Project in Parallel/Distributed Programming (236371)

EVENTSHARE – SOCIAL EVENTS SHARING

Students: Arieh Leviev
Tivan Magal

Supervisor: Prof. Roy Friedman
Mr. Alex Libov

Project Objectives

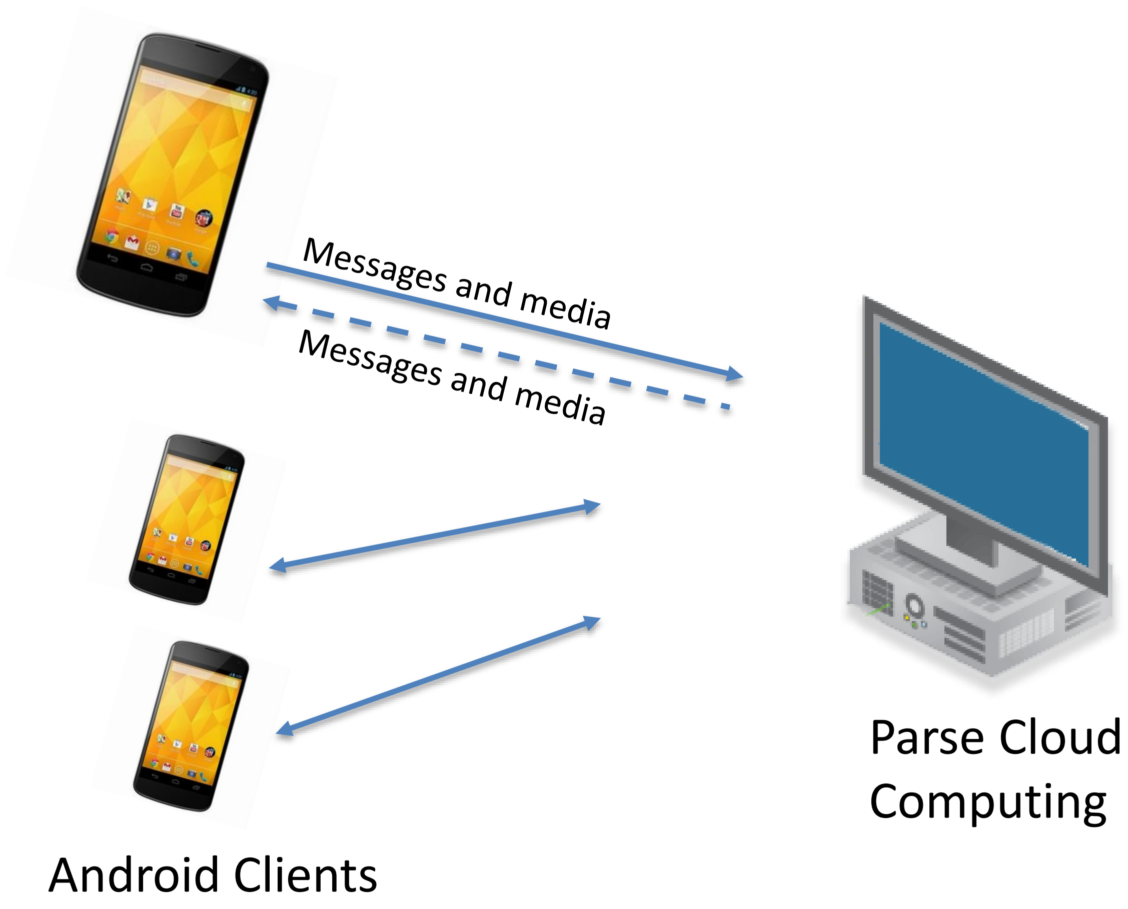
- Developing android application which establish an easy to use infrastructure for reporting and finding events such as: parties, meet-ups, trips, etc.

