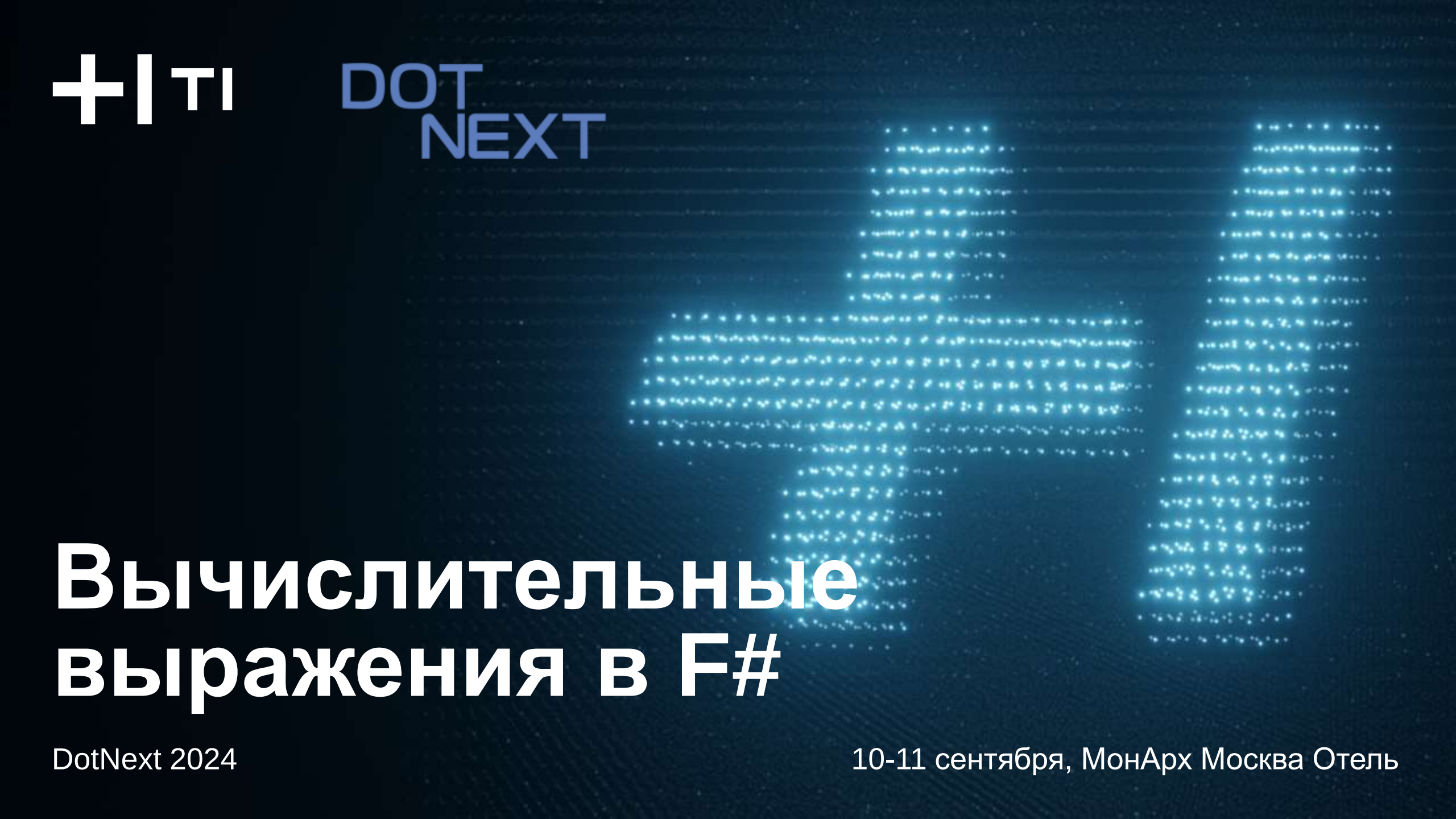


Вычислительные выражения в F#

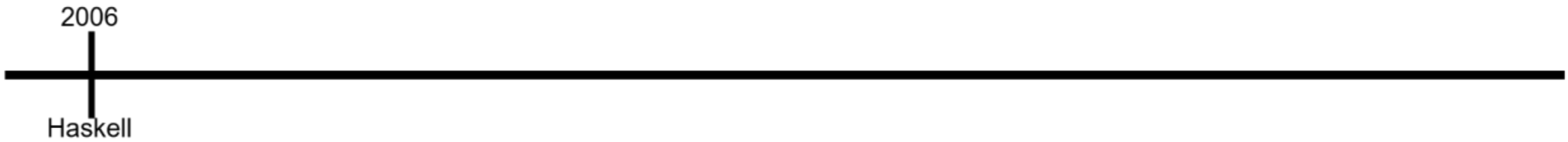
DotNext 2024

10-11 сентября, МонАрх Москва Отель

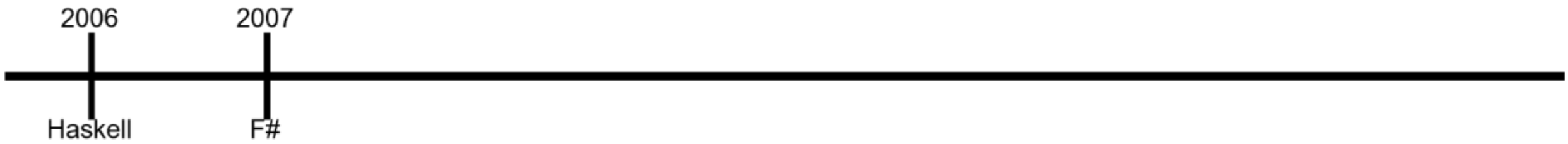
Марк Шевченко

- Программирует более тридцати лет.
- Писал на C, C++, Delphi, Perl, Java, PHP, C#, F#, Ruby, Scheme и Rust.
- Организовывает встречи в Московском клубе программистов.
- Ведущий разработчик в ИТ-холдинге Т1.

