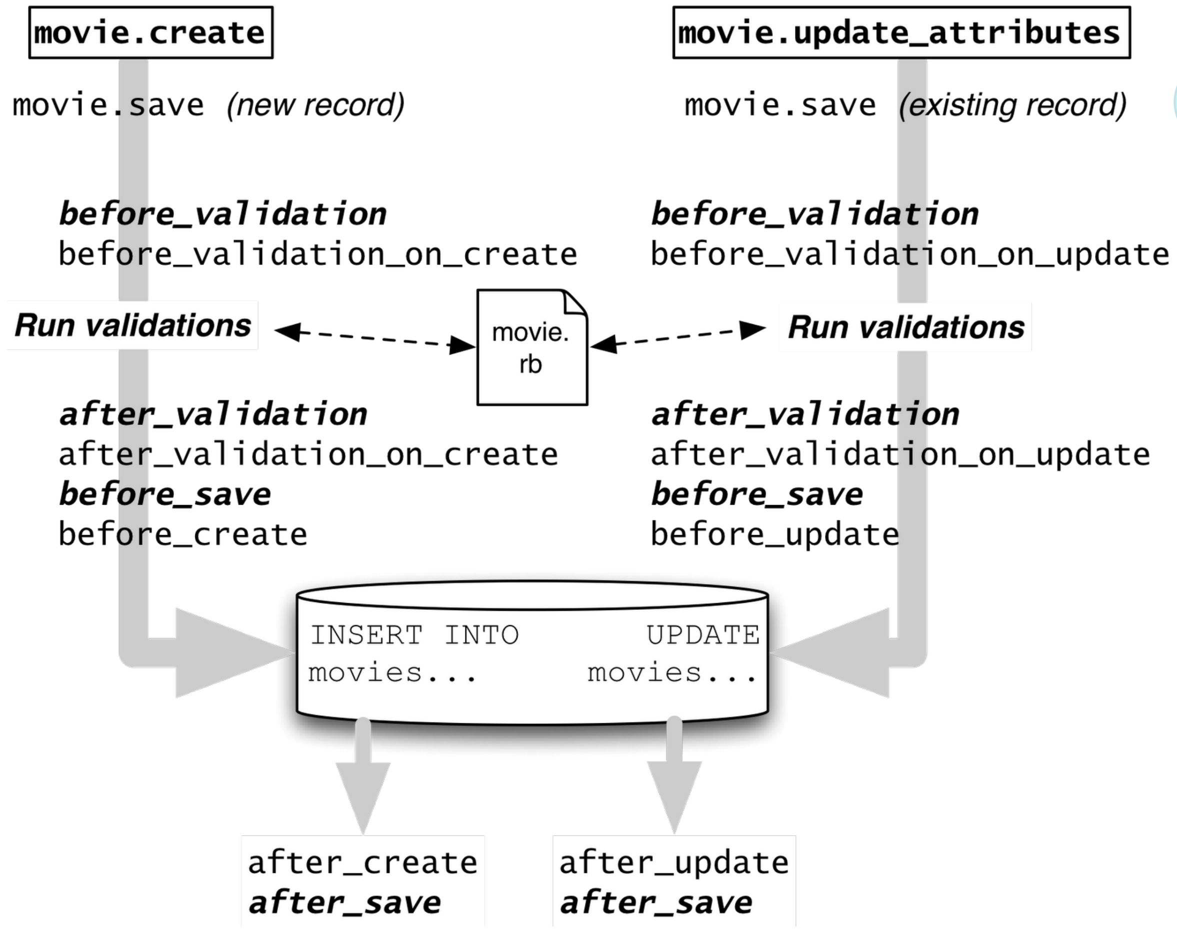
Validations vs. Filters

	Validation	**Filter**
Advice (DRYness)	Check invariants on model	Check conditions for allowing controller action to run
Pointcut	AR model lifecycle hooks	Before and/or after any public controller method
Can change execution flow?	No	Yes
Can define *advice* in arbitrary function?	Yes; shortcuts provided for common cases	Yes, must provide function
Info about errors?	Each model object has associated errors object	Capture in flash[], session[], or instance variable

