Search @twitter

Michael Busch
@michibusch
michael@twitter.com
buschmi@apache.org

Search @twitter

Agenda

- Introduction

- Search Architecture

- Inverted Index

- Ranking

Introduction

Introduction

Twitter has more than 255 million monthly active users.

Introduction

500 million tweets are sent per day.

Introduction

More than 300 billion tweets have been sent since company founding in 2006.

Introduction

Tweets-per-second world record:
33,388 TPS.

Introduction

More than 2 billion search queries per day.

Introduction

2008
2009
2010
2011
2012
2013
2014

Introduction

2008

2009

2010

2011

2012

2013

2014

Apache SOFTWARE FOUNDATION

Lucene™

twitter ipad Search

Results for **ipad** 0.30 seconds

darthvader: That's the problem with clones, once one wants an **iPad**, they all do.
1 day ago from *Twitterrific* · Reply · View Tweet
600+ recent retweets

hotdogsladies: The people who love the **iPad** they haven't used yet actually have a lot in common with the people who hate the **iPad** they haven't used yet.
7 days ago from *Birdhouse* · Reply · View Tweet
100+ recent retweets

stephenfry: I've written about the **iPad** for Time Magazine: http://stephenfry.com.
about 16 hours ago from *Twitterrific* · Reply · View Tweet
200+ recent retweets

mcteoh: is a VIP on Original Gangstaz on my iPhone! Click the link to join my gang http://bit.ly/originalgz (expand) #iphone #ipod #**ipad** #OG
less than 10 seconds ago from *MGTwitterEngine* · Reply · View Tweet

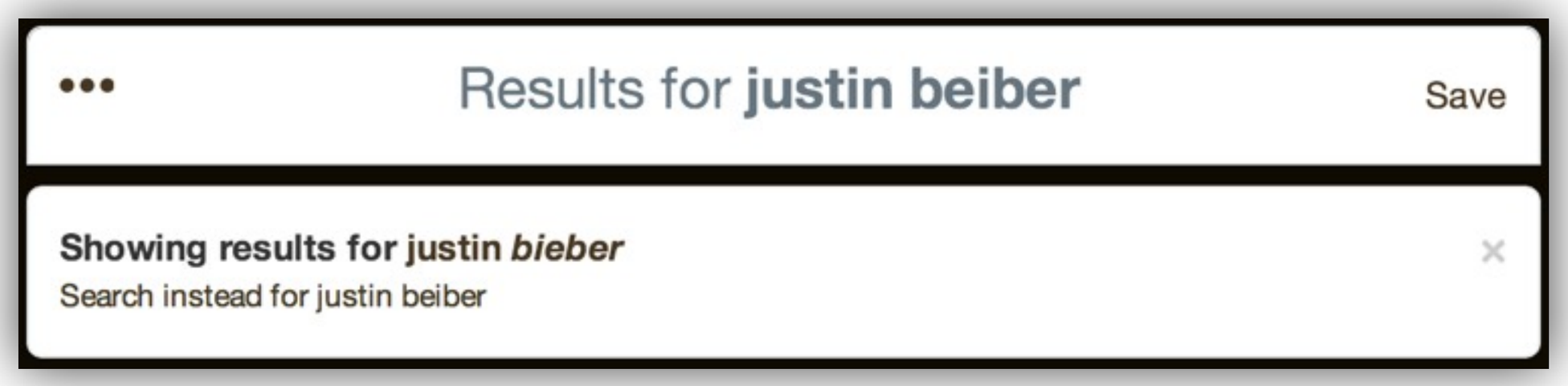
applenws: #apple Got **iPad**-itis? News tech writers start to feel an itch for device - Dallas Morning News http://bit.ly/9cow6J (expand)
less than 20 seconds ago from *twitterfeed* · Reply · View Tweet

