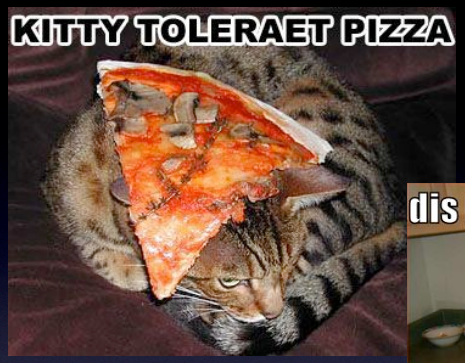

KITTY TOLERAET PIZZA


dis mine. dat his.
dont no wear yurs went...


MAH PEETSA
IS MOLTO BENNY

The evolution of load-balancing

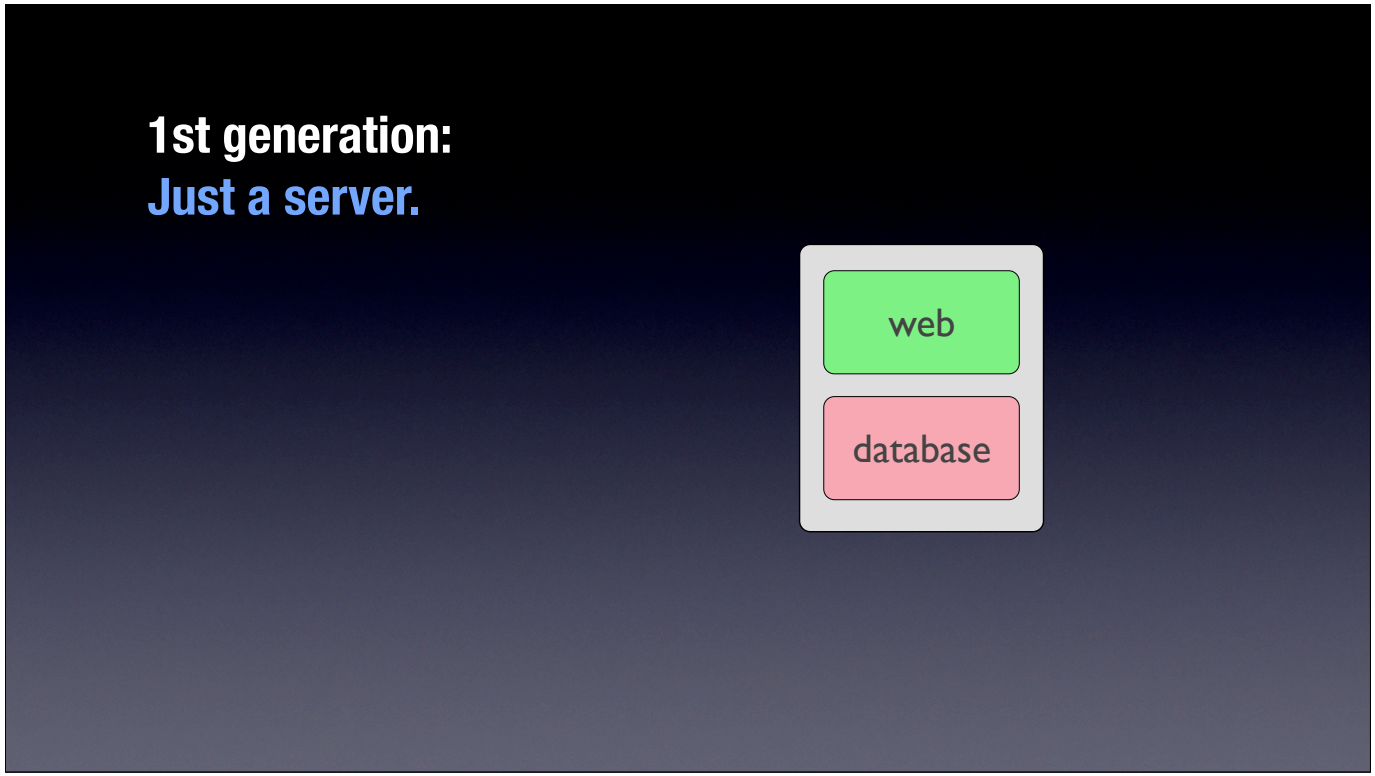
in a company remarkably like ours, with some sort of web application with a database, that might provide, say, invoicing.

- what do we want to accomplish?

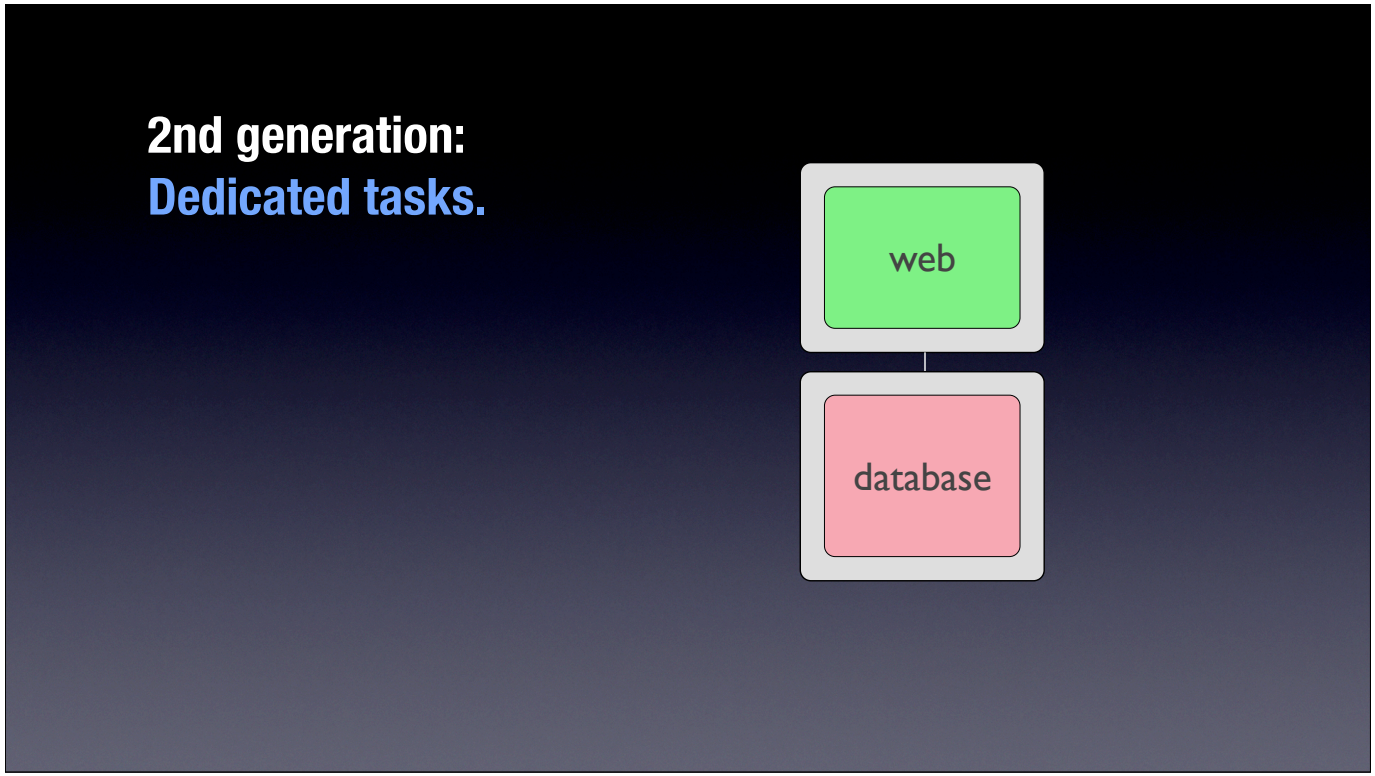
- the application is going to get busier as we get more successful
- which means taking up more server resources
- so we need to keep it running fast

- and people are counting on us to be available all the time
- turns out "all the time" is really difficult and expensive
- so it's really about minimizing downtime

- Performance and Reliability

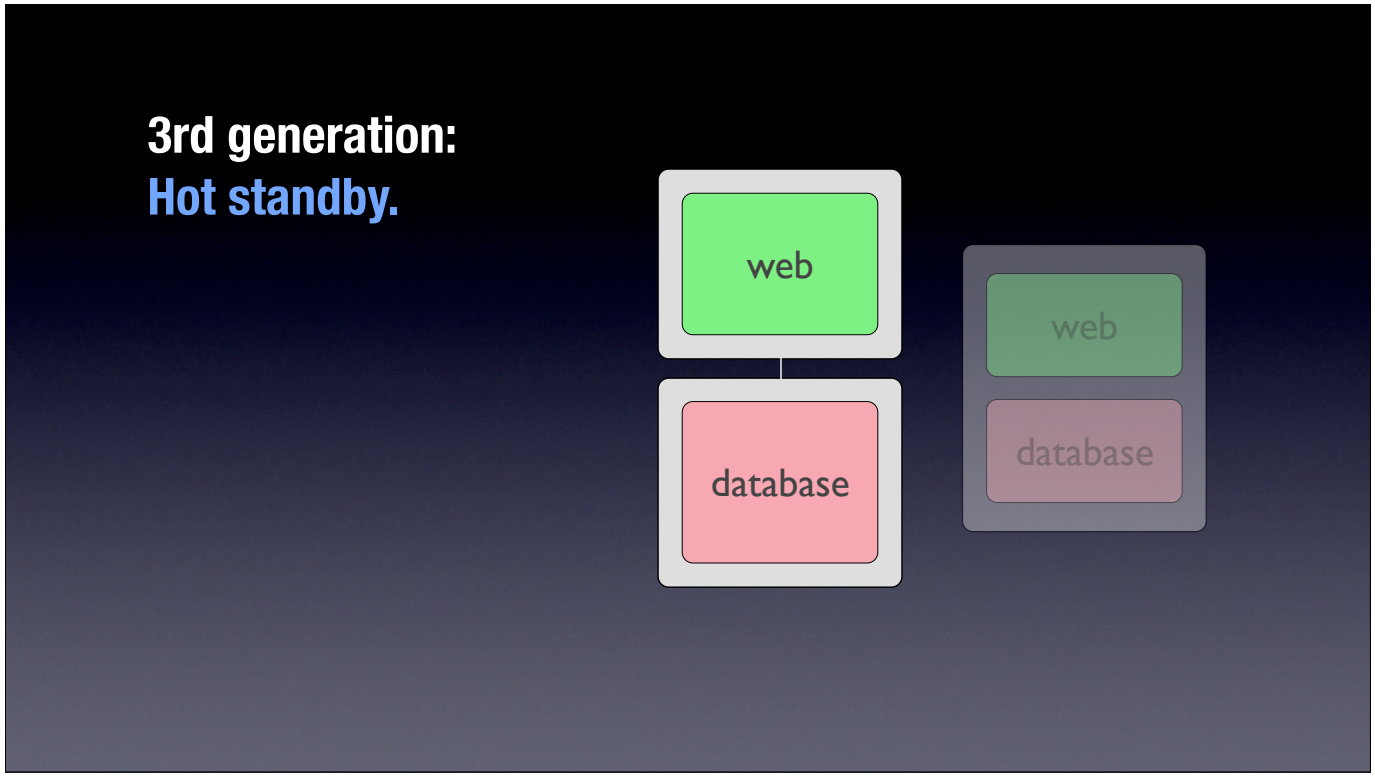


- Where everyone starts out
- Dunno if we did. probably?
- Competition for resources slows down

