




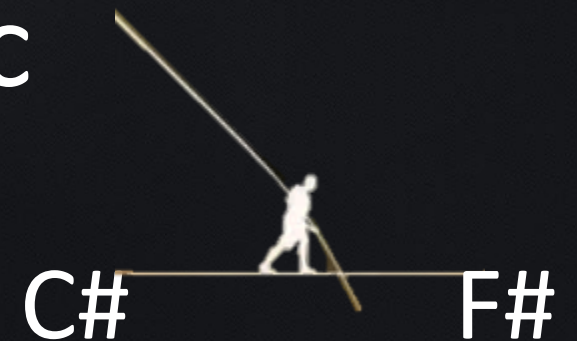
Software Engineering Conference Russia 2018

October 12-13
Moscow

(ООП, ФП) в мейнстримовом
программировании -> баланс

Виталий Камянский

 FirstLine Software

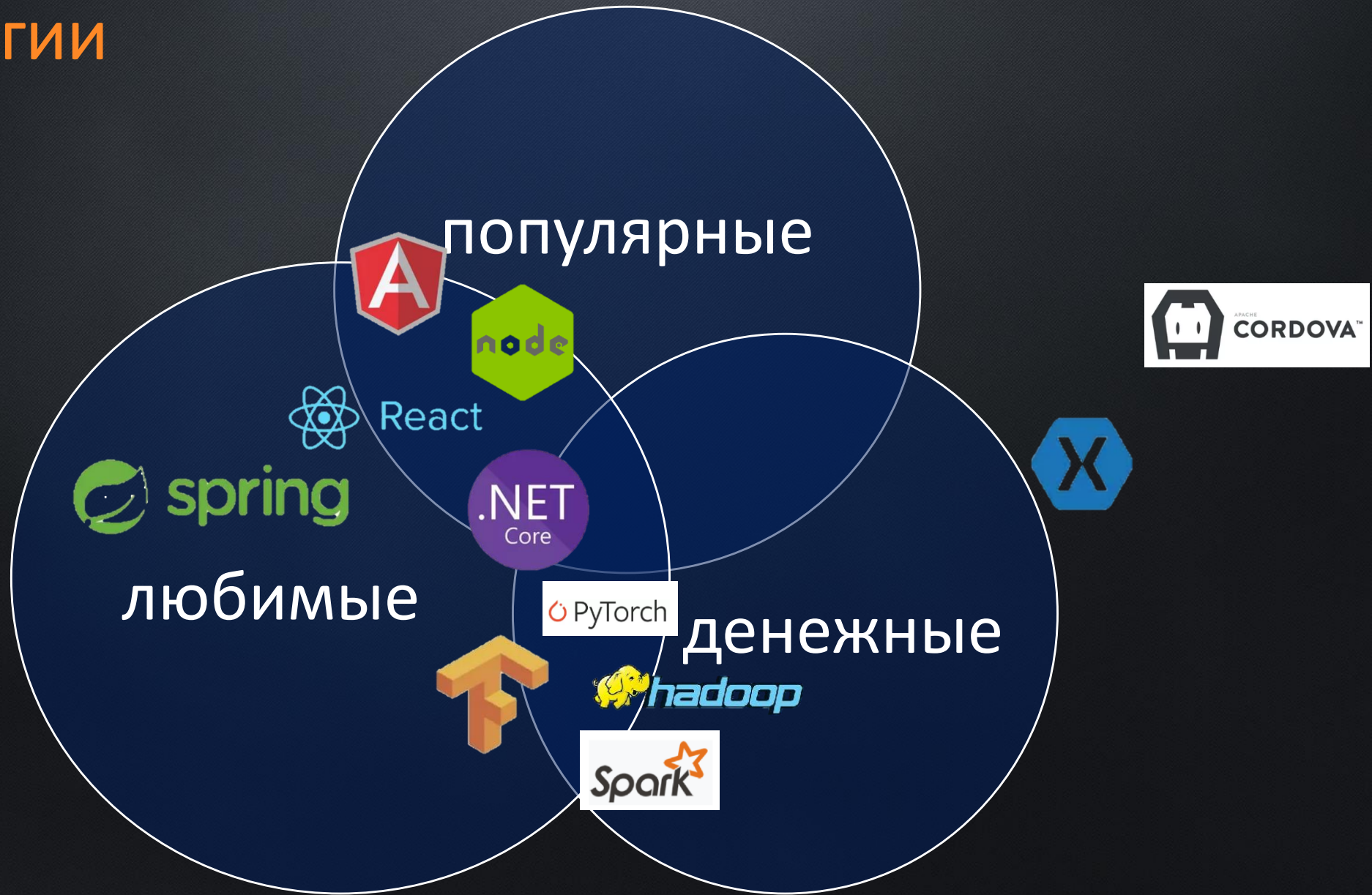


ЯЗЫКИ

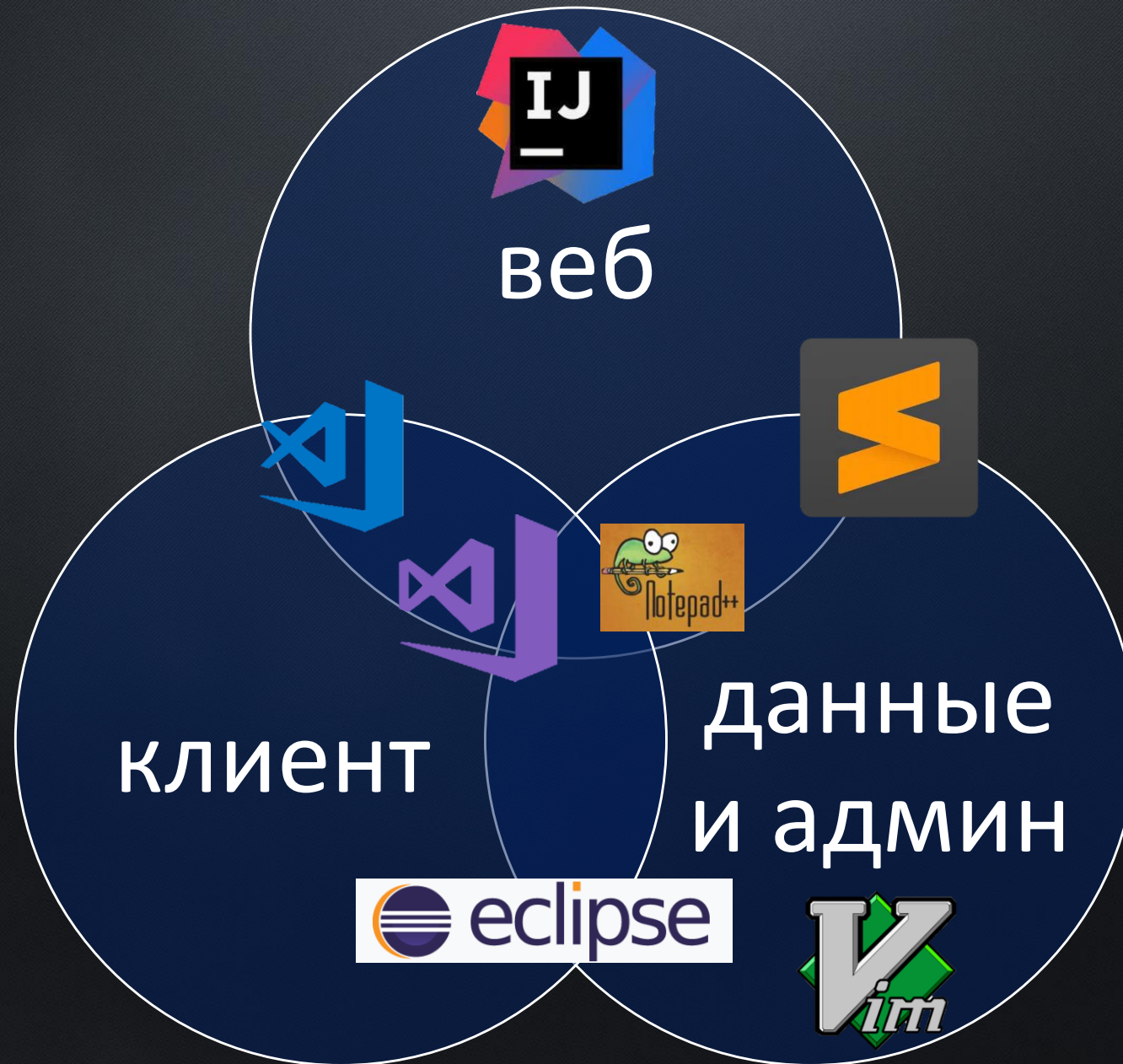
* по данным исследования Stack Overflow 2018



