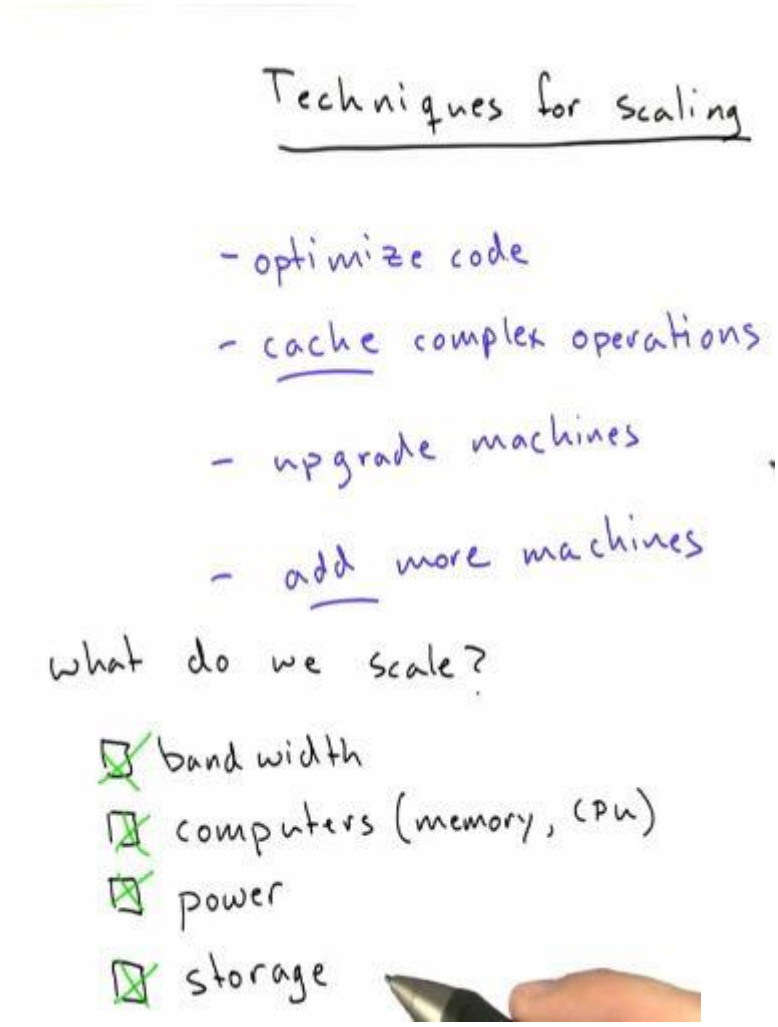
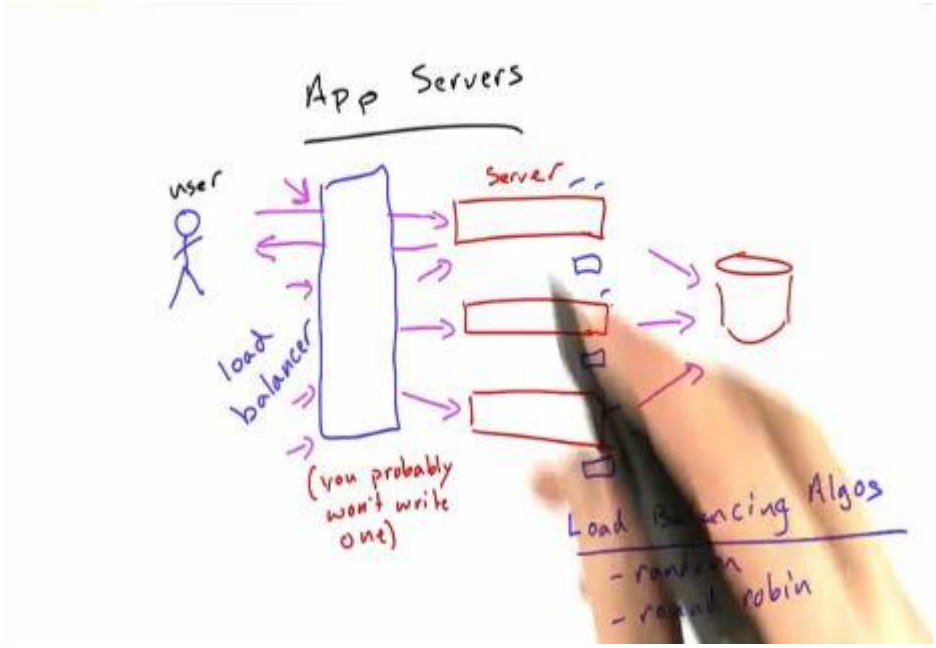
