



SPL Best Practices

January 2024

splunk>



Forward-looking statements

This presentation may contain forward-looking statements regarding future events, plans or the expected financial performance of our company, including our expectations regarding our products, technology, strategy, customers, markets, acquisitions and investments. These statements reflect management's current expectations, estimates and assumptions based on the information currently available to us. These forward-looking statements are not guarantees of future performance and involve significant risks, uncertainties and other factors that may cause our actual results, performance or achievements to be materially different from results, performance or achievements expressed or implied by the forward-looking statements contained in this presentation.

For additional information about factors that could cause actual results to differ materially from those described in the forward-looking statements made in this presentation, please refer to our periodic reports and other filings with the SEC, including the risk factors identified in our most recent quarterly reports on Form 10-Q and annual reports on Form 10-K, copies of which may be obtained by visiting the Splunk Investor Relations website at www.investors.splunk.com or the SEC's website at www.sec.gov. The forward-looking statements made in this presentation are made as of the time and date of this presentation. If reviewed after the initial presentation, even if made available by us, on our website or otherwise, it may not contain current or accurate information. We disclaim any obligation to update or revise any forward-looking statement based on new information, future events or otherwise, except as required by applicable law.

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. We undertake no obligation either to develop the features or functionalities described, in beta or in preview (used interchangeably), or to include any such feature or functionality in a future release.

Splunk, Splunk> and Turn Data Into Doing are trademarks and registered trademarks of Splunk Inc. in the United States and other countries. All other brand names, product names or trademarks belong to their respective owners.
© 2024 Splunk Inc. All rights reserved.

Please introduce Yourself!

- Name
- Role
- What is your SPL experience?
- What are your expectations?



Disclaimer

Before we begin, it's important to clarify that this workshop is designed to complement, not replace, formal Splunk Education. Our focus is on providing practical insights, hands-on experiences, and real-world applications. Consider it a supplementary, focused exploration to enrich your existing knowledge base. Let's make the most of this learning journey together, understanding that this workshop serves as a valuable addition to your educational experience.



Agenda for Today's Workshop

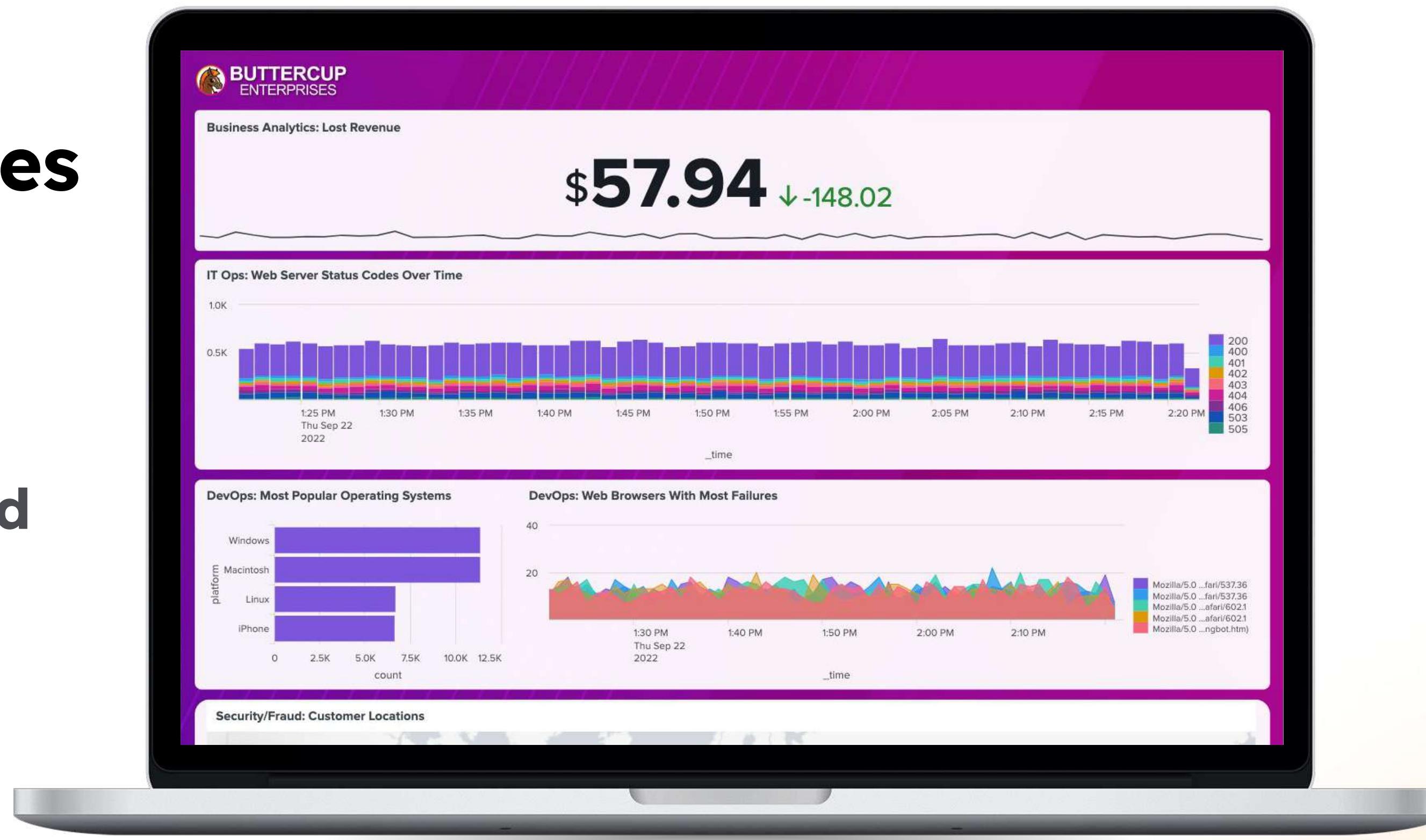
- ✓ Recap: Splunk's Search Processing Language (SPL)
- ✓ Tips & Tricks for Writing More Efficient Searches
- ✓ SPL 201 > Getting a Little More Advanced
- ✓ Tackling Difficult Data > Nested JSON
- ✓ When you lose your path, there's SPATH
- ✓ If Everything Fails, there's REX
- ✓ Resources - Lots of Resources!



Splunk4Rookies Workshop



Where we finished
last time





Today's Scenario

You are a Splunk Power User at Buttercup Enterprise

Your colleagues are new to Splunk. They need your help to write better searches and deal with complex datasets that the systems generates

Your task is to help your colleagues in getting more value out of Splunk.

Note:

All of the workshop searches
are provided within an app
installed in the lab
environment



Splunk4Ninjas - SPL Best Practices

Splunk4Ninjas - SPL Best Practices Resources ▾

Each search is clickable and will open in a new tab in the Search app

SPL Commenting

```
sourcetype="access_combined" action=purchase  
```| stats count by product_name, category```
```

### SPL#1 > Successful Purchase Actions for an IP

```
sourcetype=access_combined action=purchase status=200 125.17.14.100
```

### SPL#2 > Exploring the TERM directive

```
sourcetype=access_combined action=purchase status=200 TERM(125.17.14.100)
```

### SPL#3 > tstats with PREFIX

classic search using stats

```
index=_internal 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
```

### tstats search

