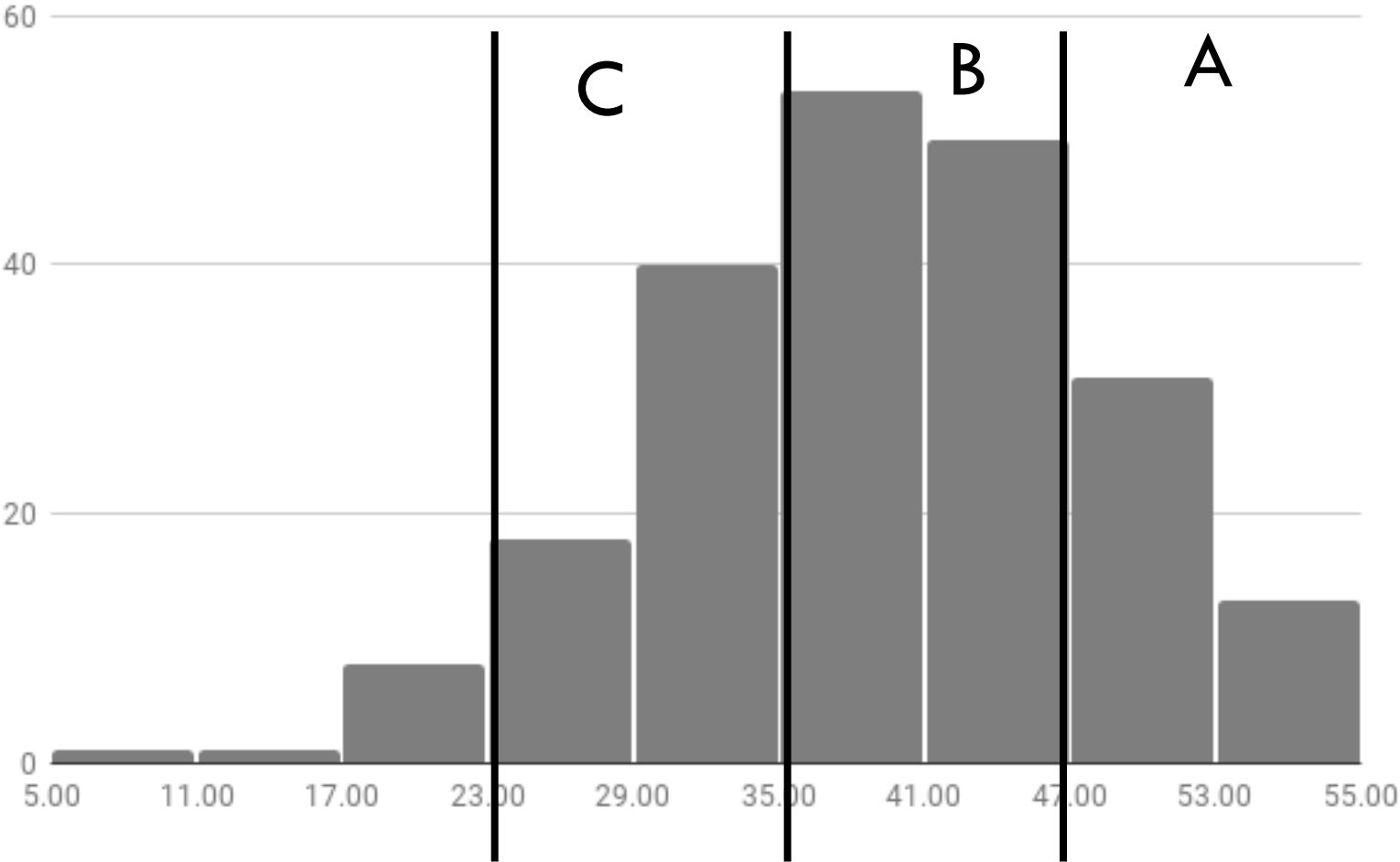


ECS 162

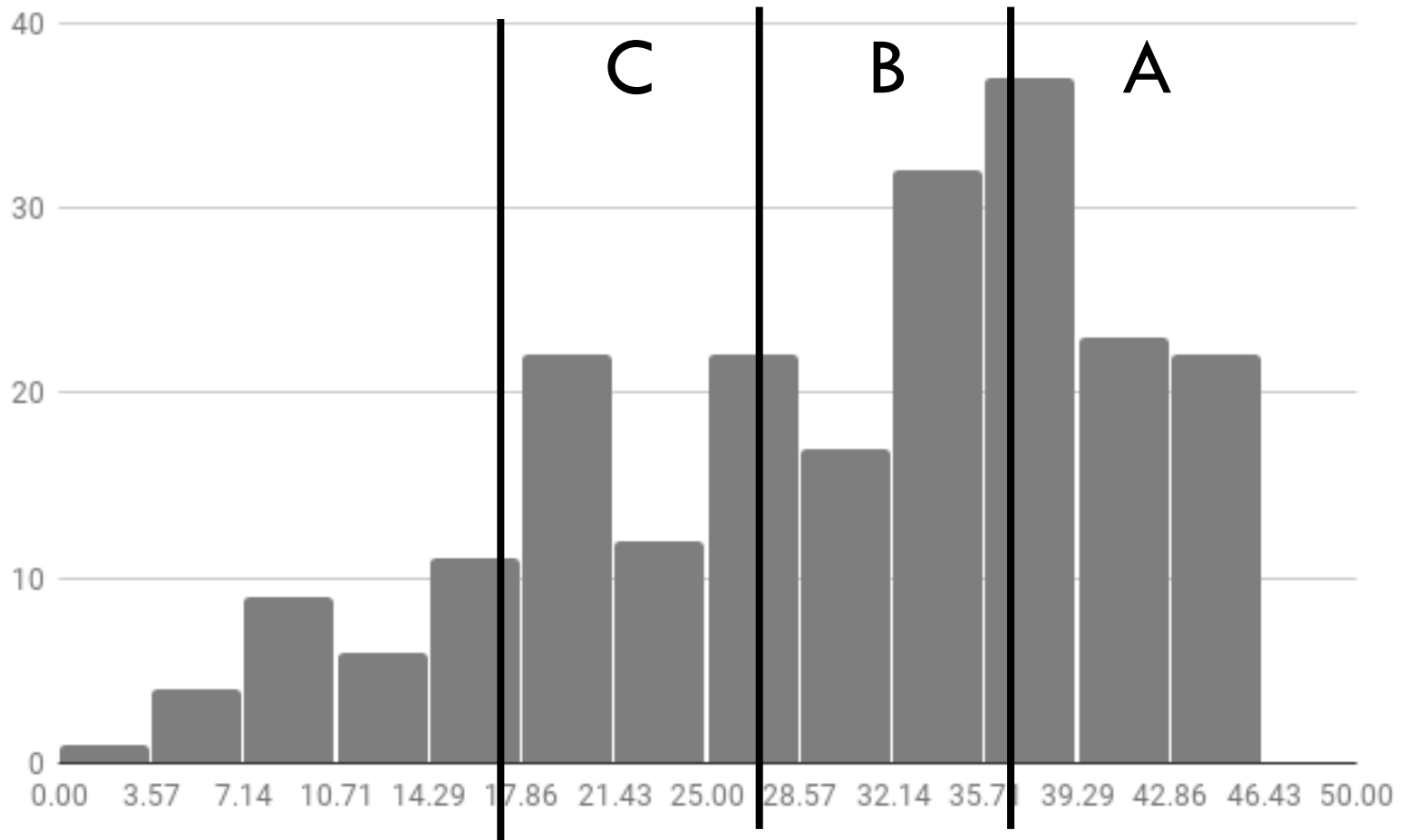
