

# Hierarchical Clustering, Topic Modeling

slides by  
George Chen  
Carnegie Mellon University  
Fall 2017

# Going from Similarities to Clusters

There's a whole zoo of clustering methods

Two main categories we'll talk about:

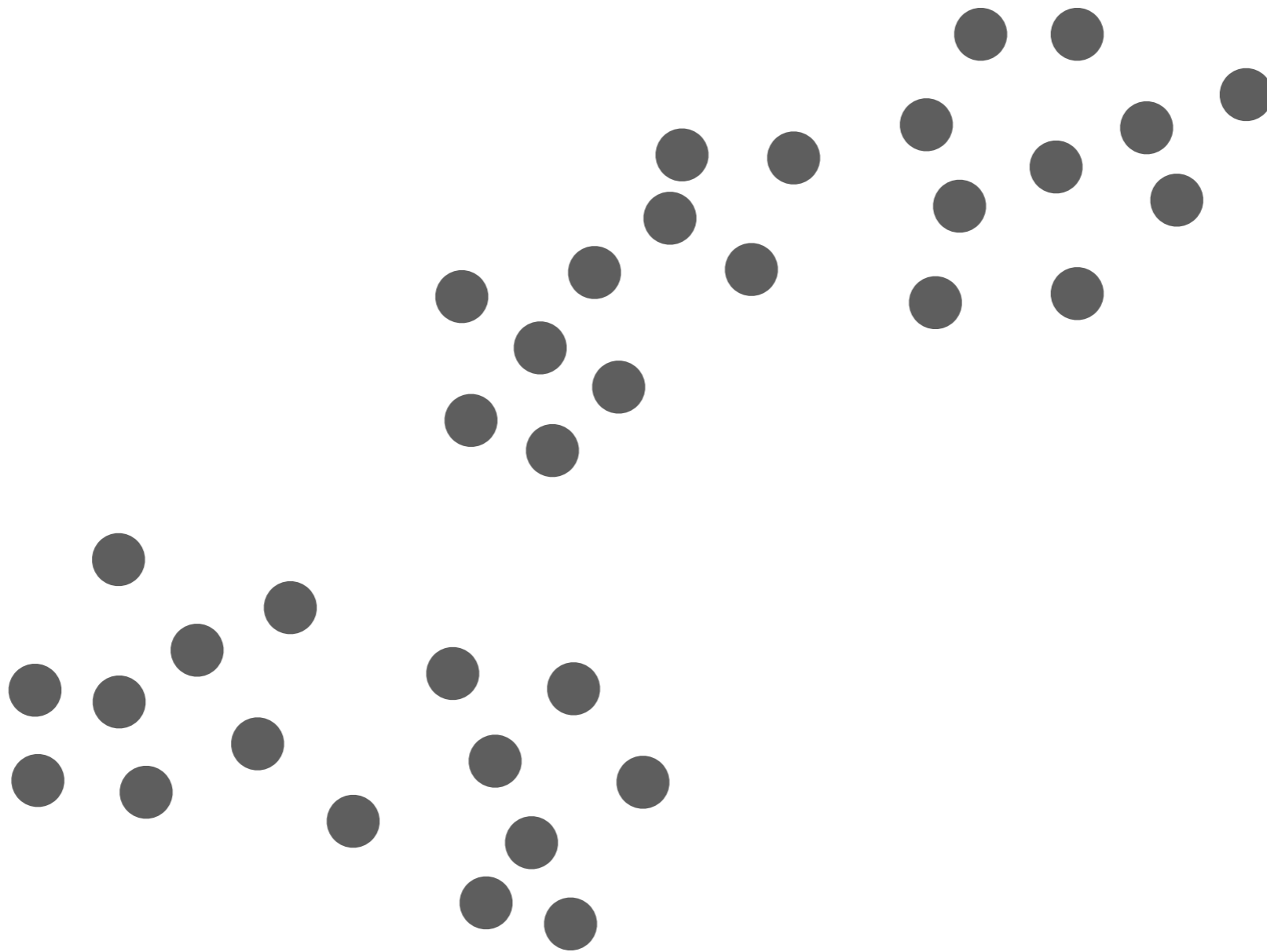
## **Generative models**

1. Pretend data generated by specific model with parameters
2. Learn the parameters ("fit model to data")
3. Use fitted model to determine cluster assignments

## **Hierarchical clustering**

- Top-down: Start with everything in 1 cluster and decide on how to recursively split
- Bottom-up: Start with everything in its own cluster and decide on how to iteratively merge clusters

# Divisive Clustering

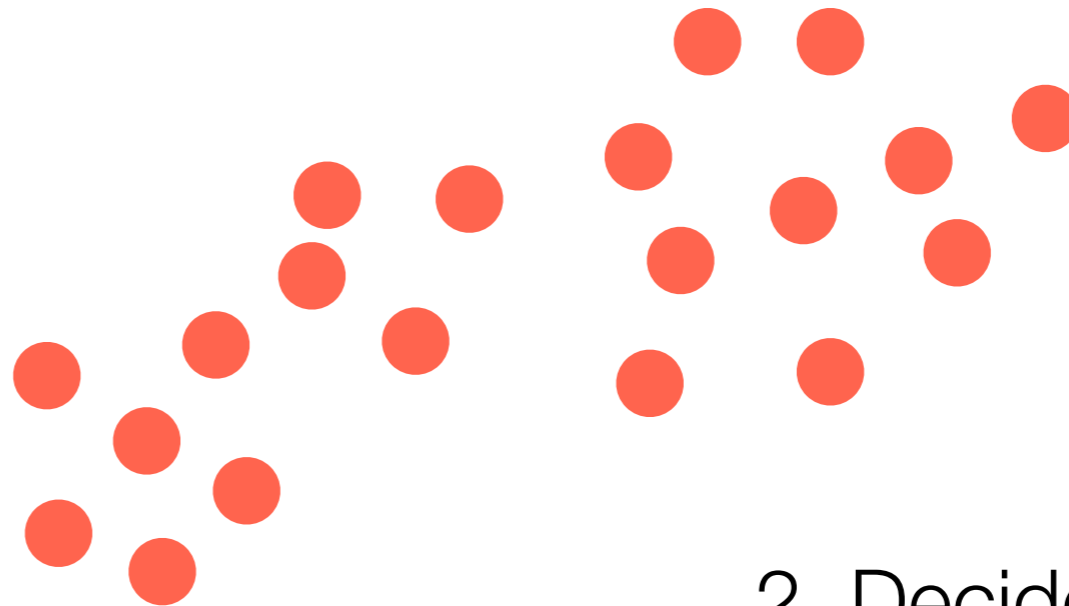
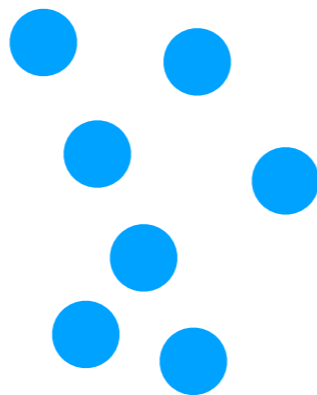
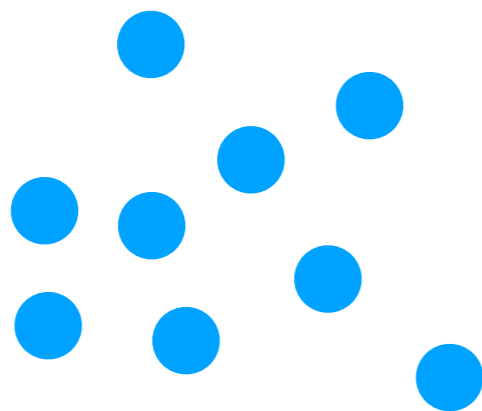


# Divisive Clustering

0. Start with everything in the same cluster

1. Use a method to split the cluster

(e.g.,  $k$ -means, with  $k = 2$ )



2. Decide on next cluster to split

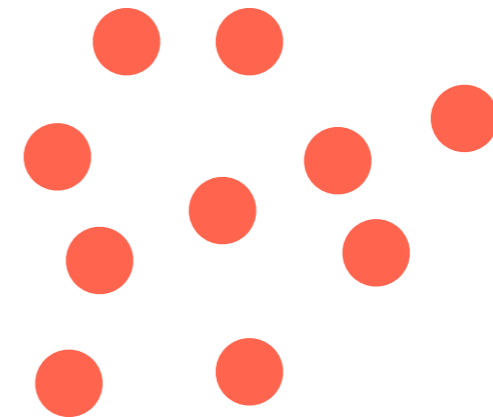
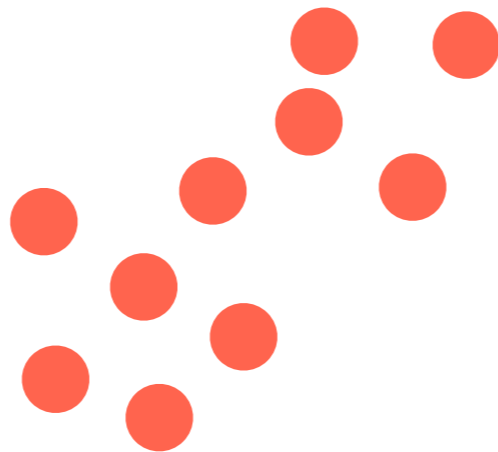
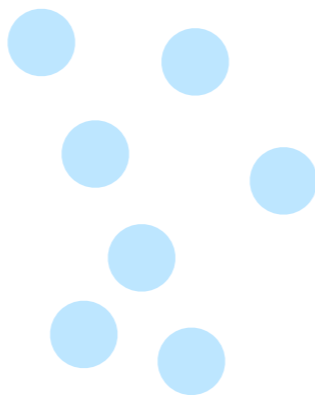
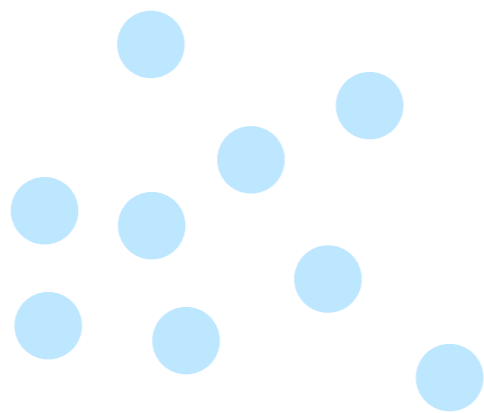
(e.g., pick cluster with highest RSS)

