Forward-Looking Statements

Fields, Indexed Tokens, and You

PLA1466B

Martin Müller

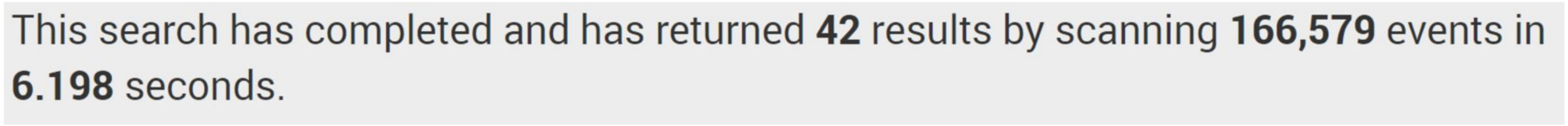
Principal Consultant | Consist

Martin Müller

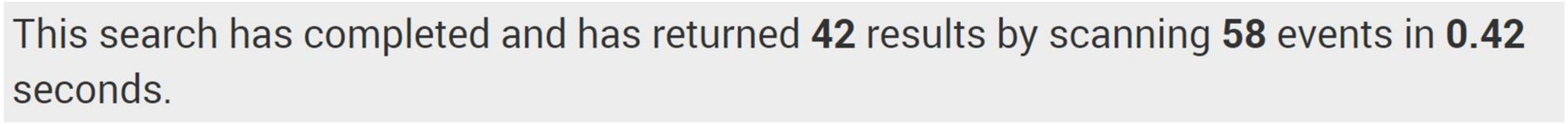
Principal Consultant
Consist Software Solutions GmbH

Why Are We Here?

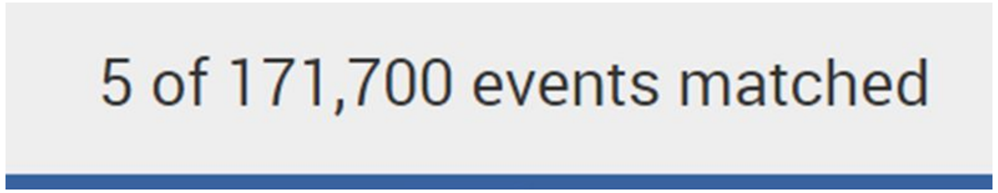
- Supercharged searches!
- I want you to turn this…


This search has completed and has returned **42** results by scanning **166,579** events in **6.198** seconds.

…into this!

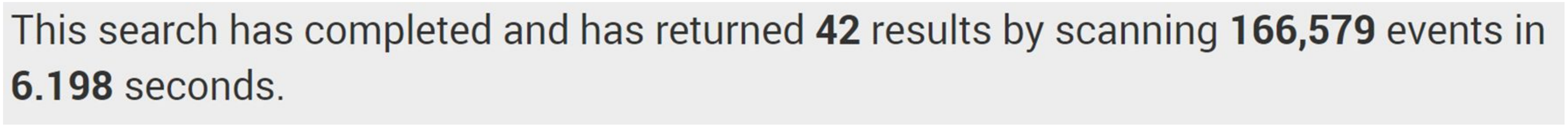

This search has completed and has returned **42** results by scanning **58** events in **0.42** seconds.

…this is bad:


5 of 171,700 events matched

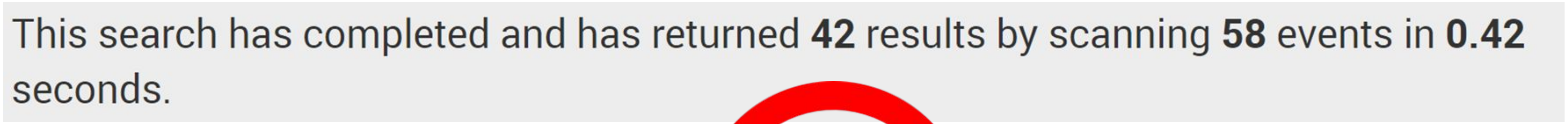
Why Are We Here?

