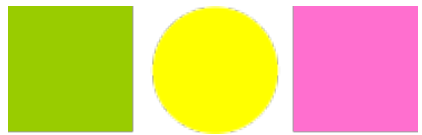
PubSubHubbub, and App Engine



Brett Slatkin
Software Engineer
Google Inc.

February 20th, 2010

Agenda

- Intro
- Demo
- Howto
- Building a hub
- Progress

Questions

Post in comments on Buzz!

http://tinyurl.com/hubbub-pycon

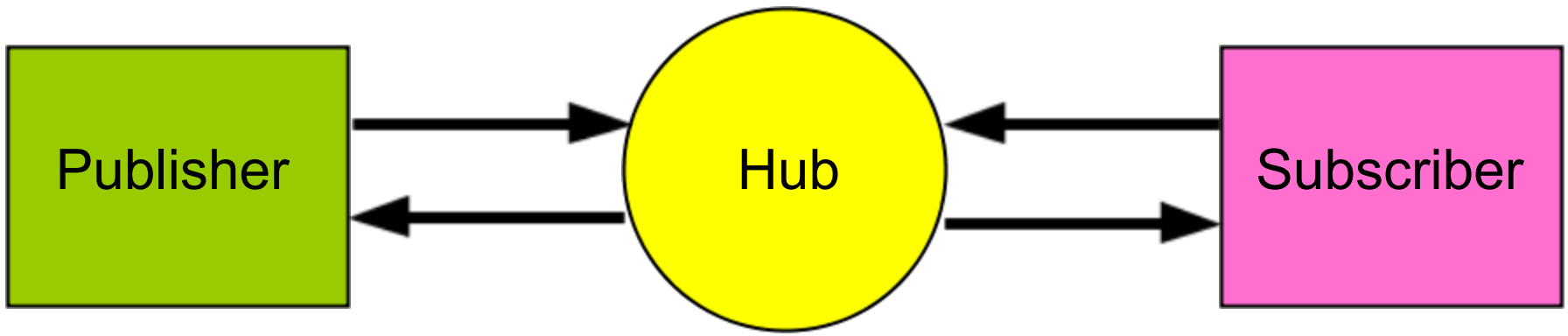
Project site

http://pubsubhubbub.googlecode.com

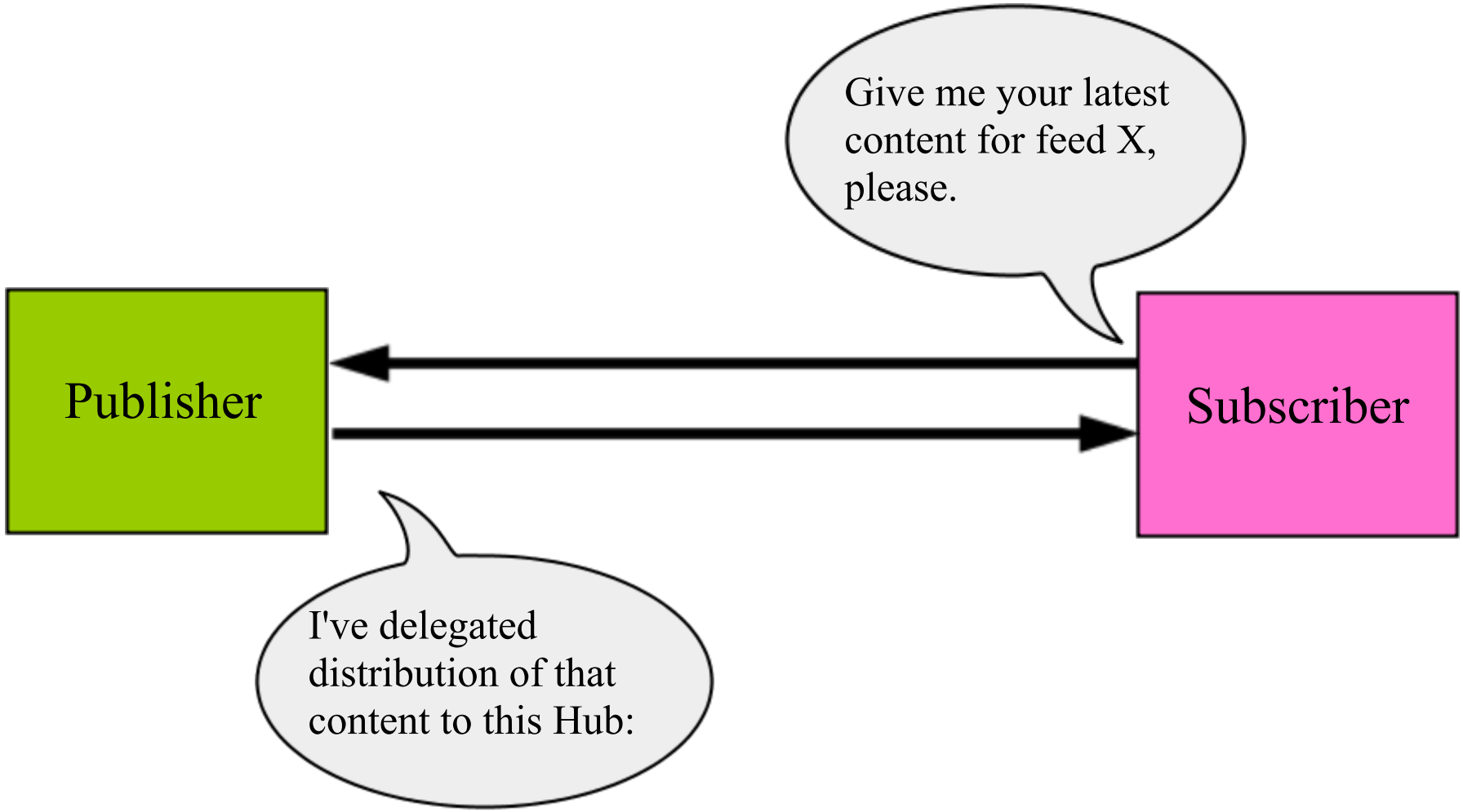
Intro

What is PubSubHubbub?

