Interactive Social Network Visualization

Lu Han
Zilong Jiao
Zhi Xing

Outline

- Accomplishments
- Project workflow
- Distributed crawlers
- Communication channel
- Cinder framework & our implementation
- Physics simulation
- Rendering
- Conclusion

SocialNetworkVisualization

27% Wed 4:12 PM

UPDATE

USERNAME: @

OK

SocialNetworkVisualization

27% Wed 4:12 PM

UPDATE

USERNAME: @

OK

SocialNetworkVisualization

28% Wed 4:14 PM

UPDATE

USERNAME: @

OK

SocialNetworkVisualization

28% Wed 4:14 PM

UPDATE

USERNAME: @

OK

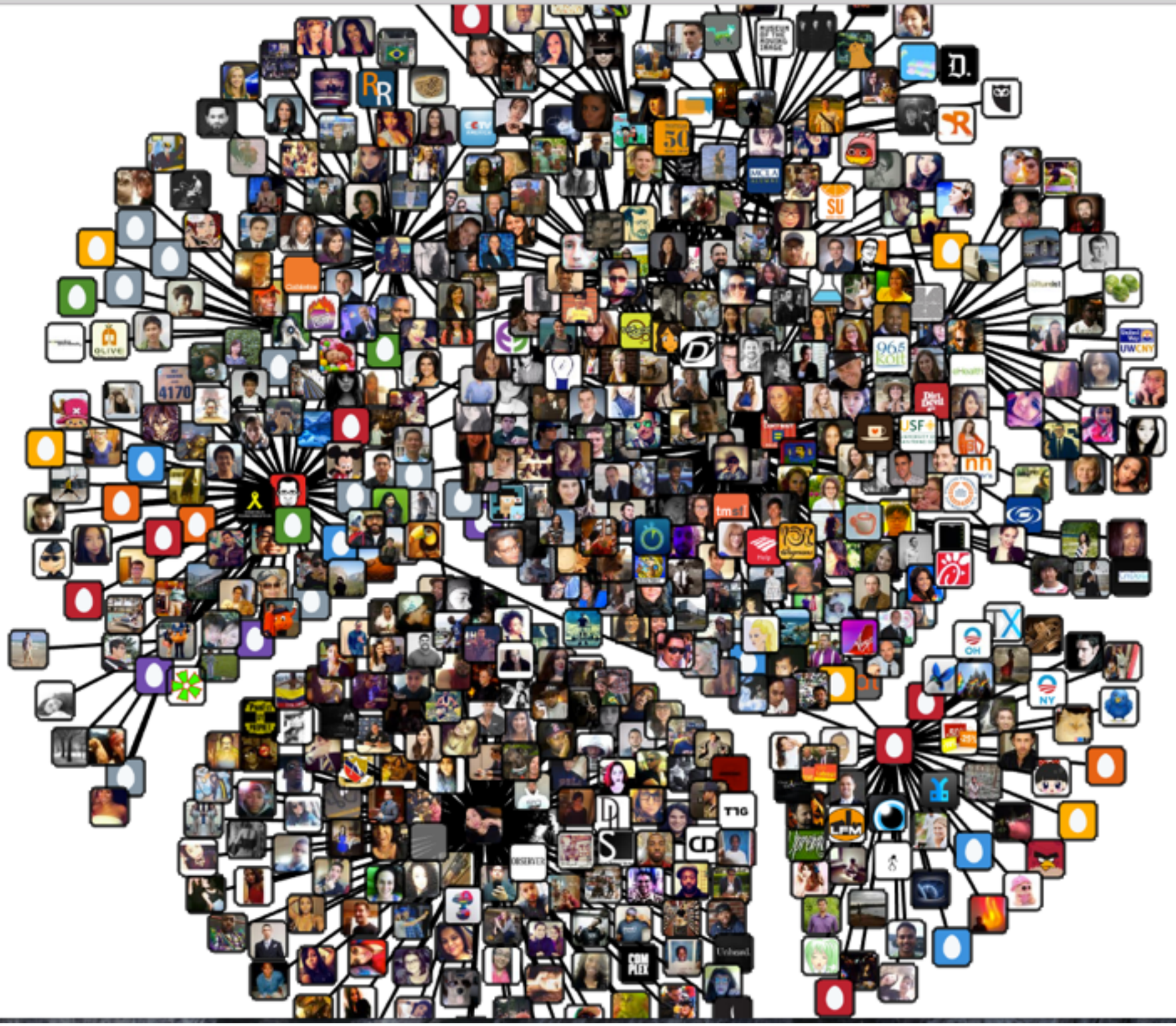
SocialNetworkVisualization

28% Wed 4:16 PM

UPDATE

USERNAME: @

OK



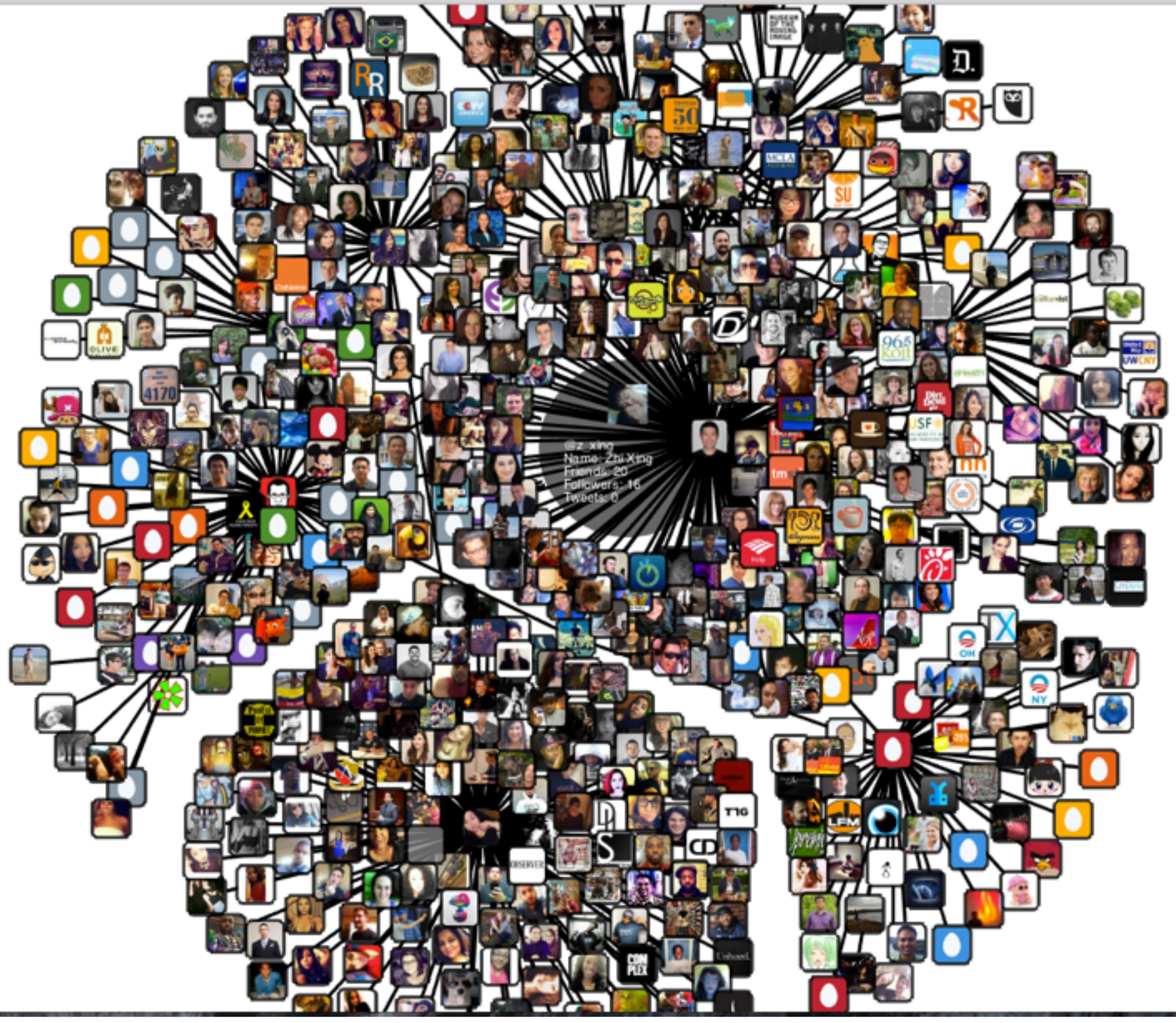
SocialNetworkVisualization

28% Wed 4:16 PM

UPDATE

USERNAME: @

OK



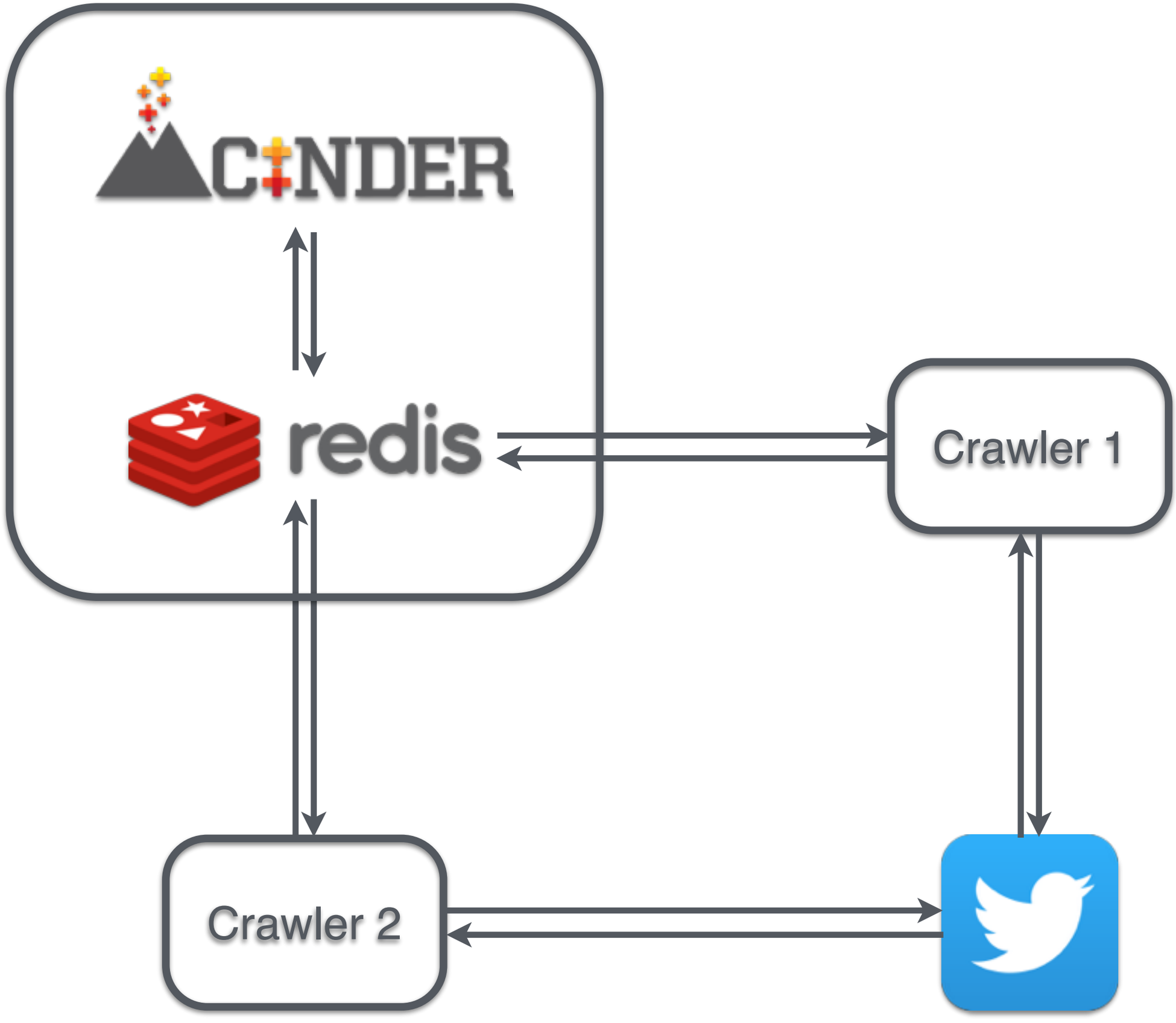
Accomplishments

- The displayed graph looks very cool
  - profile pictures and information

- The software interactively creates Twitter social network graph starting from a focused user
  - want to test Six Degree of Separation?

- The GUI is really interactive/responsive even when there're thousands of nodes
  - distributed crawlers
  - Redis database as data buffer and permanent storage
  - multithreading and memory-caching

- Simulated physics "automatically" clusters the nodes
  - repulsion, attraction, resistance, gravity and more!


CINDER
redis
Crawler 1
Crawler 2

Distributed crawlers

- Run the same python script
- Keep checking Redis for requests
- Get the neighbors of the requested user, and send everything to Redis
- Use different OAuth authorizations
- Can be added or removed at runtime
- Allow parallel updates of different users and circumvention of rate limit

