Databases & ORMs

Intro to ORMs

Database Management Sys. (DBMS)

- How to define records (Data Definition)
- Creating/Updating/Deleting Records
- Querying/Retrieval of Records
- Administering the database system

Progression of Databases

- Navigational (< 1970s)
- Relational (> 1970s)
  - Postgresql, MySQL
- NoSQL (> 2000s)
  - BigTable, Cassandra, Dynamo, Spanner

Postgres History

- 1970s (UC Berkley): INteractive Graphics REtrieval System (INGRES)
- 1980s: Postgres (“Ingres++”)
- 1995: POSTQUEL and Postgres95
- 1996: open source community adopts it
- Since: stability, testing, documentation, new features
- PostgreSQL

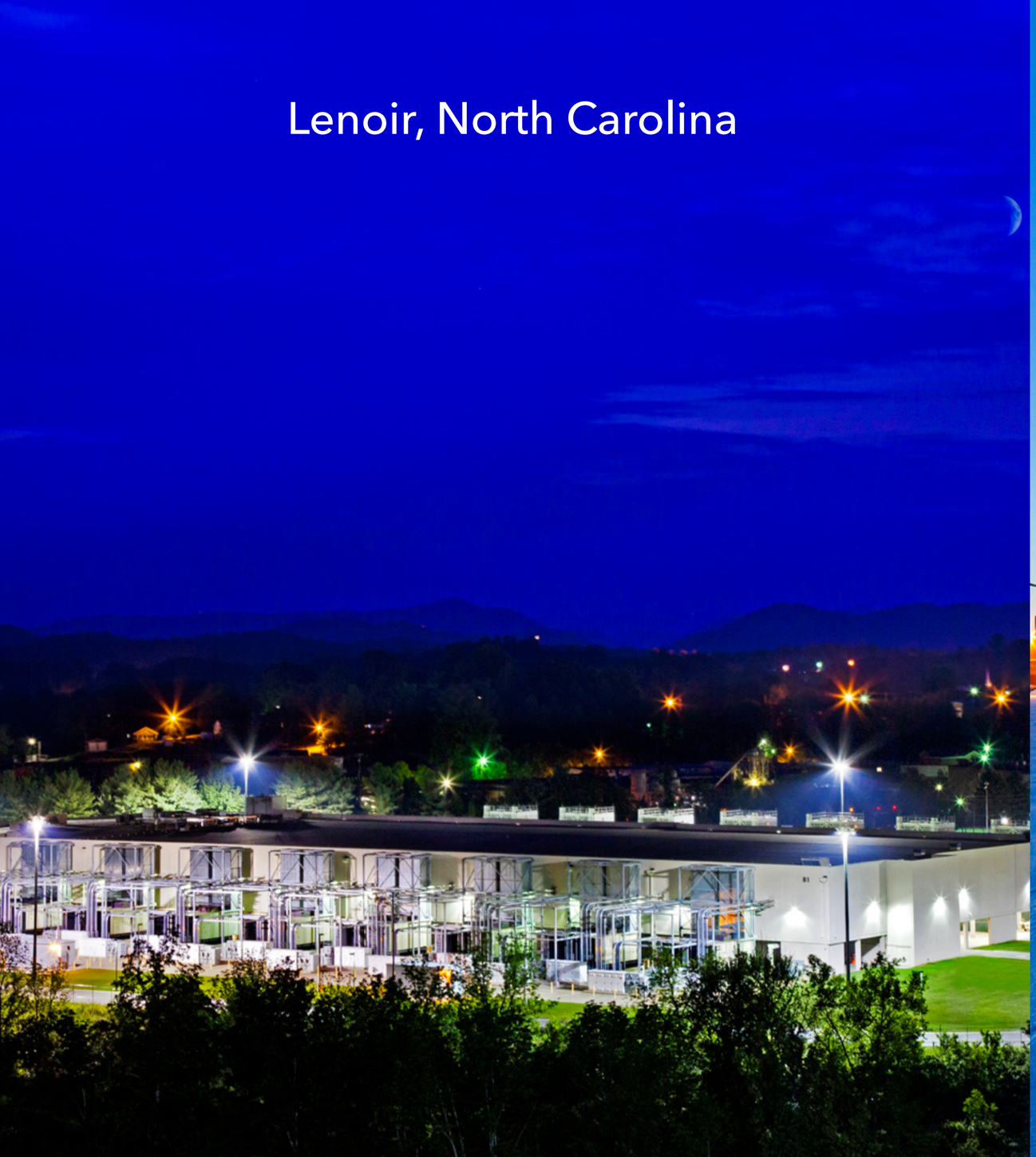
ACID

- Atomicity
- Consistency
- Isolation
- Durability

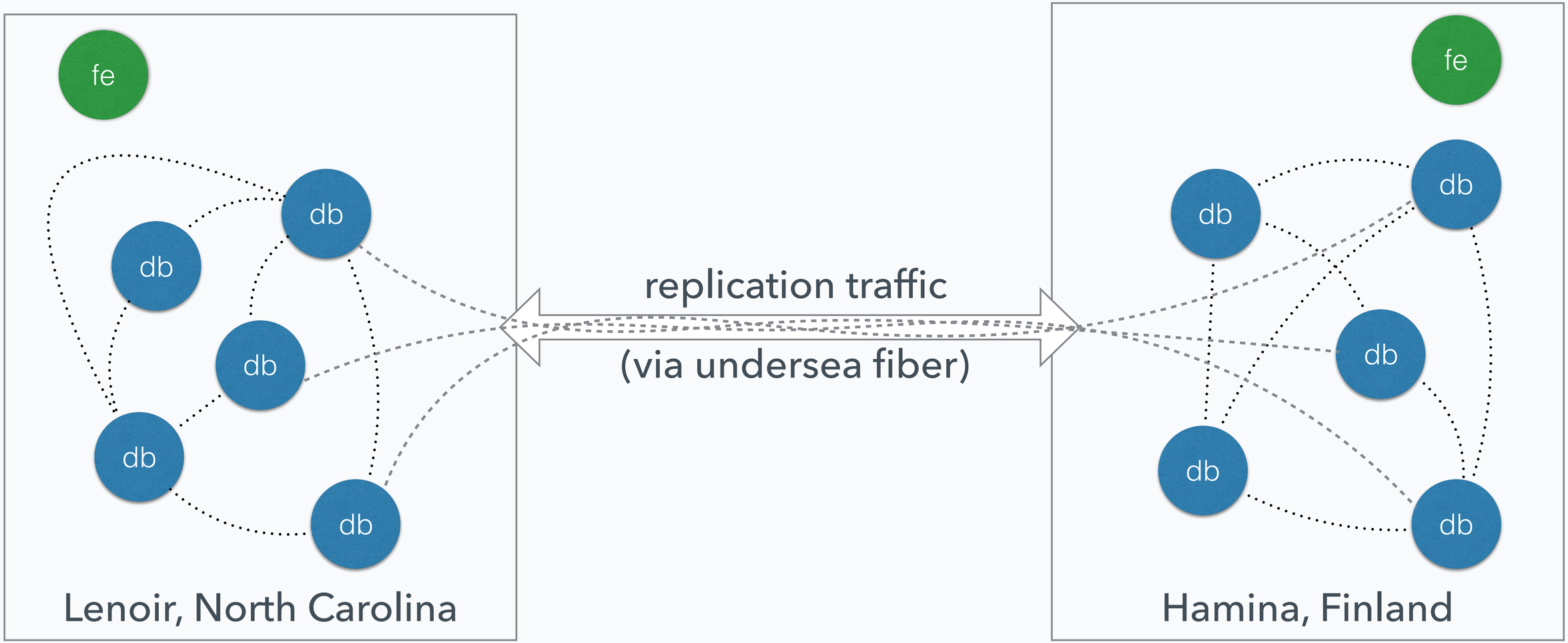
CAP Theorem

- Consistency: all clients see the same data
- Availability: each client can always read or write
- Partition Tolerance: system works even if network splits

- "Pick 2"

