

# Evolution of Web Application Architecture

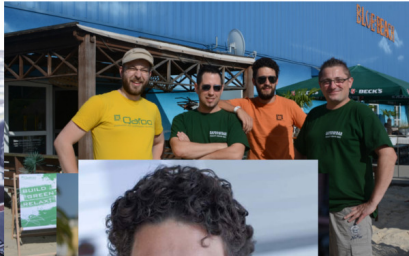
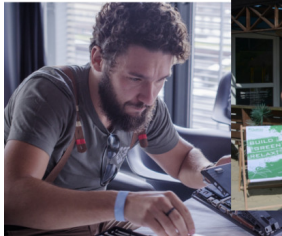
PHPDay Italy

Kore Nordmann / @koredn / <kore@qafoo.com>  
May 14th, 2016



# Hi, I'm Kore

---

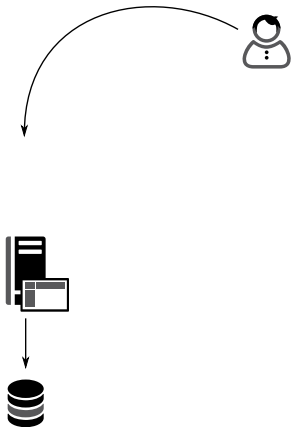


Architecture



# Evolution

---

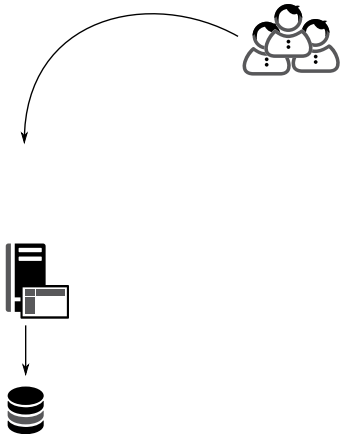


# Too many visitors



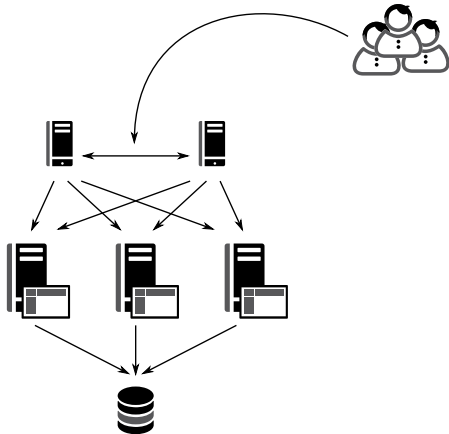
# Evolution

---



# Evolution

---



# Lessons Learned: Load Balancing

---

- ▶ Works because of HTTP & PHP
  - ▶ HTTP is LCoDC\$SS
  - ▶ PHP is build for shared-nothing
- ▶ Round Robin works best
  - ▶ Sticky sessions will overload certain servers





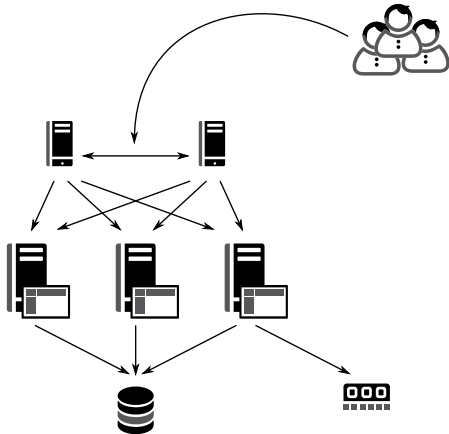
A hallway with white brick walls and a polished floor. On the right wall, a pixelated character is created using yellow, blue, white, pink, and black sticky notes. The character has a large head, small eyes, and a wide, open mouth. The hallway recedes into the distance on the left.

**Non sticky session  
- how?**



# Evolution

---



# Lessons Learned: Non-Sticky Session

---

- ▶ Put session on memcached / Redis
  - ▶ Mostly trivial because of existing extensions



IC FRUIT EXPRESS

P. F. E.

26704

CAP. 1000 LBS.

W. L. 1000 LBS.

HTWTS 1000 LBS.

13

THE DENVER ICE & COLD STORAGE CO.

