Application Architecture Overview

Dean Dieker
Software Engineer, Tapjoy

About Dean...

- Engineering degree from Olin College in 2007 (Bioengineering concentration)
- co-founded a YC company in college
- systems engineer for a defense contractor
- instructor / program director at a karate school
- Startup Institute alum (summer ‘12)
- software engineer at Tapjoy
- teacher/mentor for SI RampUp

What is a...

- Server
- Database
- Cache
- Load Balancer
- Service
- Queue

What does it mean to...

- Scale Vertically
- Scale Horizontally

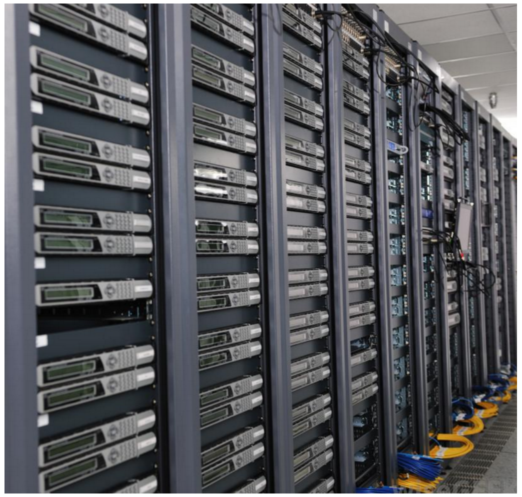
Server

Servers are referred to as ‘machines’ or ‘boxes’. If you develop locally, you might imagine a server looking like this...

http://www.visualpharm.com
http://images.wisegeek.com
http://en.wikipedia.org/wiki/Server_(computing)

Server

When in reality it looks like this...



http://www.visualpharm.com
http://images.wisegeek.com
http://en.wikipedia.org/wiki/Server_(computing)

Server

But we can still think of it like this.

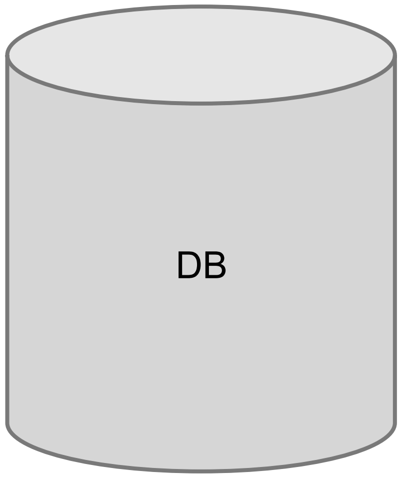
A server is just “a system (software and suitable computer hardware) that responds to requests across a computer network.” (Wikipedia)

Basically we can imagine a server as being a place where our application, database, webserver, and all dependencies (e.g. Python) are configured in such a way to actually respond to and process network requests.

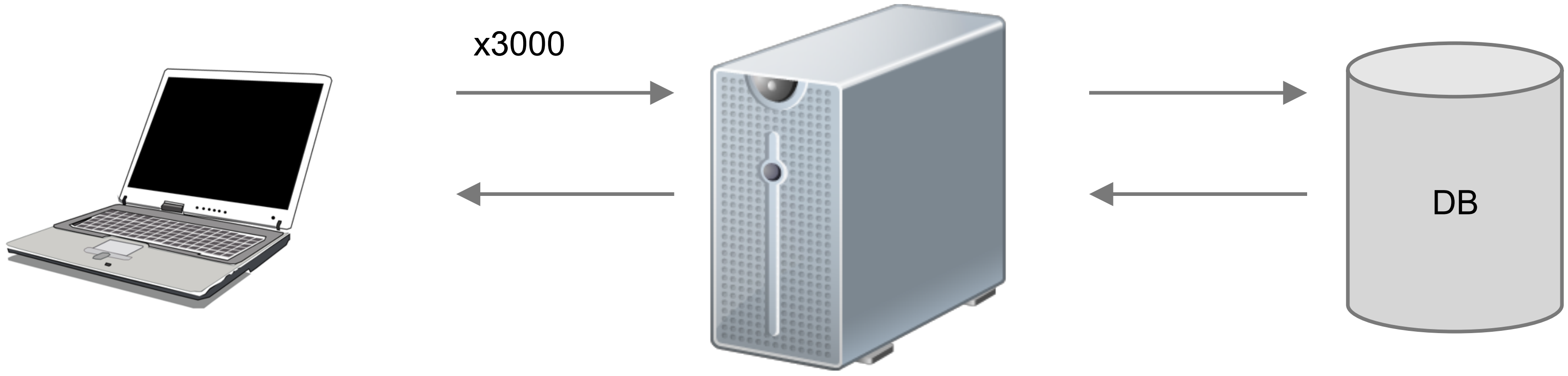
http://www.visualpharm.com
http://images.wisegeek.com
http://en.wikipedia.org/wiki/Server_(computing)

