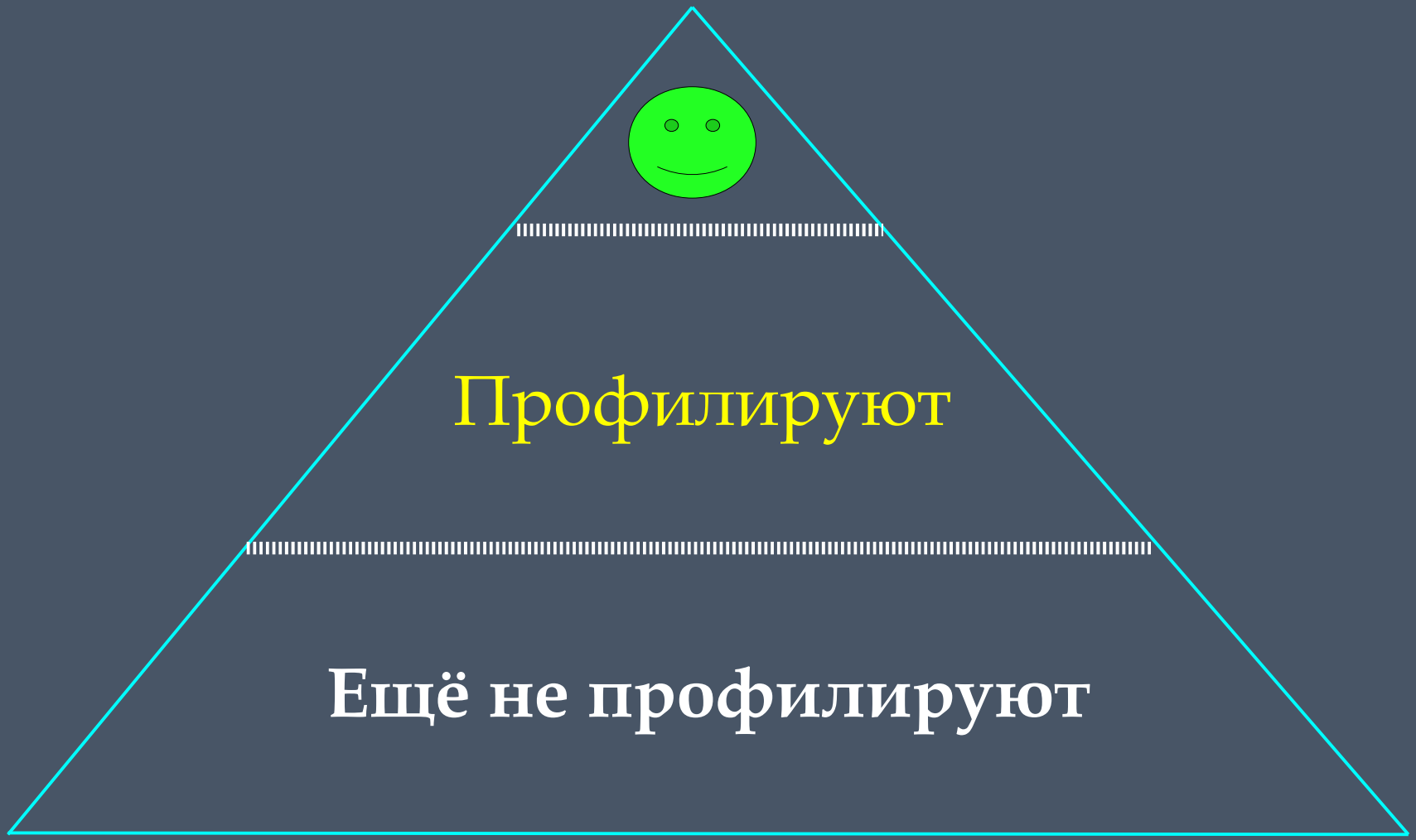
Нет слабого звена



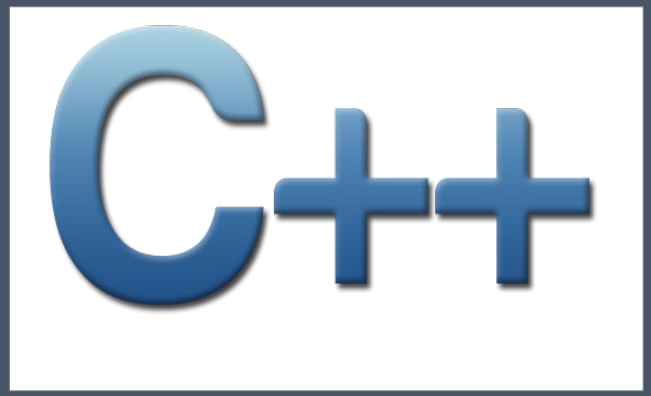
Гипотеза



Суровая реальность



