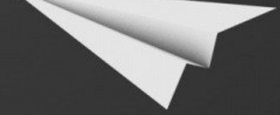
My code is slow, and why you should care

Musings of a software developer



Jason Chalom
contact@jasonchalom.com

Speed

Stability

Reliability

WHY?

- Trust in the software
- Growth

Focus:

What is your reason?

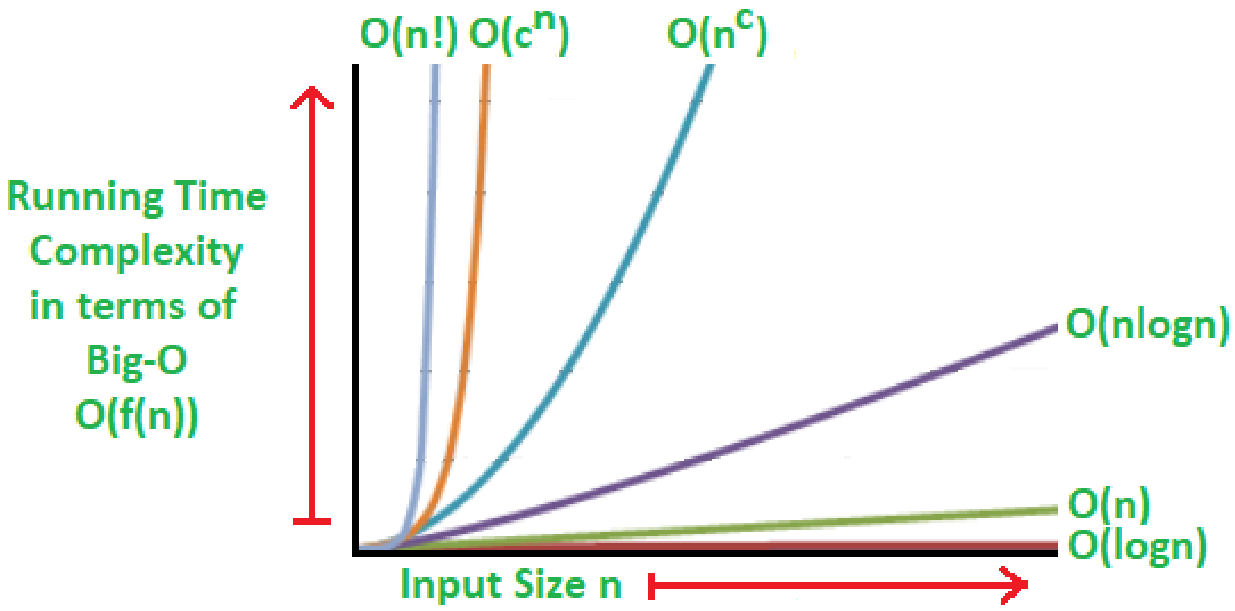
Find the objective and scope it

Measure and measure some more

Types of Performance

- Computational
  - Big O, Omega, Phi
- Data loading speeds
- Architecture and system modelling
- Dependencies and third party reliance
- Precision

0. Computational



O(n!),O(c^n),O(n^c) - Worst

O(nlogn) - Bad

O(n) - Fair

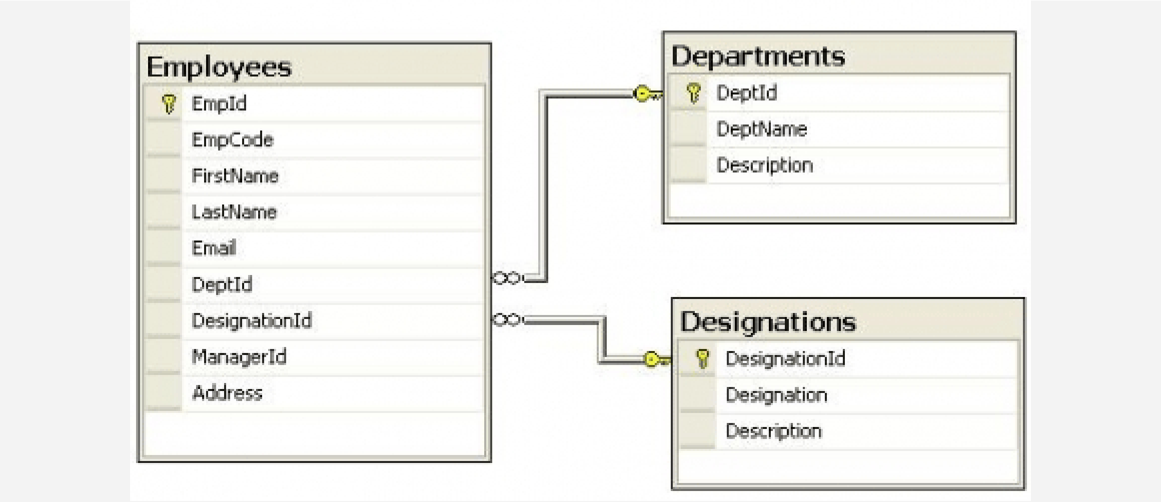
O(logn) - Good

O(1) - Best

1. Data

The usual suspect

Understand your data architecture

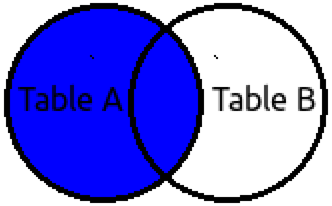


Employees
- EmpId
- EmpCode
- FirstName
- LastName
- Email
- DeptId
- DesignationId
- ManagerId
- Address

