CACHING STRATEGIES

BEN RAMSEY

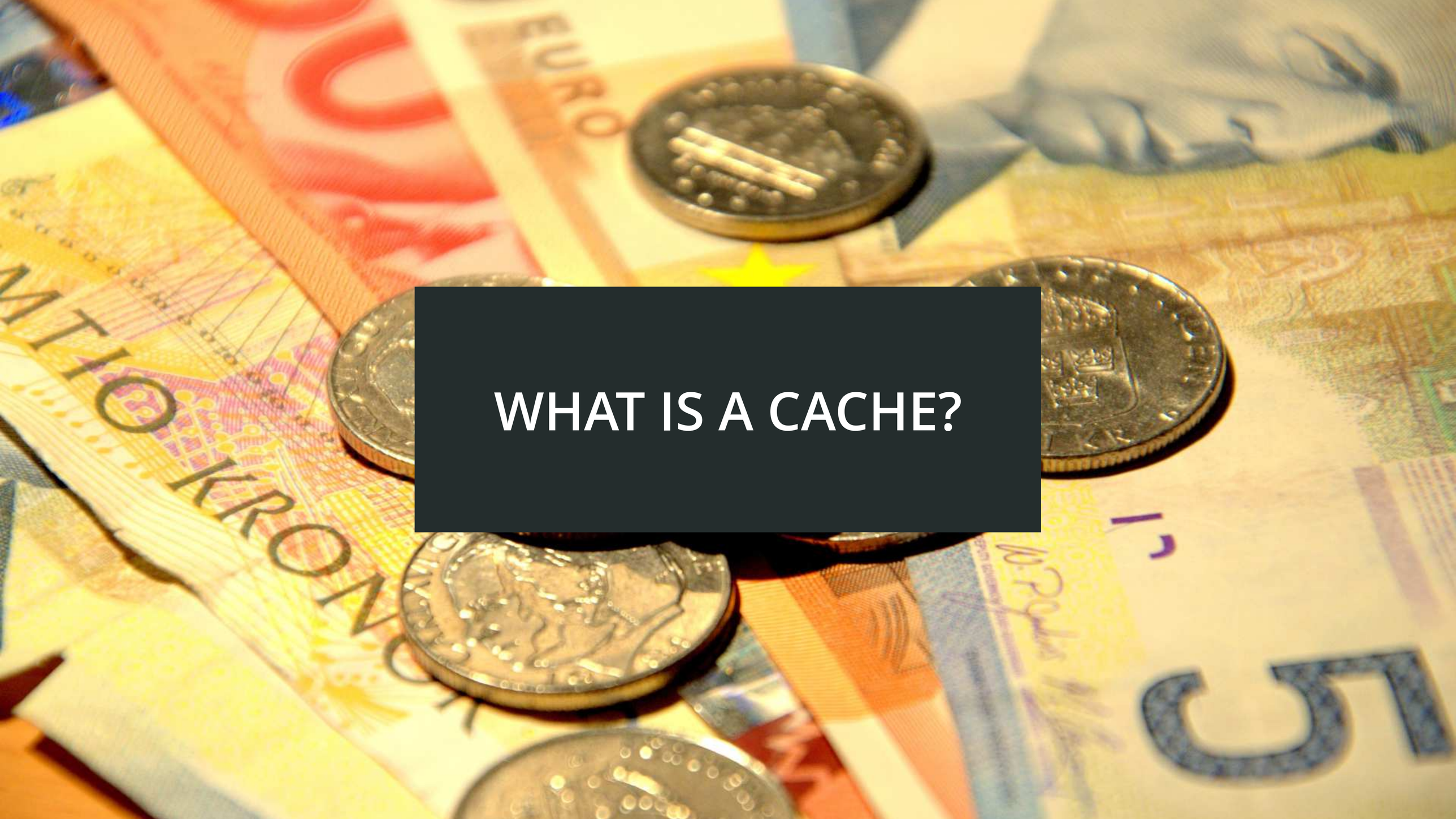
HI, I'M BEN.

I'm a web craftsman, author, and speaker. I build a platform for professional photographers at ShootProof. I enjoy APIs, open source software, organizing user groups, good beer, and spending time with my family. Nashville, TN is my home.

