March 29, 2023

A Practical Guide to Backend Caching

Dave Ormsbee, Software Architect
The Center for Reimagining Learning

Agenda

1. Definitions, Motivations, Drawbacks
2. Comparing Options
    a. Python
    b. Django Cache Framework
    c. Open edX Utilities
3. Operational Issues
4. Design Considerations

Definitions:
What is backend caching?

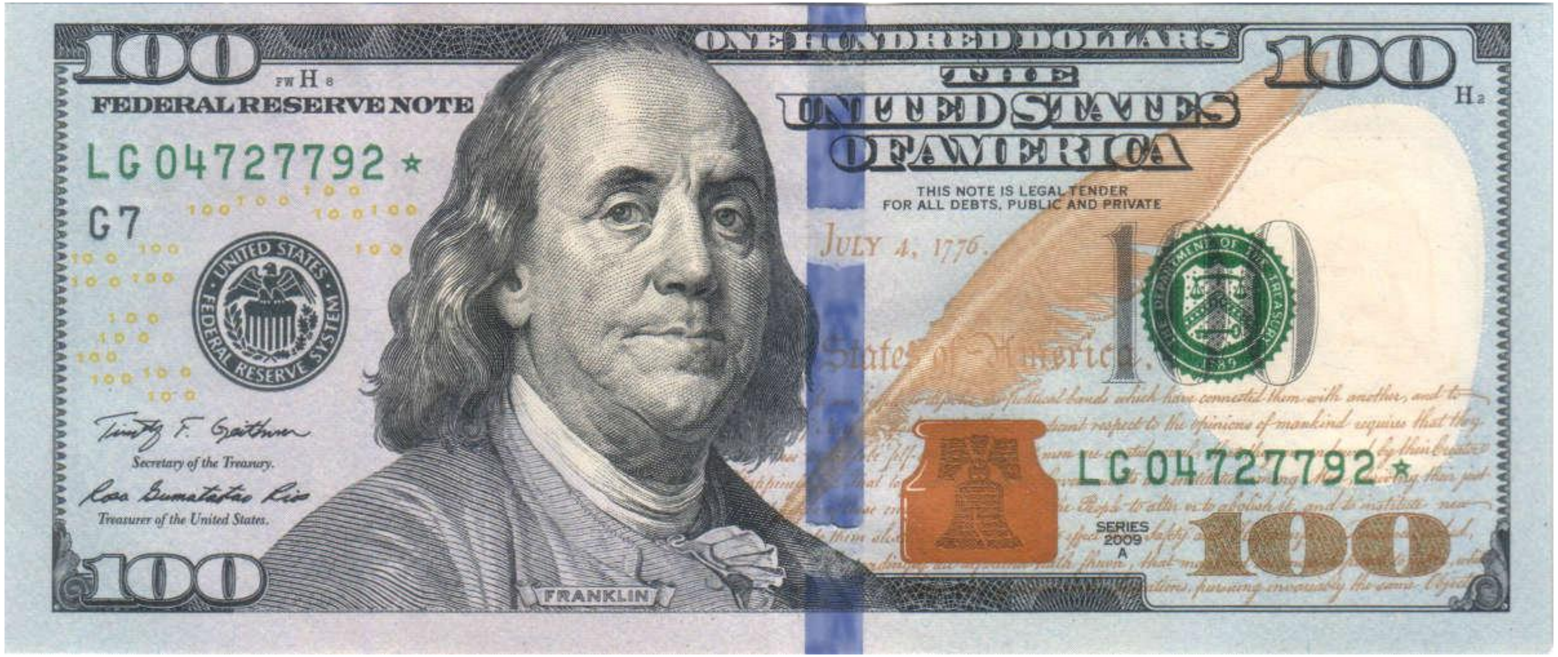
Defining Backend Caching

- **Python/Django** level caching
- Not browser/CDN caching
- Most of our caching is **read-through**
- Some of our caching is **write-through**
- **Caches are ephemeral**
    - Misses do not affect correctness
    - Data replication is not caching

Why Do We Love Caching?

Increases Speed

Reduces Costs



Real Answer: Low Effort and Risk

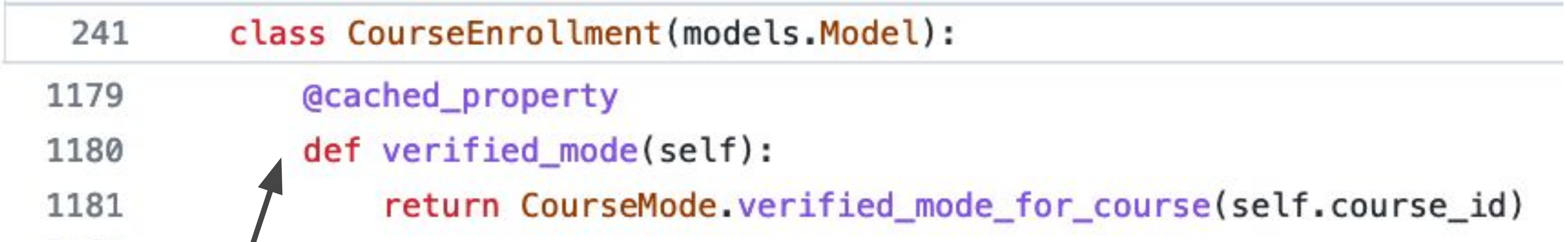
What are the Drawbacks?

- Code/Testing is More Complex (Global State)
- Behavior is Less Predictable (Cache Misses)
- Memory Leaks

Now to the fun stuff!

Python Caching: `functools` is Your Friend

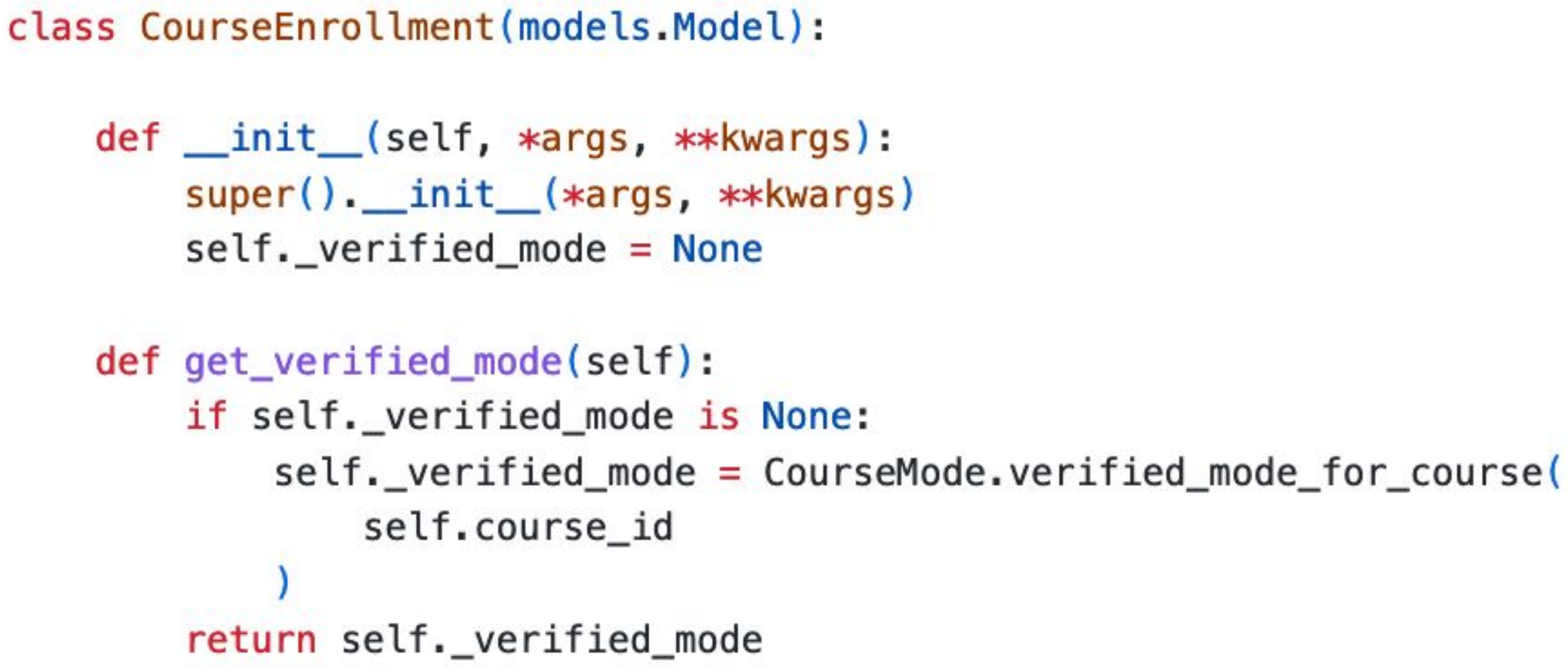
Instance Method Caching: `@cached_property`



```
class CourseEnrollment(models.Model):
    @cached_property
    def verified_mode(self):
        return CourseMode.verified_mode_for_course(self.course_id)
```

```
enrollment.verified_mode
```