- Supercharged searches!
- I want you to turn this…



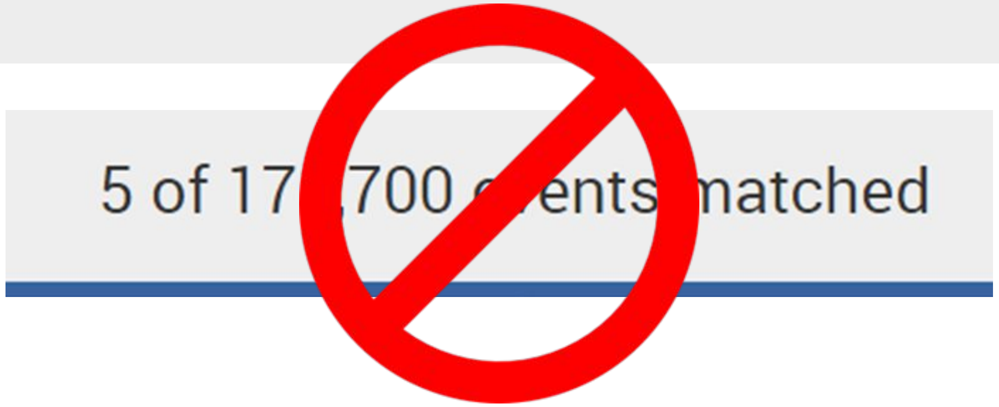
This search has completed and has returned **42** results by scanning **166,579** events in **6.198** seconds.

…into this!



This search has completed and has returned **42** results by scanning **58** events in **0.42** seconds.

…this is bad:



5 of 17[illegible],700 [illegible]ents [illegible]atched

Session Objectives

- Understand how Splunk® turns a log file into indexed tokens
  - Breakers & segmentation
- Learn how your searches make good use of indexed tokens (or not)
  - Lispy
  - Fields

Breakers & Segmentation

How does Splunk® break events into indexed tokens?

splunk> .conf22

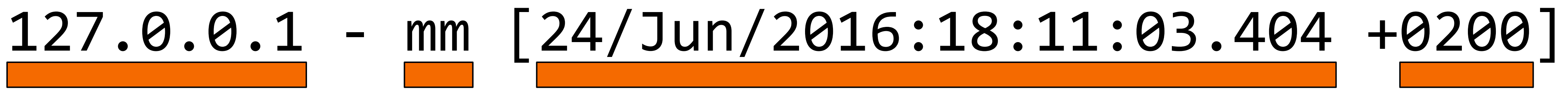
How Splunk® Chops Up an Event

- Read in a line of data, apply segmentation, store tokens in TSIDX files
- Minor breakers: `/ : = @ . - $ # % \ _`
- Major breakers: `\r\n\s\t [] <> () {} | ! ; , ‘ "` etc.
- Can be configured in segmenters.conf – but very rarely should!

```
127.0.0.1 - mm [24/Jun/2016:18:11:03.404 +0200]
```

How Splunk® Chops Up an Event

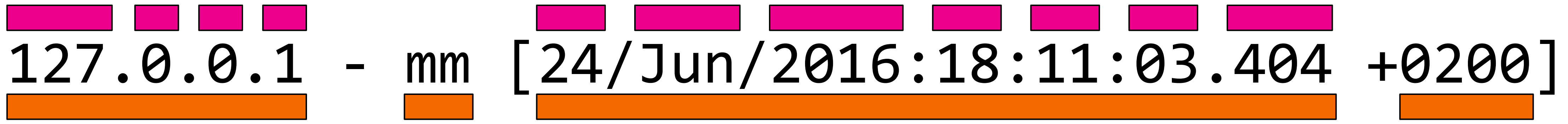
- Read in a line of data, apply segmentation, store tokens in TSIDX files
- Minor breakers: `/ : = @ . - $ # % \ _`
- Major breakers: `\r\n\s\t [] <> () {} | ! ; , ‘ "` etc.
- Can be configured in segmenters.conf – but very rarely should!



```
127.0.0.1 - mm [24/Jun/2016:18:11:03.404 +0200]
```

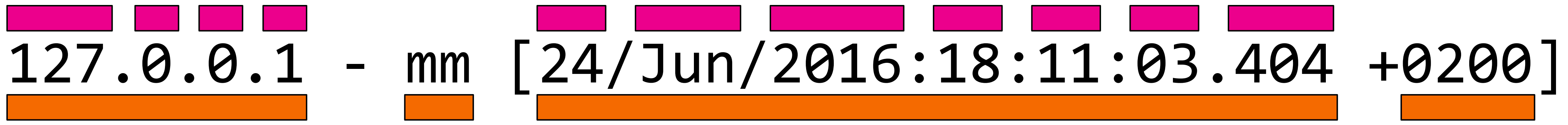
How Splunk® Chops Up an Event

- Read in a line of data, apply segmentation, store tokens in TSIDX files
- Minor breakers: `/ : = @ . - $ # % \ _`
- Major breakers: `\r\n\s\t [] <> () {} | ! ; , ‘ "` etc.
- Can be configured in segmenters.conf – but very rarely should!



```
127.0.0.1 - mm [24/Jun/2016:18:11:03.404 +0200]
```

