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
…
```

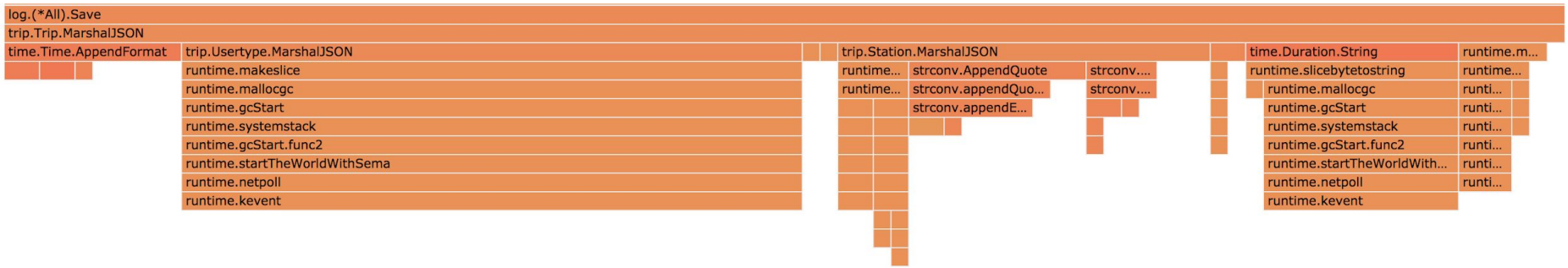
Values

```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
}
```

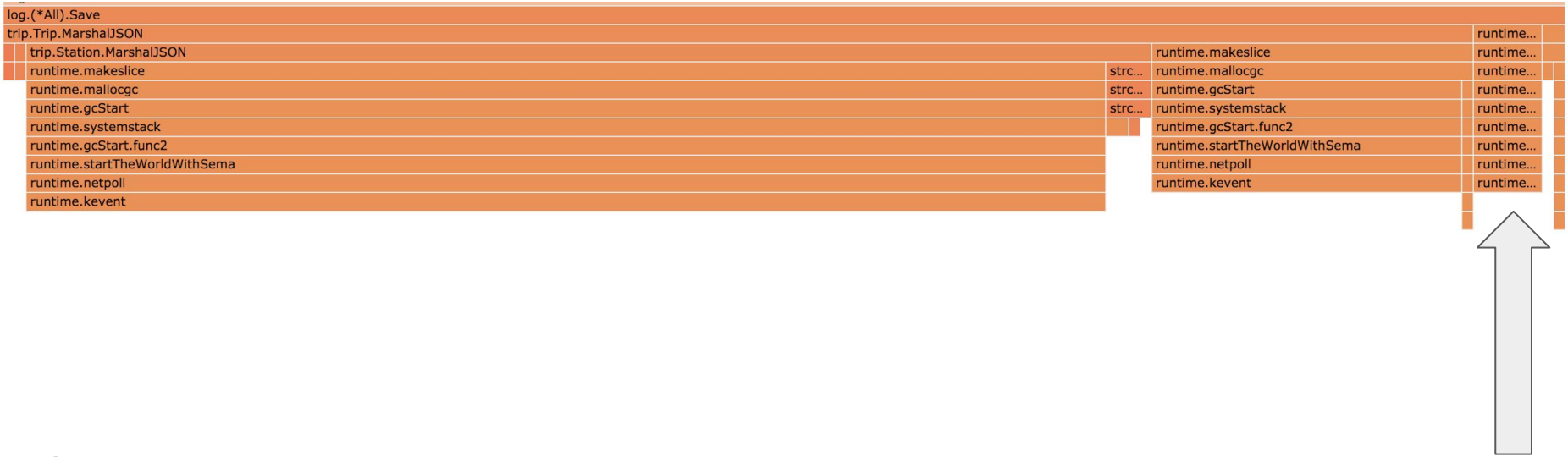
```go
// Using values likely to go to stack
func (a *All) Save(t trip.Trip) error {
  b, err := t.MarshalJSON()
  if err != nil {
    return err
  }
  b = append(b, '\n')
  a.Writer.Write(b)
  return nil
}
```

Byte slices over strings

