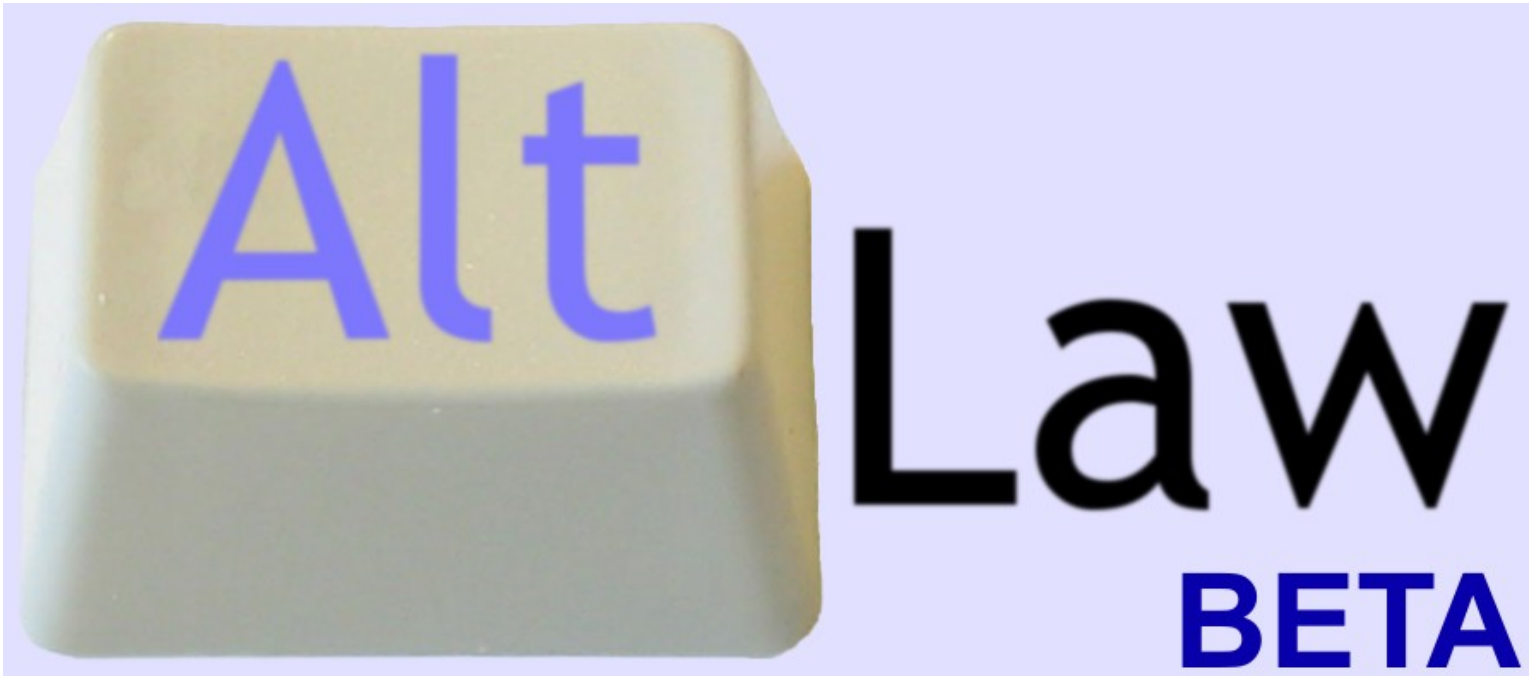
www.altlaw.org



Stuart Sierra
Program on Law & Technology
Columbia Law School
www.columbiaLawTech.org

