Data-Oriented Django

Adam Johnson

Computers are friggin’ fast

3.2 billion operations per second

3.2 billion operations per second

× 10 cores

3.2 billion operations per second

× 10 cores

= 32 billion operations per second

Grug-brained Adam

14191229747369282 79

Correction:

141912297473692827 9

Grug-brained Adam

```
  1419122974736928279
+ 9350264782187121390
```

55 seconds...

Grug-brained Adam

```
  1419122974736928279
+ 9350264782187121390
```

55 seconds…

= 10769387766924049669

Grug-brained Adam

1419122974736928279

+ 9350264782187121390

55 seconds...

= 10769387766924049669

❌

Adam: 0.018 ops/sec, 0% accuracy

Macbook: 32 billion ops/sec, 100% accuracy

Where does all that speed go?

Data-Oriented Design

It's not…

It’s not...

- Domain-Driven Design

It’s not…

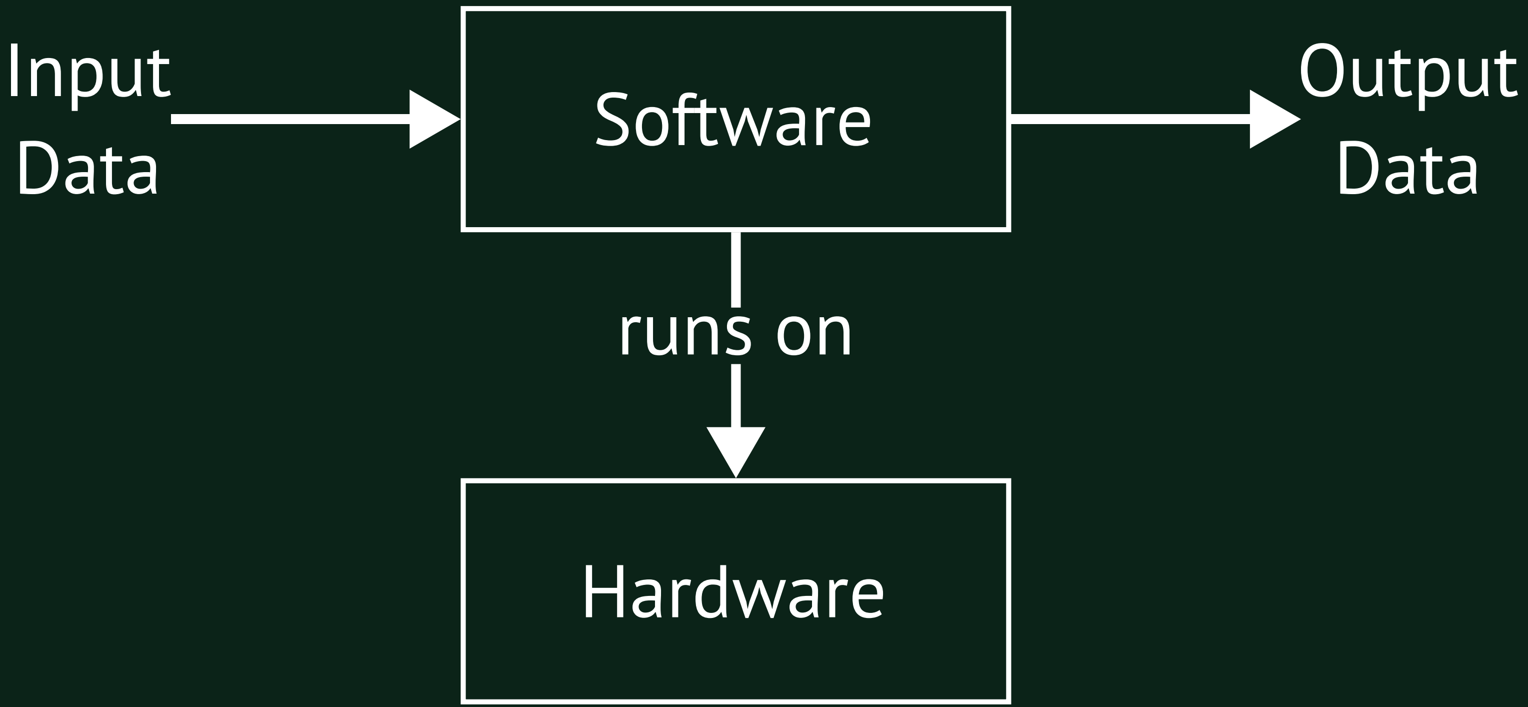
- Domain-Driven Design
- Data-Driven Design

It’s not...

- Domain-Driven Design
- Data-Driven Design
- Data-Oriented Programming

Data-Oriented Design

🧘 *Shoshin*: Beginner’s mind 🧘


Input Data
Software
Output Data
runs on
Hardware

Software’s only job is to *transform data*

Users only care about *getting their output data*

Data Characteristics

- Format
- Volume
- Latency
- Throughput
- Statistical distribution

Context is Everything

por exemplo

Check if a number exists in a given set

por exemplo

Check if a number exists in a given set

```python
numbers: set[int]

def is_match(number: int) -> bool:
    return number in numbers
```

What if there’s just a single number?

What if there’s just a single number?

```python
def is_match(number: int) -> bool:
    return number == 42
```

What if a billion numbers?

