Horizontal Scalability for Great Success

Nick Barkas
@snb

EuroPython - 22 June 2011

Outline

Introduction

- Spotify
- Kinds of scalability

Designing scalable network applications

- Distributing work
- Handling shared data

Related Spotify tools and methods

- Supervision
- Round-robin DNS and SRV records
- Distributed hash tables (DHT)

Spotify

Introduction

Spotify

What is Spotify?

On-demand music streaming service

Also play your music files, or buy mp3 downloads

Premium subscriptions with mobile and offline support, or free with ads

Create playlists available on any computer or device with Spotify

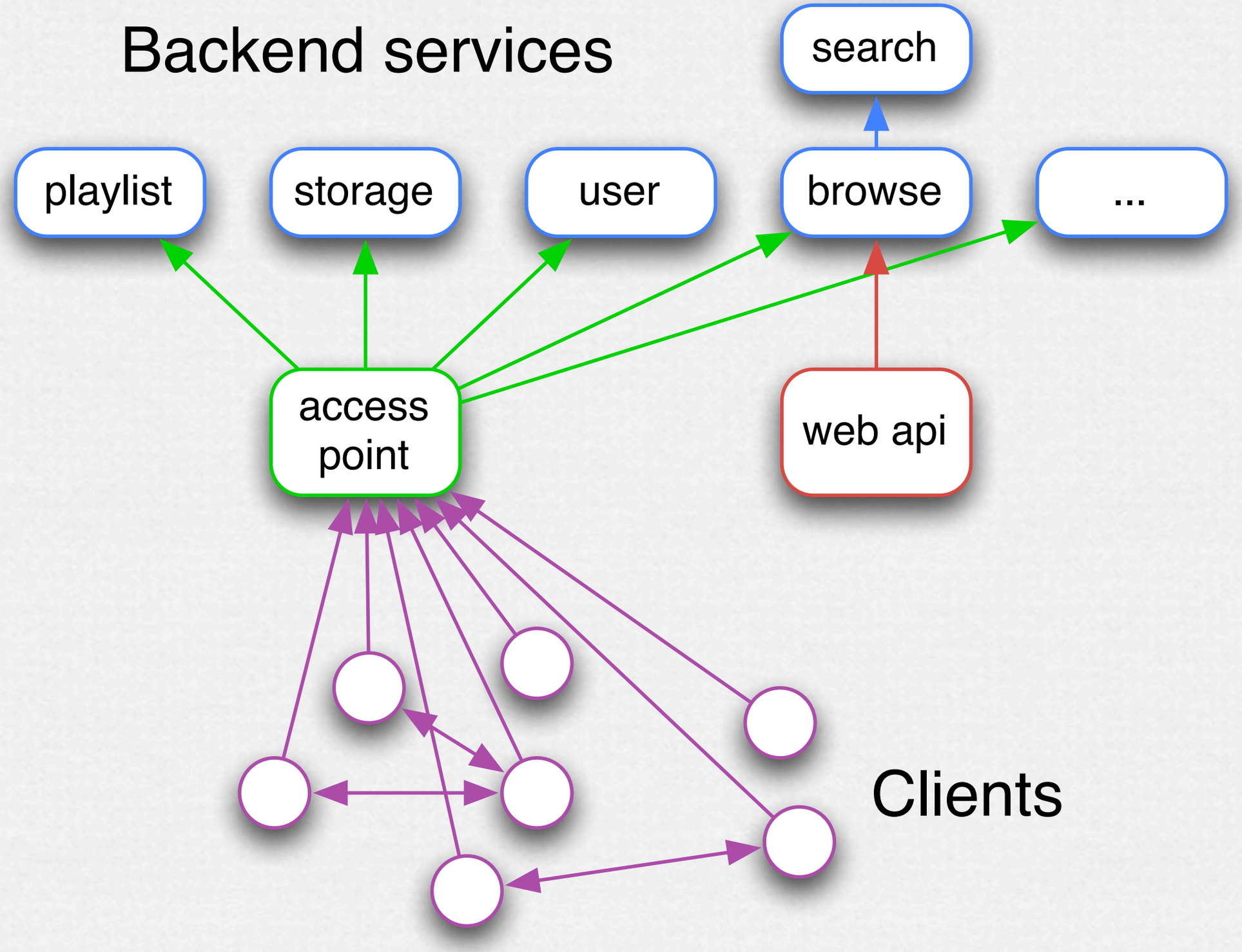
Connect with friends via Facebook and send songs to each other

