



SPL Best Practices

January 2026



Forward-looking statements

This presentation may contain forward-looking statements that are subject to the safe harbors created under the Securities Act of 1933, as amended, and the Securities Exchange Act of 1934, as amended. All statements other than statements of historical facts are statements that could be deemed forward-looking statements. These statements are based on current expectations, estimates, forecasts, and projections about the industries in which we operate and the beliefs and assumptions of our management based on the information currently available to us. Words such as "expects," "anticipates," "targets," "goals," "projects," "intends," "plans," "believes," "momentum," "seeks," "estimates," "continues," "endeavors," "strives," "may," variations of such words, and similar expressions are intended to identify such forward-looking statements. In addition, any statements that refer to (1) our goals, commitments, and programs; (2) our business plans, initiatives, and objectives; and (3) our assumptions and expectations, including our expectations regarding our financial performance, products, technology, strategy, customers, markets, acquisitions and investments are forward-looking statements. 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. Readers are cautioned that these forward-looking statements are only predictions and are subject to risks, uncertainties, and assumptions that are difficult to predict, including those identified in the "Risk Factors" section of Cisco's most recent report on Form 10-Q filed on February 20, 2024 and its most recent report on Form 10-K filed on September 7, 2023, as well as the "Risk Factors" section of Splunk's most recent report on Form 10-Q filed with the SEC on November 28, 2023. 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 Cisco or Splunk, on Cisco or Splunk's website or otherwise, it may not contain current or accurate information. Cisco and Splunk undertake no obligation to revise or update any forward-looking statements for any reason, except as required by law.

In addition, any information about new products, features, functionality or 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 or be relied upon in making a purchasing decision. We undertake no commitment, promise or 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. The development, release, and timing of any features or functionality described for our products remains at our sole discretion.

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

© 2025 Splunk LLC. All rights reserved.

Please introduce Yourself!

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



Before we get started

It is 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 **Performance!**

We will start slowly with the basics then get into actual perf. Tuning. Then, we also prepared some optional modules that may help your daily life as a Splunk Power User.



- ✓ Mission statement: Why Optimize?
- ✓ Recap: Splunk Search Processing Language (SPL)
- ✓ I am just an end user, can I really influence search performance?
- ✓ Tips & Tricks for Writing More Efficient Searches
- ✓ Optional: Macros, eval tricks, stats & eventstats, spath, rex & co.
- ✓ SPL2 – Who, What, When?
- ✓ Wait, but couldn't we automate all this with an LLM?
- ✓ Wrap Up & Resources - Lots of Resources!

**However, to
avoid being
sidetracked we
had to exclude
some topics →**



- ✓ **Dashboarding**
-> A great Dashboard starts with a great search, but this deserves its own dedicated session!
- ✓ **CIM, a.k.a. Common Information Model**
-> There is also dedicated training material.
- ✓ **Datamodels** (ok, maybe just a little bit)
-> Same as CIM
- ✓ **Clusters, and Splunk Administration in general**
-> Today's session is intended for **Power Users** rather than **Administrators**.
- ✓ **Getting Data In, Ingestion Tuning**
-> This also falls into the scope of administration



REGISTRATION

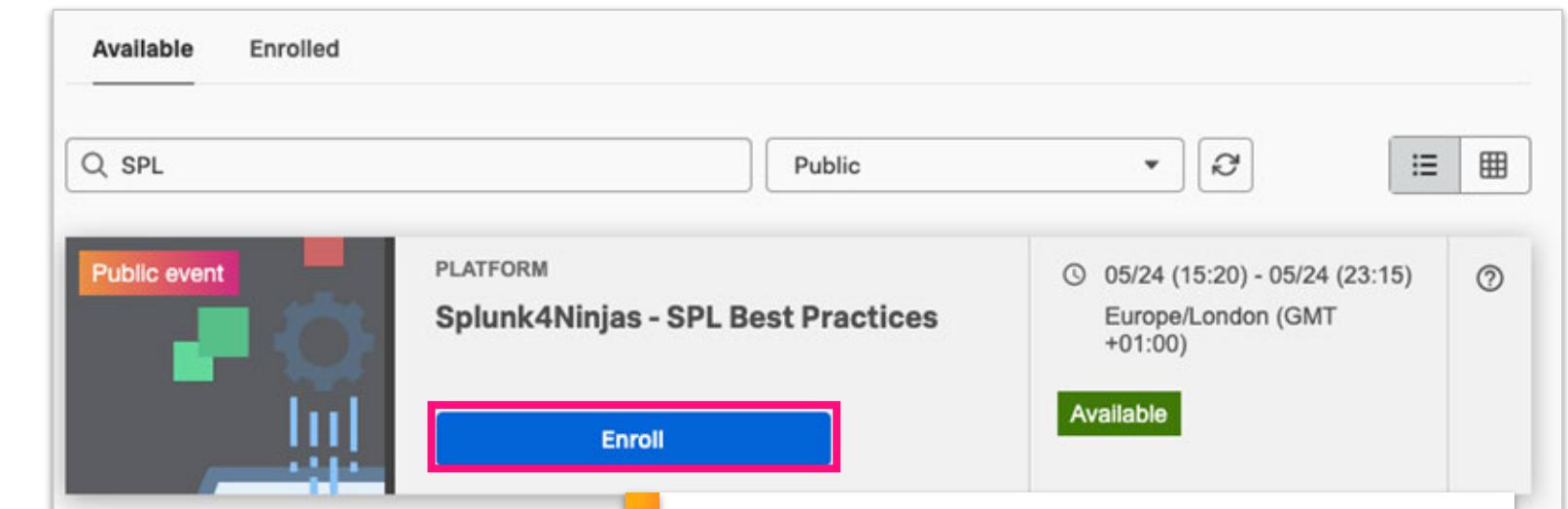
⌚ 5 MINS

Enroll in Today's Workshop

Tasks

1. Get a splunk.com account if you don't have one yet:
<https://splk.it/SignUp>
2. Enroll in the Splunk Show workshop event:
<https://splunk.show/<uniqueID>>
3. Download a copy of today's slide deck:
<https://splk.it/SPL-Attendee>

Goal



A screenshot of a web-based workshop enrollment interface. At the top, there are tabs for 'Available' and 'Enrolled'. Below that is a search bar containing 'SPL' and a dropdown menu set to 'Public'. To the right are several icons. The main content area shows a card for a 'Public event' titled 'Splunk4Ninjas - SPL Best Practices'. The card includes a small icon, the title, the platform, and a date range ('05/24 (15:20) - 05/24 (23:15)'). Below the card is a large blue 'Enroll' button. A callout bubble with the text 'Enroll in today's event' points to this button.



The Lab

2 datasets:

- The Buttercup online store
- A very minimal json sample

All searches are provided in
an

Ad-Hoc Dashboard

All the panels are clickable!

Splunk>enterprise Apps ▾ Administrator ▾ 2 Messages ▾ Settings ▾ Activity ▾ Help ▾ Find Q Splunk4Ninjas - SPL Best Practices Resources Splunk4Ninjas - SPL Best Practices

Splunk4Ninjas - SPL Best Practices ▾

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

Basic S... The TERM dir... **Tstats - Intro + Dat...** Tstats - Ninj... Eval Gymn... Ma... Stats, Eventstats, Stre... Sp... R...

Getting the list of indexed fields with walklex

```
| walklex index=main type=field  
| stats values(field) AS indexed_fields
```

Exploring the Buttercup data model

```
| from datamodel:Buttercup
```

Simple search to display the best selling product - tstats version

```
| tstats count(online_store.action) as purchase_count sum(online_store.product_price) as total_sales from datamodel=Buttercup  
where online_store.action=purchase AND online_store.status<"400" by online_store.category online_store.product_name  
| rename online_store.* as *  
| sort - total_sales
```

Simple search to display the best selling product - stats version

```
| index=main sourcetype=access_combined action=purchase status<400  
| stats count as purchase_count sum(product_price) as total_sales by category product_name  
| sort - total_sales
```

This lab environment will allow us to demonstrate the concepts.

However, be aware that the performance measurements will not be comparable to a real life production environment.

Why optimize anything? My searches are working after all!

True, but...

- Big datasets and complex use cases are less forgiving.
- If we adhere to good practises from the beginning, we won't have to do the work two times.
- Cost. Inefficient searches put more load on the infrastructure, which may have repercussions for you as a customer (depending on the hosting chosen and the licensing model).



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



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`

Commands

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

Functions

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

Clause

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

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

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

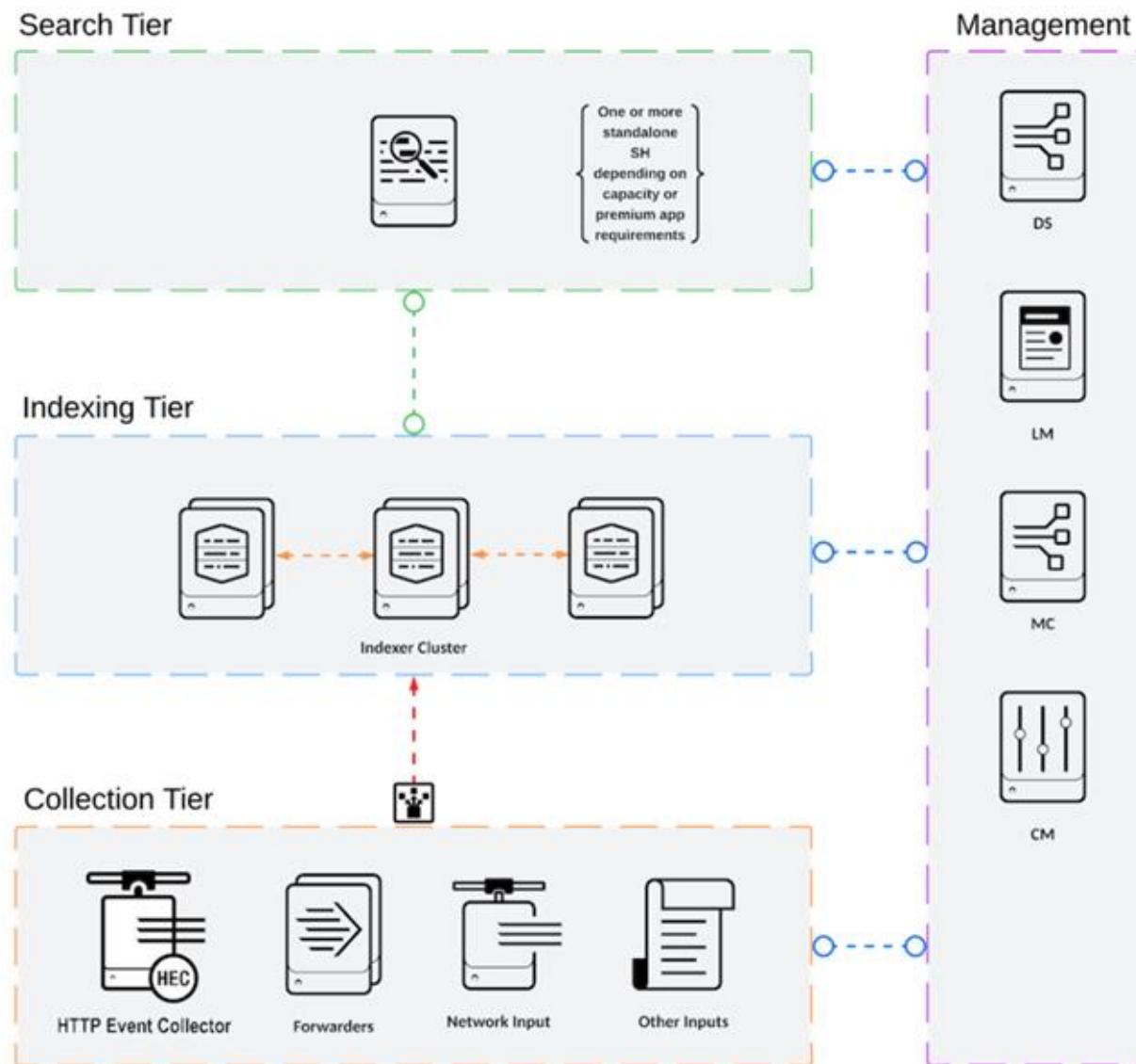
Want to know more? Check out:

Splunk Quick Reference Guide: [Splunk Quick Reference Guide](#)

Search manual: [Get started with Search \(SPL\)](#)

We should be pretty familiar with
what happens at the surface by now.
But let's dive a bit deeper...

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: [Splunk's Validated Architectures](#)

SPL Command Types

Focus of Today's Workshop

Streaming

Transforming

Distributable

Centralized

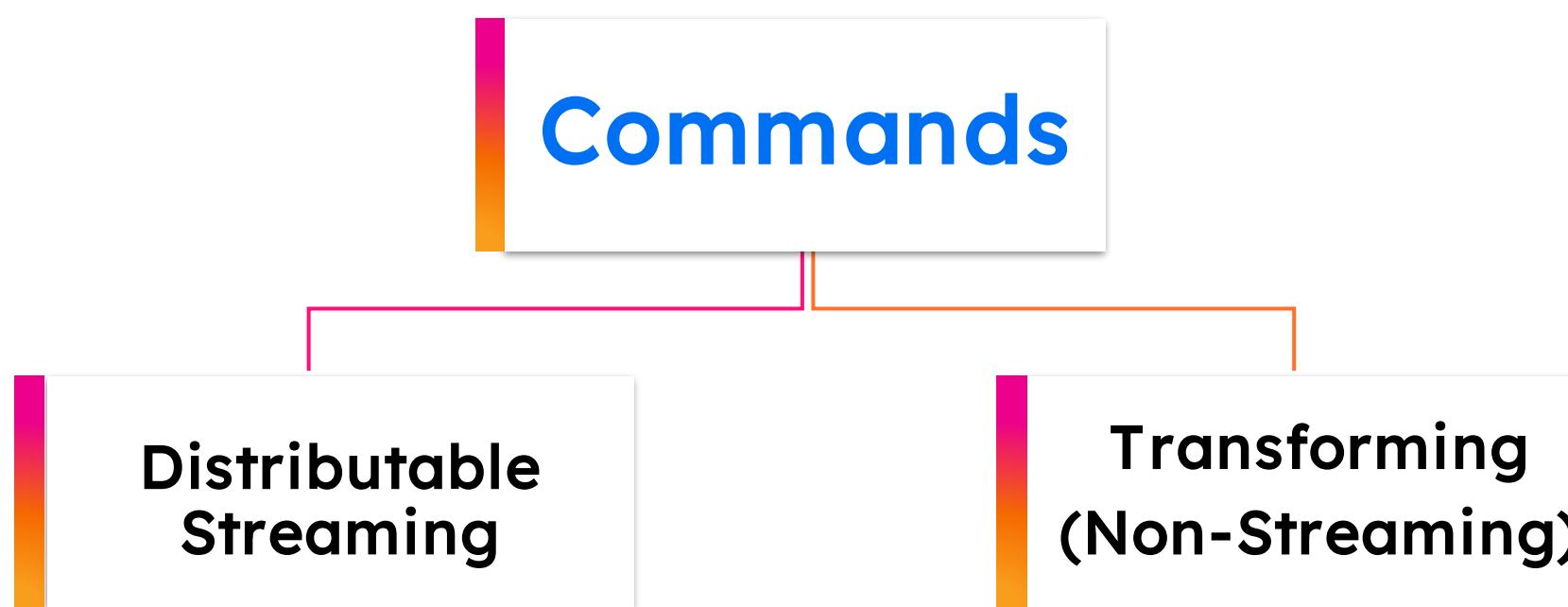
Generating

Orchestrating

Dataset processing

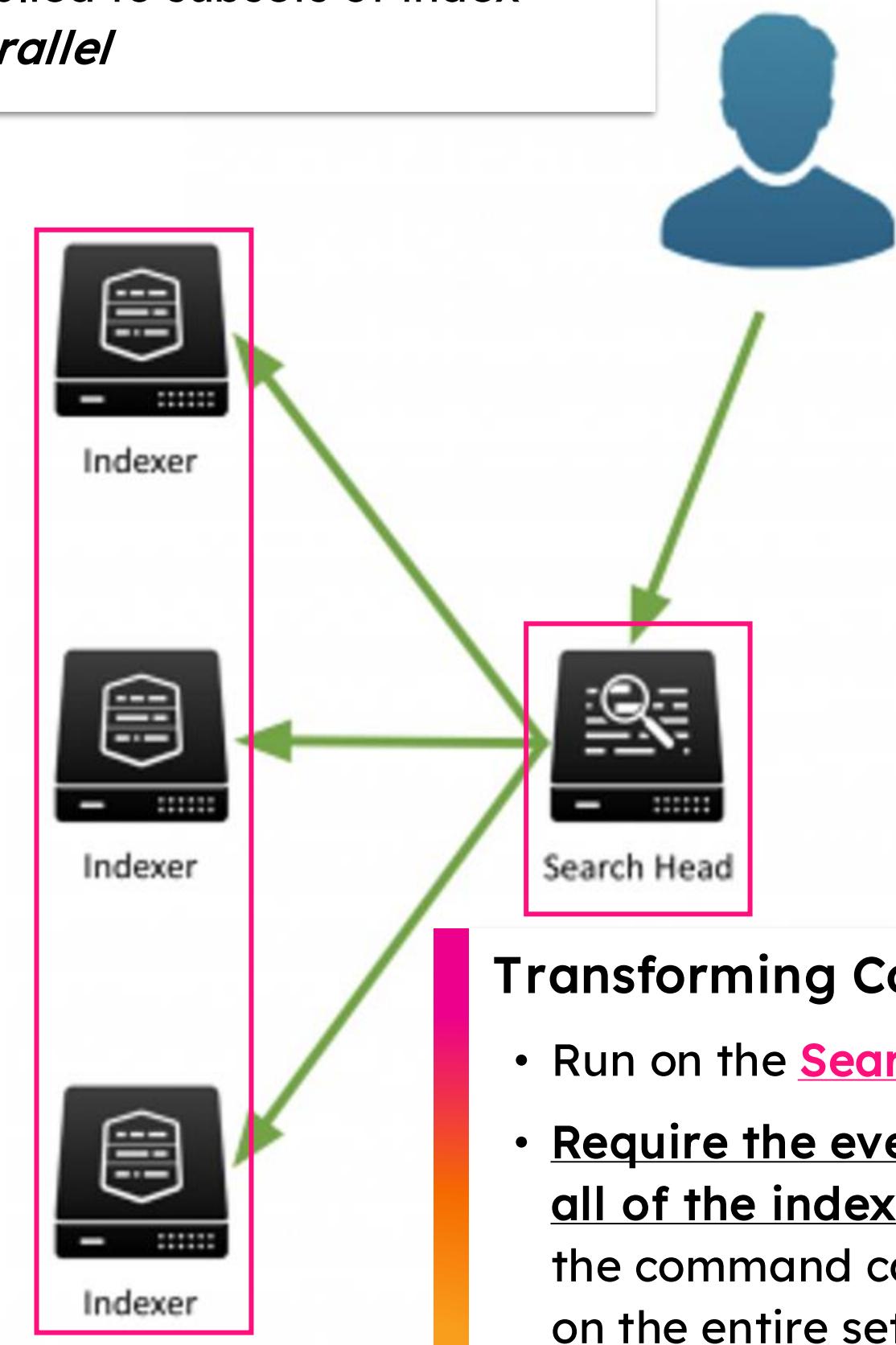
Learn more: [Types of Commands](#)

