How Slack Works

Keith Adams

kma@slack-corp.com @keithmadams facebook.com/kma

What is

Slack?

What is

Slack?

Voice Calls!

Platform!

Something about Bots!!

But first it was a

Persistent Group Messaging

Service

In this talk

- How Slack works today
    - ➞ Application logic
    - ➞ Persistence
    - ➞ Real-time messaging
    - ➞ Deferring work for later
- Problems
- What we’re doing about them

Also in this talk

- Flaws
- Challenges
- Mistakes
- Dead-ends
- Future directions

Slack Scale

- **4M DAU, 5.8M WAU**
  Peak simultaneous connected: 2.5M

- **> 2H / weekday for each active user**
  > 10H / weekday connected

- **Half of DAU outside US**

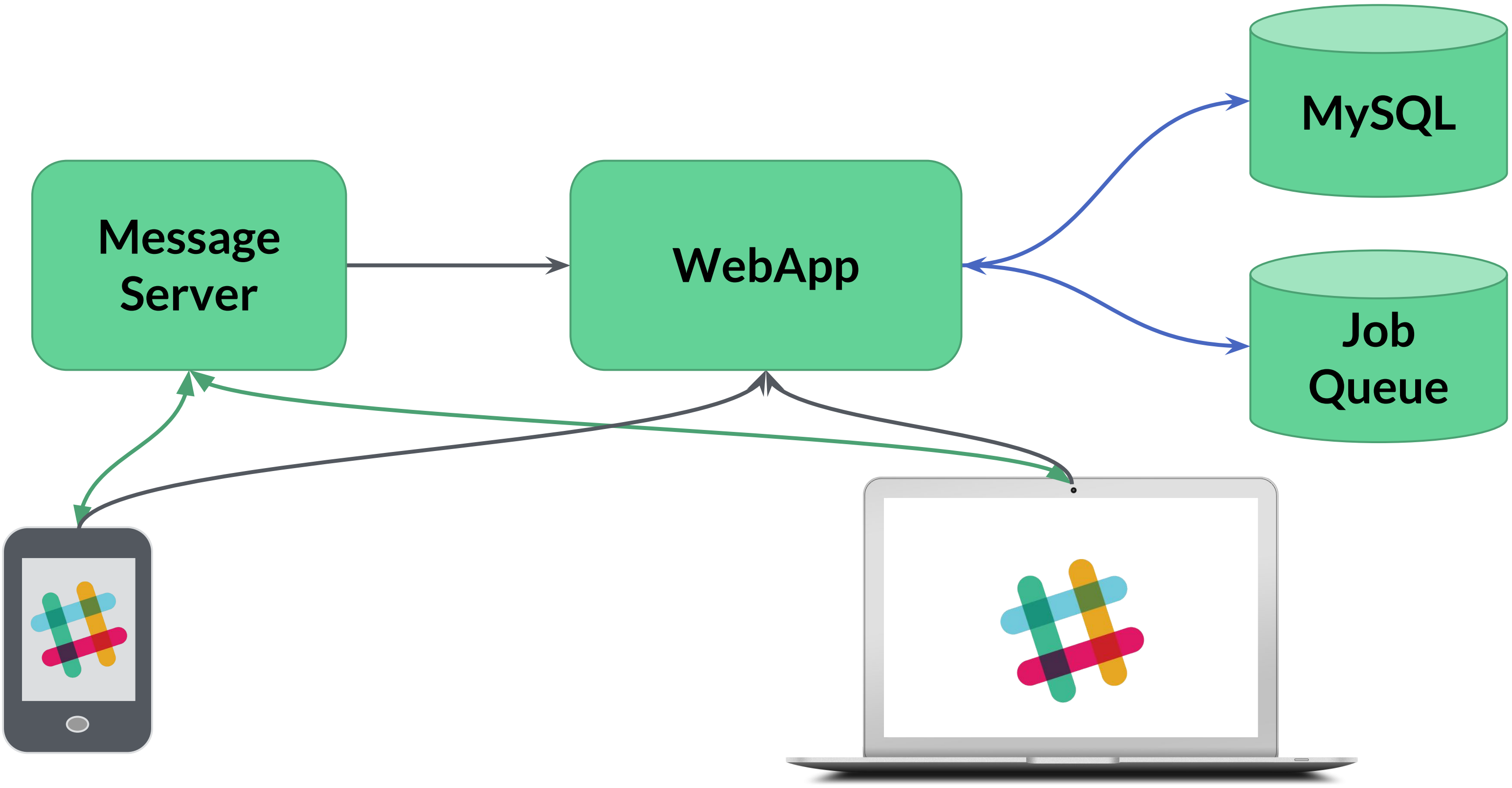
Slack House Style

- **Conservative technical taste**
  Most supporting technologies are >10 years old