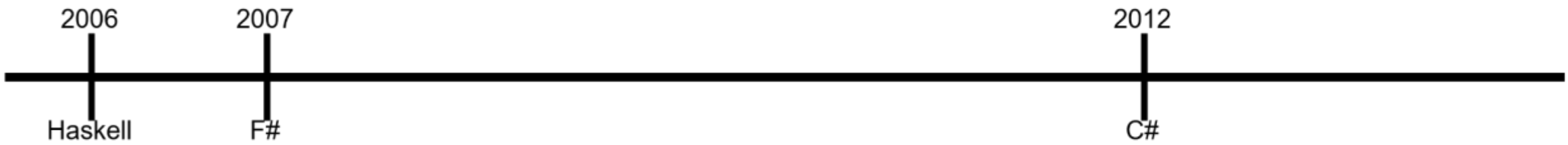
async/await



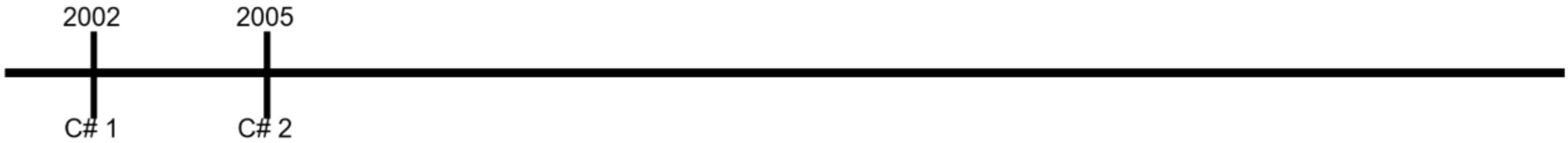
async/await



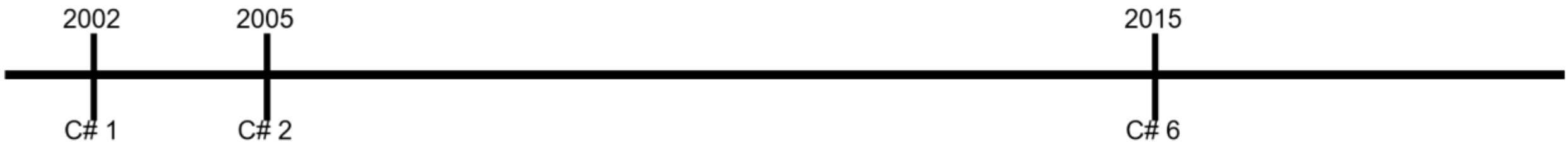
async/await



null propagation



null propagation



Computation Expressions

Логирование

```
let log value = printfn "Log: %A" value
```

Логирование

```
let width = 100
log width                    // => Log: 100
let height = 200
log height                   // => Log: 200
let area = width * height
log area                     // => Log: 20000
```

Логирование

```
type LoggerBuilder() =
    member _.Bind(x, f) =
        log x
        f x

    member _.Return(x) =
        x

let logger = new LoggerBuilder()
```

Логирование

```
logger {
    let! width = 100              // => Log: 100
    let! height = 200             // => Log: 200
    let! area = width * height // => Log: 20000

    return area
}
```

Безопасная арифметика

```
let safe_sqrt x =
    if x < 0.0
    then None
    else Some(sqrt x)

let safe_div numerator denominator =
    if denominator = 0.0
    then None
    else Some(numerator / denominator)
```

Безопасная арифметика

```
let solve_square_equation a b c =
    let d = b * b - 4.0 * a * c
    let sqrt_d = safe_sqrt d
    match sqrt_d with
    | None -> None
    | Some sqrt_d ->
        let x1 = safe_div (-b + sqrt_d) (2.0 * a)
        match x1 with
        | None -> None
        | Some x1 ->
            let x2 = (-b - sqrt_d) / (2.0 * a)
            Some(x1, x2)
```

Безопасная арифметика

```
printfn "%A" (solve_square_equation 0.0 2.0 1.0) // => None
printfn "%A" (solve_square_equation 1.0 -3.0 2.0) // => Some (2.0, 1.0)
printfn "%A" (solve_square_equation 1.0 2.0 4.0) // => None
```

Безопасная арифметика

```
