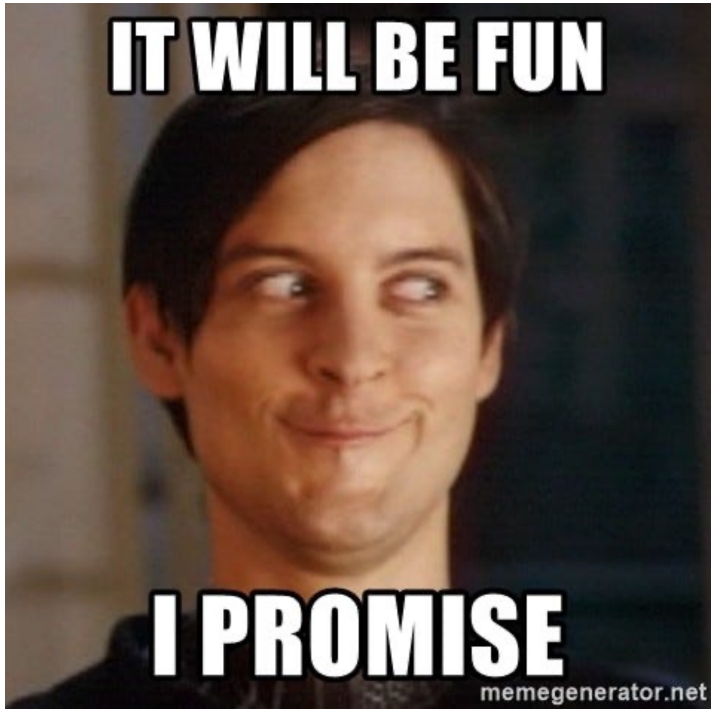


Async, Models and ORM

CSC309

Kianoosh Abbasi

UNIVERSITY OF TORONTO

Fall 2024

So far

- Modern architecture of web apps
    - Frontend & backend
    - APIs

- Server-side JavaScript
    - JS projects with Node

- Next.js API handlers

This session

- Async programming
  - Event loop and promises

- Data management
  - Model design
  - The MVC design pattern

- ORMs

API Handlers

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

- These are potentially very slow
  - Compared to the rest of the handler's job
    - Which is mostly simple object manipulation logic

