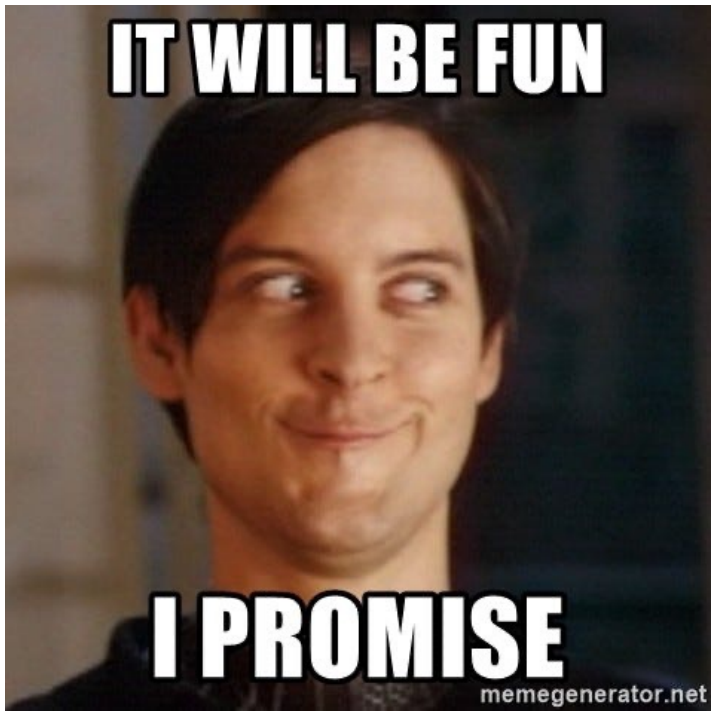


Async and Auth

CSC309

Kian Abbasi

UNIVERSITY OF TORONTO MISSISSAUGA

Winter 2025

So far

- Next.js API handlers
- MVC and model design
- Prisma ORM and CRUD

This session

- Async programming
  - Event loop and promises
- Authentication and authorization
- Tokens and sessions

API Handlers

- API handlers can do sophisticated works
  - Read from/write into the database
  - Make requests to other servers/APIs
  - File operations

- These operations could potentially be very slow
  - Compared to the simple object manipulation logic

How to optimize

- We need to exactly identify what causes the handler to be slow
  - Is it complex CPU processing? Or I/O waits?

- In computer science, there is two types of tasks:
  - I/O bound
  - CPU bound

I/O bound vs CPU bound

Visit https://softwareg.com.au/blogs/computer-hardware/io-bound-vs-cpu-bound-examples

- I/O bound
  - Has small bursts of CPU activity and then waits for I/O
  - eg. Word processor
  - Affects user interaction (we want these processes to have highest priority)

- CPU bound
  - Hardly any I/O, mostly CPU activity (eg. gcc, scientific modeling, 3D rendering, etc)
    - Useful to have long CPU bursts
  - Could do with lower priorities