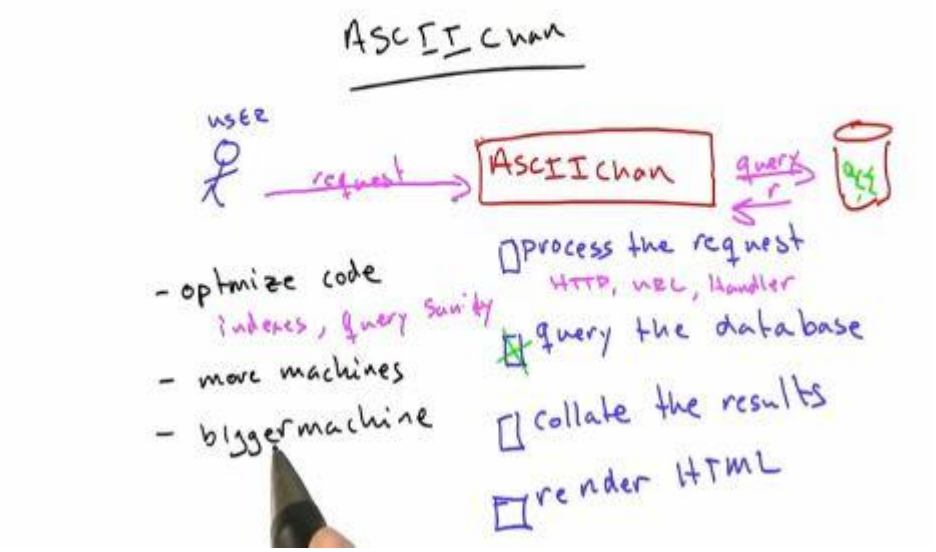
Techniques for Scaling