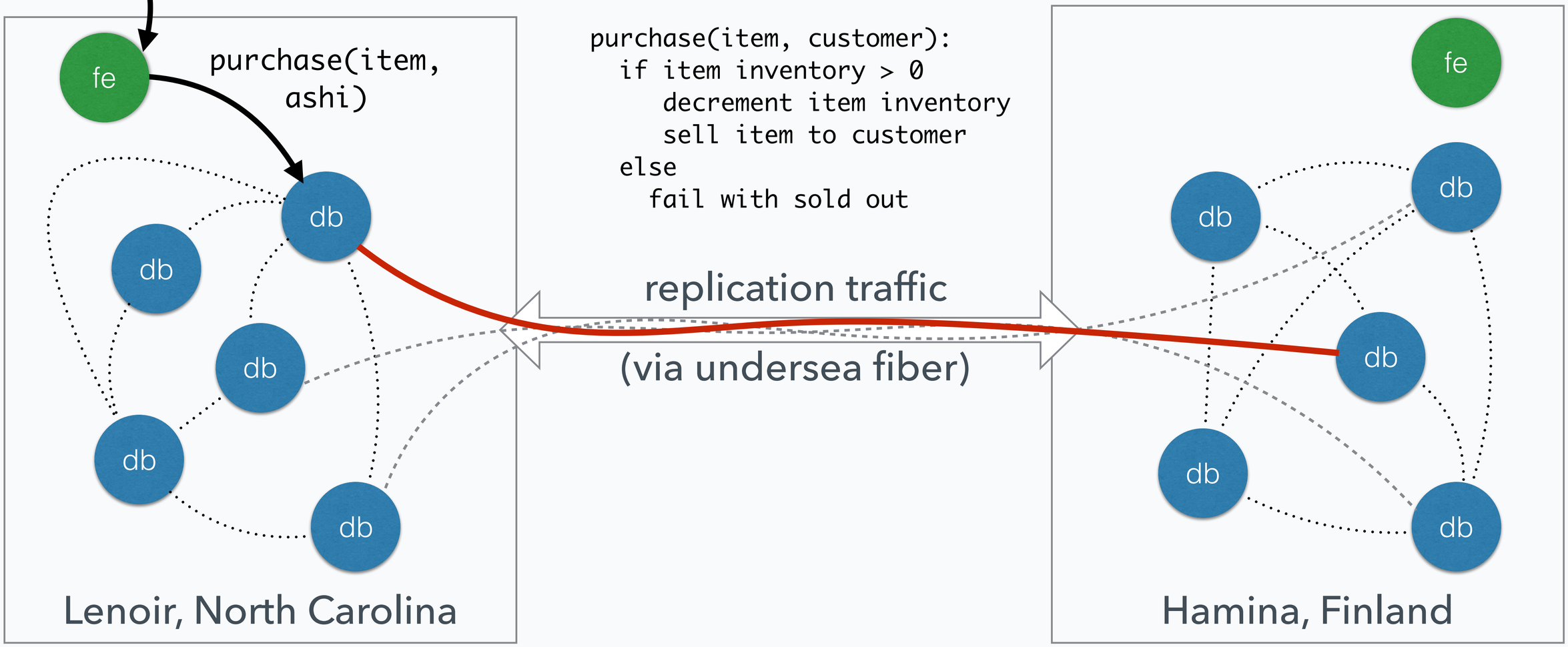

Lenoir, North Carolina

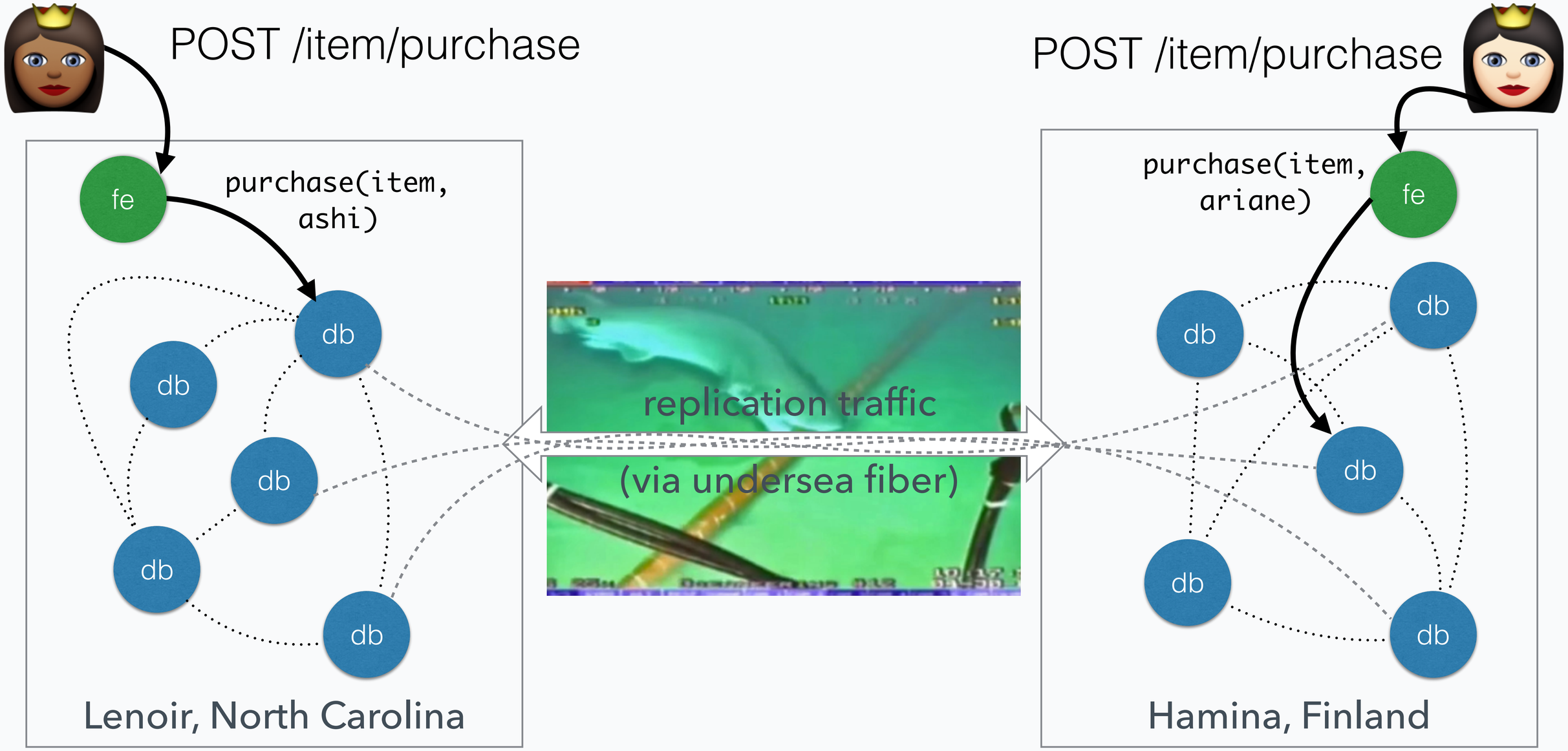
Hamina, Finland


fe
db
db
db
db
db
Lenoir, North Carolina
replication traffic
(via undersea fiber)
fe
db
db
db
db
db
db
Hamina, Finland

POST /item/purchase

```
purchase(item, customer):
  if item inventory > 0
     decrement item inventory
     sell item to customer
  else
    fail with sold out
```




POST /item/purchase
POST /item/purchase
fe
purchase(item, ashi)
db
db
db
db
db
Lenoir, North Carolina
replication traffic
(via undersea fiber)
purchase(item, ariane)
fe
db
db
db
db
db