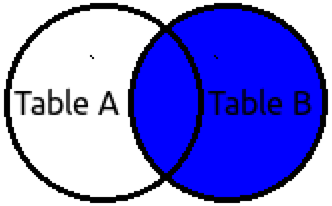
Departments
- DeptId
- DeptName
- Description

Designations
- DesignationId
- Designation
- Description

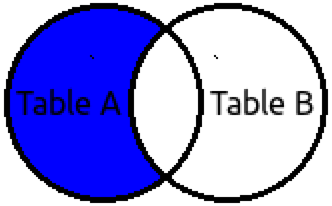
Relational Database Diagram For Sample Database



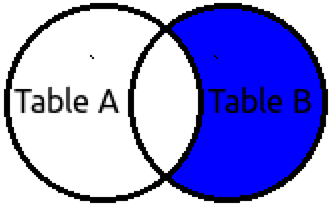
```
SELECT [list] FROM
    [Table A] A
LEFT JOIN
    [Table B] B
ON A.Value = B.Value
```



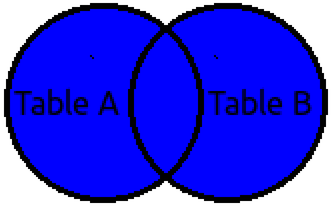
```
SELECT [list] FROM
    [Table A] A
RIGHT JOIN
    [Table B] B
ON A.Value = B.Value
```



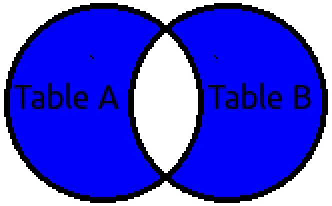
```
SELECT [list] FROM
    [Table A] A
LEFT JOIN
    [Table B] B
ON A.Value = B.Value
WHERE B.Value IS NULL
```



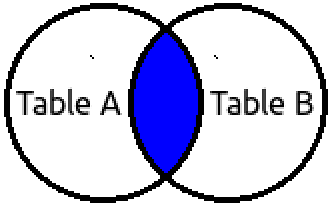
```
SELECT [list] FROM
    [Table A] A
RIGHT JOIN
    [Table B] B
ON A.Value = B.Value
WHERE A.Value IS NULL
```



```
SELECT [list] FROM
    [Table A] A
FULL OUTER JOIN
    [Table B] B
ON A.Value = B.Value
```



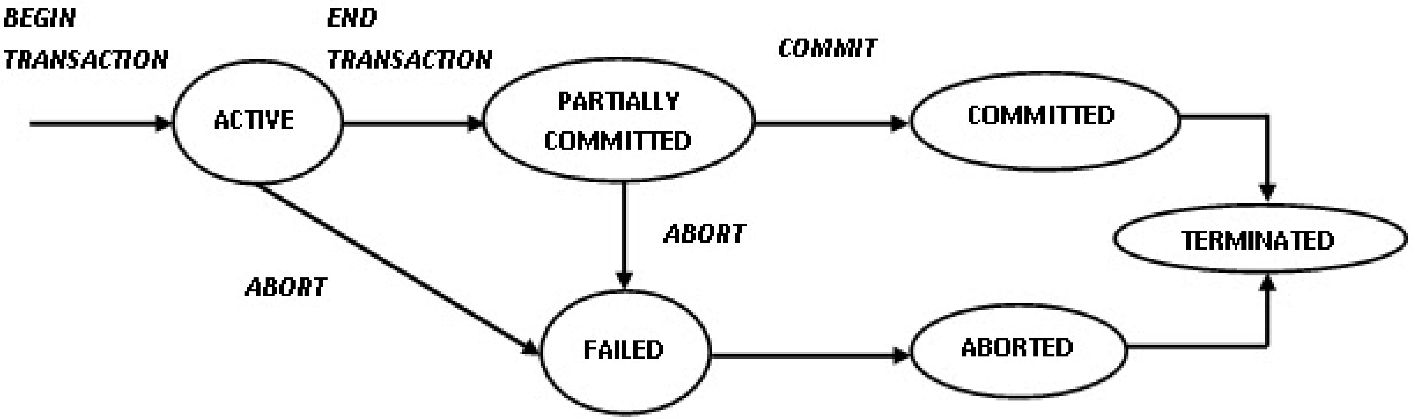
```
SELECT [list] FROM
    [Table A] A
FULL OUTER JOIN
    [Table B] B
ON A.Value = B.Value
WHERE A.Value IS NULL
OR B.Value IS NULL
```

