Google™ 09 IO

From Spark Plug to Drive Train: Life of an App Engine Request

Adam Byrtek
Uniwersytet Warszawski, 21 Oct 2010

Based on Google I/O 2009 slides by
Alon Levi

Google I/O 09

Agenda

- **Designing for Scale and Reliability**
- **App Engine: Design Motivations**
- **Life of a Request**

Request for static content

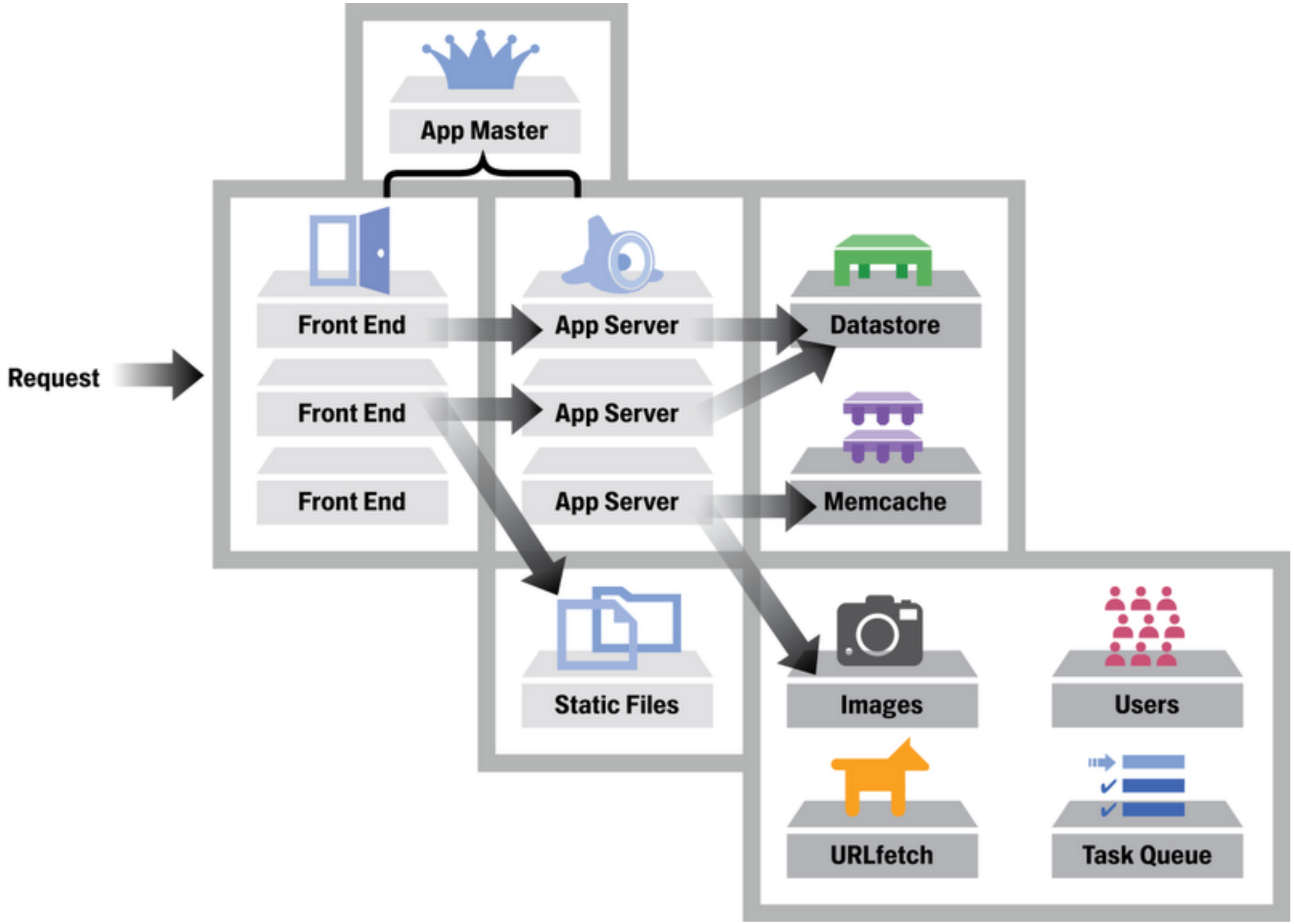
Request for dynamic content

Requests that use APIs

- **App Engine: Design Motivations, Recap**

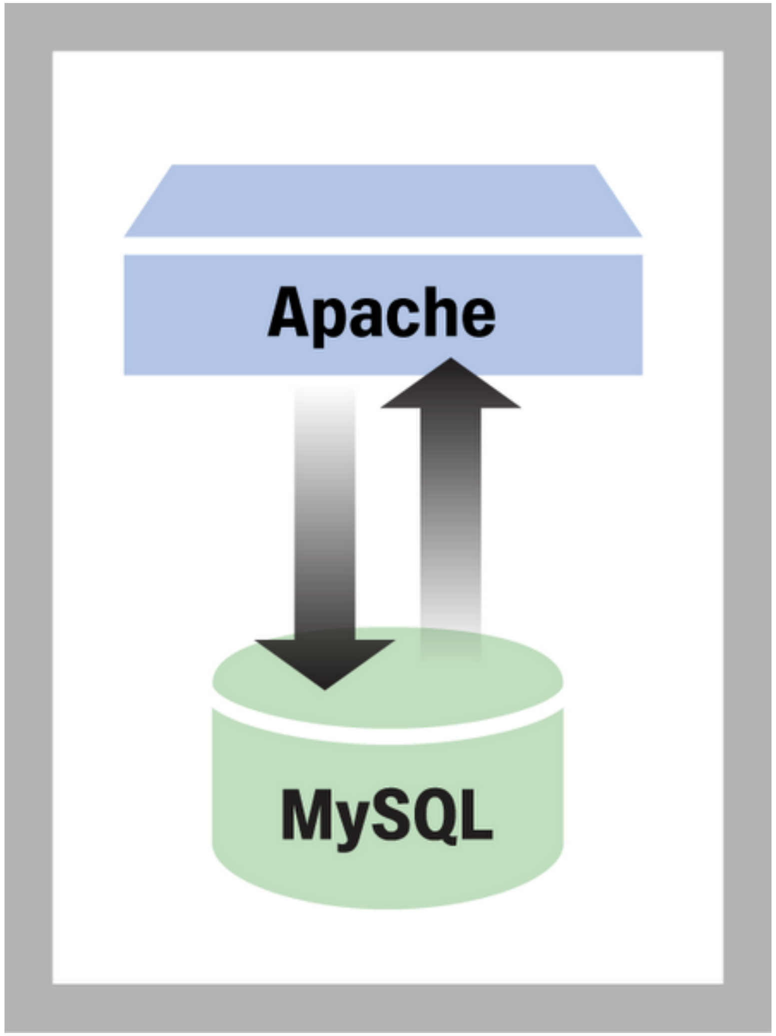
Designing for Scale and Reliability

Google App Engine



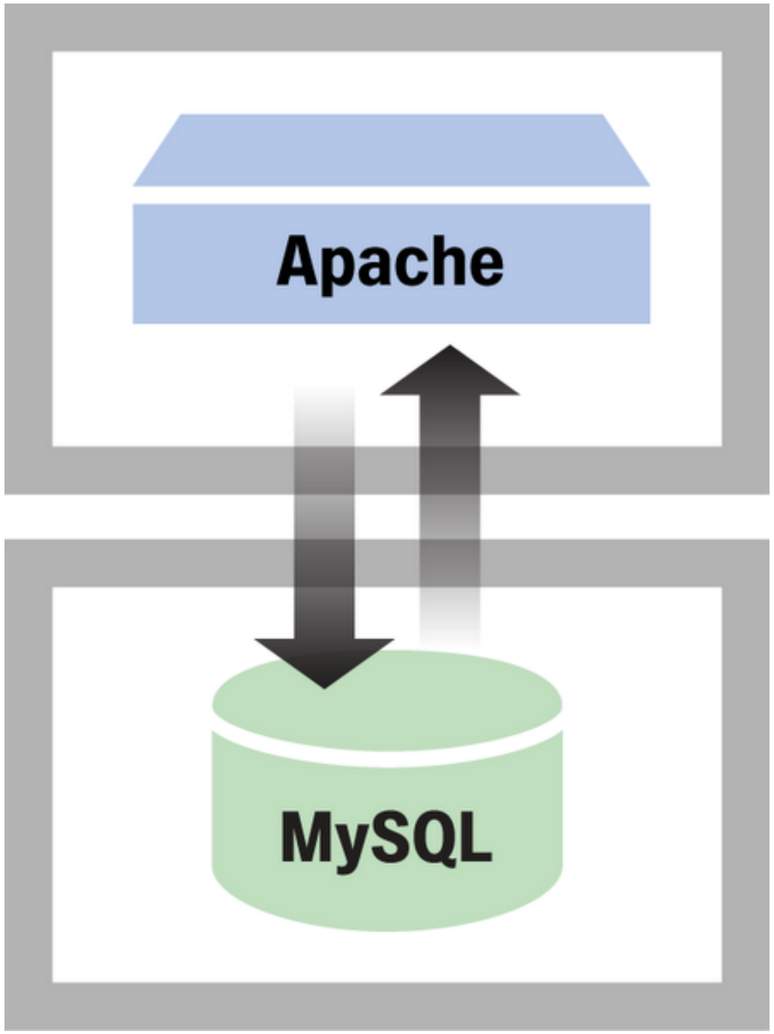
Basic LAMP

- Linux, Apache, MySQL, Programming Language
- **Scalable?**
  Shared machine for database and webserver
- **Reliable?**
  Single point of failure (SPOF)

