How Richard Found Friends

Richard Schneeman
@schneems

I hack rubies for @gowalla

Meet Richard

- Richard needs more friends
  - Richard has one or more friends on our network
- We can help him find more!

def potential_friend

- *Someone who Richard likely knows and wants to "friend" on a service (e.g. Gowalla)*

def mutual_friend

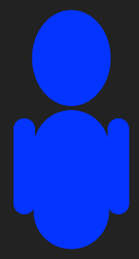
- When two users have the same friend, it is a mutual friend

Richard's Friends

Sunday, June 5, 2011

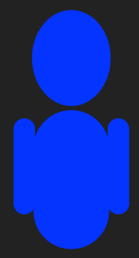
Richard's Friends

John



Richard's Friends

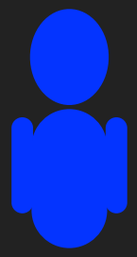
John



Jane

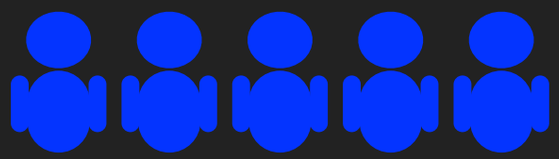
Richard's Friends

John



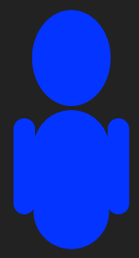
Jane

John's Friends



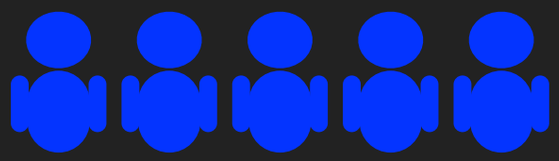
Richard's Friends

John



Jane

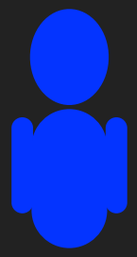
John's Friends



Jane's Friends

Richard's Friends

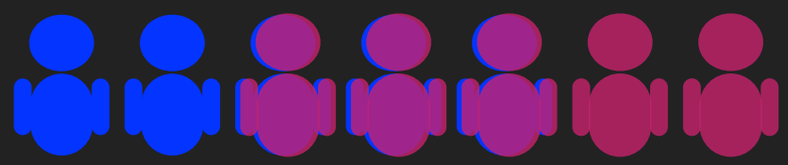
John



Jane

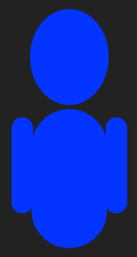
John's Friends

Jane's Friends



Richard's Friends

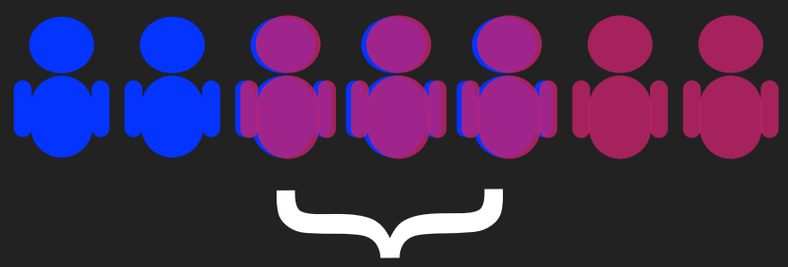
John



Jane

John's Friends

Jane's Friends



Mutual Friends

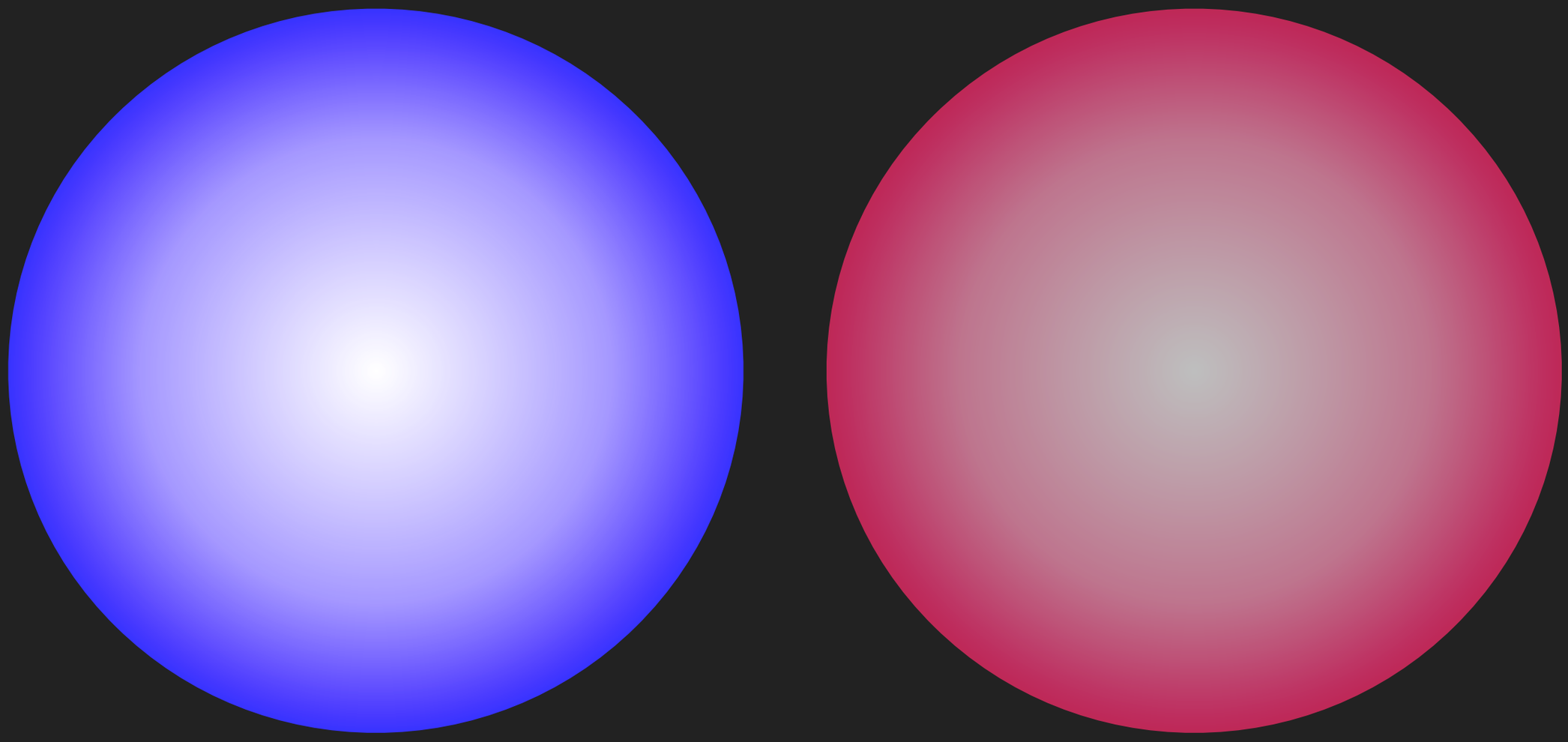
Friends of Friends

- Friends of friends are all potential friends
  - People are more likely to “friend" another user if they are a “friend of a friend"
- Mutual friend numbers provide a ranking system
  - For every overlapping friend of a friend +1