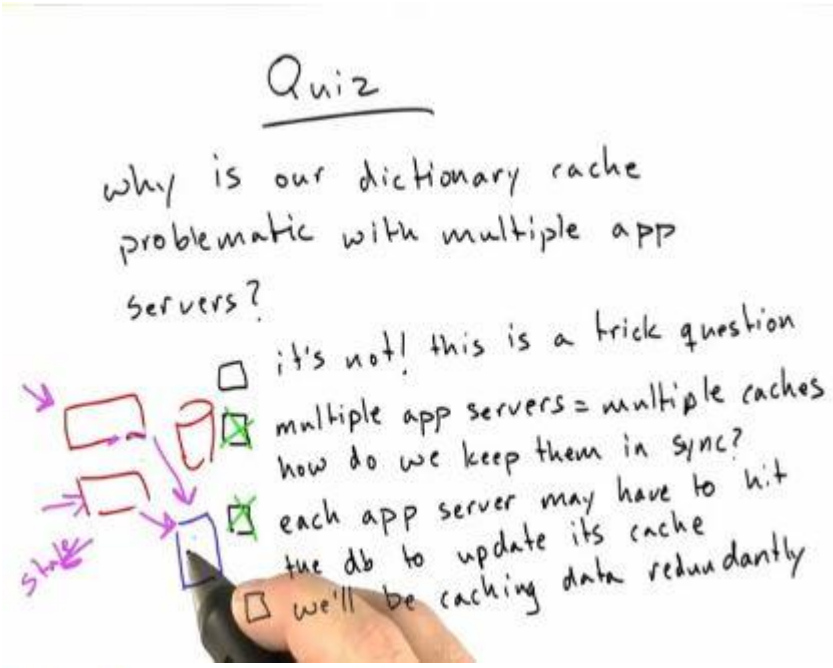


App Server Scaling

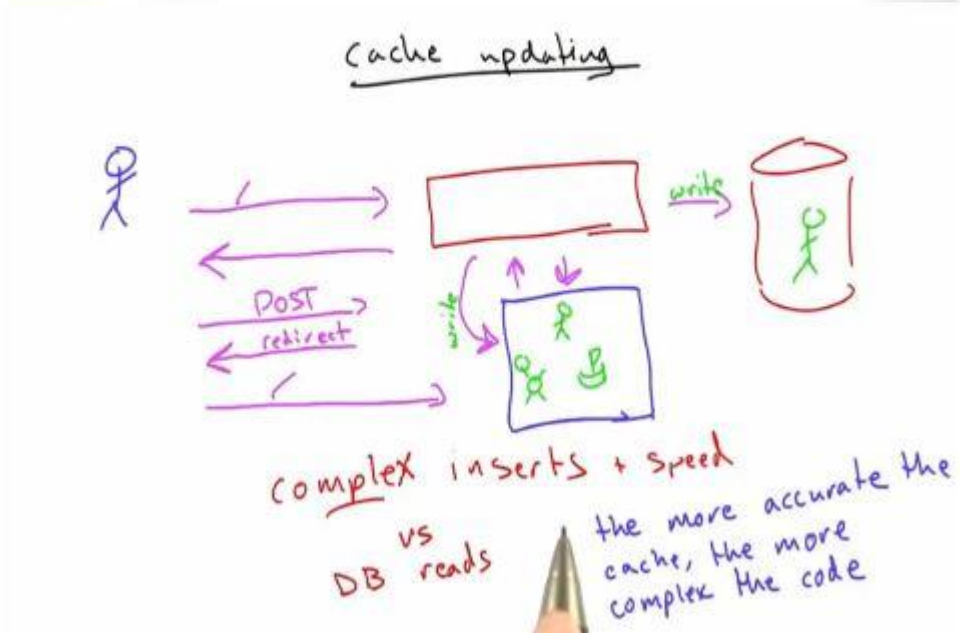


Optimizing Queries

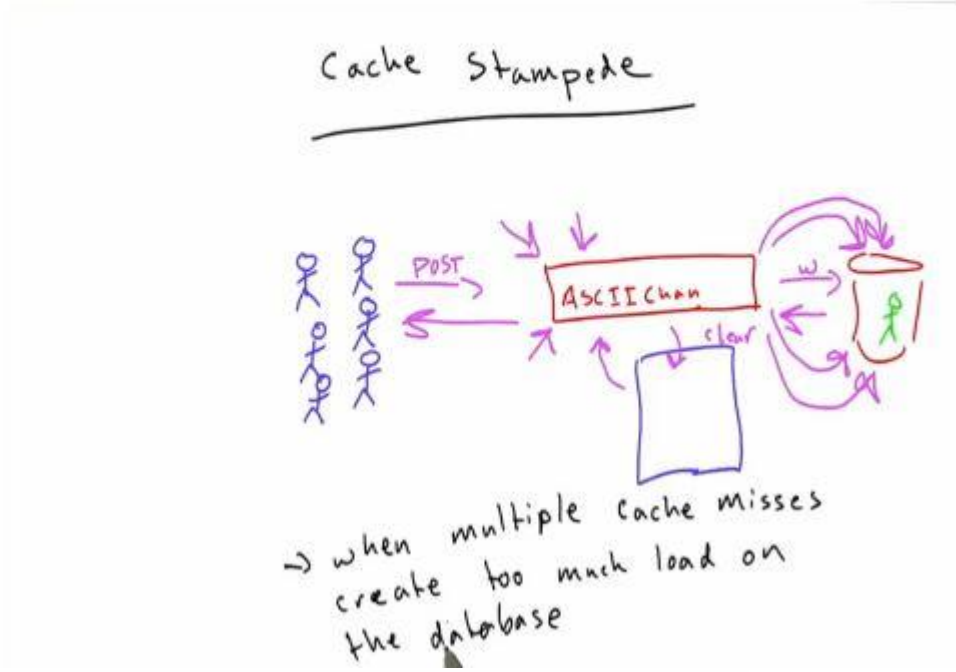




Caching Techniques



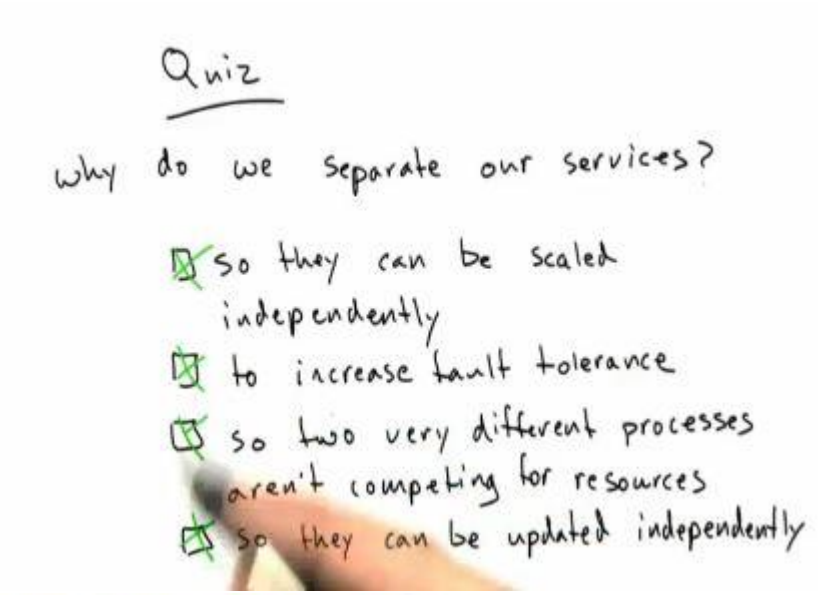
Cache Stampede



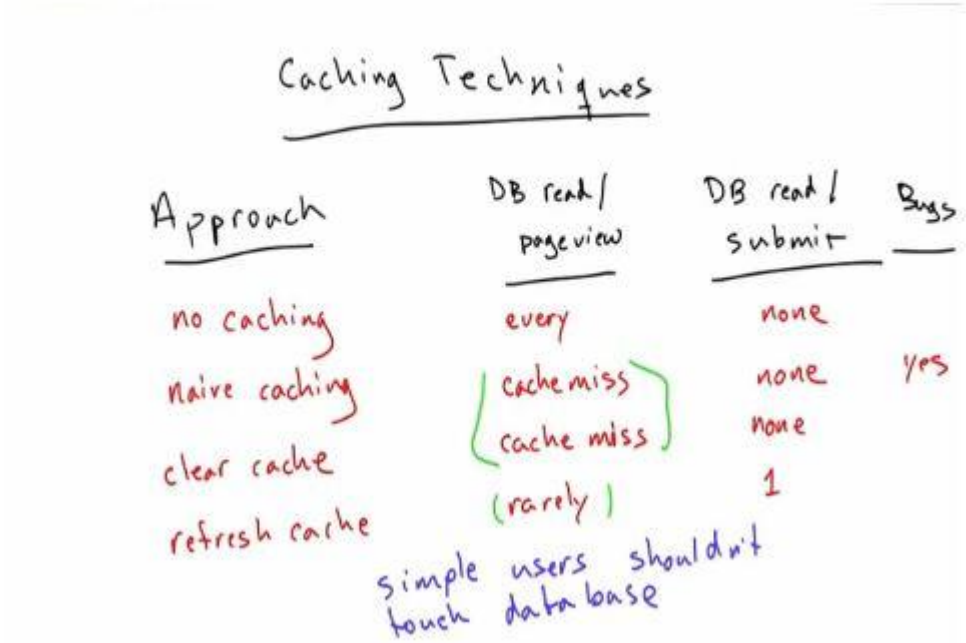
