

facebook

facebook

Asynchronous MySQL

How Facebook Queries Databases

Chip Turner – chip@fb.com

2014-04-02

Our Codebase

- A fancy website written in PHP (which became Hack)
 - Grew organically over time
 - Accrued technical debt, then paid it off as we scaled
- Many backend services written in C++
- Operations tools (99% Python, 1% PHP, 1% Perl, 1% ...)

Our Servers and Network

- Hundreds of thousands of servers
- “Many, many” webservers
- “Many” databases
- Sharded data model
- Single master, multiple replicas
 - One copy of each shard in each datacenter
 - Multiple datacenters worldwide

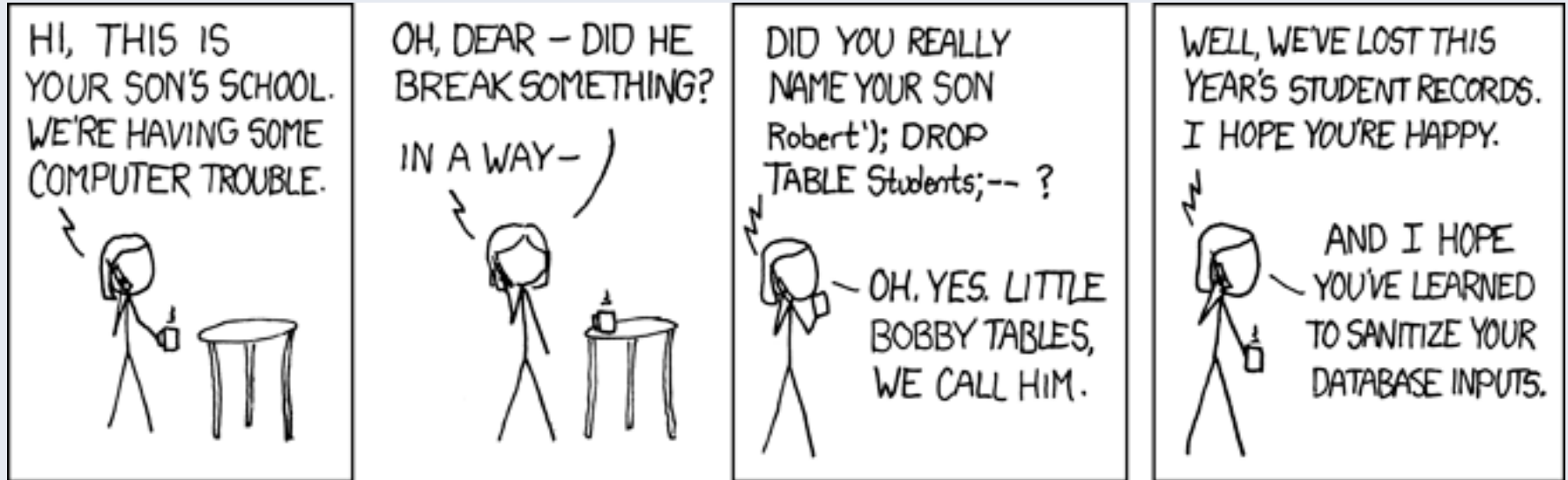
A Sense of Scale

- PHP code issues 10,000 QPS... of errors
 - Connection refused, failovers, solar flares, timeouts, you name it
 - Retries usually make these invisible to the application
- 12,000,000 QPS of actual queries from web servers
 - 8,400,000 QPS of Async MySQL queries (up from 0 one year ago)
- Average query time: 9ms
 - 30 hours of queries executed per second
- This is just PHP – does not include TAO, warehousing, or other use cases

DB Client Team

- Formed a team in early 2013 to focus on database client issues
- Most original database client code came along as necessary, not designed
- Problem space is both querying databases and finding the right database to query; this is surprisingly tricky
- Primarily OLTP workload
- Usability, security, reliability are all goals

Security - <http://xkcd.com/327/>



Security team also focuses on this area; joint responsibility.
They make it secure, we make it easier to use.

