The call stack, event loop and callback queue in Node

— **Call stack**: JavaScript keeps track of what function is being run and where it was run from. Whenever a function is to be run, it's added to the call stack

— **Callback queue** - any functions delayed from running (and run automatically by Node) are added to the callback queue when the background Node task has completed (or there's been some activity like a request)

— **Event loop** - Determines what function/code to run next from the queue(s)

Rules for the automatic execution of the JS code by Node

1. Hold each deferred function in one of the task queues when the Node background API ‘completes’
2. Add the function to the Call stack (i.e. execute the function) ONLY when the call stack is totally empty (Have the Event Loop check this condition)
3. Prioritize functions in Timer ‘queue’ over I/O queue, over setImmediate (‘check’) queue

Spread & Rest Operators

...

Spread

Used to split up array elements OR object properties

```
const newArray = [...oldArray, 1, 2]
const newObject = { ...oldObject, newProp: 5 }
```

Rest

Used to merge a list of function arguments into an array

```
function sortArgs(...args) {
    return args.sort()
}
```

Destructuring

Easily extract array elements or object properties and store them in variables

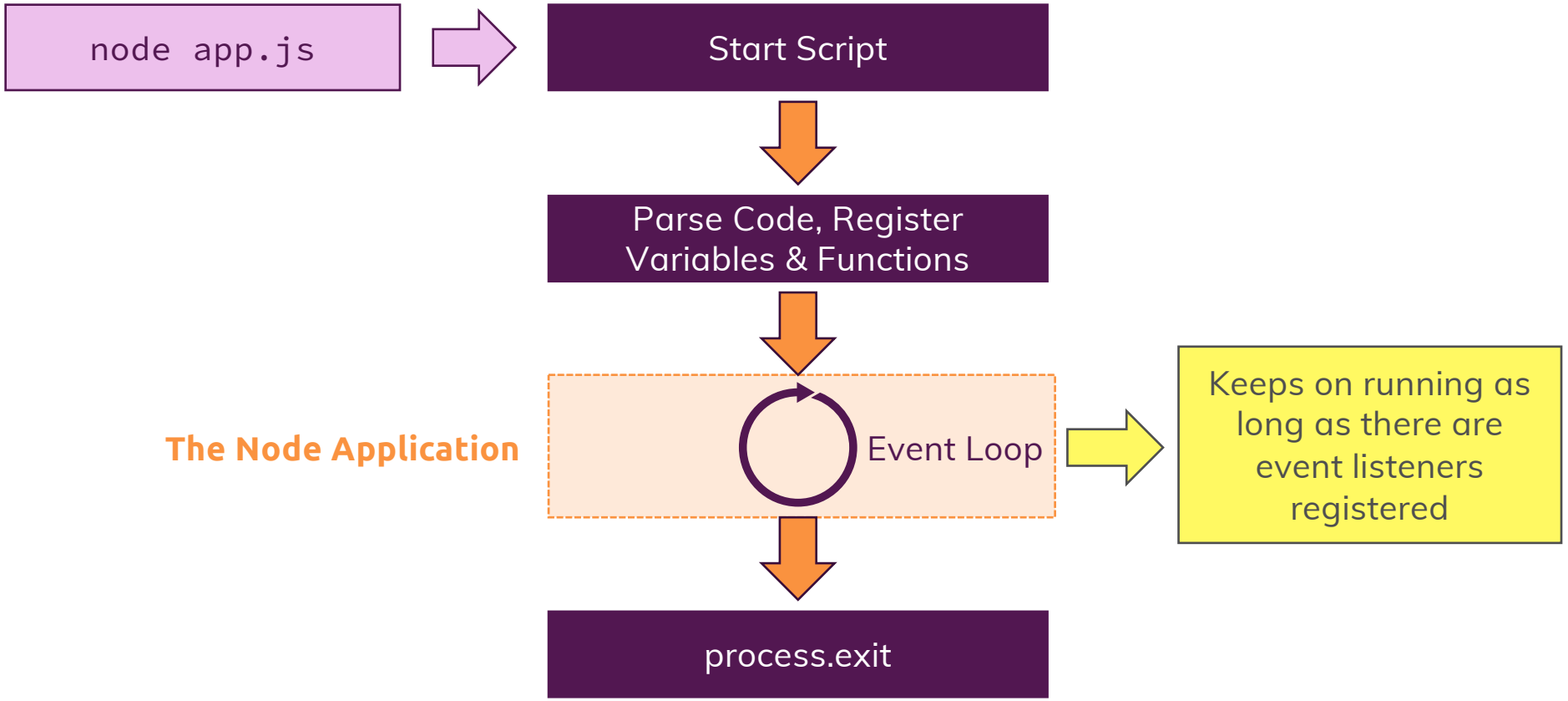
Array Destructuring

```
[a, b] = ['Hello', 'Max']
console.log(a) // Hello
console.log(b) // Max
```

