Reflective Use of Big Data Tools

Enterprise Architectures for Big Data

Know your Tools

“Having a big toolbox is impressive. But knowing **when to use what tool** and why separates the true expert from the show-off.” *Gregor Hohpe*

Types of Systems

Three Types of Systems

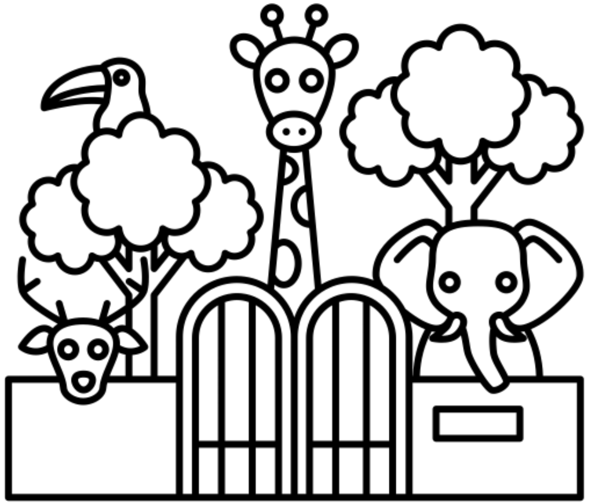
1. **Services**
   (online systems)

2. **Batch processing systems**
   (offline systems)

3. **Stream processing systems**
   (near-real-time systems)