Sync downloads and your own music files with iPods

Available today in Sweden, Norway, Finland, UK, Netherlands, France, and Spain

Spotify

Overview of Spotify network



Spotify

Scaling vertically: "bigger" machines

+ Maybe no or only small code changes
+ Fewer servers is easier operationally
- Hardware prices don't scale linearly
- Servers can only get so big
- Multithreading can be hard
- Single point of failure (SPOF)

Spotify

Scaling horizontally: more machines

+ You can always add more machines!
+ No threads (maybe)
+ Possible to run in “the cloud” (EC2, Rackspace)
− Need some kind of load balancer
− Data sharing/synchronization can be hard
− Complexity: many pieces, maybe hidden SPOFs
± Fundamental to the application’s design

Spotify

Why horizontal for Spotify?

We are too big

- Over 13 million songs
- And over 10 million users, who have lots of playlists

CPython kind of doesn't give us a choice anyway

- Global interpreter lock (GIL) = no simultaneous threads

Spotify

Why the GIL is kind of a good thing

Forces horizontally scalable design

Multiple cores require multiple Python processes

- Basically the same when scaling to multiple machines

Multi-process apps encourage share-nothing design

- Sharing nothing avoids difficult, slow synchronization

Spotify

Designing scalable network applications

Spotify

Separate services for separate features

The UNIX way: small, simple programs doing one thing well

- Can do the same with network services
- Simple applications are easier to scale
- Can focus on services with high usage/availability needs
- Development is fast and scalable too
  - If well-defined interfaces between services

Spotify

Many instances of each service

N instances/machine where N <= # cores, many machines

Need a way to spread requests amongst instances

- Hardware load balancers
- Round-robin DNS
- Proxy servers (Varnish, Nginx, Squid, LigHTTPD, Apache...)

