Dremel: Interactive Analysis of Web-Scale Datasets

Sergey Melnik, Andrey Gubarev, Jing Jing Long, Geoffrey Romer, Shiva Shivakumar, Matt Tolton, Theo Vassilakis (2010)
In *Proceedings of the VLDB Endowment*

Wenjing Lin
Role: Paper Author

01

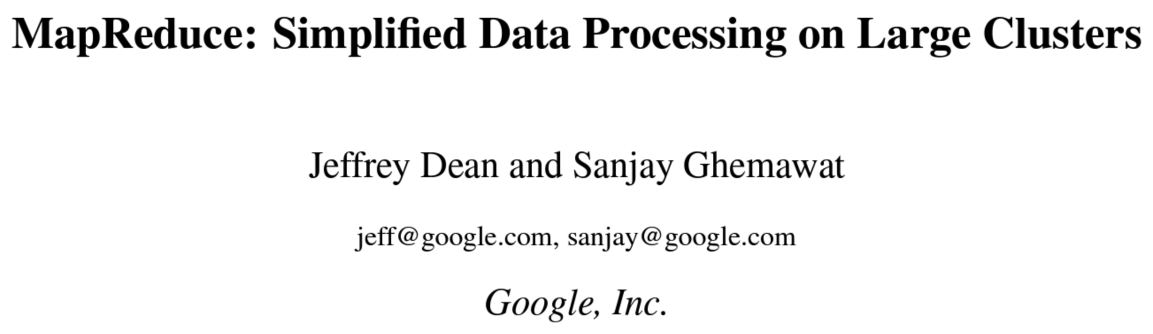
Introduction

Berkeley
UNIVERSITY OF CALIFORNIA

01 Introduction

Back in 2006

📊 "Big data" has become widespread, yet non-relational



MapReduce: Simplified Data Processing on Large Clusters

Jeffrey Dean and Sanjay Ghemawat

jeff@google.com, sanjay@google.com

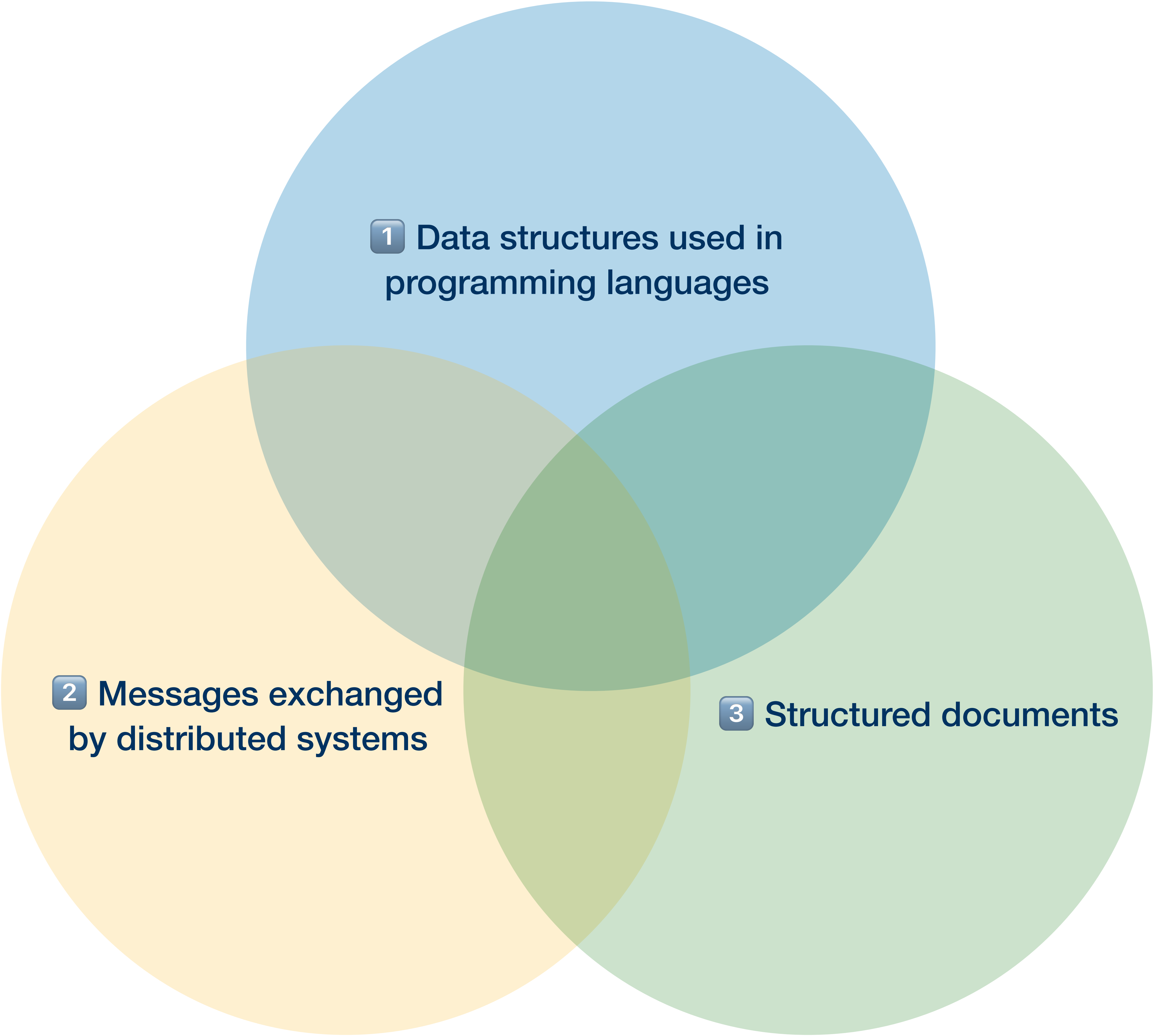
Google, Inc.

- **1998** — Google went live
- **2001** — Wikipedia launched
- **2004** — "The" Facebook launched; MapReduce paper published
- **2005** — Youtube launched
- **2006** — 🏄 **Dremel in production**; Hadoop released
- **2007** — iPhone launched

