**11-442 / 11-642:**
**Search Engines**

# Overview of the QryEval Software

Jamie Callan
Carnegie Mellon University
callan@cs.cmu.edu

## Outline

- **QryEval overview**
- **Query evaluation**
  - The Qry class
  - Iteration
  - Matching
  - Calculating scores
- **Overview of query parsing**

## QryEval

**QryEval is an application that conducts experiments**

- Read a parameter file
  - Example parameter file for HW1

```
indexPath=someDirectory/index
retrievalAlgorithm=UnrankedBoolean
queryFilePath=queries.txt
trecEvalOutputPath=HW1-queries-UB.teIn
```

  - Each homework will have additional parameters




## QryEval

**QryEval is an application that conducts experiments**

- Read a parameter file
- Read a query file (one query per line)

```
10:#OR(cheap internet)
26:#AND(lower heart rate)
71:living in india
```

## QryEval

**QryEval is an application that conducts experiments**

- Read a parameter file
- Read a query file (one query per line)
  - Parse the query

```
Qry q = QryParser.getQuery (
            "#or(living in india)" );
```

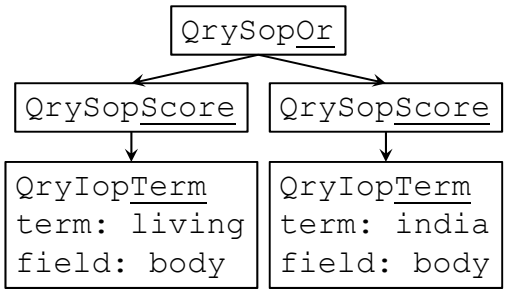
  - More on query parsing later in the lecture …




## QryEval

**QryEval is an application that conducts experiments**

- Read a parameter file
- Read a query file (one query per line)
  - Parse the query




**Qry class**
An abstract class
for all query operators

**QrySopXxx subclasses**
Sop: Score operator
Calculate a score

**QryIopXxx subclasses**
Iop: Inverted list operator
Get an inverted list

## QryEval

**QryEval is an application that conducts experiments**

- Read a parameter file
- Read a query file (one query per line)
  - Parse the query
  - Evaluate the query, using a DAAT architecture

```
q.initialize (model);    // E.g., UnrankedBoolean

while (q.docIteratorHasMatch (model)) {
  int docid = q.docIteratorGetMatch ();
  double score = ((QrySop) q).getScore (model);
  result.add (docid, score);
  q.docIteratorAdvancePast (docid);
}
```




## QryEval

**QryEval is an application that conducts experiments**

- Read a parameter file
- Read a query file (one query per line)
  - Parse the query
  - Evaluate the query
  - Write the results for the query to a file

```
11 Q0 GX270-76-5299838 1 0.0099338 HW1a
11 Q0 GX000-72-8784276 2 0.0097739 HW1a
11 Q0 GX000-25-2008761 3 0.0096334 HW1a
```

**Query Id** **Always Q0** **Doc External Id** **Doc Rank** **Doc Score** **Your Choice**

## Outline

- QryEval overview
- **Query evaluation**
  - The Qry class
  - Iteration
  - Matching
  - Calculating scores
- Overview of query parsing

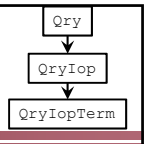



## Qry

**Qry is the base class for all query operators**

- Data that every query operator has (e.g., query arguments)
- Methods that work for all query operators (e.g., appendArg)
- Methods that each query operator must define (e.g., HasMatch)
- It has two subclasses: QryIop and QrySop
  - QrySop: The base class for operators that return scores
    - » SCORE, AND, OR, SUM, WAND, WSUM, …
  - QryIop: The base class for operators that return inverted lists
    - » TERM, NEAR, WINDOW

## Parts of an Object May Be Defined By Different Parts of the Class Hierarchy




**QryIopTerm object:** An object for storing query terms

```
String displayName;
ArrayList<Qry> args;
boolean matchStored;
int matchingDocid;
-----------------------------------
InvList invertedList;
int docIteratorIndex;
Int locIteratorIndex;
String field;
-----------------------------------
String term;
```

**Defined by Qry**

**Defined by QryIop**

**Defined by QryIopTerm**

**Defined in multiple places, but stored in one place … the object**

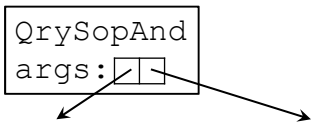
- Each object has its own args, invertedList, term string, …




## Qry

**Qry is the base class for all query operators**

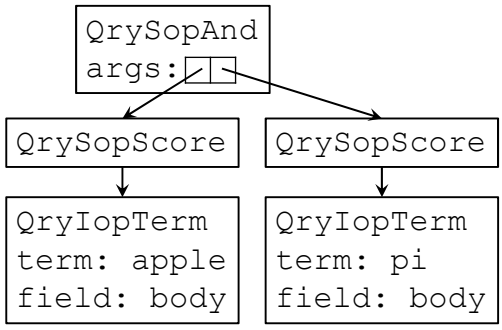
- It defines a place to store query operator arguments & other data
  - Conceptually: #AND (apple pi)
  - Actually:



## Qry

**Qry is the base class for all query operators**

- It controls how arguments are appended to the query operator
  - E.g., automatically insert a #SCORE operator between a scoring (Sop) and an inverted list (Iop) operator




13



## Qry

**Qry is the base class for all query operators**

- It defines docIterators to iterate over matching documents

```
docIteratorHasMatch     // Each subclass defines
      :     :     :
docIteratorGetMatch     // Get from HasMatch cache
docIteratorAdvancePast  // Advance past a docid
docIteratorAdvanceTo    // Advance to a docid
```

  - These are not Java-style iterators

14

## Qry

**Qry is the base class for all query operators**

- It defines docIterators to iterate over matching documents

```
docIteratorHasMatch     // Each subclass defines
```

  - There are a few standard matching styles that are implemented as utility methods

```
docIteratorHasMatchAll  // Matches all args
docIteratorHasMatchMin  // Matches min docid
```

  - When you implement `docIteratorHasMatch` for a new [query operator, retrieval model] pair, consider whether one of the standard utility methods meets your needs
    - » E.g., `docIteratorHasMatchMin` for an OR operator




