Databases and DBMS

Database definitions

A **database** (**DB**) is an organized collection of data.

- In our precedent assignment , we used a JSON file to store our figures characteristics.
- By this definition, the JSON file can be considered a database.

A **database management system** (**DBMS**) is software that handles the storage, retrieval, and updating of data.

- Examples: MongoDB, MySQL, PostgreSQL, etc.
- Usually when people say "**database**", they mean data that is managed through a DBMS.

Why use a database/DBMS

Why use a DBMS instead of saving to a JSON file?

- **fast**: can search/filter a database quickly compared to a file
- **scalable**: can handle very large data sizes
- **reliable**: mechanisms in place for secure transactions, backups, etc.
- **built-in features**: can search, filter data, combine data from multiple sources
- **abstract**: provides layer of abstraction between stored data and app(s)
    - Can change **where** and **how** data is stored without needing to change the code that connects to the database.

Why use a database/DBMS

Why use a DBMS instead of saving to a JSON file?

- Also: Some services like Heroku will not permanently save files, so using `fs` or `fs-extra` **will not work**

MongoDB

MongoDB

MongoDB: A popular open-source DBMS

- A *document-oriented* database as opposed to a *relational* database

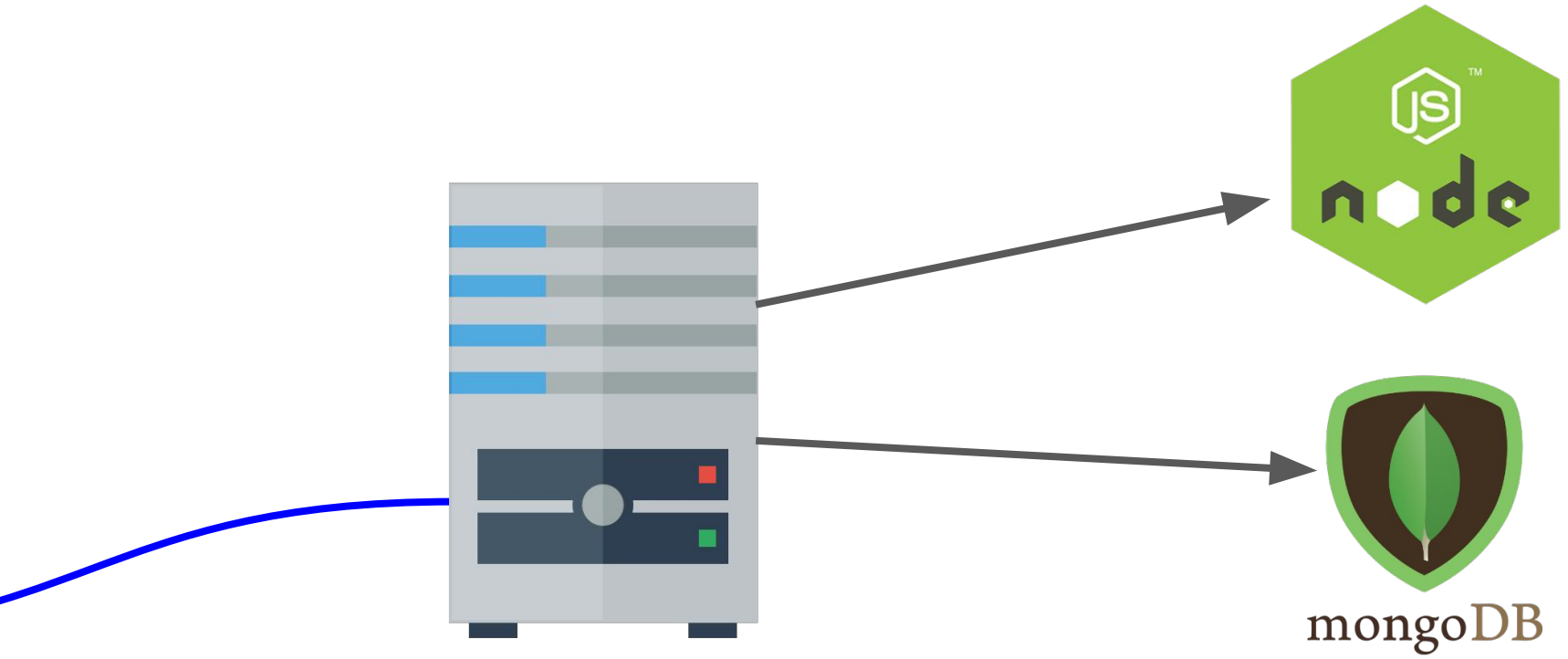
Relational database:

Name	School	Employer	Occupation
Lori	null	Self	Entrepreneur
Malia	Harvard	null	null

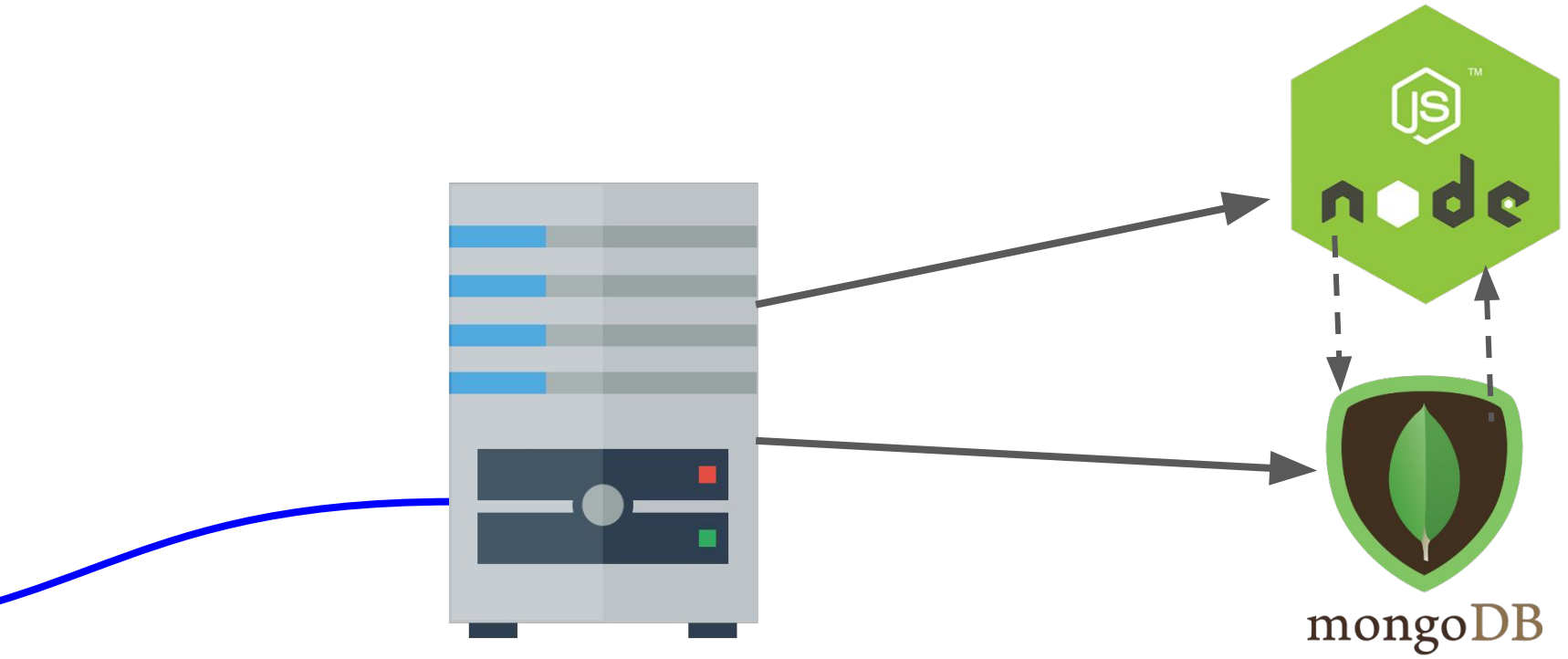
Relational databases have fixed schemas; document-oriented databases have flexible schemas

Document-oriented DB:

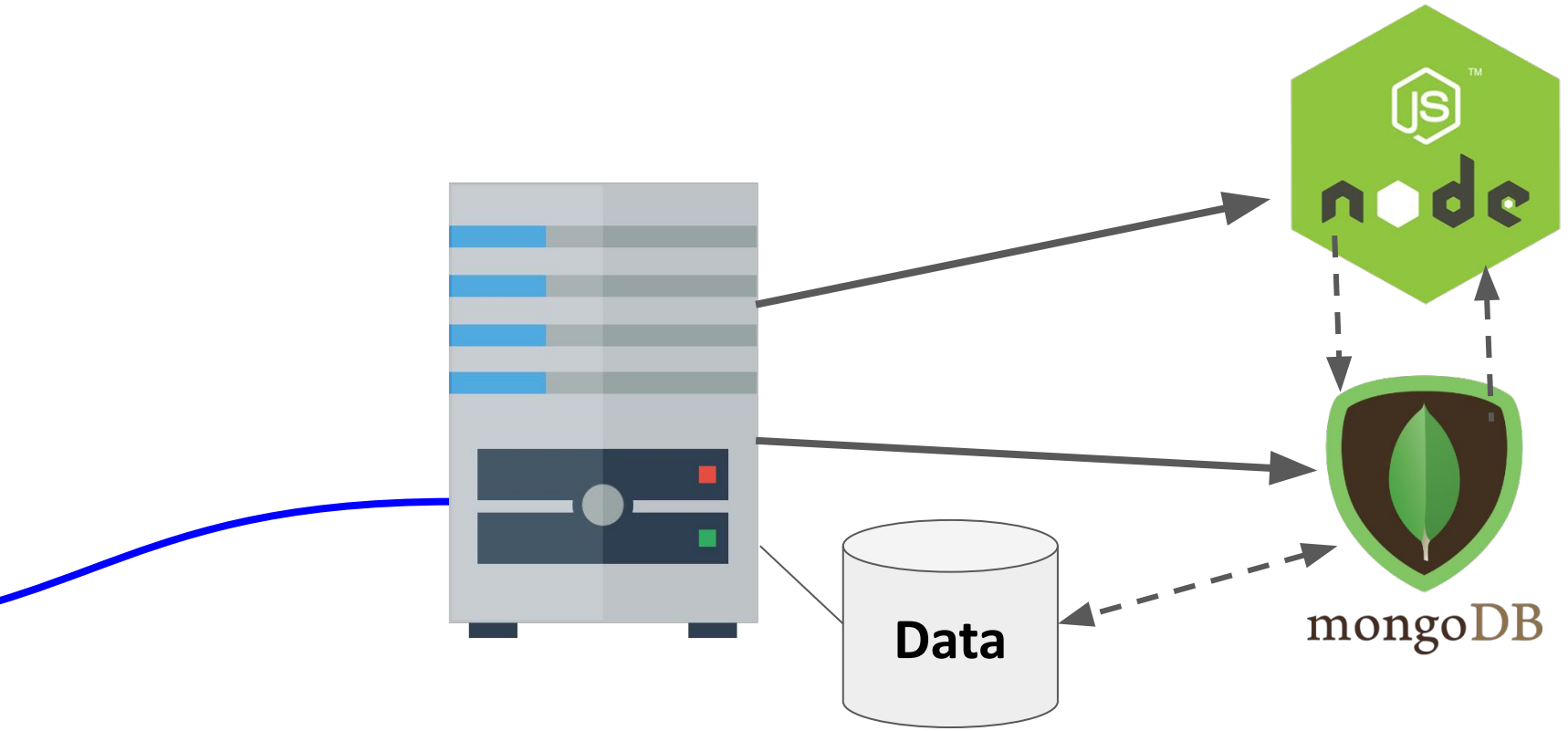
```
{
  name: "Lori",
  employer: "Self",
  occupation: "Entrepreneur"
}
{
  name: "Malia",
  school: "Harvard"
}
```



