

identifying concepts

Daniel Jackson · Autodesk, Boston · March 17-18, 2025

an example:
HackerNews

what are the concepts of Hacker News?

▲ Jackson structured programming (wikipedia.org)

Post

106 points by haakonhr 63 days ago | hide | past | favorite | 69 comments

Upvote

▲ danielnicholas 63 days ago [-]

user: danielnicholas

created: 63 days ago

karma: 11

Favorite

You might find helpful an annotated version [0] of Hoare's explanation of JSP that I edited for a Michael Jackson festschrift

; I'd point to these ideas as worth knowing:

ing problem that involves traversing c structures can be solved very systematically. HTDP addresses this class, but bases the structure only on input structure; JSP synthesized it.

Comment

- The **Karma**ne archetypal problems that, however you code, can't be pushed under the rug—most notably structure clashes—and just recognizing them

- Coroutines (or code transformation) let you structure code more cleanly when you need to read or write more than one structure. It's why real iterators (with yield), which offer a limited form of this, are (in my view) better than Java-style iterators with a next method.

- The idea of viewing a system as a collection of asynchronous processes (Ch. 11 in the JSP book, which later became JSD) with a long-running process for each real-world entity. This was a notable contrast to OOP, and led to a strategy (seeing a resurgence with event storming for DDD) that began with events rather than objects.

[0] <https://groups.csail.mit.edu/sdg/pubs/2009/hoare-jsp-3-29-09...>

▲ ob-nix 63 days ago [-]

... this brings back memories! In the late eighties I, as a teenager, found a Jackson Struct. Pr. book at the town library. I remember I was amazed at the text and wondered why I hadn't heard about the method before.

If I remember correctly did the book clearly point out backtracking as a standard method, while mentioning that most languages lacked that, so it had to be implemented manually.

Session

familiar concepts with some creative variation

▲ Jackson structured programming (wikipedia.org)

106 points by haakonhr 63 days ago | hide | past | favorite | 69 comments

▲ danielnicholas 63 days ago [-]

If you want an intro to JSP, you
in 2009.

For those who don't know JSP, I

- There's a class of programming
but bases code structure only o

- There are some archetypal pro
them helps.

- Coroutines (or code transform
iterators (with yield), which offe

- The idea of viewing a system
for each real-world entity. This
events rather than objects.

[0] <https://groups.csail.mit.edu>

“combinational creativity” [Boden]

familiar elements combined in new ways

for HackerNews, things like

a post has a title and either just a link, or just a question
no comments on a post after 2 weeks, no edits after 2 hours
can't downvote a comment until your own post upvoted

...

for a Michael Jackson festschrift

critically. HTDP addresses this class,

clashes—and just recognizing

one structure. It's why real
method.

JSD) with a long-running process
storming for DDD) that began with

▲ ob-nix 63 days ago [-]

... this brings back memories! In the late eighties I, as a teenager, found a Jackson Struct. Pr. book at the town library. I remember I was amazed at the text and wondered why I hadn't heard about the method before.

If I remember correctly did the book clearly point out backtracking as a standard method, while mentioning that most languages lacked that, so it had to be implemented manually.

defining a concept in detail

concept Upvote

purpose rank items by popularity

principle after series of upvotes of items, the items are ranked by their number of upvotes

state

by: Vote \rightarrow one User

for: Vote \rightarrow one Item

Upvote, Downvote: set Vote

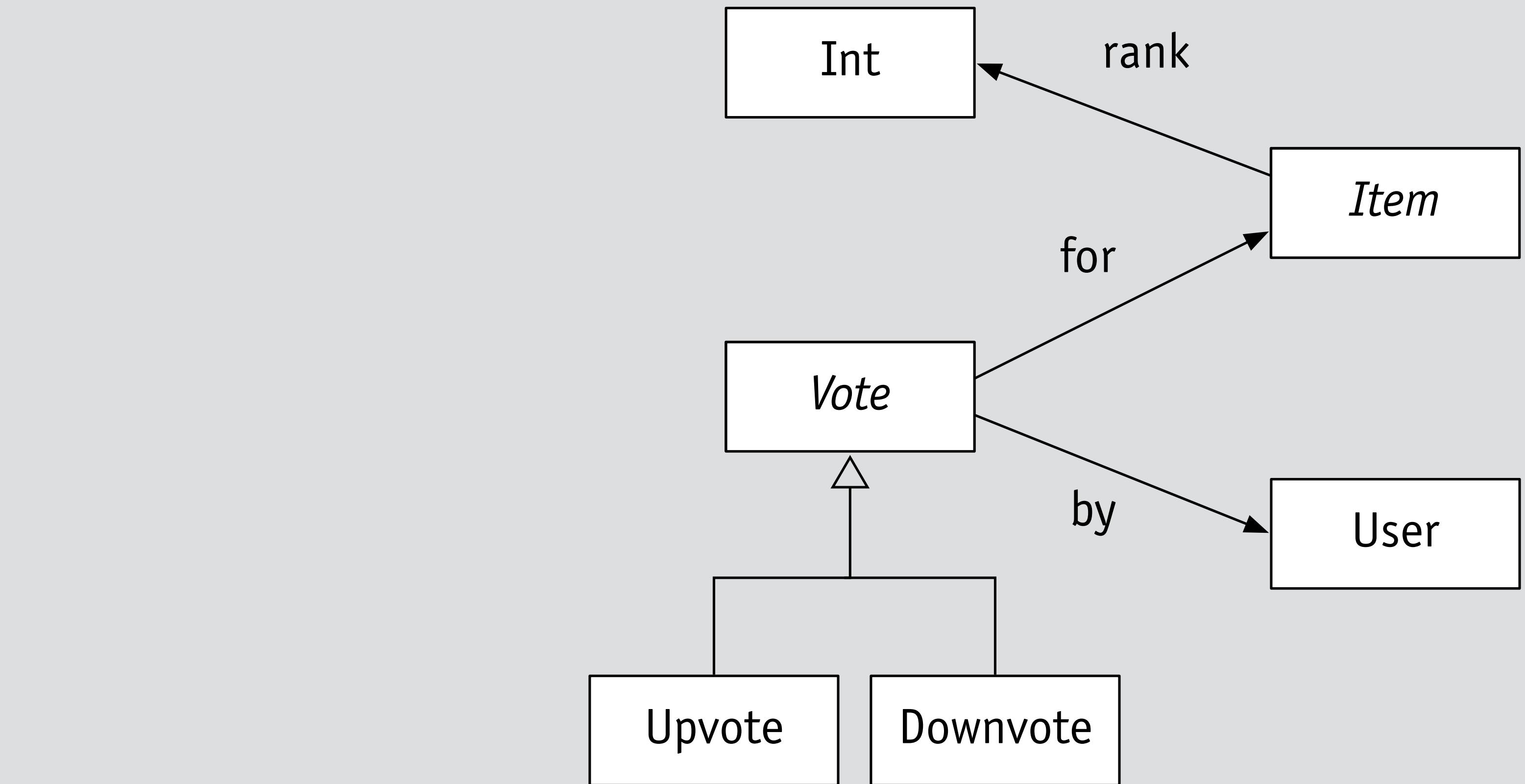
rank: Item \rightarrow one Int

actions

upvote (u: User, i: Item)

downvote (u: User, i: Item)

unvote (u: User, i: Item)



downvote (i: Item, u: User)

no downvote exists for i by u

remove all upvotes for i by u

add a downvote for i by u

update the rank of i

concepts as carriers of design knowledge

concept: Upvote

related concepts

Rating, Recommendation, Reaction, ...

design variants

downvote as unvote
use age in ranking
weigh downvotes more
various identity tactics
freezing old posts

typical uses

social media posts
comments on articles
Q&A responses



known issues

high votes can promote old content
feedback favors early upvotes
upvoting encourages echo chamber
preventing double votes

often used with

Karma, Auth, ...

concepts are about semantics, not user interface

concept Upvote

purpose rank items by popularity

principle after series of upvotes of items, the items are ranked by their number of upvotes



This is homework and I'm having a
are the definitions of the objects:

8



```
sig Library {  
    patrons : set Person,  
    on_shelves : set Book,  
}
```

1

concept Reaction

purpose support quick responses

principle when user selects reaction, it's shown to the author (often in aggregated form)

Today ▾

Daniel I think we should organize a software concepts forum.

1

concept Recommendation

purpose infer user preferences

principle user likes lead to ranking of kinds of items, thus which items are recommended



your turn: which concepts do behaviors belong to?

Upvote **Favorite** **Session**

Karma **Comment** **Post**

concepts of Hacker News

reward points only go up

you can't like a post twice

you need to login every two hours

posts contain just URLs

you can comment on a post or a comment

favorite lists are unbounded

some behaviors involve more than one

you can't downvote a post until your own post is upvoted

you can't edit a post unless you're logged in as author

some behaviors belong to one concept

behaviors that involve >1 concept

concept Upvote

purpose rank items by popularity

actions

upvote (u: User, i: Item)
downvote (u: User, i: Item)
unvote (u: User, i: Item)

suppose I want this behavior:

you can't downvote an item
until you've received
an upvote on your own post

define a new concept!

a hint: not just used by Upvote

concept Karma

purpose privilege good users

state

karma: User -> one Int

actions

reward (u: User, r: Int)
permit (u: User, r: Int)

concept Post

purpose share content

state

author: Post -> one User
body: Post -> one Text

actions

create (u: User, t: Text): Post
delete (p: Post)
edit (p: Post, t: Text)
get_author (p: Post): User

composition by synchronization

concept Upvote

actions

upvote (u: User, i: Item)
downvote (u: User, i: Item)
unvote (u: User, i: Item)

when

Web.request (upvote, u, i)
Post.get_author (i) = u'
Karma.reward (u', 10)

then

Upvote.upvote (u, i)

concept Karma

actions

reward (u: User, r: Int)
permit (u: User, r: Int)