## Iteration

**Query evaluation is divided into three parts**

1. Get all inverted lists
2. Get the docid of the next document that matches the query
3. Get the score of docid (which must be a matching document)

## Iteration

**Query evaluation is divided into three parts**

1. Get all inverted lists
   - Done during query initialization
   - There are two ways of obtaining inverted lists
     - » Read from disk, e.g., "apple"
     - » Construct dynamically, e.g., "#NEAR/3 (lady gaga)"
   - Assumption: Everything fits into RAM
     - » This is a simple system for homework
     - » A production system might process inverted lists in blocks to control memory usage




## Iteration

**Query evaluation is divided into three parts**

1. Get all inverted lists
2. Get the docid of the next document that matches the query
   - Iterate over (actual) inverted lists and (virtual) score lists
   - There are only a few matching strategies
     - » any query argument matches ("union")
     - » all query arguments match ("intersection")
   - The retrieval model determines what is considered a match
     - » E.g., Ranked Boolean: AND must match all arguments
     - » E.g., Indri: AND must match at least one argument

## Iteration

**Query evaluation is divided into three parts**

1. Get all inverted lists
2. Get the docid of the next document that matches the query
3. Get the score of docid (which must be a matching document)
   – The retrieval model determines how the score is calculated




## Iteration

**The retrieval model determines how a query operator iterates**

- Ranked Boolean AND does it differently from an Indri AND

**However, there are a few "typical" styles of iteration**

- E.g., HasMatchFirst, HasMatchAll, HasMatchMin
- These are defined in Qry.java
- Often individual query operators just call one of the standard methods
  – Before you implement something, consider whether a standard method will meet your needs

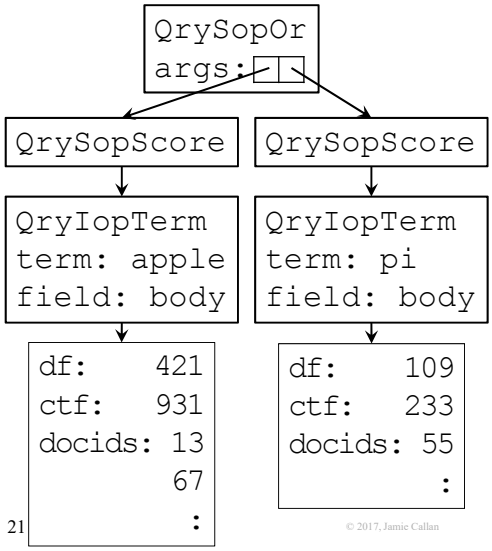
## Method QrySopOr.docIteratorHasMatch

**The OR operator matches if any argument matches**

**It uses docIteratorHasMatchMin**

- Iterate over the arguments
- Ask each argument to return its current docid
- Return the minimum docid

**Recursion handles complex queries naturally**







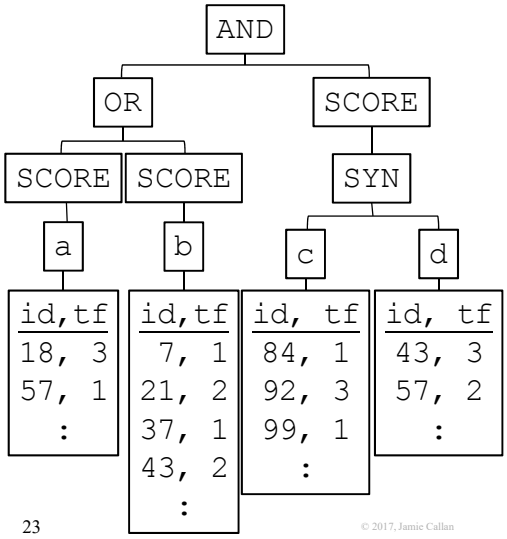
## Caching

**When a docIteratorHasMatch matches a document, it caches the docid to improve efficiency**

- docIteratorGetMatch reads the docid from the cache
- Why not have HasMatch just return the docid?
  - If there is no match, it would need to return an invalid docid or throw an exception
  - Those seem messier to me

## Matching and Scoring

**What is the first document that matches this query?**



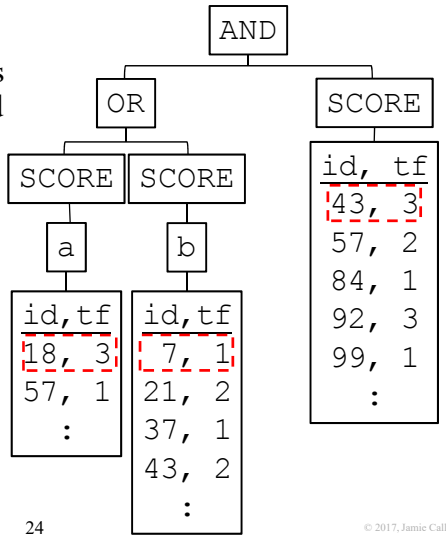

23



## Matching and Scoring

**Call q.initialize()**

- Inverted list query operators (e.g., SYN) are materialized
  - Converted to inverted lists
- Iterators are initialized

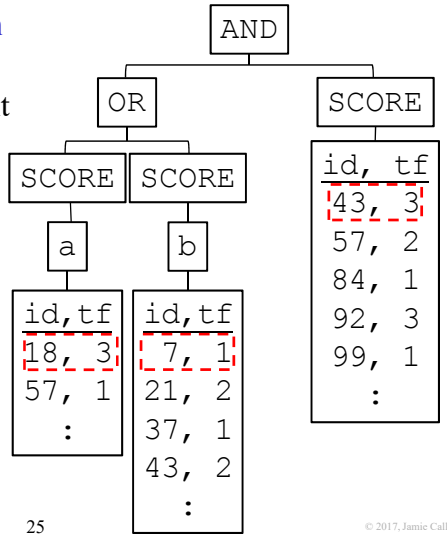



24

## Matching and Scoring

**Call q.docIteratorHasMatch on the AND operator**

- The query is structured, so it passes the request to its children
- The children are query operators, so they pass it to their children
- Eventually the requests reach inverted lists