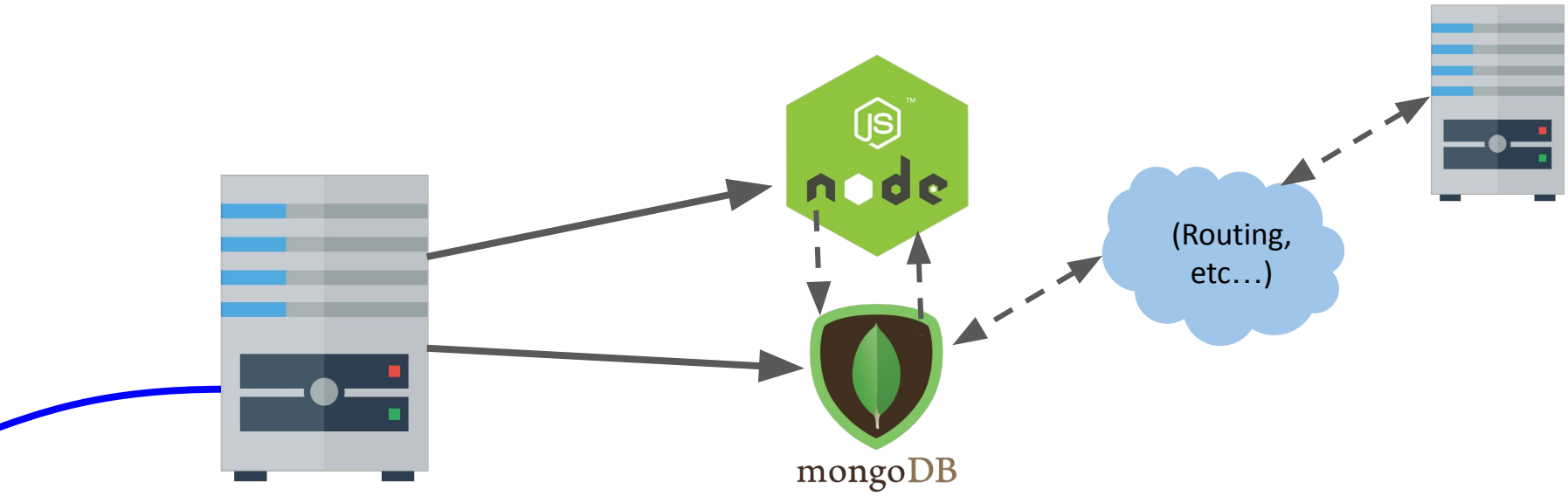
MongoDB is another **software program** running on the computer, alongside our NodeJS server program.
It is also known as the **MongoDB server**.



There are MongoDB libraries we can use in NodeJS to communicate with the MongoDB Server, which reads and writes data in the database it manages.

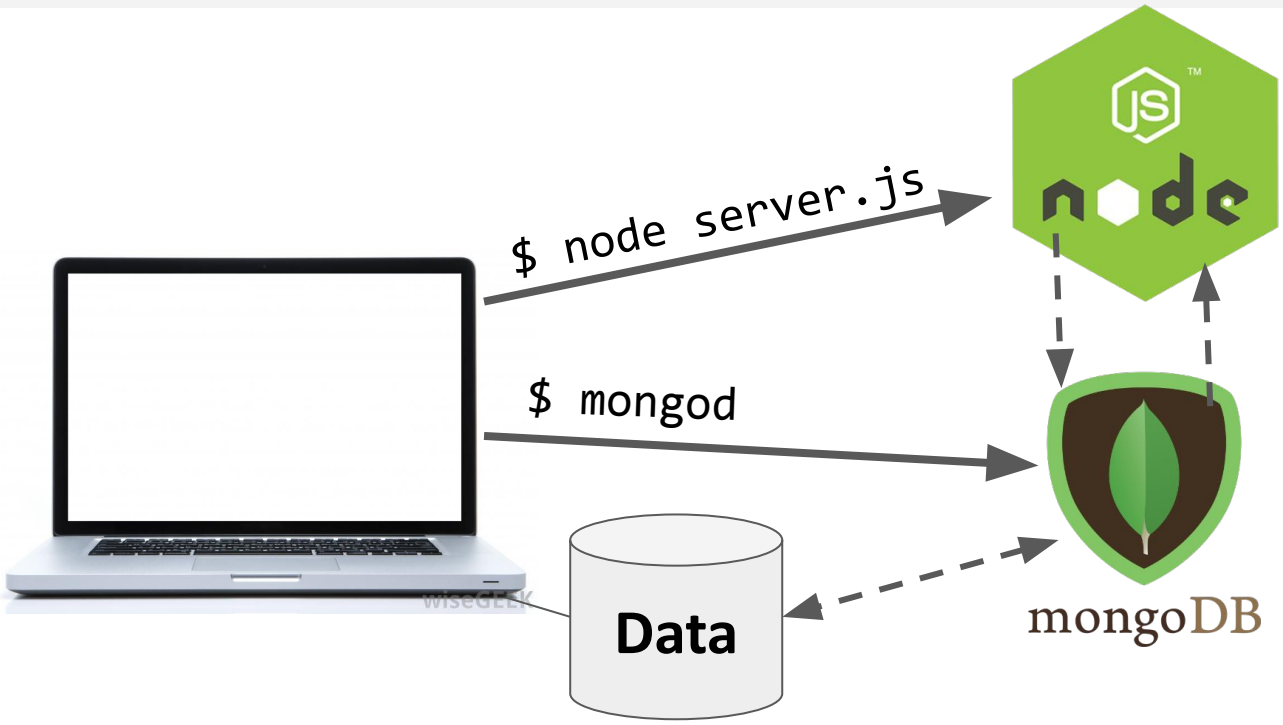


The database the MongoDB Server manages might be local to the server computer...



