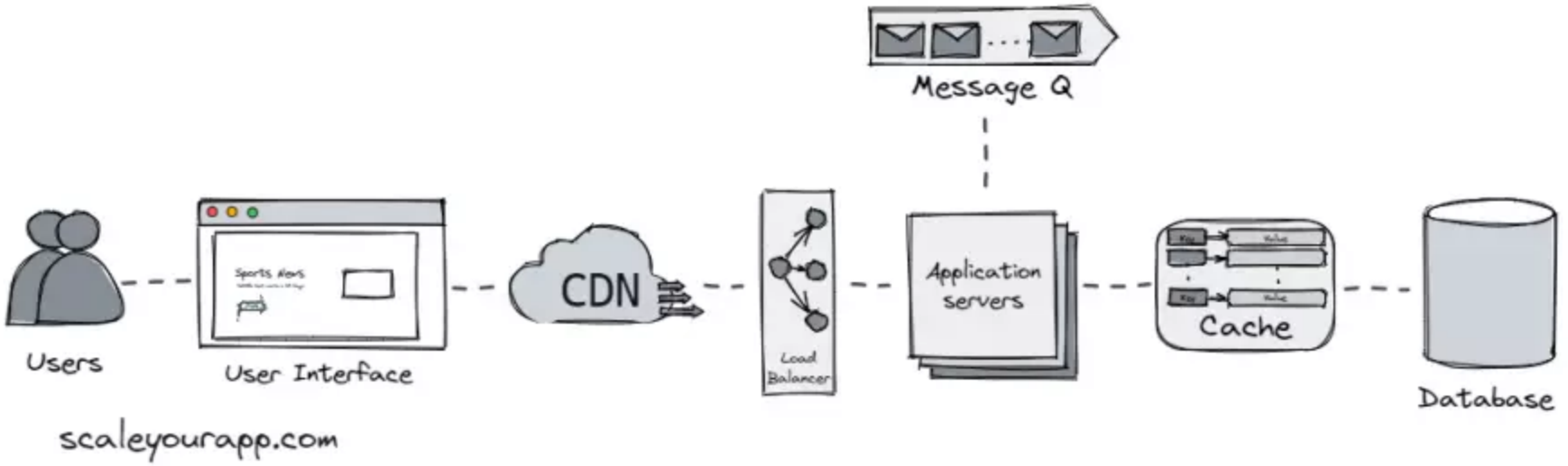
What is a database?



Generic Web Architecture

sqlite ~ 150 kloc

postgres ~ 500 kloc

```
file.write(data);
```

```
var total = 0;
for (var employee of employees) {
  total += employee.salary;
}
```

```
select sum(salary) from employee;
```

Why is a database?

Long-lived data.

What is a database?

- storage engine
- concurrency control
- query language

storage engine

dram vs disk

Intel Memory Latency Checker:

peak read bandwidth	42 gb/s
read latency	> 70ns
block size	64 b

(1s / 70ns) * 64b = 0.83 gb/s

```
var nums_xor: u64 = 0;
for (nums) |num| {
    nums_xor ^= num;
}
```

Expected:

- 14_000_000 reads/s
- 8 iters per read
- 112_000_000 iters/s
- (~= 0.83 gb/s)

```
var nums_xor: u64 = 0;
for (nums) |num| {
    nums_xor ^= num;
}
```

Actual:

- 3_106_399_506 iters/s
- (~= 23 gb/s)
- 27x better than expected!

prefetching

```
var nums_xor: u64 = 0;
var i: usize = 0;
while (i < nums.len) : (i += 1) {
    nums_xor ^= nums[random.uintLessThan(usize, nums.len)];
}
```

Expected:

- 14_000_000 reads/s
- 1 iter per read (read amplification)
- 14_000_000 iters/s

```
var nums_xor: u64 = 0;
var i: usize = 0;
while (i < nums.len) : (i += 1) {
    nums_xor ^= nums[random.uintLessThan(usize, nums.len)];
}
```

Actual:

- 51_067_426 iters/s
- 3.6x better than expected!

pipelining

speculation

fio

read bandwidth	2.6 gb/s
latency	50 us
block size	4 kb

(1s/50us) ~= 20_000 reads/s
(1s/50us) * 4kb ~= 78 mb/s

```
while (reads < block_count) {
    const offset = random.uintLessThan(u64, block_count) * block.len;
    const bytes_read = try std.os.pread(fd, block, offset);
    assert(bytes_read == block.len);
    reads += 1;
    for (block) |byte| {
        bytes_xor ^= byte;
    }
}
```

Expected:

- 20_000 reads/s

```
while (reads < block_count) {
    const offset = random.uintLessThan(u64, block_count) * block.len;
    const bytes_read = try std.os.pread(fd, block, offset);
    assert(bytes_read == block.len);
    reads += 1;
    for (block) |byte| {
        bytes_xor ^= byte;
    }
}
```

Actual:

- 18_644 reads/s
- worse than expected!