```
|tstats sum(PREFIX(kb=)) AS indexer_kb avg(PREFIX(instantaneous_kbps=)) AS
load_avg
WHERE
index=_internal
TERM(group=thruput)
BY host _time
span=1767s
```

# Recap: Search Processing Language (**SPL**)

splunk®



# SPL (Search Processing Language) Refresher

Events are retrieved

Results passed linearly through SPL commands for processing

Search Terms

index=main action=purchase

Pipe character: Output of left is input to right

e.g. index=main action=purchase

| Time                    | Event                                                                                                                                                                                                                                                                                                                                                                                                                                                                                                   |
|-------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 15/09/2022 09:12:53.163 | 12.130.60.5 - - [15/Sep/2022 09:12:53:163] "GET /product.screen?product_id=MCB-5&JSESSIONID=SD4SL3FF10ADFF4 HTTP 1.1" 401 3810 "http://www.buttercupenterprises.com/cart.do?action=purchase&itemId=EST-27&product_id=MCB-5" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_2) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/55.0.2883.95 Safari/537.36" 259 host = ip-172-31-39-95   source = /var/log/weblogs/noise_apache_2.log   sourcetype = access_combined                                         |
| 15/09/2022 09:12:48.184 | 128.241.220.82 - - [15/Sep/2022 09:12:48:184] "GET /cart.do?action=purchase&itemId=EST-21&product_id=ZSG-2&JSESSIONID=SD4SL5FF3ADFF10 HTTP 1.1" 404 2946 "http://www.buttercupenterprises.com/product.screen?product_id=ZSG-2" "Mozilla/5.0 (iPhone; CPU iPhone OS 7_0 like Mac OS X) AppleWebKit/537.51.1 Version/7.0 Mobile/11A465 Safari/9537.53 BingPreview/1.0b" 661 host = ip-172-31-39-95   source = /var/log/weblogs/noise_apache_3.log   sourcetype = access_combined                          |
| 15/09/2022 09:12:42.194 | 141.146.8.66 - - [15/Sep/2022 09:12:42:194] "POST /cart.do?action=purchase&itemId=EST-19&product_id=MCB-5&JSESSIONID=SD3SL4FF10ADFF9 HTTP 1.1" 505 3349 "http://www.buttercupenterprises.com/cart.do?action=purchase&itemId=EST-19&product_id=MCB-5" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_1) AppleWebKit/537.36 Chrome/56.0.2914.3 Safari/537.36 OPR/43.0.2431.0 (Edition developer)" 801 host = ip-172-31-39-95   source = /var/log/weblogs/noise_apache_1.log   sourcetype = access_combined |
| 15/09/2022 09:12:42.176 | 201.3.120.132 - - [15/Sep/2022 09:12:42:176] "POST /cart.do?action=purchase&itemId=EST-16&product_id=MCF-3&JSESSIONID=SD3SL7FF3ADFF3 HTTP 1.1" 200 3542 "http://www.buttercupenterprises.com/product.screen?product_id=MCF-3" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_1) AppleWebKit/537.36 Chrome/57.0.2959.0 Safari/537.36" 236                                                                                                                                                                 |

Commands

| stats count by status | rename count as "number of events"

Functions

| stats count by status

| status | count |
|--------|-------|
| 200    | 850   |
| 400    | 81    |
| 401    | 76    |
| 402    | 50    |
| 403    | 57    |

Clause

| rename count as "number of events"

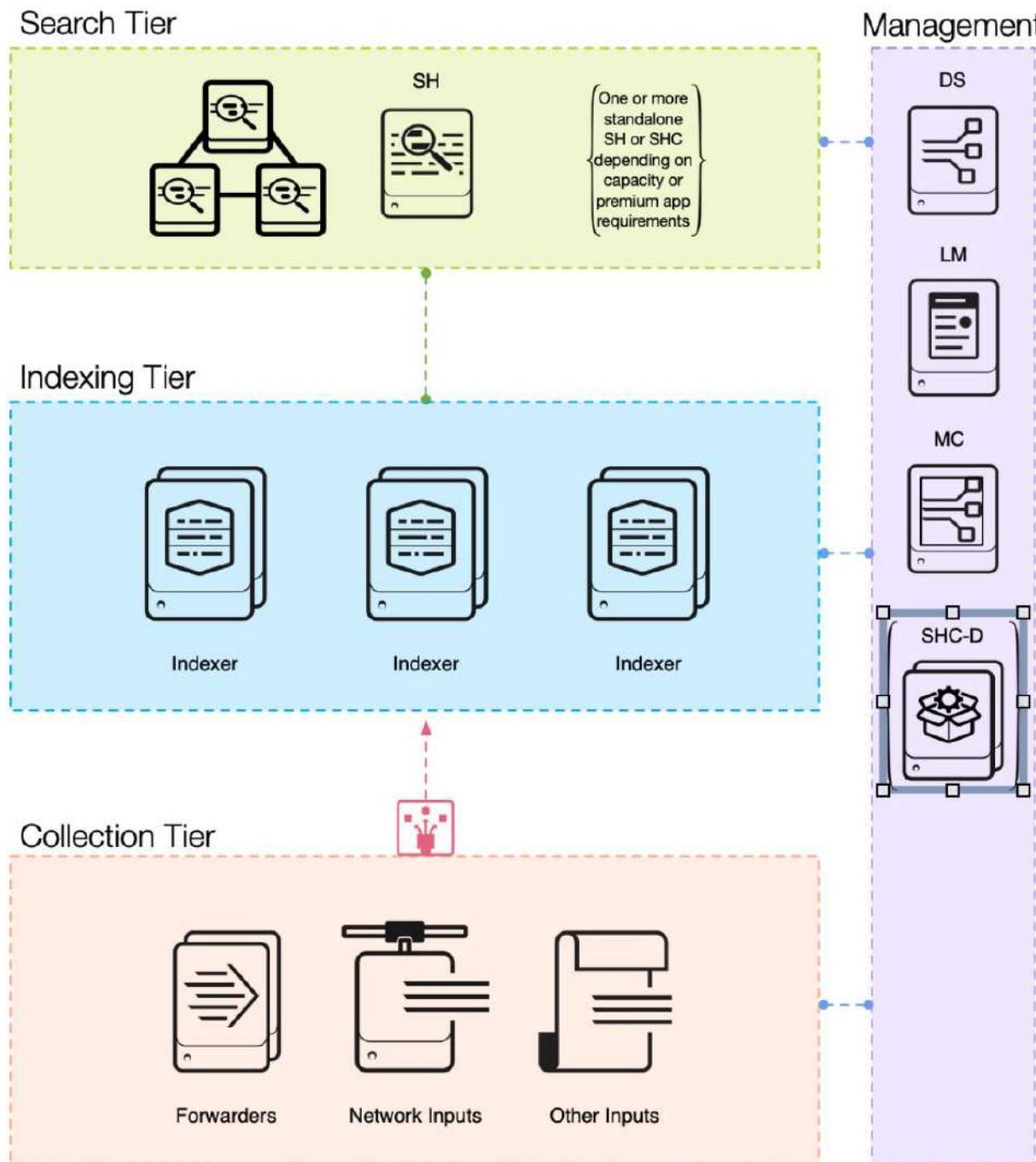
| status | number of events |
|--------|------------------|
| 200    | 850              |
| 400    | 81               |
| 401    | 76               |
| 402    | 50               |
| 403    | 57               |

Want to know more? Check out:

Splunk Quick Reference Guide: <https://splk.it/SplunkQuickRef>

Search manual: <https://splk.it/SplunkSearchManual>

# A quick word about architecture



A typical Splunk implementation uses a tiered architecture with both vertical and horizontal scalability.

Today we'll focus on the “**scale out**” part, since this will also influence how we build our “**search pipeline**”.

As seen previously, the search pipeline works in a sequential way: **...|...|...| -> results**

Although it works fine we can still make things a tiny bit **faster** if we take advantage of the underlying parallelism of the Indexing Tier.

Learn more: [https://www.splunk.com/en\\_us/pdfs/tech-brief/splunk-validated-architectures.pdf](https://www.splunk.com/en_us/pdfs/tech-brief/splunk-validated-architectures.pdf)

# SPL Command Types

## Focus of Today's Workshop

Streaming

Transforming

Generating

Orchestrating

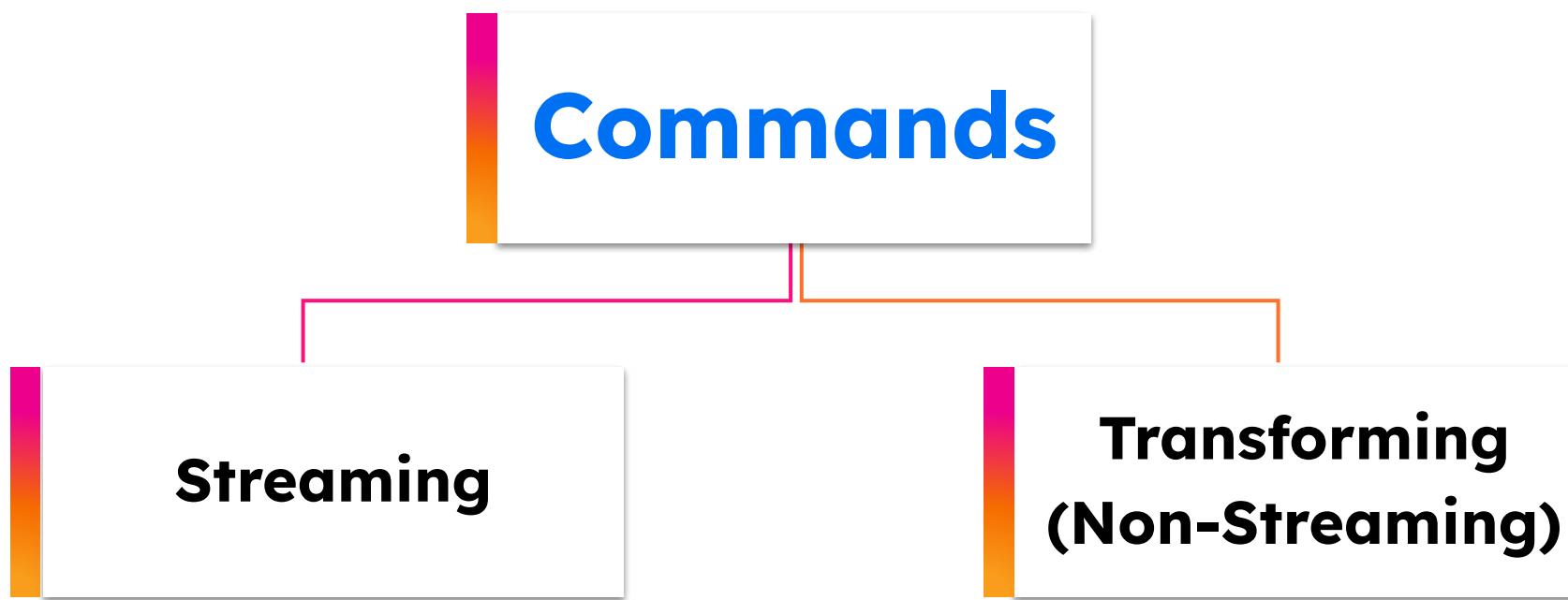
Dataset  
processing

Distributable

Centralized

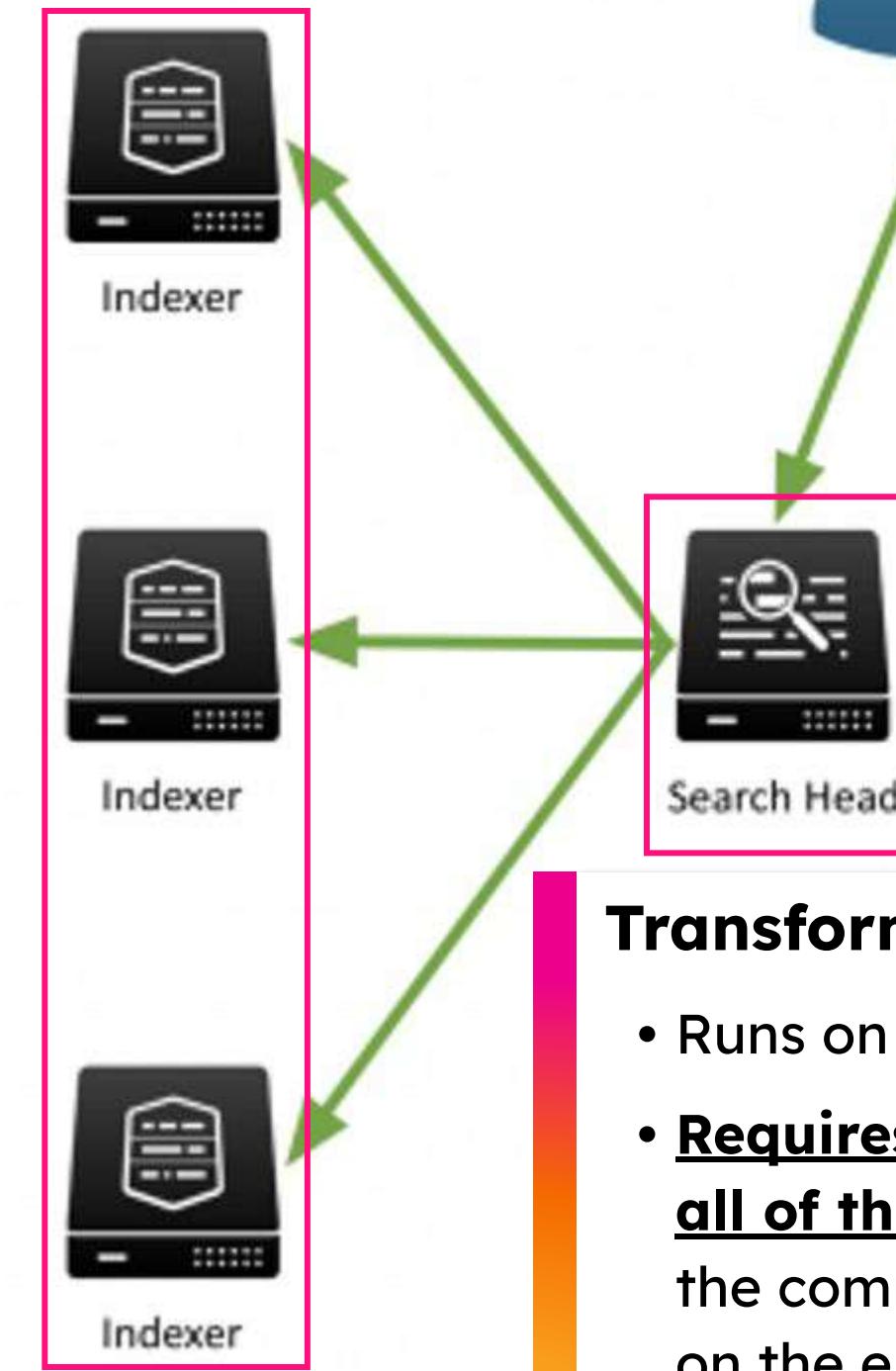
Learn more: <https://docs.splunk.com/Documentation/Splunk/latest/Search/Typesofcommands>

# Streaming and Transforming Commands



**Streaming Commands**

- Runs on the indexers
- Can be applied to subsets of index data in *parallel*



**Transforming Commands**

- Runs on the Search Head
- Requires the events from all of the indexers before the command can operate on the entire set of events.

# Comparison

## Streaming Commands

---

**eval**  
**makemv**  
**rex**  
**spath**

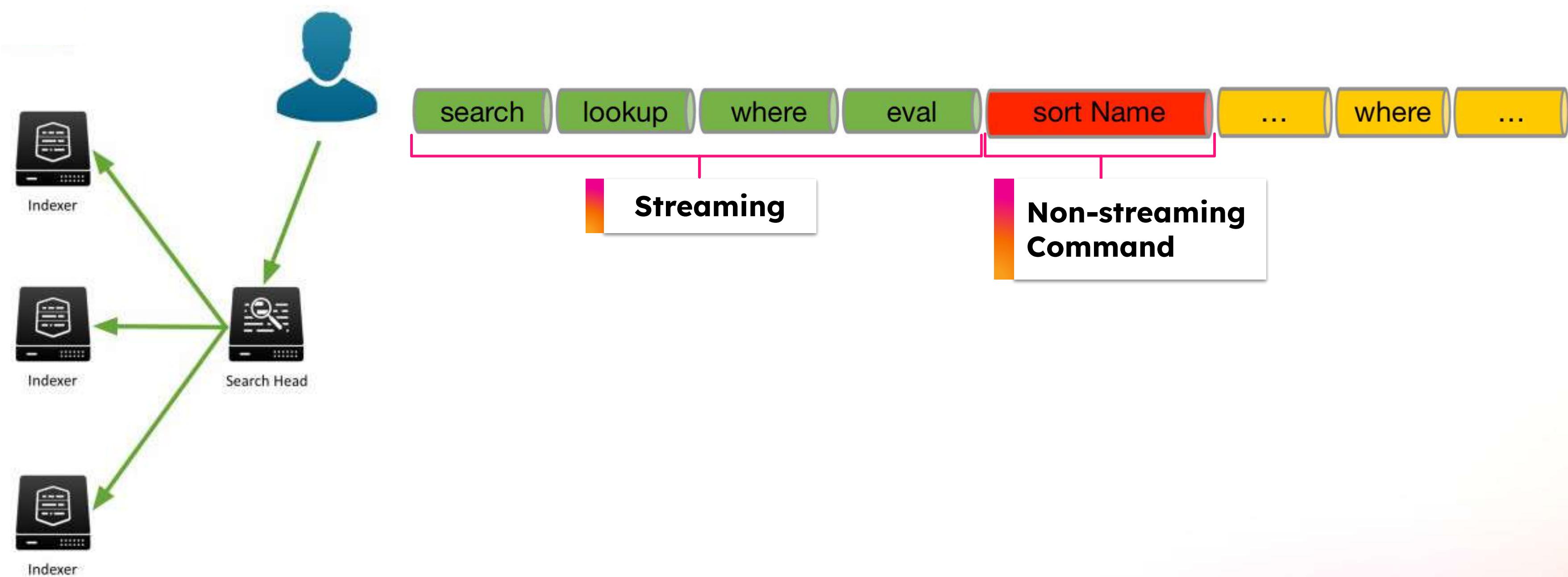
## Transforming Commands

---

**chart**  
**timechart**  
**stats**  
**top**  
**rare**

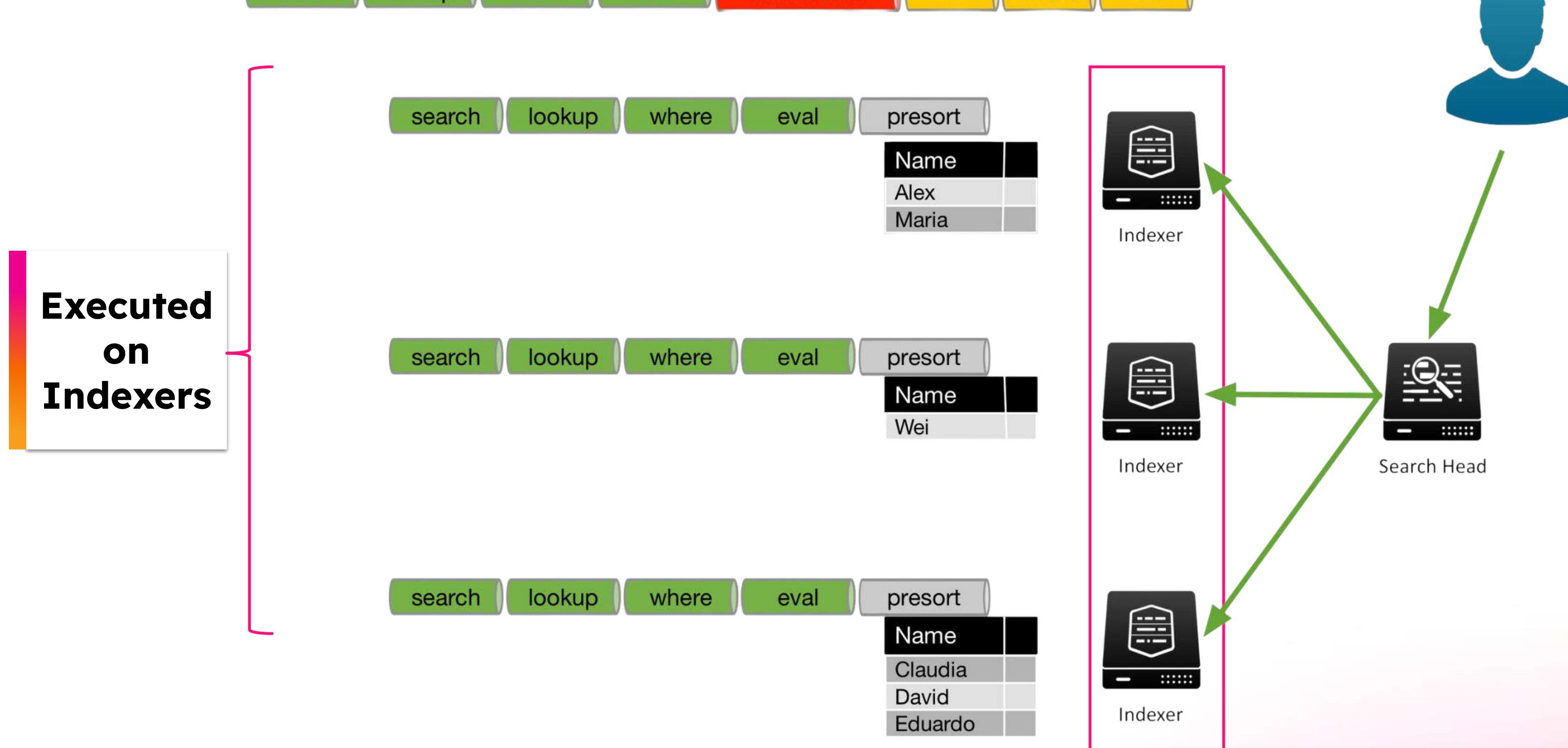
# Search Processing Example

User search > Search Head > Distributed to Indexers



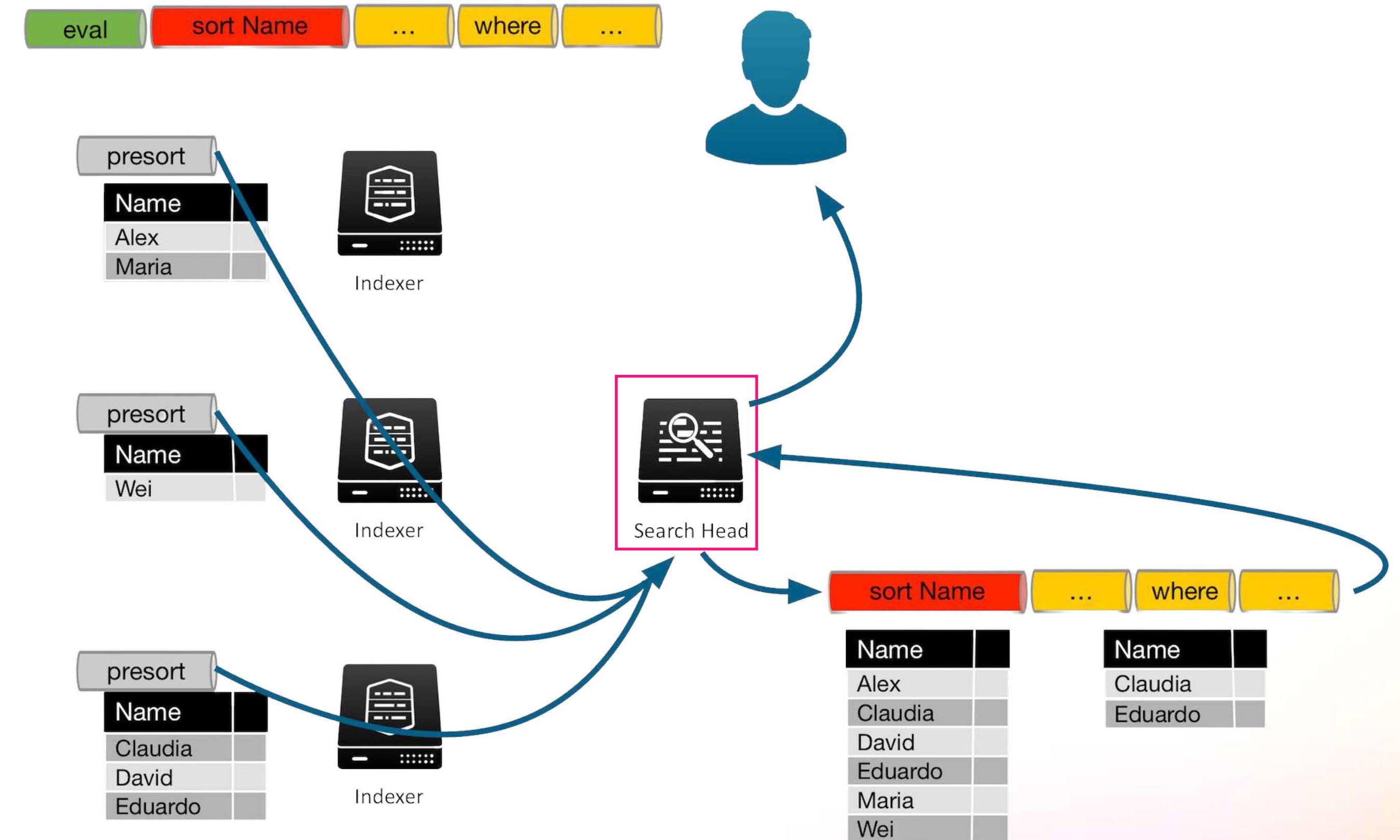
Learn more: <https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Sort#Usage>

# Search Processing Example



# Search Processing Example

- Results sent to **Search Head** > run **sort** on the **ENTIRE dataset**.
- **Following commands** run on **Search Head**.



Learn more: <https://docs.splunk.com/Documentation/Splunk/latest/Search/Writebettersearches>

# Tips for Writing Better Searches

## #1: The Basics



- **Keywords:** Search for a single word (e.g., error) or group of words (e.g., error password)
- **Booleans:** **NOT**, **OR**, **AND**
  - AND** is implied - **MUST** be uppercase
  - can use ()'s to force precedence, e.g.  
`sourcetype=vendor_sales OR  
(sourcetype=access_combined action=purchase)`
- **Phrases:** "web error" (different than web **AND** error)
- **Field Searches:** status=404, user=admin
- **Wildcard(\*):** status=40\* matches 40, 40a, 404, etc; starting keywords with a wildcard is very inefficient
- **Comparisons:** =,!,<=,>=,<,>
  - e.g. status>399, user!=admin

Learn more:

<https://docs.splunk.com/Documentation/Splunk/latest/Search/Quicktipsforoptimization>

# Tips for Writing Better Searches #2: Advanced

## > Reduce the amount of data Splunk must Search

- Specify and limit the **index(es)** and other meta-fields like **host(s)**, **source(s)** and **sourcetype(s)**
- Limit the **time range** for searching
- Fine-tune your searches to your **unique events** as much as possible
- Reduce the **number of fields** being passed down the SPL pipeline for processing (use **fields** command)

## > Distributed Search

- Place streaming commands earlier in the pipeline



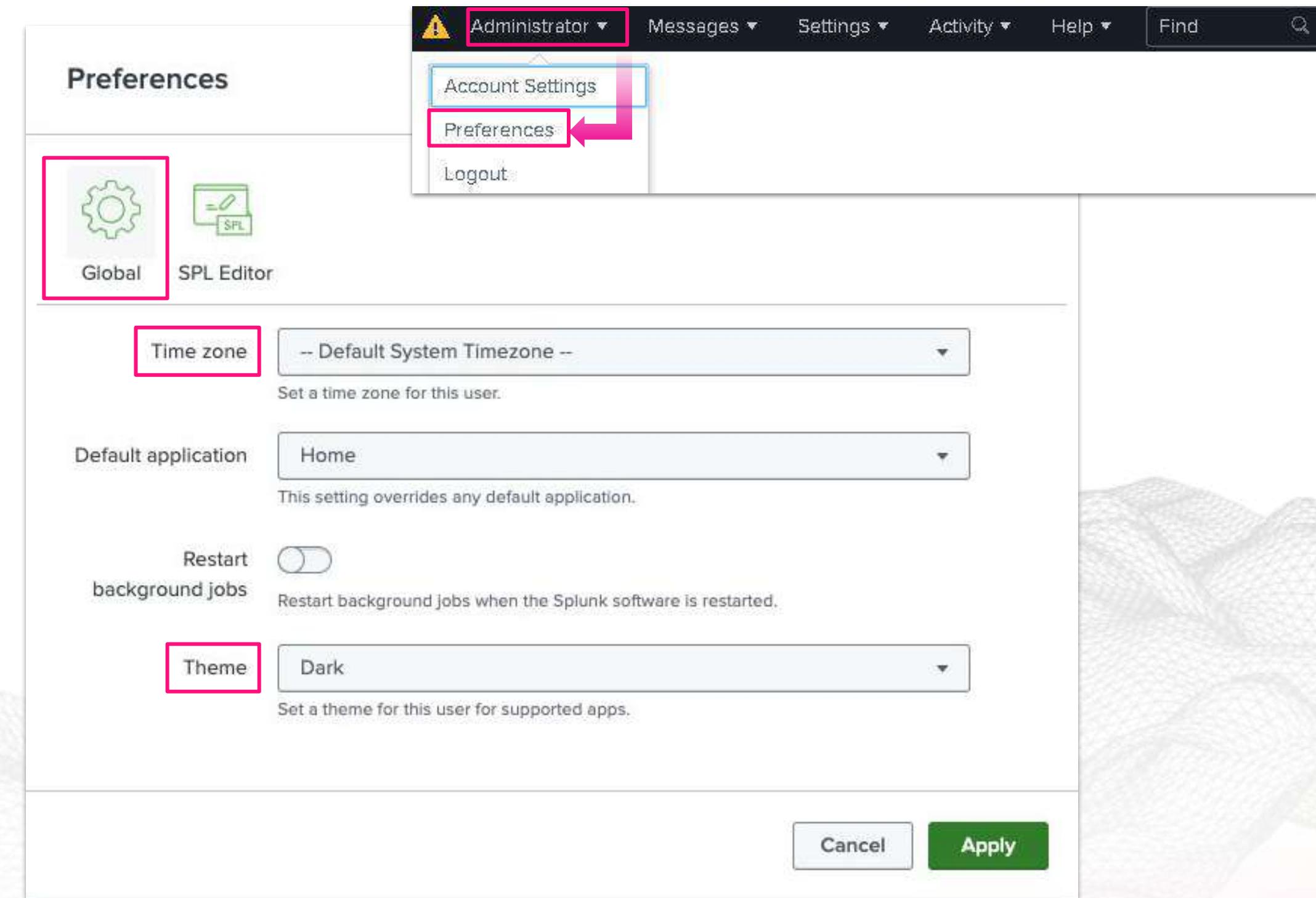
Be as specific as possible within search

Overrides the time-picker

