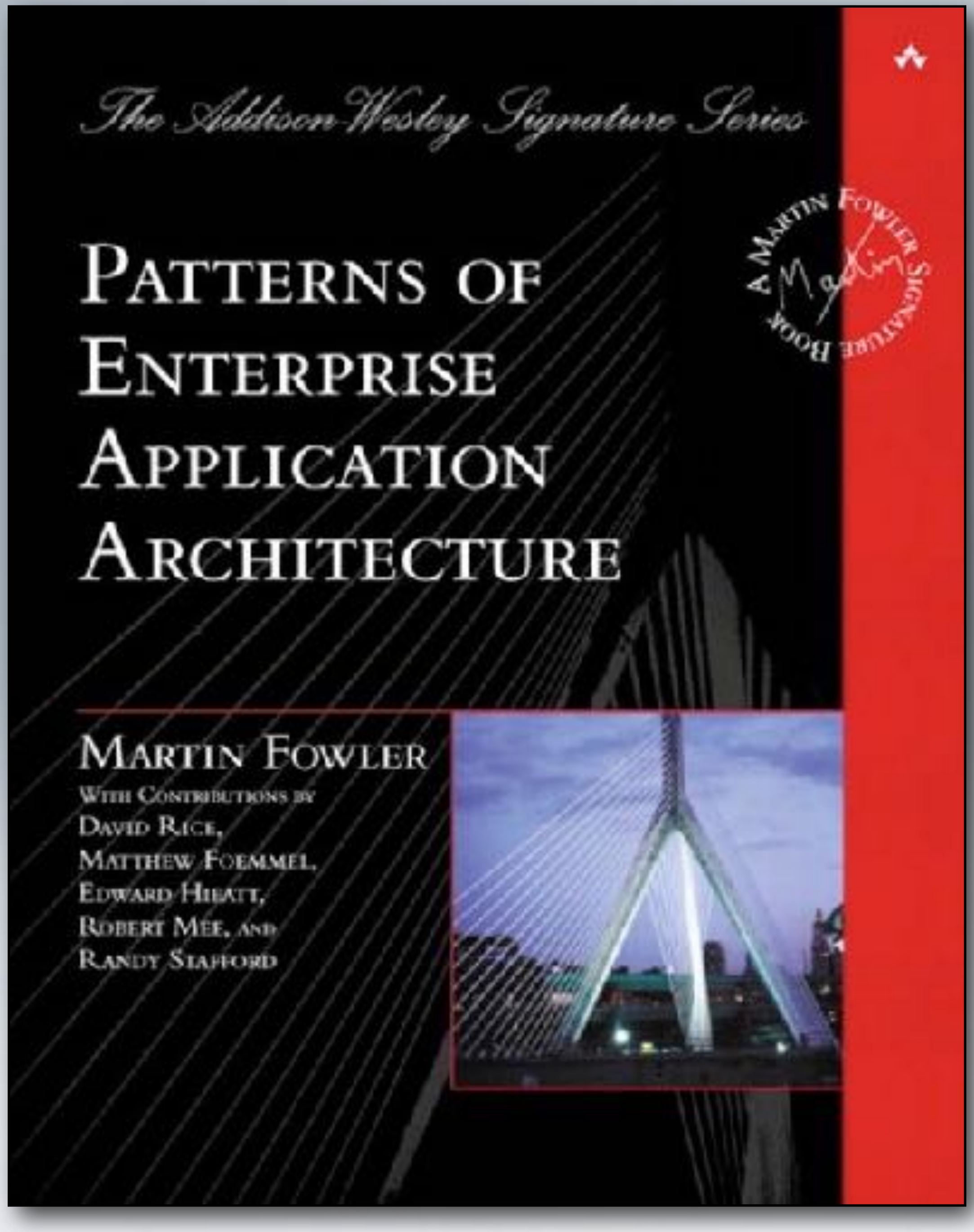
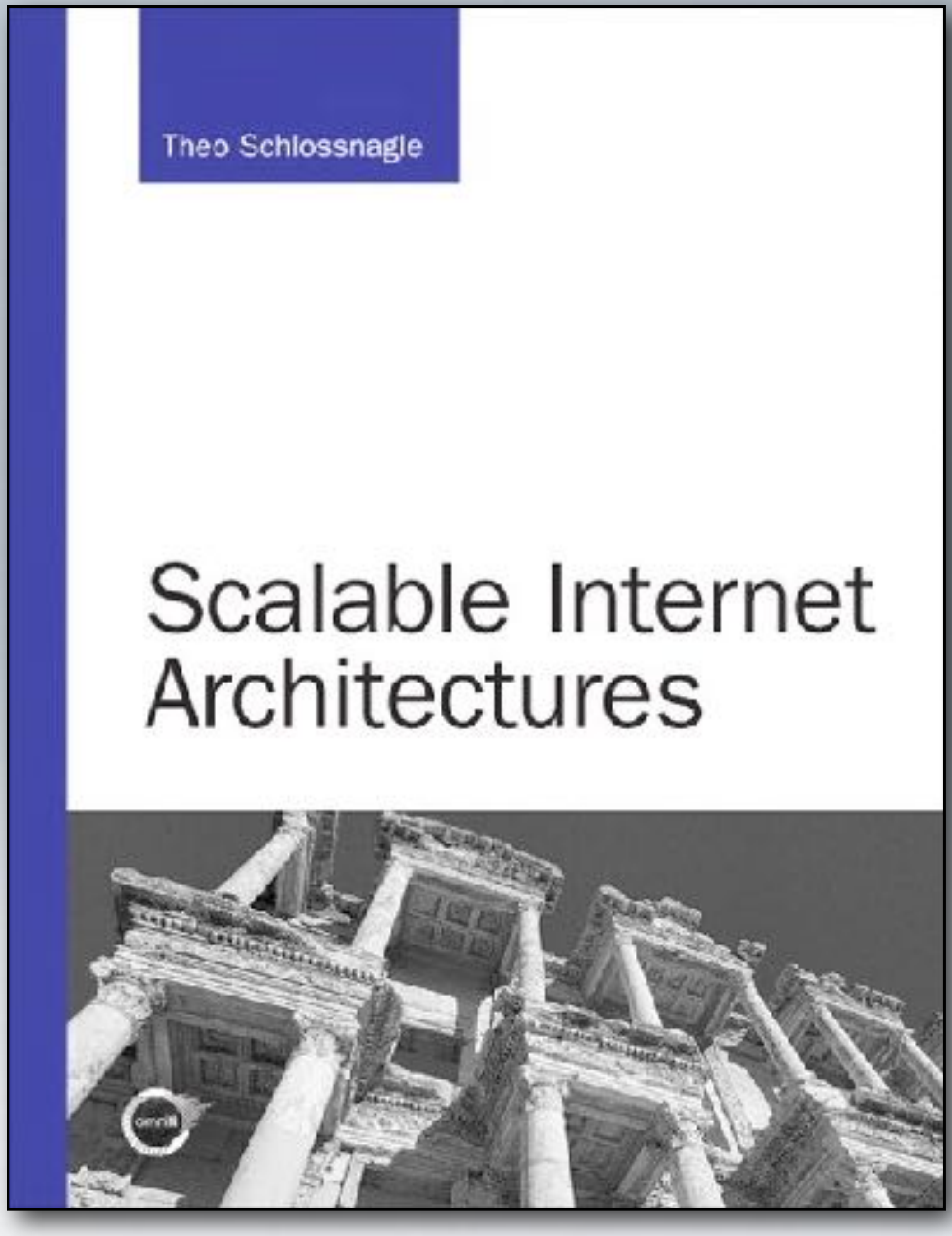
Solving the N+1 Problem:
or, "A Stitch In Time Saves Nine"