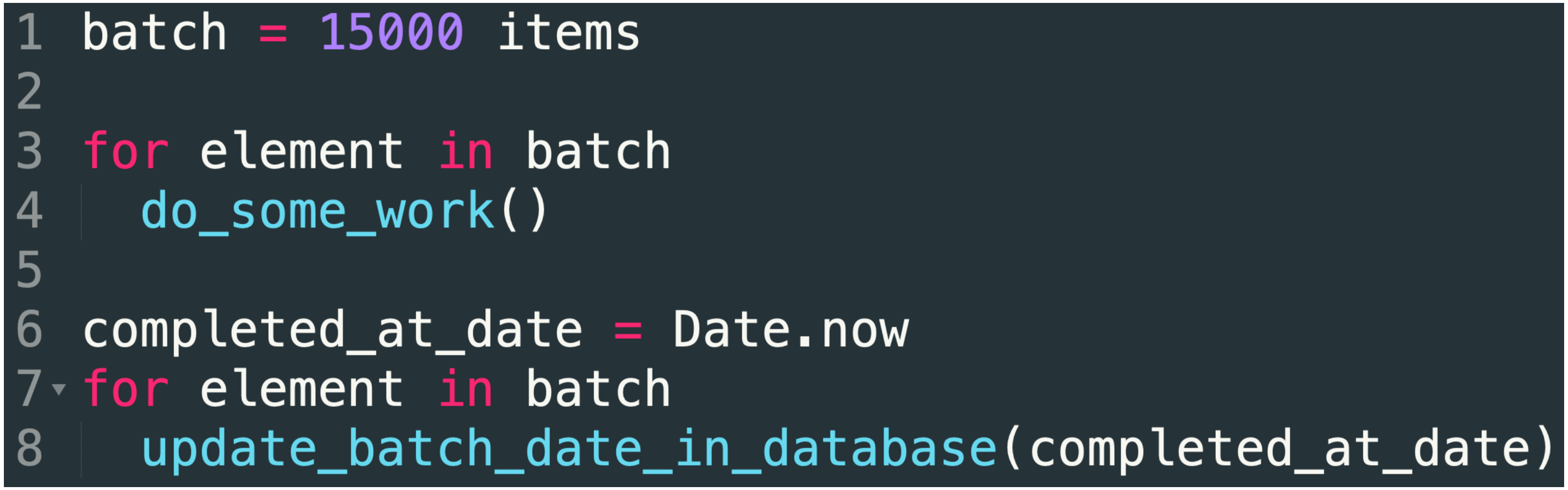


```
SELECT [list] FROM
    [Table A] A
INNER JOIN
    [Table B] B
ON A.Value = B.Value
```

Transactional databases

and batch processing

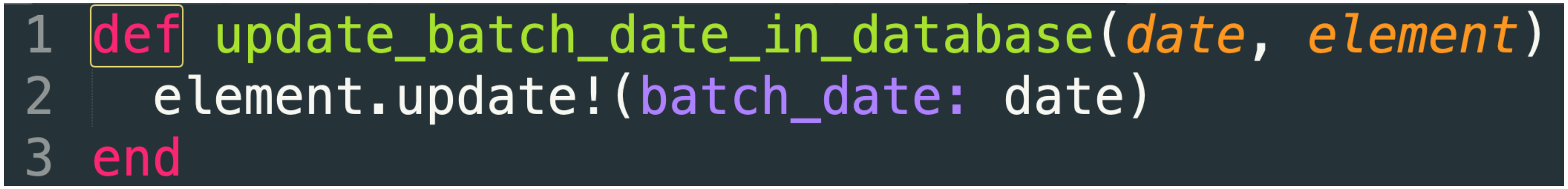

BEGIN TRANSACTION
ACTIVE
END TRANSACTION
PARTIALLY COMMITTED
COMMIT
COMMITTED
ABORT
ABORT
TERMINATED
FAILED
ABORTED



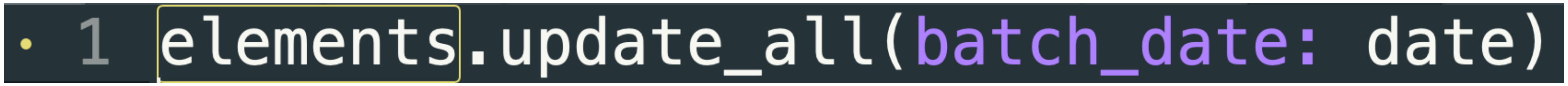
```
batch = 15000 items

for element in batch
  do_some_work()

completed_at_date = Date.now
for element in batch
  update_batch_date_in_database(completed_at_date)
```

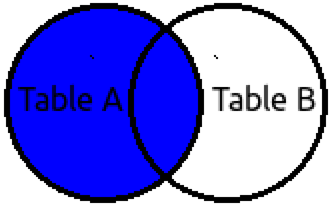


```ruby
def update_batch_date_in_database(date, element)
  element.update!(batch_date: date)
end
```

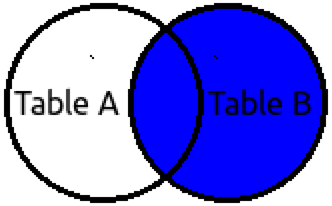


```ruby
elements.update_all(batch_date: date)
```

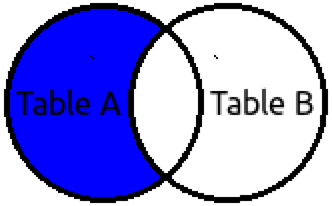
Early reduction



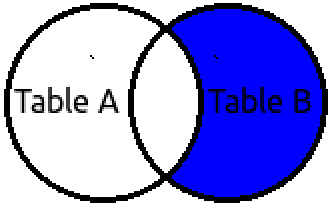
```
SELECT [list] FROM
    [Table A] A
LEFT JOIN
    [Table B] B
ON A.Value = B.Value
```



