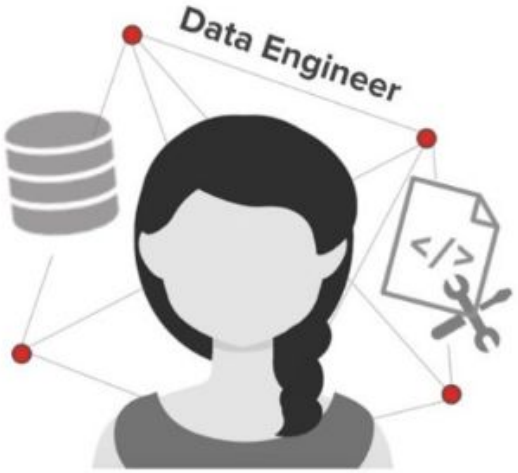
None, null, nil: lessons from caching & representing nothing with something

Felice Ho
PyTennessee 2020
Nashville, TN

I am Felice Ho



Overview

The scenario: what, why, where, how of caching

The problem with ‘nothing’

‘We have a problem’

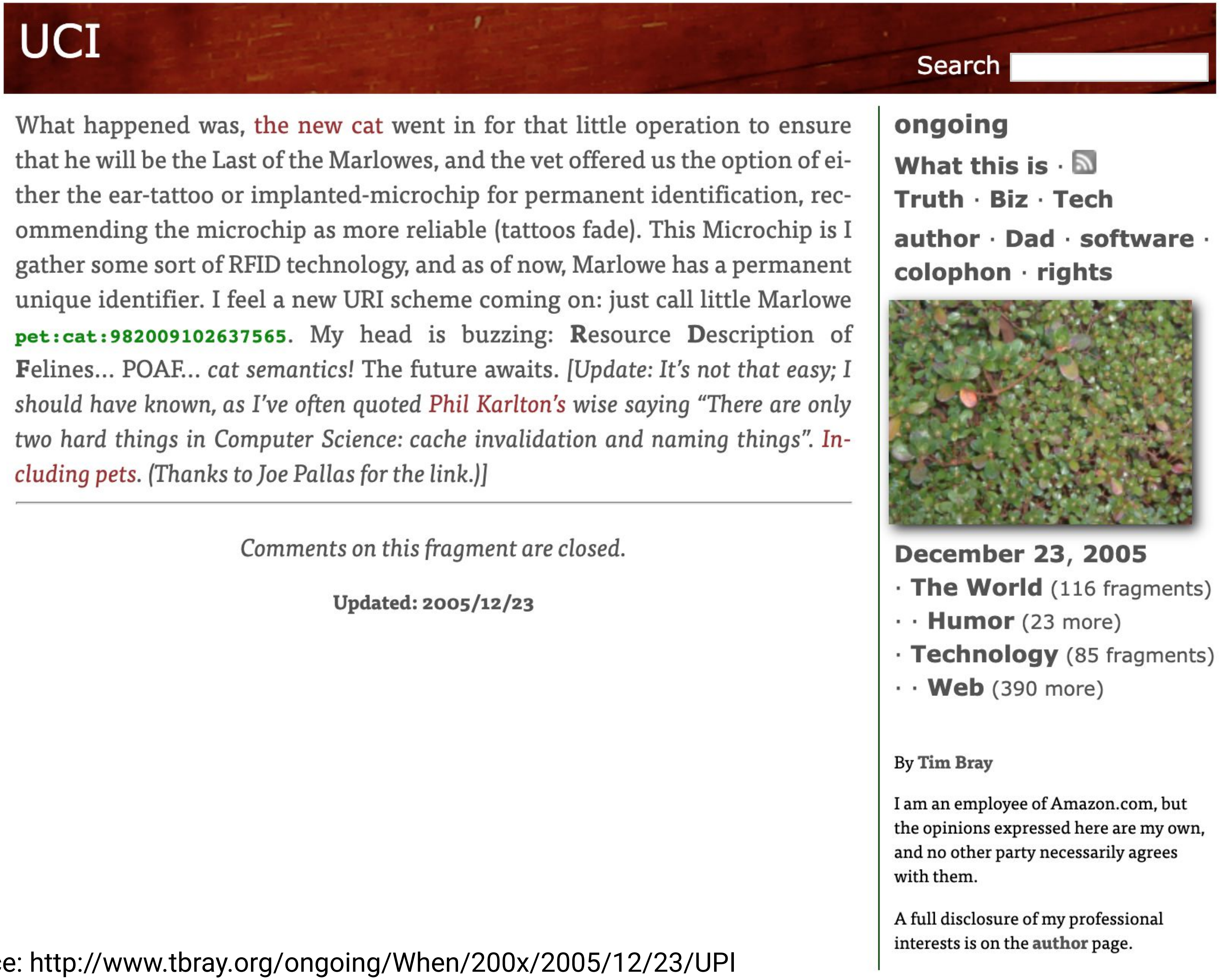
Root cause analysis

Lessons

The value of ‘nothing’

“There are only two hard things in Computer Science: cache invalidation and naming things”

- Phil Karlton



UCI

Search

What happened was, the new cat went in for that little operation to ensure that he will be the Last of the Marlowes, and the vet offered us the option of either the ear-tattoo or implanted-microchip for permanent identification, recommending the microchip as more reliable (tattoos fade). This Microchip is I gather some sort of RFID technology, and as of now, Marlowe has a permanent unique identifier. I feel a new URI scheme coming on: just call little Marlowe `pet:cat:982009102637565`. My head is buzzing: **R**esource **D**escription of **F**elines… POAF… *cat semantics!* The future awaits. *[Update: It's not that easy; I should have known, as I've often quoted Phil Karlton's wise saying "There are only two hard things in Computer Science: cache invalidation and naming things". Including pets. (Thanks to Joe Pallas for the link.)]*

Comments on this fragment are closed.

Updated: 2005/12/23

ongoing

What this is ·
Truth · Biz · Tech
author · Dad · software · colophon · rights

December 23, 2005
· **The World** (116 fragments)
· · **Humor** (23 more)
· **Technology** (85 fragments)
· · **Web** (390 more)

By **Tim Bray**

I am an employee of Amazon.com, but the opinions expressed here are my own, and no other party necessarily agrees with them.

A full disclosure of my professional interests is on the **author** page.

Source: http://www.tbray.org/ongoing/When/200x/2005/12/23/UPI

“The first place anyone found it on the internet was in Tim Bray's blog.

Tim said that he first heard it around 1996-7”

- Martin Fowler

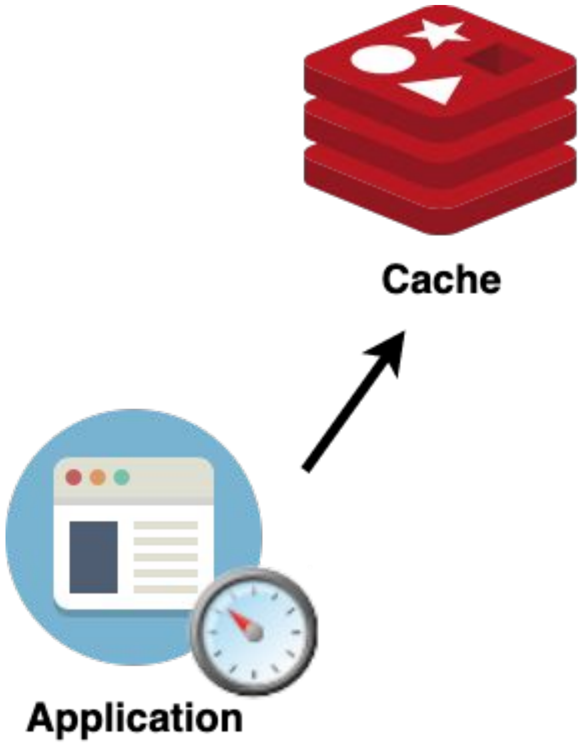
Source: https://martinfowler.com/bliki/TwoHardThings.html