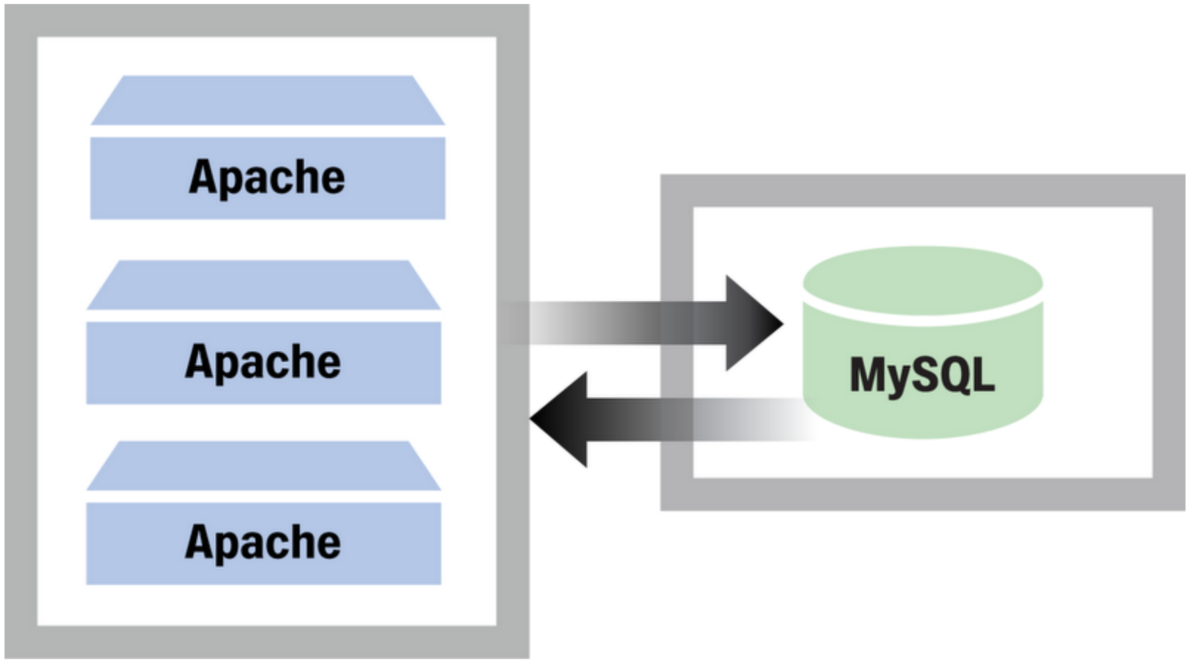


Dedicated Database

- Database running on a separate server
- **Requirements**
  Another machine plus additional management
- **Scalable?**
  Up to one web server
- **Reliable?**
  Two single points of failure



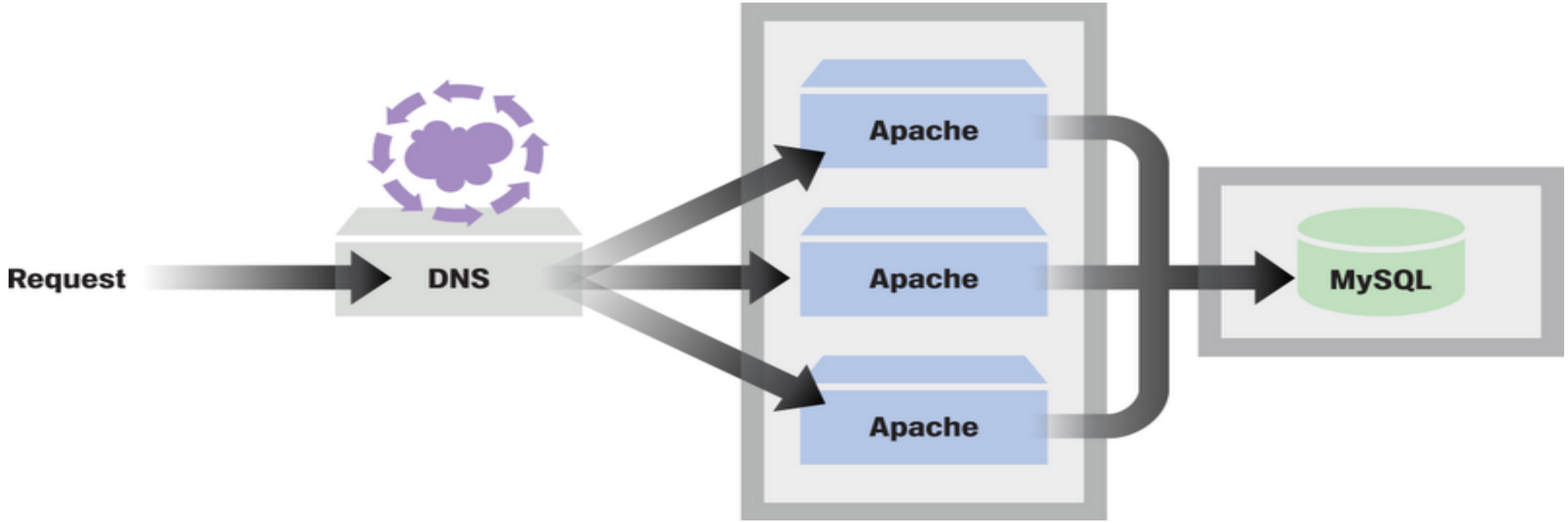
Multiple Web Servers



- **Benefits:**
  Grow traffic beyond the capacity of one webserver
- **Requirements:**
  More machines
  Set up load balancing

Multiple Web Servers

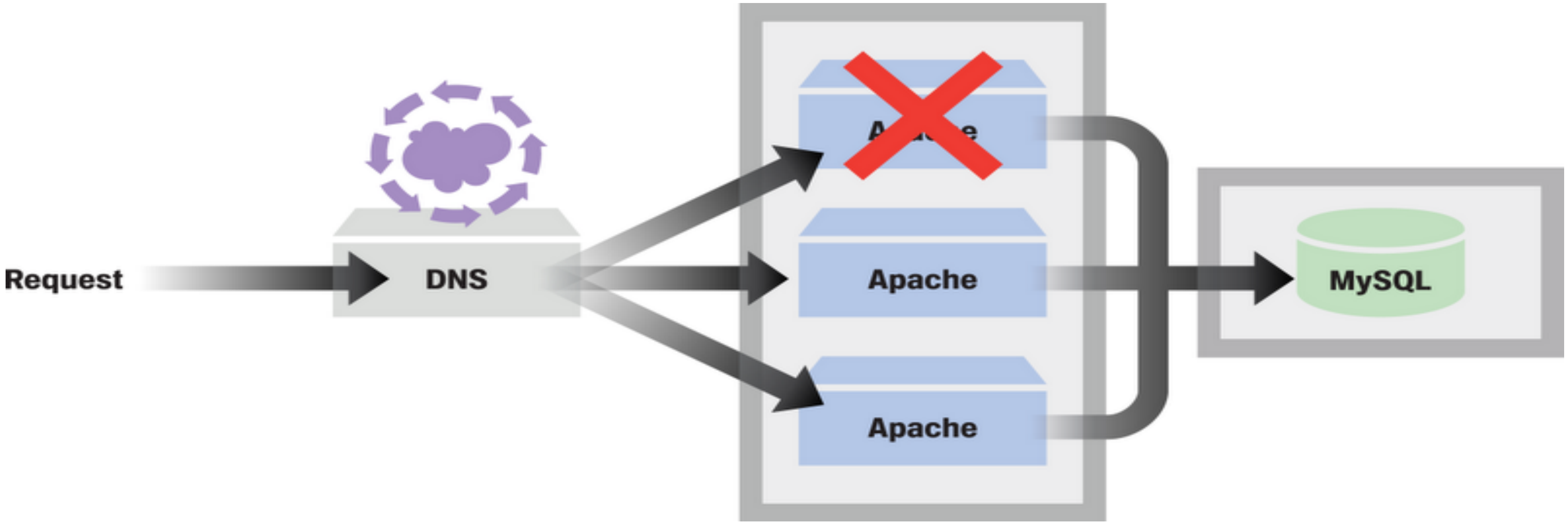
Load Balancing: DNS Round Robin



- Register list of IPs with DNS
- Statistical load balancing
- DNS record is cached with Time To Live (TTL)
  - TTL may not be respected