# Divisive Clustering

0. Start with everything in the same cluster

1. Use a method to split the cluster

(e.g.,  $k$ -means, with  $k = 2$ )



2. Decide on next cluster to split

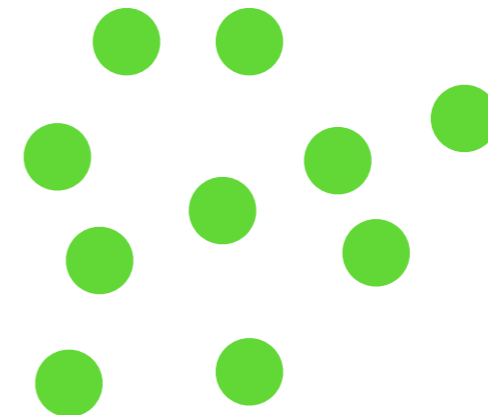
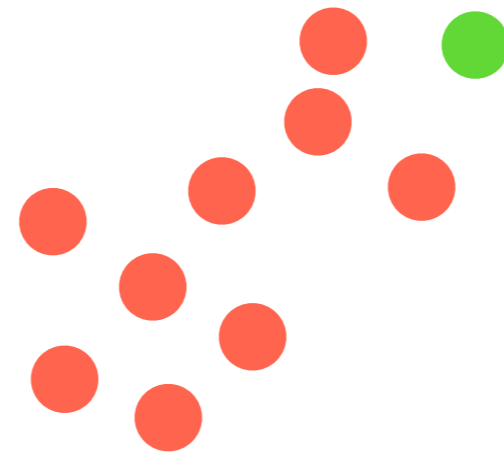
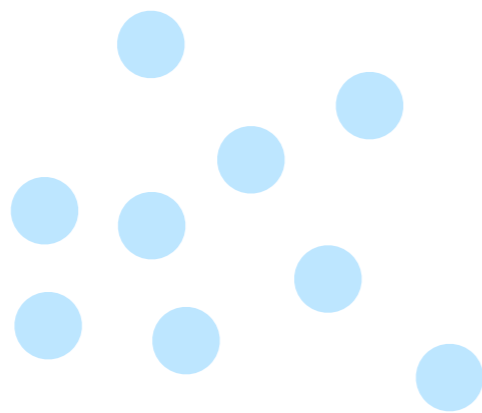
(e.g., pick cluster with highest RSS)

# Divisive Clustering

0. Start with everything in the same cluster

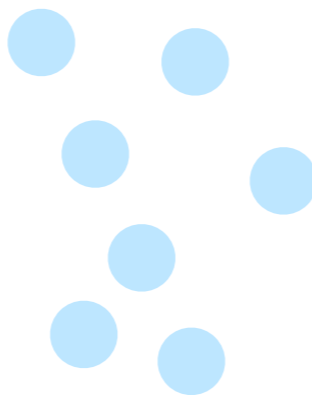
1. Use a method to split the cluster

(e.g.,  $k$ -means, with  $k = 2$ )



2. Decide on next cluster to split

(e.g., pick cluster with highest RSS)

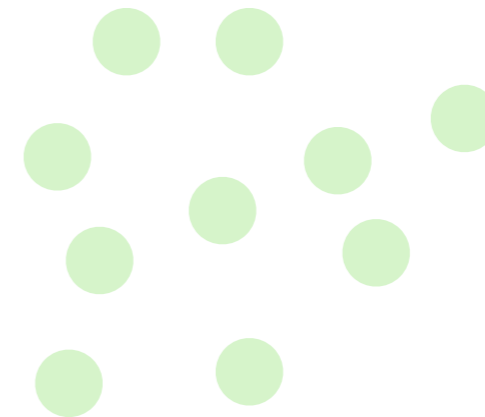
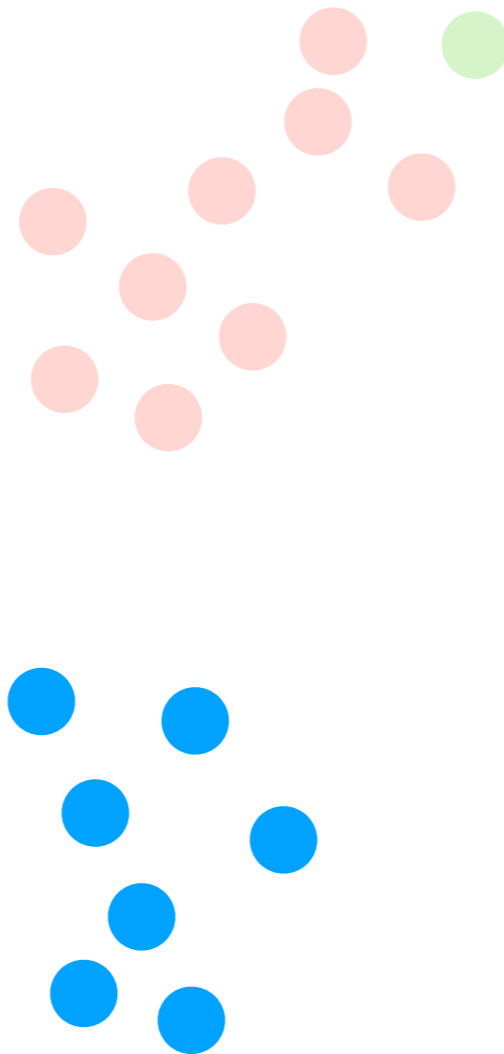
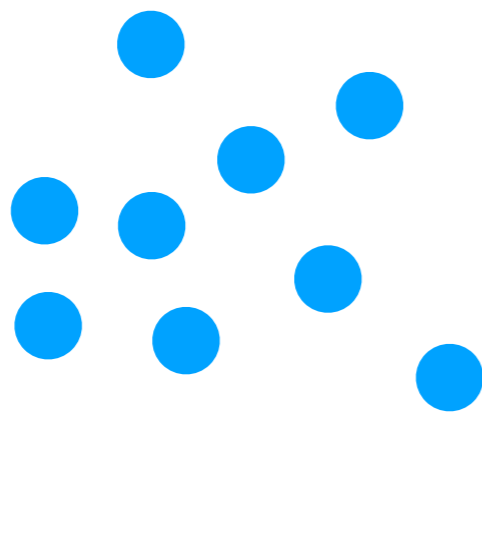


# Divisive Clustering

0. Start with everything in the same cluster

1. Use a method to split the cluster

(e.g.,  $k$ -means, with  $k = 2$ )



2. Decide on next cluster to split

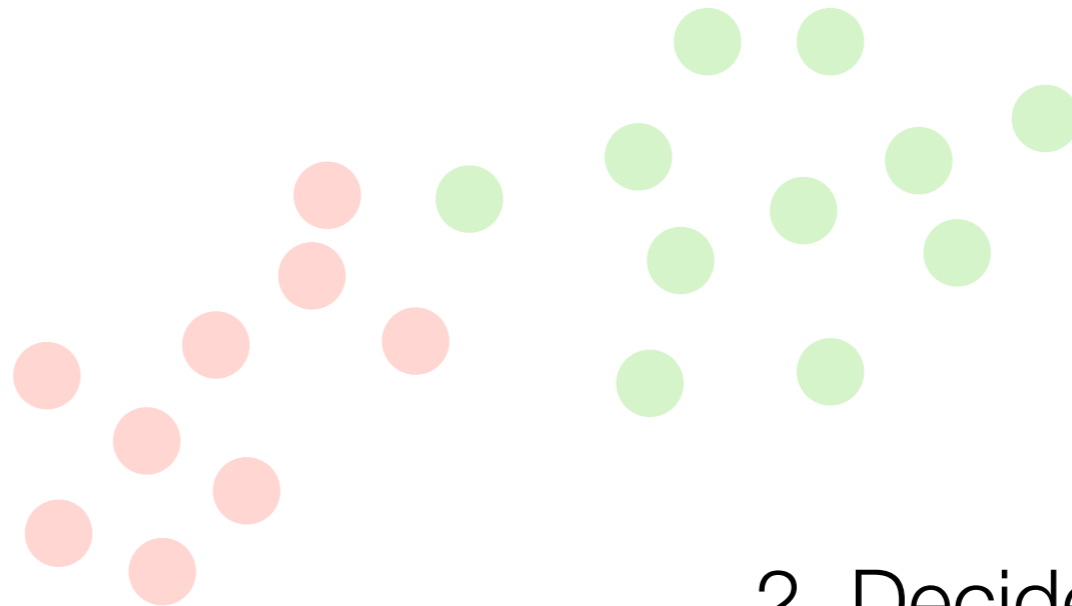
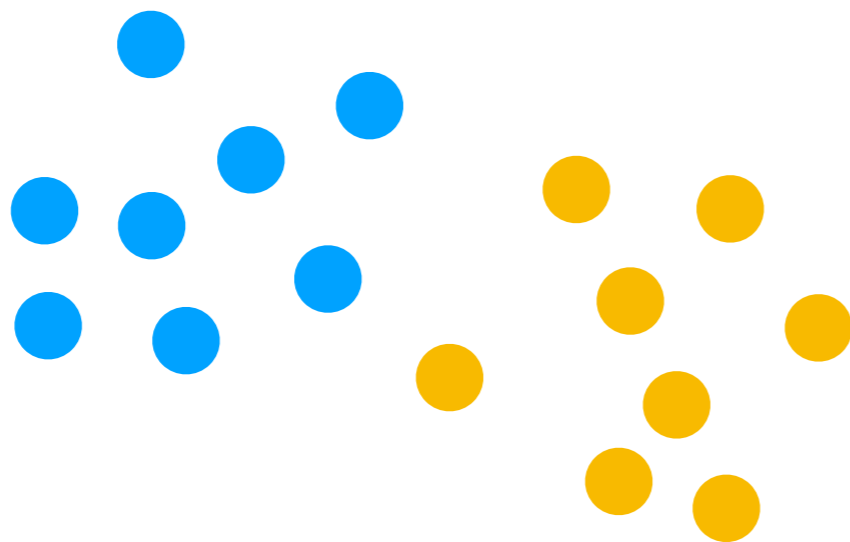
(e.g., pick cluster with highest RSS)