What if a billion numbers?

```python
def is_match(number: int) -> bool:
    with connection.cursor():
        cursor.execute(
            "SELECT 1 FROM numbers WHERE n = %s",
            (number,),
        )
        return (cursor.fetchone() is not None)
```

What if matching many numbers against few?

What if matching many numbers against few?

```python
numbers: set[int]

def matches(sought: set[int]) -> set[int]:
    return numbers.intersection(sought)
```

What if set is all odd numbers between 1 and 1 million?

What if false positives are acceptable? (bloom filter)

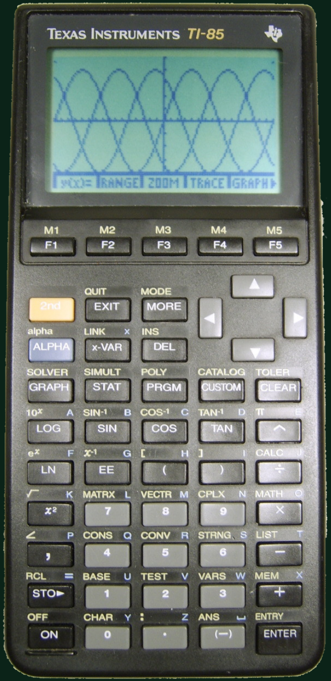
...

Implementation depends on data characteristics

Software's only job is to transform data using *specific hardware*

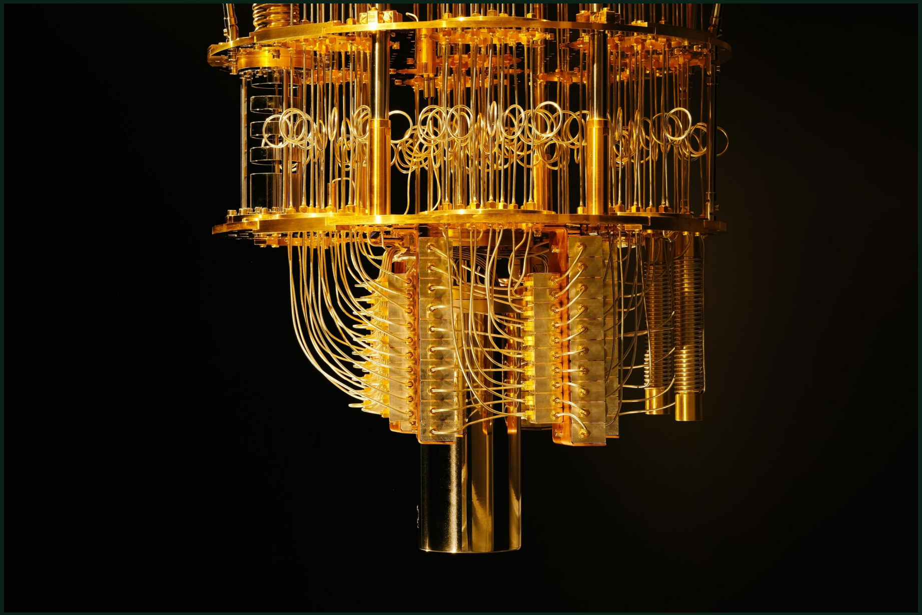
What hardware?

What hardware?



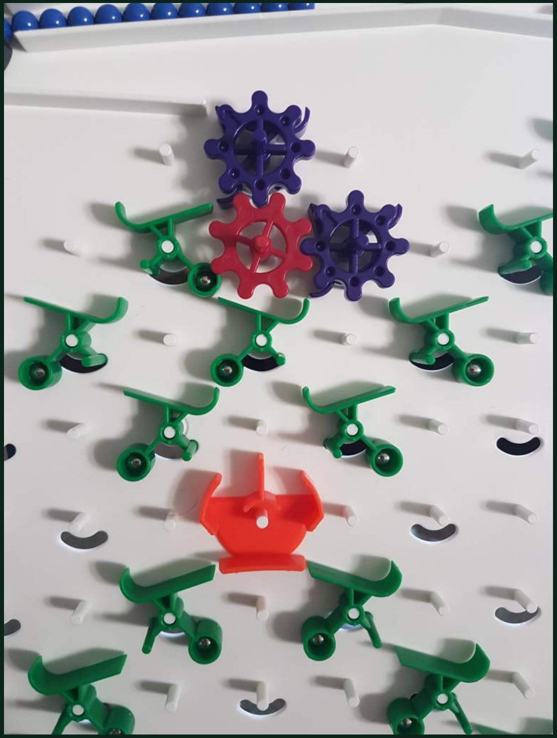
TI-85?

What hardware?



Quantum Computer?

What hardware?

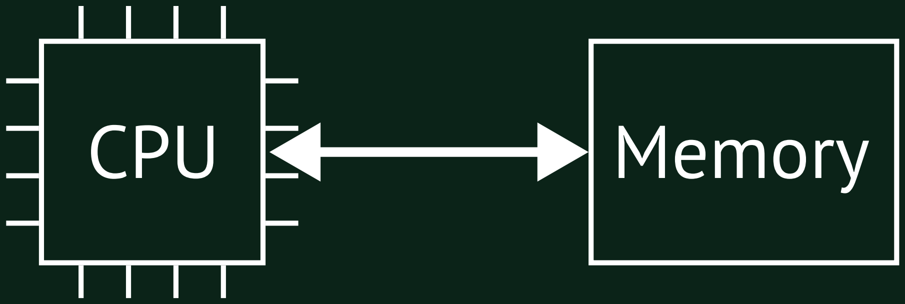


Turing Tumble?