Friends of Friends

John Doe's Friends

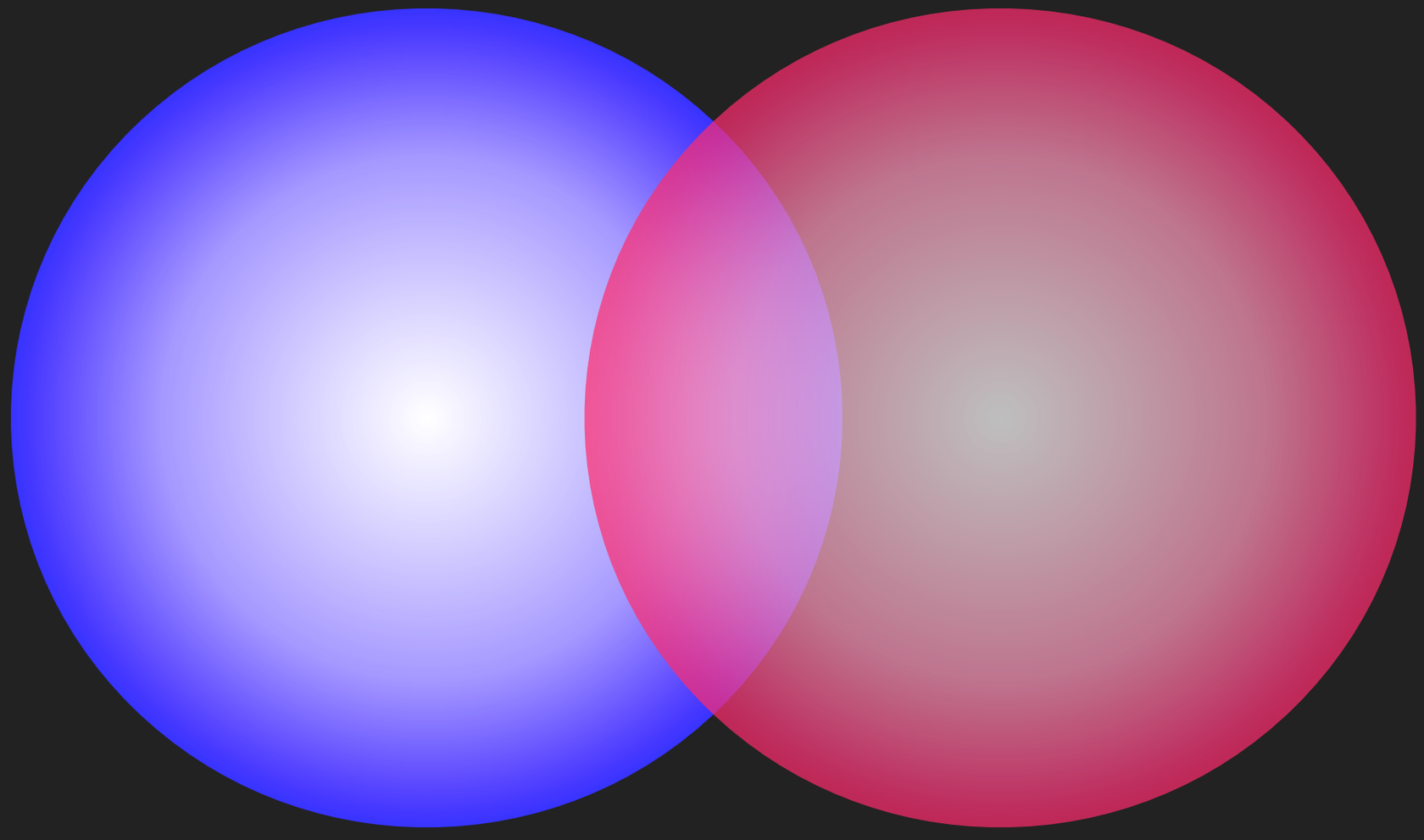
Jane Doe's Friends



Friends of Friends

John Doe's Friends

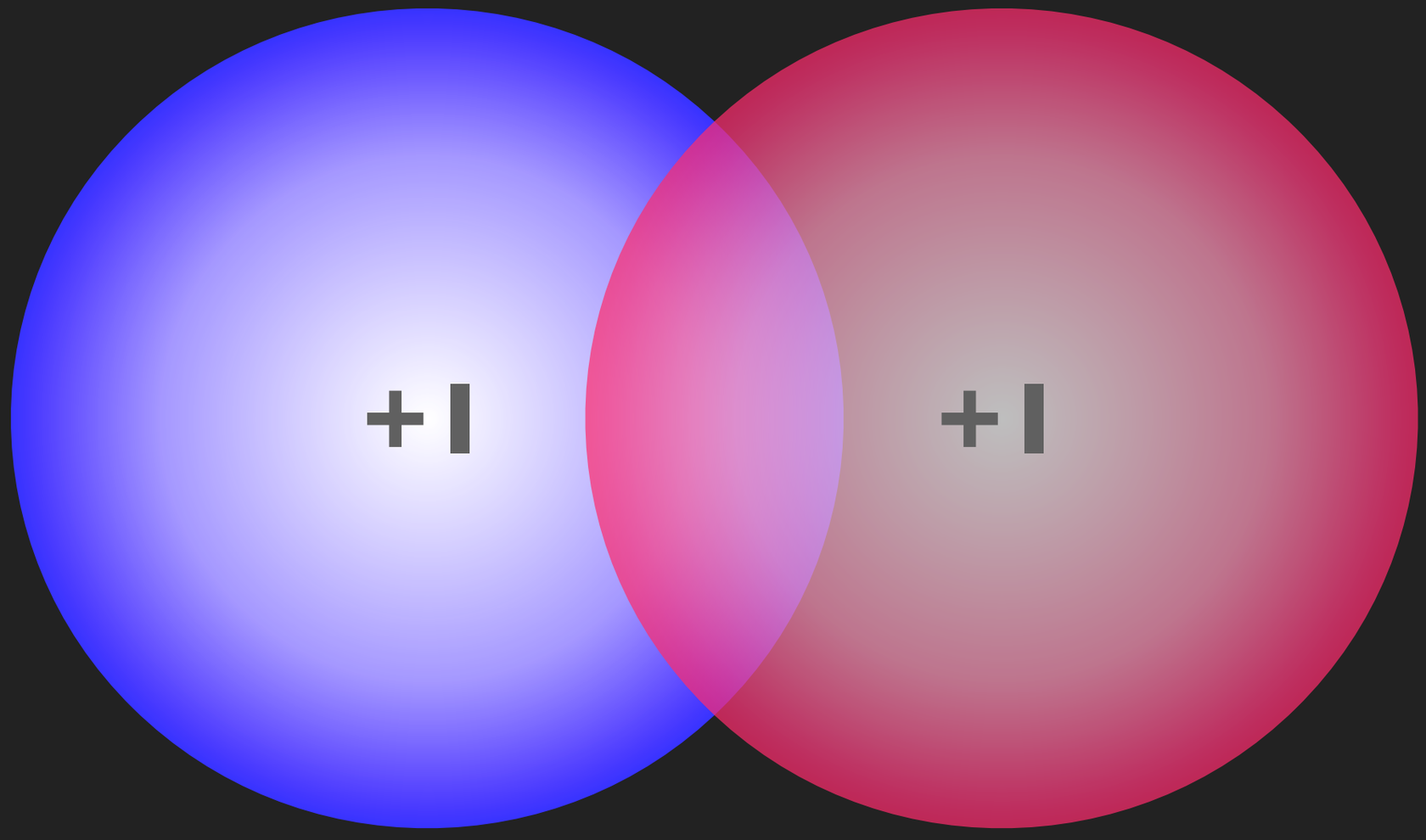
Jane Doe's Friends



Friends of Friends

John Doe's Friends

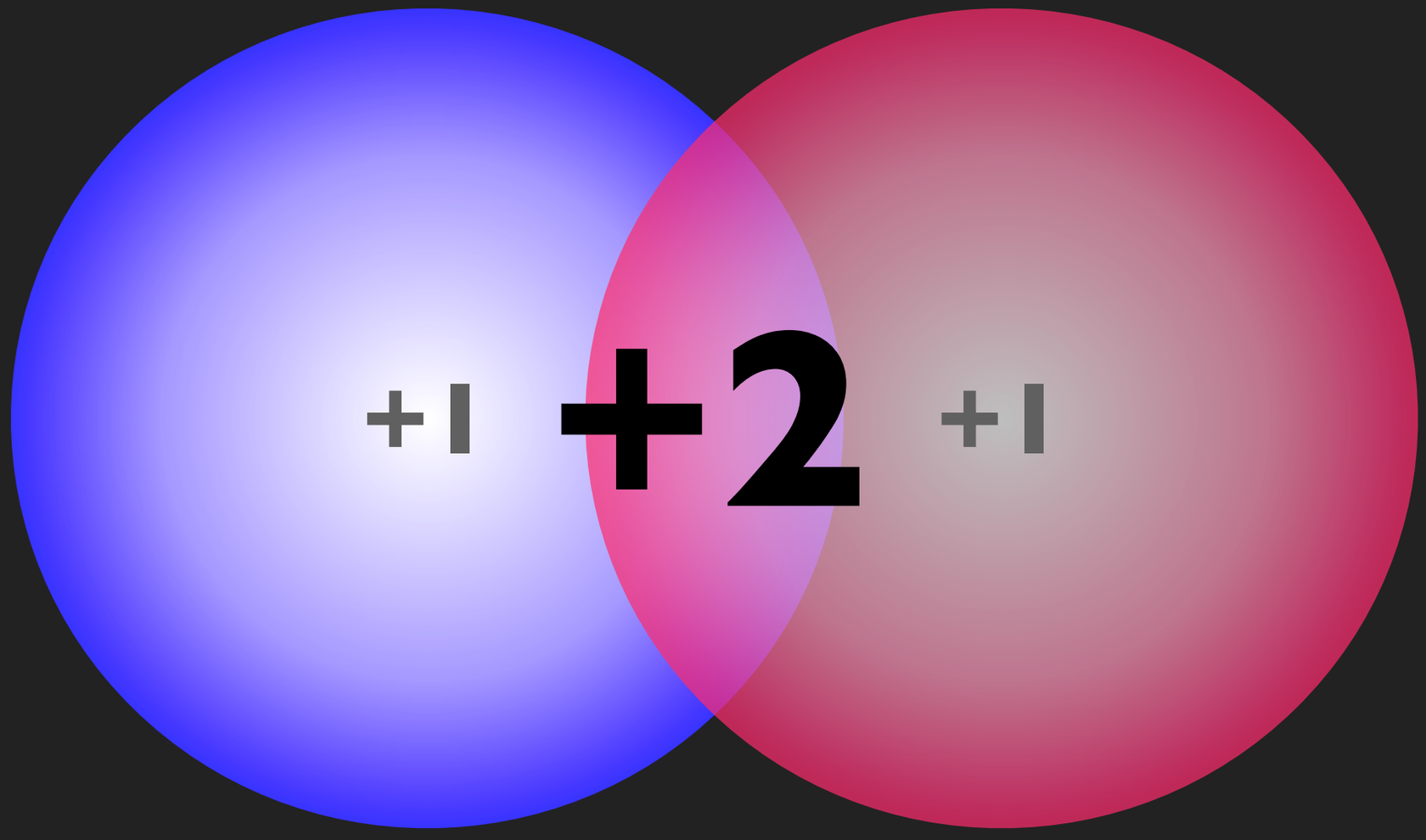
Jane Doe's Friends



Friends of Friends

John Doe's Friends

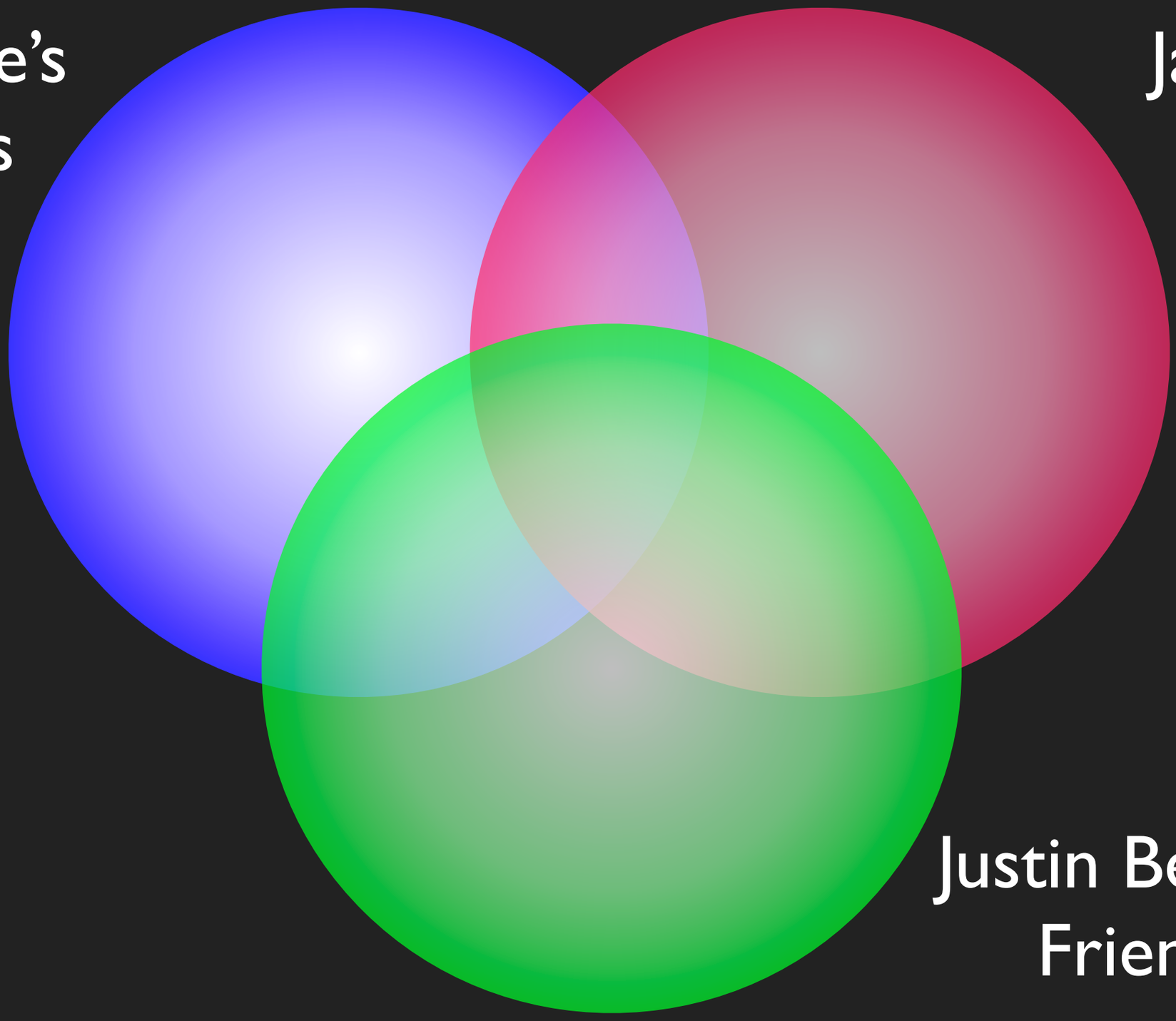
Jane Doe's Friends



Friends of Friends

John Doe's Friends

Jane Doe's Friends



Justin Beiber's Friends

Friends of Friends

John Doe's Friends

Jane Doe's Friends

+1

+2

+1

+3

+2

+2

+1

Justin Beiber's Friends

