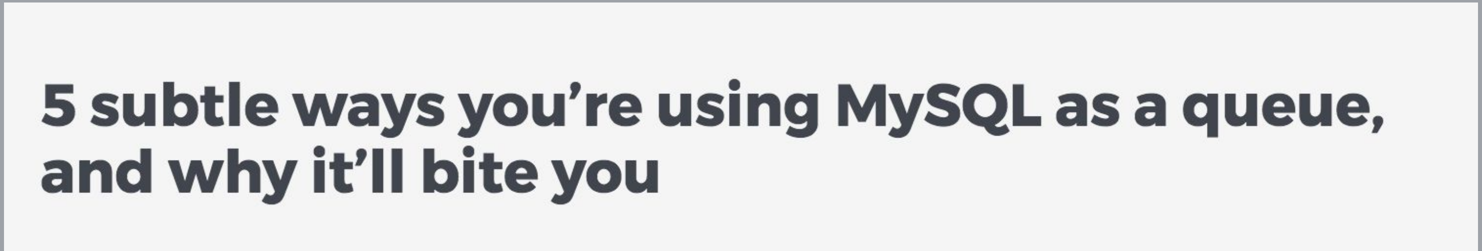
Nonconformist Resilience:

Database-backed Job Queues

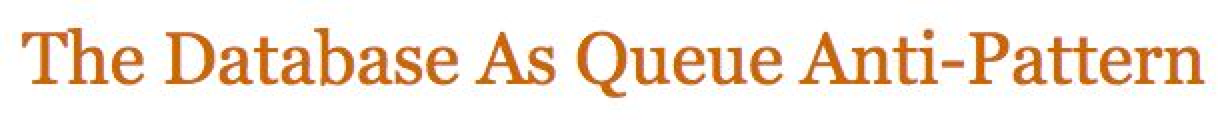
John Mileham | @jmileham

Betterment

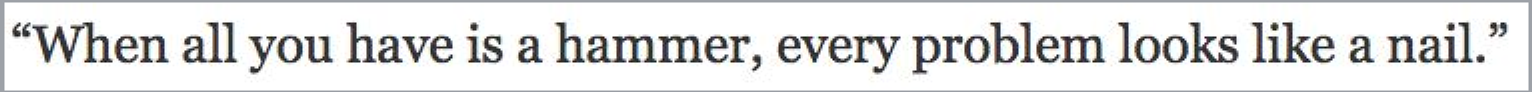
Why a database is not always the right tool for a queue based system



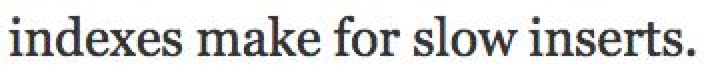
5 subtle ways you’re using MySQL as a queue, and why it’ll bite you



The Database As Queue Anti-Pattern



“When all you have is a hammer, every problem looks like a nail.”



indexes make for slow inserts.

Polling.

Locking.

Data growth.

SCALING

MANUAL CLEANUP

MANUAL HANDLING OF THE COMPLEXITY

F THE COMPLEXITY

MPLEXI

User Signup
with Email Confirmation

User Signup
with Email Confirmation

A feature so easy we’re still fighting about how to do it in 2017

Requirements:

Validate the user's profile information

Store the user record to the database

Email a link

When the link is clicked, mark the user as verified

Requirements:

Validate the user's profile information

Store the user record to the database

Email a link

When the link is clicked, mark the user as verified

Take 1:

Inline the email delivery

Take 1:

Inline the email delivery

… but it’s slow

Take 2:

Spin off a thread or use a thread pool

Take 2:

Spin off a thread or use a thread pool

… but it’s unreliable

Take 3:

