JÖNKÖPING UNIVERSITY

School of Engineering

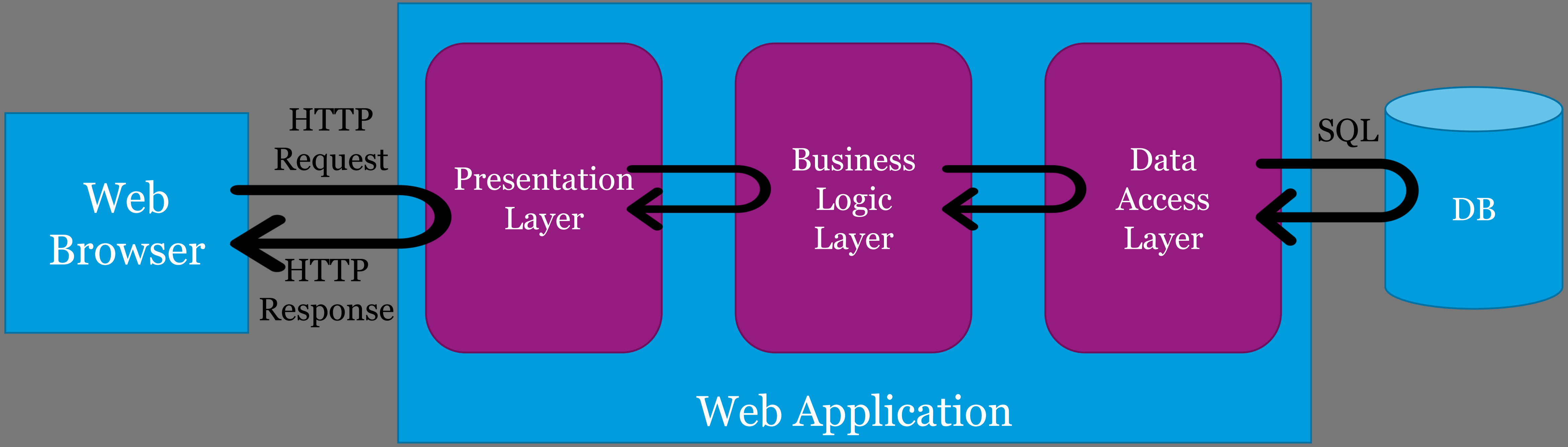
LAYERED ARCHITECTURE IN NODE.JS

Peter Larsson-Green

Jönköping University

Spring 2019

WHAT IS A LAYERED ARCHITECTURE?