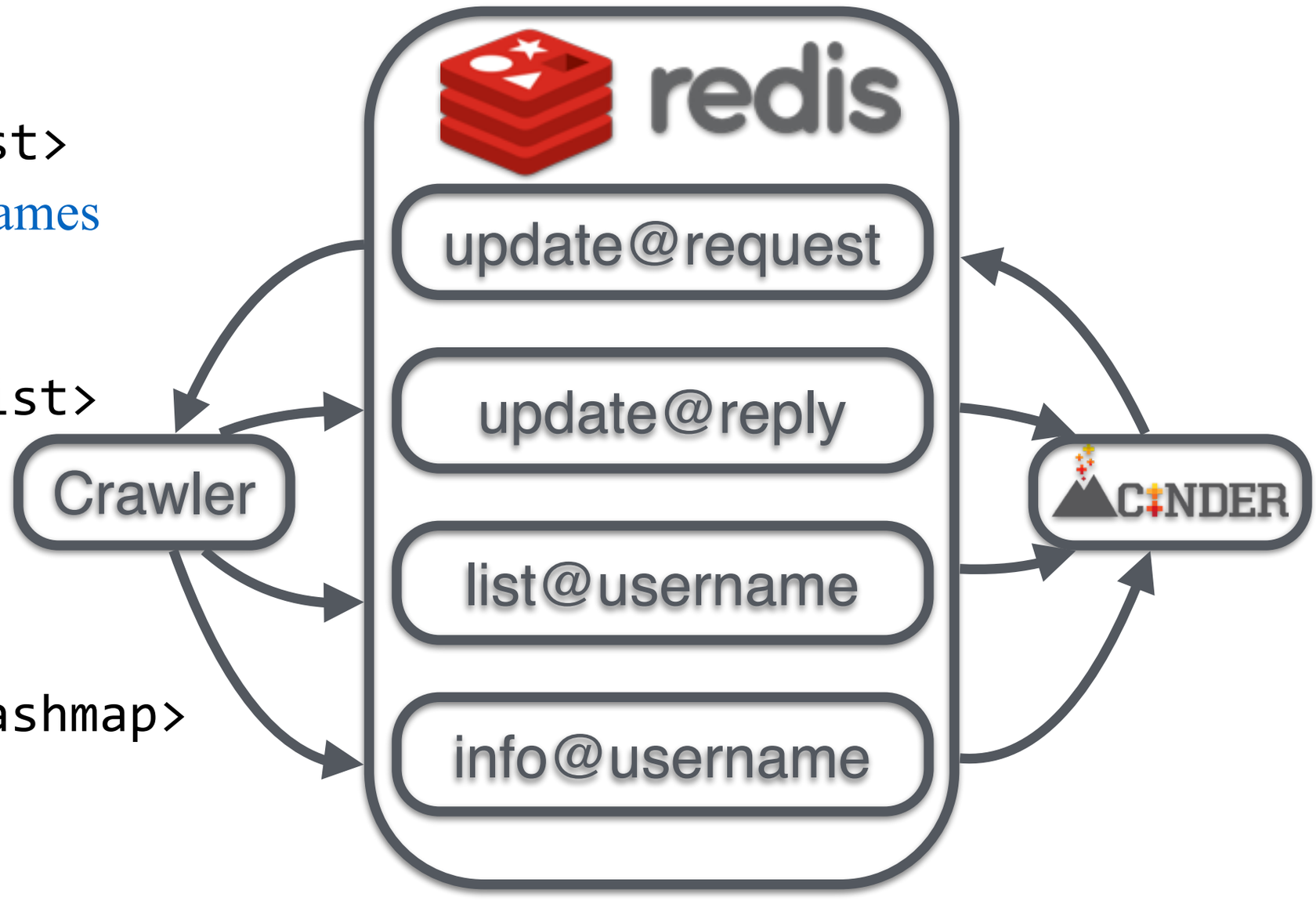
Communication channel

- `update@request <list>`
  - cinder pushes usernames
  - crawlers pops them

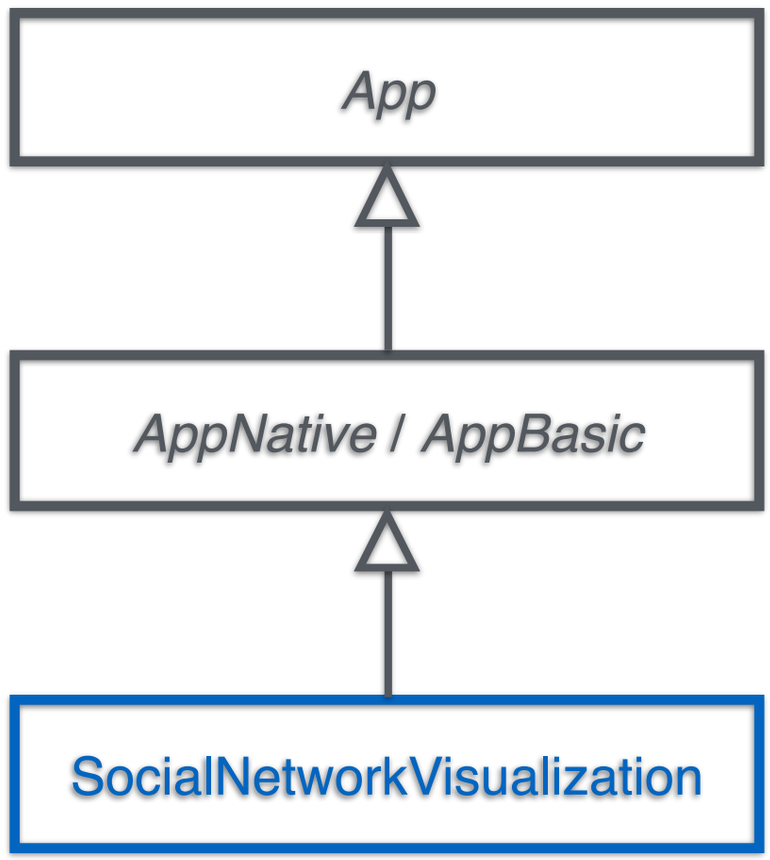
- `update@reply <list>`
  - crawler pushes usernames
  - cinder pops them

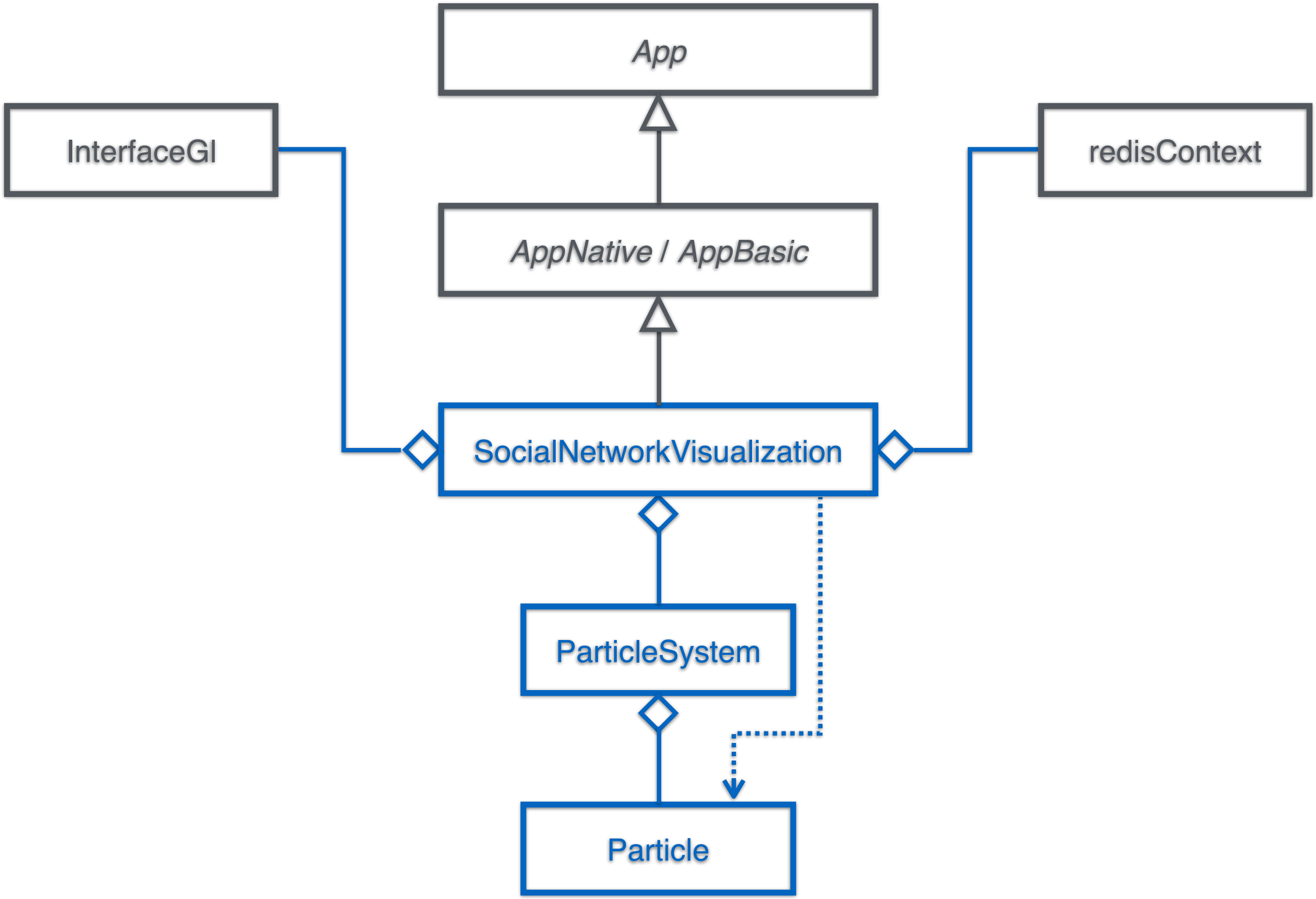
- `list@username <list>`
  - adjacency lists
  - crawlers write
  - cinder reads