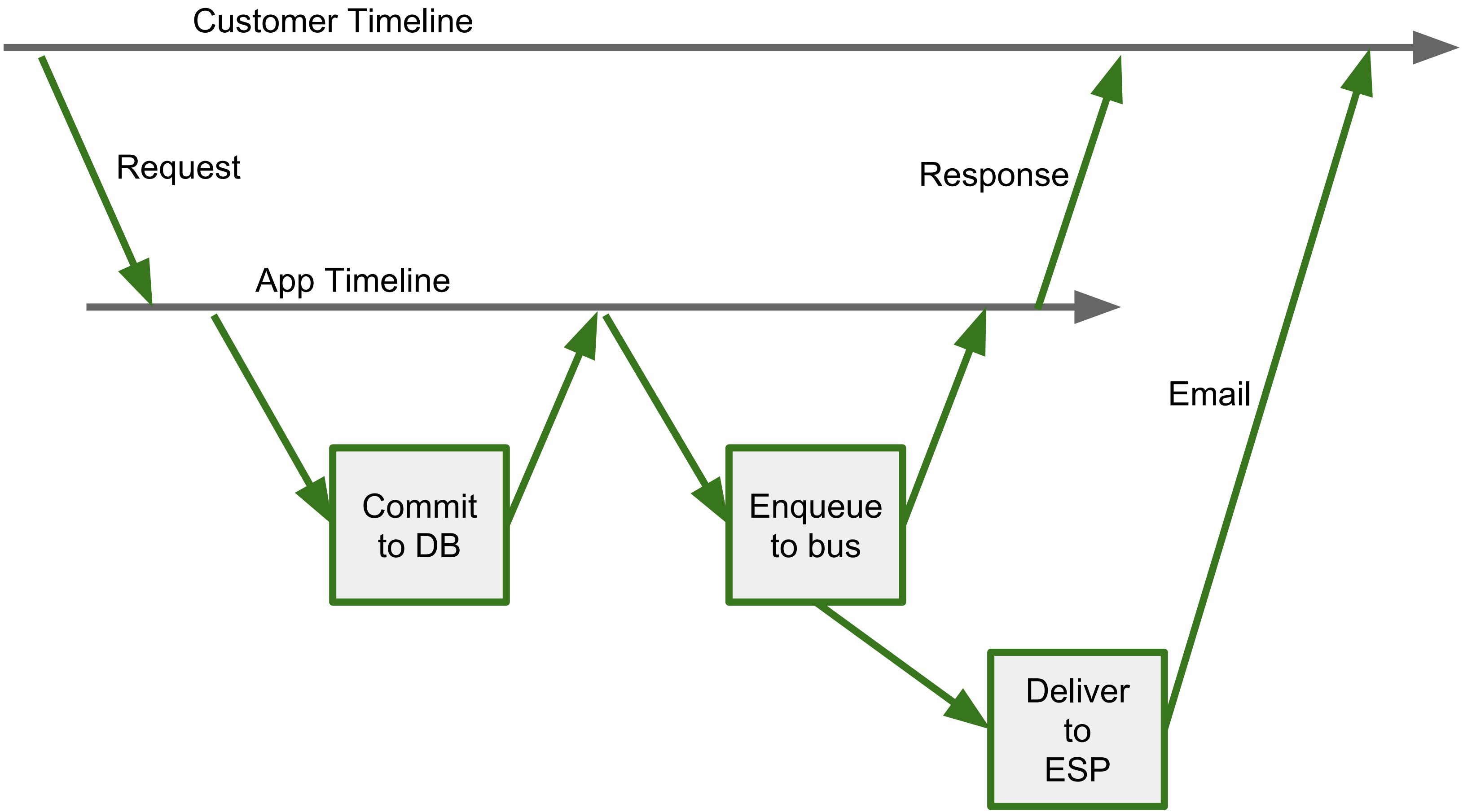
Use a grown-up message bus

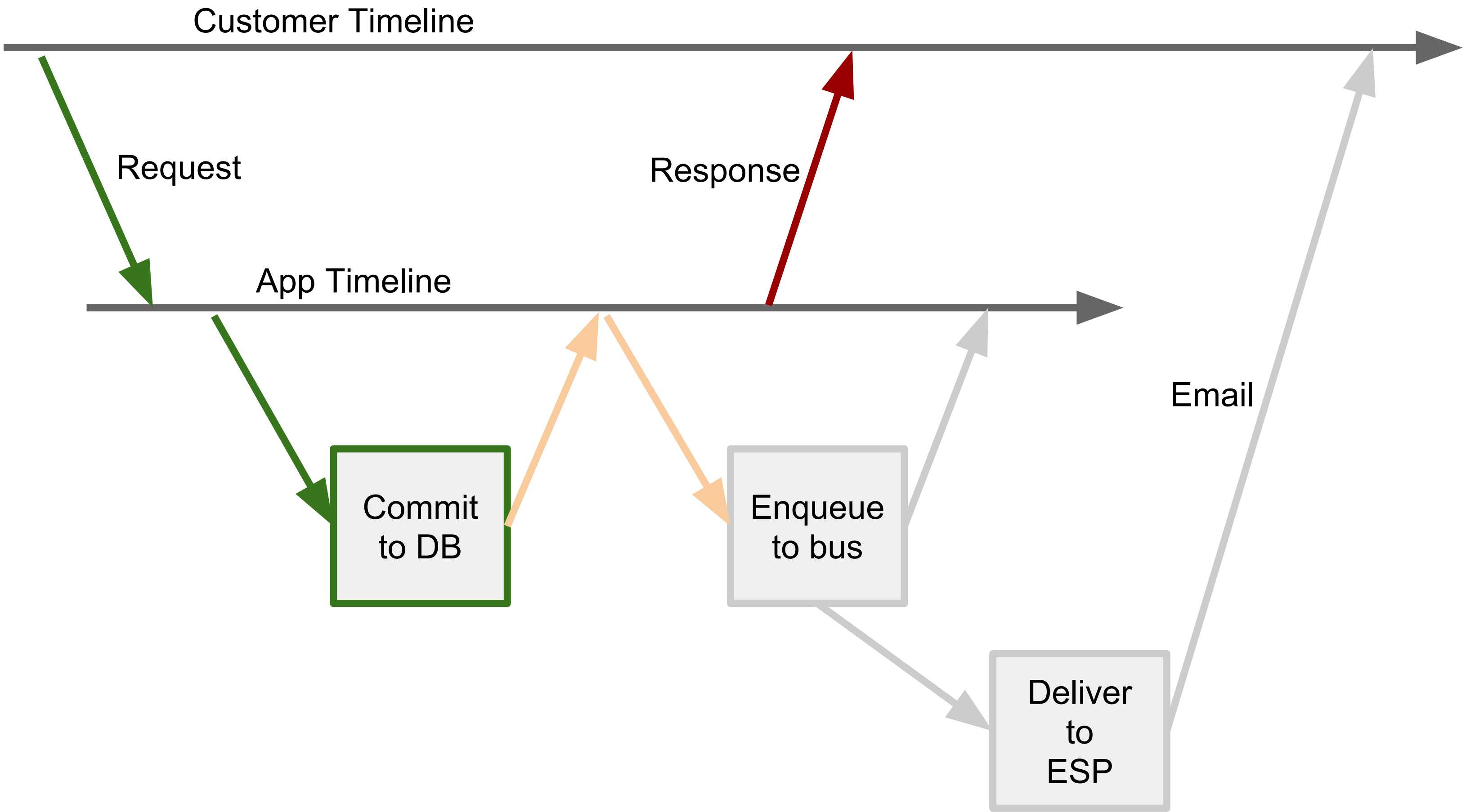
Take 3:

Use a grown-up message bus

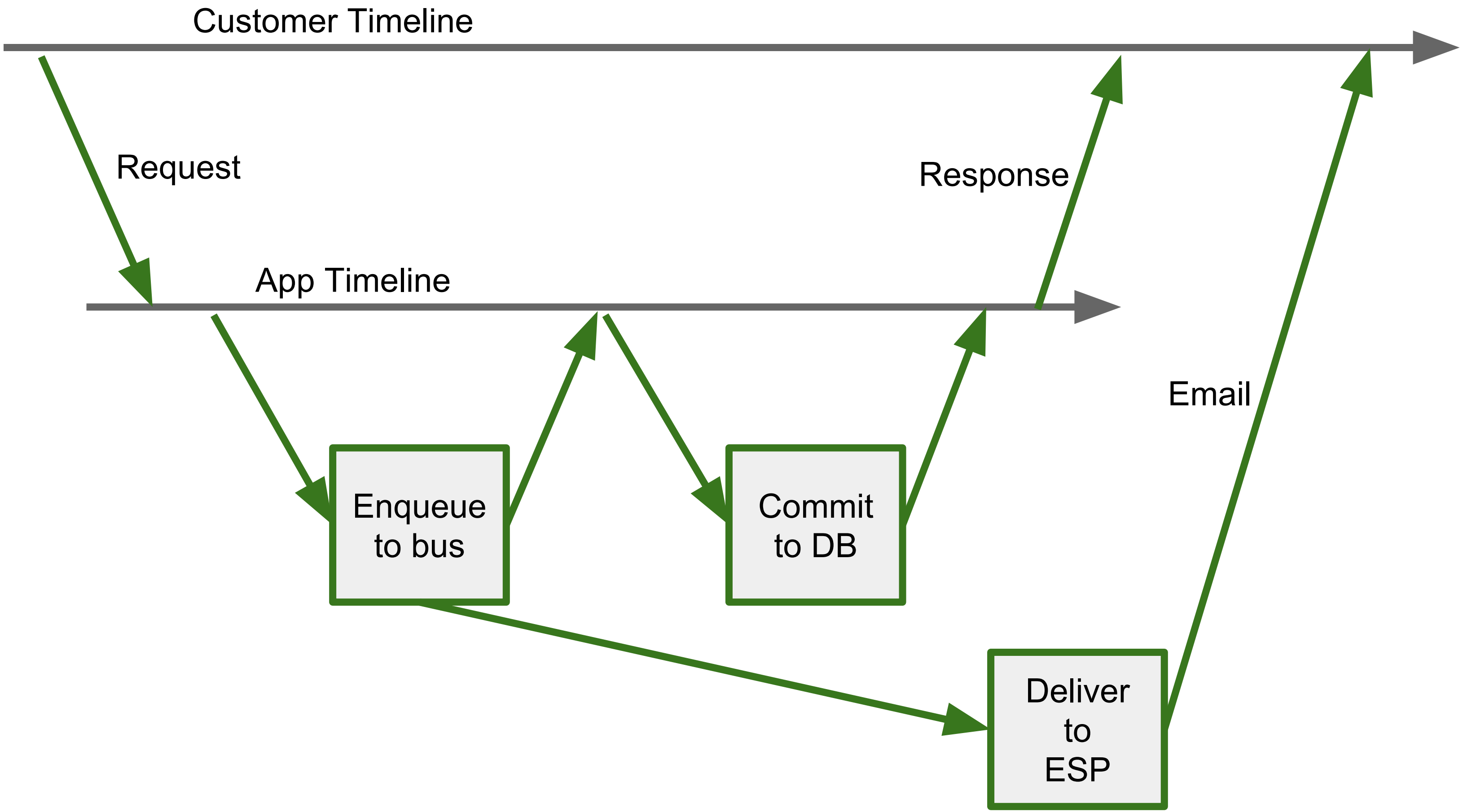
… but it’s unreliable?

