Hide-And-Seek

Indian multiplayer game in realtime

1. Abstract

Hide-And-Seek project is a 2D-Multiplayer game built with consideration with the high-performance availability and ease of process flow.

The first version was built under pure Node.js to achieve bare-metal performance.

Pure Node.js is my first choice always to deal with Real-time API development till,I learn and feel Golang is a better choice compared to Node.js.

Golang is much lighter and blazing fast compare to Node.js and takes strict on memory management by design, it will especially be designed for hard-core server development with a low memory footprint.

This white-paper will describe how to keep low-configuration hardware to fuel the multiplayer game to manage players as possible within the system constraints.

2. Resource

Hide-And-Seek designed with taking into consideration of low hardware capacity.

To keep Hide-And-Seek running,The following resource Required:

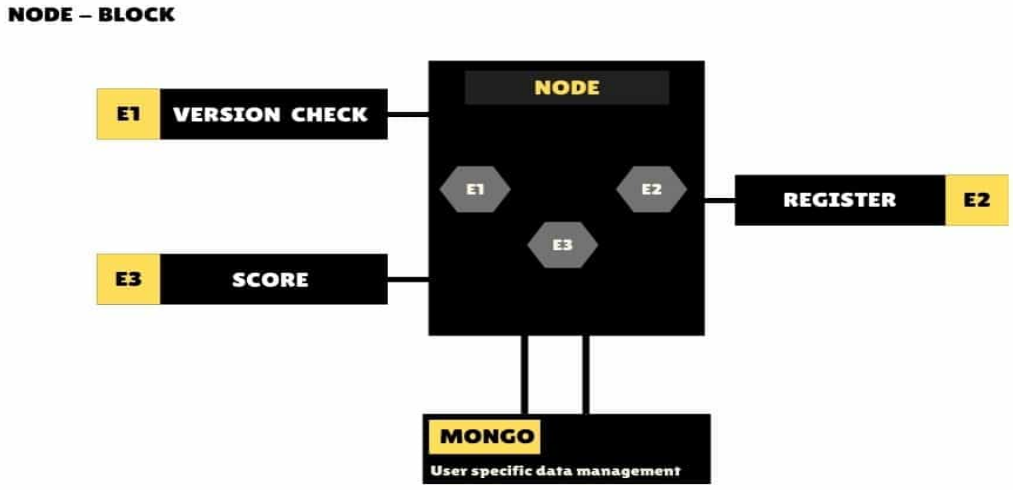
Node.js	**Version manager**
Golang	**Websocket Management**
Redis	**Game lobby Management**
MongoDb	**Game stats Management(Temporary Suspended)**

3. Separation of concern

Each Block of the resource is a delegate to do tasks rapidly and smoothly to achieve the desired goal.