Spotify

Sharding data

Each server/instance responsible for subset of data

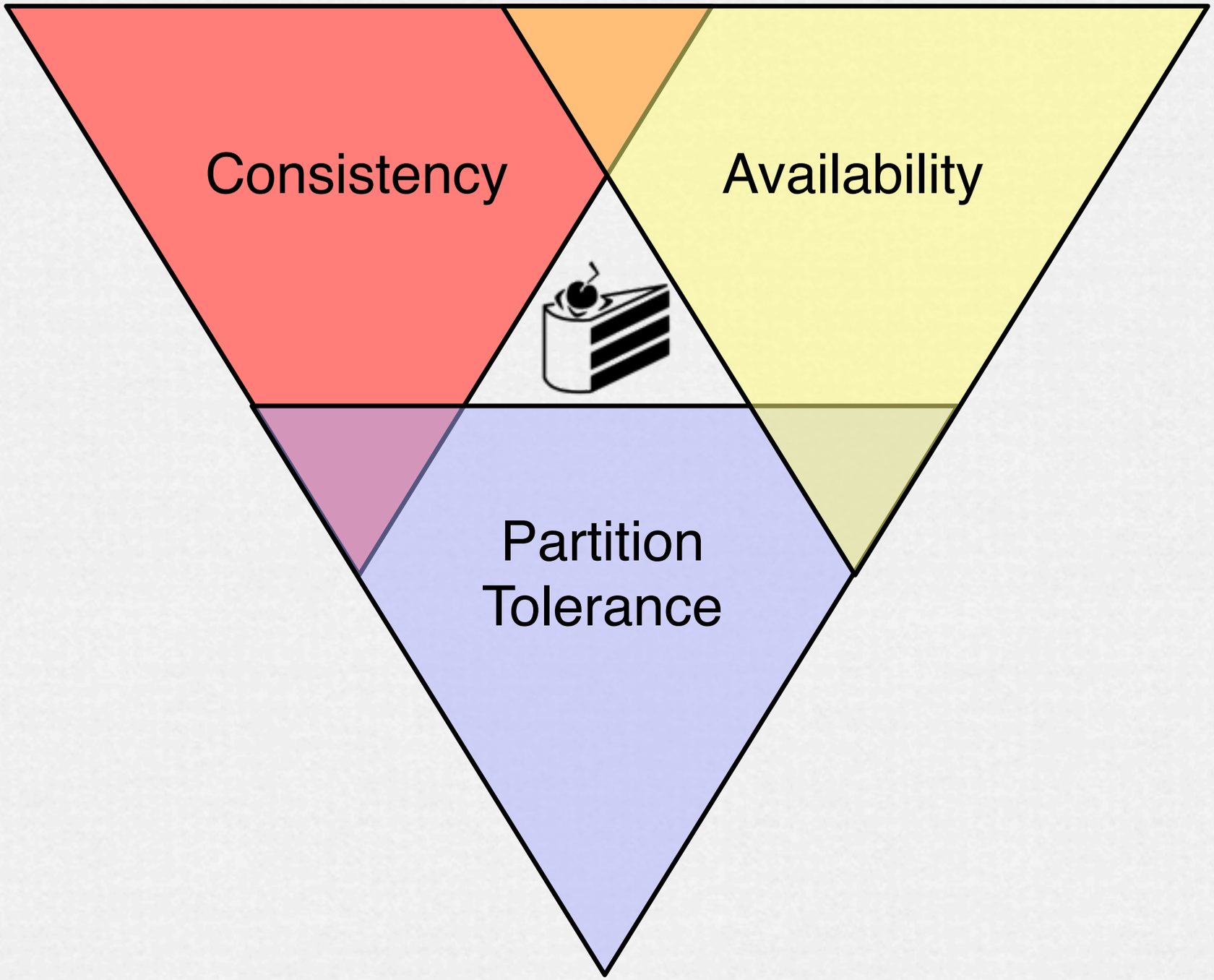
Can be easy if you share nothing

Must direct client to instance that has its data

Harder if you want things like replication

Spotify

Brewer's CAP theorem

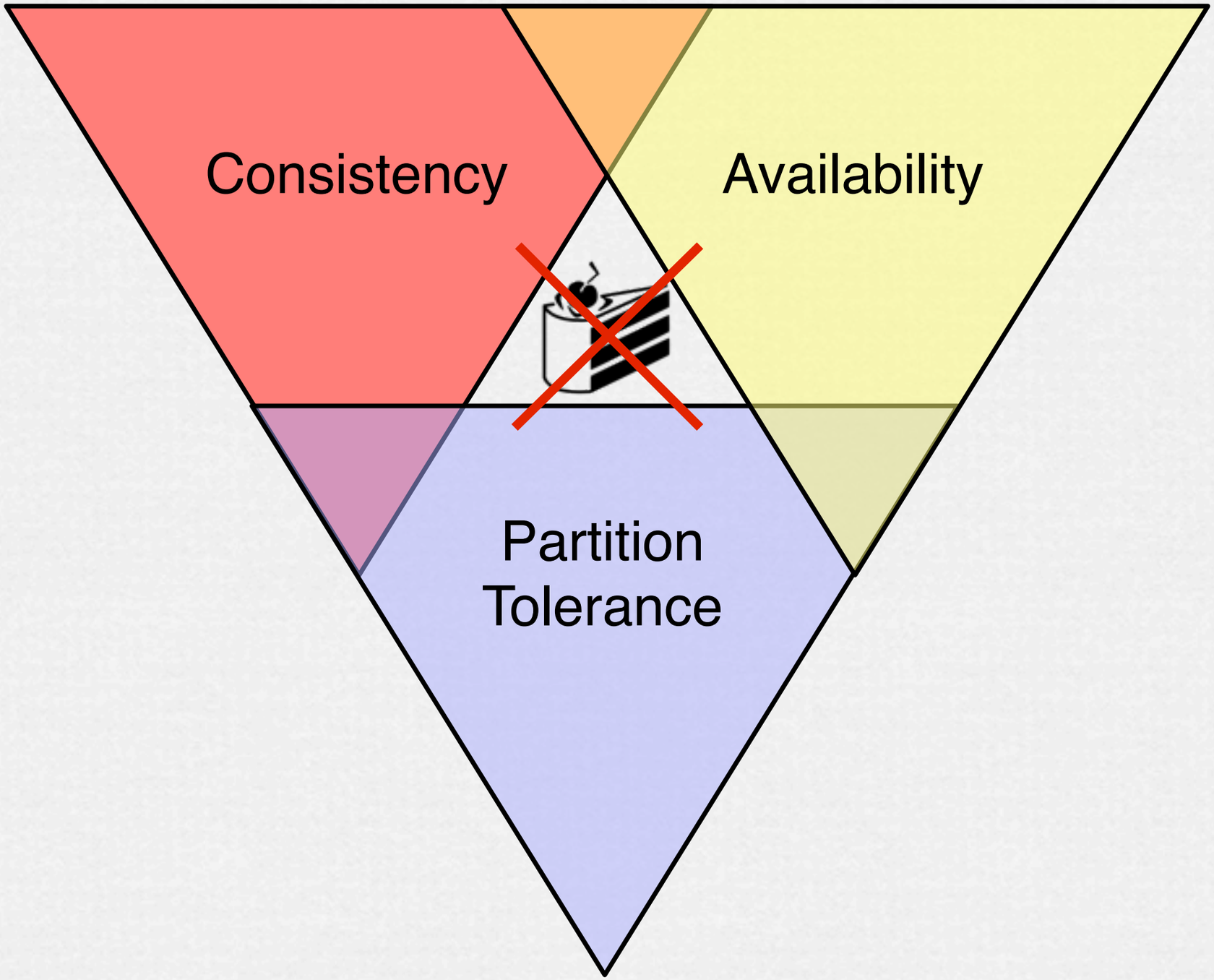


You only get to have one or two

Spotify

Image: http://thecake.info/

Brewer's CAP theorem



You only get to have one or two. The cake is a lie.

Image: http://thecake.info/

Spotify

Eventual consistency

Lots of NoSQLish options work this way

- Reads of just written data not guaranteed to be up-to-date

Example: Cassandra

- Combination of ideas from Dynamo and BigTable
- Available (fast writes, replication)
- Partition tolerant (retries later if replica node unreachable)
- Also can get consistency if willing to sacrifice the other two
- But rather young project, big learning curve

Spotify

Sometimes you need consistency

Locking, atomic operations

- Creating globally unique keys, e.g. usernames
- Transactions, e.g. billing

PostgreSQL (and other RDBMSs) are great at this

- Availability via replication, hot standby masters
- Store only what you absolutely must in global databases

Spotify

Tips for many instances of a service

Processor affinity

Watch out for connection limits (e.g. in RBDMS)

Lots of processes can share memcached

OS page cache for read-heavy data

Spotify

Related Spotify tools and methods

Spotify

Supervision

Spotify developed daemon that launches other daemons

- Usually as many instances as cores - 1
- Restarts supervised instance if one fails
- Also restarts all instances of an application on upgrade

See also: systemd

Spotify

Finding services and load balancing

Each service has an SRV DNS record

- One record with same name for each service instance
- Clients (AP) resolve to find servers providing that service
- Lowest priority record is chosen with weighted shuffle
- Clients must retry other instances in case of failures