Hamina, Finland

Tooling

Datazenit App

postgresql | hiburo.com | datazenit | local.dev | Feedback | User

root@localhost:3306 | Edit

test

Tables

categories

content

Columns

id
title
user_id
created_at
booly
gender
body
created_at_date
foo_bar
DBL
amount

cust_hist

emails

New database | New table

Filter | Query | Schema | Backups

Select id, title, body, user_id, created_at From content

Limit 20 Offset 0

Add | Condition | Sorting | History | Favorites | Save | Export | Execute

Content | Visualization

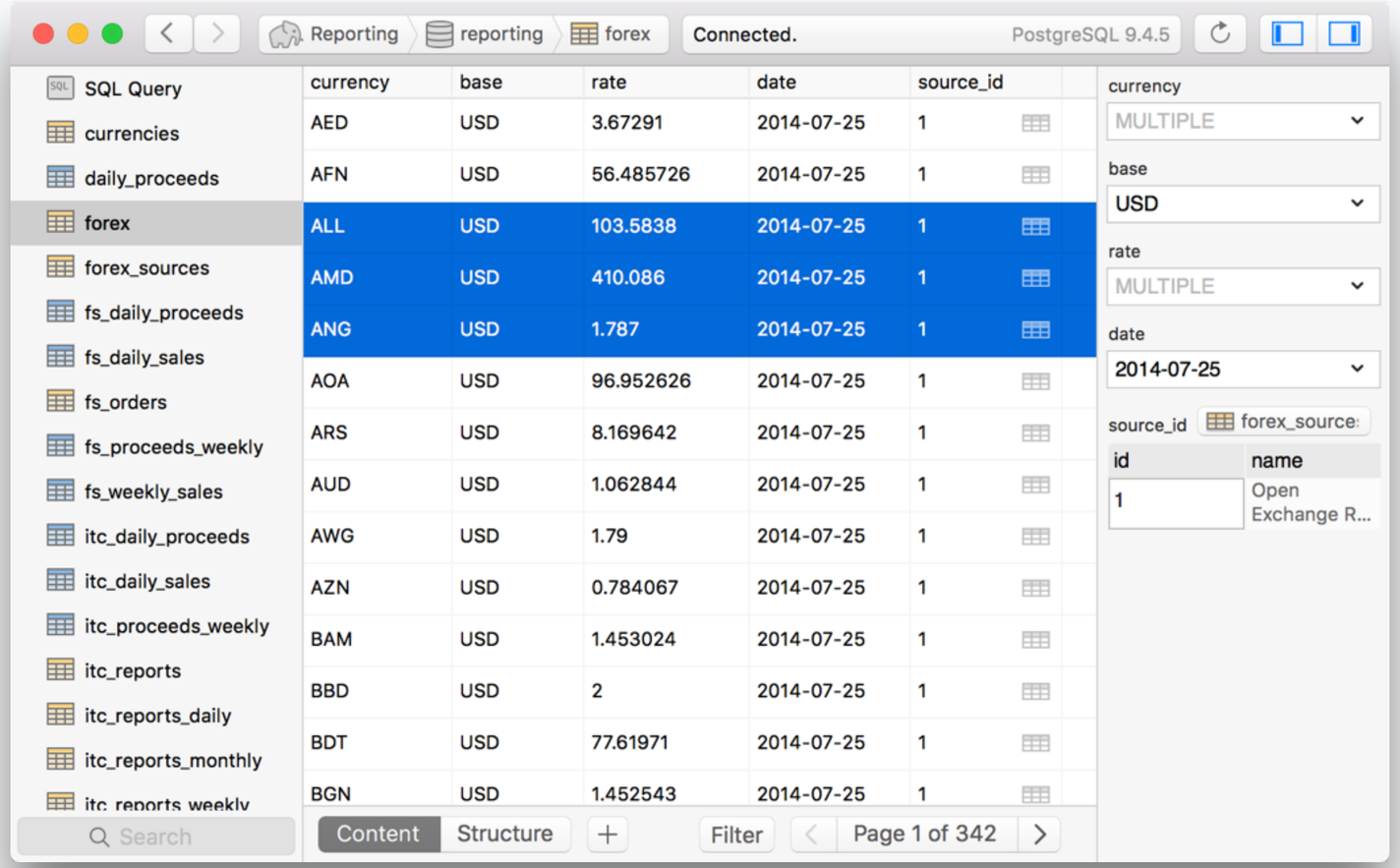
id	title	body	user_id	created_at
1	Optional actuating utilisation	Inventore quia et et eius. P...	535868	1983-11-11 00:01:22
2	Expanded dynamic superst...	Corporis et atque qui haru...	949966	1974-04-11 22:10:36
3	Fully-configurable real-time...	Eaque officia natus porro v...	44126916	2014-10-05 03:29:17
4	ineered dynamic architecture	Consequatur cumque duci...	0	2012-04-01 05:43:33
5	Customizable heuristic firm...	Est ipsa nobis et omnis. Qu...	89	1992-04-07 15:19:49
6	Total explicit collaboration	Ratione similique omnis au...	8	1978-01-24 10:44:35
7	Facetoface dynamic succe...	Nihil saepe reiciendis labor...	68030159	2006-12-26 20:38:34

