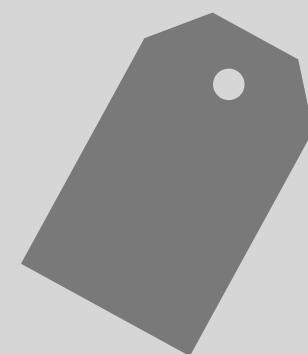


# disentangling concepts

Daniel Jackson · Autodesk · Woodinville, WA · Dec 3-5, 2024

revisiting states  
& actions

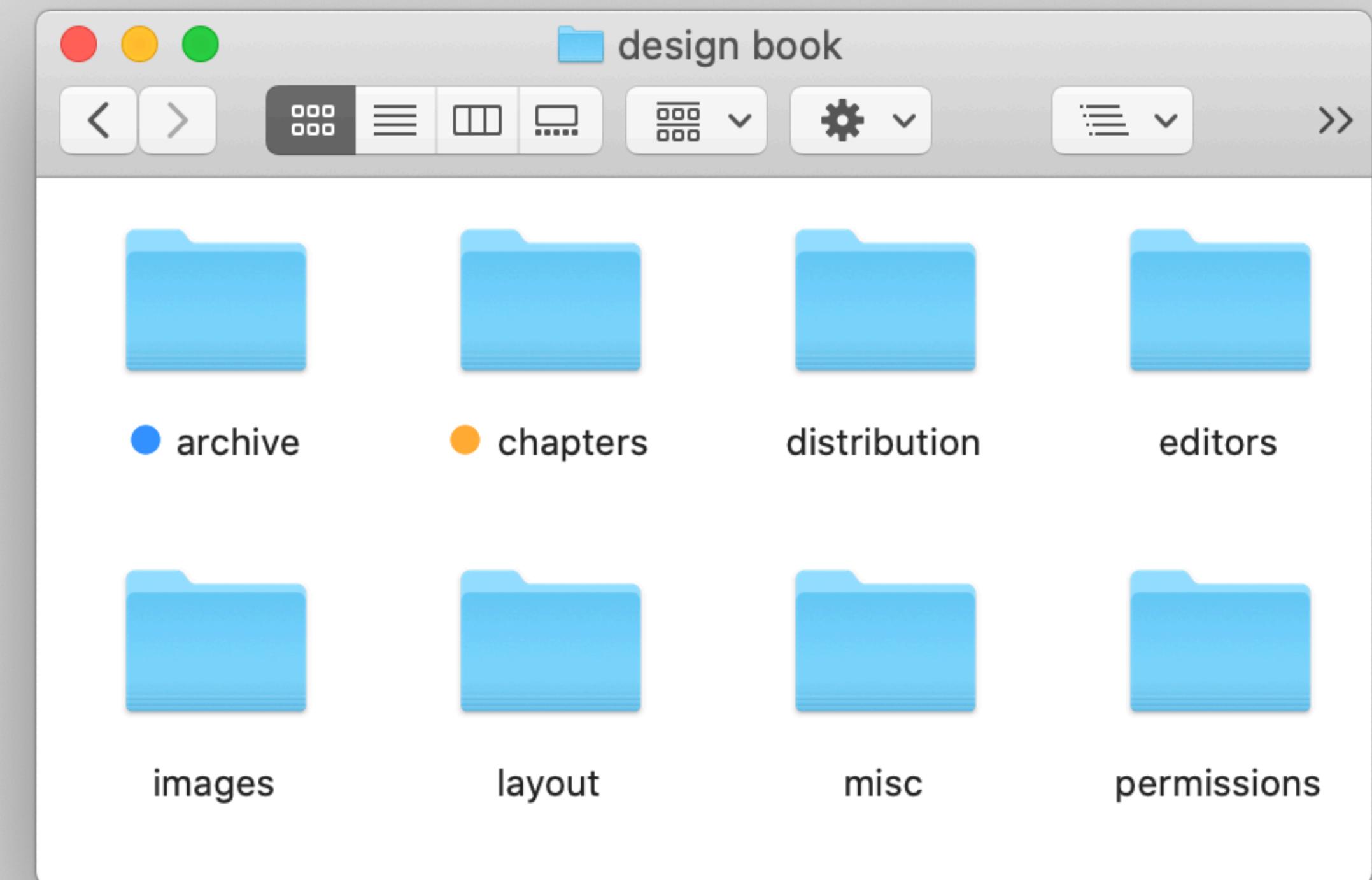
# a simple but potent concept



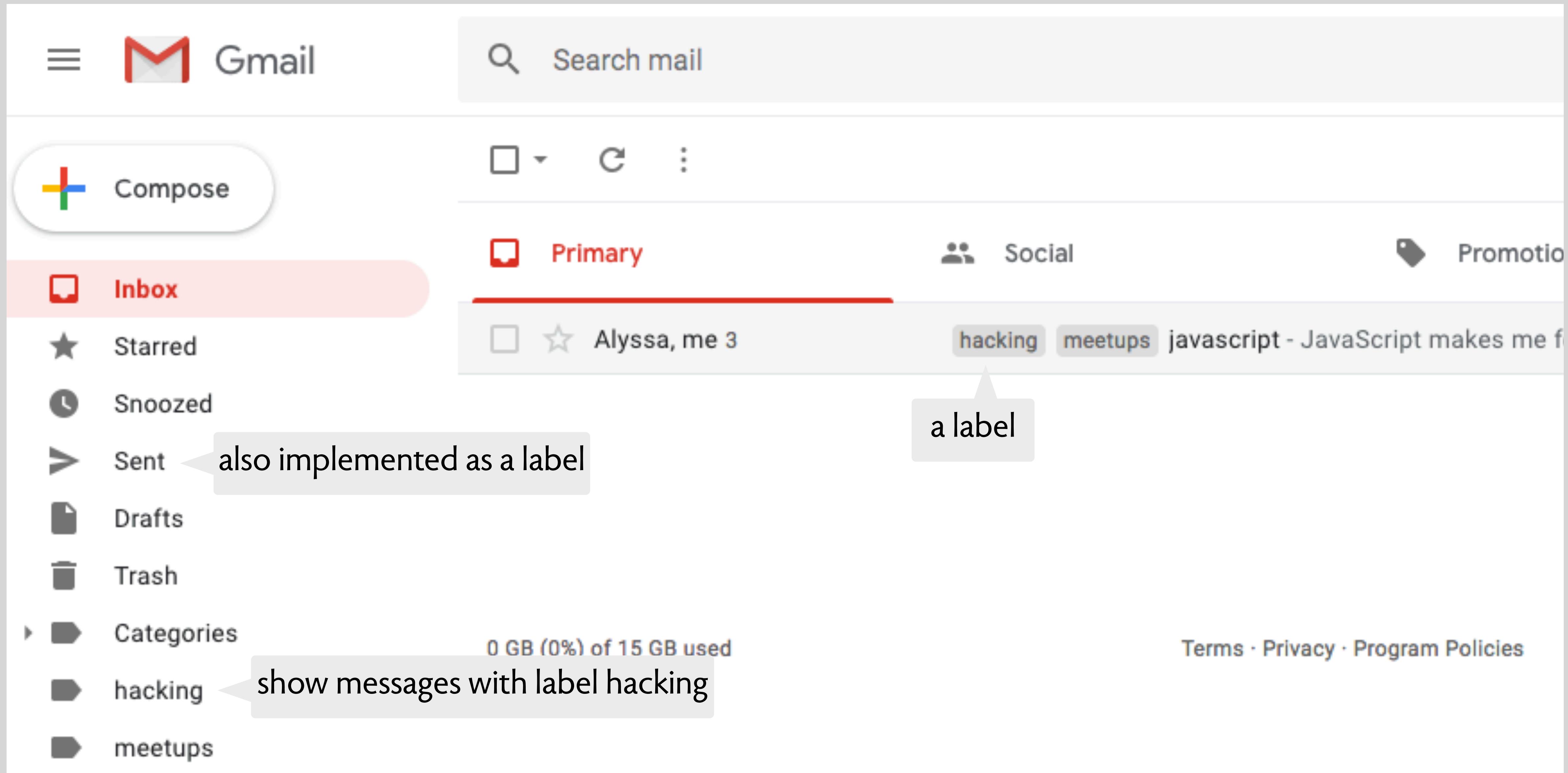
**concept** Labeling

**purpose** organize items

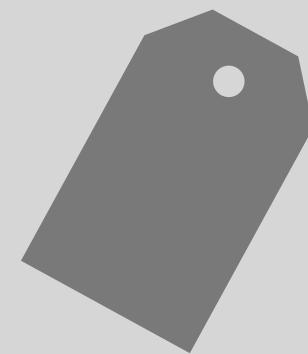
**principle** if you add a label to an item, then later you can filter on that label and find the item