paul-m-jones.com
@pmjones

https://joind.in/15630

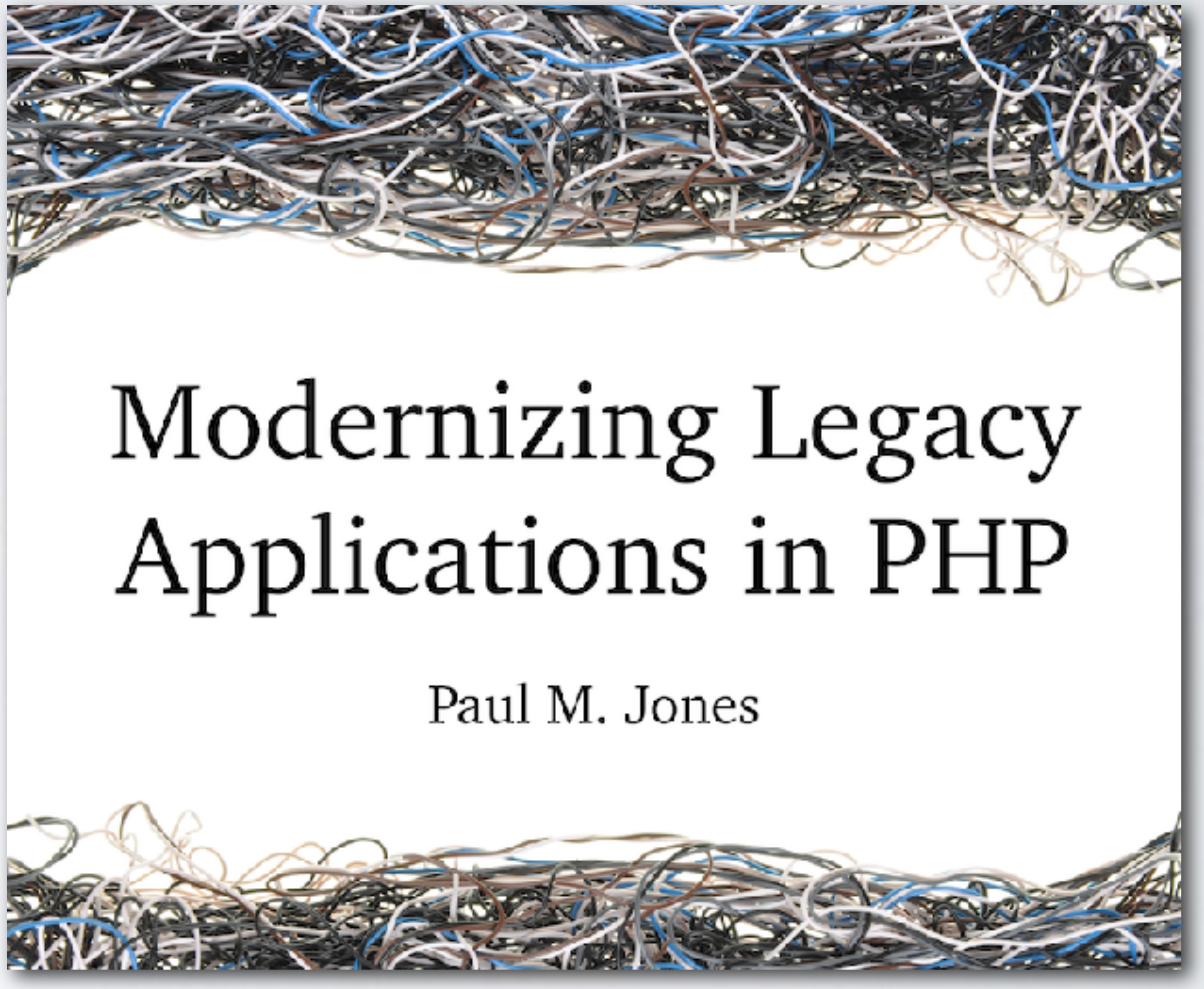
Read These


The Addison-Wesley Signature Series
Patterns of Enterprise Application Architecture
A Martin Fowler Signature Book
Martin Fowler
With Contributions by
David Rice,
Matthew Foemmel,
Edward Hieatt,
Robert Mee, and
Randy Stafford


