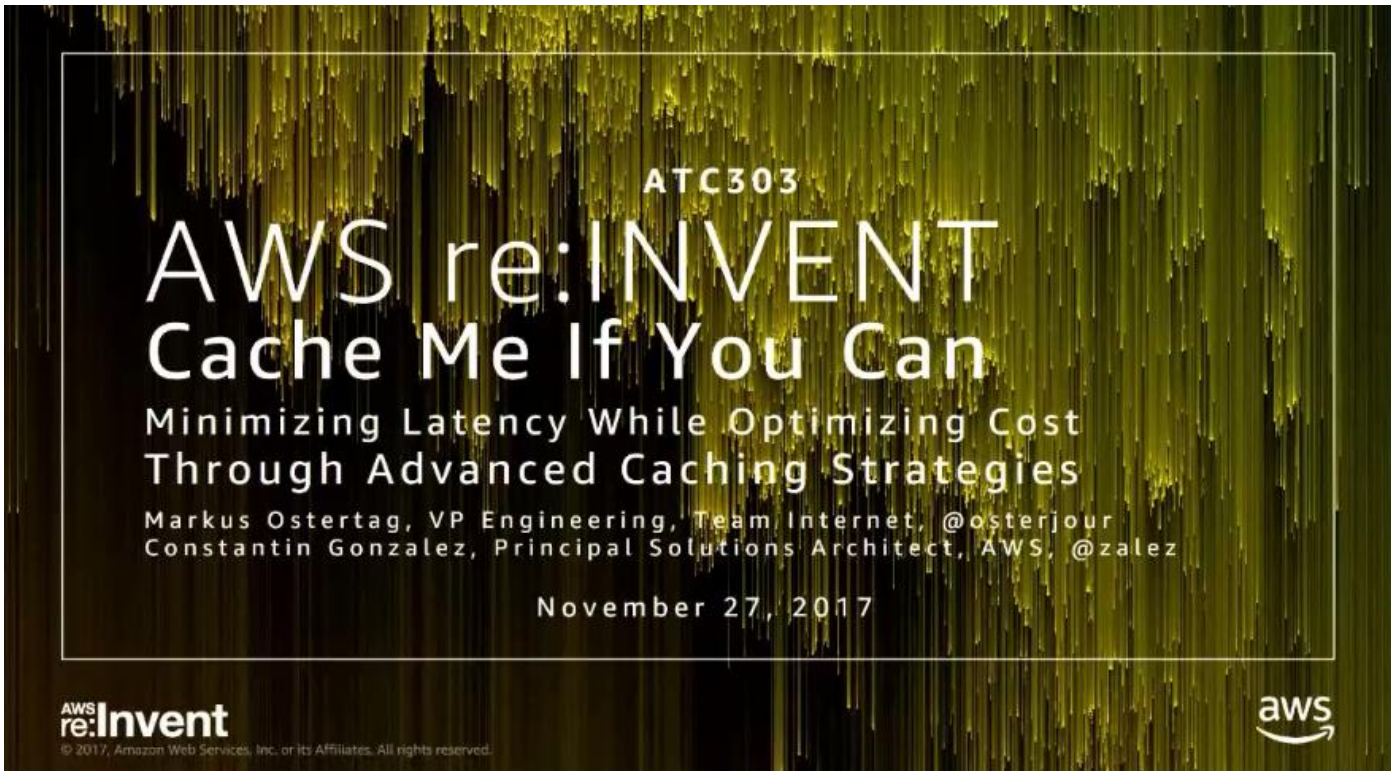


ATC303

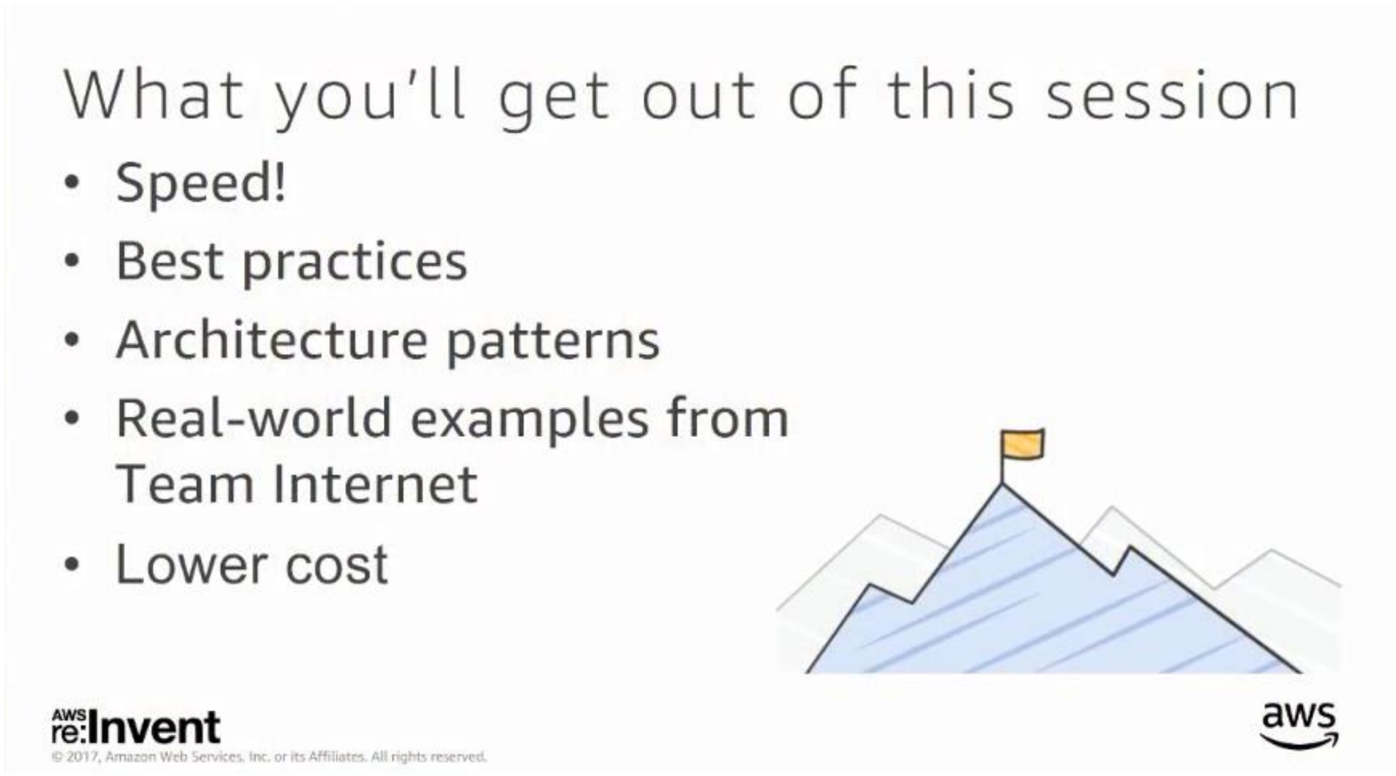
AWS re:INVENT
Cache Me If You Can

Minimizing Latency While Optimizing Cost Through Advanced Caching Strategies

Markus Ostertag, VP Engineering, Team Internet, @osterjour
Constantin Gonzalez, Principal Solutions Architect, AWS, @zalez

November 27, 2017

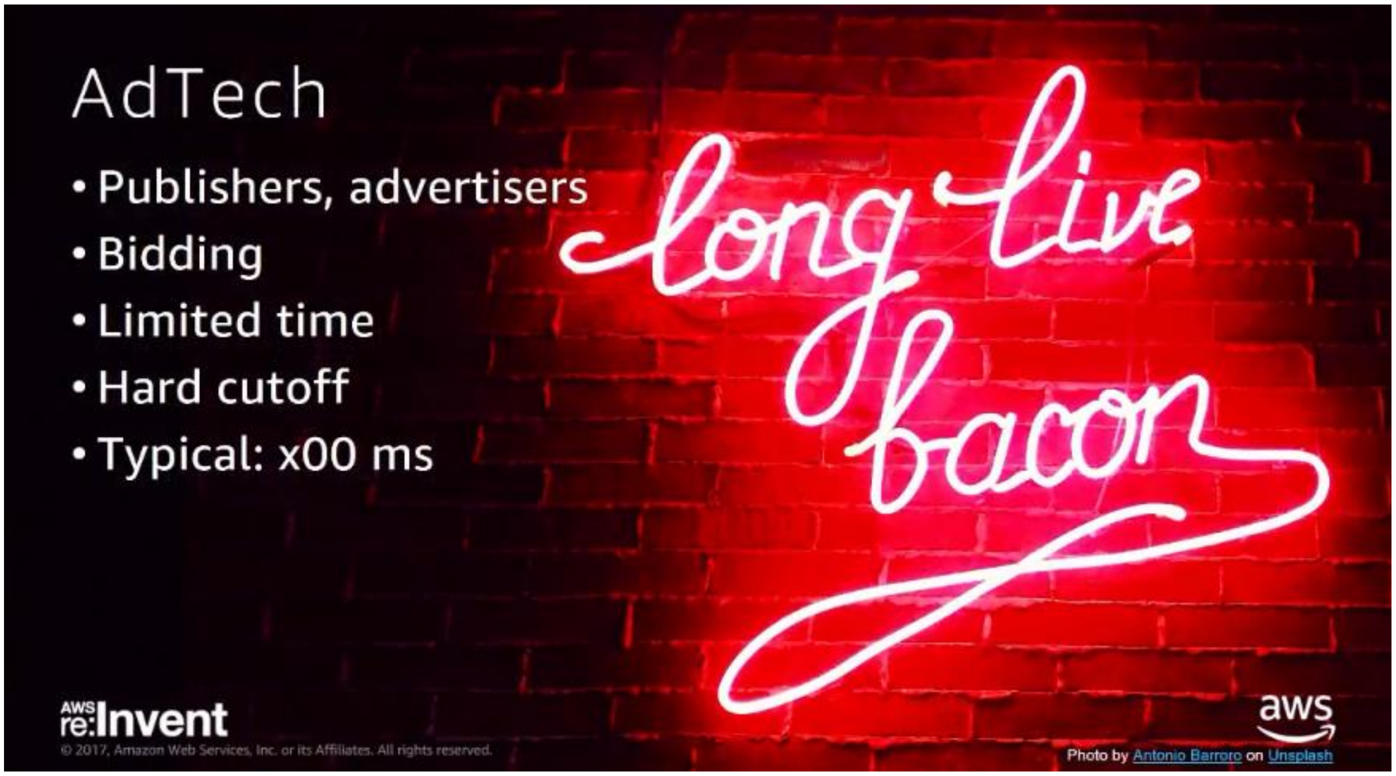
From CloudFront to ElastiCache to DynamoDB Accelerator (DAX), this is your one-stop shop for learning how to apply caching methods to your AdTech workload: What data to cache and why? What are common side effects and pitfalls when caching? What is negative caching and how can it help you maximize your cache hit rate? How to use DynamoDB Accelerator in practice? How can you ensure that data always stays current in your cache? These and many more topics will be discussed in depth during this talk and we’ll share lessons learned from Team Internet, the leading provider in domain monetization.



What you’ll get out of this session

- Speed!
- Best practices
- Architecture patterns
- Real-world examples from Team Internet
- Lower cost

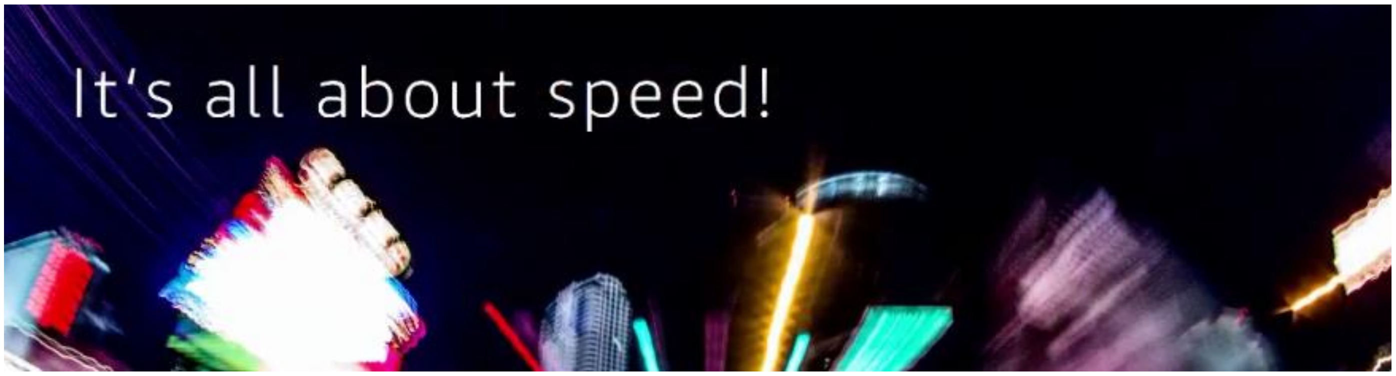
This session is all about getting more speed into your applications.



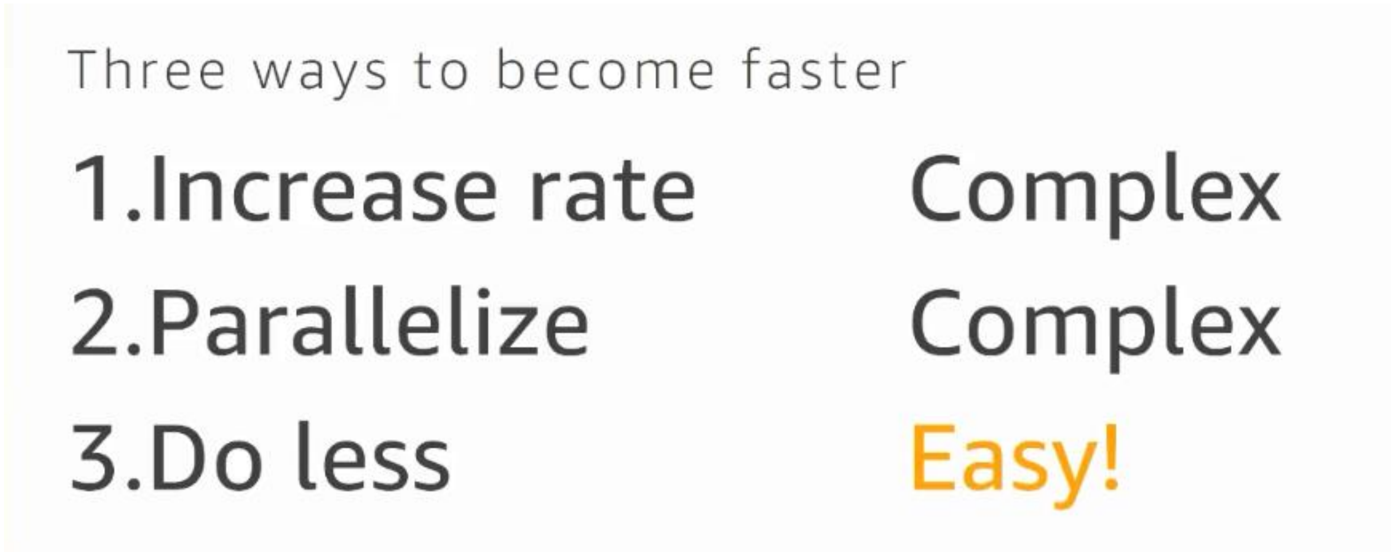
AdTech

- Publishers, advertisers
- Bidding
- Limited time
- Hard cutoff
- Typical: x00 ms

long live bacon



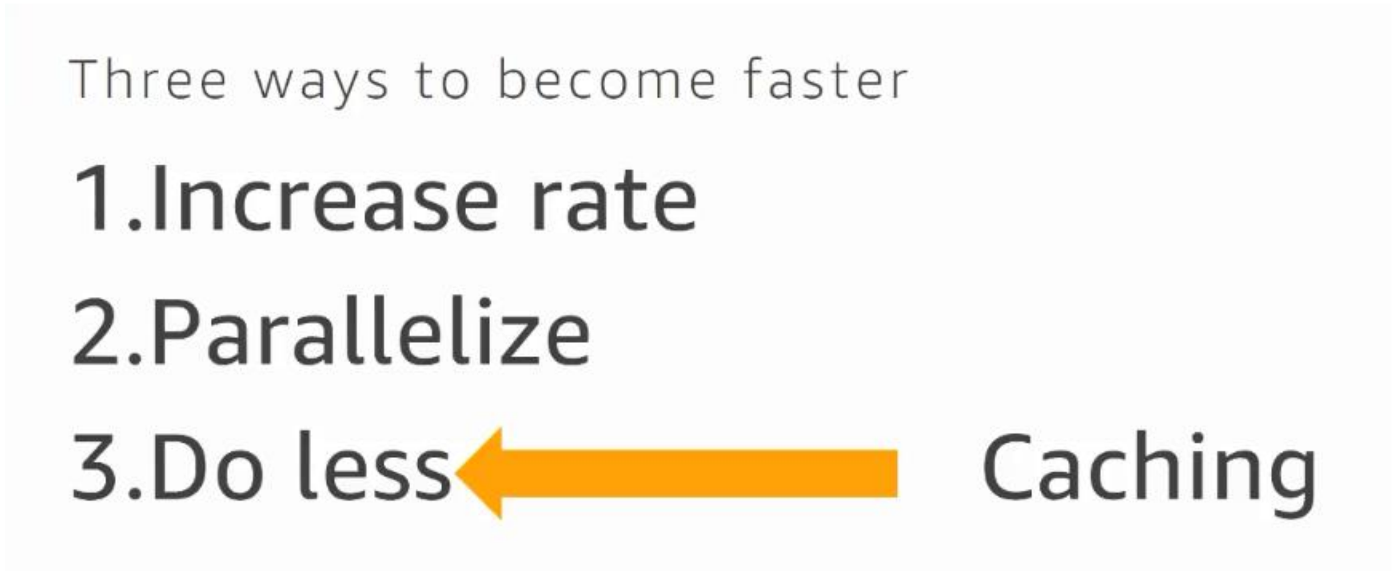
It's all about speed!



