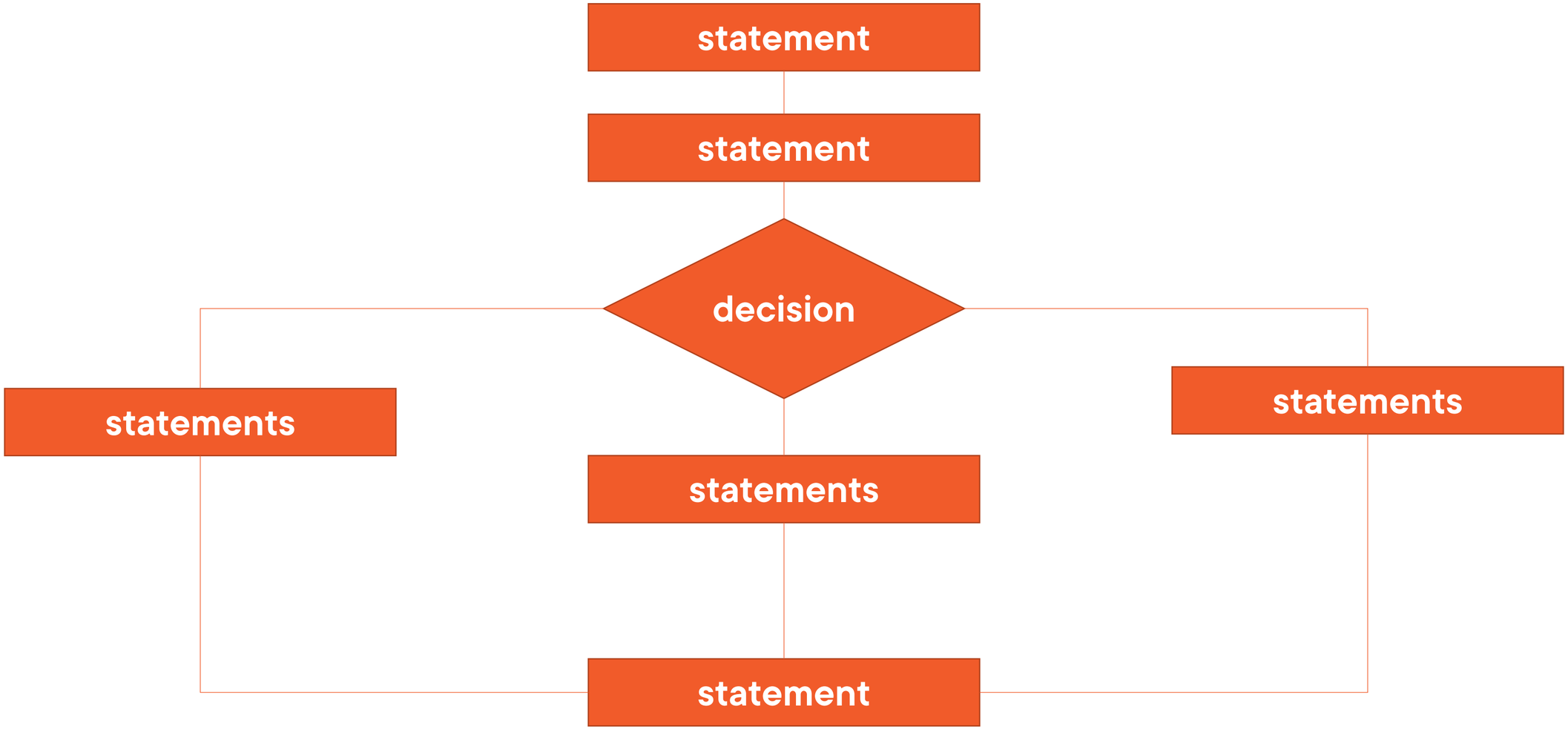
Branching

Michael VanSickle

@vansimke

Branching Logic



Introduction

- If Statements
- Switch Statements
- Deferred Functions
- Panic and Recovery
- Goto Statements

If Statements

```
if test { ... }

if test { ... }
else { ... }

if test { ... }
else if test { ... }

if test { ... }
else if test { ... }
else { ... }

if initializer; test { ... }
```

Demo

Start by showing in lang spec and effective Go

If statements

If-else

If-else if

Initializers

Switch Statements

```
switch test expression {
    case expression1:
        statements
    case expression2, expression3:
        statements
    default:
        statements
}
```

Switch Statements

```
i := 5
switch i {
    case 1:
        fmt.Println("first case")
    case 2 + 3, 2*i+3:
        fmt.Println("second case")
    default:
        fmt.Println("default case")
}
```

second case

Switch Statements

```
i := 999
switch i {
    case 1:
        fmt.Println("first case")
    case 2 + 3, 2*i+3:
        fmt.Println("second case")
    default:
        fmt.Println("default case")
}
```

default case

Switch Statements

```
switch i:=999; i {                          // initializer
    case 1:
        fmt.Println("first case")
    case 2 + 3, 2*i+3:
        fmt.Println("second case")
    default:
        fmt.Println("default case")
}
```