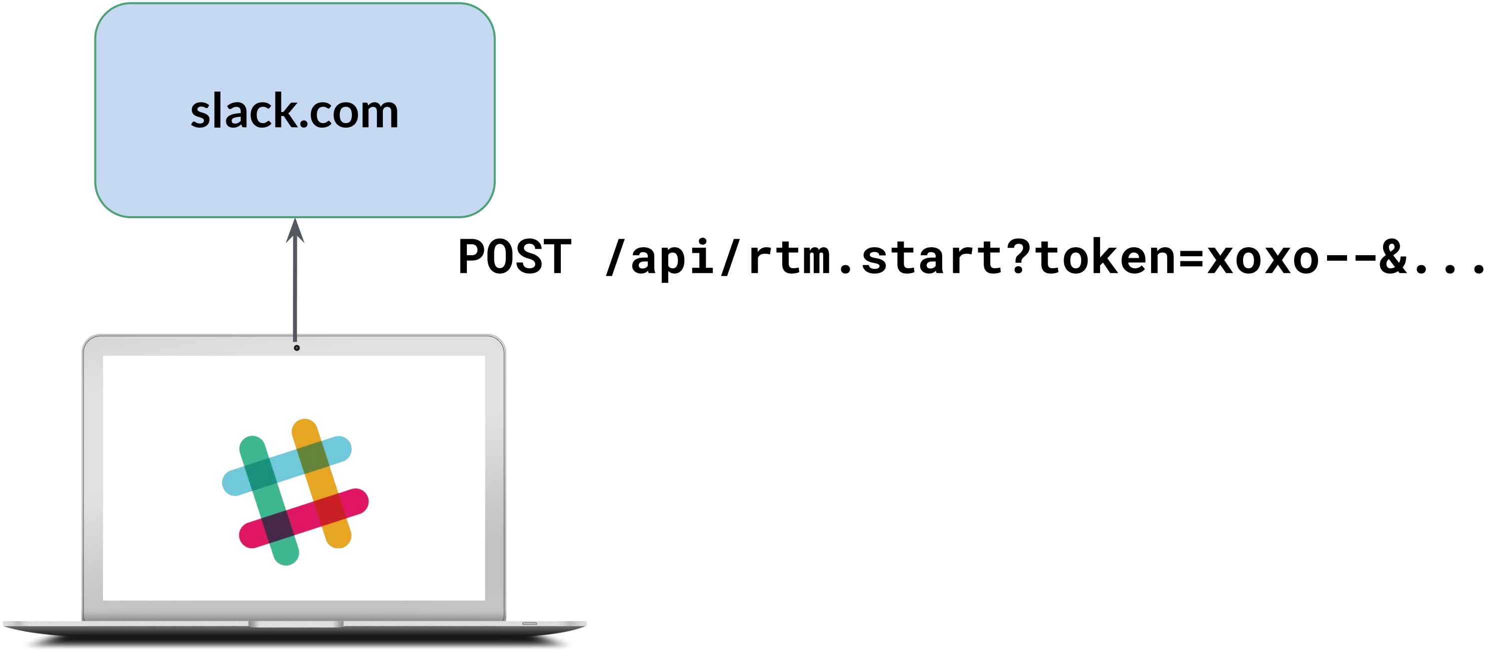
- **Willing to write a little code**
  Choose low coupling, fitness-to-purpose over DRY

- **Minimalism**
  Choose something we already operate over something new and tailor-made
  Shallow, transparent stack of abstractions