- This requires:
  - Chat messaging mechanism
  - Querying existing chatrooms – maintenance of a DB

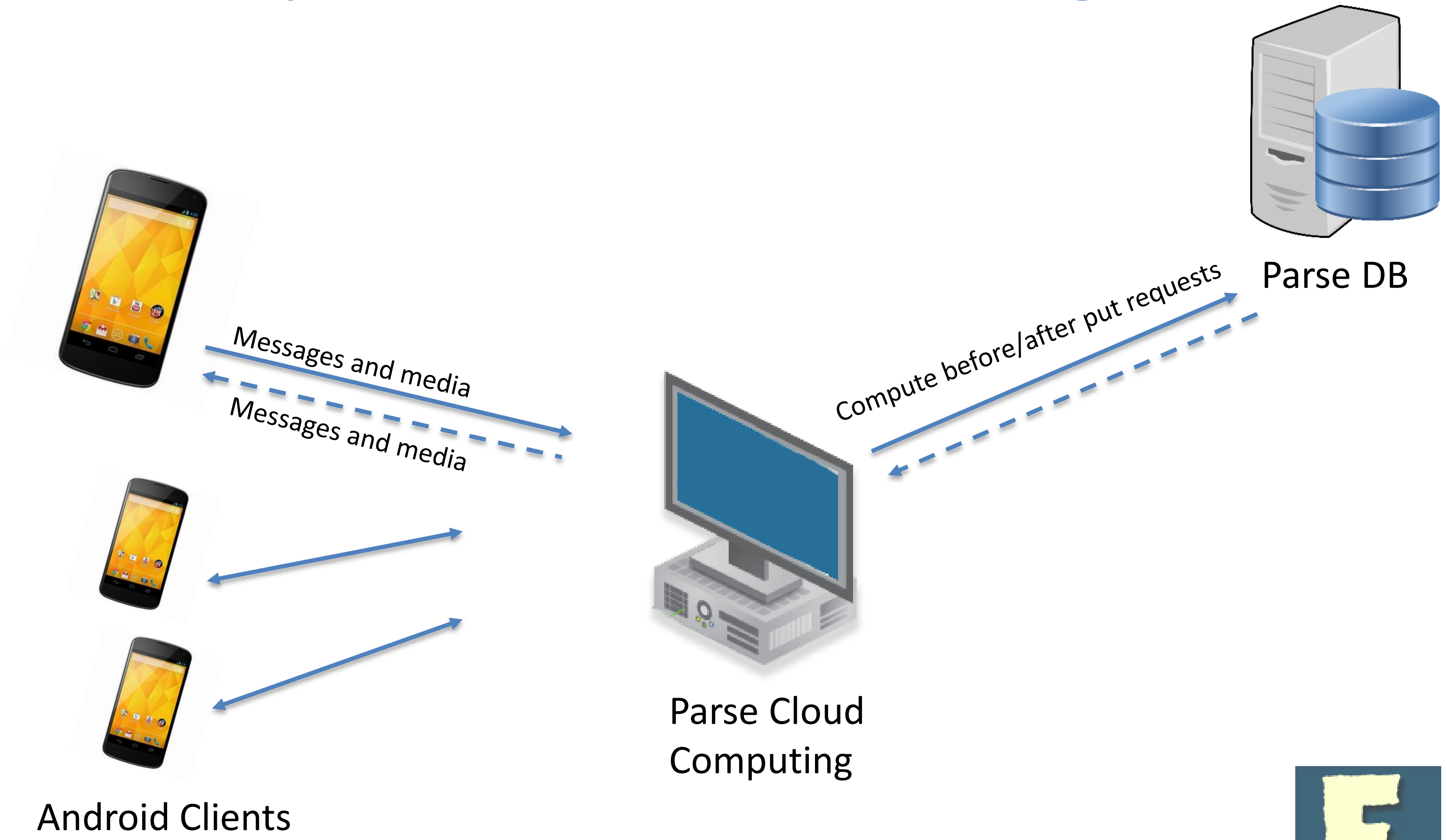
System Communication Diagram

Android Clients

