[537] Search Engines

Tyler Harter

12/10/14

Flash Review

Flash Hierarchy

Plane: 1024 to 4096 blocks
- planes accessed in parallel

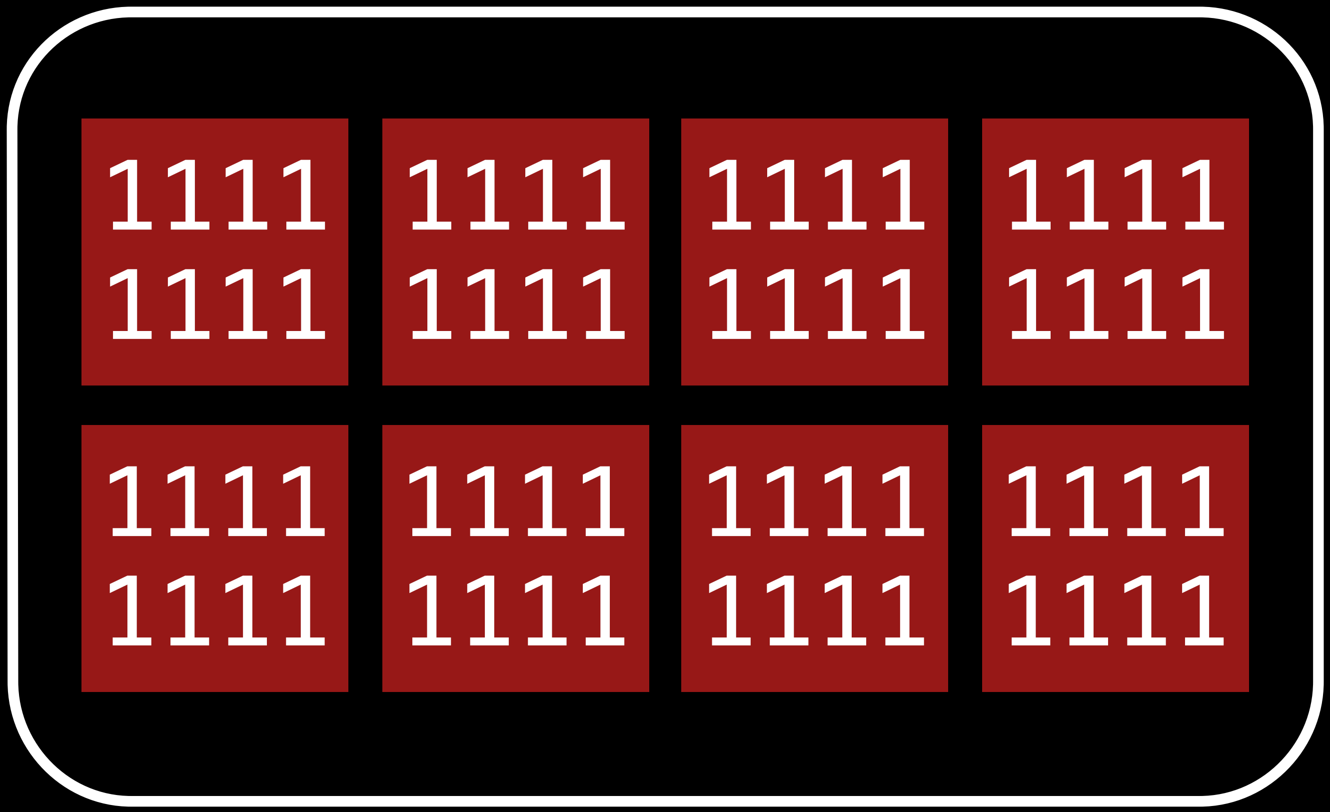
Block: 64 to 256 pages
- unit of erase

Page: 2 to 8 KB
- unit of read and program

Block

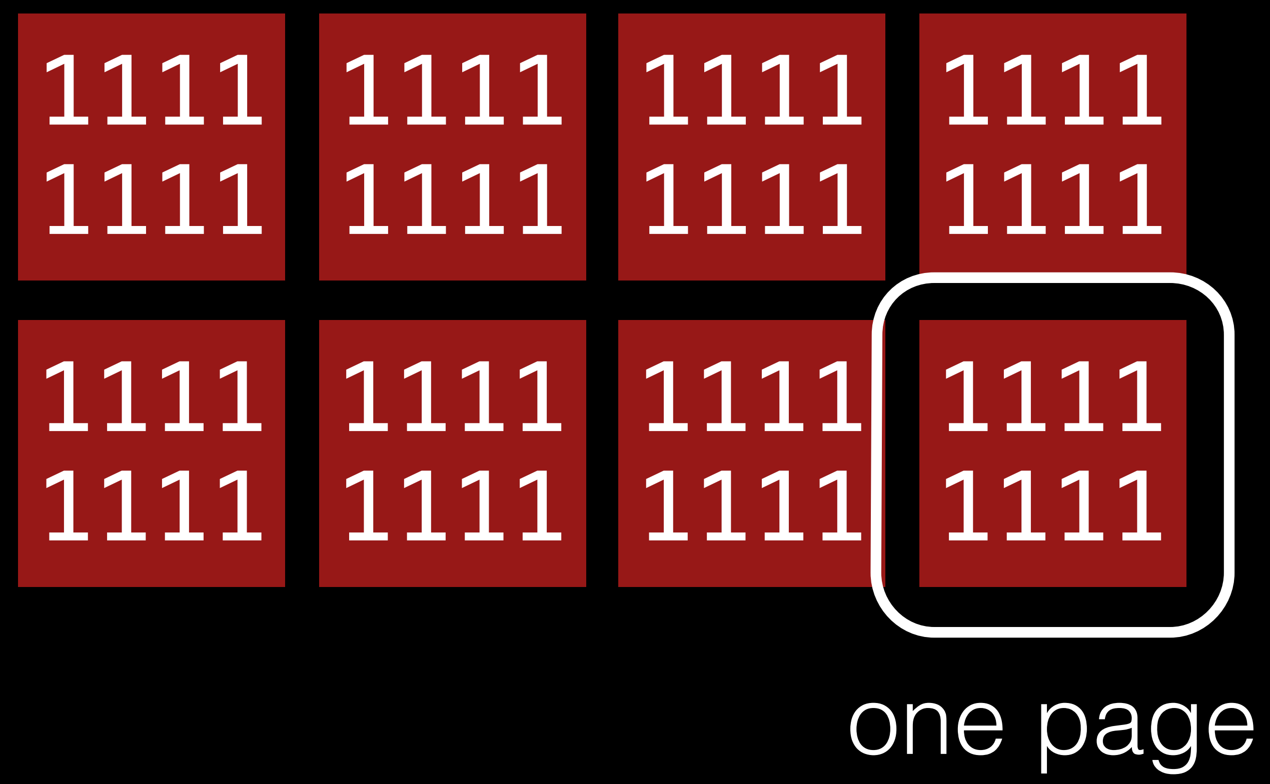
1111 1111	1111 1111	1111 1111	1111 1111
1111 1111	1111 1111	1111 1111	1111 1111

Block



one block

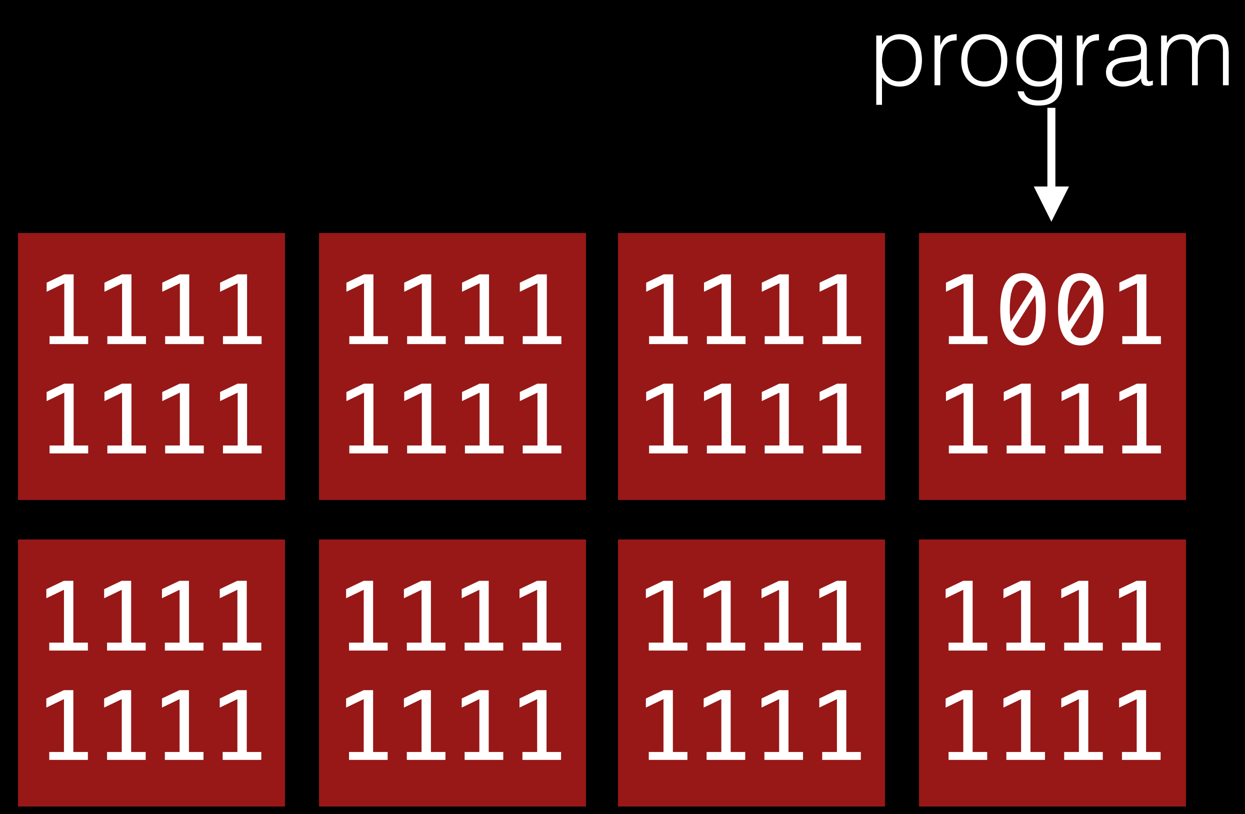
Block


1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
one page

Block


1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111

Block


program
1111
1111
1111
1111
1111
1111
1001
1111
1111
1111
1111
1111
1111
1111
1111
1111

Block

1111 1111	1111 1111	1111 1111	1001 1111
1111 1111	1111 1111	1111 1111	1111 1111

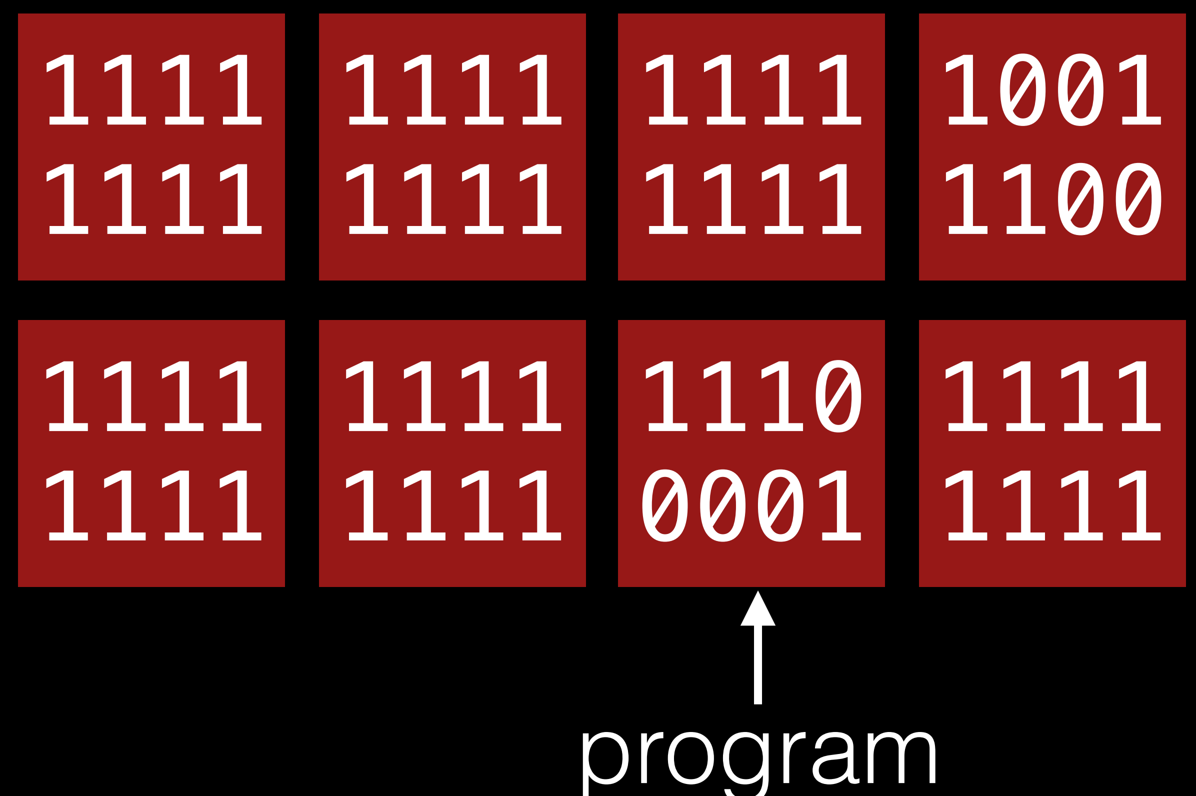
Block

program
1111 1111
1111 1111
1111 1111
1001 1100
1111 1111
1111 1111
1111 1111
1111 1111

Block

1111 1111	1111 1111	1111 1111	1001 1100
1111 1111	1111 1111	1111 1111	1111 1111

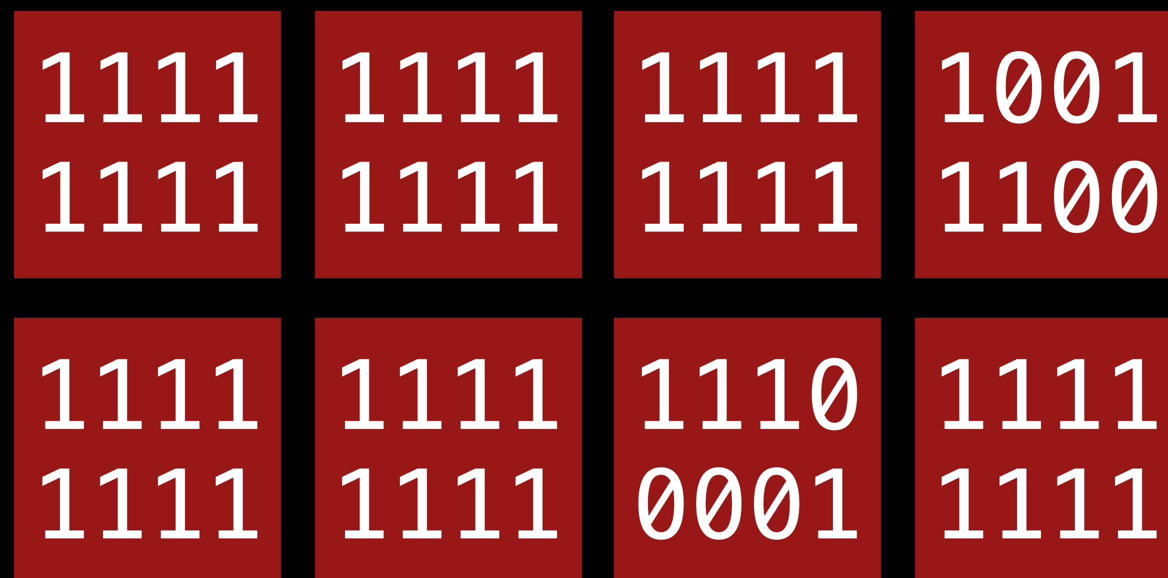
Block


1111 1111
1111 1111
1111 1111
1001 1100
1111 1111
1111 1111
1110 0001
1111 1111
program