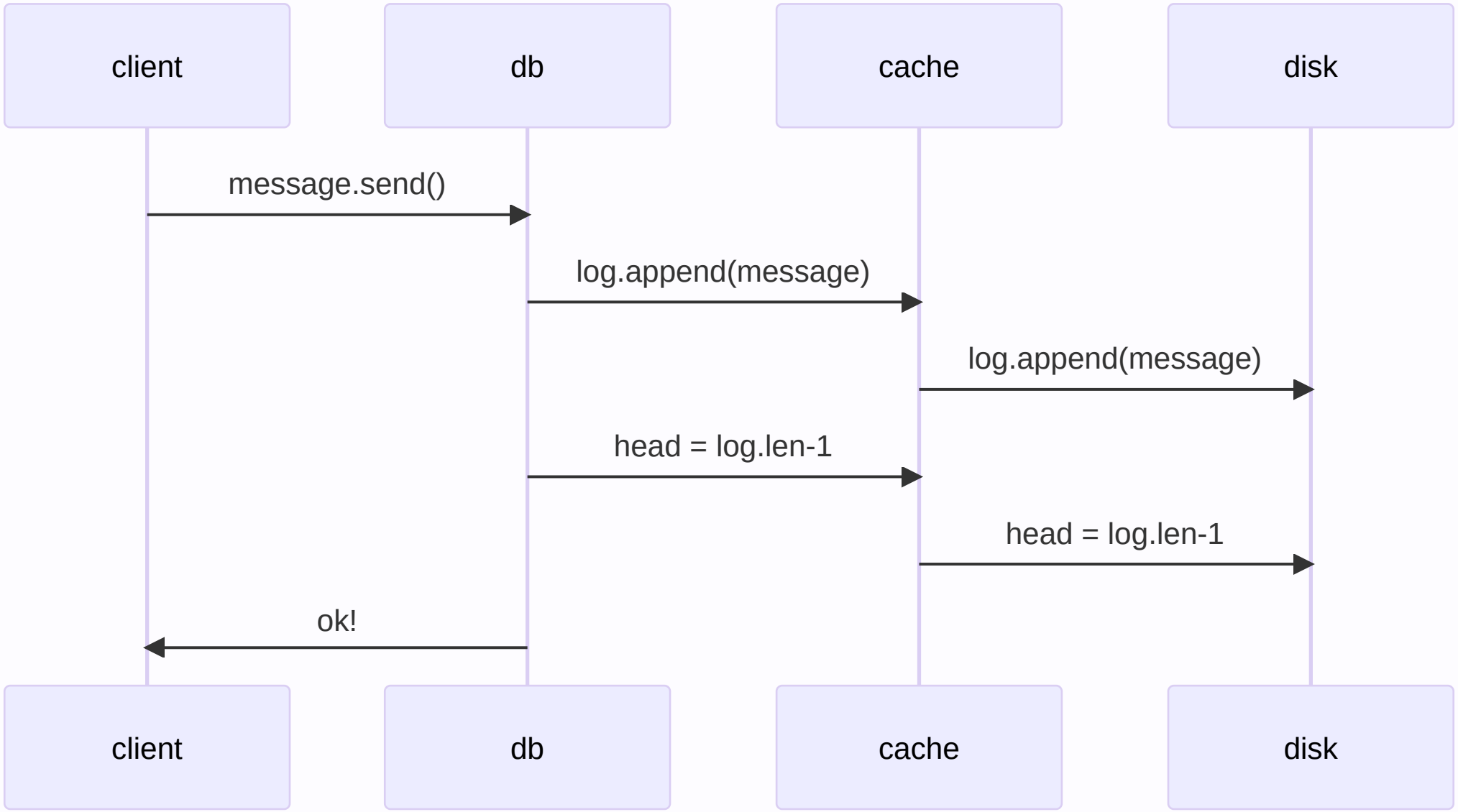
ordering

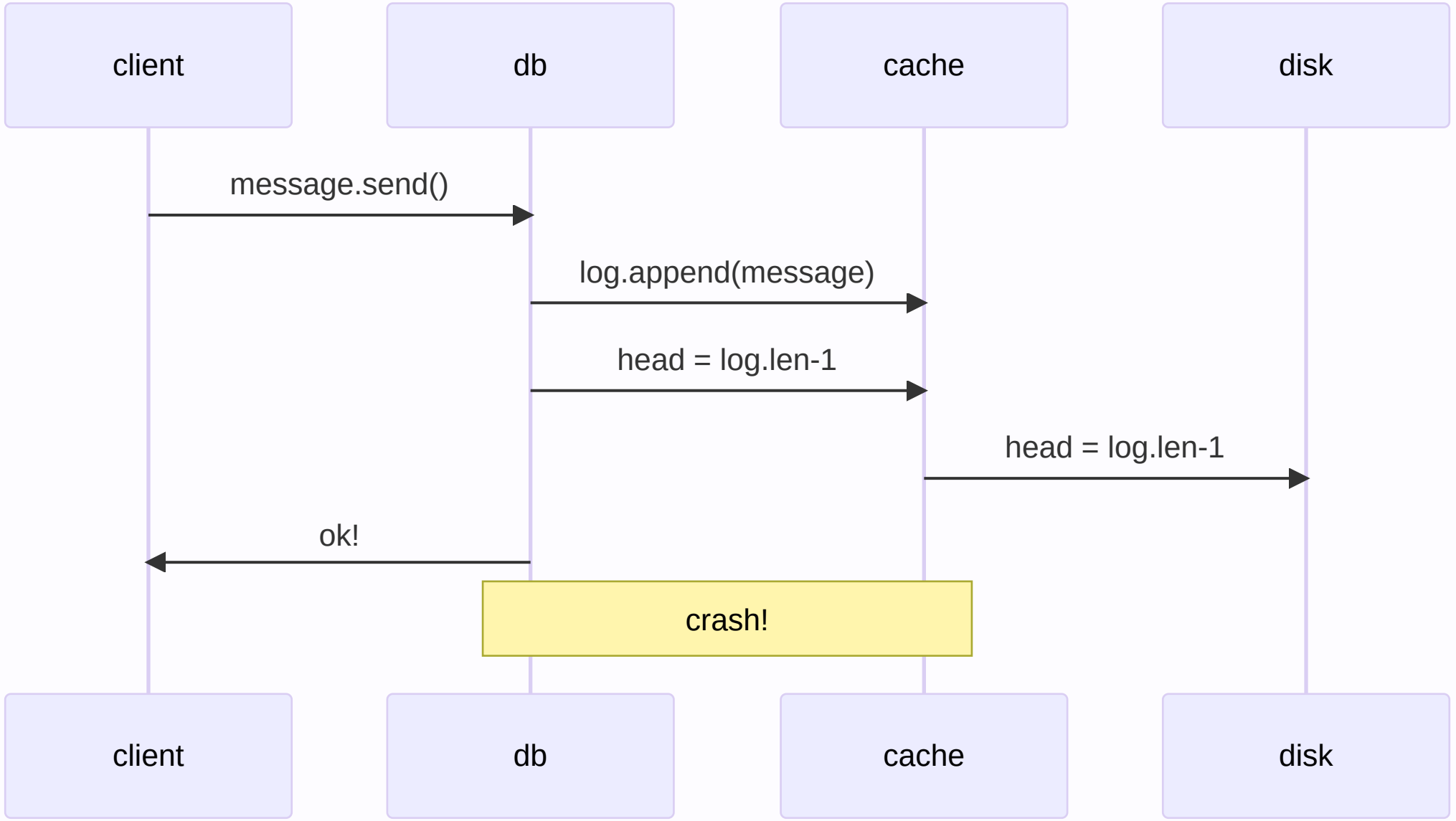
```
for (messages) |message| {
    log.append(message);
    head = log.len-1;
}
```

```
while (true) {
    print("{}", .{log.get(head)});
}
```

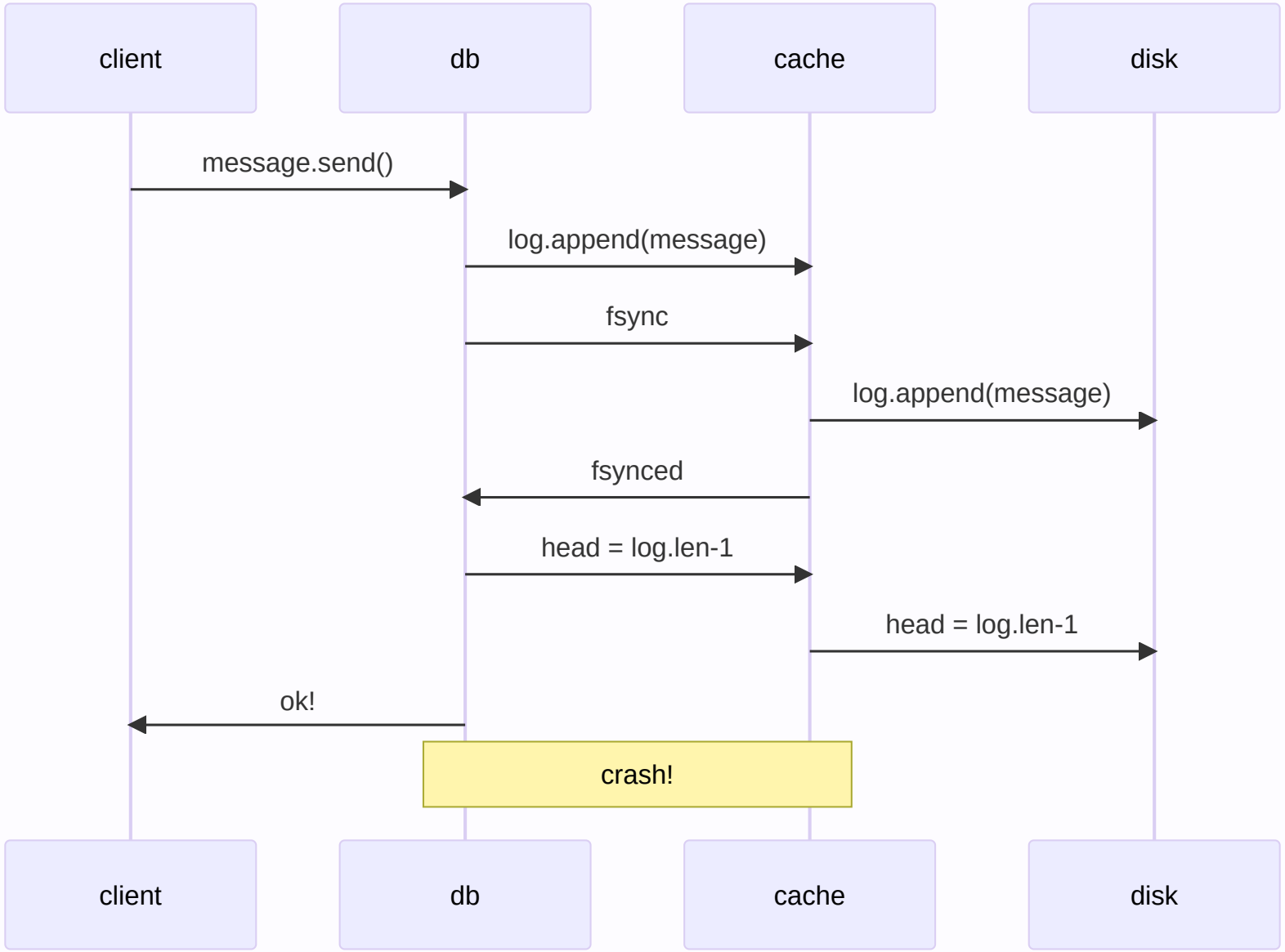
```
for (messages) |message| {
    log.append(message);
    @fence(.SeqCst);
    head = log.len-1;
}
```