Cartoon Architecture of Slack



Case Study: Login and Receive Messages



Slack’s webapp codebase

- **PHP monolith of app logic**
  <1MLoC

- **Scaled-out LAMP stack app**
  Memcache wrapped around sharded MySQL

- **Recently migrated to HHVM**
  Performance, hacklang

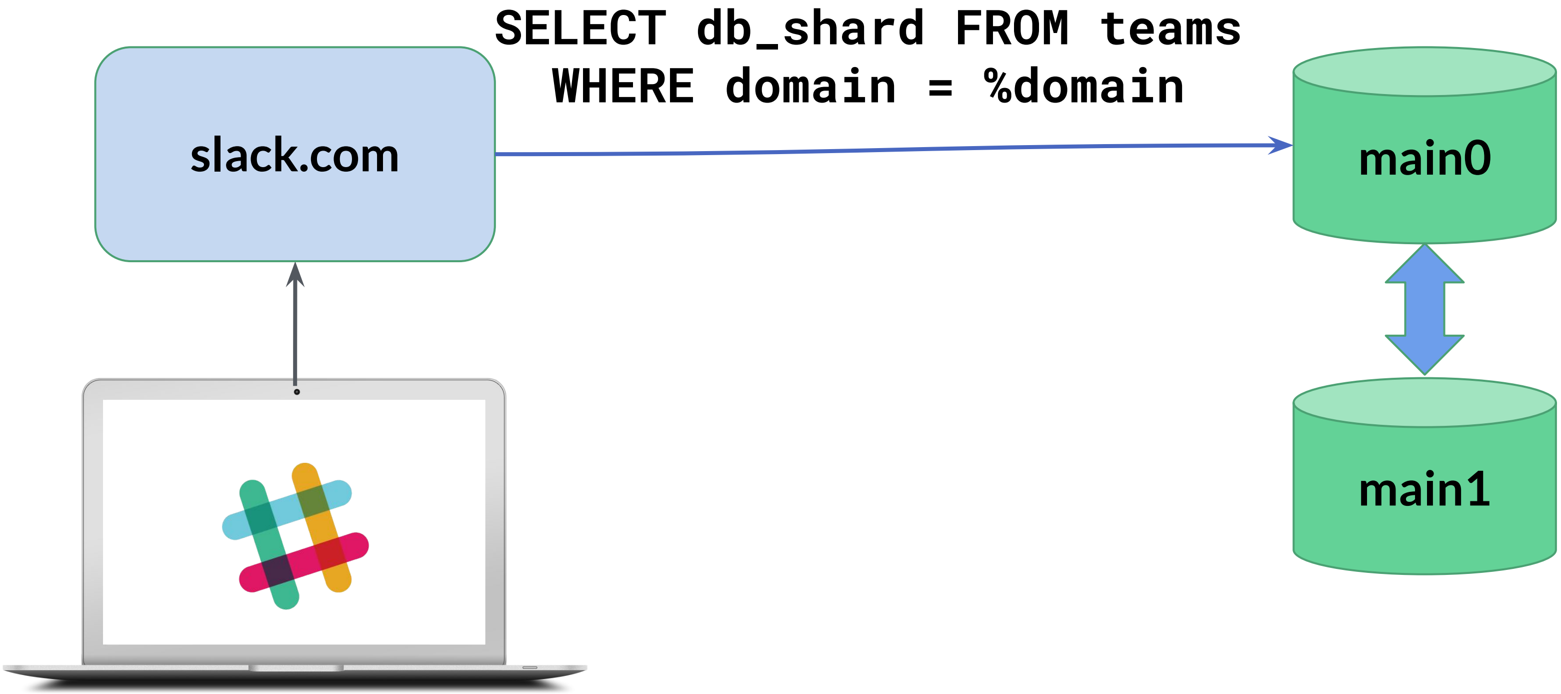
World’s shortest PHP-at-Slack FAQ

- **Q**: I hear/believe/have experienced PHP to be terrible.
  A: It sort of is, but it also works well.

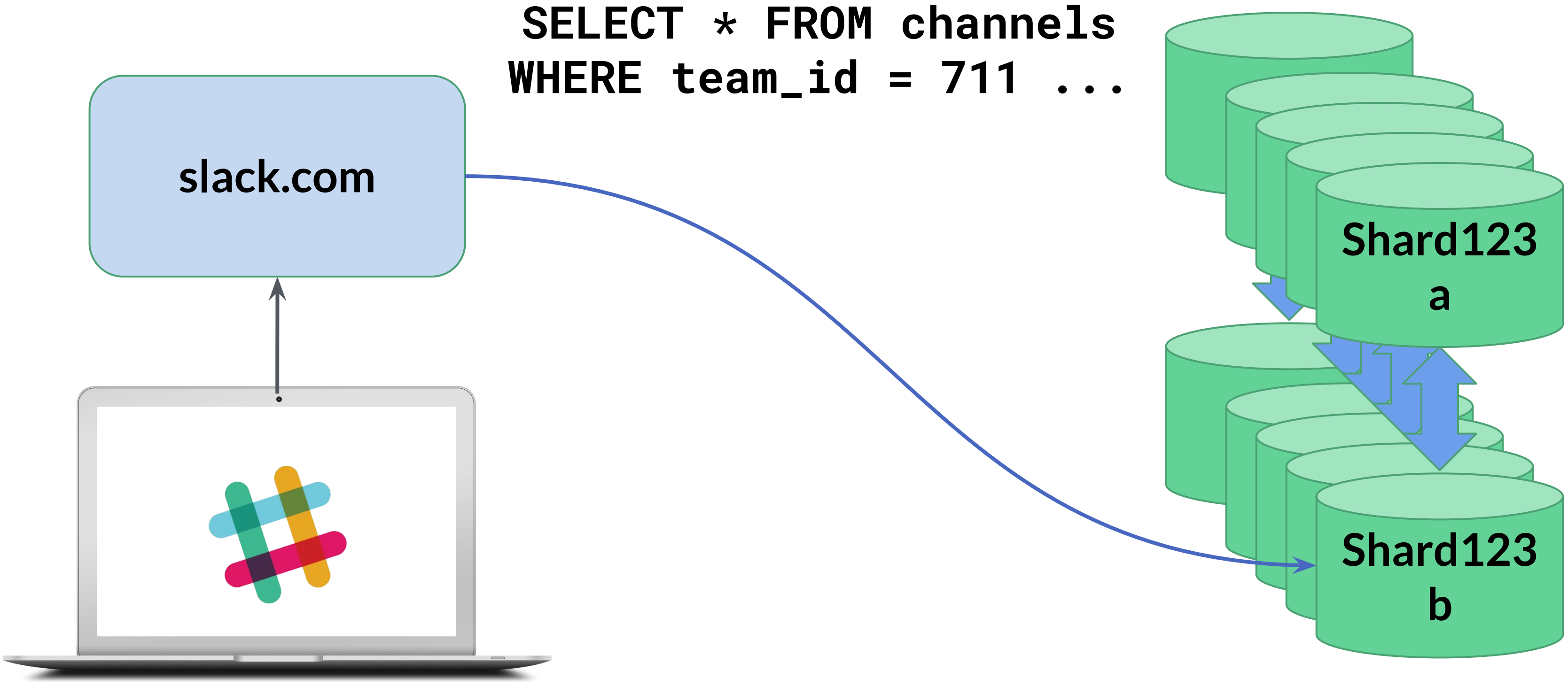
- **Q**: I’m skeptical.
  A: You’re in good company! Check out this blog post. But we should probably get on with the talk at hand ...

- **Q**: Sounds good.
  A: Right-o.

Login and Receive Messages: the “mains”



Login and Receive Messages: the shards



MySQL Shards

- **Source of truth for most customer data**
  Teams, users, channels, messages, comments, emoji, ...

