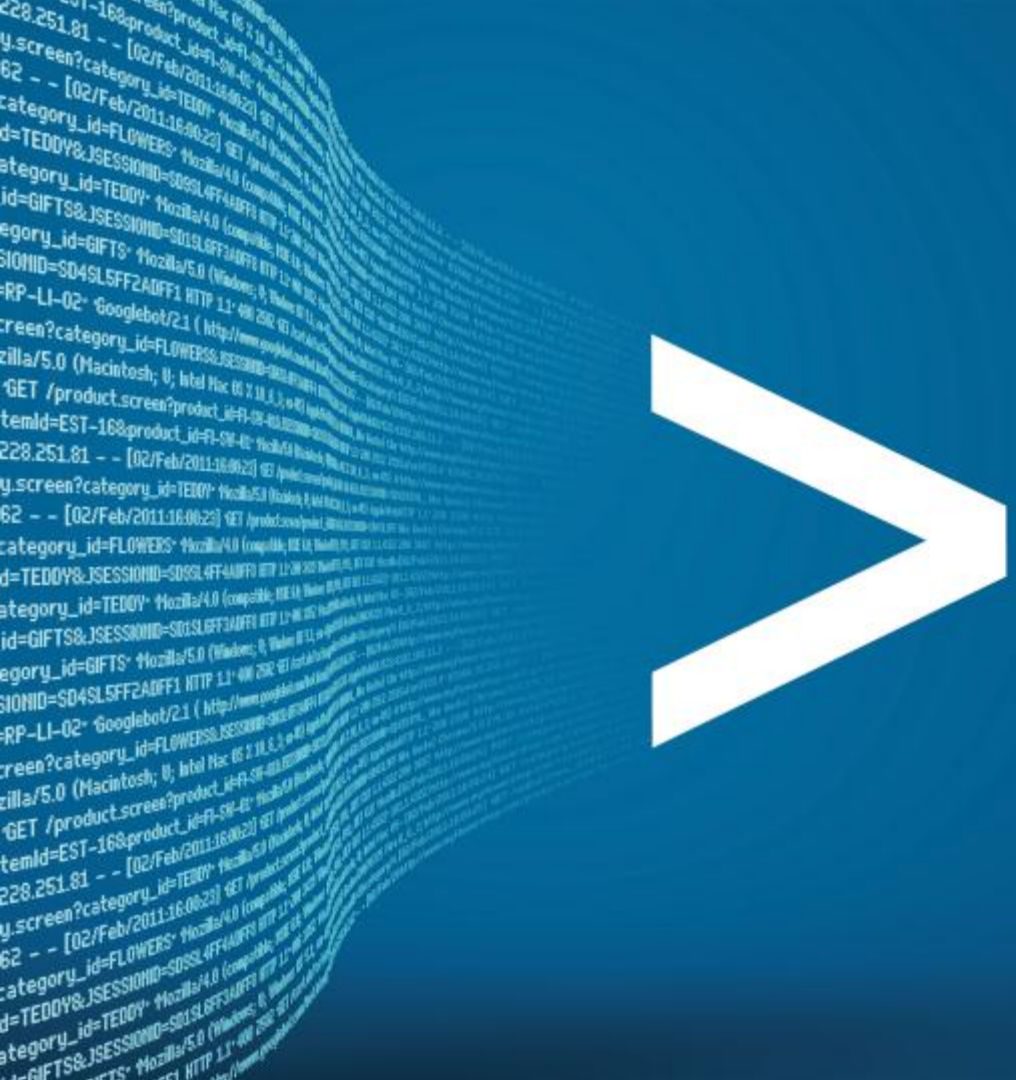
Copyright © 2014 Splunk, Inc.



A Quick Guide To Search Optimization

Naman Joshi
Snr Sales Engineer

splunk>

Disclaimer

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 the 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.

splunk> listen to your data™

Agenda

- **Search Scoping:** A little background on Splunk Internals
- **Search Optimization tools:** SOS and Job inspector
- **Laying the groundwork for:** Regular Expression optimization
- **Beyond the basics:**
  - Joining Data
  - Transactions with Stats
  - Optimizing transaction
- **Bonus:**
  - Using tstats

Who's This Dude?

Naman Joshi

nbjoshi@splunk.com
Senior Sales Engineer

- Splunk user since 2008
- Started with Splunk in Feb 2014
- Former Splunk customer in the Financial Services Industry
- Lived previous lives as a Systems Administrator, Engineer, and Architect

Philosophy behind Search Optimization

- Don't feel the need to optimize every single search - focus on those which are frequently used and have the best potential for speedup. - KISS
- Understand the whole problem
- Know a small number of tricks well

splunk> listen to your data™

How Can We Make Things Faster?

For all Searches:

- Change the physics (do something different)
- Reduce the amount of work done (optimize the pipeline)

In distributed Splunk environments particularly:

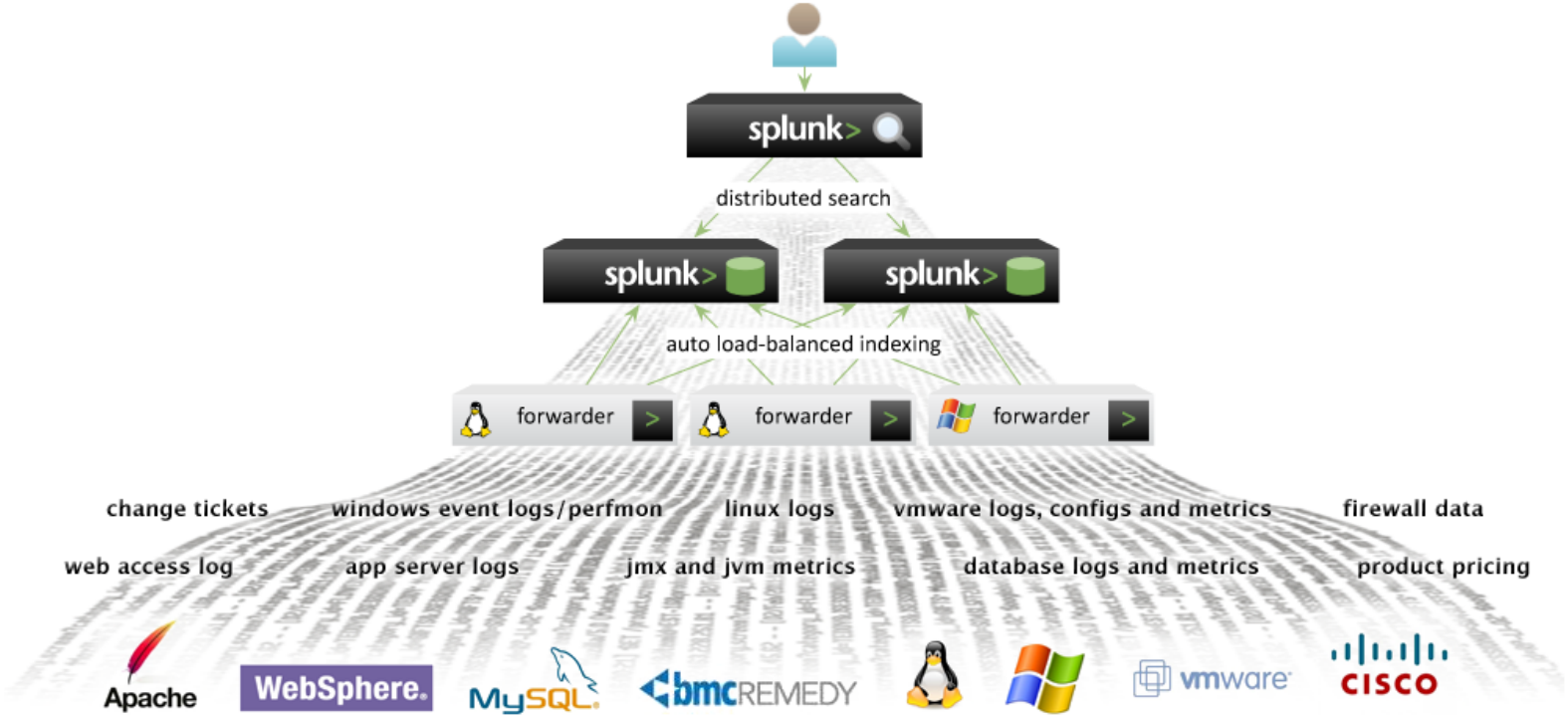
- How can we ensure as much work as possible is distributed?
- How can we ensure as little data as possible is moved?