Introduction

2008
2009
2010
2011
2012
2013
2014

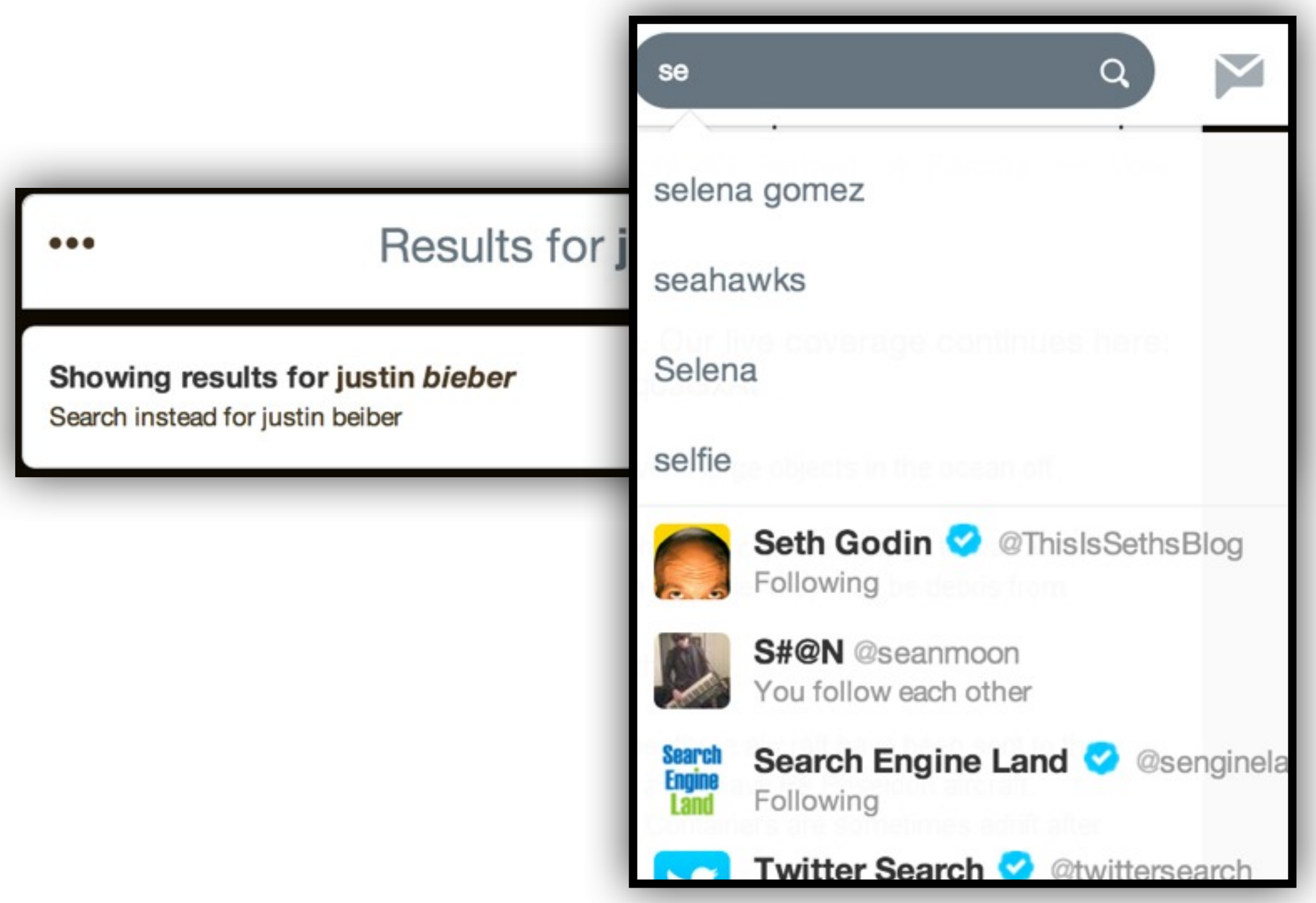
Introduction

2008
2009
2010
2011
2012
2013
2014



Introduction

2008
2009
2010
2011
2012
2013
2014



Introduction

2008

2009

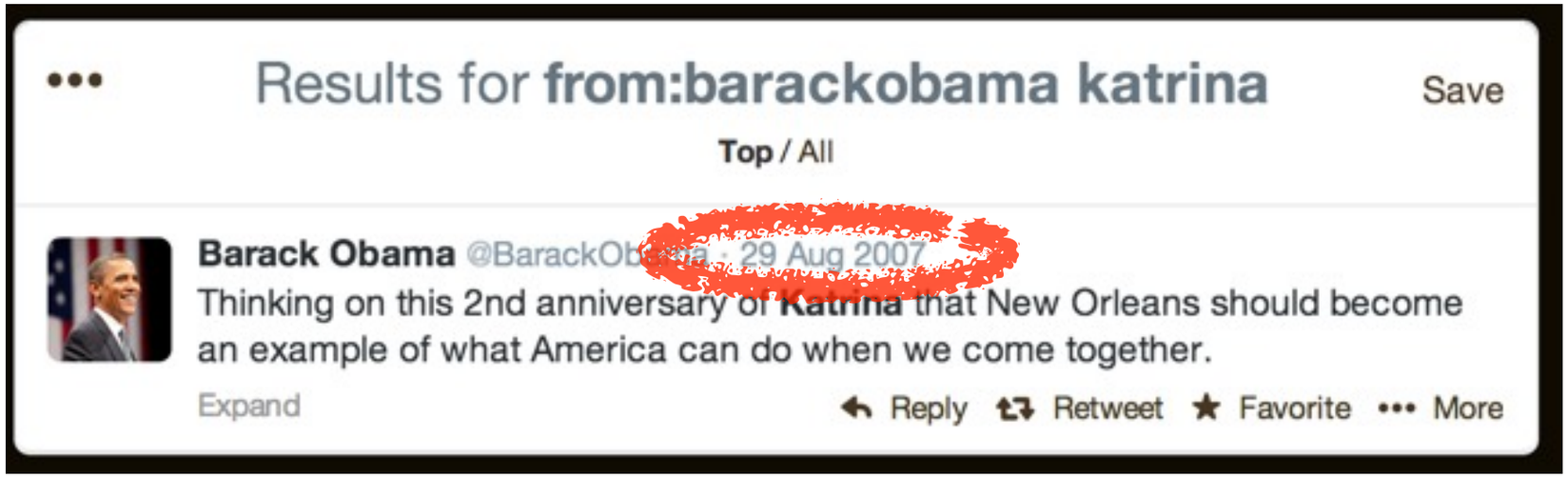
2010

2011

2012

2013

2014



Introduction

2008

2009

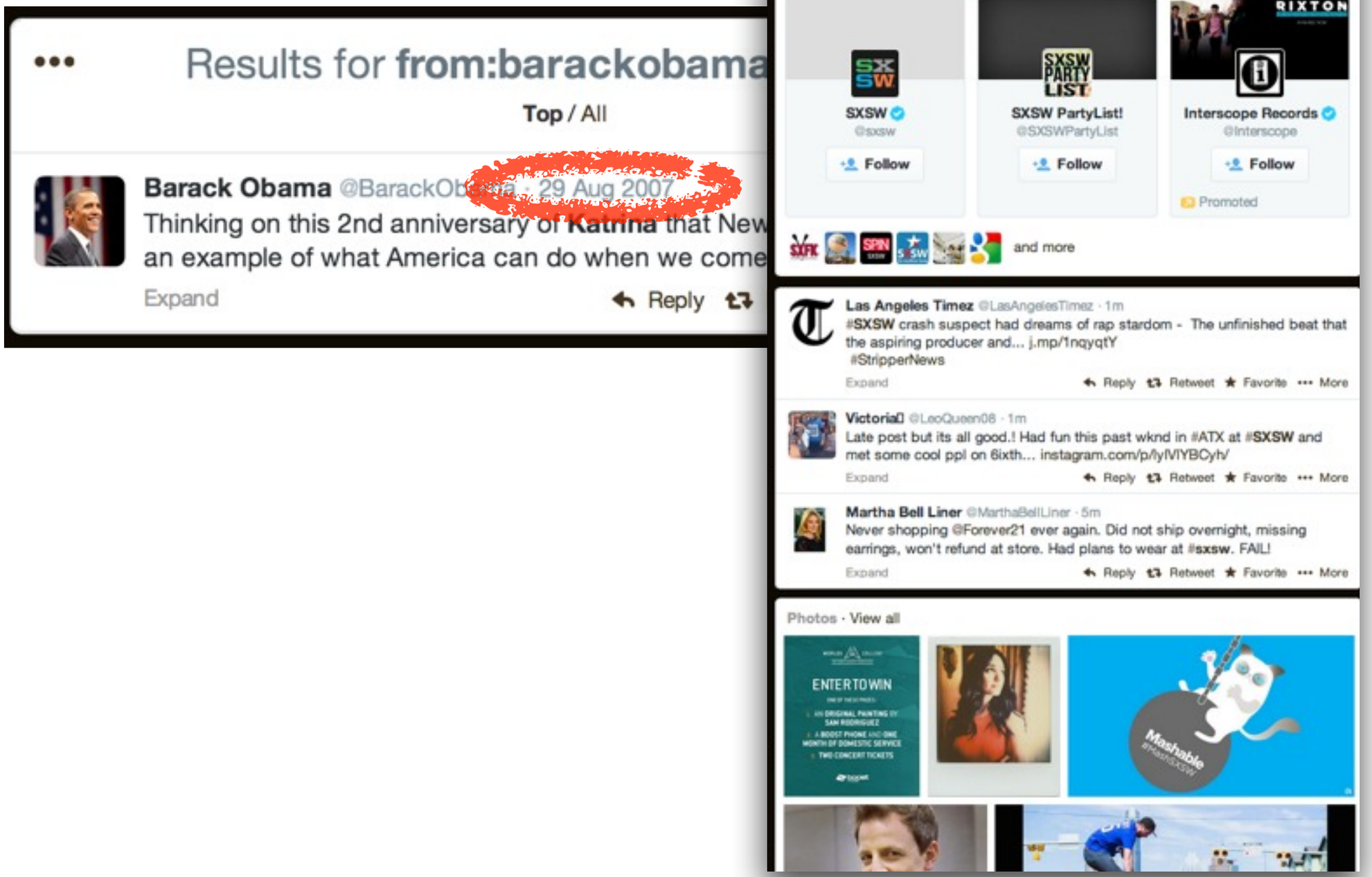
2010

2011

2012

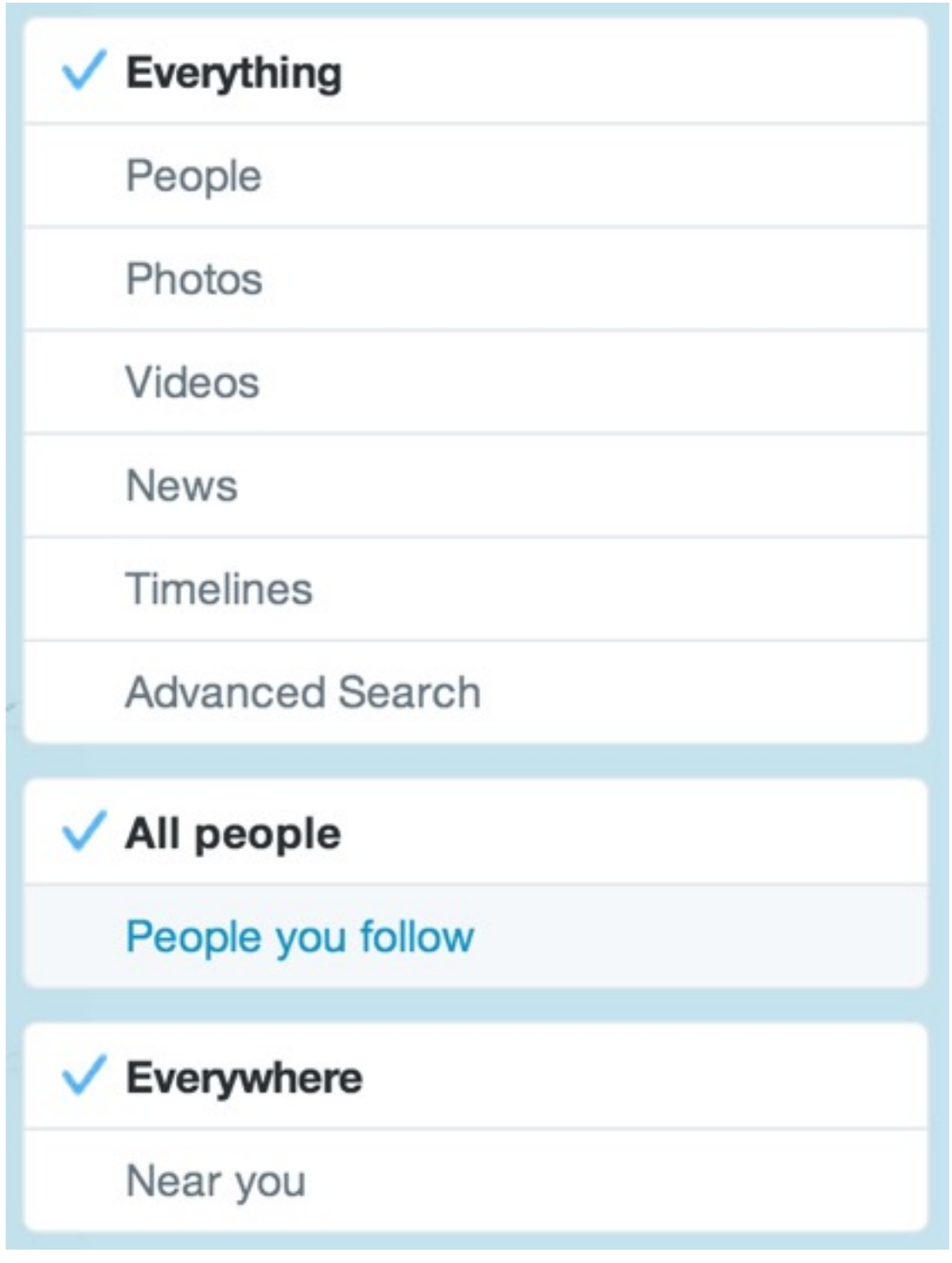
2013

2014



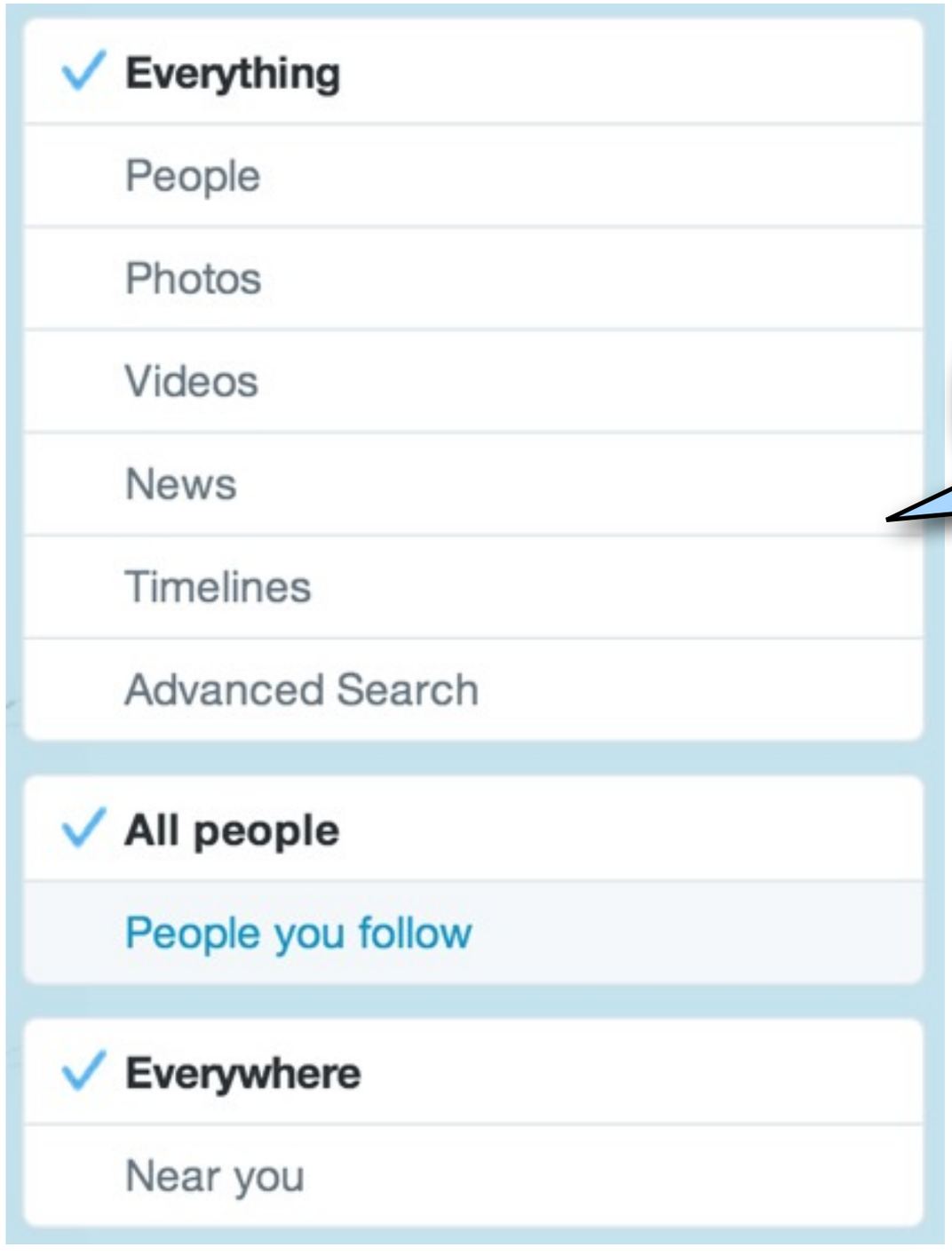
Introduction

2008
2009
2010
2011
2012
2013
2014



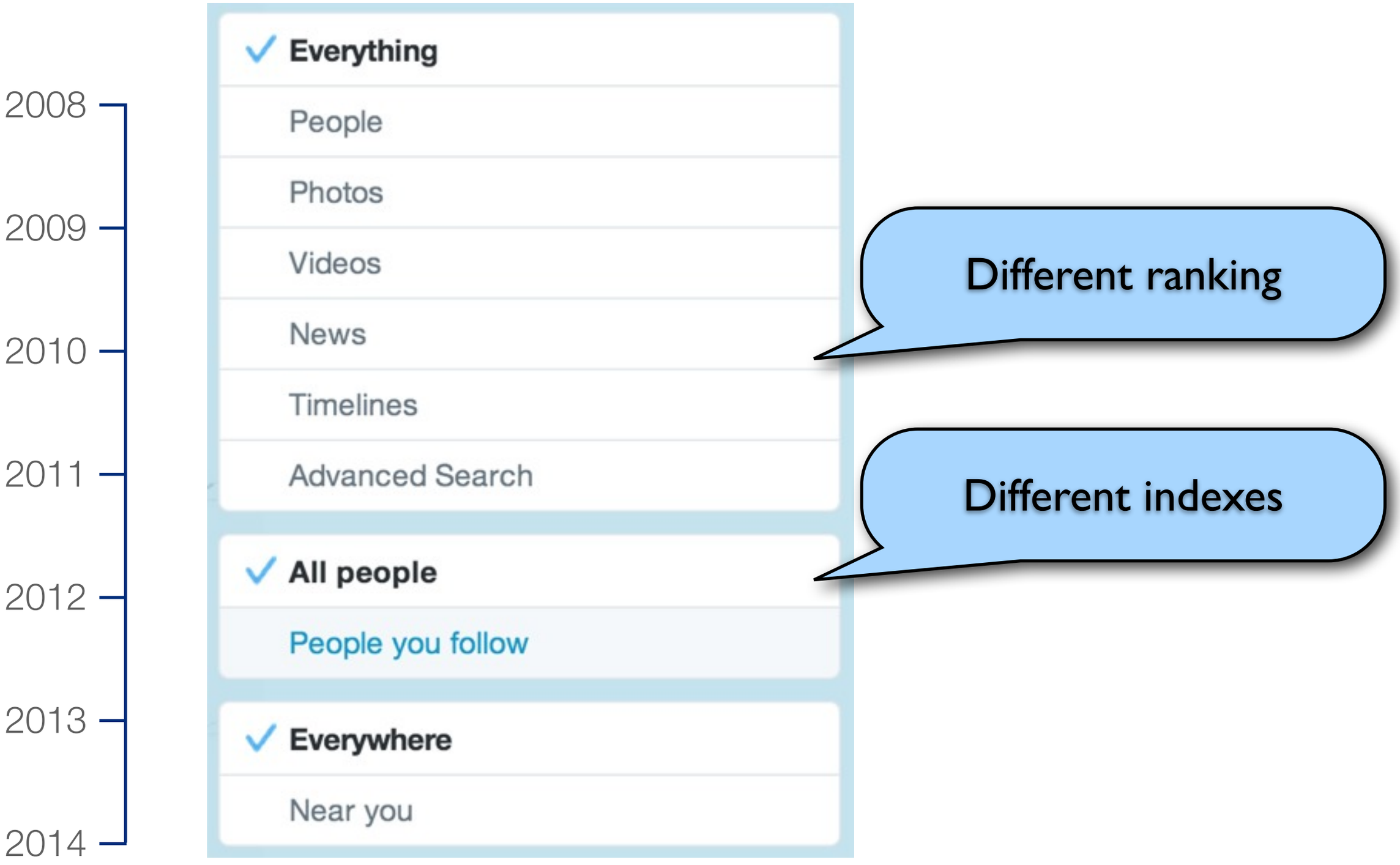
Introduction

2008
2009
2010
2011
2012
2013
2014

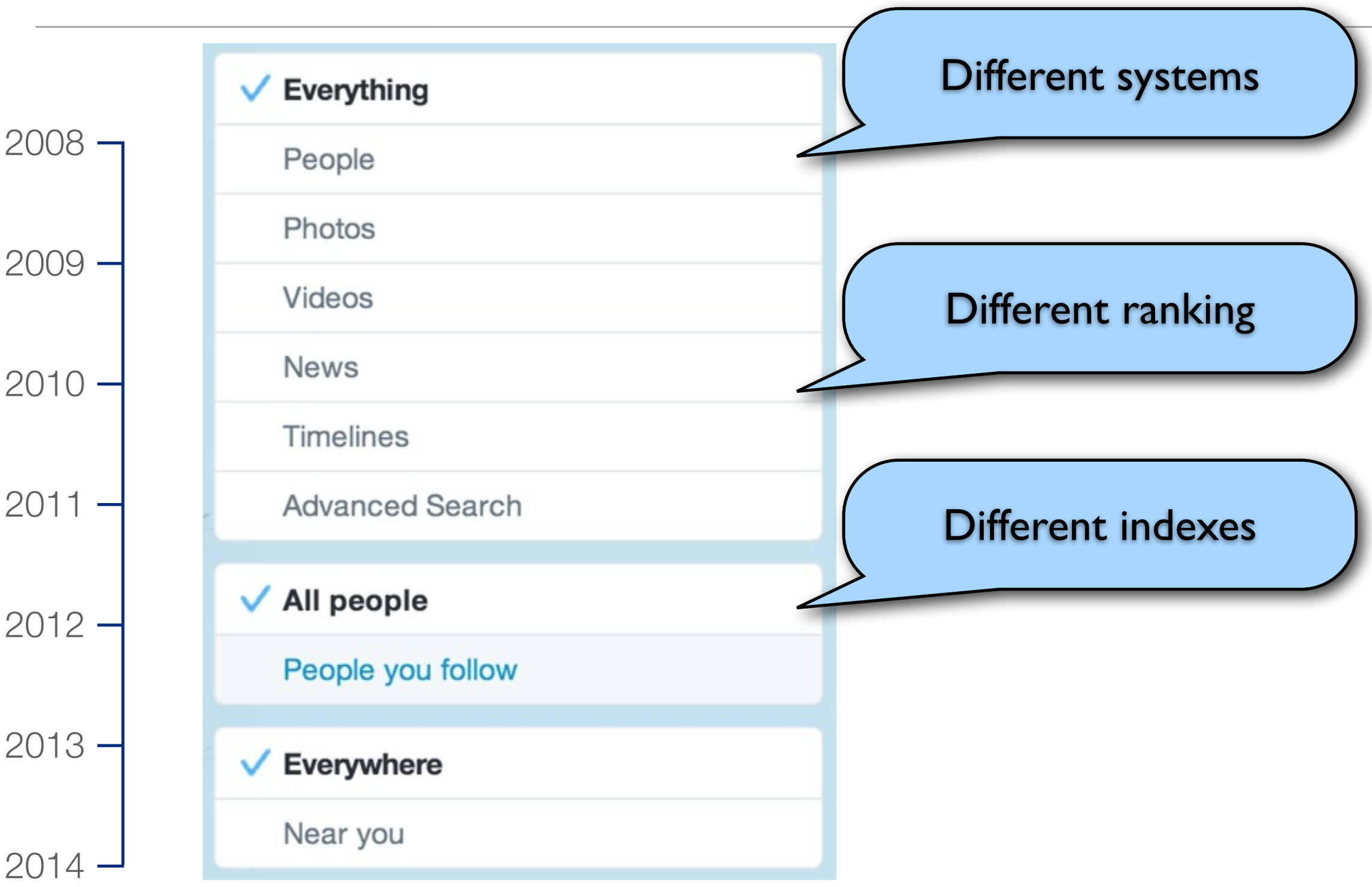


Different ranking

Introduction



Introduction



Introduction

2008 Twitter acquires Summize (MySQL-based RT search engine)

2009

2010 Modified Lucene (Earlybird) ships and replaces MySQL indexes

2011 New Earlybird features: image/video search; index compression; efficient relevance search in time-sorted index

2012

2013 Type-ahead; spelling correction; Tweet archive search on SSD with vanilla Lucene

2014 New RT posting list format that supports arbitrary document lengths, but keeps performance optimizations for tweets

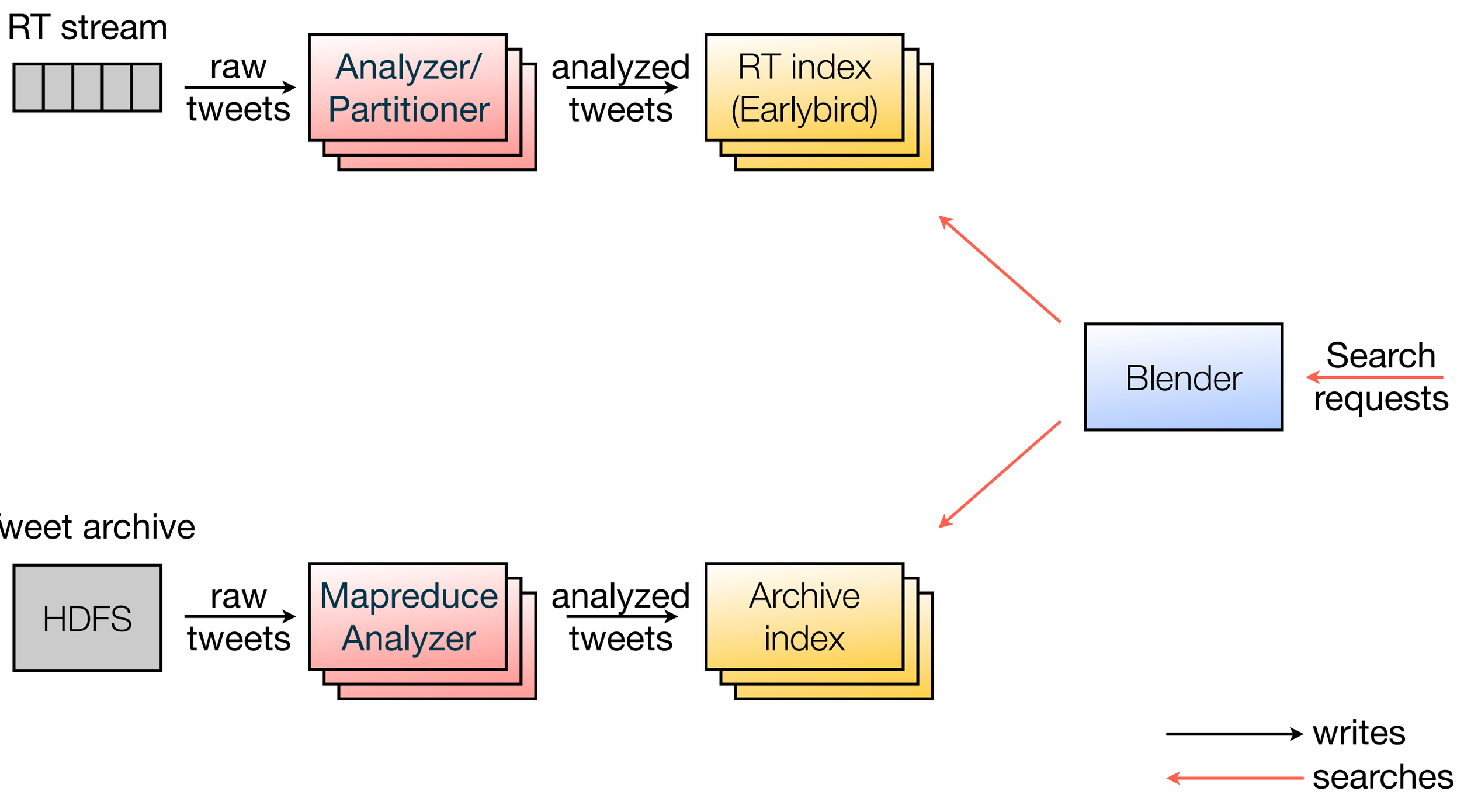
Realtime Search @twitter

Agenda

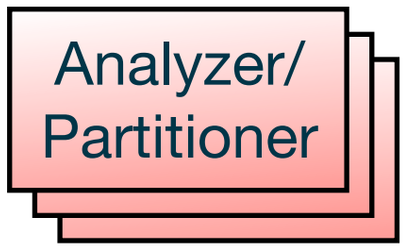
- Introduction
- Search Architecture
- Inverted Index
- Ranking

Search Architecture

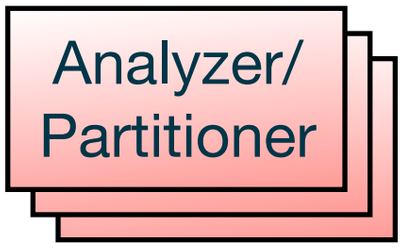
Search Architecture



Search Architecture

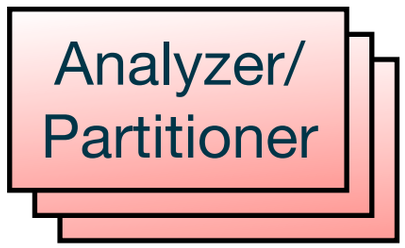


Search Architecture

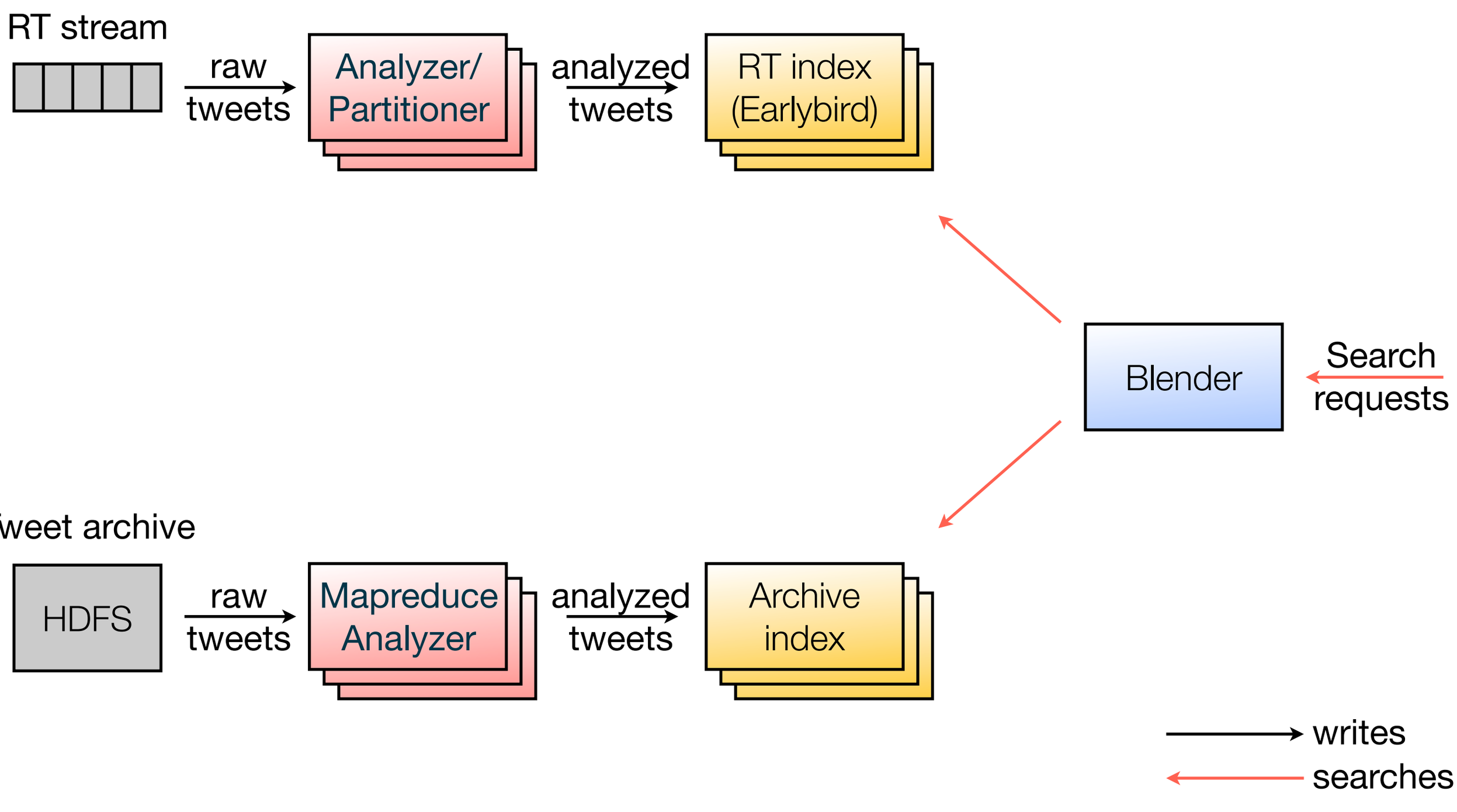


- Pre-processes Tweets for indexing
- Analyzing (tokenization/normalization) of text
- Geo-coding, URL expansion, etc.
- Hash partitioning

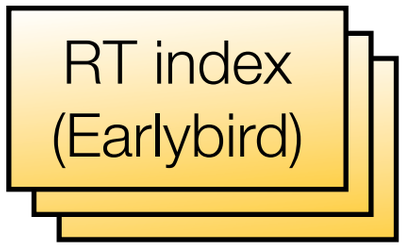
Search Architecture



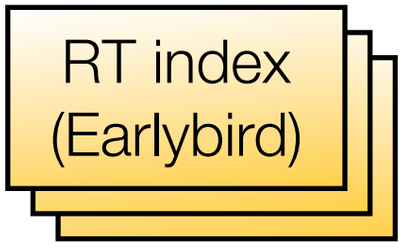
Search Architecture



Search Architecture



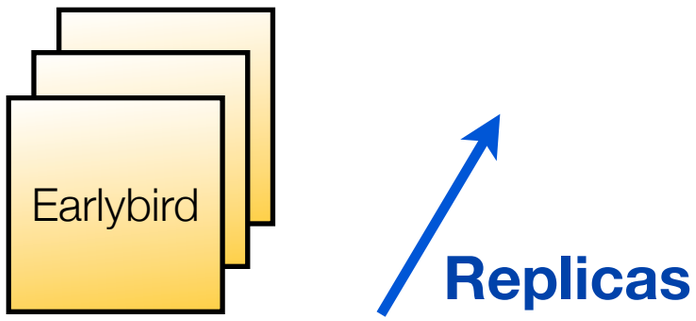
Search Architecture



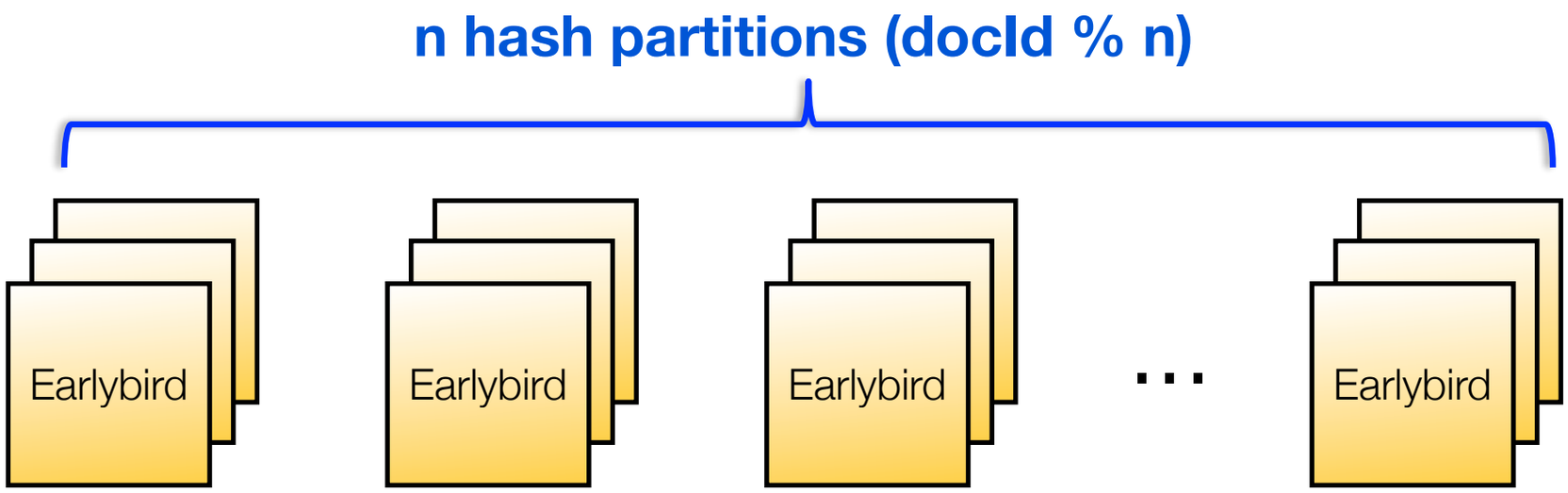
- Modified Lucene index implementation optimized for realtime search
- IndexWriter buffer is searchable (no need to flush to allow searching)
- In-memory
- Hash-partitioned, static layout