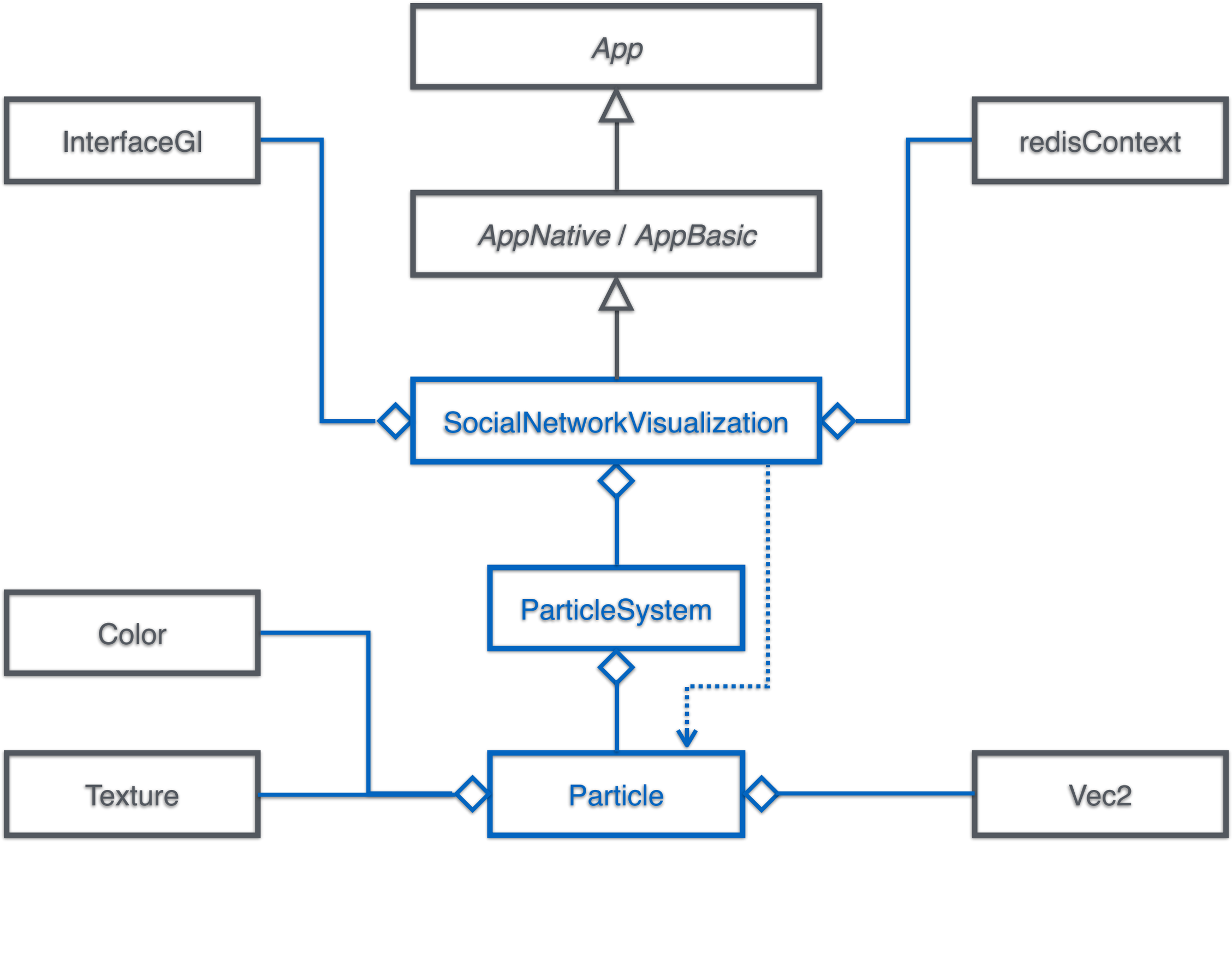
- `info@username <hashmap>`
  - user profile
  - crawlers write
  - cinder reads

