# Go Programming Language

## Goroutines and Channels

Department of Computer Science and Engineering

## Go Programming Language
### Goroutines

- In Go, each concurrently executing activity is called a **goroutine**.
- Consider a program that has two functions, one that does some computation and one that writes some output, and assume that neither function calls the other.
- A sequential program may call one function and then call the other, but in a concurrent program with two or more goroutines, calls to both functions can be active at the same time.
- A goroutine is similar to a thread.
- A goroutine is an independent function that executes simultaneously in some separate lightweight threads managed by Go
- The differences between threads and goroutines are essentially quantitative, not qualitative
- When a program starts, its only goroutine is the one that calls the main function, so we call it the main goroutine.

## Go Programming Language
## Goroutines

- New goroutines are created by the go statement
- Syntactically, a go statement is an ordinary function or method call prefixed by the key word **go**.
- A go statement causes the function to be called in a newly created goroutine.
- The go statement itself completes immediately
- **f()** // call f(); wait for it to return
- **go f()** // create a new goroutine that calls f(); don't wait

- Goroutines run in the same address space, so access to shared memory must be synchronized.

## Go Programming Language

## Channels

- A channel is a communication mechanism that lets one goroutine send values to another goroutine.
- Each channel is a conduit for values of a particular type, called the channel's element type.
- The type of a channel whose elements have type int is written **chan int**.
- To create a channel, we use the built-in make function:
- **ch := make(chan int)** // ch has type 'chan int‘
- Channel is a reference to the data structure created by make.
- When we copy a channel or pass one as an argument to a function, we are copying a reference, so caller and callee refer to the same data struc ture.
- The zero value of a channel is nil.
- Two channels of the same type may be compared using ==. The comparison is true if both are references to the same channel data structure. A channel may also be compared to nil.

# Go Programming Language

## Channels - Operations

- A channel has two principal operations, **send** and **receive**, collectively known as communications.
- A send statement transmits a value from one goroutine, through the channel, to another goroutine executing a corresponding receive expression.
- Both operations are written using the **<-** operator.
- In a send statement, the <- separates the channel and value operands.
- In a receive expression, <- precedes the channel operand.
- A receive expression whose result is not used is a valid statement.
- **ch <- x** // a send statement
- **x = <- ch** // a receive expression in an assignment statement
- **<- ch** // a receive statement; result is discarded

## Go Programming Language

### Channels - Operations

- Channels support a third operation, close, which sets a flag indicating that no more values will ever be sent on this channel
- Receive operations on a closed channel yield the values that have been sent until no more values are left
- To close a channel, we call the built-in close function: **close(ch)**

## Go Programming Language
### Unbuffered Channels

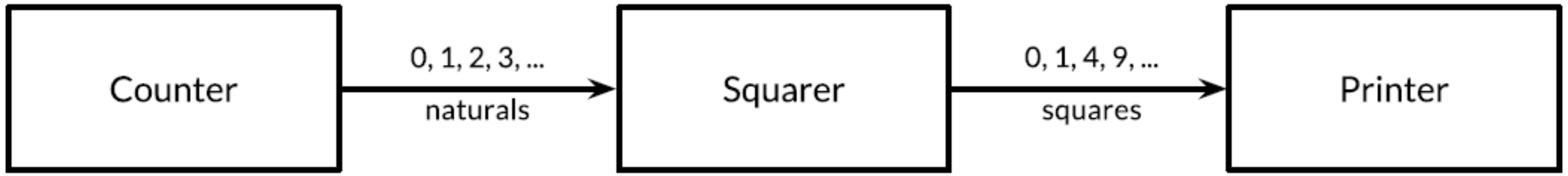
- A send operation on an unbuffered channel blocks the sending goroutine until another goroutine executes a corresponding receive on the same channel, at which point the value is transmitted and both goroutines may continue.
- If the receive operation was attempted first, the receiving goroutine is blocked until another goroutine performs a send on the same channel.
- Communication over an unbuffered channel causes the sending and receiving goroutines to synchronize. Because of this, unbuffered channels are sometimes called synchronous channels

## Buffered Channels

- A buffered channel has a queue of elements.
- The queue’s maximum size is determined when it is created, by the capacity argument to make.
- **ch = make(chan string, 3)**
- We can send up to three values on this channel without the goroutine blocking
- A fourth send statement would block.

## Go Programming Language
## Pipelines

- Channels can be used to connect goroutines together so that the output of one is the input to another. This is called a pipeline.



- The first goroutine, counter, generates the integers 0, 1, 2, ..., and sends them over a channel to the second goroutine, squarer, which receives each value, squares it, and sends the result over another channel to the third goroutine, printer, which receives the squared values and prints them

## Unidirectional Channels

- To document the intent and prevent misuse, the Go type system provides unidirectional channel types that expose only one or the other of the send and receive operations.
- The type chan <- int, a send-only channel of int, allows sends but not receives.
- The type <- chan int, a receive-only channel of int, allows receives but not sends.

```
func counter(out chan int)
func squarer(out, in chan int)
func printer(in chan int)
```

PES UNIVERSITY

CELEBRATING 50 YEARS

## THANK YOU

**Suresh Jamadagni**

Department of Computer Science and Engineering