# another application: the labeling concept in Gmail



# defining the concept's actions



**concept** Labeling

**purpose** organize items

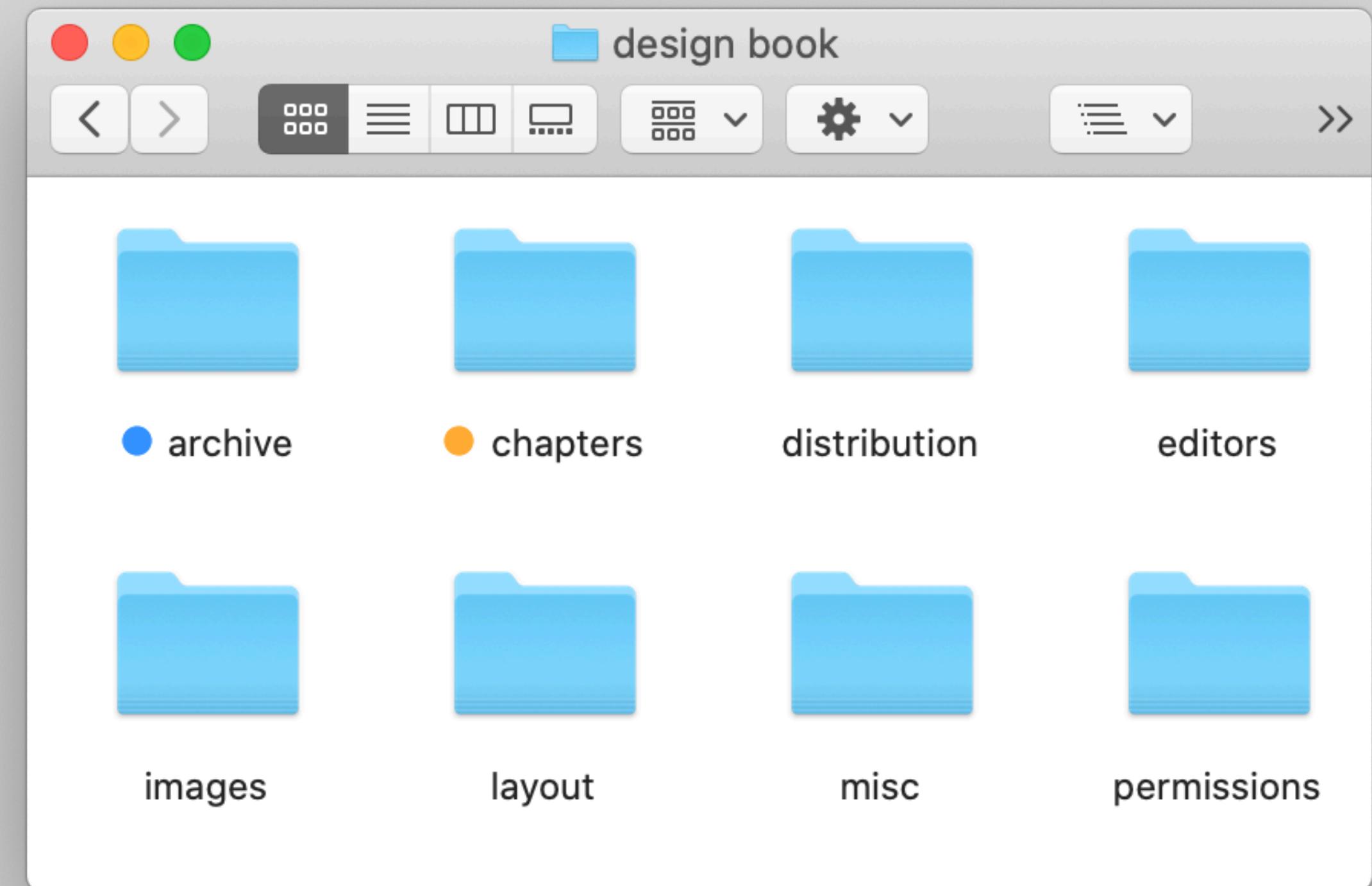
**principle** if you add a label to an item, then later you can filter on that label and find the item

## actions

add label to an item

remove label from an item

filter on a set of labels



# defining the concept's state



**concept** Labeling

**purpose** organize items

**principle** if you add a label to an item, then later you can filter on that label and find the item

**state**

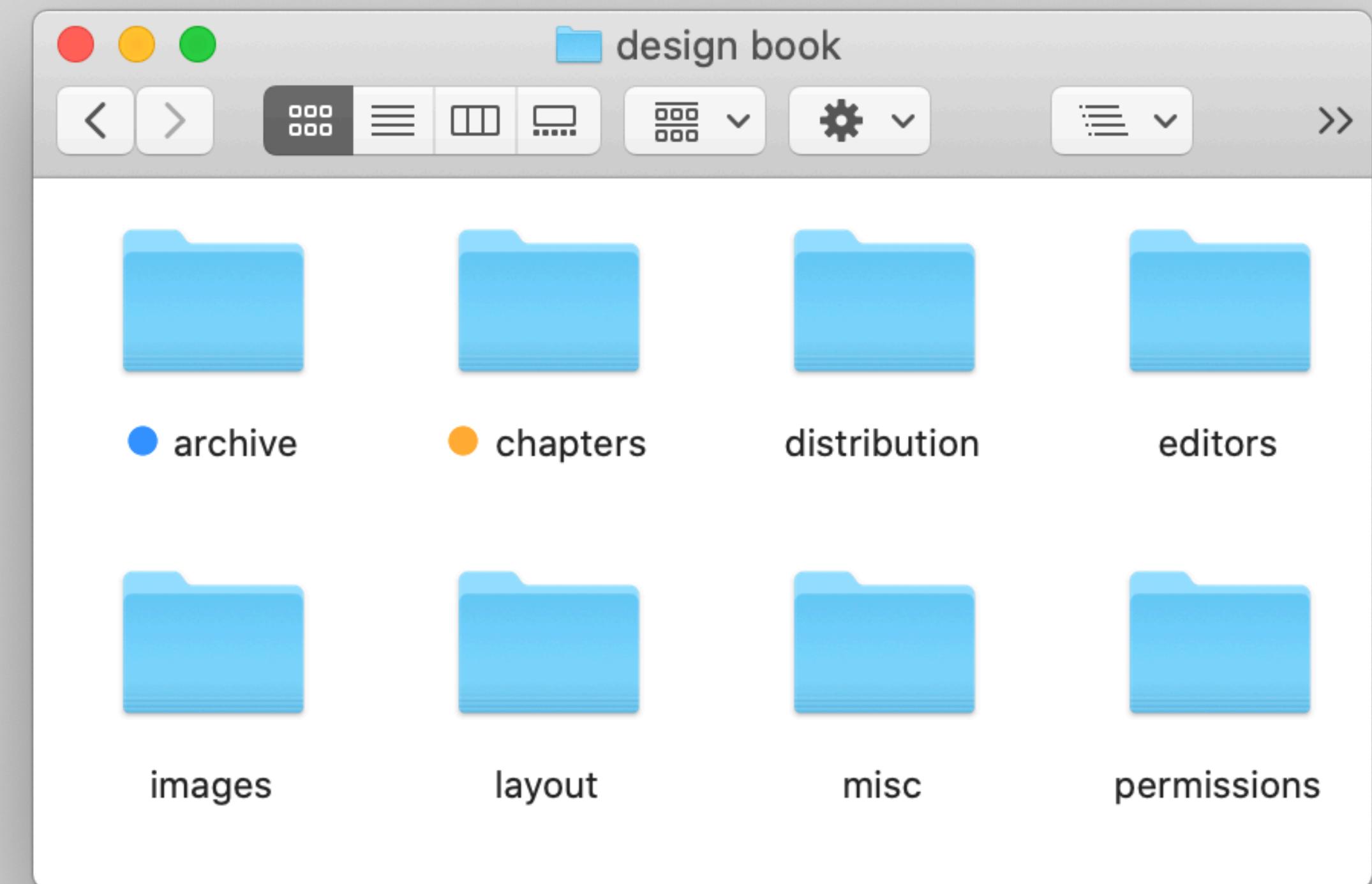
labels for each item

**actions**

add label to an item

remove label from an item

filter on a set of labels



# making the state precise



a type variable

**concept** Labeling [Item]

concept is generic

**purpose** organize items

**principle** if you add a label to an item, then later you can filter on that label and find the item

**state**

labels: Item -> **set Label**

**actions**

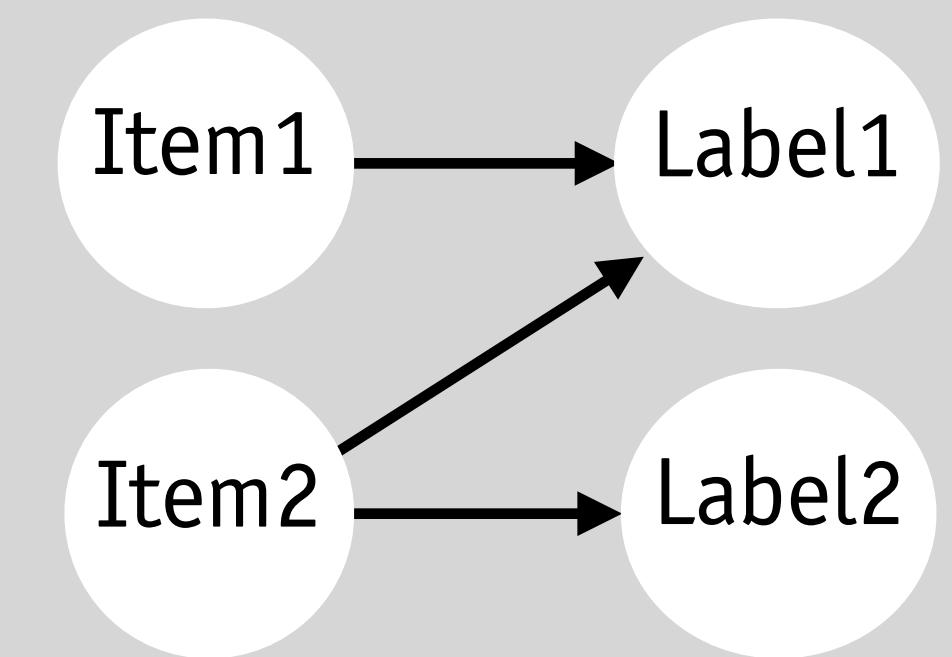
add label to an item

remove label from an item

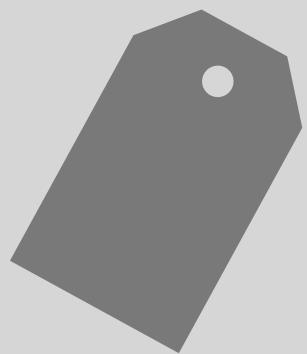
filter on a set of labels



Item1	Label1
Item2	Label1
Item2	Label2



# making the actions precise



**concept** Labeling [Item]

**purpose** organize items

**principle** if you add a label to an item, then later you can filter on that label and find the item

**state**

labels: Item  $\rightarrow$  set Label

**actions**

add (l: Label, i: Item)

