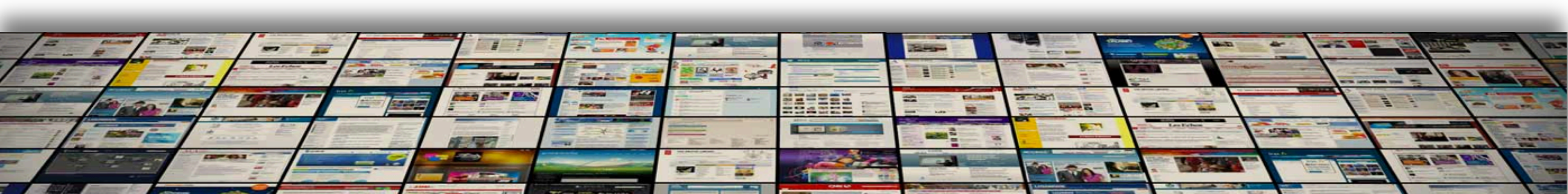
Large Scale Web Data Management at Internet Memory

Philippe Rigaux

Internet Memory Research

http://mignify.com

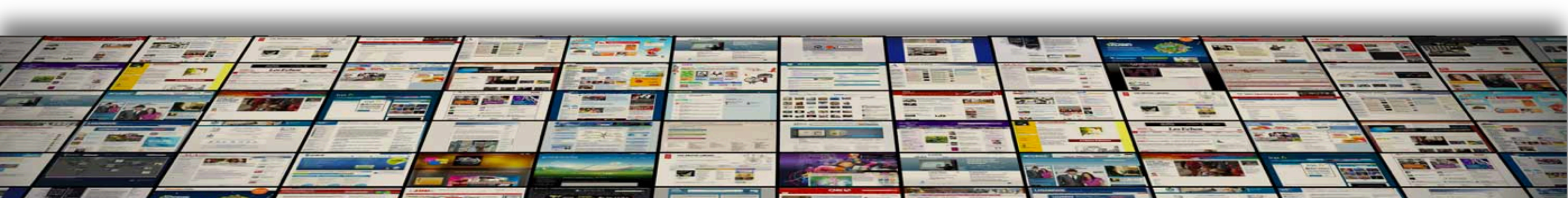


Internet Memory in brief

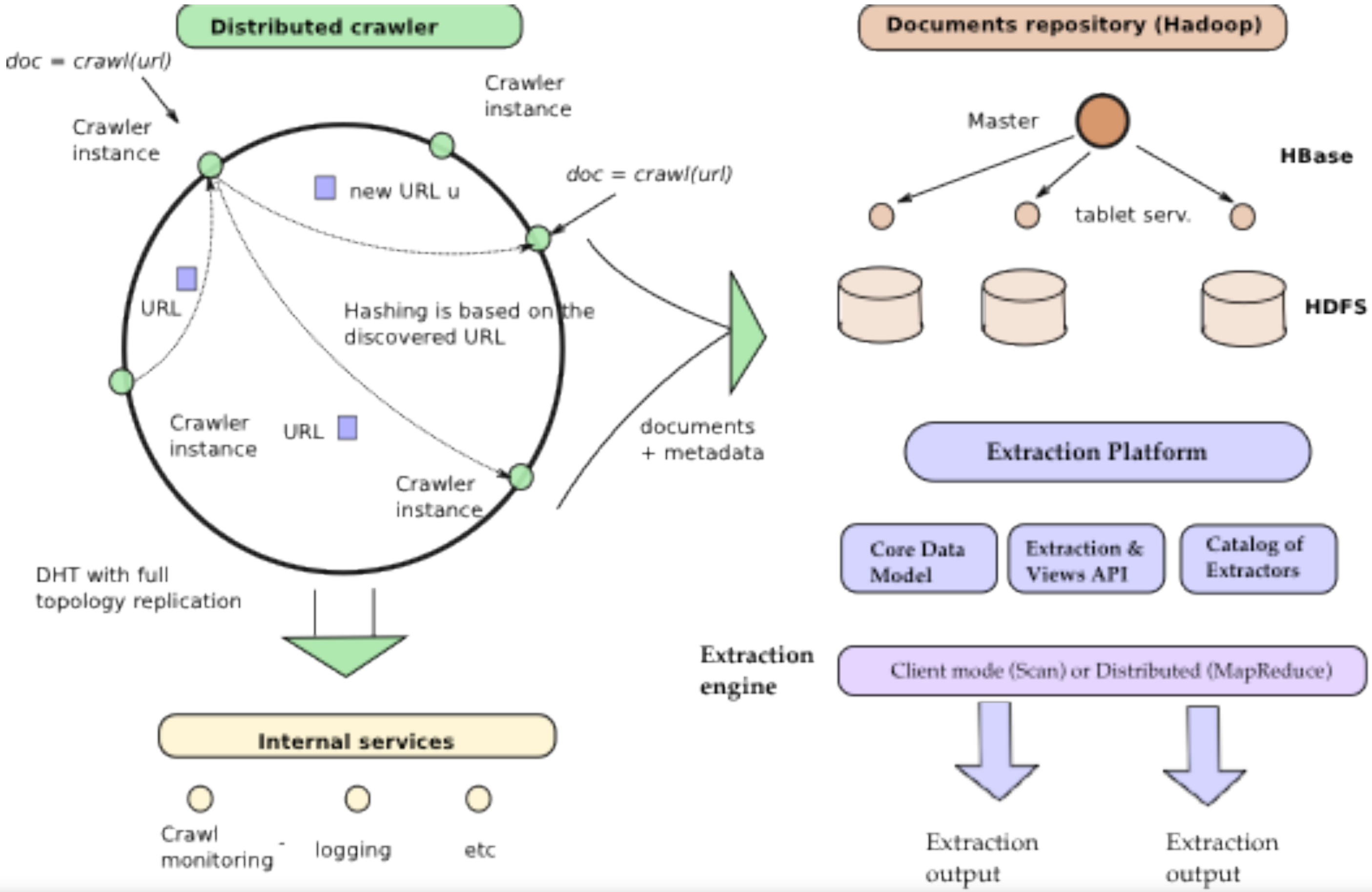
- Internet Memory Fundation (formerly European Archive)
  - A non-profit institution devoted to the preservation (archives) a Web collections
- Internet Memory Research,
  - a spin-off which provides services on Big datasets (crawls, analysis, classification, annotations)

