TSTATS and PREFIX

How to get the most out of your lexicon, with walklex, tstats, indexed fields, PREFIX, TERM and ~~CASE~~

Richard Morgan

Principal Architect | Splunk

Richard Morgan

Principal Architect – Splunk

splunk> .conf20

Forward-Looking Statements

During the course of this presentation, we may make forward-looking statements regarding future events or plans of the company. We caution you that such statements reflect our current expectations and estimates based on factors currently known to us and that actual events or results may differ materially. The forward-looking statements made in the this presentation are being made as of the time and date of its live presentation. If reviewed after its live presentation, it may not contain current or accurate information. We do not assume any obligation to update any forward-looking statements made herein.

In addition, any information about our roadmap outlines our general product direction and is subject to change at any time without notice. It is for informational purposes only, and shall not be incorporated into any contract or other commitment. Splunk undertakes no obligation either to develop the features or functionalities described or to include any such feature or functionality in a future release.

Averaging one slide very 45s

=

=

The Key to Productivity Is Work Avoidance

i.e. don’t do work you don’t have to do

Search Performance Underpins Everything

Search load is the biggest factor in sizing Splunk (not ingestion)

Faster loading dashboards

- User experience is improved with faster completing searches
- User productivity improves as run \ test cycles are accelerated

Reduces the need for precomputation (summaries)

- Summaries should be used to reduce load, not accelerate slow searches

Better performance enables more use cases

- Improvements of x10 and x100 allow users to attack new problems
- Examine weeks and months of data, instead of just hours and minutes

Reduced server load

- Support more users on less hardware
- Improves ROI on hardware investment

Search Performance Underpins Everything

Search load is the biggest factor in sizing Splunk (not ingestion)

Faster loading dashboards

- User experience is improved with faster completing searches
- User productivity improves as run \ test cycles are accelerated

Reduces the need for precomputation (summaries)

- Summaries should be used to reduce load, not accelerate slow searches

Reduction in HW costs

Better performance enables more use cases

- Improvements of x10 and x100 allow users to attack new problems
- Examine weeks and months of data, instead of just hours and minutes

Reduced server load

- Support more users on less hardware
- Improves ROI on hardware investment

Minimize Work: Select indexes

`index=search_demo*` selects directories starting with search_demo

```
(base) rmorgan-mbp-4cb4b:splunk rmorgan$ ls -al
total 824
drwx------  252 rmorgan  wheel   8064 31 Aug 11:33 .
drwx--x---    4 rmorgan  wheel    128 24 Nov  2019 ..
-rw-r--r--@   1 rmorgan  wheel  12292 22 Jul 11:09 .DS_Store
-rw-------    1 rmorgan  wheel      0 31 Aug 11:10 .dirty_database
-rw-------    1 rmorgan  wheel      3 31 Aug 11:10 _audit.dat
-rw-------    1 rmorgan  wheel      3 31 Aug 11:10 _internal.dat
drwx------    7 rmorgan  wheel    224 24 Nov  2019 _internaldb
drwx------    6 rmorgan  wheel    192 18 Oct  2019 _introspection
-rw-------    1 rmorgan  wheel      3 31 Aug 11:11 _introspection.dat
drwx------    6 rmorgan  wheel    192 20 Nov  2019 _metrics
-rw-------    1 rmorgan  wheel      3 31 Aug 11:10 _metrics.dat
drwx------    6 rmorgan  wheel    192 22 Jul 10:34 _metrics_rollup
drwx------    6 rmorgan  wheel    192 18 Oct  2019 _telemetry
-rw-------    1 rmorgan  wheel      2 31 Aug 11:20 _telemetry.dat
drwx------    6 rmorgan  wheel    192 18 Oct  2019 audit
drwx------    2 rmorgan  wheel     64 18 Oct  2019 authDb
drwx------    6 rmorgan  wheel    192 30 Aug 13:34 defaultdb
drwx------    9 rmorgan  wheel    288 31 Aug 11:42 fishbucket
drwx------    2 rmorgan  wheel     64 18 Oct  2019 hashDb
drwx------    6 rmorgan  wheel    192 31 Aug 11:10 search_demo_1
-rw-------    1 rmorgan  wheel      2 31 Aug 11:30 search_demo_1.dat
drwx------    6 rmorgan  wheel    192 31 Aug 11:10 search_demo_2
-rw-------    1 rmorgan  wheel      2 31 Aug 11:33 search_demo_2.dat
drwx------    6 rmorgan  wheel    192 18 Oct  2019 summarydb
```

When we specify indexes in our search we are narrowing the directories we wish to access.

This is the highest level of exclusion in Splunk and it is minimal requirement for high performance search.

`index=*` selects all indexes, expect for those that start with an underscore (_internal, _audit etc)

Minimize Work: Select a timerange

Applying the filter earliest=-20d latest=-10d selects buckets to **consider**

```
(base) rmorgan-mbp-4cb4b:splunk rmorgan$ ls -al search_demo/db/
total 16
drwx------  25 rmorgan  wheel   800 30 Aug 21:04 .
drwx------   6 rmorgan  wheel   192 30 Aug 19:52 ..
-rw-------   1 rmorgan  wheel  2904 30 Aug 20:14 .bucketManifest
-rw-------   1 rmorgan  wheel    10 30 Aug 19:52 CreationTime
drwx--x---   2 rmorgan  wheel    64 30 Aug 19:52 GlobalMetaData
drwx--x---  16 rmorgan  wheel   512 30 Aug 19:54 db_1598984915_1598812143_60
drwx--x---  16 rmorgan  wheel   512 30 Aug 19:55 db_1598984915_1598984915_61
drwx--x---  16 rmorgan  wheel   512 30 Aug 19:56 db_1598984916_1598984915_62
drwx--x---  15 rmorgan  wheel   480 30 Aug 19:57 db_1598984916_1598984916_63
drwx--x---  16 rmorgan  wheel   512 30 Aug 19:59 db_1598984917_1598984916_64
drwx--x---  17 rmorgan  wheel   544 30 Aug 20:00 db_1598984917_1598984917_65
drwx--x---  15 rmorgan  wheel   480 30 Aug 20:01 db_1598984918_1598984917_66
drwx--x---  16 rmorgan  wheel   512 30 Aug 20:02 db_1598984918_1598984918_67
drwx--x---  16 rmorgan  wheel   512 30 Aug 20:03 db_1598984919_1598984918_68
drwx--x---  14 rmorgan  wheel   448 30 Aug 20:04 db_1598984919_1598984919_69
drwx--x---  15 rmorgan  wheel   480 30 Aug 20:05 db_1598984920_1598984919_70
drwx--x---  16 rmorgan  wheel   512 30 Aug 20:06 db_1598984920_1598984920_71
drwx--x---  13 rmorgan  wheel   416 30 Aug 20:07 db_1598984920_1598984920_72
drwx--x---  15 rmorgan  wheel   480 30 Aug 20:08 db_1598984921_1598984920_73
drwx--x---  13 rmorgan  wheel   416 30 Aug 20:09 db_1598984921_1598984921_74
drwx--x---  14 rmorgan  wheel   448 30 Aug 20:10 db_1598984922_1598984921_75
drwx--x---  16 rmorgan  wheel   512 30 Aug 20:11 db_1598984922_1598984922_76
drwx--x---  17 rmorgan  wheel   544 30 Aug 20:12 db_1598984923_1598984922_77
drwx--x---  16 rmorgan  wheel   512 30 Aug 20:13 db_1598984923_1598984923_78
drwx--x---  13 rmorgan  wheel   416 30 Aug 21:08 hot_v1_79
```

Each bucket encodes the time range for the data it holds in EPOC time.

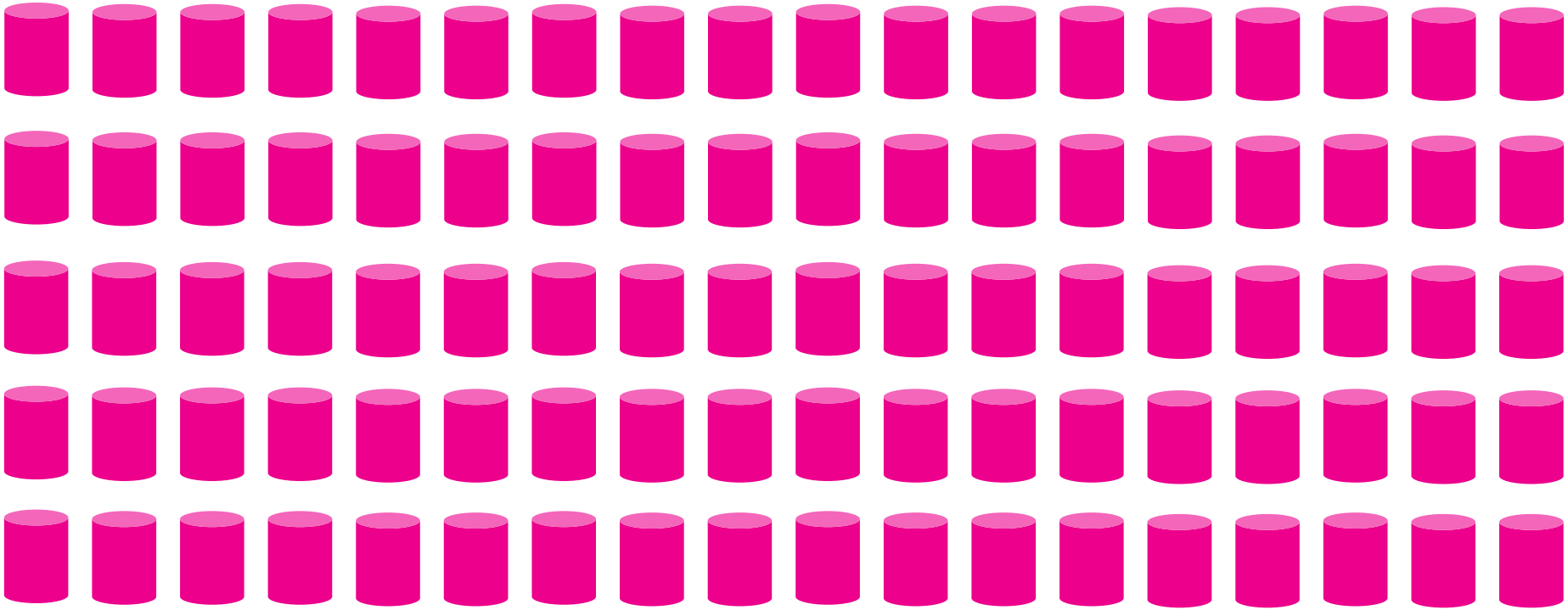
Therefore when we only consider bucket that have timestamps that fall into the time range we have specified.

Use “dbinpect” allows you understand this selection process without executing a full search.

Output: A list of Buckets to `consider`

Overstating the obvious!!!!

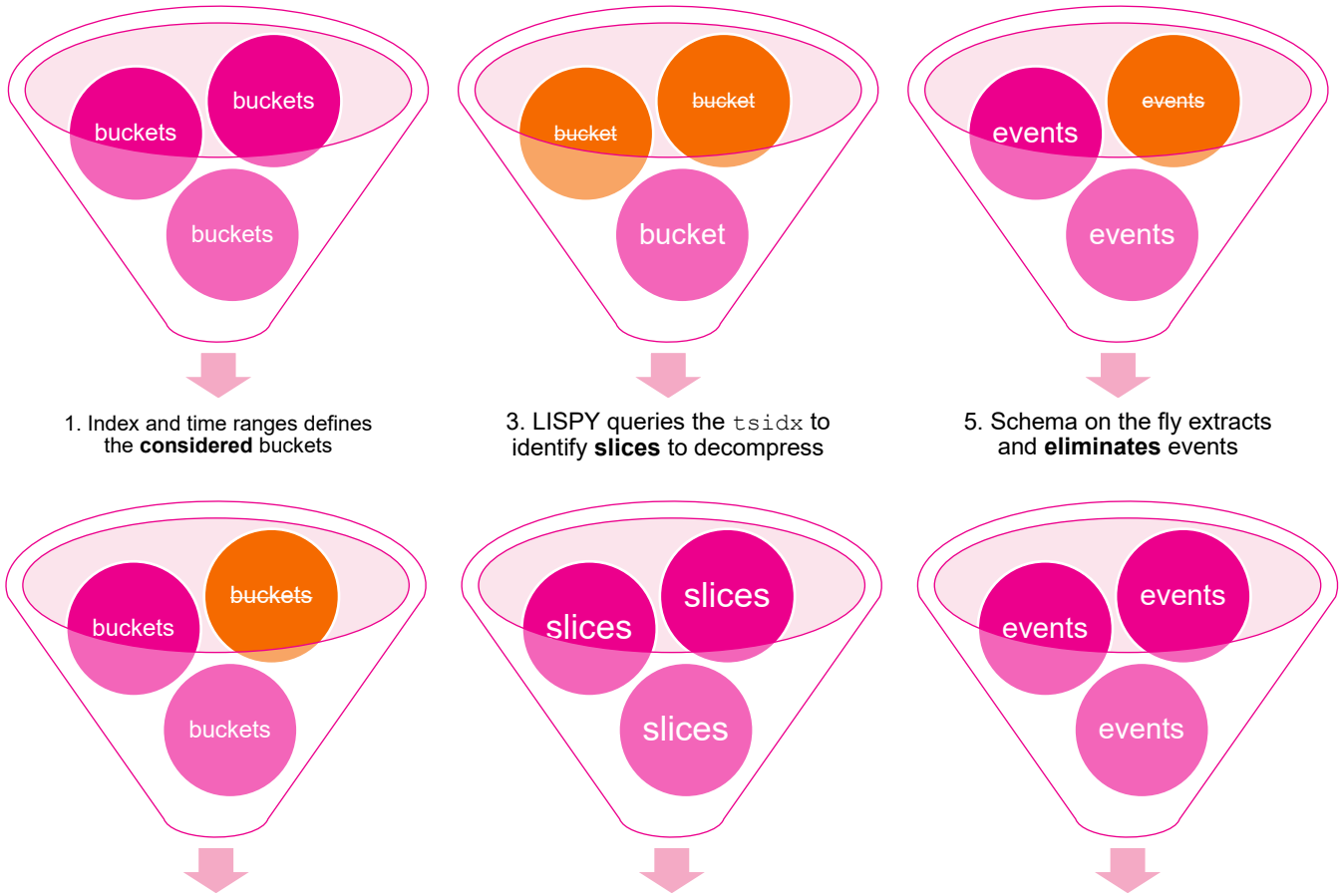
Time range + indexes selects buckets that must be processed



`considered_buckets`

WARNING: Massive oversimplification!

