Before a benchmark starts executing, Go identifies (/ filters) the benchmarks.

Once benchmarks are filtered, a pointer to B is created which contains all the filtered benchmarks in a properly called `benchFunc`. As the name suggests, this properly contains a function to invoke each benchmark.

```go
var bs []InternalBenchmark
for _, Benchmark := range benchmarks {
    if _, matched, _ := bstate.match.fullName( c: nil, Benchmark.Name); matched {
        bs = append(bs, Benchmark)
        benchName := benchmarkName(Benchmark.Name, maxprocs)
        if l := len(benchName) + bstate.extLen + 1; l > bstate.maxLen {
            bstate.maxLen = l
        }
    }
}
main := &B{
    common: common{
        name:  "Main",
        w:     os.Stdout,
        bench: true,
    },
    importPath: importPath,
    benchFunc: func(b *B) {
        for _, Benchmark := range bs {
            b.Run(Benchmark.Name, Benchmark.F)
        }
    },
    benchTime: benchTime,
    bstate:    bstate,
}
```

↑ filter

Role of bstate?

↑ each benchmark will eventually run via benchfunc

default benchTime = 1sec

To be able to run each benchmark, Go calls a function called runN on the pointer of B.

This means Go calls:

main.runN(1)

to run a single iteration on the runtime instance of benchmark (/main)

Technically, `main.runN(1)` is just an entrypoint to start the execution of filtered benchmarks via benchFunc.

```go
func (b *B) runN(n int) {
    benchmarkLock.Lock()
    defer benchmarkLock.Unlock()
    ctx, cancelCtx := context.WithCancel(context.Background())
    defer func() {
        b.runCleanup(normalPanic)
        b.checkRaces()
    }()
    // Try to get a comparable environment for each run
    // by clearing garbage from previous runs.
    runtime.GC()
    b.resetRaces()
    b.N = n
    b.loop.n = 0
    b.loop.i = 0
    b.loop.done = false
    b.ctx = ctx
    b.cancelCtx = cancelCtx

    b.parallelism = 1
    b.ResetTimer()
    b.StartTimer()
    b.benchFunc(b)
    b.StopTimer()
    b.previousN = n
    b.previousDuration = b.duration
```

The core functionality of runN includes:

1) Resetting the timer
2) Starting the timer
3) Stopping the timer.
4) Running the benchmark function

5) Tracking previous Iterations & previous Duration.

Here, `benchfunc` is a wrapper function which iterates through all benchmark definitions & invokes `Run` for each definition

```
        importPath: importPath,
        benchFunc: func(b *B) {
            for _, Benchmark := range bs {
                b.Run(Benchmark.Name, Benchmark.F)
            }
        },
```

One could say, main.RUNN(i)
is the entrypoint to run all
filtered benchmarks.