Хочешь быстро – используй C++



Такие разные CPP

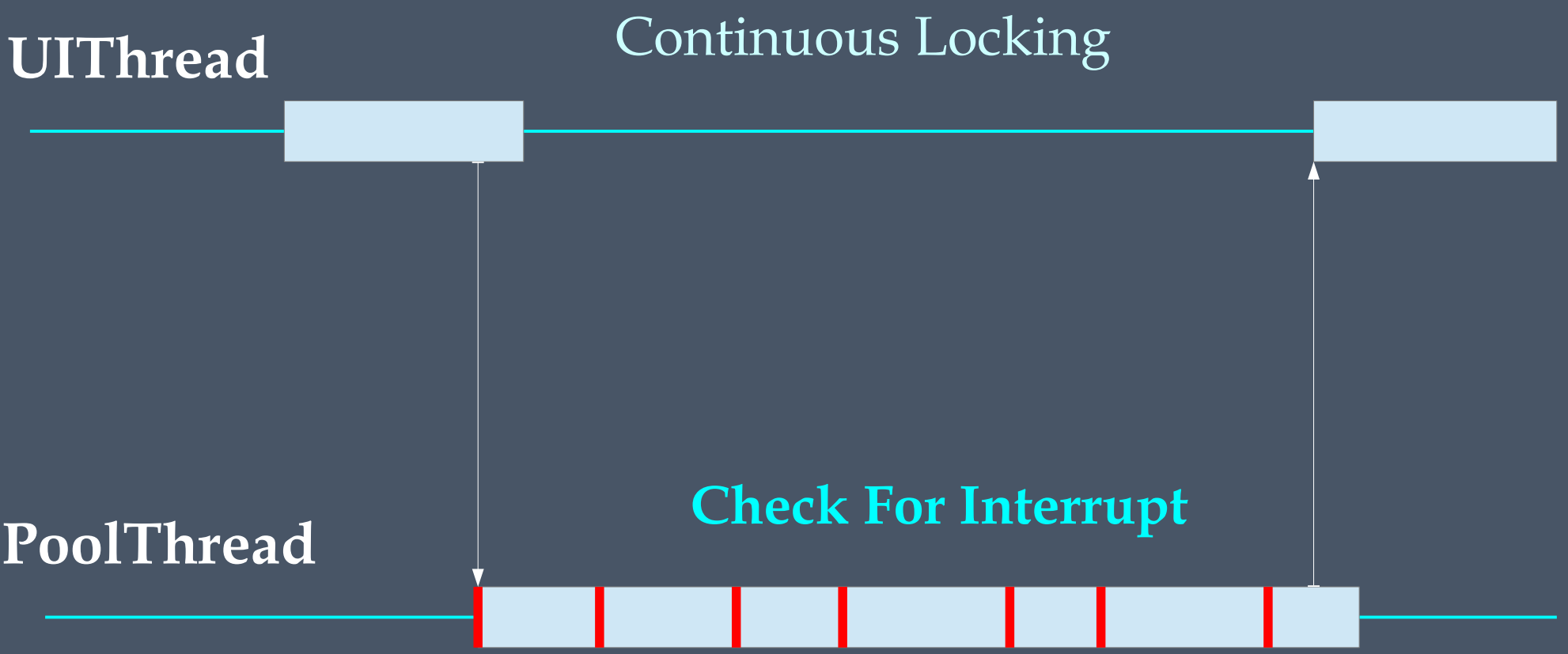
300 MB RAM
35% blocking GC

8 GB RAM
5% GC

Что? Где? Когда? Зачем?

