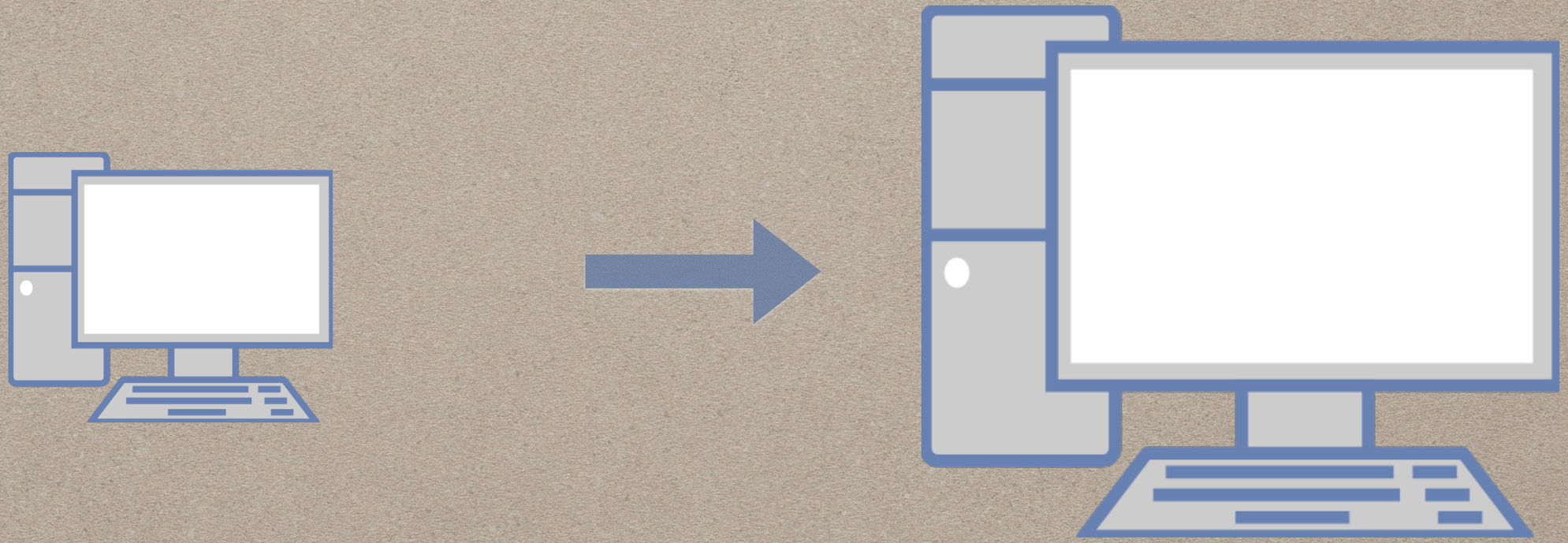
SCALING OUT:
FROM FIRST PRINCIPLES

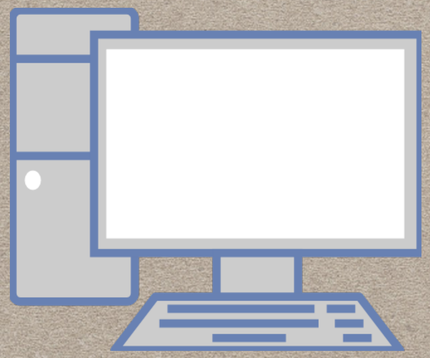
IVAN VERGILIEV

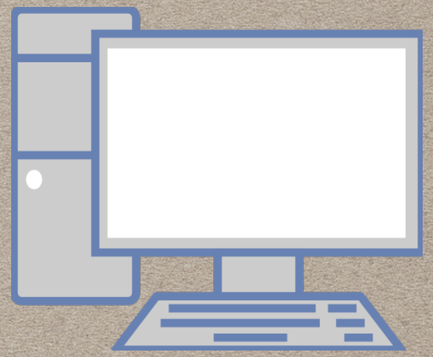
WHAT IS “SCALING”?

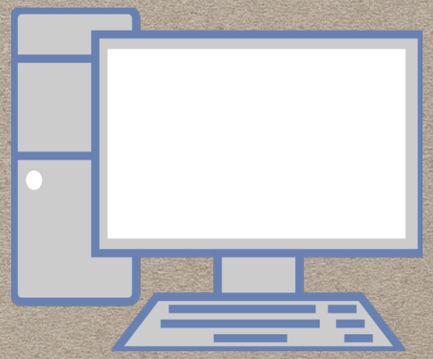
VERTICAL SCALING



HORIZONTAL SCALING







“Sure, sure, I read Hacker News too”

– People, on scaling

ABOUT ME

OR, “WHY SHOULD I BE LISTENING TO THAT GUY?”

“If it happens with a 1 in a 1,000,000 chance, it’s going to happen a bunch of times every hour.”

– Google Engineer, on “rare” errors

- Lead the development of Leanplum's Analytics infrastructure
  - ~200 machines, ~2k CPUs
- Currently at Heap
  - 1 Petabyte of data stored on a 50-machine Postgres cluster

- So, these companies exist
- And some of them even exist in Bulgaria

DISCLAIMER: COMPUTERS ARE REALLY FAST

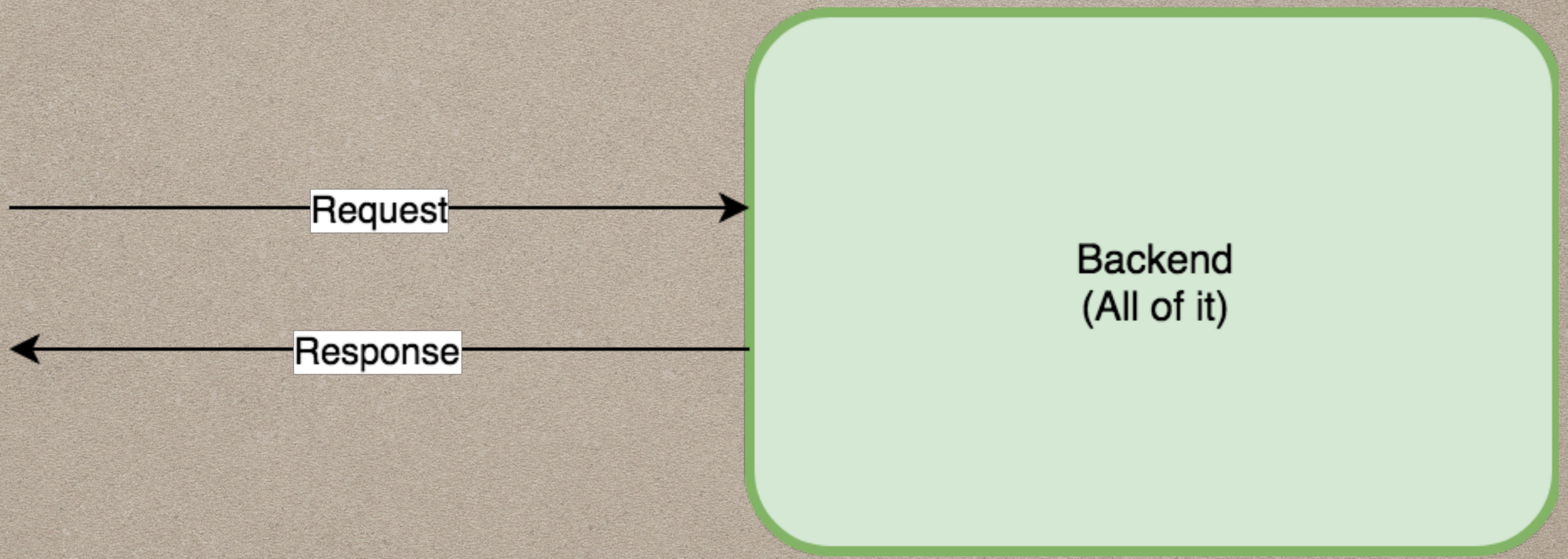
DISCLAIMER: COMPUTERS ARE REALLY FAST

- AWS offers the “X1e.32xlarge” machine type
- An X1e has 128 CPUs and 3,904 GiB of RAM
- That’s 4 TB of RAM, not 4 TB of hard disk space
- If each request needs 10ms and 300 MB of RAM, you can serve **12,000 concurrent users**

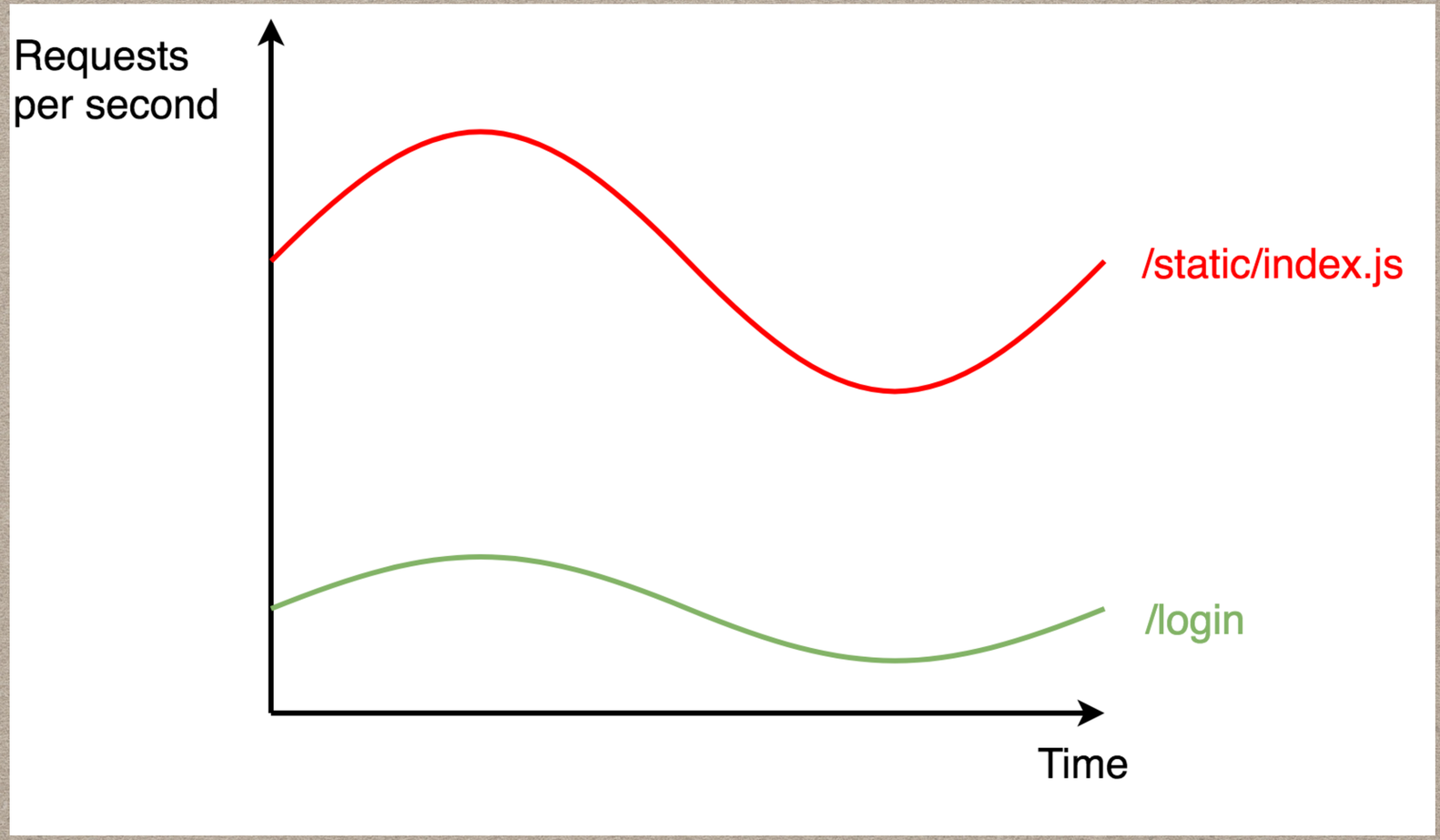
IF YOU CAN, STICK TO A SINGLE MACHINE

OTHERWISE, KEEP LISTENING

LET’S START SCALING!