Model Lifecycle Callbacks

Allows Pre and Post Operations



Validation automatically happens here

- or when you call valid?
- if fail, save will fail

model.errors is an ActiveRecord::Errors object with cool behaviors of its own

- See Screencast 7.1.1

Summary so far

- Aspect-oriented programming is a way of DRYing out cross-cutting concerns
- Ruby doesn't have fully-general AOP, but Rails provides some "predefined" pointcuts
- Validations check or assert pre/post conditions at key points during model lifecycle
- Controller filters check or assert pre/post conditions related to controller actions
- and can change control flow (redirect, render)
- Partials DRY out views (though not AOP)

Which Ruby language features support the DRYness enabled by validations & filters: (a) higher-order functions, (b) closures, (c) metaprogramming

- ☐ Only (a)
- ☐ Only (a) & (b)
- ☐ Only (a) & (c)
- ☐ (a), (b) and (c)

Single Sign-On and Third-Party Authentication *(ELLS §7.2)*

Armando Fox

Who are you and what are you doing here?

- Authentication: prove you are who you say
- Username & “secret” password
- Hold private key that matches public key
- Possess cryptographic certificate signed by a trusted third party
- Authorization: prove you are allowed to do what you’re asking
- does system record you as having privilege?
- do you have a “token” or “capability” that lets you do something?

Web 1.0

•Every site has separate passwords

•Most sites had no RESTful API, so had to actually "log in" (or simulate it)

•Doesn't work for SOA!

•Hard for services to cooperate if you need to login interactively to every service, every time

•Desired solution: single-sign-on (SSO)

•*But…don't want to reveal service A password to service B*

Third-party authentication

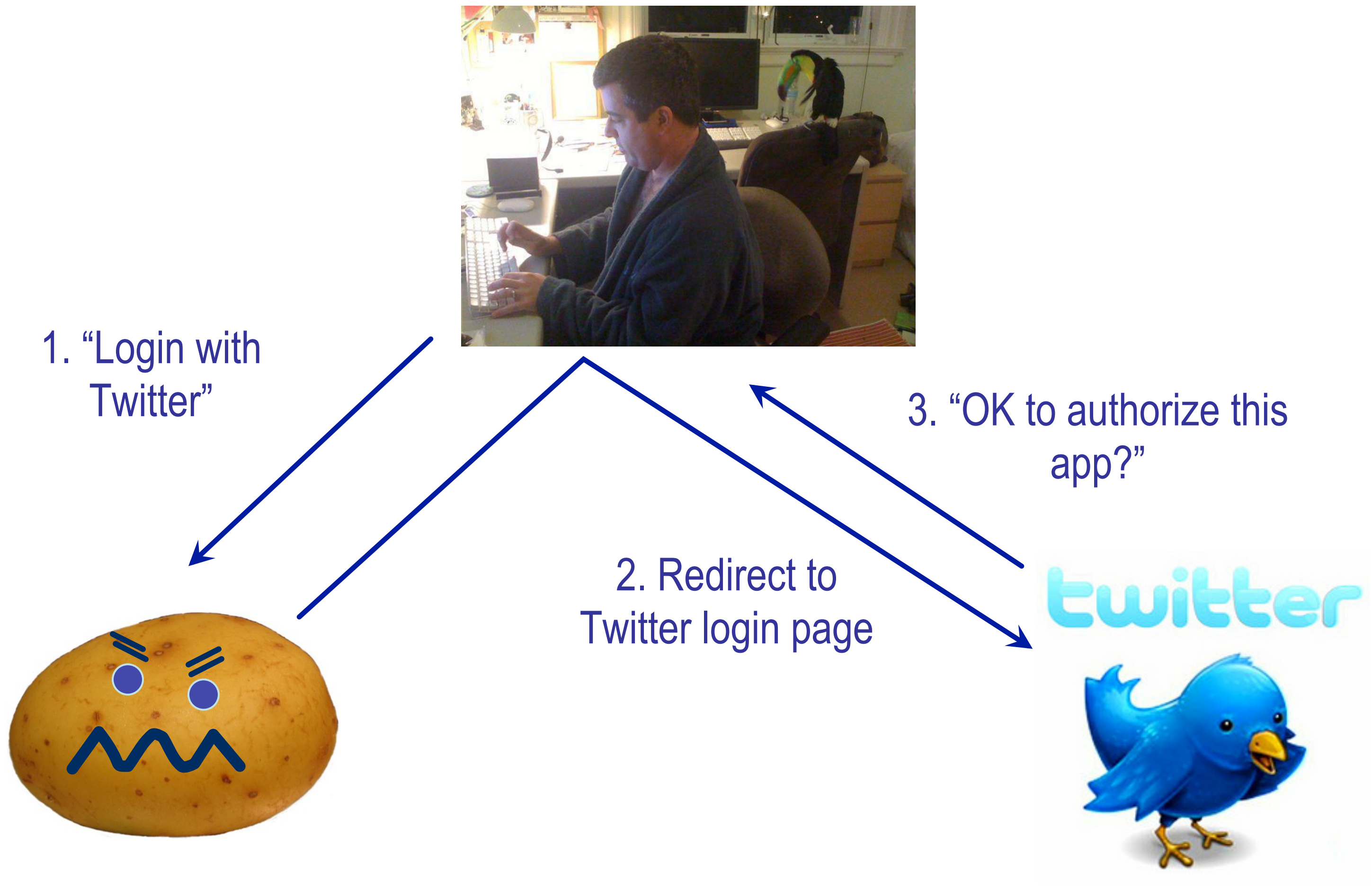
- Service A knows something about your identity
- Want to use this info with service B, *without* revealing Service A credentials
- Service A: *authentication provider* (though may also do other things)
- Auth only: OpenID, Kerberos
- Auth + other stuff: Twitter, Facebook, Google Apps, …