Theo Schlossnagle
Scalable Internet Architectures

About Me

- 8 years USAF Intelligence
- BASIC in 1983, PHP since 1999
- Jr. Developer, VP Engineering
- Aura, Radar, Relay, Arbiter
- PHP-FIG: PSR-1, PSR-2, PSR-4
- mlaphp.com



Overview

- Performance benchmarking
- The N+1 problem
- Native solutions to the N+1 problem
- Libraries to help with the N+1 problem

Performance Benchmarking

Benchmarking Subjects

- CPU
- RAM
- Disk access
- Database access
- Network access
- **Requests/second**

- Programmer productivity
- Time to initial implementation
- Time to add new major feature
- Time to fix bugs

-- numeric measurement --

-- control for variables --

Limitations of Performance

- A man's got to know his limitations
- Hardware, OS, web server, language, framework, app
- Where in the stack to expend effort?

Performance Measures

- Stock install (Amazon EC2 Large, Ubuntu, Apache, PHP, MySQL)
- Static index.html (`Hello World!`)
- Dynamic index.php (`<?php echo 'Hello World!'; ?>`)
- Database connect (`mysql_*` and `PDO` code)
- Database connect, query, and fetch (`mysql_*` and `PDO` code)

Baseline Performance

	relative	average
html	1.2514	2726.35
php	1	2178.63

5 runs of 10 users for 60 seconds, averaged

MySQL Connect

```php
$host   = 'localhost';
$user   = 'root';
$pass   = 'admin';
$dbname = 'bench';
$table  = 'hello';

$conn = mysql_connect($host, $user, $pass);
mysql_select_db($dbname);
echo "Mysql Connect!";
```

PDO Connect

```
$host   = 'localhost';
$user   = 'root';
$pass   = 'admin';
$dbname = 'bench';
$table  = 'hello';

$pdo = new PDO(
    "mysql:host=$host;dbname=$dbname",
    $user,
    $pass
);

echo "PDO Connect!";
```

Connection Performance

MySQL	**relative**	**average**
html	1.2514	2726.35
php	1	2178.63
connect	0.7926	1726.81

PDO	**relative**	**average**
html	1.2514	2726.35
php	1	2178.63
connect	0.8346	1818.3

Database Table

```
CREATE TABLE hello (
    id INT PRIMARY KEY AUTO_INCREMENT,
    ch VARCHAR(1)
);
INSERT INTO hello (ch) VALUES ('H');
INSERT INTO hello (ch) VALUES ('e');
INSERT INTO hello (ch) VALUES ('l');
INSERT INTO hello (ch) VALUES ('l');
INSERT INTO hello (ch) VALUES ('o');
INSERT INTO hello (ch) VALUES (' ');
INSERT INTO hello (ch) VALUES ('W');
INSERT INTO hello (ch) VALUES ('o');
INSERT INTO hello (ch) VALUES ('r');
INSERT INTO hello (ch) VALUES ('l');
INSERT INTO hello (ch) VALUES ('d');
INSERT INTO hello (ch) VALUES ('!');
```

MySQL Query & Fetch

```
$conn = mysql_connect($host, $user, $pass);
mysql_select_db($dbname);

$rows = mysql_query("SELECT * FROM $table ORDER BY id");
while ($row = mysql_fetch_array($rows, MYSQL_ASSOC)) {
    echo $row['ch'];
}
```

