

Database Management System (DBMS)

- How to define records (Data Definition)
- CRUD (Create / Read / Update / Delete)
- Performant (B-Trees, etc.)
- Administration
- In short: handle the file system intelligently, without re-inventing the wheel for every project.

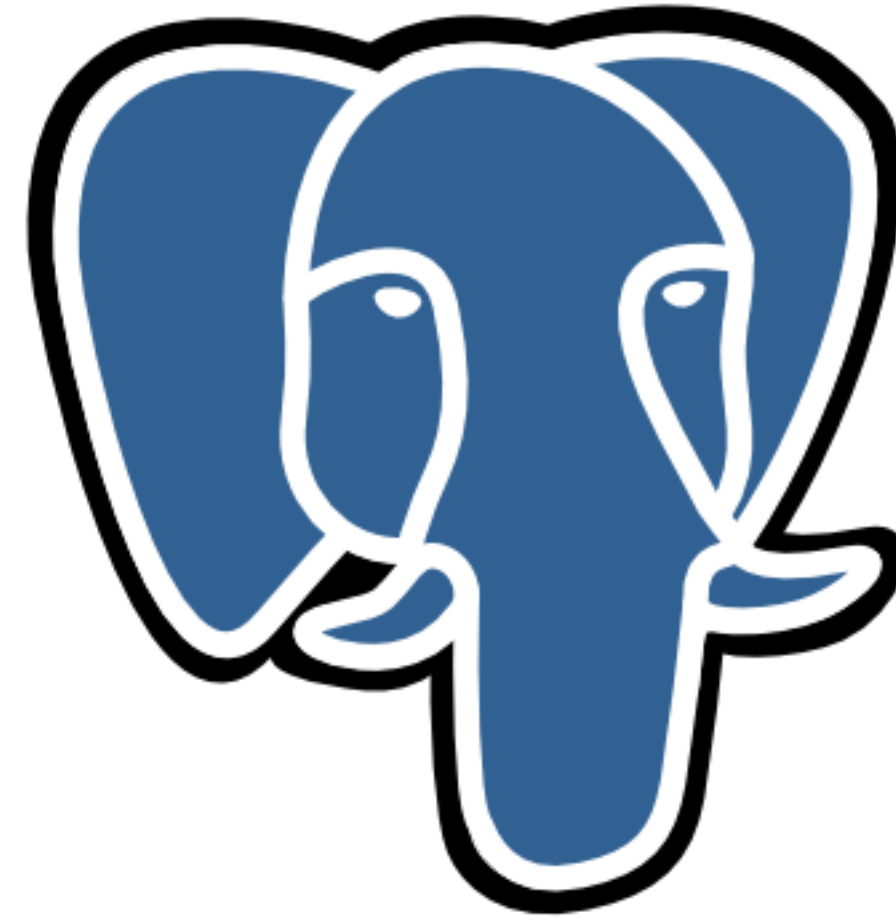
Progression of Databases

- **Navigational (< 1970s)**
 - More common during tape era; entries had references to next entries.
- **Relational (> 1970s)**
 - Based on relational (table-based) logic, see E.F. Codd.
- **NoSQL (> 2000s)**
 - "Not only SQL" — document storage, for example.



Some well-known rDBMSs





PostgreSQL

History of PostgreSQL

- 1970s at UC Berkley:
INteractive **G**raphics **R**etrieval **S**ystem (INGRES)
- 1980s: POSTGRES ("Post-Ingres")
- 1995: POSTQUEL and Postgres95.
 - `monitor -> psql`
- 1996: Adopted by the open source community
 - Ongoing: stability, testing, documentation, new features
 - PostgreSQL

Why PostgreSQL?

- Advanced, powerful, and popular
- Rapid open source development
- Highly extensible (stored procedures)
- Deep SQL standards compliance
- NoSQL ("Not Only SQL"), objective support
- Excellent transactions / ACID reliability; focus on integrity
- Remote (SQLite is embedded)
- Multi-user management / administration