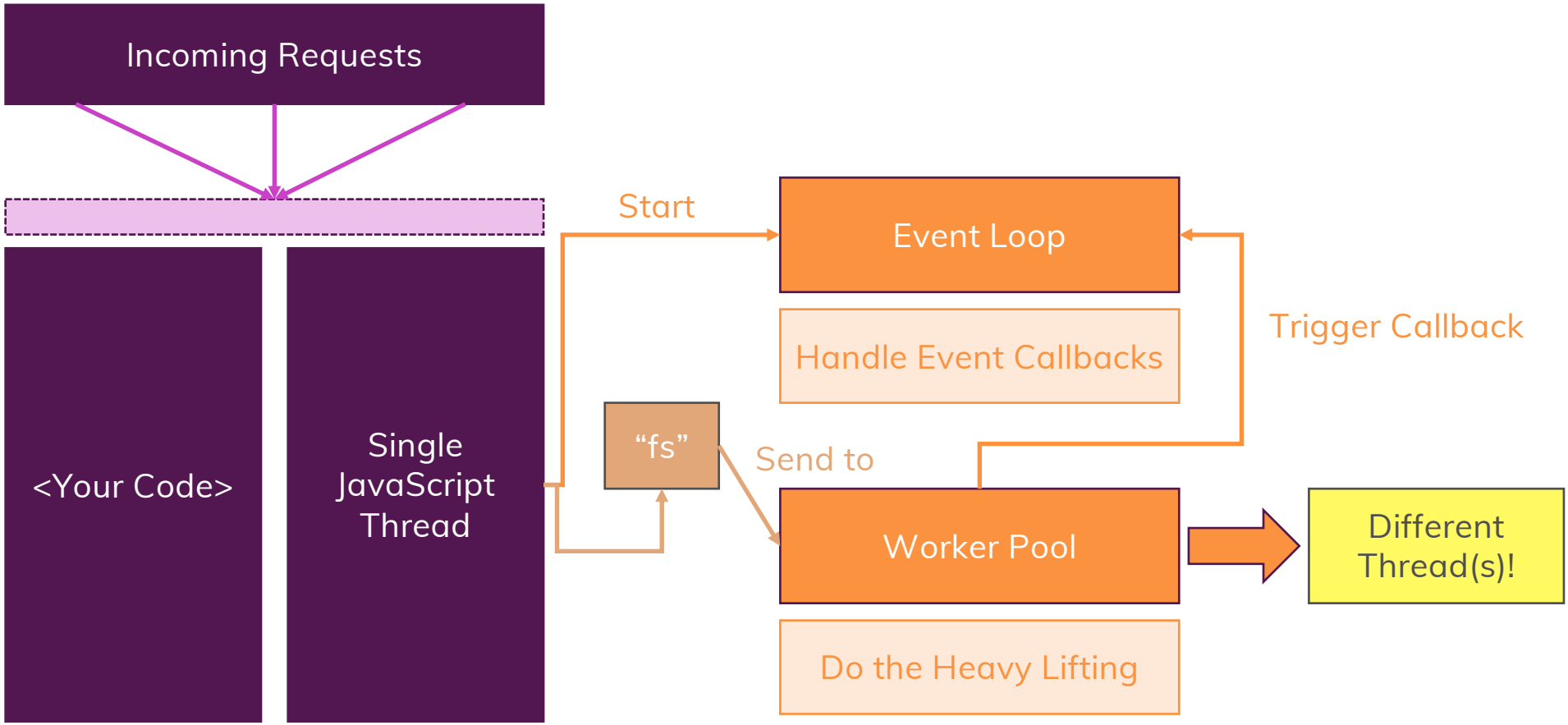
Object Destructuring

```
{name} = {name: 'Max', age: 28}
console.log(name) // Max
console.log(age) // undefined
```

