

The team... at 2am in the morning

Jamie Song - js4390@columbia.edu

Olesya Medvedeva - oam2113@columbia.edu

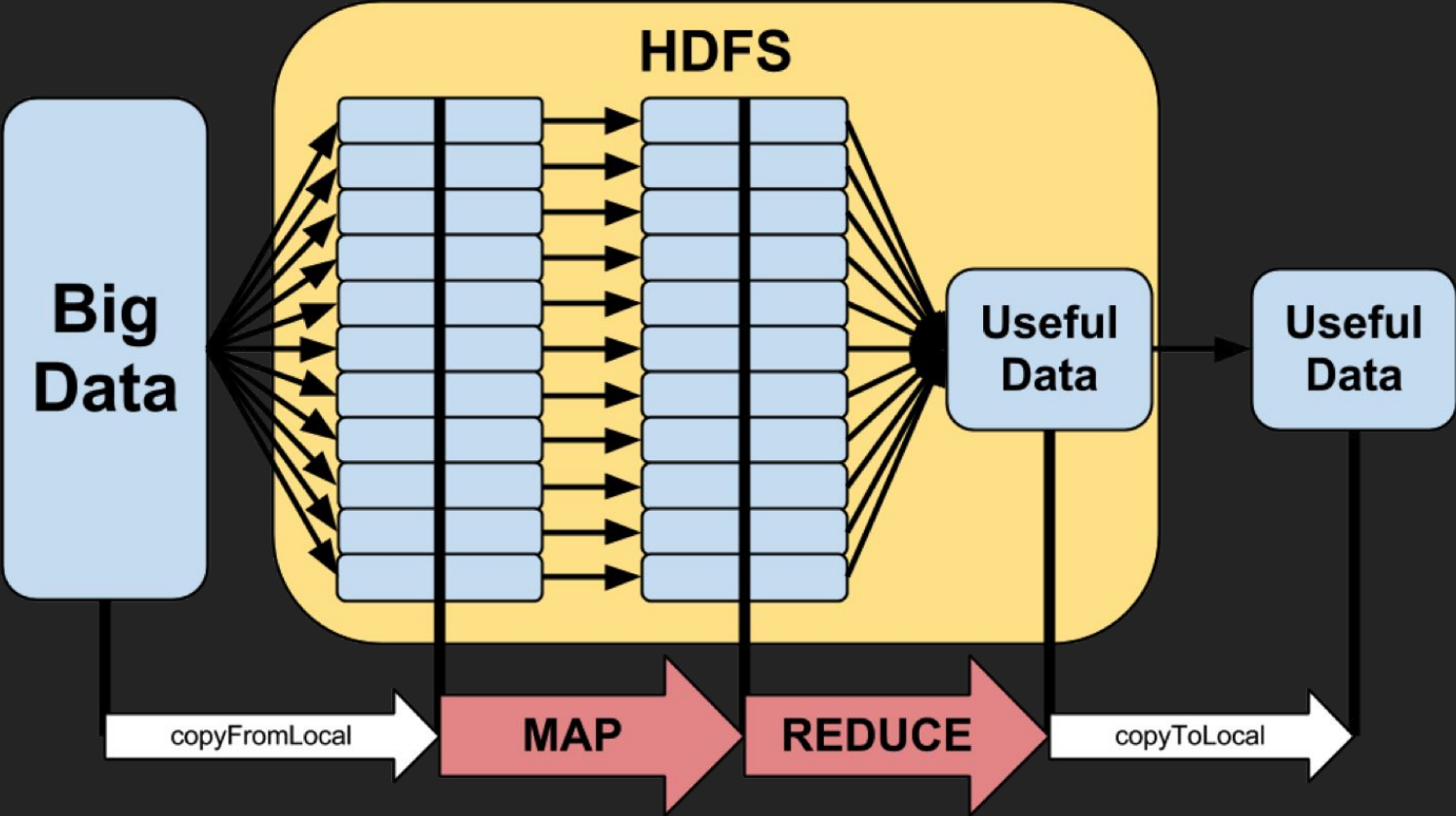
Ryan DeCosmo - rd2680@columbia.edu

Charis Lam - cl3257@columbia.edu

Concept: MapReduce

1. Large input data set. (ex. a book)
2. Data set gets split into chunks. (ex. small text files)
3. A function is applied to each chunk
(ex. return the frequency of the word 'hitchhiker')
3. Aggregate all the results into one unit. (ex. 42)

Inspiration: Apache Hadoop



Expectations:



-> BIIIIIG DATA

-> Multi-threaded on graphics card

-> GPU-accelerated,

-> In-memory

-> Map-reduce replacement for single workstation users

reality...

miniMap:

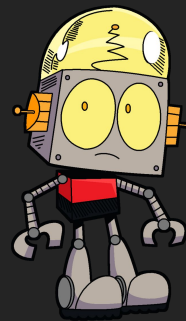
Text processing language <-

Small-to-Medium Data <-

Sorta.. multi-threaded! <-

Lower overhead than the hadoop ecosystem <-

*Ideal? For projects / researchers



so how should it work?