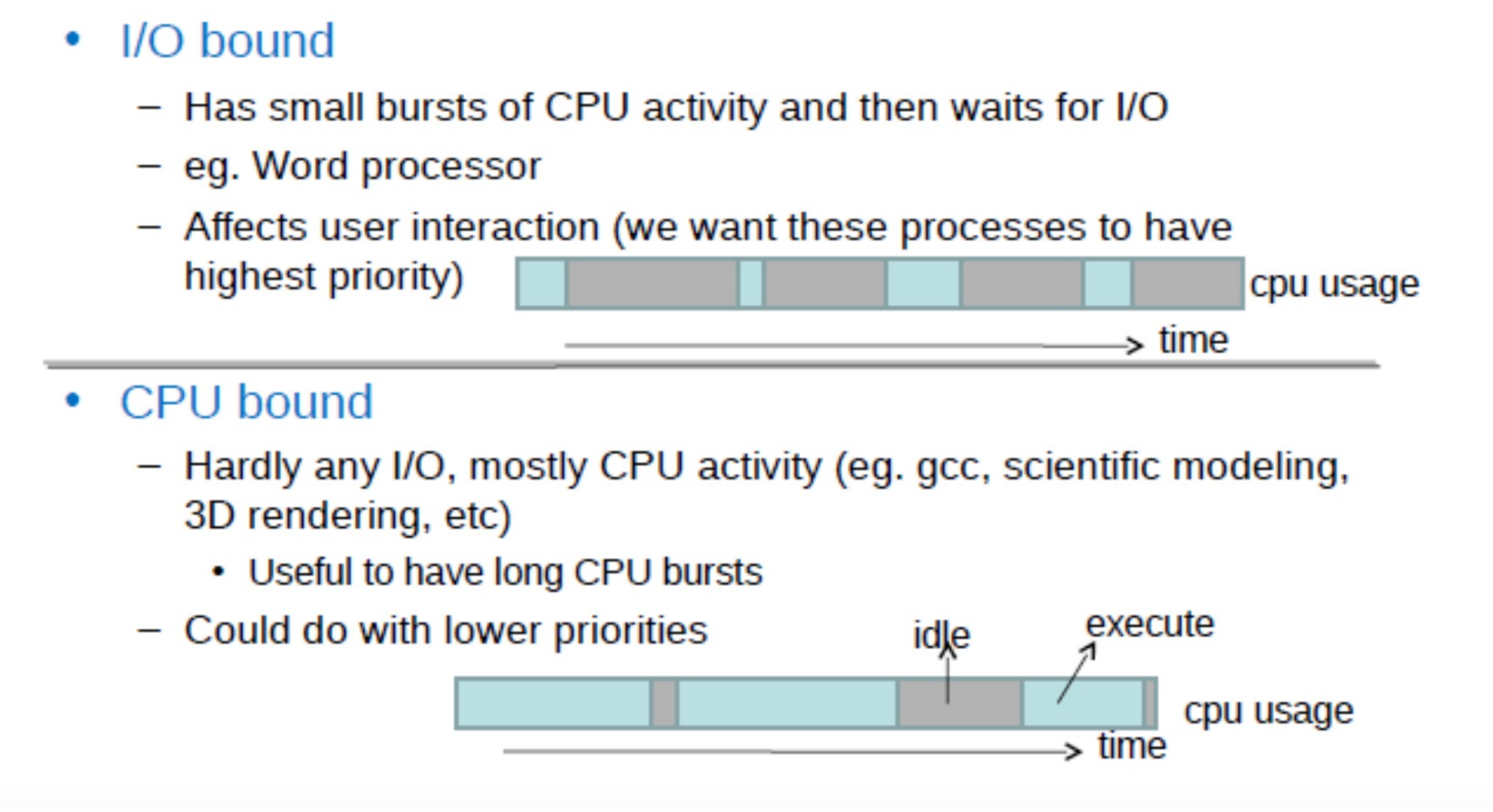
How to optimize

- 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 usage
time

- 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

idle
execute
cpu usage
time

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

Optimization

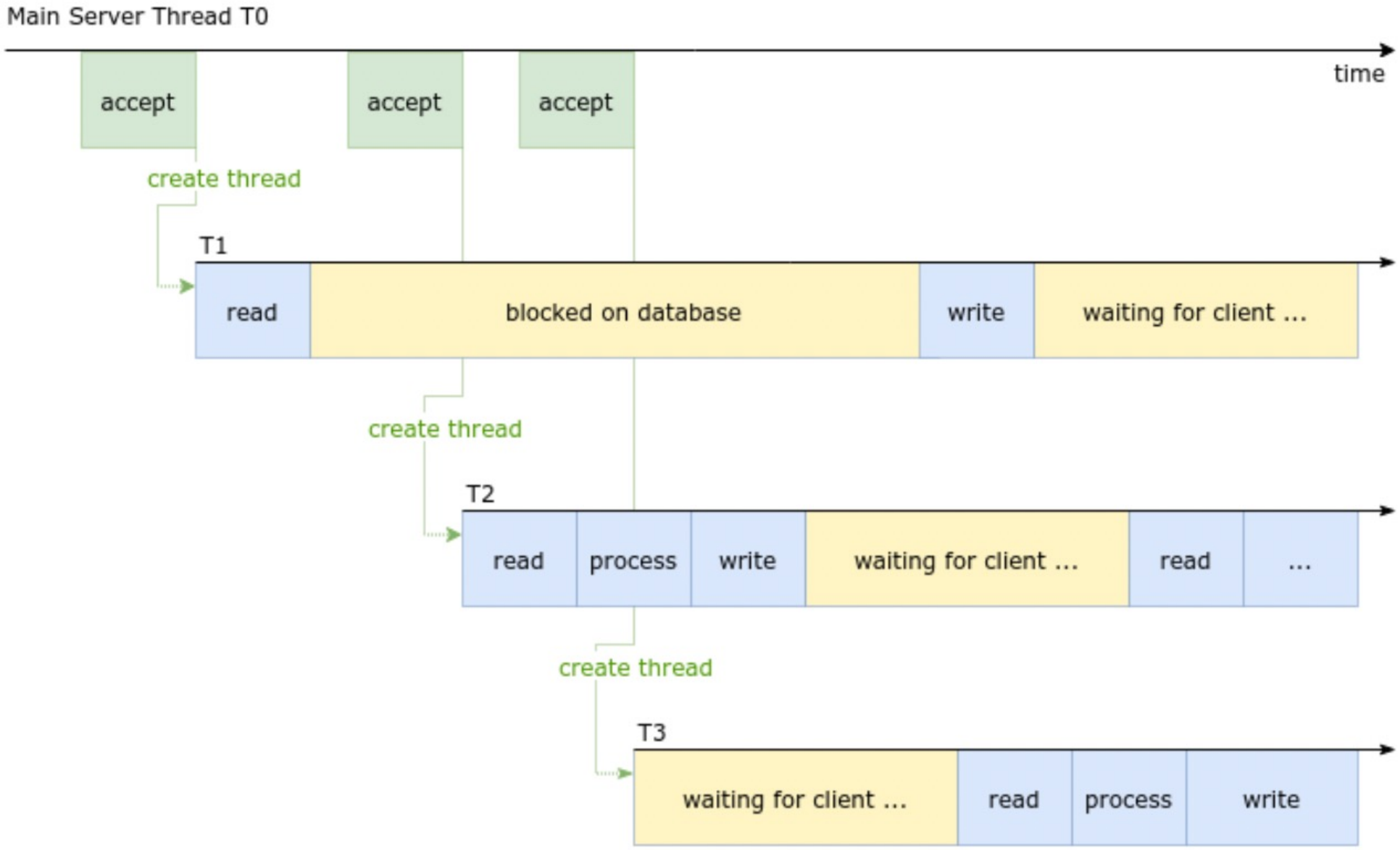
- CPU bound tasks could speed up with multi-threading
  - More CPU power -> process finishes sooner

- What about I/O bound ones?
  - More threads -> more idle threads -> more waste of resource

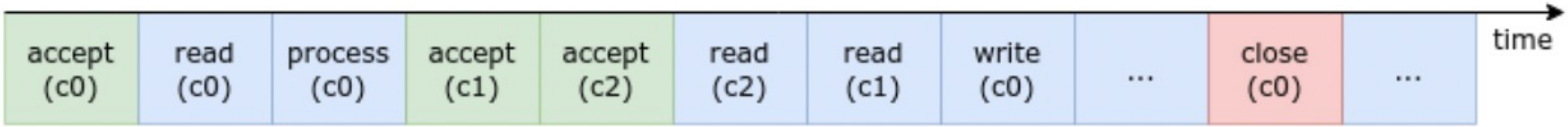
- Are API handlers I/O bound or CPU bound?

Web server architecture

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



Event loop



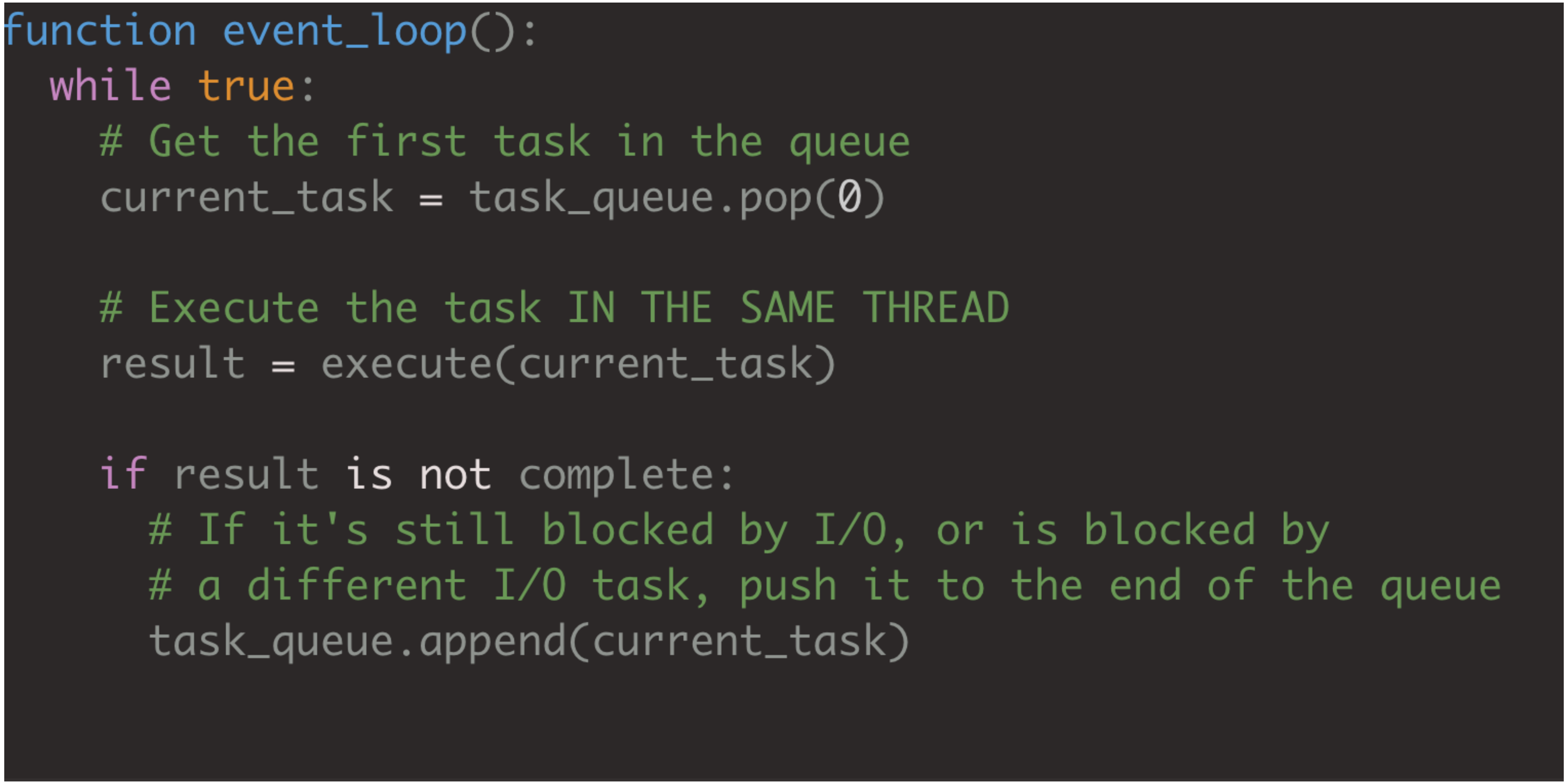
accept (c0) | read (c0) | process (c0) | accept (c1) | accept (c2) | read (c2) | read (c1) | write (c0) | ... | close (c0) | ... | time

- 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!

