Personalization Service

Design Overview

Julian Villella
Software Developer, Mobile Team

Outline

- Legacy recommender
- What the 500px platform looks like
- The personalization service
  - Infrastructure
  - Model
  - View tracking
- The A/B test
- Next steps (product and cold start)
- Q/A

Once upon a time...

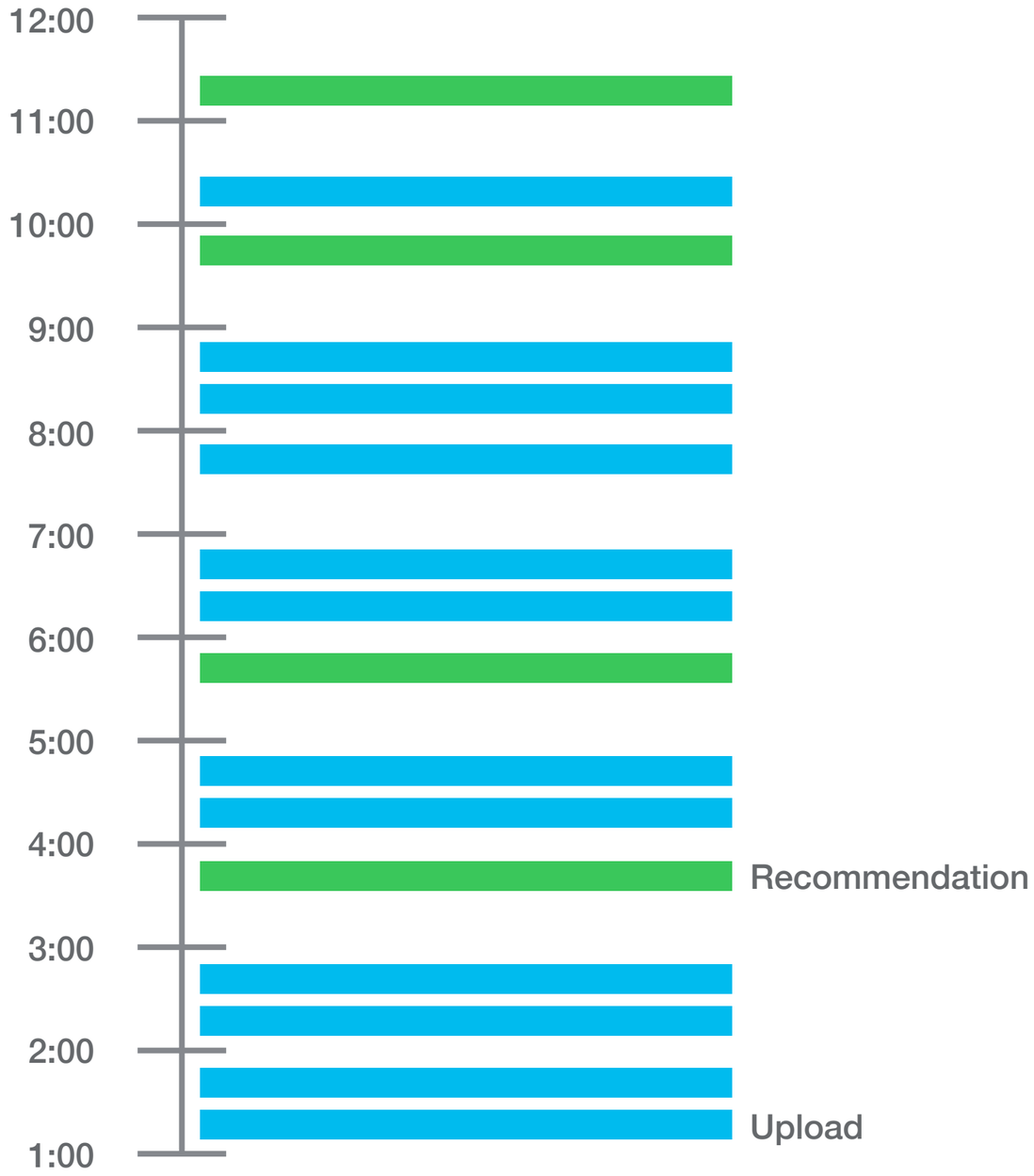
Life before we met Laval

Legacy Recommender

- Suggest photos your followee's liked
- Insert a recommendation when there is an hour gap between uploads

Timelines-based Feed Recommendations

- Query *Timelines-Service* for *like* events
- Filter out liked photos that exist in social graph
- Insert into hourly gaps (temporal merging)
- Hydrate photos, and return to client