Berkeley
UNIVERSITY OF CALIFORNIA

🔗 Nested data underlies most structured data processing

Data used in web and scientific computing



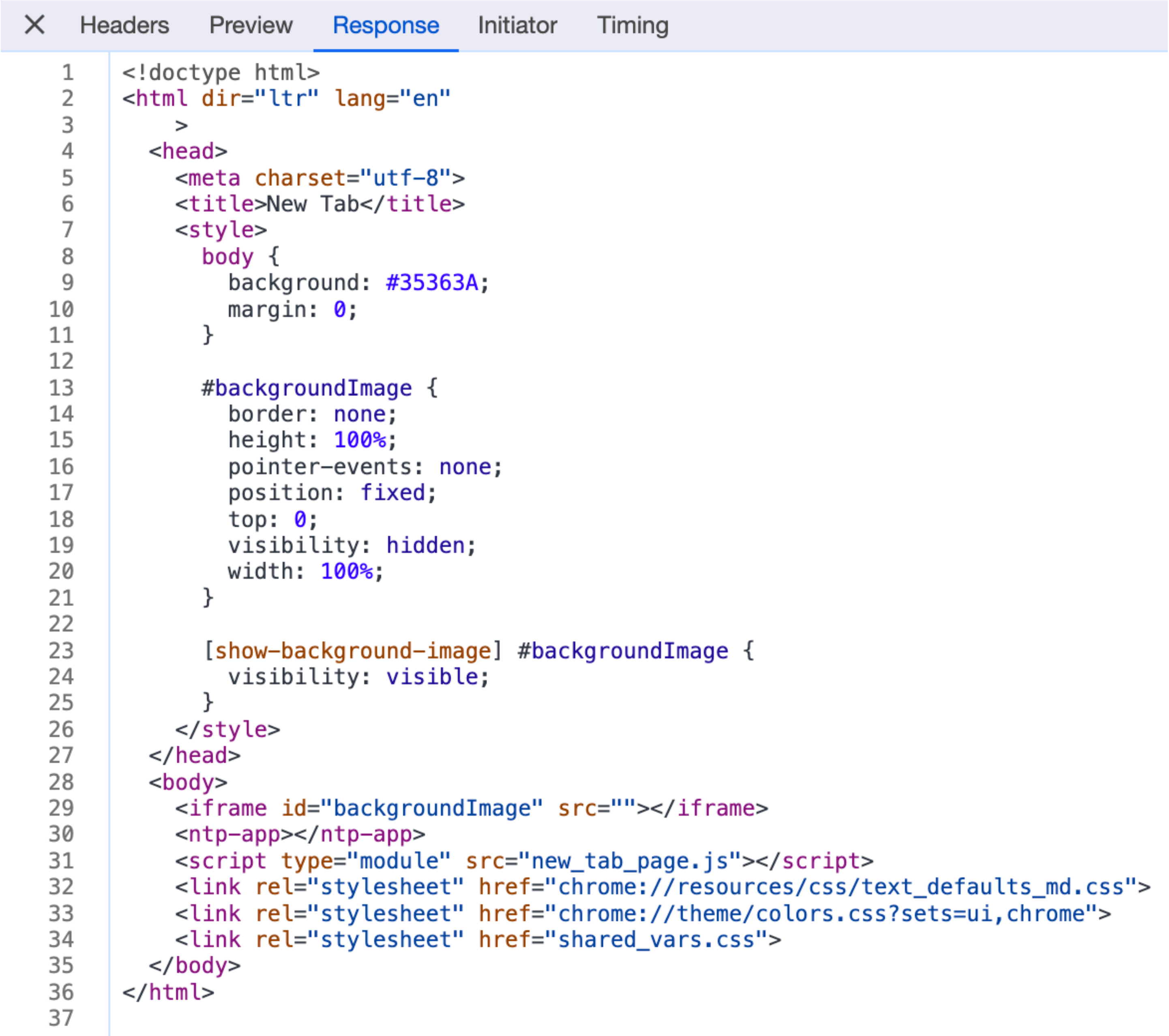
Nested representation

❌ Normalize & Recombine

✅ *In situ* operation

Dremel in Rescue

✅ Dremel supports operation on *in situ* nested data

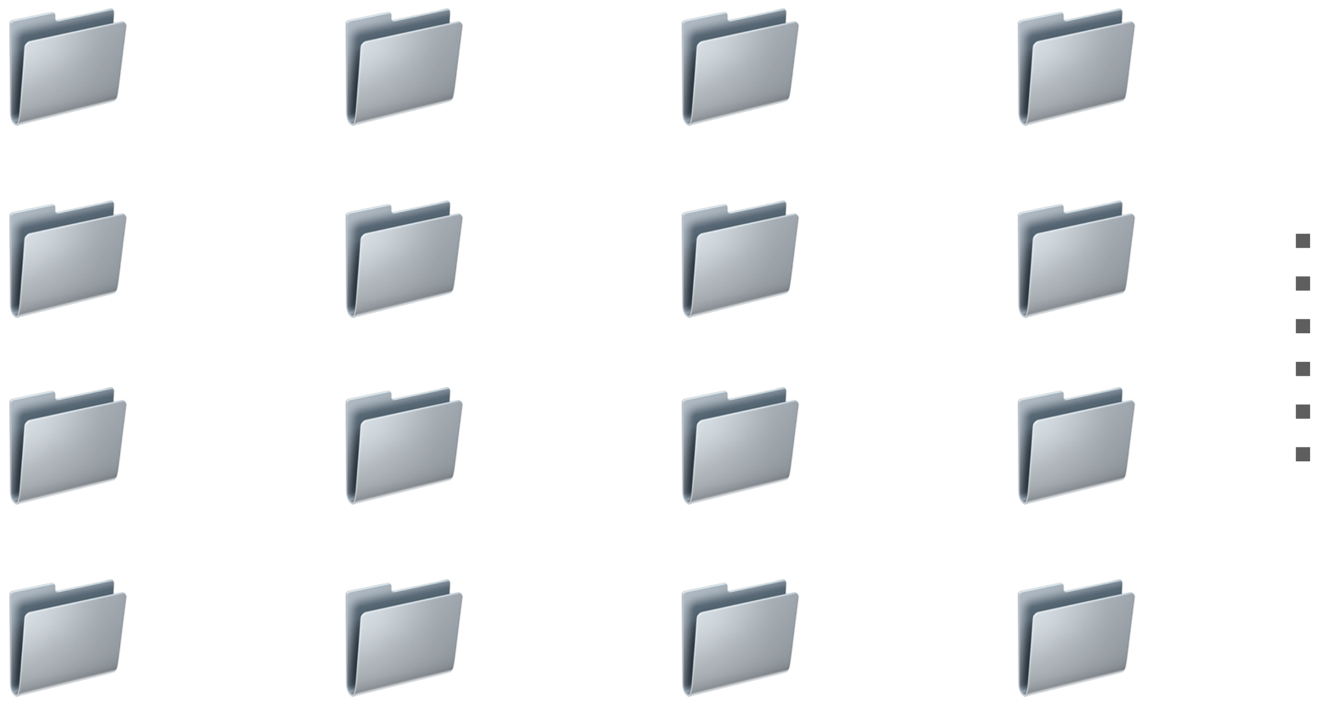


Headers Preview Response Initiator Timing

```
<!doctype html>
<html dir="ltr" lang="en"
    >
  <head>
    <meta charset="utf-8">
    <title>New Tab</title>
    <style>
      body {
        background: #35363A;
        margin: 0;
      }

      #backgroundImage {
        border: none;
        height: 100%;
        pointer-events: none;
        position: fixed;
        top: 0;
        visibility: hidden;
        width: 100%;
      }

      [show-background-image] #backgroundImage {
        visibility: visible;
      }
    </style>
  </head>
  <body>
    <iframe id="backgroundImage" src=""></iframe>
    <ntp-app></ntp-app>
    <script type="module" src="new_tab_page.js"></script>
    <link rel="stylesheet" href="chrome://resources/css/text_defaults_md.css">
    <link rel="stylesheet" href="chrome://theme/colors.css?sets=ui,chrome">
    <link rel="stylesheet" href="shared_vars.css">
  </body>
</html>
```



Traditional Relational Model

- Requires a sequence of MapReduce jobs
- BUT is usually prohibitive at web scale

Analyze outputs of MR pipelines
Rapidly prototype larger computations

Dremel

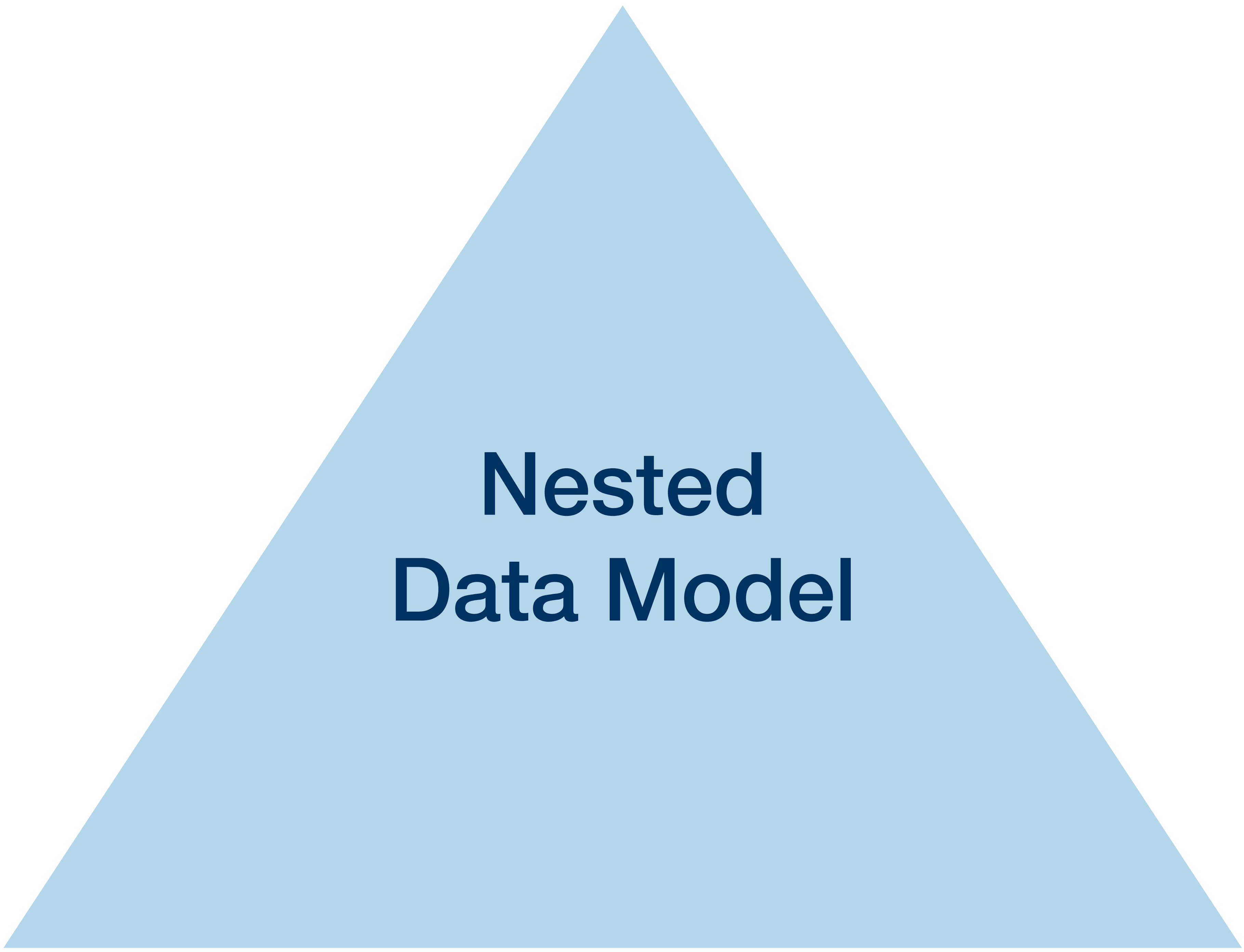
- Capable of operating on *in situ* nested data
- *In situ* refers to the ability to access data "in place"

Berkeley
UNIVERSITY OF CALIFORNIA

01 Introduction

Solution Novelty

✨ Dremel offers flexibility without sacrificing performance



Berkeley
UNIVERSITY OF CALIFORNIA

02

Data Model

Berkeley
UNIVERSITY OF CALIFORNIA

02 Data Model

Nested Data

Based on strongly-typed nested records

$$\tau = \textbf{dom} \mid \langle A_1 : \tau[*|?], \ldots, A_n : \tau[*|?] \rangle$$

- atomic type: **dom** — integers, floating-point numbers, strings, etc.
- record type: consists of i fields
- name A_i
- repeated fields: $*$
- optional fields: $?$

Schema

- Defines a record type `Document`

group: list

entries holding `DocIds` of other web pages

```
message Document {
  required int64 DocId;
  optional group Links {
    repeated int64 Backward;
    repeated int64 Forward; }
  repeated group Name {
    repeated group Language {
      required string Code;
      optional string Country; }
    optional string Url; }}
```

Sample records

r_1

```
DocId: 10
Links
  Forward: 20
  Forward: 40
  Forward: 60
Name
  Language
    Code: 'en-us'
    Country: 'us'
  Language
    Code: 'en'
  Url: 'http://A'
Name
  Url: 'http://B'
Name
  Language
    Code: 'en-gb'
    Country: 'gb'
```

r_2

```
DocId: 20
Links
  Backward: 10
  Backward: 30
  Forward:  80
Name
  Url: 'http://C'
```