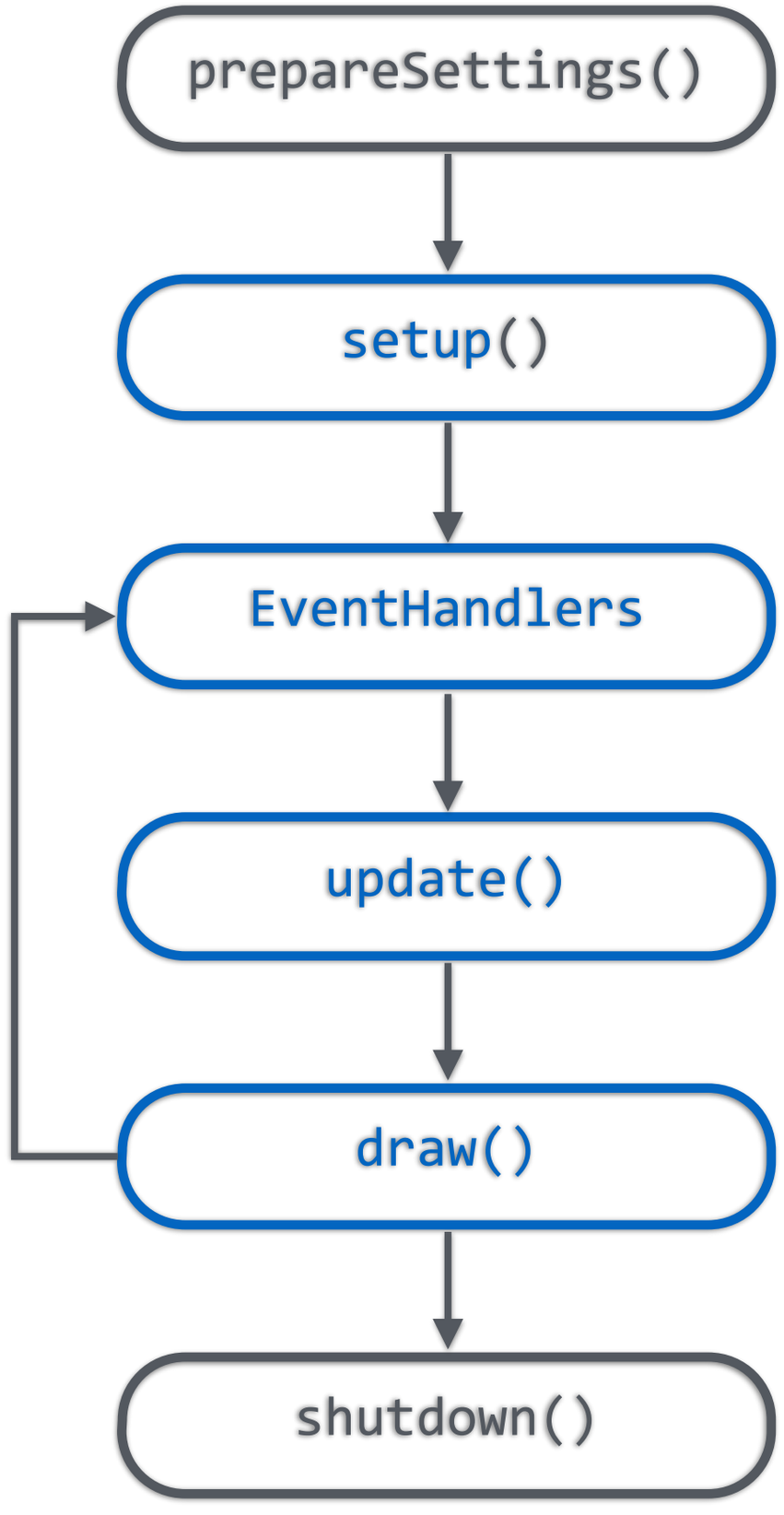


Cinder framework & our implementation

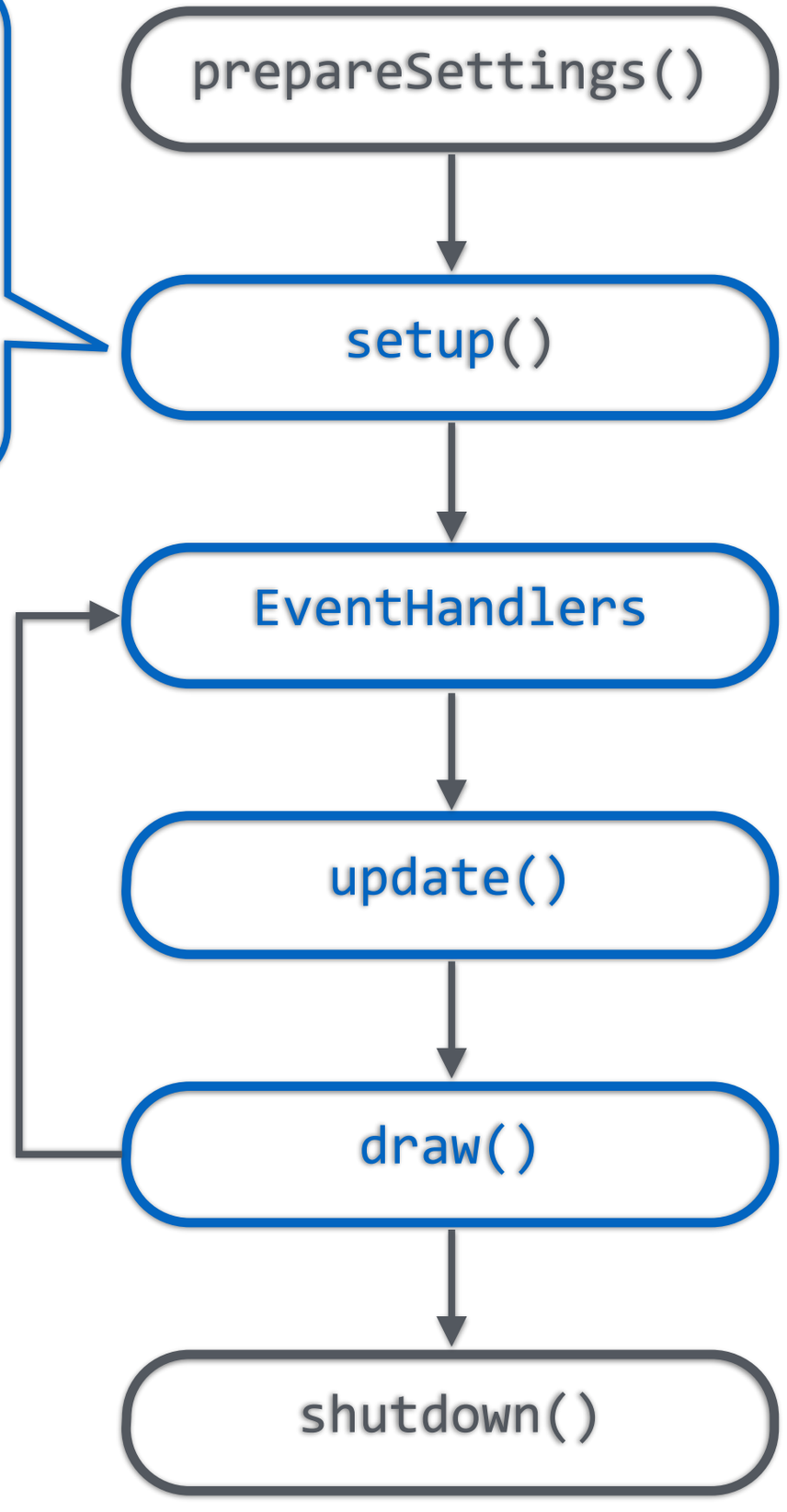


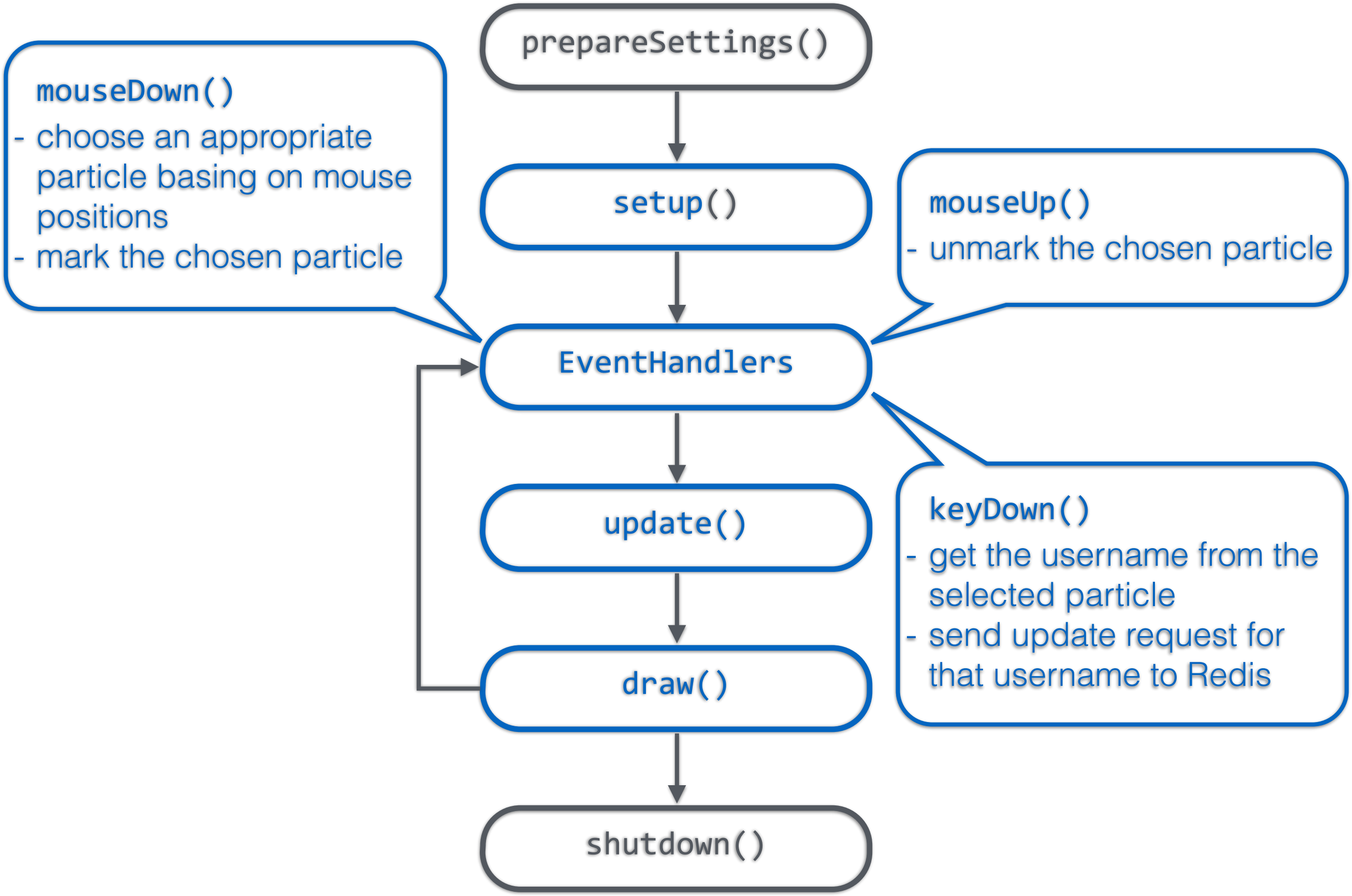

App
InterfaceGl
AppNative / AppBasic
redisContext
SocialNetworkVisualization
ParticleSystem
Particle