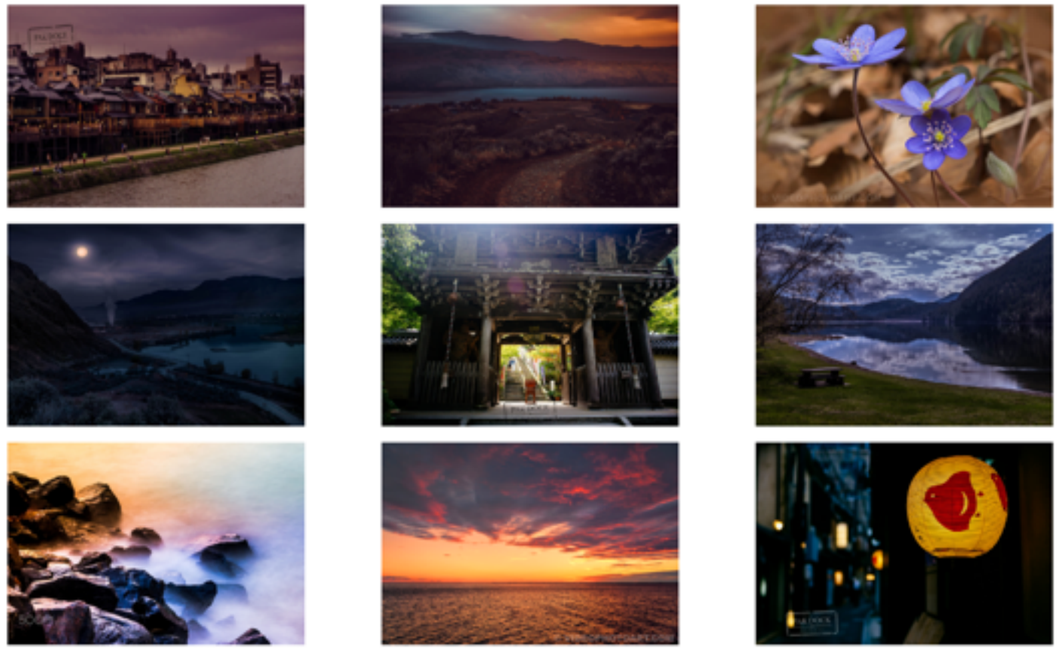
Legacy Recommender

Issues with this approach

- Incorrect assumption - people don’t necessarily like what their followee’s like
- Poor results
- Bad performance (hydration is slow)
- New users with few followers only got recommendations
- Users with many followers got no recommendations