Three ways to become faster

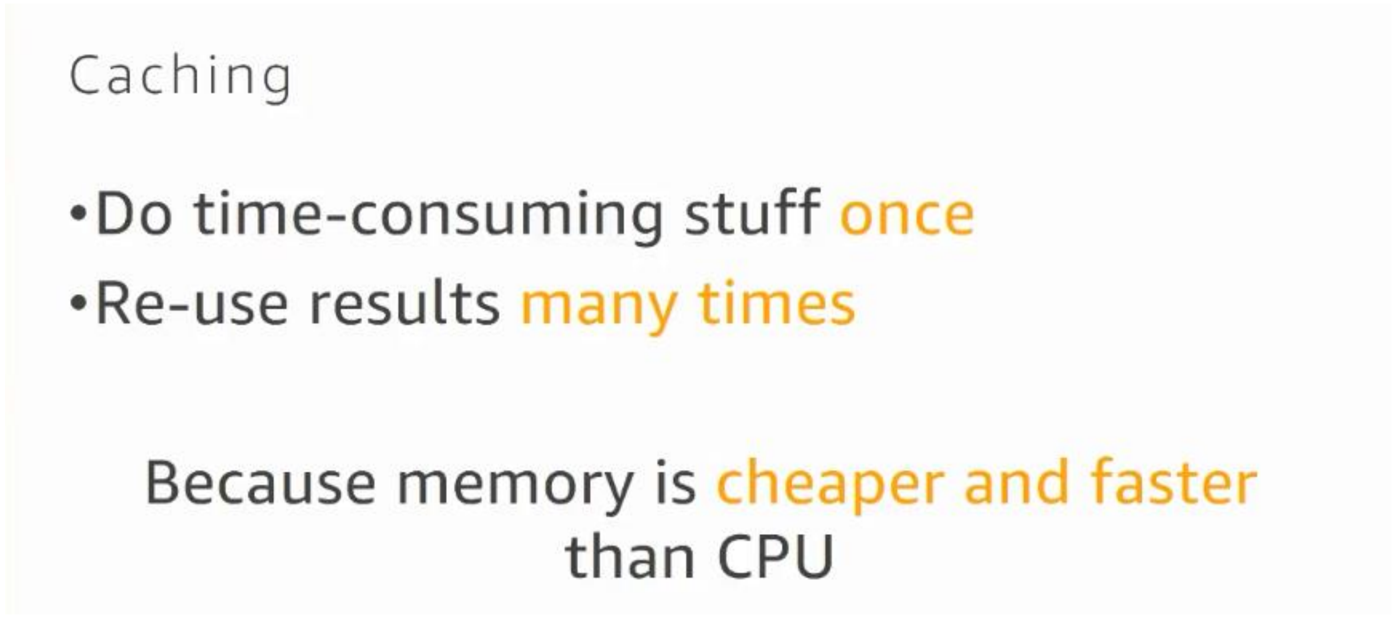
1.Increase rate	Complex
2.Parallelize	Complex
3.Do less	Easy!

To get speed into your AdTech applications and services, you can increase the rate at which you are running your transactions to minimize time taken, you can parallelize your application, or use caching



Three ways to become faster

1. Increase rate
2. Parallelize
3. Do less ← Caching



Caching

- Do time-consuming stuff once
- Re-use results many times

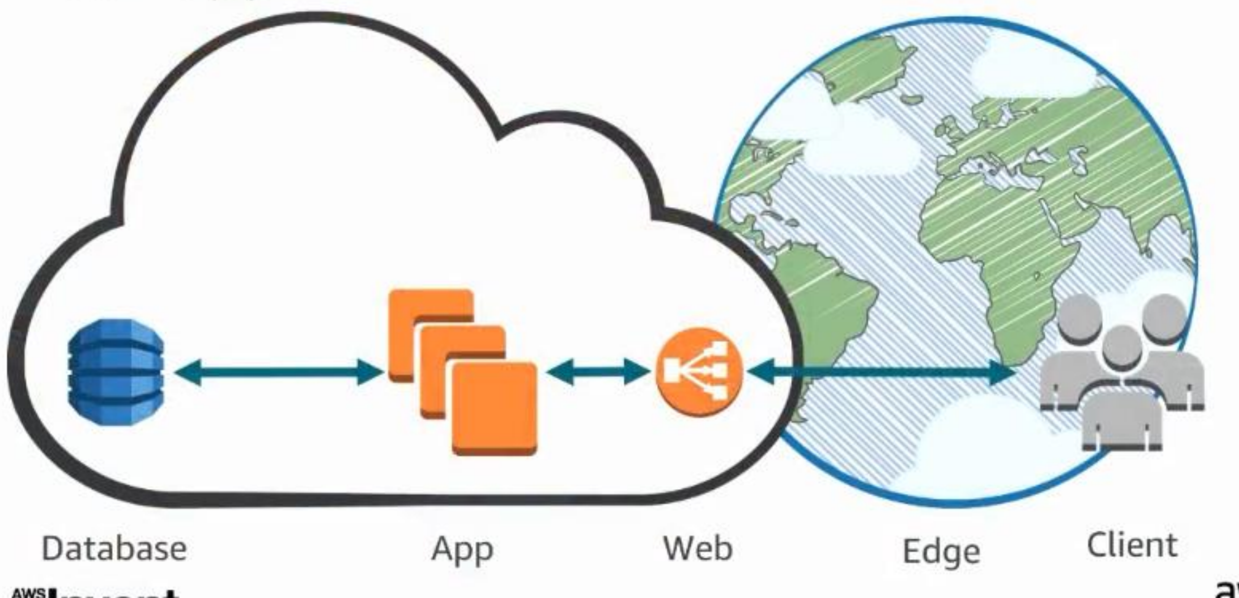
Because memory is cheaper and faster than CPU

You can get your transactions computed and then you save the results, it’s a lot easier to retrieve those results instead of computing them again

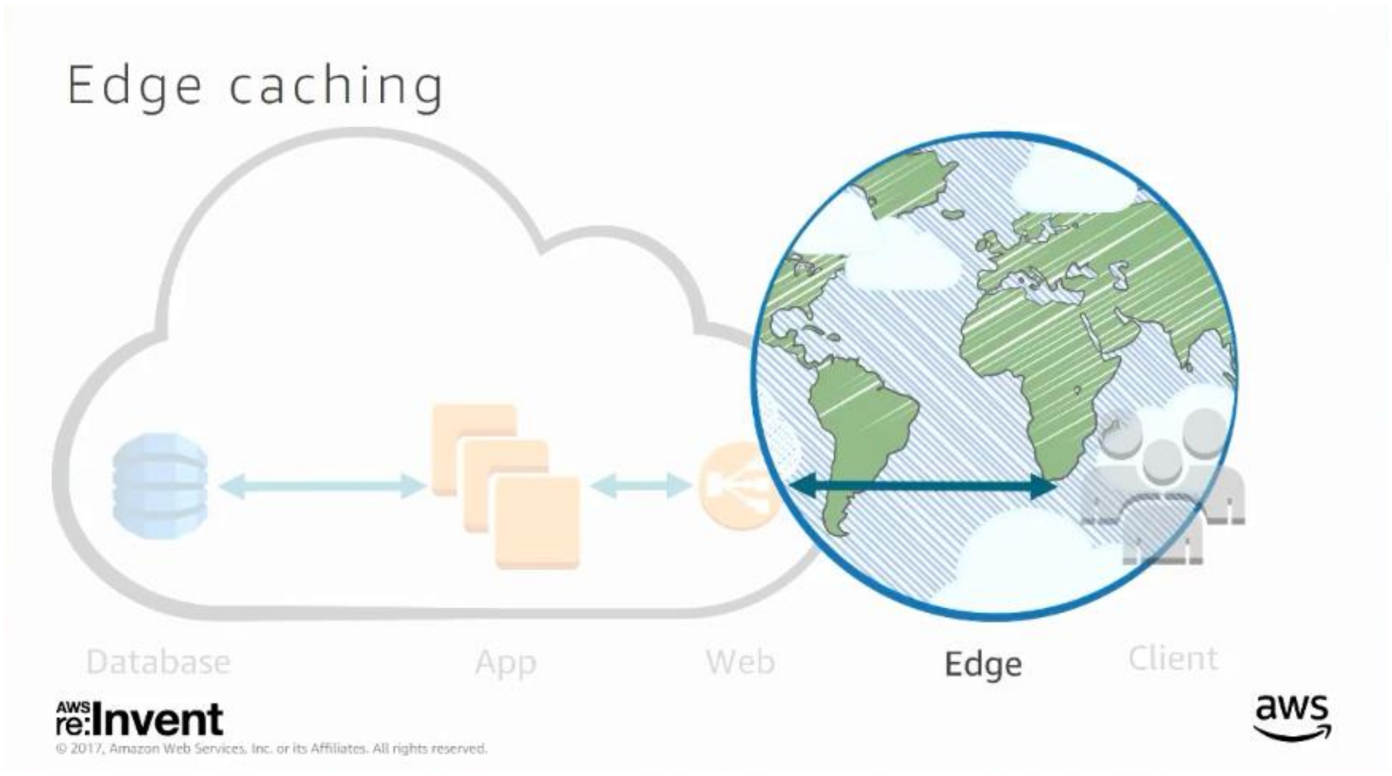
- Edge
- Web tier
- App tier
- Database

Using caching at each of these layers will save you time and money

Your app

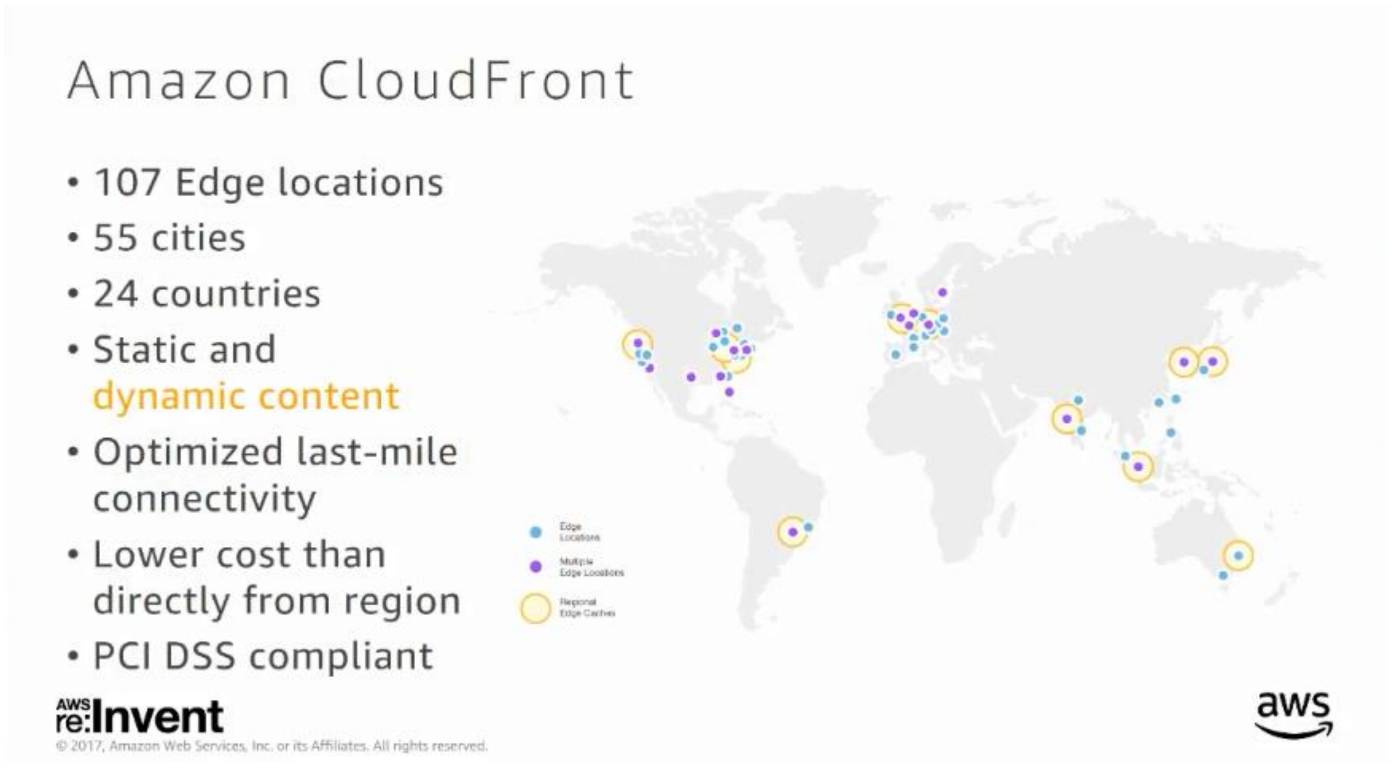


This is what most applications look like,



Edge caching

Edge caching is your first opportunity to become faster and add some speed to your applications using CloudFront



Amazon CloudFront

- 107 Edge locations
- 55 cities
- 24 countries
- Static and dynamic content
- Optimized last-mile connectivity
- Lower cost than directly from region
- PCI DSS compliant

Markus Ostertag

VP of Engineering, Team Internet AG

Who is Team Internet?

- Domain monetization business
- 35 people
- HQ in Munich, Germany
- Tech focused

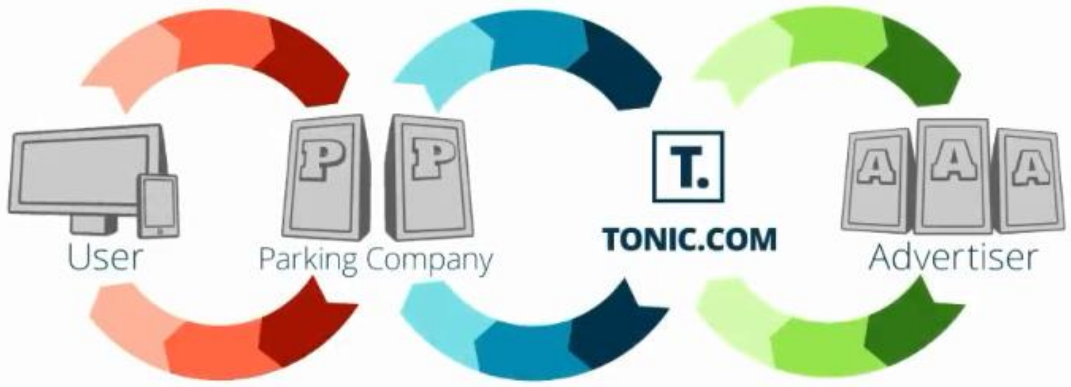
Team Internet
Ideas. Change. Markets.
A MATOMY MEDIA GROUP COMPANY

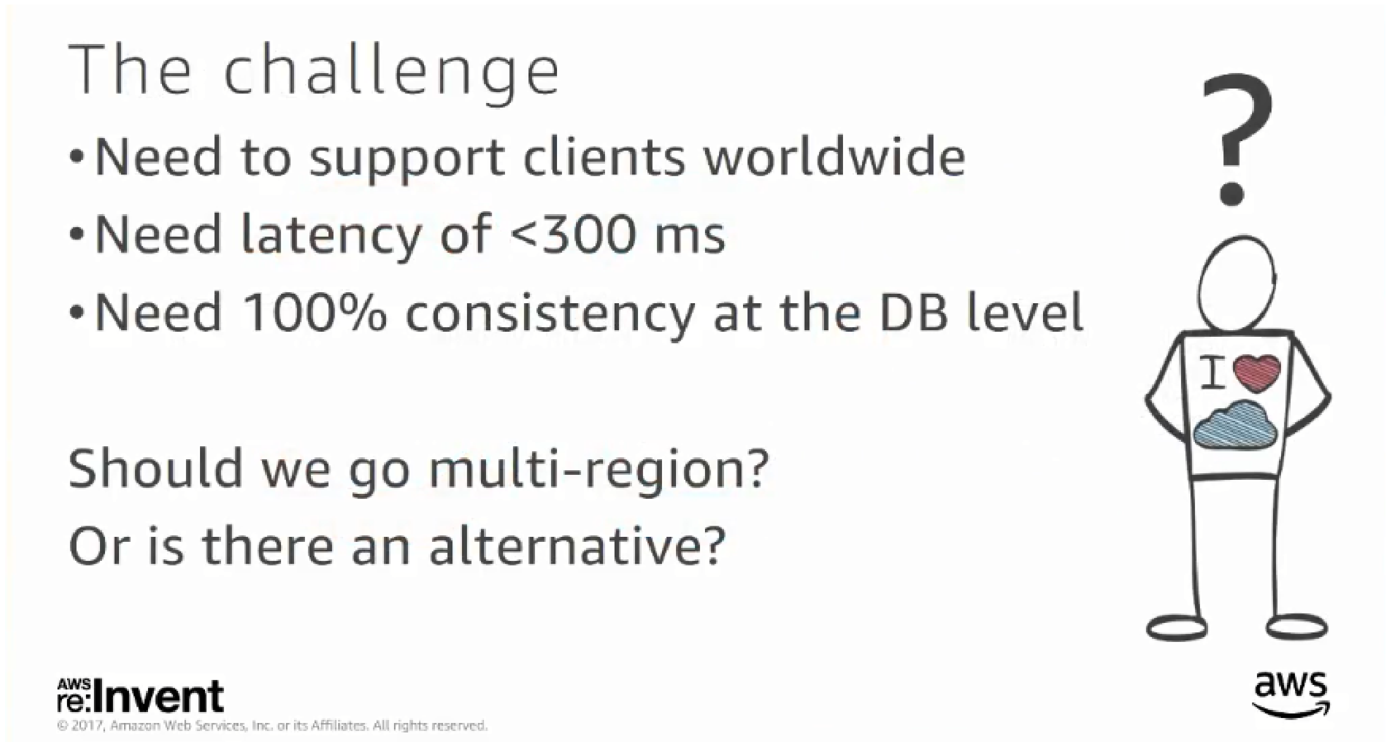
TONIC.

AWS re:Invent

aws

TONIC.