- Dotted notation
- `Name.Language.Code`
- Top-most field name is often omitted

03

Storage Model

Berkeley
UNIVERSITY OF CALIFORNIA

03 Storage Model
Goal

🎯 **Store all values of a given field consecutively**

Challenges

- 🤯 lossless representation of record structure in a columnar format
- 🤔 fast encoding
- 🤧 efficient record assembly

Lossless Representation

Introduce repetition and definition levels

Repetition levels

- Disambiguate repeated occurrences
- *“at what repeated field in the field’s path the value has repeated”*

```
DocId: 10                    r1
Links
  Forward: 20
  Forward: 40
  Forward: 60
Name
  Language
    Code: 'en-us'
    Country: 'us'
  Language
    Code: 'en'
  Url: 'http://A'
Name
  Url: 'http://B'
Name
  Language
    Code: 'en-gb'
    Country: 'gb'
```

- Path: `Name.Language.Code`
- # Repeated fields: 2 (`Name` & `Language`)
- Range of `Code` repetition level: [0, 2]

0: no repeated fields (Code: 'en-us')

2: field `Language` has repeated (Code: 'en')

1: `Name` has repeated most recently (Code: 'en-gb')

Definition levels

- *“how many fields in a path that could be undefined (optional / repeated) are actually present”*

```
DocId: 20                    r2
Links
  Backward: 10
  Backward: 30
  Forward:  80
Name
  Url: 'http://C'
```

- Missing occurrence: `Name.Language.Country`
- Definition level: **1**

```
message Document {
  required int64 DocId;
  optional group Links {
    repeated int64 Backward;
    repeated int64 Forward; }
  repeated group Name {
    repeated group Language {
      required string Code;
      optional string Country; }
    optional string Url; }}
```

Name.Language.Country

value	r	d
us	0	3
NULL	2	2
NULL	1	1
gb	1	3
NULL	0	1

03 Storage Model
Columnar Encoding

🎋 Column-striped representation of the data

Column: store as a set of blocks

compressed field values

DocId

value	r	d
10	0	0
20	0	0

Name.Url

value	r	d
http://A	0	2
http://B	1	2
NULL	1	1
http://C	0	2

Links.Forward

value	r	d
20	0	2
40	1	2
60	1	2
80	0	2

Links.Backward

value	r	d
NULL	0	1
10	0	2
30	1	2

Name.Language.Code

value	r	d
en-us	0	2
en	2	2
NULL	1	1
en-gb	1	2
NULL	0	1

Name.Language.Country

value	r	d
us	0	3
NULL	2	2
NULL	1	1
gb	1	3
NULL	0	1

NULLs: not stored explicitly; determined by the definition levels

any definition level smaller than the number of repeated and optional fields in a field's path denotes a NULL

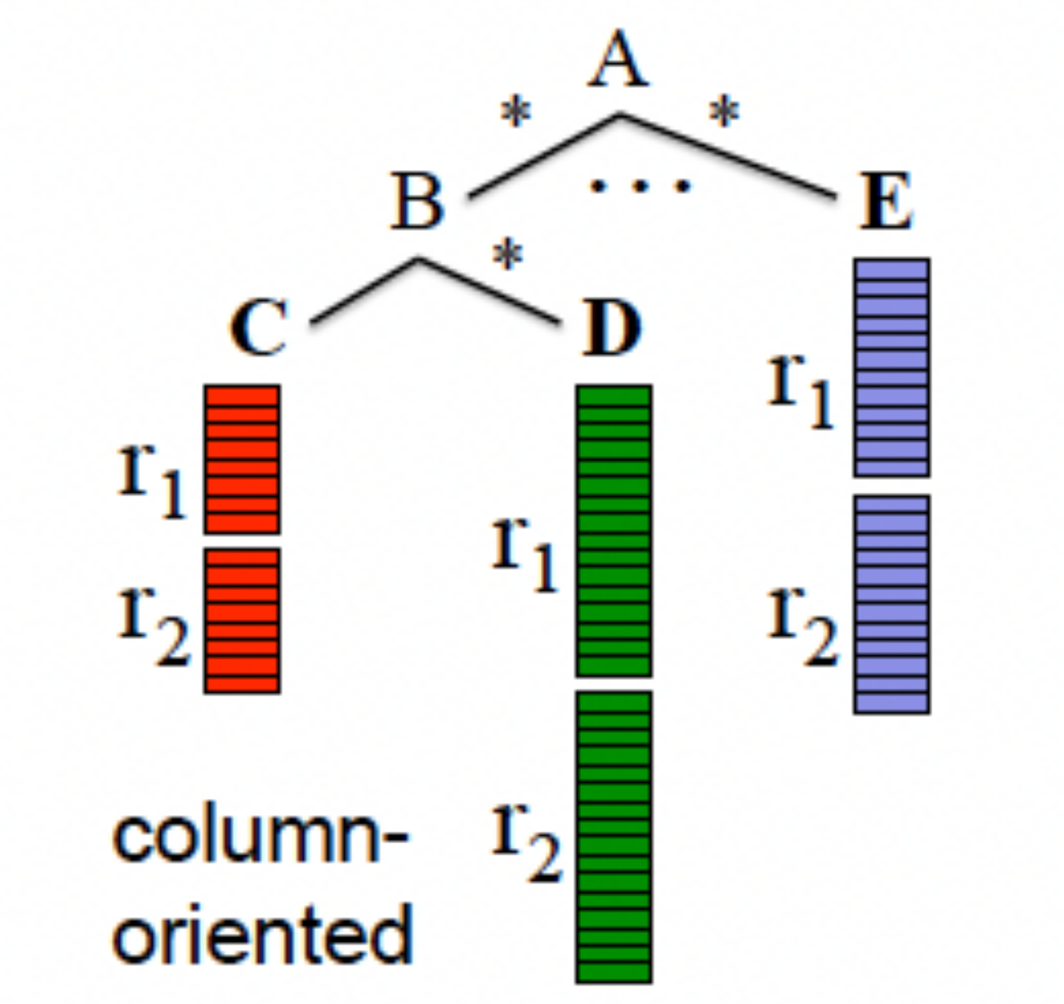
Decompose Record

Produce column stripes efficiently

```
procedure DissectRecord(RecordDecoder decoder,
                        FieldWriter writer, int repetitionLevel):
  Add current repetitionLevel and definition level to writer
  seenFields = {} // empty set of integers
  while decoder has more field values
    FieldWriter chWriter =
      child of writer for field read by decoder
    int chRepetitionLevel = repetitionLevel
    if set seenFields contains field ID of chWriter
      chRepetitionLevel = tree depth of chWriter
    else
      Add field ID of chWriter to seenFields
    end if
    if chWriter corresponds to an atomic field
      Write value of current field read by decoder
      using chWriter at chRepetitionLevel
    else
      DissectRecord(new RecordDecoder for nested record
        read by decoder, chWriter, chRepetitionLevel)
    end if
  end while
end procedure
```

FieldWrite

- a tree whose structure matches the field hierarchy in the schema
- handle missing fields cheaply



