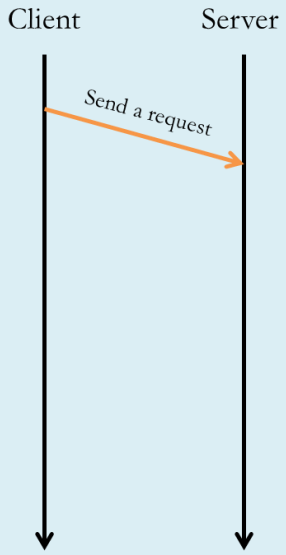
Asynchronous architectures

- Examples of asynchronous architectures
- Common design options
- Push notifications and iOS / Android
- Queueing and Twitter (aka X)

Some requests don't need a response

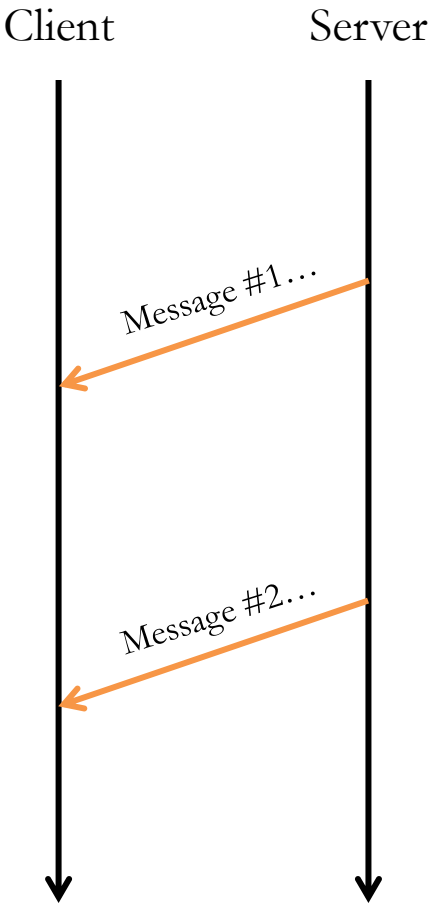
- **Sending email / txt msg**
- **Posting to social media**
- **Likes**



The server performs some function asynchronously…

Asynchronous architectures

- **When the server wants to initiate communication**
  - *Notifications on your phone…*
  - *Uber: your car has arrived*
  - *GrubHub: your food has arrived*
  - *Amazon: your package has shipped*
- **Common use cases:**
  - *Texting*
  - *Messaging apps*
  - *Notification systems*
  - *Workflow systems (documents, food delivery)*
  - *Streaming services (music, audiobooks, videos)*

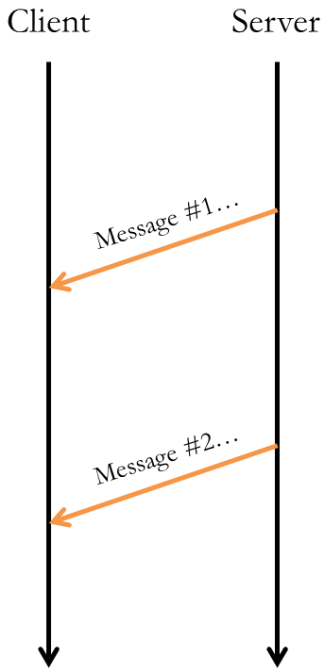


Servers notifying people

- Many services notify users by sending email or txt

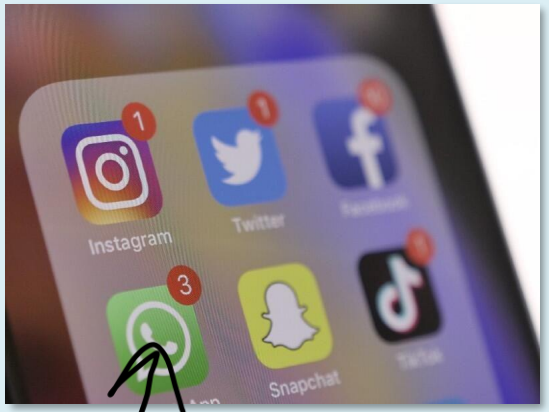
- Send email by connecting to an SMTP server
  - **AWS SES** (simple email service)
- Send SMS messages using a txting service
  - **Twilio** or **AWS SNS** (simple notification service)