Quiz

How can we avoid a cache stampede?

- [ ] replicate the db to handle more reads
- [ ] only allow one web request at a time
- [ ] only allow one db request at a time
- [x] don't clear the cache, but instead overwrite with new data



Caching Techniques



Quiz

what happens when you store more data in memcached than there is memory available?

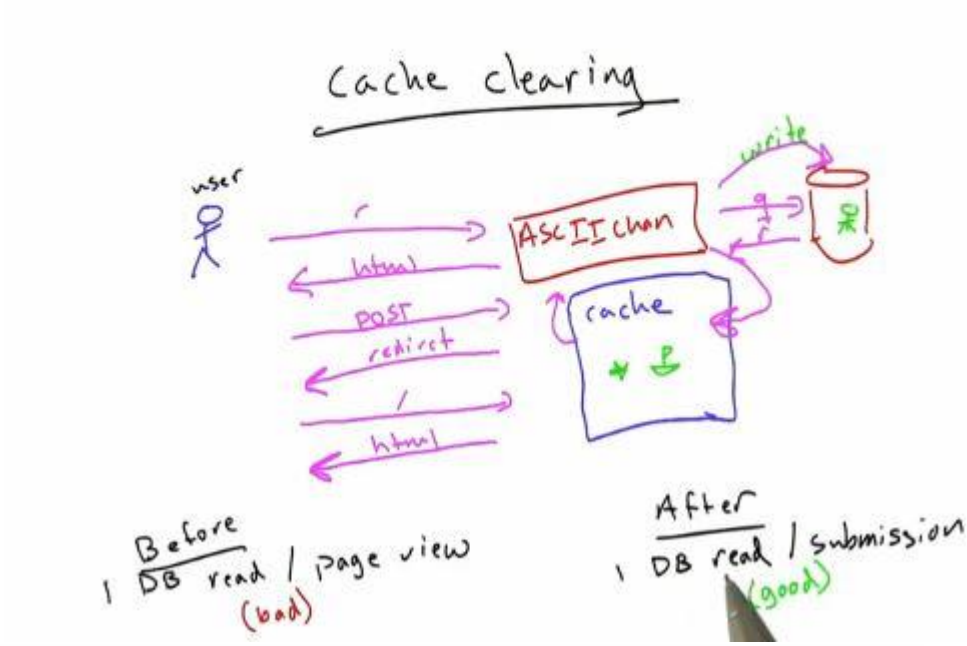
- [ ] error
- [ ] throw away data that is least frequently used LFU
- [x] throw away data that is least recently used LRU
- [ ] write the extra data to disk