- recurses into the record structure
- computes the levels for each field value

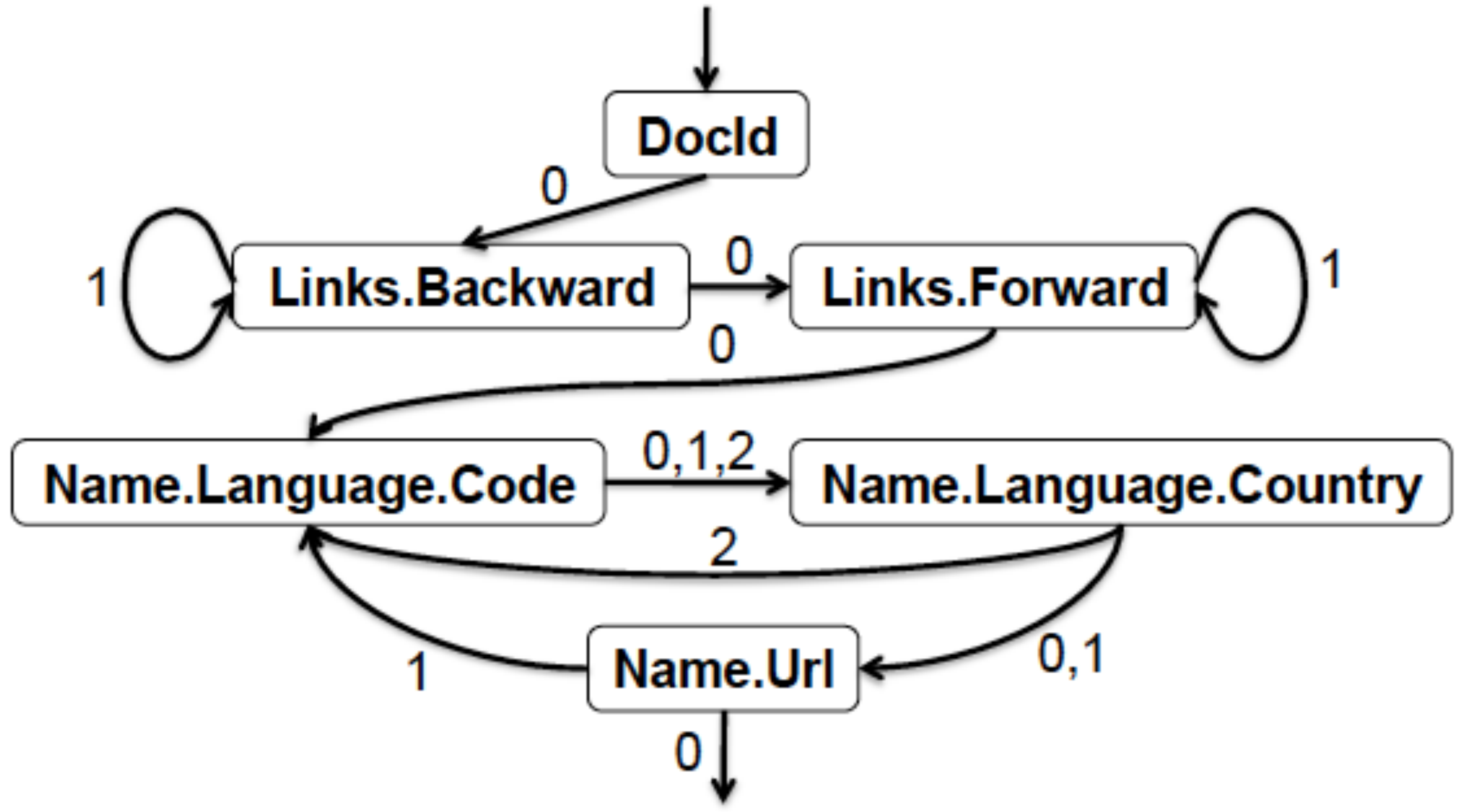
Berkeley
UNIVERSITY OF CALIFORNIA

Reassemble Record

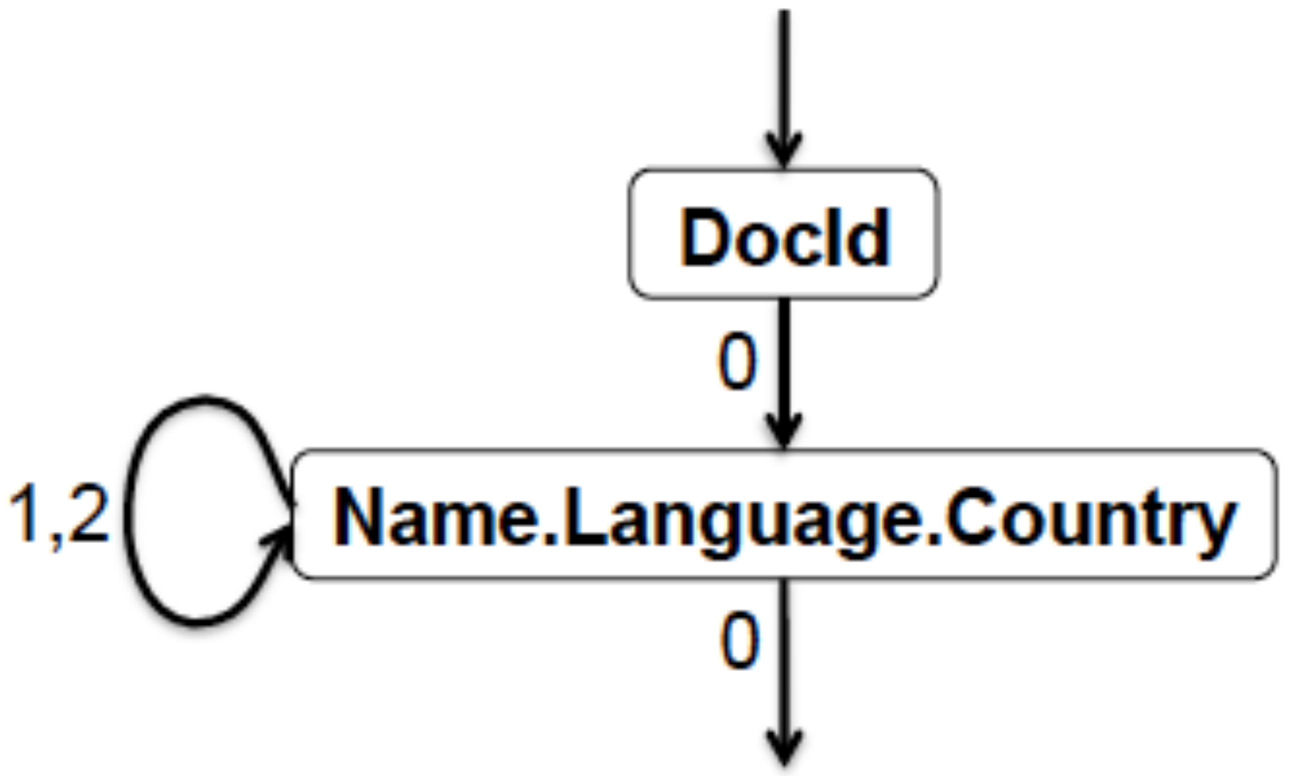
🛠️ FSM assembles records from columnar data

Goal Given a subset of fields, reconstruct the original records as if they contained just the selected fields, with all other fields stripped away

Method Create a finite state machine (FSM) that reads the field values and levels for each field, and appends the values sequentially to the output records



Complete record assembly automaton




```
DocId: 10                 s1
Name
  Language
    Country: 'us'
  Language
Name
  Language
    Country: 'gb'
```

```
DocId: 20                 s2
Name
```

Partial record assembly automaton

04

Query Language

Berkeley
UNIVERSITY OF CALIFORNIA

Sample Query

🙊 SQL-like query implementable on columnar nested data

- Aggregation is done WITHIN each `Name` subrecord
- Emits the number of occurrences of `Name.Language.Code` for each `Name` as a non-negative 64-bit integer

- Each scalar expression in the SELECT clause emits a value at the same level of nesting as the most-repeated input field used in that expression

3️⃣ Within-record Aggregation

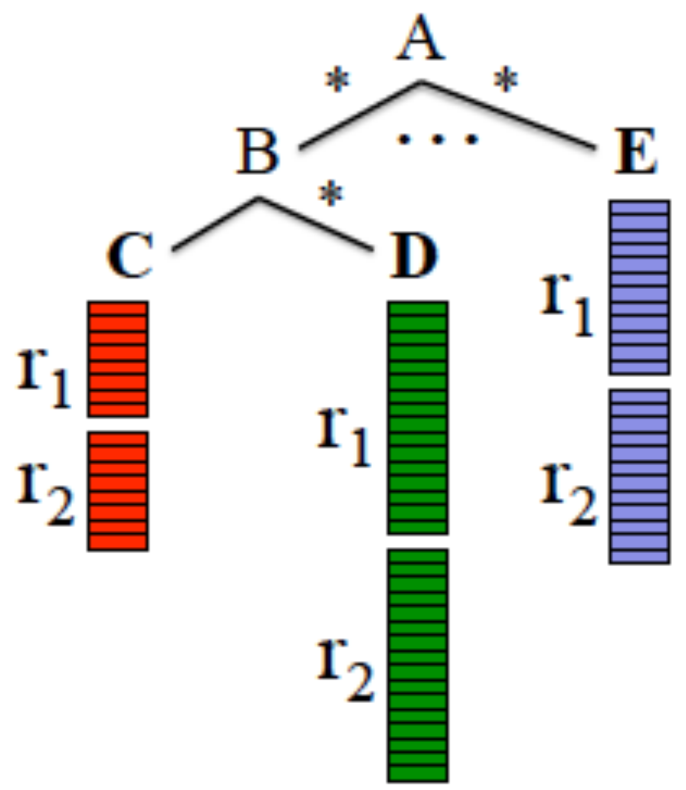
2️⃣ Projection

1️⃣ Selection

```
SELECT DocId AS Id,
  COUNT(Name.Language.Code) WITHIN Name AS Cnt,
  Name.Url + ',' + Name.Language.Code AS Str
FROM t
WHERE REGEXP(Name.Url, '^http') AND DocId < 20;
```

reference field using path expressions

$t = \{r_1, r_2\}$



- Each tree node corresponds to a field name
- Selection operator prunes away the branches of the tree that do not satisfy the specified conditions

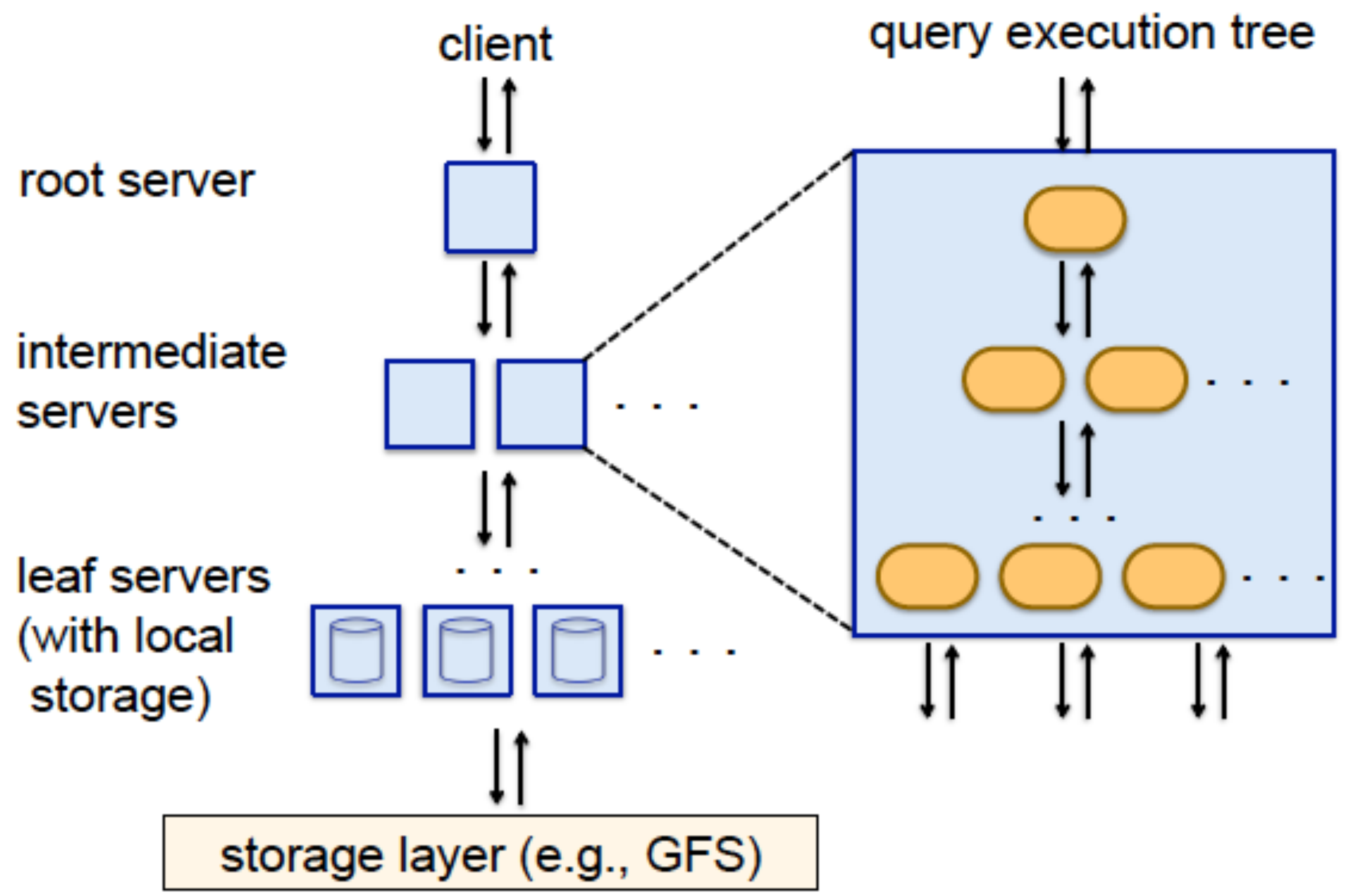
04 Query Language
Query Execution

🌳 Uses a multi-level serving tree to execute queries

Just like a web search request, a query gets pushed down the tree and is rewritten at each step
The result of the query is assembled by aggregating the replies received from lower levels of the tree