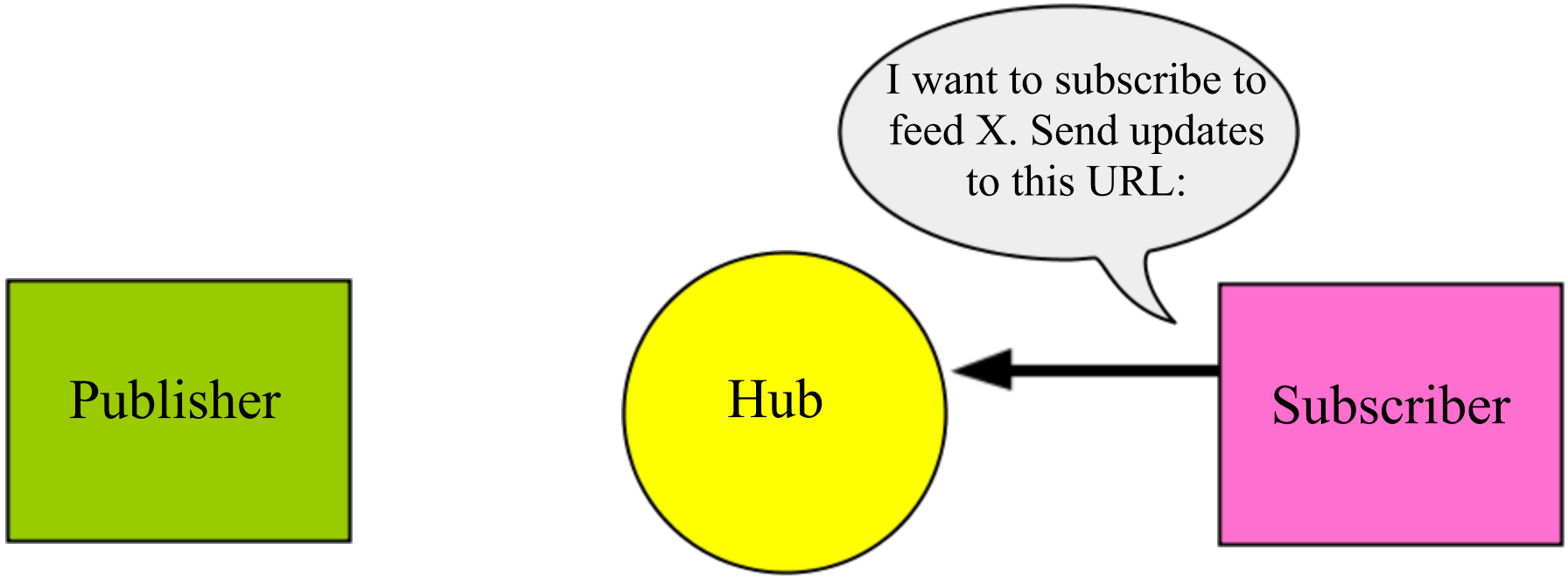
- A simple, server-to-server publish/subscribe protocol
- Uses HTTP-- worse is better
- Turns Atom and RSS feeds into real-time streams
- A **single API** for web-scale, low-latency messaging
- Three participants: Publisher, Subscriber, Hubs



