splunk> .conf2017

Fields, Indexed Tokens, And You

SPLUNK TRUST 2015-2016

SPLUNK TRUST 2016-2017

SPLUNK TRUST 2017-2018

Martin Müller | Professional Services Consultant, Consist Software Solutions GmbH

September 42nd, 2017 | Washington, DC

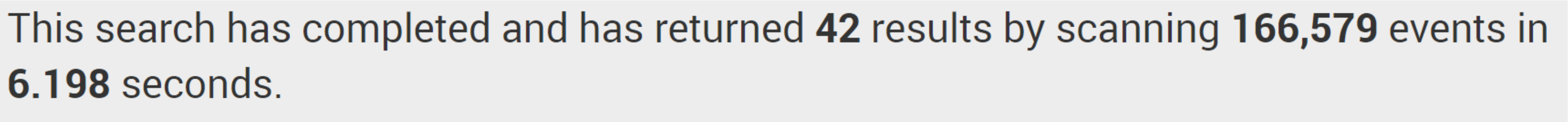
Forward-Looking Statements

During the course of this presentation, we may make forward-looking statements regarding future events or the expected performance of the company. We caution you that such statements reflect our current expectations and estimates based on factors currently known to us and that actual events or results could differ materially. For important factors that may cause actual results to differ from those contained in our forward-looking statements, please review our filings with the SEC.

The forward-looking statements made in this presentation are being made as of the time and date of its live presentation. If reviewed after its live presentation, this presentation may not contain current or accurate information. We do not assume any obligation to update any forward looking statements we may make. In addition, any information about our roadmap outlines our general product direction and is subject to change at any time without notice. It is for informational purposes only and shall not be incorporated into any contract or other commitment. Splunk undertakes no obligation either to develop the features or functionality described or to include any such feature or functionality in a future release.

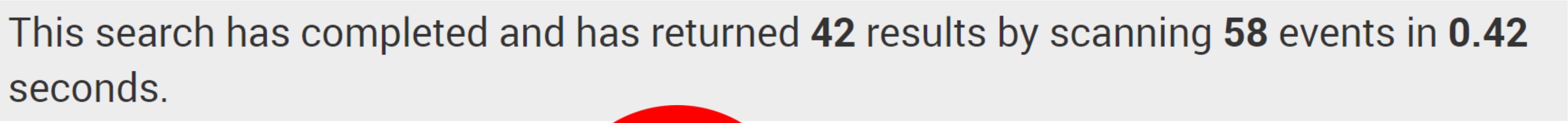
Why Are We Here?

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



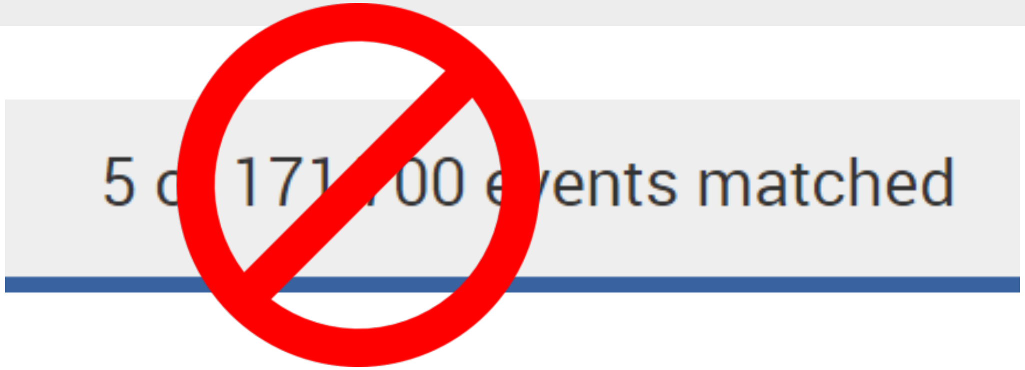
This search has completed and has returned **42** results by scanning **166,579** events in **6.198** seconds.

…into this!



