NYU | Center for Data Science

Week 02.3: Transactions

DS-GA 1004: Big Data

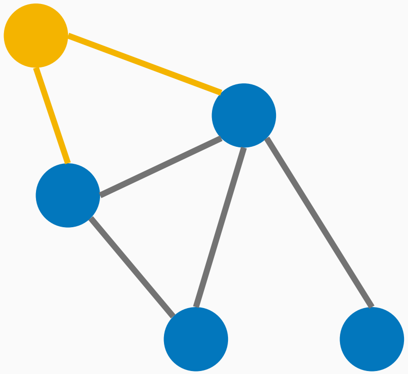
This week

- Relational databases
- SQL
- **Transactions**

More FS drawbacks: consistency!

- What if you want to impose **constraints** on your data?

- Example:
  - Guaranteeing that a graph is **connected**
  - Vertices stored in **nodes.dat**
  - Edges stored in **edges.dat**
  - What if you want to add a **vertex** and two **edges**?

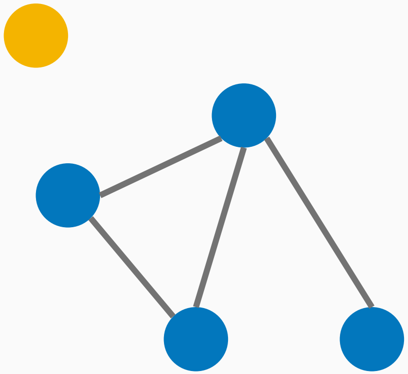


More FS drawbacks: consistency!

- What if you want to impose **constraints** on your data?

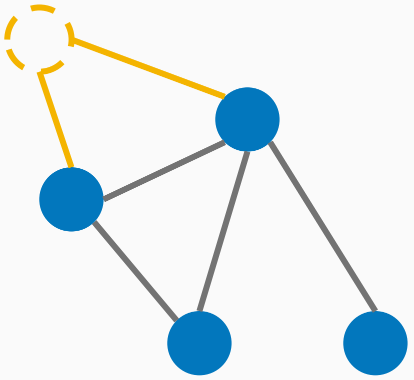
- Example:
    - Guaranteeing that a graph is **connected**
    - Vertices stored in **nodes.dat**
    - Edges stored in **edges.dat**
    - What if you want to add a **vertex** and two **edges**?

- Add vertex first: graph is **disconnected**



More FS drawbacks: consistency!

- What if you want to impose **constraints** on your data?
- Example:
    - Guaranteeing that a graph is **connected**
    - Vertices stored in **nodes.dat**
    - Edges stored in **edges.dat**
    - What if you want to add a **vertex** and two **edges**?
- Add edges first: **edges** are **invalid**



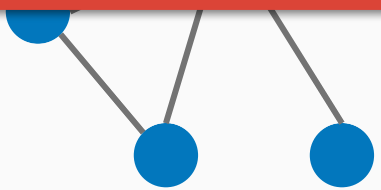
More FS drawbacks: consistency!

- What if you want to impose
- Example:
  - Guaranteeing that a graph is
  - Vertices stored in **nodes.dat**
  - Edges stored in **edges.dat**
  - What if you want to add a **vertex** and two **edges**?
- Add edges first: **edges** are **invalid**

Either operation by itself can render the data **inconsistent.**

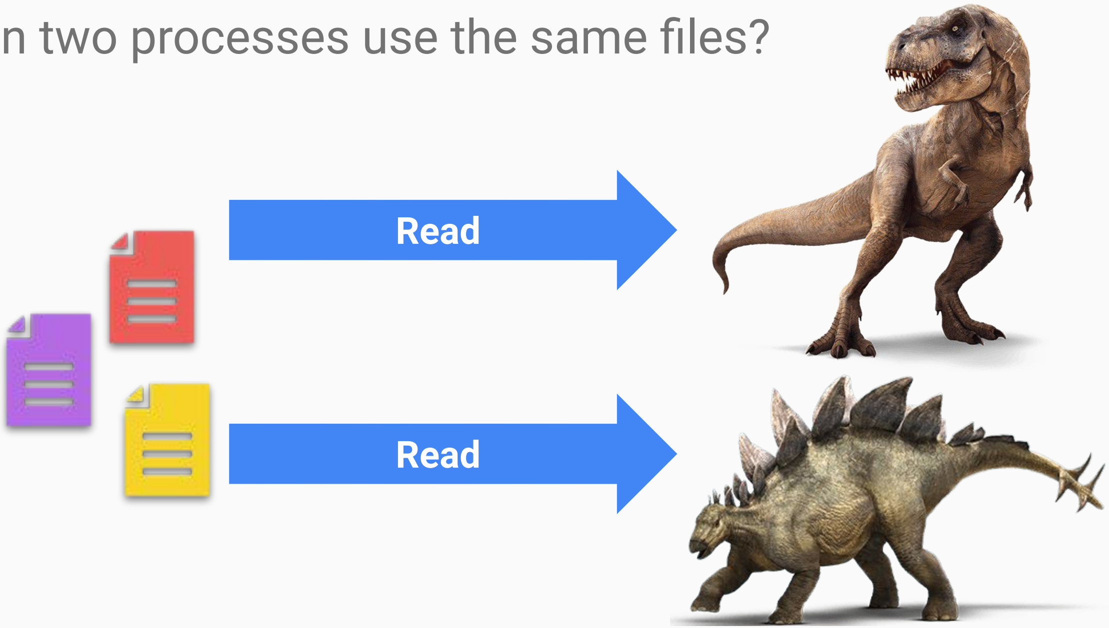
Operations need to be performed simultaneously, but file systems do not generally provide this functionality.