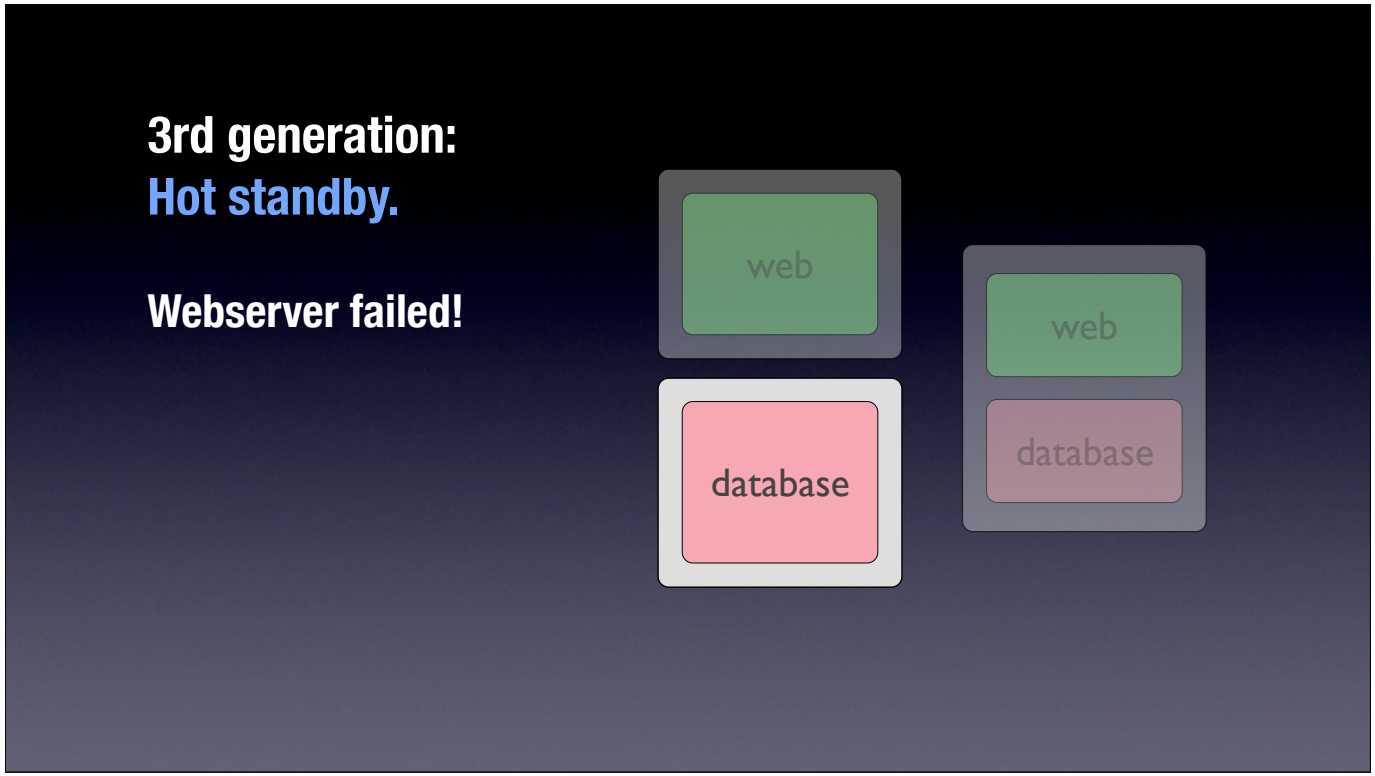


- Not competing for resources anymore
- Lightweight webserver, heavyweight database server
- Added benefit: Database server not publicly accessible anymore

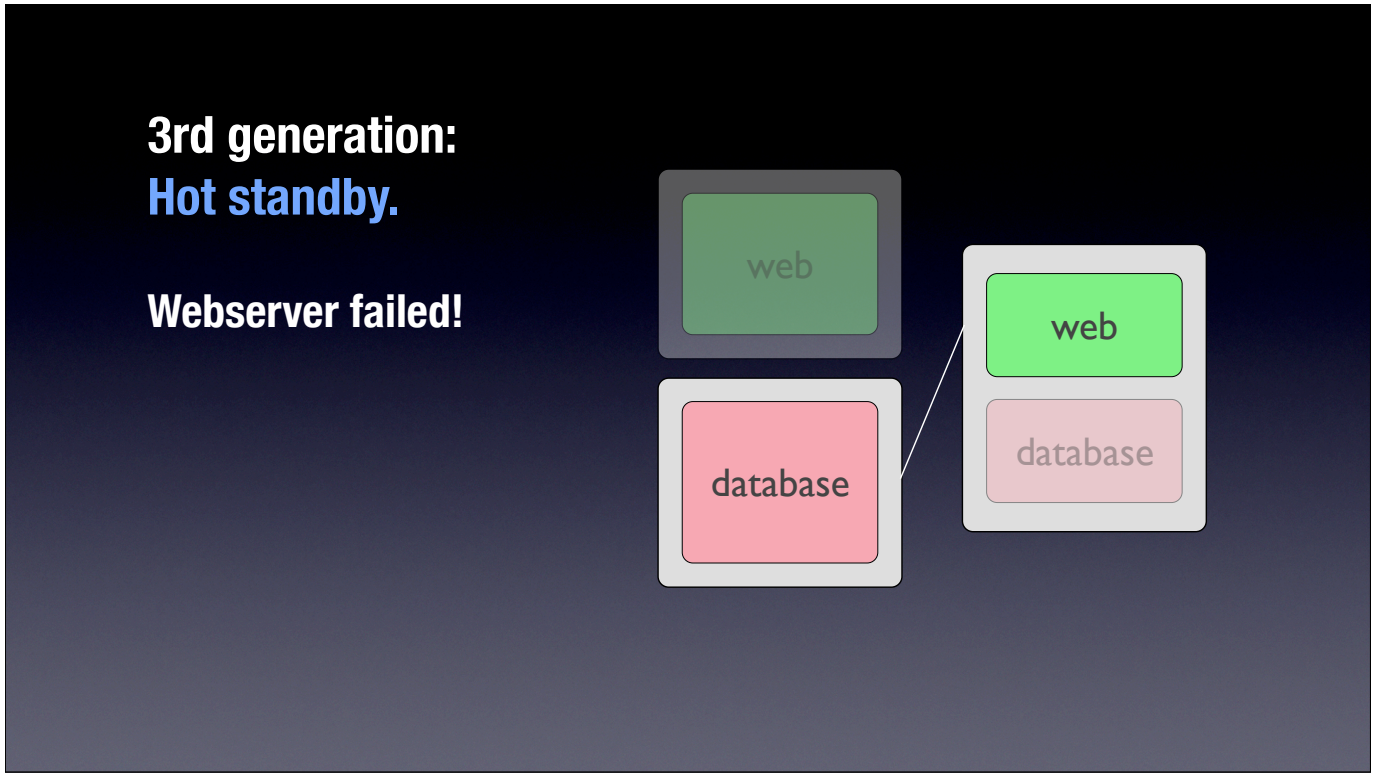
- Helps "run fast". Doesn't help "keep running"
- All of a sudden we've doubled the chances of



- Get an extra server in case something fails
- Prepared to take either role
- This is where we are right now



– Just bring up the standby as a webserver...

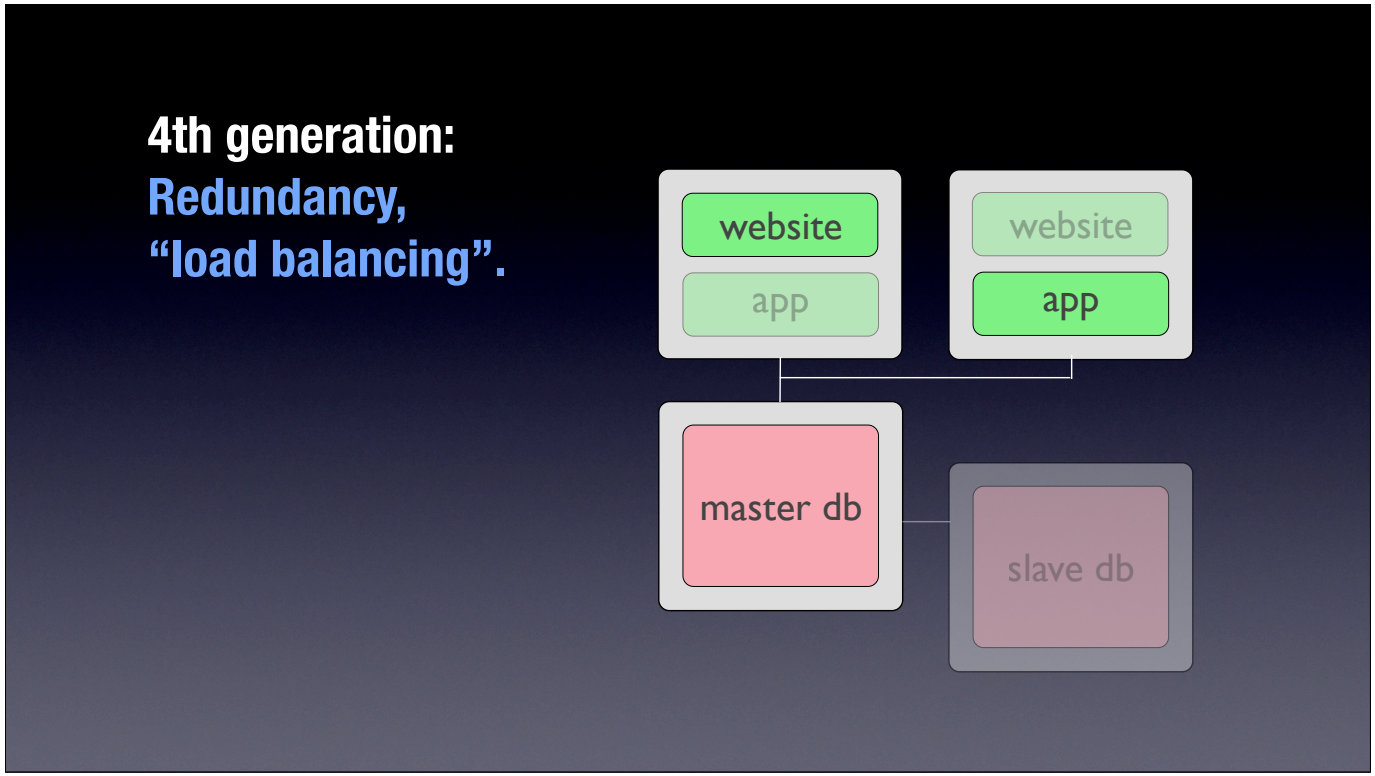


– and it's up and running again!

– Addressed reliability, but didn't help performance

– Paying for a box that just sits there doing nothing

– Tempting to put other things on that box



– Back to dedicating to web or to database (security)

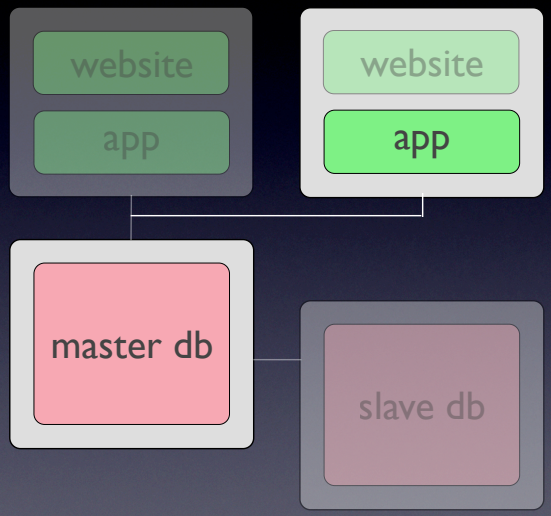
– Have to divide up tasks by type (website/app)

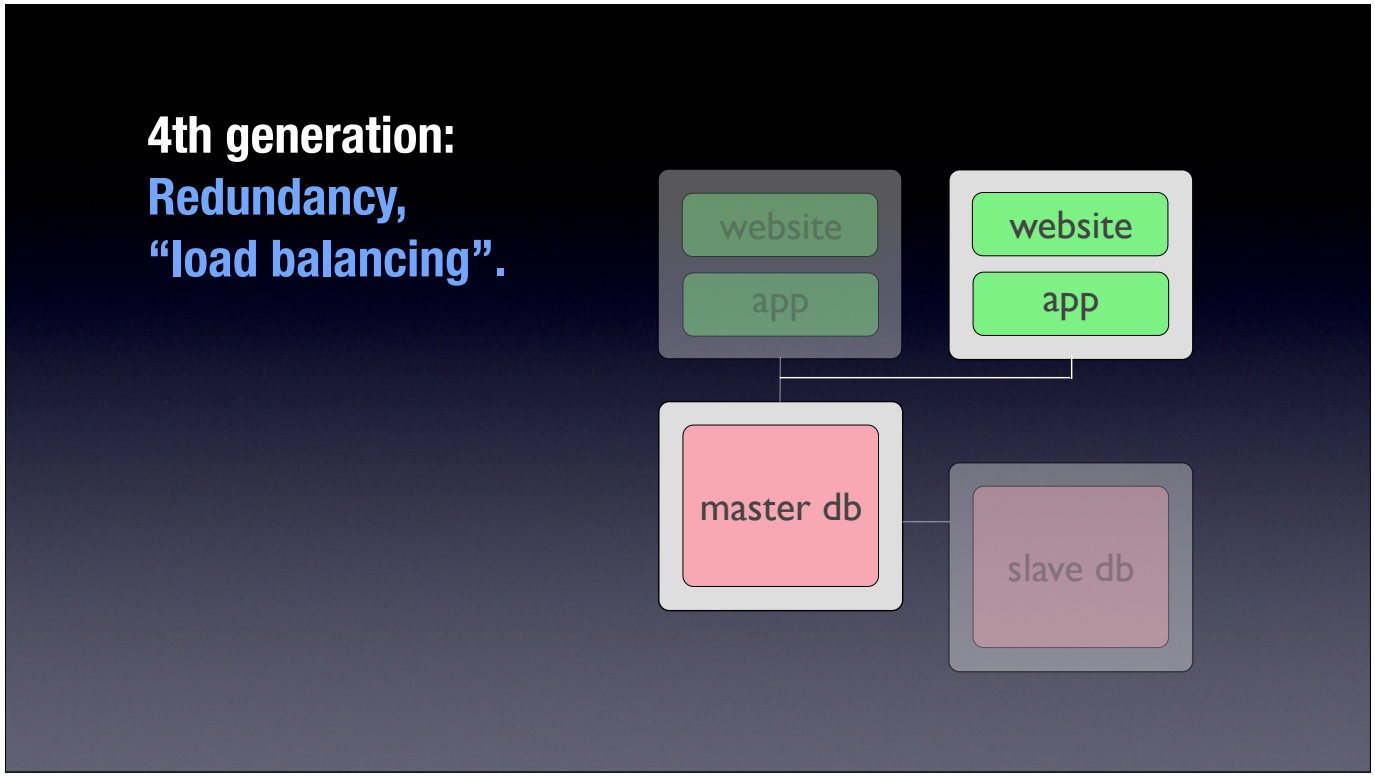
– Both webservers working hard

– "hot standby" database server turns out to be useful for backups

4th generation:
Redundancy,
“load balancing”.