splunk> listen to your data™

Quick Overview

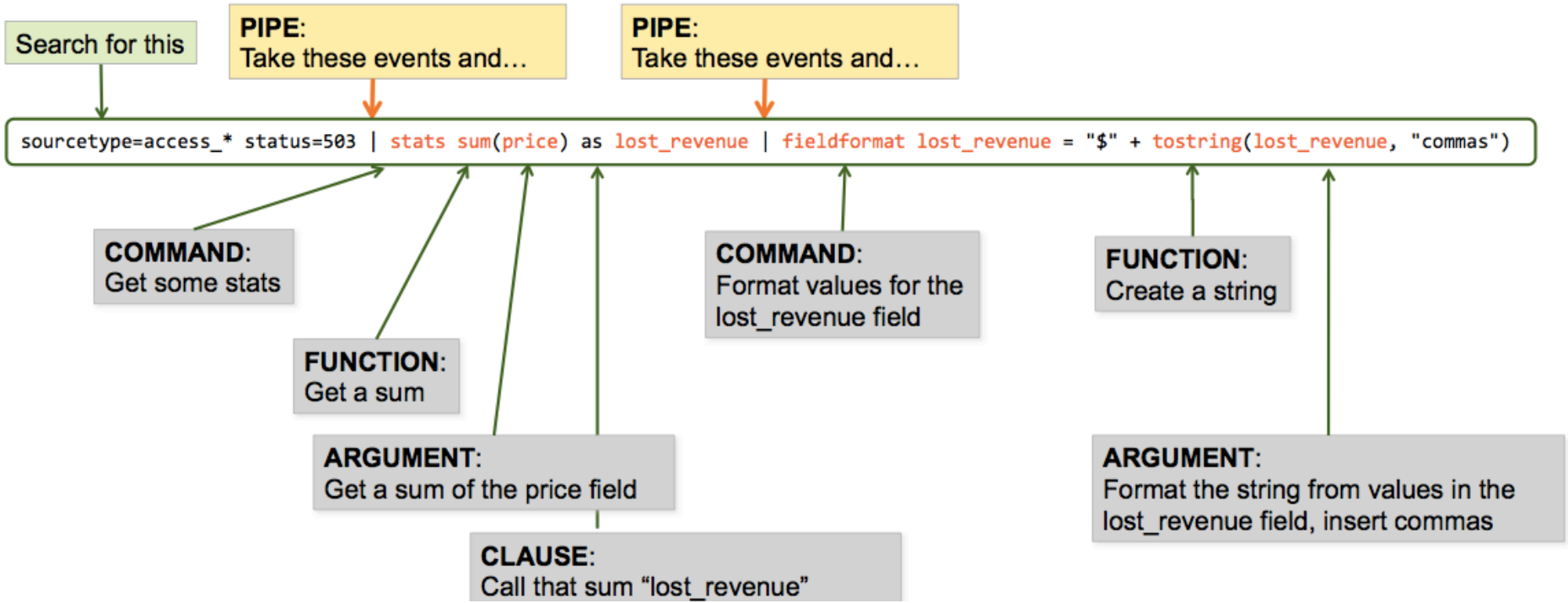
splunk>

Splunk Review



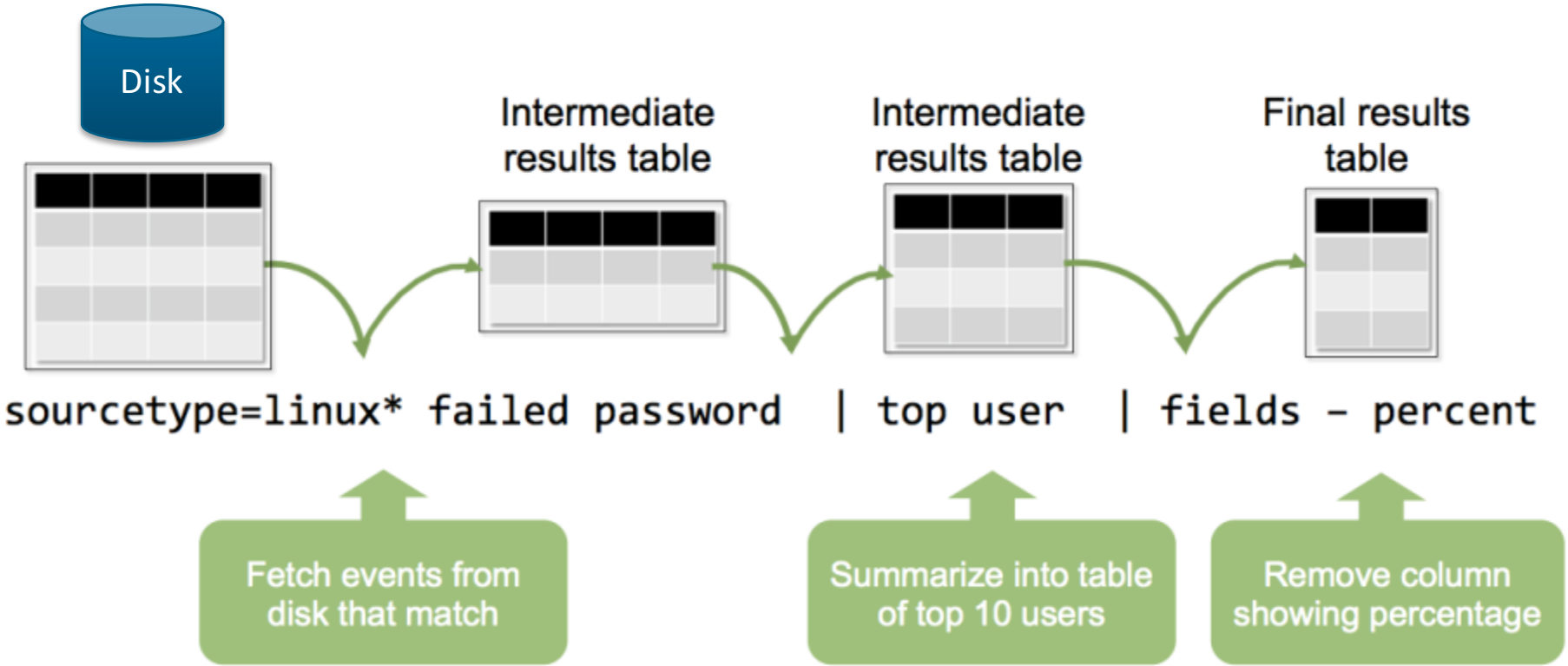
splunk> listen to your data™

Search Syntax Components



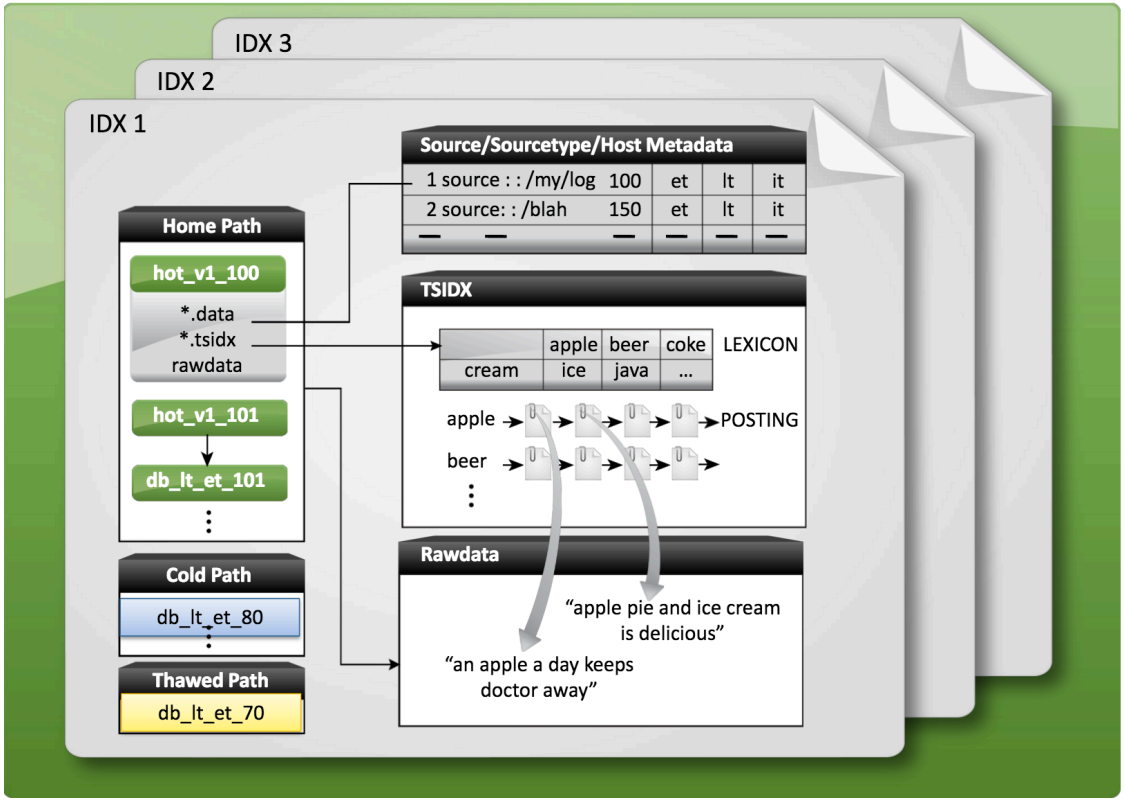
splunk> listen to your data™

Anatomy of a Search



splunk> listen to your data™

Splunk's Index Structure



splunk> listen to your data™

The Basics – Search Scoping

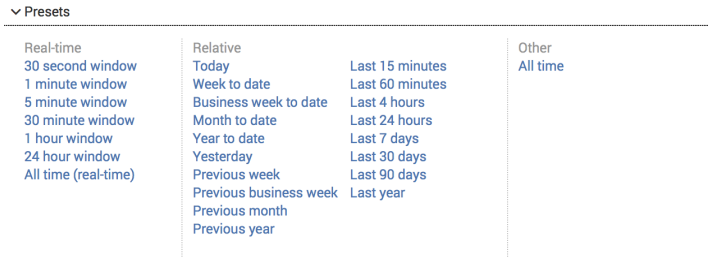
splunk>

Time Range

- Splunk organizes events into buckets by time, which contain events
- The shorter the time range the fewer buckets will be read
- !!Common Practice: Searches running over all time!!

Time Range

- Good Practice: Scope to an appropriate shorter time range (using time range picker or earliest=/latest= or _index_earliest=/_index_latest=)
- Speedup Metric: 30x – 365x
- Example: All Time -> Week to Date



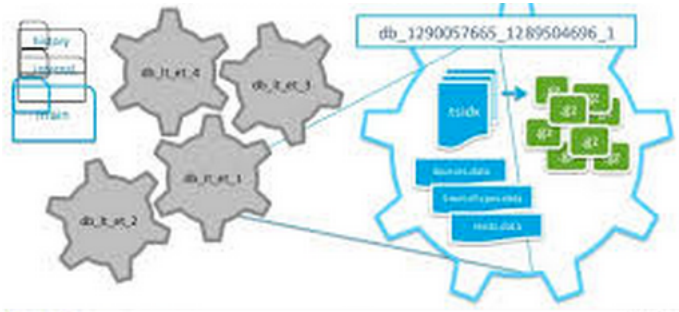
ui-prefs.conf.spec [edit]

