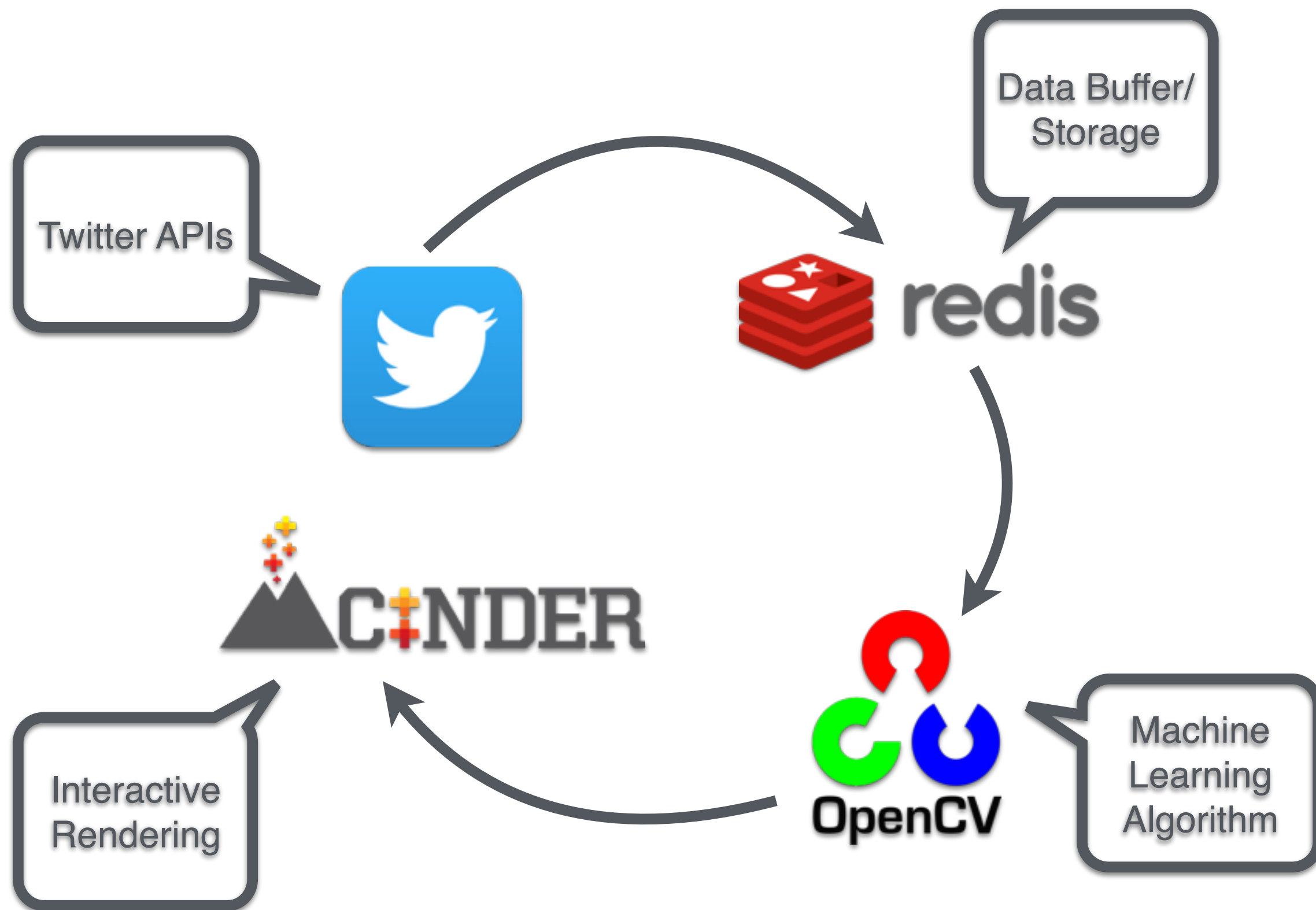