Less Magical Version

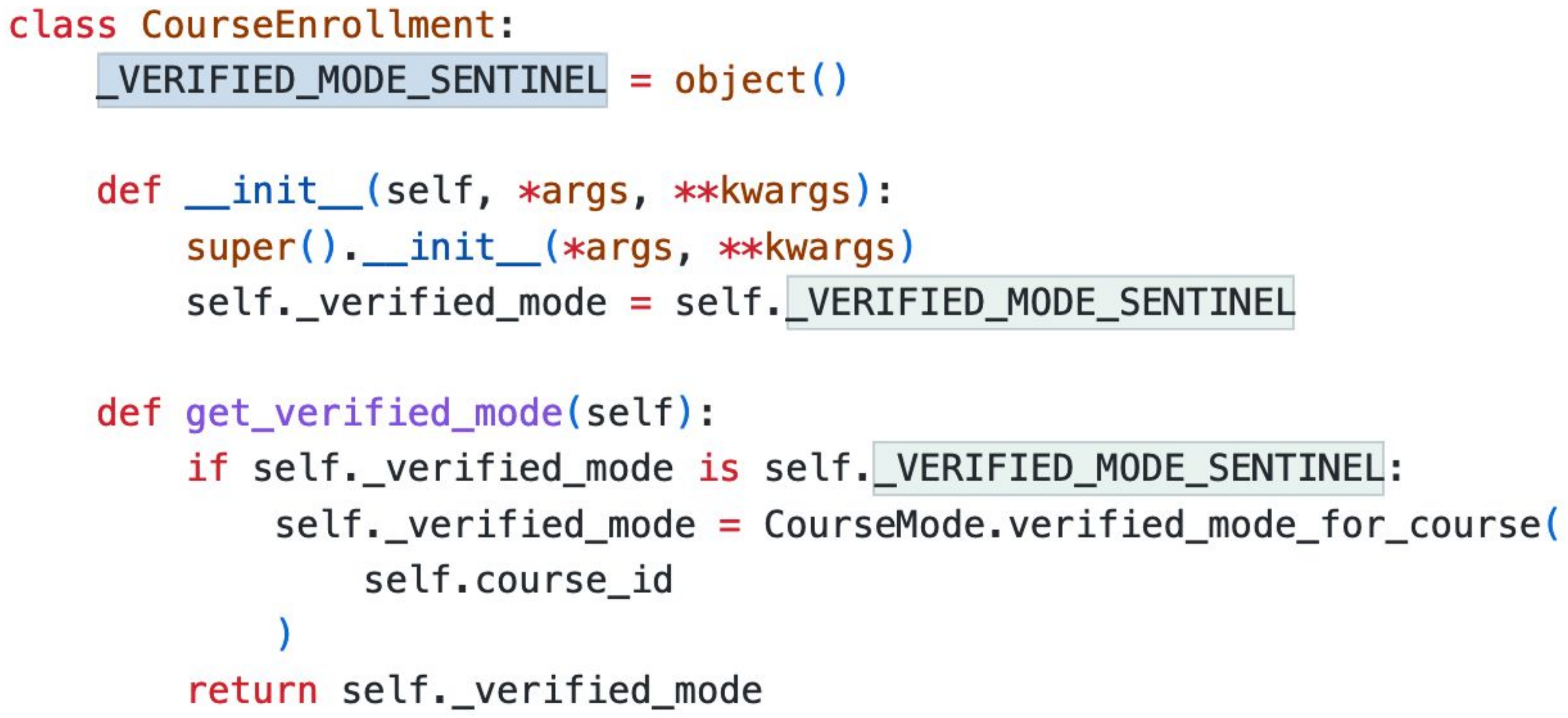


```python
class CourseEnrollment(models.Model):

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self._verified_mode = None

    def get_verified_mode(self):
        if self._verified_mode is None:
            self._verified_mode = CourseMode.verified_mode_for_course(
                self.course_id
            )
        return self._verified_mode
```

Use Sentinels to Handle None



```python
class CourseEnrollment:
    _VERIFIED_MODE_SENTINEL = object()

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self._verified_mode = self._VERIFIED_MODE_SENTINEL

    def get_verified_mode(self):
        if self._verified_mode is self._VERIFIED_MODE_SENTINEL:
            self._verified_mode = CourseMode.verified_mode_for_course(
                self.course_id
            )
        return self._verified_mode
```

Dave's Object-Oriented Rant: Don't Hide Remote Data Access

`enrollment.verified_mode`

"This is just an attribute."

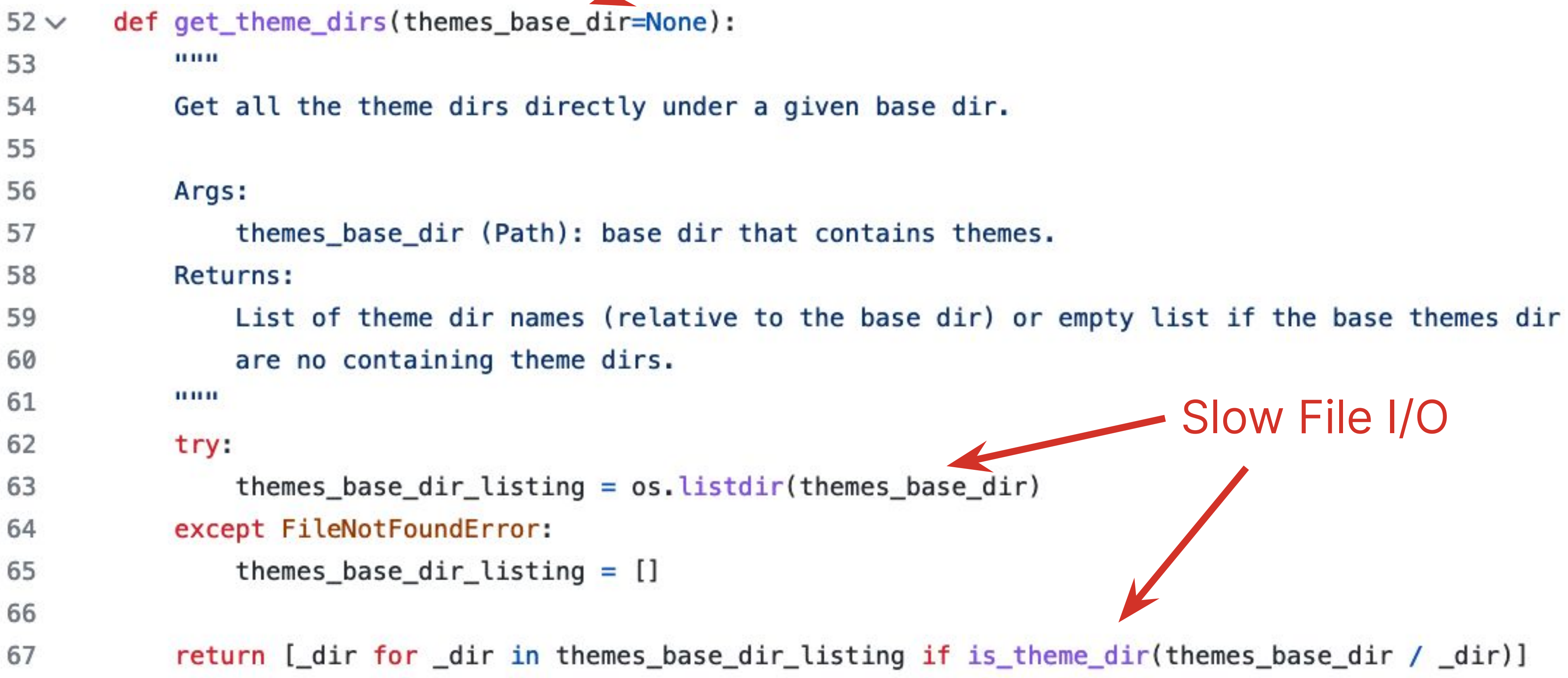
`enrollment.get_verified_mode()`

"Work is happening."

Theming & Docker
Case Study

- Adding 4 themes could triple response times
- `is_theme_dir` invoked *thousands of times* per view
- Called `posix.isdir/listdir` → filesystem access
- Page cache was holding things together with VMs
- Poor performance when using Docker deployments
- Refactor to startup initialization...?

Can’t use @cached_property



```python
def get_theme_dirs(themes_base_dir=None):
    """
    Get all the theme dirs directly under a given base dir.

    Args:
        themes_base_dir (Path): base dir that contains themes.
    Returns:
        List of theme dir names (relative to the base dir) or empty list if the base themes dir
        are no containing theme dirs.
    """
    try:
        themes_base_dir_listing = os.listdir(themes_base_dir)
    except FileNotFoundError:
        themes_base_dir_listing = []

    return [_dir for _dir in themes_base_dir_listing if is_theme_dir(themes_base_dir / _dir)]
```

Slow File I/O

Picking a Caching Solution for Theming

- **No Expiration or Invalidation**

  Themes don’t change for the lifetime of the process

- **Small Size / Few Values**

  Underlying `get_theme_dirs` mostly called the same way (optional arg)