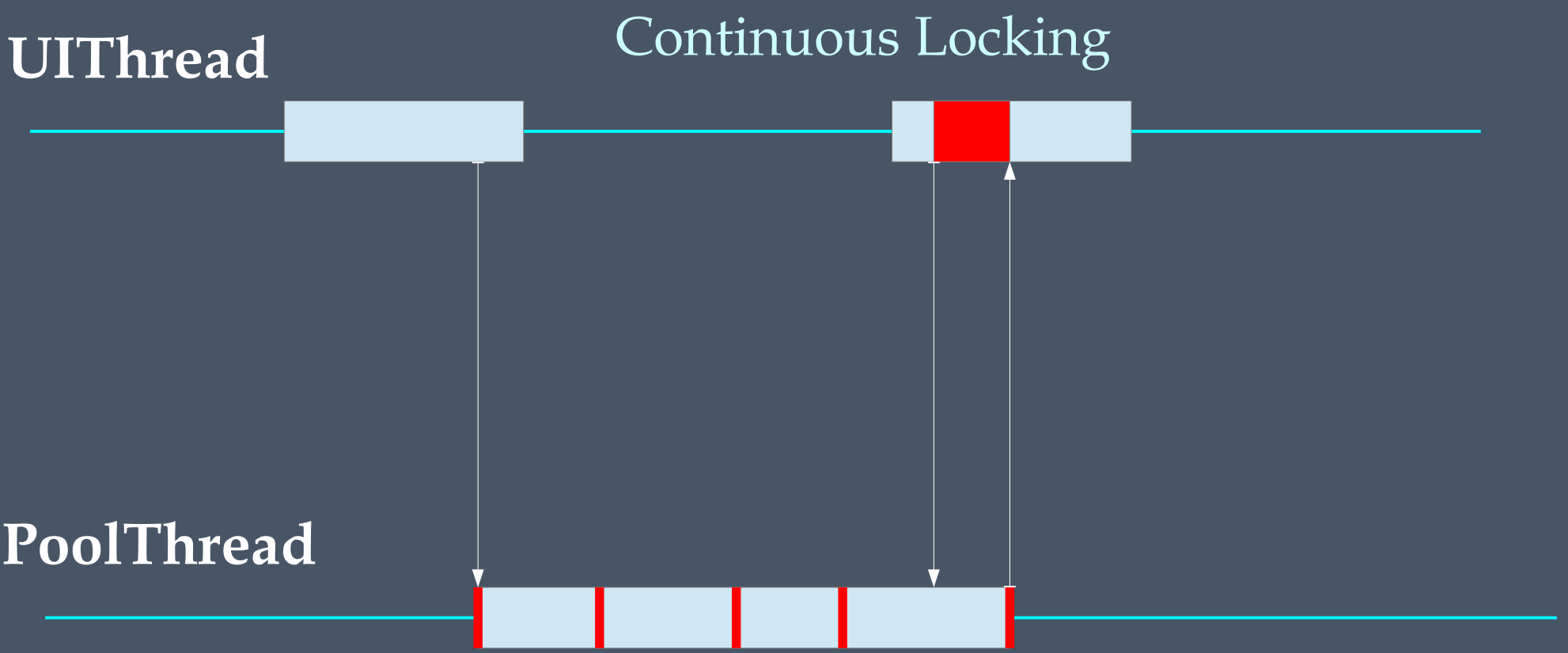
- Комплексный процесс =)
- UI-затыки
- Загрузка
- Бесконечный progress
- Оптимизация электроэнергии. Ха-ха.
- **Экономим время на профилирование и полное переделывание**