- **Replication across two DCs**
  Available for 1-DC failure

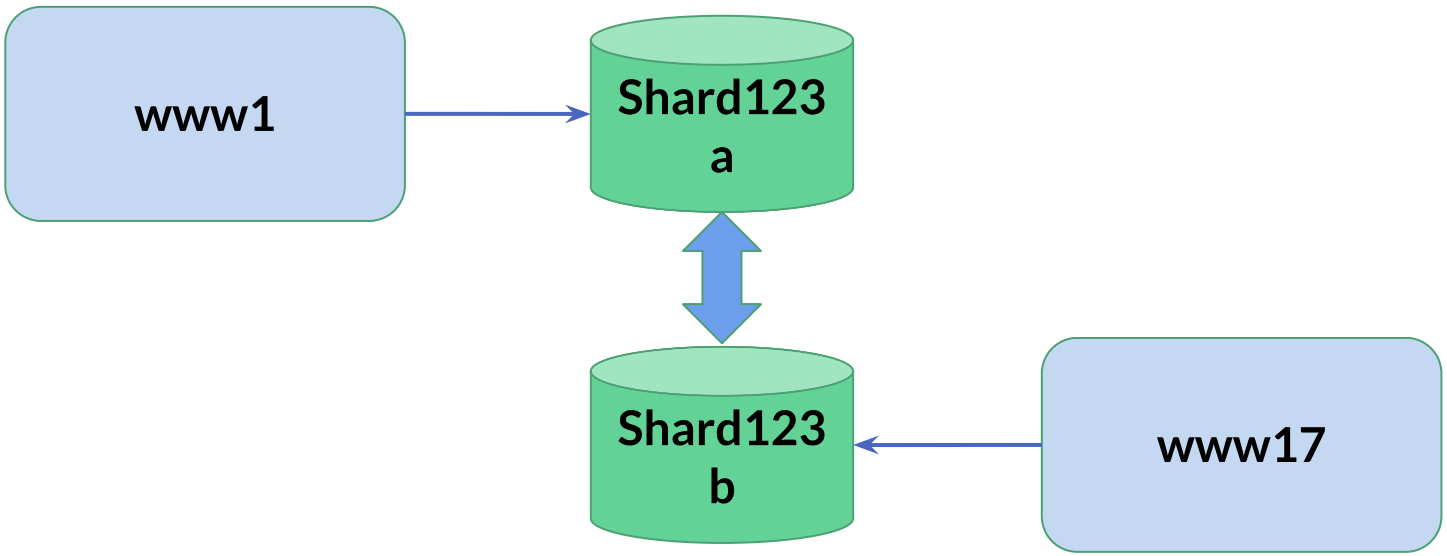
- **Sharded by team**
  For performance, fault isolation, and scalability

Why MySQL?

- Many, many thousands of server-years of working
- The relational model is a good discipline
- Experience
- Tooling

Not because of ACID, though

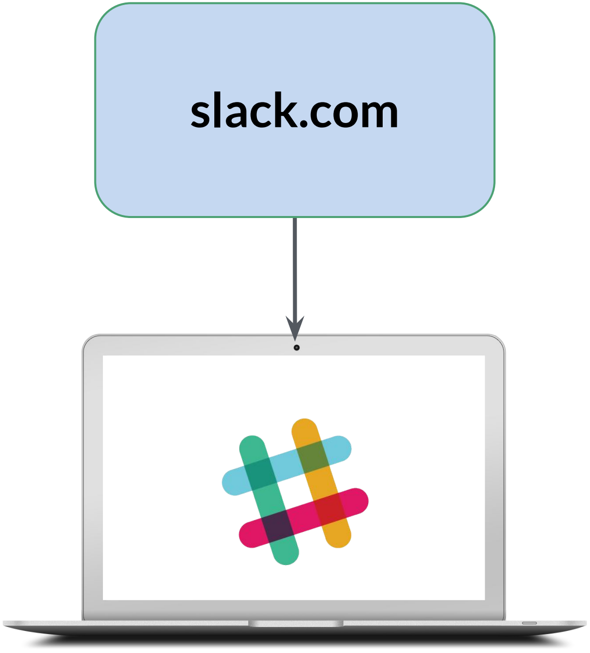
Master-Master Replication



MMR Complications

- Choosing **A** in CAP terms
- Conflicts *are* possible
  - ➞ Most resolved automatically
  - ➞ Some manually, by operator action(!)
- **`INSERT ON DUPLICATE KEY UPDATE …`**
- Partitioning by team saves us
  - ➞ Team writes cannot overlap
  - ➞ Even teams use “left” head, odd teams use “right” head