- **Low Latency is Critical**

  `get_theme_dirs` is called *thousands of times* in a request

Solution: `functools.lru_cache`

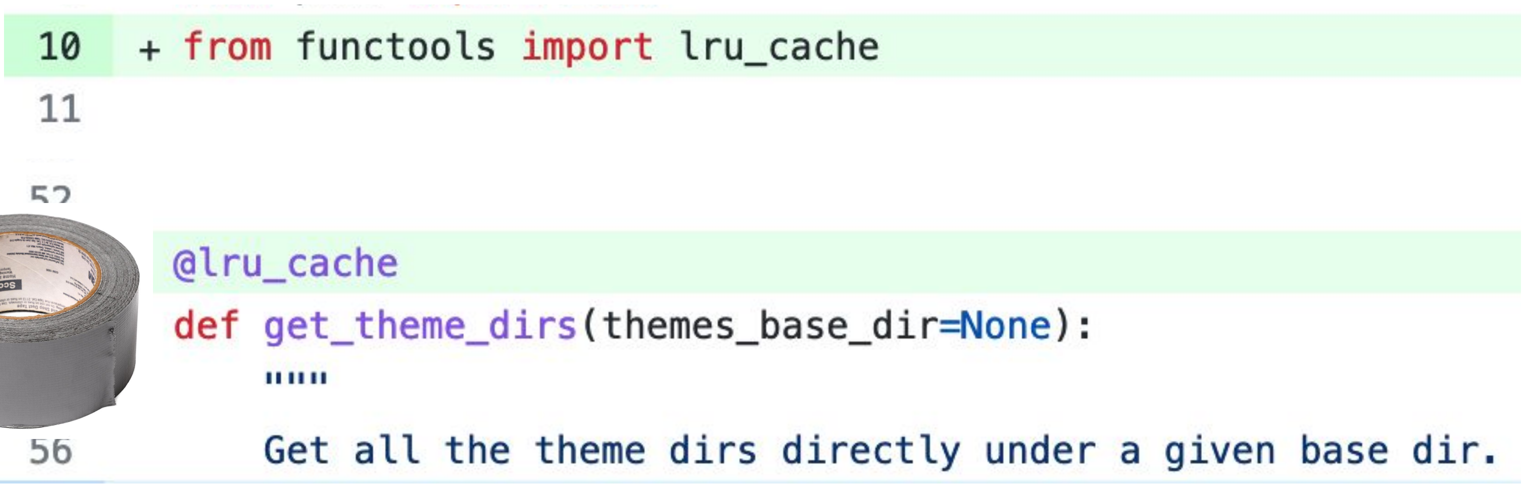
- **No Expiration or Invalidation**

  You can only clear `lru_cache`, not selectively invalidate keys
- **Small Size / Few Values**

  Default maxsize is 128, kept in memory
- **Low Latency is Critical**

  Runs in-process with a dict underneath–this is as fast as you get

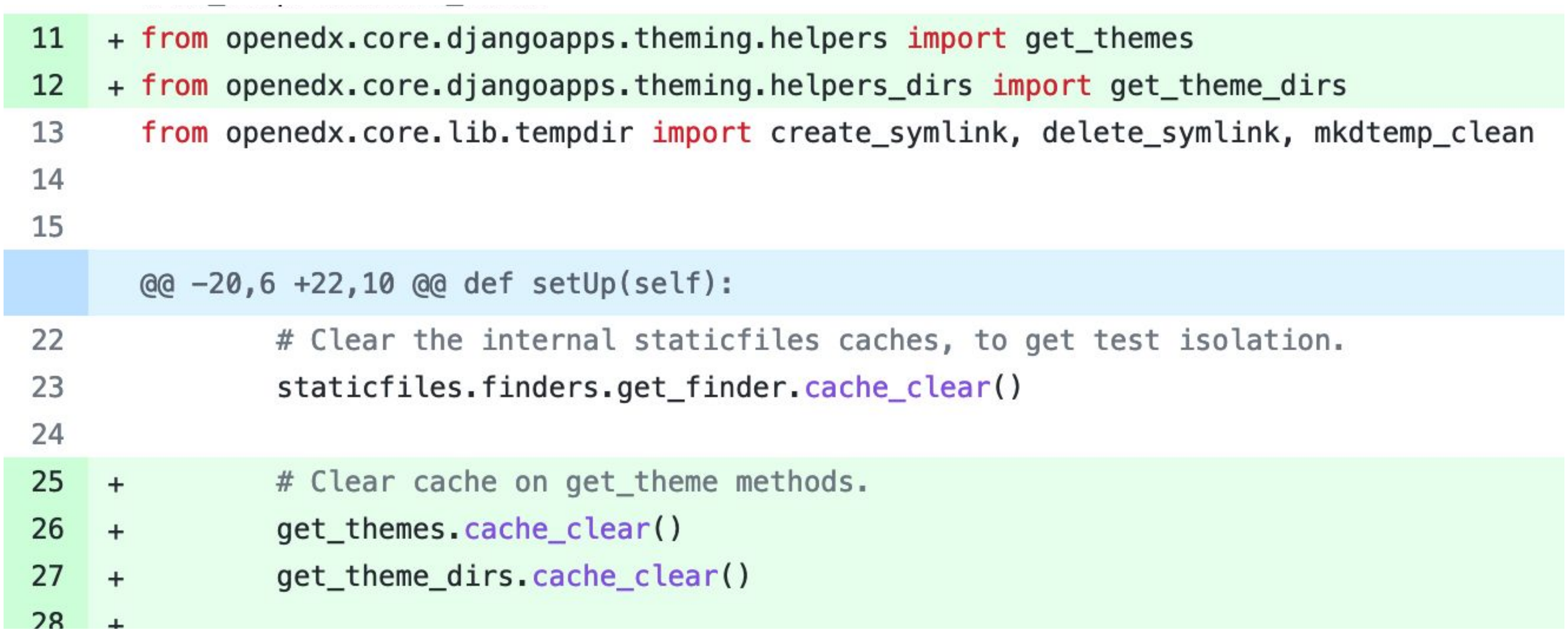
Using `lru_cache`



```
+ from functools import lru_cache


@lru_cache
def get_theme_dirs(themes_base_dir=None):
    """
    Get all the theme dirs directly under a given base dir.
```

Clearing `lru_cache` in Tests



```
+ from openedx.core.djangoapps.theming.helpers import get_themes
+ from openedx.core.djangoapps.theming.helpers_dirs import get_theme_dirs
  from openedx.core.lib.tempdir import create_symlink, delete_symlink, mkdtemp_clean


  @@ -20,6 +22,10 @@ def setUp(self):
          # Clear the internal staticfiles caches, to get test isolation.
          staticfiles.finders.get_finder.cache_clear()

+         # Clear cache on get_theme methods.
+         get_themes.cache_clear()
+         get_theme_dirs.cache_clear()
+
```

Theming Caching Results

20 line PR (edx-platform #31090)

Alejandro Cardenas (Alec4r)
eduNEXT

	Before	After
avg	3.17 s	633.36 ms
min	133.3 ms	101.13 ms
med	**3.56 s**	**606.90 ms**
max	7.38 s	2.15 s
p(90)	4.45 s	1.13 s
p(95)	**4.80 s**	**1.27 s**

Django Cache Framework

Django Cache Framework in a Nutshell

- get, set, delete
- get_many, set_many, delete_many
- View-level caching less used:
  - `cache_page`
  - `vary_on_headers, vary_on_cookie, cache_control`
- Multiple cache backends/named caches.
  - Usually Redis and Memcached

MemcachedCache

Due to a **python-memcached** limitation, it's not possible to distinguish between stored **None** value and a cache miss signified by a return value of **None** on the deprecated **MemcachedCache** backend.

```
>>> sentinel = object()
>>> cache.get('my_key', sentinel) is sentinel
>>> from django.core.cache import cache
>>> sentinel = object()
>>> cache.get('demo-key', sentinel)
<object object at 0x1024dae20>
```

Cache Invalidation is Hard

- When to Expire → Stale Data
- Key Growth → Too Many Keys to Delete
- Errors → Cleanup Failures
- Race Conditions → Inconsistent State

Create New Keys Instead!

```
cache.get(f"course_outline.{course_key}")

cache.delete(f"course_outline.{course_key}")
```

↓

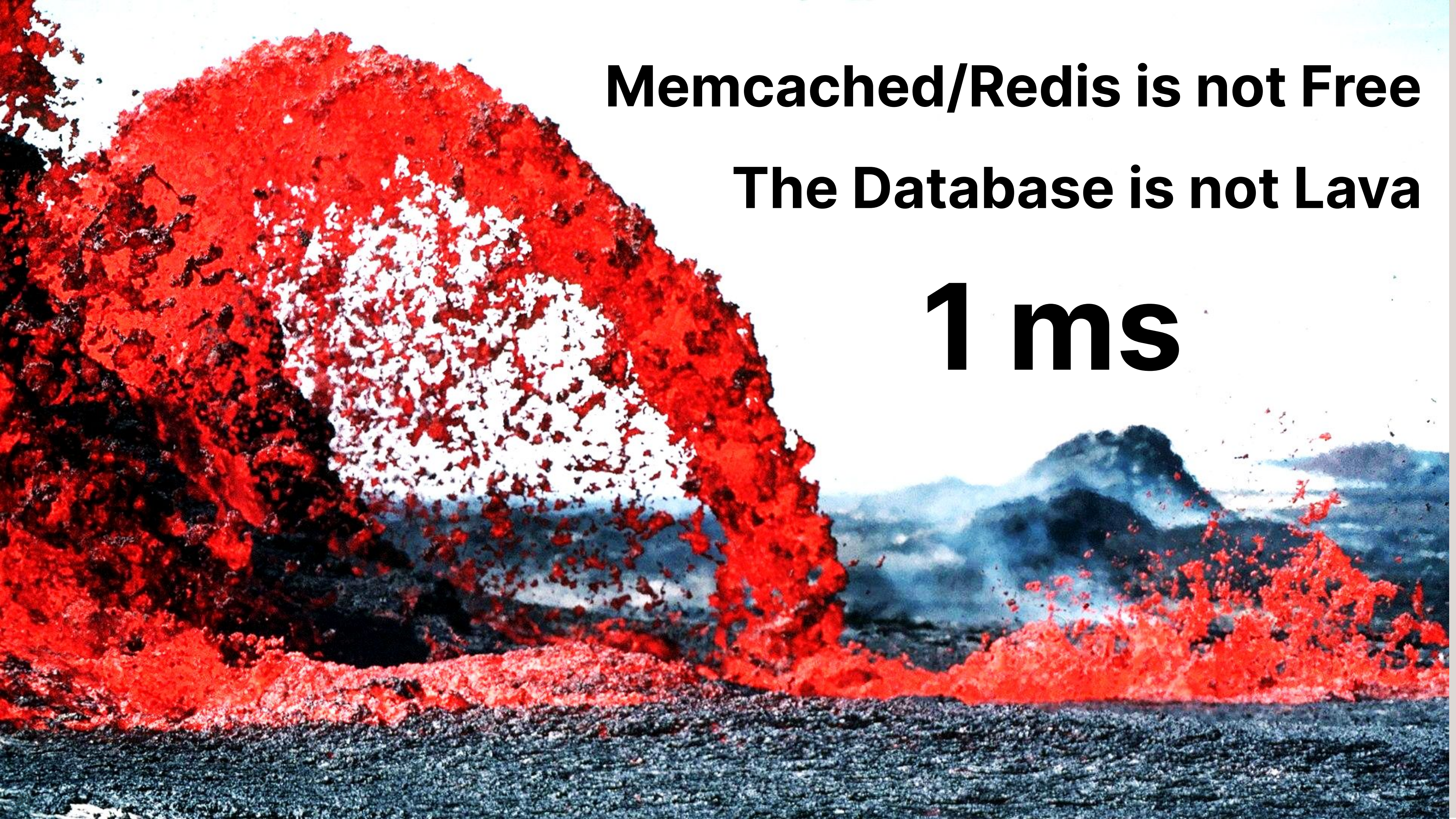
```
cache.get(f"course_outline.{course_key}.{version}")
```

