EECS 395/495 Lecture 3
Scalable Indexing, Searching, and Crawling

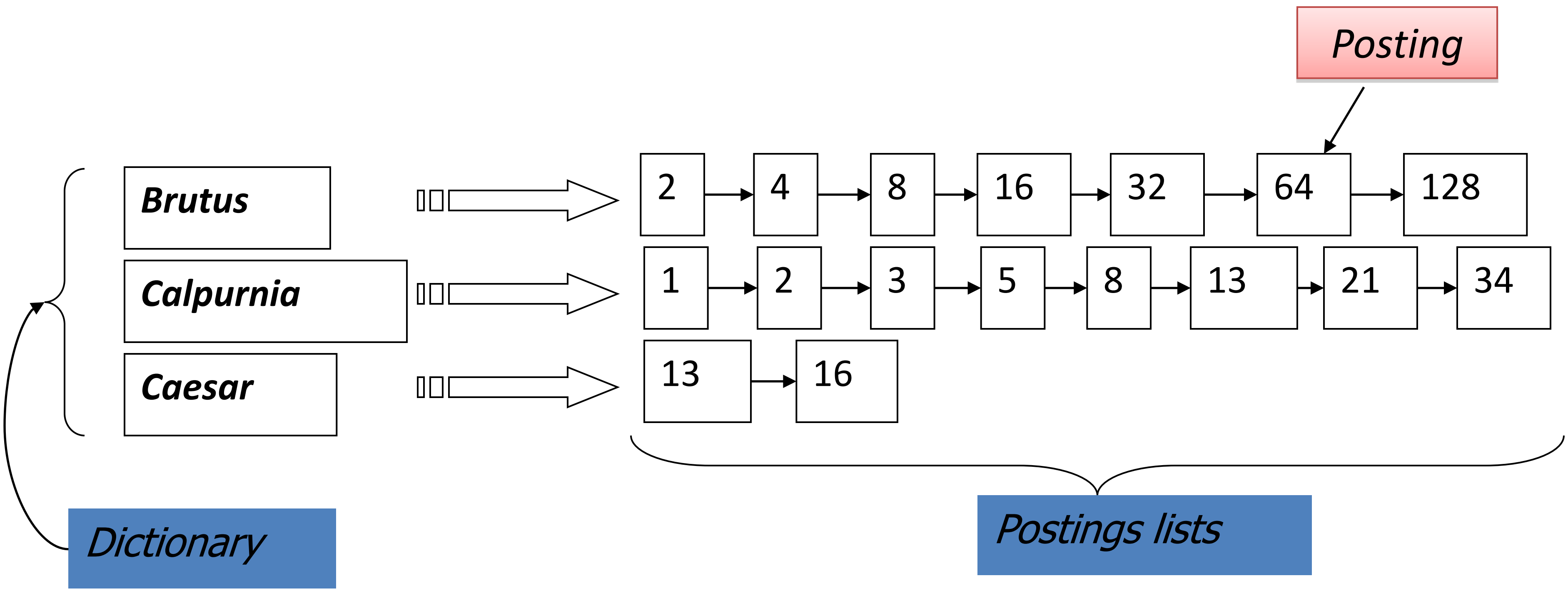
Doug Downey

Based partially on slides by Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze

Announcements

- Project “progress report” due **May 9**
  - ...reviews due **May 11**

Last time: Inverted index



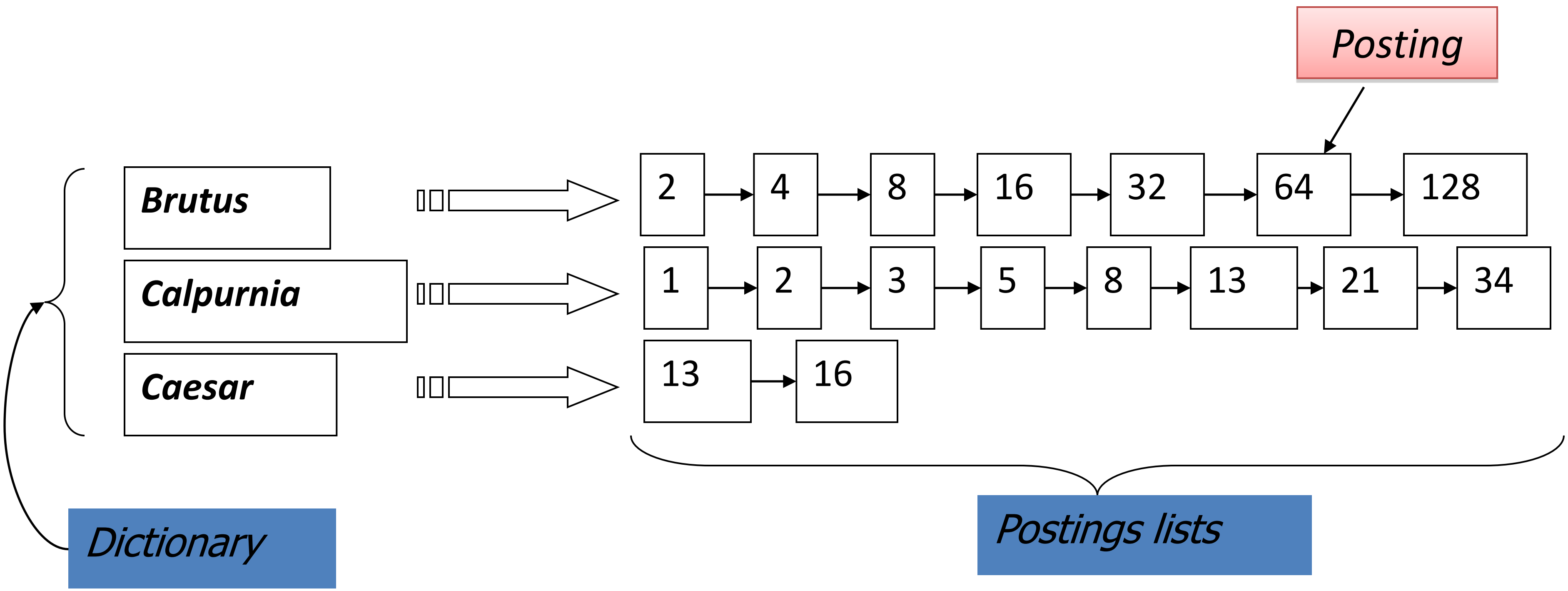
How Does Web Search Work?

- Inverted Indices
  - Inverted Indices enable fast IR
  - Tokenization & Linguistic issues
  - Handling phrase queries
  - **Scalable, distributed construction & deployment**
- Web Crawlers
- Ranking Algorithms

Today’s Outline

- Index compression
- Distributed Indexing
  - MapReduce
- Distributed Searching

Index Compression



Postings compression

- The postings file is much larger than the dictionary, factor of at least 10.
- Key goal: store each posting compactly.
  - A posting for our purposes is a docID.
  - Naïve: use $\log_2 20{,}000{,}000{,}000 \approx 34$ bits per docID.
- Our goal: use a lot less than 34 bits per docID.

Postings: two conflicting forces

- Common terms (e.g. ***the***)
  - 34 bits/posting is too expensive
  - Prefer 0/1 document incidence vector in this case
    - If ***the*** appears in half the docs => savings of **17x**!
- Rare terms (e.g. ***arachnocentric***)
  - Say it occurs once every million docs, on average
  - 0/1 document incidence vector is a disaster
    - $\log_2 1M \approx 20$ bits/occurrence uses **50,000x** less space!

Postings file entry

- We store the list of docs containing a term in increasing order of docID.
  - ***computer***: 33,47,154,159,202 …
- Consequence: it suffices to store *gaps*.
  - 33,14,107,5,43 …
- Hope: most gaps can be encoded/stored with far fewer than 34 bits.

