CSE 138 Lecture 17

this time:

- online systems vs. offline systems, raw data vs. derived data
- intro to MapReduce
- MapReduce example: inverted index from forward index

"services" or "online systems"

→ systems that accept requests from clients, and respond to clients.

- KVSes
- web servers
- caches
- databases (sometimes)

etc.

These all wait for client requests and try to handle them, ideally quickly.

- low latency (time between request and response) is important
- high availability (every request receives a response) is important

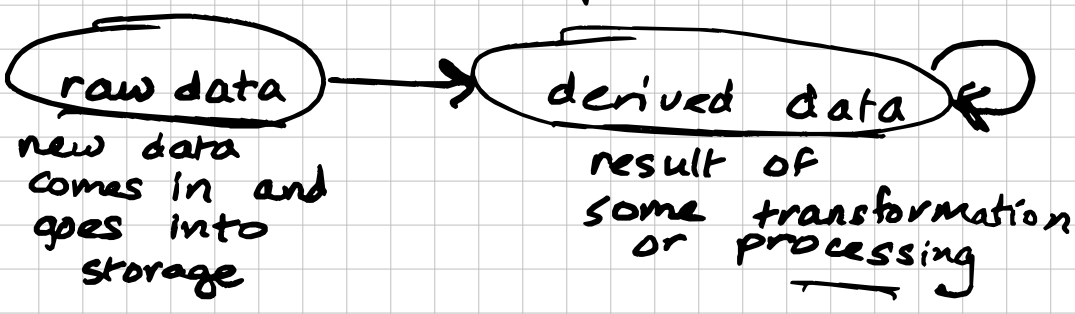
→ "batch processing systems" aka "offline systems"

- Take in a huge amount of data at once
- Process it (might take hours/days)
- Produce output data (also potentially huge)
- High throughput is important

throughput = how much data can we process in X amount of time?)

→ users accessing Facebook

→ data scientist analyzing aggregate data about how people use Facebook



MapReduce is a tool for computing derived data.

example: inverted index

Web crawler gets raw data:

forward index

Document	words
doc1	the, quick, brown, fox, ...
doc2	the, dog, is, cute, ...
doc3	i, love, my, dog
⋮	

This is formatted in a way that's bad for queries like "give me all documents containing 'fox'".

inverted index

word	Documents
fox	doc1
quick	doc1
dog	doc2, doc3, doc1
cute	doc2
⋮	⋮

To compute the inverted index from the forward index:

- for each document D in forward index:
  - for each word W in D:
    - emit (w, D)
- for each unique w, combine Ds in a list

<w, doc1>
<fox, doc1>
<dog, doc1>
<dog, doc2>
<dog, doc3>
<cute, doc2>

<quick, [doc1]>
<fox, [doc1]>
<dog, [doc1, doc2, doc3]>
<cute, [doc2]>

↑ and there's our inverted index!

This algorithm is simple, but doing at large scale is a hard distsys problem.

idea of mapReduce:

Abstract away the hard distributed systems stuff and let people plug in the conceptually simple code for particular batch processing tasks.

- or sometimes quite sophisticated!

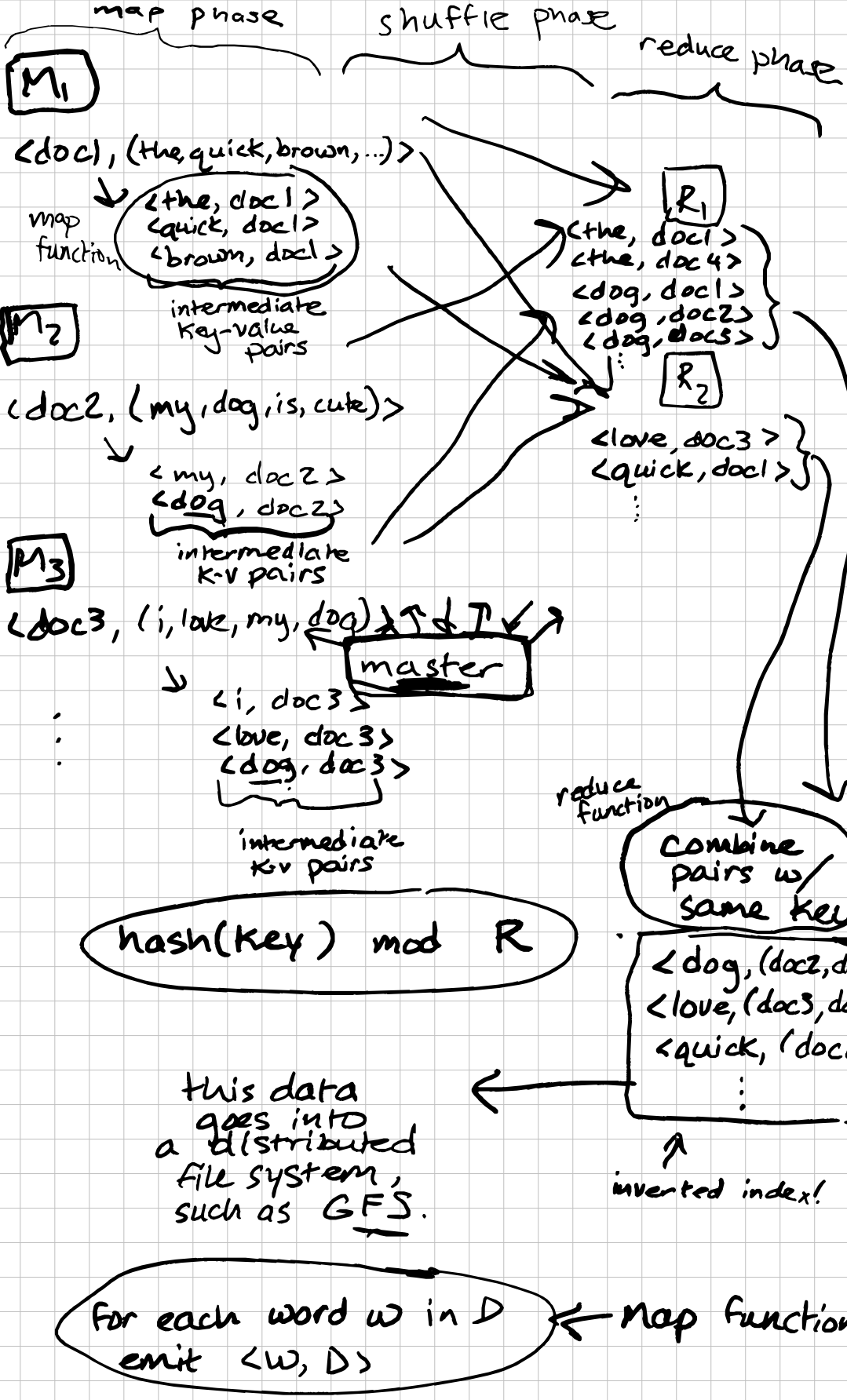
you have one part of the forward index — how to proceed?