Database

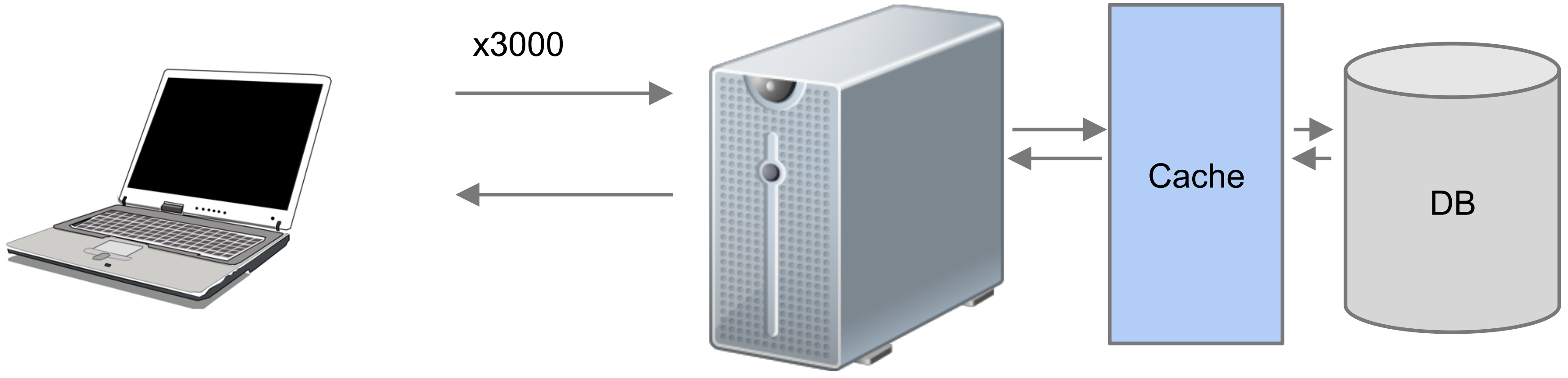
- **Relational**- mySQL, SQLite 3
- **Non-Relational**- MongoDB, CouchDB
- **Key - Value**- Redis, Riak



Cache



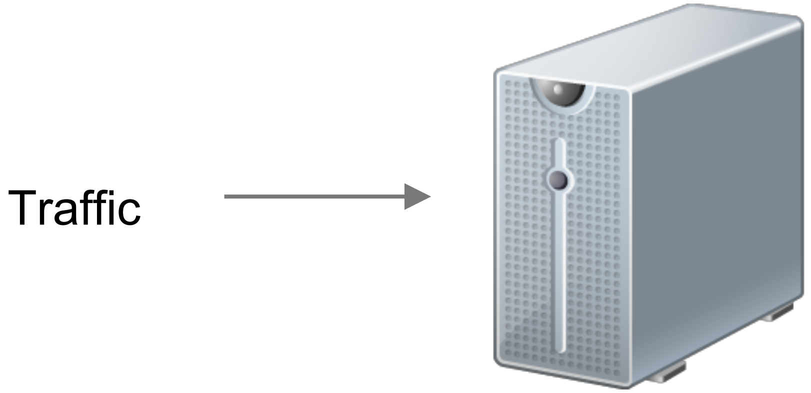
Cache



Caches are great for:

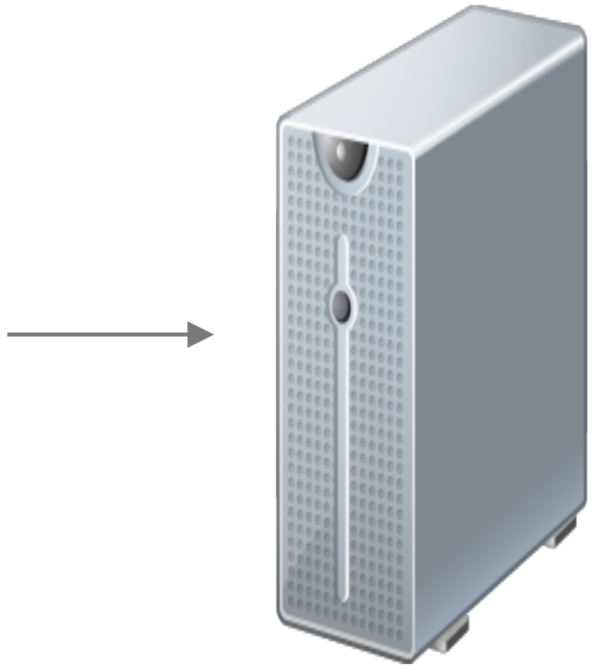
- Repetitive requests
- Predictable requests
- Data that needs to be accessed more quickly than is possible with a database query

Vertical Scaling



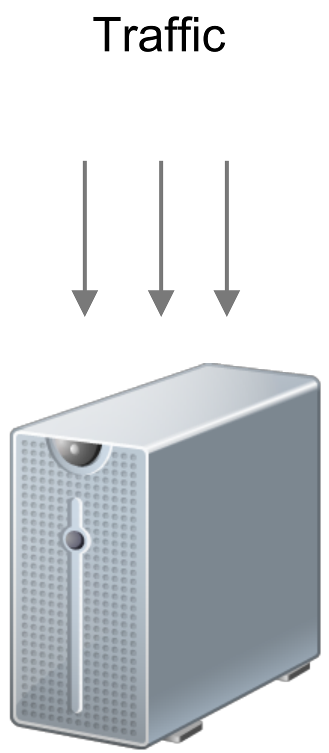
Vertical Scaling

Traffic



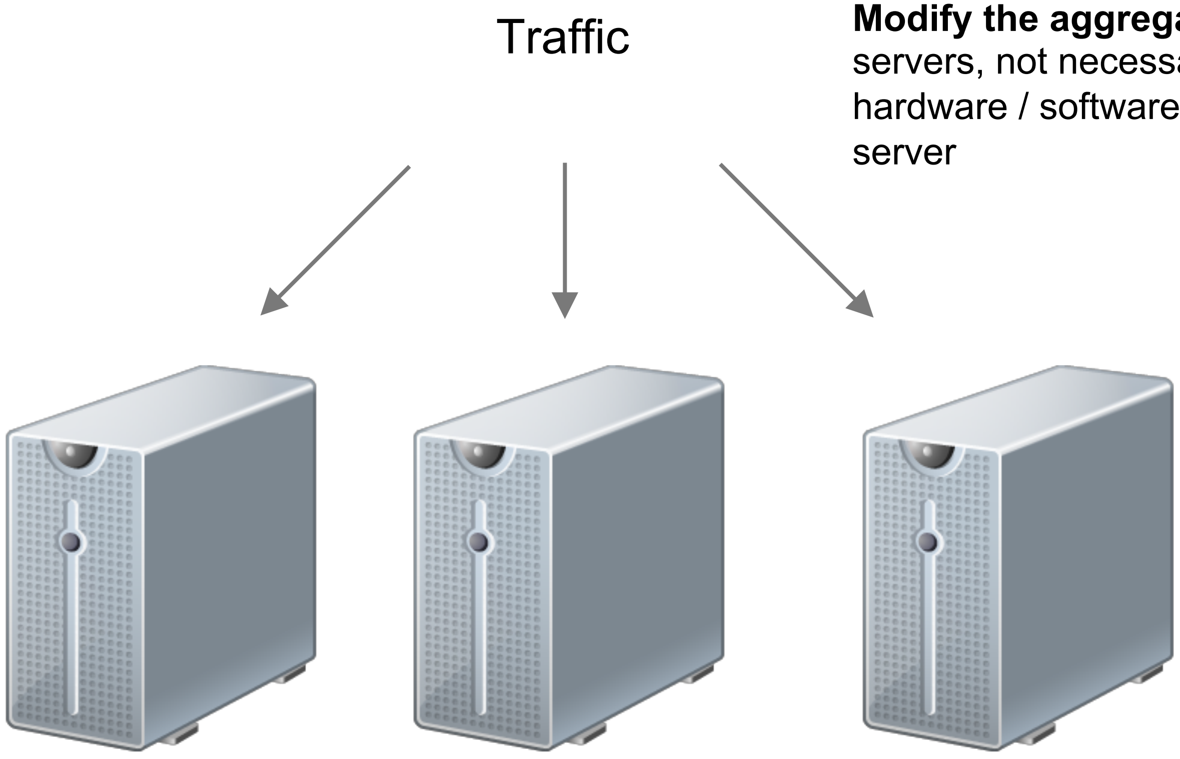
Improve each individual server:
- More / less RAM
- More / less / faster Storage
- Faster / Slower / more core CPU