miniMap()

works like MapReduce

miniMap(`File* inputFile`, `void* splitter()`, `void* mapper()`, `File* context`, `void* reducer()`)

the pieces:

- File* inputFile: an input text file
- void* splitter(): function pointer to a function that splits the input file
- mapper(): function pointer to a user defined function
- File* context: an intermediate step that outsources RAM to disk
- reducer(): function pointer to a user defined function

Function headers

File** split_by_size(int x)

File** split_by_quant(int x)

File** split_by_regex(File*, String)

void mapper(File*, File*)

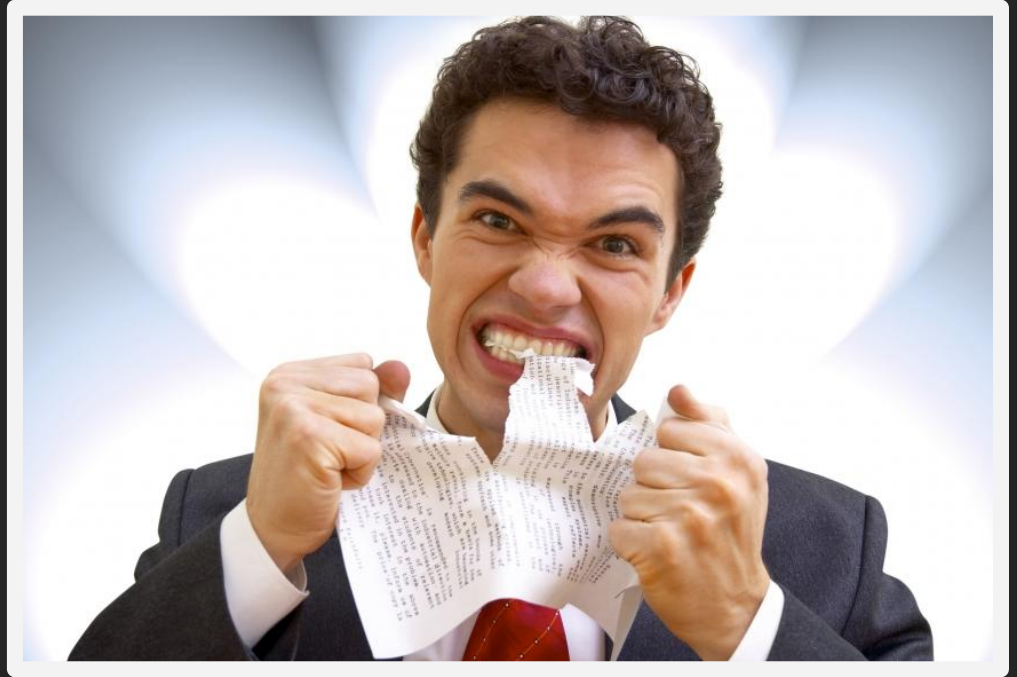
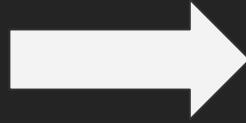
void reducer(File*)

void miniMap(input, splitter, mapper, context, reducer**)**

so how does it work?



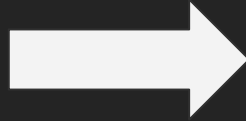
Input File



Splitter Function

so how does it work?

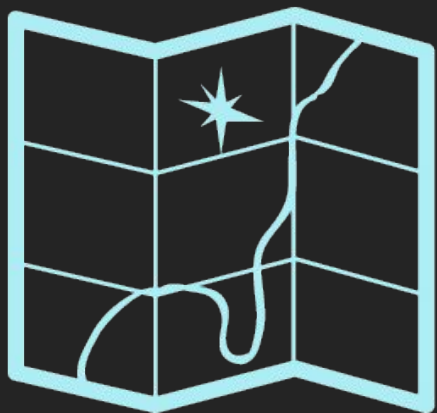
Splitter Function



Disk

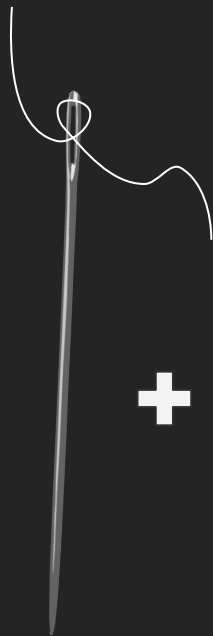


so how does it work?