type OptionBuilder() =
    member _.Bind(x, f) =
        match x with
        | None -> None
        | Some x' -> f x'

    member _.Return(x) =
        Some x

let option = new OptionBuilder()
```

Безопасная арифметика

```
let solve_square_equation a b c =
    option {
        let d = b * b - 4.0 * a * c
        let! sqrt_d = safe_sqrt d
        let! x1 = safe_div (-b + sqrt_d) (2.0 * a)
        let x2 = (-b - sqrt_d) / (2.0 * a)

        return (x1, x2)
    }
```

Слово на букву М

Вычислительные выражения — это…

- Чистый код

Вычислительные выражения — это…

- Чистый код
- Построитель (Builder)

Вычислительные выражения — это…

- **Чистый код**
- **Построитель** (Builder)
  - Bind
  - Return

Вычислительные выражения — это…

- **Чистый код**
- **Построитель** (Builder)
  - Bind
  - Return
- **Тип-обёртка** (`Option<T>`)

Функции-продолжения (continuations)

```
Int32.Parse "abc"
// System.FormatException:
// The input string 'abc' was not in a correct format.

Int32.Parse "123"
// 123
```

Функции-продолжения (continuations)

```
Int32.TryParse "abc"
// (false, 0)

Int32.TryParse "123"
// (true, 123)
```

Функции-продолжения (continuations)

```
let parseInt ifSuccess ifError (s: string) =
    match Int32.TryParse s with
    | (true, value) -> ifSuccess value
    | (false, _) -> ifError ()
```

Функции-продолжения (continuations)

```
let parseInt ifSuccess ifError (s: string) =
    match Int32.TryParse s with
    | (true, value) -> ifSuccess value
    | (false, _) -> ifError ()
```

Функции-продолжения (continuations)

```
parseInt (printfn "%d") (fun () -> printfn "Invalid format") "abc"
// => Invalid format


parseInt (printfn "%d") (fun () -> printfn "Invalid format") "123"
// => 123
```