when

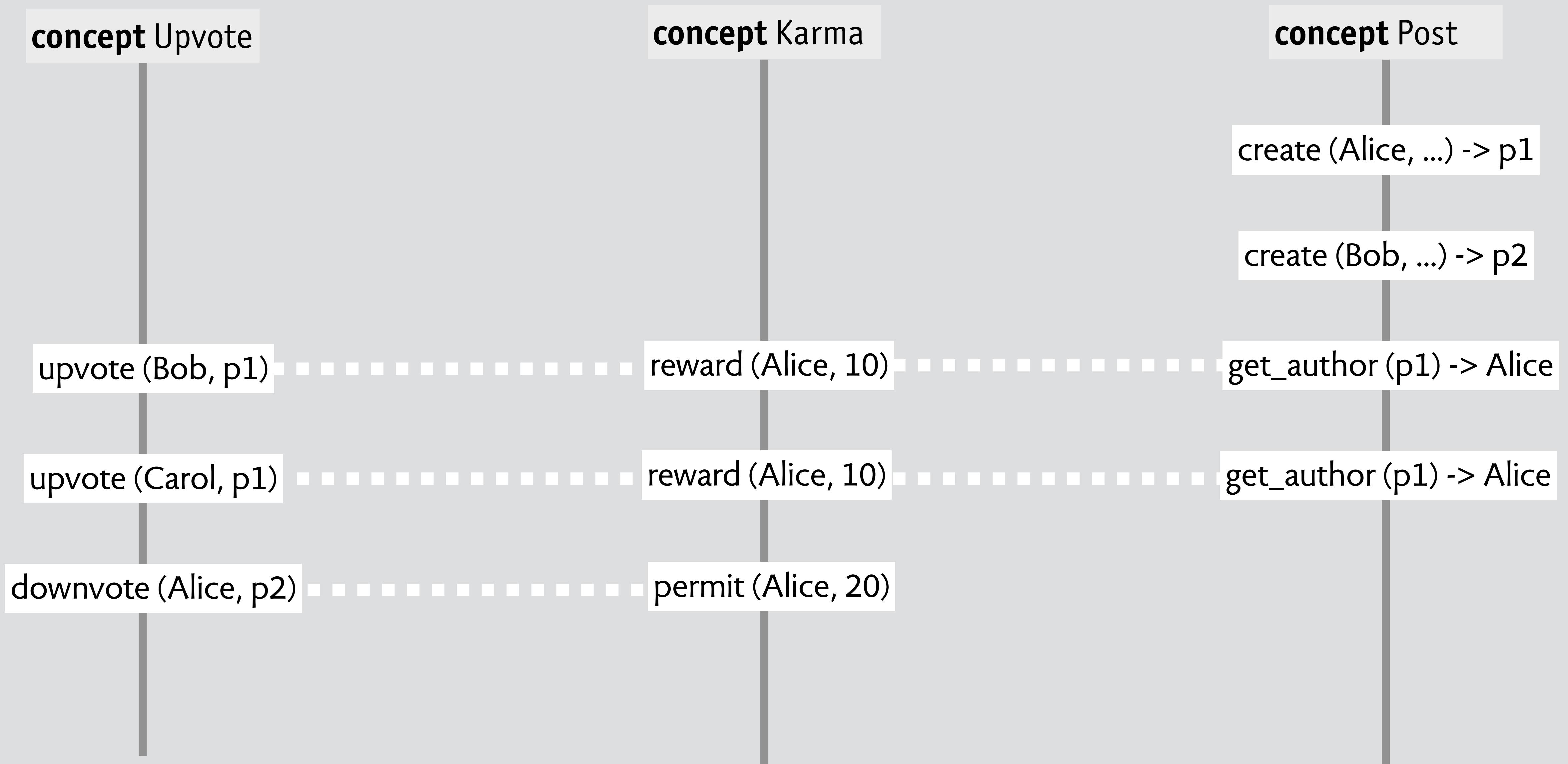
Web.request (downvote, u, i)
Karma.permit (u, 20)
then Upvote.downvote (u, i)

concept Post

actions

create (u: User, t: Text): Post
delete (p: Post)
edit (p: Post, t: Text)
get_author (p: Post): User

synchronizations at runtime



key properties

concept Upvote

upvote (Bob, p1)

upvote (Carol, p1)

downvote (Alice, p2)

app-independent

app-specific details
mostly in the syncs

concept Karma

reward (Alice, 10)

reward (Alice, 10)

permit (Alice, 20)

not a call:
decoupled

concept Post

create (Alice, ...) -> p1

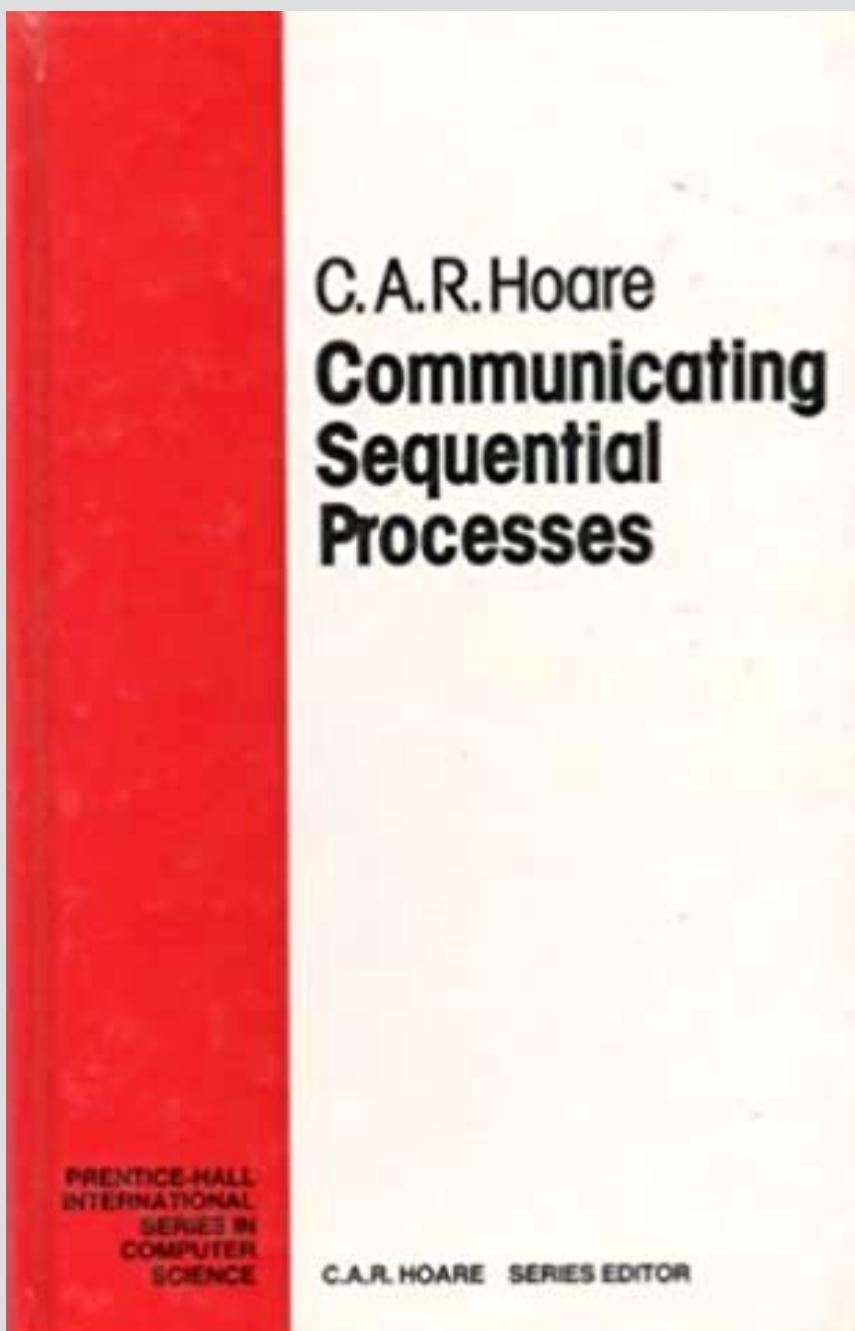
create (Bob, ...) -> p2

get_author (p1) -> Alice

get_author (p1) -> Alice

trace meets
concept spec

not a new idea



shared events
Hoare's CSP (1978)
occam (1983-1994)

Mediators:

Easing the Design and Evolution of Integrated Systems

Kevin J. Sullivan

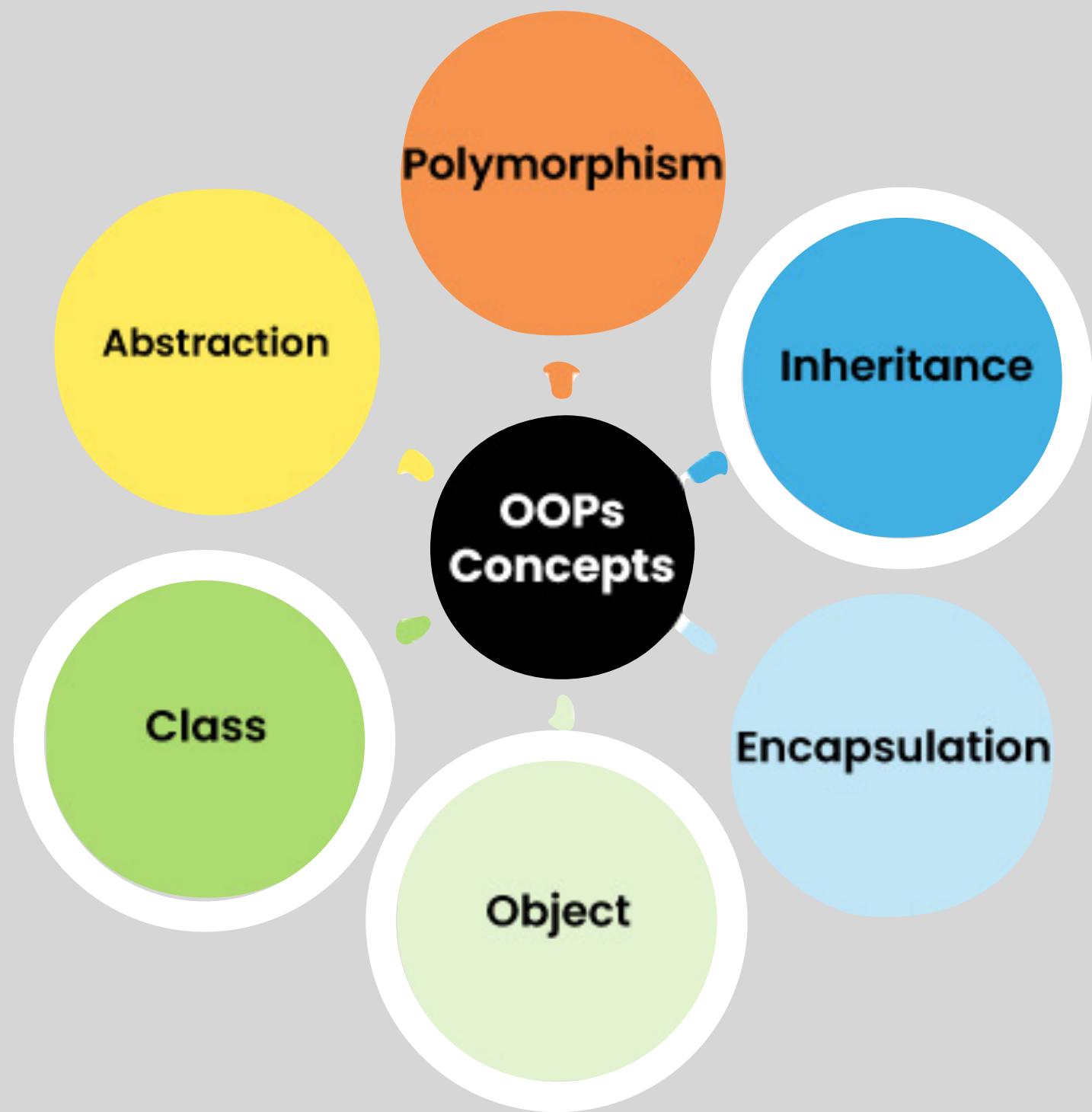
Technical Report 94-08-01

Department of Computer Science and Engineering
University of Washington

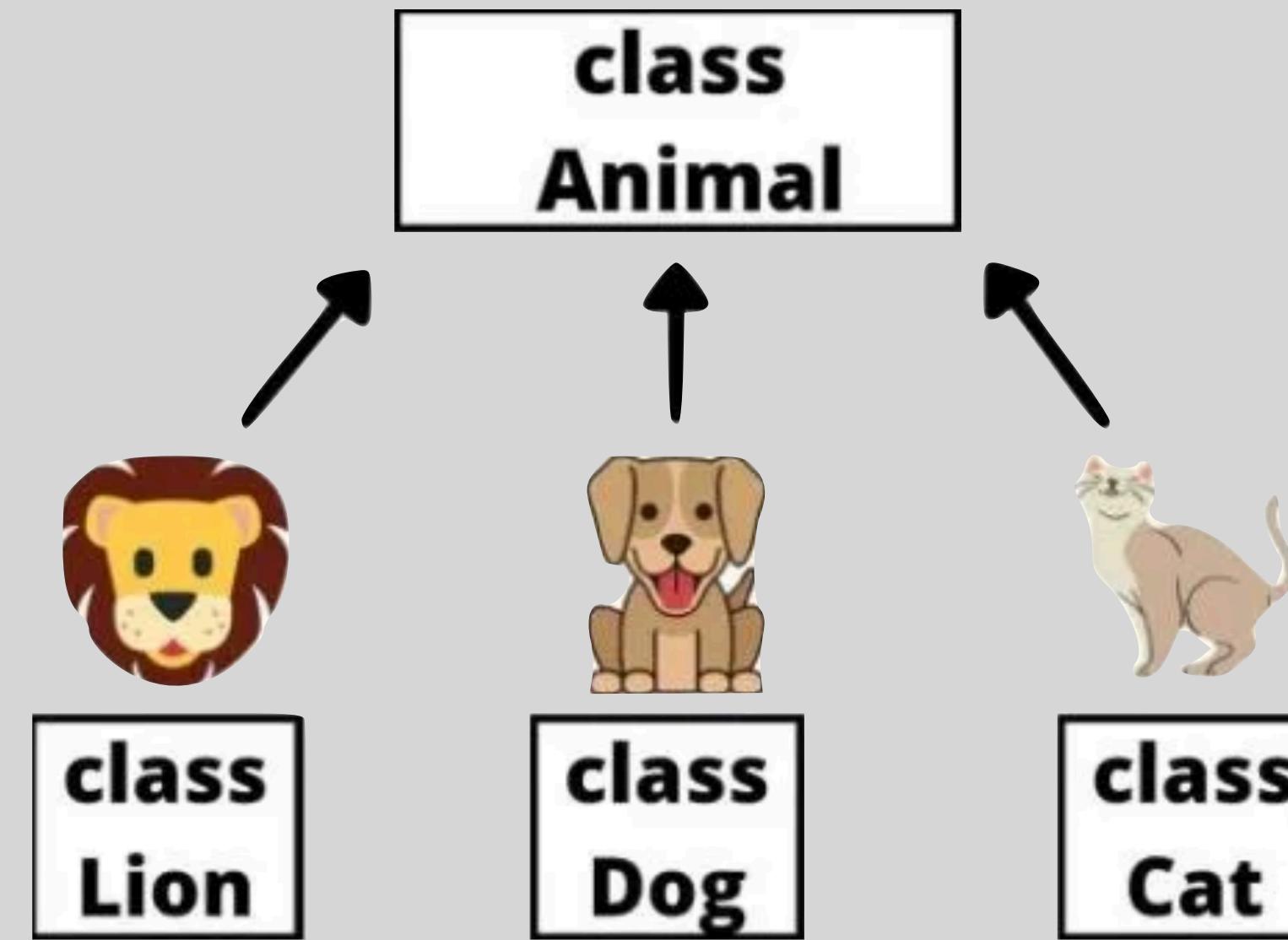
mediator pattern
Kevin Sullivan (1994)
Gang of Four (1994)

concepts vs. objects
(a coding detour)

concepts are not classes



object-oriented programming
a dominant coding paradigm
good for implementing concepts



object-oriented development
objects correspond to real world entities
embody *all* of their behavior

let's build HN with objects

```
class User {  
    String name;  
    String password;  
    User register (n, p) { ... }  
    User authenticate (n, p) { ... }  
}
```

```
class Post {  
    User author;  
    String body;  
    Post new (a, b) { ... }  
}
```

adding upvoting

```
class User {  
    String name;  
    String password;  
    User register (n, p) { ... }  
    User authenticate (n, p) { ... }  
}
```

```
class Post {  
    User author;  
    String body;  
Set [User] ups, downs;  
    Post new (a, b) { ... }  
upvote (u) { ... }  
downvote (u) { ... }  
}
```

adding karma

```
class User {  
    String name;  
    String password;  
int karma;  
    User register (n, p) { ... }  
    User authenticate (n, p) { ... }  
incKarma (i) { ... }  
bool hasKarma (i) { ... }  
}
```

```
class Post {  
    User author;  
    String body;  
    Set [User] ups, downs;  
    Post new (a, b) { ... }  
    upvote (u) { ... }  
downvote (u) {  
        if u.hasKarma (10) ... }  
    }
```

adding commenting

```
class User {  
    String name;  
    String password;  
    int karma;  
    User register (n, p) { ... }  
    User authenticate (n, p) { ... }  
    incKarma (i) { ... }  
    bool hasKarma (i) { ... }  
}
```

```
class Post {  
    User author;  
    String body;  
    Set [User] ups, downs;  
Seq [Post] comments;  
    Post new (a, b) { ... }  
    upvote (u) { ... }  
    downvote (u) {  
        if u.hasKarma (10) ... }  
addComment (c) { ... }  
}
```

what's wrong with this code?

```
class User {  
    String name;  
    String password;  
    int karma;  
  
    User register (n, p) { ... }  
    User authenticate (n, p) { ... }  
    incKarma (i) { ... }  
    bool hasKarma (i) { ... }  
}
```

```
class Post {  
    User author;  
    String body;  
    Set [User] ups, downs;  
    Seq [Post] comments;  
    Post new (a, b) { ... }  
    upvote (u) { ... }  
    downvote (u) {  
        if u.hasKarma (10) ... }  
    addComment (c) { ... }  
}
```

User authentication

Posting

Upvoting

Commenting

Karma

no separation of concerns

Post class contains posting,
commenting, upvoting, karma

dependencies between files

Post class calls User class
to get karma points

classes are novel & not reusable

Post class won't work in an app
that doesn't have karma points

can't be built independently

to build Post class, need User class
to have been built already

a different way

```
concept User {  
    Map [User, String] name;  
    Map [User, String] password;  
    User register (n, p) { ... }  
    User authenticate (n, p) { ... }  
}
```

```
concept Karma [U] {  
    Map [U, Int] karma;  
    incKarma (u, i) { ... }  
    hasKarma (u, i) { ... }  
}
```

concerns
now cleanly
separated

coupling is
gone: refs are
polymorphic

```
concept Post [U] {  
    Map [Post, U] author;  
    Map [Post, URL] url;  
    Post new (a, u) { ... }  
}
```

```
concept Upvote [U, I] {  
    Map [U, I] ups, downs;  
    upvote (u, i) { ... }  
    downvote (u, i) { ... }  
}
```

```
concept Comment [U, T] {  
    Map [Comment, U] author;  
    Map [Comment, T] target;  
    Map [Comment, String] body;  
    Comment new (a, t, b) { ... }  
}
```

web apps have this!
called a route

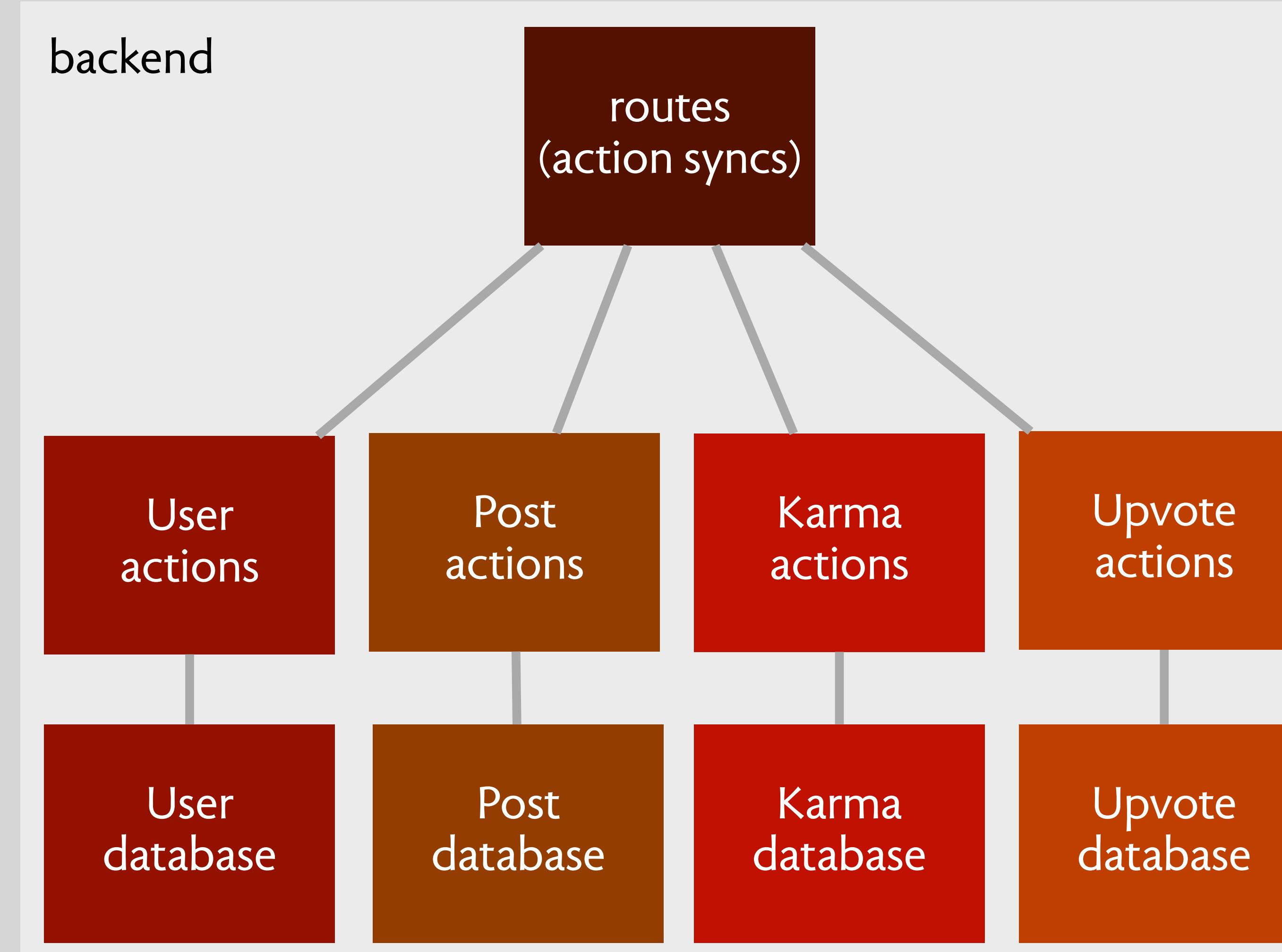
need a mediator
outside the concept

```
sync downvote (u, i) {  
    Karma.hasKarma (u, 10)  
    Upvote.downvote (u, i)  
}
```