i.labels += l    *add l to the labels of i*

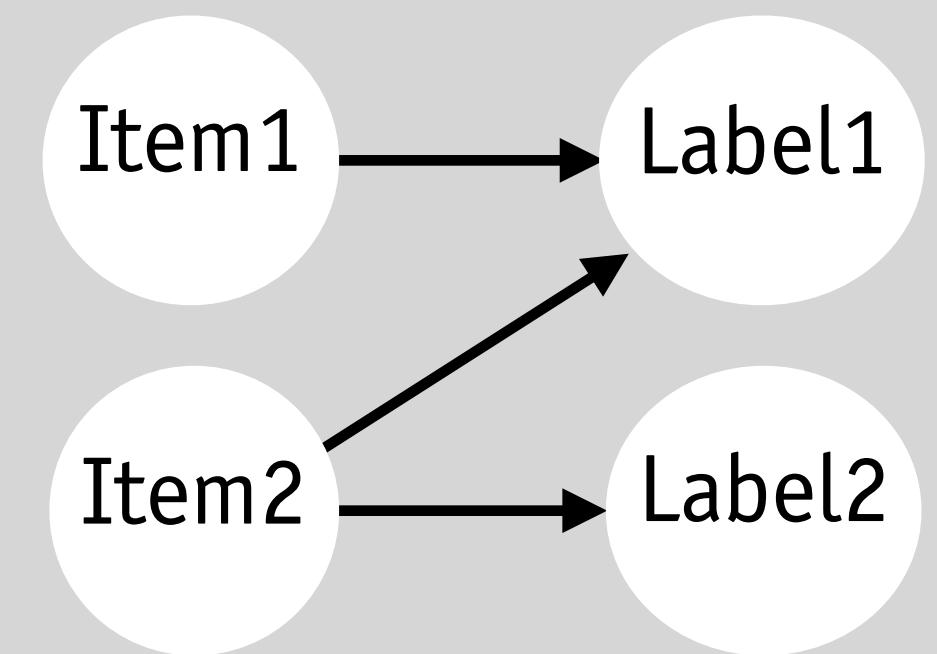
remove (l: Label, i: Item)

i.labels -= l

filter (ls: set Label): set Item

**return** {i: Item | ls in i.labels}

Item1	Label1
Item2	Label1
Item2	Label2



# check your understanding: how does an action update the state?



**concept** Labeling [Item]

**purpose** organize items

**principle** if you add a label to an item, then later you can filter on that label and find the item

**state**

labels: Item -> **set** Label

**actions**

add (l: Label, i: Item)

i.labels += l

remove (l: Label, i: Item)

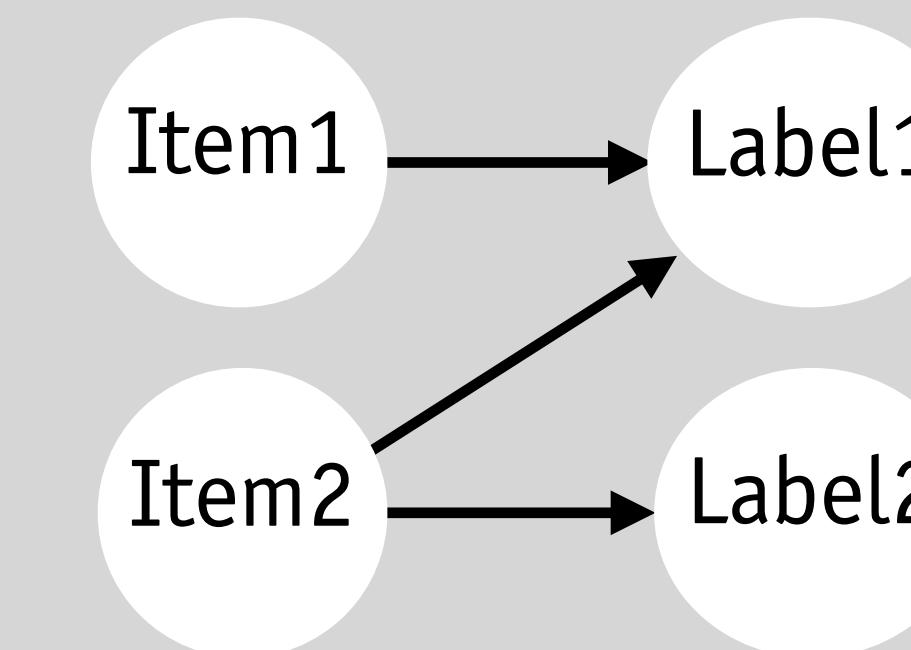
i.labels -= l

filter (ls: **set** Label): **set** Item

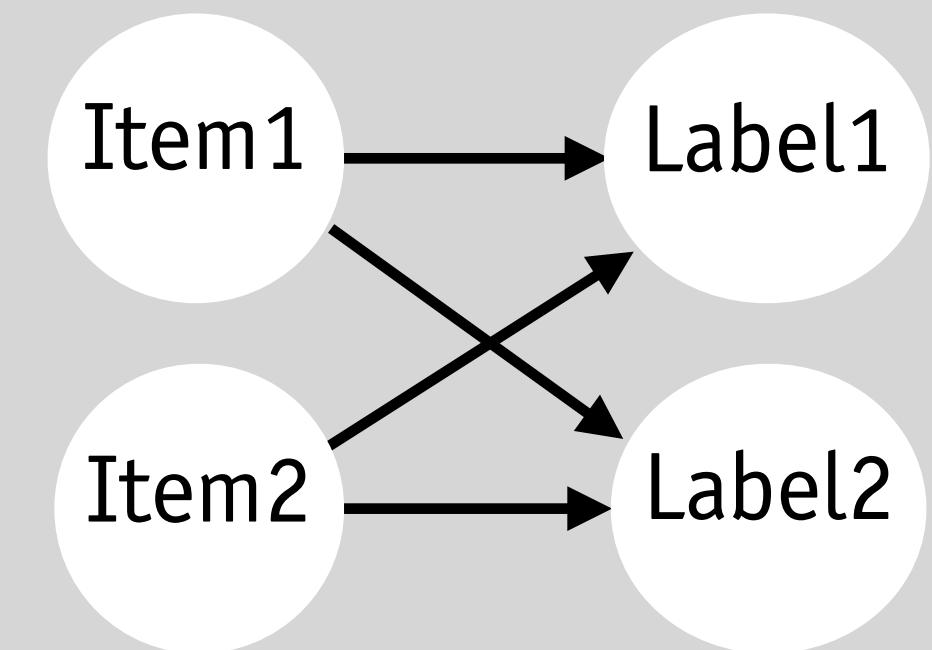
**return** {i: Item | ls **in** i.labels}

Item1	Label1
Item2	Label1
Item2	Label2

Item1	Label1
Item1	Label2
Item2	Label1



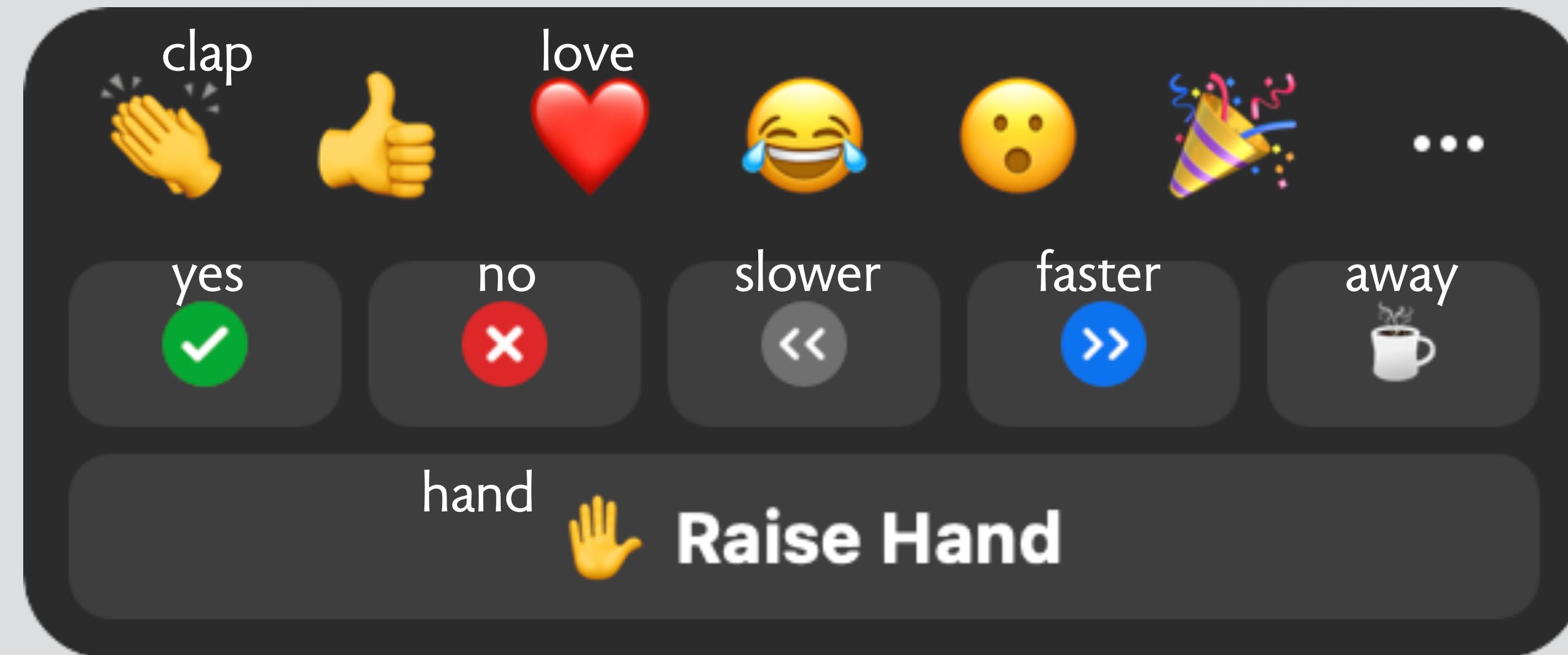
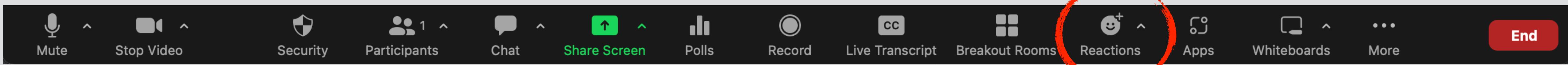
*before add (Label2, Item1)*