Streaming and Transforming Commands



Distributable Streaming Commands

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



Transforming Commands

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

A few examples

Distributable Streaming Commands

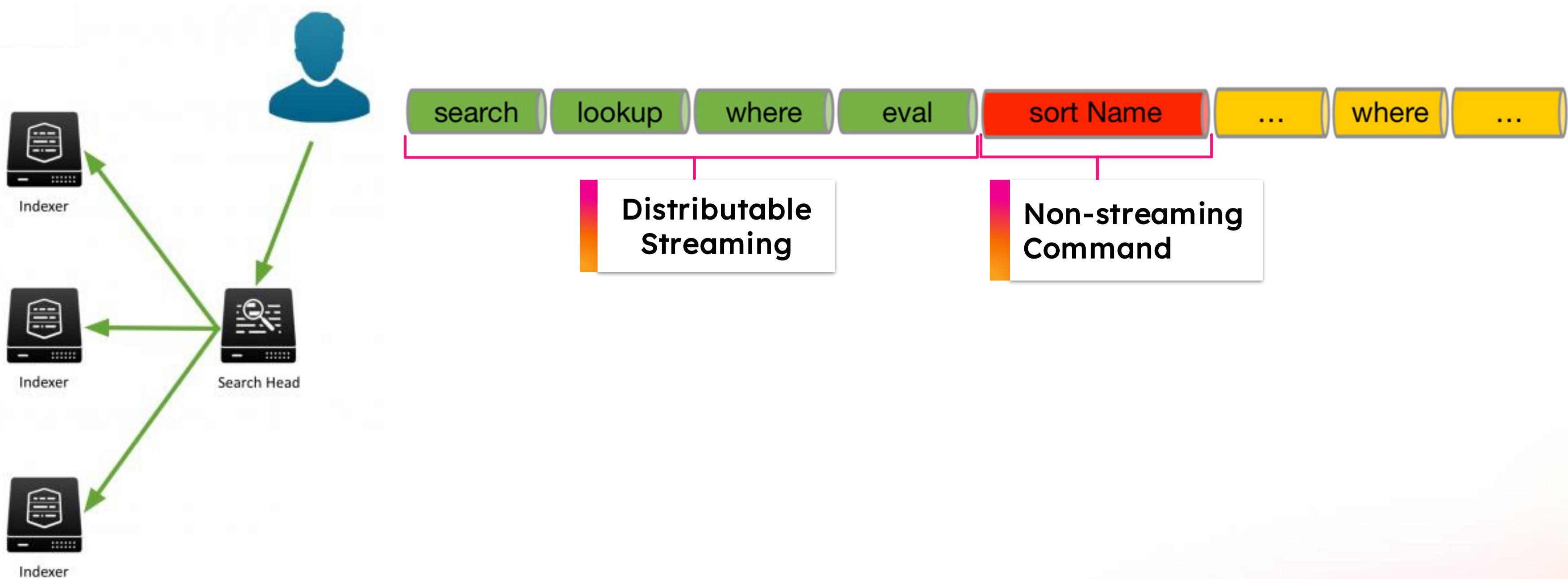
`eval`
`makemv`
`rex`
`spath`
`lookup`
`rename`

Transforming Commands

`timechart`
`chart`
`stats`
`table`
`top`
`rare`

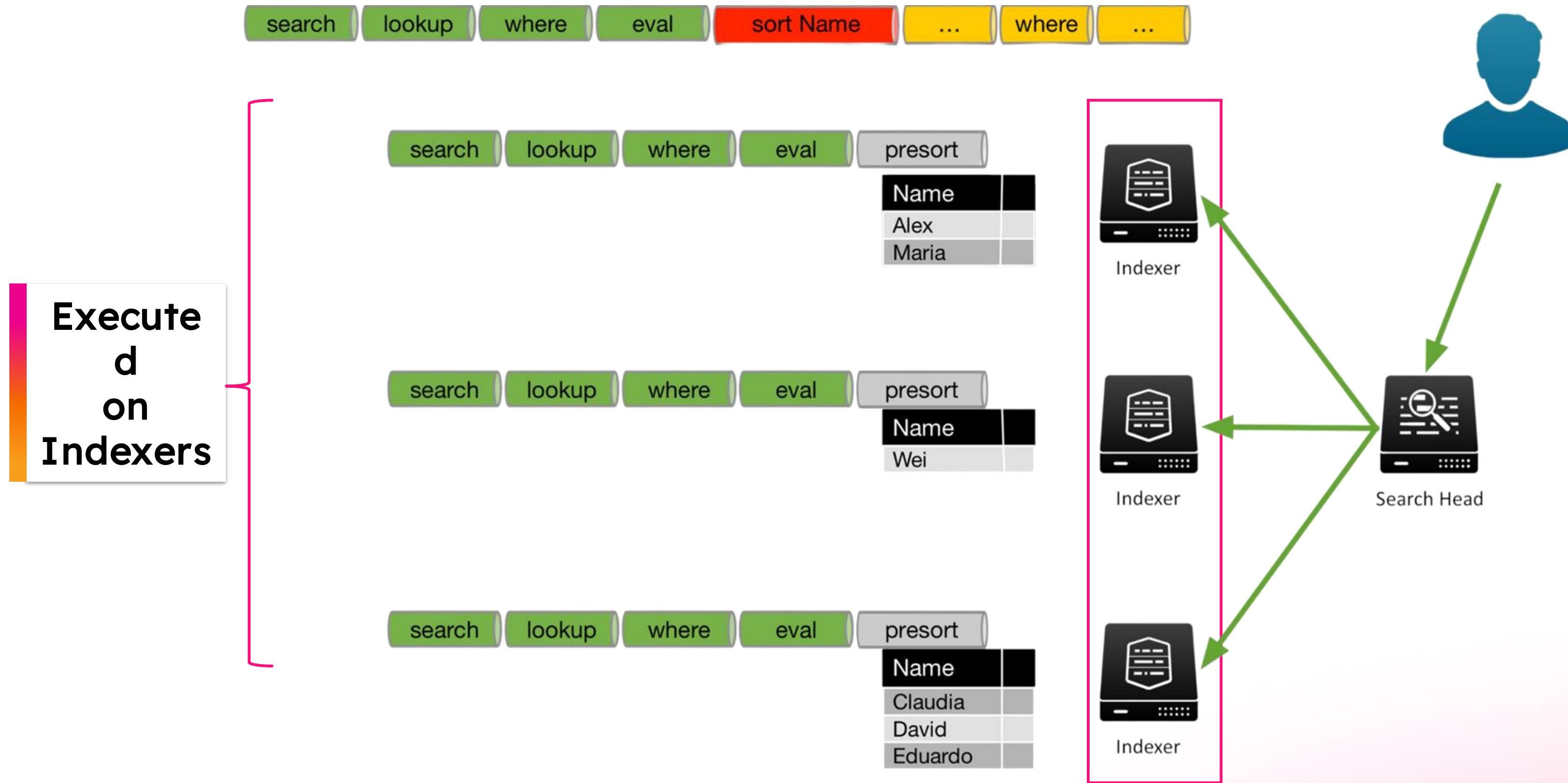
Search Processing Example

User search > Search Head > Distributed to Indexers



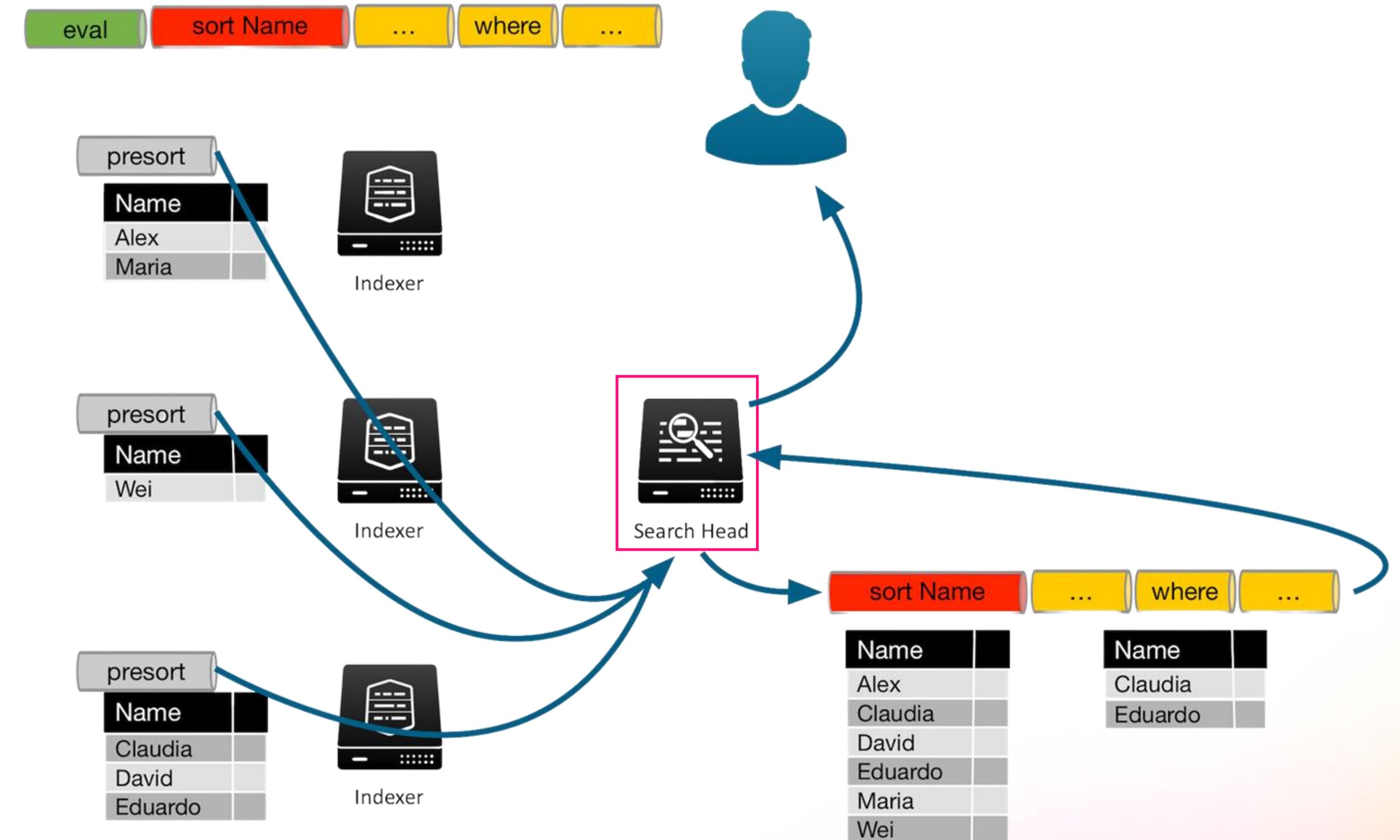
Learn more: [SPL Command "sort"](#)

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: [Write better searches](#)

SPL Command Types

1 - Find the command.

The **Command quick reference** page is a pretty good starting point.

- How to quickly find out what we're dealing with ?

Command Quick Reference

2 - Refer to the upper right index of the help page, section “Usage”



eval

Description

Syntax

Usage

Basic Examples

Extended Examples

See also

Command quick reference

The table below lists all of the search commands in alphabetical order. There is a short description of the command and links to related commands. For the complete syntax, usage, and detailed examples, click the command name to display the specific topic for that command.

Some of these commands share functions. For a list of the functions with descriptions and examples, see [Evaluation functions](#) and [Statistical and charting functions](#).

If you don't find a command in the table, that command might be part of a third-party app or add-on. For information about commands contributed by apps and add-ons, see the documentation on [Splunkbase](#).

Command	Description	Related commands
abstract	Produces a summary of each search result.	highlight
accum	Keeps a running total of the specified numeric field.	autoregress , delta , trendline , streamstats
addcoltotals	Computes an event that contains sum of all numeric fields for previous events.	addtotals , stats
addinfo	Add fields that contain common information about the current search.	search
addtotals	Computes the sum of all numeric fields for each result.	addcoltotals , stats
analyzefields	Analyze numerical fields for their ability to predict another discrete field.	anomalousvalue
anomalies	Computes an "unexpectedness" score for an event.	anomalousvalue , cluster , kmeans , outlier
anomalousvalue	Finds and summarizes irregular, or uncommon, search results.	analyzefields , anomalies , cluster , kmeans , outlier

3 - This is what we are looking for:

Usage

The `eval` command is a **distributable streaming command**. See [Command types](#).

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: [Quick Tips for Optimization](#)

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



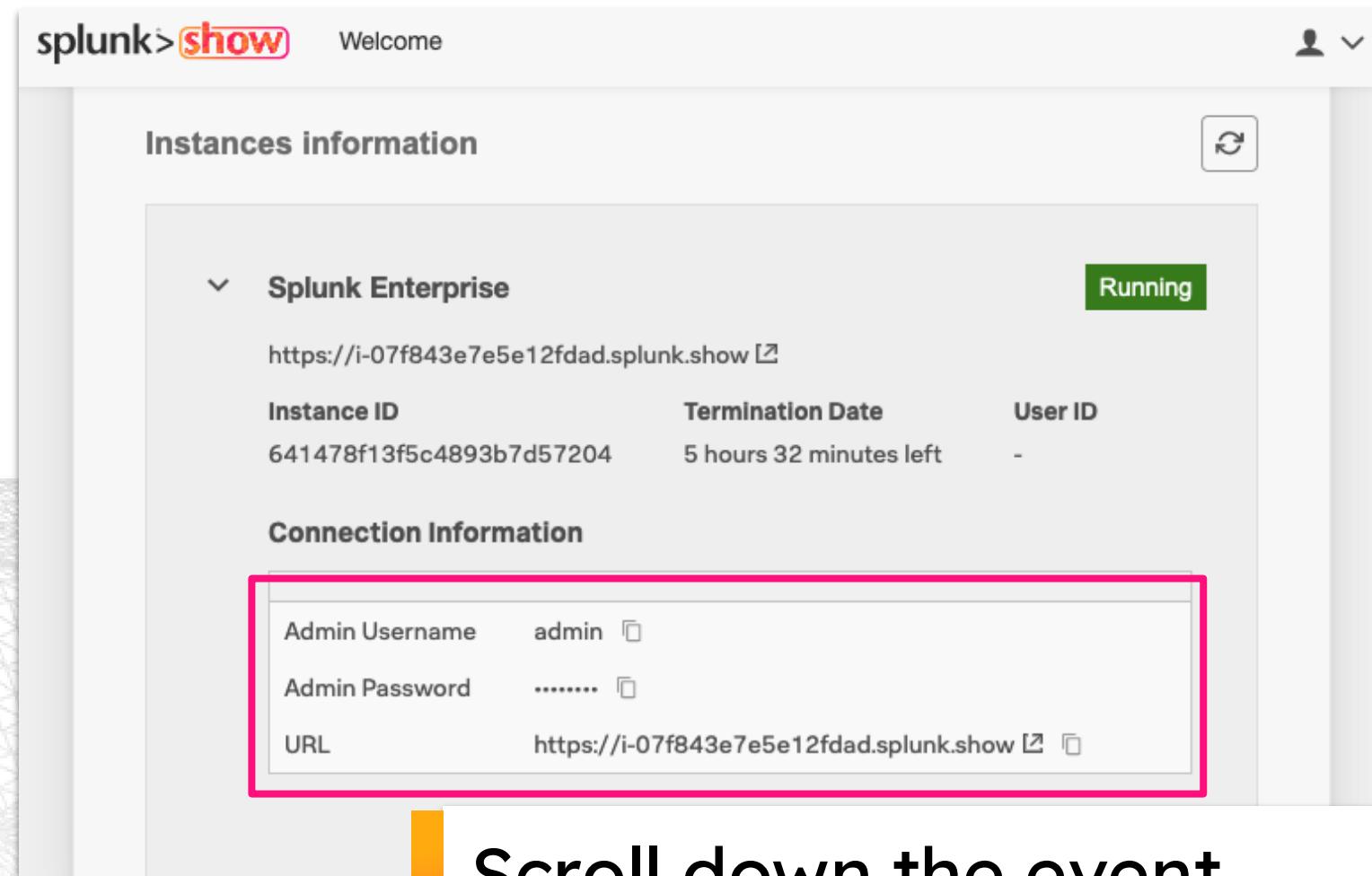
Be as specific as possible within search

```
index=my_index host=my_host sourcetype=my_src_type 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 }
```

Overrides the time-picker

Log in to Splunk

Locate your instance URL and credentials in the Splunk Show event
<https://show.splunk.com>



The screenshot shows the 'splunk>show' interface with the 'Welcome' message. Under 'Instances information', there is a section for 'Splunk Enterprise' which is 'Running'. It displays the instance URL: <https://i-07f843e7e5e12fdad.splunk.show>, an Instance ID: 641478f13f5c4893b7d57204, and a Termination Date: 5 hours 32 minutes left. Below this, under 'Connection Information', are fields for Admin Username (admin), Admin Password (redacted), and URL (<https://i-07f843e7e5e12fdad.splunk.show>). A red box highlights the connection information fields.