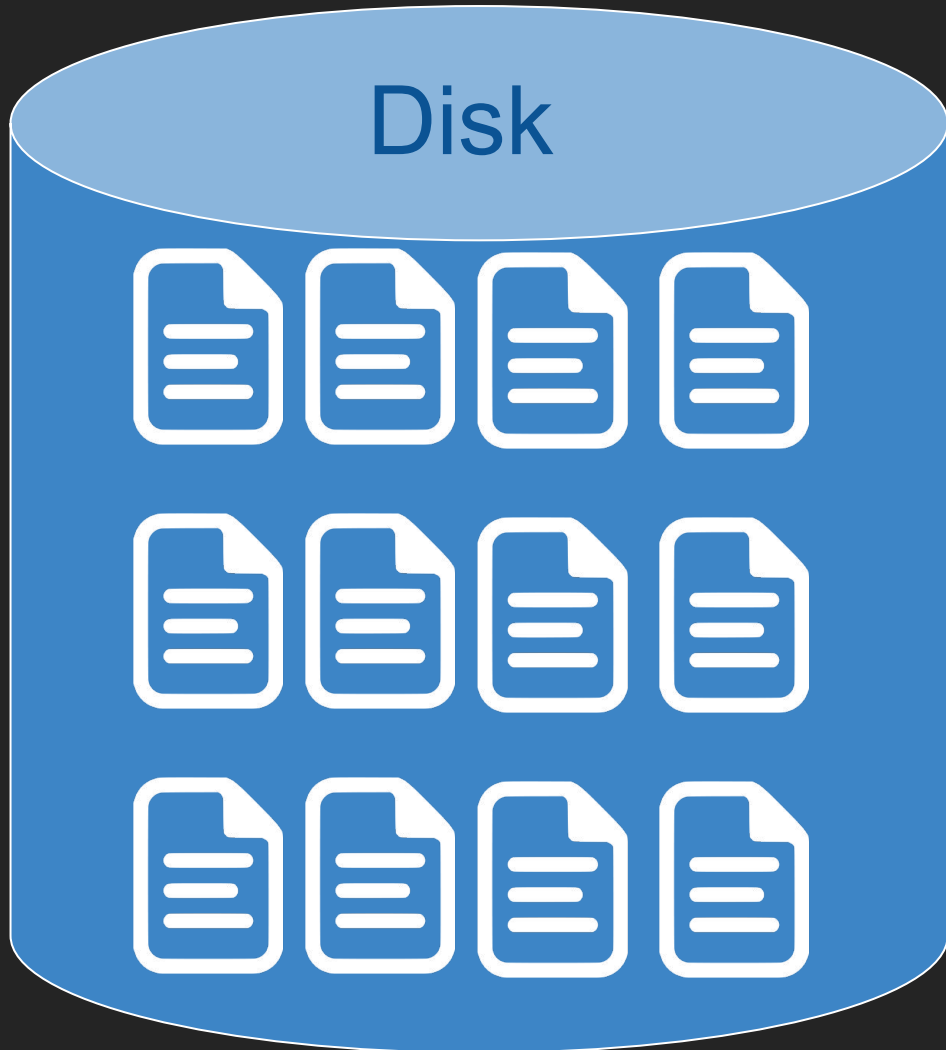
MiniMap

+



+

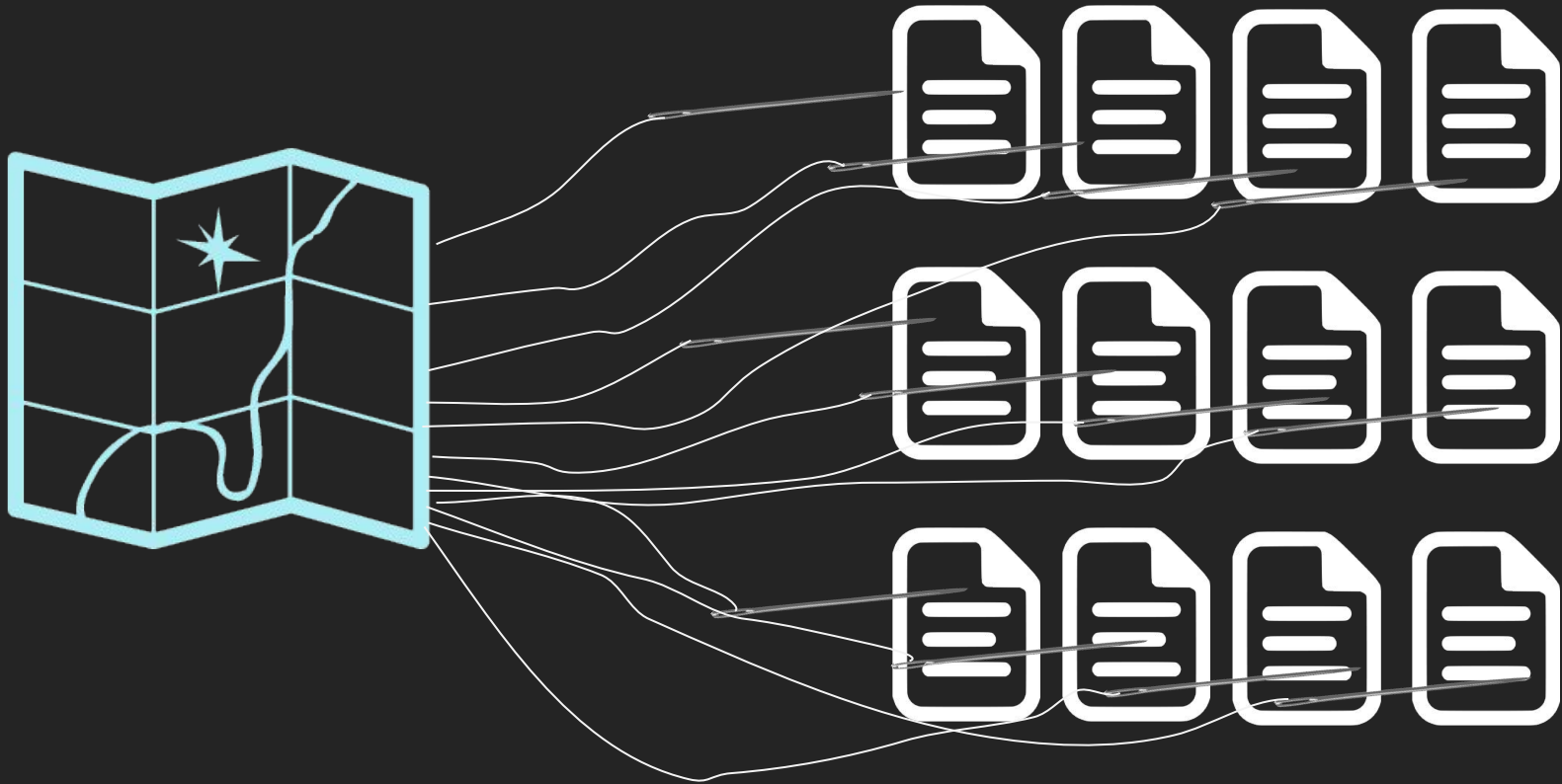