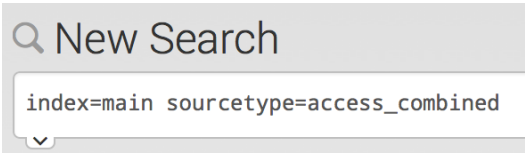
```
#   Version 6.2.5
#
# This file contains possible attribute/value pairs for ui preferences for a view.
#
# There is a default ui-prefs.conf in $SPLUNK_HOME/etc/system/default. To set custom
# configurations, place a ui-prefs.conf in $SPLUNK_HOME/etc/system/local/. To set cus
```


splunk> listen to your data™

Scope on Metadata Fields

- Index is a special field, controlling which disk location will be read to get results
- All events in Splunk have sourcetype and source fields and including these will improve speed and precision
- Common Practice: Often roles include ‘All-non internal indexes’, no index or sourcetype specfier
- Diagnostic: look for searches without explicit index= clauses





splunk> listen to your data™

Scope on Metadata Fields

- Good practice: include a specific index=, sourcetype= set of fields. If using multiple related sourcetypes, use eventtypes which also include a sourcetype scope
- Expected Speedup: 2x – 10x
- Example
  - Before : MID=*
  - After: index=cisco sourcetype=cisco:esa:textmail MID=*
  - Using Eventtypes: index=cisco eventtype=cisco_esa_email with (sourcetype="cisco:esa:textmail" OR sourcetype=cisco:esa:legacy) AND (MID OR ICID OR DCID)

splunk> listen to your data™

Search Modes

- Splunk's search modes control Splunk's tendency to extract fields, with verbose being the most expansive and exploratory and fast being the least
- Diagnostic: request.custom.display.page.search.mode = verbose
- Common Practice: Verbose Mode left on after using
- Good Practice: Use Smart or Fast mode (dashboard searches do this automatically)
- Speedup Metric: 2x -5x

splunk> listen to your data™

Inclusionary Search Terms

- Inclusionary search terms specify events to keep
- Exclusionary search terms specify events to remove
- Exclusions are appropriate for many use cases (interactive usage, exclusion of known errors, specificity)

splunk> listen to your data™

Inclusionary Search Terms

- Diagnostic: Large scan numbers versus final events
- Good Practice: Mostly inclusionary terms, small or no exclusionary terms
- Speedup Metric: 2x -20x

```
index=main sourcetype=access* NOT action=purchase
```

```
index=main sourcetype=access* AND (action=addtocart OR action=view OR action=new)
```


splunk> listen to your data™

Field Usage

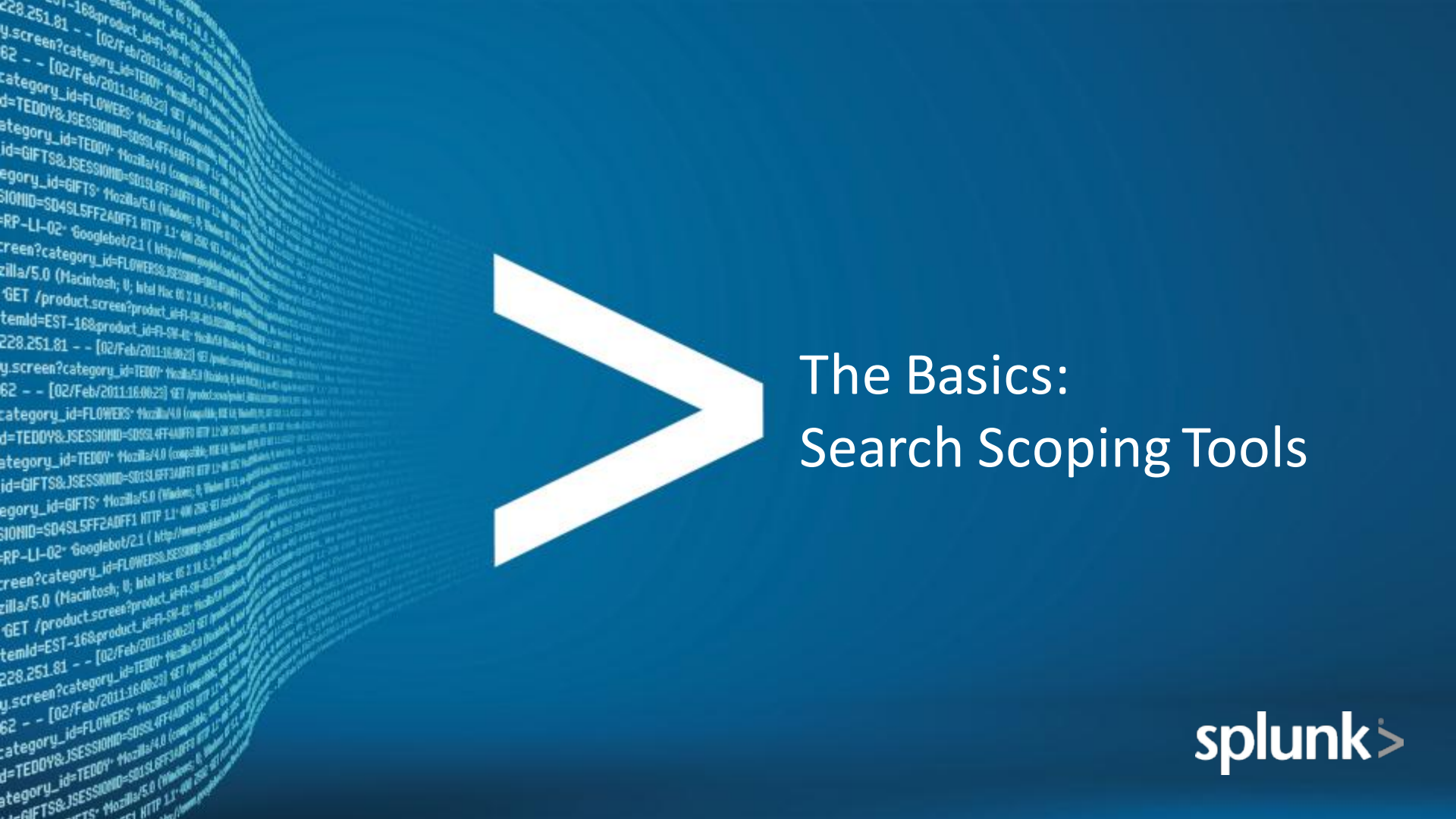
- Define fields on segmented boundaries where possible
- Splunk will try to turn field=value into value, can be customized with fields.conf/segmentors.conf
- Diagnostic: check the base lispy in your search.log

splunk> listen to your data™

Field Usage

- Good practise: Repeat field values as search terms if required, or use fields.conf
- Example:
  - Before: guid=942032a0-4fd3-11e5-acd9-0002a5d5c5
  - After: (index=server sourcetype=logins 942032a0-4fd3-11e5-acd9-0002a5d5c5 guid=942032a0-4fd3-11e5-acd9-0002a5d5c5) OR (index=client eventtype=client-login source=/var/log/client/942032a0-4fd3-11e5-acd9-0002a5d5c5)

splunk> listen to your data™



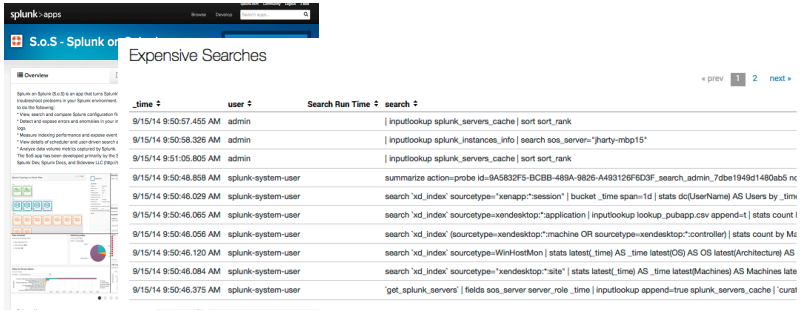
The Basics:
Search Scoping Tools

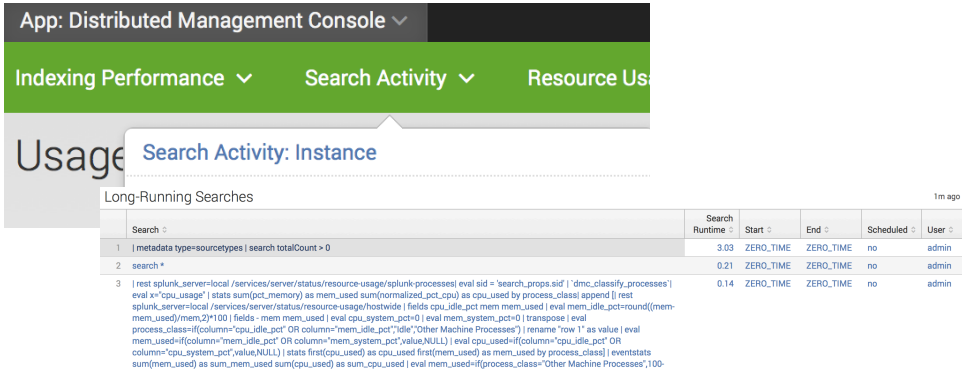
splunk>

A Word On Monitoring Searches

How do we easily identify less than optimal searches?

- SOS (Pre 6.1 Users)
- Distributed Management Console
- Job Inspector



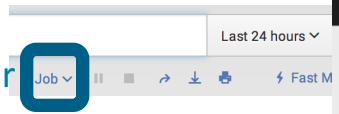


splunk> listen to your data™

Measuring Search

Using the Splunk Search Inspector

Job Inspector



Search job inspector

This search has completed and has returned **1,178** results by scanning **1,783** events in **0.169** seconds.

The following messages were returned by the search subsystem:

DEBUG: **base lispy: [AND]**
DEBUG: **search context: user="tz", app="search", bs-pathname="/opt/customers/**

(SID: 1410925035.148)

Key Metrics:

- Completion Time
- Number of Events Scanned
- Search SID

Execution costs

Duration (seconds)	Component	Invocations	Input count	Output count
0.064	command.fields	67	437,295	437,295
70.029	command.remotetl	67	437,295	-
249.679	command.search	67	-	437,295
139.011	command.search.lookups	106	516,763	516,763
65.239	command.search.typer	67	437,295	437,295
13.921	command.search.tags	67	437,295	437,295
12.363	command.search.kv	106	-	-
11.834	command.search.rawdata	106	-	-
4.719	command.search.filter	106	-	-
2.264	command.search.index	327	-	-
0.109	command.search.fieldalias	106	516,763	516,763
0.407	dispatch.createProviderQueue	1	-	-
319.567	dispatch.stream.remote	57	-	15,711,327
83.738	dispatch.stream.remote.fool06.splunk.com	14	-	3,999,012
80.418	dispatch.stream.remote.fool04.splunk.com	13	-	3,932,267
78.599	dispatch.stream.remote.fool03.splunk.com	14	-	3,983,522
76.811	dispatch.stream.remote.fool05.splunk.com	15	-	3,776,581
0.001	dispatch.stream.remote.fool02.splunk.com	1	-	19,945
0.91	dispatch.timeline	73	-	-

Timings from the search command

Timings from distributed peers

splunk> listen to your data™

Job Inspector Walkthrough – Search Command

Execution costs

Duration (seconds)	Component	Invocations	Input count	Output count
0.003	command.fields	13	189	189
0.035	command.lookup	13	189	189
0.015	command.remotetl	13	189	-
0.232	command.search	13	-	189
0.096	command.search.index	60	-	-
0.028	command.search.filter	4	-	-
0.004	command.search.calcfields	4	3,133	3,133
0.004	command.search.fieldalias	4	3,133	3,133
0	command.search.index.usec_1_8	74,656	-	-
0	command.search.index.usec_8_64	[illegible]	-	-
0.052	command.search.rawdata	4	-	-
0.045	command.search.kv	4	-	-
0.018	command.search.lookups	4	3,133	3,133
0.015	command.search.typer	13	189	189
0.013	command.search.tags	13	189	189
0.002	command.search.summary	13	-	-

Rawdata:
Improving I/O and CPU load

KV:
Are field extractions efficient

Lookups:
Used appropriately
Autolookups causing issues

Typer:
Inefficient Eventtypes

Alias:
Cascading alias

splunk> listen to your data™

Laying the Groundwork

splunk>

Field Extractions

Most fields are extracted by regular expressions. Some regular expression operations are much better performing than others.

