# Reduce, Reuse, Recycle

## strategies for minimizing garbage

Jesse Allen
@jessecarl
Software Engineer
ASAPP (www.asapp.com)

## When GC Matters

## Performance doesn’t matter

## Until it does.

## Measure it

## Know your requirements

## Find the bottleneck

## Stack is faster than heap

## Nothing is faster than stack

## Reduce

## Escape analysis

```
$ go build -gcflags="-m" ./cmd/citybike-trip-etl
…
cmd/citybike-trip-etl/main.go:91:17: leaking closure reference f
cmd/citybike-trip-etl/main.go:93:32: name escapes to heap
cmd/citybike-trip-etl/main.go:90:30: leaking param: name
cmd/citybike-trip-etl/main.go:97:30: rc escapes to heap
cmd/citybike-trip-etl/main.go:101:14: leaking closure reference loc
cmd/citybike-trip-etl/main.go:103:38: d escapes to heap
cmd/citybike-trip-etl/main.go:33:13: main ... argument does not escape
…
cmd/citybike-trip-etl/main.go:36:13: main ... argument does not escape
cmd/citybike-trip-etl/main.go:74:22: main []trip.Sink literal does not escape
…
```

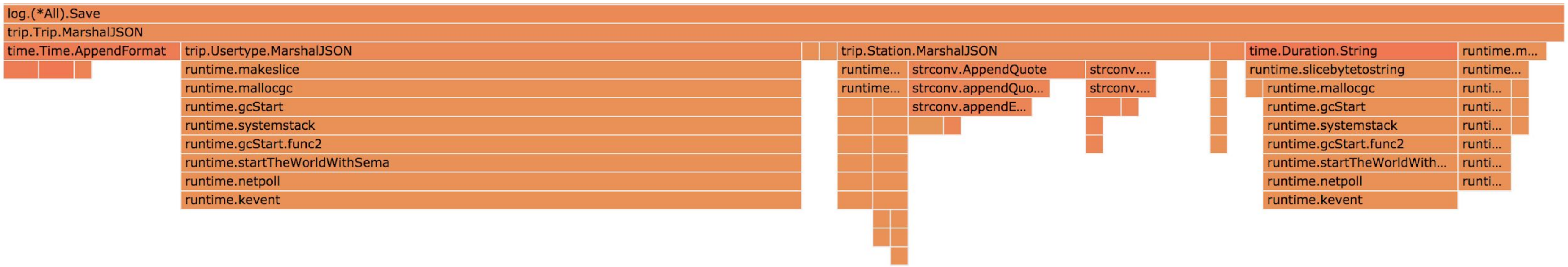
## Values

```go
// Using Pointers is likely to go heap
func (a *All) Save(t *trip.Trip) error {
  b, err := t.MarshalJSON()
  if err != nil {
    return err
  }
  b = append(b, '\n')
  _, err = a.Writer.Write(b)
  return err
}
```

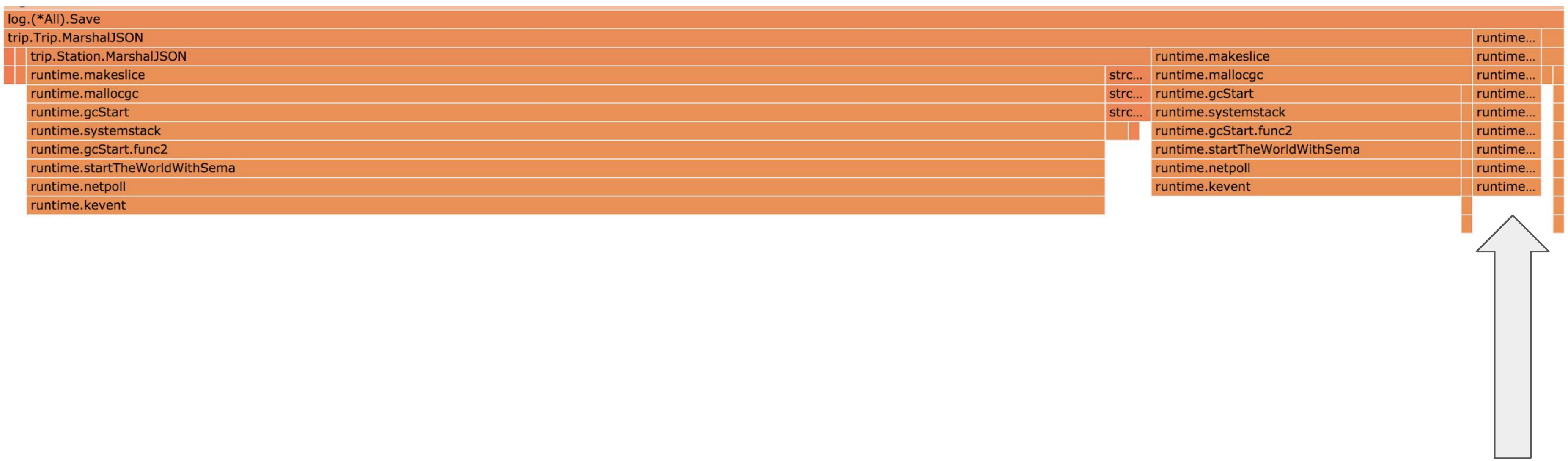
```go
// Using values likely to go to stack
func (a *All) Save(t trip.Trip) error {
  b, err := t.MarshalJSON()
  if err != nil {
    return err
  }
  b = append(b, '\n')
  a.Writer.Write(b)
  return nil
}
```