Six stages of indexer search processing



The first line of your search typically represents the greatest amount of computational effort required to execute your search.

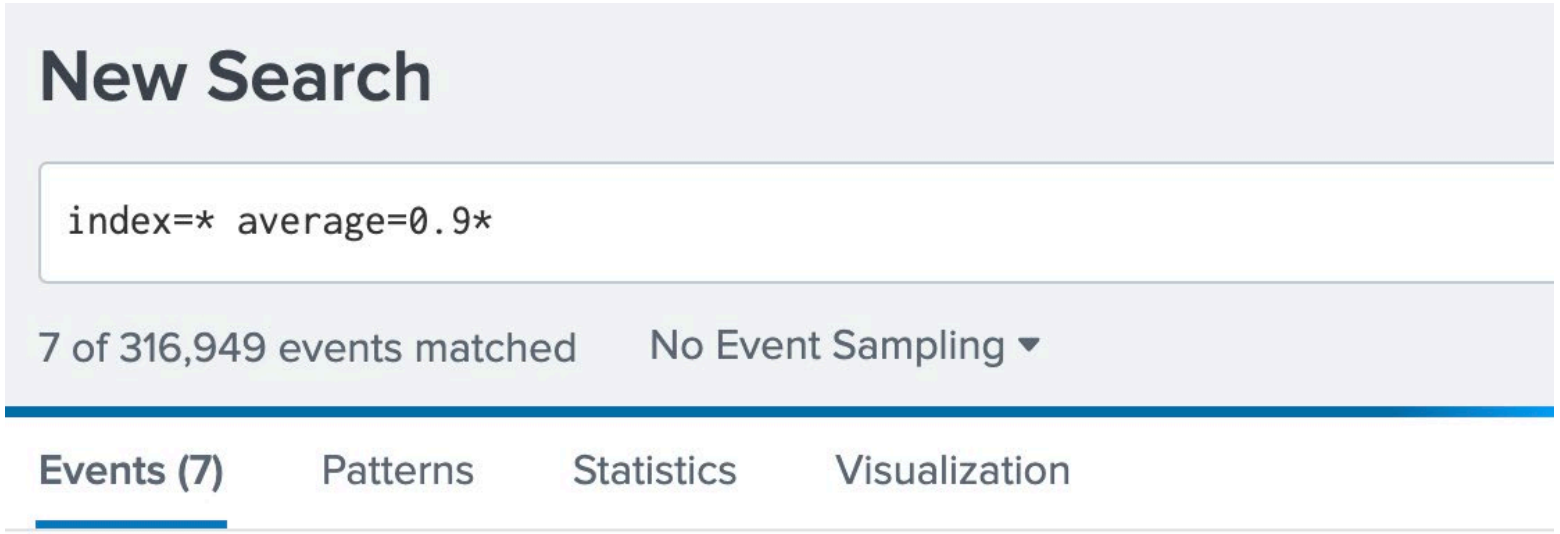
Making efficient use of the first line in your search results the greatest gains and everything else barely matters

```
index=<indexes> <constraints>
| <everything else>
```

splunk> .conf20

Scan Count Vs. Event Count

During execution you see the ratio between scan count and event count



Try and eliminate events BEFORE they are extracted from the raw data, as this avoids the CPU intensive decompression and parsing

False positive ratio

7 / 312,792

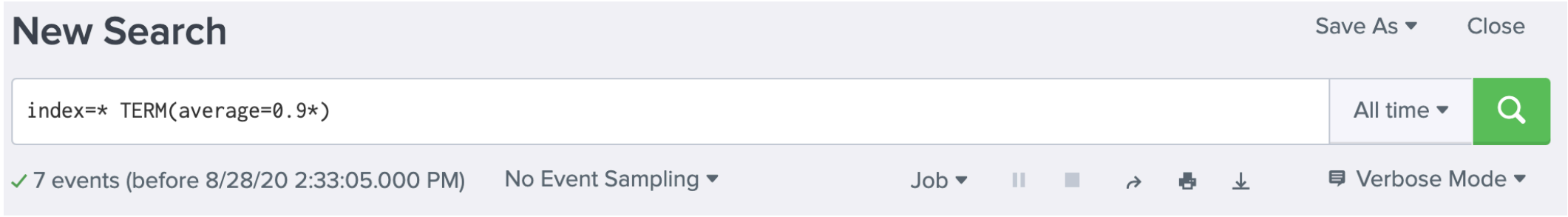
= 99.99%

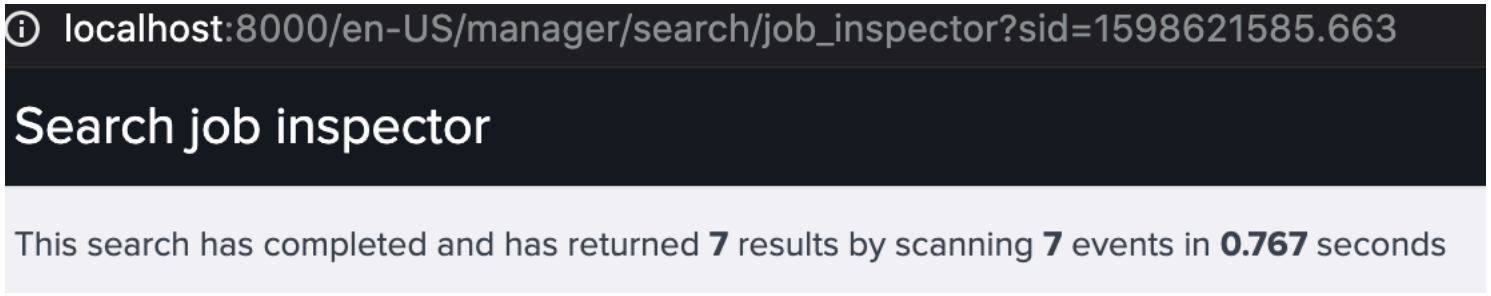
false matches

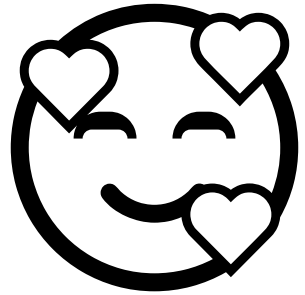
21 seconds to execute

Early Elimination Improves Performance

By introducing the TERM parameter into our search we have eliminated all false positives







`scan_count = event_count`

TERM is used in less the 1% of all customer searches executed on Splunk Cloud

The difference is in the LISPY

By introducing TERM we made changed the LISPY to be more precise

BEFORE:

- SPL = `index=* average=0.9*`
- LISPY = `[ AND 0 9* index::* ]`

AFTER:

- SPL = `index=* TERM(average=0.9*)`
- LISPY = `[ AND average=0.9* index::* ]`

MAJOR TERM

LISPY is the search language that we use to search the lexicon.

The first search looks for any event that includes all the of the **minor terms** `0 9*` in any index.

The second looks for any **major term** that starts "`average=0.9*`" in any index.

Where to MAJOR TERMS Come From?

Splunk uses a Universal indexing algorithm to tokenize events and write to index

Splunk has a two-stage parsing process

Firstly, we break up _raw with major breakers

Secondly, we apply minor breakers to the major breakers

This is configurable in limits.conf (beware changing!!!)

Major breakers

```
[ ] < > ( ) { } | ! ; , ' " * \n \r
\s \t & ? + %21 %26 %2526 %3B %7C
%20 %2B %3D -- %2520 %5D %5B %3A
%0A %2C %28 %29
```

Minor breakers

```
/ : = @ . - $  % \\ _
```

Step 1 – Applying MAJOR Breakers

How Splunk takes a log line and creates TERMS with major breakers

Input string (_raw):

```
01-27-2020 20:29:22.922 +0000 INFO Metrics - group=per_sourcetype_thruput,
ingest_pipe=0, series="splunkd", kbps=258.6201534528208, eps=1367.6474892210738,
kb=8050.1142578125, ev=42571, avg_age=145747.7853938127, max_age=2116525
```

Output MAJOR TERM list:

```
["01-27-2020", "20:29:22.922", "+0000", "info", "metrics", "-",
"group=per_sourcetype_thruput", "ingest_pipe=0", "series=", "splunkd",
"kbps=258.6201534528208", "eps=1367.6474892210738", "kb=8050.1142578125,", "ev=42571",
"avg_age=145747.7853938127", "max_age=2116525”]
```

Notice how all fields other than `series=` are tokenized into useful TERMS

Step 2 – Applying MINOR Breakers

How Splunk takes a log line and creates TERMS with minor breakers

Input array (MAJOR BREAKERS):

- ["01-27-2020", "20:41:20.355", "+0000", "info", "metrics", "-", "group=per_sourcetype_thruput", "ingest_pipe=0", "series=", "top", "kbps=23.83452969239664", "eps=155.64262209891208", "kb=743.4765625", "ev=4855", "avg_age=145747.7853938127", "max_age=2116525"]

These terms are **only** accessible with the `TERM` keyword

Output TERMS (MINOR BREAKERS):

- ["0000", "", "thruput","0", "01", "155", "20", "2020", "23", "27", "355", "41", "4765625", "4855", "64262209891208", "743", "83452969239664", "info", "metrics", "age", "avg", "eps", "ev", "group", "ingest", "kb", "kbps", "max", "per", "pipe", "series", "sourcetype"]

These terms are used for `_raw` search

SIDE NOTE: Over precision in numbers generates many unique `TERMS` and bloats the `tsidx` file

Eyeballing a log for MAJOR TERMS

Identifying and testing for MAJOR TERMS in your events is easy

Input event

```
01-27-2020 20:29:22.922 +0000 INFO Metrics - group=per_sourcetype_thruput,
ingest_pipe=0, series="splunkd", kbps=258.6201534528208, eps=1367.6474892210738,
kb=8050.1142578125, ev=42571, avg_age=145747.7853938127, max_age=2116525
```

Output token list MINOR + MAJOR

```
["01-27-2020", "20:41:20.355", "+0000", "info", ”metrics", "-",
"group=per_sourcetype_thruput", "ingest_pipe=0", "series=", "top",
"kbps=23.8345296923966 4", "eps=155.64262209891208", "kb=743.4765625", "ev=4855",
"avg_age=145747.78539381270", "max_age=2116525”, "0000", ", "thruput","0", "01", "155",
"20", "2020", "23", "27", "355", "41", "4765625", "4855", "64262209891208", "743",
"83452969239664", ”info", "metrics", "age", "avg", "eps", "ev", "group", "ingest", "kb",
"kbps", "max", "per", "pipe", "series", "sourcetype”]
```

Let's Update Our Example

Thanks to major breakers we have additional terms in our index

1. Tom. Rich and Harry
2. Bob loves Fred
3. Fred loves Susan
4. Harry loves Rich
5. Karen loves Susan
6. Loves. Susan Karen

TERM	Events with TERM
tom	1
tom.	1
rich	1,4
harry	1, 4
susan	3, 5, 6
bob	2
fred	2,3
karen	5, 6
loves	2,3,4,5,6
loves.	6

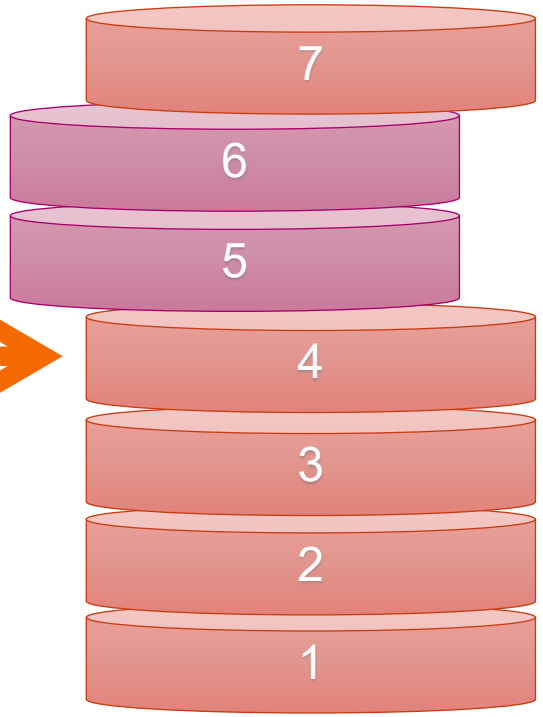
Search for Exact Match “Karen Loves Susan”

LISPY search = `[ AND karen loves susan ]`

TSIDX

TERM	Events containing TERM
tom	1
tom.	1
rich	1,4
harry	1, 4
susan	3, **5, 6**
bob	2
fred	2,3
karen	**5,6**
loves	2, 3, 4, **5, 6**
loves.	**6**

journal



The posting lists tells us that we have two slices that contain all the terms we need.