Технологии



Среды



ЯЗЫКИ

* по данным исследования Stack Overflow 2018



ФП приходит в C#





2.0

Ноябрь 2007

Март 2017

Август 2017

 A Extension(this A source){...}

 new { Name = "Anonymous" }

λ x => x + 1

<int> Enumerable.Range(6, 10)

(string, int) ("tuple", 1)

```
case switch(a)
{
    case TypeA at when at.Name=="a":
        return "it is a";
    ...
}
```

 var (name, num) = tuple;

Подходы



Сущности



Пример: список

C#

```
var list = new List<int>();  
list.AddRange(new[] { 1, 2, 3 });  
  
// Сообщение  
list.Reverse();  
  
// Вычисление  
var newList = list.Reverse<int>().ToList();
```

F#

```
let listEx () =  
    let list = []  
  
    // Вычисление  
    list |> List.append [1; 2; 3]  
    ..> List.rev
```

Пример: быстрые вычисления

Императивный подход

```
·var·tailsCount·=·0;
·foreach·(var·x·in·random1to200List)
·{
····if·(x·>·100)·tailsCount++;
·}
```

Декларативный подход

```
·tailsCount·=·random1to200List.Count(y·=>·y·>·100);
```

Пример: быстрые вычисления + волшебное слово

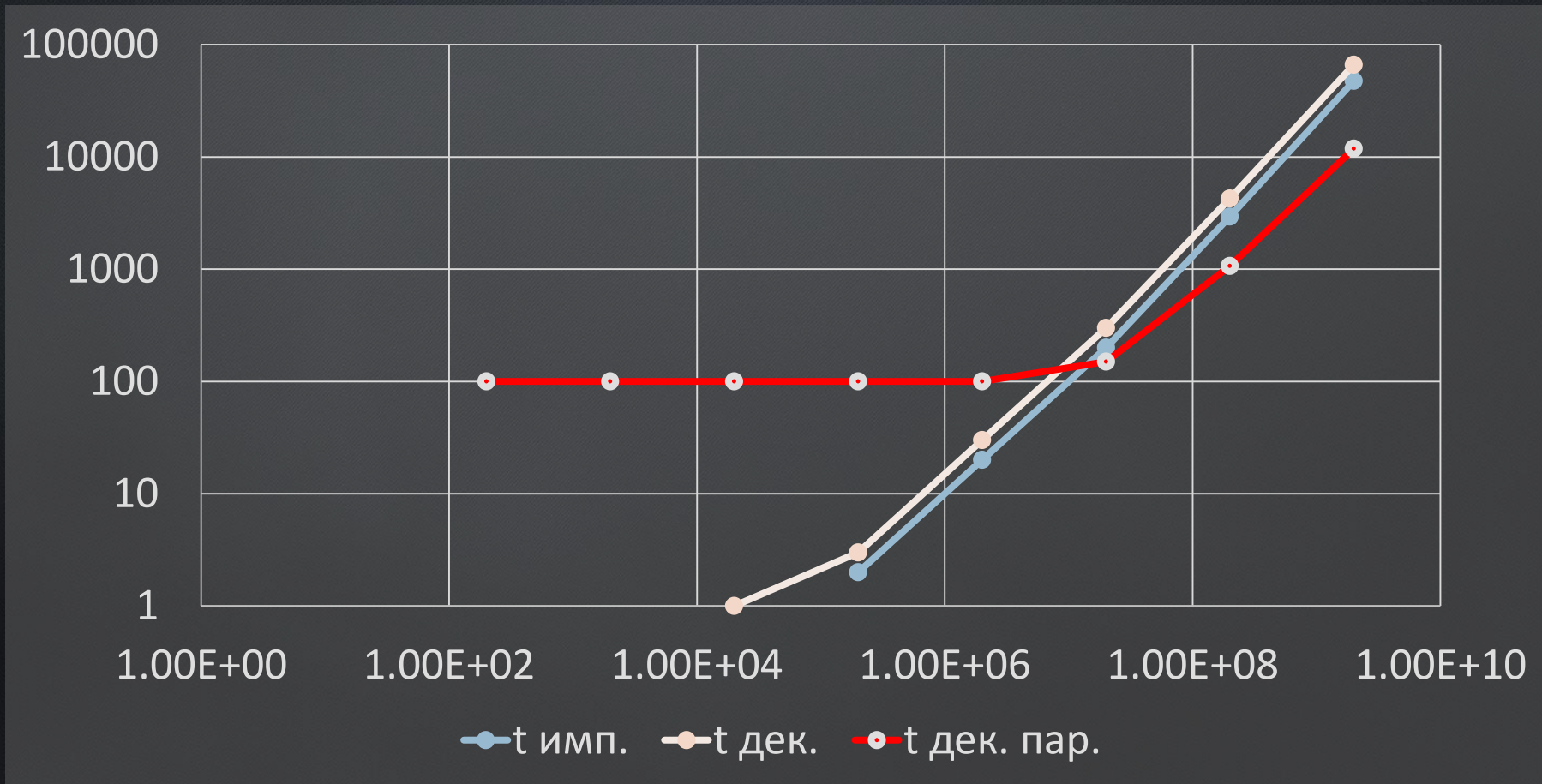
Императивный подход

```
var tailsCount = 0;  
foreach (var x in random1to200List)  
{  
    if (x > 100) tailsCount++;  
}
```