We need another layer of abstraction.



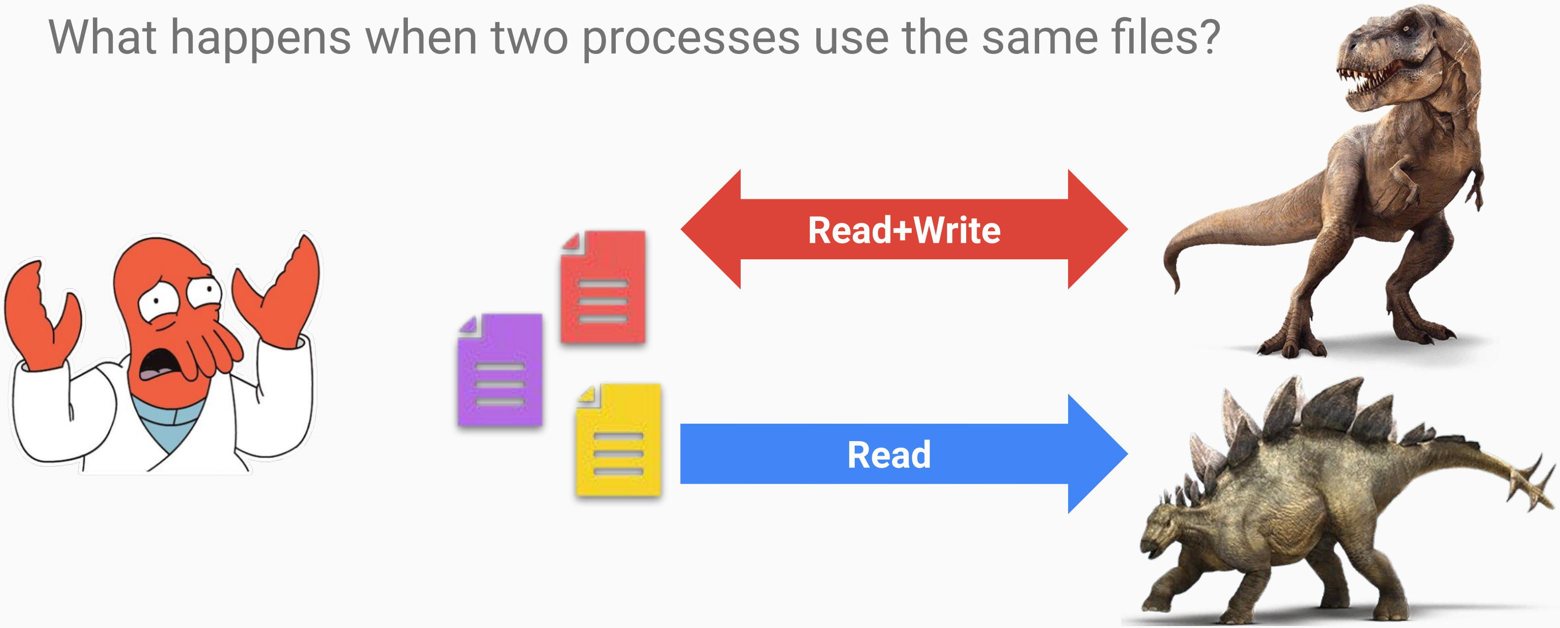
More drawbacks: concurrency!

- What happens when two processes use the same files?