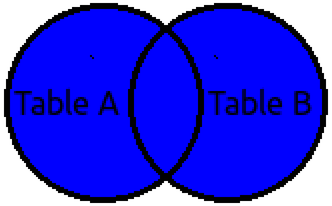
```
SELECT [list] FROM
    [Table A] A
RIGHT JOIN
    [Table B] B
ON A.Value = B.Value
```



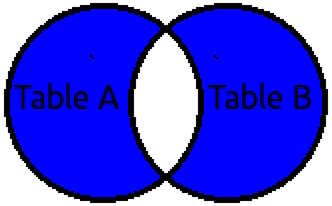
```
SELECT [list] FROM
    [Table A] A
LEFT JOIN
    [Table B] B
ON A.Value = B.Value
WHERE B.Value IS NULL
```



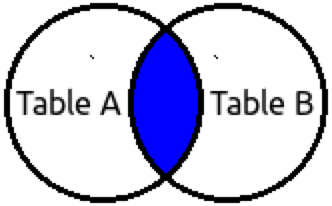
```
SELECT [list] FROM
    [Table A] A
RIGHT JOIN
    [Table B] B
ON A.Value = B.Value
WHERE A.Value IS NULL
```



```
SELECT [list] FROM
    [Table A] A
FULL OUTER JOIN
    [Table B] B
ON A.Value = B.Value
```



```
SELECT [list] FROM
    [Table A] A
FULL OUTER JOIN
    [Table B] B
ON A.Value = B.Value
WHERE A.Value IS NULL
OR B.Value IS NULL
```



```
SELECT [list] FROM
    [Table A] A
INNER JOIN
    [Table B] B
ON A.Value = B.Value
```

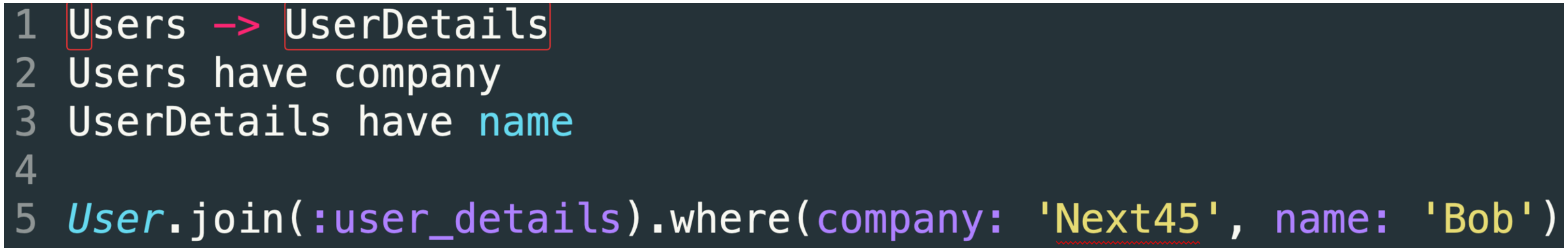
Rely on your framework

Dont rely on your framework

Offloading the computation

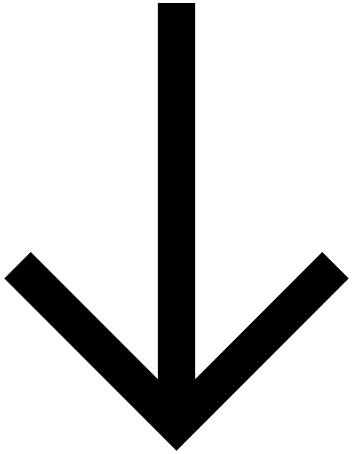
One solution now is not a solution later on

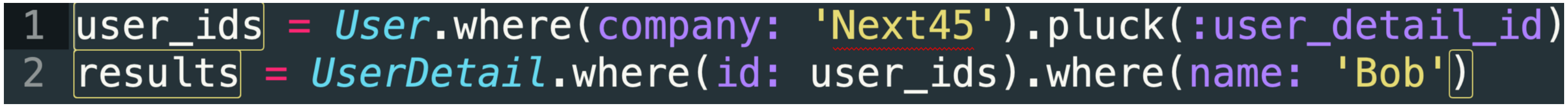
The magic `.pluck`



```
Users -> UserDetails
Users have company
UserDetails have name

User.join(:user_details).where(company: 'Next45', name: 'Bob')
```