- Books
  - *php|architect's Zend PHP 5 Certification Study Guide*
  - *PHP5 Unleashed*
- Nashville PHP & Atlanta PHP
- array_column()
- Rhumsaa\Uuid library
- virtPHP
- PHP League OAuth 2.0 Client
- Nashville Code User Group Leadership

ShootProof { }

virtPHP


WHAT IS A CACHE?

A CACHE IS...

A store of things that may be required in the future, which can be retrieved rapidly, protected, or hidden in some way.

- Animals store food in caches
- Journalists call a stockpile of hidden weapons a “weapons cache”
- Buried treasure is a cache
- Geocachers hunt for caches
- Computers and applications store data in caches

A CACHE IS...

A store of things that may be required in the future, which can be retrieved rapidly, protected, or hidden in some way.

IN COMPUTING, A CACHE IS...

A fast temporary storage where recently or frequently used information is stored to avoid having to reload it from a slower storage medium.

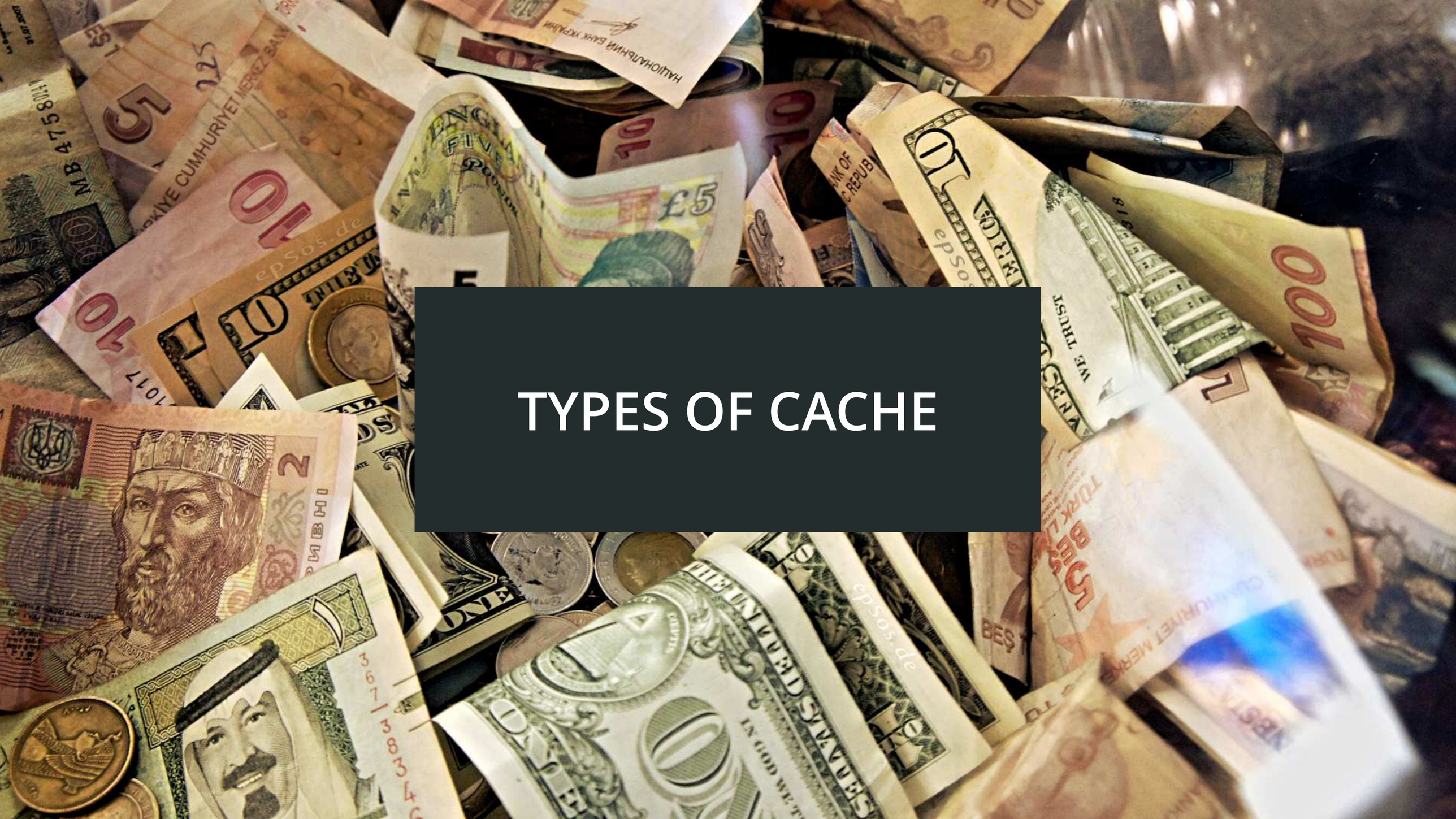
- Reduce the number of queries made to a database
- Reduce the number of requests made to services
- Reduce the time spent computing data
- Reduce filesystem access
- What else?