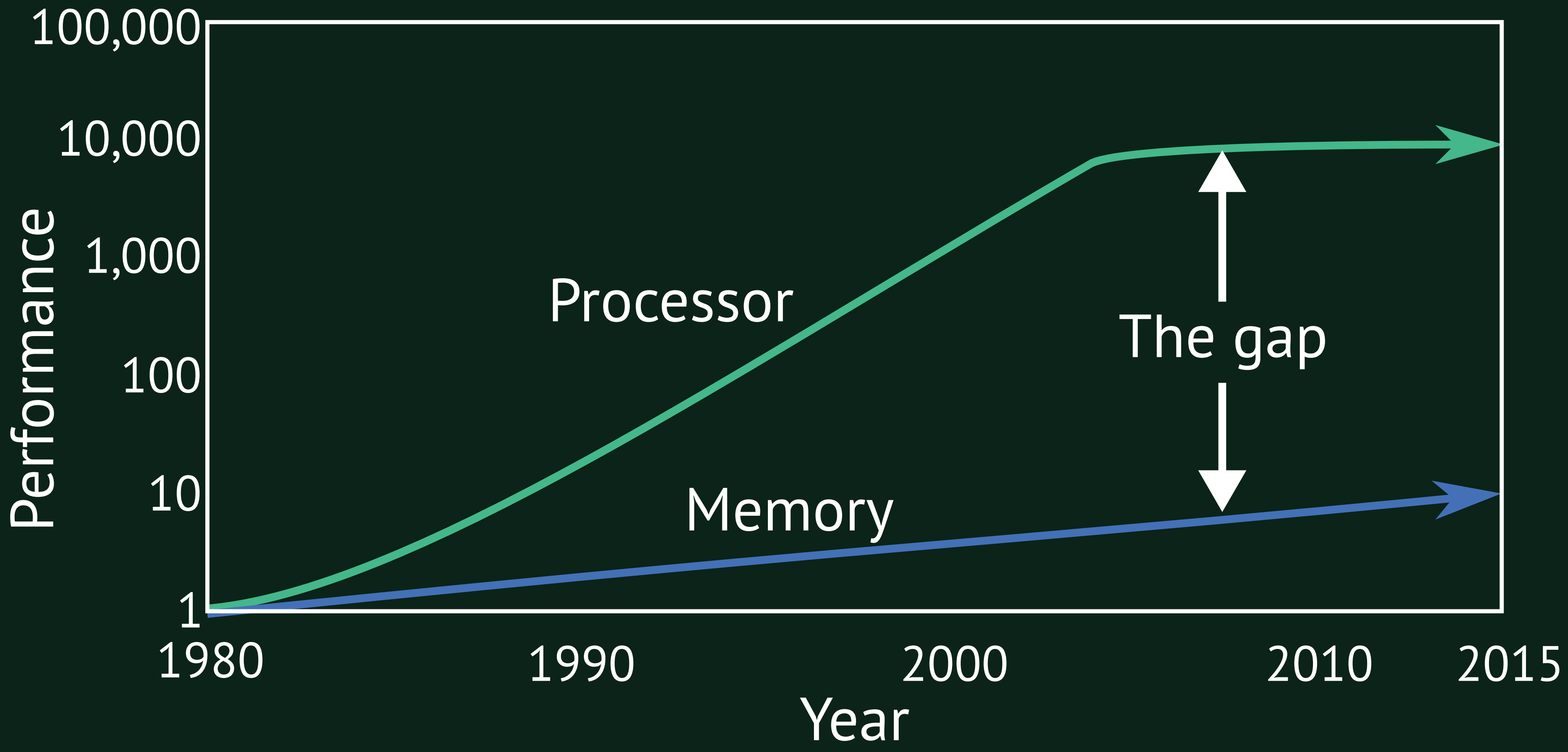
What hardware?