Logos shown for educational purposes only and are the intellectual property of their owners.

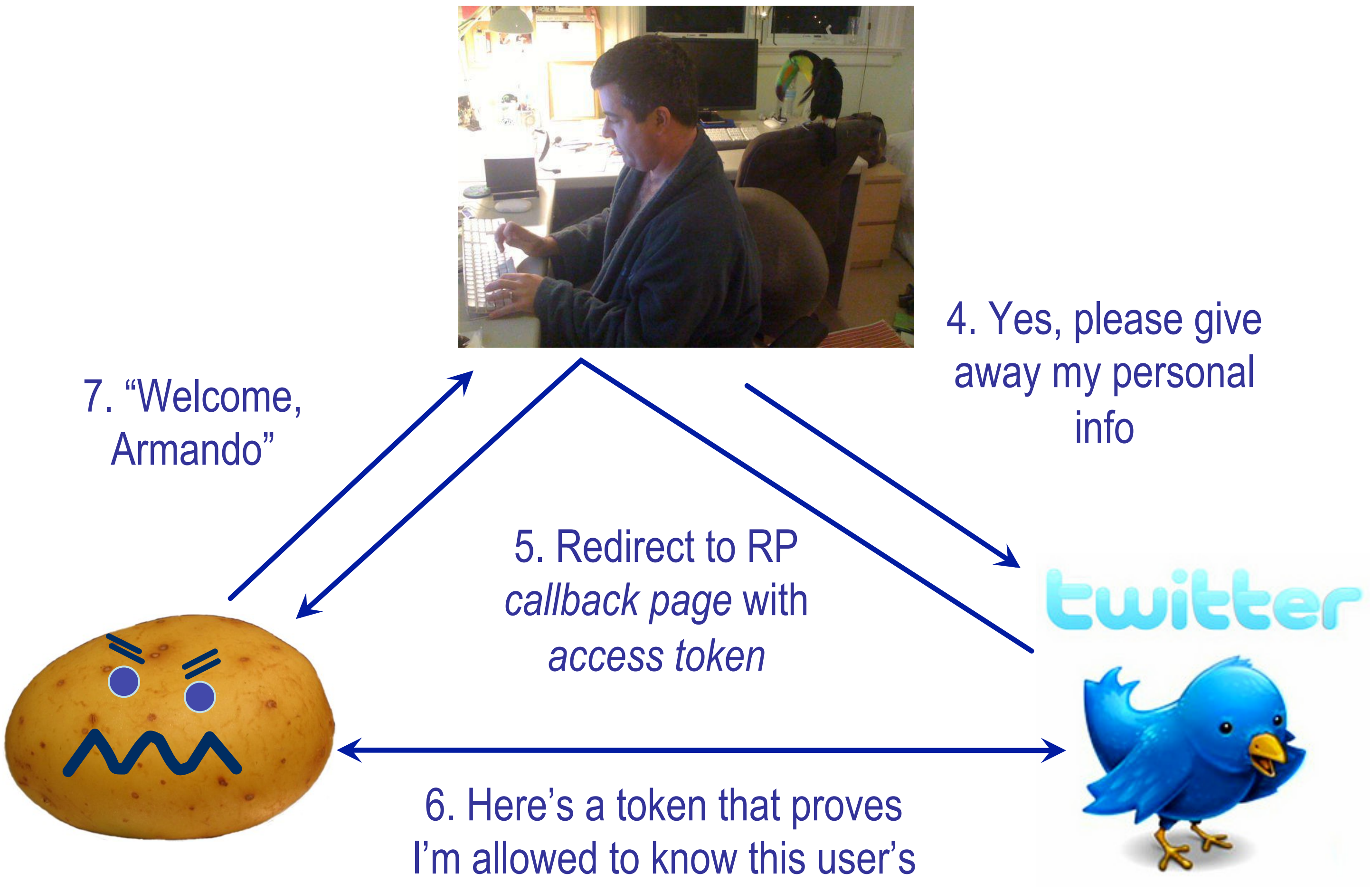
How does it work? (concepts)