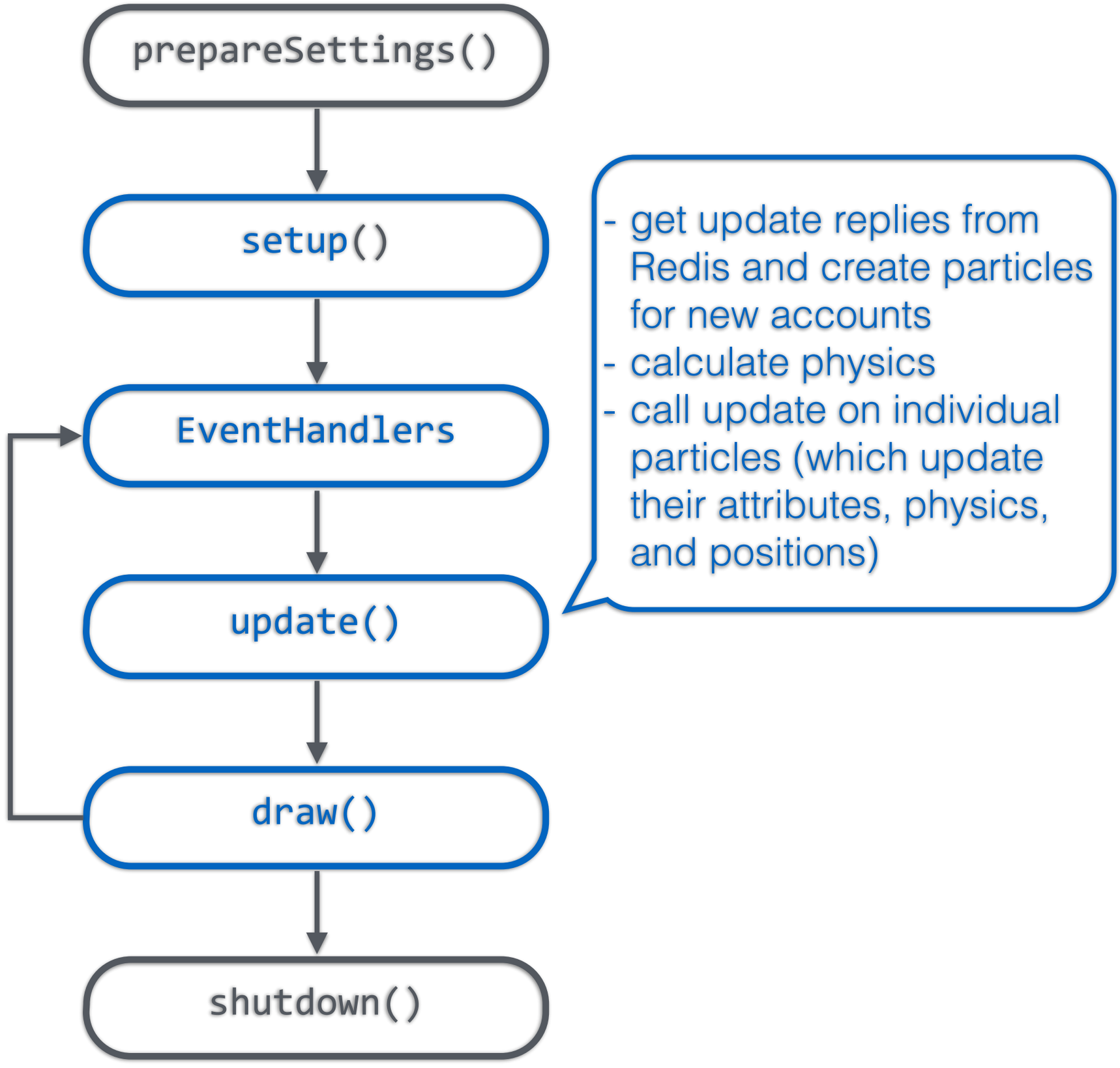

App
InterfaceGl
AppNative / AppBasic
redisContext
SocialNetworkVisualization
ParticleSystem
Color
Texture
Particle
Vec2