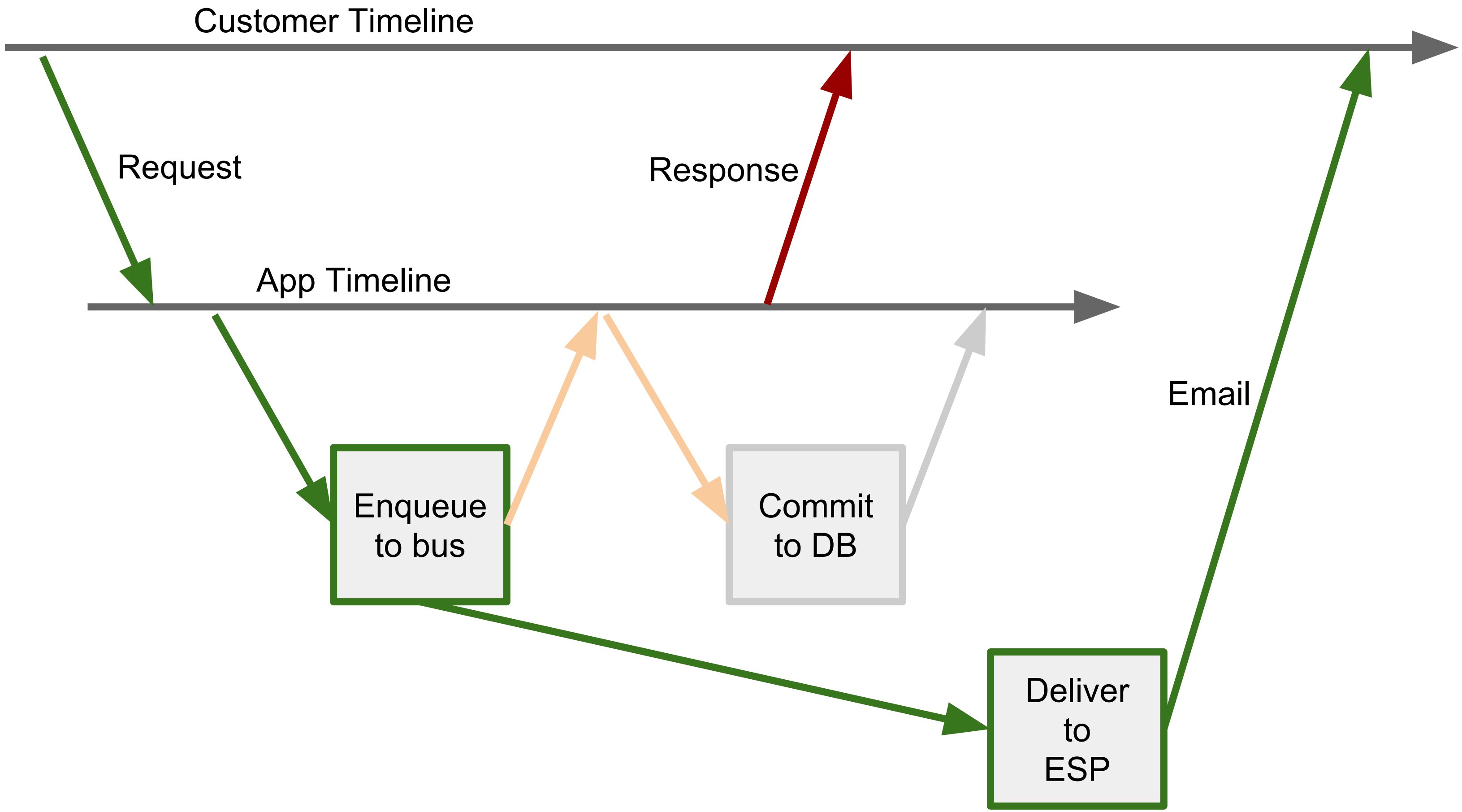
Commit-then-Enqueue

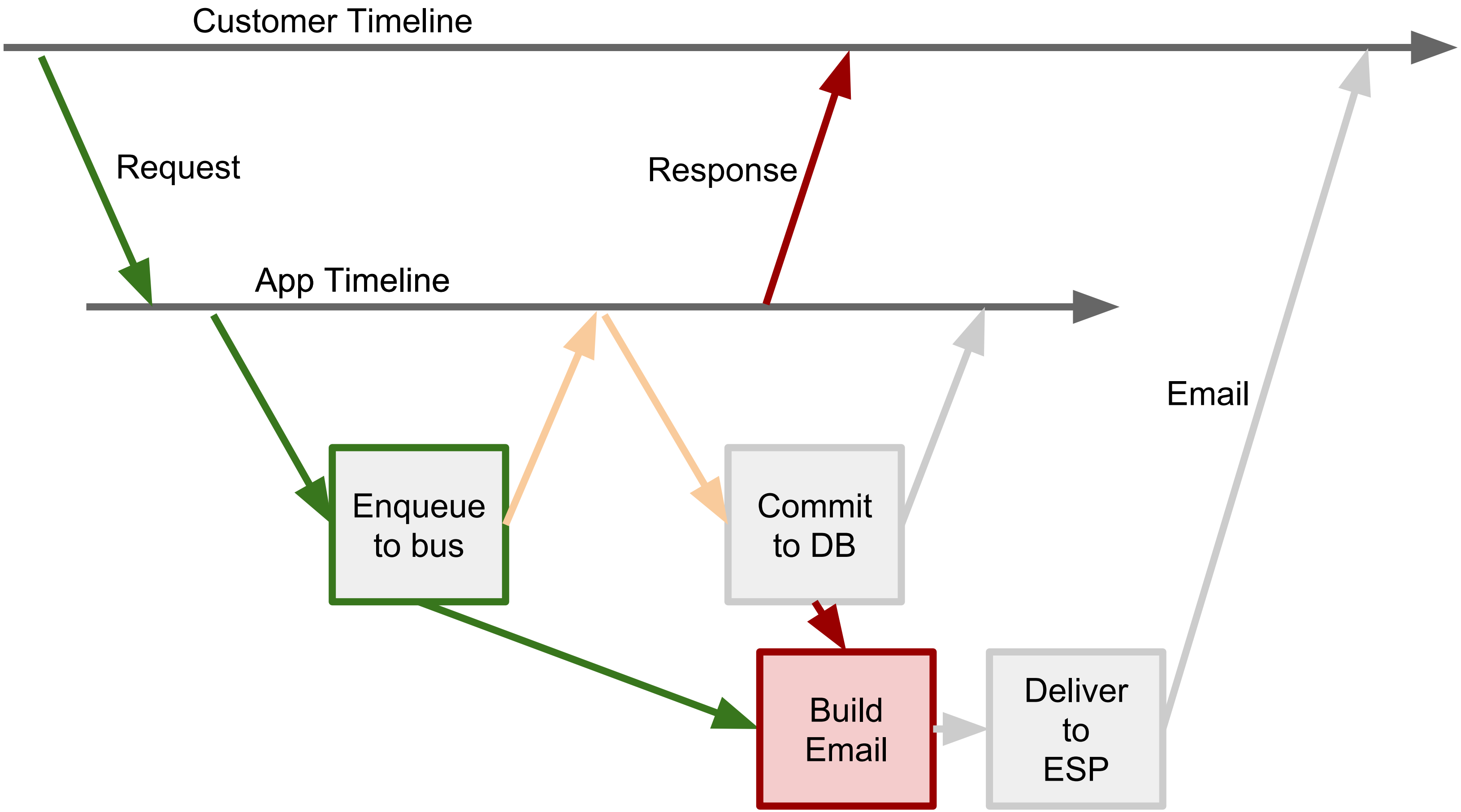

Customer Timeline
Request
Response
App Timeline
Email
Commit to DB
Enqueue to bus
Deliver to ESP


Customer Timeline
Request
Response
App Timeline
Email
Commit to DB
Enqueue to bus
Deliver to ESP

Enqueue-then-Commit


Customer Timeline
Request
Response
App Timeline
Email
Enqueue to bus
Commit to DB
Deliver to ESP


Customer Timeline
Request
Response
App Timeline
Email
Enqueue to bus
Commit to DB
Deliver to ESP


Customer Timeline
Request
Response
App Timeline
Email
Enqueue to bus
Commit to DB