IN COMPUTING, A CACHE IS...

A fast temporary storage where recently or frequently used information is stored to avoid having to reload it from a slower storage medium.

OUR FOCUS...

Caching from the perspective of a web application.



TYPES OF CACHE

- File system
- Object cache
- Shared memory
- Database
- Opcode cache
- Web cache

FILESYSTEM CACHE

Perhaps the simplest way to cache web application data: store the generated data in local files.

CACHE HTML PAGES

Generate some HTML content, store it to a local file.

```
$html = '';

// Lots of code to build the HTML
// string or page.

file_put_contents('cache.html', $html);
```

CACHE HTML PAGES

Retrieve the pre-generated contents, if available.

```
$html = file_get_contents('cache.html')

if ($html === false) {
    $html = '';
    // Generate your HTML content
    file_put_contents('cache.html', $html);
}

echo $html;
```

CACHE DATA STRUCTURES

Store populated data structures on the local filesystem.

```
// Store a configuration array
// or large recordset of static data

if (file_exists('cache.php')) {
    include 'cache.php';
}

if (!isset($largeArray)) {

    $largeArray = fooBuildData();

    $cache = "<?php\n\n";
    $cache .= '$largeArray = ';
    $cache .= var_export($largeArray, true);
    $cache .= ";\n";

    file_put_contents('cache.php', $cache);

}
```

CACHE.PHP

The created cache.php file now contains something that looks like this:

```
<?php

$largeArray = array (
  'db_name' => 'foo_database',
  'db_user' => 'my_username',
  'db_password' => 'my_password',
  'db_host' => 'localhost',
  'db_charset' => 'utf8',
);
```

OTHER APPROACHES

There are many other approaches to filesystem caching, but they’re all fundamentally the same.

- Store generated data to a file on disk.
- If available, read from that file on disk, rather than generating the data.
- If not available, generate the data and store it.
- That's how most caching works!