## Byte slices over strings

```go
// Strings are just more garbage
func (a *All) Save(t trip.Trip) error {
  b, err := t.MarshalJSON()
  if err != nil {
    return err
  }
  s := string(b) + "\n"
  _, err = a.Writer.Write([]byte(s))
  return err
}
```


log.(*All).Save
trip.Trip.MarshalJSON
time.Time.AppendFormat
trip.Usertype.MarshalJSON
trip.Station.MarshalJSON
time.Duration.String
runtime.m...
runtime.makeslice
runtime...
strconv.AppendQuote
strconv....
runtime.slicebytetostring
runtime...
runtime.mallocgc
runtime...
strconv.appendQuo...
strconv....
runtime.mallocgc
runti...
runtime.gcStart
strconv.appendE...
runtime.gcStart
runti...
runtime.systemstack
runtime.systemstack
runti...
runtime.gcStart.func2
runtime.gcStart.func2
runti...
runtime.startTheWorldWithSema
runtime.startTheWorldWith...
runti...
runtime.netpoll
runtime.netpoll
runti...
runtime.kevent
runtime.kevent

Byte Slice

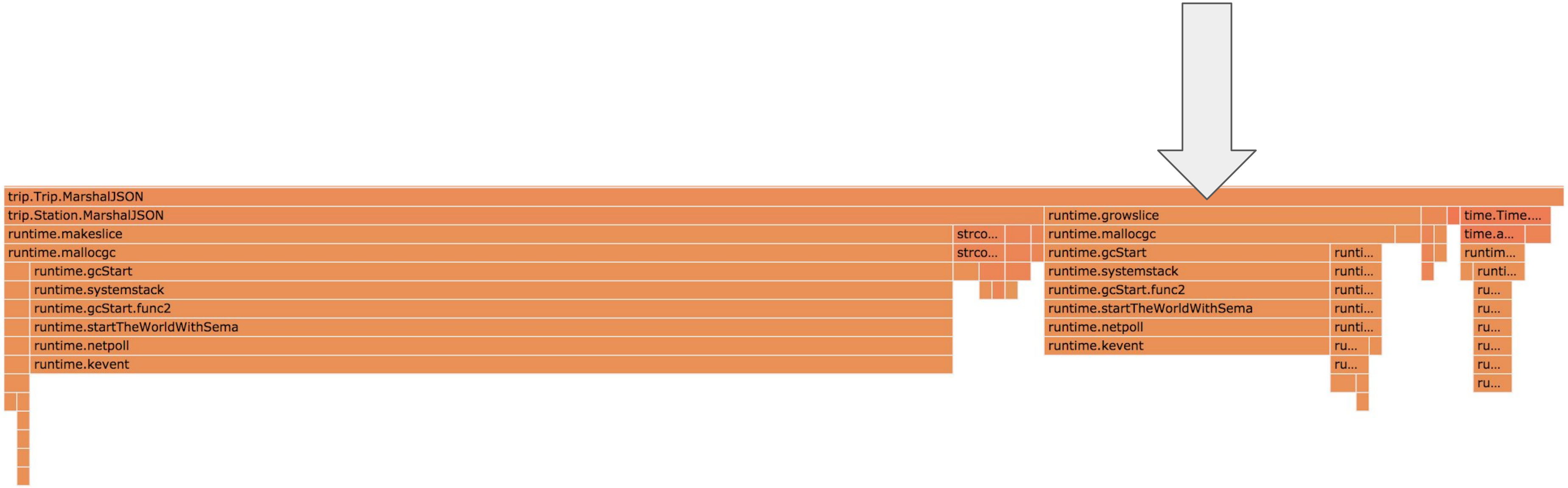

log.(*All).Save
trip.Trip.MarshalJSON
runtime...
trip.Station.MarshalJSON
runtime.makeslice
runtime.makeslice
strc...
runtime.mallocgc
runtime.mallocgc
runtime.gcStart
runtime.systemstack
runtime.gcStart.func2
runtime.startTheWorldWithSema
runtime.netpoll
runtime.kevent
runtime.gcStart
runtime.systemstack
runtime.gcStart.func2
runtime.startTheWorldWithSema
runtime.netpoll
runtime.kevent

String

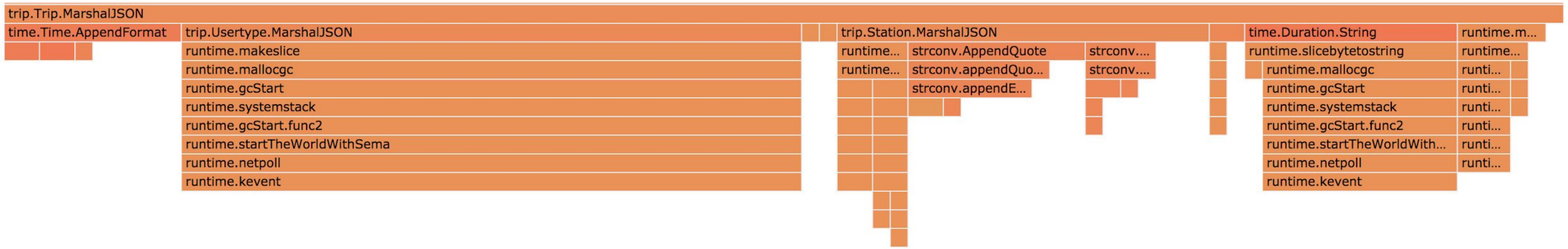
## Make with capacity

```go
func (t Trip) MarshalJSON() ([]byte, error) {
  var b []byte
  b = append(b, '{')
  b = append(b, []byte(`"trip_duration":"`)...)
  b = append(b,
[]byte(t.TripDuration.String())...)
  b = append(b, '"')
  …
  b = append(b, '}')
  return b, nil
}
```

```go
func (t Trip) MarshalJSON() ([]byte, error) {
  b := make([]byte, 0, 512)
  b = append(b, '{')
  b = append(b, []byte(`"trip_duration":"`)...)
  b = append(b,
[]byte(t.TripDuration.String())...)
  b = append(b, '"')
  …
  b = append(b, '}')
  return b, nil
}
```



