


Plan for Today

- Recitation Qs
- Setting for Map Reduce
- Design
- Failures

Logistics

- * Schedule your proj presentation
- * Design project updates released
- * Volunteers for recitation Qs next week?
- * Reminder: feedback.henrycg.com

1. "Main"

2. Think in base 2:

$1,024 = 2^{10} \approx \text{thousand}$ (K;B)

$2^{20} \approx \text{million}$ (M;B)

$2^{30} \approx \text{billion}$ (G;B)

$2^{40} \approx \text{trillion}$ (T;B)

$2^{50} \approx \text{million billion}$ (P;B)

Recitation Qs

1. What are perf goals of MpReduce?
 - ↳ Easier to process & gen big data.
2. How was M.R. implemented?
 - ↳ Framework — user writes map & reduce fns.
 - ↳ Load balancing
 - ↳ Fault tolerance (cheap HW) — pings
 - ↳ Stragglers
 - ↳ Colocation of data & compute.
3. Why was M.R. implemented this way?
 - ↳ Focus on fault tolerance
 - ↳ Parallelism!

Hugely influential

Hadoop ↗

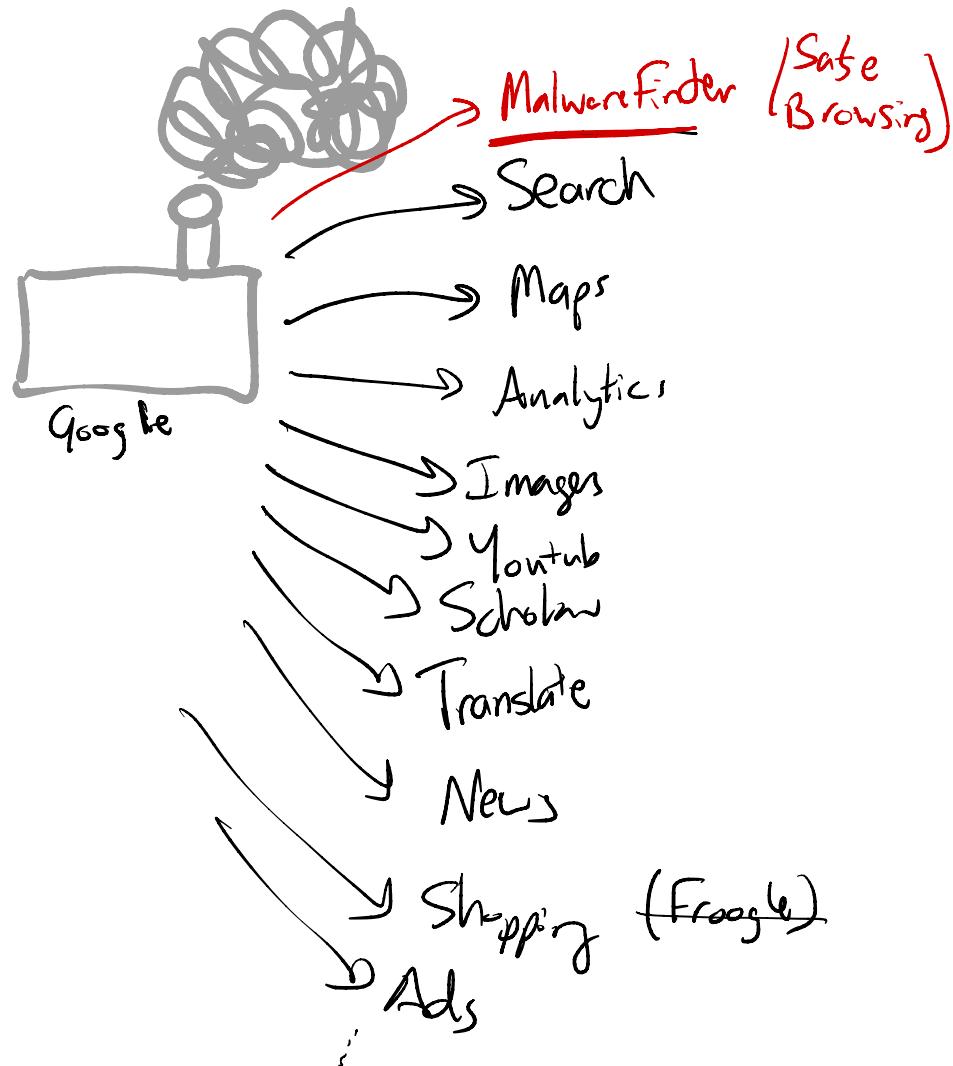
Spark ↗

{ }

The Setting



Downloading web
 $\approx 2^{60}$ bytes



MalwareFinder

- Look at every page
- Run detection alg
- Output list of suspect sites

- Vast data (web)
- Thousands of machine
- Used for many applications

 When you have 2^{Go} bytes of data, simple things become hard.