Photos a user has liked

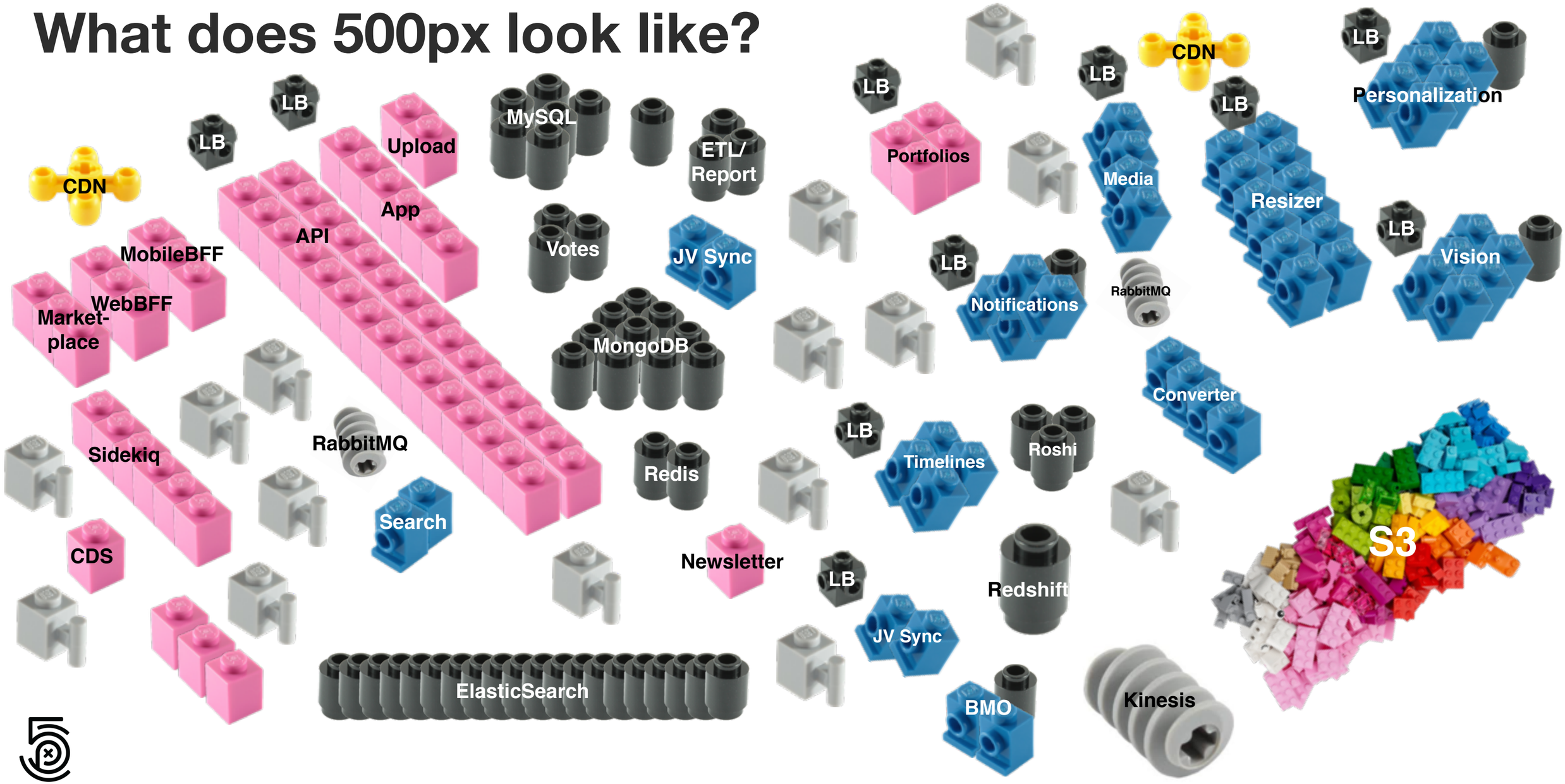
Recommendations for this user

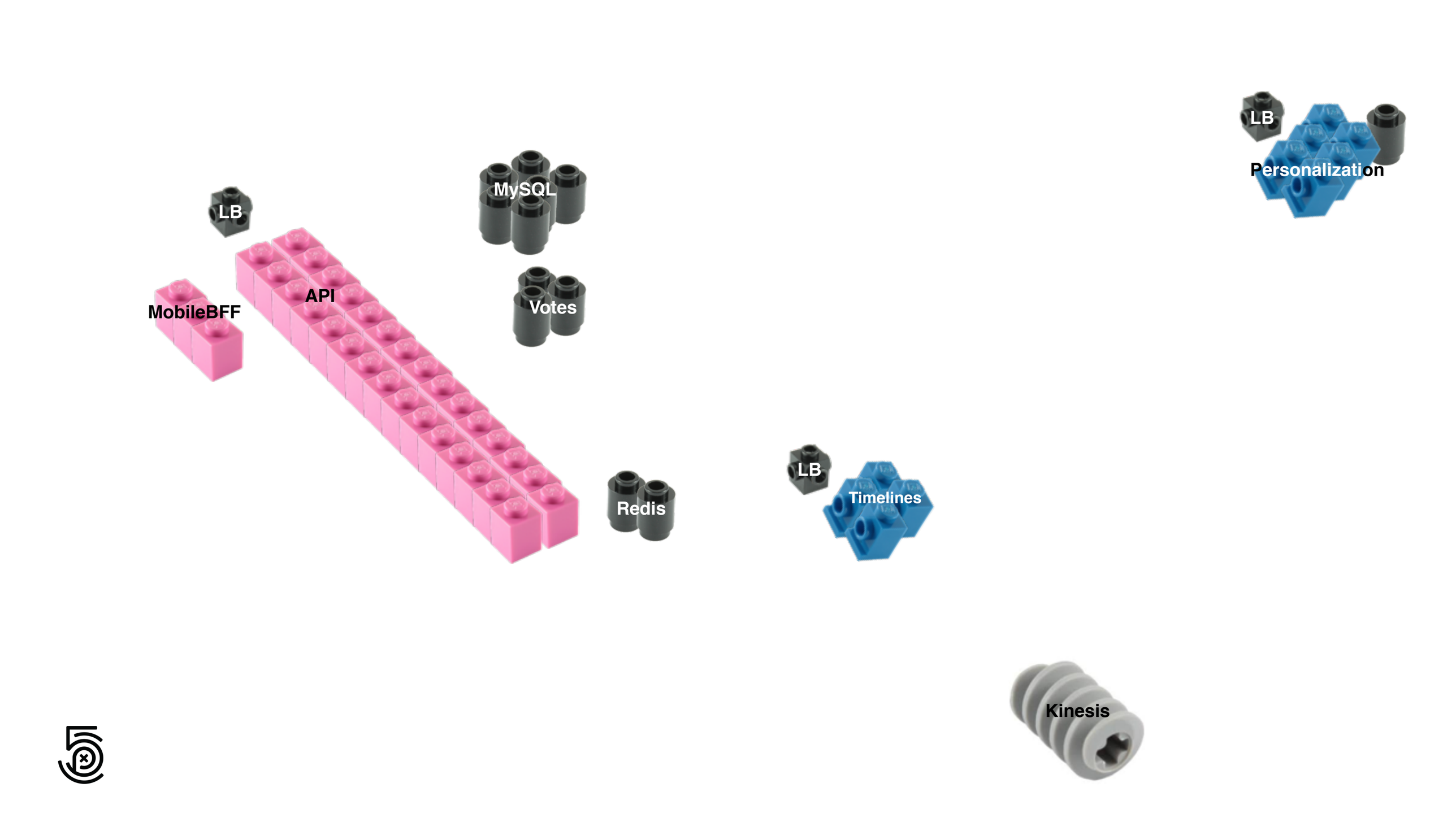