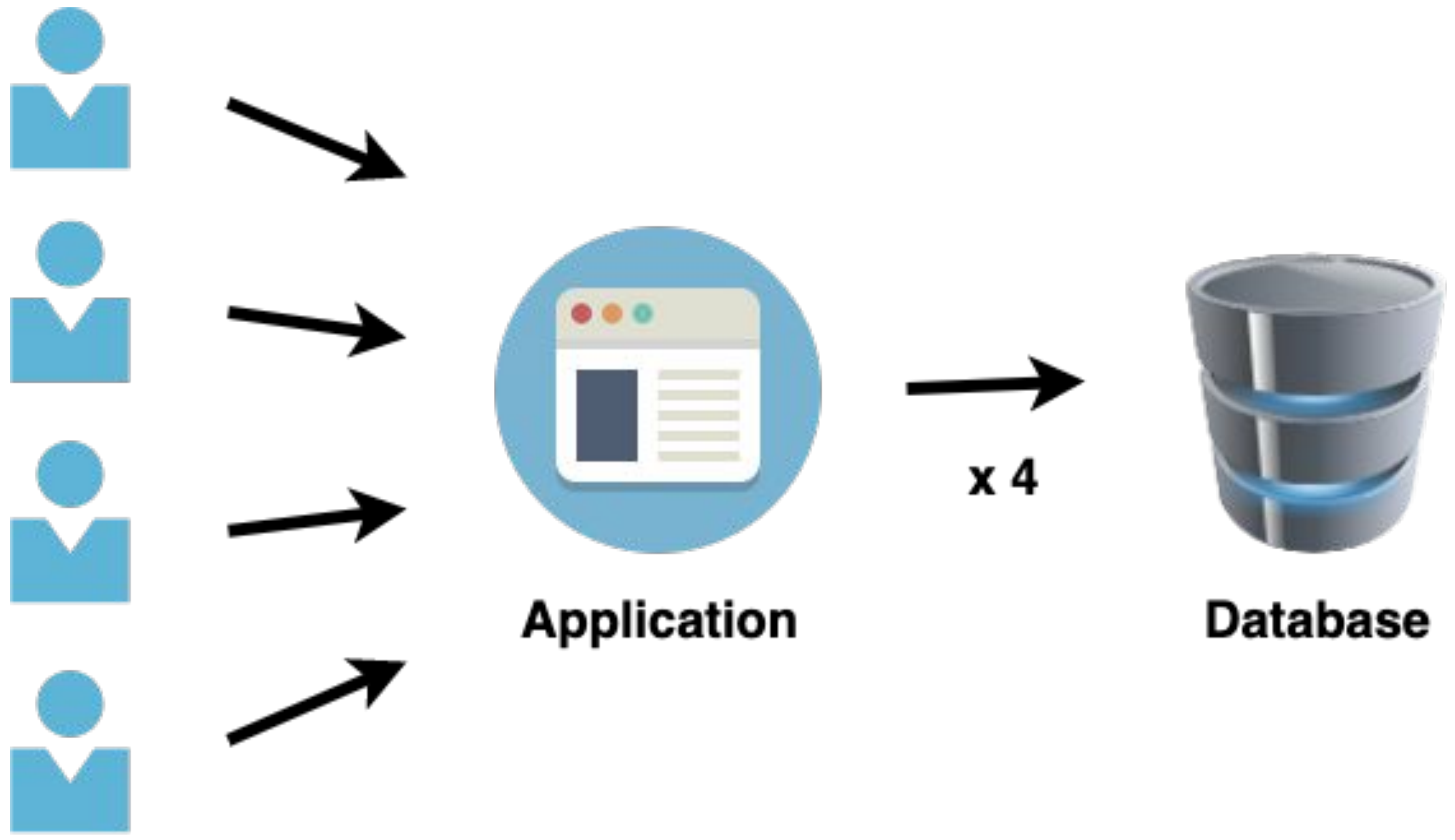
Goals of talk

How do we invalidate data in a cache?

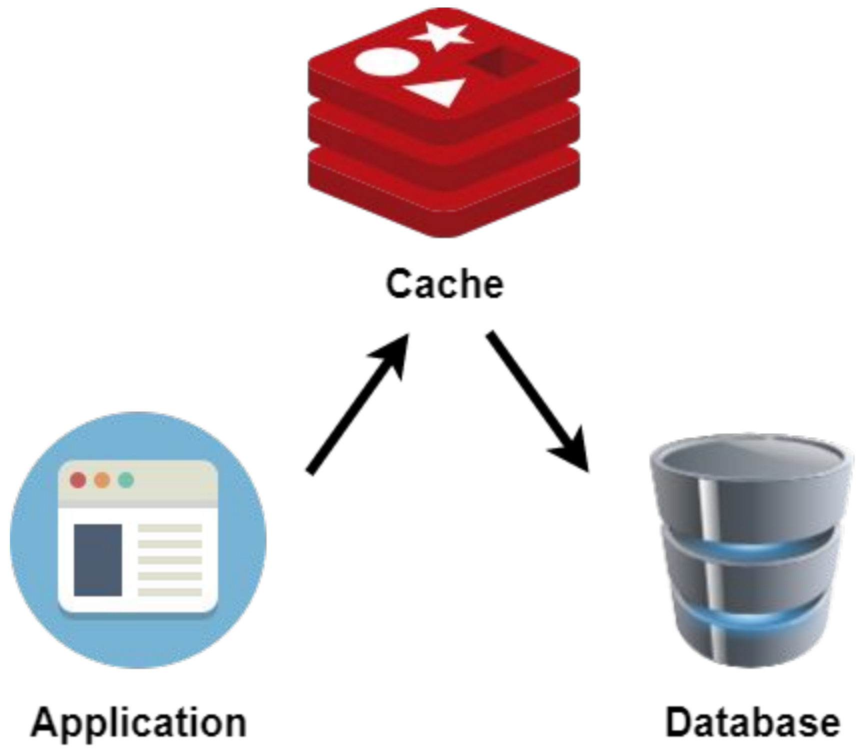
How can production break down - from *nothing*!

What
is a cache and cache invalidation?

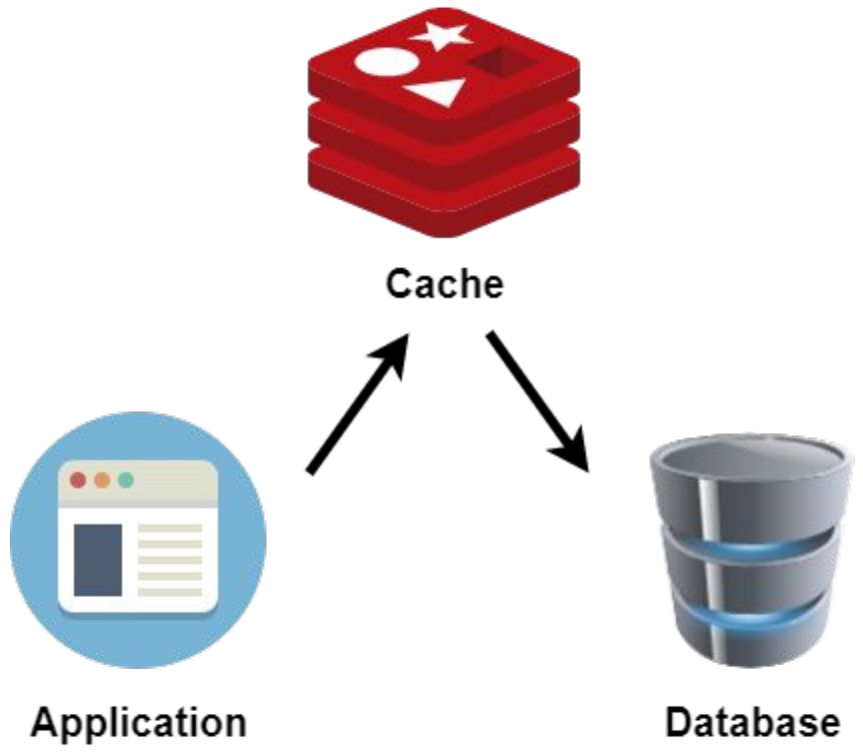

Cache
Application


x 4
Application
Database

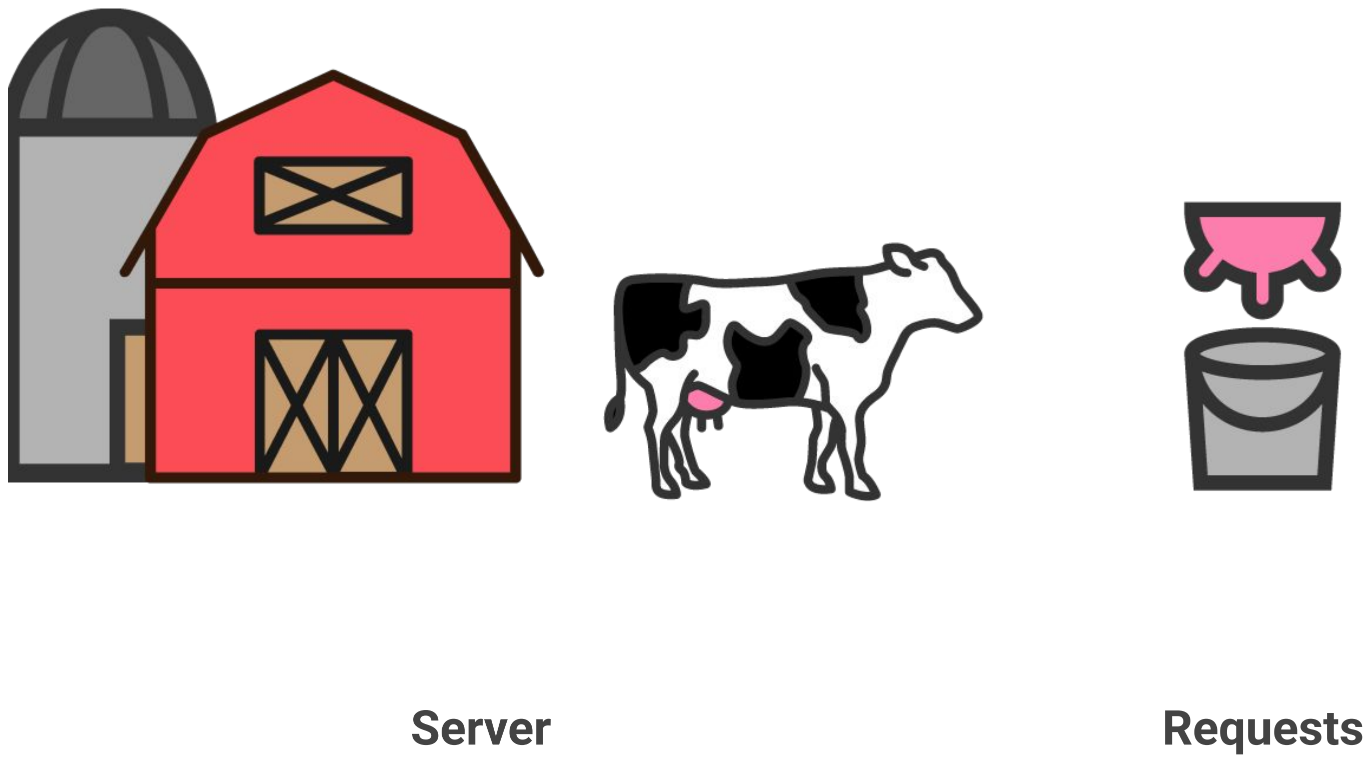
Reduce server load, improve app performance