UI модель решарпера



Under **ReadLock**

Прерывание

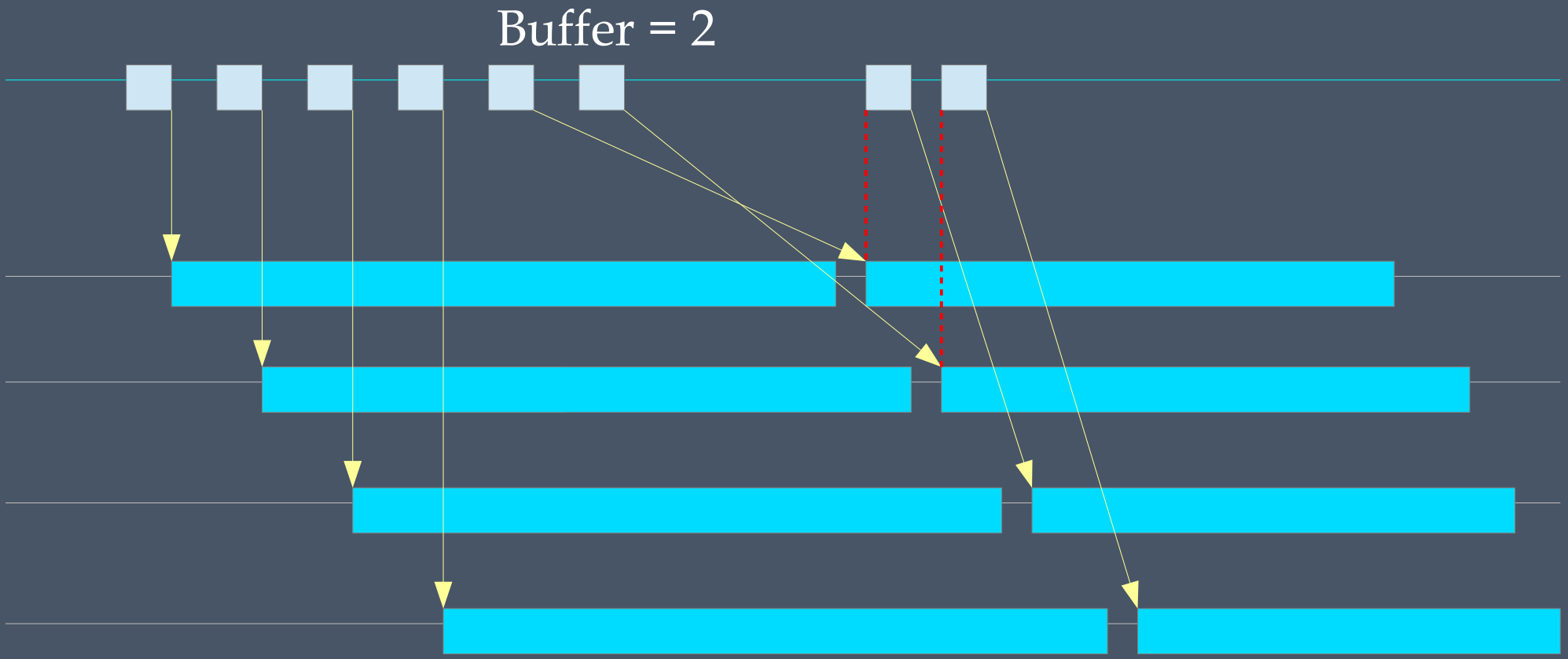

UIThread
Continuous Locking
PoolThread
Somebody asked for WriteLock?

Загрузка/выгрузка

- Загрузка классов
  - **JIT** не всемогущ
  - **Ngen**, **mpgo**
- Загрузка данных
  - Сериализуем всё
  - Последовательное чтение базы данных
- Закрытие приложение
  - Пользователь не готов ждать на закрытии