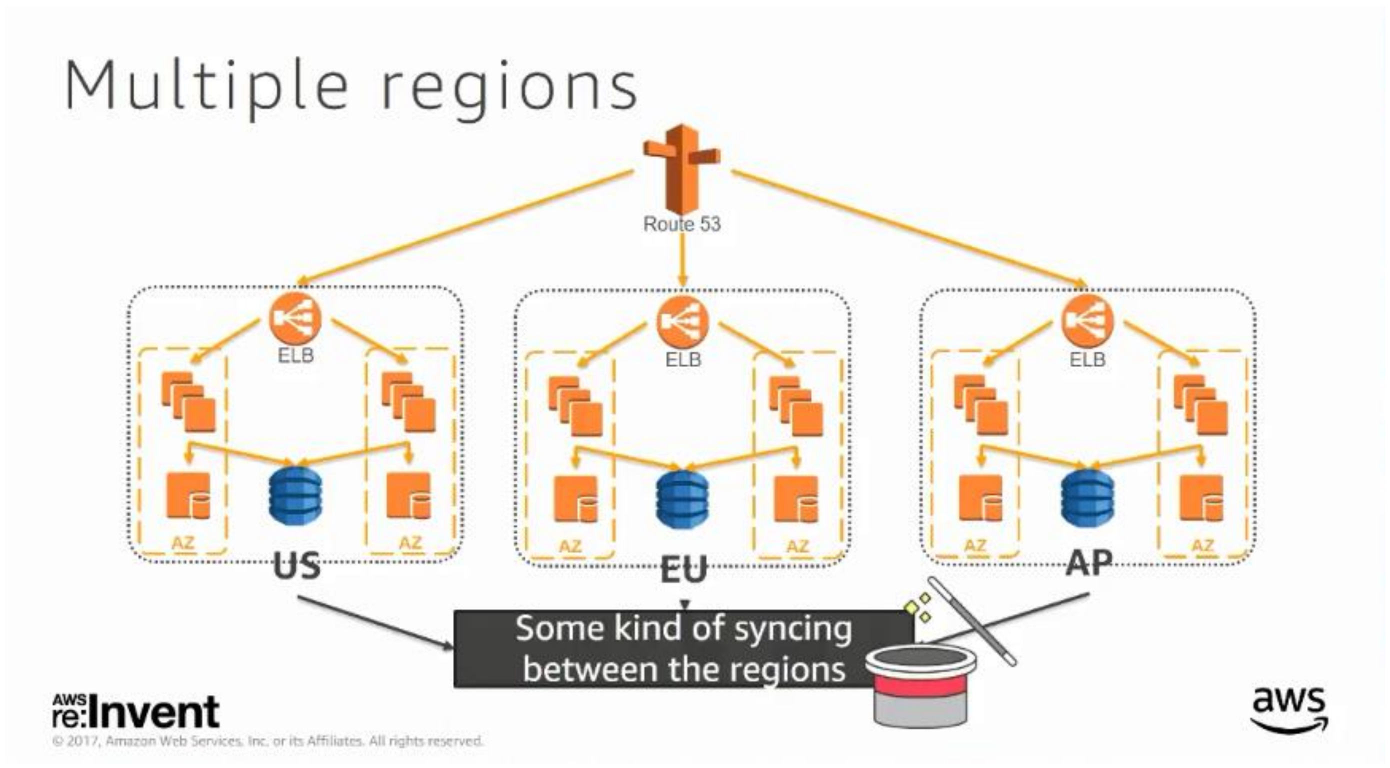




The challenge

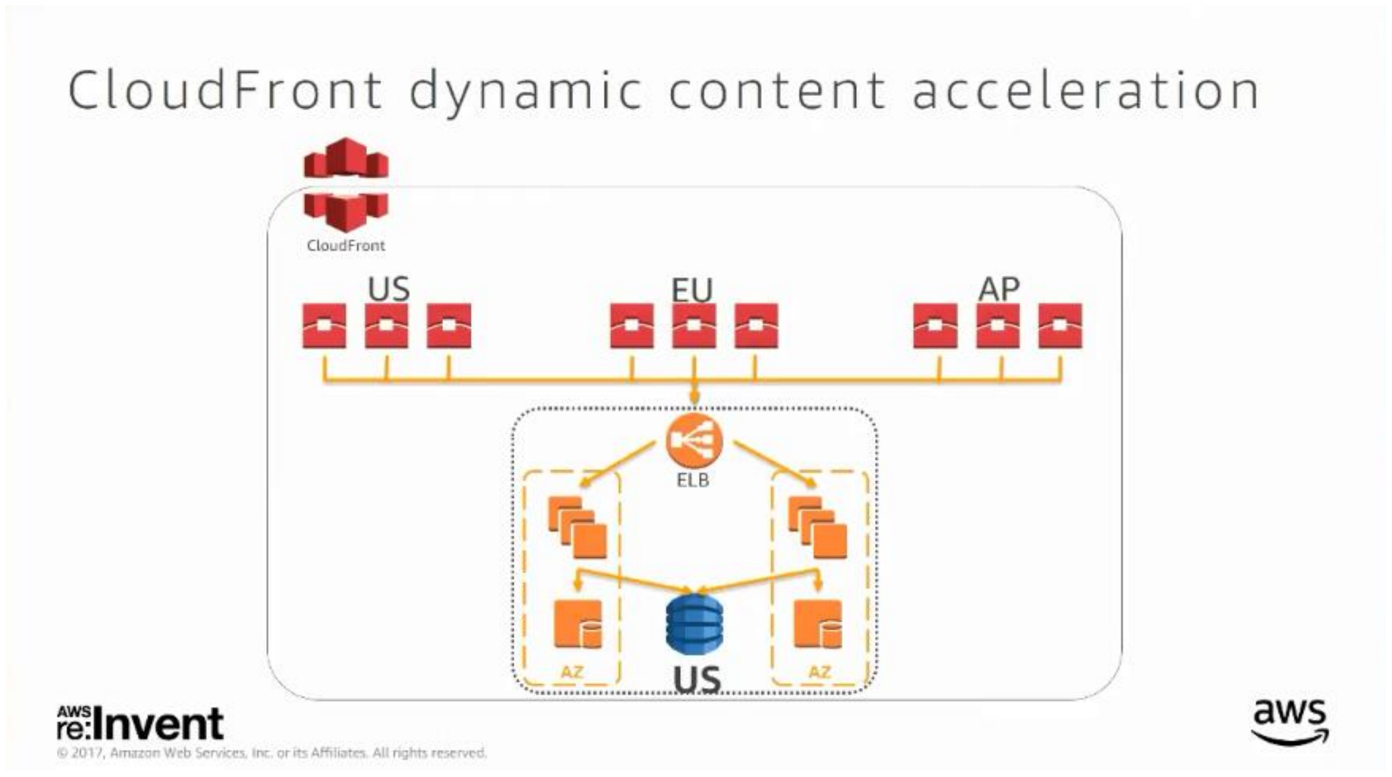
- Need to support clients worldwide
- Need latency of <300 ms
- Need 100% consistency at the DB level

Should we go multi-region?
Or is there an alternative?



Multiple regions

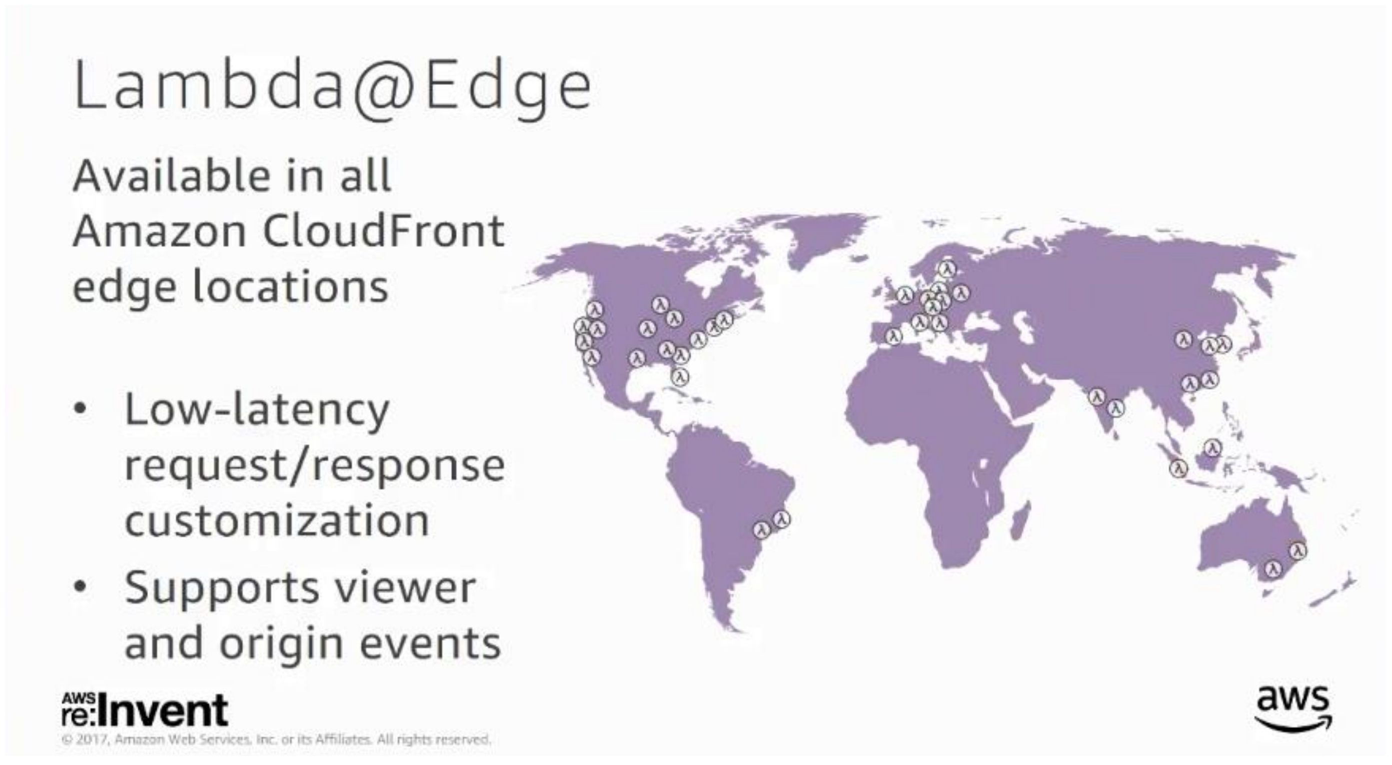
This was the architecture we had in mind before consulting AWS



CloudFront dynamic content acceleration

re:Invent

This is what was suggested to us for the real time bidding system architecture, CloudFront will help optimize the latency to our customers, and we only work out of only the US region and having CloudFront be the overlay for us around the world and help optimize the last mile to our customers.



Lambda@Edge

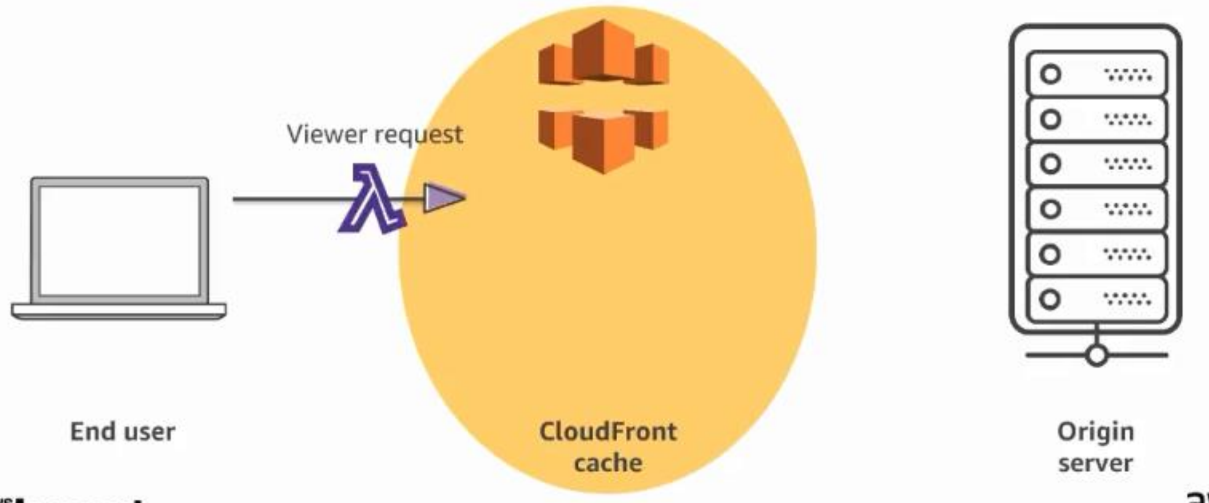
Available in all Amazon CloudFront edge locations

- Low-latency request/response customization
- Supports viewer and origin events

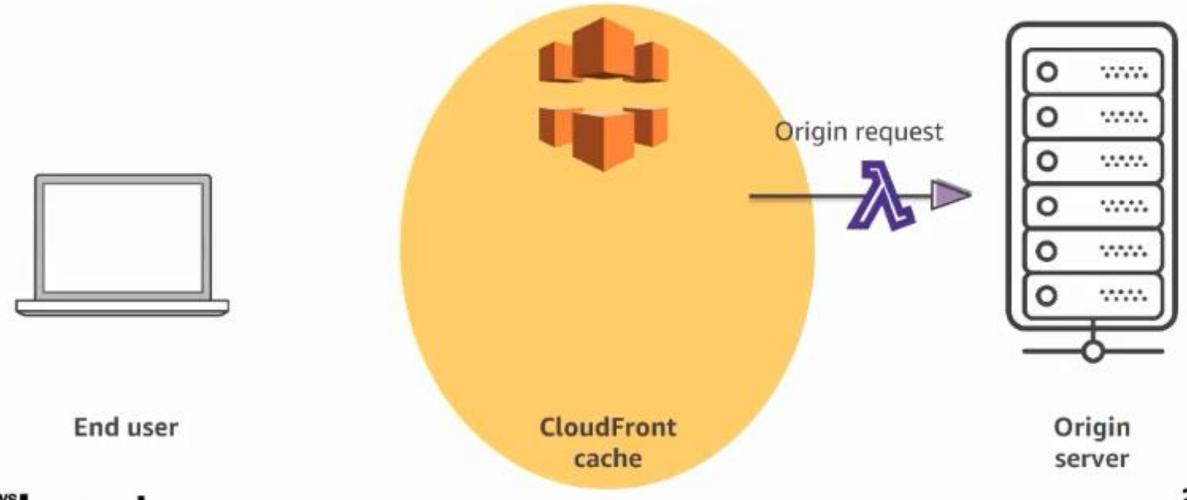
re:Invent

You can now modify the content at the edge by using lambdas at specific points in the request/response path as below

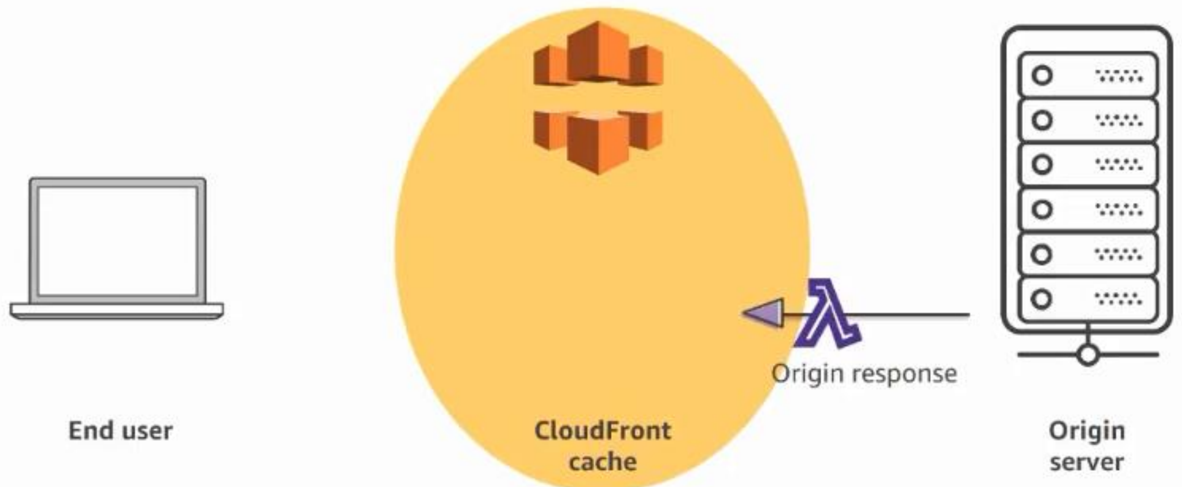
CloudFront triggers for Lambda@Edge functions