Before Making Space



## After Making Space

## Reuse

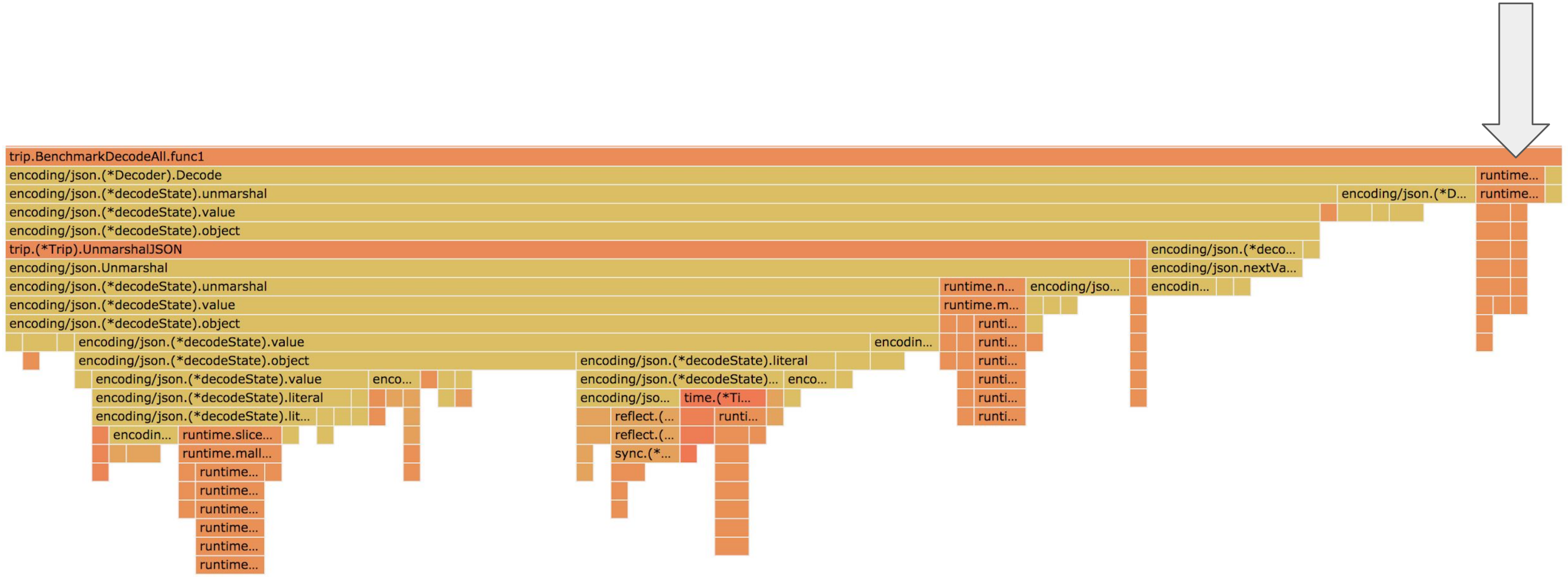
## More Byte Slices

```go
packet := make([]byte, 0, mtuSize)
chunk := make([]byte, maxChunkSize)
for i := 0; i < count; i++ {
    chunkSize, err := reader.Read(chunk)
    if err != nil && err != io.EOF {
        return 0, err
    }
    packet = append(packet, uint8(i), uint8(count)) // sequence
    packet = append(packet, chunk[:chunkSize]...)
    if _, err := gl.conn.WriteTo(packet, gl.addr); err != nil {
        return 0, err
    }