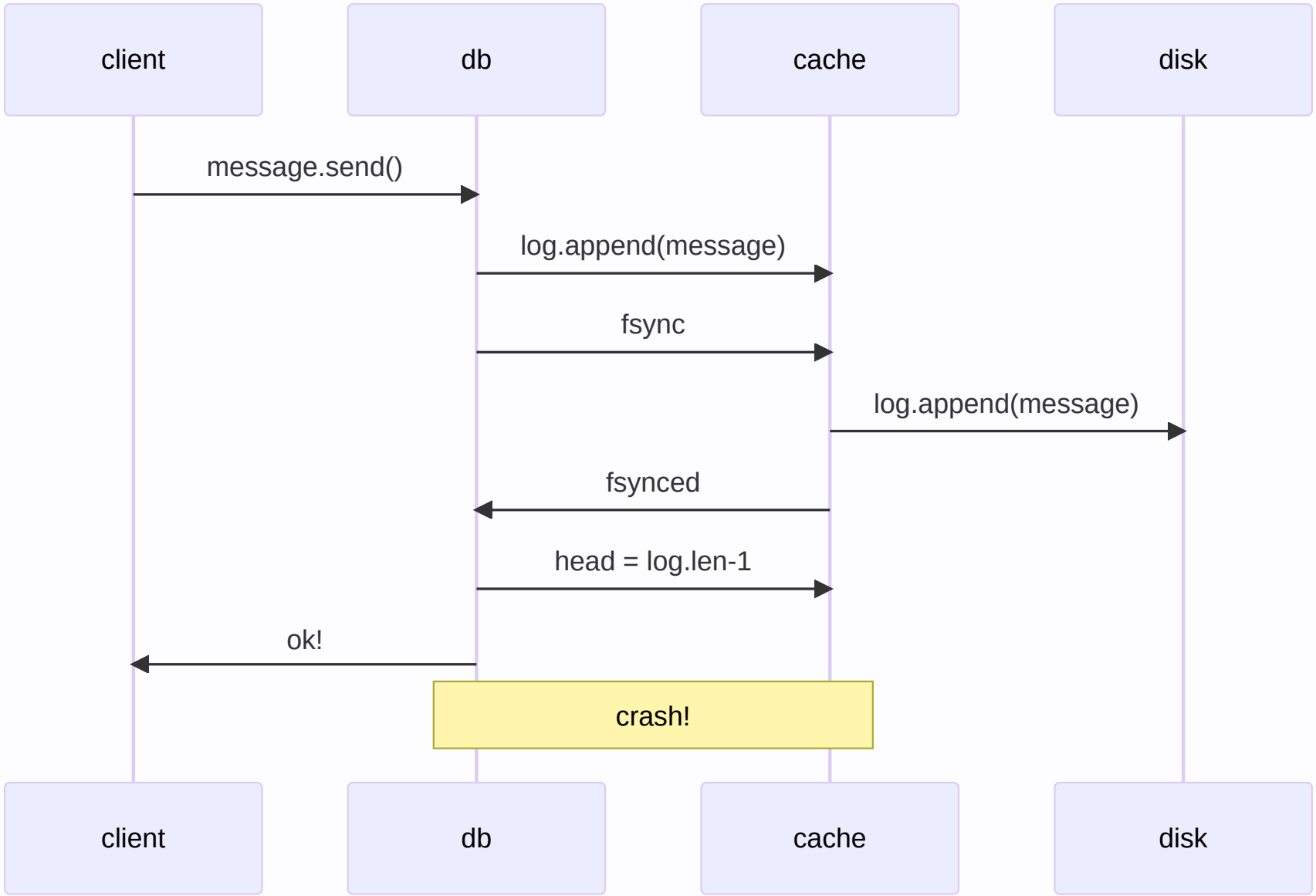
```
while (true) {
    const head_now = head;
    @fence(.SeqCst);
    print("{}", .{log.get(head_now)});
}
```

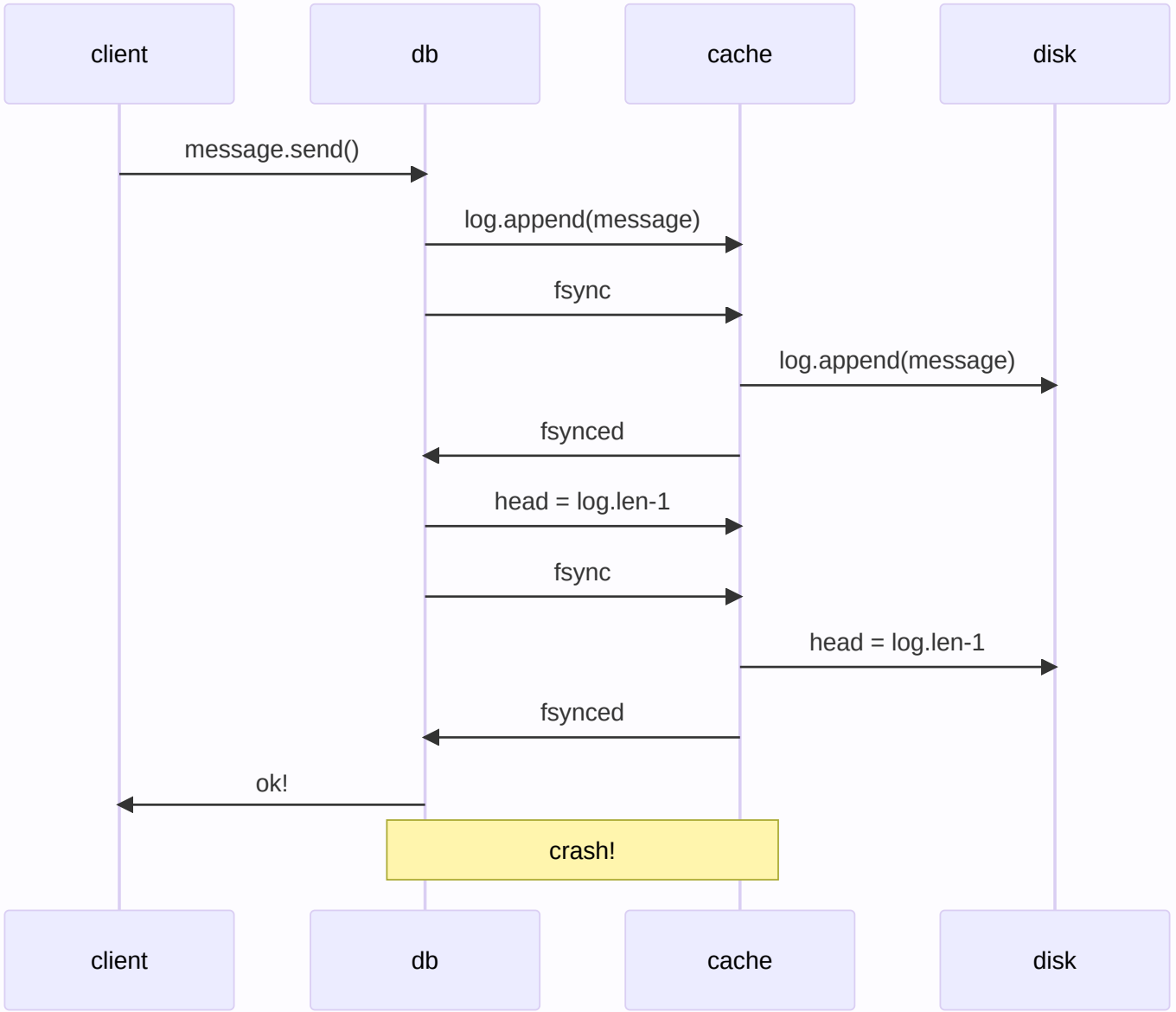

client
db
cache
disk
message.send()
log.append(message)
log.append(message)
head = log.len-1
head = log.len-1
ok!
client
db
cache
disk


client
db
cache
disk
message.send()
log.append(message)
head = log.len-1
head = log.len-1
ok!
crash!
client
db
cache
disk

fsync


client
db
cache
disk
message.send()
log.append(message)
fsync
log.append(message)
fsynced
head = log.len-1
head = log.len-1
ok!
crash!
client
db
cache
disk


client
db
cache
disk
message.send()
log.append(message)
fsync
log.append(message)
fsynced
head = log.len-1
ok!
crash!
client
db
cache
disk


client
db
cache
disk
message.send()
log.append(message)
fsync
log.append(message)
fsynced
head = log.len-1
fsync
head = log.len-1
fsynced
ok!
crash!
client
db
cache
disk

fsync every N writes	writes/s	gb/s
1	16_442	0.06
16	55_279	0.21
4096	399_272	1.52
1000000	465_775	1.77

reliability

DRAM Errors in the Wild: A Large-Scale Field Study (2009)

Correctable errors	8.2% of DIMMs per year
Uncorrectable errors	0.22% of DIMMs per year
Undetectable errors	?

