
UBM
TechWeb
GDC
Online
Game Developers Conference® Online
October 5-8, 2010 | Austin, TX
Visit www.GDCOnline.com for more information

scalability for social games

dr. robert zubek

top social game developer

YoVille™ PetVille™ zyngapoker™ FarmVille™

Mafia Wars™ Treasure Isle™ Café World™ FrontierVille™

frontierville

growth example

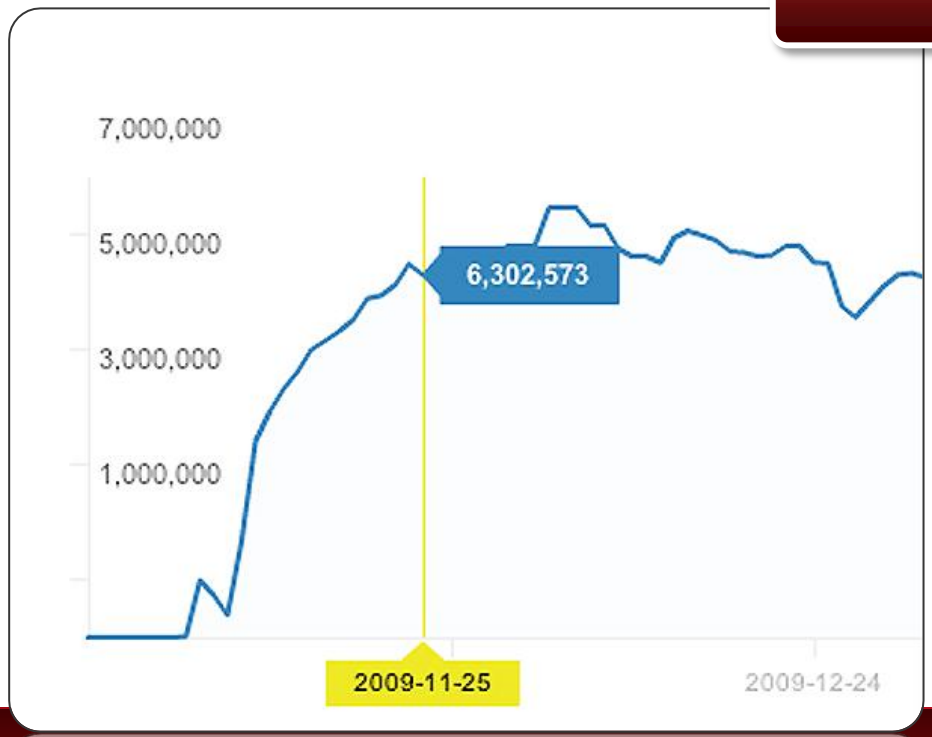
2m daus
over two days

source: developeranalytics.com

fishville

growth example

6m daus
in its first week

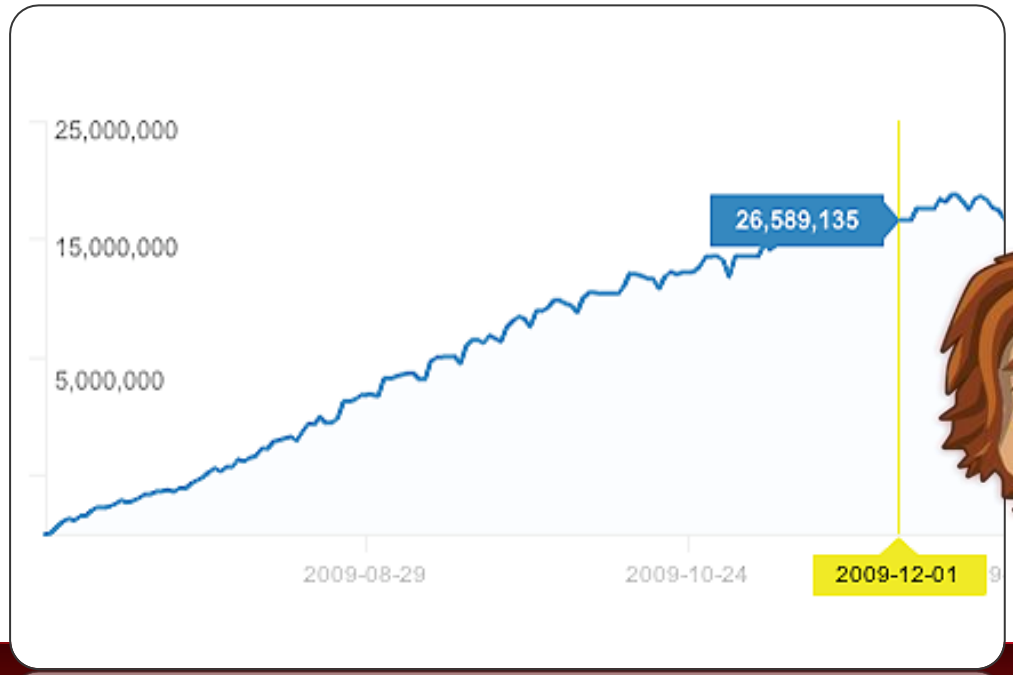


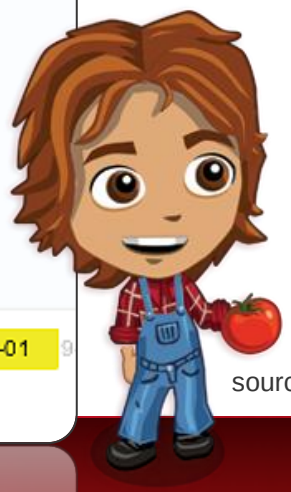
source: developeranalytics.com

farmville

growth example

25m daus
over five months





source: developeranalytics.com

talk overview

introduce game developers to best practices for large-scale web development

talk overview

introduce game developers to best practices for large-scale web development

servers

part I. game architectures

part II. scaling solutions

three major types

web server stack		web + mmo stack
html	flash	flash

client side

flash

- high production quality games
- game logic on client
- can keep open socket

html + ajax

- the game is “just” a web page
- minimal sys reqs
- maybe some Flash

Server Side

web stack

- usually based on a LAMP stack
- game logic in PHP
- HTTP communication

mixed stack

- game logic in MMO server (eg. Java)
- web stack for everything else

why web stack?