# Divisive Clustering

0. Start with everything in the same cluster

1. Use a method to split the cluster

(e.g.,  $k$ -means, with  $k = 2$ )



2. Decide on next cluster to split

(e.g., pick cluster with highest RSS)

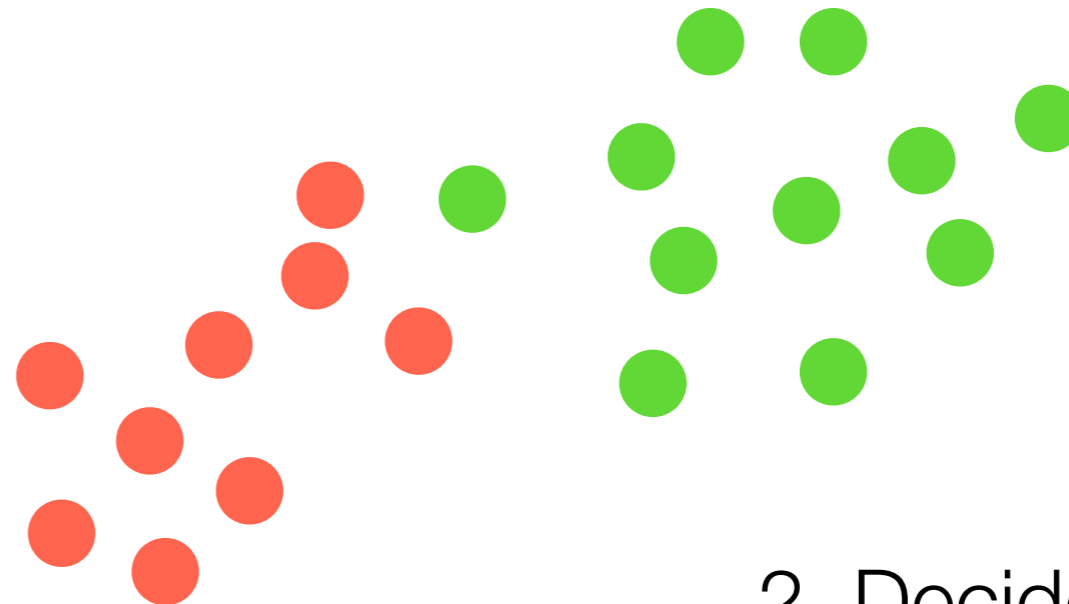
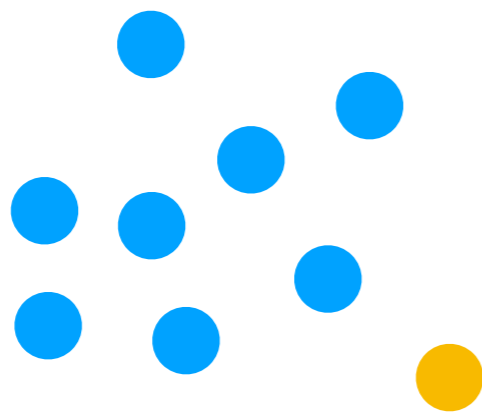


# Divisive Clustering

0. Start with everything in the same cluster

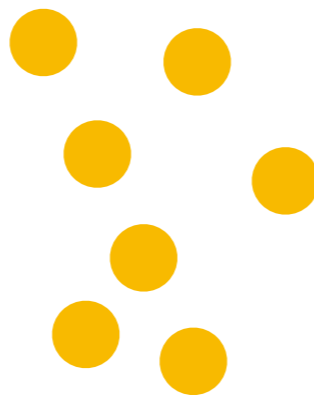
1. Use a method to split the cluster

(e.g.,  $k$ -means, with  $k = 2$ )



2. Decide on next cluster to split

(e.g., pick cluster with highest RSS)

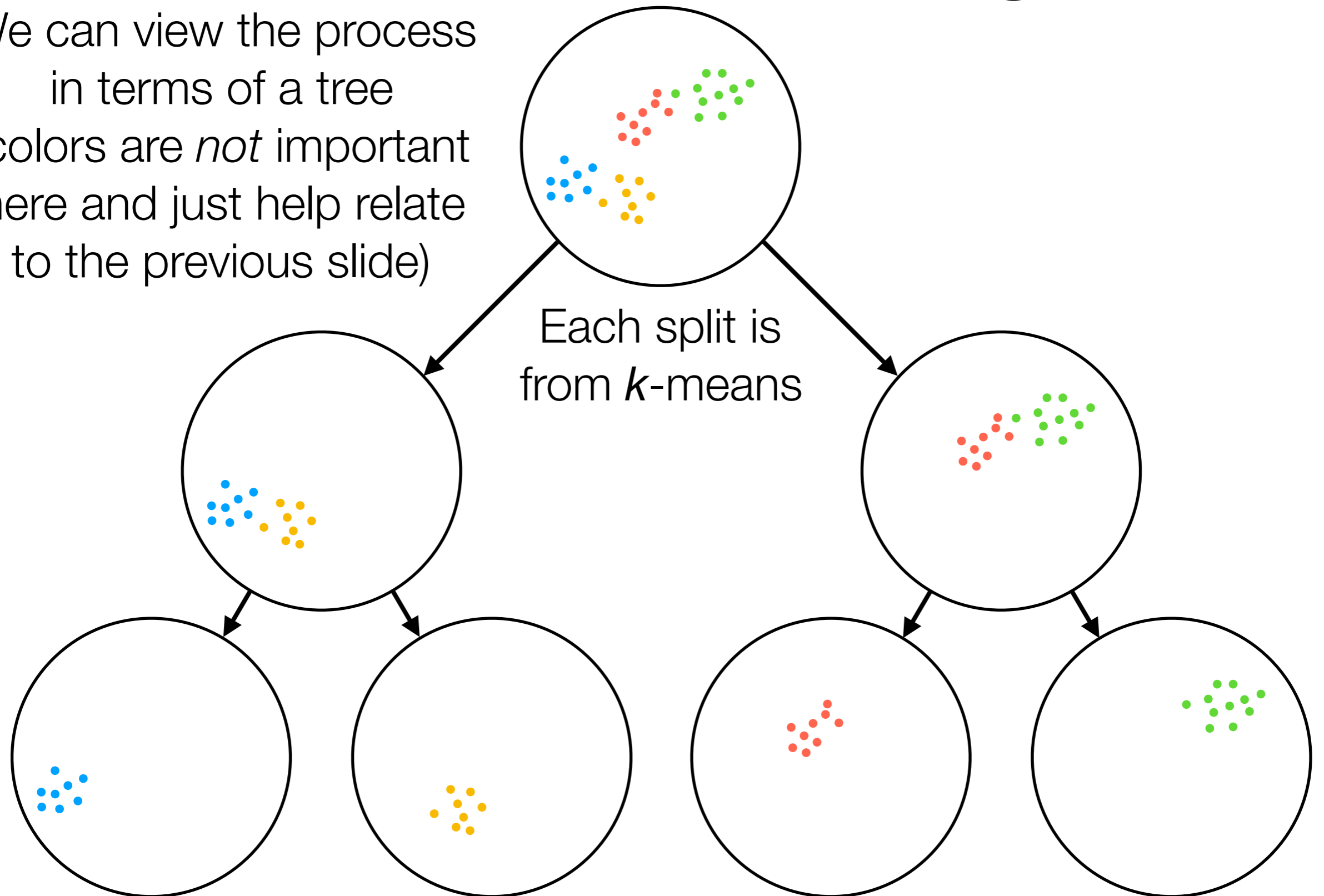


Stop splitting when some termination condition is reached

(e.g., highest cluster RSS is small enough)

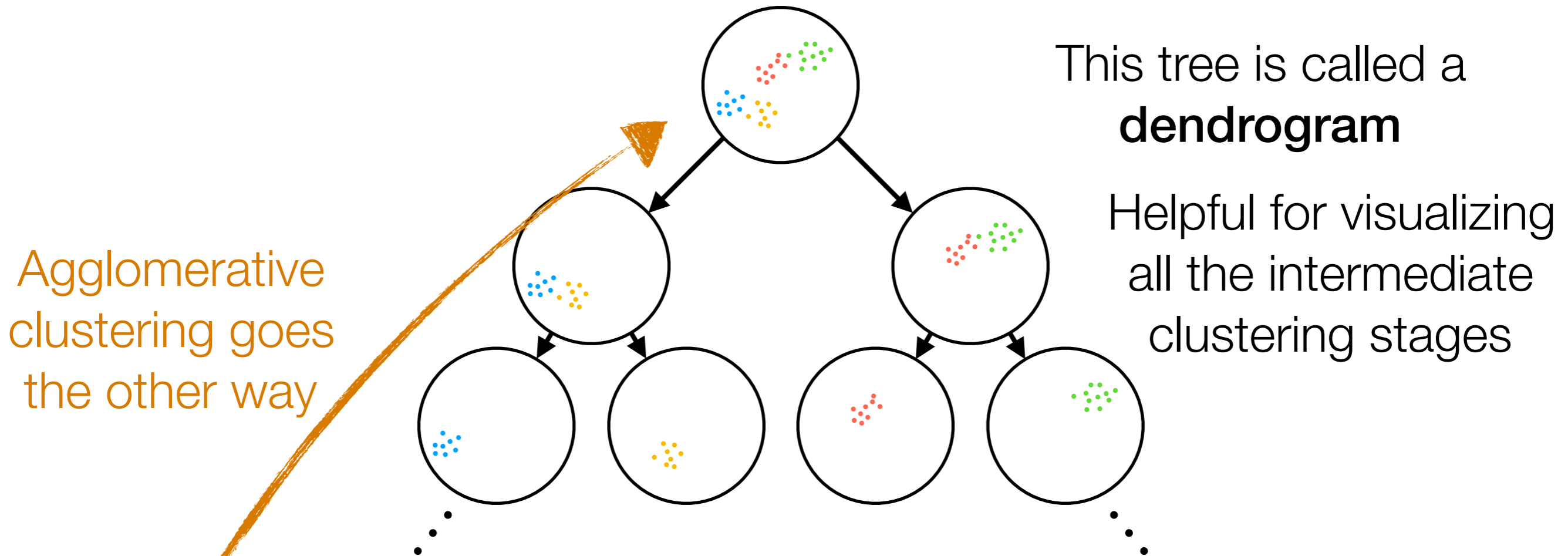
# Divisive Clustering

We can view the process  
in terms of a tree  
(colors are *not* important  
here and just help relate  
to the previous slide)