Cluster layout

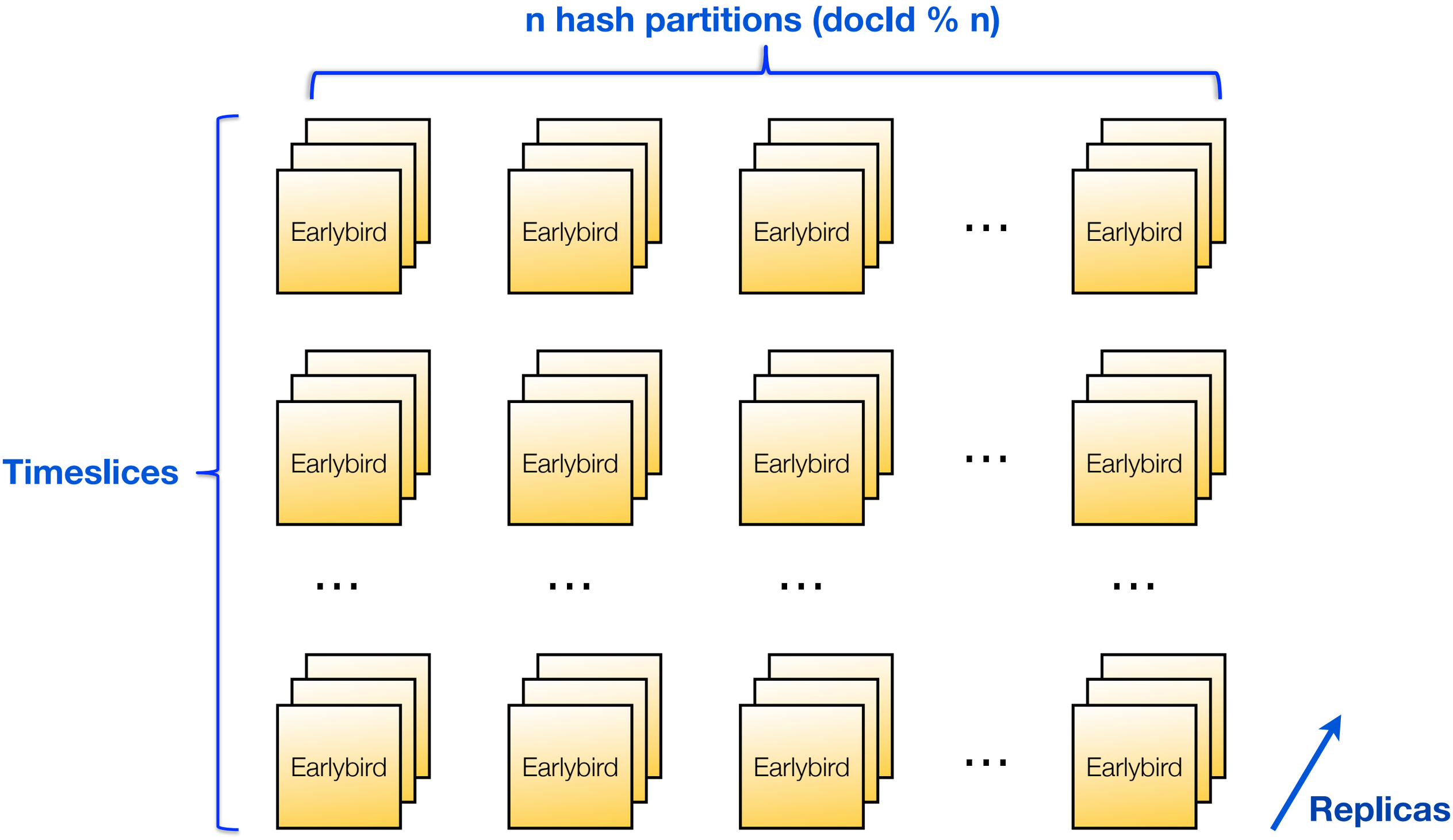
Cluster layout



Cluster layout

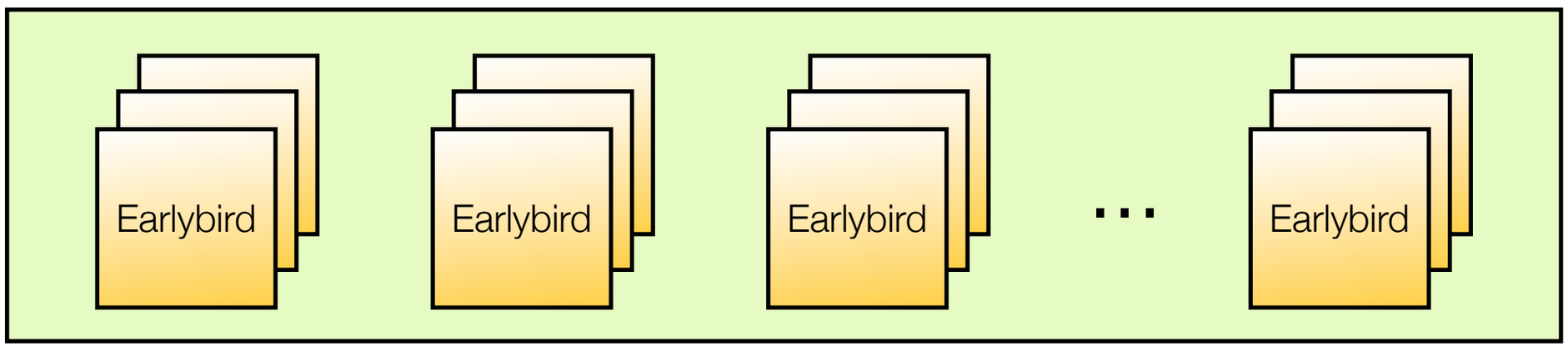


Cluster layout

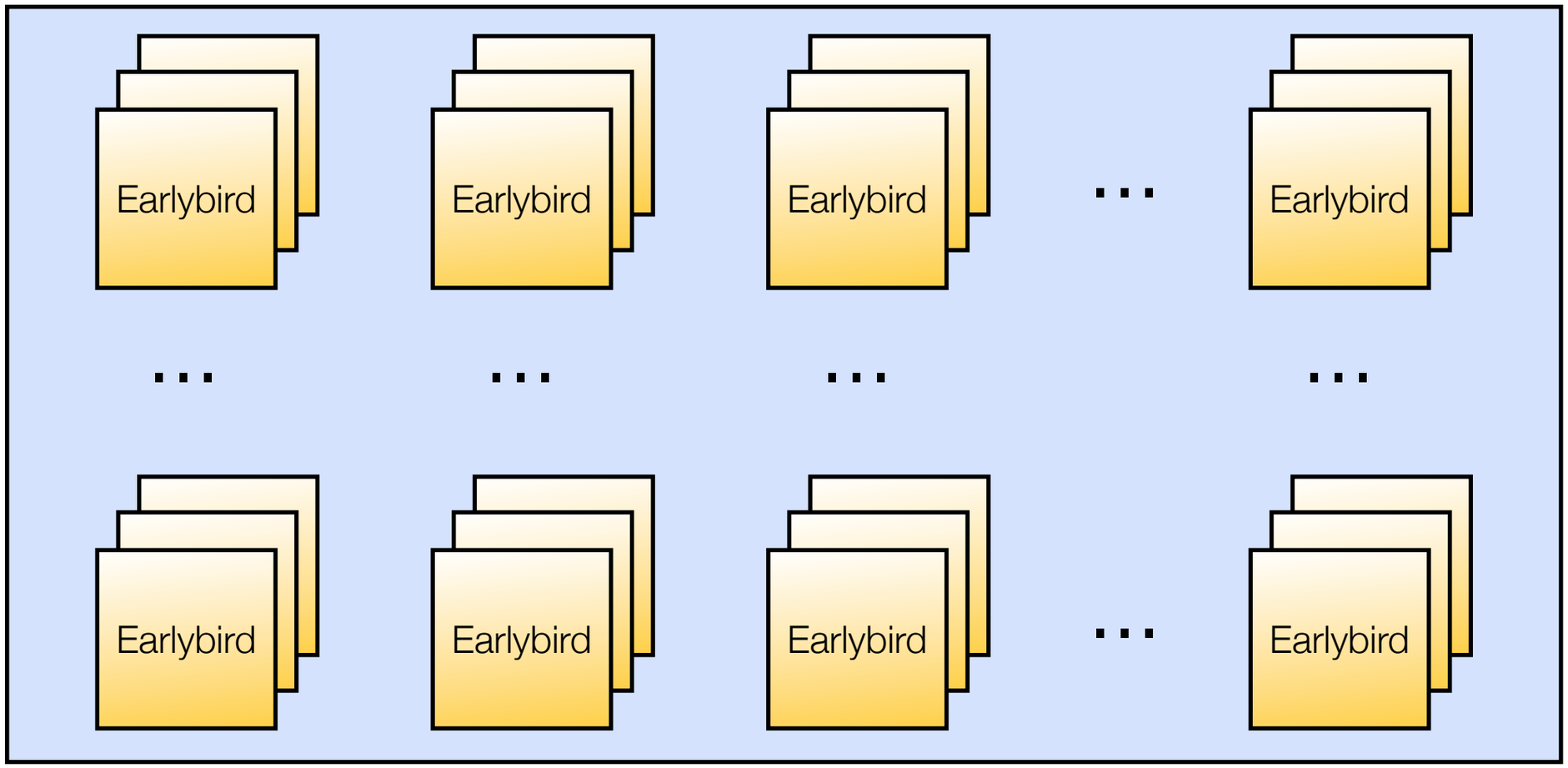


Cluster layout

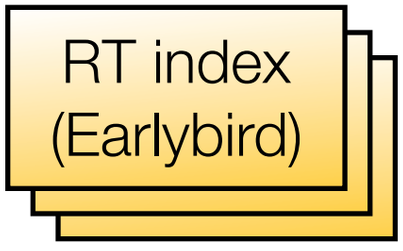
Writable timeslice



Complete timeslices

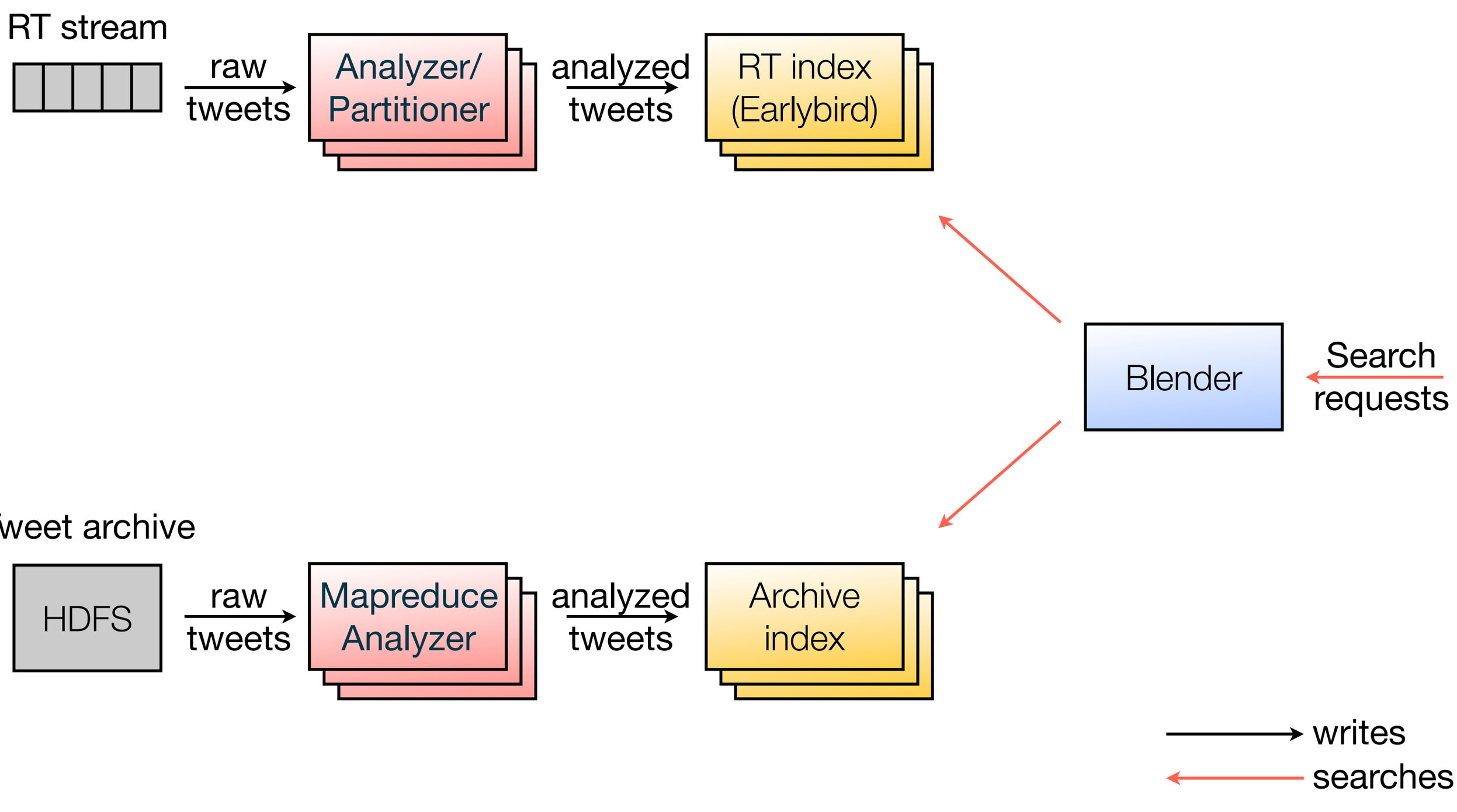


Search Architecture

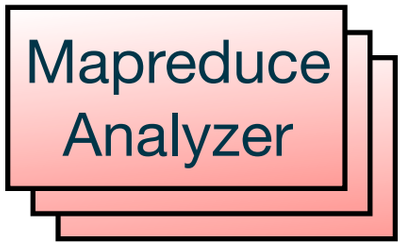


- Modified Lucene index implementation optimized for realtime search
- IndexWriter buffer is searchable (no need to flush to allow searching)
- In-memory
- Hash-partitioned, static layout

Search Architecture

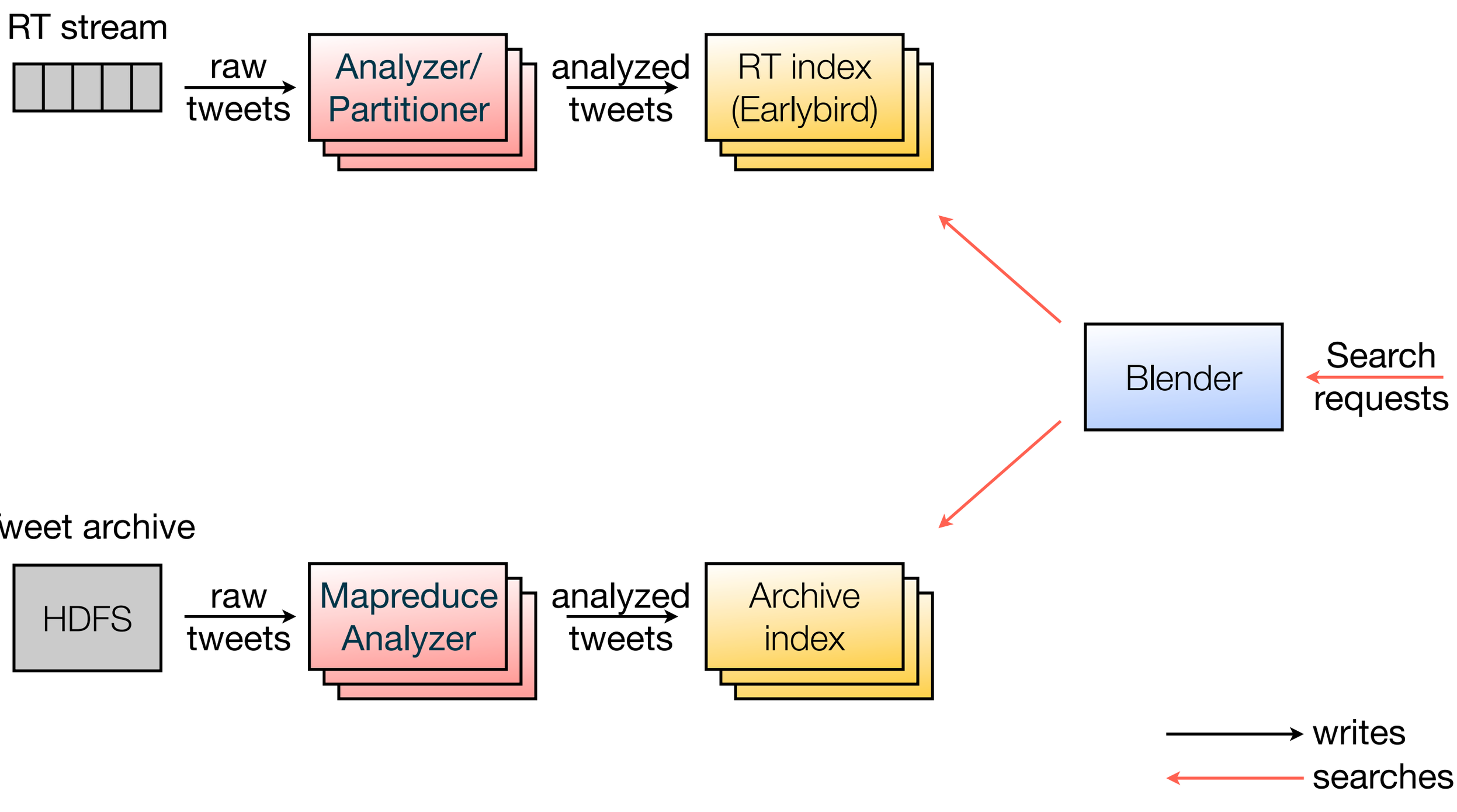


Search Architecture



- Daily jobs that process raw tweets
- Analyzes text
- Aggregates metadata and signals

Search Architecture



Search Architecture

- Standard Lucene (4.4) indexes
- Reverse time-sorted (new to old)
- Cluster layout similar to realtime search cluster

Search Architecture

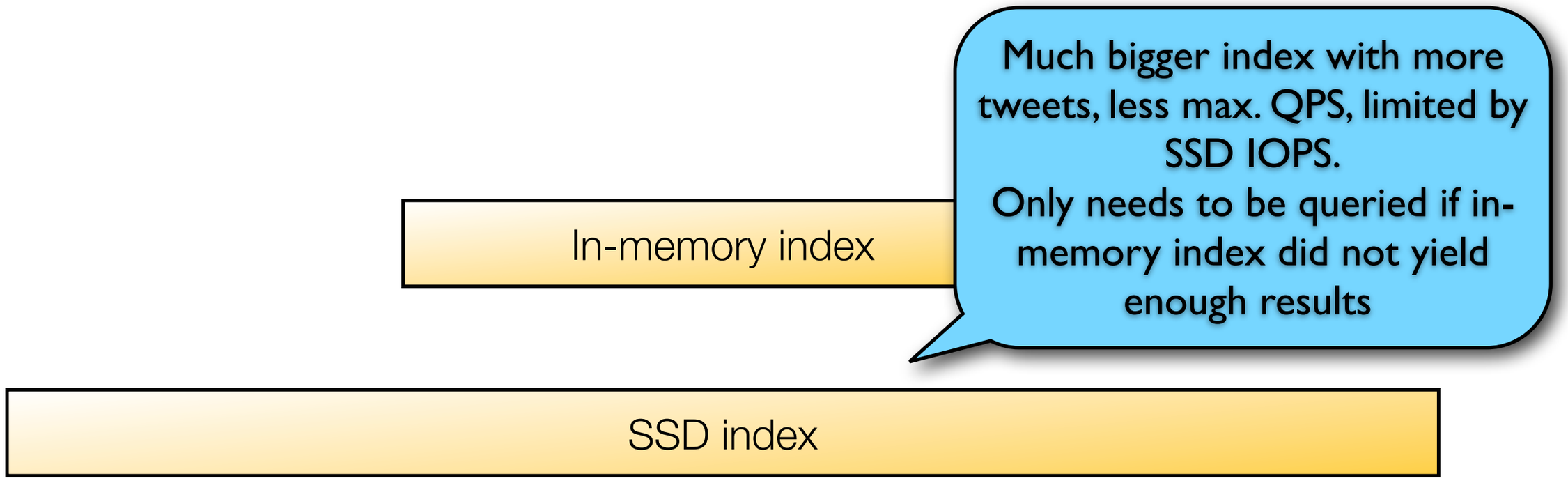
- Two tiers: In-memory and on SSD

Search Architecture

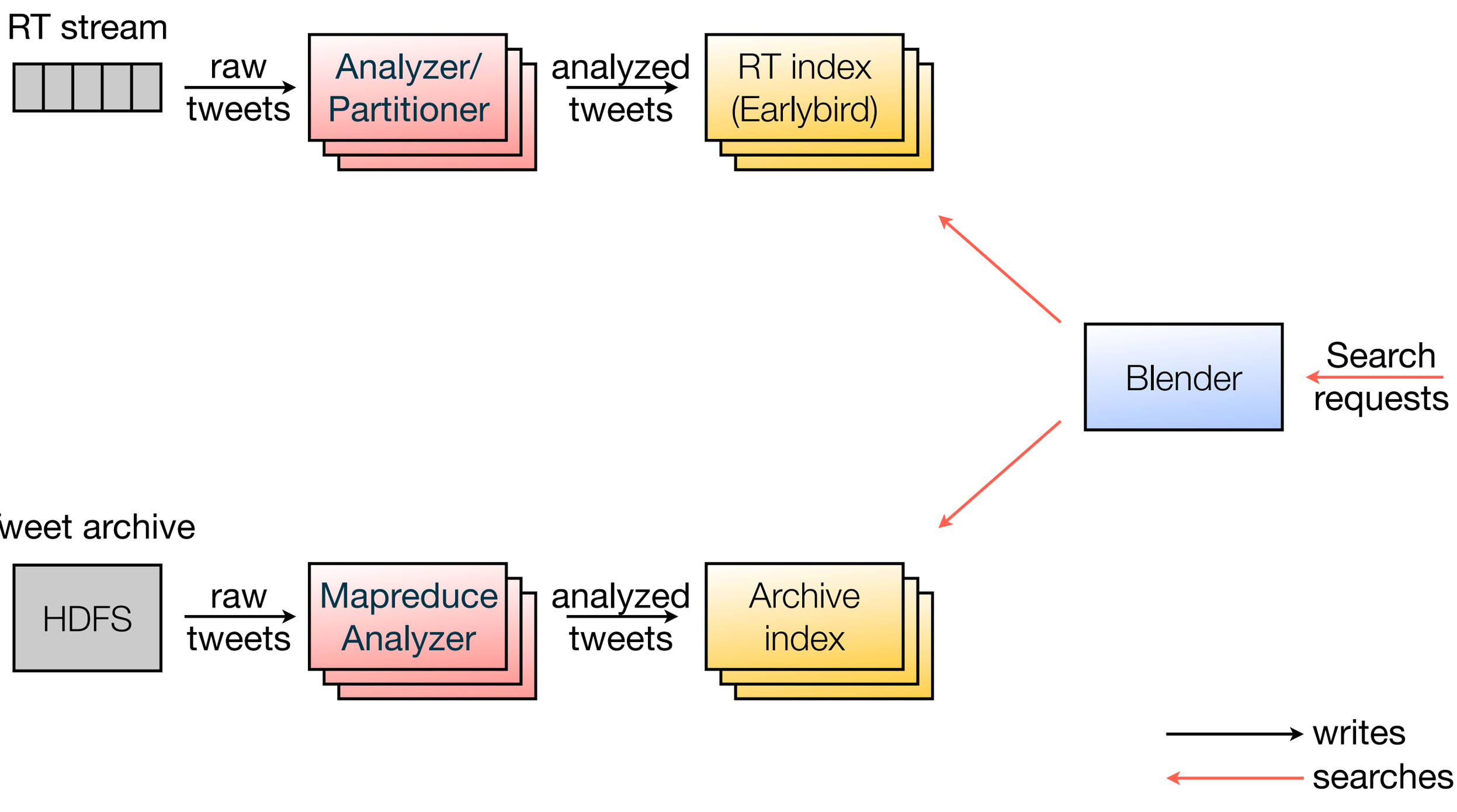
- Two tiers: In-memory and on SSD

Search Architecture

- Two tiers: In-memory and on SSD



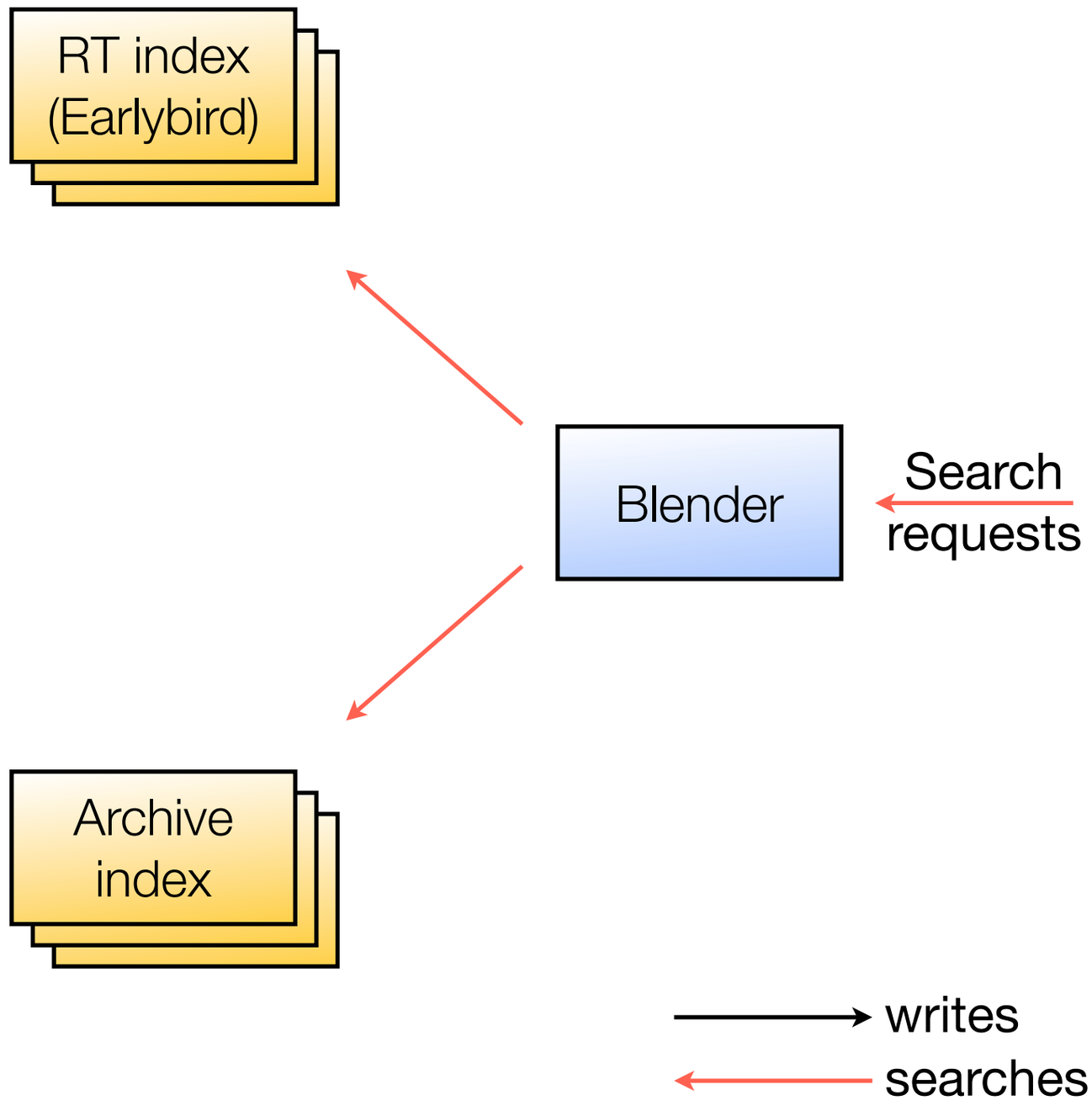
Search Architecture



Search Architecture

- Blender is our Thrift service aggregator

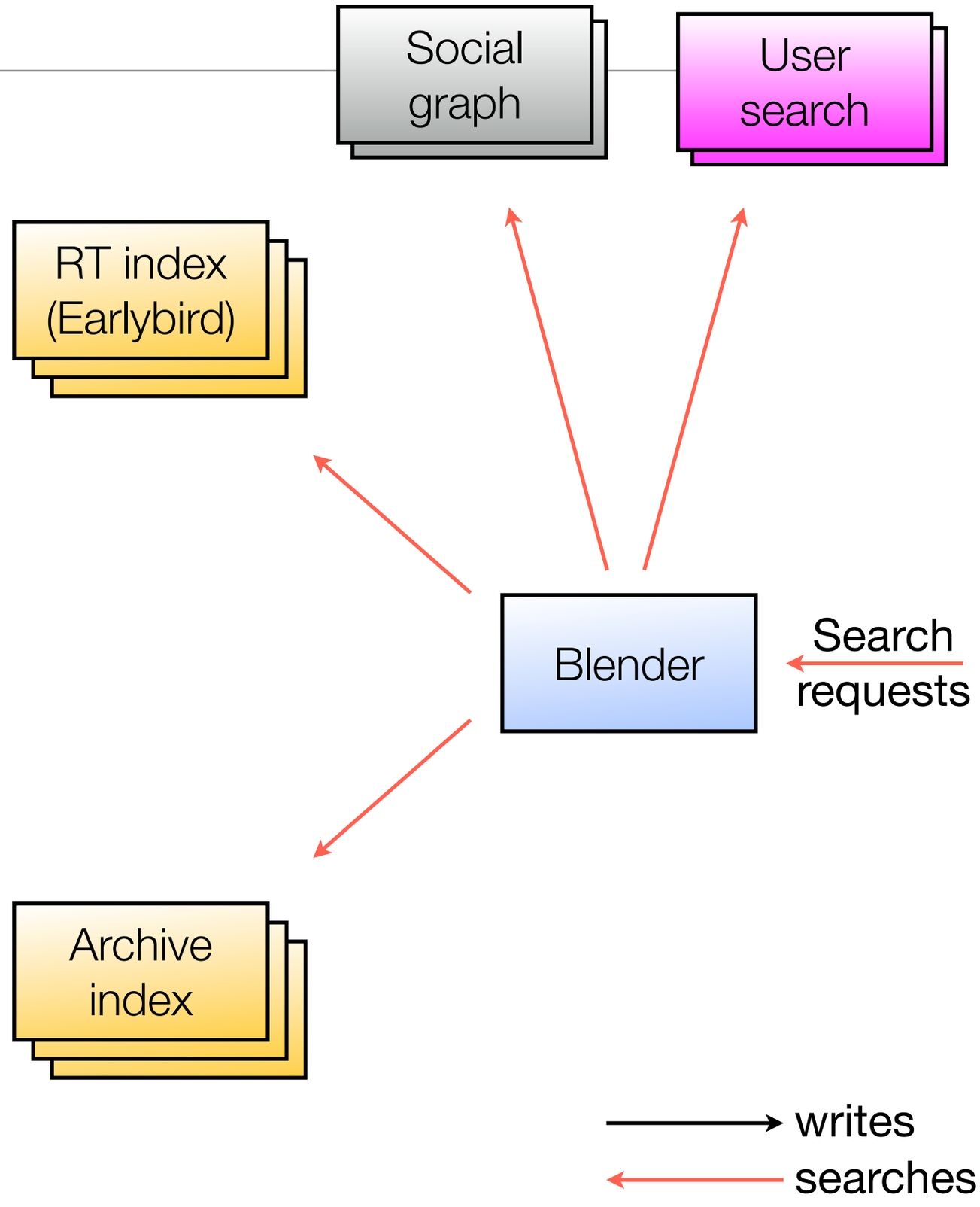
- Queries multiple Earlybirds, merges results