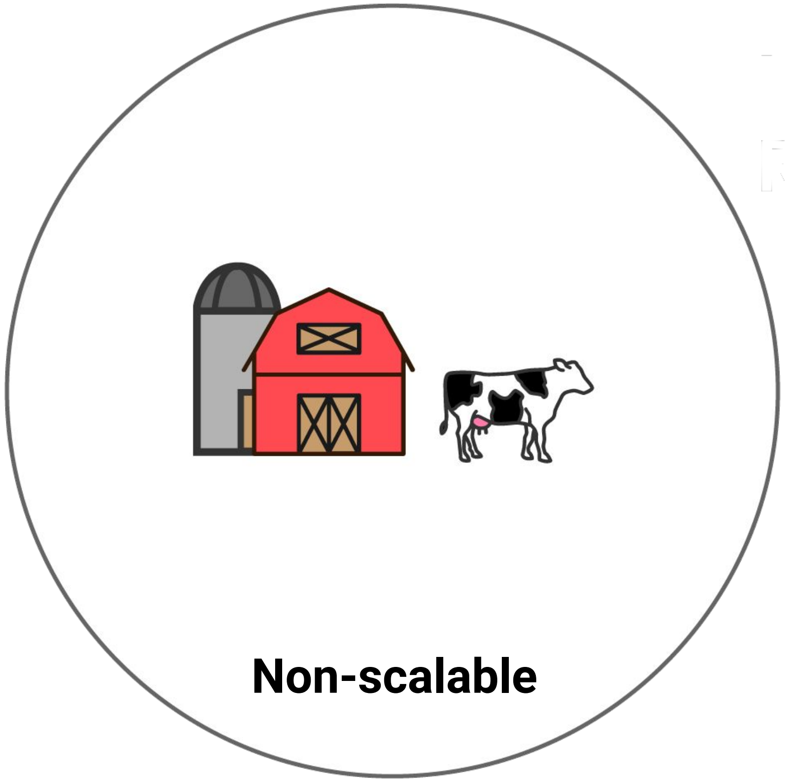
Fetch data once, read more than once



Buying milk at the supermarket



Source: http://bit.ly/2oC6L0Y

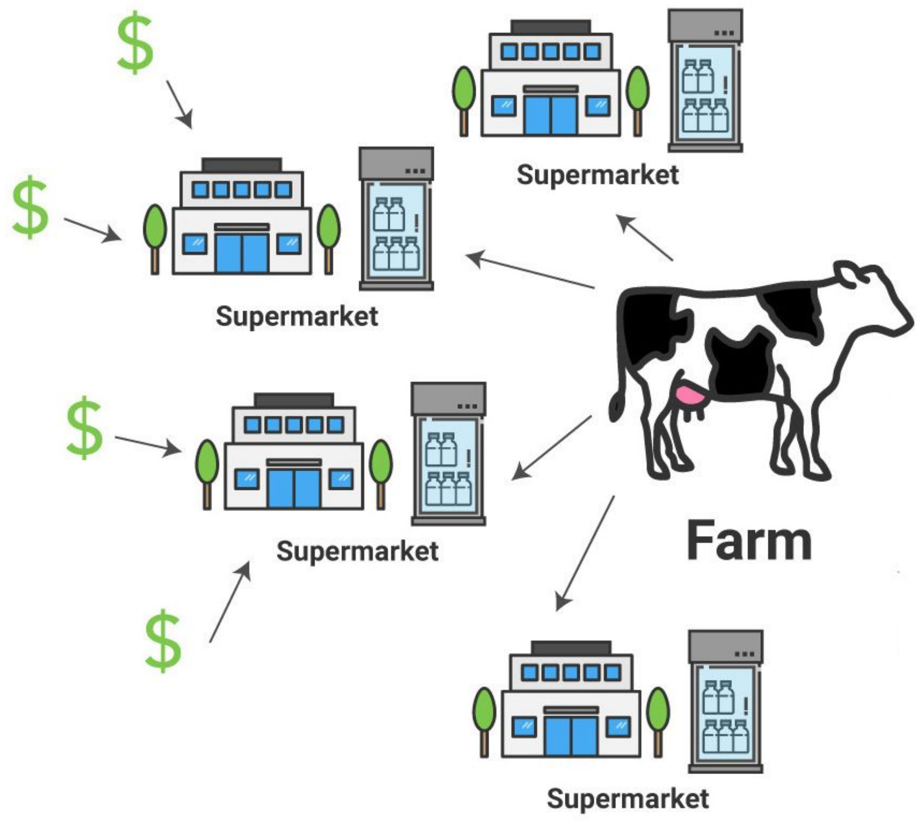


High demand

Limited supply of milk

Source: http://bit.ly/2oC6L0Y

Supermarkets store milk



Reliable, quick to access

Expiration

Scalable

Markets handle demand of consumers

Source: http://bit.ly/2oC6L0Y

Cache invalidation

Data changes without you knowing about it

Whether there is a change in data, no data, or new data

Cache needs to get updated

Cache invalidation

Data in cache is temporary

Cache needs to get updated or data removed

Market needs to know when milk is expired and to remove from shelves

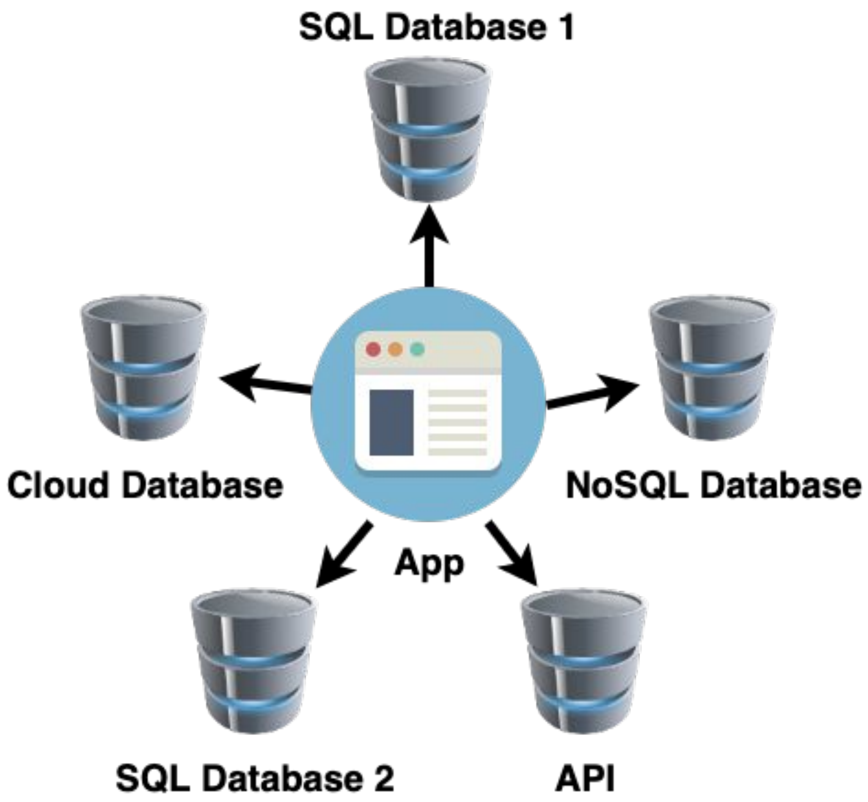
Why
are we caching and how can it help you with app performance?

The problem

Slow website and app performance

- multiple data sources

Single request requires data from different systems

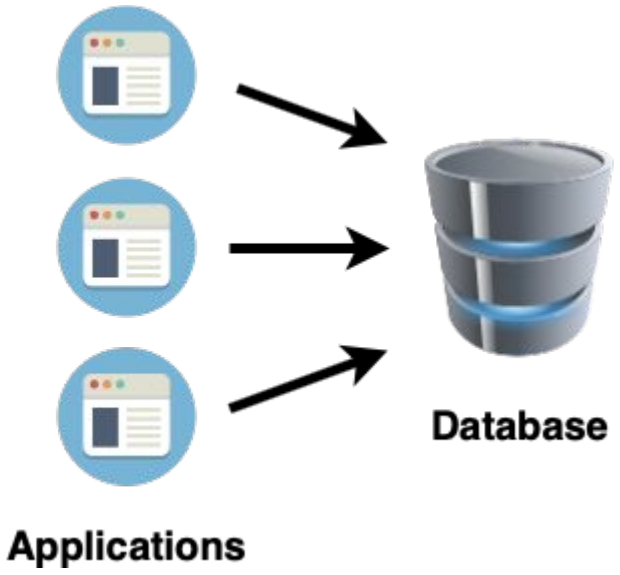


The problem

Multiple web applications

- accessing exact same data in different ways
- running similar queries at different times

High burden / load on databases



The problem

Third-party provider APIs

- rate limits
- slowness
- token issues

Errors in business systems and digital products

The ask

Build a cache for quick retrieval of data

Make it easier to build high performing web applications with fewer errors and quicker response times

The ask

Relieve SQL load on databases

Easier and more reliable path to data

Consolidated data, source of truth, consistent data across all applications

Where
are we caching data and how
can it be accessed?