💁‍♂️ General-purpose CPU

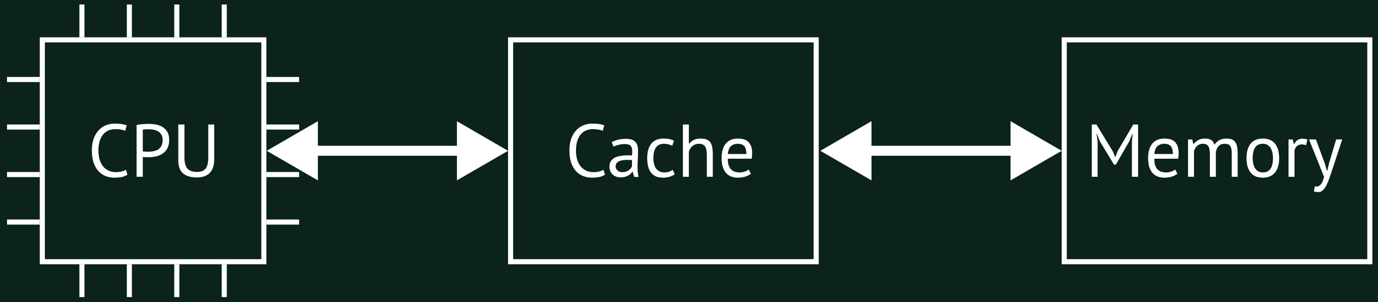
General-purpose CPU



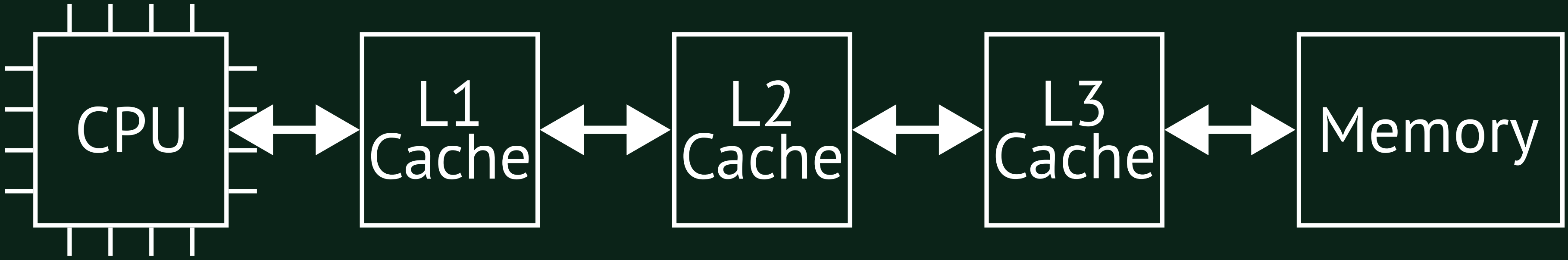
Processor-memory gap



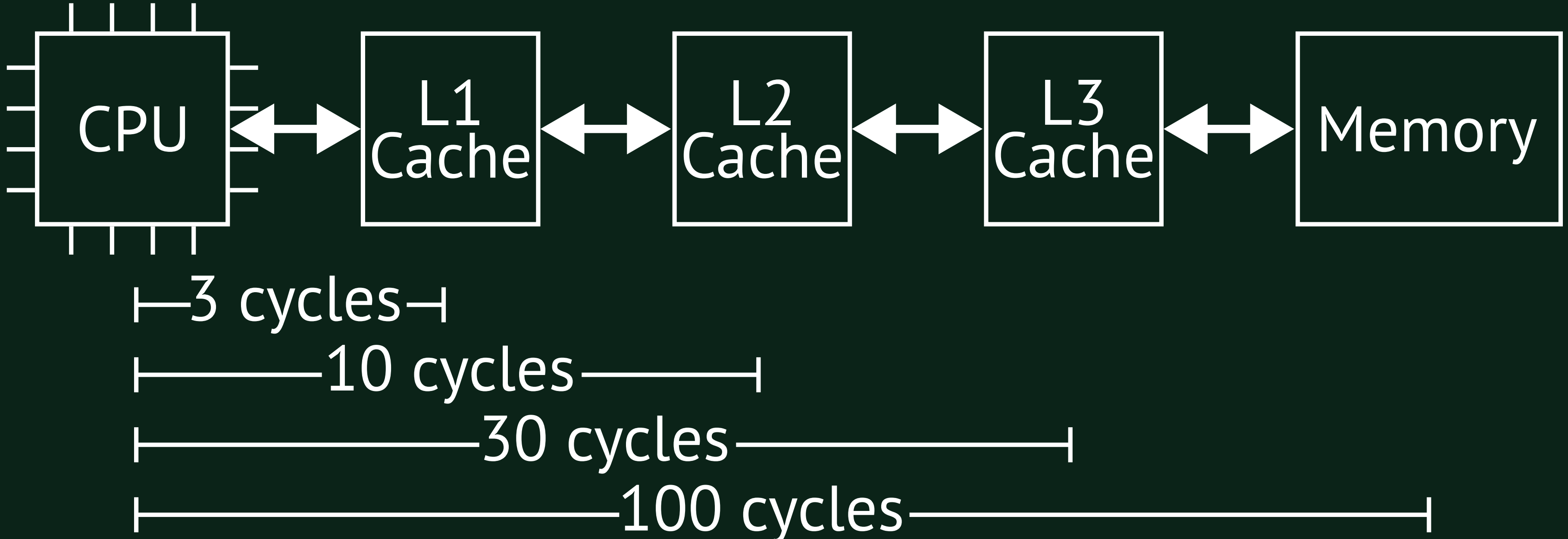
CPU caches



Three levels of cache!



Three levels of cache!