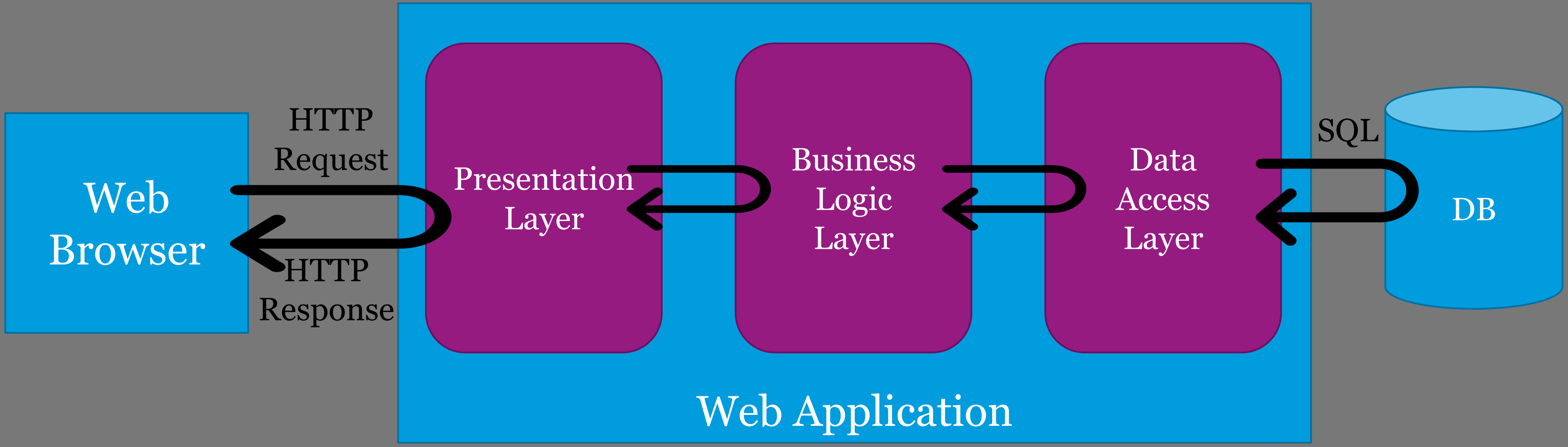


- HTTP
- HTML
- CSS
- Client-side JS
- …

- Validation
- Authorization
- …

- Storing data
- Retrieving data
- Updating data
- Deleting data
- …

WHY USE A LAYERED ARCHITECTURE?



- HTTP
- HTML
- CSS
- Client-side JS
- …

- Validation
- Authorization
- …

- Storing data
- Retrieving data
- Updating data
- Deleting data
- …

IMPLEMENTING A LAYERED ARCHITECTURE

One file per layer.

```
const accounts = [{id: 1, username: "Bob"}]
exports.getAccountById = function(id){
  return accounts.find(a => a.id == id)
}
```

dal.js

```
const bll = require('./bll')
app.get('/accounts/:id', function(req, res){
  try{
    const account = bll.getAccountById(req.params.id)
    // Send back response with account.
  }catch(error){
    // Send back response with error.
  }
})
```

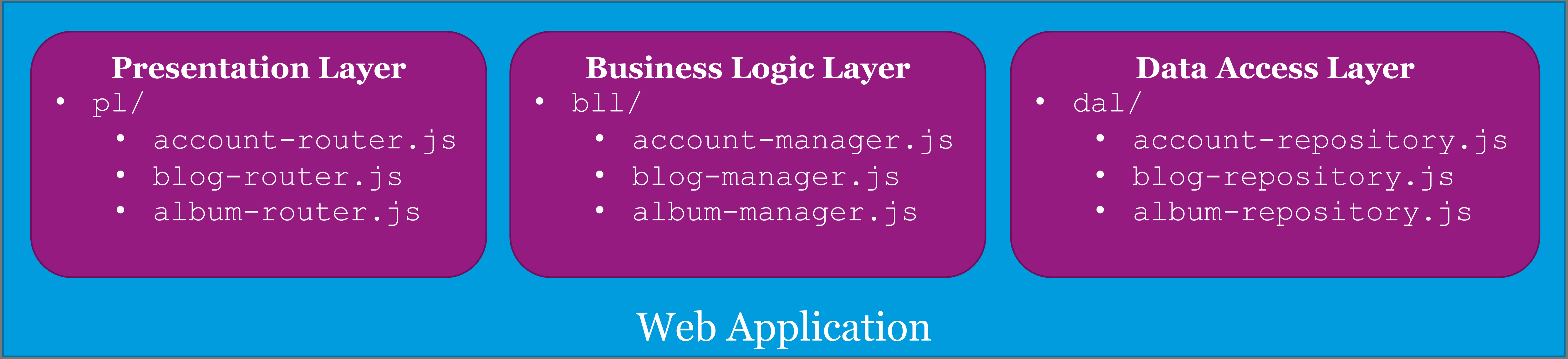
pl.js

```
const dal = require('./dal')
exports.getAccountById = function(id){
  if(/* User is logged in */){
    return dal.getAccountById(id)
  }else{
    throw "unauthorized"
  }
}
```

bll.js

IMPLEMENTING A LAYERED ARCHITECTURE

One folder per layer.



Presentation Layer

- `pl/`
  - `account-router.js`
  - `blog-router.js`
  - `album-router.js`

Business Logic Layer

- `bll/`
  - `account-manager.js`
  - `blog-manager.js`
  - `album-manager.js`

Data Access Layer

- `dal/`
  - `account-repository.js`
  - `blog-repository.js`
  - `album-repository.js`