How do we do that in our App?

- Memcache
- Redis
- Street Smarts

(Making it rain friendships)

Memcache

- *Key value store (NOSQL)*
- *Holds ruby objects*
- *Extremely fast*
- *volatile*
- *Used to (mem)cache database queries*

```
> CACHE.set("foo", "bar")
=> nil
> CACHE.get("foo")
=> "bar"
```

Memcache

Key Value Store

```
> CACHE.set("foo", "bar")
> CACHE.get("foo")
=> "bar"
```

```
> user = User.where(:username => “Schneems”).first
> CACHE.set("foo", user)
> CACHE.get("foo")
=> #<User id: 263595 ... >
```

Marshals Ruby Objects

(like a boss)

Memcache

Is Volitile

Memcache Pattern

get||Set

```
class Memcached

 def get_or(key, timeout=0, marshal=true, &block)
   get(key, marshal)
   rescue Memcached::NotFound => e
     if block_given?
       value = block.call
       set(key, value, timeout, marshal) rescue nil
       value
     else
       raise e
     end
 end

end
```

gets the value || makes the query and sets the cache

Memcache - Speed Tips

- *Cache frequently queried objects (get users)*

```
class User < ActiveRecord::Base
  def self.get(username)
    CACHE.get_or("user:#{username}") do
      User.where(:username => "#{username}").first
    end
  end
end

>> User.get("schneems")
=> #<User id: 263595 ... >
```