Cache storage - Redis

Open source, in memory data structure store

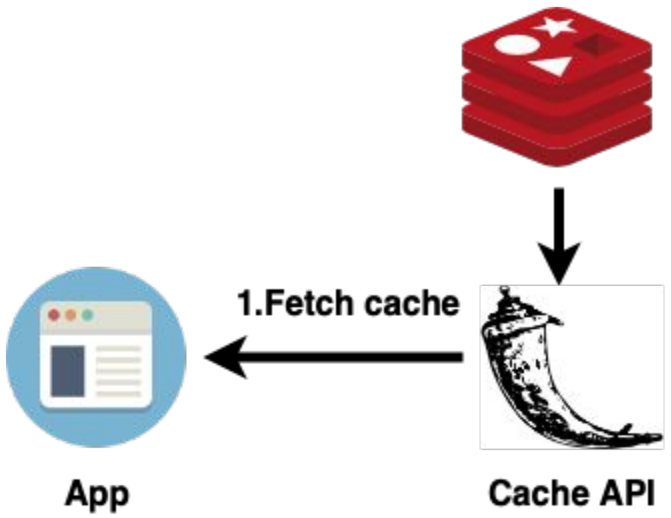
Built-in replication

Highly available

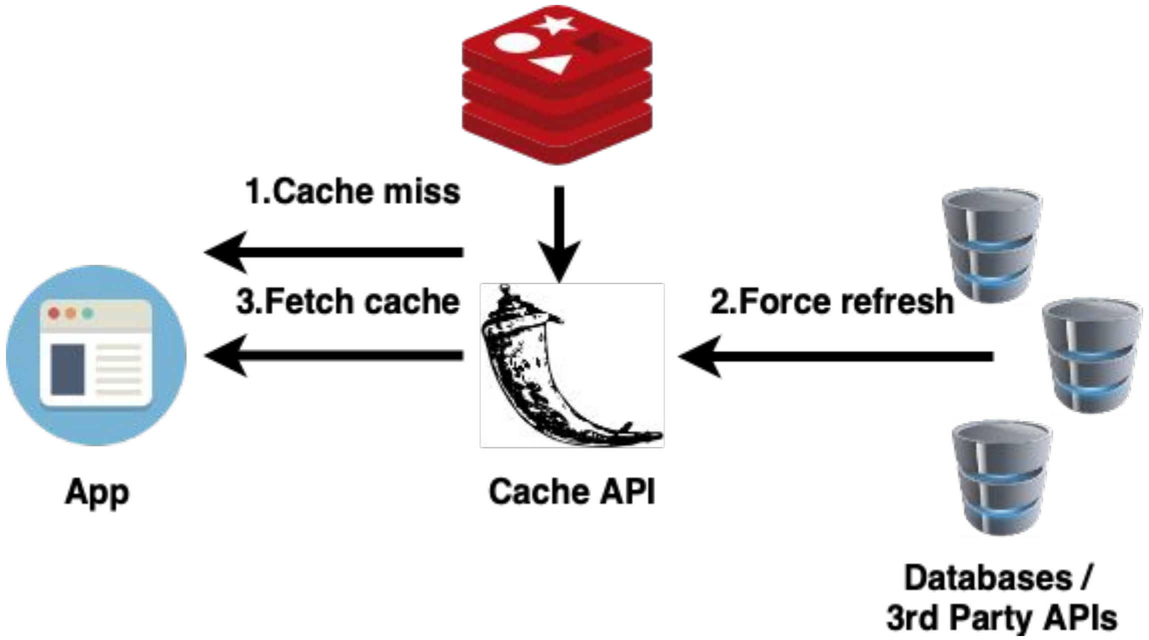
Fault tolerant

Highly scalable

Cache access - API - fetch data



Cache access - API - cache miss



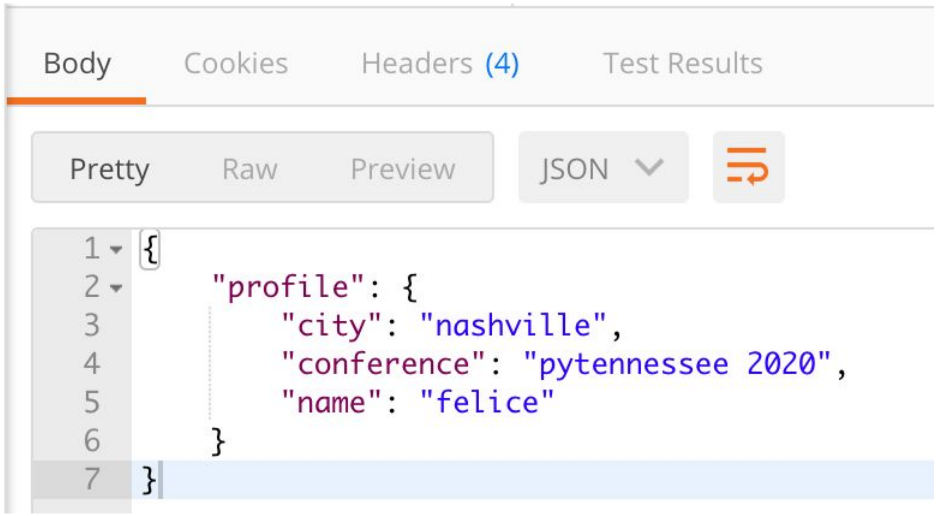
How
are we caching data?

Factors to consider - API contract

Agreement between service and client

Specifications on data and structure

-> need a JSON response string



```
{
    "profile": {
        "city": "nashville",
        "conference": "pytennessee 2020",
        "name": "felice"
    }
}
```

Factors to consider - API design

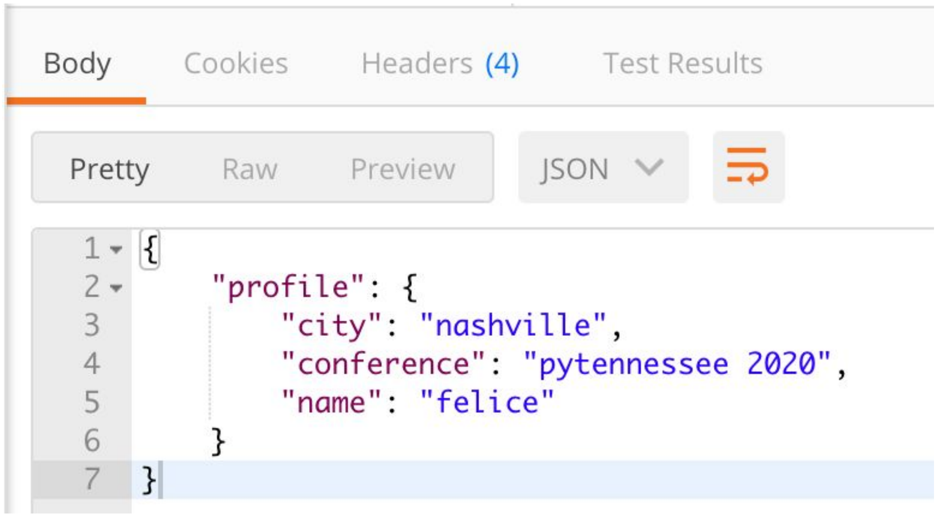
Dynamic or strict structure

- to not include or include null values

Factors to consider - API design

Dynamic structure

- removes noise, omission represents lack of value
- unclear if omissions mean unknown or truly no value

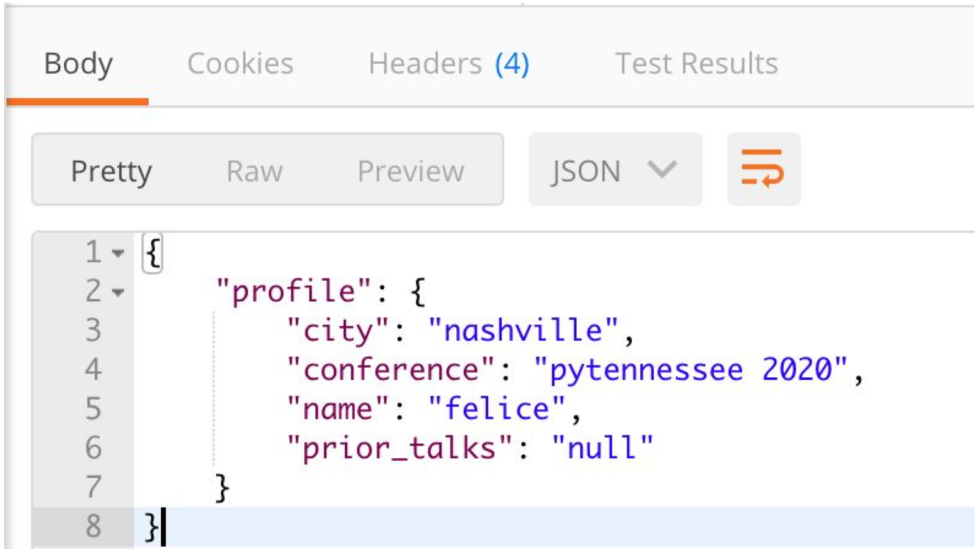


```
{
    "profile": {
        "city": "nashville",
        "conference": "pytennessee 2020",
        "name": "felice"
    }
}
```

Factors to consider - API design

Strict structure

- indicates existence of property even if there is no value
- would need to handle non-nullable fields



```
{
    "profile": {
        "city": "nashville",
        "conference": "pytennessee 2020",
        "name": "felice",
        "prior_talks": "null"
    }
}
```

Factors to consider - API design

Dynamic or strict structure

It depends…

sparse or dense data?