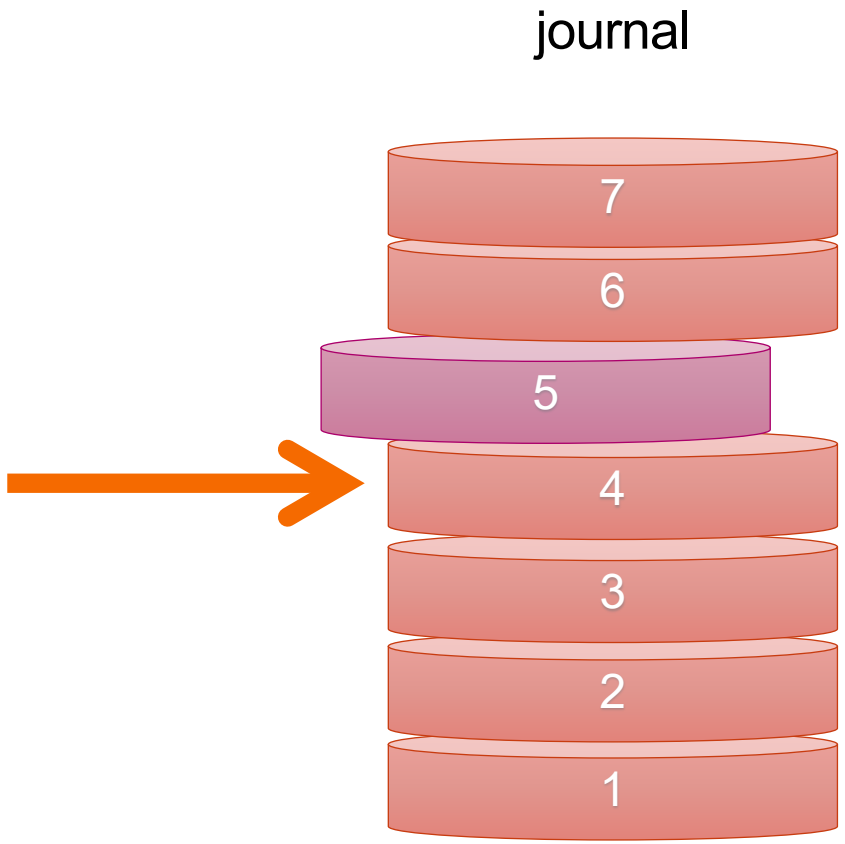
We extract these slices from the bucket, decompress and run though schema on the fly to see if they match.

Karen Loves Susan NOT TERM(loves.)

LISPY search = `[ AND karen loves susan [ NOT loves. ] ]`

TSIDX

TERM	Events with TERM
tom	1
tom.	1
rich	1,4
harry	1, 4
susan	3, **5, 6**
bob	2
fred	2,3
karen	**5,6**
loves	2, 3, 4, **5, 6**
loves.	**6**



But excluding ”loves.” (with the comma) we have stopped the need to open and parse slice 6.

This means only a single event is parsed onto index on the fly.

The false positive ratio is now 0% - doubling performance

"walklex" Lets to You Inspect the Lexicon

We can see INDEXED FIELDS when `type=fieldvalue`

```
| union
   [| walklex index=_internal pattern=host* type=fieldvalue
   | search term="host::c0m1*"
   | eval type="indexed field"
   | head 2 ]
    [| walklex index=_internal pattern=host* type=term
    | search term="host=c0m1*"
    | eval type="term"
    | head 2]
| rename source as bucket count as "instances of term found in bucket"
| table term type "instances of term found in bucket" bucket
```

Last 15 minutes

✓ 4 events (1/27/20 7:03:19.000 PM to 1/27/20 7:18:19.000 PM) No Event Sampling Job Smart Mode

Events Patterns **Statistics (4)** Visualization

20 Per Page Format Preview

term	type	instances of term found in bucket	bucket
host::c0m1-i-08665f45a4e3e5bab.cloudzero-quake.stg.splunkcloud.com	indexed field	66	_internal~43417~2550A42E-95FF-40DC-B0DA-7FE3B8743D4F
host::c0m1-i-0726bc1270cc03b56.csms-lzh7gm-12847.stg.splunkcloud.com	indexed field	68	_internal~2101~B92D0871-D670-4440-ADD9-DB879BE0D049
host=c0m1-i-074226ec4850444d1.serenaandlily.splunkcloud.com	term	59	_internal~2096~E0F7077A-FBA6-4C68-9E4D-068AD7ABBF27
host=c0m1-i-0db6769afc717272e.stack-cr3.splunkworks.lol	term	55	_internal~2096~E0F7077A-FBA6-4C68-9E4D-068AD7ABBF27

"walklex" Lets to You Inspect the Lexicon

We can see TERMS when `type=term`

```
| union
    [| walklex index=_internal pattern=host* type=fieldvalue
    | search term="host::c0m1*"
    | eval type="indexed field"
    | head 2 ]
    [| walklex index=_internal pattern=host* type=term
    | search term="host=c0m1*"
    | eval type="term"
    | head 2]
| rename source as bucket count as "instances of term found in bucket"
| table term type "instances of term found in bucket" bucket
```

Last 15 minutes

✓ 4 events (1/27/20 7:03:19.000 PM to 1/27/20 7:18:19.000 PM) No Event Sampling Job Smart Mode

Events Patterns **Statistics (4)** Visualization

20 Per Page Format Preview

term	type	instances of term found in bucket	bucket
host::c0m1-i-08665f45a4e3e5bab.cloudzero-quake.stg.splunkcloud.com	indexed field	66	_internal~43417~2550A42E-95FF-40DC-B0DA-7FE3B8743D4F
host::c0m1-i-0726bc1270cc03b56.csms-lzh7gm-12847.stg.splunkcloud.com	indexed field	68	_internal~2101~B92D0871-D670-4440-ADD9-DB879BE0D049
host=c0m1-i-074226ec4850444d1.serenaandlily.splunkcloud.com	term	59	_internal~2096~E0F7077A-FBA6-4C68-9E4D-068AD7ABBF27
host=c0m1-i-0db6769afc717272e.stack-cr3.splunkworks.lol	term	55	_internal~2096~E0F7077A-FBA6-4C68-9E4D-068AD7ABBF27

Splunk Has Two Major Search Options

`_raw search` has the most versatility, but advanced users use `tstats`

Fast ⟷ versatile

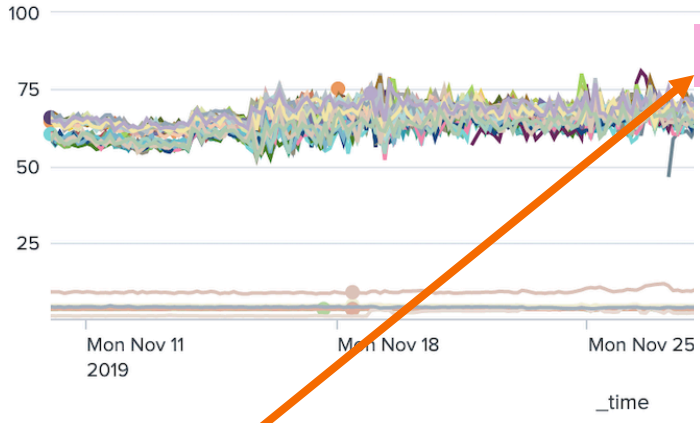
Example: Splunk's Hostwide Metrics (-31d)

Hostwide metrics uses "INDEXED_JSON" and can be queried both ways

134.169 seconds - rmorgan__rmorgan__search__RMD56faa7475a361f3b7_1576103206.70553

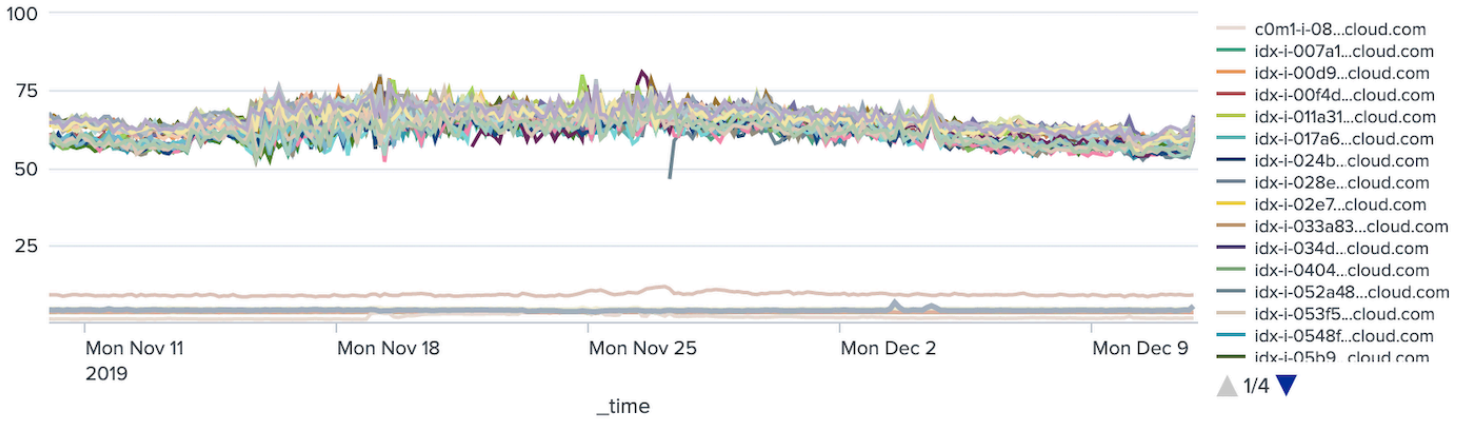
search index=_introspection data.cpu_user_pct=* host=* | timechart span=11040sec limit=0 p95(data.cpu_user_pct) as data.cpu_user_pct by host

Raw search
134 secs



3.442 seconds - rmorgan__rmorgan__search__search3_1576103206.70554

| tstats p95(data.cpu_user_pct) as cpu_user_pct where component=hostwide index=_introspection host=* by host _time span=11040sec | timechart span=11040sec limit=0 p95(cpu_user_pct) as cpu_user_pct by host



Equivalent tstats search
3 secs

Improvement is x39 faster for the same result set

splunk> .conf20

The Need for Indexed Fields Limits `tstats` Adoption

The prerequisite of indexed fields means its application is limited

Few `searches` can be converted to `tstats`

It is difficult to discover the existence of indexed fields when available

- The `walklex` function introduced in 7.3 helps
- The existence of TERMS can be inferred from log data

Although barely undocumented `tstats` supports the TERM() directive

Indexed Field Creation

There are various ways to get indexed fields into Splunk

At ingestion we can extract metadata from raw event and create indexed fields

- Uses props and transforms, normally via REGEX, sometimes INGEST_EVAL
- This is discouraged in favor of search time extractions

Some structured data sources can optionally create indexed fields automatically

- INDEXED_EXTRACTIONS works with CSV and JSON data
- This can bloat the TSIDX file, and is frequently disabled

HTTP Event collector has a "fields" section

- Slightly dangerous as clients define indexed fields and can bloat TSIDX

Post ingestion we use an create a datamodel

- Data models are based entirely on indexed fields, no raw events, just TSIDX files
- Building the data model requires a raw search, this can hide the true cost

How to Get the Most From Indexed Fields

If review complex pipeline configurations is your bag, you'll love this talk!

Shameless self promotion!!!

Platform Advanced

⊕ **PLA1154C - Advanced pipeline configurations with INGEST_EVAL and CLONE_SOURCETYPE**

Vladimir Skoryk, Principal Professional Services Architect, Splunk

Richard Morgan, Principal Architect, Splunk

INGEST_EVAL allows you to use eval statements to apply advanced logic to the pipeline process. There are various applications where this makes sense, including complex routing policies, event enrichment, event sampling, subsecond alerting, index...

Industry: Not industry specific
Products: Splunk Enterprise
Skill Level: Advanced

`tstats` Supports TERM

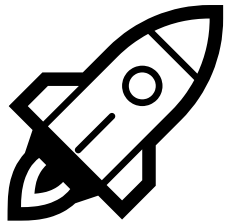
This is a log line from ITSI, lots of useful TERMS in here

```
09/24/2020 09:26:00 +0000, search_name="Indicator - Shared -
5dd8512622092b554f3e7da7 - ITSI Search", search_now=1600939560.000,
info_min_time=1600939260.000, info_max_time=1600939560.000,
info_search_time=1600939594.150, qf="", kpi="Average Alert Severity",
kpiid="ec77165d-e79f-4379-9534-3479954e64a6", urgency=5, serviceid="9a6bdac6-
fa6c-423e-81dc-785dbf75637e", itsi_service_id="9a6bdac6-fa6c-423e-81dc-
785dbf75637e", is_service_aggregate=1, is_entity_in_maintenance=0,
is_entity_defined=0, entity_key=service_aggregate, is_service_in_maintenance=0,
kpibasesearch=5dd8512622092b554f3e7da7, is_filled_gap_event=0,
alert_color="#F26A35", alert_level=5, alert_value=5, itsi_kpi_id="ec77165d-e79f-
4379-9534-3479954e64a6", is_service_max_severity_event=1, alert_severity=high,
alert_period=1, entity_title=service_aggregate, hostname="https://itsi-
search.customer.com:443"
```

We can use TERM on any of the tokens highlighted in yellow, but notice the one in RED

tstats Supports TERM

Some simple searches can be expressed with TERM



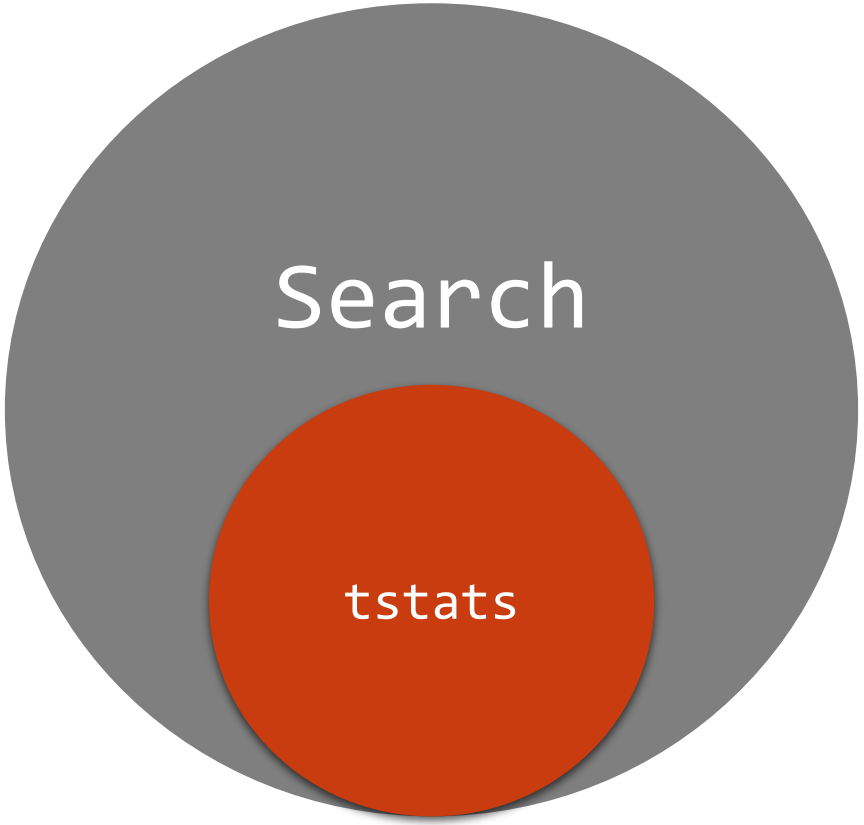
tstats version is **48x** faster

```
index=itsi_summary TERM(alert_severity=*)
| timechart span=1sec count by alert_severity
```

```
| tstats prestats=t count where index=itsi_summary TERM(alert_severity=high) by _time span=1sec
| fillnull "high" alert_severity
| tstats prestats=t append=t count where index=itsi_summary TERM(alert_severity=low) by _time span=1sec
| fillnull "low" alert_severity
| tstats prestats=t append=t count where index=itsi_summary TERM(alert_severity=medium) by _time span=1sec
| fillnull "medium" alert_severity
| tstats prestats=t append=t count where index=itsi_summary TERM(alert_severity=normal) by _time span=1sec
| fillnull "normal" alert_severity
| tstats prestats=t append=t count where index=itsi_summary TERM(alert_severity=unknown) by _time span=1sec
| fillnull "unknown" alert_severity
| timechart limit=50 span=1sec count by alert_severity
```