CloudFront triggers for Lambda@Edge functions

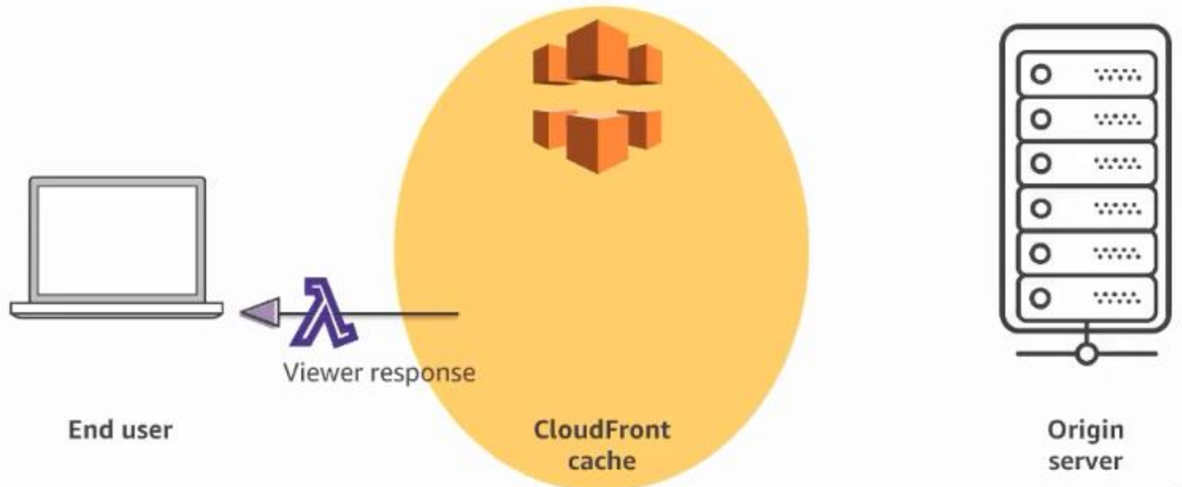


CloudFront triggers for Lambda@Edge functions

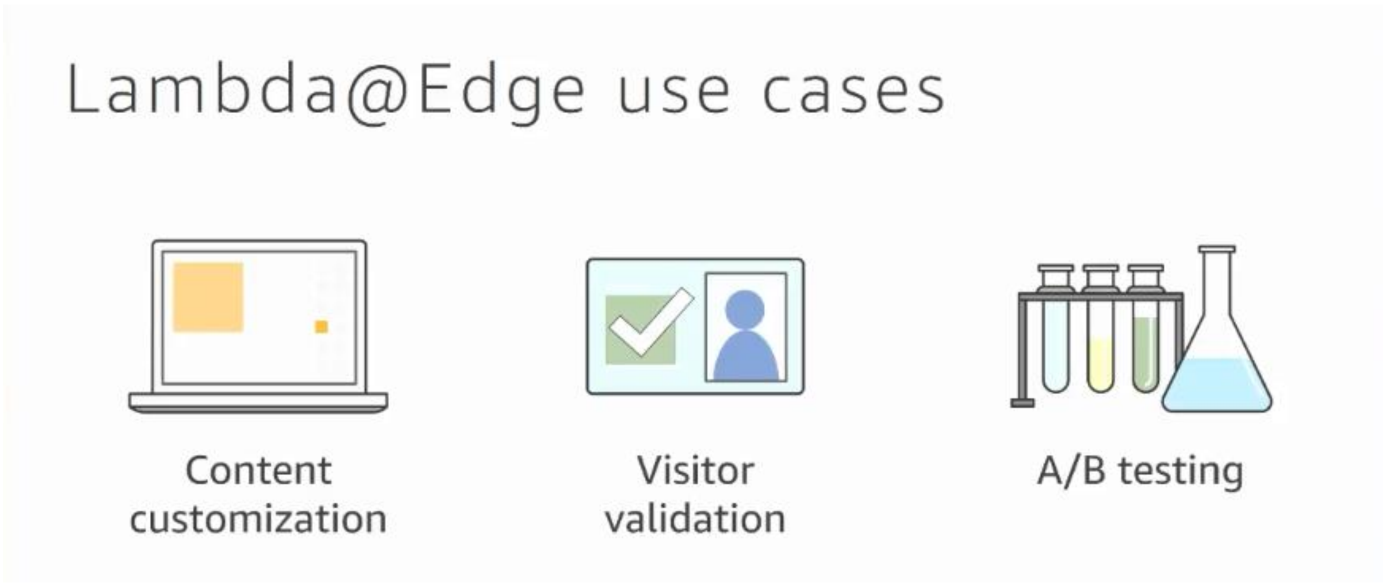


re:Invent

CloudFront triggers for Lambda@Edge functions



re:Invent

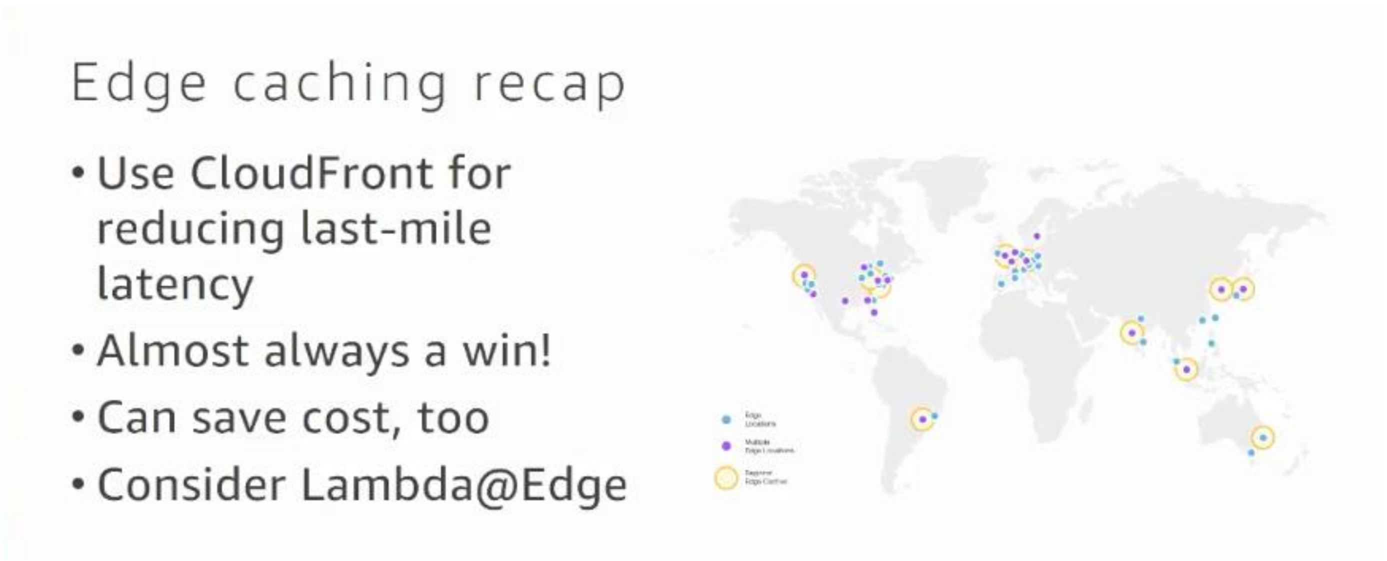


Lambda@Edge use cases

Content customization

Visitor validation

A/B testing

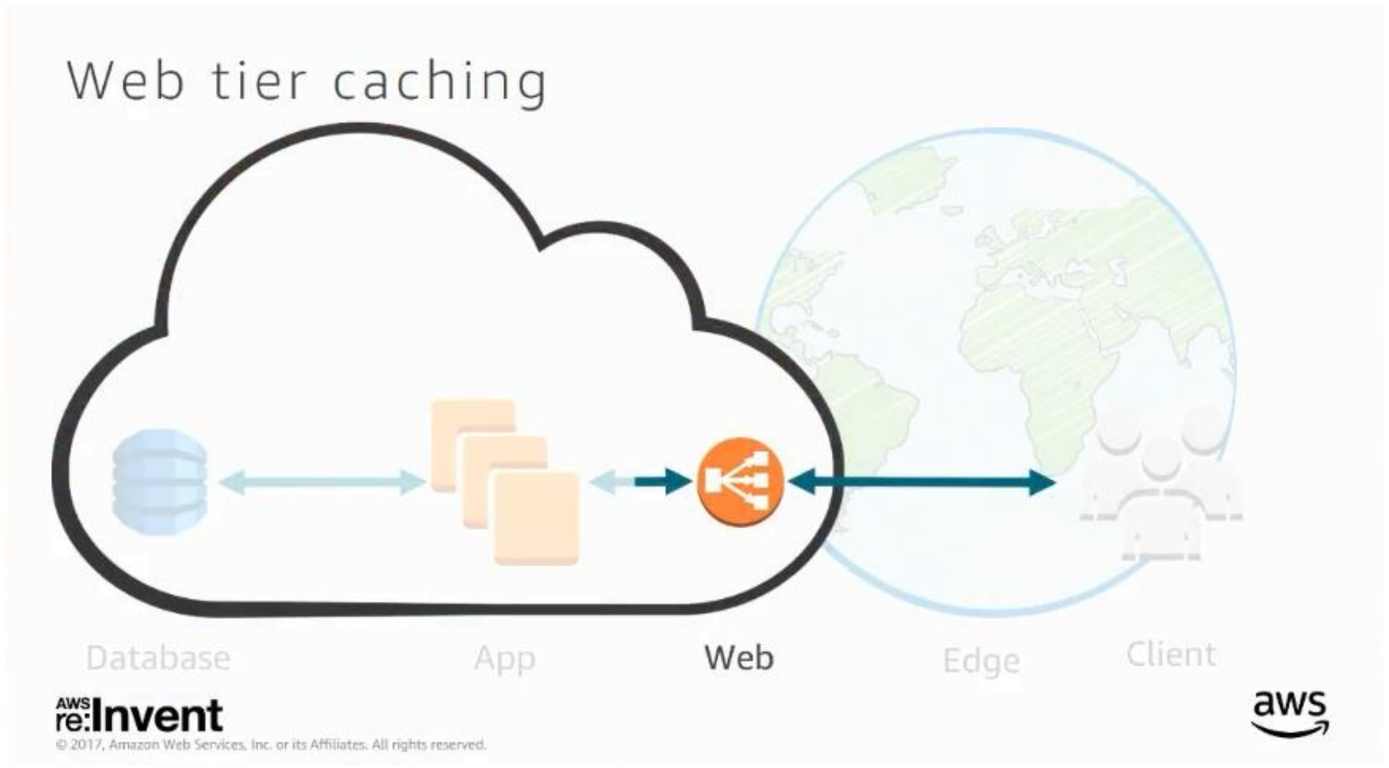


Edge caching recap

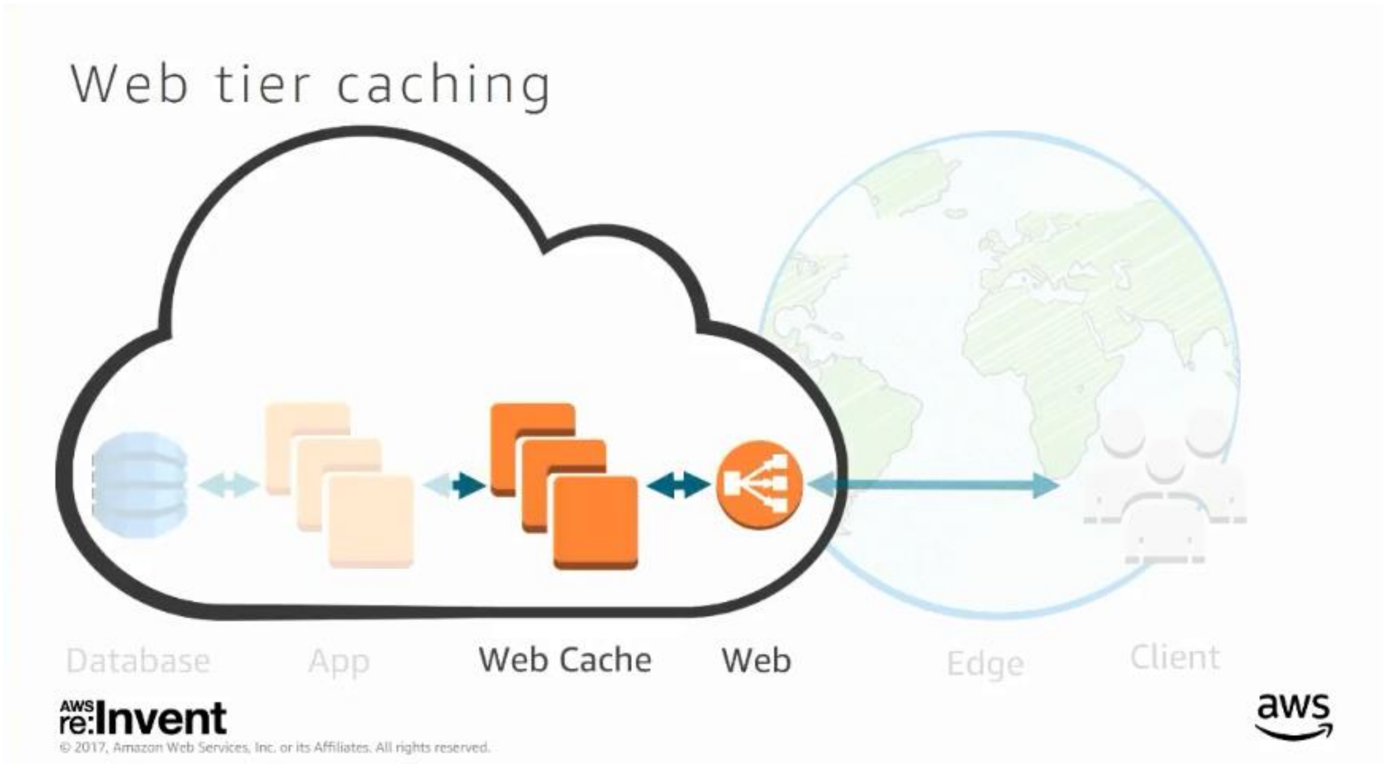
- Use CloudFront for reducing last-mile latency
- Almost always a win!
- Can save cost, too
- Consider Lambda@Edge

- Edge ✓
- Web tier
- App tier
- Database

Web tier caching

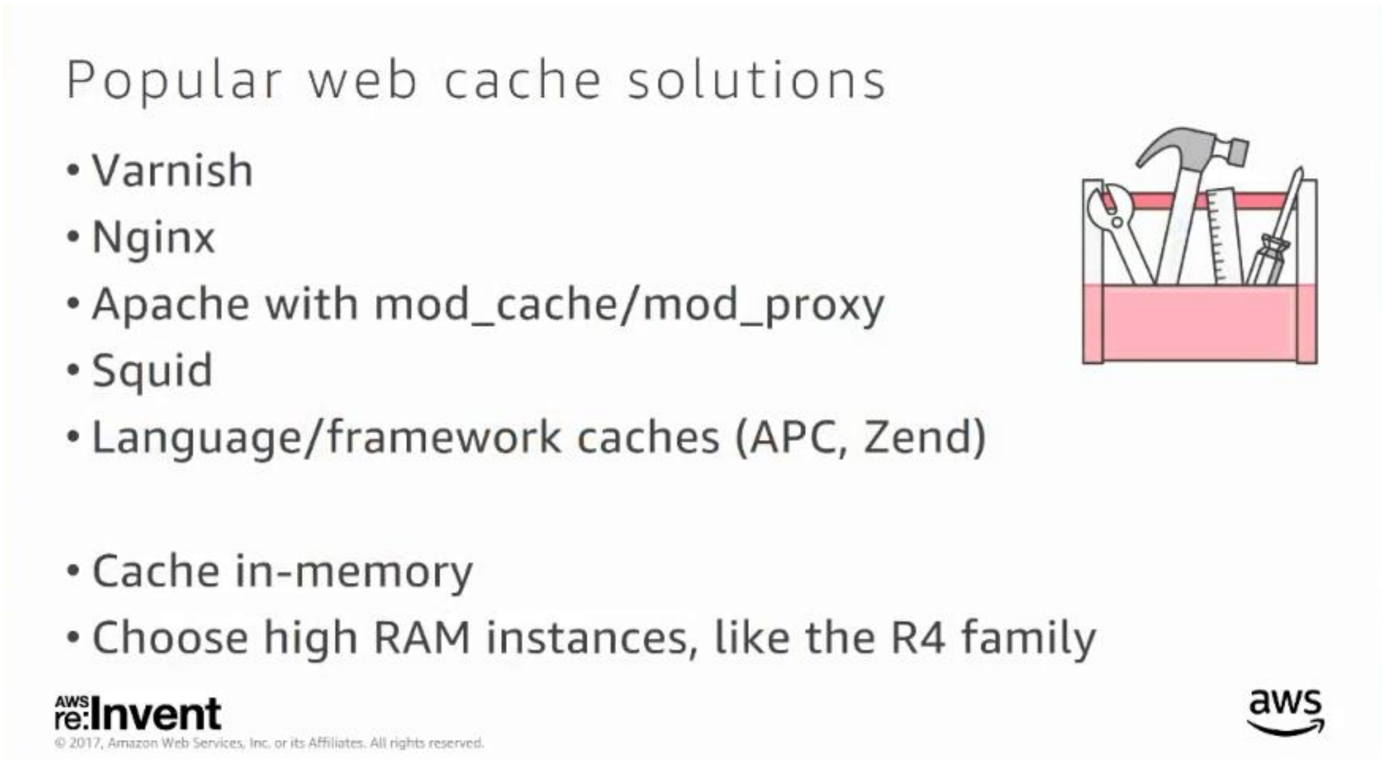


This is the bit where you deliver content to your users



Web tier caching

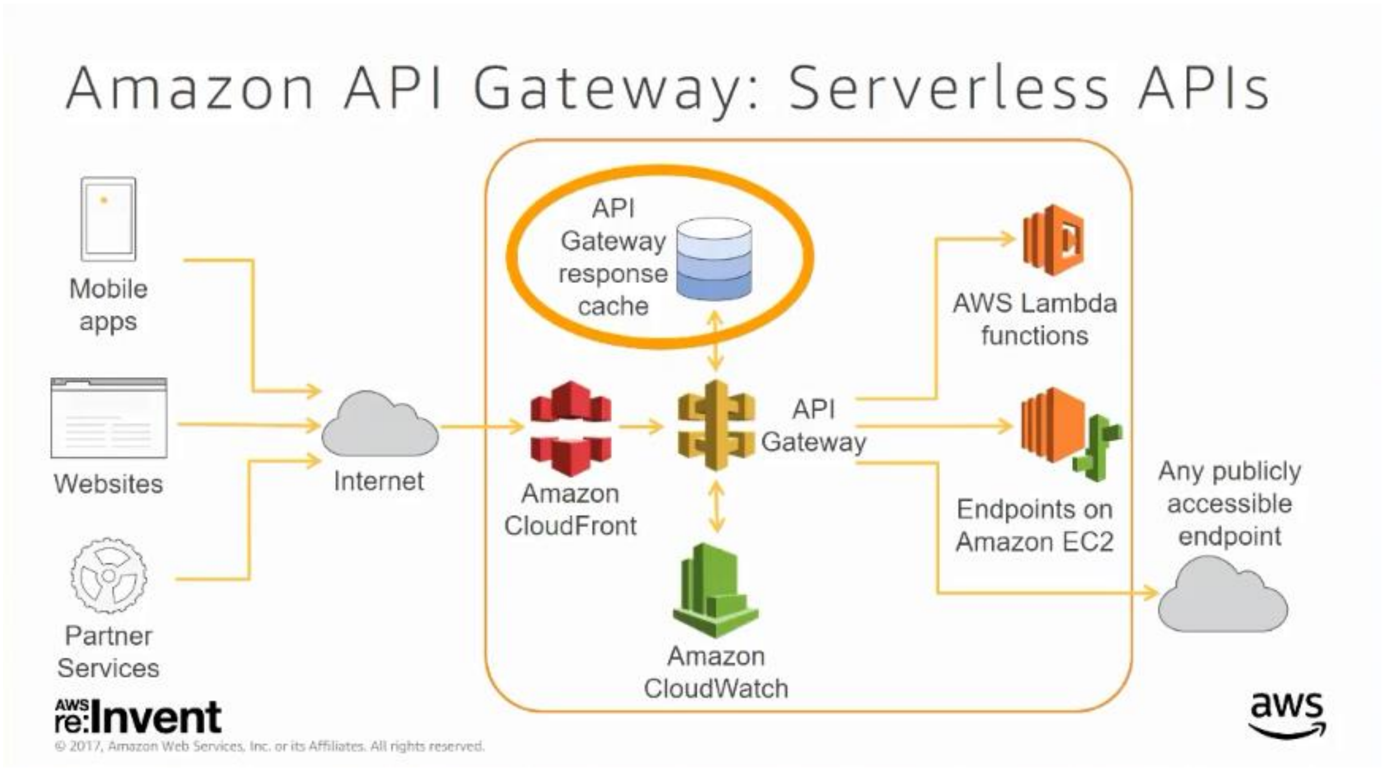
You can put a cache here before it hits your application, you can decide what to cache upfront by trying to cache your HTTP responses in-memory at the web layer. This means that you should use EC2 instances that have more memory