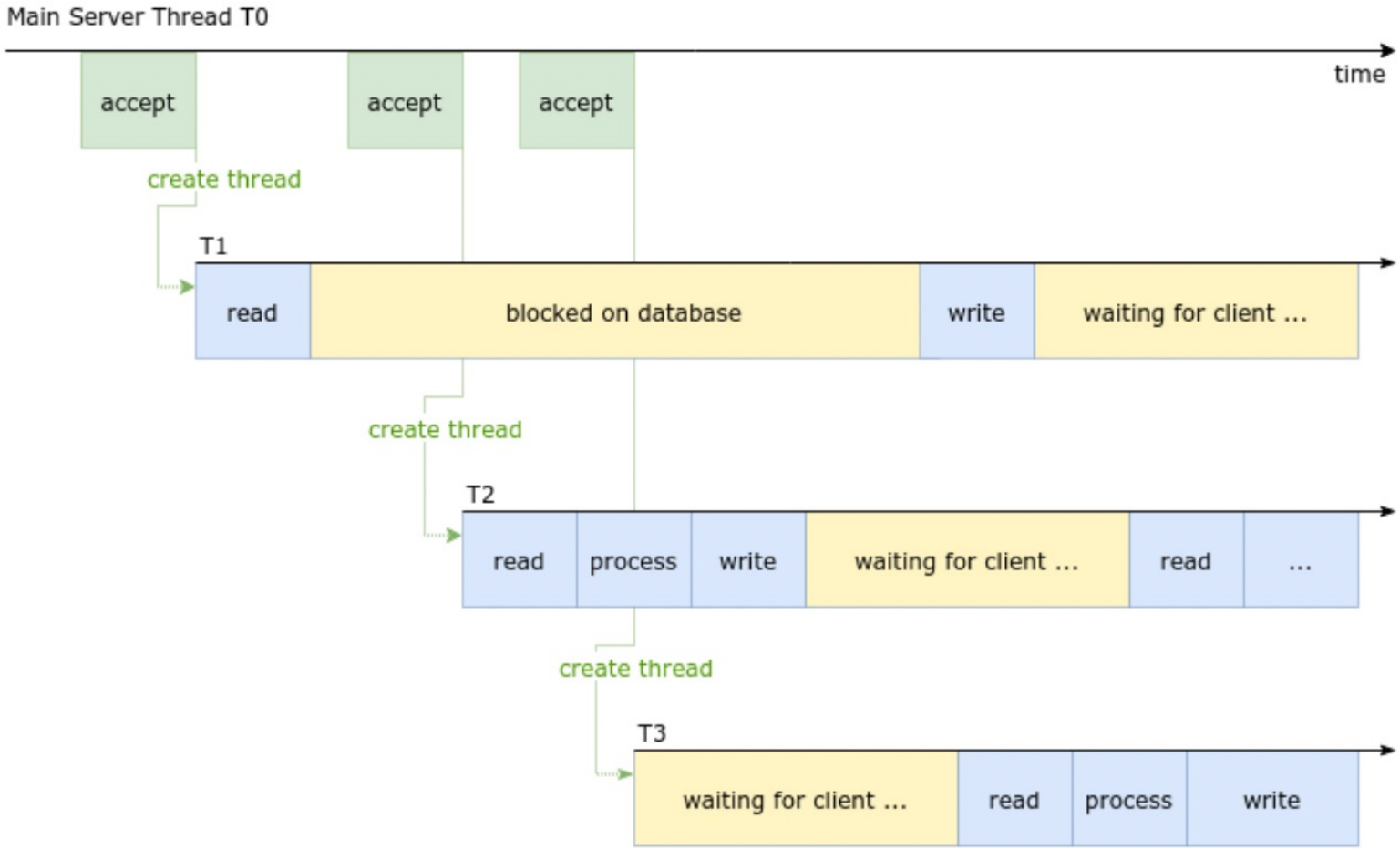
Source: http://www.cse.iitm.ac.in/~chester/courses/16o_os/slides/7_Scheduling.pdf

Optimization

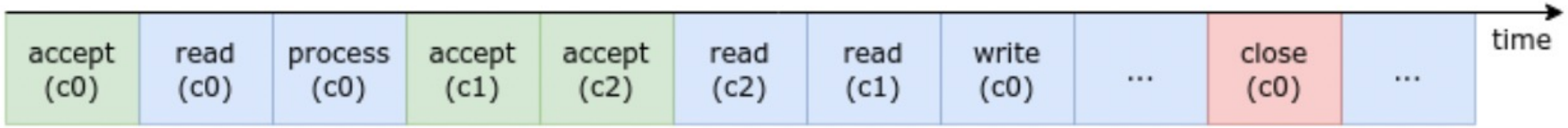
- CPU bound tasks could speed up with multi-threading
  - More CPU power -> process finishes sooner
- What about I/O bound ones?
  - More threads -> more idle threads -> more waste of resource
- Are API handlers I/O bound, or CPU bound?

Web server architecture

Visit https://levelup.gitconnected.com/event-driven-servers-a-intuitive-study-6d1677818d2a



