Forget databases - use files!

Wolfgang Gassler
ViennaDB June 2025, Vienna

Thx to Brian Ecker for his contributions

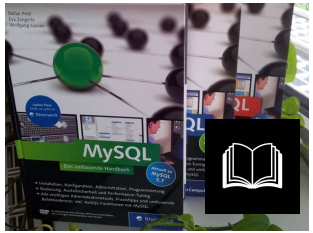
Im nächsen Projekt verwende ich das DBMS ...

About me

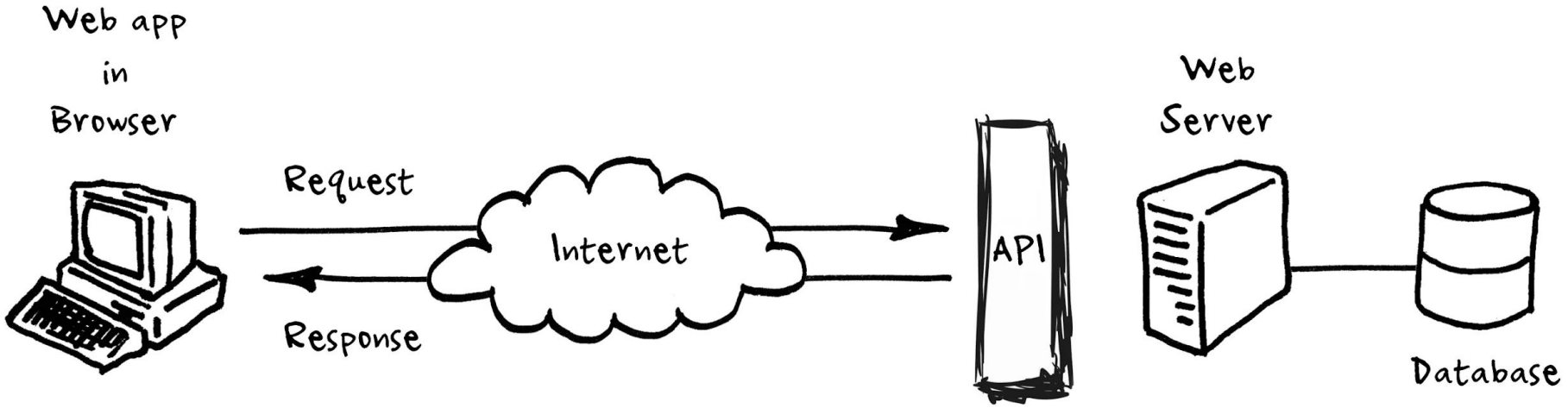
Wolfgang Gassler, born in Innsbruck, Austria

Computer Science (Databases), Ph.D

- 25 years in industry (Perl, PHP3)
- 10 years Academic Researcher, Lecturer, PM
- 2016 Engineering/DataScience Manager trivago
- Since 2020 Tech Consultant inlupus




ARCHITECTURE
Web Server
App Server
User
Auth Service
Cache
App Server
Database


Web app
in
Browser
Request
Response
Internet
API
Web
Server
Database

```php
$result = $mysqli->query(sprintf('SELECT * FROM data.data WHERE id IN (%s)',
implode(",",$ids)));

$data = array();
while ($row = $result->fetch_assoc()) {
   $features = array();

   if ($row['wifi']) { $features[] = "wifi"; }
   if ($row['24reception']) { $features[] = "24reception"; }
   if ($row['pool']) { $features[] = "pool"; }

   $data[] = array('id'=>$row['id'], 'name'=>$row['name'], 'features' => $features);
}
echo var_dump($data);
echo "done";
```