Delete Rows returned: 20 Rows selected: 3 Tip: press cmd+a to select/deselect all rows

Datazenit

Tooling

Postico



Sequelize

- Sequelize is an Object-Relational Mapper (ORM)
- Access SQL databases from Node.js
- Sequelize features:
  - Schema modeling/validation
  - Data casting (convert SQL types to JS types)
  - Query building
  - Hooks (code that runs pre/post save/delete/update)
  - Class and instance methods of models
  - Getters, setters, and virtual fields

Tables

Models

+

=

+

Rows

Instances

Sequelize Basics

- Make a Model (interactive blueprint object)
- Extend the Model with Hooks, Class & Instance Methods, Virtuals, etc.
- Connect/sync the completed Model to an *actual* table in an *actual* SQL database
- Use the Model (Table) to create/find Instances (Table)
- Use the Instances to save/update/delete

Create a Model

```
var Sequelize = require('sequelize');
var db = new Sequelize('postgres://localhost:5432/twitter');

var User = db.define('User', {
  name: Sequelize.String,
  pictureUrl: Sequelize.String
});
```

Sync Model to Table

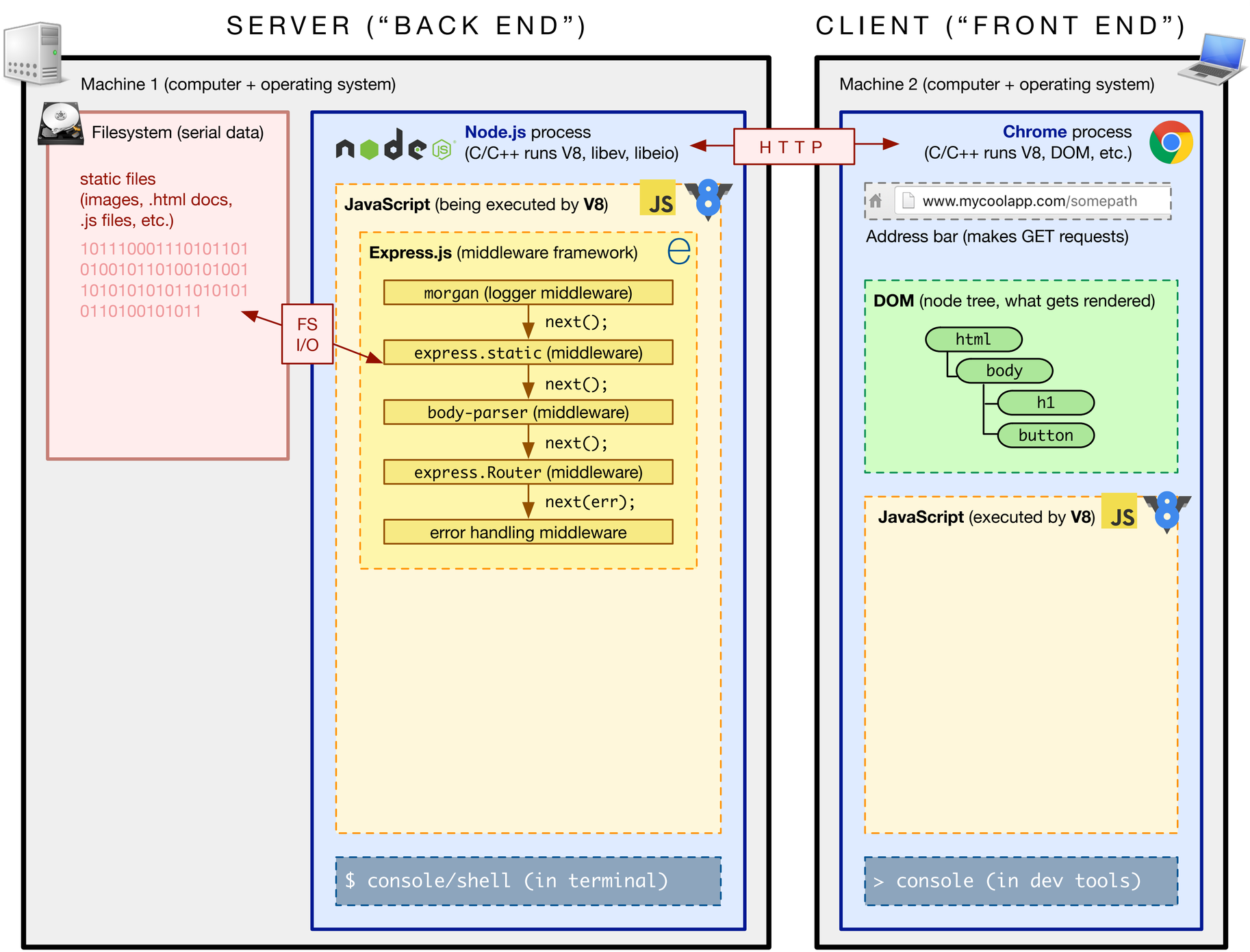
```
User.sync().then(…);
```

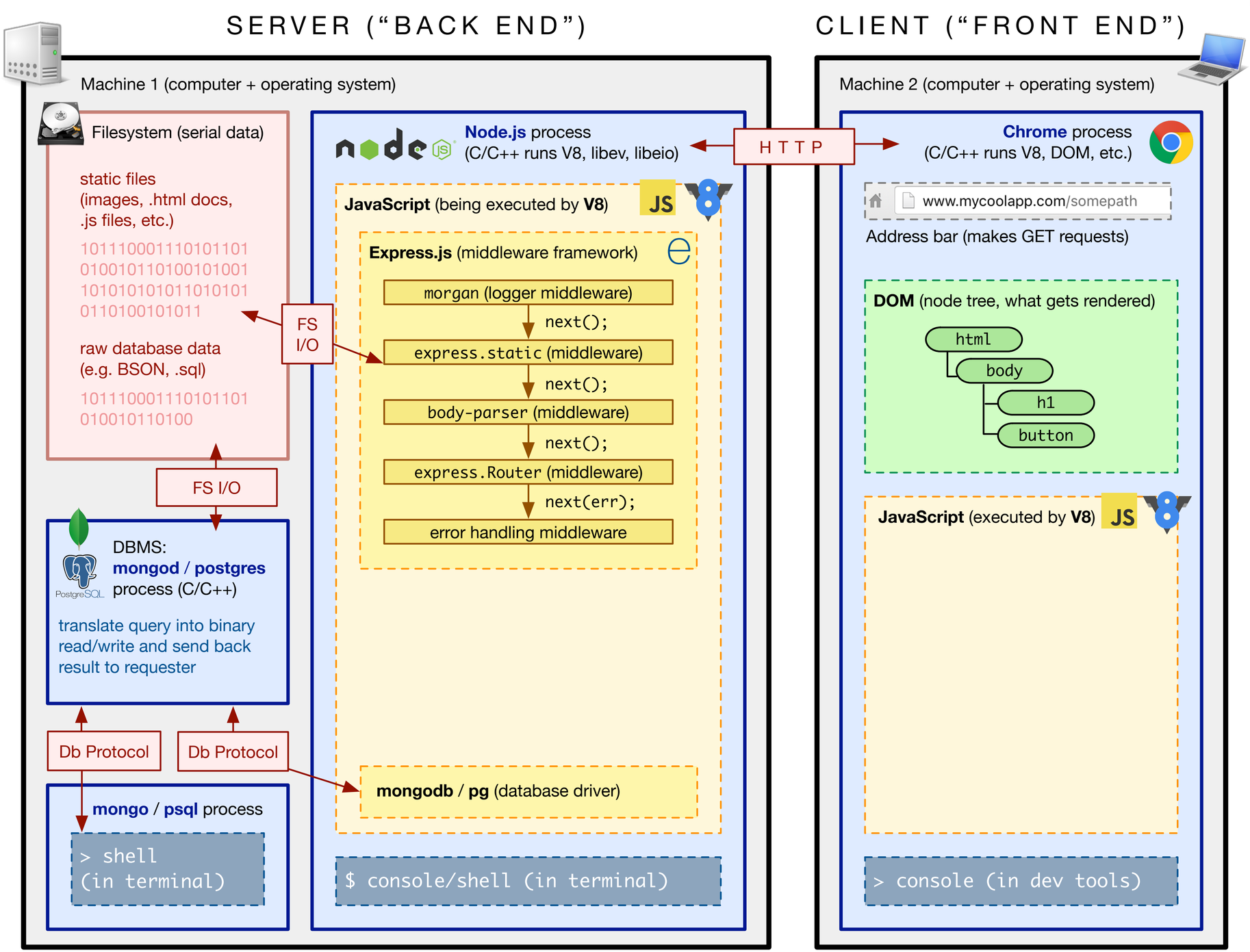
Model & Instance Usage

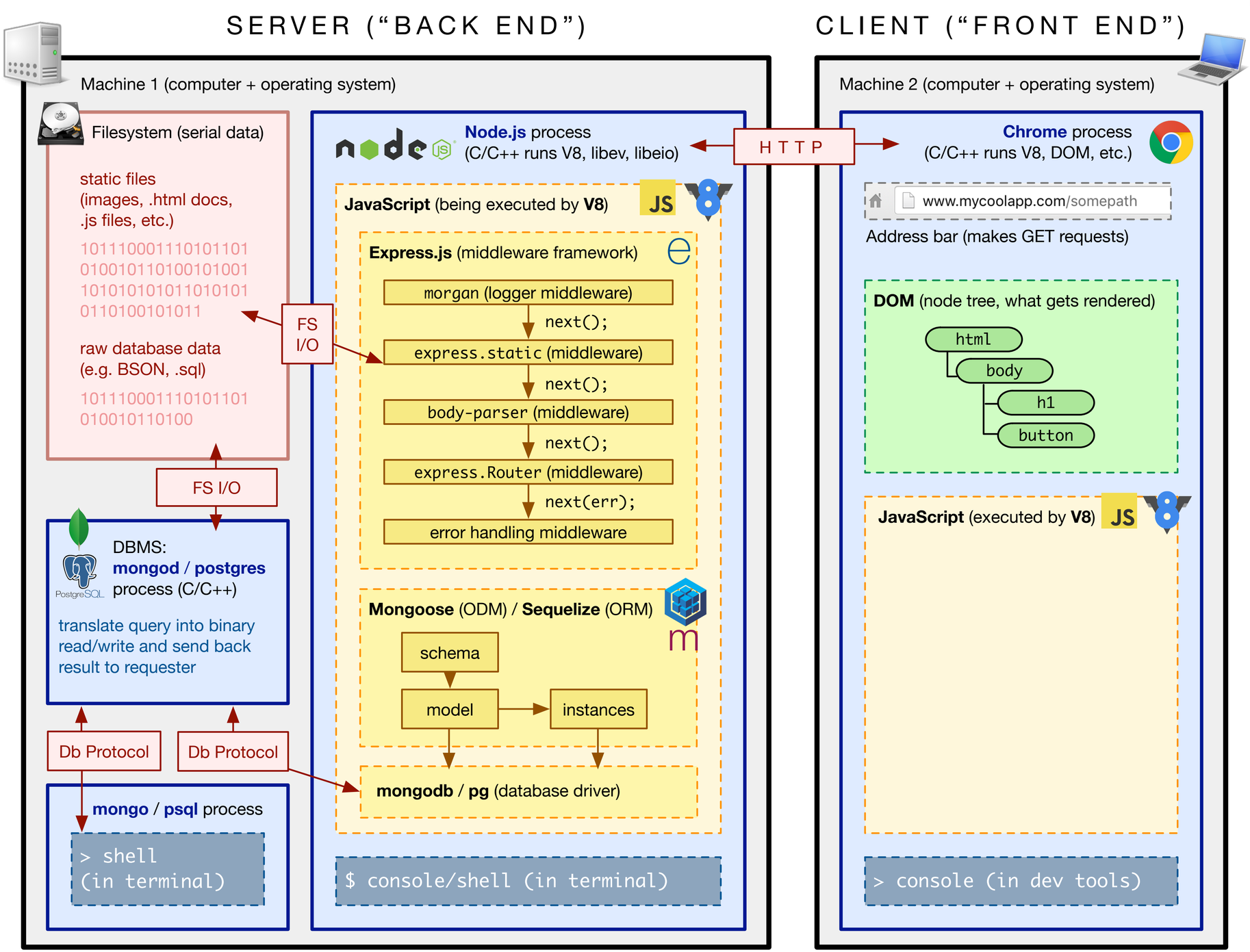
```
var aPerson = User.build({
  name: "David",
  pictureUrl: "http://fillmurrary.com/10/10"
});

aPerson.save()
.then(…);

User.findAll()
.then(…);
```