Функции-продолжения (continuations)

```
parseInt (fun i -> Some i) (fun _ -> None) "abc"
// None

parseInt (fun i -> Some i) (fun _ -> None) "123"
// Some 123
```

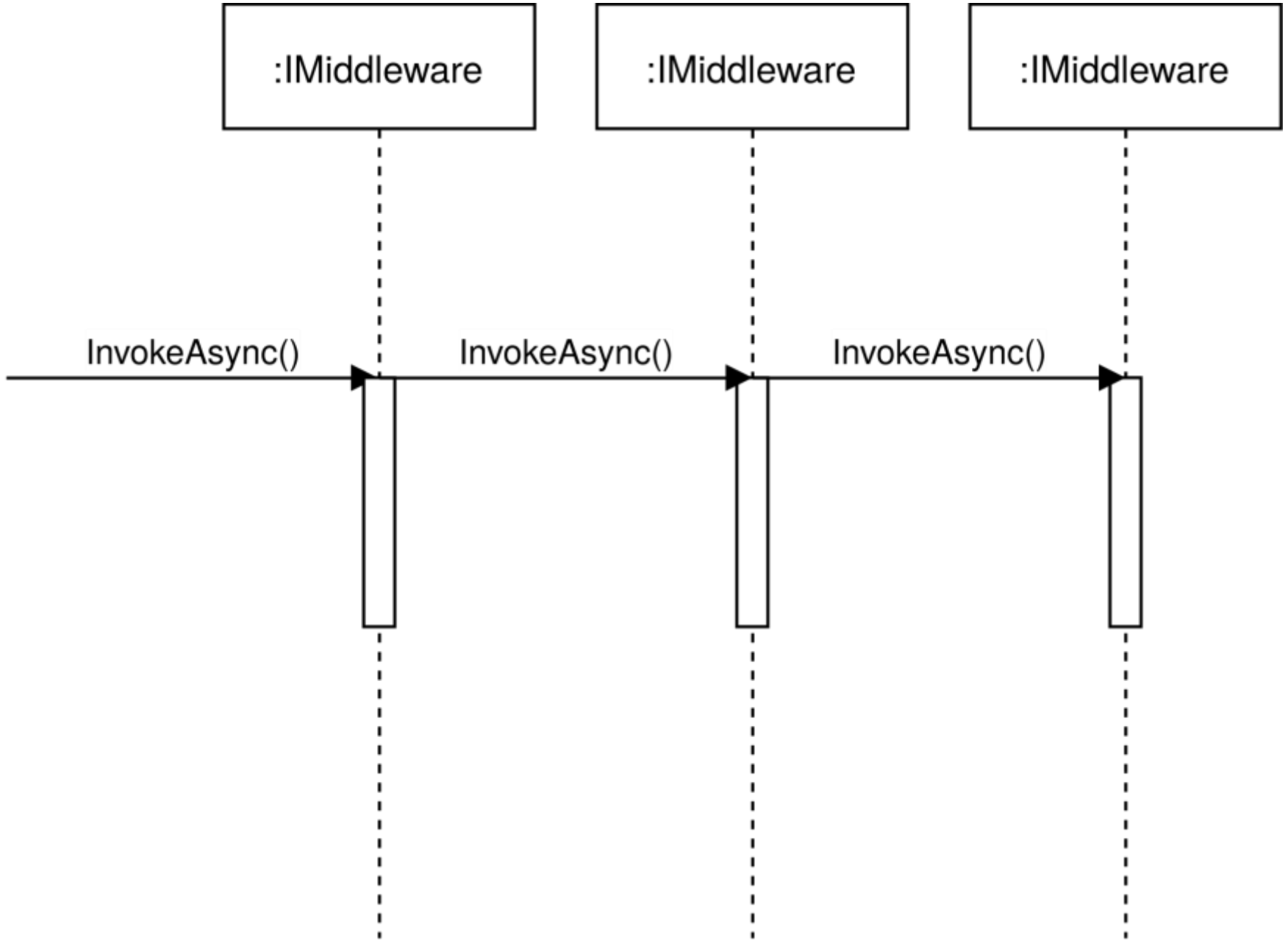
Функции-продолжения (continuations)

```
parseInt id (fun _ -> failwith "Invalid format") "abc"
// System.Exception: Invalid format

parseInt id (fun _ -> failwith "Invalid format") "123"
// 123
```