Or it could be stored on other server computer(s)
("cloud storage").

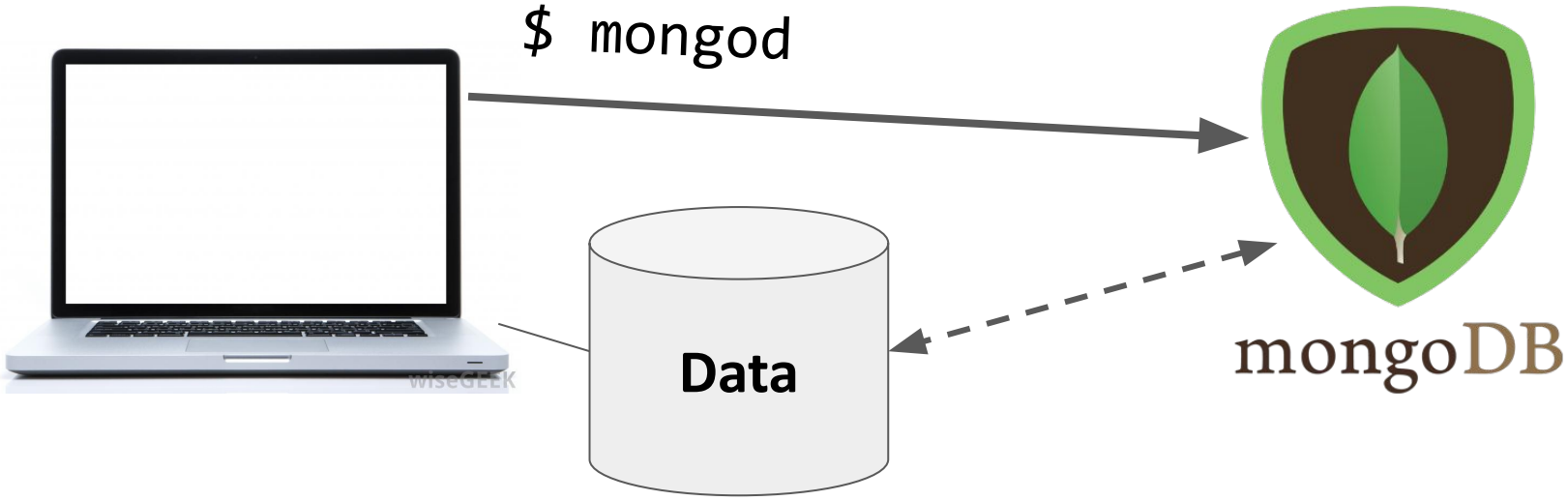
System overview



For development, we will have 2 processes running:

- node will run the main server program on port 3000
- **mongod will run the database server on a port 27017**