-> null values not included in API response

Factors to consider - the data itself

Transactional / point of sale

Web and application data

CRM

Data warehouse

-> update strategy needs to include all data sources

Factors to consider - update strategy

Cache warming

Time to live (TTL)

Cache miss functionality in API

-> need to ensure accurate and relevant data in cache

Caching options in Redis

```
# Python interface to the Redis key-value store.
# pip install redis
import redis
redis_conn = redis.Redis(
    host='localhost', port=6379, db=0
)
```

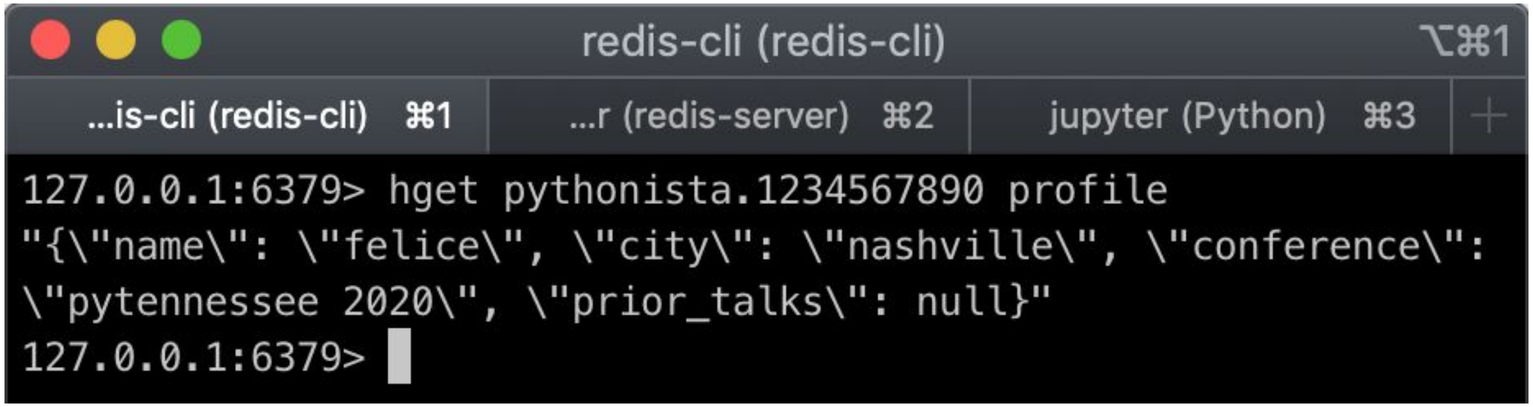
Caching options in Redis

```python
# Cache as json encoded string
import json

# hset pythonista.1234567890 profile '{"name": "felice ho", "city": "nashv
profile_dict = {
    "name": "felice",
    "city": "nashville",
    "conference":"pytennessee 2020",
    "prior_talks": None
}
profile_json = json.dumps(profile_dict)

redis_conn.hset(
    name='pythonista.1234567890',
    key='profile',
    value=profile_json
)
```

Caching options in Redis



```
127.0.0.1:6379> hget pythonista.1234567890 profile
"{\"name\": \"felice\", \"city\": \"nashville\", \"conference\":
\"pytennessee 2020\", \"prior_talks\": null}"
127.0.0.1:6379>
```

Caching options in Redis

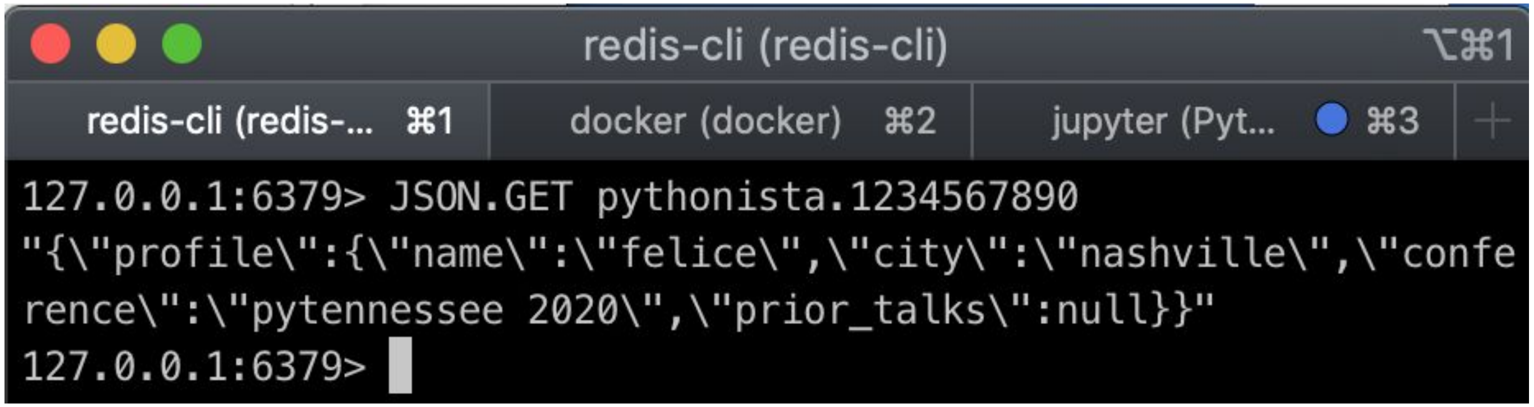
```python
# Cache via ReJSON: native JSON data type
# pip install rejson
from rejson import Client, Path

rj_conn = Client(
    host='localhost',
    port=6379,decode_responses=True
)

# JSON.SET pythonista.1234567890 . '{"profile": {"name": "felice ho", "cit
profile_dict = {
    "name": "felice",
    "city": "nashville",
    "conference":"pytennessee 2020",
    "prior_talks": None
}
obj = {"profile": profile_dict}

rj_conn.jsonset(
    'pythonista.1234567890', Path.rootPath(), obj
)
```

Caching options in Redis



```
127.0.0.1:6379> JSON.GET pythonista.1234567890
"{\"profile\":{\"name\":\"felice\",\"city\":\"nashville\",\"confe
rence\":\"pytennessee 2020\",\"prior_talks\":null}}"
127.0.0.1:6379>
```

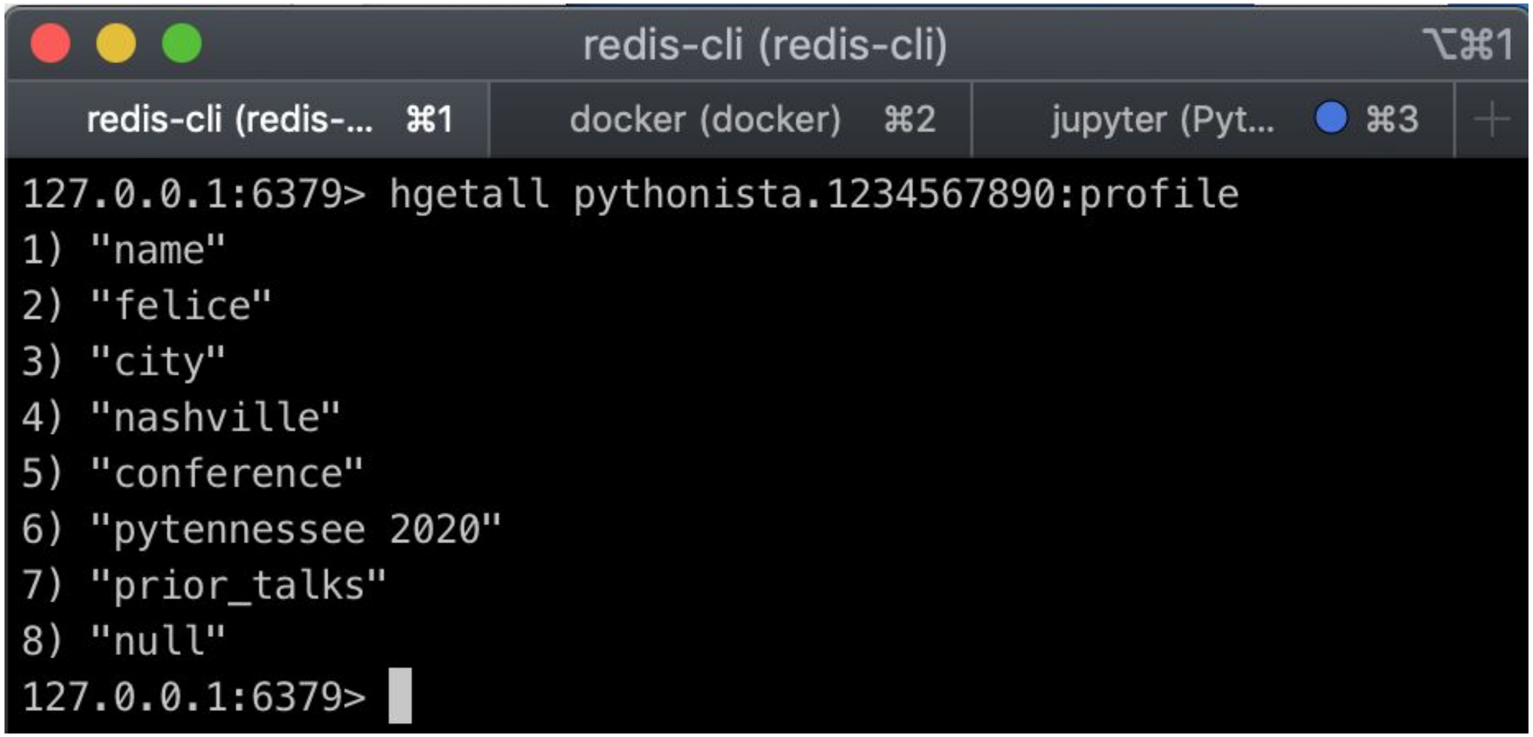
Caching options in Redis

```
# Cache as hashes, Redis key/value pairs
# hmset pythonista.1234567890:profile name 'felice ho' city 'nashville' co
profile_dict = {
    "name": "felice",
    "city": "nashville",
    "conference":"pytennessee 2020",
    "prior_talks": "null"
}

redis_conn.hmset(
    name='pythonista.1234567890:profile',
    mapping=profile_dict
)
```