Simple example fetching 10 hotels from MySQL

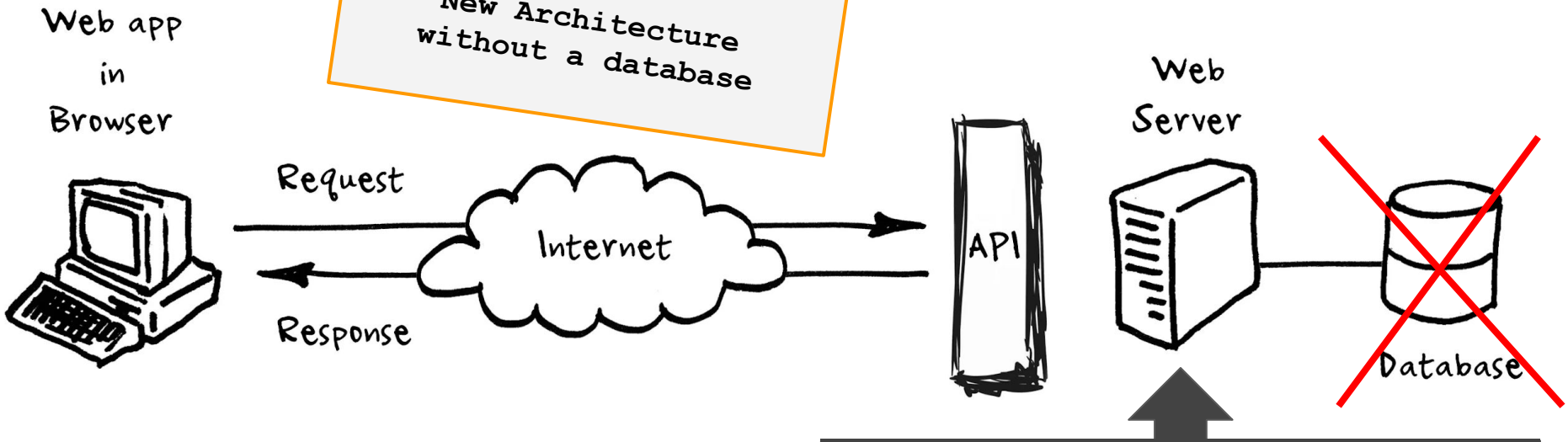
```
WARMING DB CACHE:
Requests      [total, rate]            3000, 300.10
Duration      [total, attack, wait]    10.024828s, 9.996664667s, 28.163333ms
Latencies     [mean, 50, 95, 99, max]  77.600269ms, 41.286497ms, 151.693676ms, 976.306614ms, 2.68813865s
Bytes In      [total, mean]            7699671, 2566.56
Bytes Out     [total, mean]            0, 0.00
Success       [ratio]                  100.00%
Status Codes  [code:count]             200:3000
Error Set:
```

Attack Statistics 2020
Database based version

```
DB BASED ATTACK:
Requests      [total, rate]            3000, 300.10
Duration      [total, attack, wait]    10.039157s, 9.996664667s, 42.492333ms
Latencies     [mean, 50, 95, 99, max]  46.124773ms, 34.386086ms, 108.493327ms, 204.989259ms, 382.092842ms
Bytes In      [total, mean]            7699814, 2566.60
Bytes Out     [total, mean]            0, 0.00
Success       [ratio]                  100.00%
Status Codes  [code:count]             200:3000
Error Set:
```


beautiful
CRAZY

New Architecture
without a database




```
bash# ls -l /tmp/data/ | head -n 20
total 40000
-rw-r--r-- 1 www-data www-data 121 Mar 17 12:34 1
-rw-r--r-- 1 www-data www-data 123 Mar 17 12:34 10
-rw-r--r-- 1 www-data www-data 125 Mar 17 12:34 100
-rw-r--r-- 1 www-data www-data 127 Mar 17 12:34 1000
-rw-r--r-- 1 www-data www-data 129 Mar 17 12:34 10000
-rw-r--r-- 1 www-data www-data 127 Mar 17 12:34 1001
-rw-r--r-- 1 www-data www-data 127 Mar 17 12:34 1002
-rw-r--r-- 1 www-data www-data 127 Mar 17 12:34 1003
-rw-r--r-- 1 www-data www-data 127 Mar 17 12:34 1004
-rw-r--r-- 1 www-data www-data 127 Mar 17 12:34 1005
-rw-r--r-- 1 www-data www-data 127 Mar 17 12:34 1006
-rw-r--r-- 1 www-data www-data 127 Mar 17 12:34 1007
-rw-r--r-- 1 www-data www-data 127 Mar 17 12:34 1008
```

```php
$data = array();
foreach($ids as $id) {
    $data[] = include(CONFIG_DATA_PATH."/".$id);
}

echo var_dump($data);
echo "done";
```

