It's a Tree... It's a Graph... It's a Traph!!!!

Designing an on-file multi-level graph index for the Hyphe web crawler

Paul Girard • Mathieu Jacomy • Benjamin Ooghe-Tabanou • Guillaume Plique

SciencesPo
MÉDIALAB

Equipex DIME-SHS ANR-10-EQPX-19-01

https://medialab.github.io/hyphe-traph/fosdem2018

-

http://bit.ly/fosdem-traph

Hyphe?

- A web corpus curation tool
- A research-driven web crawler
- Demo: http://hyphe.medialab.sciences-po.fr/demo/
- v1.0 finally easily installable via Docker

URL HTTP://EN.WIKIPEDIA.ORG/WIKI/CAT

↓ Tokenizing

Tokenized URL HTTP:// EN. WIKIPEDIA .ORG /WIKI /CAT

↓ Reordering

LRU .ORG | WIKIPEDIA | EN. | /WIKI | /CAT

...a real LRU actually looks like this:

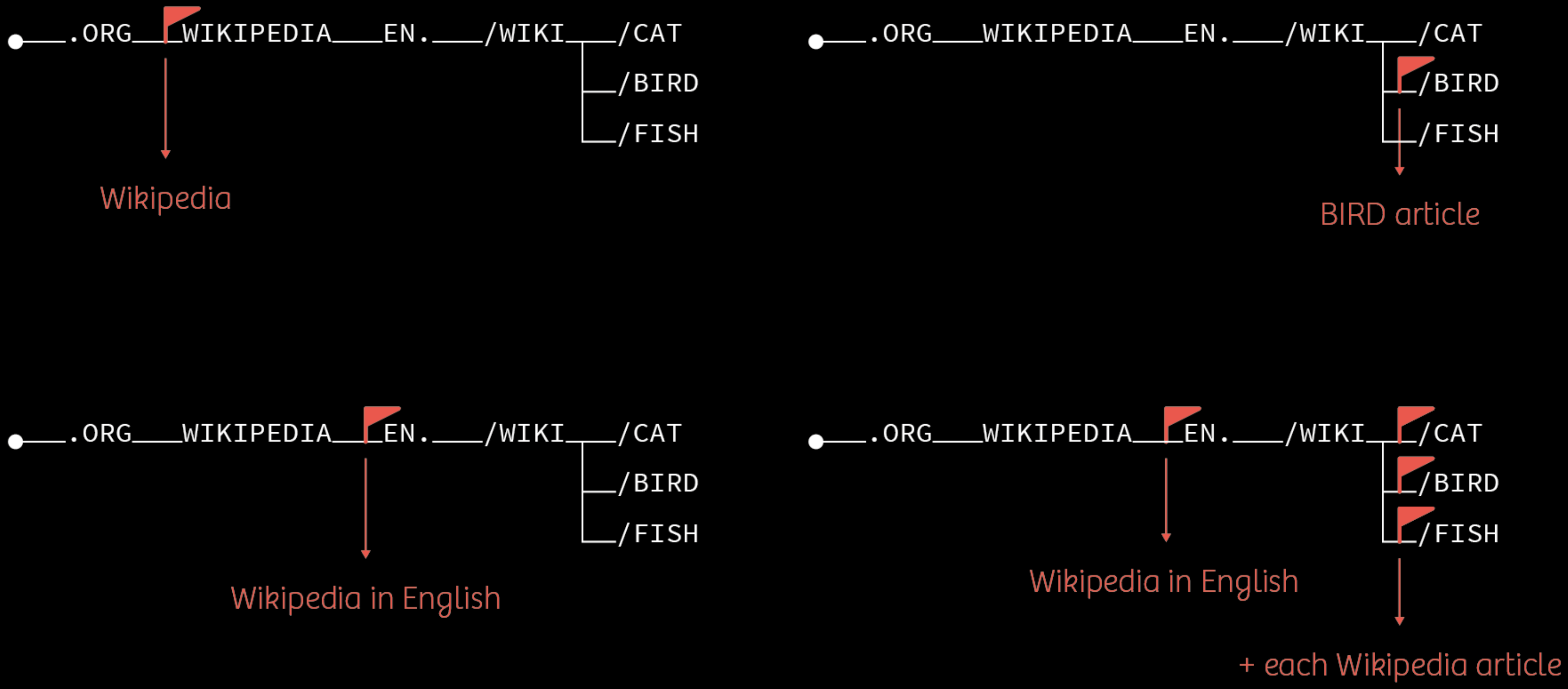
`s:https|h:org|h:wikipedia|h:en|p:wiki|p:Cat|`

.ORG	WIKIPEDIA	EN.	/WIKI	/CAT		
.ORG	WIKIPEDIA	EN.	/WIKI	/BIRD		
.ORG	WIKIPEDIA	EN.	/WIKI	/FISH		
.ORG	WIKIPEDIA	EN.	/W	/INDEX.PHP	?TITLE=CAT&ACTION=HISTORY	

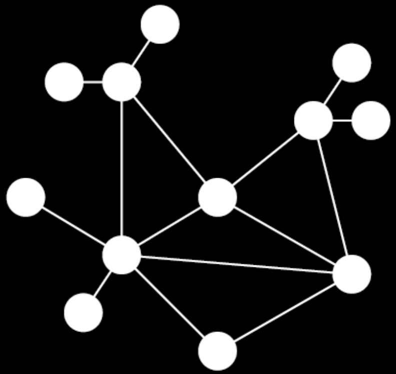
LRUs have a tree structure

```
●—.ORG__WIKIPEDIA__EN.__┬_/WIKI__┬_/CAT
                        │        ├_/BIRD
                        │        └_/FISH
                        │
                        └_/W__/INDEX.PHP__?TITLE=CAT&ACTION=HISTORY
```

Web entities are represented as flags in the LRU tree

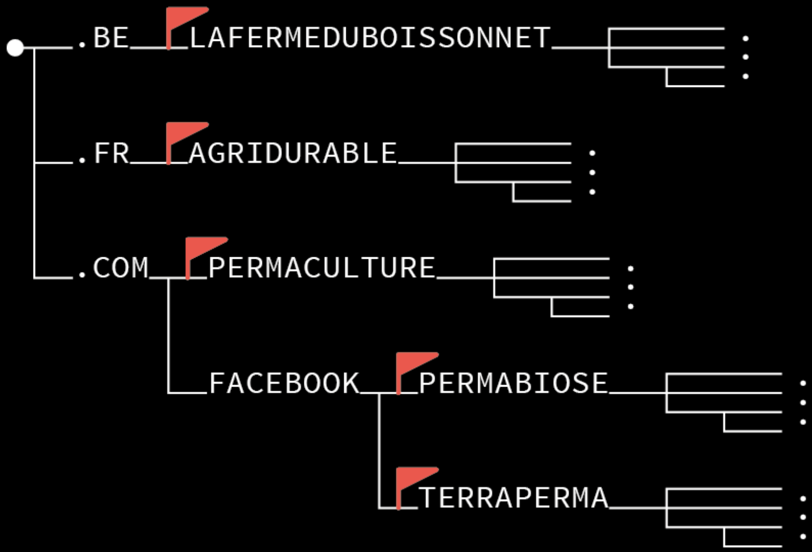


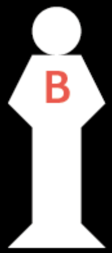
Audrey studies the permaculture community