    packet, chunk = packet[:0], chunk[:maxChunkSize]
}
```

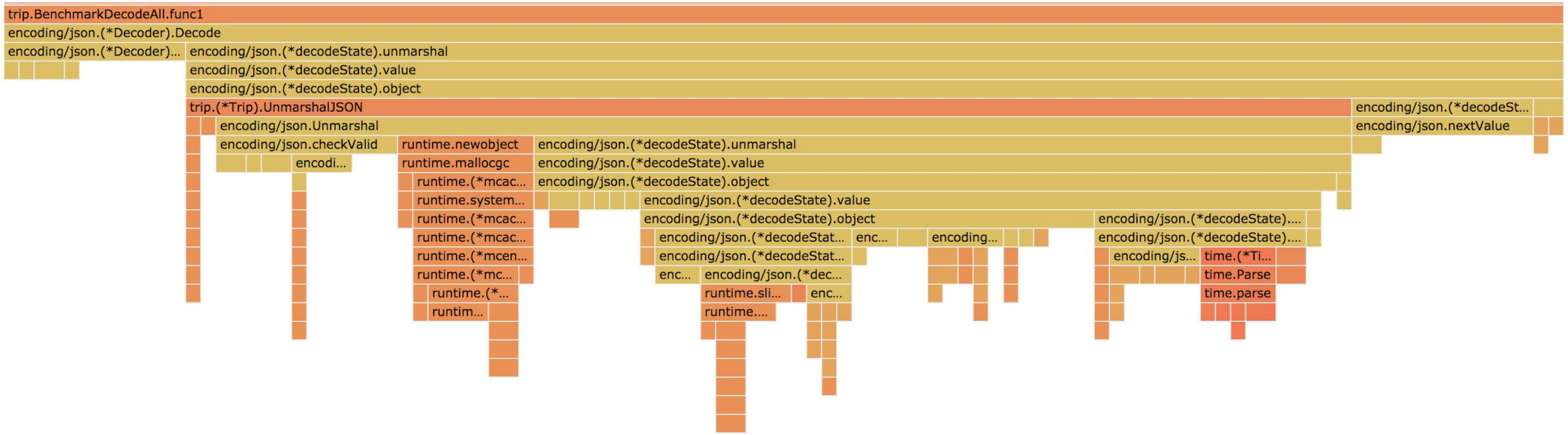
## Structs

```go
dec := json.NewDecoder(bytes.NewReader(blob))
saver := nopSaver{}
for dec.More() {
    var trip Trip
    err := dec.Decode(&trip)
    if err != nil {
        return err
    }
    saver.Save(trip)
}
```

```go
dec := json.NewDecoder(bytes.NewReader(blob))
saver := nopSaver{}
var trip Trip
for dec.More() {
    err := dec.Decode(&trip)
    if err != nil {
        b.Fatal(err)
    }
    saver.Save(trip)
    trip = Trip{}
}
```