What are we doing, and why?

- We collect and store collections of Web resources/documents
  - A Big Data Warehouse made from Web documents
  - We chose the Hadoop suite for scalable storage (HDFS), data modeling (HBase) and distributed computation (MapReduce)
- We support navigation in archived collections
- We produce structured content *extracted* (from single documents) and *aggregated* (from groups)



Overview of Mignify



Talk Outline

- Crawler (brief); document repository (brief)
- Design of Mignify
  - How we build an ETL platform on top of Hadoop
- Open issues
  - What we could probably do better
  - What we don’t do at all (for the moment)

Questions at any time !

Web-scale crawling issues (1)

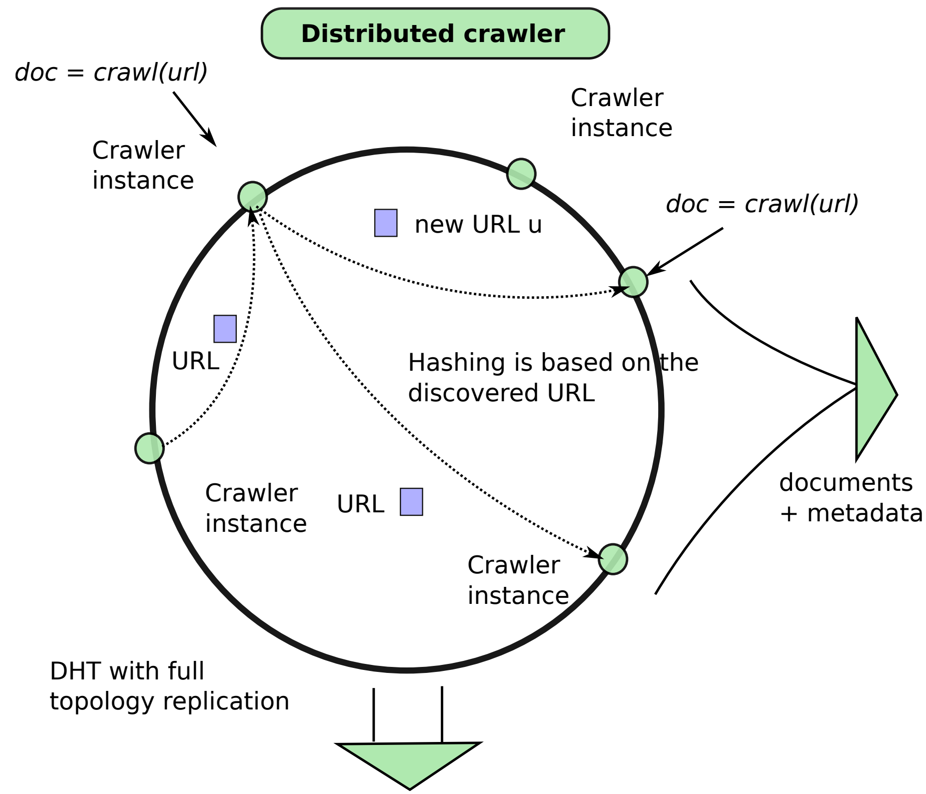
- For each found URL, we need to check that it has not already been found.
  - Known as the « frontier » problem
- We implemented a dedicated data structure
  - Relies on a bunch of efficient techniques: sequential scans, signatures, Bloom filters

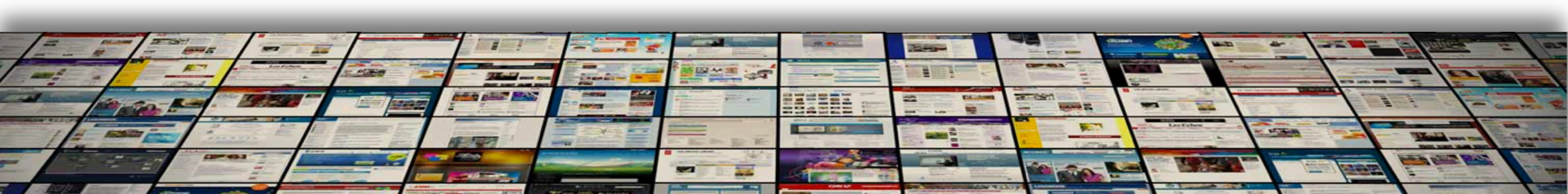
  (+) Sustains a throughput of ~100 docs/s

  (-) Huge latency.