PDO Query & Fetch

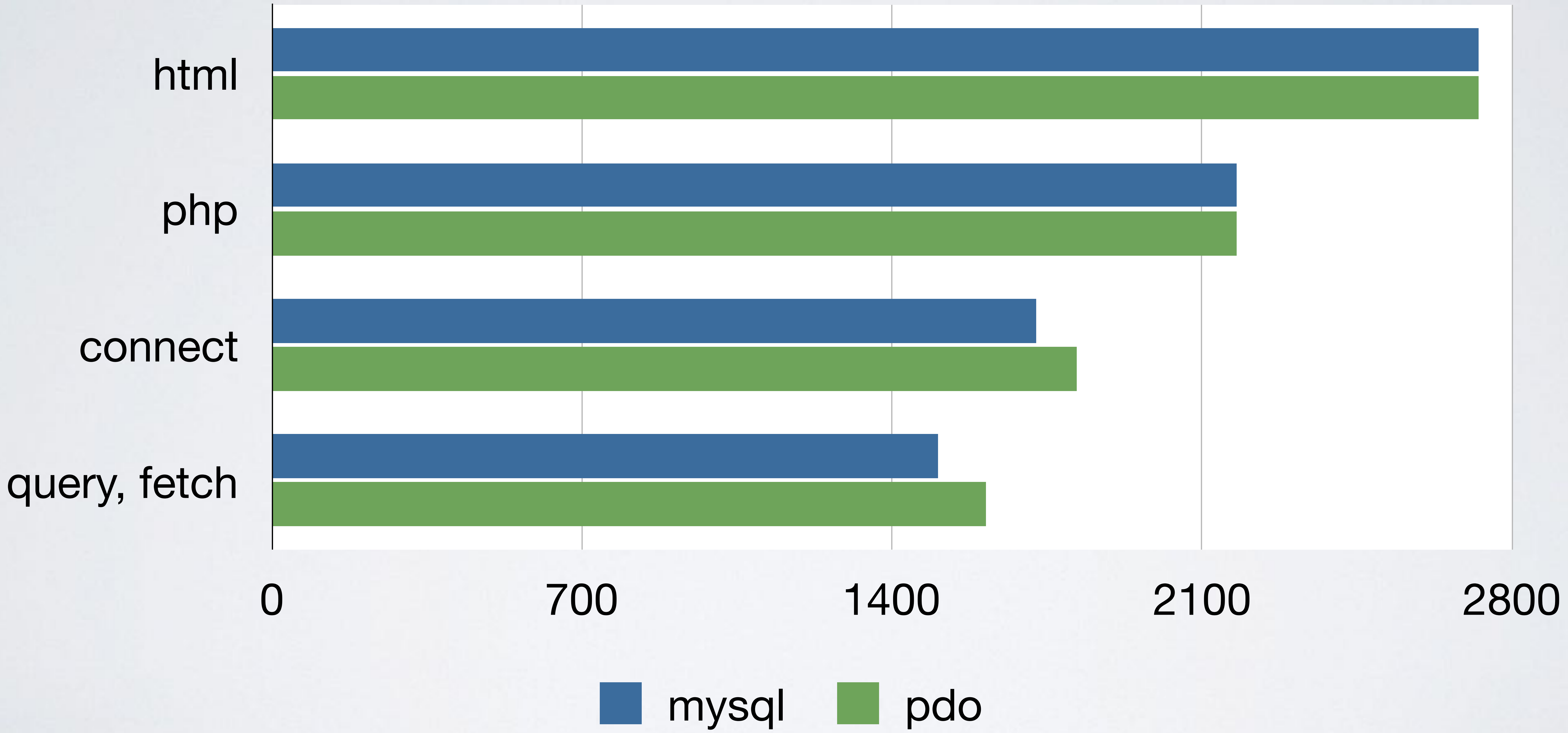
```
$pdo = new PDO(
    "mysql:host=$host;dbname=$dbname",
    $user,
    $pass
);

$stmt = $pdo->prepare("SELECT * FROM $table ORDER BY id");
$stmt->execute();

$rows = $stmt->fetchAll(PDO::FETCH_ASSOC);
foreach ($rows as $row) {
    echo $row['ch'];
}
```