```
index=my_index host=my_host sourcetype=my_srctype field1=val1 raw_text_search earliest=-1h latest=now
| eval field2=do_some_eval_here } ← Streaming commands used earlier in the search pipeline
| rename field2 as new_field2 } ← Reduce number of fields in pipeline
| fields field1, new_field2 } ← Transforming commands used later in the search pipeline
| stats count(field1) by new_field2 }
```

# Initial Set-up

- It's important to set-up your **timezone**
- Dark theme can be selected for **supported Apps**



Learn more: <https://docs.splunk.com/Documentation/Splunk/latest/Search/Abouttimezones>

# UI Improvements

The screenshot illustrates the SPL Editor Preferences dialog, which has been updated with several user interface improvements. A large double-headed arrow in the center indicates the transition between the old and new versions of the dialog.

**Old Version (Left):**

- Header:** The title bar includes the "Administrator" dropdown, "Messages", "Settings", "Activity", "Help", and a "Find" search bar.
- Navigation:** A vertical sidebar on the left lists "Account Settings", "Preferences" (which is highlighted with a red border), and "Logout".
- Content Area:** The main area displays the "Preferences" configuration screen. It features two tabs: "General" (selected) and "Themes".
- Search Assistant:** Includes settings for "Search assistant" (options: Full, Compact, None), with a note explaining the difference between Full and Compact modes.
- Line Numbers:** Includes a setting for "Line numbers" with a note about its purpose.
- Search Auto-format:** Includes a setting for "Search auto-format" with a note about its purpose.
- Buttons:** At the bottom are "Cancel" and "Apply" buttons.

**New Version (Right):**

- Header:** The title bar includes the "Administrator" dropdown, "Messages", "Settings", "Activity", "Help", and a "Find" search bar.
- Navigation:** The sidebar is no longer present; instead, the "Preferences" link is now part of the top-level navigation menu under "Account Settings".
- Content Area:** The main area displays the "Preferences" configuration screen. It features two tabs: "General" and "Themes" (selected).
- Search Assistant:** Includes settings for "Search bar theme" (options: Black on White, Light Theme, Dark Theme, selected).
- Preview:** A preview window shows SPL search syntax with syntax highlighting.
- Buttons:** At the bottom are "Cancel" and "Apply" buttons.

# SPL Commenting

- Use three backticks (```) before and after your comment
- Comment out portions of your search to help identify and isolate problems
- To make very long SPL easier to read, add comments directly after the pipe (|)

```
index=security sourcetype=linux_secure
| ```single-series column chart```
chart count over vendor_action
```

```
index=security sourcetype=linux_secure
```| single-series column chart  
chart count over vendor_action```
```

```
index=security "failed password" earliest=-14d@d latest=@d  
| ```line chart with week-to-week comparison```  
timechart span=1d count as Failures  
timewrap 1w  
rename _time as Day  
eval Day = strftime(Day, "%A")
```

Example:

Search

```
sourcetype=access_* status=200 ```Get all successful website access events.  
| stats count AS views count(eval(action="addtocart")) AS addtocart count(eval(action="purchase")) AS purchases by productName  
    ```Create counts of site views, add-to-cart actions, and purchase actions. Break them out by product name.  
| eval viewsToPurchases=(purchases/views)*100 ```Find the ratio of site views to purchases.
| eval cartToPurchases=(purchases/addtocart)*100 ```Find the ratio of add-to-cart actions to purchases.
| table productName views addtocart purchases viewsToPurchases cartToPurchases ```Put all this data into a table.
| rename productName AS "Product Name", views AS "Views", addtocart as "Adds To Cart", purchases AS "Purchases" ```Rename some table
columns.
```

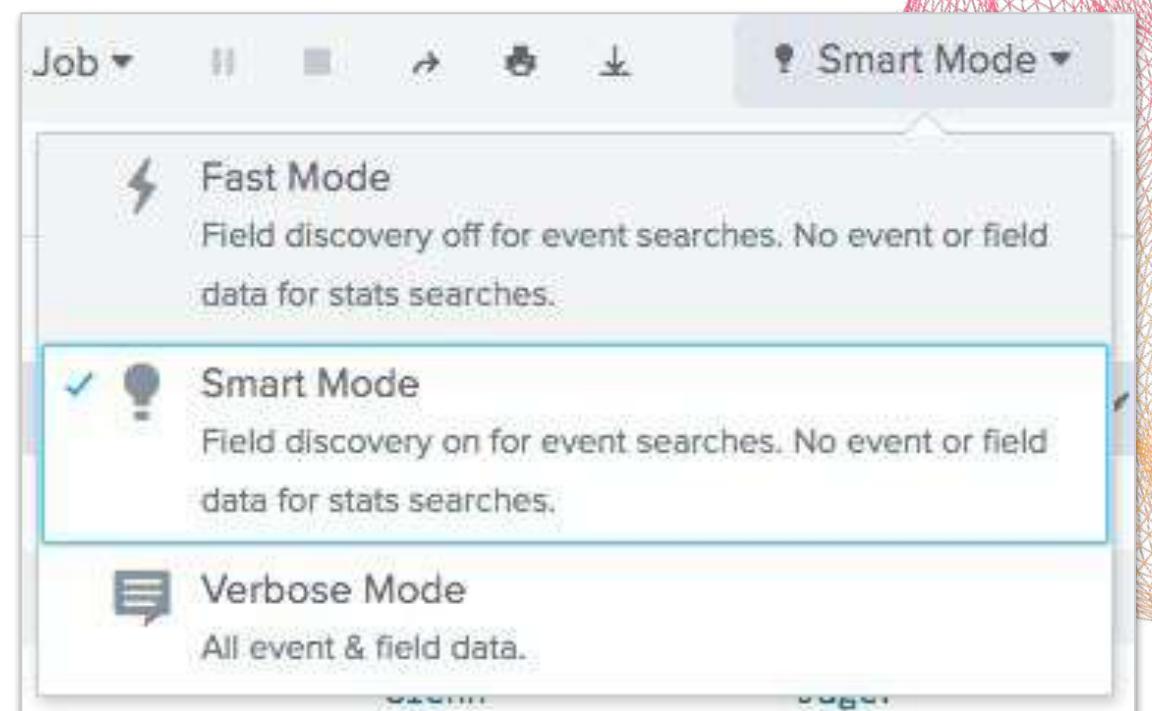
### Useful for:

- Explaining each "**step**" of a complicated search that is shared with other users.
- Discussing ways of **improving** a search with other users.
- Leaving **notes** for yourself in unshared searches that are works in progress.
- **Troubleshooting** searches by running them with chunks of SPL "**commented out**".

# Splunk Search Modes

## Fast vs. Smart vs. Verbose

- **Fast:** emphasizes speed over completeness
- **Smart:** balances speed and completeness (default)
- **Verbose:** emphasizes completeness

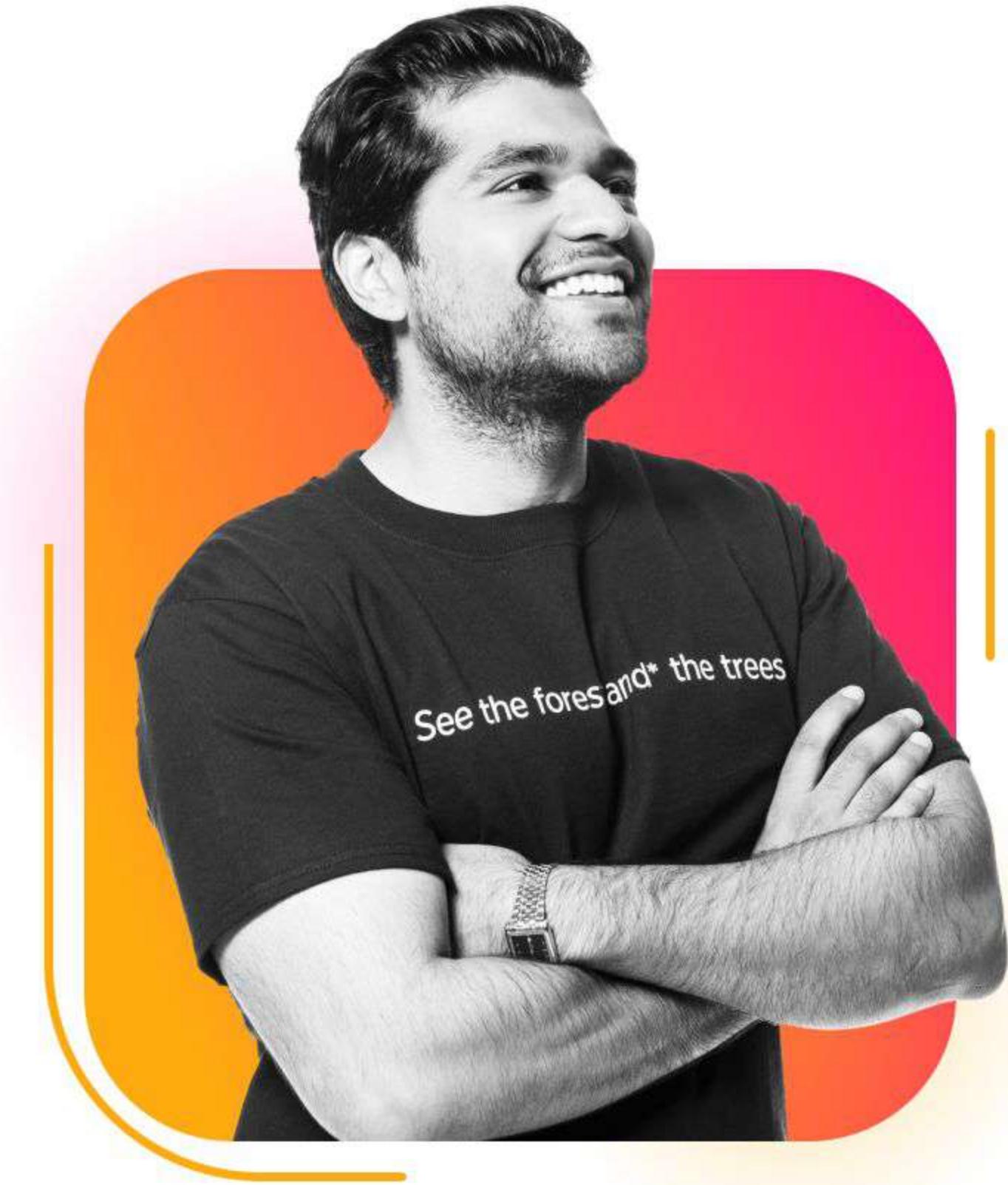


Learn more:

<https://docs.splunk.com/Documentation/Splunk/latest/Search/Changethesearchmode>

# Tips & Tricks for Writing More Efficient Searches

splunk®



**SPL#1 >**

**Successful**

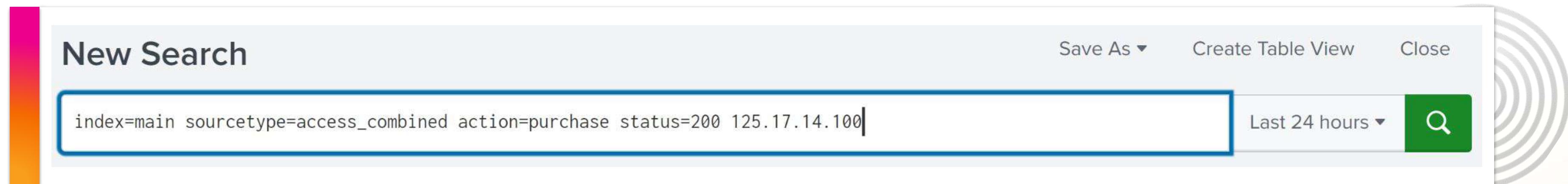
**Purchase for an**

**IP**



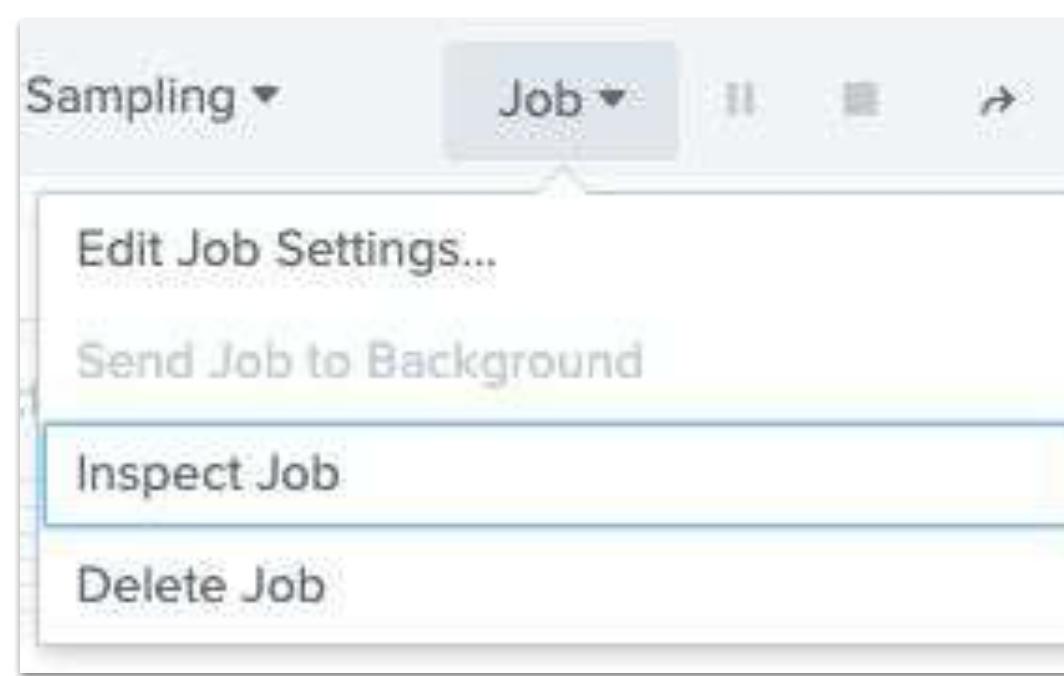
# SPL#1 > Successful Purchase Actions for an IP address

Let's take a very common use case: Finding out how many successful actions were performed by the address 125.17.14.100 during the last hour.



# Measure the Search Execution Time

Inspect Job > check the Search Summary



Search job inspector

This search has completed and has returned **397** results by scanning **931** events in **0.205** seconds  
(SID: 1621398493.191) [search.log](#)

Check search duration

Execution costs

| Duration (seconds) | Component                    | Invocations | Input count | Output count |
|--------------------|------------------------------|-------------|-------------|--------------|
| 0.00               | command.fields               | 6           | 397         | 397          |
| 0.07               | command.search               | 6           | -           | 397          |
| 0.01               | command.search.expand_search | 2           | -           | 7            |
| 0.00               | command.search.calcfields    | 3           | 931         | 931          |

# Raw Text Searches in Splunk are great!

Easy to start off... but are they efficient in huge datasets??

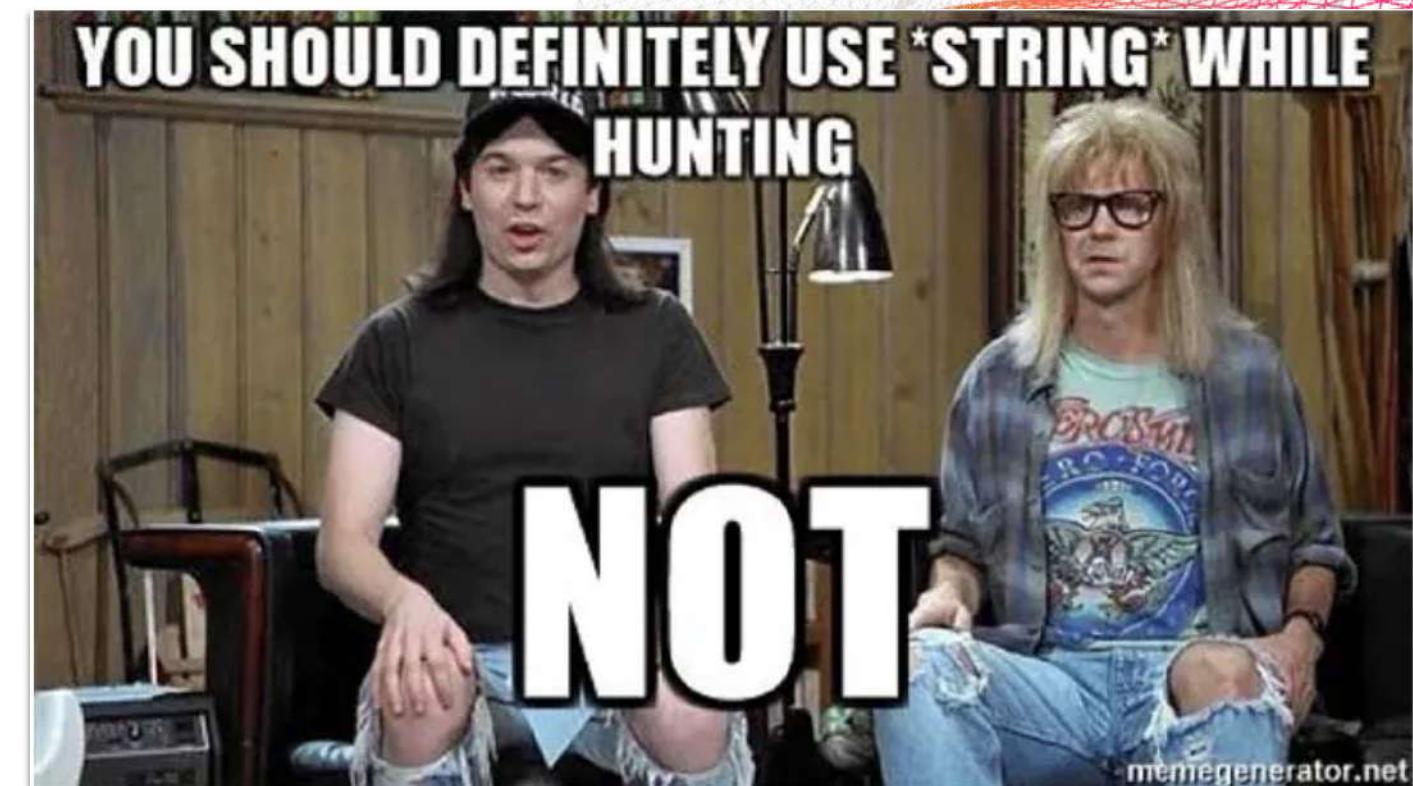
Let's see what happens when we ask Splunk to Search for an IP address.

## Pros:

- ✓ Very easy to build searches.
- ✓ Use raw **text** and **keywords** that you want to find in data.
- ✓ Use **wildcards** to find multiple patterns.
- ✓ Build logic using **AND/OR** and brackets ()

## Cons:

- ✗ Inefficient when searching huge datasets.
- ✗ Applied on Big datasets leads to slower outputs.



# What Happens on Data Ingestion?

**Event segmentation – data is broken into smaller segments for indexing**

- > Data is segmented by **separating terms** into smaller pieces
- > First with **major breakers** and then with **minor breakers**

Raw Data

91.205.189.15 -- [13/Aug/2019:18:22:16]

**Major segment breaker**  
(space is the breaker)

91.205.189.15

-

-

13/Aug/2019:18:22:16

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

91  
205  
189  
15  
91.205  
91.205.189  
91.205.189.15  
205.189  
and so forth

**Minor segment breaker**  
**dot (.)** breaks it further



# How is Data being Stored ?

## LEXICON

| Raw Events      |  |
|-----------------|--|
| 10.0.0.6        |  |
| 9/28/2016       |  |
| jeff@splunk.com |  |



| Term            | Postings List |
|-----------------|---------------|
| 0               | 0             |
| 6               | 0             |
| 9               | 1             |
| 10              | 0             |
| 28              | 1             |
| 2016            | 1             |
| 10.0.0.6        | 0             |
| 9/28/2016       | 1             |
| com             | 2             |
| jeff            | 2             |
| splunk          | 2             |
| jeff@splunk.com | 2             |

# Searching for Raw data

What happens when search string has Segment Breakers

New Search

index=main sourcetype=access\_combined action=purchase status=200 125.17.14.100

All time 🔍

✓ 12,729 events (before 1/8/24 3:49:51.000 PM) No Event Sampling

Job ▾

Events (12,729) Patterns Statistics Visualization

Format Timeline ▾ - Zoom Out + Zoom to Selection × Deselect

Edit Job Settings...  
Send Job to Background  
Inspect Job  
Delete Job

Search job inspector

This search has completed and has returned 14 results.