Web-scale crawling issues (2)

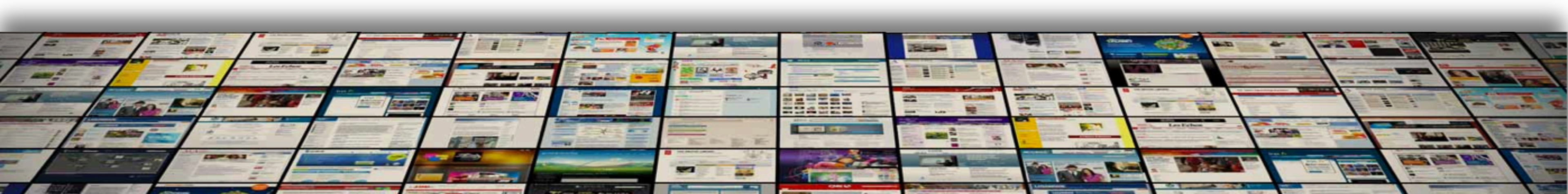
- We want to scale out (horizontal distribution)
- Solution based on consistent hashing
- Perf. Proportional to the number of participants





Our collections

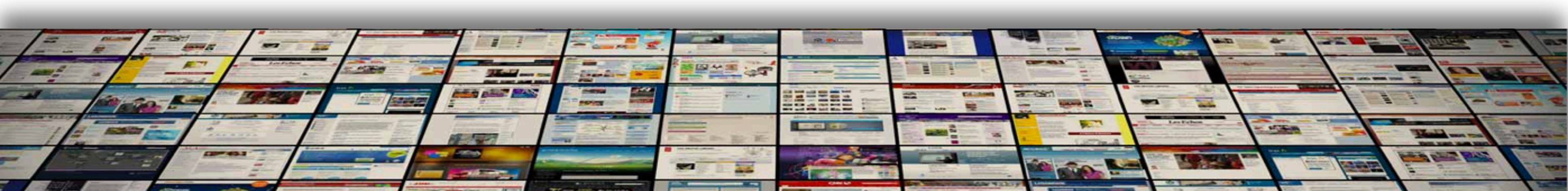
- Periodic (monthly) crawls of the Web graph
  - Ex: we can collect 1 Billion resources during a 3-weeks campaign, with a 10-servers cluster
- No complex operations at crawl time
  - A « collection » is a set of unfiltered Web docs = mostly garbage!
- We store and post-process collections
  - Size? Hundreds of TBs.



The document repository

Built on the three main components of Hadoop

- Hadoop Distributed File System (HDFS)
  - A FS specialized in large, append-only files, with replication and fault-tolerance
- HBase, a distributed document store
  - More to come soon
- MapReduce = distributed computing
  - Very basic, but designed for robustness