PREFIX Directive Added to `tstats` In v8

With PREFIX indexed fields are not longer a prerequisite for `tstats`



With PREFIX many more `searches` can be converted to `tstats` in v8

The extension massively increases the instances where `tstats` can be used

PREFIX allows TERMS to be processed as if they were indexed fields, for example:

- Indexed field search: `| tstats count by host`
- TERM search: `| tstats count by PREFIX(host=)`

PREFIX is also supported in aggregators:

- Indexed field search: `| tstats sum(PREFIX(value=))`

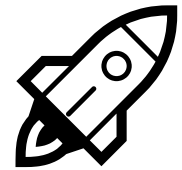
tstats Supports PREFIX()

PREFIX greatly simplifies our search

Who needs data models??

```
| tstats prestats=t count where index=itsi_summary TERM(alert_severity=high) by _time span=1sec
| fillnull "high" alert_severity
| tstats prestats=t append=t count where index=itsi_summary TERM(alert_severity=low) by _time span=1sec
| fillnull "low" alert_severity
| tstats prestats=t append=t count where index=itsi_summary TERM(alert_severity=medium) by _time span=1sec
| fillnull "medium" alert_severity
| tstats prestats=t append=t count where index=itsi_summary TERM(alert_severity=normal) by _time span=1sec
| fillnull "normal" alert_severity
| tstats prestats=t append=t count where index=itsi_summary TERM(alert_severity=unknown) by _time span=1sec
| fillnull "unknown" alert_severity
| timechart limit=50 span=1sec count by alert_severity
```

```
| tstats count where index=itsi_summary TERM(alert_severity=*)
        by PREFIX(alert_severity=) _time span=1sec
| rename alert_severity= as alert_severity
| xyseries _time alert_severity count
```



prefix version is **3x** faster again!

Q. What is the ingestion over 24 hours?

Every host generates metrics about its ingestion throughput very 30 seconds

```
01-21-2020 12:25:44.311 +0000 INFO Metrics - group=thruput,
ingest_pipe=1, name=thruput, instantaneous_kbps=3.366894499322308,
instantaneous_eps=12.163696322058637, average_kbps=47.777961955016565,
total_k_processed=31355244, kb=104.6298828125, ev=378, load_average=2.42
```

Load average = how hard the server is working
Kb = the data processed since the last reading
Instantaneous_kbps = the ingestion rate at point of measurement
Pipeline = the ingestion pipeline the reading is from

Search conversion `raw -> tstats`

This search demonstrated a 10x performance improvement over 24 hours

Raw search

```
index=_internal host IN (idx*) group=thruput name=thruput
| bin span=1767s _time
| stats
    sum(kb) as indexer_kb
    avg(instantaneous_kbps) as instantaneous_kbps
    avg(load_average) as load_avg
    by host _time
```

PREFIX search

```
| tstats
    sum(PREFIX(kb=)) as indexer_kb
    avg(PREFIX(instantaneous_kbps=)) as instantaneous_kbps
    avg(PREFIX(load_average=)) as load_avg
    where
    host IN (idx*) index=_internal
       host=idx* TERM(group=thruput) TERM(name=thruput)
    by host _time span=1767s
```

`prefix` version is

30x faster!

How did cachemgr behave over 24 hours?

Metrics.log group=cachemgr_bucket

```
09-21-2020 12:10:41.051 +0000 INFO  Metrics - group=cachemgr_bucket, open=4557, close=4561, cache_hit=4557, open_buckets=4

09-21-2020 12:10:44.330 +0000 INFO  Metrics - group=cachemgr_bucket, open=3550, close=3550, cache_hit=3550, open_buckets=4

09-21-2020 12:10:39.985 +0000 INFO  Metrics - group=cachemgr_bucket, open=3412, close=3415, cache_hit=3412, open_buckets=4

09-21-2020 12:10:44.102 +0000 INFO  Metrics - group=cachemgr_bucket, register_start=1, open=4096, close=4100, cache_hit=4096, open_buckets=6

09-21-2020 12:10:45.709 +0000 INFO  Metrics - group=cachemgr_bucket, register_start=1, register_end=1, open=3162, close=3164, cache_hit=3162, open_buckets=5

09-21-2020 12:10:41.229 +0000 INFO  Metrics - group=cachemgr_bucket, register_cancel=1, open=4794, close=4796, cache_hit=4794, open_buckets=7

09-21-2020 12:10:10.012 +0000 INFO  Metrics - group=cachemgr_bucket, open=4783, close=4779, cache_hit=4783, open_buckets=8

09-21-2020 12:10:23.227 +0000 INFO  Metrics - group=cachemgr_bucket, register_start=1, open=2896, close=2896, cache_hit=2896, open_buckets=4
```

Search conversion `raw -> tstats`

How did cache behave over 24 hours?

Raw search

```
index=_internal host IN (idx*) TERM(group=cachemgr_bucket)
| bin span=1798s _time
| stats
    sum(absent_summary_skipped) as absent_summary_skipped
    sum(bootstrap_summary) as bootstrap_summary
    sum(cache_hit) as cache_hit
    sum(cache_miss) as cache_miss
    sum(close) as close
    sum(close_all) as close_all
    by host _time
```

PREFIX search

```
| tstats
    sum(PREFIX(absent_summary_skipped=)) as absent_summary_skipped
    sum(PREFIX(bootstrap_summary=)) as bootstrap_summary
    sum(PREFIX(cache_hit=)) as cache_hit
    sum(PREFIX(cache_miss=)) as cache_miss
    sum(PREFIX(close=)) as close
    sum(PREFIX(close_all=)) as close_all
    where
    index=_internal host IN (idx*) TERM(group=cachemgr_bucket)
    by host _time span=1798s
```

`prefix` version is

25x faster!

Other segmenters.conf Options

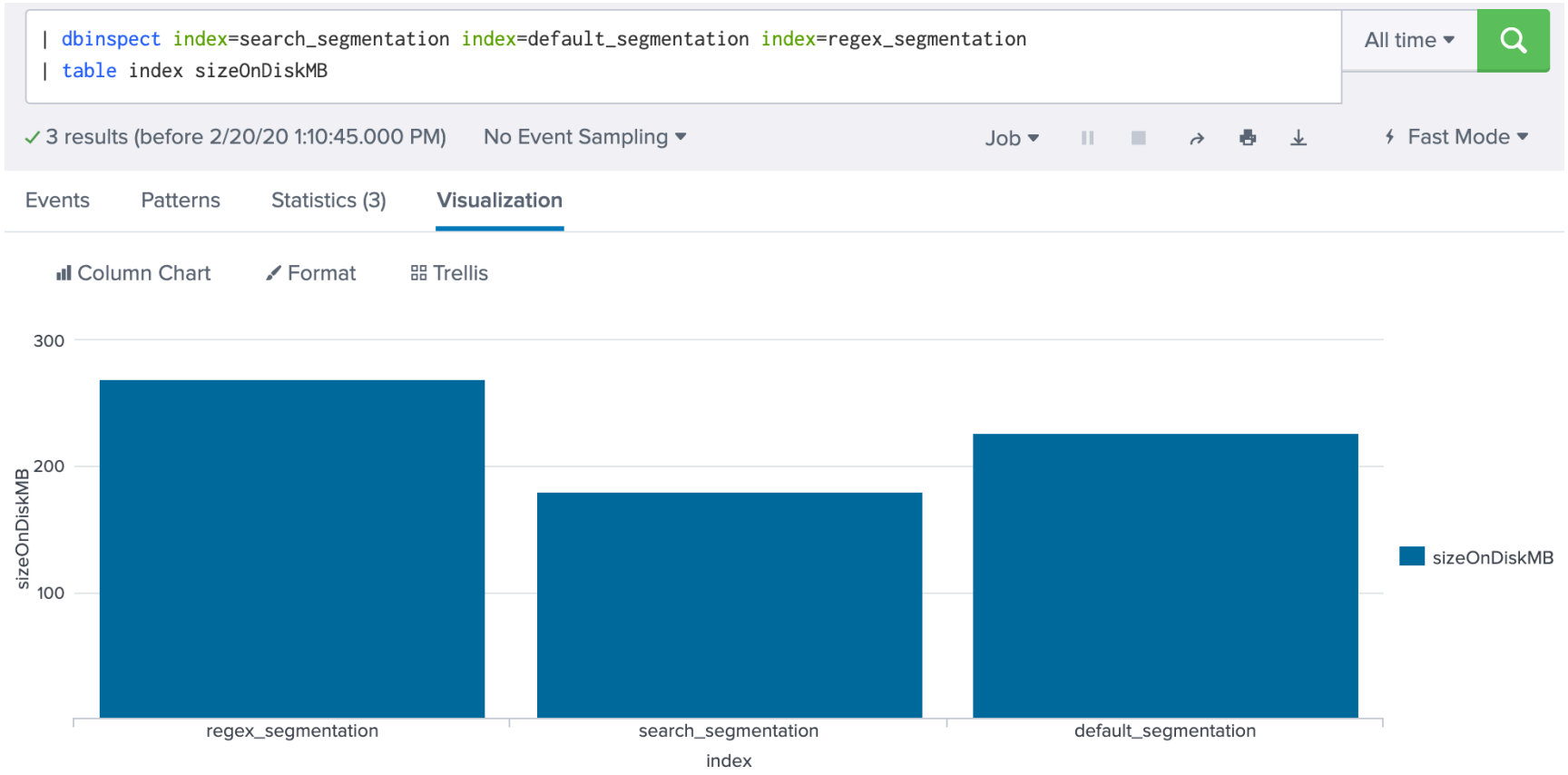
You can disable major breakers per sourcetype by indexing with “search”

```
[full]

[indexing]
# change INTERMEDIATE_MAJORS to "true" if you want an ip address to appear in typeahead as a, a.b, a.b.c, a.b.c.d
# the typical performance hit by setting to "true" is 30%
INTERMEDIATE_MAJORS = false

[search]
MAJOR = [ ] < > ( ) { } | ! ; , ' " \n \r \s \t & ? + %21 %26 %2526 %3B %7C %20 %2B %3D -- %2520 %5D %5B %3A %0A %2C %28 %29 / : = @ . - $ # % \\ _
MINOR =

[standard]
MAJOR = [ ] < > ( ) { } | ! ; , ' " * \n \r \s \t / : = @ . ? - & $ # + % _ \\ %21 %26 %2526 %3B %7C %20 %2B %3D -- %2520
MINOR =

[inner]
MAJOR = [ ] < > ( ) { } | ! ; , ' " * \n \r \s \t / : = @ . ? - & $ # + % _ \\ %21 %26 %2526 %3B %7C %20 %2B %3D -- %2520
MINOR =

[outer]
MAJOR = [ ] < > ( ) { } | ! ; , ' " * \n \r \s \t & ? + %21 %26 %2526 %3B %7C %20 %2B %3D -- %2520
MINOR =
```

WARNING: play with segmenters at your peril!