Note: Starting with redis-py 3.0, `None` is no longer accepted as input for keys or values. Same for `True` or `False`. Users will need to cast these values explicitly before sending them to redis-py.
Source: https://github.com/andymccurdy/redis-py/issues/190

Caching options in Redis



```
127.0.0.1:6379> hgetall pythonista.1234567890:profile
1) "name"
2) "felice"
3) "city"
4) "nashville"
5) "conference"
6) "pytennessee 2020"
7) "prior_talks"
8) "null"
127.0.0.1:6379>
```

Caching strategy

JSON string vs. hashes

- no notable performance difference
- hashes slightly faster with help of Lua and cjson

The problem with ‘nothing’

Representing nothing with something

Keep placeholder value for keys even if null

Recognize data changed from existing to not existing

Else appears as if something exists, when it doesn’t causing invalid data in cache

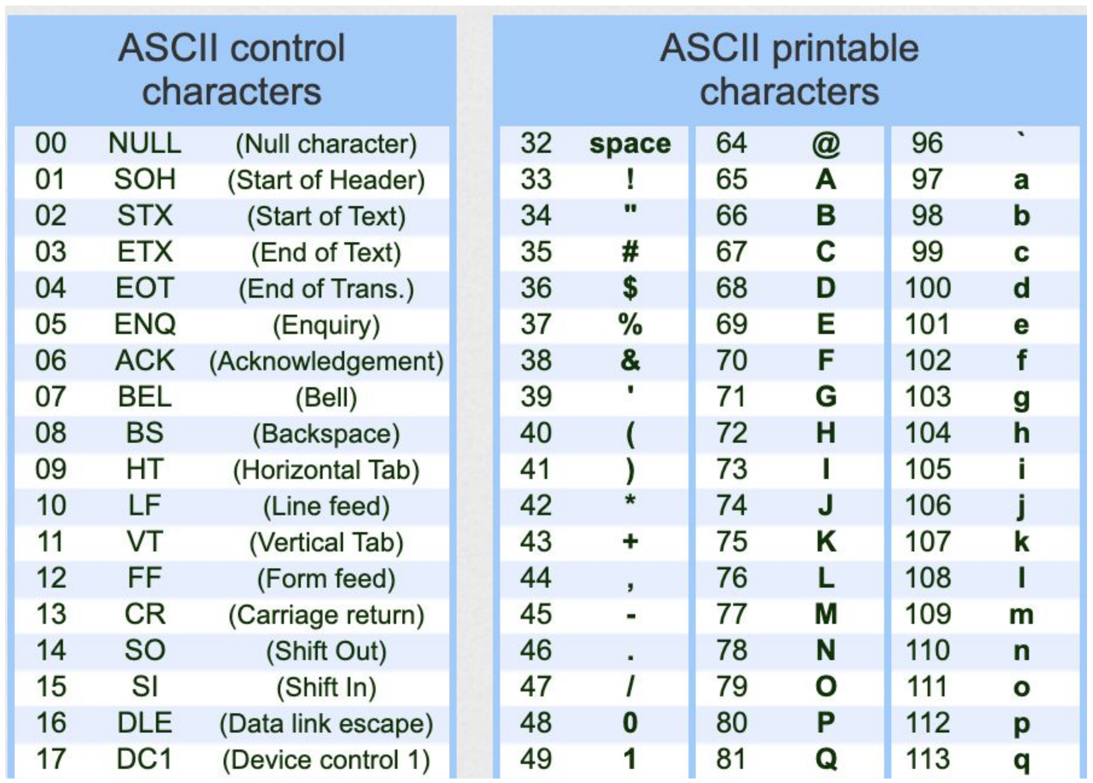
What is null?

Value assigned to a variable to represent

- no value / non value
- neutral behavior
- absence of data / useful value
- nothing

What is null?

Represented with zeros but not same value as zero



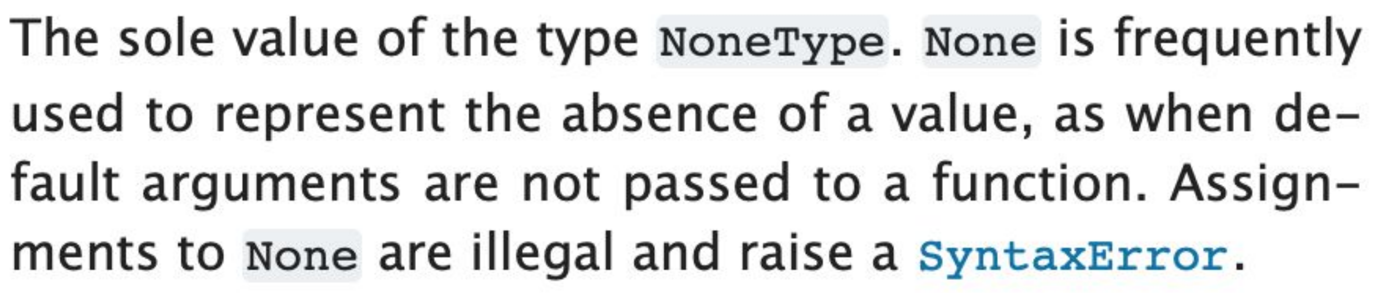
ASCII control characters		
00	NULL	(Null character)
01	SOH	(Start of Header)
02	STX	(Start of Text)
03	ETX	(End of Text)
04	EOT	(End of Trans.)
05	ENQ	(Enquiry)
06	ACK	(Acknowledgement)
07	BEL	(Bell)
08	BS	(Backspace)
09	HT	(Horizontal Tab)
10	LF	(Line feed)
11	VT	(Vertical Tab)
12	FF	(Form feed)
13	CR	(Carriage return)
14	SO	(Shift Out)
15	SI	(Shift In)
16	DLE	(Data link escape)
17	DC1	(Device control 1)

ASCII printable characters					
32	space	64	@	96	`
33	!	65	A	97	a
34	"	66	B	98	b
35	#	67	C	99	c
36	$	68	D	100	d
37	%	69	E	101	e
38	&	70	F	102	f
39	'	71	G	103	g
40	(	72	H	104	h
41	)	73	I	105	i
42	*	74	J	106	j
43	+	75	K	107	k
44	,	76	L	108	l
45	-	77	M	109	m
46	.	78	N	110	n
47	/	79	O	111	o
48	0	80	P	112	p
49	1	81	Q	113	q

What is the problem here?

Serialization of null is represented differently

Python:

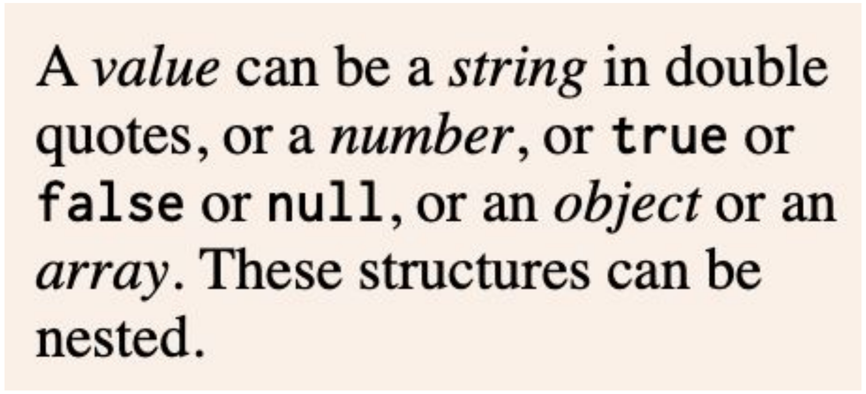

The sole value of the type `NoneType`. `None` is frequently used to represent the absence of a value, as when default arguments are not passed to a function. Assignments to `None` are illegal and raise a `SyntaxError`.

Source: https://docs.python.org/3/library/constants.html

What is the problem here?