Build Email
Deliver to ESP

Distributed Transactions

You could make the enqueue and the database commit atomic via a distributed transaction manager, but:

- Mature, robust, distributed transaction managers aren’t available for all platforms
- They’re usually proprietary
- Even where they exist, these tools have nuanced configuration, can have operational warts and are another subsystem that requires care and feeding
- The additional network pings necessary to coordinate the commit between the datastores can cause write performance problems

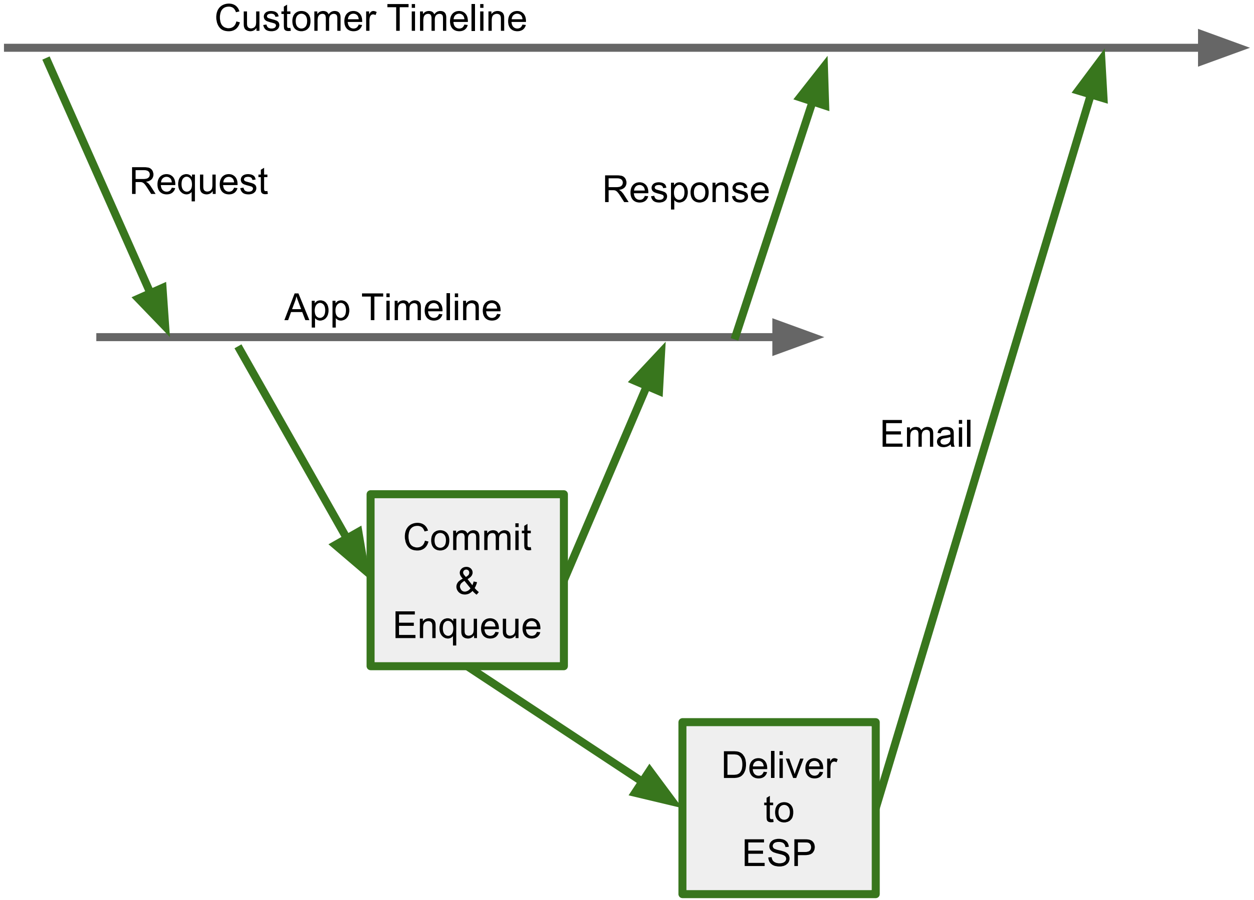
Take 4:

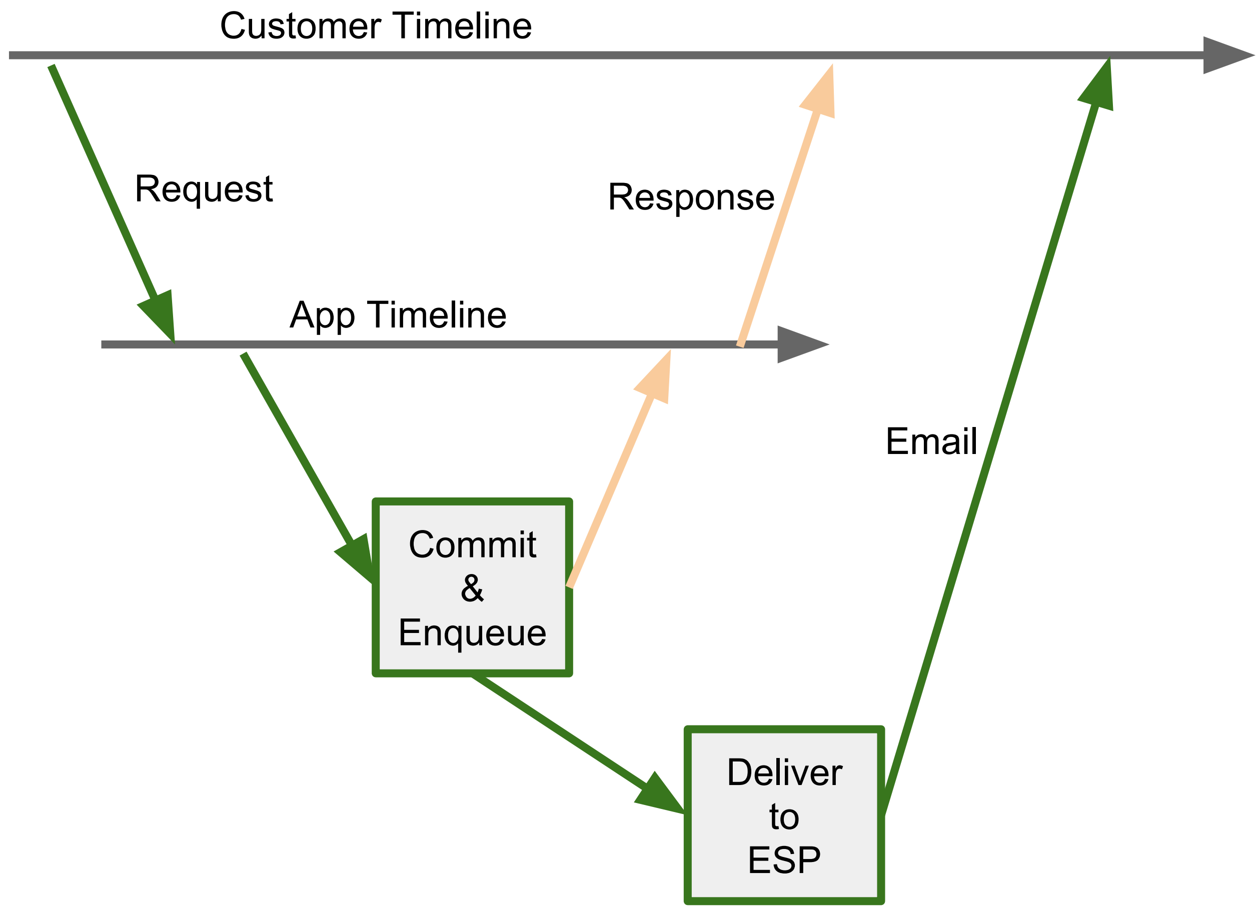
Use the database as a queue

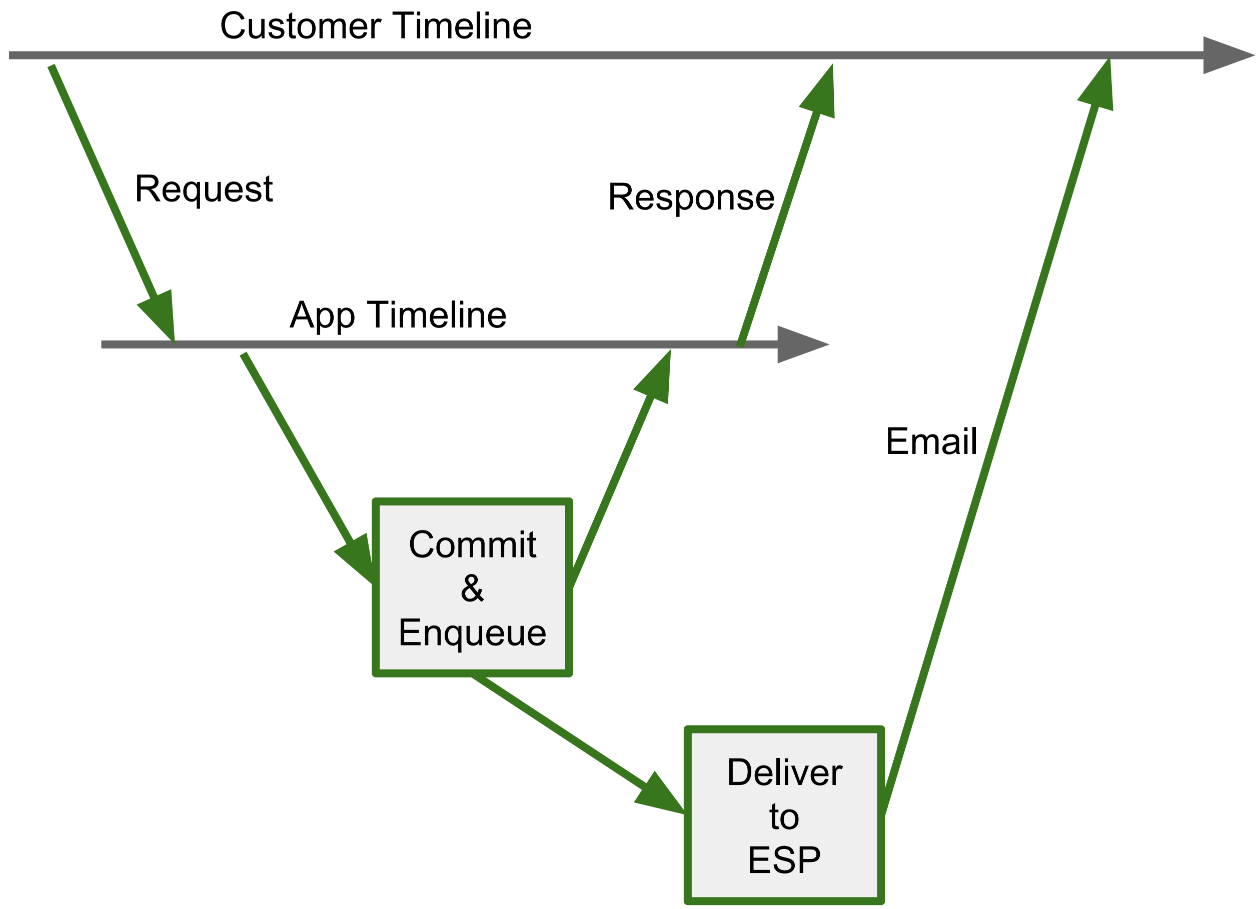
Take 4:

Use the database as a queue

… but it won’t scale


Customer Timeline
Request
Response
App Timeline
Email
Commit & Enqueue
Deliver to ESP


Customer Timeline
Request
Response
App Timeline
Email
Commit
&
Enqueue
Deliver
to
ESP


Customer Timeline
Request
Response
App Timeline
Email
Commit
&
Enqueue
Deliver
to
ESP

Robust By Default

Addressing the pitfalls

Because everything is a tradeoff

MANUAL HANDLING OF THE COMPLEXITY

DJ: Retry with Exponential Backoff

Two key columns: `run_at`, and `attempts`.

- Jobs are picked up oldest first
- Only jobs with a `run_at` in the past are workable
- When a job fails, a new future `run_at` is calculated from the previous `run_at` and number of previous `attempts`
- After too many failures (days later), a job will stop being attempted

Message Bus Solution: DLQs

Messages don’t have a desired delivery time in a message bus, so exponential backoff isn’t feasible.

Message delivery will be attempted a preconfigured number of times, and then transferred to a dead-letter queue, or a cascading set of queues to approximate exponential backoff.

DJ: Priority

Delayed::Job will work off the highest priority first.

Pickup is simply a matter of sorting on `priority` and then `run_at`.

We use priority to establish different service level objectives for different kinds of work.

Allows developer not to worry about resourcing their jobs, leaning into DJ.

Allows DJ to fully utilize its worker capacity.

Message Bus Solution: Topics

Message busses can’t as easily support priority.

To assure resource availability for important work, work is shunted to a specific topic or queue with its own resource pool.

Strong assurance that one job type won’t exhaust resources of another type.

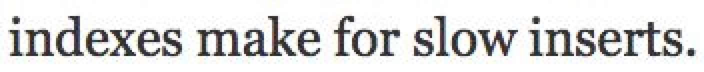
But you must resource each topic individually.

DJ’s got topics too ;)

Even though it’s not the only way to organize work, if you have a mission critical work stream that must be processed no matter what, you can use a specialized queue to keep its workers separate.

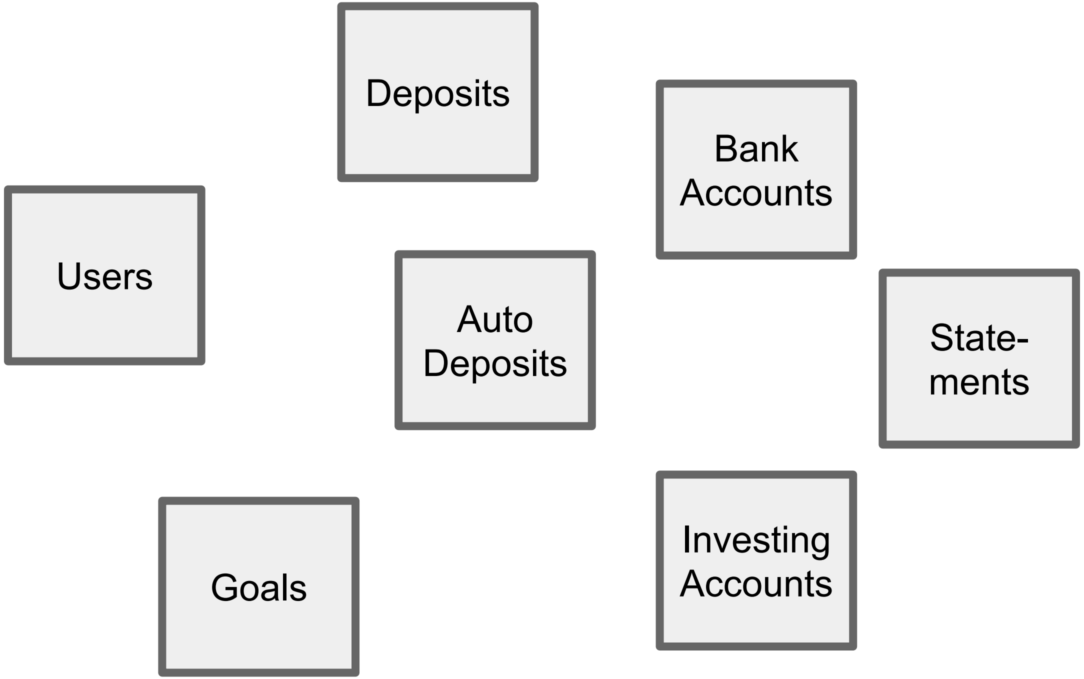
Opt in for as much control as you need, only when you need it.

More Featureful, Not Less



indexes make for slow inserts.