Webserver fails!

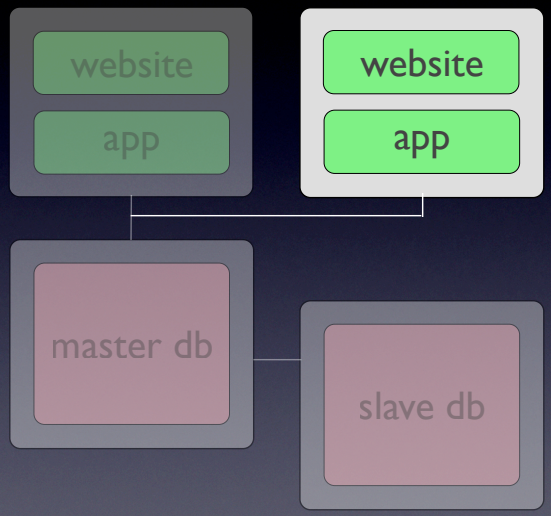


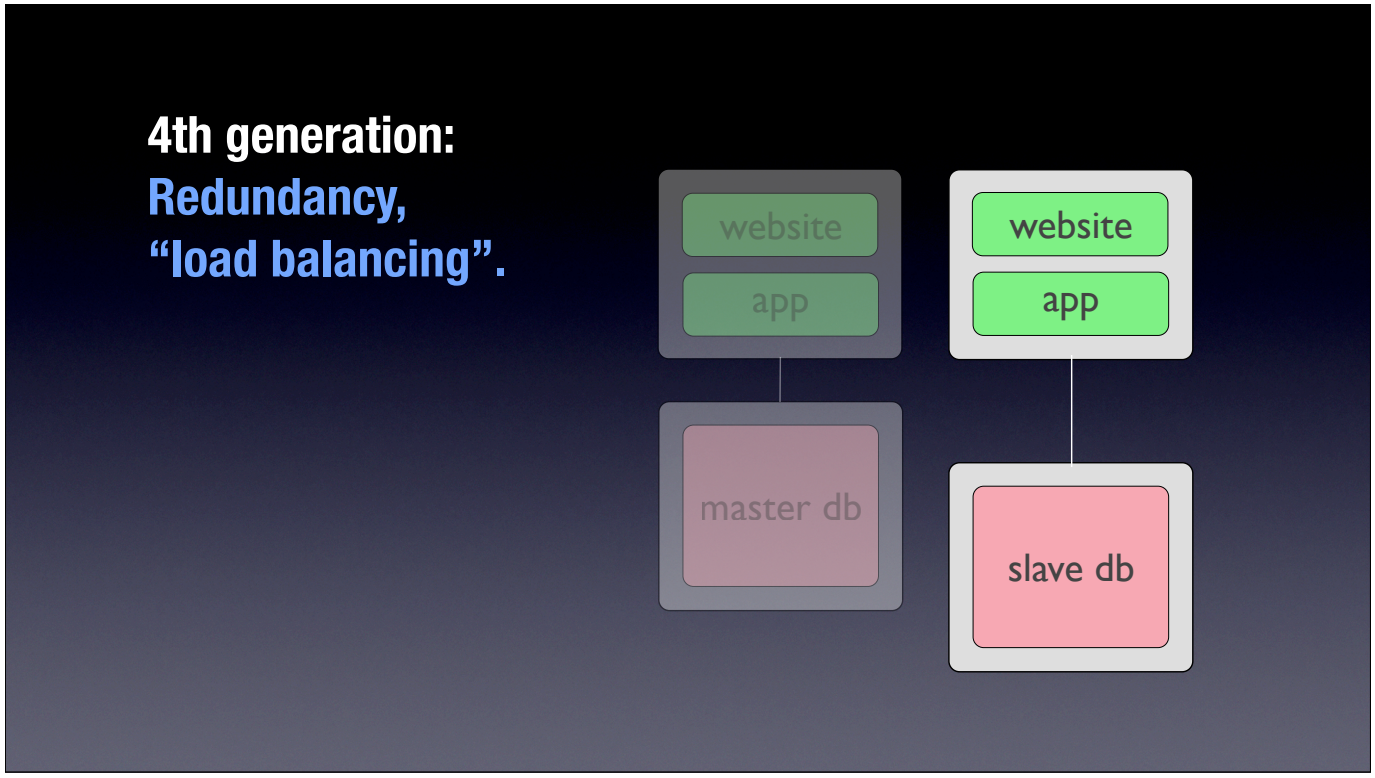


- just promote webserver!
- slows down a bit, but that accompanies failure

4th generation:
Redundancy,
“load balancing”.

Database server fails!





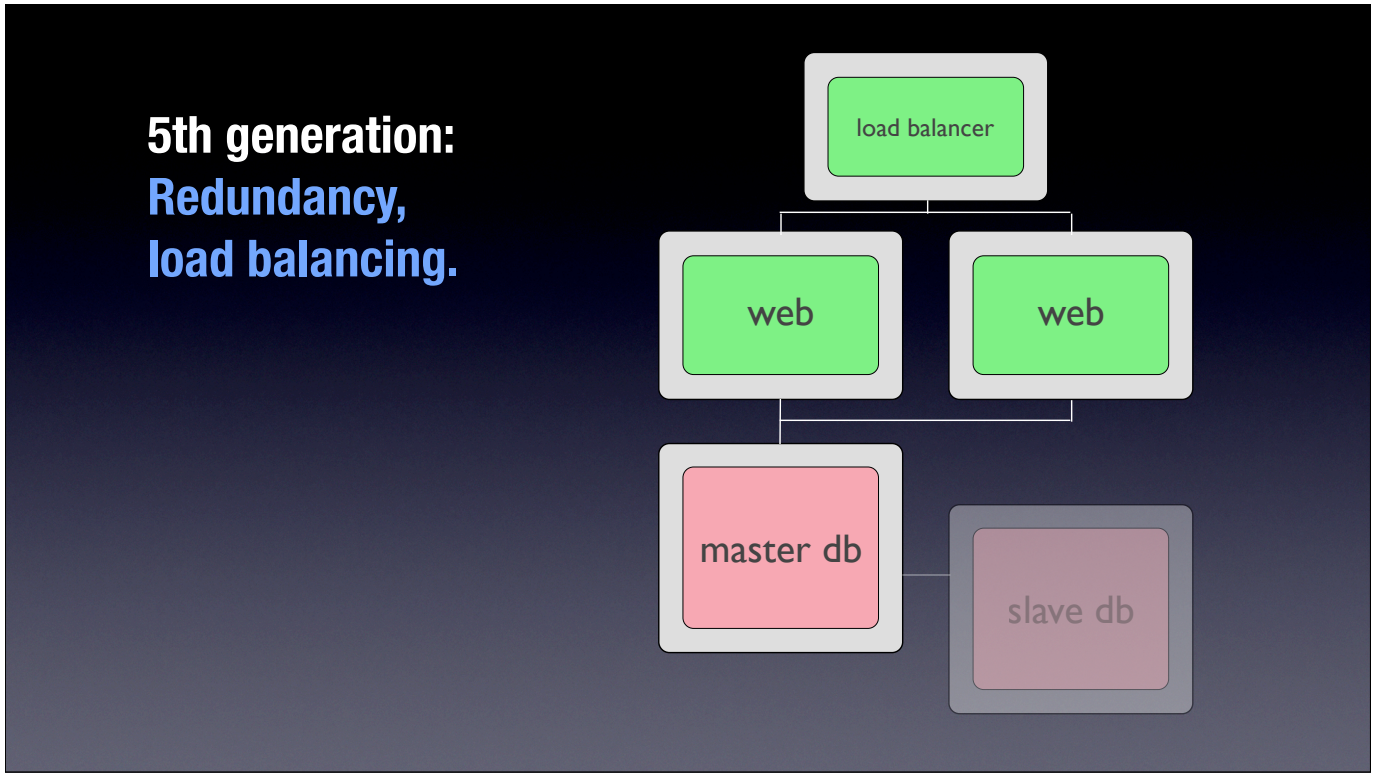
– just promote slave!

summary:

– Run fast: Splits up load, two webservers running all the time,

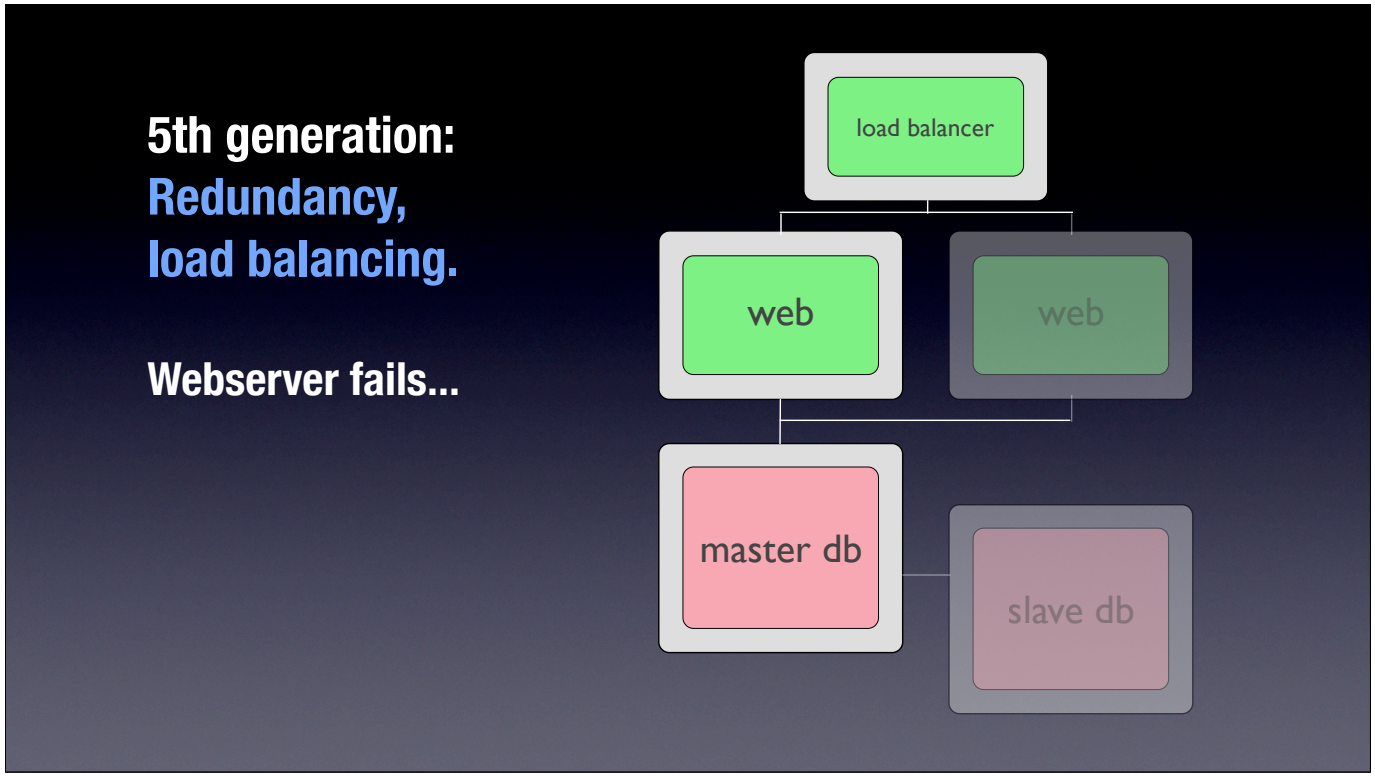
one can't step on the other

– Keep running: taking out one server doesn't hurt (much)