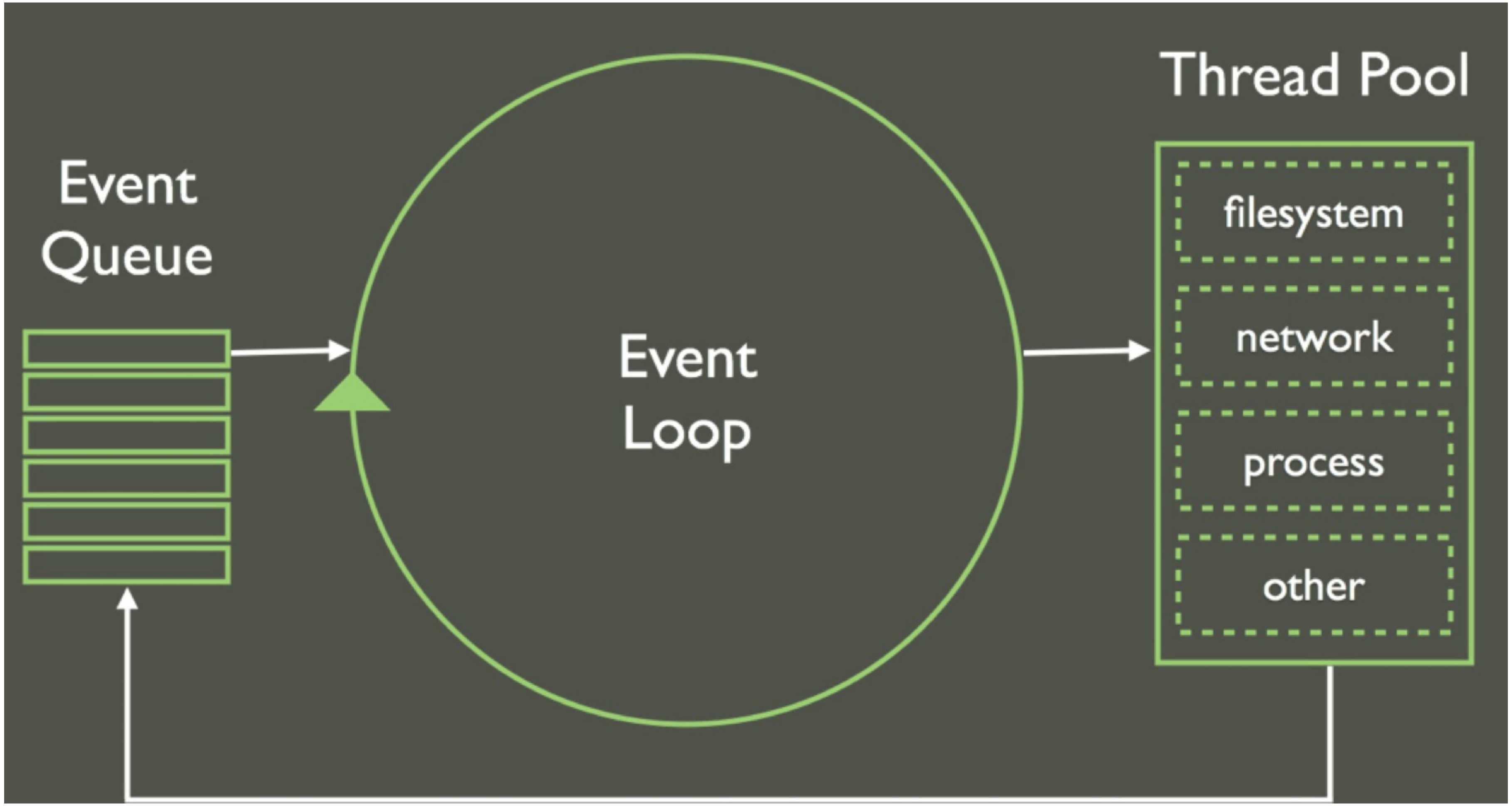
Event loop logic (simplified)