Idea: Give programmer a simple API.

map(key₁, val₁) → [(key₂, val₂), ...]

reduce(key₂, [val₂, ...]) → [val₂, ...]

Example: PageRank*

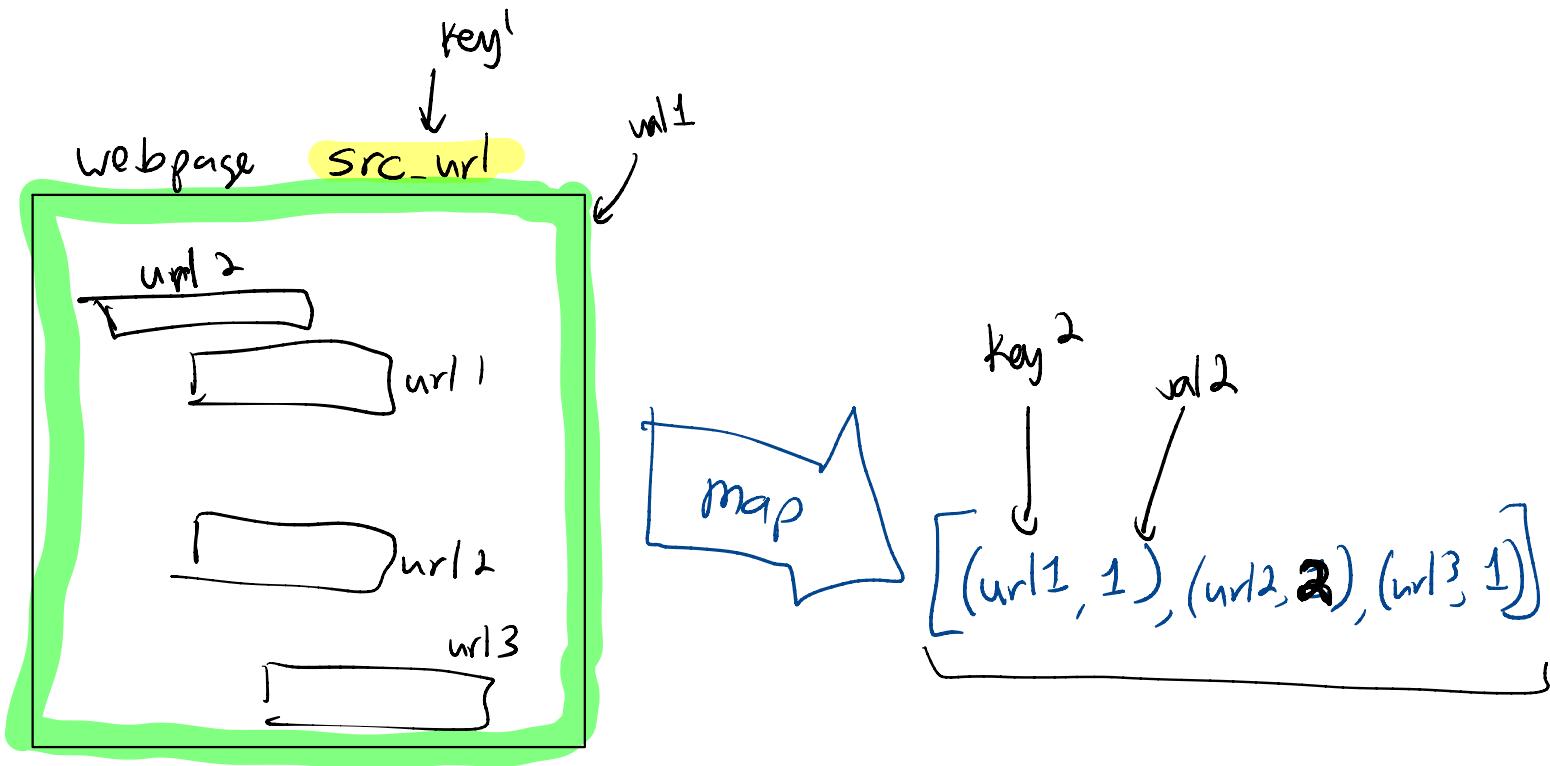
For each URL u how many pages link to u?

map(src-url,

) →

reduce()

Map



key ↓ *value* ↓

```
map(url, html) {  
    // Find all links in html page  
    // Append to list  
    // Output the list.  
}
```