Fast-forwarding to today...

The 500px Platform Architecture

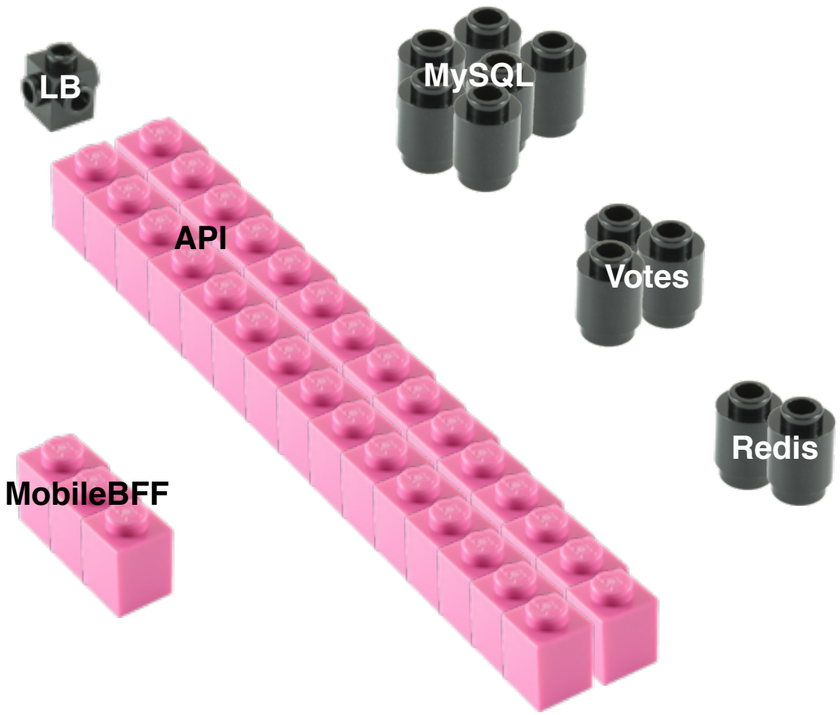
What does 500px look like?