Block

1111 1111	1111 1111	1111 1111	1001 1100
1111 1111	1111 1111	1110 0001	1111 1111

Block


1111
1111
1111
1111
1111
1111
1001
1100
1111
1111
1111
1111
1110
0001
1111
1111

erase

Block

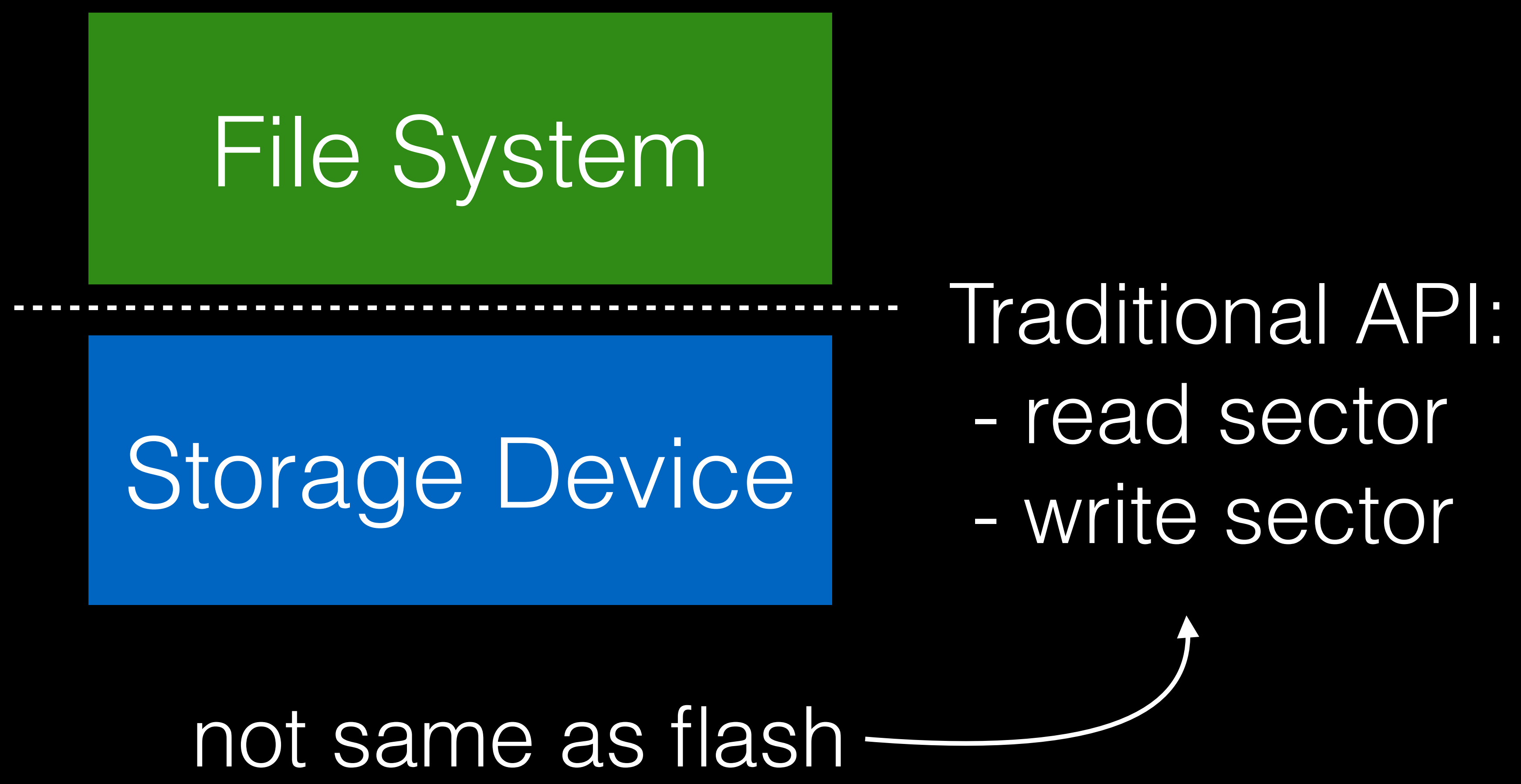
1111 1111	1111 1111	1111 1111	1111 1111
1111 1111	1111 1111	1111 1111	1111 1111