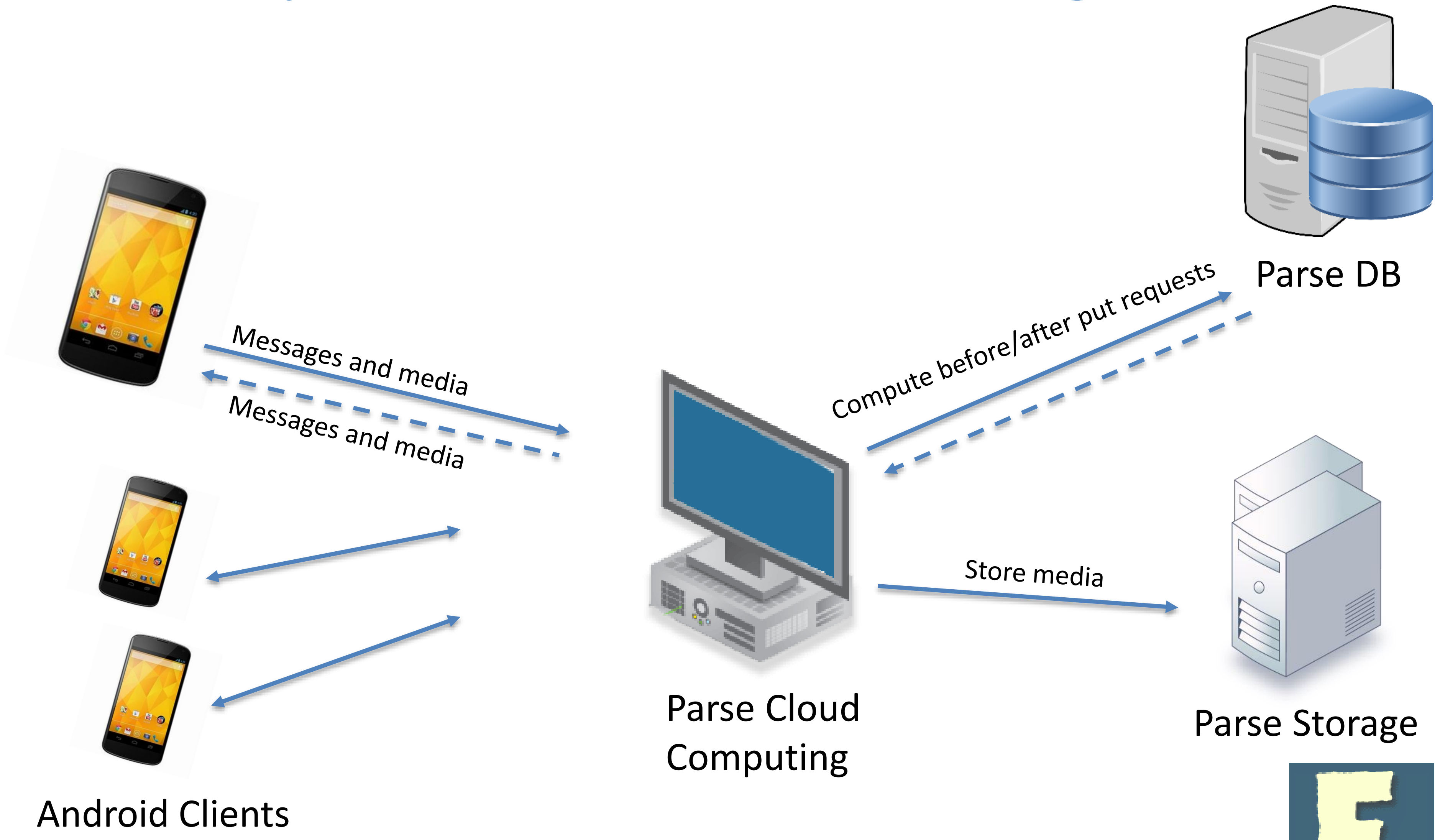
System Communication Diagram



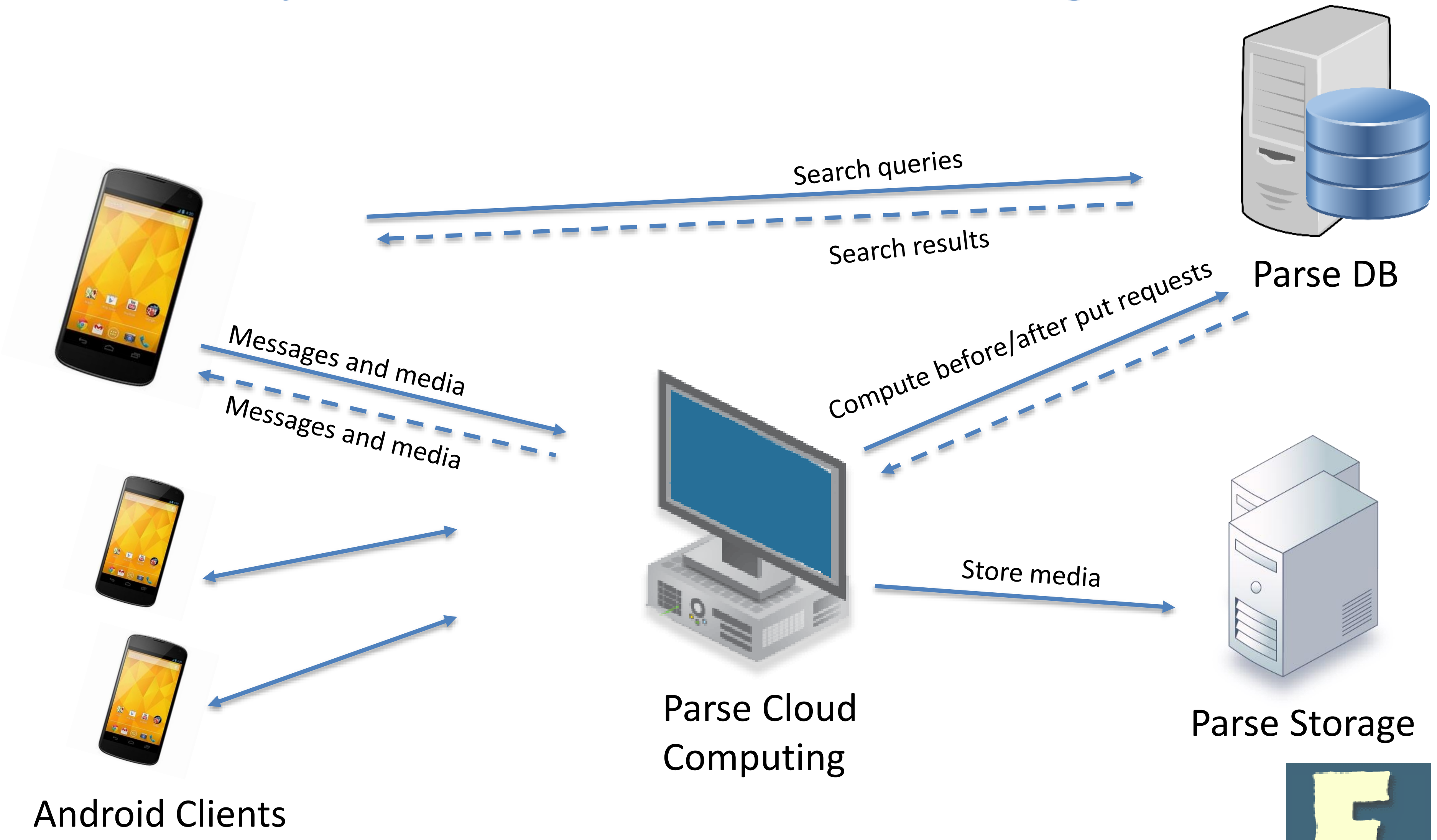
System Communication Diagram



System Communication Diagram



System Communication Diagram