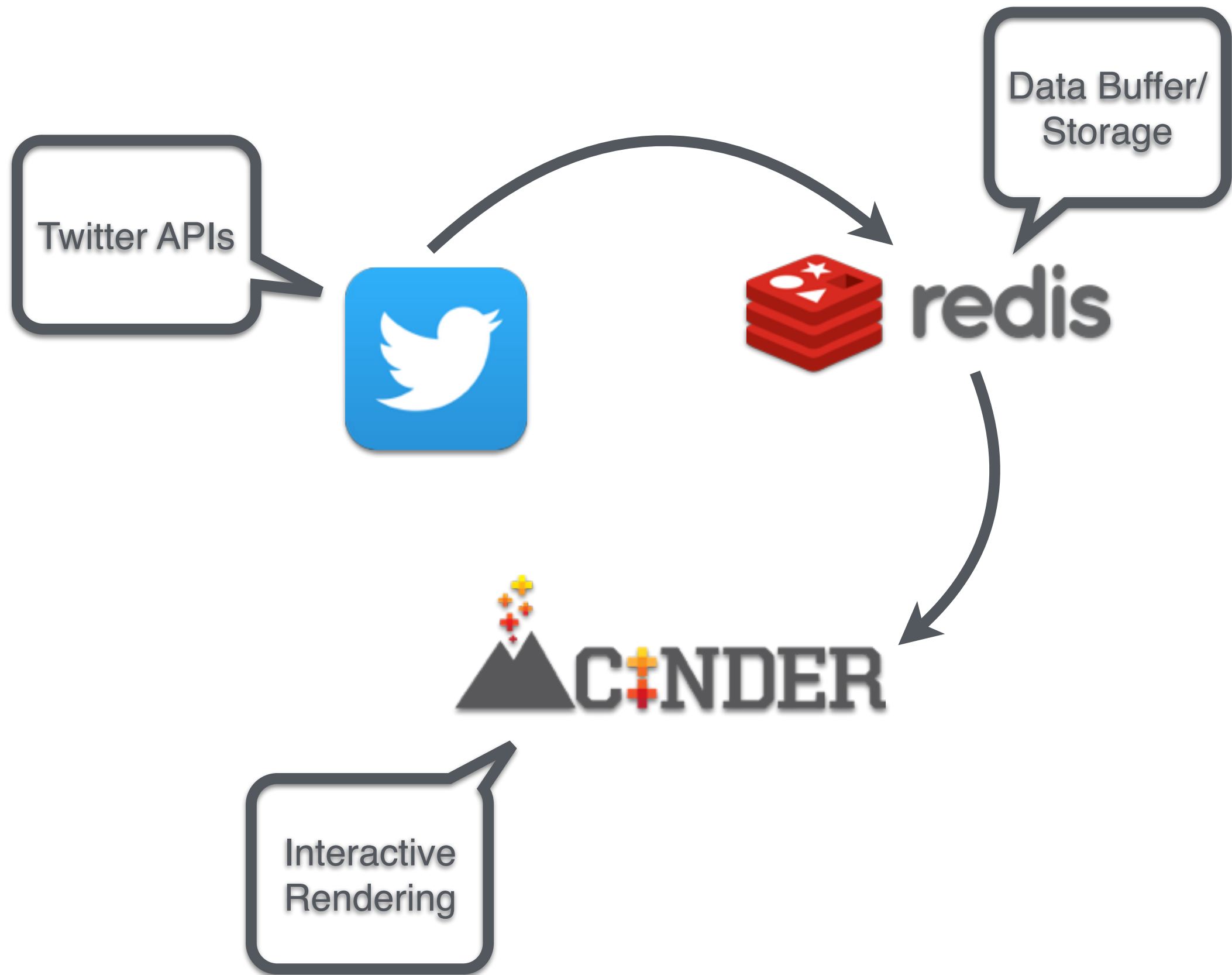