(SID: 1622461331.83) [search.log](#)

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

UnifiedSearch - Expanded index search = (status=200 125.17.14.100)  
UnifiedSearch - base lispy: [ AND 100 125 14 17 200 ]

Minor Segment (.)  
It breaks the key word to look for these values with **AND** condition.

Learn mode: <https://docs.splunk.com/Documentation/Splunk/latest/Search/Eventsegmentationandsearching>

TSTATS and PREFIX: <https://conf.splunk.com/files/2020/slides/PLA1089C.pdf>

**SPL#2 >**

# Exploring the **TERM** directive



# The TERM directive is an **easy** way to improve search...



The TERM() directive is useful for finding **terms** more efficiently, when:

- They **contain minor breakers**
- Are **bounded by major breakers**, such as spaces or commas
- They **don't contain major breakers**

# How can I use it?

It is **quite easy** to try, although not necessarily intuitive at first.  
Trial and error **will** be needed.

**Here is our previous search modified to include TERM for the IP address**

## New Search

```
index=main sourcetype=access_combined status=200 TERM(125.17.14.100)
```

**Let's run it again and see how it performs**

**When to use TERM:**

[https://docs.splunk.com/Documentation/Splunk/latest/Search/UseCASEandTERMtomatchphrases#When\\_to\\_use\\_TERM](https://docs.splunk.com/Documentation/Splunk/latest/Search/UseCASEandTERMtomatchphrases#When_to_use_TERM)

# Measure again, compare results

The Job Inspector is our friend

Inspect Job > Check the Search Summary, then head to the search.log & look for the “base lousy” line

New Search

index=main sourcetype=access\_combined action=purchase status=200 TERM(125.17.14.100)

Save As ▾ Create Table View Close

All time ▾ 🔍

✓ 12,729 events (before 1/8/24 3:49:51.000 PM) No Event Sampling ▾

Events (12,729) Patterns Statistics Visualization

Format Timeline ▾ - Zoom Out + Zoom to Selection × Deselect

Job ▾

Edit Job Settings...  
Send Job to Background  
**Inspect Job**  
Delete Job

Search job inspector

This search has completed and has returned 14 events.

(SID: 1622461331.83) [search.log](#)

UnifiedSearch - Expanded index search = (status=200 TERM(125.17.14.100))

UnifiedSearch - base lousy: [ AND 125.17.14.100 200 ]

Minor Segment (.)  
breakers are not used.  
The **Exact string** is  
searched in the data.

# Do's and Don't's of the TERM directive

Your term **must** be bounded by major segmenters.

- E.g. **spaces, tabs, carriage returns**.
- See **segmenters.conf** for the complete list.

Your term **cannot** contain major segmenters.

## Few Major Breakers:

- A space
- A newline
- A tab
- Square brackets [ ]
- Parenthesis ( )
- Curly brackets { }
- An exclamation point !
- A question mark ?
- A semicolon ;
- A comma ,
- Single and double quotation marks ''
- The ampersand sign &

```
ip 10.0.0.6 - 807256800 GET /images/launchlogo.gif
```



```
ip=TERM(10.0.0.6)
```

```
ip=10.0.0.6 - 807256804 GET /shuttle/missions.html
```



```
TERM(ip=10.0.0.6)
```

```
ip10.0.0.6 - 807256944 GET /history/history.html
```



```
TERM(ip10.0.0.6)
```

```
10.0.0.6:80 - 807256966 GET /skylab/skylab-4.html
```



```
TERM(10.0.0.6*)
```

```
9/28/16 1:30 PM - name=Willy Wonka sex=m age=46
```



```
TERM("Willy Wonka")
```

## Learn more:

> segmenters.conf: <https://docs.splunk.com/Documentation/Splunk/latest/Admin/Segmentersconf>

> CASE and TERM: <https://docs.splunk.com/Documentation/Splunk/latest/Search/UseCASEandTERMtomatchphrases>

# Task using SPL#2 > Explore TERM Directive

## Some more examples to try out



### Task

Exploring the usage of TERM directive

Try the searches and investigate/discuss the following:

- 1) Are all of these 3 searches going to **work**?
- 2) In **search.log** find “**base lispy**” and review the **output**.

a) TERM(131.178.233.243)

b) TERM(status=200)

c) product\_id=TERM(MCB-6)

# Search Optimization - Wrap Up

We have learned today some **general principles** behind **efficient searches**:

- Retrieve only the required data.
- Move as little data as possible.
- Parallelize as much work as possible.
- Pick appropriate time frames.

**Within the search, we want to:**

- Filter as much as possible in the initial search.
- Perform joins and lookups on only the required data.
- Perform evaluations on the minimum number of events possible.
- Position commands that bring data to the search head towards the end of your search.

**Reference:**

<https://docs.splunk.com/Documentation/Splunk/latest/Search/Aboutoptimization>

# The “optimizedSearch”

Behind the scenes, Splunk also applies built-in optimizations that analyze and process searches for maximum efficiency.

The built-in optimizations can reorder search processing, so that as many commands as possible are run in parallel on the indexers before sending the search results to the search head for final processing.

While this helps up to some point, the built-in optimizations won't be able to fix the logic of an inefficient search.

But we can still have a look at what the **optimized search** looks like, and re-use this to improve our initial search.

You can see the reordered search using  
the **Job Inspector>Search Job Properties >>>>**

**Reference:**

<https://docs.splunk.com/Documentation/Splunk/latest/Search/Built-inoptimization>

## Search job inspector

This search has completed and has returned 1 results by scanning 332 events in 0.152 seconds

(SID: 1704991313.1215) [search.log](#) [Job Details Dashboard](#)

> Execution costs

> Search job properties

# The optimizedSearch - Example

The original search - it could be better!

```
index=main sourcetype=access_combined
| stats count as sales sum(product_price) as revenue by product_name
action status
| search product_name="Batguy Slippers"
| search action=purchase
| search status=200
```

Job Inspector >  
Search Job Properties

Locate the optimized search in the Job Properties, copy and paste in a new search

```
optimizedSearch
| search (action=purchase index=main product_name="Batguy Slippers" sourcetype=access_combined status=200) | stats count as sales
sum(product_price) as revenue by product_name action status| search (action=purchase product_name="Batguy Slippers" status=200)
```

```
(action=purchase index=main product_name="Batguy Slippers"
sourcetype=access_combined status=200)
| stats count as sales sum(product_price) as revenue by product_name
action status
| search (action=purchase product_name="Batguy Slippers" status=200)
```

The search criterias have  
been added to the  
beginning of the search  
pipeline.

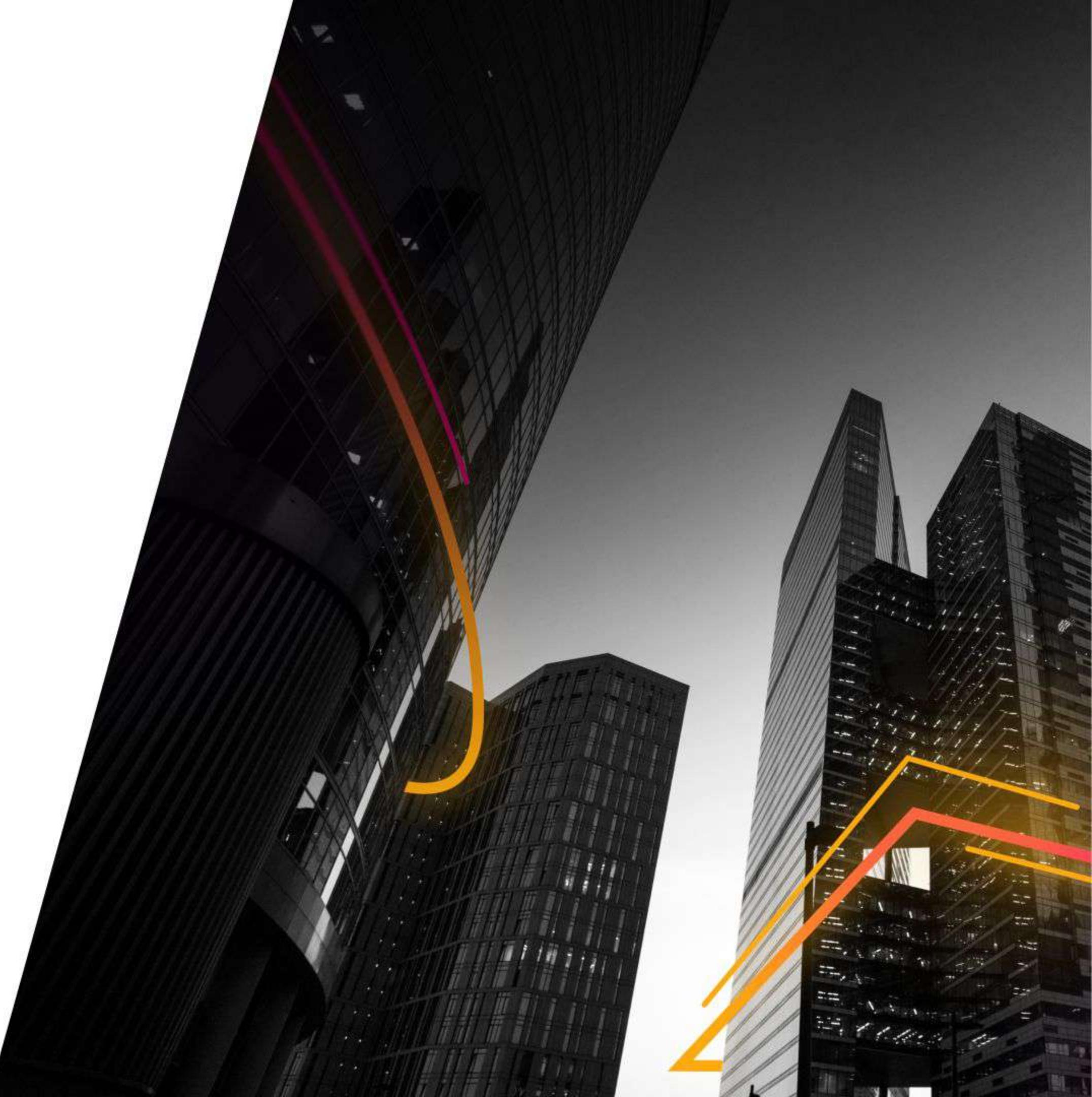
The criterias at the end  
were grouped together, but  
they are now redundant.

References: <https://docs.splunk.com/Documentation/Splunk/latest/Search/ViewsearchjobpropertieswiththeJobInspector>  
<https://docs.splunk.com/Documentation/Splunk/latest/Search/Built-inoptimization>

# **SPL#3 >**

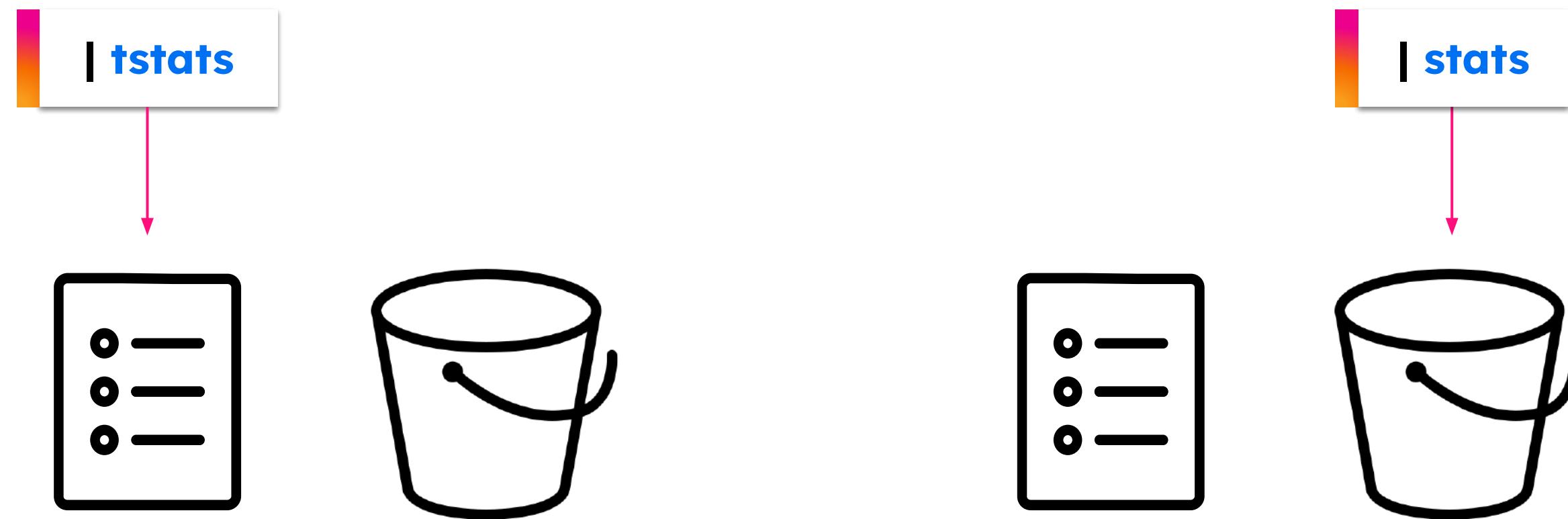
## **tstats with**

## **PREFIX**



# Why use **tstats**?

- **tstats** is faster than **stats** because it only looks at the **indexed metadata** (the **tsidx files**), while **stats** looks at the **raw data** from the upstream query.



Learn more: <https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Tstats>

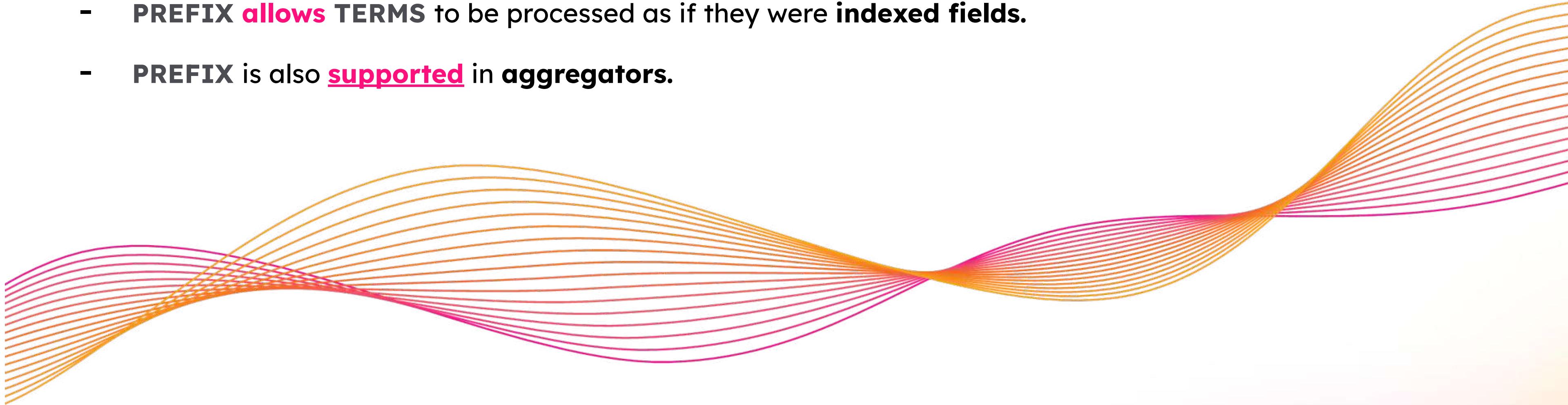
# Tstats limitations and challenges

- Pre-existence of indexed fields - indexed fields can be either from **indexed data**, or from **Data Model acceleration**.
  - At ingestion we can extract **metadata** from **raw event** and create indexed fields.
  - Some **structured data** sources can optionally create indexed fields **automatically**.
  - The **HTTP Event collector** has a “**fields**” section.
  - Post ingestion we can create a **data model**.
- It is difficult to discover the existence of **indexed fields** when available
  - The **walklex** command can help

```
| walklex index=main type=field
| stats values(field) AS indexed_fields
```
  - The existence of **TERMS** can be inferred from log data

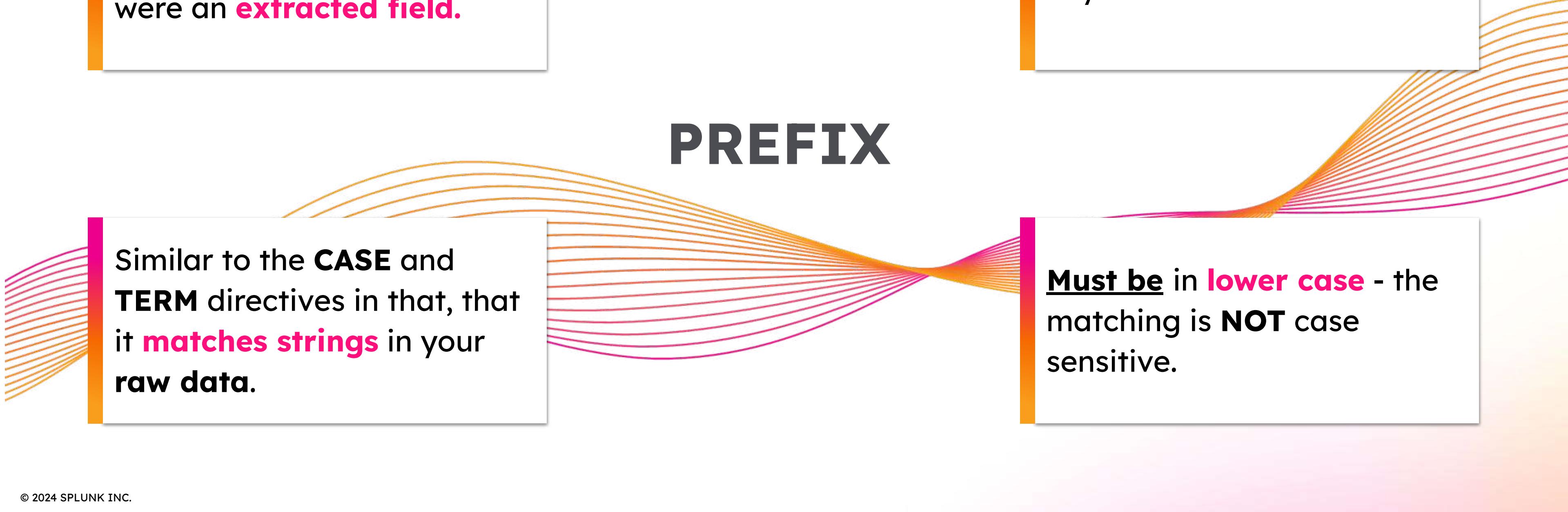
# Introducing **tstats** with **PREFIX**

- With **PREFIX**, **indexed fields** are no longer a requirement.
- The **PREFIX** directive massively increases the instances where **tstats** can be used.
- **PREFIX** **allows** **TERMS** to be processed as if they were **indexed fields**.
- **PREFIX** is also supported in **aggregators**.