Inspect a TSIDX File (CLI)



```
127.0.0.1 - mm [24/Jun/2016:18:11:03.404 +0200]
```

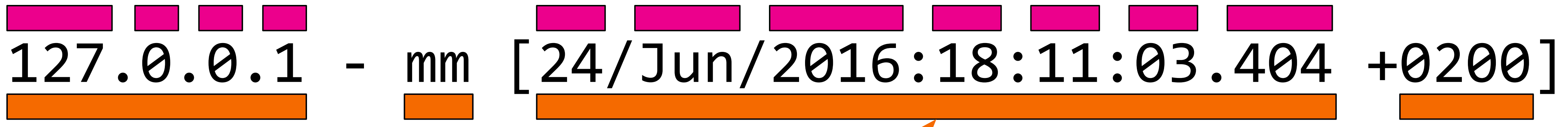
```
splunk cmd walklex ..\var\lib\splunk\<index>\db\<bucket>\<filename>.tsidx ""
```

my needle:

```
2 1 host::localhost
3 1 -
4 1 0
5 1 0200
6 1 03
7 1 1
8 1 11
9 1 127
10 1 127.0.0.1
11 1 18
12 1 2016
13 1 24
14 1 24/jun/2016:18:11:03.404
15 1 404
27 1 jun
29 1 mm
```

Each token is a pointer to the raw event

Inspect a TSIDX File (CLI)



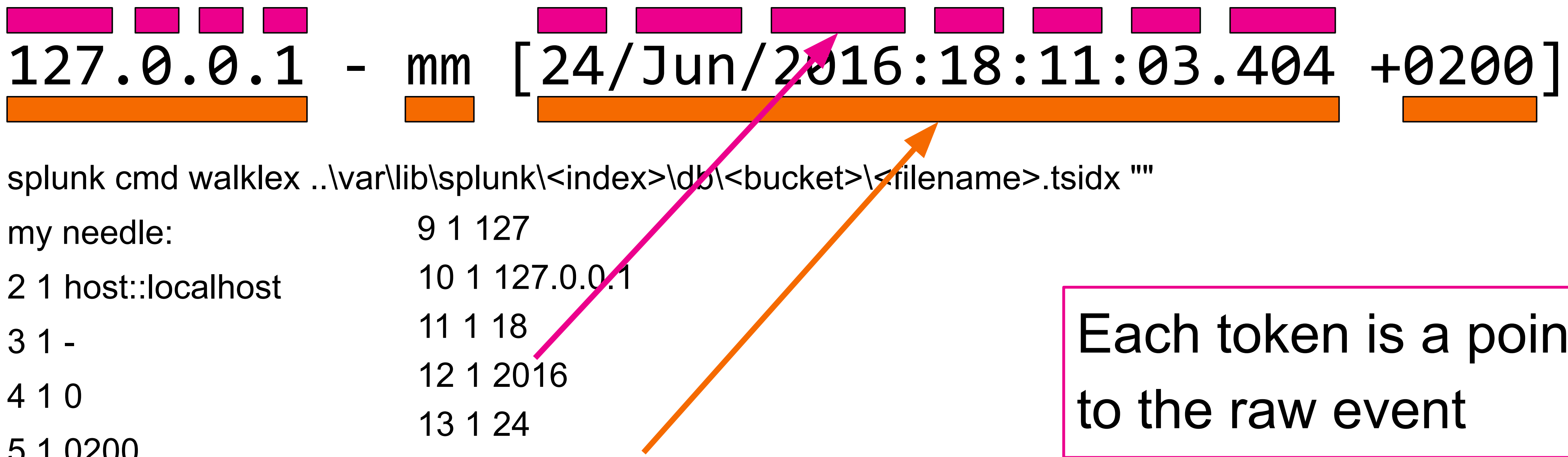
```
127.0.0.1 - mm [24/Jun/2016:18:11:03.404 +0200]
```

```
splunk cmd walklex ..\var\lib\splunk\<index>\db\<bucket>\<filename>.tsidx ""
```

```
my needle:
2 1 host::localhost
3 1 -
4 1 0
5 1 0200
6 1 03
7 1 1
8 1 11
9 1 127
10 1 127.0.0.1
11 1 18
12 1 2016
13 1 24
14 1 24/jun/2016:18:11:03.404
15 1 404
27 1 jun
29 1 mm
```