Работа с диском

- Препроцессируем в один поток



GroupingEvent

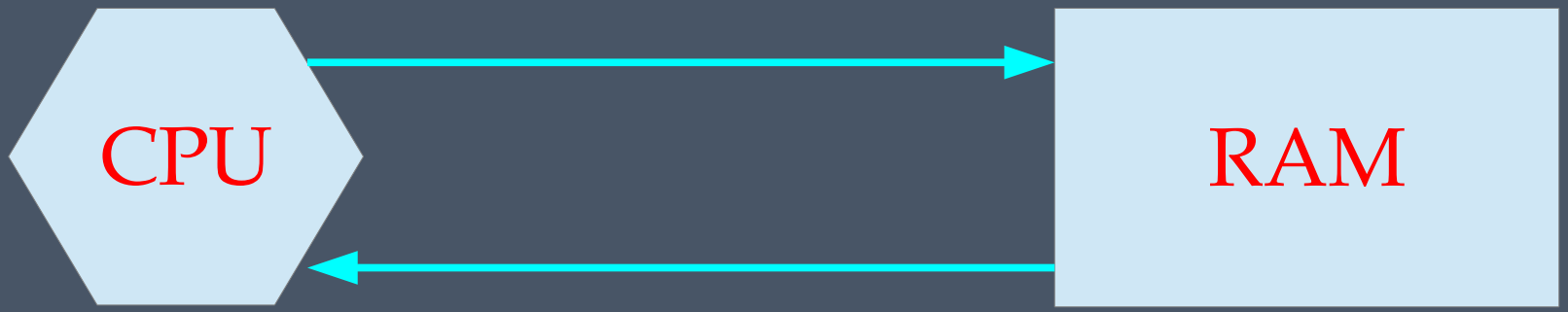
prolongate

no prolongate

Общая форма: Handle(T) **vs** Handle(Enumerable(T))

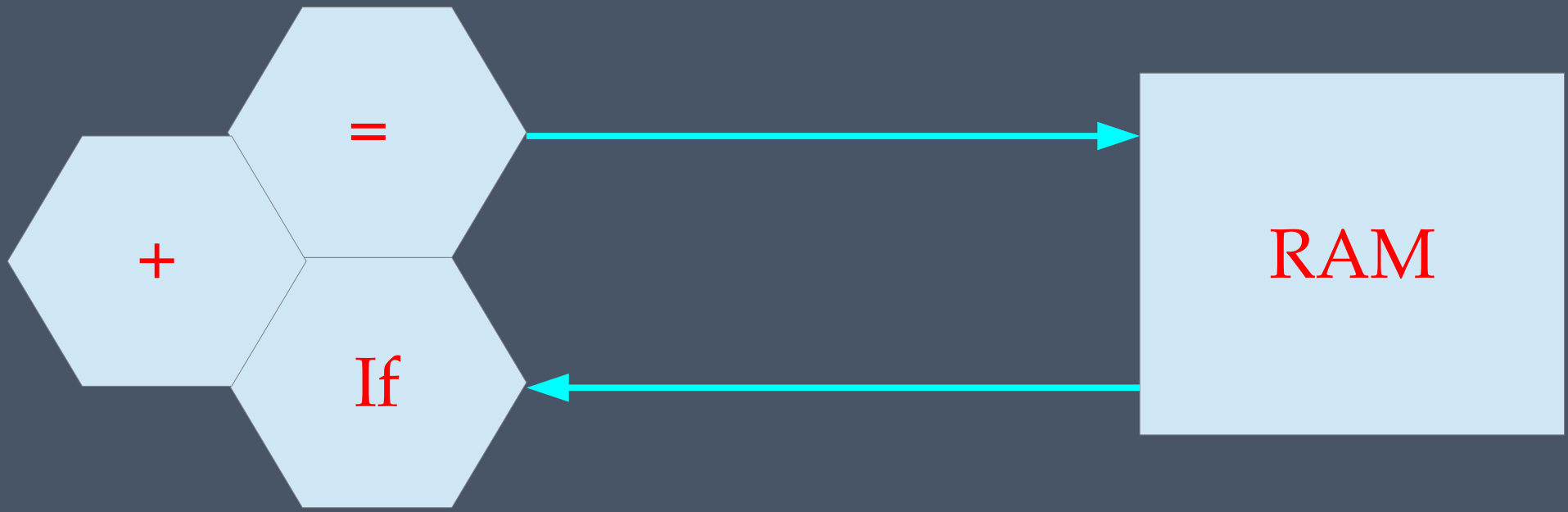
Алгоритмическая оптимизация

- Кнут и Корман
  - Иногда вам нужно писать бинарный поиск и сортировку
  - O(**log n**), O(**n**), O(**n^2**)



Алгоритмическая оптимизация

- Улучшаем модель
- Сортировка живой коллекции
- Проверяем в tracing-режиме **dotTrace**



Бумажки и навыки

ДИПЛОМ

ЛВ № 317429

Настоящий диплом выдан .....

