Serving Web Content with Dynamic Process Networks in Go

Jim Whitehead - University of Oxford
Department of Computer Science

Compositional concurrency

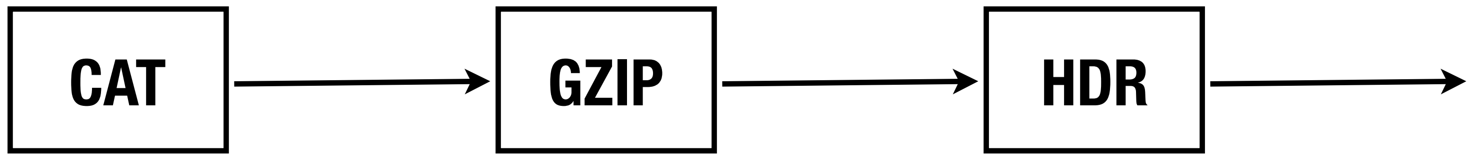
- Unix Pipes and Filters:

```
tr -c A-Za-z ‘\n’ | sort | uniq -c | sort -nr
```

- Message = stream of bytes
- Simple and understandable

Building a web server

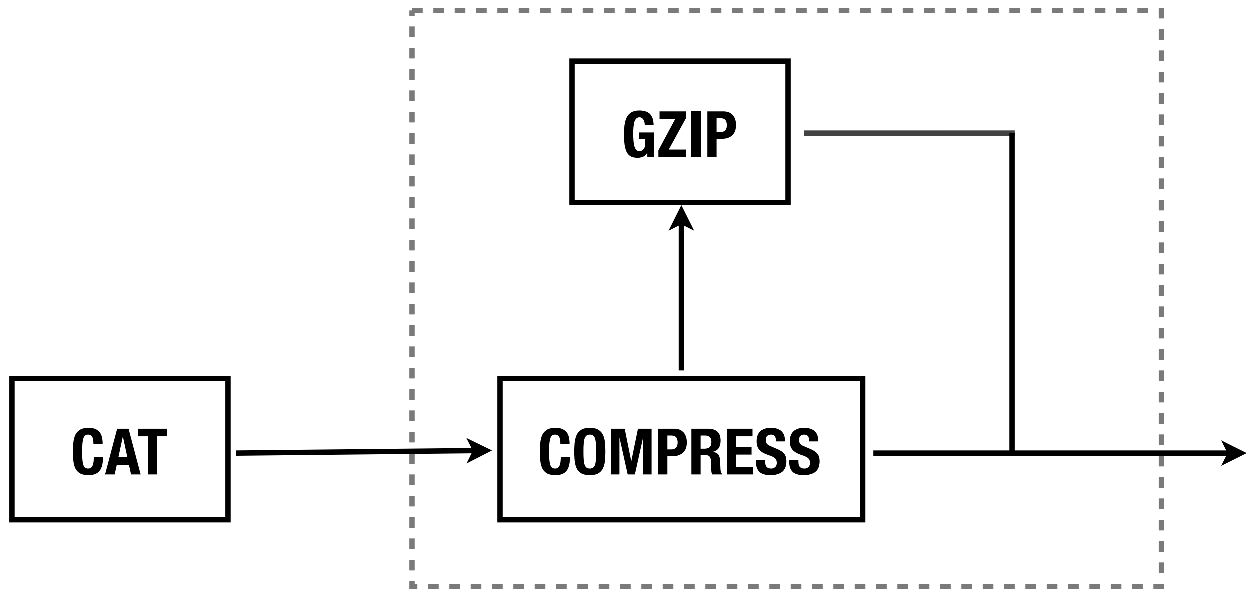
- Tractable
- Explicit
- Avoid wizardry



```
HTTP/1.0 200 OK
CONTENT-ENCODING: GZIP
CONTENT-LENGTH: 4109
CONTENT-TYPE: TEXT/HTML; CHARSET=UTF-8
DATE: FRI, 20 MAY 2011 12:33:38 GMT
LAST-MODIFIED: MON, 14 FEB 2011 12:25:42 GMT

WO?6?;?W?@?A??E▯N6?&?Z????EW
                            ?@K?ŘU??C
? ??{W?????;?/▯O/Q?+M?>{ZF?(I2}~????%?▯}????OHJE?W?:I??(*?O$?J??W'???D?*?
B?CV?_G~?3?}M???ꞆS|ZZ?Y[)R<??ZN[)鍜P?TV?^?4?F??JZ?R|3??Z?GG?????麑EڿQ??
O8???:??в????NG?%?Ç^PIE1?2???Z?X??J=??OWJ?#??1???Ɠ??8?T??V?U|??-?K??A4I????
S???\U?+?+???????-?
                                                             P??K?E=O???
```