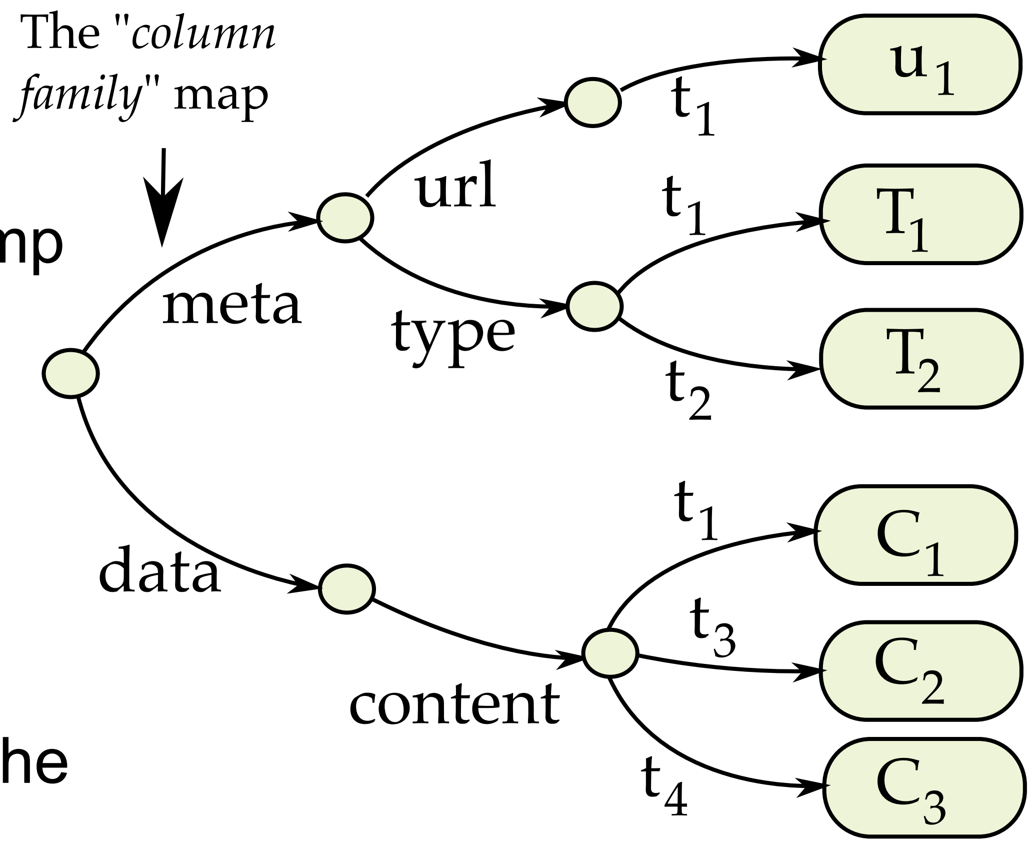


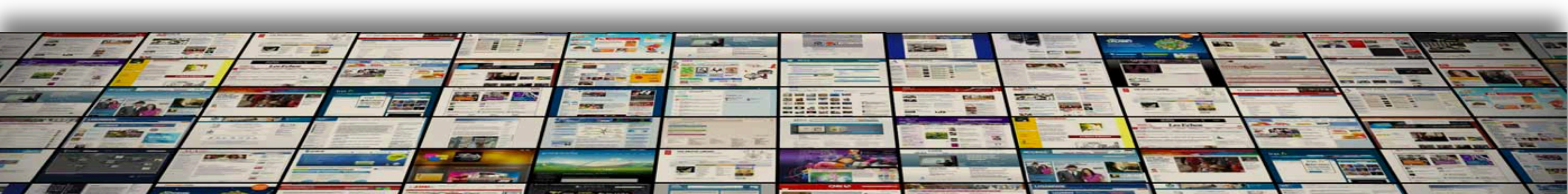
HBase values have a structure

A row has a three-levels structure:

- Each value has a timestamp
- Values are grouped in families
- And a row consists of several families.

NB: timestamps might be different from one value to the other in a same row.



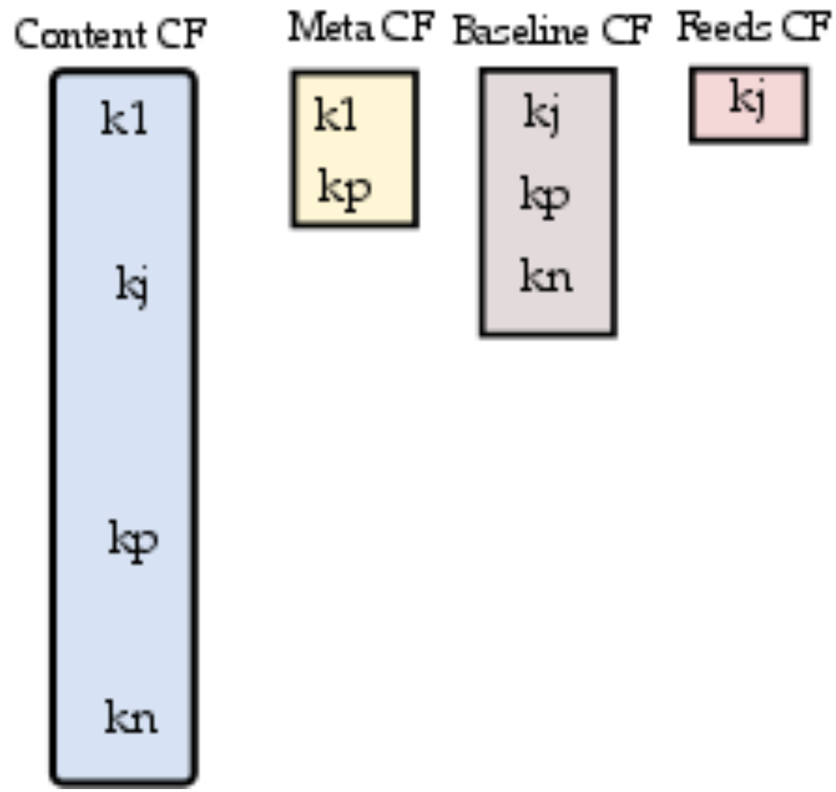


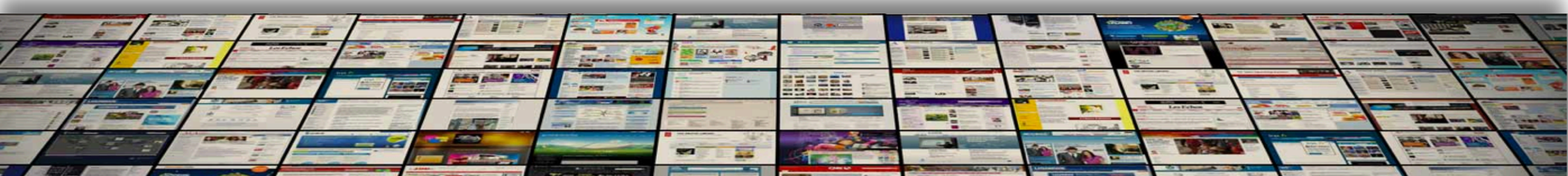
BTW, why do we have families?