Each token is a pointer to the raw event

Inspect a TSIDX File (CLI)



```
127.0.0.1 - mm [24/Jun/2016:18:11:03.404 +0200]
```

```
splunk cmd walklex ..\var\lib\splunk\<index>\db\<bucket>\<filename>.tsidx ""
```

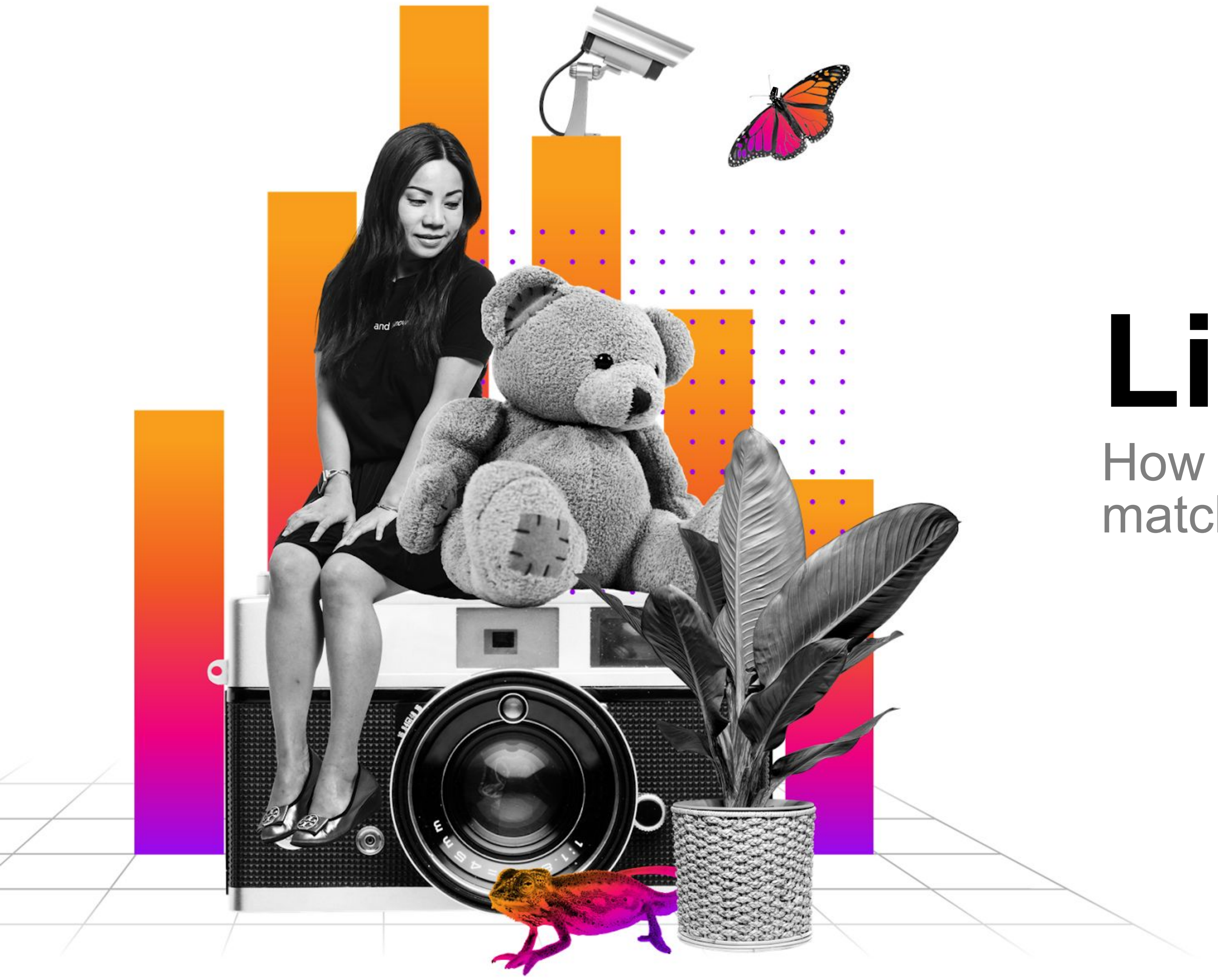
```
my needle:
2 1 host::localhost
3 1 -
4 1 0
5 1 0200
6 1 03
7 1 1
8 1 11
9 1 127
10 1 127.0.0.1
11 1 18
12 1 2016
13 1 24
14 1 24/jun/2016:18:11:03.404
15 1 404
27 1 jun
29 1 mm
```