Multiple Web Servers

Load Balancing: DNS Round Robin

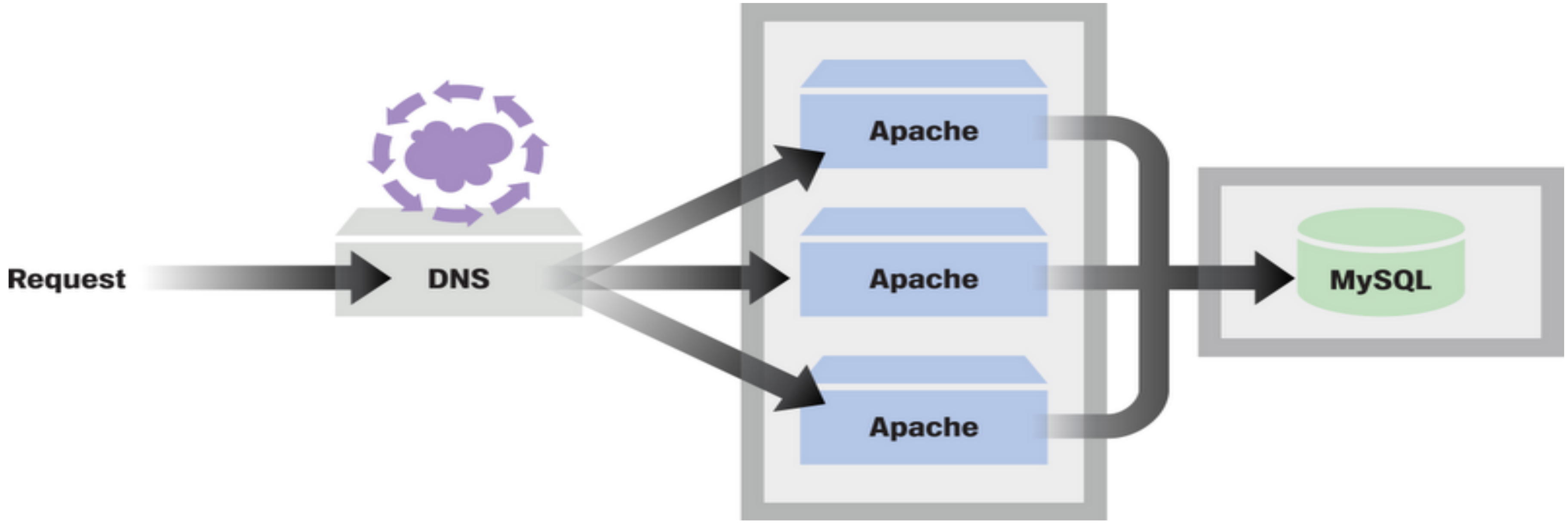


- Register list of IPs with DNS
- Statistical load balancing
- DNS record is cached with Time To Live (TTL)
    - TTL may not be respected

Now wait for DNS changes to propagate :-(

Multiple Web Servers

Load Balancing: DNS Round Robin



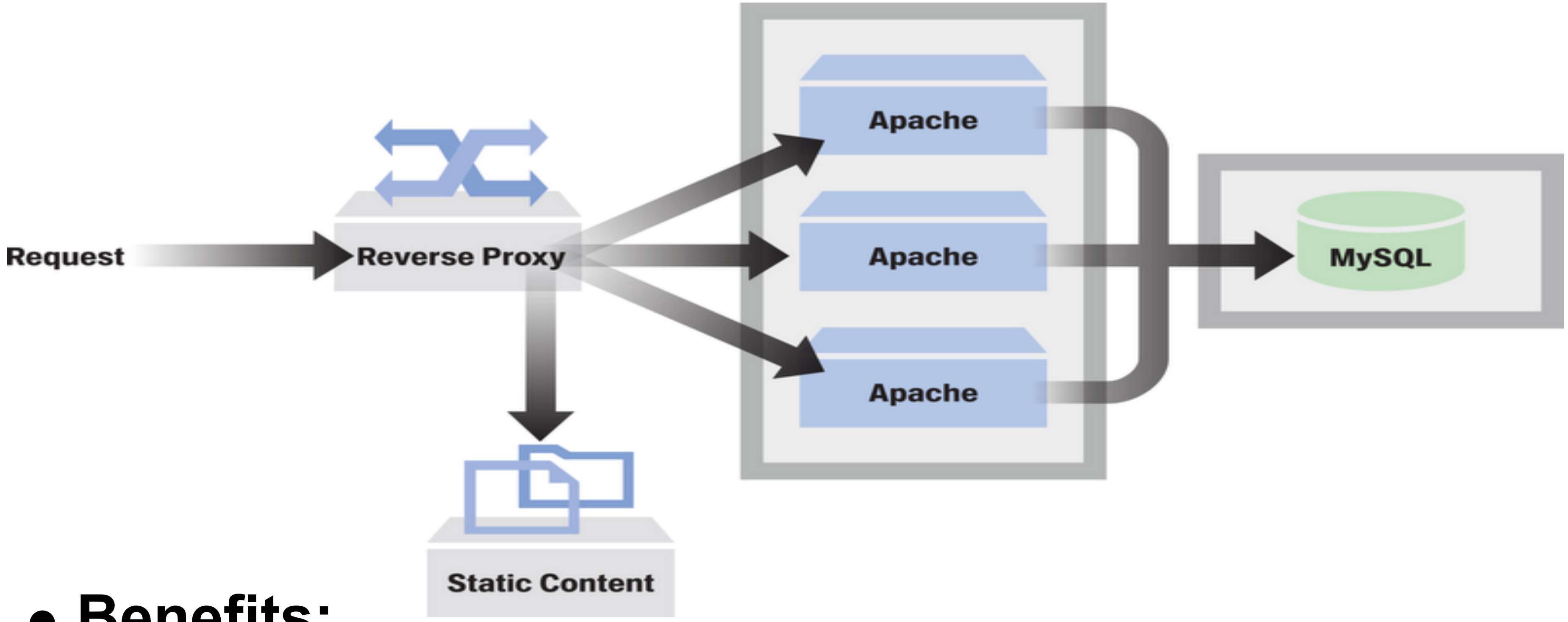
- **Scalable?**

Add more webservers as necessary
Still I/O bound on one database

- **Reliable?**

Cannot redirect traffic quickly
Database still SPOF

Reverse Proxy



- **Benefits:**

Custom Routing

  - Specialization
  - Application-level load balancing

- **Requirements:**

More machines

Configuration and code for reverse proxies

Reverse Proxy

- **Scalable?**

Add more web servers
Specialization
Bound by

  - Routing capacity of reverse proxy
  - One database server

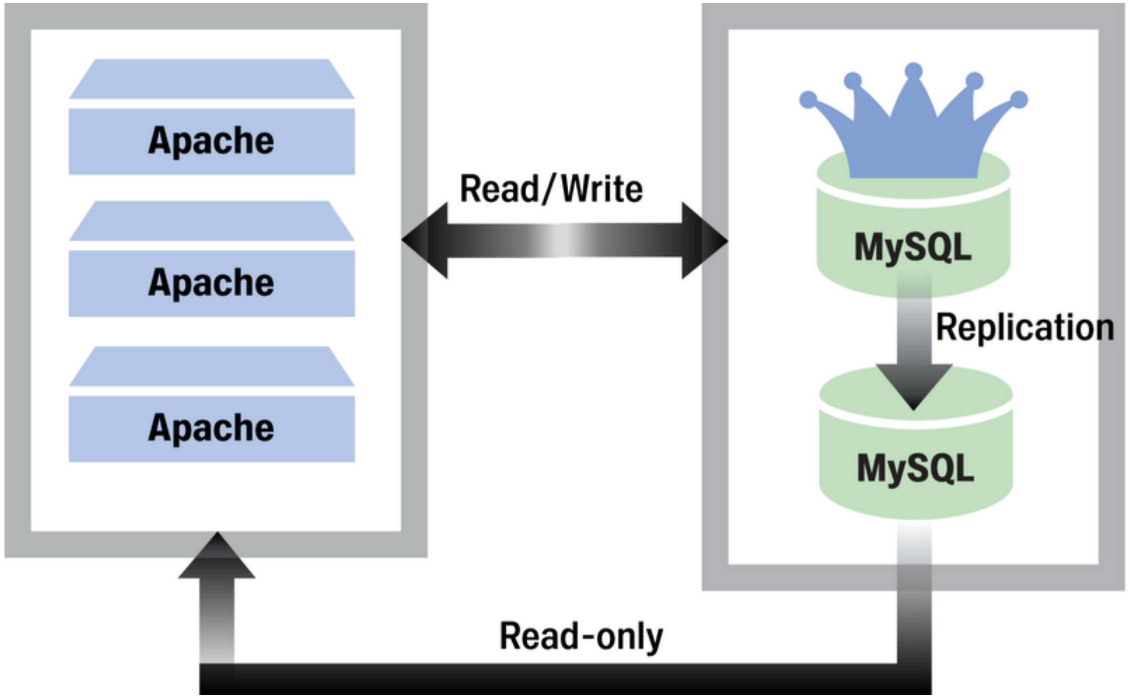
- **Reliable?**

Agile application-level routing
Specialized components are more robust
Multiple reverse proxies requires network-level routing