WEB PROGRAMMING

5/22

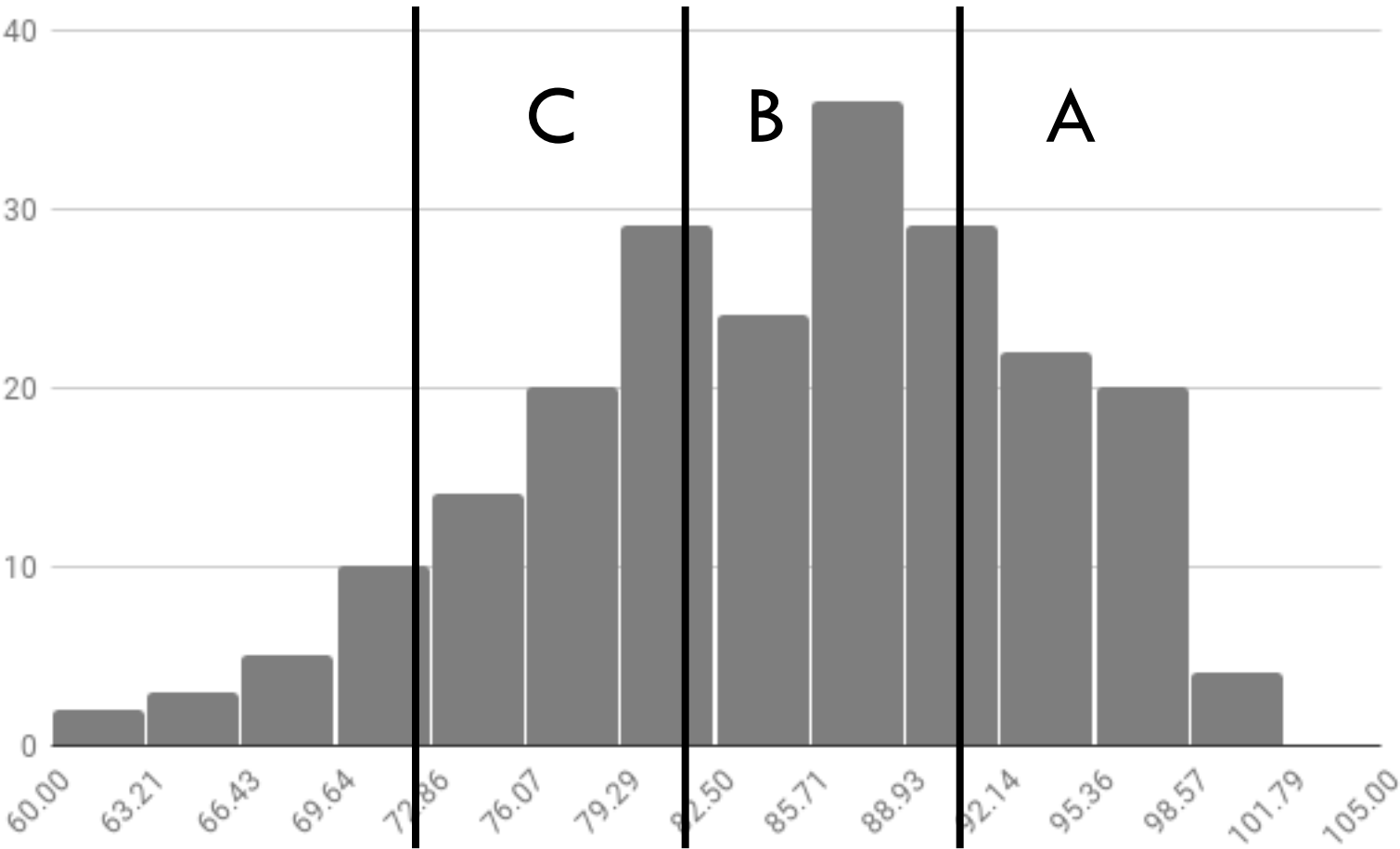
Midterm - MC



Midterm - Programming



Midterm - Combined



What does “logged in” mean?
