Making New Cache Keys is Easy

- When to Expire → Never! It's always Truth.
- Key Growth → No cleanup
- Errors → The next request fixes it
- Race Conditions → Versions are isolated

But wait, where does the version come from?

```
cache.get(f”course_outline.{course_key}.{version}”)
```

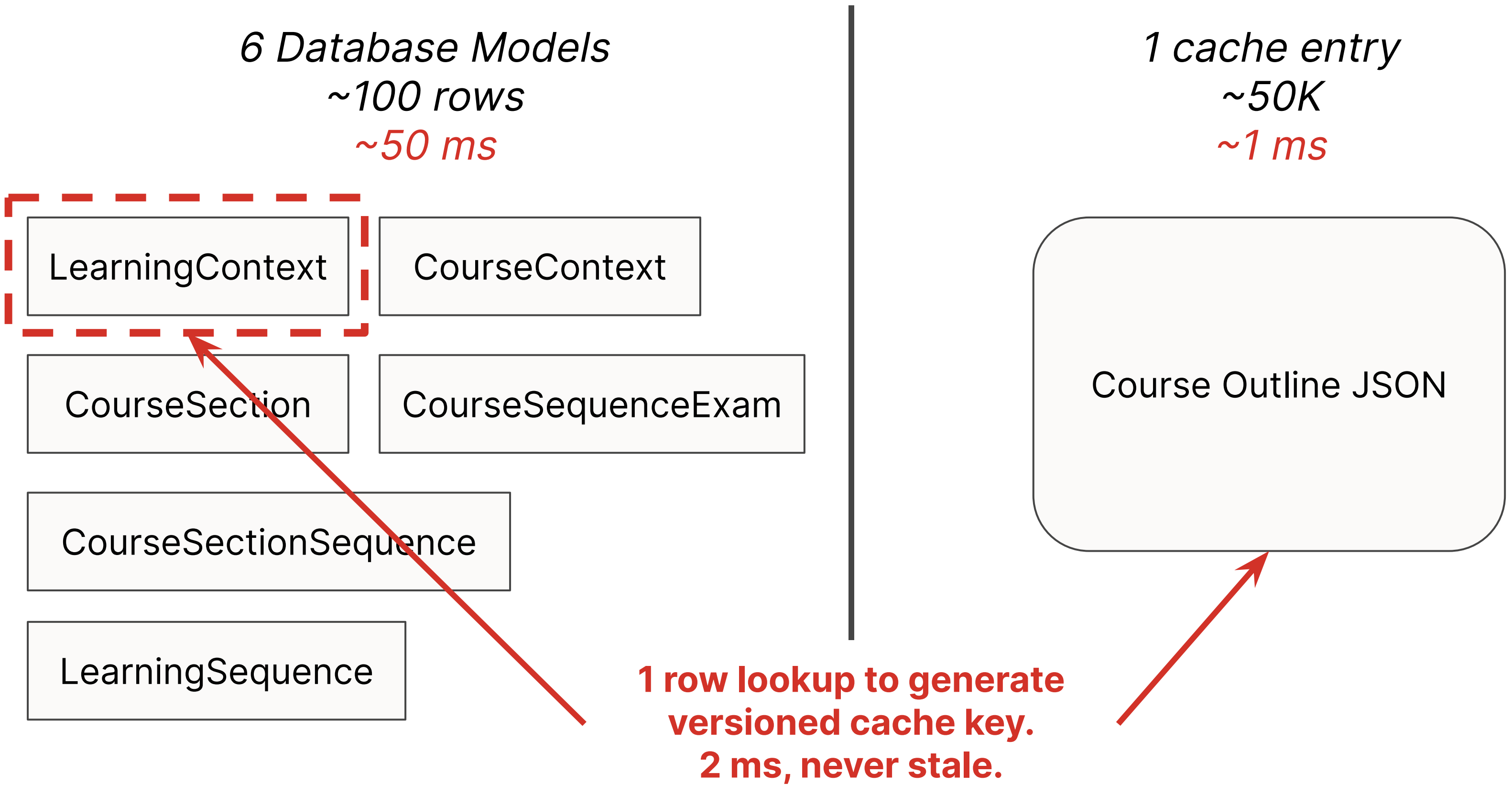


Memcached/Redis is not Free

The Database is not Lava

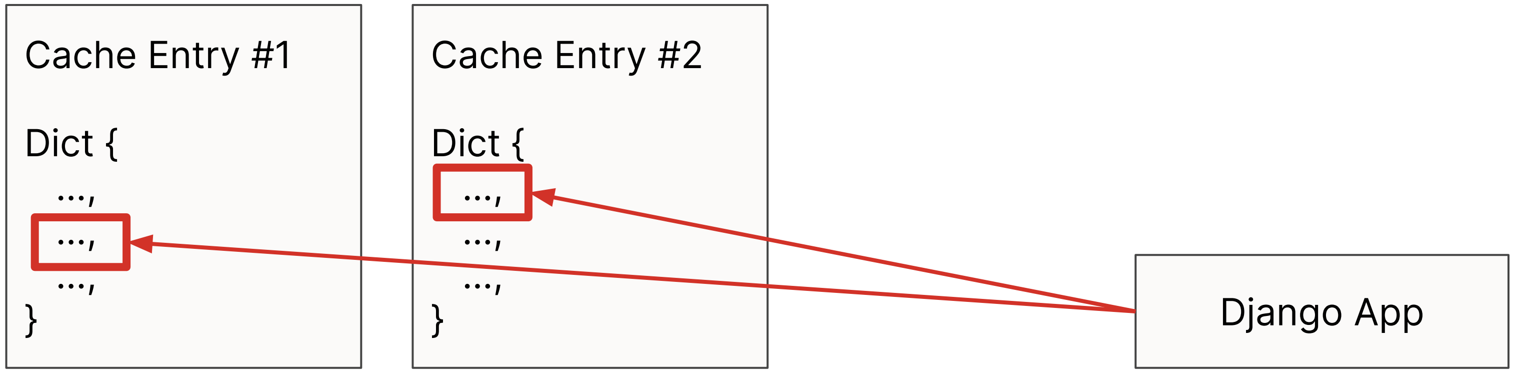
1 ms

Database vs. Cache: Course Outlines



Database vs. Cache: Programs Cache (Old)

- Cache entry for mapping of Programs → Course Runs
- Does. Not. Scale.
- **DO NOT USE CACHE ENTRIES AS A DATABASE**



Database

Complex Models

Persistent

More Expensive

Cache

Simple Key/Value

Ephemeral

Cheaper

A Few Odds and Ends...