Interactive Big-Data Visualization

Lu Han
Zilong Jiao
Zhi Xing





Outline

- Twitter REST API
- Redis database
- Twitter => Redis
- Cinder Library
- Redis => Cinder
- Cinder Physics Simulation
- Project directions

Twitter REST API



- GET users/show
 - <https://api.twitter.com/1.1/users/show.json>
 - get a variety of information about a user
- GET friends/list
 - <https://api.twitter.com/1.1/friends/list.json>
 - get a collection of user objects for users the specified user is following
- GET followers/list
 - <https://api.twitter.com/1.1/followers/list.json>
 - get a collection of user objects for users following the specified user

Redis



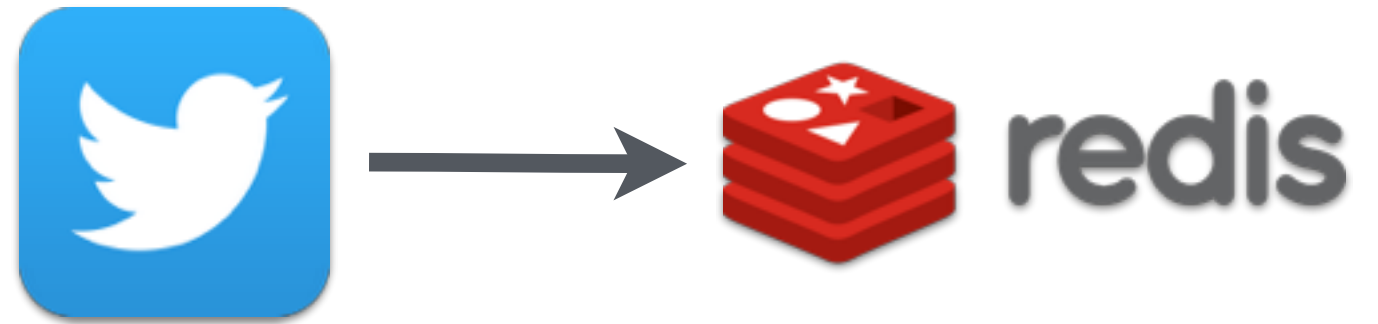
- SET key value
- GET key
- DEL key
- KEYS pattern
 - all the keys matching the given pattern

Redis



- RPush key element
 - push an element to the back of a list
- LPush key element
 - push an element to the front of a list
- LRange key begin end
 - the elements from 'begin' to 'end'
- LLen key
 - the size of the list
- RPop key element
- LPop key element
- ... (<http://redis.io/commands>)

Back end



- Crawl the social network starting from the specified user via Twitter REST API
- Save the adjacency list to Redis database using Redis Python client

Back end



redis

- `oauth_login()`
 - establish authorized connection
- `make_twitter_request(twitter_api_func, max_errors=100, *args, **kw)`
 - helper function to call the API functions
 - `twitter_api_func`: the API function
 - `max_errors`: maximum number of errors to be dealt with
 - `*args`: arbitrary number of arguments
 - `**kw`: arbitrary number of keyword arguments