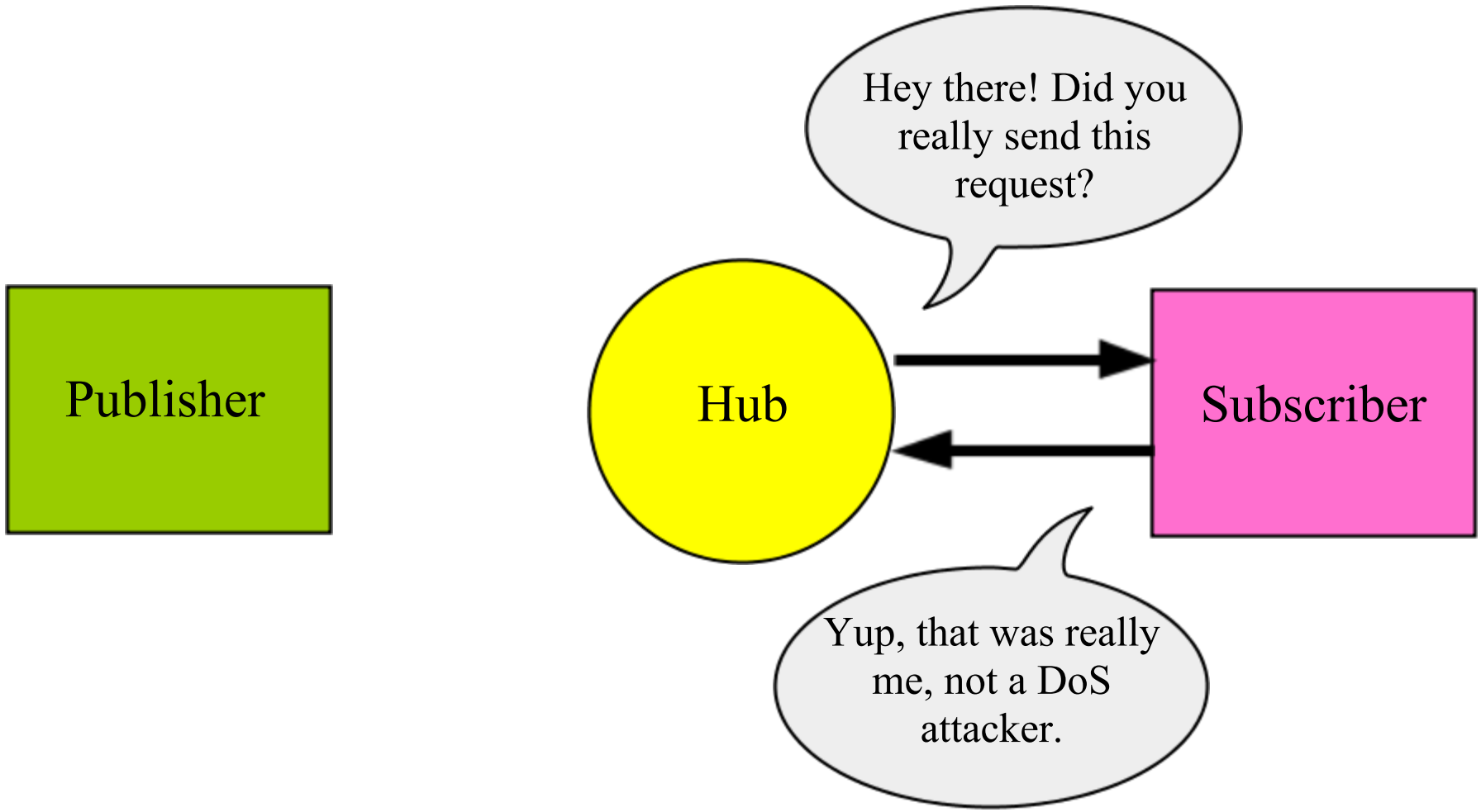
1. Subscriber polls Publisher's feed. The feed contains a link to the Hub.



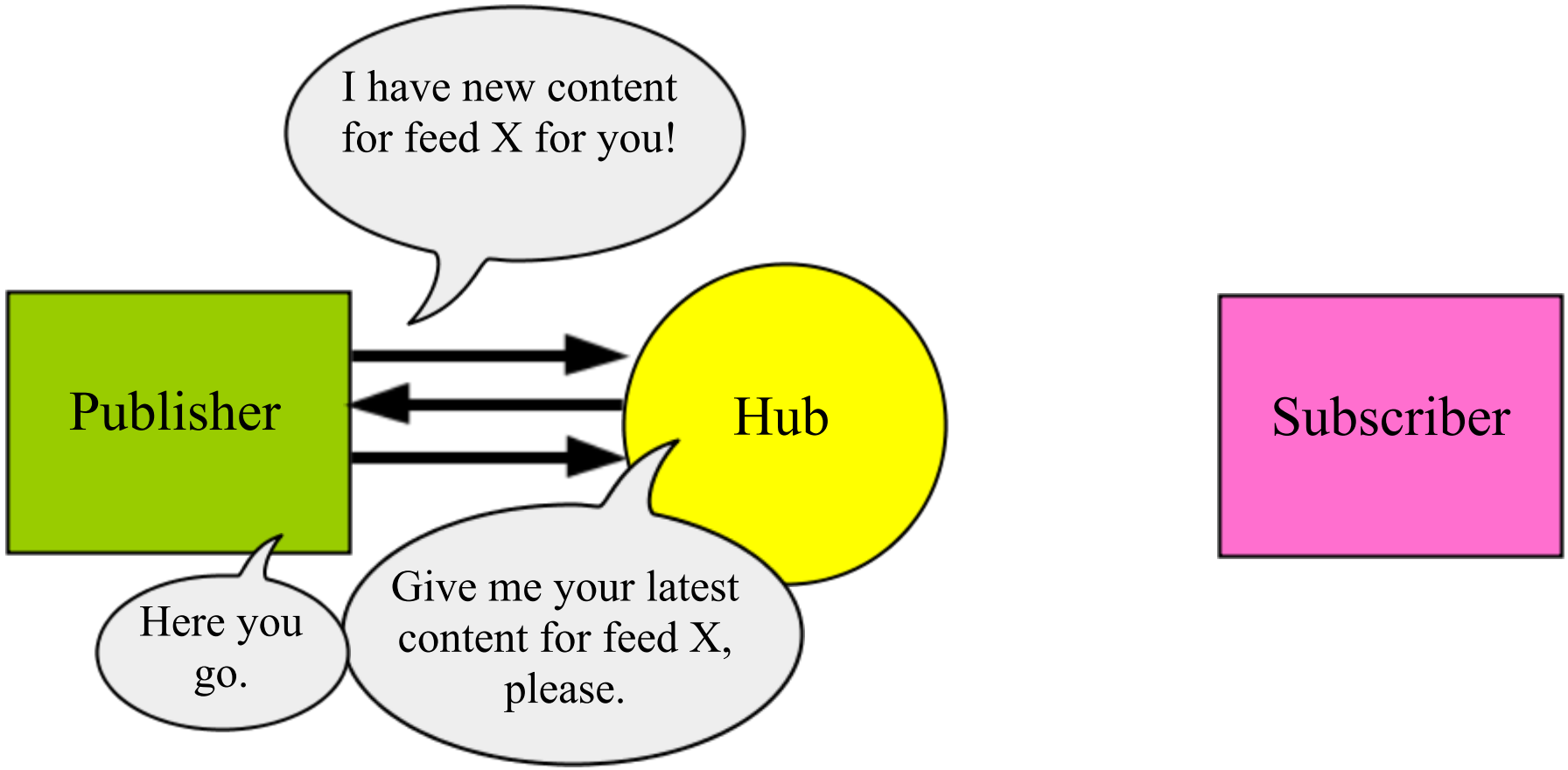
2. Subscriber POSTs subscription request to the Hub.