- HTTP scales very well
- stateless request/response
- easy to load-balance across servers
- easy to add more servers
- all the good stuff about turnkey solutions ☺

some departures from web

- games are:
  - very stateful
  - write-heavy
  - sensitive to order of execution
- HTTP is request/response
  - data push is desirable but hard

mmo servers

- not the focus of this talk
- socket servers with game logic:
  - persistent socket connection per client
  - live game support: chat, live events
  - supports data push
  - keeps game state in memory

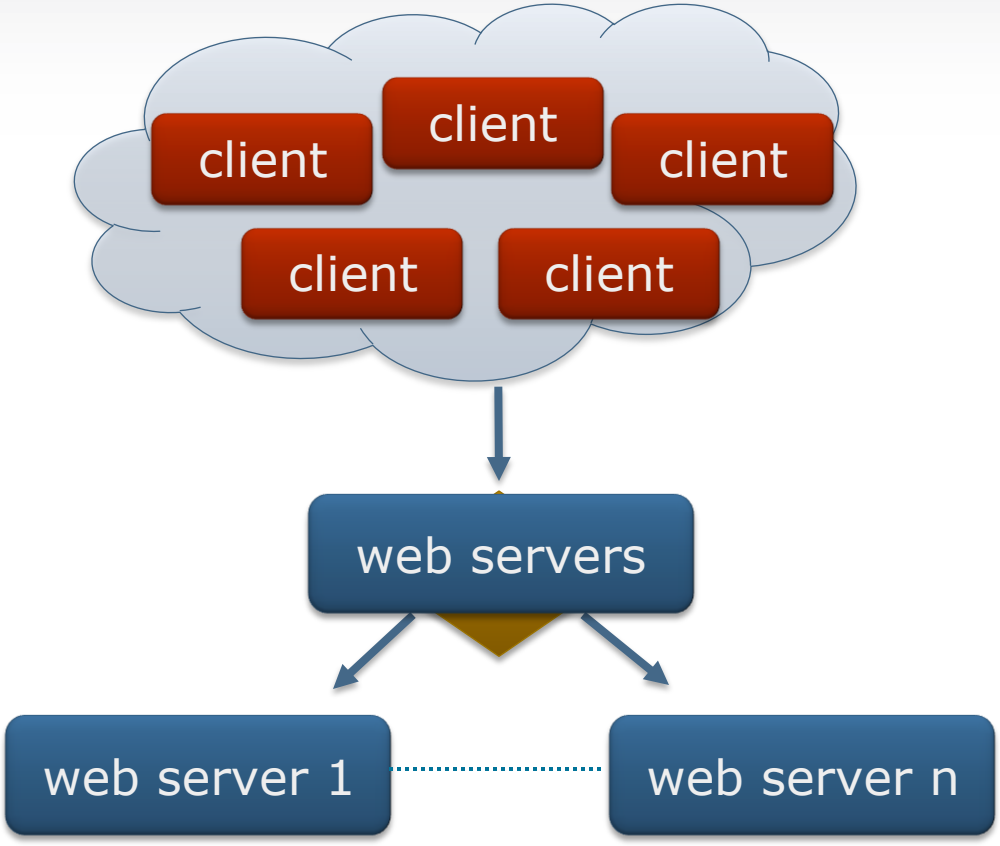
social network integration

- “easy” because not related to scaling ☺
- calling the host network to get a list of friends, post something, etc.
- networks provide REST APIs, and sometimes client libraries