erase

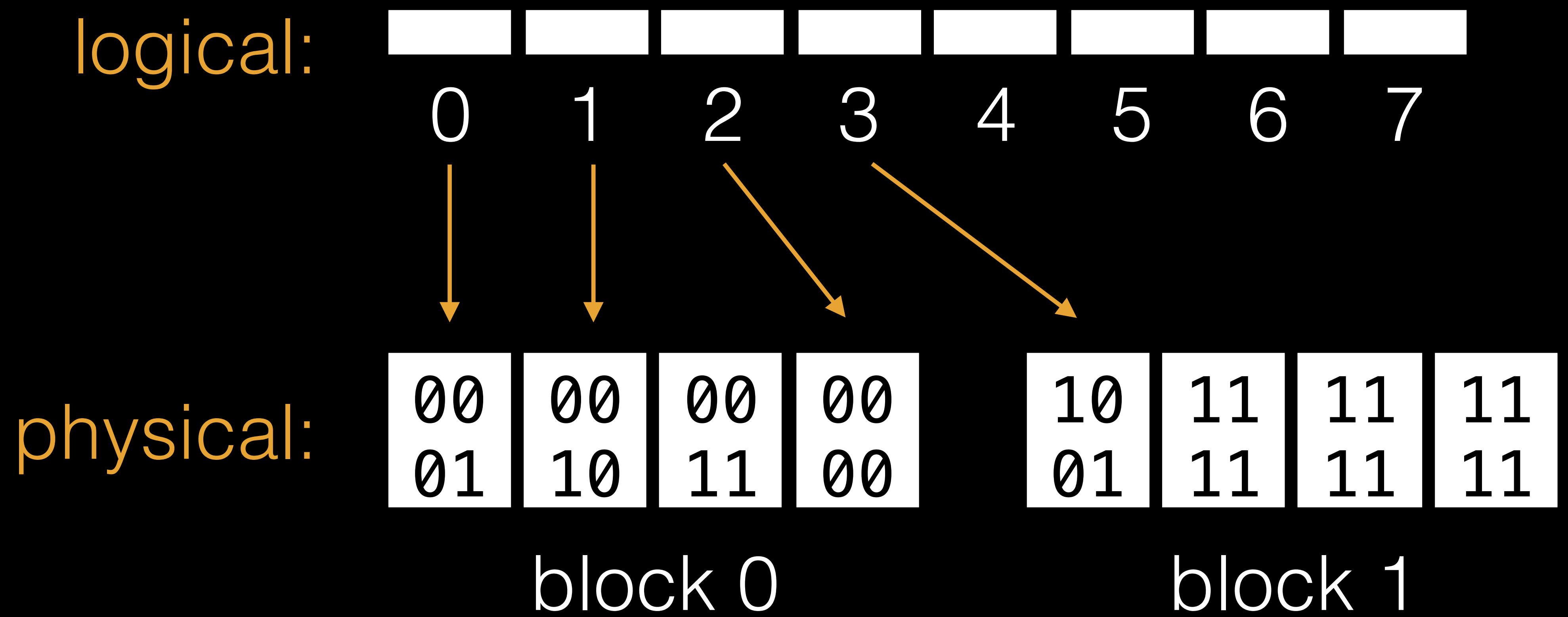
Block


1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111
1111

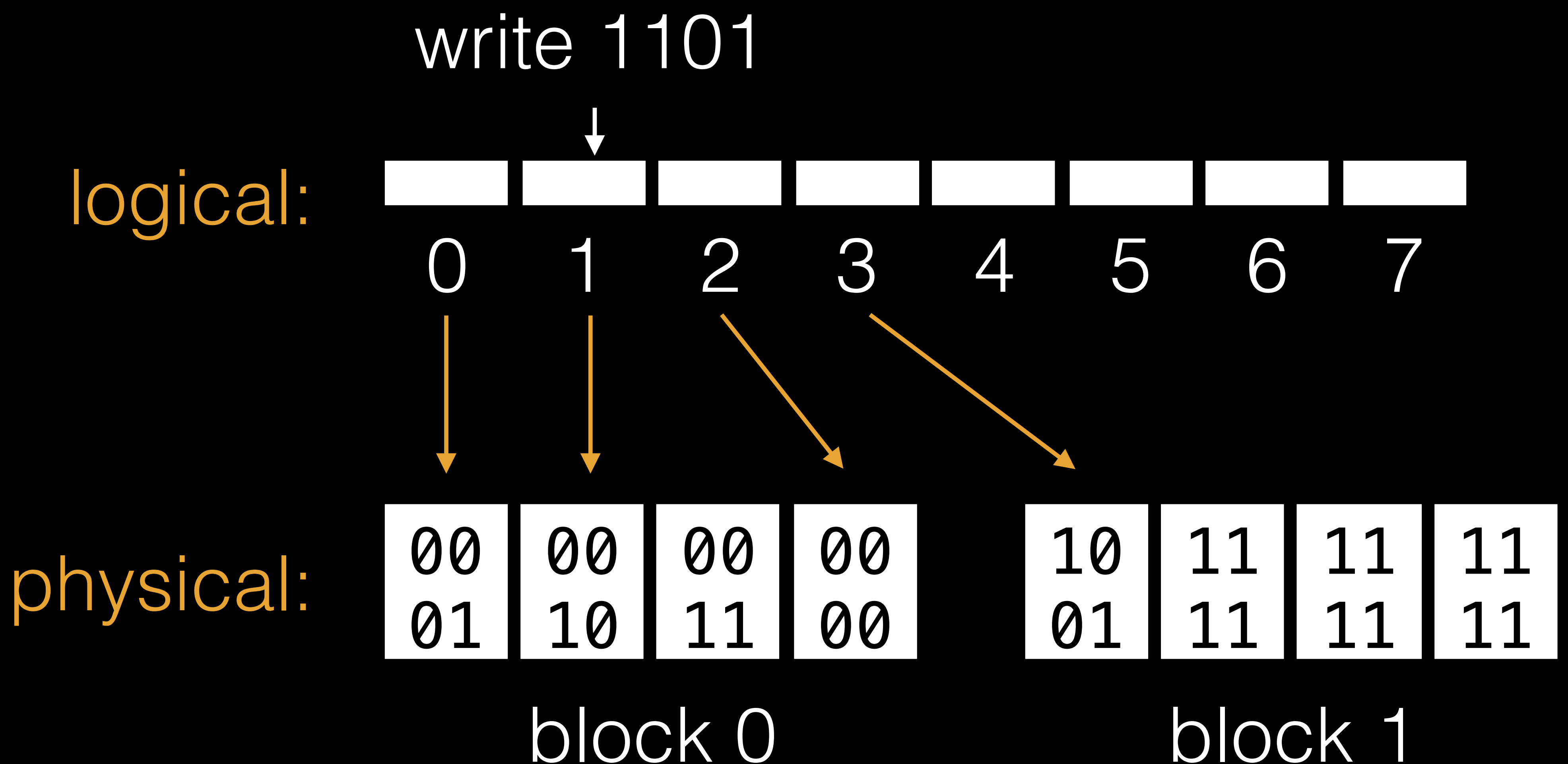
Traditional File Systems



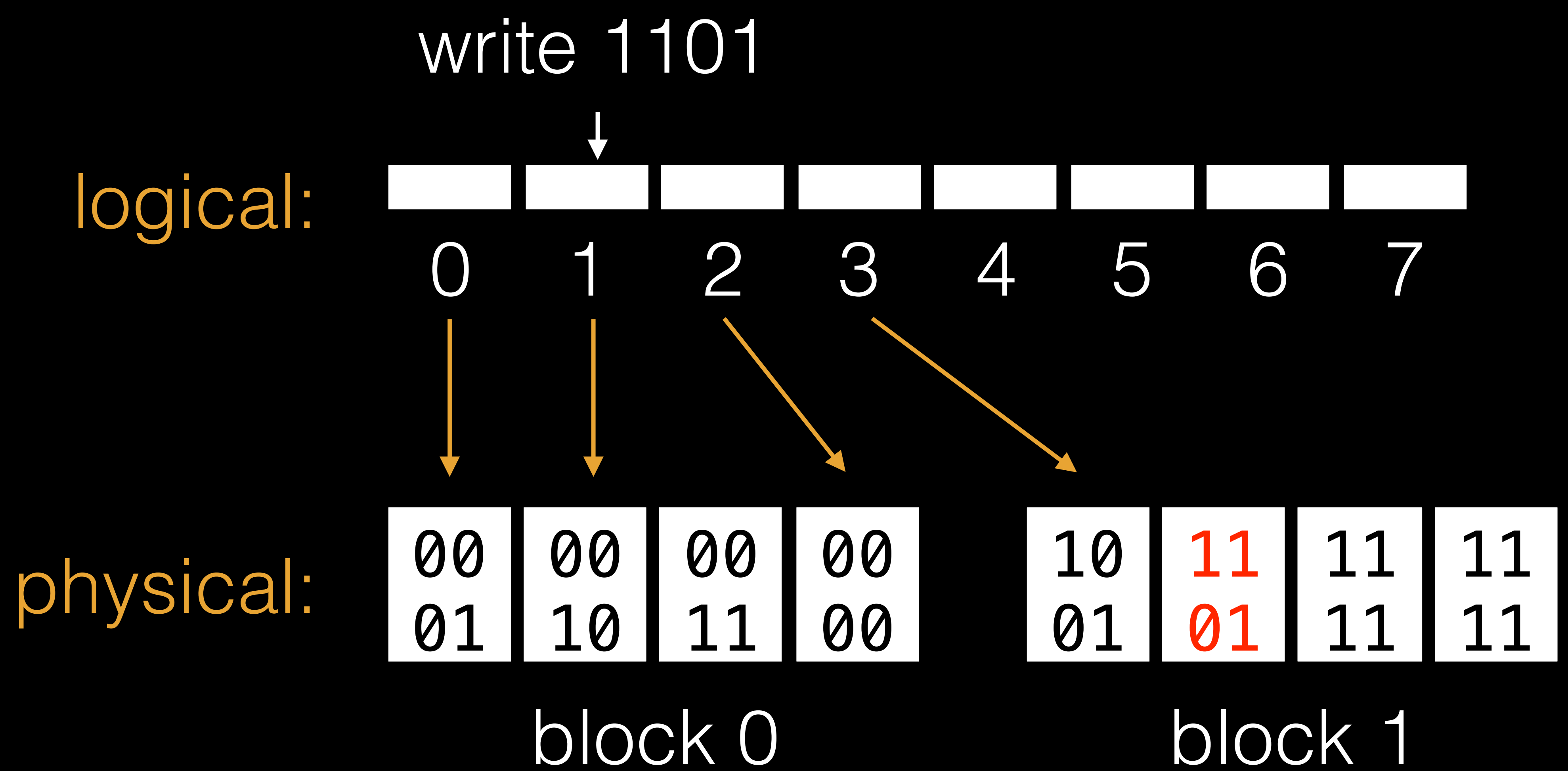
Flash Translation Layer



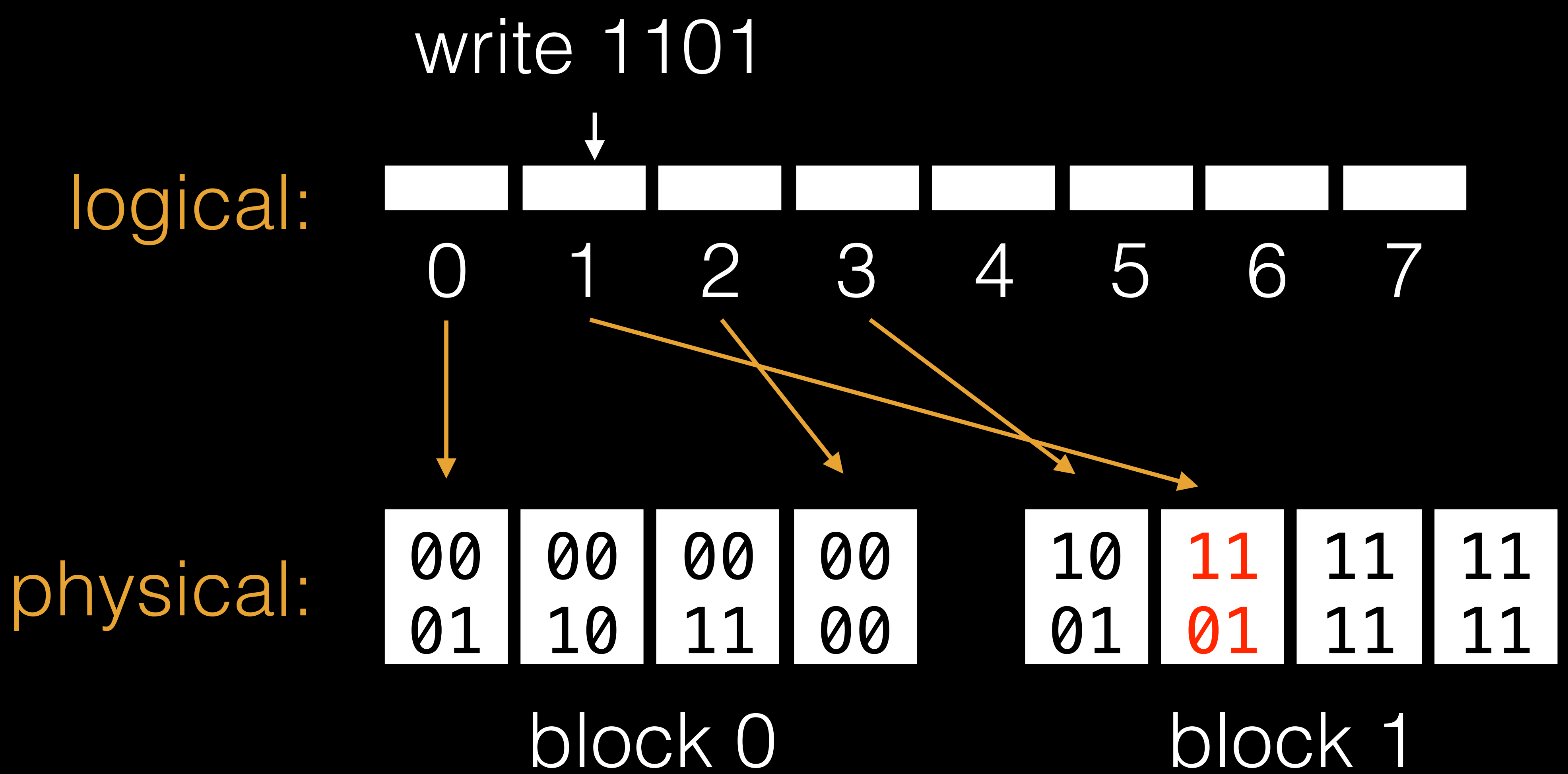
Flash Translation Layer



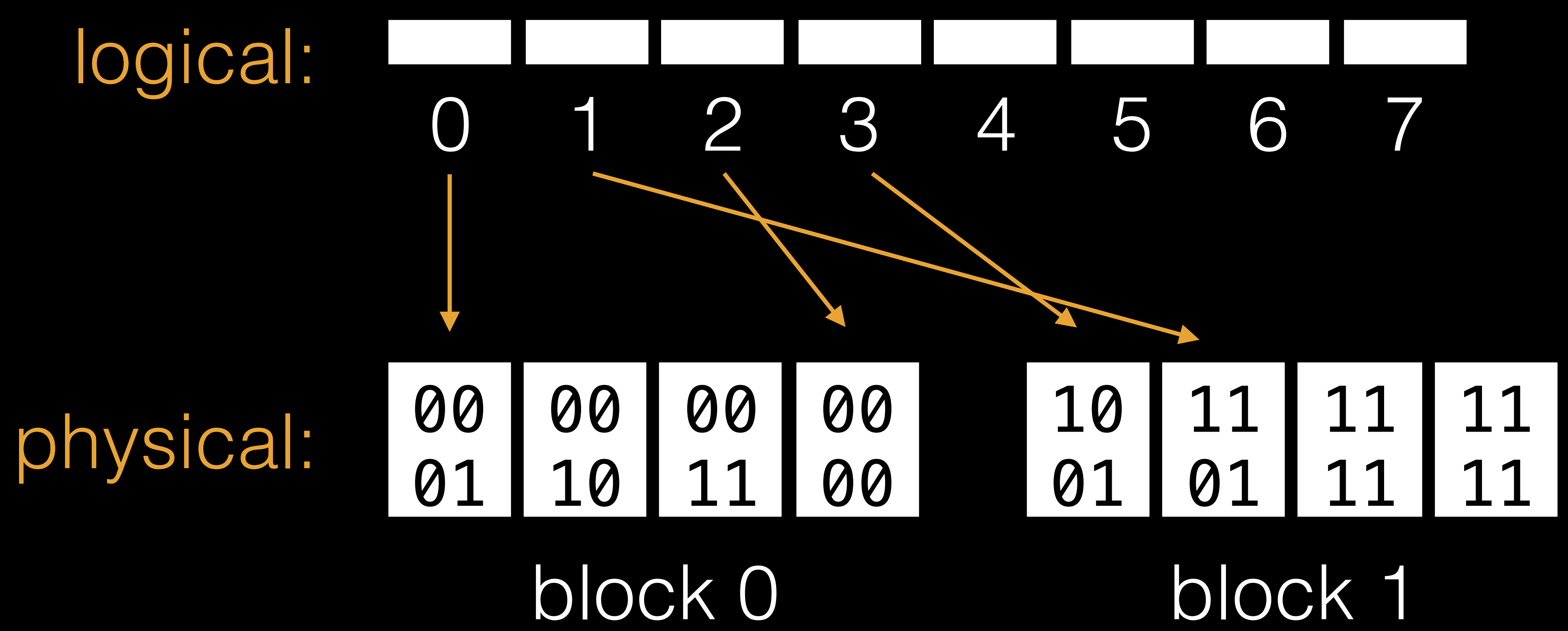
Flash Translation Layer



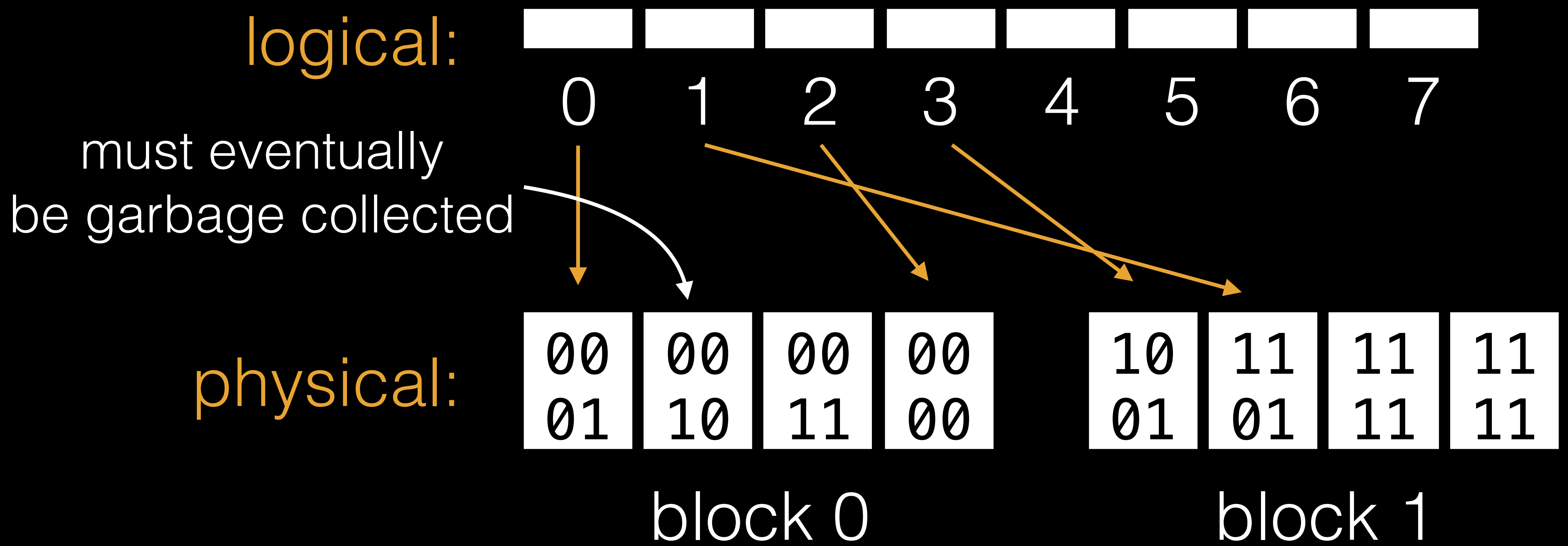
Flash Translation Layer



Flash Translation Layer



Flash Translation Layer



MapReduce Review

ZIP	Sale
53715	100
92245	10
53703	15
93422	45
99210	9
54622	20

ZIP	Sale
53715	100
92245	10
53703	15
93422	45
99210	9
54622	20

mapper 1

53715	100
92245	10
53703	15

mapper 2

93422	45
99210	9
54622	20

ZIP	Sale
53715	100
92245	10
53703	15
93422	45
99210	9
54622	20

mapper 1

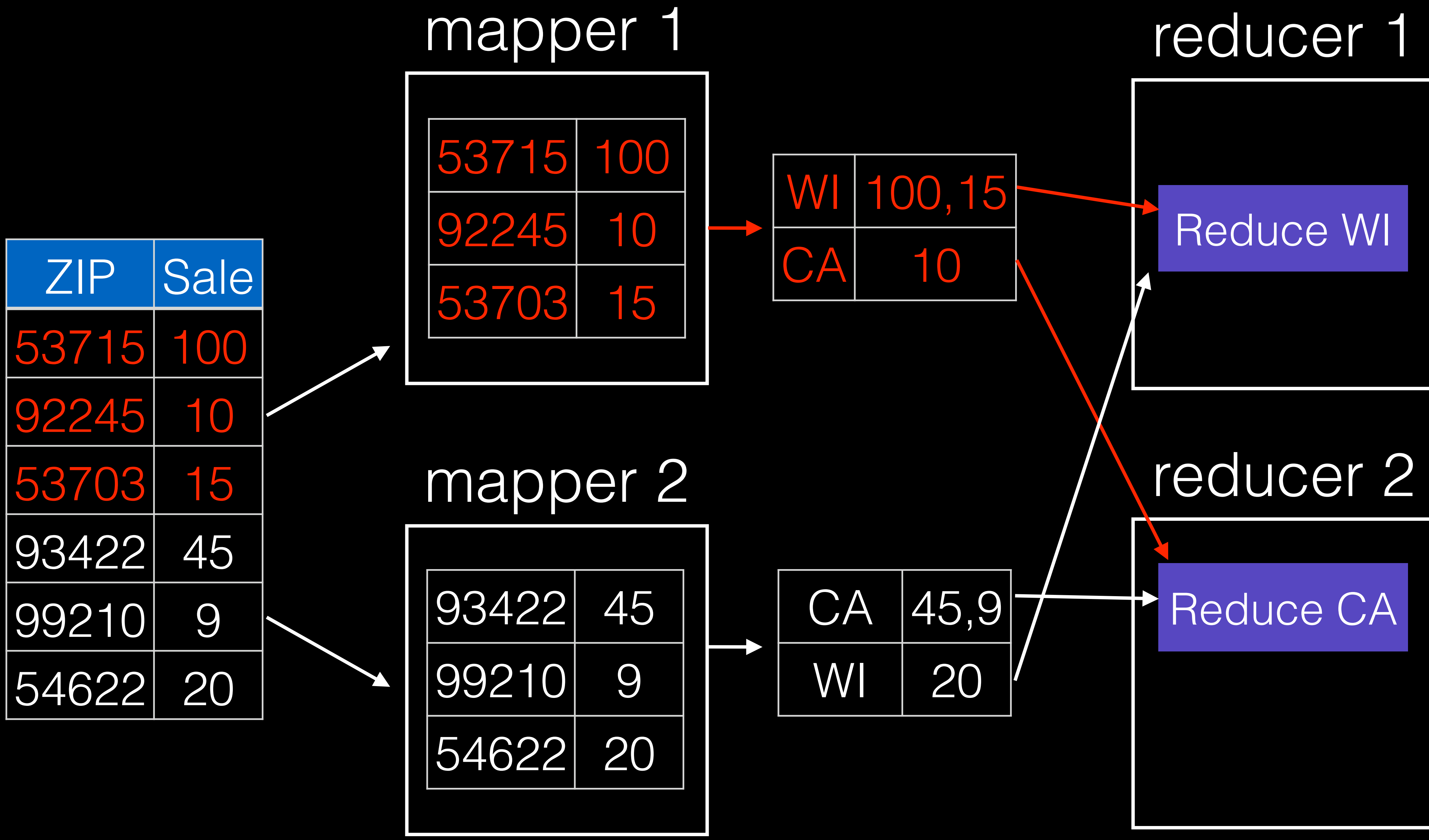
53715	100
92245	10
53703	15

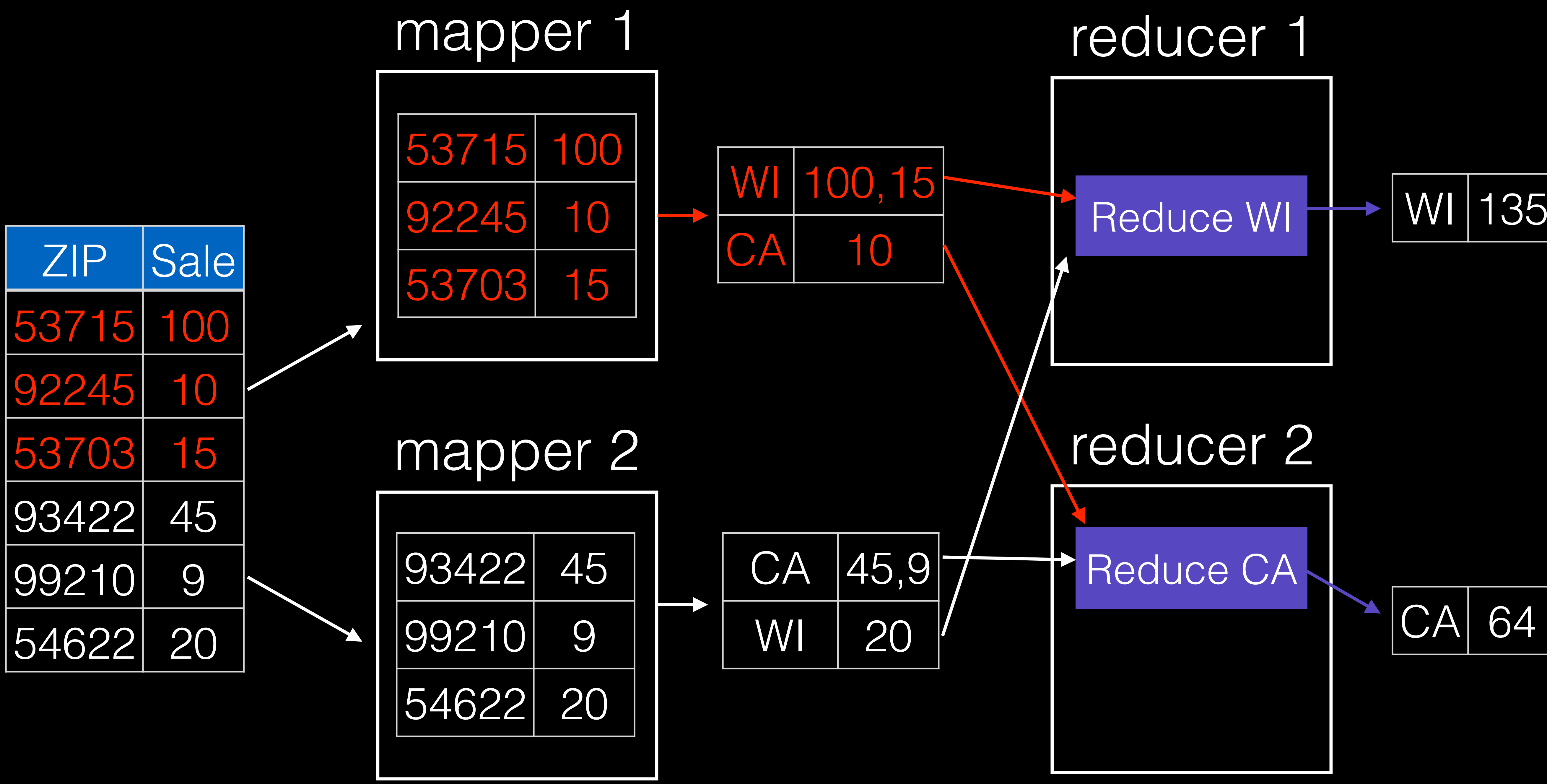
WI	100,15
CA	10

mapper 2

93422	45
99210	9
54622	20

CA	45,9
WI	20


mapper 1
reducer 1
ZIP
Sale
53715
100
92245
10
53703
15
93422
45
99210
9
54622
20
WI
100,15
CA
10
Reduce WI
mapper 2
reducer 2
CA
45,9
WI
20
Reduce CA



ZIP
Sale
53715
100
92245
10
53703
15
93422
45
99210
9
54622
20
mapper 1
mapper 2
WI
100,15
CA
10
CA
45,9
WI
20
reducer 1
reducer 2
Reduce WI
Reduce CA
WI
135
CA
64

```
public void map(LongWritable key, Text value) {
   String line = value.toString();
   StringToke st = new StringToke(line);
   while (st.hasMoreTokens())
      output.collect(st.nextToken(), 1);
}

public void reduce(Text key,
                   Iterator<IntWritable> values) {
  int sum = 0;
  while (values.hasNext())
    sum += values.next().get();
  output.collect(key, sum);
}
```

WordCount

Search Engines

Search Engine Goal

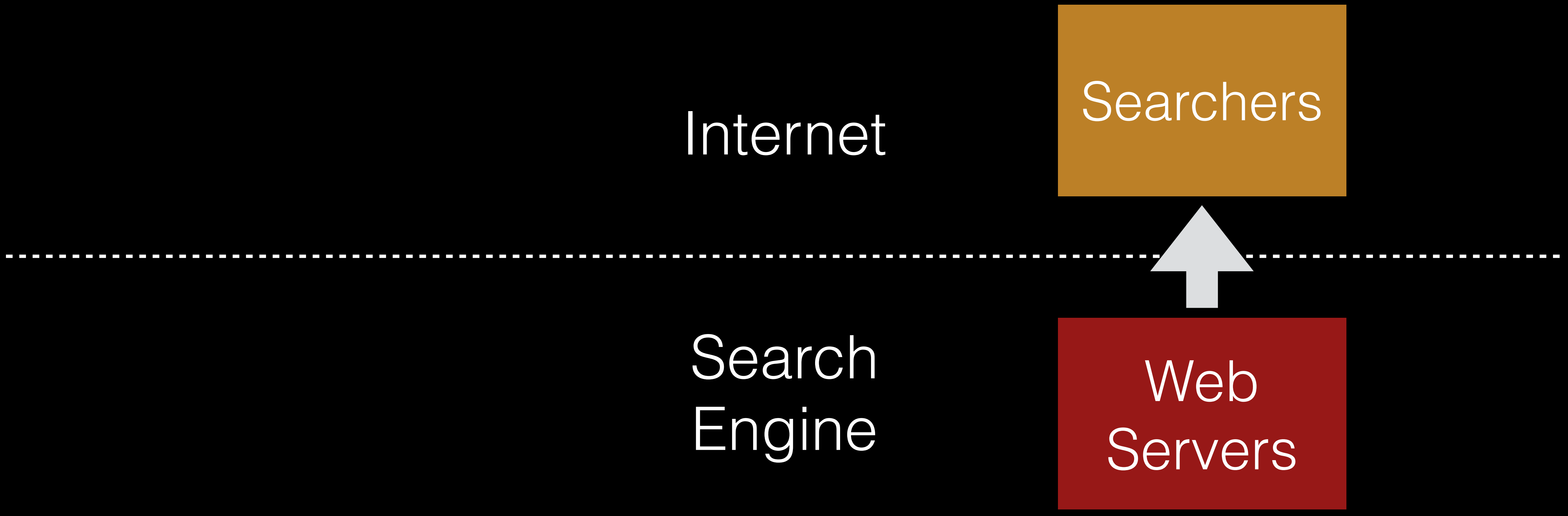
Users should be able to enter search phrases.