Each token is a pointer to the raw event

Inspect a TSIDX File (SPL™)

- New search command in 7.3: `| walklex`
- Caveats apply around hot buckets, full list in docs: https://docs.splunk.com/Documentation/Splunk/8.2.5/SearchReference/walklex
- Works in SplunkCloud® too, no need for CLI access

- List indexed fields: `| walklex index=_internal type=field`
- List indexed values: `| walklex index=_internal type=fieldvalue`
- Search for tokens: `| walklex index=_internal type=term prefix=foo`



Lispy

How does Splunk® find events matching your search?

Lispy??

- Lispy expressions are predicates Splunk® platform uses to locate events
- Awesome for debugging and performance tuning

- Square brackets, prefix notation for operators? That's lispy.
- Search for `splunk.conf 2022 – Las Vegas, NV` and you get this:
  `[ AND 2022 conf las nv splunk vegas ]`

- All events matching the predicate are scanned
  - "Scanned" includes these steps: Read journal slice off disk, uncompress, fields, eventtypes, tags, lookups, postfilter
- The fewer events you need to scan, the faster your search

- Lispy is visible in `search.log`: `<timestamp> INFO  UnifiedSearch - base lispy: [ ... ]`
- Check lispy efficiency by comparing `eventCount` with `scanCount` from the Job Inspector

How to Find Naughty Searches?

```
index=_audit search_id TERM(action=search) (info=granted OR info=completed)
| transaction search_id startswith=(info=granted) endswith=(info=completed)
| eval lispy_efficiency = event_count / scan_count
| where scan_count > 100 AND total_run_time > 5 AND lispy_efficiency < 0.5
| table _time total_run_time event_count scan_count
        lispy_efficiency user savedsearch_name search
```

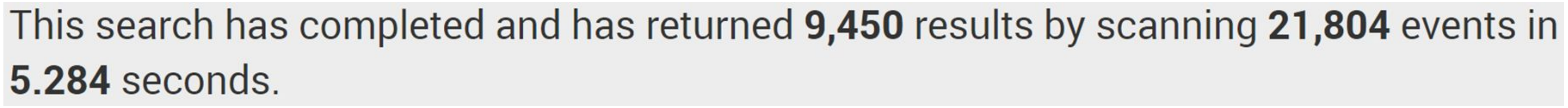
- Adjust thresholds as needed
- Finds some false positives, e.g. itself 😅
- Stats? Sure:

```
index=_audit search_id TERM(action=search) (info=granted OR info=completed)
| stats first(_time) as _time first(total_run_time) as total_run_time
        first(event_count) as event_count first(scan_count) as scan_count first(user) as user
        first(savedsearch_name) as savedsearch_name first(search) as search by search_id
| eval lispy_efficiency = event_count / scan_count
| where lispy_efficiency < 0.5 AND total_run_time > 5 AND scan_count > 100
```

Building the Lispy for a Search

- Every breaker is a major breaker
- Remove duplicates, sort alphabetically

- `127.0.0.1` becomes `[ AND 0 1 127 ]`
- Load all events off disk that contain all three tokens – `scanCount`
- Filter for `127.0.0.1` in the raw event – `eventCount`



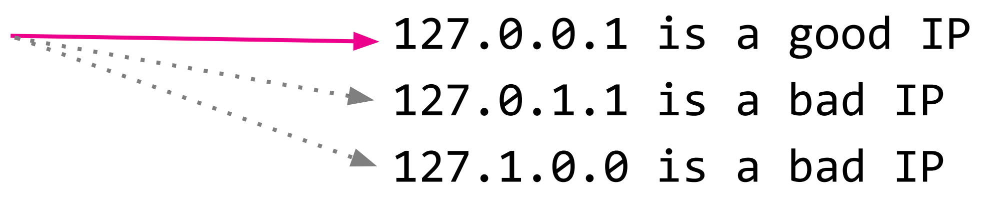
This search has completed and has returned **9,450** results by scanning **21,804** events in **5.284** seconds.