- Before we work on the login procedure, we need to think about what it means to be logged in.
- How does the system remember that we are logged in?
- How does it use this information to handle requests?
- Let's start with something that seems irrelevant...

Cookies

- Recall that the Server has no memory.
- Anything the Server does want to remember about an interaction he sends to the Browser, as a cookie.
- The Browser sends the cookie back to the Server with every subsequent HTTP request, for some specific period of time, or maybe forever.
- Mostly this happens without user intervention, although you can go into your browser and delete cookies.

Some of my Chrome cookies



 mail.google.com Cache Storage, 3 cookies, Database storage, Local storage, Service Workers, Database storage,		
 server162.site 2 cookies, Database storage		
 uk.businessinsider.com File system		
 unsplash.com 1 cookie		
 web.cs.ucdavis.edu Database storage		

Sessions

- Consider a Web app with a login page.
- The Server has to remember that the user is logged in, so that he can give him access to his flashcards, or bank account, or whatever.
- This is an exception to the usual rule that the Server has no memory.
- Also, he needs to recognize HTTP requests from different users, and give each one their own data.

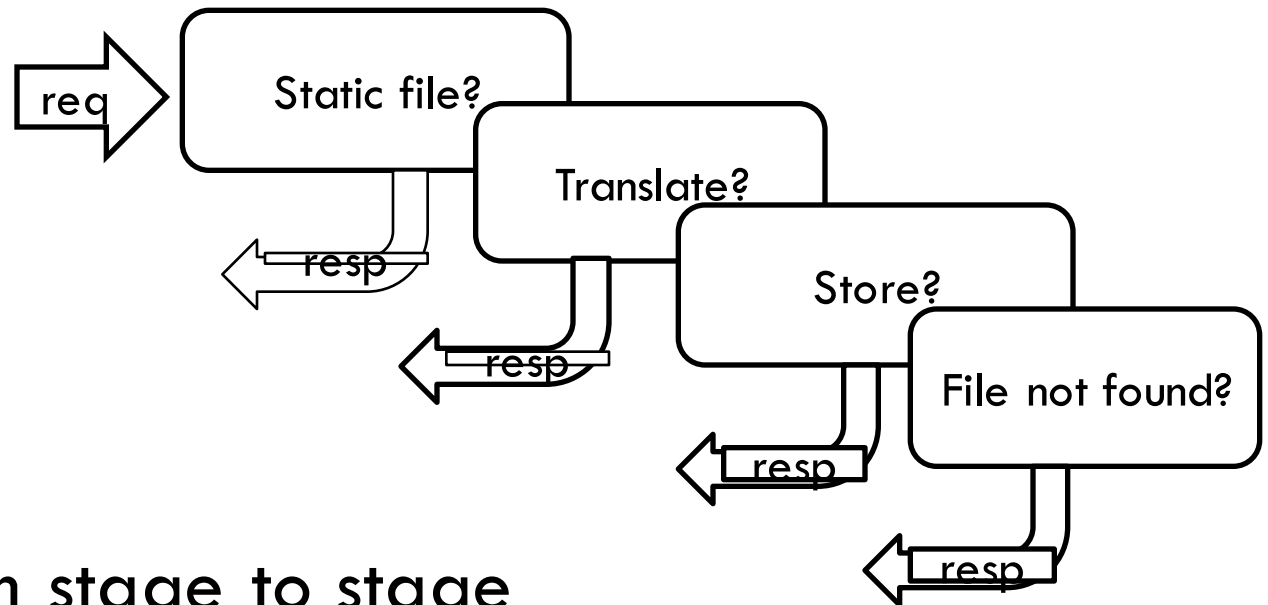
Sessions

- The Server sends a session cookie (aka session id, session token) to the Browser, which is a big random number.
- The Browser includes the session cookie in all subsequent HTTP requests, just like she would with any other cookie.
- The Server sees the cookie and looks up the user.
- The Server provides user data in an HTTP response only if the cookie is found.