Simple example fetching 10 hotels from files

```php
<?php return array(
    'id'=>5,
    'name'=>'hotel 5',
    'features'=>array('pool','wifi','24hreception'));
```

pregenerated PHP file

```
WARMING FILE CACHE:
Requests      [total, rate]            3000, 300.10
Duration      [total, attack, wait]    10.010827s, 9.996664667s, 14.162333ms
Latencies     [mean, 50, 95, 99, max]  22.122388ms, 20.672375ms, 34.719268ms, 45.305007ms, 70.978285ms
Bytes In      [total, mean]            7523591, 2507.86
Bytes Out     [total, mean]            0, 0.00
Success       [ratio]                  100.00%
Status Codes  [code:count]             200:3000
Error Set:
```

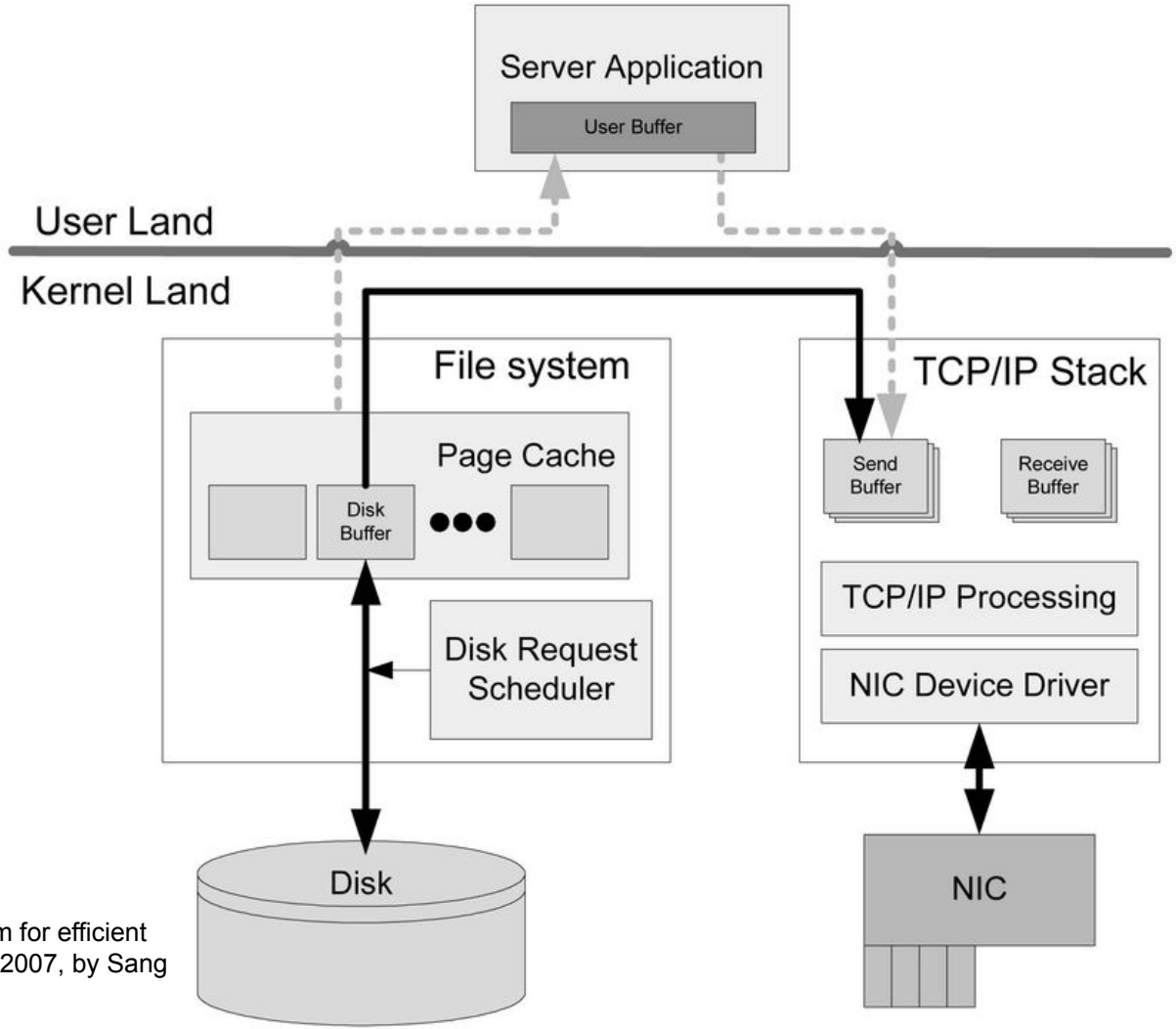
Attack Statistics 2020
File based version

```
FILE BASED ATTACK:
Requests      [total, rate]            3000, 300.10
Duration      [total, attack, wait]    10.007853s, 9.996664667s, 11.188333ms
Latencies     [mean, 50, 95, 99, max]  19.327833ms, 19.155892ms, 27.93551ms, 35.90069ms, 55.080323ms
Bytes In      [total, mean]            7523335, 2507.78
Bytes Out     [total, mean]            0, 0.00
Success       [ratio]                  100.00%
Status Codes  [code:count]             200:3000
Error Set:
```

```
DB BASED ATTACK:
Latencies     [mean, 50, 95, 99, max]  46.124773ms, 34.386086ms, 108.493327ms, 204.989259ms, 382.092842ms
```