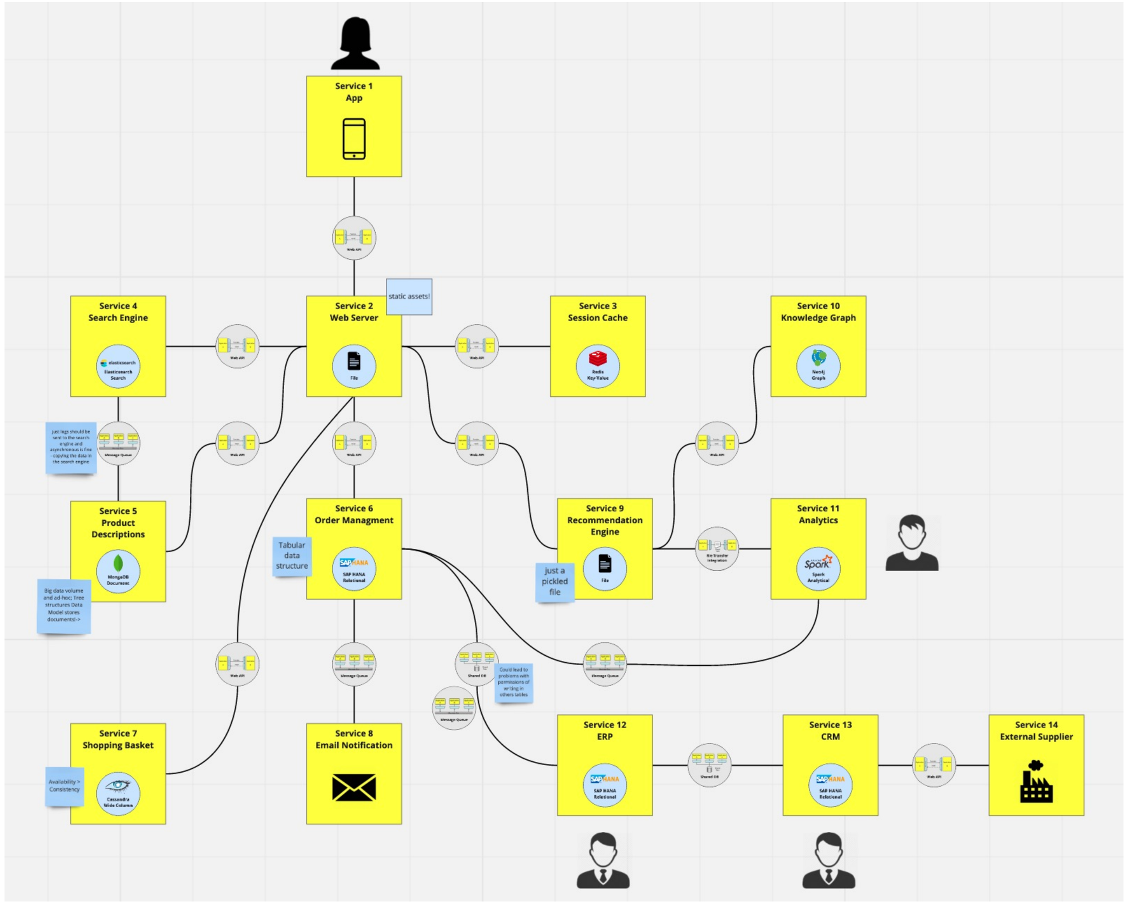
Three Types of Systems

1. **Services** (online systems)
   - A service waits for a request or instruction from a client to arrive.
   - When one is received, the service tries to handle it as quickly as possible and sends a response back.
   - **Response time** is usually the primary measure of performance of a service, and **availability** is often very important.
2. **Batch processing systems** (offline systems)
   - A batch processing system takes a **large amount of input data**, runs a job to process it, and produces some output data.
   - Jobs often take a while (from a few minutes to several days), so **there normally isn't a user waiting** for the job to finish.
   - Instead, batch jobs are often scheduled to **run periodically** (for example, once a day).
   - The primary performance measure of a batch job is usually **throughput.**
3. **Stream processing systems** (near-real-time systems)
   - Stream processing is somewhere between online and offline/batch processing (so it is sometimes called near-real-time or nearline processing).
   - Like a batch processing system, a stream processor consumes inputs and produces outputs (rather than responding to requests).
   - However, a **stream job operates on events shortly after they happen**, whereas a batch job operates on a fixed set of input data.
   - This difference allows stream processing systems to have **lower latency** than the equivalent batch systems.