в том, что он ..... в 19 ..... году поступил .....
в .....

и в 19 ..... году окончил ..... полный курс .....

по специальности .....

Решением Государственной экзаменационной комиссии от „ ..... “ ..... 19 ..... г.

присвоена квалификация .....

Председатель Государственной экзаменационной комиссии

Ректор

Секретарь

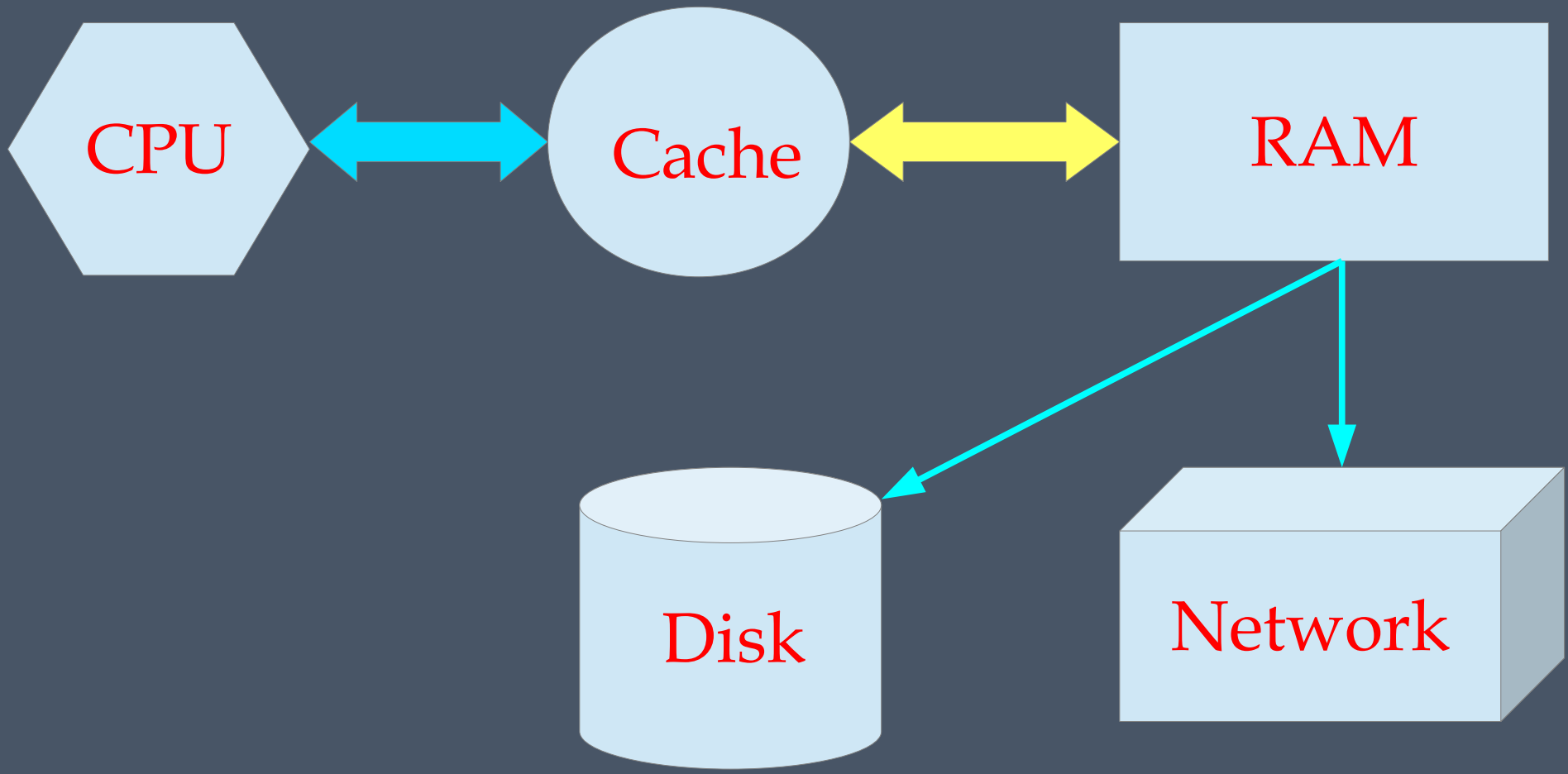
М. П.