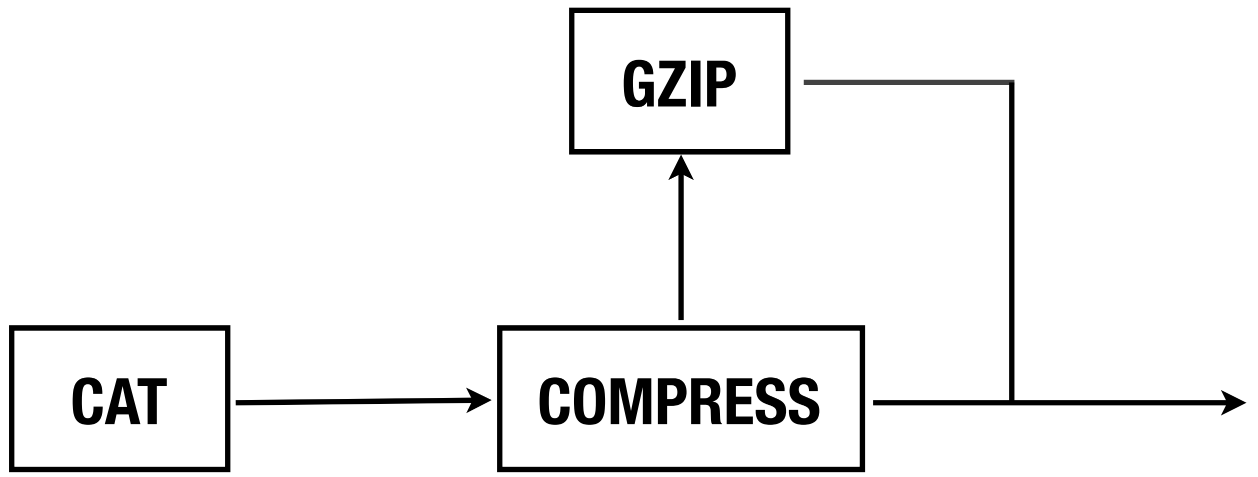
- Message = header + stream of bytes
- Component-specific headers

