Succinct

Enabling Queries on Compressed Data
Distributed data stores

Fundamental building blocks for many cloud services

Amazon DynamoDB

MongoDB

cassandra

elasticsearch

memCached

redis

MICA

RAMCloud

MemC3

VERTICA

SAP HANA
**Records, primary key, secondary keys**

Data stored as collection of records

<table>
<thead>
<tr>
<th>USER</th>
<th>POSITION</th>
<th>BOSS</th>
<th>BOSS’ Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anurag</td>
<td>Student</td>
<td>Rachit</td>
<td>Serious, Pushy</td>
</tr>
<tr>
<td>.....</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rachit</td>
<td>Postdoc</td>
<td>Ion</td>
<td>Curious, funny</td>
</tr>
</tbody>
</table>
# Records, primary key, secondary keys

Data stored as collection of records

## Primary key

<table>
<thead>
<tr>
<th>USER</th>
<th>POSITION</th>
<th>BOSS</th>
<th>BOSS’ Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anurag</td>
<td>Student</td>
<td>Rachit</td>
<td>Serious, Pushy</td>
</tr>
<tr>
<td>.....</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rachit</td>
<td>Postdoc</td>
<td>Ion</td>
<td>Curious, funny</td>
</tr>
</tbody>
</table>
Records, primary key, secondary keys

Data stored as collection of records

<table>
<thead>
<tr>
<th>USER</th>
<th>POSITION</th>
<th>BOSS</th>
<th>BOSS’ Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anurag</td>
<td>Student</td>
<td>Rachit</td>
<td>Serious, Pushy</td>
</tr>
<tr>
<td>......</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Rachit</td>
<td>Postdoc</td>
<td>Ion</td>
<td>Curious, funny</td>
</tr>
</tbody>
</table>
Queries in data stores

Queries on primary keys

• GET, PUT, DELETE
• Very efficient [MICA, Redis, RAMCloud]
Queries in data stores

Queries on primary keys
  • GET, PUT, DELETE
  • Very efficient [MICA, Redis, RAMCloud]

Queries on secondary keys
  • E.g., search
  • Executed via
    • Data scans [columnar stores]
    • Indexes [Cassandra, MongoDB]
  • Either slow or large storage overhead
For now, focus on:

- Queries on flat (unstructured) files
- Very powerful primitive
  - Allows queries on semi-structured data
  - Discussed later
Example: Search( , file)

Search( )
Example: Search(  , file)

Search(  )

Data Scans
Example: Search( green, file)
Example: `Search( , file)`

Data Scans

Search( )

Low storage

High Latency
Example: Search( , file)

- **Data Scans**
- **Indexes**

- Low storage
- High Latency
Example: `Search( , file)`

Data Scans

Indexes

Search( )

Low storage  High Latency

0, 10, 14, 16, 19, 26, 29
1, 4, 5, 8, 20, 22, 24
2, 15, 17, 27
3, 6, 7, 9, 12, 13, 18, 23 ..
11, 21
Example: Search(  , file)

Data Scans

Indexes

Low storage

High Latency
Example: Search(■, file)

Data Scans

Indexes

Search(■)

Low storage
High Latency

High storage
Low Latency

0, 10, 14, 16, 19, 26, 29
1, 4, 5, 8, 20, 22, 24
2, 15, 17, 27
3, 6, 7, 9, 12, 13, 18, 23...
11, 21
Latency with increasing input size

Query Latency versus Input size.
Latency with increasing input size

Query Latency

Input size

Data scans

Scans in faster storage
Latency with increasing input size

- Data scans
- Scans in slower storage
Latency with increasing input size

Query Latency

Input size

Data scans

Indexes
Latency with increasing input size

- **Query Latency**
  - **Indexes**
    - Indexes in faster storage
  - Data scans

- **Input size**
Indexes in slower storage

Latency with increasing input size

Query Latency

Input size

Indexes

Data scans
Latency with increasing input size

Query Latency

Input size

Data scans

Indexes

executing queries off slower storage
Push **more data in faster storage**

- $\text{Storage} < \text{Input data size}$
Succinct

Push more data in faster storage
  • Storage < Input data size

Execute queries directly on compressed data
  • No data scans
  • No data decompression
Push more data in faster storage
  • Storage < Input data size

Execute queries directly on compressed data
  • No data scans
  • No data decompression

