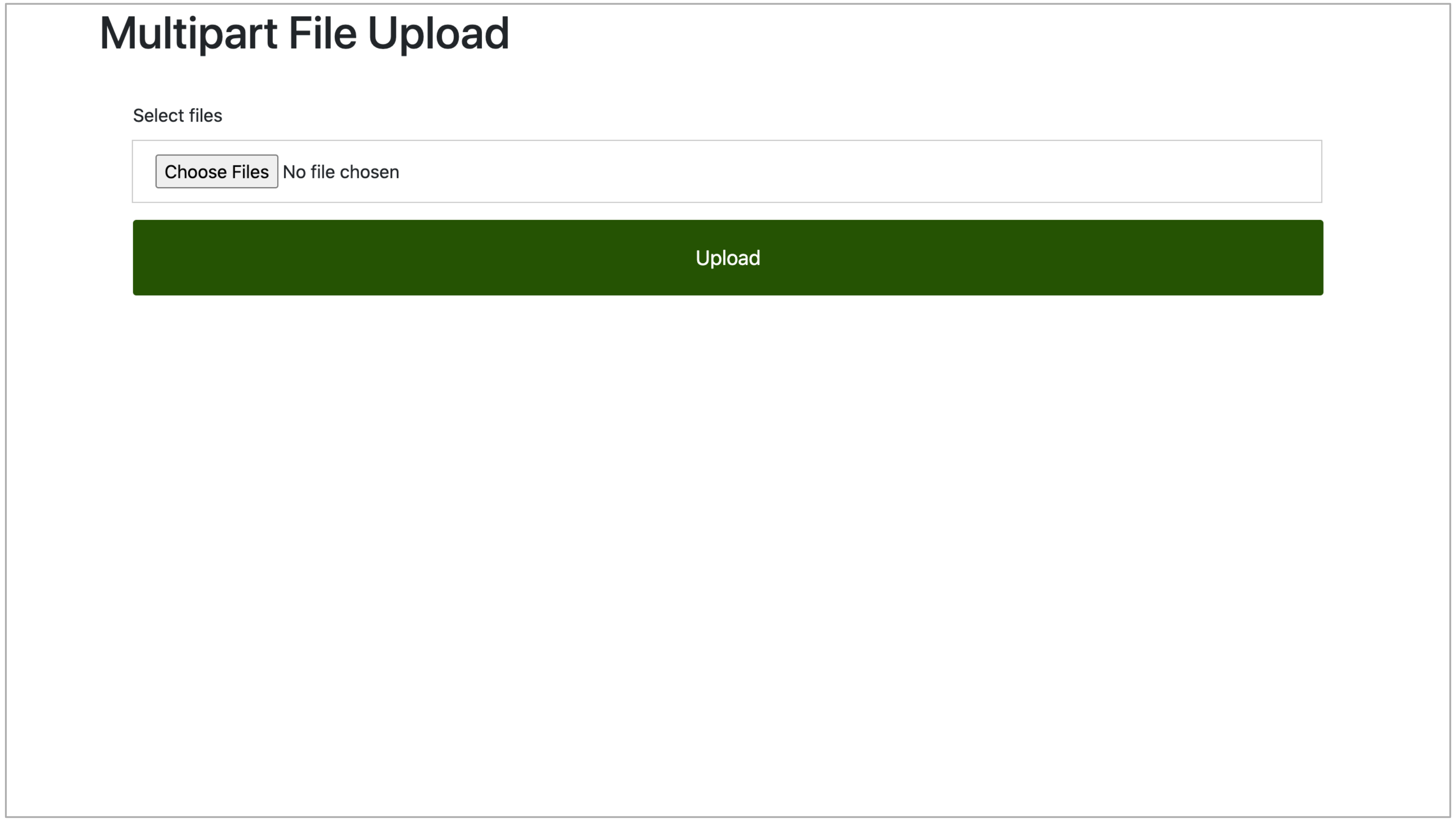
# Mastering Large-Size HTTP Requests in the Modern Web

Cherie Hsieh

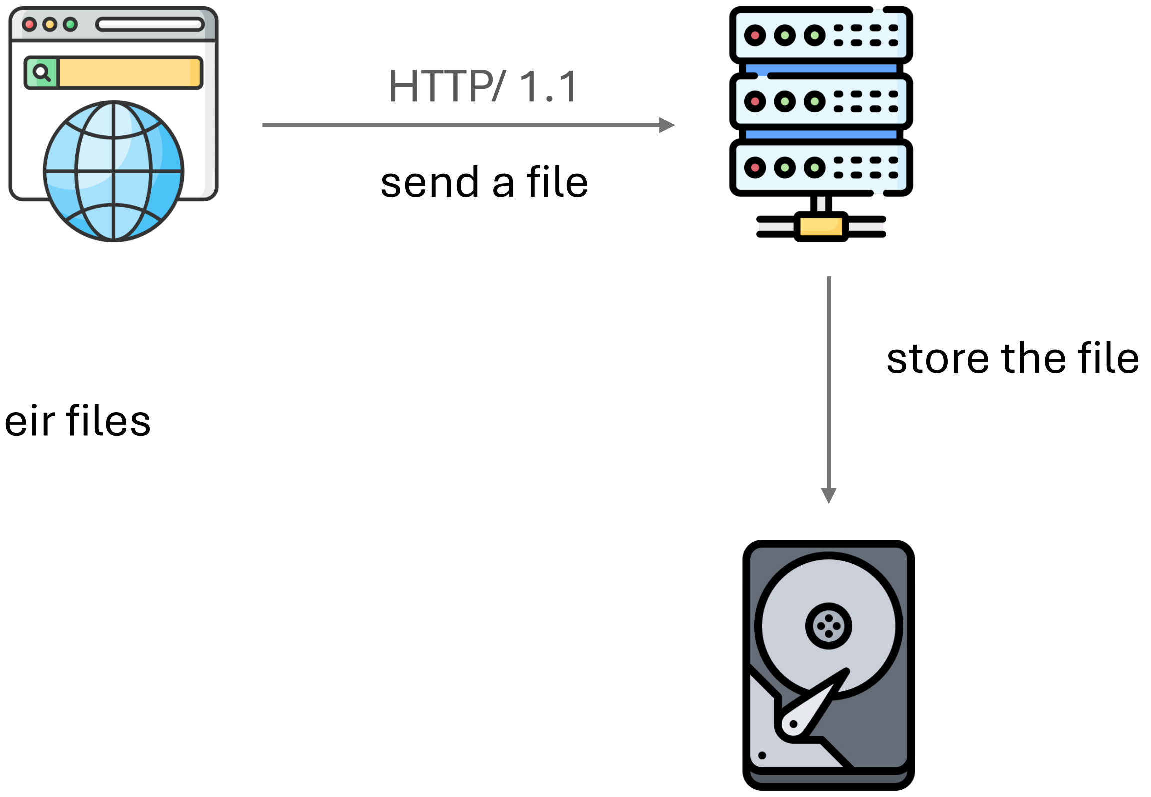
## About Me

- 💻 跨領域工作
  - 網路行銷 & PM
  - Web 前端工程師
  - Web Golang 後端工程師
  - 韌體工程師
  - TSMC SRE
- 💻 Golang Taiwan Co-organizer