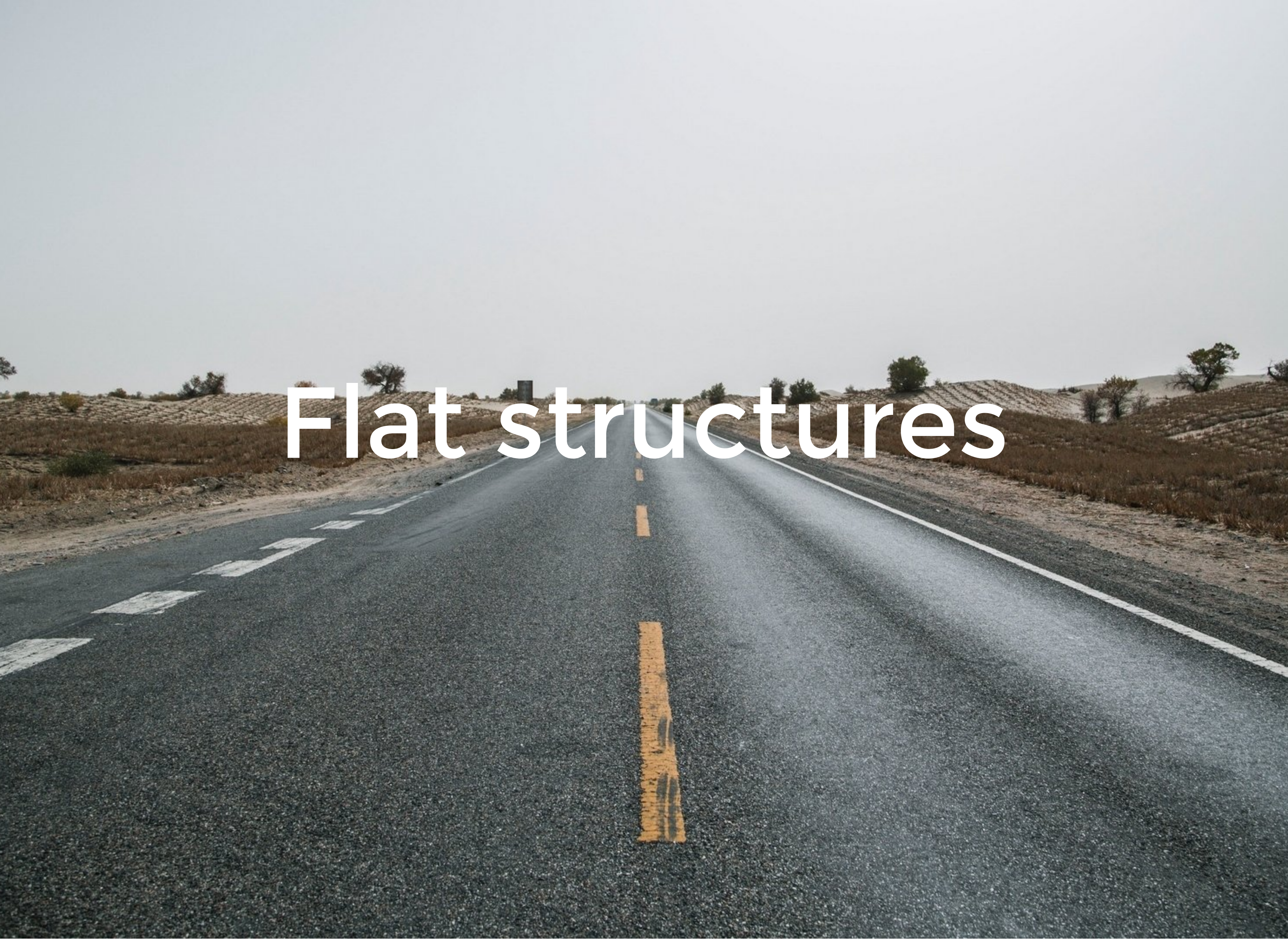
```
user_ids = User.where(company: 'Next45').pluck(:user_detail_id)
results = UserDetail.where(id: user_ids).where(name: 'Bob')
```

2. Architecture

Why solution x?