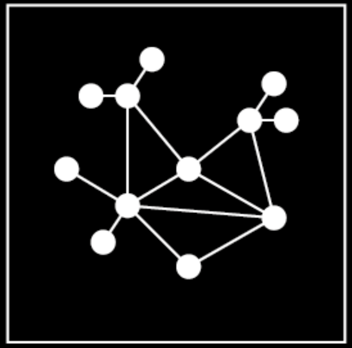
Web entities = ACTORS

Web entities are websites (domains) or profiles on platforms





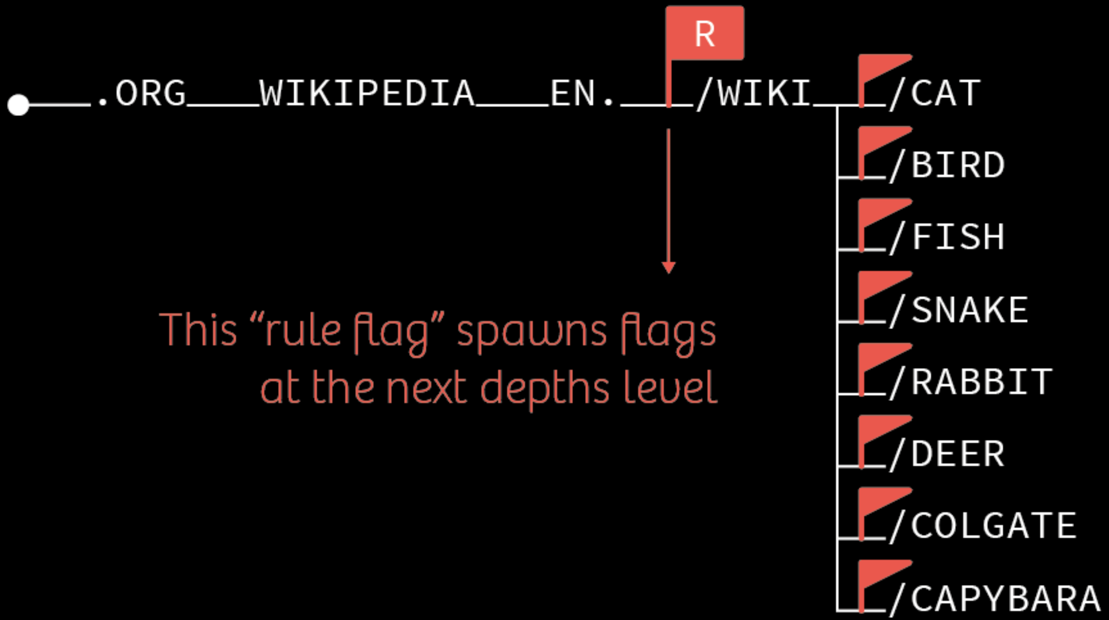
Bernhard studies how animals are represented on Wikipedia