Betterment’s Schema



Deposits

Bank Accounts

Users

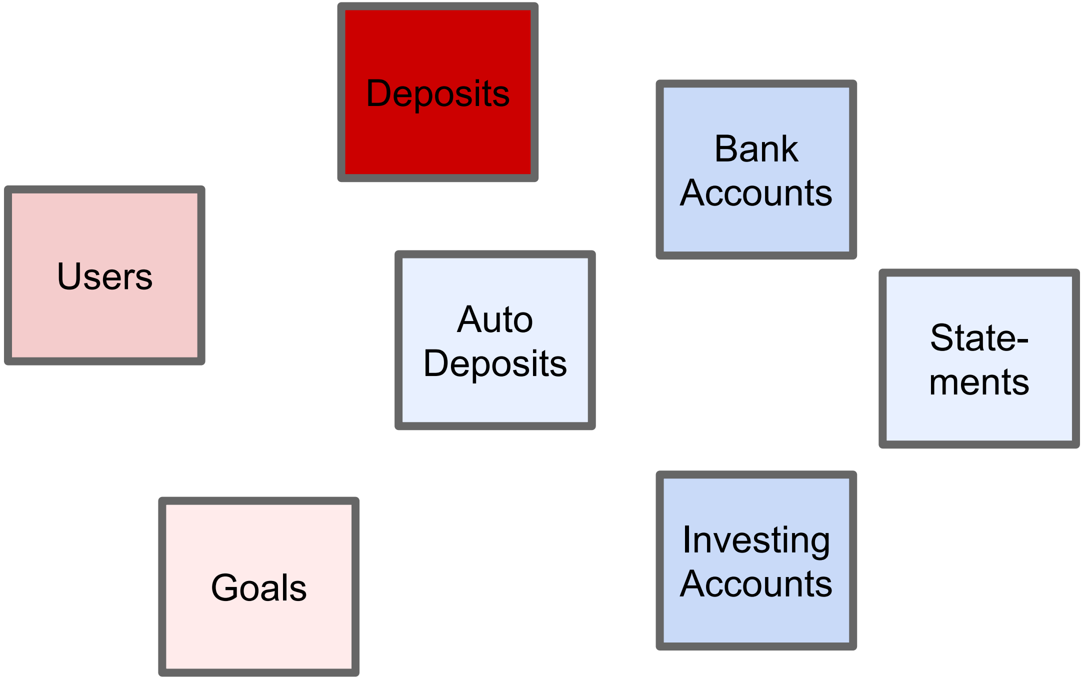
Auto Deposits

State-ments

Goals

Investing Accounts

Betterment’s Schema



Deposits

Bank Accounts

Users

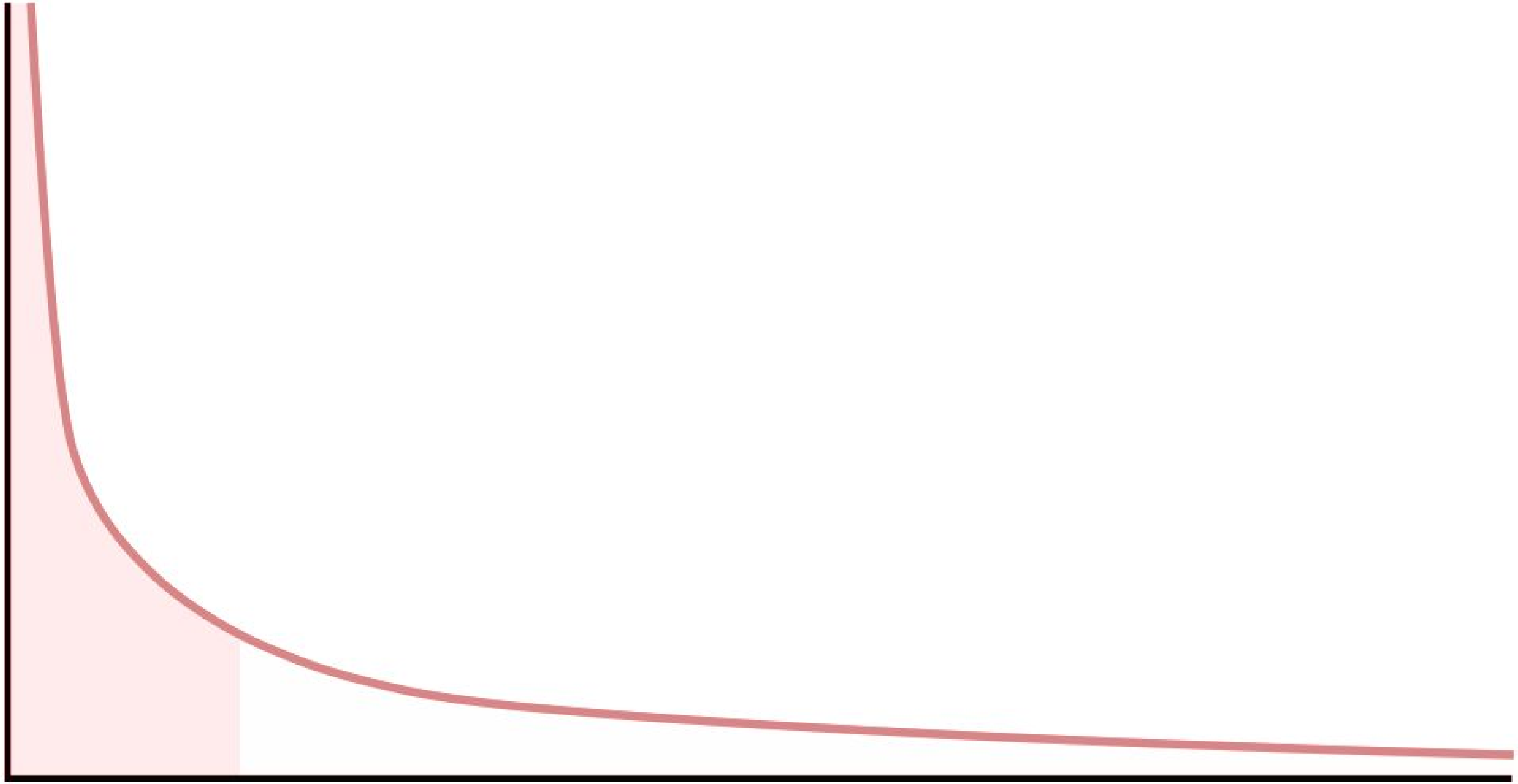
Auto Deposits

State-ments

Goals

Investing Accounts

Power Law Distribution



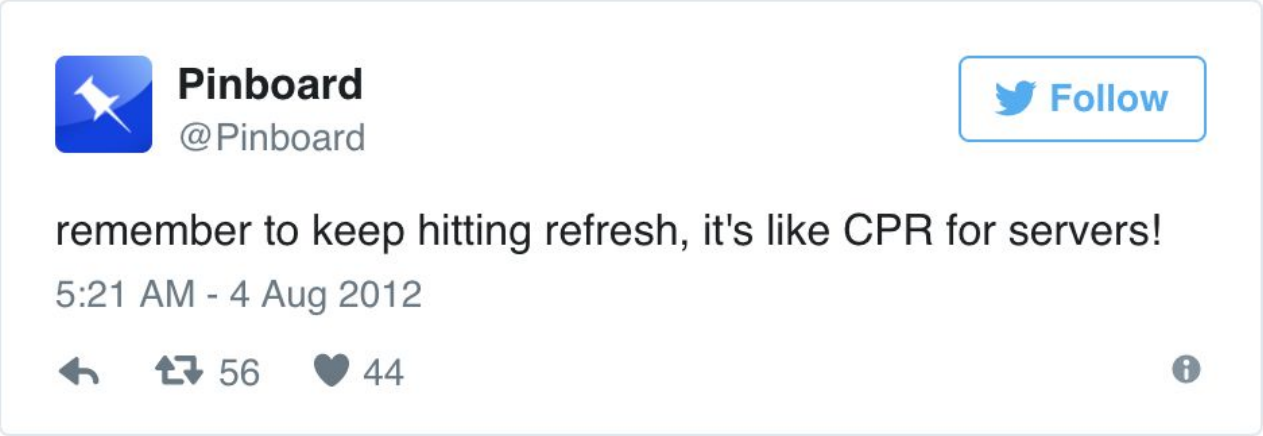
The Message Bus Isn’t a Silver Bullet

Polling.

Coordinated Polling

- Your application chooses a global polling interval, say a half second.
- Every active worker process inserts itself into an `active_workers` table with a `last_active_at` timestamp and maintains it every 30 seconds or so.
- Every few seconds, each worker queries the number of recently active workers.
- It then multiplies the global polling interval by the number of workers and adds random jitter to prevent thundering herds
- Your app converges on the desired polling interval at arbitrary worker scale

Data growth.



remember to keep hitting refresh, it's like CPR for servers!

5:21 AM - 4 Aug 2012

When is a DB-backed queue the right tool?

1. Should your app use a DB at all?

You should be using an ACID SQL DB if:

- You have a read-heavy usage pattern
- You value agility in supporting new use cases
- You aren’t launching directly into #webscale