Implications

- Use smaller representations
- Lay out data in access order

Python’s built-in types are big

```
In [1]: 64 // 8
Out[1]: 8

In [2]: import sys

In [3]: sys.getsizeof(9001)
Out[3]: 28
```

Python’s built-in types are big

```
In [4]: numbers = list(range(1000))

In [5]: len(numbers) * 8
Out[5]: 8000

In [6]: (
   ...:     sys.getsizeof(numbers)
   ...:     + sum(sys.getsizeof(n) for n in numbers)
   ...: )
Out[6]: 36052
```

C data types much slimmer

```
In [7]: import array

In [8]: numbers2 = array.array('Q', range(1000))

In [9]: sys.getsizeof(numbers2)
Out[9]: 8320
```

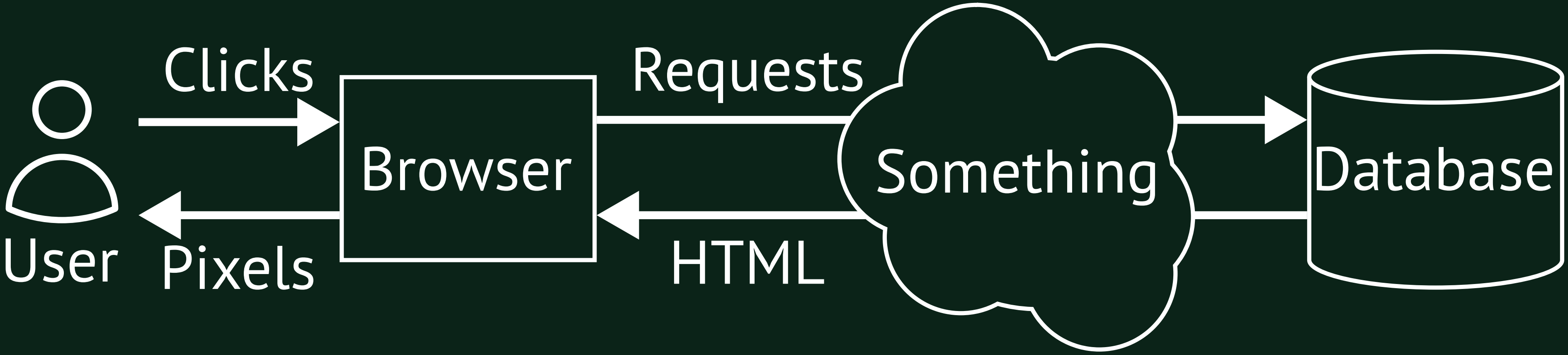
C data types much slimmer

```
In [7]: import array

In [8]: numbers2 = array.array('Q', range(1000))

In [9]: sys.getsizeof(numbers2)
Out[9]: 8320
```