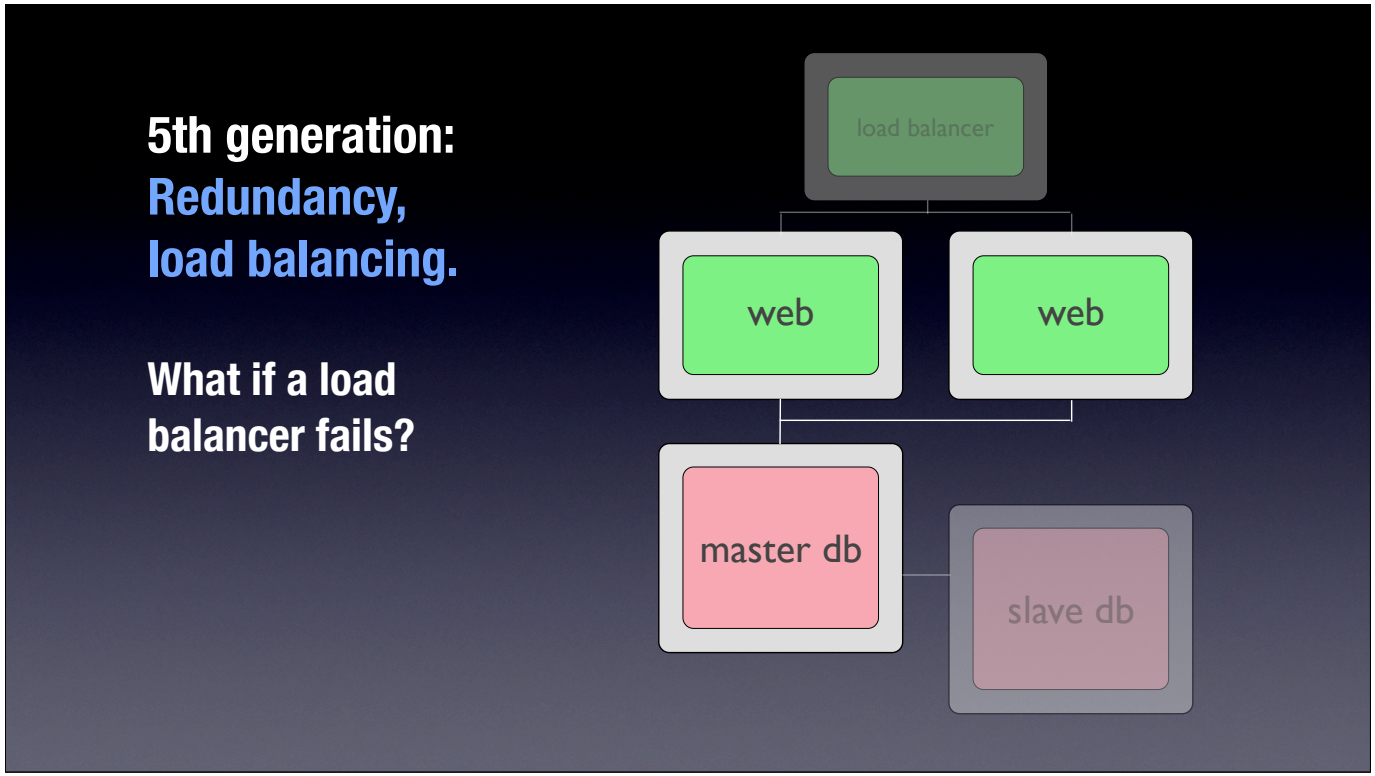


What does a load balancer do?

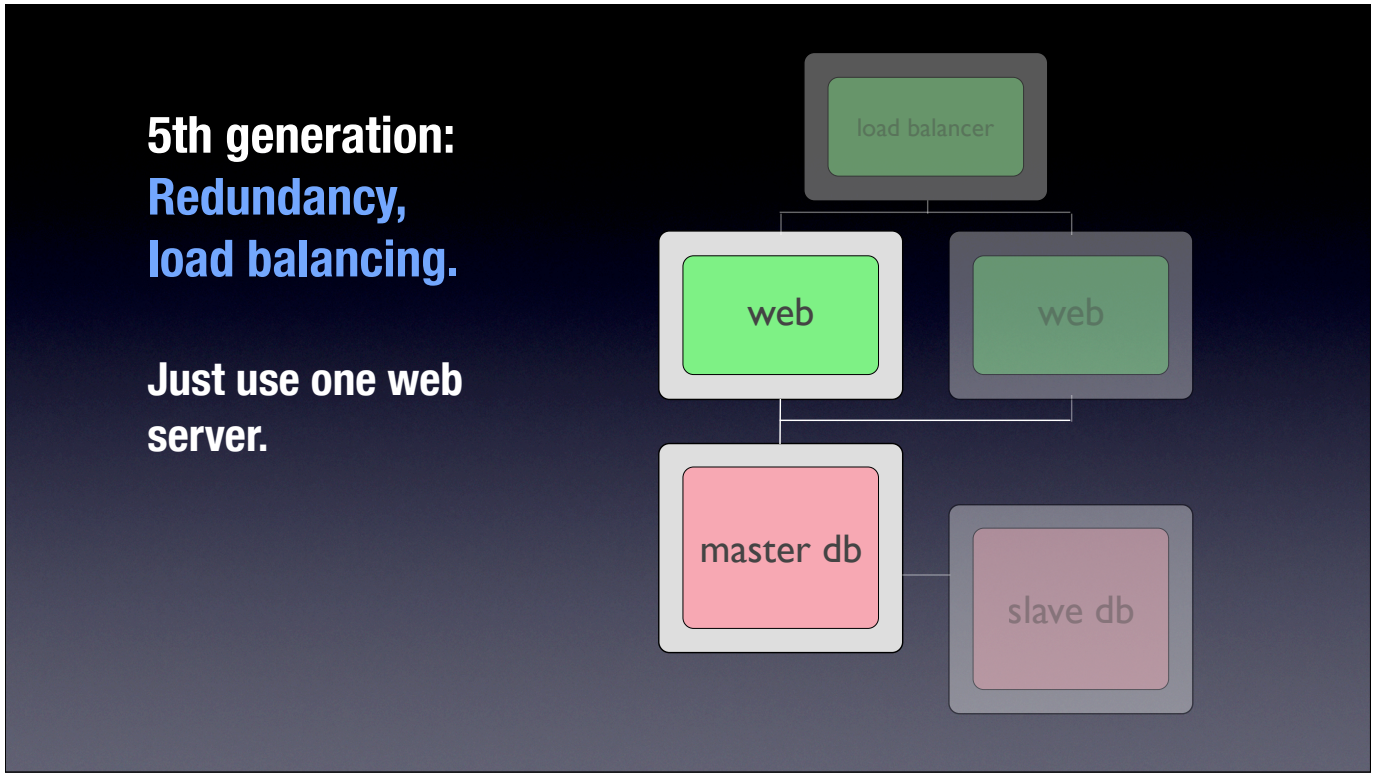
- takes request and hands it to a webserver "backend"
- webserver doesn't know anything's up
- load balancer watches response time, and prefers faster servers
  - fewer requests to slower (= busier) servers
  - no requests to failed servers