An Analysis of Latent Sector Errors in Disk Drives (2007)

An Analysis of Data Corruption in the Storage Stack (2008)

Latent sector error	1.4% of disks per year
Corruption	0.042% of disks per year
Misdirected read/write	?

Caveats...

(ECC) DRAM:

- caching
- prefetching
- pipelining
- speculative execution
- cheap fences
- corrects single bit errors
- detects double bit errors

Disks:

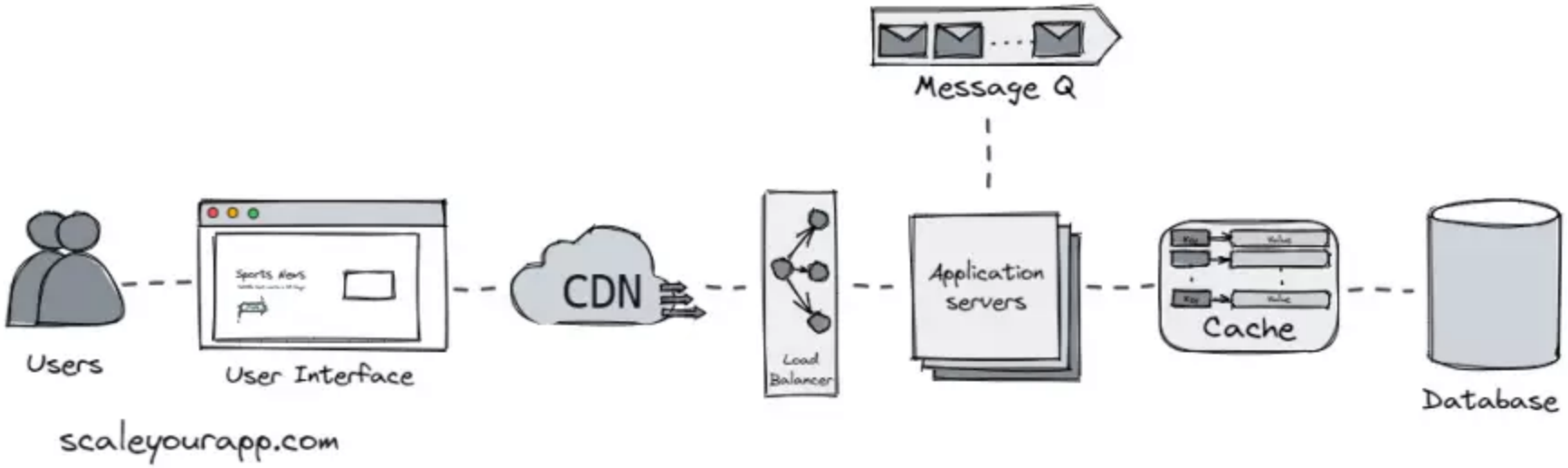
- ~~caching~~
- ~~prefetching~~
- ~~pipelining~~
- ~~speculative execution~~
- expensive fsync
- does not detect corruption
- does not detect misdirected reads/writes

query language

Why do query languages like sql, graphql, cypher etc exist?

Query languages offer:

- Latency hiding (send code to data)
- Concurrency hiding (via query planner)
- Physical/logical data independence



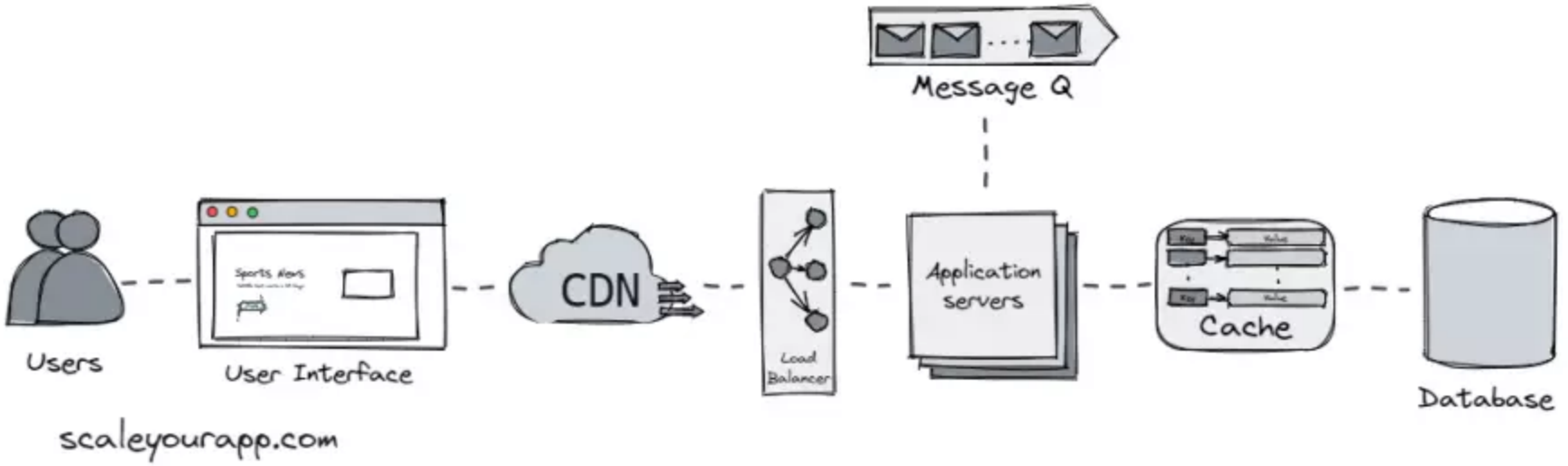
Generic Web Architecture

Amdahl's law:

$$Speedup = \frac{1}{(1-p) + p/N}$$

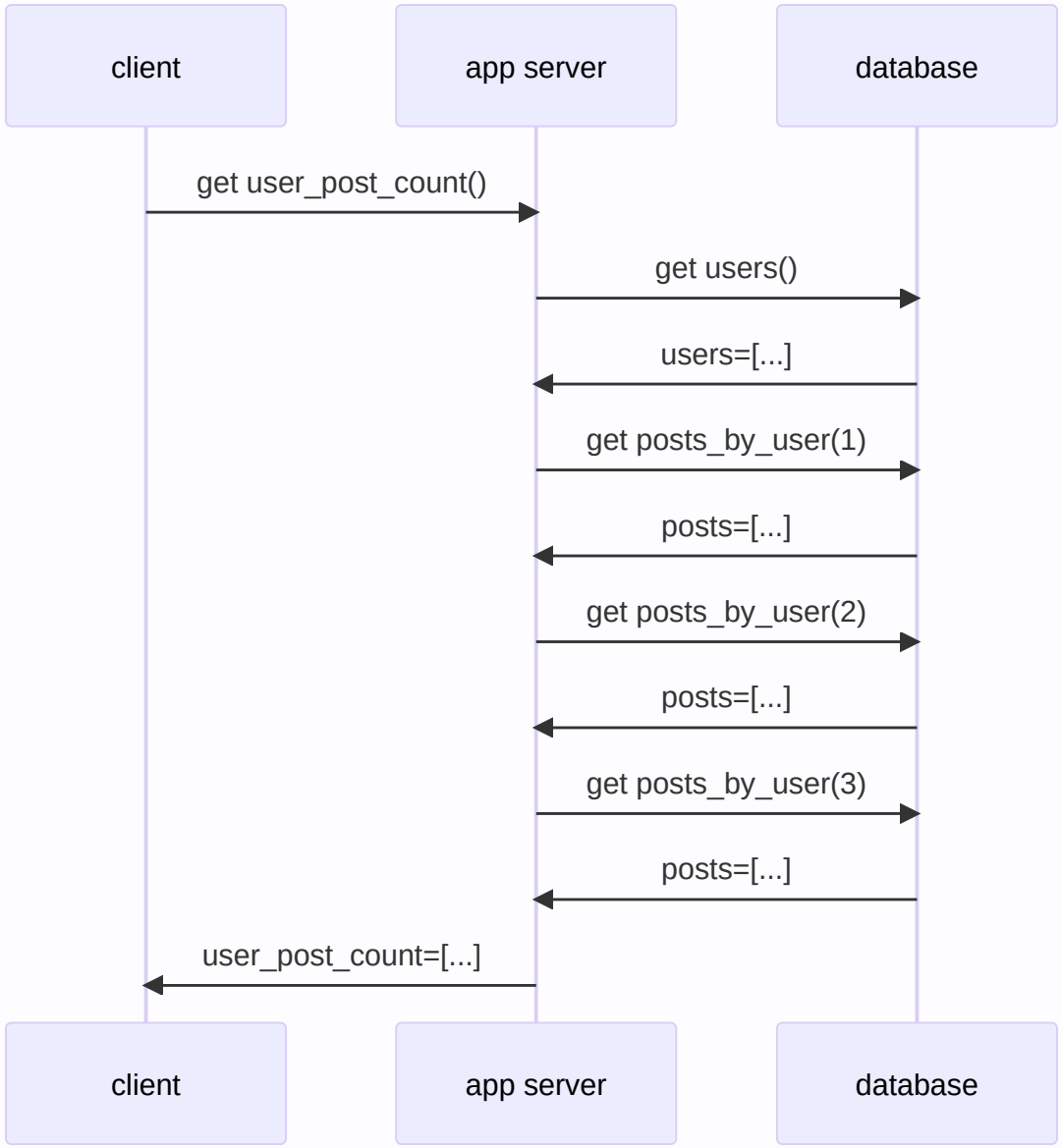
Half-assed Amdahl's law:

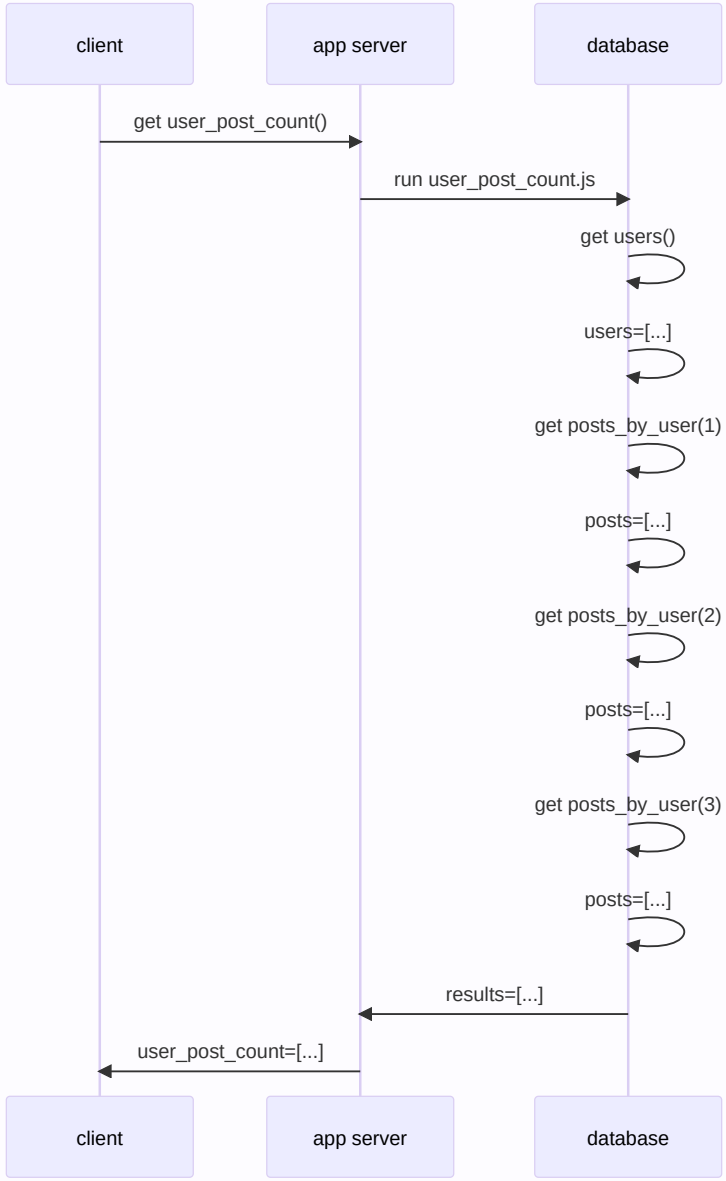
embarassingly parallel	=>	scale it horizontally
not embarassingly parallel	=>	get someone else to do it

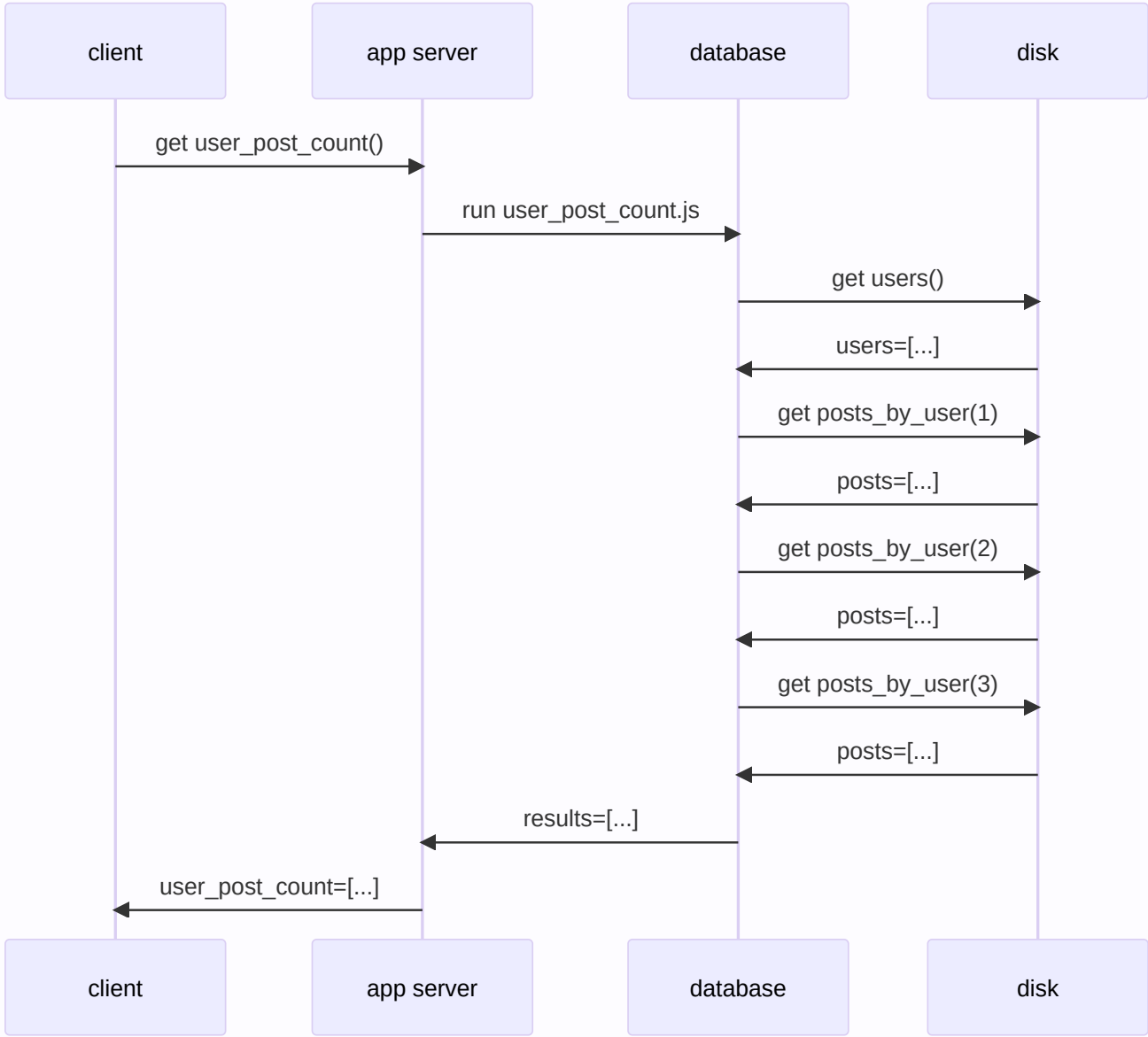


Generic Web Architecture

```
var results = [];
for (var user of db.users) {
    let posts = db.posts_by_user.lookup(user.id);
    results.append([user.id, posts.len]);
}
return results;
```