- Receives incoming queries
- Reads metadata from tables
- Routes the queries to the next level

- Communicate w/ storage layer
- Access the data on local disk



SELECT A, COUNT(B) FROM T GROUP BY A

- Root server determines all *tablets* (horizontal partitions)
- Rewrites the query

SELECT A, SUM(c) FROM (R_1^1 UNION ALL ... R_n^1) GROUP BY A

Tables $R_1^1, \ldots, R_n^1$ sent to the nodes $1,\ldots,n$ at level 1 of the serving tree

R_i^1 = SELECT A, COUNT(B) AS c FROM T_i^1 GROUP BY A

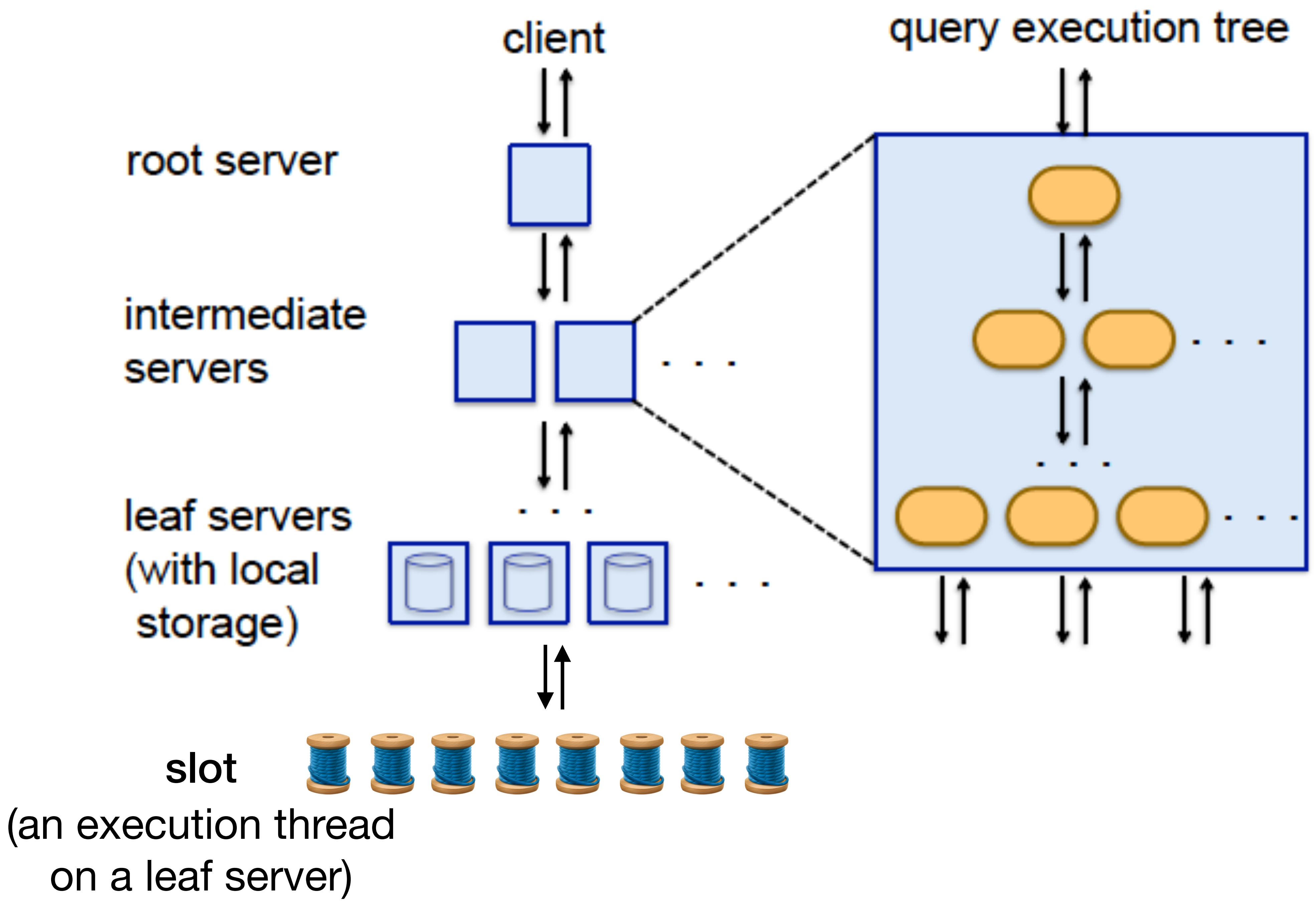
T_i^1 is a disjoint partition of tablets in *T* processed by server *i* at level 1

Reach the leaves, scan the tablets in *T* in parallel

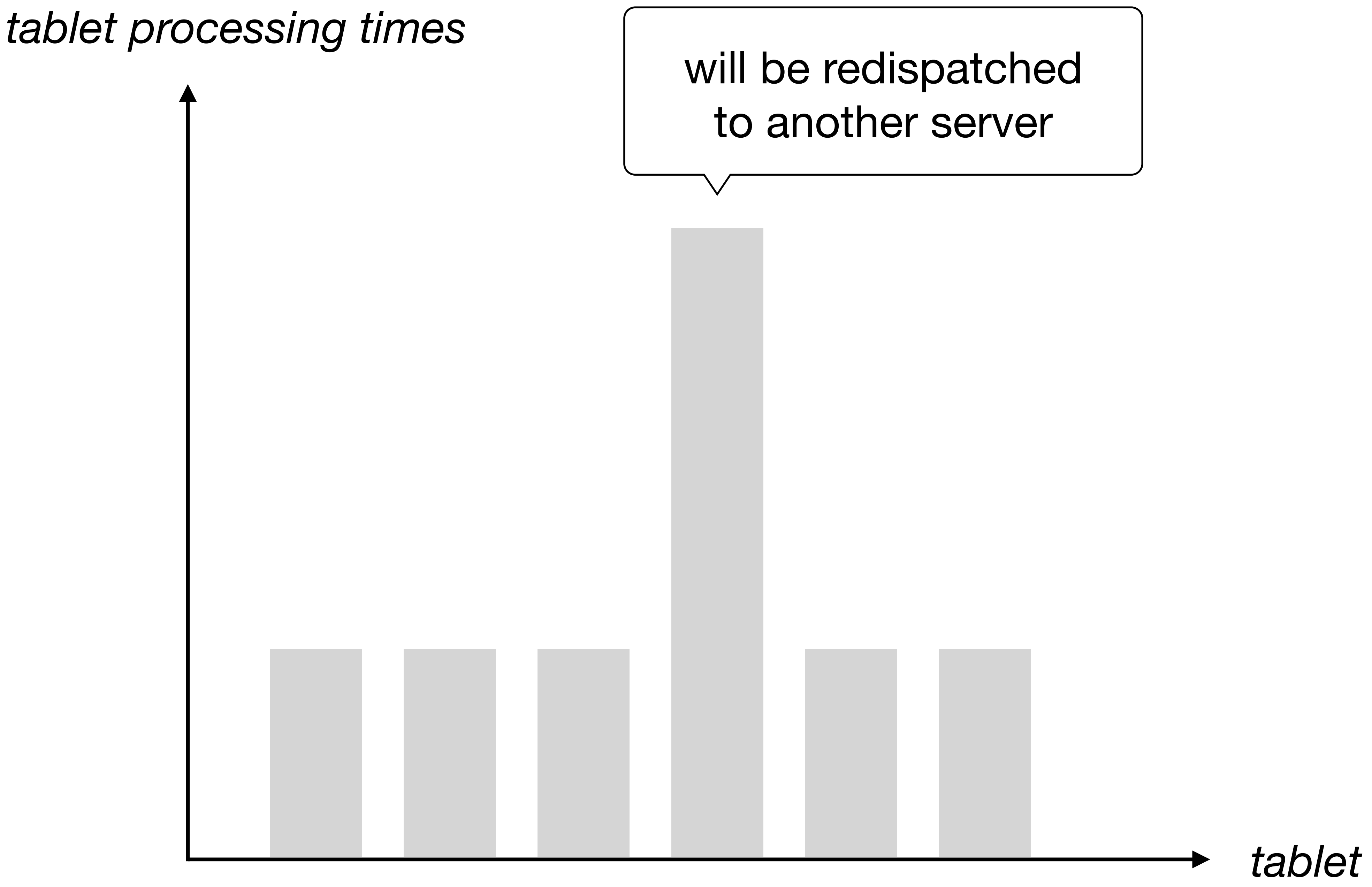
Berkeley UNIVERSITY OF CALIFORNIA

Schedule queries & provide fault tolerance

Processing units: SLOT



Tablet processing times histogram



05

Experiments

Berkeley
UNIVERSITY OF CALIFORNIA

05 Experiments Local Disk

💾 Columnar storage outperforms record-wise storage when few columns are read

Goal

Examine performance tradeoffs of columnar vs. record-oriented storage

Data

Table name	# of records	Size	# of fields	Data center	Replicate factor
T1	85 billion	87 TB	270	A	3×