Tooling

- psql
- CLI

```
psql [local]:5432 glebec # ~ \l
List of databases

```

Name	Owner	Encoding	Collate	Ctype	Access privileges
assessmentexpresssequeliza	fullstack	UTF8	en_US.UTF-8	en_US.UTF-8	
author	glebec	UTF8	en_US.UTF-8	en_US.UTF-8	
checkpoint_angular	glebec	UTF8	en_US.UTF-8	en_US.UTF-8	
checkpoint_express_review	glebec	UTF8	en_US.UTF-8	en_US.UTF-8	
glebec	glebec	UTF8	en_US.UTF-8	en_US.UTF-8	
juke	glebec	UTF8	en_US.UTF-8	en_US.UTF-8	
postgres	glebec	UTF8	en_US.UTF-8	en_US.UTF-8	
sequelizecheckpoint	fullstack	UTF8	en_US.UTF-8	en_US.UTF-8	
template0	glebec	UTF8	en_US.UTF-8	en_US.UTF-8	=c/glebec glebec=CTc/glebec
template1	glebec	UTF8	en_US.UTF-8	en_US.UTF-8	=c/glebec glebec=CTc/glebec
tripolanner	glebec	UTF8	en_US.UTF-8	en_US.UTF-8	
twitterdb	glebec	UTF8	en_US.UTF-8	en_US.UTF-8	
wikistack	glebec	UTF8	en_US.UTF-8	en_US.UTF-8	

```
(13 rows)

psql [local]:5432 glebec # ~ \c author
You are now connected to database "author" as user "glebec".

psql [local]:5432 glebec # author \dt
List of relations

```

Schem	Name	Type	Owner
public	stories	table	glebec
public	users	table	glebec

```
(2 rows)

psql [local]:5432 glebec # author
```