client
app server
database
get user_post_count()
get users()
users=[...]
get posts_by_user(1)
posts=[...]
get posts_by_user(2)
posts=[...]
get posts_by_user(3)
posts=[...]
user_post_count=[...]
client
app server
database

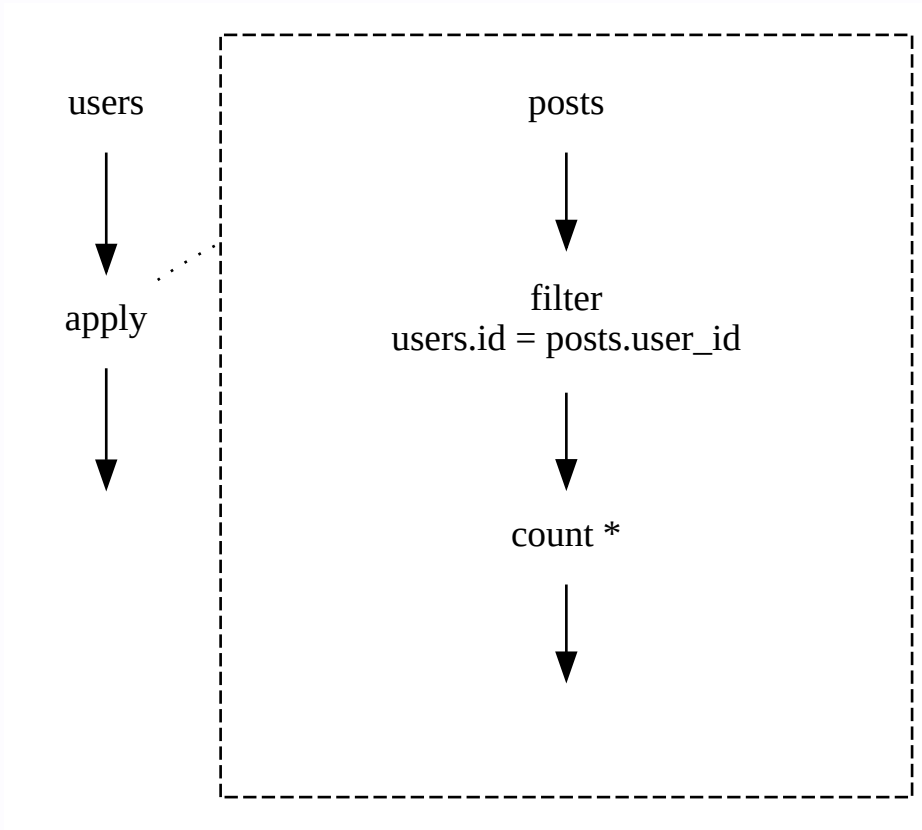

client
app server
database
get user_post_count()
run user_post_count.js
get users()
users=[...]
get posts_by_user(1)
posts=[...]
get posts_by_user(2)
posts=[...]
get posts_by_user(3)
posts=[...]
results=[...]
user_post_count=[...]
client
app server
database

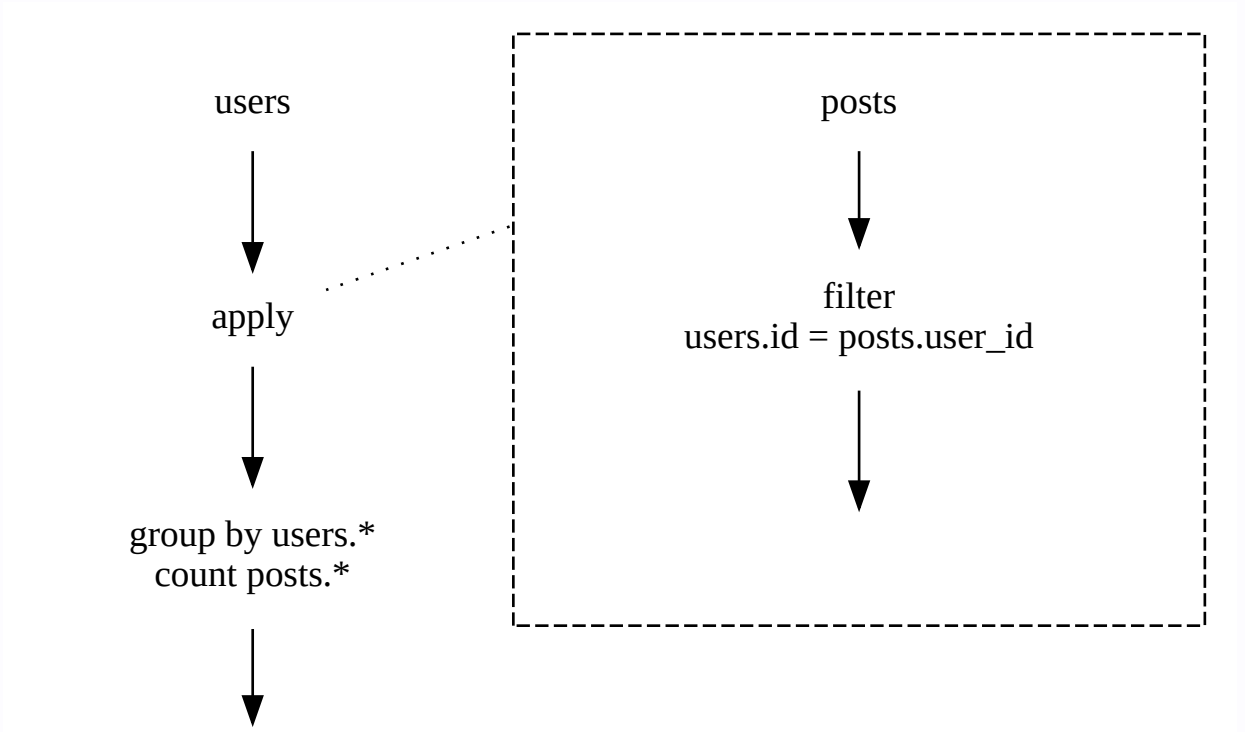

client
app server
database
disk
get user_post_count()
run user_post_count.js
get users()
users=[...]
get posts_by_user(1)
posts=[...]
get posts_by_user(2)
posts=[...]
get posts_by_user(3)
posts=[...]
results=[...]
user_post_count=[...]
client
app server
database
disk