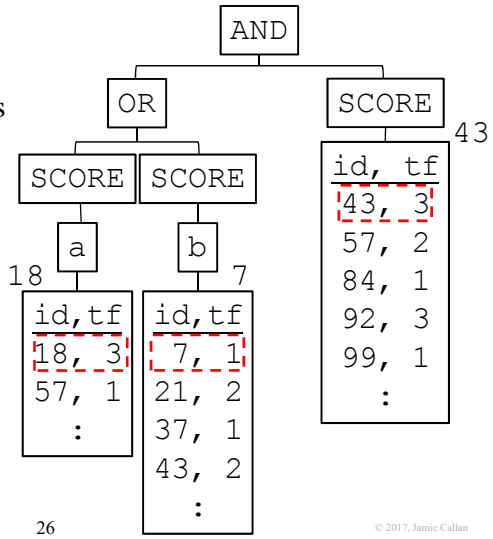





## Matching and Scoring

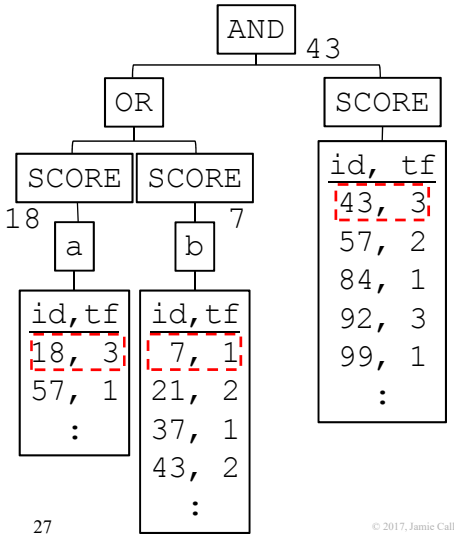
**The inverted lists return their next matching docid**

- The ids get passed upwards



## Matching and Scoring

**Each operator calculates its next matching document**

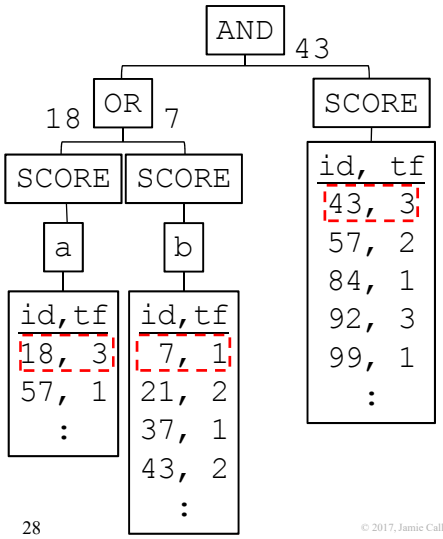



27



## Matching and Scoring

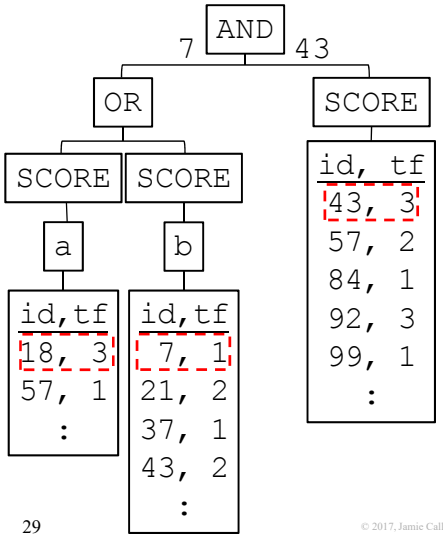
**Each operator calculates its next matching document**




28

## Matching and Scoring

**Each operator calculates its next matching document**



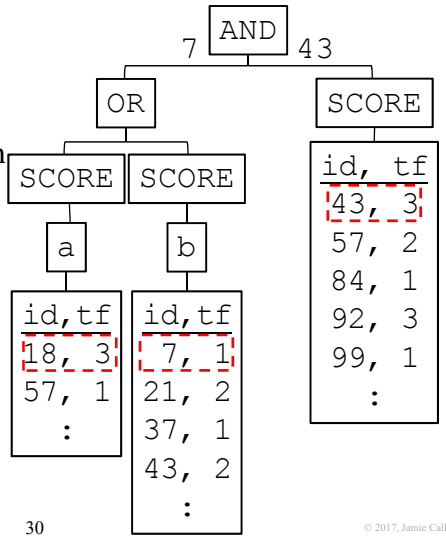

29



## Matching and Scoring

**Each operator calculates its next matching document**

- AND doesn't have a match
- It knows that the next match must have docid ≥ 43
- It tells all children to advance their iterators to docid 43 (or the next docid after 43)

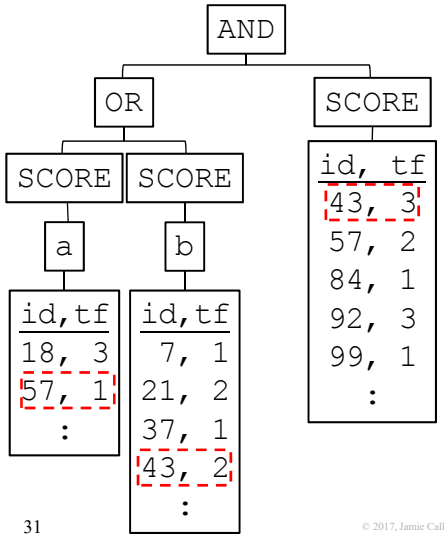



30

## Matching and Scoring

**AND polls its children again (this is a while loop)**

- The children are query operators, so they pass it to their children
- Eventually the requests reach inverted lists