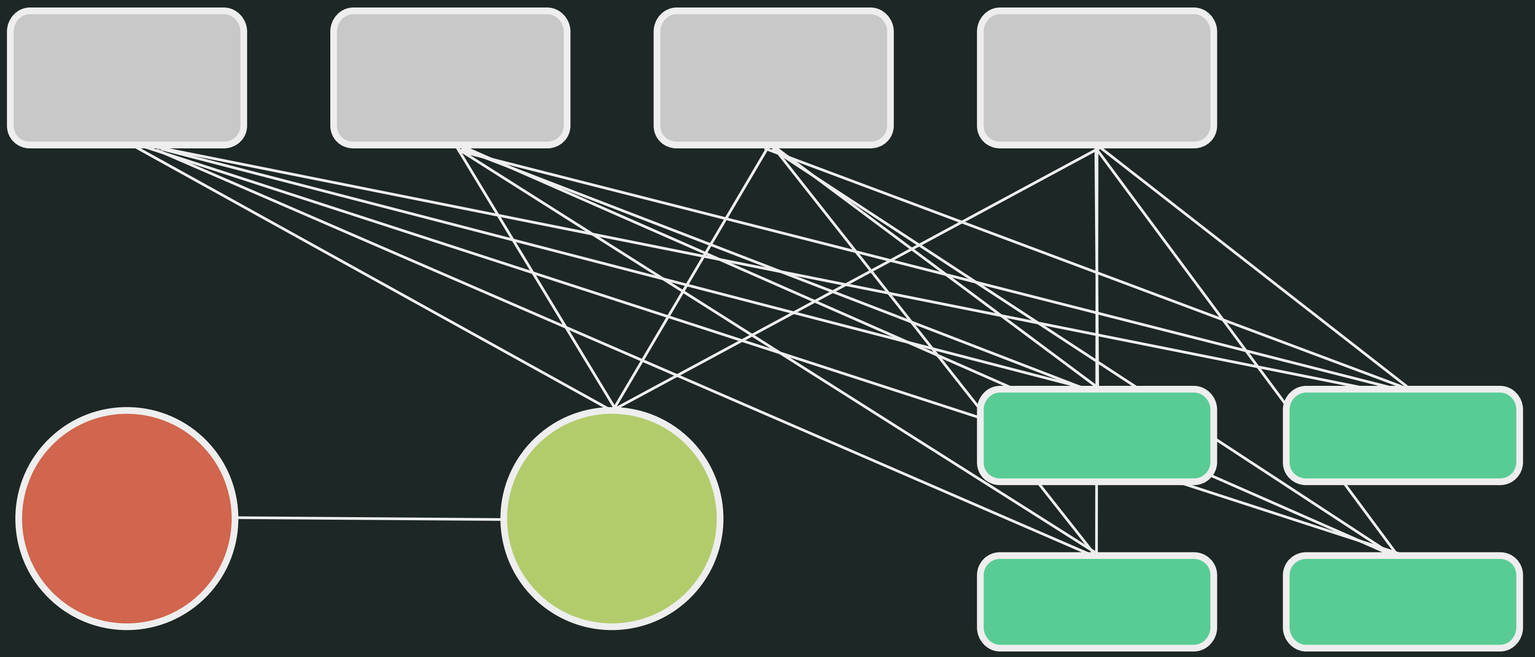
OBJECT CACHE

A variety of key-value arbitrary data stores exist.

MEMCACHED

Memcached is a distributed memory object caching system designed to store small chunks of arbitrary data.

- Simple key/value dictionary
- Runs as a daemon
- Everything is in memory
- Simple protocol for access over TCP and UDP
- Designed to run in a distributed pool of instances
- Instances are not aware of each other; client drivers manage the pool



PECL/MEMCACHED

Pecl/memcached is one of two PHP extensions for communicating with a pool of memcached servers. pecl.php.net/package/memcached

```
$memcache = new Memcached();

$memcache->addServers([
    ['10.35.24.1', '11211'],
    ['10.35.24.2', '11211'],
    ['10.35.24.3', '11211'],
]);
```

GET AND SET WITH PECL/MEMCACHED

Use a key to set and retrieve data from a pool of memcached servers.

```
$book = $memcache->get('9780764596346');

if ($book === false) {
    if ($memcache->getResultCode() == Memcached::RES_NOTFOUND) {
        $book = Book::getByIsbn('9780764596346');
        $memcache->set($book->getIsbn(), $book);
    }
}
```

REDIS

Redis is another type of key-value data store, with some key differences.

- Supports strings and other data types:
  - Lists
  - Sets
  - Sorted sets
  - Hashes
- Persistence
- Replication (master-slave)
- Client-level clustering but built-in clustering in beta

PREDIS

Predis is perhaps the most popular and full-featured PHP client library for Redis. github.com/nrk/predis

```
$redis = new Predis\Client([
    'tcp://10.35.24.1:6379?alias=first-node',
    'tcp://10.35.24.2:6379?alias=second-node',
    'tcp://10.35.24.3:6379?alias=third-node',
]);
```

GET AND SET WITH PREDIS

In it’s simplest form, Predis behaves similar to the memcached client. However, it can perform complex operations, so check the docs.

```
$pageData = $redis->get('homePageData');

if (!$pageData) {
    if (!$redis->exists('homePageData')) {
        $pageData = getHomePageData();
        $redis->set('homePageData', $pageData);
    }
}
```

```php
$redis->hmset('car', [
    'make' => 'Honda',
    'model' => 'Civic',
    'year' => 2008,
    'license number' => 'PHP ROX',
    'years owned' => 1,
]);

echo $redis->hget('car', 'license number');

$redis->hdel('car', 'license number');

$redis->hincrby('car', 'years owned', 1);

$redis->hset('car', 'year', 2010);

var_dump($redis->hgetall('car'));
```

SHARED MEMORY CACHE

Shared memory is often a faster, more efficient alternative to the filesystem for caching local data to be shared by processes.

SHMOP

If your PHP has been built with --enable-shmop, then you may use the shmop_* functions to interact with shared memory.

php.net/shmop

```
$id = shmop_open(123, 'c', 0644, 1000000);

$bytes = shmop_write($id, 'Hello', 0);

$data = shmop_read($id, 0, $bytes);

shmop_close($id);
```

MAINTAINING SHMOP

You will need to keep track of where everything lives in your shared memory block, though.