Learn more: <https://conf.splunk.com/files/2020/slides/PLA1089C.pdf>

# What is the PREFIX directive?



Search on a **raw segment** in your **indexed data** as if it were an **extracted field**.

Locate a **recurring segment** in your **raw event data**.

Similar to the **CASE** and **TERM** directives in that, that it **matches strings** in your **raw data**.

**Must be in lower case** - the matching is **NOT** case sensitive.

## PREFIX

# Differences between a classic search and tstats

Consider the following log:

```
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
```

# Differences between a **classic** search and **tstats**

```
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
```

## Classic search

```
index=_internal 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
```

# Differences between a **classic** search and **tstats**

```
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
```

## Classic search

```
index=_internal 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
```

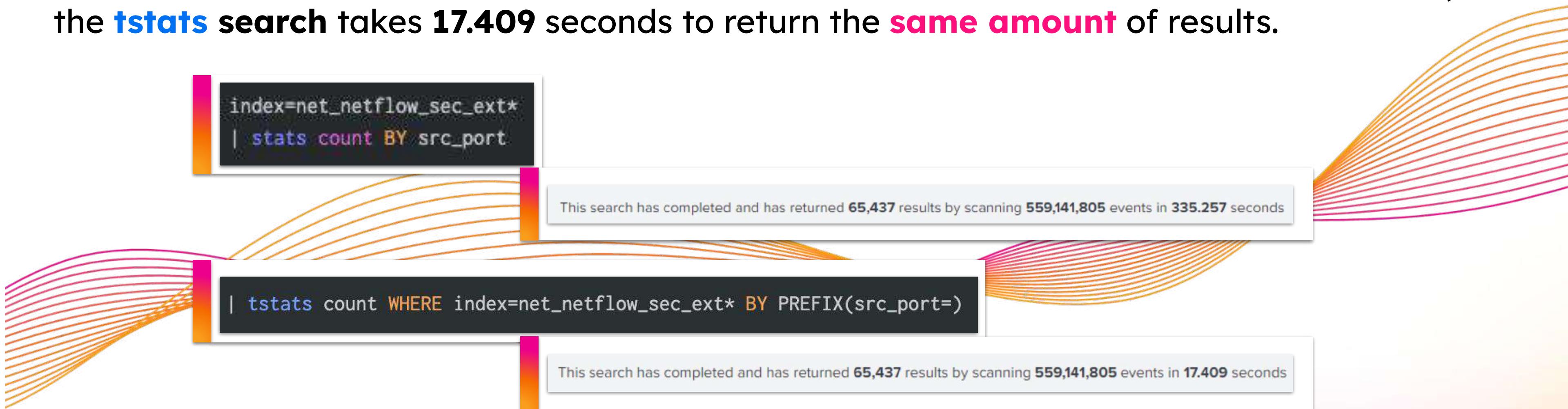
## tstats search

```
| tstats sum(PREFIX(kb=)) AS indexer_kb avg(PREFIX(instantaneous_kbps=)) AS instantaneous_kbps avg(PREFIX(load_average=)) AS load_avg
WHERE
index=_internal
TERM(group=thruput)
TERM(name=thruput)
BY host _time
span=1767s
```

# Simple search across Netflow logs

Now let's have a look at a **real world** sample **dataset** to see the **difference** between **tstats** and a **raw search**, where we can see the difference between one and the other. The total number of events in this case is **559,141,805**.

The **difference** is clear: while the **raw search** takes **335.257** seconds to return our results, the **tstats search** takes **17.409** seconds to return the **same amount** of results.



# tstats Challenge

How many actions are performed by **jsessionid**, for the **most active 10 sessions?**

```
| tstats count(PREFIX(action=)) AS count WHERE index=main BY PREFIX(jsessionid=)
| rename jsessionid= AS JSESSIONID
| eval JSESSIONID=upper(JSESSIONID)
| sort - count
| head 10
```

| JSESSIONID       | count |
|------------------|-------|
| SD10SL10FF4ADFF7 | 12    |
| SD6SL4FF10ADFF7  | 12    |
| SD9SL5FF4ADFF3   | 12    |
| SD2SL2FF2ADFF8   | 11    |
| SD4SL1FF10ADFF4  | 11    |
| SD4SL2FF3ADFF6   | 11    |
| SD5SL2FF2ADFF5   | 11    |
| SD6SL8FF6ADFF8   | 11    |
| SD7SL8FF10ADFF8  | 11    |
| SD10SL6FF6ADFF1  | 10    |

**SPL#4 >**  
**makeresults**  
**and eval**



# eval command

For example, **defining a variable** in programming

The **eval** command **calculates an expression** and puts the resulting value into a **search results field**.

**Example:** ... | **eval** velocity=distance/time

If the fields velocity  
is **not present** in events

If the field velocity  
**is present** in events

Creates a **New Field** “**velocity**” &  
add it to search  
results

**Overwrites** the  
value  
of field “**Velocity**”  
in search results

Calculates **velocity**  
if “**distance**” &  
“**time**” fields **are**  
**present** in the  
events

# eval Command

Versatile command for calculations and operations

What can be done with the **eval** command? A few **examples**:

## 1. If/else conditions

```
... | eval error=if(status==200, "OK", "Problem")
```

## 2. Case

```
... | eval error_msg=case(status==404, "Not found", status==503,
"Service Unavailable", status==200, "OK")
```

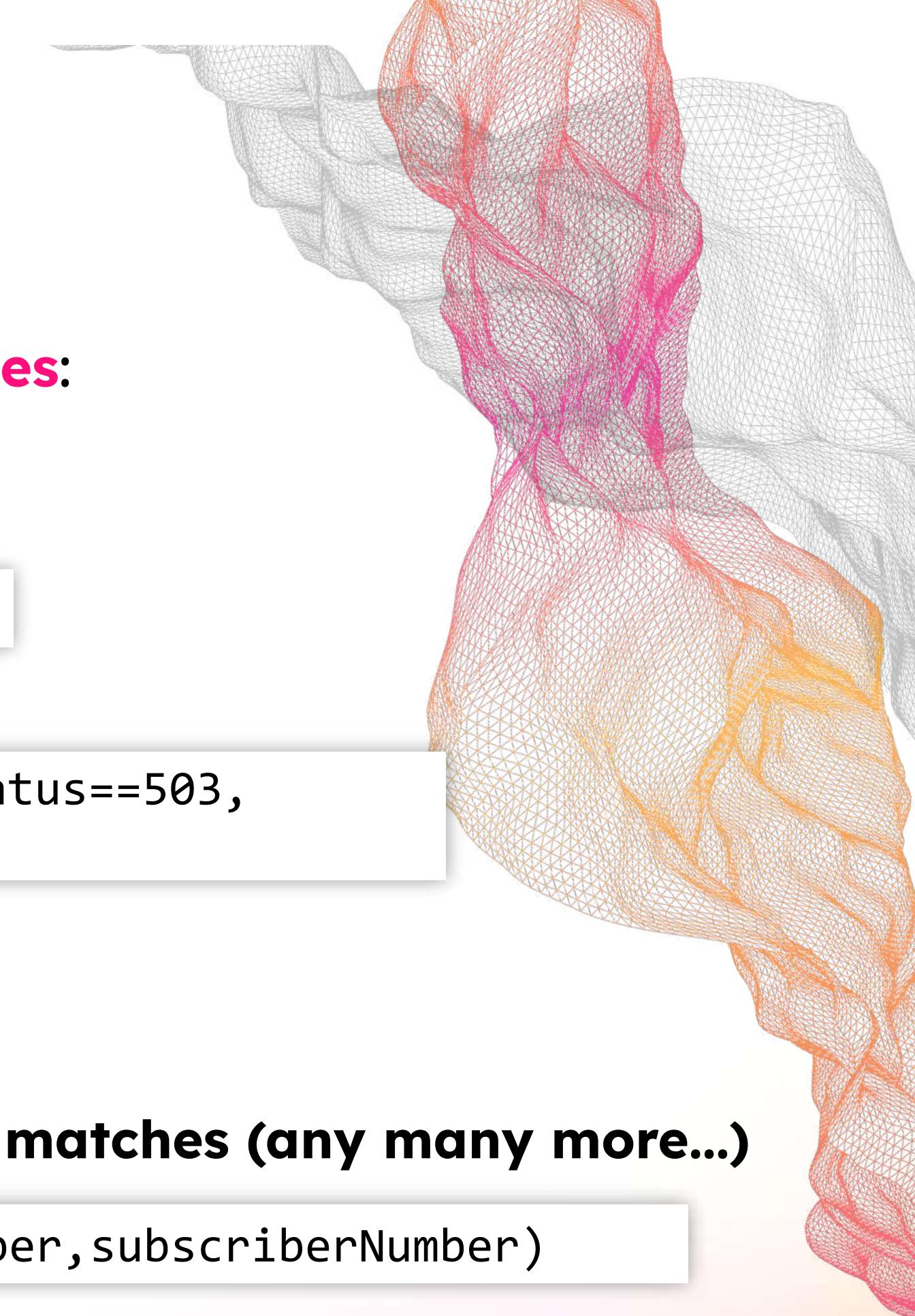
## 1. Mathematical Operations

```
... | eval velocity=round(distance/time,2)
```

## 2. Field operations – combine, split, concatenate, check matches (any many more...)

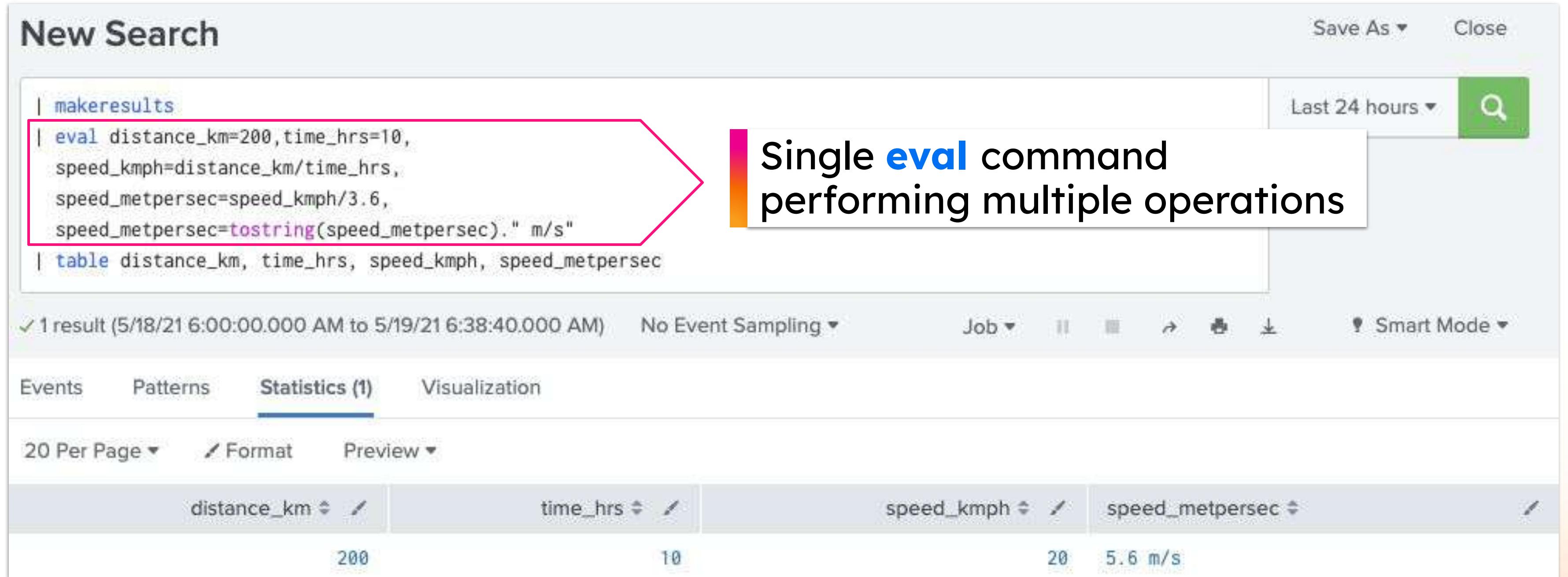
```
sourcetype=A OR sourcetype=B | eval phone=coalesce(number,subscriberNumber)
```

> Learn more: <https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Eval>



# SPL#4 > makeresults Command

**eval** executes sequentially > allows **Mathematical**, **Boolean** & **String** calculations



The screenshot shows the Splunk search interface with a search bar containing the following SPL command:

```
| makeresults
| eval distance_km=200,time_hours=10,
 speed_kmph=distance_km/time_hours,
 speed_meters_per_sec=speed_kmph/3.6,
 speed_meters_per_sec=tostring(speed_meters_per_sec)." m/s"
| table distance_km, time_hours, speed_kmph, speed_meters_per_sec
```

A pink arrow points from the explanatory text below to the eval command in the search bar.

**Single eval command performing multiple operations**

The search results pane shows one result from May 18, 2021, to May 19, 2021. The Statistics tab is selected, displaying the following summary table:

| Field                | Value   |
|----------------------|---------|
| distance_km          | 200     |
| time_hours           | 10      |
| speed_kmph           | 20      |
| speed_meters_per_sec | 5.6 m/s |

# SPL#4 > Using eval to customize outputs

Use **eval** to **format** the final **output**.

## Try out the Searches



### Task

Trying your hands at eval  
command

In our **dataset** we have **3 categories** of products:

- 1) **Books**
- 2) **Gifts**
- 3) **Clothing**

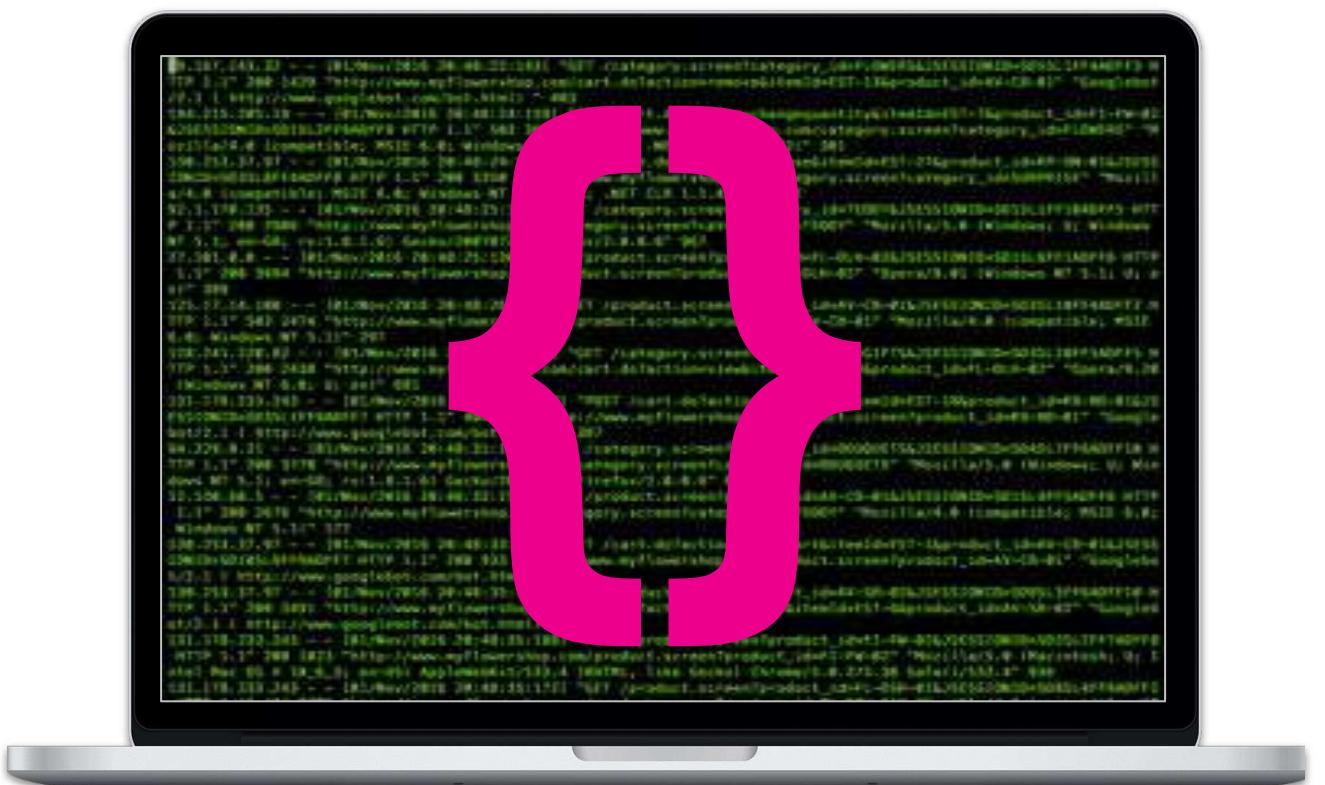
Depending upon the **top category** in your **dataset**, find the **top sold** category and **output** the **final message** as one of below.

Most of Our Customers Love Reading Books

Most of Our Customers Love Sharing Gifts

Most of Our Customers Love Trying New Clothes

# SPL#5 > eval's Hidden Feature



# eval and the Field Name Trick

We can use **eval** to dynamically create **additional fields** based on the value of another field. The **trick** is to use curly braces “**{}**”.

The screenshot shows a Splunk search interface. On the left, a search command is displayed:

```
sourcetype = access_combined
| eval is_{category} = 1
```

Below the search bar, the results show three new fields created by the eval command:

- # is\_Books 1
- # is\_Clothing 1
- # is\_Gifts 1

On the right, a detailed view of the "is\_Books" field is shown:

**is\_Books**

1 Value, 39.248% of events

**Reports**

| Average over time | Maximum value over time | Minimum value over time |
|-------------------|-------------------------|-------------------------|
| Top values        | Top values by time      | Rare values             |

Events with this field

Avg: 1 Min: 1 Max: 1 Std Dev: 0

| Values | Count  | %    |
|--------|--------|------|
| 1      | 21,337 | 100% |

Learn more: <https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Eval>

# **eval** and the Field Name Trick

That is nice to know, but what problem are we solving here? **What's the point?**



# **eval** and the Field Name Trick

Let's demonstrate further with a practical exercise  
a.k.a. **Challenge Time!**

How could we count the **Total Sales** of each product **category**, and get the results in a format **suitable** to use for a **multi-metric** or a **base search** like below?

| Total_Sales_of_Books | Total_Sales_of_Clothing | Total_Sales_of_Gifts |
|----------------------|-------------------------|----------------------|
| 41305.21             | 103928.31               | 29798.31             |

# eval and the Field Name Trick

We can get to the results with **stats** alone, but the **output format** is not ideal for a **base search**. Since we would like to have one field per category.

Flipping lines and columns using the **transpose** command can help !

```
index=main sourcetype=access_combined action=purchase status=200
| stats sum(product_price) as Total_Sales by category
| transpose header_field=category
| fields - column
| rename * as Total_Sales_of_*
```

| Total_Sales_Of_Books | Total_Sales_Of_Clothing | Total_Sales_Of_Gifts |
|----------------------|-------------------------|----------------------|
| 7610.78              | 18648.66                | 5520.81              |

# eval and the Field Name Trick

Quite often with **SPL**, we can get **similar or identical results** using **differents methods**.

Let's try now with **timechart**.