Parse Services

Parse

Parse Services

- Database
- Cloud Computing
- Push Notifications
- Storage

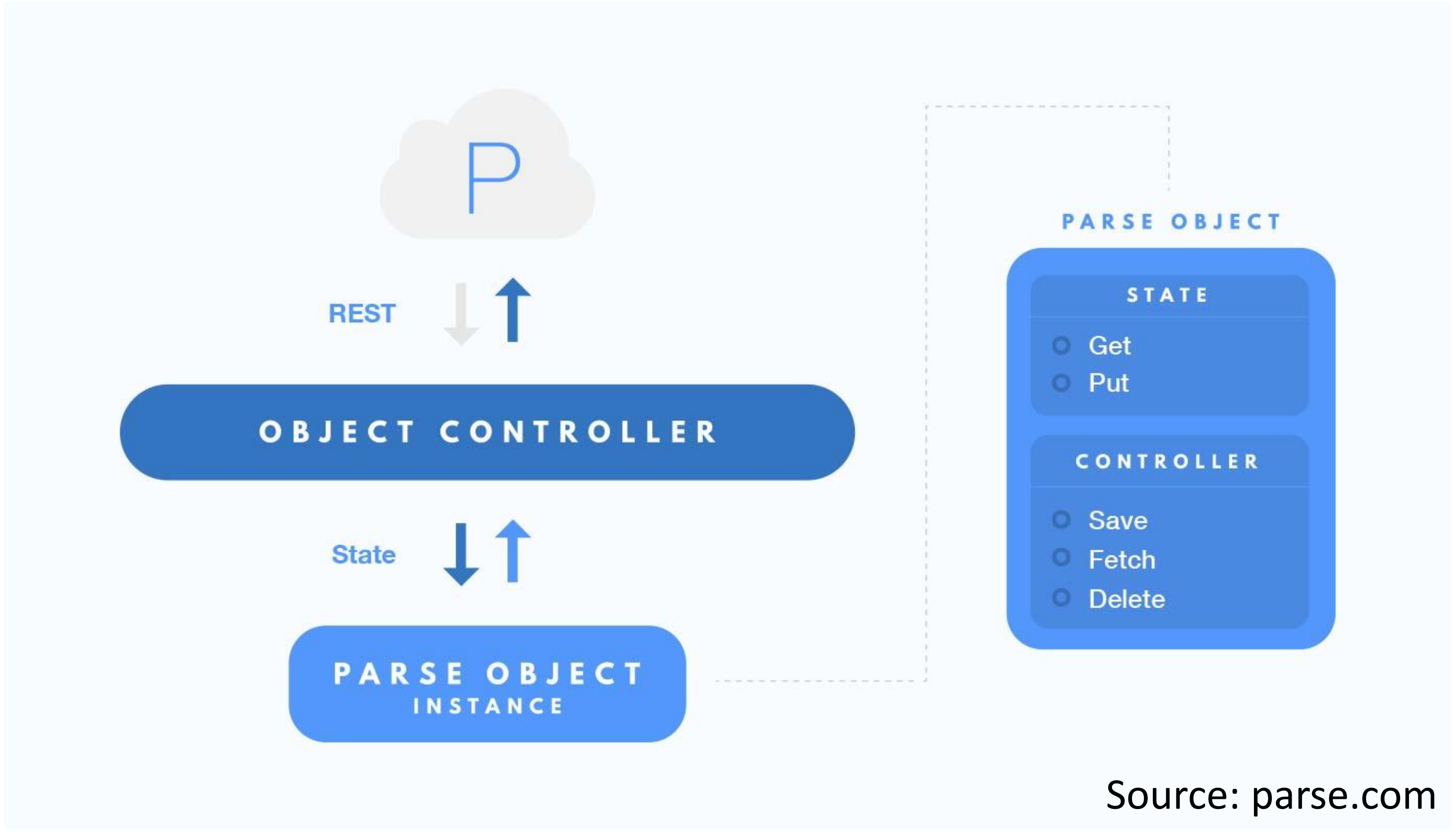
- Parse SDK

Parse SDK

As a “decoupled architecture”, database entities are encapsulated in a special object for each SDK Parse support - iOS, Android, JS, PHP, .net, Windows Phone and more.

For all SDKs, routines invoke remote parse routines via REST api.