Data Sources – Large Corpora

Paul Ohm, U. of Colorado Law
bulk.altlaw.org

7 GB **200,000 cases**

public.resource.org

2 GB **700,000 cases**

Cornell Legal Information Institute
www.law.cornell.edu

0.7 GB **60,000 statutes**

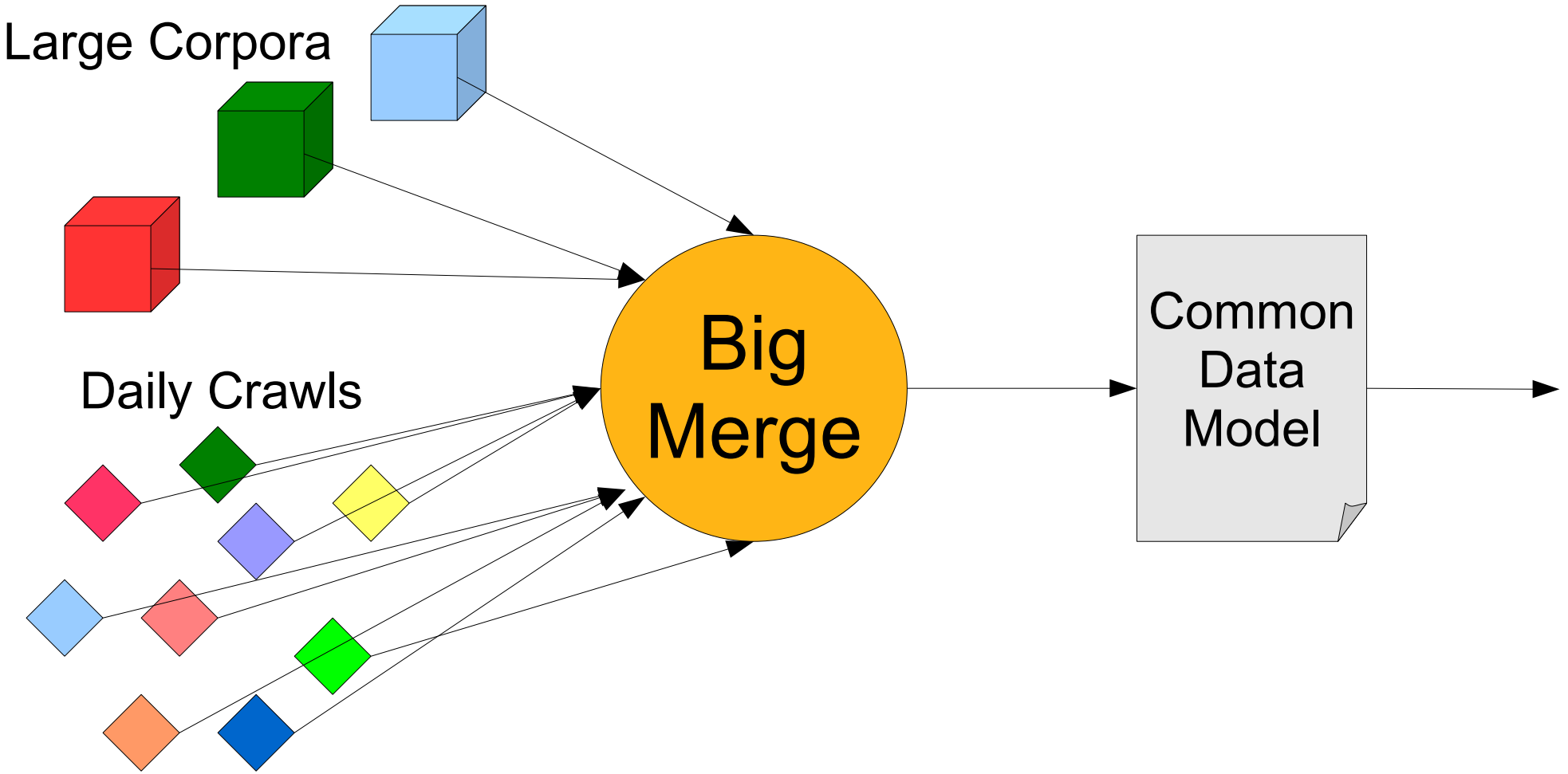
Data Sources – Court Web Sites

www.supremecourtus.gov
www.ca1.uscourts.gov
www.ca2.uscourts.gov
www.ca3.uscourts.gov
www.ca4.uscourts.gov
www.ca5.uscourts.gov
www.ca6.uscourts.gov
. . .

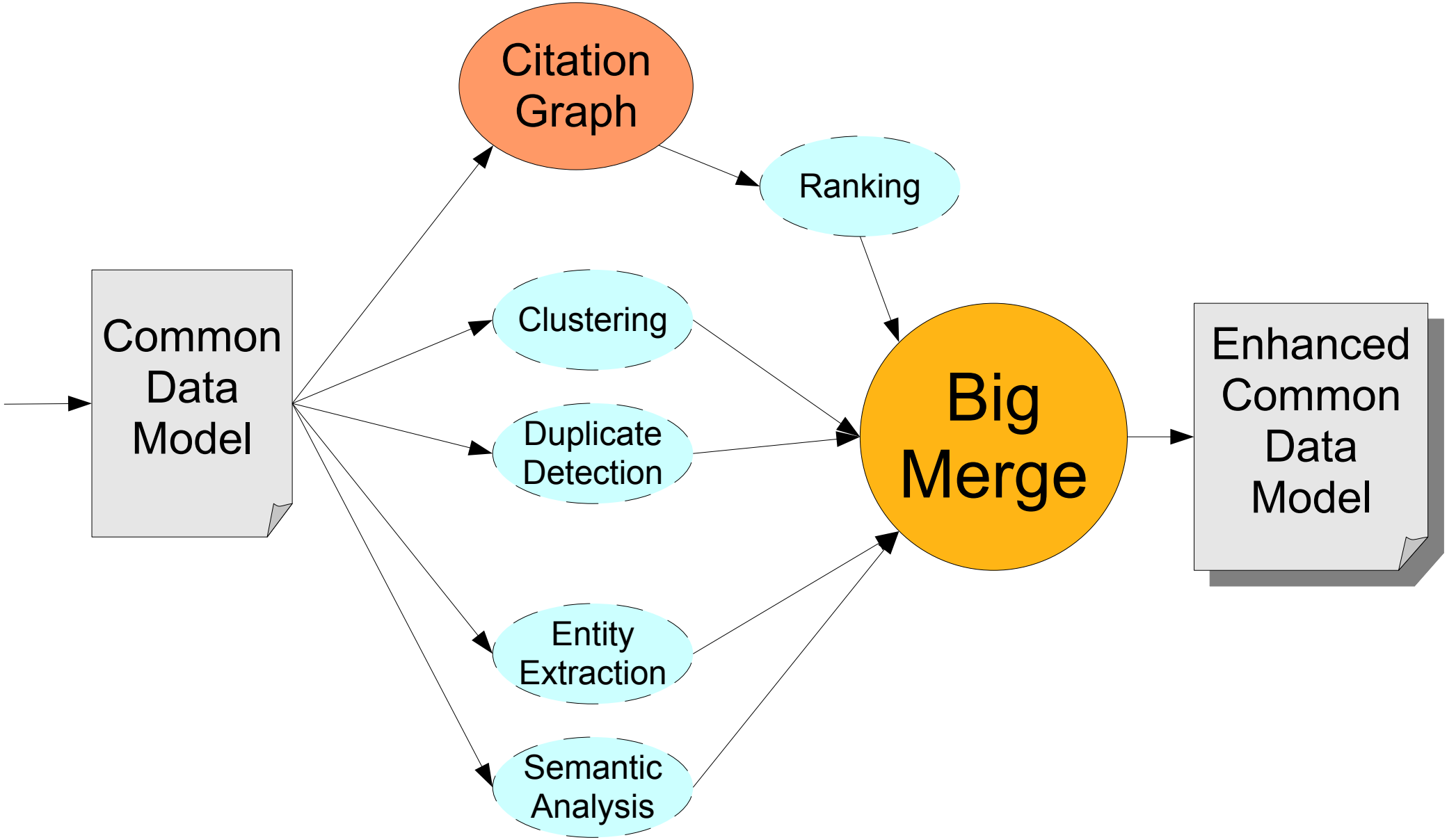
14 appeals courts total
94 district courts
?? state courts
?? local/other courts