Case Study: Login and Receive Messages

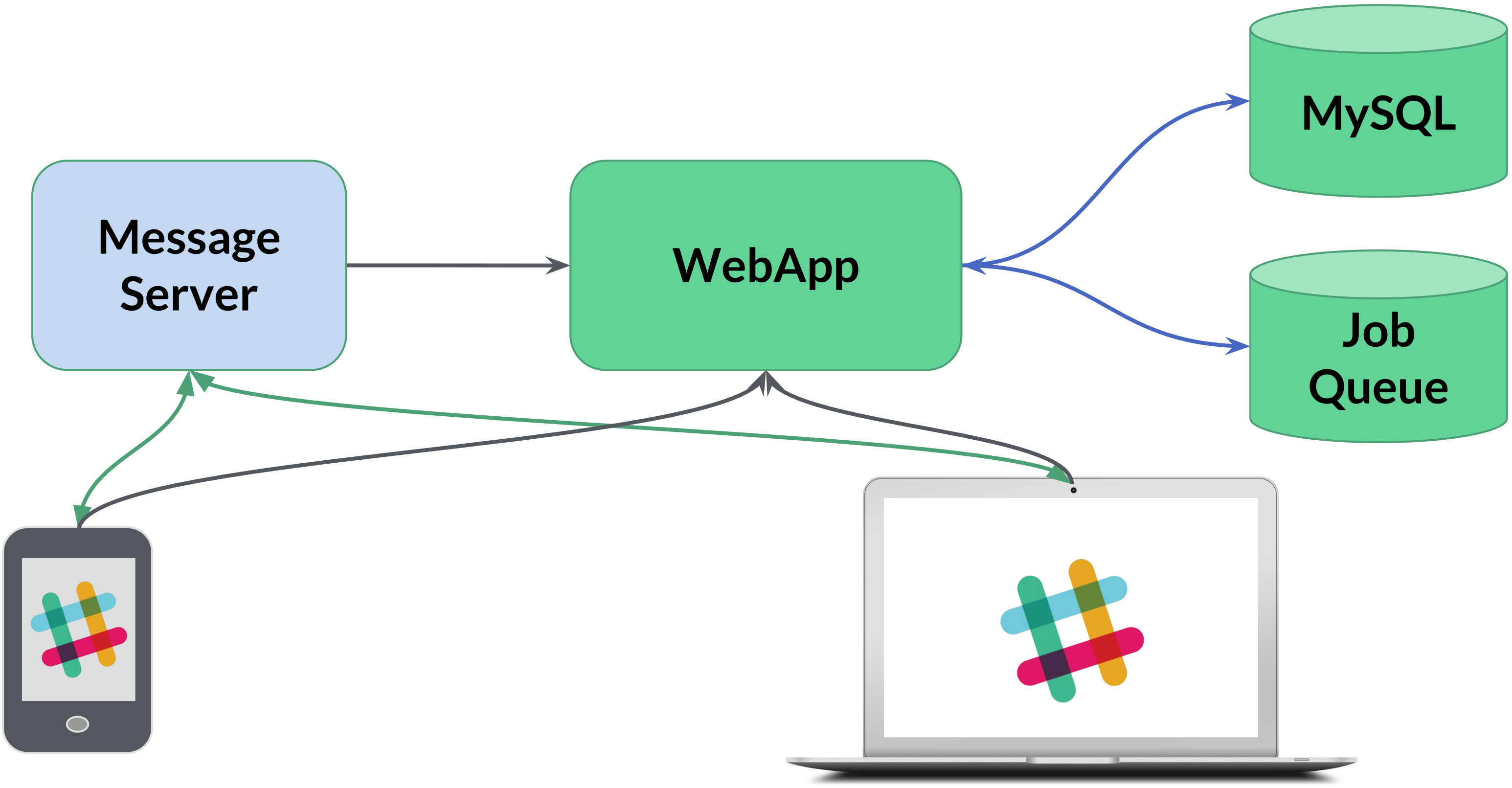



```
{
   “ok”: true,
   “url”:
“wss:\/\/ms9.slack-msgs.com\/websocket
\/7I5yBpcvk”,
  …
}
```

Rtm.start payload

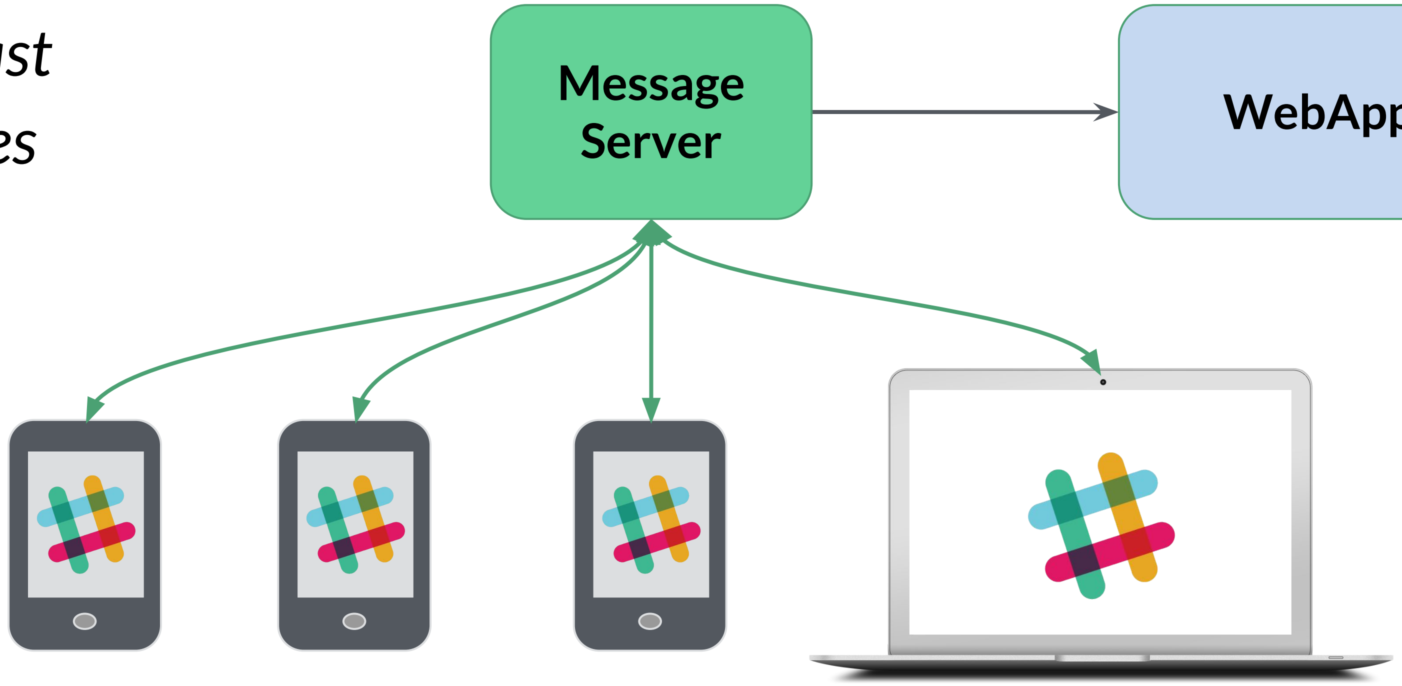
- Rtm.start returns an *image of the whole team*
- Architecture of clients
  - ➞ Eventually consistent snapshot of whole team
  - ➞ Updates trickle in through the web socket
- Guarantees responsive clients
- ...once connection is established