Event loop



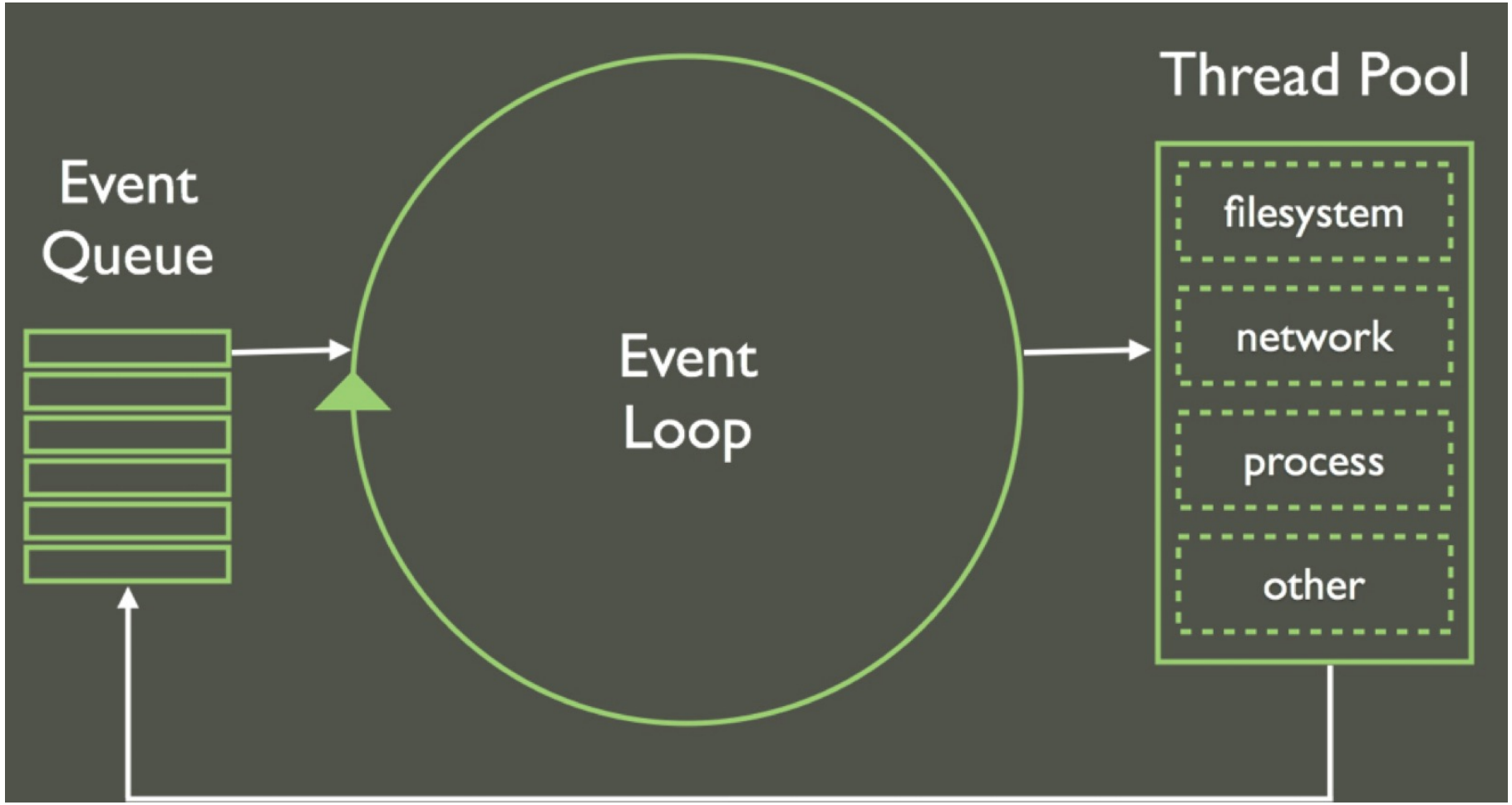
- A smart way to do more work with the same CPU power!
- Take control from the idling task and give to another task that needs it now!
- All done in just one thread!

Event loop logic (simplified)

