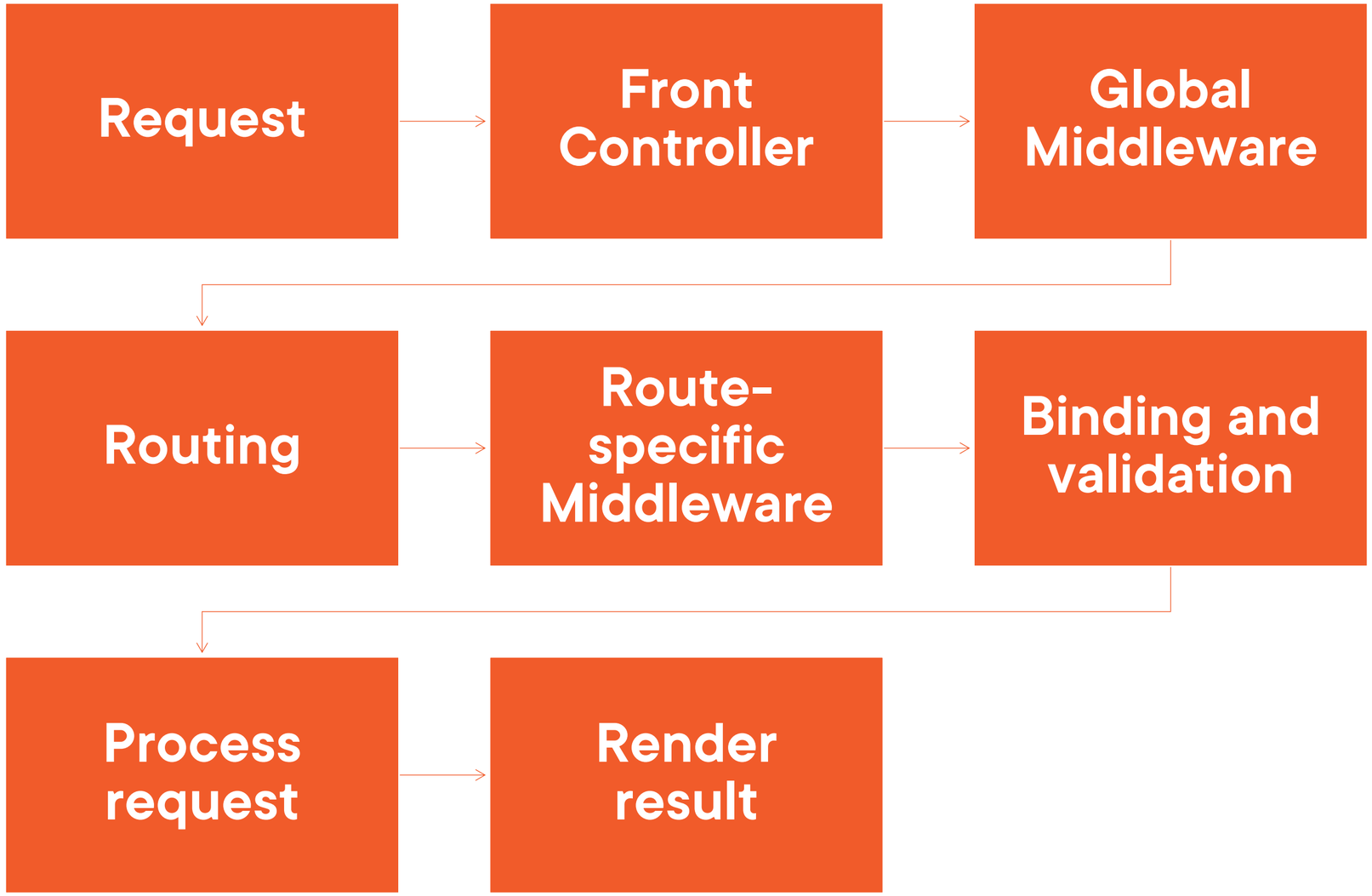
Generating Responses

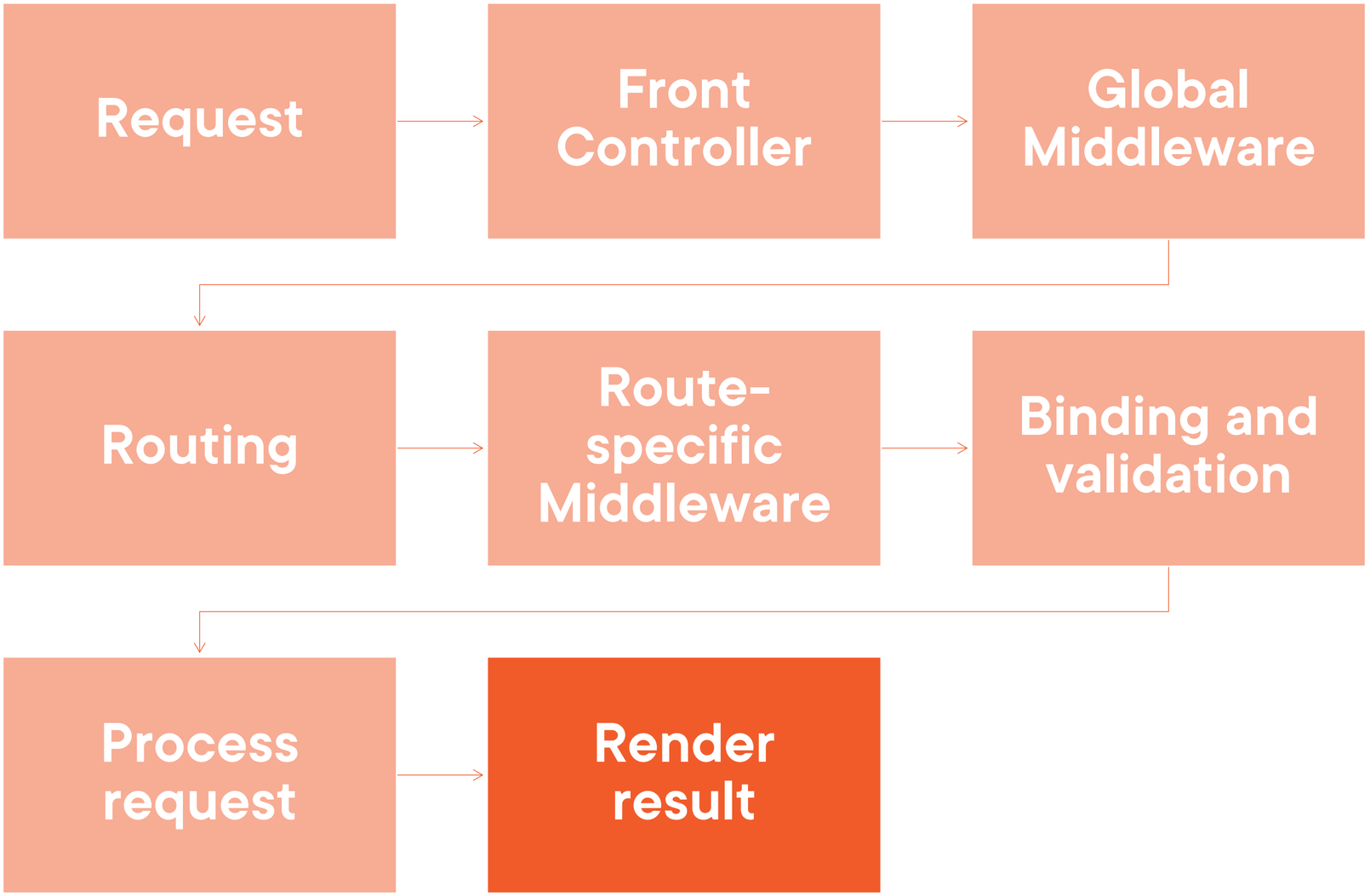
Michael Van Sickle

@vansimke

HTTP Request Pipeline



HTTP Request Pipeline



Overview

Serving unstructured data

Rendering HTML

Rendering other data types

Responding with File Data

File

FileFromFS

FileAttachment

```
var c *gin.Context
c.File(filepath string)

c.FileFromFS(filepath string,
     fs http.FileSystem)

c.FileAttachment(filepath,
     filename string)
```

- **Stream file contents from disk into response body**

- **Stream file content from http.FileSystem into response body**

- **Send file as attachment**

Responding with File Data

```
router.GET(“/reports”, func(c *gin.Context) {
  c.File(“/path/to/report.csv”)

  fs := gin.Dir(“./root/of/filesystem”, true)
  c.FileFromFS(“./reportfromfs.csv”, fs)

  c.FileAttachment(“/path/to/attachment.csv”, “nameonclient.csv”)
})
```