Scroll down the event page and expand the Splunk Enterprise section to view your login details

Log in to your Splunk instance

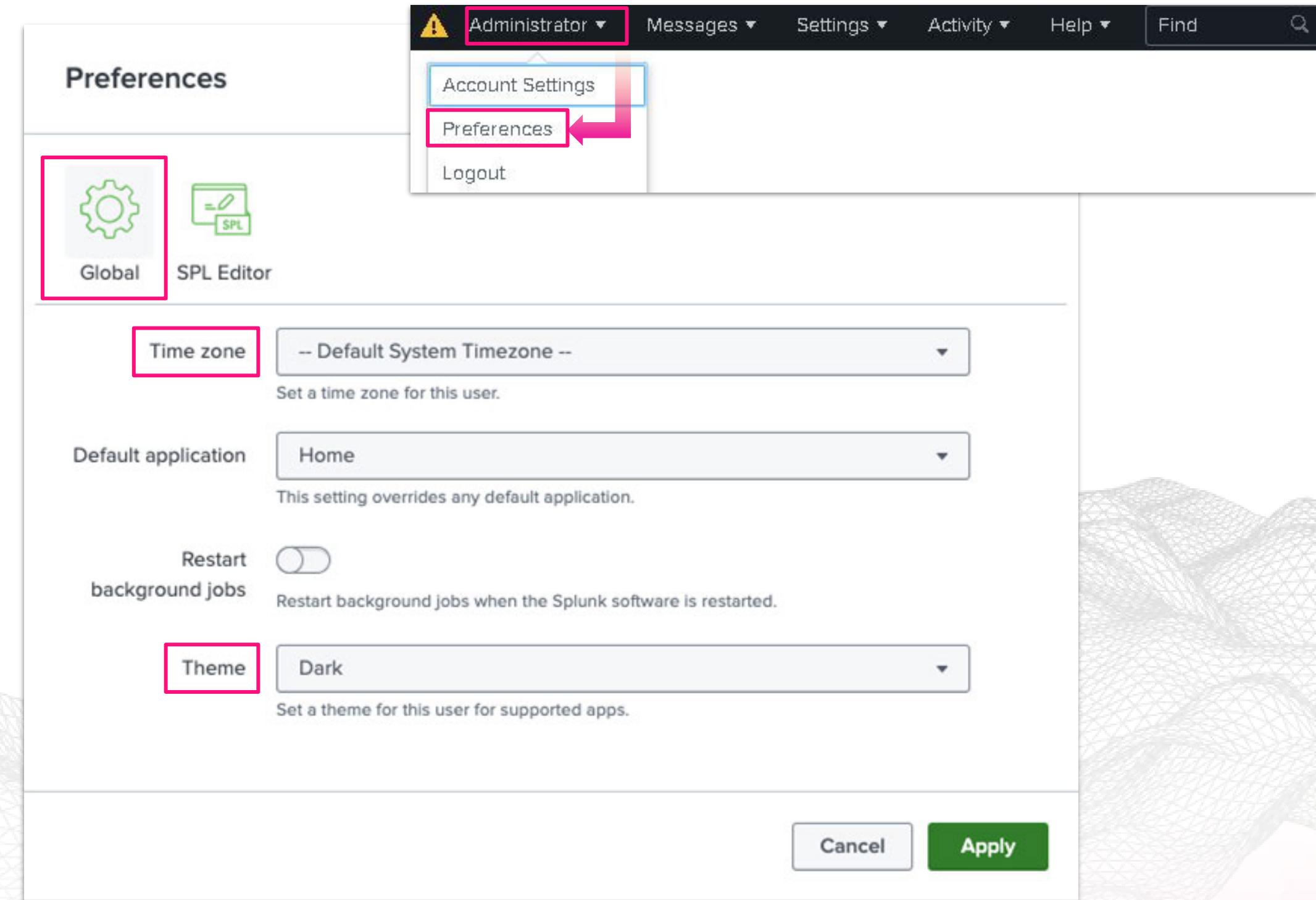


The screenshot shows a login screen for 'splunk>enterprise'. It features a large orange arrow pointing from the 'splunk>show' page to this screen. The login form has fields for 'Username' (admin) and 'Password' (redacted), and a 'Sign In' button. The background of the login screen is filled with log entries.

Login using the credentials from Splunk Show

Initial Set-up

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



Learn more: [How Time Zones are Processed by the Splunk Platform](#)

UI Improvements

The screenshot illustrates the SPL Editor Preferences dialog, which has been updated with several user interface improvements. At the top, the 'Administrator' dropdown is highlighted with a yellow warning icon. The 'Preferences' button in the top-left corner of the dialog is also highlighted with a pink border.

Left Panel (Original View):

- Global:** A gear icon.
- SPL Editor:** A document icon with a pencil and 'SPL' text.

The main content area displays a message about the advanced editor's auto-formatting features. Below it, there are two tabs: **General** (highlighted with a pink border) and **Themes**.

Search assistant: Options include **Full** (selected), **Compact**, and **None**. A note states: "Full search assistant is useful when first learning to create searches. Compact provides more succinct assistance."

Line numbers: An switch is turned off, with the note: "Shows numbers next to each line in the search syntax."

Search auto-format: An switch is turned on, with the note: "Automatically format search syntax to improve readability."

Buttons at the bottom: **Cancel** and **Apply**.

Right Panel (Improved View):

- Global:** A gear icon.
- SPL Editor:** A document icon with a pencil and 'SPL' text.

The main content area displays the same message about the advanced editor. Below it, the **Themes** tab is selected (highlighted with a pink border).

Search bar theme: Options include **Black on White**, **Light Theme**, and **Dark Theme** (selected).

Preview: A code snippet is shown in a dark-themed preview window:

```
sourcetype=access_* status=200 action=purchase  
| stats count AS "Total Purchased", dc(productId)  
    AS "Total Products", values(productName) AS  
    "Product Names" BY clientip  
| rename clientip AS "VIP Customer"
```

Buttons at the bottom: **Cancel** and **Apply**.

A large double-headed horizontal arrow is positioned between the two panels, indicating a comparison or migration from the original state to the improved state.

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.
```

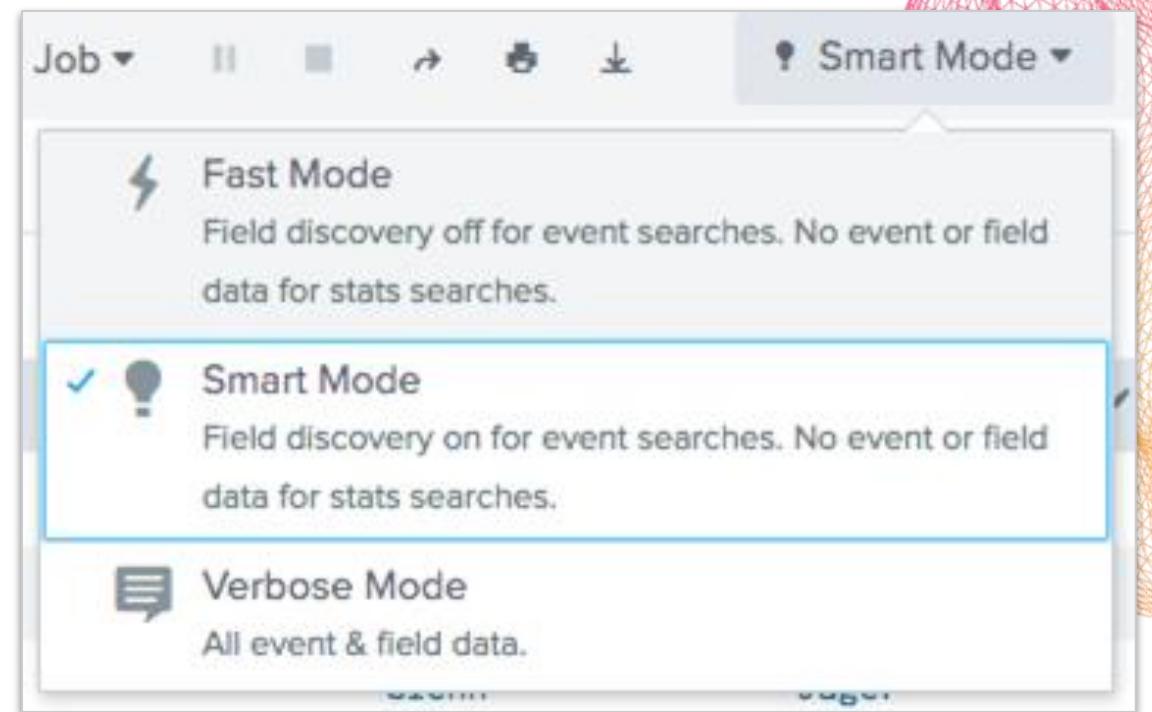
Useful for:

- 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: [Splunk's Search Modes](#)

Tips & Tricks for Writing More Efficient Searches



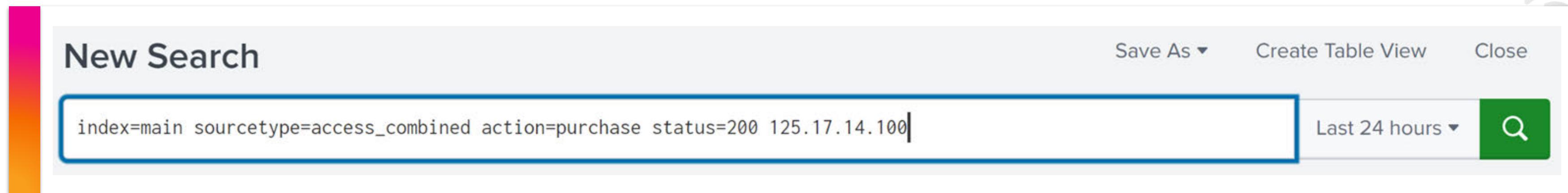
SPL# >

**Dealing with
delimiters and
measuring
search
performance**



SPL# > Counting successful Purchase Actions for a specific IP address

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



Measuring the Search Execution Time

Inspect Job > check the Search Summary

The screenshot shows the Splunk interface. On the left, there is a sidebar with options: Sampling ▾, Job ▾ (highlighted with a red arrow), II, Edit Job Settings..., Send Job to Background, Inspect Job (highlighted with a blue border), and Delete Job. Below the sidebar is a decorative graphic of concentric circles.

The main area is titled "Search job inspector". It displays the message: "This search has completed and has returned 397 results by scanning 931 events in 0.205 seconds" (SID: 1621398493.191) [search.log](#). A callout box on the right says "Check the search duration".

A table titled "Execution costs" provides detailed timing information:

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	-	-
0.00	command.search.calcfIELDS	3	931	931

Raw Text Searches in Splunk are great!

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

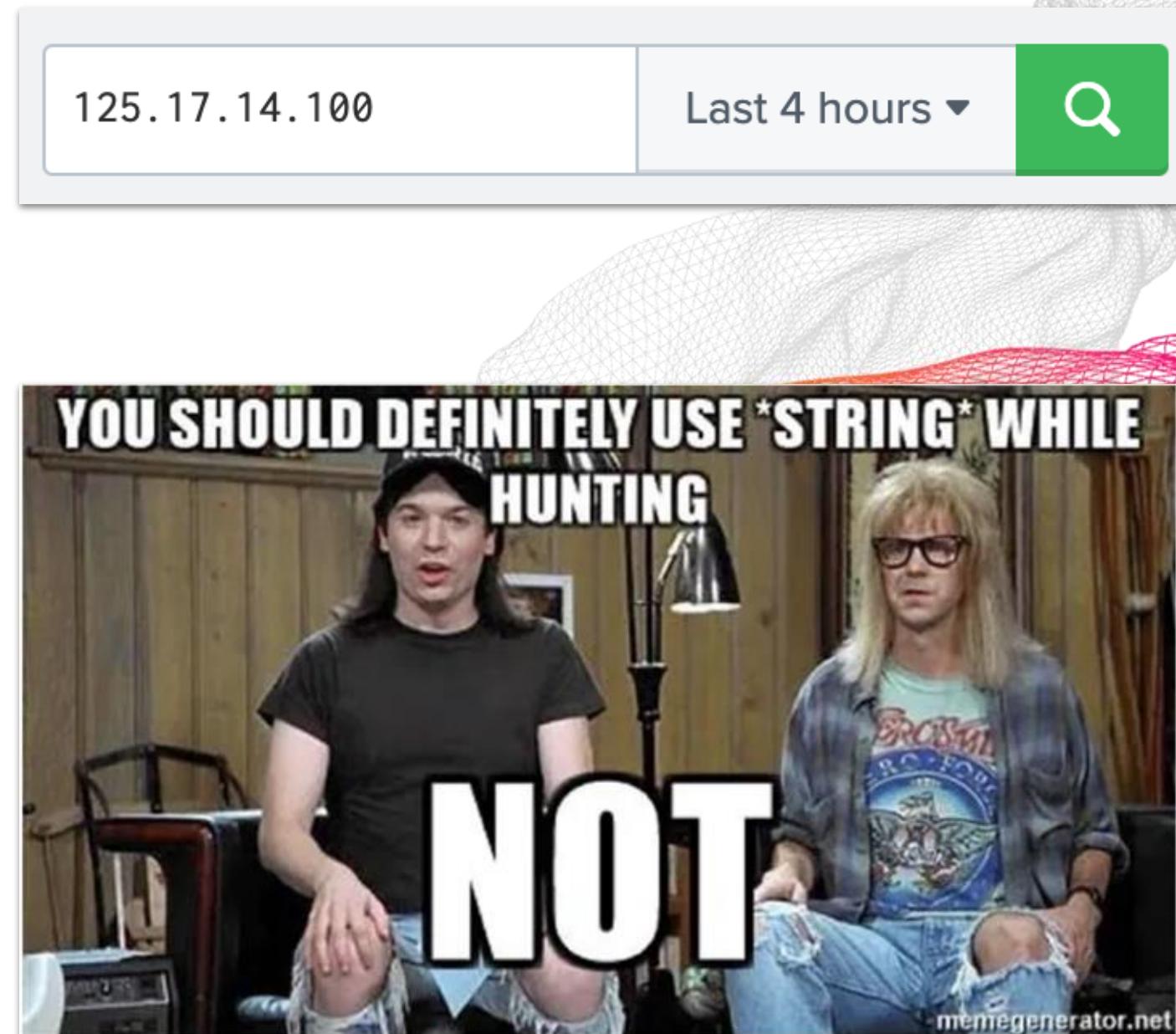
Let's see what happens when we ask Splunk to Search for an IP address.

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 during 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 125.17.14.100 - - [20/Aug/2024 08:20:54]

Major segment breaker
(space is the breaker)

125.17.14.100
-
-
20/Aug/2024
08:20:54

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

125
17
14
100
125.17
125.17.14
125.17.14.100
17.14
and so forth

Minor segment breaker
dot (.) breaks it further

Learn more: [About Event Segmentation](#)

How is Data being Stored ?

raw data + dictionary a.k.a. the
(as well as couple other things we won't detail today)

