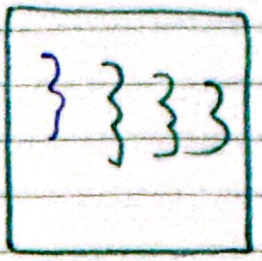
Property of the half blood prince
RPC and Threads

Golang is a really good language to build performant system since it is type safe, memory safe and garbage collected.

Threads will be the main tool to manage concurrency.



you have a program and one adress space in a serial program you will have just one thread of execution one program, one set of registers, one stack that are describing the current state of program.

In a threaded program you will have multiple threads, with seperate program counter, seperate set of registers and a seperate stack for each of threads so they can have their own thread of control and be executing in different part of prog
Through threads we can make use of multi-core parallesim, that is really beneficial if you have a program that can run its CPU cycle on different cores.

- I/O concurrency
- Parallasim
- Convenience

} use of threads

Process is a single program that you are running and sort of single address space a bunch of memory for the process, and inside the process you might have multiple threads running go program → creates one unix process and sort of memory area, and when your go process creates go routines they are sitting inside that one process.

Many threads running can cause race conditions, hence we use locks

```
mu.lock()
n=n+1
mu.unlock()
```

- Coordination

When we want different threads to interact with each other that is called coordination, we can make use of channels (sending data from one thread to another) sync.cond (kick to let know continue what it was doing), wait Group (running a lot of go routines and waiting for them to finish.

Deadlock

$T_1 \longrightarrow T_2$

Both threads cannot do anything.

- Web Crawlers

```
type fetchState struct{
    mu    sync.mutex
    fetched  map[string]bool
}

func concurrentMutex(url string, fetcher Fetcher, f
*fetchstate){
    f.mu.lock()
    already : f.fetched[url]
    f.mu.Unlock()
    if already{
        return}

    urls, err := fetcher.Fetch(url)
    if err != nil{return}
    var done sync.waitGroup
    for _, u = range urls{
        done.add(1)
        go func(u string){
            concurrentMutex(u, fetcher, f)
done.wait}(go.Done
```