## User Scenarios



# User Scenarios

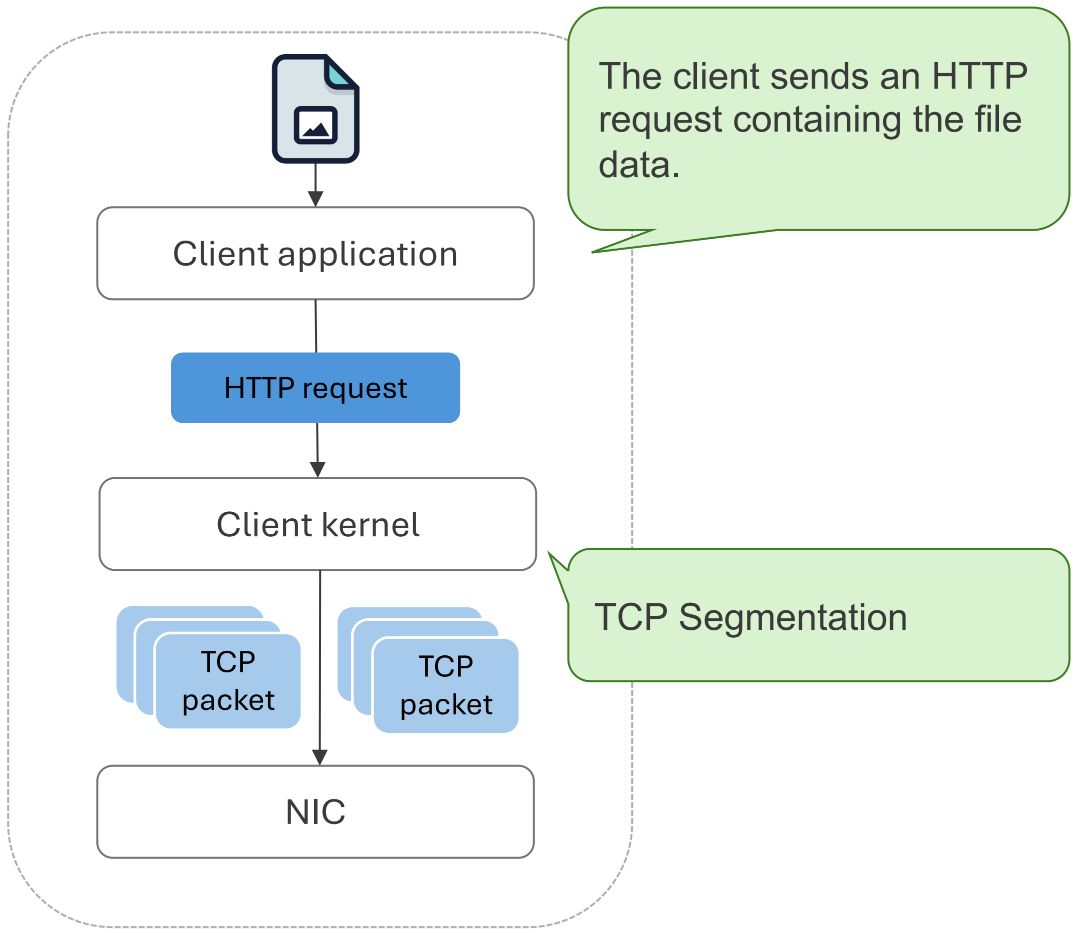


- A file uploading system that helps user upload their files
- Using HTTP/1.1 protocol
- Sync API

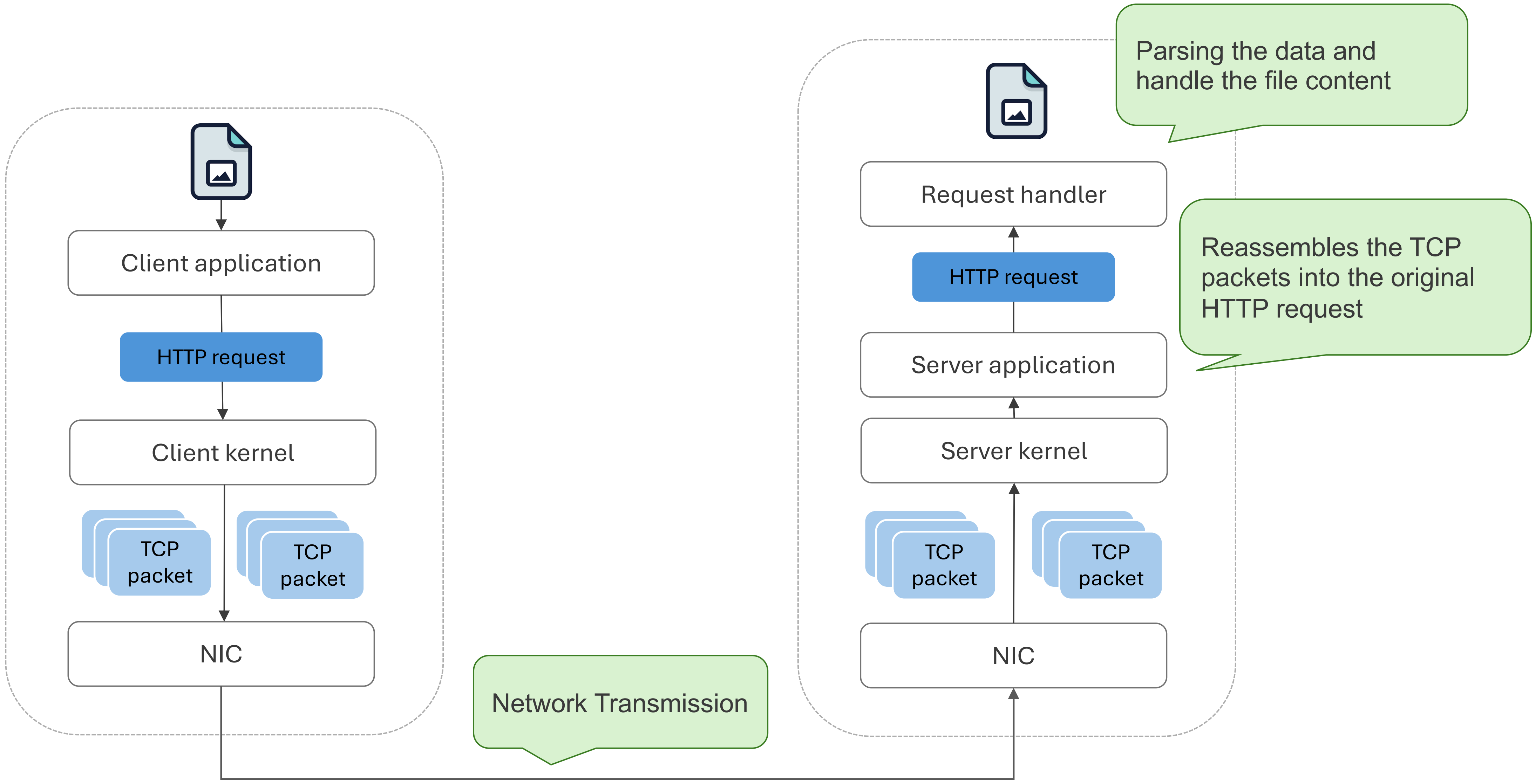
## How to Handle Large File Uploads Efficiently