```
// 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
runtime.makeslice
runtime.mallocgc
runtime.gcStart
runtime.systemstack
runtime.gcStart.func2
runtime.startTheWorldWithSema
runtime.netpoll
runtime.kevent
trip.Station.MarshalJSON
runtime...
runtime...
strconv.AppendQuote
strconv.appendQuo...
strconv.appendE...
strconv....
strconv....
time.Duration.String
runtime.slicebytetostring
runtime.mallocgc
runtime.gcStart
runtime.systemstack
runtime.gcStart.func2
runtime.startTheWorldWith...
runtime.netpoll
runtime.kevent
runtime.m...
runtime...
runti...
runti...
runti...
runti...
runti...

Byte Slice

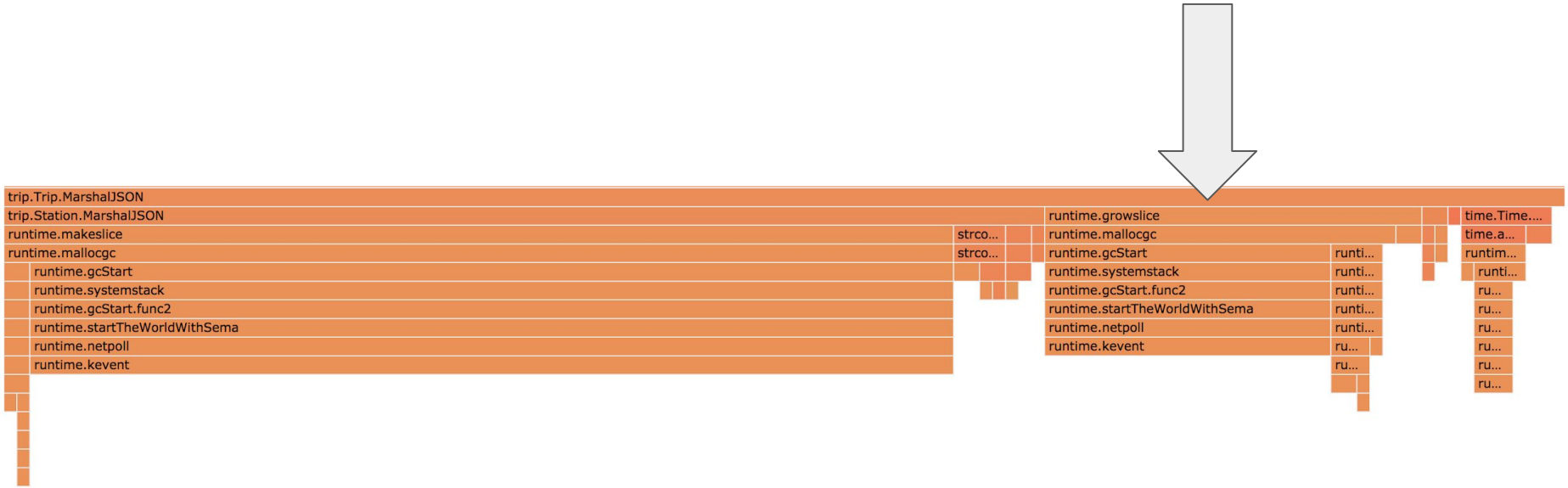

log.(*All).Save
trip.Trip.MarshalJSON
trip.Station.MarshalJSON
runtime.makeslice
runtime.mallocgc
runtime.gcStart
runtime.systemstack
runtime.gcStart.func2
runtime.startTheWorldWithSema
runtime.netpoll
runtime.kevent
strc...
runtime...