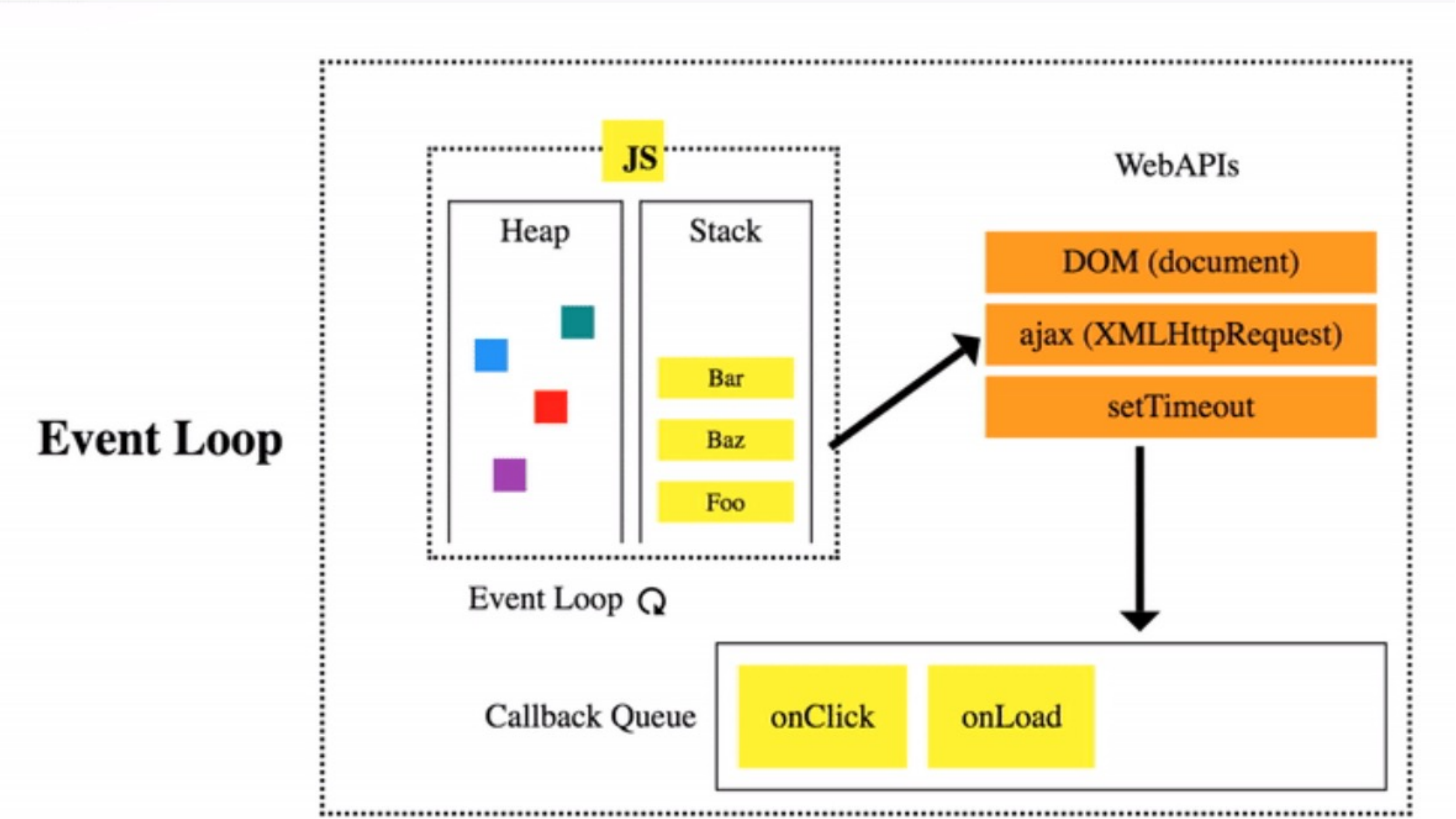


```
function event_loop():
  while true:
    # Get the first task in the queue
    current_task = task_queue.pop(0)

    # 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





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

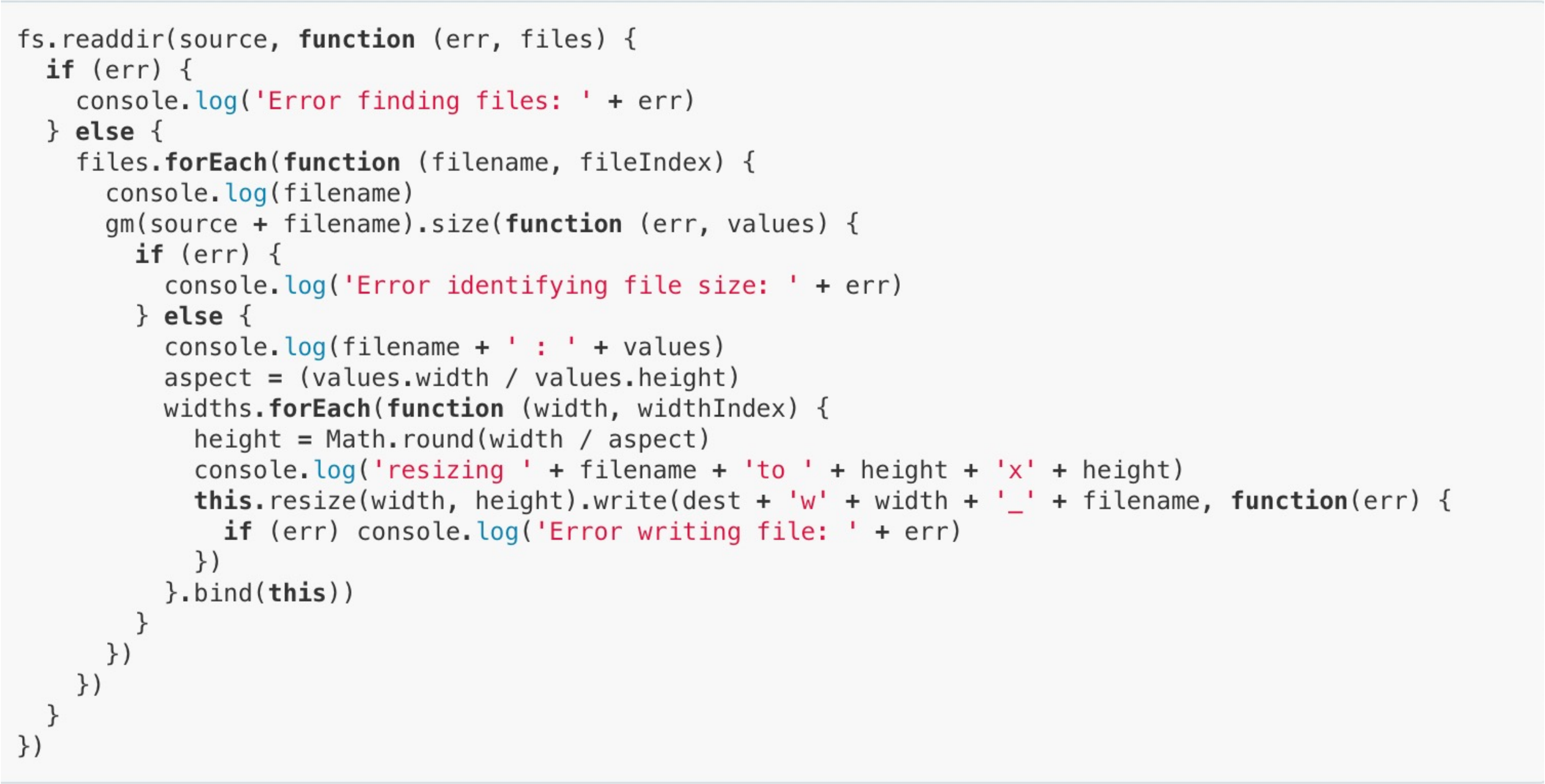
Async programming

- Not naturally available in 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!

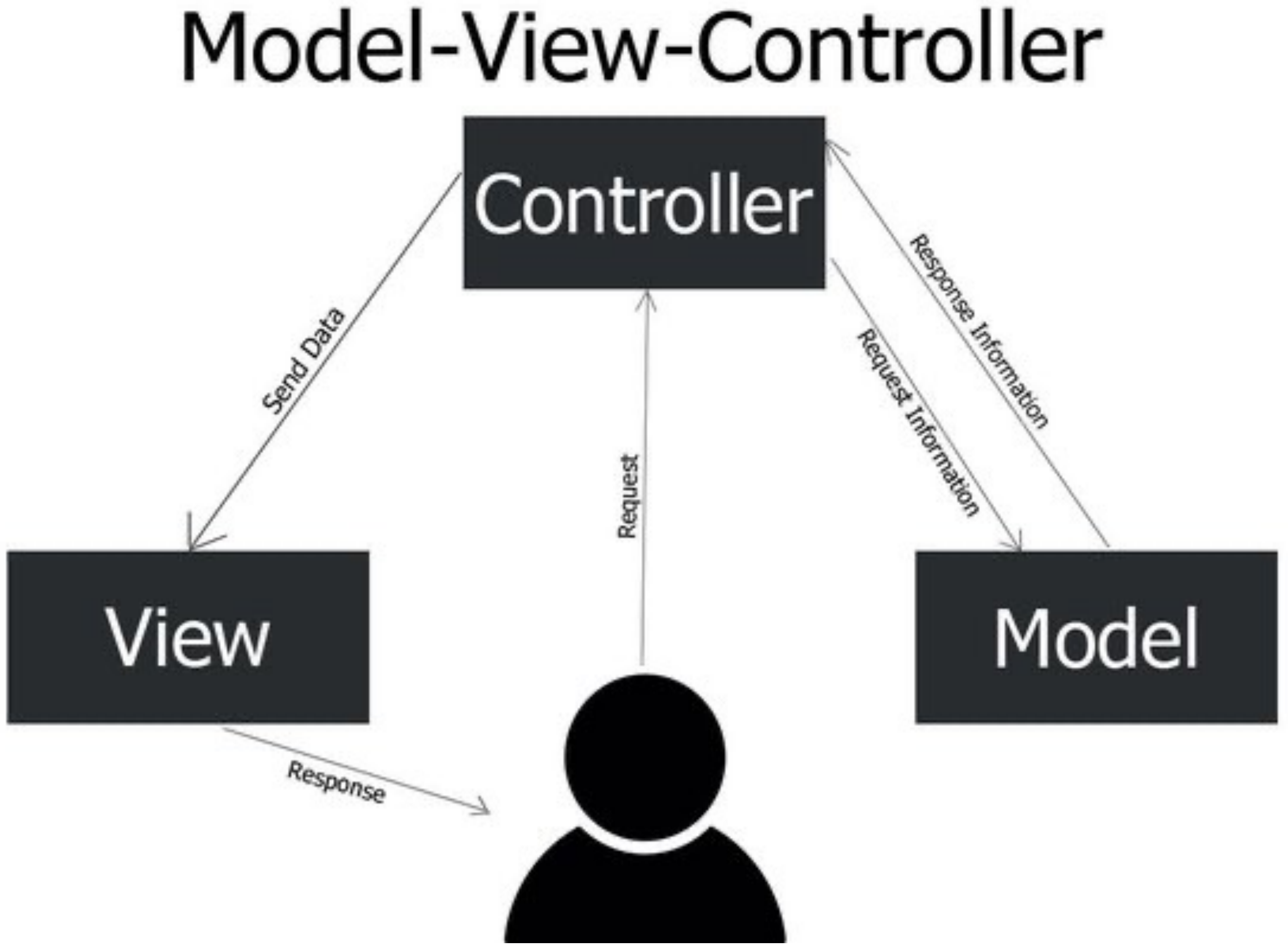
Async programming in JavaScript

- 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
  }
}
```