Target: read-heavy append-only workloads
Succinct

Push more data in faster storage
  • Storage < Input data size

Execute queries directly on compressed data
  • No data scans
  • No data decompression

Target: read-heavy append-only workloads

Non-goals: Fault tolerance, data consistency
Example: Search(  , file)

Succinct
Example: Search( , file)
Example: Search( , file)

Succinct

Queries executed directly on the compressed representation
Example: Search( , file)

Succinct

Queries executed directly on the compressed representation

Low storage
Low Latency
Example: Search( , file)

Succinct

Queries executed directly on the compressed representation

Low storage
Low Latency

What makes Succinct unique
Example: Search( , file)

**Succinct**

- Low storage
- Low Latency
- Queries executed directly on the compressed representation

**What makes Succinct unique**

| No additional indexes | Query responses embedded within the compressed representation |
Example: Search(\text{\textcolor{green}{[\ ]}}, \text{file})

<table>
<thead>
<tr>
<th>Succinct</th>
<th>What makes Succinct unique</th>
</tr>
</thead>
<tbody>
<tr>
<td>Queries executed directly on the compressed representation</td>
<td>No additional indexes</td>
</tr>
<tr>
<td>Low storage</td>
<td>Query responses embedded within the compressed representation</td>
</tr>
<tr>
<td>Low Latency</td>
<td>No data scans</td>
</tr>
<tr>
<td></td>
<td>Functionality of indexes</td>
</tr>
</tbody>
</table>
Example: \texttt{Search(\quad\text{file})}

**Succinct**

- Queries executed directly on the compressed representation
- Low storage
- Low Latency

**What makes Succinct unique**

<table>
<thead>
<tr>
<th>No additional indexes</th>
<th>Query responses embedded within the compressed representation</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>No data scans</th>
<th>Functionality of indexes</th>
</tr>
</thead>
</table>

| No decompression | Queries directly on the compressed representation (except for data access queries) |
Search latency, qualitatively

Query Latency

Input size

Data scans

Indexes

Succinct
Search latency, qualitatively

Query Latency

Input size

Data scans

Avoiding data scans

Indexes

Succinct
Search latency, qualitatively

Query
Latency

Input size

Indexes

Succinct

Data scans

Avoiding queries off slower storage
What do we lose?

Succinct

- No secondary indexes
- No data scans, no data decompression
What do we lose?

Succinct

- No secondary indexes
- No data scans, no data decompression

What do we lose?
What do we lose?

**Succinct**

- No secondary indexes
- No data scans, no data decompression

**What do we lose?**

- Preprocessing expensive (4GB/hour/core)
- CPU (for data access)
- Sequential scan throughput (13Mbps/thread)
- In-place updates
Succinct Data Representation

Builds upon a large body of theory results
Builds upon a large body of theory results

- FM-index
  - BWT-based technique
- Compressed Suffix arrays
  - Suffix array based technique
Succinct Data Representation

Builds upon a large body of theory results

- FM-index
  - BWT-based technique
- Compressed Suffix arrays
  - Suffix array based technique

Succinct

- Builds upon latter due to simplicity
- Improved data structures (1.25–3x more space efficient)
- New query algorithm (2.3x faster on average)
Array of suffixes (AoS)

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>a</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
<td>y</td>
</tr>
</tbody>
</table>


**Step 1:**

**Construct set of suffixes**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>a</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
<td>y</td>
</tr>
</tbody>
</table>
Array of suffixes (AoS)

Step 1:
Construct set of suffixes

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>a</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
<td>y</td>
</tr>
</tbody>
</table>
Array of suffixes (AoS)

Step 1:
Construct set of suffixes

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>a</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
<td>y</td>
</tr>
<tr>
<td>h</td>
<td>a</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
<td>y</td>
</tr>
</tbody>
</table>
Array of suffixes (AoS)

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>a</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
<td>y</td>
</tr>
<tr>
<td>h</td>
<td>a</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
<td>y</td>
</tr>
</tbody>
</table>

Step 1: Construct set of suffixes
Array of suffixes (AoS)

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>a</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
<td>y</td>
</tr>
<tr>
<td>h</td>
<td>a</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
<td>y</td>
</tr>
<tr>
<td>a</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td></td>
</tr>
</tbody>
</table>