LB
MySQL
LB
Personalization
API
MobileBFF
Votes
Redis
LB
Timelines
Kinesis

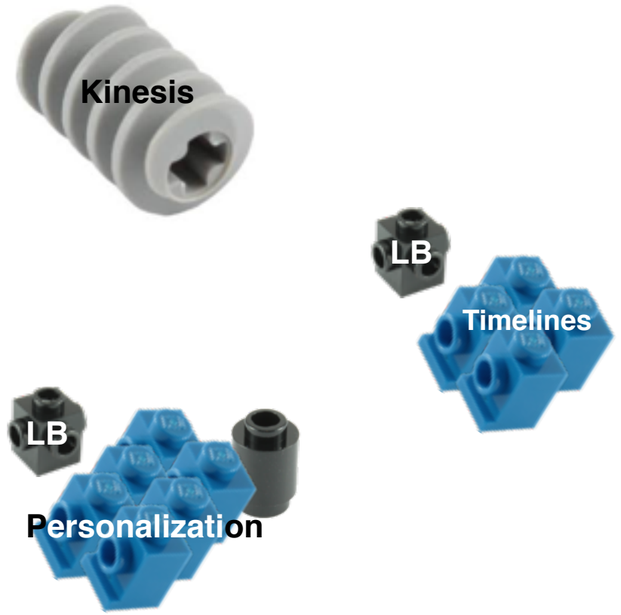
Let’s Build - Part I

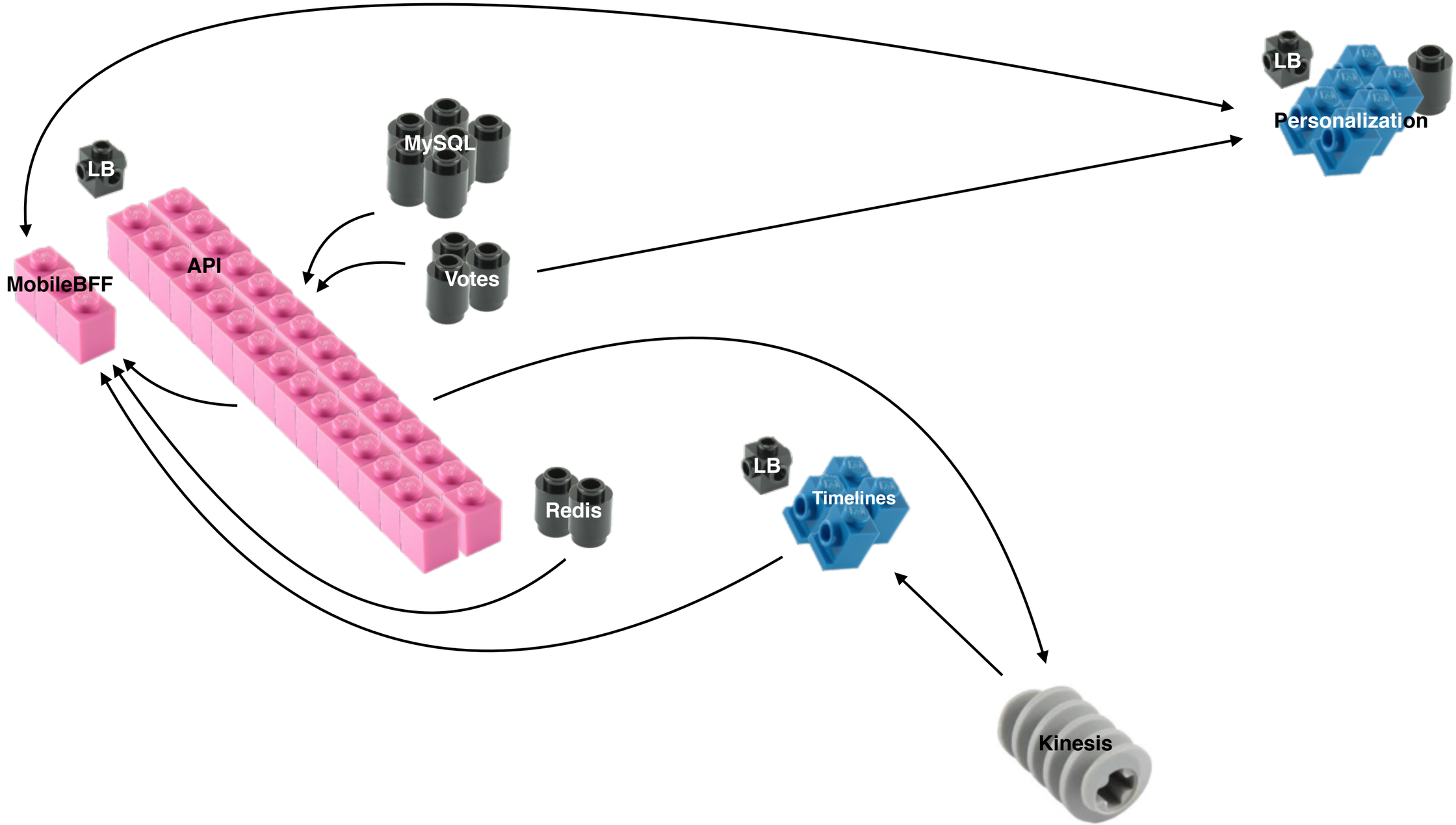
- **500px API**
  - Monolithic Rails application
- **MySQL**
  - 500px Platform storage (users, photo metadata, etc.)
- **Votes DB**
  - Separate database for scalability
- **Redis**
  - In-memory data structure store
  - For this project, used for rollout
- **Mobile BFF**
  - Presentation-layer for our Mobile Apps



Let’s Build - Part II

- **Kinesis Activity Stream**
  - API writes activity events (e.x. user liked photo)
  - Consumers read events they are interested in (e.x. notifications, timelines)
- **Timelines Service**
  - Exposes timeline events - likes, photos publishes, comments, etc.
- **Personalization Service**
  - Our new recommender exposed as an API