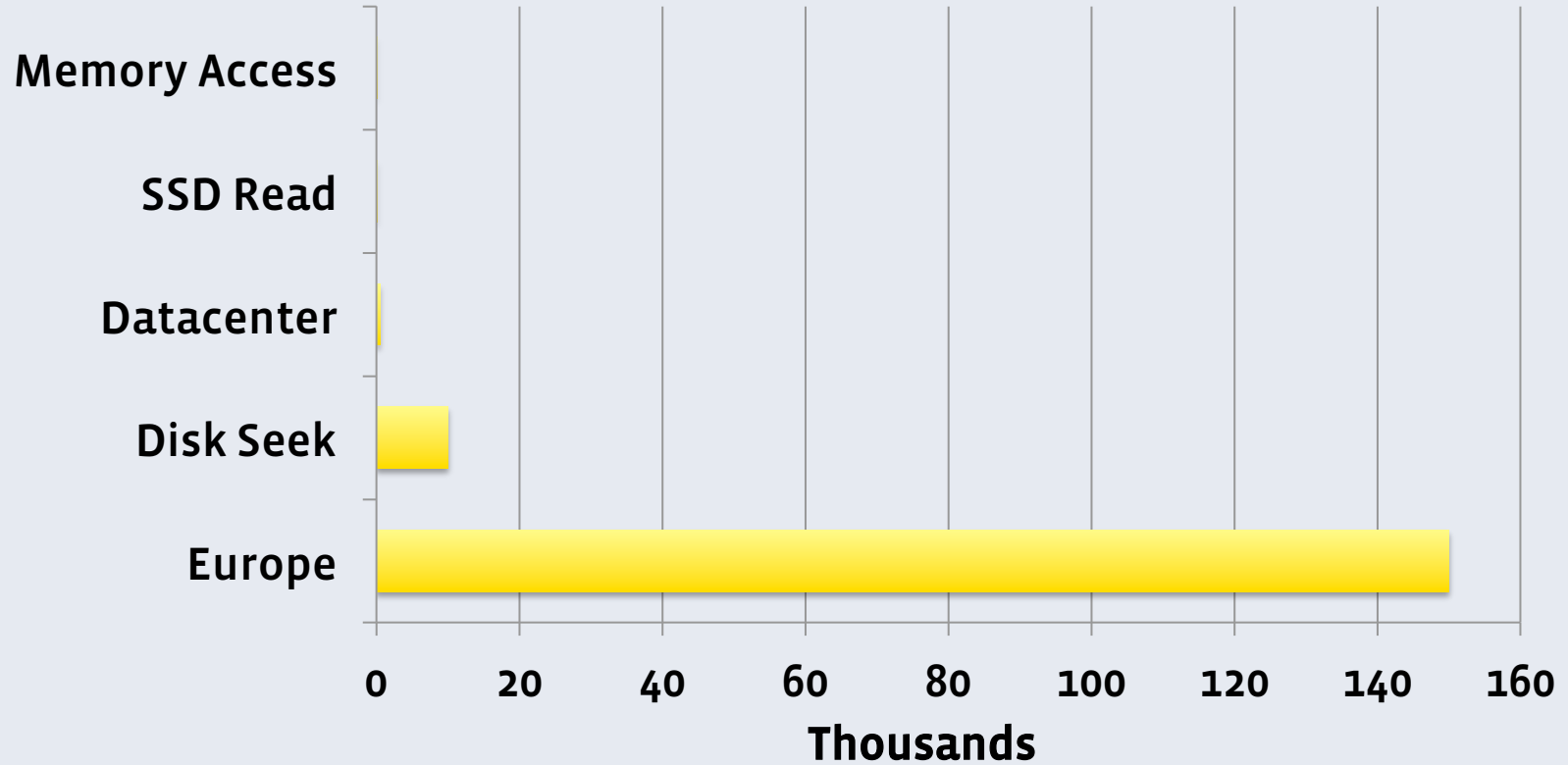
Different Kinds of Performance

- Throughput
 - Queries per second or rows per second
 - Usually more interesting at the database level
- Latency
 - (Milli)seconds per query
 - Usually more interesting at the client level
 - p50/p95/p99 are all important
- OLTP is about both, but client performance is mostly about latency

It's all about the microseconds

- Memory access: 0.1 microseconds
- SSD Read: 150 microseconds
- Network round trip in a datacenter: 500 microseconds
- Disk Seek: 10,000 microseconds (10 milliseconds)
- California to Europe round trip: 150,000 microseconds (150 milliseconds)
- <https://gist.github.com/hellerbarde/2843375> - based on work by Peter Norvig and Jeff Dean

A Picture Helps



facebook.com (web and mobile)

How most websites work?