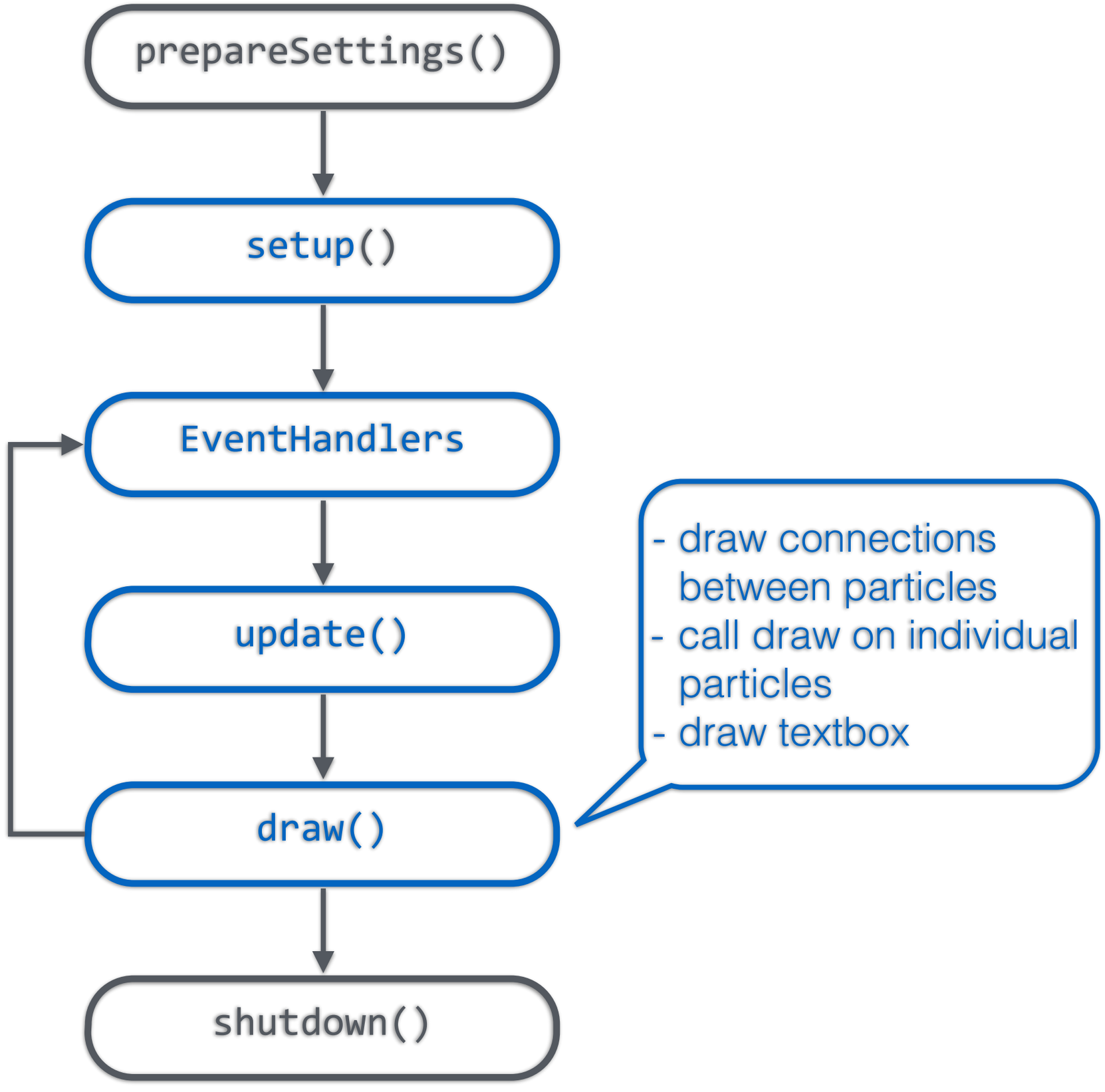

prepareSettings()
setup()
EventHandlers
update()
draw()
shutdown()

- parse arguments
- create textbox
- connect to Redis
- send update request for the focused account to Redis

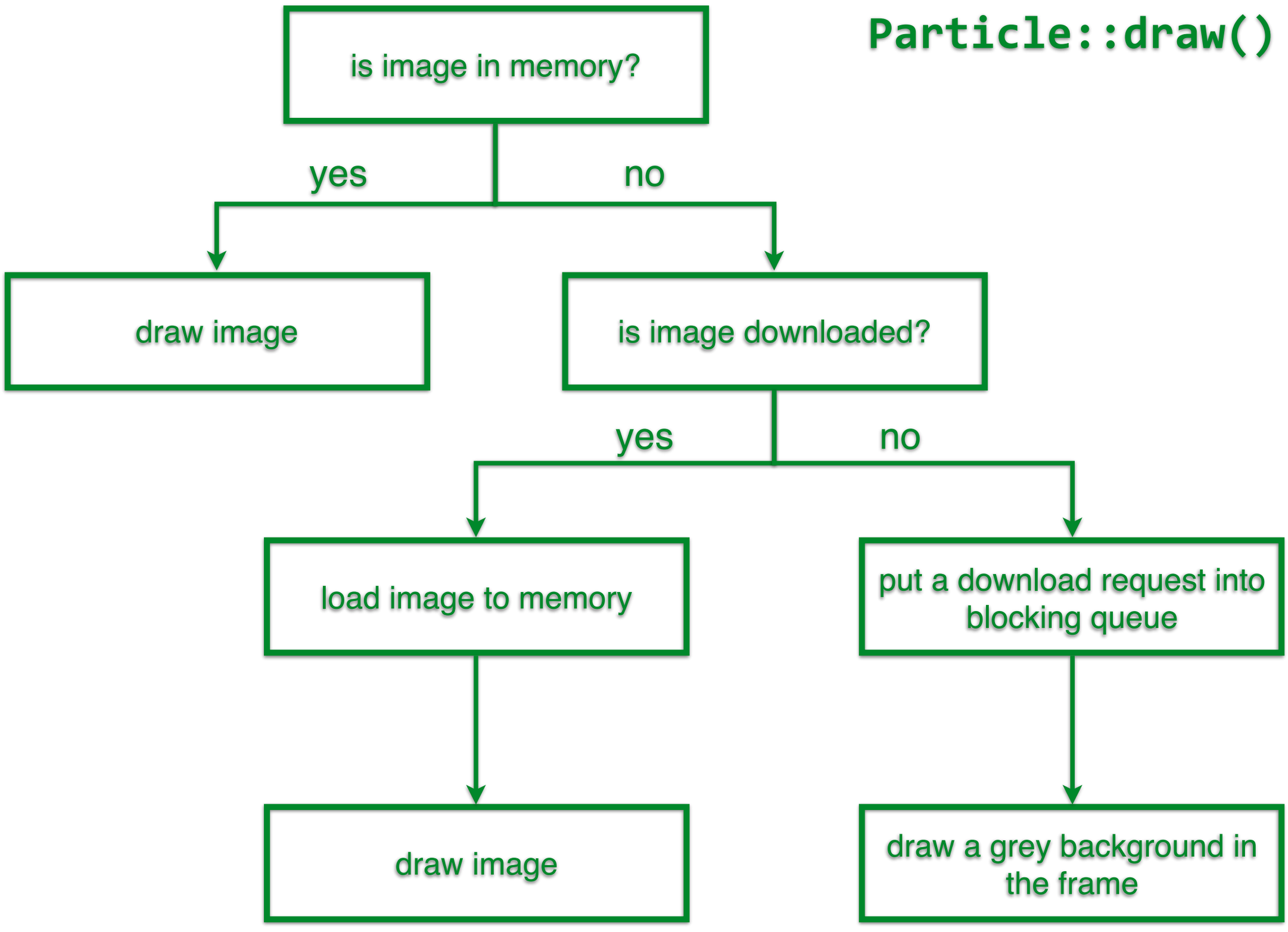



prepareSettings()
setup()
EventHandlers
update()
draw()
shutdown()
mouseDown()
- choose an appropriate particle basing on mouse positions
- mark the chosen particle
mouseUp()
- unmark the chosen particle
keyDown()
- get the username from the selected particle
- send update request for that username to Redis


prepareSettings()
setup()
EventHandlers
update()
draw()
shutdown()
- get update replies from Redis and create particles for new accounts
- calculate physics
- call update on individual particles (which update their attributes, physics, and positions)


prepareSettings()
setup()
EventHandlers
update()
draw()
shutdown()
- draw connections between particles
- call draw on individual particles
- draw textbox

`Particle::draw()`



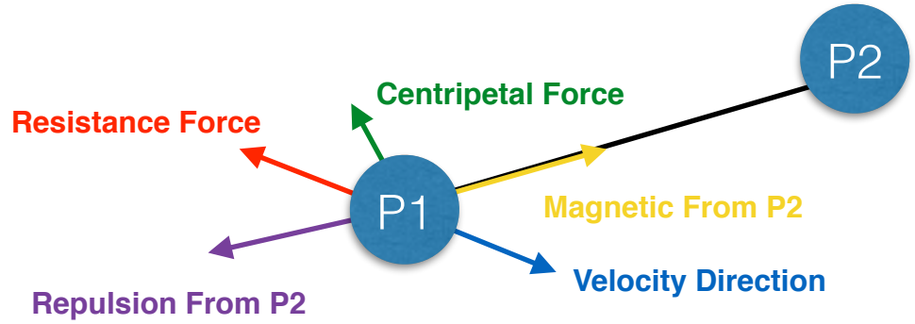
Set Up

- **Particle System**
  - Particles —> Map<key=Name, value=Particle>
  - Edges —> Vector<(Particle, Particle)>
  - Define default distance of edges
  - Assign initial coordinates for each particle
    - particle will be evenly surrounded by its neighbor particles.