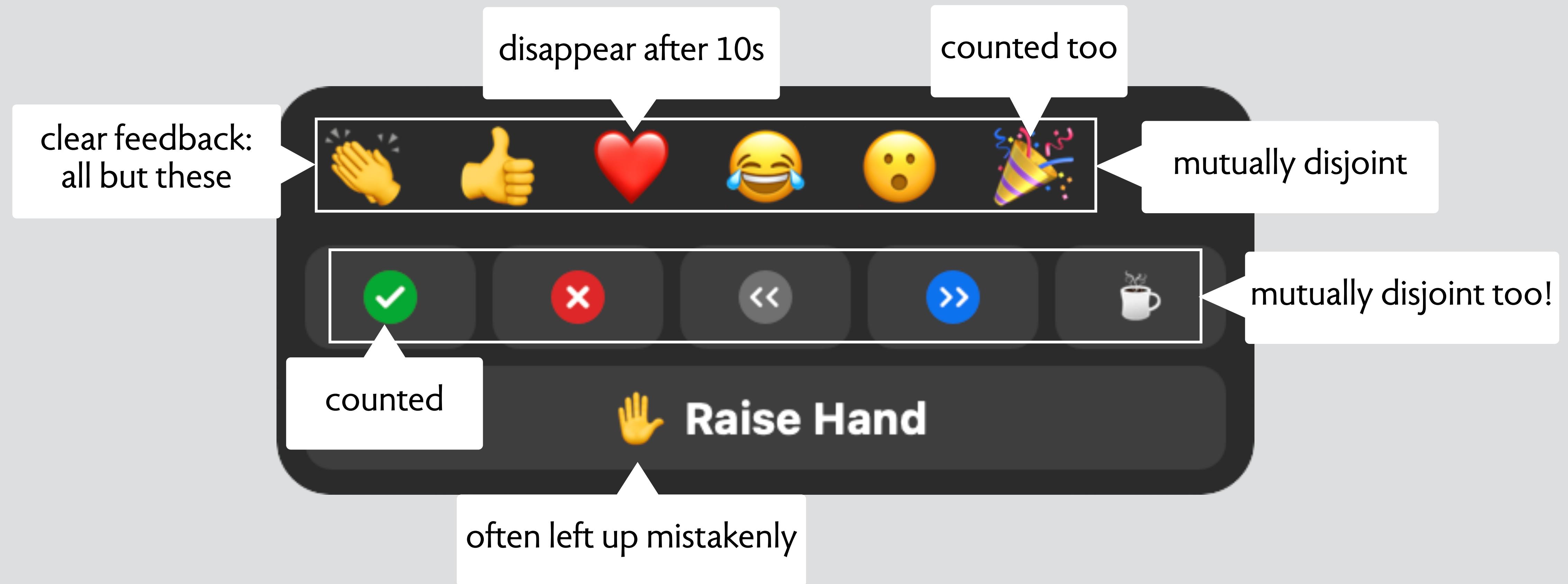
*after add (Label2, Item1)*

Zoom’s “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

exercise: redesigning  
Zoom’s “reactions”

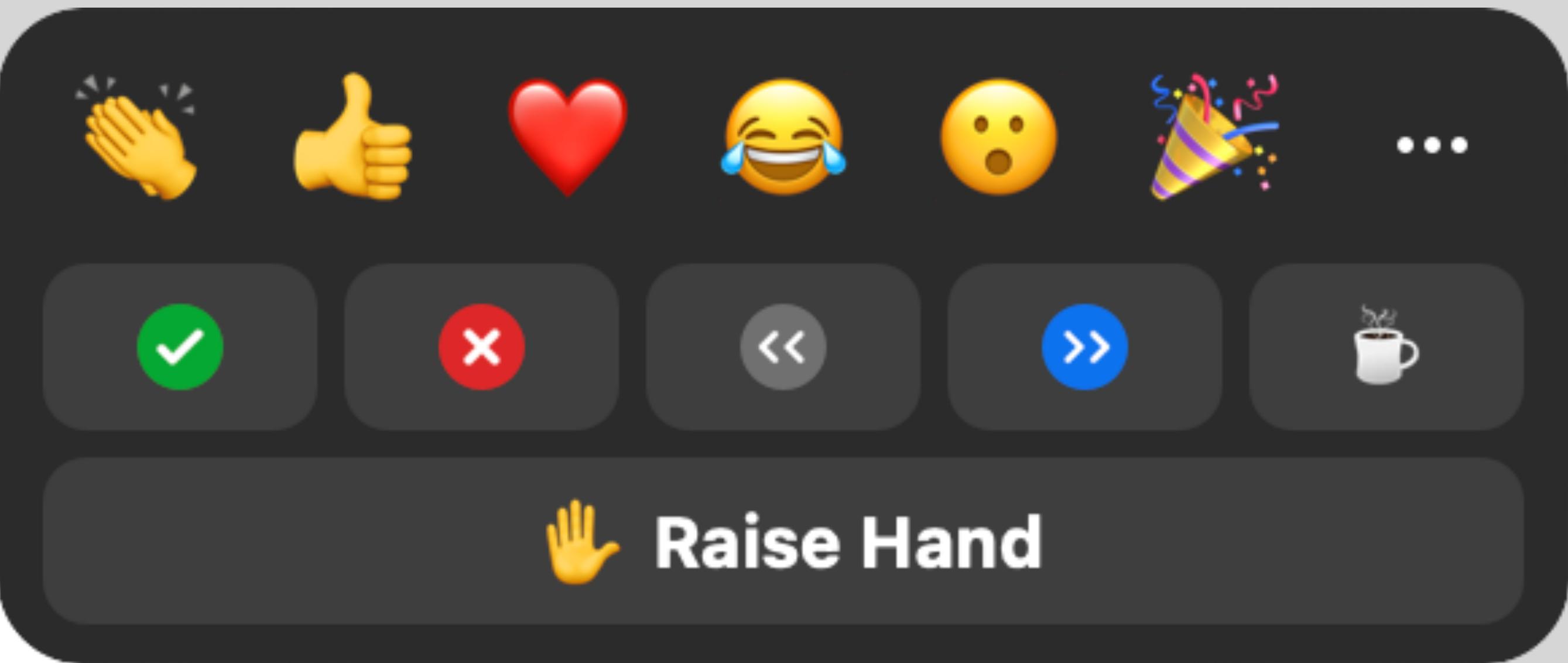
# can we do better?

## goals

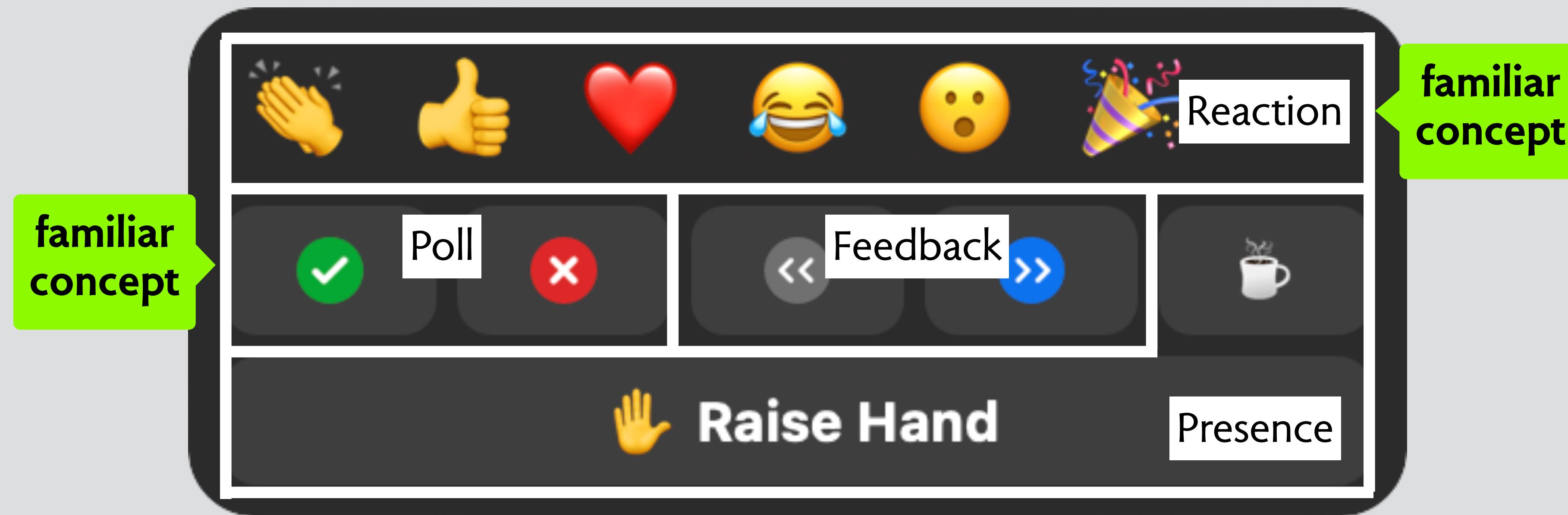
- break the behavior into a small set of concepts
- use familiar concepts whenever possible
- make each concept simple, robust & understandable
- leave some flexibility to synchronizations