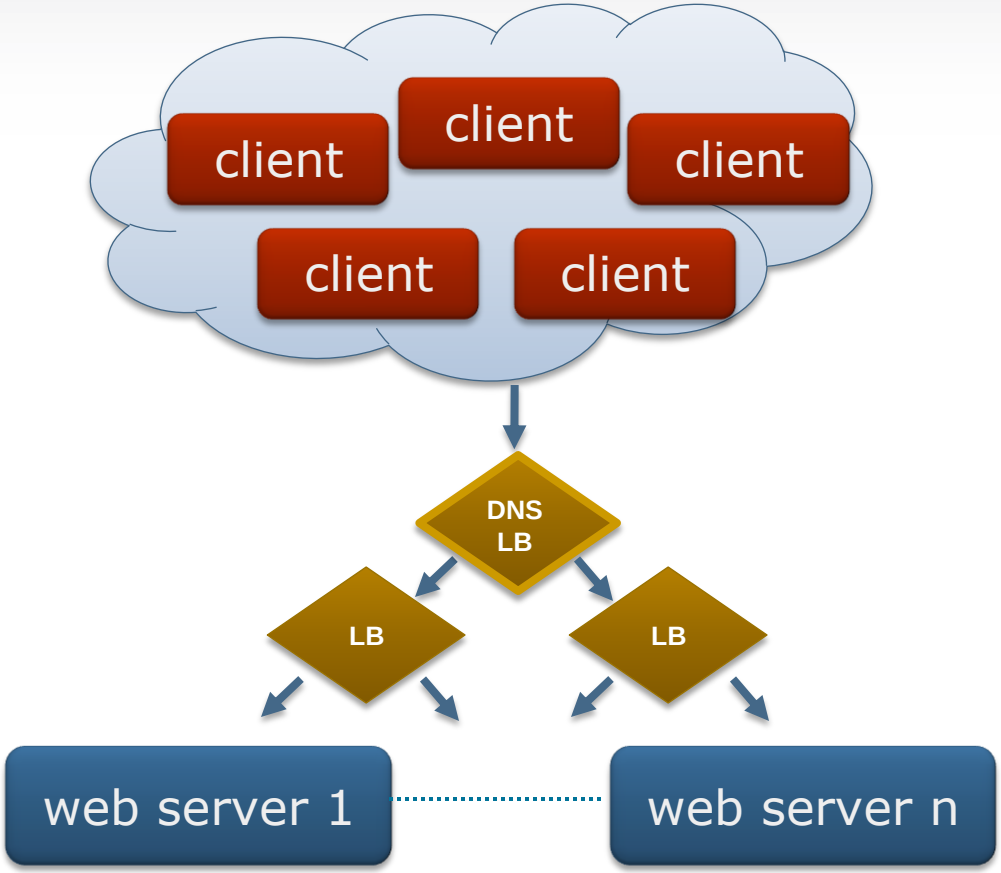
part II. scaling solutions

1. web server scaling
2. web programming model
3. database
4. memcache and caching

1. web server scaling



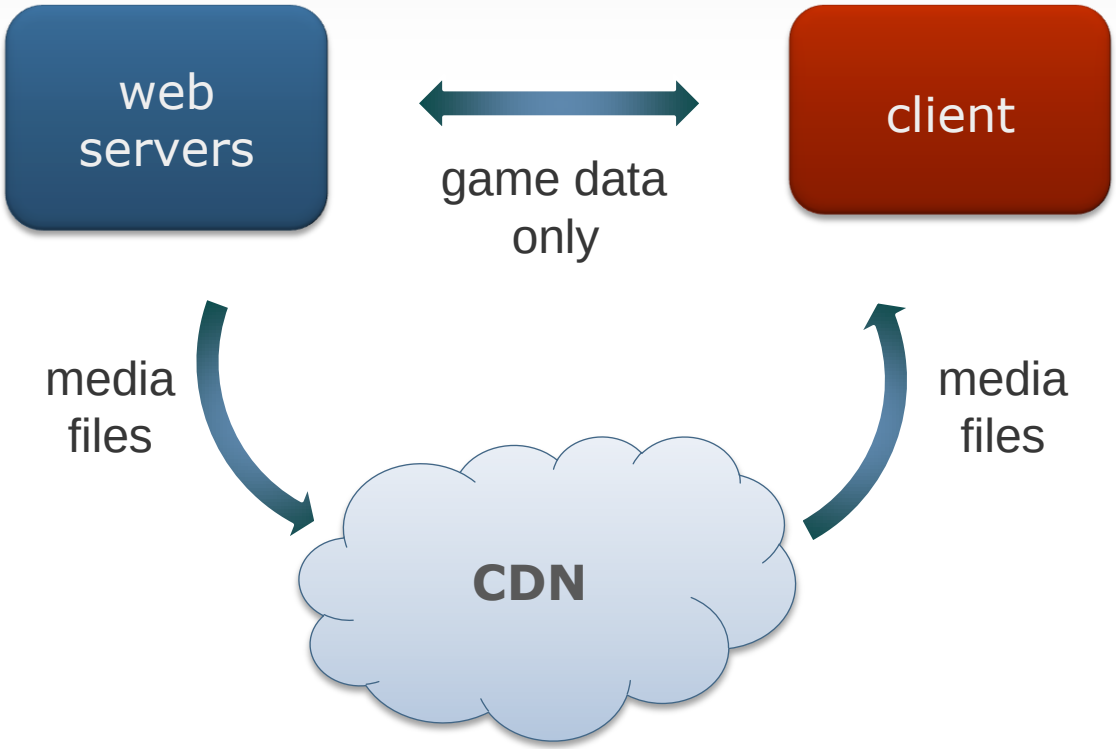
1. web server scaling



server affinity issues

- load balancing works by spreading requests across different servers
- where do player requests go:
  - same server every time?
  - different server each time?
- affinity would make programming easier
  - but it's hard to guarantee
  - don't assume affinity