Field extractions can overlap – multiple TA's on the same source type for example.

Fields can also be from indexed extractions or structured search time parsing, as well as calculated (eval) fields and lookups

splunk> listen to your data™

Duplicate Structured Fields

- Sometimes both indexed extractions and search time parsing are enabled for a CSV or JSON sourcetype. This is repeated unnecessary work, and confusing
- Diagnostic: duplicate data in multivalued fields
- Good Practice: Disable the search time KV
- Example:

```
[my_custom_indexed_csv]
# required on SH
KV_MODE=csv
# required on forwarder
INDEXED_EXTRACTIONS = CSV
```

```
[my_custom_indexed_csv]
# required on SH
KV_MODE=none
# required on forwarder
INDEXED_EXTRACTIONS = CSV
```


splunk> listen to your data™

Basic Regular Expression Best Practice

- Backtracking is expensive
- Diagnostic: high kv time
- Good Practices:
  - Prefer + to *
  - Extract multiple fields together where they appear and are used together
  - Simple expressions are usually better (e.g. IP addresses)
  - Anchor cleanly
  - Test and benchmark for accuracy and speed

splunk> listen to your data™

Basic Regular Expression Best Practice Examples

Before

- ’ (?P<messageid>[^]+)

After

- ^\S+\s+\d+\s+\d\d:\d\d:\d\d \s+\w+\[\d+\]\s+\w+\s+'\d+ \.\d+\.\d+\s+(?P<messageid >[^]+)

splunk> listen to your data™

Reading Job Inspector - search.kv

Execution costs

Duration (seconds)	Component	Invocations	Input count	Output count
0.014	command.fields	14	11,572	11,572
0.012	command.head	14	10,000	10,000
0.017	command.prehead	14	11,572	10,000
0.184	command.rex	14	10,000	10,000
0.634	command.search	28	11,572	23,144
0.037	command.search.index	14	-	-
0.029	command.search.filter	27	-	-
0.017	command.search.fieldalias	13	11,572	11,572
0.014	command.search.calcfields	13	11,572	11,572
0	command.search.index.usec_1_8	2,995	-	-
0	command.search.index.usec_8_64	672	-	-
0.285	command.search.lookups	13	11,572	11,572
0.159	command.search.kv	13	-	-

Search.KV=

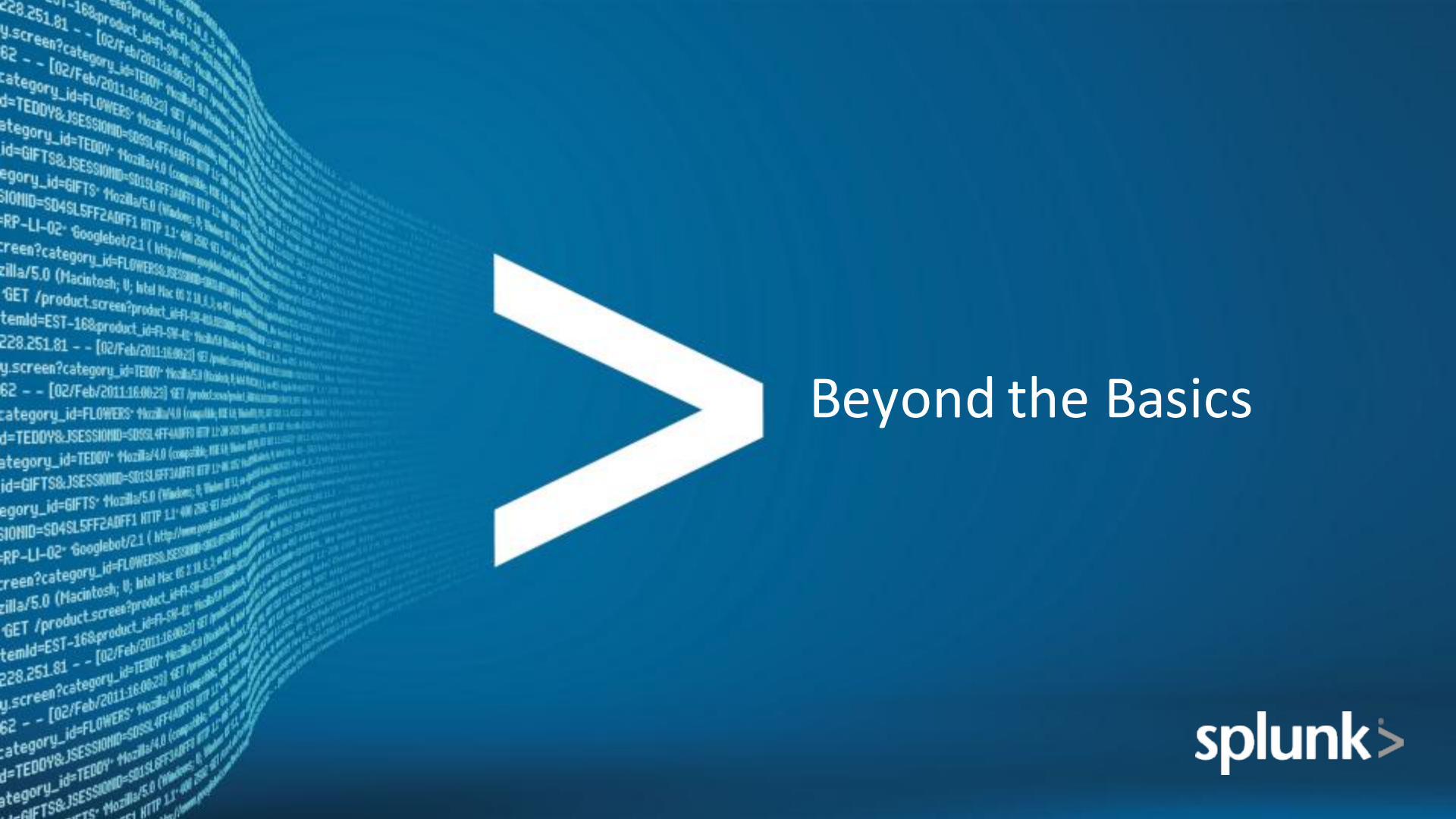
Time taken to apply field extractions to events

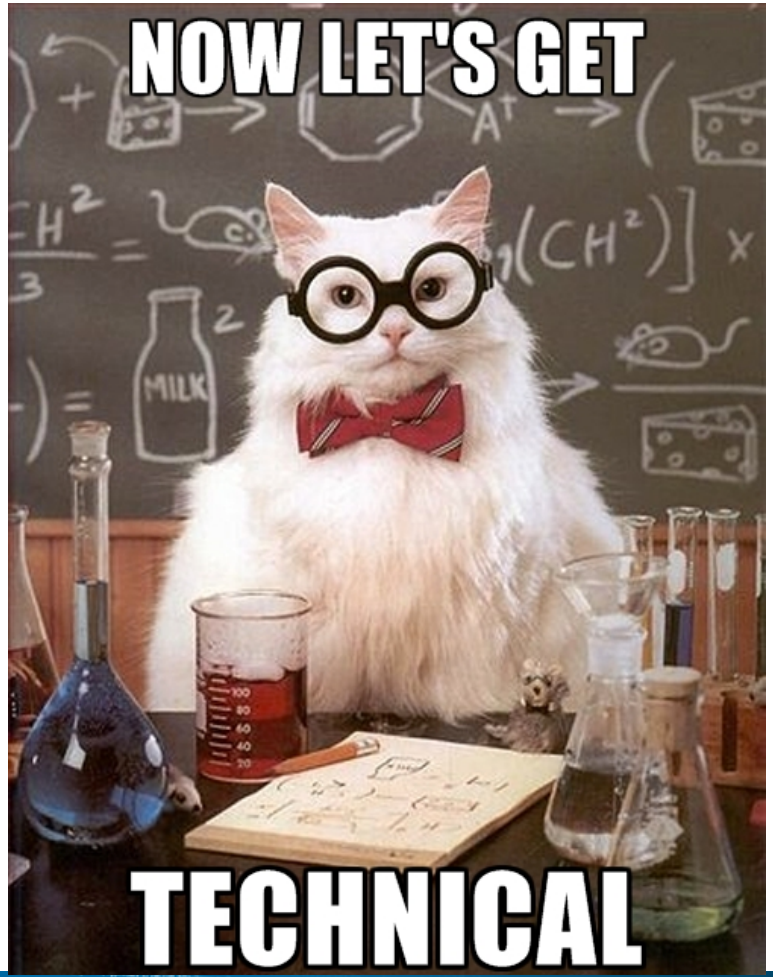
How do you optimize this?

Regex optimizations

- Avoid Backtracking
- Use + over *
- Avoid greedy operators .*?
- Use of Anchors ^ $
- Non Capturing groups for repeats
- Test! Test! Test!

splunk> listen to your data™


Beyond the Basics
splunk>



splunk> listen to your data™

Pretty Searches: Keep it Kosher

Weak:

```
… | rename machine as “host for later” | sort “host for later” |
timechart count by “host for later” span=1h
```

Strong:

```
…| timechart span=1h count by machine
  | sort machine
  | rename machine as “host for later”
```

- new pipe = new line + space + pipe
- | <command> <params> <processing>
- cosmetics at end

splunk> listen to your data™

Pretty Searches: foreach is Clean

Weak:

```
…| timechart span=1h limit=0 sum(eval(b/pow(1024,3))) as size by
st
```

Strong:

```
…| timechart span=1h limit=0 sum(b) by st
  | foreach * [ eval <<FIELD>> = '<<FIELD>>' / pow( 1024 , 3 ) ]
```


splunk> listen to your data™

Pretty Searches: coalesce's Cooler Than if

Weak:

```
…| eval size = if( isnull(bytes) , if( isnull(b) , "N/A" , b ) ,
bytes )
```

Strong:

```
…| eval size = coalesce( bytes , b , "N/A" )
```


splunk> listen to your data™

Faster Searching: Less is More

Weak:

```
iphone
  | stats count by action
  | search action=AppleWebKit
```

Strong:

```
iphone action=AppleWebKit
  | stats count
```


splunk> listen to your data™

Faster Search: Be Specific

Weak:

```
iphone
  | stats count by action
```

Strong:

```
index=oidemo host=dmzlog.splunktel.com sourcetype=access_combined
source=/opt/apache/log/access_combined.log iphone
user_agent="*iphone*”
  | stats count by action
```

Time selector and eventtypes/tags!

splunk> listen to your data™

Faster Searching: Require Fields

Weak:

```
iphone
  | stats count by action
```

Wrong Results:

Pulls both phone=iphone and user_agent=*iphone*

Strong:

```
phone=iphone action=*
  | stats count by action
```


splunk> listen to your data™

Faster Searching: Stats vs dedup/transaction

Weak:

```
… phone=*
  | dedup phone
  | table phone
  | sort phone
```

```
… phone=*
  | transaction host
  | table host, phone
```

Strong:

```
… phone=*
  | stats count by phone, host
  | fields - count
```


splunk> listen to your data™

Faster Searching: Avoid Subsearches

Weak:

```
index=burch | eval blah=yay
 | append [ search index=simon | eval blah=duh ]
```

Strong:

```
( index=burch … ) OR ( index=simon …)
 | eval blah=case( index==”burch" , "yay" , index==”simon" ,
"duh" )
```


splunk> listen to your data™

Faster Searching: NOT NOTs

Weak:

```
index=burch NOT blah=yay
```

Strong:

```
index=burch blah=duh
```

```
index=burch blah!=yay
```


splunk> listen to your data™

Search Commands: Transaction

Weak:

```
…| transaction host
```

Mo data, Mo problems!