Cartoon Architecture of Slack



Message Delivery

Persist, broadcast messages



Wrinkles in Message Server

- Race between rtm.start and connection to MS
  → *Event log* mechanism

- Glitches, delays, net partitions while persisting
  → In-memory queue of pending sends
  → Queue depth sensitive barometer of system health

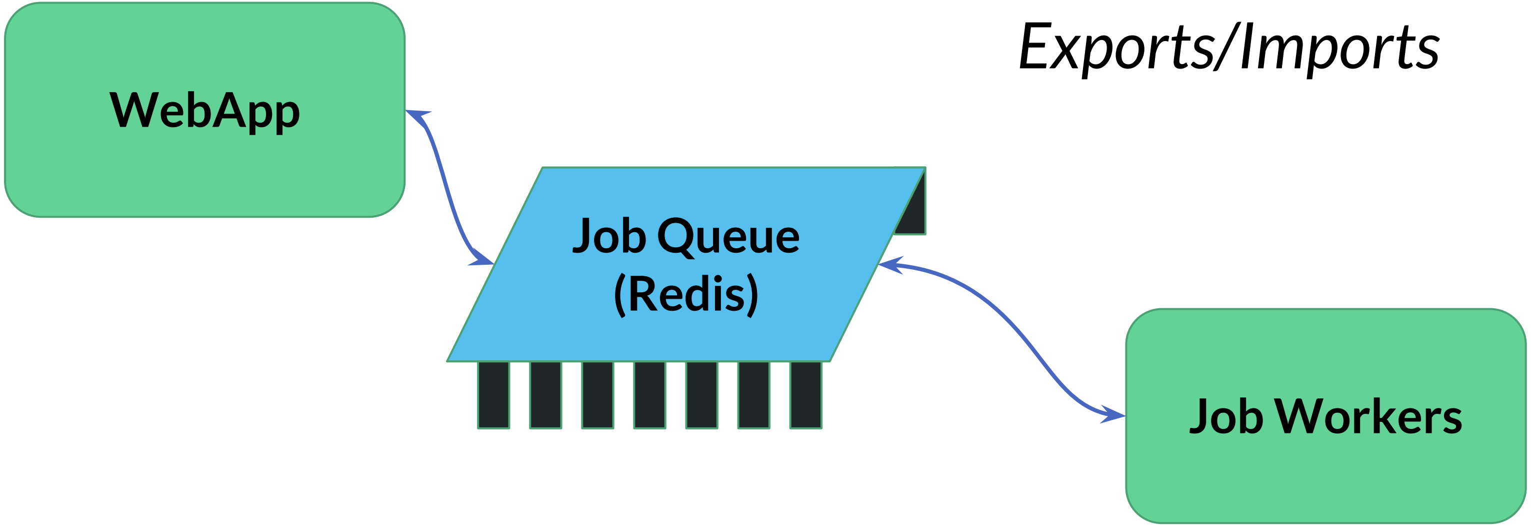
- Most messages are *presence*

Deferring Work ✨

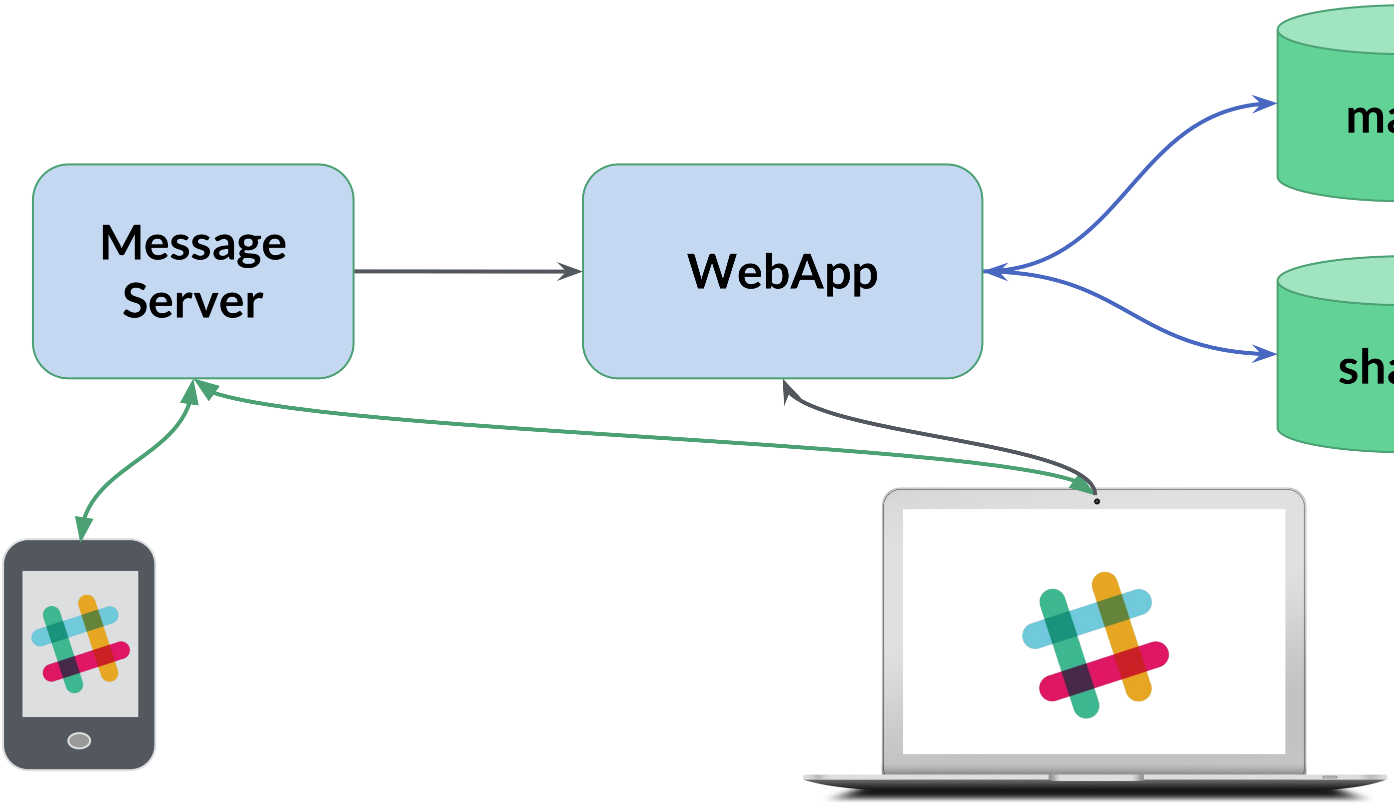
Link unfurling

Search indexing

Exports/Imports



Putting it all together



Things missing from the cartoon

- Memcache wrapped around many DB accesses
  - ➞ Case-by-case
  - ➞ Manual
- Computed data service (CDS)
  - ➞ Provides ML models via Thrift interface
- Rate-limiting around critical services
- Search!
  - ➞ Solr
  - ➞ Team-partitioned
  - ➞ fed from job queue workers

Slack Today: The Good Parts

- Team-partitioning
  - ➞ Easy scaling to lots of teams
  - ➞ Isolates failures and perf problems
  - ➞ Makes customer complaints easy to field
  - ➞ Natural fit for a paid product