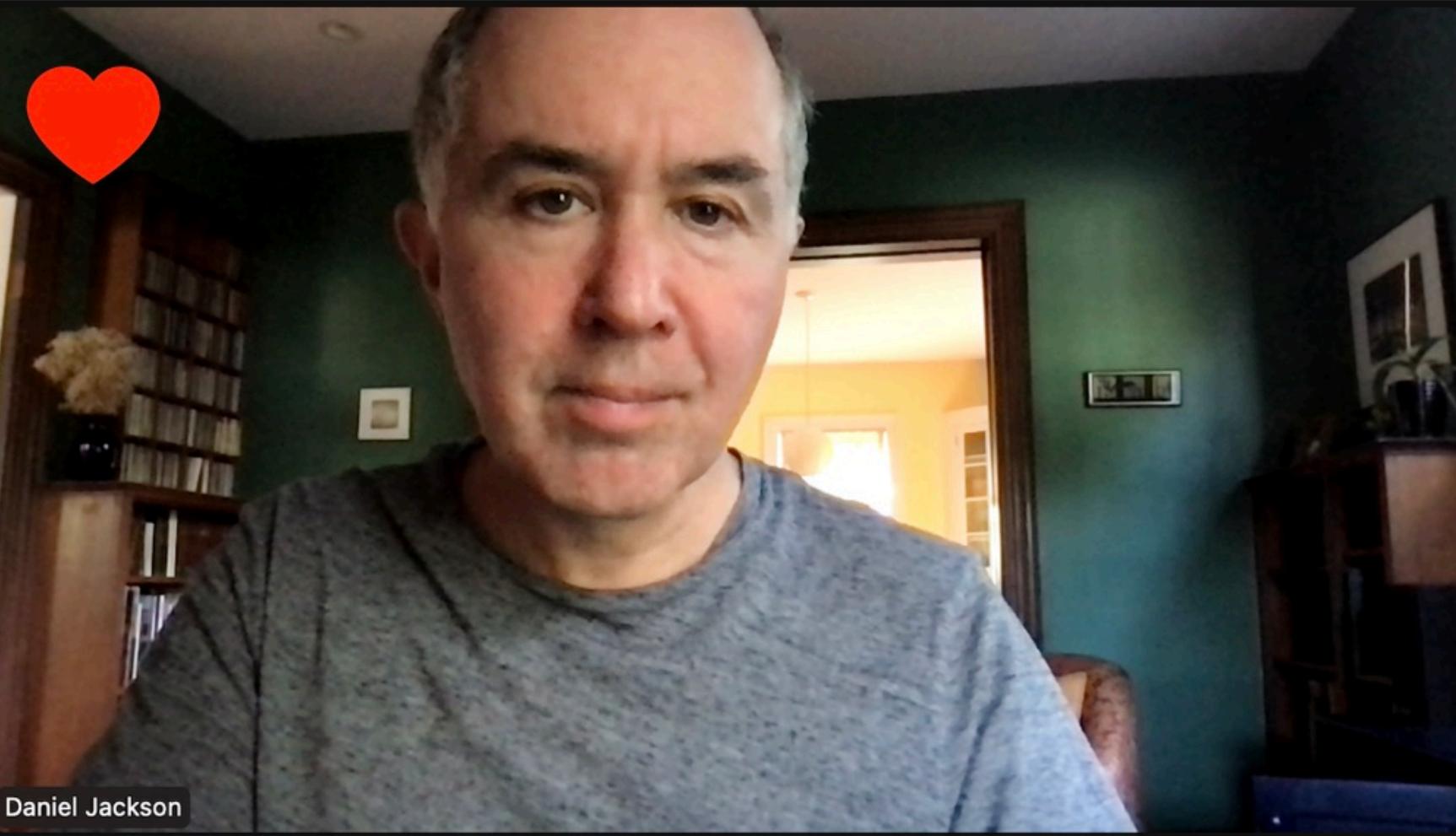
## strategy

1. factor roughly into concepts
2. outline each concept (name, purpose, OP, actions, state)
3. consider syncs, and adjust concepts if necessary
4. evaluate to ensure anomalies (esp. disjointness) are fixed



# my take: splitting into coherent concepts





## Presence

## Chat

## Reaction

## Feedback

Request to speak      Watching/listening

Speaking

I'm away

Audience

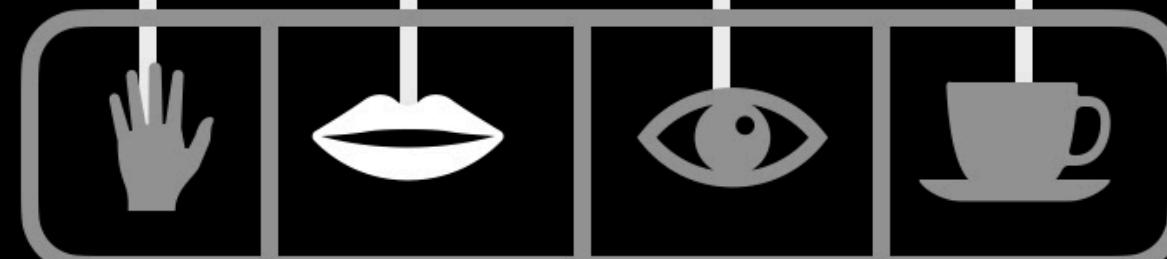
Other emoji

Recent emoji

Just right

Slow down

Speed up



Everyone ▾      Type here...



## **concept** Presence [User]

**purpose** manage modes of users in meeting

**principle** a user joins a meeting in listening mode, and can switch to requesting and (when called on) talking mode and then back again to listening

### **state**

```
let Mode =  
{listening, talking, requesting, absent}  
a set of users  
for each user  
a mode
```

### **actions**

```
join (u: User, m: Mode)  
change_to_mode (u: User, m: Mode)  
leave (u: User)  
is_present (u: User)
```

### **design questions**

what mode does a user join in?  
do we need an action to delete a poll?  
can a user change their response?  
what can host control?

**concept** Audio [User]

**purpose** manage audio muting

**principle** a user joins a meeting muted and can unmute to speak and then mute again to avoid being heard

**state**

a set of users  
for each user  
whether muted or not

**actions**

join (u: User)  
mute (u: User)  
unmute (u: User)  
leave (u: User)

**design questions**

what mode does a user join in? where set?  
is video hiding the same concept? part of this?

## **concept** Polling [User]

**purpose** get group opinion on questions

**principle** you open a poll, users respond and tallies of yes/no are available

### **state**

a set of polls  
for each poll

a question text

a set of responses  
for each response

a responding user

a yes or no response

yes-total, no-total // derived

### **actions**

open (question: Text): Poll

respond (u: User, p: Poll, r: Bool)

close (p: Poll)

### **design questions**

should polling go beyond binary?

can you vote both yes and no?

do we need an action to delete a poll?

can a user change their response?

### **looking forward**

do we really need a feedback concept? isn't it the same as this one?

**concept** SpeakerFeedback [User]

**purpose** offer feedback to speaker

**principle** users can request that the speaker go slower or faster, and an ongoing tally is available

**state**

a set of users requesting slower  
a set of users requesting faster

**actions**

request\_slower (u: User)  
request\_faster (u: User)  
clear (u: User)

**design questions**

should requests expire?

should requests be clearable by speaker?

**concept** Reaction [User]

**purpose** let users convey reactions

**principle** users react and the reactions  
are visible to all

**state**  
a set of reactions  
for each reaction  
a reacting user  
an emoji

**actions**  
react (u: User, e: Emoji)

**design questions**

can users react with multiple emojis?  
should reactions expire?  
should there be a clear action?

# Presence/Audio

```
when Presence.change_to_mode(u, listening)
sync
  Audio.mute (u)
```

```
when Presence.change_to_mode(u, speaking)
sync
  Audio.unmute (u)
```

## design questions

unmute when going absent?  
or let user set this as preference?  
same syncs for video hiding?

# Presence/SpeakerFeedback

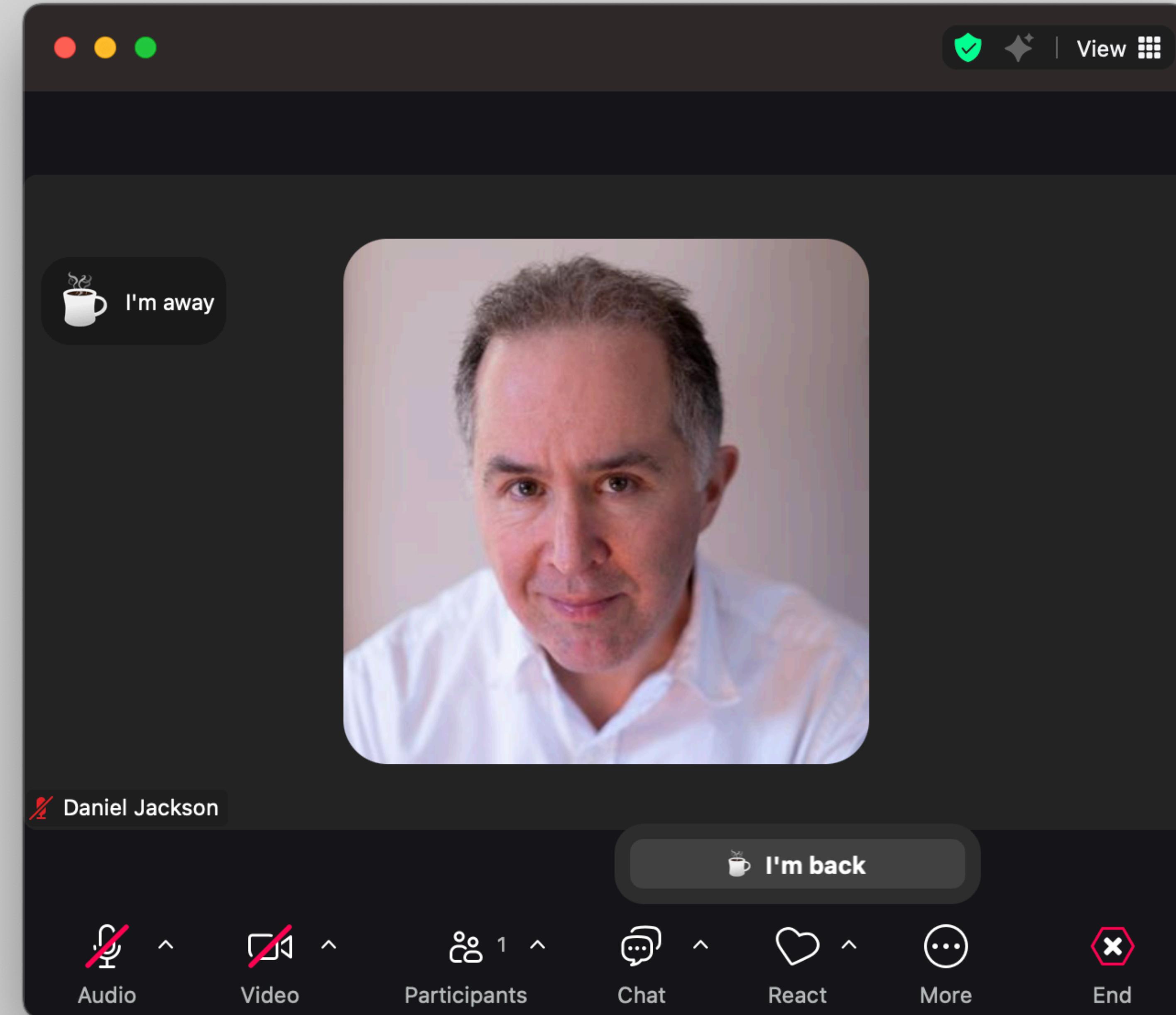
```
when SpeakerFeedback.request_slower (u)
sync
Presence.is_present (u)
```

```
when SpeakerFeedback.request_slower (u)
sync
Presence.change_to_mode (u, listening)
```