**Step 1:**
Construct set of suffixes
Step 1: Construct set of suffixes

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>a</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
<td>y</td>
</tr>
<tr>
<td>h</td>
<td>a</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
<td>y</td>
</tr>
<tr>
<td>a</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>p</td>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td></td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>u</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>p</td>
<td>y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Array of suffixes (AoS)

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>a</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
<td>y</td>
</tr>
<tr>
<td>h</td>
<td>a</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
<td>y</td>
</tr>
<tr>
<td>h</td>
<td>a</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
<td>y</td>
</tr>
<tr>
<td>a</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>u</td>
<td>p</td>
<td>y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>p</td>
<td>y</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Step 2: Sort suffixes (lexicographically)
Array of suffixes (AoS)
Step 3: Mark location of each sorted suffix (in input file)

Suffix array (AoS2Input)

0 1 2 3 4 5 6 7 8 9
happy puppy

happy puppy
appy puppy
ppy puppy
ppy puppy
ppy puppy
ppy puppy
ppy puppy
ppy puppy
ppy puppy
ppy puppy
ppy puppy
ppy puppy
ppy puppy
ppy puppy
ppy puppy
ppy puppy
ppy puppy
ppy puppy
ppy puppy
ppy puppy
**Step 3:**
Mark location of each sorted suffix (in input file)

### Suffix array (AoS2Input)

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>a</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
<td>y</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>a</th>
<th>p</th>
<th>p</th>
<th>y</th>
<th>p</th>
<th>u</th>
<th>p</th>
<th>p</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>a</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>p</th>
<th>y</th>
<th>p</th>
<th>u</th>
<th>p</th>
<th>p</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
<td>y</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>p</th>
<th>y</th>
<th>p</th>
<th>u</th>
<th>p</th>
<th>p</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>p</td>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
<td>y</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>y</th>
<th>p</th>
<th>u</th>
<th>p</th>
<th>p</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
<td>y</td>
</tr>
</tbody>
</table>

| y | p | u | p | p | y |
Step 3: Mark location of each sorted suffix (in input file)

Suffix array (AoS2Input)

0 1 2 3 4 5 6 7 8 9
h a p p y p u p p y

1 2 3 4 5 6 7 8 9 10
a p p y p u p p y y
h a p p y p u p p y
p p y
p p y p u p p y
p y
p y p u p p y
u p p y
y p u p p y
y p u p p y
Step 3: Mark location of each sorted suffix (in input file)

Suffix array (AoS2Input)

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>a</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
<td>p</td>
</tr>
</tbody>
</table>

1. a p p y p u p p y
   h a p p y p u p p y
   p p y
   p p y
   p u p p y
   p y
   p y
   p y
   u p p y
   y
   y p u p p y
### Suffix array (AoS2Input)

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>a</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
<td>y</td>
</tr>
</tbody>
</table>

**Step 3:**
Mark location of each sorted suffix (in input file)

1.  a p p y p u p p y y
2.  h a p p y p u p p y
3.  p p y
4.  p p y p u p p y
5.  p u p p y
6.  p y
7.  p y p u p p y
8.  u p p y
9.  y p u p p y
10. y p u p p y
Suffix array (AoS2Input)

AoS2Input
Succinct Data Representation

0 1 2 3 4 5 6 7 8 9
happy puppy

happy puppy
Succinct Data Representation

Array of Suffixes (AoS)
(suffixes in sorted order)
Succinct Data Representation

Array of Suffixes (AoS)
(suffixes in sorted order)

AoS2Input
(location of suffix in file)
Succinct Data Representation

Arbitrary substring search via binary search
Problem 1: Size of AoS

\[ O(n^2) \text{ bits} \]
Succinct Data Representation

Problem 1: Size of AoS
$O(n^2)$ bits

Problem 2: Size of AoS2 Input
Each entry takes $\log(n)$ bits
Succinct Data Representation

0 1 2 3 4 5 6 7 8 9
happy puppy puppy puppy puppy puppy puppy puppy puppy puppy

1 0 7 2 5 8 3 6 9 4
Succinct Data Representation

0 1 2 3 4 5 6 7 8 9

happy puppy puppy
### Succinct Data Representation

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>a</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
<td>y</td>
</tr>
</tbody>
</table>

---

The diagram illustrates a succinct data representation, showing the characters 'happiness' with corresponding binary indices.
Succinct Data Representation
### Succinct Data Representation

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>a</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
<td>y</td>
</tr>
</tbody>
</table>

![Diagram of succinct data representation]