Memcache:
(I wanna Go Fast)


```
# Database
> Benchmark.measure do
    User.where(:username => "schneems").first
  end.real
=> 0.09 s

# Memcache
> Benchmark.measure do
    User.get("schneems")
  end.real
=> 0.0009 s
```

oh yeah

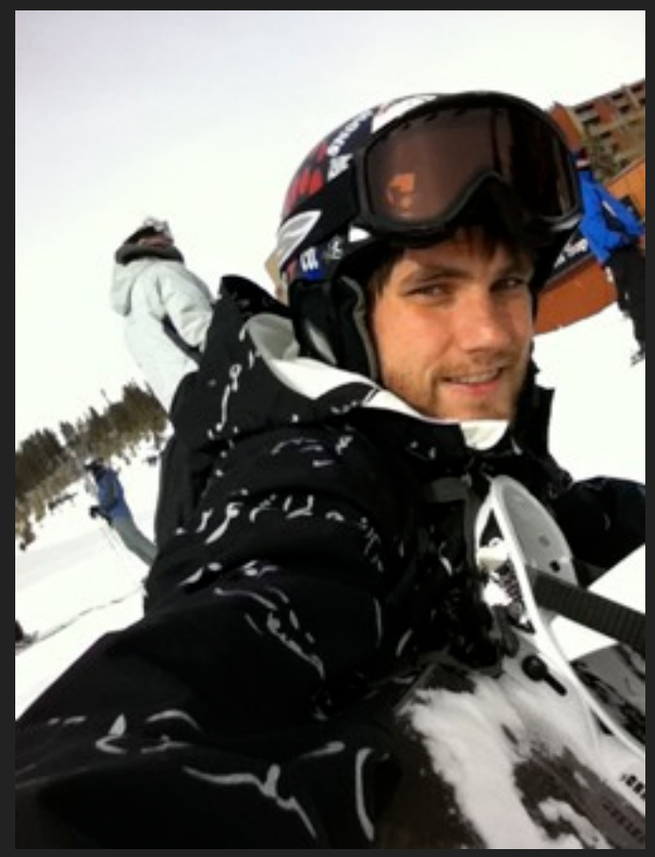
Memcache - Speed Tips

- Cache expensive queries (Friend IDs)

```
class User < ActiveRecord::Base
  def get_friend_ids
    CACHE.get_or("friend_ids:#{self.id}") do
      friends = Friendships.where(:user_1_id => self.id)
      friends.map {|friend| friend.user_2_id }
    end
  end
end

>> u = User.get("schneems")
>> u.get_friend_ids
=> [1,2,3,...]
```

Memcache



- Key/Value store (NOSQL)
- Holds ruby objects
- **Extremely** fast
- volatile
- Can be used to cache expensive database queries

```
> CACHE.set("foo", "bar")
=> nil
> CACHE.get("foo")
=> "bar"
```

Redis

- Key value store (NOSQL: yes another one)
  - does NOT store ruby objects...boo :' (
    - Native data structures...yay :)
- pretty-darn fast
- non-volatile!!!!!!!!!!
- Used to easily store values

Redis

keys & values & stores

```
> REDIS.set("chunky", "bacon")
=> "OK"
> REDIS.get("chunky")
=> "bacon"
```

http://jimneath.org/2011/03/24/using-redis-with-ruby-on-rails.html

Redis - Datatypes

- String
- Set

  (collection of strings with no duplicates)
- Sorted Set
- List

  (sorted collection of strings like and array)
- Hash

http://jimneath.org/2011/03/24/using-redis-with-ruby-on-rails.html

Redis - Lists

keys & values & stores

```
> REDIS.lpush("lists:are:like:an:array", 1)
=> 1
> REDIS.lpush("lists:are:like:an:array", 2)
=> 2
> REDIS.lpush("lists:are:like:an:array", 3)
=> 3
> REDIS.lrange("lists:are:like:an:array", 0, -1)
=> ["3", "2", "1"]
> REDIS.lrange("lists:are:like:an:array", 0, -1).map(&:to_i)
=> [3, 2, 1]
```

http://jimneath.org/2011/03/24/using-redis-with-ruby-on-rails.html

Redis - Lists

Store “ignored” potential friends

```
def ignore_potential_friend(friend_id)
    REDIS.lpush(ignore_facebook_friend_key, friend_id)
end
```

Retrieve array of ignored IDs

```
def ignored_potential_friend_ids
    REDIS.lrange(not_potential_friend_ids_key, 0, -1).map(&:to_i)
end
```

Redis - Recap

- Key value store (yes another one)
  - does NOT store ruby objects...boo :' (
  - Native data structures...yay :)
- pretty-darn fast
- non-volatile!!!!!!!!!!
- Used to easily store values

Redis - Extra Credit

Getting Started

http://jimneath.org/2011/03/24/using-redis-with-ruby-on-rails.html

Resque

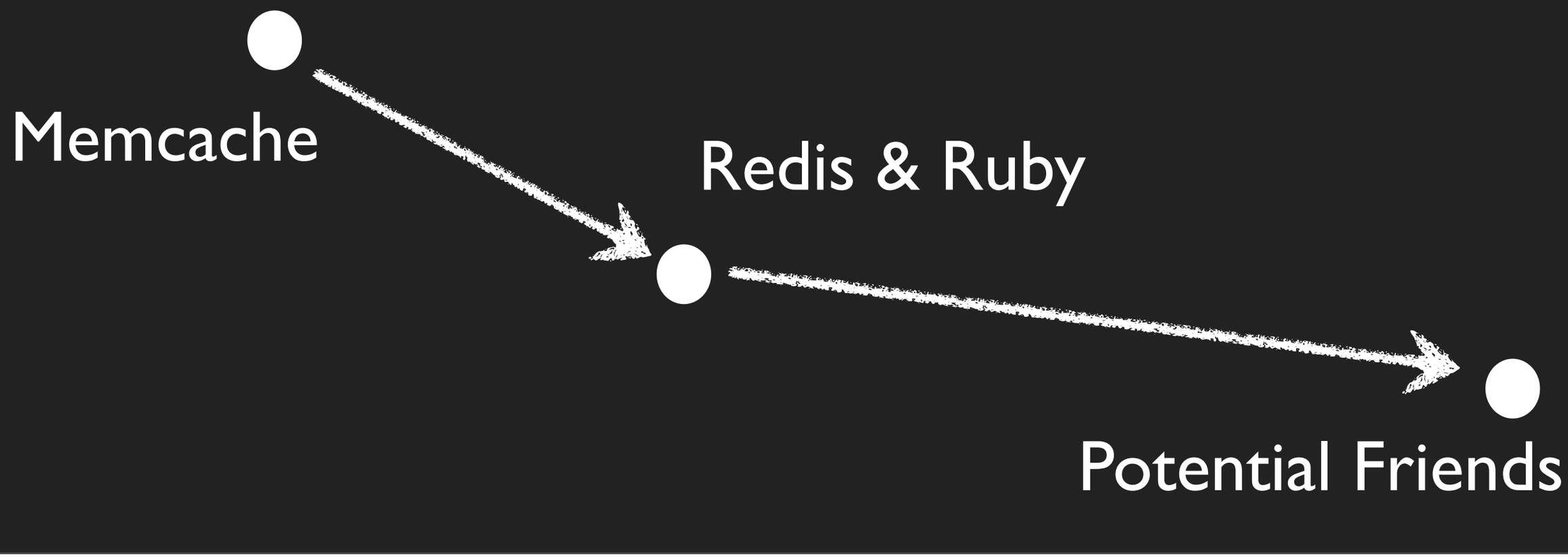
Background jobs with Redis (and style)