Serialization of null is represented differently

JSON:

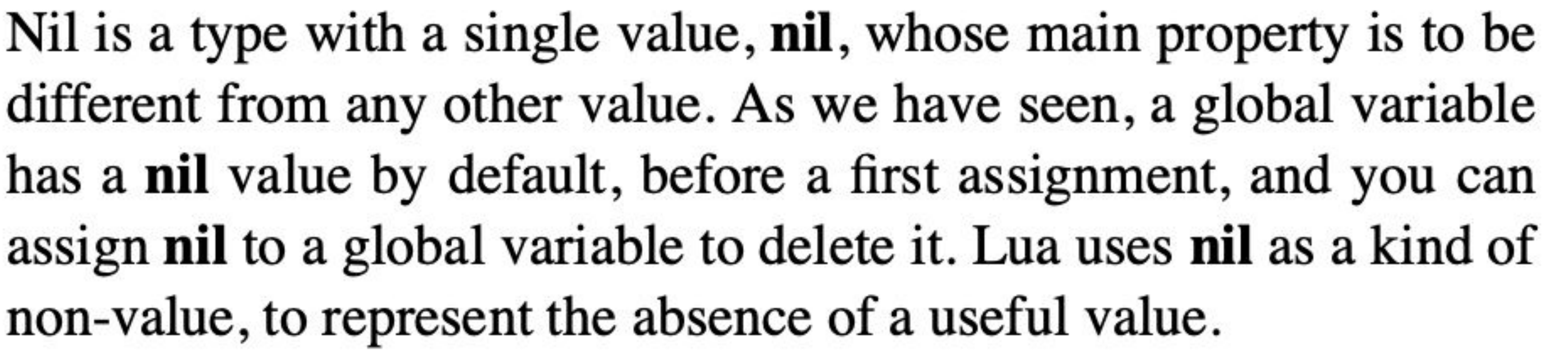

A *value* can be a *string* in double quotes, or a *number*, or `true` or `false` or `null`, or an *object* or an *array*. These structures can be nested.

Source: https://www.json.org

What is the problem here?

Serialization of null is represented differently

Lua:



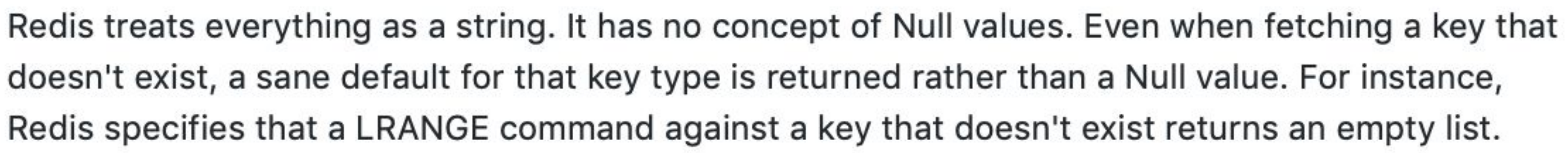
Nil is a type with a single value, **nil**, whose main property is to be different from any other value. As we have seen, a global variable has a **nil** value by default, before a first assignment, and you can assign **nil** to a global variable to delete it. Lua uses **nil** as a kind of non-value, to represent the absence of a useful value.

Source: http://www.lua.org/pil/2.1.html

What is the problem here?

Serialization of null is represented differently

Redis: it’s not possible!



Redis treats everything as a string. It has no concept of Null values. Even when fetching a key that doesn't exist, a sane default for that key type is returned rather than a Null value. For instance, Redis specifies that a LRANGE command against a key that doesn't exist returns an empty list.

Source: https://github.com/andymccurdy/redis-py/issues/190

What is the problem here?

It is up to the language or library to determine how to represent null

Storing null in Redis

Store as empty string

- is value actually an empty string or null

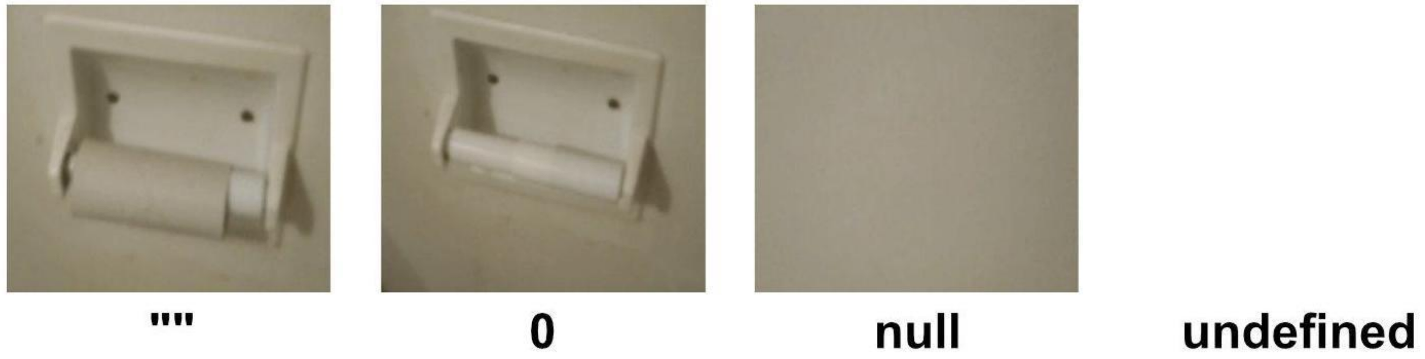


Photo credit: http://bit.ly/2n8qK7c

Storing null in Redis

Use sentinel value to represent null

- "null" not same as null

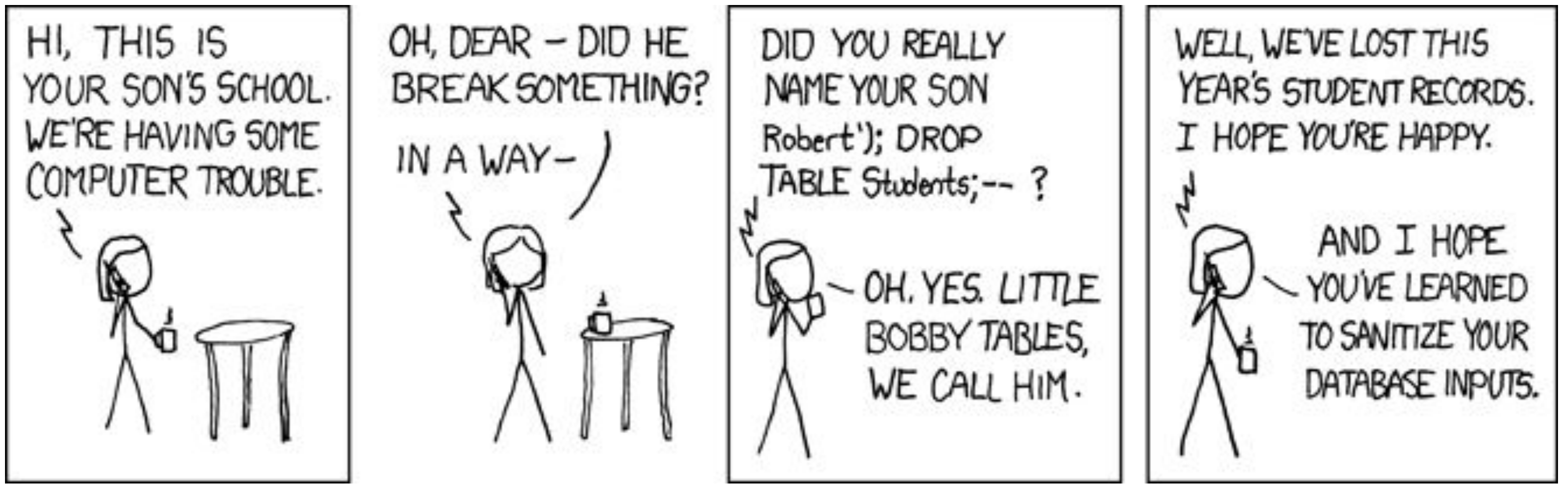

HI, THIS IS YOUR SON'S SCHOOL. WE'RE HAVING SOME COMPUTER TROUBLE.
OH, DEAR – DID HE BREAK SOMETHING?
IN A WAY–
DID YOU REALLY NAME YOUR SON Robert'); DROP TABLE Students;-- ?
OH. YES. LITTLE BOBBY TABLES, WE CALL HIM.
WELL, WE'VE LOST THIS YEAR'S STUDENT RECORDS. I HOPE YOU'RE HAPPY.
AND I HOPE YOU'VE LEARNED TO SANITIZE YOUR DATABASE INPUTS.