Alt Backend: Django Local Memory Cache

- `django.core.cache.backends.locmem.LocMemCache`
- Very fast
- Process-specific
- **Unlike `lru_cache`, entries can expire**
- Memory Leaks – keys only removed on access
- Less useful than it sounds

Even Less Useful Backends

- Dummy Cache (for dev)

  `django.core.cache.backends.dummy.DummyCache`

- File Based Cache (don’t use this)

  `django.core.cache.backends.filebased.FileBasedCache`

- Database Cache (don’t use this)

  `django.core.cache.backends.db.DatabaseCache`

Open edX Caching Utilities

(OEP-22, edx-django-utils)

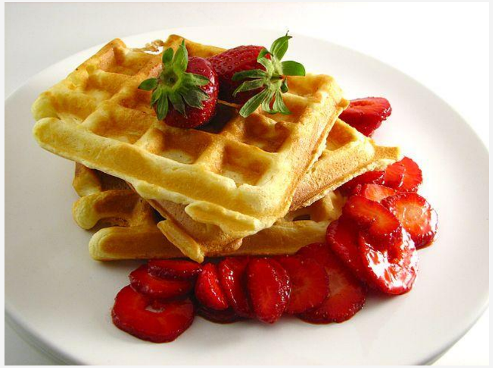
Special Thanks to Robert Raposa & Chris Lee (edX/2U)

Memcached/Redis caching is

10,000x slower

than Python caching

Course Waffle Flags
Case Study



- Size: *Large number of keys (100K+)*

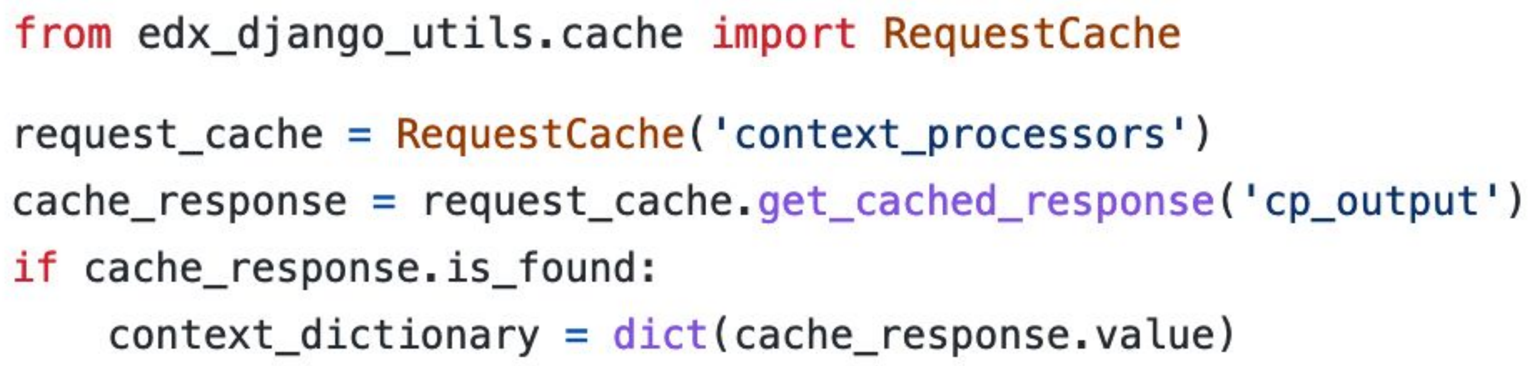
  Django `LocMemCache` leaks memory
- Frequency: *Hundreds of lookups per request*

  Memcached/Redis is too slow
- Lifetime: *Less than a minute*

  `@lru_cache` has no key timeout/invalidation

RequestCache

- In-memory cache cleared at the end of a request
- `edx_django_utils.cache.middleware.RequestCacheMiddleware`



```python
from edx_django_utils.cache import RequestCache

request_cache = RequestCache('context_processors')
cache_response = request_cache.get_cached_response('cp_output')
if cache_response.is_found:
    context_dictionary = dict(cache_response.value)
```

RequestCache vs. @cached_property

- View function
  - function
    - **for loop**
      - function
        - function
          - function
            - function
              - function
                - **Cache access**

Pass cache object through all these layers?

Can I Use Memcached/Redis and RequestCache together?

RequestCache + Django Cache

= TieredCache

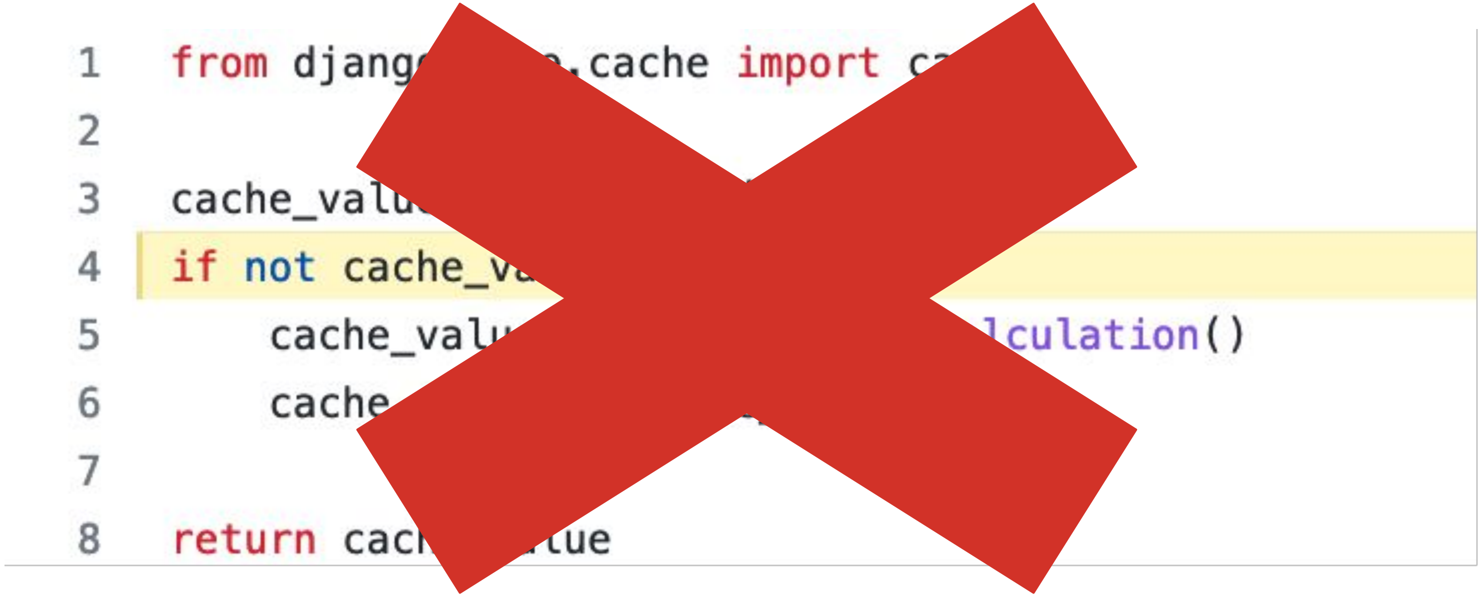
TieredCache Example: Course Outlines

```
cache_key = "learning_sequences.api.get_course_outline.v2.{}.{}".format(
    course_context.learning_context.context_key,
    course_context.learning_context.published_version,
)
outline_cache_result = TieredCache.get_cached_response(cache_key)
if outline_cache_result.is_found:
    return outline_cache_result.value


TieredCache.set_all_tiers(cache_key, outline_data, 300)
```

README: https://tinyurl.com/edx-django-utils-cache

Side Note: Cache Misses and “Falsy” Values



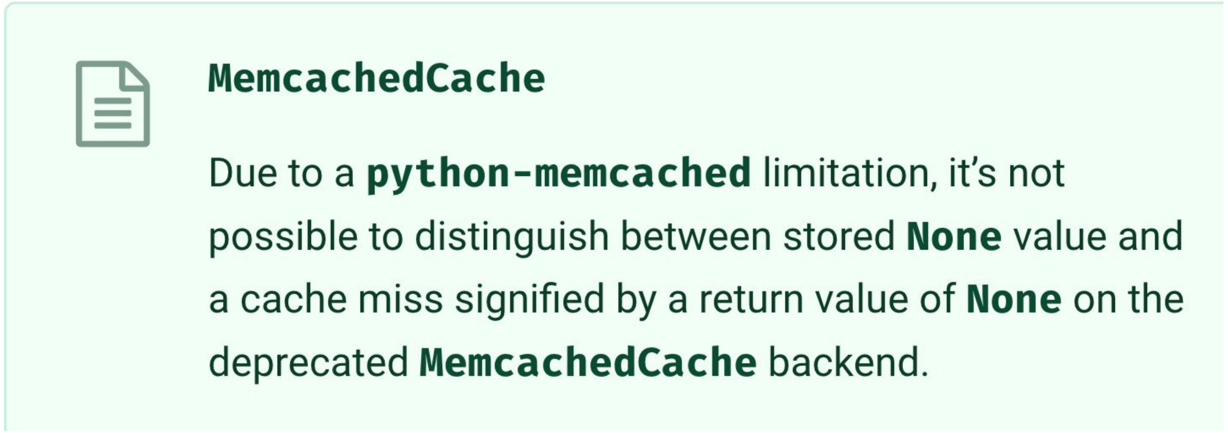
```
from djang      .cache import ca

cache_valu
if not cache_va
    cache_valu                    lculation()
    cache

return cac      lue
```

Side Note: CachedResponse

```
outline_cache_result = TieredCache.get_cached_response(cache_key)
if outline_cache_result.is_found:
    return outline_cache_result.value
```



MemcachedCache

Due to a **python-memcached** limitation, it's not possible to distinguish between stored **None** value and a cache miss signified by a return value of **None** on the deprecated **MemcachedCache** backend.