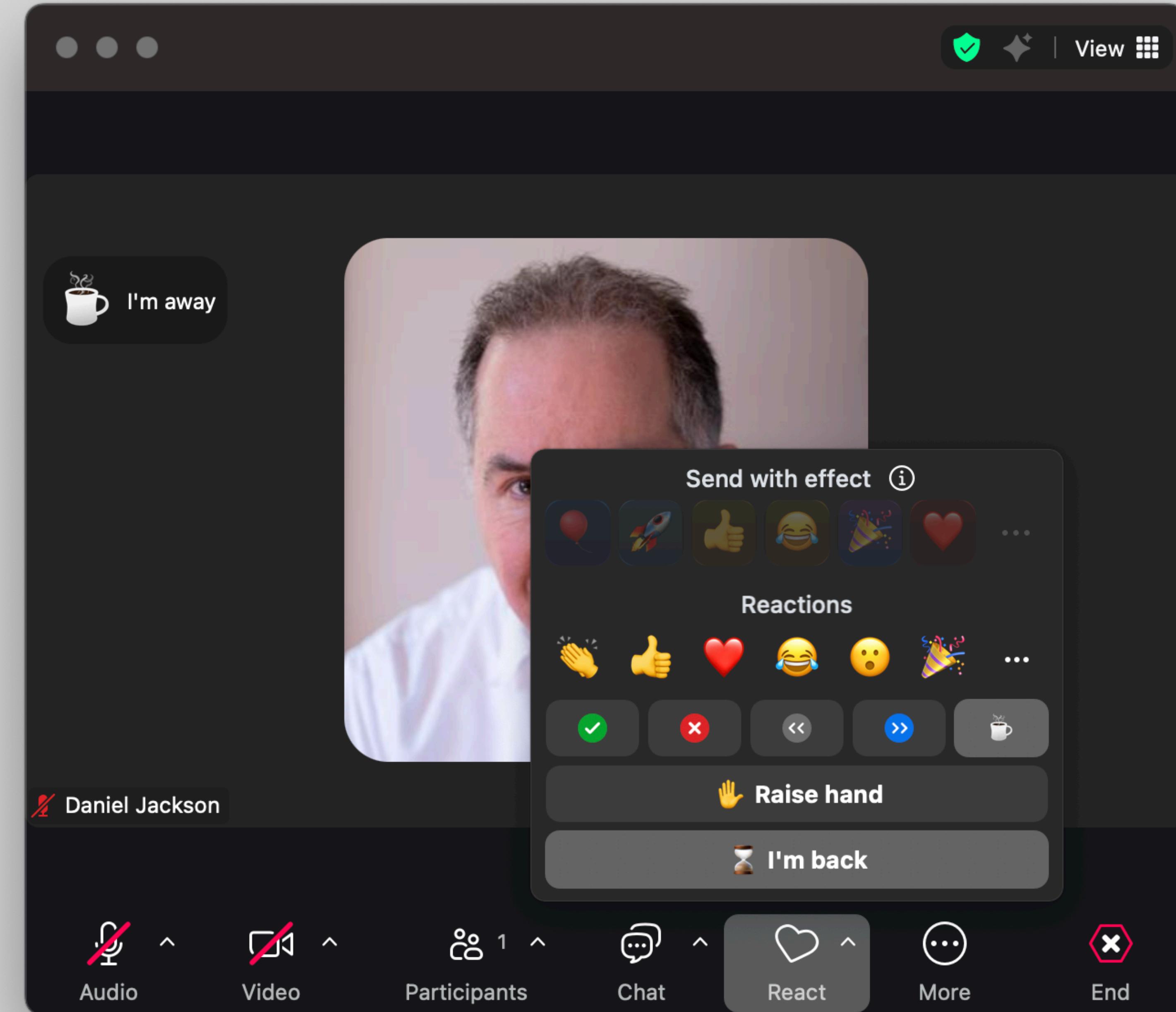
## design questions

also prevent poll response?  
also prevent unmuting?

# looking at Zoom's latest design (1)

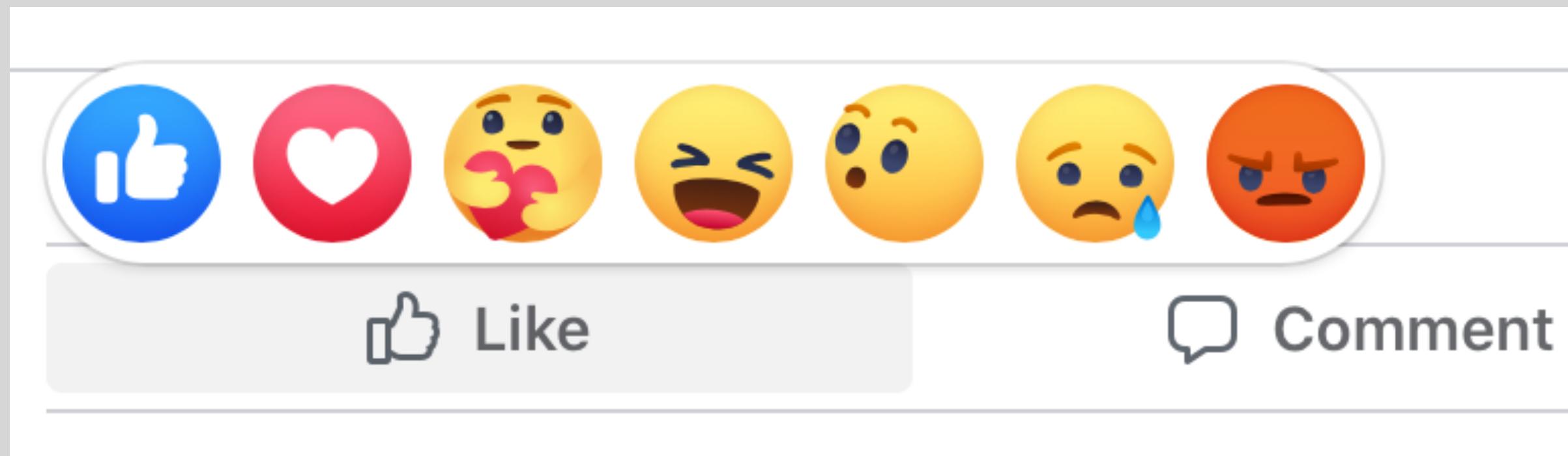


# looking at Zoom's latest design (2)



# Facebook's “reactions”

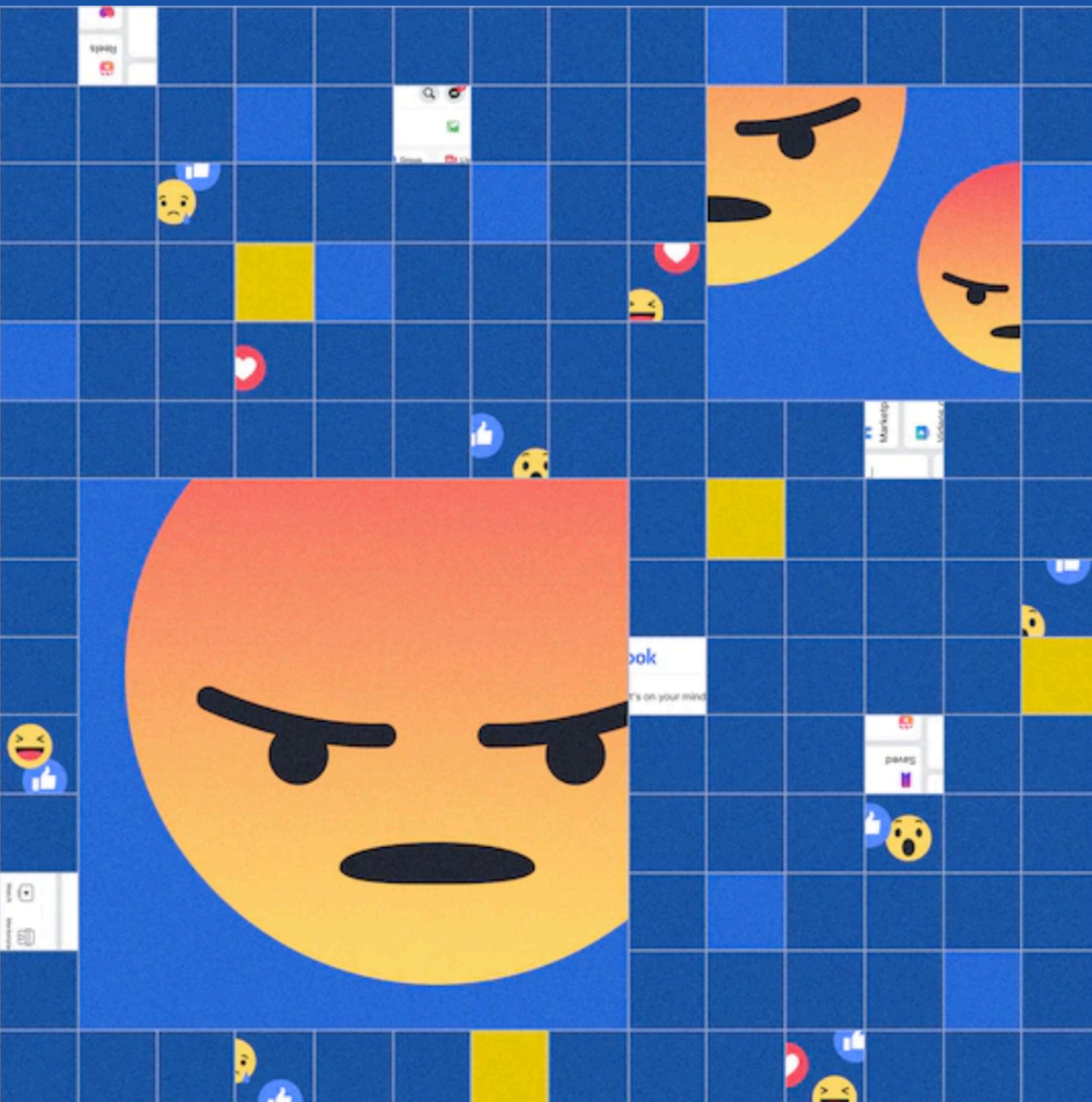
# do angry reactions promote posts?