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

…this is bad:

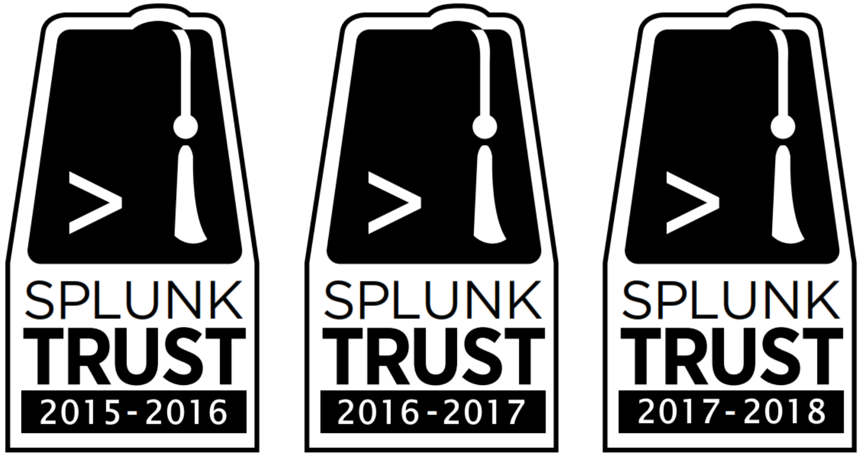


5 of 171,[illegible]00 events matched

Who’s That Guy?

- Professional Services Consultant II, Certified Architect II, Splunk Trustee x3
- Seven years at EMEA Splunk Partner CONSIST Business Information Technology
- Heavy Splunker since 2012

- Get in touch with me: martin.mueller@consist.de
- Give karma at Splunk Answers: martin_mueller
- Join us on Slack: splunk-usergroups.signup.team



Session Objectives

- Understand how Splunk turns a logfile into indexed tokens
- Learn how your searches make good use of indexed tokens (or not)

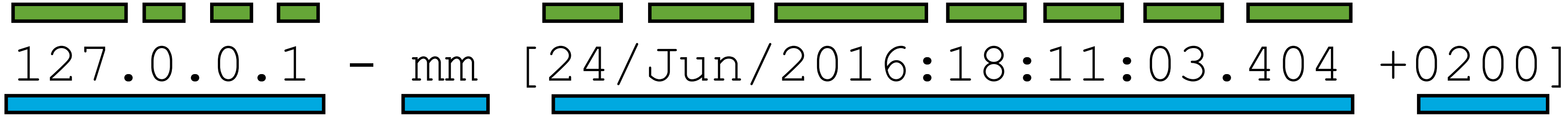
- Topics in detail:
  - Breakers & Segmentation
  - Lispy
  - Fields

Breakers & Segmentation

How Does Splunk Break Events Into Indexed Tokens?