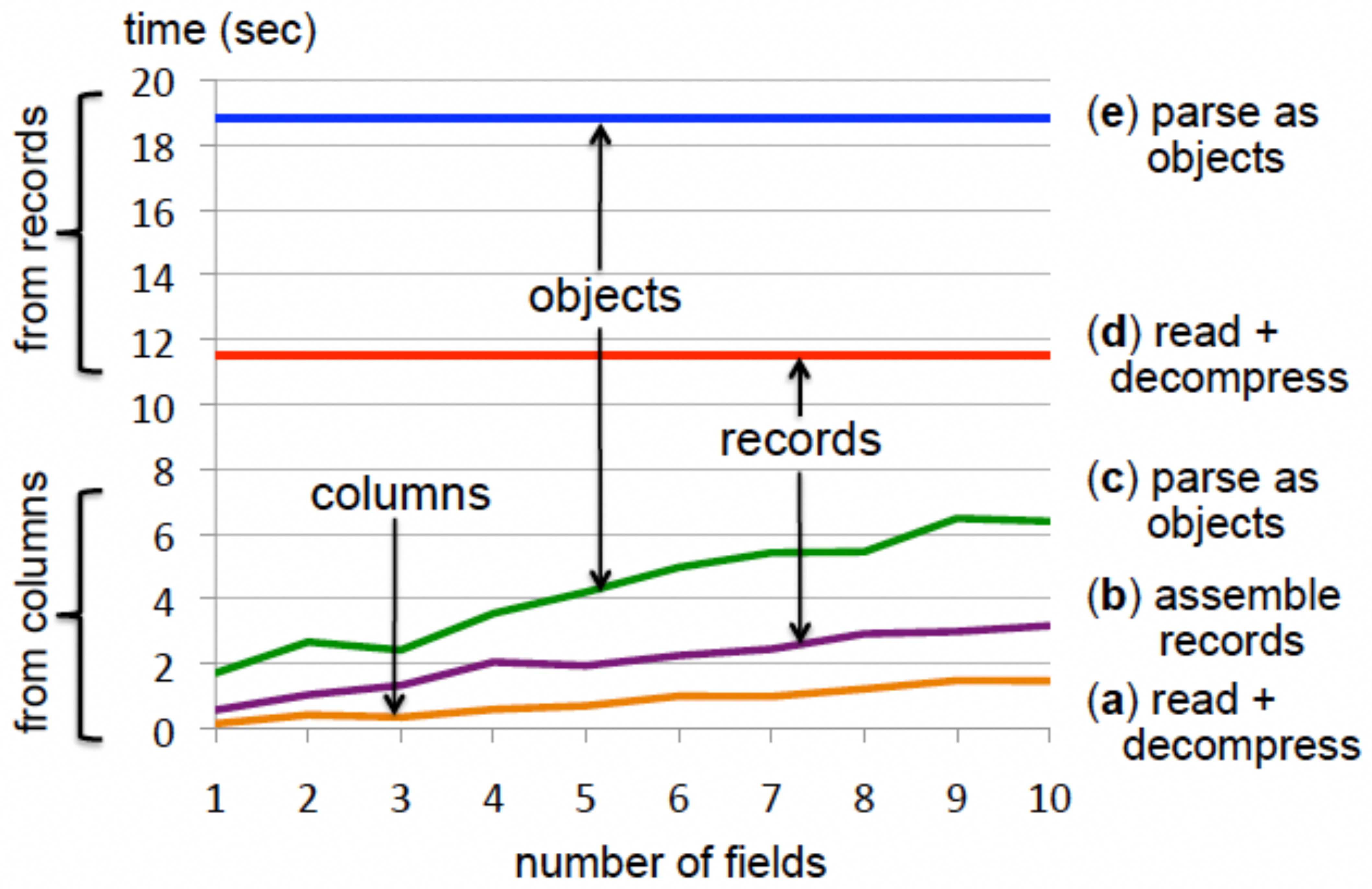
- 1GB fragment of table T1 containing ~300k rows
- Stored on local disk, ~375MB in compressed representation

Task

- Read & uncompress data
- Assemble & parse records

Result

- When few columns are read, the gains of columnar representation are of about an order of magnitude
- Retrieval time for columnar nested data grows linearly with the number of fields
- Record assembly and parsing are expensive, each potentially doubling the execution time



🏃 Execution efficiency: Dremel > MR-col > MR-records

Goal

Illustrate a MR and Dremel execution on columnar vs. record-oriented data

Data

Table name	# of records	Size	# of fields	Data center	Replicate factor
T1	85 billion	87 TB	270	A	3×

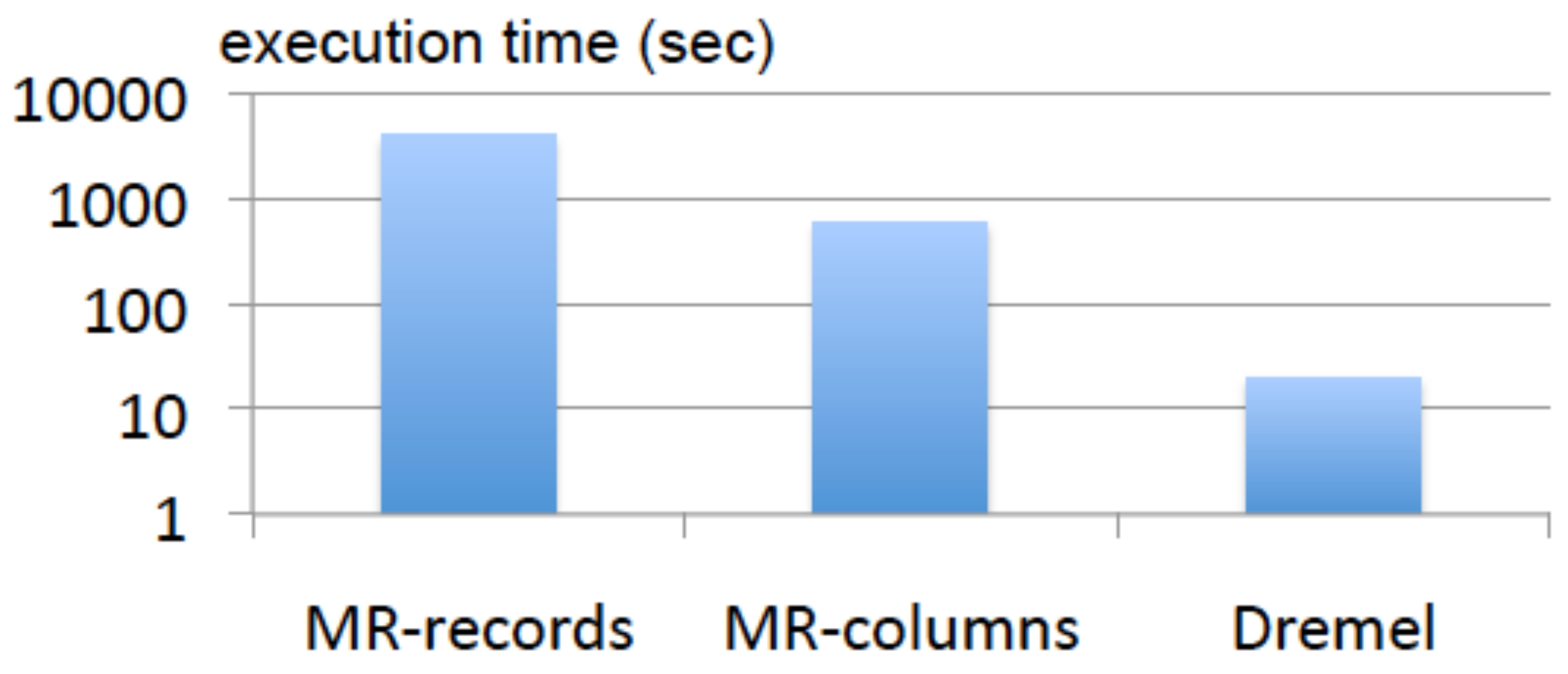
Task

- Count the average number of terms in a field `txtField` in table T1

Q_1: SELECT SUM(CountWords(txtField)) / COUNT(*) FROM T1

Result

	MR-records	MR-columns	Dremel
Workers / nodes	3000	3000	3000
Data read	87 TB	0.5 TB	0.5 TB



05 Experiments

Serving Tree Topology

🌲 Aggregations returning many groups benefit from multi-level serving trees

Goal

Impact of the serving tree depth on query execution times

Data

Table name	# of records	Size	# of fields	Data center	Replicate factor
T2	24 billion	13 TB	530	A	3×

- Each record has a repeated field `item` containing a numeric `amount`

```
repeated group item {
  optional int64 amount; }
```