Sequelize

- Lives inside Node.js process
- Knows how to communicate to various SQL DBMSs, including PostgreSQL


SERVER ("BACK END")
CLIENT ("FRONT END")
Machine 1 (computer + operating system)
Filesystem (serial data)
static files
(images, .html docs,
.js files, etc.)
101110001110101101
010010110100101001
101010101011010101
0110100101011
FS
I/O
Node.js process
(C/C++ runs V8, libev, libeio)
JavaScript (being executed by V8)
JS
Express.js (middleware framework)
morgan (logger middleware)
next();
express.static (middleware)
next();
body-parser (middleware)
next();
express.Router (middleware)
next(err);
error handling middleware
$ console/shell (in terminal)
HTTP
Machine 2 (computer + operating system)
Chrome process
(C/C++ runs V8, DOM, etc.)
www.mycoolapp.com/somepath
Address bar (makes GET requests)
DOM (node tree, what gets rendered)
html
body
h1
button
JavaScript (executed by V8)
JS
> console (in dev tools)


SERVER ("BACK END")
CLIENT ("FRONT END")
Machine 1 (computer + operating system)
Filesystem (serial data)
static files
(images, .html docs,
.js files, etc.)
101110001110101101
010010110100101001
101010101011010101
0110100101011
raw database data
(e.g. BSON, .sql)
101110001110101101
010010110100
FS I/O
FS I/O
DBMS:
mongod / postgres
process (C/C++)
PostgreSQL
translate query into binary
read/write and send back
result to requester
Db Protocol
Db Protocol
mongo / psql process
> shell
(in terminal)
Node.js process
(C/C++ runs V8, libev, libeio)
JavaScript (being executed by V8)
JS
Express.js (middleware framework)
morgan (logger middleware)
next();
express.static (middleware)
next();
body-parser (middleware)
next();
express.Router (middleware)
next(err);
error handling middleware
mongodb / pg (database driver)
$ console/shell (in terminal)
HTTP
Machine 2 (computer + operating system)
Chrome process
(C/C++ runs V8, DOM, etc.)
www.mycoolapp.com/somepath
Address bar (makes GET requests)
DOM (node tree, what gets rendered)
html
body
h1
button
JavaScript (executed by V8)
JS
> console (in dev tools)


SERVER ("BACK END")
CLIENT ("FRONT END")
Machine 1 (computer + operating system)
Filesystem (serial data)
static files
(images, .html docs,
.js files, etc.)
101110001110101101
010010110100101001
101010101011010101
0110100101011
raw database data
(e.g. BSON, .sql)
101110001110101101
010010110100
FS I/O
FS I/O
DBMS:
mongod / postgres
process (C/C++)
PostgreSQL
translate query into binary
read/write and send back
result to requester
Db Protocol
Db Protocol
mongo / psql process
> shell
(in terminal)
Node.js process
(C/C++ runs V8, libev, libeio)
JavaScript (being executed by V8)
JS
Express.js (middleware framework)
morgan (logger middleware)
next();
express.static (middleware)
next();
body-parser (middleware)
next();
express.Router (middleware)
next(err);
error handling middleware
Mongoose (ODM) / Sequelize (ORM)
schema
model
instances
mongodb / pg (database driver)
$ console/shell (in terminal)
HTTP
Machine 2 (computer + operating system)
Chrome process
(C/C++ runs V8, DOM, etc.)
www.mycoolapp.com/somepath
Address bar (makes GET requests)
DOM (node tree, what gets rendered)
html
body
h1
button
JavaScript (executed by V8)
JS
> console (in dev tools)

Wikistack

- Build a Wikipedia clone
- Walk you through installing and using sequelize
- Application of <u>everything</u> we've learned so far