Horizontal Scaling

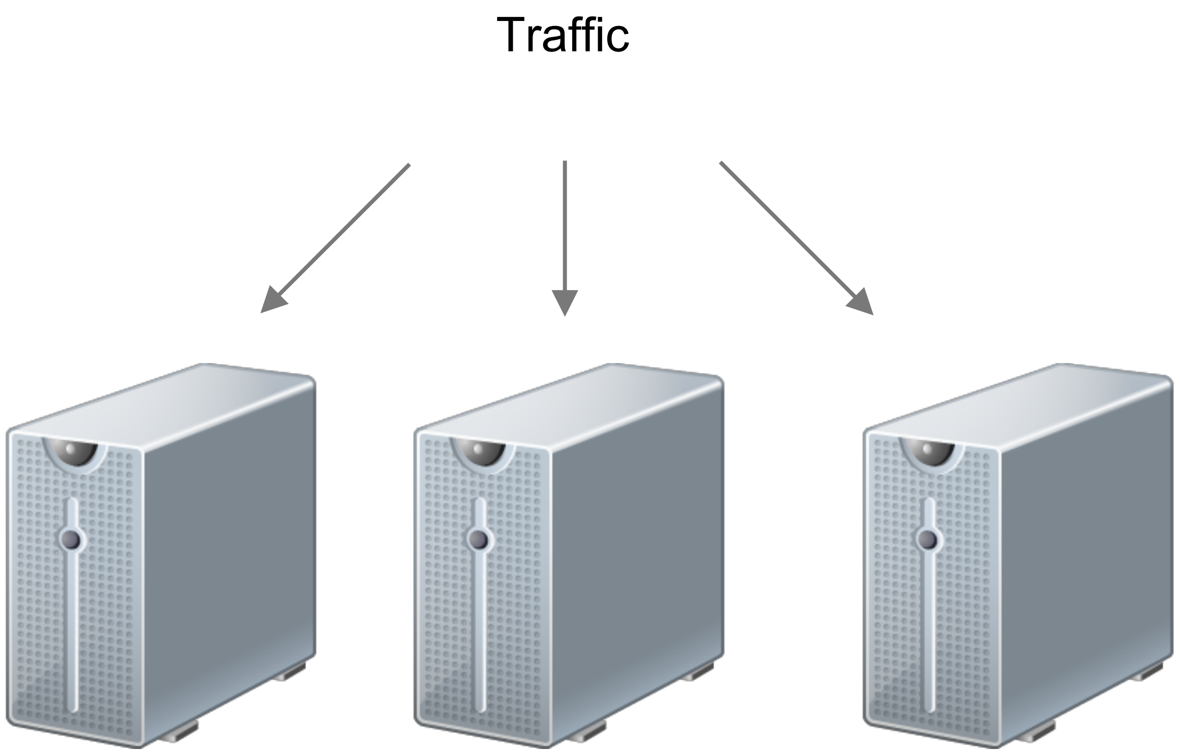


Horizontal Scaling

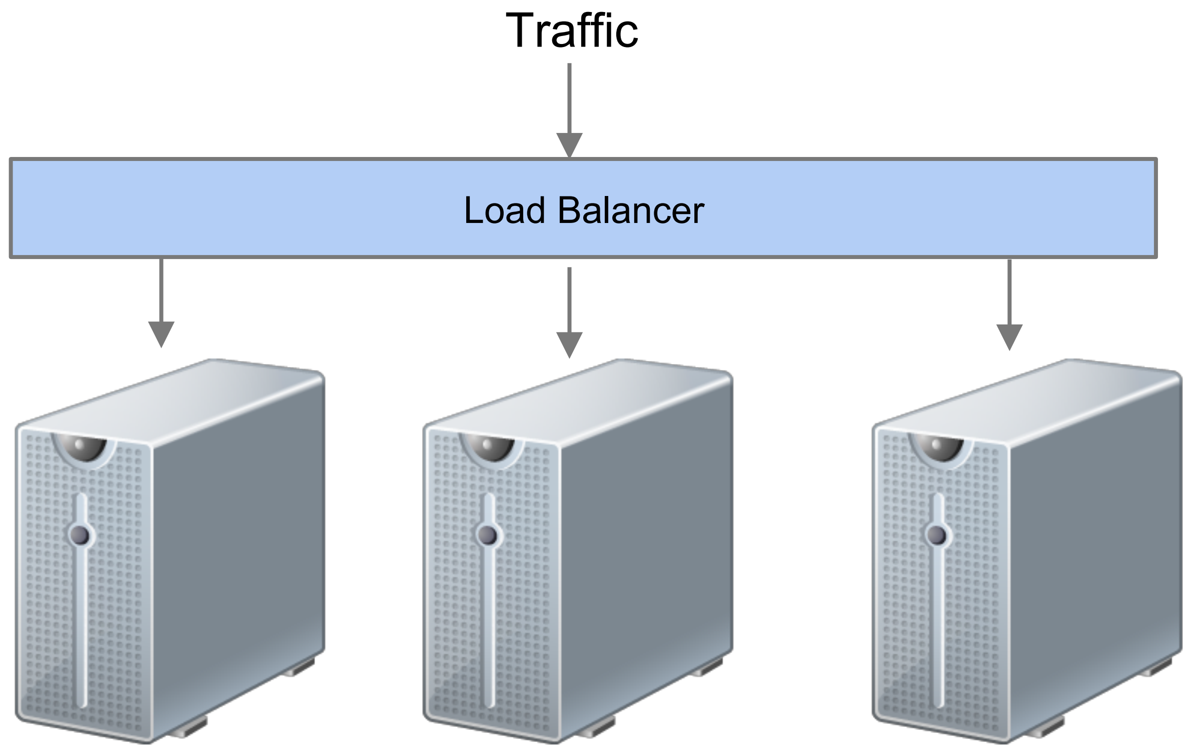
Modify the aggregate by adding more servers, not necessarily changing the hardware / software of each individual server



Load Balancer



Load Balancer



Service

Imagine a site that lets pet owners text their pets with updates while on vacation.

Considerations:

1. Registering a pet to the site is very performance-expensive