Popular web cache solutions

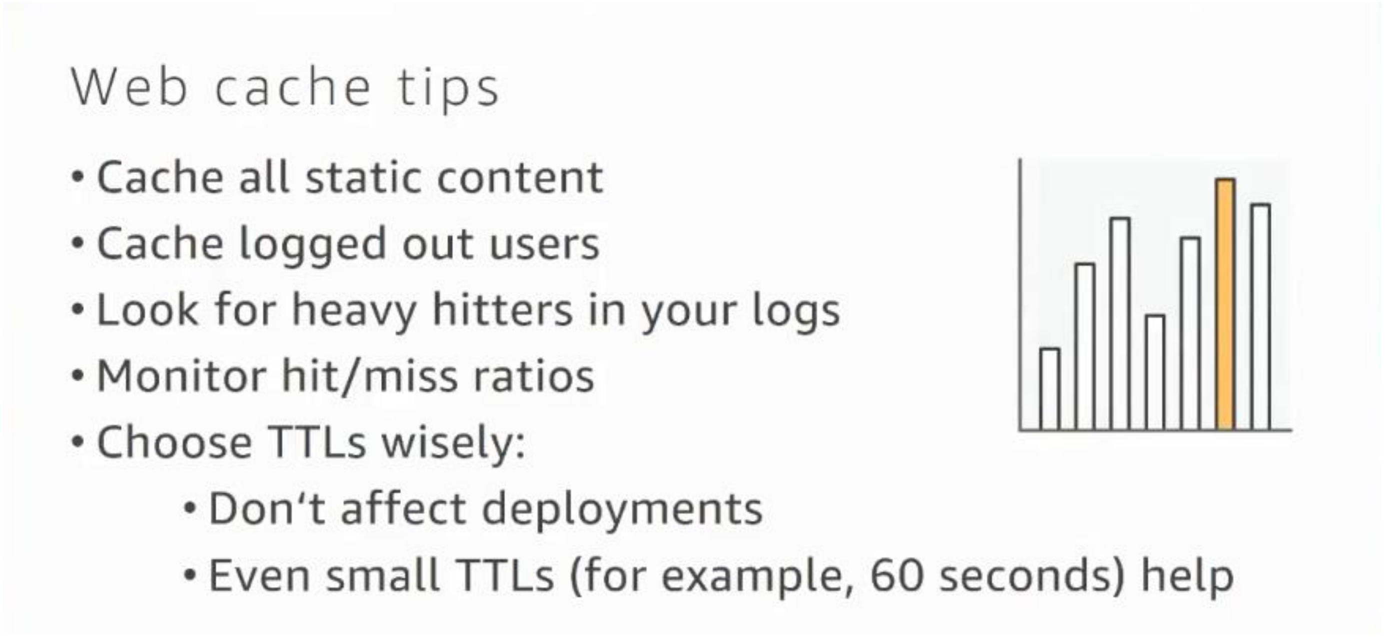
- Varnish
- Nginx
- Apache with mod_cache/mod_proxy
- Squid
- Language/framework caches (APC, Zend)

- Cache in-memory
- Choose high RAM instances, like the R4 family



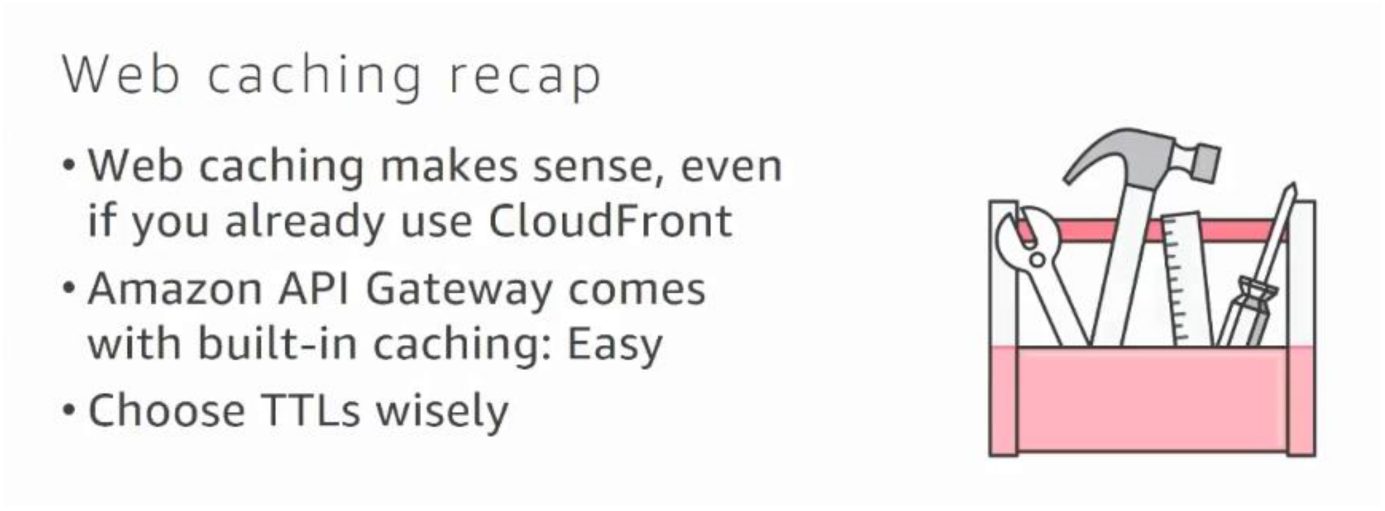
Amazon API Gateway: Serverless APIs

You can add an in-memory cache layer from an API Gateway on top of your HTTP stack to be used for caching some of your responses



Web cache tips

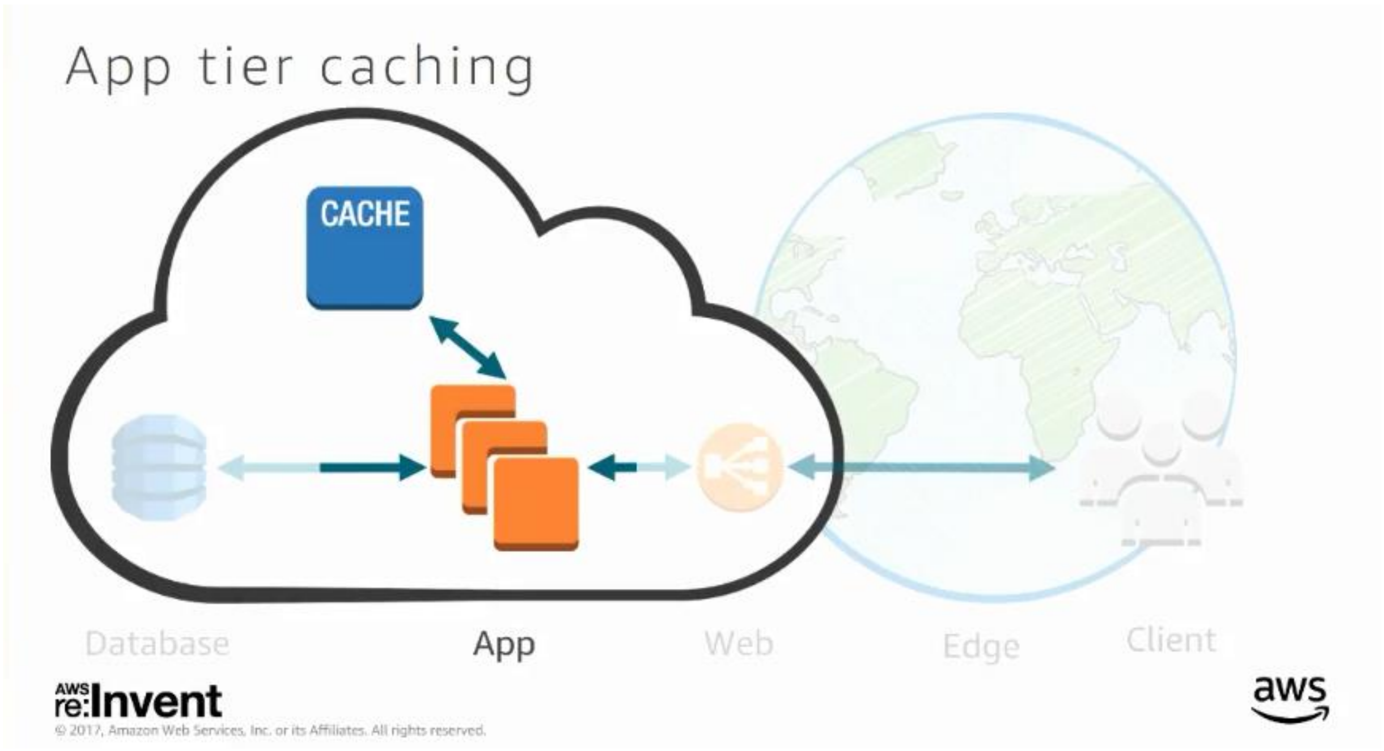
- Cache all static content
- Cache logged out users
- Look for heavy hitters in your logs
- Monitor hit/miss ratios
- Choose TTLs wisely:
    - Don't affect deployments
    - Even small TTLs (for example, 60 seconds) help



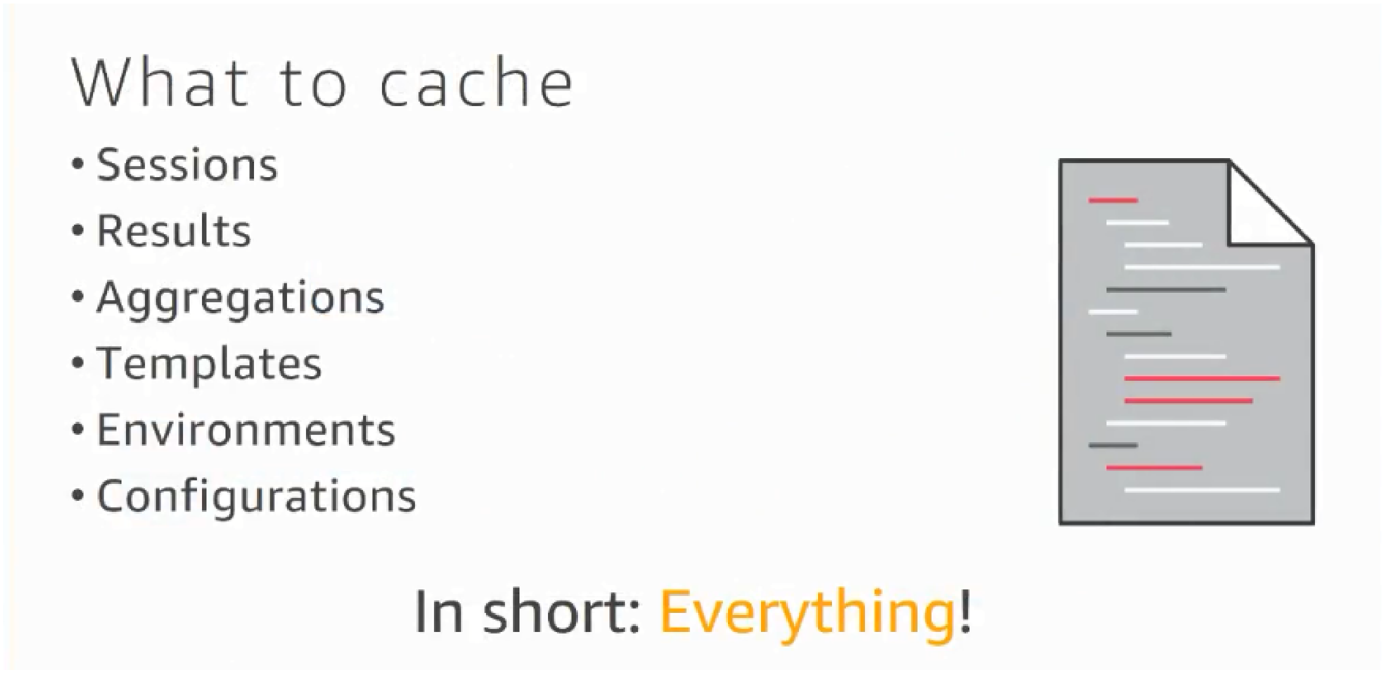
Web caching recap

- Web caching makes sense, even if you already use CloudFront
- Amazon API Gateway comes with built-in caching: Easy
- Choose TTLs wisely

- Edge ✓
- Web tier ✓
- App tier
- Database


App tier caching
CACHE
Database
App
Web
Edge
Client
AWS re:Invent
© 2017, Amazon Web Services, Inc. or its Affiliates. All rights reserved.
aws

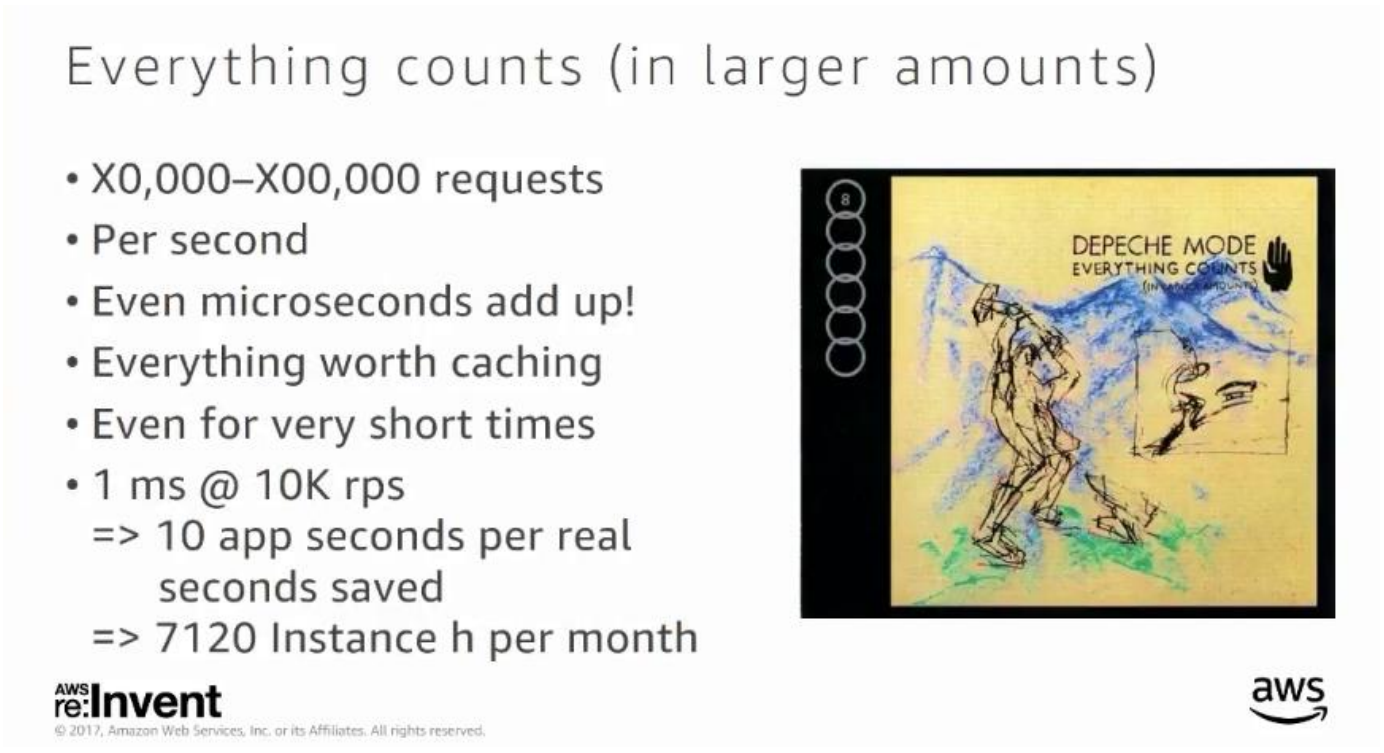
You can also put a cache on top of your applications that are running on some EC2 servers to help reduce latency



What to cache

- Sessions
- Results
- Aggregations
- Templates
- Environments
- Configurations

In short: Everything!



Everything counts (in larger amounts)

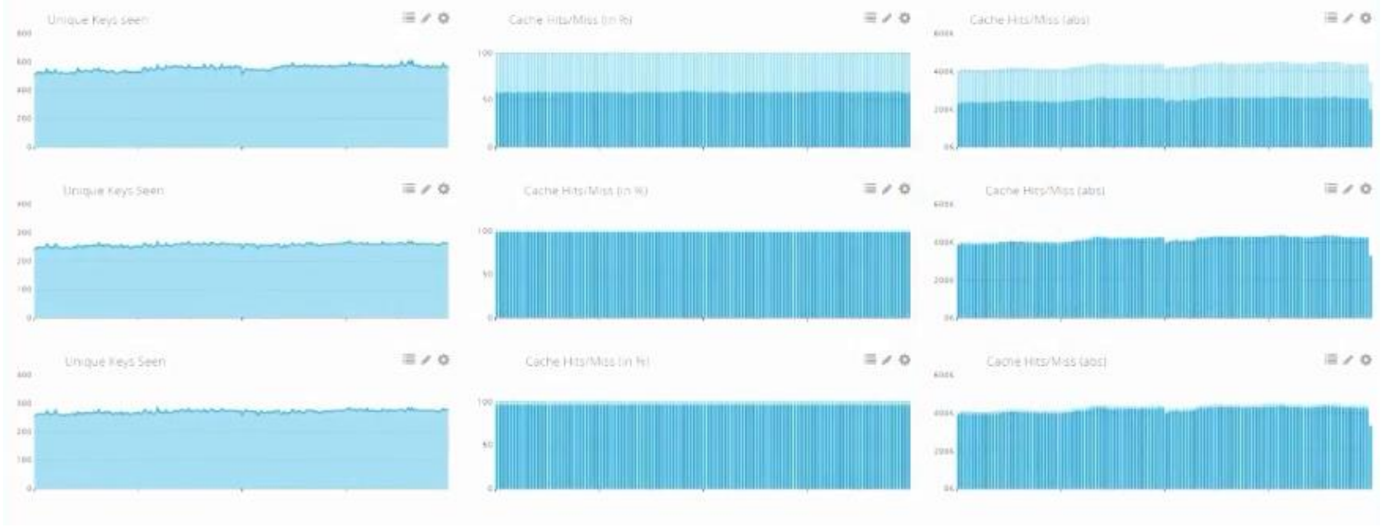
- X0,000–X00,000 requests
- Per second
- Even microseconds add up!
- Everything worth caching
- Even for very short times
- 1 ms @ 10K rps
  => 10 app seconds per real seconds saved
  => 7120 Instance h per month

DEPECHE MODE
EVERYTHING COUNTS
(IN LARGER AMOUNTS)