Midterm is up to the previous slide!

MVC



Source: https://bap-software.net/en/knowledge/mvc-model/

MVC in web apps

- View: the frontend components
  - HTML, CSS, client-side JS logic
    - In frontend is complex, it could have controller logic as well

- Controller: the API handler logic, Next.js framework, etc.
  - Request handling, interaction with client, querying database, etc.

- Model: Data management logic
  - How data should be defined, what fields are there, how it is stored in database

Data persistence

- We have not stored/read data so far!
  - Every web application needs a persistent storage

- Many different databases are around
  - Relational: Postgres, MySQL
  - Non-relational: Cassandra, MongoDB

- Node.js supports various database backends

Do we need Node.js support?

- Technically, we can make a TCP connection to any database and run queries
- But this is a terrible idea!
  - WHY?
- How can the framework/language help us out?

Object Relational Mapper

- Provides an abstraction over the underlying database queries
- Method/attribute accesses are translated to queries
- Results are wrapped by objects/attributes

Object Relational Mapper

- Simplicity: No need to use SQL syntax
- Consistency: Everything is in the same language (JS)
- Can switch database backend easily

- Enables Object Oriented Programming
- Runs a secure and efficient query
    - SQL injection, atomicity, etc.
- But for super-efficient queries, you might still need to run raw queries

SQLite

- Light-weight database that stores everything in one single file
- Follows standard SQL syntax
- Great option for development: no setup/installation required
- For production, switch to a more sophisticated database

Models

- Represents, and manages application’s data
  - The M from MVC

- Typically defined as classes

- Thanks to ORM, automatically mapped to a table in the database

Node.js ORMs

- Several ORMs exist
  - Prisma
  - Sequelize
  - TypeORM

- In this course, we use Prisma
  - Simple and very popular

Prisma

Visit https://www.prisma.io/docs/getting-started/quickstart

- Install via

```
npm i prisma @prisma/client @prisma/studio
```

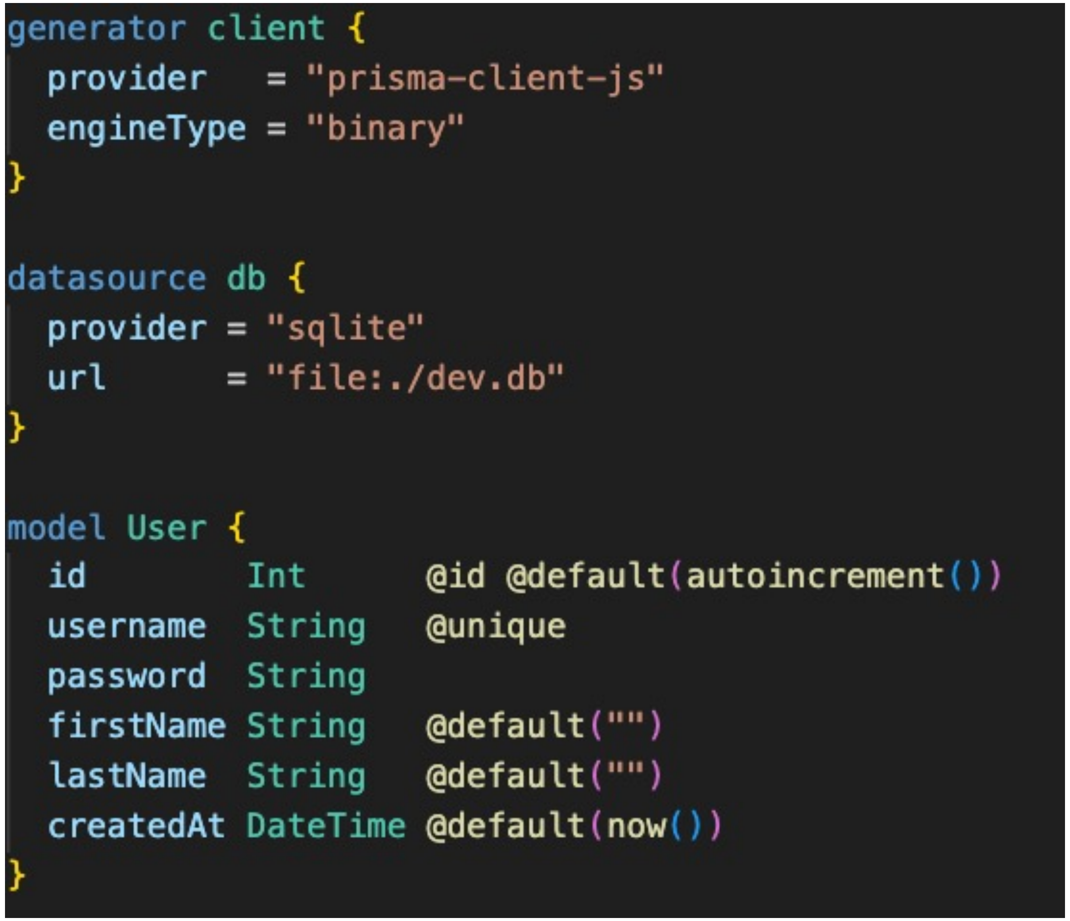
- Run `npx prisma init`
  - Creates a file named `schema.prisma`