2. Processing the texts is quick, but has a lot of throughput
3. We experience seasonal spikes (Winter Holidays, Thanksgiving, etc.)

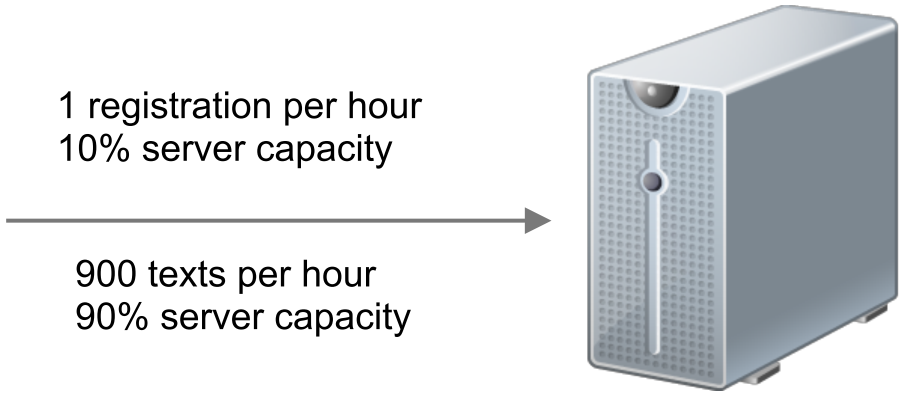
Image: http://www.mid-day.com/

Service

Imagine a site that lets pet owners text their pets with updates while on vacation.

Considerations:

1. Registering a pet to the site is very performance-expensive
2. Processing the texts is quick, but has a lot of throughput
3. We experience seasonal spikes (Winter Holidays, Thanksgiving, etc.)