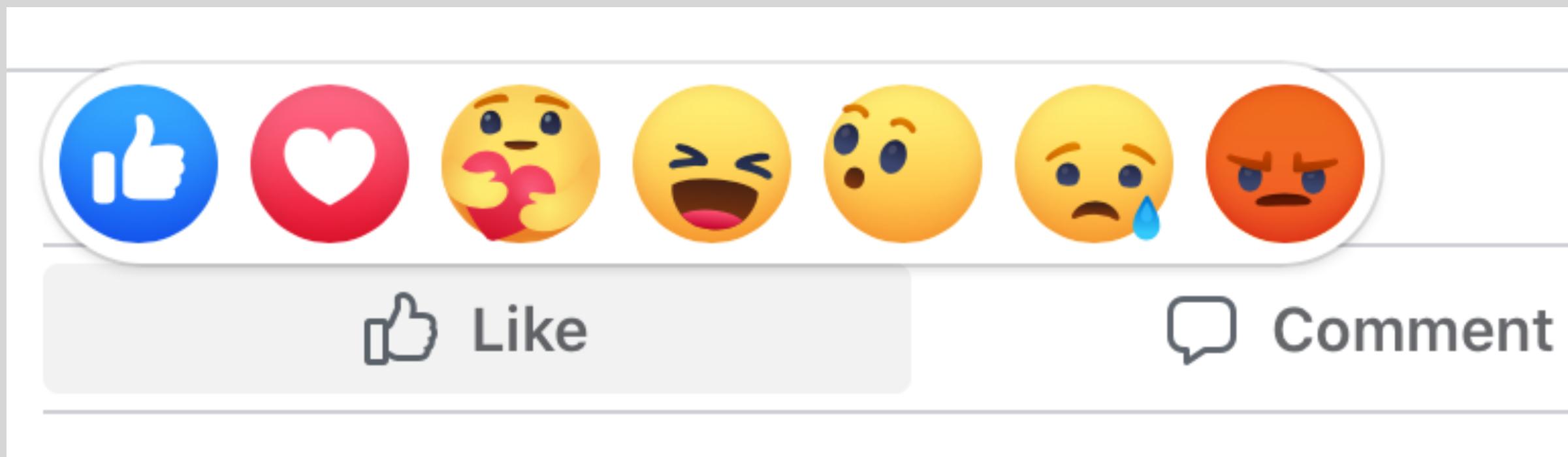
**Facebook under fire**

# Five points for anger, one for a ‘like’: How Facebook’s formula fostered rage and misinformation

Facebook engineers gave extra value to emoji reactions, including ‘angry,’ pushing more emotional and provocative content into users’ news feeds



exercise: can you analyze this in terms of concepts?



# three concepts we saw before

## concept Upvote

**purpose** rank items by popularity

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

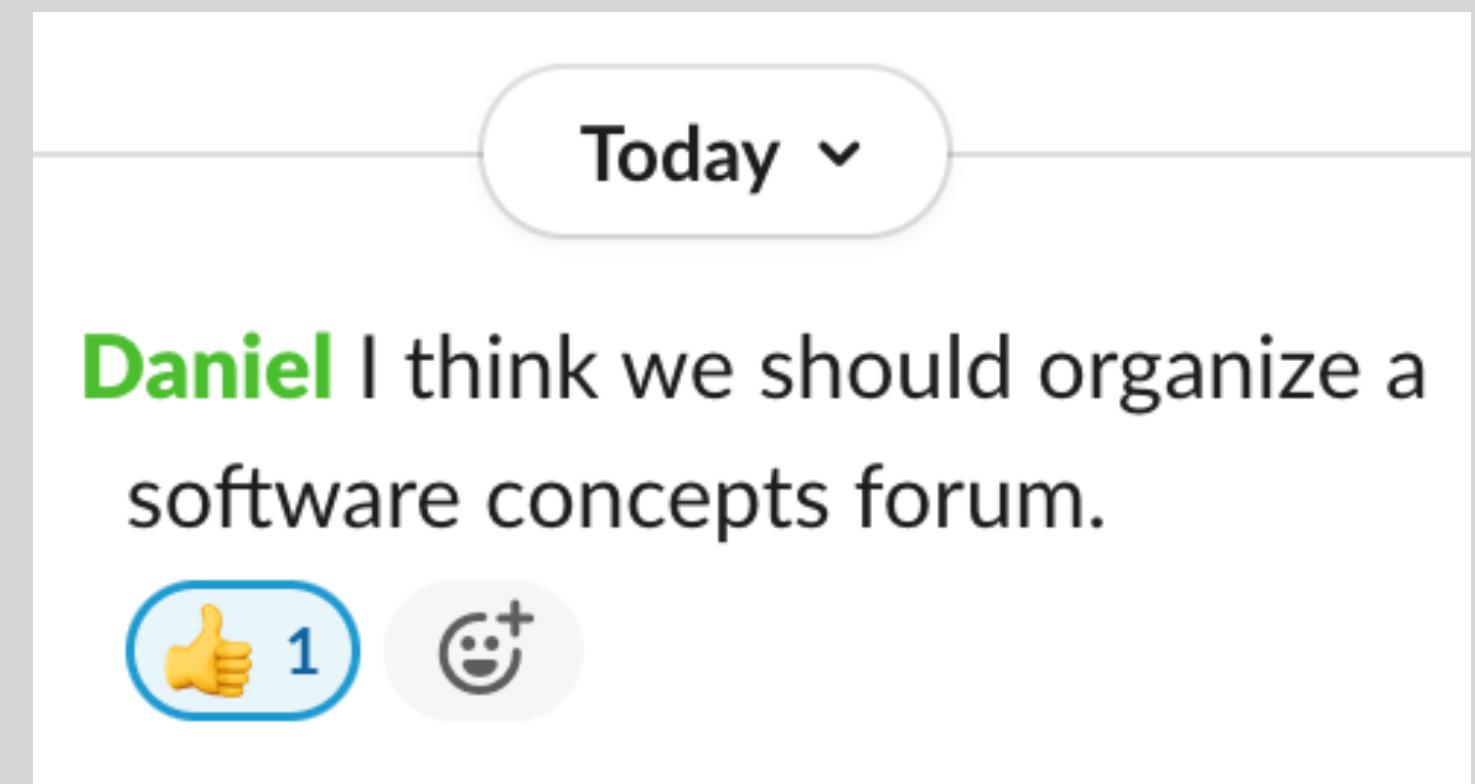
8 This is homework and I'm having a  
are the definitions of the objects:

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

## concept Reaction

**purpose** support quick responses

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



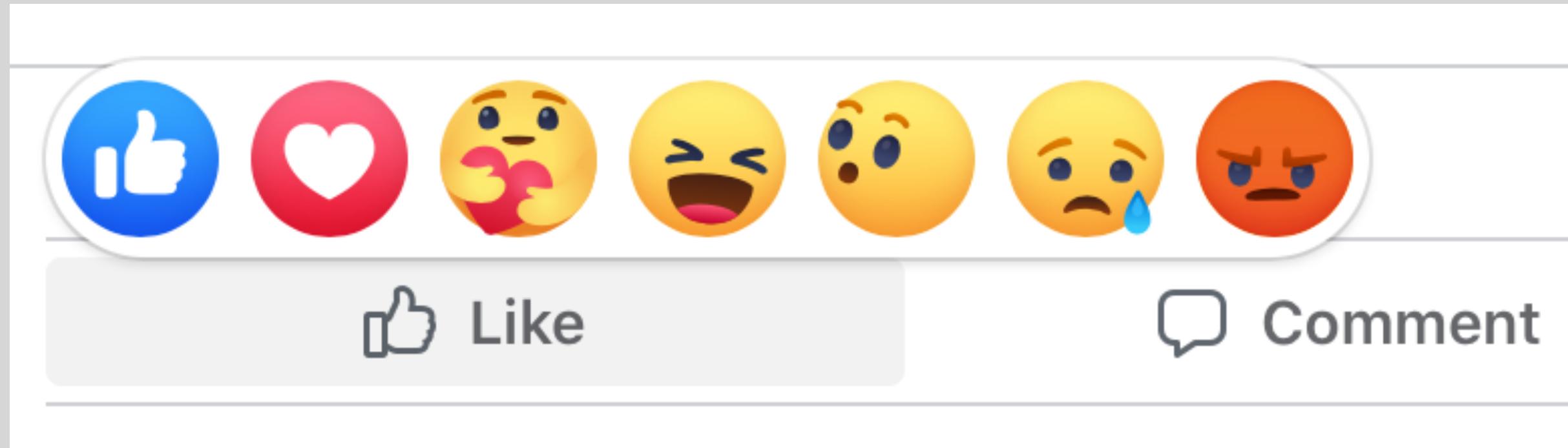
## concept Recommendation

**purpose** infer user preferences

**principle** user's likes lead to ranking of kinds of items, determining which items are recommended



# a concept diagnosis



**concept** Upvote

**purpose** rank items by popularity

**actions**

upvote (u: User, i: Item) .....