While this is **not the fastest** way, we can also get pretty **good results**:

```
index=main sourcetype=access_combined action=purchase
status=200
| eval category="Total_Sales_of_"+' category'
| timechart span=1w sum(product_price) by category
| fields - _time
```

Total\_Sales\_Of\_Books ↴ ✓

193332.22

Total\_Sales\_Of\_Clothing ↴ ✓

477569.83

Total\_Sales\_Of\_Gifts ↴ ✓

139191.12

# eval and the Field Name Trick

Here's a **faster** and **more elegant** way using **eval** and the **field name trick**.

- We keep only the **purchase actions**
- We create a new field called “**Total\_Sales\_of\_{category}**” and give each event the **product\_price** as **value**
- As the last step, we combine with **stats** and **wildcards** to calculate the **total amount sold per category**

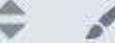
```
index=main sourcetype=access_combined action=purchase
status=200
| eval Total_Sales_of_{category} = product_price
| stats sum(Total_Sales_of_*) as Total_Sales_of_*
```

Total\_Sales\_of\_Books 

41305.21

Total\_Sales\_of\_Clothing 

103928.31

Total\_Sales\_of\_Gifts 

29798.31

# SPL#6 >

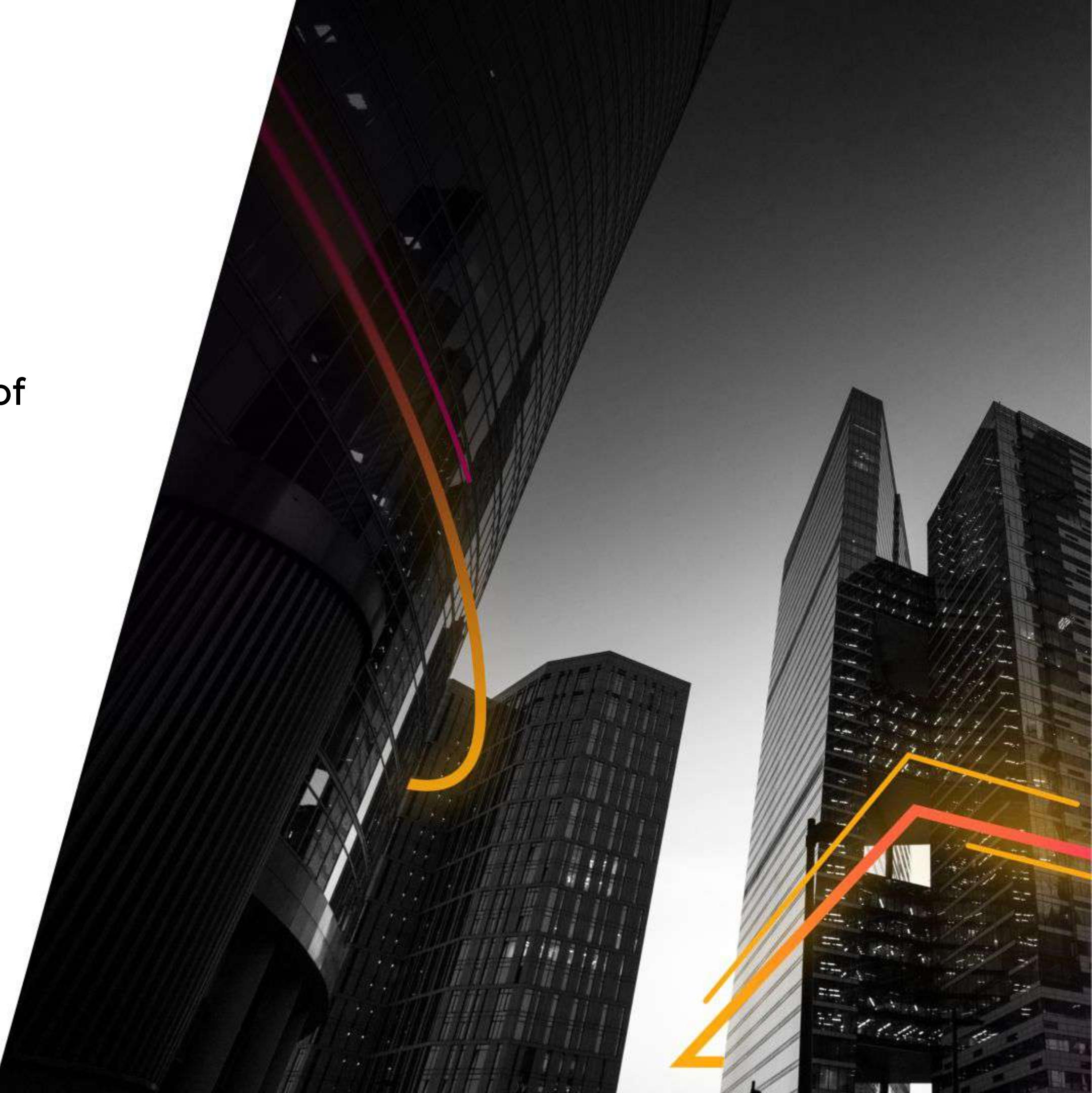
## Complex Queries



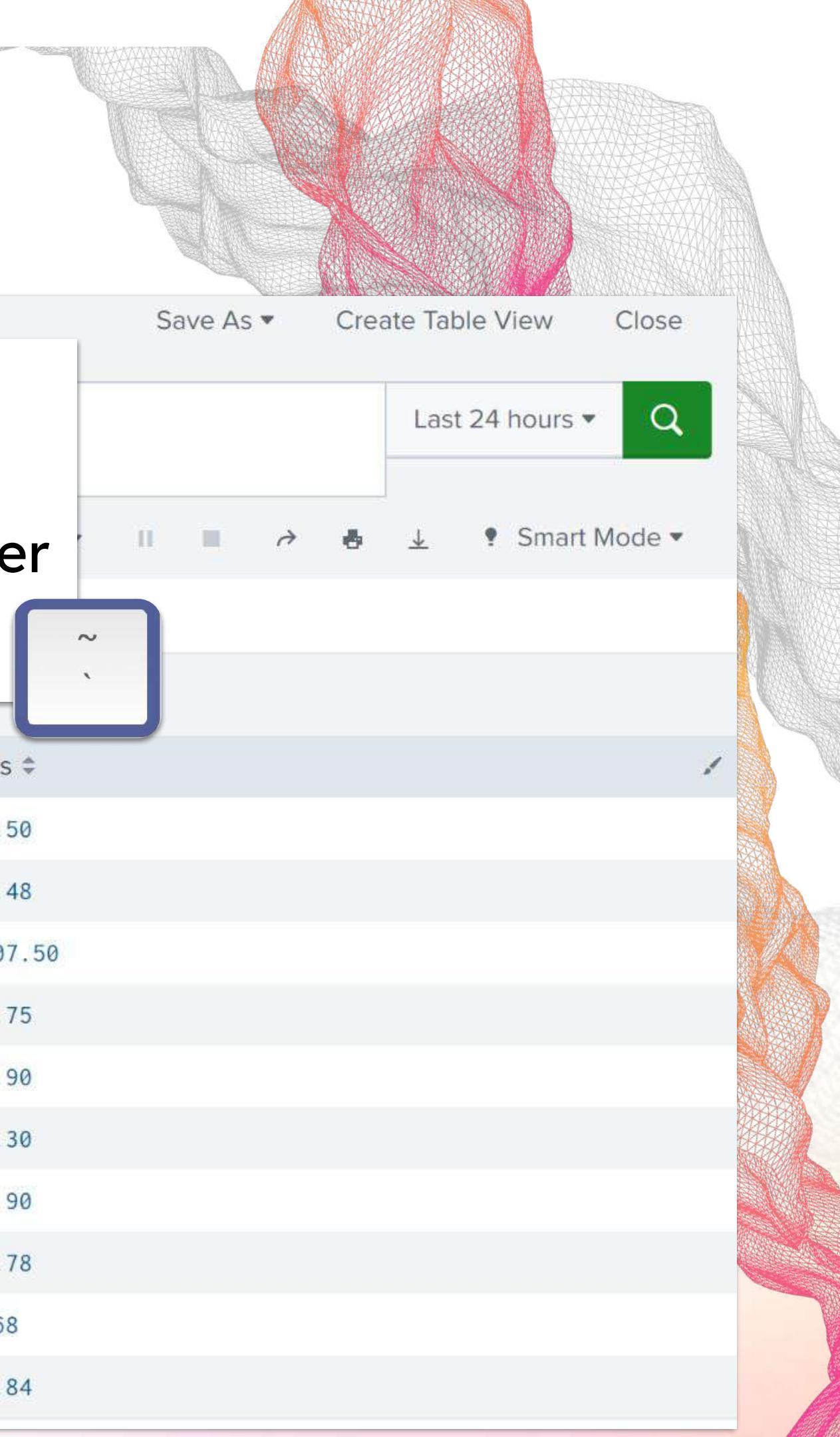
```
index=main sourcetype="complex_query"
| eval ticket_end_time =
 if(actionCode=="D", strftime(actionTime,"%Y/%m/%d
%H:%M:%S"),NULL)
| eval ticket_start_time =
 if(actionCode=="I", strftime(firstOccurrence,"%Y/%m/%d
%H:%M:%S"),NULL)
| transaction serverName serverSerial ticketNumber
| eval ticketDuration = ticket_end_time - ticket_start_time
| eval pretty_ticketDuration =
 floor(ticketDuration/60/60)." Hours ".floor(floor(ticketDuration -
 (floor(ticketDuration/60/60)*60*60))/60)." Min ".
 floor(ticketDuration%60)." Sec."
| stats avg(ticketDuration) as Average_Ticket_Duration
 list(pretty_ticketDuration) as pretty_ticketDuration by
 serverName serverSerial
```

# 'Macros'

- Reusable **search strings** or portions of search strings
- Time range **independent**
- Pass **arguments** to the search



# Using Macros



- Surround the name with **backtick characters** ` in order to invoke macros



# Creating a Macro

The screenshot shows the Splunk UI with the following navigation path:

- Administrator
- 3 Messages
- Settings (highlighted with a pink box)

The Settings menu is expanded, showing several categories:

- KNOWLEDGE**: Searches, reports, and alerts; Data models; Event types; Tags; Fields; Lookups; User interface; Alert actions; **Advanced search** (highlighted with a pink box); All configurations.
- DATA**: Data inputs; Forwarding and receiving; Indexes; Report acceleration summaries; Virtual indexes.
- DISTRIBUTED ENVIRONMENT**: Indexer clustering; Forwarder management; Data Fabric; Federated search; Distributed search.
- USERS AND AUTHENTICATION**: Roles; Users; Tokens; Password management; Authentication methods.
- SYSTEM**: Server settings; Server controls; Health report manager; RapidDiag; Instrumentation; Workload management; Mobile settings.

1. Click on **Settings**

2. Click on **Advanced search**

3. Click on **+ Add new**

4. Choose **Destination app, Name, Definition**

5. Specify a **name**

6. Specify **SPL**

7. Click on **Save**

The dialog is titled "Search macros" and has the following fields:

- Destination app**: S4N\_SPLBP
- Name \***: ConvertUSD
- Definition \***:

```
stats sum(product_price) as total_sales by product_name
| eval total_sales=("$" + tostring(round(total_sales,2),"commas"))
```
- Use eval-based definition?
- Arguments**: Enter a comma-delimited string of '-' characters.
- Validation Expression**: Enter an eval or boolean expression that runs over macro arguments.
- Validation Error Message**: Enter a message to display when the validation expression returns 'false'.

Cancel

Save

# Expand Macros SPL

New Search

Save As ▾ Create Table View Close

```
index=main sourcetype=access_combined action=purchase status=200
| `ConvertUSD`
```

✓ 162,648 events (07/01/2024 17:00:00.000 to 08/01/2024 17:15:18.000)

Events Patterns Statistics (10) Visualization

20 Per Page ▾ Format Preview ▾

product\_name ▾

Batguy Slippers

Batguy Watch

Costume- ManHawk

Double Fudge Sundae

Mad Comics- Batguy

Mad Comics- Bronze Man

Mad Comics- Flyman

Pony Potpourri

Waterproof Scratch and Sniff

Zombie Survival Guide

⌘ command + shift + E

OR

ctrl + shift + E

## Expanded Search String

X

```
index=main sourcetype=access_combined action=purchase status=200
| stats sum(product_price) as total_sales by product_name
| eval total_sales=("$" + tostring(round(total_sales,2),"commas"))
```

Cancel

Open as new search

# Adding Macro Arguments

Destination app: S4N\_SPLBP

Name \*: ConvertUSD\_2arguments(2)

Definition \*:  
stats \$function\$(product\_price) as \$newname\$ by product\_name | eval \$newname\$=("\$" +  
tostring(round(\$newname\$,2),"commas"))

Use eval-based definition?

Arguments:  
function, newname

Validation Expression:

Validation Error Message:

Cancel **Save**

- Great way to make `macros` more flexible
- Number of arguments needs to be included in parenthesis in the name field
- Argument names are surrounded by dollar(\$) signs

# Using Macro with arguments

Allows us to **change** variables **easily** at **search time**.

New Search

Save As ▾ Create Table View Close

```
index=main sourcetype=access_combined action=purchase status=200
| `ConvertUSD_2arguments(avg,average_price)`
| sort average_price
```

Last 24 hours ▾

✓ 179,317 events (07/01/2024 17:00:00.000 to 08/01/2024 17:43:27.000) No Event Sampling ▾ Job ▾ II ■ ↗ + ↓ Smart Mode ▾

Events Patterns Statistics (10) Visualization

20 Per Page ▾ Format Preview ▾

| product_name                 | average_price |
|------------------------------|---------------|
| Mad Comics- Batguy           | \$12.70       |
| Mad Comics- Bronze Man       | \$12.70       |
| Mad Comics- Flyman           | \$12.70       |
| Zombie Survival Guide        | \$15.21       |
| Double Fudge Sundae          | \$22.75       |
| Batguy Slippers              | \$25.70       |
| Waterproof Scratch and Sniff | \$4.99        |
| Batguy Watch                 | \$9.99        |
| Pony Potpourri               | \$9.99        |
| Costume- ManHawk             | \$97.50       |

# Arguments Validation

Destination app S4N\_SPLBP

Name \* Enter the name of the macro. If the search macro takes an argument, indicate this by appending the number of arguments to the name. For example: mymacro(2)  
ConvertUSD\_3arguments(3)

Definition \* Enter the string the search macro expands to when it is referenced in another search. If arguments are included, enclose them in dollar signs. For example: \$arg1\$  
stats \$function\$(product\_price) as \$newname\$ by product\_name | \$format\$ \$newname\$=("\$" + tostring(round(\$newname\$,2),"commas"))

Use eval-based definition?

Arguments Enter a comma-delimited string of argument names. Argument names may only contain alphanumeric, '\_' and '-' characters.  
function, newname, format

Validation Expression Enter an eval or boolean expression that runs over macro arguments.  
\$format\$="fieldformat" OR \$format\$="eval"

Validation Error Message Enter a message to display when the validation expression returns 'false'.  
ConvertUSD\_3arguments(3) must include a stats function, a field name for the new value, and a format arg

- **Validation expression arguments to be surrounded by dollar(\$) signs.**
- **Can use an eval expression or boolean operators.**
- **Provide informative message for the end user in case the `macro` returns with error.**

# Arguments order

**Arguments** needs to be **passed** in the **order** they appear in the **definition**

New Search

```
index=main sourcetype=access_combined action=purchase status=200
| `ConvertUSD_3arguments(sum,total_sales,fieldformat)`
| `ConvertUSD_3arguments(sum,total_sales,fieldformat)`
| sort total_sales
```

1 2 3

Last 24 hours

✓ 187,187 events (07/01/2024 17:00:00.000 to 08/01/2024 17:56:31.000) No Event Sampling Job ▾ II ▾ Smart Mode ▾

Events Patterns Statistics (10) Visualization

20 Per Page ▾ Format Preview ▾

| product_name                 | total_sales    |
|------------------------------|----------------|
| Waterproof Scratch and Sniff | \$92,953.72    |
| Batguy Watch                 | \$183,506.31   |
| Pony Potpourri               | \$190,029.78   |
| Mad Comics- Flyman           | \$237,172.50   |
| Mad Comics- Batguy           | \$237,490.00   |
| Mad Comics- Bronze Man       | \$240,322.10   |
| Zombie Survival Guide        | \$283,240.62   |
| Double Fudge Sundae          | \$421,944.25   |
| Batguy Slippers              | \$485,473.00   |
| Costume- ManHawk             | \$1,834,072.50 |

# Macro validation

Executing the **macro** **incorrectly** will produce an **error** for the end user that we specified earlier in our “**Validation error message**” field.

## New Search

```
index=main sourcetype=access_combined action=purchase status=200
| `ConvertUSD_3arguments(sum,total_sales,count)`
|sort total_sales
```

Last 24 hours ▾ 

 Error in 'SearchParser': Encountered the following error while validating macro 'ConvertUSD\_3arguments(sum,total\_sales,count)':  
ConvertUSD\_3arguments(3) must include a stats function, a field name for the new value, and a format argument of eval or fieldformat..

# Nested Macros

Destination app **S4N\_SPLBP**

Name \* Enter the name of the macro. If the search macro takes an argument, indicate this by appending the number of arguments to the name. For example: mymacro(2)  
**EUSales**

Definition \* Enter the string the search macro expands to when it is referenced in another search. If arguments are included, enclose them in dollar signs. For example: \$arg1\$  
**index=main sourcetype="access\_combined" action=purchase status=200  
|iplocation clientip  
|rename Country as country  
|lookup geo\_attr\_countries.csv country  
|search region\_un=Europe  
|`ConvertUSD`**

Use eval-based definition?

Arguments Enter a comma-delimited string of argument names. Argument names may only contain alphanumeric, '\_' and '-' characters.

Validation Expression Enter an eval or boolean expression that runs over macro arguments.

Validation Error Message Enter a message to display when the validation expression returns 'false'.