LB
Personalization
MySQL
LB
API
MobileBFF
Votes
LB
Timelines
Redis
Kinesis


The Personalization Service
Just for you

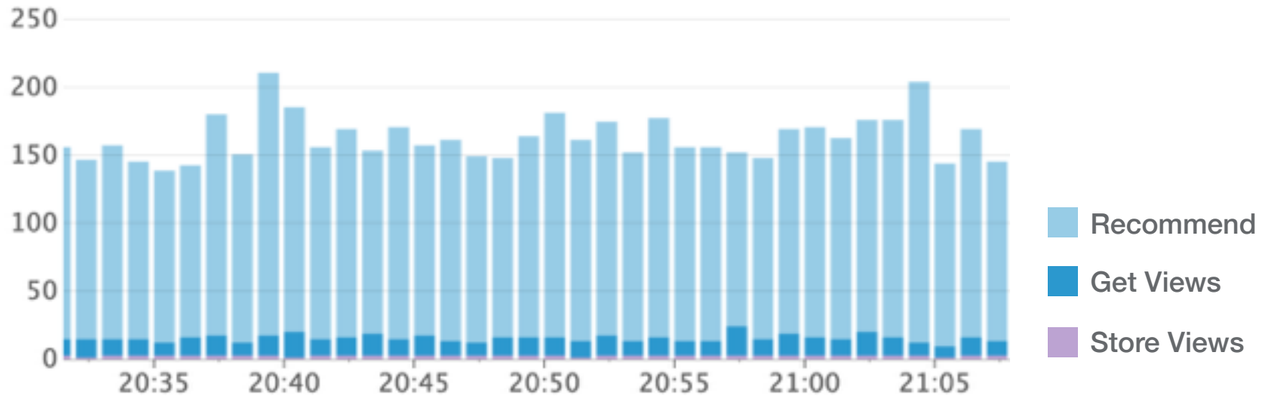
Infrastructure

- Flask app around the WALS/ALS recommender
- Provides recommendations for a given user
- Sits in AWS as an EC2 instance
- Clients (Mobile BFF) sit in OVH have to make HMAC signed requests
- View data is stored in ElastiCache as sorted sets

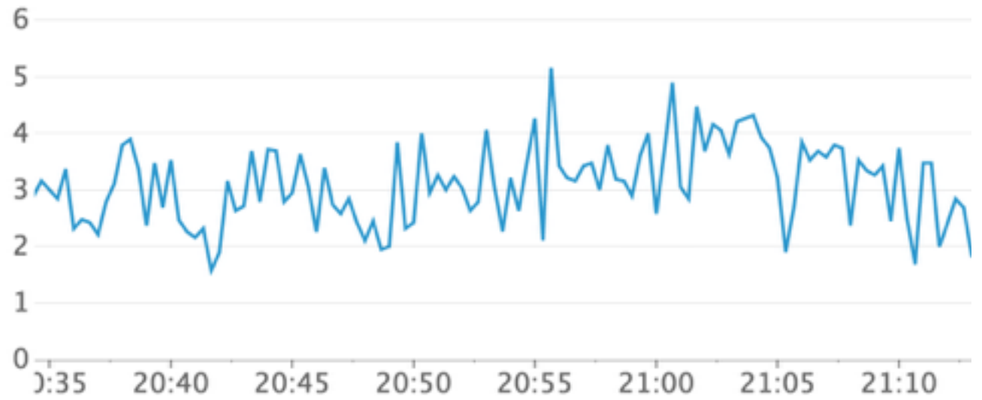
AWS EC2 Instance Types

	Box 1	Box 2	Trainer
Model	t2.medium	t2.medium	m4.xlarge
vCPU	2	2	4
Mem (GiB)	4	4	16