AND and OR Behave

Search	Lispy
foo bar (implicit AND)	[AND bar foo]
foo OR bar	[OR bar foo]
(a AND b) OR (c AND d)	[OR [AND a b] [AND c d]]
(a OR b) AND (c OR d)	[AND [OR a b] [OR c d]]

NOT Can Be Tricky

- `NOT bad` works as expected: `[ NOT bad ]`
- Load all events that don‘t have that token



```
127.0.0.1 is a good IP
127.0.1.1 is a bad IP
127.1.0.0 is a bad IP
```

- How do you translate `NOT 127.0.0.1`?
- `[ NOT [ AND 0 1 127 ] ]`?

NOT Can Be Tricky

- `NOT bad` works as expected: `[ NOT bad ]`
- Load all events that don't have that token

- How do you translate `NOT 127.0.0.1`?
- `[ NOT [ AND 0 1 127 ] ]`?
- That would rule out `127.0.1.1`!

```
127.0.0.1 is a good IP
127.0.1.1 is a bad IP
127.1.0.0 is a bad IP
```

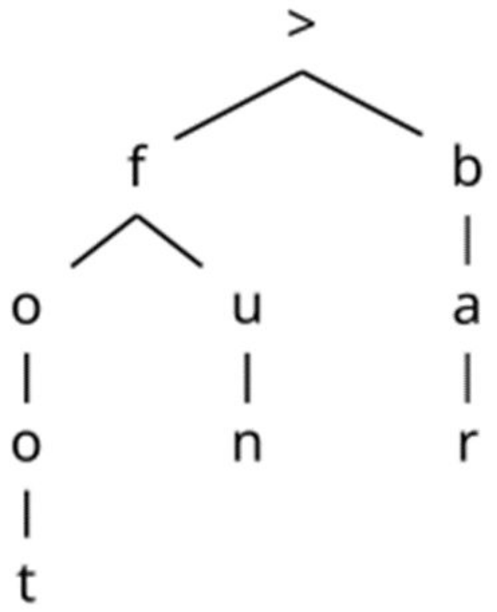
NOT Can Be Tricky

- `NOT bad` works as expected: `[ NOT bad ]`
- Load all events that don't have that token

- How do you translate `NOT 127.0.0.1`?
- `[ NOT [ AND 0 1 127 ] ]`?
- That would rule out `127.0.1.1`!
- The sad reality: `[ AND ]`
- Same story with `NOT "foo bar"`

```
127.0.0.1 is a good IP
127.0.1.1 is a bad IP
127.1.0.0 is a bad IP
```

Wildcards



- Filter for partial matches of indexed tokens
- Imagine indexed tokens are stored as a tree, where each node contains a list of events
- Beware of wildcards at the beginning!

Search	Lispy
foo*	[AND foo*]
f*o	[AND f*o]
*foo	[AND]

TERM()

- Force lispy to use a complex token as a whole, ignoring breakers
- `TERM(127.0.0.1)` becomes `[ AND 127.0.0.1 ]`
- Allows leading wildcards, `TERM(*foo)` becomes `[ AND *foo ]`
- Enables inexact `tstats` queries \o/
  `| tstats count where index=_* TERM(*ucketMover)`
- Can be used with fields: `component=TERM(*ucketMover)`

- Beware: Crawling the index for leading wildcards is IO-intensive
- Related: `CASE(FoO)` doesn't change lispy, just post-filters for case sensitivity

Fields

How are fields used to find events?

splunk> .conf22

Search-Time Vs Index-Time Fields

- Search-time fields are extracted from the raw event while the search runs
- Default assumption: Field values are made up of whole indexed tokens
- `exception=java.lang.NullPointerException` yields `[ AND java lang NullPointerException ]`
- Great flexibility, decent search performance, some pitfalls

- Index-time fields are stored in tsidx files during ingest
- Search for `source=foo timestartpos>0`, get `[ AND source::foo [ GT timestartpos 0 ] ]`
- Great search performance, no flexibility, some disk space overhead