Tooling

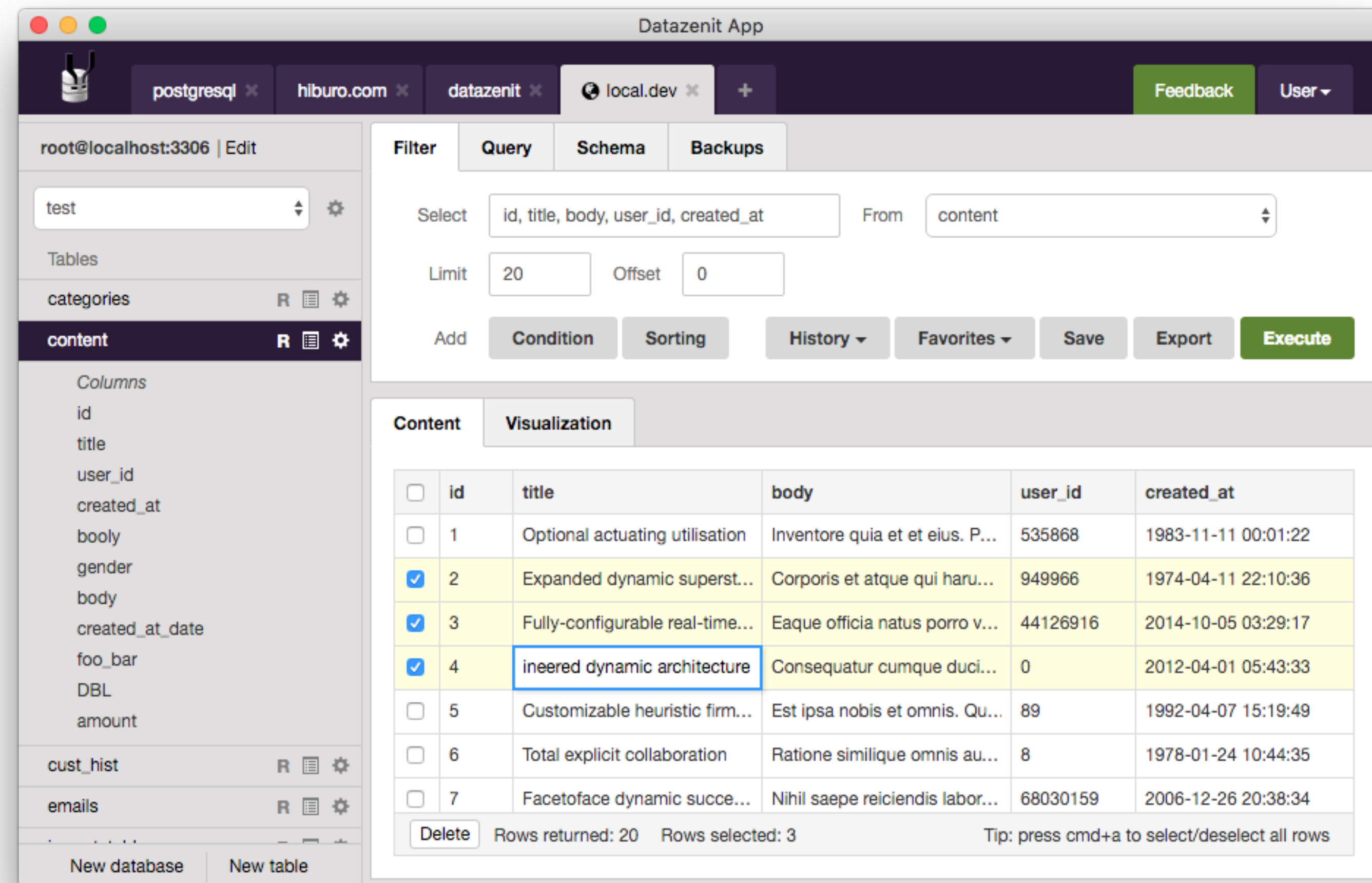
- Postico
- Free

The screenshot shows the Postico application interface. At the top, there's a navigation bar with tabs for 'Reporting', 'reporting', and 'forex'. The 'forex' tab is selected. Below the navigation bar, there's a status bar showing 'Connected.' and 'PostgreSQL 9.4.5'. On the left side, there's a sidebar with a list of tables: 'SQL Query', 'currencies', 'daily_proceeds', 'forex', 'forex_sources', 'fs_daily_proceeds', 'fs_daily_sales', 'fs_orders', 'fs_proceeds_weekly', 'fs_weekly_sales', 'itc_daily_proceeds', 'itc_daily_sales', 'itc_proceeds_weekly', 'itc_reports', 'itc_reports_daily', 'itc_reports_monthly', and 'itc_reports_weekly'. The 'forex' table is selected. The main area displays a table with the following columns: 'currency', 'base', 'rate', 'date', and 'source_id'. The table contains 15 rows of data, with the first three rows highlighted in blue. On the right side, there's a panel with filters for 'currency', 'base', 'rate', and 'date'. Below these filters, there's a section for 'source_id' with a dropdown menu showing 'forex_source!'. At the bottom, there's a search bar and a pagination bar showing 'Page 1 of 342'.

currency	base	rate	date	source_id
AED	USD	3.67291	2014-07-25	1
AFN	USD	56.485726	2014-07-25	1
ALL	USD	103.5838	2014-07-25	1
AMD	USD	410.086	2014-07-25	1
ANG	USD	1.787	2014-07-25	1
AOA	USD	96.952626	2014-07-25	1
ARS	USD	8.169642	2014-07-25	1
AUD	USD	1.062844	2014-07-25	1
AWG	USD	1.79	2014-07-25	1
AZN	USD	0.784067	2014-07-25	1
BAM	USD	1.453024	2014-07-25	1
BBD	USD	2	2014-07-25	1
BDT	USD	77.61971	2014-07-25	1
BGN	USD	1.452543	2014-07-25	1