- Per-team Message Server
  - ➞ Low-latency broadcasts

Some Hard Cases

Hard scenarios

- Mains failures
- Rtm.start on large teams
- Mass reconnects

Mains failure

- 1 master fails, partner takes over

- If *both* fail?
  - ➞ Many users can proceed via memcache
  - ➞ For the rest Slack is down
  - ➞ Quite possible if failure was load-induced

Rtm.start for large teams

- Returns image of *entire* team
- Channel membership is $O(n^2)$ for n users

Mass reconnects

- A large team loses, then regains, office Internet connectivity
- n users perform $O(n^2)$ rtm.start operations
- Can ‘melt’ the team shard

What are we going to

Do

about it?

Scale-out mains

- Replace *mains* spof
- With what? We're not sure yet
- Kicking the tires carefully on a scary change

Rtm.start for large teams

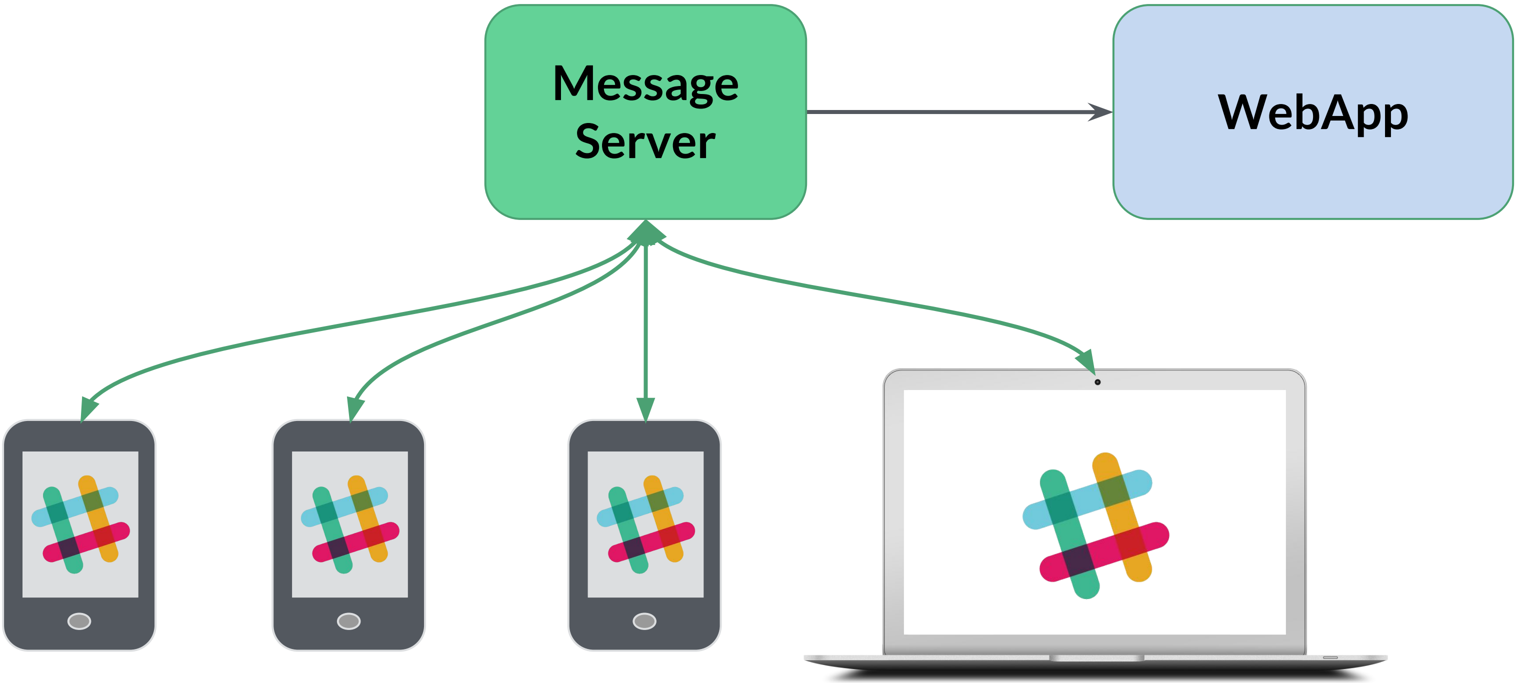
- Incremental work
  - → Current p95,p99: 221ms, 660ms
- Core problem: channel membership is $O(n^2)$
- Change APIs so clients can load channel members lazily
- Much harder than it sounds!