Example: Three postings entries

	encoding	postings list									
THE	docIDs	...		283042		283043		283044		283045	...
	gaps				1		1		1		...
COMPUTER	docIDs	...		283047		283154		283159		283202	...
	gaps				107		5		43		...
ARACHNOCENTRIC	docIDs	252000		500100							
	gaps	252000	248100								

Preserves linear-time merge

– just keep track of sum as you traverse postings

Variable length encoding

- Aim:
  - For ***arachnocentric***, use ~20 bits/gap entry.
  - For ***the***, use ~1 bit/gap entry.
  - More generally:
    - to encode a gap with integral value G, use ~$\log_2 G$ bits
- Standard, fixed n-bit **`int`** won't work here
  - Solution: Variable length codes

Variable Byte (VB) codes

- For a gap value *G*
  - Begin with one byte to store *G* and dedicate 1 bit in it to be a continuation bit *c*
  - If *G* ≤127, binary-encode it in the 7 available bits and set *c* =1
  - Else
    - Encode *G*'s lowest-order 7 bits, put in rightmost byte
    - Encode *G*'s higher-order bits in additional bytes to left
    - When finished, set continuation bits:
      last byte *c*=1; other bytes *c*=0

Example

docIDs	824	829	215406
gaps		5	214577
VB code	00000110 10111000	10000101	00001101 00001100 10110001

Postings stored as the byte concatenation
000001101011100010000101000011010000110010110001

Key property: VB-encoded postings are uniquely prefix-decodable.

For a small gap (5), VB uses a whole byte.

Benefits from index compression

- Shrink index to ~15% the size of original text!
- …but we ignored positional information
  - In practice savings are more limited (about 30%-50% of original text)
  - Techniques are substantially the same

Today’s Outline

- Index compression
- **Distributed Indexing**
  - **MapReduce**
- Distributed Searching

Indexing at Web-scale….

- Sort by terms.

Core indexing step.

And we have to sort postings too…how do we do this with 20 billion documents?

Term	Doc #
I	1
did	1
enact	1
julius	1
caesar	1
I	1
was	1
killed	1
…	
so	2
let	2
it	2
be	2
with	2
caesar	2
the	2
…	

Term	Doc #
ambitious	2
be	2
brutus	1
brutus	2
capitol	1
caesar	1
caesar	2
caesar	2
did	1
enact	1
hath	1
I	1
I	1
i'	1
it	2
julius	1
killed	1
killed	1

www.rice.edu

http://www.sabahan.com/2006/08/03/picture-of-google-data-center/

Distributed indexing

- For web-scale indexing:
  use a distributed computing cluster
- Individual machines are fault-prone
  - Can unpredictably slow down or fail
- How do we exploit such a pool of machines?

Data centers

- Mainly commodity machines
- Distributed around the world
- Estimates for Google:
  - At least 1 million servers, est. 2% of worldwide supply (datacenterknowledge.com, 2010)
  - Approximately 200,000 new servers each quarter, based on expenditures of $1.6B/year

http://www.circleid.com/posts/20101021_googles_spending_spree_24_million_servers_and_counting/

- *"According to Microsoft research chief Rick Rashid, around 20 per cent of all the servers sold around the world each year are now being bought by a small handful of internet companies - he named Microsoft, Google, Yahoo and Amazon."*
  http://blogs.ft.com/techblog/2009/03/how-many-computers-does-the-world-need/

Google data centers

- If in a non-fault-tolerant system with 1000 nodes, each node has 99.9% uptime, what is the uptime of the system?
- Answer: 63%

Distributed indexing

- Maintain a *master* machine directing the indexing job – considered “safe”.
- Break up indexing into sets of (parallel) tasks.
- Master machine assigns each task to an idle machine from a pool.

Parallel tasks

- We will use two sets of parallel tasks
  - Parsers
  - Inverters
- Break the input document corpus into *splits*
- Each split is a subset of documents