- 0
- 1
- 7
- 2
- 5
- 8
- 3
- 6
- 9
- 4

- 3
- 0
- 6
- 9
Succinct Data Representation

h a p p y p u p p y

1 0 7 2 5 8 3 6 9 4

3 0 6 9 4
Succinct Data Representation

h a p p y p u p p y

1 0 7 2 5 8 3 6 9 4

3 0 6 7 9 2 4
Succinct Data Representation

0 1 2 3 4 5 6 7 8 9
h a p p y p u p p y

1 0 7 2 5 8 3 6 9 4

3 0 5 6 7 9 2 4
Succinct Data Representation

0 1 2 3 4 5 6 7 8 9
happy puppy puppy

1 0 7 2 5 8 3 6 9 4

3 0 5 6 7 8 9 2 4
Succinct Data Representation

0 1 2 3 4 5 6 7 8 9
happy puppy puppy

1 0 7 2 5 8 3 6 9 4

3 0 5 6 7 8 9 2 1 4
Succinct Data Representation

0 1 2 3 4 5 6 7 8 9
happy puppy

1 0 7 2 5 8 3 6 9 4

NextCharIdx
(index of suffix after removing the first character)
**Succinct Data Representation**

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>a</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
<td>y</td>
</tr>
</tbody>
</table>

| 1 | 0 | 7 | 2 | 5 | 8 | 3 | 6 | 9 | 4 |

NextCharIdx
(index of suffix after removing the first character)

| 3 | 0 | 5 | 6 | 7 | 8 | 9 | 2 | 1 | 4 |
Succinct Data Representation

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>a</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
<td>y</td>
</tr>
</tbody>
</table>

NextCharIdx
(index of suffix after removing the first character)
Succinct Data Representation

<table>
<thead>
<tr>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
</tr>
</thead>
<tbody>
<tr>
<td>h</td>
<td>a</td>
<td>p</td>
<td>p</td>
<td>y</td>
<td>p</td>
<td>u</td>
<td>p</td>
<td>p</td>
<td>y</td>
</tr>
</tbody>
</table>

No redundancy in AoS2Input

NextCharIdx
(index of suffix after removing the first character)
Succinct Data Representation

NextCharIdx
(index of suffix after removing the first character)

NextCharIdx
(index of next larger value in AoS2Input)
Succinct Data Representation

NextCharIdx
(index of suffix after removing the first character)

NextCharIdx
(index of next larger value in AoS2Input)
Succinct Data Representation

NextCharIdx
(index of suffix after removing the first character)

NextCharIdx
(index of next larger value in AoS2Input)
Succinct Data Representation

<p>| | | | | | | | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>1</td>
<td>2</td>
<td>3</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>7</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>hap</td>
<td>ppy</td>
<td>puppy</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

No redundancy in AoS2Input
Succinct Data Representation

Solution: Sample!
Succinct Data Representation

0 1 2 3 4 5 6 7 8 9
happy puppy

1 7 5 3 9

1 0 7 2 5 8 3 6 9 4
Succinct Data Representation

Problem: How to find unsampled values?
Problem: How to find unsampled values?

Solution: Follow NextCharIdx pointers until you hit a sampled value.
Succinct Data Representation

Problem: How to find unsampled values?

Solution: Follow NextCharIdx pointers until you hit a sampled value.
Succinct Data Representation

Problem: How to find unsampled values?

Solution: Follow NextCharIdx pointers until you hit a sampled value.
Succinct Data Representation

Problem: How to find unsampled values?

Solution: Follow NextCharIdx pointers until you hit a sampled value.
Succinct Data Representation

Increasing Integer Sequences
Succinct Data Representation

Increasing Integer Sequences

Can be compressed!
Data Model and API
Data Model and API

Input: flat (unstructured) files
Data Model and API

Input: flat (unstructured) files
Data Model and API

Input: flat (unstructured) files
Data Model and API

Input: flat (unstructured) files

Search: returns offsets of arbitrary strings in uncompressed file
Data Model and API

Input: flat (unstructured) files

Extract: returns data at arbitrary offsets in uncompressed file
Input: flat (unstructured) files

Data Model and API

Count: returns count of arbitrary strings in uncompressed file

Original Input

Succinct

Search( ) = \{0, 10, 14, 16, 19, 26, 29\}

Extract(0, 5) = \{ , , , , \}

Count( ) = 7
Data Model and API

Input: flat (unstructured) files

Original Input

Succinct

Search( ) = \{0, 10, 14, 16, 19, 26, 29\}
Extract(0, 5) = \{   ,   ,   ,   ,   \}
Count( ) = 7
Append( , , , , )
Input: flat (unstructured) files

Data Model and API

Search( ) = \{0, 10, 14, 16, 19, 26, 29\}

Extract(0, 5) = \{ , , , , \}

Count( ) = 7

Append( , , , , )

Range and Wildcard queries
Power of queries on flat files

Why flat unstructured files?
Why flat unstructured files?