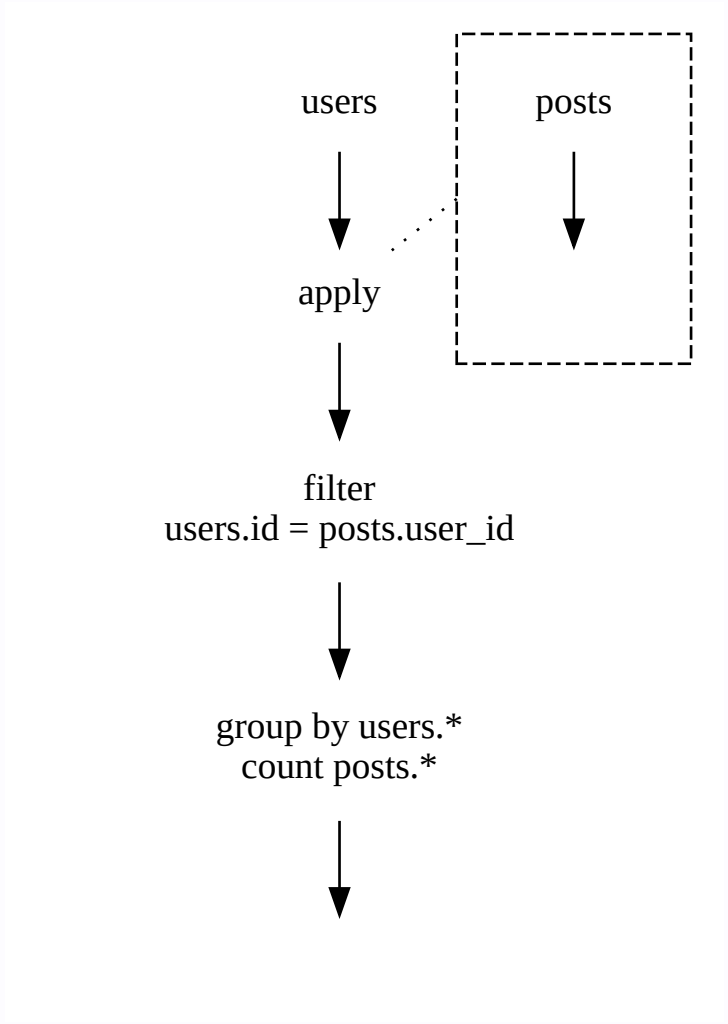
Options:

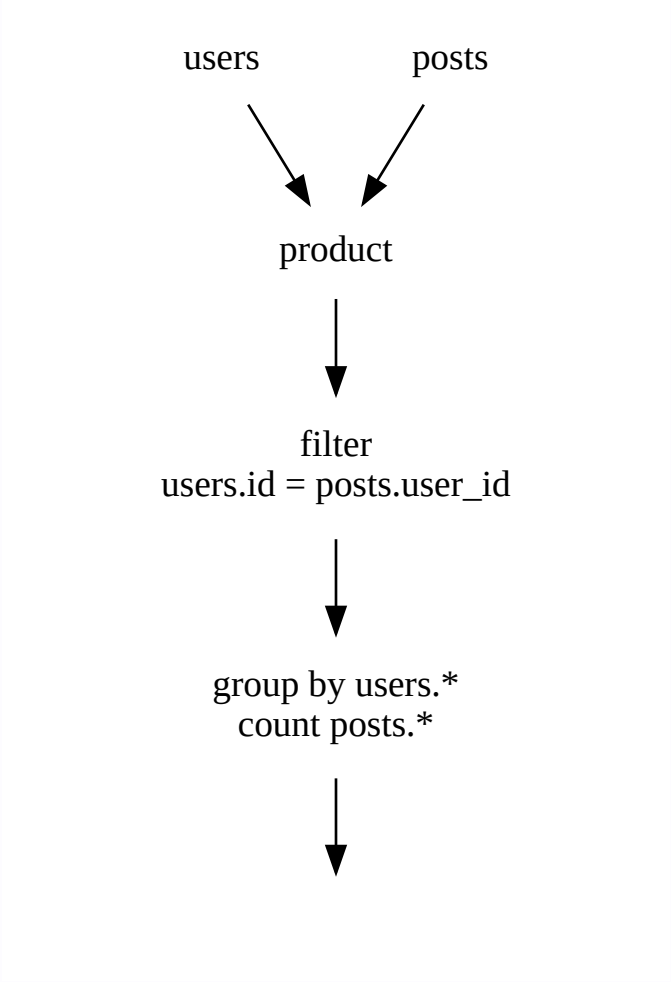
- Run many queries concurrently (=> concurrency control).
- Automatically break query into concurrent sections (=> query planner).

```
select
  users.id,
  (
      select count(*)
      from posts
      where posts.user_id = users.id
  )
from users
```


users
posts
filter
users.id = posts.user_id
apply
count *


users
posts
apply
filter
users.id = posts.user_id
group by users.*
count posts.*


users
posts
apply
filter
users.id = posts.user_id
group by users.*
count posts.*


users
posts
product
filter
users.id = posts.user_id
group by users.*
count posts.*

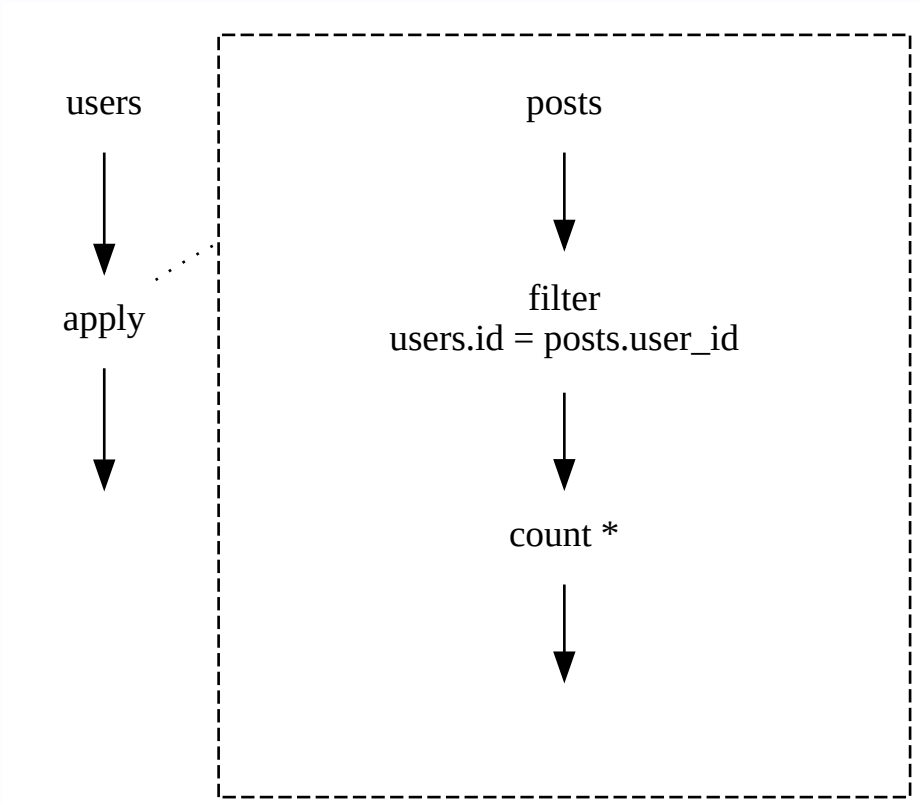
```
fn scan(table: Table, callback: fn ([]Row) void) void {
    for (table.block_addresses()) |address| {
       const block = disk.read(address);
       callback(decode_block(block));
    }
}
```

```
fn scan(table: Table, callback: fn ([]Row) void) void {
    const addresses = table.block_addresses());
    for (addresses) |address, i| {
       if (i < addresses.len) {
           disk.prefetch(addresses[i + prefetch_count]);
       }
       const block = disk.read(address);
       callback(decode_block(block));
    }
}
```

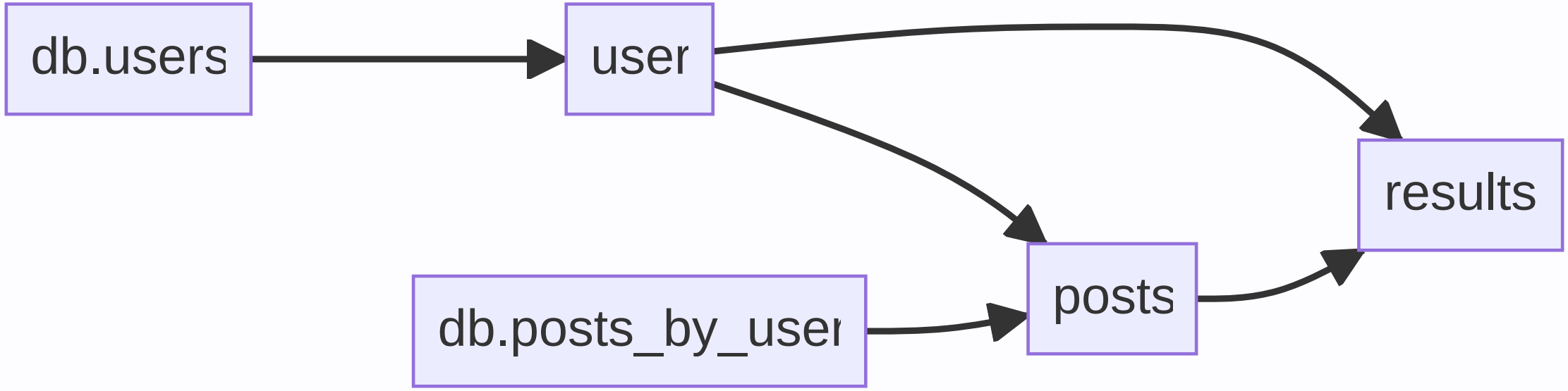
Why can we prefetch with sql and not javascript?