- Building block: *tamper-evident secure token*
- Using cryptography, I create a string that:
  - Only I can decrypt (decode)
  - I can detect if it's been tampered with
  - No one else could have created it without knowing my secret key
- Usually, string just contains a "handle" to valuable info that I store myself
  - Receive string => I know I can "trust" the handle

Third-Party Auth with Twitter & RottenPotatoes


1. "Login with Twitter"
2. Redirect to Twitter login page
3. "OK to authorize this app?"
twitter

Third-Party Auth with Twitter & RottenPotatoes



4. Yes, please give away my personal info

7. “Welcome, Armando”

5. Redirect to RP *callback page* with *access token*

6. Here’s a token that proves I’m allowed to know this user’s name

How does it work? (MVC)

- Model *session* as its own entity
- *session controller* creates and deletes session, handles interaction with auth provider
- Once user is authenticated, we need a local *users* model to represent him/her
- session[] remembers primary key (ID) of "currently authenticated user"
- OmniAuth gem helps a lot by providing uniform API to different "strategies"

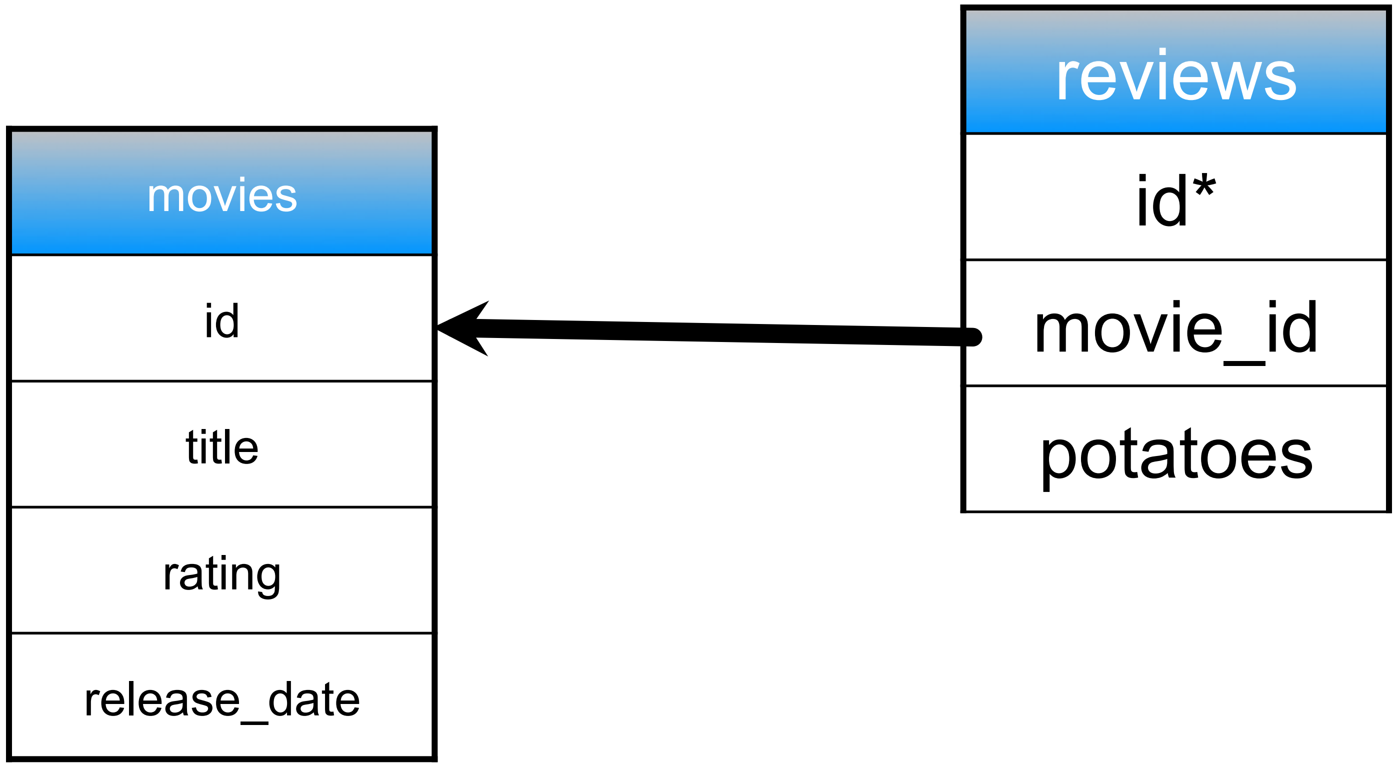
Which is **true** about third-party authentication between a requester and a provider?