No Object Reuse



## Some Object Reuse

## Caution with Concurrency

```go
packet := make([]byte, 0, mtuSize)
chunk := make([]byte, maxChunkSize)
for i := 0; i < count; i++ {
    chunkSize, err := reader.Read(chunk)
    if err != nil && err != io.EOF {
        return 0, err
    }
    packet = append(packet, uint8(i), uint8(count)) // sequence
    packet = append(packet, chunk[:chunkSize]...)
    if _, err := gl.conn.WriteTo(packet, gl.addr); err != nil {
        return 0, err
    }
    packet, chunk = packet[:0], chunk[:maxChunkSize]
}
```

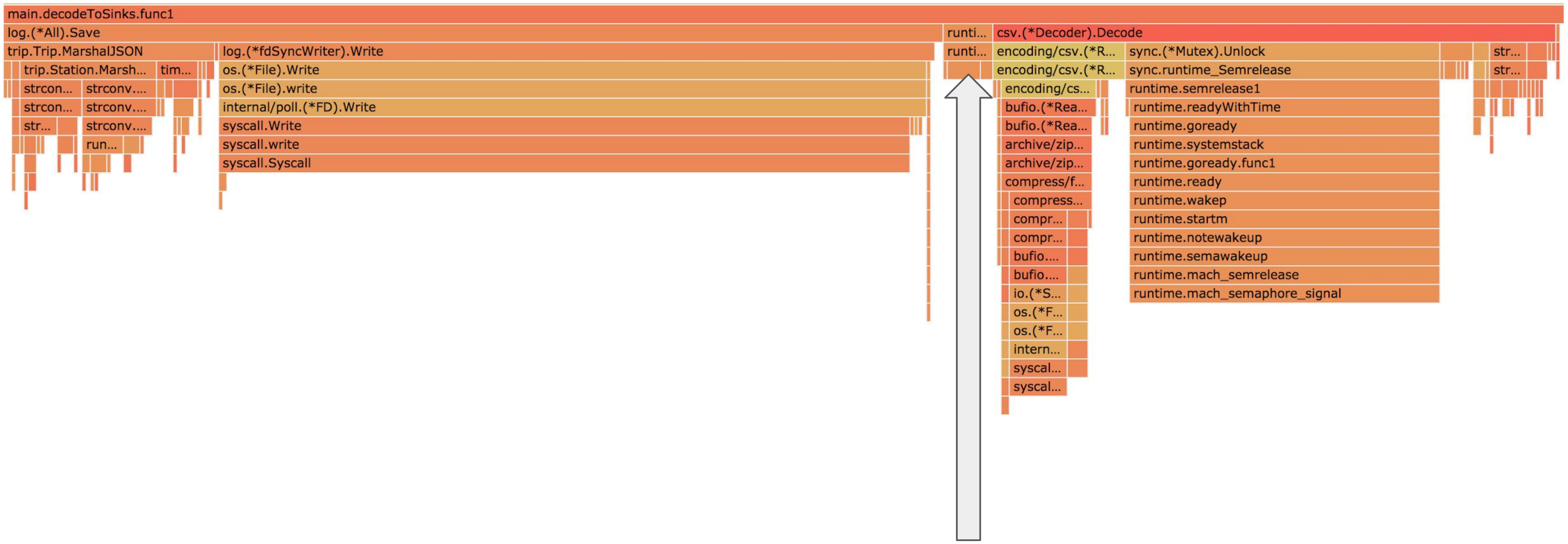
```go
packet := make([]byte, 0, mtuSize)
chunk := make([]byte, maxChunkSize)
for i := 0; i < count; i++ {
    chunkSize, err := reader.Read(chunk)
    if err != nil && err != io.EOF {
        return 0, err
    }
    packet = append(packet, uint8(i), uint8(count)) // sequence
    packet = append(packet, chunk[:chunkSize]...)
    if _, err := gl.conn.WriteTo(packet, gl.addr); err != nil {
        return 0, err
    }
    packet, chunk = packet[:0], chunk[:maxChunkSize]
}
```

```go
func (w *Writer) Write(b []byte) (int, error) {
    c := make([]byte, len(b))
    n := copy(c, b)
    go w.NextThing(c)
    return n, nil
}
```