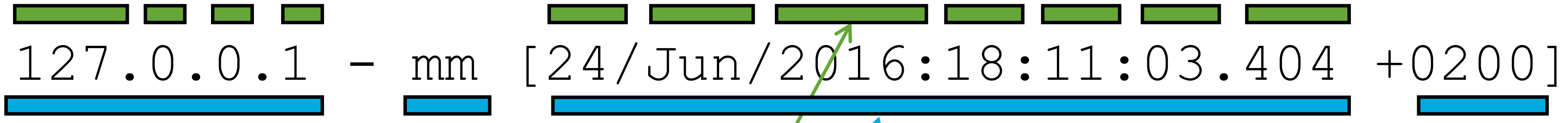
How Splunk Chops Up An Event

- 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



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

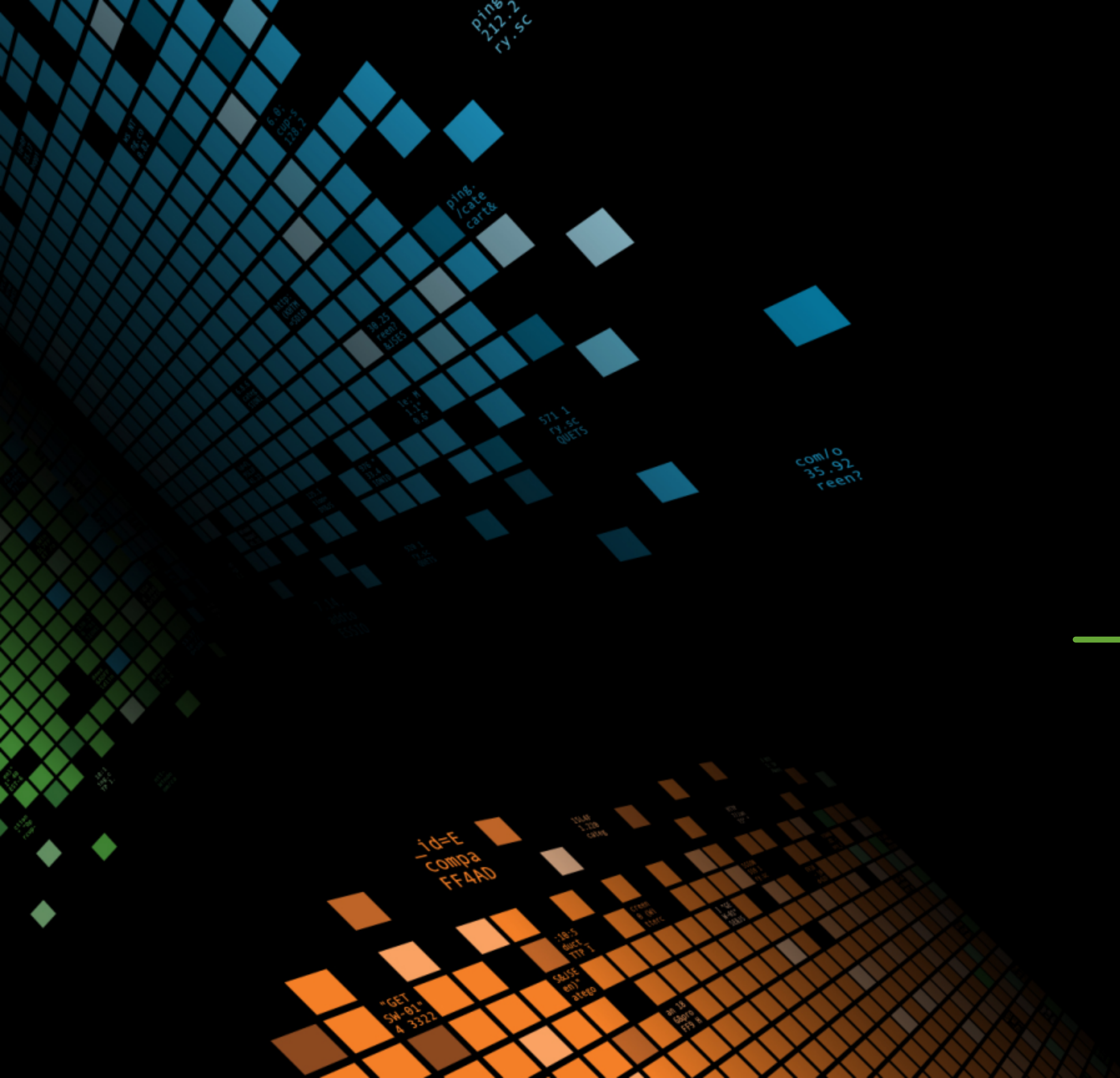
```
bin>splunk cmd walklex ..\var\lib\splunk\conf2016_segmentation\db\hot_v1_1\1466784663-1466784663-15369347184008592423.tsidx ""
```

```
my needle:
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

Room For Optimization

- Look for high-cardinality groups of tokens you don‘t search for
- Common offender: Textual timestamp representations: `24/jun/2016:18:11:03.404`
- You don‘t filter for „events from June“ by searching for `jun`
- Segmenters.conf lets you filter out unwanted parts of your events
- Beware: Easy to break stuff, hard to define filters in some cases
- More info available at http://www.duanewaddle.com/splunk-bucket-lexicons-and-segmentation/



Lispy

How Does Splunk Find Events Matching Your Search?

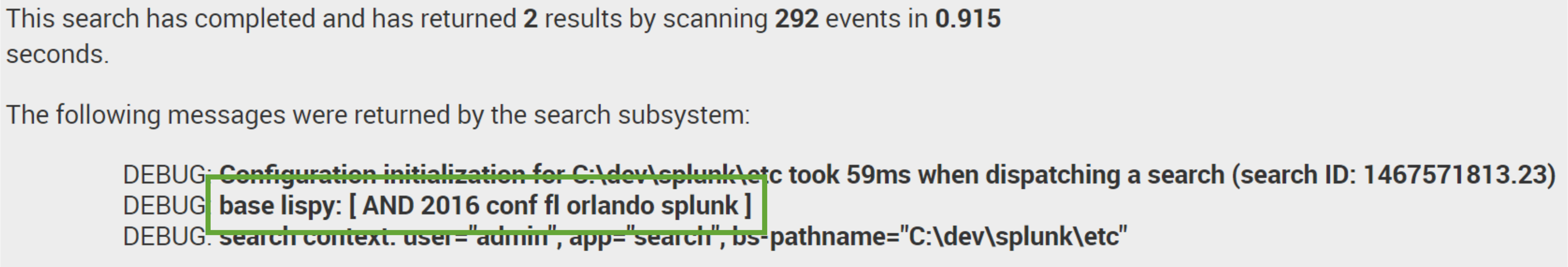
Lispy??

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

- Square brackets, prefix notation for operators? That‘s lispy.
- Search for `splunk.conf 2018 – Orlando, FL` and you get
  `[ AND 2018 conf fl orlando splunk ]`
- All events matching the predicate are scanned

Job Inspector

- Since 6.2, lispy is by default only visible in `search.log`
- Enable the old-fashioned header in `limits.conf`:
  `[search_info] infocsv_log_level=DEBUG`



This search has completed and has returned **2** results by scanning **292** events in **0.915** seconds.

The following messages were returned by the search subsystem:

```
DEBUG: Configuration initialization for C:\dev\splunk\etc took 59ms when dispatching a search (search ID: 1467571813.23)
DEBUG: base lispy: [ AND 2016 conf fl orlando splunk ]
DEBUG: search context: user="admin", app="search", bs-pathname="C:\dev\splunk\etc"
```

- Check lispy efficiency by comparing `eventCount/scanCount`

How To Find Naughty Searches?

Find start and end events for 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
```

Group by search ID

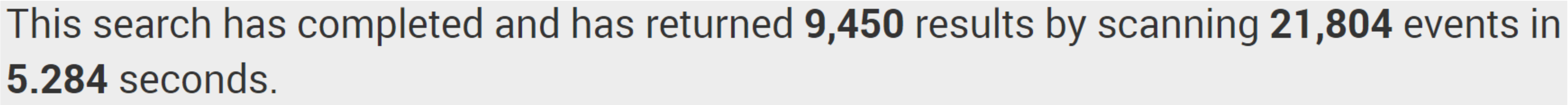
Do maths, apply filters, and select fields

- 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
- Some additional optimizations
- `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 ] ]`

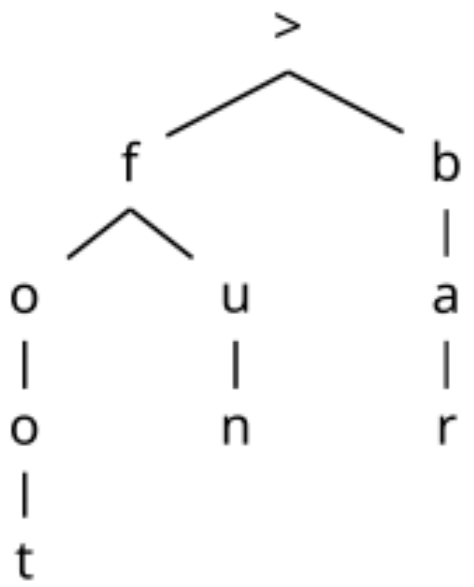
NOT Can Be Tricky

- `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 ] ]`?
- That would rule out `127.0.1.1`!
- The sad reality: `[ AND ]`
- Same story with `NOT "foo bar"`

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 ]`

Wildcards Can Be Tricky

- Wildcards in combination with breakers lead to unexpected results

- `Hello W*rld` gives you `[ AND hello w*rld ]` – great!
- `Hello*World` gives you `[ AND hello*world ]` – oops!
- There is no indexed token matching this lispy!

Wildcards Can Be Really Tricky

- Wildcards in combination with breakers lead to unexpected results

- Say your events contain `java.lang.NullPointerException`
- Indexed tokens: `java lang NullPointerException`
  `java.lang.NullPointerException`
- `java*Exception` / `[ AND java*Exception ]` – great!
- `java.lang.*Exception` / `[ AND java lang ]` – fine!
- `java.lang*Exception` / `[ AND java lang*Exception ]` – oops!