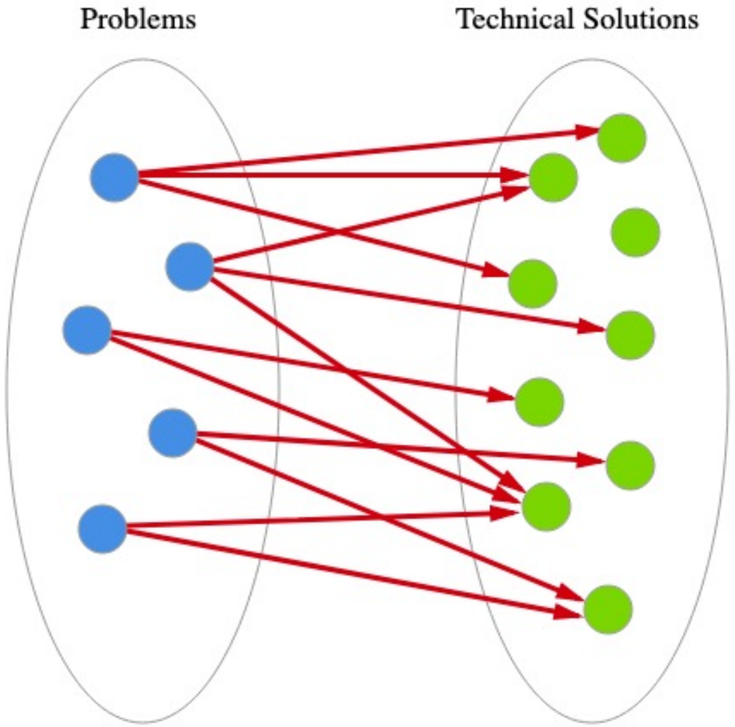


Solution creep

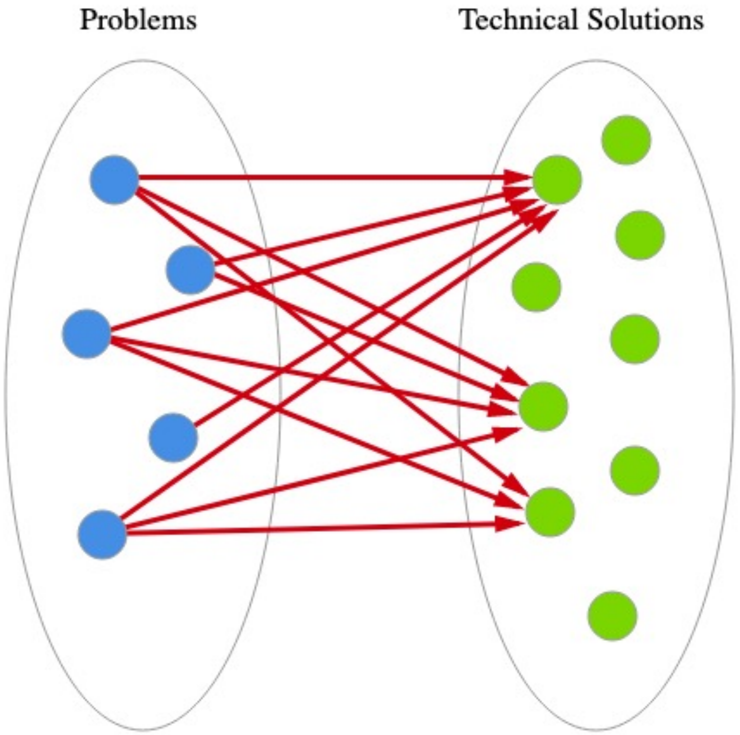
Zilondu Architectural Kata



The way you might choose technology in a world where choices are cheap: "pick the right tool for the job."



The way you choose technology in the world where operations and maintainability are a serious concerns (i.e., "reality")

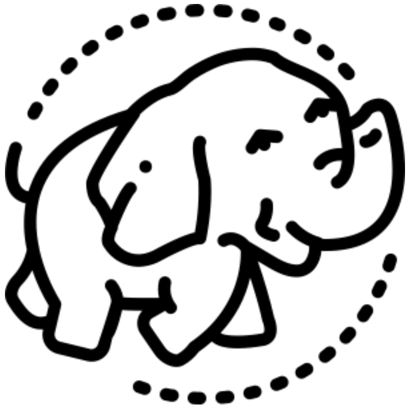


Use Innovative Solutions Strategically

- Metaphor: You have only three innovation tokens
- Where do you want to differentiate?
- For the rest: choose well-understood boring technologies

Unknown Unknowns

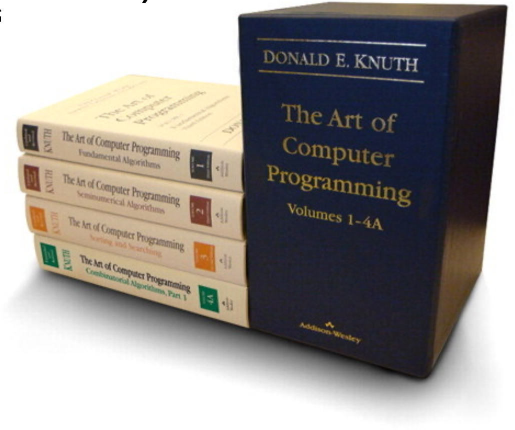
- Known Knowns
  - Things you know
  - We know that the average response time would decrease by 80%
- Known unkowns
  - Things you know that you don't know
  - We don't know what happens when this database hits 100% CPU.
- Unknown unknowns
  - Things you don't know that you don't know
  - Ups, it did not even occur to us that accessing the statics of the Java Virtual Machine would cause the Garbage Collection to pause.
  - You are always hit by the bus you did not see.