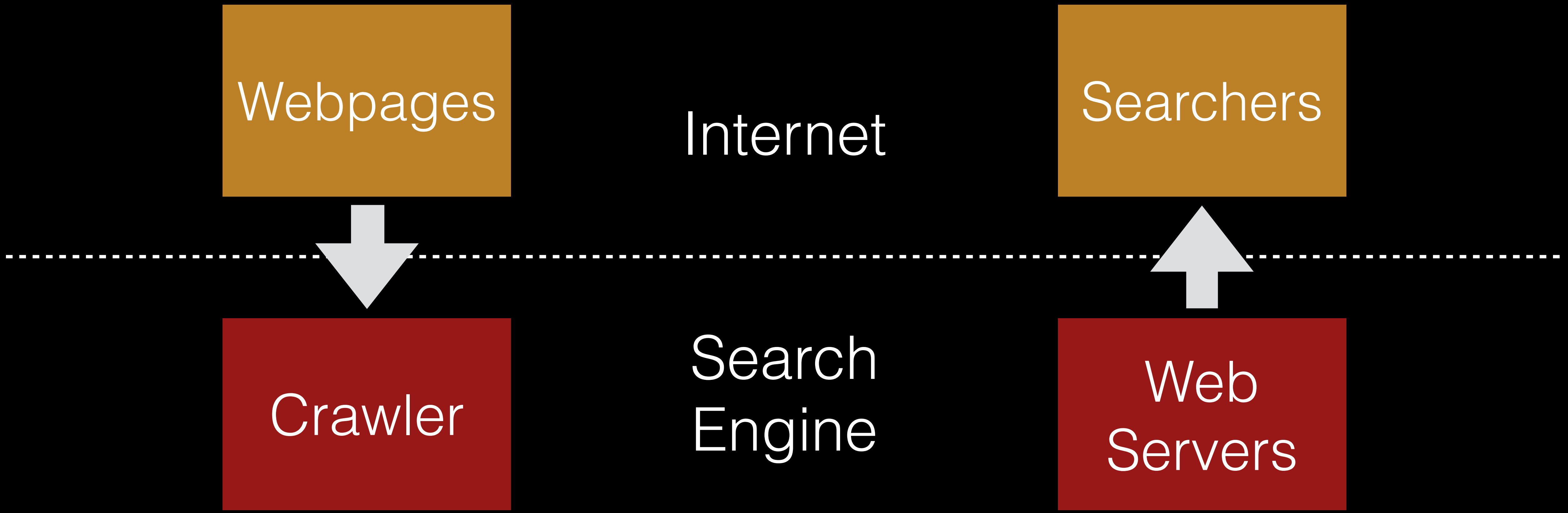
Want to return results that are:
- high quality (how to judge?)
- relevant

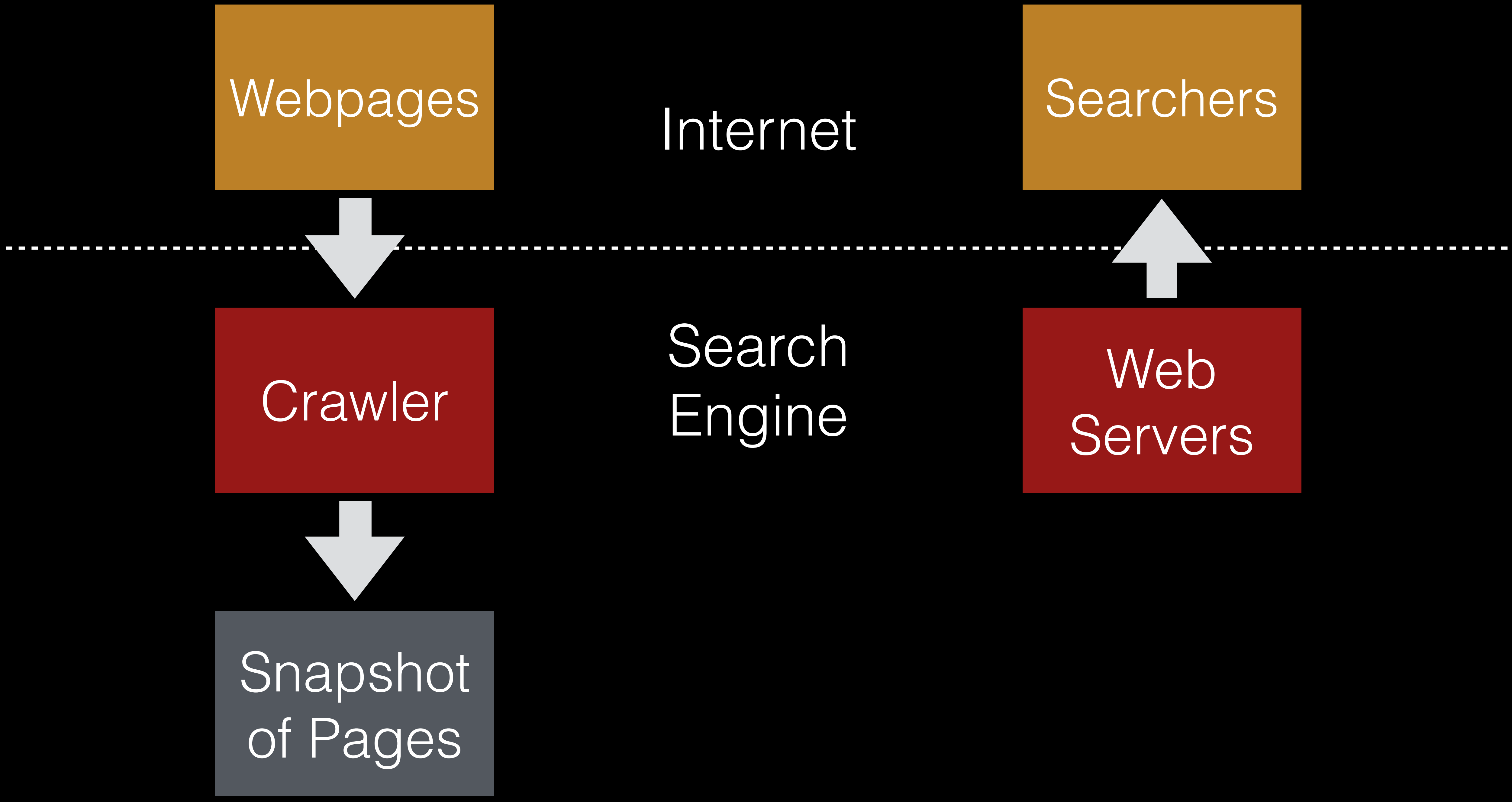
It's ok to do a lot of processing online, but searches must be fast!