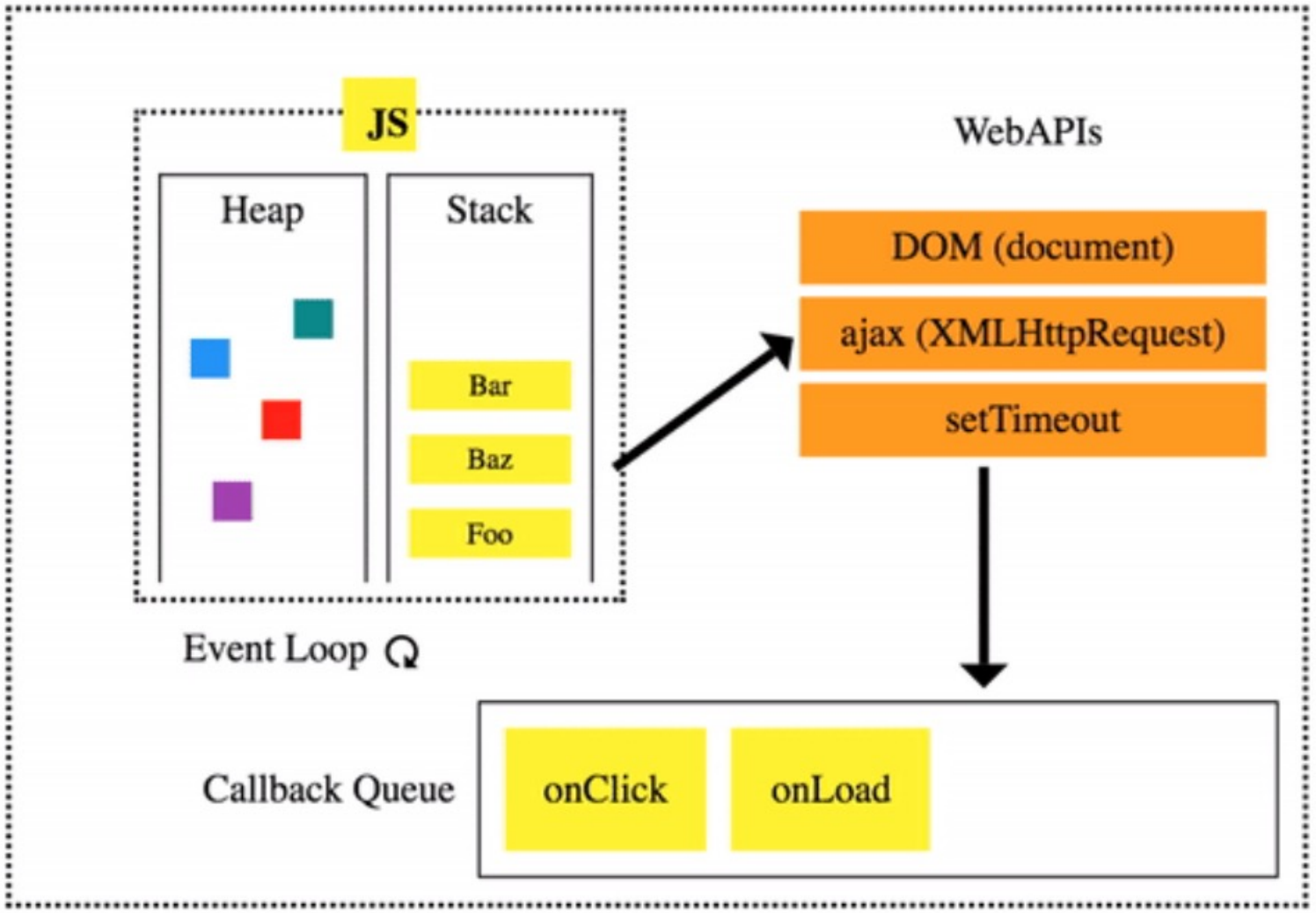
```
function event_loop():
  while true:
    # Get the first task in the queue
    current_task = task_queue.pop(0)

    # Execute the task IN THE SAME THREAD
    result = execute(current_task)

    if result is not complete:
      # If it's still blocked by I/O, or is blocked by
      # a different I/O task, push it to the end of the queue
      task_queue.append(current_task)
```

Event loop logic

Visit https://www.youtube.com/watch?v=zphcsoSJMvM



Event Loop



Source: https://medium.com/@Rahulx1/understanding-event-loop-call-stack-event-job-queue-in-javascript-63dcd2c71ecd

Async programming

- Not natively supported by many languages
  - C, C++, Java, Python (until 3.4)

- Workarounds
  - Callbacks
  - Promises

Callback hell!