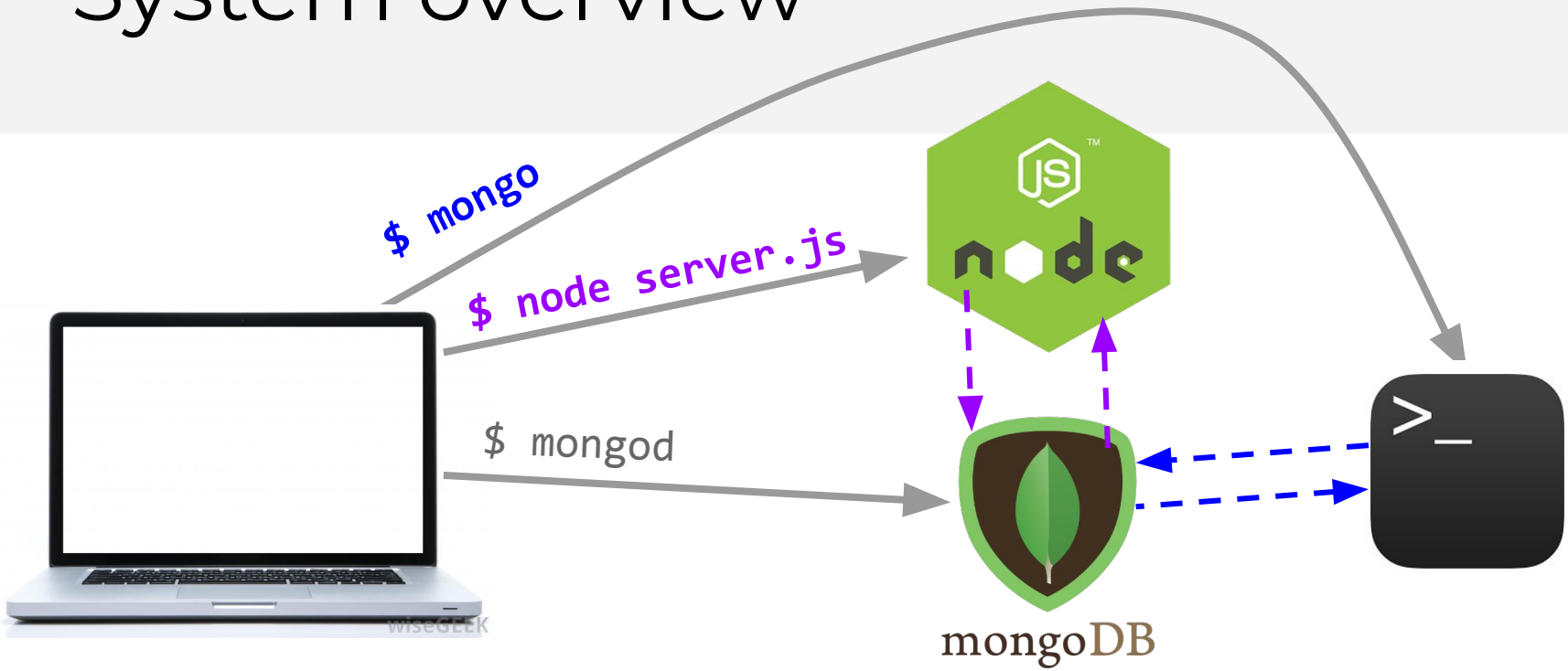
System overview



The mongod server will be bound to port 27017 by default

- The mongod process will be listening for messages to manipulate the database: insert, find, delete, etc.

System overview



We will be using two ways of communicating to the MongoDB server:

- NodeJS libraries
- mongo command-line tool

MongoDB concepts

Database:

- A container of MongoDB **collections**

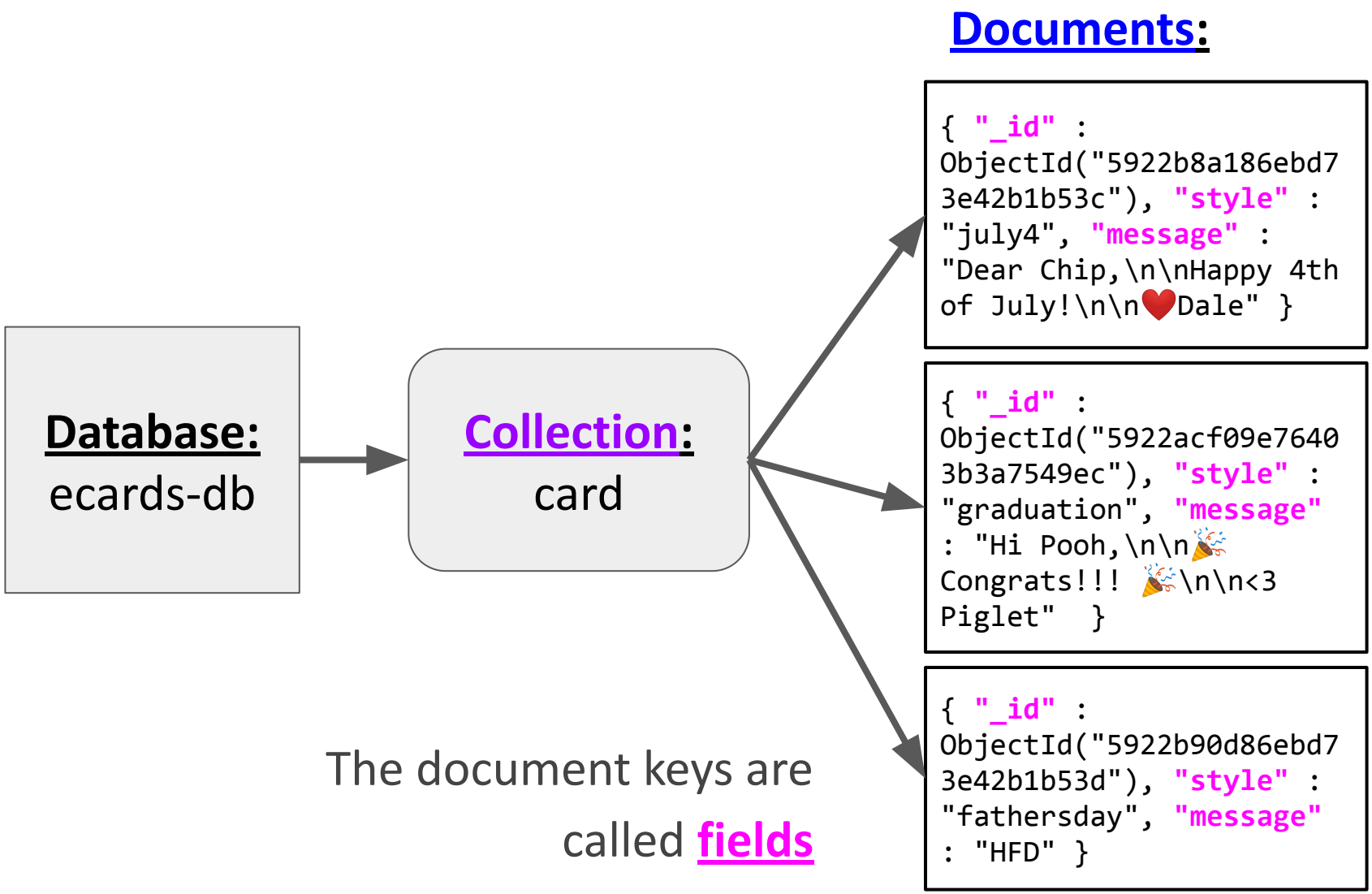
Collection:

- A group of MongoDB **documents**.
- (**Table** in a relational database)

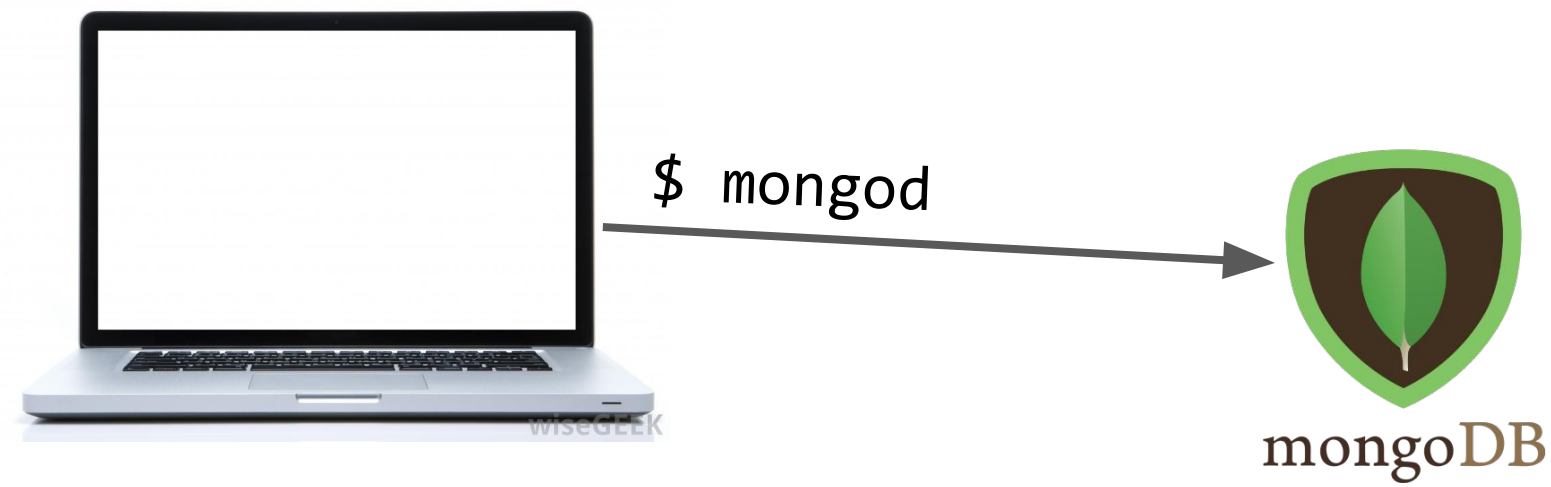
Document:

- A JSON-like object that represents one instance of a collection (**Row** in a relational database)
- Also used more generally to refer to any set of key-value pairs.