Where do you lose time?

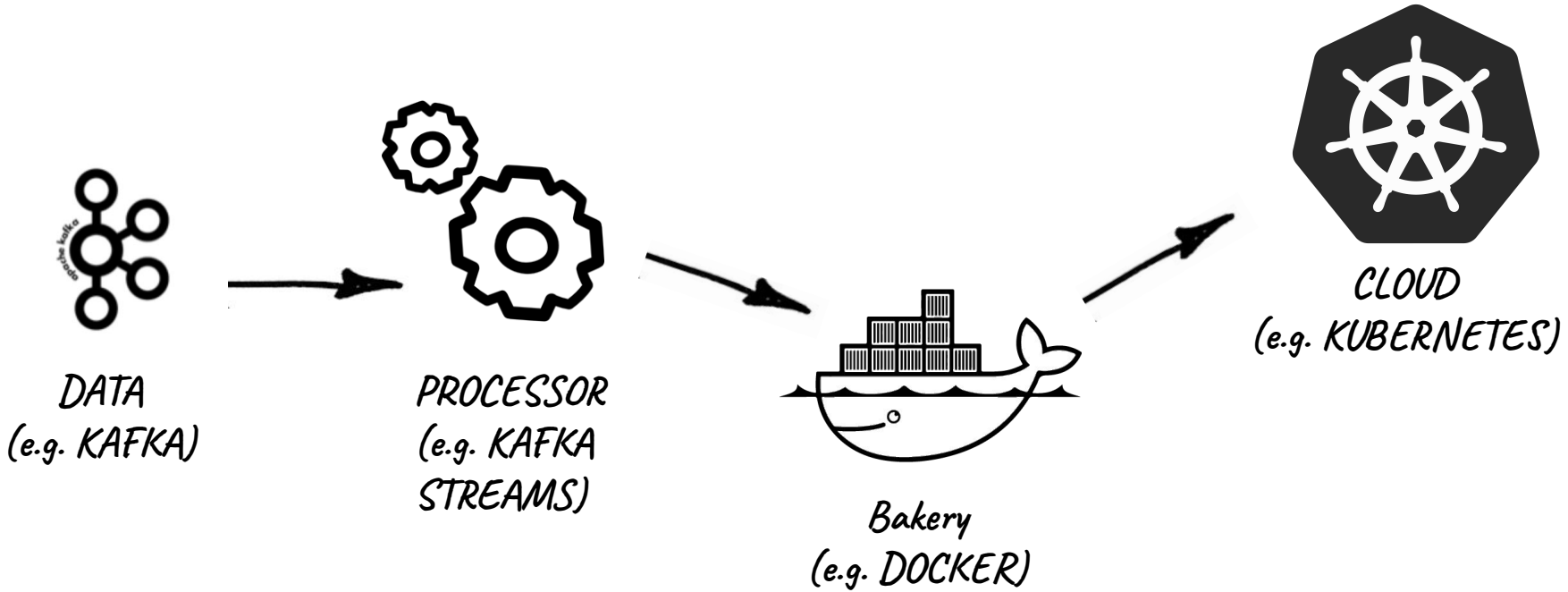
- Connection handshake (incl. authentication, permissions, etc.)
- Connection cleanup
- Network stack overhead (TCP/IP, physical connection between servers)
- Query parsing, execution plan generation, security, ACID (TAs, isolation, etc.)
- Query execution (especially complexer queries such as joins)
- Data mapping in the application (e.g. ORM)