MONOLITH



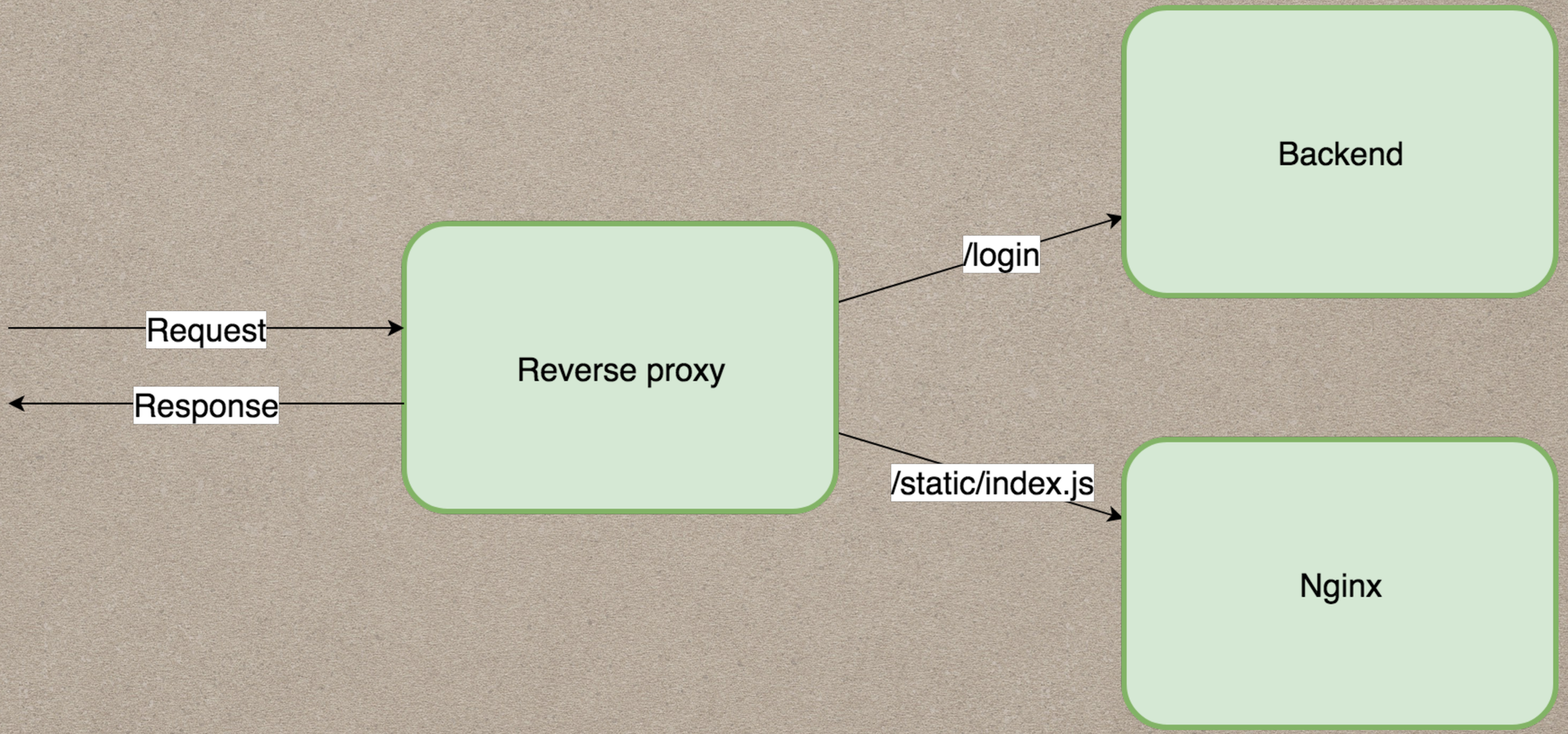
MARKETING CAMPAIGN

CAUSING YOUR MARKETING PAGE TO BLOW UP

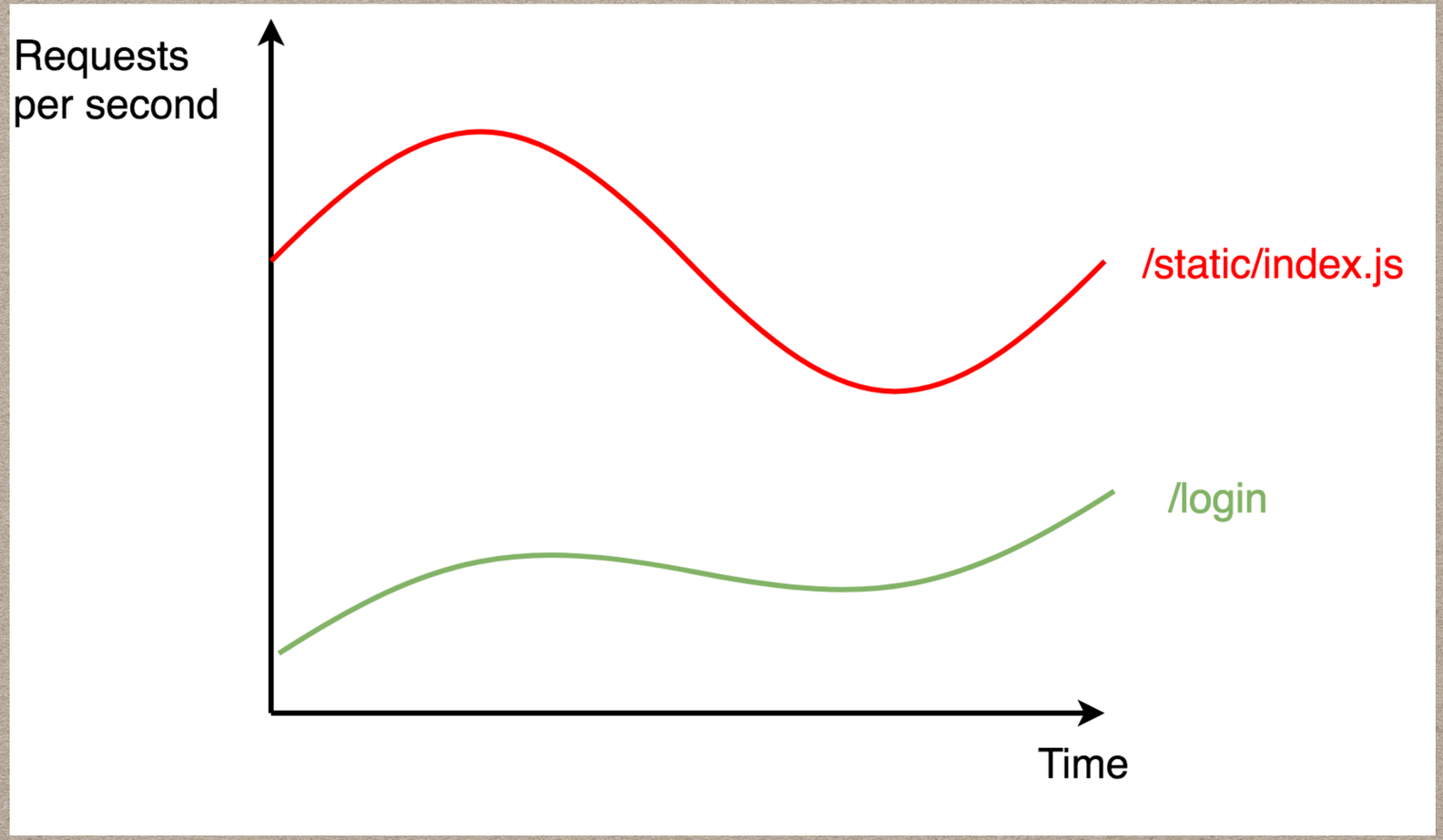
YOUR MARKETING PAGE DOESN'T RUN APPLICATION LOGIC!

SERVING STATIC FILES: PROBLEM SETTING

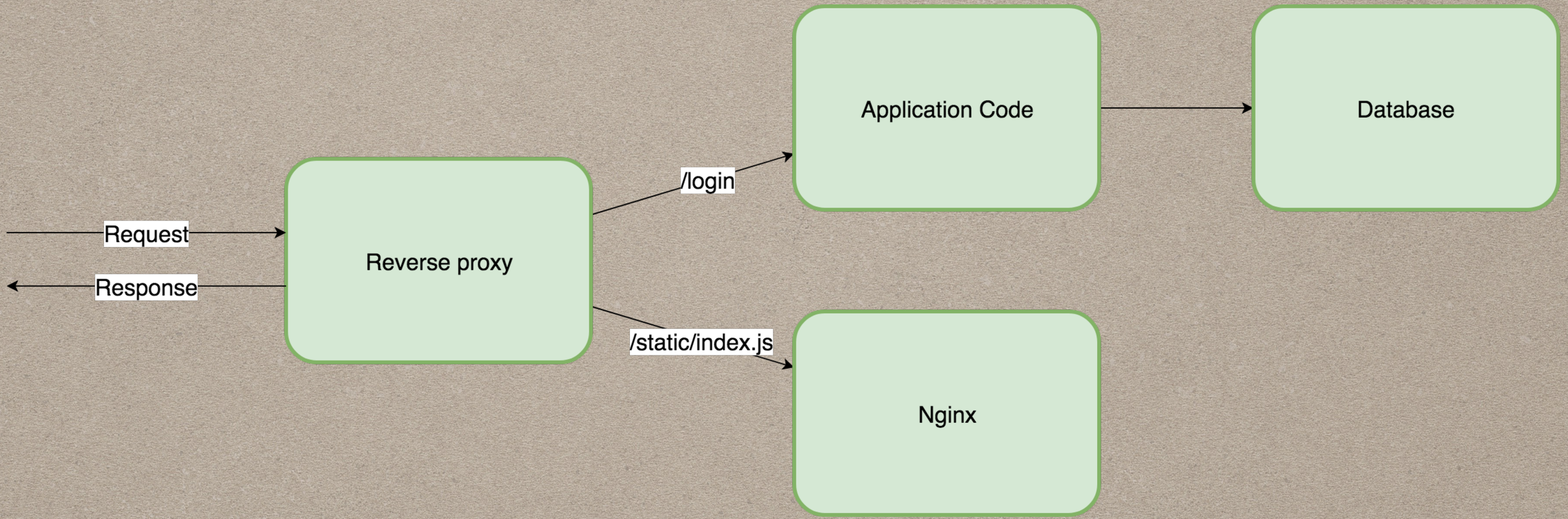
1. Read a file from disk
2. Return it to client
3. Do it all ***REALLY FAST!!!***