Threads



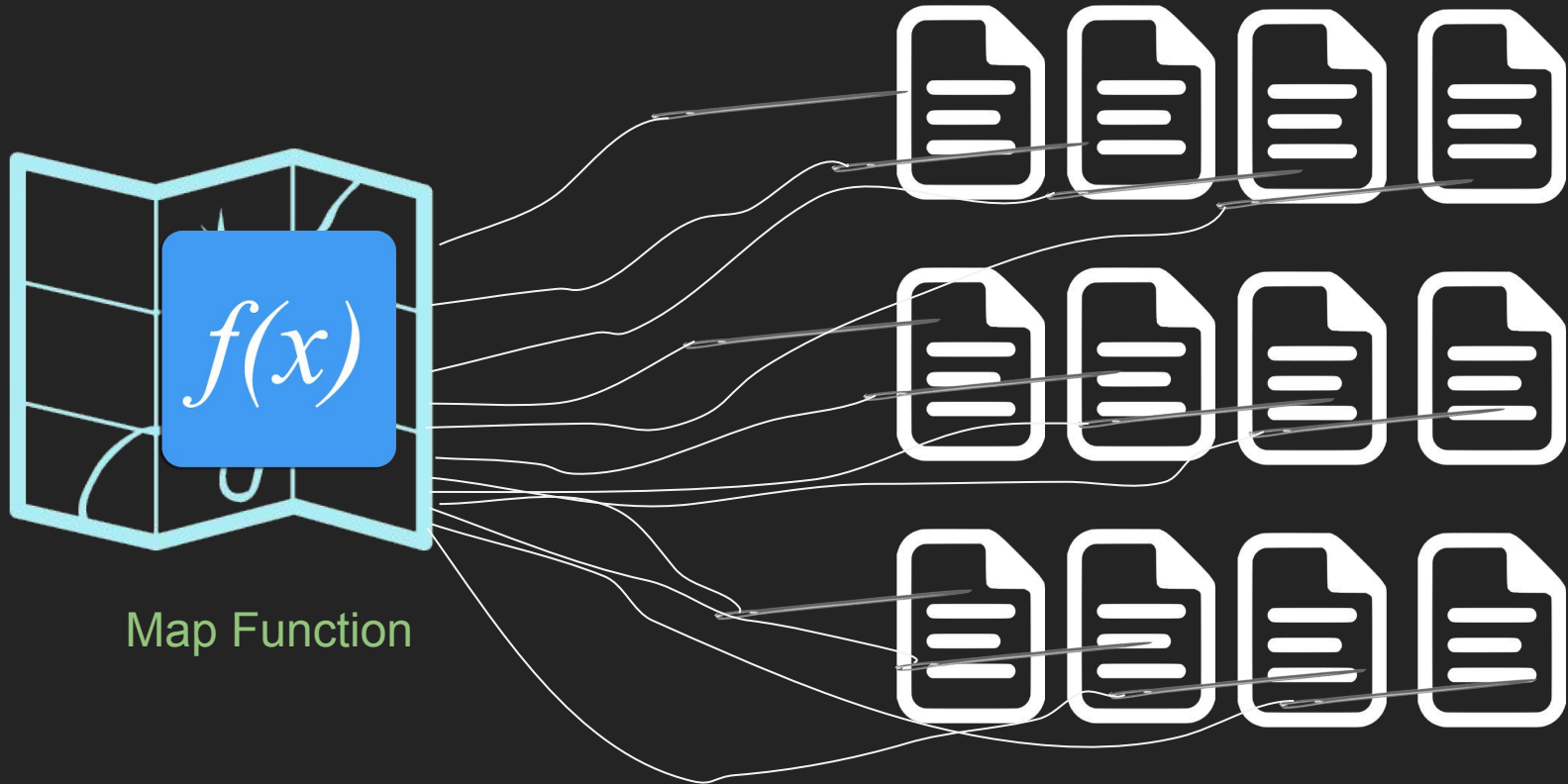
Disk

so how does it work?