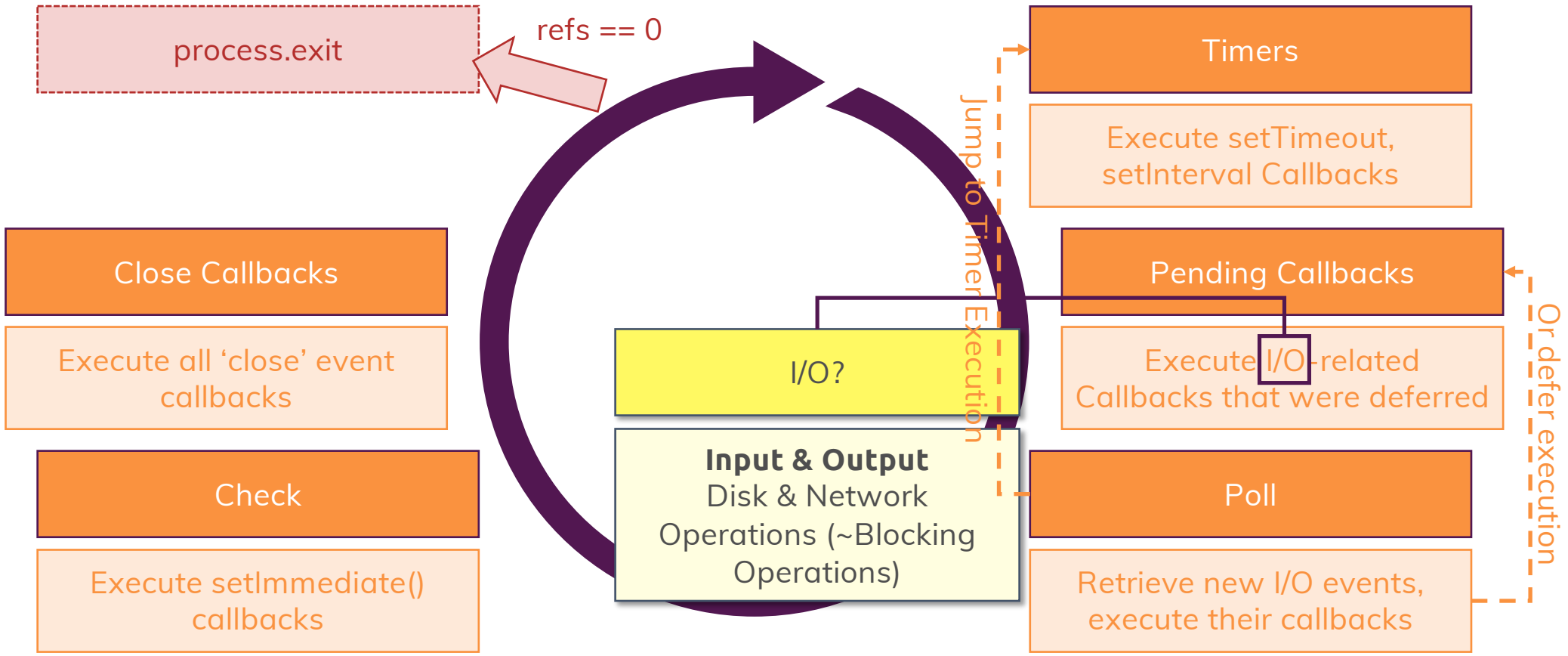
Node.js Program Lifecycle



Single Thread, Event Loop & Blocking Code



The Event Loop



ACADEMIND

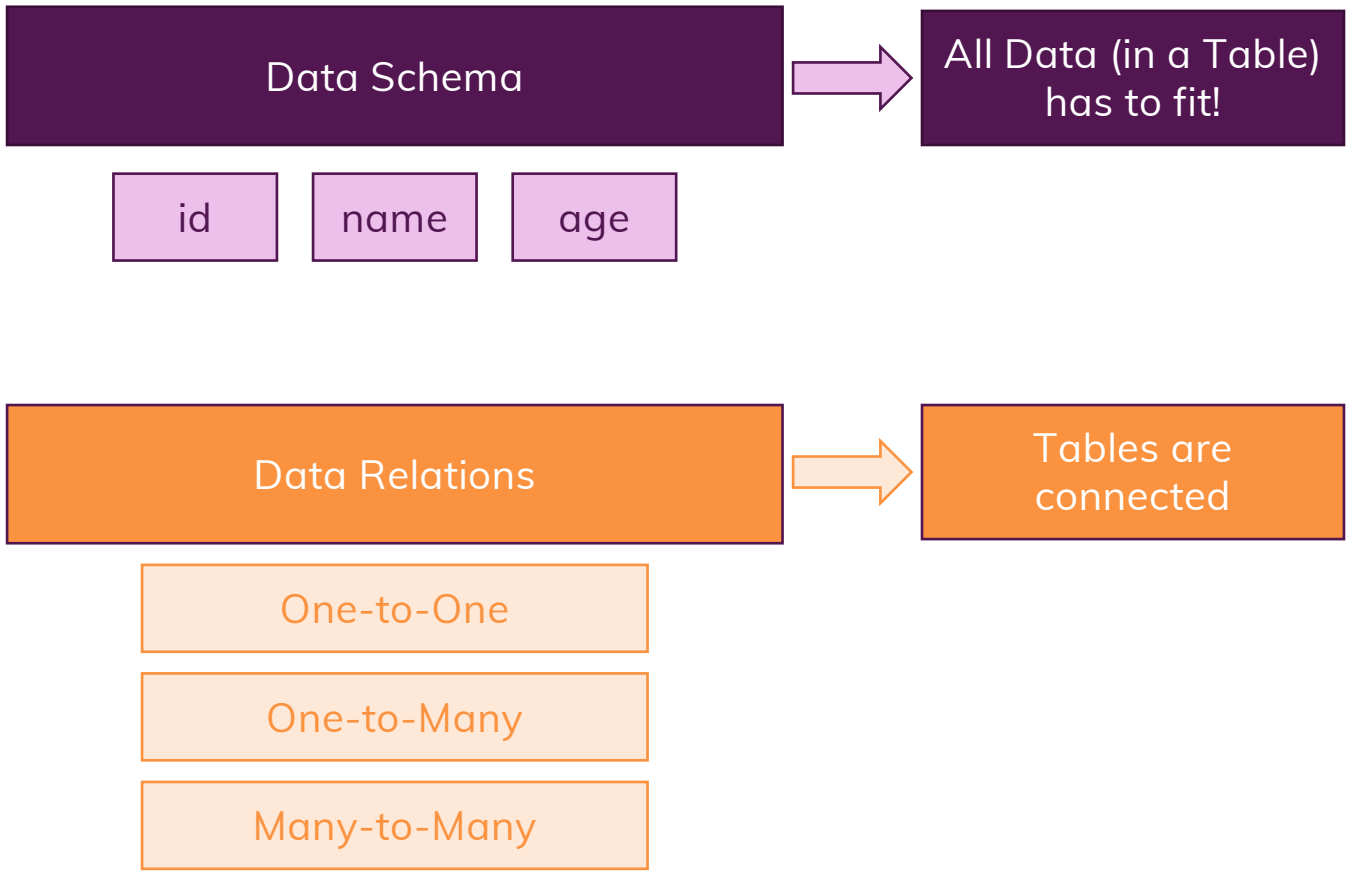
Types of Errors

Syntax Errors

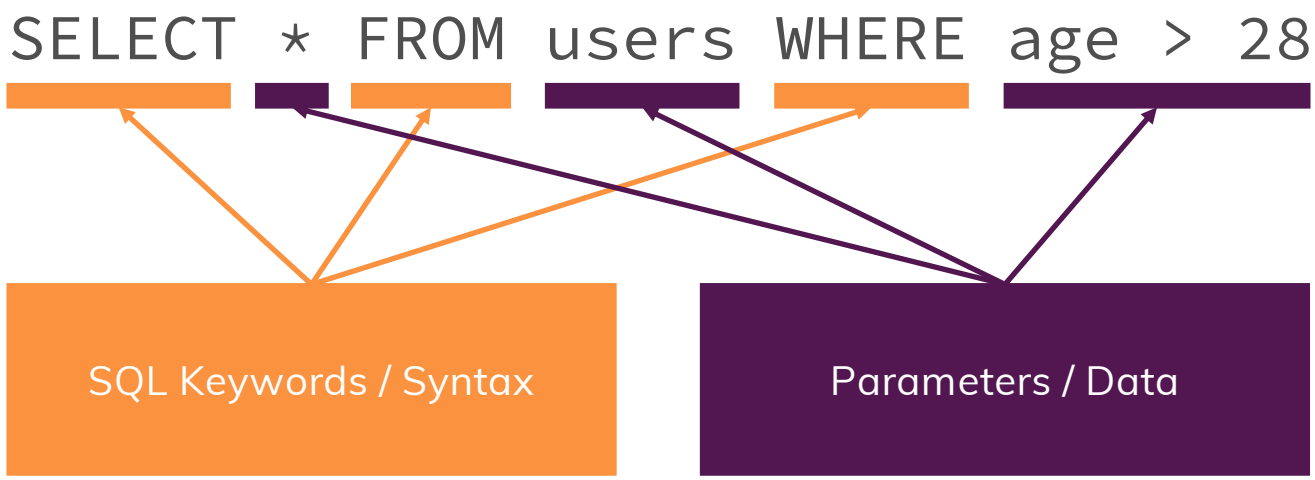
Runtime Errors

Logical Errors

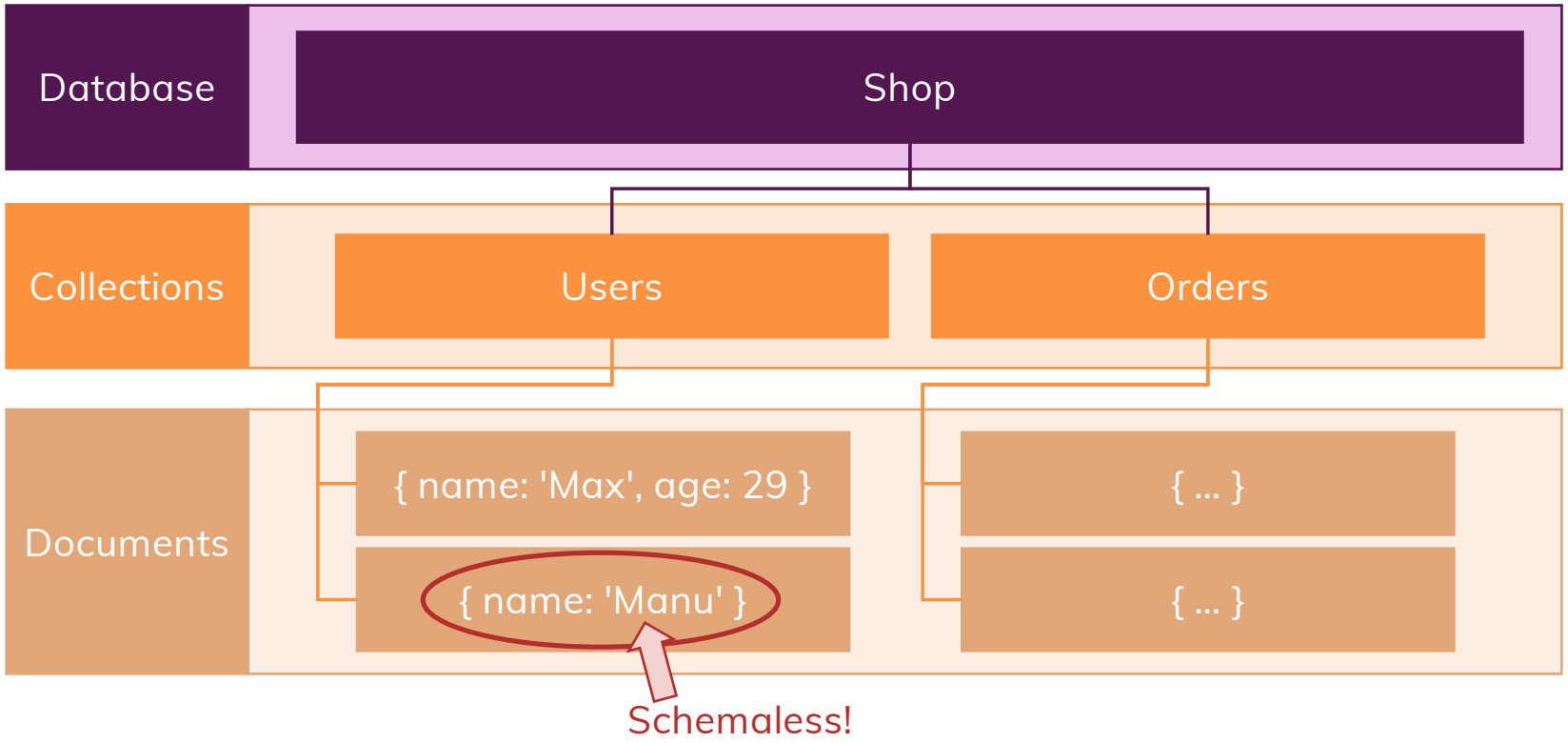
Core SQL Database Characteristics



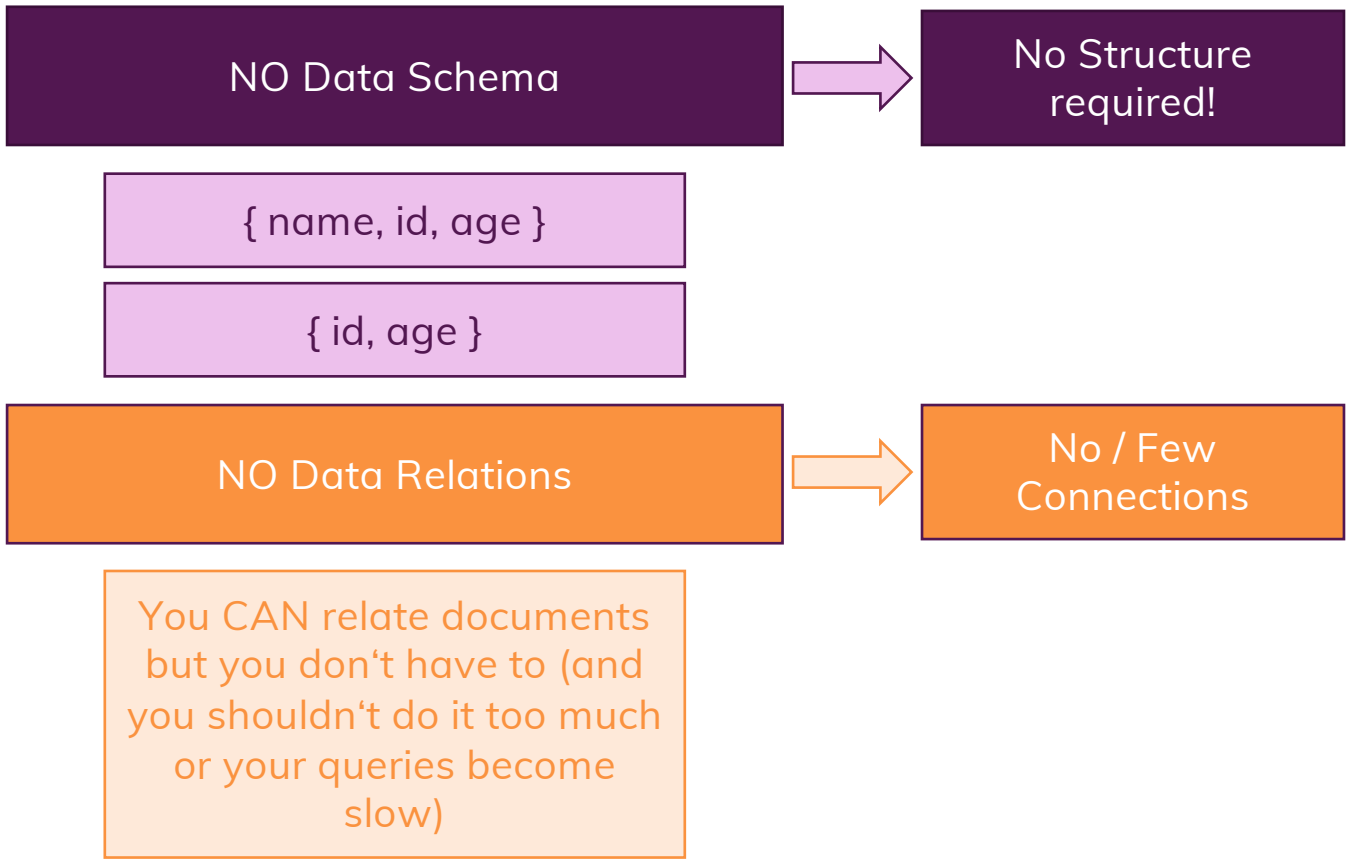
SQL Queries


SELECT * FROM users WHERE age > 28