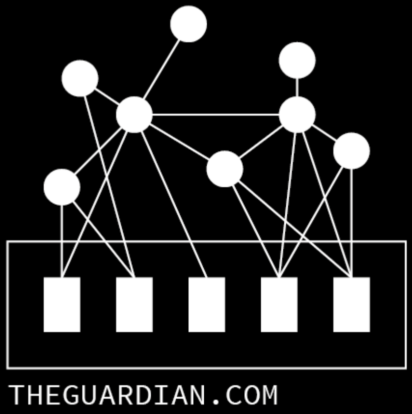
WIKIPEDIA

Web entities = DOCUMENTS

Web entities are Wikipedia articles

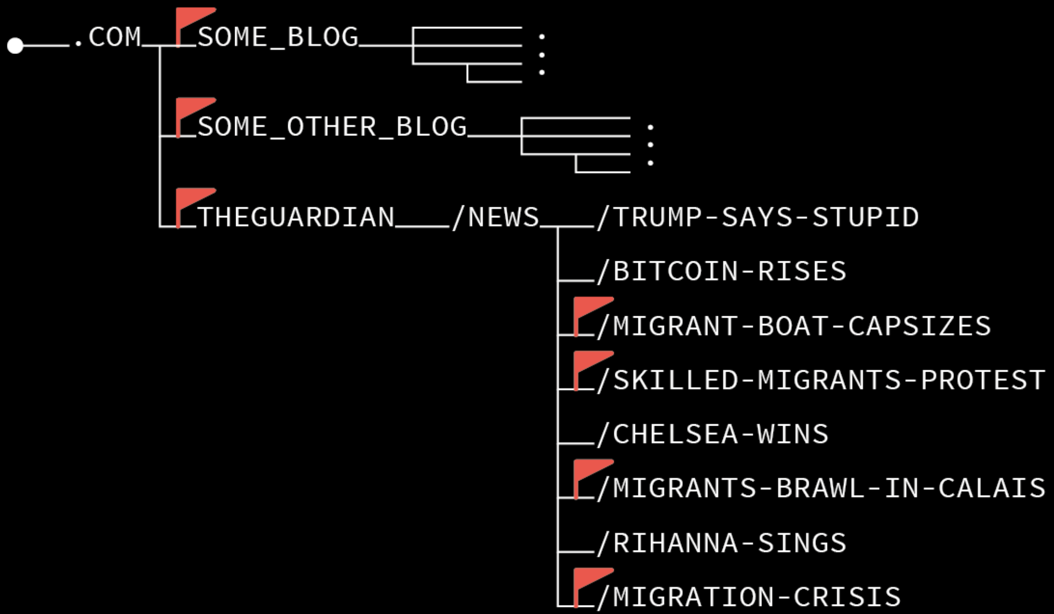


Carla studies how immigration is discussed in the public debate

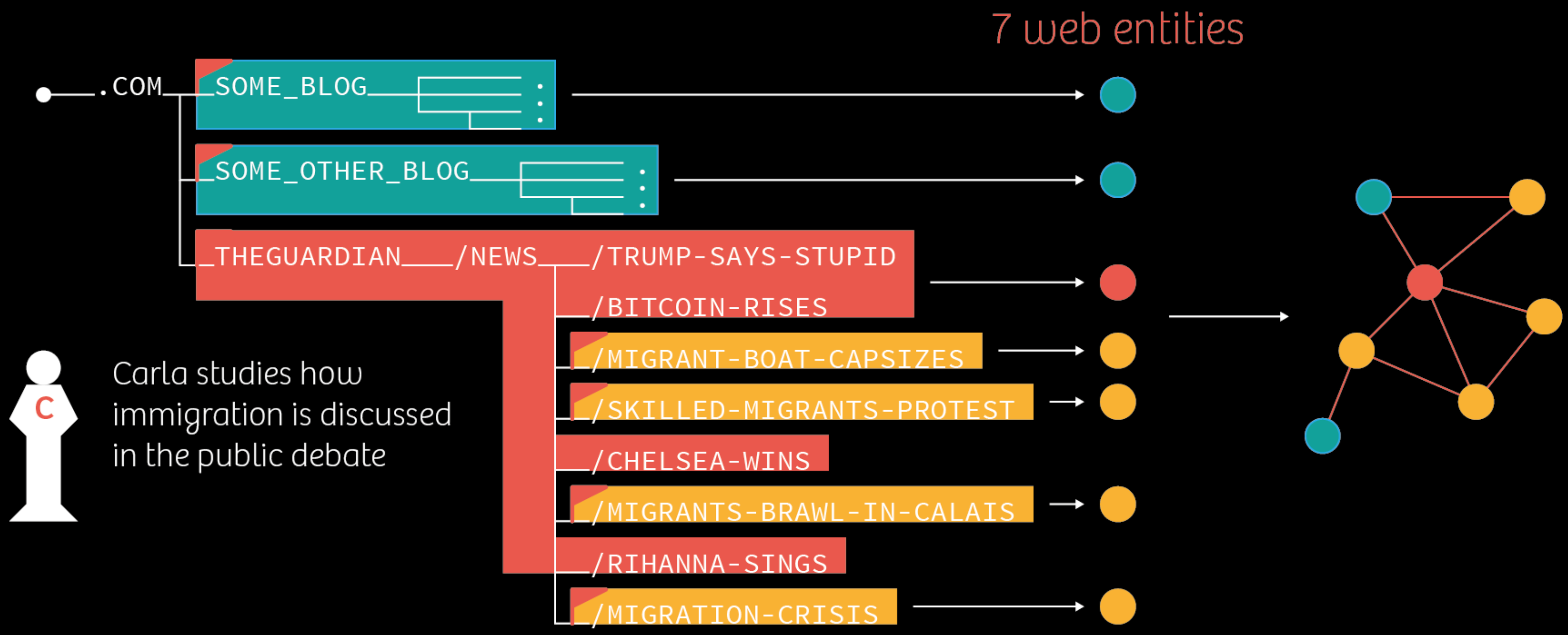


Web entities
= ACTORS
+ DOCUMENTS
Who cites what?

Web entities are websites, blogs, profiles (actors)
+ some special articles from the Guardian newspaper



A tree of URLs and a graph of links



Structure's requirements

- Add tens of millions of LRUs
- Add hundreds of millions of links
- Edit web entity boundaries (move the flags) without re-indexing
- Get all the pages of a web entity
- Get the web entity graph sitting on top of the pages' one

It's a tree

It's a graph

How to implement that?

I.

Lucene

A tree?

- index of pages
- filter by prefix

A graph?

- index of pages couples
- agregate links by couples of prefixes

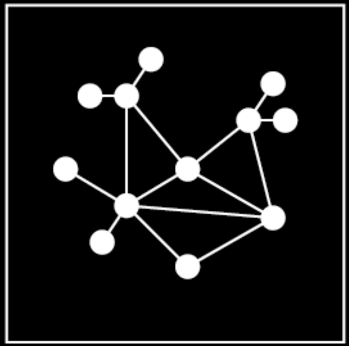
Problem

- Links between web entities are agregates
- web entities are dynamic

-> WE links should be computed, not stored

Remember Bernhard?

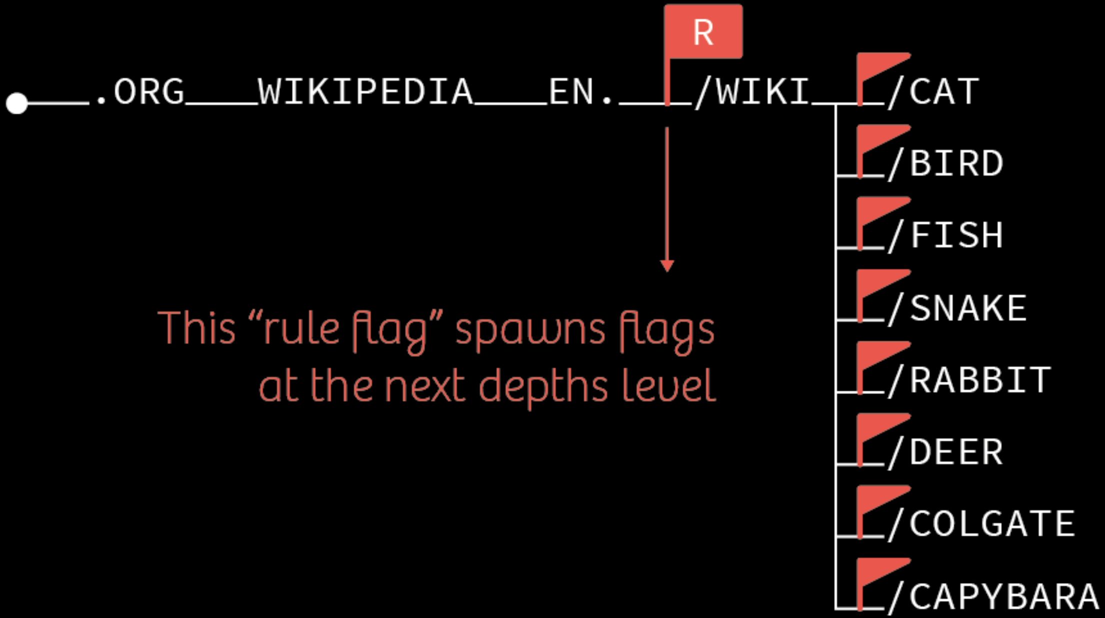
Bernhard studies how animals are represented on Wikipedia