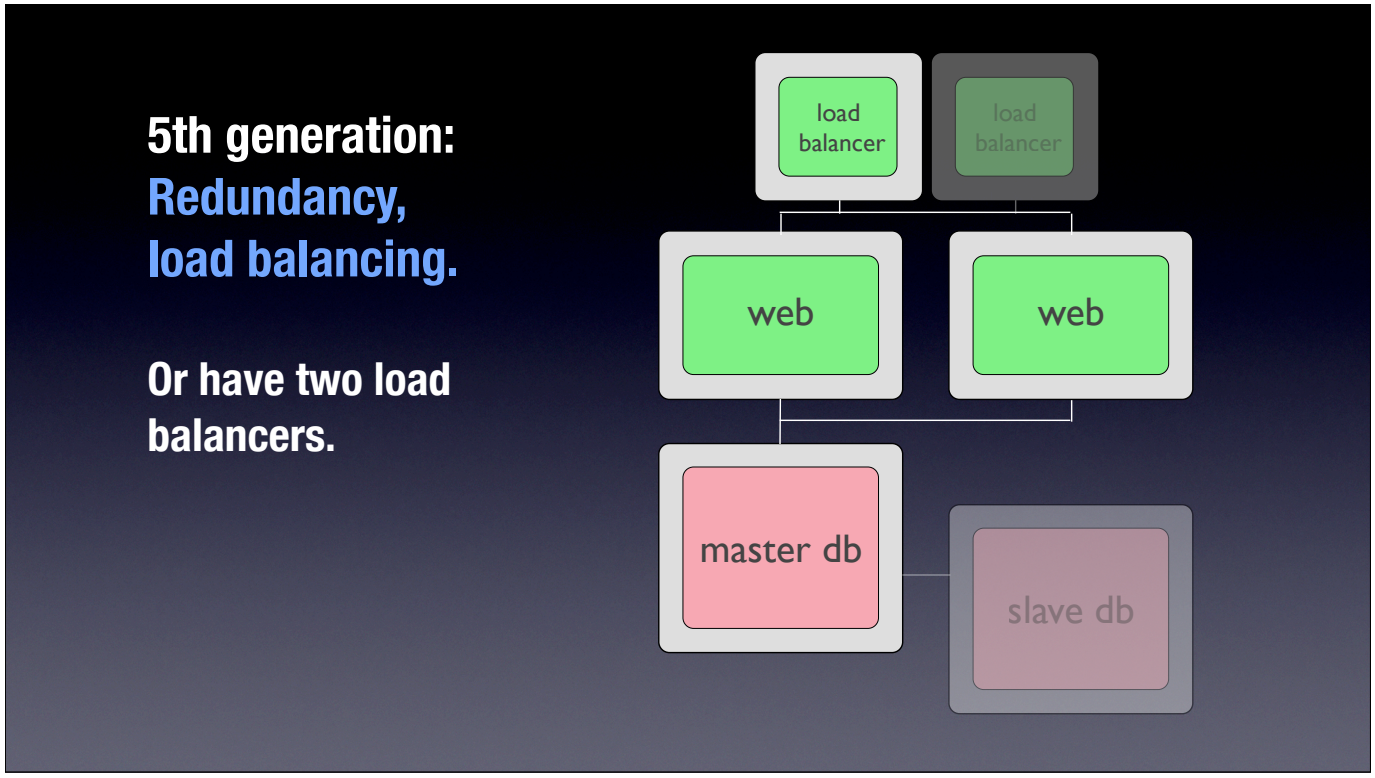
just keeps running



– in this setup, you're down to one webserver *anyhow*



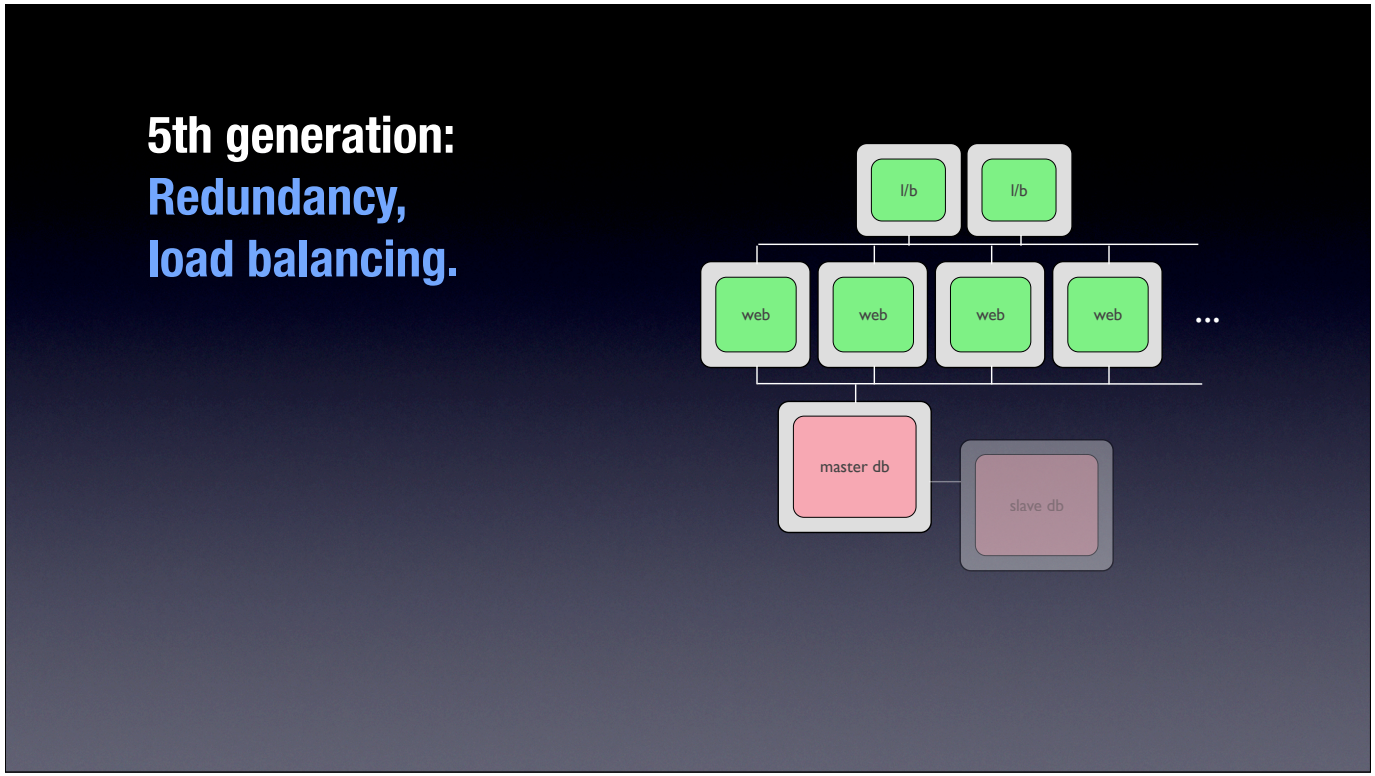
– so just use one webserver.



- when one fails, the other keeps going.
- this is not difficult to automate!

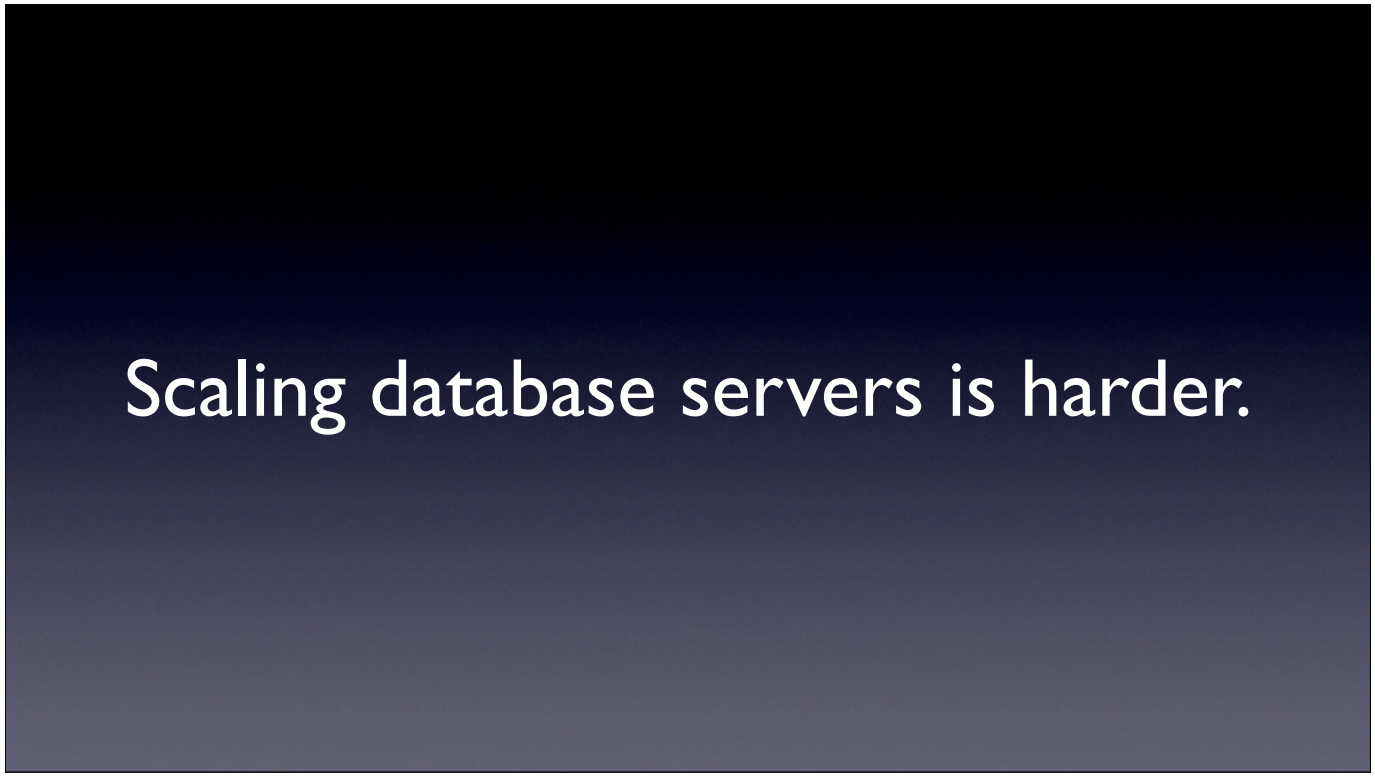
Automation so far

- Load-balancers each detect when a webserver fails
- Load-balancers together detect when each other fails



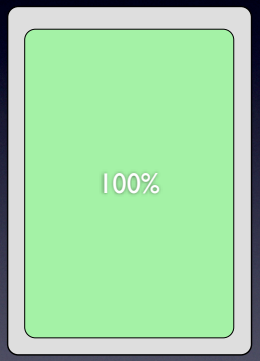
Web solved.

- That's basically how web load balancing works.
- It keeps scaling
- More resources with every server, and one failure means less and
less

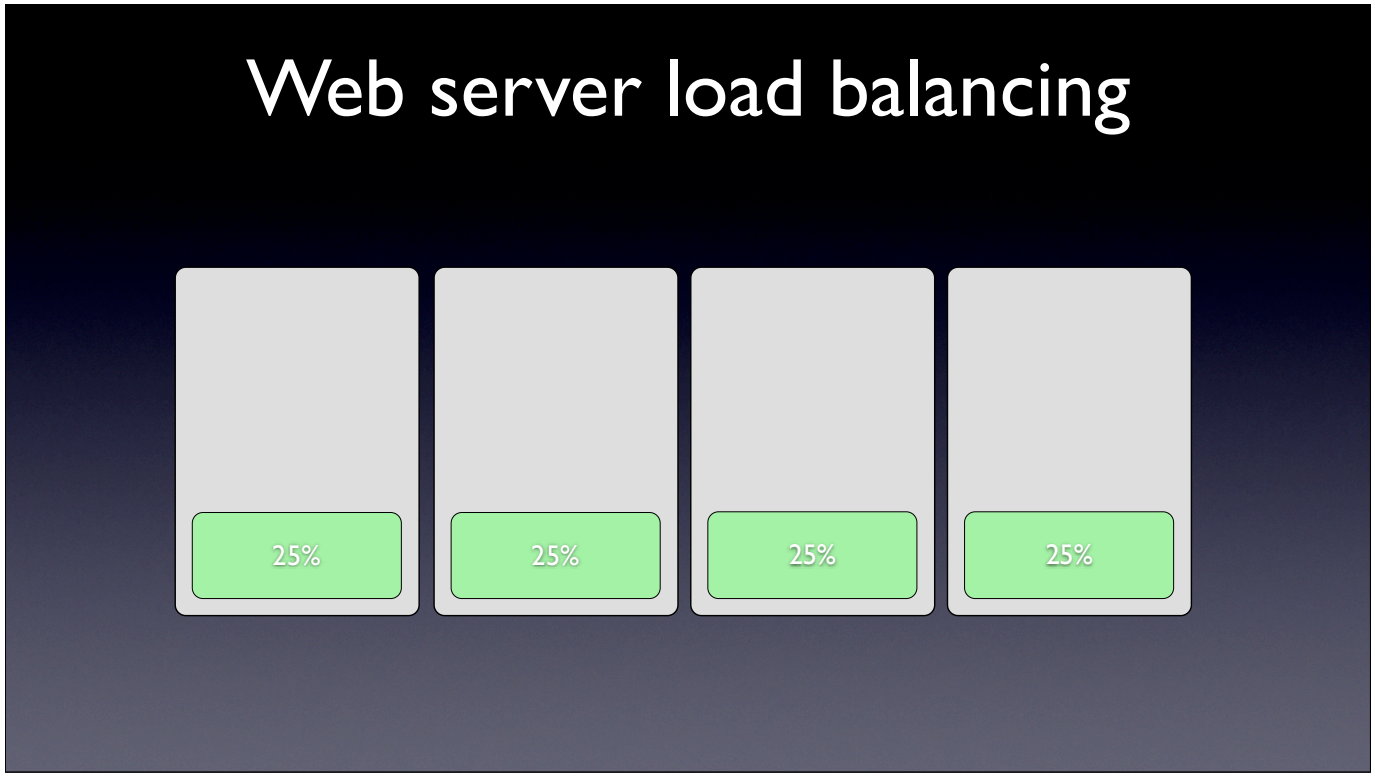


- Webservers can be ignorant of each other
- If one webserver handles request, the others don't.
- That's not true for databases.
- Look at how load changes with more servers...