- Does not work well for notifying **apps**…



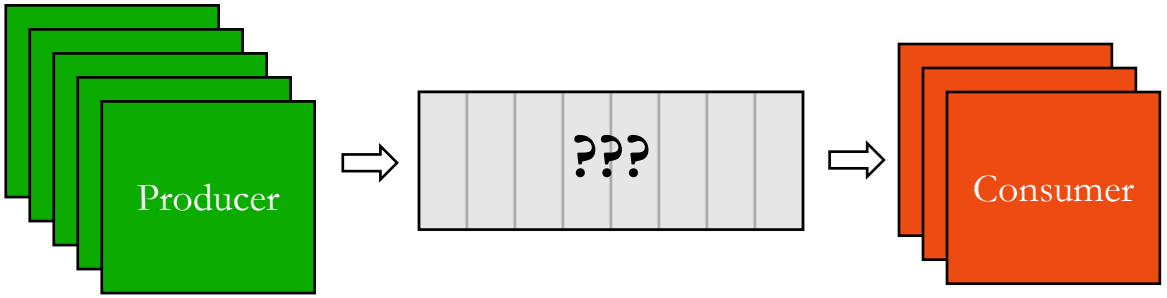
Servers notifying apps

- **Most notifications are targeting the apps on your phone**
  - *Messages, WhatsApp, Instagram, Snapchat, TikTok, Twitter/X, …*



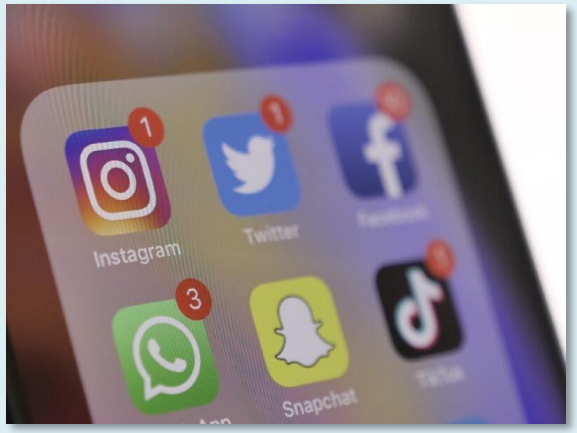
Decoupling enables scaling

Asynchronous systems decouple "producers" from "consumers"