## Recycle

## Free Lists

```go
sem := make(chan struct{}, 64)
for dec.More() {
    sem <- struct{}{}
    go func() {
        defer func() { <-sem }()
        var t trip.Trip
        d.Decode(&t)
        s.Save(t)
    }()
}
```

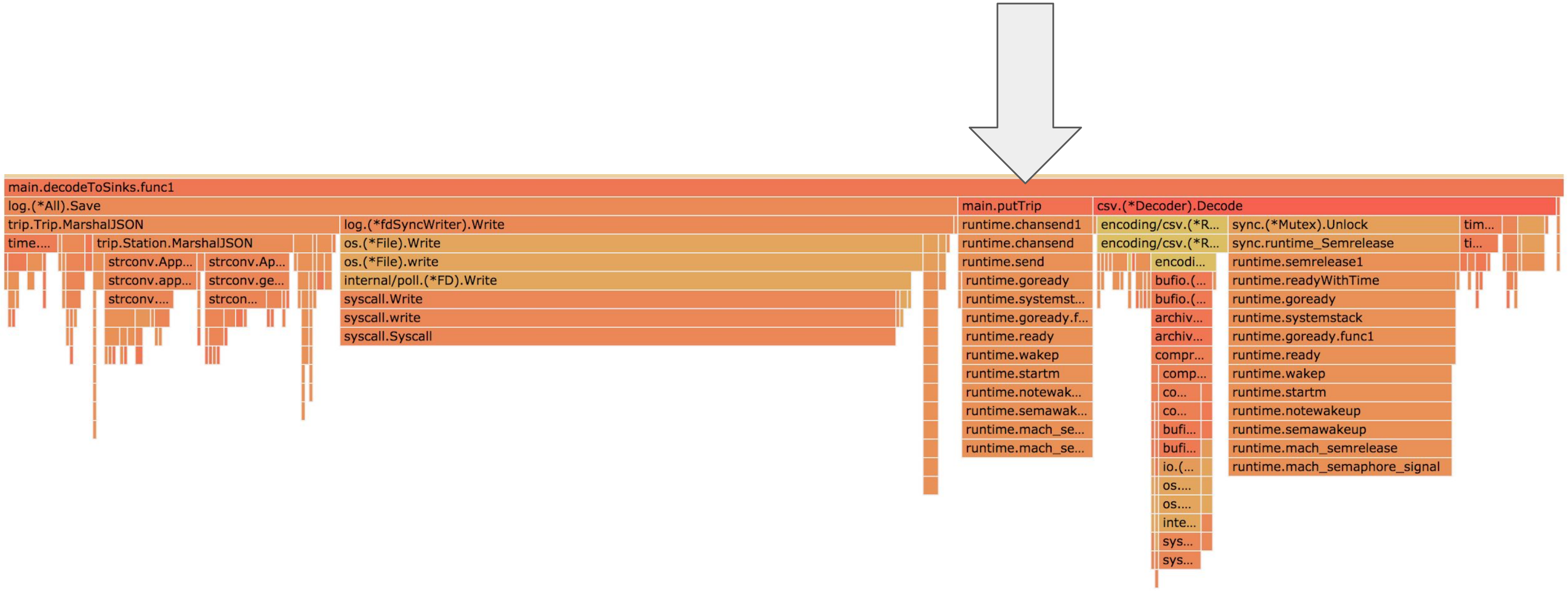


No Recycling

```go
sem := make(chan struct{}, 64)
for dec.More() {
    sem <- struct{}{}
    go func() {
        defer func() { <-sem }()
        t := getTrip()
        defer putTrip(t)
        d.Decode(t)
        s.Save(*t)

    }()
}
```

```go
var tripList = make(chan *trip.Trip, 16)
func init() {
  for {
    select {
    case tripList <- new(trip.Trip):
    default:
      return
    }
  }
}
func getTrip() *trip.Trip { return <-tripList }
func putTrip(t *trip.Trip) {
  *t = trip.Trip{}
  tripList <- t
}
```