Декларативный подход

```
tailsCount = random1to200List.AsParallel().Count(y => y > 100);
```

Быстрые вычисления: скорость



Быстрые вычисления (часть 2)

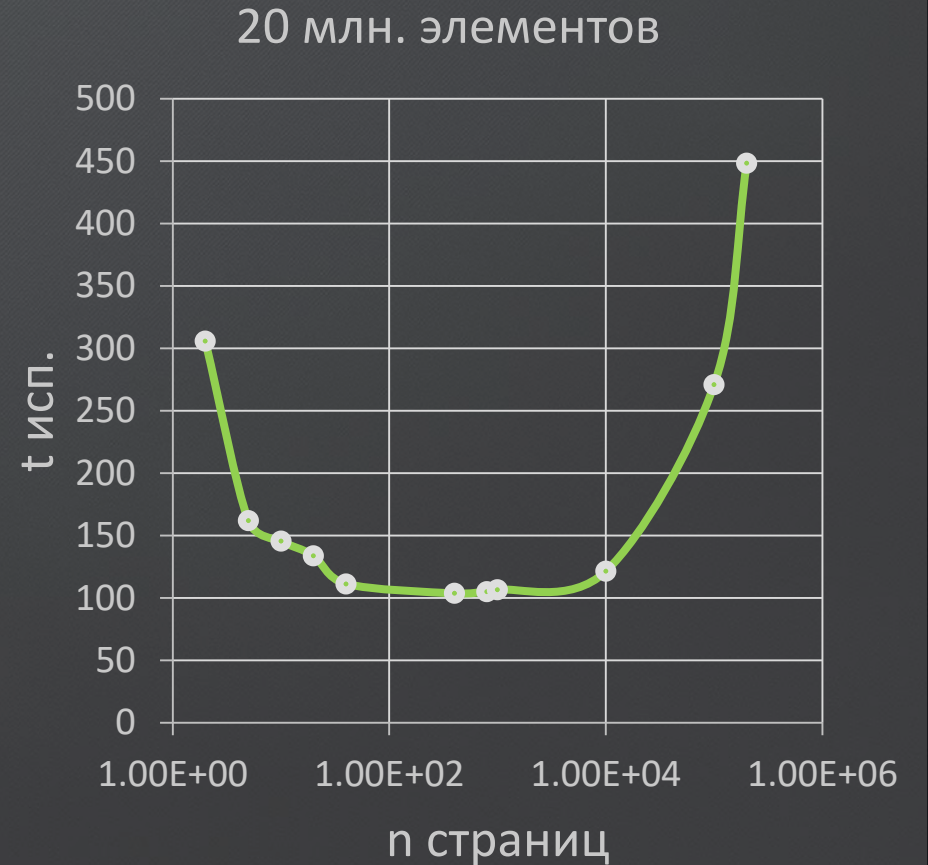
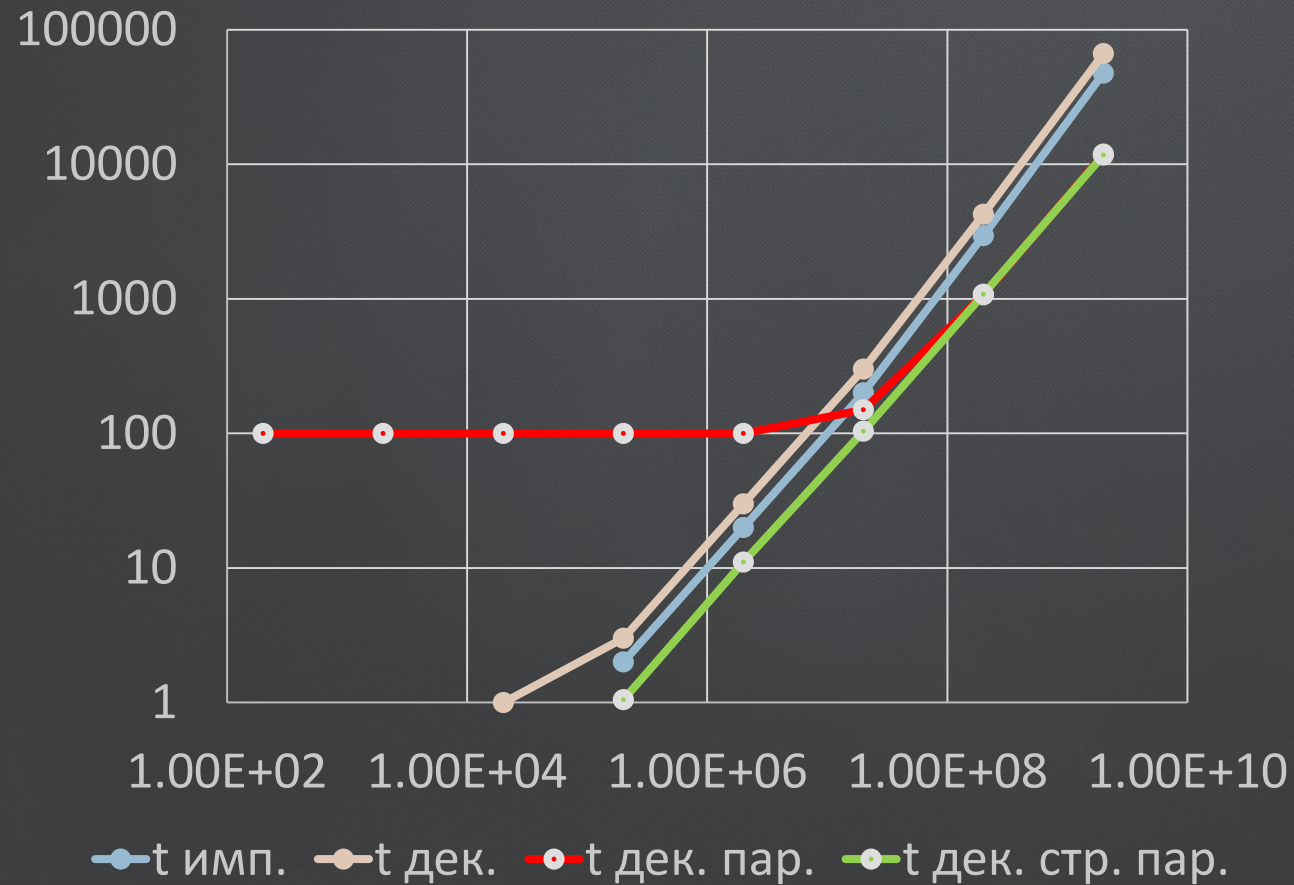
Менее тривиальный способ параллелизации со страницами и разделами

```
int ParallelCount<T>(List<T> list, Func<T, bool> predicate, int numPages)
{
    var totalItems = list.Count;

    return Enumerable.Range(0, numPages)
        .Select(x => list.GetRange(x * (totalItems / numPages), totalItems / numPages))
        .AsPartitioned(numPages).AsParallel()
        .Sum(x => x.Sum(y => y.Count(predicate)));
}
```

```
ParallelCount(random1to200List, x => x > 100, numPages);
```

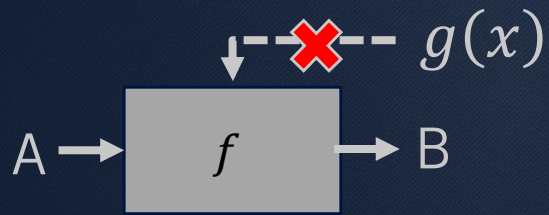
Быстрые вычисления: скорость (часть 2)