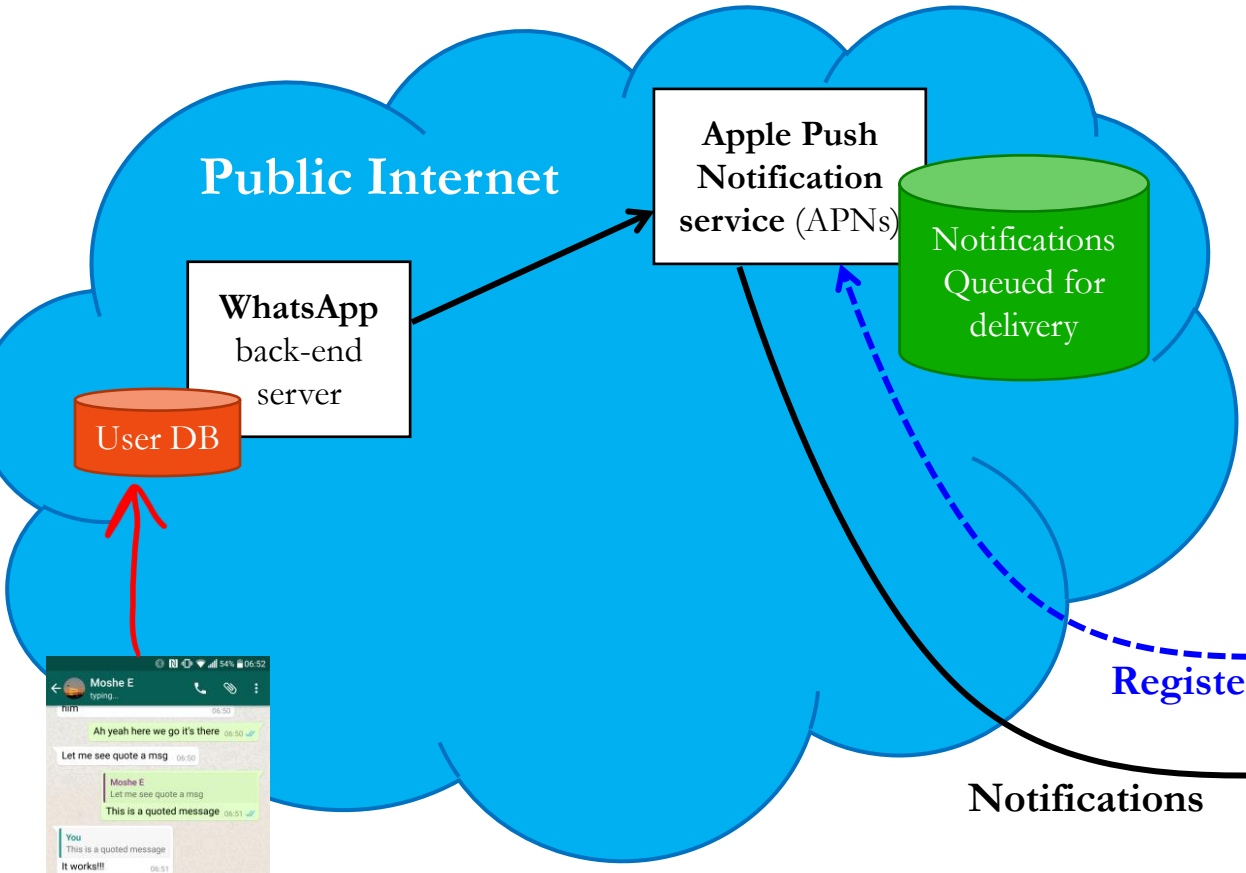


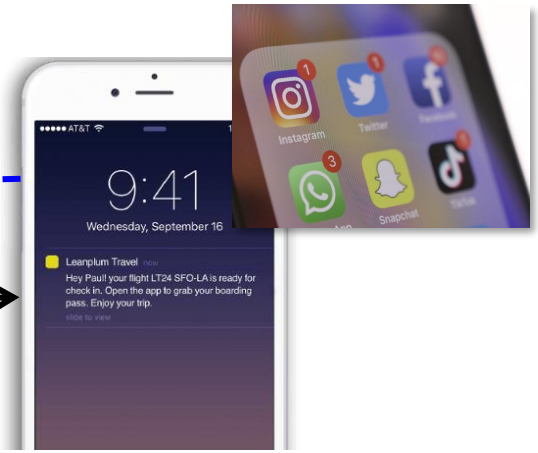
Push notifications

- **Push notifications are a great way for servers / systems to reach us…**



Push Notification Service



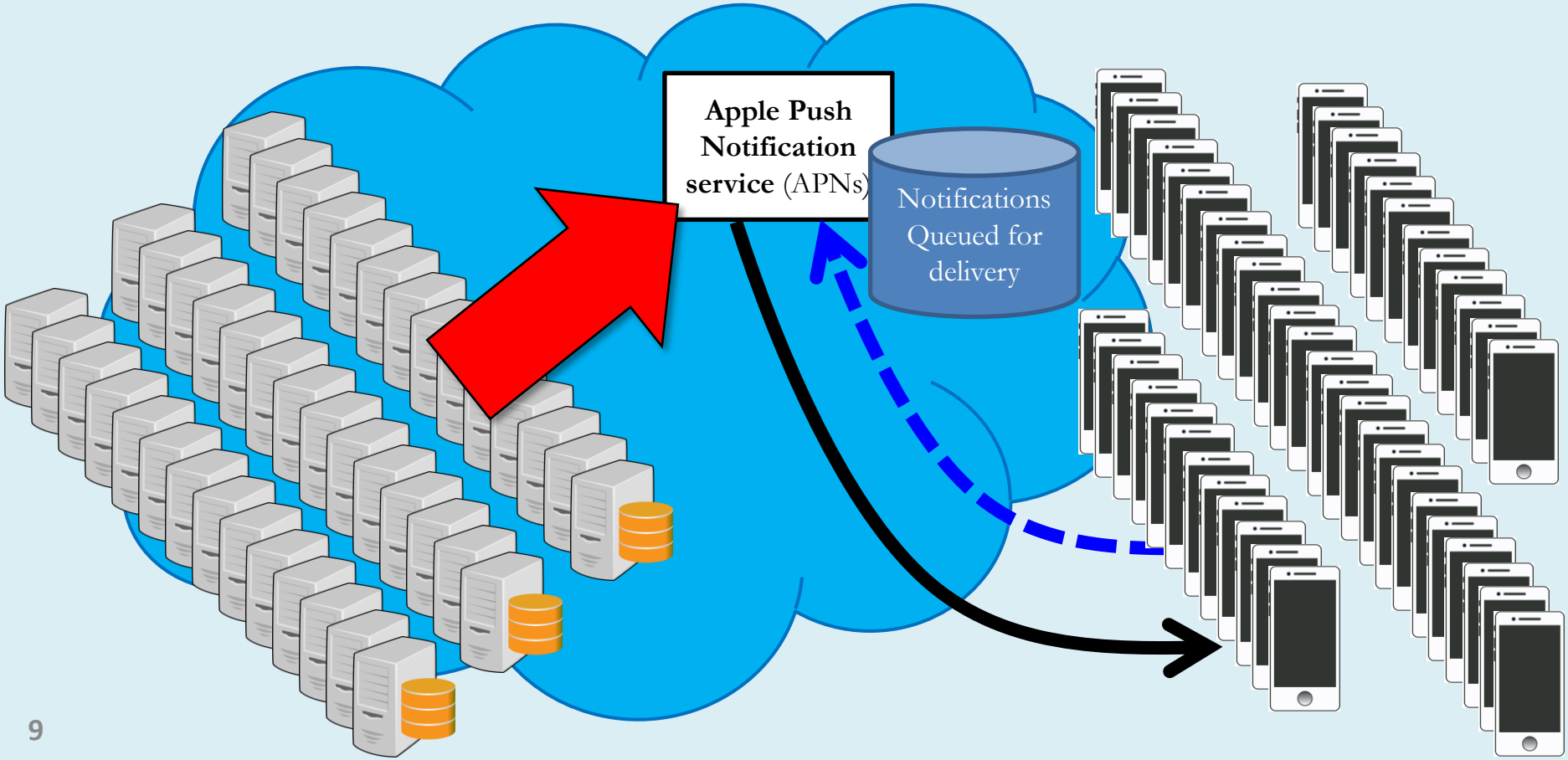


Key points:

- Client OS registers one connection for all apps
- App servers send notifications to device through Apple (iOS) or Google (Android) notification service
- Completely asynchronous

Think about the scale…

- Thousands of producers
- Producing billions of notifications
- Delivered to billions of consumers

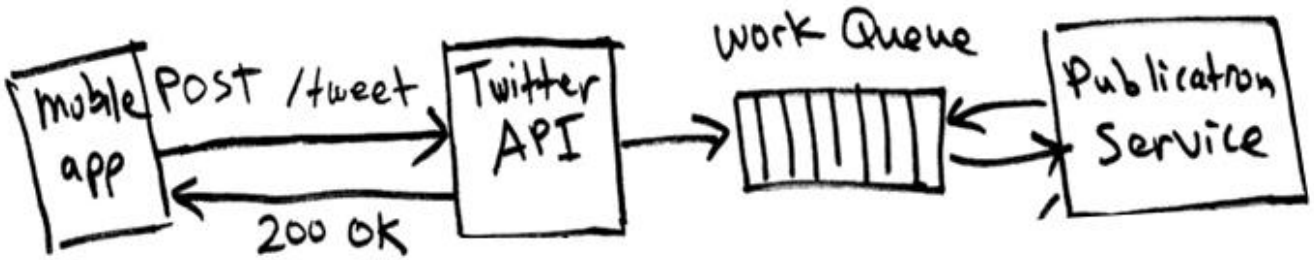


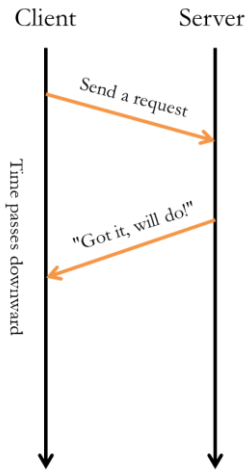
Design options

- **Classis async architectures use queueing**
  - *AWS **SQS** == simple queueing service*
- **Other options:**
  - *Use a **database** to store / retrieve notifications*
  - ***SNS** == simple notification service (real-time, e.g. chat msgs)*
  - ***S3** / **DynamoDB** / **Kinesis** == predefined event-based triggers*
  - ***EventBridge** == custom event service*
  - ***Step Functions** == build workflows with Lambda functions*

Message queues

- Message queues store requests that need to be processed
- Client calls server --- server enqueues a work message
- Another server dequeues message & processes