`item.amount` repeats about 40 billion times in the dataset

Task

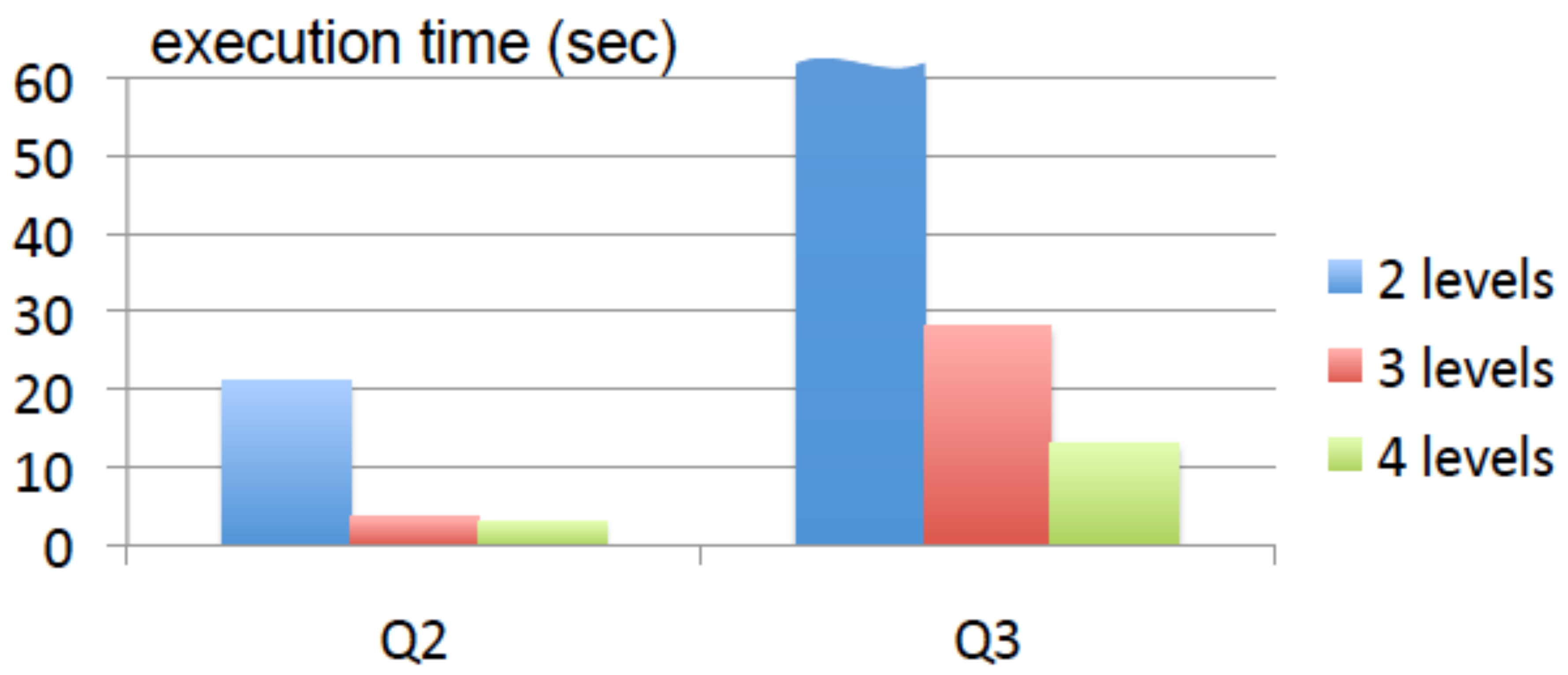
- Sums up the item amount by country
- Returns a few hundred records

Q_2: SELECT country, SUM(item.amount) FROM T2
GROUP BY country

- Performs a GROUP BY on a text field `domain` with a selection condition
- Produces around 1.1 million distinct domains

Q_3: SELECT domain, SUM(item.amount) FROM T2
WHERE domain CONTAINS '.net'
GROUP BY domain

Result



2 levels: a single root server communicates directly with the leaf servers
3 levels: 1 : 100 : 2900
4 levels: 1 : 10 : 100 : 2900

Per-tablet Histograms

📉 99% of Q2/Q3 tablets are processed under 1s/2s

Goal

Drill deeper into what happens during query execution

Data

Table name	# of records	Size	# of fields	Data center	Replicate factor
T2	24 billion	13 TB	530	A	3×

Task

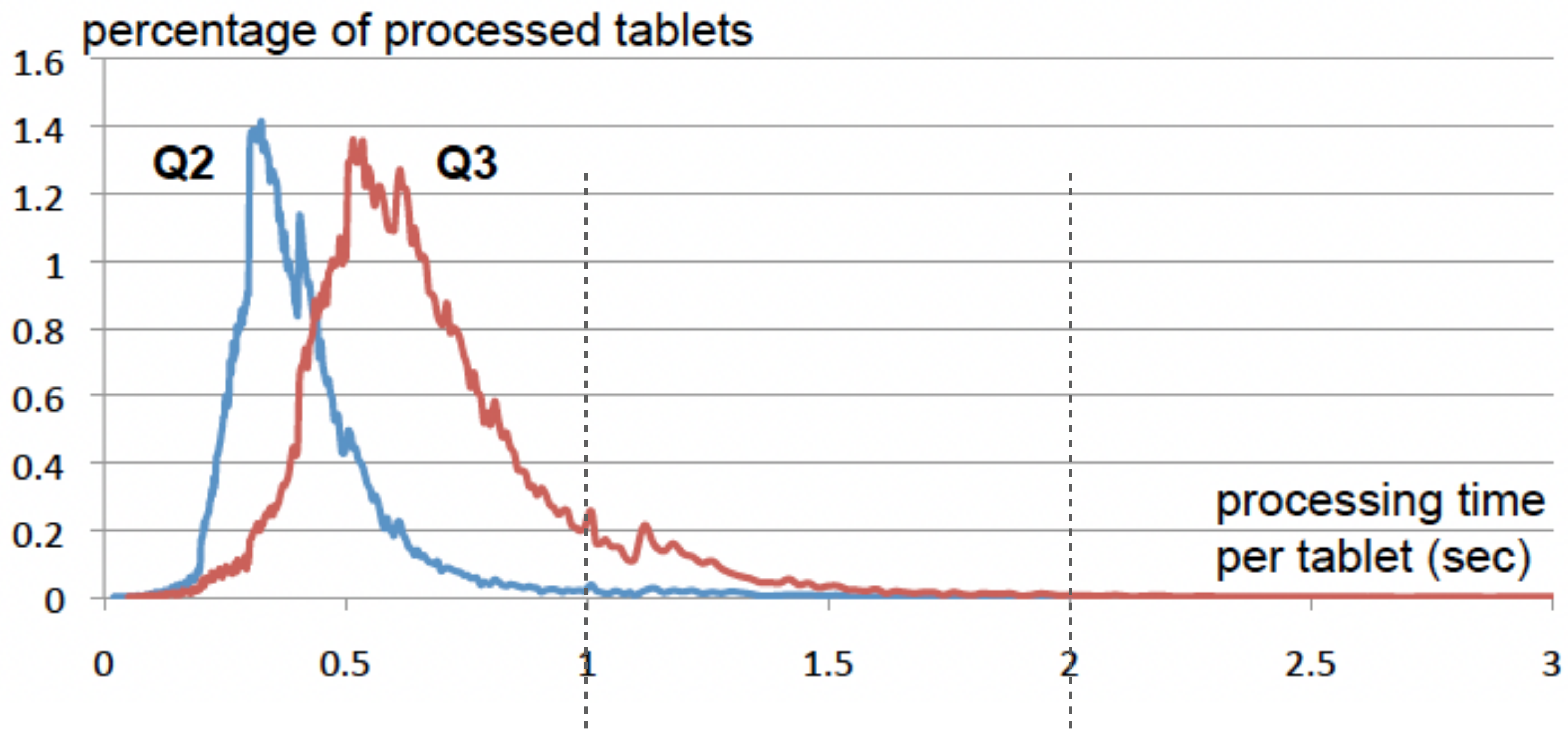
- Sums up the item amount by country
- Returns a few hundred records

Q_2: SELECT country, SUM(item.amount) FROM T2
GROUP BY country

- Performs a GROUP BY on a text field `domain` with a selection condition
- Produces around 1.1 million distinct domains

Q_3: SELECT domain, SUM(item.amount) FROM T2
WHERE domain CONTAINS '.net'
GROUP BY domain

Result



05 Experiments
Within-record Aggregation

✂️ Cheaper processing due to nesting support

Goal

Examine the performance of Query Q4 run on Table T3

Data

Table name	# of records	Size	# of fields	Data center	Replicate factor
T3	4 billion	70 TB	1200	A	3×

Task

- Within-record aggregation: Counts all records where the sum of `a.b.c.d` values occurring in the record are larger than the sum of `a.b.p.q.r` values

Q_4 : SELECT COUNT(c1 $>$ c2) FROM
(SELECT SUM(a.b.c.d) WITHIN RECORD AS c1,
SUM(a.b.p.q.r) WITHIN RECORD AS c2
FROM T3)

Result

- Due to column striping only 13GB (out of 70TB) are read from disk and the query completes in 15 seconds

05 Experiments
Scalability

⬆️ A larger system can be as effective as a smaller one in terms of resource usage, yet allows faster execution

Goal

Illustrate the scalability of the system on a trillion-record table

Data

Table name	# of records	Size	# of fields	Data center	Replicate factor
T4	1+ trillion	105 TB	50	B	3×