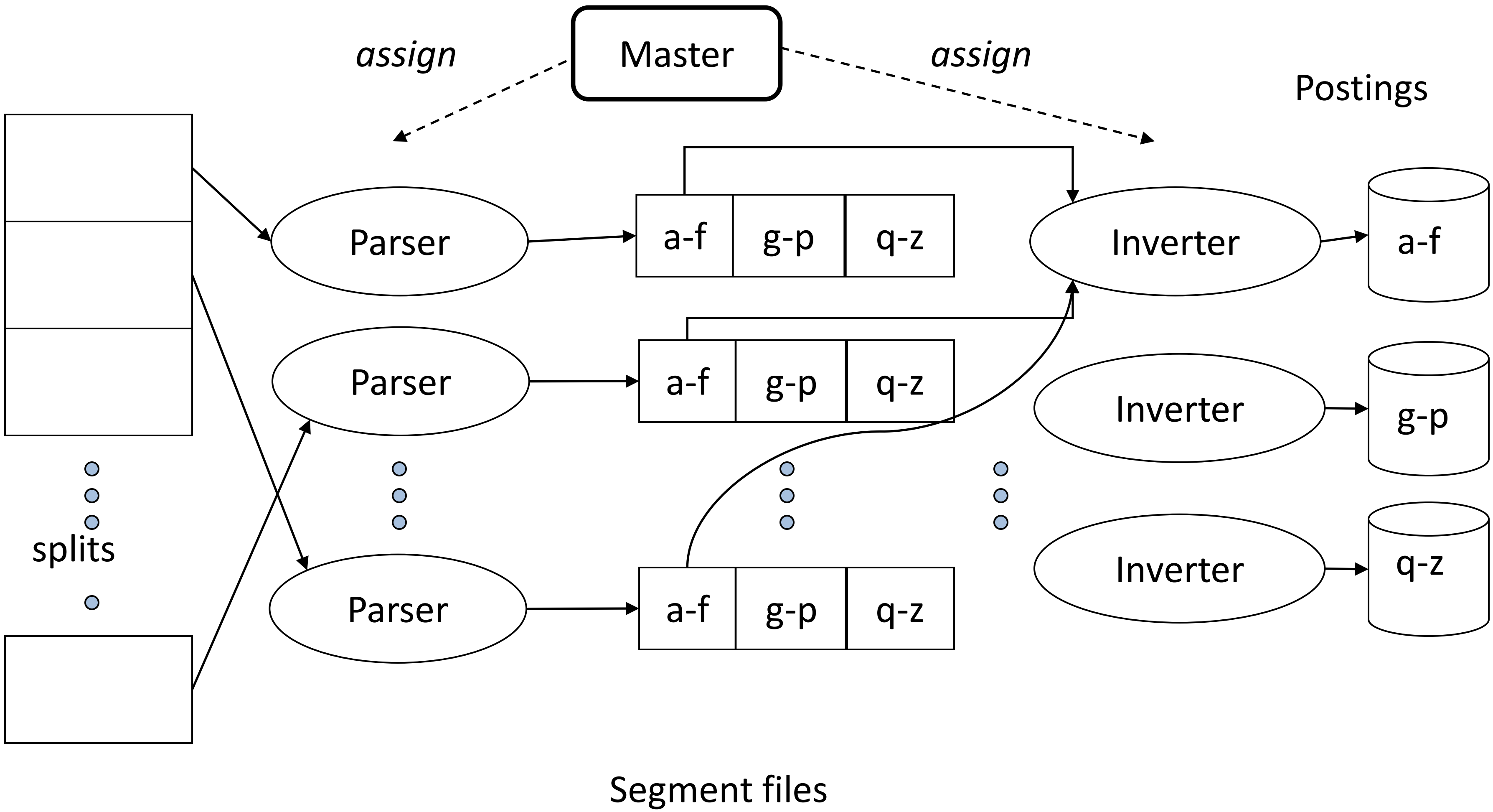
Parsers

- Master assigns a split to an idle parser machine
- Parser reads a document at a time and emits (term, doc) pairs
- Parser writes pairs into *j* partitions
- Each partition is for a range of terms' first letters
  - (e.g., ***a-f, g-p, q-z***) – here *j*=3.

Inverters

- An inverter collects all (term,doc) pairs (= postings) for one term-partition.
- Sorts and writes to postings lists
- Important: Each partition is small enough such that this sort can be done on a single machine
  - so in practice we'd use large *j* (many partitions) e.g., ***al-aq*** rather than ***a-j***

Data flow



Master
assign
assign
Postings
Parser
Parser
Parser
a-f
g-p
q-z
a-f
g-p
q-z
a-f
g-p
q-z
Inverter
Inverter
Inverter
a-f
g-p
q-z
splits
Segment files

MapReduce

- The index construction algorithm we just described is an instance of **MapReduce**
  - a robust and conceptually simple *framework* for distributed computing [Dean and Ghemawat 2004]
  - Handles the "distributed" part
- The Google indexing system (ca. 2002) consists of a number of phases, each until very recently implemented in MapReduce.