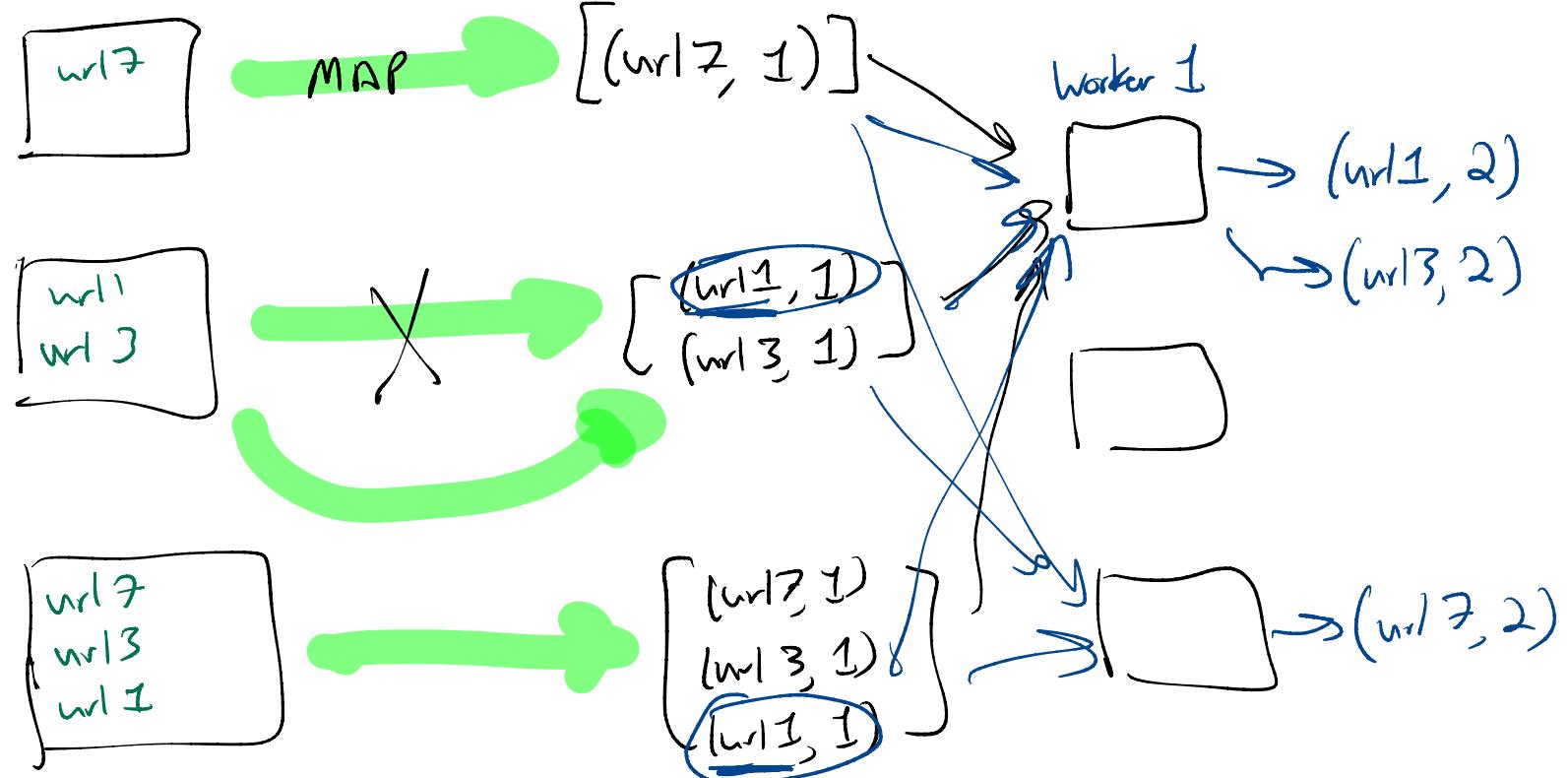
reduce (key2, [val2, ...]) → [val2, ..., -] {
 ↑
 dest URL ↑
 # of times
 was linked to from
 different pages

// Sum up input values.

// Output the sum.

}

Web



1. One main server (as in GFS.)

2.

When not to use...

Plan for Today

- Recitation Qs
- Setting for Map Reduce
- Design
- Failures

Logistics

- * Schedule your proj presentation
- * Design project updates released
- * Volunteers for recitation Qs next week?
- * Reminder: feedback.henrycg.com

Recitation Qs

1.

What are the perf goals of M.R?

↳ big jobs --- run in parallel!

2.

How was MR implemented to meet these goals?