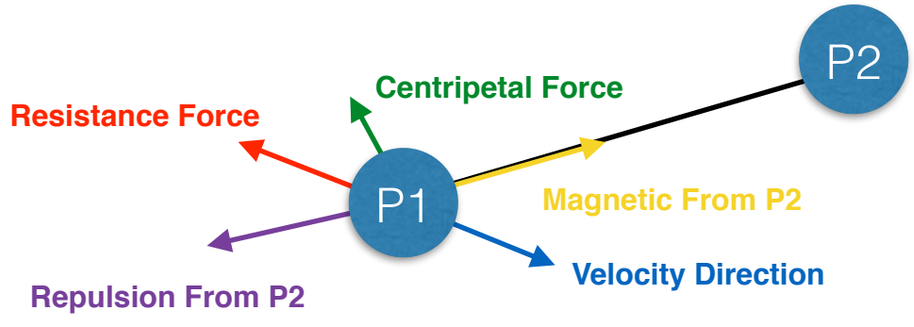
- **Particle**
  - radius; (the size of the particle)
  - position (current & previous);
  - color;
  - force;
  - mass;

Update

- In order to animate our particle system, we need to update position of each particle according to physics principle.
- Each particle has:
  - Repulsion Force; (Particle to Particle)
  - Magnetic; (Edge between particle)
  - Resistance Force; (Proportional to particle velocity but opposite direction)
  - Centripetal Force; (Proportional to particle and center of screen and lways points to the center)



- Pseudo Code for updating the particles:



update() {

if two particles run into each other:

stop calculating force for this pair of particles

else:

calculate **repulsion force** for this pair of particles (P1,P2), according to equations
(In terms of particle P1)

Repulsion_Force = Repulsion_Force_Vector * Repulsion_Coefficient;

Repulsion_Coefficient=(current_distance-2*default_distance)*constant*current_distance* mass;

Repulsion_Force_Vector= Position_of_P1 - Position_of_P2;

calculate **magnetic force** between each particle pair, according to equation

Magnetic_Force=Velocity_Direction_Vector * (actual_distance-default_distance)

update each particle based on the final force

}

- Pseudo Code for updating the particles:

particle.update(){

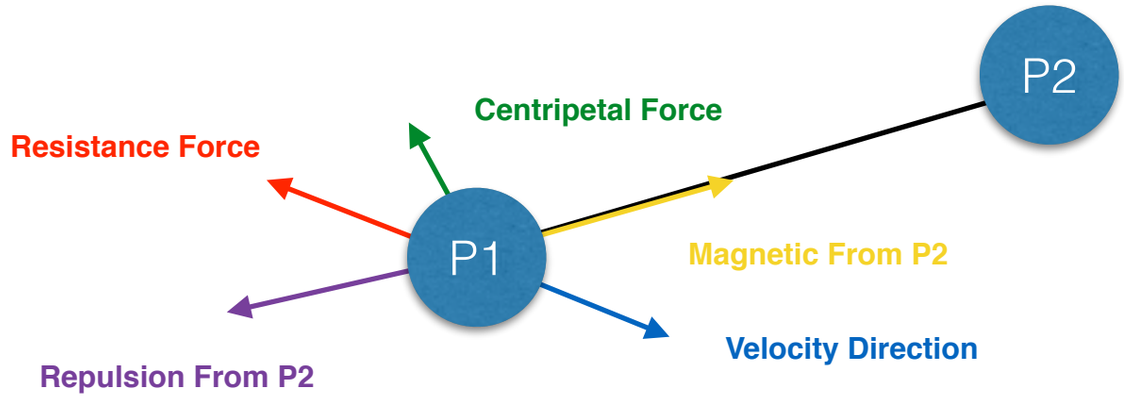
if (current_particle is selected){

increase mass of particle;

}else {

use default mass;

}



calculate **velocity** as below:

velocity= (particle_current_position - particle_previous_position) * decay_coefficient

Note: decay_coefficient ensures that the particle will not pass its balance point while being updated

calculate **centripetal force:**

centripetal_force=gravity_direction_vector * gravity_coefficient * distance_to_screen_center

centripetal_direction_vector=particle_current_position - screen_center

Sum up all computed force to get **Final Force;**

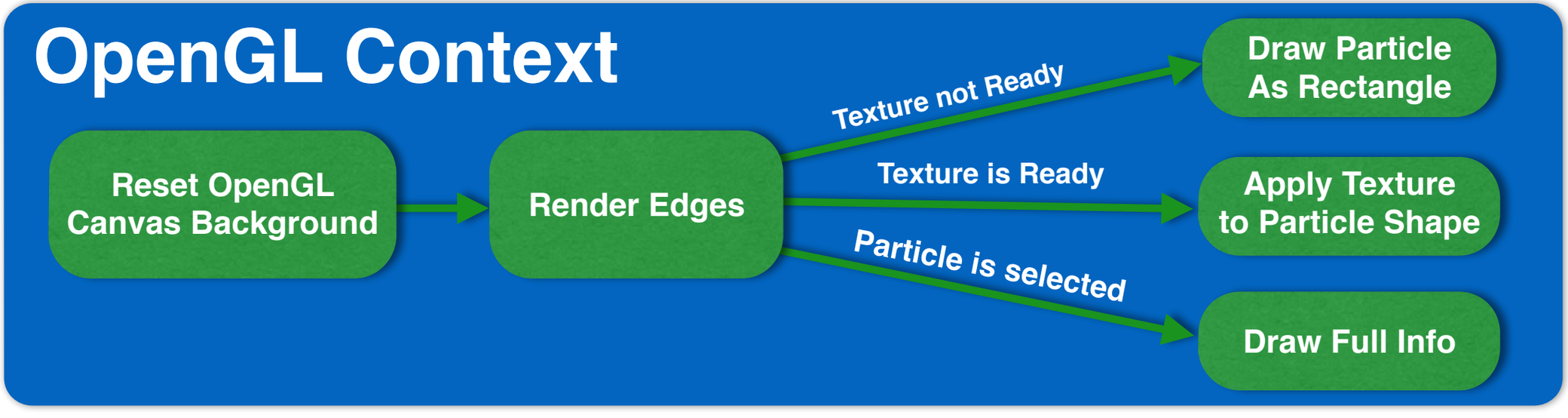
Update current position of the particle as below:

particle_current_position += velocity+ final_force / mass;

}

Rendering

- Graphic in Cinder is rendered by using OpenGL
- Cinder provides good package for OpenGL functions
  - automatically set up OpenGL context
  - automatically apply texture image to mesh or shape contour
    - **gl::draw()** does them all
- Can use customized shader (GLSL) to render advanced image



Rendering

- Draw shapes in Cinder
  - **gl::drawSolidRect();**
  - **gl::drawStrokedRoundedRect, etc.**
- Draw texture in Cinder
  - **gl::draw(image, top_left_corner, bot_right_corner);**
- Draw text in Cinder
  - **gl::TextureFontRef text;**
  - **text.drawStringWrapped();**
- If render purely with OpenGL, developers have to much more code to achieve similar result.
  - E.g. While drawing rectangle, user has to specify 4 different vertexes and then connect them with straight line in correct order.

Conclusion

- **Cinder is very easy to use**
  - nicely packed built-in OpenGL library
  - automatically generate templet for creative coding
  - automatically does rendering context set up
- **Redis is a very good data buffer**
  - support multi-language
  - acting as communication channel between different program components, which can be written by different languages
  - can retrieve and store data in different format

Conclusion

- **Difficulties**
  - Stabilizing physics simulation
    - particles can pass their balance point
  - Slow rendering speed
    - need to download to many images
    - loading images as texture is slow
- **Finding way out & Surprises**
  - using memory caching to reduce the slowing image loading speed
  - using multithreading to concurrently download images
  - add resistance force, velocity decaying method and centripetal force to stabilize physics simulation
  - our program, which can only render at most 200 particles before, now can render thousands particles.
  - the simulation is stable, and all the particles intends to get stuck into a location eventually

Thank you.

Q & A