Run() is the entrypoint to Run an individual benchmark.

It accepts a function name & a benchmark function:

It is responsible for the following:

① Creating a pointer to B, for the individual benchmark

② Invoking a function to run the first iteration of the benchmark.

③ Invoking a function to run N iterations.

```
func (b *B) Run(name string, f func(b *B)) bool {
    sub := &B{
        common: common{
            signal:  make(chan bool),
            name:    benchName,
            parent:  &b.common,
            level:   b.level + 1,
            creator: pc[:n],
            w:       b.w,
            chatty:  b.chatty,
            bench:   true,
        },
        importPath: b.importPath,
        benchFunc:  f,
        benchTime:  b.benchTime,
        bstate:     b.bstate,
    }
    if partial {...}

    if b.chatty != nil {...}

    if sub.run1() {
        sub.run()
    }
    b.add(sub.result)
    return !sub.failed
}
```

sub → Runtime representation for the benchmark

Benchmark function

→ comes from main B

first iteration (SIMILAR TO WARM UP)

N iterations

Run runs individual benchmarks

Run1() is responsible for running the first iteration of the benchmark.

Run1() does the following:

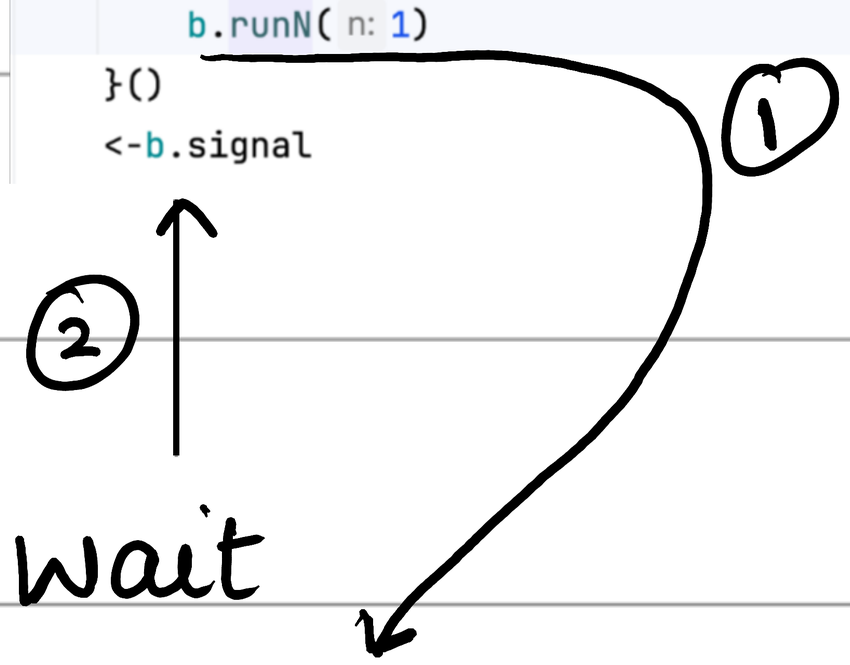
① Launches a goroutine to invoke runN(1) → which will run 1 iteration of the benchmark fn.

② wait for the goroutine to finish.

③ Determine if more iterations are needed → UNKNOWN

```go
func (b *B) run1() bool {
    if bstate := b.bstate; bstate != nil {
        // Extend maxLen, if needed.
        if n := len(b.name) + bstate.extLen + 1; n > bstate.maxLen {
            bstate.maxLen = n + 8 // Add additional slack to avoid too many jumps in size.
        }
    }
    go func() {
        // Signal that we're done whether we return normally
        // or by FailNow's runtime.Goexit.
        defer func() {
            b.signal <- true
        }()

        b.runN( n: 1)
    }()
    <-b.signal
```



run a single iteration of the benchmark.

```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
```

runN runs

n iterations

(currently 1)

of the benchmark