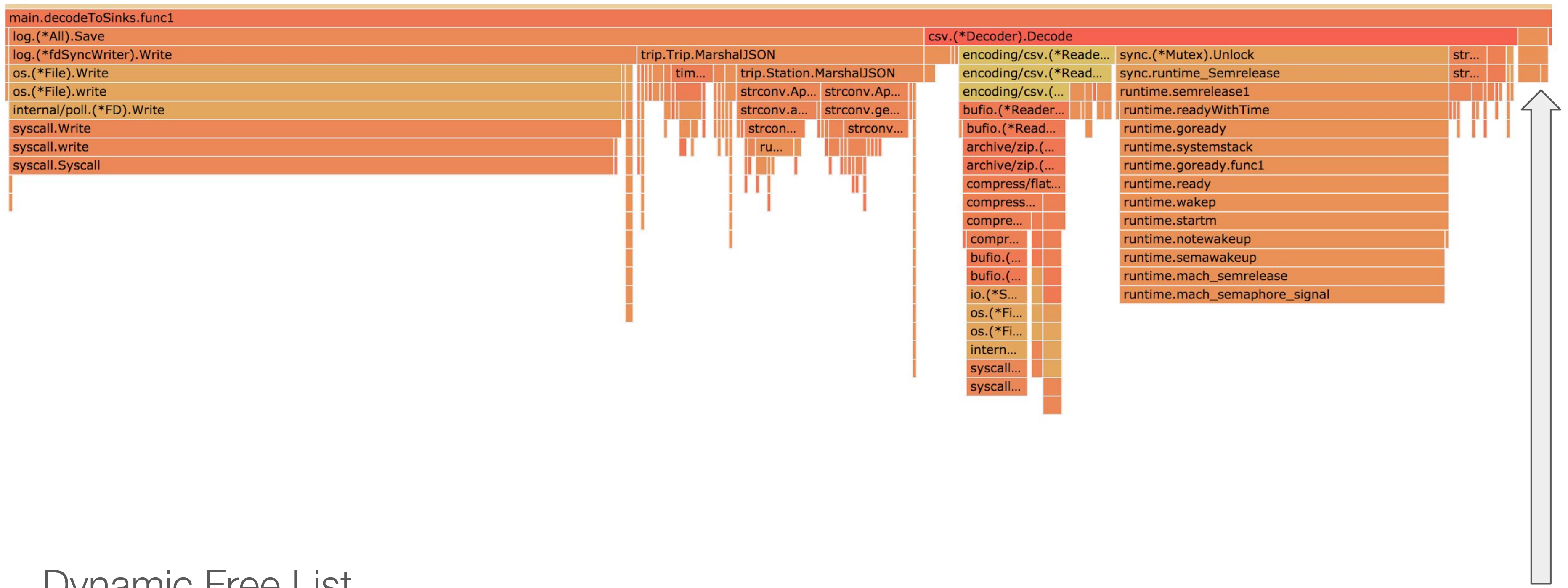

main.decodeToSinks.func1
log.(*All).Save
main.putTrip
csv.(*Decoder).Decode
trip.Trip.MarshalJSON
log.(*fdSyncWriter).Write
runtime.chansend1
encoding/csv.(*R...
sync.(*Mutex).Unlock
tim...
time....
trip.Station.MarshalJSON
os.(*File).Write
runtime.chansend
encoding/csv.(*R...
sync.runtime_Semrelease
ti...
strconv.App...
strconv.Ap...
os.(*File).write
runtime.send
encodi...
runtime.semrelease1
strconv.app...
strconv.ge...
internal/poll.(*FD).Write
runtime.goready
bufio.(...
runtime.readyWithTime
strconv....
strcon...
syscall.Write
runtime.systemst...
bufio.(...
runtime.goready
syscall.write
runtime.goready.f...
archiv...
runtime.systemstack
syscall.Syscall
runtime.ready
archiv...
runtime.goready.func1
runtime.wakep
compr...
runtime.ready
runtime.startm
comp...
runtime.wakep
runtime.notewak...
co...
runtime.startm
runtime.semawak...
co...
runtime.notewakeup
runtime.mach_se...
bufi...
runtime.semawakeup
runtime.mach_se...
bufi...
runtime.mach_semrelease
io.(...
runtime.mach_semaphore_signal
os....
os....
inte...
sys...
sys...

Fixed Pool

```go
sem := make(chan struct{}, 64)
for dec.More() {
    sem <- struct{}{}
    go func() {
        defer func() { <-sem }()
        t := getTrip()
        defer putTrip(t)
        d.Decode(t)
        s.Save(*t)

    }()
}
```

```go
var tripList = make(chan *trip.Trip, 16)
func getTrip() *trip.Trip {
  select {
  case t := <-tripList:
    return t
  default:
    return new(trip.Trip)
  }
}
func putTrip(t *trip.Trip) {
  *t = trip.Trip{}
  select {
  case tripList <- t:
  default:
  }
}
```