We could keep splitting until the leaves each have 1 point

# Divisive Clustering



Divisive clustering uses *global* information and keeps splitting



We could keep splitting until the leaves each have 1 point

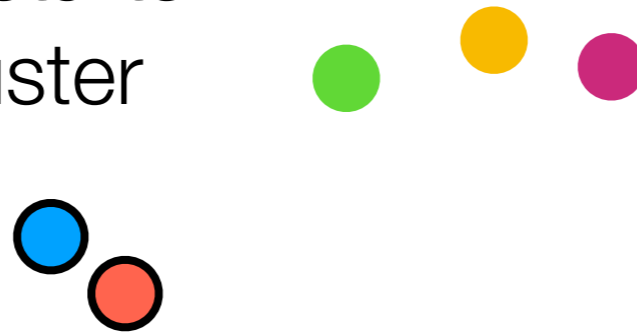
# Agglomerative Clustering

0. Every point starts  
as its own cluster



# Agglomerative Clustering

0. Every point starts  
as its own cluster



1. Find the “most similar” two clusters  
(e.g., pick pair of clusters with  
closest cluster centers)

# Agglomerative Clustering

0. Every point starts  
as its own cluster

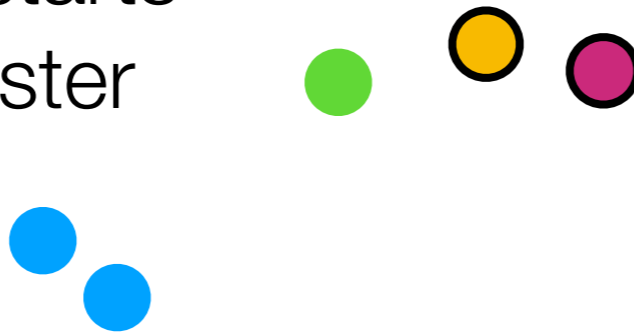


1. Find the “most similar” two clusters  
(e.g., pick pair of clusters with  
closest cluster centers)

2. Merge them

# Agglomerative Clustering

0. Every point starts  
as its own cluster

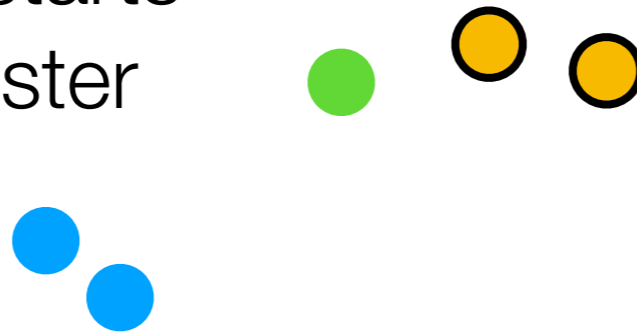


1. Find the “most similar” two clusters  
(e.g., pick pair of clusters with  
closest cluster centers)

2. Merge them

# Agglomerative Clustering

0. Every point starts  
as its own cluster



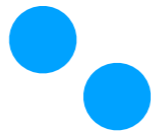
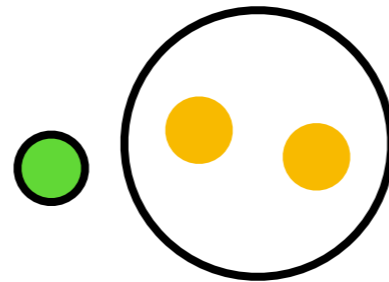
1. Find the “most similar” two clusters  
(e.g., pick pair of clusters with  
closest cluster centers)

2. Merge them



# Agglomerative Clustering

0. Every point starts as its own cluster

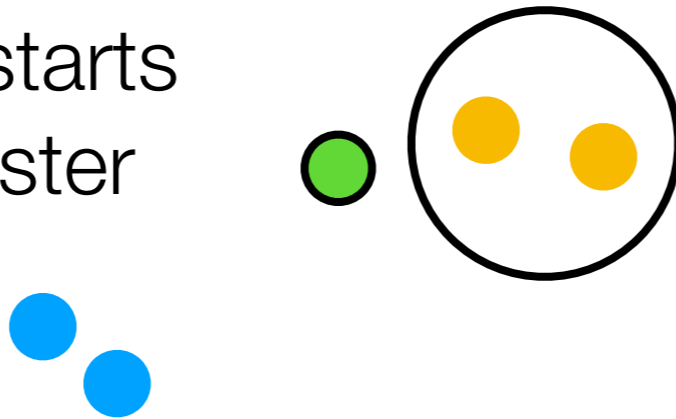


1. Find the “most similar” two clusters  
(e.g., pick pair of clusters with  
closest cluster centers)

2. Merge them

# Agglomerative Clustering

0. Every point starts as its own cluster

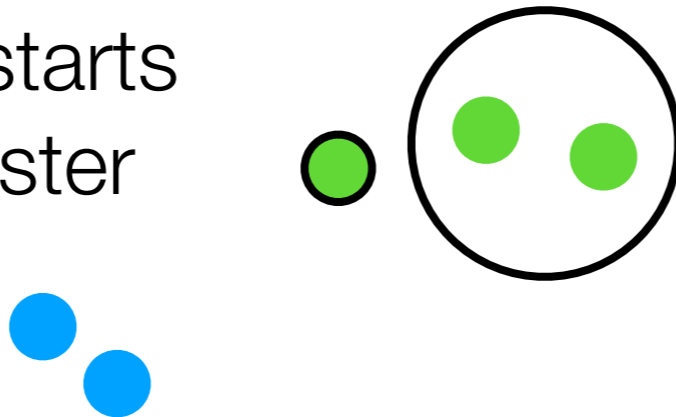


1. Find the “most similar” two clusters  
(e.g., pick pair of clusters with  
closest cluster centers)

2. Merge them

# Agglomerative Clustering

0. Every point starts as its own cluster

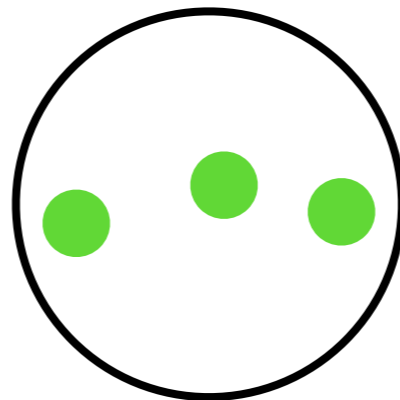
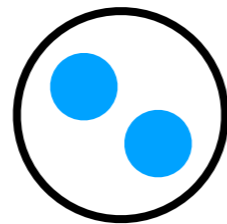