WIKIPEDIA

Web entities = DOCUMENTS

Web entities are Wikipedia articles



This “rule flag” spawns flags at the next depths level

Limits

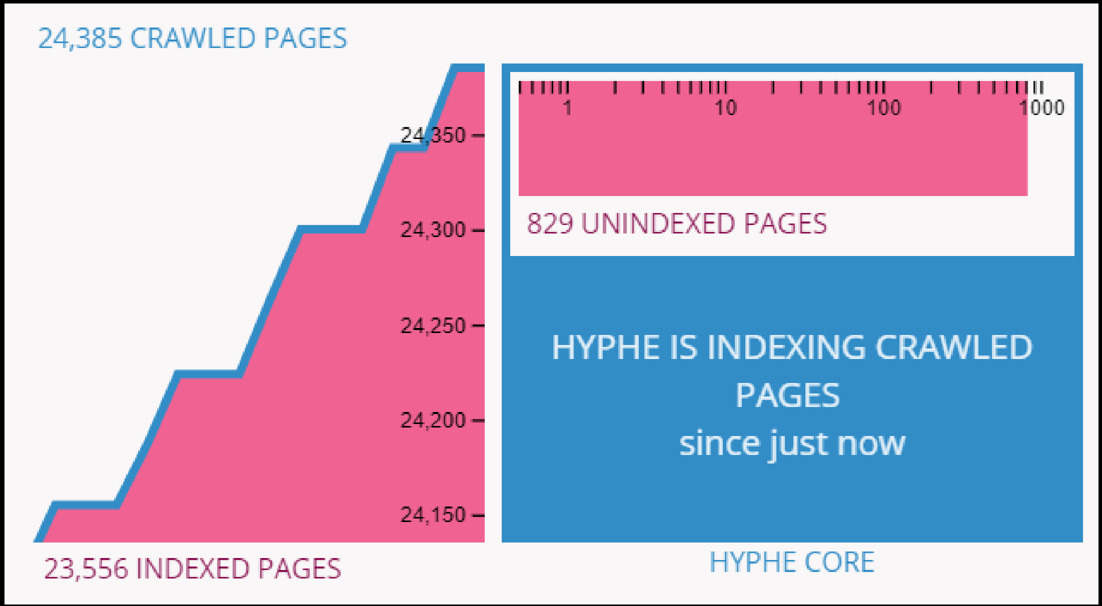
- Agregate links for list of prefixes
- but NOT for sub-prefixes!

-> complex slow queries

Turnarounds

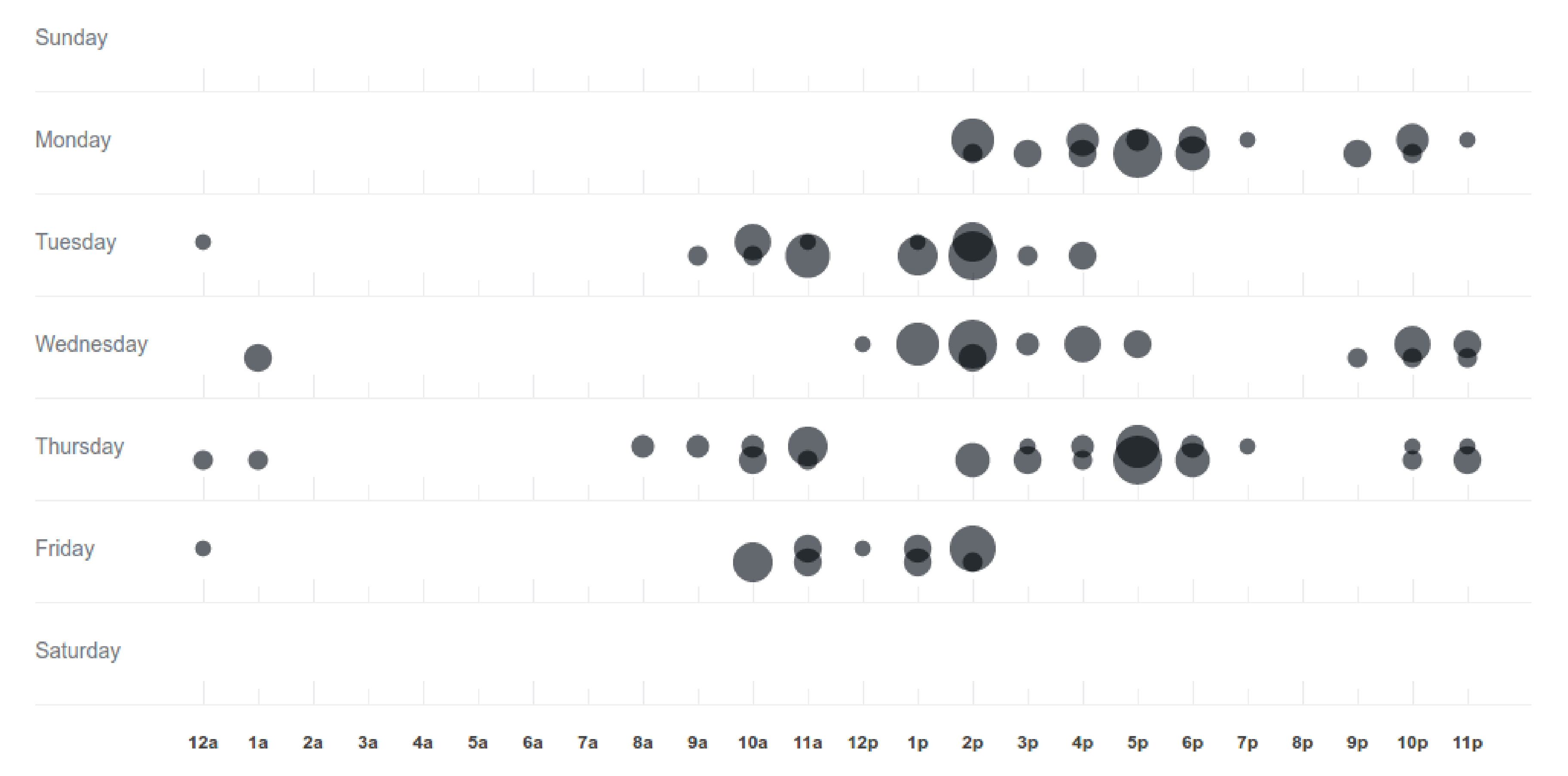
- Pages/Web entities links caching in Lucene
- Web entities links caching in RAM

indexation is slower than crawling...