web apps have this!
called a database

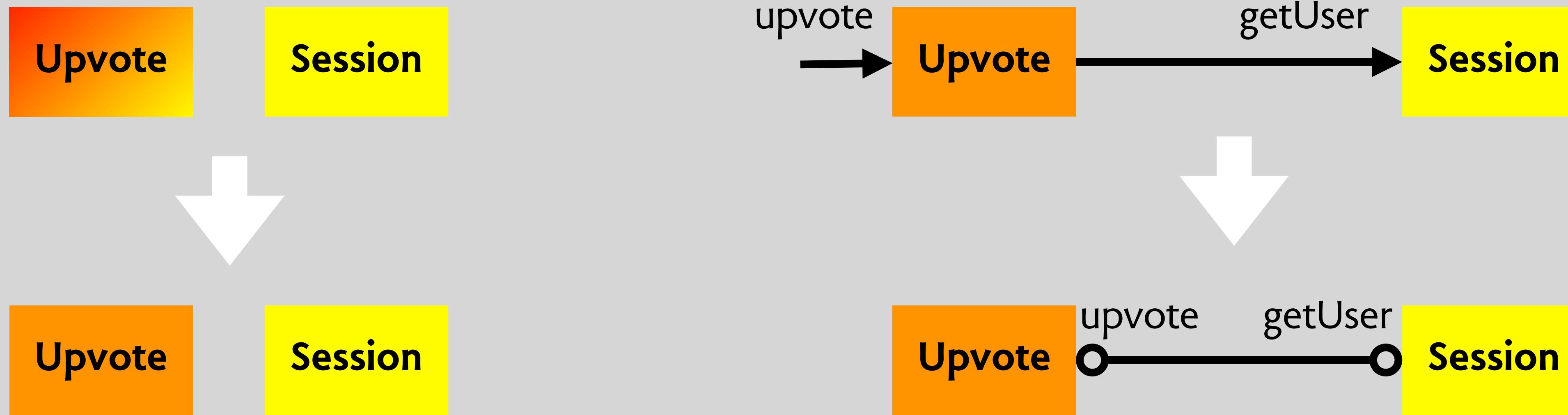
need two-way map
target to comment

a new way to structure a web app



no
dependencies
between
concepts!

key lessons: coherence & independence



coherence

concept embodies a unit of reusable function
concept contains all and only that function

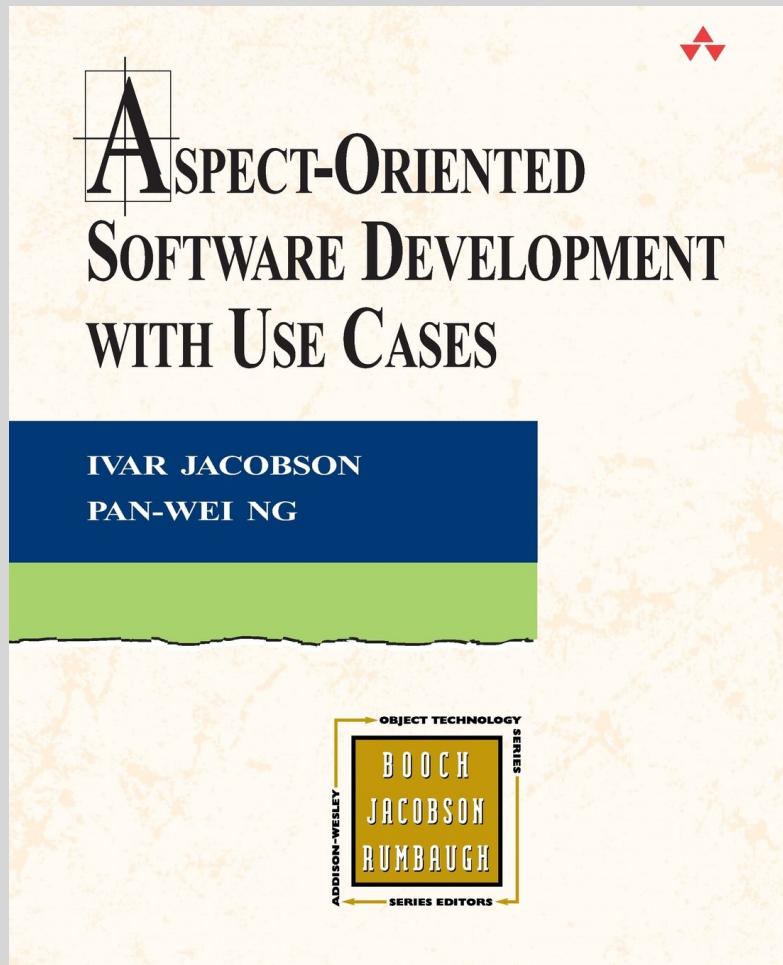
independence

each concept can be understood by itself
concepts don't refer to each other

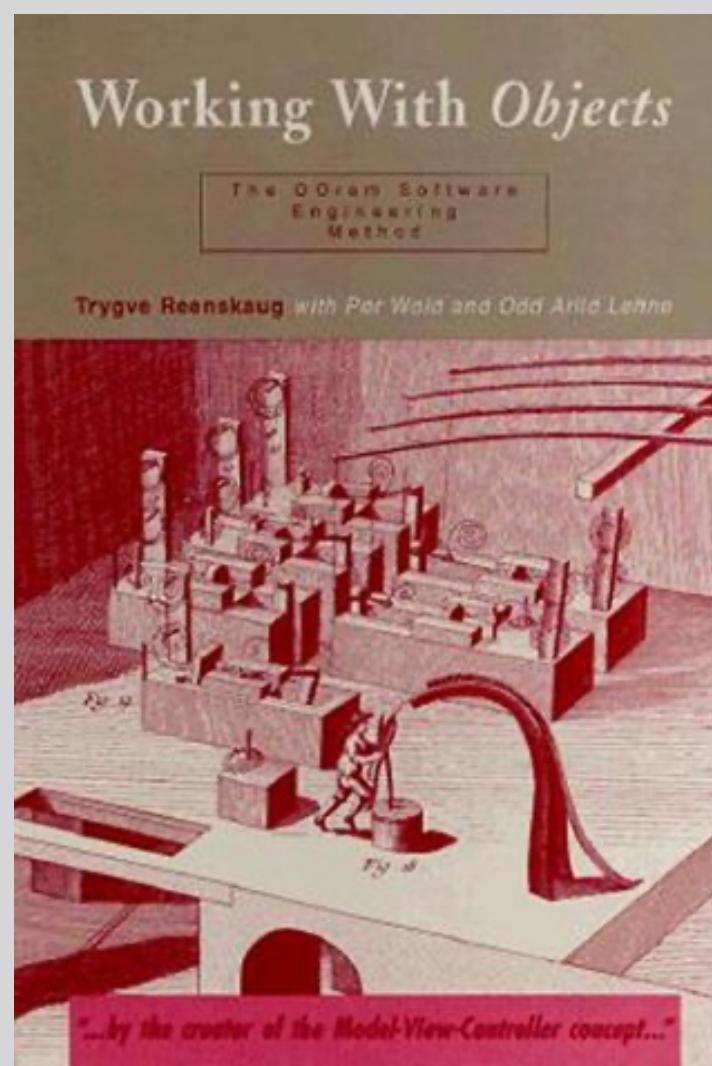
example: preventing double voting
Upvote stops 2x voting by linking voter to vote
Session ensures voter is the logged in user

example: preventing double voting
Upvote doesn't call Session to get user
synchronization ensures voter param is user

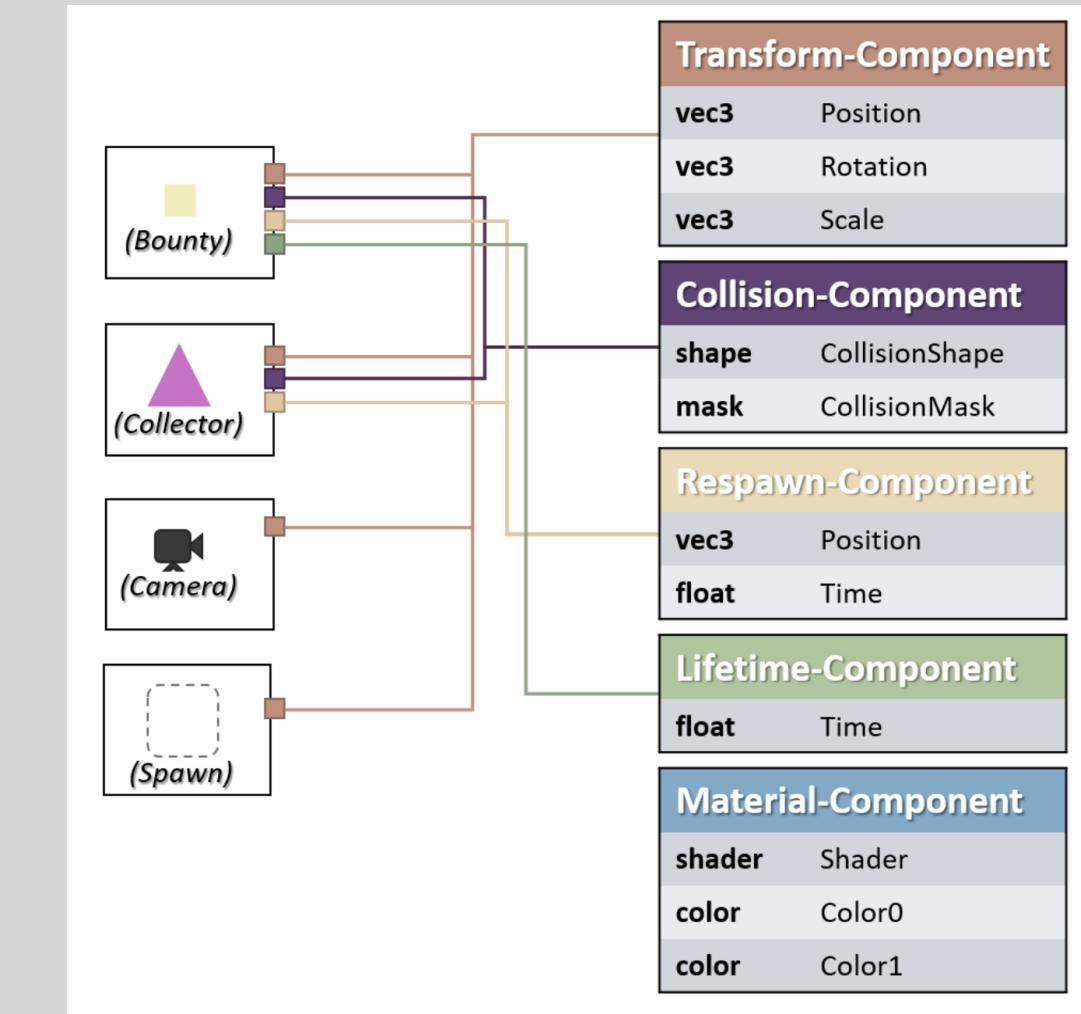
long history of attempts to fix OOP



Aspect-oriented Programming
Kiczales et al (1997)



Role-oriented Programming
Reenskaug et al (1983)



Entity-component system
Scott Bilas et al (2002)

your turn:
a Twitter/X puzzle



Andy Ostroy

@AndyOstroy



Seems the only #Wall @realDonaldTrump's built is the one between him and @FLOTUS #Melania #trump



♡ 8,221 8:15 PM - May 2, 2017



↪ 4,022 people are talking about this



MELANIA TRUMP liked your Tweet

Seems the only #Wall @realDonaldTrump's built is the one between him and @FLOTUS #Melania #trump pic.twitter.com/XiNd2jiLUF

your turn: what does heart button do? and what did ML think it did?