Testing Segmentation Options on splunkd.log

Major breakers are very expensive on storage if you don't use them



Removing all major breakers drops bucket size by 20%

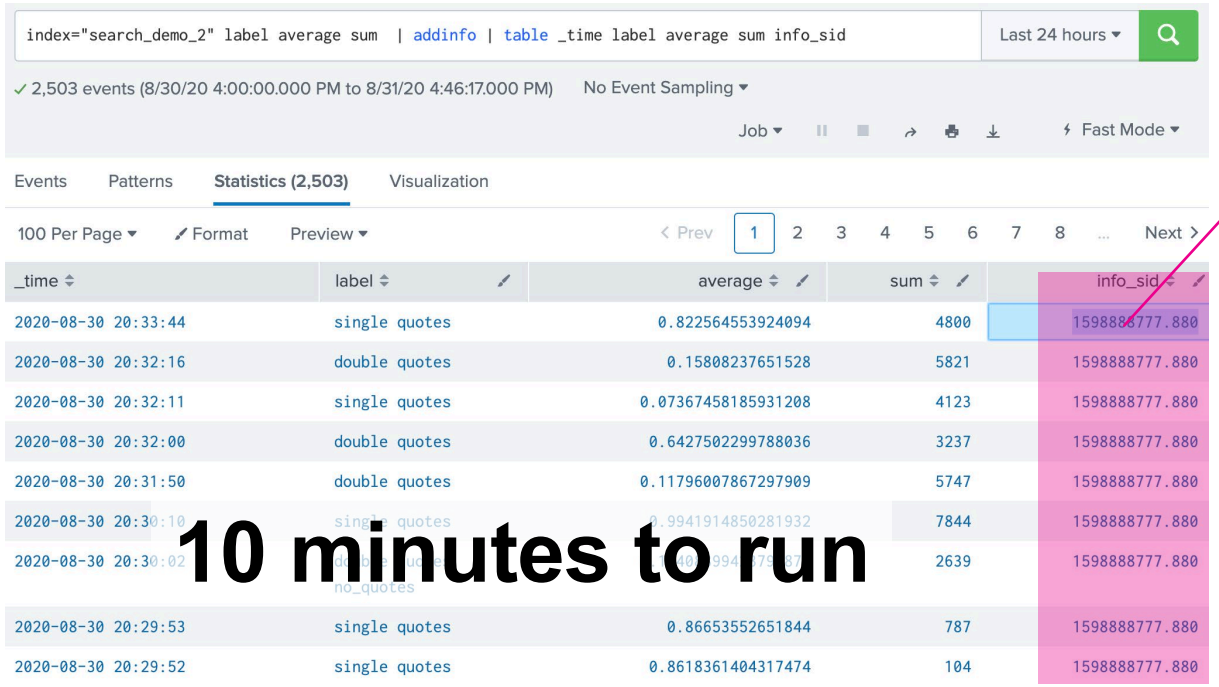
Using regex to extract all attribute value pairs, including quoted strings increased the size of the search segmentation by 50%

Switching from default to regex extraction caused an increase of 18%

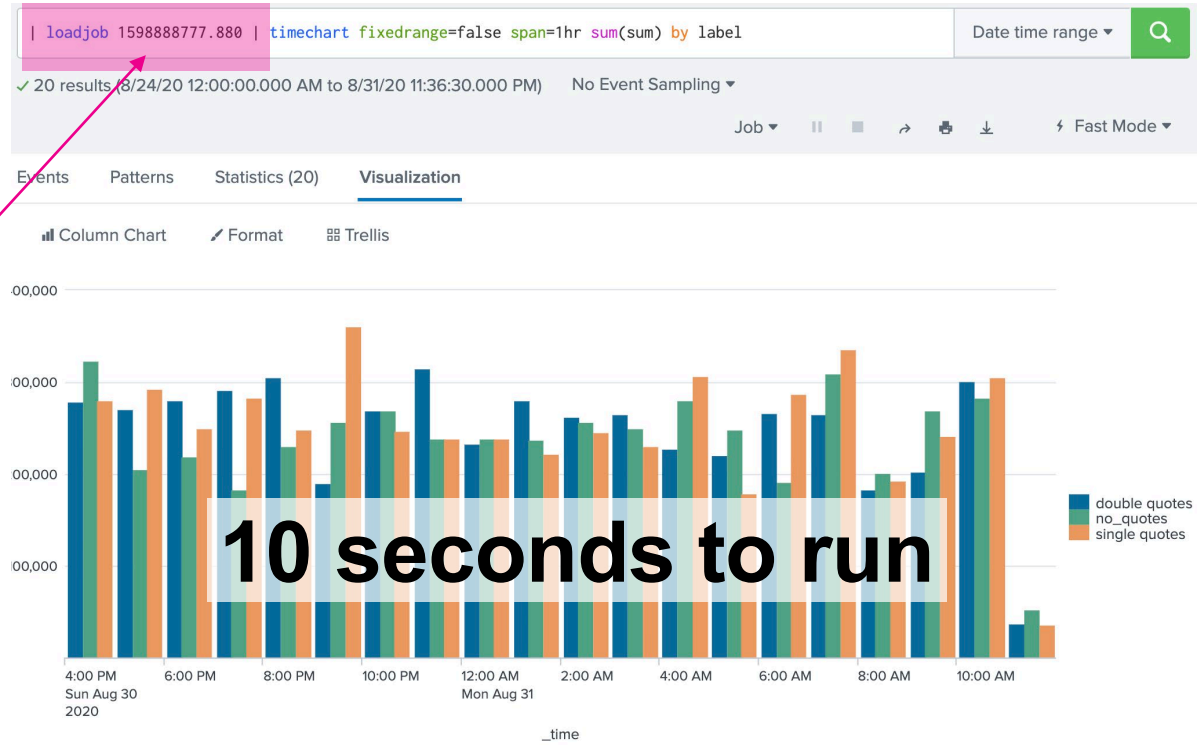
splunk> .conf20

Work Avoidance – Loadjob

You can execute a search in one location and then use the results it in another



10 minutes to run



10 seconds to run

When developing complex searches on large data sets, avoid repeatedly reloading event data from indexers as you iterate towards your solution

splunk> .conf20

Work Avoidance – Dashboard Base Searches

Run base searches once, use child searches to modify the base data set

```
<form>
  <search id="run_once">
    <query>
        index="search_demo_2" label average sum
        | timechart sum(sum) avg(sum)
    </query>
  </search>
  <search base="run_once">
    <query>
        | table _time $show_field$
    </query>
  </search>
  <fieldset>
    <input type="dropdown" token="show_field">
      <label>show field</label>
      <choice value="avg(sum)">avg</choice>
      <choice value="sum(sum)">sum</choice>
    </input>
  </fieldset>
</form>
```

Base search contains no tokens, it remains static

The child search contains the token and is reevaluated whenever it is updates

The user can modify the `$show_field$` token without causing the base search to execute

This is how you build is a high-performance interactive dashboards

Free performance boost! 1/2

Make your buckets smaller and your searches go slightly faster by updating the config

Use this one!

```
journalCompression = gzip|lz4|zstd
* The compression algorithm that splunkd should use for the rawdata journal
  file of new index buckets.
* This setting does not have any effect on already created buckets. There is
  no problem searching buckets that are compressed with different algorithms.
* "zstd" is only supported in Splunk Enterprise version 7.2.x and higher. Do
  not enable that compression format if you have an indexer cluster where some
  indexers run an earlier version of Splunk Enterprise.
* Default: gzip
```

We have been improving the compression on buckets, have you updated your configurations yet?

Free performance boost 2/2

The TSIDX files are normally bigger than the journal, so use latest compression

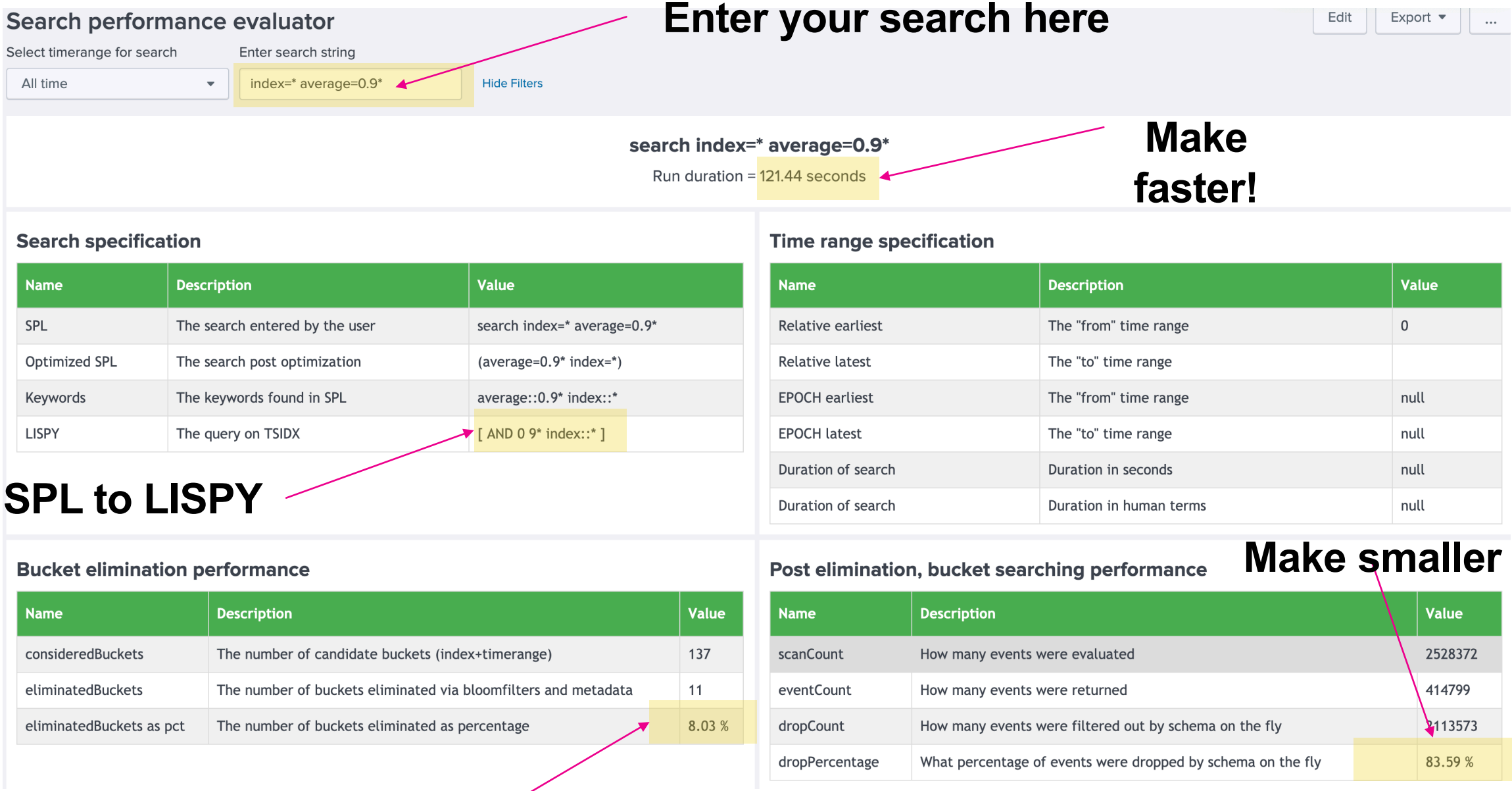
Use level 3!

```
tsidxWritingLevel = [1|2|3]
* Enables various performance and space-saving improvements for tsidx files.
* For deployments that do not have multi-site index clustering enabled,
    set this to the highest value possible for all your indexes.
  * For deployments that have multi-site index clustering, only set
    this to the highest level possible AFTER all your indexers in the
    cluster have been upgraded to the latest code level.
  * Do not configure indexers with different values for 'tsidxWritingLevel'
    as downlevel indexers cannot read tsidx files created from uplevel
peers.
  * The higher settings take advantage of newer tsidx file formats for
    metrics and log events that decrease storage cost and increase
performance
* Default: 1
```

Who doesn't want this for free?

Everybody Gets a Dashboard

https://github.com/silkyrich/cluster_health_tools/blob/master/default/data/ui/views/search_performance_evaluator.xml