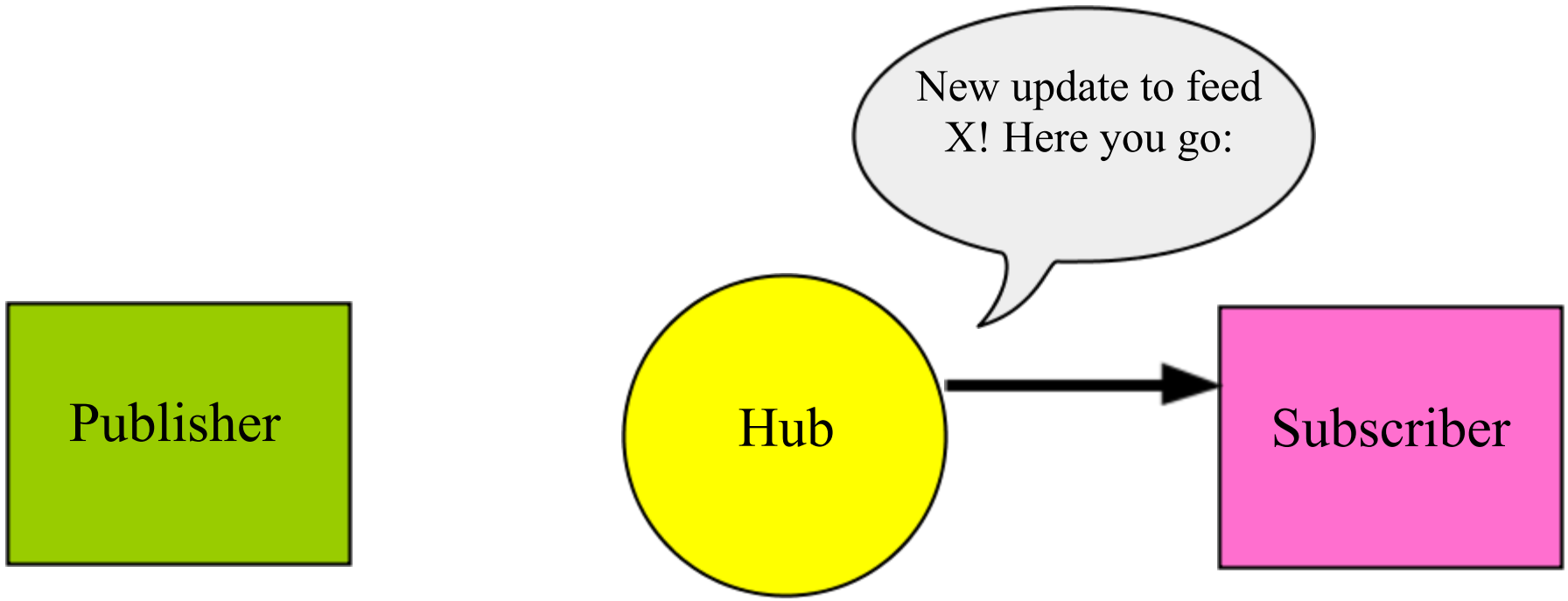
3. Hub POSTs to the endpoint URL to verify the request was authentic; Subscriber responds with confirmation to the Hub.



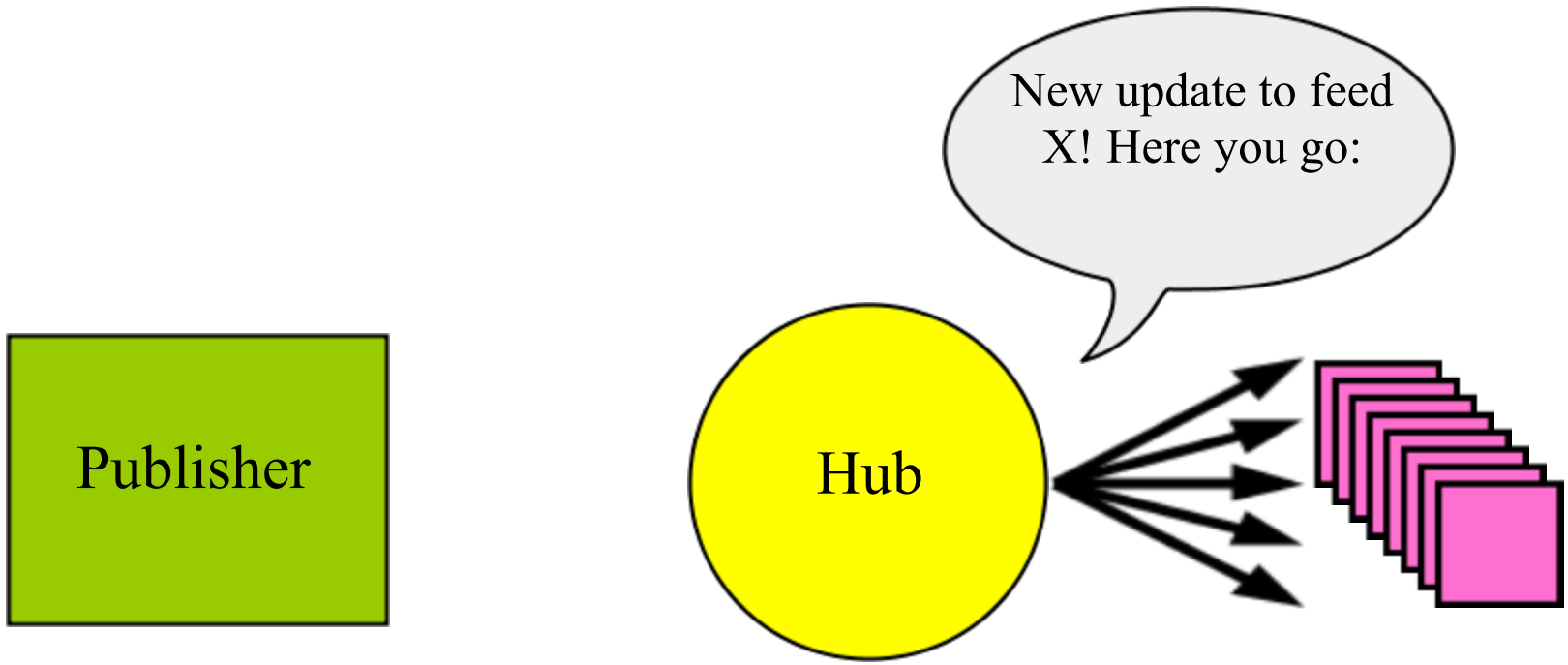
4. Publisher notifies Hub about updates by POSTing feed URLs to the Hub; Hub pulls the feed again to find new entries.