Schema for index construction in MapReduce

- **Schema of map and reduce functions**
  - **map:** input → list(<k, v>)   **reduce:** (k,list(v)) → output

- **Instantiation of the schema for index construction**
  - **map:** web collection → list(termID, docID)
  - **reduce:** (<termID1, list(docID)>, <termID2, list(docID)>, …) → (postings list1, postings list2, …)

MapReduce Example for Index Construction

- **map:**

d2 : C died. d1 : C came, C c′ed.

→

<C,d2>, <died,d2>, <C,d1>, <came,d1>, <C,d1>, <c′ed,d1>

- **reduce:**

(<C,(d2,d1,d1)>, <c′ed,(d1)> ,<came,(d1)>, <died,(d2)>)

→

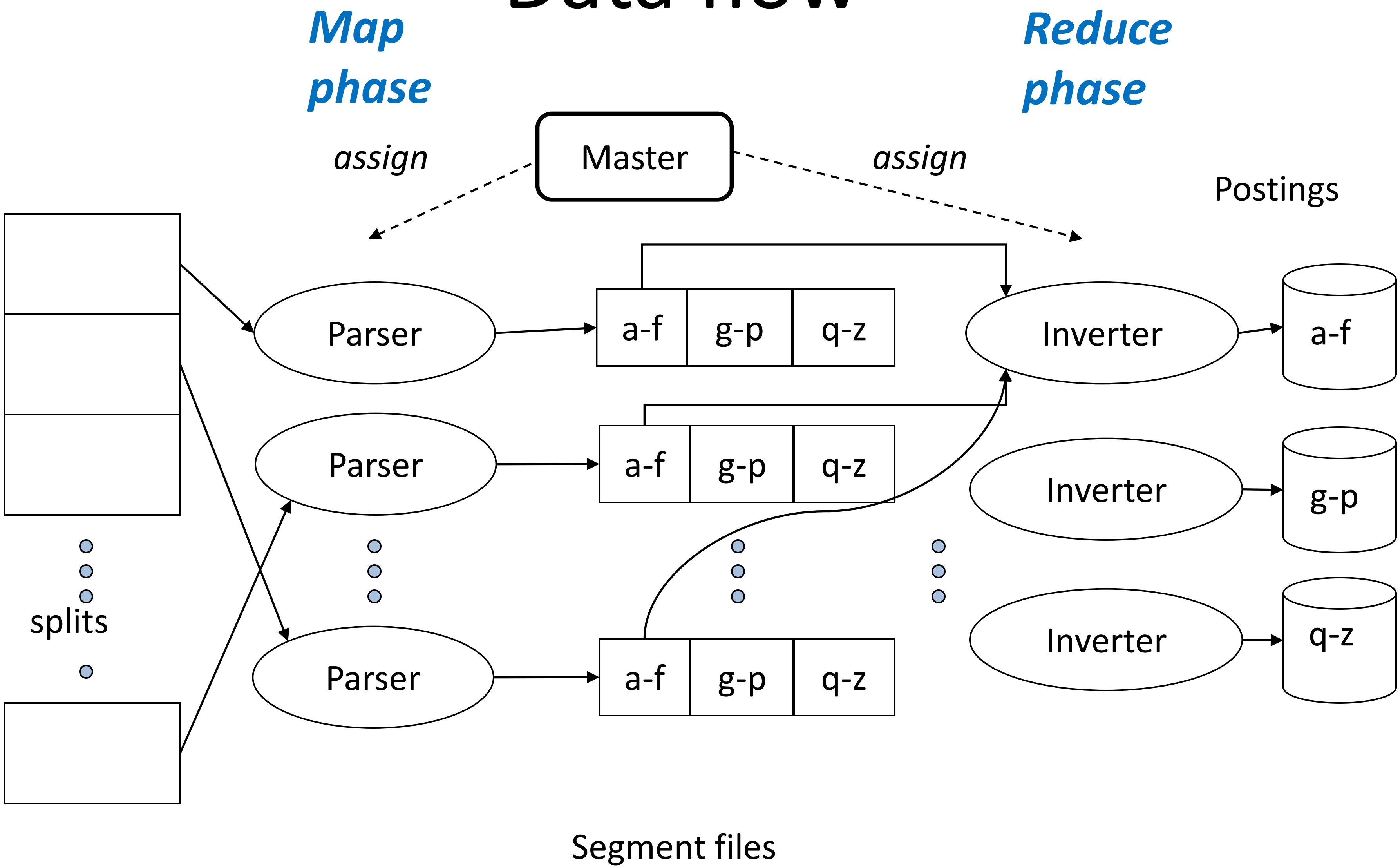
<C,(d1:2,d2:1)>, <c′ed,(d1:1)>, <came,(d1:1)>,
<died,(d2:1)>,

In key-sorted order.