- Or even if you are, your app doesn’t exist primarily to solve a graph problem
  - if you’re going big and still want to use SQL, your dataset must inherently shardable

2. Are your clients human?

If clients are interacting with your app like humans, i.e.:

- They do individual operations at a reasonable pace
- The don’t generate batches of 10,000 operations at once

Then you’re looking still looking good.

3. Are Your Bulk Operations Cool?

- Are there relatively few of them?
- Are they customer experience-impacting?
- Are they no more than daily?

All Yes? All Set.

Operating a DB-backed Queue

Alerting Needs

Two key alerts:

1. Max attempt count
2. Max age

Both metrics are partitioned by job priority.

Max Attempt Count

Total backoff time function: `n == 0 ? 0 : n ** 4 + 5 + backoff(n-1)`

- First retry in 6 seconds
- Third retry in 2 minutes
- Fifth retry in 16 minutes
- Tenth retry in 7 hours
- Twentieth retry in 8 days

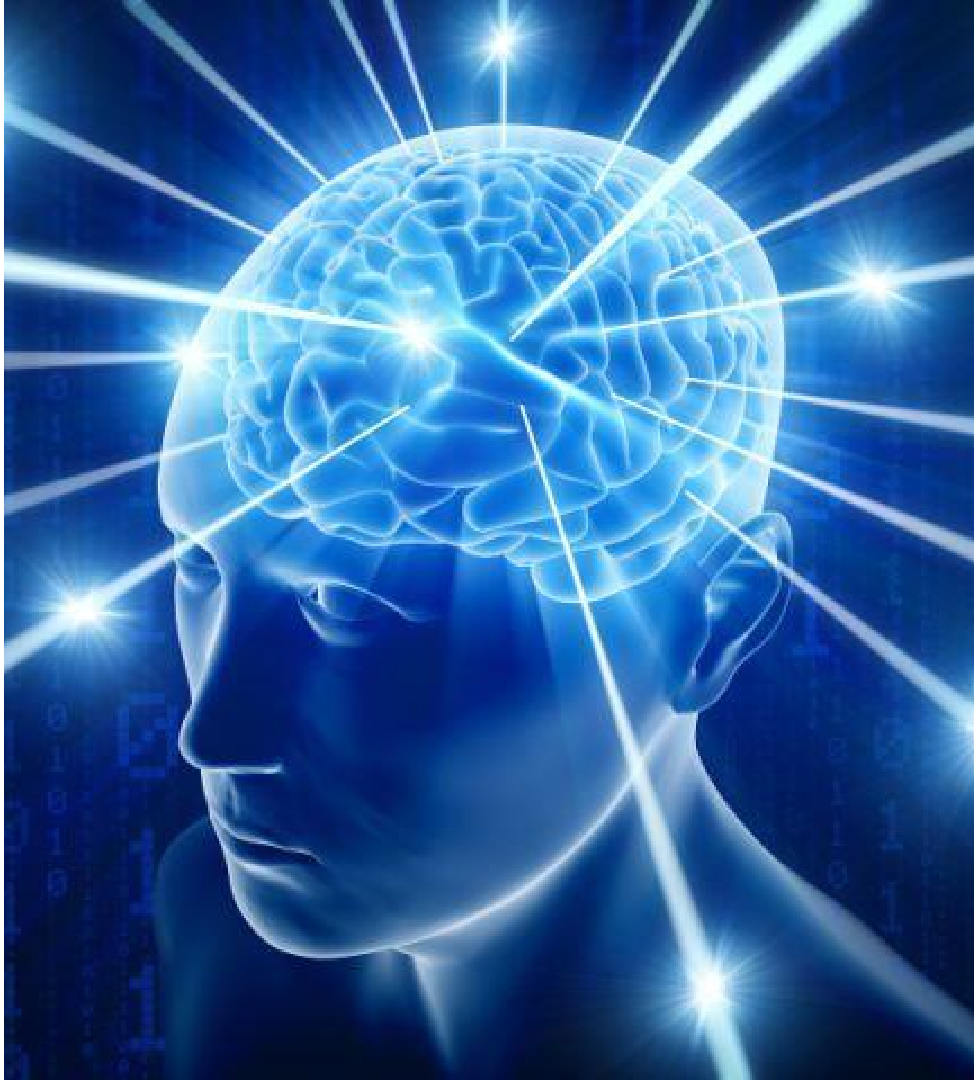
Our thresholds:

- INTERACTIVE errors after 2 attempts (~30 seconds)
- EVENTUAL errors after 8 attempts (~2.5 hours)

Max Age

Age is defined as now() - run_at.

This is your brain on DJ



(your message may vary)

Why not just use DJ?

Why not just use DJ?

Date

June 27th, 2017

Author

John Mileham | @jmileham

Betterment

(is hiring)