Because families are stored indepently from one another (they just share the key)

- The content CF is the largest one.
- The Meta CF is smaller because values are smaller.
- The feeds is much smaller because of small values AND less rows represented

The smaller the CF = the more efficient any processing





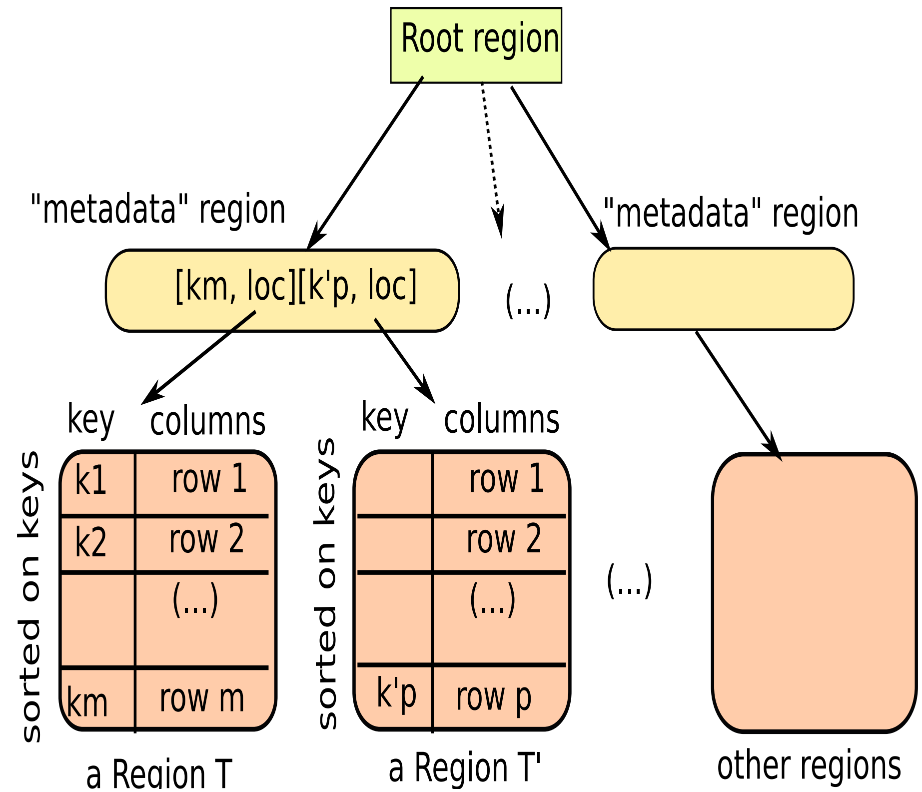
Some HBase internals

HBase sorts resources by *key*.

A set of resources in a given range constitutes a *region; regions* are assigned to *region servers*.

Key search? HBase quickly finds the region / server.

Scan? HBase distributes the scan to the region servers.



Hadoop essentials

Hadoop brings

- Linear scalability
- Data locality
  - All computations are « pushed » near the data
- Fault tolerance
  - The computation eventually finishes, even if a components fails,

… and it happens very often

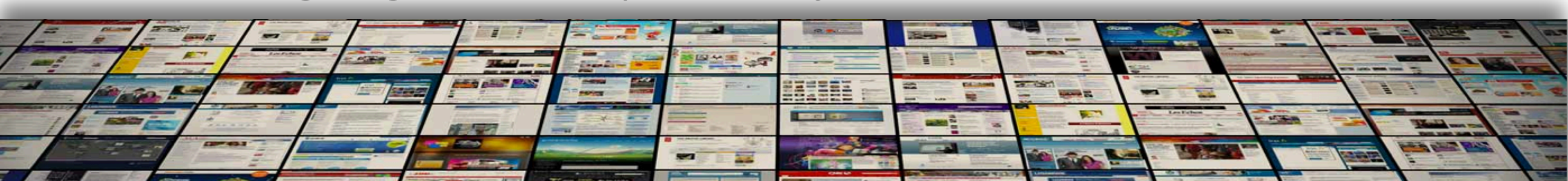
Mignify, a Web-scale Extraction Platform

Extraction: Typical scenarios

- Full text indexers
  - Collect documents, prepare for indexing
- Wrapper extraction
  - Get structured information from web sites
- Entity annotation
  - Annotate documents with entity references (eg, Yago)
- Classification
  - Aggregate subsets (e.g., domains); find relevant topics

Extraction Platform Main Principles

- A *framework* for *specifying* data extraction from very large datasets
  - Easy integration and application of new extractors
- High level of genericity in terms of (i) data sources, (ii) extractors, and (iii) data sinks
  - An extraction process « specification » combines these elements in so-called *views* and *subscriptions*.
- [currently] A single extractor engine
  - Based on the specification, data extraction is processed by a single, generic MapReduce job.