Sessions

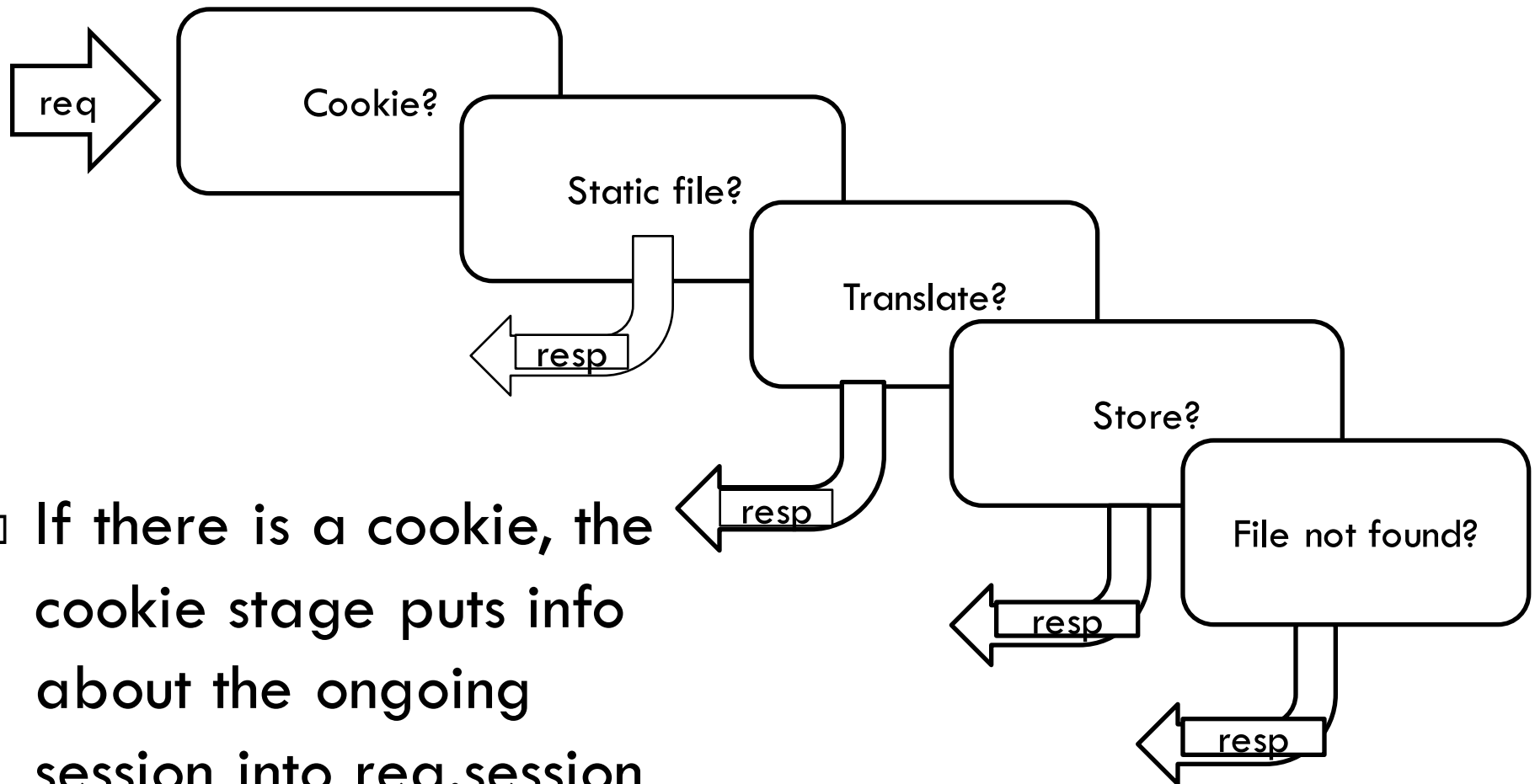
- This is what it means to be logged into a Web site.
- When you log off, the Server forgets your session cookie.
- Users rarely log out of Web sites, they just wander off, or close their browser.
- The Server forgets the session key after a while, maybe a couple of hours.
- Browsers may or may not forget session keys when you close tabs or close the browser.