1. Find the “most similar” two clusters  
(e.g., pick pair of clusters with  
closest cluster centers)

2. Merge them

# Agglomerative Clustering

0. Every point starts as its own cluster



1. Find the “most similar” two clusters  
(e.g., pick pair of clusters with  
closest cluster centers)

2. Merge them

# Agglomerative Clustering

0. Every point starts  
as its own cluster



1. Find the “most similar” two clusters  
(e.g., pick pair of clusters with  
closest cluster centers)

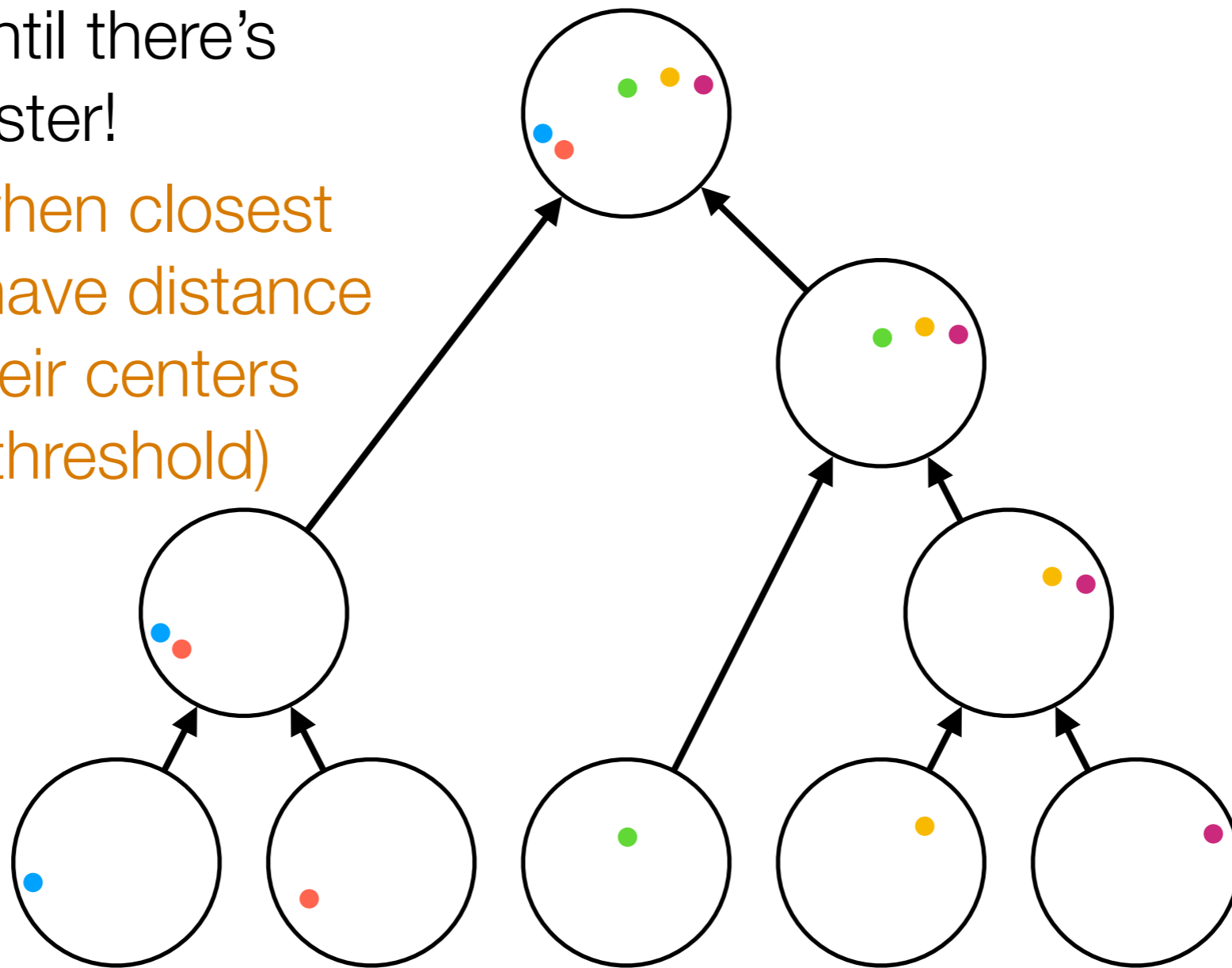
2. Merge them

# Agglomerative Clustering

Don't have to keep merging until there's 1 cluster!

(e.g., stop when closest two clusters have distance between their centers exceed a threshold)

Dendrogram



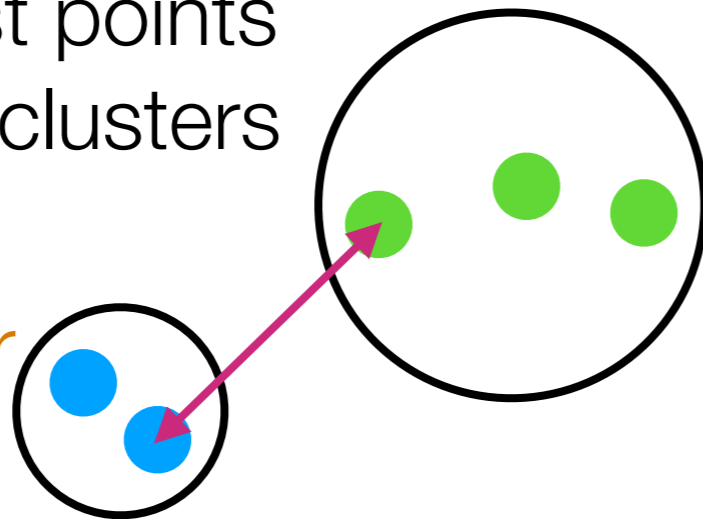
Agglomerative clustering uses *local* information and keeps merging

# Agglomerative Clustering

Some ways to define what it means for two clusters to be “close” (needed to find most similar clusters):

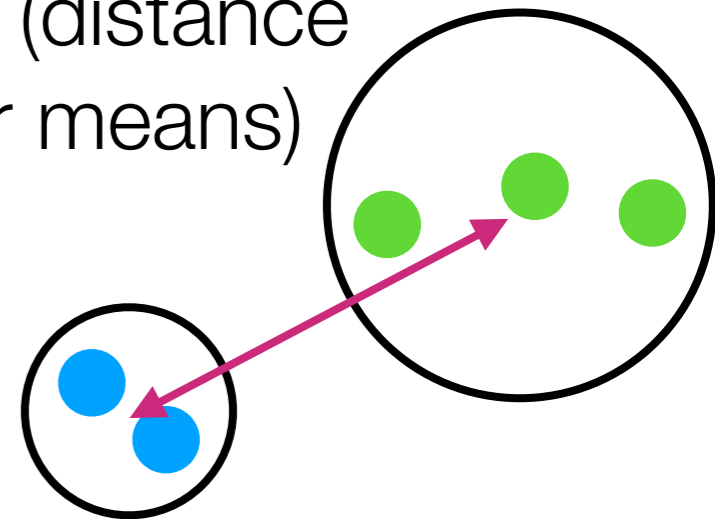
**Single linkage:** use distance between closest points across the two clusters

Can end up chaining together too many things



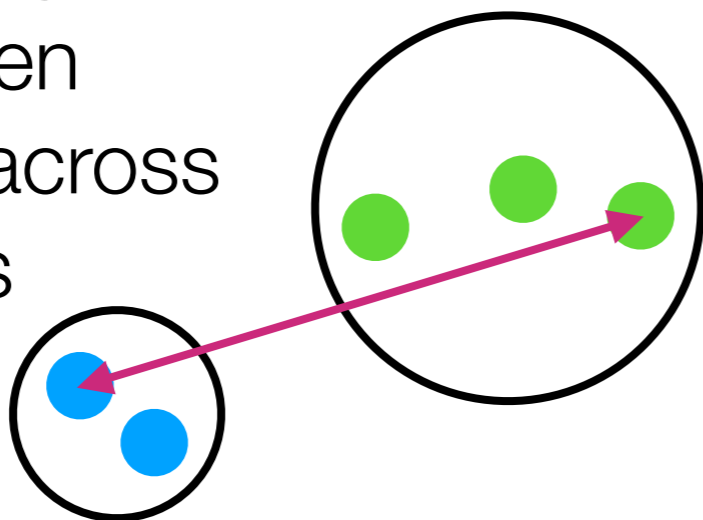
**Centroid linkage:** what we saw already (distance between cluster means)

Ignores # items in each cluster

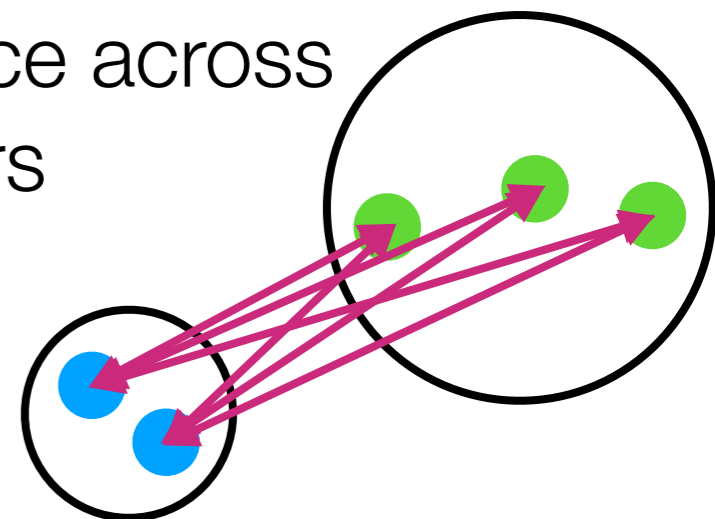


**Complete linkage:** use distance between farthest points across the two clusters

Get “crowding” behavior



**Average linkage:** use average distance across all possible pairs

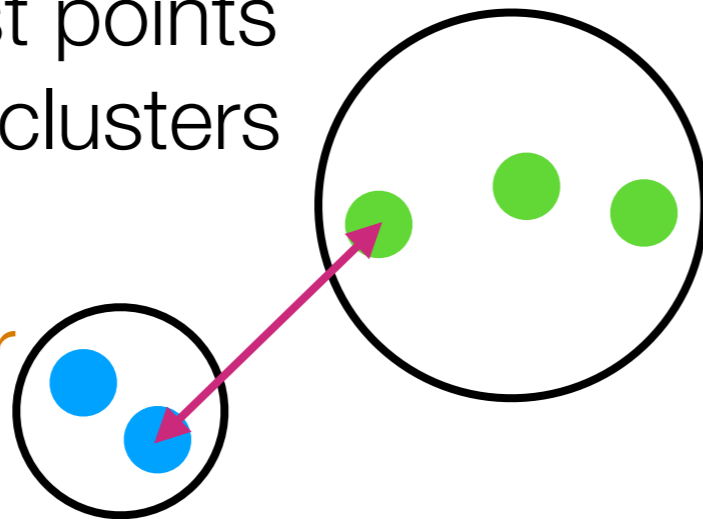


# Agglomerative Clustering

Clustering stays the same with monotonic transform of distance

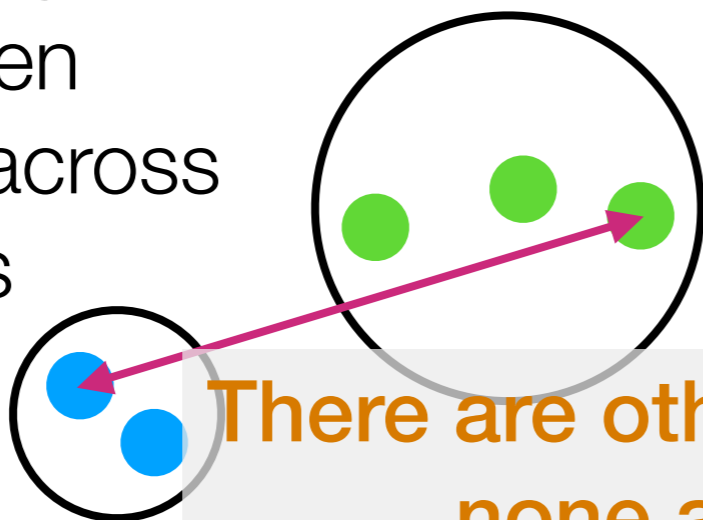
**Single linkage:** use distance between closest points across the two clusters