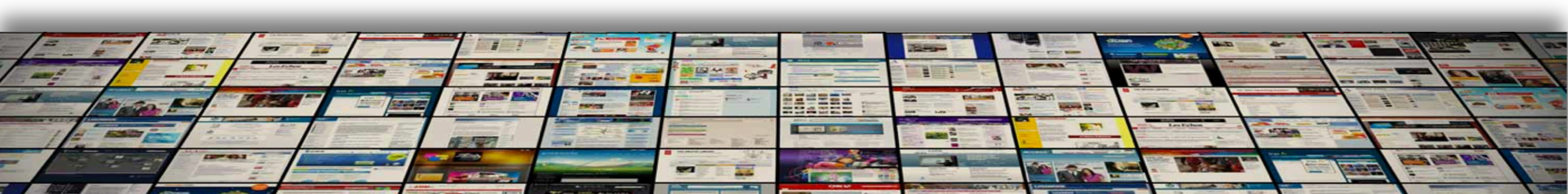
What is an Extractor

A software component which

- Takes a resource as input
- Produces a set of features as output
- Does so very efficiently

Some examples

- MIME type of a document
- Title and Plain text from HTML or PDF
- Title, author, date from RSS feeds

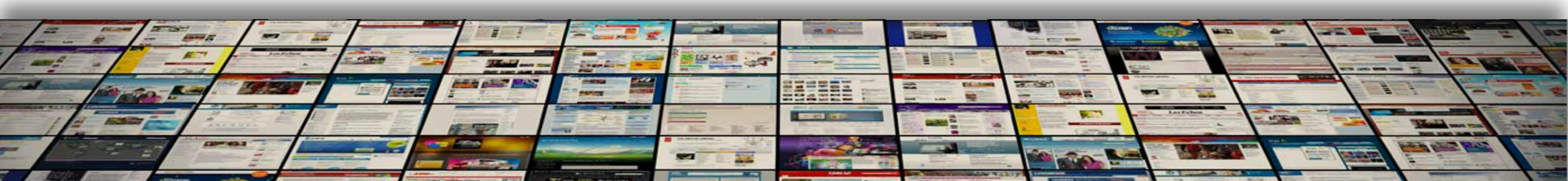


Extractors: Typical performance

Individual CPU cost (on an 2.5ghz i3)

	Time per Doc(ms)	*Docs per s*
Mime Type Extractor	0,52	1923,08
HTML Text Extractor	7,50	133,33
PDF Text Extractor	75,70	13,21
Content Shingle	9,90	101,01
Structure Shingle	9,23	108,34
Feed Extractor	1,2	833,33

NB: 1,000 res/s => 11 days of processing for 1 Billion docs (one server)