- ☐ Once completed, the requester can do anything *you* can do on the provider
- ☐ If your credentials on the requester are compromised, your credentials on the provider are also compromised
- ☐ If the provider revokes access, the requester no longer has any of your info
- ☐ Access can be time-limited to expire on a pre-set date

*

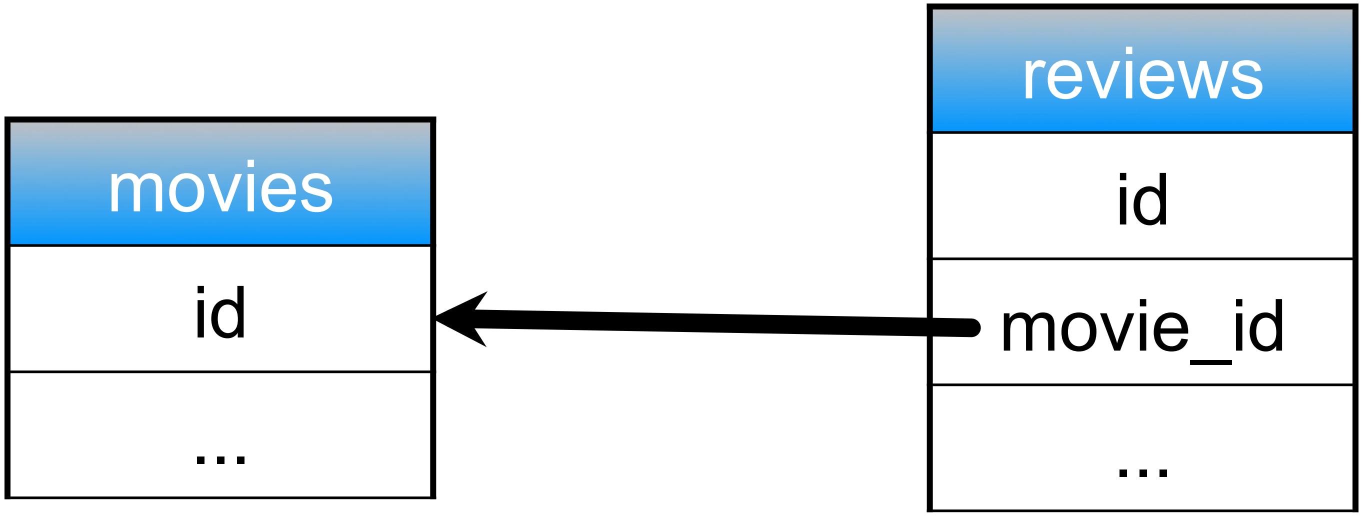
Expressing “Has Many” in terms of Relational DB model

- foreign key (FK) in one table **refers** to the primary key of another table



Databases 101

- joins are queries that **combine** records from 2 or more tables using PKs and FKs



SELECT *
FROM movies, reviewsWHERE movie
reviews.movie_id

Cartesian product

Cartesian Product

table 'artists'

id	name
10	Justin
11	Shakira
12	Britney

table 'reviews'

id	desc	artist_id
30	"Terrible"	12
31	"Passable"	11
32	"Please"	10

Cartesian product: artists JOIN reviews

artists.id	artists.name	reviews.id	reviews.desc	reviews.artist_id
10	Justin	30	"Terrible"	12
10	Justin	31	"Passable"	11
10	Justin	32	"Please"	10
11	Shakira	30	"Terrible"	12
11	Shakira	31	"Passable"	11
11	Shakira	32	"Please"	10