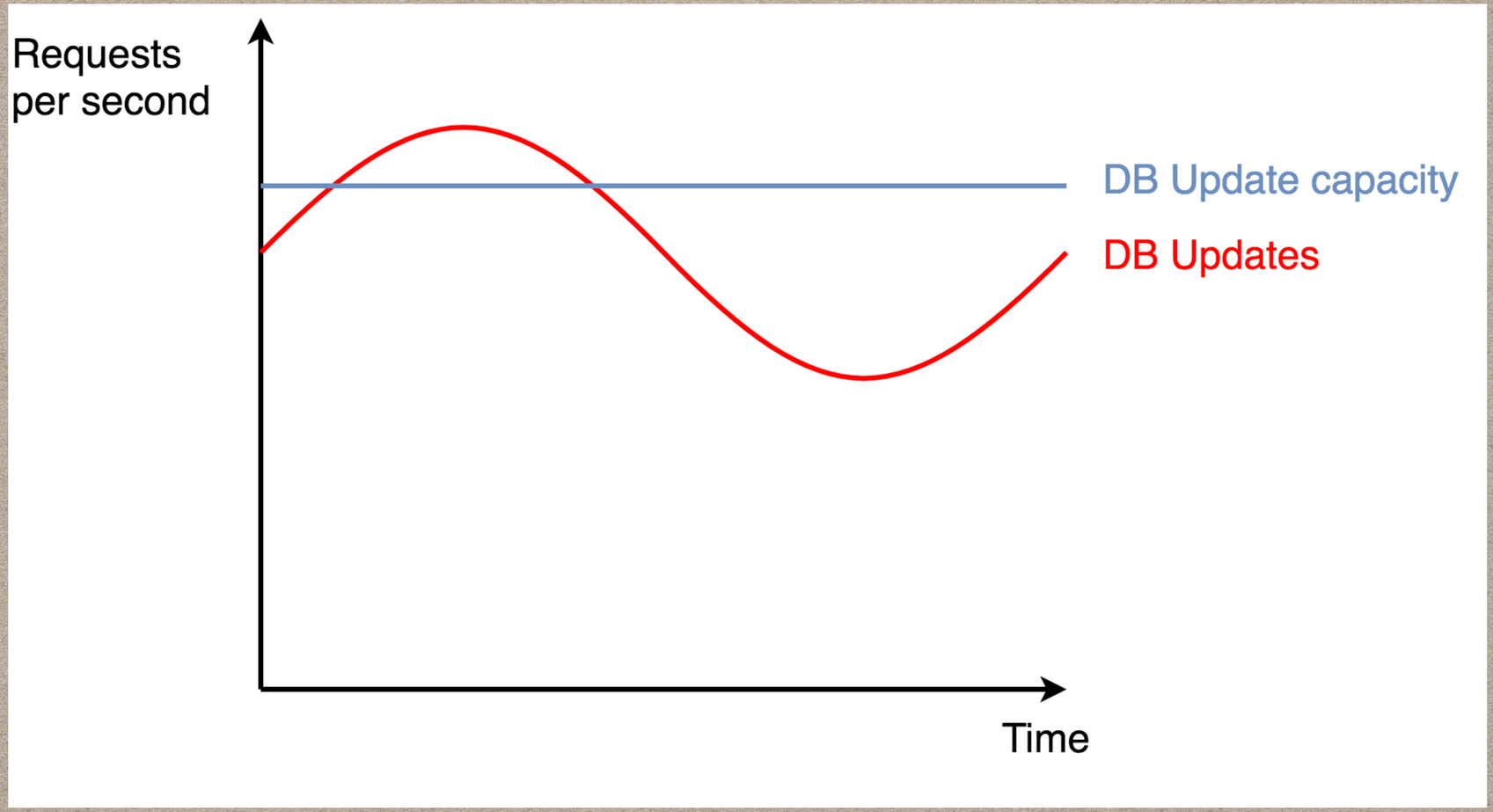
NGINX FOR STATIC ASSETS



MARKETING EFFORTS START TO PAY OFF



MOVE THE DATABASE TO A SEPARATE MACHINE

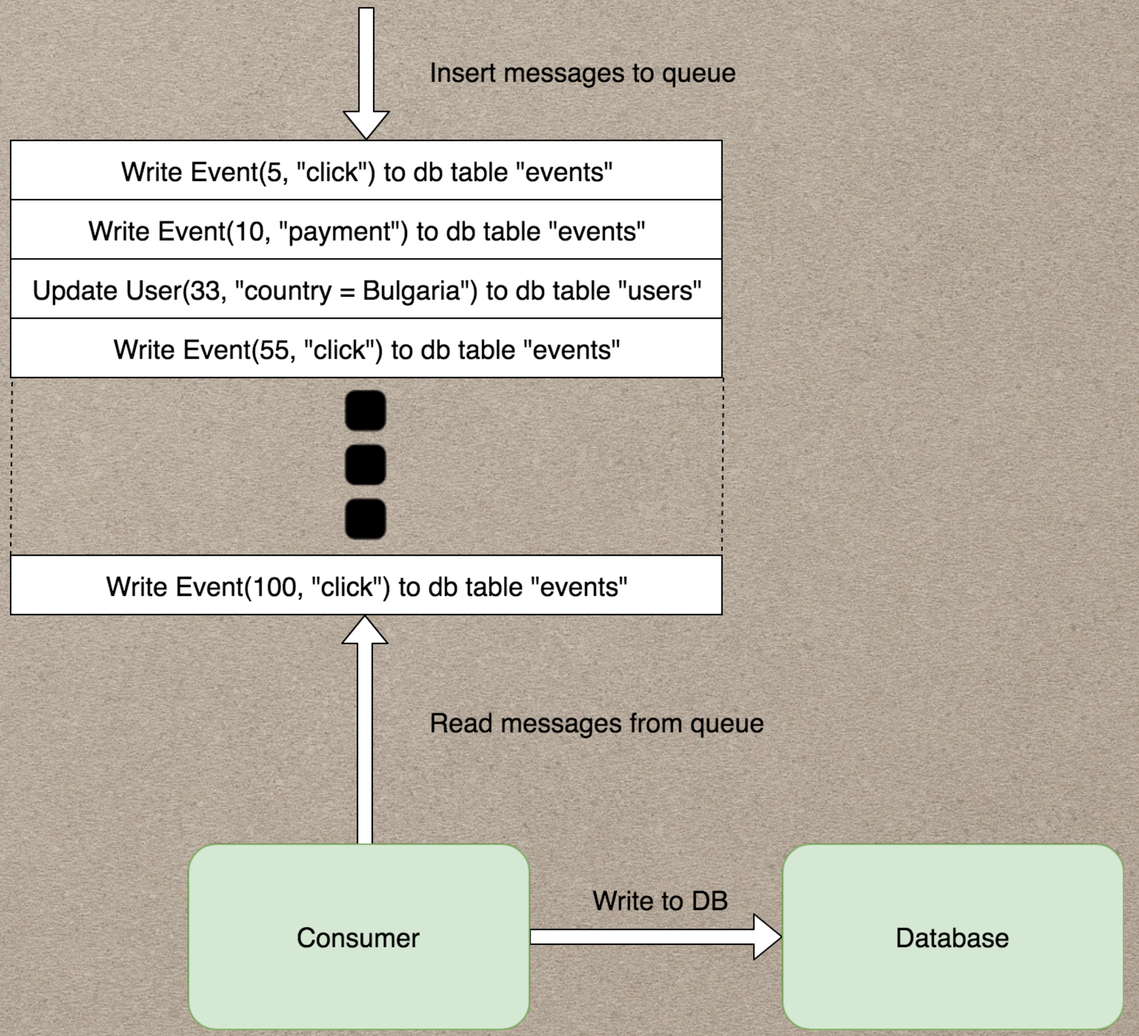


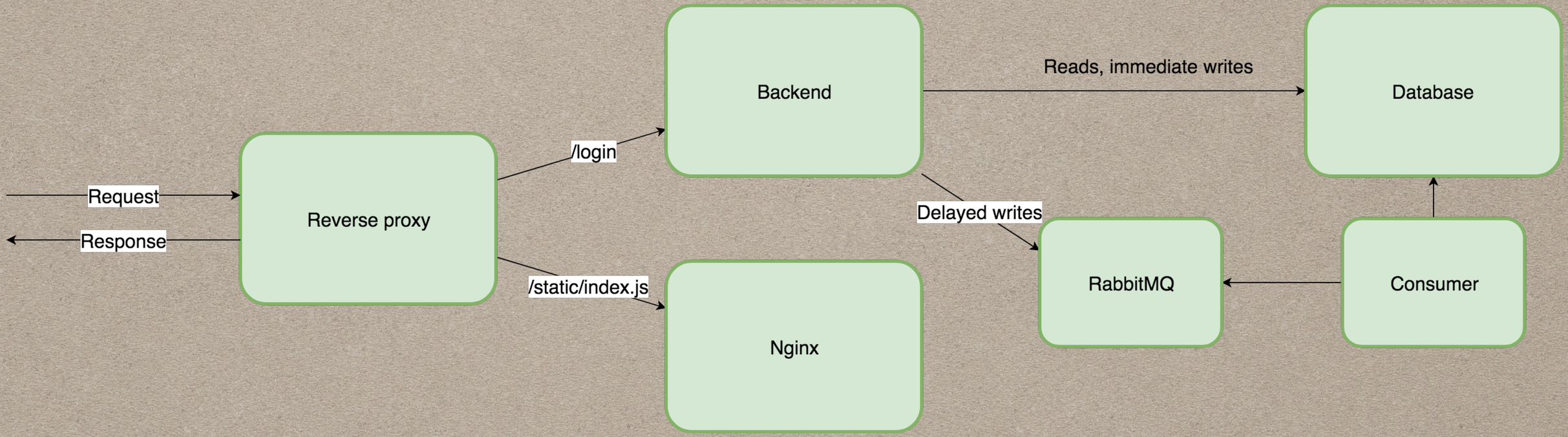
DATABASE CAN'T DEAL WITH UPDATE SPIKES

THE DB HAS ENOUGH TOTAL CAPACITY

- We could keep up with all data over a given day, just not at the exact rate we receive it
- Let’s decouple the time we receive the data from the time we write it