Multiple threads

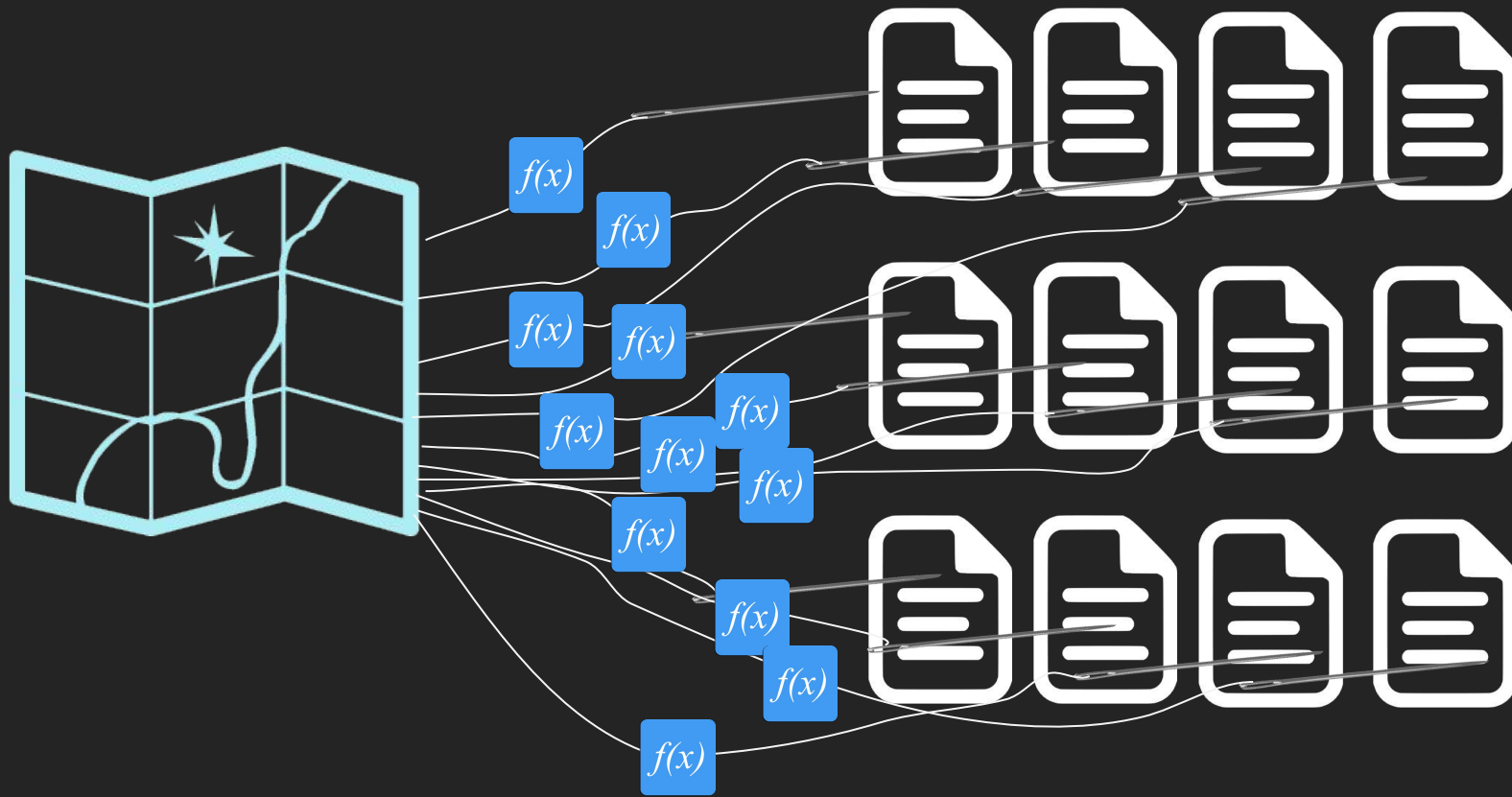


so how does it work?