Strong:

```
…| transaction maxspan=10m maxevents=100 …
```


splunk> listen to your data™

Search Commands: Time and Units

Weak:

```
…| eval new_time = <ridiculous string edits>
```

Strong:

```
…| convert ctime(*ime)
```

```
…| bin span=1h _time
```

```
…| eval pause = tostring( pause , "duration" )
```


splunk> listen to your data™

Search Commands: Metadata

Weak:

```
index=*
  | stats count by host
```

Strong:

```
 | metadata index=* type=hosts
```


splunk> listen to your data™

Search Commands: Eventcount

Weak:

```
index=*
  | stats count by index
```

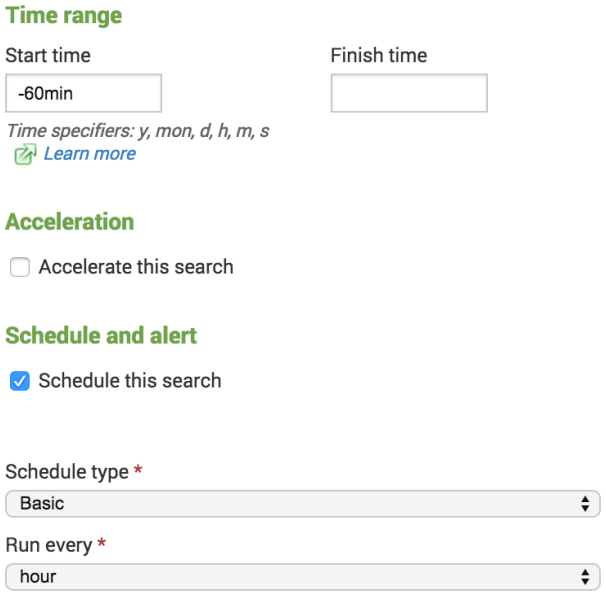
Strong:

```
 | eventcount summarize=false index=*
```

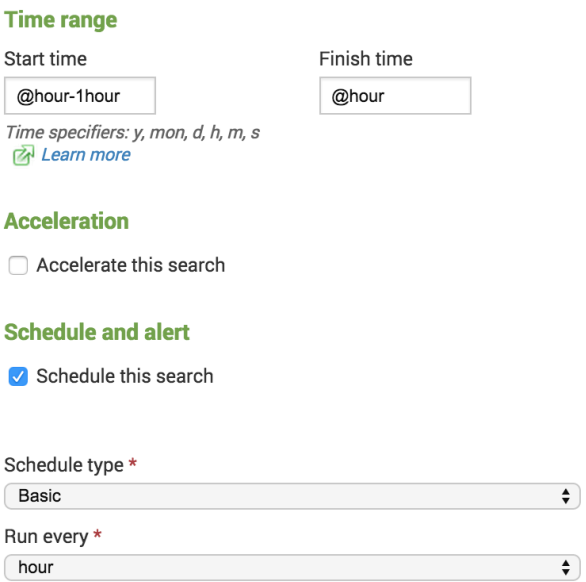

splunk> listen to your data™

Accurate Results: Snap-To Times

Weak



Strong



splunk> listen to your data™

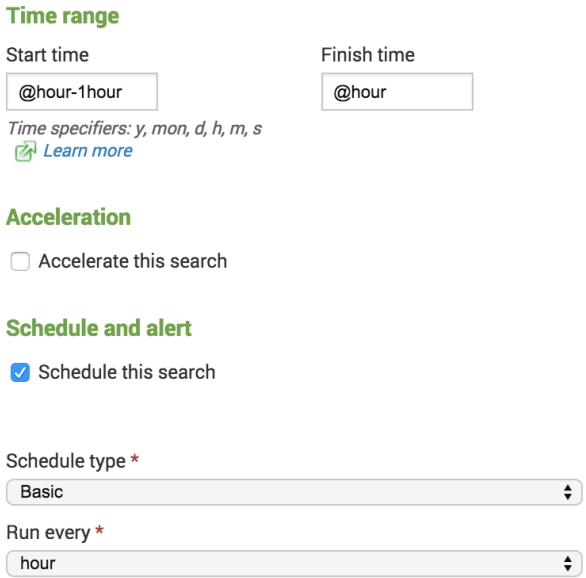
Accurate Results: Time Fields

Weak

Search

```
earliest=-24hours latest=now
...
```

Strong



splunk> listen to your data™

Accurate Results: Realistic Alerts

Weak

- Static conditions
  - | where count>10
- Spam
  - Avg

Strong

Actionable:

- stddev
- percXX

Find anomalies when outside statistical “normal”

Plug: Tom LaGatta

splunk> listen to your data™

Lookups: Best Practice

- Use gzipped CSV for large lookups
- Add automatic lookups for commonly used fields
- Scope time based lookups cleanly
- Order lookup table by ‘key’ first then values
- When building lookups, use inputlookup and stats to combine (particularly useful for ‘tracker’ type lookups)
- Splunk will index large lookups

splunk> listen to your data™

Reading Job Inspector - search.lookups

Execution costs

Duration (seconds)	Component	Invocations	Input count	Output count
0.011	command.fields	11	45,054	45,054
12.554	command.search	11	-	45,054
0.433	command.search.calcfields	28	45,054	45,054
0.145	command.search.fieldalias	28	45,054	45,054
0.031	command.search.index	33	-	-
0	command.search.index.usec_1_8	16	-	-
0	command.search.index.usec_8_64	14	-	-
6.663	command.search.lookups	28	45,054	45,054

Search.lookups =
Time to apply lookups to search

How do you optimize this?

- Use Appropriately (at end of search)
- Autolookups maybe causing issues

splunk> listen to your data™

Joins: Overview

Splunk has a join function which is often used by people with two kinds of data that they wish to analyze together. It's often less efficient than alternative approaches.

- Join involves setting up a subsearch
- Join is going to join all the data from search a and search b, usually we only need a subset
- Join often requires all data to be brought back to the search head

splunk> listen to your data™

Joins With Stats: Good Practice

- values(field_name) Is great
- range(_time) Is often a good duration
- dc(sourcetype) Is a good way of knowing if you actually joined multiple sources up or only have one part of your dataset
- eval Can be nested inside your stats expression
- searchmatch Is nice for ad-hoc grouping, could also use eventtypes if disciplined

splunk> listen to your data™

Stats & Values

```
index=_internal  sourcetype=splunkd OR sourcetype=scheduler
| stats values(user) AS user values(group) AS group values(run_time)  AS run_time by date_hour
```

Values returns all of the distinct values of the field specified

Return the values of the existing user field and call the resulting field user

Group all of the previous fields by the date_hour field

- Use |stats values(*<field name>*) -or- |stats list(*<field name>*) instead of | join
  - values(): returns distinct values in lexicographical order
  - list(): returns all values and preserves the order

splunk> listen to your data™

Joins : Example

- Before:
  - Search A | fields TxnId,Queue | join TxnId [search B or C | stats min(_time) as start_time, max(_time) as end_time by TxnId | eval total_time = end_time - start_time] | table total_time,Queue
- After
  - A OR B OR C | stats values(Queue) as Queue range(_time) as duration by TxnId
- With more exact semantics:
  - A OR B OR C | stats values(Queue) as Queue range(eval(if(searchmatch("B OR C"), _time, null()))) as duration

splunk> listen to your data™

Reading Job Inspector - search.join

Execution costs

Duration (seconds)	Component	Invocations	Input count	Output count
0.00	command.fields	3	208	208
0.01	command.join	4	208	208
0.03	command.search	6	208	416
0.00	command.search.filter	5	-	-

Search.join =
Time to apply join to search

How do you optimize this?

- Consider a dataset that is mostly error free and has a single unique identifier for related records
- Errors tie into the unique identifier
- Find the details of all errors
- Use a subsearch to first get a list of unique identifiers with errors:
- index=foo sourcetype=bar [search index=foo sourcetype=bar ERROR | top limit=0 id | fields id]

splunk> listen to your data™

Using subsearch effectively

- Consider a dataset that is mostly error free and has a single unique identifier for related records
- Errors tie into the unique identifier
- Find the details of all errors
- Use a subsearch to first get a list of unique identifiers with errors:
- index=foo sourcetype=bar [search index=foo sourcetype=bar ERROR | top limit=0 id | fields id]

splunk> listen to your data™

Reading Job Inspector - Subsearch Example

Execution costs

Duration (seconds)		Component	Invocations	Input count	Output count
	0.048	command.fields	61	140,079	140,079
	0.96	command.remotetl	61	140,079	-
	1.633	command.search	122	140,079	280,158
	0.229	command.search.filter	106	-	-
	0.087	command.search.index	110	-	-
	0.045	command.search.calcfields	45	140,079	140,079
	0.045	command.search.fieldalias	45	140,079	140,079
	0	command.search.index.usec_1_8	754	-	-
	0	command.search.index.usec_8_64	79	-	-
	0.958	command.search.rawdata	45	-	-
	0.243	command.search.tags	61	140,079	140,079
	0.052	command.search.kv	45	-	-

Search.rawdata =
Time to read actual events from rawdata files

How do you optimize this?