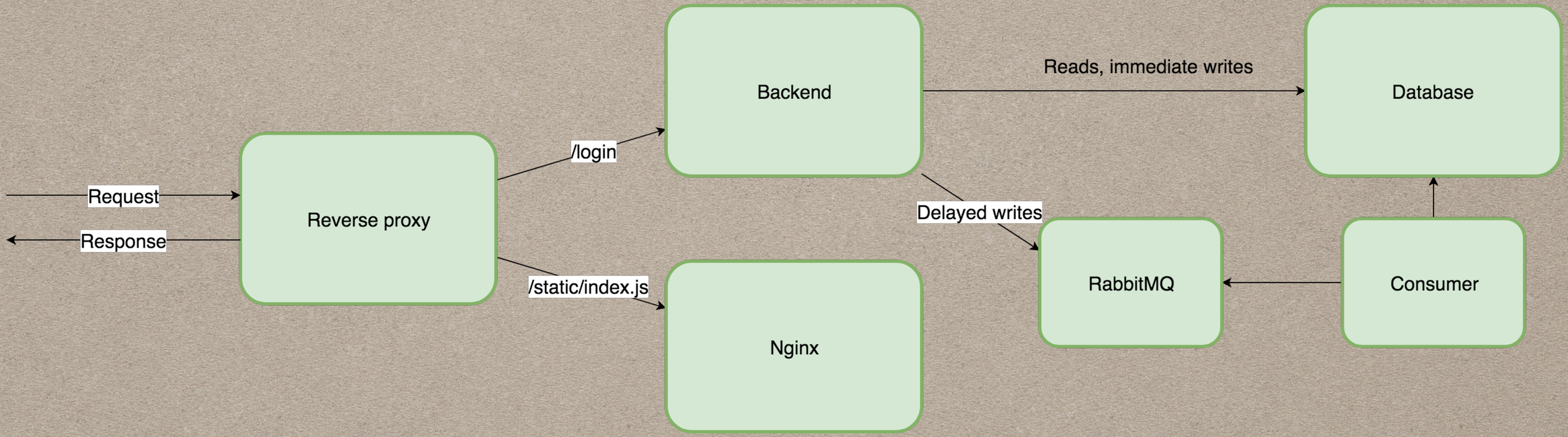
ADD A MESSAGE QUEUE IN FRONT OF THE DB





ADD A MESSAGE QUEUE IN FRONT OF THE DB

GOTCHA: CAREFUL WITH READ-YOUR-WRITES

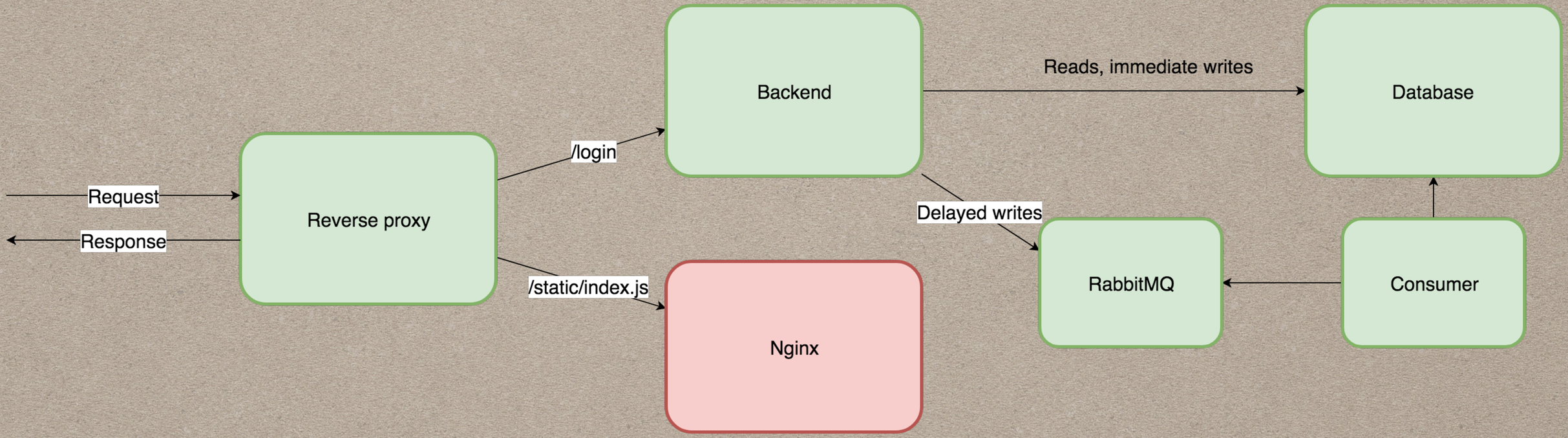


SEPARATE FUNCTIONALITIES LIVE ON SEPARATE MACHINES

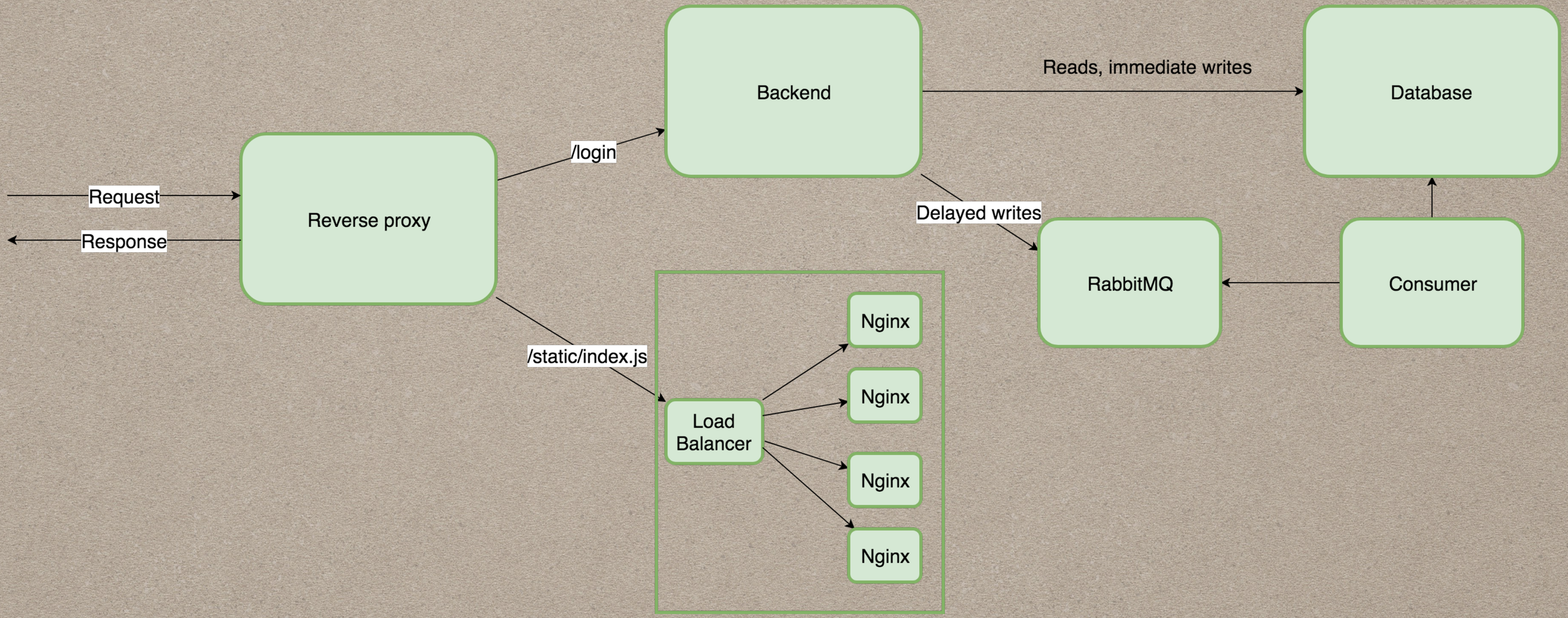
SO FAR, SO GOOD

HERE COMES THE FUN PART

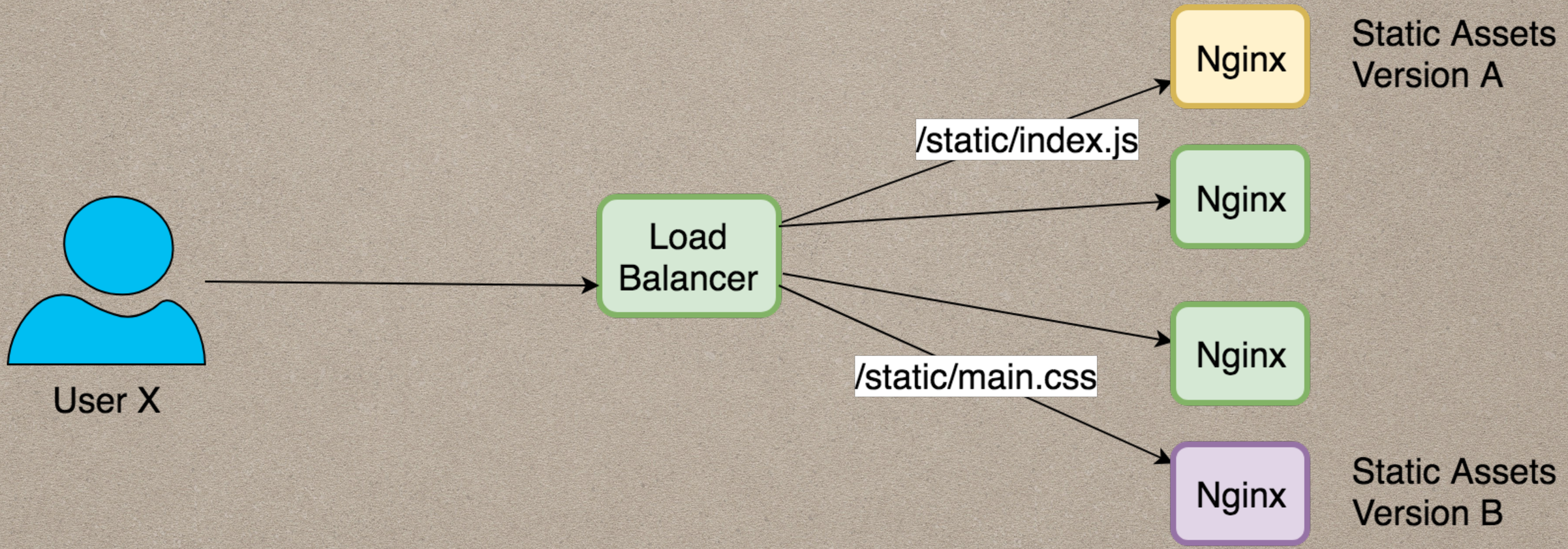
THE FUN PART IS WHEN INDIVIDUAL FUNCTIONALITIES DON’T FIT ON INDIVIDUAL MACHINES