- Webserver receives a request
- Authenticates user, loads some form of HTML template
- Usually iterates over “blocks” or “sections” to stitch together HTML
 - Iteration is the keyword – iterate means serial
- Some sections need to query databases, query memcache, do RPCs, etc
- Mobile is similar, but less HTML and more APIs

A Typical Facebook Newsfeed

The image shows a screenshot of a Facebook newsfeed interface. The top navigation bar is dark blue with the Facebook logo on the left, a search bar in the center, and the user's name 'Kris Stang' and 'Home' on the right. The main content area is divided into three columns. The left column contains navigation links: 'News Feed', 'Messages', 'Events' (with a count of 2), 'Photos', 'GROUPS' (with a count of 13), 'FRIENDS' (with counts for 'Close Friends' (7), 'Family' (3), 'Black & White' (20+), and 'Los Gatos Area' (20+)), and 'APPS' (with counts for 'App Center' (3) and 'Music' (3)). The middle column shows a post by 'Betsy Case' with the text 'Majestic birds! — at Arizona-Sonora Desert Museum.' and a large image of a blue falcon. Below the image are interaction options: 'Like · Comment · Share · 17 minutes ago'. A notification says 'Lizzy Taylor likes this.' and there is a comment input field. The bottom of the middle column shows a partial post by 'Robin Matthews' who 'shared a link'. The right column contains a 'Create Event' button, a notification for '1 MyCalendar - Birthday request', and a 'People You May Know' section with two suggestions: 'Pamela Adams' (15 mutual friends) and 'John Barry' (7 mutual friends), each with an 'Add Friend' button. At the bottom right, there is a footer with 'Facebook © 2012', 'English (US)', and links for 'Privacy · Terms · Cookies'.

facebook Search for people, places and things Kris Stang Home

Update Status Add Photo / Video

What's on your mind?

FAVORITES

- News Feed
- Messages
- Events 2
- Photos

GROUPS

- Photography... 13
- Create Group...

FRIENDS

- Close Friends 7
- Family 3
- Black & White 20+
- Los Gatos Area 20+

APPS


- App Center 3
- Music 3
- Notes
- Links
- Pokes

INTERESTS

- Subscriptions
- Add Interests...

Betsy Case

Majestic birds! — at Arizona-Sonora Desert Museum.



Like · Comment · Share · 17 minutes ago ·

Lizzy Taylor likes this.

Write a comment...

Robin Matthews shared a link

Create Event

1 MyCalendar - Birthday request

People You May Know See All

- Pamela Adams 15 mutual friends Add Friend
- John Barry 7 mutual friends Add Friend

Facebook © 2012
English (US) · Privacy · Terms · Cookies
More ▾

So, how to parallelize?

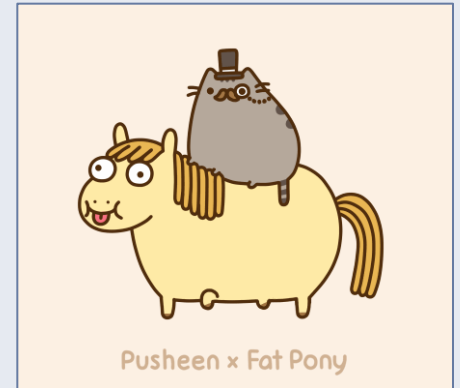
- It's hard!
- Threads are heavy weight, complex, and risk deadlocks and races
- Possibly dozens of databases participate in rendering one page
- Dozens of RPC servers, too
- And hundreds of memcache servers!