Tooling

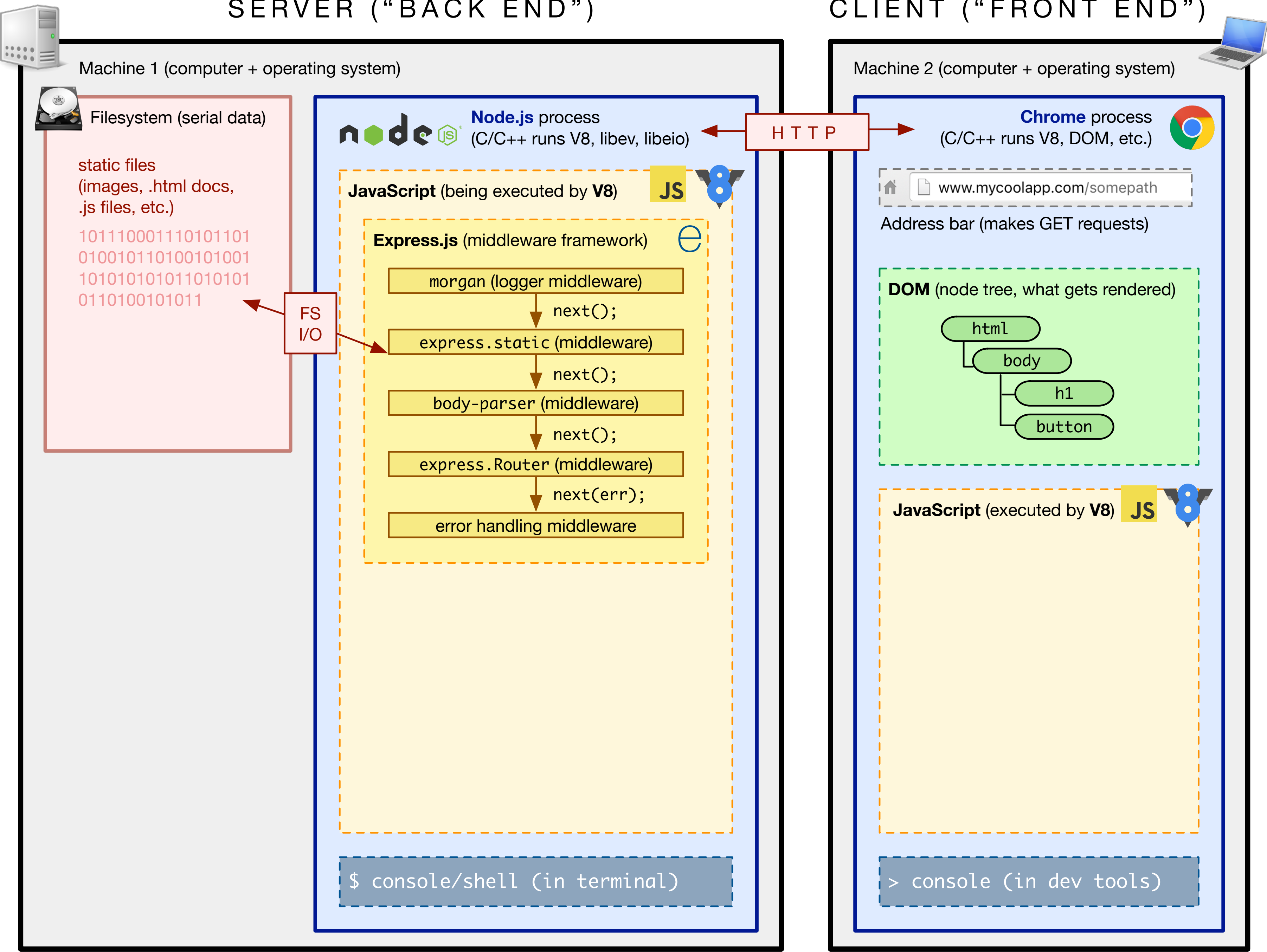
- Datanenit
- \$\$\$



Etc.