↳ library

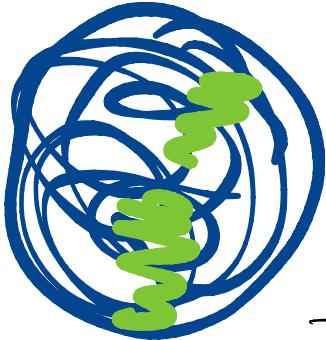
↳ split input, Map input blocks \rightarrow intermediate results, \curvearrowright reduce \rightarrow output

3.

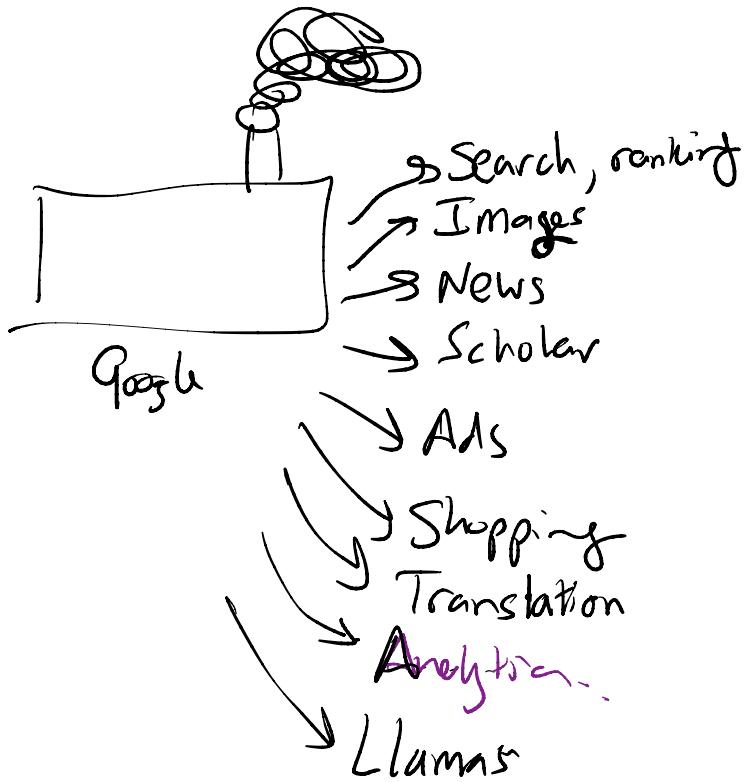
Why was M.R. implemented this way?

↳ Abstract away the detail

↳ User focuses on task at hand
(load balancing, failure recovery)



Download web



- Lots of data $\approx 2^{60}$ bytes
- Thousands of machines
- Many applications

"Hadoop"

Idea: Give programmers a simple interface to a **HUGE** data set.

API:

Data set: $\{(key_1, val_1), \dots\}$

map (key_1, val_1) $\rightarrow [(key_2, val_2), \dots]$

reduce ($key_2, [val_2, \dots]$) $\rightarrow \underline{val_2}$

Example: find URLs of all domains containing "llamas"