- 20-40 new cases daily
- PDF, WordPerfect, HTML, plain text

Back-end (1)



Back-end (2)


Citation Graph
Ranking
Common Data Model
Clustering
Duplicate Detection
Big Merge
Enhanced Common Data Model
Entity Extraction
Semantic Analysis

MapReduce – it works for Google

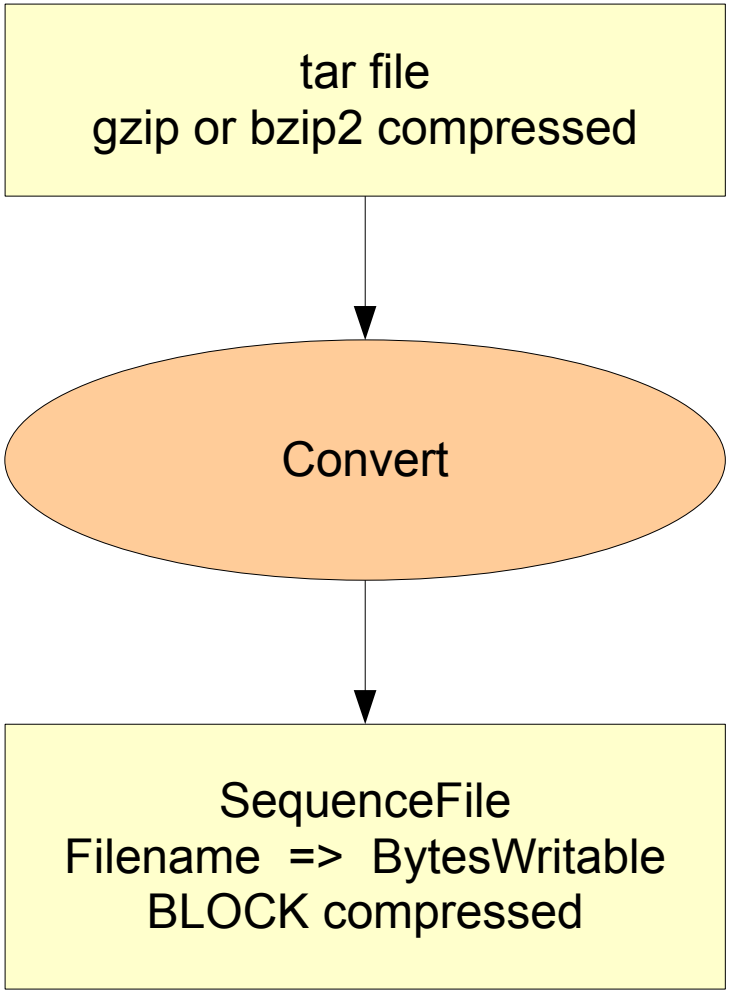
- Disk is the bottleneck.
- *Seek time* is the bottleneck.

MapReduce:
- Minimize disk seeks.
- Run at full transfer rate.

“Disk is the new tape.”