Request Duration Breakdown



Nginx Req/s



Model

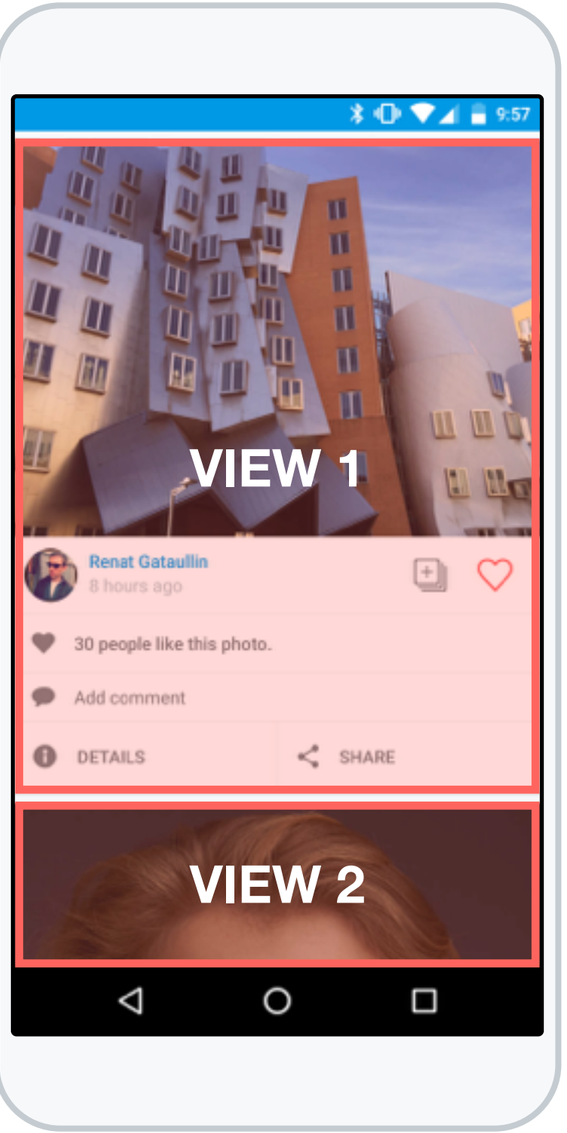
- Multi-model (WALS, and NWALS), exposed as,
  - `GET /photos/recommendations?recommender=<name>`

Training

- Spin up m4.xlarge (4 vCPU, 16GB) instance at midnight to perform training
  - Up for 2 hours, to allow training to complete
- Query votes and views
  - Minimum 10 votes per user
  - Drop spammers and overactive users (skew results)
  - Only count votes w/o corresponding unvotes
  - Minimum 120 votes per photo
  - Drop very popular photos (skew results)

View Tracking

- View data is stored in ElastiCache as sorted sets, exposed as,
  - **`POST /photos/views`**
- Redis' sorted set sorts by a user-defined *score.* We use the view's timestamp.
- Enforce 3 month limit (`expire`, `zremrangebyrank`)
- Client tracks photo views on the feed when a cell comes into view
- Posts those views to the Mobile BFF on the next feed request (ordering matters!)
- Mobile BFF forwards to Personalization Service



The A/B Test

Rolling out the new recommenders

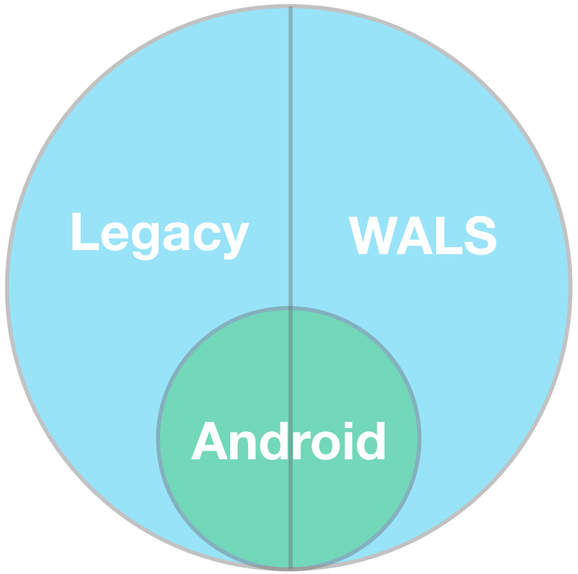
Controlled A/B Test

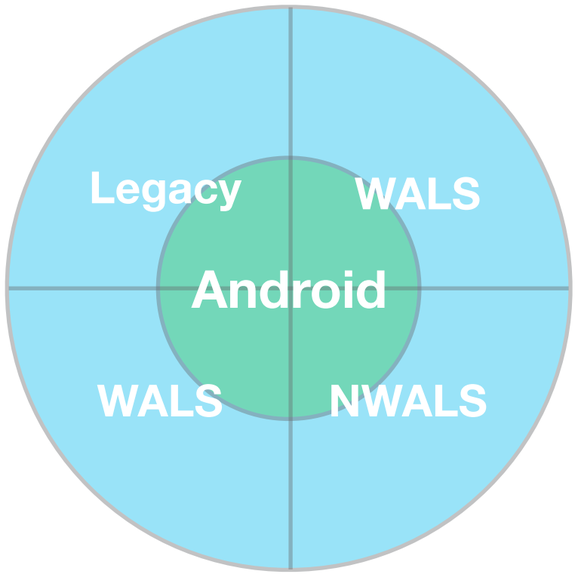
Preliminary Work

- Update legacy recommender to use fixed number of recommendations
- Update Android client to talk to new feed endpoint

