HBase
Introduction to column oriented databases

Luís **Cipriani**
@lfcipriani (twitter, linkedin, github, ...)
22o. GURU (2012-02-25) - Sao Paulo/Brazil

Abril

ME

Abril MÍDIA

ALEXANDRIA

plataforma

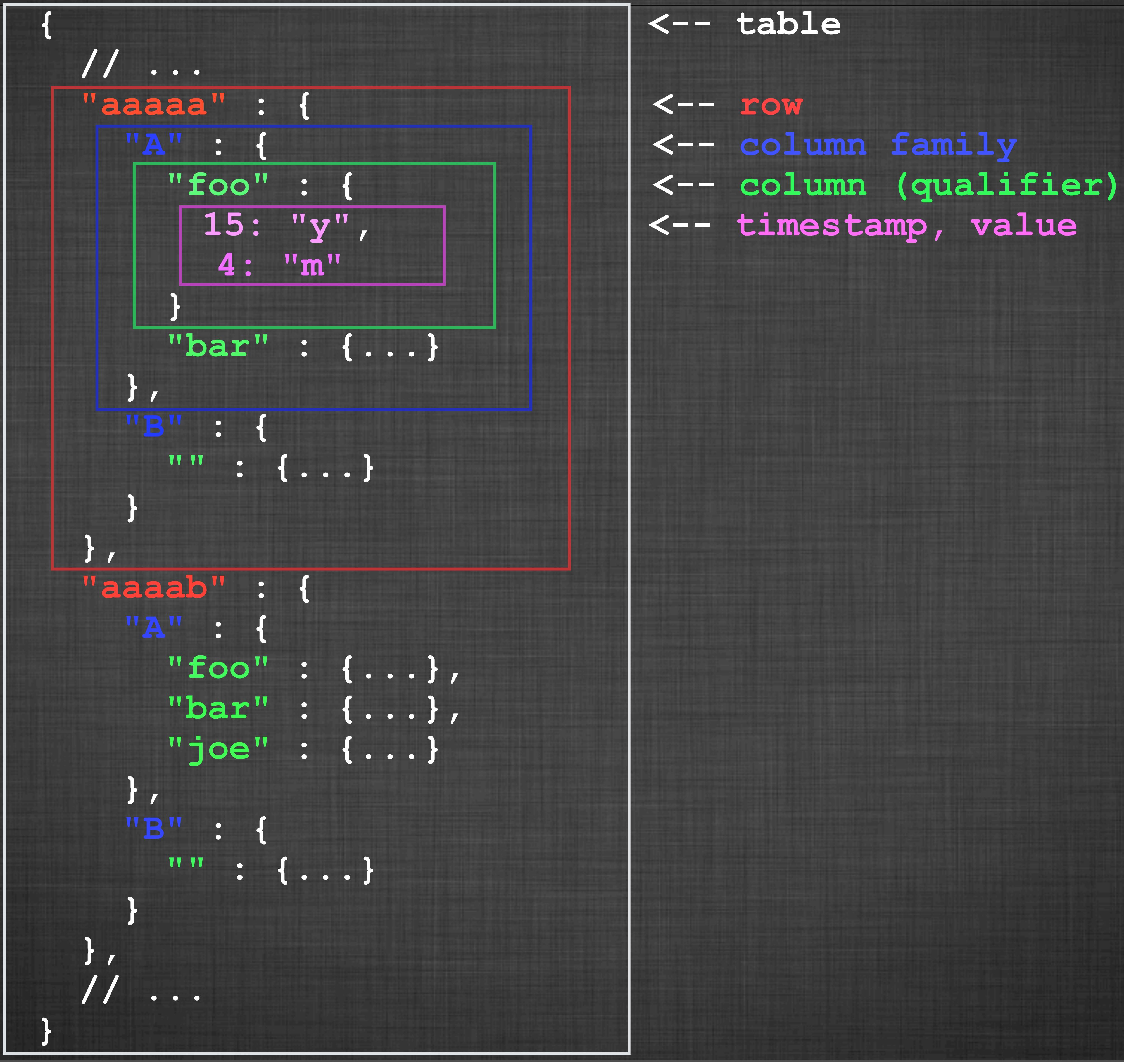
dev in SAMPA

intro

"~~A BigTable~~ HBase is a sparse, distributed, persistent multidimensional sorted map"

