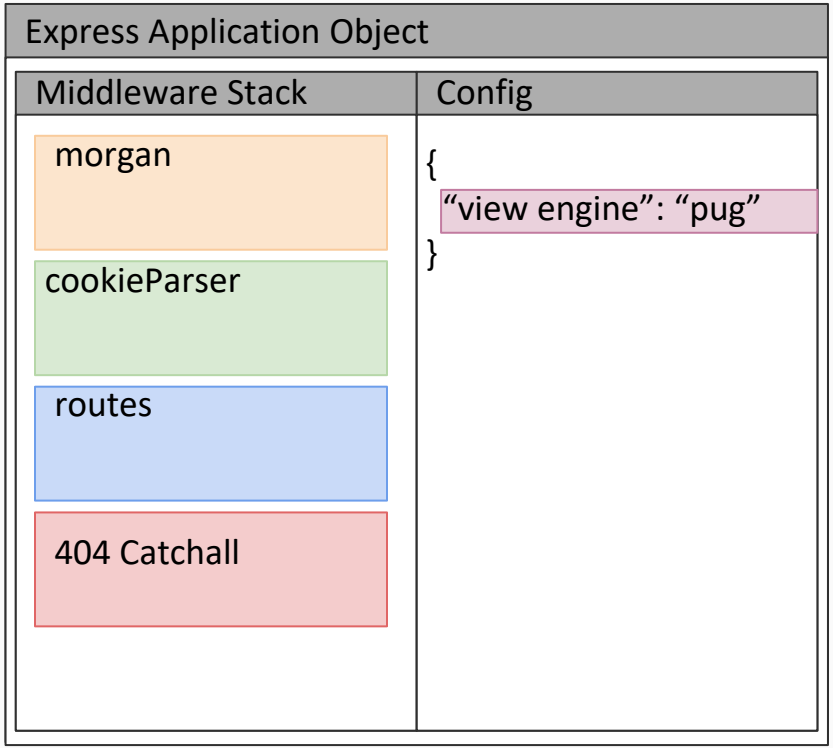
Express Middleware Pipeline

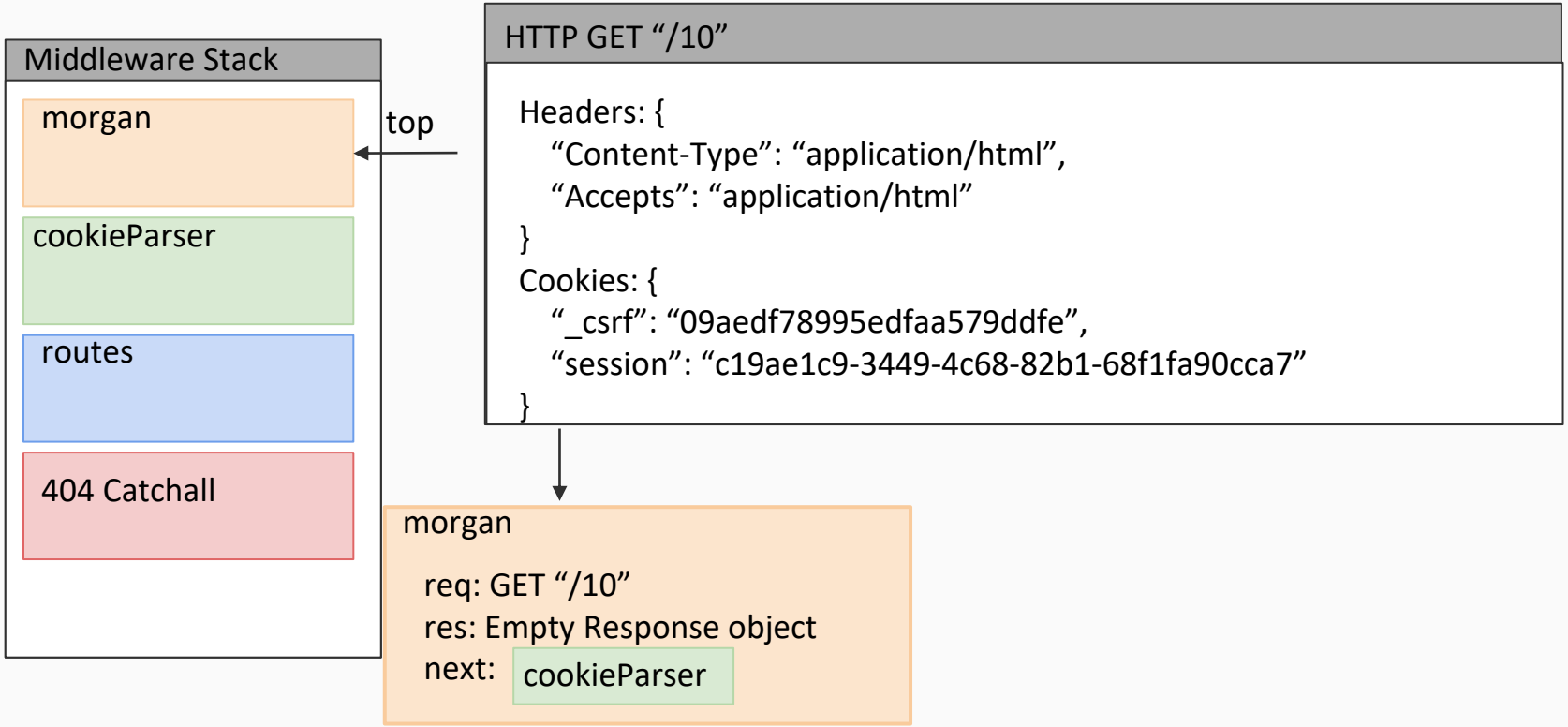
“Register” middleware

```
1  const express = require('express');
2  const morgan = require('morgan');
3
4  const routes = require('./routes');
5
6  const app = express();
7  app.set('view engine', 'pug');
8
9  app.use(morgan('dev'));
10 app.use(cookieParser());
11 app.use(routes);
12
13 app.use((req, res, next) => {
14 ——const err = new Error('The reque
15 ——err.status = 404;
16 ——next(err);
17 });
18
```