SQL Keywords / Syntax
Parameters / Data

NoSQL

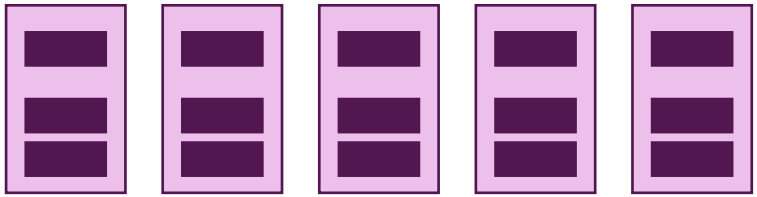

Database
Shop
Collections
Users
Orders
Documents
{ name: 'Max', age: 29 }
{ ... }
{ name: 'Manu' }
{ ... }
Schemaless!

NoSQL Characteristics



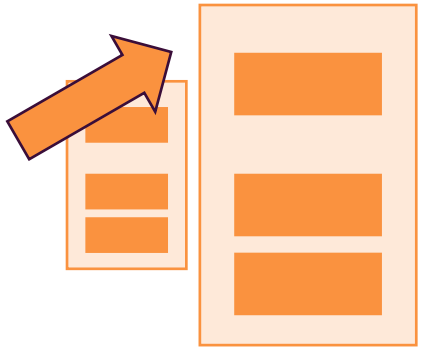
Horizontal vs Vertical Scaling

Horizontal Scaling



Add More Servers (and merge Data into one Database)

Vertical Scaling



Improve Server Capacity / Hardware

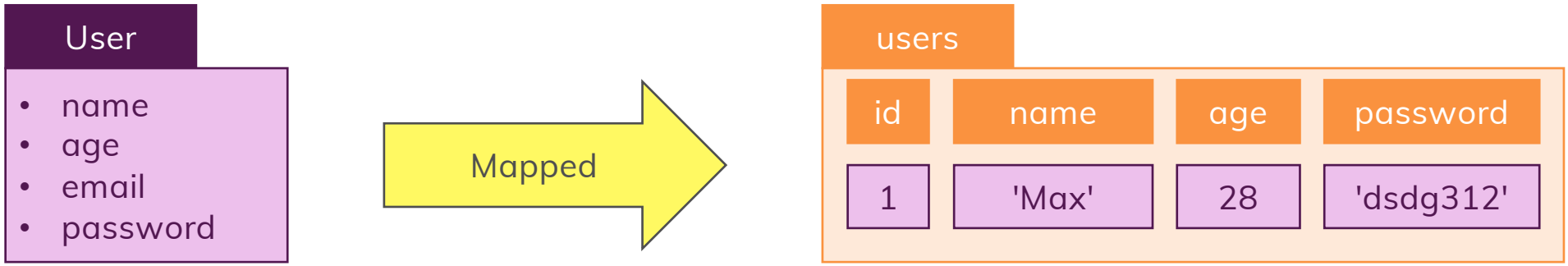
ACADEMIND

SQL vs NoSQL

SQL	NoSQL
Data uses Schemas	Schema-less
Relations!	No (or very few) Relations
Data is distributed across multiple tables	Data is typically merged / nested in a few collections
Horizontal scaling is difficult / impossible; Vertical scaling is possible	Both horizontal and vertical scaling is possible
Limitations for lots of (thousands) read & write queries per second	Great performance for mass read & write requests

What is Sequelize?