under the constraints

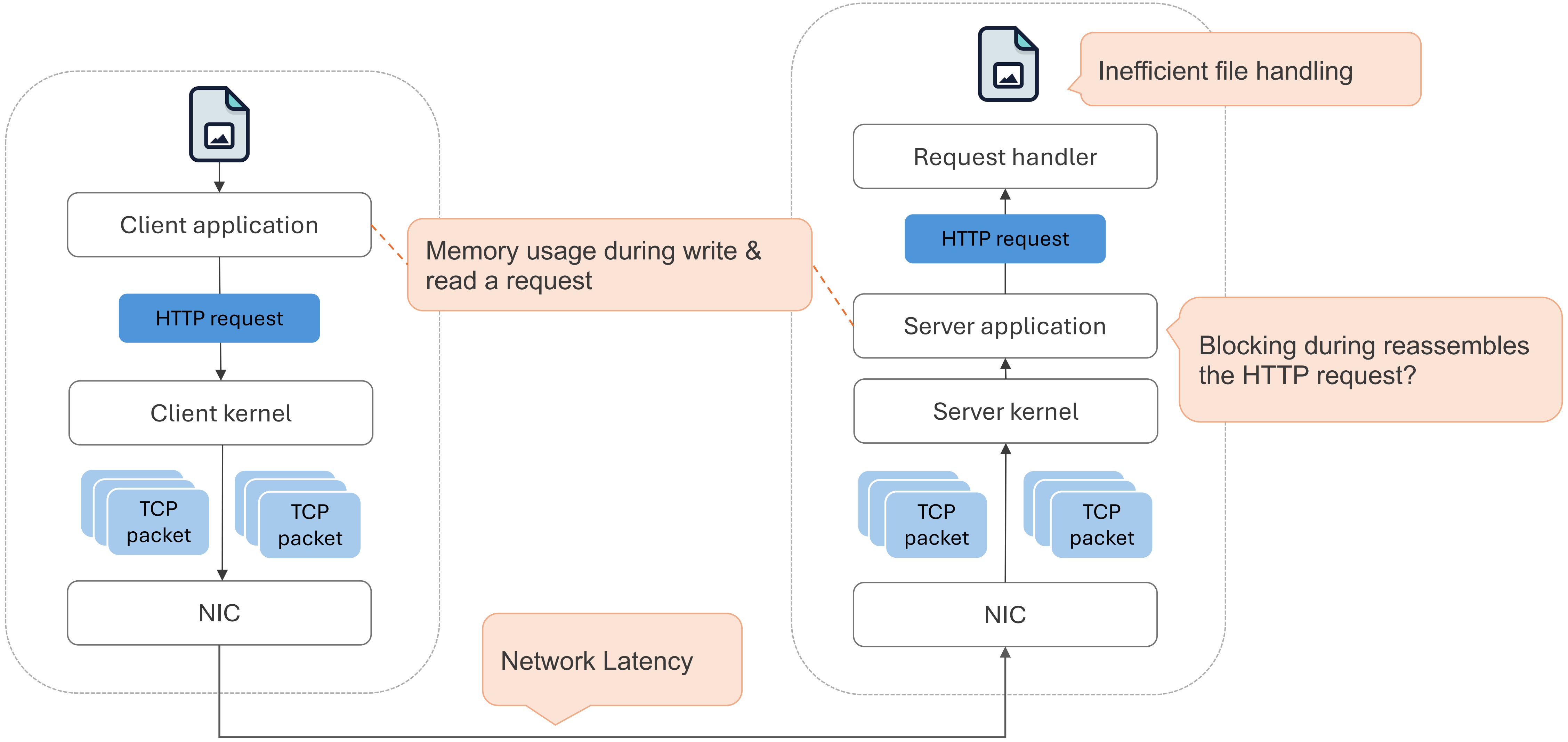
# A Journey of a File Uploading Request



# A Journey of a File Uploading Request



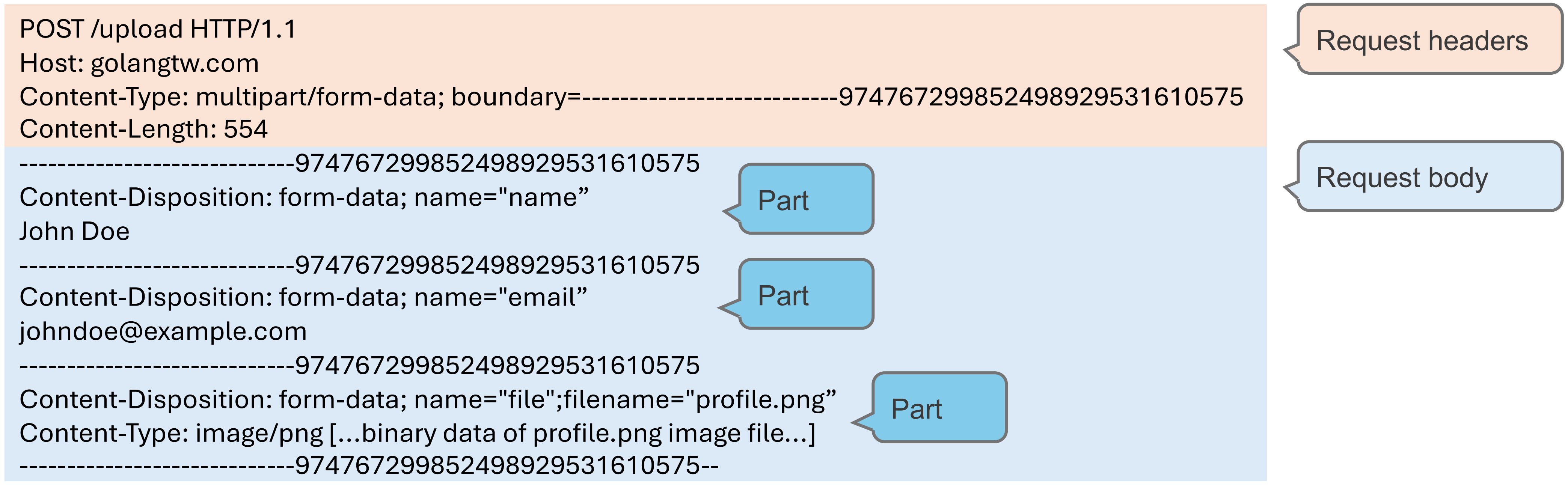
# Risks


Client application
HTTP request
Client kernel
TCP packet
TCP packet
NIC
Memory usage during write & read a request
Network Latency
Inefficient file handling
Request handler
HTTP request
Server application
Blocking during reassembles the HTTP request?
Server kernel
TCP packet
TCP packet
NIC

## How Golang net/http Package Handle the Request

How Golang net/http package handle the request

## HTTP Request Example



```
POST /upload HTTP/1.1
Host: golangtw.com
Content-Type: multipart/form-data; boundary=---------------------------974767299852498929531610575
Content-Length: 554
-----------------------------974767299852498929531610575
Content-Disposition: form-data; name="name”
John Doe
-----------------------------974767299852498929531610575
Content-Disposition: form-data; name="email”
johndoe@example.com
-----------------------------974767299852498929531610575
Content-Disposition: form-data; name="file";filename="profile.png”
Content-Type: image/png [...binary data of profile.png image file...]
-----------------------------974767299852498929531610575--
```

# How Golang net/http Package Handle the Request

```
POST /upload HTTP/1.1
Host: golangtw.com
Content-Type: multipart/form-data; boundary=---------------------------97476729985249892953161057 5
Content-Length: 554
---------------------------974767299852498929531610575
Content-Disposition: form-data; name="name”
John Doe
---------------------------974767299852498929531610575
Content-Disposition: form-data; name="email”
johndoe@example.com
---------------------------974767299852498929531610575
Content-Disposition: form-data; name="file";filename="profile.png”
Content-Type: image/png [...binary data of profile.png image file...]
---------------------------974767299852498929531610575--
```

Parse the request and pass it to the handler

```go
func handleUpload(w http.ResponseWriter, r *http.Request) {
  if err := r.ParseMultipartForm(maxMemory); err != nil {
    http.Error(w, "failed to parse form dat                    st)
    return
  }

  file, _, err := r.FormFile(fileKey)
  if err != nil {
    http.Error(w, "failed to retriev                                uest)
    return
  }
  defer file.Close()

  fileBytes, err := io.ReadAll(file)
  if err != nil {
    http.Error(w, "failed to read the file", http.StatusInternalServerError)
    return
  }
  // more code...
  slog.Info("successfully")
}
```

Parse the parts

Access the uploaded file

## Question 1

## Does net/http read the all content before passing the request to user’s handlers?

```
POST /upload HTTP/1.1
Host: golangtw.com
Content-Type: multipart/form-data; boundary=---------------------------974767299852498929531610575
Content-Length: 554
-----------------------------974767299852498929531610575
Content-Disposition: form-data; name="name”
John Doe
-----------------------------974767299852498929531610575--
```

The Go net/http package is responsible for **parsing the header and identifying the presence of a body.**

It doesn't automatically read the entire request body.

Question 2

## What happens if the size of the content being uploaded is large?

```go
func handleUpload(w http.ResponseWriter, r *http.Request) {
  if err := r.ParseMultipartForm(maxMemory); err != nil {
    http.Error(w, "failed to parse form data", http.StatusBadRequest)
    return
  }}
```

A file part exceeds **maxMemory**, it's written to a temporary file on disk.

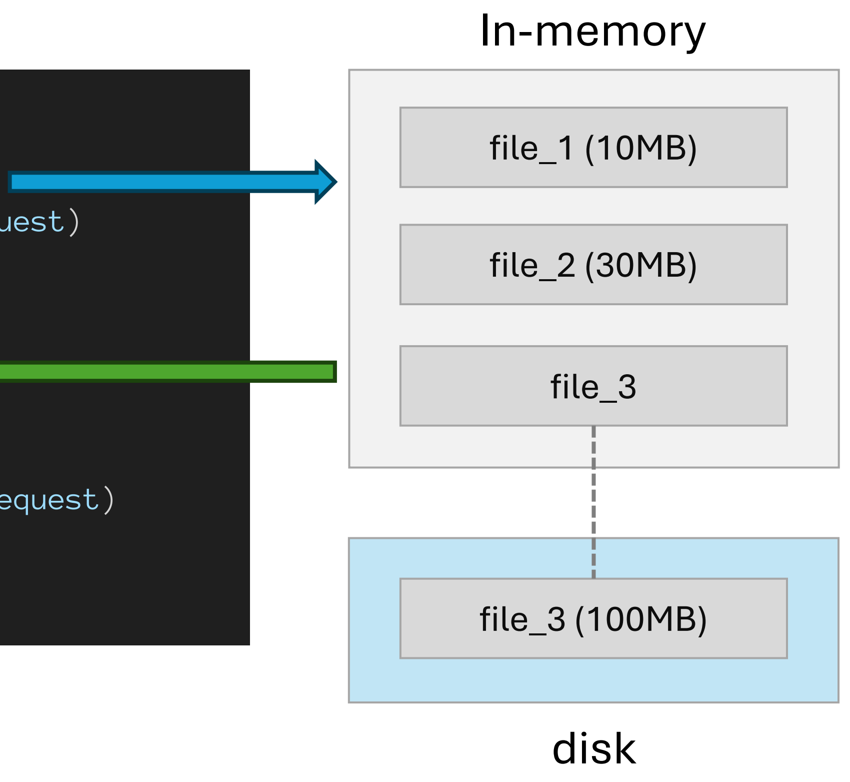
Smaller file parts might be kept entirely in memory.