Shmop is not a key-value store.

```
$data = shmop_read($id, 100, 3);

var_dump($data); // string(3) ""

// "\u0000\u0000\u0000"
```

SYSTEM V SHARED MEMORY

If your PHP has been built with --enable-sysvshm, then you may use the shm_* functions to interact with shared memory.

php.net/sem

```
$config = [1, 2, 3, 4];

$shm = shm_attach(123, 1000000, 0644);

shm_put_var($shm, 42, $config);

$config = shm_get_var($shm, 42);
```

/DEV/SHM

Many Linux systems these days automatically provide RAM disk mounted at /dev/shm. You may write to this in the same way you write to the filesystem, but it's all in memory.

```
$configFile = '/dev/shm/config.php';

if (file_exists($configFile)) {
    include $configFile;
}

if (!isset($config)) {

    $config = getConfiguration();

    $cache = "<?php\n\n";
    $cache .= '$config = ';
    $cache .= var_export($config, true);
    $cache .= ";\n";

    file_put_contents($configFile, $cache);

}
```

DATABASE CACHE

Databases often have their own built-in caching mechanisms, and sometimes it’s useful to generate your own views.

QUERY CACHE

The query cache stores the SELECT statement together with the results. It returns these results for identical queries received later.

```
mysql> SHOW VARIABLES LIKE 'have_query_cache';
+------------------+-------+
| Variable_name    | Value |
+------------------+-------+
| have_query_cache | YES   |
+------------------+-------+
```

MAINTENANCE

You can maintain the MySQL Query Cache with these commands.

```
mysql> FLUSH QUERY CACHE;

mysql> RESET QUERY CACHE;

mysql> SHOW STATUS LIKE 'Qcache%';

mysql> SHOW VARIABLES LIKE 'query_cache_%';
```

CHANGE THE CACHE SIZE

Use Qcache_lowmem_prunes to determine how much memory to allocate to the query cache.

```
mysql> SET GLOBAL query_cache_size = 1000000;
Query OK, 0 rows affected (0.04 sec)

mysql> SHOW VARIABLES LIKE 'query_cache_size';
+------------------+--------+
| Variable_name    | Value  |
+------------------+--------+
| query_cache_size | 999424 |
+------------------+--------+
1 row in set (0.00 sec)
```

SET THE CACHE TYPE

Changing the cache type can affect what happens by default to your SELECT statements.

```
mysql> SET GLOBAL query_cache_type = OFF;

mysql> SET GLOBAL query_cache_type = ON;

mysql> SET GLOBAL query_cache_type = DEMAND;
```

TO CACHE OR NOT TO CACHE

Depending on the cache type you've selected, you may choose to cache or not to cache a specific query.

```
SELECT SQL_CACHE id, name FROM customer;

SELECT SQL_NO_CACHE id, name FROM customer;
```

MATERIALIZED VIEWS

Sometimes queries with expensive joins need to be run beforehand, storing the results for later retrieval.

- Supported natively in Oracle and PostgreSQL
- Standard MySQL views do not solve this problem
- Triggers, stored procedures, and application code may be used to generate materialized views
- Simply a denormalized set of results, useful for fast queries

OPCODE CACHE

An opcode cache is a place to store precompiled script bytecode to eliminate the need to parse scripts on each request.

OPCACHE

The OPcache extension is bundled with PHP 5.5.0 and later. It is also available as an extension for PHP 5.2, 5.3, and 5.4. It is recommended over the older APC extension, which performed a similar function.

php.net/opcache

```
// php.ini configuration

opcache.enable = "1"
opcache.memory_consumption = "64"
opcache.validate_timestamps = "0"
```

OPCACHE FUNCTIONS

OPCache comes with some useful functions that allow you to manage the scripts that have been cached.

```
opcache_compile_file($scriptPath)

opcache_get_configuration()

opcache_get_status()

opcache_invalidate($scriptPath)

opcache_reset()
```