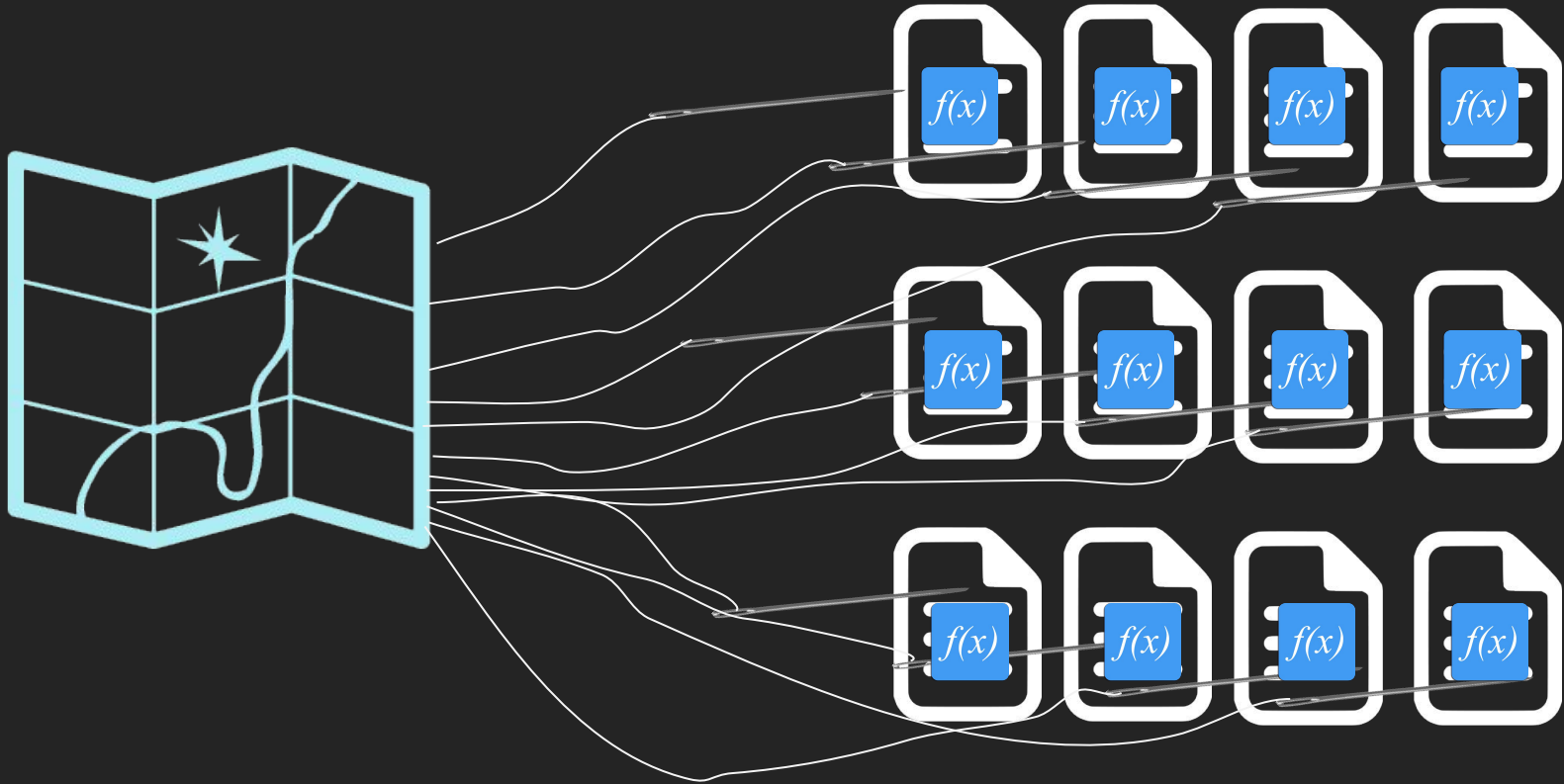
Architecture

Applied using threads



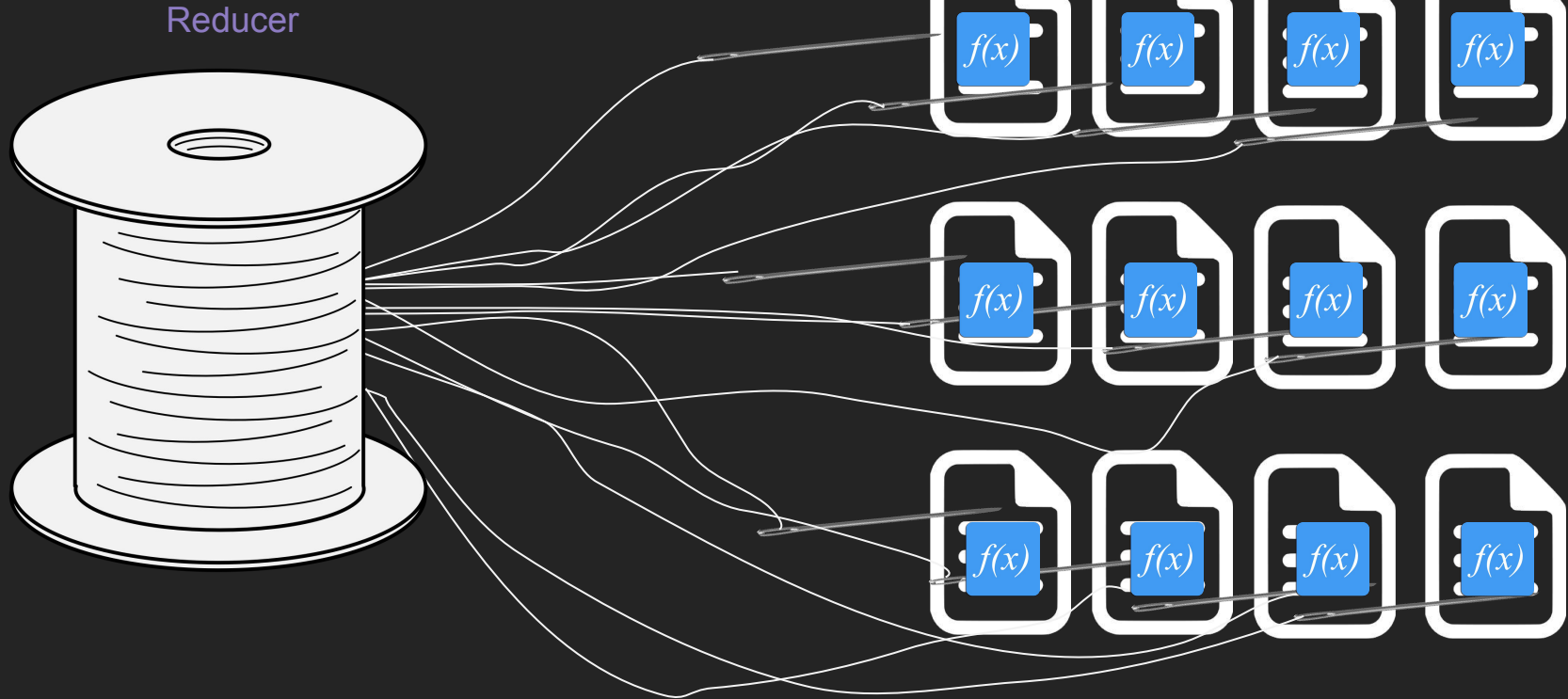
so how does it work?

Each file chunk has the
map function applied to it



so how does it work?

Reducer combines data from mapper threads



Result:

A hand wearing a yellow nitrile glove holds a white marker, pointing to a screen displaying JavaScript code. The code is a complex script for a web application, featuring various functions for data manipulation, user interaction, and DOM manipulation. The code is written in a dark-themed editor with light-colored text. The hand is positioned on the left side of the frame, with the marker pointing towards the center of the code block.

```
var a = $(use).val(); if (0 == a.length) { return; } for (var a = replacements, b = [...], c = 0; c < a.length; c++) { 0 == use_array(a[c], b) && b.push(a[c]); } return b; } function liczenie() { for (var a = $("#user_logged").val(), a = replaceAll(" ", "", a), a = a.replace(/+(?= )/g, ""), a = a.split(" "), b = [], c = 0; c < a.length; c++) { 0 == use_array(a[c], b) && push(a[c]); } c = {}; c.words = a.length; c.unique = b.length - 1; return c; } function use_unique(a) { for (var b = [], c = 0; c < a.length; c++) { 0 == use_array(a[c], b) && b.push(a[c]); } return b.length; } function count_array_gan() { var a = 0, b = $("#user_logged").val(), b = b.replace(/(\r\n|\n|\r)/gm, ""); b = replaceAll(" ", "", b), b = replace(/+(?