```
125.17.14.100 - - [22/Oct/2024 06:55:45] "POST /product.screen?uid=100dea6a-2e3f-489b-901  
12.130.60.5 - - [22/Oct/2024 06:55:45] "GET /product.screen?uid=a9564fea-1f07-451d-9332-9  
12.130.60.4 - - [22/Oct/2024 06:55:45] "GET /product.screen?uid=4de2e850-8fd0-4451-94e4-2  
27.35.11.11 - - [22/Oct/2024 06:55:45] "POST /product.screen?uid=046182fc-e40a-47a4-b39c-  
125.17.14.100 - - [22/Oct/2024 06:55:45] "GET /product.screen?uid=7b0a0ca4-96f8-4d85-b462-  
195.80.144.22 - - [22/Oct/2024 06:55:45] "POST /product.screen?uid=cf721ea8-0c3d-46d8-9b4  
12.130.60.5 - - [22/Oct/2024 06:55:45] "GET /cart.do?action=remove&product_id=DFS-2&JSESSION  
141.146.8.66 - - [22/Oct/2024 06:55:45] "GET /cart.do?action=changequantity&product_id=M  
84.34.159.23 - - [22/Oct/2024 06:55:45] "GET /product.screen?uid=a2e15361-de09-4fe4-82ae-  
125.17.14.100 - - [22/Oct/2024 06:55:45] "GET /product.screen?uid=6d3e3d04-1532-483c-9752-  
69.80.0.18 - - [22/Oct/2024 06:55:45] "GET /cart.do?action=view&product_id=PP-5&JSESSION  
201.122.42.235 - - [22/Oct/2024 06:55:46] "GET /product.screen?uid=f3104afb-bfd4-49fb-938  
130.253.37.97 - - [22/Oct/2024 06:55:46] "GET /product.screen?uid=c7a82d76-e6c3-4ea6-95a8-  
141.146.8.66 - - [22/Oct/2024 06:55:46] "GET /product.screen?uid=c304dc64-d2a2-4d32-963e-  
125.17.14.100 - - [22/Oct/2024 06:55:46] "GET /product.screen?uid=6e231484-0dbd-4fce-ae40-
```

LEXICON

Term	Postings List
GET	1,2,3,5...
POST	0,3,...
125	0,4,9,14,...
17	0,4,9,14,...
14	0,4,9,14,...
100	0,4,9,14,...
125.17	0,4,9,14,...
125.17.14	0,4,9,14,...
125.17.14.100	0,4,9,14,...
17.14	0,4,9,14,...
17.14.100	0,4,9,14,...

Learn more: [How the Indexer Stores Indexes](#)

Introducing LISPY

LISPY refers to a type of **internal query syntax** used by Splunk Search Processing Language (SPL) behind the scenes. LISPY stands for "LISP-like." It represents the way Splunk structures its search requests when they are processed internally.

It is not typically something users interact with directly, but understanding LISPY can be helpful for advanced troubleshooting and when debugging complex search queries.

Key points about LISPY:

- **LISP-like syntax:** The structure resembles LISP programming language, which uses nested lists and parentheses to represent function calls and operations.
- **Internal representation:** SPL queries are converted into LISPY format by Splunk to be processed more efficiently by the search engine.
- **Advanced usage:** It is mainly useful for Splunk administrators and developers who need to dive deep into query execution or troubleshoot search performance issues.

Optimising your search for optimal LISPY can significantly reduce the workload on the indexers by limiting the quantity of data processed.

LISPY mentions found in the wild:

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

(.Conf talk by Richard Morgan: "[TSTATS and PREFIX](#)", Slide 14,)

"LISPY expressions are predicates Splunk uses to locate events"

(.Conf talk by Martin Mueller: "[Fields, Indexed Tokens and You](#)", Slide 11,)

Searching for Raw data

What happens when the search string contains Segment Breakers

The screenshot shows the Splunk interface with a search bar at the top containing "index=main 125.17.14.100". Below the search bar, it says "3,065 events (20/08/2024 07:13:00.000 to 20/08/2024 08:13:41.000)". A dropdown menu is open over the "Job" button, with "Inspect Job" highlighted. To the right, a modal window titled "Search job inspector" displays the message: "This search has completed and has returned (SID: 1724141621.27)" followed by links "search.log" and "Job Details".

Events (3,065) Patterns Statistics Visualization

Format Timeline - Zoom Out + Zoom to Selection X Deselect

index=main 125.17.14.100 Last 60 minutes Q

✓ 3,065 events (20/08/2024 07:13:00.000 to 20/08/2024 08:13:41.000)

No Event Sampling ▾

Job ▾

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

Search job inspector

This search has completed and has returned
(SID: 1724141621.27) [search.log](#) [Job Details](#)

And here is how the LISPY for our IP Address looks in the Job Inspector.
Don't you notice anything weird?

Expanded index search = '(125.17.14.100 index=main)
base lispy: [AND 100 125 14 17 index::main]'

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

Learn more: [Event Segmentation and Searching](#)

SPL# >

Exploring the **TERM** directive



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



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: [Use CASE\(\) and TERM\(\) to Match Phrases](#)

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)

All time Job Events (12,729) Patterns Statistics Visualization

Save As ▾ Create Table View Close

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

Events (12,729) 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

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'ts 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: [Segmenters.conf](#)
- > CASE and TERM: [Use CASE\(\) and TERM\(\) to Match Phrases](#)

Explore the 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)

TERM - The easy way

a.k.a. “Hovering Method”

Try to run a search and move your mouse slowly over the raw events.

Whatever is under the cursor will change color segment by segment, encompassing minor breakers.

The boundaries of the biggest continuous block of data that can be highlighted at once are major breakers, and therefore a good fit for the TERM directive!

i	Time	Event
>	14/10/2024 12:04:52.000	200.6.134.23 - - [14/Oct/2024 12:04:52] "GET /product.screen?uid=0af7b10f-c0f7-4e51-bd7e-5e0df0db9122&product_id=PP-5" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_1) AppleWebKit/602.2.14 Version/10.0.1 Sa host = Domane-Demo- source = /var/log/weblogs/noise_apache.log24 sourcetype = access_combined

It is now as simple as copy pasting the chosen segment into your search:

```
index=main sourcetype=access_combined TERM(uid=0af7b10f-c0f7-4e51-bd7e-5e0df0db9122)
```



No need to remember by heart the list of major and minor breakers :-D

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: [About Search Optimization](#)

SPL# > Tstats

Stats is good, but

Tstats is...

Better ?

Prettier ?

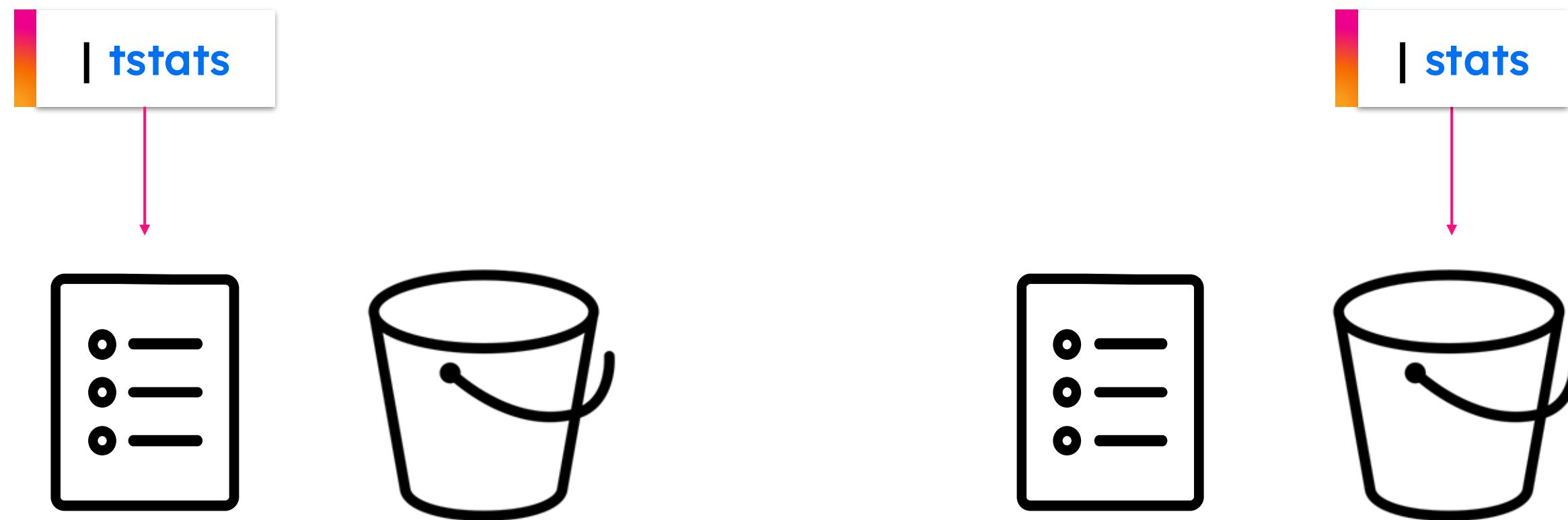
Easier ?

---> FASTER!



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: [SPL Command - "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
 - The existence of **TERMS** can be inferred from log data

```
| walklex index=main type=field  
| stats values(field) AS indexed_fields
```

Learn more: [SPL Command - "walklex"](#)

**SPL# > Tstats the
easy way**

=

**with
a Data Model**

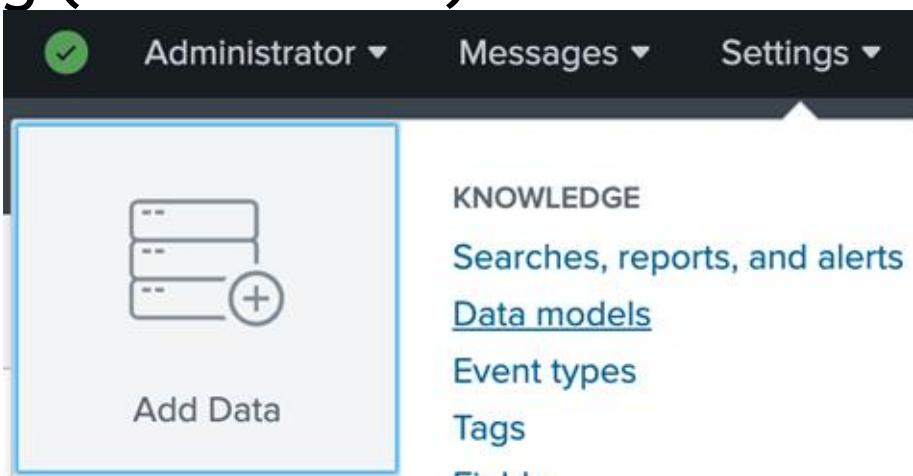


Tstats with a Data Model

We added to our sandboxes a Data Model called

Buttercup / online_store

It simply reuses the existing (search time) fields from the main index



[Data Models deserve a dedicated session, so we won't get too much into details here.]

In case you want to catch up, we can recommend:

[Data Models Education Course](#)

[Splunk Workshops](#)

Splunk4Ninjas - Common Information Model

Splunk4Ninjas - Data Onboarding]

Datasets	online_store
EVENTS	online_store
CONSTRAINTS	
index=main	
INHERITED	
_time	
host	
source	
sourcetype	
EXTRACTED	
action	
bytes	
category	
clientip	
file	
JSESSIONID	
method	
product_id	
product_name	
product_price	
referer	
referer_domain	
status	

Tstats with a Data Model

It's easy to explore our **Buttercup** data model: just use the **from** command + ":" + Data Model name:

The screenshot shows the Splunk search interface. The search bar at the top contains the command `from datamodel:Buttercup`. Below the search bar, it says "36,384 events (10/04/2025 11:55:00.000 to 10/04/2025 12:55:40.000)" and "No Event Sampling". The main area displays a timeline from 10/04/2025 11:55:00.000 to 10/04/2025 12:55:40.000, with a green bar representing the event count. Below the timeline, there are tabs for "Events (36,384)", "Patterns", "Statistics", and "Visualization". The "Events" tab is selected. At the bottom, there are buttons for "Format Timeline", "Zoom Out", "Zoom to Selection", and "Deselect". The event list table has columns for "Time" and "Event". The sidebar on the left lists "SELECTED FIELDS" (host 1, source 1, sourcetype 1) and "INTERESTING FIELDS" (action 5, # bytes 100+, category 3, clientip 67, file 2, JSESSIONID 100+, method 2, product_id 10, product_name 10, product_price 7, referer 10, referer_domain 1, status 9, uid 100+, uri 100+, uri_path 2, uri_query 100+, user 1, useragent 22, version 1).

INTERESTING FIELDS

a action 5
bytes 100+
a category 3
a clientip 67
a file 2
a JSESSIONID 100+
a method 2
a product_id 10
a product_name 10
product_price 7
a referer 10
a referer_domain 1
status 9
a uid 100+
a uri 100+
a uri_path 2
a uri_query 100+
a user 1
a useragent 22
version 1



All the fields are there, we can run any search as usual. Pretty boring...

So what about **tstats** then?

Tstats with a Data Model

This is where it becomes interesting: you can use **tstats** with all the **fields** declared in your DM.

The **syntax (and order)** are a bit different but it shouldn't be too difficult to figure out:

A screenshot of a Splunk search interface. The search bar contains the following command:

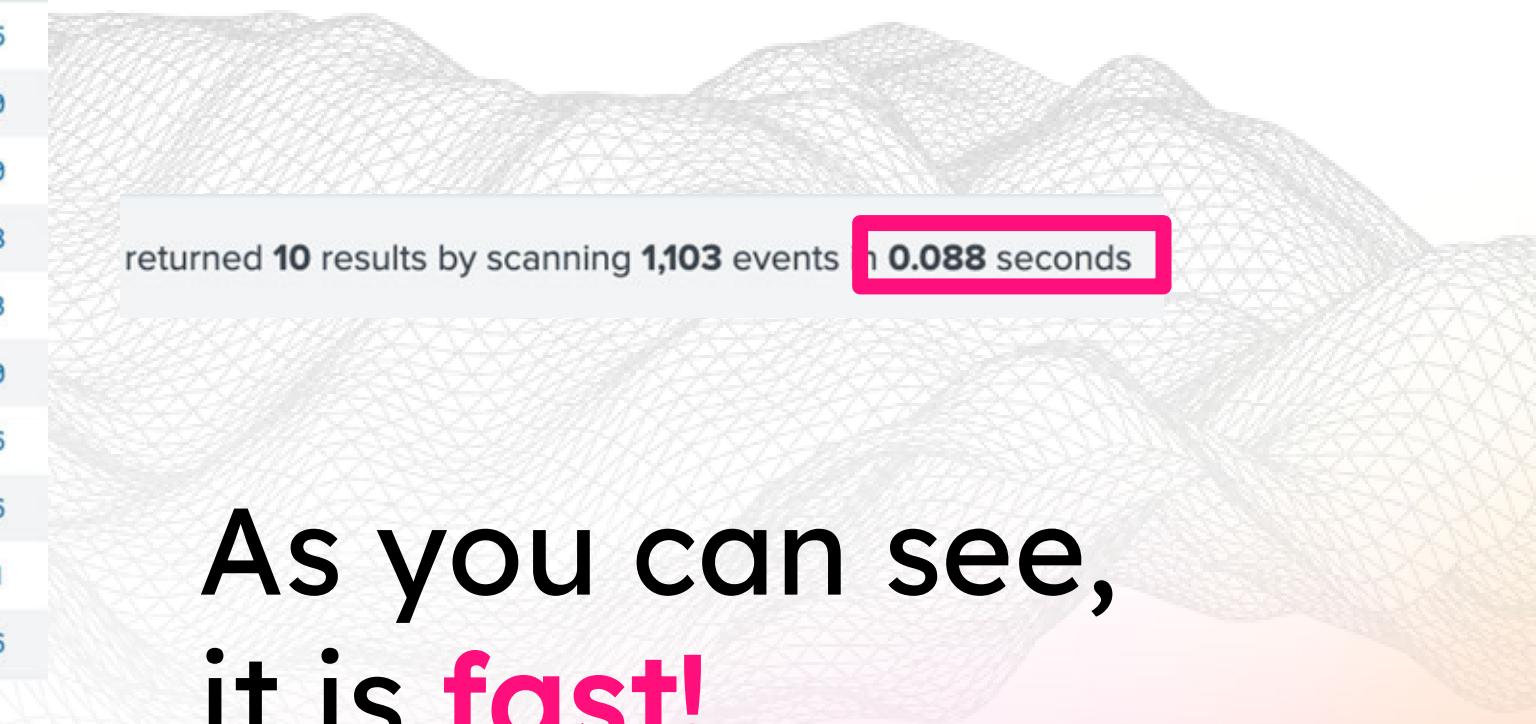
```
| tstats count as purchase_count sum(online_store.product_price) as total_sales from datamodel=Buttercup  
where online_store.action=purchase AND online_store.status<"400" by online_store.category online_store.product_name  
| rename online_store.* as *\n| sort - total_sales
```

The search results table has the following columns: category, product_name, purchase_count, and total_sales. The results show various products and their sales figures. A pink arrow points from the text "This search will tell us what product is selling the best:" to the "total_sales" column header in the table.

category	product_name	purchase_count	total_sales
Clothing	Costume- ManHawk	87	8482.5
Clothing	Batguy Slippers	100	2570.0
Gifts	Double Fudge Sundae	82	1865.50
Books	Mad Comics- Flyman	124	1574.8
Books	Zombie Survival Guide	93	1414.53
Books	Mad Comics- Bronze Man	100	1270.0
Books	Mad Comics- Batguy	98	1244.6
Clothing	Batguy Watch	104	1038.96
Gifts	Pony Potpourri	89	889.11
Gifts	Waterproof Scratch and Sniff	104	518.96

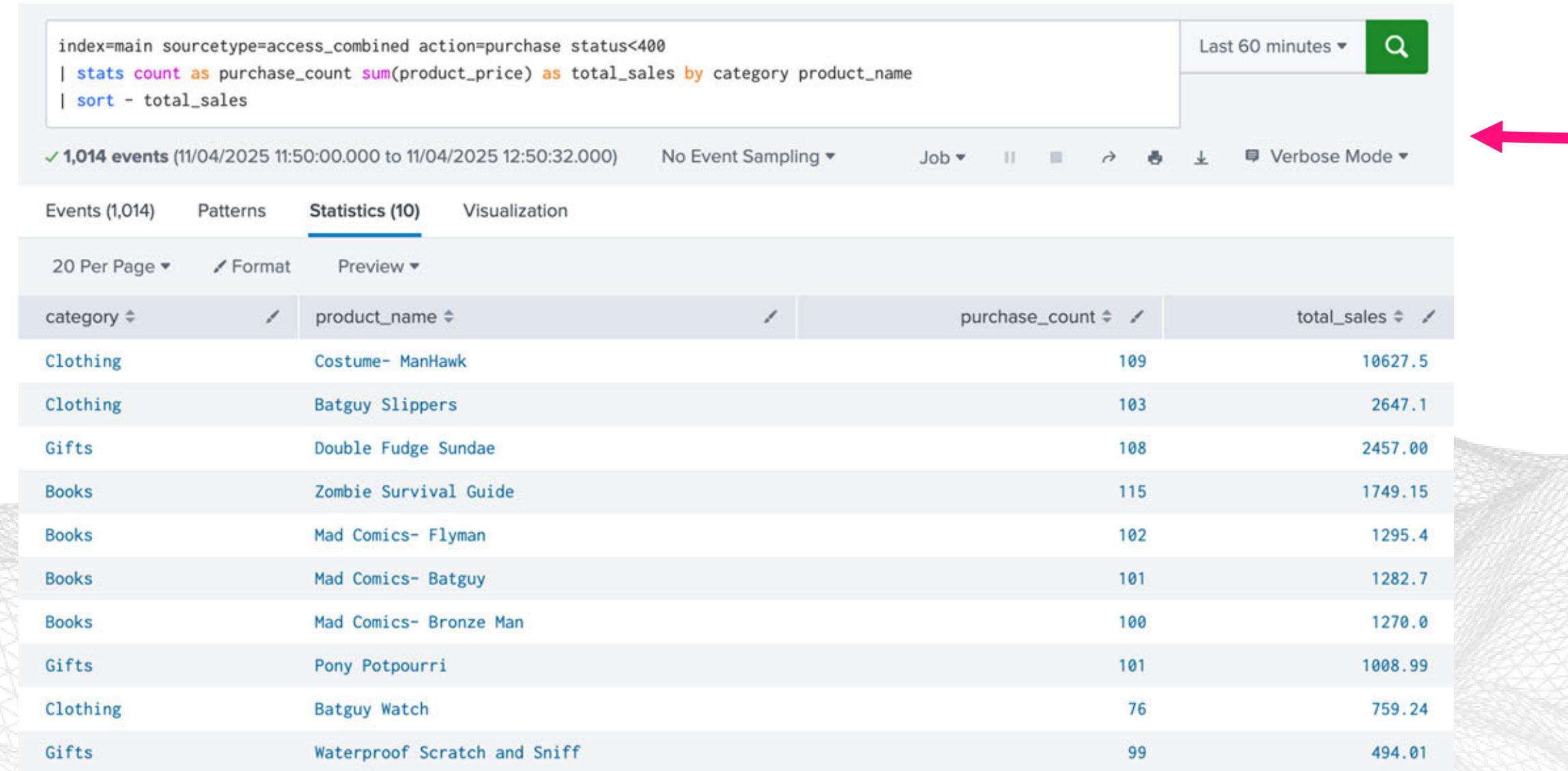
This search will tell us what product is selling the best:

- Aggregations first (**count, sum**)
- Then the data source (**from**)
- Then the filter (**where**)



Tstats with a Data Model (spoiler: stats is slower)

As you will find out, the same search with stats is significantly slower:



The screenshot shows a Splunk search interface. The search bar contains the following command:

```
index=main sourcetype=access_combined action=purchase status<400  
| stats count as purchase_count sum(product_price) as total_sales by category product_name  
| sort - total_sales
```

The search results table has four columns: category, product_name, purchase_count, and total_sales. The results are as follows:

category	product_name	purchase_count	total_sales
Clothing	Costume- ManHawk	109	10627.5
Clothing	Batguy Slippers	103	2647.1
Gifts	Double Fudge Sundae	108	2457.00
Books	Zombie Survival Guide	115	1749.15
Books	Mad Comics- Flyman	102	1295.4
Books	Mad Comics- Batguy	101	1282.7
Books	Mad Comics- Bronze Man	100	1270.0
Gifts	Pony Potpourri	101	1008.99
Clothing	Batguy Watch	76	759.24
Gifts	Waterproof Scratch and Sniff	99	494.01

A bit simpler to write, true.
But look at the performance.

With bigger datasets, leveraging data model acceleration with tstats is well worth the effort!

returned 10 results by scanning 1,847 events in 0.145 seconds

tstats cannot be matched when it comes to speed...



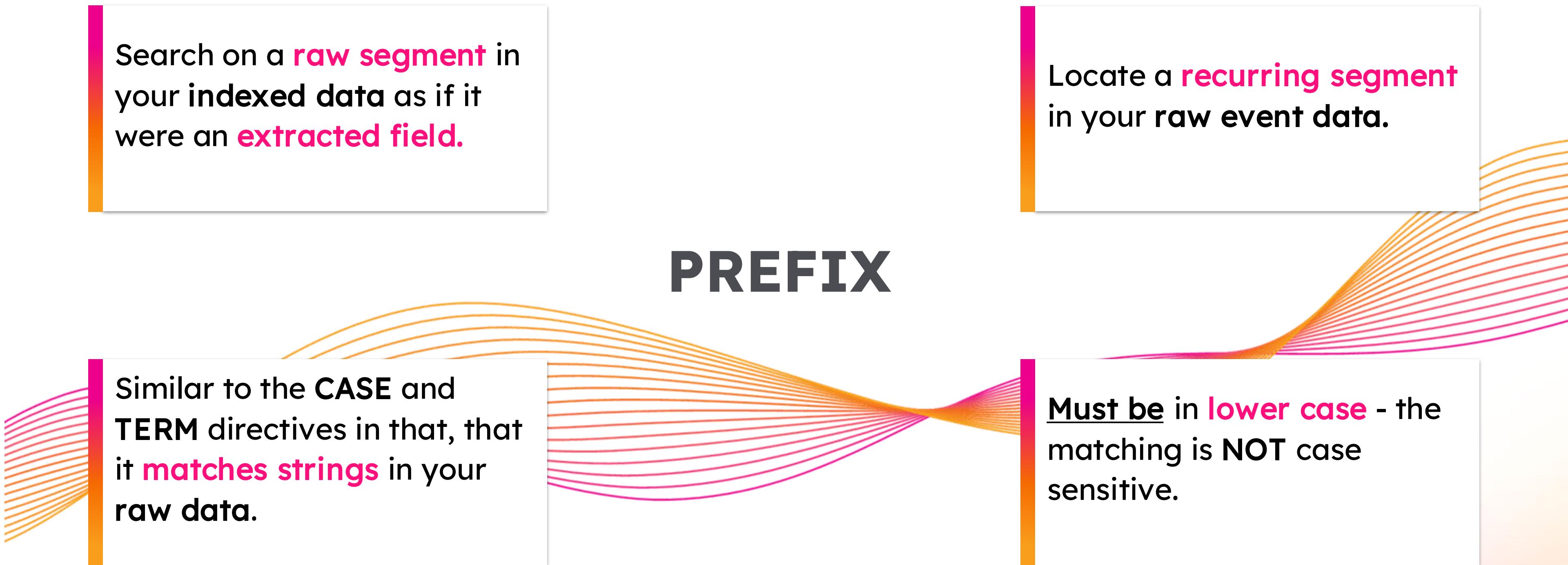
SPL# >

**Time for the
Ninja Trick!**

**tstats with
PREFIX**



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**.

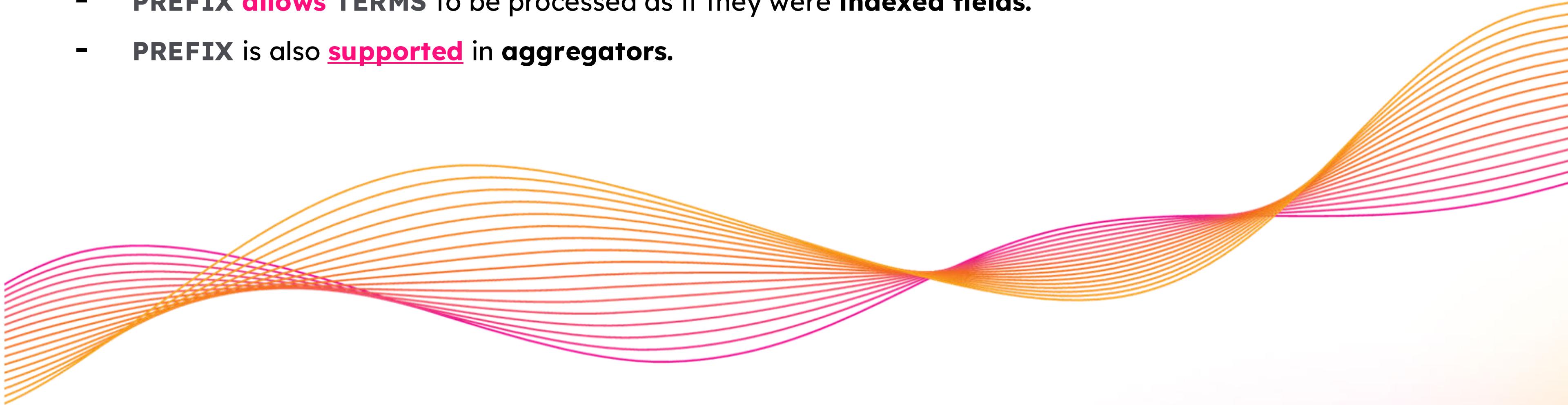
Similar to the **CASE** and **TERM** directives in that, that it **matches strings** in your **raw data**.

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

tstats with PREFIX - why does it matters



- 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: [TSTATS and PREFIX](#)

tstats + prefix in action

index=main sourcetype=access_combined action=purchase
| stats count by JSESSIONID

Last 24 hours

Let's pick a simple use case:
Counting the number of purchase events for each user session over 24h.
Keep the top ten.

JSESSIONID	Count
SD5SL9FF7ADFF7	16
SD1SL4FF5ADFF10	15
SD3SL3FF9ADFF7	15
SD5SL8FF9ADFF8	15
SD6SL2FF10ADFF2	15
SD8SL2FF9ADFF6	15
SD10SL8FF6ADFF2	14
SD10SL9FF3ADFF4	14

First we will try a regular **stats query with our sample dataset over 24h.**

It should take about a second.

The goal is to get the **results, but also to see the raw events**

(Verbose Mode recommended)

The screenshot shows the Splunk Job Inspector interface. At the top, a search bar contains the following SPL command:

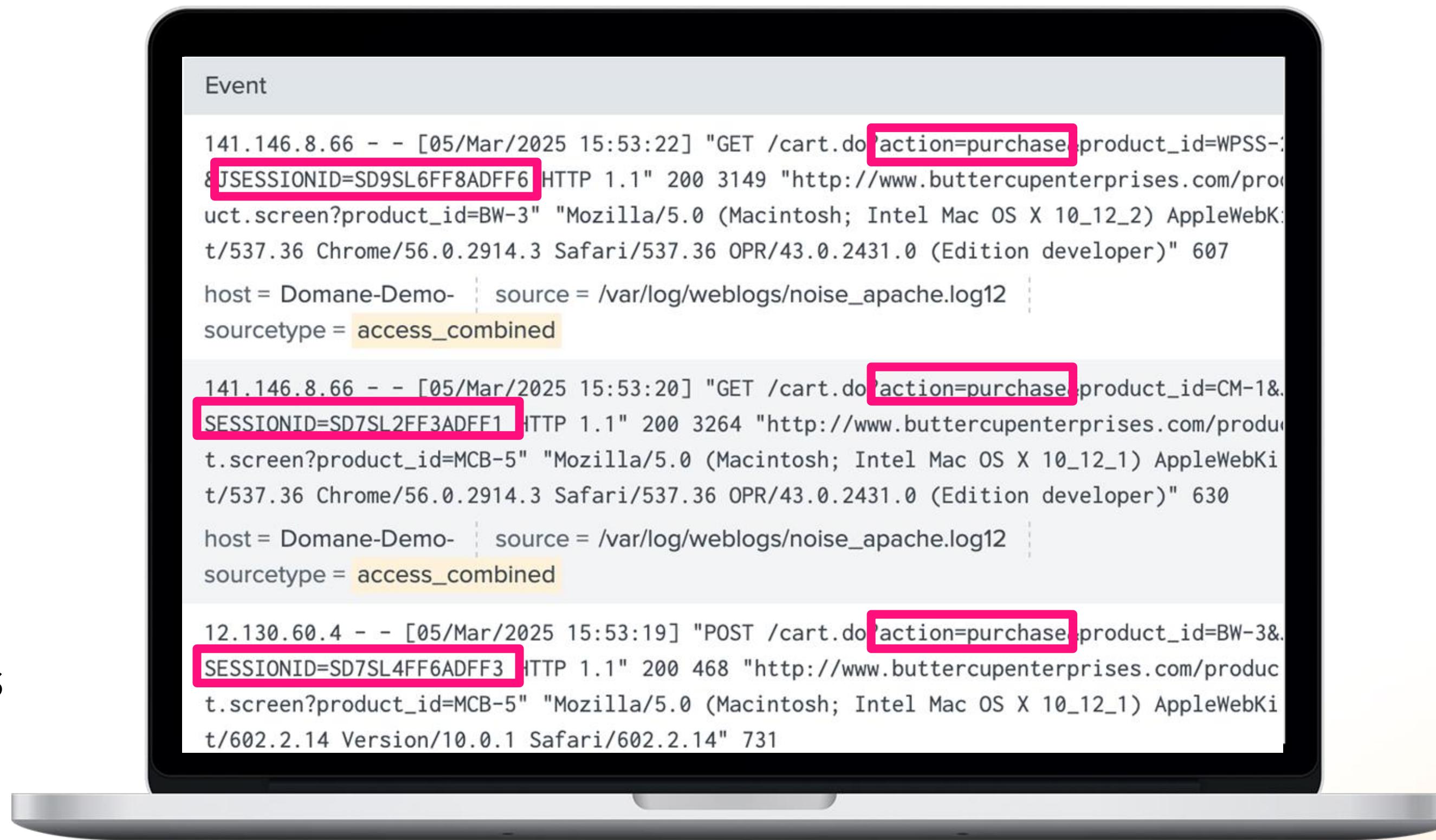
```
index=main sourcetype=access_combined action=purchase  
| stats count by JSESSIONID  
| sort - count  
| head 10
```

Below the search bar, the status is "41,798 events (20/04/2025 20:00:00.000 to 21/04/2025 20:17:33.000)" and "No Event Sampling". The "Statistics (10)" tab is selected. Under "Format", "Preview: On" is chosen. The results table lists 10 JSESSIONID entries with their respective counts:| JSESSIONID | count |
| --- | --- |
| SD8SL7FF6ADFF4 | 18 |
| SD1SL1FF4ADFF10 | 16 |
| SD5SL9FF7ADFF7 | 16 |
| SD1SL4FF5ADFF10 | 15 |
| SD3SL3FF9ADFF7 | 15 |
| SD5SL8FF9ADFF8 | 15 |
| SD6SL2FF10ADFF2 | 15 |
| SD8SL2FF9ADFF6 | 15 |
| SD10SL8FF6ADFF2 | 14 |
| SD10SL9FF3ADFF4 | 14 |

At the bottom of the interface, a message states: "Search job inspector" and "This search has completed and has returned 10 results by scanning 41,798 events in 1.083 seconds". Below this, it says "(SID: 1745266653.2145) [search.log](#) [Job Details Dashboard](#)".

Next, we try to identify recurring segments, all while paying attention to the major breakers (same hovering technique as with the **TERM directive).**

This seems to work here. In real life this is not always the case...



The image shows a smartphone displaying a Splunk search results page. The screen is titled "Event" and contains three log entries. Each entry is highlighted with a pink rectangular box around the "action=purchase" part of the URL. The first two entries are GET requests, and the third is a POST request. The log entries include IP addresses, timestamps, URLs, user agents, and host information.

```
Event

141.146.8.66 - - [05/Mar/2025 15:53:22] "GET /cart.do?action=purchase;product_id=WPSS-1&JSESSIONID=SD9SL6FF8ADFF6 HTTP 1.1" 200 3149 "http://www.buttercupenterprises.com/product.screen?product_id=BW-3" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_2) AppleWebKit/537.36 Chrome/56.0.2914.3 Safari/537.36 OPR/43.0.2431.0 (Edition developer)" 607
host = Domane-Demo- | source = /var/log/weblogs/noise_apache.log12
sourcetype = access_combined

141.146.8.66 - - [05/Mar/2025 15:53:20] "GET /cart.do?action=purchase;product_id=CM-1&SESSIONID=SD7SL2FF3ADFF1 HTTP 1.1" 200 3264 "http://www.buttercupenterprises.com/product.screen?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)" 630
host = Domane-Demo- | source = /var/log/weblogs/noise_apache.log12
sourcetype = access_combined

12.130.60.4 - - [05/Mar/2025 15:53:19] "POST /cart.do?action=purchase;product_id=BW-3&SESSIONID=SD7SL4FF6ADFF3 HTTP 1.1" 200 468 "http://www.buttercupenterprises.com/product.screen?product_id=MCB-5" "Mozilla/5.0 (Macintosh; Intel Mac OS X 10_12_1) AppleWebKit/602.2.14 Version/10.0.1 Safari/602.2.14" 731
```

Finally,

We use the **fields recurring segments we located earlier to construct our query.**

Filters go inside a **TERM directive, Aggregations & Group By fields inside a **PREFIX****

Search time reduced to ~0.157 seconds

Mission Accomplished !