5. When Hub receives new update to feed X, it POSTs the update to the Subscriber's endpoint URL.



6. If feed X has multiple subscribers, the Hub sends updates to all of them. This reduces load on the Publisher.



Demo

Howto

How-to for Publishers

1. Add a declaration in your feed with your Hub(s) of choice

```
<link rel="hub"
  href="https://pubsubhubbub.appspot.com/"/>
```

2. Add something to your feed!

3. Send a ping to your Hubs with the feed URL

```
POST / HTTP/1.1
Content-Type: application/x-www-form-urlencoded
...

hub.mode=publish&hub.url=<your feed>
```

4. 204 response = Success, 4xx = Bad request, 5xx = Try again

How-to for Subscribers

1. Detect a Hub declaration in a feed
2. Send a subscribe request to the feed's Hub

```
POST / HTTP/1.1
Content-Type: application/x-www-form-urlencoded
...

hub.mode=subscribe&hub.verify=sync&
hub.topic=<feed URL>&hub.callback=<callback URL>
```

3. Hub will send a request to verify the subscription

```
GET /callback?hub.challenge=<random> HTTP/1.1

HTTP/1.1 200
...
<echo random>
```

How-to for Subscribers

Process new content from Hubs

```
POST /callback HTTP/1.1
Content-Type: application/atom+xml
...
<?xml version="1.0" encoding="utf-8"?>
<feed xmlns="http://www.w3.org/2005/Atom">
  <title>Awesome feed</title>
  <link rel="hub"
   href="http://pubsubhubbub.appspot.com"/>
  ...
  <entry>
    ...
  </entry>
</feed>
```

Building a hub

The role of the Hub

- Distinct functions
  - Accept and verify subscriptions to new topics
  - Receive pings from publishers, retrieve content
  - Extract new/updated items from feed
  - Send all subscribers the new content

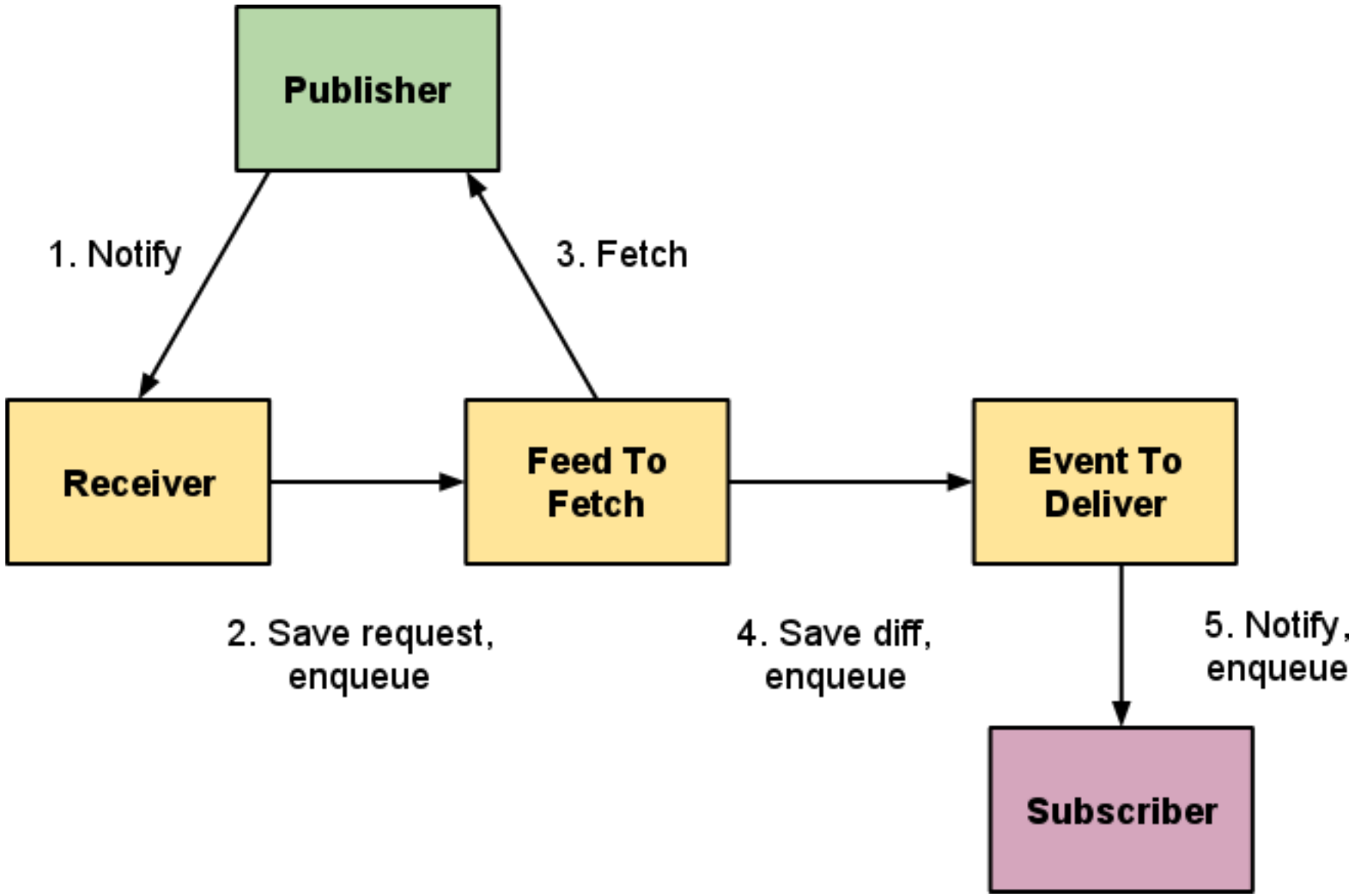
- Logical component
  - Publishers may be their own Hub
  - Combined Hub/Publisher has local speed-up