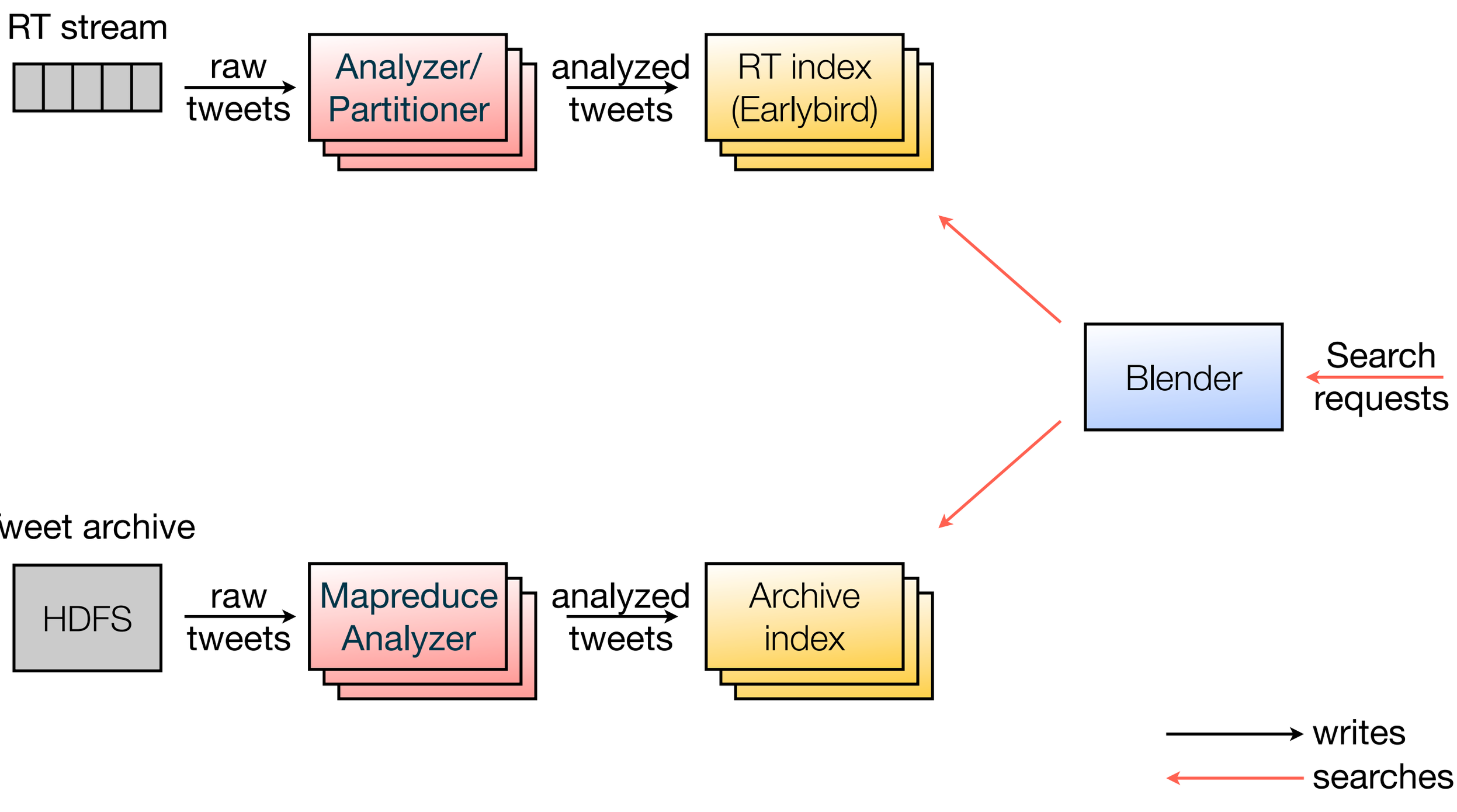
Search Architecture

- Blender is our Thrift service aggregator

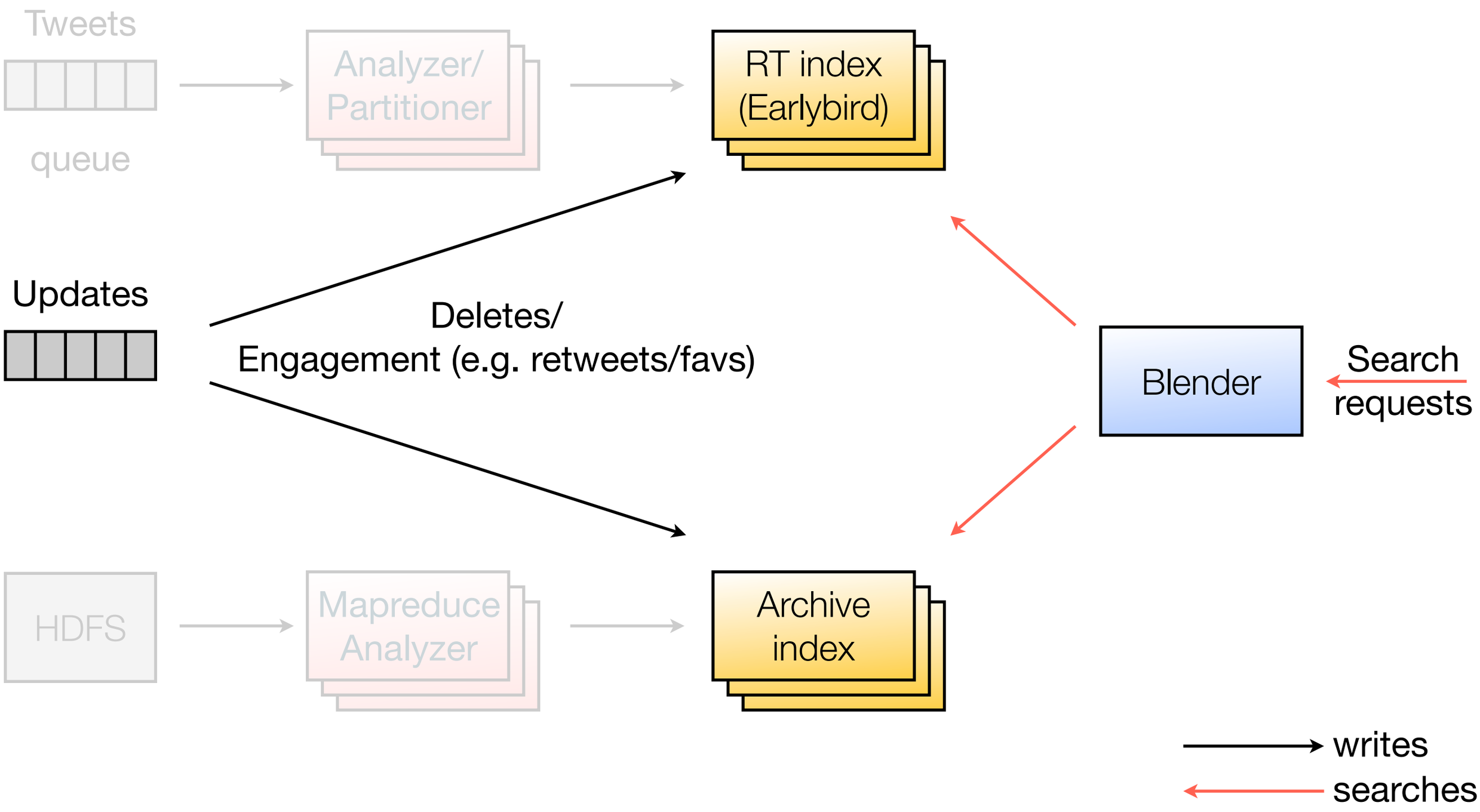
- Queries multiple Earlybirds, merges results



Search Architecture



Search Architecture



Realtime Search @twitter

Agenda

- Introduction
- Search Architecture
- Inverted Index
- Ranking

Inverted Index

Inverted Index 101

1	The old night keeper keeps the keep in the town
2	In the big old house in the big old gown.
3	The house in the town had the big old keep
4	Where the old night keeper never did sleep.
5	The night keeper keeps the keep in the night
6	And keeps in the dark and sleeps in the light.

Table with 6 documents

Example from:
Justin Zobel , Alistair Moffat,
Inverted files for text search engines,
ACM Computing Surveys (CSUR)
v.38 n.2, p.6-es, 2006

Inverted Index 101

1	The old night keeper keeps the keep in the town
2	In the big old house in the big old gown.
3	The house in the town had the big old keep
4	Where the old night keeper never did sleep.
5	The night keeper keeps the keep in the night
6	And keeps in the dark and sleeps in the light.

Table with 6 documents

term	freq	
and	1	<6>
big	2	<2> <3>
dark	1	<6>
did	1	<4>
gown	1	<2>
had	1	<3>
house	2	<2> <3>
in	5	<1> <2> <3> <5> <6>
keep	3	<1> <3> <5>
keeper	3	<1> <4> <5>
keeps	3	<1> <5> <6>
light	1	<6>
never	1	<4>
night	3	<1> <4> <5>
old	4	<1> <2> <3> <4>
sleep	1	<4>
sleeps	1	<6>
the	6	<1> <2> <3> <4> <5> <6>
town	2	<1> <3>
where	1	<4>

Dictionary and posting lists

Inverted Index 101

Query: keeper

1	The old night keeper keeps the keep in the town
2	In the big old house in the big old gown.
3	The house in the town had the big old keep
4	Where the old night keeper never did sleep.
5	The night keeper keeps the keep in the night
6	And keeps in the dark and sleeps in the light.

Table with 6 documents

term	freq	
and	1	<6>
big	2	<2> <3>
dark	1	<6>
did	1	<4>
gown	1	<2>
had	1	<3>
house	2	<2> <3>
in	5	<1> <2> <3> <5> <6>
keep	3	<1> <3> <5>
keeper	3	<1> <4> <5>
keeps	3	<1> <5> <6>
light	1	<6>
never	1	<4>
night	3	<1> <4> <5>
old	4	<1> <2> <3> <4>
sleep	1	<4>
sleeps	1	<6>
the	6	<1> <2> <3> <4> <5> <6>
town	2	<1> <3>
where	1	<4>

Dictionary and posting lists

Inverted Index 101

Query: keeper

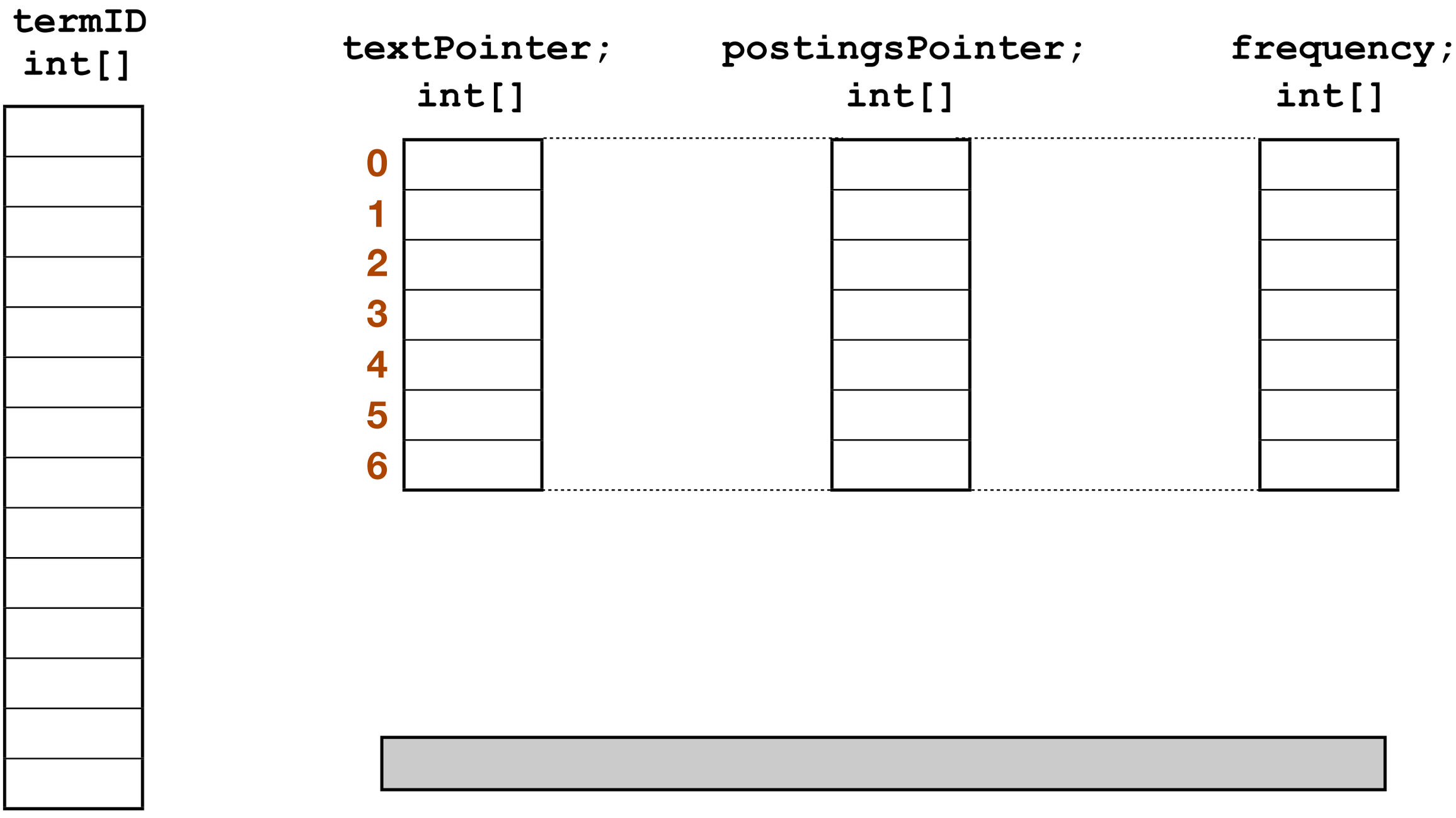
1	The old night **keeper** keeps the keep in the town
2	In the big old house in the big old gown.
3	The house in the town had the big old keep
4	Where the old night **keeper** never did sleep.
5	The night **keeper** keeps the keep in the night
6	And keeps in the dark and sleeps in the light.

Table with 6 documents

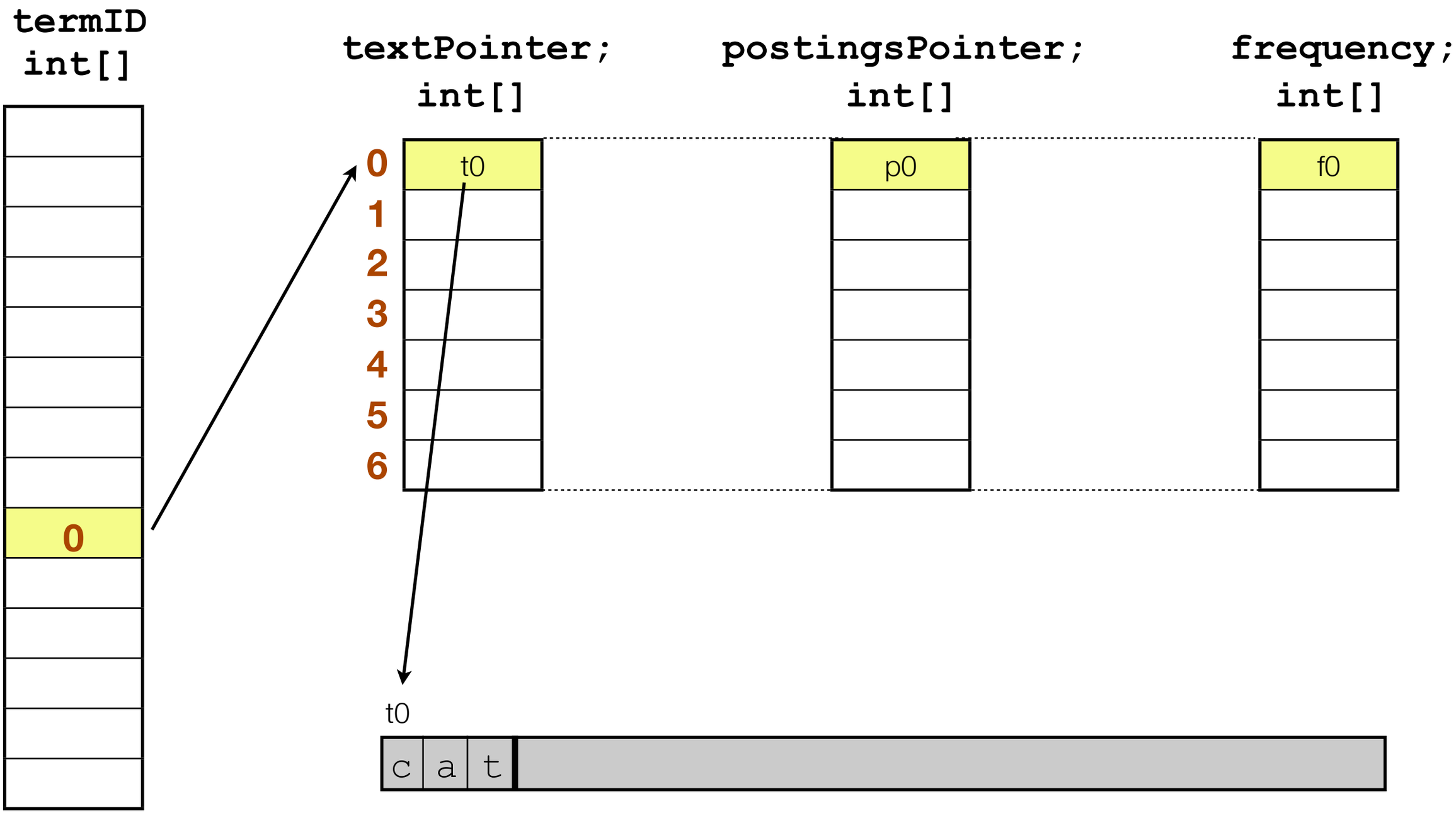
term	freq	
and	1	<6>
big	2	<2> <3>
dark	1	<6>
did	1	<4>
gown	1	<2>
had	1	<3>
house	2	<2> <3>
in	5	<1> <2> <3> <5> <6>
keep	3	<1> <3> <5>
keeper	**3**	**<1> <4> <5>**
keeps	3	<1> <5> <6>
light	1	<6>
never	1	<4>
night	3	<1> <4> <5>
old	4	<1> <2> <3> <4>
sleep	1	<4>
sleeps	1	<6>
the	6	<1> <2> <3> <4> <5> <6>
town	2	<1> <3>
where	1	<4>