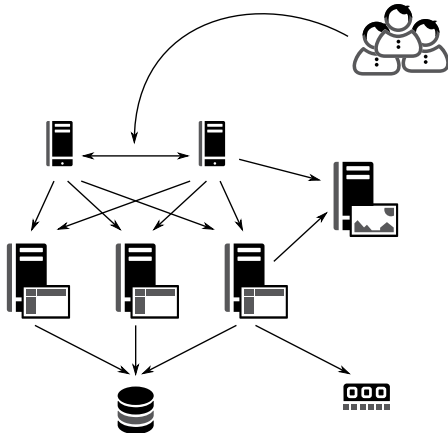
THE D

Where to put the  
static data?



# Evolution

---



# Lessons Learned: Static Files

---

- ▶ NFS will eventually lead to dead locks
  - ▶ ... still seems the most popular solution around.
- ▶ Multiple domains can hurt performance (TCP slow start)
- ▶ Using dedicated CDN providers can help
  - ▶ Content locality



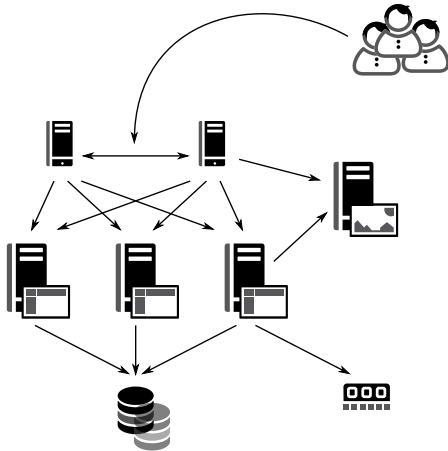
**Database servers  
too slow...**



**SLOW**

# Evolution

---





# Lessons Learned: Replicate Database

---

- ▶ Master Slave Replication is fairly easy to set up
  - ▶ Obviously only scales READs
  - ▶ WRITES are usually not your first problem



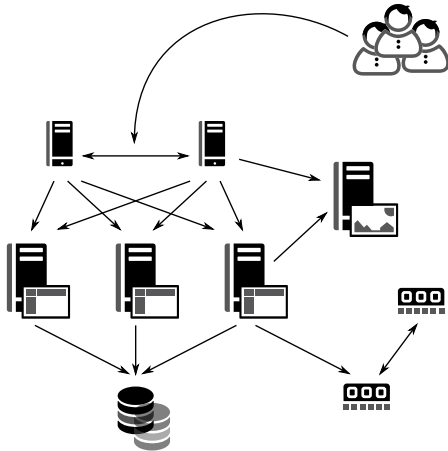
A museum gallery filled with ornate golden objects in glass display cases. The central focus is a large, multi-tiered golden chandelier or vase-like object on a white pedestal. Other cases contain various golden vessels, plates, and decorative items. The lighting is dramatic, highlighting the intricate details of the objects.

**Database servers  
too expensive...**



# Evolution

---



# Lessons Learned: Cache With Memcache

---

- ▶ Cache all the things in *memory*
  - ▶ Cache entities
  - ▶ Cache collections
  - ▶ Full page cache
- ▶ Cache invalidation

*There are three hard things in Computer Science:  
Cache invalidation and off by one errors.*

- ▶ Cache dependency calculation
- ▶ The paging problem



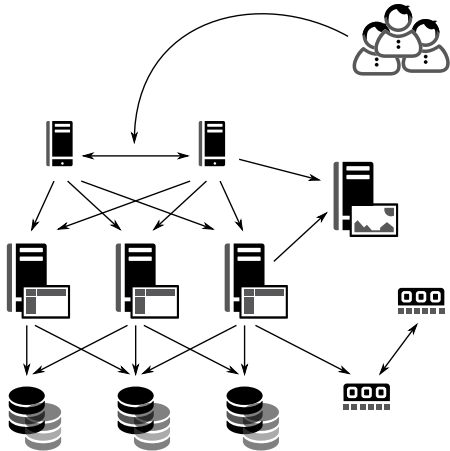
A person is shown from the side, typing on a typewriter. Their head is completely obscured by a large, messy pile of crumpled white paper. The scene is dimly lit, with a strong light source from the right illuminating the typewriter and the person's hands. The background is dark.