Web Application

RELYING ON ABSTRACTION

```
const express = require('express')
const accountManager = require('../bll/account-manager')
const router = express.Router()
router.get("/:id", function(req, res){
  const id = req.params.id
  try{
    const account = accountManager.getAccountById(id)
    const model = {account: account}
    res.render("account.hbs", model)
  }catch(error){
    const model = {error: error}
    res.render("error.hbs", model)
  }
})
module.exports = router
```

pl/account-router.js

```
const accounts = [{id: 1, username: "Bob"}]
exports.getAccountById = function(id){
  return accounts.find(a => a.id == id)
}
```

dal/account-repository.js

```
const accountRepo = require('../dal/account-repository')
exports.getAccountById = function(id){
  // Throws the exception "unauthorized"
  // if the user is not allowed to get the
  // account, otherwise returns back the
  // account with the given id.
}
```

bll/account-manager.js

LEAKY ABSTRACTION

A layer should not be dependent on the implementation of the layer it makes use of, only its interface.

```
const accounts = [{id: 1, username: "Bob"}]
exports.getAccountById = function(id){
  return accounts.find(a => a.id == id)
}
```

dal/account-repository.js

```
const sqlite = require('sqlite3')
const db = new sqlite.Database("my-db.db")
exports.getAccountById = function(id){
  const query = "SELECT * FROM accounts WHERE id = ?"
  db.get(query, [id], function(error, account){
    return account
  })
}
```

dal/account-repository.js

```
const accountRepo = require('../dal/account-repository')
exports.getAccountById = function(id){
  if(/* User is logged in */){
    return accountRepo.getAccountById(id)
  }else{
    throw "unauthorized"
  }
}
```

bll/account-manager.js

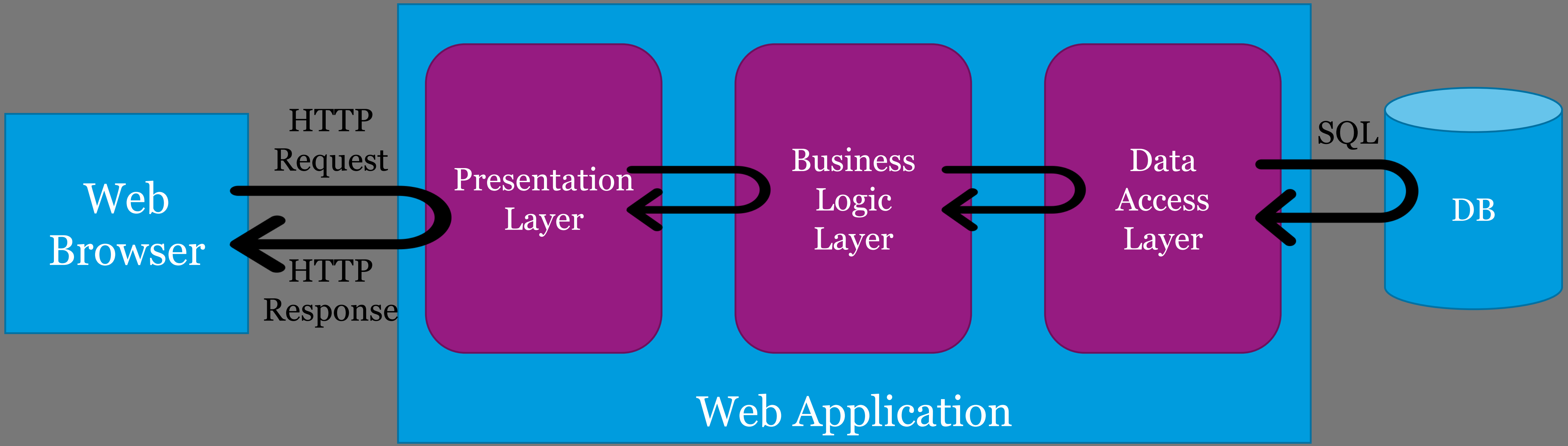
IMPROVING PERFORMANCE

Sometimes you move the responsible down one layer to improve the performance.

- Example: creating new account:
  - Proper way:
    - BLL: Ask DAL if there exists a user with the given username.
    - BLL: If no, ask DAL to create a new account with the given username.
  - Common way:
    - BLL: Ask DAL to create a new account with the given username.
    - DAL: Use a unique constraint on username.
- Example: max number of characters in username.

HANDLING ERRORS

Errors needs to be propagated to the outer layers.


Web Browser
HTTP Request
HTTP Response
Presentation Layer
Business Logic Layer
Data Access Layer
Web Application
SQL
DB