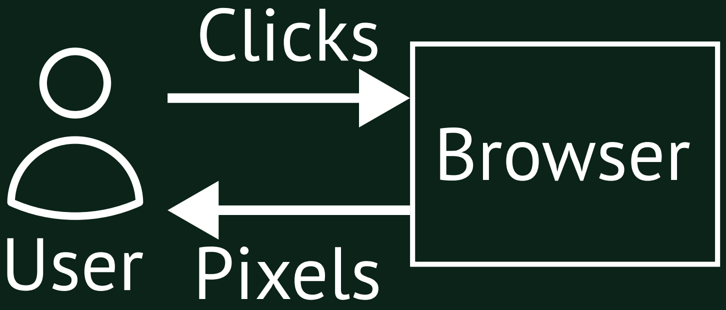
Normally use numpy or pandas for arrays

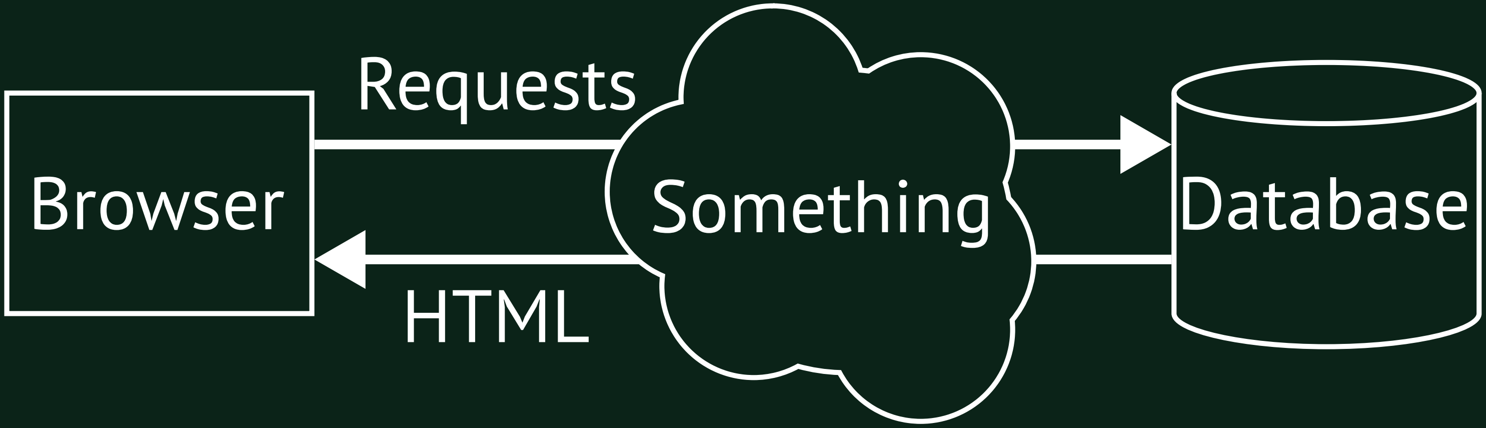
Data-Oriented Design

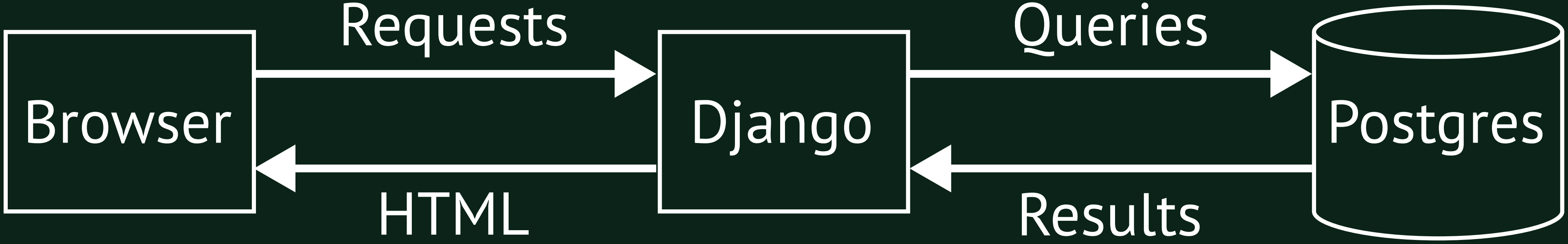
for the Web

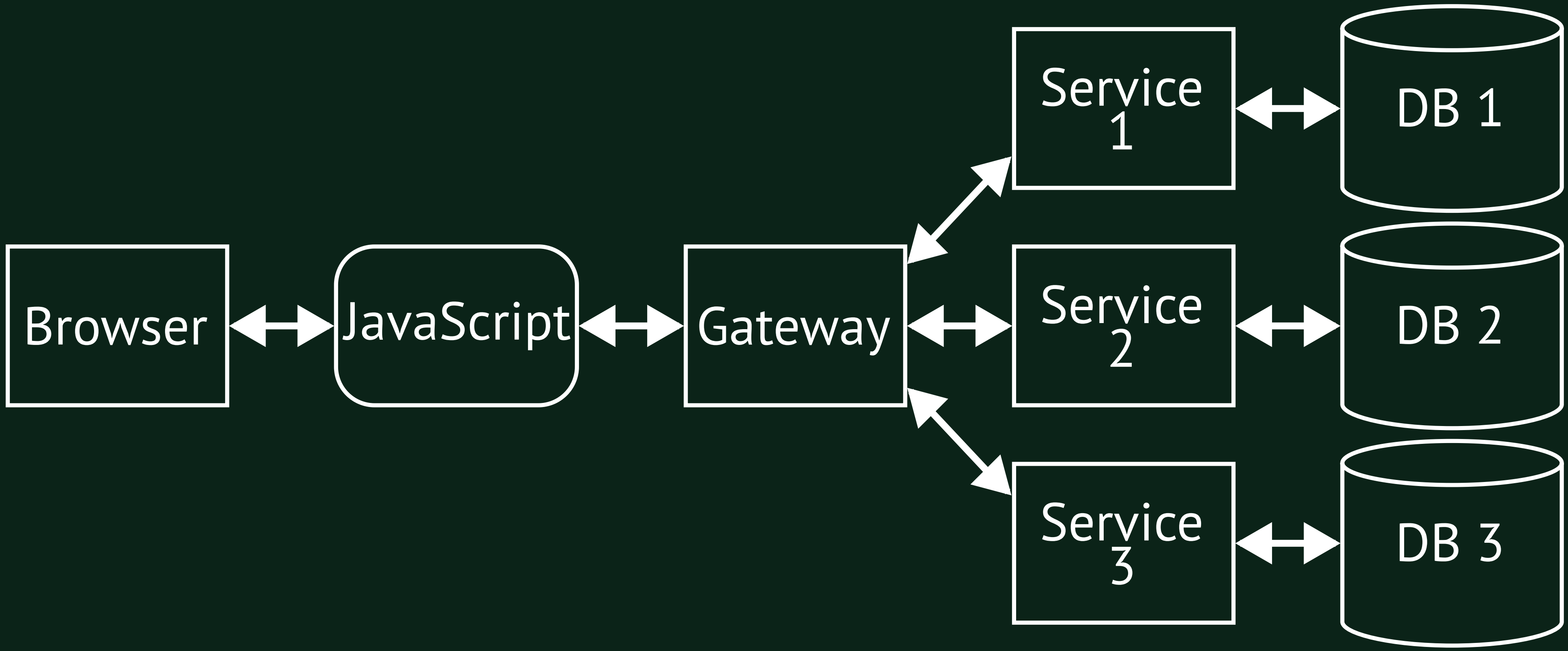

Clicks
Requests
User
Browser
Something
Database
Pixels
HTML

already hella fast




Requests
Browser
Something
Database
HTML


Requests
Queries
Browser
Django
Postgres
HTML
Results


Browser
JavaScript
Gateway
Service 1
Service 2
Service 3
DB 1
DB 2
DB 3

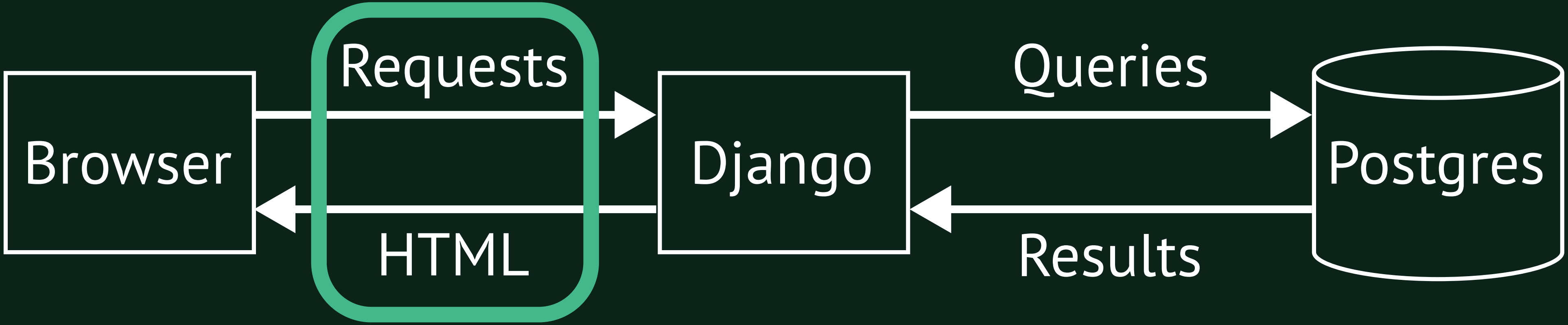
How fast?

⩽100ms - 👌👌

⩽1s - 👌

⩽3s - 🫳 - at 3s lose ~50% of visitors

>10s - users likely to retry, or give up


Requests
Queries
Browser
Django
Postgres
HTML
Results

Speed up request/response cycle

Speed up request/response cycle

1. Write minimal, performant HTML

Speed up request/response cycle