- Consider a dataset that is mostly error free and has a single unique identifier for related records
- Errors tie into the unique identifier
- Find the details of all errors
- Use a subsearch to first get a list of unique identifiers with errors:
- index=foo sourcetype=bar [search index=foo sourcetype=bar ERROR | top limit=0 id | fields id]

splunk> listen to your data™

Key Items To Consider In Job Inspector

Job Inspector Conclusions: Search Command Summary

Component	Description
index	**look in tsidx files for where to read in rawdata**
rawdata	**read actual events from rawdata files**
kv	**apply fields to the events**
filter	filter out events that don't match (e.g., fields, phrases)
alias	rename fields according to props.conf
lookups	create new fields based on existing field values
typer	assign eventtypes to events
tags	assign tags to events

splunk> listen to your data™

Job Inspector Conclusions: Distributed Search Summary

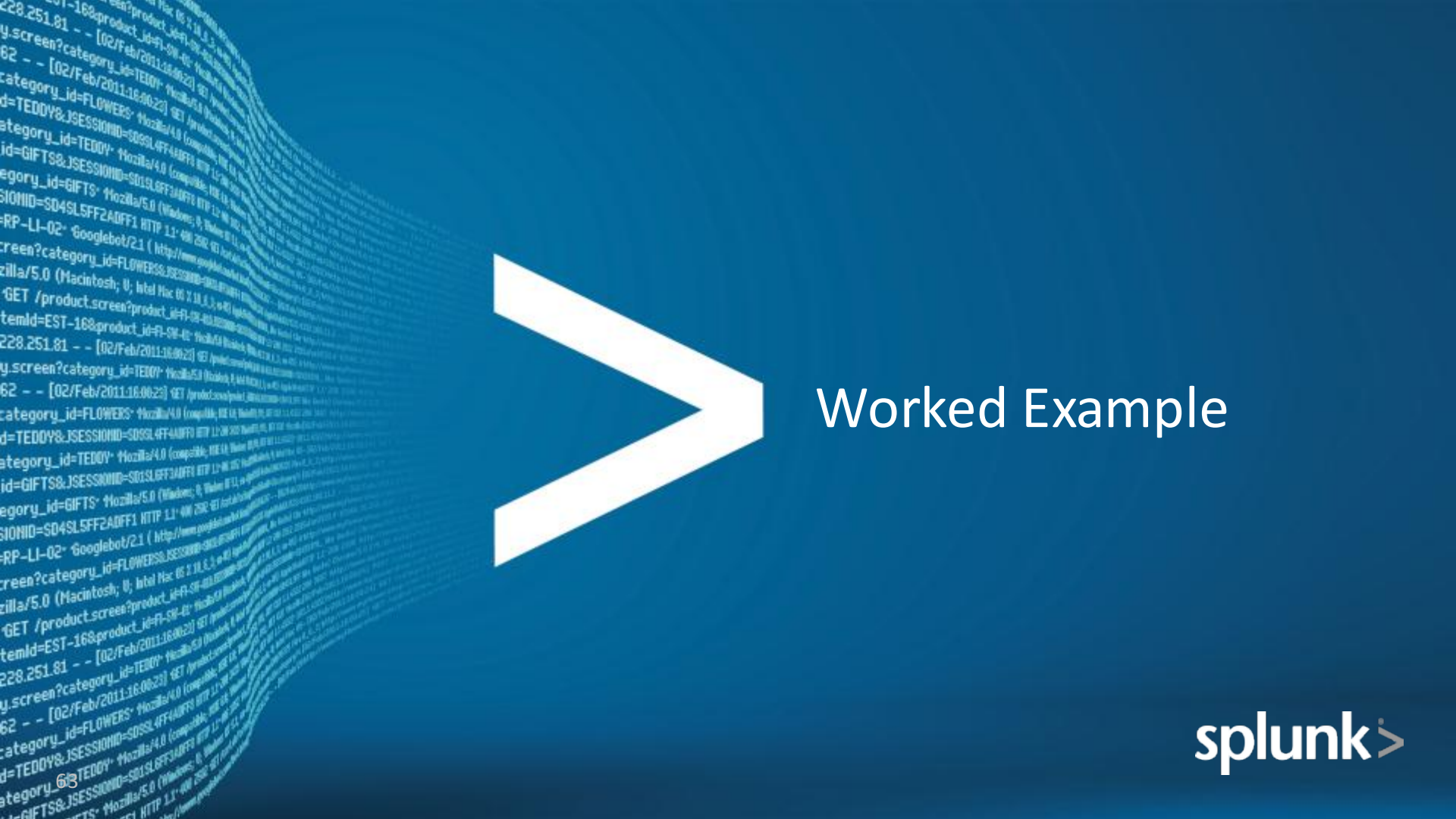
Metric	Description	Area to review
createProviderQueue	The time to connect to all search peers.	Peer conductivity
fetch	**The time spent waiting for or fetching events from search peers.**	**Faster Storage**
stream.remote	**The time spent executing the remote search in a distributed search environment, aggregated across all peers.**	
evaluate	The time spent parsing the search and setting up the data structures needed to run the search.	Possible bundle issues

splunk> listen to your data™

Job Inspector / Search.log

Field	Description	Area to review
remoteSearch	The parallelizable portion of the search	Maximize the parallelizable part.
Base lispy / keywords	**The tokens used to read data from the index and events**	**Ensure contains field tokens**
eventSearch	The part of the search for selecting data	
reportSearch	The part of the search for processing data	

splunk> listen to your data™



Worked Example

Stats vs Transaction

Search Goal: compute statistics on the duration of web session (JSESSIONID=unique identifier):

Not so Great:

```
> sourcetype=access_combined | transaction JSESSIONID | chart count by duration span=log2
```

Much Better:

```
> | stats range(_time) as duration by JSESSIONID | chart count by duration span=log2
```


splunk> listen to your data™

Use Stats To Maximal Effect

- Replace simple transaction or join usage with stats
- Stats count range(_time) dc(sourcetype) values(field) values(error) by unique_id
  - Gives you duration – range(_time)
  - Find incomplete ‘transactions’ with dc(sourcetype)
  - Find errors with values(error)
  - Find context with values(field)

splunk> listen to your data™

Use Stats To Maximal Effect

- Consider using a base stats before expensive operations like eventstats or transaction or another stats:
  - | eval orig_time = _time | bucket _time span=1h| stats count range(orig_time) as duration by unique_id _time | eventstats avg(duration) as avg | where duration>avg

splunk> listen to your data™

Reading Job Inspector - Stats Example

Execution costs

Duration (seconds)		Component	Invocations	Input count	Output count
	0.048	command.fields	61	140,079	140,079
	0.96	command.remotetl	61	140,079	-
	1.633	command.search	122	140,079	280,158
	0.229	command.search.filter	106	-	-
	0.087	command.search.index	110	-	-
	0.045	command.search.calcfields	45	140,079	140,079
	0.045	command.search.fieldalias	45	140,079	140,079
	0	command.search.index.usec_1_8	754	-	-
	0	command.search.index.usec_8_64	79	-	-
	0.958	command.search.rawdata	45	-	-
	0.243	command.search.tags	61	140,079	140,079
	0.052	command.search.kv	45	-	-

Search.rawdata =
Time to read actual events from rawdata files

How do you optimize this?

- Filtering as much as possible
- Add Peers
- Allocating more CPU, improving I/O

splunk> listen to your data™

For More Info

- http://dunca.nturnbull.com/splunk/search
- http://docs.splunk.com/Documentation/Splunk/latest/Search/Writebettersearches
- http://docs.splunk.com/Documentation/Splunk/latest/Knowledge/ViewsearchjobpropertieswiththeJobInspector

splunk> listen to your data™

Bonus: Using tstats

- When using indexed extractions, data can be queried with tstats, allowing you to produce stats directly without a prior search
- Similarly data models can be queried with tstats (speedup on accelerated data models)
- Bonus: tstats is available against host source sourcetype and _time for all data (see also the metadata/metasearch command)
- Good Practice:
  - Use tstats directly for reporting searches where available
  - Read just the columns you need
  - Multiple queries usually better than a datacube style search

splunk> listen to your data™

Key Take away: Search Best Practice

Bad Behavior	Good Behavior	Performance Improvement	Comment
All Time Searches	-24h@h	365x 30x	Limit Time Range
>*	index=xyz source=www	10-50%	Index and default fields
> foo | search bar	> foo bar	30%	Combine Searches
Verbose Mode	Fast/Smart	20-50%	Fast Mode
A NOT B	A AND C AND D AND E	5-50%	Avoid NOTS
Searches over large datasets	Data Models and Report Acceleration	1000%	Use Intelligently
Searches over long periods	Summary Indexing	1000%	Use Sparingly

splunk> listen to your data™


Thankyou!
splunk>

Typical "*my Splunk is not performing well*" conversation

A: My Splunk is slow.

B: Okay, so what exactly is slow?

A: I dunno, it just feels slow…maybe I'll just get some SSDs.

splunk> listen to your data™

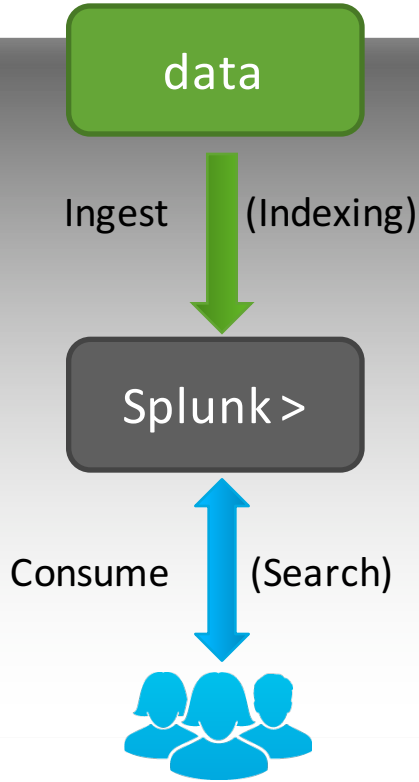
Splunk, like all distributed computing systems, has various bottlenecks that manifest themselves differently depending on workloads being processed.