http://research.google.com/archive/bigtable.html

intro > data model



```
{                                     <-- table
  // ...
  "aaaaa" : {                         <-- row
    "A" : {                           <-- column family
      "foo" : {                       <-- column (qualifier)
        15: "y",                      <-- timestamp, value
         4: "m"
      }
      "bar" : {...}
    },
    "B" : {
      "" : {...}
    }
  },
  "aaaab" : {
    "A" : {
      "foo" : {...},
      "bar" : {...},
      "joe" : {...}
    },
    "B" : {
      "" : {...}
    }
  },
  // ...
}
```

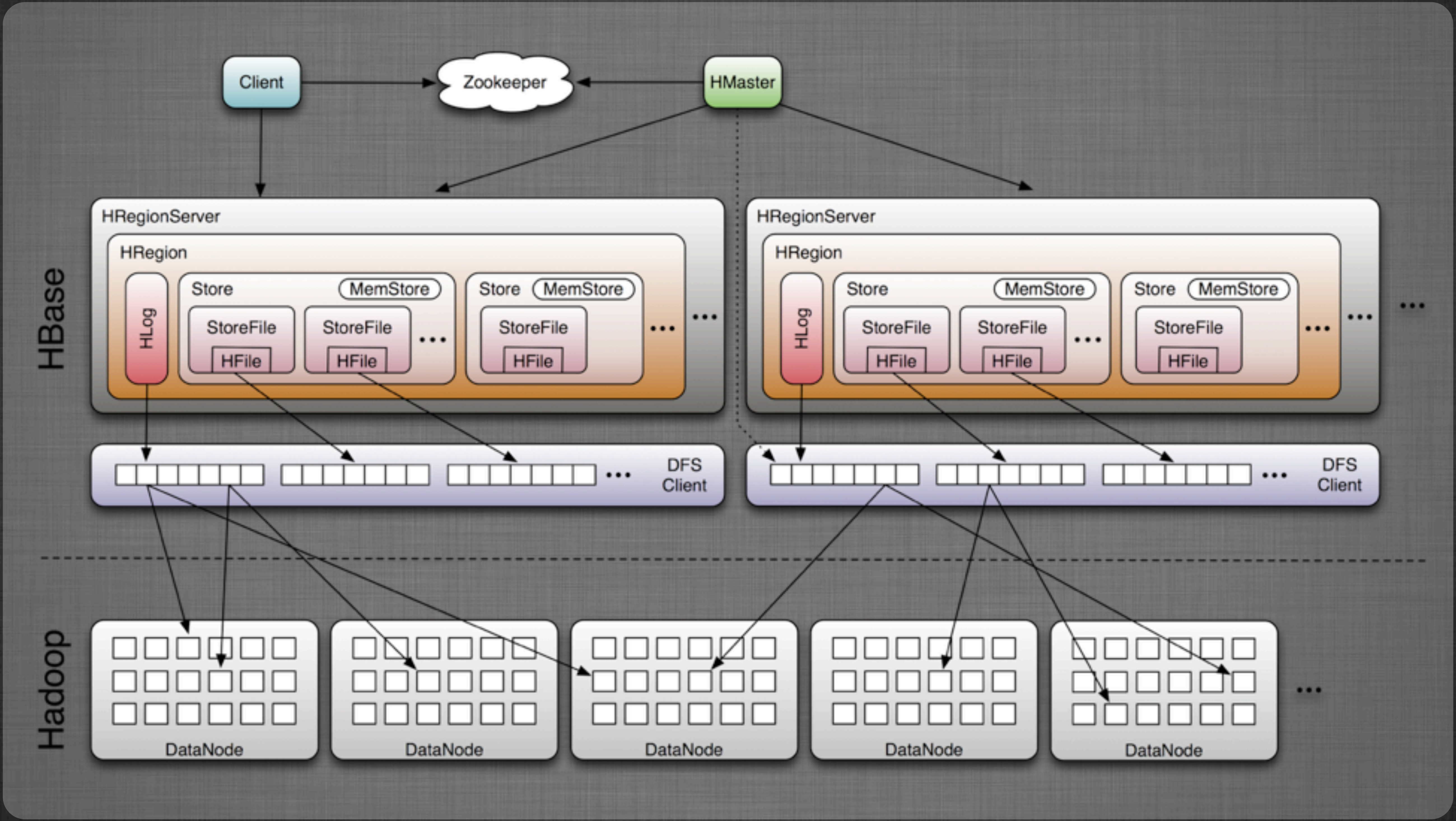
intro > data model

(Table, RowKey, Family, Column, Timestamp) → Value

intro > hadoop stack

- hadoop HDFS (or not)
- hadoop MapReduce
- hadoop ZooKeeper
- hadoop HBase
- hadoop Hue, Whirr, etc...