3.1. Node Block

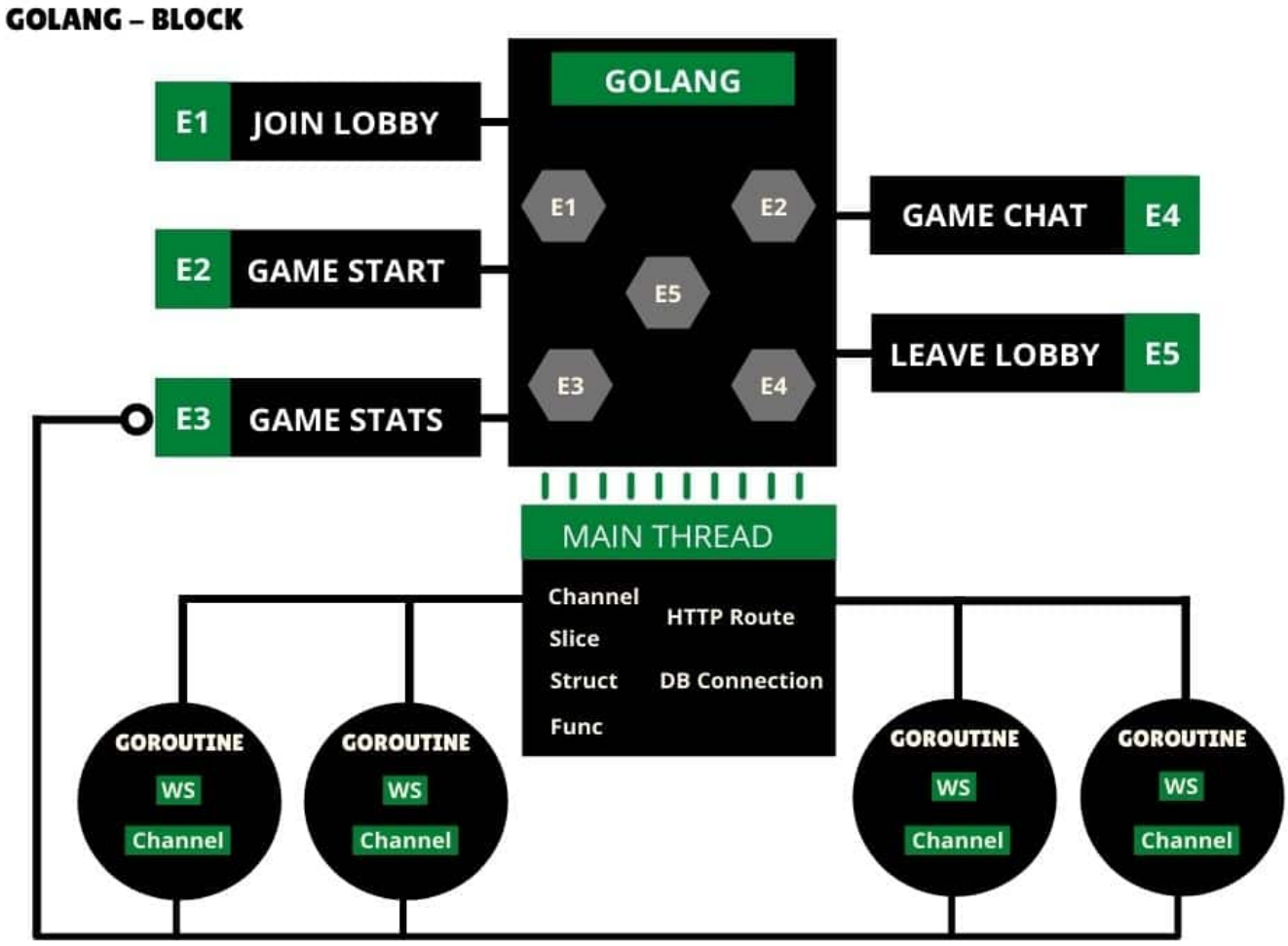
Node.js will delegate for simple version manager and to keep user information sync with the database.



3.2. Golang Block

Golang deal with all Websocket operation and realtime data with strict memroy management

Every keypress event during gameplay will dispatch to all players in the lobby within a few milliseconds



Websocket request & response data flow handle in Goroutines

Note: Each Websocket client will handle independently and simultaneously with a tiny function called Go-Route, Go-Route initially created with only 2KB of stack size.