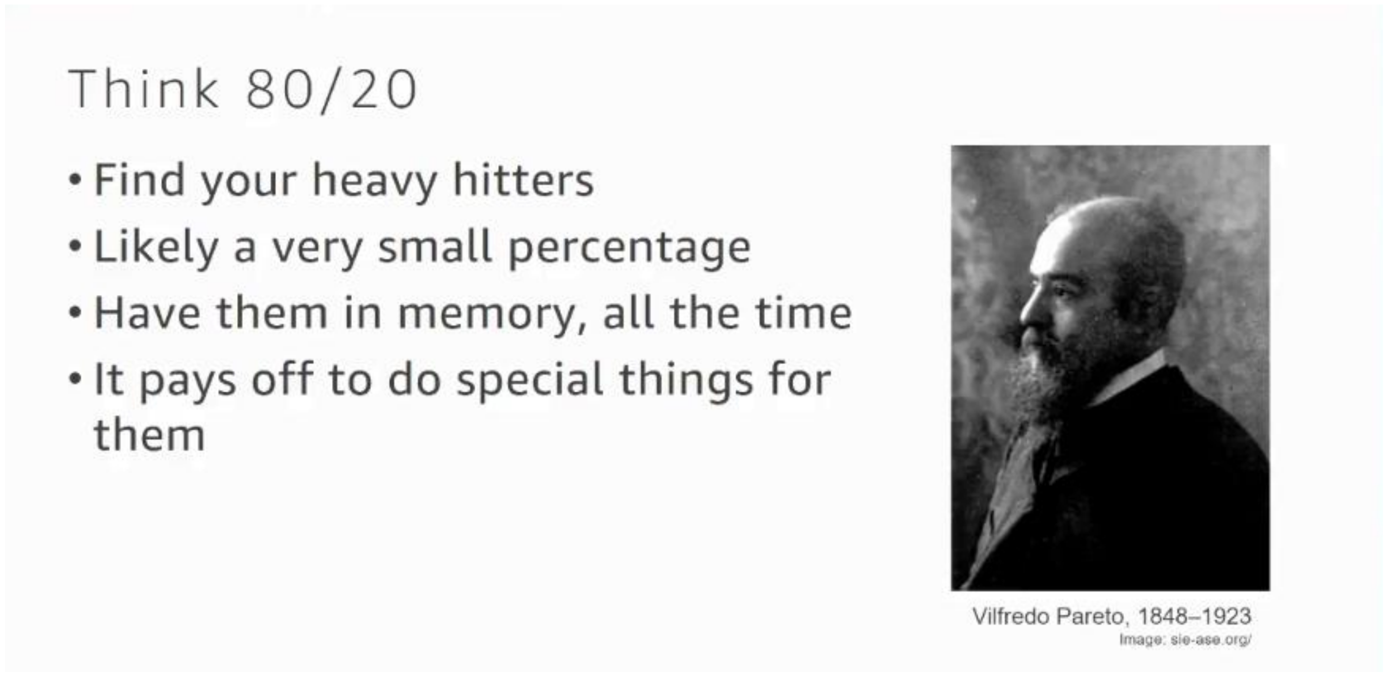
AWS re:Invent

aws

Monitor everything



It is important to know what you can cache and how you can cache. Above is one of our data dashboards that helps us figure out what to cache. We are caching domains, advertisers, budgets, etc. We are using 3 main columns like *unique keys seen in a specific timeframe* that tells us how much stuff we are caching in-memory, the ***cache hit/miss ratio (number and percentage)*** tells us how well we are using the cache we have and how much of the overall requests we are serving from the cache.

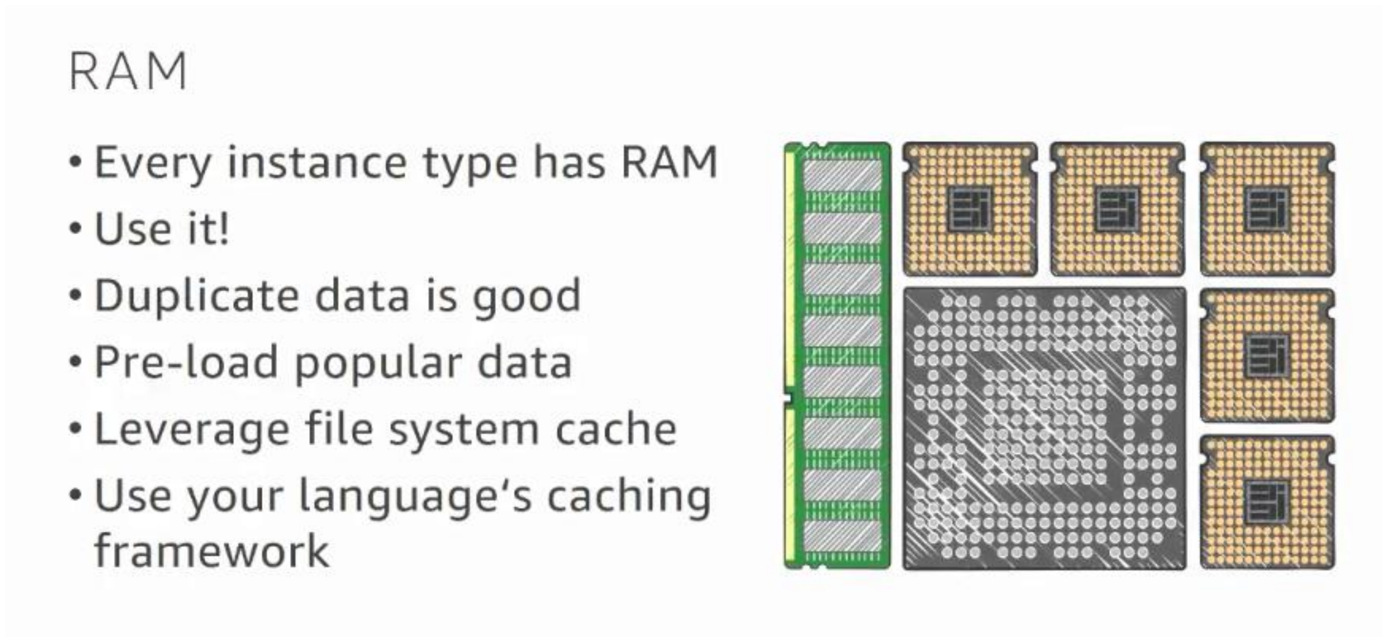


Think 80/20

- Find your heavy hitters
- Likely a very small percentage
- Have them in memory, all the time
- It pays off to do special things for them

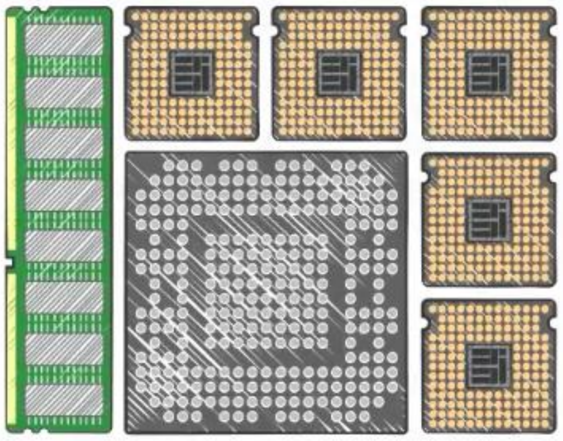
Vilfredo Pareto, 1848–1923
Image: sie-ase.org/

You need to cache things specially for your heavy hitters to improve responses to them,

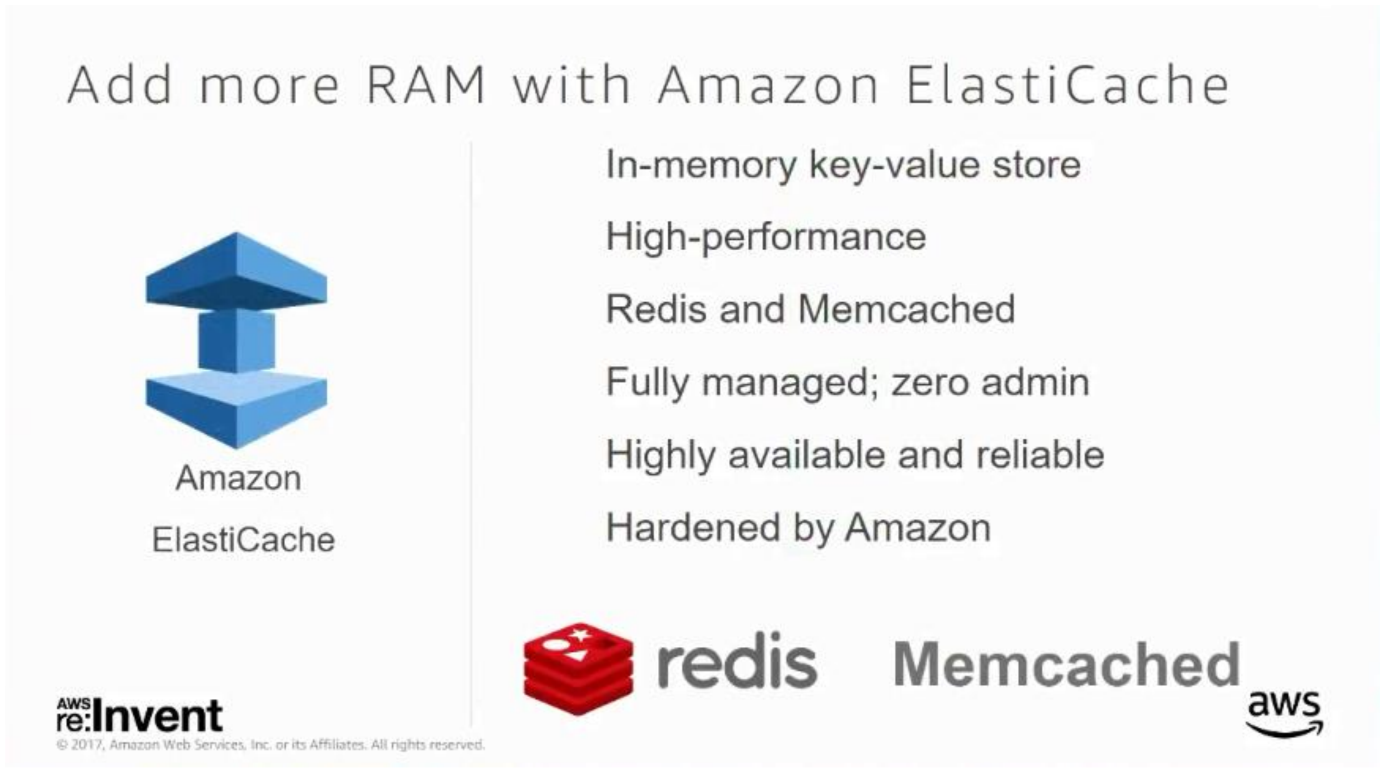


RAM

- Every instance type has RAM
- Use it!
- Duplicate data is good
- Pre-load popular data
- Leverage file system cache
- Use your language's caching framework



You need to monitor RAM usage, if you are not using a lot of your RAM then you can cache things in-memory to use spare RAM. You can use idle time to pre-compute things into your cache



Add more RAM with Amazon ElastiCache

Amazon
ElastiCache

In-memory key-value store

High-performance

Redis and Memcached

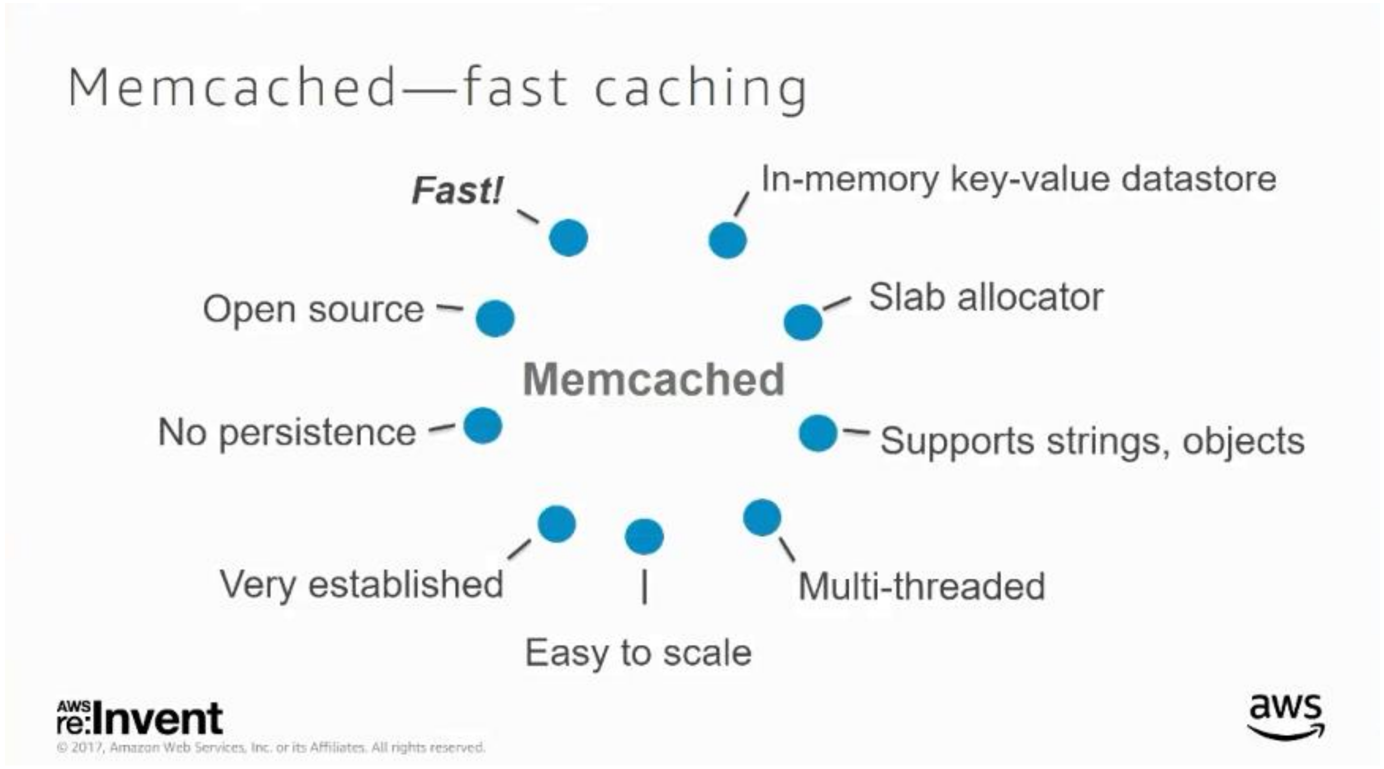
Fully managed; zero admin

Highly available and reliable

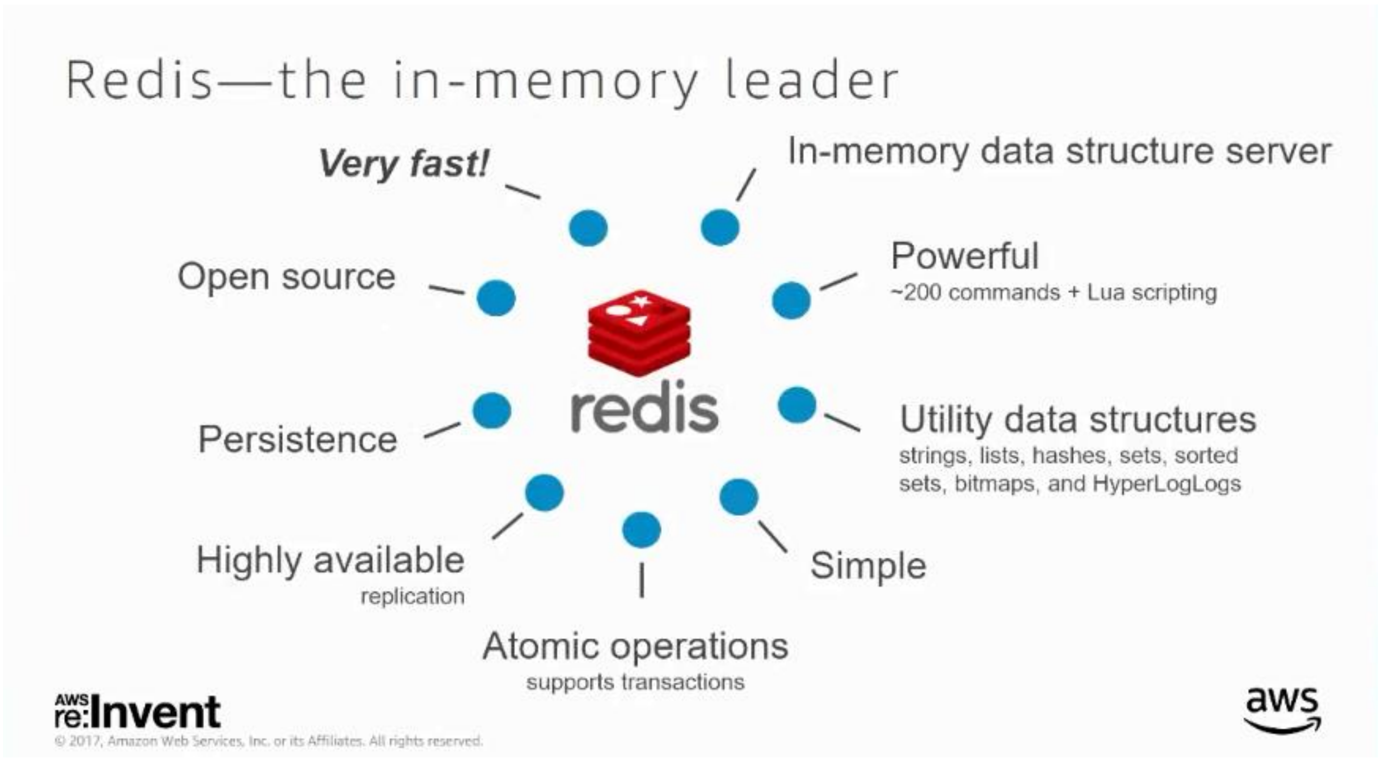
Hardened by Amazon

redis
Memcached
AWS re:Invent
aws

ElastiCache is a RAM-as-a-service option that you can use if your instance does not have enough RAM



Memcached—fast caching



Redis—the in-memory leader

redis

Very fast!