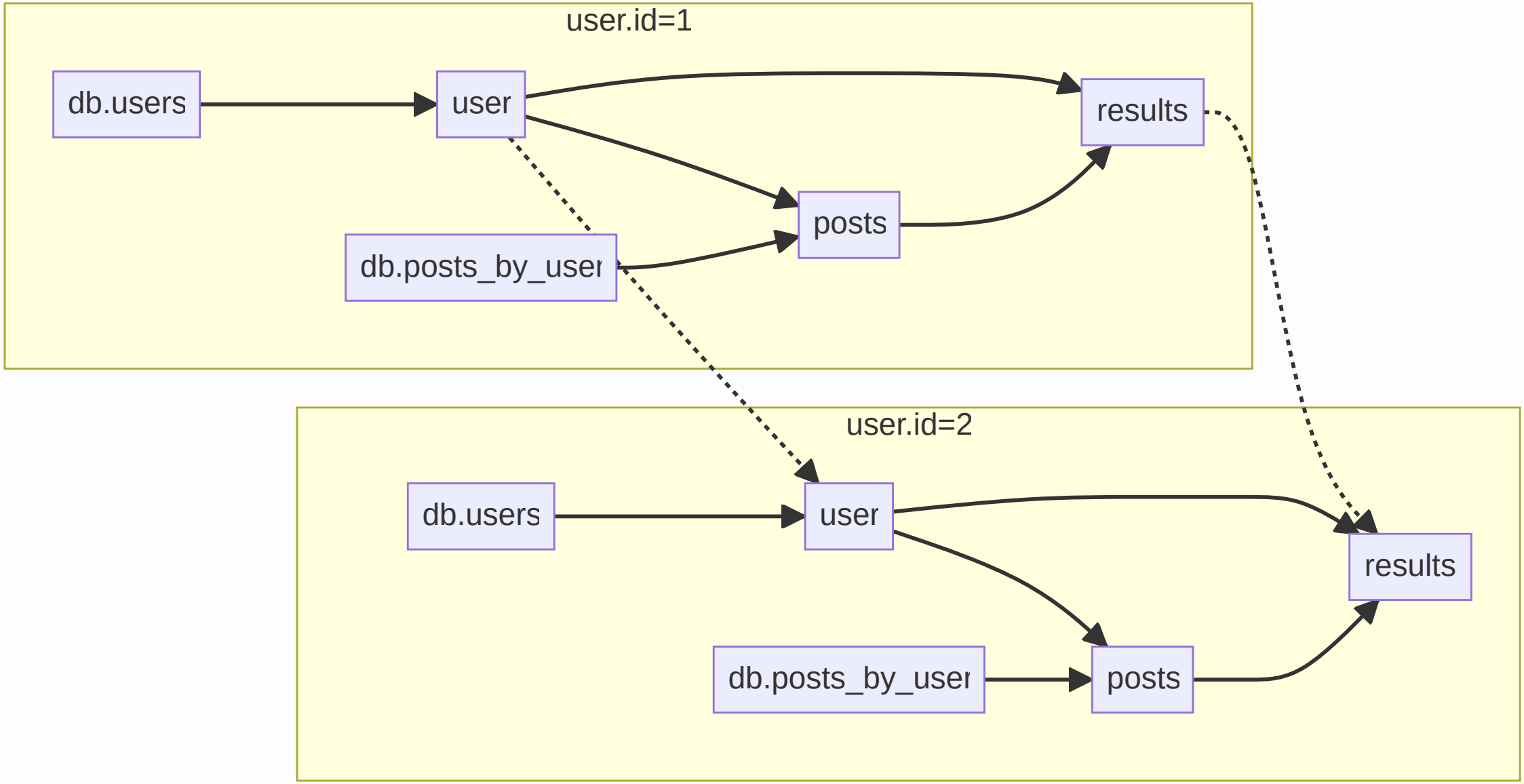
SQL is declarative

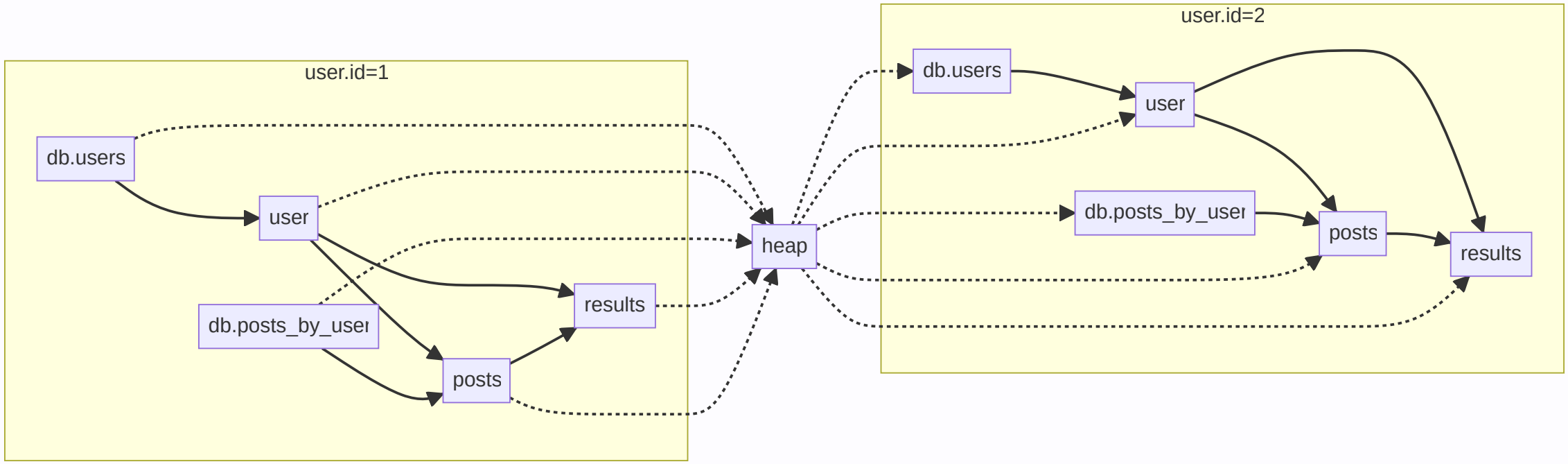
SQL contains magic beans


users
posts
filter
users.id = posts.user_id
apply
count *

```
var results = [];
for (var user of db.users) {
    let posts = db.posts_by_user.lookup(user.id);
    results.append([user.id, posts.len]);
}
return results;
```


db.users
user
results
db.posts_by_user
posts


user.id=1
db.users
user
results
posts
db.posts_by_user
user.id=2
db.users
user
results
db.posts_by_user
posts


user.id=1
db.users
user
db.posts_by_user
posts
results
heap
user.id=2
db.users
user
db.posts_by_user
posts
results

Programming languages:

- pointers to heap
- aliasing
- mutable heap
- heap contains pointers

Query languages:

- ids
- snapshots
- atomic change

Query languages - visible complexity:

- language
- query planner and optimizer
- plan interpreter or compiler
- client-side libraries

Query languages - invisible complexity:

- code duplication
- feral concurrency control

What is a database?

What is a database?

- storage engine
- concurrency control
- query language

Why is a database?

Long-lived data.

FIN

jamie@scattered-thoughts.net