The Million Little Files Problem

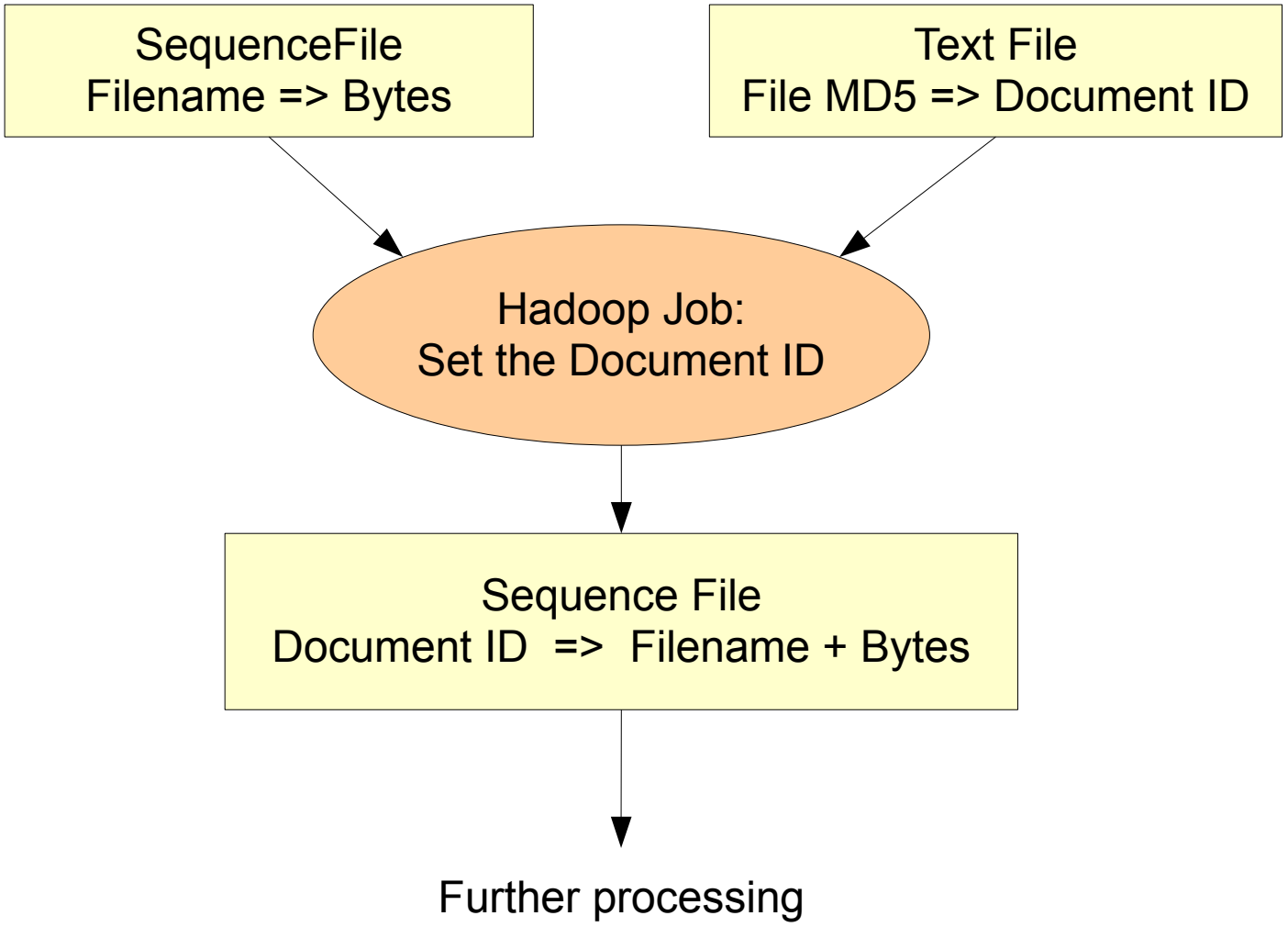
http://stuartsierra.com/2008/04/24/a-million-little-files