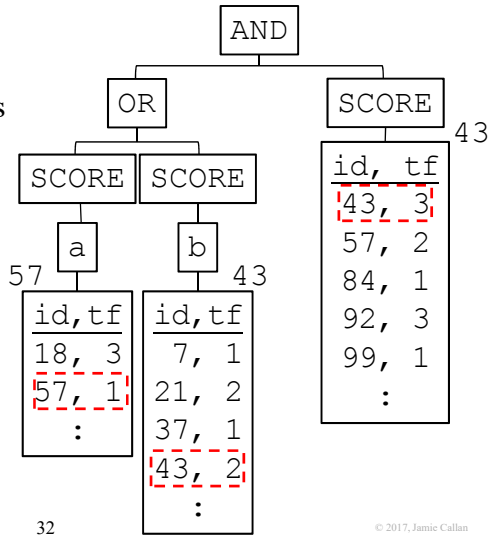





## Matching and Scoring

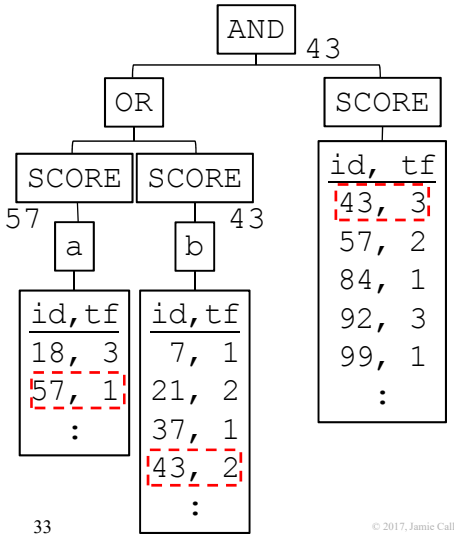
**The inverted lists return their next matching docid**

- The ids get passed upwards



## Matching and Scoring

**Each operator calculates its next matching document**

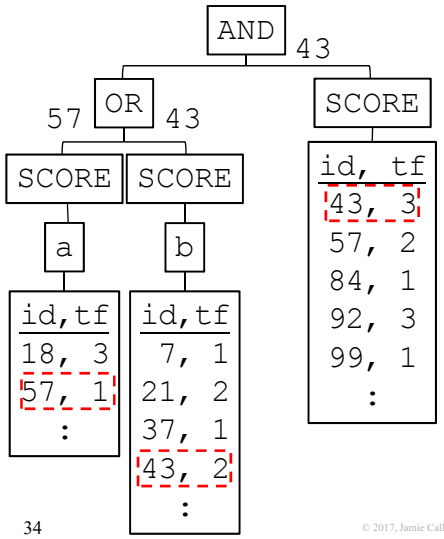



33



## Matching and Scoring

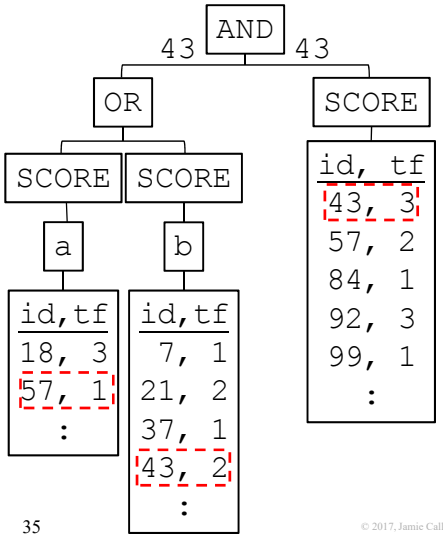
**Each operator calculates its next matching document**




34

## Matching and Scoring

**Each operator calculates its next matching document**

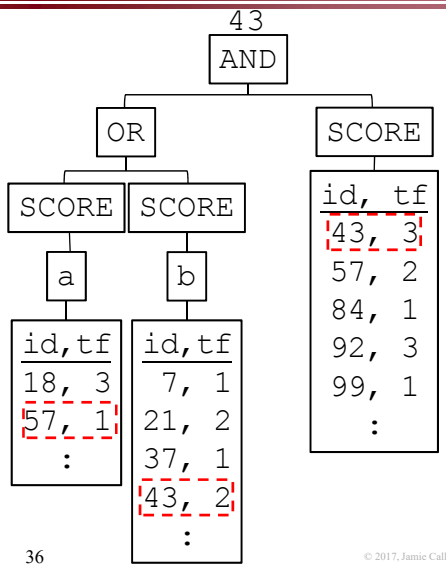



35



## Matching and Scoring

**Each operator calculates its next matching document**

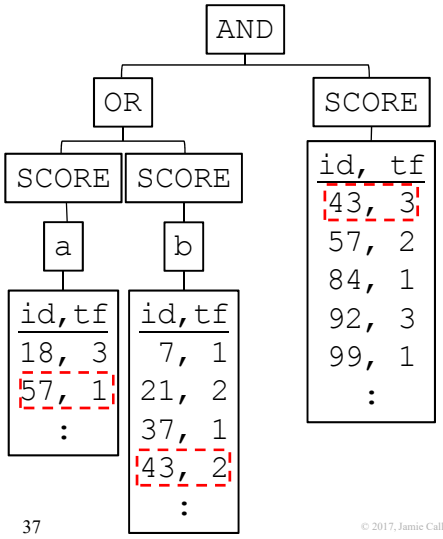



36

## Matching and Scoring

**Call q.getScore on the AND operator**

- AND has cached the fact that docid 43 matches
- The query is structured, so AND passes the request to its children
- The children are query operators, so they pass it to their children
- Eventually the requests reach SCORE operators