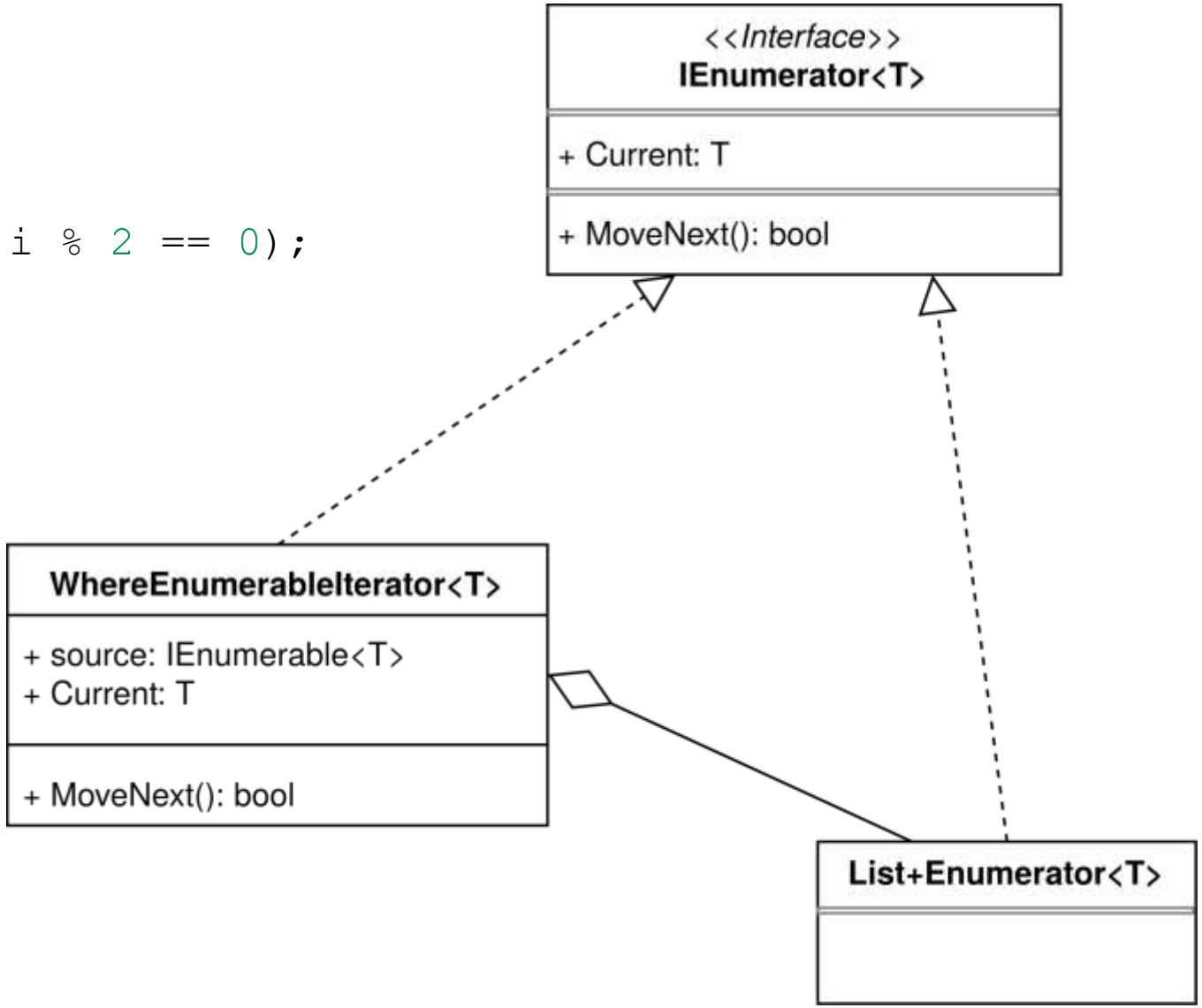
Continuation Passing Style

Цепочка ответственности



Декоратор

```
var list = new List<int>();
list.Add(1);
list.Add(2);
list.Add(3);
var evens = list.Where(i => i % 2 == 0);
```