Решения

	императивный подход	декларативный подход
объекты	-> ООП пример: Windows Forms	-> композиция с DSL пример: XAML / Autofac
данные	-> Процедурное программирование Циклы, ветвление, вызов подпрограмм, переменные общего доступа	-> вычисления с DSL пример: SQL / LINQ

Чистые функции → внутренние DSL



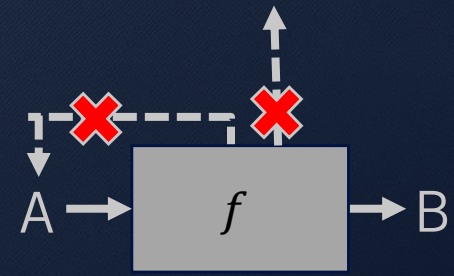
ссылочная
прозрачность

D



честность

I



отсутствие
побочных
эффектов

S

Внутренние DSL: что нужно ещё?

- Функции как данные λ

```
Func<int, string> addIntText = x => addTypeText(x);  
var funArray = new[] { addIntText };
```

- Возможность строить из функций pipeline  

```
var numbers = new[] { 1, 2, 3, -1, -2 };  
var averagePositive = numbers.Where(x => x > 0).Average();
```

Пример: DSL для сжатия

Сборка алгоритма декларативная

Алгоритм работы со Stream императивный

```
DataFuncs.Copy()  
.....FromFile("C:\\projects\\try.txt")  
.....ToZipPart("try.txt", DateTime.Now)
```

```
⏏ (extension) Action<ZipOutputStream> Action<Stream>.ToZipPart(string fileName, DateTime creationDateTime)
```

```
.....ToZip(3)  
.....ToFile("C:\\projects\\try.zip");
```

(демонстрация)

Пример: Autofac и функции

Иньекция фабричной функции в свойство

```
internal class MainWindowViewModel : IMainWindowViewModel
{
    public IAppSettingsService AppSettingsService { get; set; }
    public Func<ILoginViewModel> GetLoginViewModel { get; set; }
}
```

Настройка контейнера

```
var viewModelAssembly = typeof(MainWindowViewModel).Assembly;

builder.RegisterAssemblyTypes(viewModelAssembly)
    .Where(x => x.Name.EndsWith("ViewModel", StringComparison.InvariantCultureIgnoreCase))
    .AsImplementedInterfaces()
    .PropertiesAutowired()
    .SingleInstance();
```

В F# СВОИ ПЛЮСЫ

размеченные объединения

```
type CardSystem =  
    | MC  
    | Visa  
    | Mir  
  
type CardNumber =  
    | Number of int  
  
type PaymentMethod =  
    | CashOnDelivery  
    | Card of CardSystem * CardNumber
```

[<Measure>] единицы измерения

```
[<Measure>  
type RUR;  
[<Measure>  
type USD;  
  
type Payment =  
    | Dollars of float<USD>  
    | Rubles of float<RUR>
```

match сравнение с образцом

```
let checkPayment payment =  
    match payment with  
    | Some (Card (Mir, num), Rubles r) -> true  
    | None -> true  
    | _ -> false
```

Выводы

- Экземпляры типа (**данные**) или класса (**объекты**)? – зависит от контекста операции
- ФП – надёжные вычисления с **чистыми** функциями + функции как **данные**
- Внутренние DSL – **композиция** сущностей + **декларативность** + удобство
- Элементы ФП и DSL в ООП-языке делают **ООП** комфортнее и сами полезны
- 2 языка: многополярность – один из подходов

Ссылки

- Исследование Stack Overflow 2018
<https://insights.stackoverflow.com/survey/2018>
- Tomas Petricek, Jon Skeet “Real-World Functional Programming”
<http://functional-programming.net/rwfp/>
- Scott Wlaschin “Domain Modeling Made Functional”
<https://pragprog.com/book/swdddf/domain-modeling-made-functional>
- Bartosz Milewski “Category Theory for Programmers”
<https://github.com/hmemcpy/milewski-ctfp-pdf>

Вопросы

- Виталий Камянский
- Почта: vitaly.kamiansky@firstlinesoftware.com
- Телефон: +7 (921) 322 05 96
- <https://github.com/vkamiansky>