II.
Coding retreat

- One week
- Four brains
- TANT LAB @Copenhaguen
- 2 prototypes
    - Neo4J POC
    - Java Tree POC

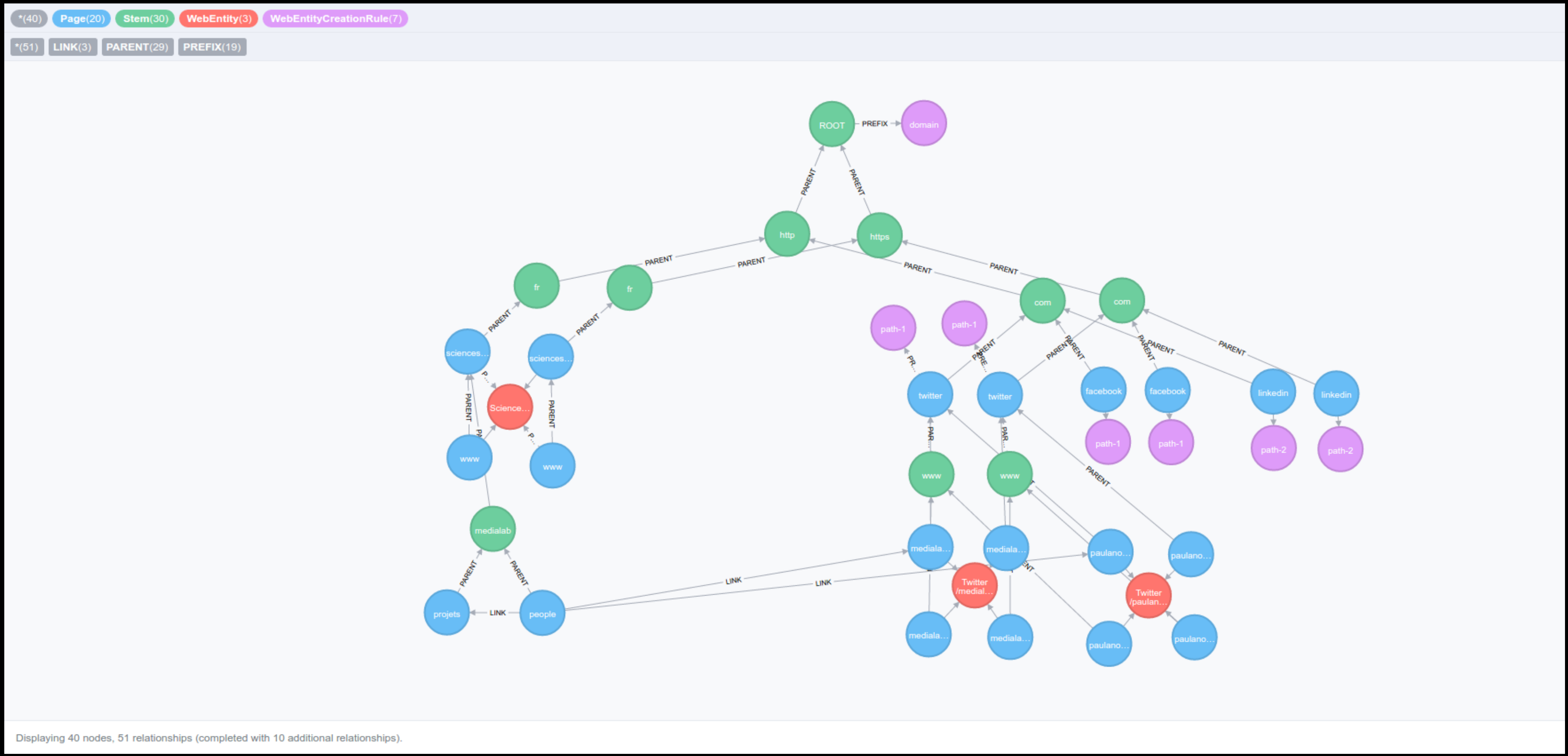

Sunday
Monday
Tuesday
Wednesday
Thursday
Friday
Saturday
12a
1a
2a
3a
4a
5a
6a
7a
8a
9a
10a
11a
12p
1p
2p
3p
4p
5p
6p
7p
8p
9p
10p
11p

IPA
Doszt?

III.

Prototype A - Neo4J

A tree? A graph?



Challenge: complex querying

- `UNWIND`
- `FOREACH`
- `REDUCE`
- `CASE`
- `COALESCE`
- stored procedures...

Indexing pages

```
// Indexing a batch of LRUs represented as lists of stems.
UNWIND $lrus AS stems
WITH [{lru: ""}] + stems AS stems
WITH stems[size(stems)-1].lru as lru, extract(n IN range(1, size(stems) - 1) | {first: stems[n - 1], second: stems[n]}) AS tuples
UNWIND tuples AS tuple

FOREACH (_ IN CASE WHEN NOT coalesce(tuple.second.page, false) THEN [1] ELSE [] END |
  MERGE (a:Stem {lru: tuple.first.lru})
  MERGE (b:Stem {lru: tuple.second.lru})
    ON CREATE SET
      b.type = tuple.second.type,
      b.stem = tuple.second.stem,
      b.createdTimestamp = timestamp()
  MERGE (a)<-[:PARENT]-(b)
)

FOREACH (_ IN CASE WHEN coalesce(tuple.second.page, false) THEN [1] ELSE [] END |
  MERGE (a:Stem {lru: tuple.first.lru})
  MERGE (b:Stem {lru: tuple.second.lru})
    ON CREATE SET
      b.type = tuple.second.type,
      b.stem = tuple.second.stem,
      b.createdTimestamp = timestamp(),
      b.crawledTimestamp = tuple.second.crawlTimestamp,
      b.crawlDepth = tuple.second.crawlDepth,
      b.linked = coalesce(tuple.second.linked, false),
      b:Page
    ON MATCH SET
      b.crawlDepth =
        CASE
          WHEN tuple.second.crawlDepth < b.crawlDepth
          THEN tuple.second.crawlDepth
          ELSE b.crawlDepth
          END,
      b.crawled = coalesce(tuple.second.crawled, b.crawled),
      b.linked = coalesce(tuple.second.linked, b.linked),
      b:Page
  MERGE (a)<-[:PARENT]-(b)
);
```

Links agregation V8 and 10 (out of 10)

```
// name: get_webentity_links_v8
MATCH path = (sourcePage:Page)-[:PARENT*0..]->(:Stem)-[:PREFIX]->(:WebEntity)
WITH sourcePage, path
ORDER BY length(path) ASC
WITH sourcePage, head(collect(last(nodes(path)))) AS sourceWe
MATCH (sourcePage)-[:LINK]->(targetPage:Page)
WITH sourcePage, targetPage, sourceWe, count(*) AS weight
MATCH path = (targetPage)-[:PARENT*0..]->(:Stem)-[:PREFIX]->(:WebEntity)
WITH sourcePage, targetPage, path, sourceWe, weight
ORDER BY length(path) ASC
WITH sourcePage, targetPage, sourceWe, head(collect(last(nodes(path)))) AS targetWe, weight
RETURN sourceWe.name, targetWe.name, sum(weight) AS weight;

[...]

// name: get_webentity_links_v10
MATCH (source:Page)-[:LINK]->(target:Page)
CALL hyphe.traverse(source) YIELD node AS sourceWe
CALL hyphe.traverse(target) YIELD node AS targetWe
RETURN sourceWe.name, targetWe.name;
```

It's not as straightforward to traverse trees in Neo4j as it seems.

IV.

Prototype B - The Traph

Designing our own on-file index

To store a somewhat complicated multi-level graph of URLs

People told us NOT to do it

It certainly seems crazy...