= )/g, ""); inp_array = b.split(" "); input_sum = inp_array.length; for (var b = [], a = [], c = 0; a < inp_array.length; a++) { 0 == use_array(inp_array[a], c) && (c.push(inp_array[a]), b.push({word: inp_array[a], use_class: 0})), b[b.length - 1].use_class = use_array(b[b.length - 1].word, inp_array); } a = b; output_words = a.length; a.sort(dynamicSort("use_class")); a.reverse(); b = a.map(function(a, b) { return a; }); b = indexOf_keyword(a, ""); -1 < b && a.splice(b, 1); b = indexOf_keyword(a, ""); -1 < b && a.splice(b, 1); return a; } function replaceAll(a, b, c) { return a.replace(new RegExp(b, "g"), c); } function use_array(a, b) { for (var c = 0, d = 0; d < b.length; d++) { b[d] = a.indexOf_keyword(a, b) { for (var c = -1, d = 0; d < a.length; d++) { if (a[d] == b[c]) { c++; } } return c; } function dynamicSort(a) { var b = 1; "-" === a ? b = -1 : b = 1; return function(c, d) { return(c[a] < d[a] ? -1 : c[a] > d[a] ? 1 : 0) * b; } } function occurrences(a, b, c) { a += ""; b += ""; if (0 >= b.length) { return a.length + 1; } var f = 0, f = 0; for (c = c ? 1 : b.length; c <= a.length; c++) { if (f = a.indexOf(b, f), 0 <= f) { d++, f += c; } else { break; } } return d; } $("#go-button").click(function() { var a = parseInt($("#limit_val").val()), a = Math.min(a, 200), a = Math.min(a, parseInt(h().unique)); limit_val = parseInt($("#limit_val").val()); limit_val = a; $("#limit_val").val(a); update_slider(); function(limit_val) { $("#word-list-out").html(""); var b = k(); h(); var c = l(), a = "", d = parseInt($("#limit_val").val()), f = parseInt($("#slider_shuffle_number").val()); function("LIMIT_total:" + d); function("rand:" + f); d < f && (f = d, function("check_rand:\u00f3\u00f3rand:" + f + "tops:" + d)); var n = [], d = d - f, e; if (0 < c.length) {
```