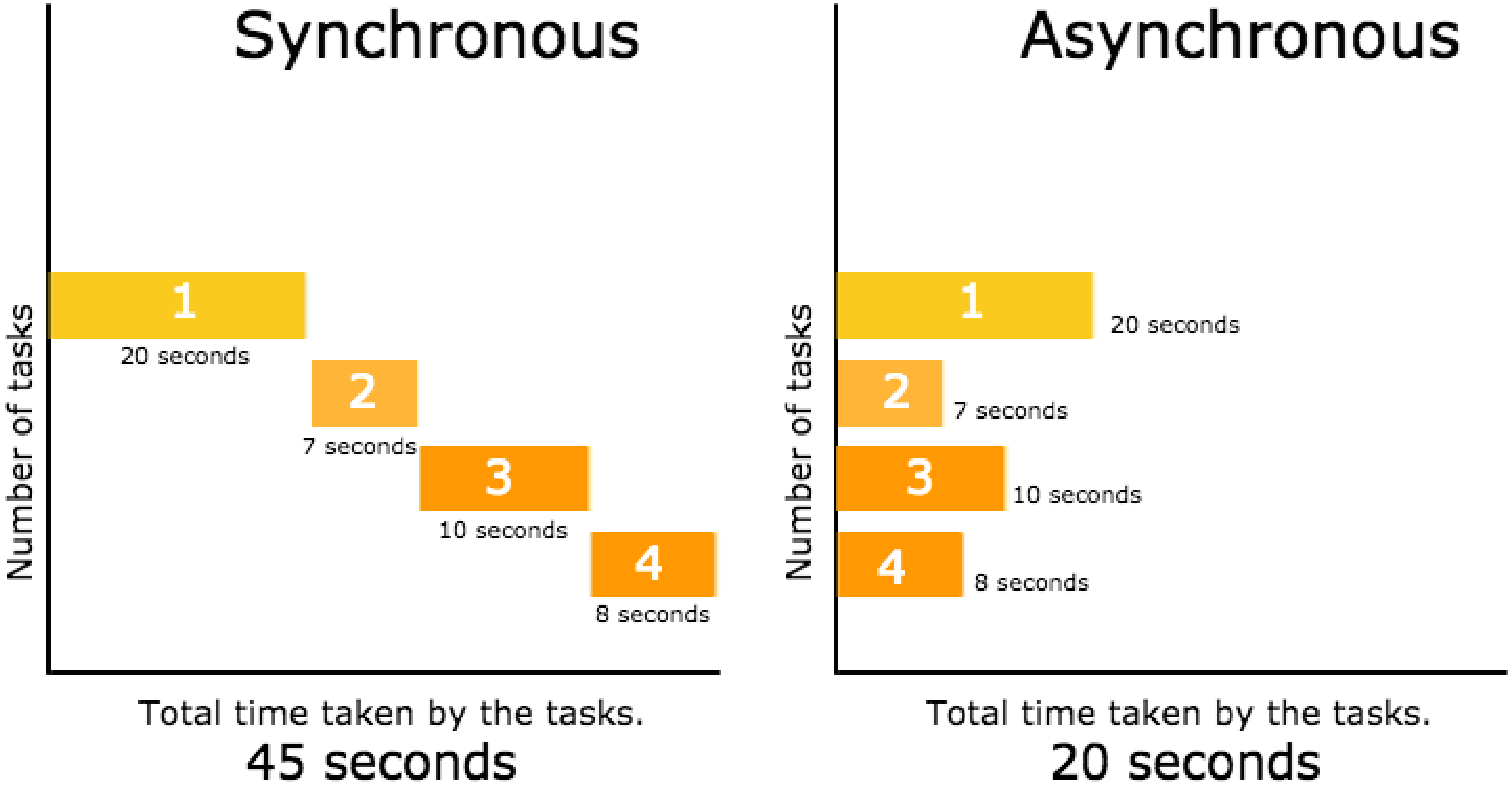
NoSQL

Flat structures



ElasticSearch

Async rather than sync


Synchronous
Number of tasks
1
20 seconds
2
7 seconds
3
10 seconds
4
8 seconds
Total time taken by the tasks.
45 seconds
Asynchronous
Number of tasks
1
20 seconds
2
7 seconds
3
10 seconds
4
8 seconds
Total time taken by the tasks.
20 seconds