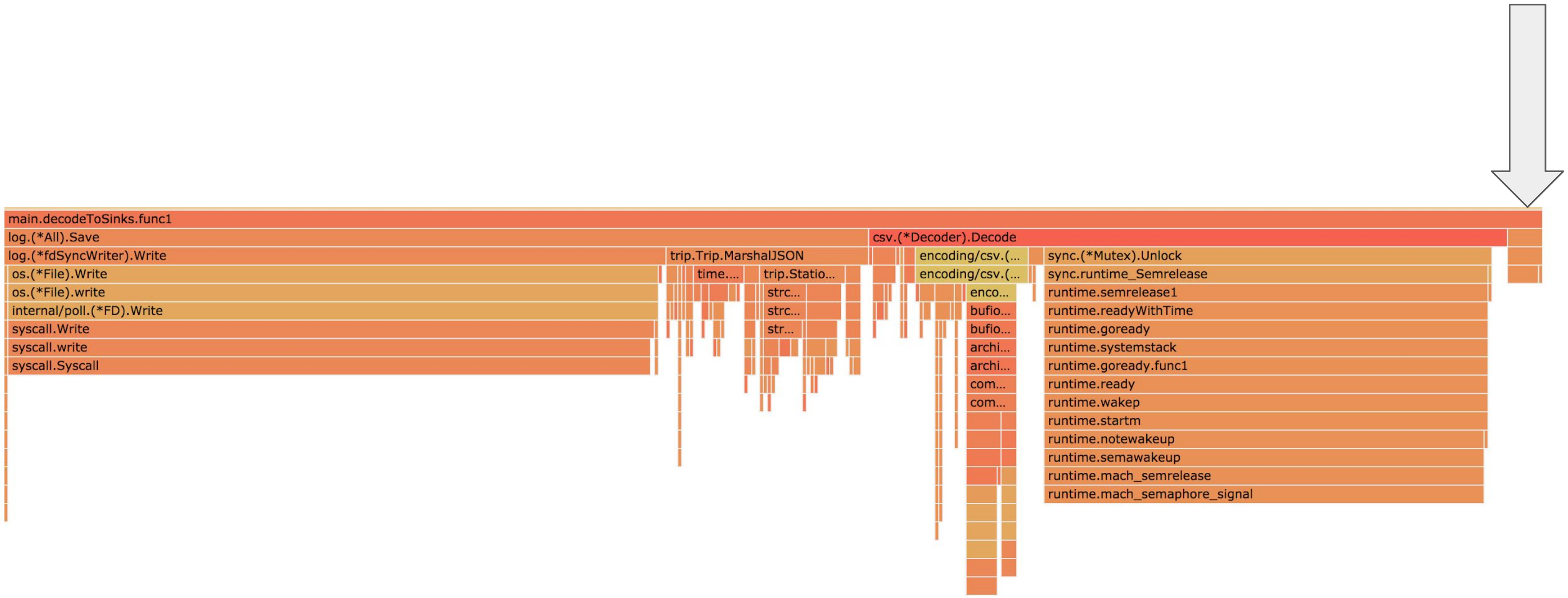
## Dynamic Free List

## Pools

```go
sem := make(chan struct{}, 64)
for dec.More() {
    sem <- struct{}{}
    go func() {
        defer func() { <-sem }()
        t := getTrip()
        defer putTrip(t)
        d.Decode(t)
        s.Save(*t)

    }()
}
```

```go
var tripPool = sync.Pool{
    New: func() interface{} {
        return new(trip.Trip)
    },
}
func getTrip() *trip.Trip {
    return tripPool.Get().(*trip.Trip)
}
func putTrip(t *trip.Trip) {
    *t = trip.Trip{}
    tripPool.Put(t)
}
```


main.decodeToSinks.func1
log.(*All).Save
csv.(*Decoder).Decode
log.(*fdSyncWriter).Write
trip.Trip.MarshalJSON
encoding/csv.(...
sync.(*Mutex).Unlock
os.(*File).Write
time....
trip.Statio...
encoding/csv.(...
sync.runtime_Semrelease
os.(*File).write
strc...
enco...
runtime.semrelease1
internal/poll.(*FD).Write
strc...
bufio...
runtime.readyWithTime
syscall.Write
str...
bufio...
runtime.goready
syscall.write
archi...
runtime.systemstack
syscall.Syscall
archi...
runtime.goready.func1
com...
runtime.ready
com...
runtime.wakep
runtime.startm
runtime.notewakeup
runtime.semawakeup
runtime.mach_semrelease
runtime.mach_semaphore_signal

sync.Pool

# Thank You