In Express Server Pipeline



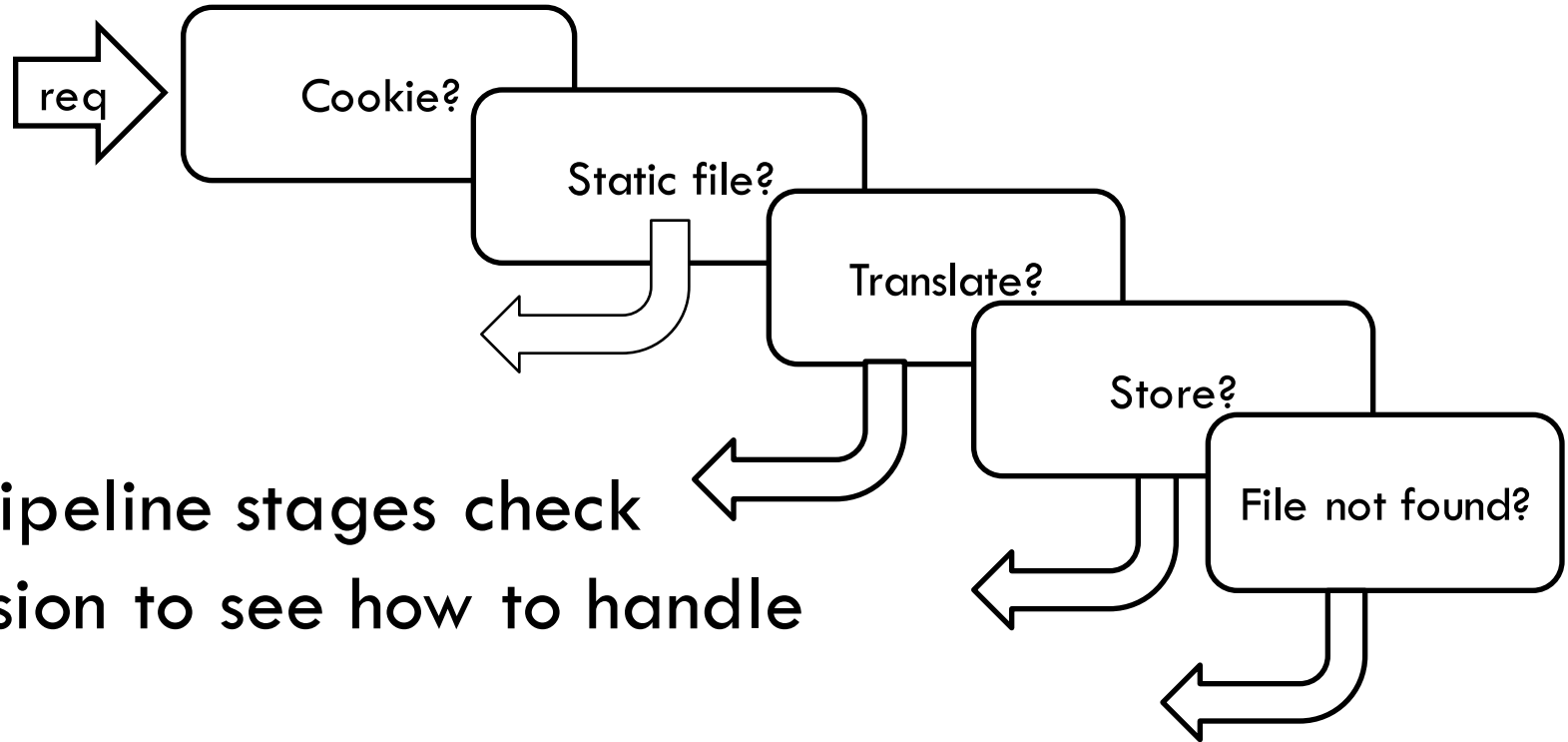
- Pipeline passes (req, res) pair from stage to stage until response gets sent.
- Additional information can be put into in (req, res) as we go along.