Rollout

- Randomly sample users from recommender's vocabulary into buckets stored in redis
  - Repeat over period of time as vocabulary changes (currently manual)
- Mobile BFF talks to redis to determine which recommender to use for client request



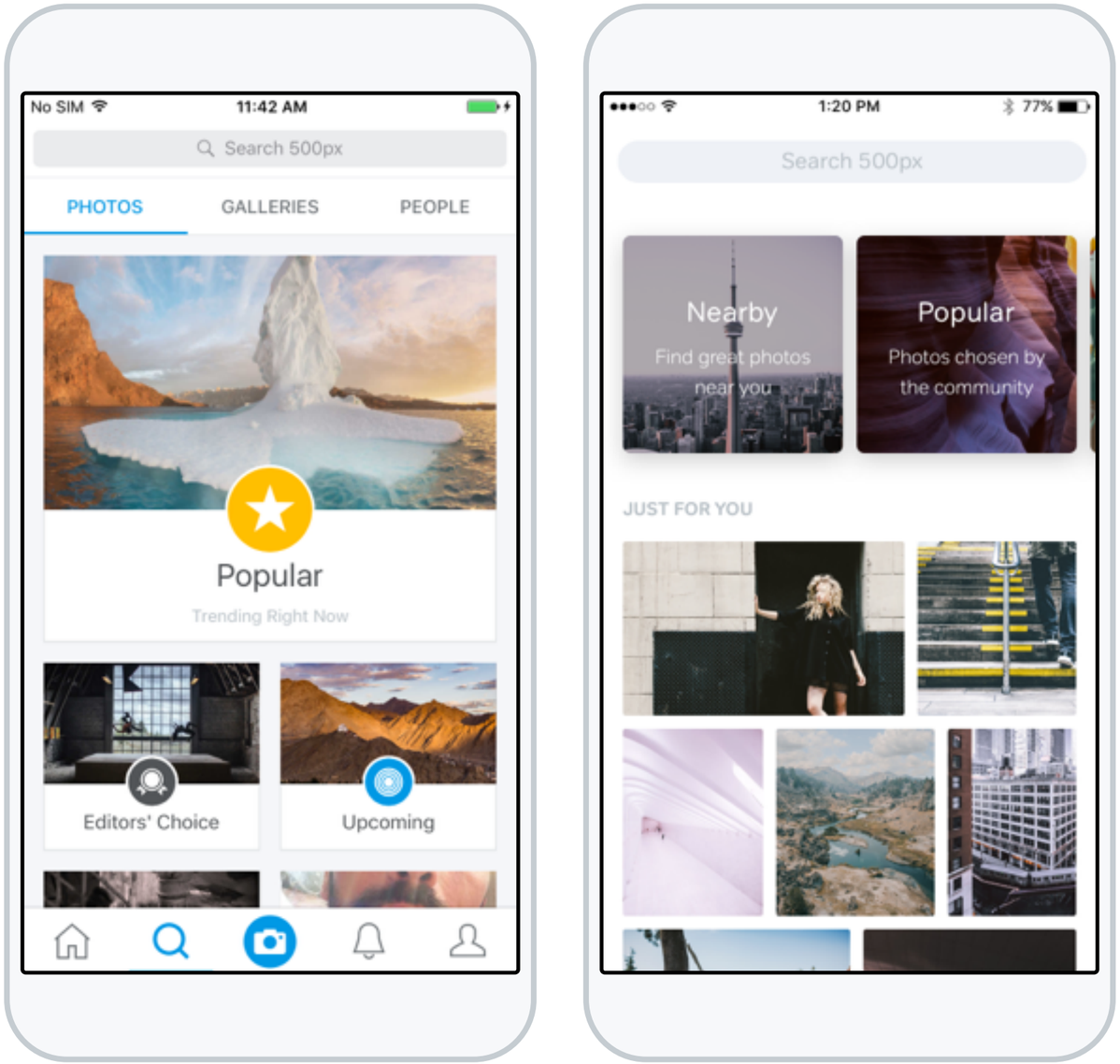


Next Steps

So…what's the plan?

Next Steps: Product

- Finish A/B test and rollout chosen recommender to all users in its vocabulary
- Extend to iOS (currently in App Store Review process)
- Extend to the web which currently has no feed recommendations
- Experiment usage outside the home feed
  - E.x. revamping discover - currently everyone sees the same set of photos



Prototype

Next Steps: Cold Start

- Update onboarding to collect positive and negative signals
- Fold in new users after onboarding
- If user skips, can show popular images
  - Until we have enough votes to recommend