Nov 2, 2015: Twitter changes Favorite (Star) to Like (Heart)

We are changing our star icon for favorites to a heart and we'll be calling them likes... **We know that at times the star could be confusing, especially to newcomers.** You might like a lot of things, but not everything can be your favorite. *Twitter press release*

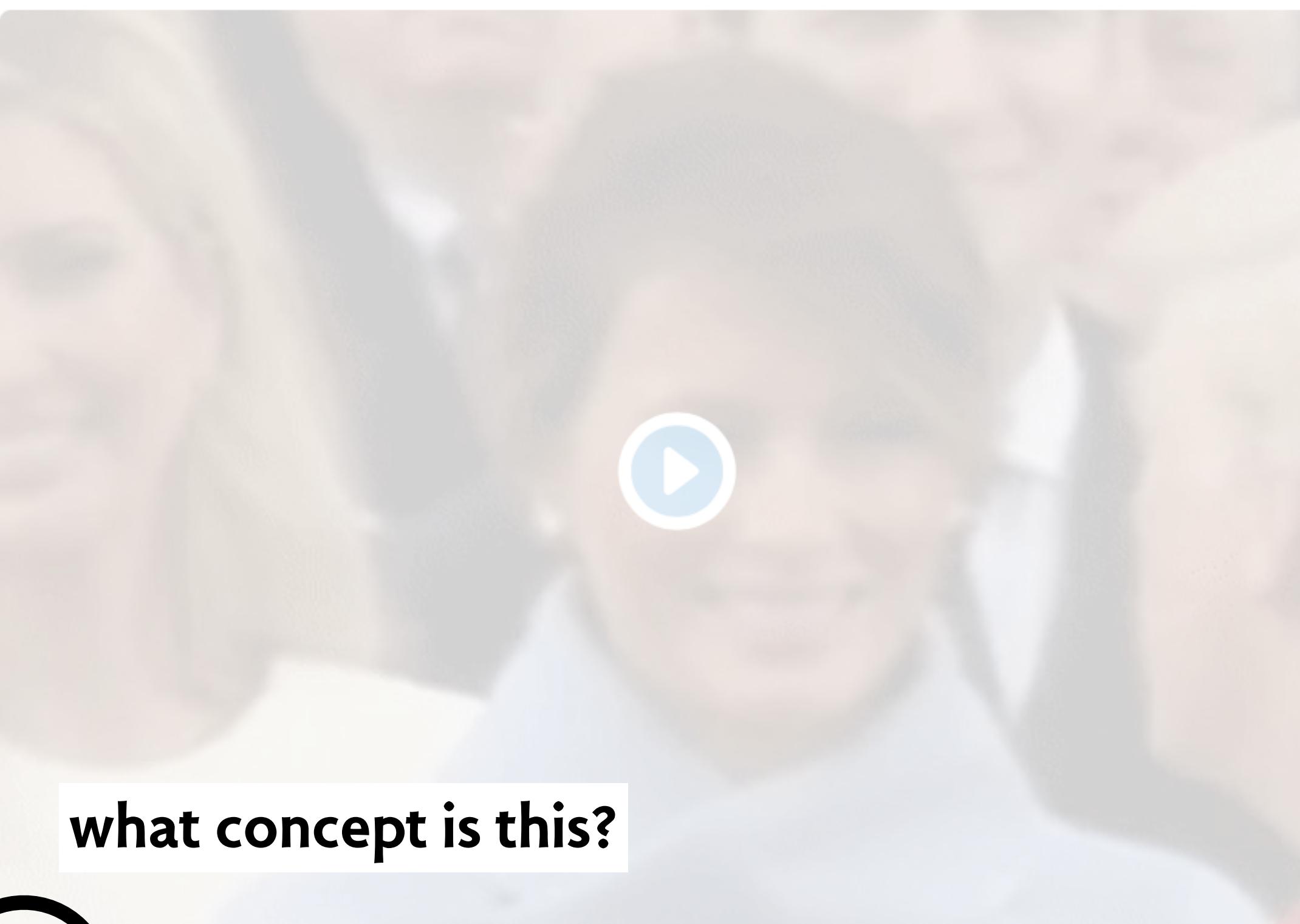


Andy Ostroy

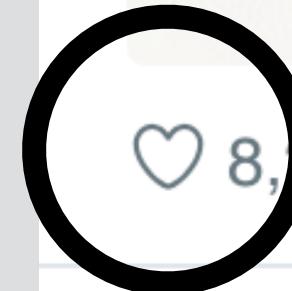
@AndyOstroy



Seems the only #Wall @realDonaldTrump's built is the one between him and @FLOTUS #Melania #trump



what concept is this?



8,221 8:15 PM - May 2, 2017



4,022 people are talking about this



concept Upvote

purpose rank items by popularity

principle after upvotes, ranked by num upvotes

?



concept Bookmark

purpose save items to revisit

principle save then select from private list later



The Boston Globe ✅ @BostonGlobe · 21h

Andrew Yang would fine gunmakers for deaths caused by their products.



Yang would fine gunmakers for deaths caused by their products - Th...

You probably know Andrew Yang wants to give every American \$1,000 a month. Something you might not know: He wants to fine gun ...

🔗 bostonglobe.com

29

9



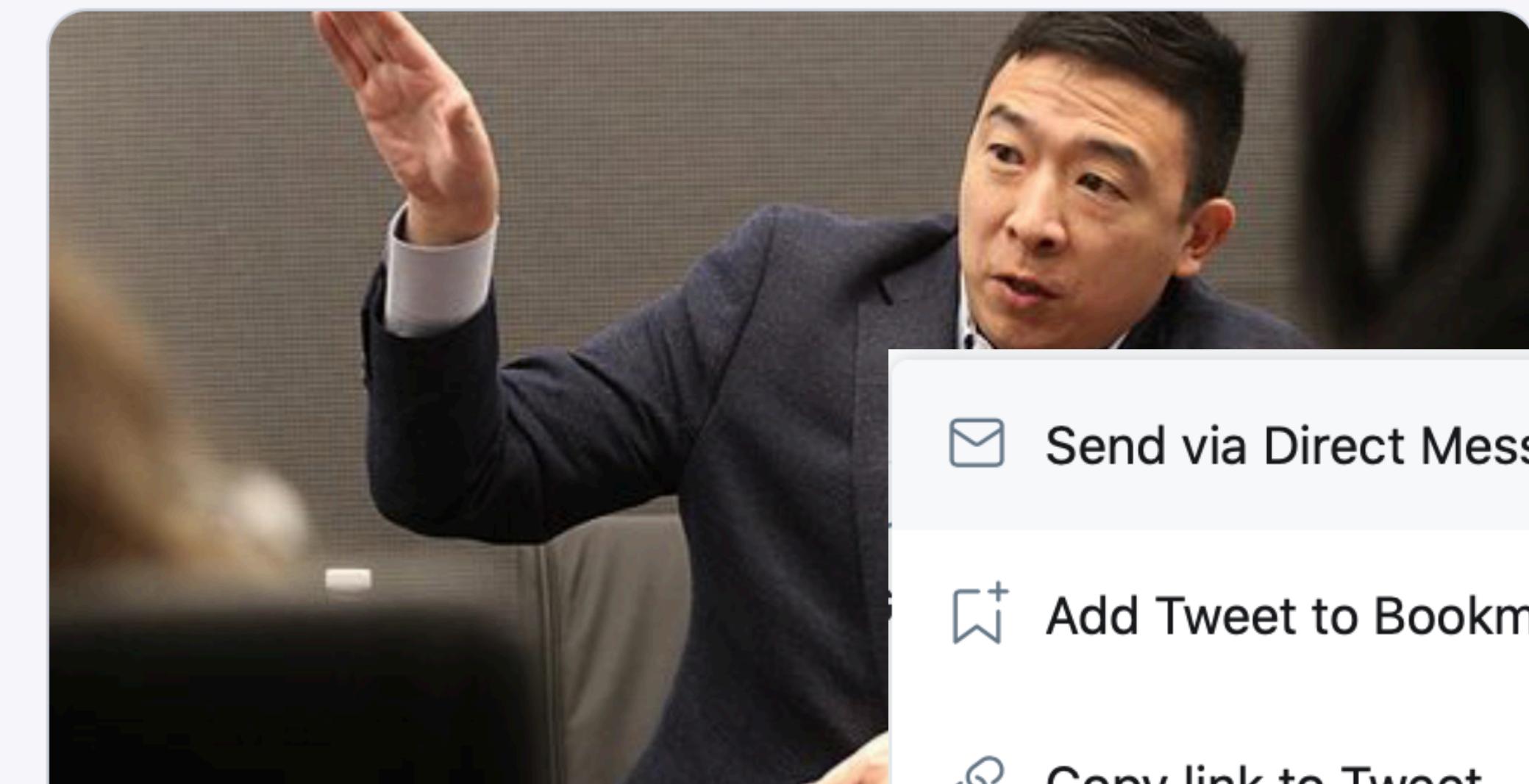
concept Upvote

purpose rank items by popularity



The Boston Globe ✅ @BostonGlobe · 21h

Andrew Yang would fine gunmakers for deaths caused by their products.



Yang would fine gunmakers for deaths ca...