String

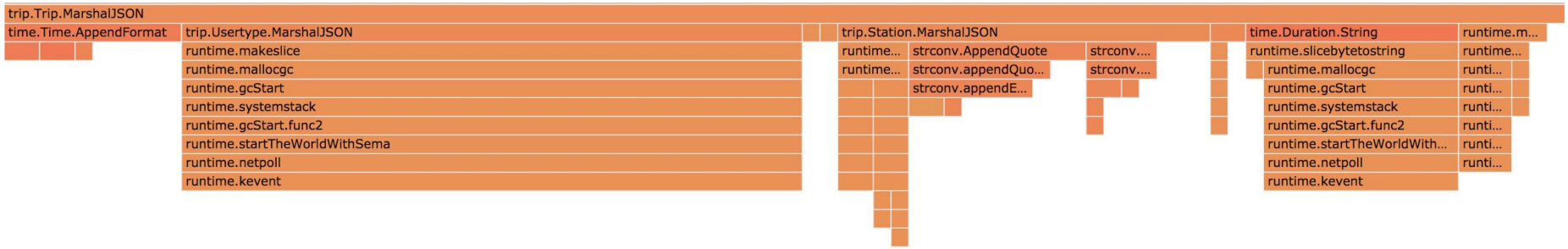
Make with capacity

```
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
}
```

```
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

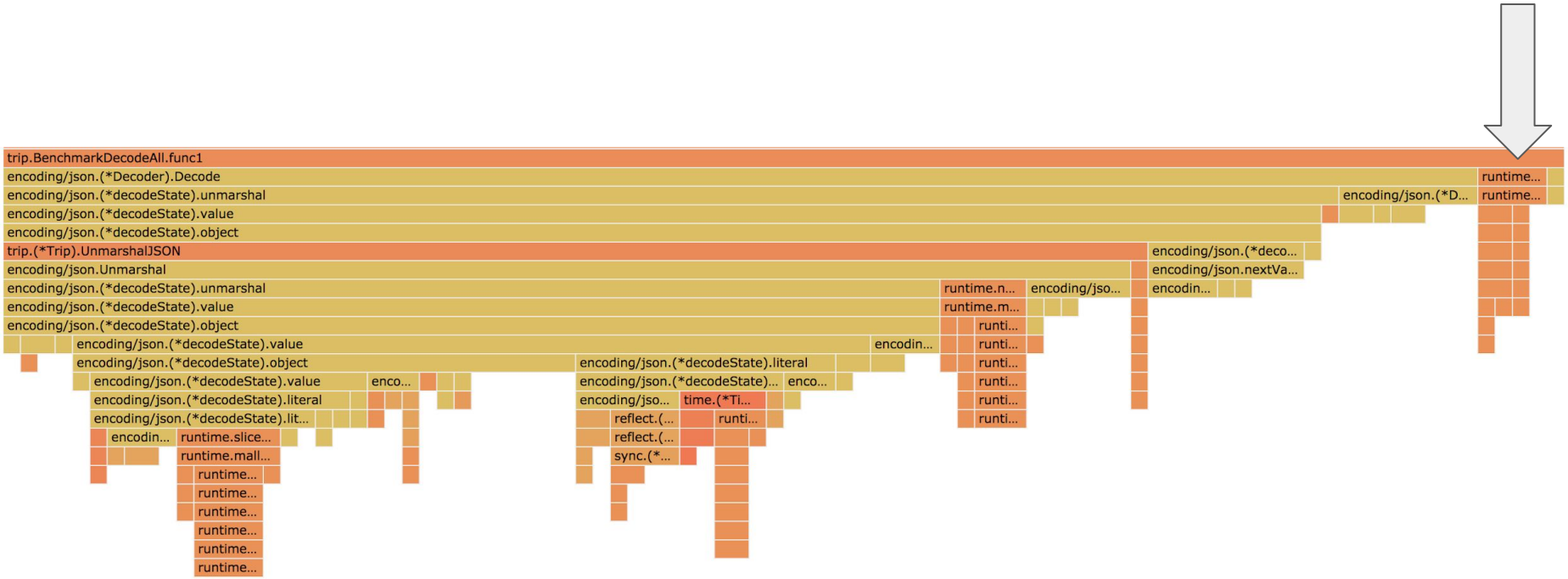
More Byte Slices