Know when to use Hadoop

Premature Optimization is the Root of all Evil

“The real problem is that programmers have spent far too much time worrying about efficiency in the wrong places and at the wrong times; **premature optimization is the root of all evil** (or at least most of it) in programming.“
Donald Knuth



Limitations of Hadoop

- Batch only processing
- Most Hadoop analyses are using a small subset of the data
- Even at Facebook and Microsoft most of the analysis process use less than 100 GB
- RAM gets cheaper

Source: Rowstron, Antony, et al. (2012)

Hadoop / Big Data: Use the right tool!

- Do you really need Hadoop?
- Or is a Command Line or a Python Script
  or a PostgreSQL Database enough?

- But my data is hundreds of megabytes! Excel won't load it.
- But my data is 10 gigabytes!
- But my data is 100GB/500GB/1TB!
- But my data is more than 10TB!

Source: https://www.chrisstucchio.com/blog/2013/hadoop_hatred.html

NoSQL Features in PostgreSQL

NoSQL Features in PostgreSQL

- Mix of Relational and NoSQL
  - JSON Columns
    - https://www.postgresql.org/docs/current/static/functions-json.html
  - Key/Value Columns (Hstore)
    - https://www.postgresql.org/docs/current/static/hstore.html
  - Full Text Search
    - https://www.postgresql.org/docs/current/textsearch.html
  - Geographical Data
    - PostGIS: https://postgis.net/
- Scaling
  - Partitioning and Replication are possible
  - Not yet so good scale out mechanisms like MongoDB
  - Different priority than NoSQL ➔ Consistency!
- Reliability / High Available:
  - https://www.postgresql.org/docs/current/high-availability.html

PostgreSQL JSON

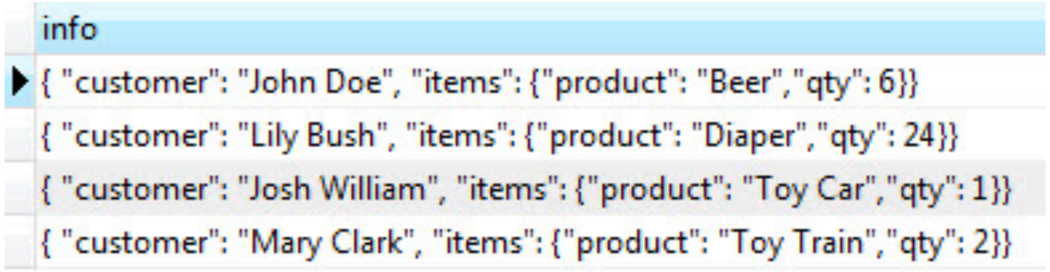
```
CREATE TABLE orders (
     ID serial NOT NULL PRIMARY KEY,
     info json NOT NULL
);

INSERT INTO orders (info)
VALUES
     (
          '{ "customer": "John Doe",
          "items": {"product": "Beer","qty": 6}}'
     );
```

PostgreSQL JSON

- **Querying JSON data**

- `SELECT info FROM orders;`



info
{ "customer": "John Doe", "items": {"product": "Beer","qty": 6}}
{ "customer": "Lily Bush", "items": {"product": "Diaper","qty": 24}}
{ "customer": "Josh William", "items": {"product": "Toy Car","qty": 1}}
{ "customer": "Mary Clark", "items": {"product": "Toy Train","qty": 2}}

PostgreSQL JSON

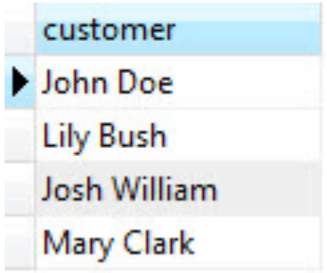
- **Querying JSON data**
- The operator ->>
  - Returns Text
- The operator ->
  - Returns JSON object