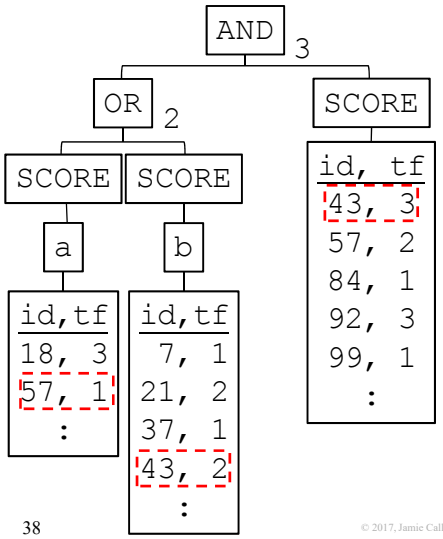

37



## Matching and Scoring

**The SCORE operators return scores**

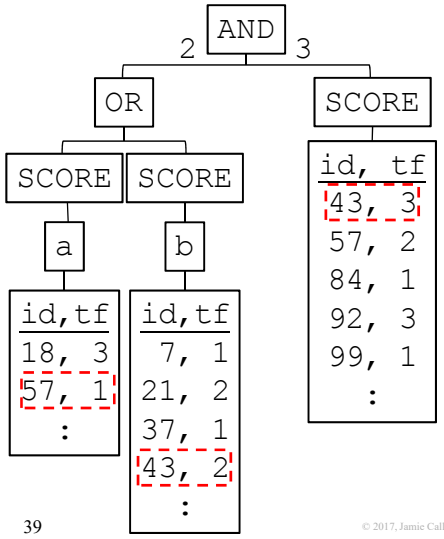
- Assume ranked Boolean




38

## Matching and Scoring

**The OR operator combines scores**

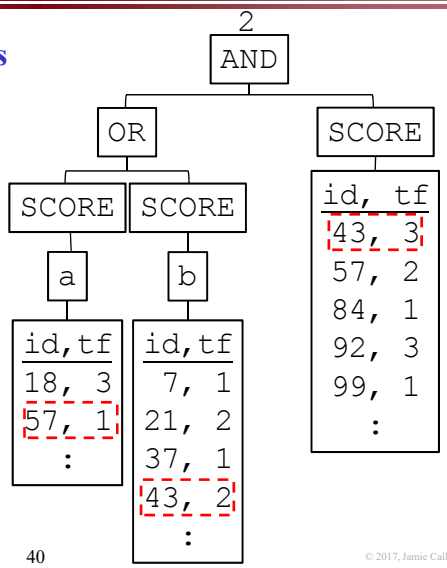



39



## Matching and Scoring

**The AND operator combines scores**

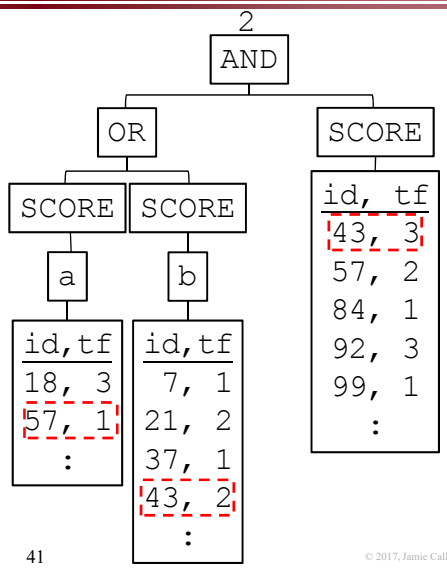



40

## Matching and Scoring

**Easy!**

**Repeat for the next match**




41



## Calculating Scores

**Each score operator (QrySopXxx) implements getScore (RetrievalModel r)**

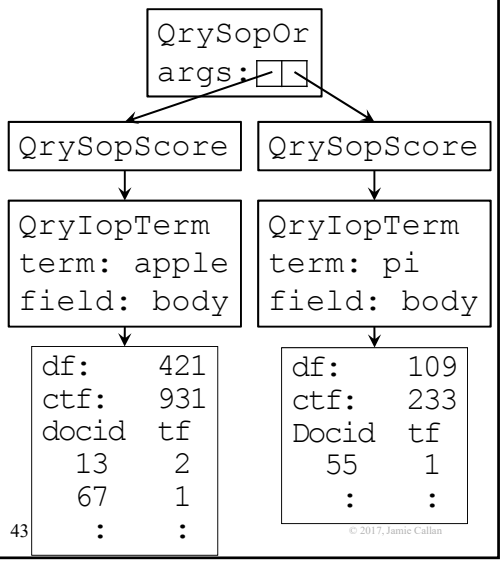
- Traverse the query to calculate a score for the current docid
  - We know that it matches, so just calculate a score
- The retrieval model tells the operator what strategy to use for calculating the score
  - For HW1, RankedBoolean and UnrankedBoolean
  - For HW2, BM25 and Indri
  - Retrieval models for HW2 will also store parameters

42

## Method QrySopOr.getScore

**If the model is unranked Boolean, return a 1**

**If the model is ranked Boolean, what is returned?**







## Qry Class Summary

**The Qry class implements DAAT scoring**

- Iterate over (actual) inverted lists and (virtual) score lists
- Several general ways to match a query operator to a document
  - Match all arguments, any argument, …
- Allows you to add different ways to calculate document scores
  - Unranked boolean, ranked boolean, …
- Much use of inheritance and recursion
  - Minimizes redundant implementation ☺
  - Requires greater understanding ☹

## Outline

- QryEval overview
- Query evaluation
  - The Qry class
  - Iteration
  - Matching
  - Calculating scores
- **Overview of query parsing**




## Query Parsing

**QryParser is a simple query parser**

```
Qry q = QryParser.getQuery (
            "#or(living in india)" );
```

## Query Parsing

**Pop the operator from the string**

- Leftmost token, starts with '#'
- Create the operator (e.g., QrySopOr)

```
#OR(a b c)

#AND(a #OR(b c) d)
```

**Find the list of query arguments**

- Delimited by leftmost '(' and its matching ')'
- For each argument
  - If it is not a stopword
    - » If it is a term, create a term object (i.e., QryIopTerm)
    - » Otherwise, recursively call the query parser
    - » Add the result (a Qry object) to the operator argument list
      - A #SCORE operator may be inserted automatically




## Query Parsing

```
#AND(a b c)
```

## Query Parsing

**#AND(a b c)**

**operator**

**Allocate an AND operator.**

```
QrySopAnd
args:
```





## Query Parsing

**#AND(a b c)**

**Left '(' and right ')'**

```
QrySopAnd
args:
```

## Query Parsing

**#AND(a b c)**

**arguments**

QrySopAnd
args:

51

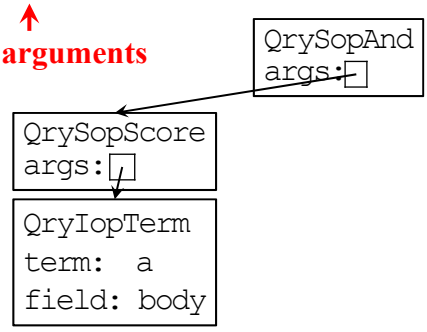


## Query Parsing

**#AND(a b c)**

**arguments**

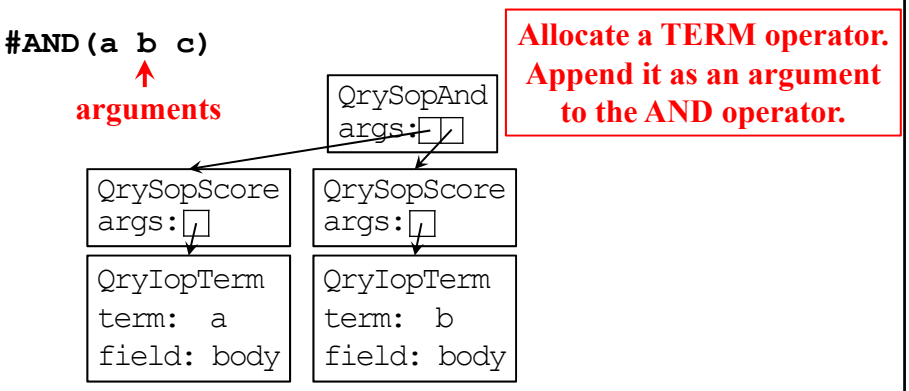
**Allocate a TERM operator. Append it as an argument to the AND operator.**