**Too many writes**



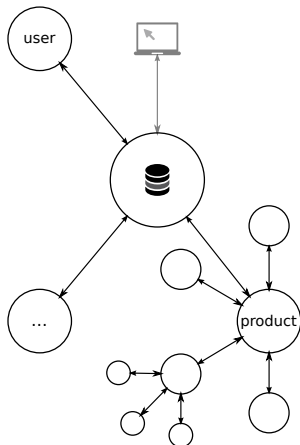
# Evolution

---



# Sharding

- ▶ Split tables across multiple nodes
  - ▶ Vertical sharding
- ▶ Shard by consistent hashing
  - ▶ Horizontal sharding



# Lessons Learned: Sharding

---

- ▶ Shard by table
  - ▶ ... or even shard by consistent hash per entity
- ▶ No referential integrity checking
- ▶ Queries are limited to sharding solution
- ▶ Schema updates across multiple shards are *fun*





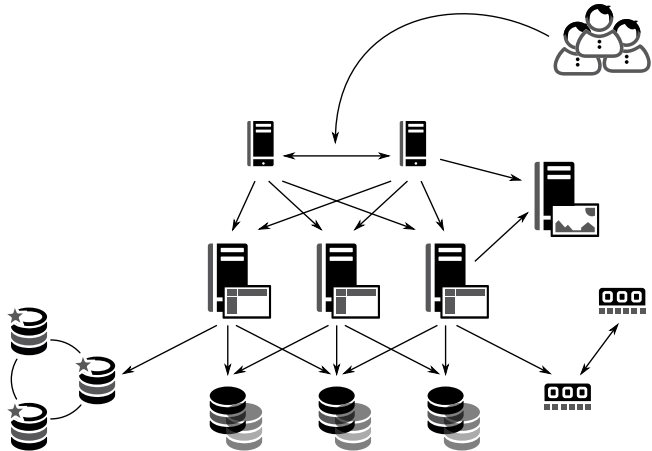


**Setup too complex**



# Evolution

---



# Lessons Learned: NoSQL

---

- ▶ Usually solves one problem really well:
  - ▶ Sharding
  - ▶ Multi-Master-Replication
  - ▶ Cross-shard queries
- ▶ Usually omits:
  - ▶ SQL
  - ▶ Referential Integrity
- ▶ ... we lost all relevant features from Relational Database Management Systems anyways...

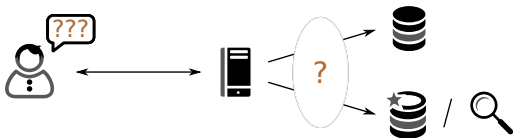


# Keeping data consistent across multiple nodes

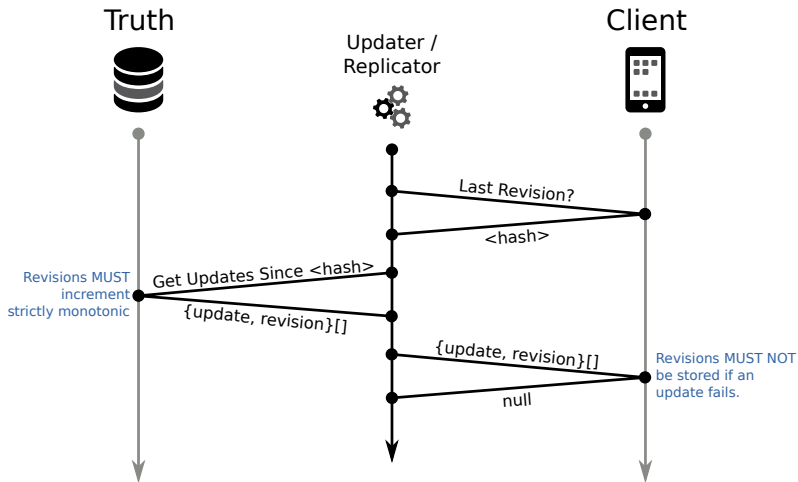


# Data Consistency Across Nodes

---



# Eventual Consistency



# Lessons Learned: Data Consistency

---

- ▶ Embrace Eventual Consistency
  - ▶ Compaction is hard
  - ▶ Data migrations are hard