- Prisma generates JS classes from its schema file
  - And syncs it with the database schema
  - More on that later in the course

The schema file

- The schema file is not a JS file
  - It's Prisma's custom language

- Model definition is something in between classes and tables



```
generator client {
  provider    = "prisma-client-js"
  engineType = "binary"
}

datasource db {
  provider = "sqlite"
  url      = "file:./dev.db"
}

model User {
  id        Int      @id @default(autoincrement())
  username  String   @unique
  password  String
  firstName String   @default("")
  lastName  String   @default("")
  createdAt DateTime @default(now())
}
```

Sync with database

- The schema file does not automatically impact anything!

- To generate the relevant JS classes:
  - Run `npx prisma generate`

- To sync the schema with the database:
  - Run `npx prisma migrate dev`

- More on these commands later in the course!

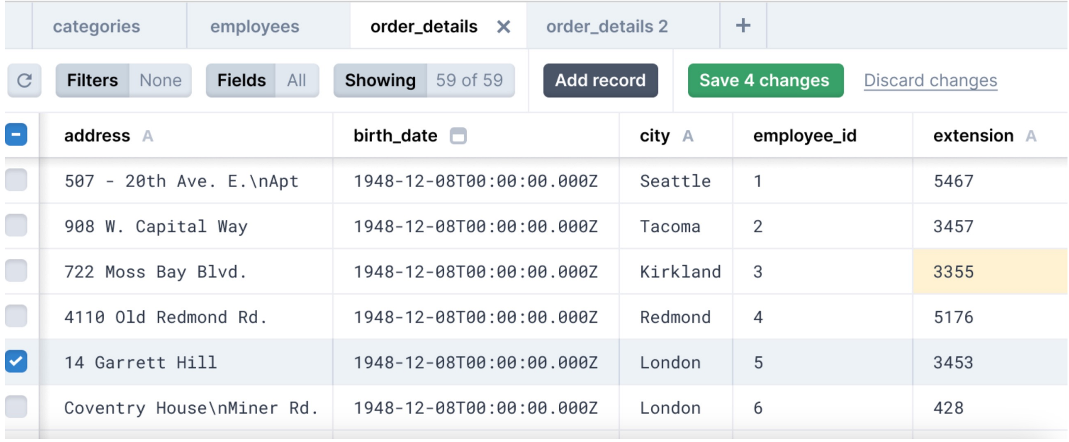
View the database

- Prisma studio

```
npx prisma studio
```

- Access from localhost:5555

- Great visual tool to browse the tables and modify data



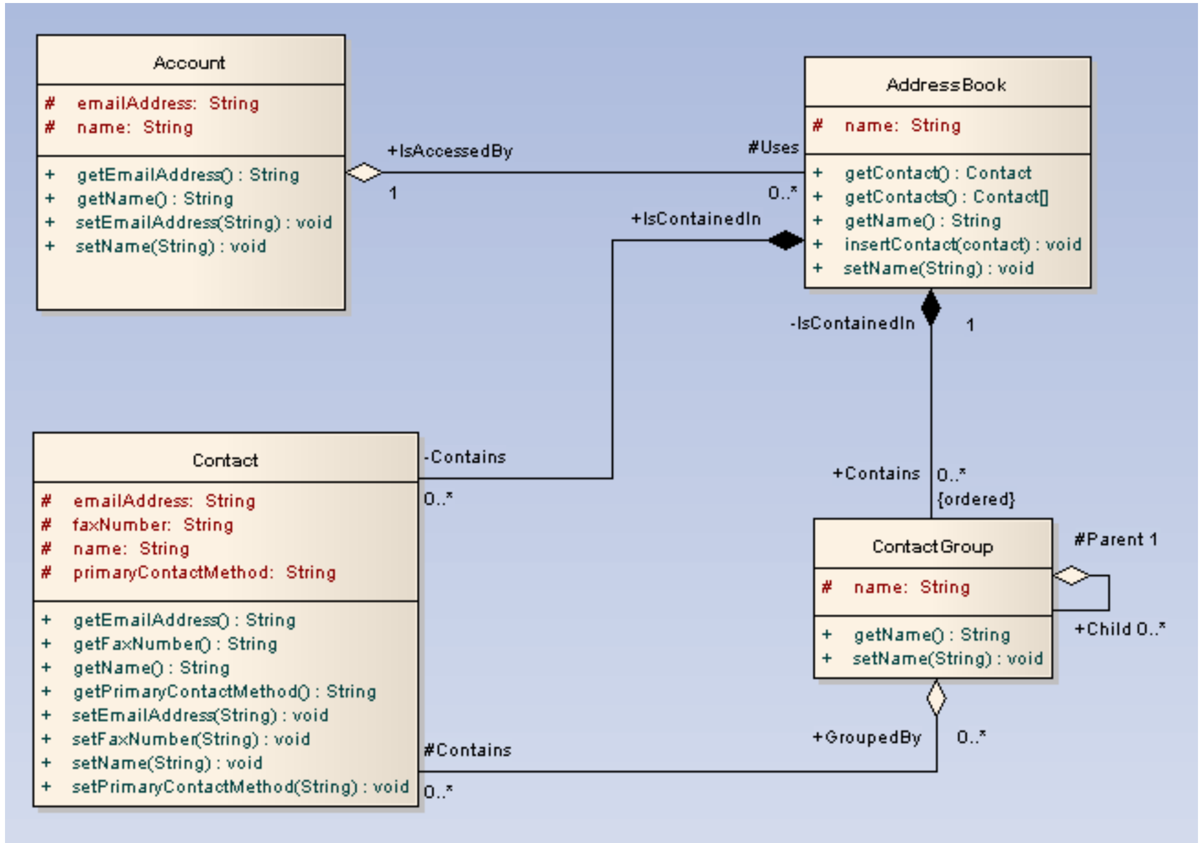
	address	birth_date	city	employee_id	extension
□	507 - 20th Ave. E.\nApt	1948-12-08T00:00:00.000Z	Seattle	1	5467
□	908 W. Capital Way	1948-12-08T00:00:00.000Z	Tacoma	2	3457
□	722 Moss Bay Blvd.	1948-12-08T00:00:00.000Z	Kirkland	3	3355
□	4110 Old Redmond Rd.	1948-12-08T00:00:00.000Z	Redmond	4	5176
☑	14 Garrett Hill	1948-12-08T00:00:00.000Z	London	5	3453
□	Coventry House\nMiner Rd.	1948-12-08T00:00:00.000Z	London	6	428