ActiveRecord Associations

- allows manipulating DB-managed associations more Rubyistically
- after setting things up correctly, you don't have to worry (much) about keys and joins

```
class Movie < ActiveRecord::Base
  has_many :reviews
end
class Review < ActiveRecord::Base
  belongs_to :movie
end
```

"The foreign key *belongs to* me"

Basic idea…

- reviews table gets a *foreign key* (FK) field that has primary key of Movie a review is for
- Dereference movie.reviews == perform database join to find reviews where movie_id == movie.id
- Dereference review.movie == look up the *one* movie whose PK id == review.movie_id
- Note! must add FK fields using a migration!

http://pastebin.com/hfvramxQ

Association *proxy* methods

- Now you can say: `@movie.reviews` *# Enumerable of reviews*
- And also go the other way: `@review.movie` *# what movie is reviewed?*
- You can add new reviews for a movie:

```
@movie = Movie.where("title='Fargo'")
@movie.reviews.build(:potatoes => 5)
@movie.reviews.create(:newspaper=>'Chron', ...)
          # how are these different from just new() & create()?
@movie.reviews << @new_review
# instantly updates @new_review's FK in database!
@movie.reviews.find(:first,:conditions => '...')
```

How does it work?

- Models involved in an association are expected to have attribute for foreign key of owning object
  - e.g., movie_id in reviews table
- ActiveRecord manages this field in both database & in-memory AR object
- Don’t manage it yourself!
  - Harder to read
  - May break if database schema doesn’t follow Rails conventions

How does it work?

- How are table names and column names determined from has_many ?

- What if we omit the belongs_to ?

- Are Associations implemented as a class or a Module?

Rails Cookery #4

To add a one-to-many association:

1. Add has_many to *owning side* and belongs_to to *owned side* model files
2. Create migration to add foreign key to *owned side* that references *owning side*
3. Apply migration
4. rake db:test:prepare to regenerate test database schema

Which of these, if any, is NOT a correct way of saving a new association, given m is an *existing* movie:

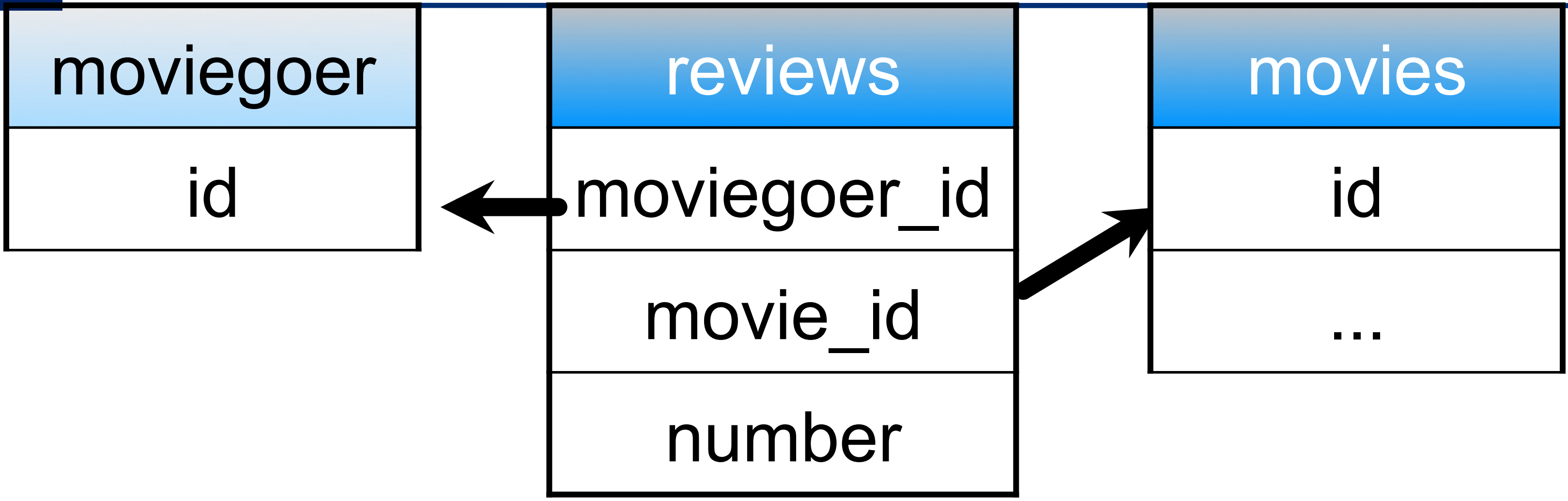
- ☐ Review.create!(:movie_id=>m.id, :potatoes=>5)
- ☐ r = m.reviews.build(:potatoes => 5)
  r.save!
- ☐ m.reviews << Review.new(:potatoes=>5)
  m.save!
- ☐ All will work