The role of the Hub

- Scalability
  - # of subscribers & feeds, update frequency
  - Delegation of content distribution (= bandwidth)

- Reliability
  - Retry fetch, delivery, idempotence

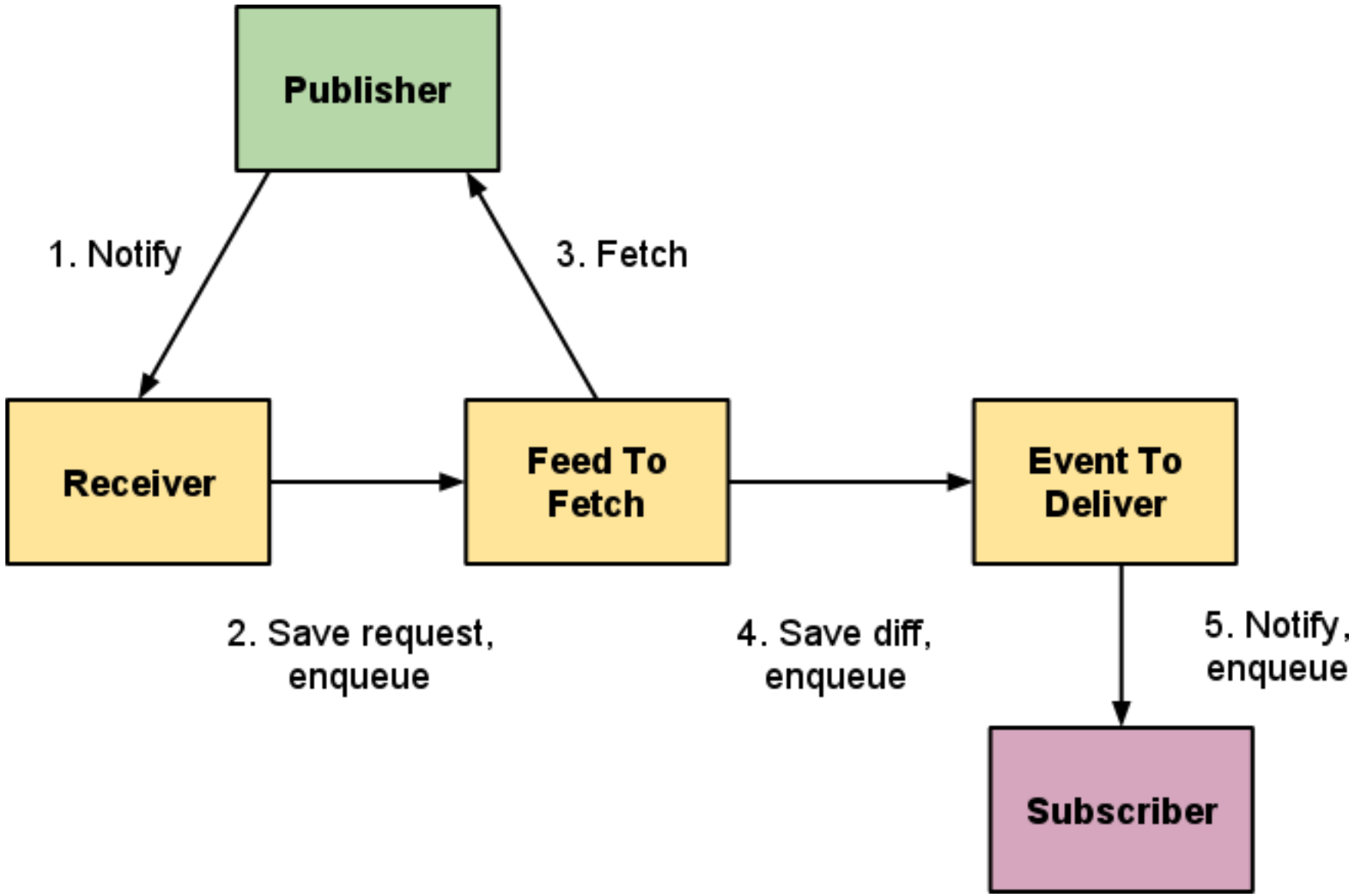
Pipeline


Publisher
1. Notify
3. Fetch
Receiver
Feed To Fetch
Event To Deliver
2. Save request, enqueue
4. Save diff, enqueue
5. Notify, enqueue
Subscriber