Side Note: Force Cache Miss with TieredCache

```
MIDDLEWARE = (
    'edx_django_utils.cache.middleware.RequestCacheMiddleware',
    'django.contrib.sessions.middleware.SessionMiddleware',
    ...
    # TieredCacheMiddleware middleware must come after these.
    'edx_django_utils.cache.middleware.TieredCacheMiddleware',
)
```

```
HTTP GET /api/v1/resource?force_cache_miss=true
```

Testing: `CacheIsolationTestCase`

- edx-platform: `openedx.core.djangolib.testing.utils`
- It *should* be in edx-django-utils
- `CacheIsolationMixin` and `CacheIsolationTestCase`
- Resets the Request Cache and Django Caches between tests

Test That Your Caching Actually Works

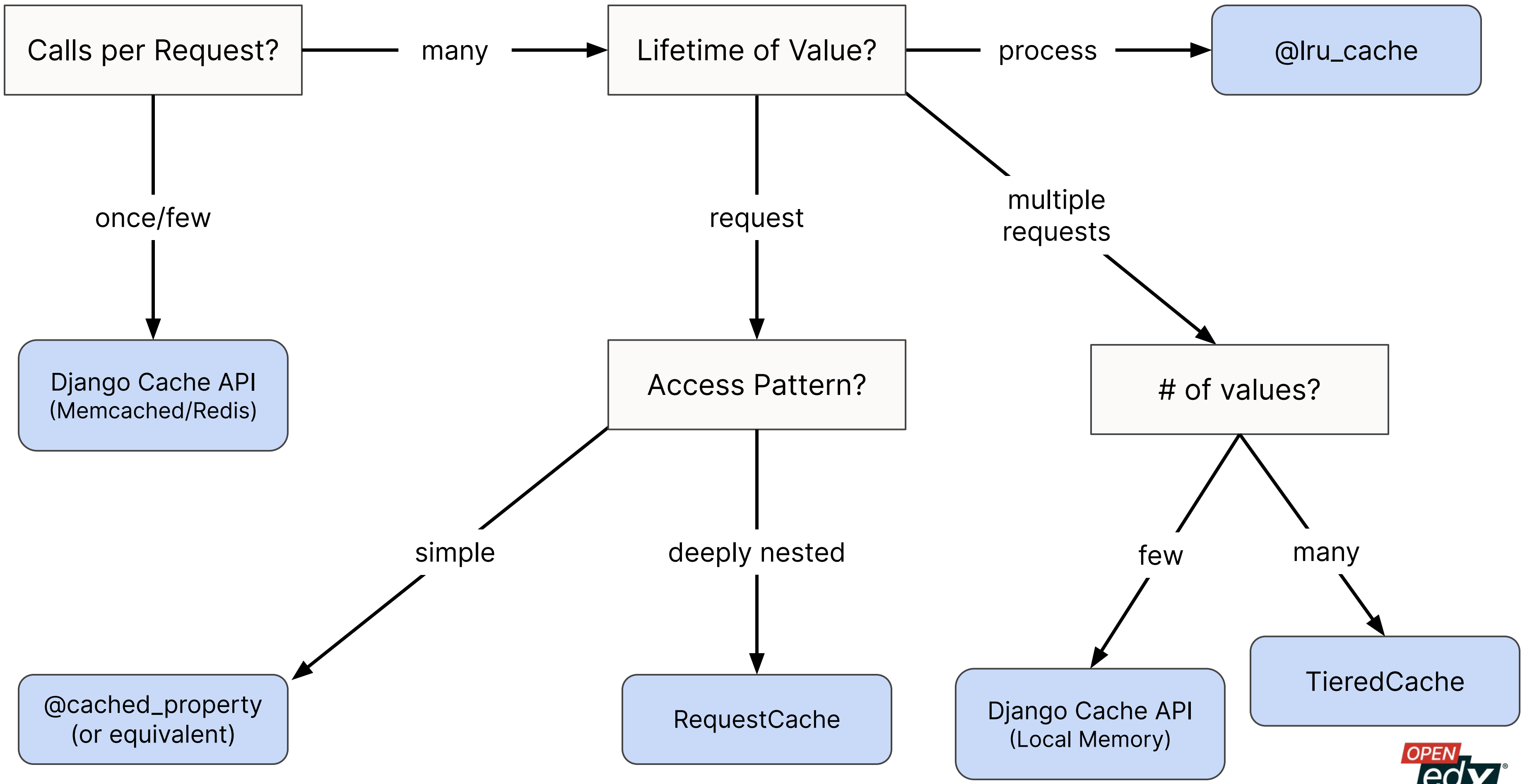
- `assertNumQueries` in uncached and cached states
- Do query counts only on `api.py` tests, not views.

```
# Uncached access always makes five database checks: LearningContext,
# CourseSection (+1 for user partition group prefetch),
# CourseSectionSequence (+1 for user partition group prefetch)
with self.assertNumQueries(5):
    uncached_outline = get_course_outline(self.course_key)
    assert uncached_outline == self.course_outline

# Successful cache access only makes a query to LearningContext to check
# the current published version. That way we know that values are never
# stale.
with self.assertNumQueries(1):
    cached_outline = get_course_outline(self.course_key)
```

Let’s Review!

Deciding our Caching Strategy…


Calls per Request?
many
Lifetime of Value?
process
@lru_cache
once/few
request
multiple requests
Django Cache API (Memcached/Redis)
Access Pattern?
# of values?
simple
deeply nested
few
many
@cached_property (or equivalent)
RequestCache
Django Cache API (Local Memory)
TieredCache