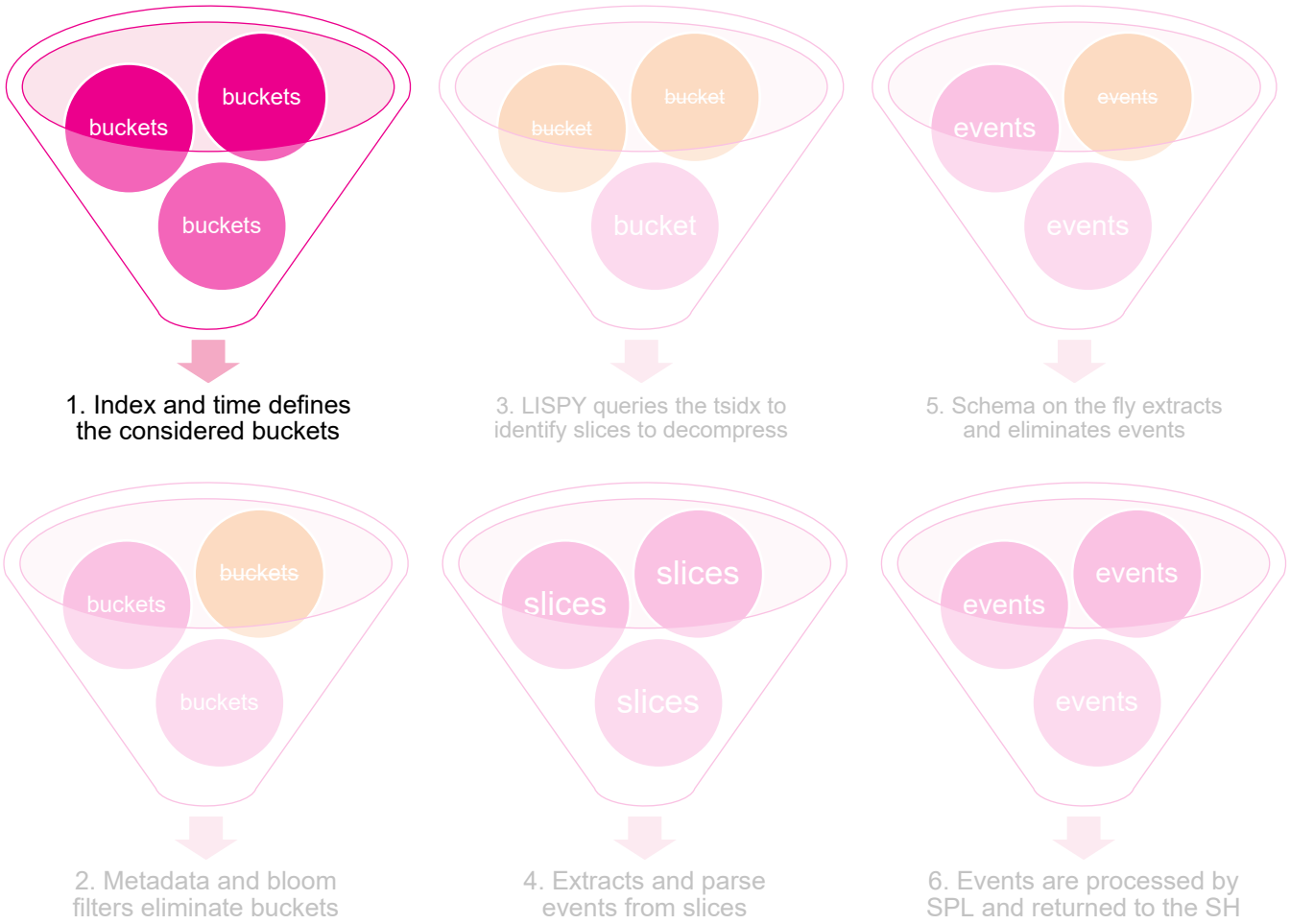
splunk> .conf20

Thank You

Please provide feedback via the

SESSION SURVEY

1. Index and time defines considered buckets

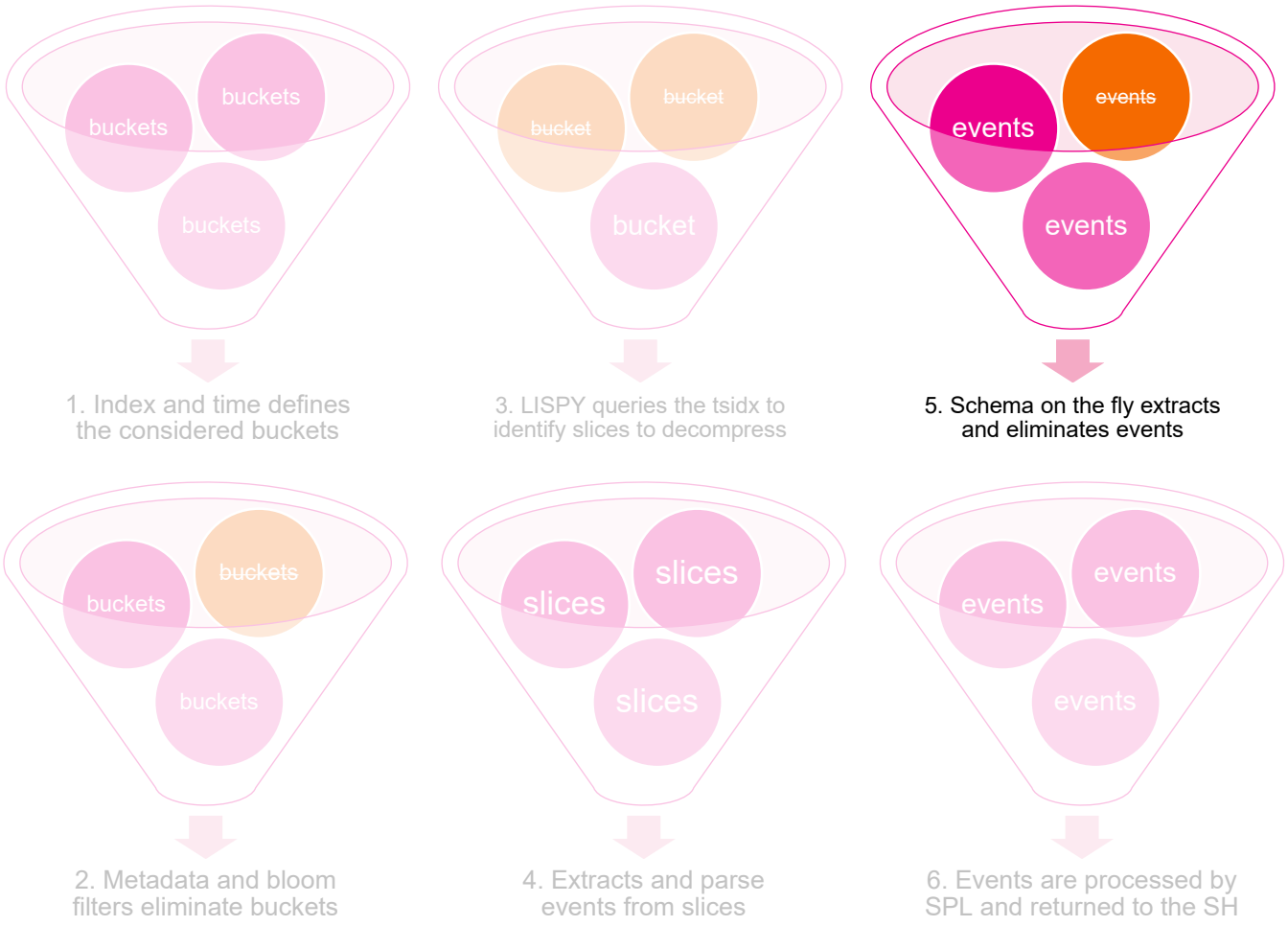


All searches are executed with an index and a time range. This defines our list of buckets to consider.

The first performance tip is to make this as tight as possible.

Minimize indexes and narrow the time range

Why is performance so bad?

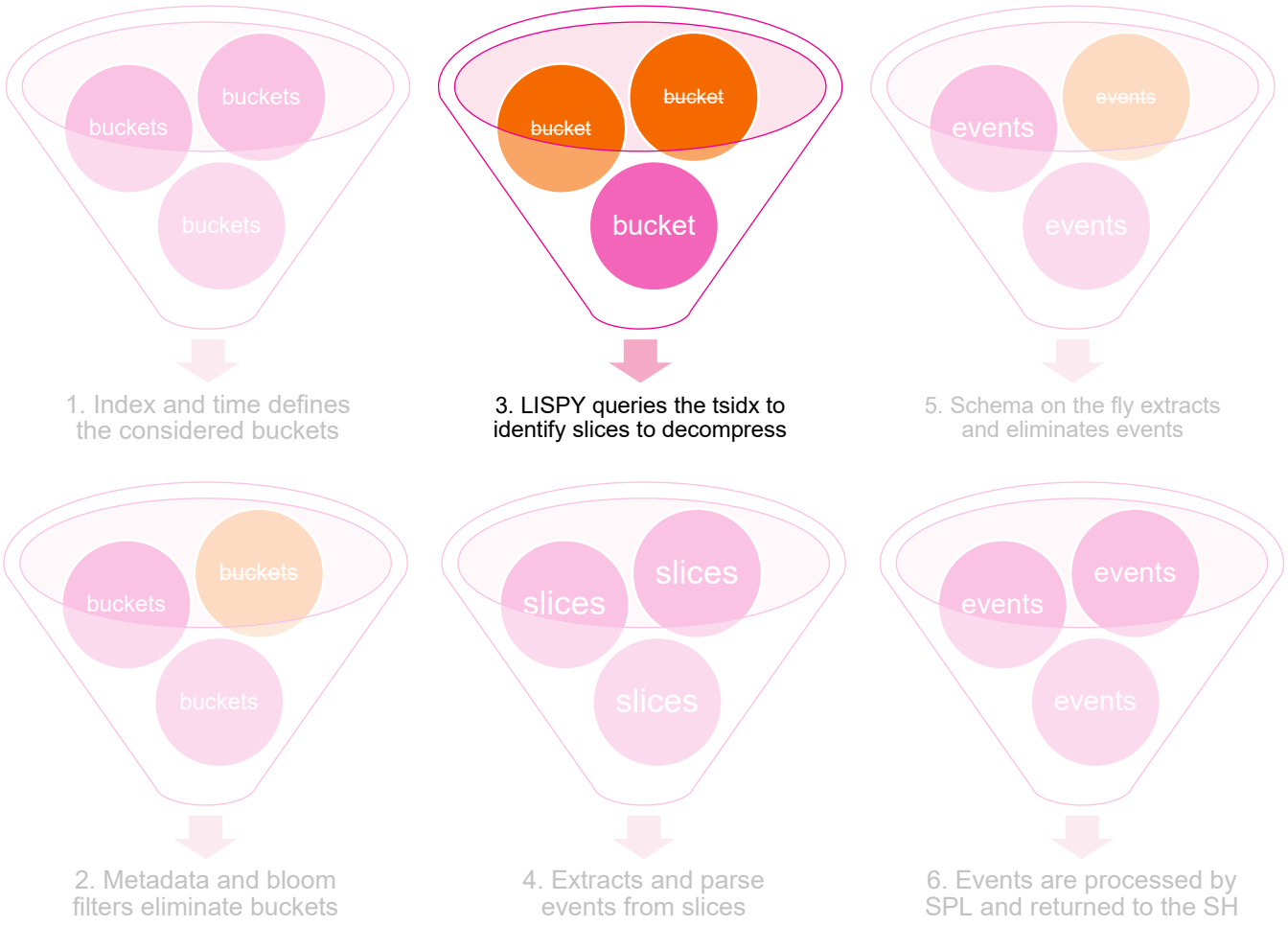


When the scan count is high and the event count is low we are filtering events during schema on the fly.

This is the most expensive place to filter as we have downloaded buckets, open the `tsidx`, extracted slices fully parsed events.

Minimize filtering during schema on the fly

What happened?

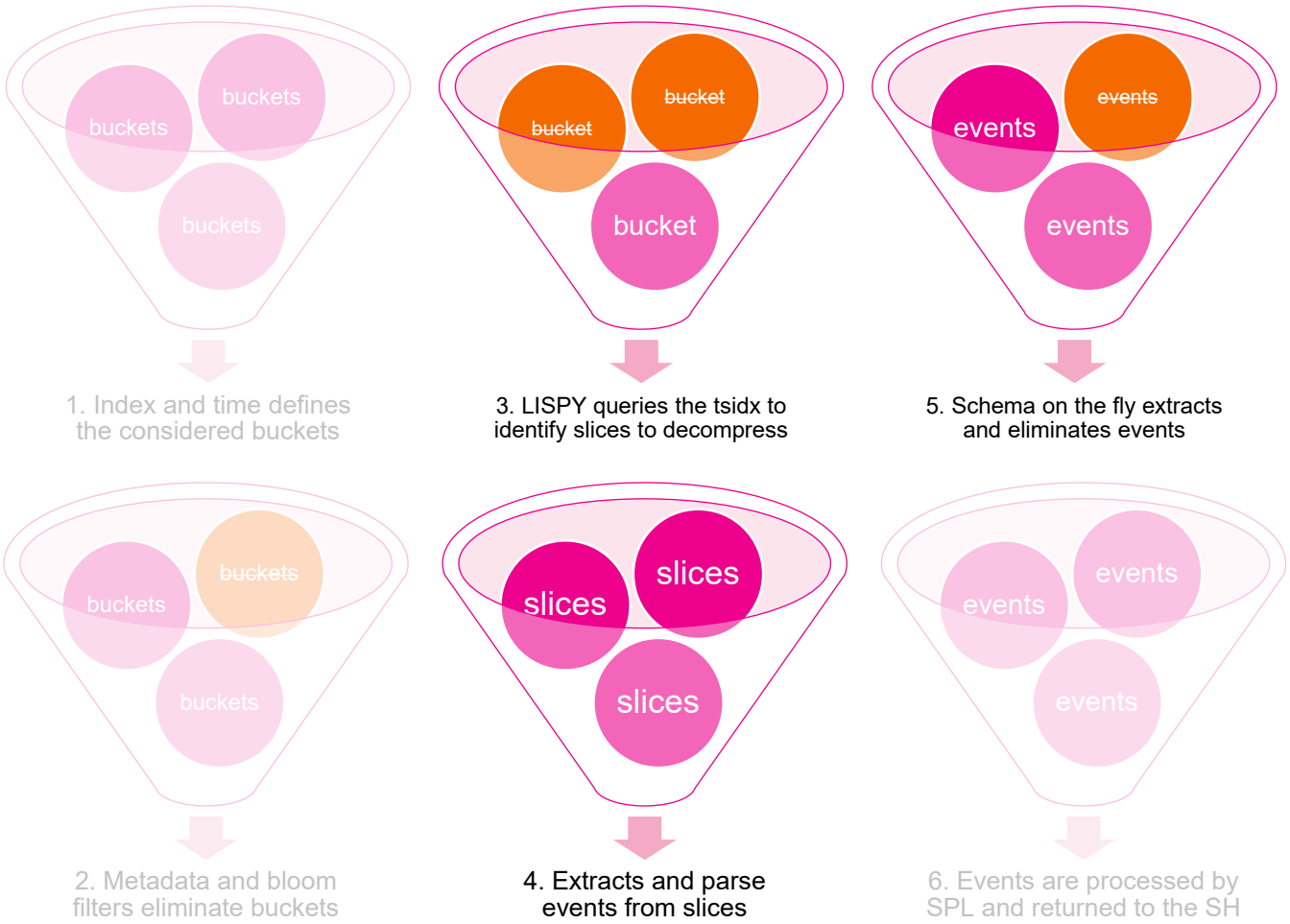


By introducing TERM to our search we were able to improve elimination earlier in the pipeline.

Doing so saves downloading journal files from SmartStore, and reduces CPU required for decompression and parsing

Minimize filtering during schema on the fly stage

Processing the considered buckets



After we have selected our range of buckets to search we must find and extract the data from them to do so.

Where the filtering is performed can have a dramatic impact to search performance.