Город ..... „ ..... “ ..... 19 ..... г.

Регистрационный № .....

Московская типография Гознака.1980.

Взрослая оптимизация



Простые правила

- Пространственная локальность
  - Последовательная работа с массивом
- Branch prediction и инлайнинг
- Обходимся без MemoryBarrier-ов
- Неидиоматичный код
  - Struct arrays
  - Ручное управление памятью

Многозадачность

- Task<T> - удобное, но медленное API
- Естественный параллелизм
- AutoResetEvent, ReadWriteLock, DateTime.Now – медленно, тестируйте.
- Алгоритмы на Interlocked
- Постарайтесь оптимизировать всё в одном потоке

Какая преждевременная оптимизация нужна

- Struct everywhere (ex. List vs IList)
- Оптимизируем ваш **Util**
  - Никакого LINQ, замыканий, params, yield, боксинга, weak-ref, finalization,
  - Тулы: **dotMemory**, **dotPeek**, **dotTrace** **Timeline**, **ReSharper**
- Initial capacity, гибридные структуры
- Если не пробуем – не учимся.

Замыкаем код на Util

- Будьте нетерпимы к “самопалу” в фичёвом коде!

```
public class Foo
{
  private readonly IDictionary<string, BigObject> myMap = new Dictionary<string, BigObject>(100500);

  public BigObject GetOrCreateObject(string key)
  {
    BigObject res;
    if (myMap.TryGetValue(key, out res)) return res;

    res = CalculateObject(key);
    myMap[key] = res;
    return res;
  }

  private BigObject CalculateObject(string key)
  {
    //Do sime stuff
    return new BigObject();
  }
```

HashCode

```
public static int GetPlatformIndependentHashCode([CanBeNull] this string s)
{
  if (s == null) return 0;

  int hash = 19;
  for (int i = 0; i < s.Length; i++)
  {
    {
      hash = hash*31 + s[i];
    }
  }
  return hash;
}
```

“Уже украденное до нас”

```
public static unsafe int MurmurHash3(byte* pData, int nLen, uint seed)
{
  unchecked
  {
    Byte* data = pData;
    int nblocks = nLen / 4;

    UInt32 h1 = seed;

    UInt32 c1 = 0xcc9e2d51;
    UInt32 c2 = 0x1b873593;

    // DWORD blocks
    UInt32* blocks = (UInt32*)(data + nblocks * 4);

    for(int i = -nblocks; i != 0; i++)
    {
      UInt32 k1_1 = blocks[i];

      k1_1 *= c1;
      k1_1 = (k1_1 << 15) | (k1_1 >> 17); // rotl 15
      k1_1 *= c2;

      h1 ^= k1_1;
      h1 = (h1 << 13) | (h1 >> 19); // rotl 13
      h1 = h1 * 5 + 0xe6546b64;
    }
```

Сила C++: stackalloc

- В C# очень печальный **stackalloc**
- Изобретём свой **стэкаллок**
  - **TLS**-объект **UnsafeWriter**
  - 1 MB стека + возможность расширения
  - Свой **TaskScheduler**
  - Ну и свои маршаллеры

Где использовать?

- Когда в руках молоток – всё гвоздём кажется
- ReSharper **GoToEverything**
  - Выбираем все символы
  - Таски по 5000 items
  - Создаём объект IdentifierMatcher один на таску. Прекалькуляция по шаблону.
  - Внутри работаем с таблицей (конечным автоматом) на unsafe памяти

Резюмируя