Parse Database - Decoupled Architecture



Source: parse.com

Parse Database

- Relational database
- Row store
- Allows to store object pointers, and include them when fetching (join op)

Parse Cloud Code

- JavaScript SDK for Parse, based on the popular Backbone.js framework
- Easy deployment, no dependencies needed
- Allows RPC for custom defined methods
- Allows before/after objects save hooks

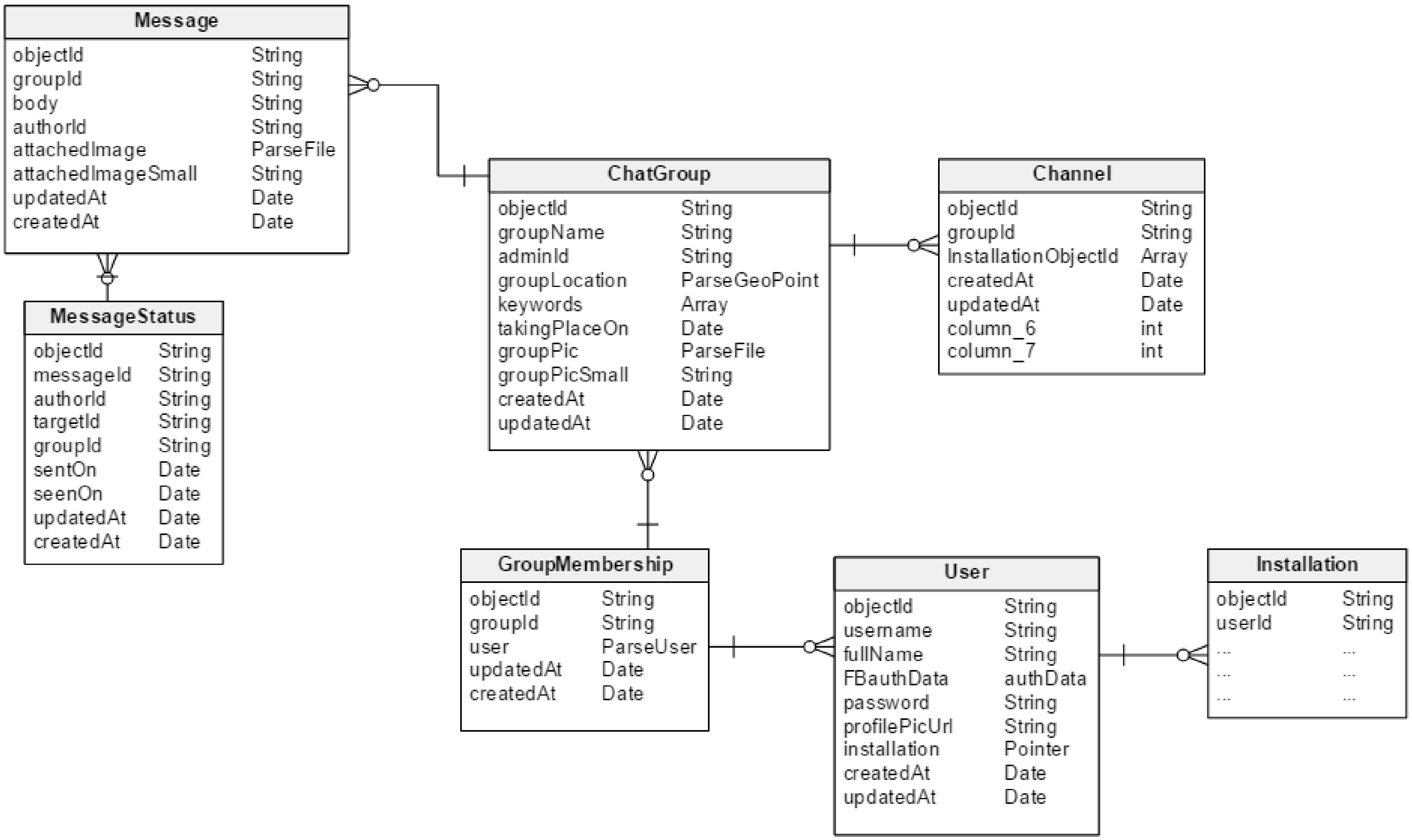
Parse Storage

- File entities are encapsulated with *ParseFile* object, holding its remote url on parse storage and several other attributes.
- The controller defines methods for async upload/download.