scaling content delivery



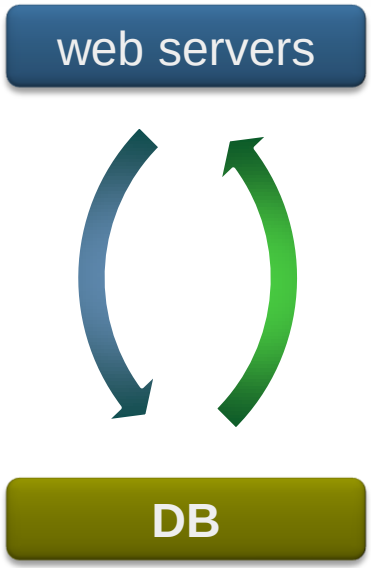
2. web programming model

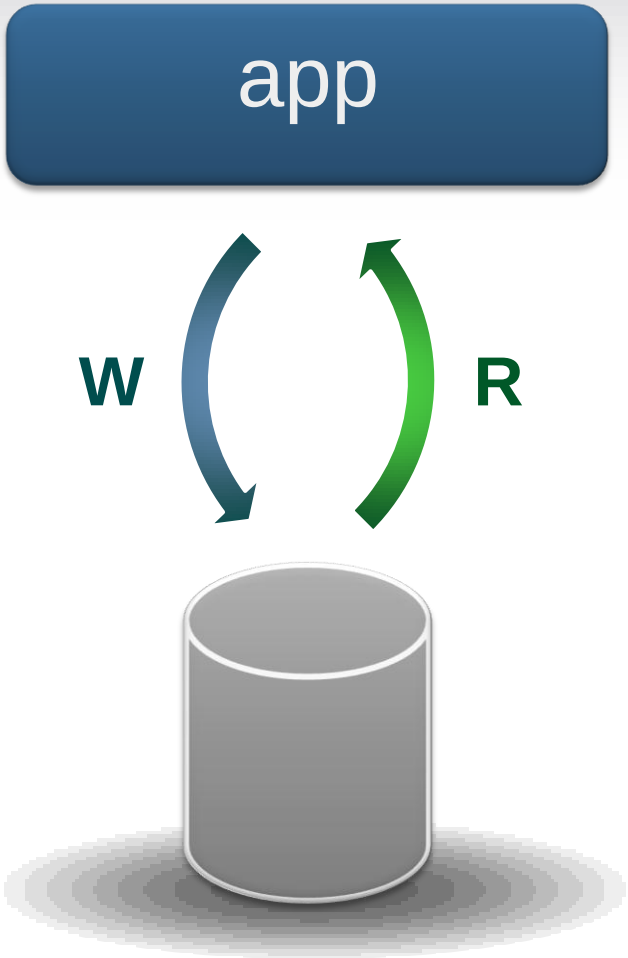
- LAMP example:
  - apache server to handle HTTP requests
  - PHP to implement game logic
- each request is completely separate:
  - spins up Apache process + PHP
  - processes request, produces results
  - cleans up and finishes

PHP for game logic

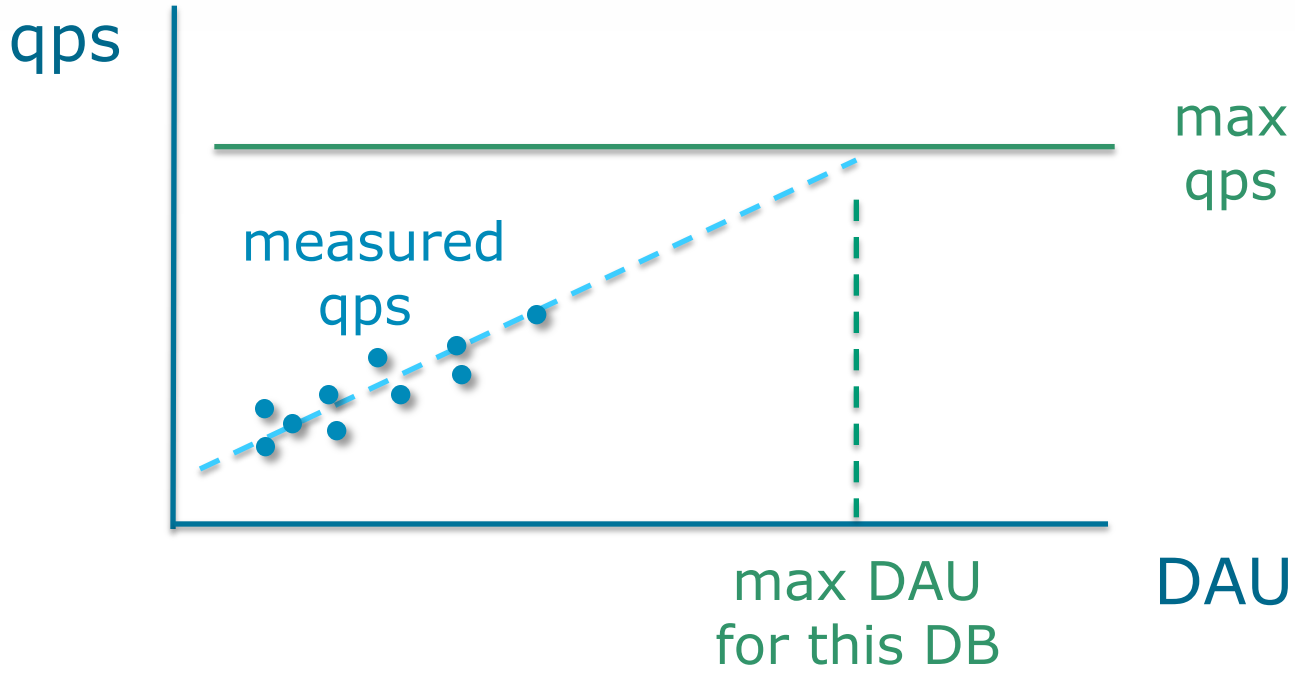
- we tend to use PHP
  - facebook used to only provide PHP libs
  - mature integration into server stack
- but use whatever works best for your team
- regardless of language, stateless web programming model is a scalability win