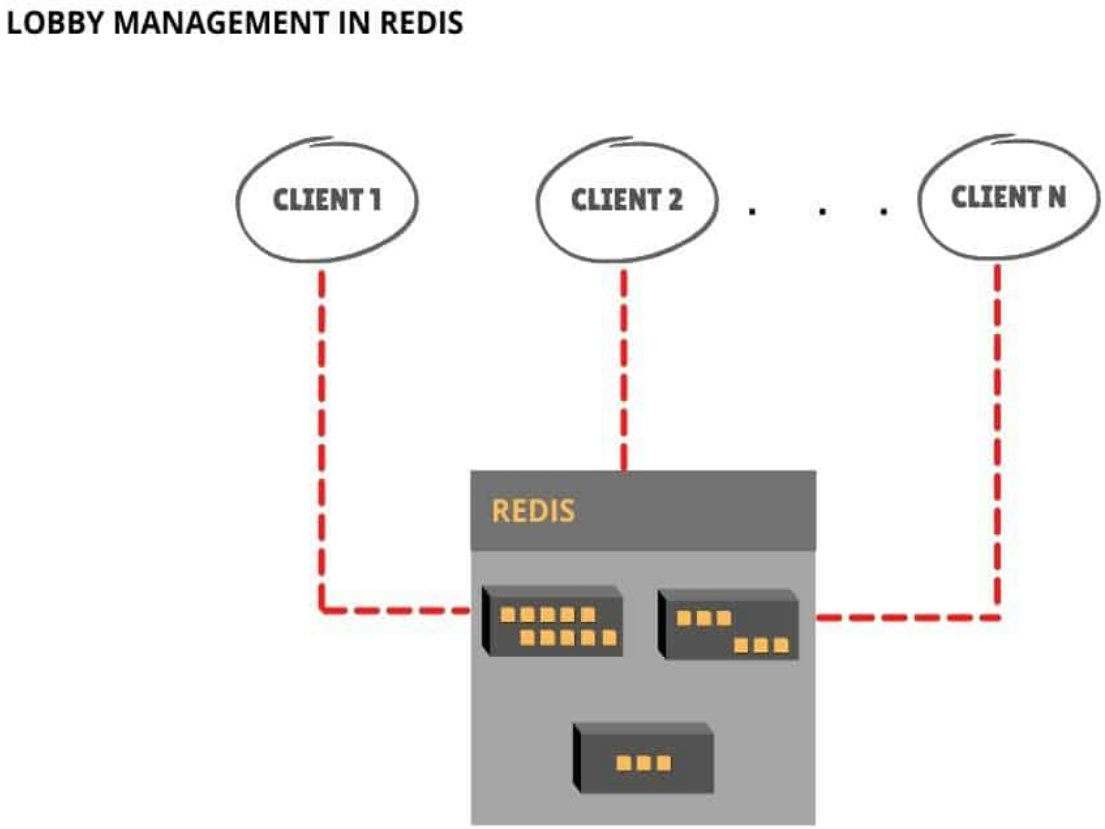
3.3. Redis Block

Redis block is holding every full and partial game lobby key in the form of a Redis-List.

Each player either initiate the lobby(First) or join lobby(Rest) by 'lpush' to maintain the unsorted list by insertion order.

The list contains the tag name(Combination of game version and map version) And the value will be player hash.

Redis lobby retains in memory till All users got disconnect before game start or Game started by admin.



3.4. Mongo Block

Mongo Temporary suspend for the operation to keep victory record of the player.

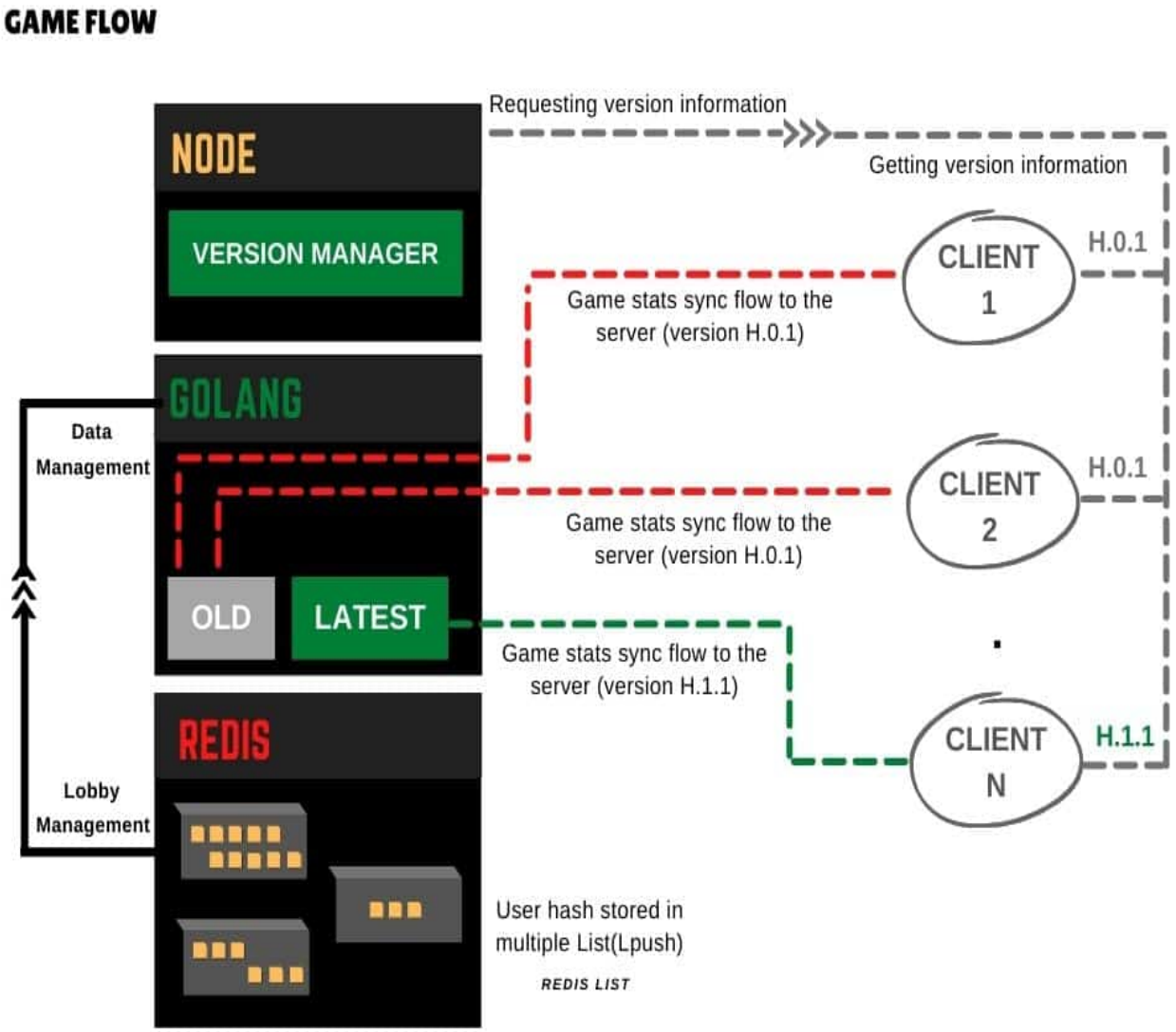
MongoDB will use in case of a Data management scenario involved shortly But for the present moment of Hide-And-Seek full focus on managing the resource to scale out a maximum number of players in real-time management instead of data-management.