Visit http://callbackhell.com

```
fs.readdir(source, function (err, files) {
  if (err) {
    console.log('Error finding files: ' + err)
  } else {
    files.forEach(function (filename, fileIndex) {
      console.log(filename)
      gm(source + filename).size(function (err, values) {
        if (err) {
          console.log('Error identifying file size: ' + err)
        } else {
          console.log(filename + ' : ' + values)
          aspect = (values.width / values.height)
          widths.forEach(function (width, widthIndex) {
            height = Math.round(width / aspect)
            console.log('resizing ' + filename + 'to ' + height + 'x' + height)
            this.resize(width, height).write(dest + 'w' + width + '_' + filename, function(err) {
              if (err) console.log('Error writing file: ' + err)
            })
          }.bind(this))
        }
      })
    })
  }
})
```

Promises

- Example:

```
callExternalAPI(...)
  .then(result => readFromDb(...))
  .then(result => writeIntoDb(...))
  .then(result => produceResponse(...))
  .catch(failureCallback)
```

- Code does not get nested like callbacks
  - But all subsequent logic (even sync) will be in `then` clauses

Async programming

- Async functions
  - Available in JavaScript, Python, Go, …
- The exact same flow of code as in sync programming
  - At every I/O blocking task, put `await`
  - The rest is handled by the interpreter, event loop, etc.
- Life could not be easier!

Async programming in JavaScript

- Example

```
async function handler(req, res) {
  try{
    const apiResponse = await callExternalAPI(...)
    const readResponse = await readFromDb(...)
    const writeResponse = await writeIntoDb(...)

    // produce and return result
  } catch (error) {
    // failure callback
  }
}
```

Auth

Authentication vs Authorization

- Authentication:
  - + Who's calling?
  - - This is Daniel Liu
  - + Is it really Daniel Liu?
- Obtains user information from user/pass, session, API key, fingerprints, etc.

- Authorization:
  - Does Daniel Liu have enough access and permissions (aka authorized) to make this request?
- Checks user's properties and permissions

Authentication

- Client should tell us who they are
- Through request headers
- Several authentication methods
  - Basic auth
  - Session auth
  - Token auth

Basic auth

- Simply sends credentials at every request
  - User/pass, fingerprints, face ID, etc.
- No concept of login and logout
- So risky: transfers raw sensitive data many times
  - If compromised, huge damage is incurred
- Not used in modern applications

Session auth

- Client sends user/pass at login
  - Or fingerprints, face ID, etc.
- If successful, server creates and stores a session id
  - Mapped to user
- Session id returned in the response
  - Browser saves it in cookies
- Browsers sends the same session id at next requests
  - Incognito tab: browser does not send the same session id

Token auth

Visit: https://www.javainuse.com/jwtgenerator

- Instead of a random session id, the token can contain information about the user

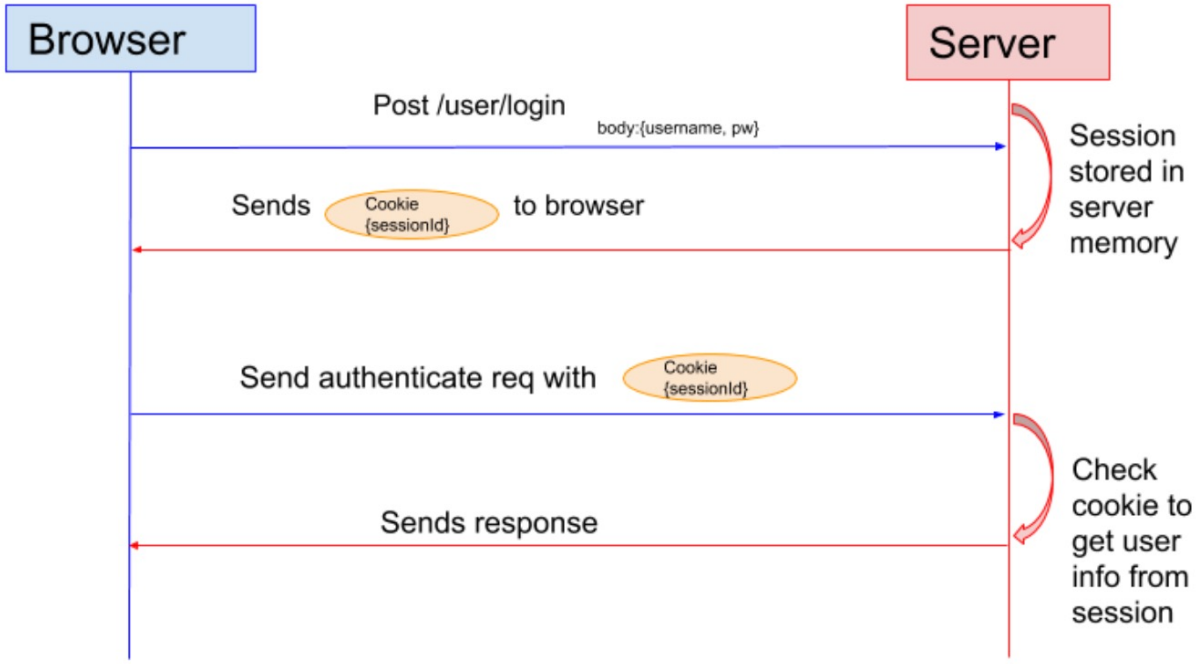
- It can be a JSON string

```
{ "userId": "134234", "expiresAt": 1722720863 }
```