TPF: TCP plugged file system for efficient data delivery over TCP, May 2007, by Sang Seok LimKyu Ho Park

Static Site Generators

Architecture: Reverse Caching / Shift Left



Advantages: Performance

- Fast due to OS optimizations
- Reads scale with number of application nodes (fan out)
- Move joins + fetching to stream processor layer (do work only once, not with every single request)

The difference between expert and advanced/intermediate technical staff is that the advanced engineer has an understanding of complex solution and mistakenly tries to apply them everywhere, so the net effect is to increase complexity. The expert typically sees simple solutions and method of resolving complexity and has a net effect of reducing overall system complexity.

Source: https://news.ycombinator.com/item?id=21377349

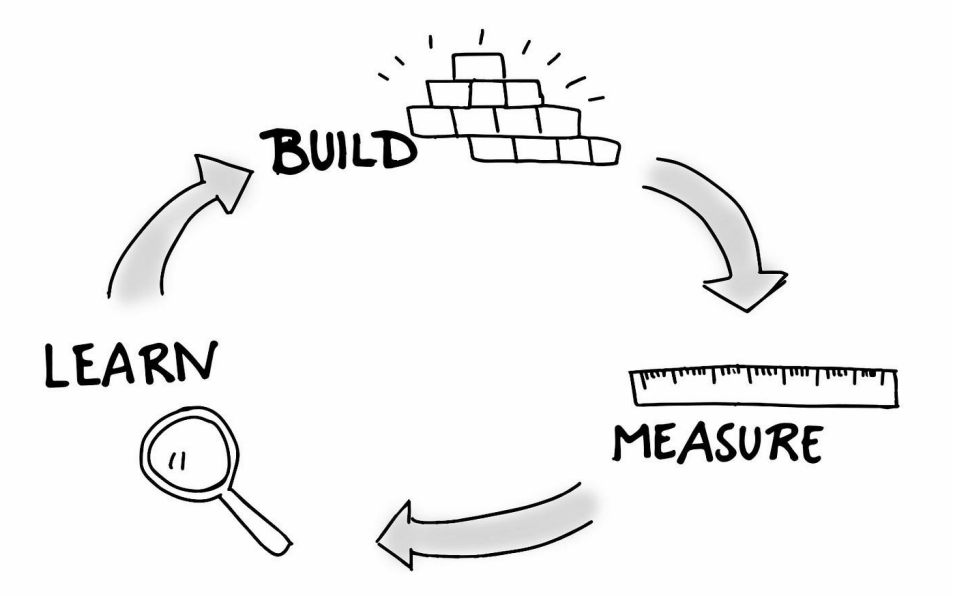
Keep it stupid simple

One system vs. application and database

- Simpler monitoring and maintenance
- No more db versioning, migrations
- Idempotence, Reproducibility
- Simple roll backs (if files baked into docker image)