1. Write minimal, performant HTML
2. HTTP Caching

Speed up request/response cycle

1. Write minimal, performant HTML
2. HTTP Caching
3. HTTP/3, or at least 2

Speed up request/response cycle

1. Write minimal, performant HTML
2. HTTP Caching
3. HTTP/3, or at least 2
4. Response compression (`GZipMiddleware`)

Speed up request/response cycle

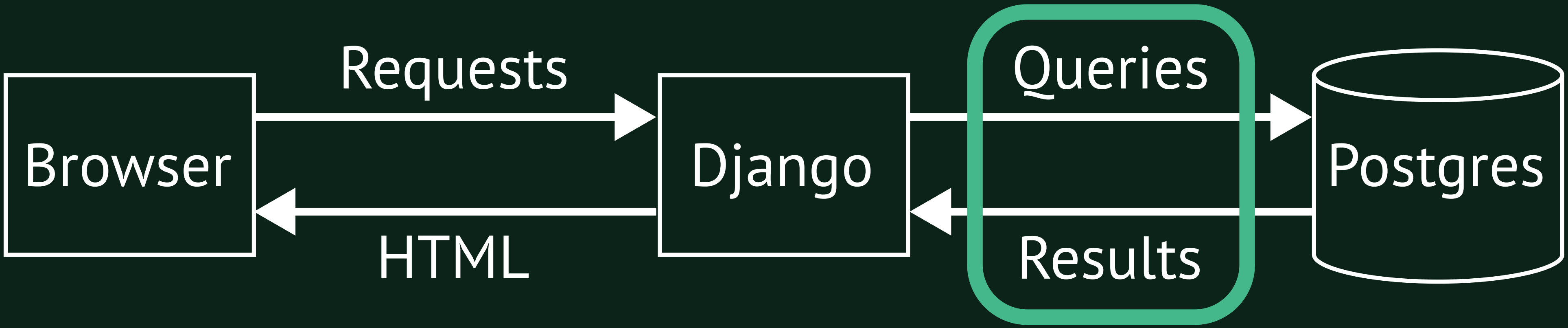
1. Write minimal, performant HTML
2. HTTP Caching
3. HTTP/3, or at least 2
4. Response compression (`GZipMiddleware`)
5. HTML minification (django-minify-html)

Speed up request/response cycle

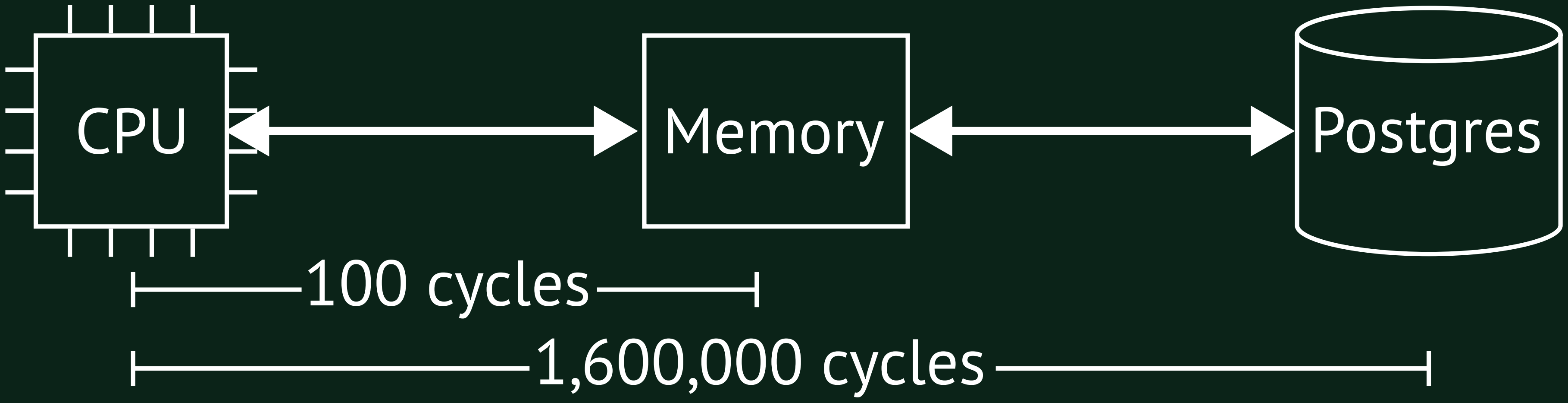
1. Write minimal, performant HTML
2. HTTP Caching
3. HTTP/3, or at least 2
4. Response compression (`GZipMiddleware`)
5. HTML minification (django-minify-html)
6. Use a CDN (Content Delivery Network)