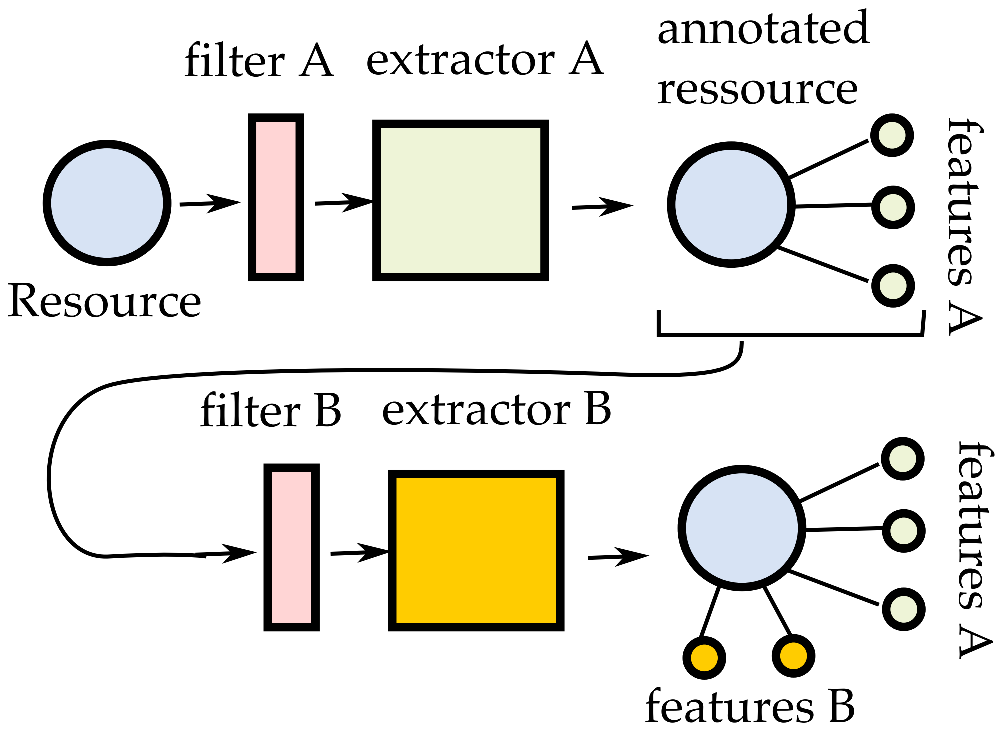
The Extraction Pipeline

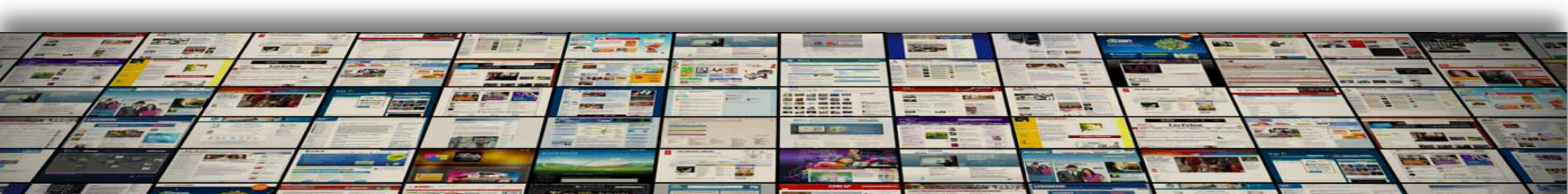
Given a *resource* as input, we apply a *filter* and an *extractor*

Each extractor produces *features* which annotate the resource.

The annotated resource is given as input to the next pair (filter, extractor)

And we iterate.





Aggregation / Classification

A software component which

- Takes a group of features (produced by extractors)
- Computes some aggregated value from this group
- A classifier is a special kind of aggregator

An example

- Group resources by domain
- Compute features with an extractor
- For each group, apply a SPAM classifier

What do we call « view »

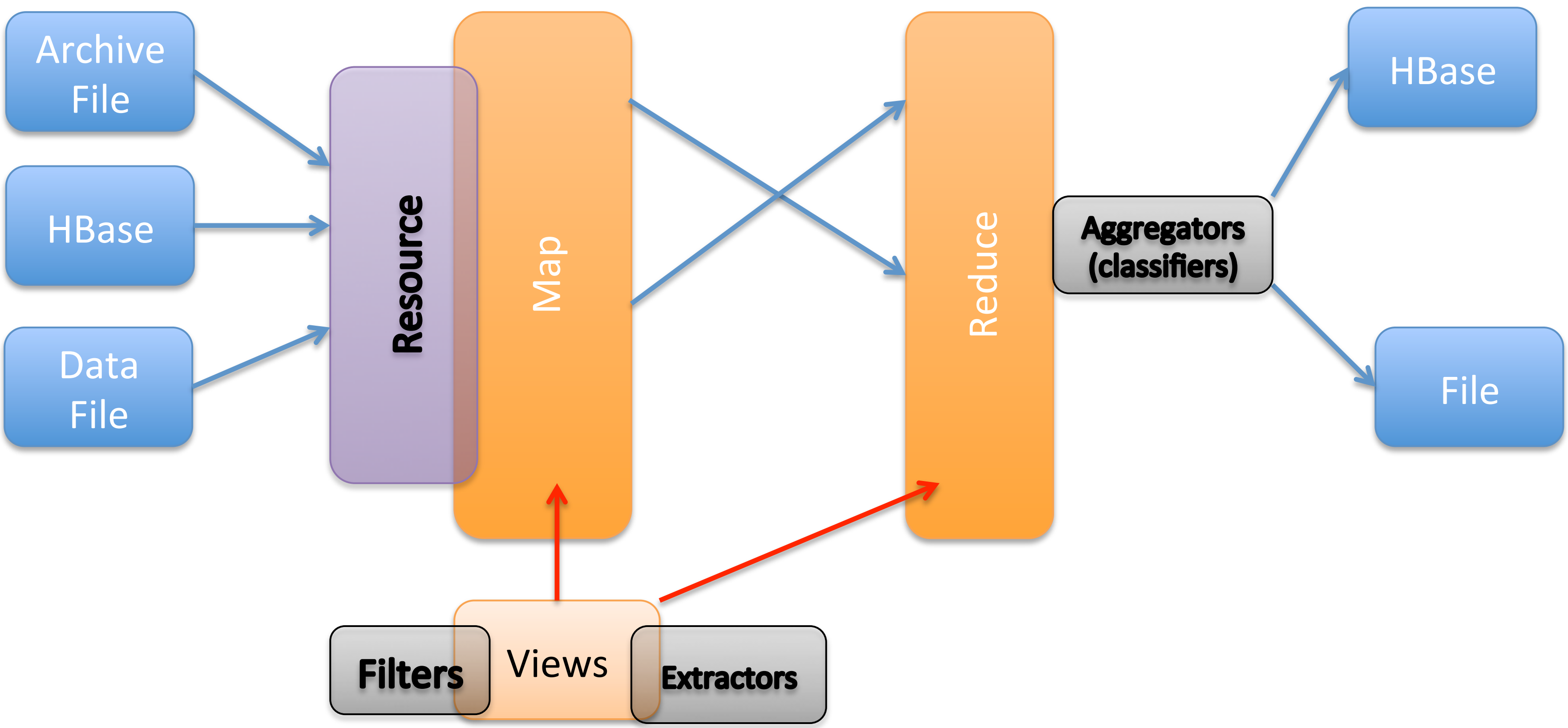
Extractions are specified as « views »

- The input (HBase, files, data flows, other…) and output (idem)
- The list of extractors to apply at resource level
- The list of aggregators to apply at group level (opt.)