```
| tstats count where index=main sourcetype=access_combined TERM(action=purchase) by PREFIX(jsessionid=)
| rename jsessionid= as JSESSIONID
| eval JSESSIONID=upper(JSESSIONID)
| sort - count
| head 10
```

42,281 events (20/04/2025 20:00:00.000 to 21/04/2025 20:34:19.000) No Event Sampling ▾ Job ▾ Last 24 hours ▾ Verbose Mode ▾

Events (42,281) Patterns Statistics (10) Visualization

Show: 20 Per Page ▾ Format ▾ Preview: On

JSESSIONID	count
SD8SL7FF6ADFF4	18
SD1SL1FF4ADFF10	16
SD5SL2FF9ADFF7	16

Search job inspector

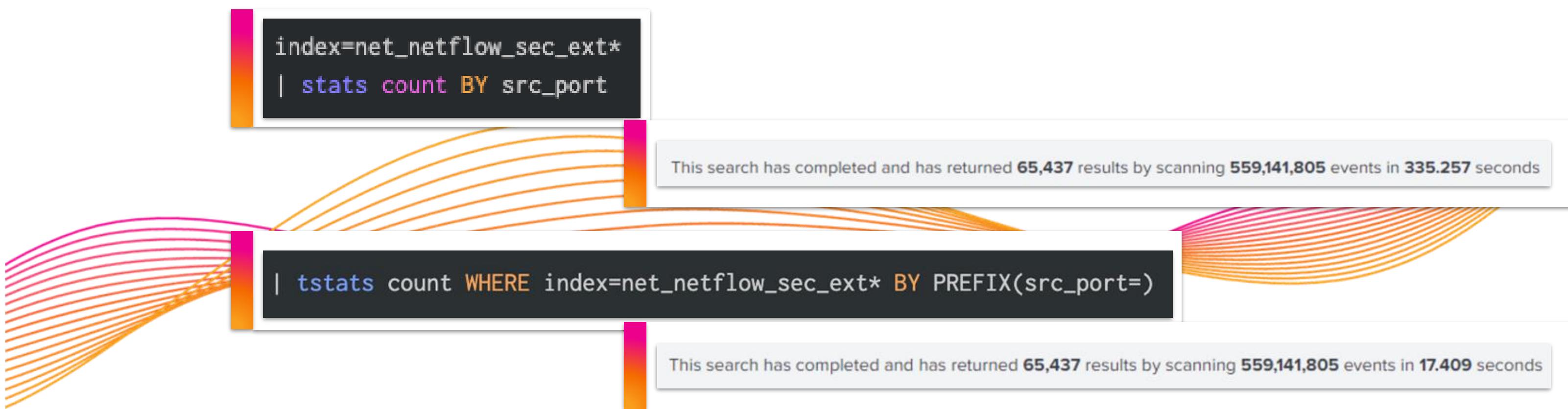
This search has completed and has returned 10 results by scanning 42,281 events in 0.157 seconds

(SID: 1745267659.2157) [search.log](#) [Job Details Dashboard](#)

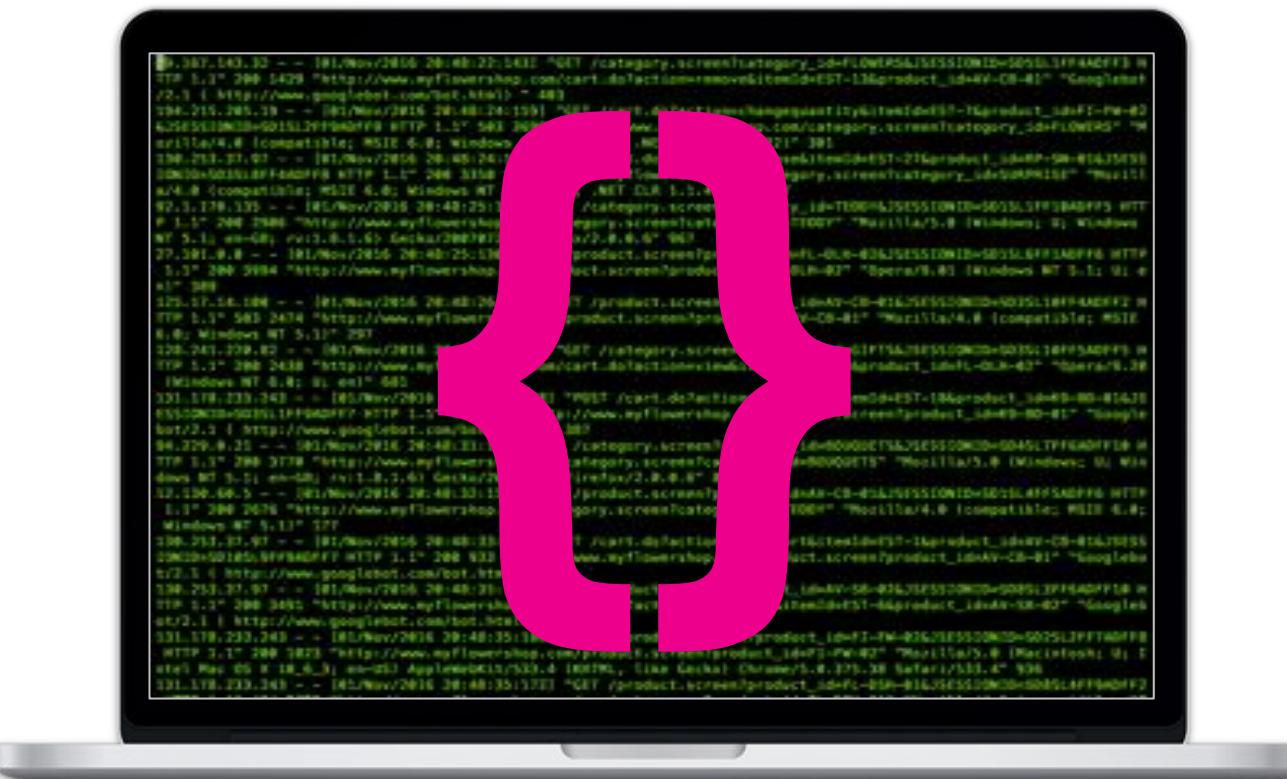
Simple search across Netflix logs

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.



SPL#> eval's Hidden Feature



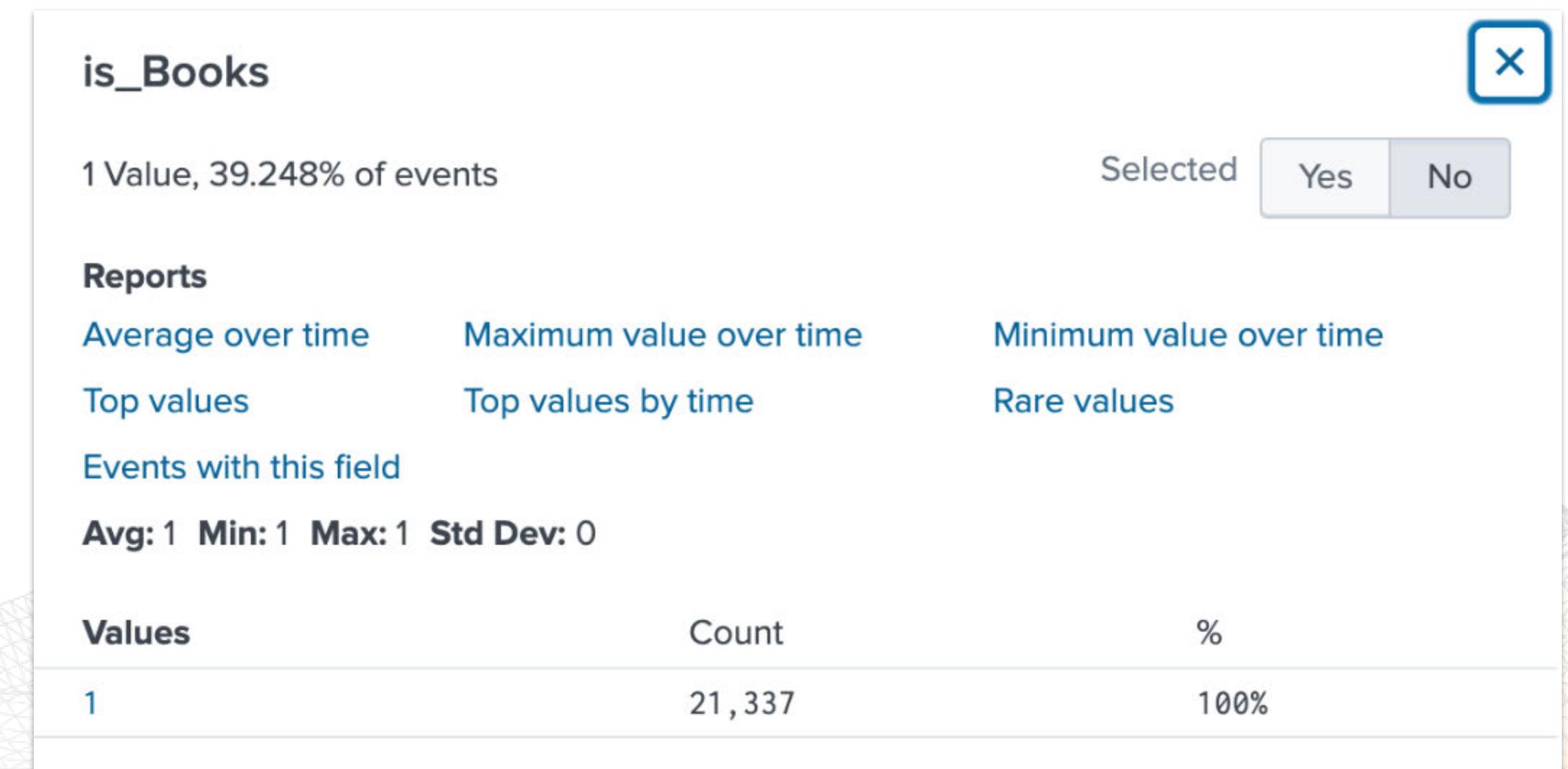
eval and the Field Name Trick



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

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

```
# is_Books 1  
# is_Clothing 1  
# is_Gifts 1
```



Learn more: [SPL Command - "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  

41305.21

Total_Sales_of_Clothing  

103928.31

Total_Sales_of_Gifts  

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# > Macros



```
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

The screenshot shows the Splunk interface with a search bar containing the query: `index=main sourcetype=access_combined action=purchase status=200 | `ConvertUSD``. The search results table has two columns: `product_name` and `total_sales`. A blue box highlights the backtick character (~) in the search bar, and another blue box highlights the backtick character (~) in the table header for the `total_sales` column.

product_name	total_sales
Batguy Slippers	\$422,636.50
Batguy Watch	\$159,360.48
Costume- ManHawk	\$1,590,907.50
Double Fudge Sundae	\$366,570.75
Mad Comics- Batguy	\$206,209.90
Mad Comics- Bronze Man	\$208,775.30
Mad Comics- Flyman	\$205,574.90
Pony Potpourri	\$165,054.78
Waterproof Scratch and Sniff	\$80,997.68
Zombie Survival Guide	\$246,462.84

Learn more: [Define Search Macros in Settings](#)

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 \$422,636.50

Batguy Watch \$159,360.48

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

SPL# >

eventstats and streamstats





Use Case Example

Thinking beyond stats command

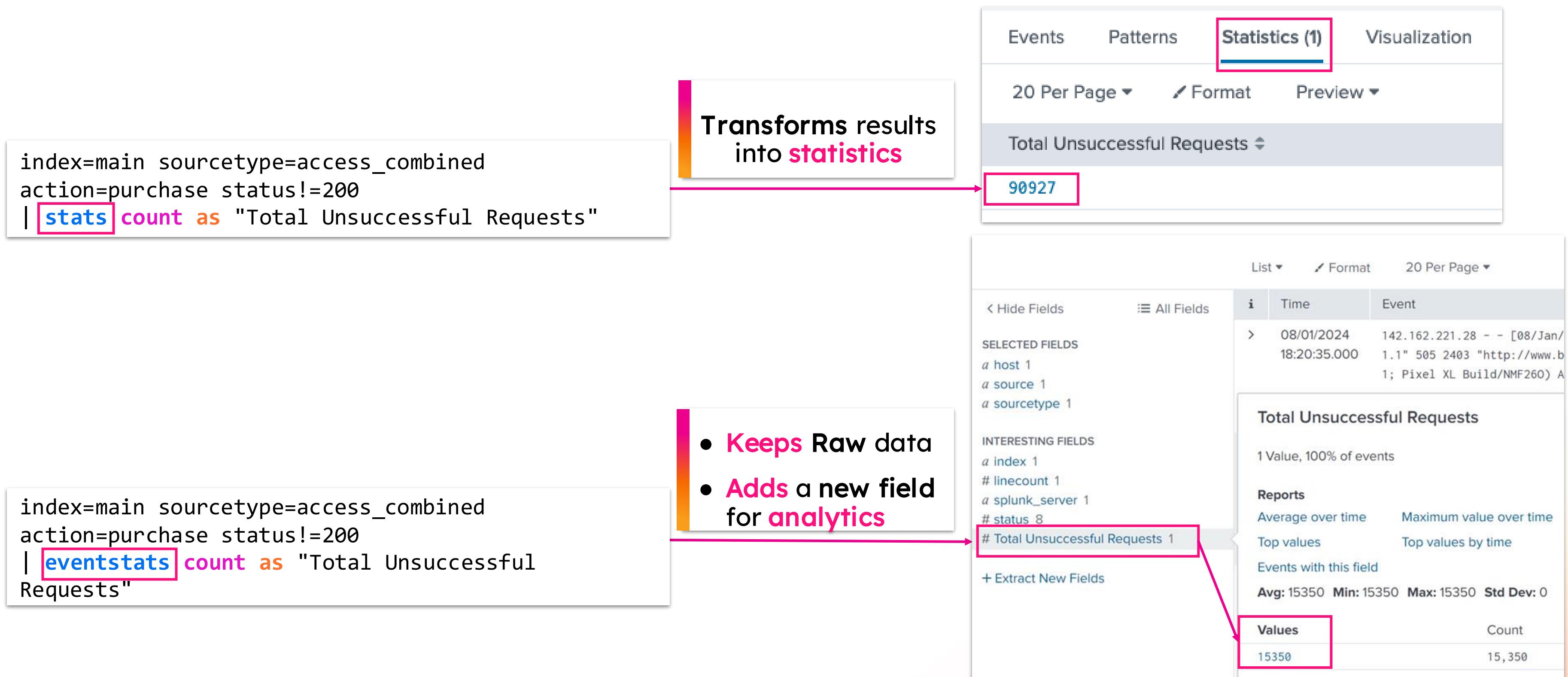
stats may not always be helpful !

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**

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

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> <code>count()</code> <code>distinct_count()</code> <code>estdc()</code> <code>estdc_error()</code>	<code>exactperc<int>()</code> <code>max()</code> <code>median()</code> <code>min()</code> <code>mode()</code>	<code>perc<int>()</code> <code>range()</code> <code>stdev()</code> <code>stdevp()</code>	<code>sum()</code> <code>sumsq()</code> <code>upperperc<int>()</code> <code>var()</code> <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>		

SPL# > 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"
```

The screenshot shows the Splunk search interface. 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 1" field, which is highlighted with a pink border. The main panel displays the results of the search. The title is "Total Unsuccessful Requests". It shows "1 Value, 100% of events". Under "Reports", there are four options: "Average over time", "Maximum value over time", "Minimum value over time", "Top values", "Top values by time", and "Rare values". Below these are links for "Events with this field" and "Avg: 23607 Min: 23607 Max: 23607 Std Dev: 0". A table titled "Values" shows one row: "23607" with a "Count" of "23,607" and a "%" of "100%". On the right, there is a "Selected" button with "Yes" and "No" options, and a blue "X" icon.

New field
created with
total count of
unsuccessful
requests

Learn more: [SPL Command - "eventstats"](#)

SPL# > 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 various product-related fields like product_id, product_name, product_price, and several 'a' fields for referer and splunk_server. A specific field, '# Total Unsuccessful Requests 100+', is highlighted with a red box and an orange arrow pointing from the explanatory text above. To the right, a summary table titled 'Total Unsuccessful Requests' provides statistical details: >100 Values, 100% of events. It includes sections for Reports (Average over time, Maximum value over time, Minimum value over time; Top values, Top values by time, Rare values) and Events with this field. Summary statistics shown are Avg: 11959, Min: 1, Max: 23917, Std Dev: 6904.387530162734. The 'Top 10 Values' section shows two entries: '1' and '10', both 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: [SPL Command - "streamstats"](#)

SPL# >

Percentage of Total Revenue



SPL# > Percentage of Total Revenue

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

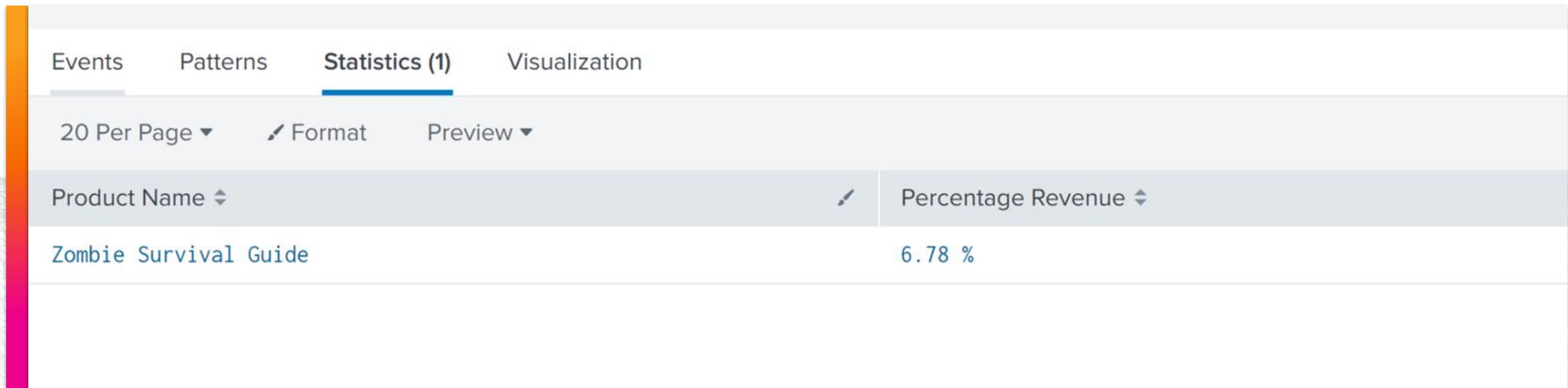
product_name	per_prod_rev	total_rev	rev_pct
Zombie Survival Guide	217731.15	3225141.63	6.7510570

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# > Challenge Task

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



SPL# > 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
| where product_name="Zombie Survival Guide"
| eval percent_revenue=round(rev_per_prod/total_rev*100,2)." %", "ProductName"=product_name
| stats values(percent_revenue) as "Percentage Revenue" by "Product Name"
```

SPL# >

Popularity Ranking



SPL# > 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# > Using **streamstats** and **eventstats**

stats may not be always **sufficient** !