Model design

- MUST be done before coding starts
- Independent of programming language and framework
- Changing the models is not always easy
  - Especially in the production phase
- Models involve user data: the most sensitive part of your application
  - It’s important to design secure and efficient models

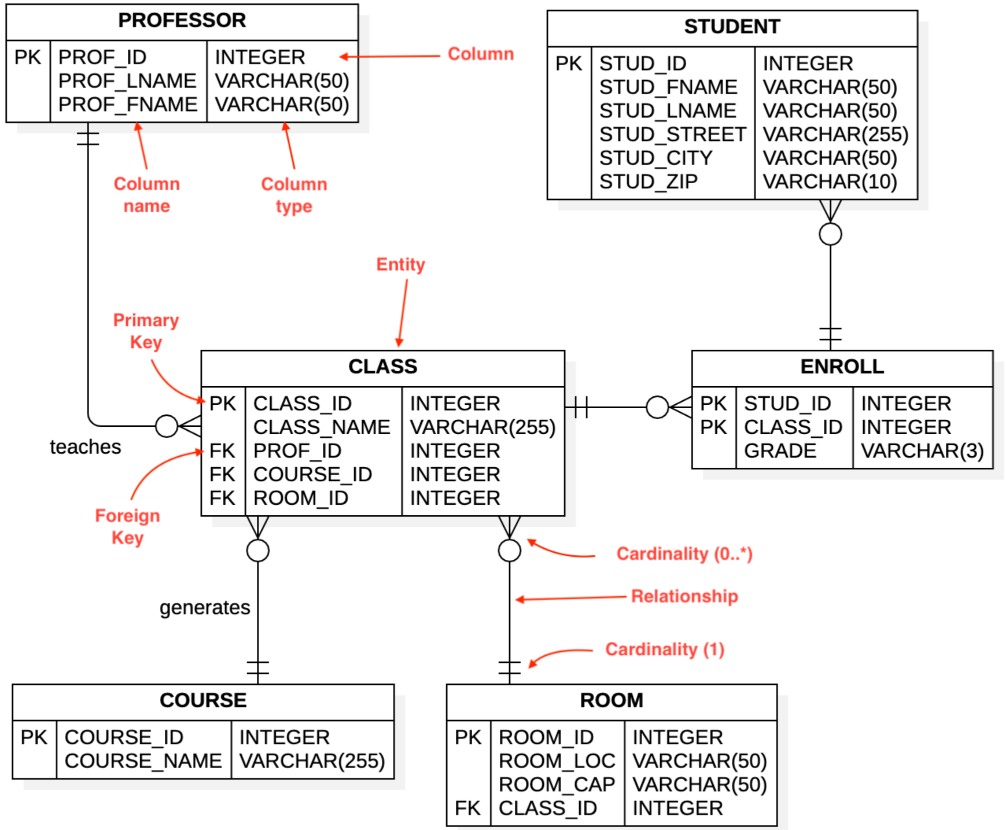
Model design

Class diagram



Source: https://sparxsystems.com/images/screenshots/uml2_tutorial/cl01.png

ER diagram



PROFESSOR		
PK	PROF_ID	INTEGER
	PROF_LNAME	VARCHAR(50)
	PROF_FNAME	VARCHAR(50)

STUDENT		
PK	STUD_ID	INTEGER
	STUD_FNAME	VARCHAR(50)
	STUD_LNAME	VARCHAR(50)
	STUD_STREET	VARCHAR(255)
	STUD_CITY	VARCHAR(50)
	STUD_ZIP	VARCHAR(10)

CLASS		
PK	CLASS_ID	INTEGER
	CLASS_NAME	VARCHAR(255)
FK	PROF_ID	INTEGER
FK	COURSE_ID	INTEGER
FK	ROOM_ID	INTEGER

ENROLL		
PK	STUD_ID	INTEGER
PK	CLASS_ID	INTEGER
	GRADE	VARCHAR(3)

COURSE		
PK	COURSE_ID	INTEGER
	COURSE_NAME	VARCHAR(255)

ROOM		
PK	ROOM_ID	INTEGER
	ROOM_LOC	VARCHAR(50)
	ROOM_CAP	VARCHAR(50)
FK	CLASS_ID	INTEGER

Source: https://docs.staruml.io/working-with-additional-diagrams/entity-relationship-diagram

Model design

- Example: an online shopping application
  - Potential models: user, store, product, order, shipment, etc.

- Example: a learning management system (LMS)
  - Potential models: user, course, student, assignment, etc.

- Example: a news application
  - Potential models: user, news, reporter, comment, report, etc.

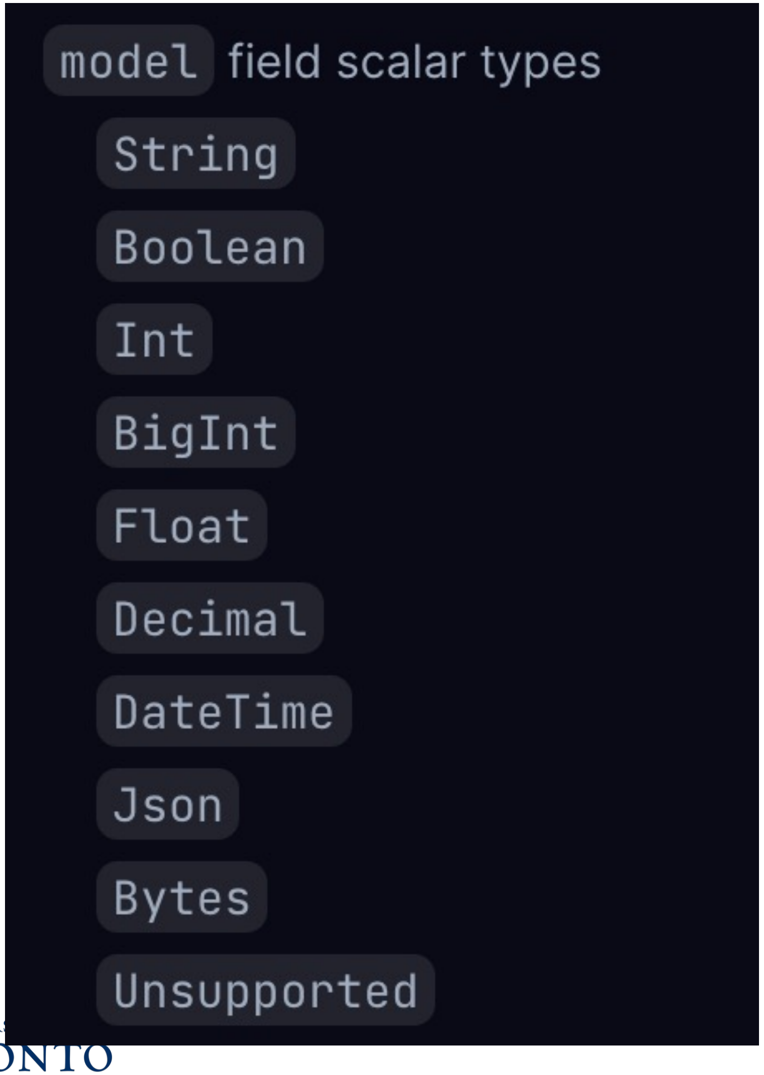
Prisma schema

- Data source: type and address of the database
  - provider could be `sqlite`, `mysql`, `postgresql`, etc.
  - url could be file address or server address with credentials