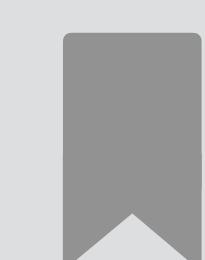
You probably know Andrew Yang wants t... a month. Something you might not know:

🔗 bostonglobe.com

29

9

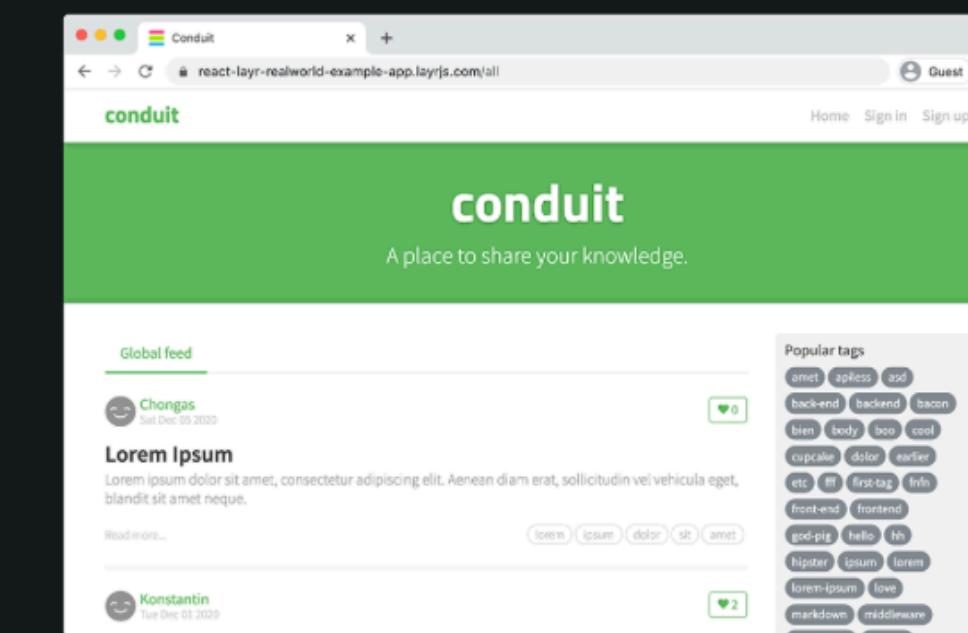
94



concept Bookmark

purpose save items to revisit

your turn:
RealWorld API



The mother of all demo apps

See how the exact same application is built using different libraries and frameworks.

[Frontend](#)[Backend](#)[Fullstack](#)**LANGUAGES**

All

Java

TypeScript

Go

Python

Kotlin

JavaScript

C#

.NET + Minimal API

Erikvdv/realworldapiminimal

C#

ActixWeb + Diesel

snamiki1212/realworld-v1-rust-actix-web-diesel

Rust

Adonis

Utwo/adonis-realworld-example-app

JavaScript

Adonis

TypeScript

how many concepts can you find in this endpoint?

RealWorld

Search K

Implementation creation ▾
Introduction
Features
Expectations

Specifications ▾
Frontend specifications ▾
Templates
Styles
Routing
API
Tests

Backend specifications ▾
Introduction
Endpoints
API response format

CORS
Error handling
Postman
Tests

Mobile specifications

Community ▾
Authors
Resources

Users (for authentication)

```
{  
  "user": {  
    "email": "jake@jake.jake",  
    "token": "jwt.token.here",  
    "username": "jake",  
    "bio": "I work at statefarm",  
    "image": null  
  }  
}
```

Profile

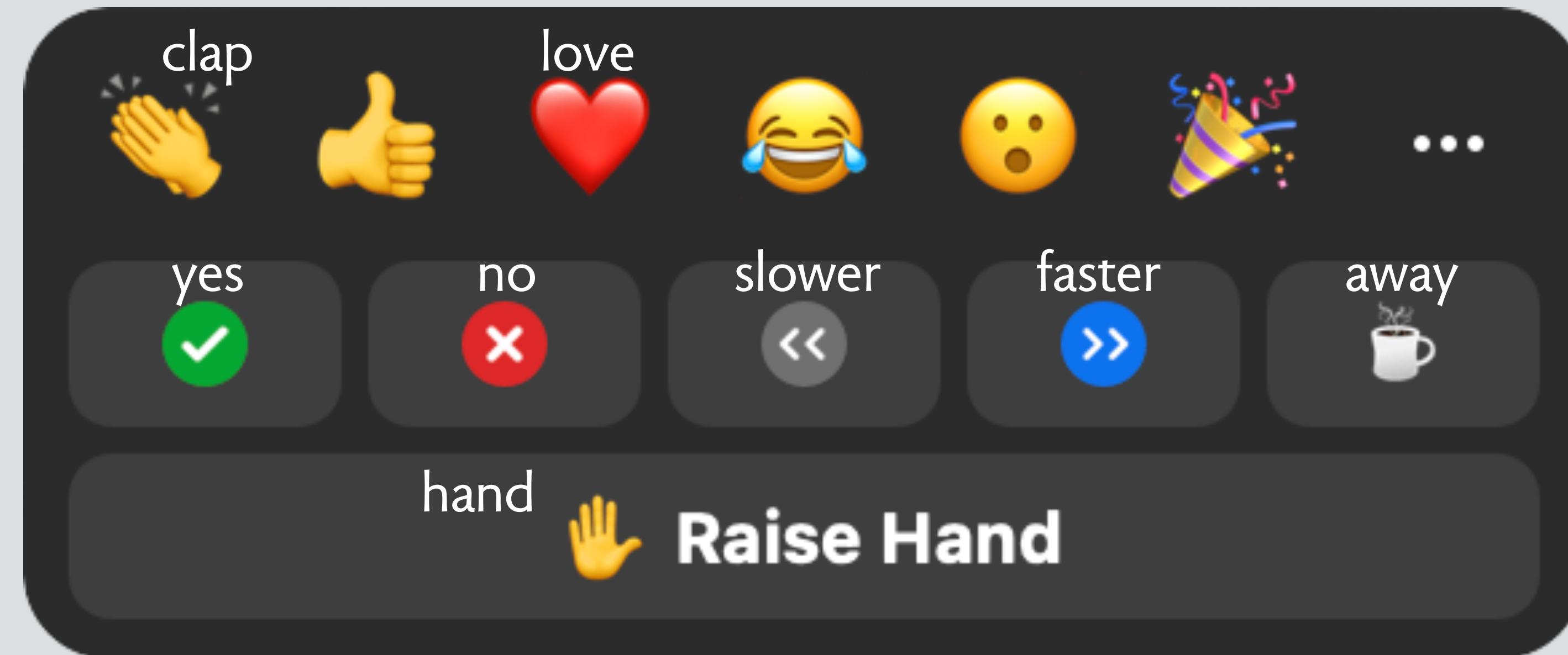
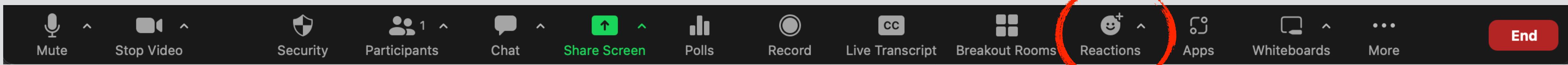
```
{  
  "profile": {  
    "username": "jake",  
    "bio": "I work at statefarm",  
    "image": "https://api.realworld.io/images/smiley-cyrus.jpg",  
    "following": false  
  }  
}
```

On this page

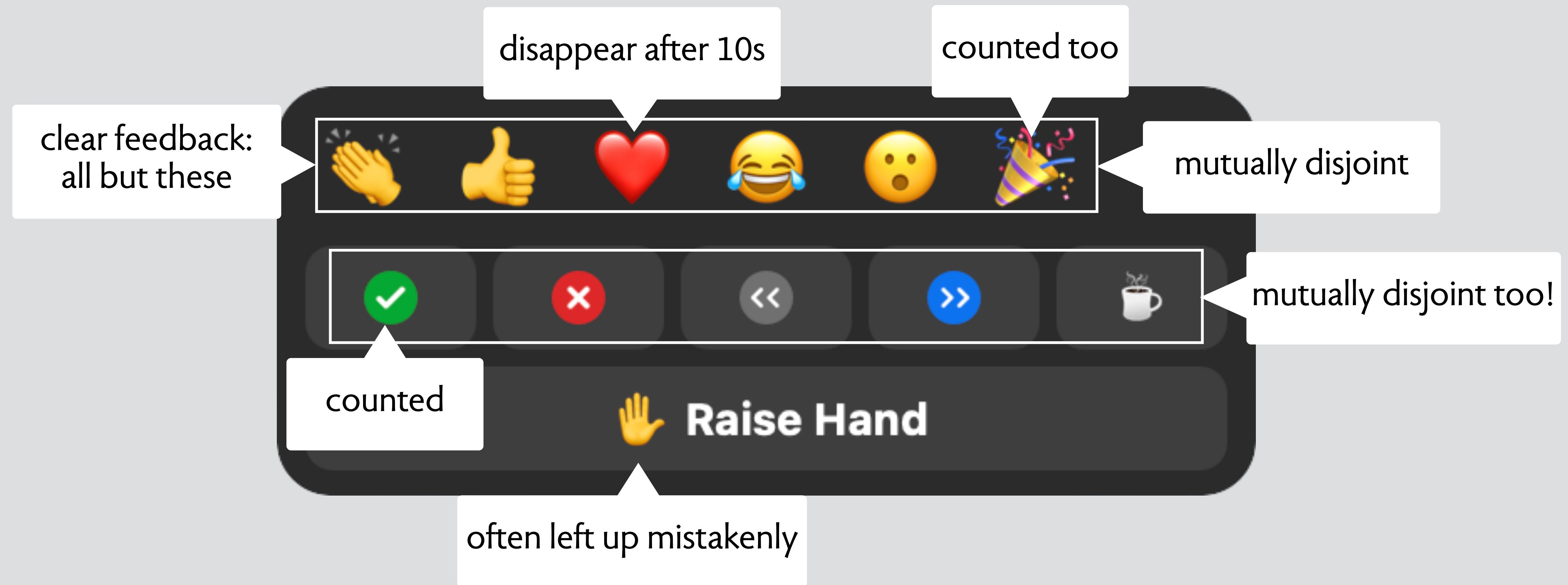
Overview
JSON Objects returned by API:
Users (for authentication)
Profile
Single Article
Multiple Articles
Single Comment
Multiple Comments
List of Tags

exercise:
Zoom reactions

Zoom's reactions



anomalous behaviors



functions by reaction type

Reaction	Disappears	Counted	Cancel by host
Emojis	✓	(✓)	
Yes/no		✓	✓
Slow/speed		✓	✓
Away		(✓)	(✓)
Hand		(✓)	✓



yes



yes, but should probably be no

disjointness of reaction types: my take

Reaction	Emojis	Yes/no	Slow/speed	Away	Hand
Emojis	✓				
Yes/no		✓	(✓)	(✓)	(✓)
Slow/speed		(✓)	✓	(✓)	(✓)
Away		(✓)	(✓)	✓	(✓)
Hand		(✓)	(✓)	(✓)	✓