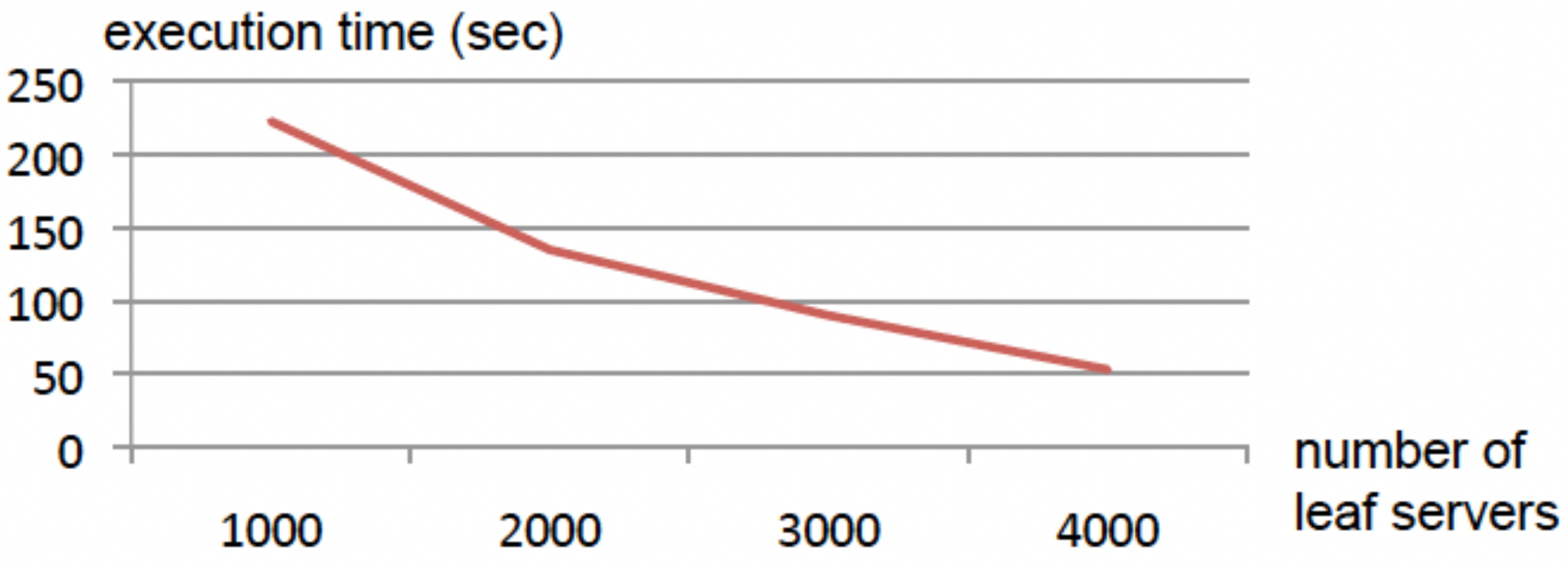
Task

- Select top-20 `aid`'s and their number of occurrences in Table T4

Q_5: SELECT TOP(aid, 20), COUNT(*) FROM T4
WHERE bid = {value1} AND cid = {value2}

Result

- Total expended CPU time is nearly identical at ~300k seconds
- User-perceived time decreases near-linearly with the growing size of the system



Berkeley
UNIVERSITY OF CALIFORNIA

🐢 A small fraction of the tablets take a lot longer

Goal Show the impact of stragglers

Data

Table name	# of records	Size	# of fields	Data center	Replicate factor
T5	1+ trillion	20 TB	30	B	2×

- The likelihood of stragglers slowing the execution is higher since there are fewer opportunities to reschedule the work

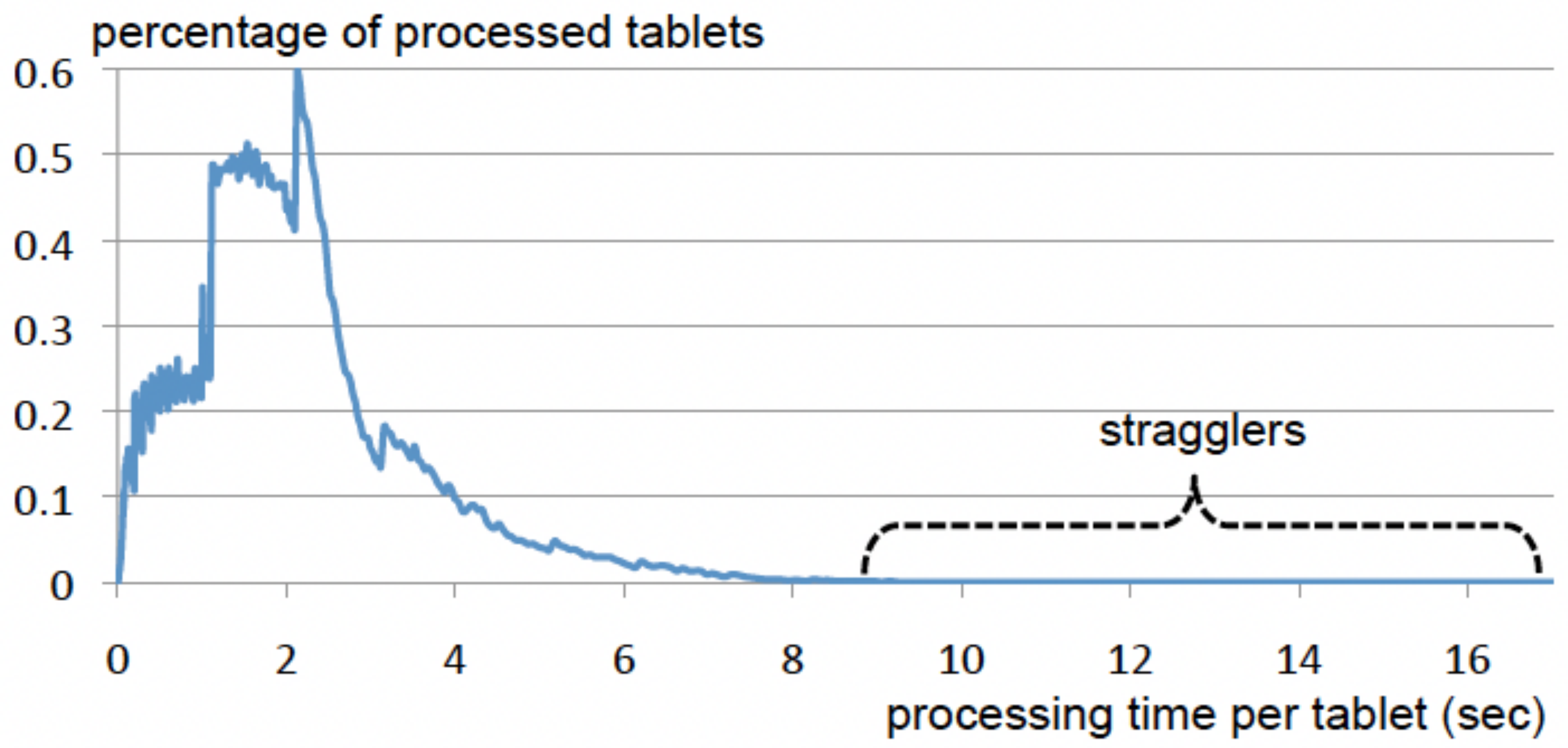
Task

- Read over 1TB of compressed data

Q_6: SELECT COUNT(DISTINCT a) FROM T5

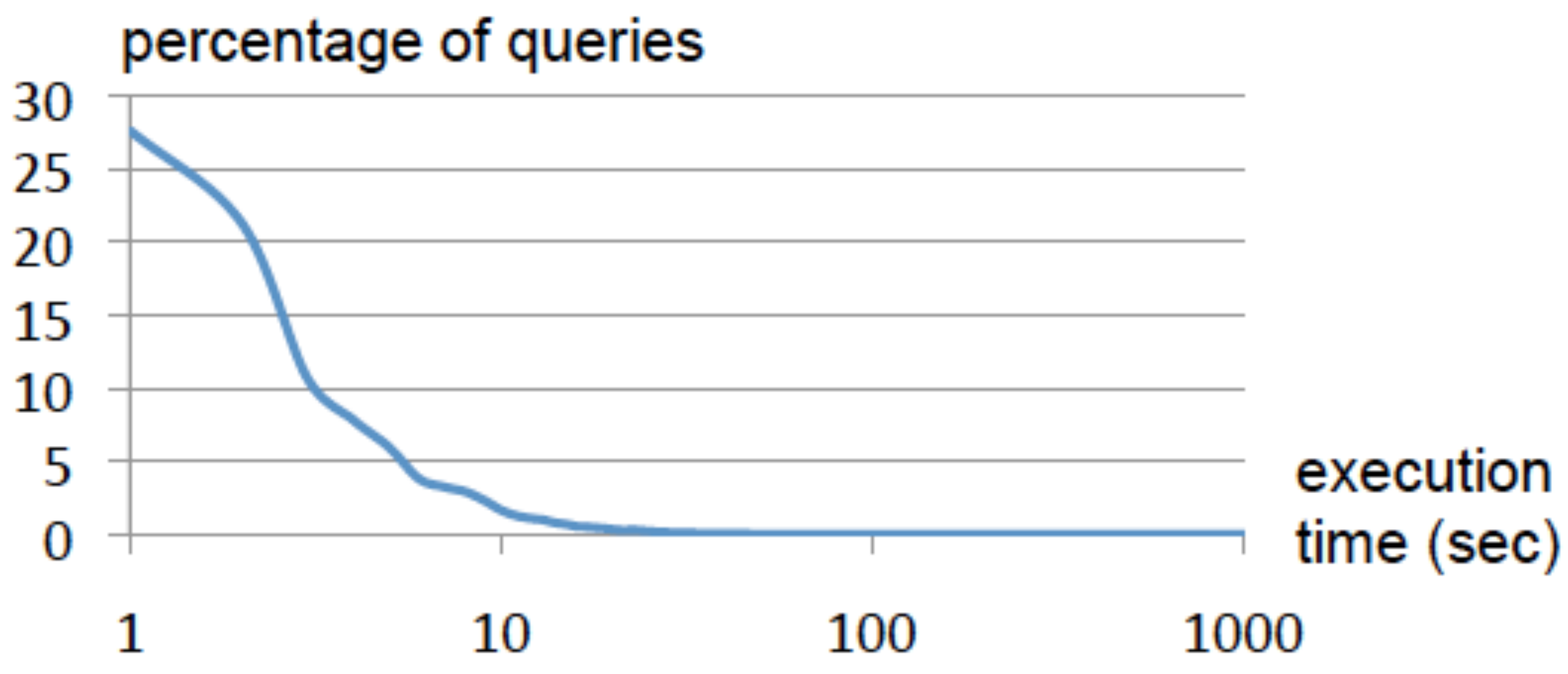
Result

- Processing time for 99% of the tablets is below 5 seconds per tablet per slot



🚀 The bulk of a web-scale dataset can be scanned fast

Most queries are processed <10 seconds, well within the interactive range



Thank You!

Berkeley
UNIVERSITY OF CALIFORNIA