In a Perfect World...

- We could convert synchronous to asynchronous with minimal code changes
- Avoid locking, synchronization, and deadlocks
- Keep it light weight
- Keep it readable and maintainable
- While we're making wishes, let's ask for a pony, too

Can we do it? Yes!

- Use generators for simple code that is re-entrant
- Very light weight
 - Less overhead, less complexity vs threads
- Code looks almost the same as synchronous code
- Fit well into our existing PHP codebase
- Later added `async` and `await` keywords to HHVM/Hack



What is a generator?

- A generator is a special kind of function where you “yield” values
- After you yield, other code runs...
- ... but after that code gets its turn, you resume!
- Co-operative multitasking
- State (local variables) stay inside function but execution hops in and out
- Code looks very natural, but amazing things happen
- In HHVM, we have new keywords: `async`, `await`

A PHP Example

```
1 <?php
2
3 function check_host($host) {
4     $conn = mysql_connect($host, ...);
5     $result = mysql_query("SHOW STATUS LIKE 'Threads_connected'", $conn);
6     $row = mysql_fetch_row($result);
7     return $row[0];
8 }
9
10 $total_threads = 0;
11 foreach ($host_list as $host) {
12     $total_threads += check_host($host);
13 }
14
15 print("There are $total_threads threads\n");
```

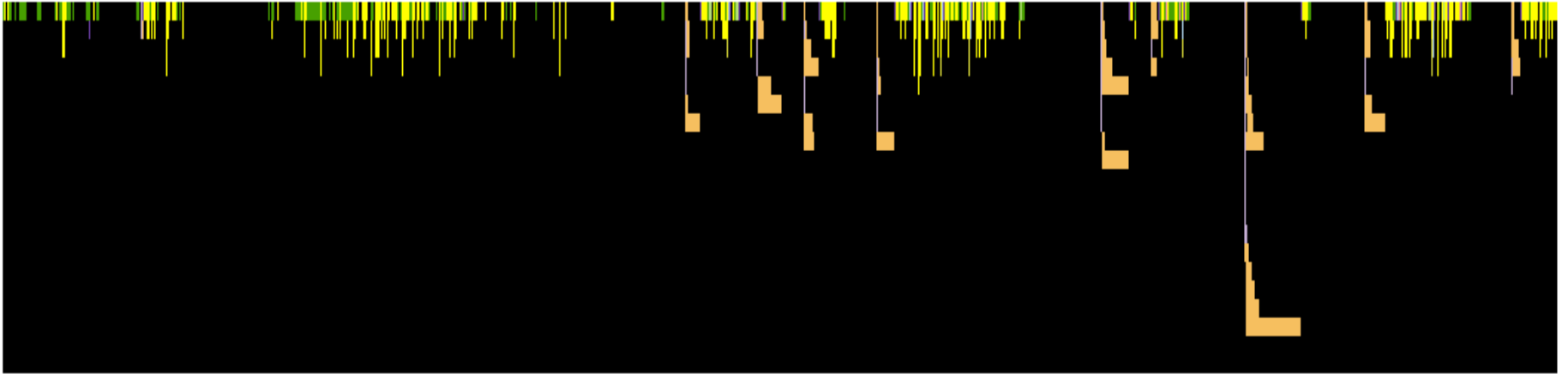
An Asynchronous PHP Example

```
1 <?php
2
3 async function gen_check_host($host) {
4     $conn = await gen_mysql_connect($host, ...);
5     $result = await gen_mysql_query("SHOW STATUS LIKE 'Threads_connected'", $conn);
6     $row = await gen_fetch_row($result);
7     return $row[0];
8 }
9
10 $jobs = array();
11 foreach ($host_list as $host) {
12     $jobs[] = gen_check_host($host);
13 }
14
15 $results = await_all($jobs);
16 $total_threads = 0;
17
18 foreach ($result as $thread_count) {
19     $total_threads += $result;
20 }
21
22 print("There are $total_threads threads\n");
```

That's the key idea

- Cooperative multitasking
- Code running queries (aka business logic) looks familiar
- Complexity is in the server framework, not the code you write
- async functions are cooperative, await lets the server do something else like other queries

A Visualization of a Random Request



- X-axis is time
- Y-axis is depth of concurrency
- Colors are types of operations
- Data dependencies prevent total concurrency

Other languages, use cases

- Python
 - Threads are terrible, so async is a big win
 - Great for operations tooling
 - Uses gevent or another async framework
- C++
 - Basis for many important services; ads, spam detection, search
 - Threads, but larger scale problems; threads+async is nuclear
 - No generators, mainly callbacks (or fibers, but...)