File of clean, useful Data

Built-in Types

- ints

- bool

- float

- String

- void

- File

- Array

- Array pointer

```
A.Int -> i32_t
A.Bool -> il_t
A.Float -> float_t
A.String -> str_t
A.Void -> void_t
A.File -> void_ptr
A.ArrayType(typ, size) -> (match typ with
| A.Int -> array_t i32_t size
| A.Float -> array_t float_t size
| A.Bool -> array_t il_t size
| A.File -> array_t void_ptr size
| _ -> raise ( UnsupportedArrayType )

A.ArrayPointer(t) -> (match t with
| A.Int -> pointer_t i32_t
| A.Float -> pointer_t float_t
| _ -> raise (IllegalPointerType))
```

Built-in functions.. links to C standard library!

Prints:

`print()`, `printb()`, `printbig()`, **`printstring()`**

Splitters:

`split_by_size()`, **`split_by_quant()`**, **`split_by_regex()`**

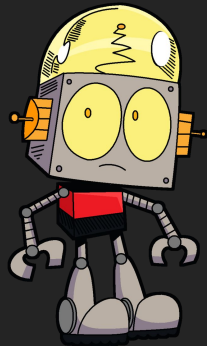
File:

`open()`, `readFile()`, `isFileEnd()`, `close()`

String:

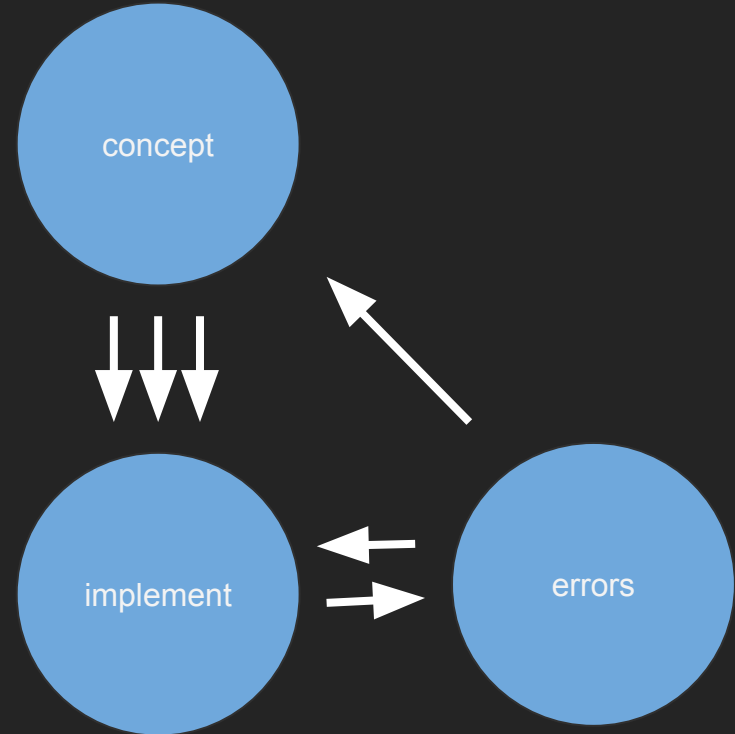
`strstr()`

demo!



Our process:

- Weekly meetings
- Internal implementation goals
- Iterative cycle of concept and coding!



possible directions that Minimap could take:

GPU acceleration using Nvidia CUDA

Multi-Node Support (multiple multi-core PCs)

Optimize File I/O - Sequential Offset (like Kafka)

THANKS FOR LISTENING



ANY QUESTION