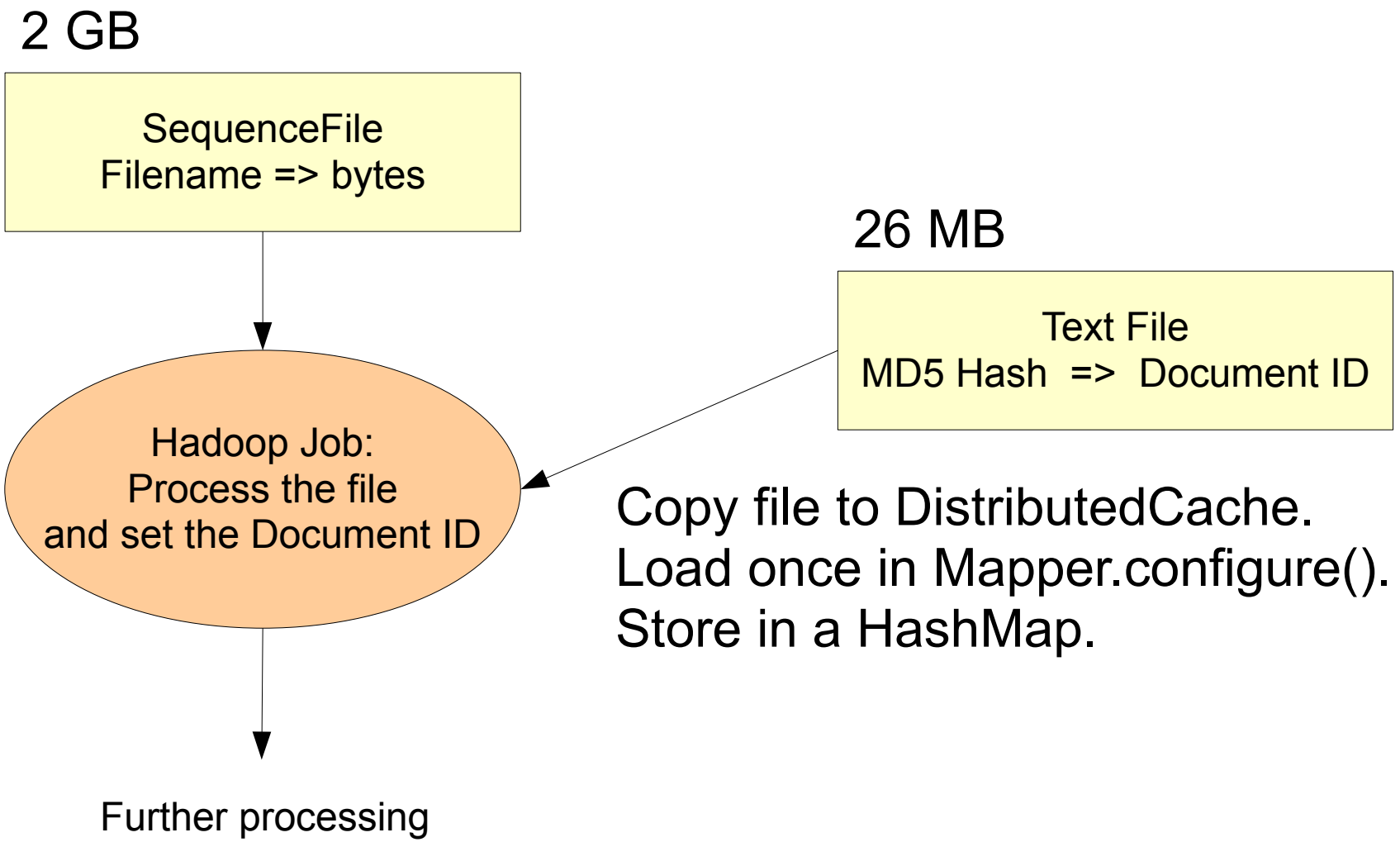
- Cannot run in parallel
- Slow: 20 MB / minute
- Only has to be done once

Newer solution in 0.19:
Hadoop Archives (HAR files)

The Document ID Problem

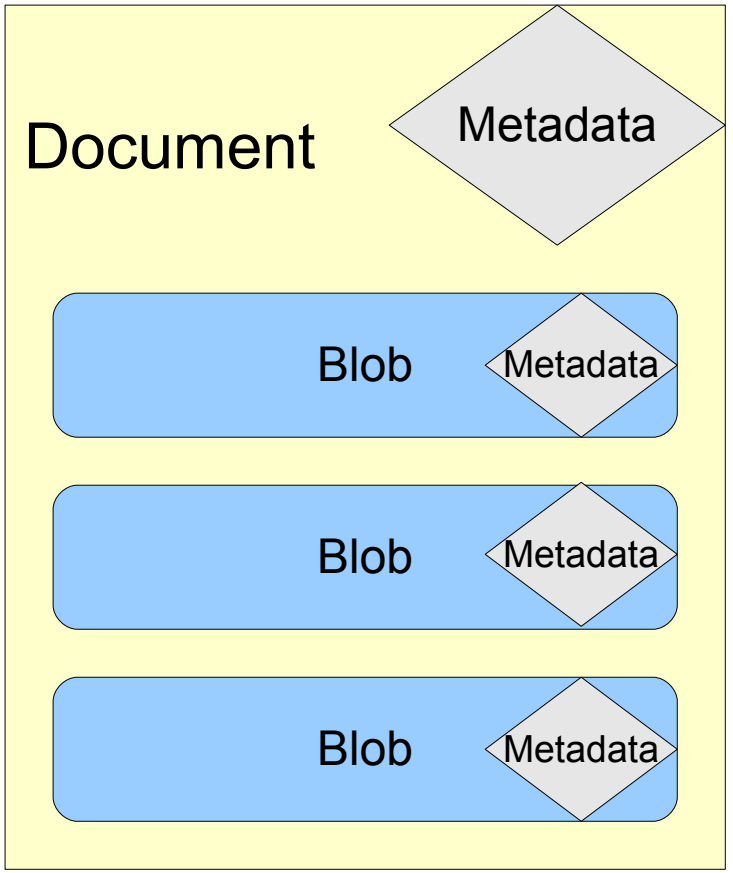


If it Fits in RAM, Keep it in RAM



The State Management Problem

The Document-Blob Model



or, the “Store Everything” model.

or, Stuart's worst idea ever.