Objects persistence and Local Datastore

- “saveEventually” – object persist request from client. If the client doesn’t have a network connection, the object will be stored safely on the device until a new connection has been established (lies below retries with exp. backoff).
- Parse local datastore - a local datastore which can be used to store and retrieve ParseObjects, even when the network is unavailable.

Database Schema



Message	
objectId	String
groupId	String
body	String
authorId	String
attachedImage	ParseFile
attachedImageSmall	String
updatedAt	Date
createdAt	Date

MessageStatus	
objectId	String
messageId	String
authorId	String
targetId	String
groupId	String
sentOn	Date
seenOn	Date
updatedAt	Date
createdAt	Date

ChatGroup	
objectId	String
groupName	String
adminId	String
groupLocation	ParseGeoPoint
keywords	Array
takingPlaceOn	Date
groupPic	ParseFile
groupPicSmall	String
createdAt	Date
updatedAt	Date

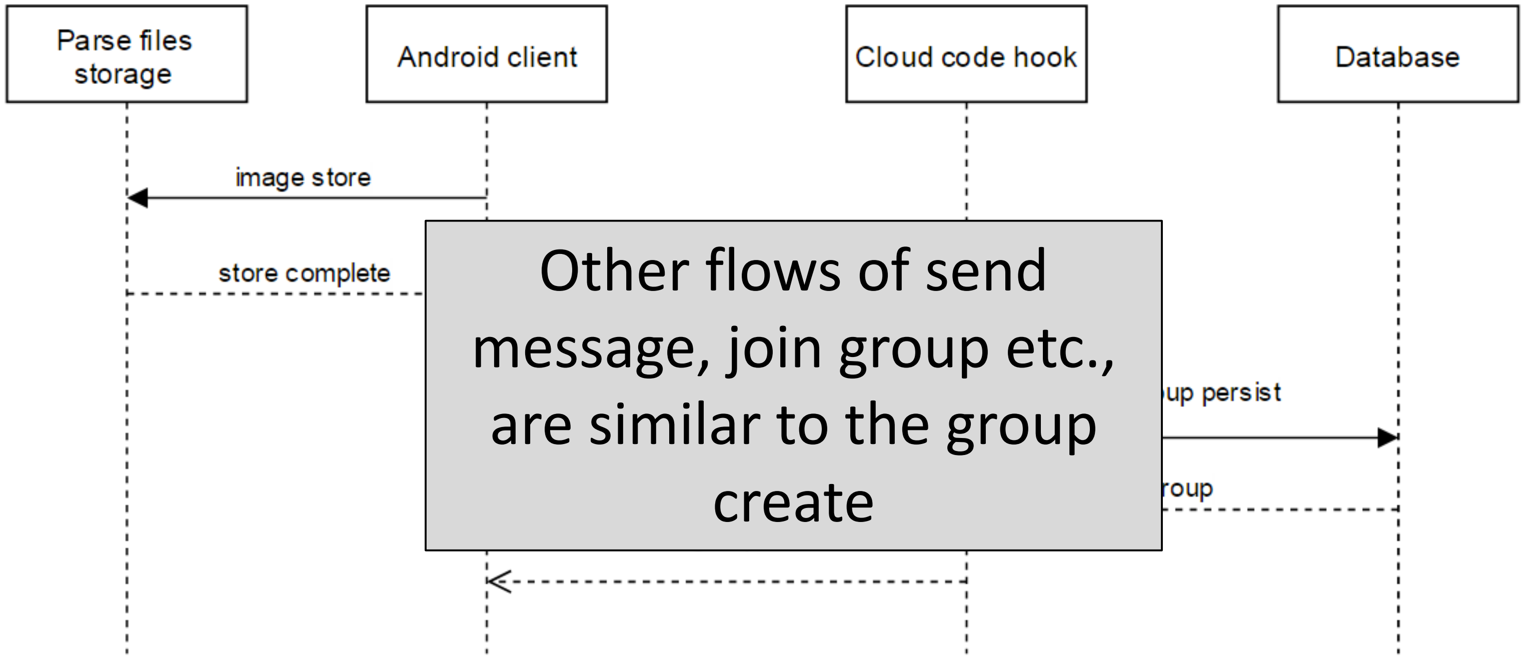
Channel	
objectId	String
groupId	String
InstallationObjectId	Array
createdAt	Date
updatedAt	Date
column_6	int
column_7	int