Default: **32 MB**

## Question 2

## What happens if the size of the content being uploaded is large?

```go
func handleUpload(w http.ResponseWriter, r *http.Request) {

  if err := r.ParseMultipartForm(maxMemory); err != nil {
    http.Error(w, "failed to parse form data", http.StatusBadRequest)
    return
  }

  file, _, err := r.FormFile(fileKey)

  if err != nil {
    http.Error(w, "failed to retrieve the file", http.StatusBadRequest)
    return
  }
}
```



## Risks

```go
func handleUpload(w http.ResponseWriter, r *http.Request) {
  if err := r.ParseMultipartForm(maxMemory); err != nil {
    http.Error(w, "failed to parse form data", http.StatusBadRequest)
    return
  }

  file, _, err := r.FormFile(fileKey)
  if err != nil {
    http.Error(w, "failed to retrieve the file", http.StatusBadRequest)
    return
  }
  defer file.Close()

  fileBytes, err := io.ReadAll(file)

  if err != nil {
    http.Error(w, "failed to read the file", http.StatusInternalServerError)
    return
  }
  // more code...
  slog.Info("successfully")
}
```

handling many requests concurrently

many parts or complex structures within the form data

excessive memory usage

slow disk writes

## Streaming Data: Chunked Request

## Streaming Data: Chunked Request HTTP/1.1

```
POST /upload HTTP/1.1
Host: golangtw.com
Transfer-Encoding: chunked
Content-Disposition: attachment; filename="golang_is_good.txt"

2000\r\n
[8192 bytes of file content]\r\n
```

Content-Length header is not required

Request 1

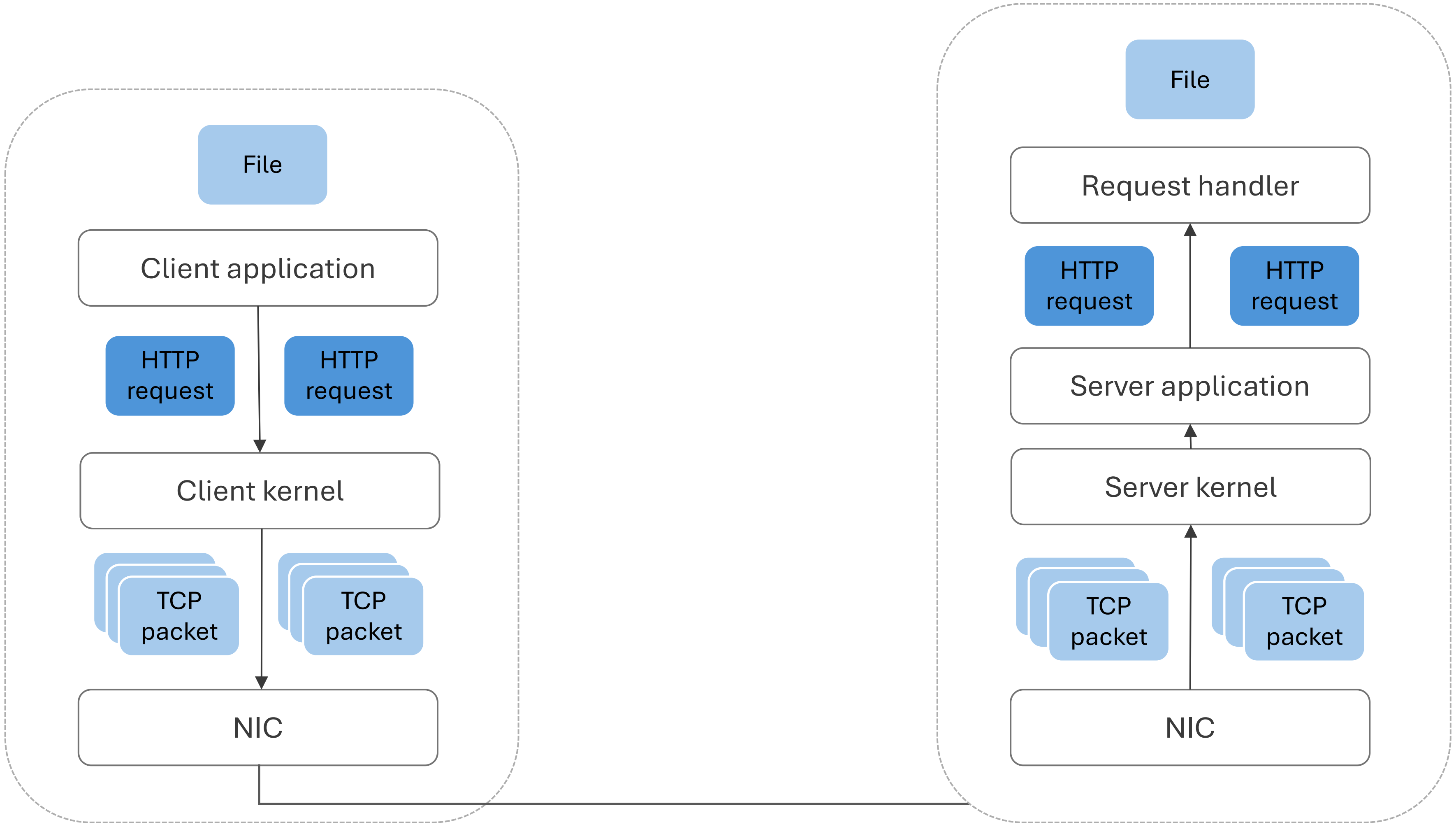
```
2000\r\n
[8192 bytes of file content]\r\n
```

Request 2

```
1500\r\n
[6144 bytes of file content]\r\n
0\r\n  // Final chunk with size 0
```