  - DNS Round Robin (again)
  - Fancy network routing hardware

Database is still SPOF

Master-Slave Database



- **Benefits:**

Better read throughput
Invisible to application

- **Requirements:**

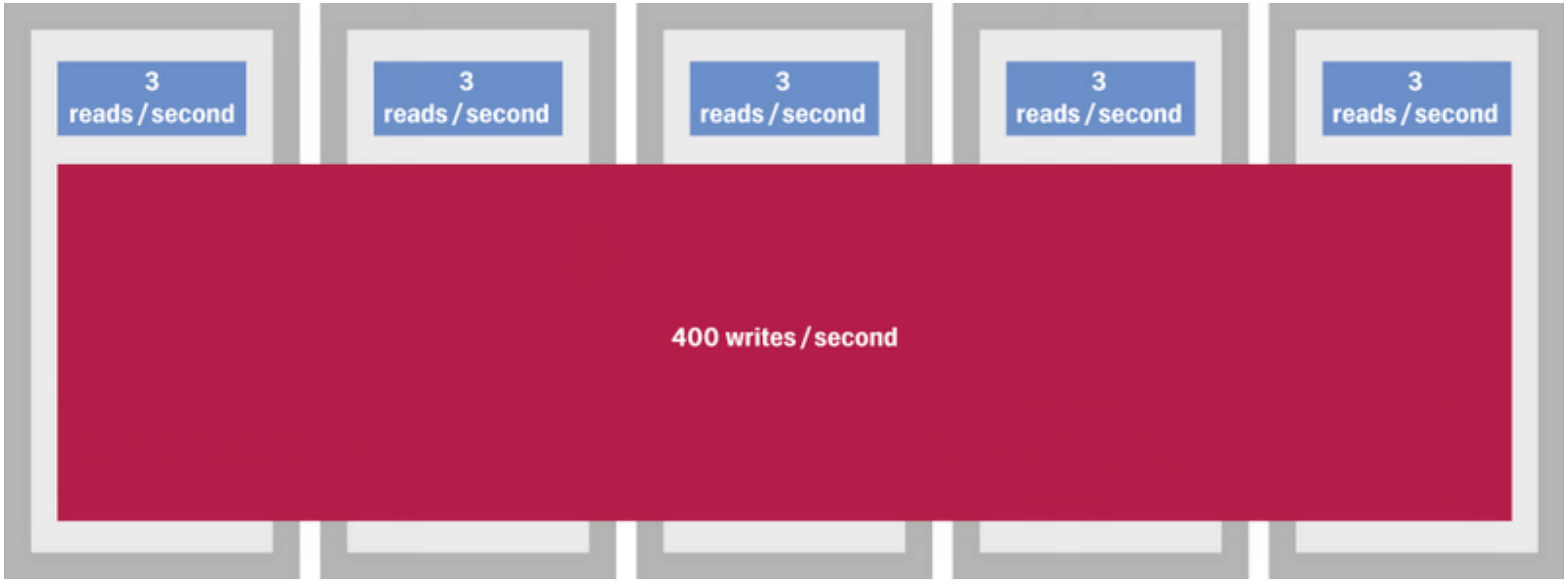
Even more machines
Changes to MySQL

Master-Slave Database

- **Scalable?**

Scales read rate with # of servers

  - But not writes



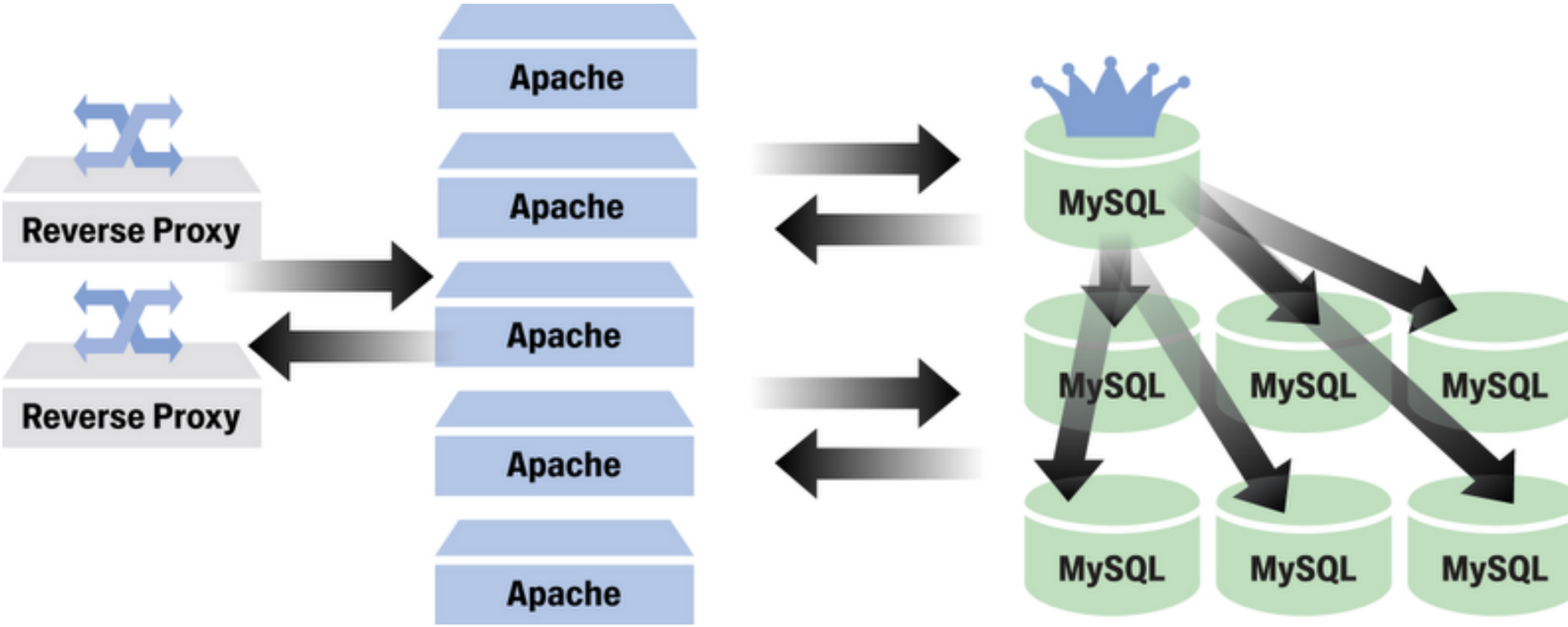
What happens eventually?

