The Go Programming Language

Overview

- ➔ Go, commonly known as Golang, is an open source programming language that makes it easy to build simple, reliable, and efficient software.

- ➔ It was conceived on November 10, 2009 by Google engineers Robert Griesemer, Rob Pike, and Ken Thompson.

Hardware Limitations

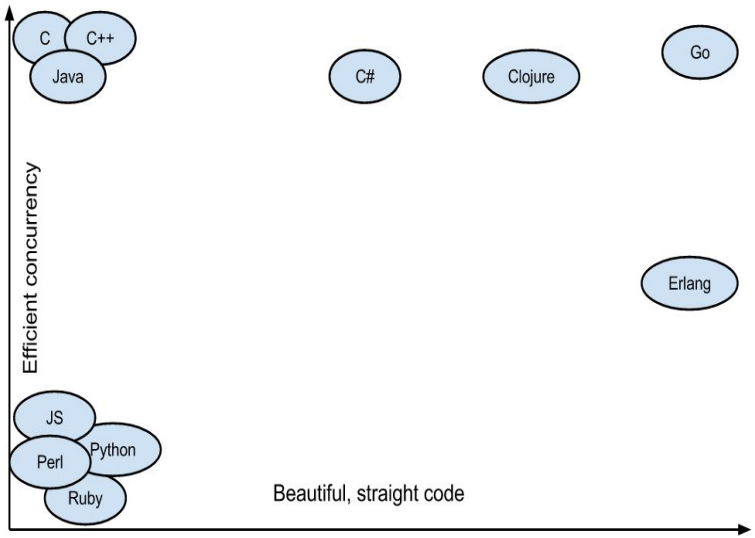
- ➔ Current scenario
  - ◆ Multi-core processors
  - ◆ Hyper Threading
  - ◆ More cache for increased performance
- ➔ Issues?
  - ◆ Not scalable
  - ◆ Higher Cost
- ➔ More efficient software is the only go (LOL)

Multithreading

- ➔ Most modern programming languages support multithreading
- ➔ Problems?
  - ◆ Concurrency
  - ◆ Thread-locking
  - ◆ Race Conditions
  - ◆ Deadlocks
- ➔ Building a multithreaded application is difficult on these languages!

Goroutines for concurrency!

- ➔ Go has goroutines instead of threads
- ➔ Consume at most 2 KB of memory from the heap
- ➔ Benefits:
  - ◆ growable segmented stacks
  - ◆ faster startup time than threads
  - ◆ built-in primitives to communicate safely between themselves
  - ◆ single goroutine can run on multiple threads



Go brings best of both Worlds!

- ➔ Go is a compiled language - performance is nearer to Low-level Languages

- ➔ Uses Garbage Collection for removal and allocation of objects

Language Design

Declaration of variables

```
var i int
```

Variables declared without an explicit initial value are given their *zero value*.

Inserting Packages

```
import (
	"fmt"
	"math"
)
```

Constants

```
const Pi = 3.14
```

Defining a new type

```
type ipv4addr uint32
```

For Loop Syntax

```
func main() {
	sum := 0
	for i := 0; i < 10; i++ {
		sum += i
	}
	fmt.Println(sum)
}
```

Structures

```
type Vertex struct {
	X int
	Y int
}
```

Arrays

```
var a [2]string
a[0] = "Hello"
a[1] = "World"
```

Function Definition

```
func pow(x, n, lim float64) float64
{
	if v := math.Pow(x, n); v < lim
{
		return v
	} else {
		fmt.Printf("%g >=
%g\n", v, lim)
	}
	return lim
}
```

Switch Case

```
switch os := runtime.GOOS; os {
	case "darwin":
		fmt.Println("OS X.")
	case "linux":
		fmt.Println("Linux.")
	default:
		// freebsd, openbsd,
		// plan9, windows...
		fmt.Printf("%s.", os)
	}
```

Language Design

Basic Types in Go

- Bool
- String
- int int8 int16 int32 int64
- uint uint8 uint16 uint32 uint64 uintptr
- byte // alias for uint8
- rune // alias for int32 //represents a Unicode code point
- float32 float64
- complex64 complex128

Maps

```
var m map[string]Vertex
m = make(map[string]Vertex)
m["Bell Labs"] = Vertex{
        40.68433, -74.39967,
}
fmt.Println(m["Bell Labs"])
```

Goroutines

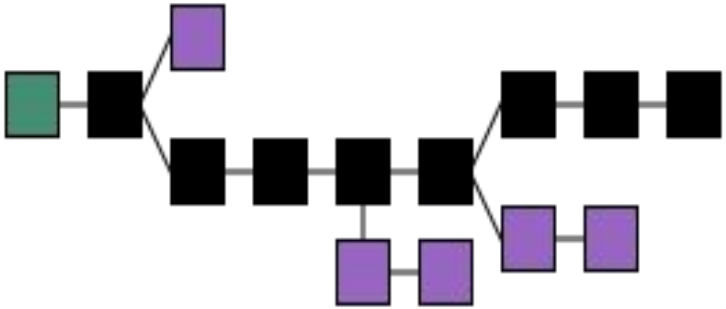
A goroutine is capable of running concurrently with other functions. To create a goroutine we use the keyword go followed by a function invocation:

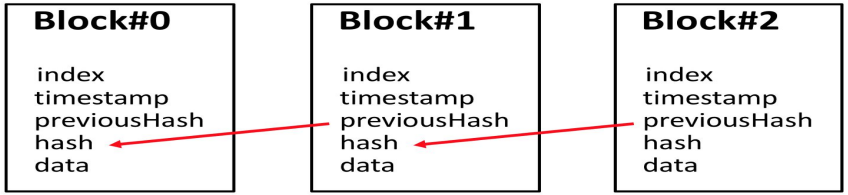
```
func main() {
  go f(0)
  var input string
  fmt.Scanln(&input)
}
```

Building a simple Blockchain Application using Golang

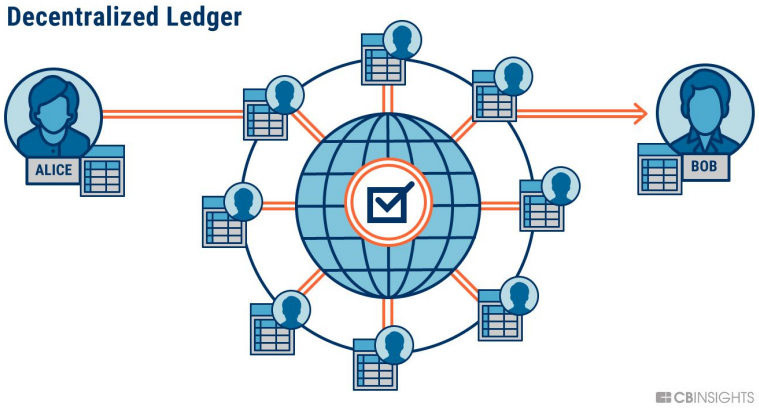
What is a Blockchain?

- A blockchain is a growing list of records, called *blocks*, which are linked using **cryptography**.
- Each block contains a cryptographic **hash value of the current block, a hash value of the previous block, a timestamp, and transaction data**





ssaurel.com/blog



Features of a Blockchain

Good Morning =
E526F13918F16C1C65FC4AC51ABE8B5B991769AE6718495A7AD9984406A14A2C

Good Mornin =
551294185A8D2AD6A0C72EC63FF7D68F4C4AC538B334D3256AFE21DE001E7C26

- Decentralization - Distributed Ledger
  - Accessibility of data
  - Consensus by a set of peers and not computers
  - No central authority
- Immutability
  - Resistance to change because of Hash Values
- Security
  - Forgery and mutation detection
  - Highly secure and resistant to forgery and mutation
- Proof of Work
  - Confirms transactions and produces new blocks to the chain

Code Snippets

```
// Blockchain ...
type Blockchain struct {
    Chain               []Block       `json:"chain"`
    CurrentTransactions []Transaction `json:"current_transactions
    Nodes               []string      `json:"nodes"`
}

// Block ...
type Block struct {
    Index        uint64        `json:"index"`
    Timestamp    time.Time     `json:"timestamp"`
    Transactions []Transaction `json:"transactions"`
    Proof        uint64        `json:"proof"`
    PreviousHash string        `json:"previous_hash"`
}

// Transaction ...
type Transaction struct {
    Sender    string `json:"sender"`
    Recipient string `json:"recipient"`
    Amount    uint64 `json:"amount"`
}
```

```
func (bc *Blockchain) validChain(chain []Block) bool {
    lastBlock := chain[0]
    currentIndex := 1
    for currentIndex < len(chain) {
        block := chain[currentIndex]
        if block.PreviousHash != hash(lastBlock) {
            return false
        }
        if !validProof(lastBlock.Proof, block.Proof, lastBlock.PreviousHash) {
            return false
        }
        lastBlock = block
        currentIndex++
    }
    return true
}
```

```
func (bc *Blockchain) newBlock(proof uint64, previousHash string) Block {
    block := Block{
        Index:        uint64(len(bc.Chain) + 1),
        Timestamp:    time.Now(),
        Transactions: bc.CurrentTransactions,
        Proof:        proof,
        PreviousHash: previousHash,
    }
    bc.CurrentTransactions = nil
    bc.Chain = append(bc.Chain, block)
    return block
}

func (bc *Blockchain) newTransaction(sender, recipient string, amount uint64) uint64 {
    transaction := Transaction{
        Sender:    sender,
        Recipient: recipient,
        Amount:    amount,
    }
    bc.CurrentTransactions = append(bc.CurrentTransactions, transaction)
    return bc.lastBlock().Index + 1
}
```

```
// NewBlockchain ..
func NewBlockchain() *Blockchain {
    bc := new(Blockchain)
    bc.CurrentTransactions = nil
    bc.Chain = nil
    bc.Nodes = nil
    bc.newBlock(uint64(100), "1")
    return bc
}
```

Code Snippets

```
161
162 func hash(block Block) string {
163     jsonBytes, _ := json.Marshal(block)
164     hashBytes := sha256.Sum256(jsonBytes)
165     return hex.EncodeToString(hashBytes[:])
166 }
167
168 func validProof(lastProof, proof uint64, lastHash string) bool {
169     guess := fmt.Sprintf("%x%x%x", lastProof, proof, lastHash)
170     guessBytes := sha256.Sum256([]byte(guess))
171     guessHash := hex.EncodeToString(guessBytes[:])
172     return guessHash[:4] == "0000"
173 }
174
175 // Mine ...
176 type Mine struct {
177     Message      string        `json:"message"`
178     Index        uint64        `json:"index"`
179     Transactions []Transaction `json:"transactions"`
180     Proof        uint64        `json:"proof"`
181     PreviousHash string        `json:"previous_hash"`
182 }
183
```

Thank You!