```
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]
}
```

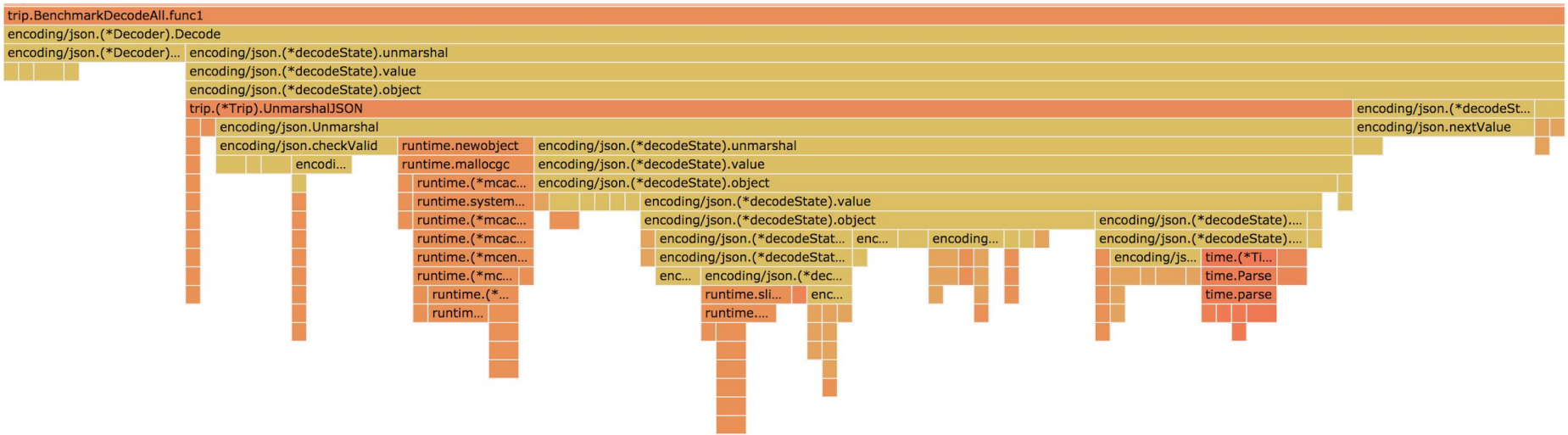
Structs

```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

```
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]
}
```