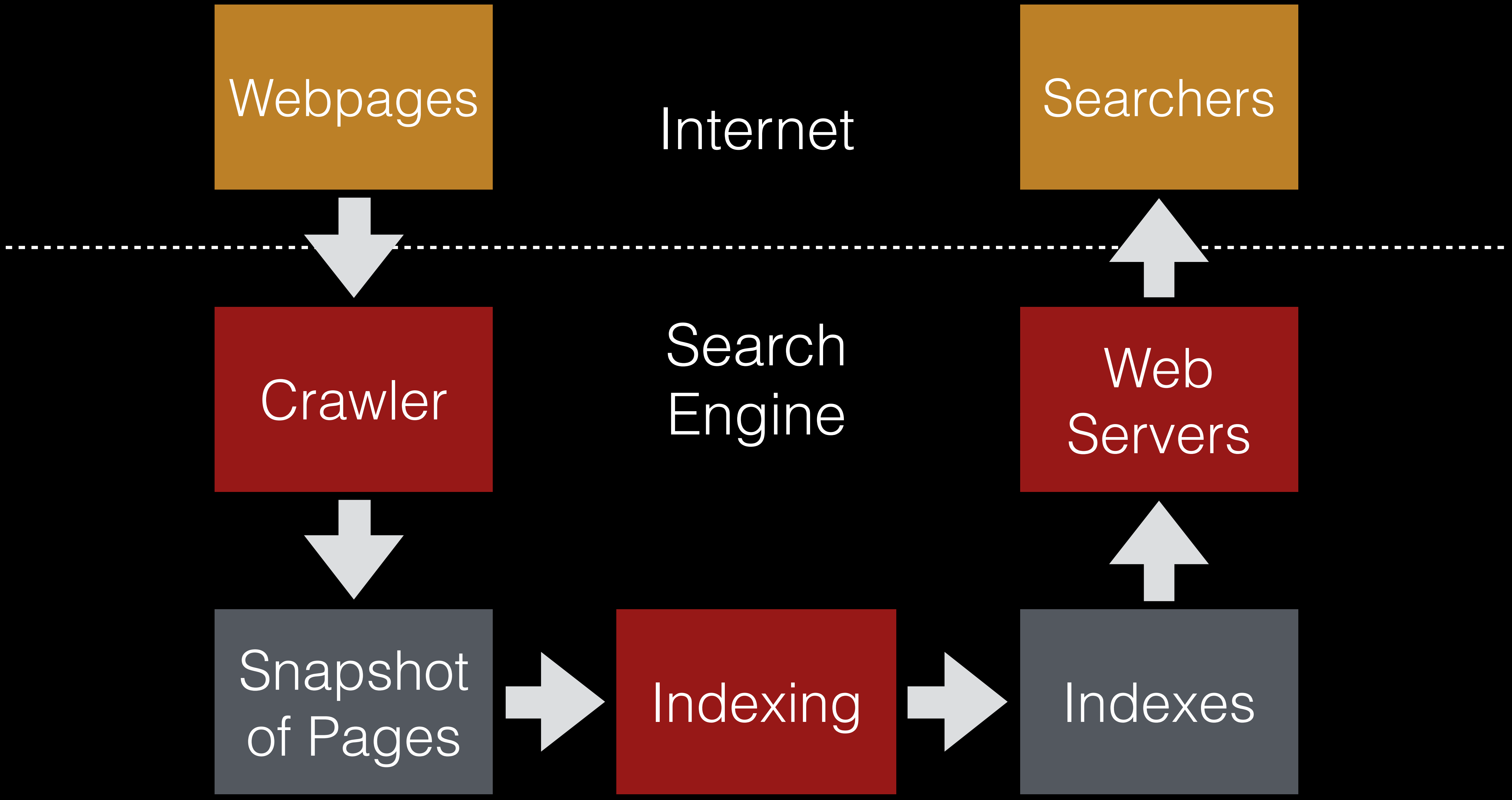
Internet

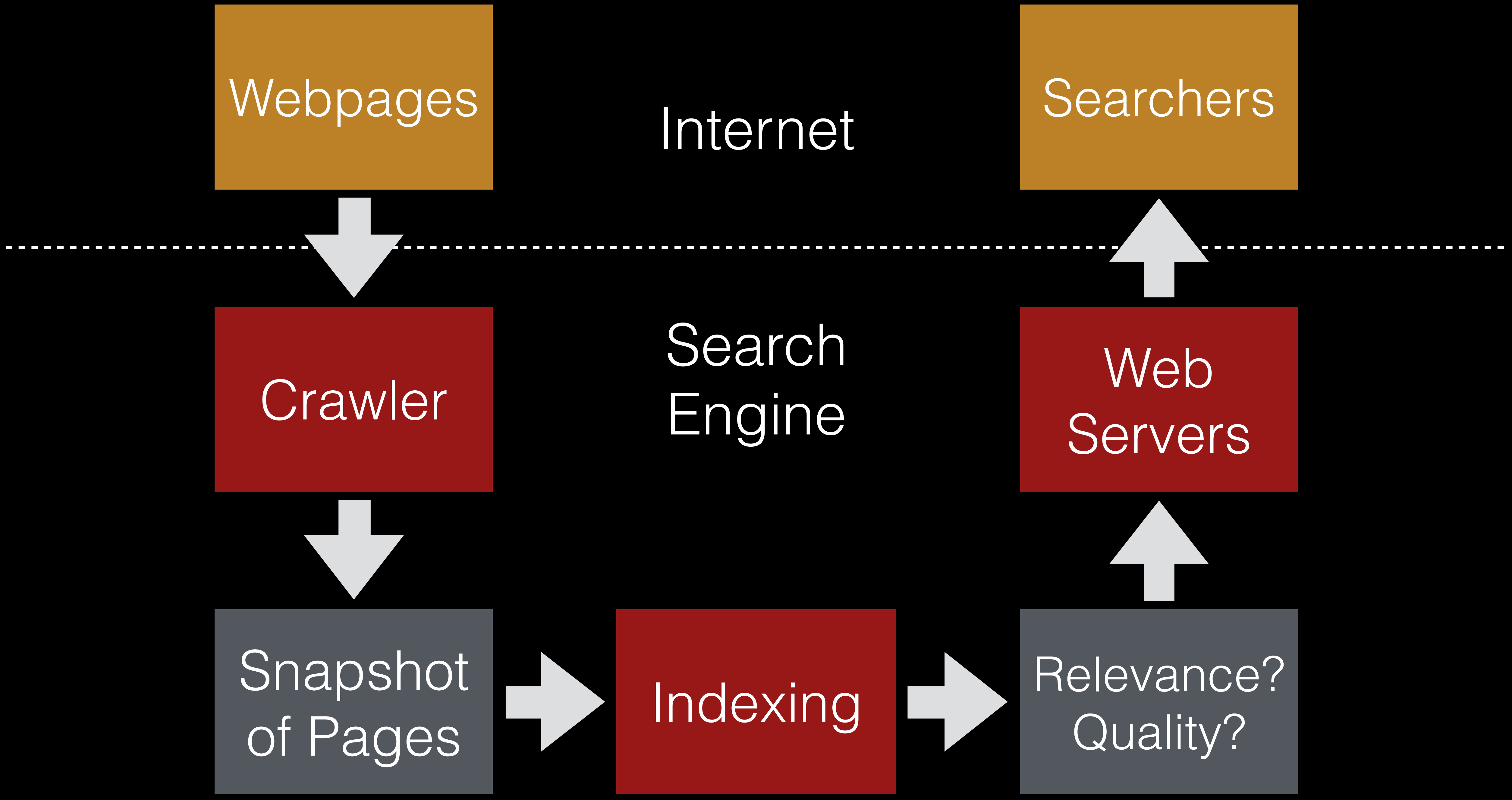
Search
Engine

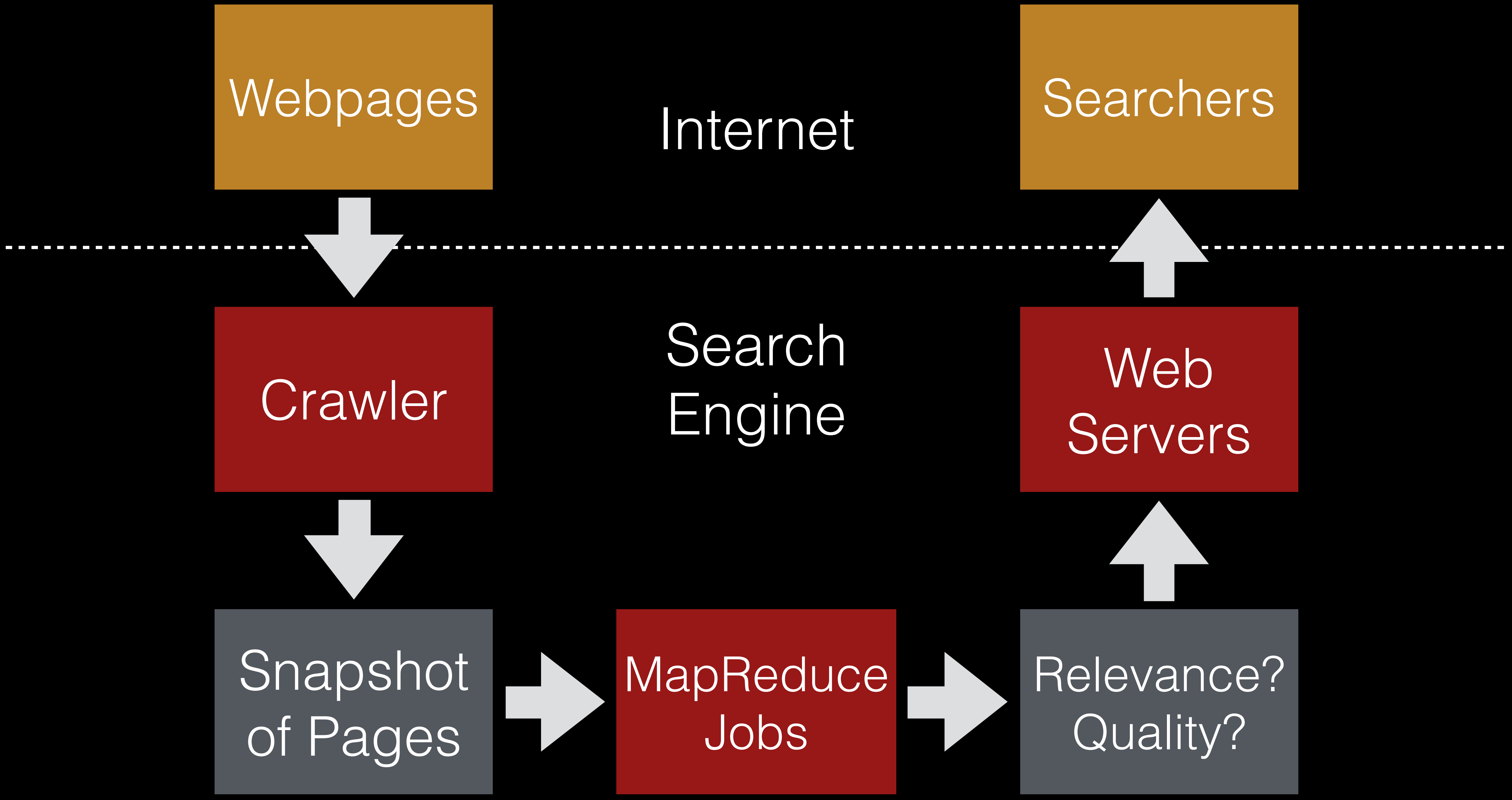

Internet
Searchers
Search Engine
Web Servers


Webpages
Internet
Searchers
Crawler
Search
Engine
Web
Servers


Webpages
Internet
Searchers
Crawler
Search
Engine
Web
Servers
Snapshot
of Pages


Webpages
Internet
Searchers
Crawler
Search Engine
Web Servers
Snapshot of Pages
Indexing
Indexes


Webpages
Internet
Searchers
Crawler
Search Engine
Web Servers
Snapshot of Pages
Indexing
Relevance? Quality?

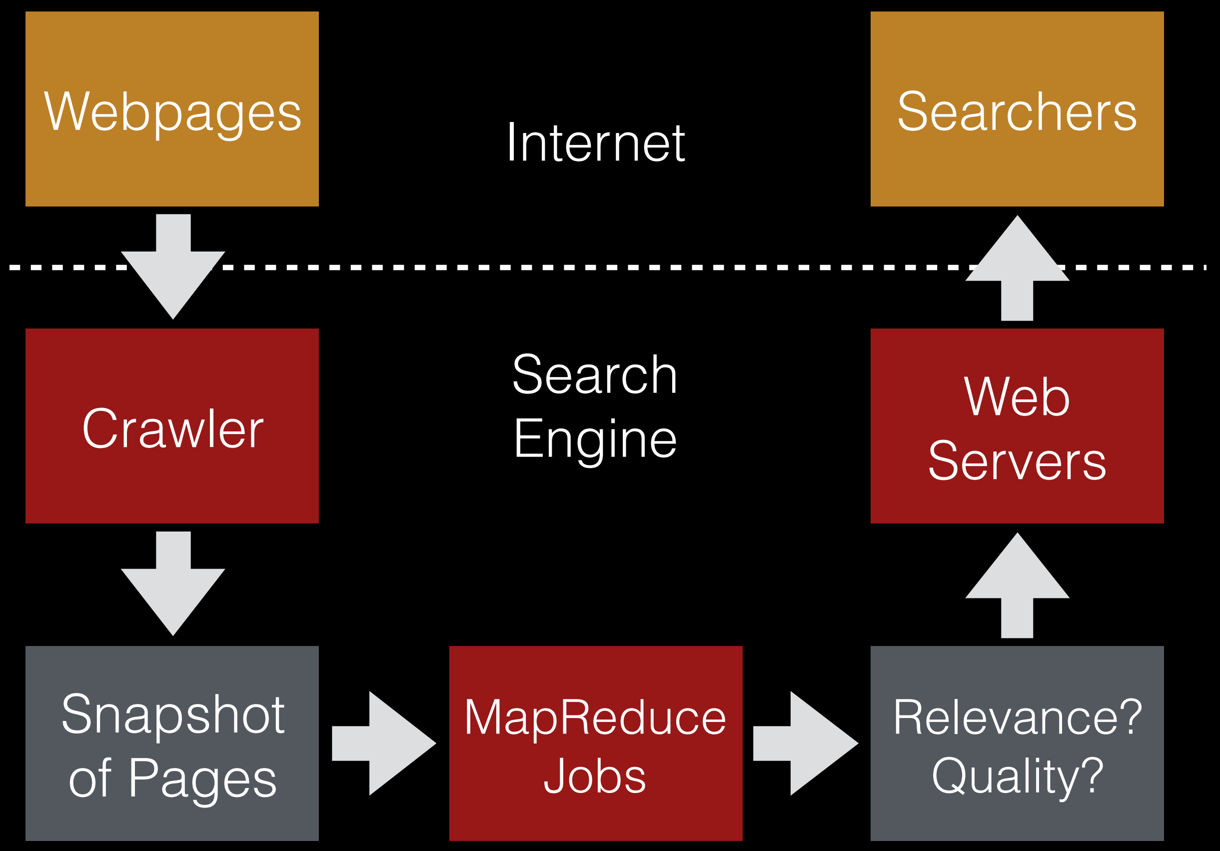

Webpages
Internet
Searchers
Crawler
Search Engine
Web Servers
Snapshot of Pages
MapReduce Jobs
Relevance? Quality?

Outline

Web Crawling

Indexing
- PageRank
- Inverted Indexes

Searching

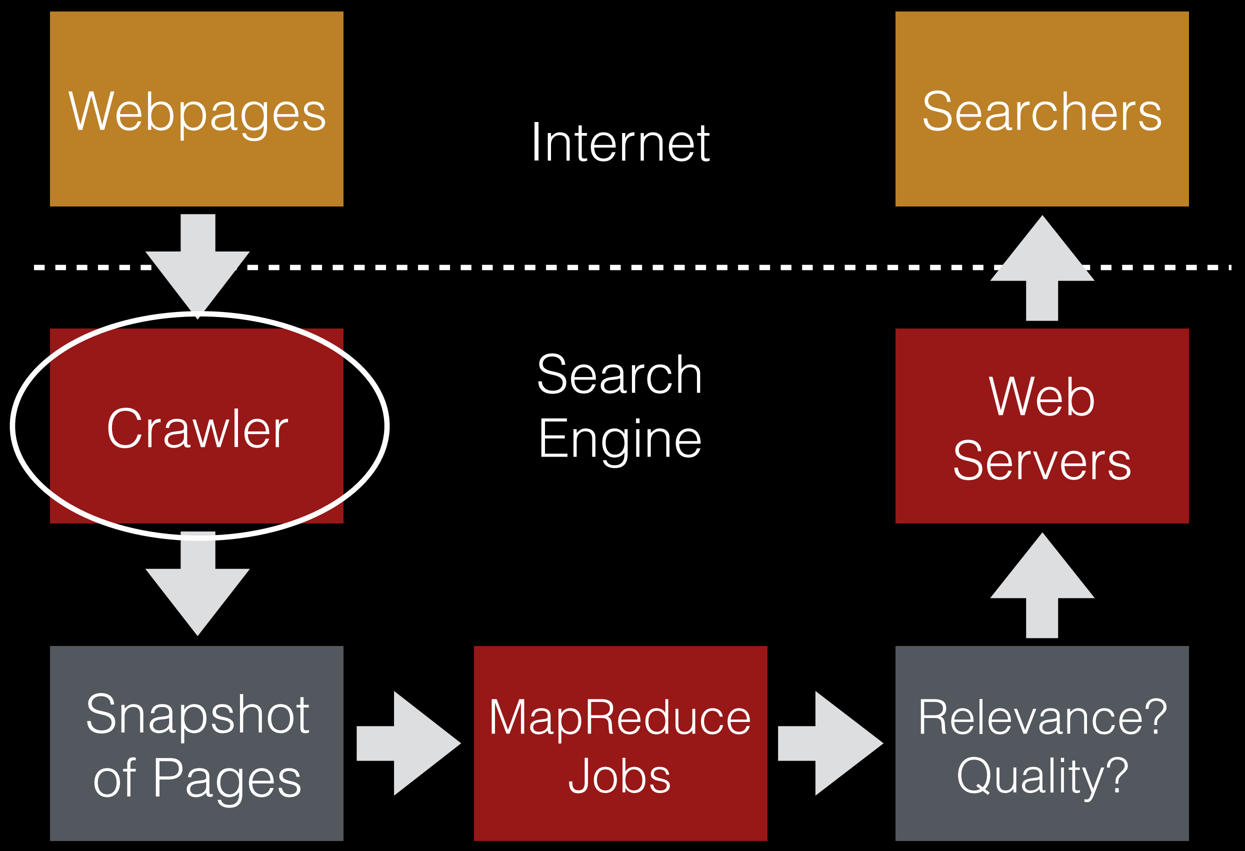


Outline

Web Crawling

Indexing
- PageRank
- Inverted Indexes

Searching



Web Crawler

Maintain list of pages to crawl.

Grabbing/saving a copy removes work from list.

Fetched pages may have more links, leading to more work.

Fetching a Page

1. convert domain name to IP address.

2. fetch page from server at IP address.

High-performance crawlers maintain a very large DNS cache to minimize step 1.

Spider Traps

Server returns data so that page example.com/N has a link to example.com/(N+1).

From crawler's perspective, web is infinite!

Prioritize via heuristics (avoid dynamic content) and quality rankings (later).

robots.txt

robots.txt file can tell crawlers not to crawl. Example:

```
User-agent: googlebot         # all Google services
Disallow: /private/           # disallow this directory

User-agent: googlebot-news    # only the news service
Disallow: /                   # disallow everything

User-agent: *                 # any robot
Disallow: /something/         # disallow this directory
```

Some web developers set up intentional spider traps to punish crawlers that ignore these.

example source: http://en.wikipedia.org/wiki/Robots_exclusion_standard

“Almost daily, we receive an email something like, ‘*Wow, you looked at a lot of pages from my web site. How did you like it?*’”

Sergey Brin + Lawrence Page

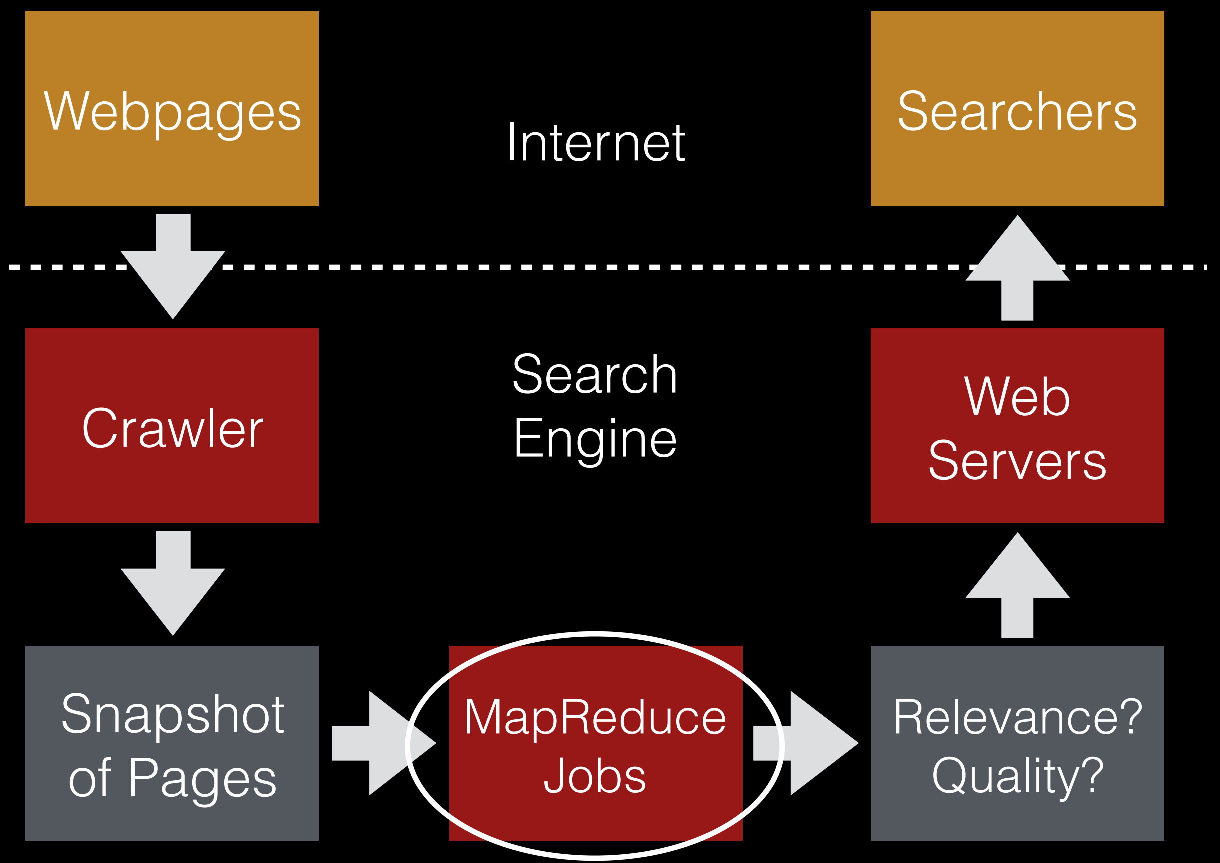
Source: The Anatomy of a Large-Scale Hypertextual Web Search Engine (1998)

Outline

Web Crawling

Indexing
- PageRank
- Inverted Indexes

Searching



Quality Problem

Web pages “proliferate free of quality control”.

Contrast with peer-reviewed academic papers.

Need to infer quality from the web graph.

Quality Problem

Web pages "proliferate free of quality control".

Contrast with peer-reviewed academic papers.

Need to infer quality from the web graph.

Give every page a singe PageRank score representing quality.