The State Management Problem

- “Out of the Tar Pit”
  http://lambda-the-ultimate.org/node/1446

Essential State

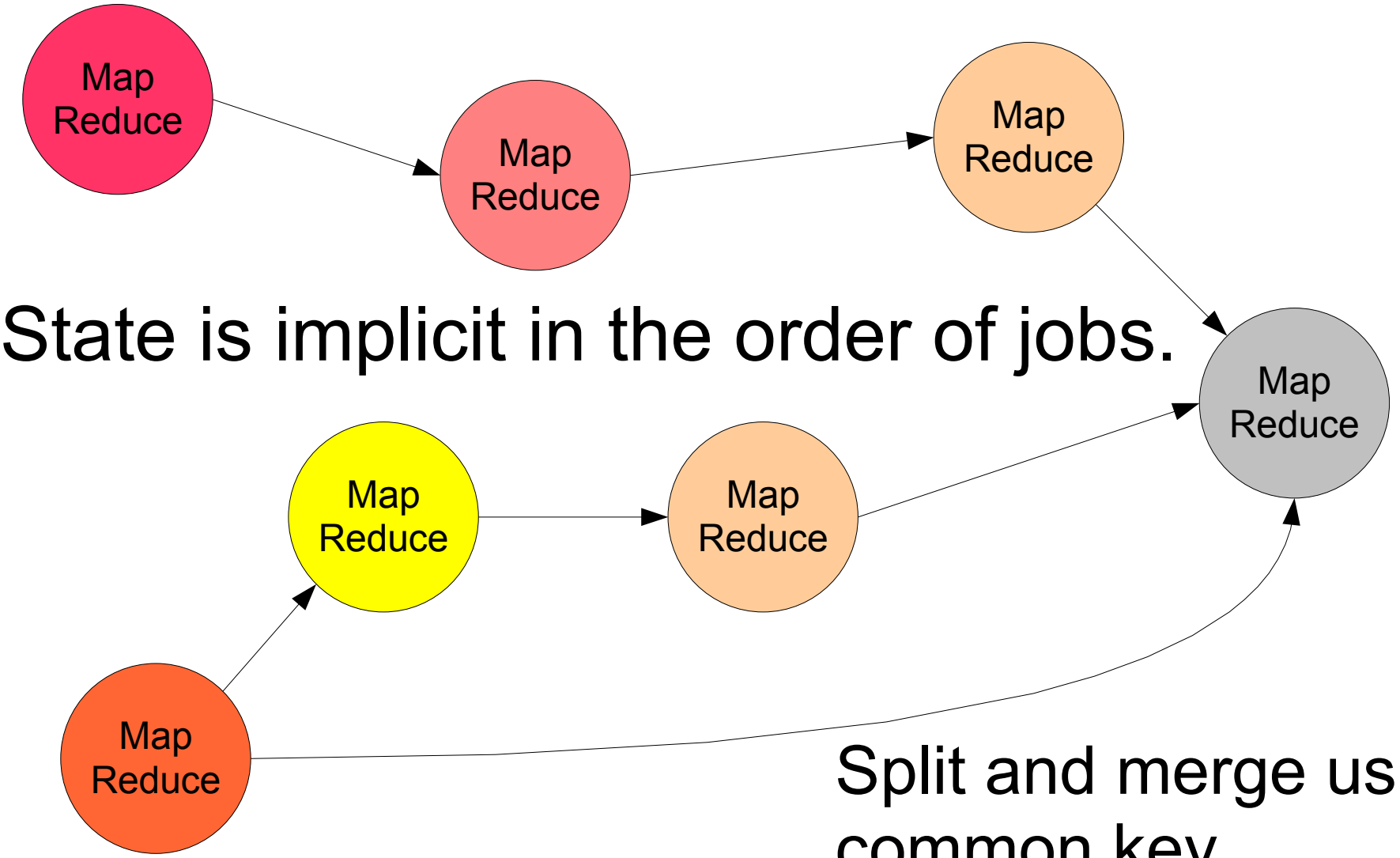
You have to store this.

Derived State

Never store this.

Cache as needed
for performance.