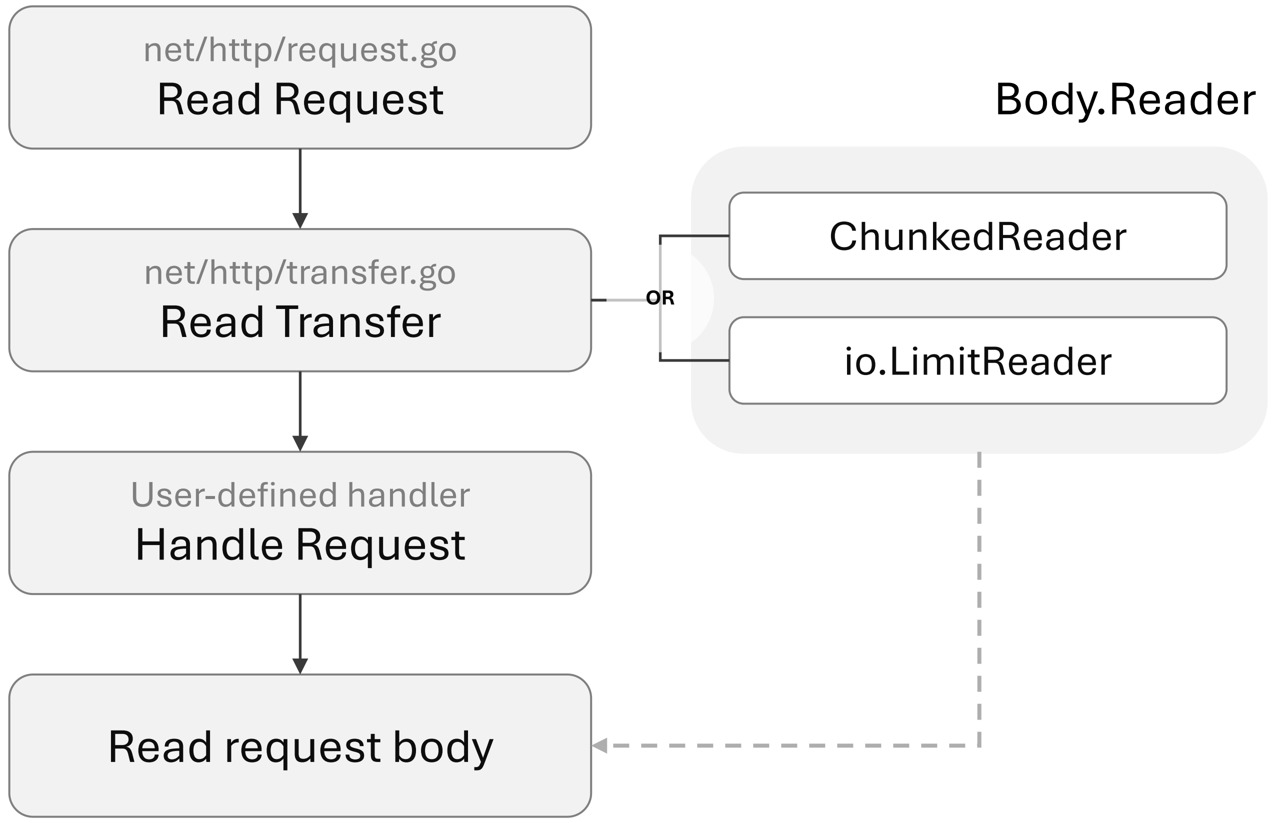
Request 3

## Streaming Data: Chunked Request HTTP/1.1



## How Golang net/http package handle chunked requests

The server receives a request with Transfer-Encoding:chunked, it **automatically dechunks** the data for user. User can access the dechunked data through the http.Request.Body field.

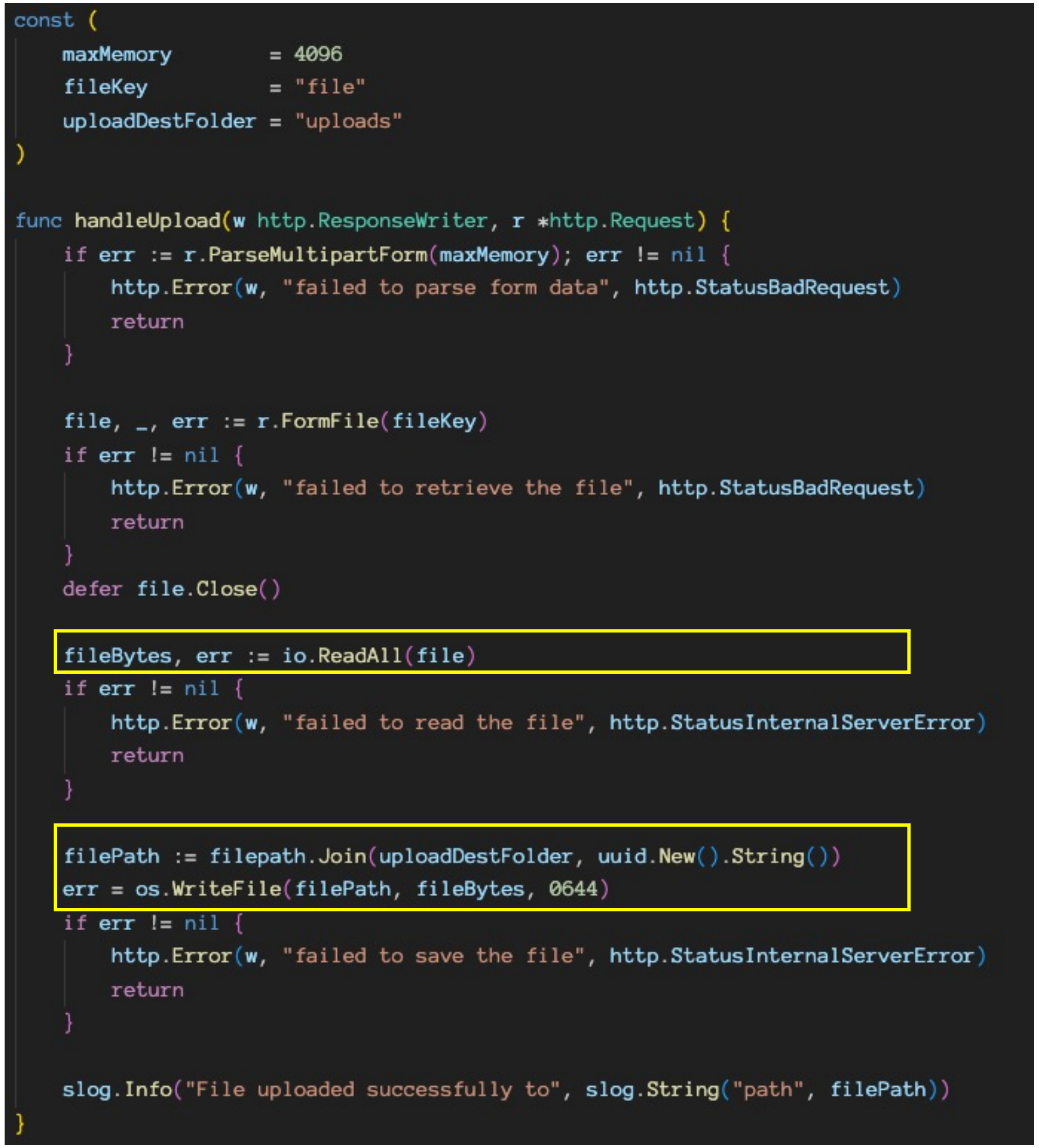


How can we optimize the code for better performance

How can we optimize the code for better performance

## Server Side - Original

excessive memory usage

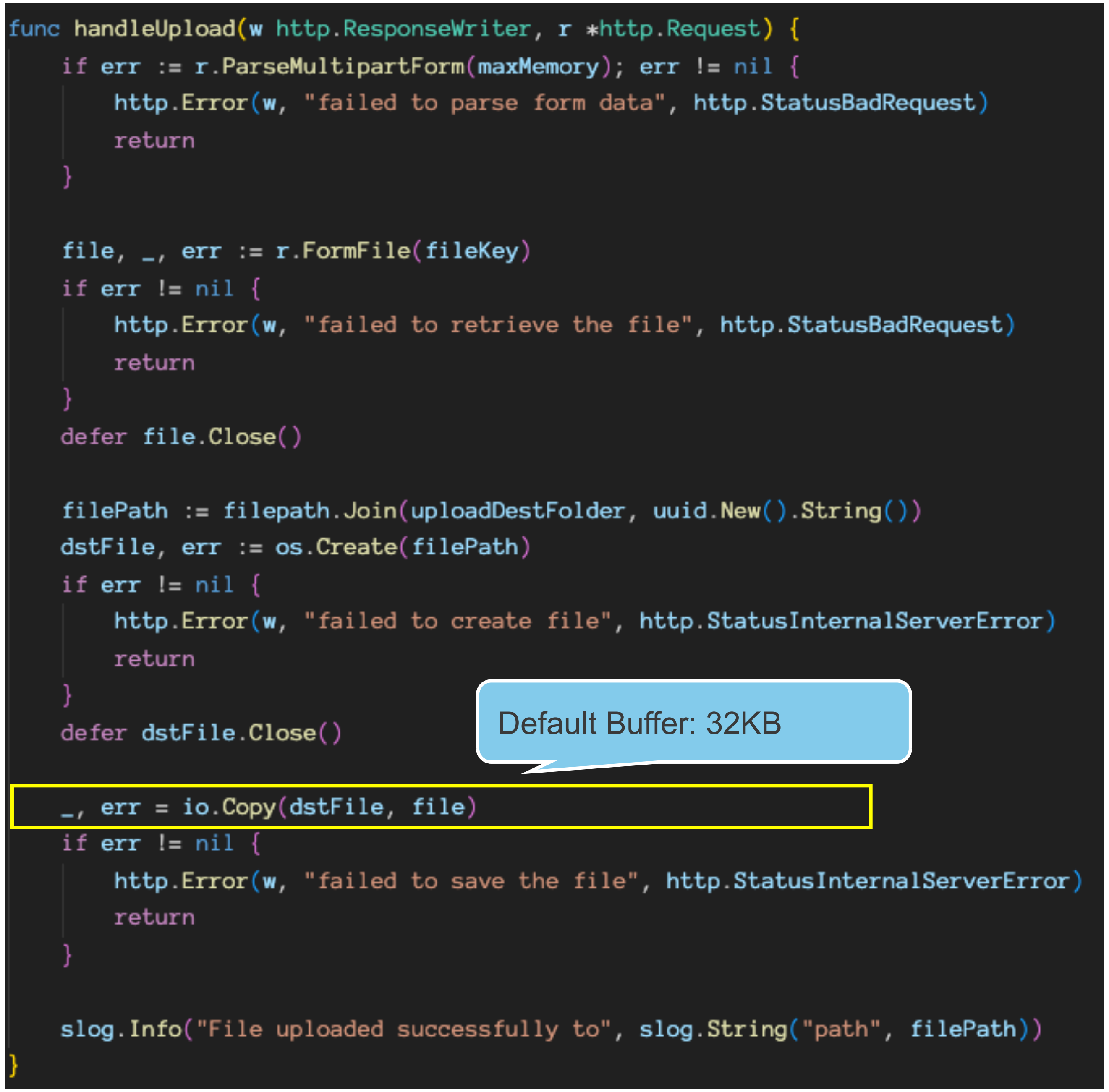

```go
const (
    maxMemory        = 4096
    fileKey          = "file"
    uploadDestFolder = "uploads"
)

func handleUpload(w http.ResponseWriter, r *http.Request) {
    if err := r.ParseMultipartForm(maxMemory); err != nil {
        http.Error(w, "failed to parse form data", http.StatusBadRequest)
        return
    }

    file, _, err := r.FormFile(fileKey)
    if err != nil {
        http.Error(w, "failed to retrieve the file", http.StatusBadRequest)
        return
    }
    defer file.Close()

    fileBytes, err := io.ReadAll(file)
    if err != nil {
        http.Error(w, "failed to read the file", http.StatusInternalServerError)
        return
    }

    filePath := filepath.Join(uploadDestFolder, uuid.New().String())
    err = os.WriteFile(filePath, fileBytes, 0644)
    if err != nil {
        http.Error(w, "failed to save the file", http.StatusInternalServerError)
        return
    }

    slog.Info("File uploaded successfully to", slog.String("path", filePath))
}
```