https://github.com/defunkt/resque

redis

Street Smarts

- *Ruby array 'tricks'*
- *Connecting the dots*



Street Smarts

Ruby array Manipulation

Intersection (&)

```
> [1,2,3] & [3,4,5]
=> [3]
> [1,2,3,4] & [3,4,5]
=> [3,4]
```

Difference (-)

```
> [1,2,3] - [3,4,5]
=> [1,2,4,5]
> [1,2,3,4] - [3,4,5]
=> [1,2,5]
```

Street Smarts

Ruby array Manipulation

Concatenation (+)

```
> [1,2] + [3,4,5]
=> [1,2,3,4,5]
```

Street Smarts

Ruby array Manipulation

inject - turning arrays into other stuff

```
[5,6,7,8,9,10].inject(0) {|sum, n| sum + n }
=> 45
```

Street Smarts

Ruby array Manipulation

inject - turning arrays into other stuff

```
[5,6,7,8,9,10].inject(0) {|sum, n| sum + n }
=> 45
```

inject - build hashes from arrays

```
> array = [5,6,7,8,9,10]
> array.inject({}) {|hash, id| hash[id.to_s] = 1; hash }
=> {"6"=>1, "7"=>1, "8"=>1, "9"=>1, "10"=>1, "5"=>1}
```

inject - build hashes from arrays

```
> array.inject({}) {|hash, id|
                    puts "hash: #{hash.inspect}"
                    puts "id: #{id}"
                    hash[id.to_s] = 1;
                    hash }
hash: {}
id: 5
hash: {"5"=>1}
id: 6
hash: {"6"=>1, "5"=>1}
id: 7
hash: {"6"=>1, "7"=>1, "5"=>1}
id: 8
hash: {"6"=>1, "7"=>1, "8"=>1, "5"=>1}
id: 9
hash: {"6"=>1, "7"=>1, "8"=>1, "9"=>1, "5"=>1}
id: 10
=> {"6"=>1, "7"=>1, "8"=>1, "9"=>1, "10"=>1, "5"=>1}
```

inject - build hashes from arrays

```
> array.inject({}) {|hash, id|
                    puts "hash: #{hash.inspect}"
                    puts "id: #{id}"
                    hash[id.to_s] = 1;
                    hash }
hash: {}
id: 5
hash: {"5"=>1}
id: 6
hash: {"6"=>1, "5"=>1}
id: 7
hash: {"6"=>1, "7"=>1, "5"=>1}
id: 8
hash: {"6"=>1, "7"=>1, "8"=>1, "5"=>1}
id: 9
hash: {"6"=>1, "7"=>1, "8"=>1, "9"=>1, "5"=>1}
id: 10
=> {"6"=>1, "7"=>1, "8"=>1, "9"=>1, "10"=>1, "5"=>1}
```

inject - build hashes from arrays

```
> array.inject({}) {|hash, id|
                    puts "hash: #{hash.inspect}"
                    puts "id: #{id}"
                    hash[id.to_s] = 1;
                    hash }
hash: {}
id: 5
hash: {"5"=>1}
id: 6
hash: {"6"=>1, "5"=>1}
id: 7
hash: {"6"=>1, "7"=>1, "5"=>1}
id: 8
hash: {"6"=>1, "7"=>1, "8"=>1, "5"=>1}
id: 9
hash: {"6"=>1, "7"=>1, "8"=>1, "9"=>1, "5"=>1}
id: 10
=> {"6"=>1, "7"=>1, "8"=>1, "9"=>1, "10"=>1, "5"=>1}
```

inject - *build hashes from arrays*

```
> array.inject({}) {|hash, id|
                    puts "hash: #{hash.inspect}"
                    puts "id: #{id}"
                    hash[id.to_s] = 1;
                    hash }
hash: {}
id: 5
hash: {"5"=>1}
id: 6
hash: {"6"=>1, "5"=>1}
id: 7
hash: {"6"=>1, "7"=>1, "5"=>1}
id: 8
hash: {"6"=>1, "7"=>1, "8"=>1, "5"=>1}
id: 9
hash: {"6"=>1, "7"=>1, "8"=>1, "9"=>1, "5"=>1}
id: 10
=> {"6"=>1, "7"=>1, "8"=>1, "9"=>1, "10"=>1, "5"=>1}
```

inject - *build hashes from arrays*

```
> array.inject({}) {|hash, id|
                    puts "hash: #{hash.inspect}"
                    puts "id: #{id}"
                    hash[id.to_s] = 1;
                    hash }
hash: {}
id: 5
hash: {"5"=>1}
id: 6
hash: {"6"=>1, "5"=>1}
id: 7
hash: {"6"=>1, "7"=>1, "5"=>1}
id: 8
hash: {"6"=>1, "7"=>1, "8"=>1, "5"=>1}
id: 9
hash: {"6"=>1, "7"=>1, "8"=>1, "9"=>1, "5"=>1}
id: 10
=> {"6"=>1, "7"=>1, "8"=>1, "9"=>1, "10"=>1, "5"=>1}
```