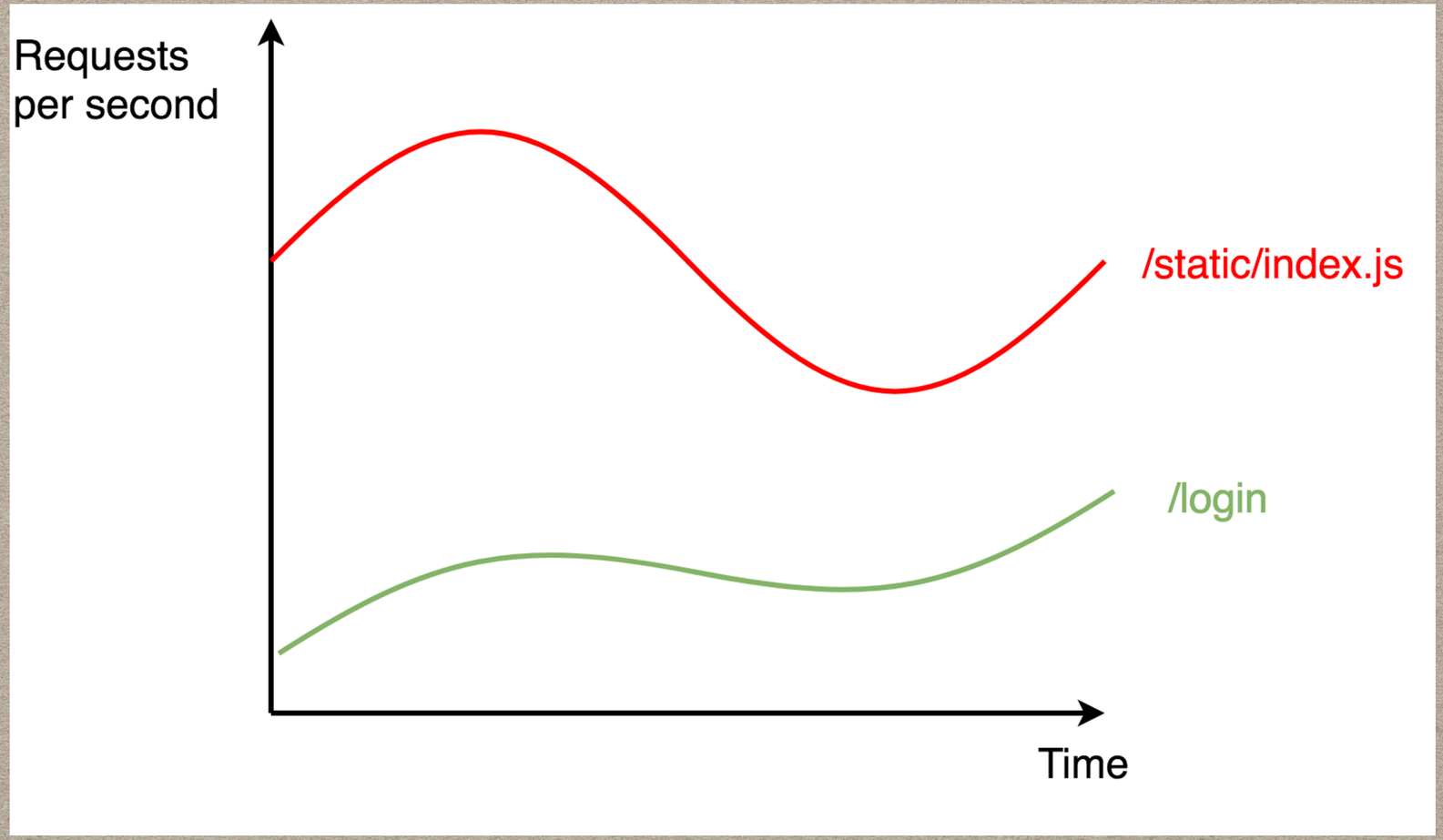
NEXT FUNDING ROUND, YOUR MARKETING SITE IS (ALMOST LITERALLY) ON FIRE AGAIN



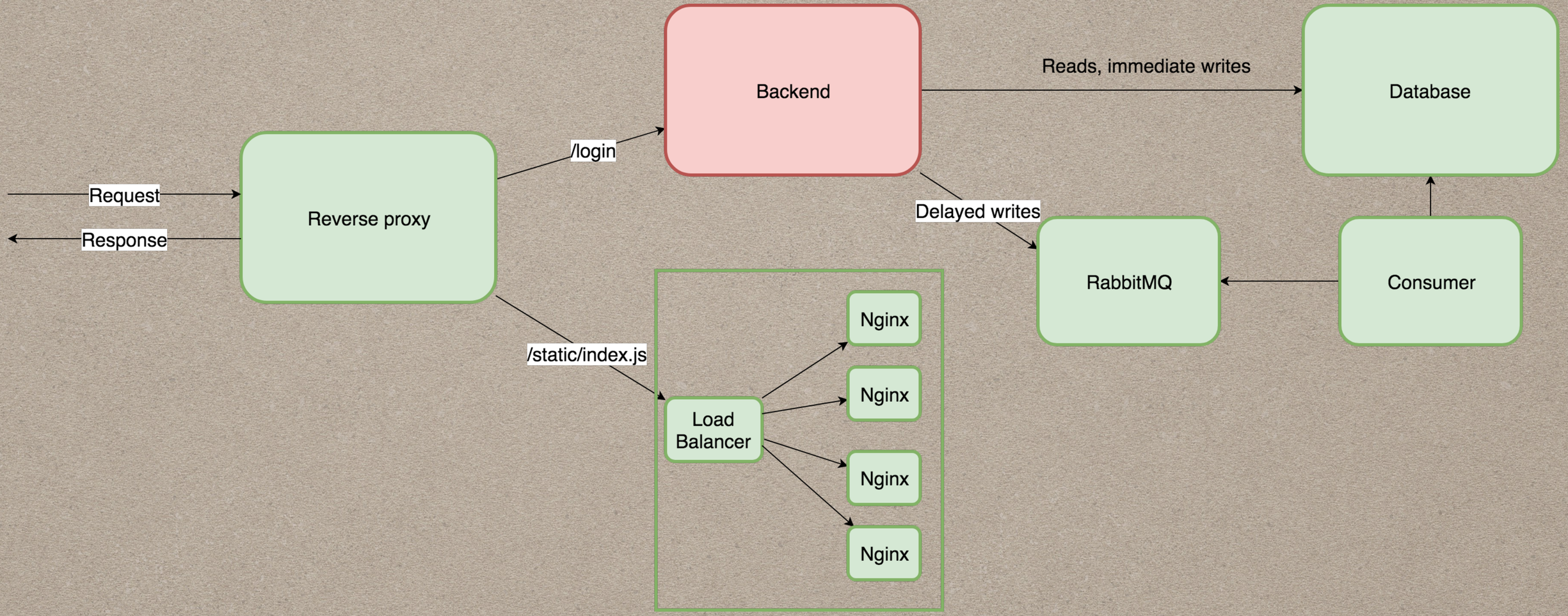
ALL THE SERVERS ARE SERVING THE SAME CONTENT



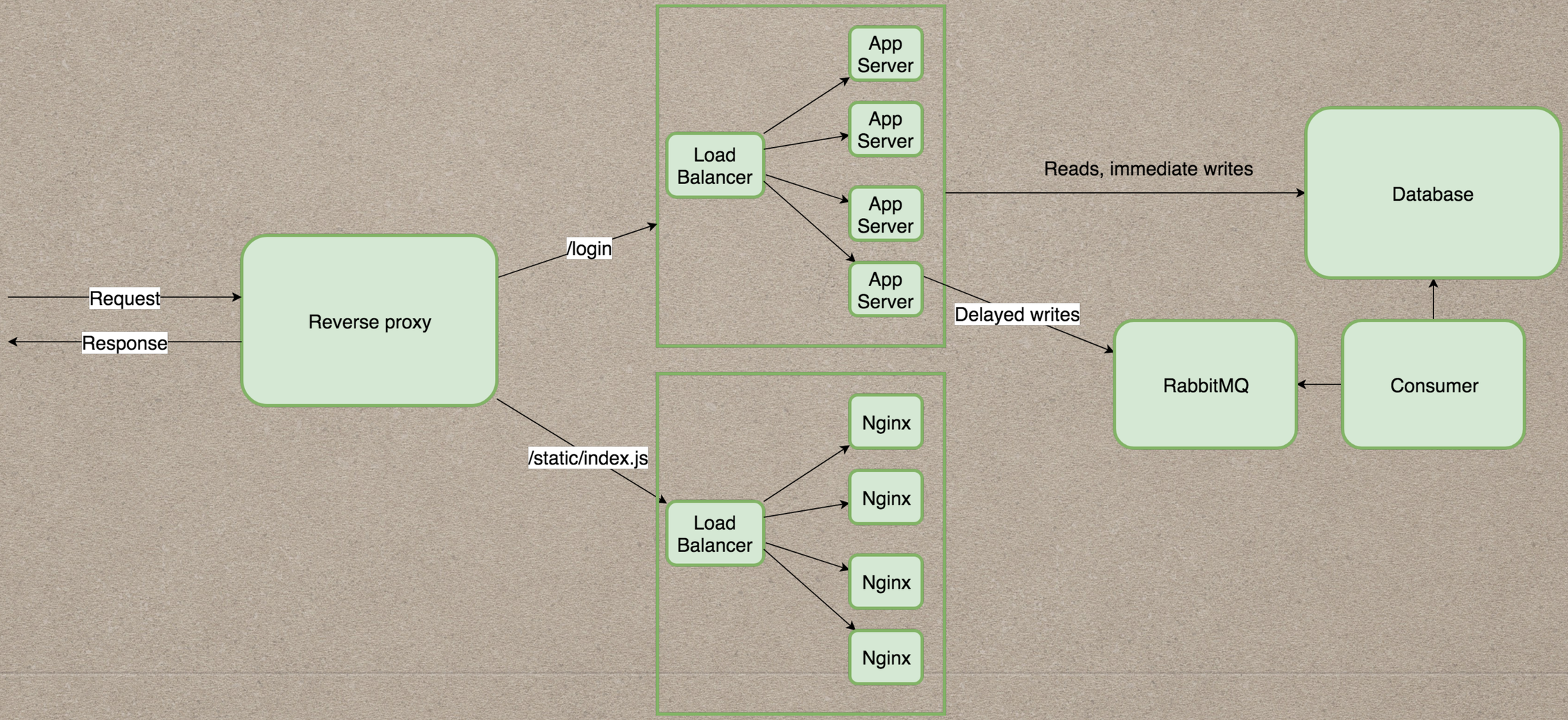
NEED TO BE CAREFUL WITH DEPLOYS



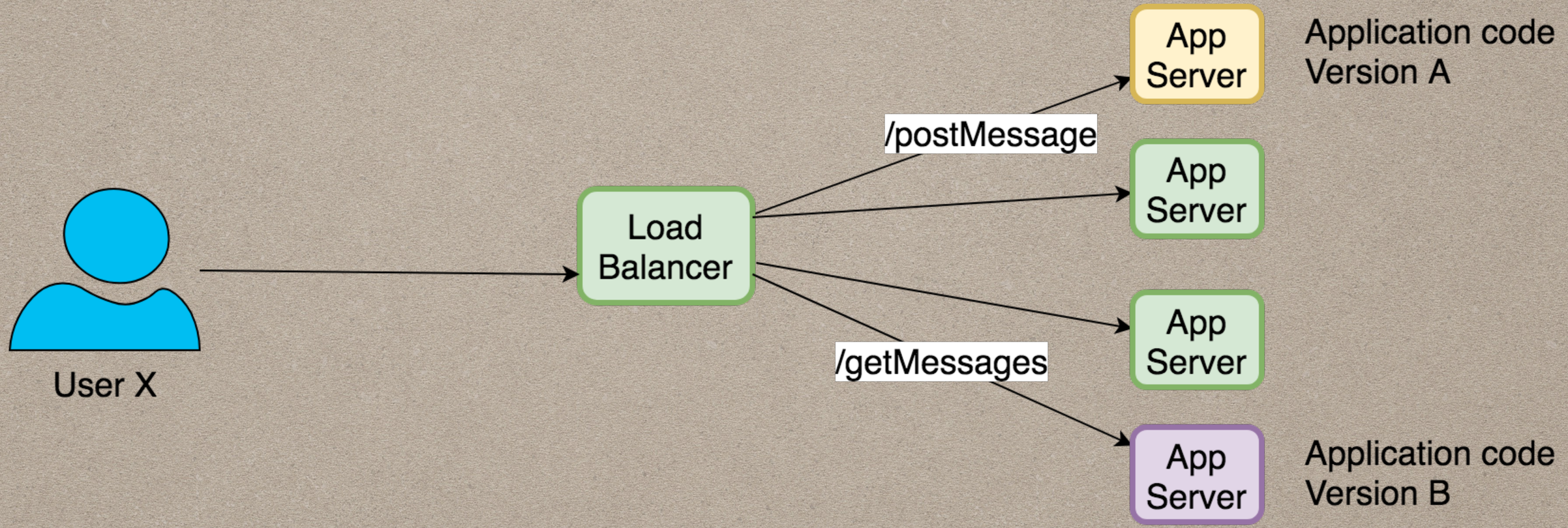
NEXT UP, WE FINALLY HIT A POINT WHERE OUR APPLICATION CODE IS CAUSING LOAD ISSUES!



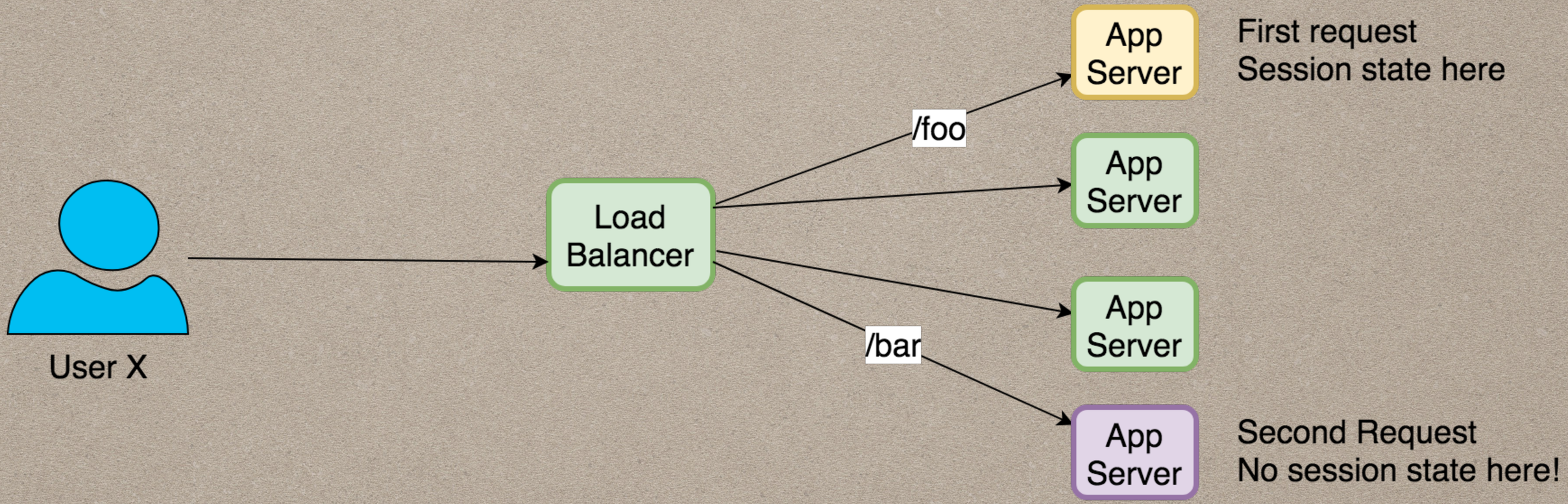
NEXT UP, WE FINALLY HIT A POINT WHERE OUR APPLICATION CODE IS CAUSING LOAD ISSUES!