Operational Issues

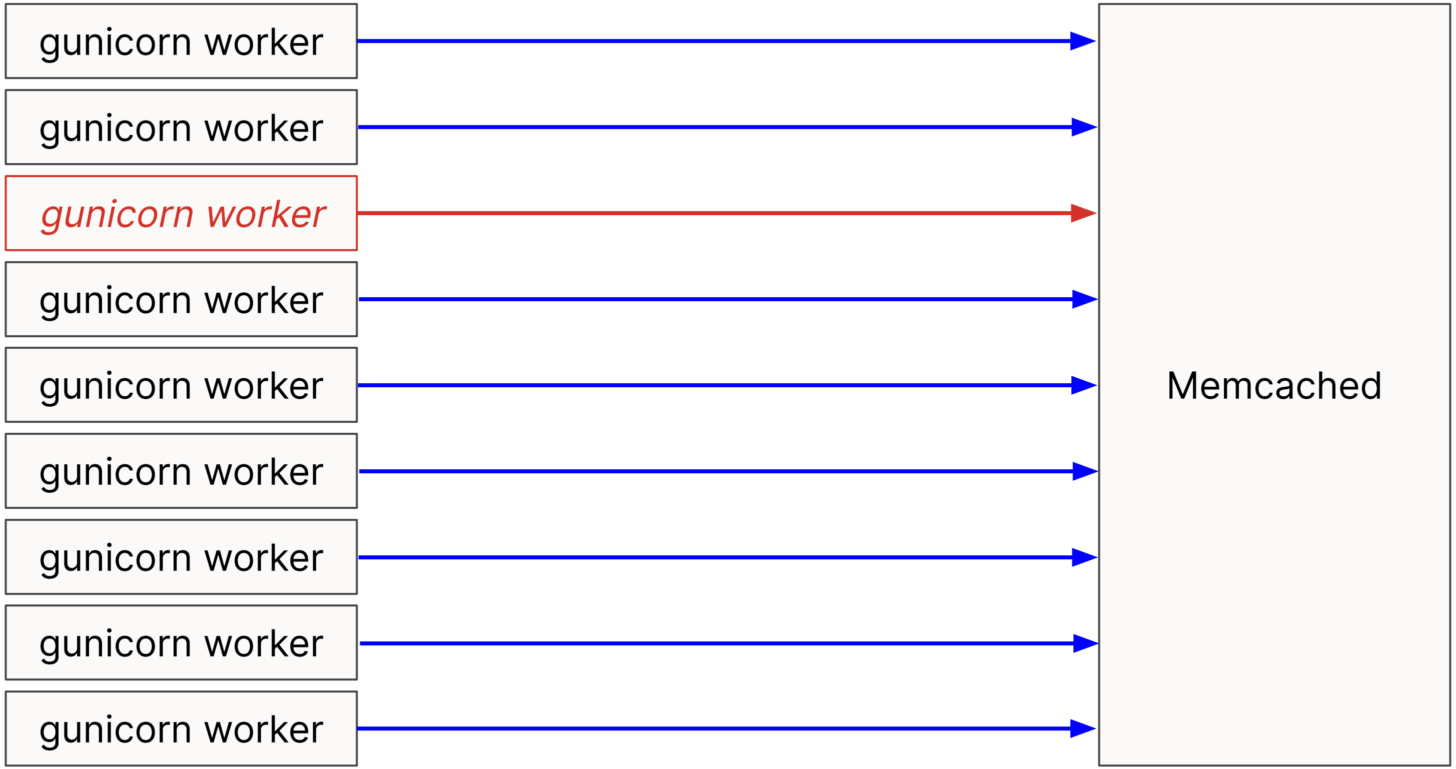
Memcached Size Limit

- `max_item_size` default is 1 MB
- Raise it to 2 MB

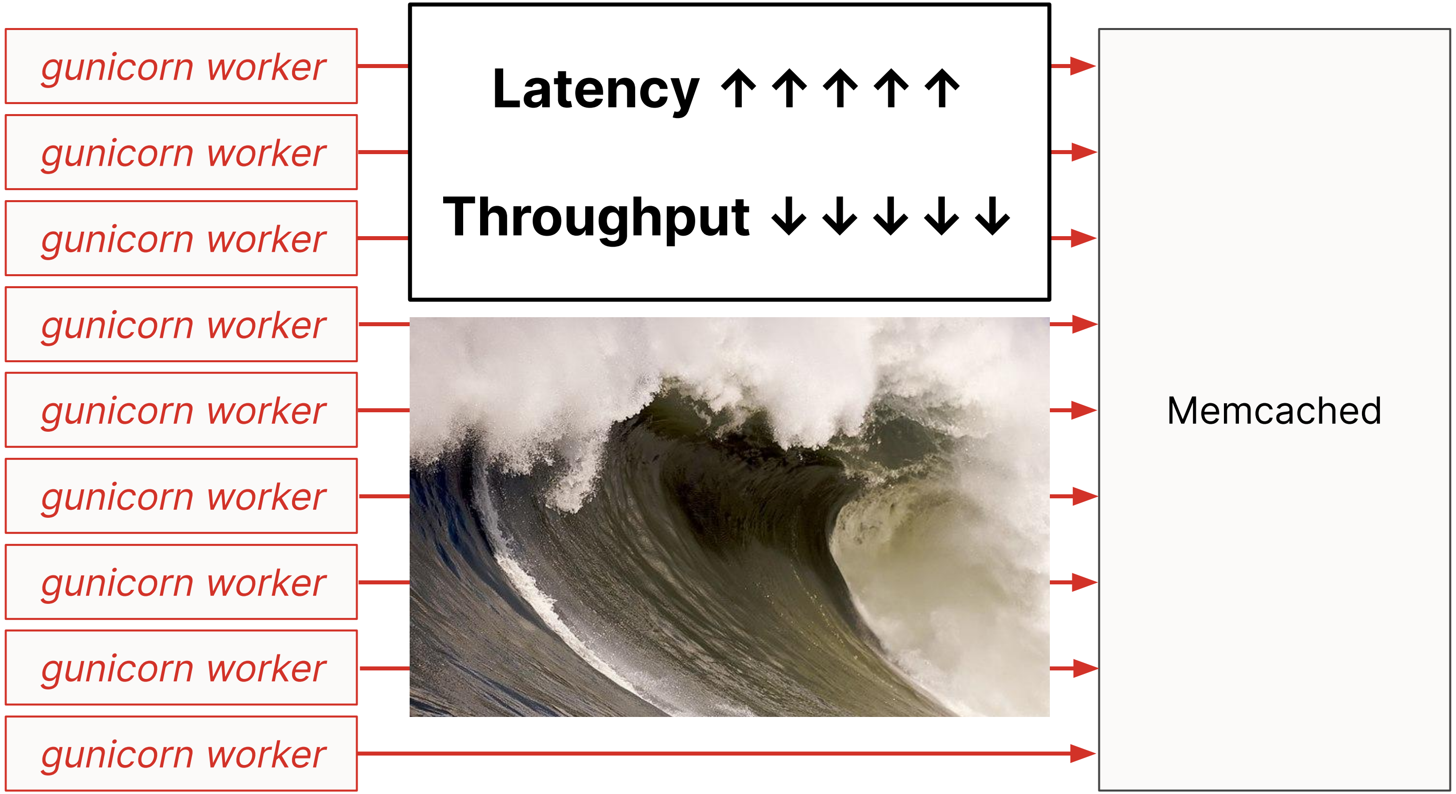
Memcached/Redis and Silent Failures

- `get/set` **will silently fail** if it can't reach the cache
  - Misconfiguration or server failure
- Takes ~1 minute to reconnect for Memcached

Cache Stampede / Dog-piling



Cache Stampede / Dog-piling

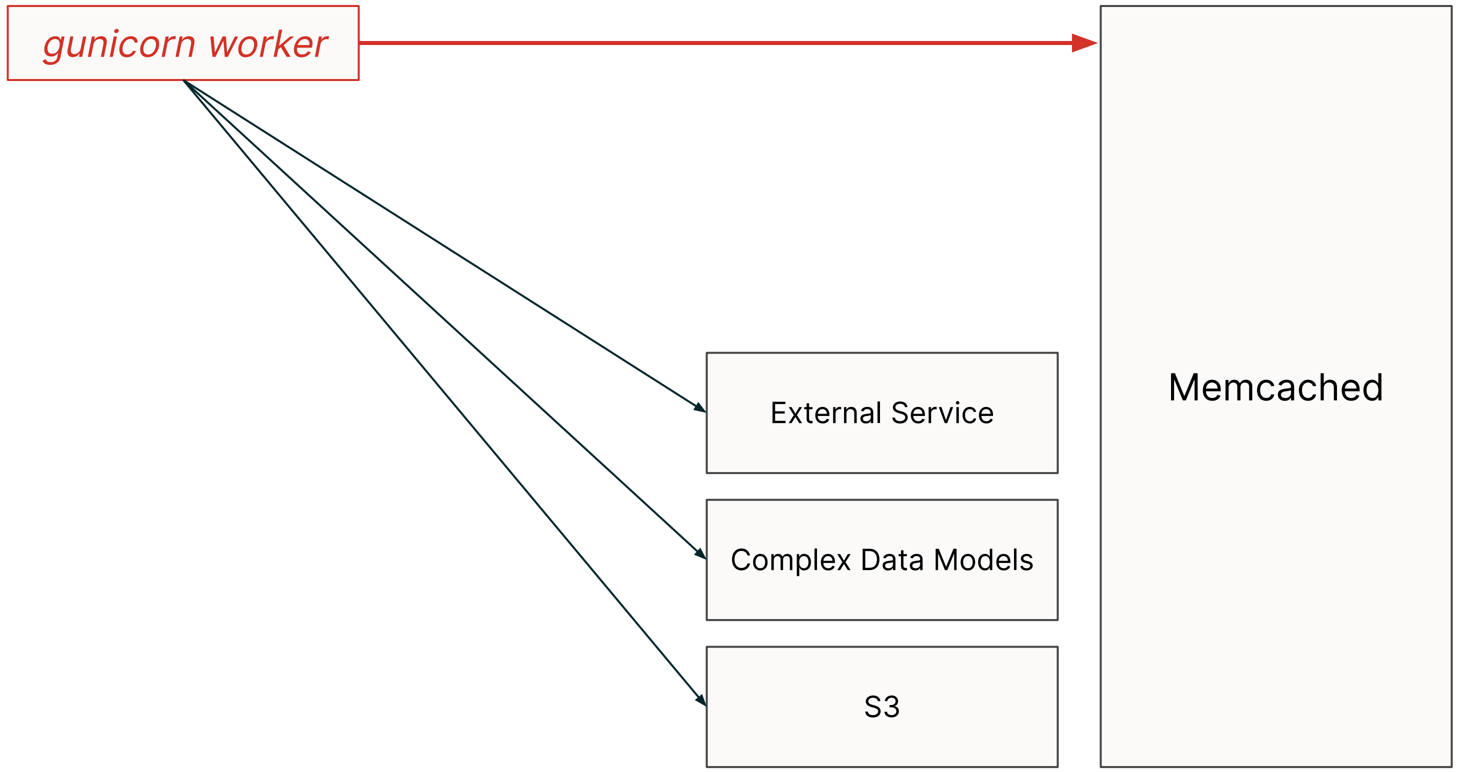


Cache Stampede Solution 1: Add Jitter

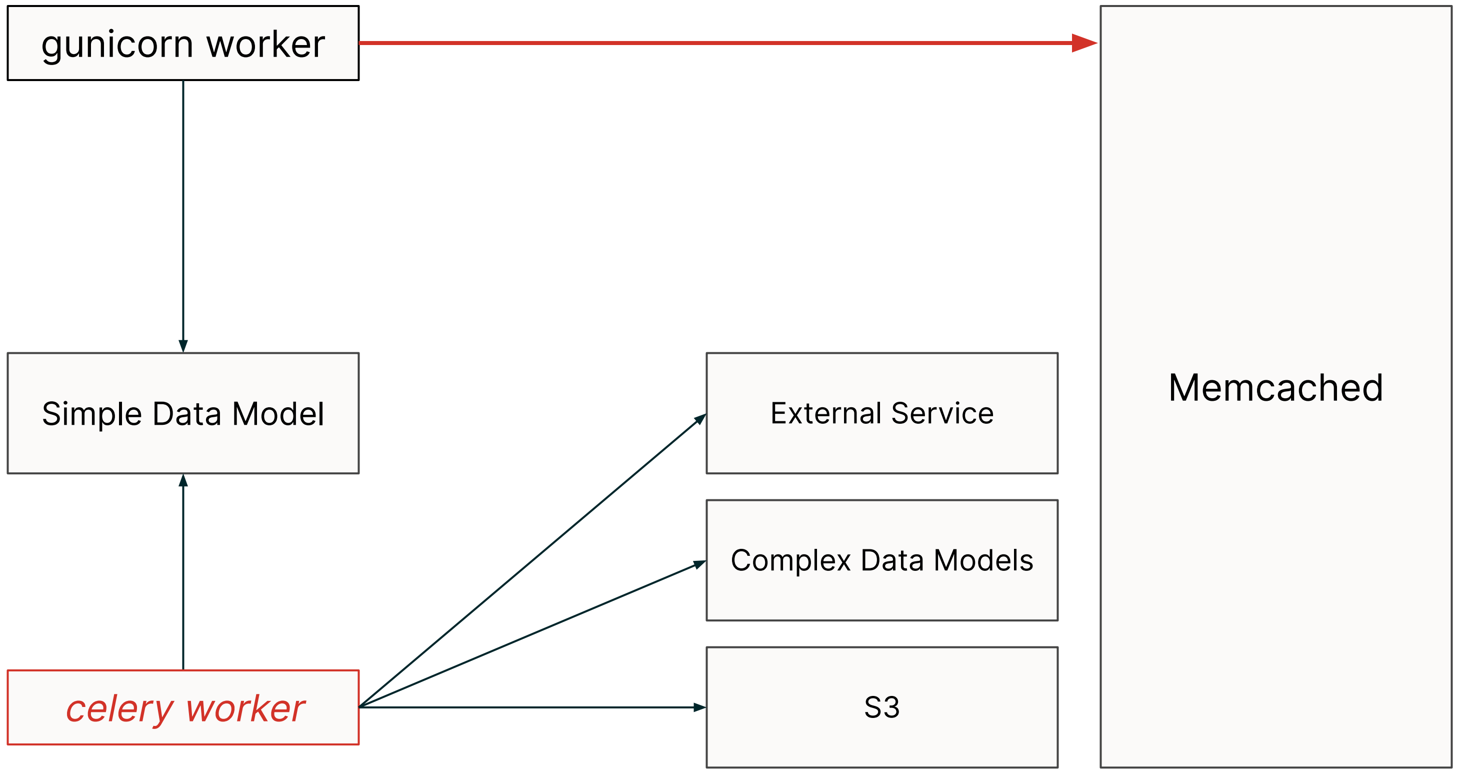
```
cache.set(cache_key, timeout=timeout)
```

```
jitter = random.randint(0, 300)
cache.set(cache_key, timeout=timeout + jitter)
```