**Note:** Qry.appendArg automatically inserts #SCORE operators

52

## Query Parsing

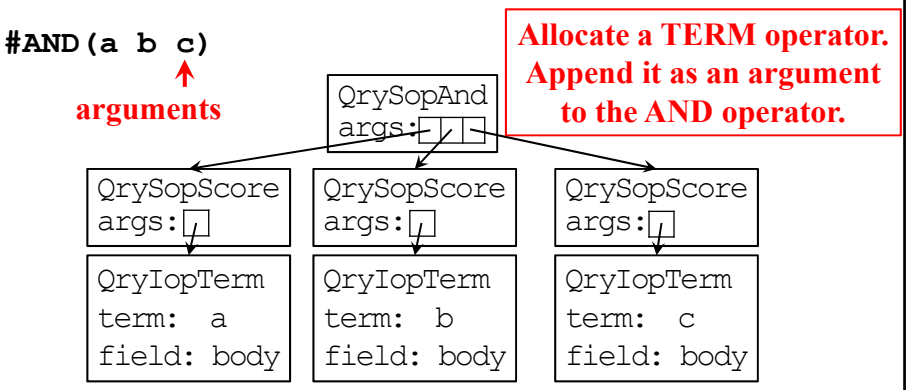



**Note:** Qry.appendArg automatically inserts #SCORE operators

53



## Query Parsing




**Note:** Qry.appendArg automatically inserts #SCORE operators

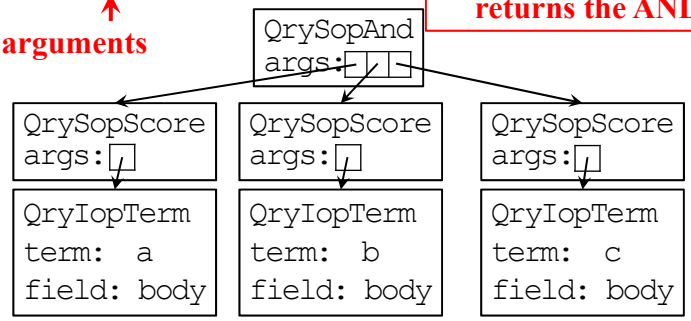
54

## Query Parsing

**#AND(a b c)**

**arguments**

**Query parsing completes & returns the AND node.**








## Query Parsing

**#AND(a #OR(b c) d)**

## Query Parsing

```
#AND(a #OR(b c) d)
```

operator

Allocate an AND operator.

```
QrySopAnd
args:
```

57



## Query Parsing

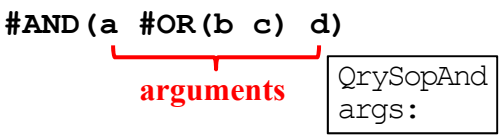
```
#AND(a #OR(b c) d)
```

Left '(' and right ')'

```
QrySopAnd
args:
```

58

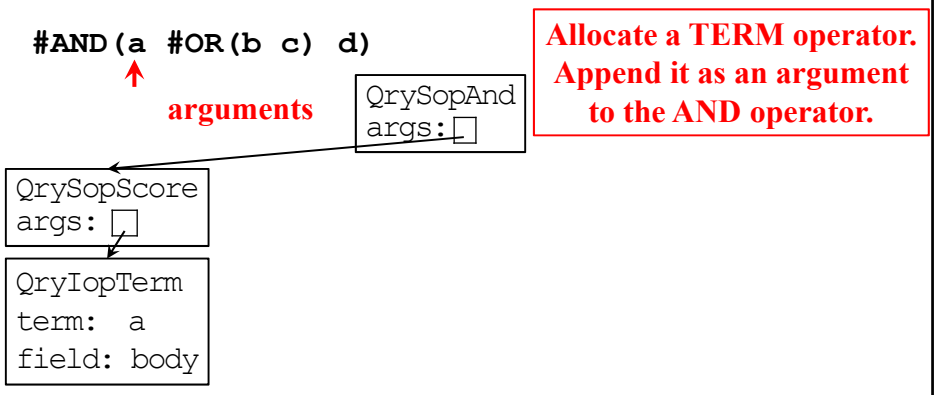
## Query Parsing





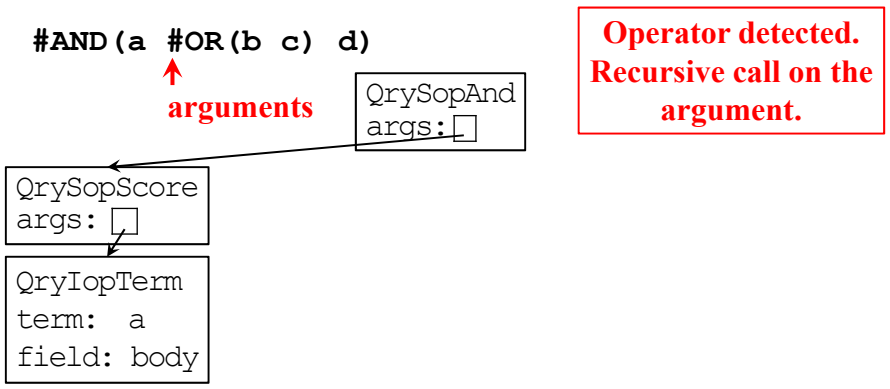



## Query Parsing




**Note:** Qry.appendArg automatically inserts #SCORE operators

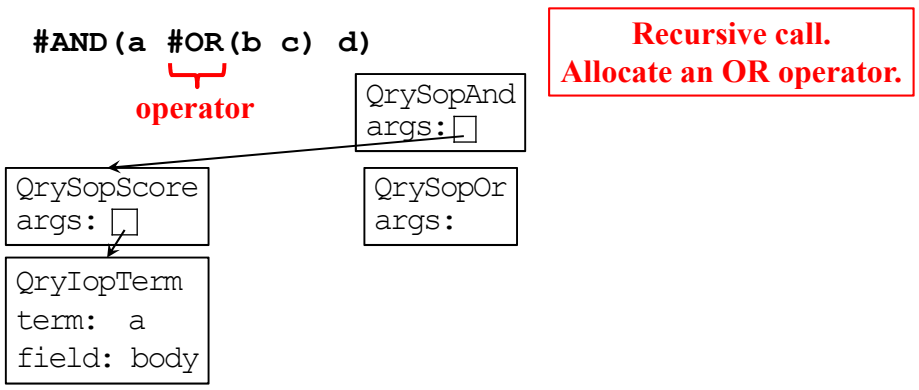
## Query Parsing



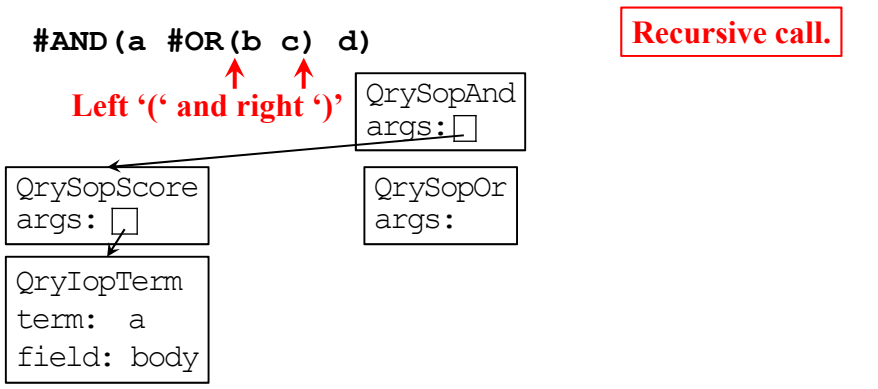




## Query Parsing



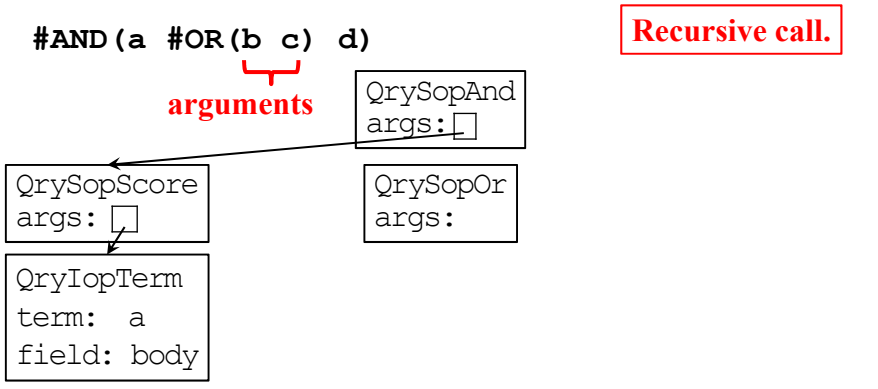
## Query Parsing




63

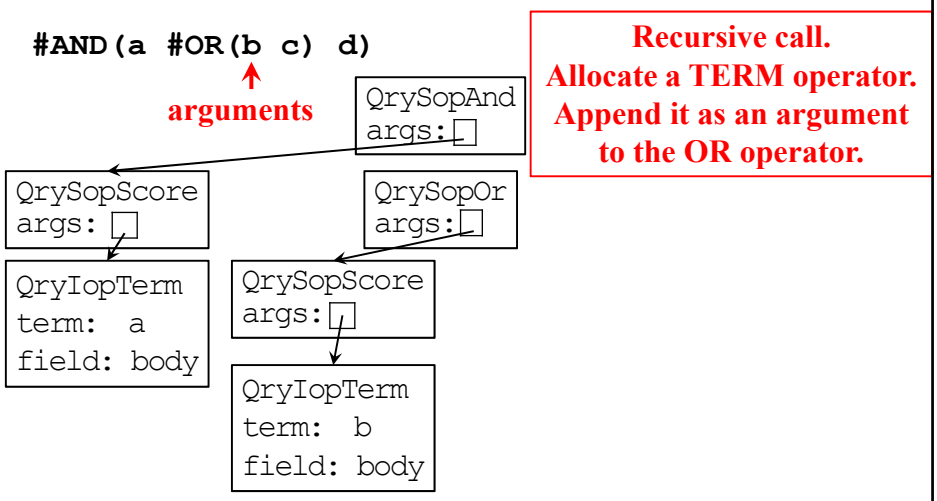


## Query Parsing




64

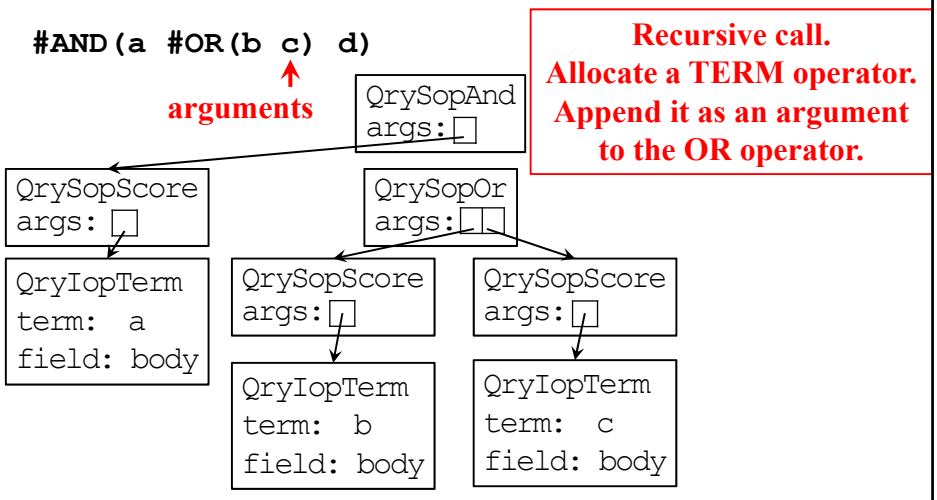
## Query Parsing




**Note:** Qry.appendArg automatically inserts #SCORE operators

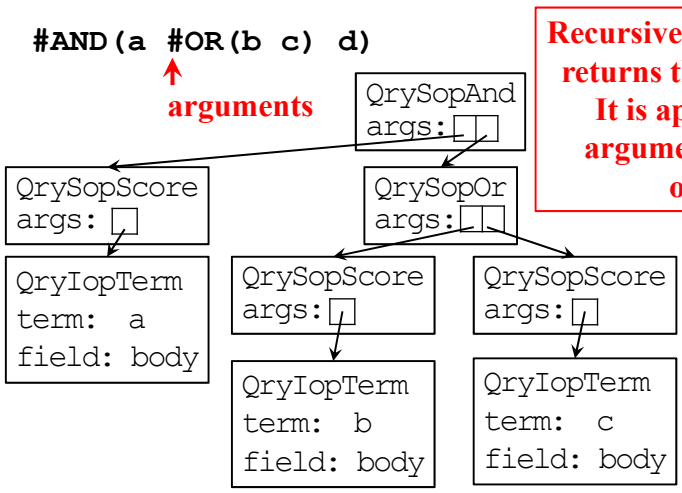



## Query Parsing




**Note:** Qry.appendArg automatically inserts #SCORE operators

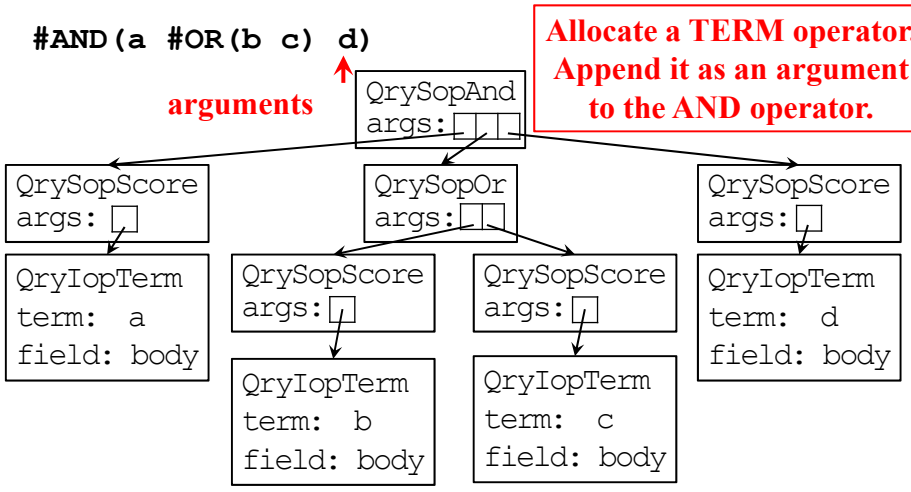