Source: https://www.postgresqltutorial.com/postgresql-json/

PostgreSQL JSON

- The operator ->>
  - Returns Text

```
SELECT info ->> 'customer' AS customer
FROM orders;
```

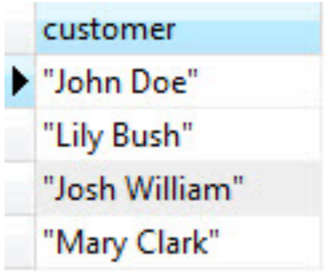


```
customer
John Doe
Lily Bush
Josh William
Mary Clark
```

PostgreSQL JSON

- The operator ->
  - Returns JSON

```
SELECT info -> 'customer' AS customer
FROM orders;
```

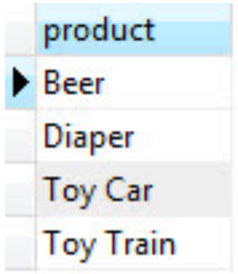


```
customer
"John Doe"
"Lily Bush"
"Josh William"
"Mary Clark"
```

PostgreSQL JSON

- The operator ->
  - Returns JSON

```
SELECT info -> 'items' ->> 'product' as product
FROM orders ORDER BY product;
```

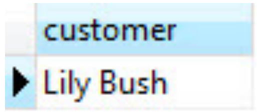


product
Beer
Diaper
Toy Car
Toy Train

PostgreSQL JSON

- Where clause with JSON

```
SELECT info ->> 'customer' AS customer
FROM orders
WHERE info -> 'items' ->> 'product' = 'Diaper'
```

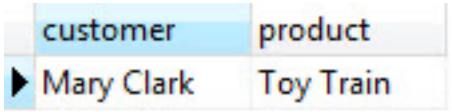


customer
Lily Bush

PostgreSQL JSON

- Where clause with JSON and type casting from string to integer

```
SELECT info ->> 'customer' AS customer,
info -> 'items' ->> 'product' AS product
FROM orders
WHERE CAST ( info -> 'items' ->> 'qty' AS INTEGER ) = 2
```



customer	product
Mary Clark	Toy Train

Source: https://www.postgresqltutorial.com/postgresql-json/

PostgreSQL HSTORE

```
CREATE TABLE books (
      id serial primary key,
      title VARCHAR (255),
      attr hstore
);
```

PostgreSQL HSTORE

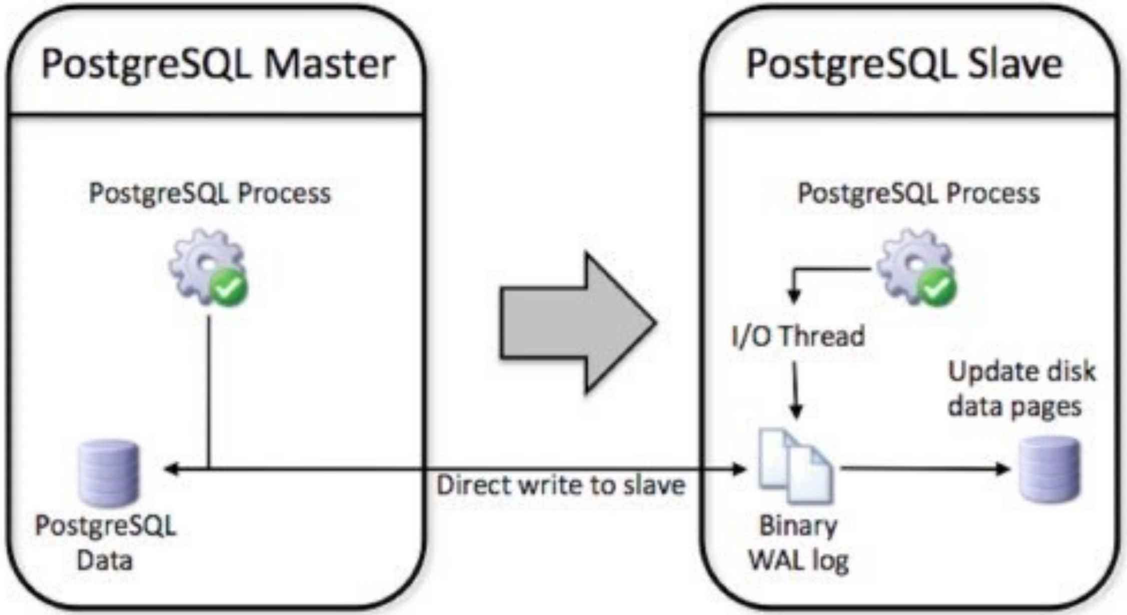
```
INSERT INTO books (title, attr)
VALUES
 (
    'PostgreSQL Cheat Sheet',
    '
        "paperback" => "5",
        "publisher" => "postgresqltutorial.com",
        "language"  => "English",
        "ISBN-13"   => "978-1449370001",
        "weight"    => "1 ounces“
    '
);
```