**Cancel** **Save**



**Inner `Macros` should always be created first before passing them to the outer macros !**

**SPL#7 >**

**eventstats and  
streamstats**

**Search**





## Use Case Example

Thinking beyond stats command

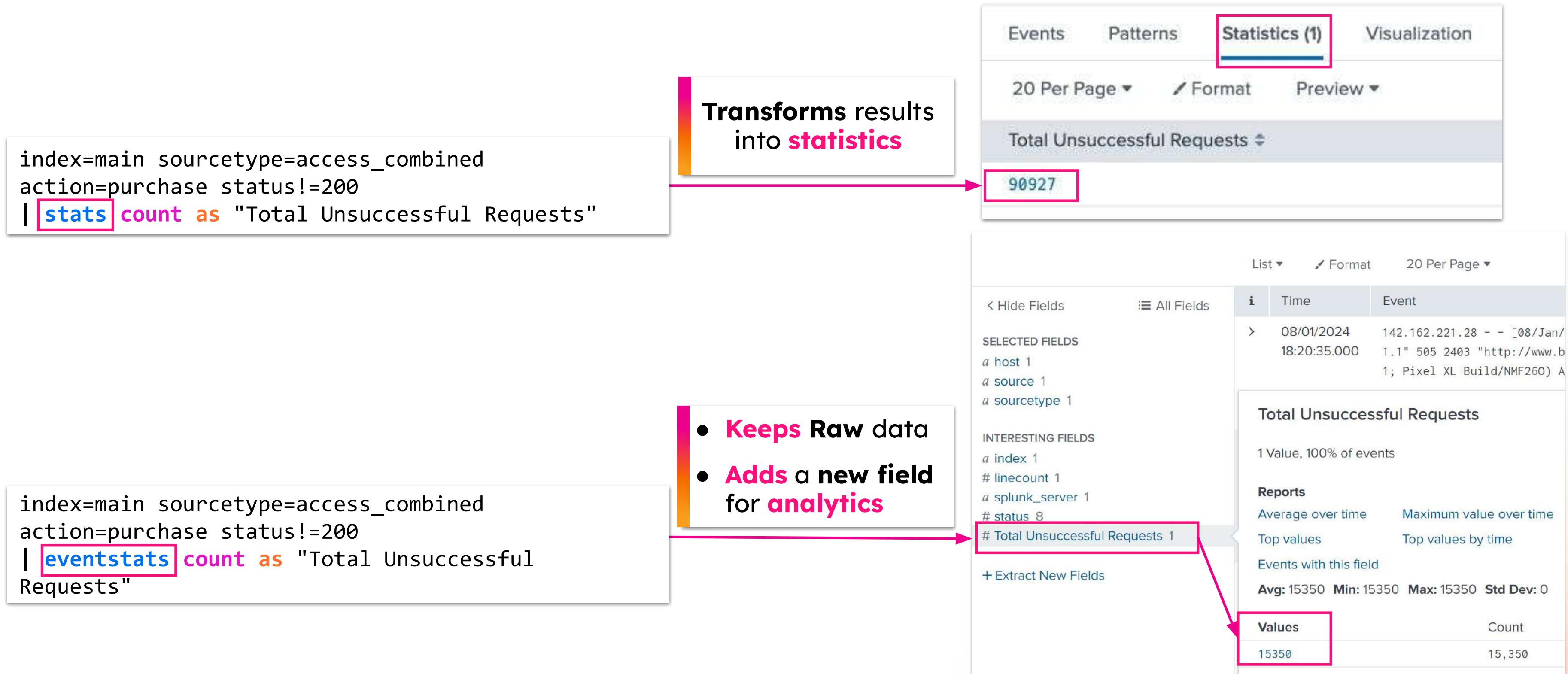
# stats may not always be helpful !

So **how** do you start **solving** such a **use case**?

- How many **transactions** did it take to increase the **total revenue** from **\$500** to **\$1000**
- What is the **total number of successful transactions**?

# stats is a transforming command

While  
eventstats and streamstats add **new fields** to search while **retaining raw data**



# Beyond stats, eventstats and streamstats

**Adding** Aggregate results to result set for **additional** calculations

## | eventstats

Generates **summary statistics** from fields in all your events and **saves** those statistics in a **new field**

## | streamstats

Adds **cumulative summary statistics** to **all search results** in a **streaming manner**

| <b>stats command</b><br>(works on all results)                                                                   | <b>eventstats command</b><br>(works on all results)                                                                                            | <b>streamstats command</b><br>(works on each event at the time its seen)  |
|------------------------------------------------------------------------------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------|
| Events are transformed into a <b>table of aggregated search results</b>                                          | Aggregations are placed into a <b>new field that is added to each of the events</b> in your output                                             | Calculates <b>statistics for each event</b> at the time the event is seen |
| You <b>can only</b> use the fields in your <b>aggregated results</b> in <b>subsequent commands</b> in the search | You <b>can use</b> the fields in your events in <b>subsequent commands</b> in your search, because the events <b>have not been transformed</b> |                                                                           |

# Statistics functions available with eventstats and streamstats

Similar to stats – more complex functions can be used

| Type of function                     | Supported functions and syntax                                                                                                    |                                                                                                                                 |                                                                                                         |                                                                                                                                |  |
|--------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------|--|
| Aggregate functions                  | <code>avg()</code><br><code>count()</code><br><code>distinct_count()</code><br><code>estdc()</code><br><code>estdc_error()</code> | <code>exactperc&lt;int&gt;()</code><br><code>max()</code><br><code>median()</code><br><code>min()</code><br><code>mode()</code> | <code>perc&lt;int&gt;()</code><br><code>range()</code><br><code>stdev()</code><br><code>stdevp()</code> | <code>sum()</code><br><code>sumsq()</code><br><code>upperperc&lt;int&gt;()</code><br><code>var()</code><br><code>varp()</code> |  |
| Event order functions                | <code>earliest()</code>                                                                                                           | <code>first()</code>                                                                                                            | <code>last()</code>                                                                                     | <code>latest()</code>                                                                                                          |  |
| Multivalue stats and chart functions | <code>list(X)</code>                                                                                                              | <code>values(X)</code>                                                                                                          |                                                                                                         |                                                                                                                                |  |

Learn more: [https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Eventstats#Stats\\_function\\_options](https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Eventstats#Stats_function_options)

# SPL#7 > eventstats Search

Search for **failed purchase transactions** over the **last 24 hours**

## Solution:

```
index=main sourcetype=access_combined action=purchase status!=200
| eventstats count as "Total Unsuccessful Requests"
```

A screenshot of the Splunk interface showing the results of an eventstats search. On the left, a sidebar lists various fields: a referer (100+), a referer\_domain (1), a req\_time (100+), a splunk\_server (1), # status (8), # timeendpos (8), # timestamppos (8), # Total Unsuccessful Requests (1), a uri (100+), a uri\_path (3), a uri\_query (100+), a user (1), a useragent (22), and # version (1). An orange arrow points from a callout box to the '# Total Unsuccessful Requests' field, which is highlighted with a pink border. The main panel displays the results for the '# Total Unsuccessful Requests' field. It shows a single value of 1, representing 100% of events. Below this, there are sections for Reports, Values, and a detailed summary table.

**New field created with total count of unsuccessful requests**

| Values | Count  | %    |
|--------|--------|------|
| 23607  | 23,607 | 100% |

Learn more: <https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Eventstats>

# SPL#7 > streamstats Search

Search for **failed purchase transactions** over the **last 24 hours**

**Solution:**

```
index=main sourcetype=access_combined status!=200 action=purchase
| streamstats count as "Total Unsuccessful Requests"
```

**New field created in streaming manner >**

**Adding the count as it finds a new event with status!=200 and adds count to event**

The screenshot shows a Splunk search interface. On the left, a list of fields is displayed, including '# other 100+', 'a product\_id 10', 'a product\_name 10', '# product\_price 7', 'a punct 9', 'a referer 100+', 'a referer\_domain 1', 'a req\_time 100+', 'a splunk\_server 1', '# status 8', '# timeendpos 8', '# timestamppos 8', and '# Total Unsuccessful Requests 100+'. The '# Total Unsuccessful Requests 100+' field is highlighted with a red box. An orange arrow points from this box to the right side of the screen. On the right, a detailed view of the '# Total Unsuccessful Requests' field is shown under the 'Reports' section. It includes statistics: Avg: 11959, Min: 1, Max: 23917, Std Dev: 6904.387530162734. Below this, a table titled 'Top 10 Values' lists the value '1' with a count of 1 and a percentage of 0.004%, and the value '10' with a count of 1 and a percentage of 0.004%.

| Top 10 Values | Count | %      |
|---------------|-------|--------|
| 1             | 1     | 0.004% |
| 10            | 1     | 0.004% |

Learn more: <https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Streamstats>

**SPL#8 >**

**Percentage of  
Total Revenue**



# SPL#8 > Percentage of Total Revenue

Percentage of **total** revenue generated by “Zombie Survival Guide” in the **last 4 hours**

| product_name          | per_prod_rev | total_rev  | rev_pct   |
|-----------------------|--------------|------------|-----------|
| Zombie Survival Guide | 217731.15    | 3225141.63 | 6.7510570 |

## Solution:

```
index=main sourcetype=access_combined action=purchase status=200
| eventstats sum(product_price) as total_rev
| eventstats sum(product_price) as rev_per_prod by product_id
| eval percent_revenue=rev_per_prod/total_rev*100
| where product_name="Zombie Survival Guide"
| stats values(rev_per_prod) as per_prod_rev, values(total_rev) as total_rev,
values(percent_revenue) as rev_pct by product_name
```

# SPL#8 > Challenge Task

What **search** should you **run** to **show** the **table** below?

The screenshot shows a Splunk search results page. At the top, there are tabs for 'Events', 'Patterns', 'Statistics (1)', and 'Visualization'. The 'Statistics (1)' tab is selected, indicated by a blue underline. Below the tabs are filters: '20 Per Page ▾', 'Format' (with a pencil icon), and 'Preview' (with a dropdown arrow). The main area displays a single row of data. The first column is 'Product Name' with a dropdown arrow, showing 'Zombie Survival Guide'. The second column is 'Percentage Revenue' with a dropdown arrow, showing '6.78 %'. A vertical color bar on the left side of the interface transitions from orange at the top to red at the bottom.

| Product Name          | Percentage Revenue |
|-----------------------|--------------------|
| Zombie Survival Guide | 6.78 %             |

# SPL#8 > Challenge Task Solution

## Solution:

```
index=main sourcetype=access_combined action=purchase status=200
| eventstats sum(product_price) as total_rev
| eventstats sum(product_price) as rev_per_prod by product_id
| eval percent_revenue=rev_per_prod/total_rev*100
| stats values(rev_per_prod) as per_prod_rev, values(total_rev) as total_rev,
values(percent_revenue) as rev_pct by product_name
| eval rev_pct=round(rev_pct,2)." %"
| where product_name="Zombie Survival Guide"
| rename rev_pct as "Percentage Revenue", product_name as "Product Name"
| table "Product Name", "Percentage Revenue"
```

**SPL#9 >**

# Popularity Ranking



# SPL#9 > Popularity Ranking

Popularity ranking of each **product** within its **category** in the **last 24 hours**

## Solution:

```
index=main sourcetype="access_combined" action=purchase status=200
| stats count by product_name, category
| sort - count
| streamstats count as rank by category
| stats list(rank) as "Category Rank", list(product_name) as "Product Name" by category
```

| category | Category Rank | Product Name                 |
|----------|---------------|------------------------------|
| Books    | 1             | Mad Comics- Flyman           |
|          | 2             | Mad Comics- Bronze Man       |
|          | 3             | Zombie Survival Guide        |
|          | 4             | Mad Comics- Batguy           |
| Clothing | 1             | Batguy Watch                 |
|          | 2             | Batguy Slippers              |
|          | 3             | Costume- ManHawk             |
| Gifts    | 1             | Pony Potpourri               |
|          | 2             | Waterproof Scratch and Sniff |
|          | 3             | Double Fudge Sundae          |

# SPL#10 > Using streamstats and eventstats

**stats** may not be always **helpful** !



## Challenge Task

How many **transactions** did it take to increase the **total revenue** from **\$500** to **\$1000** and what was the **total number of successful transactions**?

### Use Case Breakdown:

1. Find **successful purchase events**
2. Calculate the **total revenue** from the point in time when **it was \$500** and then **increased to \$1000**
3. Calculate the **total number of events** it took to go from **\$500 to \$1000**
4. Calculate the **number of total transactions**

# SPL#10 > Challenge

How many **transactions** did it take to increase the **total revenue** from **\$500 to \$1000** and what was the **total number** of **successful transactions**?

| Transaction Count <500 | Transaction Count 500<1000 | Transaction Count >1000 | Total Transaction Count | Total Sales |
|------------------------|----------------------------|-------------------------|-------------------------|-------------|
| 25                     | 27                         | 24649                   | 24701                   | 556919.86   |

## Solution:

```
index=main sourcetype=access_combined action=purchase status=200
| reverse
| streamstats sum(product_price) as revenue_tracker
| eval txn_count = case(revenue_tracker<500, "<500",
revenue_tracker>=500 AND revenue_tracker<1000, "500<1000",
revenue_tracker>=1000, ">1000"),
txn_count_{txn_count}=txn_count
| stats count(txn_count_*) as "Transaction Count *" count as "Total Transaction Count"
sum(product_price) as "Total Sales"
```

# **SPL#11 >**

## **spath as an SPL Command**



# Extract Specific Information

## Default > Splunk Extracts JSON

Directly in SPL

| **spath** [**input=<field>**] [**output=<field>**]  
**path=<datopath>**

**spath**  
command

As **function** within  
**eval** command

| **eval A=spath(X,Y)**

# SPL#11 > spath as an SPL command

Use **spath** command **directly** within SPL

## Syntax

```
| spath [input=<field>] [output=<field>] [path=<datapath> | <datapath>]
```

Field to **read & extract** values from

**Output** written to this field

**Location path** to the **value** needed for **extraction**

Learn more: <https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Spath>

# SPL#11 > spath as an SPL command

Use **spath** command **directly** within SPL

**Solution:**

```
index=sample_data application_performance
| spath input=_raw output=new_json_field path=application_performance{}
```

```
Errors per Minute 1
Exceptions per Minute 1
HTTP Error Codes per Minute 1
a index 1
Infrastructure Errors per Minute 1
linecount 1
a new_json_field 14
Normal Average Response Time (ms)
1
a punct 1
```

Explore the newly created field “**new\_json\_field**”

**new\_json\_field**  
14 Values, 100% of events

Reports  
Top values      Top values by time  
Events with this field

Top 10 Values

```
{"frequency": "ONE_MIN", "metricId": 22750664, "metricName": "PWNY|Application Summary|Average Response Time (ms)", "metricPath": "Overall Application Performance|Average Response Time (ms)", "metricValues": [{"count": 185, "current": 0, "max": 1, "min": 0, "occurrences": 0, "standardDeviation": 0, "startTimeInMillis": 1613031900000, "sum": 42, "useRange": true, "value": 0}]}]
```

- Nested **JSON Array** under **application\_performance** is **extracted & added** into **new\_json\_field**.
- **Note:** There are **14 values** extracted. We have **2 events** with **7 values** each embedded within **application\_performance**.

# **SPL#12 >**

## **spath as an eval**

# **Function**



# Extract Specific Information

Default > Splunk Extracts JSON

Directly in SPL

| **spath** [**input=<field>**] [**output=<field>**]  
**path=<datopath>**

**spath**  
command

As **function** within  
**eval** command

| **eval A=spath(X,Y)**

# SPL#12 > spath as an eval function

Use **spath** function **directly** within the **eval** command

**Syntax:**

```
| eval A=spath(X,Y)
```

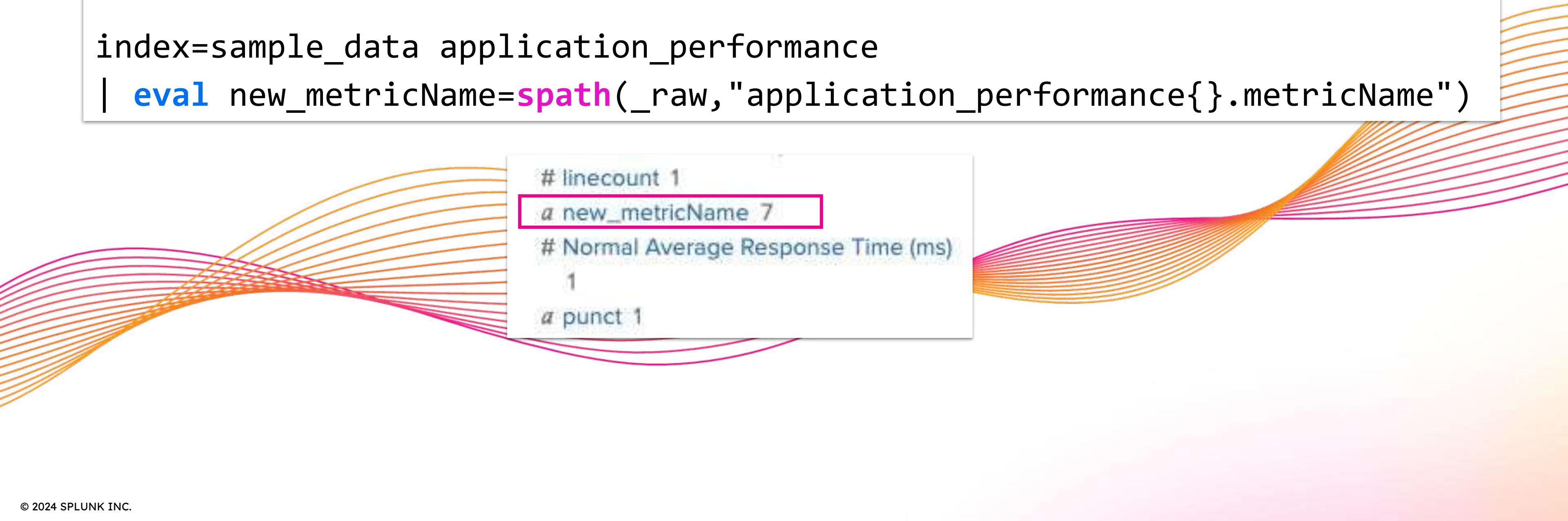
From **structured** data type(**X**) in **XML** or **JSON** format; extract value at **Y** and assign to **A**

# SPL#12 > spath as an eval function

## SPL#12 > Search 1 of 2 over All Time

### Solution:

```
index=sample_data application_performance
| eval new_metricName=spath(_raw,"application_performance{}.metricName")
```



```
linecount 1
a new_metricName 7
Normal Average Response Time (ms)
1
a punct 1
```

# SPL#12 > spath as an eval function

## SPL#12 > Search 2 of 2 over All Time

### Solution:

```
index=sample_data application_performance
| eval new_metricName=spath(_raw,"application_performance{4}.metricName")
| eval new_metricValue=spath(_raw,"application_performance{4}.metricValues{}.current")
```

# linecount 1  
a new\_metricName 1  
# new\_metricValue 1  
# Normal Average Response Time (ms)  
1  
a punct 1

Learn more: <https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Spath>

# Validate if a new field was created



# Validate if a new field was created

Explore the **output field(s)**:

## Solution:

```
index=sample_data application_performance
| spath input=_raw output=new_json_field path=application_performance{}
| eval new_metricName=spath(new_json_field,"metricName")
```

The **SPL** does **NOT**  
extract new field  
**“new\_metricName”**

Why not?

```
a index 1
Infrastructure Errors per Minute 1
linecount 1
a new_json_field 14
Normal Average Response Time (ms)
1
a punct 1
```