Agenda

1. **Introduction**
   What this presentation is all about
2. **Search and workload elimination**
   How search works and where time is spent
3. **How the index is built**
   How universal indexing builds the lexicon
4. **Bloomfilter elimination**
   How bloomfilters accelerate _raw search
5. **Advanced indexing with Major breaker**
   How major breakers and turbo charge elimination
6. **Introducing `tstats`**
   How `tstats` delivers further performance improvements
7. **Other tricks and a performance dashboard**
   loadjob, base searches and take away dashboard

Let's recap for a minute

Explaining TSIDX and the Lexicon

Universal indexing breaks down the log lines and extracts the tokens to build a map

1. Tom. Rich and Harry
2. Bob loves Fred
3. Fred loves Susan
4. Harry loves Rich
5. Karen loves Susan
6. Loves. Susan Karen

Universal indexing

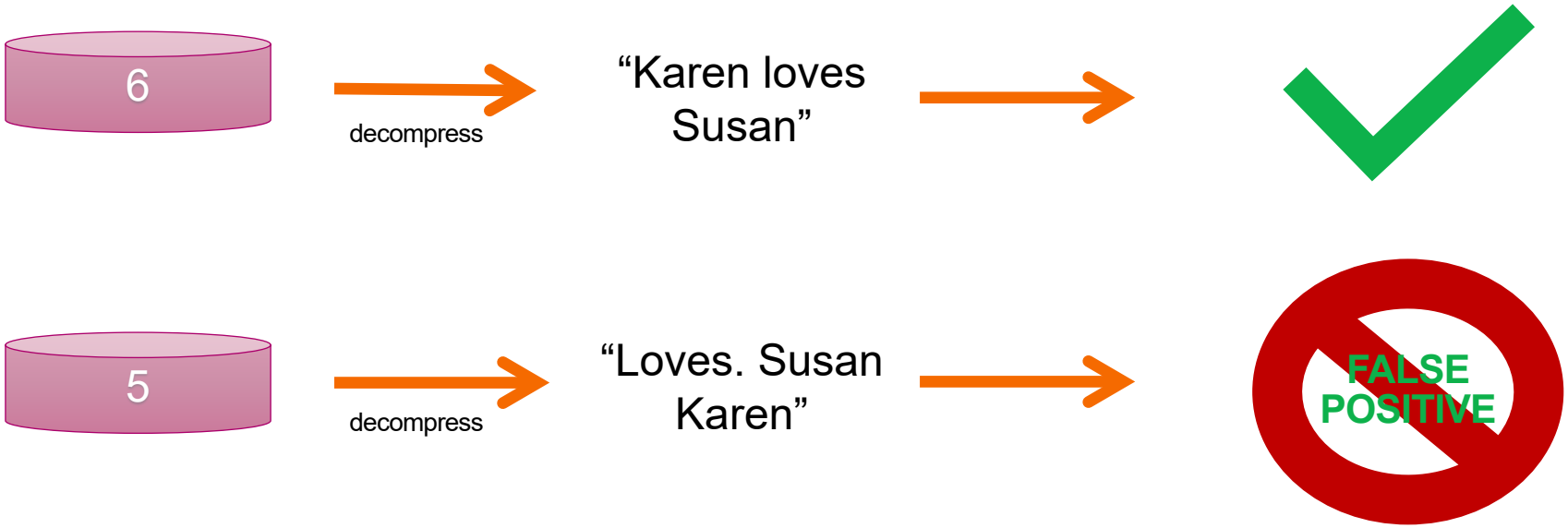
Minor breakers

The lexicon is composed of lowercase TERMS

TERM	Events with TERM
tom	1
rich	1,4
harry	1, 4
susan	3, 5, 6
bob	2
fred	2,3
karen	5, 6
loves	2,3,4,5,6

“Karen Loves Susan” matched two events

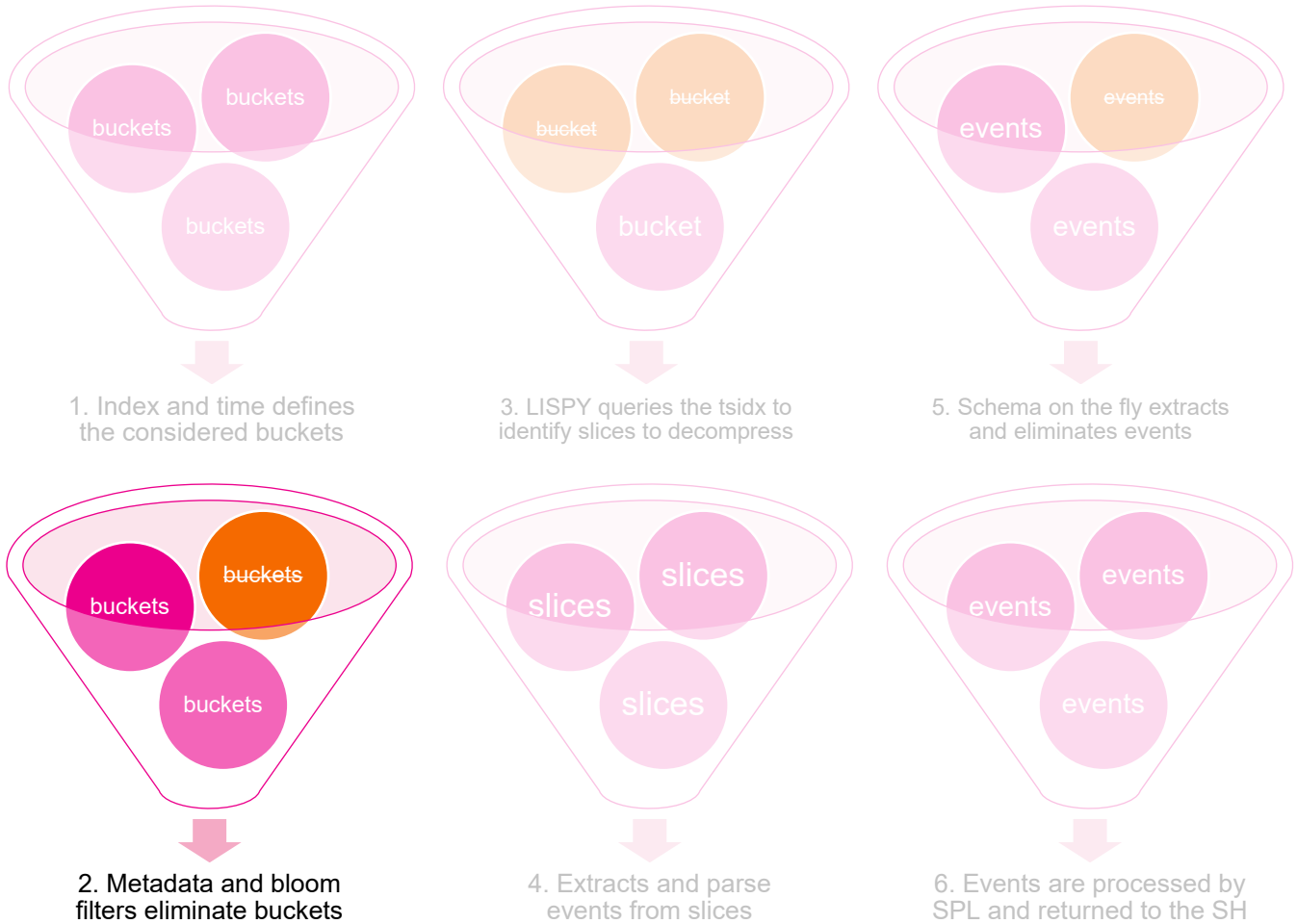
We have extracted two **slices**, **scanned** two events and returned one **event**




```
scan_count=2, event_count=1
```

Implies a 50% event elimination during schema on the fly

Bloomfilters and metadata eliminate buckets



Buckets that are eliminated to not have to be further processed, plus we don't need to download tsidx or the journal

Dependent on search, data and event distribution Splunk can eliminate up to 99% of buckets.

Second performance tip maximize elimination

Use host, source and sourcetype plus spare terms to help bucket elimination.

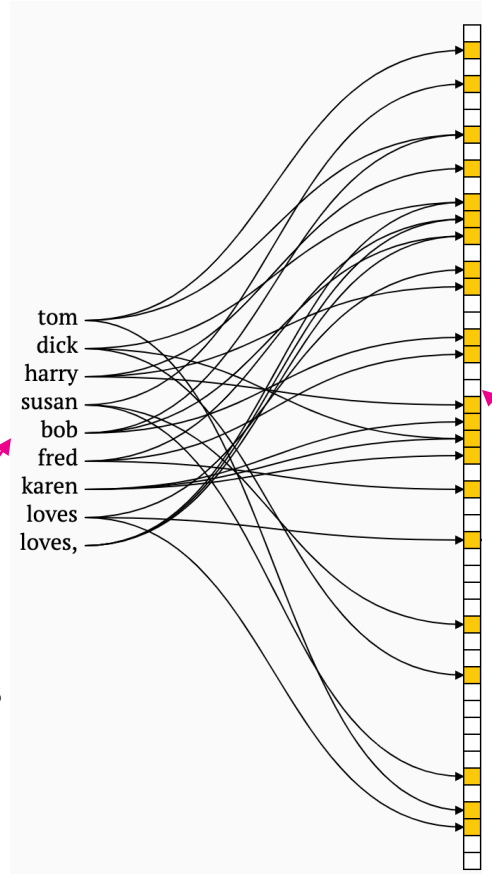
splunk> .conf20

Agenda

1. **Introduction**
   What this presentation is all about
2. **Search and workload elimination**
   How search works and where time is spent
3. **How the index is built**
   How universal indexing builds the lexicon
4. **Bloomfilter elimination**
   How bloomfilters accelerate _raw search
5. **Advanced indexing with Major breaker**
   How major breakers and turbo charge elimination
6. **Introducing `tstats`**
   How `tstats` delivers further performance improvements
7. **Other tricks and a performance dashboard**
   loadjob, base searches and take away dashboard

How Bloom Filters Eliminate Whole Buckets

Credit to the interactive tool:



The list of terms held in the lexicon

The output bit map for the list of TERMS

Bloom filters are a useful acceleration technology for evaluating set membership.

They are able to 100% accuracy in testing for the existence of terms, but less so for the absence.

The likelihood of false positives decreases as the size of the array is increased.

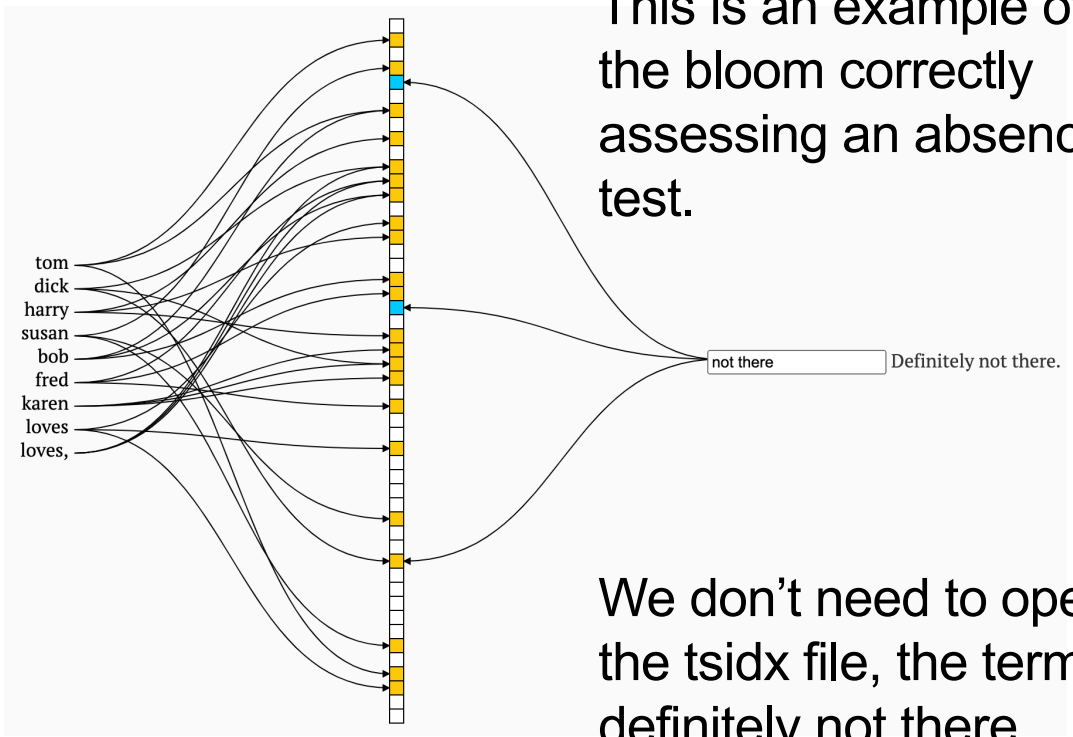
In the example we have loaded in the terms from our example lexicon and how they are translated to setting bits in the array.

Splunk auto tunes the size of the bloom filter to maintain a good balance between size and accuracy (often above 99%).

https://www.jasondavies.com/bloomfilter/

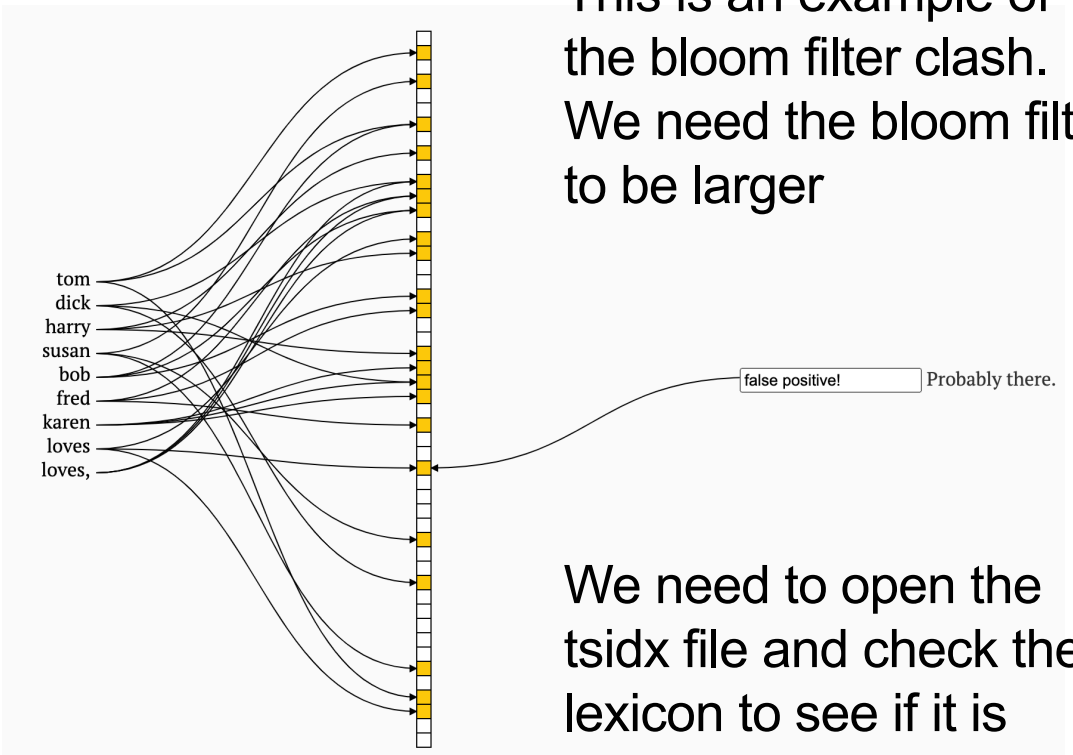
Looking Up Non-existent Terms

A positive false, and a false negative



This is an example of the bloom correctly assessing an absence test.