- Winston Churchill

splunk> listen to your data™

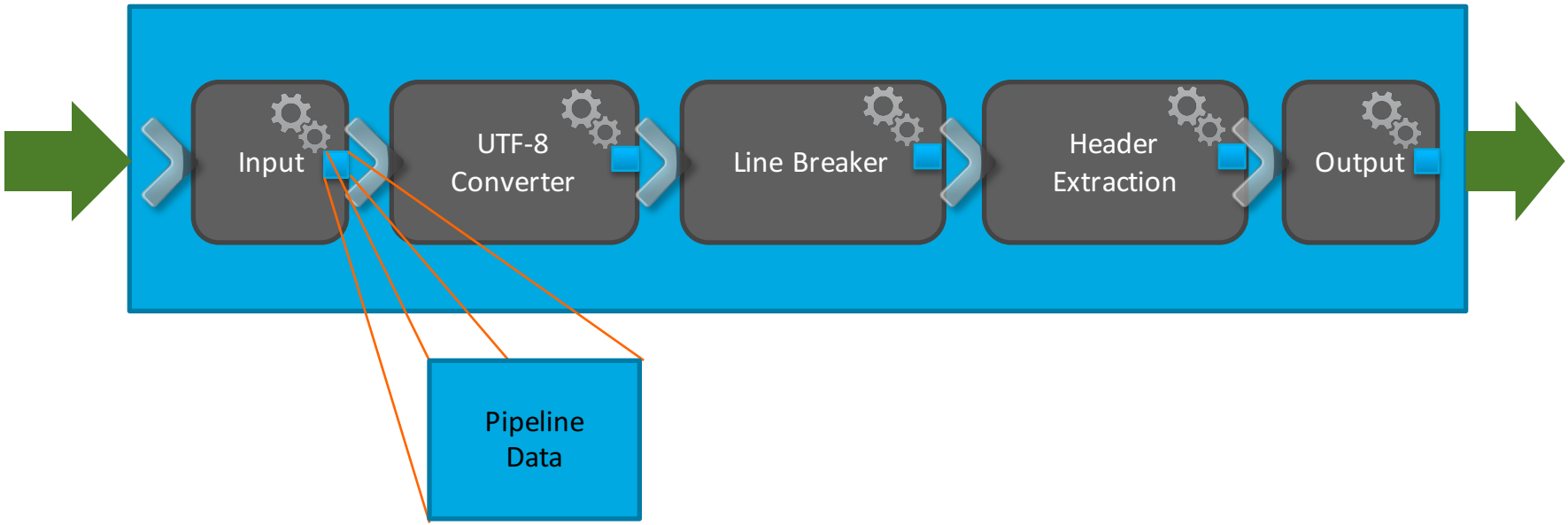
Identifying performance bottlenecks

- Understand data flows
  - Splunk operations pipelines
- Instrument
  - Capture metrics for relevant operations
- Run tests
- Draw conclusions
  - Chart and table metrics, looks for emerging patterns
- **Make recommendations**



splunk> listen to your data™

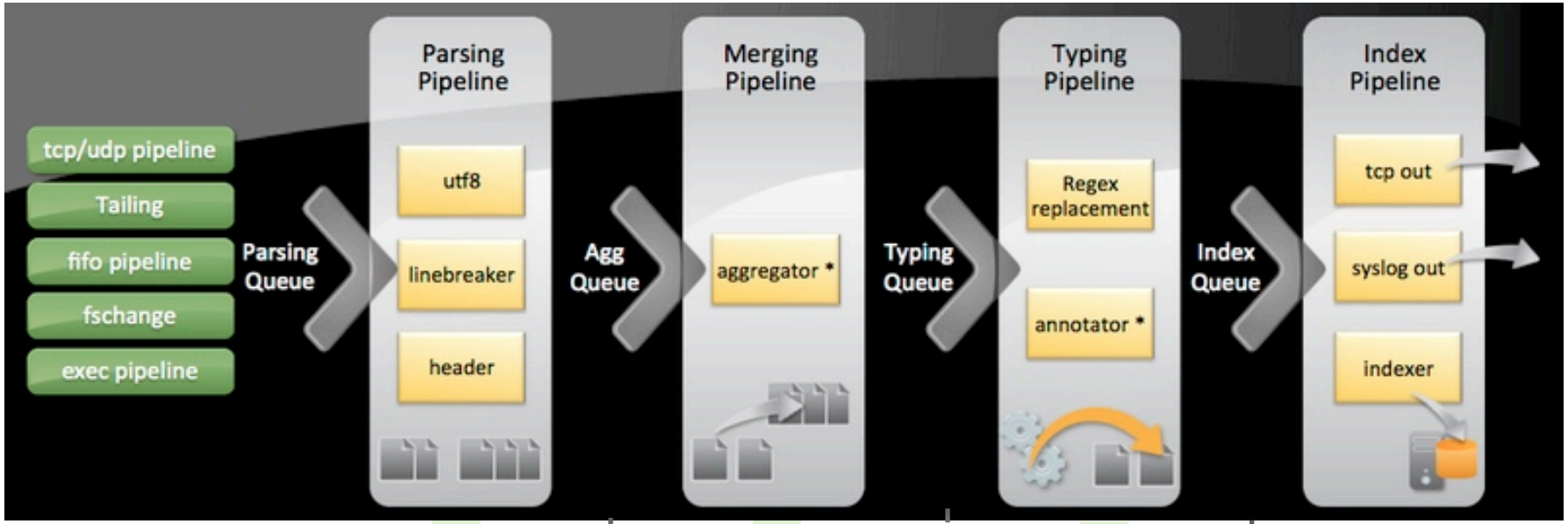
Put that in your pipeline and process it



Splunk data flows thru several such pipelines before it gets indexed

splunk> listen to your data™

A ton of pipelines



splunk> listen to your data™

Index-time processing

Event Breaking	LINE_BREAKER <where to break the stream>
	SHOULD_LINEMERGE <enable/disable merging>
Timestamp Extraction	MAX_TIMESTAMP_LOOKAHEAD <# chars in to look for ts>
	TIME_PREFIX <pattern before ts>
	TIME_FORMAT <strptime format string to extract ts>
Typing	ANNOTATE_PUNCT <enable/disable punct:: extraction>

splunk> listen to your data™

Testing: dataset A

- 10M syslog-like events:

```
. . .
Sat, 06 Apr 2014 15:55:39 PDT <syslog message >
Sat, 06 Apr 2014 15:55:40 PDT <syslog message >
Sat, 06 Apr 2014 15:55:41 PDT <syslog message >
. . .
```

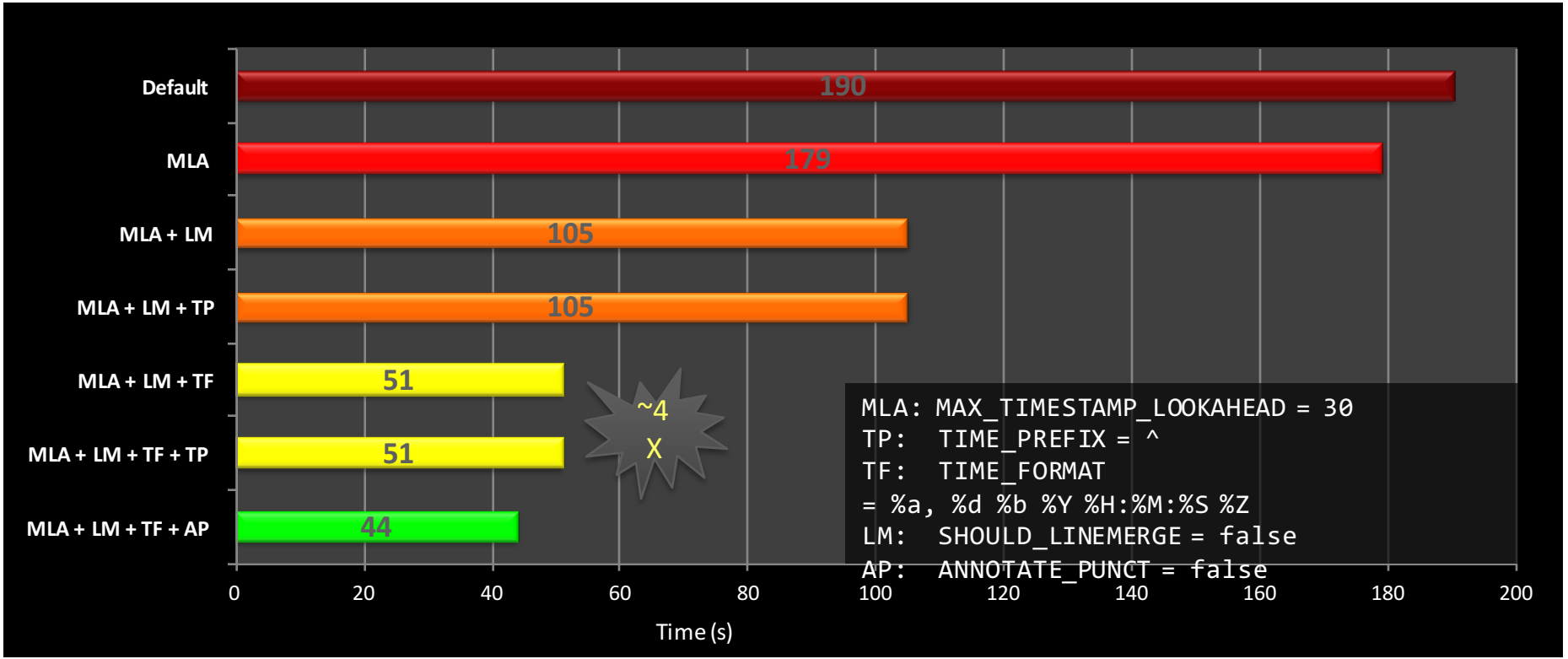
- Push data thru:
  - **Parsing > Merging > Typing** Pipelines
    - **Skip Indexing**
  - Tweak various props.conf settings

```
MLA: MAX_TIMESTAMP_LOOKAHEAD = 30
TP:  TIME_PREFIX = ^
TF:  TIME_FORMAT
= %a, %d %b %Y %H:%M:%S %Z
LM:  SHOULD_LINEMERGE = false
     ANNOTATE_PUNCT
```