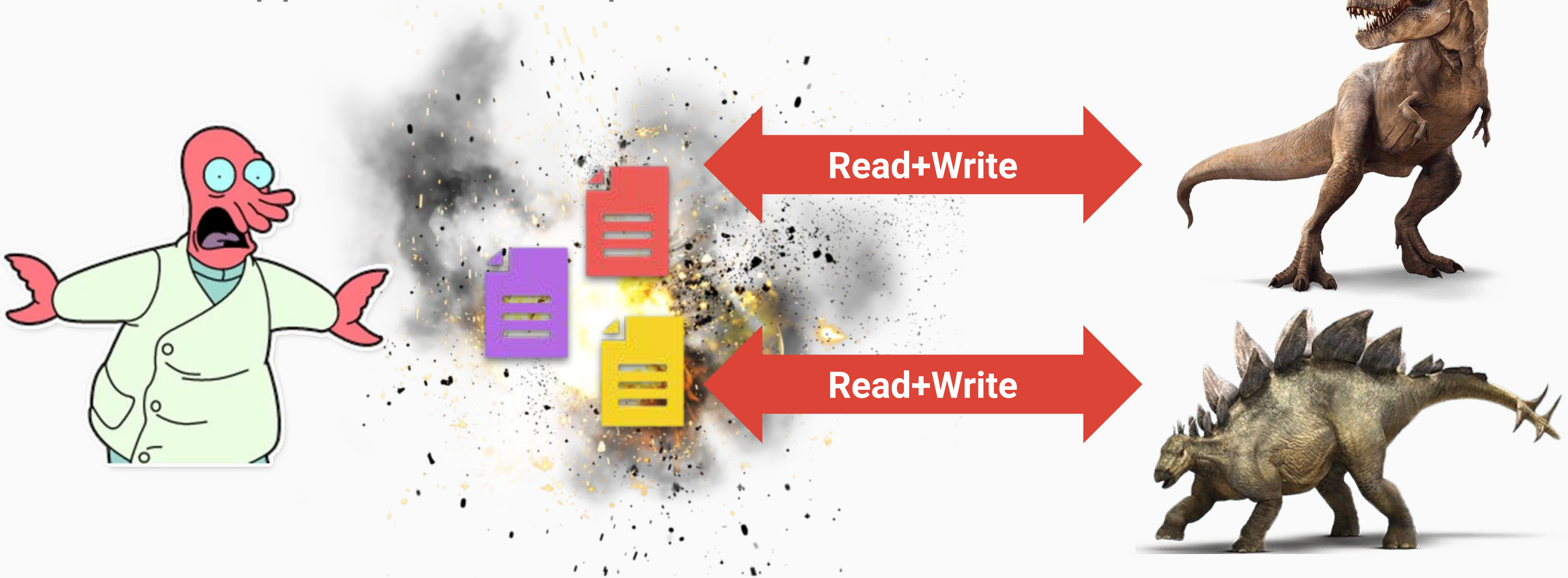
More drawbacks: concurrency!

- What happens when two processes use the same files?



More drawbacks: concurrency!

- What happens when two processes use the same files?



ACID

ACID

Atomicity	Operations are all-or-nothing (No partial updates; operations bundled in **transactions**)
Consistency	Transactions move from one **valid** state to another
Isolation	Concurrent operations do not depend on **order** of execution
Durability	Completed transactions are **permanent** (usually implemented by flushing to disk before completion)

Atomicity in practice

- When modifying tables, wrap query statements in

  BEGIN TRANSACTION; [queries]; **COMMIT**;

  or

  BEGIN TRANSACTION; [queries]; **ROLLBACK**;

- Different DBMS use slightly different syntax (**START**, **BEGIN**)

- If a query **fails** mid-transaction, uncommitted changes will be **abandoned**
  - ⇒ DB is always left in a **valid state**

Consistency in practice

- **Consistency** is maintained by **schema**
  - Schema can add basic value checks as well

```
CREATE TABLE Dinosaur (    id INTEGER PRIMARY KEY,
                           species TEXT NOT NULL,
                           height NUMBER    CHECK (height >= 1.0))
```

- If data does not fit the **schema**, the operation **fails immediately**
  - ⇒ DB cannot enter an **invalid state**
- **Cascading delete** can be used to maintain foreign key constraints

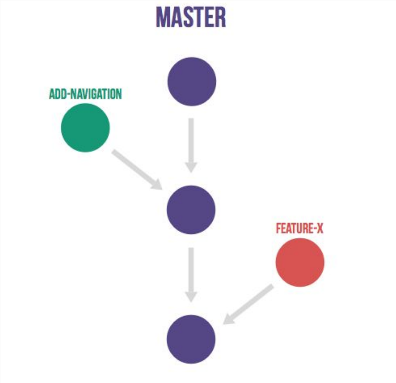
Isolation in practice

- Usually achieved by **locking** the database during modification
  - Only one transaction can hold the lock at a time
- This becomes a real problem for distributed databases!
  - Locking the entire DB would stall everyone
- As we'll see next week, **Map-Reduce** side-steps this problem

You’ve seen some of this already!

- **git** can be seen as a kind of distributed (non-relational) database

- **Atomicity**:
  - Changes are staged independently (**git add**) into a transaction
  - Transactions are finalized atomically (**git commit**)
- **Consistency**:
  - Conflicting changes are detected and forbidden (**merge conflict**)
- **Isolation**:
  - Different **branches** or **clones** can be modified independently
- **Durability**:
  - Punt to the file-system



Summary

Relational database management systems

- Relational model is pretty cool!
  - Tables and joins are a simple, flexible model for many kinds of data!
  - SQL is a little weird, but powerful
- RDBMS provide
  - Data abstraction
  - A common language to interfacing with data (SQL)
  - Concurrent access