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# > 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

```
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# >

spath as an SPL Command



Extract Specific Information

Default > Splunk Extracts JSON

Directly in SPL

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

spath
command

As **function** within
eval command

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

SPL# > 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: [SPL Command - "spath"](#)

SPL# > 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{}
```

Explore the newly created field “**new_json_field**”

```
# 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
```

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# >

spath as an **eval**

Function



Extract Specific Information

Default > Splunk Extracts JSON

Directly in SPL

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

spath
command

As function within
eval command

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

SPL# > spath as an eval function

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

Syntax:

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

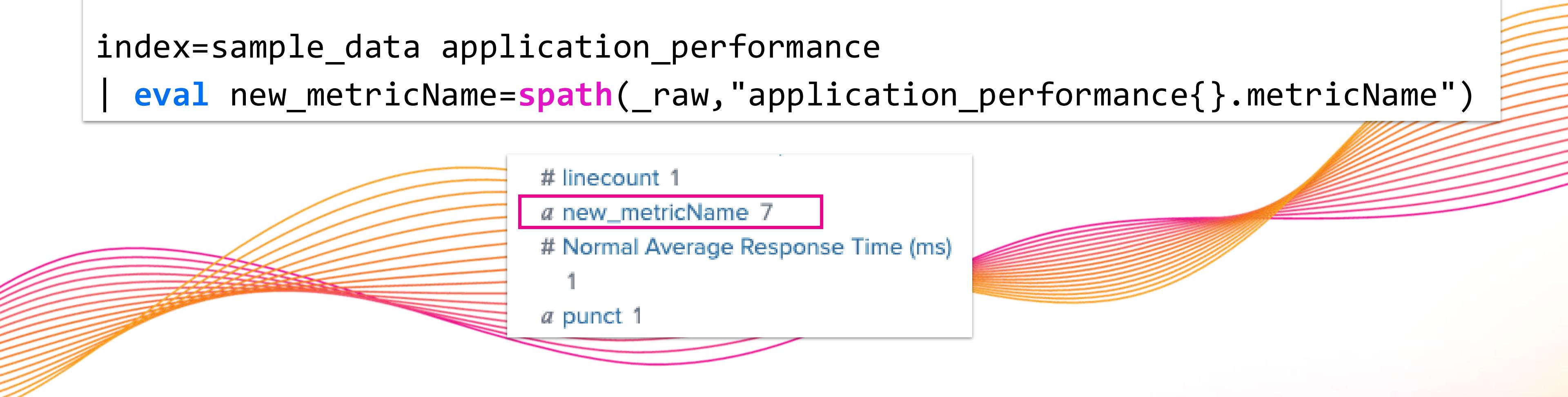
From structured data type(**X**) in XML or JSON format; extract value at **Y** and assign to **A**

SPL# > 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")
```



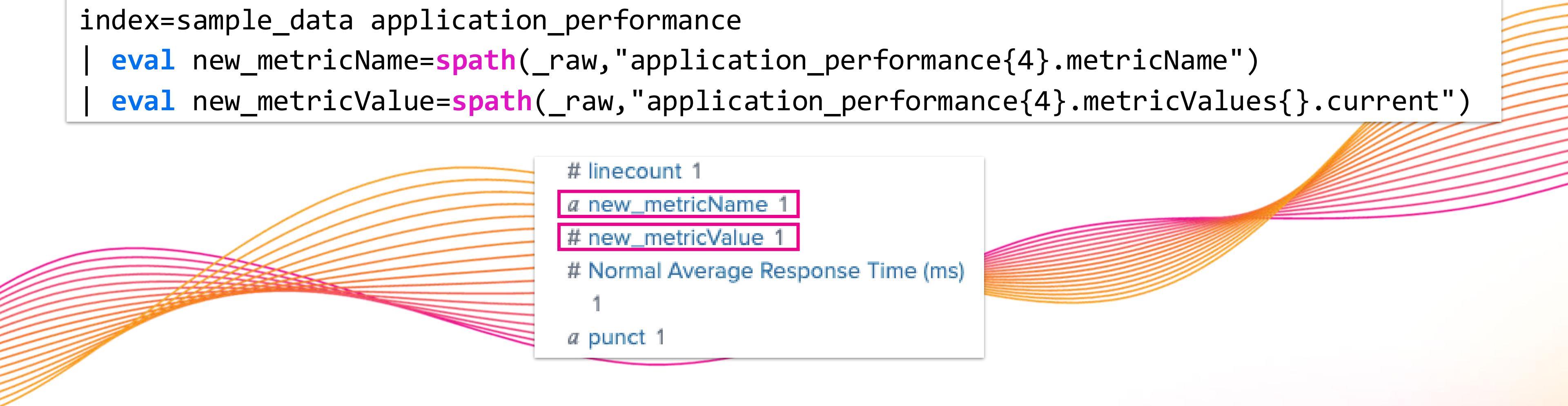
```
# linecount 1  
a new_metricName 7  
# Normal Average Response Time (ms)  
1  
a punct 1
```

SPL# > 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: [SPL Command - "spath"](#)

Validate if a new field was created



Validate if a new field was created

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

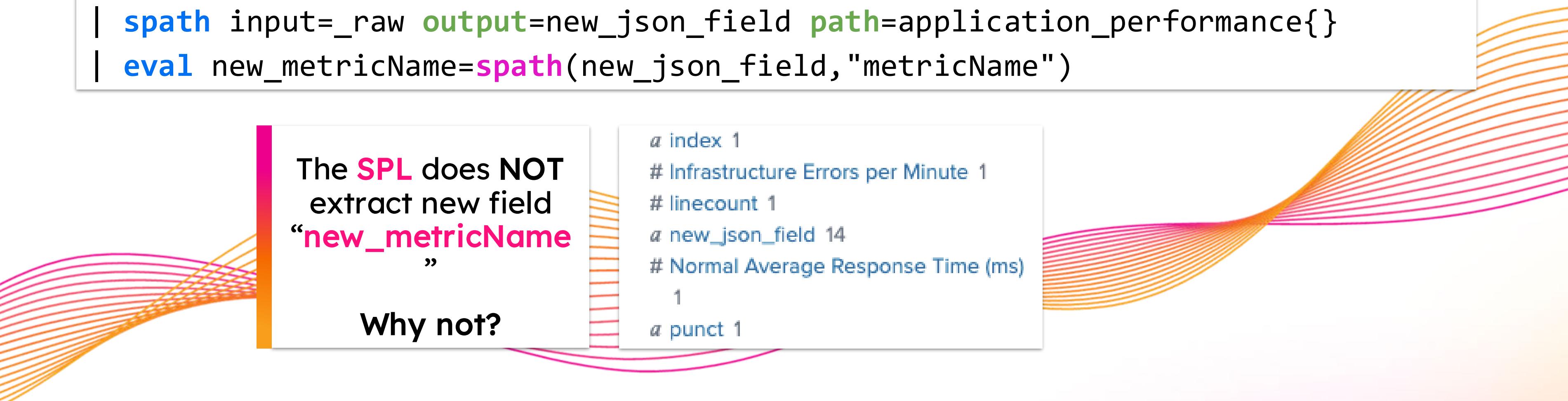
Solution:

```
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

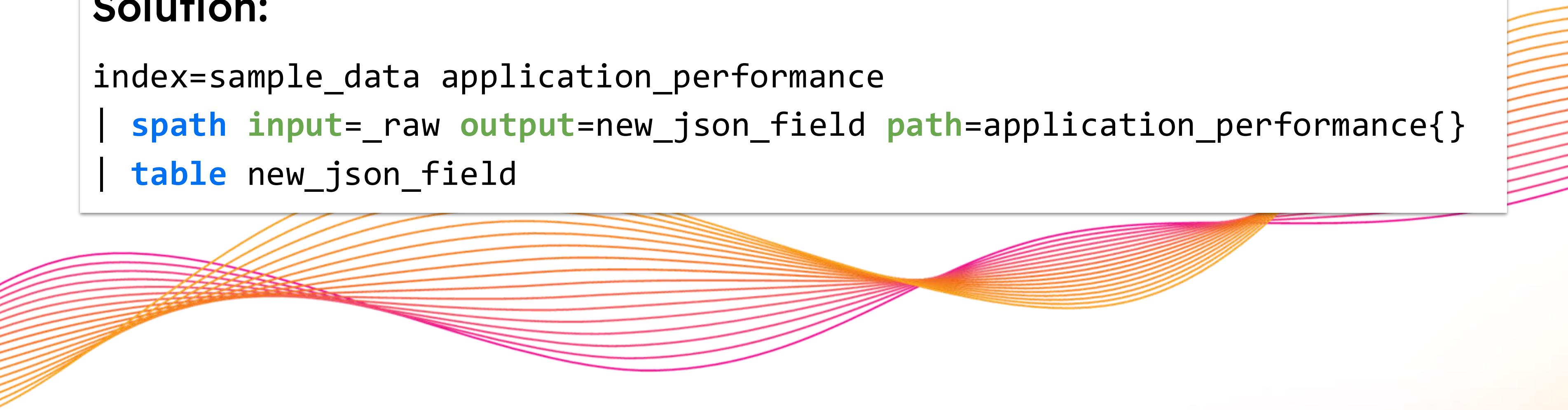


Find out why the new field was not created

spath command creates a multi-value field:

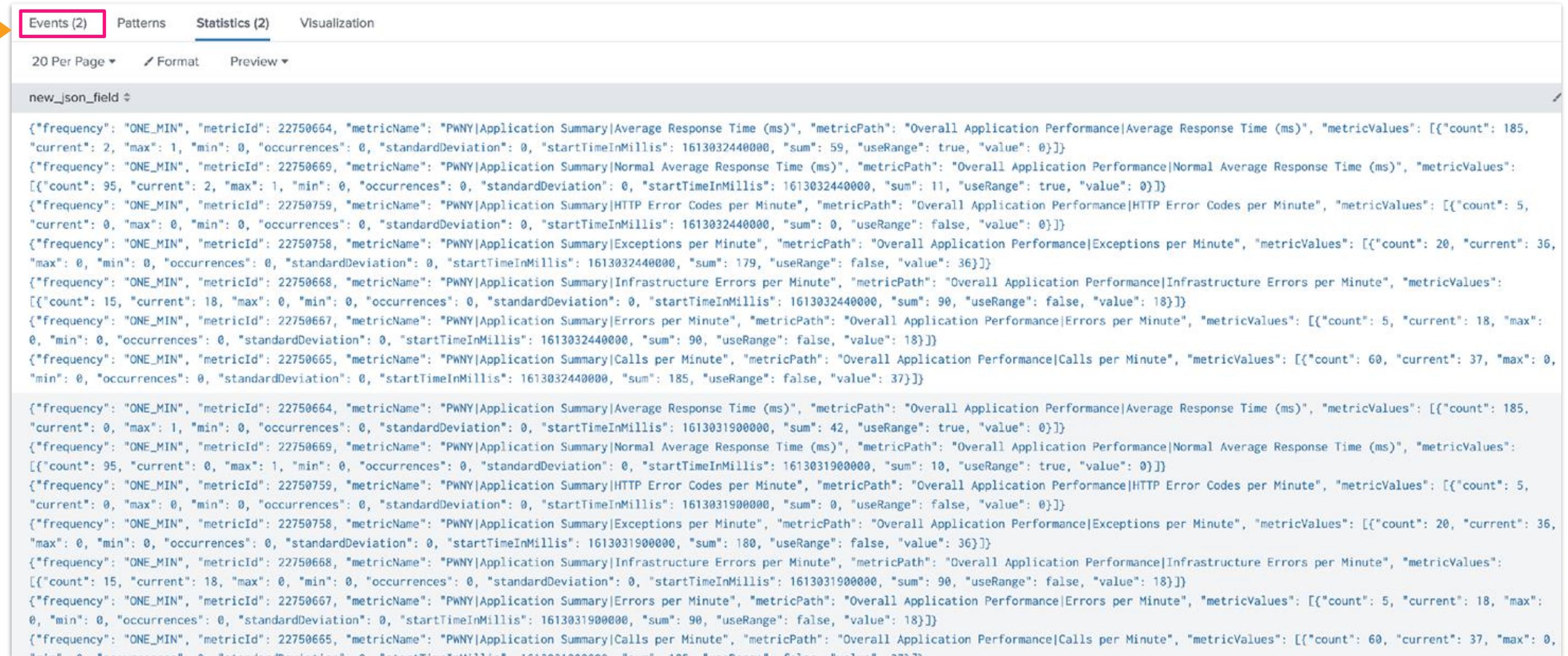
Solution:

```
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 interface with the following details:

- Events (2)** is selected in the top navigation bar.
- The search results show two events under the heading "new_json_field".
- The first event contains 18 JSON objects, each representing a different metric path and its statistics.
- The second event also contains 18 JSON objects, representing the same set of metrics.
- An orange arrow points from the text "Two events returned with multiple values" to the "Events (2)" button in the top bar.

```
{"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": 22750657, "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}], {"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}], {"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": 0, "max": 1, "min": 0, "occurrences": 0, "standardDeviation": 0, "startTimeInMillis": 1613031900000, "sum": 10, "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": 1613031900000, "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": 1613031900000, "sum": 180, "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": 1613031900000, "sum": 90, "useRange": false, "value": 18}], {"frequency": "ONE_MIN", "metricId": 22750657, "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": 1613031900000, "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": 1613031900000, "sum": 185, "useRange": false, "value": 37}]}
```

Learn more: [SPL Command - "spath"](#)

SPL# >

The **mvexpand** Command



SPL# > 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# >

Introducing the **rex** Command

- **Improperly formatted data**
- **multi format events**
- **field extractions**
- **data transformation**



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
- The previous example 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 <b>PWNY-Apps-Weblogic-server317-servicesAdminServer</b>. <br><b>JVM Heap utilization is too high</b> started violating and is now <b>warning</b>. <br>All of the following conditions were found to be violating<br>For Node <b>PWNY-Apps-Weblogic-server317-servicesAdminServer</b>:<br>1) JVM|Memory:Heap|Used % Condition<br><b>Used %'s</b> value <b>76.00</b> was <b>greater than</b> the threshold <b>75.00</b> for the last <b>30</b> minutes.<br>", "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 <b>PWNY-Apps-Weblogic-server318-webAdminServer</b>. <br><b>JVM Heap utilization is too high</b> started violating and is now <b>warning</b>. <br>All of the following conditions were found to be violating<br>For Node <b>PWNY-Apps-Weblogic-server318-webAdminServer</b>:<br>1) JVM|Memory:Heap|Used % Condition<br><b>Used %'s</b> value <b>76.00</b> was <b>greater than</b> the threshold <b>75.00</b> for the last <b>30</b> minutes.<br>", "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](#)

```
^(?:[^\\n]*)(2)\\w+</\\w+>\\s+\\w+\\s+<\\w+>(?P<mem_used_value>[^<]+)
```

[View in Search](#)

[Edit the Regular Expression](#)

SPL# > 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# > 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 command is displayed:

```
# 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
```

On the right, the search results are shown for an event on 11/02/2021 at 07:45:55.001. The event details are as follows:

```
> 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 'a delivery 1' in the search command to the 'delivery' field in the event details. Another red arrow points from the highlighted 'appd_app_uuid' in the search command to the 'appd_app_uuid' field in the event details.

SPL# > Introducing the rex command

Working with multiformat events

Use Case: From health rule violation events (json), extract “Memory Heap used % value” (xml)

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

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

SPL# > 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>(.\+?))\<**

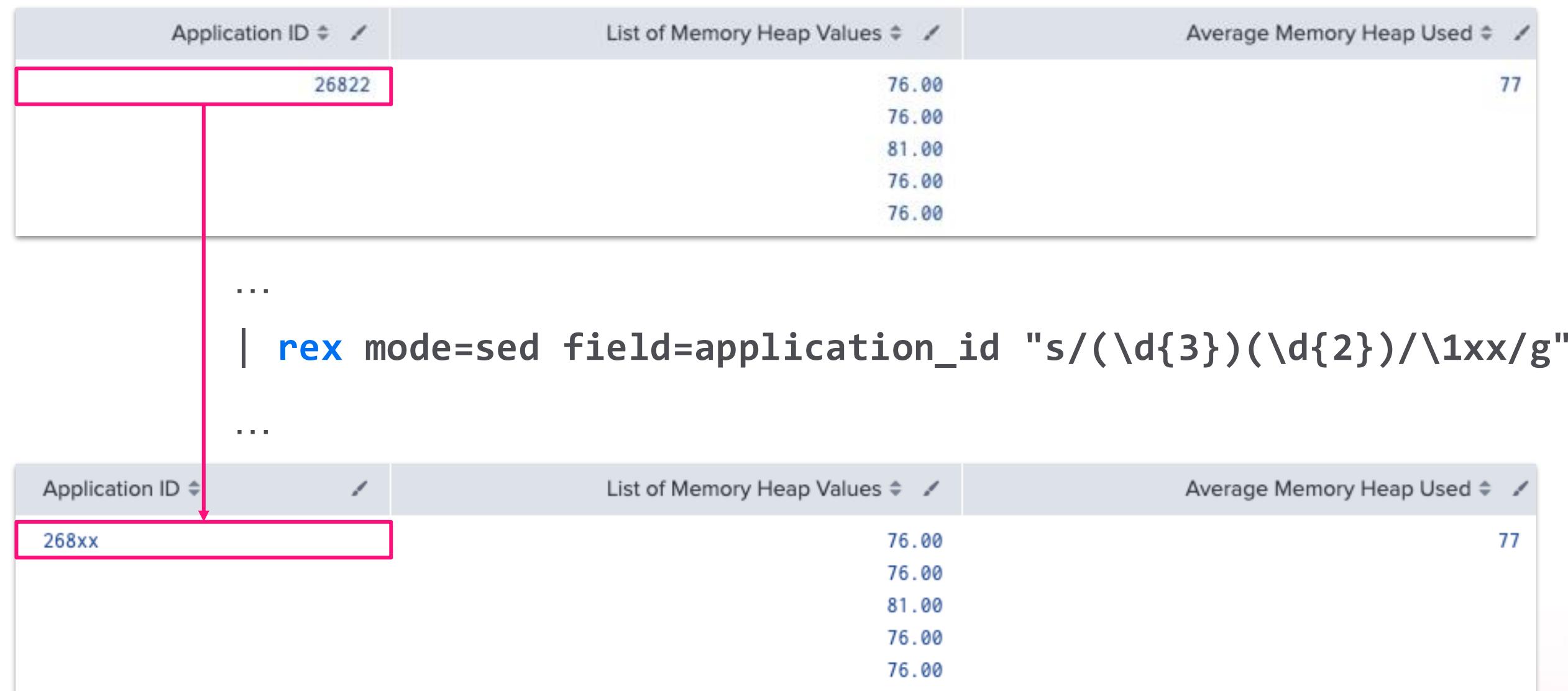
SPL# >

**Using rex
command with
SED Mode**



SPL# > Using the 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.



Good Rex / Bad Rex

A.k.a “Who’s been a good boy?”



Good Rex / Bad Rex

The scene

Considering our healthrule violations data

```
index=sample_data healthruleViolations earliest=0
```

We want to get the Controller FQDN value:

```
{ [-]  
  account_name: buttercup-dev  
  appd_app_uuid: PWNY:26822:buttercup-dev.saas.appdynamics.com  
  application_id: 26822  
  application_name: PWNY  
  healthruleViolations: [ [+]  
    ]  
}
```

Good Rex / Bad Rex - The Bad Way

appd_app_uuid: PWNY:26822:buttercup-dev.saas.appdynamics.com

The bad Regex

```
| rex "\w+.*:(?<FQDN>([^\n]*))?"
```

Why is it bad?