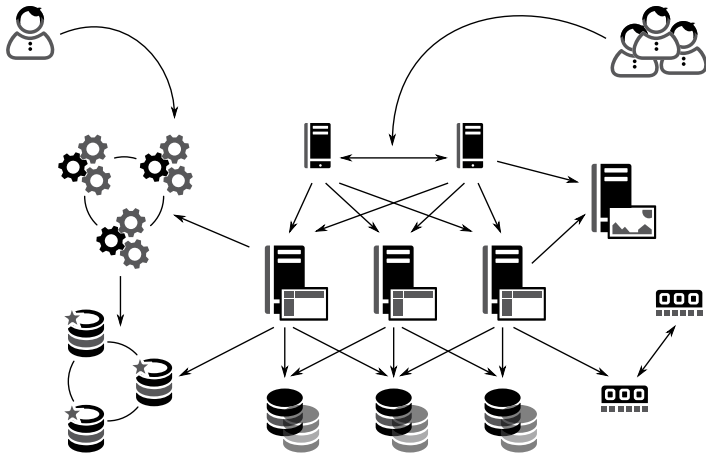
**Business wants to  
query data**





# Evolution

---



# Lessons Learned: Map-Reduce

---

- ▶ Execute queries on distributed databases
- ▶ New query language to learn
  - ▶ Your developers write analysis scripts, instead of the business analysts writing slow SQL queries

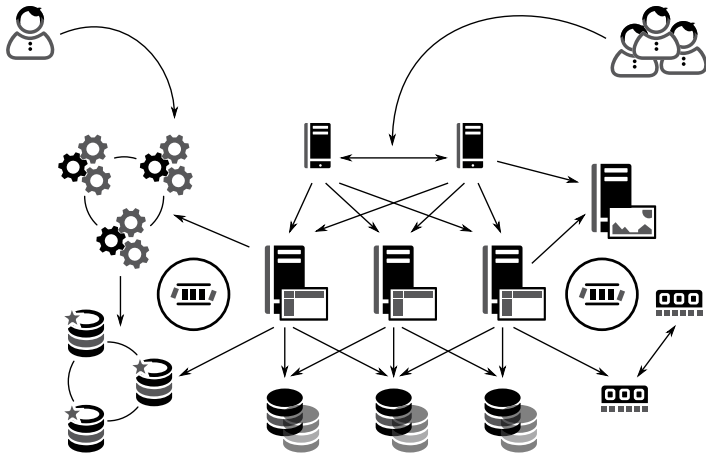


# How to orchestrate?



# Evolution

---



# Lessons Learned: Queues

---

- ▶ Queues can ensure data is processed asynchronously
  - ▶ Data consistency must be ensured even when pushing into queues
  - ▶ Following the data flow of an action can be “tricky”
- ▶ Used to distribute data between systems



# Evolution

---



# Microservices

---

Apply **Seperation of Concerns** on service level to allow for seperate teams & technologies per concern.

- ▶ Microservices **can** simplify things:
  - ▶ Choose optimal technology stack per team & concern
- ▶ Microservices **will** also complicate things:
  - ▶ Automated deployment is a must
  - ▶ Service orchestration is still a problem
  - ▶ Service downtimes and latency must be handled gracefully (Eventual Consistency)
- ▶ Big Data™ will stay a problem
- ▶ Sensible services are often not *micro* any more. . .

# Lessons Learned (subjective)

---

- ▶ Boring technology choices will often work best
  - ▶ Just start & stay with LAMP?
- ▶ Only bring in shiny new technologies with care
  - ▶ There are enough reasons to eventually do that, though





# The Hipster Says:

---



**DO NOT USE  
HIPSTER TECH!\***

\* Except you evaluated  
it as the correct solution  
for your case

# Conclusion

---

- ▶ There are many developers, documentation & experience for boring technologies
- ▶ Evaluate before adding new technologies (ATAM)
- ▶ Do not jump on every bandwagon – this includes microservices
- ▶ Data Consistency accross nodes is hard & important



<https://joind.in/talk/3152e>



THANK YOU

Rent a quality expert  
[qafoo.com](https://qafoo.com)