We don’t need to open the tsidx file, the term is definitely not there



This is an example of the bloom filter clash. We need the bloom filter to be larger

We need to open the tsidx file and check the lexicon to see if it is really there.

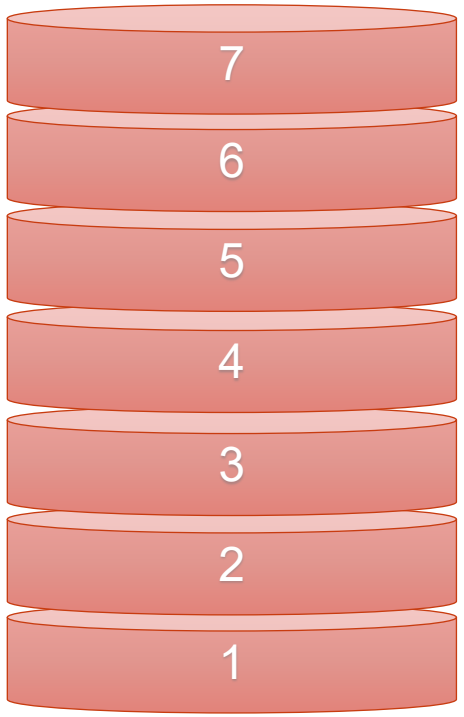
The Bucket \ Journal is Composed of Slices

The Postings list maps TERMS to locations into its associated bucket

TSIDX

TERM	Slices containing TERM
tom	1
rich	1,4
harry	1, 4
susan	3, 5, 6
bob	2
fred	2,3
karen	5, 6
loves	2,3,4,5,6

journal



The TSIDX file maps TERMS found in the lexicon to slices to decompress in the journal file.

Given these locations we can decompress the slices required and inspect the _raw string.

Note that the need to support slices is the reason bucket compression can use lz4, zstd and gzip, but will never support

A "Bucket" is a Directory

A bucket is a collection of files held in a directory structure; notable files highlighted

```
(base) rmorgan-mbp-4cb4b:splunk rmorgan$ ls -al search_demo/db/db_1596632603_1596618900_87/
total 17936
drwx--x---  16 rmorgan  wheel       512  7 Aug 20:03 .
drwx------   8 rmorgan  wheel       256 28 Aug 10:14 ..
-rw-------   1 rmorgan  wheel         8  7 Aug 20:03 .rawSize
-rw-------   1 rmorgan  wheel         7  7 Aug 20:03 .sizeManifest4.1
-rw-------   1 rmorgan  wheel    503929  7 Aug 20:03 1596620994-1596618900-4712026901567338950.tsidx
-rw-------   1 rmorgan  wheel   3727073  7 Aug 20:02 1596632603-1596620225-4538014197027015779.tsidx
-rw-------   1 rmorgan  wheel     57894  7 Aug 20:02 Hosts.data
-rw-------   1 rmorgan  wheel       118  7 Aug 20:02 SourceTypes.data
-rw-------   1 rmorgan  wheel       669  7 Aug 20:02 Sources.data
-rw-------   1 rmorgan  wheel   1429857  7 Aug 20:02 Strings.data
-rw-------   1 rmorgan  wheel    208669  7 Aug 20:03 bloomfilter
-rw-------   1 rmorgan  wheel        75  7 Aug 20:03 bucket_info.csv
-rw-------   1 rmorgan  wheel   2545204  7 Aug 20:03 merged_lexicon.lex
-rw-------   1 rmorgan  wheel        49  7 Aug 20:03 optimize.result
drwx------   5 rmorgan  wheel       160  7 Aug 20:03 rawdata
-rw-------   1 rmorgan  wheel        97  7 Aug 20:03 splunk-autogen-params.dat
(base) rmorgan-mbp-4cb4b:splunk rmorgan$ ls -al search_demo/db/db_1596632603_1596618900_87/rawdata/
total 1568
drwx------   5 rmorgan  wheel       160  7 Aug 20:03 .
drwx--x---  16 rmorgan  wheel       512  7 Aug 20:03 ..
-rw-------   1 rmorgan  wheel    773899  7 Aug 20:03 journal.zst
-rw-------   1 rmorgan  wheel       144  7 Aug 20:03 slicemin.dat
-rw-------   1 rmorgan  wheel      1200  7 Aug 20:03 slicesv2.dat
```

TSIDX files that point TERMS into slices found in the journal

A list of the hosts, sourcetypes and sources found in this bucket

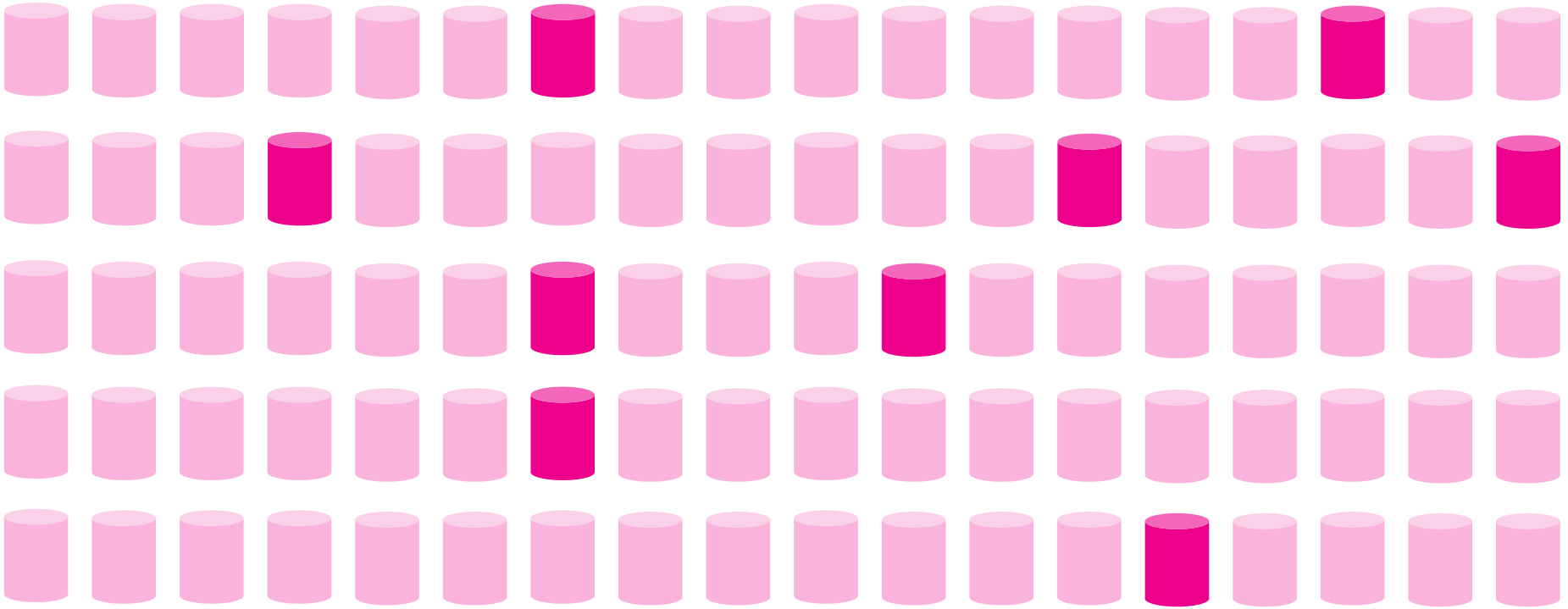
Bloomfilters are computed when buckets are closed

The journal file that contains the actual raw data compressed together

splunk> .conf20

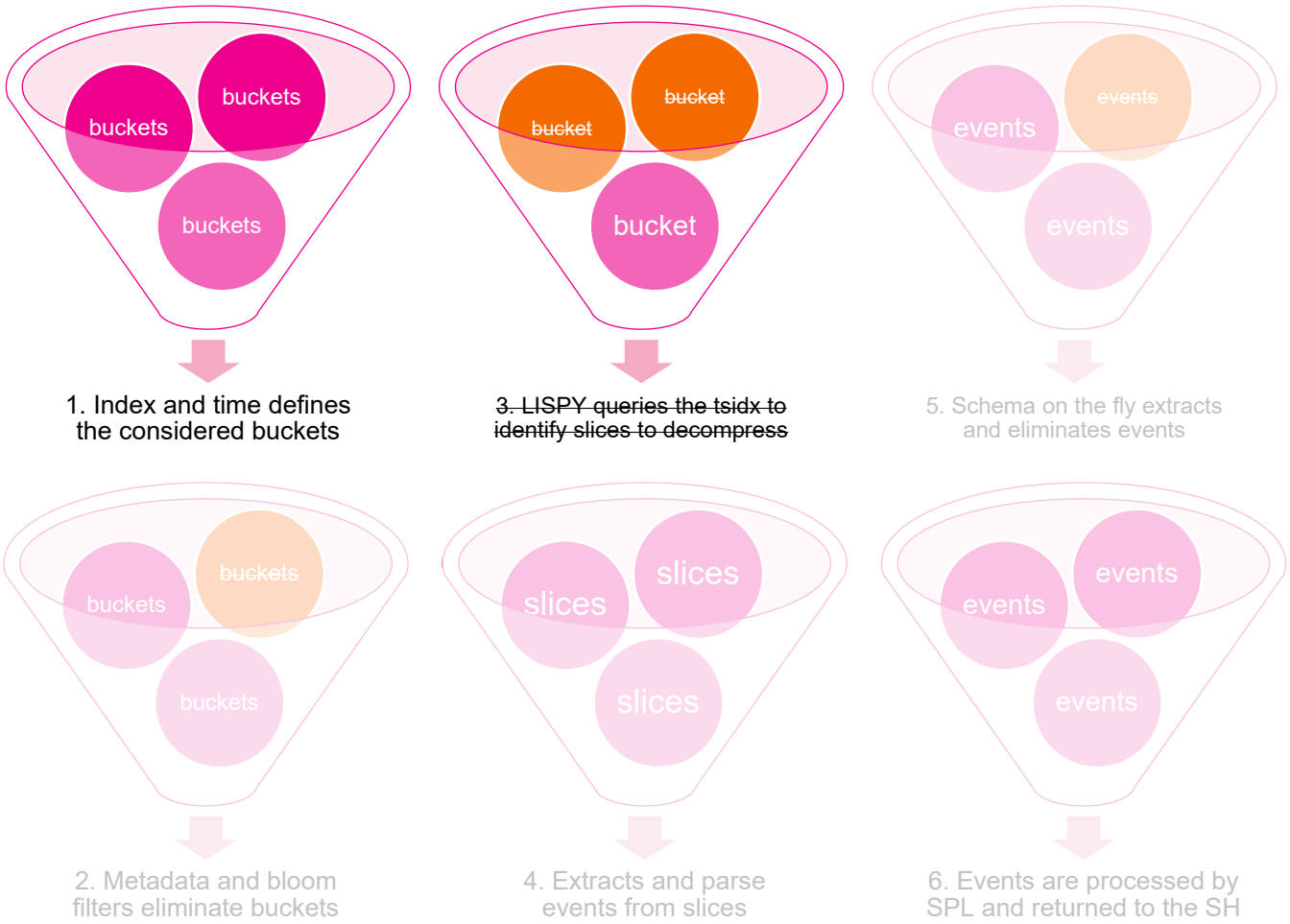
Eliminated buckets

Bloomfilters and metadata allows us to eliminate buckets early, avoiding work



`considered_buckets vs eliminated_buckets`

`tstats` Processes `tsidx` Files Only



The primary reason why tstats is so highly performant is that it works exclusively on the TSIDX files.

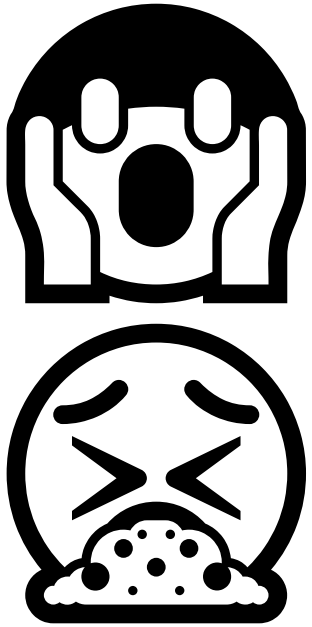
This means that it does no decompression or parsing, saving a huge amount of computation.

Unlike `_raw search` or `mstats` it doesn't support any bucket elimination.

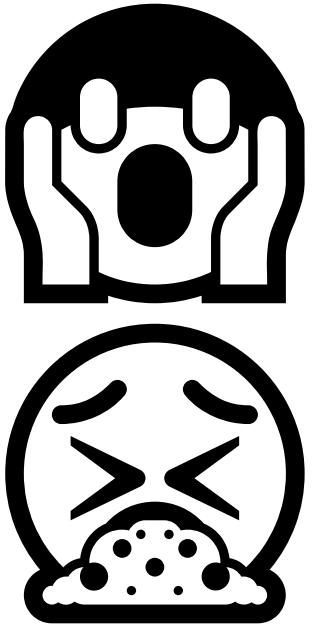
This is likely to feature in future releases.

TSIDX reduction is destroyer of performance

Deletes the tsidx files but keeps the bloomfilters, disables almost all work load elimination



```
enableTsidxReduction = <boolean>
* Whether or not the tsidx reduction capability is enabled.
* By enabling this setting, you turn on the tsidx reduction capability.
  This causes the indexer to reduce the tsidx files of buckets when the
  buckets reach the age specified  by 'timePeriodInSecBeforeTsidxReduction'.
* CAUTION: Do not set this setting to "true" on indexes that have been
  configured to use remote storage with the "remotePath" setting.
* Default: false
```



Just don’t do it!

Frozen Buckets Have No Metadata

WARNING: this slide only exists to justify the image!

Don’t tell Elisa

The freezing process removes the metadata from a bucket.

The journal file contains all the information required to rebuild the various metadata files.

This is how buckets are unfrozen.