architecture

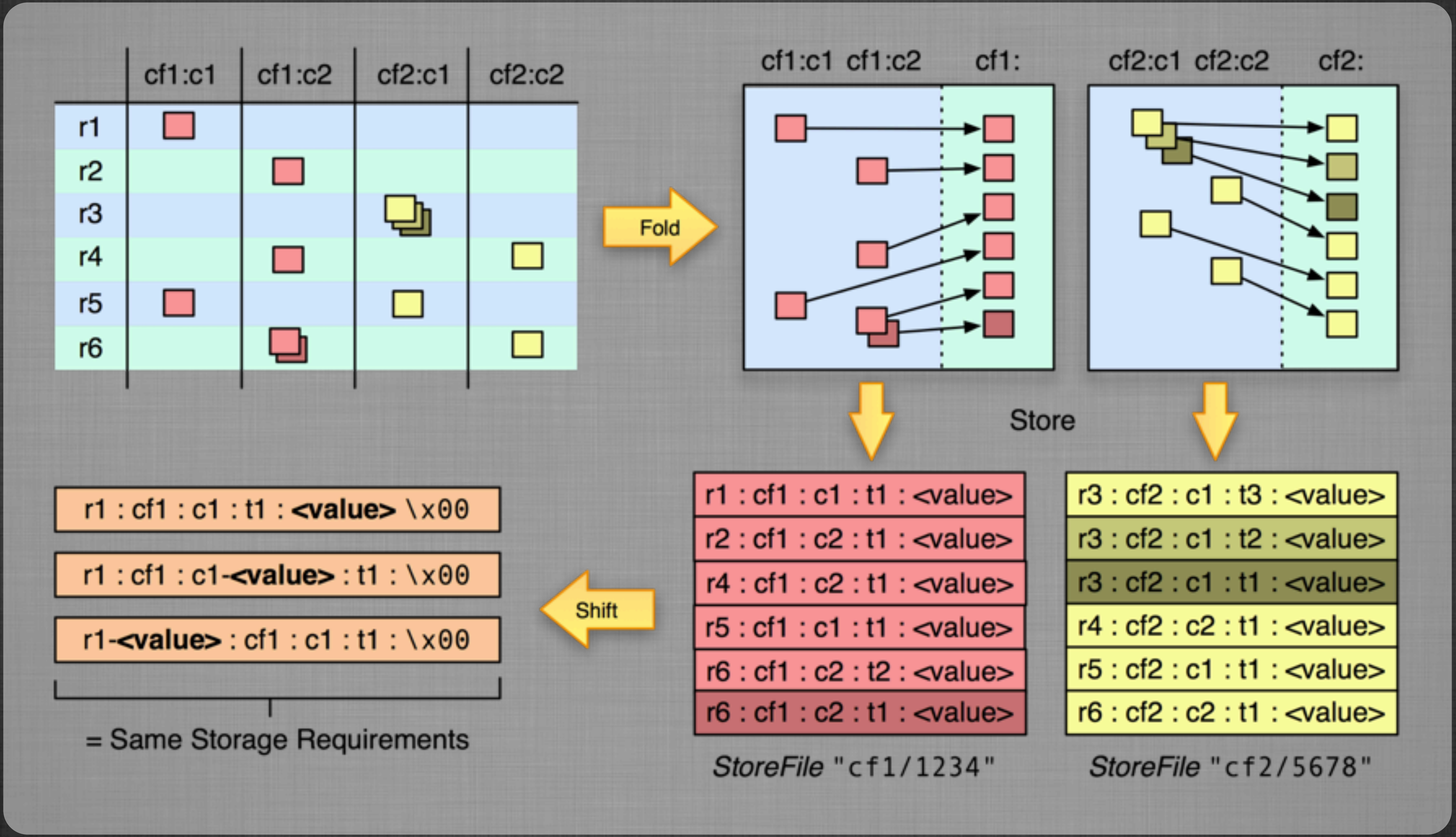


Abril

key design > read/write model

- randon reads (**get**)
- sequential reads (**scan**)
  - partial key scans
- writes (**put = update**)

key design > storage model



r1 : cf1 : c1 : t1 : **<value>** \x00

r1 : cf1 : c1-**<value>** : t1 : \x00

r1-**<value>** : cf1 : c1 : t1 : \x00

= Same Storage Requirements

r1 : cf1 : c1 : t1 : <value>
r2 : cf1 : c2 : t1 : <value>
r4 : cf1 : c2 : t1 : <value>
r5 : cf1 : c1 : t1 : <value>
r6 : cf1 : c2 : t2 : <value>
r6 : cf1 : c2 : t1 : <value>

StoreFile "cf1/1234"

r3 : cf2 : c1 : t3 : <value>
r3 : cf2 : c1 : t2 : <value>
r3 : cf2 : c1 : t1 : <value>
r4 : cf2 : c2 : t1 : <value>
r5 : cf2 : c1 : t1 : <value>
r6 : cf2 : c2 : t1 : <value>

StoreFile "cf2/5678"

http://ofps.oreilly.com/titles/9781449396107/advanced.html

key design > strategies

- tall-narrow vs flat-wide
- partial key scans
- pagination
- time series
  - salting
  - field swap
  - randomization
- secondary indexes