Once the Game grows, Mongo service will resume and allow Node-Block to consume and write the game stats on Disk.

Note: In case of MongoDB engine require,The mongo pool will be instantiated with Node.js

4. Overall Game Flow

Take a look at full process flow where every segment communicate with each other to fulfill the request/ response cycle of the user(Client)



5. Trade-off

In the case of few resources and more task management, an organization has to think about trade offs between working approach and user experience to achieve smooth operation.

In my opinion, user-experience is more prioritized than the working approach.

The work-approach will become more rigorous with fewer resources in hand to deliver a better user experience(Performance).

Keep your eyes on every task and make each tiny more useful rather than depend on a system to automate the auxiliary job.

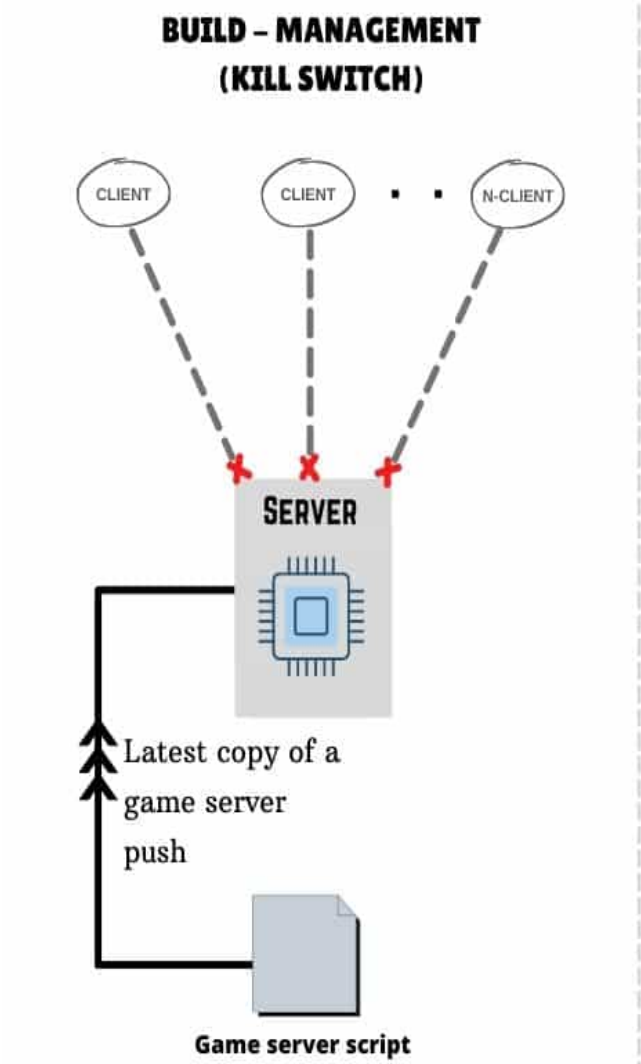
Keep system less busy with tasks in which it requires to delegate continuously and keep consuming more memory(Even for a shorter amount of period).