In-memory data structure server

Open source

Powerful
~200 commands + Lua scripting

Persistence

Utility data structures
strings, lists, hashes, sets, sorted sets, bitmaps, and HyperLogLogs

Highly available
replication

Simple

Atomic operations
supports transactions

AWS re:Invent

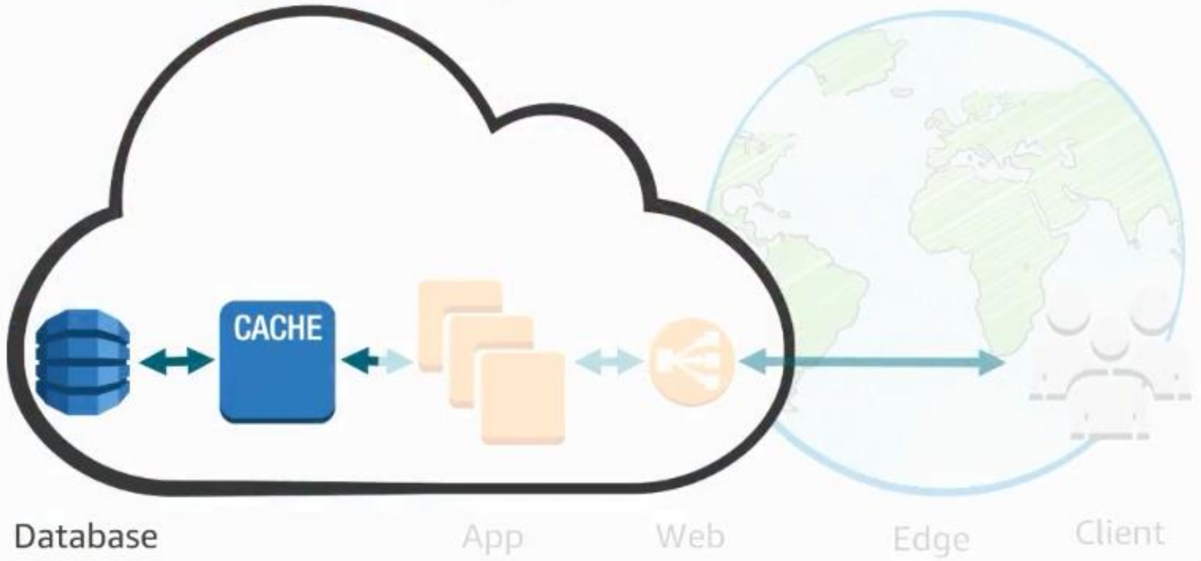
aws

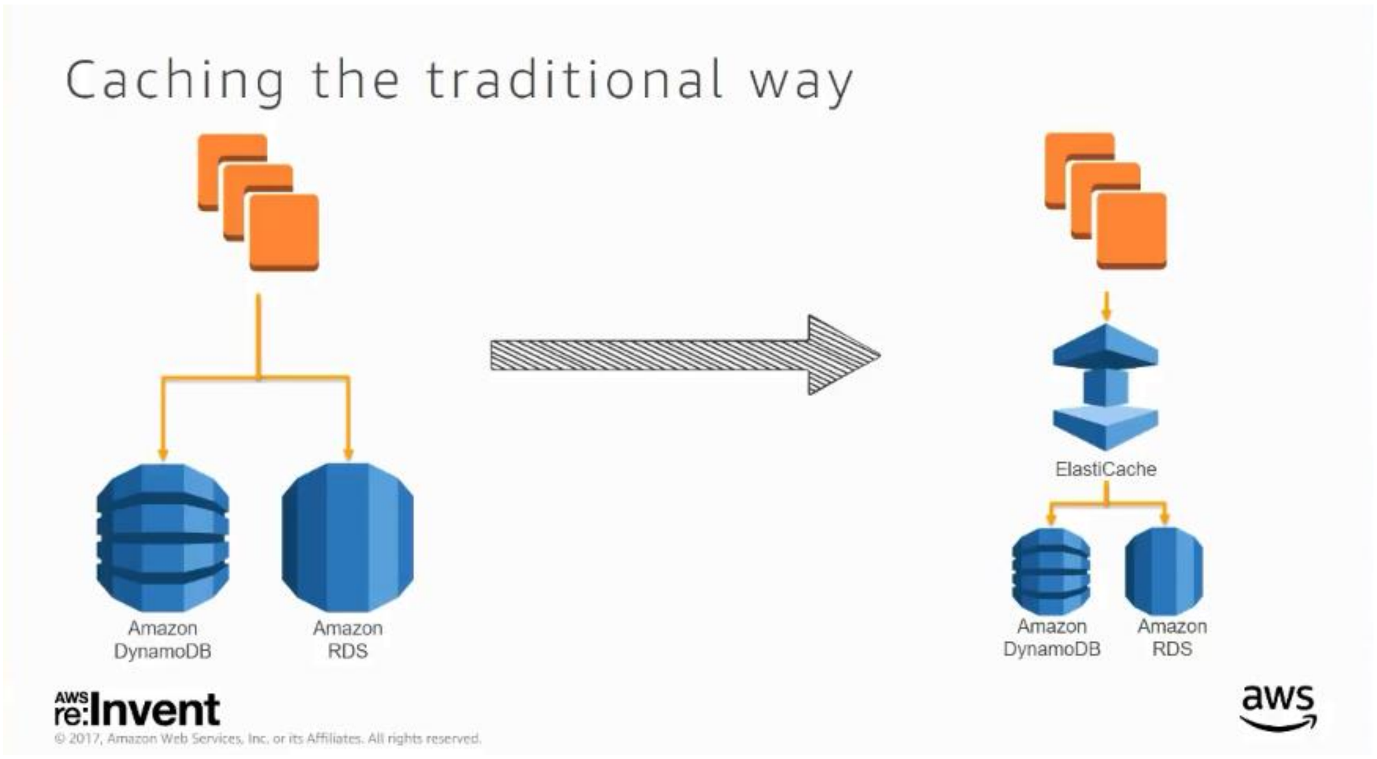
App caching recap

- Monitor everything!
- Find the 20% of users with 80% impact
- Cache everything, in RAM!
- Consider using ElastiCache

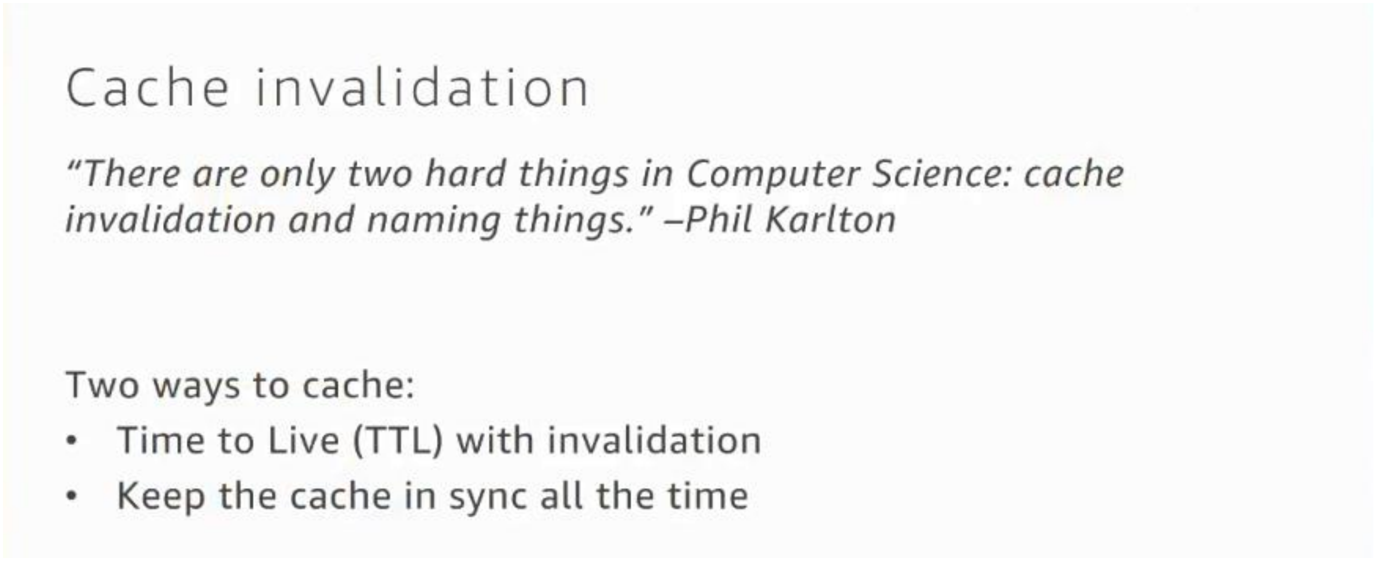
- Edge ✓
- Web tier ✓
- App tier ✓
- Database

Database caching




Caching the traditional way

We can make our application talk to the cache first when trying to get data before we talk to our database


Cache invalidation

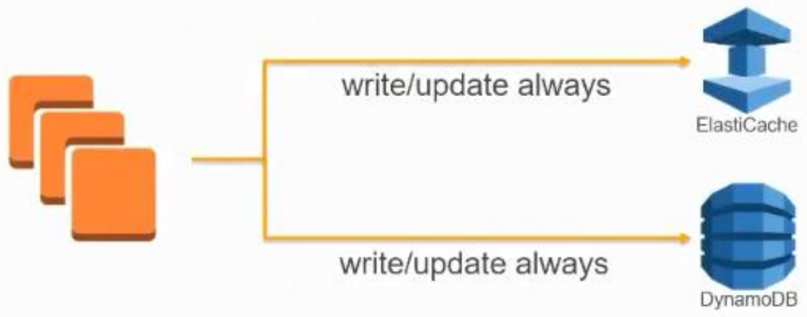
"There are only two hard things in Computer Science: cache invalidation and naming things." –Phil Karlton

Two ways to cache:

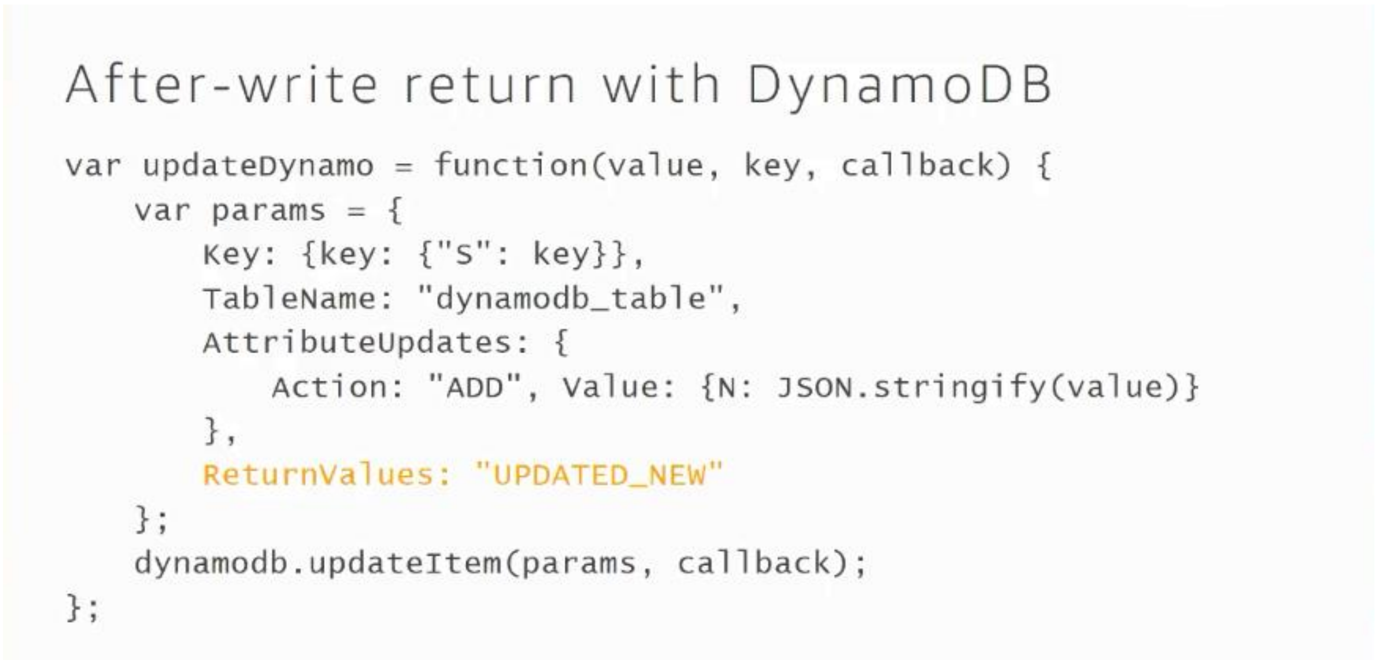
- Time to Live (TTL) with invalidation
- Keep the cache in sync all the time

We can put a TTL on the key in the cache we are using like Redis or Memcached so that the cache can invalidate the cache object after TTL elapses. The 2nd option is to always keep the cache in sync using synchronous write.

Synchronous writes



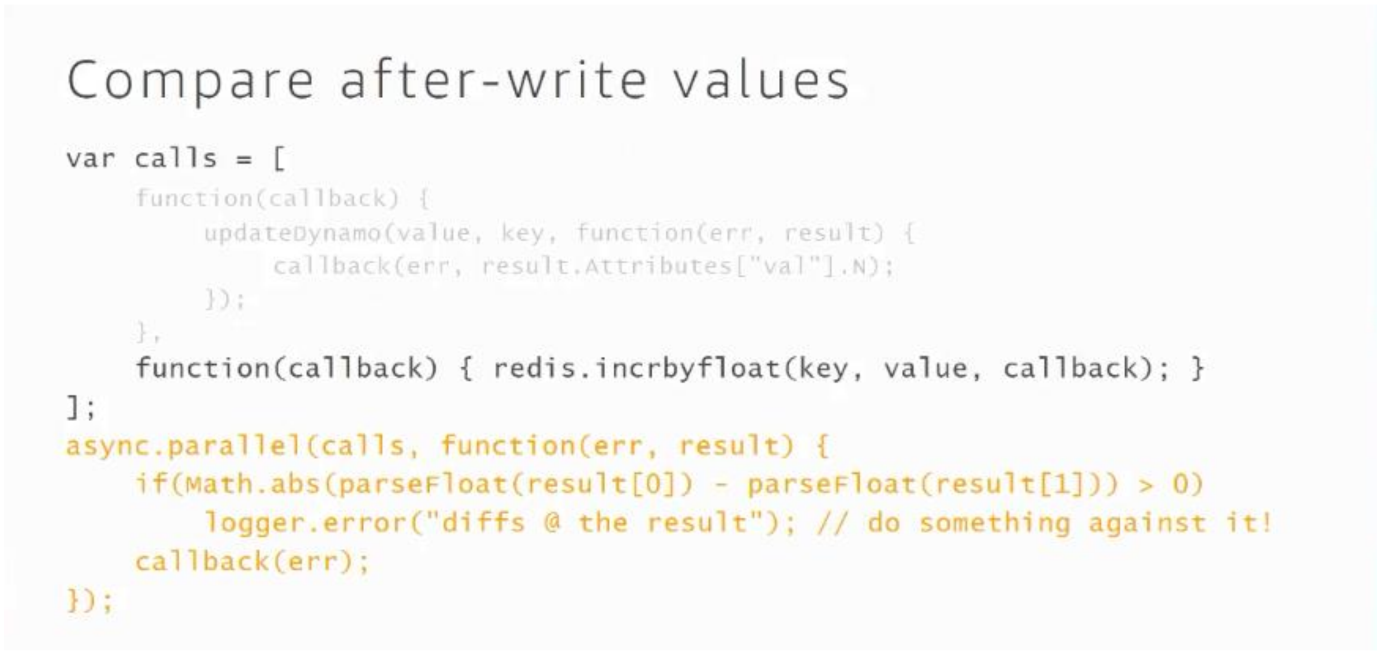
Sync/check in the app with after-write return values



After-write return with DynamoDB

```
var updateDynamo = function(value, key, callback) {
    var params = {
        Key: {key: {"S": key}},
        TableName: "dynamodb_table",
        AttributeUpdates: {
            Action: "ADD", Value: {N: JSON.stringify(value)}
        },
        ReturnValues: "UPDATED_NEW"
    };
    dynamodb.updateItem(params, callback);
};
```

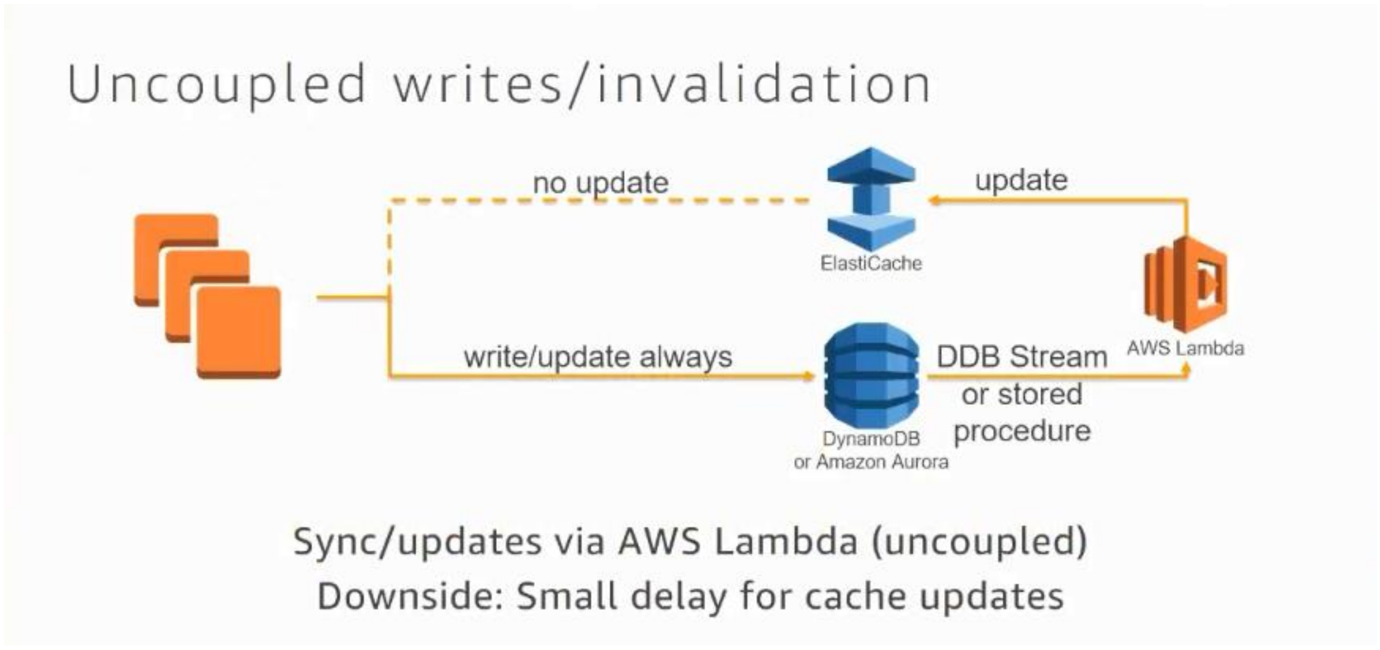
This is a call against DynamoDB, this will make DynamoDB give us the newly written data back