Wild HTTP Request appeared!

Invoke the Middleware Pipeline



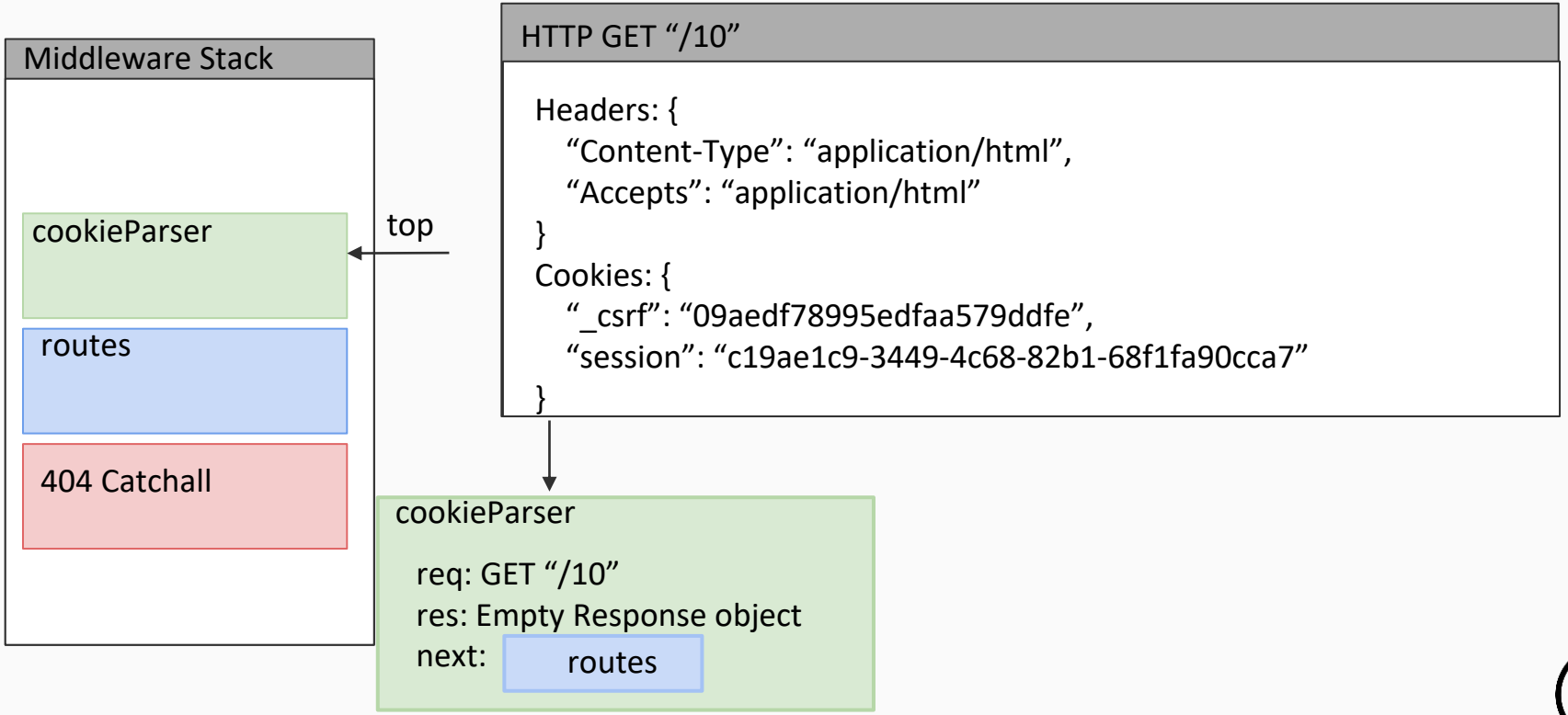
Morgan registers `logReqest` onFinished

morgan

req: GET “/10”
res: Empty Response object
next: cookieParser

```
133     if (immediate) {
134       // immediate log
135       logRequest()
136     } else {
137       // record response start
138       onHeaders(res, recordStartTime)
139
140       // log when response finished
141       onFinished(res, logRequest)
142     }
143
144     next()
145   }