3. Dependencies

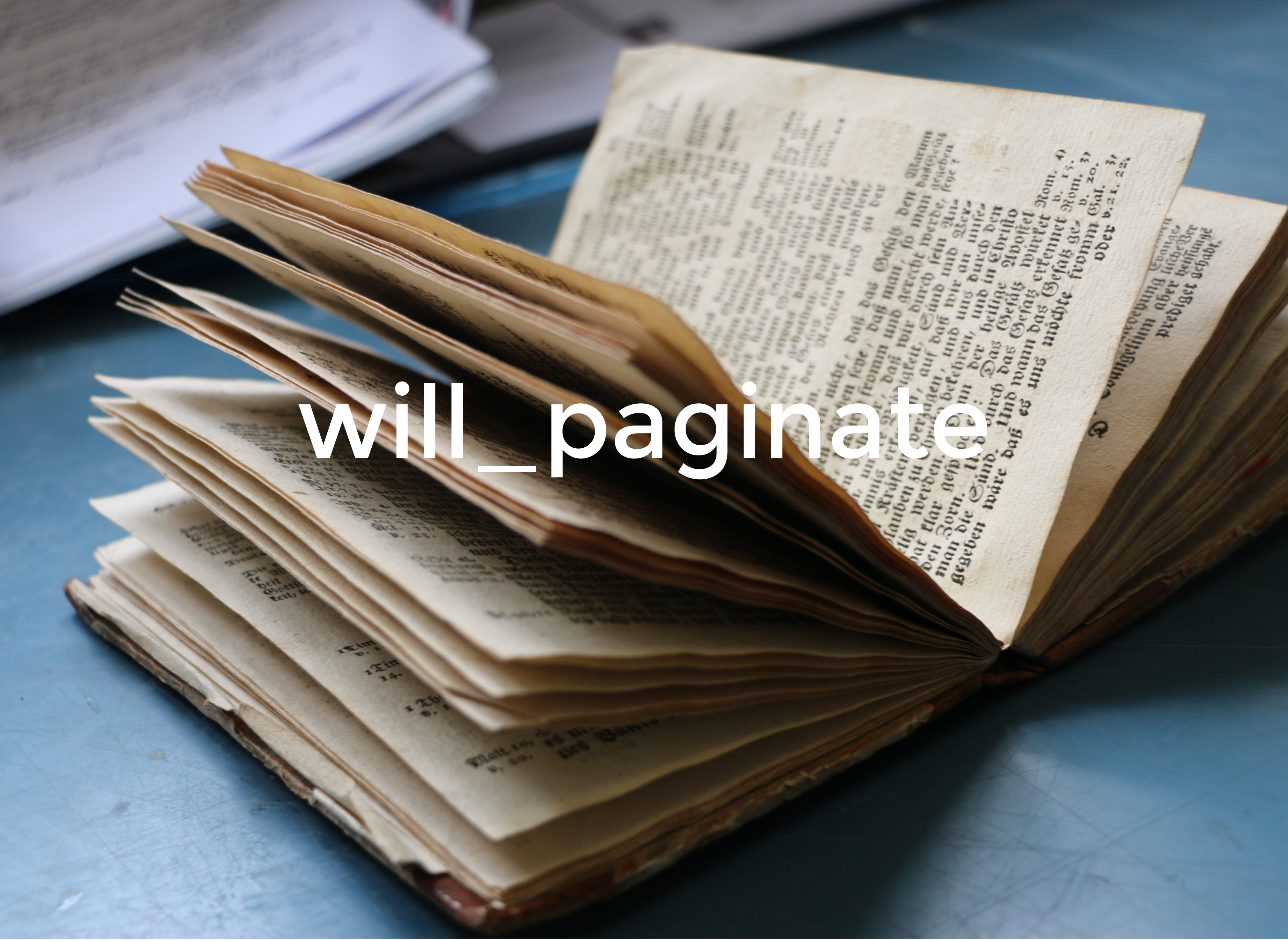
Pros

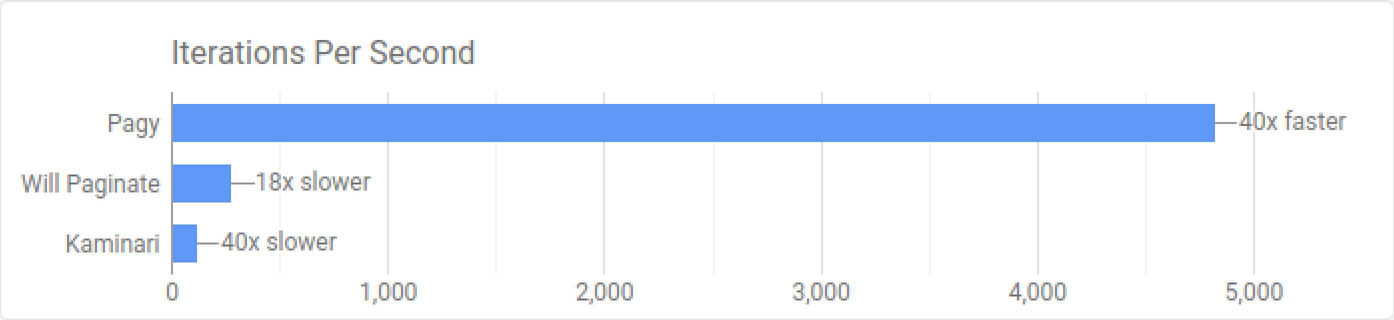
- Makes development faster
- Someone else has solved the problem
- Abstraction