Component networks

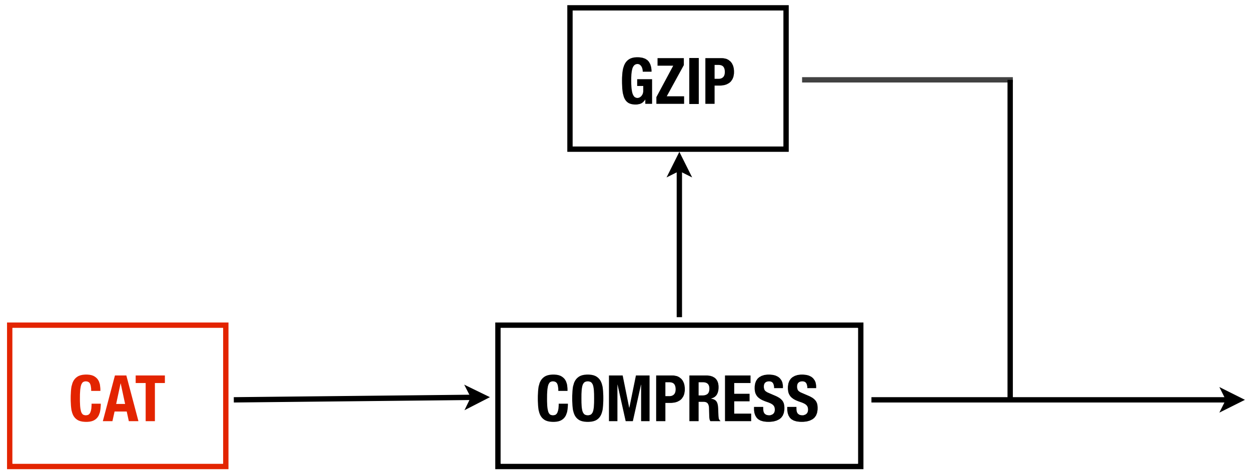


- Two stages
- All possible paths

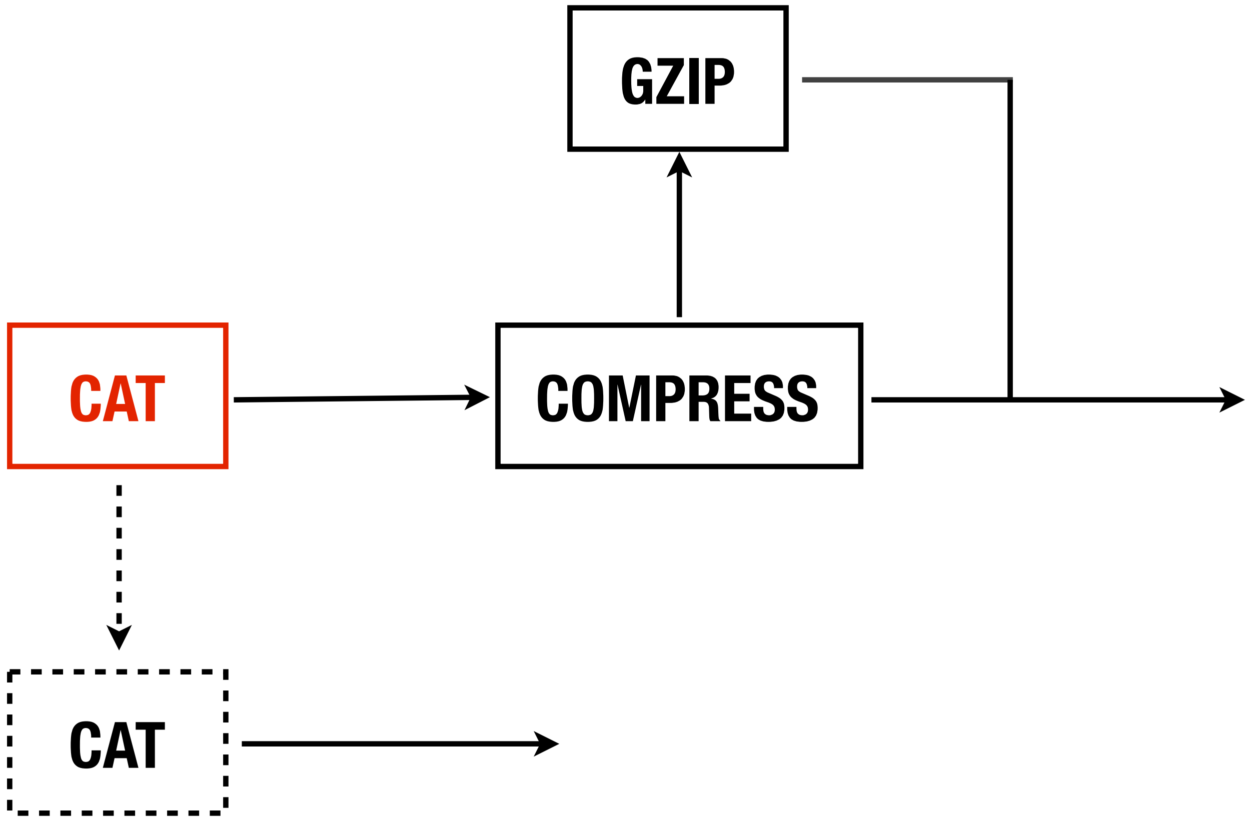
Handling a request



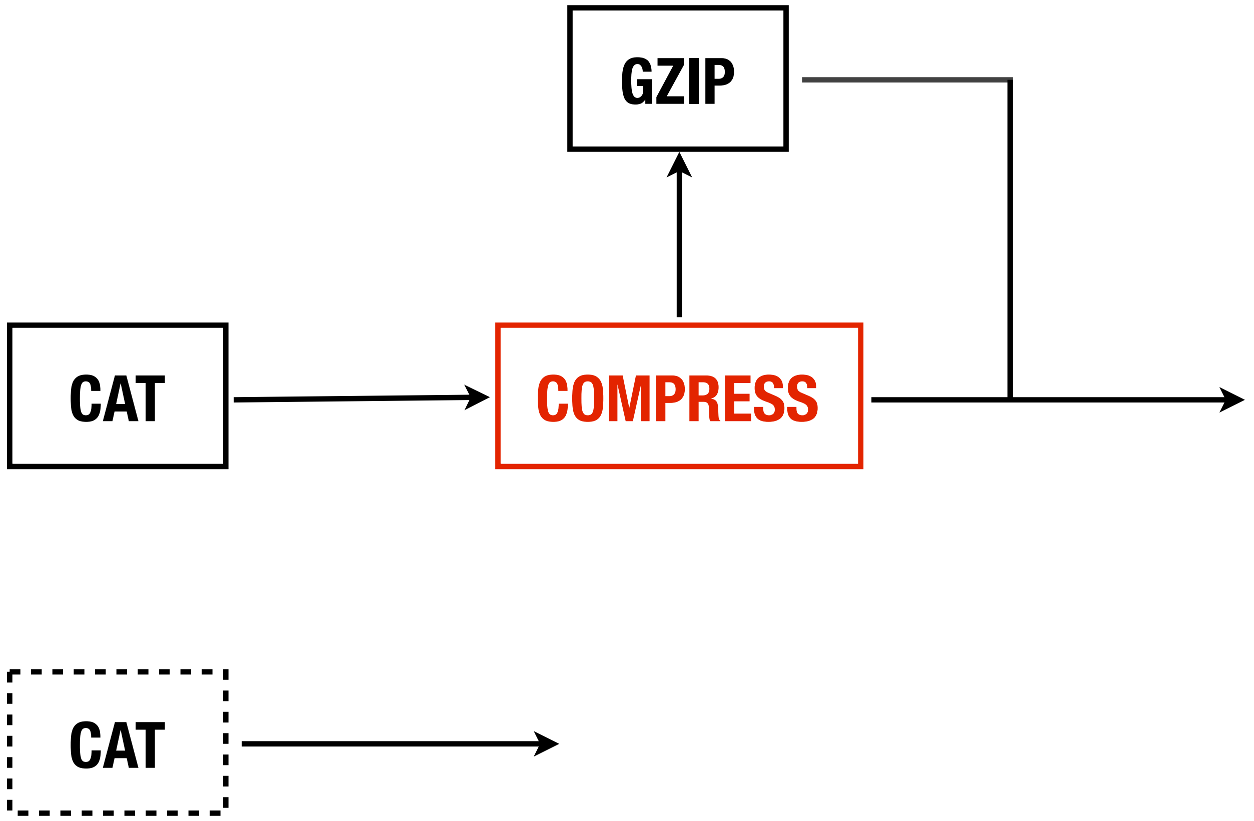