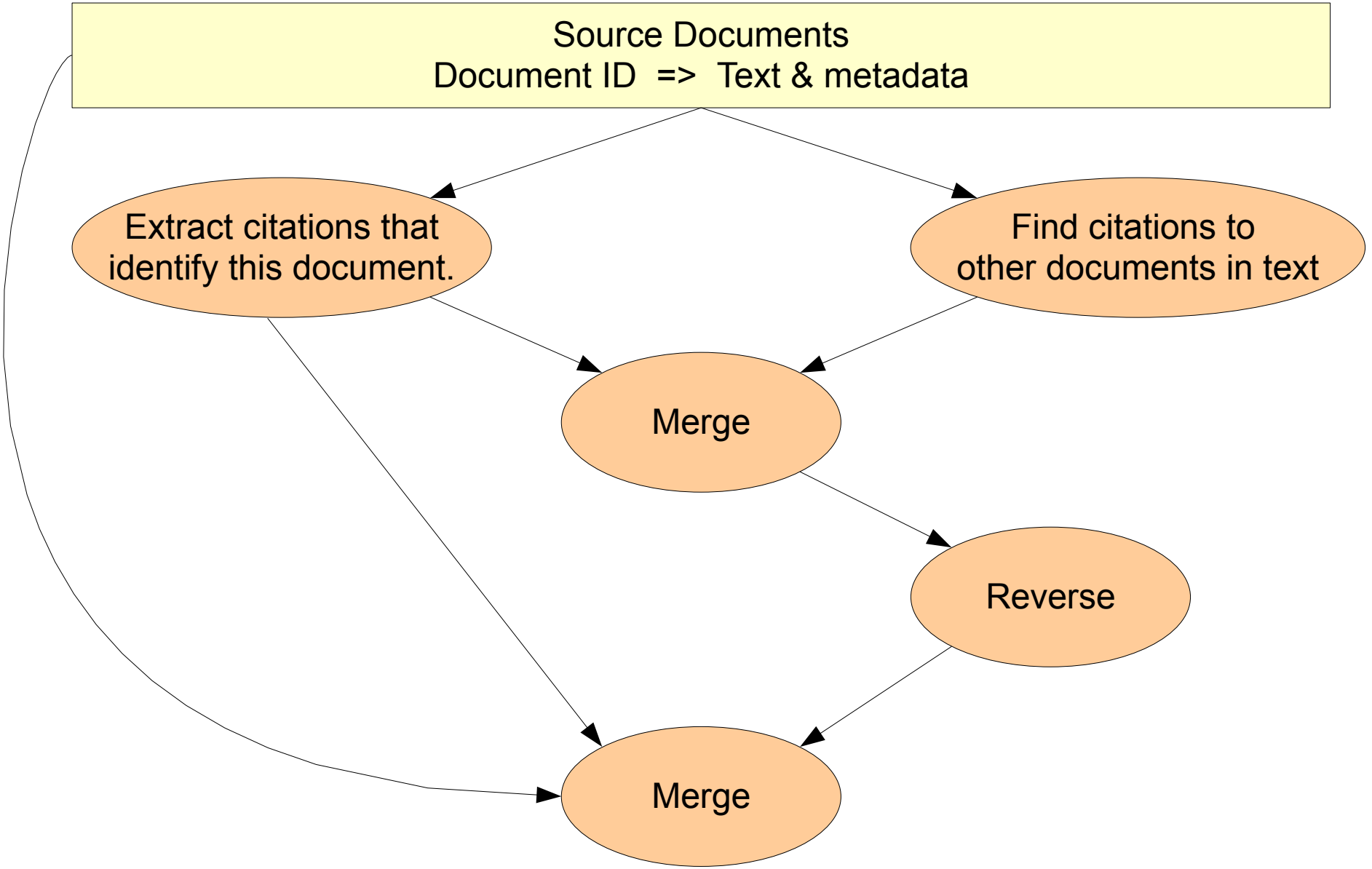
The State Management Problem



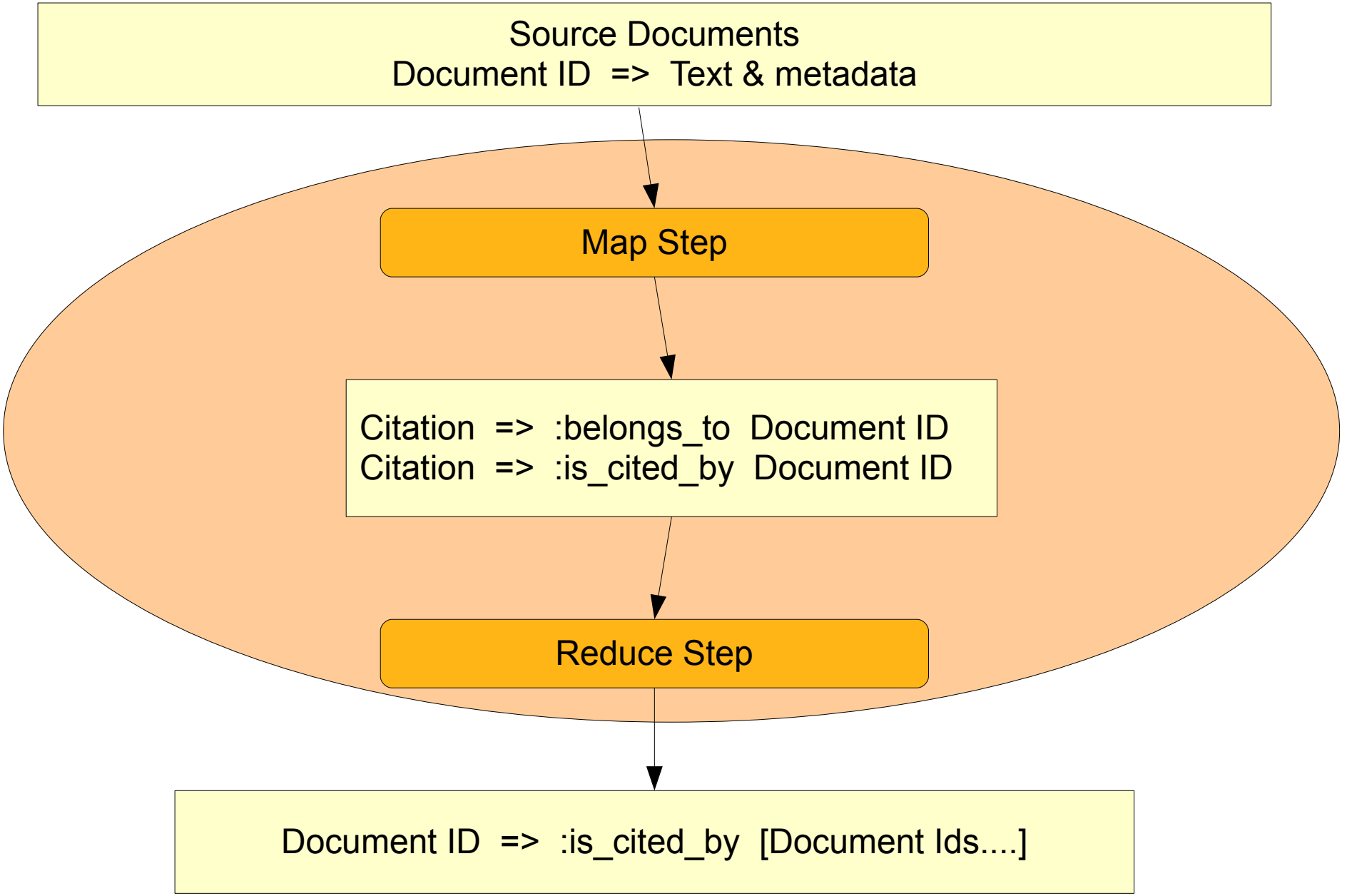
State is implicit in the order of jobs.