Receive publish events

1. Write a "feed to fetch" record
    - URL to update, TTL
    - Primary key = URL hash (DHT write)
2. Enqueue a task

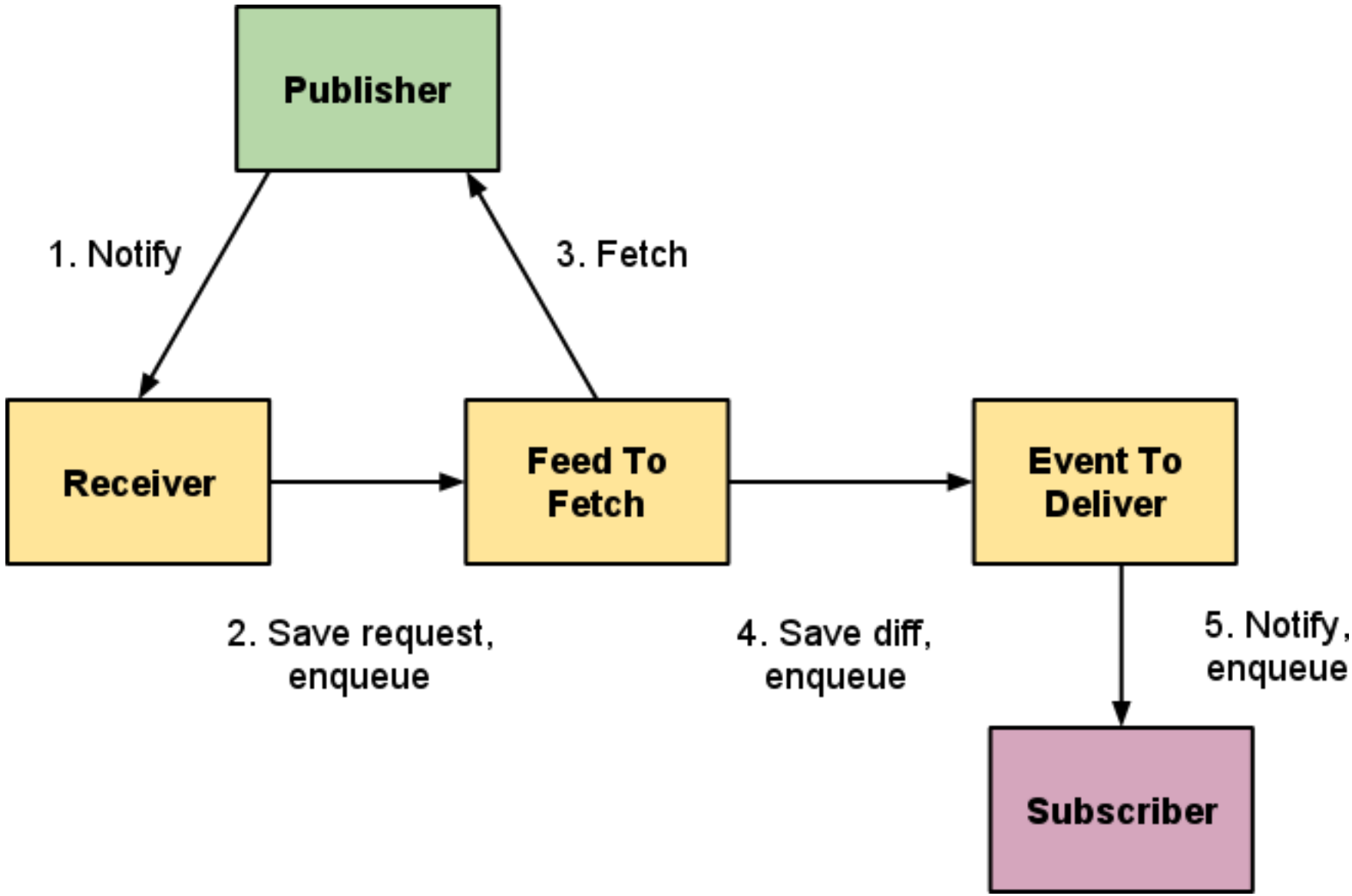
Pipeline


Publisher
1. Notify
3. Fetch
Receiver
Feed To Fetch
Event To Deliver
2. Save request, enqueue
4. Save diff, enqueue
5. Notify, enqueue
Subscriber

Fetch feeds

1. Fetch the feed
2. Parse its contents
    - SAX parser
3. Check for known feed entries
    - Primary key = Atom ID hash (DHT parallel lookup)
    - Diff by content hash
4. Save the diff to an "event to deliver" record
5. Enqueue a task