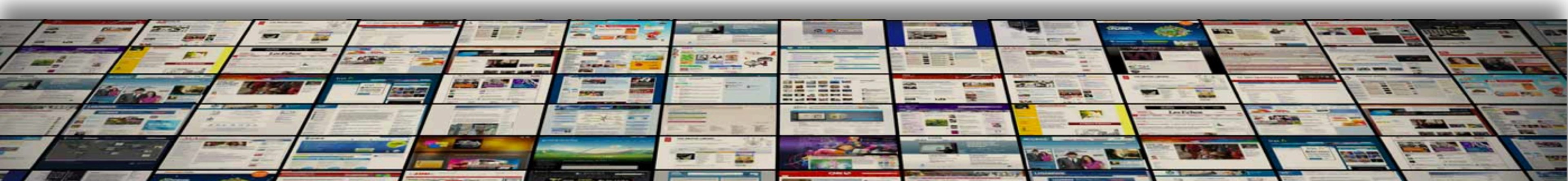
We attach views to collection, store and maintain the view results (*materialized views*).

Initially we compute the views with a generic MapReduce job

Evaluation: a generic MapReduce

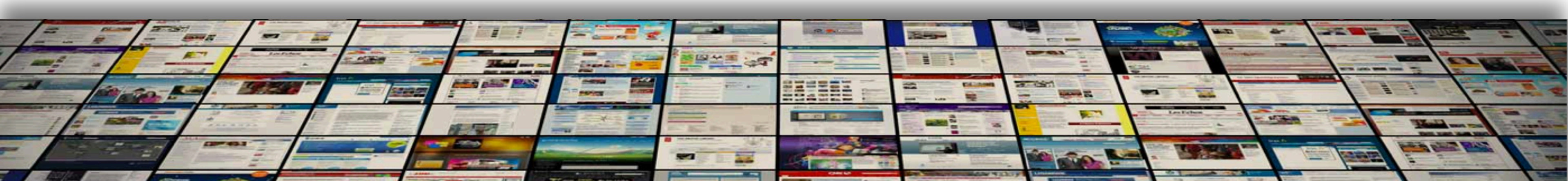


Lessons learned: could we do better, … and what we don't do at all



Revisiting our approach

- We started from the bottom
  - Storage, replication fault-tolerance: essential
  - A single primitive for computation. Robust but slow and very limited
- We added an « abstract » layer (data model, kind of declarative extraction queries)
  - In the wake other attempts (Pig, Hive); tailored to our needs
- And we wrote an engine that evaluates the queries with the primitive

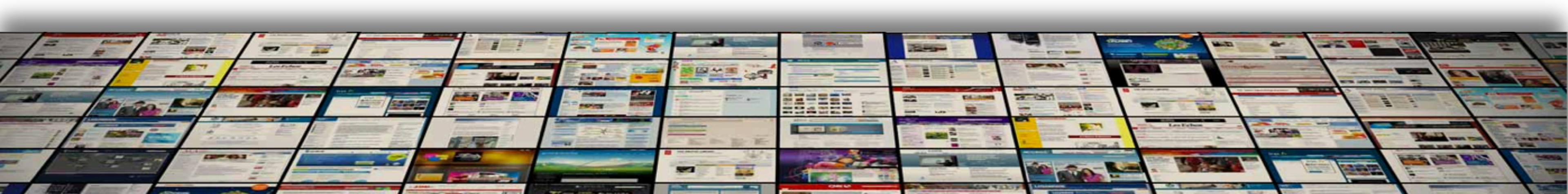


We need to be more expressive

- For the moment:
  - kind a multi-GROUP BY queries
  - Based on a single data source
- What if we want to combine multiple sources ?
  - Our annotations, and the ontology their refer to
- We need some syntactic extension (not essential)
- And we need a way to evaluate that: very painful with Hadoop!

We need a more powerful engine

- Basis of MapReduce
  - Two primitives which take user-defined functions (UDF, black boxes) as input
  - Designed for easy parallelisation (independence) and fault-tolerance (materialization)
- Can we extend the primitives to other second-order functions (eg, joins)
  - Yes, with additional constaints on the UDF
  - Work in progress with TU Berlin, Stratosphere



Views and indexes

- We produce materialized views
  - Stored independently
  - Much more compact than original data
- Could be used for
  - Query answering (that’s the goal)
  - And as access paths to raw documents

Example: « give me all Blobs in French devoted to Syrian civil war ».

INTERNET MEMORY

Conclusion

- Mignify = a big Data Warehouse with
  - Completely unstructed raw data (initially)
  - Targets Petabytes of data
- Many problems typical of DWHs
  - Extraction (ETL), aggregation, views…
- Need to be (partially) revisited in the context of very heterogenous data and large-scale distribution