key design > example

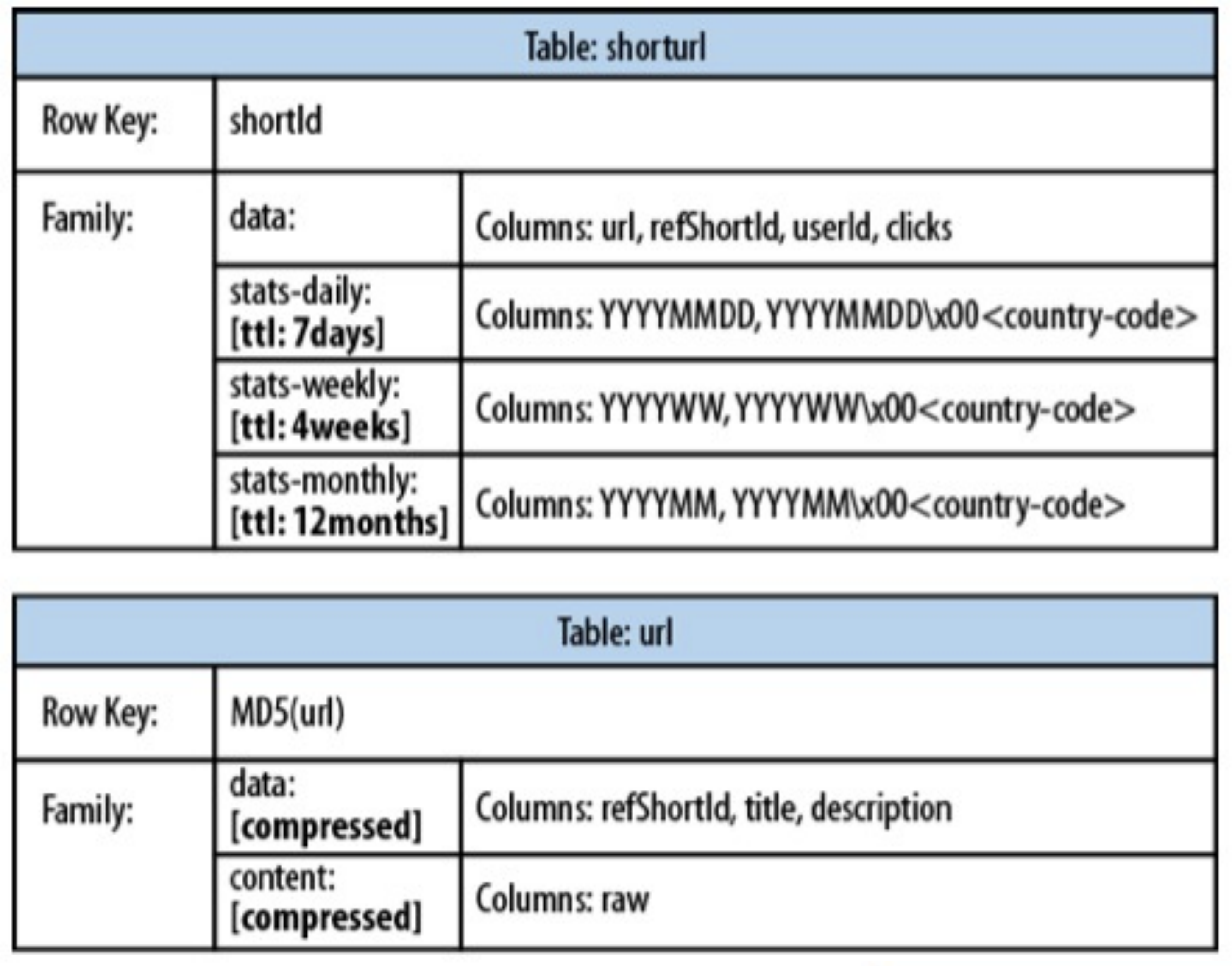


Table: shorturl		
Row Key:	shortId	
Family:	data:	Columns: url, refShortId, userId, clicks
	stats-daily: **[ttl: 7days]**	Columns: YYYYMMDD, YYYYMMDD\x00<country-code>
	stats-weekly: **[ttl: 4weeks]**	Columns: YYYYWW, YYYYWW\x00<country-code>
	stats-monthly: **[ttl: 12months]**	Columns: YYYYMM, YYYYMM\x00<country-code>

Table: url		
Row Key:	MD5(url)	
Family:	data: **[compressed]**	Columns: refShortId, title, description
	content: **[compressed]**	Columns: raw