Gee, it’s easy

- You write two methods:
  - **map:**
    takes a data object and emits (key, value) pairs
  - **reduce:**
    takes (key, value) pairs for **one** key and merges them
- You specify:
  - A *partition function* for reduce (which keys go where)
  - Parameters *M* (number of map tasks) and *R* (number of reduce tasks – same as *j*)

Data flow


Map phase
Reduce phase
assign
Master
assign
Postings
Parser
a-f
g-p
q-z
Inverter
a-f
Parser
a-f
g-p
q-z
Inverter
g-p
splits
Parser
a-f
g-p
q-z
Inverter
q-z
Segment files

MapReduce Code Example

- Word Counting
  - In a set of a million documents, how often does each word appear?
    - **Map**: for a given doc, emit <*word*, 1> for each *word* that appears
    - **Reduce:** Output sum

```
// User's map function
class WordCounter : public Mapper {
 public:
  virtual void Map(const MapInput& input) {
    const string& text = input.value();
    const int n = text.size();
    for (int i = 0; i < n; ) {
      // Skip past leading whitespace
      while ((i < n) && isspace(text[i]))
        i++;

      // Find word end
      int start = i;
      while ((i < n) && !isspace(text[i]))
        i++;
      if (start < i)
        Emit(text.substr(start,i-start),"1");
    }
  }
};
REGISTER_MAPPER(WordCounter);
```

[Dean and Ghemawat, 2004]

```
// User's reduce function
class Adder : public Reducer {
  virtual void Reduce(ReduceInput* input) {
    // Iterate over all entries with the
    // same key and add the values
    int64 value = 0;
    while (!input->done()) {
      value += StringToInt(input->value());
      input->NextValue();
    }

    // Emit sum for input->key()
    Emit(IntToString(value));
  }
};
REGISTER_REDUCER(Adder);
```

[Dean and Ghemawat, 2004]

Fun with MapReduce (1 of 4)

- Distributed grep
  - In a set of files, which lines include the word ***arachnocentric***?
    - **Map**: scan text and output matching lines
    - **Reduce:** Copy the input

Fun with MapReduce (2 of 4)

- Monte Carlo simulation
  - E.g., Given computer players of backgammon A and B, how often will A beat B?
    - **Map**: play a game and emit <A, 1> or <B, 1> based on who wins
    - **Reduce:** Sum
  - post-processing: compute the ratio

Fun with MapReduce (3 of 4)

- Boolean Satisfiability
  - Given a boolean formula in 3-CNF, e.g.:
    `(x1 v -x3 v x7) ^ (x4 v x5 v -x6) ^` …
    How many distinct assignments to variables (i.e. xi = true|false) make the formula true?
  - Create *M* = 2^*k* splits, one for each assignment
    - **Map**: Output <whatever, 1> *iff* the assignment makes the formula true
    - **Reduce:** Sum

Fun with MapReduce (4 of 4)

- Link Graph Index
  - Build an index to answer "which pages link to me" queries (i.e. Google's **`link:`** operator)
    - **Map**: For document *d1*, output <*dj*, *d1*> for all <a href =*dj*> tags therein
    - **Reduce:** Sort by values into posting lists
  - Partition function (which **map** keys are grouped together)
    - Sorted, e.g. ranges of domains (***www.aa*-www.ai****)

Today's Outline

- Index compression
- Distributed Indexing
  - MapReduce
- **Distributed Searching**

Distributed Searching

- We described indexing into a *term-partitioned* structure
  - Each machine handles some fraction of terms (e.g. ***al-aq***)
- Another possibility:
  - *Document-partitioned:* one machine handles a subset of documents

Back to MapReduce

- How do we change our implementation to be document-partitioned?
- One possibility:
  - **map:** output <doc-group | word-id, docID>
  - **reduce:** sum and output posting list as before
  - **partition function:** by doc-group

Next Time(s)

- Web Crawlers, Link Analysis, Information Extraction