We need
to go deeper

Прямой конвейер (pipe forward)

```
printfn "%f" (sqrt 2.0) // => 1.414214

2.0 |> sqrt |> printfn "%f" // => 1.414214
```

Ключевое слово `let`

```
let width = 100
let height = 200
let area = width * height
area

// 20000
```

Ключевое слово `let`

```
let width = 100 in
 let height = 200 in
   let area = width * height in
     area

// 20000
```

Ключевое слово `let` и лямбды

```
let width = 100 in
 let height = 200 in
   let area = width * height in
     area

100 |> (fun width ->
 200 |> (fun height ->
   width * height |> (fun area ->
     area)))
```

pipeInto

```
let pipeInto (expression, lambda) =
    lambda expression

pipeInto (100, fun width ->
  pipeInto (200, fun height ->
    pipeInto (width * height, fun area ->
      area)))
```

pipeInto

```
let pipeInto (expression, lambda) =
    log expression
    lambda expression

pipeInto (100, fun width ->                  // => Log: 100
  pipeInto (200, fun height ->               // => Log: 200
    pipeInto (width * height, fun area ->    // => Log: 20000
      area)))
```

pipeInto и Построитель

```
let pipeInto (expression, lambda) =
    log expression
    lambda expression

type LoggerBuilder() =
  member _.Bind(x, f) =
        log x
        f x

  member _.Return(x) =
         x
```

Построитель, Bind, Return

```
let logger = new LoggerBuilder()

logger.Bind(100, (fun width ->
  logger.Bind(200, (fun height ->
    logger.Bind(width * height, (fun area ->
      logger.Return(area)))))))
```

do-нотация

```
logger {
    let! width = 100
    let! height = 200
    let! area = width * height

    return area
} // 20000
```

return

```
option {
  let! a = safe_div 12 4
  let! b = safe_div a 0
  let! c = safe_div b 2

  return c
} // Option<int>.None
```

Другие методы

- Bind
- BindN
- Return
- ReturnFrom
- BindReturn
- BindNReturn
- MergeSources
- MergeSourcesN
- Delay
- Run
- Yield
- YieldFrom
- Combine
- Zero
- For
- While
- TryFinally
- TryWith
- Using
- Quote

goroutine

goroutine

```
let actions = goroutine {
    yield print "0.1"
    yield print "0.2"

    yield! goroutine {
        yield print "1.1"
        yield print "1.2"
        yield print "1.3"
    }

    yield print "0.3"

    yield! goroutine {
        yield print "2.1"
        yield print "2.2"
        yield print "2.3"
    }

    yield print "0.4"
    yield print "0.5"
}
```

goroutine

print “0.1”	print “1.1”	print “2.1”
print “0.2”	print “1.2”	print “2.2”
print “0.3”	print “1.3”	print “2.3”
print “0.4”		
print “0.5”		

goroutine

```
type Action =
    | Atom of (unit -> unit)
    | Fork of Action list

let print v =
    (fun () -> printfn "%A" v)
```