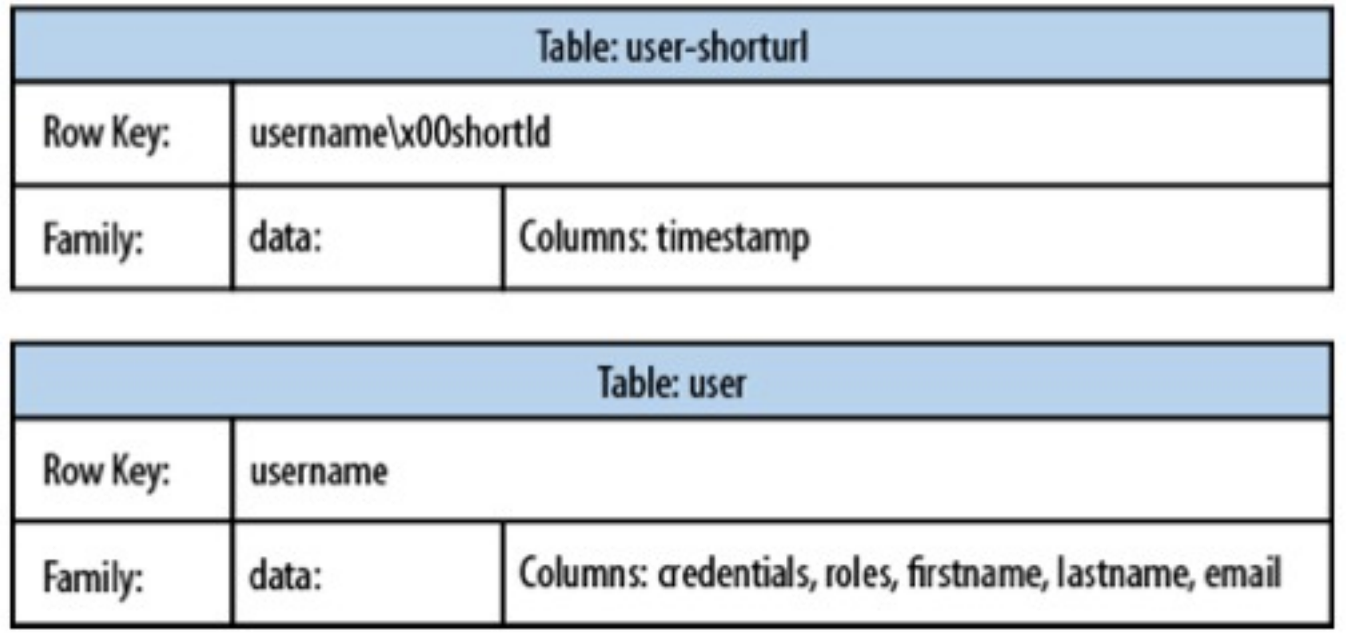


Table: user-shorturl		
Row Key:	username\x00shortId	
Family:	data:	Columns: timestamp

Table: user		
Row Key:	username	
Family:	data:	Columns: credentials, roles, firstname, lastname, email

development

- installation modes
  - standalone, pseudo-distributed, distributed
- JRuby console
- Access
  - java/jruby API (more features)
  - entrypoints REST, Thrift, Avro, Protobuffers
  - there several other libs

cons

- complex config and maintenance
- hot regions