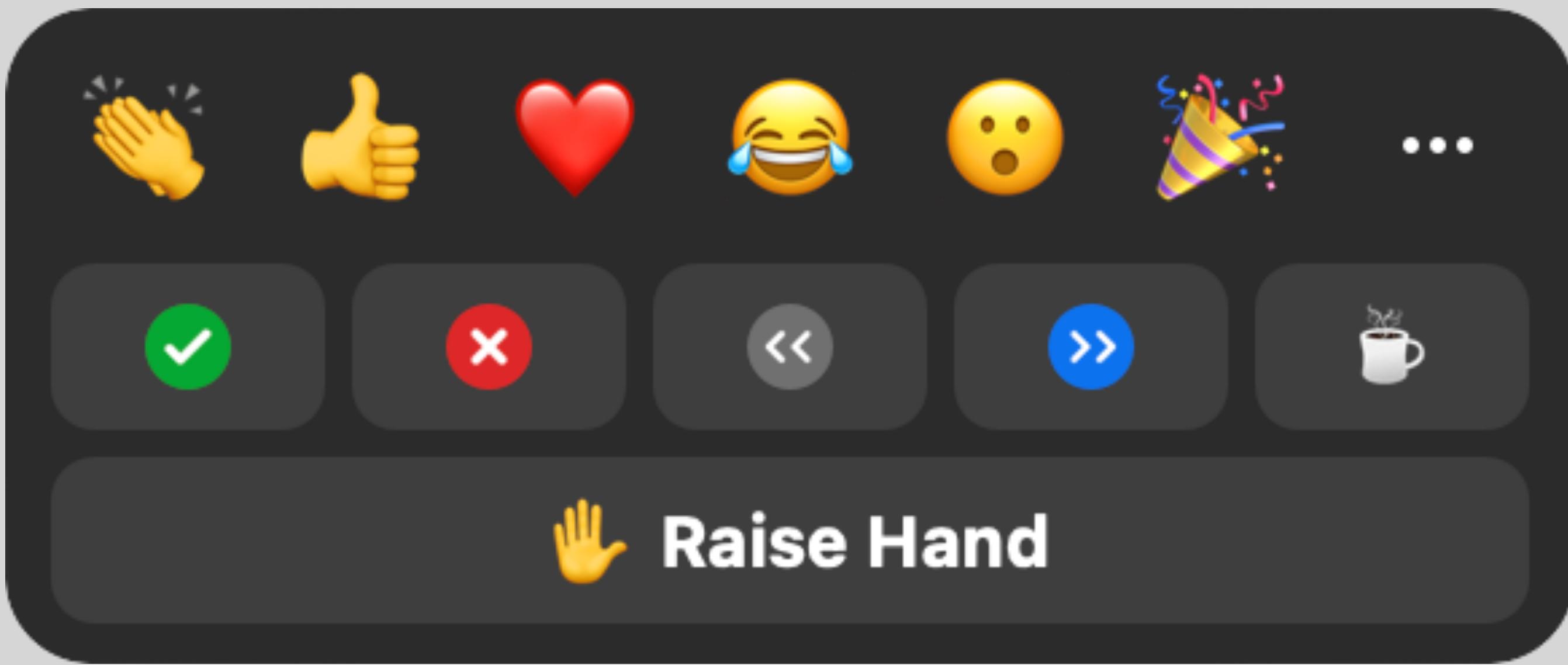


yes



yes, but should probably be no

redesigning Zoom



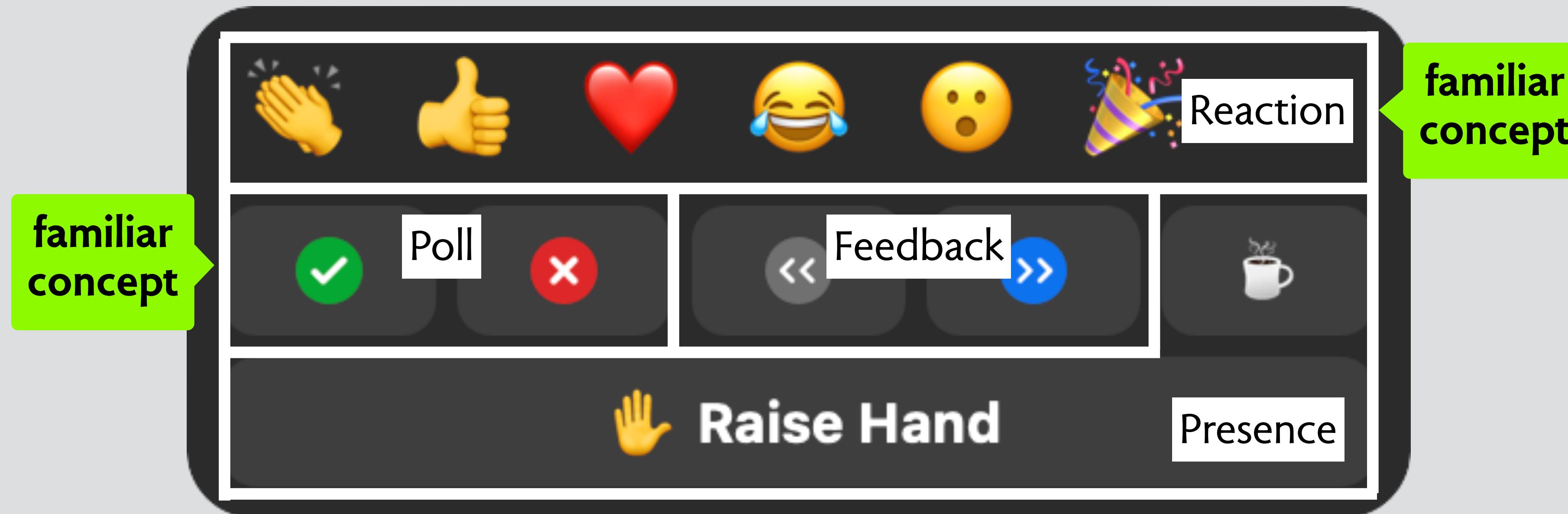
what concepts are involved?

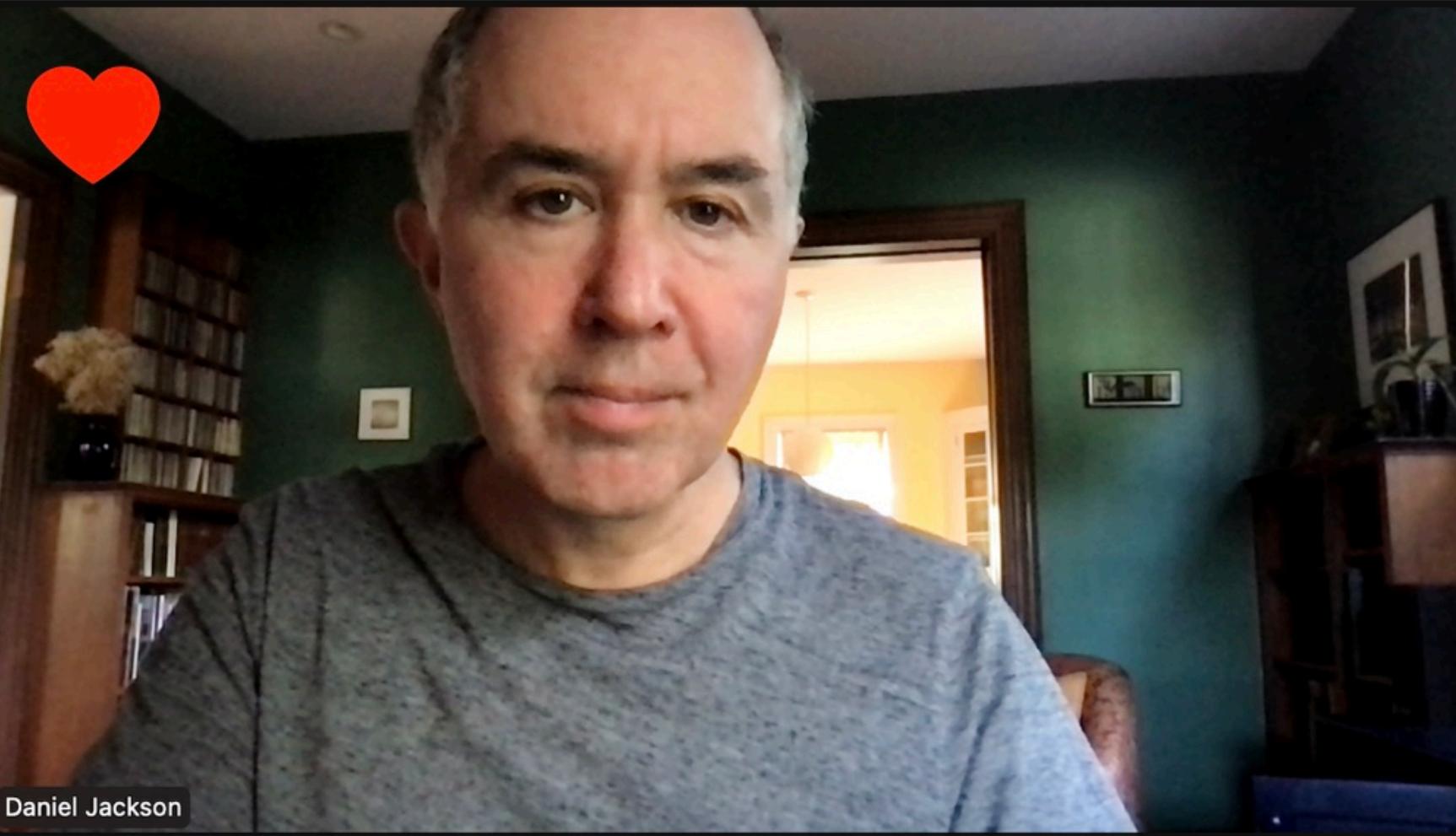
can you identify the concepts behind this widget?
separate them more cleanly?
classify into familiar and novel concepts?

can you do better?