Can end up chaining together too many things



**Complete linkage:** use distance between farthest points across the two clusters

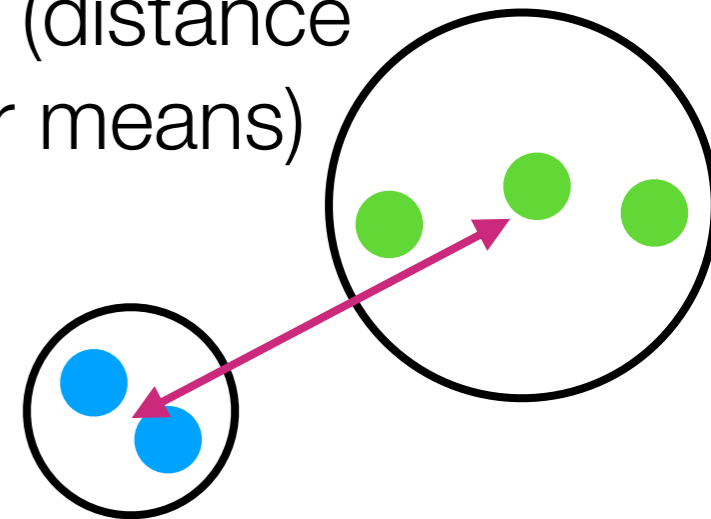
Get "crowding" behavior



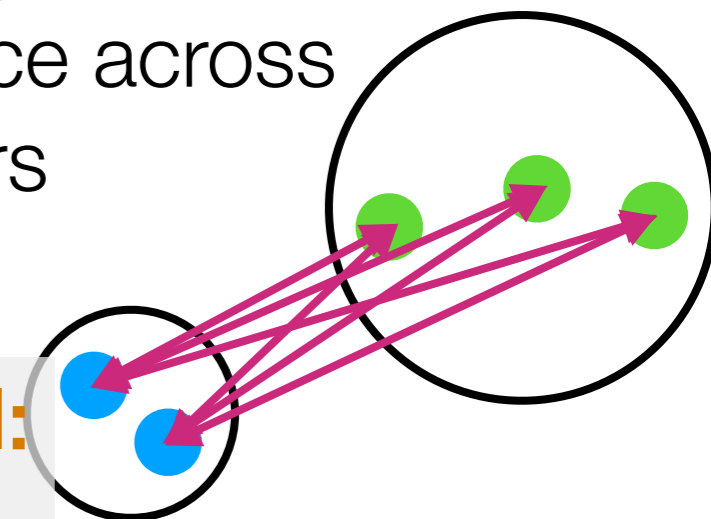
Clustering can change with monotonic transform of distance

**Centroid linkage:** what we saw already (distance between cluster means)

Ignores # items in each cluster



**Average linkage:** use average distance across all possible pairs



There are other ways as well:  
none are perfect



# Going from Similarities to Clusters

There's a whole zoo of clustering methods

Two main categories we'll talk about:

## **Generative models**

1. Pretend data generated by specific model with parameters
2. Learn the parameters ("fit model to data")
3. Use fitted model to determine cluster assignments

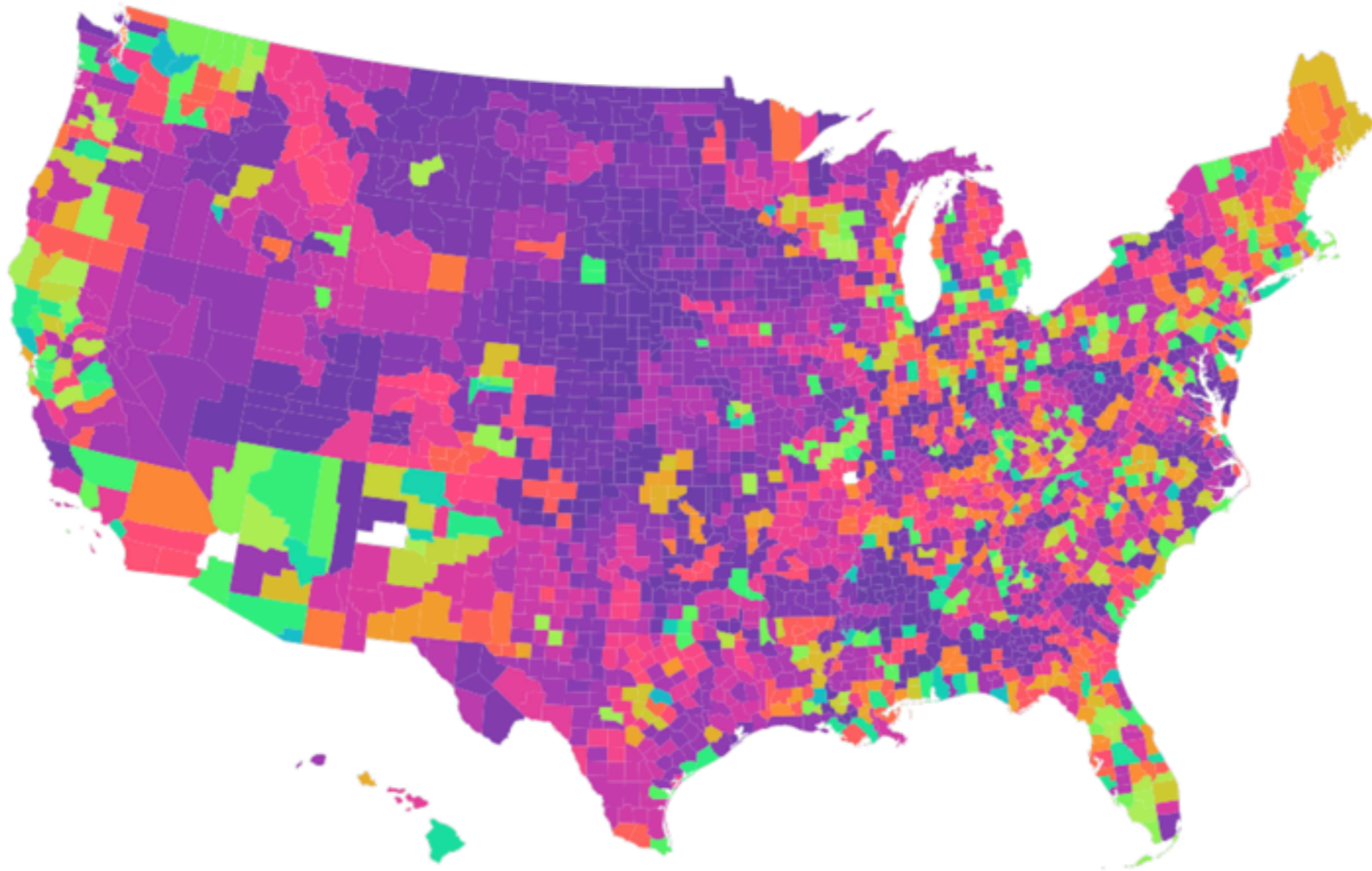
## **Hierarchical clustering**

- Top-down: Start with everything in 1 cluster and decide on how to recursively split
- Bottom-up: Start with everything in its own cluster and decide on how to iteratively merge clusters

# Example: Clustering on U.S. Counties

# Example: Clustering on U.S. Counties

(using opioid death rate data across 37 years)



No need to predict which cluster new counties should belong to, since we're already looking at all U.S. counties!

*Image source: Amanda Coston*

# Clustering

## Generative models

1. Pretend data generated by specific model with parameters
2. Learn the parameters ("fit model to data")
3. Use fitted model to determine cluster assignments

## Hierarchical clustering

Top-down: Start with everything in 1 cluster and decide on how to recursively split

Bottom-up: Start with everything in its own cluster and decide on how to iteratively merge clusters

## *Many more methods we didn't cover*

- `sklearn` has a whole bunch more (*not* close to exhaustive)
- Also: remember the recommendation system setup?
- **Co-clustering** is the problem of clustering both users and items at the same time (`sklearn` has a few methods)

# How to Choose a Clustering Method?

**In general: not easy!**

Some questions to think about:

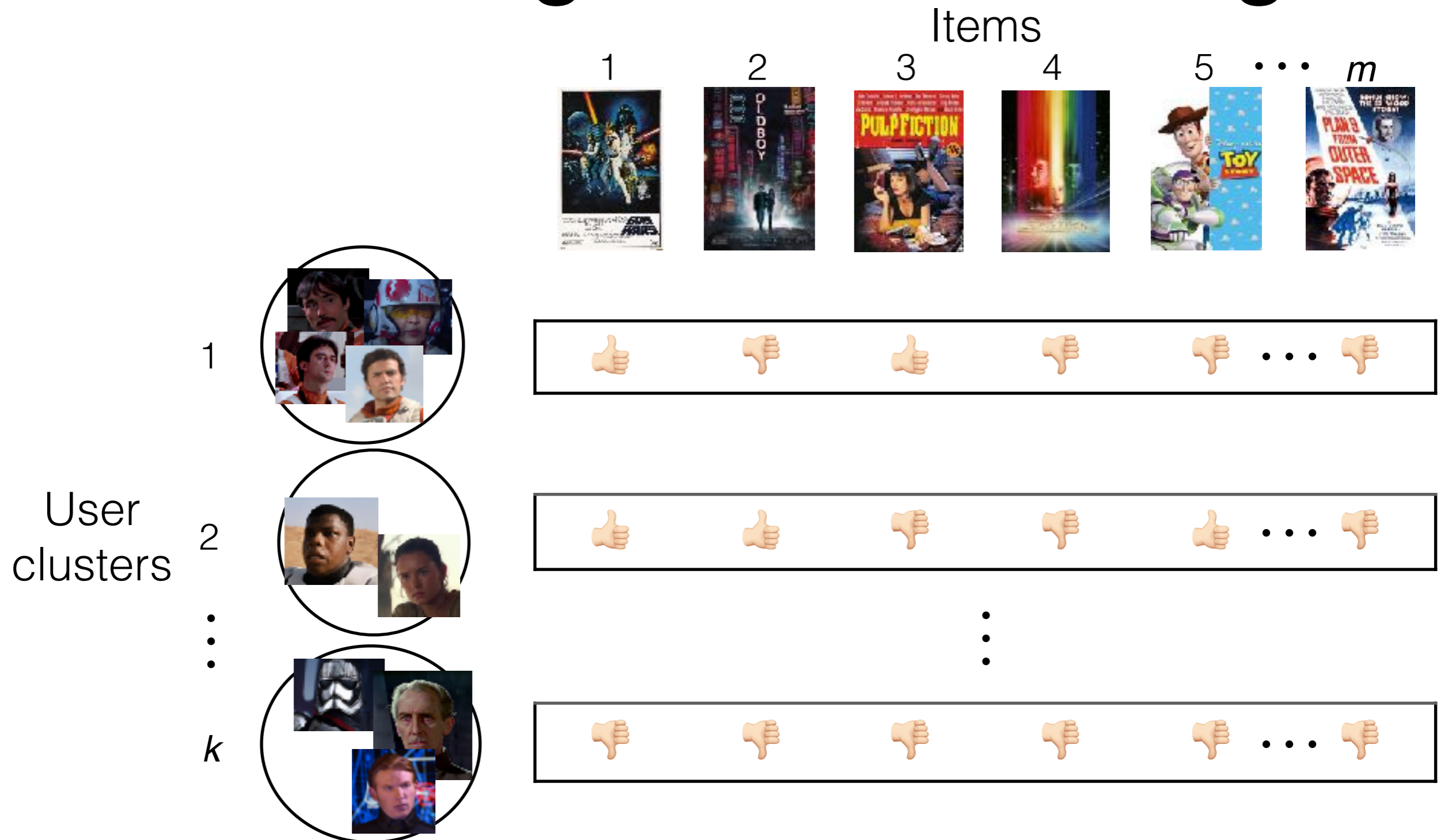
- What features to even cluster on?
- Does Euclidean distance make sense for your application, or do you use some custom distance function?
- Do you care about figuring out which cluster new points belong to?
- After you run the clustering algorithm, look at what data points ended up in the same cluster and make visualizations (e.g., histogram of various feature values)
  - Do the clusters seem interpretable to you?
  - Compare the cluster centers: do two clusters seem a bit too close and should be merged?
- Can you come up with some heuristic score function to say how good a cluster assignment is?

# Clustering

- It's possible that several clustering methods give similar results (*which is great!* — it means that there are some reasonably “stable” clusters in your data)
  - Example: *tons* of clustering methods can figure out from senate voting data who Democrats and Republicans are (of course, *without* knowing each senator's political party)
- Ultimately, *you* have to decide on which clustering method and number of clusters make sense for your data
  - Do not just blindly rely on numerical metrics (e.g., CH index)
  - Interpret the clustering results in the context of the application you are looking at

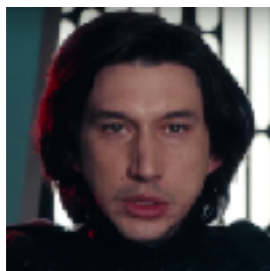
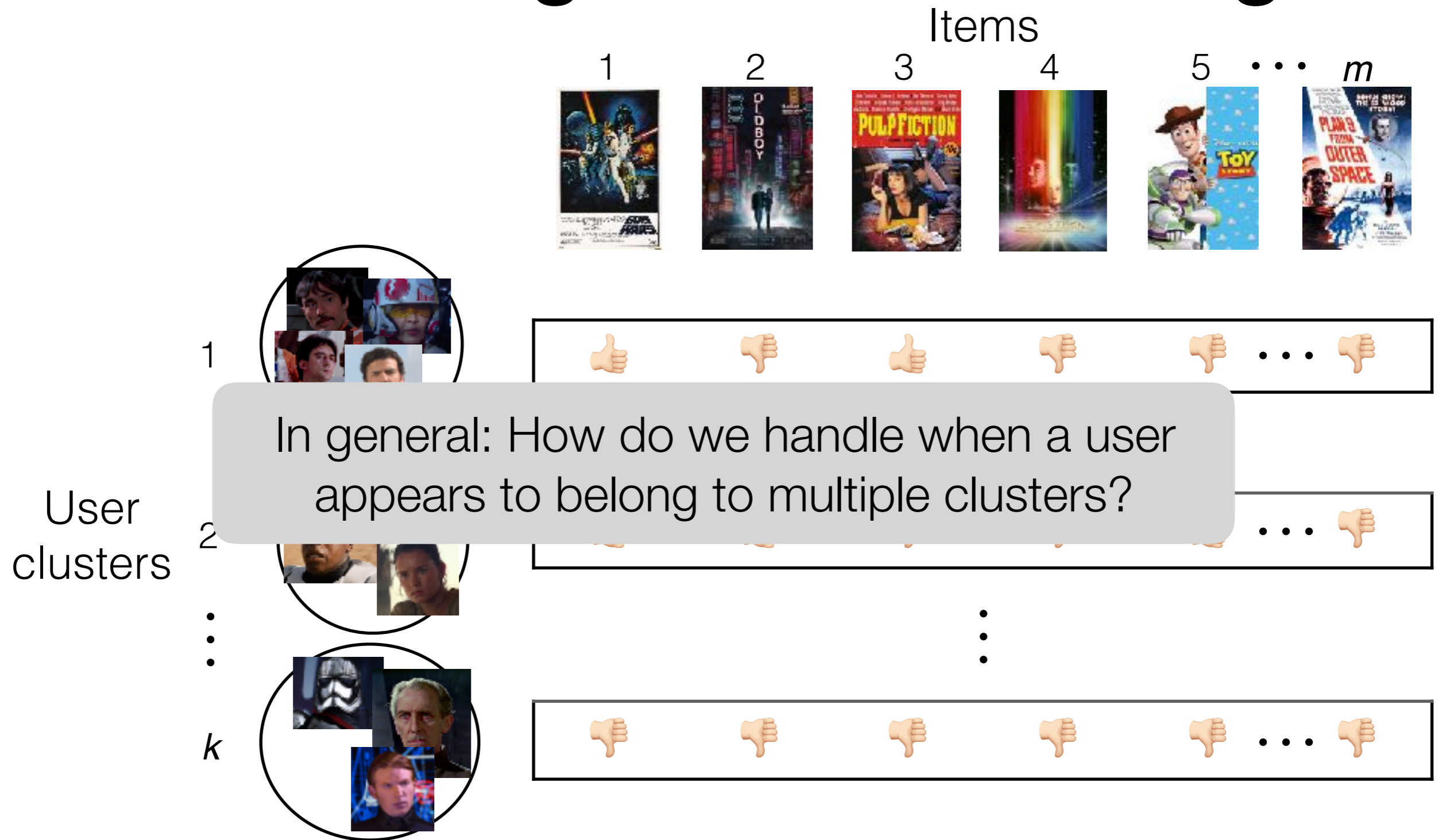
Later in the course: If you can set up a prediction task, then you can use the prediction task to help guide the clustering

# Is Clustering Structure Enough?



What if these two users shared a Netflix account (and used the same user profile)?

# Is Clustering Structure Enough?



What if these two users shared a Netflix account (and used the same user profile)?



# Topic Modeling

## Movie recommendation

Each user is part of multiple “clusters”/topics

Each cluster/topic consists of a bunch of movies  
(example clusters: “sci-fi epics”, “cheesy rom-coms”)

## Text

Each document is part of multiple topics

Each topic consists of a bunch of regularly co-occurring words  
(example topics: “sports”, “medicine”, “movies”, “finance”)

## Health care

Each patient’s health records explained by multiple “topics”

Each topic consists of co-occurring “events”  
(example topics: “heart condition”, “severe pancreatitis”)

# Topic Modeling

## Movie recommendation

Each user is part of multiple “clusters”/topics

Each cluster/topic consists of a bunch of movies  
(example clusters: “sci-fi epics”, “chessy rom-coms”)

In all of these examples:

- Each data point (a feature vector) is part of multiple topics

Each topic corresponds to specific feature values in the feature vector likely appearing in documents (example topics: “science fiction”, “chessy rom-coms”)

## Health care

Each patient’s health records explained by multiple “topics”

Each topic consists of co-occurring “events”  
(example topics: “heart condition”, “severe pancreatitis”)