How can we optimize the code for better performance

## Server Side - Limit Read

| Benchmark | Average Time per Operation (ns) | Average Memory Allocation (B) |
|---|---|---|
| ReadAll | 350,250,692 | 615,239,886 |
| **LimitRead** | **124,428,701** | **7,316** |

Limit Read the source content into the destination



```go
func handleUpload(w http.ResponseWriter, r *http.Request) {
    if err := r.ParseMultipartForm(maxMemory); err != nil {
        http.Error(w, "failed to parse form data", http.StatusBadRequest)
        return
    }

    file, _, err := r.FormFile(fileKey)
    if err != nil {
        http.Error(w, "failed to retrieve the file", http.StatusBadRequest)
        return
    }
    defer file.Close()

    filePath := filepath.Join(uploadDestFolder, uuid.New().String())
    dstFile, err := os.Create(filePath)
    if err != nil {
        http.Error(w, "failed to create file", http.StatusInternalServerError)
        return
    }
    defer dstFile.Close()

    _, err = io.Copy(dstFile, file)
    if err != nil {
        http.Error(w, "failed to save the file", http.StatusInternalServerError)
        return
    }

    slog.Info("File uploaded successfully to", slog.String("path", filePath))
}
```

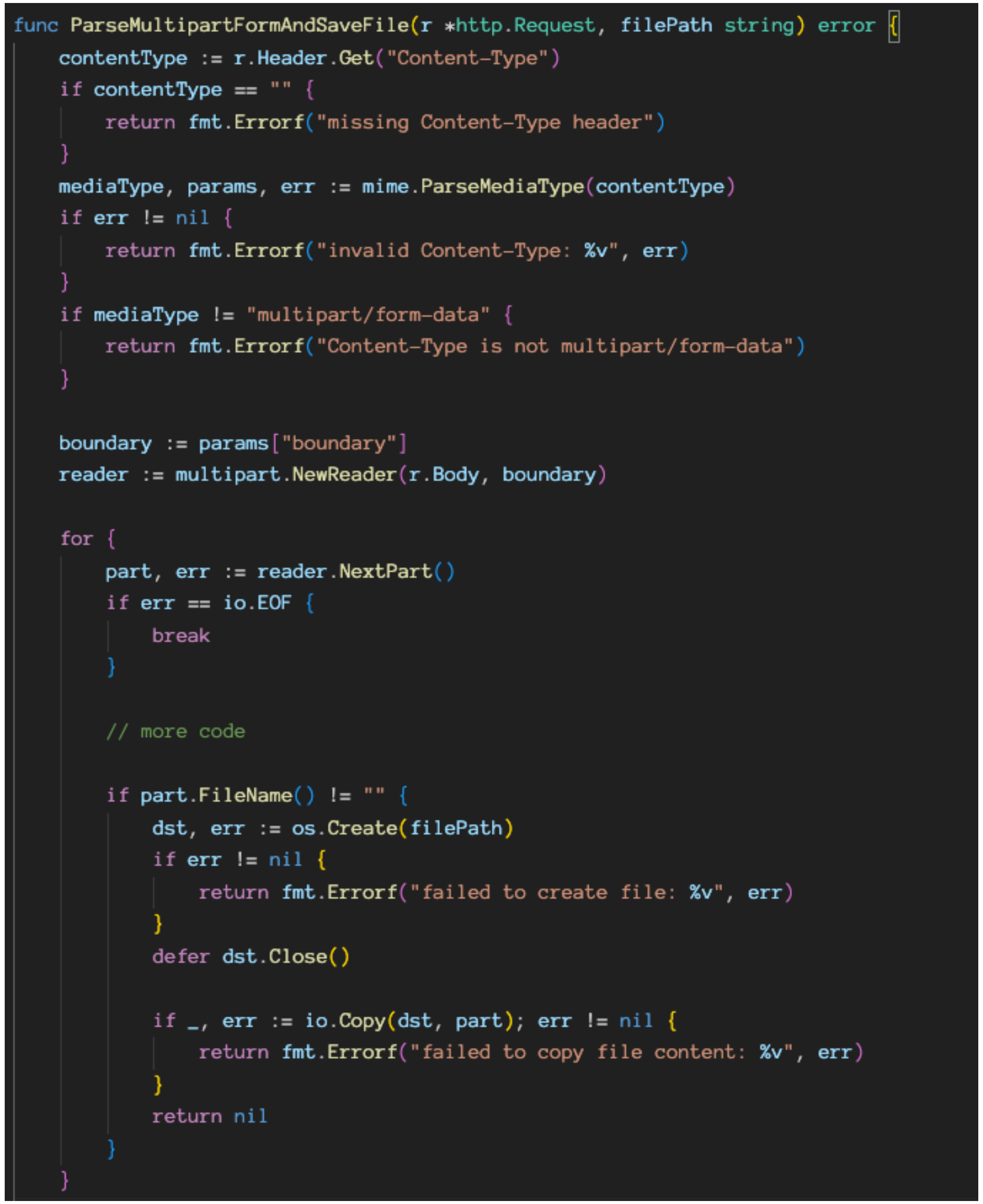
Default Buffer: 32KB

How can we optimize the code for better performance

## Server Side - Custom

| Benchmark | Average Time per Operation (ns) | Average Memory Allocation (B) |
|---|---:|---:|
| ReadAll | 350,250,692 | 615,239,886 |
| LimitRead | 124,428,701 | 7,316 |
| **Custom** | **89,152,902** | **5,301** |