Compare after-write values

```
var calls = [
    function(callback) {
        updateDynamo(value, key, function(err, result) {
            callback(err, result.Attributes["val"].N);
        });
    },
    function(callback) { redis.incrbyfloat(key, value, callback); }
];
async.parallel(calls, function(err, result) {
    if(Math.abs(parseFloat(result[0]) - parseFloat(result[1])) > 0)
        logger.error("diffs @ the result"); // do something against it!
    callback(err);
});
```

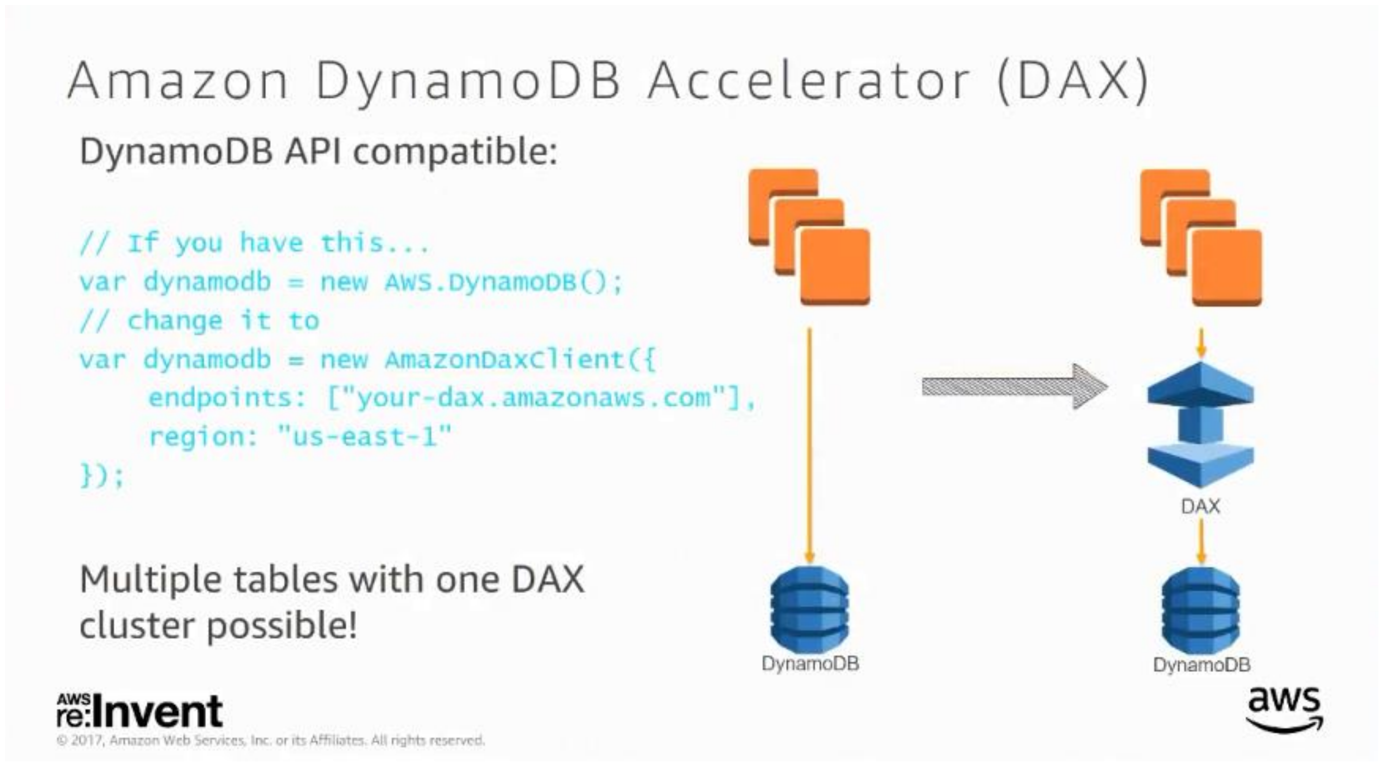
We can also do the same thing with Redis in parallel to when we do the write to DynamoDB. This allows us to know if our cache is out of sync with the Database.



Uncoupled writes/invalidation

Sync/updates via AWS Lambda (uncoupled)
Downside: Small delay for cache updates

In case we don’t want to implement changes in our apps, we can use a lambda function to make sure that our cache is in sync with the database. Our app can continue to write to the database while it can read first from the cache.



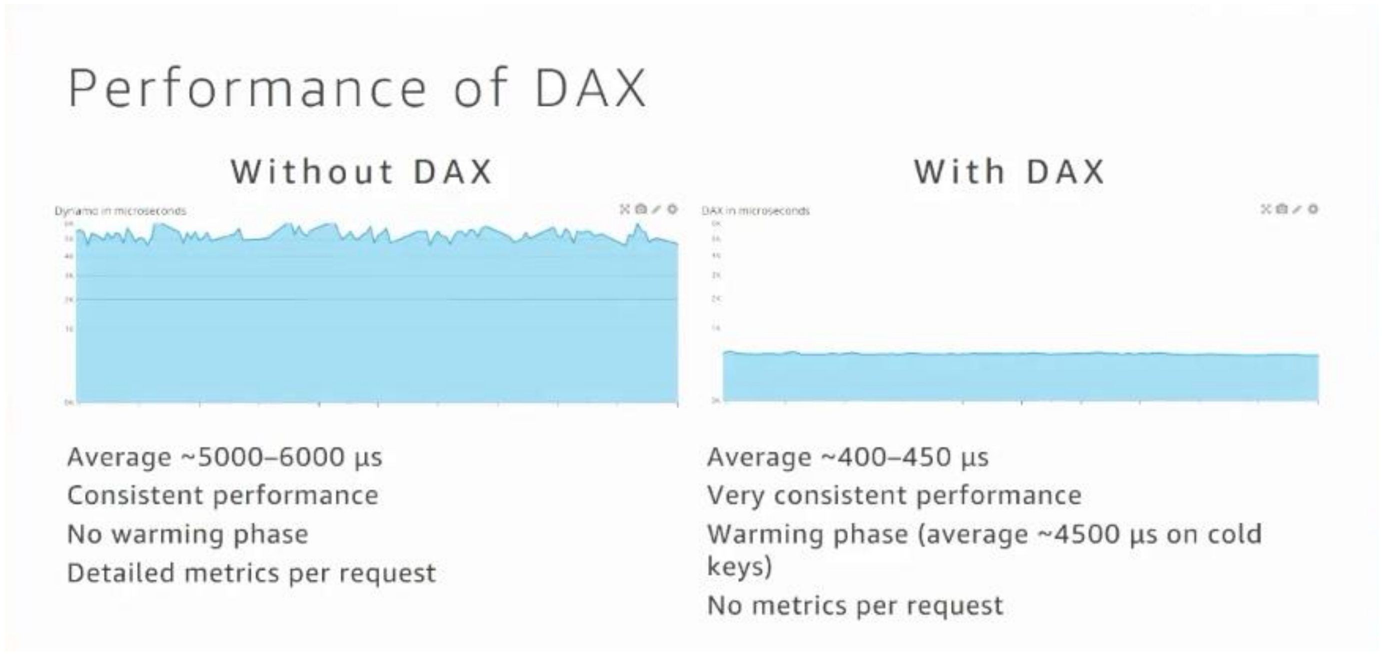
Amazon DynamoDB Accelerator (DAX)

DynamoDB API compatible:

```
// If you have this...
var dynamodb = new AWS.DynamoDB();
// change it to
var dynamodb = new AmazonDaxClient({
    endpoints: ["your-dax.amazonaws.com"],
    region: "us-east-1"
});
```

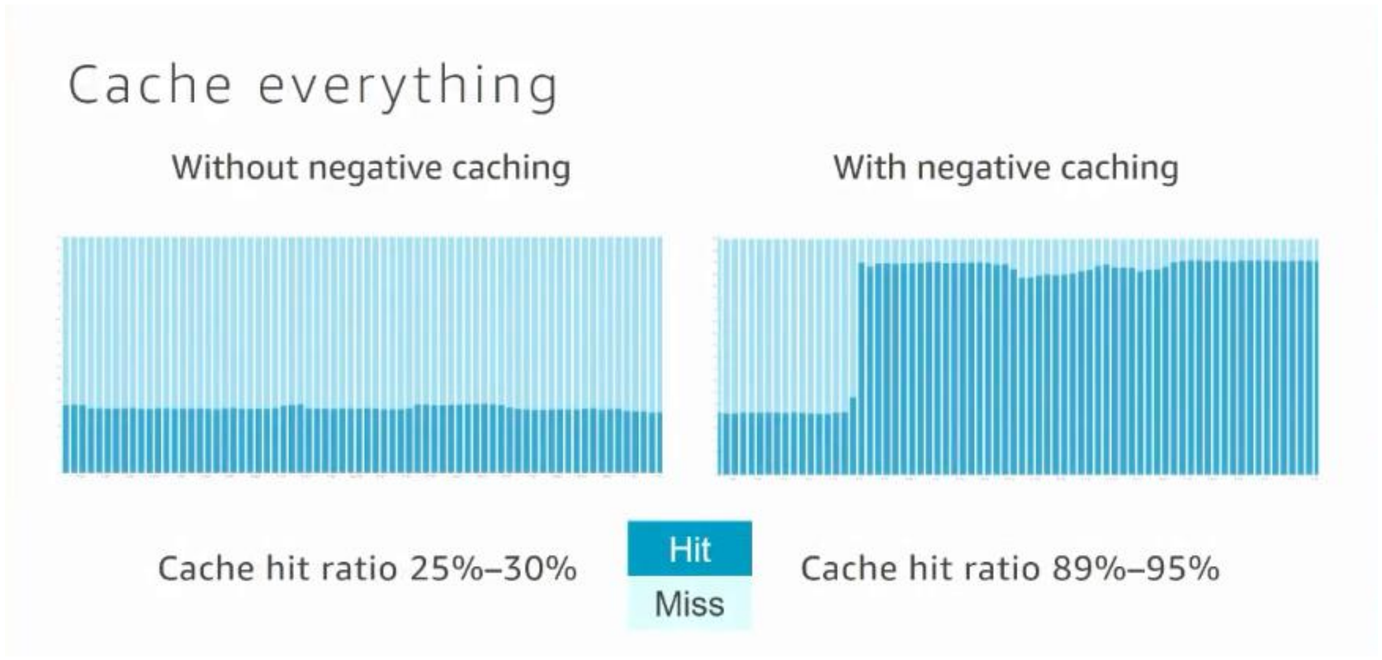
Multiple tables with one DAX cluster possible!

We can instead talk to DAX client (instead of the DynamoDB client) all the time and not worry about the cache anymore, DAX will take care of the caching and the TTL.



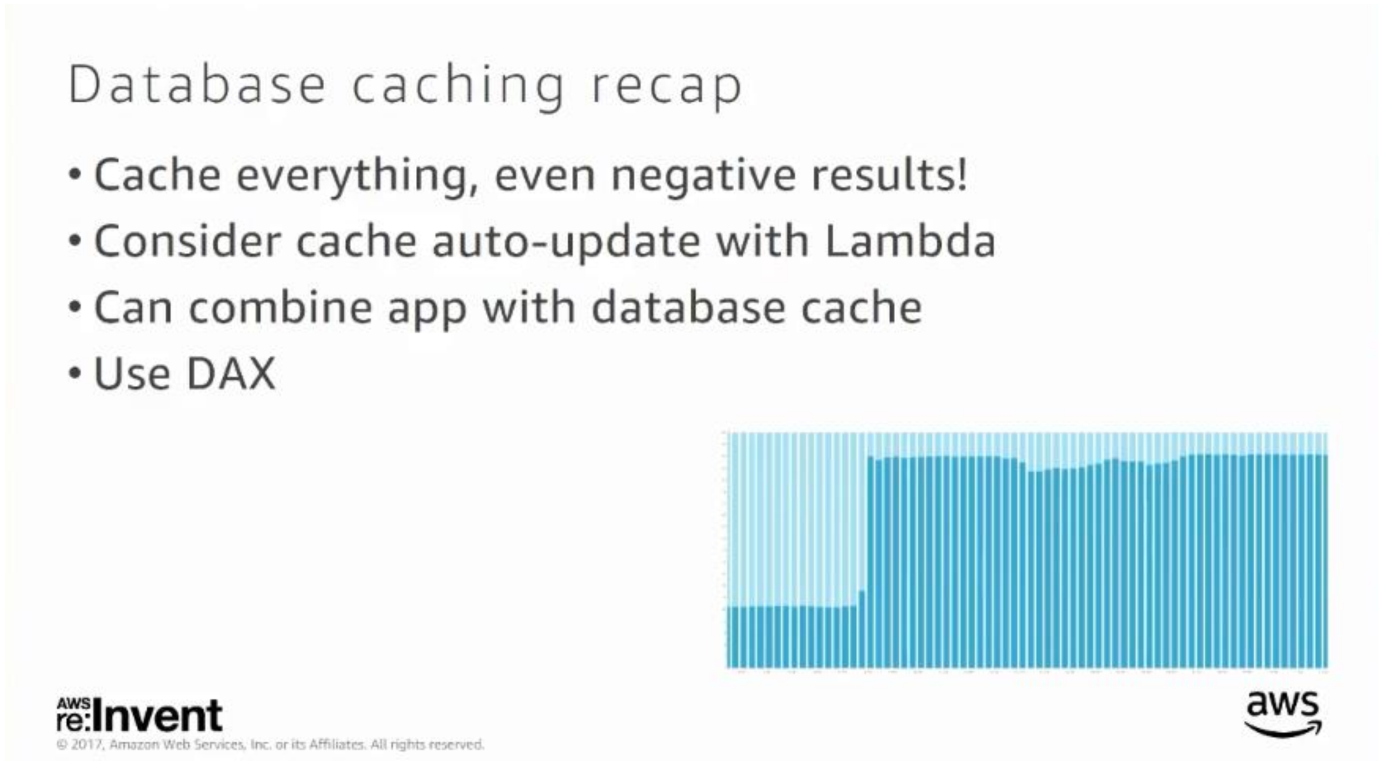
Performance of DAX

Without DAX	With DAX
Average ~5000–6000 µs	Average ~400–450 µs
Consistent performance	Very consistent performance
No warming phase	Warming phase (average ~4500 µs on cold keys)
Detailed metrics per request	No metrics per request



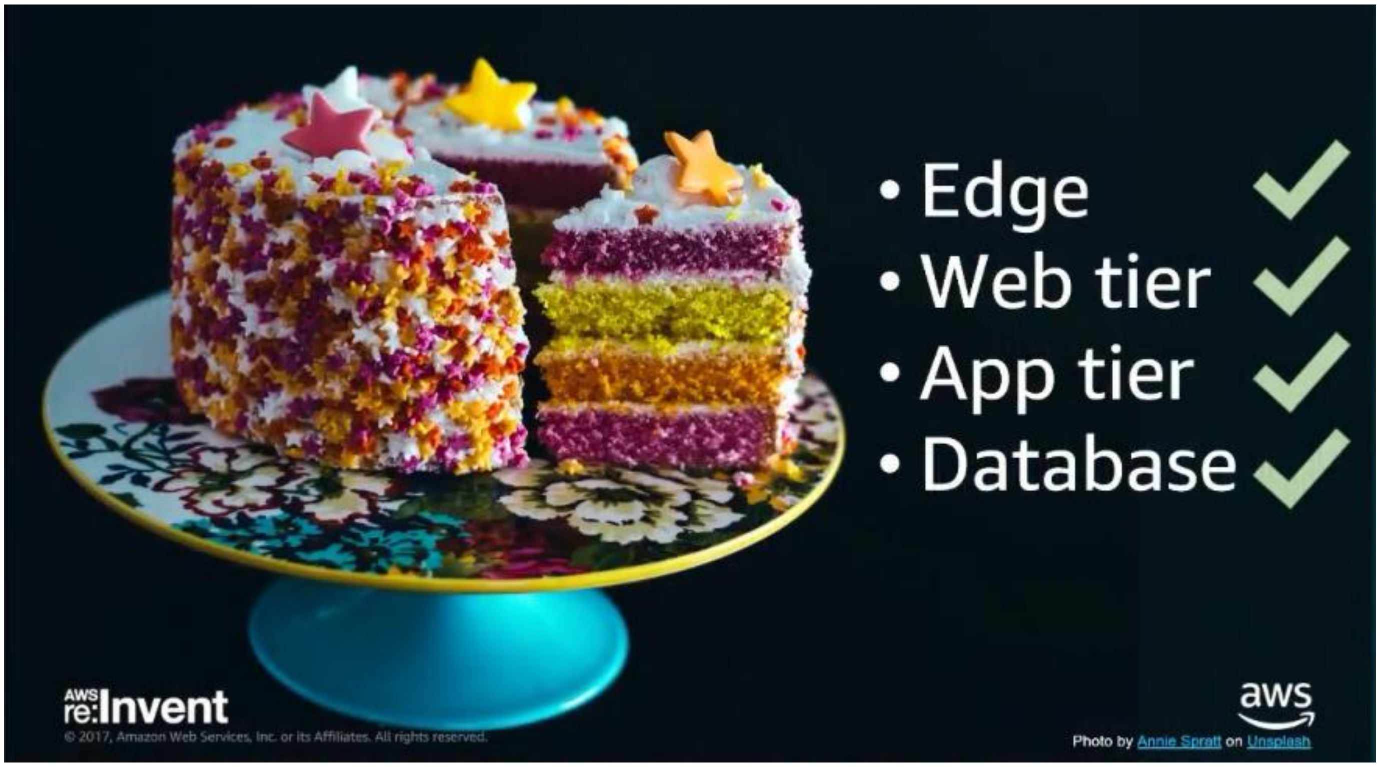
Cache everything

Negative result caching is when we are not saving the 'no data found' result from DynamoDB for some queries, then we keep on asking DynamoDB about those same queries over and over again. You need to cache your negative result caching scenarios.



Database caching recap

- Cache everything, even negative results!
- Consider cache auto-update with Lambda
- Can combine app with database cache
- Use DAX



- Edge ✓
- Web tier ✓
- App tier ✓
- Database ✓

Things to do:

- Check out Amazon CloudFront/API Gateway
- Monitor everything
- Understand your top 20% of users
- Consider adding a web cache
- Explore app caches with Amazon ElastiCache
- Use DAX
- Cache everything!

Speed on!

Photo by Robin Pierre on Unsplash

Liked this talk? Meet us again!

ARC303–Running Lean Architectures: How to Optimize for Cost Efficiency

Tuesday, November 29, 2:30 p.m.

Venetian, Level 2, Venetian E