Web server load balancing

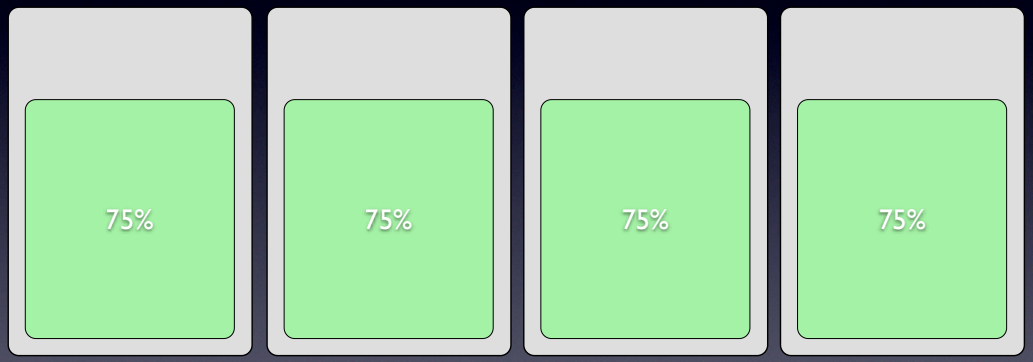


Web server load balancing

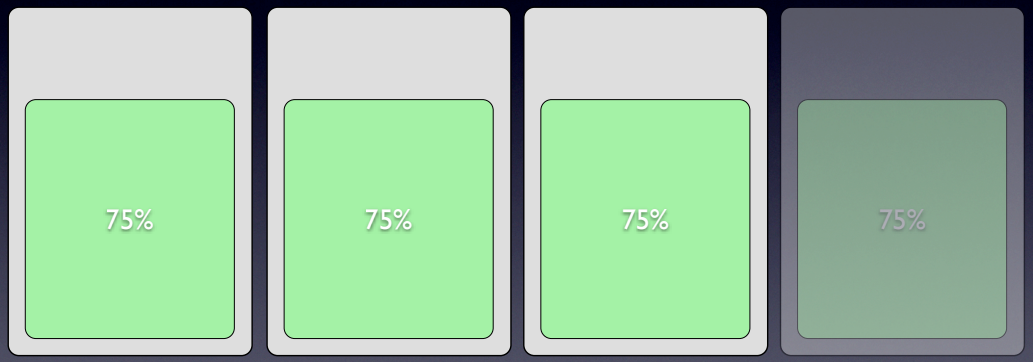


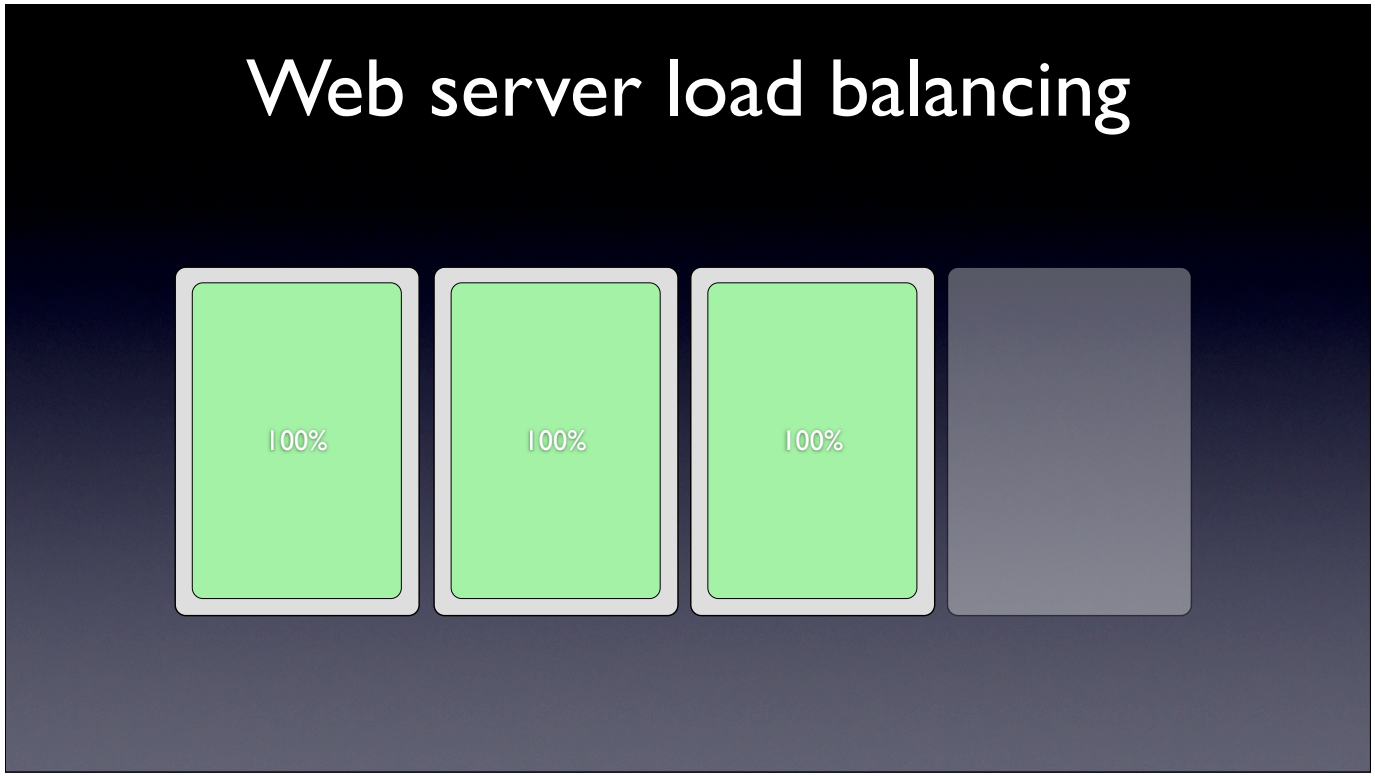
- Not *exactly* linear, but first approximation.

Web server load balancing

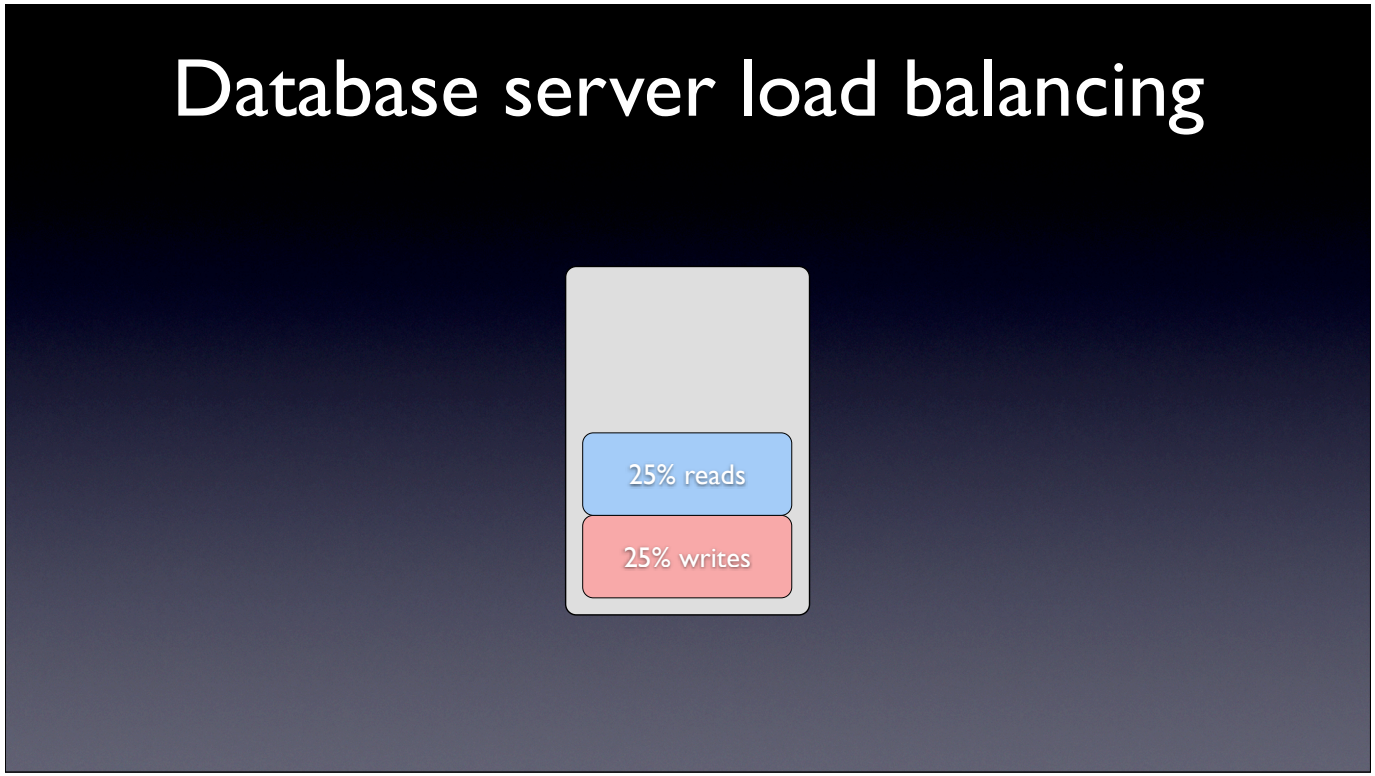


Web server load balancing





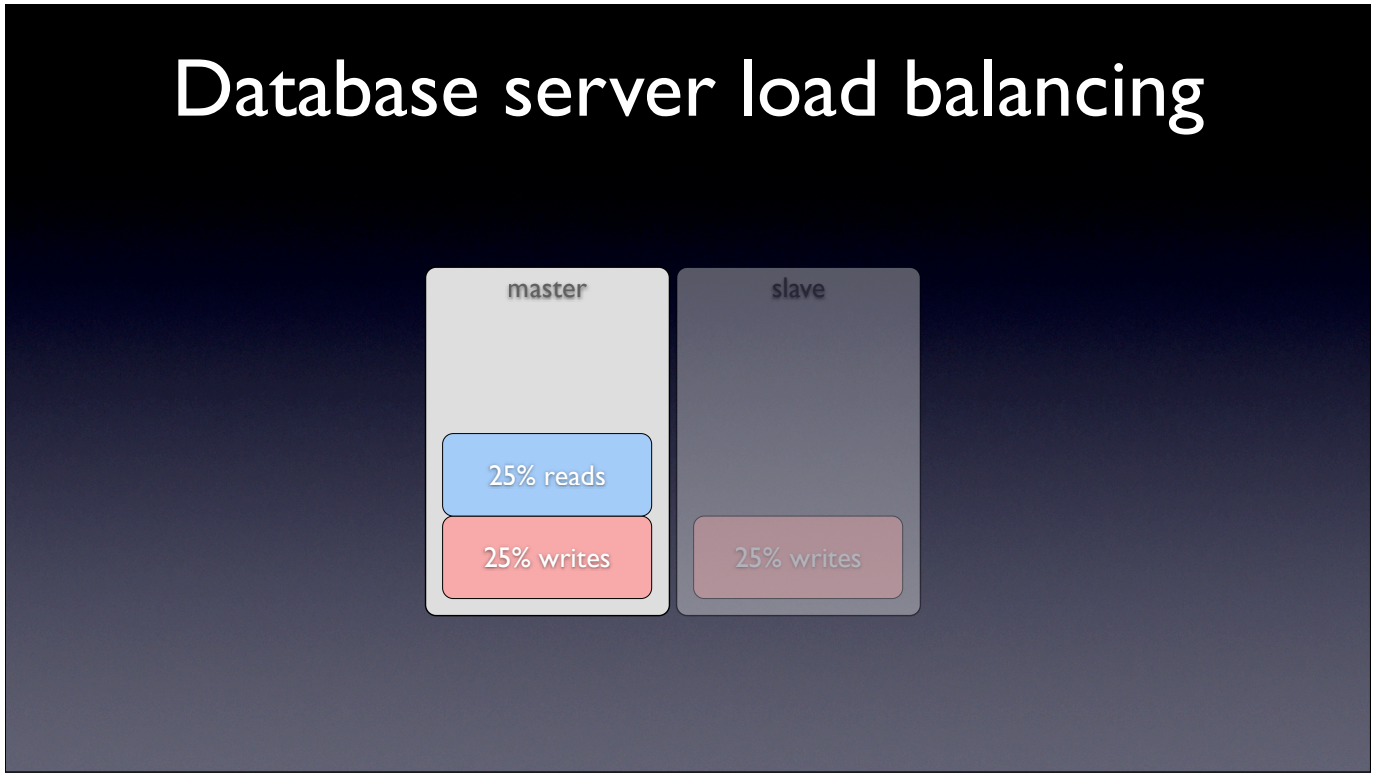
- capacity planning
- need to say "We can afford to have ___ fail"
- clearly, with 4 at 75%, we can afford to have 0 fail.
- Need to have 1/N room.



- Difference here is reads and writes
- You can read from any database server
- But that means that writes have to happen to *all* of them.

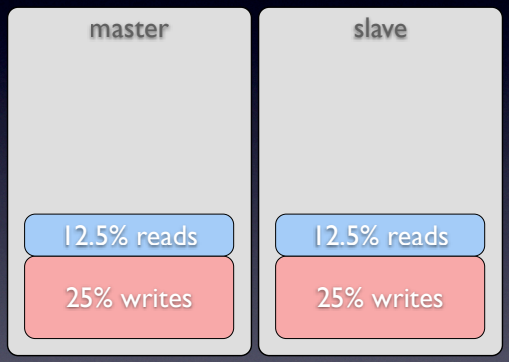
- So here's a half-loaded database server
- Half reads, half writes. Not realistic, usually much more reads

Database server load balancing



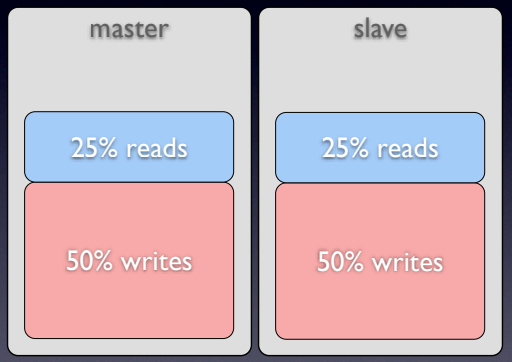
- Replication takes the writes from one and runs them on another
  - actually copies SQL statements over
  - Note that this *increased* the number of operations
  - No performance benefit!