Connect, Query, Fetch Performance

<u>MySQL</u>	**relative**	**average**
html	1.2514	2726.35
php	1	2178.63
connect	0.7926	1726.81
connect, query, fetch	0.6907	1504.76

<u>PDO</u>	**relative**	**average**
html	1.2514	2726.35
php	1	2178.63
connect	0.8346	1818.3
connect, query, fetch	0.7397	1611.61

Overall Performance



The N+1 Problem

Background

- Performance problems in application report
- 2m rows into 40k record objects, 3+ hours
- Reduced dataset to 2000 rows and 40 record objects
- Profiler: 201 queries
- 1 query, plus 5 additional queries per record

N+1 in PHP

```php
// 1 query to get 10 posts
$stmt  = 'SELECT * FROM posts LIMIT 10';
$posts = $sql->fetchAll($stmt);

// 10 queries for comments (1 per post)
$stmt = 'SELECT * FROM comments WHERE post_id = ?';
foreach ($posts as &$post) {
   $bind = array($post['id']);
   $rows = $sql->fetchAll($stmt, $bind);
   $post['comments'] = $rows;
}
```

```php
$posts = array(
    0 => array(
        'id' => '1',
        'body' => 'Post text',
        'comments' => array(
            0 => array(
                'id' => '1',
                'post_id' => '1',
                'body' => 'Comment 1 text'
            ),
            // ...
            9 => array(
                'id' => '9',
                'post_id' => '1',
                'body' => 'Comment 10 text'
            ),
        ),
    ),
    // ...
    9 => array(...),
);
```

Why It’s A Problem

- Each relationship is one extra query per master row
- 5 relationships == 5 queries per master row
- 10 records means 50 added queries
- 40,000 records means 200,000 added queries
- Performance drag. Need to use fewer queries.

Why Does N+1 Happen?

CRUDdy Mindset

- Create, read, update, delete
- Record-oriented focus
- ActiveRecord, RowDataGateway
- Collections are secondary
- In a hurry? Treat collection as a series of single records in a loop

BREAD Instead

- Browse, read, edit, add, delete
- “Browse” is a first-class requirement
- TableModule, TableDataGateway
- Build collections of records right away
- Efficient collection building lends itself to efficient record building

Single-Query Solution

Single Query: Intro

- Select all results, including relationships, in a single query
- Loop through results to marshal into domain objects

Single Query: One-to-One

```php
// one-to-one
$stmt = 'SELECT posts.*, stats.hit_count FROM posts
         LEFT JOIN stats ON stats.post_id = posts.id
         LIMIT 10';

$rows = $sql->fetchAll($stmt);
$posts = array();
foreach ($rows as $post) {
    $post['stats']['hit_count'] = $post['hit_count'];
    unset($post['hit_count']);
    $posts[] = $post;
}
```

Single Query: One-to-Many

```
$stmt = 'SELECT posts.*, comments.* FROM posts
         LEFT JOIN comments ON comments.post_id = posts.id';

$rows = $sql->fetchAll($stmt);

// posts.id posts.author_id posts.title comments.id comments.body
// 1        3               Frist Psot! 1           Initial comment
// 1        3               Frist Psot! 2           Another comment
// 1        3               Frist Psot! 3           Third comment
// 1        3               Frist Psot! 4           Oh come on
// 2        5               Second post 5           1st comment on post 2
// 2        5               Second post 6           2nd comment on post 2
// 2        5               Second post 7           3rd comment on post 2
```

Single Query: One-to-Many

```
$posts = array();
foreach ($rows as $row) {
    $post_id = $row['posts.id'];

    if (! isset($posts[$post_id])) {
        $posts[$post_id] = array(
            'id' => $row['posts.id'],
            'title' => $row['posts.title'],
        );
    }

    $posts[$post_id]['comments'][] = array(
        'id'   => $row['comments.id'],
        'body' => $row['comments.body'],
    );
}
```