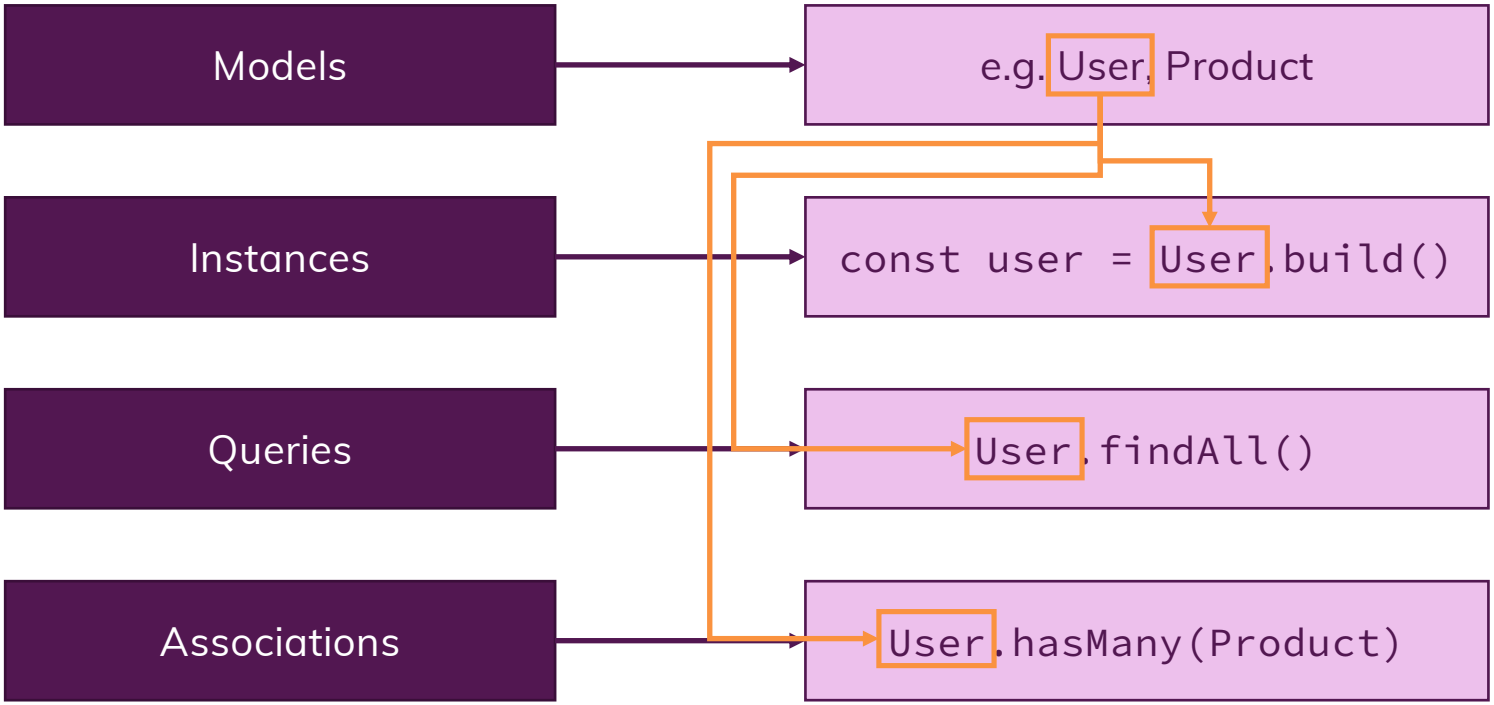
An Object-Relational Mapping Library




```
INSERT INTO users VALUES (1, 'Max', 28, 'dsdg312')
```

```
const user = User.create({ name: 'Max', age: 28, password: 'dsdg312' })
```

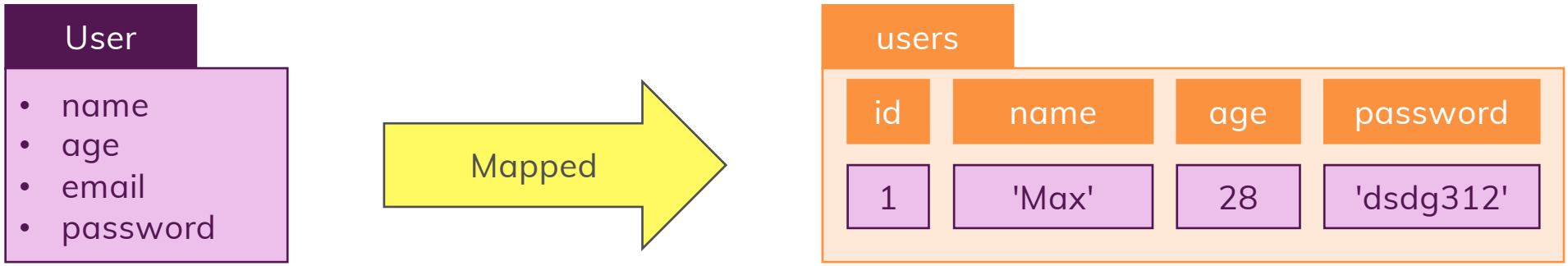
Core Concepts



ACADEMIND

What is Mongoose?

A Object-Document Mapping Library

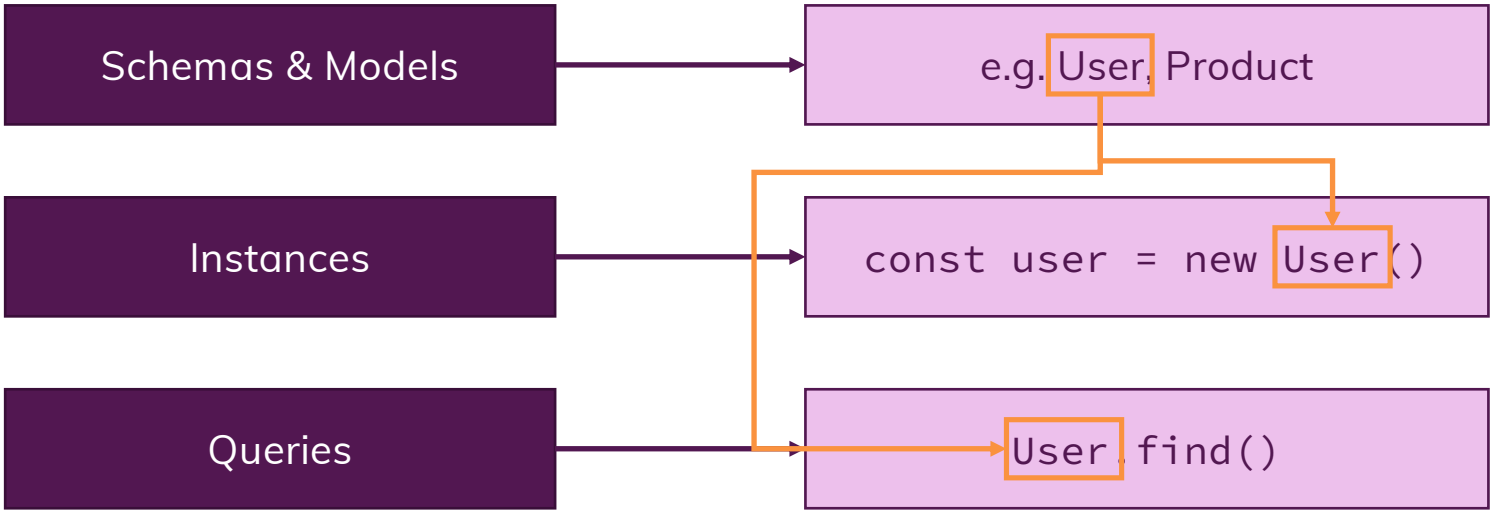



```
db.collection('users').insertO    name: 'Max', age: 28, password:
                                  'ds      ' })
```

```
const user = User.create({ name: 'Max', age: 28, password: 'dsdg312' })
```