Dictionary and posting lists

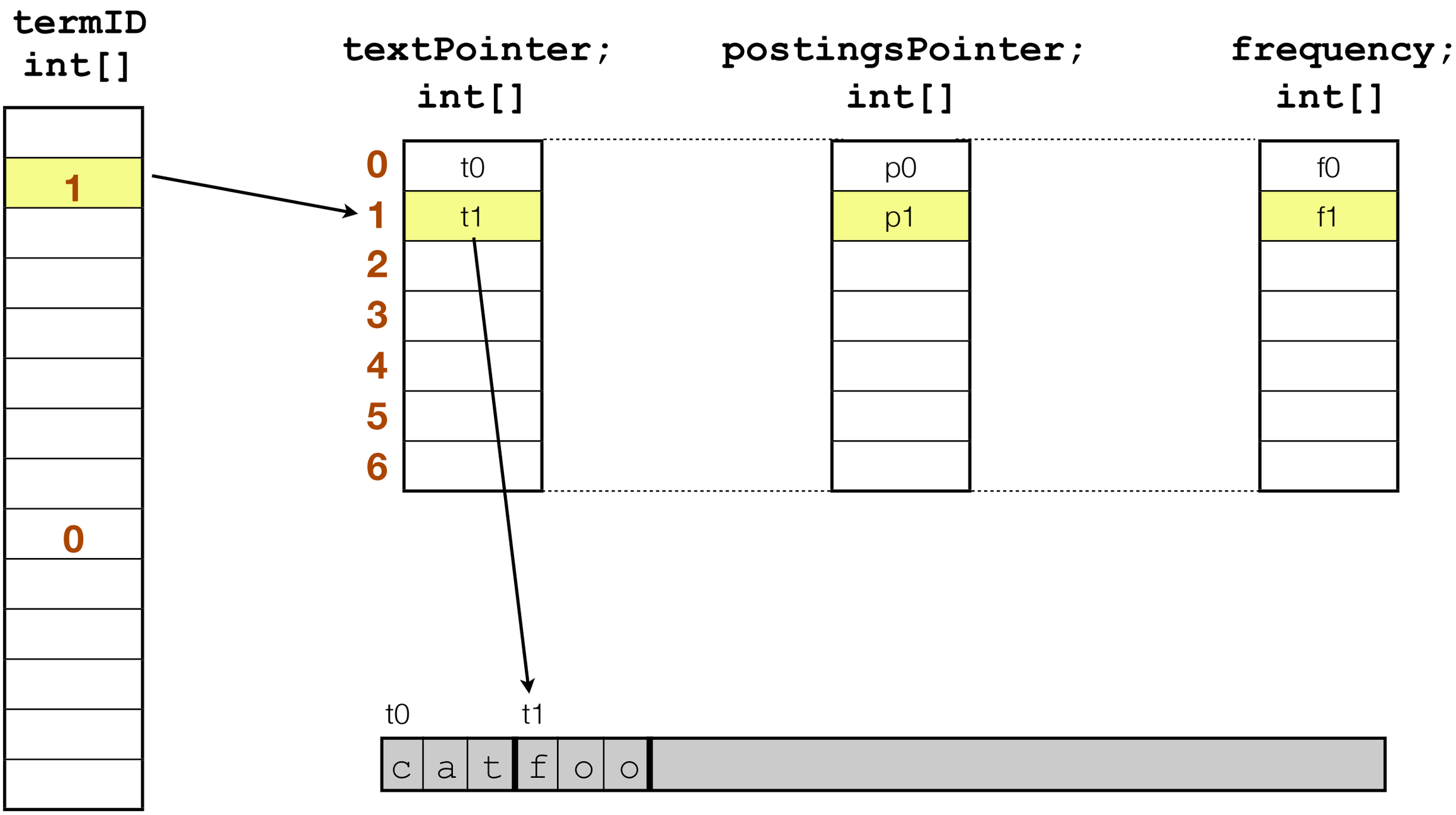
Term dictionary



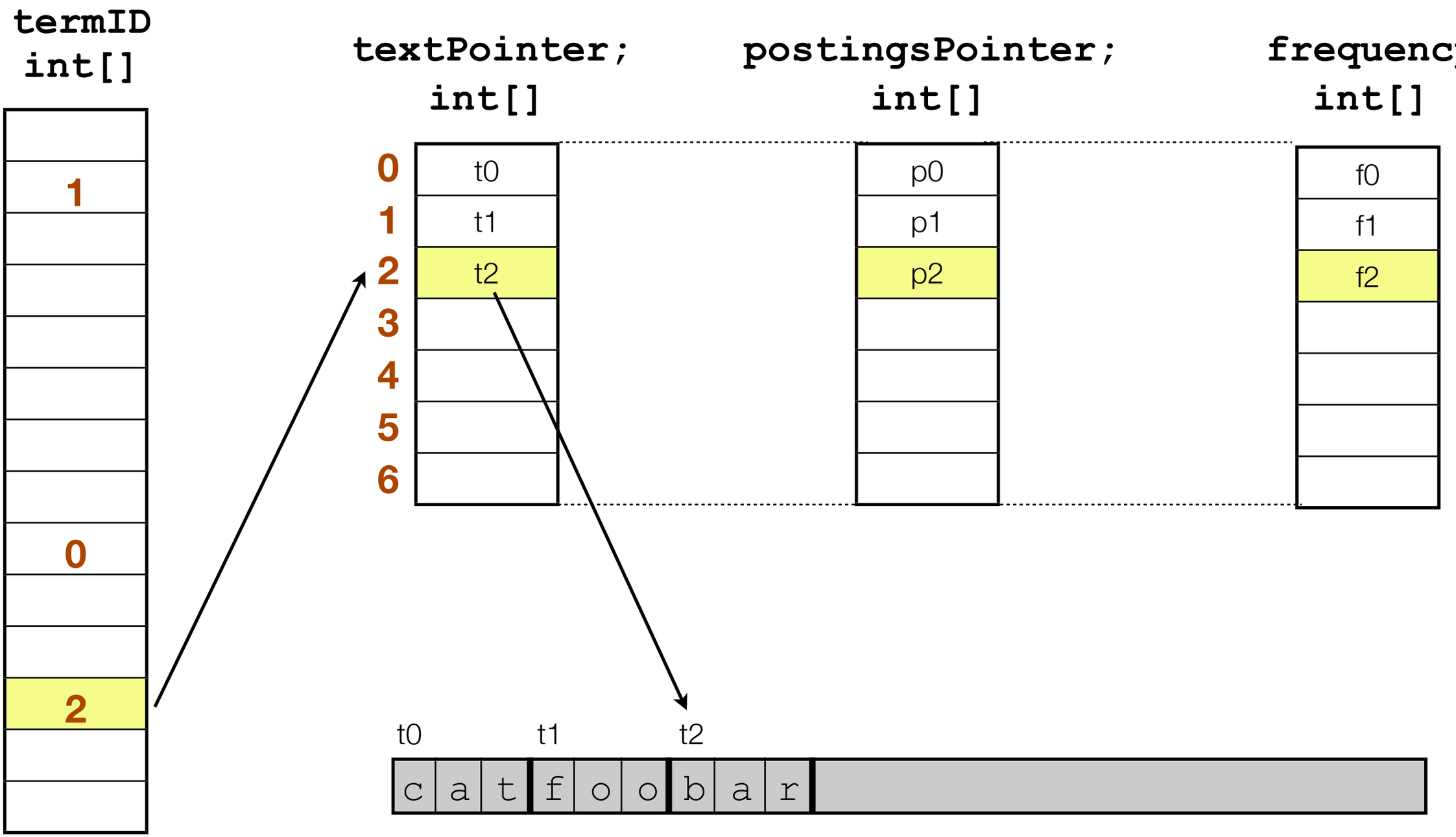
Term dictionary



Term dictionary



Term dictionary



Posting list encoding

Doc IDs to encode: 5, 15, 9000, 9002, 100000, 100090

Posting list encoding

Doc IDs to encode: 5, 15, 9000, 9002, 100000, 100090

Delta encoding:

5	10	8985	2	90998	90

Posting list encoding

Doc IDs to encode: 5, 15, 9000, 9002, 100000, 100090

Delta encoding:

5	10	8985	2	90998	90

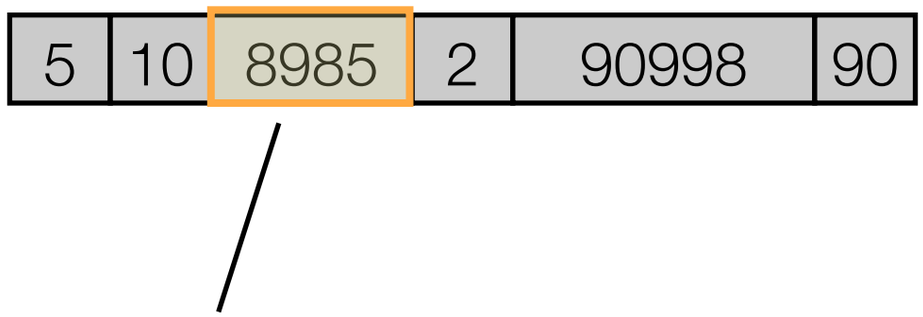
VInt compression: 00000101

Values 0 <= delta <= 127 need one byte

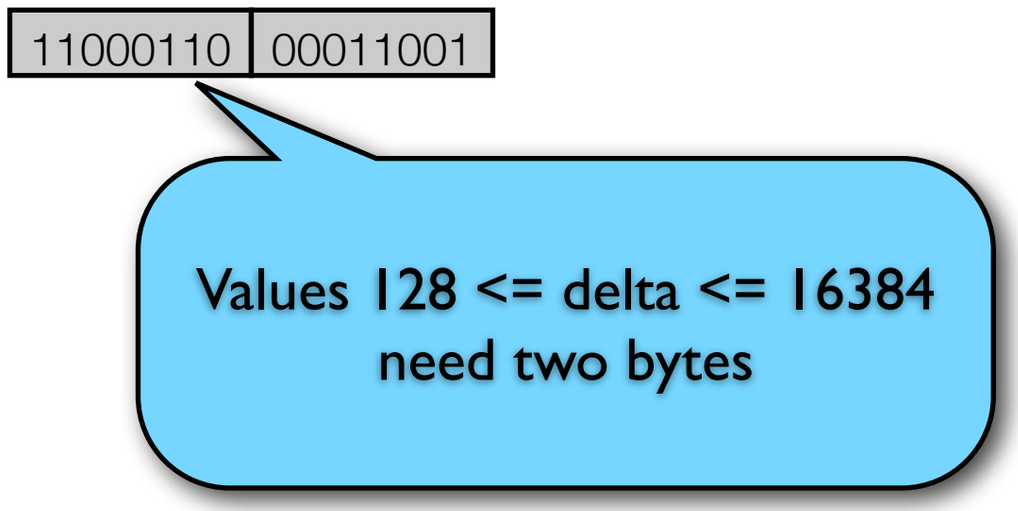
Posting list encoding

Doc IDs to encode: 5, 15, 9000, 9002, 100000, 100090

Delta encoding:



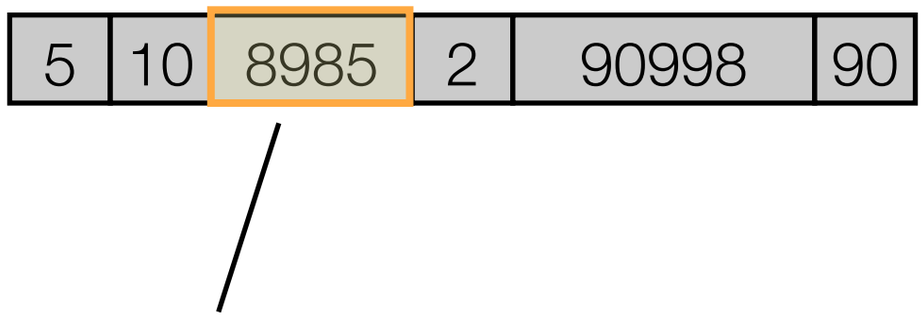
VInt compression:



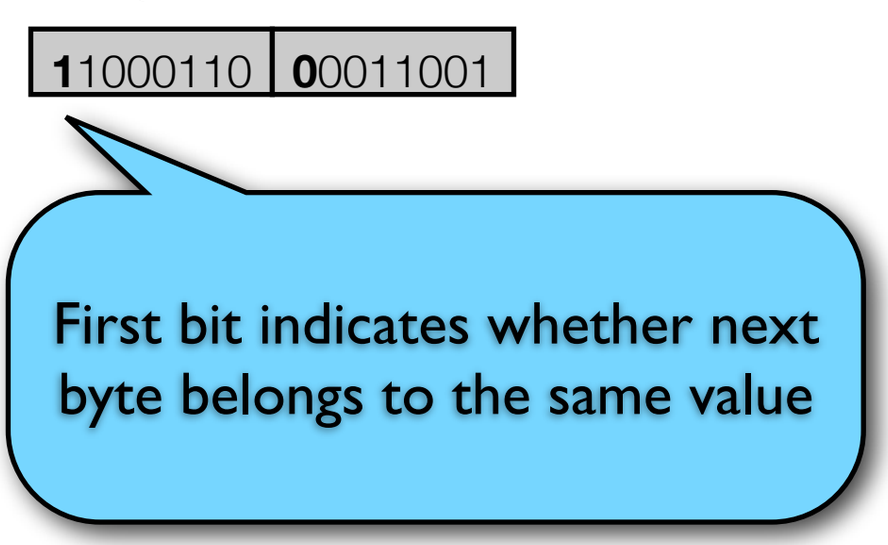
Posting list encoding

Doc IDs to encode: 5, 15, 9000, 9002, 100000, 100090

Delta encoding:



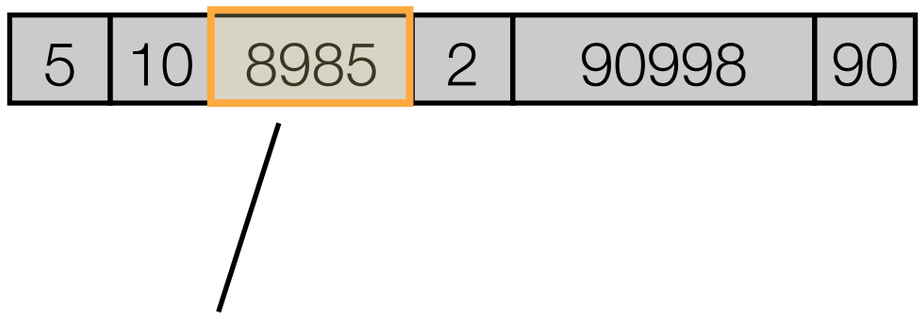
VInt compression:



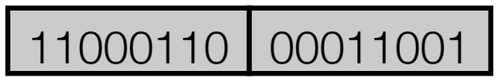
Posting list encoding

Doc IDs to encode: 5, 15, 9000, 9002, 100000, 100090

Delta encoding:



VInt compression:



- Variable number of bytes - a VInt-encoded posting can not be written as a primitive Java type; therefore it can not be written atomically

Posting list encoding

Doc IDs to encode: 5, 15, 9000, 9002, 100000, 100090

Delta encoding:

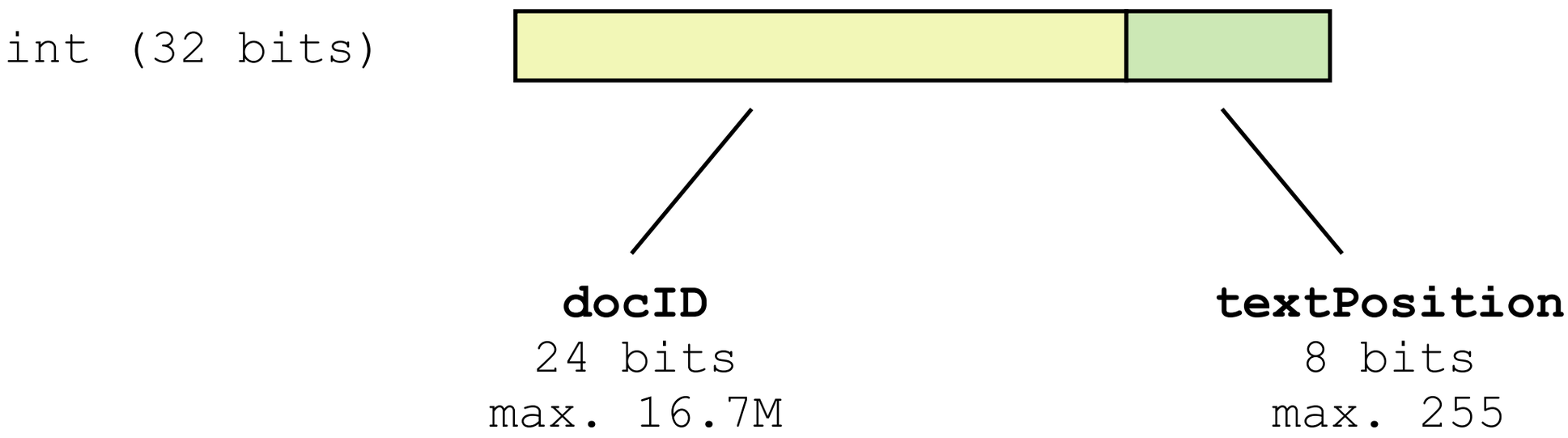
5	10	8985	2	90998	90

Read direction

- Each posting depends on previous one; decoding only possible in old-to-new direction

- With recency ranking (new-to-old) no early termination is possible

Posting list encoding in Earlybird v1

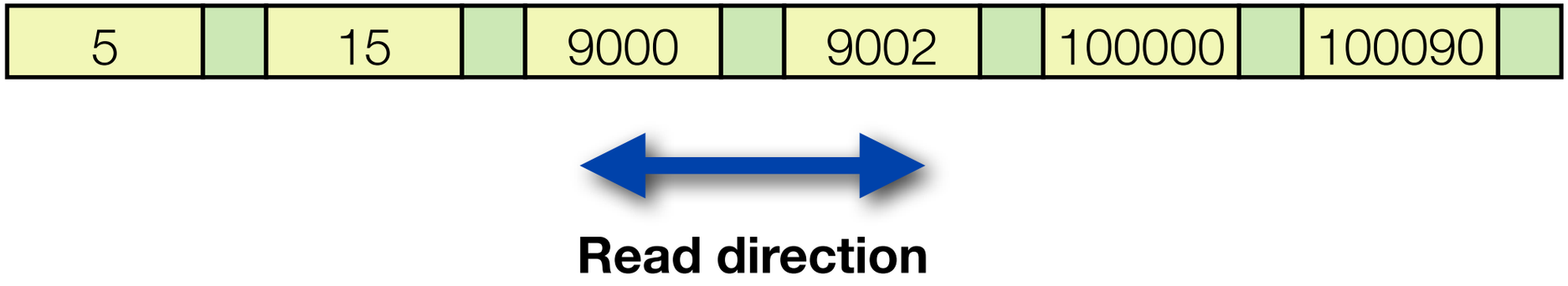


- Tweet text can only have 140 chars

Posting list encoding in Earlybird v1

Doc IDs to encode: 5, 15, 9000, 9002, 100000, 100090

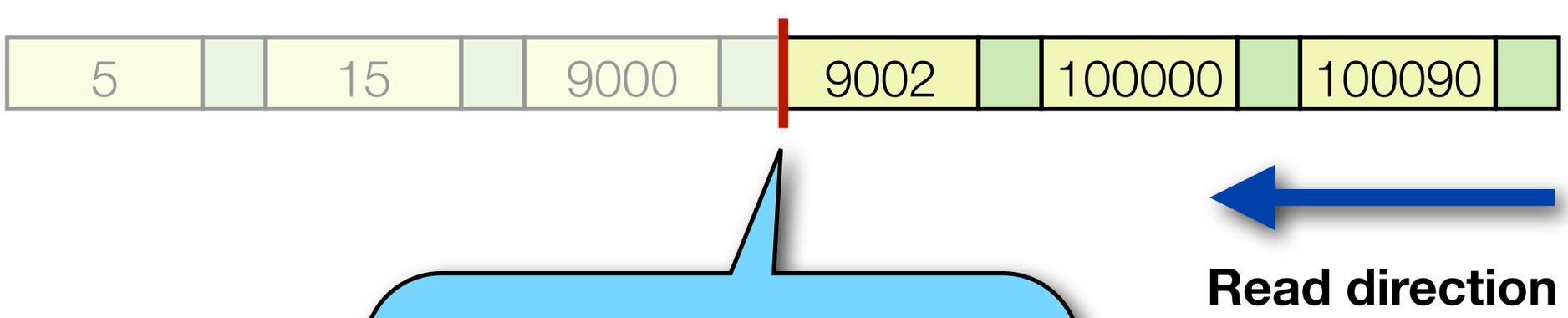
Earlybird encoding:



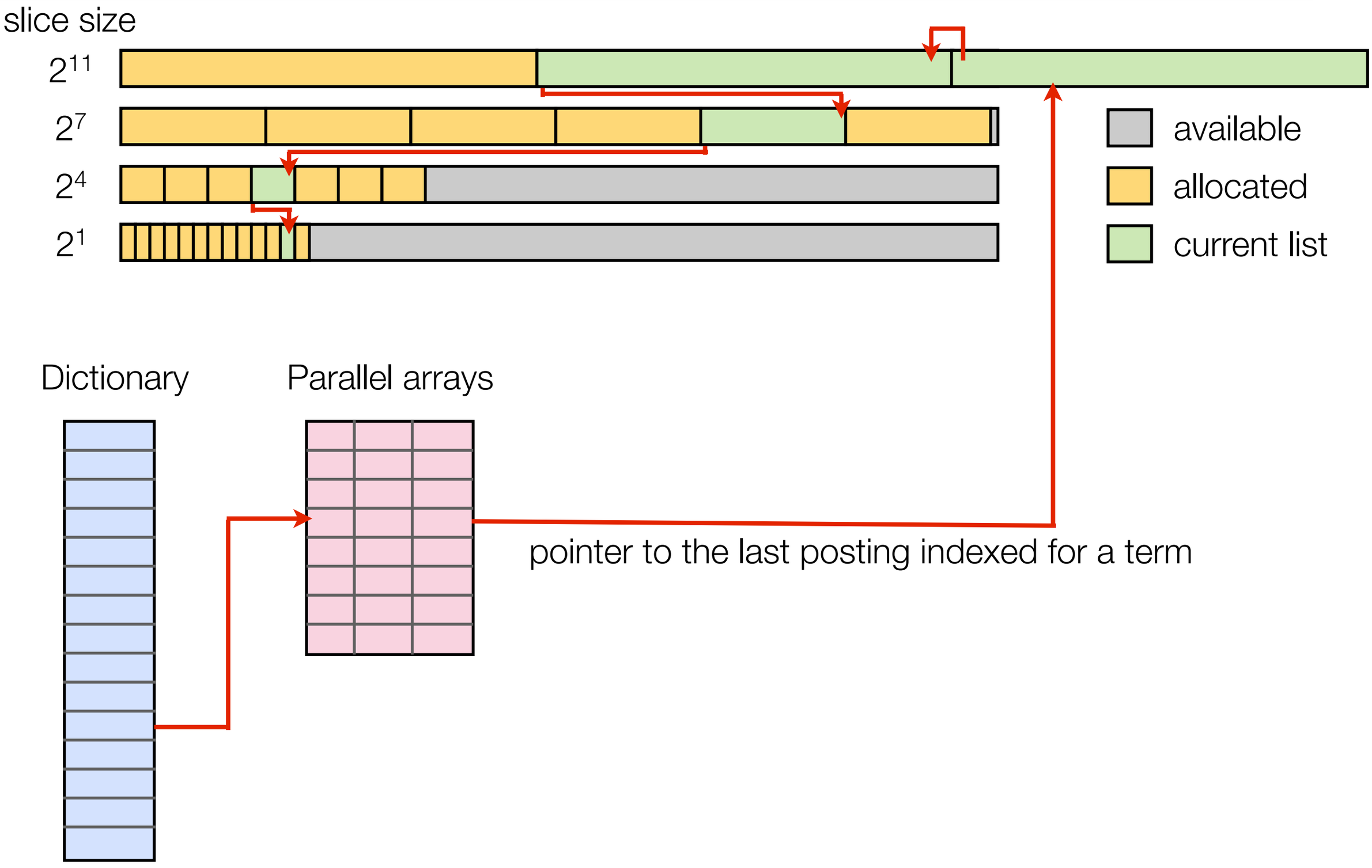
Early query termination

Doc IDs to encode: 5, 15, 9000, 9002, 100000, 100090

Earlybird encoding:



Memory model



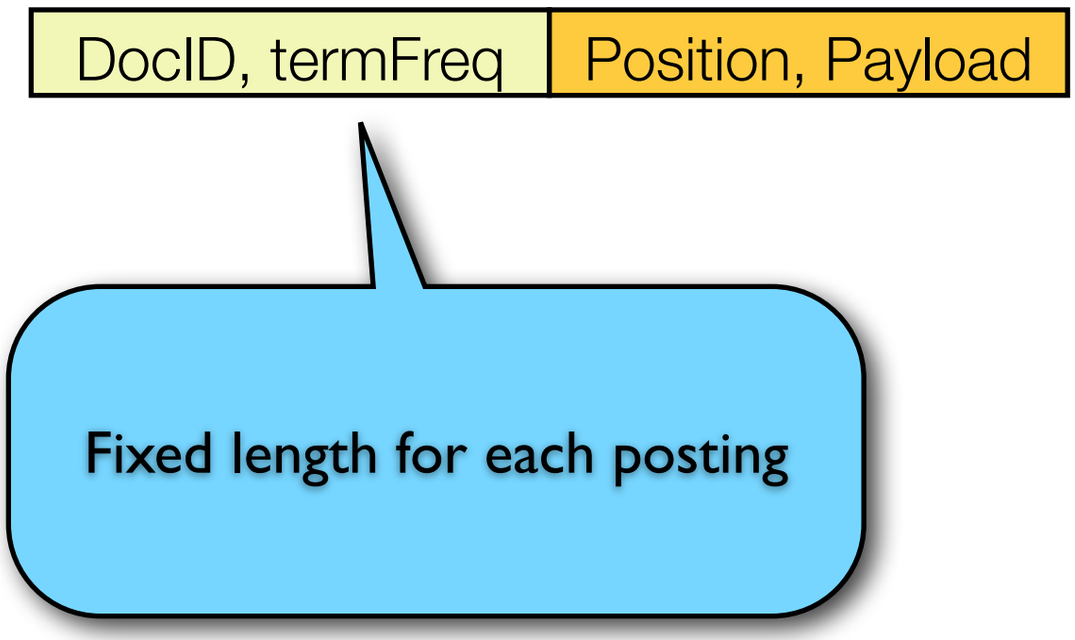
Posting list encoding - Summary

- Integers can be written atomically in Java
- Backwards traversal easy on absolute docIDs (not deltas)
- Repeating docIDs if a term occurs multiple times in the same document only works for small docs
- Stored in equally sized int[] arrays to reduce garbage collection costs
- Max. segment size: 2^24 = 16.7M tweets