```

next up is the cookieParser



cookieParser extracts cookies from the request

cookieParser

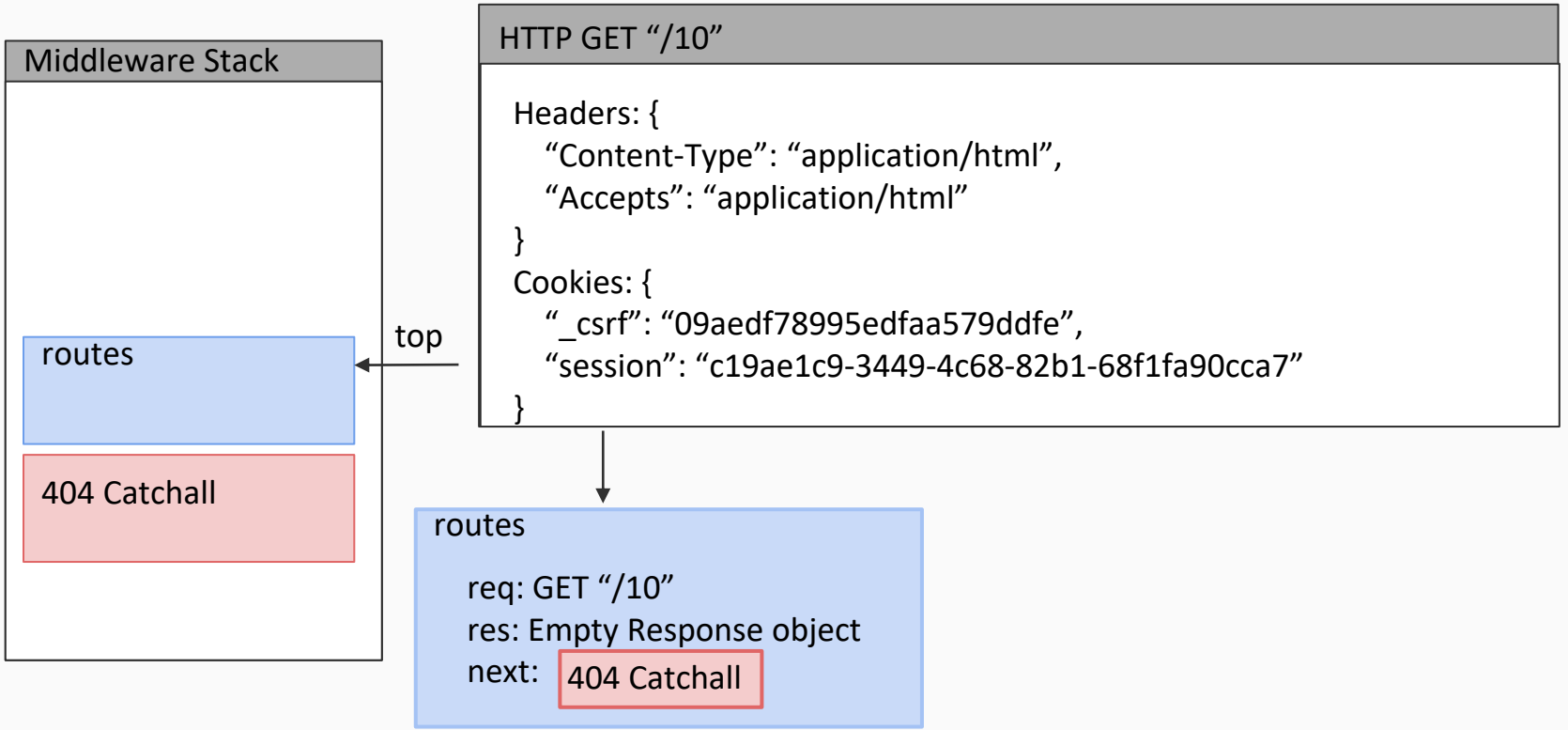
req: GET “/10”
res: Empty Response object
next: routes

```
return function cookieParser (req, res, next) {
  if (req.cookies) {
    return next()
  }

  var cookies = req.headers.cookie
```

```
// no cookies
if (!cookies) {
  return next()
}

req.cookies = cookie.parse(cookies, options)
```

```
// parse JSON cookies
req.cookies = JSONCookies(req.cookies)

next()
```