Searching For Index-Time Fields

- Splunk® assumes all fields are search-time fields unless defined otherwise
- Force treatment as index-time field by searching `index_delta::8` to get `[ EQ index_delta 8 ]`

- Global settings in `fields.conf` have always been available: `[index_delta] INDEXED = true`
- Turns `index_delta=8` into `[ EQ index_delta 8 ]` for all sourcetypes

- Teach Splunk® 8.1+ with scoped `fields.conf`: `[sourcetype::splunkd::index_*] INDEXED = true`
-
```
[ OR [ AND sourcetype::splunkd [ EQ index_delta 8 ] ]  ← indexed field for splunkd only
     [ AND 8 [ NOT sourcetype::splunkd ] ]             ← search-time behavior for the rest
]
```

Calculated Fields

- Call an eval expression at search time: `[stanza] EVAL-vendor="Splunk®"`
- Field values don't need to be indexed tokens, harder to filter in lispy
- TL;DR: `coalesce()` and `if()` are good

- Some types of expressions propagate into lispy (fast)
  - coalesce(dest_host, dest_ip) and if(cond, dest_host, dest_ip) are lispy'd like a dest_host OR dest_ip
  - lower(dest_host) is treated like dest_host
  - vendor="Splunk®" scans the entire sourcetype but filters well for others: [OR sourcetype::splunkd splunk ®]
  - vendor="`Buttercup`" filters well: `[ AND buttercup ]`
- Some types of expressions don't propagate into lispy (slow)
  - `case(cond1, val1, cond2, val2)` is not lispy'd like `val1 OR val2` → always use `if()`
  - `nullif(val, "-")` is not lispy'd → always use `if()`
  - Value-changing operations such as arithmetic or string operations can't be propagated or worked around
- With some expressions it depends
  - `lower(coalesce(dest_host, dest_ip))` is slow, `coalesce(lower(dest_host), lower(dest_ip))` is fast ¯\_(ツ)_/¯

Comparisons

- Access logs, search for server errors: `status>=500`
- What indexed token to scan for? None, load all the things: `[ AND ]`

- Can be solved by listing values: `status IN (500,501,502,503,504,505,506,507,508,510,511)`
- Can be solved with a lookup of known server error codes (CIM App)
- Can be solved with an indexed field

- Non-solution: `status=5*`, lispy is `[ AND 5* ]`
- Too many events have a token beginning with 5 somewhere: times, IPs, bytes, versions, etc.
- Really, really, REALLY bad: `status=2*`
  - Many events contain nearly-unique 2022-02-22T22:22:22.222222222Z tokens, can be very slow

Value Uniqueness

- `2022-06-15 12:34:56.789 uid=2022 syscall=2 ...`
- Search for `uid=2022`, get `[ AND 2022 ]`
- Token `2022` is not very unique, scans all events from that year
- Common offenders: Small numbers, `true`, `yes`, `ERROR`, etc.

- Can be solved with an indexed field
- Can sometimes be solved with `TERM(uid=2022)`
  - Beware of `uid="2022"` in your raw event – major breakers break `TERM()`

Remember `NOT`? Tricky…

- `NOT bad` worked well: `[ NOT bad ]`
- What about `NOT field=bad`?
- Index-time? No problem: `[ NOT field::bad ]`
- Search time? `[ NOT bad ]`?

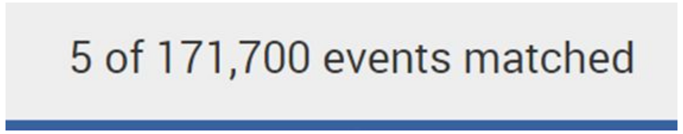
Remember `NOT`? Tricky…

- `NOT bad` worked well: `[ NOT bad ]`
- What about `NOT field=bad`?
- Index-time? No problem: `[ NOT field::bad ]`
- Search time? `[ NOT bad ]`?

- That would rule out events like this: `field=good otherfield=bad`!
- Instead, Splunk® must scan all the events

Key Takeaways

Job Inspector,
Job Inspector,
Job Inspector!

1. Love thy Job Inspector
2. Think of lispy when writing searches
3. Level 2: Think in lispy
4. Carefully consider opportunities for index-time fields
5. Give extra scrutiny to…
   a. Searches using wildcards
   b. Small numbers
   c. Filtering with `NOT` – especially for fields
   d. Calculated fields
   e. These: 
      5 of 171,700 events matched

Thank You

splunk> .conf22