Spotify

Finding services and load balancing

Example SRV record

```
_frobnicator._http.example.com. 3600  SRV  10   50   8081 frob1.example.com.
```

name *TTL* *type* *prio* *weight* *port* *host*

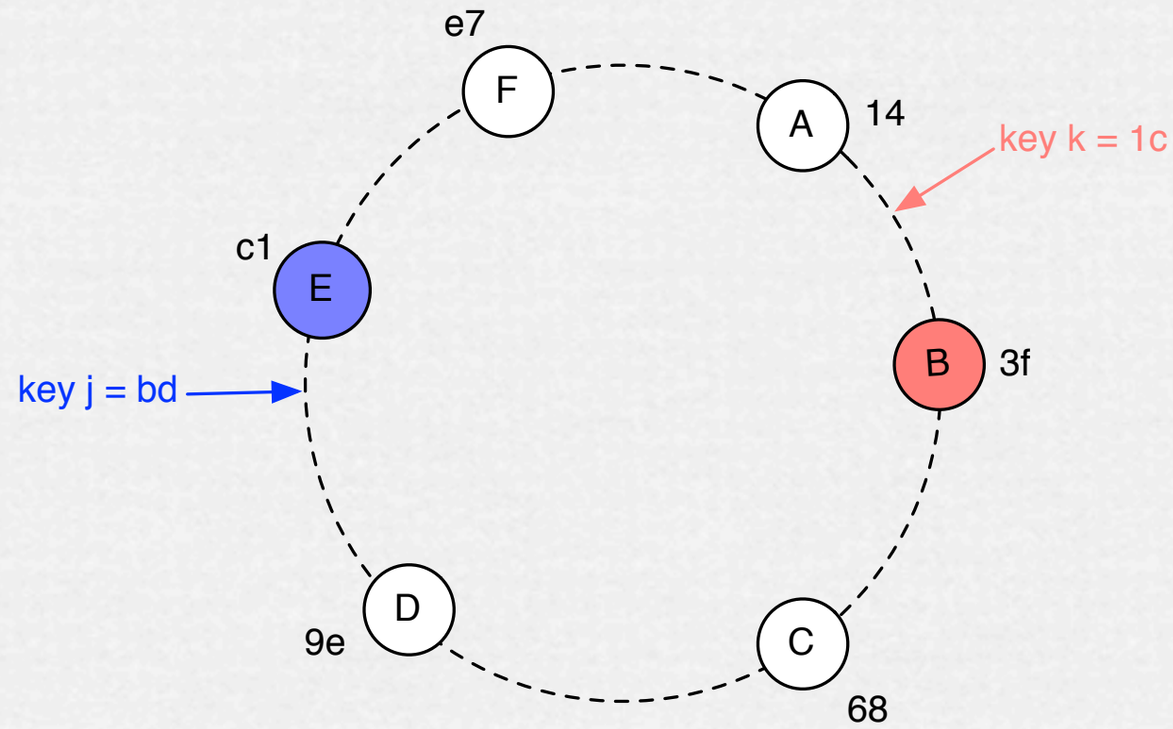
Sometimes also use Varnish or Nginx for HTTP services

- Can have caching too

Spotify

Distributed hash tables (DHT) in DNS

Service instances correspond to a range of hash keys



- Distributes data among service instances
  - Instance B owns keys in (14, 3f], E owns (9e, c1]
- Redundancy? Hash again, write to replica instance
- Must transition data when ring changes
- Good also for non-sharded data: cache locality

Spotify

DHT DNS record examples

Ring segment, per instance

```
tokens.8081.frob1.example.com. 3600 TXT "00112233445566778899aabbccddeeff"
```

name: tokens.port.host. *TTL* *type* *last key*

Configuration of DHT

```
config._frobnicator._http.example.com.   3600 TXT "slaves=0"
```

name: config.srv_name. *TTL* *type* *no replication*

```
config._frobnicator._http.example.com.   3600 TXT "slaves=2 redundancy=host"
```

name: config.srv_name. *TTL* *type* *three replicas on separate hosts*

Spotify

Further reading about DHTs

"Chord: A scalable peer-to-peer lookup service for internet applications"

- http://portal.acm.org/citation.cfm?id=964723.383071

"Dynamo: Amazon's Highly Available Key-value Store"

- http://portal.acm.org/citation.cfm?id=1294281

Spotify

One last thing to remember

/dev/null is web scale!

Spotify

Image: http://www.xtranormal.com/watch/6995033/

Questions?

Or write me something: snb@spotify.com, @snb on Twitter

Spotify

Thank you