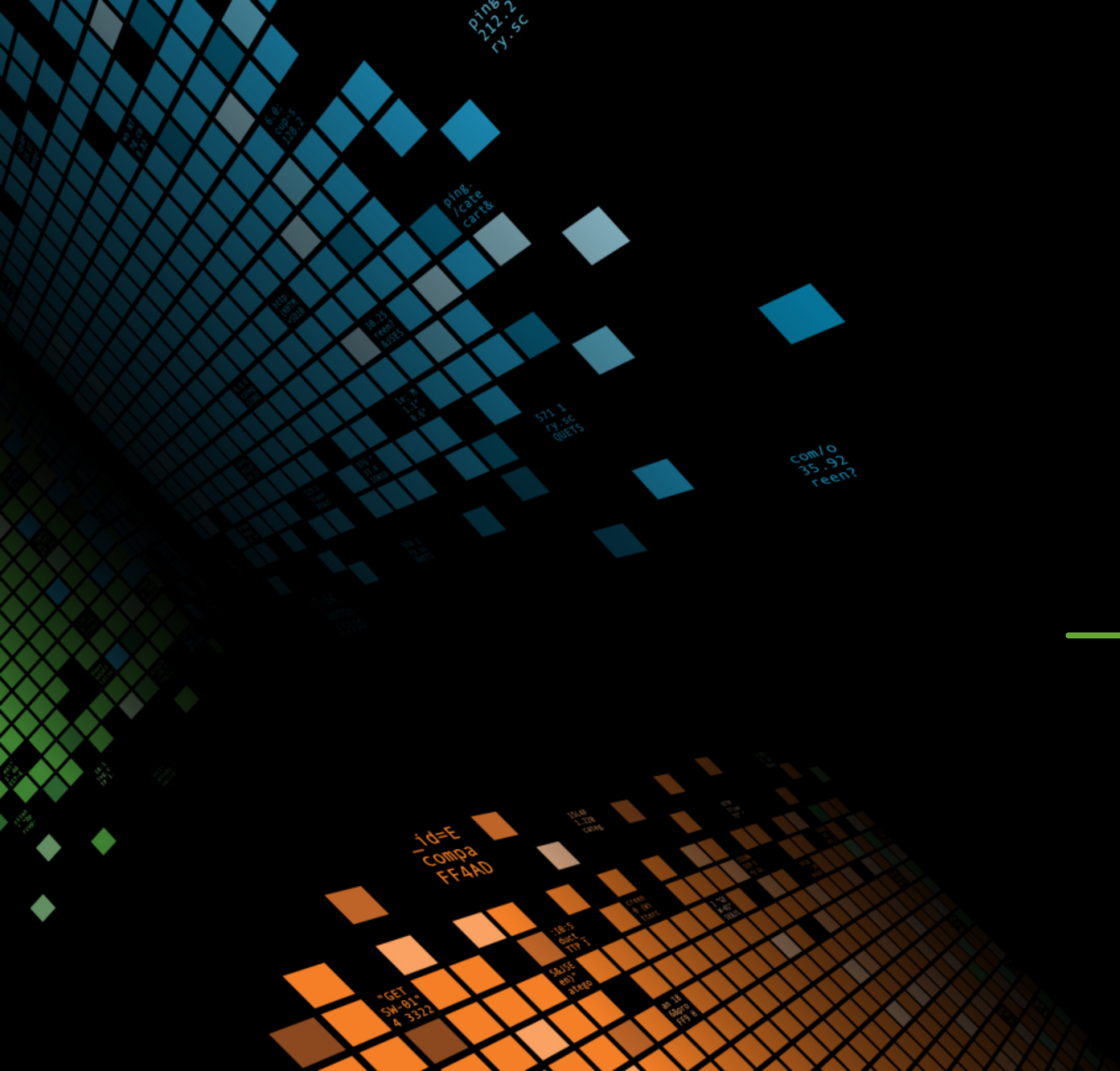
- In short: Be very very careful around wildcards

TERM()

- Force lispy to use a complex token, ignore 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)` produces case-sensitive lispy expressions