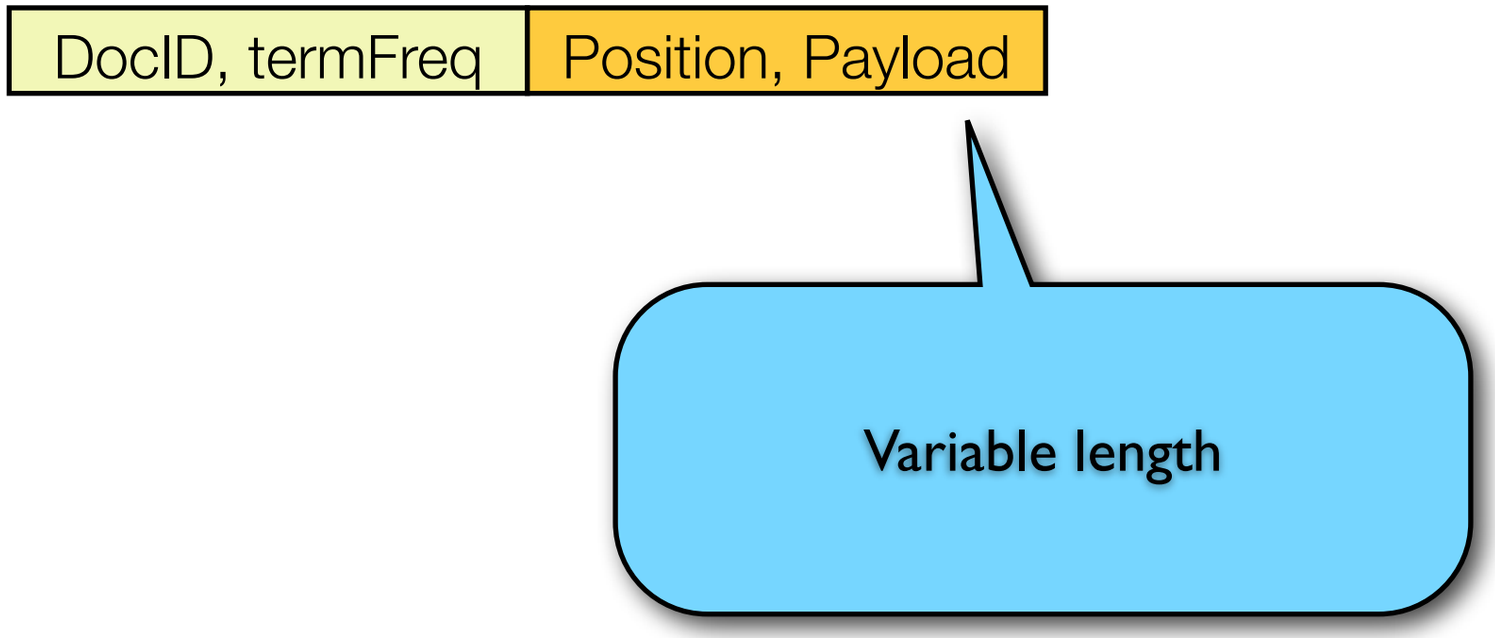
New posting list encoding

- Objectives:
  - 32 bit positions and variable-length payloads
  - Store term frequency (TF) instead of repeating docIDs
- Keep:
  - Concurrency model
  - Space-efficiency for short documents
  - Performance

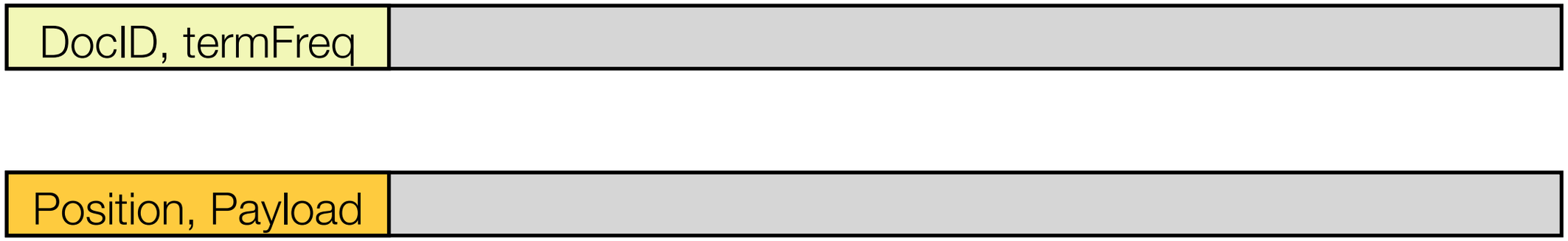
New posting list encoding



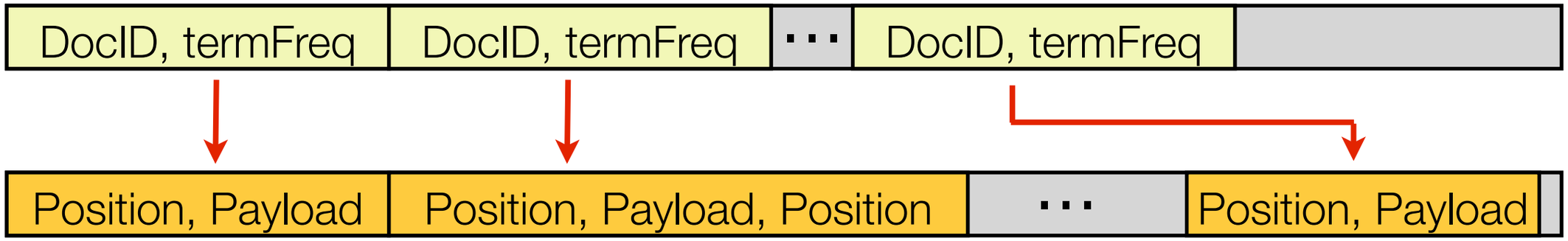
New posting list encoding



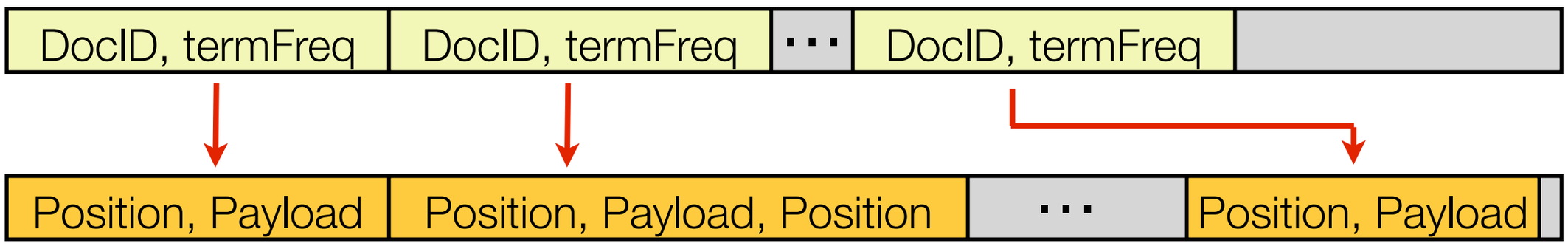
New posting list encoding



New posting list encoding

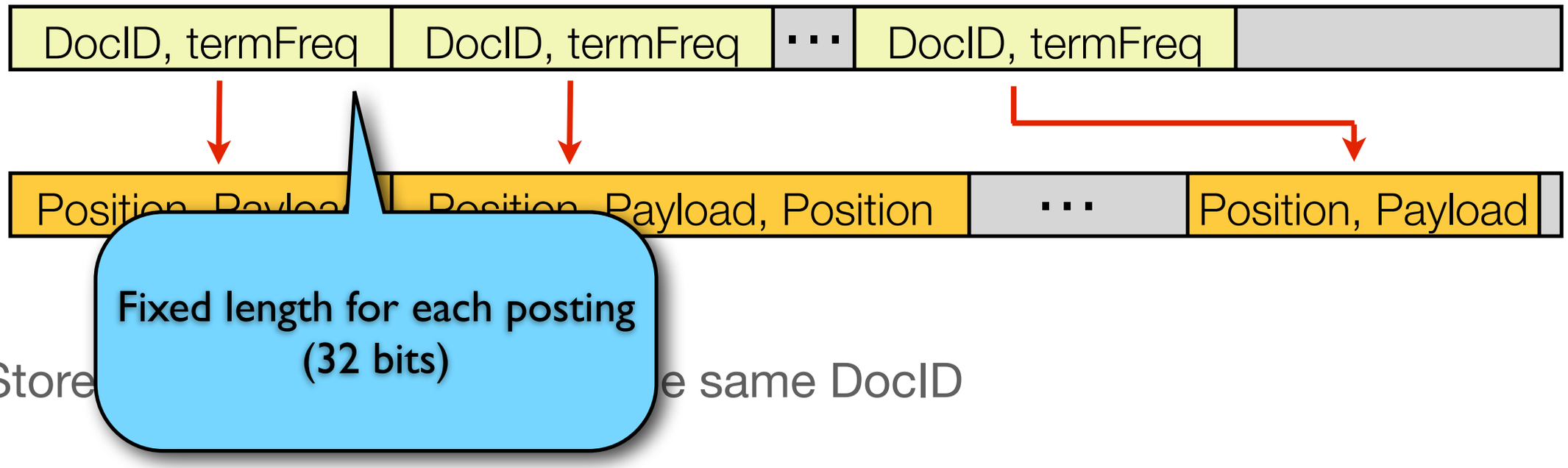


New posting list encoding



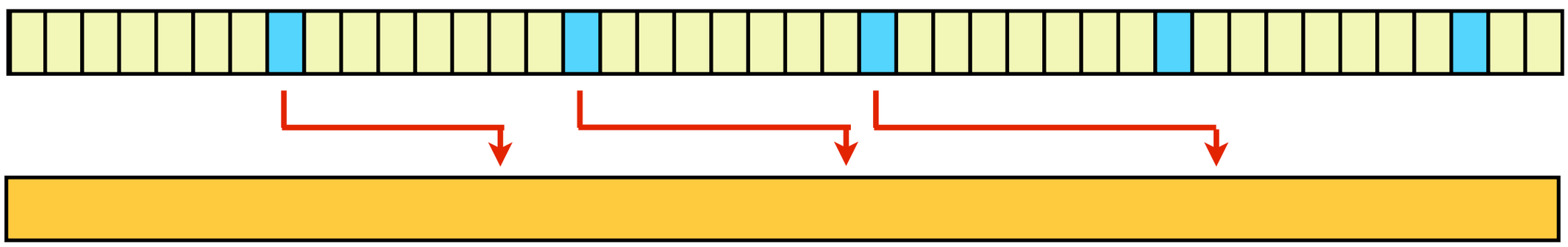
- Store TF instead of repeating the same DocID
- Store DocID/TF pairs separately from position/payloads
- Find a way to synchronously decode the two streams without storing a pointer for each posting (expensive)

New posting list encoding



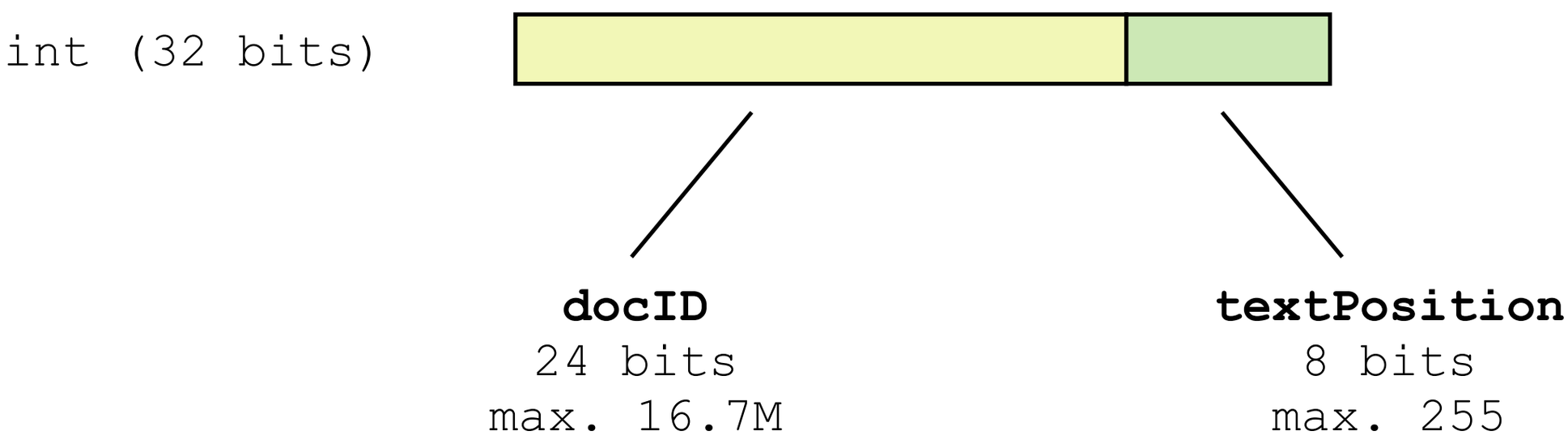
- Store e same DocID
- Store DocID/TF pairs separately from position/payloads
- Find a way to synchronously decode the two streams without storing a pointer for each posting (expensive)

New posting list encoding



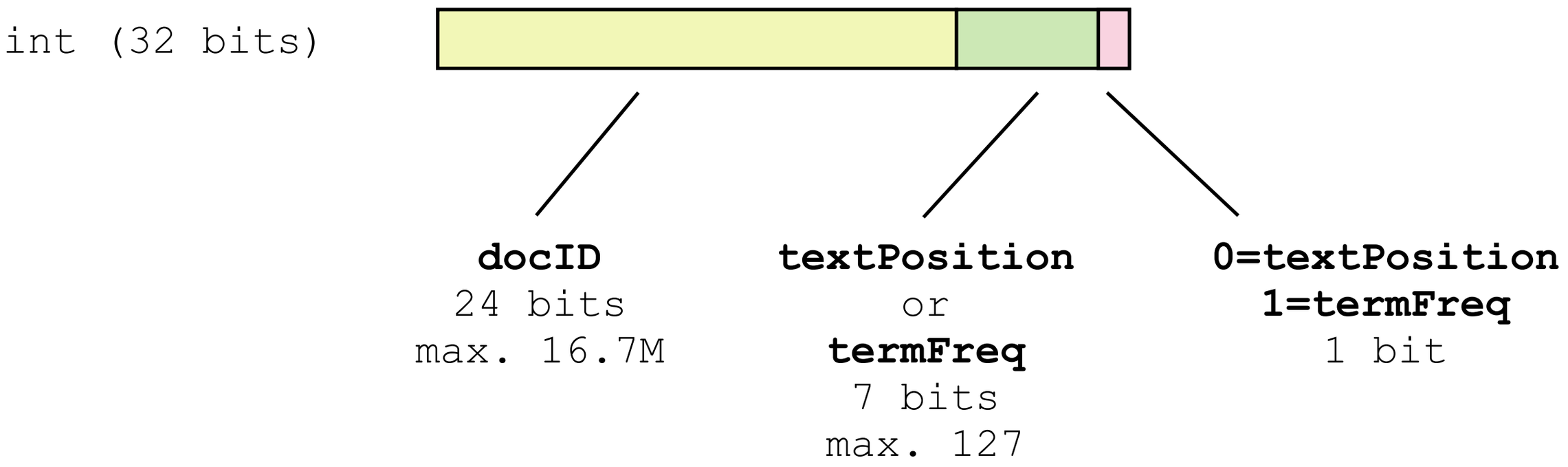
- Idea: Use an embedded skip list as periodical “synchronization points”
- Keeps memory overhead for pointers low and improves search performance

New posting list encoding



- Observation: Most tweets don’t need all 8 bits for text position
- Idea: Use the position “inlining” approach for short documents, but support Lucene’s 32-bit positions and variable length payloads

New posting list encoding



As a storage optimization, the text position is stored **with** the docID if:

- *termFreq* == 1 (term occurs once only in the doc) AND
- *textPosition* <= 127 AND
- Posting has no payload AND
- Posting is not at a skip point of the docID posting list (see later).

New posting list encoding - Summary

- Support for 32 bit positions and arbitrary length payloads stored in separate data structure

- Performance and space consumption very similar compared to previous encoding for tweet search

- Skip lists used for speed and synchronization points

- For short documents positions can still be inlined

Realtime Search @twitter

Agenda

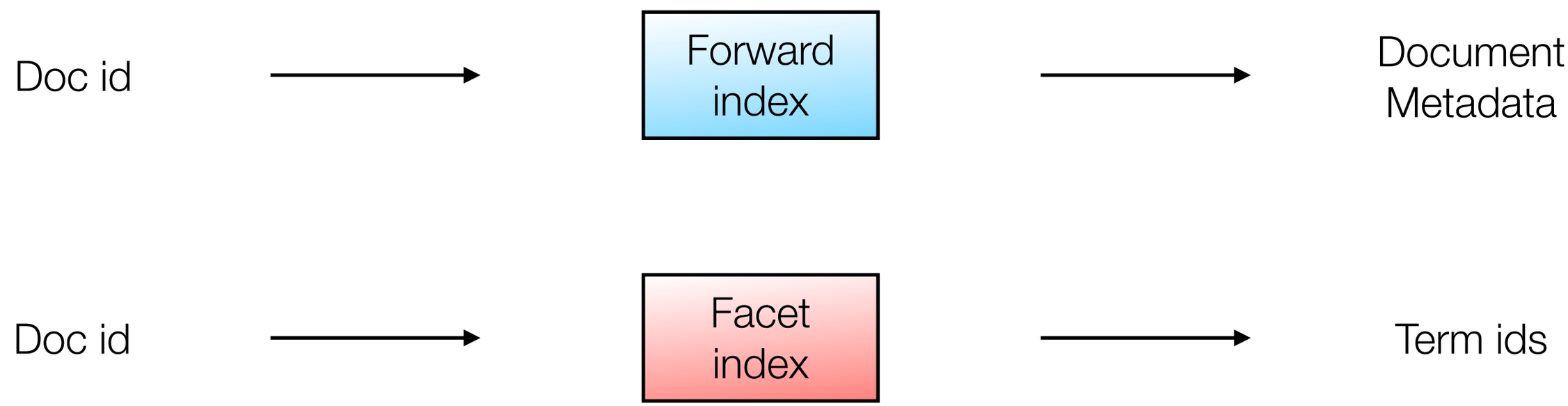
- Introduction
- Search Architecture
- Inverted Index
- Ranking

Ranking

Storing tweet metadata

Storing tweet metadata

Storing tweet metadata



Storing tweet metadata

Forward index

- Similar to Lucene's DocValues
- Stores tweet features, such as retweet, favorite, reply counts
- In memory, updatable in-place
- Type-system; supports packing multiple values into single ints

Storing tweet metadata

Forward index

- Scale: We have > 300B tweets

Storing tweet metadata

Forward index

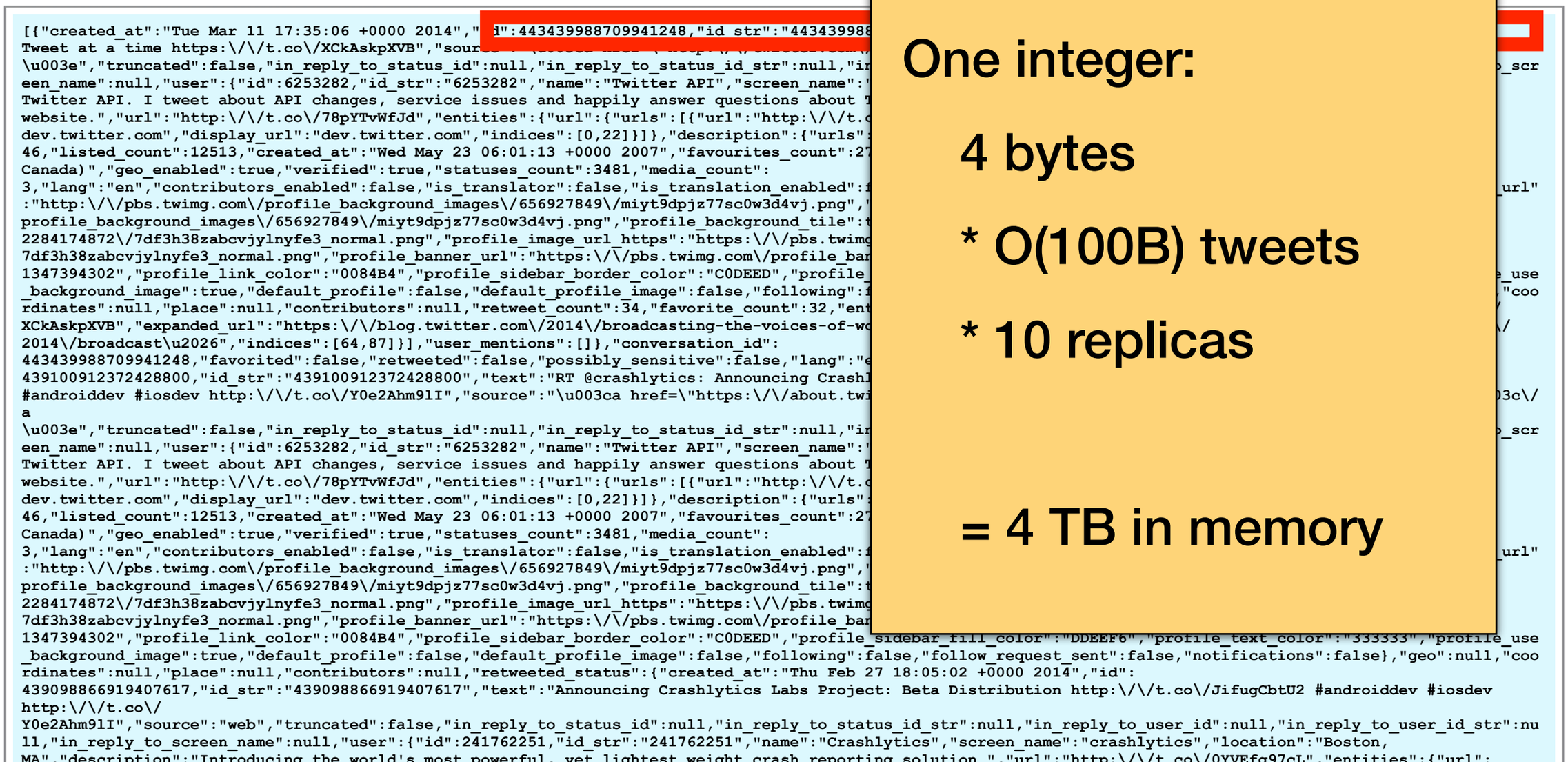
- Scale: We have > 300B tweets


```
[{"created_at":"Tue Mar 11 17:35:06 +0000 2014","id":443439988709941248,"id_str":"443439988709941248","text":"Broadcasting the voices of WordPress users, one
Tweet at a time https:\/\/t.co\/XCkAskpXVB","source":"\u003ca href=\"https:\/\/about.twitter.com\/products\/tweetdeck\" rel=\"nofollow\"\u003eTweetDeck
\u003e","truncated":false,"in_reply_to_status_id":null,"in_reply_to_status_id_str":null,"in_reply_to_user_id":null,"in_reply_to_user_id_str":null,"in_reply_to_scr
een_name":null,"user":{"id":6253282,"id_str":"6253282","name":"Twitter API","screen_name":"twitterapi","location":"San Francisco, CA","description":"The Real
Twitter API. I tweet about API changes, service issues and happily answer questions about Twitter and our API. Don't get an answer? It's on my
website.","url":"http:\/\/t.co\/78pYTvWfJd","entities":{"url":{"urls":[{"url":"http:\/\/t.co\/78pYTvWfJd","expanded_url":"http:\/\/
dev.twitter.com","display_url":"dev.twitter.com","indices":[0,22]}]},"description":{"urls":[]}},"protected":false,"followers_count":2091868,"friends_count":
46,"listed_count":12513,"created_at":"Wed May 23 06:01:13 +0000 2007","favourites_count":27,"utc_offset":-25200,"time_zone":"Pacific Time (US &
Canada)","geo_enabled":true,"verified":true,"statuses_count":3481,"media_count":
3,"lang":"en","contributors_enabled":false,"is_translator":false,"is_translation_enabled":false,"profile_background_color":"C0DEED","profile_background_image_url"
:"http:\/\/pbs.twimg.com\/profile_background_images\/656927849\/miyt9dpjz77sc0w3d4vj.png","profile_background_image_url_https":"https:\/\/pbs.twimg.com\/
profile_background_images\/656927849\/miyt9dpjz77sc0w3d4vj.png","profile_background_tile":true,"profile_image_url":"http:\/\/pbs.twimg.com\/profile_images\/
2284174872\/7df3h38zabcvjylnyfe3_normal.png","profile_image_url_https":"https:\/\/pbs.twimg.com\/profile_images\/2284174872\/
7df3h38zabcvjylnyfe3_normal.png","profile_banner_url":"https:\/\/pbs.twimg.com\/profile_banners\/6253282\/
1347394302","profile_link_color":"0084B4","profile_sidebar_border_color":"C0DEED","profile_sidebar_fill_color":"DDEEF6","profile_text_color":"333333","profile_use
_background_image":true,"default_profile":false,"default_profile_image":false,"following":false,"follow_request_sent":false,"notifications":false},"geo":null,"coo
rdinates":null,"place":null,"contributors":null,"retweet_count":34,"favorite_count":32,"entities":{"hashtags":[],"symbols":[],"urls":[{"url":"https:\/\/t.co\/
XCkAskpXVB","expanded_url":"https:\/\/blog.twitter.com\/2014\/broadcasting-the-voices-of-wordpress-users-one-tweet-at-a-time","display_url":"blog.twitter.com\/
2014\/broadcast\u2026","indices":[64,87]}],"user_mentions":[]},"conversation_id":
443439988709941248,"favorited":false,"retweeted":false,"possibly_sensitive":false,"lang":"en"},{"created_at":"Thu Feb 27 18:13:10 +0000 2014","id":
439100912372428800,"id_str":"439100912372428800","text":"RT @crashlytics: Announcing Crashlytics Labs Project: Beta Distribution http:\/\/t.co\/JifugCbtU2
#androiddev #iosdev http:\/\/t.co\/Y0e2Ahm91I","source":"\u003ca href=\"https:\/\/about.twitter.com\/products\/tweetdeck\" rel=\"nofollow\"\u003eTweetDeck\u003c\/
a
\u003e","truncated":false,"in_reply_to_status_id":null,"in_reply_to_status_id_str":null,"in_reply_to_user_id":null,"in_reply_to_user_id_str":null,"in_reply_to_scr
een_name":null,"user":{"id":6253282,"id_str":"6253282","name":"Twitter API","screen_name":"twitterapi","location":"San Francisco, CA","description":"The Real
Twitter API. I tweet about API changes, service issues and happily answer questions about Twitter and our API. Don't get an answer? It's on my
website.","url":"http:\/\/t.co\/78pYTvWfJd","entities":{"url":{"urls":[{"url":"http:\/\/t.co\/78pYTvWfJd","expanded_url":"http:\/\/
dev.twitter.com","display_url":"dev.twitter.com","indices":[0,22]}]},"description":{"urls":[]}},"protected":false,"followers_count":2091868,"friends_count":
46,"listed_count":12513,"created_at":"Wed May 23 06:01:13 +0000 2007","favourites_count":27,"utc_offset":-25200,"time_zone":"Pacific Time (US &
Canada)","geo_enabled":true,"verified":true,"statuses_count":3481,"media_count":
3,"lang":"en","contributors_enabled":false,"is_translator":false,"is_translation_enabled":false,"profile_background_color":"C0DEED","profile_background_image_url"
:"http:\/\/pbs.twimg.com\/profile_background_images\/656927849\/miyt9dpjz77sc0w3d4vj.png","profile_background_image_url_https":"https:\/\/pbs.twimg.com\/
profile_background_images\/656927849\/miyt9dpjz77sc0w3d4vj.png","profile_background_tile":true,"profile_image_url":"http:\/\/pbs.twimg.com\/profile_images\/
2284174872\/7df3h38zabcvjylnyfe3_normal.png","profile_image_url_https":"https:\/\/pbs.twimg.com\/profile_images\/2284174872\/
7df3h38zabcvjylnyfe3_normal.png","profile_banner_url":"https:\/\/pbs.twimg.com\/profile_banners\/6253282\/
1347394302","profile_link_color":"0084B4","profile_sidebar_border_color":"C0DEED","profile_sidebar_fill_color":"DDEEF6","profile_text_color":"333333","profile_use
_background_image":true,"default_profile":false,"default_profile_image":false,"following":false,"follow_request_sent":false,"notifications":false},"geo":null,"coo
rdinates":null,"place":null,"contributors":null,"retweeted_status":{"created_at":"Thu Feb 27 18:05:02 +0000 2014","id":
439098866919407617,"id_str":"439098866919407617","text":"Announcing Crashlytics Labs Project: Beta Distribution http:\/\/t.co\/JifugCbtU2 #androiddev #iosdev
http:\/\/t.co\/
Y0e2Ahm91I","source":"web","truncated":false,"in_reply_to_status_id":null,"in_reply_to_status_id_str":null,"in_reply_to_user_id":null,"in_reply_to_user_id_str":nu
ll,"in_reply_to_screen_name":null,"user":{"id":241762251,"id_str":"241762251","name":"Crashlytics","screen_name":"crashlytics","location":"Boston,
MA","description":"Introducing the world's most powerful, yet lightest weight crash reporting solution.","url":"http:\/\/t.co\/0YVEfq97cL","entities":{"url":
```