Master-Slave Database

- **Reliable?**

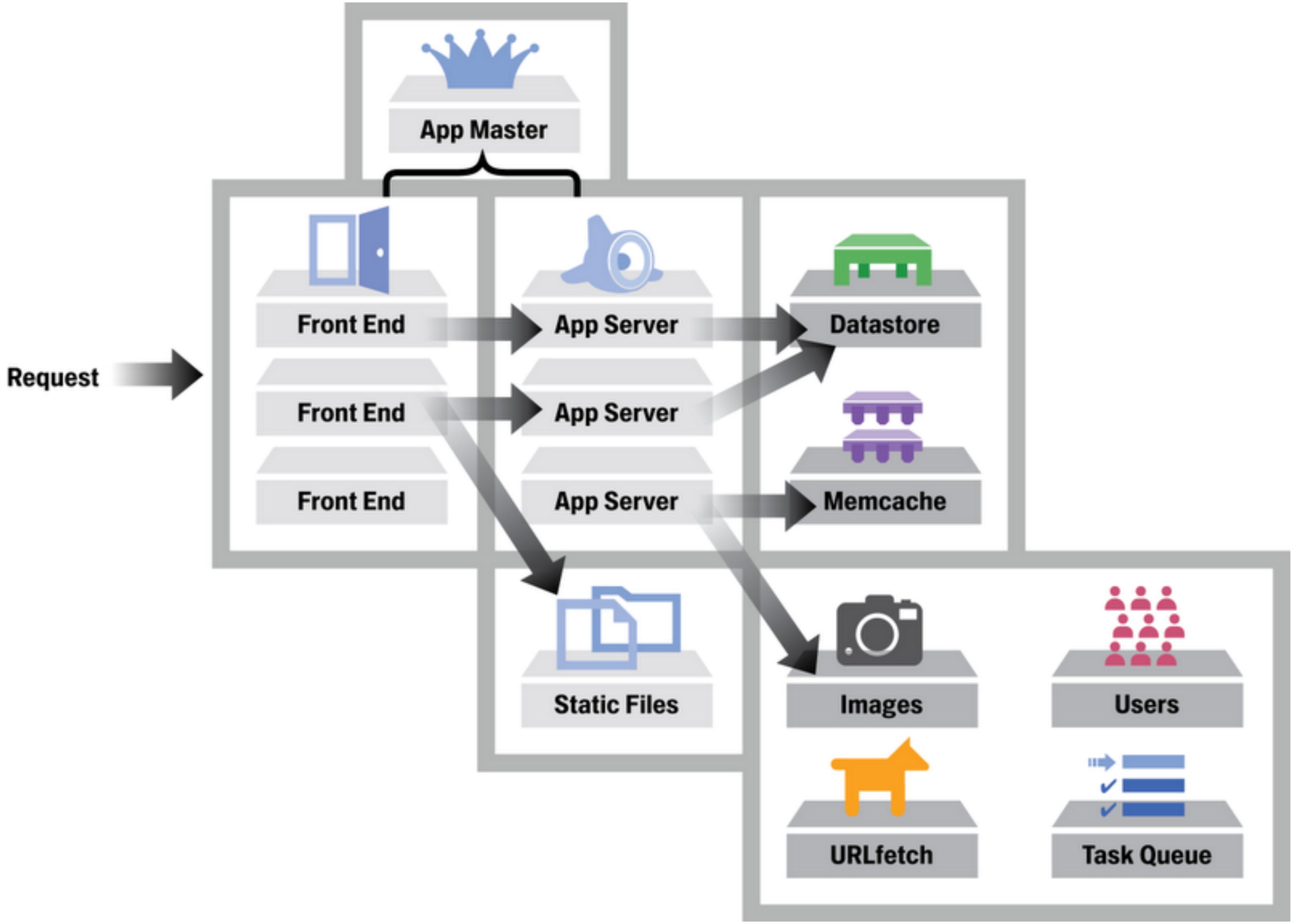
Master is SPOF for writes
Master may die before replication

Partitioned Database



- **Benefits:**
  Increase in both read and write throughput
- **Requirements:**
  Even more machines
  Lots of management
  Re-architect data model
  Rewrite queries

The App Engine Stack



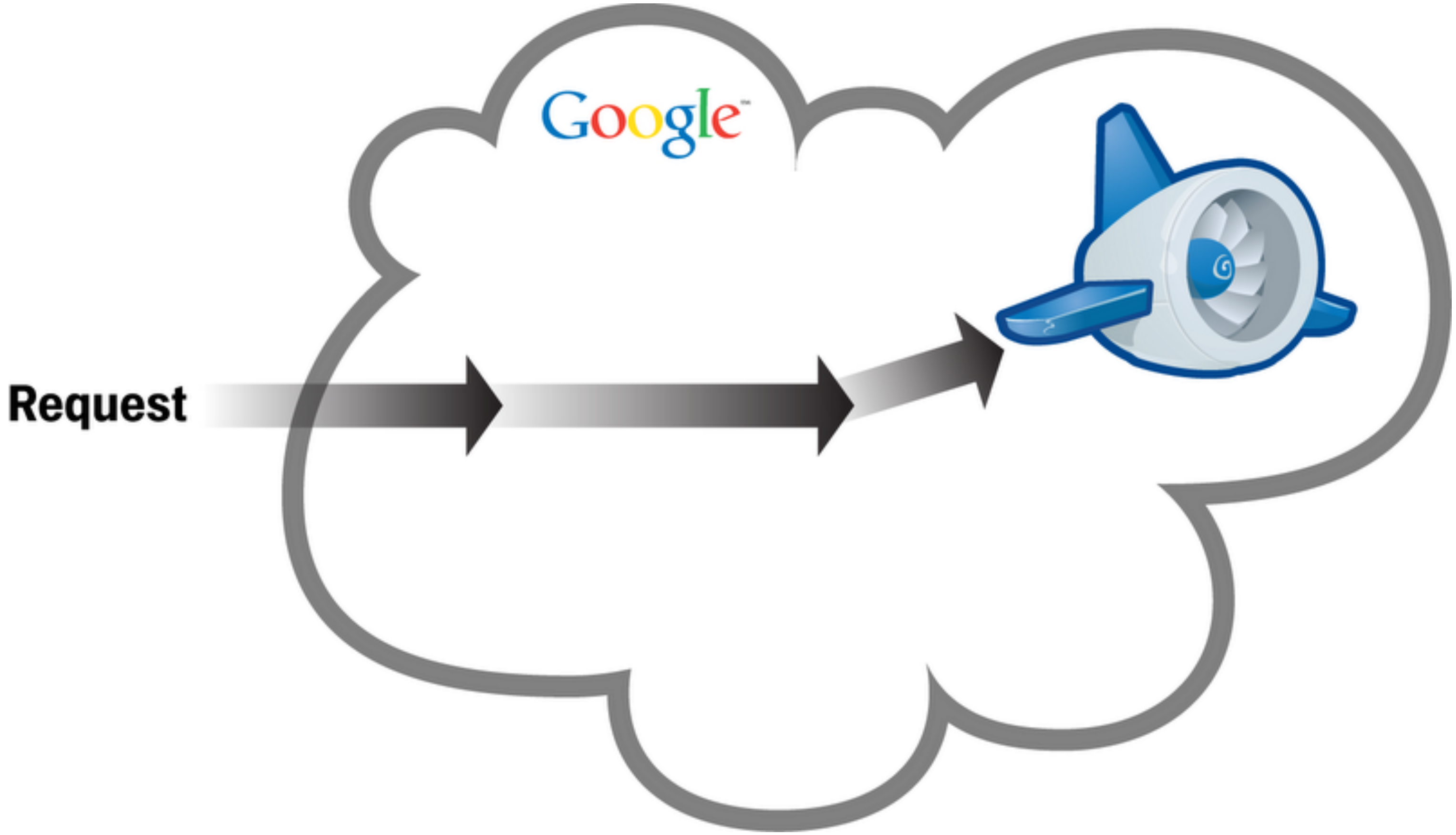
App Engine:
Design Motivations

Design Motivations

- **Build on Existing Google Technology**
- **Provide an Integrated Environment**
- **Encourage Small Per-Request Footprints**
- **Encourage Fast Requests**
- **Maintain Isolation Between Applications**
- **Encourage Statelessness and Specialization**
- **Require Partitioned Data Model**

Life of a Request:
Request for Static Content

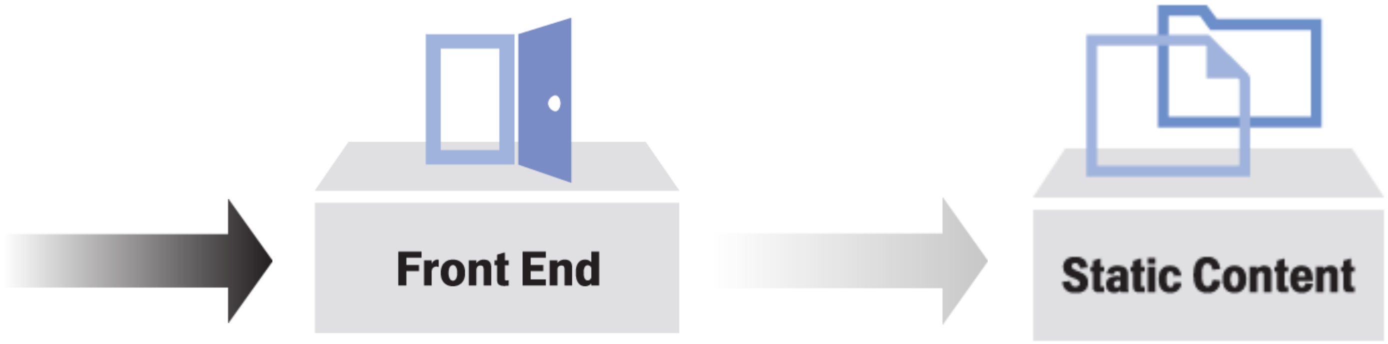
Request for Static Content on Google Network



- Routed to the nearest Google datacenter
- Travels over Google's network
    - Same infrastructure other Google products use
    - Lots of advantages for free