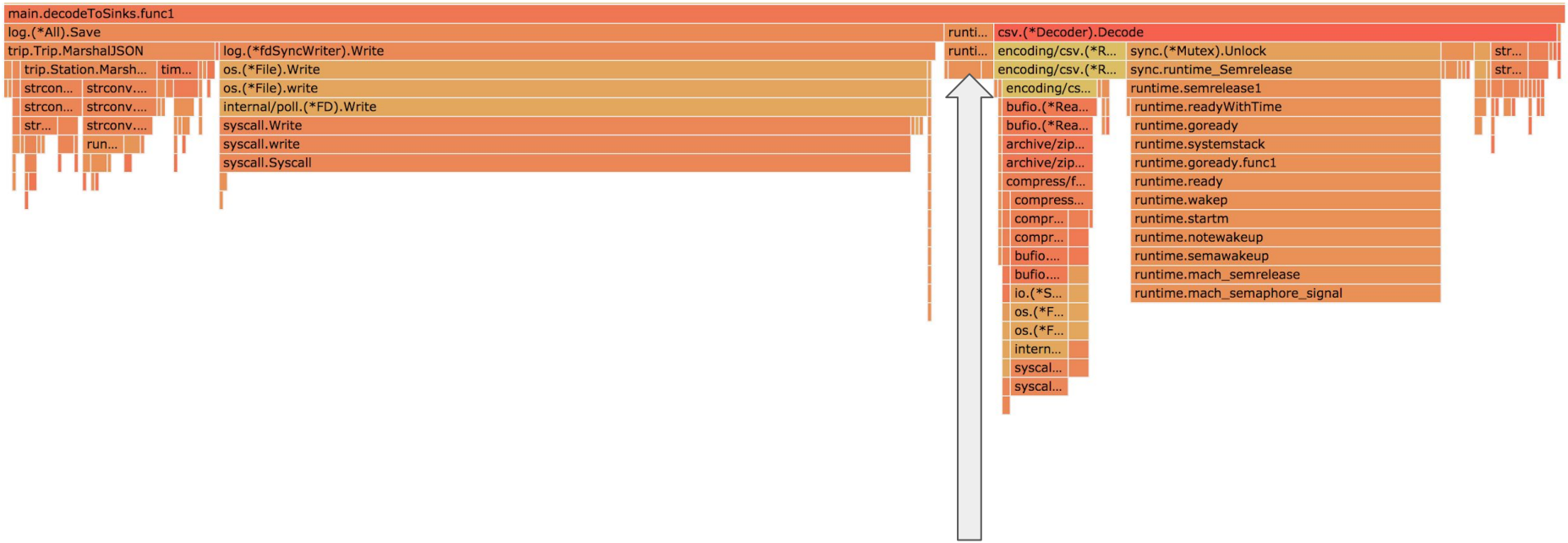
```
packet := make([]byte, 0, mtuSize)
chunk := make([]byte, maxChunkSize)
for i := 0; i < count; i++ {
    chunkSize, err := reader.Read(chunk)
    if err != nil && err != io.EOF {
        return 0, err
    }
    packet = append(packet, uint8(i), uint8(count)) // sequence
    packet = append(packet,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

```
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)
    }()
}
```

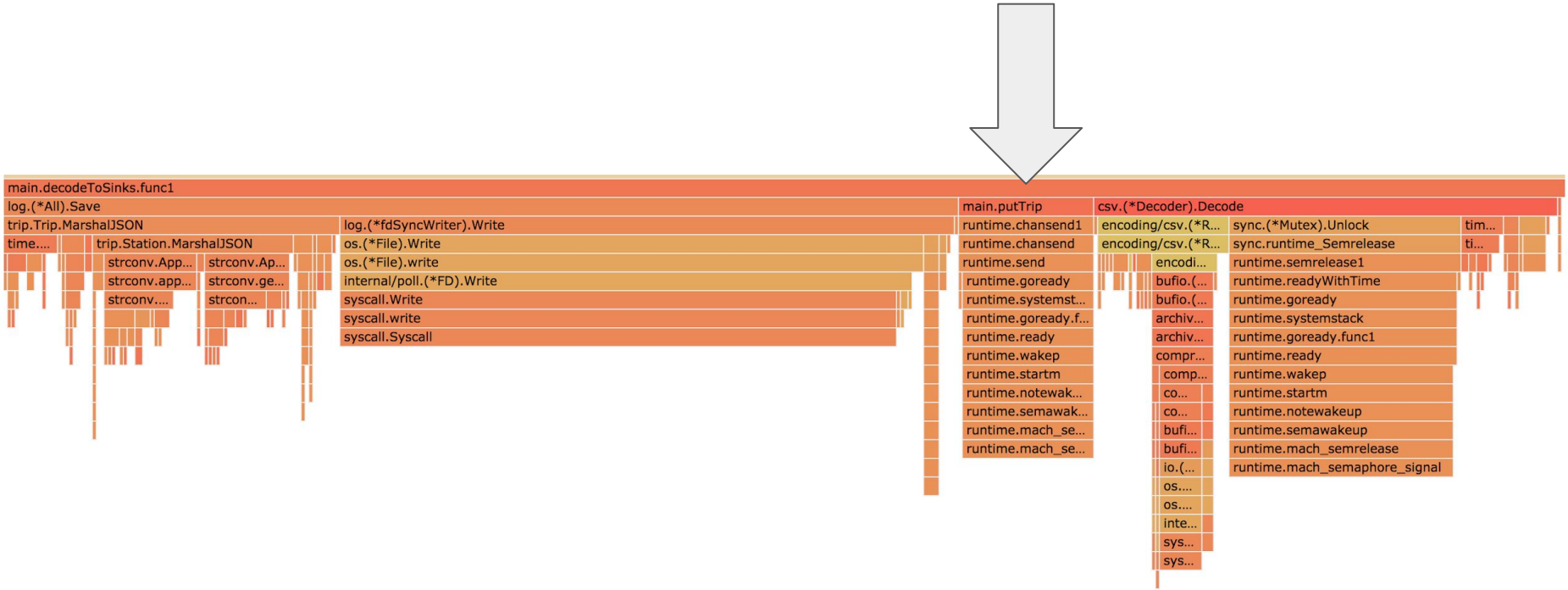


No Recycling