Photo credit: https://xkcd.com/327/

Trust your encoder

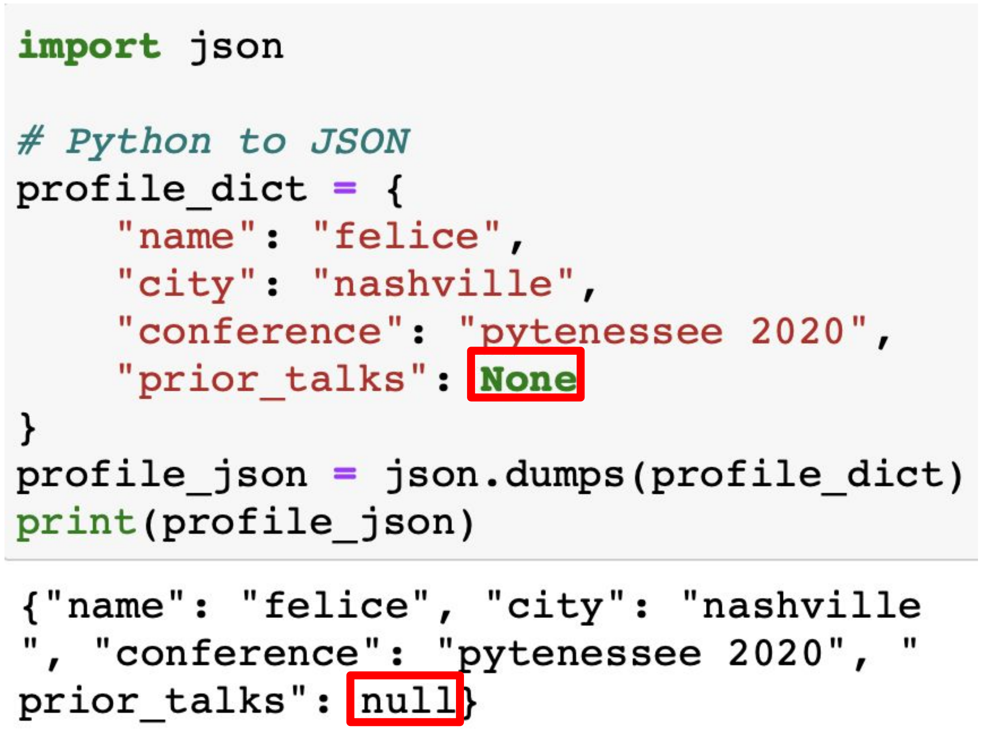
Encoder will serialize into what you need

Keep null values in native format before encoding

None **null** **nil** **(your choice)**

Trust your encoder

Mapping key/pair values from Python dictionary to JSON string



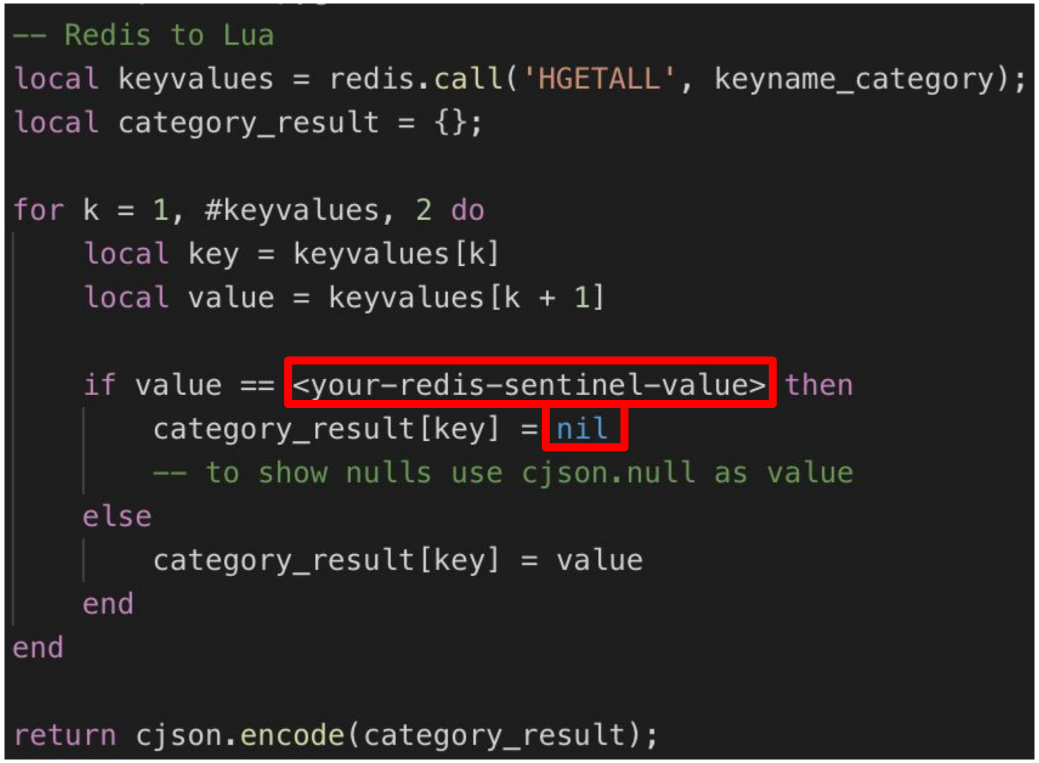
```
import json

# Python to JSON
profile_dict = {
    "name": "felice",
    "city": "nashville",
    "conference": "pytenessee 2020",
    "prior_talks": None
}
profile_json = json.dumps(profile_dict)
print(profile_json)
```

```
{"name": "felice", "city": "nashville
", "conference": "pytenessee 2020", "
prior_talks": null}
```

Trust your encoder

Mapping key/pair values from Redis using Lua to encode to JSON string



```
-- Redis to Lua
local keyvalues = redis.call('HGETALL', keyname_category);
local category_result = {};

for k = 1, #keyvalues, 2 do
    local key = keyvalues[k]
    local value = keyvalues[k + 1]

    if value == <your-redis-sentinel-value> then
        category_result[key] = nil
        -- to show nulls use cjson.null as value
    else
        category_result[key] = value
    end
end

return cjson.encode(category_result);
```

Encoding vs. Serialization

Encoder

Helps with converting data into a certain representation, from one format into another

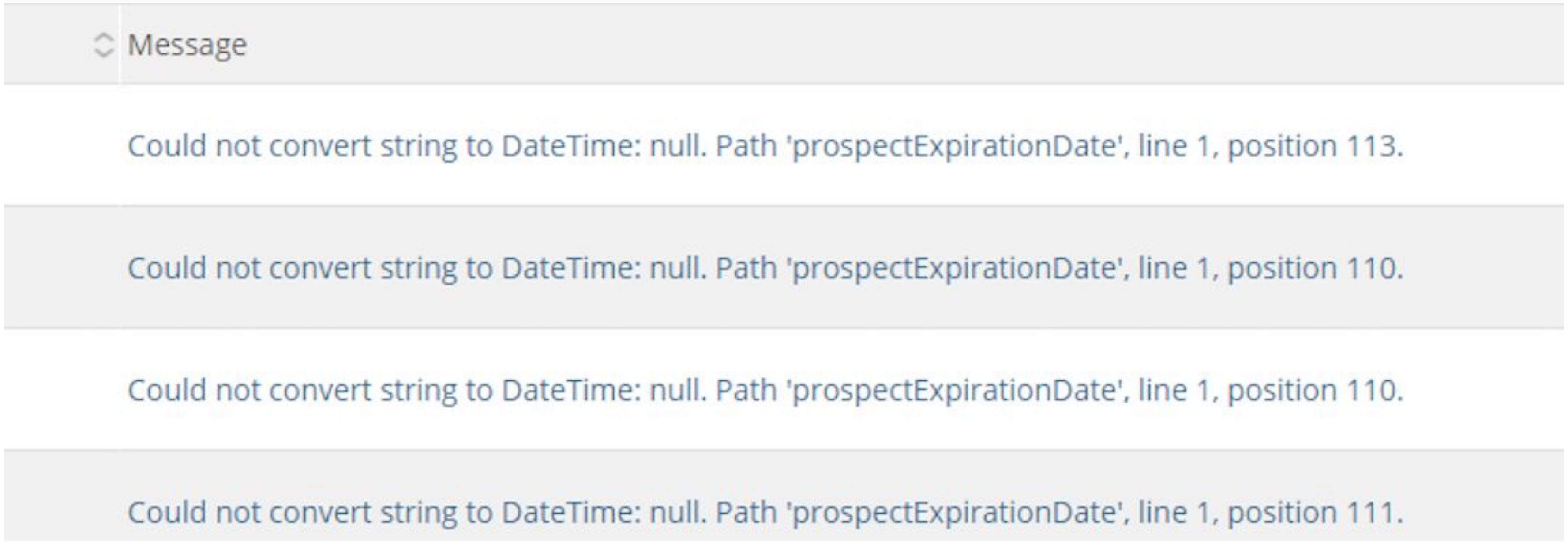
-> Python to JSON

Serialization

Process of translating an object into a format that can be stored or transmitted, and reconstructed later

-> JSON is a serialization format for client server communication

‘We have a problem’



Message
Could not convert string to DateTime: null. Path 'prospectExpirationDate', line 1, position 113.
Could not convert string to DateTime: null. Path 'prospectExpirationDate', line 1, position 110.
Could not convert string to DateTime: null. Path 'prospectExpirationDate', line 1, position 110.
Could not convert string to DateTime: null. Path 'prospectExpirationDate', line 1, position 111.

Storing null in Redis

Redis treats everything as a string. It has no concept of Null values. Even when fetching a key that doesn't exist, a sane default for that key type is returned rather than a Null value. For instance, Redis specifies that a LRANGE command against a key that doesn't exist returns an empty list.

“null” not same as null

Root cause analysis

Photo credit: Abbott and Costello

Lessons

Represent nothing with something

Nothing is recognized differently

Handle non-nullable values appropriately

Be aware of how your source data systems and tools handle null values

Left join (important) data

Include records

- with values at one point
- that matter if they no longer have values
- or otherwise not removed via TTL

Trust your encoder

Serialize ‘nothing’ in native form, no matter which language, tool, or format you are using

The value of ‘nothing’

“I call it my billion-dollar mistake. It was the invention of the null reference in 1965. … I couldn’t resist the temptation to put in a null reference, simply because it was so easy to implement. This has led to innumerable errors, vulnerabilities, and system crashes, which have probably caused a billion dollars of pain and damage in the last forty years.”

- Tony Hoare
  2009

Embracing null

Useful for cache invalidation

Web applications

- reduced errors, quicker response times

Databases

- reduced SQL load

Thank you!

Slides:
http://bit.ly/pytn-nonenullnil

Blog post:
http://bit.ly/globey-part-one

LinkedIn:
in /feliceho/