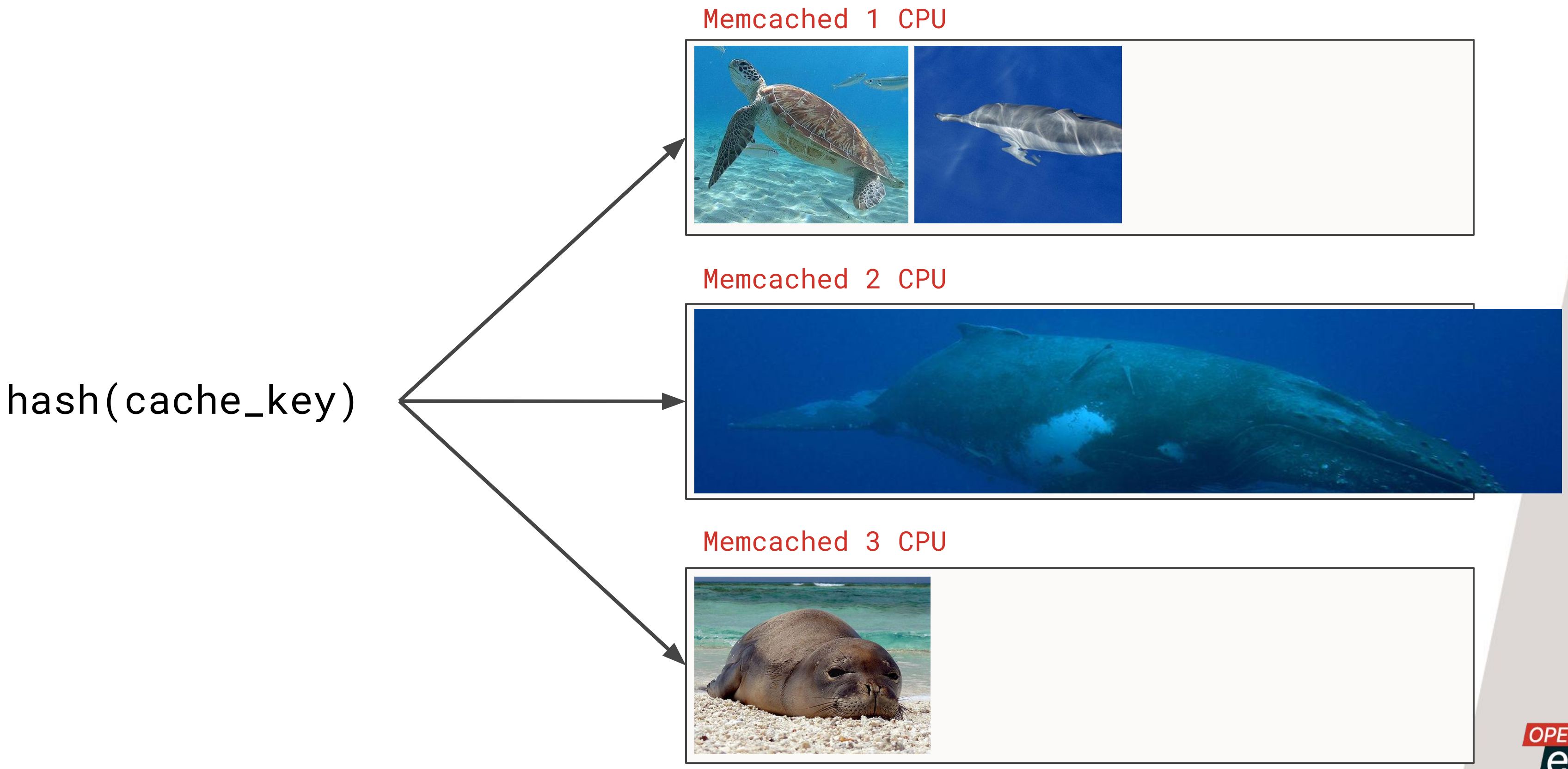
Cache Stampede Solution: Replicate Data



Cache Stampede Solution 2: Replicate Data

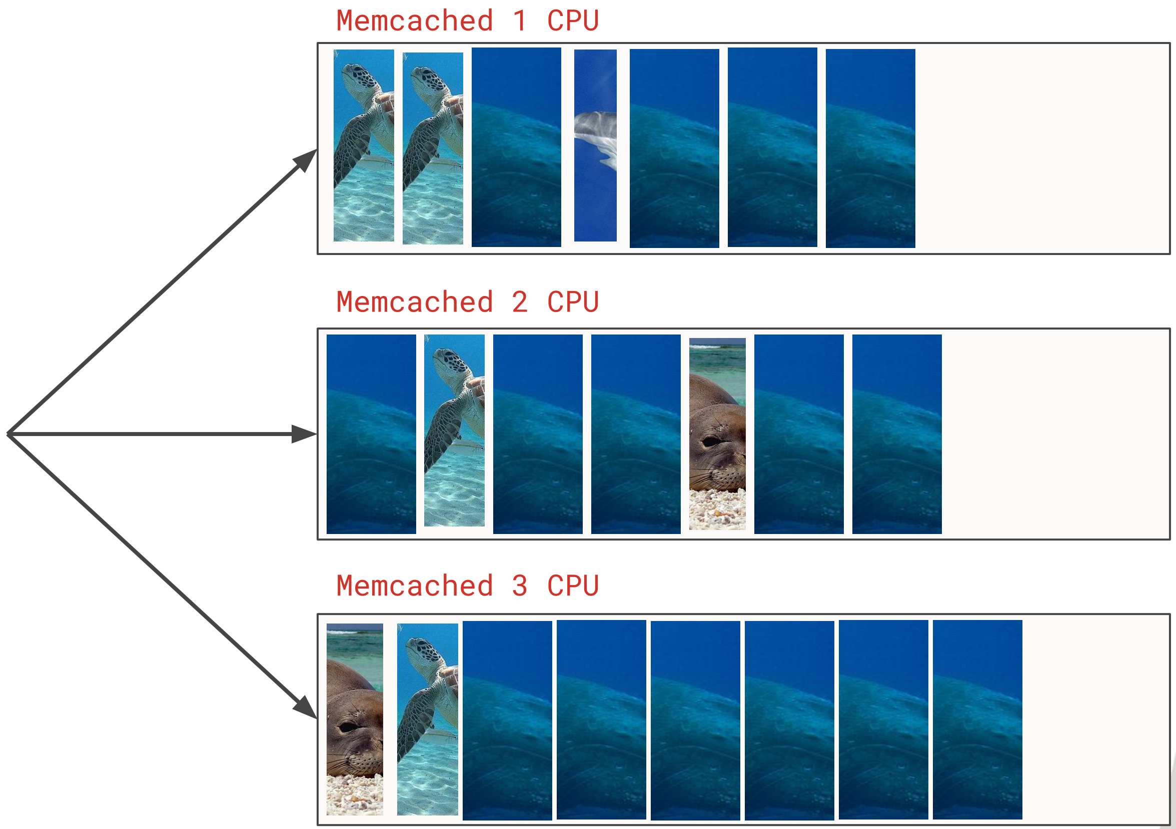


Load Balancing Course-based Cache Keys



Distribute With Randomization?

```
hash(
  cache_key +
  randint(1,10)
)
```



Design for Caching

It doesn't always have to be

What’s Safe to Cache?

```
problem_block.get_html()  # Can we cache this?
```

`get_html()` =

- Authored Content
- User state
- Feature Flags
- ?????

+

Think in Terms of Data Transformation

$$\text{Rendered ProblemBlock} = \boxed{\text{Authored Content}} + \text{User State}$$

$$\text{Course Outline for a Student} = \boxed{\text{All Course Sections and Subsections}} + \boxed{\text{Track-Specific}} + \text{User-Specific}$$

Manage Your Remote Data Access

- View function
  - **for loop**
    - function
      - function
        - function
          - **Cache get key 1 (first loop)**
    - function
      - function
        - function
          - **Cache get key 2 (next loop)**

This has a global picture. (Prefetch here.)

These have to fetch one at a time.

Looking at the Bigger Picture

```
cache.get(“course_waffle.{feature-1}”)
cache.get(“course_waffle.{feature-1}.{org}”)
cache.get(“course_waffle.{feature-1}.{course_key}”)
# Repeat for feature 2, 3, 4, etc. (most are empty)
```

Cache Entry:

All Active Features

Cache Entry:

Org Feature Overrides

Cache Entry:

Course Feature Overrides

```
def my_view(course_id, …):
    CourseWaffleFlag.prefetch(course=course_key)
```

Explicitly Model Remote Data Access

Caching Isn’t Something You Have to Hide!

Thank you!

Any questions?

Dave Ormsbee (dave@tcril.org)

GitHub team: @openedx/perf-interest