**Find out why the  
new field was  
not created**



# Find out why the new field was not created

**spath** command creates a **multi-value** field:

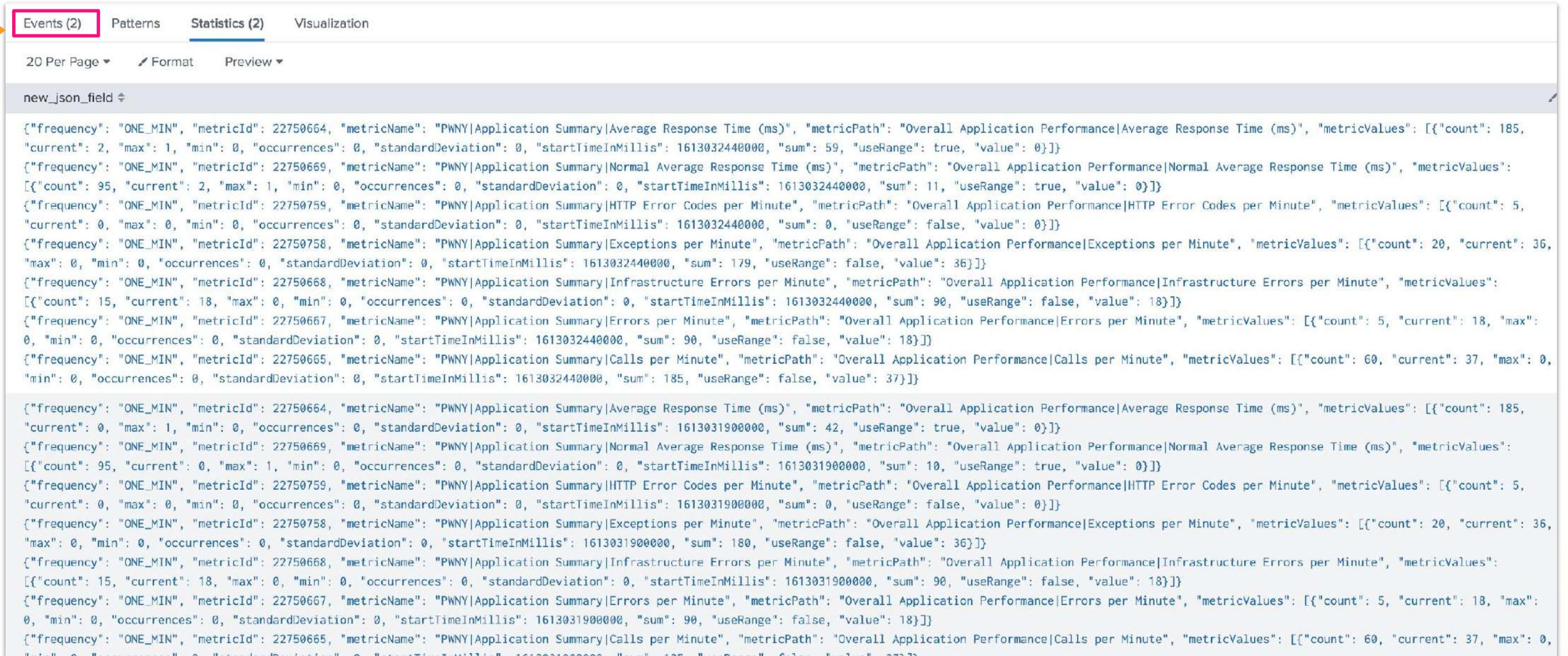
## Solution:

```
index=sample_data application_performance
| spath input=_raw output=new_json_field path=application_performance{}
| table new_json_field
```



# Find out why the new field was not created

Two events  
returned with  
multiple values



The screenshot shows the Splunk Statistics view with two events listed. An orange arrow points from the text "Two events returned with multiple values" to the Statistics tab in the top navigation bar. The first event is titled "new\_json\_field". The JSON output for the first event is as follows:

```
{"frequency": "ONE_MIN", "metricId": 22750664, "metricName": "PWNY|Application Summary|Average Response Time (ms)", "metricPath": "Overall Application Performance|Average Response Time (ms)", "metricValues": [{"count": 185, "current": 2, "max": 1, "min": 0, "occurrences": 0, "standardDeviation": 0, "startTimeInMillis": 1613032440000, "sum": 59, "useRange": true, "value": 0}], {"frequency": "ONE_MIN", "metricId": 22750669, "metricName": "PWNY|Application Summary|Normal Average Response Time (ms)", "metricPath": "Overall Application Performance|Normal Average Response Time (ms)", "metricValues": [{"count": 95, "current": 2, "max": 1, "min": 0, "occurrences": 0, "standardDeviation": 0, "startTimeInMillis": 1613032440000, "sum": 11, "useRange": true, "value": 0}], {"frequency": "ONE_MIN", "metricId": 22750759, "metricName": "PWNY|Application Summary|HTTP Error Codes per Minute", "metricPath": "Overall Application Performance|HTTP Error Codes per Minute", "metricValues": [{"count": 5, "current": 0, "max": 0, "min": 0, "occurrences": 0, "standardDeviation": 0, "startTimeInMillis": 1613032440000, "sum": 0, "useRange": false, "value": 0}], {"frequency": "ONE_MIN", "metricId": 22750758, "metricName": "PWNY|Application Summary|Exceptions per Minute", "metricPath": "Overall Application Performance|Exceptions per Minute", "metricValues": [{"count": 20, "current": 36, "max": 0, "min": 0, "occurrences": 0, "standardDeviation": 0, "startTimeInMillis": 1613032440000, "sum": 179, "useRange": false, "value": 36}], {"frequency": "ONE_MIN", "metricId": 22750668, "metricName": "PWNY|Application Summary|Infrastructure Errors per Minute", "metricPath": "Overall Application Performance|Infrastructure Errors per Minute", "metricValues": [{"count": 15, "current": 18, "max": 0, "min": 0, "occurrences": 0, "standardDeviation": 0, "startTimeInMillis": 1613032440000, "sum": 90, "useRange": false, "value": 18}], {"frequency": "ONE_MIN", "metricId": 22750667, "metricName": "PWNY|Application Summary|Errors per Minute", "metricPath": "Overall Application Performance|Errors per Minute", "metricValues": [{"count": 5, "current": 18, "max": 0, "min": 0, "occurrences": 0, "standardDeviation": 0, "startTimeInMillis": 1613032440000, "sum": 90, "useRange": false, "value": 18}], {"frequency": "ONE_MIN", "metricId": 22750665, "metricName": "PWNY|Application Summary|Calls per Minute", "metricPath": "Overall Application Performance|Calls per Minute", "metricValues": [{"count": 60, "current": 37, "max": 0, "min": 0, "occurrences": 0, "standardDeviation": 0, "startTimeInMillis": 1613032440000, "sum": 185, "useRange": false, "value": 37}]}]
```

The second event is also titled "new\_json\_field" and has identical JSON output.

Learn more: <https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/Spath>

**SPL#13 >**

# The **mvexpand** Command



# SPL#13 > The mvexpand Command

Expand and extract the field

## Solution:

```
index=sample_data application_performance
| spath input=_raw output=new_json_field
path=application_performance{}
| mvexpand new_json_field
| eval new_metricName=spath(new_json_field,"metricName")
```

Did the **SPL**  
**extract** the  
**new\_metricName**  
field?

```
linecount 1
a new_json_field 14
a new_metricName 7
Normal Average Response Time (ms)
1
a punct 1
```

**SPL#14 >**

# Introducing the **rex** Command

Dealing with Improperly formatted data



# Field Extraction and Regex Refresher

## Extracting the value of the **Heap Memory** usage from the **raw data**

- After extraction, Splunk provides the **Regular Expression** used to provide the **instructed field**
- **SPL#14** shows how to incorporate the **rex** command (**regular expression**) within the **SPL**

### Select Fields

Highlight one or more values in the sample event to create fields. You can indicate one value is required, meaning it must exist in an event for the regular expression to match. Click on highlighted values in the sample event to modify them. To highlight text that is already part of an existing extraction, first turn off the existing extractions. Learn more ↗

```
{"account_name": "buttercup-dev", "appd_app_uuid": "PWNY:26822:buttercup-dev.saas.appdynamics.com", "application_id": "26822", "application_name": "PWNY", "healthrule_violations": [{"affectedEntityDefinition": {"entityId": 179966, "entityType": "APPLICATION_COMPONENT_NODE", "name": "PWNY-Apps-Weblogic-server317-servicesAdminServer"}, "deepLinkUrl": "https://buttercup-dev.saas.appdynamics.com/#location=APP INCIDENT_DETAIL_MODAL&incident=3313892&application=26822", "description": "AppDynamics has detected a problem with Node PWNY-Apps-Weblogic-server317-servicesAdminServer.
 JVM Heap utilization is too high started violating and is now warning.
 All of the following conditions were found to be violating
For Node PWNY-Apps-Weblogic-server317-servicesAdminServer:
1) JVM|Memory:Heap|Used % Condition
Used %'s value 76.00 was greater than the threshold 75.00 for the last 30 minutes.
", "detectedTimeInMillis": 0, "endTimeInMillis": 0, "id": 3313892, "incidentStatus": "OPEN", "name": "JVM Heap utilization is too high", "severity": "WARNING", "startTimeInMillis": 1613029555001, "triggeredEntityDefinition": {"entityId": 104997, "entityType": "POLICY", "name": "JVM Heap utilization is too high"}}, {""affectedEntityDefinition": {"entityId": 180065, "entityType": "APPLICATION_COMPONENT_NODE", "name": "PWNY-Apps-Weblogic-server318-webAdminServer"}, "deepLinkUrl": "https://buttercup-dev.saas.appdynamics.com/#location=APP INCIDENT_DETAIL_MODAL&incident=3313951&application=26822", "description": "AppDynamics has detected a problem with Node PWNY-Apps-Weblogic-server318-webAdminServer.
 JVM Heap utilization is too high started violating and is now warning.
 All of the following conditions were found to be violating
For Node PWNY-Apps-Weblogic-server318-webAdminServer:
1) JVM|Memory:Heap|Used % Condition
Used %'s value 76.00 was greater than the threshold 75.00 for the last 30 minutes.
", "detectedTimeInMillis": 0, "endTimeInMillis": 0, "id": 3313951, "incidentStatus": "OPEN", "name": "JVM Heap utilization is too high", "severity": "WARNING", "startTimeInMillis": 1613031535001, "triggeredEntityDefinition": {"entityId": 104997, "entityType": "POLICY", "name": "JVM Heap utilization is too high"}}]}
```

Hide Regular Expression ▾

[View in Search ↗](#)

[Edit the Regular Expression](#)

# SPL#14 > Introducing the **rex** command

## Rex Command Syntax:

```
| rex [field=<field>] [max_match=<int>] [offset_field=<string>]
(<regex-expression> | mode=sed <sed-expression>)
```

## Today's focus:

- | rex field=<field> "<regex-expression>"
- | rex field=<field> mode=sed "<sed-expression>"

# SPL#14 > Introducing the **rex** command continued

**Example:** Extracting the **delivery** method (saas) value from the **healthrule\_violations** events in the **sample\_data** index

```
index=sample_data healthruleViolations
| rex field=appd_app_uuid "^\w+:\d+:[\w\-\]+\.(\?<delivery>\w+)\."
```

JSON field which contains the **value** we need

Name of **new field**

The screenshot shows a Splunk search interface. On the left, the search bar contains the SPL command. On the right, the search results pane displays a single event. The event details are as follows:

```
date_hour 1
date_mday 1
date_minute 1
date_month 1
date_second 1
date_wday 1
date_year 1
date_zone 1
a delivery 1
a healthrule_violations().affectedEntity
Definition.entityId 3
11/02/2021 07:45:55.001 { [-]
account_name: buttercup-dev
appd_app_uuid: PWNY:26822:buttercup-dev saas.appdynamics.com
application_id: 26822
application_name: PWNY
healthrule_violations: [[+]
]
}
```

A red arrow points from the highlighted 'delivery' field in the search bar to the 'delivery' field in the event details. Another red arrow points from the highlighted 'appd\_app\_uuid' field in the search bar to the 'appd\_app\_uuid' field in the event details.

# SPL#14 > Introducing the rex command

## Working with **improperly formatted key/value** in JSON

**Use Case:** From health rule violation **events**, extract “Memory Heap used % **value**”

Use the search string  
“**index=sample\_data**  
**healthruleViolations**”  
to get relevant **events**.

```
Event
{
 account_name: buttercup-dev
 appd_app_uuid: PWNY:26822:buttercup-dev.saas.appdynamics.com
 application_id: 26822
 application_name: PWNY
 healthruleViolations: [
 {
 affectedEntityDefinition: {
 description: AppDynamics has detected a problem with Node PWNY-Apps-Weblogic-server317-servicesAdminServer.
JVM Heap utilization is too high started violating and is now warning.
All of the following conditions were found to be violating
For Node PWNY-Apps-Weblogic-server317-servicesAdminServer:
1) JVM|Memory:Heap|Used % Condition
Used %'s value 76.00 was greater than the threshold 75.00 for the last 30 minutes.

 detectedTimeInMillis: 0
 endTimeInMillis: 0
 id: 3313892
 incidentStatus: OPEN
 name: JVM Heap utilization is too high
 severity: WARNING
 startTimeInMillis: 1613029555001
 triggeredEntityDefinition: {
 ...
 }
 }
]
}
```

The required **value** is not extracted and is within the “**description**” field under “**healthruleViolations**” **JSON object**

# SPL#14 > Introducing the rex challenge

**Challenge:** Calculate the **average** Memory Heap Usage, and **display** it next to a list of values by **Application ID**, as exemplified in the following table:

```
index=sample_data healthruleViolations
| rex field=healthruleViolations{}.description "Used\s%\s\<\b\>\svalue\s\<b\>(?\<mem_used_value>(.\+?))\<"
| stats list(mem_used_value) AS "List of Memory Heap Values", avg(mem_used_value) AS "Average Memory Heap Used"
BY application_id
| rename application_id AS "Application ID"
```

| Application ID | List of Memory Heap Values | Average Memory Heap Used |
|----------------|----------------------------|--------------------------|
| 26822          | 76.00                      | 77                       |
|                | 76.00                      |                          |
|                | 81.00                      |                          |
|                | 76.00                      |                          |
|                | 76.00                      |                          |

## Hints:

- Run the search against **All Time**
- Use the **rex** command to extract the **Memory Heap Used %** value (this can be found in the **healthruleViolations{}.description** field)
- An example of the regular expression is: **Used\s%\s\<\b\>\svalue\s\<b\>(?\<mem\_used\_value>(.\+?))\<**

# **SPL#15 >**

## **Using rex**

### **command with**

#### **SED Mode**



# SPL#15 > Using rex command with SED Mode

Let's consider we want to **anonymize** the **last two digits** of the **application\_id** in our table. To achieve this, we can use the **rex** command with the **mode=sed** option. One of the **most common use cases** for this is **data anonymization at ingest time**.

The diagram illustrates the process of anonymizing the last two digits of the application\_id field using the rex command with mode=sed. It consists of two tables and a command line.

**Top Table:** Shows the original data. The Application ID column contains "26822". The List of Memory Heap Values column contains "76.00", "76.00", "81.00", "76.00", and "76.00". The Average Memory Heap Used column contains "77".

**Middle Section:** Contains three dots ("...") indicating continuation, followed by the SPL command:

```
| rex mode=sed field=application_id "s/(\d{3})(\d{2})/\1xx/g"
```

**Bottom Table:** Shows the result after applying the rex command. The Application ID column now contains "268xx". The List of Memory Heap Values column remains the same: "76.00", "76.00", "81.00", "76.00", and "76.00". The Average Memory Heap Used column remains "77".

# Splunk Resources

Where to go after  
today's workshop



# Splunk's Thriving Community

## Splunk Community

The screenshot shows the Splunk Community homepage. At the top, it displays "Learn, Give Back, Have Fun" with stats: 1,382 Online Now, 123K Discussions, and 51.7K Solutions. Below this, there are three main sections: "Ask questions. Get answers. Find technical product solutions from passionate experts in the Splunk community.", "Meet virtually or in-person with local Splunk enthusiasts to learn tips & tricks, best practices, new use cases and more.", and "Search, vote and request new enhancements (ideas) for any Splunk solution - no more logging, support tickets." There are also links to "Sign In to Ask A Question", "Sign In to Join A Group", and "Sign In to Submit an Idea".

## Splunk Events

The screenshot shows the Splunk Events page. It features a large orange header with the text "Splunk Events" and a sub-header: "Join us at an event near you to gain new skills, expand your network and connect with the Splunk community." Below this, there are filters for "Upcoming Events", "All Event Types", "All Geographies", and "All Solution Areas". A specific event is listed: "Automation for the Modern SOC: Strategies for Smarter Security Operations" on November 9, 2021, at 10:00 AM (GMT). The description notes that analysts are drowning in security alerts and operations work is rife with monotonous, repetitive tasks.

## Documentation

The screenshot shows the Splunk Documentation homepage, titled "splunk> docs". It features a search bar and a navigation bar with links to "Products", "Solutions", "Why Splunk?", "Resources", and "Splixicon". The main content area is divided into several categories: "Platform" (Splunk Cloud Platform™, Splunk® Enterprise, Splunk® Universal Forwarder, Splunk® Data Stream Processor, Splunk® Cloud Services), "Security" (Splunk® Enterprise Security, Splunk® SOAR (Cloud), Splunk® SOAR (On-premises), Splunk® Phantom, Splunk® User Behavior Analytics, Splunk® Mission Control), "IT" (IT Operations Overview, Splunk® IT Service Intelligence, Splunk® IT Essentials Work, Splunk® IT Essentials Learn, Splunk® App for Content Packs, Splunk® On-Call), "Observability" (Splunk® Observability Cloud), "Apps and add-ons" (Splunk® Supported Add-ons), and "Developer tools" (Splunk® Add-on Builder).

## Developer Resources

The screenshot shows the Splunk Dev resources page, titled "Welcome to splunk>dev". It features sections for "Develop for Splunk Cloud and Splunk Enterprise" and "Develop for Observability". The "Develop for Splunk Cloud and Splunk Enterprise" section includes a link to "Build apps and integrations for Splunk Cloud and Splunk Enterprise, test in your free development Splunk platform instance, and deliver in the Splunkbase marketplace.". The "Develop for Observability" section includes a link to "Manage, integrate with, and access features of your Splunk Infrastructure Monitoring organization with the API."

## Splunkbase

The screenshot shows the Splunkbase homepage. It features a central graphic of various app icons and the text "Get more out of Splunk with applications". Below this is a search bar and a section titled "Trending Apps on Splunkbase" featuring several app cards: "Optimizely Marketplace Add-on for Splunk" by Optimizely, "Splunk Enterprise Security" by Splunk Inc., "Forensit Technology Add-on for Splunk" by Forensit Technologies, "Splunk Machine Learning Toolkit" by Splunk Inc., and "Splunk Cloud Platform" by Splunk Inc.

## Education

The screenshot shows the Splunk Training & Certification course catalog. It features a search bar and a "Course Catalog" section with a "Start Your Journey" button. Below this are sections for "What Is Splunk?", "Intro to Splunk", and "Using Fields". The "What Is Splunk?" section describes it as an eLearning course that introduces students to what machine data is, how Splunk can leverage it, and how to use dashboards and explore.

## Splunk Lantern

The screenshot shows the Splunk Lantern Resource Hub. It features a search bar and a "Splunk Lantern Resource Hub" section with the tagline "Lighting your way with clear and actionable guidance from Splunk experts". Below this are sections for "Security Use Case Guidance", "IT Use Case Guidance", and "Cloud Monitoring". The "Security Use Case Guidance" section includes articles on "Compliance and Data Privacy", "Operational Foundations", "Security Investigations", and "Threat Intelligence and Threat Hunting". The "IT Use Case Guidance" section includes articles on "Infrastructure Performance Monitoring", "IT Investigation and Troubleshooting", and "Service Insights".

# Splunk Education

Courses relevant to this workshop

- [Using Fields](#)
- [Search Under the Hood](#)
- [Search Optimization](#)
- [Multivalue Fields](#)



# Deep Dive (**Self - Study**)

- **Tips & Tricks for Optimization:**  
<https://docs.splunk.com/Documentation/Splunk/latest/Search/Quicktipsforoptimization>
- **Fields, Indexed Tokens and You (.conf presentation):**  
<https://conf.splunk.com/files/2019/slides/FN1003.pdf>
- **TSTATS and PREFIX (.conf presentation):**  
<https://conf.splunk.com/files/2020/slides/PLA1089C.pdf>
- **JSON Functions:**  
<https://docs.splunk.com/Documentation/Splunk/latest/SearchReference/JSONFunctions>
- **Joining datasets/emulating SQL behavior:**  
[&](https://docs.splunk.com/Documentation/SplunkCloud/latest/SearchReference/SQLtoSplunk)  
<https://conf.splunk.com/files/2019/slides/FNC2751.pdf>

# Thank you