...



**concept** Reaction

**purpose** convey emotion to author

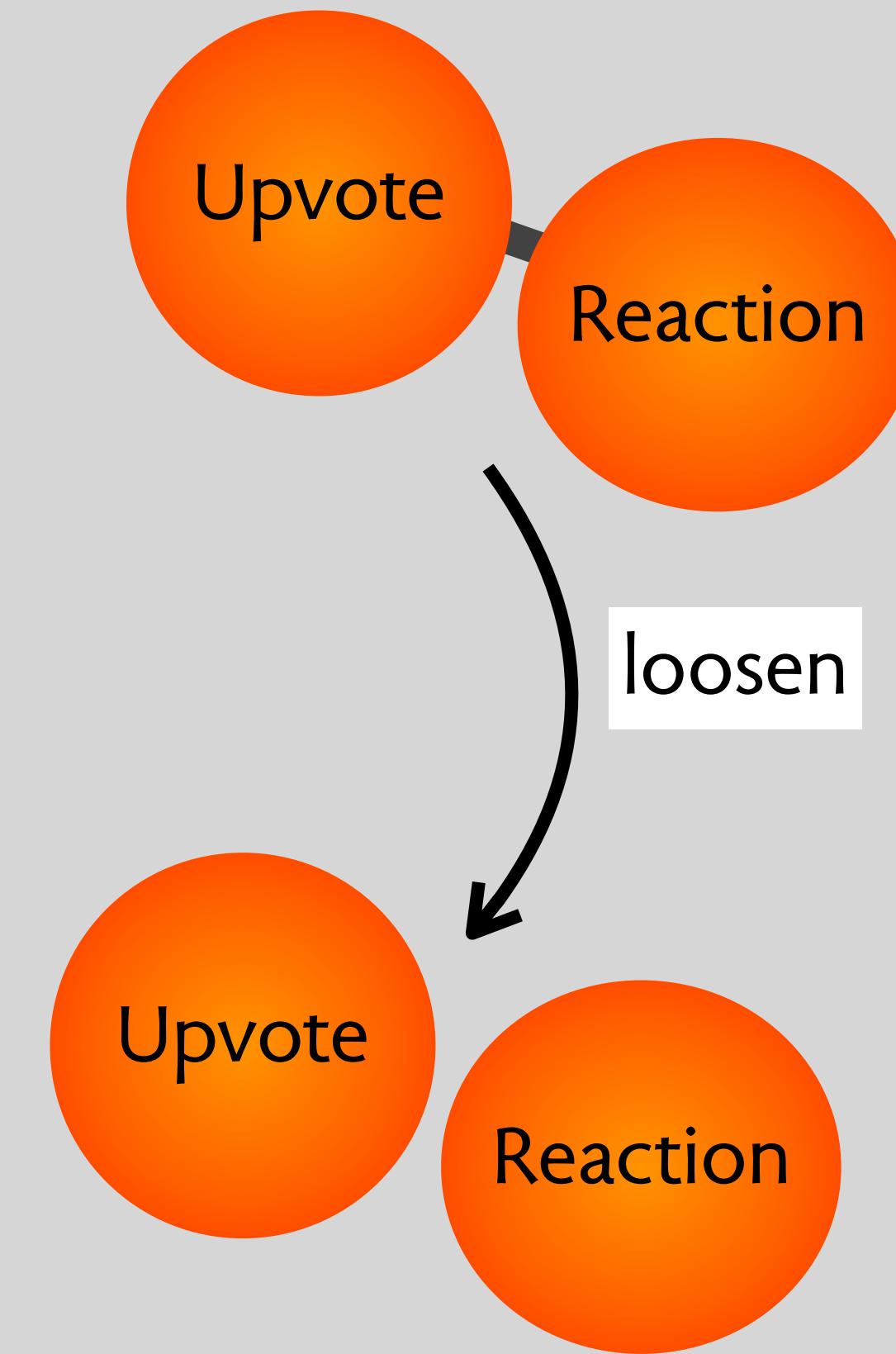
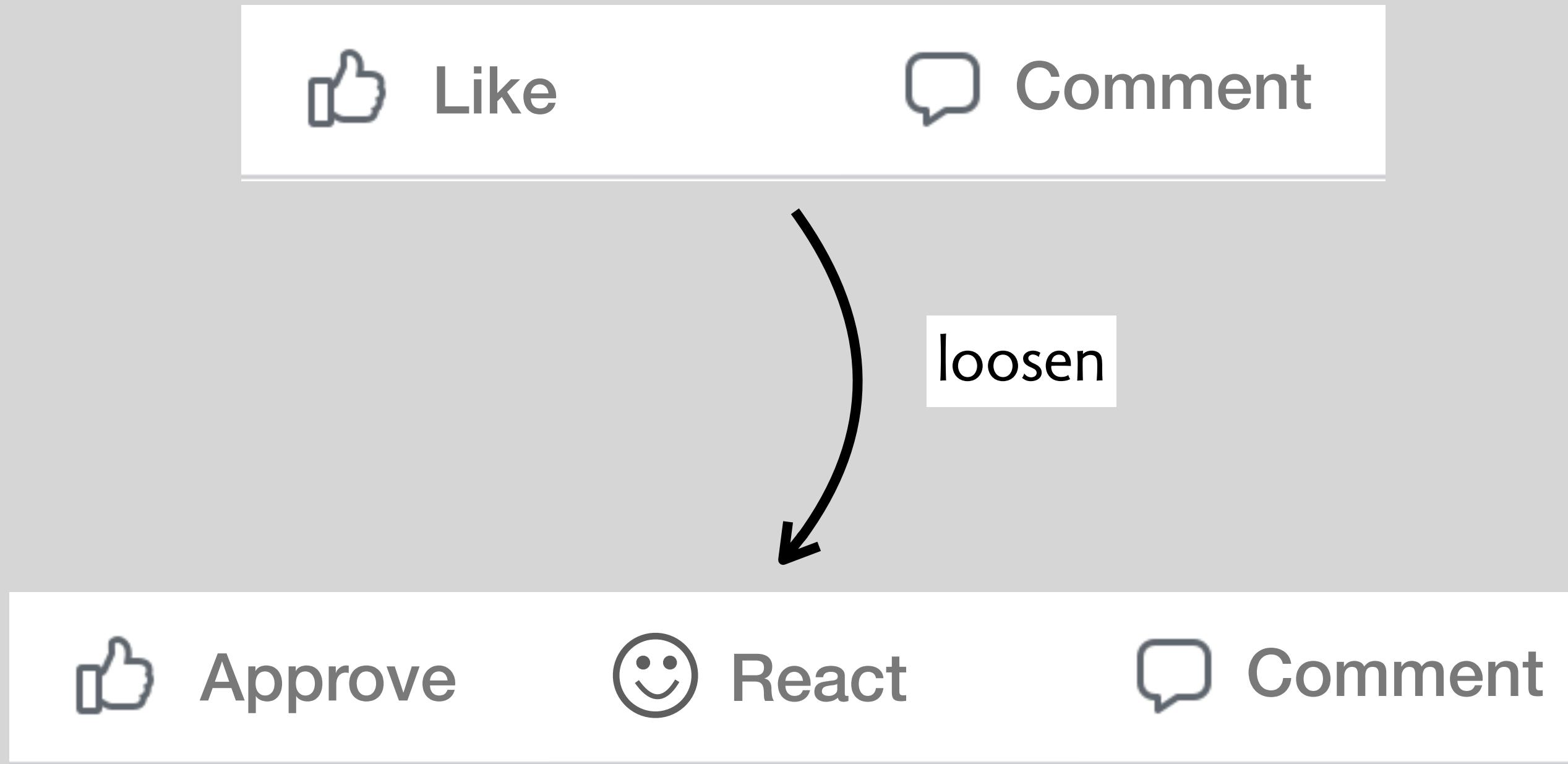
**actions**

reactAngry (u: User, i: Item) .....

...

unwanted  
sync?

# a facebook loosening: a good or bad design move?



exercise:  
Autodesk concepts

**consider an area of functionality in an Autodesk product**  
limit to a single scenario, eg evaluating metrics against one target-set

**find a couple of concepts that**  
covers the essential functionality  
make concepts smaller to separate concerns  
make concepts larger to encapsulate related functionality

**consider synchronizations between concepts**  
have you left enough flexibility?  
can you synchronize as tightly as you want?

**concept** Evaluation [Subject]

**purpose**

**principle**

**state**

**actions**

new\_outcome (): Outcome

new\_target (m: Metric, val: Real, lo, hi: Real + None): Target

add\_target (o: Outcome, t: Target)

new\_analysis (s: Subject, readings: set (Metric, Real)): Analysis

evaluate (o: Outcome, a: Analysis): Report

**terminology**

using current catalog terms

outcome is desired outcome, set of targets

metric is something like “square footage”

analysis is set of metrics with values

subject is generic term for model etc

report is result of evaluation, currently unspecified

**design questions**

who defines metrics and where are they stored?

**concept** Analysis [Subject]

**purpose**

**principle**

**state**

**actions**

analyze (s: Subject): set (Metric, Real)

**design questions**

how to decouple analysis from model?

takeaways

# disentangling: bad smells and design moves

**complex & ad hoc**  
functionality, special cases

**confused purpose**  
not clear what it's for

**overloading**  
1 concept : N purposes



**make it orthogonal**  
so more options for user

**make it familiar**  
recognize an existing concept

**make it reusable**  
factor out a handy concept

**make it generic**  
concept works more widely

**make it customizable**  
by changing syncs

what's next

**disentangling: a kind of refactoring**  
existing functionality conflates concepts  
disentangling separates them out

**can you just invent the right concepts?**  
as you design a new function, embody in concepts

**the QDM**  
a general strategy for inventing effective concepts