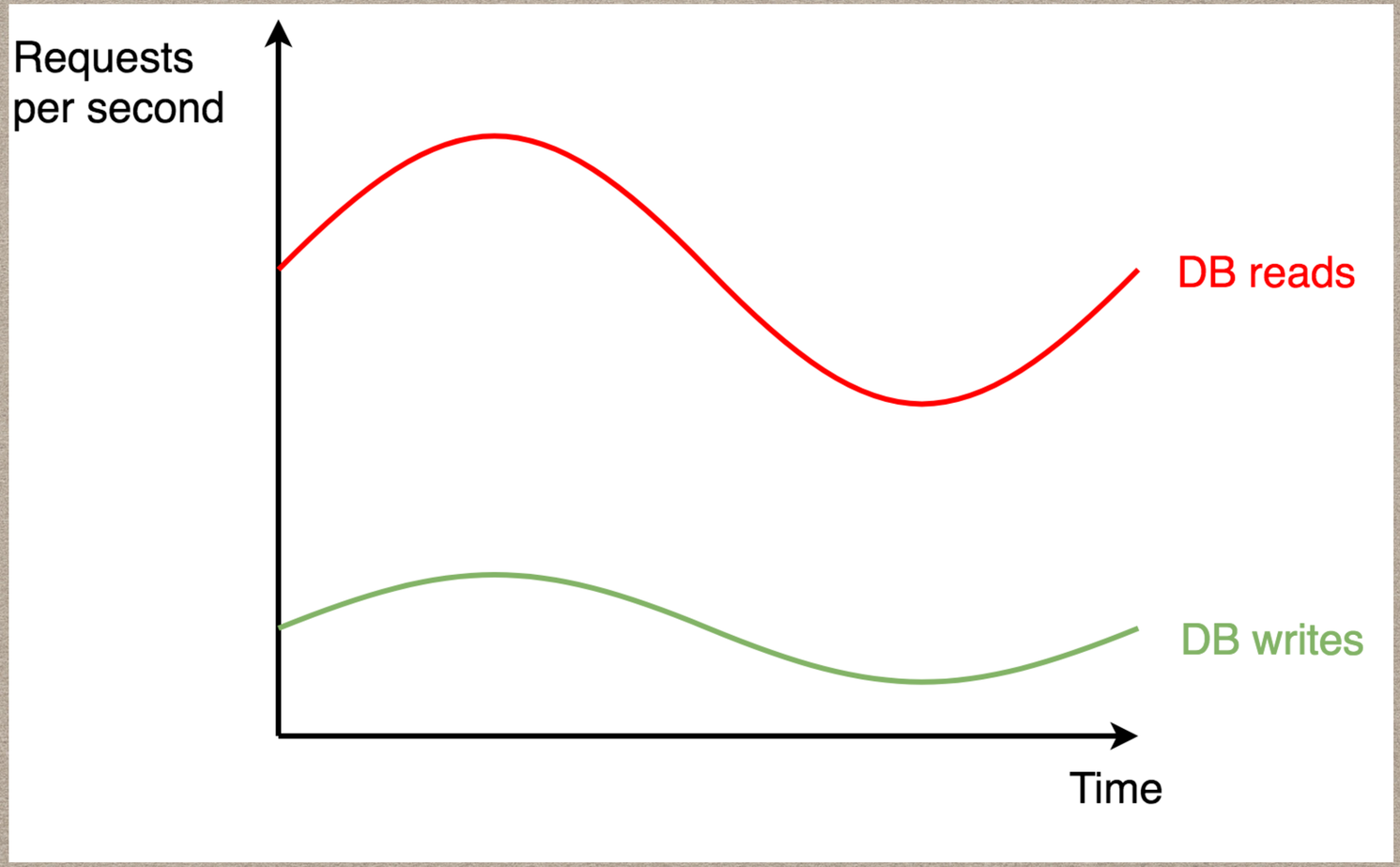
ALL THE APP SERVERS ARE STILL DOING THE SAME THING - PROCESSING REQUESTS



NEED TO BE CAREFUL WITH DEPLOYING AGAIN



NEED TO BE CAREFUL WITH IN-MEMORY STATE



NEXT UP, YOU REALIZE YOUR USERS ARE PASSIVE CONSUMERS

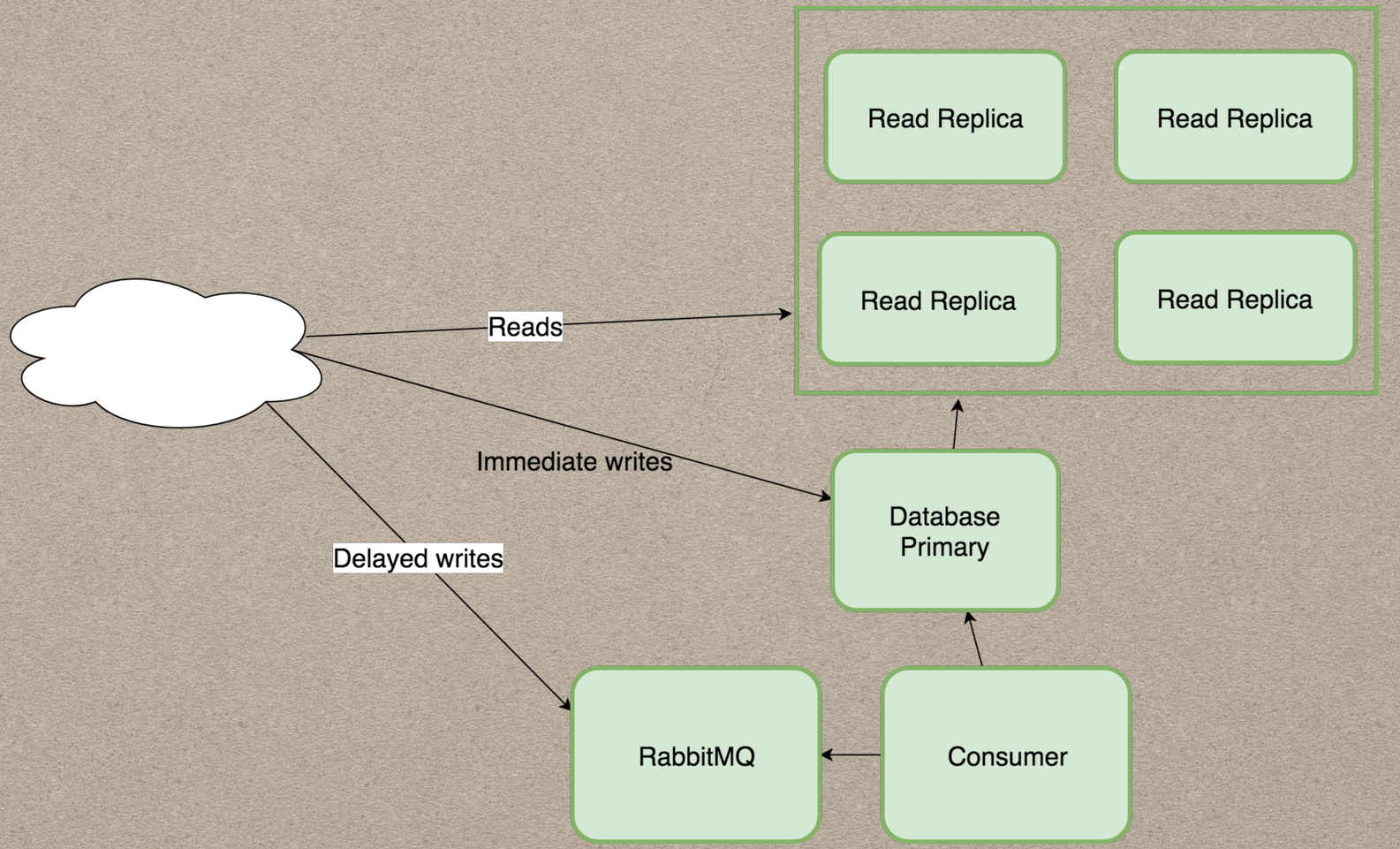
FACEBOOK'S "TAO" PAPER

- "TAO: Facebook's Distributed Data Store for the Social Graph"
- 0.2% of the total requests involved writes

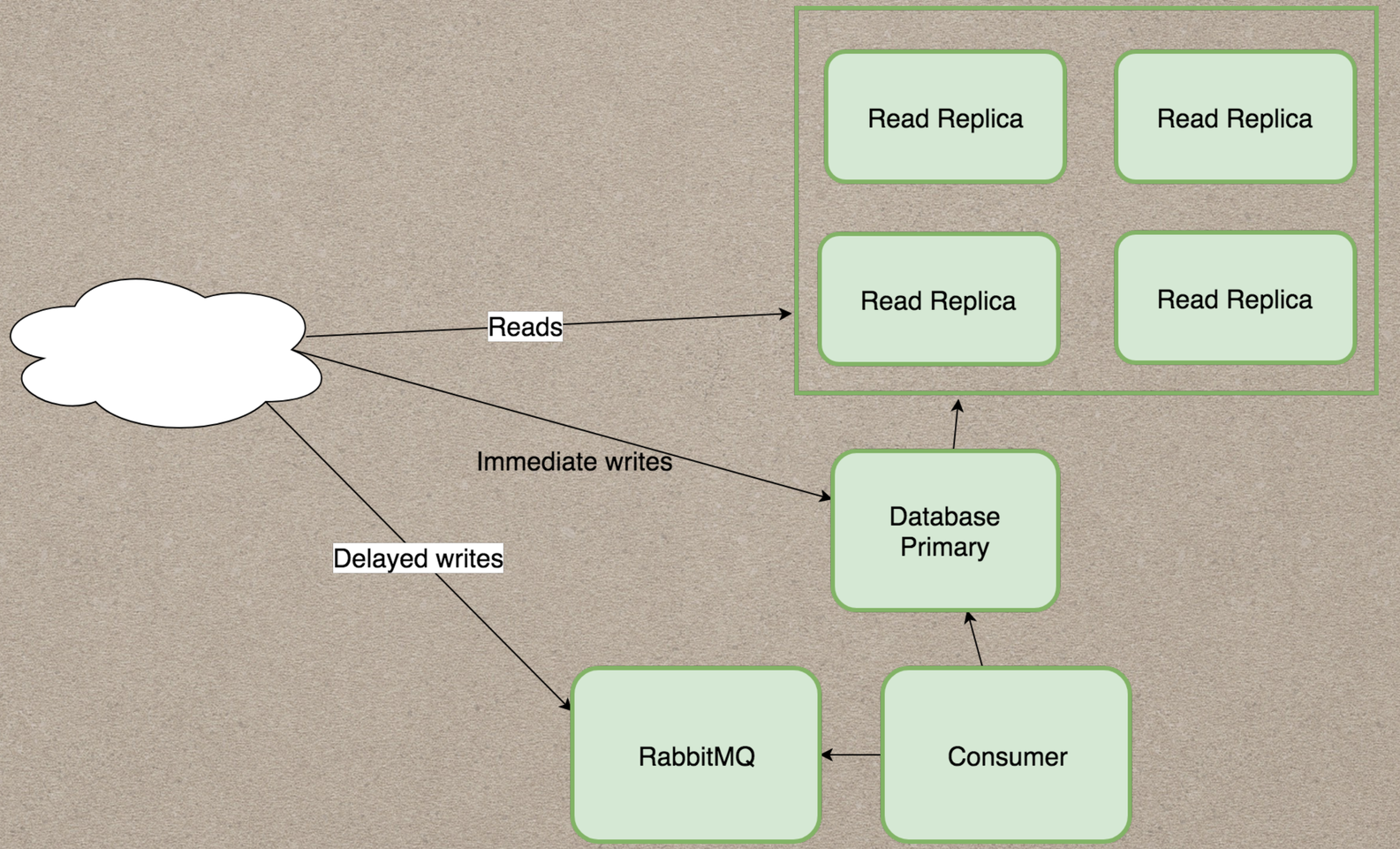
read requests	99.8 %	write requests	0.2 %
assoc_get	15.7 %	assoc_add	52.5 %
assoc_range	40.9 %	assoc_del	8.3 %
assoc_time_range	2.8 %	assoc_change_type	0.9 %
assoc_count	11.7 %	obj_add	16.5 %
obj_get	28.9 %	obj_update	20.7 %
		obj_delete	2.0 %

Figure 3: Relative frequencies for client requests to TAO from all Facebook products. Reads account for almost all of the calls to the API.