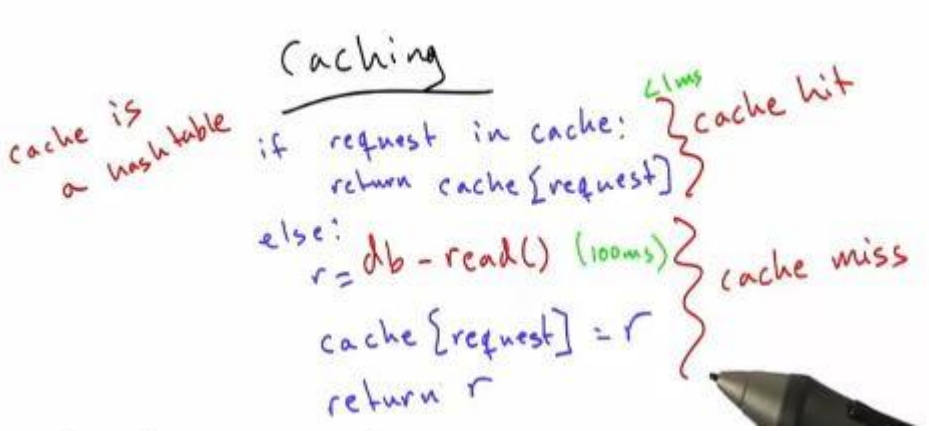
Quiz

How can we fix our stale cache problem?

- [ ] improve the cache to automatically expire after some time
- [x] after submitting, clear the cache
- [x] after submitting, update the cache
- [ ] don't cache, find a different approach

Cache Clearing

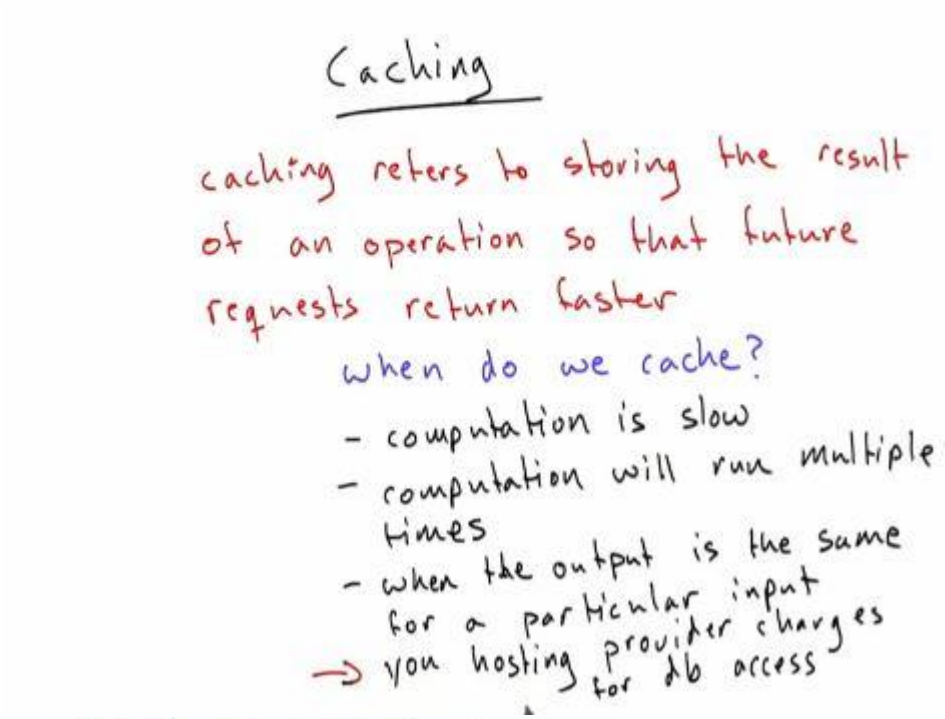




why do we scale?

- [x] So we can serve more requests concurrently
- [x] So we can store more data
- [x] So we're more resilient to failure
- [x] So we can serve requests faster

Caching




```
# database, or a request to another web service.
def complex_computation(a, b):
    time.sleep(.5)
    return a + b


# QUIZ - Improve the cached_computation() function below so that it caches
# results after computing them for the first time so future calls are faster
cache = {}
def cached_computation(a, b):
    key = (a,b)
    if key in cache:
        r = cache[key]
    else:
        r = complex_computation(a, b)
        cache[key] = r
    return r

start_time = time.time()
print cached_computation(5, 3)
print "the first computation took %f seconds" % (time.time() - start_time)
```

RUN

```
8
the first computation took 0.500199 seconds
```

Broken Submissions