GroupMembership	
objectId	String
groupId	String
user	ParseUser
updatedAt	Date
createdAt	Date

User	
objectId	String
username	String
fullName	String
FBauthData	authData
password	String
profilePicUrl	String
installation	Pointer
createdAt	Date
updatedAt	Date

Installation	
objectId	String
userId	String
...	...
...	...
...	...

Group create flow



Other flows of send message, join group etc., are similar to the group create

Cloud code hooks:

- "before save" hook responsible for down scaling the group image
- "after save" hook responsible for persisting group memberships to DB and sending push notifications to new group members (besides creator). Creator will receive the newly persisted object with its id.

Android Notification Receiver

Receives push notification and decides on the action to perform.

- New group/message – persist it to the local datastore. (and change display accordingly)
- Group update - display accordingly

Message status

Android client posts a new message.

- Upon response success, message status changes to “delivered” (✓).

- Upon push notification that all recipients saw the message (opened the chat room), message status changes to “seen by all” (✓).

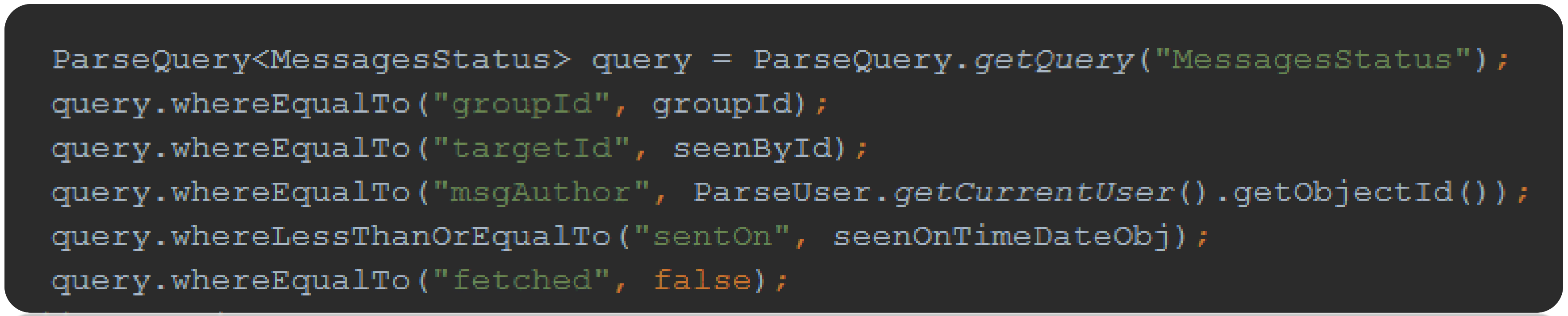
Seen-notification logic

Android client posts a series of messages.

- Cloud code creates in the DB MessageStatus record, for each message, for each recipient.
- Upon recipient opens the chat group, a RPC is invoked, with the current timestamp.
- The remote procedure updates all MessageStatus records that are still unseen, as seen by this recipient, and a push notification is sent to the author.

Seen-notification logic cont.

- Upon receiving the “seenBy” push, android client fetches the relevant MessageStatus records.



```
ParseQuery<MessagesStatus> query = ParseQuery.getQuery("MessagesStatus");
query.whereEqualTo("groupId", groupId);
query.whereEqualTo("targetId", seenById);
query.whereEqualTo("msgAuthor", ParseUser.getCurrentUser().getObjectId());
query.whereLessThanOrEqualTo("sentOn", seenOnTimeDateObj);
query.whereEqualTo("fetched", false);
```

Seen-notification logic cont.

- Upon receiving the “seenBy” push, android client fetches the relevant MessageStatus records.
- When all recipients read the message, the status changes to “seen by all”, and cloud code discards the MessageStatus records for that message.

That’s all folks!