- Building an on-file structure from scratch is not easy.
- Why would you do that instead of relying on some already existing solution?
- What if it crashes?
- What if your server unexpectedly shuts down?

Not so crazy

- You cannot get faster than a tailored data structure (that's a fact).
- We don't need deletions (huge win!).
- No need for an **ACID** database (totally overkill).

We just need an index

- An index does not store any "original" data because...
- ...a MongoDB already stores the actual data in a reliable way.
- [insert joke about MongoDB being bad]
- This means the index can be completely recomputed and its utter destruction does not mean we can lose information.

So, what's a Traph?


IT'S A TRAPH

The traph is a "subtle" mix between a Trie and a Graph.

Hence the incredibly innovative name...

A Trie of LRUs

Carla studies how immigration is discussed in the public debate

Storing a Trie on file

Using fixed-size blocks of binary data (ex: 10 bytes).

We can read specific blocks using pointers in a random access fashion.

Accessing a specific's page node is done in `O(m)`.

```
[char|flags|next|child|parent|outlinks|inlinks]
```

A Graph of pages

The second part of the structure is a distinct file storing links between pages.

We need to store both out links and in links.

```
(A)->(B)

(A)<-(B)
```

Storing links on file

Once again: using fixed-sized blocks of binary data.

We'll use those blocks to represent a bunch of linked list of stubs.

```
[target|weight|next]
```

Linked lists of stubs

```
{LRUTriePointer} => [targetA, weight] -> [targetB, weight] ->
```

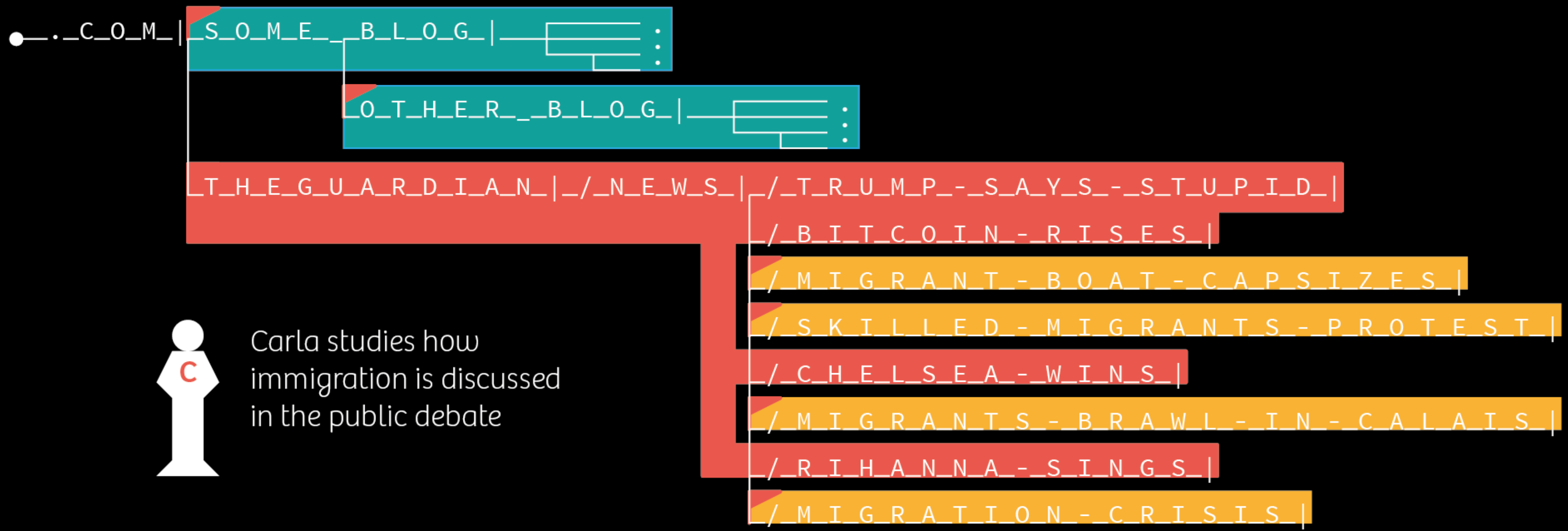
We can now store our links.

We have a graph of pages!

What about the multi-level graph?

What we want is the graph of **webentities** sitting above the graph of pages.

We "just" need to flag our Trie's nodes for webentities' starting points.



.COM
SOME_BLOG
OTHER_BLOG
THEGUARDIAN
/NEWS
/TRUMP-SAYS-STUPID
/BITCOIN-RISES
/MIGRANT-BOAT-CAPSIZES
/SKILLED-MIGRANTS-PROTEST
/CHELSEA-WINS
/MIGRANTS-BRAWL-IN-CALAIS
/RIHANNA-SINGS
/MIGRATION-CRISIS
C
Carla studies how
immigration is discussed
in the public debate

So now, finding the web entity to which belongs a page is obvious when traversing the Trie.

What's more, we can bubble up in `O(m)`, if we need to, when following pages' links (this can also be easily cached).


._C_O_M_|_S_O_M_E___B_L_O_G_|
_O_T_H_E_R___B_L_O_G_|
_T_H_E_G_U_A_R_D_I_A_N_|_/_N_E_W_S_|_/_T_R_U_M_P_-_S_A_Y_S_-_S_T_U_P_I_D_|
_/_B_I_T_C_O_I_N_-_R_I_S_E_S_|
_/_M_I_G_R_A_N_T_-_B_O_A_T_-_C_A_P_S_I_Z_E_S_|
_/_S_K_I_L_L_E_D_-_M_I_G_R_A_N_T_S_-_P_R_O_T_E_S_T_|
_/_C_H_E_L_S_E_A_-_W_I_N_S_|
_/_M_I_G_R_A_N_T_S_-_B_R_A_W_L_-_I_N_-_C_A_L_A_I_S_|
_/_R_I_H_A_N_N_A_-_S_I_N_G_S_|
_/_M_I_G_R_A_T_I_O_N_-_C_R_I_S_I_S_|
C
Carla studies how immigration is discussed in the public debate

What's more, if we want to compute the webentities' graph, one just needs to perform a DFS on the Trie.

This seems costly but:

- No other way since we need to scan the whole index at least once.
- The datastructure is quite lean and you won't read so much.

Was it worth it?

Benchmark on a 10% sample from a sizeable corpus about privacy.

- Number of pages: **1 840 377**
- Number of links: **5 395 253**
- Number of webentities: **20 003**
- Number of webentities' links: **30 490**


REST API
WebEnt
WELinks

Indexation time

- **Lucene** • 1 hour & 55 minutes
- **Neo4j** • 1 hour & 4 minutes
- **Traph** • 20 minutes

Graph processing time

- **Lucene** • 45 minutes
- **Neo4j** • 6 minutes
- **Traph** • 2 minutes 35 seconds

Disk space

- **Lucene** • 740 megabytes
- **Neo4j** • 1.5 gigabytes
- **Traph** • 1 gigabyte

After Copenhagen

We decided to redevelop the structure in **python** to limit the amount of different languages used by Hyphe's core.