Strategy: Count Backlinks

Importance:

A = 1

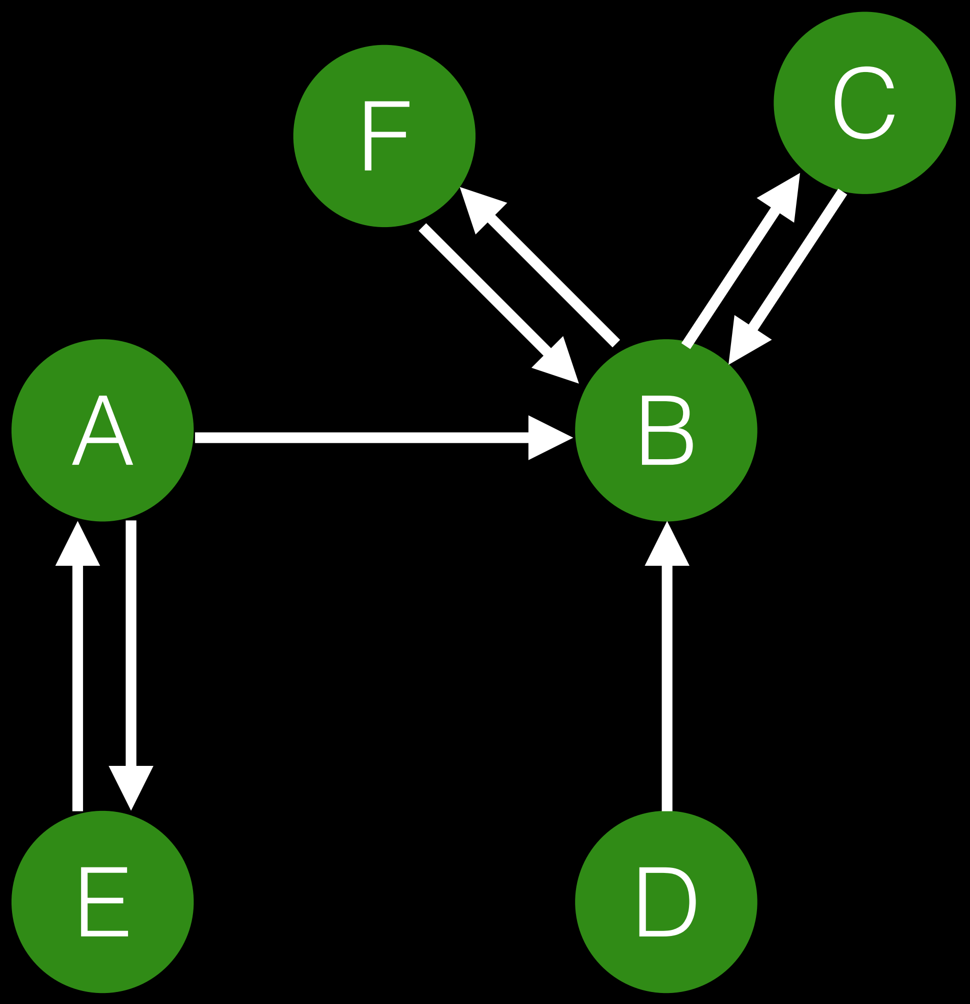
B = 4

C = 1

D = 0

E = 1

F = 1



Strategy: Count Backlinks

Importance:

A = 1

B = 4

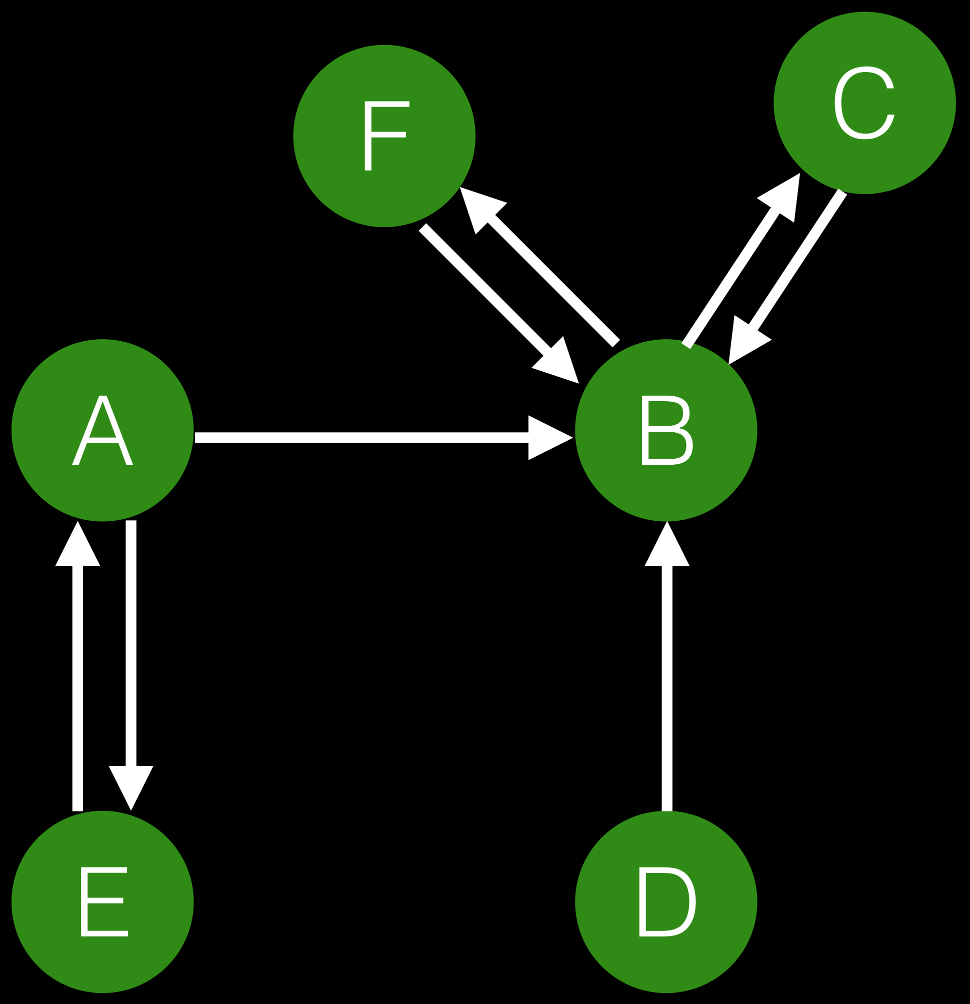
C = 1

D = 0

E = 1

F = 1

should A
get 2 “votes”?



Strategy: Count Backlinks

Importance:

A = 1

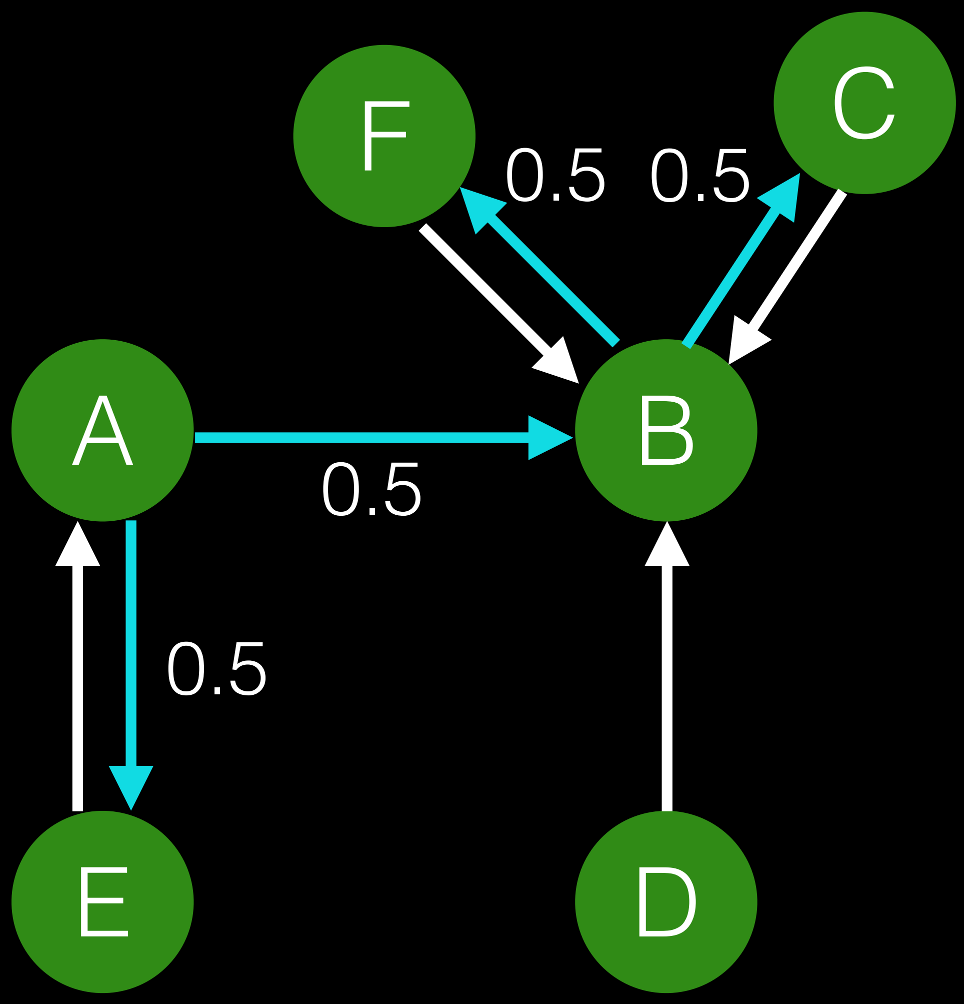
B = 3.5

C = 0.5

D = 0

E = 0.5

F = 0.5



Strategy: Count Backlinks

Importance:

A = 1

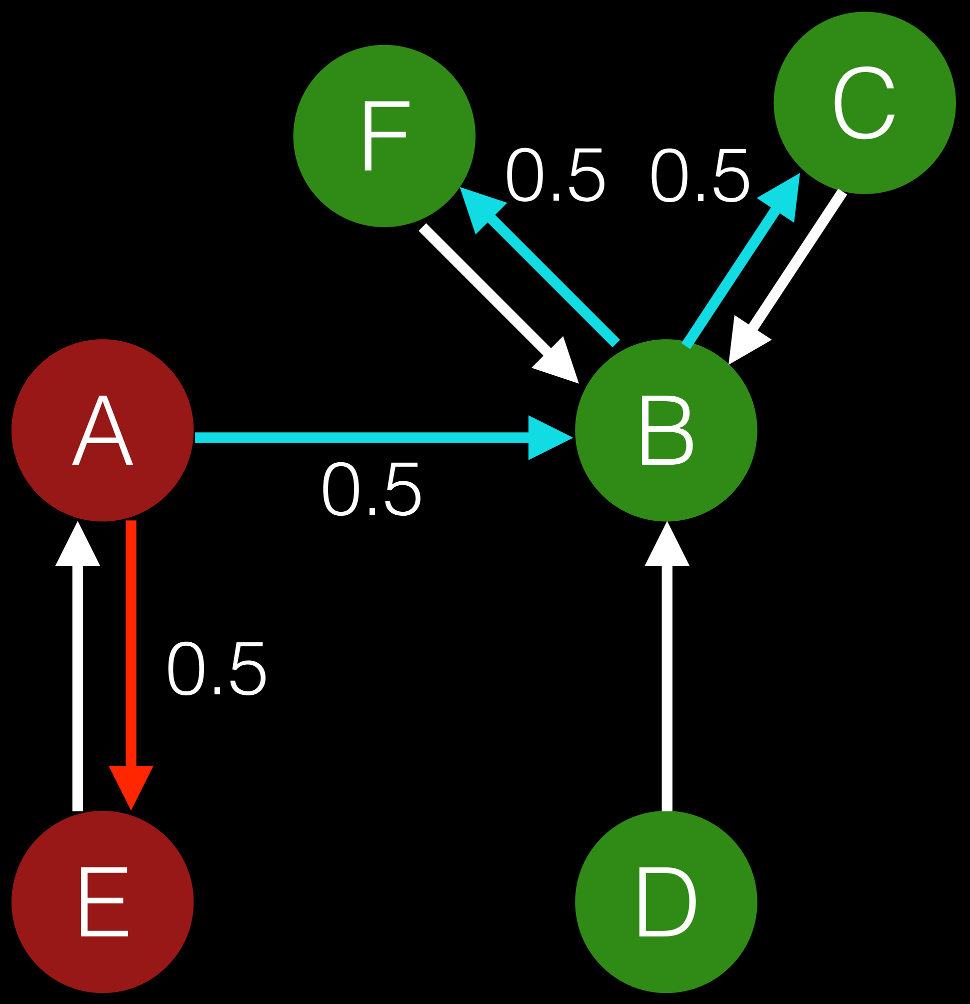
B = 3.5

C = 0.5

D = 0

E = 0.5 (from A's vote)

F = 0.5



Strategy: Count Backlinks

Importance:

A = 1

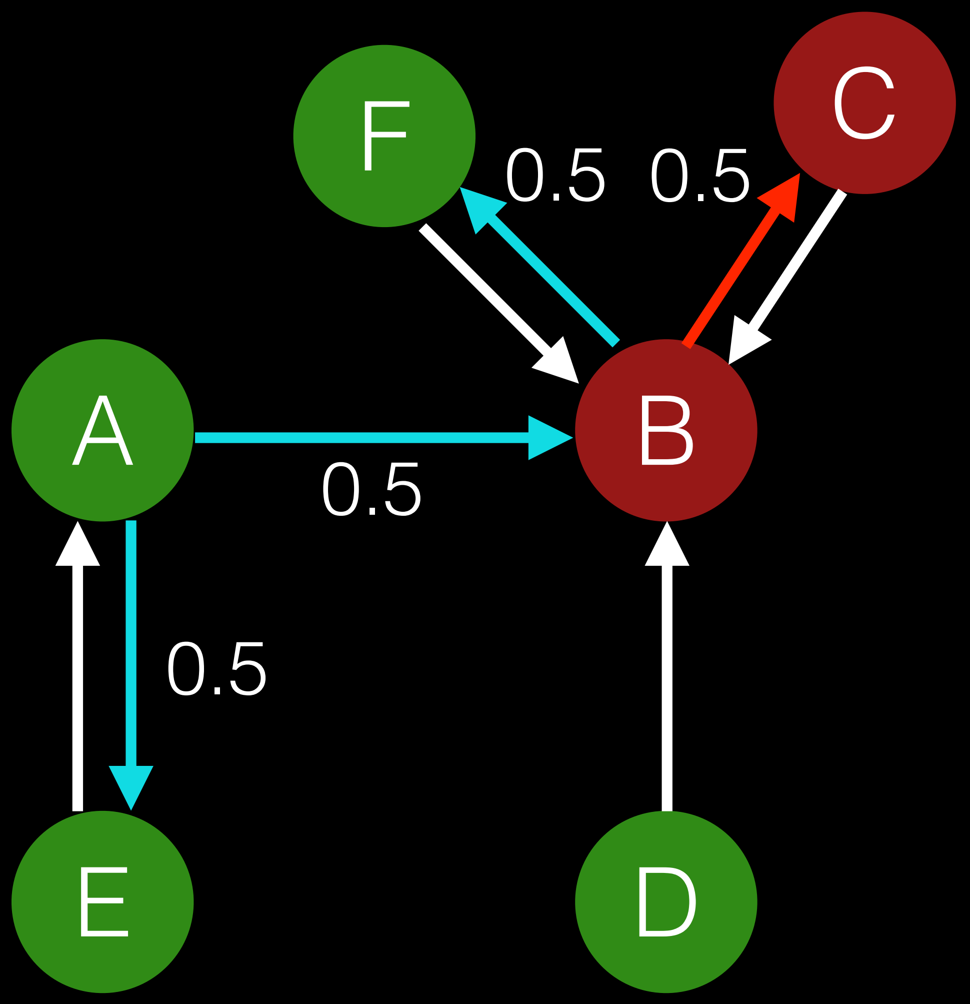
B = 3.5

C = 0.5 (from B's vote)

D = 0

E = 0.5 (from A's vote)

F = 0.5



Strategy: Count Backlinks

Importance:

A = 1

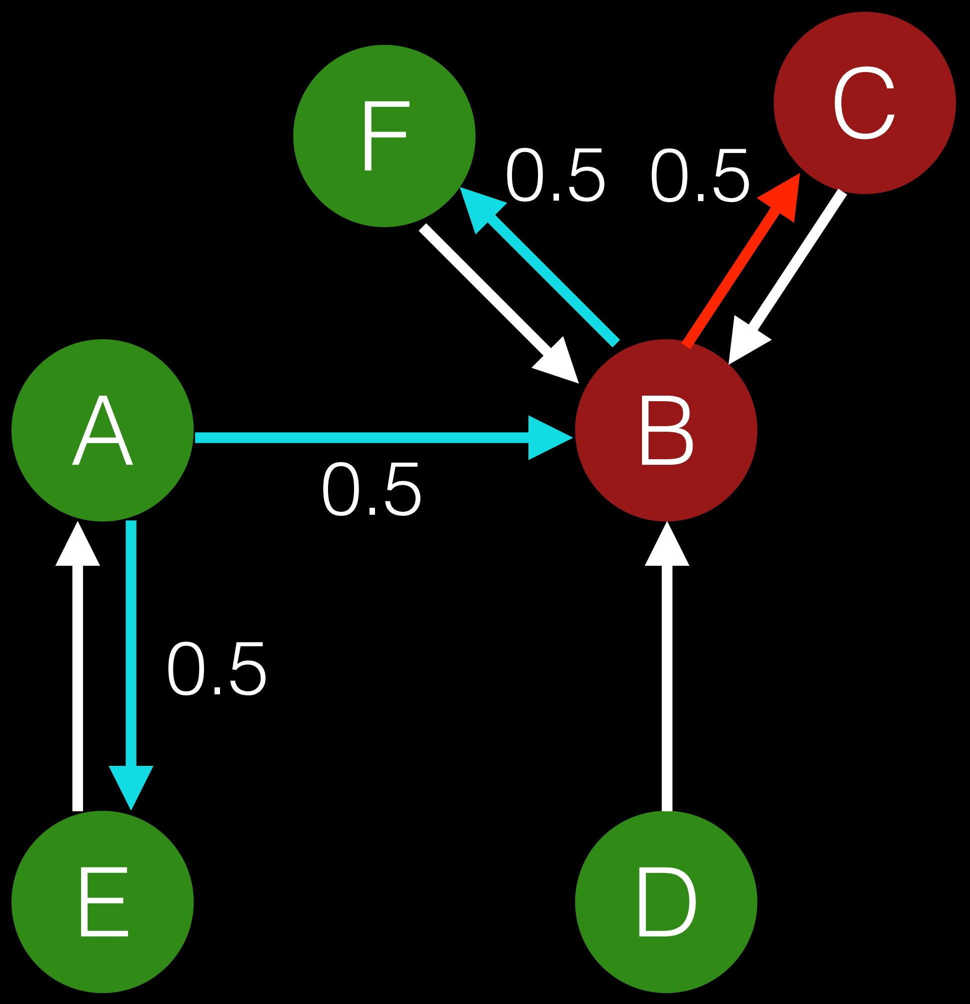
B = 3.5

C = 0.5 (from B's vote)

D = 0

E = 0.5 (from A's vote)

F = 0.5



Why do A and B get same votes? B is more important.