MongoDB example



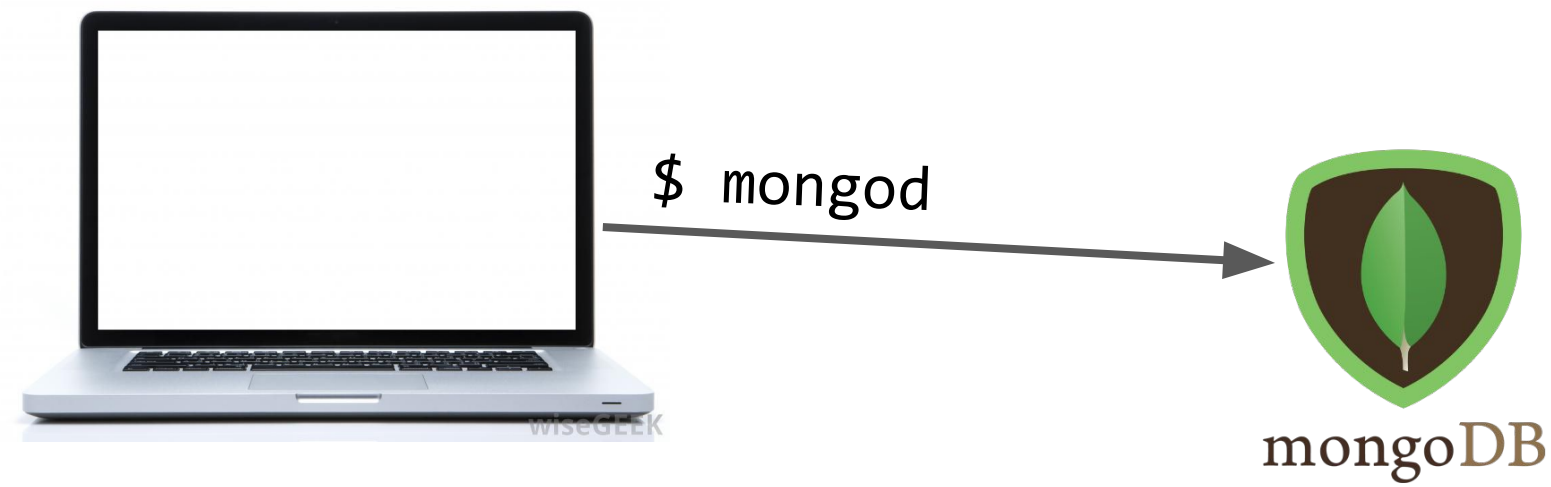
mongod: Database process



When you install MongoDB, it will come with the `mongod` command-line program. This launches the MongoDB database management process and binds it to port 27017:

```
$ mongod
```

mongo: Command-line interface



You can connect to the MongoDB server through the **mongo** shell:

```
$ mongo
```

mongo shell commands

`> show dbs`

- Displays the databases on the MongoDB server

`> use` ***databaseName***

- Switches current database to ***databaseName***
- The ***databaseName*** does not have to exist already
  - It will be created the first time you write data to it

`> show collections`

- Displays the collections for the current database

mongo shell commands

> db.***collection***

- Variable referring to the ***collection*** collection

> db.***collection***.find(***query***)

- Prints the results of ***collection*** matching the query
- The ***query*** is a MongoDB Document (i.e. a JSON object)
  - To get everything in the ***collection*** use `db.collection.find()`
  - To get everything in the collection that matches x=foo, `db.collection.find({x: 'foo'})`

mongo shell commands

> db.***collection***.findOne(***query***)

- Prints the first result of ***collection*** matching the query

> db.***collection***.insertOne(***document***)

- Adds ***document*** to the ***collection***
- ***document*** can have any structure

```
> db.test.insertOne({ name: 'dan' })
> db.test.find()
{ "_id" : ObjectId("5922c0463fa5b27818795950"), "name" : "dan" }
```

MongoDB will automatically add a unique **_id** to every document in a collection.

mongo shell commands

> db.***collection***.deleteOne(***query***)

- Deletes the first result of ***collection*** matching the query

> db.***collection***.deleteMany(***query***)

- Delete multiple documents from ***collection***.
- To delete all documents, db.***collection***.deleteMany()

> db.***collection***.drop()

- Removes the collection from the database

mongo shell

When should you use the `mongo` shell?

- Adding test data
- Deleting test data

MIT License