goroutine

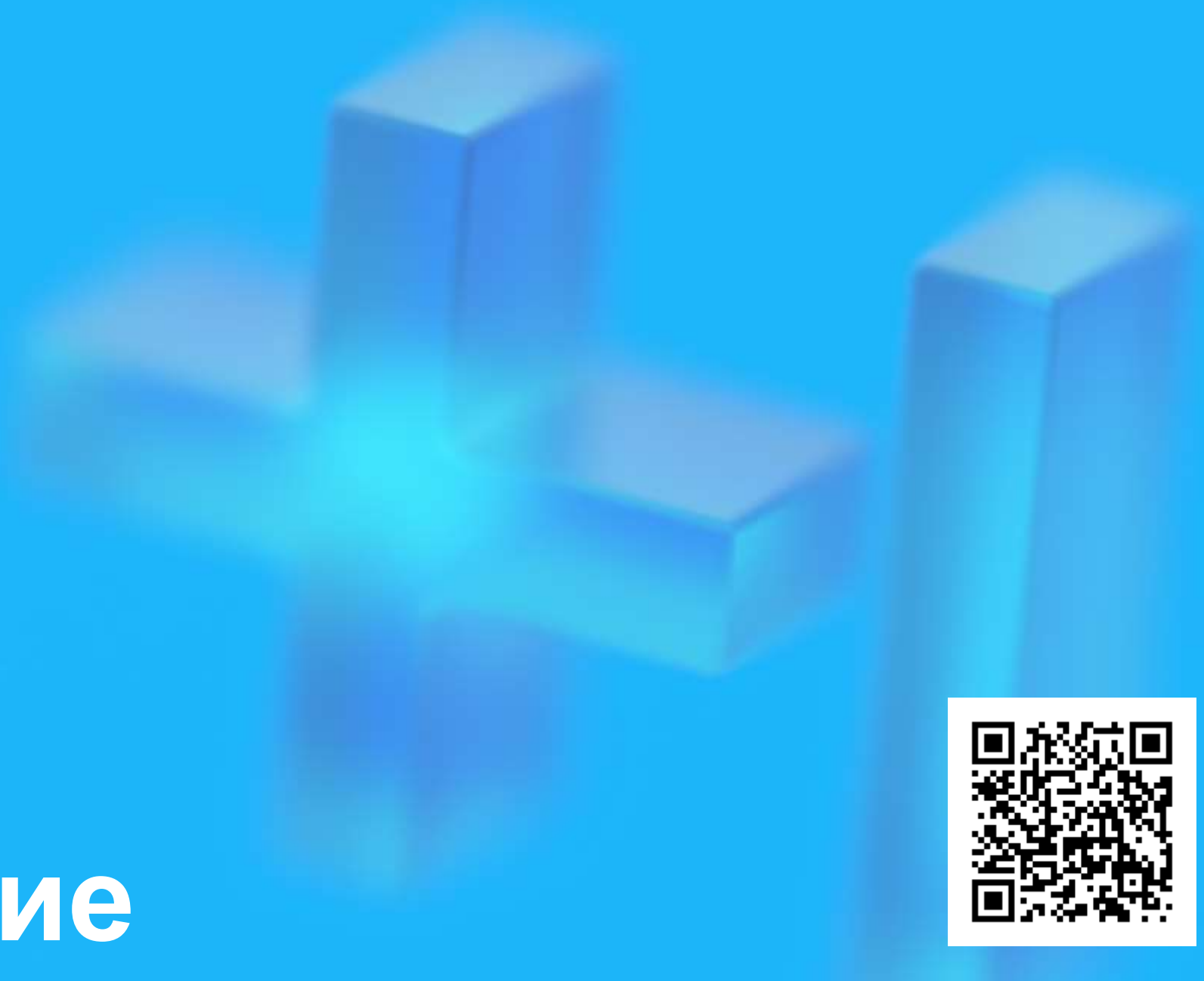
```
let rec run_first_actions = function
    | process::processes ->
        match process with
        | Atom action::actions ->
            action()
            actions::run_first_actions processes
        | Fork actions2::actions1 ->
            actions1::actions2::run_first_actions processes
        | [] -> run_first_actions processes
    | [] -> []
```

goroutine

```
"0.1"
"0.2"
"0.3"
"1.1"
"1.2"
"0.4"
"2.1"
"1.3"
"0.5"
"2.2"
"2.3"
```

+| TI

- Логирование
- Безопасная арифметика
- Построитель и тип-обёртка
- Функции-продолжения
- Ключевое слово let, лямбды
- pipeInto и метод Bind
- do-нотация
- Return
- Другие методы
- goroutine



Спасибо за внимание