Circular Votes

Want: number of votes you get determines number of votes you give.

Problem: changing A’s votes changes B’s votes changes A’s votes…

Circular Votes

Want: number of votes you get determines number of votes you give.

Problem: changing A's votes changes B's votes changes A's votes…

Fortunately, if you just keep updating every PageRank, it eventually converges.

Convergence Goal (Simplified)

Rank(x) = “sum of all votes for x”

“x” is a page, Rank(x) is its PageRank.

Convergence Goal (Simplified)

$$\text{Rank}(x) = \sum_{y \in \text{LinksTo}(x)} \text{“y's vote for x”}$$

LinksTo(x) is the set of all pages linking to x.

Convergence Goal (Simplified)

$$\text{Rank}(x) = \sum_{y \in \text{LinksTo}(x)} \frac{\text{Rank}(y)}{N_y}$$

N_y is the number of links from y to other pages.

Convergence Goal (Simplified)

$$\text{Rank}(x) = c \sum_{y \in \text{LinksTo}(x)} \frac{\text{Rank}(y)}{N_y}$$

Normalize with “c” to get desired amount of “rank” in system.

Convergence Goal (Simplified)

keep updating rank for every page
until ranks stop changing much

$$\mathrm{Rank}(x) = c \sum_{y \in \mathrm{LinksTo}(x)} \frac{\mathrm{Rank}(y)}{N_y}$$

Intuition: Random Surfer

Imagine!

1. a bunch of web surfers start on various pages
2. they randomly click links, forever
3. you measure webpage visit frequency

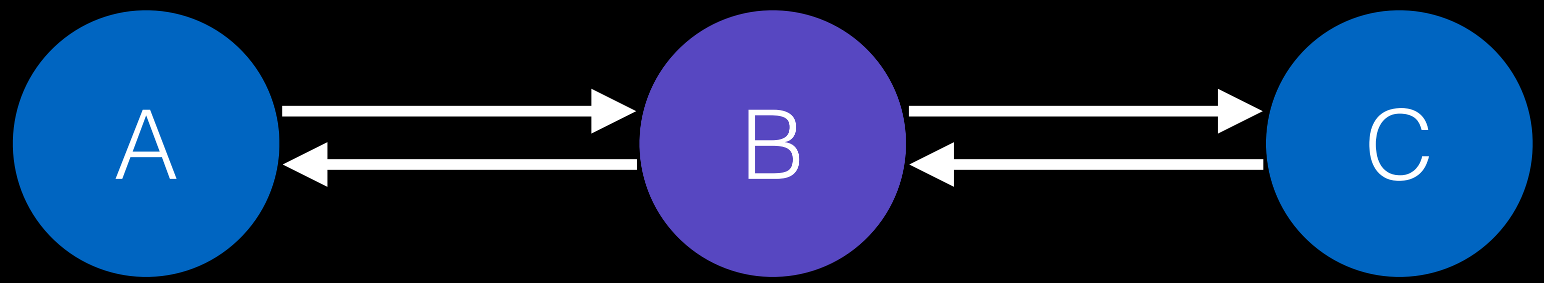
Intuition: Random Surfer

Imagine!

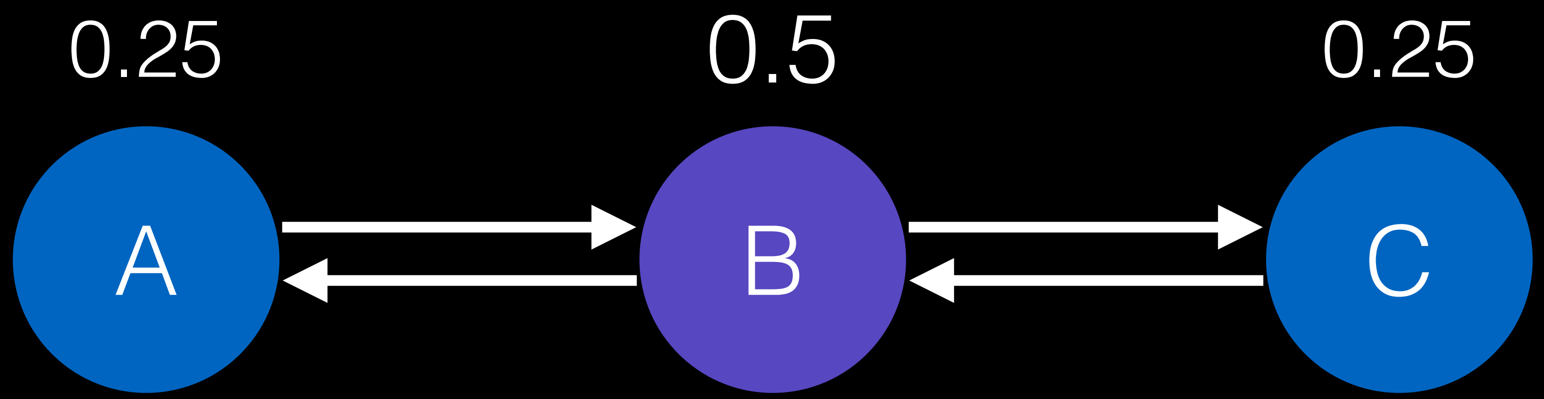
1. a bunch of web surfers start on various pages
2. they randomly click links, forever
3. you measure webpage visit frequency

Visit frequency will be proportional to PageRank.

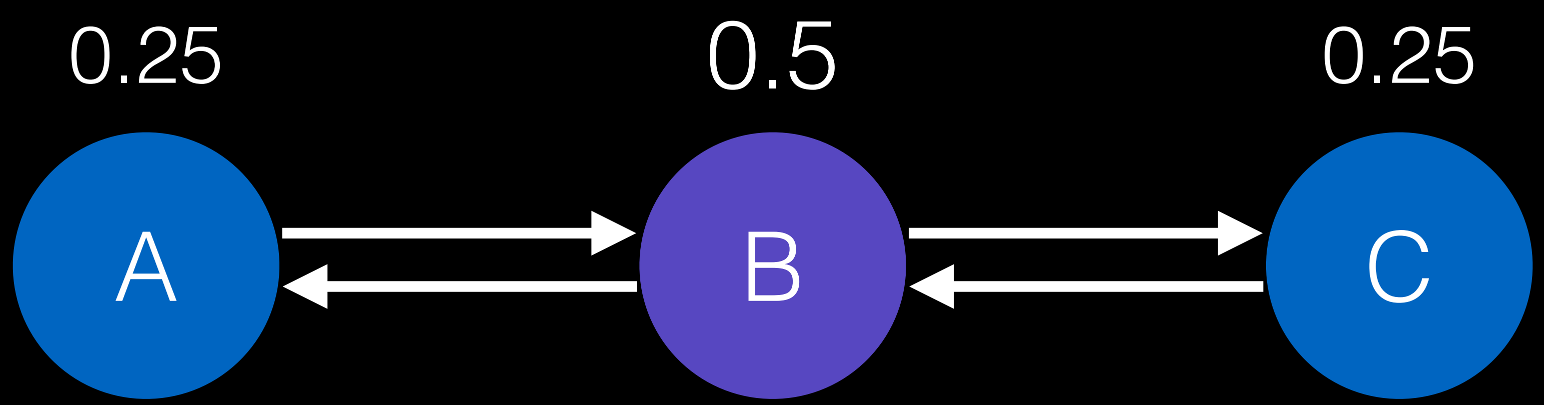
Graph 1



Graph 1



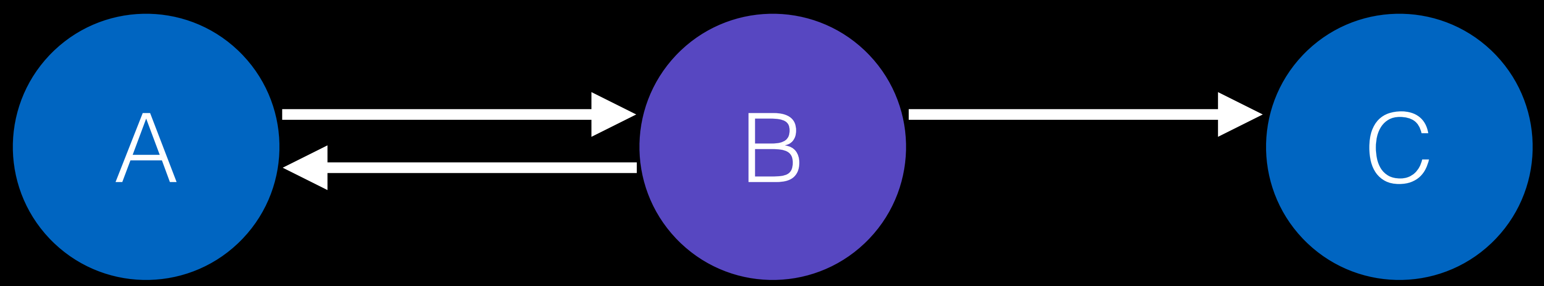
Graph 1



Rank(B) = (0.25 / 1) + (0.25 / 1) = 0.5
Rank(A) = (0.5 / 2) = 0.25
Rank(C) = (0.5 / 2) = 0.25

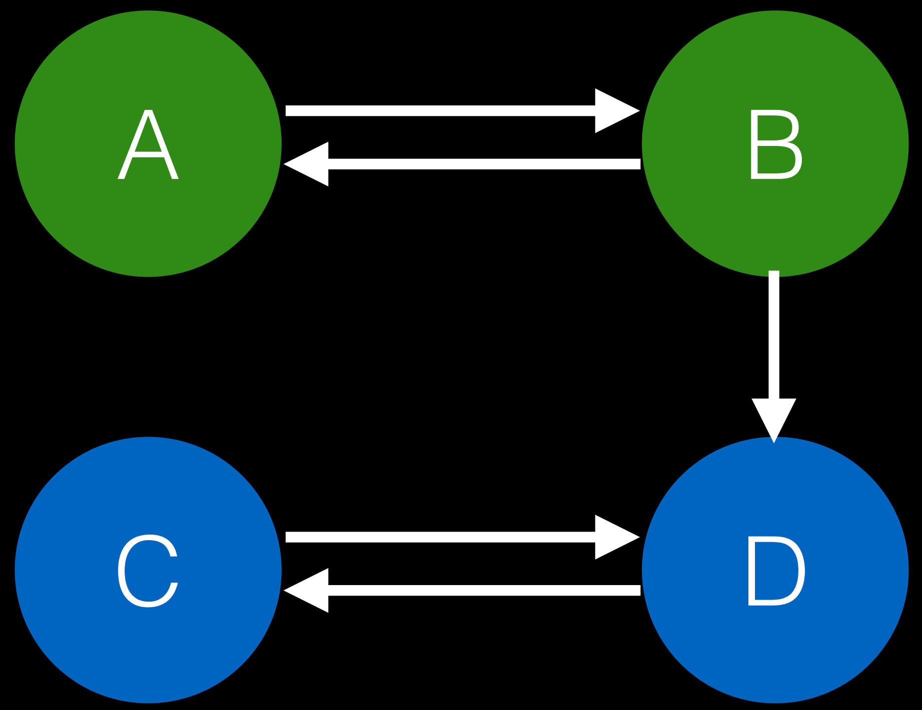
$$\text{Rank}(x) = c \sum_{y \in \text{LinksTo}(x)} \frac{\text{Rank}(y)}{N_y}$$

Graph 2



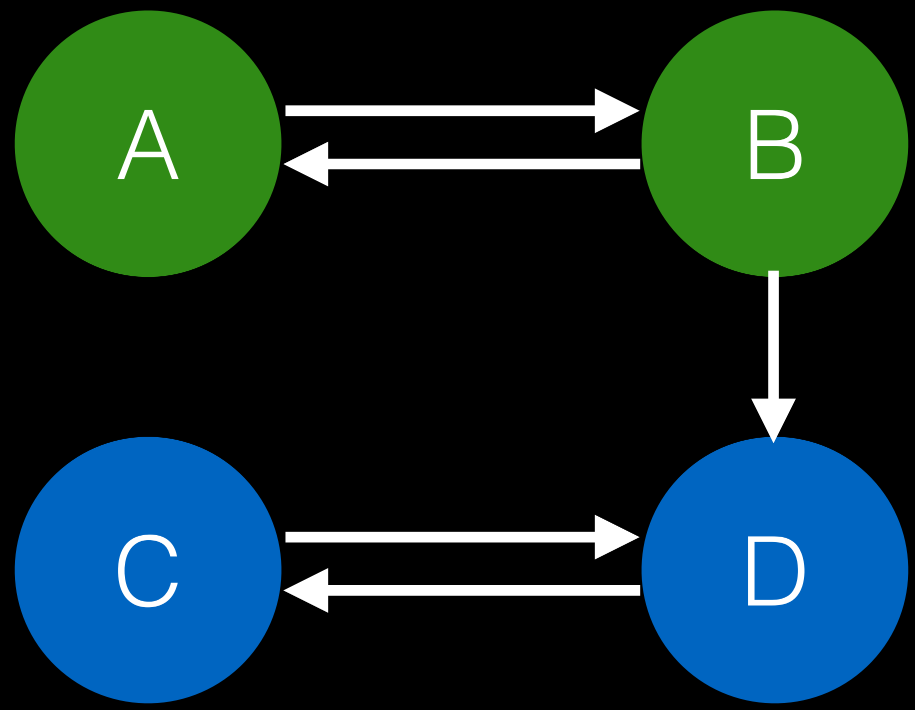
Problem: random surfers without links die.
(and take the rank with them!)

Graph 3



Problem: ???

Graph 3



Problem: Surfers get stuck in C and D.
C+D called a rank “sink”. A and B get 0 rank.

Problems

Problem A: dangling links

Problem B: rank sinks

Solution?

Problems

Problem A: dangling links

Problem B: rank sinks

Solution?

Surfers should jump to new random page with some probability.

Computation

```
ranks = INIT_RANKS; //rank for each page
do {
  new_ranks = compute_ranks(ranks, edges);
  change = compute_diff(new_ranks, ranks);
  ranks = new_ranks;
} while (change > threshold);
```

Computation

```
ranks = INIT_RANKS; //rank for each page
do {
   new_ranks = compute_ranks(ranks, edges);
   change = compute_diff(new_ranks, ranks);
   ranks = new_ranks;
} while (change > threshold);
```

Many MapReduce jobs can be used.

Computation

```
ranks = INIT_RANKS; //rank for each page
do {
   new_ranks = compute_ranks(ranks, edges);
   change = compute_diff(new_ranks, ranks);
   ranks = new_ranks;
} while (change > threshold);
```

Many MapReduce jobs can be used.