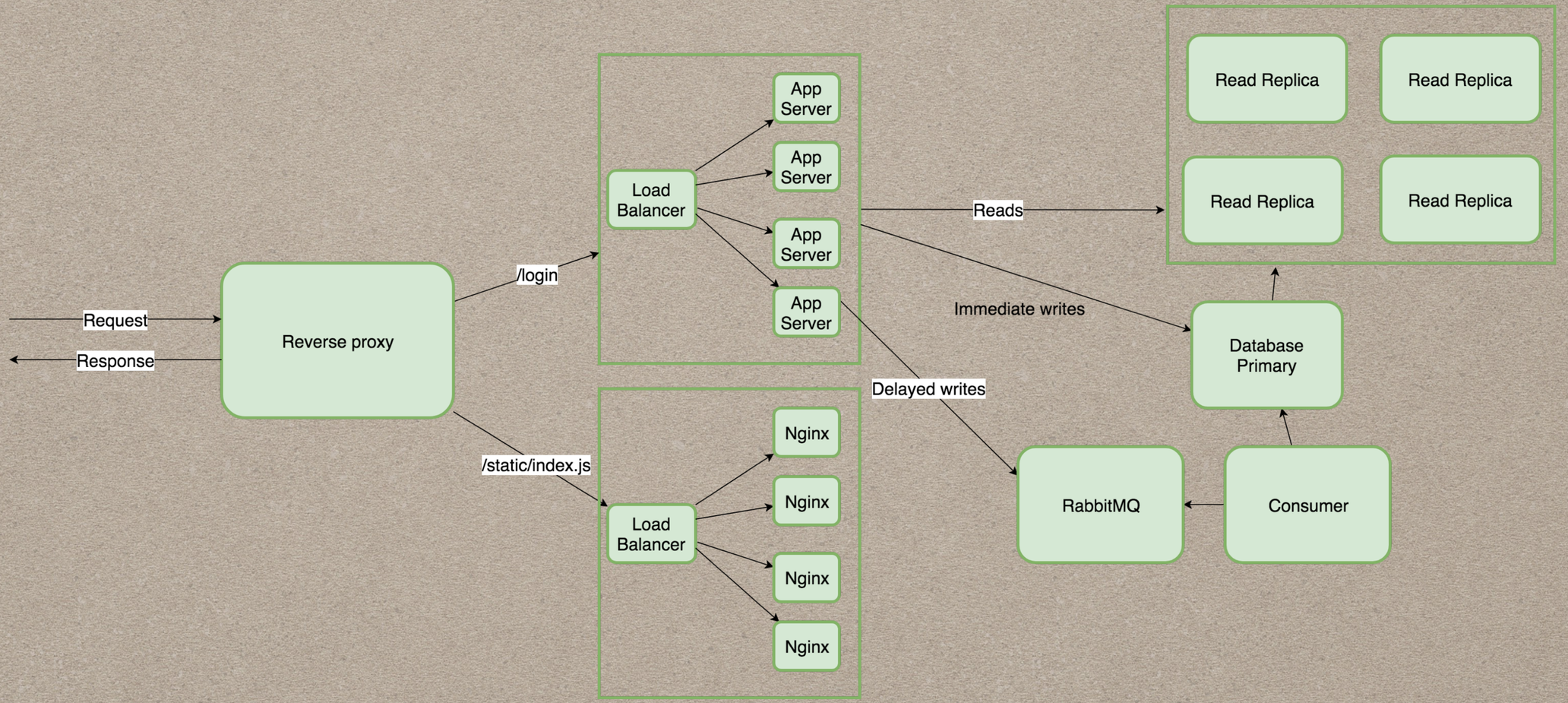
- Well, this is kind of disappointing for retention
- But it helps with the architecture!



ADD READ REPLICAS



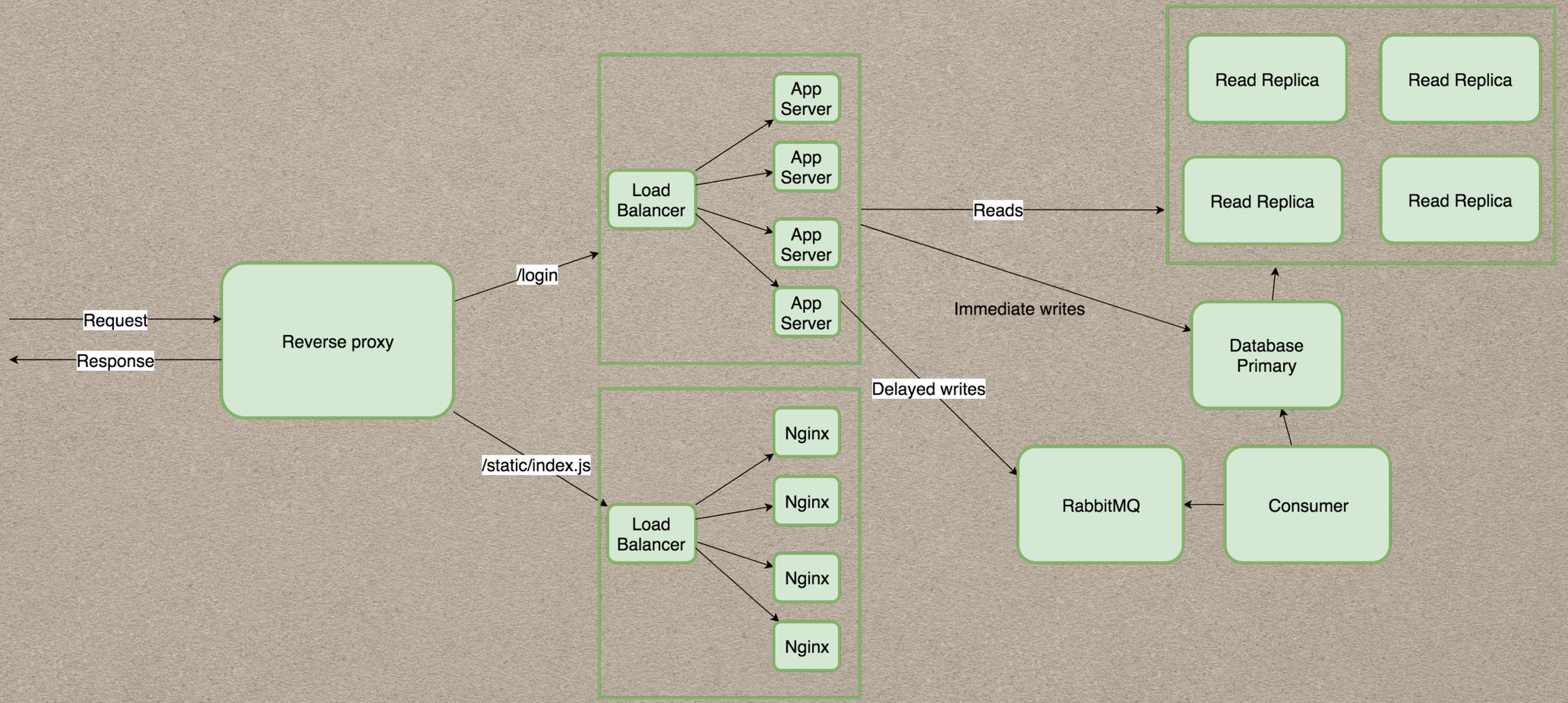
NEED TO THINK ABOUT STALE READS



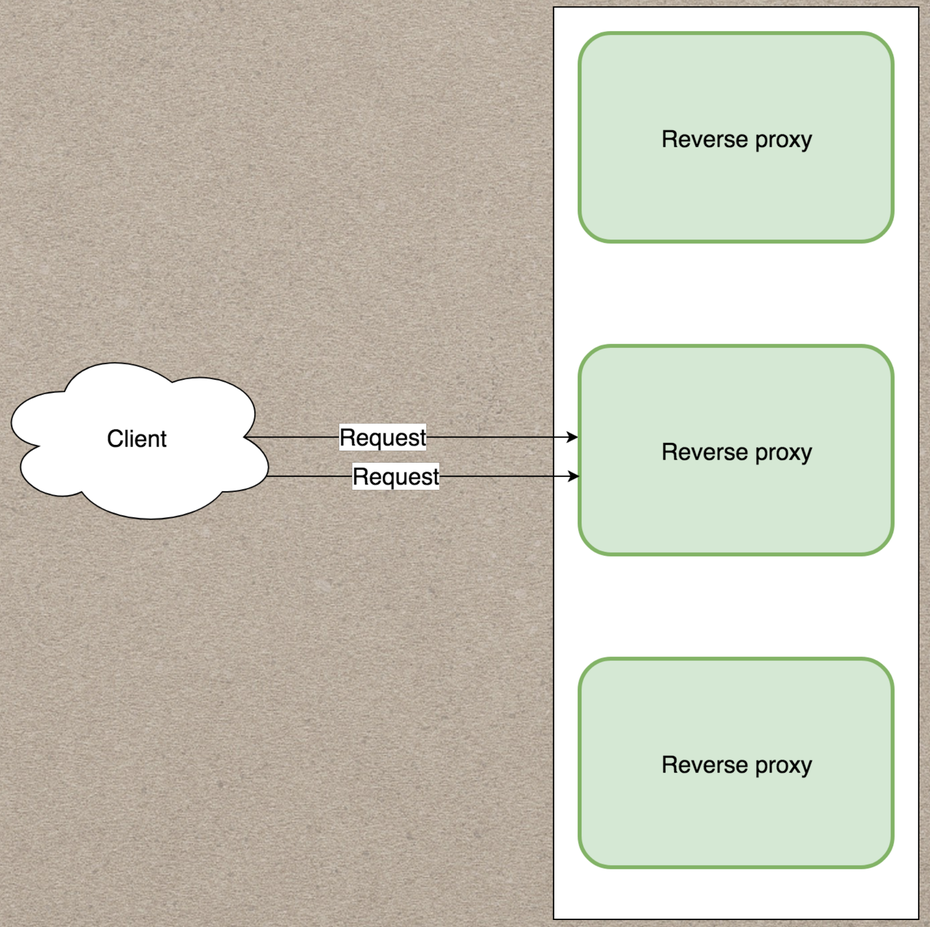
ARCHITECTURE SO FAR

LET’S LOOK AT SOMETHING MORE EXTREME!

WE GOT THE MAIN THINGS SCALED - WHAT COULD BE NEXT?