inject - *build hashes from arrays*

```
> array.inject({}) {|hash, id|
                    puts "hash: #{hash.inspect}"
                    puts "id: #{id}"
                    hash[id.to_s] = 1;
                    hash }
hash: {}
id: 5
hash: {"5"=>1}
id: 6
hash: {"6"=>1, "5"=>1}
id: 7
hash: {"6"=>1, "7"=>1, "5"=>1}
id: 8
hash: {"6"=>1, "7"=>1, "8"=>1, "5"=>1}
id: 9
hash: {"6"=>1, "7"=>1, "8"=>1, "9"=>1, "5"=>1}
id: 10
=> {"6"=>1, "7"=>1, "8"=>1, "9"=>1, "10"=>1, "5"=>1}
```

inject - build hashes from arrays

```
> array.inject({}) {|hash, id|
                    puts "hash: #{hash.inspect}"
                    puts "id: #{id}"
                    hash[id.to_s] = 1;
                    hash }
hash: {}
id: 5
hash: {"5"=>1}
id: 6
hash: {"6"=>1, "5"=>1}
id: 7
hash: {"6"=>1, "7"=>1, "5"=>1}
id: 8
hash: {"6"=>1, "7"=>1, "8"=>1, "5"=>1}
id: 9
hash: {"6"=>1, "7"=>1, "8"=>1, "9"=>1, "5"=>1}
id: 10
=> {"6"=>1, "7"=>1, "8"=>1, "9"=>1, "10"=>1, "5"=>1}
```

inject - build hashes from arrays

```
> array.inject({}) {|hash, id|
                    puts "hash: #{hash.inspect}"
                    puts "id: #{id}"
                    hash[id.to_s] = 1;
                    hash }
hash: {}
id: 5
hash: {"5"=>1}
id: 6
hash: {"6"=>1, "5"=>1}
id: 7
hash: {"6"=>1, "7"=>1, "5"=>1}
id: 8
hash: {"6"=>1, "7"=>1, "8"=>1, "5"=>1}
id: 9
hash: {"6"=>1, "7"=>1, "8"=>1, "9"=>1, "5"=>1}
id: 10
=> {"6"=>1, "7"=>1, "8"=>1, "9"=>1, "10"=>1, "5"=>1}
```

inject - *build hashes from arrays*

```
> array.inject({}) {|hash, id|
                    puts "hash: #{hash.inspect}"
                    puts "id: #{id}"
                    hash[id.to_s] = 1;
                    hash }
hash: {}
id: 5
hash: {"5"=>1}
id: 6
hash: {"6"=>1, "5"=>1}
id: 7
hash: {"6"=>1, "7"=>1, "5"=>1}
id: 8
hash: {"6"=>1, "7"=>1, "8"=>1, "5"=>1}
id: 9
hash: {"6"=>1, "7"=>1, "8"=>1, "9"=>1, "5"=>1}
id: 10
=> {"6"=>1, "7"=>1, "8"=>1, "9"=>1, "10"=>1, "5"=>1}
```

"the best method in Ruby...ever"
- Adam Keys

Street Smarts

Connect the Dots

Street Smarts

Connect the Dots

- *Get a user*
- *Get their friends*
- *Get their friends-friends (Friends of a Friend)*
- *Remove current friend IDs and ignored IDs*
- *Mutual FOAFs counts (Friends of a Friend) are rank*
- *Sort by rank, and suggest to the user*

Friend IDs

Connect the Dots

Get friend's IDs

```
> richard = User.get("Schneems")
> richard.get_friend_ids
=>[1, 2] # [john, jane]
```

Get your friend's friend IDs

```
> friends_of_a_friend_ids = []
> richard.get_friend_ids do |friend_id|
>  friends_of_a_friend_ids << User.get_someones_friend_ids(friend_id)
> end.flatten
=>[3, 4, 5, 6, 7, 3, 4, 5, 8, 9, 10]
  # john and jane's friend id's
```

FOAF IDs

Friend of a Friend Ids

```
> friends_of_a_friend_ids = []
> richard.get_friend_ids do |friend_id|
>  friends_of_a_friend_ids << User.get_someones_friend_ids(friend_id)
> end.flatten
=>[3, 4, 5, 6, 7, 3, 4, 5, 8, 9, 10]
  # john and jane's friend id's
```

What does this give us?

```
=>[3, 4, 5, 6, 7, 3, 4, 5, 8, 9, 10]
```

(hint, its potential friend IDs)

Remove Bad Suggestions

- Don't suggest
  - self
  - Pending or accepted friend request
  - Previously ignored potential friends

```
def not_potential_friend_ids
  @not_potential_friend_ids ||=
    CACHE.get_or("users:not_potential_friends_key:#{self.id}", 24.hours) {
      [self.id] +
      Friendship.where(:user_1_id => self.id).map(&:user_2_id) +
      ignored_potential_friend_ids
    }
end
```

Put it all Together

Sooooooo... close

```
def potential_friend_ids(options = {})
  potential_friend_ids = friends_of_a_friend_ids - not_potential_friend_ids
  potential_ids_with_score = potential_friend_ids.
                             inject({}) {|sorted_id_hash, friend_id|
                                         if sorted_id_hash[friend_id].blank?
                                            sorted_id_hash[friend_id] = 1
                                         else
                                            sorted_id_hash[friend_id] += 1
                                         end
                                         sorted_id_hash }
  sorted_potential_friend_ids = potential_ids_with_score.sort {|x, y|
                                        y.last <=> x.last}.
                                        map(&:first)
  return sorted_potential_friend_ids
end
```

Put it all Together

It works!!!!!!