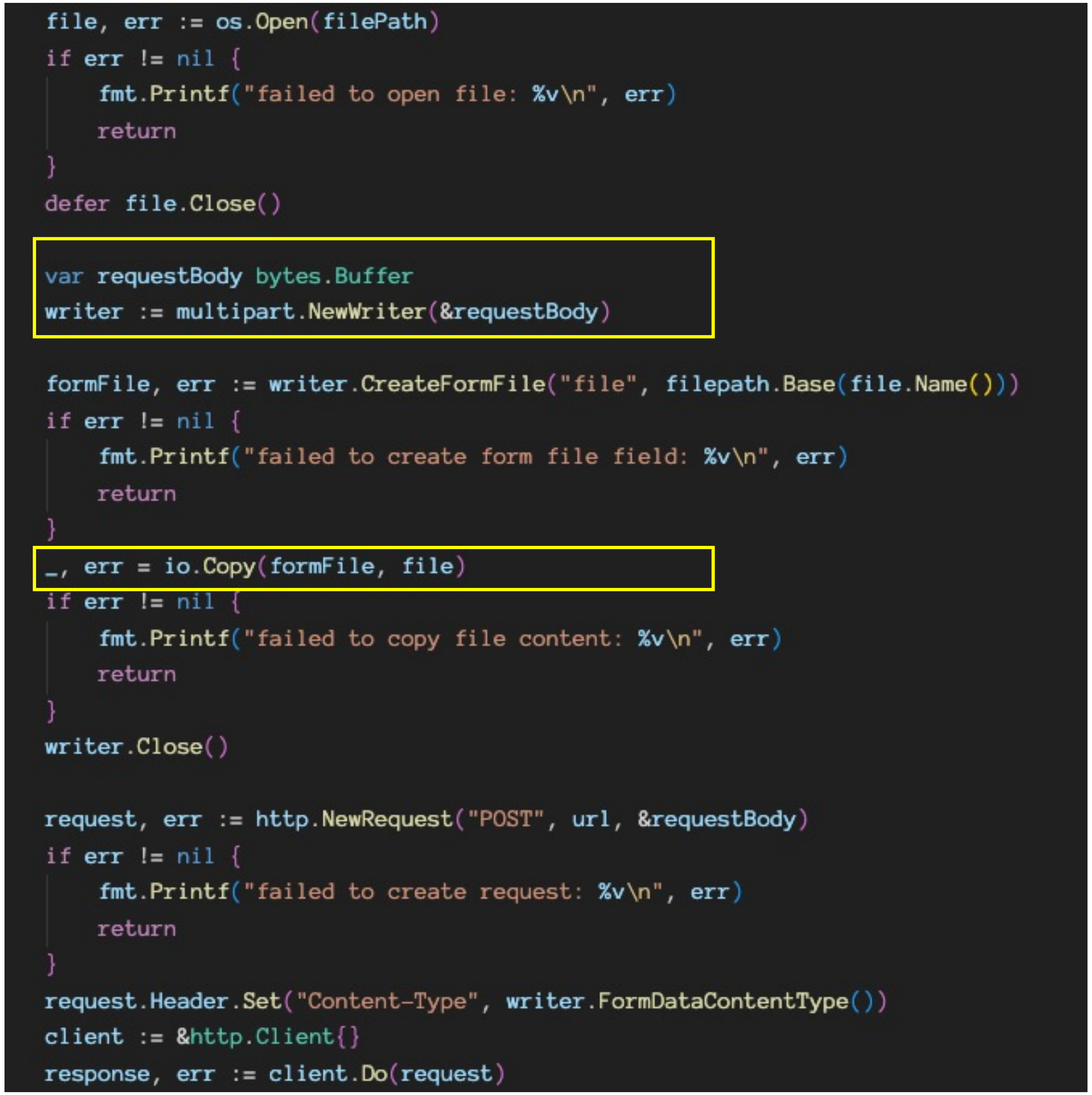
😁 Avoid saving the content into temp. file



```go
func ParseMultipartFormAndSaveFile(r *http.Request, filePath string) error {
    contentType := r.Header.Get("Content-Type")
    if contentType == "" {
        return fmt.Errorf("missing Content-Type header")
    }
    mediaType, params, err := mime.ParseMediaType(contentType)
    if err != nil {
        return fmt.Errorf("invalid Content-Type: %v", err)
    }
    if mediaType != "multipart/form-data" {
        return fmt.Errorf("Content-Type is not multipart/form-data")
    }

    boundary := params["boundary"]
    reader := multipart.NewReader(r.Body, boundary)

    for {
        part, err := reader.NextPart()
        if err == io.EOF {
            break
        }

        // more code

        if part.FileName() != "" {
            dst, err := os.Create(filePath)
            if err != nil {
                return fmt.Errorf("failed to create file: %v", err)
            }
            defer dst.Close()

            if _, err := io.Copy(dst, part); err != nil {
                return fmt.Errorf("failed to copy file content: %v", err)
            }
            return nil
        }
    }
```

How can we optimize the code for better performance

## Client Side - Original

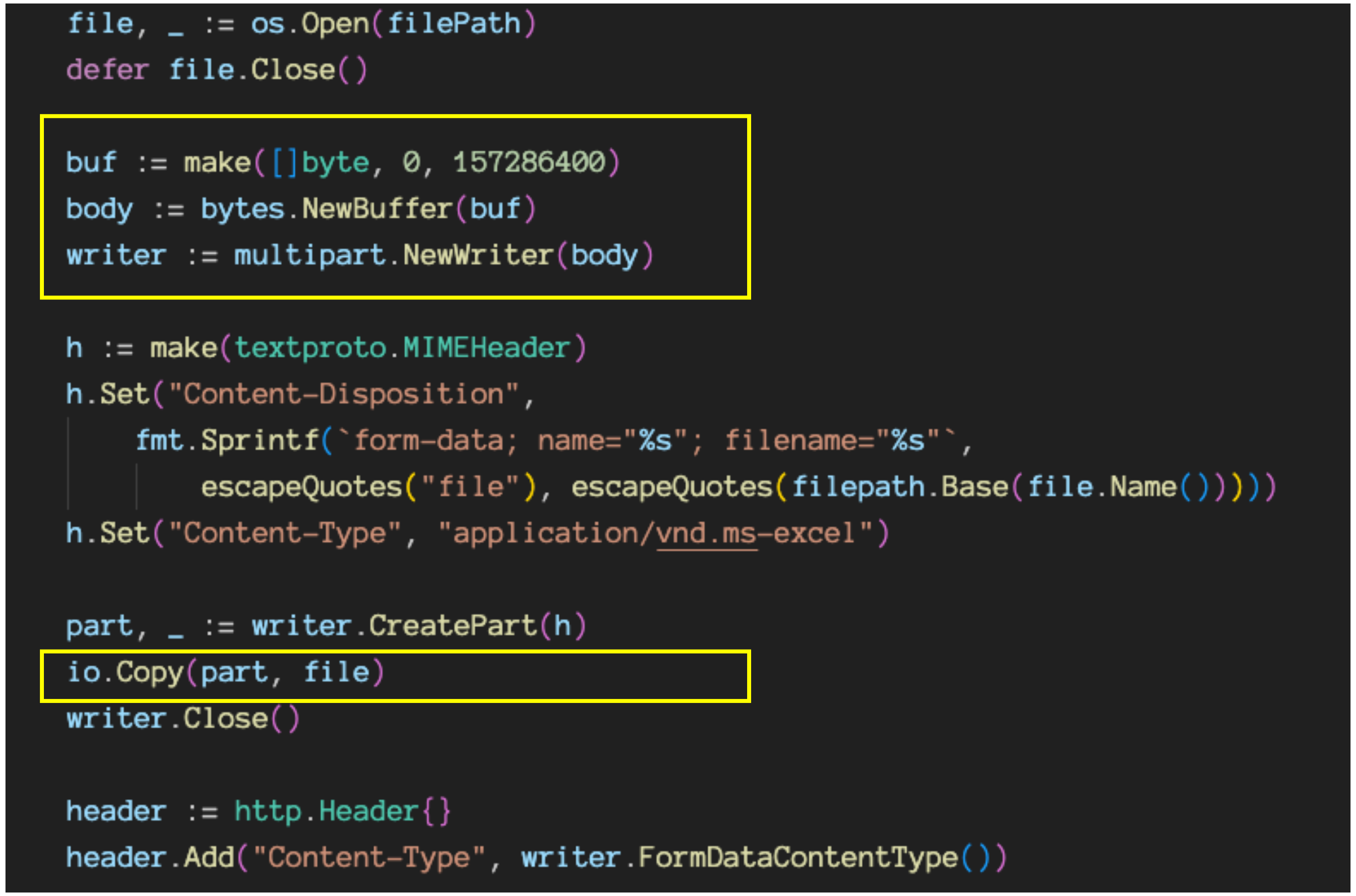


```go
file, err := os.Open(filePath)
if err != nil {
    fmt.Printf("failed to open file: %v\n", err)
    return
}
defer file.Close()

var requestBody bytes.Buffer
writer := multipart.NewWriter(&requestBody)

formFile, err := writer.CreateFormFile("file", filepath.Base(file.Name()))
if err != nil {
    fmt.Printf("failed to create form file field: %v\n", err)
    return
}
_, err = io.Copy(formFile, file)
if err != nil {
    fmt.Printf("failed to copy file content: %v\n", err)
    return
}
writer.Close()

request, err := http.NewRequest("POST", url, &requestBody)
if err != nil {
    fmt.Printf("failed to create request: %v\n", err)
    return
}
request.Header.Set("Content-Type", writer.FormDataContentType())
client := &http.Client{}
response, err := client.Do(request)
```