- Define one `model` for each model in the ER (or class) diagram
  - Add fields from diagrams as well
  - Mapped to database column by the ORM

Fields

Visit https://www.prisma.io/docs/orm/reference/prisma-schema-reference#model-fields



`model` field scalar types

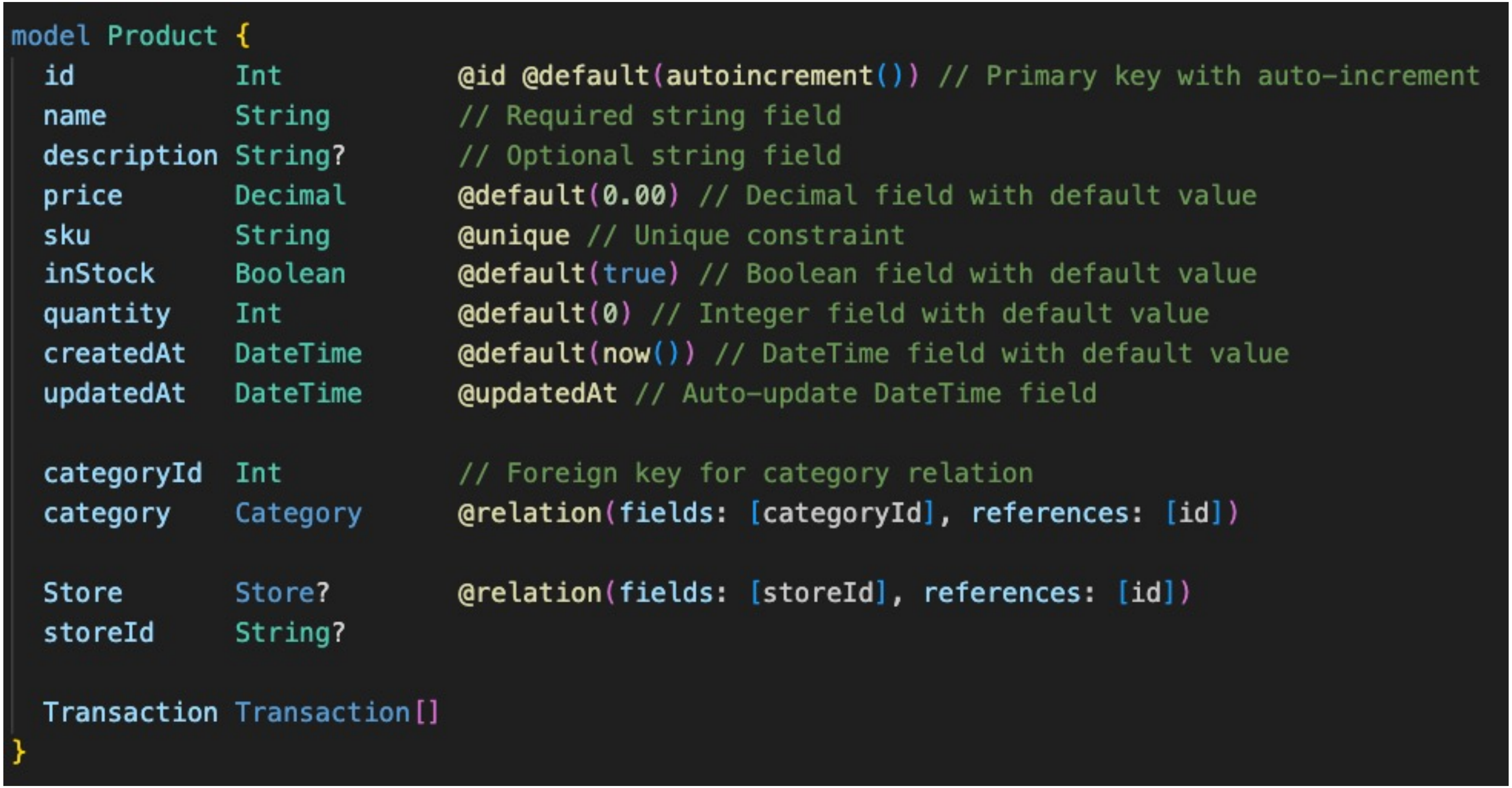
- `String`
- `Boolean`
- `Int`
- `BigInt`
- `Float`
- `Decimal`
- `DateTime`
- `Json`
- `Bytes`
- `Unsupported`

- Field Attributes

  `@id`
  `@default`
  `@unique`
  `@map`
  `@index`
  …
- Model attributes

  `@@unique`
  `@@map`
  …

Example model



```
model Product {
  id          Int           @id @default(autoincrement()) // Primary key with auto-increment
  name        String        // Required string field
  description String?       // Optional string field
  price       Decimal       @default(0.00) // Decimal field with default value
  sku         String        @unique // Unique constraint
  inStock     Boolean       @default(true) // Boolean field with default value
  quantity    Int           @default(0) // Integer field with default value
  createdAt   DateTime      @default(now()) // DateTime field with default value
  updatedAt   DateTime      @updatedAt // Auto-update DateTime field

  categoryId  Int           // Foreign key for category relation
  category    Category      @relation(fields: [categoryId], references: [id])

  Store       Store?        @relation(fields: [storeId], references: [id])
  storeId     String?

  Transaction Transaction[]
}
```

Null values

- The ? symbol indicates a nullable field
- Having default values is encouraged over null values
    - Null introduces typing complexity, potential for crashes, etc.
- Examples:
    - Empty string, False, 0
- When to use null?
    - When the default value is really distinct from null (e.g., 0 vs null)
    - Depends on the use case

ID (primary key)

- Encouraged to define a separate, automatic field for id
  - Either auto-incrementing integer or a Universally Unique Identifier (UUID)

- Over time, nearly every assumption initially made about the model changes
  - Changing the primary key is almost impossible

Relations

- Use `@relation` for many-to-one and one-to-many relations
  - Defined as a foreign key
  - Also define a column that stores the id of the referenced model

- Example:

```
categoryId Int
category   Category @relation("CategoryProduct", fields:
[categoryId], references: [id])
```

- Reverse traversal done by a field in the original model

```
product Product[] @relation("CategoryProduct")
```

Other relations

- One-to-one relations
  - Similar to one-to-many
  - Mark the foreign key column with `@unique`

- Many-to-many relations
  - Simply define an array at each end
  - Turned into a separate table by the ORM
  - See www.prisma.io/docs/orm/prisma-schema/data-model/relations/many-to-many-relations

Next session

- Querying the database in Next.js API handlers
    - CRUD

- Midterm at 6pm!