# Latent Dirichlet Allocation (LDA)

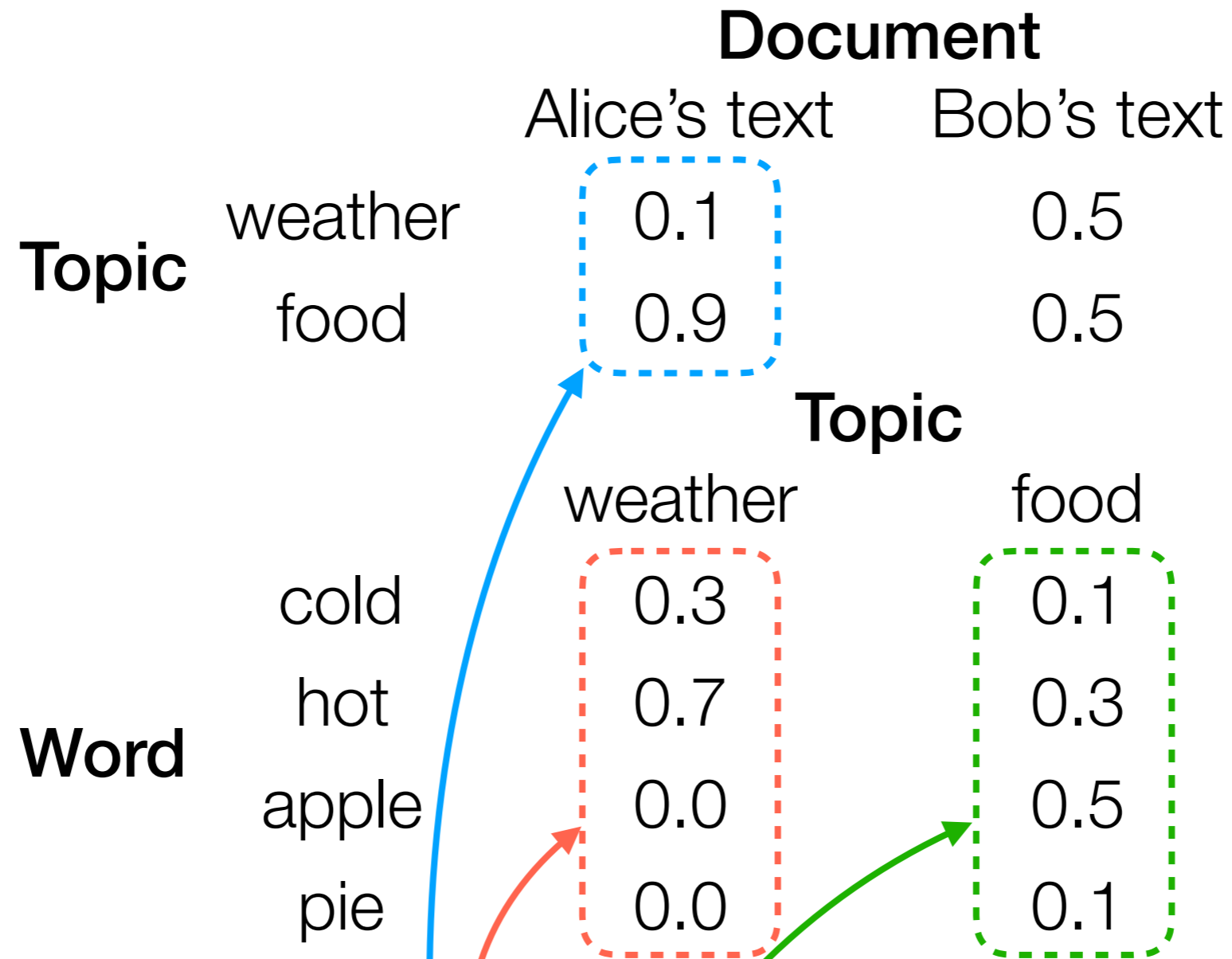
- Easy to describe in terms of text (but works for not just text)
- Input: “document-word” matrix, and pre-specified # topics  $k$

		Word			
		1	2	...	$d$
Document	1				
	2				
	⋮				
	$n$				

$i$ -th row,  $j$ -th column: # times word  $j$  appears in doc  $i$

- Output: what the  $k$  topics are (details on this shortly)

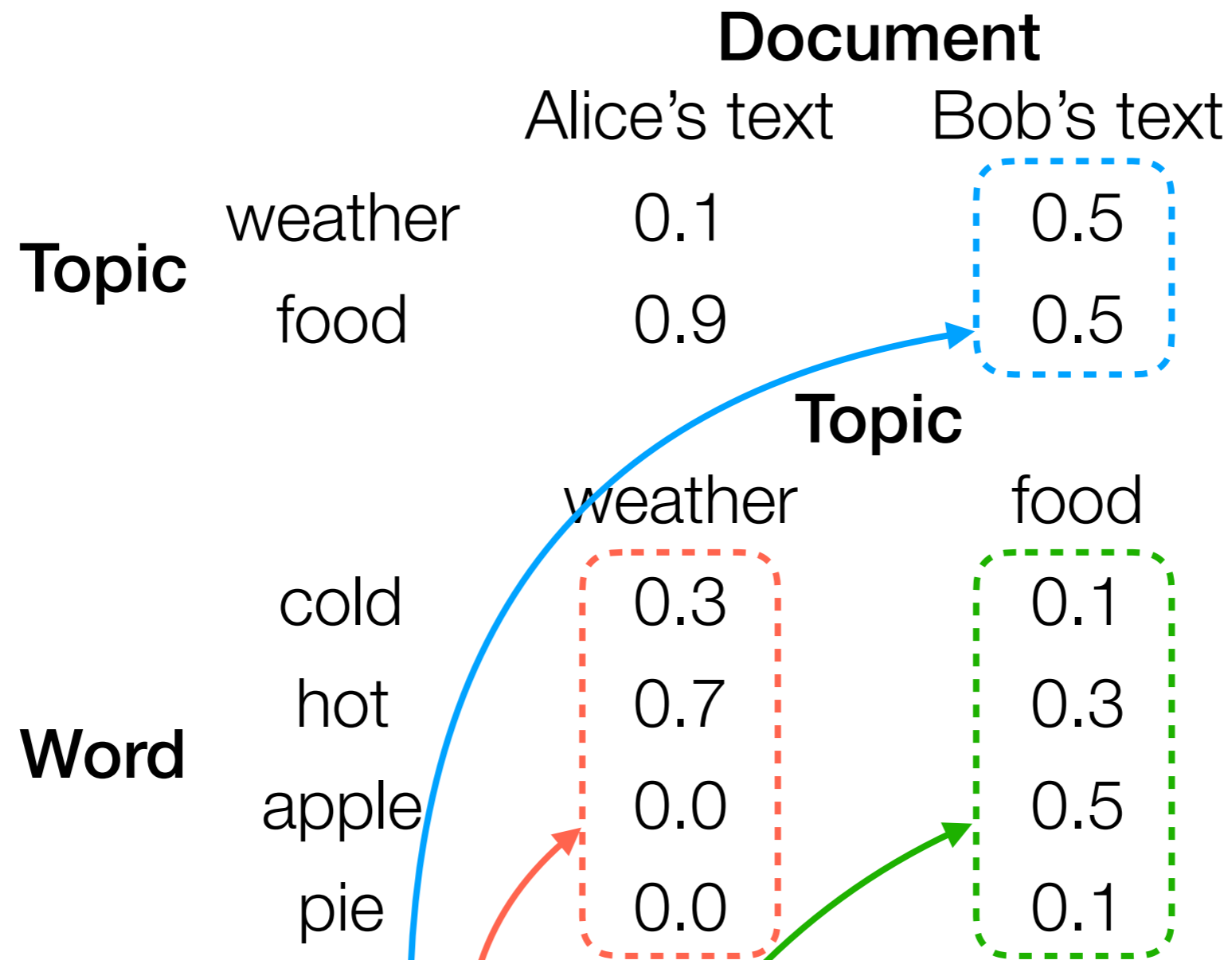
# LDA Example



Each word in Alice's text is generated by:

1. Flip 2-sided coin for Alice
2. If weather: flip 4-sided coin for weather  
If food: flip 4-sided coin for food

# LDA Example



Each word in Bob's text is generated by:

1. Flip 2-sided coin for Bob
2. If weather: flip 4-sided coin for weather  
If food: flip 4-sided coin for food

# LDA Example

		Document	
		Alice's text	Bob's text
Topic	weather	0.1	0.5
	food	0.9	0.5

		Topic	
		weather	food
Word	cold	0.3	0.1
	hot	0.7	0.3
	apple	0.0	0.5
	pie	0.0	0.1

Each word in doc.  $i$  is generated by:

1. Flip 2-sided coin for doc.  $i$
2. If weather: flip 4-sided coin for weather  
If food: flip 4-sided coin for food

# LDA Example

		Document	
		Alice's text	Bob's text
Topic	weather	0.1	0.5
	food	0.9	0.5

		Topic	
		weather	food
Word	cold	0.3	0.1
	hot	0.7	0.3
	apple	0.0	0.5
	pie	0.0	0.1

Each word in doc.  $i$  is generated by:

1. Flip 2-sided coin for doc.  $i$
2. If weather: flip 4-sided coin for weather  
If food: flip 4-sided coin for food

“Learning the topics” means figuring out these 4-sided coin probabilities

# LDA



LDA models each word in document  $i$  to be generated as:

- Randomly choose a topic  $Z$  (use topic distribution for doc  $i$ )
- Randomly choose a word (use word distribution for topic  $Z$ )



# LDA

- Easy to describe in terms of text (but works for not just text)
- Input: “document-word” matrix, and pre-specified # topics  $k$

		Word			
		1	2	...	$d$
Document	1				
	2				
	⋮				
	$n$				

$i$ -th row,  $j$ -th column: # times word  $j$  appears in doc  $i$

- Output: the  $k$  topics' distribution of words

# LDA

Demo

# How to Choose Number of Topics $k$ ?

Bayesian nonparametric variant of LDA:  
Hierarchical Dirichlet Process (HDP)

(similar to how we went from GMM to DP-GMM)

Something like CH index is also possible:

For a specific topic, look at the  $m$  most probable words (“top words”)

**Coherence (within cluster/topic variability):**

$$\sum_{\substack{\text{top words } v, w \\ \text{that are not the same}}} \log \frac{(\# \text{ documents with at least one appearance of } v \text{ and } w) + \epsilon}{\# \text{ documents with at least one appearance of } w}$$

choose something small like 0.01

**Inter-topic similarity (between cluster/topic variability):**

Can average each of these across the topics

Count # top words that do not appear in any of the other topics'  $m$  top words  
(number of “unique words”)

# Topic Modeling

- There are actually *many* topic models, not just LDA & HDP
  - Correlated topic models, Pachinko allocation, biterm topic models, anchor word topic models, ...
- Dynamic topic models: tracks how topics change *over time*
  - This sort of idea could be used to figure out how user tastes change over time in a recommendation system
    - Could try to see if there are existing patterns for how certain topics become really popular