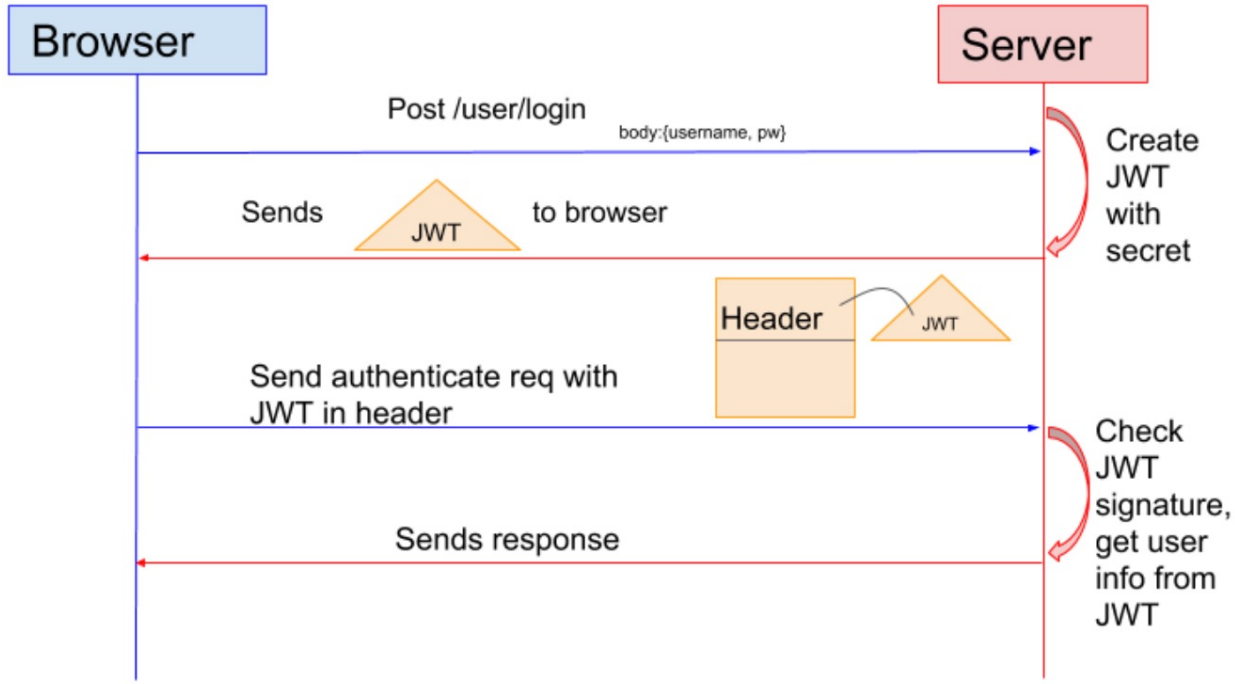
- Must be signed by the server to avoid attacks
  - Turned into a seemingly random string

```
eyJhbGciOiJIUzI1NiJ9.eyJ1c2VySWQiOiIxMjM0IiwiZXhwaXJlc0F0IjoiMTcyMjcyMDg2MyJ9.UsTi2eDC5hrI1uqv-JzUf384g0QznPZomPfzJbdnMtY
```

Session auth



Token auth



Source: https://sherryhsu.medium.com/session-vs-token-based-authentication-11a6c5ac45e4

Session vs token auth

Session auth

- Less scalable
  - server stores all sessions
  - An additional database query per request
- More control
  - server can revoke a session

Token auth

- Simpler
  - No database interaction
- More scalable
  - Client in charge of storing the token
- Less control
  - Not revocable. True logout is not possible