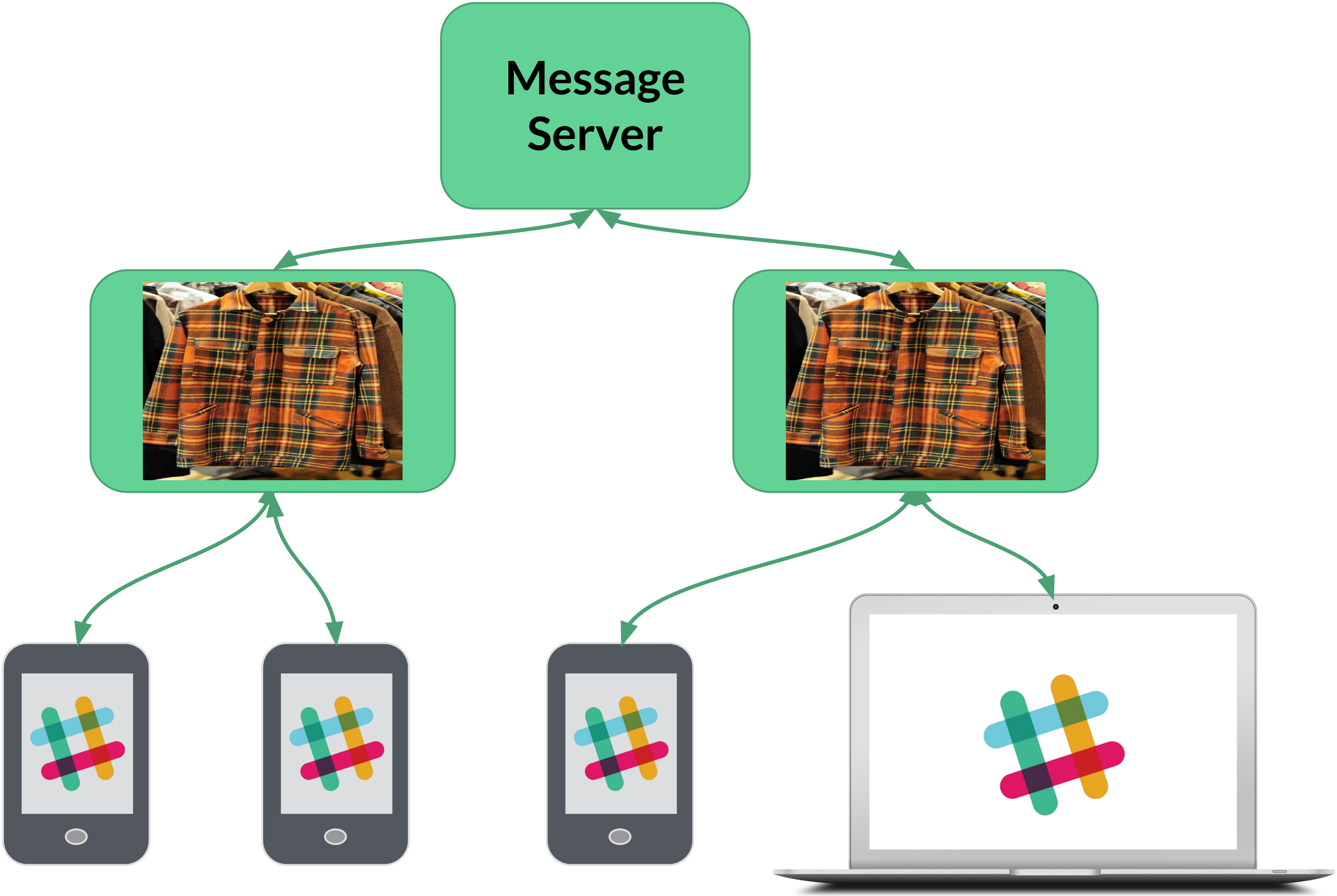
Mass reconnects

- Introducing flannel
- Application-level edge cache

Message Delivery

Pre-Flannel




Message
Server

Flannel status

- On for a few teams
- Rolling out to you soon with any luck

Phew

Stuff I had to leave out

- Lots of client tech!
- Voice
- Backups
- Data warehouse
- Search
- Deploying code
- Monitoring and alerting

Wrapping up

- Sketch of how Slack works
  - ➞ Application Logic
  - ➞ Persistence
  - ➞ Real-time messaging
  - ➞ Asynchronous Work
- Problems
- What we’re doing about them

There is a lot left to do

slack.com/jobs

•••

Deployable Message Server

- Channel-sharded message bus

- Flannel discovers Channel servers via Consul
  - → Scatters user writes
  - → Gathers channel reads

- Failures do not need reconnects