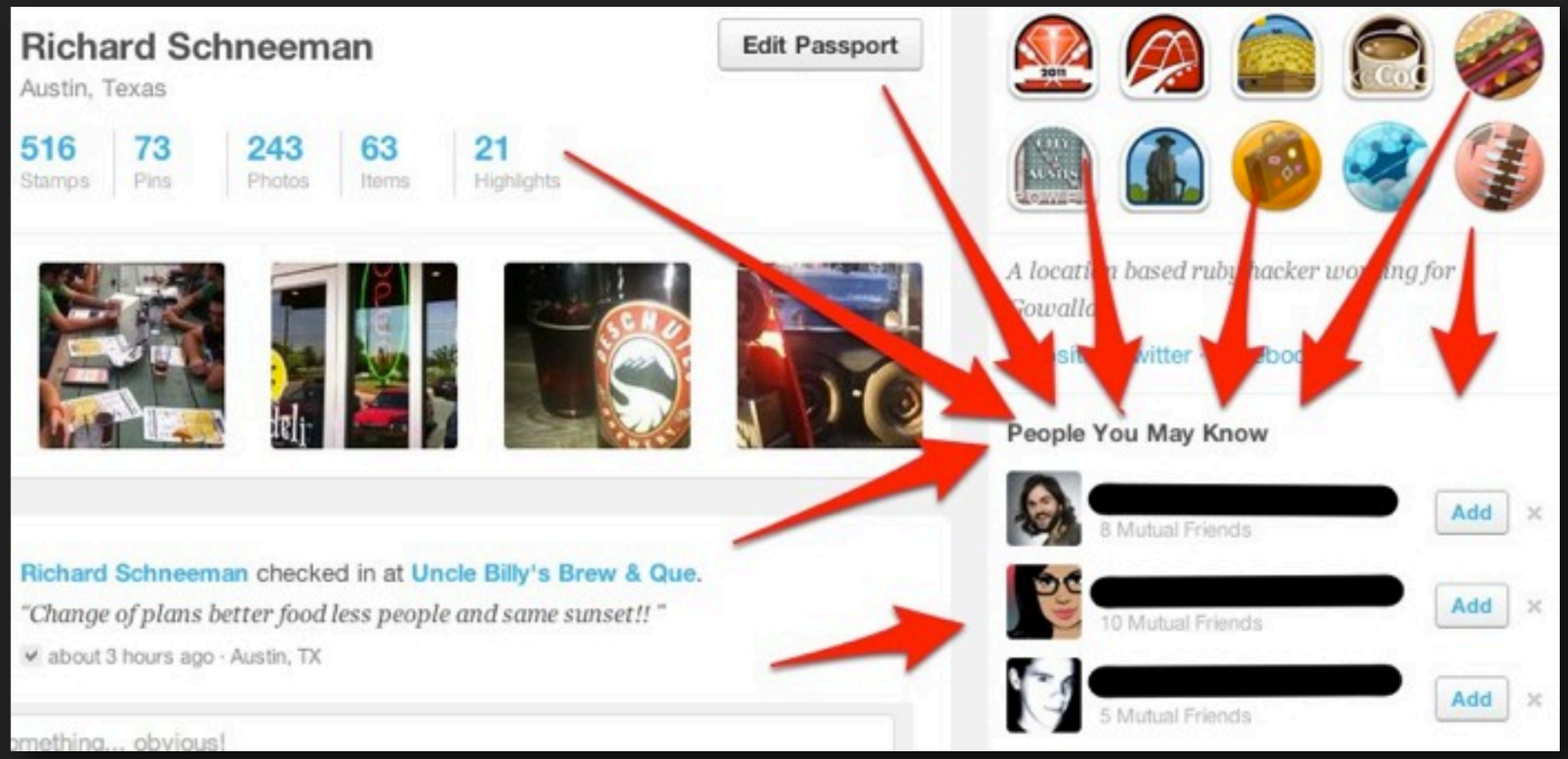
```
def potential_friends(options = {})
    # TODO add memcached to this call in a production app
    User.where("id in (?)", potential_friend_ids.
         first(options[:limit])).shuffle
end
```

Boom!!

```
> User.get(“schneems”).potential_friends(:limit => 2)
> [#<User id: 3 ... >, #<User id: 5 ... >]
```

Goes the dynamite

IT WORKS!!



From	Subject	Da
Christopher	Christopher wants to be your friend	Ma
Jeremy	Jeremy wants to be your friend	Ma
Katie	Katie wants to be your friend	Ma
Tyler	Tyler wants to be your friend	Ma
Adrian	Adrian wants to be your friend	Ma
Chris	Chris wants to be your friend	Ma

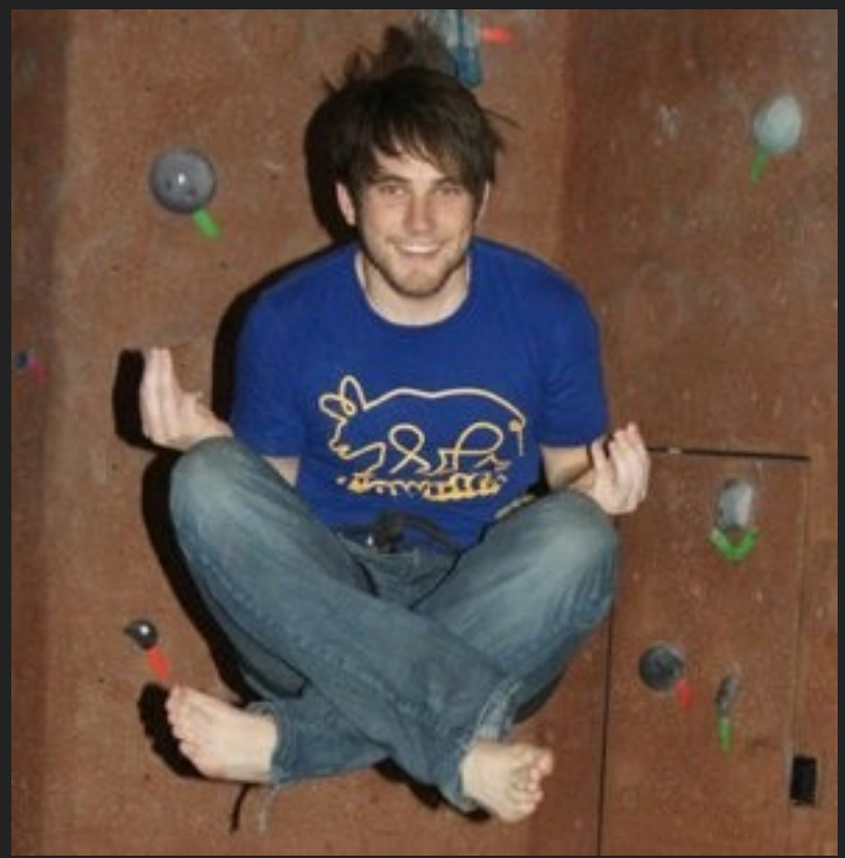
Try It Out

- Sign up on Gowalla
  - http://gowalla.com/join
- Add friends or connect facebook
- Visit your passport
- Add and ignore friends to your content

Thats100% Pure Crunk Juice

Sunday, June 5, 2011

Questions?



Twitters: @schneems

github.com/schneems