We made some new discoveries on the way and improved the performance of the Traph even more.

https://github.com/medialab/hyphe-traph

Bonus section

- Single character trie is slow: stem level is better
- We had to find a way to store variable length stems
- Results were bad at beginning because of linked lists
- We had to organize children as binary search trees: this is a ternary search tree
- We tried to use auto-balancing BSTs but this was useless since crawl order generate enough entropy
- Finally we switched to using varchars(255) rather than trimming null bytes to double performance.

(Related slides are vertical)

The issue with single characters

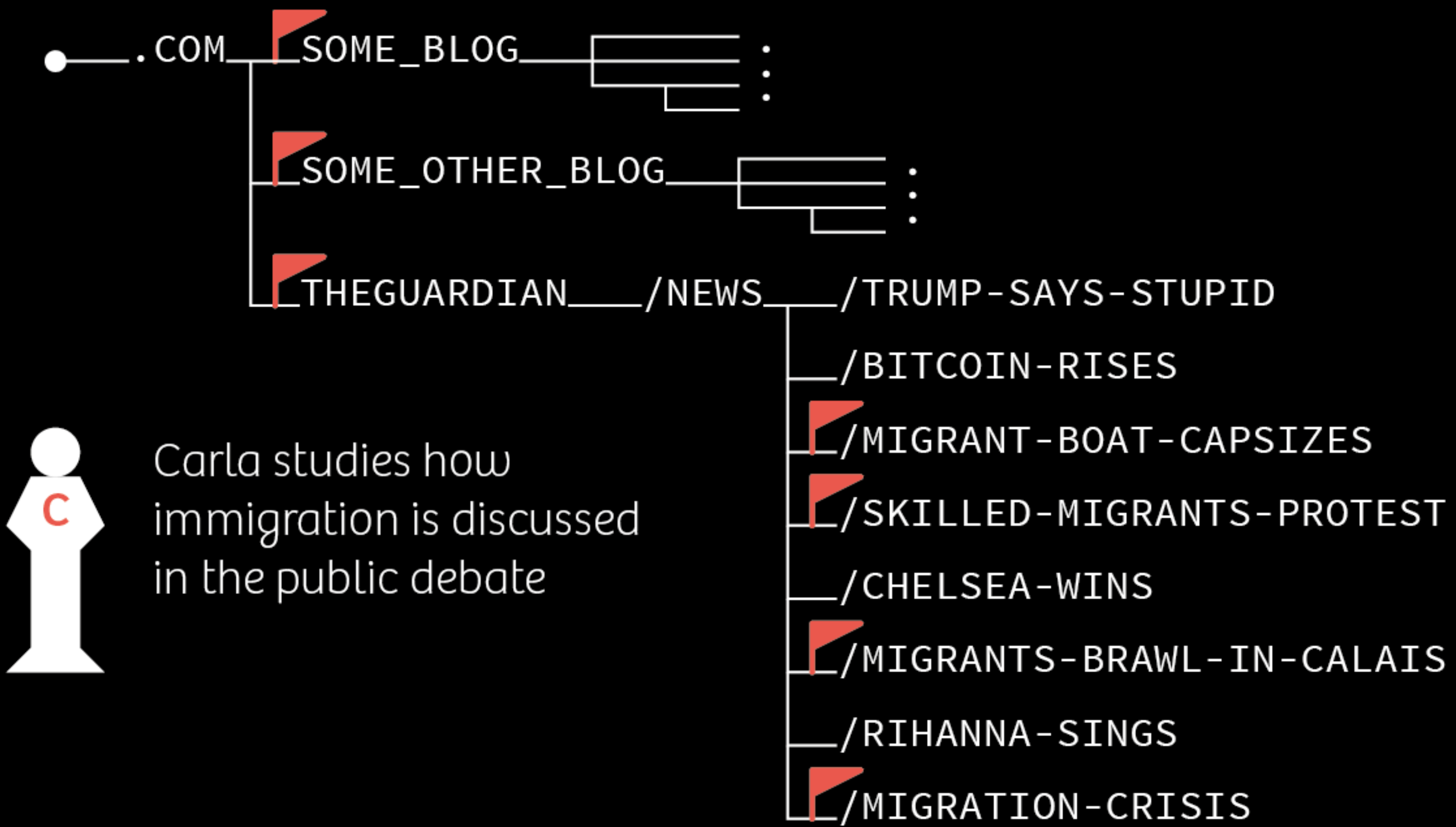
Our initial implementation was using single LRU characters as nodes.

Wastes a lot of spaces: more nodes = more pointers, flags etc.

More disk space = longer queries because we need to read more data from the disk.

We can do better: nodes should store LRU **stems**!


._C_O_M_|
_S_O_M_E_-_B_L_O_G_|
_O_T_H_E_R_-_B_L_O_G_|
_T_H_E_G_U_A_R_D_I_A_N_|_/_N_E_W_S_|
_/_T_R_U_M_P_-_S_A_Y_S_-_S_T_U_P_I_D_|
_/_B_I_T_C_O_I_N_-_R_I_S_E_S_|
_/_M_I_G_R_A_N_T_-_B_O_A_T_-_C_A_P_S_I_Z_E_S_|
_/_S_K_I_L_L_E_D_-_M_I_G_R_A_N_T_S_-_P_R_O_T_E_S_T_|
_/_C_H_E_L_S_E_A_-_W_I_N_S_|
_/_M_I_G_R_A_N_T_S_-_B_R_A_W_L_-_I_N_-_C_A_L_A_I_S_|
_/_R_I_H_A_N_N_A_-_S_I_N_G_S_|
_/_M_I_G_R_A_T_I_O_N_-_C_R_I_S_I_S_|
C
Carla studies how immigration is discussed in the public debate



Carla studies how immigration is discussed in the public debate

Fragmented nodes

Problem: stems can have variable length.

Fixed-size binary blocks => we need to be able to fragment them.

```
[stem|flags|next|parent|outlinks|inlinks] ... [tail?]
        ^
    has_tail?
```

Results were disappointing...

- **Character level** • 5 400 000 reads / 1 001 000 total blocks
- **Stem level** • 12 750 000 reads / 56 730 total blocks

Stem level had far less blocks and was orders of magnitudes lighter.

Strangely, it was way slower because we had to read a lot more.

Linked lists hell

Node's children stored as linked lists.

This means accessing a particular child is `O(n)`.

At character level, a list cannot be larger than `256` since we store a single ascii byte.