Split and merge using common key.

The Citation Linking Problem



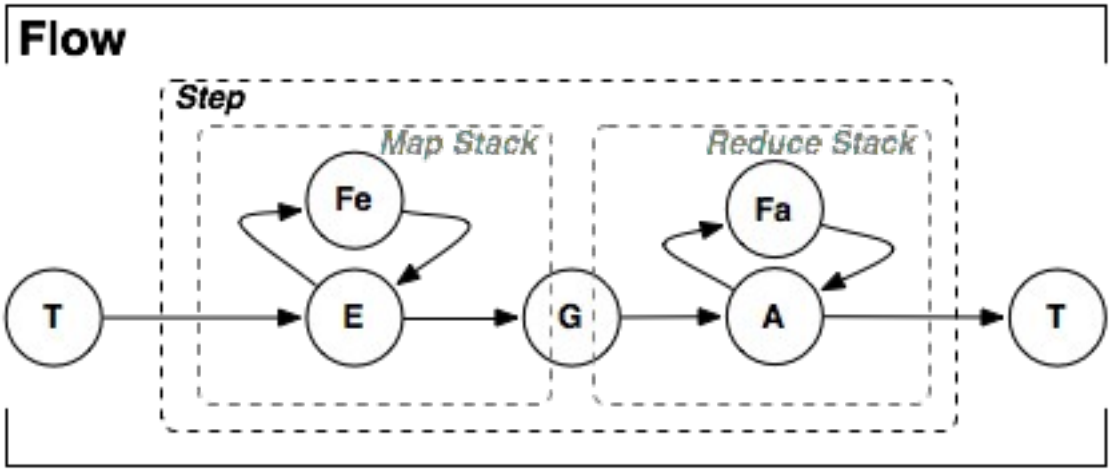
The Citation Linking Problem



```
(ns org.altlaw.jobs.link.linkincites-mapred
    (:gen-class
     :extends org.apache.hadoop.mapred.MapReduceBase
     :implements [org.apache.hadoop.mapred.Mapper
                  org.apache.hadoop.mapred.Reducer])
    (:use org.altlaw.util org.altlaw.util.hadoop)
    (:refer clojure.set))

(import-hadoop)


(defn -map [this wkey wvalue output reporter]
  (.collect output wkey
            (if (instance? Text wvalue)
              wvalue
              (Text. (str wvalue)))))


(defn -reduce [this wdocid ivalues output reporter]
  (let [values (doall (map (comp read-string str) (iterator-seq ivalues)))
        cited-docids (filter integer? values)
        citing-doc (select-keys
                    (first (filter (complement integer?) values))
                    [:docid :doctype :citations :court :date :name])]
    (doseq [docid cited-docids]
      (.collect output (IntWritable. docid)
                (Text. (pr-str {:incites #{citing-doc}}))))))
```

www.clojure.org

Cascading



What I Learned

- Clojure + Hadoop = awesome.
- If it fits in RAM, keep it in RAM.
- Don't store what you can recompute.
- Plan to run every job repeatedly.
- Design jobs to be chained.
- A job can have heterogeneous output.
- Test on small samples.