Single Query: Review

- Loop through result set to marshal into domain objects
- Fine when you have only "to-one" relationships
- "To-many" relationships introduce complexity (esp. more than one)
  - Result set is larger and more repetitive
  - Less efficient to marshal
  - Difficult to LIMIT/OFFSET

Query-and-Stitch Solution

Query-and-Stitch: Intro

- One query for the master set
- Loop through master set to key on identity field
- One query for related set, against all rows in master set
- Loop through related set and stitch into master set

Query-and-Stitch: Master Set

```
// 1 query to get 10 posts.
$stmt = 'SELECT * FROM posts LIMIT 10';
$rows = $sql->fetchAll($stmt);

// Find the ID of each the post
// and key the $posts array on them.
$posts = array();
foreach ($rows as $post) {
    $id = $post['id'];
    $posts[$id] = $post;
}
```

Query-and-Stitch: Related Set

```
// 1 query to get all comments for all posts at once.
$stmt = 'SELECT * FROM comments
        WHERE post_id IN (:post_ids)';
$bind = array('post_ids' => array_keys($posts));
$rows = $sql->fetchAll($stmt, $bind);

// Stitch into posts.
foreach ($rows as $comment) {
    $id = $comment['post_id'];
    $posts[$id]['comments'][] = $comment;
}
```

Query-and-Stitch: Review

- One added loop (stitching into master set) but 9 fewer queries
- Best for “to-many” relationships but works for “to-one” as well
- Easy to do LIMIT/OFFSET
- Easy to add multiple related sets
  - One query to get results
  - One loop to stitch into master set

Query-and-Stitch: Performance

- 40k records from 2m rows (5 relationships)
- From 200,001 queries to 6 (1 master, 5 related)
- From 3+ hours to ~5 minutes

Automating Query-and-Stitch

ORM

- Query-and-stitch is used by many (most? all?) ORMs for eager-fetch
- ORMs are disliked by a non-trivial set of developers
  - Overhead of including and learning the ORM system
  - Non- or pseudo-SQL query construction, hard to hand-tune
  - Opaque behavior, ineffective/unpredictable in edge cases, resource hog
  - Lazy loading of individual results will reintroduce N+1

Aura.Marshal: Intro

- The problem is not SQL
- The problem is marshaling result sets into domain objects
- Aura.Marshal handles only marshaling, not queries
  - Specify types and relationship fields
  - Load types with results from your own queries
  - Wires up the results into domain objects on fetch

Aura.Marshal: Types

```php
$manager->setType('posts', array(
    'identity_field' => 'id',
    'relation_names' => array(
        'comments'    => array(
            'relationship'  => 'has_many',
            'native_field'  => 'id',
            'foreign_field' => 'post_id'
        ),
    ),
));

$manager->setType('comments', array(
    'identity_field' => 'id',
    'relation_names' => array(
        'post'        => array(
            'foreign_type'  => 'posts',
            'relationship'  => 'belongs_to',
            'native_field'  => 'post_id',
            'foreign_field' => 'id'
        ),
    ),
));
```

Aura.Marshal: Loading

```php
// load posts and get back IDs
$stmt = 'SELECT * FROM posts LIMIT 10';
$result = $sql->fetchAll($stmt);
$post_ids = $manager->posts->load($result);

// load comments for posts
$stmt = 'SELECT * FROM comments
         WHERE post_id IN (:post_ids)';
$bind = array('post_ids' => $post_ids);
$result = $sql->fetchAll($stmt, $bind);
$manager->comments->load($result);
```

Aura.Marshal: Retrieval

```php
foreach ($manager->posts as $post) {
    echo 'Post titled ' . $post->title
        . 'has ' . count($post->comments)
        . '.' . PHP_EOL;
}
```

Conclusion

Conclusion

- Performance benchmarking
- Example of N+1 in PHP
- Mindset: CRUD vs BREAD
- Solutions: single query, query-and-stitch
- Aura.Marshal package as one way of automating

leanpub.com/sn1php

paul-m-jones.com
@pmjones

https://joind.in/15630

Thanks!