After the first iteration is done for a benchmark, , the remaining iterations are executed in run() method.

called from Run()

```
func (b *B) run() {
    labelsOnce.Do(func() {
        fmt.Fprintf(b.w, format: "goos: %s\n", runtime.GOOS)
        fmt.Fprintf(b.w, format: "goarch: %s\n", runtime.GOARCH)
        if b.importPath != "" {
            fmt.Fprintf(b.w, format: "pkg: %s\n", b.importPath)
        }
        if cpu := sysinfo.CPUName(); cpu != "" {
            fmt.Fprintf(b.w, format: "cpu: %s\n", cpu)
        }
    })
    if b.bstate != nil {
        // Running go test --test.bench
        b.bstate.processBench(b) // Must call doBench.
    } else {
        // Running func Benchmark.
        b.doBench()
    }
}
```

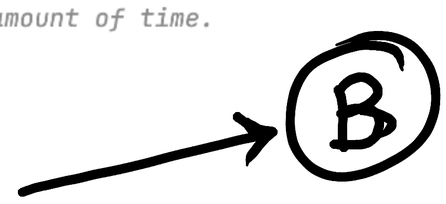
Let us consider bstate is nil, it will call `doBench()`.

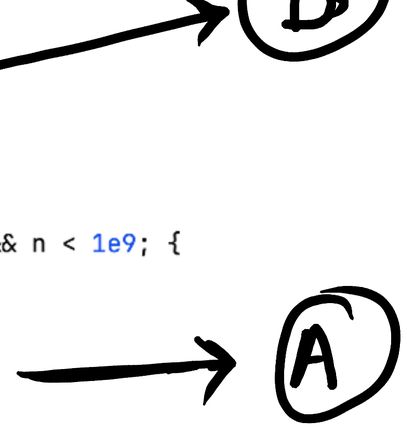
doBench() creates a goroutine, which calls launch() to perform iterations. doBench() waits for all iterations to complete.

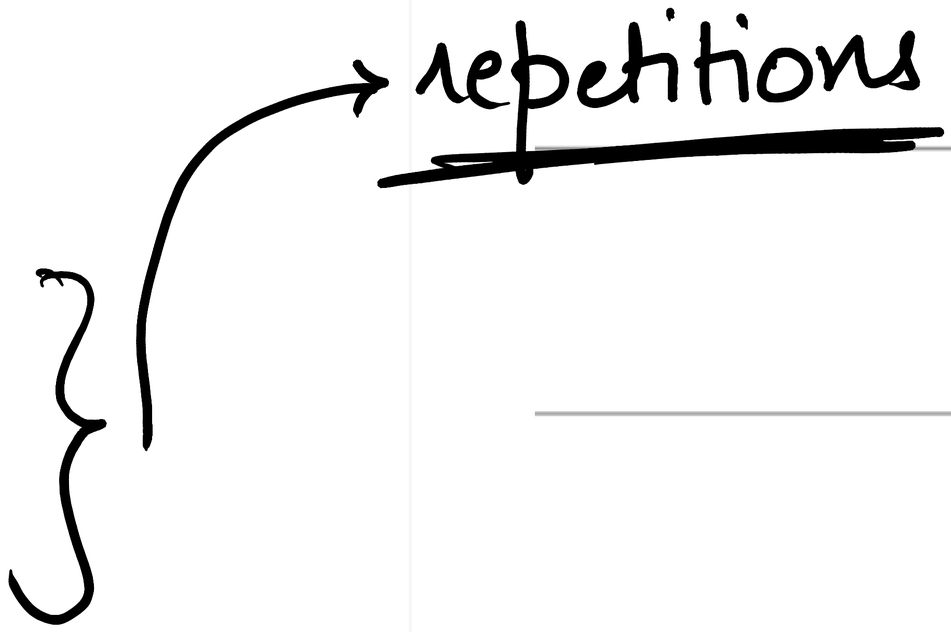
```
func (b *B) doBench() BenchmarkResult {
    go b.launch()
    <-b.signal
    return b.result
}
```

launch() is invoked in a goroutine to perform N iterations for a benchmark.

```go
func (b *B) launch() {
    /.../
    defer func() {
        b.signal <- true
    }()
    /.../
    if b.loop.n == 0 {
        // Run the benchmark for at least the specified amount of time.
        if b.benchTime.n > 0 {
            /.../
            if b.benchTime.n > 1 {
                b.runN(b.benchTime.n)
            }
        } else {
            d := b.benchTime.d
            for n := int64(1); !b.failed && b.duration < d && n < 1e9; {
                last := n
                // Predict required iterations.
                goalns := d.Nanoseconds()
                prevIters := int64(b.N)
                n = int64(predictN(goalns, prevIters, b.duration.Nanoseconds(), last))
                b.runN(int(n))
            }
        }
    }
    b.result = BenchmarkResult{N: b.N, T: b.duration, Bytes: b.bytes, MemAllocs: b.netAllocs, MemBytes: b.netBytes, Extra: b.extra}
}
```







Assume $b.loop.n = 0$, also $b.benchTime.n = 0$. We will consider (A) first.

benchmark duration, default = 1sec.

```
d := b.benchTime.d
for n := int64(1); !b.failed && b.duration < d && n < 1e9; {
    last := n
    // Predict required iterations.
    goalns := d.Nanoseconds()
    prevIters := int64(b.N)
    n = int64(predictN(goalns, prevIters, b.duration.Nanoseconds(), last))
    b.runN(int(n))
}
```

Under A, we have:

① Repeating till either n crosses $1e^9$ or duration goes beyond d.

② Predicting iterations

③ Running iterations

④ Collecting results after repetitions are done.

Open: ?

Why is BenchmarkResult created after all repetitions are done for a benchmark?

We don't want intermediate results

Under B, " f b.benchTime.n > 1 ",

run N iterations defined by

b.benchTime.n.

```
if b.benchTime.n > 1 {
    b.runN(b.benchTime.n)
}
```

Why > 1 ?

Because 1st iteration is already done...

Open:

processBench

Quick note:

Under A, we are doing R repetitions and each repetition we are doing some i iterations where $i > 1$.