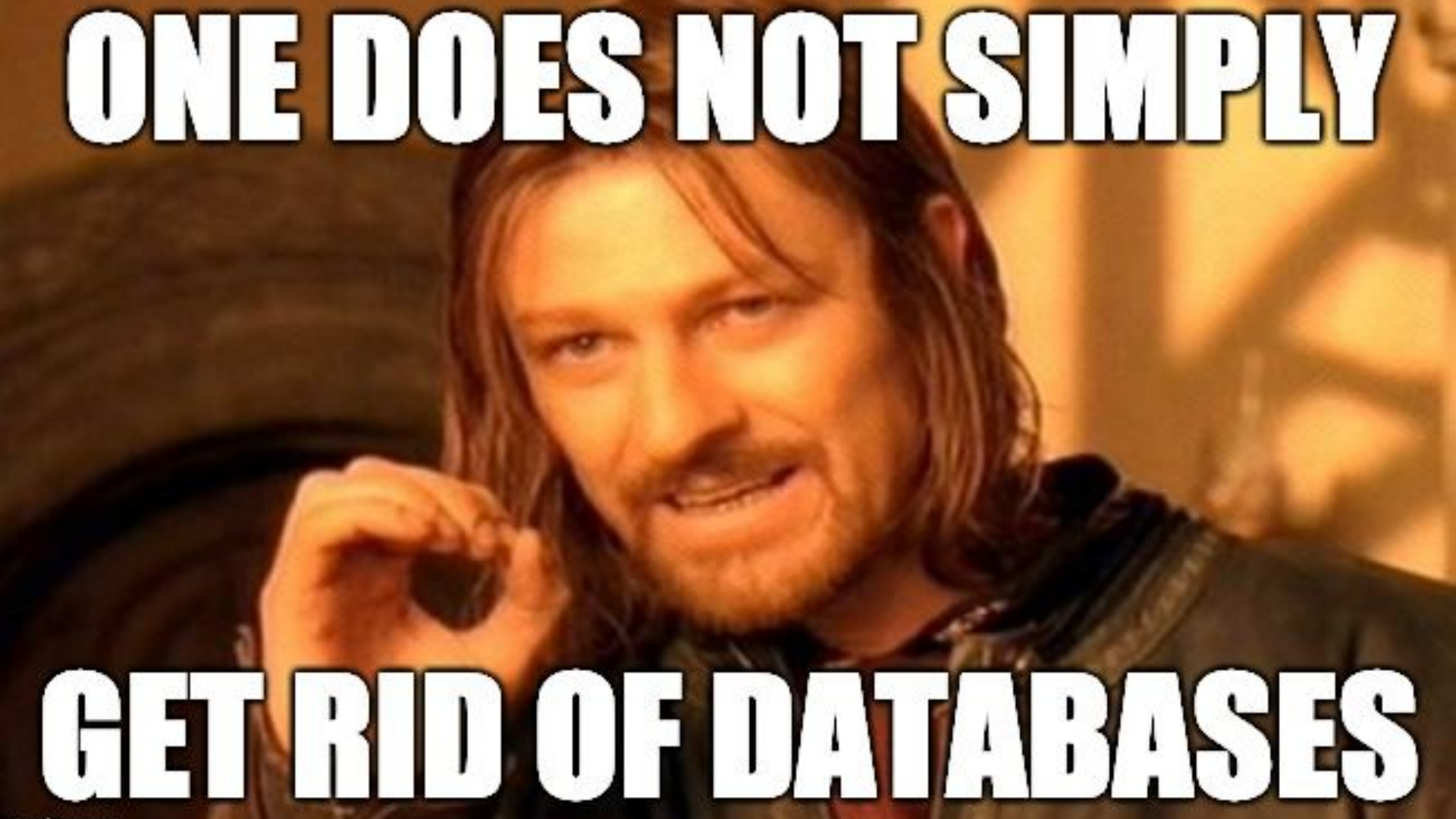
KISS in Side Projects: Example WhatsApp Bot