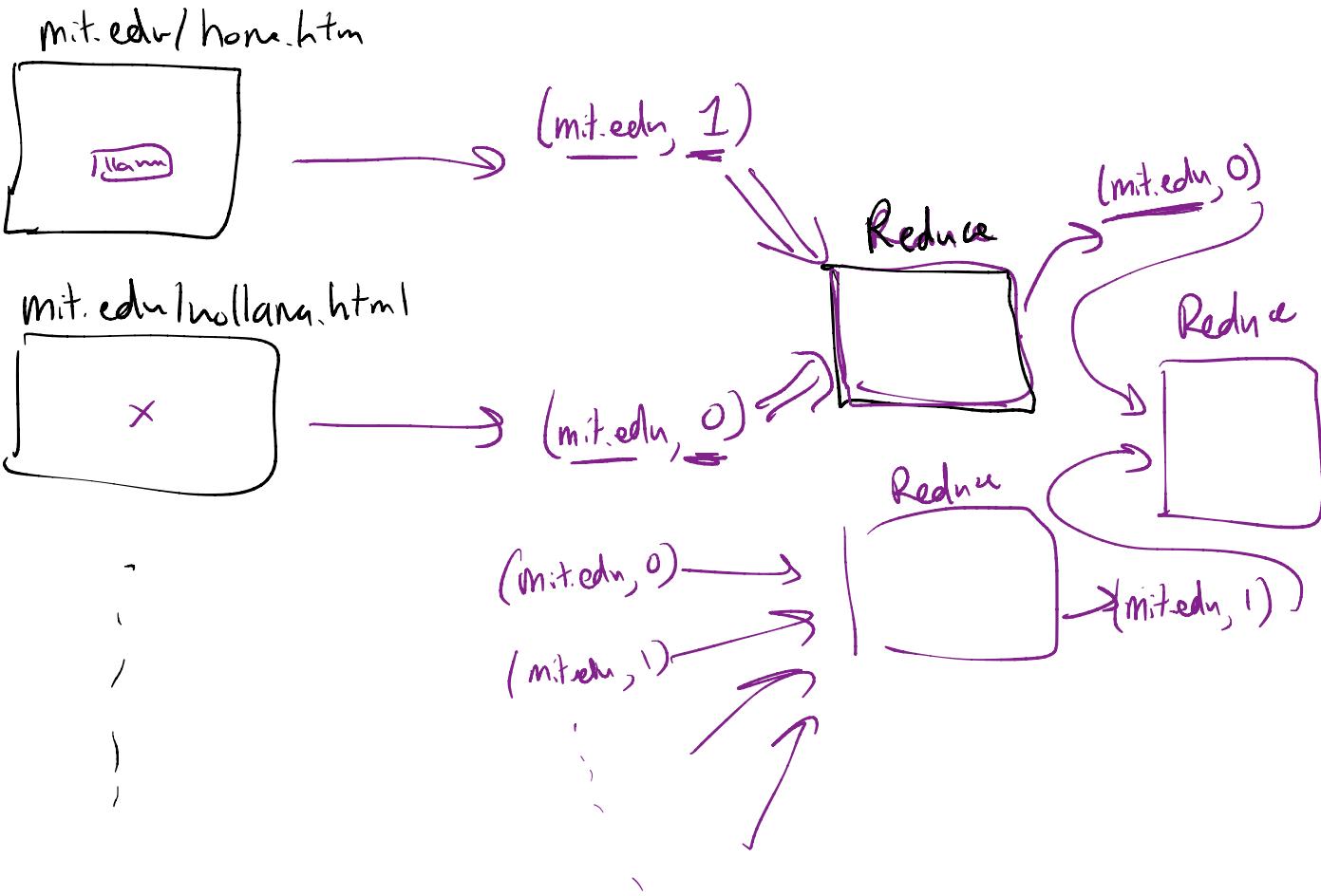
{ mit.edu/llama.htm
mit.edu/ }

Data: $\{(URL, html_content), \dots\}$

map (URL, html) $\rightarrow [(hostname, 1)]$

reduce (hostname, [1]) \leftarrow Almost trivial

Output: $(key_2, [val_2, \dots])$



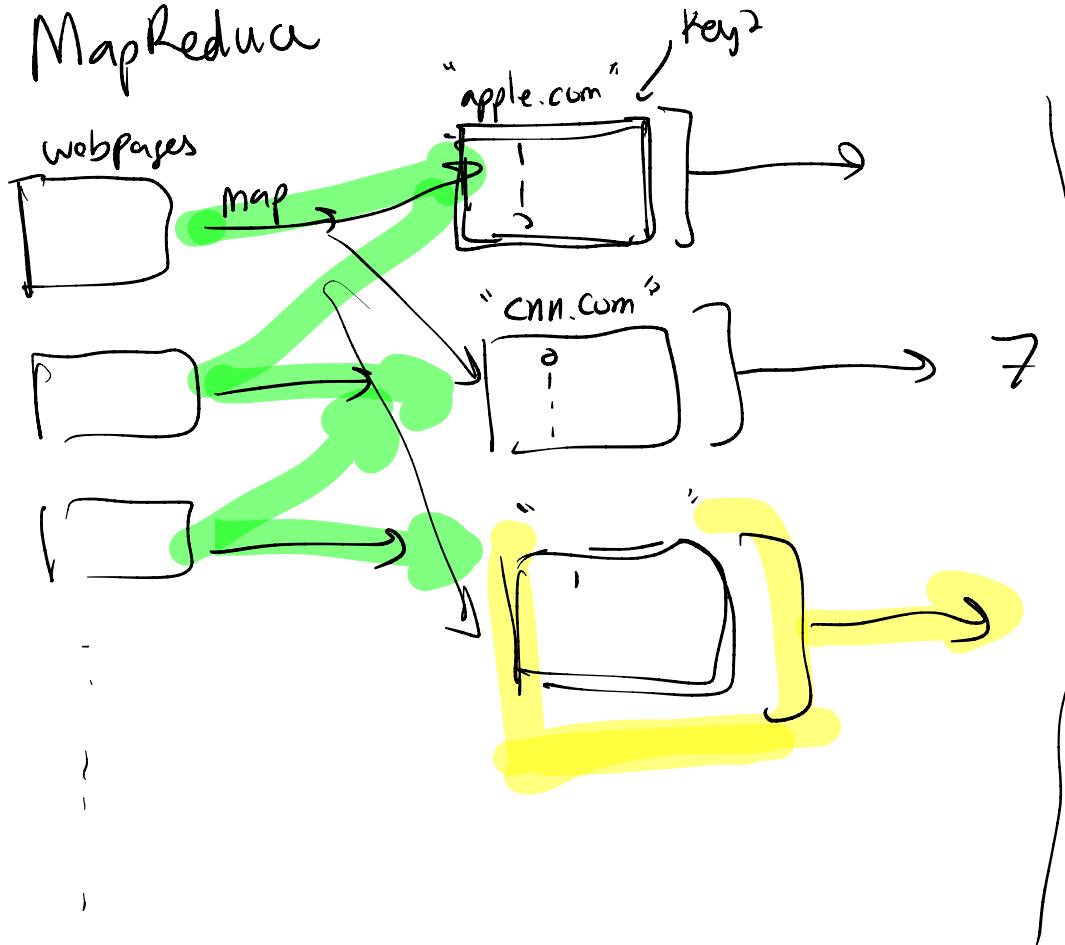
In Python:

$$\left[a, b, c, d, \dots \right] \xrightarrow{\text{map}} \left[f(a), f(b), f(c), \dots \right]$$

Reduce

$$\left[a, b, c, d, \dots \right] \xrightarrow{\text{out} = g(g(g(a,b),c),d)}$$


MapReduce

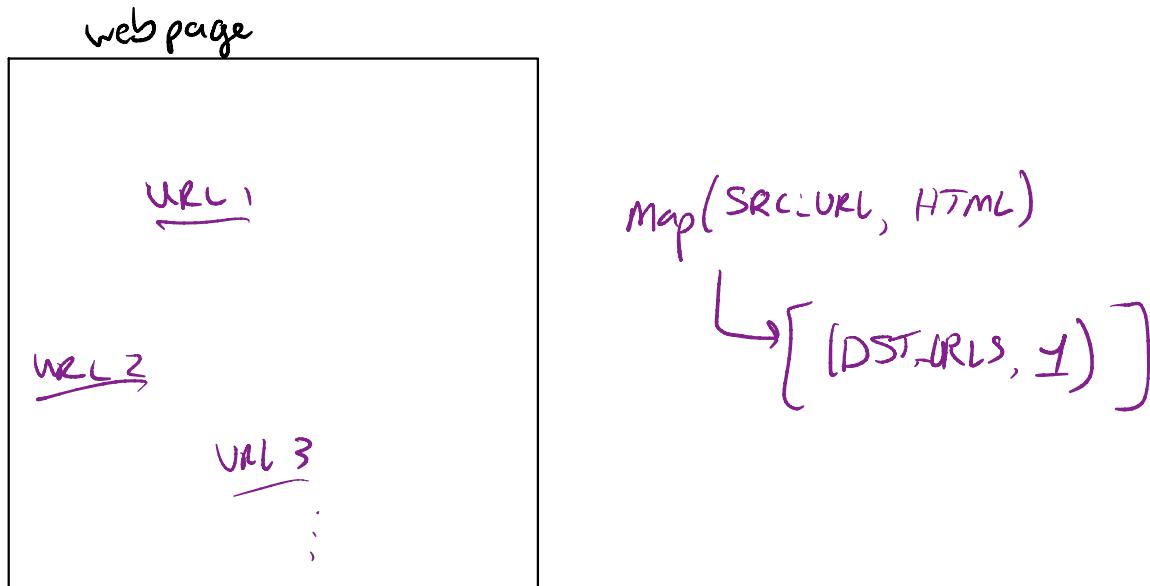


User doesn't have to worry about:

- * who runs the code
- * fault tolerance
- * storage
- * locality
- * stragglers

Example of map fn w/ many outputs?

$$\text{map}(\text{key1}, \text{val1}) \rightarrow [(\text{key2}, \text{val2}), \dots]$$



Po11: MapReduce it?

Plan: MapReduce

- * The setting
- * System design
- * MapReduce it?
- * Failures

Logistics

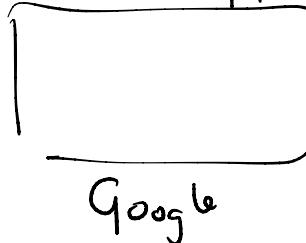
- * Schedule your project presentation!
- * Design project updates released

The Setting



The Web

Download pages in crawl



Images

Translate

Search index & rank

News

Training ML models

User data mgmt
analytics

:

:

Shopping

Find all the
pages hosting malware

- Lots of data $\approx 2^{GS}$ bytes
- 10,000s machines
- Many applications

Idea: Give programmer a simple way to interact with very large data set.
[Force app developer to expose parallelism.]