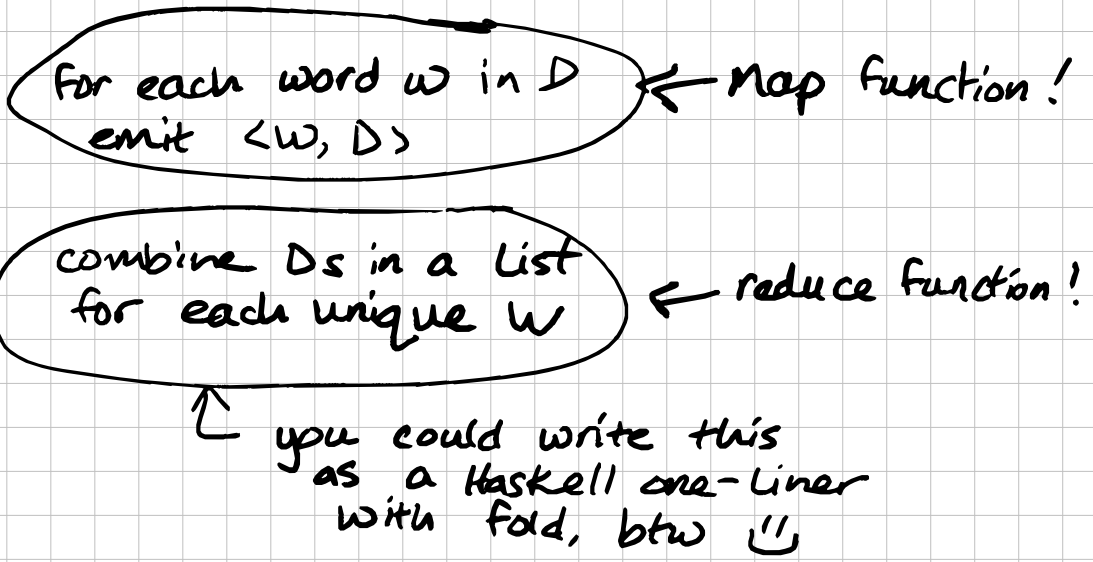
doc1 "the", "quick", "brown", "fox"

Recall that this is our first step:

for each document D in forward index:
  for each word W in D:
    emit <W, D>

This can be done in parallel for all the documents!
"embarrassingly parallel"





in MapReduce you only specify the above two functions that are specific to your computation, and maybe some configuration settings. The framework does the rest.

open-source clone of MapReduce: Hadoop! and HDFS.

its well-known successor is Spark!

Many batch processing tasks can fit into this model

- distributed grep ← search
- distributed sort
- distributed word count
- etc.

3 phases of a MR job:

map phase - programmer-supplied map function runs on all the input data, produces intermediate K-V pairs on map workers

M_1 M_2 M_3 . . .

shuffle phase - intermediate K-V pairs get moved to reduce workers

R_1 R_2 . . .

reduce phase - programmer-supplied reduce function runs on intermediate K-V pairs on reduce workers.

<dog, doc1>
<dog, doc2>
<cute, doc3>

Suppose we want a word count:

<doc1, (the, quick, brown, fox)>
<doc2, (my, dog, is cute)>
⋮

<dog, 1>
<cute, 1>
<dog, 1>

} map phase, emit these pairs

<dog, 5000>
<cute, 5,000,000>
⋮

reduce phase, compute sum of values for each unique key.