Mappers Send Votes From Pages

```
public void map(…) {
   double rank = value.get();
   String linkstring = dataval.toString();
   output.collect(key, RETAINFAC);

   String[] links = linkstring.split(" ");
   double delta = rank * DAMPINGFAC / links.length;

   for(String link : links)
      output.collect(link, delta);
}
```

Adapted from: https://code.google.com/p/i-mapreduce/wiki/PagerankExample

Reducers Sum Votes for Each Page

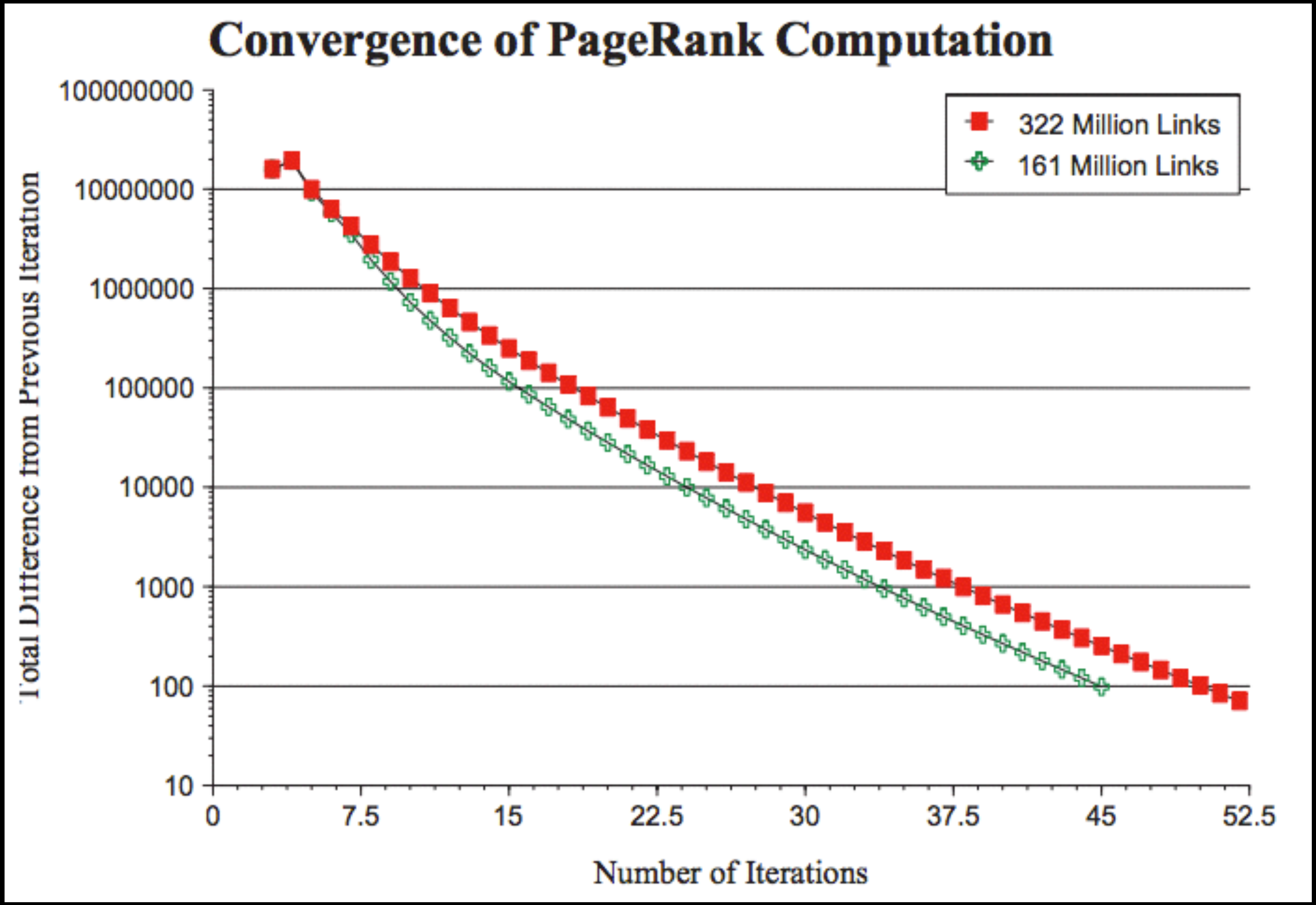
```
public void reduce(…) {
   double rank = 0.0;
   while(values.hasNext())
      rank += values.next().get();
   output.collect(key, new DoubleWritable(rank));
}
```

Adapted from: https://code.google.com/p/i-mapreduce/wiki/PagerankExample

Computation

```
ranks = INIT_RANKS; //rank for each page
do {
   new_ranks = compute_ranks(ranks, edges);
   change = compute_diff(new_ranks, ranks);
   ranks = new_ranks;
} while (change > threshold);
```

What is “change” over time?



The PageRank Citation Ranking: Bringing Order to the Web
(http://ilpubs.stanford.edu:8090/422/1/1999-66.pdf)

Personalized Search

Quality is subjective, and different measures may be best for different people.

Currently, our random surfer occasionally jumps to a random page. PageRank reflects this.

Personalized strategy: bias random jumps towards pages relevant to type of user.

“To test the utility of PageRank for search, we built a web search engine called Google”

Larry Page *etal.*

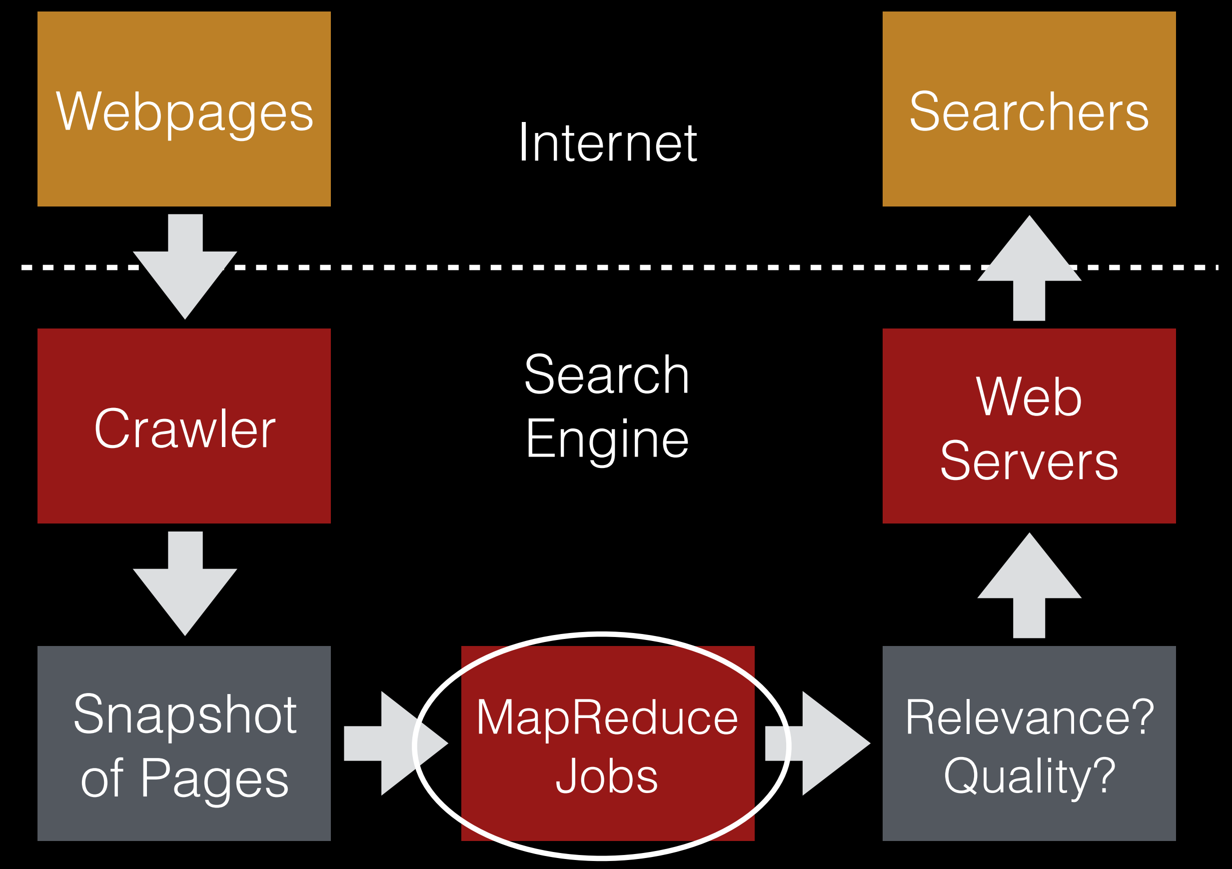
The PageRank Citation Ranking: Bringing Order to the Web
(http://ilpubs.stanford.edu:8090/422/1/1999-66.pdf)

Outline

Web Crawling

Indexing
- PageRank
- Inverted Indexes

Searching



Relevance Problem

A website may be important, but is it relevant to the user's current query?

Infer relevance by page contents, such as:

- html body
- title
- meta tags
- headers
- etc

Indexing

Strategy: indexing.

Generate files organize by topic, keyword, or some other criteria that organize documents.

For a given word, we want to be able to find all related documents.

Representation

For fast processing, assign:
- docID to each unique page
- wordID to each unique word on the web

http://www.example.com/…

Lorem ipsum dolor sit amet, lorem soluta delicata no vim. Te vel facete ornatus, mei aeque maiestatis te.

Representation

For fast processing, assign:
- docID to each unique page
- wordID to each unique word on the web

http://www.example.com/…

Lorem ipsum dolor sit amet, lorem soluta delicata no vim. Te vel facete ornatus, mei aeque maiestatis te.

→

docID=1442

5 922 2 66 42 5 15 79 1431 21 3 22 68 12 47 887 244 3

Forward Index

docID=1442

5 922 2 66 42 5 15 79
1431 21 3 22 68 12 47
887 244 3

docID=9977

522 141 553 999 243
66 42 5 15 79 15 79
1431 21 3 22

…

forward index

docID	wordID
1442	5
1442	922
1442	2
1442	66
1442	42
1442	5
…	…

Inverted Index

forward index

docID	wordID
1442	5
1442	922
1442	2
1442	66
1442	42
1442	5
...	...

Inverted Index

forward index

docID	wordID
1442	5
1442	922
1442	2
1442	66
1442	42
1442	5
...	...

docID	wordID
1442	5
1442	922
1442	2
1442	66
1442	42
1442	5
...	...

Inverted Index

forward index

docID	wordID
1442	5
1442	922
1442	2
1442	66
1442	42
1442	5
...	...

swap columns

wordID	docID
5	1442
922	1442
2	1442
66	1442
42	1442
5	1442
...	...

Inverted Index

forward index

docID	wordID
1442	5
1442	922
1442	2
1442	66
1442	42
1442	5
...	...

sort by wordID

wordID	docID
1	244
2	1442
5	1442
5	1442
5	999
6	133
...	...

Inverted Index

forward index

docID	wordID
1442	5
1442	922
1442	2
1442	66
1442	42
1442	5
…	…

inverted index

wordID	docID
1	244
2	1442
5	1442,1442,999
6	133,411
7	1442,133,999
9	411,875
…	…

Pages without Text

What if pages have no text?

When computing the inverted index for a page, include text of hyperlinks referring to that page.

Extra Metadata

Extra information makes inverted index more useful. E.g., word position, text type, etc.

wordID	docID
1	244
2	1442
5	1442, 1442, 999
…	…

Extra Metadata

Extra information makes inverted index more useful. E.g., word position, text type, etc.

wordID	docID
1	(244,14,h1)
2	(1442,56,h4)
5	(1442,32,b), (1442,10,i), (999,80,h4)
…	…

Computing Inverted Index with MapReduce

Mapper: read words from files
- out key: word
- out val: file name

Reducer: make list of file names
- out key: word
- out val: list of file names

Inverted Index: Mapper

```
public void map(…) {
   FileSplit fileSplit = reporter.getInputSplit();
   String fileName = fileSplit.getPath().getName();

   StringToke itr = new StringToke(val);
   while (itr.hasMoreTokens())
      output.collect(itr.nextToken(), fileName);
}
```

Adapted from: https://developer.yahoo.com/hadoop/tutorial/module4.html#solution

Inverted Index: Reducer

```
public void reduce(…) {
   StringBuilder toReturn = new StringBuilder();
   while (values.hasNext()){
      toReturn.append(values.next().toString() + “ “);

   output.collect(key, toReturn));
}
```

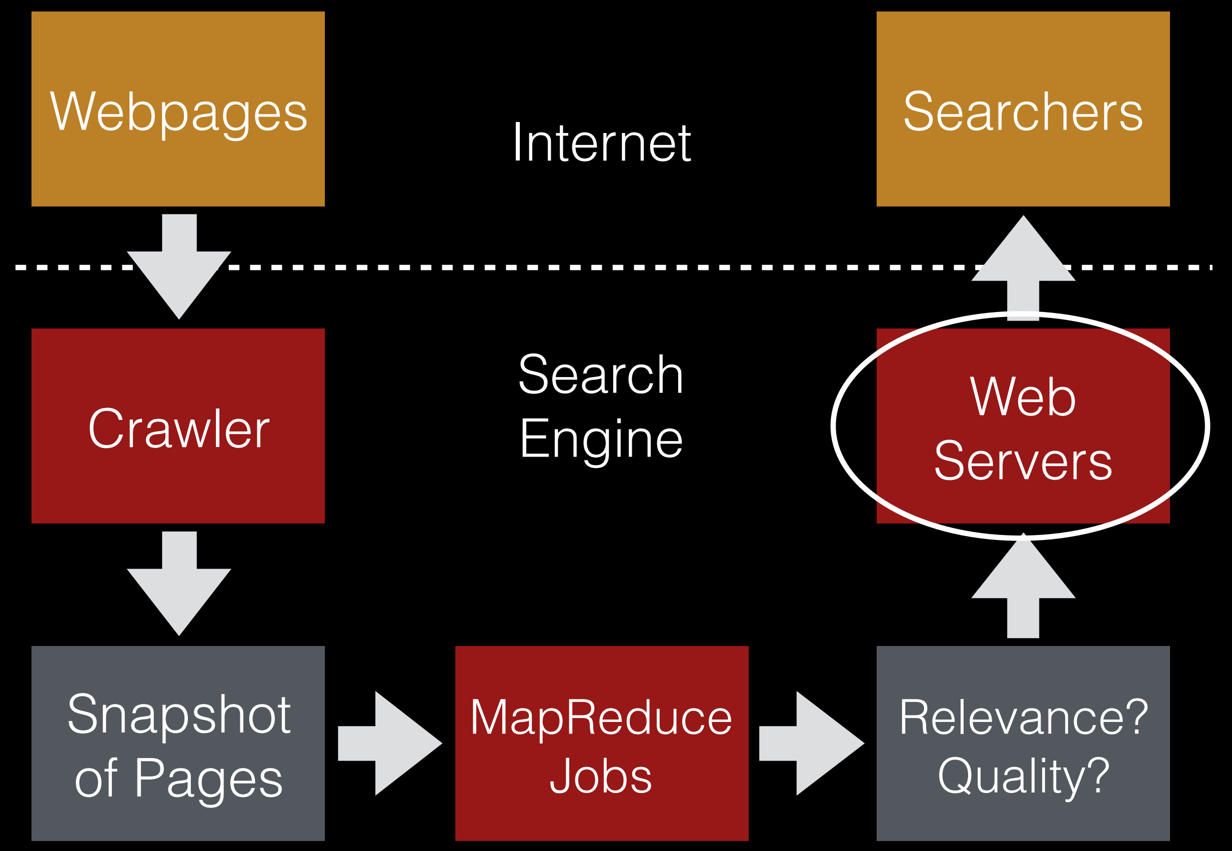
Adapted from: https://developer.yahoo.com/hadoop/tutorial/module4.html#solution

Outline

Web Crawling

Indexing
- PageRank
- Inverted Indexes

Searching



One-word Queries

Inverted index may be split into “posting files” across many machines. wordID => machine is known.

Front-end server takes query, converts to wordID.

Front-end fetches docID’s from server with posting file.

docID’s are sorted based on PageRank and relevance and returned to user.

Multi-Word Queries

Query is converted into list of wordIDs.

docID's from the posting files for each wordID are retrieved.

The lists of docID's can be unioned (OR)
or intersected (AND).

Position metadata is useful: documents with words near each other are preferred.

Phrase Search

Again use position metadata from posting list.

Only look for documents with adjacent query words.

wordID	docID
hello	(244,14,h1), (999,2,h1), (999,103,b)
world	(244,56,h4), (999,104,b)
…	…

Phrase Search

Again use position metadata from posting list.

Only look for documents with adjacent query words.

wordID	docID
hello	(244,14,h1), (999,2,h1), (999,103,b)
world	(244,56,h4), (999,104,b)
…	…

Search for “hello world” return docID 999, but not 244.

Search is Resource Intense

Indexes greatly reduce data that must be considered relative to the grep approach.

However! Most of the data read from the posting lists won't be relevant, so a lot of data must be scanned.

Summary

Crawler: watch for robots.txt

PageRank: simulate random surfer

Inverted Index: list of docs containing a word

Search: take intersection of posting lists

Announcements

Last class. :(

Feedback forms: volunteer?

Office hours after class in lab.

p5a and p5b due Fri. Hard deadline on Dec 17th.

T-Shirts ordered for malloc winners.

Final @ 10:05am next Tue. Review to be planned.