Back end



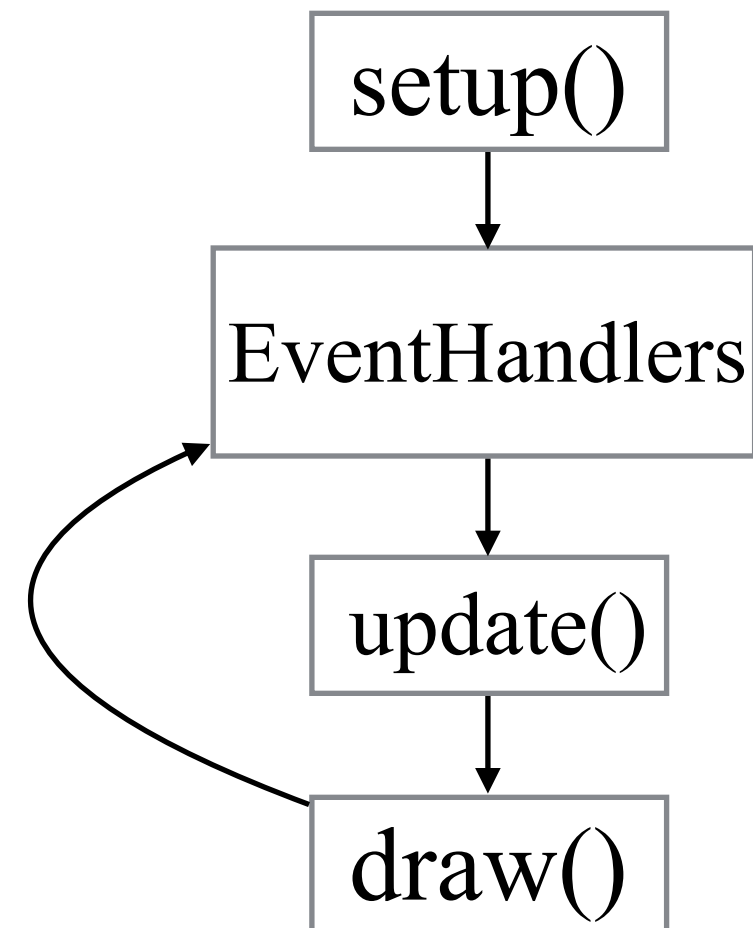
redis

- `get_friends_followers(twitter_api,
 screen_name=None,
 user_id=None,
 friends_limit=sys.maxint,
 followers_limit=sys.maxint)`
 - get the friends and followers profiles for the specified user
 - `twitter_api`: established connection
 - `screen_name`: the username of a user
 - `user_id`: the ID number of a user
 - `friends_limit`: maximum number of friends to be obtained
 - `followers_limit`: maximum number of followers to be obtained
- `get_reciprocal_friends(twitter_api,
 screen_name=None,
 user_id=None,
 limit=sys.maxint)`
 - get reciprocal friends of the specified user
- `get_twitter_network(screen_name, limit=3)`
 - crawl and store the social network
 - `limit`: maximum number of hops for the final network

Create Cinder Project



- Download Cinder Package
- Use TinderBox
- Basic Project
 - `setup()`
 - `mouseDown()` (EventHandlers)
 - `update()`
 - `draw()`



Get Data



redis



CINDER

- Connecting

```
redisContext *c = redisConnectWithTimeout("127.0.0.1", 6379, timeout);  
if (c != NULL && c->err)  
{  
    printf("Error: %s\n", c->errstr);  
}
```

- Sending commands

```
reply = (redisReply*)redisCommand(c, "KEYS *");
```

- Using replies

REDIS_REPLY_ARRAY

- Freeing Memory

```
void freeReplyObject(void *reply);
```

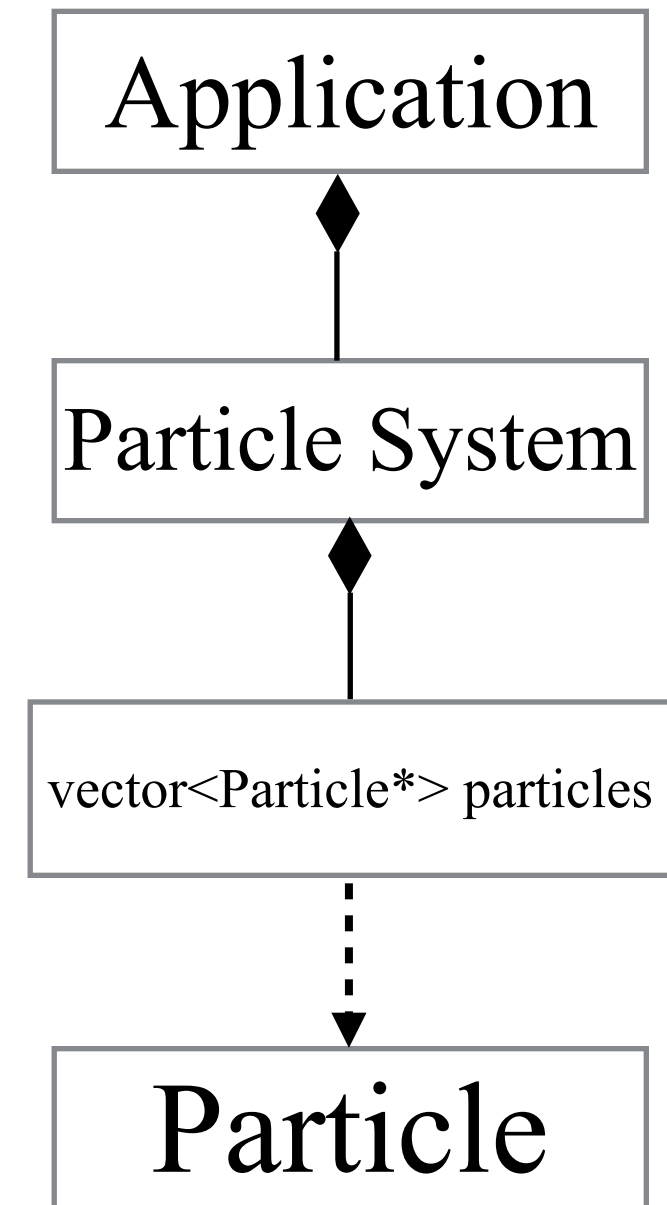
- Cleaning up

```
void redisFree(redisContext *c);
```

Project Structure



- Particle
 - Small Graphic Object
- Particle System
 - Computational Particle Collection
- Application
 - Executable





Setup

- `setup()`:

- initialize distance between two nodes

- construct a list of `Particle()` objects

- store each particle pair into separate list

- `Particle`:

- radius (the size of the particle)

- position

- velocity scalar (adjust the speed of the particle)

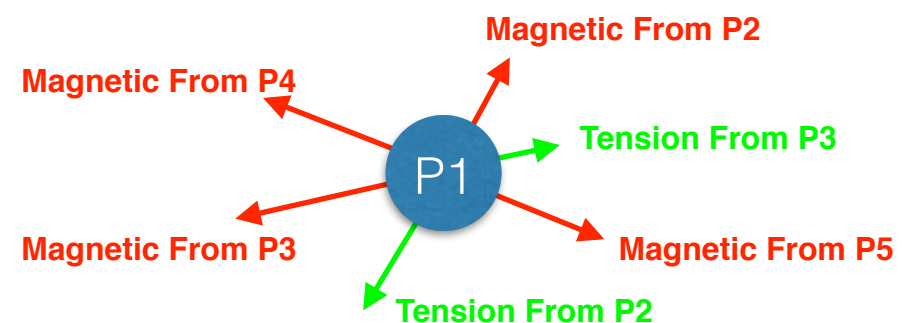
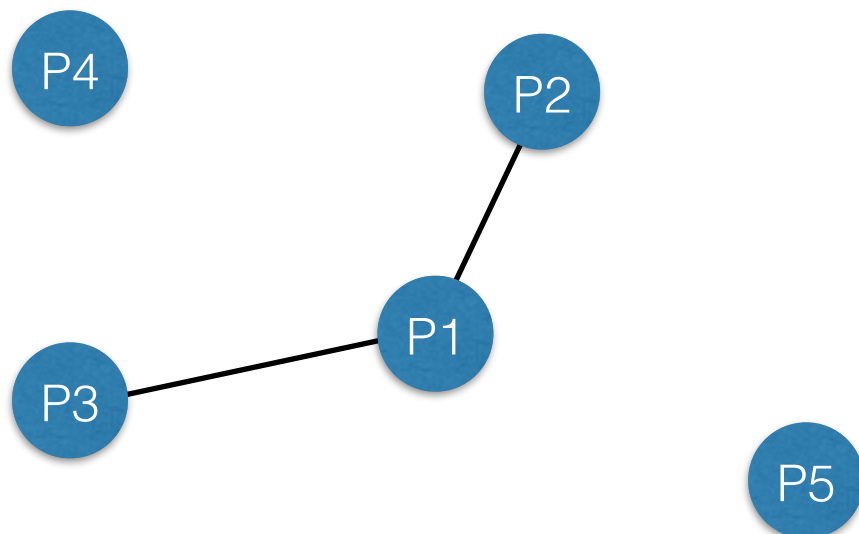
- force ($current_velocity = previous_velocity + (force / mass) * time$)

- mass

Update



- In order to animate our particle system, we need to update position of each particle according to physics principle
- Each particle will have **magnetic force** for all the other particles
- Particles are grouped as pairs
- Particles in each pair has **tension** between each other
- Particles will be stabilized, when each particle's total force equals to zero
 - $total_force = |magnetic_force| - |tension_force|$



Update



- `update()`:

compute velocity vector for each pair of particles

if two particles run into each other:

stop calculating force for this pair of particles

else:

calculate **magnetic force value** for this pair of particles, according to equation

$$Magnetic_Force_Value = (2 * default_distance - actual_distance) * mag_coefficient$$

update **magnetic force vector** for each particle, according to equation

$$Magnetic_Force_Vector = Velocity_Vector * Force_Value$$

calculate **tension** between each particle pair, according to equation

$$Tension = Velocity_Vector * (actual_distance - default_distance)$$

sum up tension and magnetic force to get final force

update particle system based on the final force

Interaction



- User interaction:

upon clicking on particles on screen, particles will move with cursor:

when mouse is pressed:

set up a range (min_dist , max_dist) for choosing handle particle

select one of particles falling within the range as handle particle

set a flag to notice `update()` to set the handler particle's position to be current cursor position