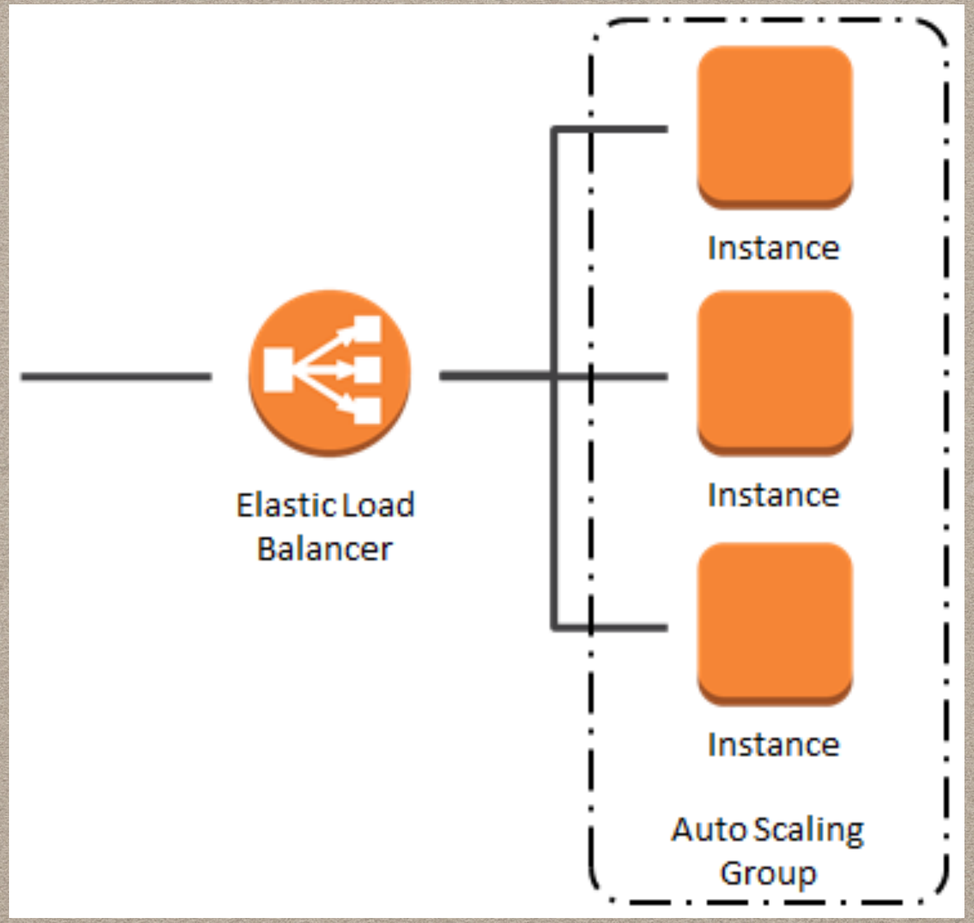
EVERY SINGLE REQUEST PASSES THROUGH THE REVERSE PROXY



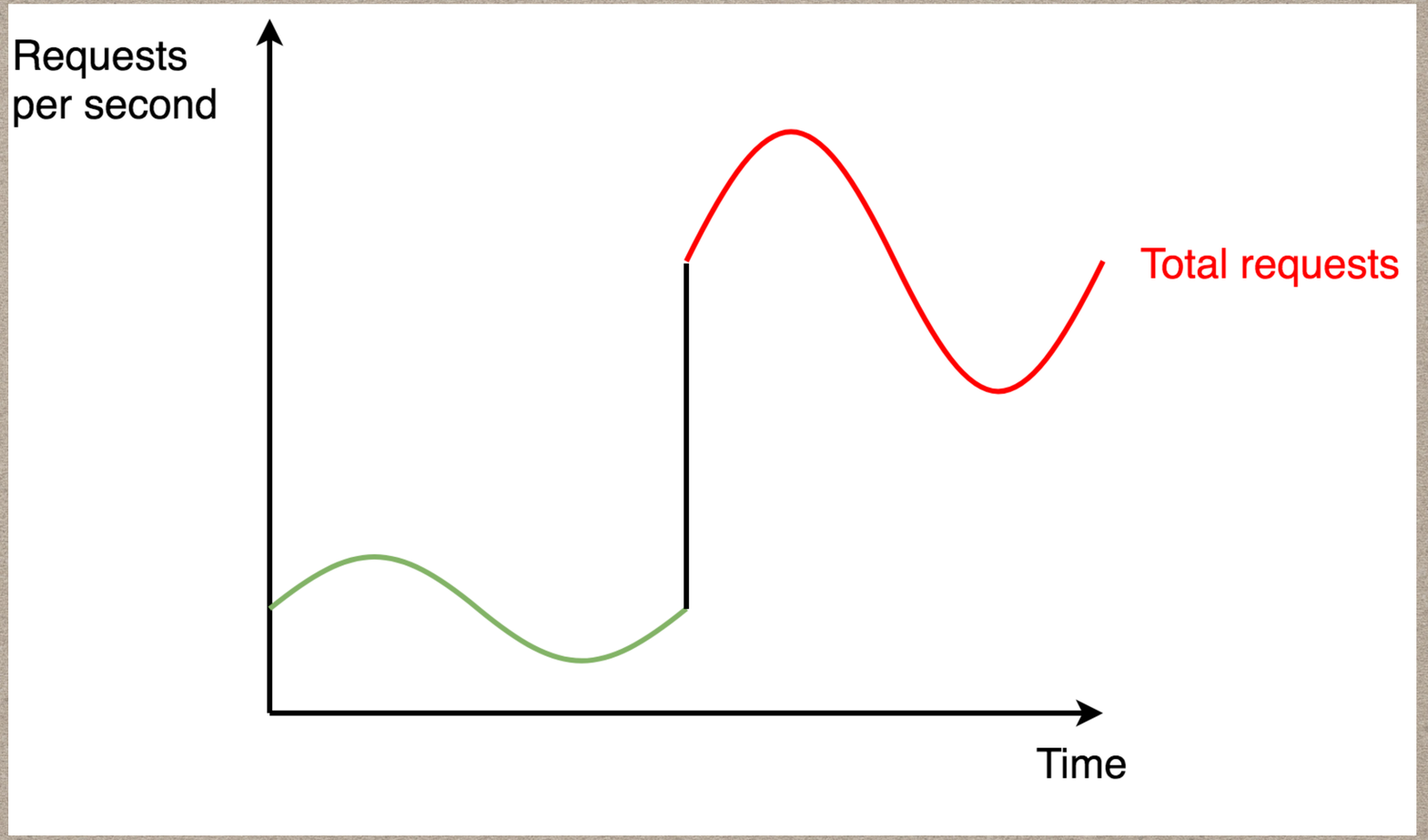
MULTIPLE REVERSE PROXIES

HOW DOES THE CLIENT KNOW WHICH REVERSE PROXY TO CONTACT?

DNS ROUND-ROBIN



AWS ELASTIC LOAD BALANCER



ELB COULDN'T HANDLE TRAFFIC SPIKES FAST ENOUGH

SCALING OUT: PRINCIPLES

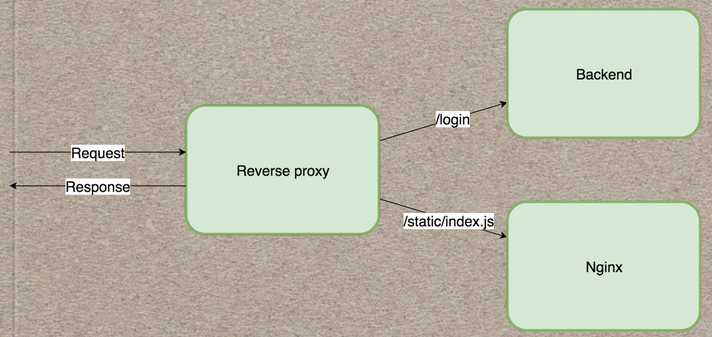
WHEN, WHAT, AND HOW

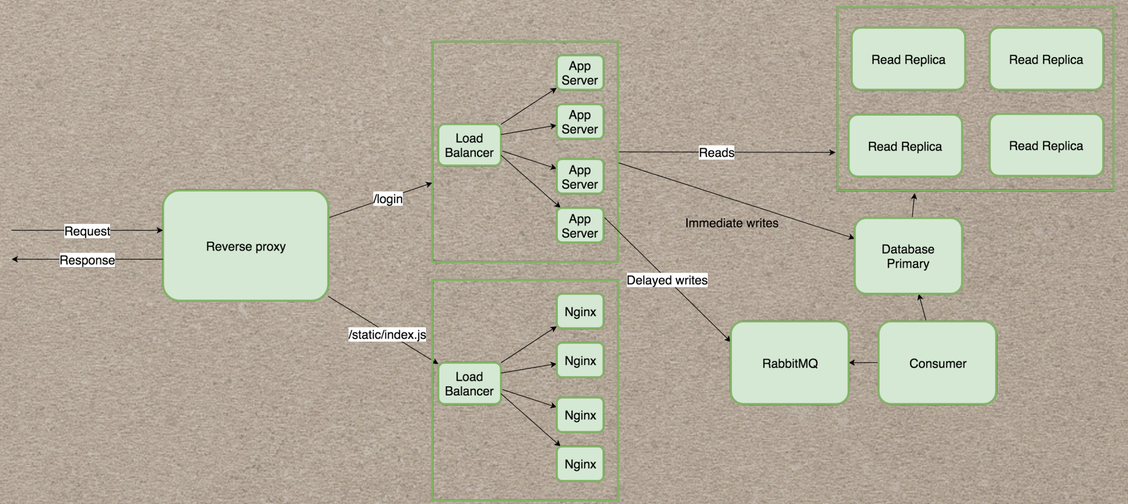
WHEN SHOULD YOU SCALE?

SHORTLY BEFORE THE LOAD BECOMES TOO MUCH FOR THE CURRENT SYSTEM

WHY NOT SCALE EARLIER?

- Which would you rather maintain?
- Notice all the “Need to think about [X]” as we scaled more and more





WHY SCALE *BEFORE* THE LOAD IS TOO MUCH?

- Your new system is likely to break the first few times you try it
- When things go wrong, you have an existing system to fall back to

WHAT SHOULD YOU SCALE?

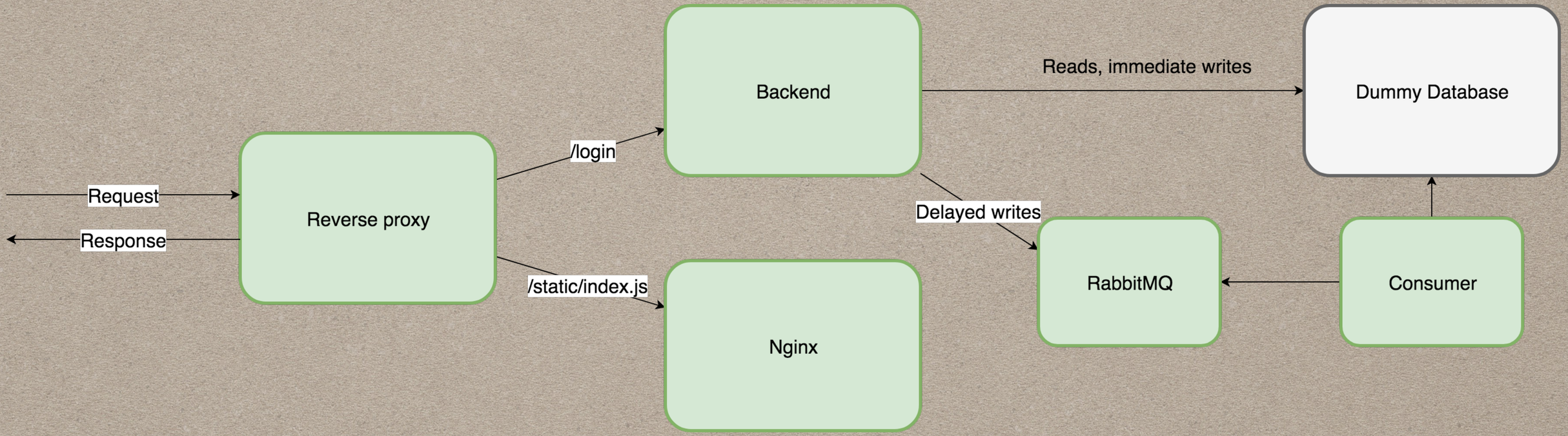
- Find the bottleneck in the existing system and scale that

HOW DO YOU FIND THE BOTTLENECK?

- This might be tricky in a large, inter-connected system
- For example, delays in the database could surface first as slow application requests

HOW DO YOU FIND THE BOTTLENECK?

- Instrument the system and look at metrics
- Profile the components
- Load test the system



LOAD TESTING WITH DUMMY REPLACEMENTS

HOW SHOULD YOU SCALE?

FIND THE SIMPLEST SOLUTION THAT WILL WORK FOR A REASONABLE AMOUNT OF TIME

"... THAT WILL WORK FOR A REASONABLE AMOUNT OF TIME"

- Try to predict the load in the future
  - Based on metrics growth, expected sales or marketing campaigns, etc.
- If a new system will take 2 months to build and solve your problems for the next 1 month, it's likely not worth it

THINK ABOUT EVOLVING YOUR ARCHITECTURE

QUESTIONS?

THANKS!

IVAN VERGILIEV