How does it actually work?

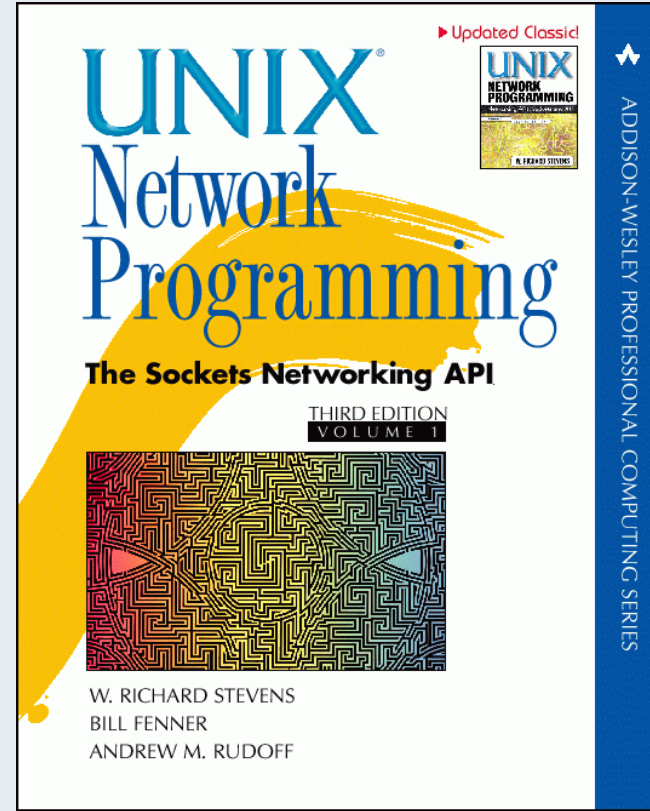
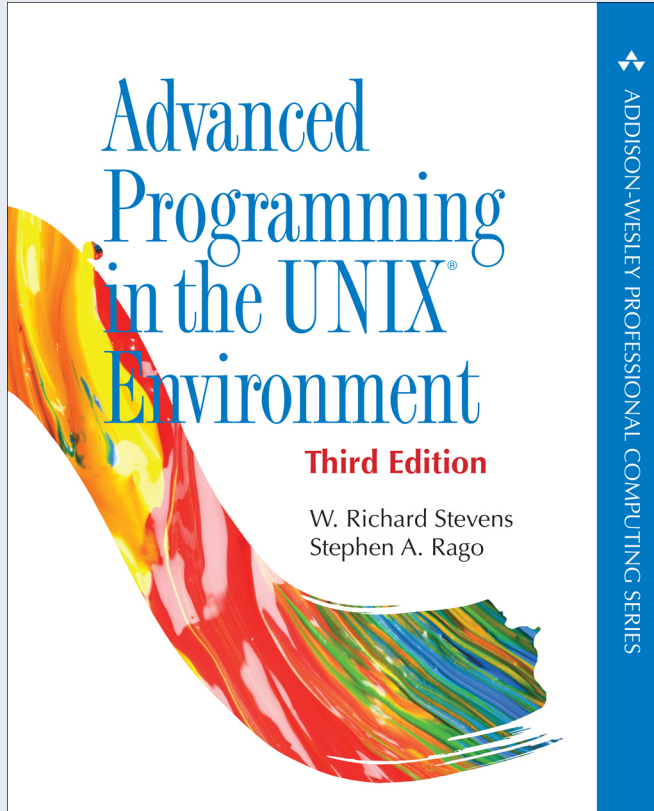
The Magic of Async MySQL

- Extend libmysqlclient to use non-blocking sockets
- API indicates when it is waiting for a read or write
- Expose the file descriptor to feed to UNIX primitives (select/poll/etc)
- Client library has state machines that go through connecting, querying, timeouts, etc
- Same APIs work in async or sync mode; same data structures; very familiar if you've used the library before. Lets you gradually ease into async.