In Express Server Pipeline



- If there is a cookie, the cookie stage puts info about the ongoing session into req.session

In Express Server Pipeline



- Later pipeline stages check `req.session` to see how to handle request
- For instance, static server might not serve some files to a user who is not logged in

Implementation in Express

- We use two new modules:

```
const express = require('express');
```

```
const passport = require('passport');
```

```
const cookieSession = require('cookie-session');
```

- Passport will handle login process but also sets up and stays involved in session.
- Cookie-session sends cookies to Browser, checks them on every incoming HTTP request.

Cookie pipeline stage

- Right at beginning of pipeline

```
app.use(cookieSession({  
  maxAge: 6 * 60 * 60 * 1000,  
  keys: ['snickerdoodle']  
}));
```

- maxAge is how long the Server will remember the cookie, in ms. So this is how long?

Cookie pipeline stage

- Right at beginning of pipeline

```
app.use(cookieSession({  
  maxAge: 6 * 60 * 60 * 1000,  
  keys: ['snickerdoodle']  
}));
```

- maxAge is how long the Server will remember the cookie, in ms. So this is how long?

$1000\text{ms}/\text{sec}, 6*60*60 \text{ sec} = 6 \text{ hrs}$

Cookie pipeline stage

- Right at beginning of pipeline

```
app.use(cookieSession({  
  maxAge: 6 * 60 * 60 * 1000,  
  keys: ['snickerdoodle']  
}));
```

- keys is a list of random keys used to cryptographically sign session cookies.

Encryption and session cookies

- Session id is:

“session1 ”+hash(“session1 ”+“snickerdoodle”)

where hash() is SHA1 encryption

- Server checks that hash(“session1 ”+“snickerdoodle”)
is really the second half of the returned cookie. This
at least shows that it is one of it’s own cookies (since
somebody trying to fake it does not know the key
“snickerdoodle”).
- It is way better to use a real random string instead
of “snickerdoodle”; why?

Passport session stages

```
app.use(passport.initialize());
```

```
app.use(passport.session());
```

- Passport helps us set up data in the req object that later pipeline stages can use.
- Example: if the request was from a valid session, show the card creation page, otherwise redirect to login.
- Example: return the user name provided by Google in response to an AJAX query.

Passport session stages

```
app.use(passport.initialize());
```

```
app.use(passport.session());
```

- The first one initializes req somehow for further Passport stages
- The second one attaches user information to req, in req.user. It calls a function deserializeUser, which we get to write. deserializeUser can, for instance, take information out of an sqlite3 User database table, based on an input userID.

Using session info

- Middleware to check if user is logged in, if not, redirect to login:

```
function isAuthenticated(req, res, next) {  
  if (req.user) {  
    console.log("Authenticated user", req.user);  
    next();  
  } else { res.redirect('/login.html'); }  
}
```

Using session information in req obj

- How would we teach our Server to answer an AJAX query for the user's name, eg.

`server162.site:[port]/query/name=true`

- One approach: have the passport session pipeline stage put user's name in `req.user.name` (it can get the user's name from their Google profile when they log in).
- Then, add a new middleware stage that answers the query by returning `req.user.name` in the json body of the response.