How can we optimize the code for better performance

## Client Side - Pre-Allocate

😁 Pe-allocate the buffer to prevent slice.extend

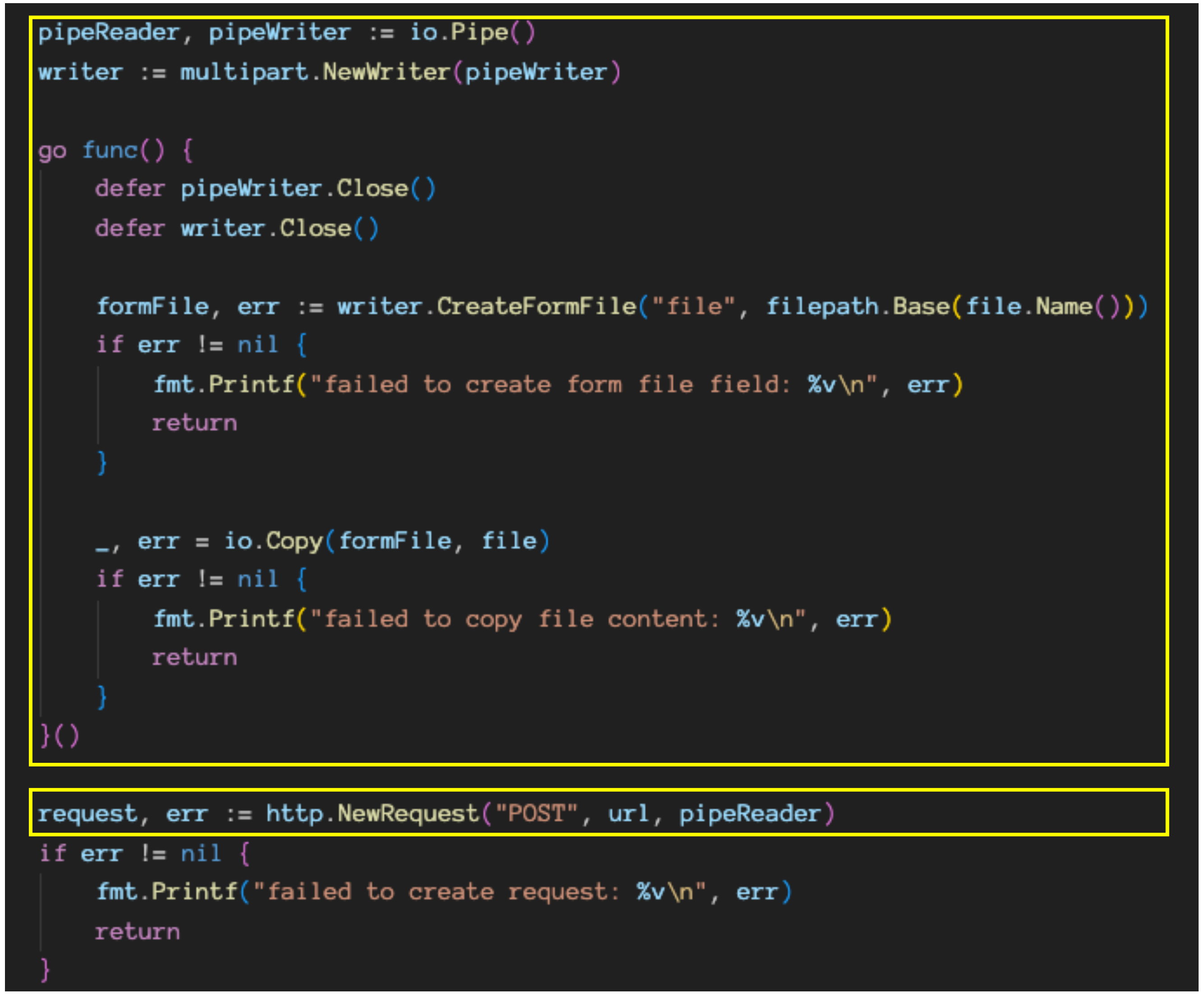


```go
file, _ := os.Open(filePath)
defer file.Close()

buf := make([]byte, 0, 157286400)
body := bytes.NewBuffer(buf)
writer := multipart.NewWriter(body)

h := make(textproto.MIMEHeader)
h.Set("Content-Disposition",
    fmt.Sprintf(`form-data; name="%s"; filename="%s"`,
        escapeQuotes("file"), escapeQuotes(filepath.Base(file.Name()))))
h.Set("Content-Type", "application/vnd.ms-excel")

part, _ := writer.CreatePart(h)
io.Copy(part, file)
writer.Close()

header := http.Header{}
header.Add("Content-Type", writer.FormDataContentType())
```

How can we optimize the code for better performance

## Client Side - Pipe

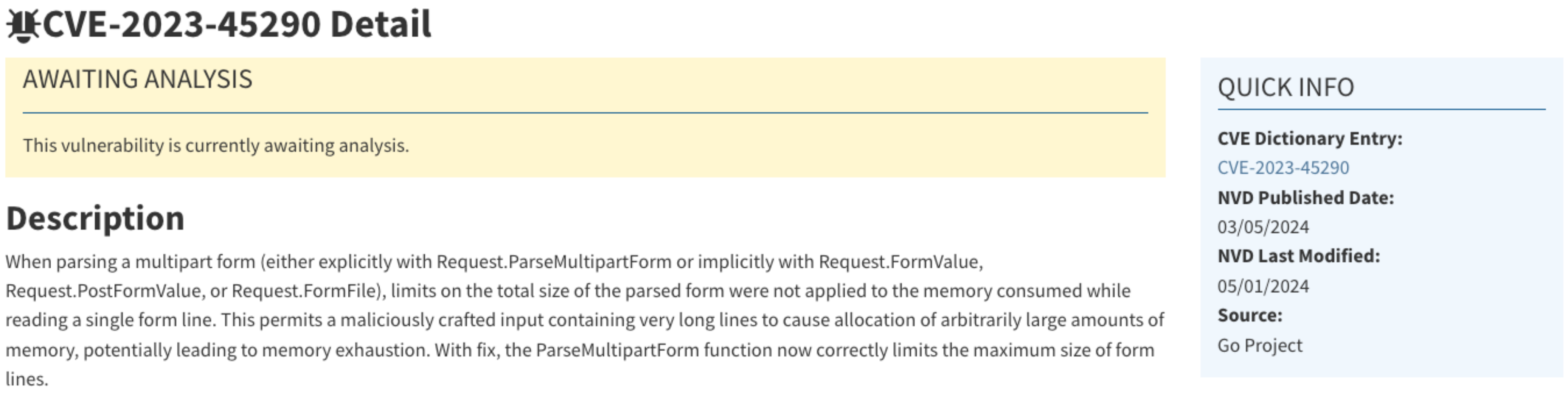
😁 Using io.Pipe to prevent load the entire file into memory.



```go
pipeReader, pipeWriter := io.Pipe()
writer := multipart.NewWriter(pipeWriter)

go func() {
    defer pipeWriter.Close()
    defer writer.Close()

    formFile, err := writer.CreateFormFile("file", filepath.Base(file.Name()))
    if err != nil {
        fmt.Printf("failed to create form file field: %v\n", err)
        return
    }

    _, err = io.Copy(formFile, file)
    if err != nil {
        fmt.Printf("failed to copy file content: %v\n", err)
        return
    }
}()

request, err := http.NewRequest("POST", url, pipeReader)
if err != nil {
    fmt.Printf("failed to create request: %v\n", err)
    return
}
```

Security Issue

## Memory Exhaustion in Request.ParseMultipartForm



### CVE-2023-45290 Detail

AWAITING ANALYSIS

This vulnerability is currently awaiting analysis.

**Description**

When parsing a multipart form (either explicitly with Request.ParseMultipartForm or implicitly with Request.FormValue, Request.PostFormValue, or Request.FormFile), limits on the total size of the parsed form were not applied to the memory consumed while reading a single form line. This permits a maliciously crafted input containing very long lines to cause allocation of arbitrarily large amounts of memory, potentially leading to memory exhaustion. With fix, the ParseMultipartForm function now correctly limits the maximum size of form lines.

QUICK INFO

**CVE Dictionary Entry:**
CVE-2023-45290
**NVD Published Date:**
03/05/2024
**NVD Last Modified:**
05/01/2024
**Source:**
Go Project

Fixed: 2024/05/06

## Wrapping Up

- How the Golang net/http package handles HTTP requests
  - Passes requests to user-defined handlers without reading the entire request body
  - May store form data in temporary files if the size exceeds a limit
  - Automatically handles chunked requests

- Performance improvement in server-side
  - Use limit reading method to read the request body
  - Customizing the file-handling process to avoid saving the content to a temporary file

- Performance improvement in client-side
  - Using io.Pipe to prevent load the entire file into memory