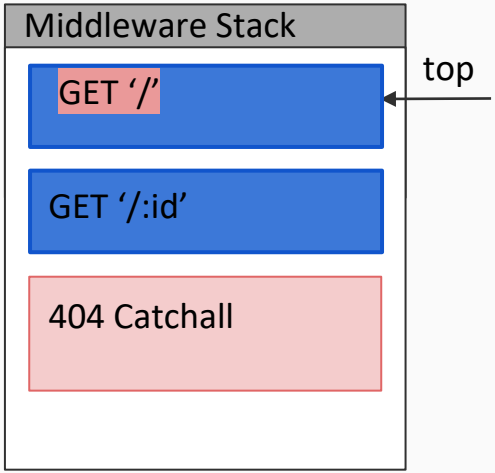
next up is the Router



Each route is pushed onto middleware stack

routes

req: GET “/10”
res: Empty Response object
next: 404 Catchall

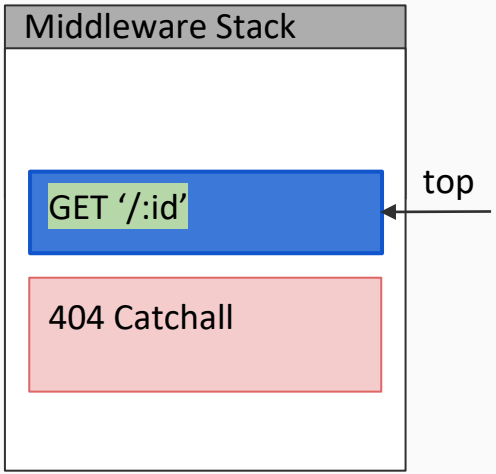



```
const csurf = require('csrf');
const csrfHandler = csurf({cookie: true});

const router = express.Router();

router.get('/', csrfHandler, async (req, res, next) => {
    try {
        const books = await db.Book.findAll({order: [['title', 'ASC']] });
        res.render('index', {title: 'Home', books });
    } catch (err) {
        next(err);
    }
});

router.get('/:id(\\d+)', csrfHandler, async (req, res, next) => {
    try {
        const book = await db.Book.find(req.body.id);
        res.render('book_view', {title: book.title, book });
    } catch (err) {
        next(err);
    }
});

module.exports = router;
```

Until a route is matched

routes

req: GET “/10”
res: Empty Response object
next: 404 Catchall



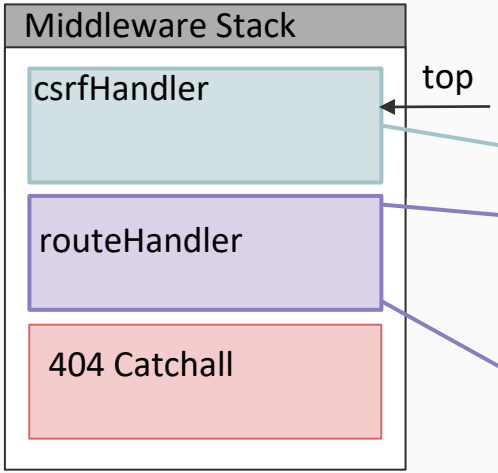

```
const csurf = require('csrf');
const csrfHandler = csurf({cookie: true});

const router = express.Router();

router.get('/', csrfHandler, async (req, res, next) => {
    try {
        const books = await db.Book.findAll({order: [['title', 'ASC']] });
        res.render('index', {title: 'Home', books });
    } catch (err) {
        next(err);
    }
});

router.get('/:id(\\d+)', csrfHandler, async (req, res, next) => {
    try {
        const book = await db.Book.find(req.body.id);
        res.render('book_view', {title: book.title, book });
    } catch (err) {
        next(err);
    }
});

module.exports = router;
```