```
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)

    }()
}
```

```
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
}
```

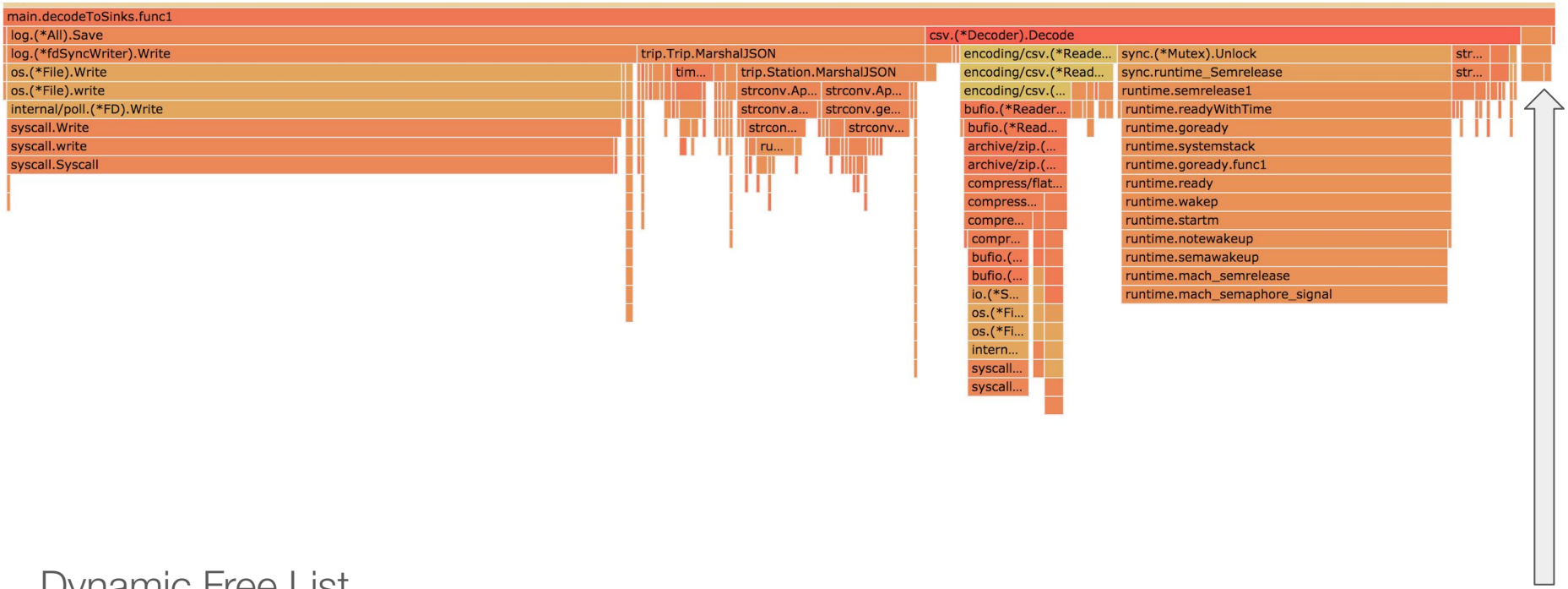

main.decodeToSinks.func1
log.(*All).Save
trip.Trip.MarshalJSON
time....
trip.Station.MarshalJSON
strconv.App...
strconv.app...
strconv....
strconv.Ap...
strconv.ge...
strcon...
log.(*fdSyncWriter).Write
os.(*File).Write
os.(*File).write
internal/poll.(*FD).Write
syscall.Write
syscall.write
syscall.Syscall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

Fixed Pool