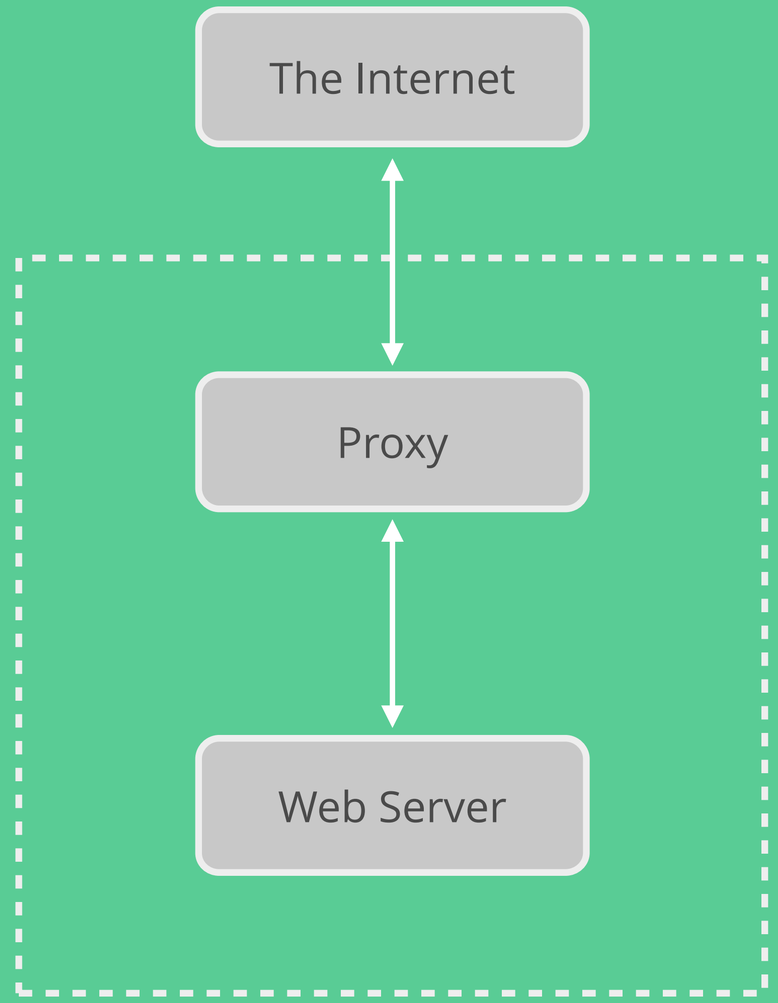
WEB CACHE

A web cache stores whole web objects, such as HTML pages, style sheets, JavaScript, and images.

REVERSE PROXY CACHE

A reverse proxy cache retrieves resources on behalf of a client from one or more servers and caches them at the proxy, usually according to cache control rules and expiration headers.

Sometimes called “web accelerators.”



EXAMPLES

There are many tools to help set up or use reverse proxy caches.

- Varnish Cache
- NGINX Content Caching
- Apache Traffic Server
- Squid
- Various CDNs provide this as part of their services

CONTENT DELIVERY NETWORK (CDN)

A CDN is a large distributed system of servers deployed in multiple data centers across the globe, with the purpose of delivering data from the “edges” to speed up delivery of content to users near those edge locations.

- Akamai Technologies
- Limelight Networks
- Level 3 Communications
- Amazon CloudFront
- Windows Azure CDN
- CloudFlare


CACHE ALL THE
THINGS!

THANK YOU. ANY QUESTIONS?

If you want to talk more, feel free to contact me.

benramsey.com
@ramsey
github.com/ramsey
ben@benramsey.com

This presentation was created using Keynote. The design was inspired by the Catalyst web theme created by Pixelarity. The text is set in Open Sans. The source code is set in Ubuntu Mono. The iconography is provided by Font Awesome.

All photographs are used by permission under a Creative Commons license. Please refer to the Photo Credits slide for more information.

Caching Strategies

Ramsey, Ben. “Caching Strategies.” Nashville PHP User Group. iostudio, Nashville. 13 Jan. 2015. Conference presentation.

PHOTO CREDITS

1. “Lucky Loonie” by Sharon Drummond, CC BY-NC-SA 2.0
2. “Forex Money for Exchange in Currency Bank” by epSos.de, CC BY 2.0
3. “Cash Register” by Steve Snodgrass, CC BY 2.0
4. “Euro Note Currency” by www.TheEnvironmentalBlog.org, CC BY-NC-ND 2.0
5. “Various Currencies” by Bradley Wells, CC BY-NC-SA 2.0