when mouse is released

reset the flag to notice `update()` to treat handler particle as normal particle

Rendering



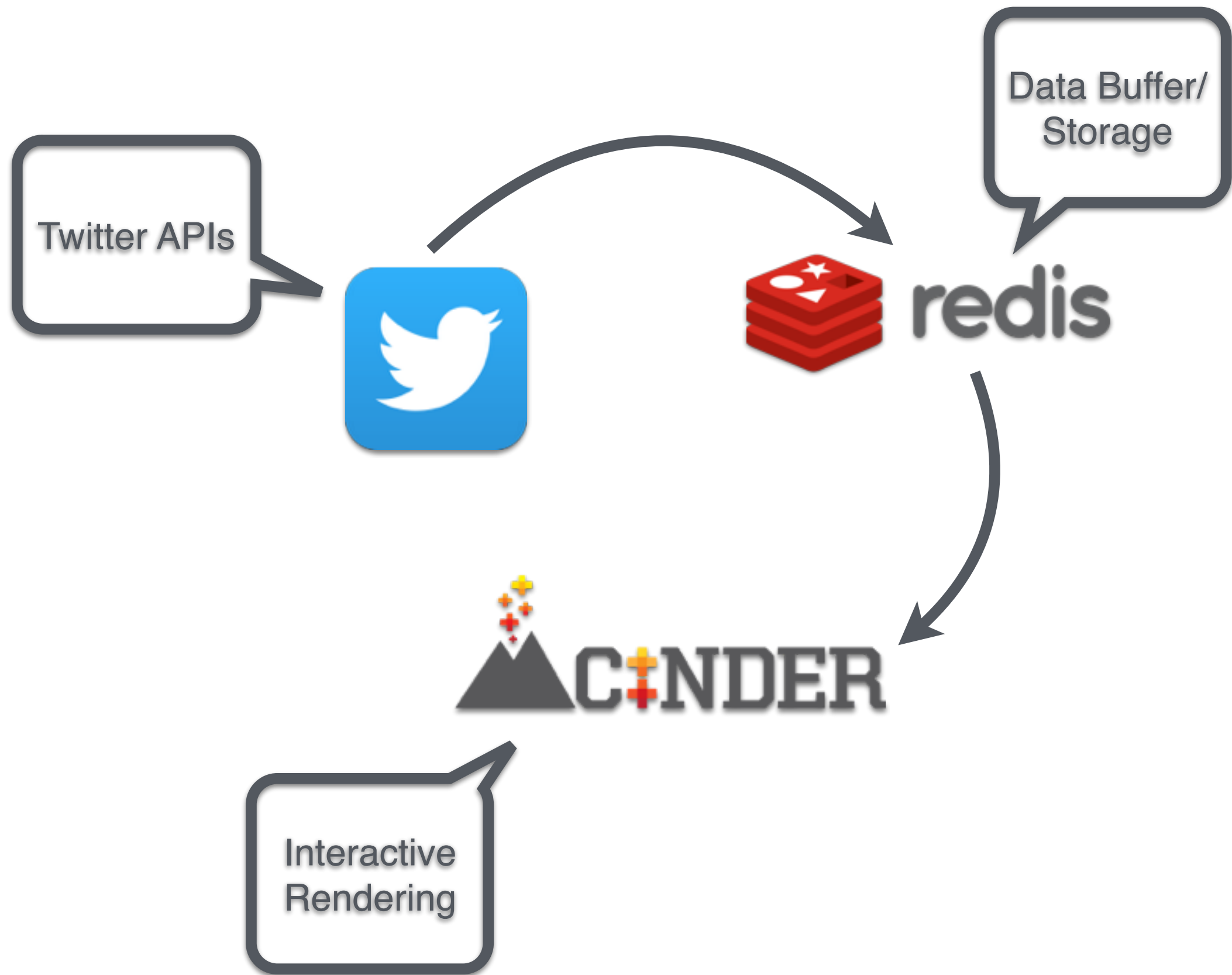
- Cinder has packed built-in OpenGL functions for rendering
 - set up OpenGL context
 - window, color, viewport, etc.
 - each particle object has its own draw()
 - draw a basic black circle based on all updated parameters within the particle
 - iteratively call each particle's draw() to draw entire scene

Project directions

- Zoom-in details
- 3D social network
- Performance optimization
- Distributed crawlers

Outline

- Twitter REST API (<https://dev.twitter.com/rest/public>)
- Redis database (<http://redis.io>)
- Twitter => Redis (<https://github.com/andymccurdy/redis-py>)
- Cinder (<http://libcinder.org/docs/v0.8.6/>)
- Redis => Cinder (<https://github.com/redis/hiredis>)
- Cinder Physics Simulation (Cinder Creative Coding Cookbook by Rui Madeira and Dawid Gorny)
- Project directions



Q & A

Thank you.