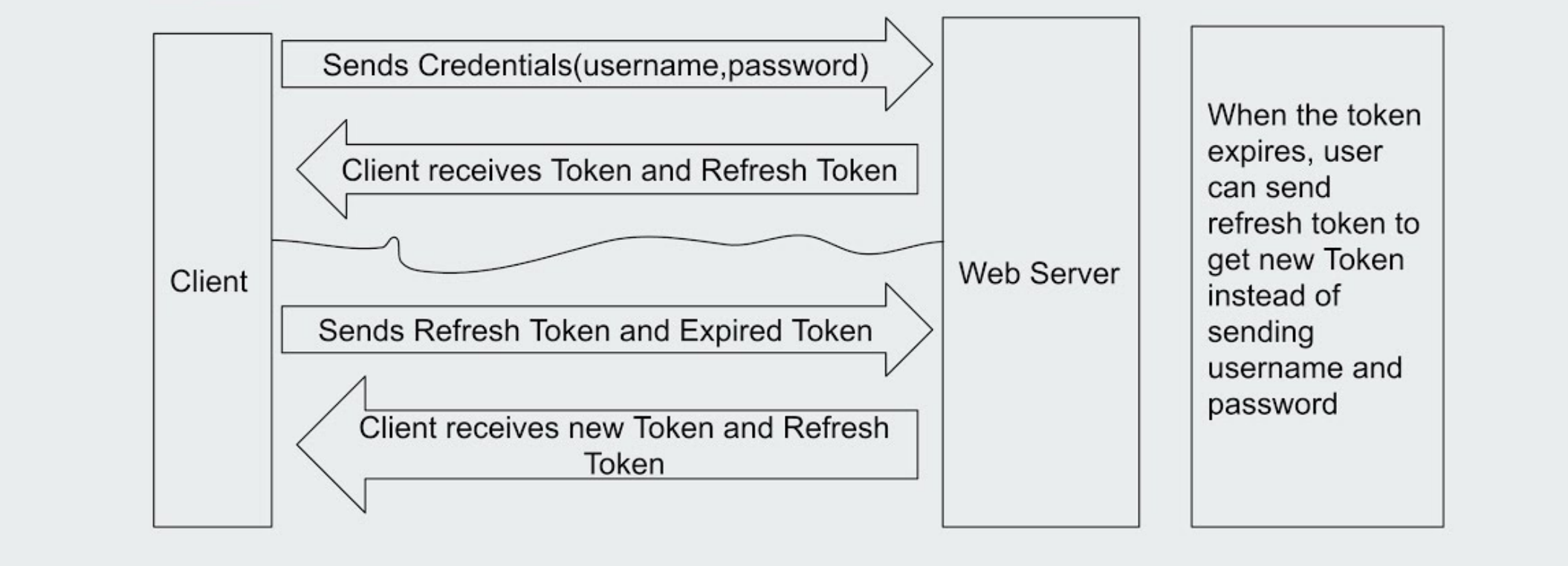
Best practices

- Token auth is preferred in modern web apps
  - Because of simplicity and scalability
- Known as JSON Web Token (JWT)
- They are generally very safe
  - Constructing a counterfeit token is practically impossible
- The main risk: compromised tokens
  - Tokens are not revocable: They should not be sent over and over

Best practices

- Short-lived tokens
  - Access tokens should expire within 15 minutes to an hour
- Having user authenticate every hour is a very bad UX
- Refresh tokens
  - Signed using a different secret
  - Can only be used to generate a new access token

Refresh tokens



Source: https://www.youtube.com/watch?v=yadjfgDBSiM&themeRefresh=1

Refresh tokens

- Refresh tokens last much longer
  - From hours to days or even weeks
- Upon receiving a 401 Unauthorized response:
  - Try refreshing the token first
  - Resend the request with the new access token
- Session continuity
  - User only re-authenticates when refresh token expires

Exercise: JWT auth in Next.js

Authorization

- Often, you should check several conditions before executing the API handler logic
  - Is the user authenticated?
  - Does the user have enough access?
    - e.g., being the owner of the store, or a follower of the author
- Return a 403 Unauthorized in those cases
- Should be re-usable logic, ideally separate from the handler logic
  - Often in middlewares

Next session

- Migrations
- Workflow and assumptions
- Conflict resolution