PostgreSQL HSTORE

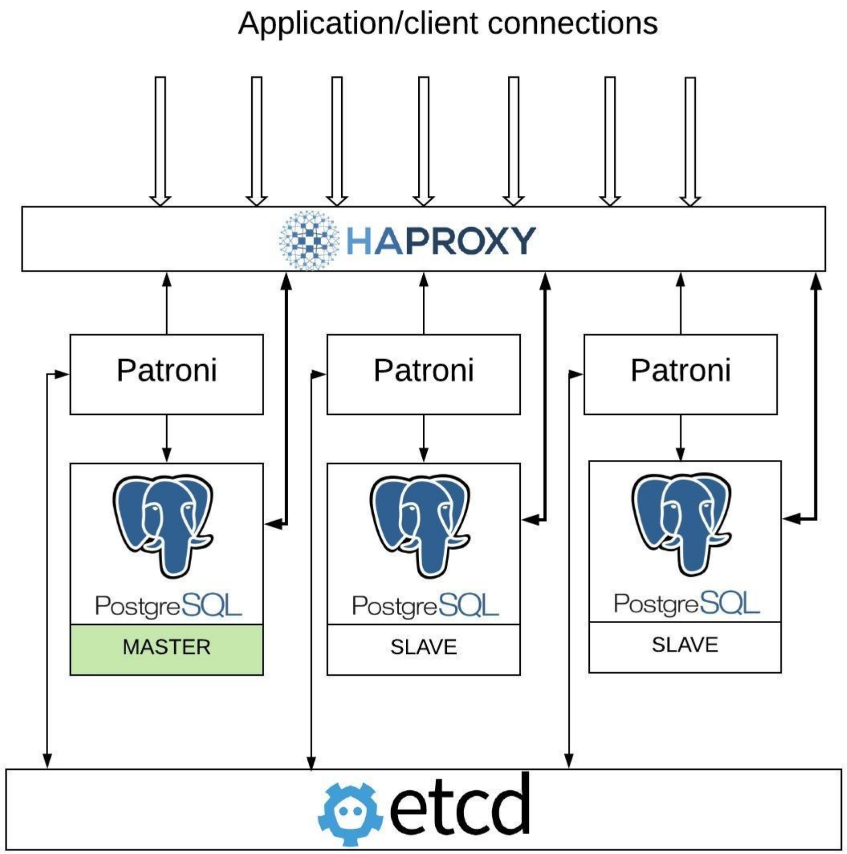
```
SELECT
     attr -> 'weight' AS weight
FROM
     books
WHERE
     attr -> 'ISBN-13' = '978-1449370000'
```

But what about high reliability?

- Streaming replication of the Write Ahead Log (WAL) to a standby following node



Automatic failover



Source: https://www.percona.com/blog/2018/09/28/high-availability-for-enterprise-grade-postgresql-environments/