Logical Switch

```
switch i := 8; true {
    case i < 5:
        fmt.Println("i is less than 5")
    case i < 10:
         fmt.Println("i is less than 10")
    default:
        fmt.Println("i is greater than or equal to 10")
}
```

Logical Switch

```
switch i := 8; {                                // true is implied
    case i < 5:
        fmt.Println("i is less than 5")
    case i < 10:
         fmt.Println("i is less than 10")
    default:
        fmt.Println("i is greater than 10")
}
```

Demo

Start by showing in lang spec and effective Go

Switch on condition

Logical switch

fallthrough and continue

initializer

Deferred Functions

Deferred Functions

Deferred Functions

```
func main() {
    fmt.Println("main 1")

    defer fmt.Println("defer 1")

    fmt.Println("main 2")

    defer fmt.Println("defer 2")
}
```

```
main 1
main 2
defer 2
defer 1
```

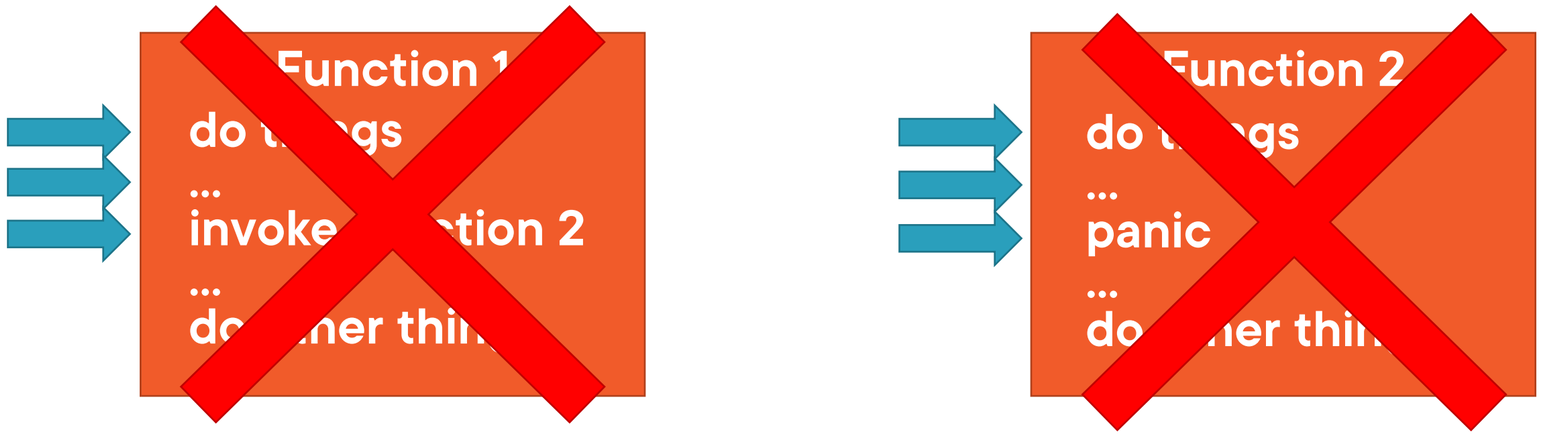
Demo

Start by showing in lang spec and effective Go

Deferred statements

- show opening database and querying it, use this to show why defers execute in FILO order

Panic


Function 1
do things
...
invoke function 2
...
do another thing
Function 2
do things
...
panic
...
do another thing

Panic and Recover

Panic and Recover

```
func main() {
    fmt.Println("main 1")
    func1()
    fmt.Println("main 2")
}

func func1() {
    fmt.Println("func1 1")
    panic("uh-oh")
    fmt.Println("func1 2")
}
```

main 1
func1 1
<<panic>>

Panic and Recover

```
func main() {
    fmt.Println("main 1")
    func1()
    fmt.Println("main 2")
}

func func1() {
    defer func() {
        fmt.Println(recover())
    }()
    fmt.Println("func1 1")
    panic("uh-oh")
    fmt.Println("func1 2")
}
```

```
main 1
func1 1
uh-oh
main 2
```

Demo

Panic and recover

Goto Statements

```
func myFunc() {
  i := 10
  if i < 15 {
     goto myLabel
  }
myLabel:
  j := 42
  for ; i < 15; i++ {
     ...
  }
}
```

Can leave a block

Can jump to containing block

Can’t jump after variable declaration

Can’t jump into another block

Summary

- If Statements
- Switches
- Deferred Functions
- Panic and Recovery
- Goto Statements