API: Data set: $\{(key, value), \dots\}$
URL ↑
 HTML page

$\text{map}(key1, val1) \rightarrow [(key2, val2), \dots, \dots]$

$\text{reduce}(key2, [val2, \dots, \dots]) \rightarrow val2$

→ User implements map, reduce.

→ System handles: failures, scheduling, data placement, ...
machines, load balancing,

Example: Page popularity

Q: For each URL u , how many pages link to URL u ?

↳ search

Data : $\{(\text{page-name}, \text{page-contents}), \dots\}$

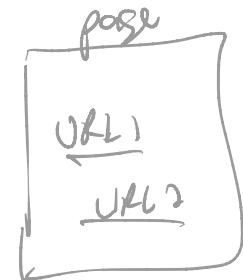
$\text{map}(\text{page-name}, \text{page-contents}) \rightarrow [(\underline{\text{url}}_1, 1), (\underline{\text{url}}_2, 1), \dots]$

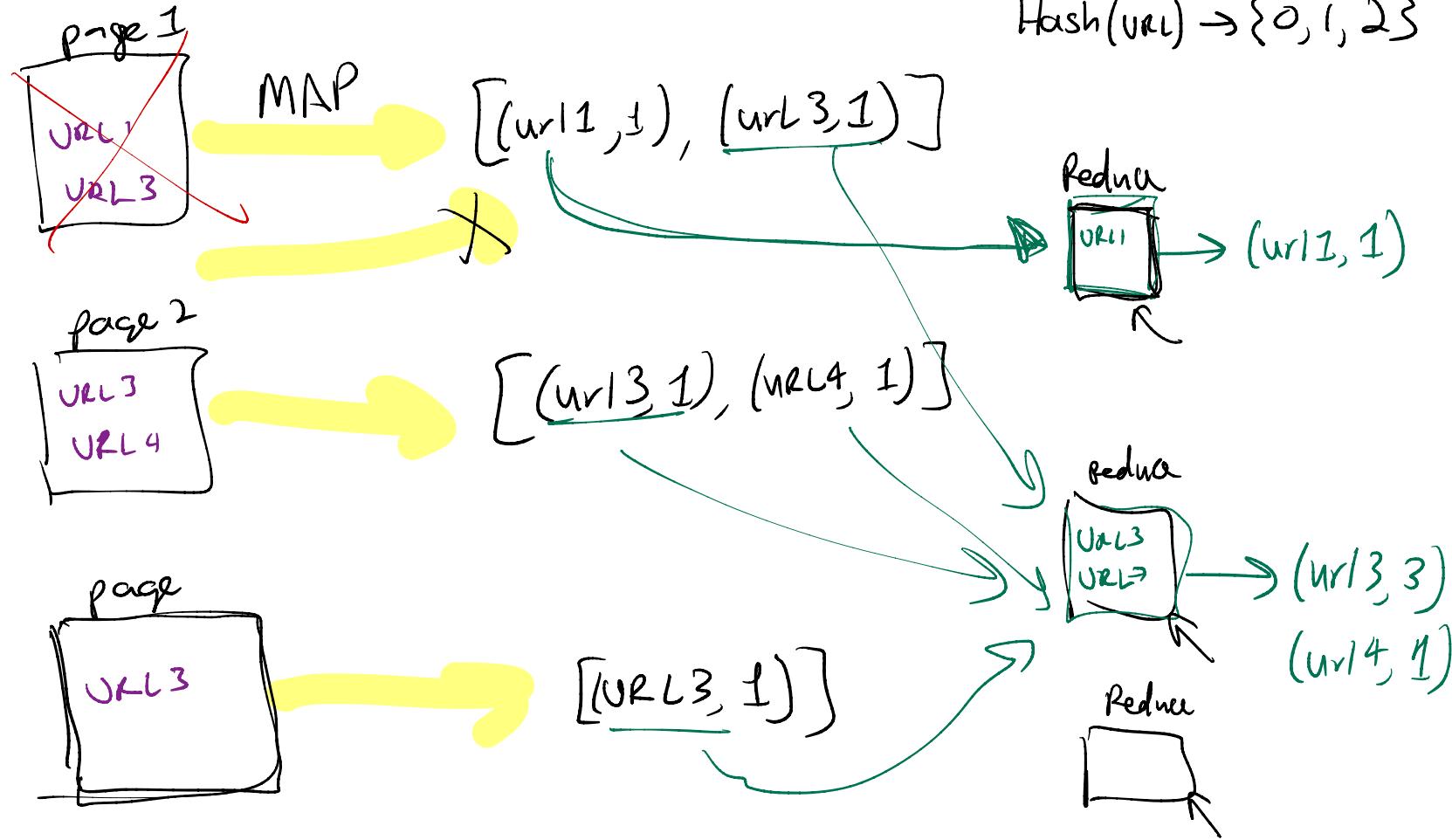
// Find all outgoing links on page

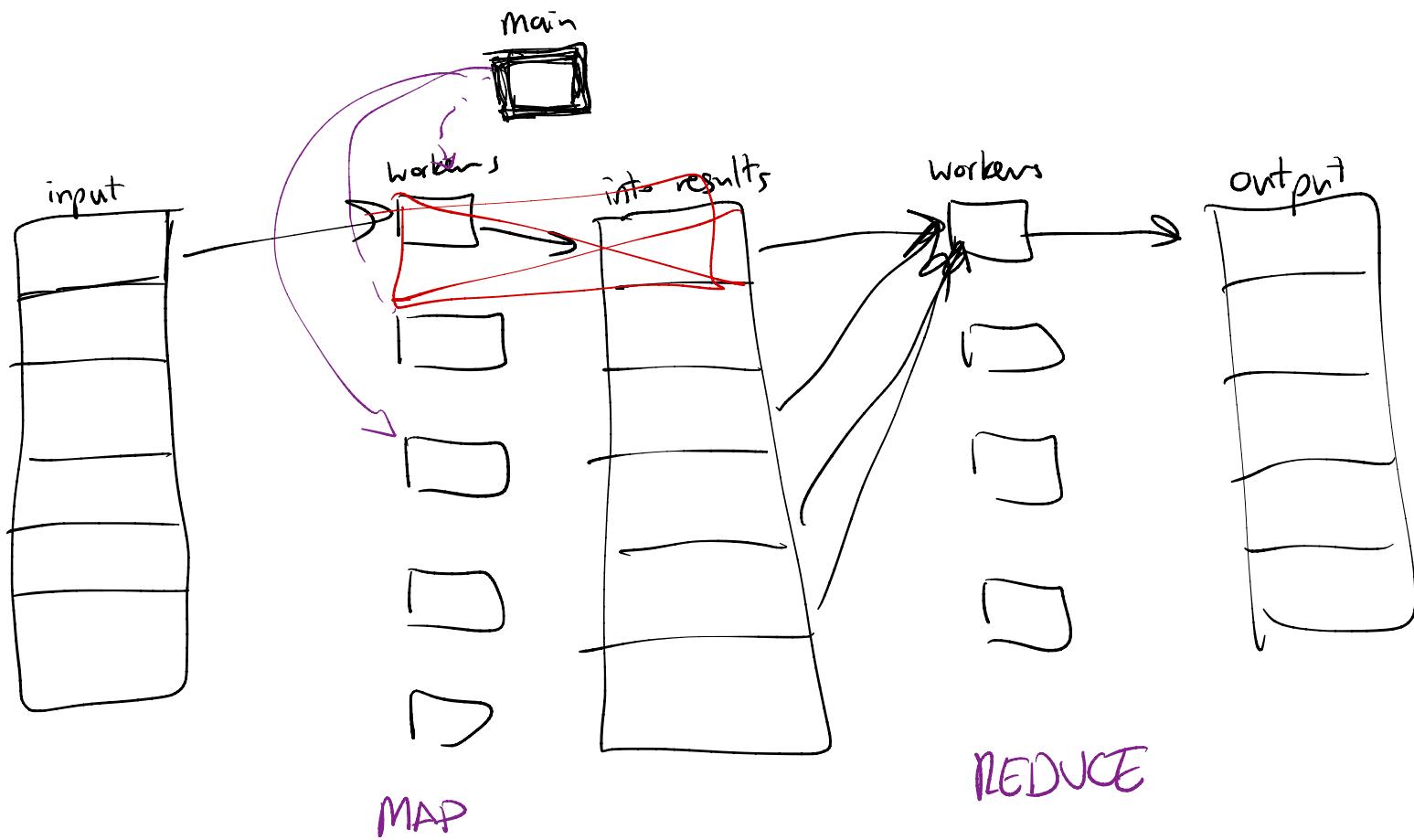
// write their URLs to output

$\text{reduce}(\underline{\text{URL}}, [1, 1, 1, 1, 1]) \rightarrow 5$

// Count the # of 1s in input







Say have n machines in data center.

Each fails w.p. $\frac{1}{1000}$

$$\Pr[\text{main fail}] = \frac{1}{1000}$$

independent

$$\Pr[\text{worker fails}] = 1 - \left(1 - \frac{1}{1000}\right)^n$$

$$n \leq 1000$$

$$\frac{1}{2}$$