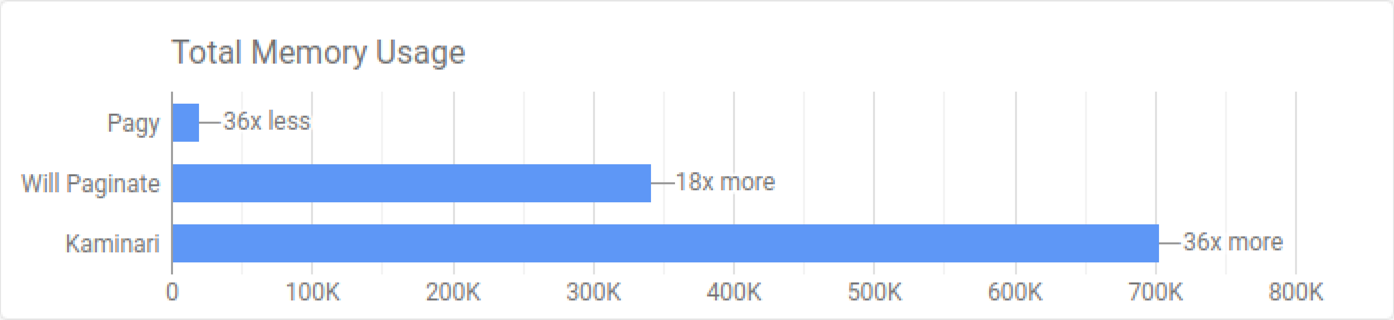
Cons

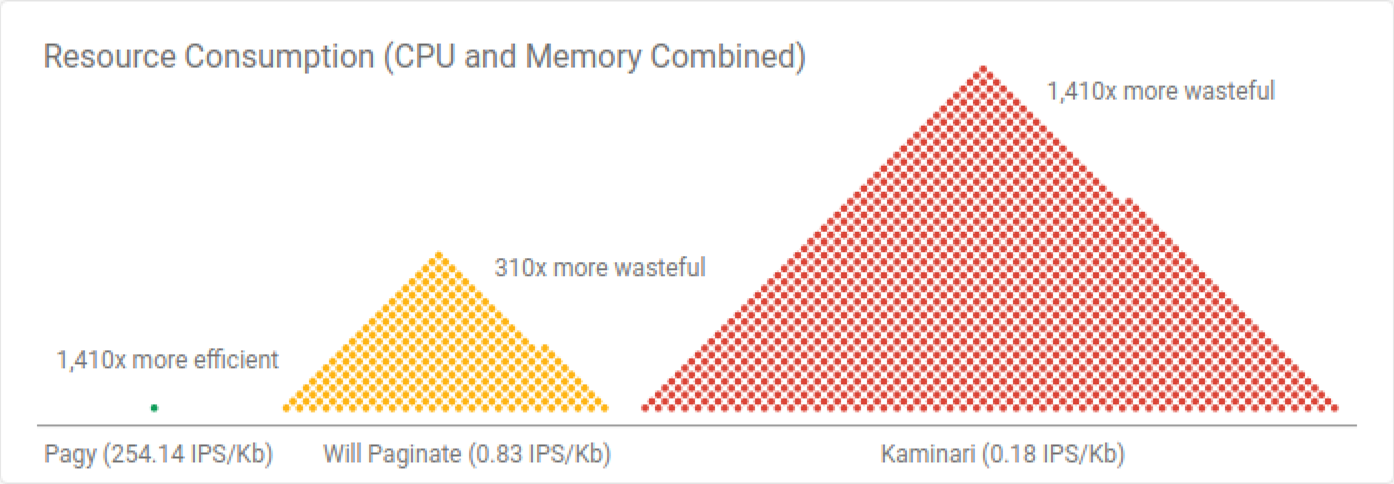
- Have to maintain
- May have own internal bugs
- May no longer be maintained

will_paginate









*claimed by pagy

Metrics that were in place failed here

- Experimental results proved value of the change

Stability

Headroom

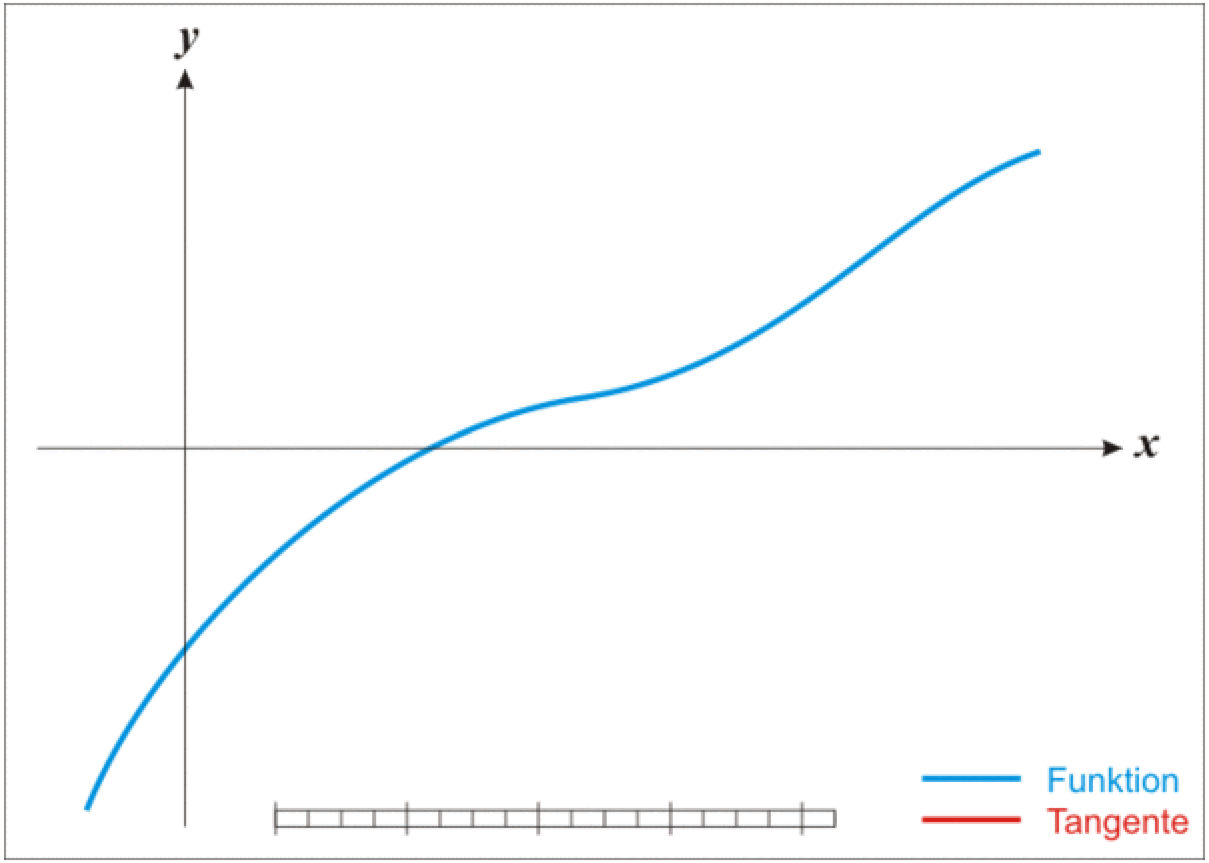
Less Bloat

Lessons:

- Sometimes do it yourself
- Experiment on hunches
- Always re-evaluate your strategies when they appear to not be working

4. Precision

Error of computation



Takeaways

Hmm Pizza ...

- Performance issues are healthy
- Solutions => time and cost
- Scaling does not always work
- Follow convention more than not
- Always trade-offs
- What works today may not tomorrow

Thank you