Responding with Arbitrary Data

Data

DataFromReader

Stream

```
var c *gin.Context
c.Data(code int,
    contentType string,
    data []byte)

c.DataFromReader(code int,
    contentLength int64,
    contentType string,
    reader io.Reader,
    extraHeaders map[string]string)

c.Stream(step func(w io.Writer) bool)
    bool
```

- **Stream data into response body**

- **Stream data from io.Reader into response body**

- **Stream response to io.Writer, returns true if client disconnected in middle of stream**

Responding with Arbitrary Data

```
var data = []byte(“Mary had a little lamb”)
var buffer = bytes.NewBuffer(data)

router.GET(“/reports”, func(c *gin.Context) {
  c.Data(http.StatusOK,
    “text/plain”,
    data)

  c.DataFromReader(http.StatusOK,
    int64(buffer.Len()),
    “text/plain”,
    buffer,
    map[string]string{
      “Content-Disposition”, `attachment; filename=“songlyrics.txt”`
    })
})
```

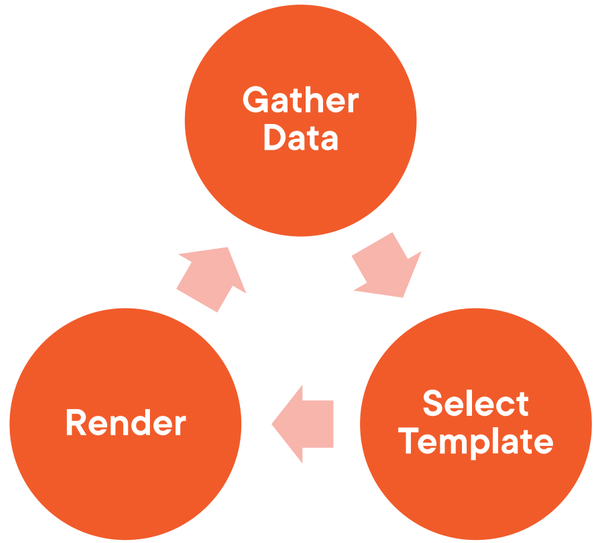
Responding with Arbitrary Data

```
func streamer(file string) func(io.Writer) bool {
  f, err := os.Open(file)
  // handle error case

  return func(step io.Writer) bool {
    for {
      data := make([]byte, 64 *2^10 /* 64 kB */)
      if _, err := f.Read(data); err == nil {
        _, err := step.Write(data)
        // handle error case
        return true // keep stream open
      } else {
        return false // close the stream
      }
    }
  }
}

router.GET(“/resource/to/stream”, func(c *gin.Context) {
   cancelled := c.Stream(streamer(pathtofile)) // true if client aborted stream
})
```

Rendering HTML

Load templates

```
gin.Engine.LoadHTMLFiles
gin.Engine.LoadHTMLGlob
gin.Engine.SetHTMLTemplate
```




```
gin.Context.HTML
```

Loading HTML Templates

```
import “html/template”

func main() {
  router := gin.Default()

  router.LoadHTMLFiles(“./templates/index.tmpl”, “./templates/users.tmpl”)

  router.LoadHTMLGlob(“./templates/*”)

  t := template.Must(
    template.ParseFiles(“./templates/index.tmpl”, “./templates.users.tmpl”,
  )

  router.SetHTMLTemplate(t)
}
```

Rendering HTML from Templates

```
var html = `
  <html>
  <head></head>
  <body>
    <h1>Hello, {{.name}}</h1>
  </body>
  </html>
`

func main() {
  router := gin.Default()
  template := template.New(“index.tmpl”)
  template.Parse(html)
  router.SetHTMLTemplate(template)

  router.GET(“/hello”, func(c *gin.Context) {
    c.HTML(http.StatusOK,
      “index.tmpl”,
      map[string]any{“name”:”Gophers”},
  })
}
```

```
var c *gin.Context

c.String(code int, format string,
    values ...any)
c.JSON(code int, obj any)
c.XML(code int, obj any)
c.YAML(code int, obj any)
c.ProtoBuf(code int, obj any)

c.SecureJSON(code int, obj any)
c.JSONP(code int, obj any)
c.AsciiJSON(code int, obj any)
c.PureJSON(code int, obj any)

c.Render(code int, r render.Render)
```

- **Render a formatted string**
- **Render JSON**
- **Render XML**
- **Render YAML**
- **Render ProtoBuf (obj must be a protobuf object!)**
- **Various flavors of JSON encoding are supported**
- **Generate response using provided Render object**

Summary

Serving unstructured data

Rendering HTML

Rendering other data types