Database server load balancing

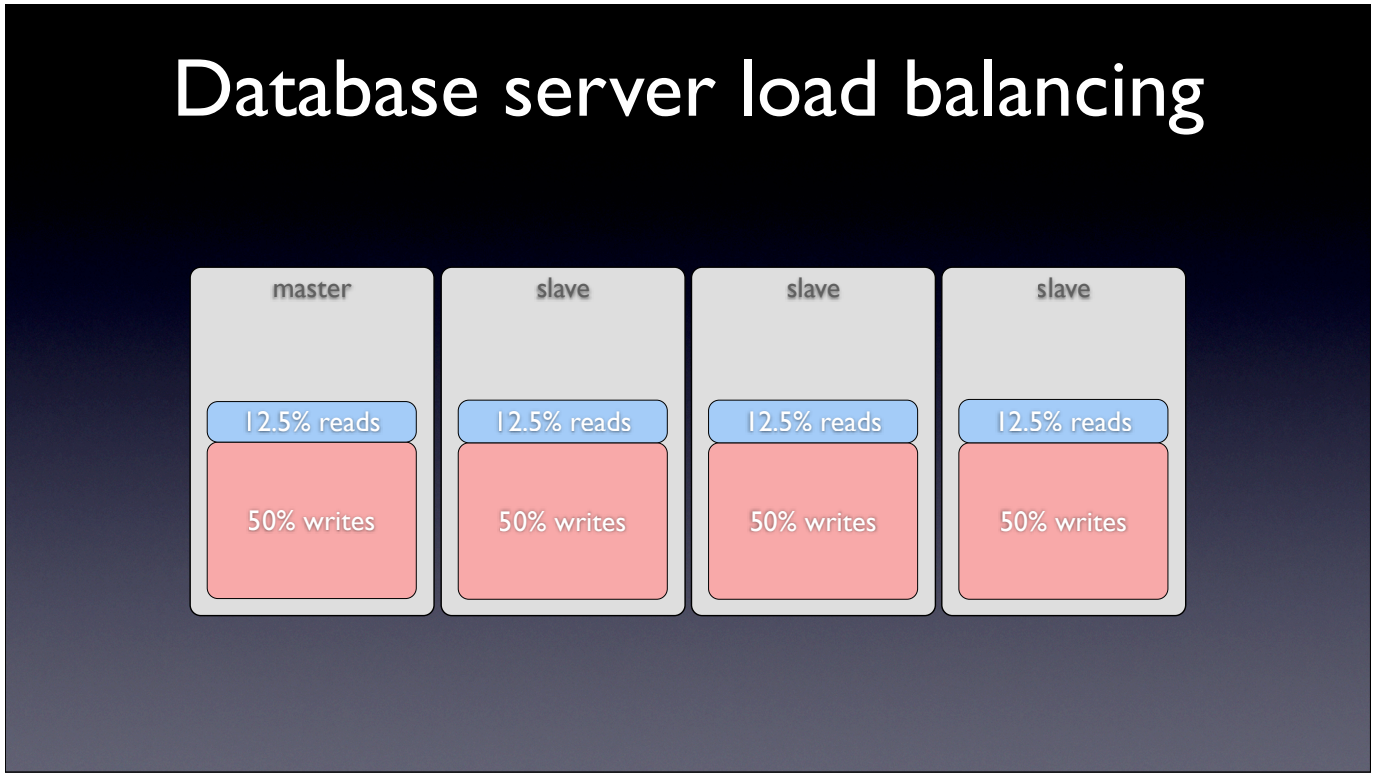


- Aha, we're load-balanced now!
- Wait, we've gone from 50% utilization to 37% even though we doubled
  the amount of hardware.
- Reads are independent
- Writes are dependent!

Database server load balancing

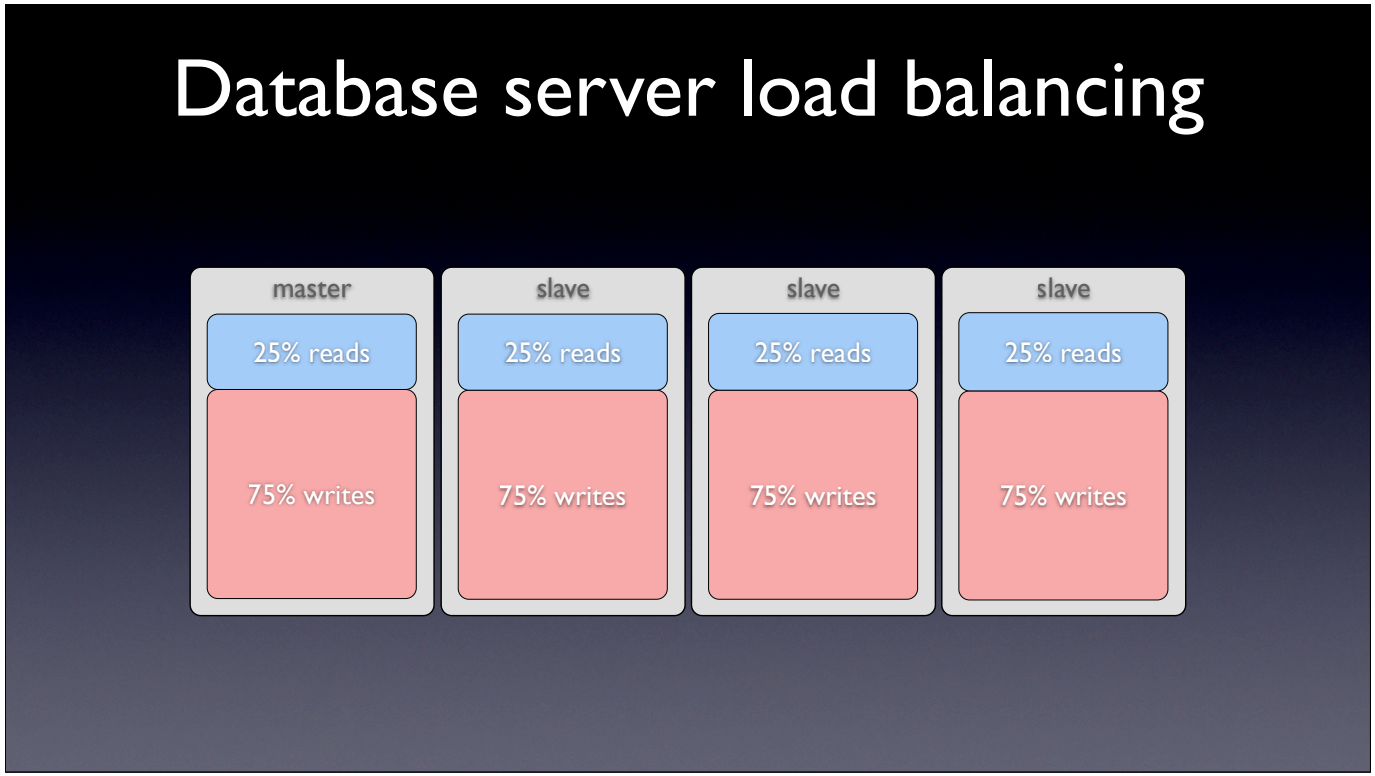


- twice as busy
- both 75% utilized! do something!



GET MORE!

- uh oh.
- Two more servers only got us from 75% to 62.5%.
- Clearly this isn't going to work.



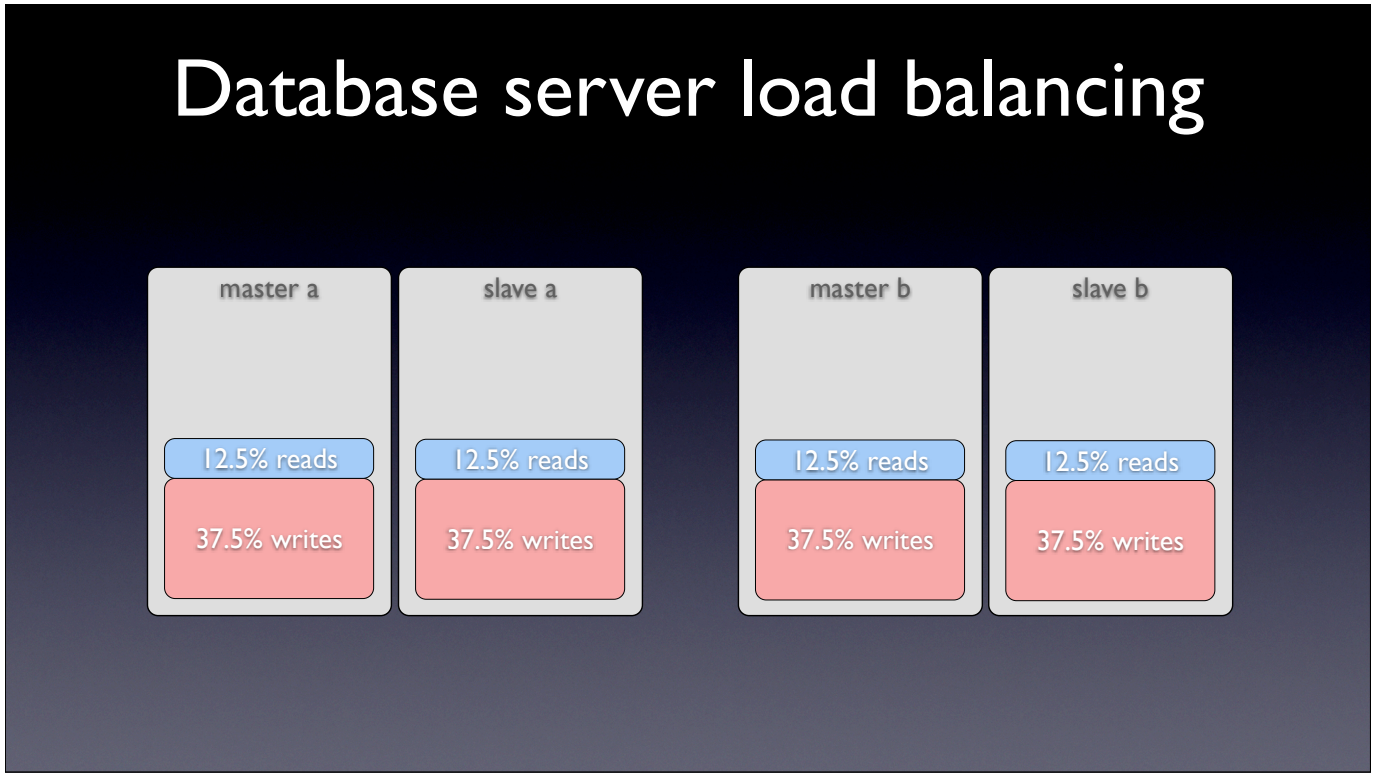
– Now adding more servers is just going to share that 25% across.

– One more takes us from 100% to 95%.

– FOUR more takes us from 100% to 87.5%.

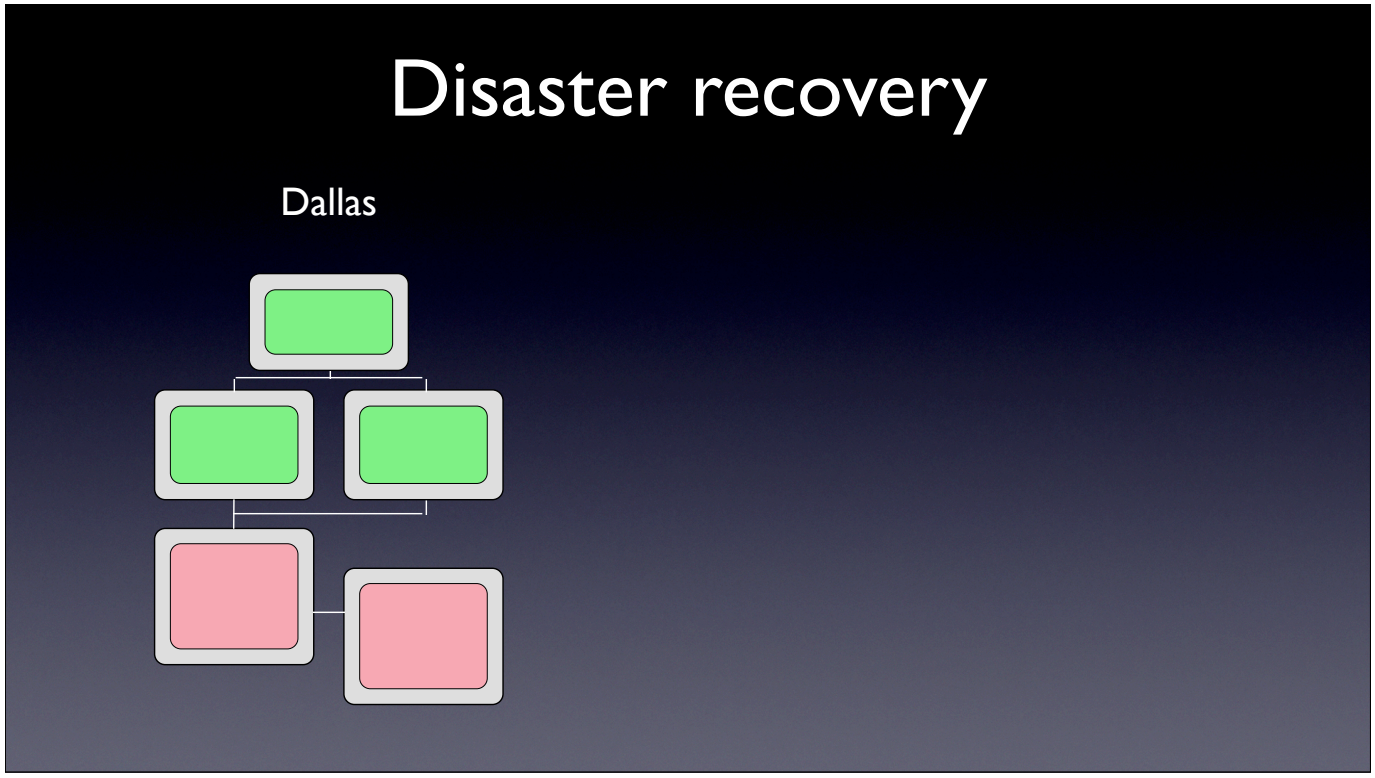
– What if one fails?

