GO RUNTIME / PLAYBOOK

Quick Answers Edition

14 concise answers selected from the complete Go Runtime & Concurrency Interview Playbook.

FREE SAMPLE	2026 EDITION	ENGLISH PDF

A useful sample, not a teaser

Read every 30-second answer from Part I. The full edition adds the reasoning, implementation details, code, version notes, exercises, and extended question bank behind these answers.

FULL EDITION	31 pages · $19

01 / GARBAGE COLLECTION

How does Go garbage collection work?

30-SECOND ANSWER

Modern Go uses a concurrent tracing collector. Conceptually, tri-color marking separates objects into white, gray, and black sets. Reachable objects are traced from roots; objects left unreachable can be reclaimed.

02 / SCHEDULER

What do G, M, and P represent?

30-SECOND ANSWER

G is a goroutine, M is an operating-system thread, and P is the scheduler resource required to execute Go code. An M normally needs a P to run a G.

03 / MEMORY

How does Go allocate small and large objects?

30-SECOND ANSWER

Allocation is optimized through per-P caches, central free lists, and the heap. Small objects use size classes; sufficiently large objects are allocated directly from the heap.

04 / CHANNELS

What information does a channel maintain?

30-SECOND ANSWER

A channel maintains element metadata, buffer state for buffered channels, send/receive indices, and queues of blocked senders and receivers. The runtime synchronizes access to this state.

05 / MAPS

Are built-in maps safe for concurrent use?

30-SECOND ANSWER

No. Concurrent access is safe only when all accesses are reads and no goroutine mutates the map. Concurrent reads and writes require synchronization or a concurrency-safe abstraction.

06 / MAPS

When is sync.Map appropriate?

30-SECOND ANSWER

sync.Map is optimized for specific concurrent workloads, especially write-once/read-many caches or cases where goroutines mostly access disjoint key sets. A regular map protected by a Mutex or RWMutex is often clearer.

07 / CONCURRENCY

How does select behave?

30-SECOND ANSWER

select waits until one channel operation can proceed. If several cases are ready, one ready case is chosen pseudo-randomly. A default case makes the operation non-blocking.

08 / LANGUAGE MECHANICS

How does defer work?

30-SECOND ANSWER

Deferred calls run in last-in, first-out order when the surrounding function returns, including during panic unwinding. The compiler and runtime use several strategies to reduce defer overhead.

09 / INTERFACES

What is stored in an interface value?

30-SECOND ANSWER

Conceptually, an interface value contains dynamic type information and dynamic value information. A non-empty interface also needs method-dispatch information for the implemented interface.

10 / CONTEXT

What problem does context solve?

30-SECOND ANSWER

context.Context carries cancellation, deadlines, and request-scoped values across API boundaries. It lets related goroutines stop work when the originating request is canceled or times out.

11 / ALLOCATION

What is the difference between make and new?

30-SECOND ANSWER

new(T) allocates zeroed storage for T and returns *T. make initializes a slice, map, or channel and returns the initialized value itself, not a pointer to it.

12 / SYNCHRONIZATION

How does sync.Mutex behave under contention?

30-SECOND ANSWER

The fast path acquires an unlocked mutex atomically. Under contention, goroutines may briefly spin and then park. The runtime includes fairness mechanisms to prevent an unlucky waiter from starving indefinitely.

13 / COMPILER

What is escape analysis?

30-SECOND ANSWER

Escape analysis is a compile-time process that helps the compiler decide whether a value can remain on a stack or must be allocated on the heap.

14 / DATA STRUCTURES

What is the difference between an array and a slice?

30-SECOND ANSWER

An array has a fixed length that is part of its type and assignment copies the array value. A slice is a descriptor over an underlying array, containing a pointer, length, and capacity.

Ready to go deeper?

The full edition turns each short answer into an interview-ready explanation with runtime context, caveats, production guidance, and concrete examples.

•	Complete Part I explanations for all core topics
•	Extended question bank with lower-level runtime coverage
•	Code examples, concurrency exercises, and interview checklist
•	Version-aware notes that separate guarantees from implementation details

GET THE COMPLETE PLAYBOOK **$19**

Return to the product page to get the full edition.