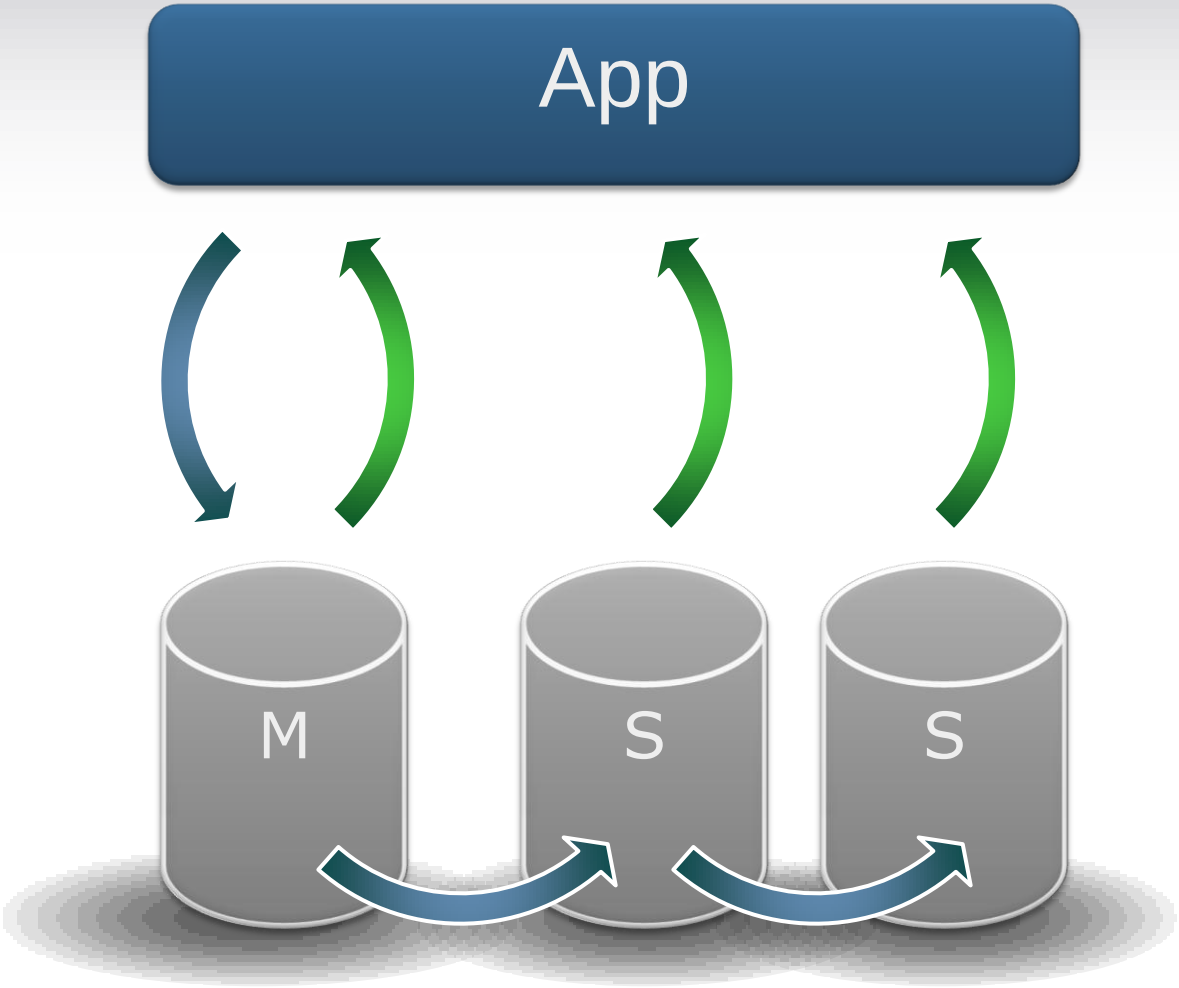
3. database

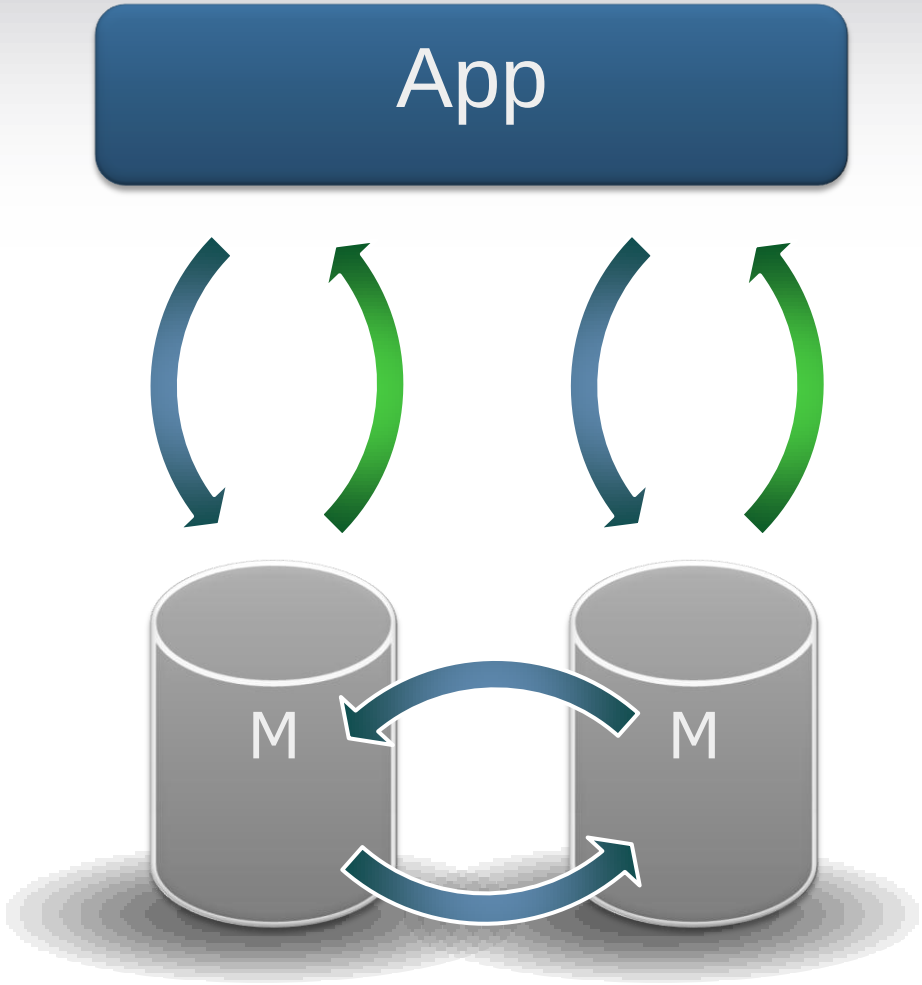


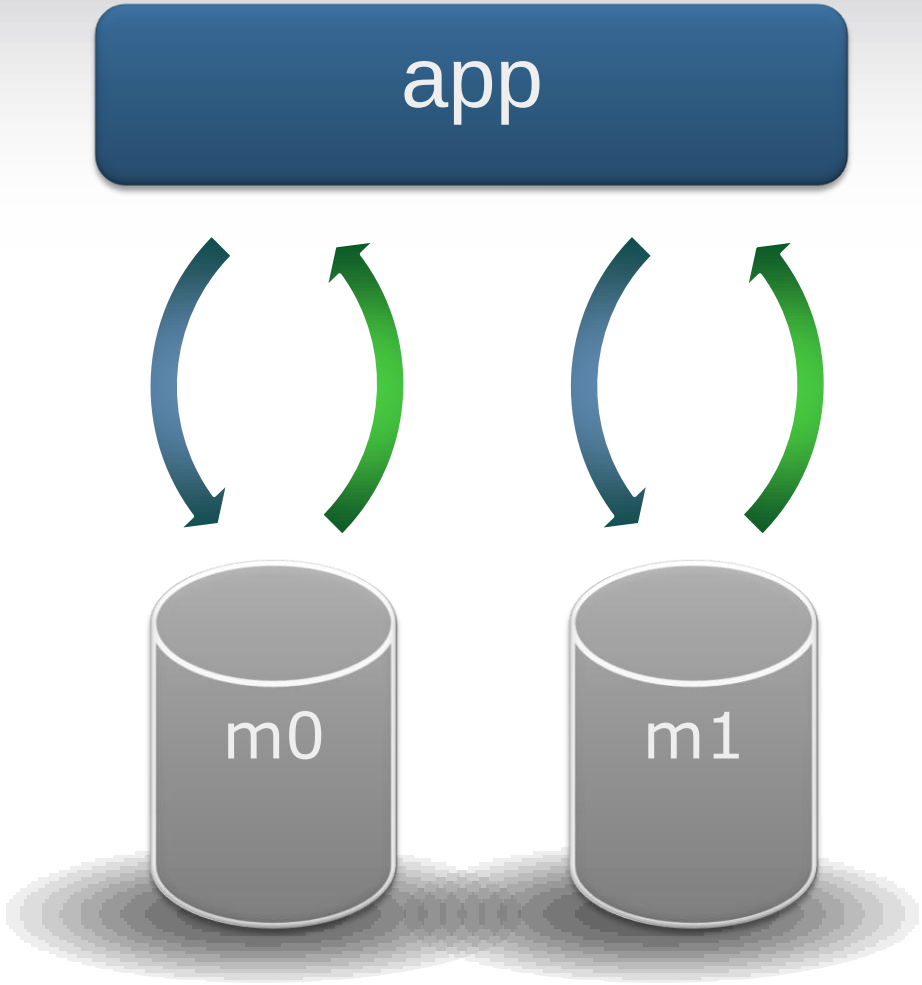

app
W
R

Measuring DB limits




App
M
S
S


App
M
M


app
m0
m1

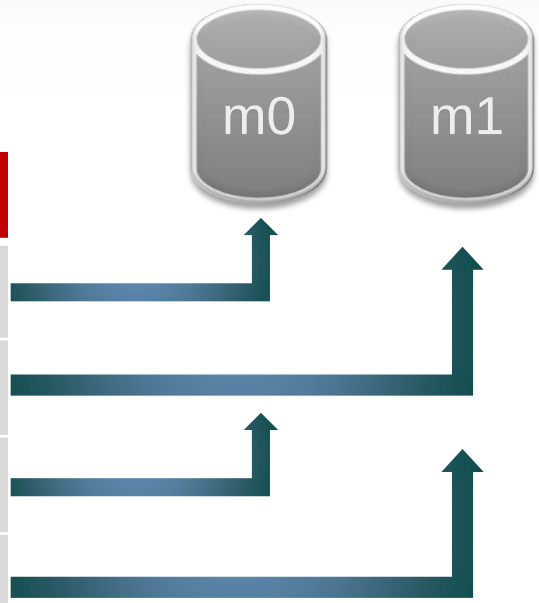
how to partition data?