Resources

- MDN: Web Performance
- web.dev/learn
- WebPageTest.org
- web.dev/measure


Browser
Requests
HTML
Django
Queries
Results
Postgres

Speed up query/result cycle


CPU
Memory
Postgres
100 cycles
1,600,000 cycles

Speed up queries and results

Speed up queries and results

1. Avoid N+1 queries

Speed up queries and results

1. Avoid N+1 queries
2. Split models

Speed up queries and results

1. Avoid N+1 queries
2. Split models
3. Multiple counts in one pass

1. Avoid N+1 queries

```python
books = Book.objects.order_by("title")
for book in books:
    print(book.title, "by", book.author.name)
```

1. Avoid N+1 queries

```
books = Book.objects.order_by("title")
for book in books:
    print(book.title, "by", book.author.name)
```

1. Iterate `books`
2. For each of **N** books:
   fetch `book.author`

1. Avoid N+1 queries

```python
books = (
    Book.objects.order_by("title")
    .select_related("author")
)
for book in books:
    print(book.title, "by", book.author.name)
```

1. Avoid N+1 queries

```
books = (
    Book.objects.order_by("title")
    .select_related("author")
)
for book in books:
    print(book.title, "by", book.author.name)
```

1. Fetch books with author joined in

1. Avoid N+1 queries

`book.name`	`author.name`
The Hundred and One Dalmatians	Dodie Smith
The Lost World	Arthur Conan Doyle
The Hound of the Baskervilles	Arthur Conan Doyle
His Last Bow	Arthur Conan Doyle

1. Avoid N+1 queries

```python
books = (
    Book.objects.order_by("title")
    .prefetch_related("author")
)
for book in books:
    print(book.title, "by", book.author.name)
```

1. Avoid N+1 queries

```
books = (
    Book.objects.order_by("title")
    .prefetch_related("author")
)
for book in books:
    print(book.title, "by", book.author.name)
```

1. Fetch `books`
2. Fetch related `authors` for all books

1. Avoid N+1 queries

django-auto-prefetch

```python
books = Book.objects.order_by("title")
for book in books:
    print(book.title, "by", book.author.name)
```

1. Avoid N+1 queries

django-auto-prefetch

```
books = Book.objects.order_by("title")
for book in books:
    print(book.title, "by", book.author.name)
```

1. Fetch books
2. On first access of `book.author`:
   Fetch related `authors` for all books

2. Split models

```python
class User(AbstractUser):
    avatar = models.ImageField(...)
    ...
```

2. Split models

```python
class User(AbstractUser):
    avatar = models.ImageField(...)
    ...
```

Task: *Store user’s ACME access token and refresh token*

2. Split models

Not great:

```
class User(AbstractUser):
    avatar = models.ImageField(...)
    ...
    acme_access_token = models.TextField()
    acme_access_expires = models.DateTimeField()
    acme_refresh_token = models.TextField()
```

2. Split models

Not great:

```
class User(AbstractUser):
    avatar = models.ImageField(...)
    ...
    acme_access_token = models.TextField()
    acme_access_expires = models.DateTimeField()
    acme_refresh_token = models.TextField()
```

Slows down every place users are queried

2. Split models

```
class User(AbstractUser):
    avatar = models.ImageField(...)
    ...

class UserAcmeToken(models.Model):
    user = models.OneToOneField(User, primary_key=True)
    access_token = models.TextField()
    access_expires = models.DateTimeField()
    refresh_token = models.TextField()
```

👍👍👍

3. Multiple counts in one pass

```
published_count = (
    author.book_set.filter(verified=True).count()
)
unpublished_count = (
    author.book_set.filter(verified=False).count()
)
```

3. Multiple counts in one pass

```
counts = (
    author.book_set.aggregate(
        verified=Count('pk', filter=Q(verified=True)),
        unverified=Count('pk', filter=Q(verified=False)),
    )
)
```

Resources

- Docs: Database access optimization
- The Temple of Django Database Performance
- Post: “Django and the N+1 Queries Problem”
- django-debug-toolbar (or Kolo)

Data-Oriented Design

Software’s only job is to *transform data*

Users only care about *getting their output data*

Resources

- Mike Acton - Data-Oriented Design and C++
- Andrew Kelley - Practical DOD
- Andreas Fredriksson - Context is Everything

Thank you! 🤗

- Adam Johnson
- @adamchainz on GitHub & Twitter
- me@adamj.eu
- github.com/adamchainz/talk-data-oriented-django
- Books: **Boost Your Django DX** & **Speed Up Your Django Tests**