- Lazy wildcard (.*?) scan the entire text looking for matches, increasing backtracking.
- Non-matching characters [^\n] look for characters that do not match \n
- Greedy quantifier (*) matches 0 or more occurrences of a character
- No specification (field=<value>) will run on the entire raw data.

As a result, our field extraction is not only performing poorly, but it's also not extracting what we need.

```
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 <b>PWNY-Apps-Weblogic-server317-servicesAdminServer</b> <br><h>TVM Heap utilization is too high</h> started violating and is now <h>warning</h> <br>All of the following conditions were found to be violating<br>For Node <h>PWNY-Apps-Weblogic-server317-
```

Good Rex / Bad Rex - The Good Way

appd_app_uuid: PWNY:26822:buttercup-dev.saas.appdynamics.com

The good Regex

```
| rex field=appd_app_uuid "\w{4}:\d{5}:(?<FQDN>[\w\-\.\.]+)"
```

Why is it good?

- Specific matching of pattern prefix \w{4}:\d{5}: Splunk knows as precisely as possible what to look for before the extraction
- Inclusion rather than exclusion (\w) in most cases, inclusion will be slightly better than exclusion
- No greedy/ undefined quantifiers ({4}) tell Splunk exactly how many characters to expect
- Specification (field=appd_app_uuid) avoids scanning the entire _raw data

As a result, our field extraction is now optimized and extracting exactly what we need

buttercup-dev.saas.appdynamics.com

Good Rex / Bad Rex - additional considerations

- There is no “one size fits all”. Always make sure that the expression matches your needs.
 - Greedy quantifiers, source field specifications, etc can’t always be avoided/ specified
- Be as specific/ explicit as you can
- Don’t need the field? Don’t extract it
- Always need the field? Extract it at search time
- Only looking for 1 match? Use the option `max_match=1`
- The rex command has several options. Make sure to read through the documentation to find one that helps you

Learn more: [SPL Command - "rex"](#)

Before we close this session...

There's something we should really talk about...



SPL2 when, what, where?

When do I need to use SPL or SPL2?

- If you are working with data management, for example building Edge Processor pipelines, SPL2 is the language to use. However, for your search queries, alerts, reports, dashboards and more, SPL is still the preferred language.

What is SPL2?

- Product-agnostic, intuitive language that includes the best of both query and scripting languages.
- Supports both SPL and SQL syntax patterns.
- Designed to work with the variety of runtimes in the Splunk portfolio.
- Fully backwards compatible with SPL, and can operate in parallel with SPL.

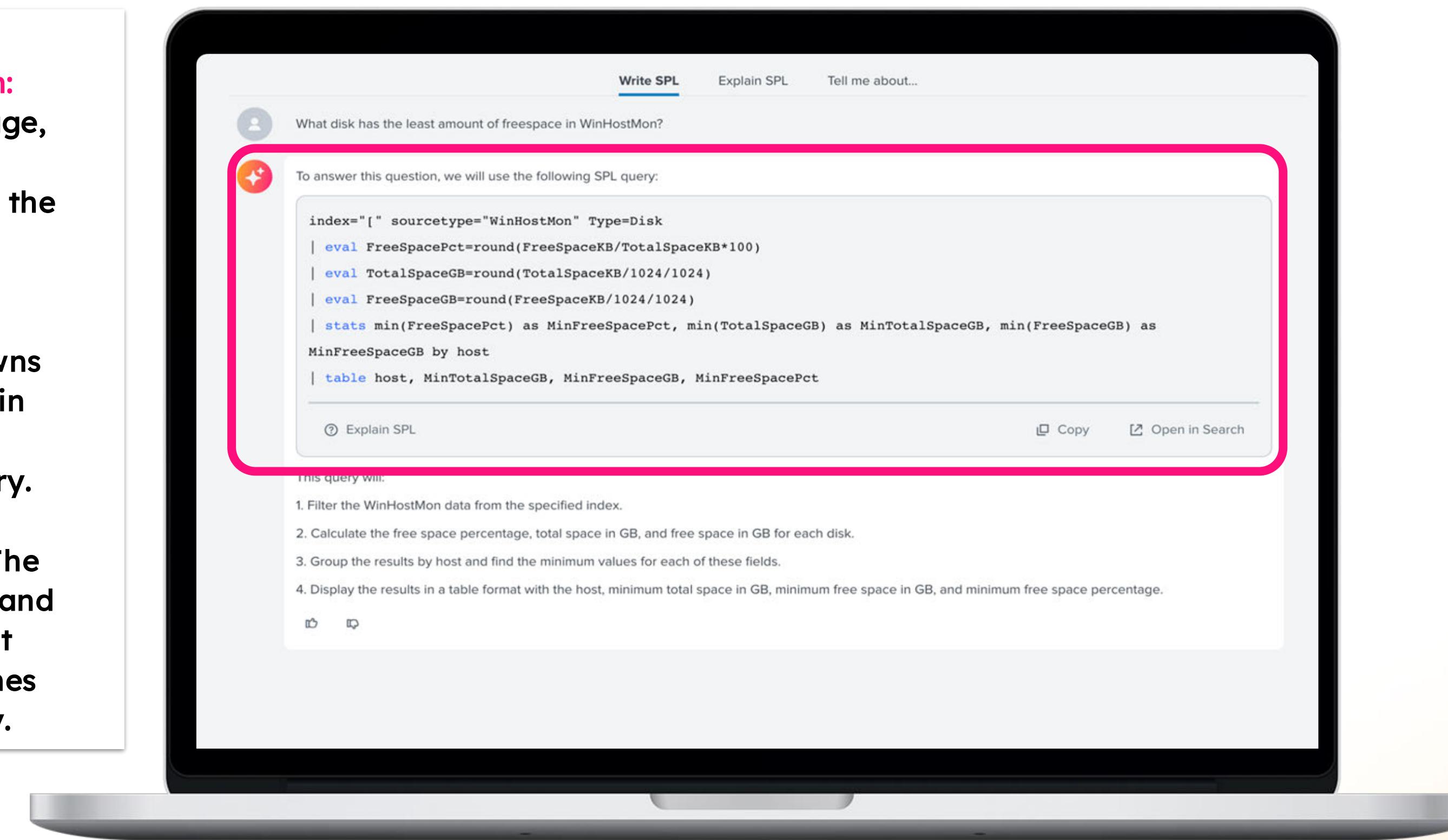
Where is SPL2 used?

- Currently, SPL2 is aimed at **system administration**. It can be used in both splunkd and non splunkd environments. Examples of non-splunkd environments are O11y metrics, Ingest Processor and Edge Processor

Differences Between SPL and SPL2

AI Assistant for SPL - Is it useful? YES !

- ▶ **Natural Language to SPL Translation:** Users can input tasks in plain language, and the assistant converts them into executable SPL searches, simplifying the query creation process.
- ▶ **SPL Explanation and Learning:** The assistant provides detailed breakdowns of existing SPL queries, aiding users in understanding and learning SPL by explaining each component of a query.
- ▶ **Adaptive Query Recommendations:** The AI assistant suggests improvements and alternative SPL queries based on best practices, helping users refine searches for better performance and accuracy.



[Splunk AI Assistant for SPL](#)

Since SAIA 1.4, an **SPL optimization** feature is available !

Splunk AI Assistant for SPL features

You can complete the following tasks with Splunk AI Assistant for SPL:

- Write a task in plain, natural language and convert this task into a usable SPL search.
- Copy and paste an SPL search and convert this search into a detailed breakdown in plain language.
- Optimize a new or existing SPL search by applying the best Splunk practices in the context of your environment.
- Input a plain, natural language question, or a Splunk platform term or product name and have the app provide information on Splunk product concepts and product functionality, sourced from Splunk documentation, community forums, and other Splunk training materials and resources.
- Iterate on task responses with a conversational user experience.
- Accelerate these tasks within a familiar Splunk interface.

Optimization in Splunk AI Assistant for SPL

So what can it do?

3 "optimization strategies" are available (as of SAIA 1.4):

- Choosing the right index
- Using tstats
- Eliminating unnecessary wildcards

While the use cases are still specific, this is useful already.

Ex: Converting an existing search to **tstats** for example, can be a real time saver.

Between SAIA and the built-in search optimizations, writing production grade SPL searches has never been easier.

What it cannot do just yet is to construct complex multi-step searches (or can it?)

Optimization examples

Some examples are better than a long story

Use case	Input	GPT Output	Llama Output
Select correct index	sourcetype=access_combined status=404 stats count by clientip	index=web_logs sourcetype=access_combined status=404 stats count by clientip	index=web sourcetype=access_combined status=404 stats count by clientip
	search index=* error stats count by host	index IN (_audit, _introspection, _cmc_summary, _telemetry, summary) message="error" stats count by host	search index=main sourcetype=error stats count by host
	index=* fields _time userIdentity.arn eventName	index="aws" sourcetype="aws:cloudtrail" fields _time userIdentity.arn eventName	search index=_internal sourcetype=stash earliest=-1h@h latest=-0h@h fields _time userIdentity.arn eventName
	index=_internal sourcetype== stats count by host, sourcetype	tstats count FROM index=_internal WHERE sourcetype IN ("splunkd", "splunk_web_access") BY host, sourcetype	tstats count WHERE index=_internal BY host, sourcetype

Optimization examples

Splunk Resources

Where to go after
today's workshop



Our world-class community is here to help.



Splunk community

Splunk's user community is 230,000 members strong and they have answered 125K+ product questions with 150+ user groups worldwide.



Partner network

Splunk partners help deliver complete solutions and accelerate time to value for customers.



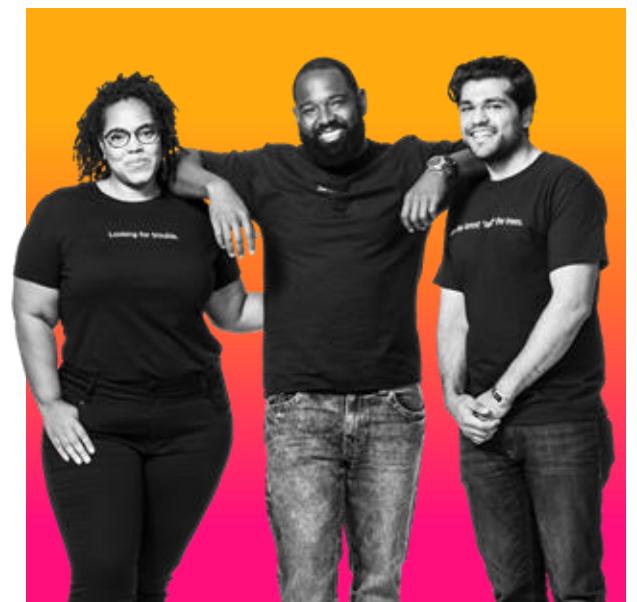
Customer success and education

Our support experts can help optimize your environment and our education resources can help equip teams for success.



Professional services

Splunk experts specialize in finding out what is important to you and can help make your top priorities a reality.



SURGe

Our security experts are dedicated to providing timely recommendations on the latest global breaking security news and how to protect your organization from evolving threats.

Splunk's Thriving Community

Splunk Community

The screenshot shows the Splunk Community homepage. At the top, it displays "Learn, Give Back, Have Fun". Below this are statistics: "1,382 Online Now", "123K Discussions", and "51.7K Solutions". A search bar and a "Sign In" button are at the top right. The main content area features three icons: a speech bubble for asking questions, a group icon for joining groups, and a lightbulb for submitting ideas. Below these are links to "Ask a Question", "Join a Group", and "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 subtext: "Join us at an event near you to gain new skills, expand your network and connect with the Splunk community." Below this is a section titled "Upcoming Events" with a single event listed: "Automation for the Modern SOC: Strategies for Smarter Security Operations" on November 9, 2021, at 10:00 AM (GMT). There are filters for "All Event Types", "All Geographies", and "All Solution Areas".

Documentation

The screenshot shows the Splunk Documentation homepage. It has a dark blue header with the text "splunk> docs". Below the header are sections for "Platform", "Security", "IT", "Observability", "Apps and add-ons", and "Developer tools". Each section lists various Splunk products or services, such as Splunk Cloud Platform, Splunk Enterprise, and Splunk SOAR.

Developer Resources

The screenshot shows the Splunk Dev resources page. It features a "Welcome to splunk>dev" section with a "Build apps that Turn Data Into Doing™ with Splunk" message. Below this are two main sections: "Develop for Splunk Cloud and Splunk Enterprise" and "Develop for Observability". The "Develop for Splunk Cloud and Splunk Enterprise" section includes a link to "Test in your free development Splunk platform instance, and deliver in the Splunkbase marketplace".

Splunkbase

The screenshot shows the Splunkbase homepage. It features a "Get more out of Splunk with applications" section with a 3D grid of app icons. Below this is a "Trending Apps on Splunkbase" section with several app cards. At the bottom, there is a "Collections" tab and a "Submit an App" button.

Education

The screenshot shows the Splunk Course Catalog page. It features a "Course Catalog" section with a "Start Your Journey" button. Below this are sections for "What is Splunk?", "Intro to Splunk", and "Using Fields". There is also a "Filter Courses" sidebar with options for "Content Type" and "Certification".

Splunk Lantern

The screenshot shows the Splunk Lantern Resource Hub page. It features a "Splunk Lantern Resource Hub" section with a "Lighting your way with clear and actionable guidance from Splunk experts" message. Below this are sections for "Security Use Case Guidance" and "IT Use Case Guidance", each listing various articles and topics.

Splunk Education

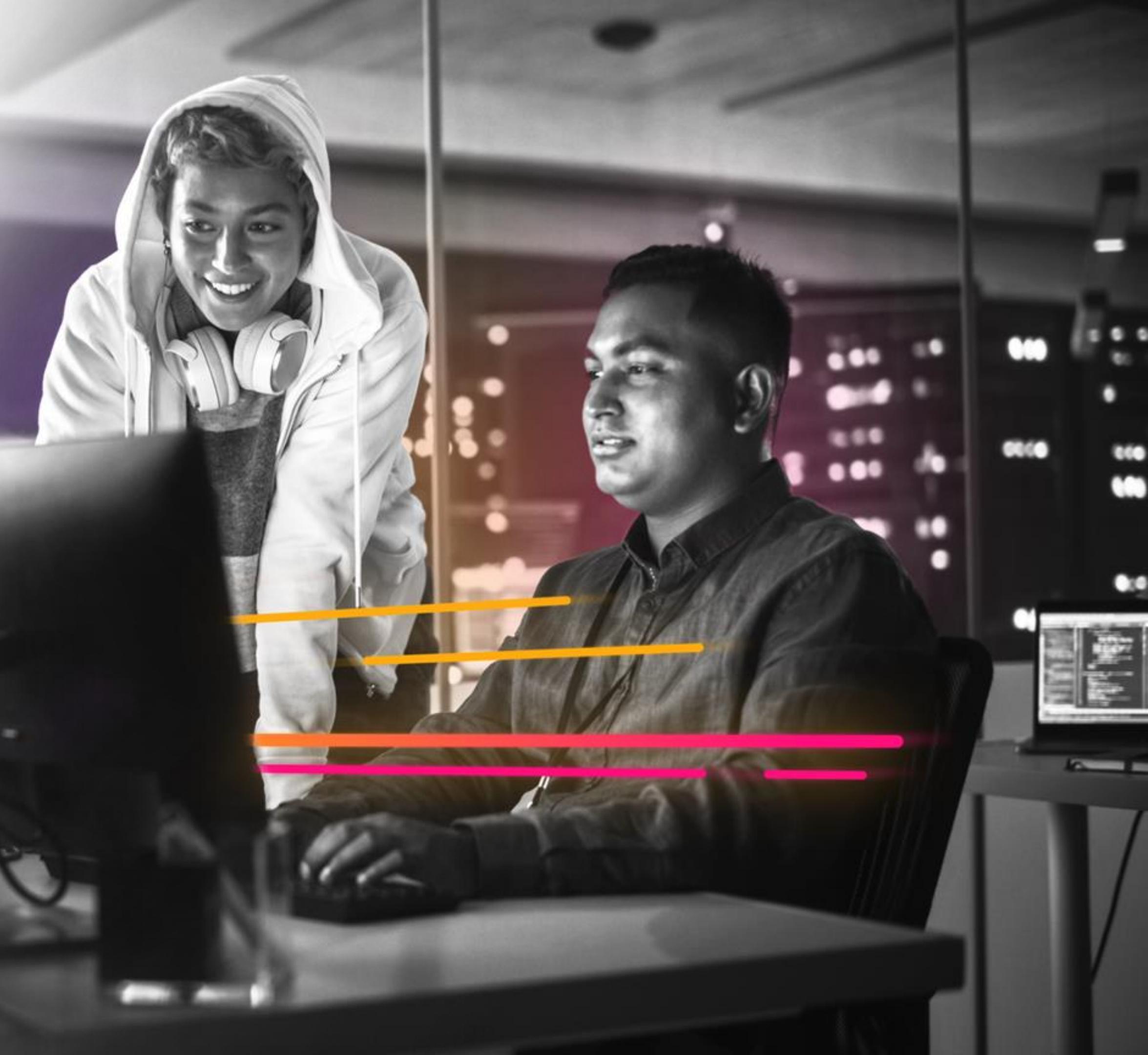
Courses relevant to this workshop

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



Deep Dive Self-Study

- Tips & Tricks for Optimization:
[Quick tips for optimization - Splunk Documentation](#)
- Fields, Indexed Tokens and You (.conf presentation):
 - [Fields, Indexed Tokens and You](#)
 - [Video](#)
- TSTATS and PREFIX (.conf presentation):
[TSTATS and PREFIX](#)
- JSON Functions:
[JSON functions - Splunk Documentation](#)
- Joining datasets/emulating SQL behavior:
 - [Splunk SPL for SQL users](#)
 - [Master Joining Datasets Without Using Join](#)
 - [Video](#)



Thank you!