- Many powerful abstractions on top
  - Key-value store [Dynamo, MICA]
  - Tables [Cassandra, BigTable]
  - Documents [MongoDB]
Why flat unstructured files?

- Many powerful abstractions on top
  - Key-value store [Dynamo, MICA]
  - Tables [Cassandra, BigTable]
  - Documents [MongoDB]
Why flat unstructured files?

- Many powerful abstractions on top
  - Key-value store [Dynamo, MICA]
  - Tables [Cassandra, BigTable]
  - Documents [MongoDB]
Why flat unstructured files?

- Many powerful abstractions on top
  - Key-value store [Dynamo, MICA]
  - Tables [Cassandra, BigTable]
  - Documents [MongoDB]
Why flat unstructured files?

- Many powerful abstractions on top
  - Key-value store [Dynamo, MICA]
  - Tables [Cassandra, BigTable]
  - Documents [MongoDB]
Why flat unstructured files?

- Many powerful abstractions on top
  - Key-value store [Dynamo, MICA]
  - Tables [Cassandra, BigTable]
  - Documents [MongoDB]
Handling Appends

Fine-grained updates of compressed data challenging

- Succinct does not support in-place updates
- Uses a multi-store architecture for appends
- Similar to SILT [SOSP’11]
Handling Appends

Fine-grained updates of compressed data challenging

- Succinct does not support in-place updates
- Uses a multi-store architecture for appends
- Similar to SILT [SOSP’11]

Challenge
Handling Appends

Fine-grained updates of compressed data challenging

- Succinct does not support in-place updates
- Uses a multi-store architecture for appends
- Similar to SILT [SOSP’11]

Challenge

- Multi-store architectures not new
- Challenge: supporting queries efficiently
- Details in paper
End-to-end System
End-to-end System

LogStore
End-to-end System

SuffixStore

LogStore
End-to-end System

SuccinctStores

SuffixStore

LogStore
End-to-end System

Coordinator

SuccinctStores  SuffixStore  LogStore
End-to-end System

Cluster Management

- Maintains lists of active servers
- Periodic heartbeats

Coordinator

SuccinctStores

SuffixStore

LogStore
End-to-end System

Coordinator

Data Management

- Updated pointers to locate data
- Pushes these pointers to servers
End-to-end System

Coordinator

SuccinctStores

SuffixStore

LogStore
End-to-end System

Client

Coordinator

Query (secondary key)

SuccinctStores

SuffixStore

LogStore
End-to-end System

Coordinator

SuccinctStores

SuffixStore

LogStore

Client

Query
(secondary key)
<table>
<thead>
<tr>
<th>Evaluation</th>
</tr>
</thead>
</table>
| **Datasets** | SmallKV, LargeKV from Conviva customers  
Record length: 140B (SmallKV), 1.3kB (LargeKV) |
| **Cluster** | Amazon EC2, 10 machines, m1.xlarge machines |
| **Workload** | YCSB for primary; YCSB mapped to secondary keys |
| **Systems** | MongoDB, Cassandra, HyperDex, DB-X |
| **Caveat** | Absolute numbers are dataset dependent |
10–11x lower storage

Take-away: Succinct pushes 10–11x more data in faster storage than systems with similar functionality (including every single bit stored)
Take-away: Succinct achieves performance comparable to existing open-source systems for queries on primary attributes
Throughput for 95% SEARCH + 5% PUT (SmallKV)

Take-away: Succinct by pushing more data in faster storage provides performance similar to existing systems for 10-11x larger data sizes.
Summary

Succinct pushes 10-11x more data in faster storage (compared to popular open-source systems)

Succinct executes queries directly on compressed representation

Avoids data scans and data decompression costs

Enables interactive queries on secondary keys (for semi-structured data)