The route handlers are pushed onto the stack

Route GET “/10”

req: GET “/10”
res: Empty Response object
next: csrfHandler

- Express adds each of the matched route’s handlers to the stack.
- They are added from the right to the left - outside-in.




```
router.get('/:id(\\d+)', csrfHandler, async (req, res, next) => {
    try {
        const book = await db.Book.find(req.body.id);
        res.render('book_view', {title: book.title, book });
    } catch (err) {
        next(err);
    }
});

module.exports = router;
```

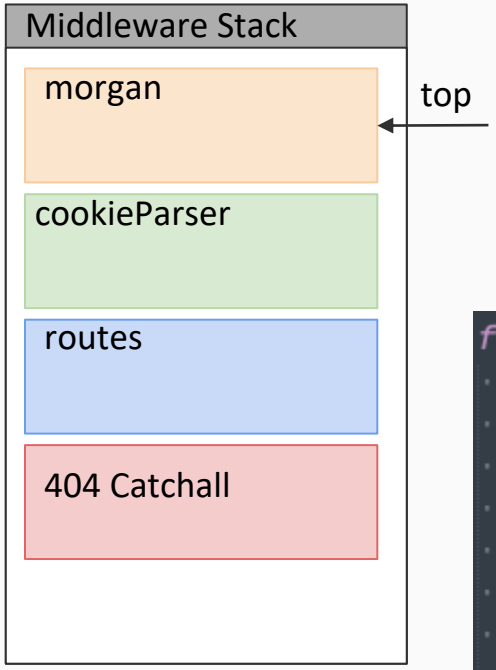
CSRF verifies the token

csrfHandler

req: GET “/10”
res: Empty Response object
next: routeHandler

```
// verify the incoming token
if (!ignoreMethod[req.method] && !tokens.verify(secret, value(req))) {
  return next(createError(403, 'invalid csrf token', {
    code: 'EBADCSRFTOKEN'
  }))
}

next()
```

Demystifying **next()**



next() will pop a middleware off of the appropriate stack

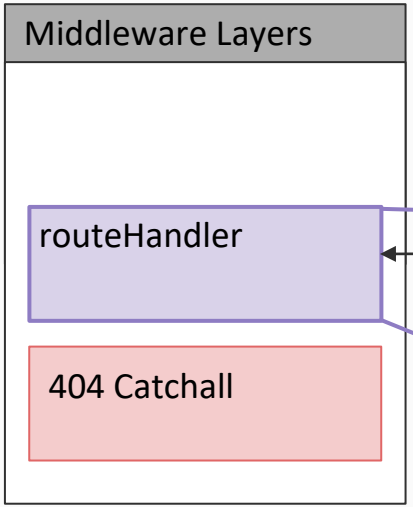
- The errorMiddleware stack if an error is present
- The middleware stack if not

```
function nextFactory(middleware, errorMiddleware) {
  return (err) => {
    if (err !== undefined) {
      const nextErrorMiddleware = errorMiddleware.pop();
      nextErrorMiddleware(err);
    } else {
      const nextMiddleware = middleware.pop();
      nextMiddleware();
  }
}
```

res.**send** short-circuits middleware

Route GET “/10”

req: GET “/10”
res: Empty Response object
next: 404 Catchall



top

```
router.get('/:id(\\d+)', csrfHandler, async (req, res, next) => {
    try {
        const book = await db.Book.find(req.body.id);
        res.render('book_view', {title: book.title, book });
    } catch (err) {
        next(err);
    }
});

module.exports = router;
```

Invocations of **res.send** and **res.render** short-circuit middleware

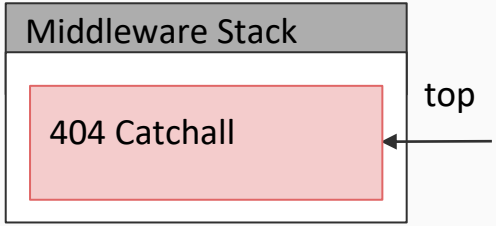
They **immediately** send a response to the browser, and complete the request

Usually, this is the only time that the response object is used

If none of the middleware match

404 Catchall

req: GET "**/profile**"
res: Empty Response object
next:




```
app.use((req, res, next) => {
    const err = new Error('The requested page couldn\'t be found.');
    err.status = 404;
    next(err);
});
```

- Invoke's next with an error
- Triggers the Error Middleware Stack