– Writes slowly consume all the headroom.

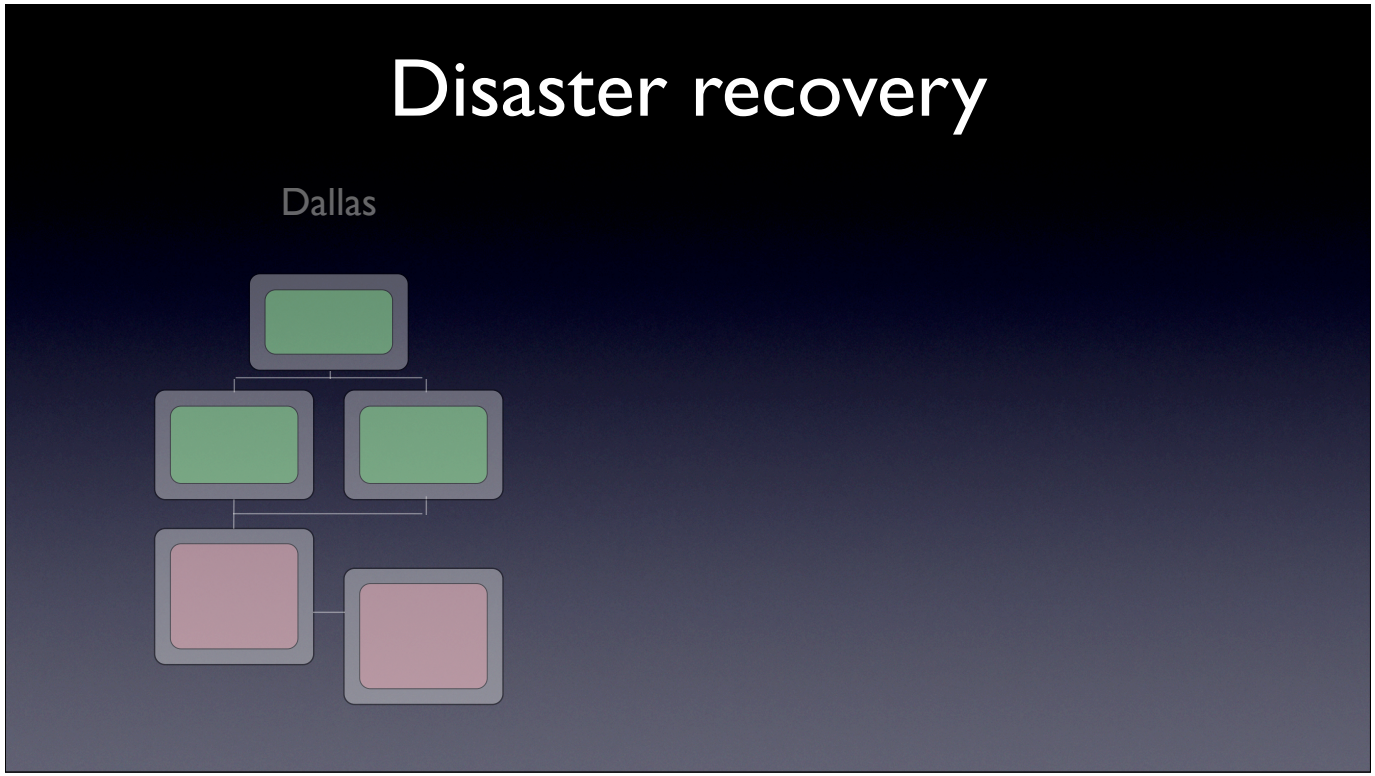


- Introduce independence
- Cut write load in half, literally
- Note that we still need pairs, so we have redundancy
- Expensive move: code has to account for "where is the data?"
    - and "Where do I put this new data?"
- ORM solves part of this

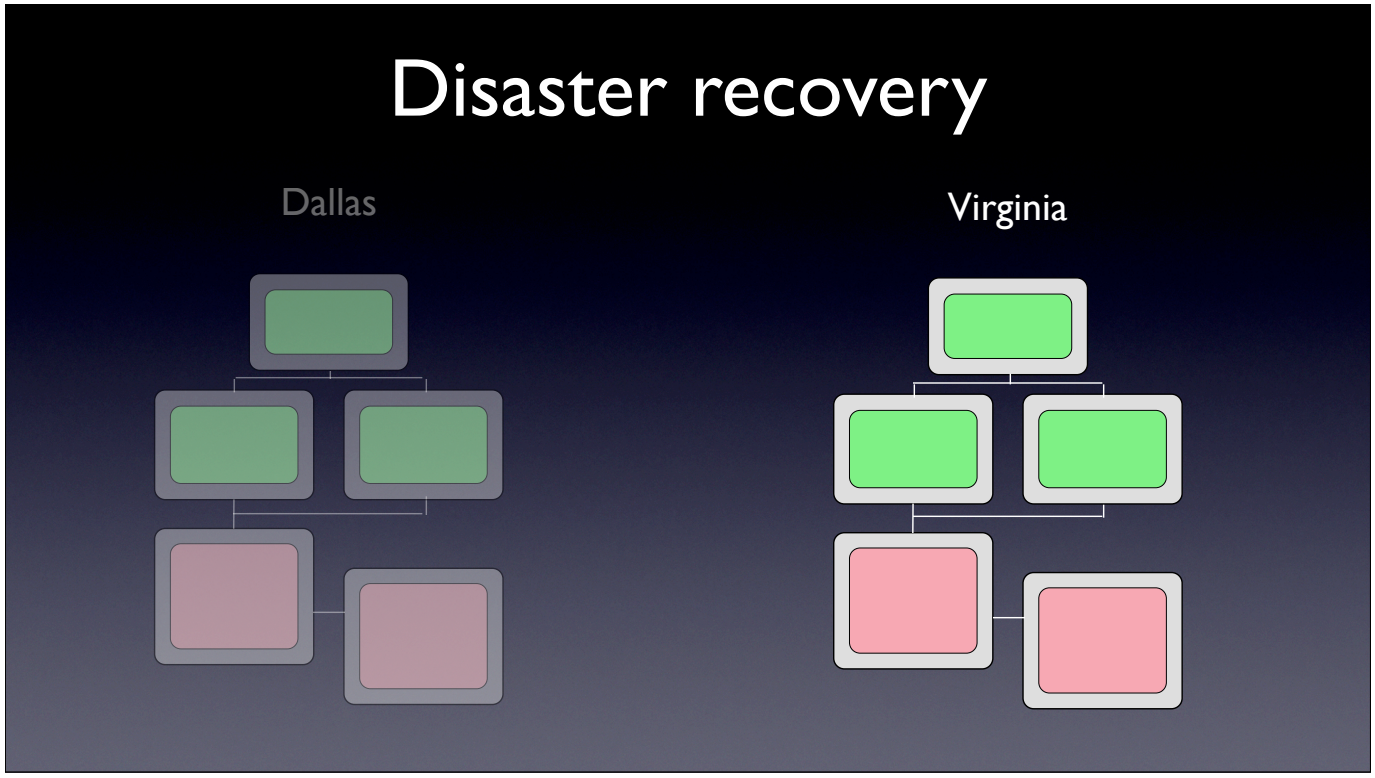
- Haven't talked about disaster recovery.



- Purring along normally, then a truck runs into the transformer.
- This happened to us last.. November?

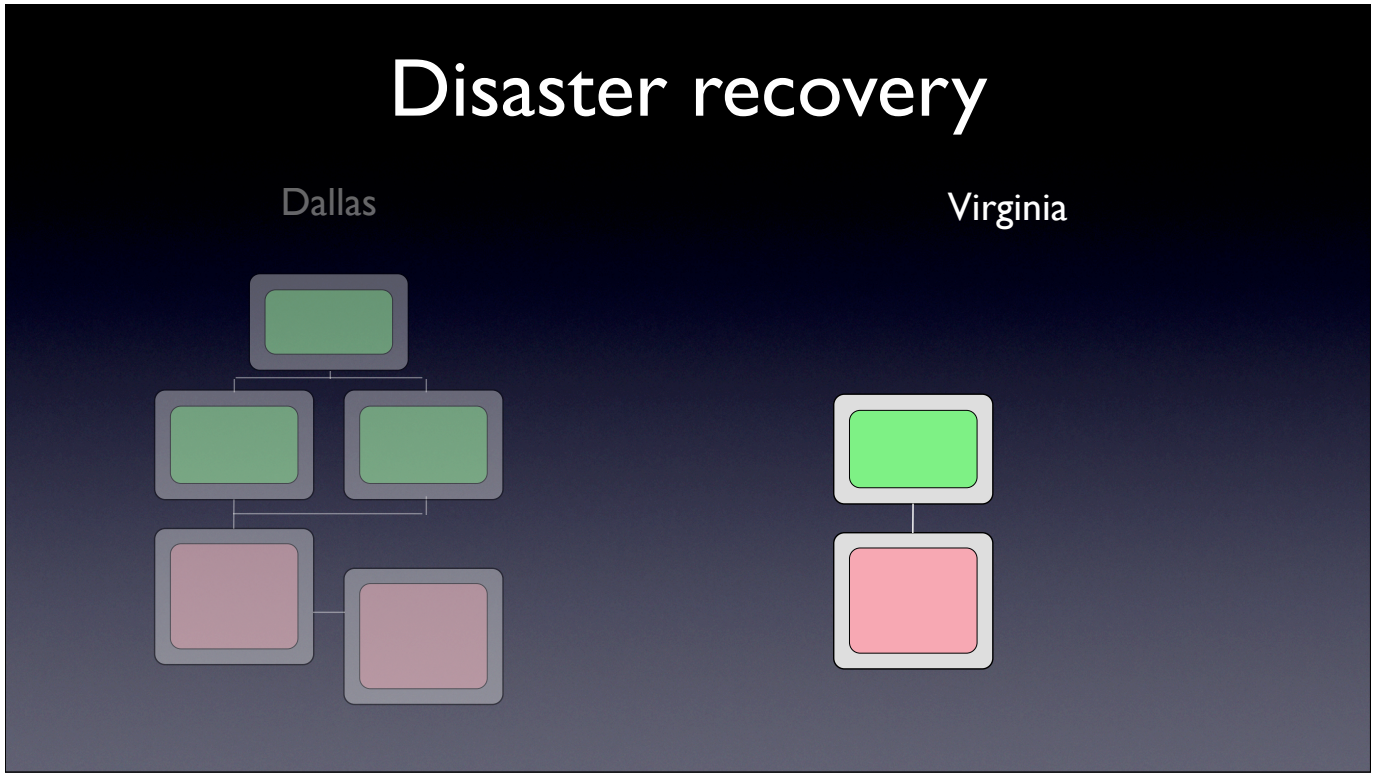


- All of a sudden you have no servers at all.



DISASTER RECOVERY SITE

- Copy of production site ready to go
- This doubles your IT budget for things you can't use.
  - If you use them, you can't fail over to them
  - Or if you do, where do you put the things you used?



- Bare-bones setup in Virginia
- Enough to "limp by"
- Failing over would be a last resort
- Solves budget problem, but not the maintain-and-recover issue

- This is partly a marketing feature rather than something we'd

Run fast, keep running.