Handling a request



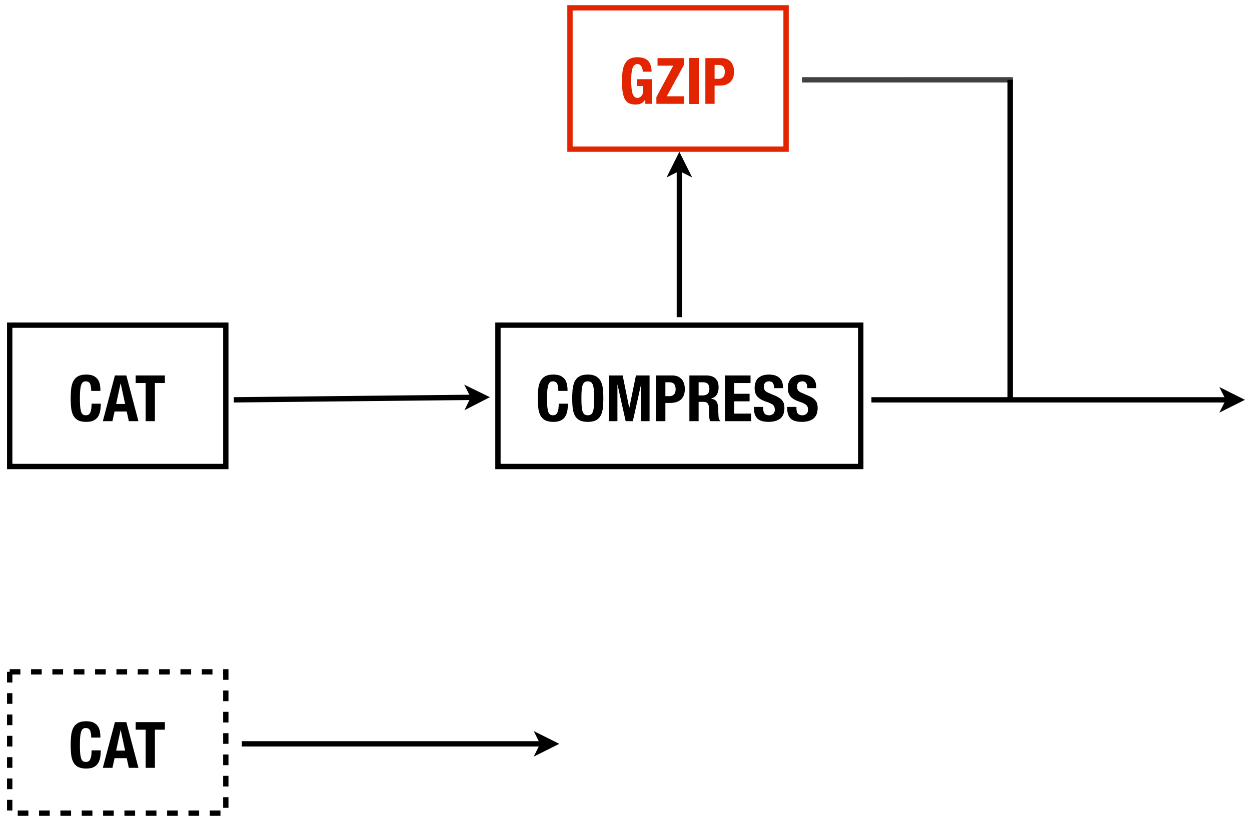
Handling a request



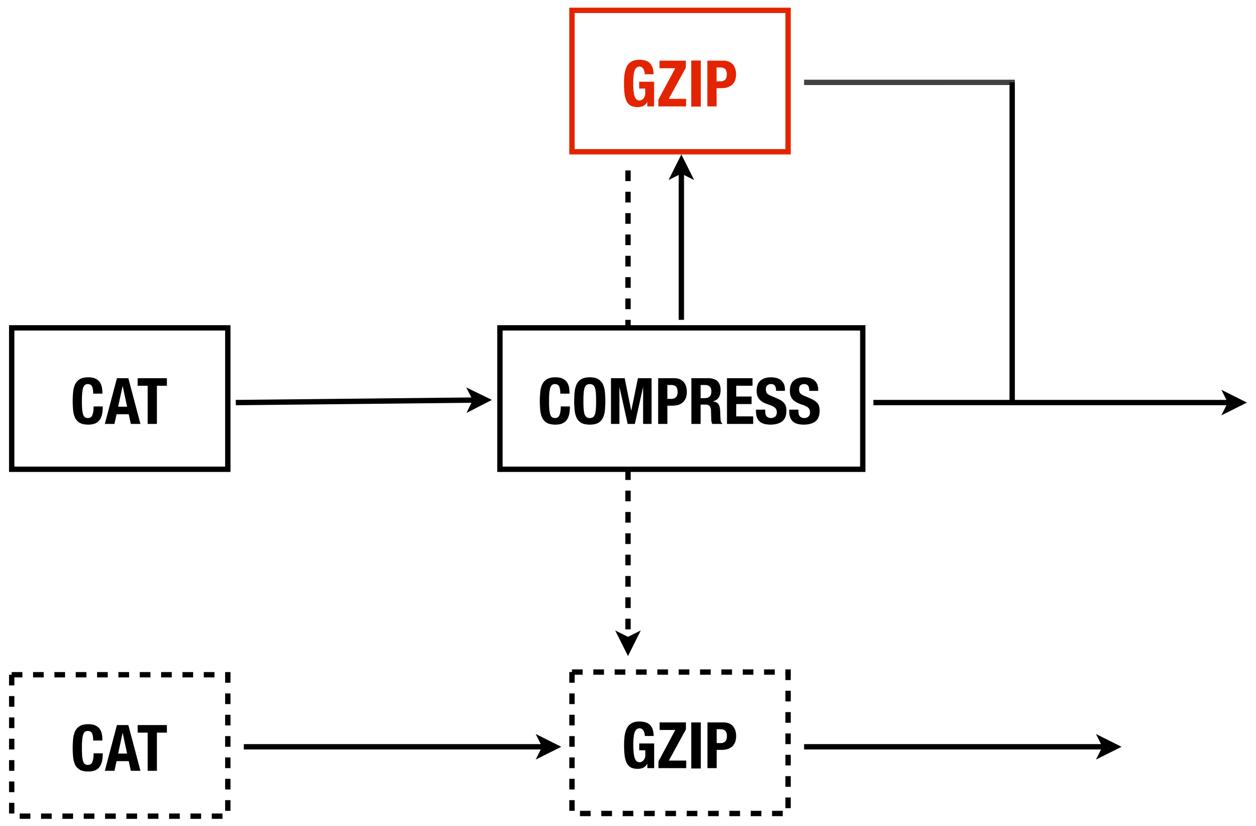
Handling a request



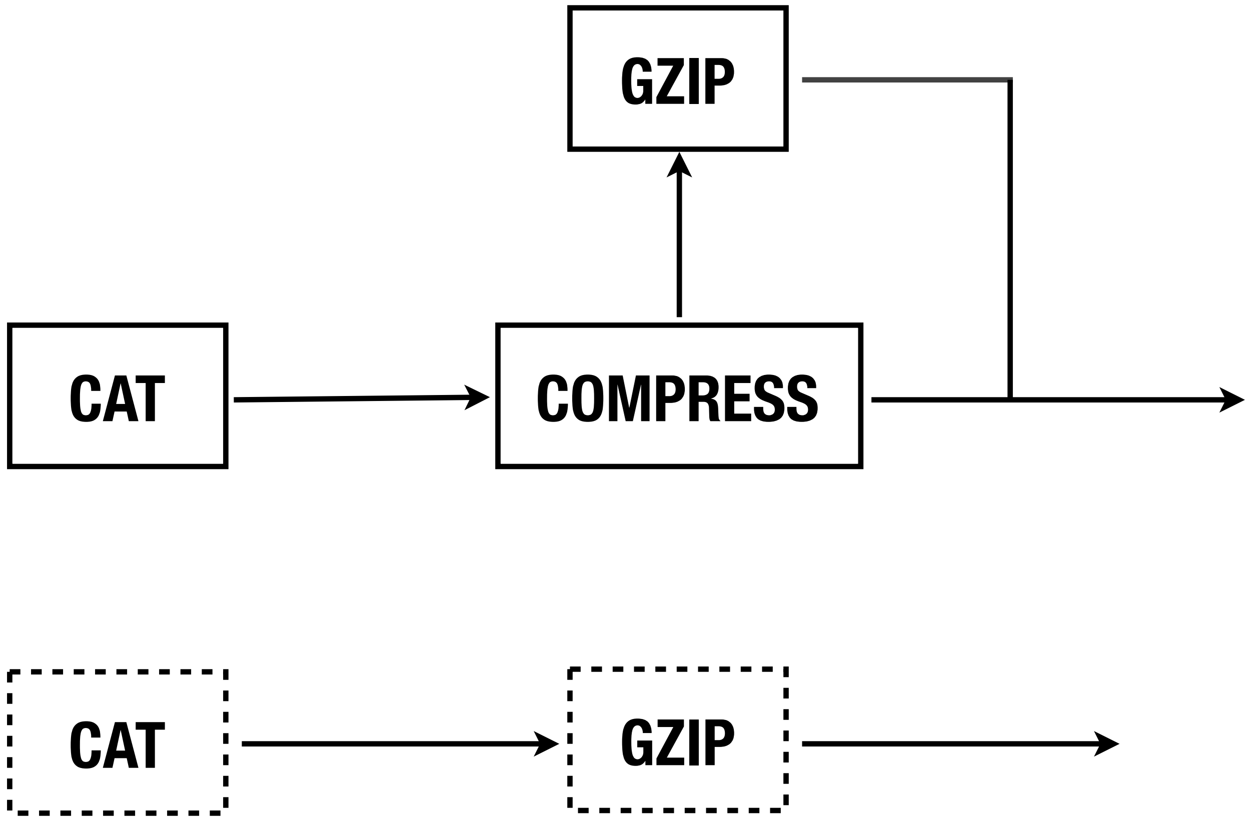
Handling a request



Handling a request



Handling a request



Source component

```go
func Cat(conn *Conn, writer io.WriteCloser) bool {
    conn.status = http.StatusOK
    conn.SetHeader("Content-type", "text/html; charset=utf-8")

    generate := func() {
        WriteFromFile(writer, "index.html")
        writer.Close()
    }

    go generate()
    return true
}
```