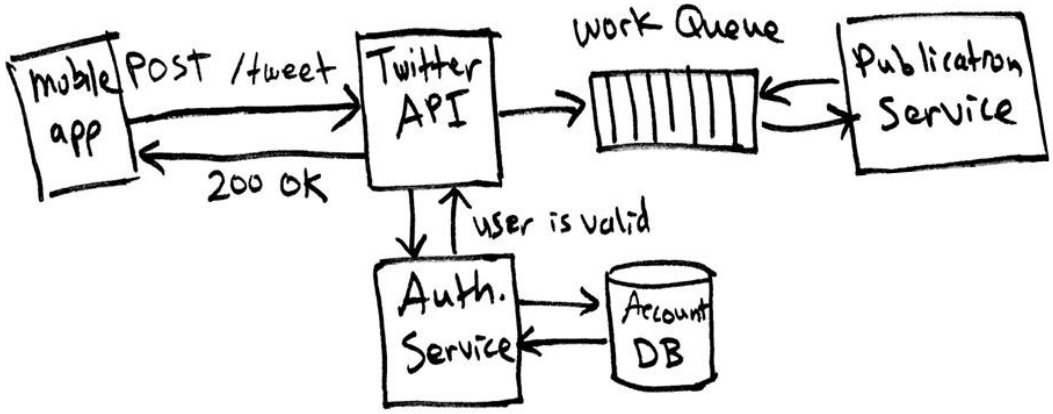
Queueing scales

- Message queues are a simple kind of database – store work requests
- Queues *smooth* demand peaks by **decoupling** producers & consumers
- Many producers and many consumers can connect to a queue

- AWS **SQS** == Simple Queueing Service

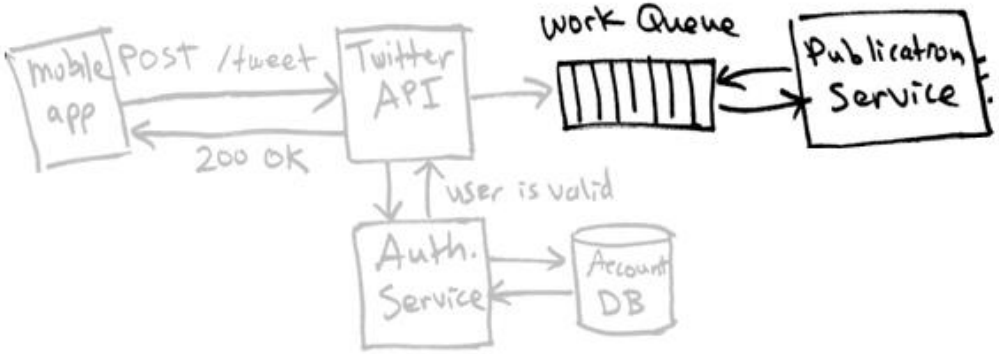
Example: Twitter (aka X)

1. Client posts tweet



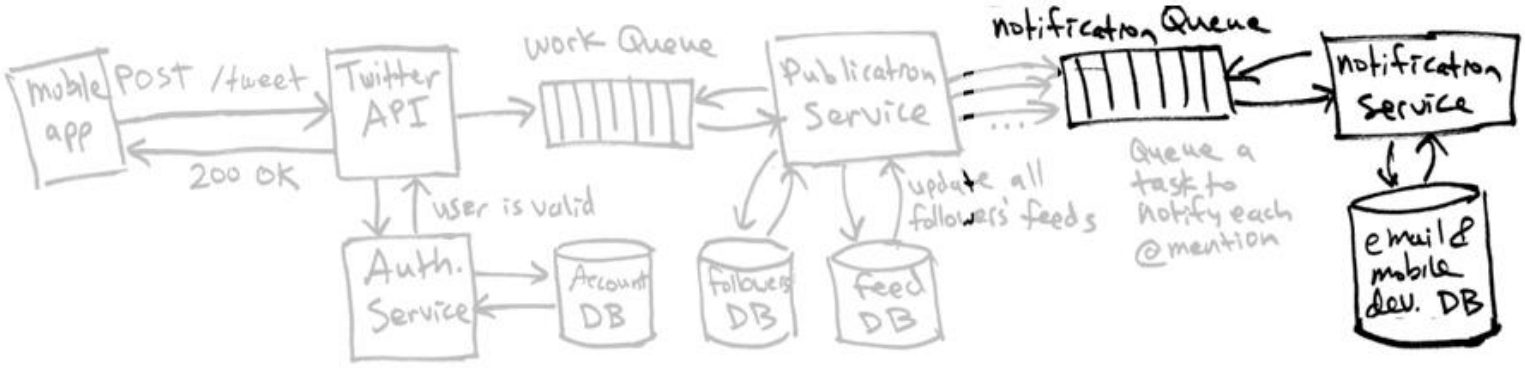
- API service receives request
- Puts a tweet-creation job on a queue
- Sends "acknowledgement" response back to user

2. Publication service handles the post



- Publication service fetches message
- Posts tweet
- Gets a list of followers to receive the tweet
- Adds the new tweet to all the followers' feeds

3. Notification service alerts users



- There may be millions of notifications in the queue based on the original tweet
- Notification service dequeues each one and notifies user
- What happens if there is a failure?
  - Retry a few times, and then give up --- the original tweeter does not care

That's it, thank you!