- Simplest storage that meets today's needs
- Write one big JSON file
- Backup JSON with every write
- Implement event sourcing approach
- Add subscription/payment



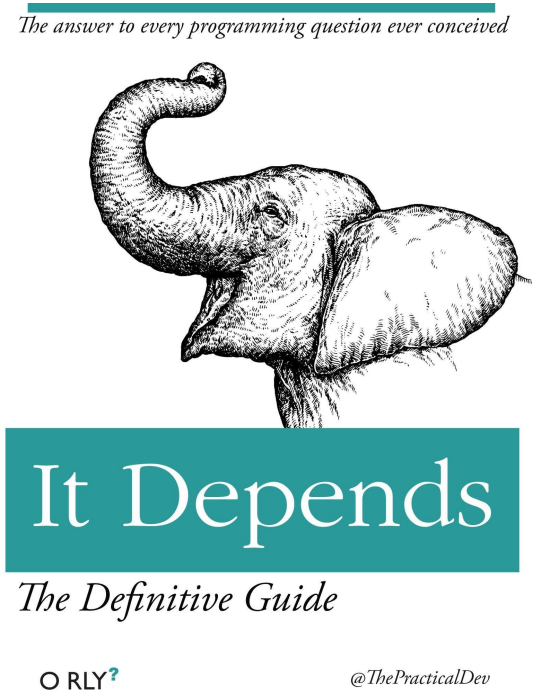
Summary

- Files are not slow, Databases are not always the solution
- File systems are heavily optimized and provide built in (OS) caching layers
- Move heavy work (data fetching, joins, mapping) to asynchronous processes (e.g. stream based file generation). Serialize database query results to disk (reverse cache, shift left)
- #Simplicity & #Scalability (fan out)


ONE DOES NOT SIMPLY
GET RID OF DATABASES

Disadvantages/But....

- ...you can tweak the database.
  - Have you ever had to tweak the filesystem?
- ...then I have to deploy for every data change
  - If deployment is painful for you, you should work on your deployment pipeline
  - When data changes frequently, a stream consumer on the node might help but could be more difficult to scale and maintain
- ...real-time writes (ACID) are not possible
- ...does not scale in case of “big data”
- ...does not work when data has to be close to real time
- ...possible overhead due to implementation of your own replication system



PHP Version:

DB WARMED - Final test will start
Latencies [min, mean, 50, 90, 95, 99, max] 755.966µs, 1.541ms, 1.529ms, 2.004ms, 2.139ms, 2.399ms, 4.693ms

FILES WARMED - Final test will start
Latencies [min, mean, 50, 90, 95, 99, max] 507.911µs, 1.185ms, 1.193ms, 1.638ms, 1.76ms, 1.99ms, 2.323ms

SQLITE WARMED - Final test will start
Latencies [min, mean, 50, 90, 95, 99, max] 551.974µs, 1.281ms, 1.292ms, 1.759ms, 1.884ms, 2.154ms, 2.786ms

Attack Statistics 2025

Node.js Version:

DB WARMED - Final test will start
Latencies [min, mean, 50, 90, 95, 99, max] 952.225µs, 2.075ms, 1.485ms, 2.191ms, 5.129ms, 20.655ms, 33.994ms

MEMORY WARMED - Final test will start
Latencies [min, mean, 50, 90, 95, 99, max] 227.506µs, 551.641µs, 530.019µs, 750.61µs, 835.942µs, 972.757µs, 2.105ms

SQLITE WARMED - Final test will start
Latencies [min, mean, 50, 90, 95, 99, max] 539.922µs, 1.034ms, 993.95µs, 1.264ms, 1.389ms, 1.67ms, 3.552ms

Source: https://github.com/woolfg/no-dbms-use-files

Summary

- Files are not slow, Databases are not always the solution
- File systems are heavily optimized and provide built in (OS) caching layers
- Move heavy work (data fetching, joins, mapping) to asynchronous processes (e.g. stream based file generation)
- Serialize database query results to disk

- For some use cases you might still need a database ;)

https://engineeringkiosk.dev/ep129
https://engineeringkiosk.dev/ep99