ACADEMIND

Core Concepts



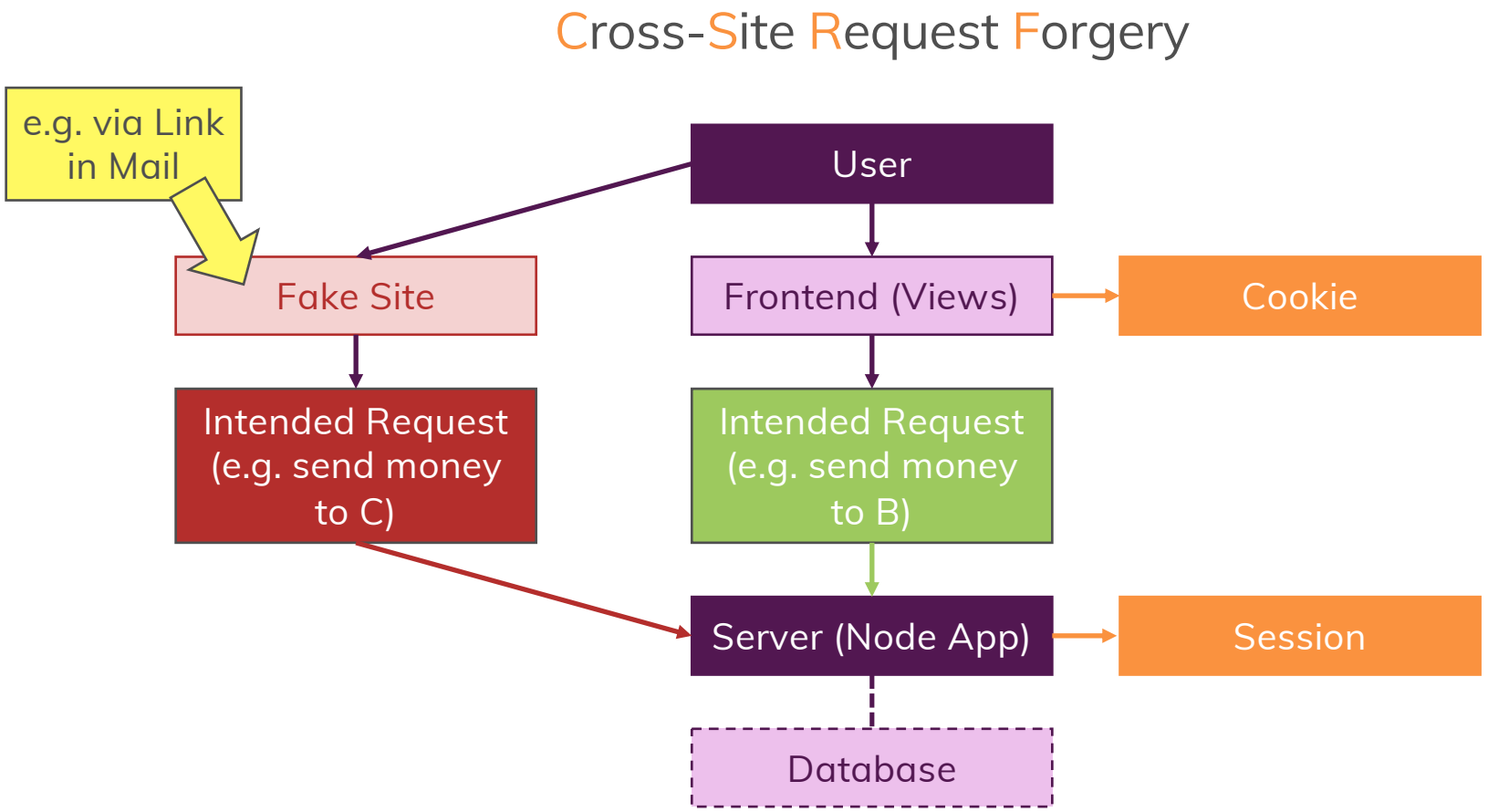
Sessions & Cookies

Persisting Data across Requests

When to use What

Cookies	Session
Stored on client	Stored on server
(Ad) Tracking	Authentication Status Management (across Requests)
Authentication Session Management	General Cross-Request Data Management

CSRF Attacks



ACADEMIND

Errors & Http Response Codes

2xx (Success)	200	Operation succeeded
	201	Success, resource created
3xx (Redirect)	301	Moved permanently
4xx (Client-side error)	401	Not authenticated
	403	Not authorized
	404	Not found
	422	Invalid input
5xx (Server-side error)	500	Server-side error

Http Methods (Http Verbs)

More than just GET & POST

GET	POST	PUT
Get a Resource from the Server	Post a Resource to the Server (i.e. create or append Resource)	Put a Resource onto the Server (i.e. create or overwrite a Resource)