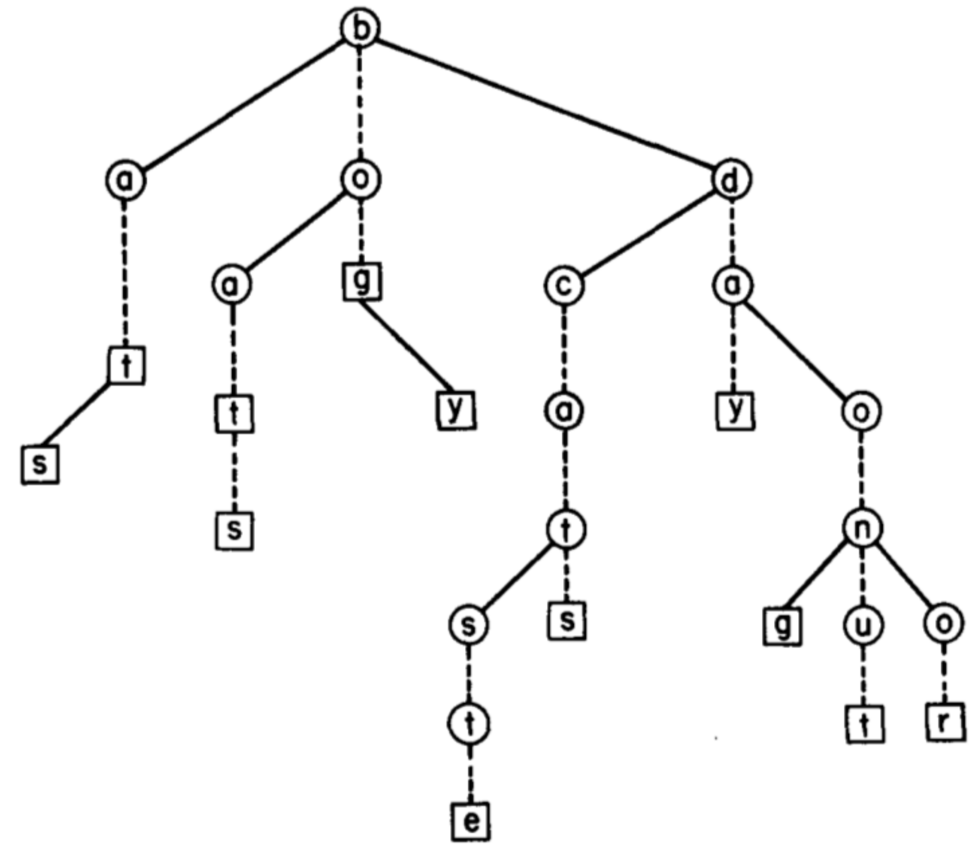
At stem level, those same linked lists will store a lot more children.

The Ternary Search Tree

We had to organize children differently.

We therefore implemented a Ternary Search Tree.

This is a Trie whose children are stored as binary search trees so we can access children in `O(log n)`.


b
a
o
d
t
s
a
g
c
a
t
y
a
y
o
s
t
n
s
s
g
u
o
t
t
r
e

Indexation time

- **Python character level traph** • 20 minutes
- **Python stem level traph** • 8 minutes

Graph processing time

- **Python character level traph** • 2 minutes 43 seconds
- **Python stem level traph** • 27 seconds

Disk space

- **Python character level traph** • 827 megabytes
- **Python stem level traph** • 270 megabytes

About balancing

Binary search trees can degrade to linked lists if unbalanced.

We tried several balanced BSTs implementations: treap & red-black.

This slowed down writes and did nothing to reads.

It seems that the order in which the crawled pages are fed to the structure generate sufficient entropy.

Takeaway bonus: varchars(255)

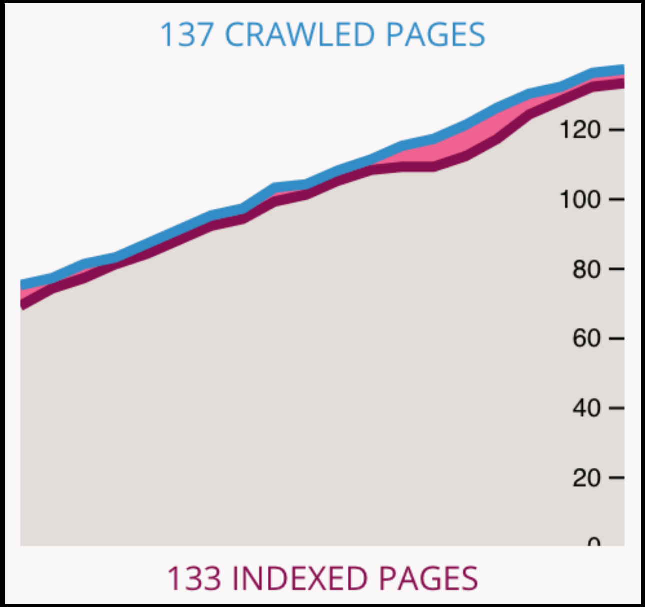
Sacrificing one byte to have the string's length will always be faster than manually dropping null bytes.

traph/lru_trie/node.py

```diff
@@ -20,7 +20,7 @@
 # architecture).
 
 # TODO: it's possible to differentiate the tail's blocks format if needed
-LRU_TRIE_NODE_FORMAT = '75sBI6Q'
+LRU_TRIE_NODE_FORMAT = '75pBI6Q'
 LRU_TRIE_NODE_BLOCK_SIZE = struct.calcsize(LRU_TRIE_NODE_FORMAT)
 LRU_TRIE_FIRST_DATA_BLOCK = LRU_TRIE_HEADER_BLOCKS * LRU_TRIE_NODE_BLOCK_SIZE
 LRU_TRIE_STEM_SIZE = 75
@@ -155,7 +155,7 @@ def read(self, block):
 
                 while True:
                     data = struct.unpack(LRU_TRIE_NODE_FORMAT, self.storage.read(current_block))
-                    chars = data[0].rstrip('\x00')
+                    chars = data[0]
 
                     chunks.append(chars)
 
@@ -250,7 +250,7 @@ def is_tail(self):
     def stem(self):
         chars = self.data[LRU_TRIE_NODE_STEM]
 
-        return chars.rstrip('\x00') + self.tail
+        return chars + self.tail
 
     def set_stem(self, stem):
 
```

Huge win! - 2x boost in performance.

Here we are now.
We went from 45 minutes to 27 seconds!



The web is the bottleneck again!

The current version of Hyphe uses this index in production!

A final mea culpa

Yes we probably used Lucene badly.

Yes we probably used Neo4j badly.

But. If you need to twist a system that much - by tweaking internals and/or using stored procedures - aren't you in fact developing something else?

But...

We are confident we can further improve the structure.

And that people in this very room can help us do so!

Thanks for your attention.