Request for Static Content

Routing at the Front End



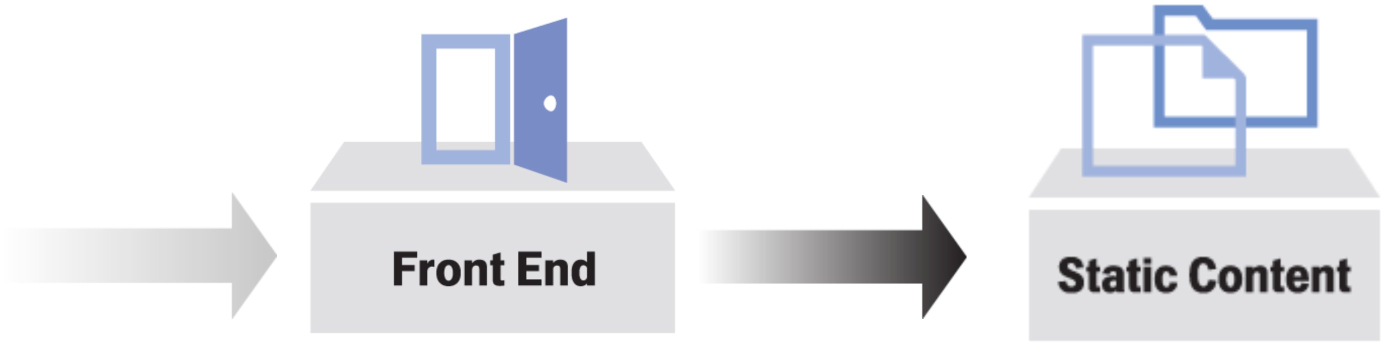
- **Google App Engine Front Ends**

Load balancing
Routing

- Frontends route static requests to specialized serving infrastructure

Request for Static Content

Static Content Servers



- **Google Static Content Serving**

Built on shared Google Infrastructure

- Static files are physically separate from code files
- How are static files defined?

Request for Static Content

What content is static?

Java Runtime: appengine-web.xml

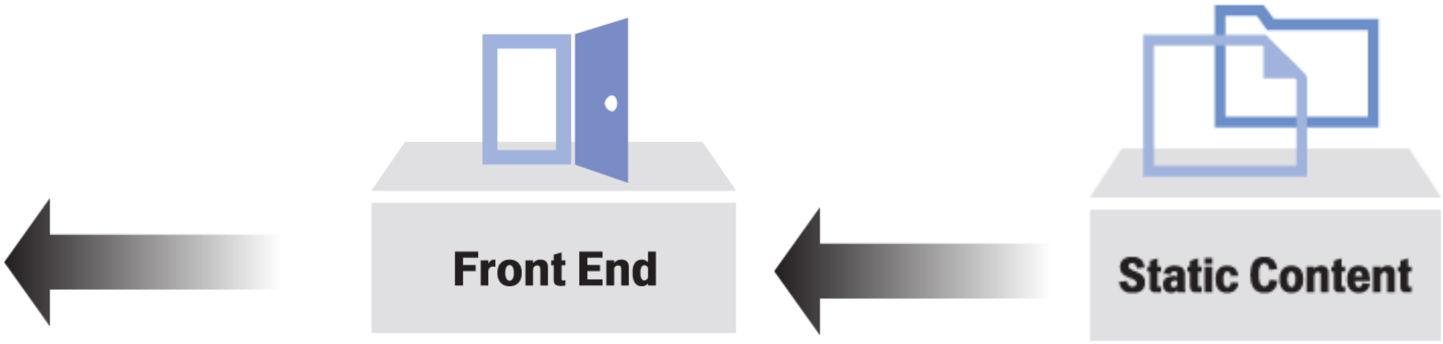
```
...
<static>
  <include path="/**.png" />
  <exclude path="/data/**.png />
</static>
...
```

Python Runtime: app.yaml

```
...
- url: /images
static_dir: static/images
OR
- url: /images/(.*)
static_files: static/images/\1
upload: static/images/(.*)
...
```

Request For Static Content

Response to the user



- Back to the Front End and out to the user
- Specialized infrastructure
    - App runtimes don't serve static content

Life of a Request:
Request for Dynamic Content

Request for Dynamic Content: New Components

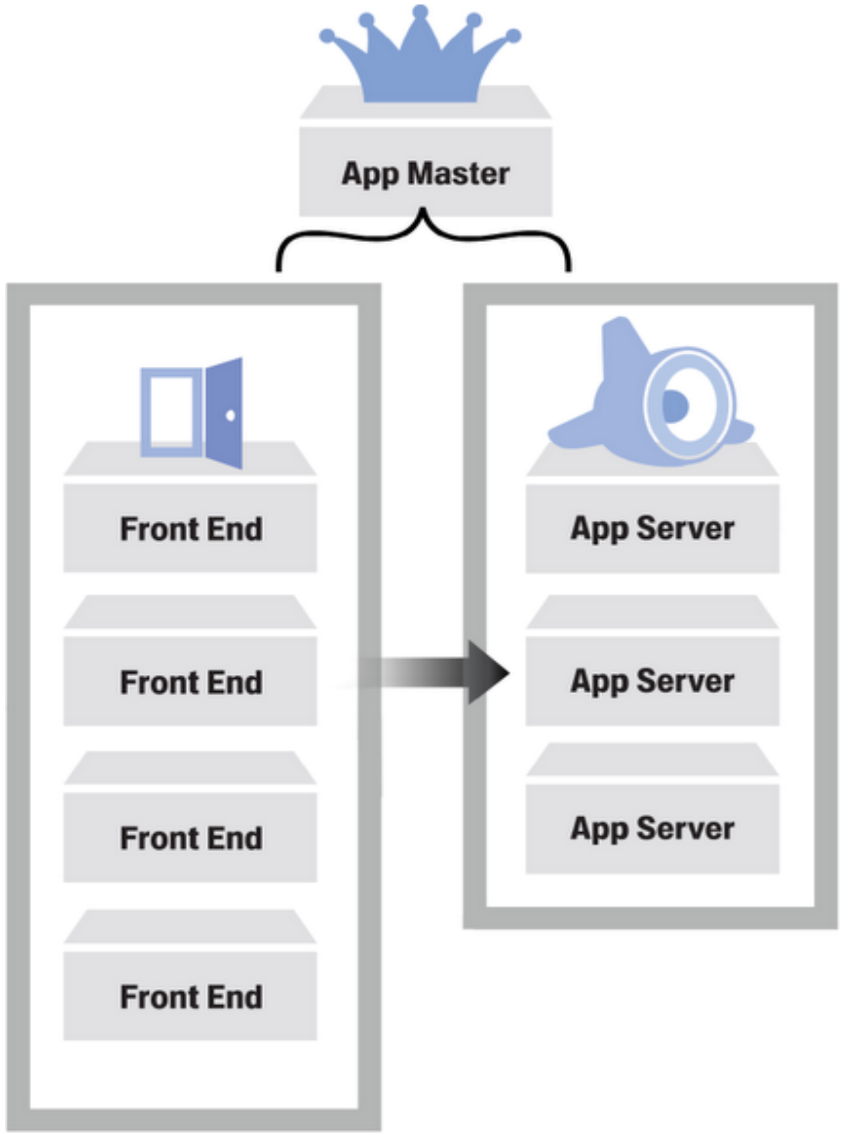
App Servers and App Master

- **App Servers**

Serve dynamic requests
Where your code runs

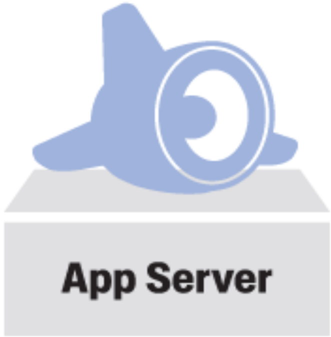
- **App Master**

Schedules applications
Informs Front Ends



Request for Dynamic Content: Appservers

What do they do?