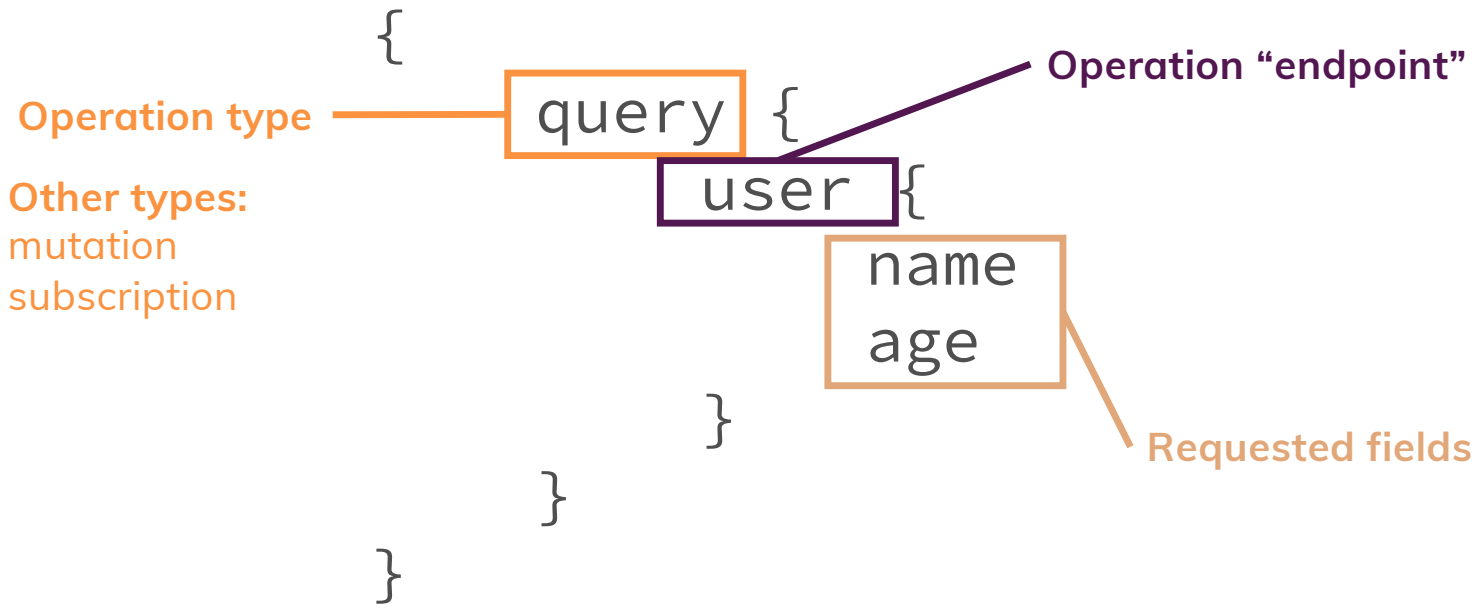
PATCH	DELETE	OPTIONS
Update parts of an existing Resource on the Server	Delete a Resource on the Server	Determine whether follow-up Request is allowed (sent automatically)

REST Principles

Uniform Interface	Stateless Interactions
Clearly defined API endpoints with clearly defined request + response data structure	Server and client don't store any connection history, every request is handled seperately

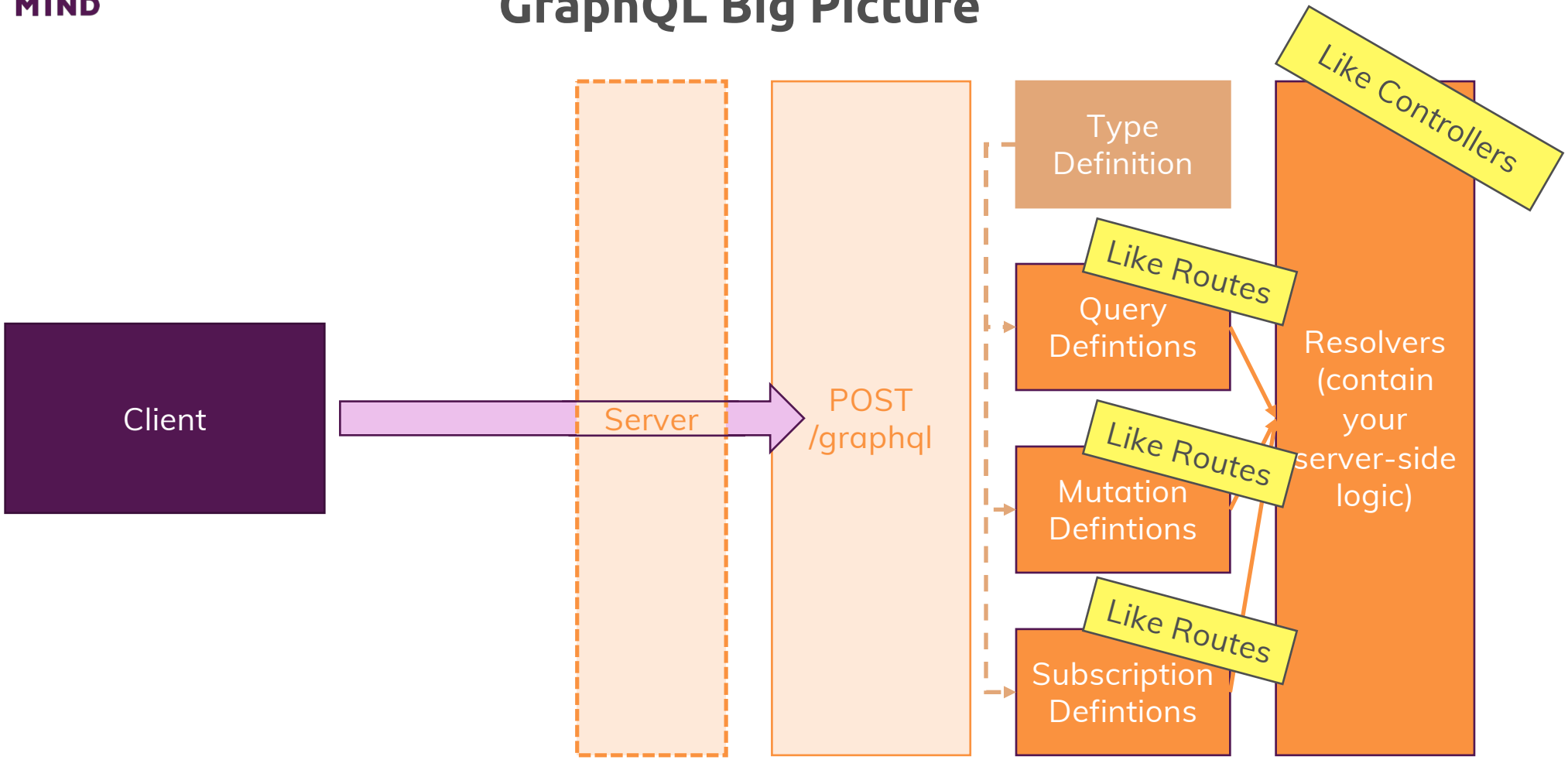
Cacheable	Client-Server	Layered System	Code on Demand
Servers may set caching headers to allow the client to cache responses	Server and client are separated, client is not concerned with persistent data storage	Server may forward requests to other APIs	Executable code may be transferred from server to client

A GraphQL Query



ACADEMIND

GraphQL Big Picture



Using SSL/TLS