Fields

Unprecedented

Search-time Fields

- Field values are extracted from the raw event while the search runs
- Default assumption: Field values are whole indexed tokens
- `exception=java.lang.NullPointerException` becomes
  `[ AND java lang NullPointerException ]`
- Actual field extractions and post-filtering happens after loading raw events
- Pro: Flexibility, scoping, mostly decent performance
- Con: Terrible performance in some cases, partial tokens pitfall

Index-time Fields

- Default fields: `host`, `source`, `timestartpos`, etc.
- Custom fields in `transforms.conf` (`WRITE_META=true`)
- Pro: Search performance
- Con: Flexibility, lack of sourcetype namespace
- Con if over-used: Indexing overhead, disk space

- Search for `sourcetype=foo timestartpos>0`

  ```
  [ AND sourcetype::foo [ GT timestartpos 0 ] ]
  ```

Define Custom Index-time Fields

- `transforms.conf`: `REGEX`, `FORMAT`, `WRITE_META`
- `props.conf`: `TRANSFORMS-class = stanza`
- `fields.conf`: `[fieldname] INDEXED = true`

- ...`fields.conf`?
- Tells the search that a field is expected as an indexed field (lispy `::`)
- Not scoped to a `props.conf` stanza such as sourcetype!
- Trying to work around `fields.conf` with field aliases is futile
- Use `field::value` in search to explicitly access indexed field

Calculated Fields

- Call an eval expression at search time: `[stanza] EVAL-answer=42`
- Field values don’t have to be indexed tokens, hard to filter in lispy
- `answer=42` becomes `[ OR 42 sourcetype::stanza ]`
- Scan all events for the field value plus all events for that stanza
- Common use case: CIM normalization, e.g. Bluecoat TA:
  `EVAL-dest=coalesce(dest_ip, dest_host)`
- No pre-search optimization
- Use sparingly when searching by a field

Comparisons

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

- Can be solved with a lookup of known server error codes (CIM App)
- Can be solved with an indexed field

- Non-solution: `status=5*`, `[ AND 5* ]`
- Too many events have a `5*` token somewhere

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 has to scan all the events

Value Uniqueness

- `2016-09-28 12:34:56.789 uid=2016 syscall=2`
- Search for `uid=2016`, get `[ AND 2016 ]`
- Token is not very unique, scans all events from that year
- Common offenders: Small integers, `true`, `yes`, `ERROR`, etc.

- Can be solved with an indexed field
- Can sometimes be solved with `TERM(uid=2016)`
- Beware of `uid="2016"` – major breakers break `TERM()`

Fields From Partial Tokens

- Any financial services people? – `DE44500105175407324931`
- Extract fields: `(?<country>[A-Z][A-Z])(?<check>\d\d)`…
- Search for `country=DE`, get lispy `[ AND DE ]` – oops!
- Can be fixed by `fields.conf` (but beware of scoping!)

```
[country] INDEXED_VALUE = <VALUE>*
```

- Search for `check=44` – fixing in `fields.conf` gets ugly

```
[check] INDEXED_VALUE = *<VALUE>*
[check] INDEXED_VALUE = false
```

What About Accelerations?

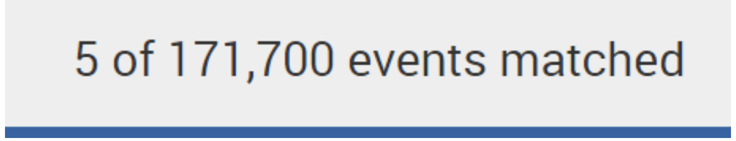
- Accelerated Datamodels and Reports get filled by frequent searches
- Users of accelerations get a large performance boost regardless of their lispy efficiency – good!

- However!
- The frequent summarizing searches should be well-optimized
- Rule of thumb: The more often something will run for a long time into the future, the more time you should spend on optimizations

Key Takeaways

Job Inspector,
Job Inspector,
Job Inspector!

1. Love thy Job Inspector
2. Start to 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…
   - Searches using wildcards
   - Small numbers
   - Filtering through `NOT` – especially for fields
   - Calculated fields
   - These:



5 of 171,700 events matched

Thank You

Don't forget to **rate this session** in the .conf2017 mobile app

splunk> .conf2017