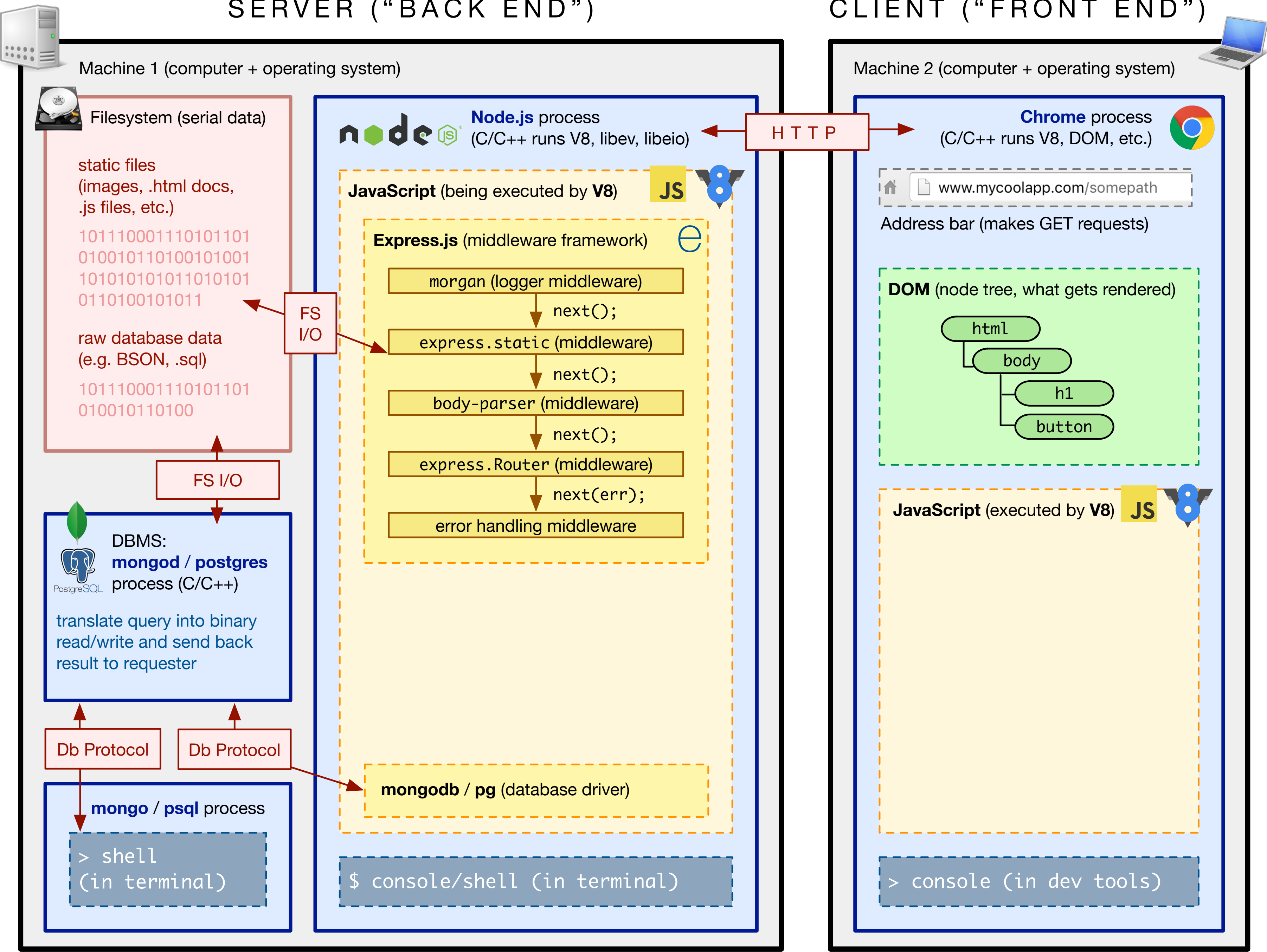
SERVER ("BACK END")

CLIENT ("FRONT END")



SERVER ("BACK END")

CLIENT ("FRONT END")



postgres process



- The rDBMS itself; a *daemon* (background process)
- Waits for incoming SQL
- Knows how to read/write to disk in a performant way
- Sends back results

**Where does the "incoming
SQL" come from?**

Query sources

- **psql CLI**
 - human input as text
- **GUI like Postico, Datazenit**
 - human actions turned into SQL queries
- **...and other applications**
 - "somehow" communicate with the postgres process

How to transmit SQL text to app?
How can postgres be "waiting for SQL"?
And how do the results get "sent back"?

Postgres is a TCP server!



- Listening on a TCP port (5432 by default) for *requests*
- Does disk access
- Sends back a TCP *response* to the *client* that made the requests

**OK, Postgres is a TCP server.
Is it... HTTP?**

Postgres uses the postgres:// protocol

	Transport Protocol	Message Protocol	Content Type
Node + Express	TCP/IP	http://	Anything: HTML, JSON, XML, TXT, etc.
Postgres	TCP/IP	postgres://	SQL

For HTTP clients, the TCP/IP was handled for you by the browser or Node. How can our JS app communicate with the postgres server?

*“Let's implement the postgres protocol in
JavaScript ourselves!”*

– AMBITIOUS MCOVERKILL

<https://www.postgresql.org/docs/current/static/protocol.html>

*“On second thought...
has anyone done this for us?”*

– SANEY MCREASONABLE

Node-postgres

- **npm library:** `npm install pg --save`
- *database driver*
- implements the postgres protocol in a Node module (JS!)
- Gives us a `client` object that we can pass SQL to
- Asynchronously talks via postgres protocol / TCP to postgres
- gives us a callback with `rows` array of resulting table