Tricks to help out

- Re-used existing MySQL test suite in async mode
- Partial deployment, able to turn on and off on the fly
- Very tight monitoring; plotting errors as we enabled features
- Able to turn async into sync on a per-callsite basis if something interacts badly
- Initially implemented it “beside” existing MySQL APIs in 5.1; in 5.6, we refactored and cleaned up (Oracle: check out our patches kthx)

It's all in these awesome books



Why you should use async

- You can use webscalesql as your MySQL client library against normal MySQL/MariaDB servers – this is 100% client side. Use it today without touching your servers.
- Using Python+MySQL+Threads? Run, don't walk, to webscalesql.org
- Fairly simple way to parallelize important parts of your site
- Also, it's fun! Querying a thousand databases in <1 second is addictive.

Want to play with it?

- webscalesql.org is the way!
- Diffs pending review for webscalesql. You can grab it here:
 - <https://github.com/chipturner/webscalesql-5.6/tree/webscalesql-5.6.17>
- Python extensions are also available:
 - <https://github.com/chipturner/MySQLdb1/tree/nonblocking>
- Very committed to webscalesql.org; ask questions there, or of me directly:
chip@fb.com

Questions! Answers!

(come visit the Facebook in the Exhibit Hall, get some sweet, sweet swag)

facebook

C example (sorry)

```
MYSQL *STDCALL real_connect_wrapper(MYSQL *mysql, const char *host, ...) {
    int error = 1;
    if (!mysql_real_connect_nonblocking(
        mysql, host, user, passwd, db, port, unix_socket, client_flag)) {
        return (MYSQL *)NULL;
    }
    while (mysql_real_connect_nonblocking_run(mysql, &error) ==
        NET_ASYNC_NOT_READY) {
        int result = socket_event_listen(mysql->net.async_blocking_state,
            mysql_get_file_descriptor(mysql));
    }
    if (error)
        return (MYSQL *)NULL;
    else
        return mysql;
}
```

```
int STDCALL query_wrapper(MYSQL *mysql, const char *query) {
    int error = 1;
    while (mysql_real_query_nonblocking(mysql, query, strlen(query), &error) ==
        NET_ASYNC_NOT_READY) {
        int result = socket_event_listen(mysql->net.async_blocking_state,
            mysql_get_file_descriptor(mysql));
    }
    if (error) {
        return 1;
    }
    return 0;
}
```

```
MYSQL_ROW STDCALL fetch_row_wrapper(MYSQL_RES *res) {
    MYSQL_ROW row;
    MYSQL *mysql = res->handle;
    while (mysql_fetch_row_nonblocking(res, &row) == NET_ASYNC_NOT_READY) {
        int result = socket_event_listen(mysql->net.async_blocking_state,
            mysql_get_file_descriptor(mysql));
    }
    return row;
}
```

Let's see a Python example

```
def check_host(host):
    conn = MySQLdb.connect(host, ...)
    cursor = conn.cursor()
    cursor.execute("SHOW STATUS LIKE 'Threads_connected'")
    result = cursor.fetchall()
    return result[0][1]

total_threads = 0
for host in open("/etc/hosts.txt"):
    total_threads += check_host(host)

print("Total threads: %d" % total_threads)
```


Other options

- Use C fibers inside libmysqlclient itself, means state stays on the stack
 - Fibers are a library, not a language feature. Not very compatible, doesn't scale as well since most codebases aren't designed this way from the beginning
- Use native Python/PHP client libraries that we could do nonblocking with directly
 - Even less compatible, each solution was one-off, and diverged from mainline client library
- Overall, happy with how it's worked out and would do it the same way again. It was more work, but we think it was the most robust option.