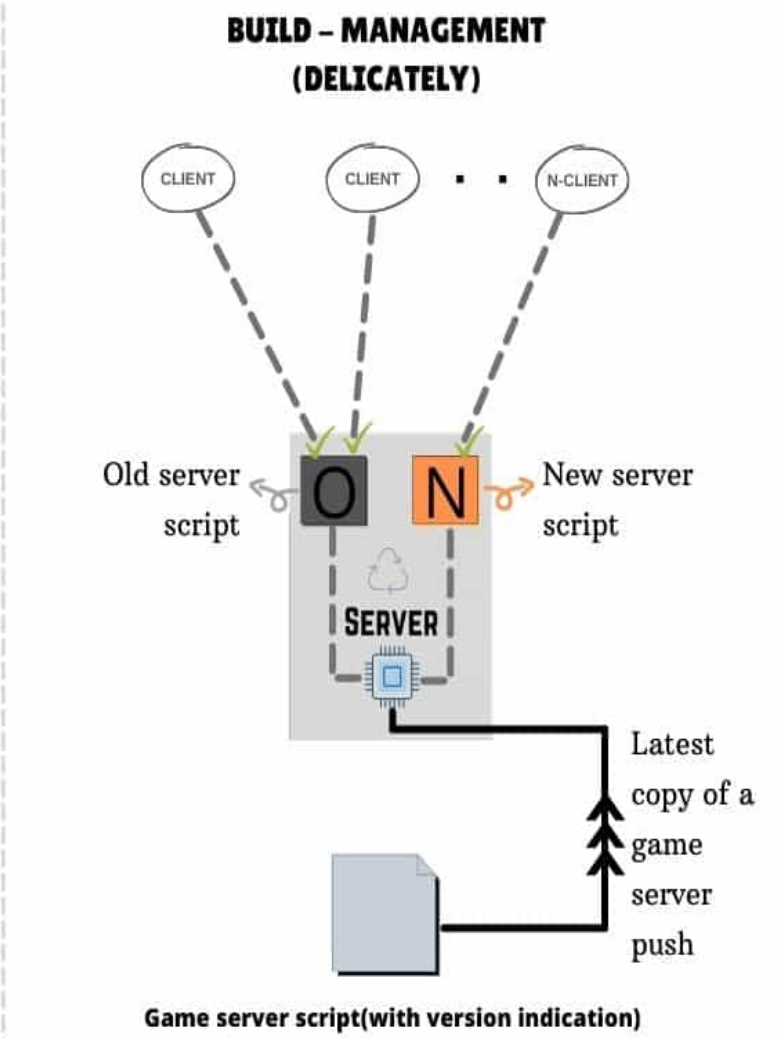
6. Build Management

After experiencing a few build management systems, I've decided to keep the build system memory footprint as small as possible.

Which lead to the conclusion that the manual build approach will be memory-efficient to keep service up-to-date with more system memory to utilization for game.

Hide-And-Seek uses Version manager build-system approach(Delicately).





7. Server Optimization

Keep track of every Linux process and disable that service that is not suitable/required or unnecessary for the game server, Which makes the server heavier.

Hide-And-Seek is designed for trial basis game-server, Thus the initial version will not keep the record of each victory or game stats, Which leads to stop MongoDB service from Linux services.