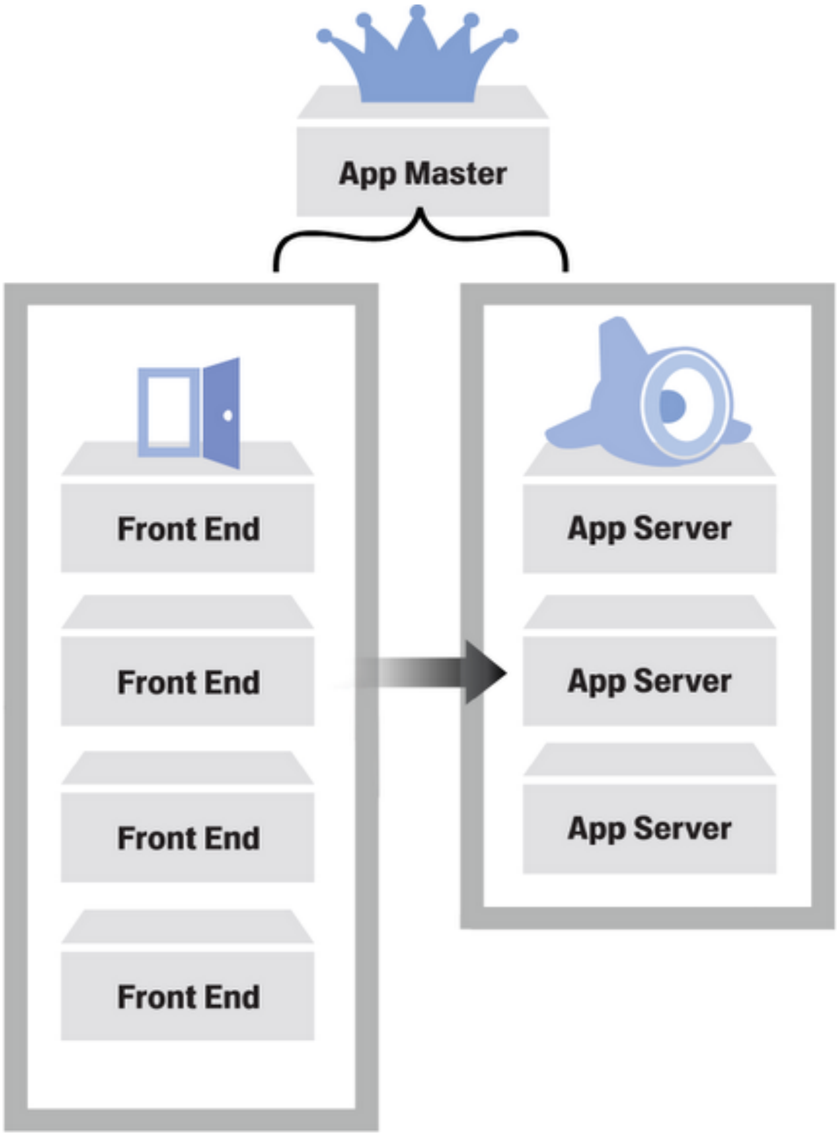


- Many applications
- Many concurrent requests
    - Smaller app footprint + fast requests = more apps
- Enforce Isolation
    - Keeps apps safe from each other
- Enforce statelessness
    - Allows for scheduling flexibility
- Service API requests

Request For Dynamic Content

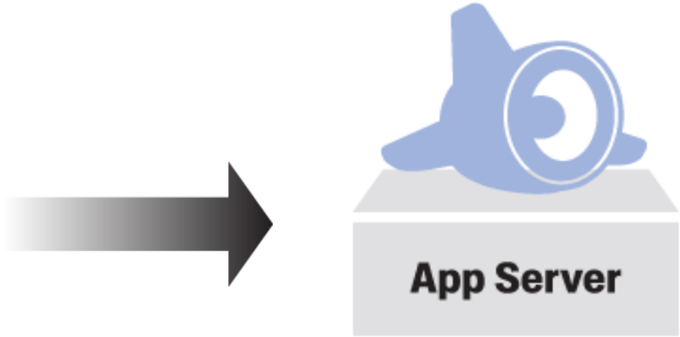
Routing at the Frontend

- Front Ends route dynamic requests to App Servers



Request for Dynamic Content

App Server



1. Checks for cached runtime
   - If it exists, no initialization
2. Execute request
3. Cache the runtime
   - System is designed to maximize caching

- Slow first request, faster subsequent requests
- Optimistically cache data in your runtime!

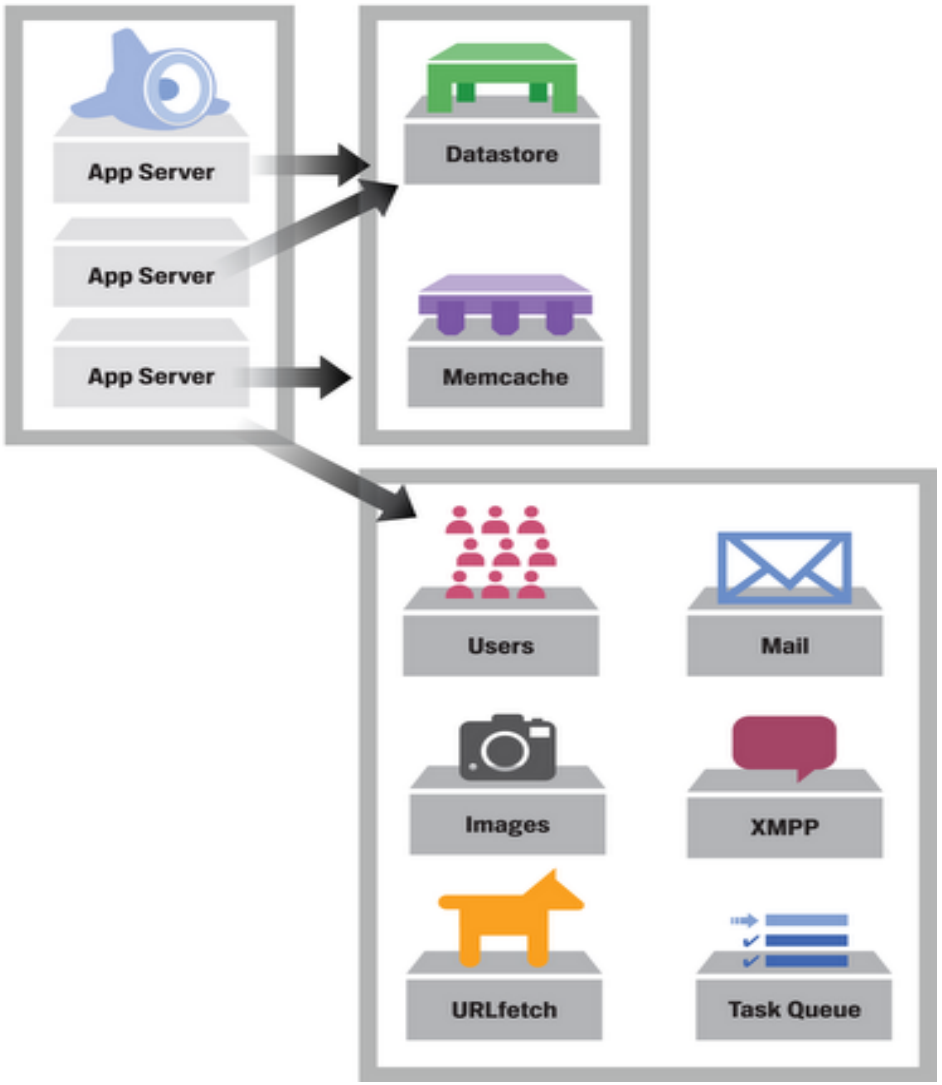
Life of a Request: Requests accessing APIs