```
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)

    }()
}
```

```
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:
  }
}
```



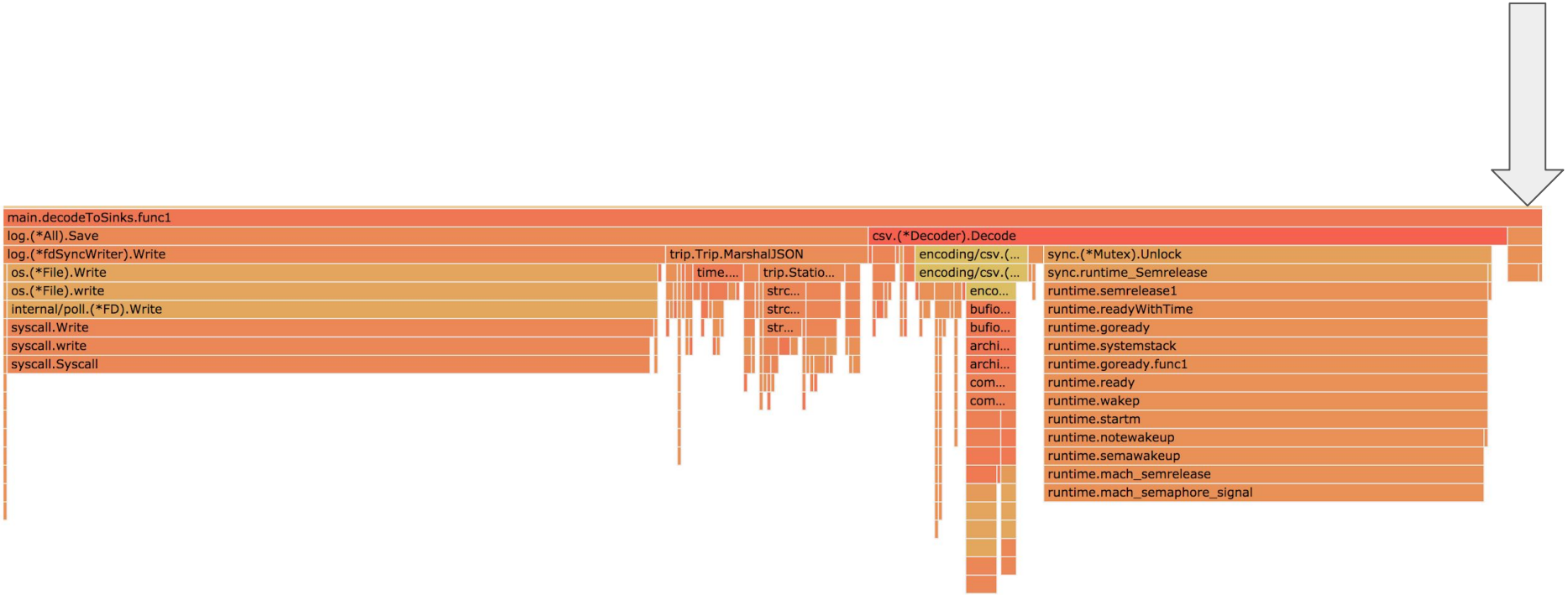
Dynamic Free List

Pools

```
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)

    }()
}
```

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

sync.Pool

Thank You