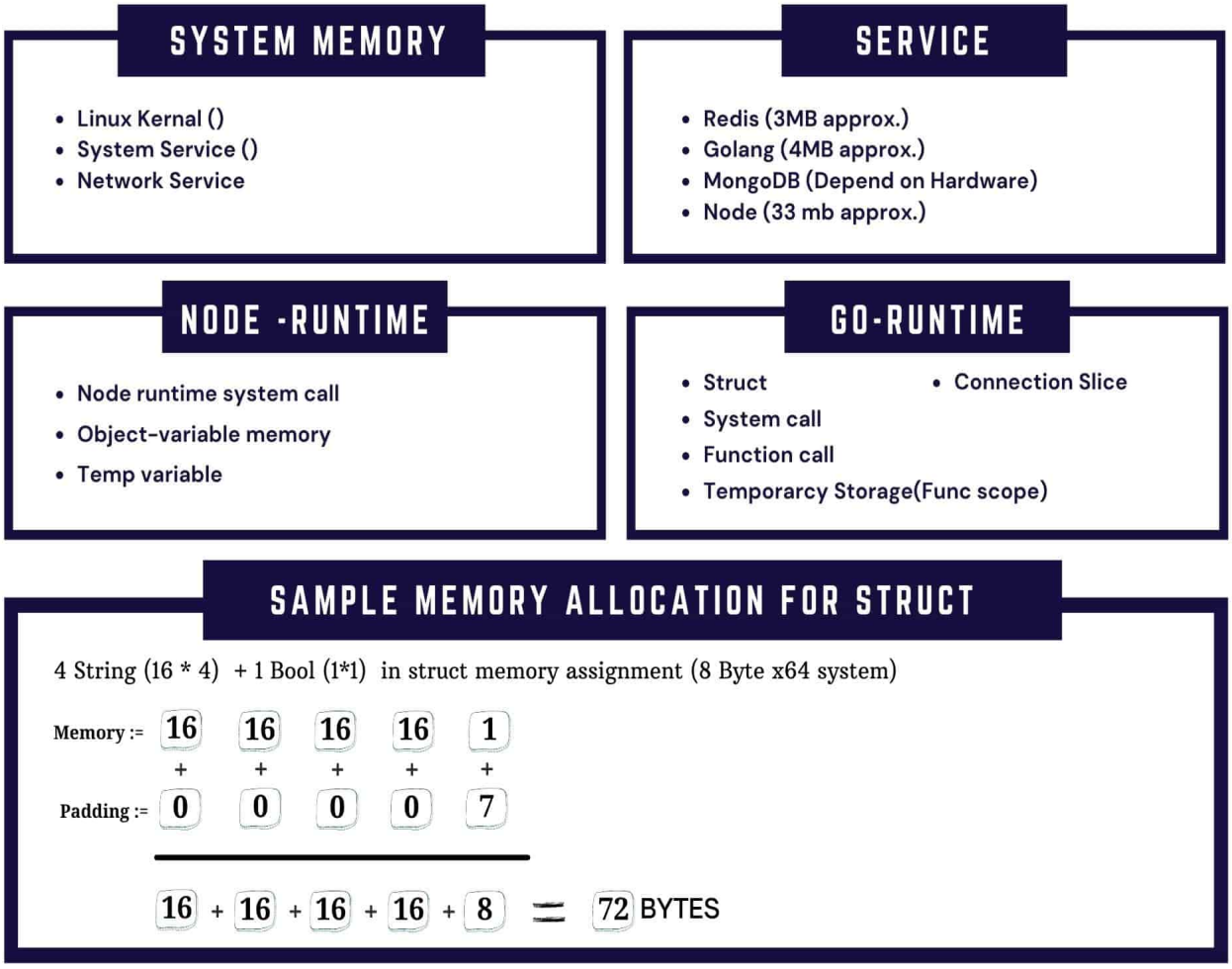
Useful commands to identify / Stop linux Services

htop or ps aux -A	List of process with memory
sudo pmap -x <Process id>	Get memory of process id[Total Heap,RSS,Dirty Memory]
systemctl stop <Service>	Stop service from linux service list
kill -9 <Process id>	Kill process with SIGKILL

8. Enumeration (approx.)

Every Byte matters, In this situation, always expect that we have less memory to execute, which will help to write better code

Hide-And-Seek memory(Approx.) deconstruction as follow:



Most of the time Golang runtime execution will deal with system-call(Golang stack memory) and rest memory allocated to a custom-defined struct(User struct, join lobby struct, etc.) where player's data dumped into the designated struct.

In this scenario Tracking, struct memory allocation is also a crucial part of memory management.

9 Room For Enhancement

I started Multiplayer logic with Node.js and switched to Golang to get more performance, yet some languages are superior than Golang to achieve more performance-optimized code and blazing fast speed. I can think of two languages Rust and C++ to achieve the desired performance and speed. So maybe the sky has a limit our imagination don't

10 Conclusion

Every step should follow while having fewer resources to implement stiff task management, these strategies will pay off with smooth operation eventually.

'Every Byte Consider As lunch-Bite of a Memory' if you eat less or more you get out of shape, Try to achieve perfection.

Many feedback daunt me but it might be worth try it,so submit your feedback.

Twitter : https://twitter.com/jswalker_in

Email : jayesh@jswalker.in