- two most common ways:
  - vertical – by table
    - easy but doesn't scale with DAUs
  - horizontal – by row
    - harder to do, but gives best results!
    - different rows live on different DBs
    - need a good mapping from row # to DB

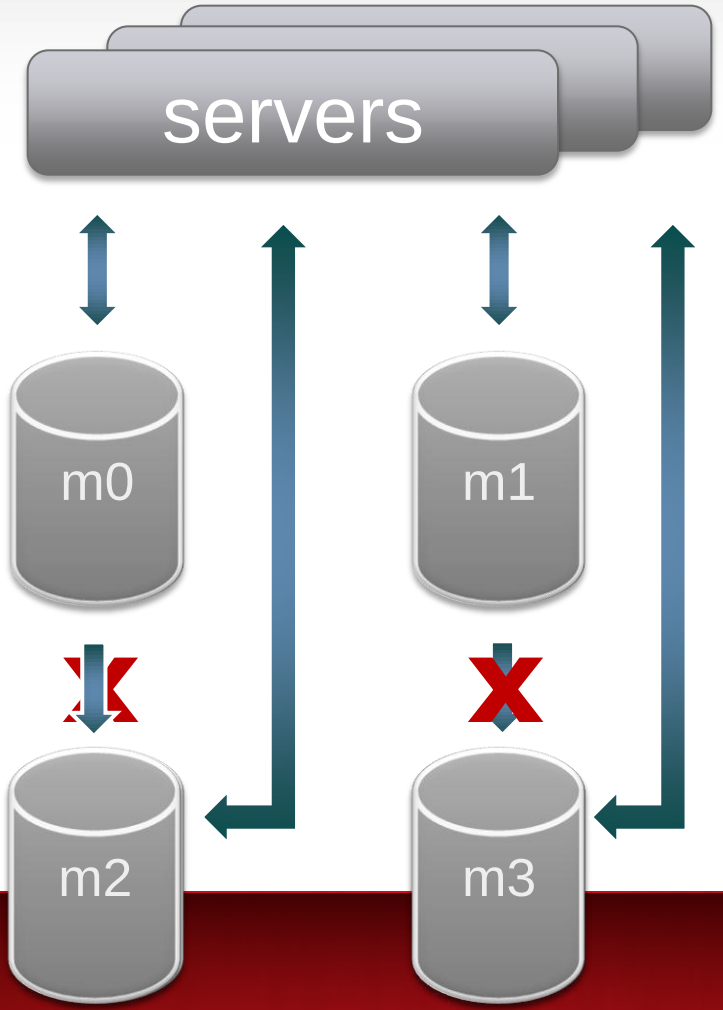
row striping

- row-to-shard mapping:
- primary key modulo # of DBs
- like a “logical RAID 0”
- more clever schemes exist, of course

id	data
100	foo
101	bar
102	baz
103	xyzzy



scaling out your shards



sharding in a nutshell

- It’s just data striping across databases (“logical RAID 0”)
  - there’s no automatic replication, etc.
  - no magic about how it works
  - that’s also what makes it robust, easy to set up and maintain!

sharding surprises

be careful with joins

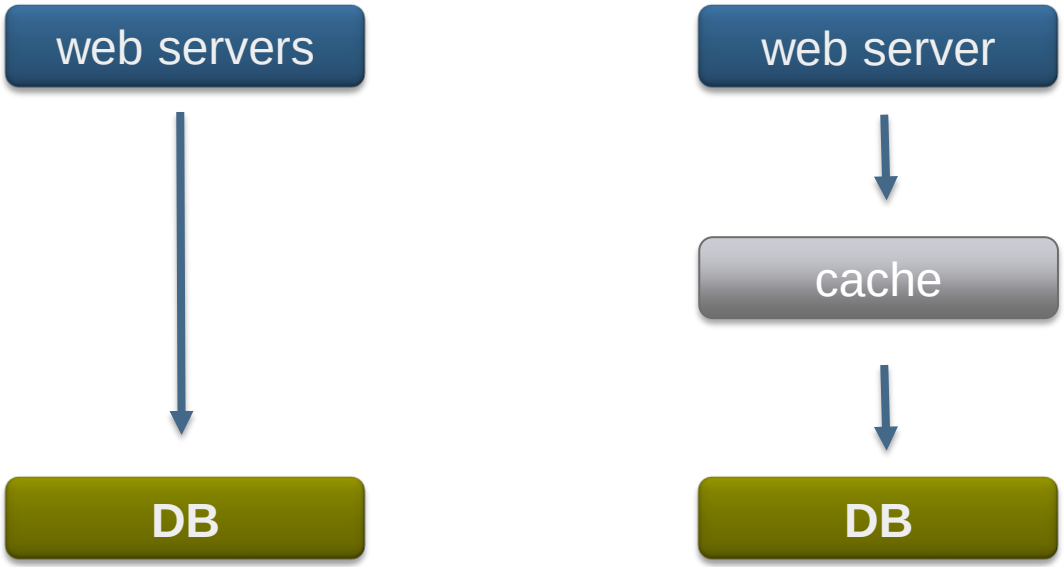
- can’t do joins across shards
- instead, do multiple selects, or denormalize your data

sharding surprises

skip transactions and foreign key constraints

- CPU-expensive; easier to pay this cost in the application layer (just be careful)
- the more you keep in RAM, the less you'll need these

4. caching



4. caching

- speed up access to commonly used data
- prevents you from hitting the DB all the time

memcached

- very popular in-memory cache.
- stores simple key-value pairs
  - set *[key] [data] [expiration]*
  - get *[key]* => *[data]*
  - add / delete / etc.
- atomic check-and-set!
  - cas *[key] [data]*
  - Useful for synchronization

memcached

- what to put there?
  - structured game data
    - eg. “uid_123” => {name:”Robert”, …}
  - use CAS to implement mutexes for concurrent actions
    - eg. make sure two web servers aren’t updating the same data at the same time
    - standard concurrency problem

storage model

- internal storage: pools of fixed-size elements
  - pools of 1kb objects, 2kb objects, etc .
  - called “slabs” in memcache parlance
  - makes for very fast allocation – at a price

caveats

- unexpected evictions
  - you can run out of room in a slab before you run out of room globally
  - then oldest data will get evicted, even if it hasn't reached expiration date
- small game changes can increase evictions, which will increase DB load