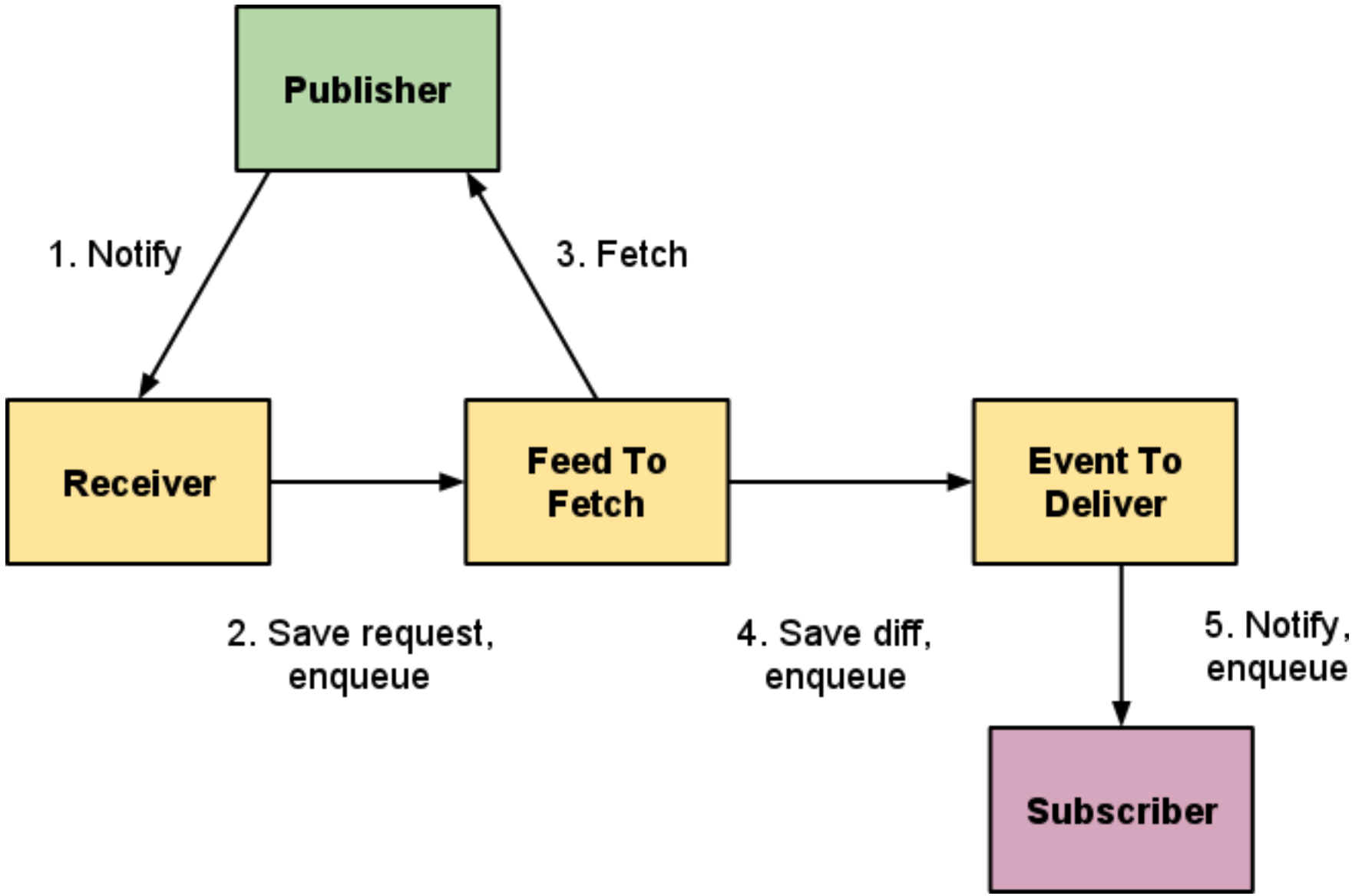
Pipeline


Publisher
1. Notify
3. Fetch
Receiver
Feed To Fetch
Event To Deliver
2. Save request, enqueue
4. Save diff, enqueue
5. Notify, enqueue
Subscriber

Deliver notifications

1. Query the list of "subscription" entities
    - Key-ordering = Bigtable linear scan
2. Send async URL fetch calls to N of them
3. Save the failures to the "event to deliver" record
    - Retry later
    - Debugging
4. Enqueue a task to handle next N

Pipeline


Publisher
1. Notify
3. Fetch
Receiver
Feed To Fetch
Event To Deliver
2. Save request, enqueue
4. Save diff, enqueue
5. Notify, enqueue
Subscriber

Implementation

- App Engine app
  - Each pipeline stage is a request handler
  - Stages run through one or more queues
  - Each stage is repeatable without harm (idempotent)

- Task queue API magic
  - Fast-path execution
  - Transactional tasks prevent drops

Transactional tasks

```
def txn():
  url = 'http://example.com/feed.xml'
  f = FeedToFetch(url=url)
  f.put()
  taskqueue.add(
    '/worker/fetch',
    params=dict(url=url),
    transactional=True)

db.run_in_transaction(txn)
```

Progress

Progress

- Over 100 Million feeds are PubSubHubbub-enabled
- Companies: Google, Posterous, FriendFeed (FB), livedoor, Six Apart, LiveJournal, LazyFeed, Superfeedr, MySpace, Tumblr, TwitterFeed, Netvibes, Cliqset, Gnip, Gawker...
- Google products: Buzz, FeedBurner, Blogger, Reader shared items, Google Alerts, Fastflip, ...

- Publisher clients: Perl, PHP, Python, Ruby, Java, Haskell, C#, MovableType, WordPress, Django, Zend, Drupal
- Active mailing list with 350+ members
- More publishers, subscribers, hubs, apps on the way

Project info

- Project page:
  - http://pubsubhubbub.googlecode.com
  - Full reference Hub source code with tests
  - Example publisher and subscriber apps

- Buzz API
  - http://code.google.com/apis/buzz