## Query Parsing




**Recursive call completes & returns the OR operator. It is appended as an argument to the AND operator.**




## Query Parsing



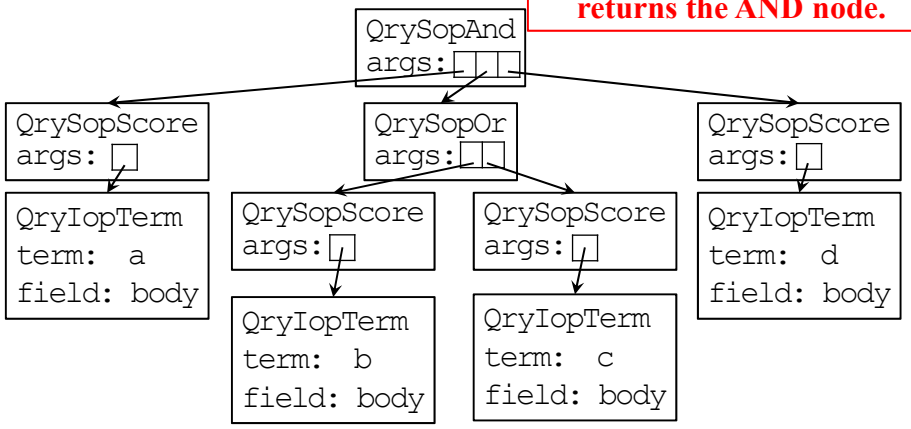

**Allocate a TERM operator. Append it as an argument to the AND operator.**

**Note:** Qry.appendArg automatically inserts #SCORE operators

## Query Parsing

**#AND(a #OR(b c) d)**

**Query parsing completes & returns the AND node.**








## Query Parsing

**a b c**

## Query Parsing

```
a b c
```

**This is a syntax error because there is no query operator**

**We want the search engine to support unstructured queries**

- **Solution:** The caller must add the default query operator
- E.g., `a b c` → `#OR(a b c)`

**Every retrieval model has a default query operator**

- But it isn't the same for every retrieval model
- The query parser doesn't know which retrieval model will be used
- So, the query parser can't apply the default query operator for you
- Your code must do it




## Query Parsing

**You will need to modify the query parser**

- HW1: Add new query operators (e.g., #AND, #NEAR/n)
- HW2: Add support for query operators that require weights
  - E.g., #WSUM (0.5 barack 1.0 obama)

## Outline

- **QryEval overview**
- **Query evaluation**
  - The Qry class
  - Iteration
  - Matching
  - Calculating scores
- **Overview of query parsing**