Source component

```
func Cat(conn *Conn, writer io.WriteCloser) bool {
    conn.status = http.StatusOK
    conn.SetHeader("Content-type", "text/html; charset=utf-8")

    generate := func() {
        WriteFromFile(writer, "index.html")
        writer.Close()
    }

    go generate()
    return true
}
```

Source component

```
func Cat(conn *Conn, writer io.WriteCloser) bool {
    conn.status = http.StatusOK
    conn.SetHeader("Content-type", "text/html; charset=utf-8")

    generate := func() {
        WriteFromFile(writer, "index.html")
        writer.Close()
    }

    go generate()
    return true
}
```

Source component

```
func Cat(conn *Conn, writer io.WriteCloser) bool {
    conn.status = http.StatusOK
    conn.SetHeader("Content-type", "text/html; charset=utf-8")

    generate := func() {
        WriteFromFile(writer, "index.html")
        writer.Close()
    }

    go generate()
    return true
}
```

Source component

```go
func Cat(conn *Conn, writer io.WriteCloser) bool {
    conn.status = http.StatusOK
    conn.SetHeader("Content-type", "text/html; charset=utf-8")

    generate := func() {
        WriteFromFile(writer, "index.html")
        writer.Close()
    }

    go generate()
    return true
}
```

Source component

```
func Cat(conn *Conn, writer io.WriteCloser) bool {
    conn.status = http.StatusOK
    conn.SetHeader("Content-type", "text/html; charset=utf-8")

    generate := func() {
        WriteFromFile(writer, "index.html")
        writer.Close()
    }

    go generate()
    return true
}
```

Source component

```
func Cat(conn *Conn, writer io.WriteCloser) bool {
    conn.status = http.StatusOK
    conn.SetHeader("Content-type", "text/html; charset=utf-8")

    generate := func() {
        WriteFromFile(writer, "index.html")
        writer.Close()
    }

    go generate()
    return true
}
```

Filter Component

```
func Gzip(c *Conn, rd io.ReadCloser, wr io.WriteCloser) bool {
     c.SetHeader("Content-Encoding", "gzip")

     transform := func() {
         WriteGzipped(wr, rd)
         rd.Close()
         wr.Close()
     }

     go transform()
     return true
}
```

Filter Component

```
func Gzip(c *Conn, rd io.ReadCloser, wr io.WriteCloser) bool {
    c.SetHeader("Content-Encoding", "gzip")

    transform := func() {
        WriteGzipped(wr, rd)
        rd.Close()
        wr.Close()
    }

    go transform()
    return true
}
```

Filter Component

```
func Gzip(c *Conn, rd io.ReadCloser, wr io.WriteCloser) bool {
     c.SetHeader("Content-Encoding", "gzip")

     transform := func() {
         WriteGzipped(wr, rd)
         rd.Close()
         wr.Close()
     }

     go transform()
     return true
}
```

Creating a webpipes server

```
func main() {
     http.Handle("/", webpipes.Chain(
         webpipes.FileServer("http-data", "/"),
         webpipes.CompressPipe,
         webpipes.OutputPipe,
     )
     http.Handle("/images", ...)

     server := http.Server{
         Addr: ":12345",
     }
     log.Printf("Starting test server on %s", server.Addr)
     err := server.ListenAndServe()
     if err != nil {
         log.Fatalf("Error: %s", err)
     }
}
```

Creating a webpipes server

```go
func main() {
	http.Handle("/", webpipes.Chain(
		webpipes.FileServer("http-data", "/"),
		webpipes.CompressPipe,
		webpipes.OutputPipe,
	)
	http.Handle("/images", ...)

	server := http.Server{
		Addr: ":12345",
	}
	log.Printf("Starting test server on %s", server.Addr)
	err := server.ListenAndServe()
	if err != nil {
		log.Fatalf("Error: %s", err)
	}
}
```