how might you change Zoom's design?
could change UI and behavior for muting, eg

my take: splitting into coherent & independent concepts





Presence

Chat

Reaction

Feedback

Request to speak Watching/listening

Speaking

I'm away

Audience

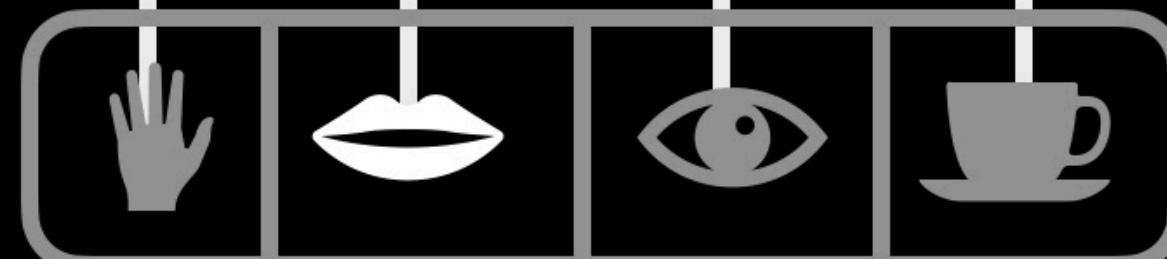
Other emoji

Recent emoji

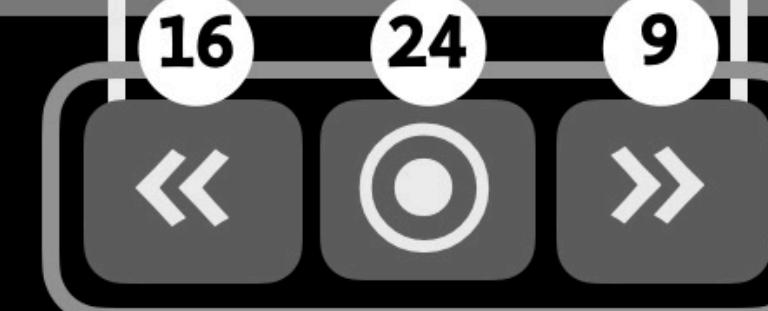
Just right

Slow down

Speed up



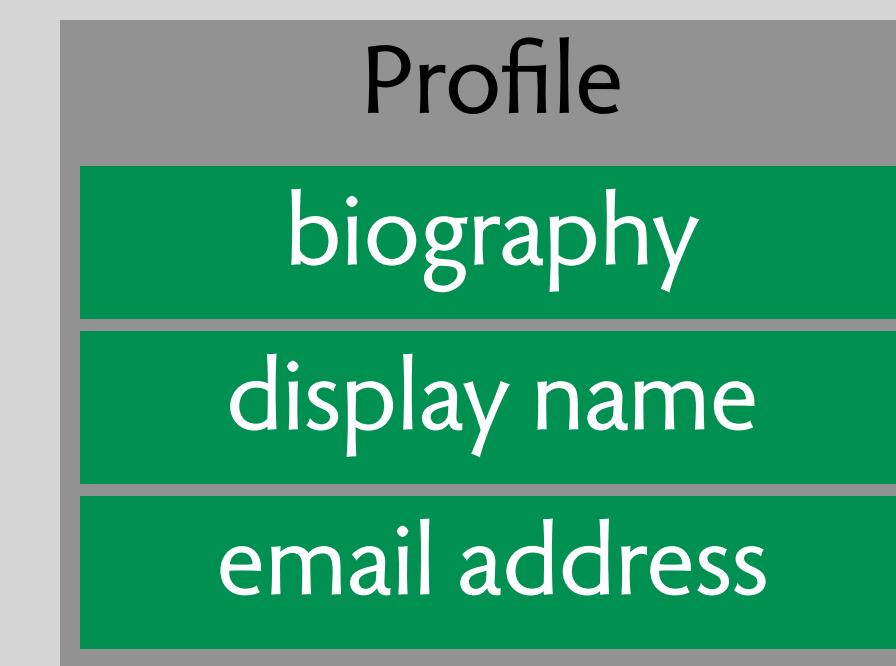
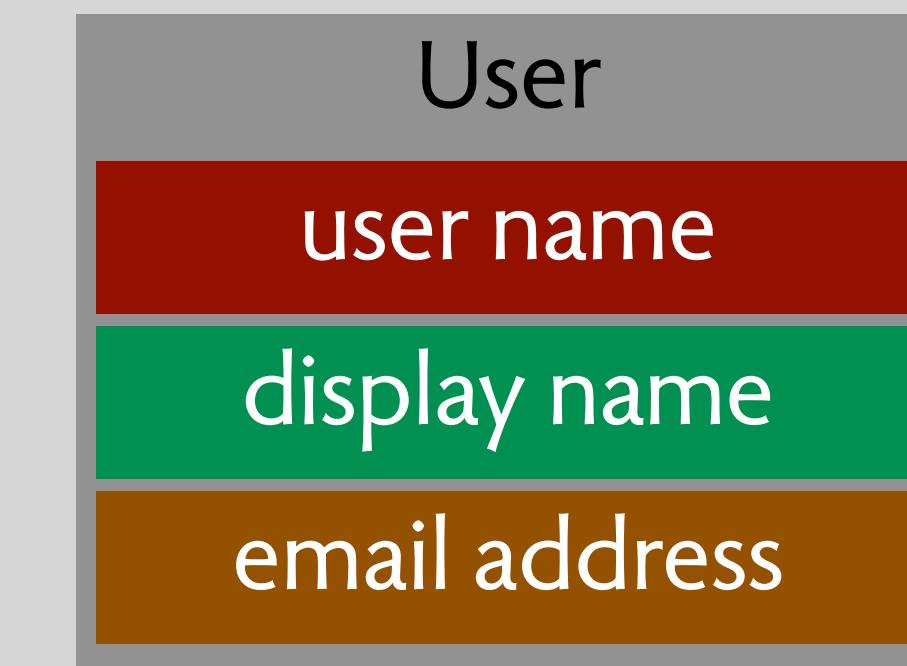
Everyone ▾ Type here...



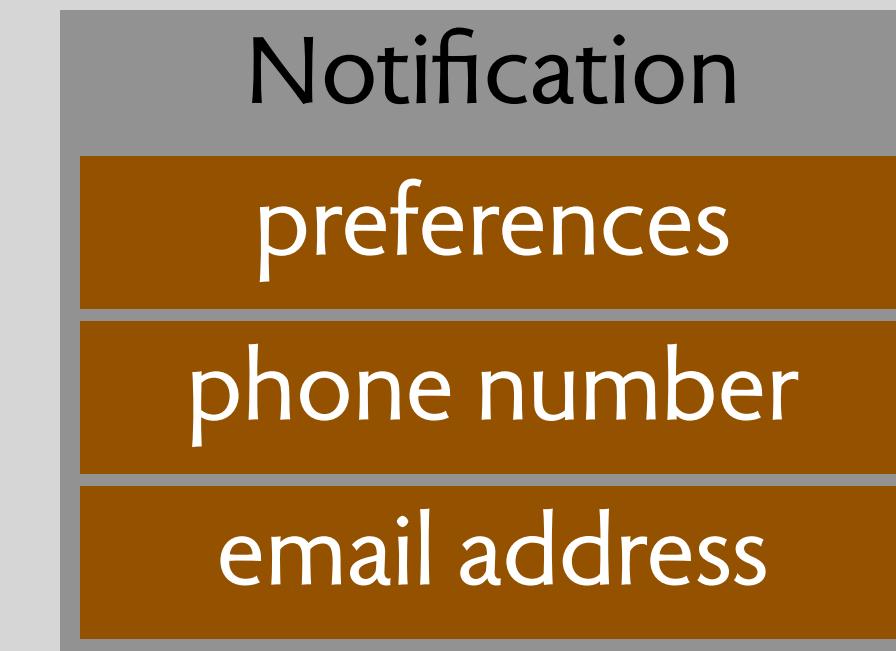
takeaways

key idea #1: coherence

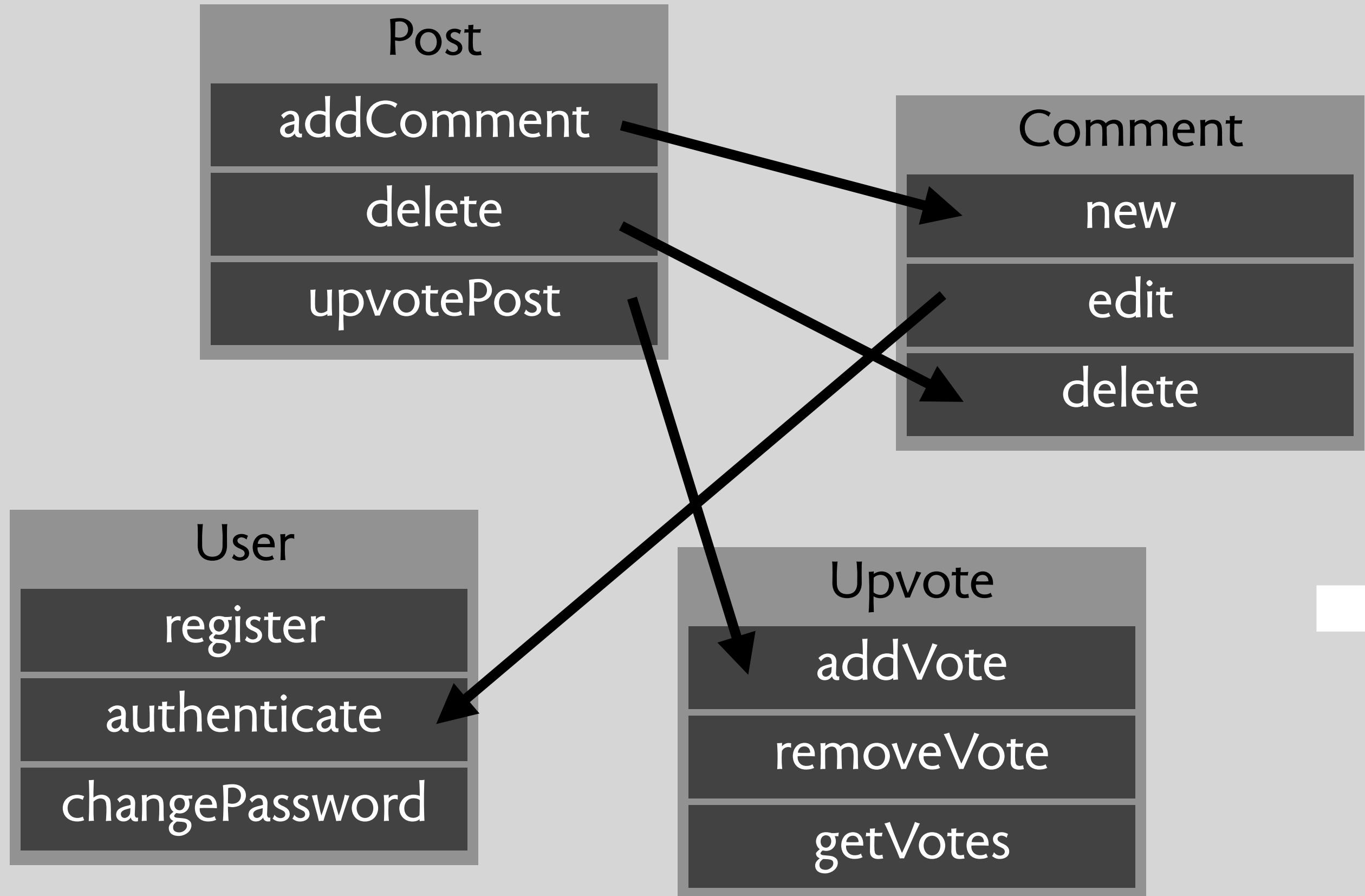
objects tend to
conflate functions



concepts separate
functions



key idea #2: independence



in OOP, method calls couple classes together so they can't stand alone

in concept design, syncs are defined externally so concepts can stand alone

what's next?

what next?

designing concepts
getting to the details
defining behavior: states & actions
purposes & principles