API Requests

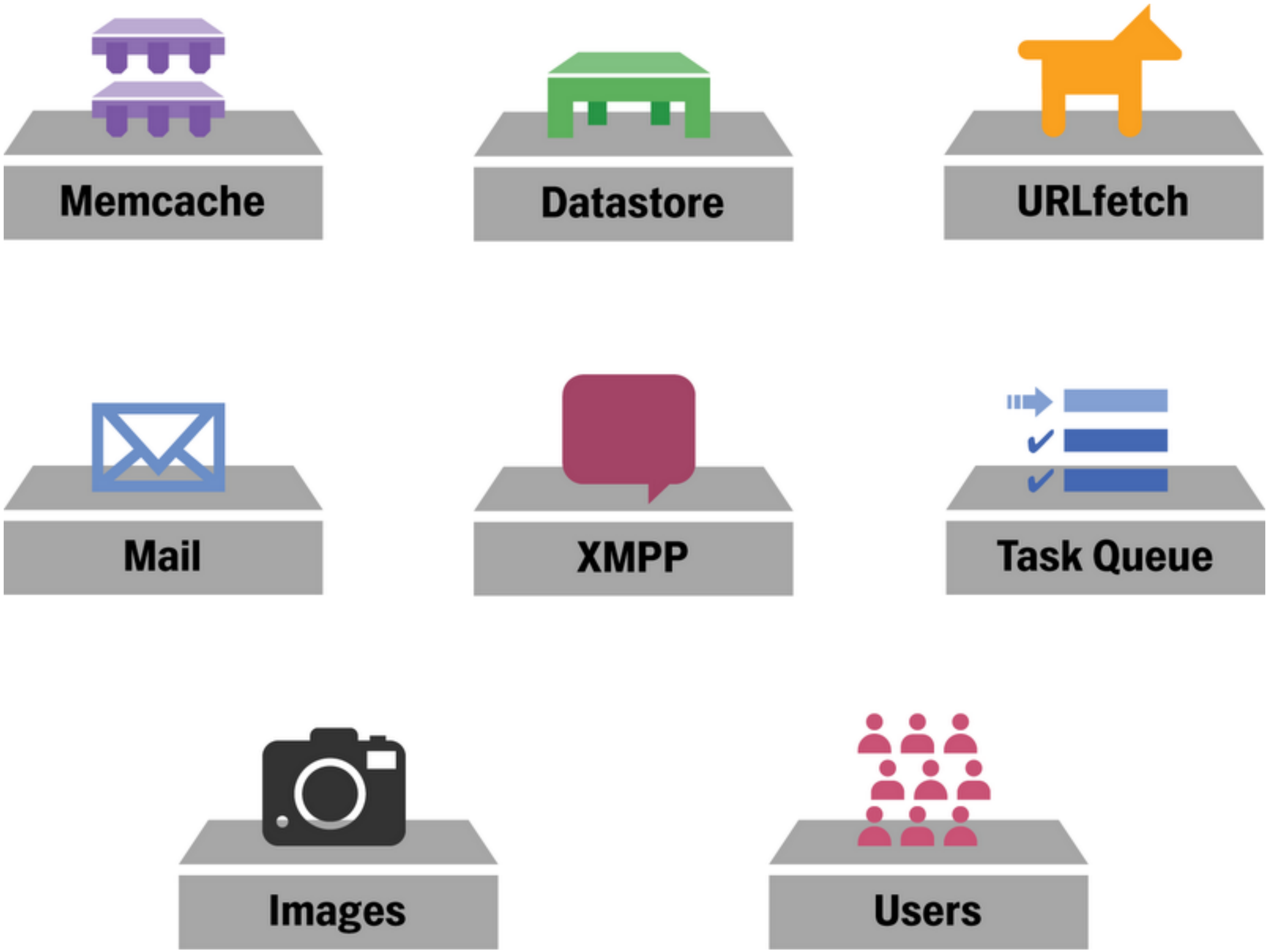
App Server

1. App issues API call
2. App Server accepts
3. App Server blocks runtime
4. App Server issues call
5. Returns the response

- Use APIs to do things you don't want to do in your runtime, such as...

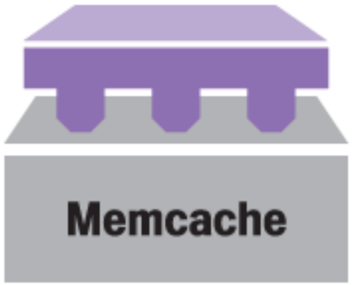


APIs


Memcache
Datastore
URLfetch
Mail
XMPP
Task Queue
Images
Users

Memcacheg

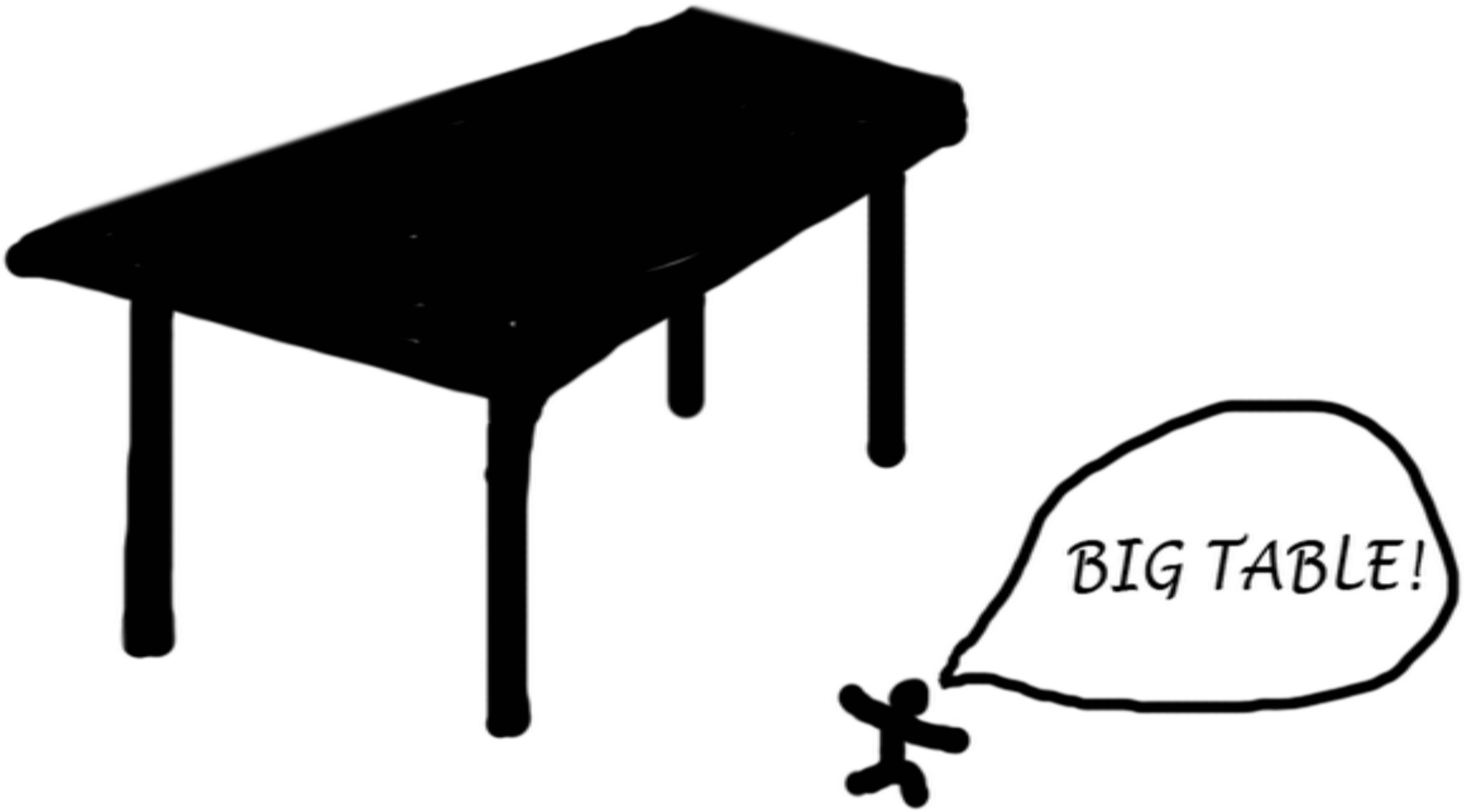
A more persistent in-memory cache



- Distributed in-memory cache
- memcacheg
  - Also written by Brad Fitzpatrick
  - adds: set_multi, get_multi, add_multi
- Optimistically cache for optimization
- Very stable, robust and specialized

The App Engine Datastore

Persistent storage



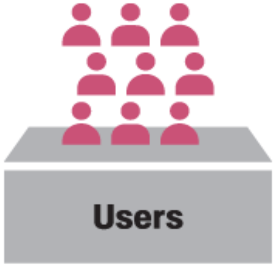
The App Engine Datastore

Persistent storage

- Your data is already partitioned
  - Use Entity Groups
- Explicit Indexes make for fast reads
  - But slower writes
- Replicated and fault tolerant
  - On commit: $\geq$3 machines
  - Geographically distributed
- Bonus: Keep globally unique IDs for free

Other APIs

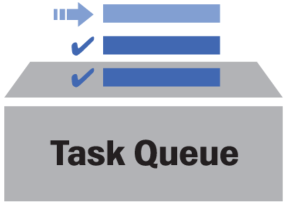
- GMail



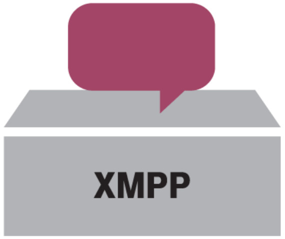
- Google Accounts

- Picasaweb

- Gadget API



- On roadmap



- Google Talk

App Engine:
Design Motivations, Recap

Provide an Integrated Environment

- **Why?**

Manage all apps together

- **What it means for you:**

Follow best practices
Some restrictions
Use our tools

- **Benefits:**

Use our tools
Admin Console
All of your logs in one place
No machines to configure or manage
Easy deployment

Encourage Small Per-Request Footprints

- **Why?**

Better utilization of App Servers
Fairness

- **What it means for your app:**

Less Memory Usage
Limited CPU

- **Benefits:**

Better use of resources

Encourage Fast Requests

- **Why?**

Better utilization of appservers
Fairness between applications
Routing and scheduling agility

- **What it means for your app:**

Runtime caching
Request deadlines

- **Benefits:**

Optimistically share state between requests
Better throughput
Fault tolerance
Better use of resources

Maintain Isolation Between Apps

- **Why?**

Safety
Predictability

- **What it means for your app:**

Certain system calls unavailable

- **Benefits:**

Security
Performance

Encourage Statelessness and Specialization

- **Why?**

App Server performance
Load balancing
Fault tolerance

- **What this means for you app:**

Use API calls

- **Benefits:**

Automatic load balancing
Fault tolerance
Less code for you to write
Better use of resources

Require Partitioned Data Model

- **Why?**

The Datastore

- **What this means for your app:**

Data model + Indexes

Reads are fast, writes are slower

- **Benefits:**

Design your schema once

  - No need to re-architect for scalability

More efficient use of cpu and memory

Questions?

Google™ 09 IO