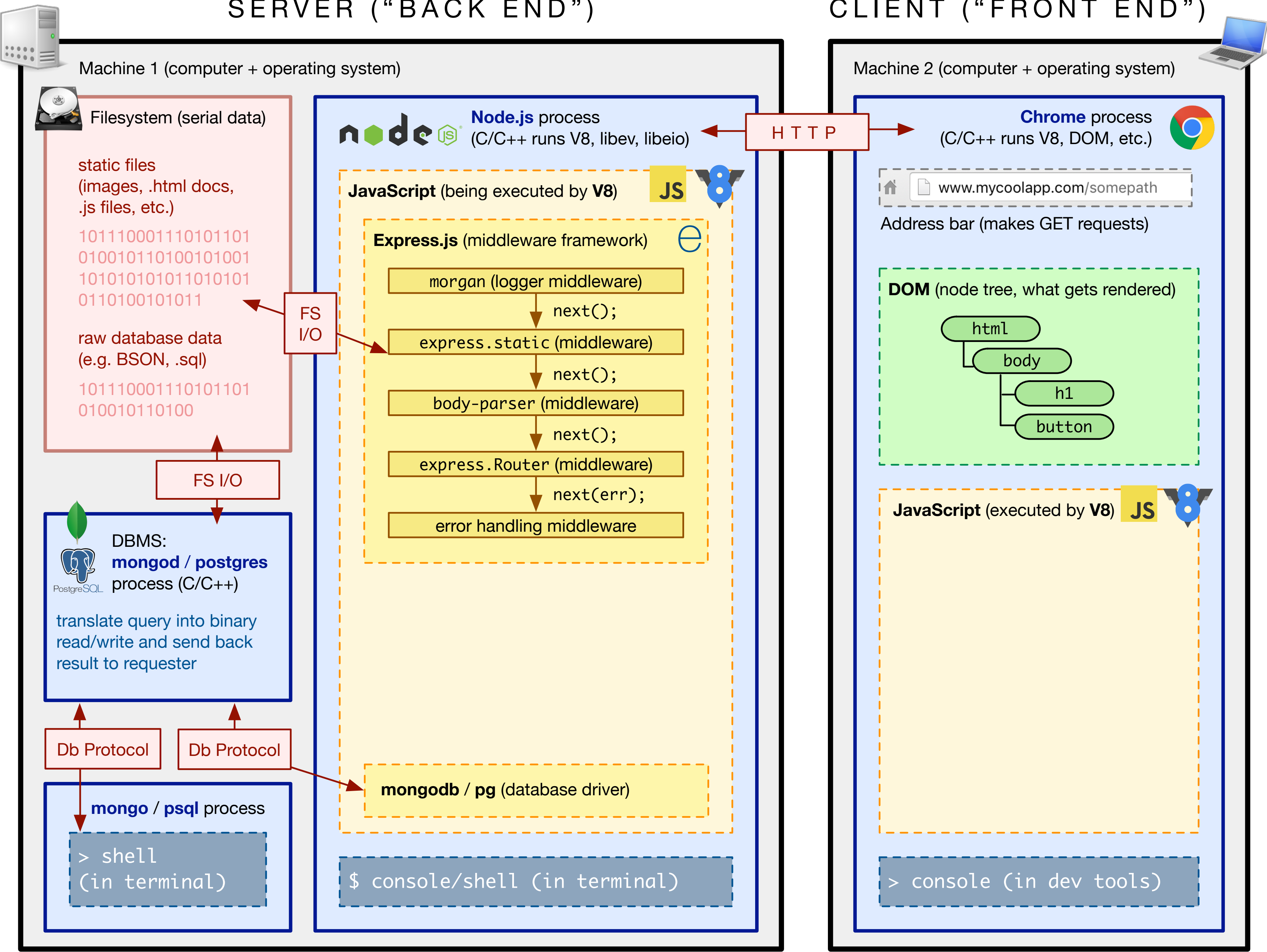


Example

```
client.query('SELECT * FROM users', function (err, data) {  
  if (err) return console.error(err);  
  data.rows.forEach(function (rowObject) {  
    console.log(rowObject); // { name: 'Claire' }  
  });  
});
```

SERVER ("BACK END")

CLIENT ("FRONT END")



Final note: `returning`

- SQL comes in slightly different "dialects" depending on your RDBMS of choice (SQLite, MySQL, PostgreSQL etc.)
- PostgreSQL has a very convenient keyword `returning`
- Used during INSERT, UPDATE
- Returns the row(s) inserted/updated
- May come in handy during workshop, so check it out!