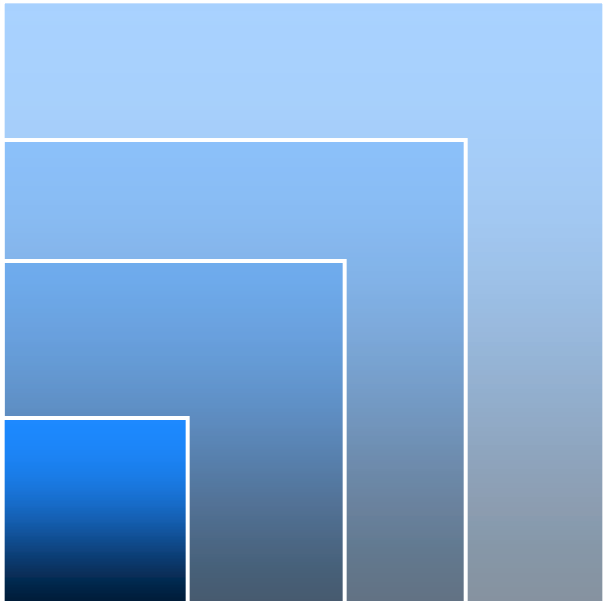
memcached

- easy to scale horizontally
  - Comes with key-based horizontal sharding
- it's a cache, not a DB
  - no persistence! No fallover!

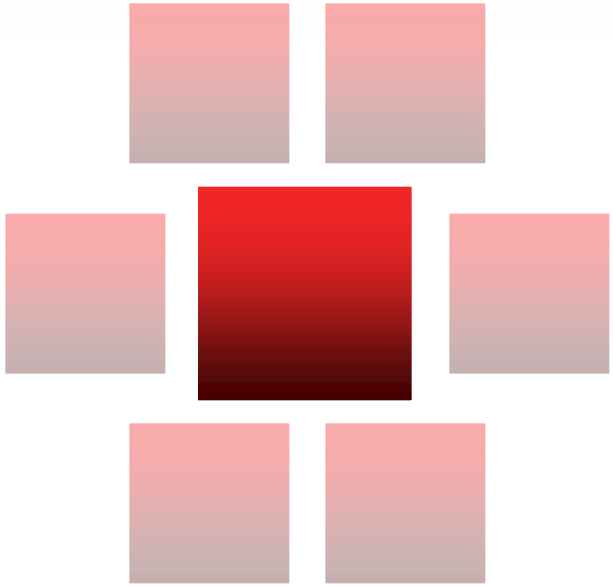
scaling summary

scaling

growth example



i need a bigger box

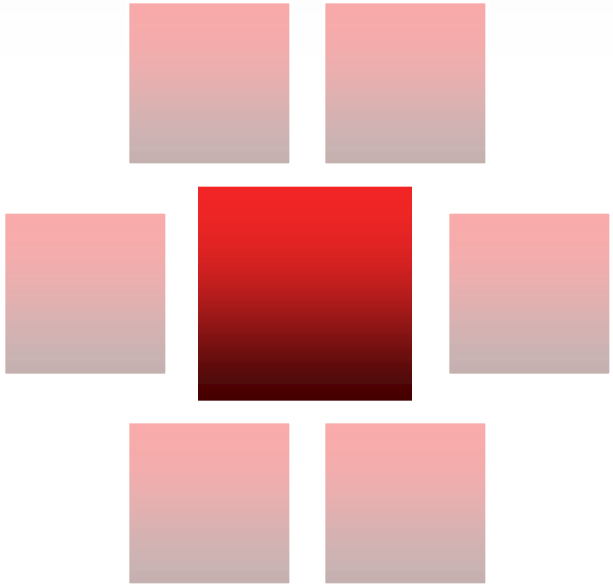


i need more boxes

scaling

scale out has been a clear win:

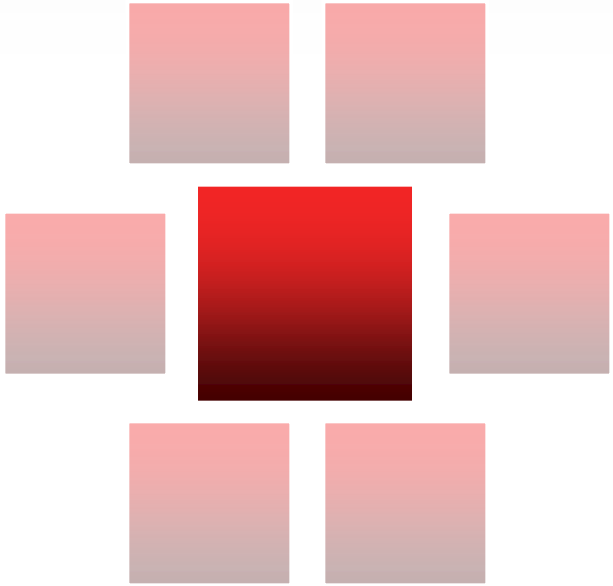
- at some point you can't get a box big enough, quickly enough
- much easier to add more boxes
- but: requires architectural support in all layers



scaling

you have to scale out everywhere

- web
- caches
- DB



The End


zynga


FRONTIER
VILLE


FARMVILLE


YoVille


Café
World


PETVILLE


Treasure Isle


zyngapoker


FishVille


Mafia Wars