Storing tweet metadata

Forward index

- Scale: We have > 300B tweets



Feature representation

Feature representation

Feature representation

Engagement
1
2
3
4
5
6
7
8
9
10
20
30
40
50
100
200
300
400
500
1000
5000
10000
100000

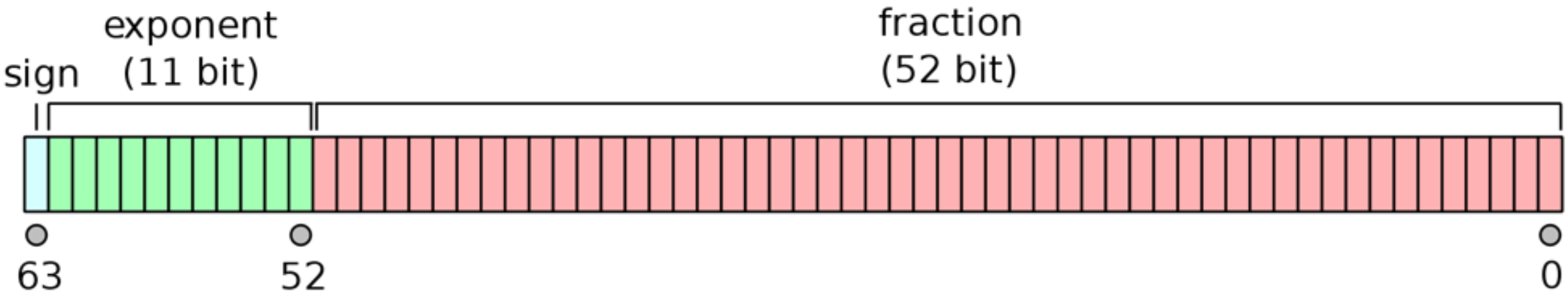
Feature representation

Engagement	Log2(E) + 1
1	1
2	2
3	3
4	3
5	3
6	4
7	4
8	4
9	4
10	4
20	5
30	6
40	6
50	7
100	8
200	9
300	9
400	10
500	10
1000	11
5000	13
10000	14
100000	18

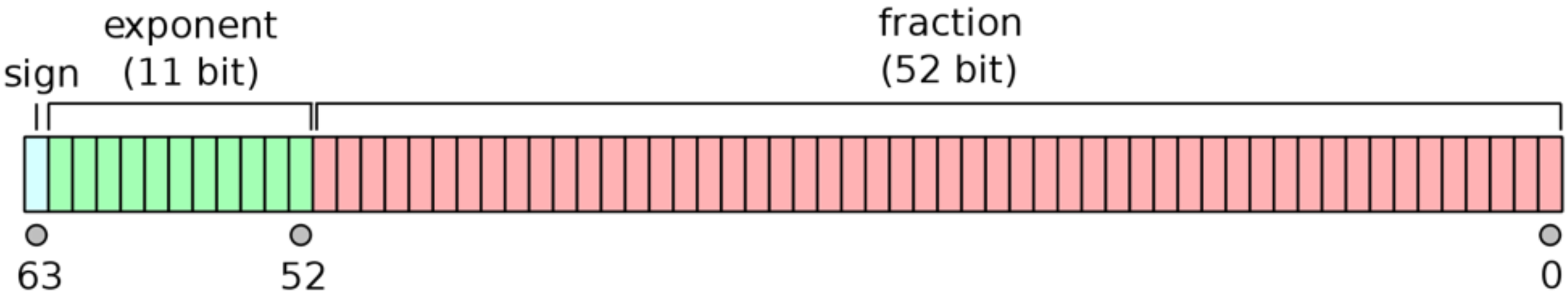
Feature representation

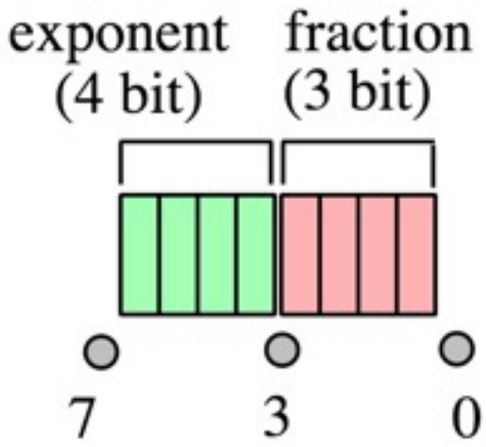
Engagement	Log2(E) + 1	Log1.1(E) + 1
1	1	1
2	2	8
3	3	13
4	3	16
5	3	18
6	4	20
7	4	21
8	4	23
9	4	24
10	4	25
20	5	32
30	6	37
40	6	40
50	7	42
100	8	49
200	9	57
300	9	61
400	10	64
500	10	66
1000	11	73
5000	13	90
10000	14	98
100000	18	122

Feature representation



Feature representation



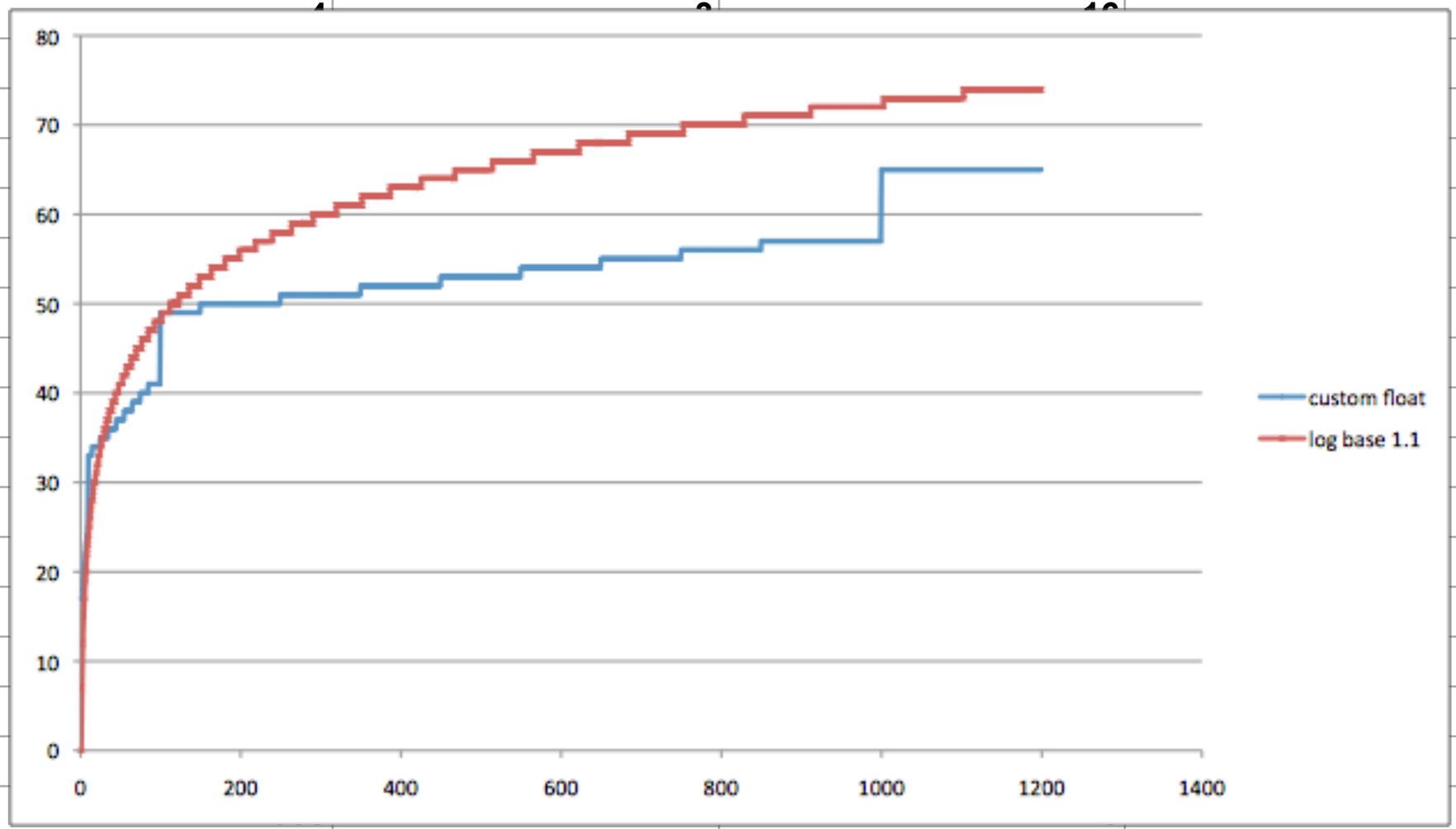


Feature representation

Engagement	Log2(E) + 1	Log1.1(E) + 1	Custom float
1	1	1	33
2	2	8	34
3	3	13	35
4	3	16	36
5	3	18	37
6	4	20	38
7	4	21	39
8	4	23	40
9	4	24	41
10	4	25	49
20	5	32	50
30	6	37	51
40	6	40	52
50	7	42	53
100	8	49	65
200	9	57	66
300	9	61	67
400	10	64	68
500	10	66	69
1000	11	73	81
5000	13	90	85
10000	14	98	97
100000	18	122	113

Feature representation

Engagement	Log2(E) + 1	Log1.1(E) + 1	Custom float
1	1	1	33
2	2	8	34
3	3	13	35
			36
			37
			38
			39
			40
			41
			49
			50
			51
			52
			53
			65
			66
			67
			68
			69
			81
5000	13	90	85
10000	14	98	97
100000	18	122	113



Relevance ranking

- Task: Find the best tweets in the time-sorted index

Relevance ranking

- Features
  - Query-dependent
    - E.g. Lucene text score, language
  - Query-independent
    - Static signals (e.g. text quality)
    - Dynamic signals (e.g. retweets)

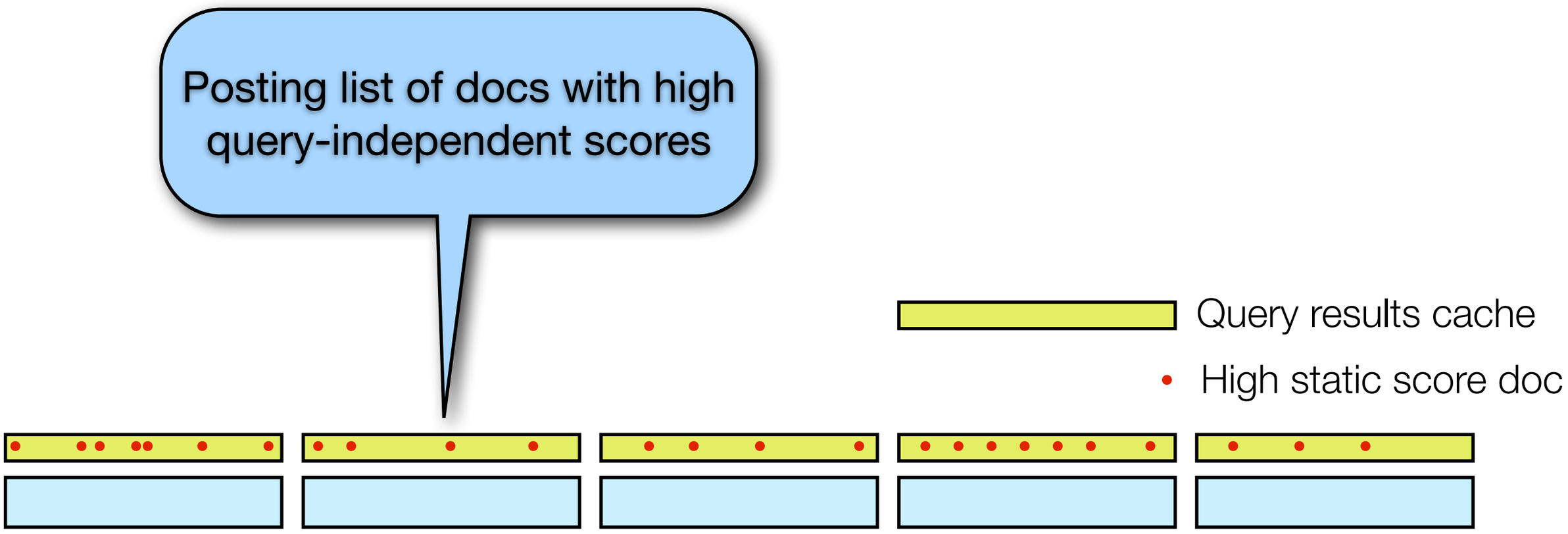
Relevance ranking

- Task: Find the best tweets in the time-sorted index
- We could sort the index by query-independent scores
  - Hard to achieve in RT index
  - Not all queries use relevance ranking
- Idea: Skip lists for documents with high query-independent scores

Relevance ranking

Many Earlybird segments (8M documents each)

Relevance ranking



Relevance ranking

```
filterName: toptweets1
params:
- retweetWeight: 0.33
- faveWeight: 0.50
resultType: BitSet
cacheModeOnly: true
schedule:
- {segment: 0, seconds: 60}
- {segment: 1, seconds: 600}
```

Relevance ranking

```
filterName: toptweets1
params:
- retweetWeight: 0.33
- faveWeight: 0.50
resultType: BitSet
cacheModeOnly: true
schedule:
- {segment: 0, seconds: 60}
- {segment: 1, seconds: 600}
```

Different internal representation for sparse vs dense lists

Relevance ranking

```
filterName: toptweets1
params:
- retweetWeight: 0.33
- faveWeight: 0.50
resultType: BitSet
cacheModeOnly: true
schedule:
- {segment: 0, seconds: 60}
- {segment: 1, seconds: 600}
```

Different internal representation for sparse vs dense lists

Recompute for recent content?

Relevance ranking

```
filterName: toptweets1
params:
- retweetWeight: 0.33
- faveWeight: 0.50
resultType: BitSet
cacheModeOnly: true
schedule:
- {segment: 0, seconds: 60}
- {segment: 1, seconds: 600}
```

Different internal representation for sparse vs dense lists

Recompute for recent content?

Older content may not need frequent updates

Relevance ranking

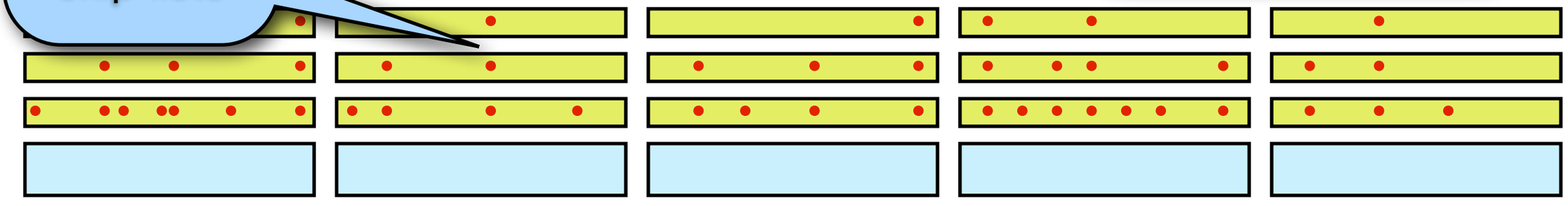
```
filterName: toptweets1
params:
- retweetWeight: 0.33
- faveWeight: 0.50
resultType: BitSet
cacheModeOnly: true
schedule:
- {segment: 0, seconds: 60}
- {segment: 1, seconds: 600}
```

Different internal representation for sparse vs dense lists

Recompute for recent content?

Older content may not need frequent updates

Hierarchical skip lists

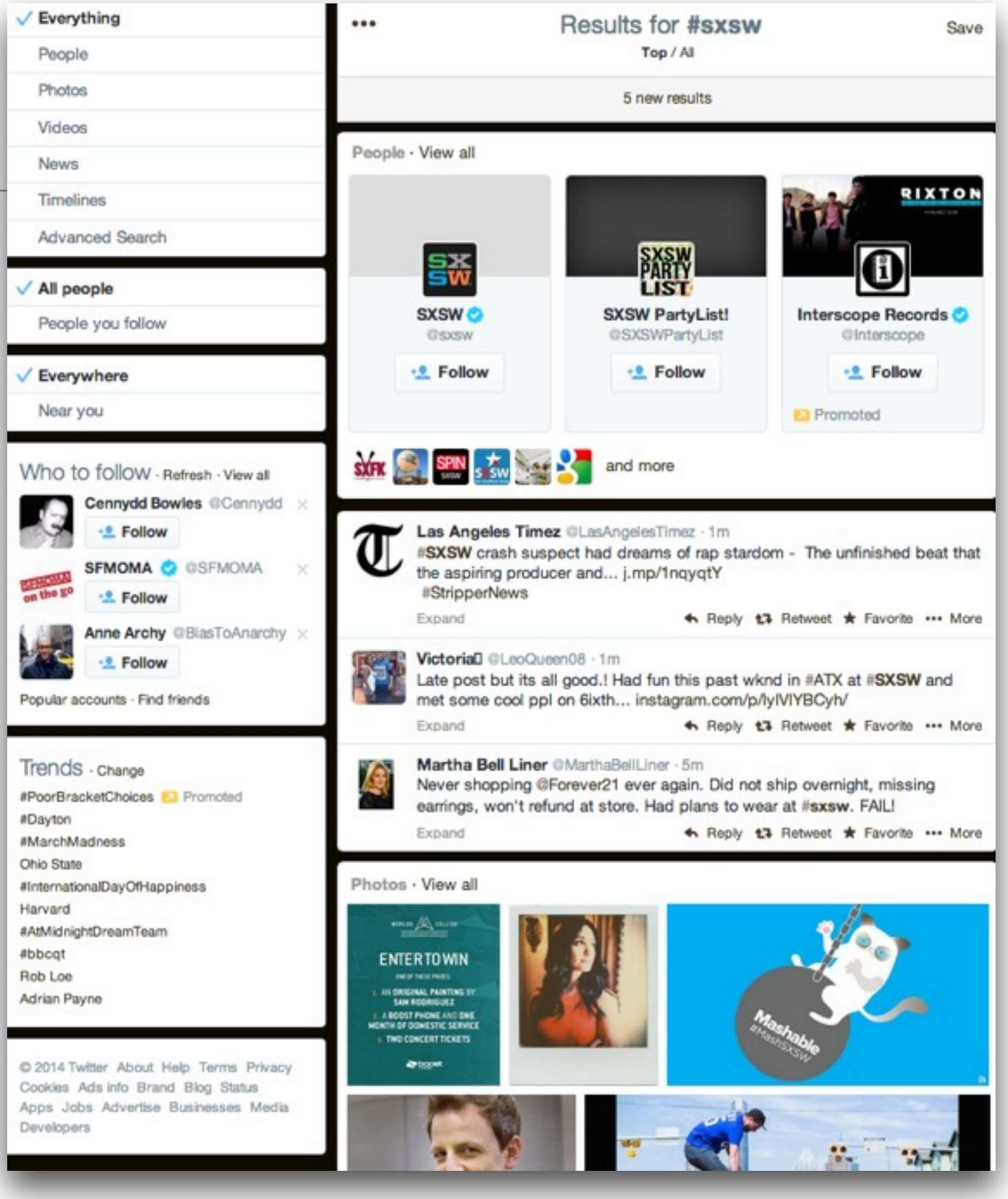


Many Earlybird segments (8M documents each)