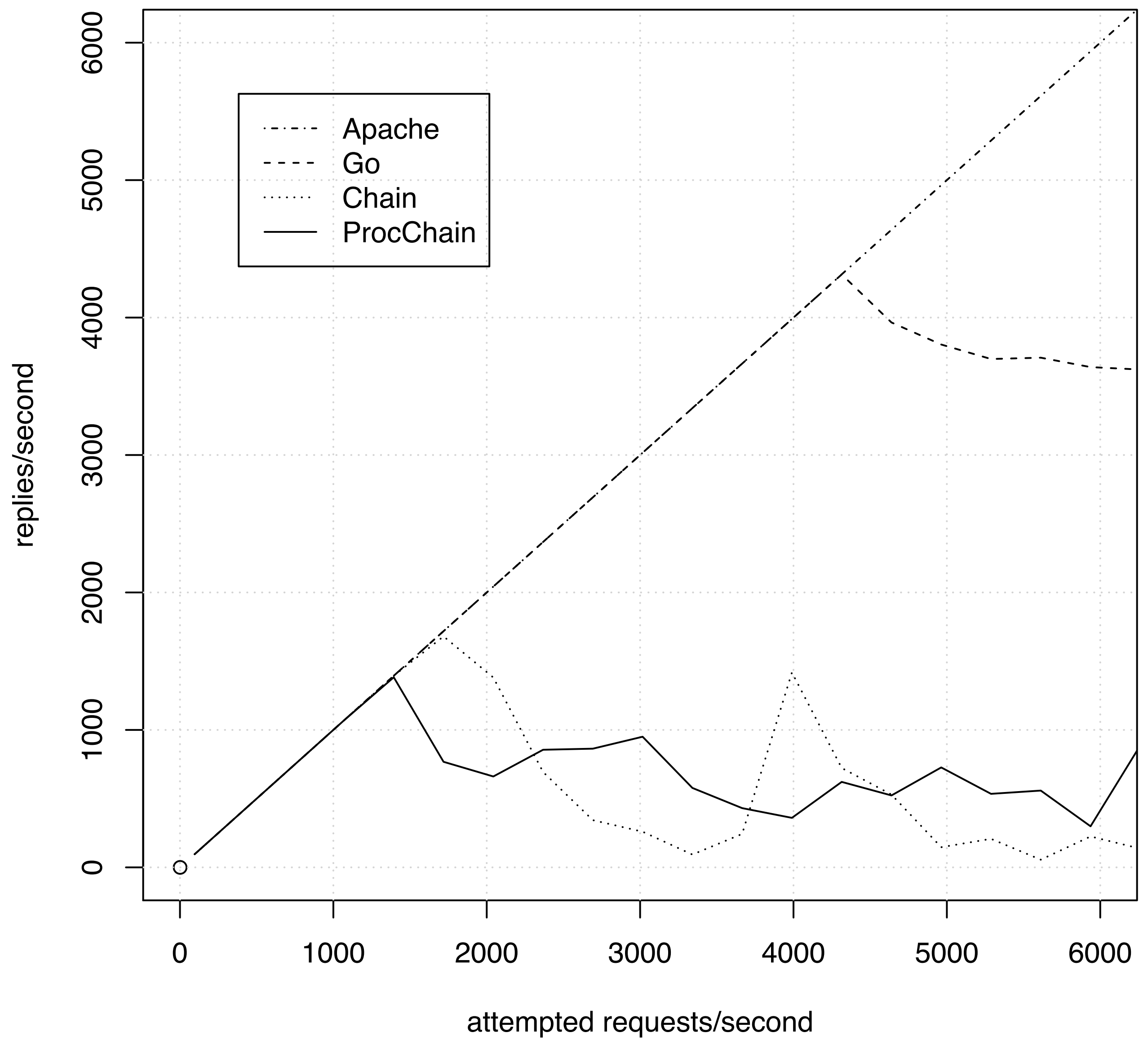
Creating a webpipes server

```
func main() {
    http.Handle("/", webpipes.Chain(
        webpipes.FileServer("http-data", "/"),
        webpipes.CompressPipe,
        webpipes.OutputPipe,
    )
    http.Handle("/images", ...)

    server := http.Server{
        Addr: ":12345",
    }
    log.Printf("Starting test server on %s", server.Addr)
    err := server.ListenAndServe()
    if err != nil {
        log.Fatalf("Error: %s", err)
    }
}
```

Creating a webpipes server

```
func main() {
	http.Handle("/", webpipes.Chain(
	    webpipes.FileServer("http-data", "/"),
	    webpipes.CompressPipe,
	    webpipes.OutputPipe,
	)
	http.Handle("/images", ...)

	server := http.Server{
	    Addr: ":12345",
	}
	log.Printf("Starting test server on %s", server.Addr)
	err := server.ListenAndServe()
	if err != nil {
	    log.Fatalf("Error: %s", err)
	}
}
```

Performance



Conclusions

- Unix pipelines for the web
- Flexible
- Understandable
- Details in the paper...