- no secondary index built-in
- no transactions built-in
- complex schema design

pros

- distributed
- scalable (auto-sharding)
- built on Hadoop stack
- handles Big Data
- high performance for write and read
- no SPOF
- fault tolerant, no data loss
- active community

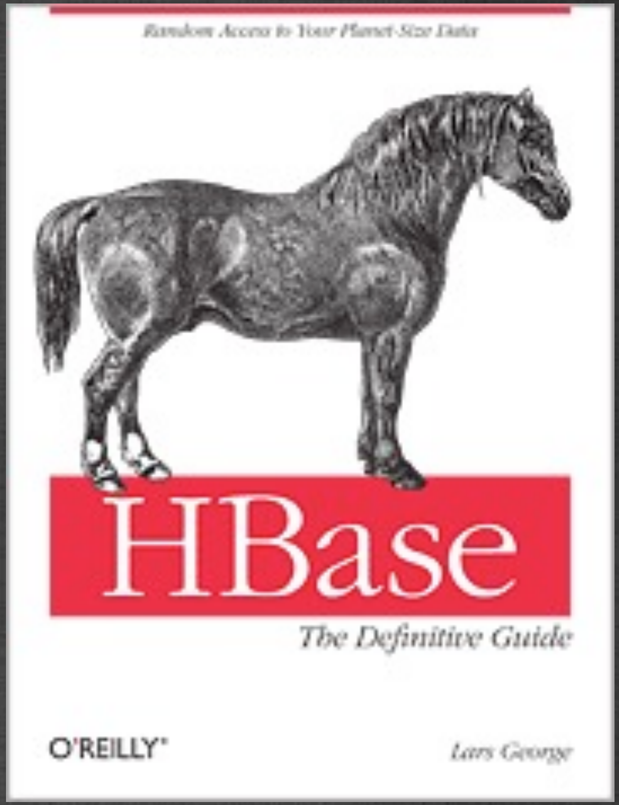
http://engineering.abril.com.br/

http://abr.io/hbase-intro

https://pinboard.in/u:lfcipriani/t:hbase/

http://hbase.apache.org/

?



http://shop.oreilly.com/product/0636920014348.do

Abril