Many-to-many associations

- Scenario: Moviegoers rate Movies
  - a moviegoer can have many reviews
  - but a movie can also have many reviews

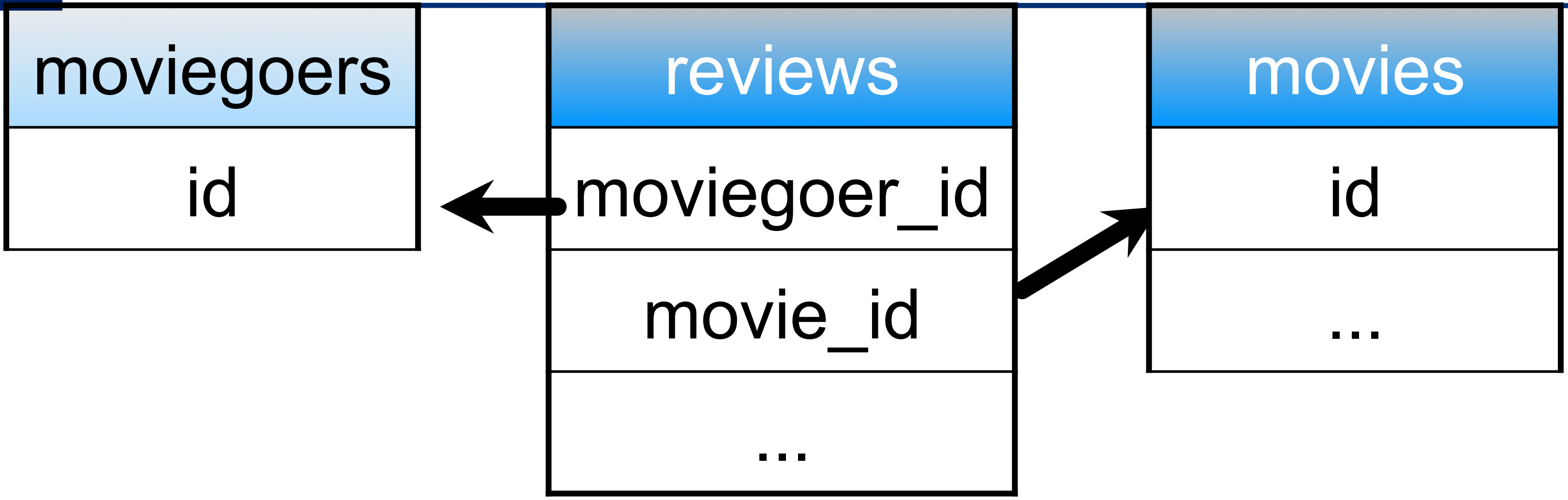
- Why can't we do this with *has_many* and *belongs_to*?
- New approach: create a new AR model to model the *multiple association*

Many-to-many



- moviegoer: has_many :reviews
- movie: has_many :reviews
- review: belongs_to :moviegoer
  belongs_to :movie
- How to get all movies reviewed by some moviegoer?

has_many :through



- moviegoer:
  has_many :reviewshas_many :movies, :through => :reviews
- movie: has_many :reviews
  has_many :moviegoers, :through => :reviews
- reviews: belongs_to :moviegoer
  belongs_to :movie

Through

- Now you can do:

```
@user.movies # movies rated by user
@movies.users # users who rated this movie
```

- My potato scores for R-rated movies

```
@user.reviews.select {
  |r| r.movie.rating == 'R' }
```

In previous example using has_many :through, can we say

@user.movies << movie ?

- ☐ Yes, since it quacks like a collection
- ☐ Yes, since user is the “owning” side of the association
- ☐ Yes, as long as movies table has user_id field
- ☐ No