Text My Pet Server

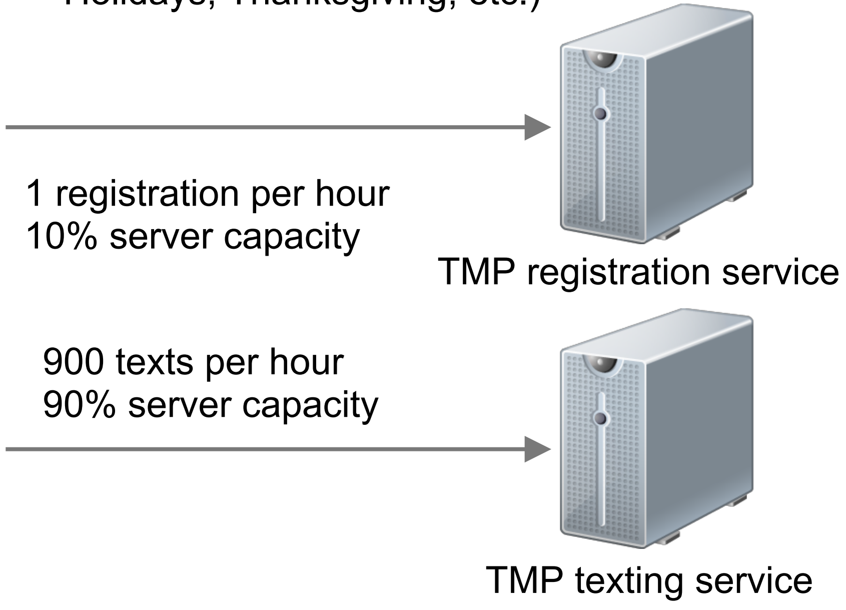
If we get one additional registration per hour, we need a second server. That costs $$.

Image: http://www.mid-day.com/

Service

Imagine a site that lets pet owners text their pets with updates while on vacation.

Considerations:
1. Registering a pet to the site is very performance-expensive
2. Processing the texts is quick, but has a lot of throughput
3. We experience seasonal spikes (Winter Holidays, Thanksgiving, etc.)



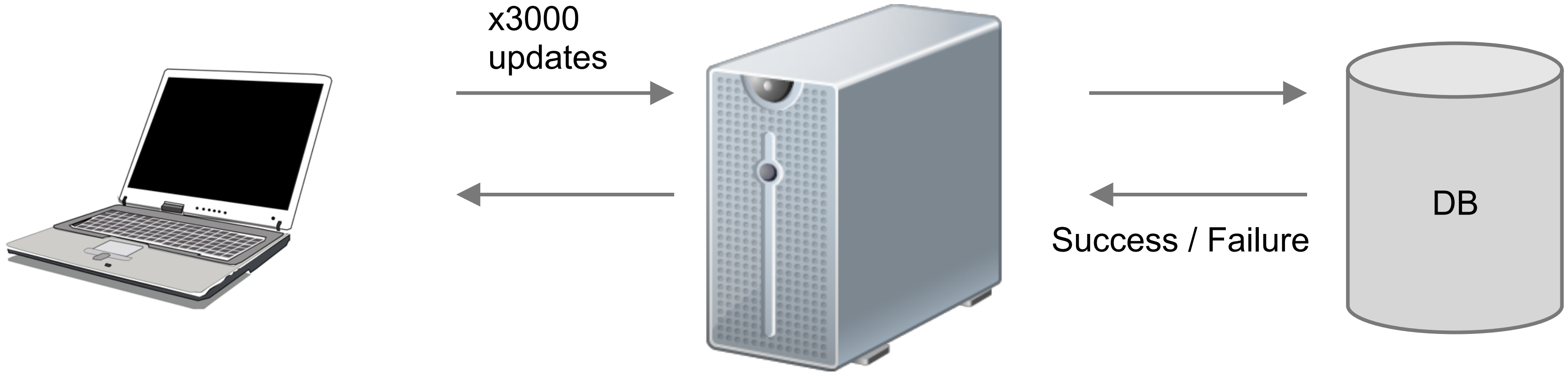
Now we can scale our registration and texting services separately, and can get more appropriate ‘sized’ servers for each.

Image: http://www.mid-day.com/

Cases for Service Oriented Architecture

- Performance Profiles
  - process A is really compute-intensive!
  - process B is really memory-intensive!
  - process C is bound by long IO times to a db!
- Maintainability
  - If I make a change to process A, does it somehow impact (or break!) process B?
  - How easy is it to phase in / phase out functionality?
- Testability
  - Can I guarantee that I can quickly find the root cause?
  - If I add a feature, can I know the scope of the performance impact?