Relevance ranking - Summary

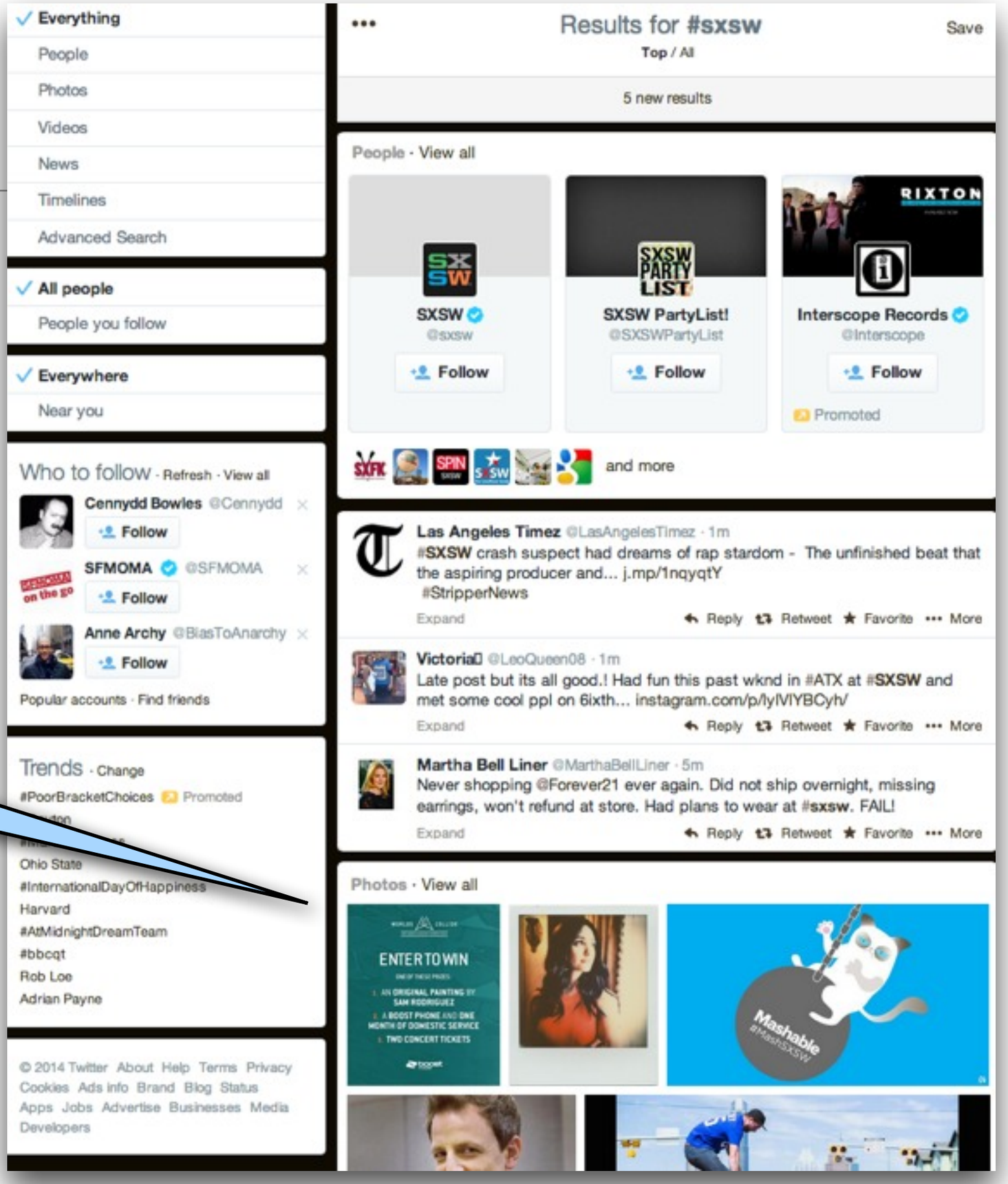
- RT index ordered by time; new tweets can simply be appended
- Forward index updated regularly with engagement features
- Background thread regularly recomputes query-independent toptweet skiplists
- High performance achieved with combination of skip lists and early termination

Universal Search



Universal Search

Photos



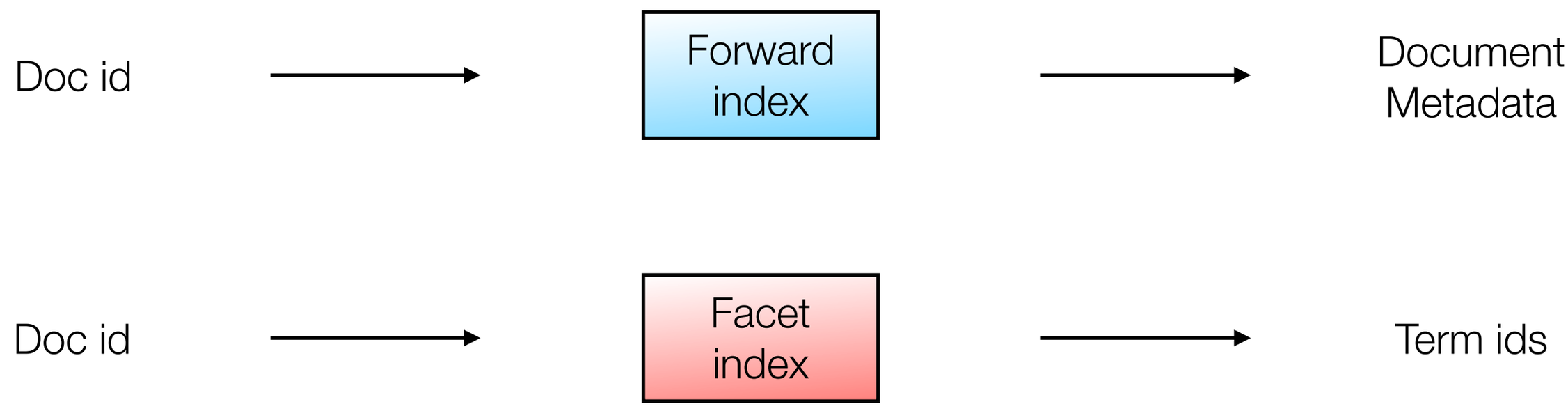
Searching for top entities within Tweets

- Task: Find the best photos in a subset of tweets
- We could use a Lucene index, where each photo is a document
- Problem: How to update existing documents when the same photos are tweeted again?
- In-place posting list updates are hard
- Lucene’s updateDocument() is a delete/add operation - expensive and not order-preserving

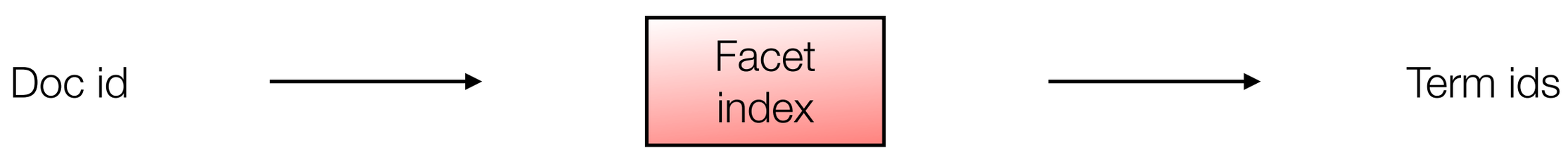
Searching for top entities within Tweets

- Task: Find the best photos in a subset of tweets
  - Could we use our existing time-ordered tweet index?
  - Facets!

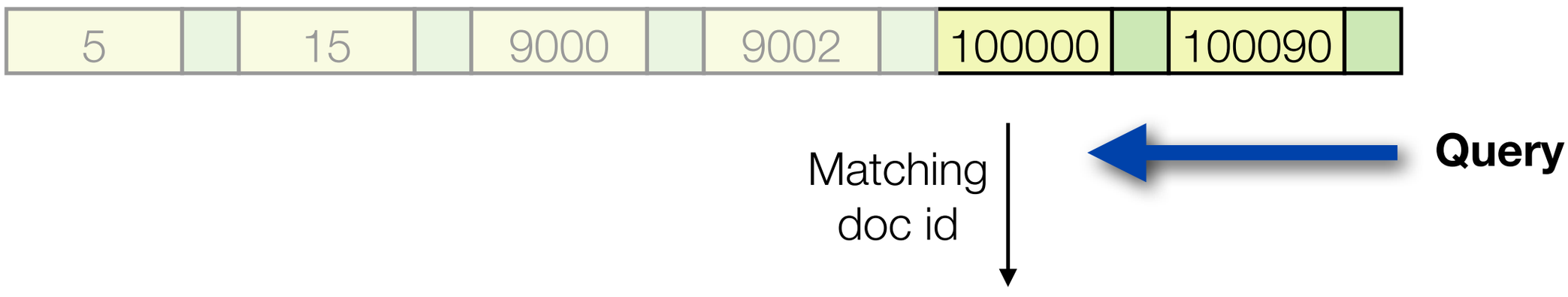
Searching for top entities within Tweets

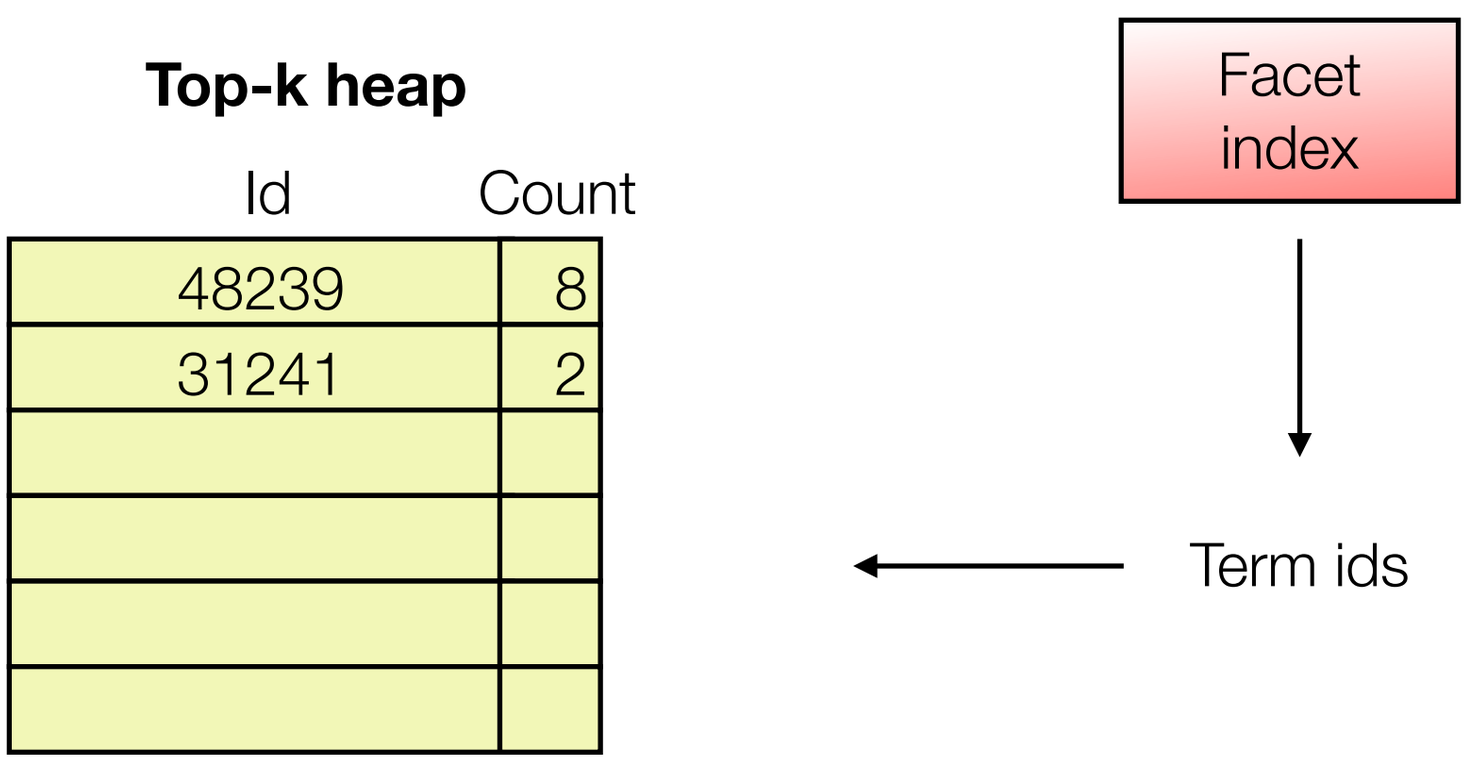


Storing tweet metadata

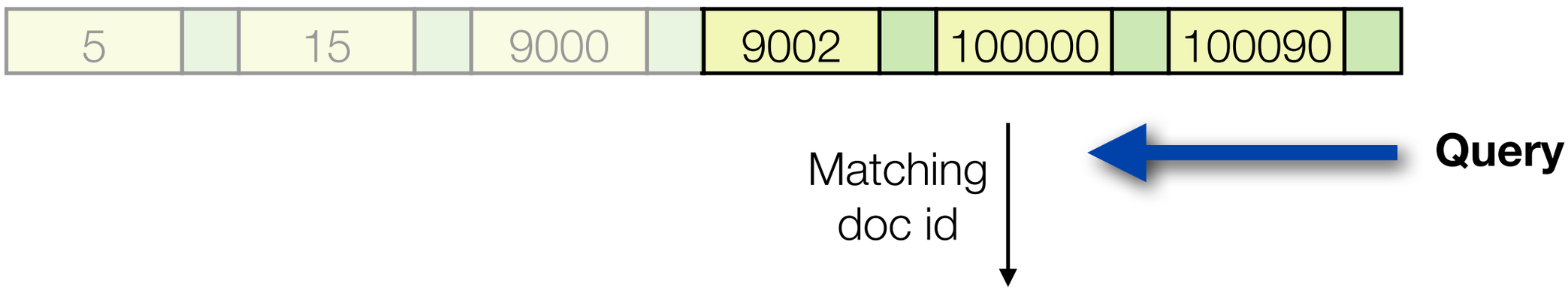


Searching for top entities within Tweets



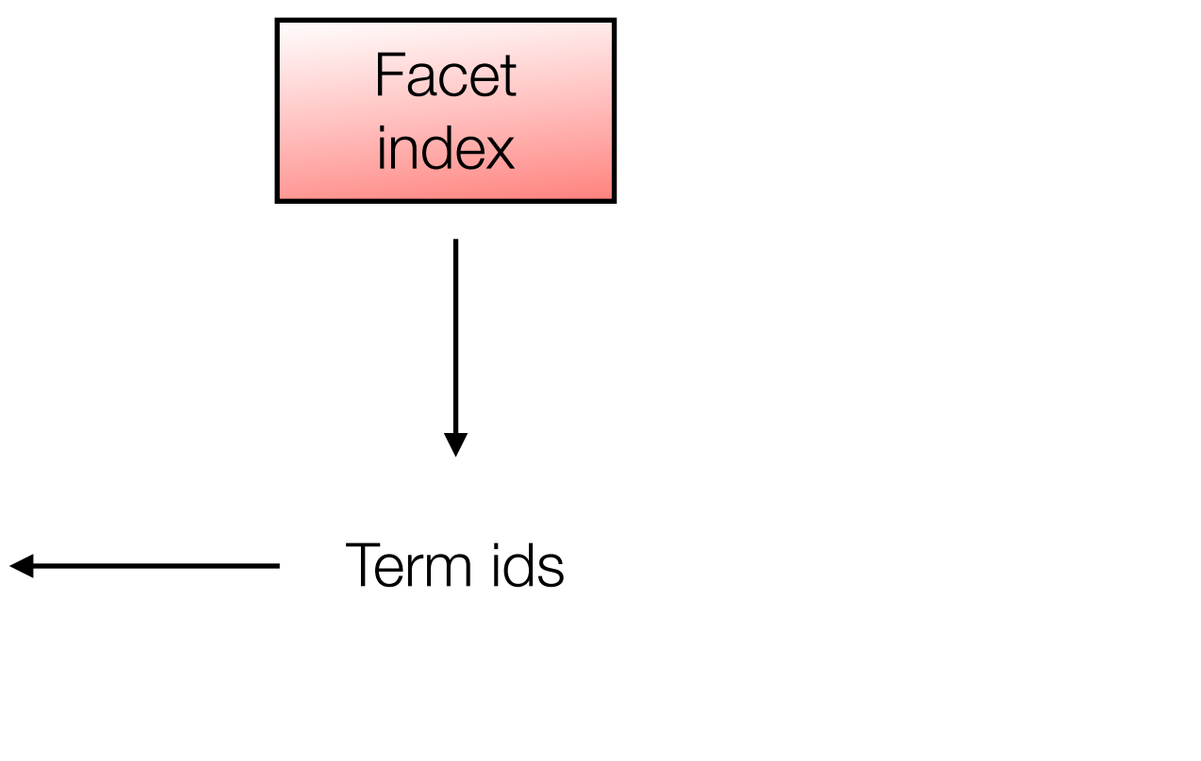


Searching for top entities within Tweets

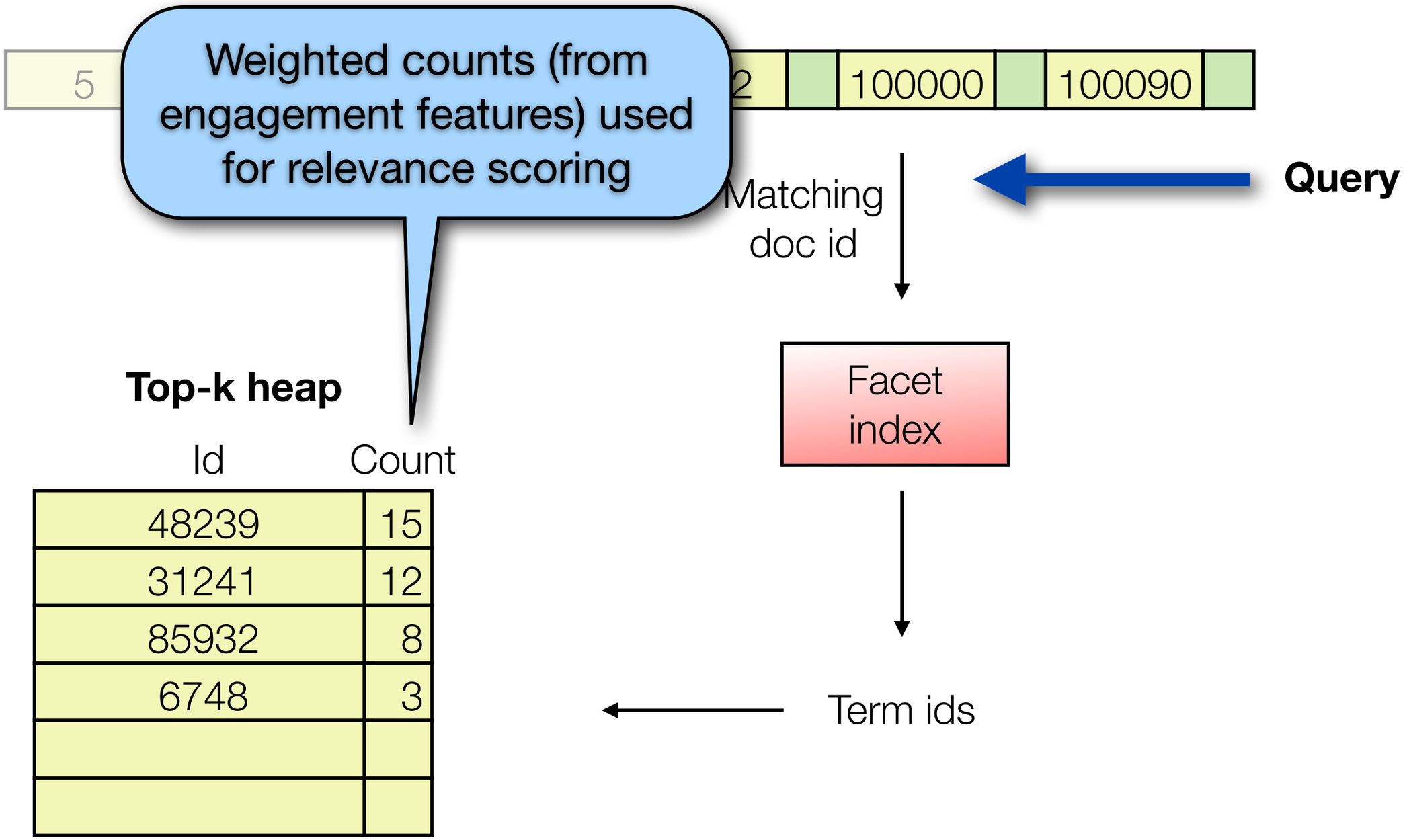


Top-k heap

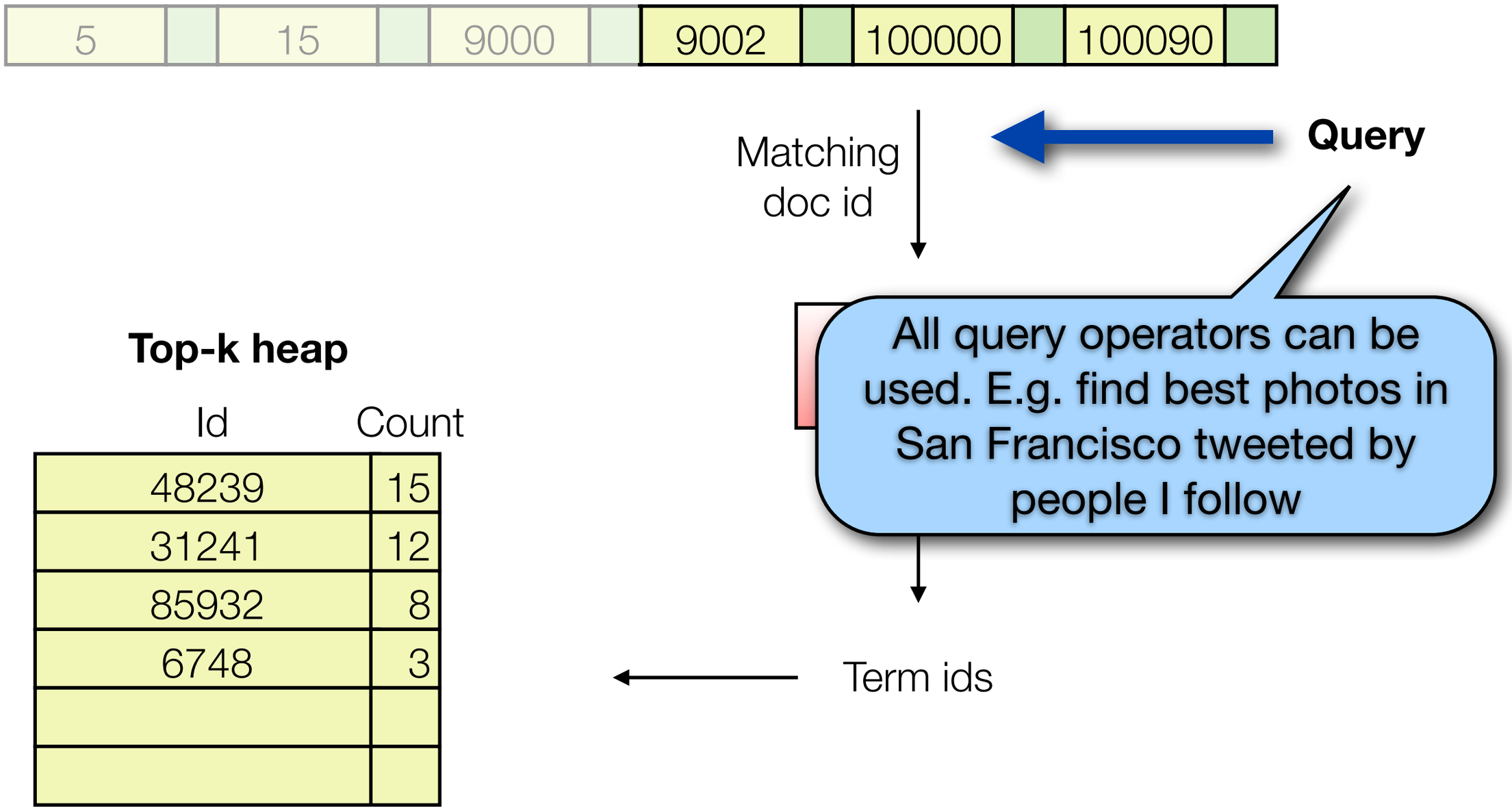
Id	Count
48239	15
31241	12
85932	8
6748	3



Searching for top entities within Tweets



Searching for top entities within Tweets



Searching for top entities within Tweets

Searching for top entities within Tweets

Id	Count
48239	45
31241	23
85932	15
6748	11
74294	8
3728	5

Label	Count
pic.twitter.com/jknui4w	45
pic.twitter.com/dslkfj83	23
pic.twitter.com/acm3ps	15
pic.twitter.com/948jdsd	11
pic.twitter.com/dsjkf15h	8
pic.twitter.com/irnsoa32	5

Summary

- Indexing tweet entities (e.g. photos) as facets allows to search and rank top-entities using a tweets index

- All query operators supported

- Documents don’t need to be reindexed

- Approach reusable for different use cases, e.g.: best vines, hashtags, @mentions, etc.

Questions?

Michael Busch
@michibusch
michael@twitter.com
buschmi@apache.org