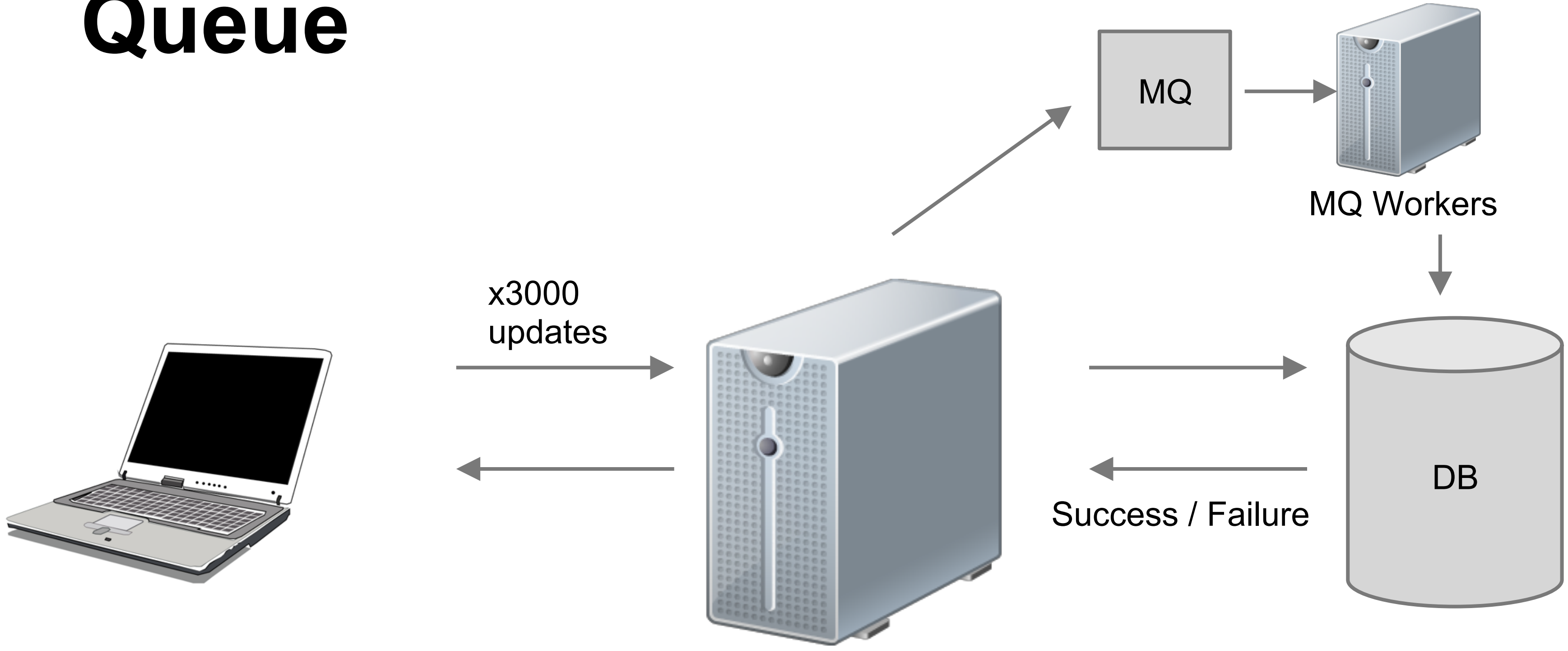
Queue



For the special case where:

- the request doesn’t need to know if the transaction was successful or not
- it is acceptable to complete the transaction ‘out of band’ (a.k.a. out of the request cycle)
- long running transactions (resizing images, video)
- Multiple services should know about the request at the same time

Queue



For the special case where:

- the request doesn’t need to know if the transaction was successful or not
- it is acceptable to complete the transaction ‘out of band’ (a.k.a. out of the request cycle)
- long running transactions (resizing images, video)
- Multiple services should know about the request at the same time

Exercise!

Congratulations! You are a software engineer at a startup. You have been asked to come up with some ideas on how to make a system more scalable. What you know:

- The database and application are on the same server
- There are three types of requests:
  - Registration - require DB access, and knowledge of successful transaction
  - Lookups - generally very **repetitive**, may not require direct access to the database
  - Updates - should eventually update information from the DB; we do not need to know if they are successful right away.

What would you do?

Questions!

Dean Dieker
ddieker@gmail.com
@martialdean