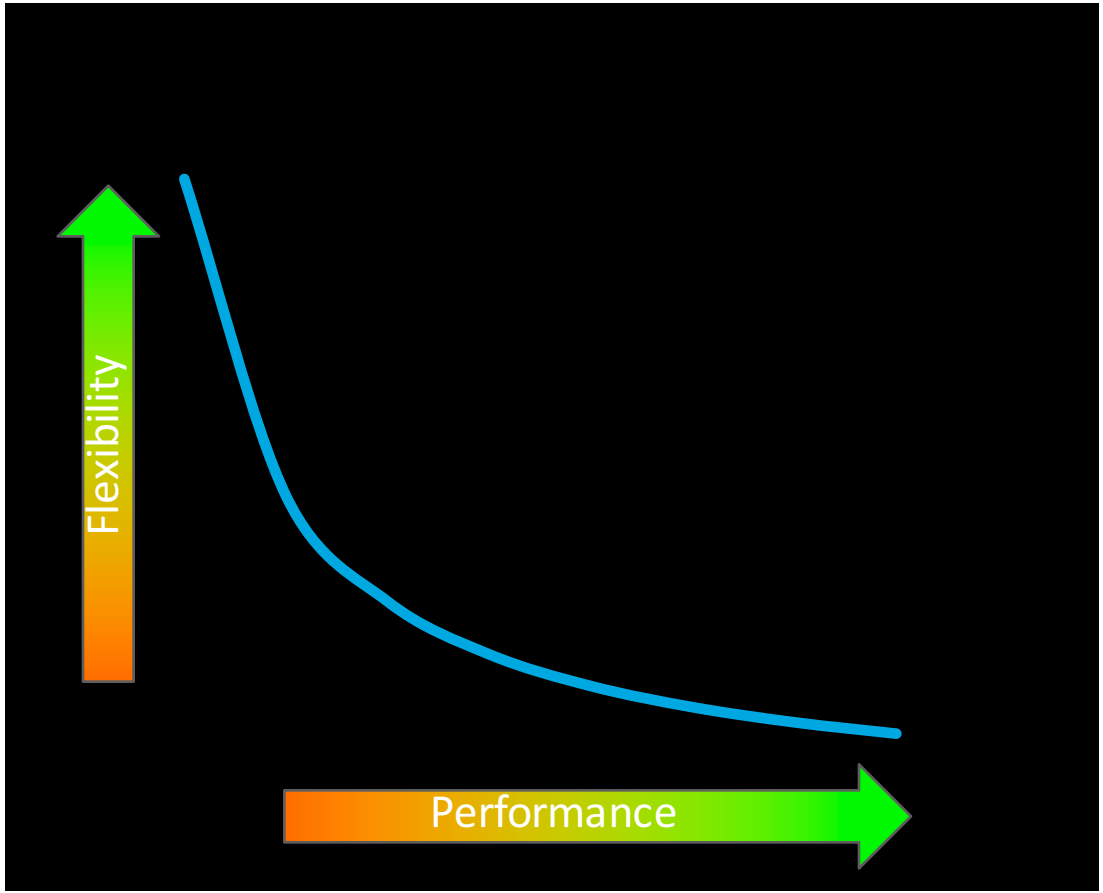
- **Measure**

splunk> listen to your data™

Index-time pipeline results



splunk> listen to your data™



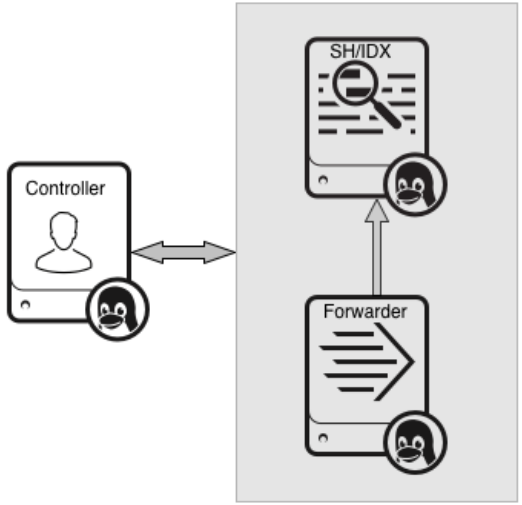
- All pre-indexing pipelines are expensive at default settings.
  - Price of flexibility
- If you're looking for performance, minimize generality

  - `LINE_BREAKER`
  - `SHOULD_LINEMERGE`
  - `MAX_TIMESTAMP_LOOKAHEAD`
  - `TIME_PREFIX`
  - `TIME_FORMAT`

splunk> listen to your data™

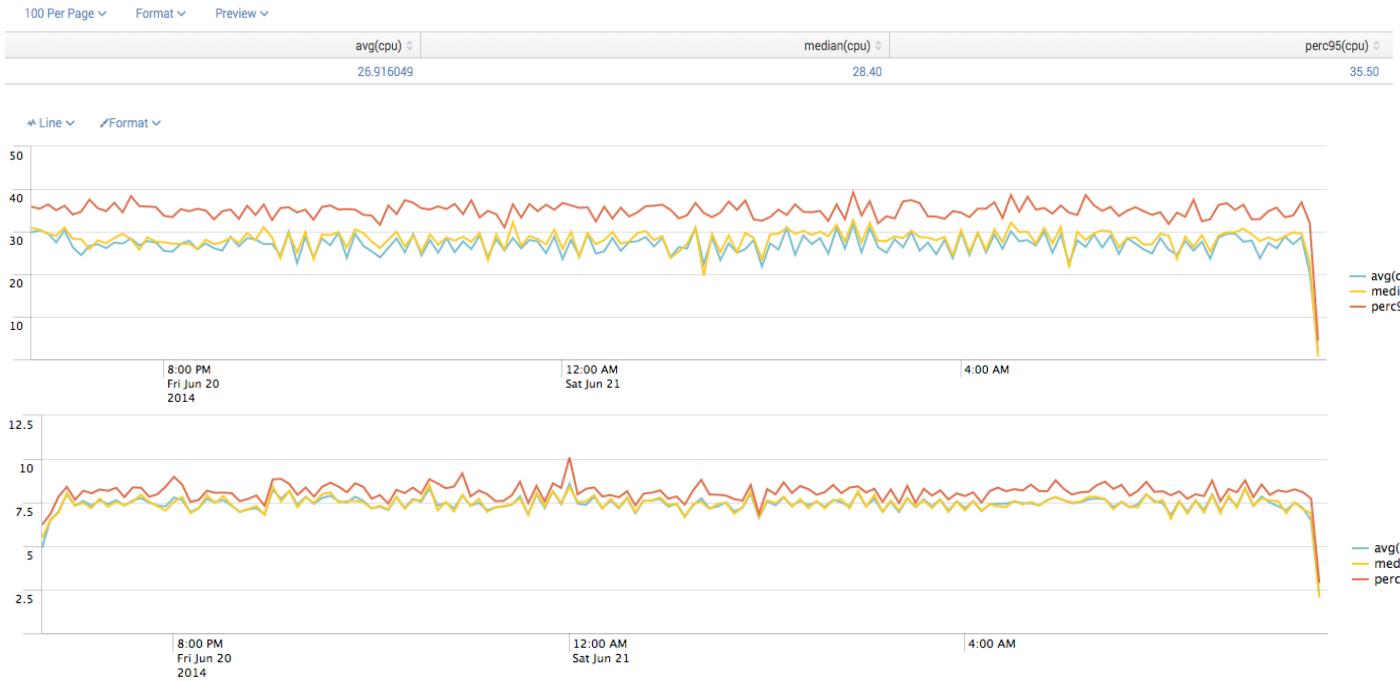
Next: let's index a dataset B

- Generate a much larger dataset (1TB)
  - High cardinality, ~380 Bytes/event, 2.86B events
- Forward to indexer as fast as possible
  - Indexer:
    - 12 CPU@2.67Ghz HT
    - 12GB RAM,
    - 14x15KRPM @146GB/ea
  - No other load on the box
- **Measure**



splunk> listen to your data™

Indexing: CPU



CPU

Load

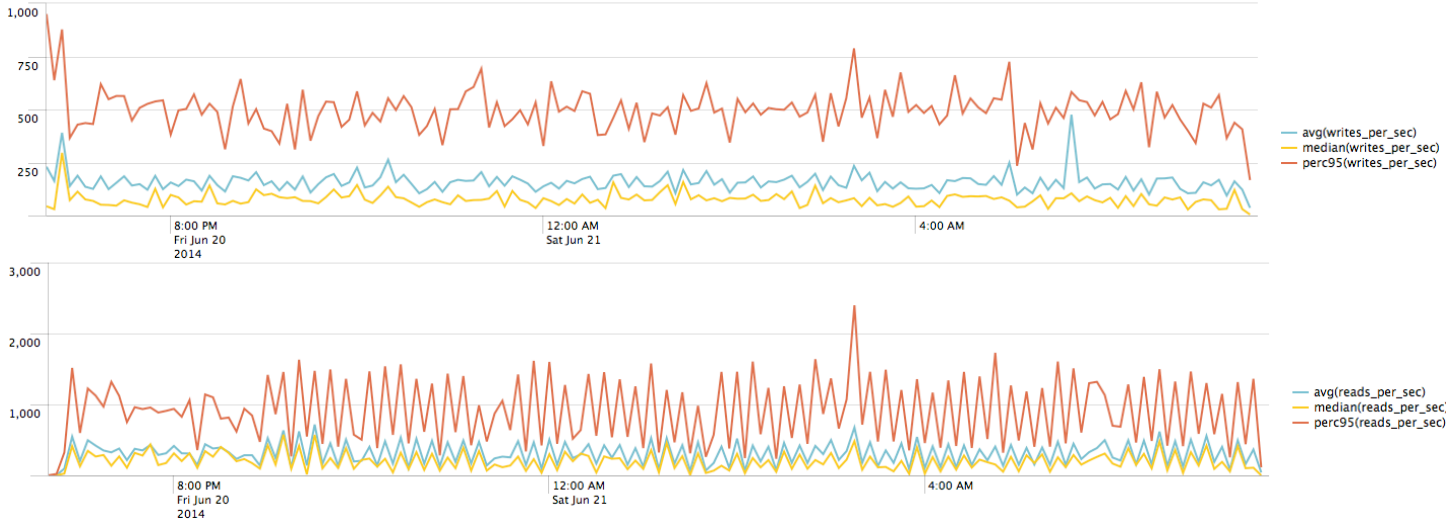
splunk> listen to your data™

Indexing: IO

100 Per Page ▾ Format ▾ Preview ▾

avg(writes_per_sec)	median(writes_per_sec)	perc95(writes_per_sec)	avg(reads_per_sec)	median(reads_per_sec)	perc95(reads_per_sec)
142.126479	57.00	510.00	277.303063	123.00	1140.00

Line ▾ Format ▾



Writes

Reads

splunk> listen to your data™

Indexing Test Findings

- CPU Utilization
  - **~35%** in this case, **4-5** Real CPU Cores
- IO Utilization
  - Characterized by both reads and writes but not as demanding as search. Note the *splunk-optimize* process.
- Ingestion Rate
  - **22MB/s**
  - "Speed of Light" – no search load present on the server

splunk> listen to your data™

Index Pipeline Parallelization

- Splunk 6.3+ can maintain multiple independent pipelines sets
  - i.e. same as if each set was running on its own indexer
- If machine is under-utilized (CPU and I/O), you can configure the indexer to run **2** such sets.
- Achieve roughly **double** the indexing throughput capacity.
- Try not to set over **2**
- Be mindful of associated resource consumption

splunk> listen to your data™

Indexing Test Conclusions

- **Distribute** as much as you can – Splunk scales horizontally
- Enable more pipelines but be aware of compute tradeoff
- **Tune** event **breaking** and **timestamping** attributes in props.conf whenever possible
- Faster disk (ex. SSDs) would not have necessarily improved indexing throughput by much
- Faster, but not more, CPUs would have improved indexing throughput – modulo pipeline parallelization

splunk> listen to your data™

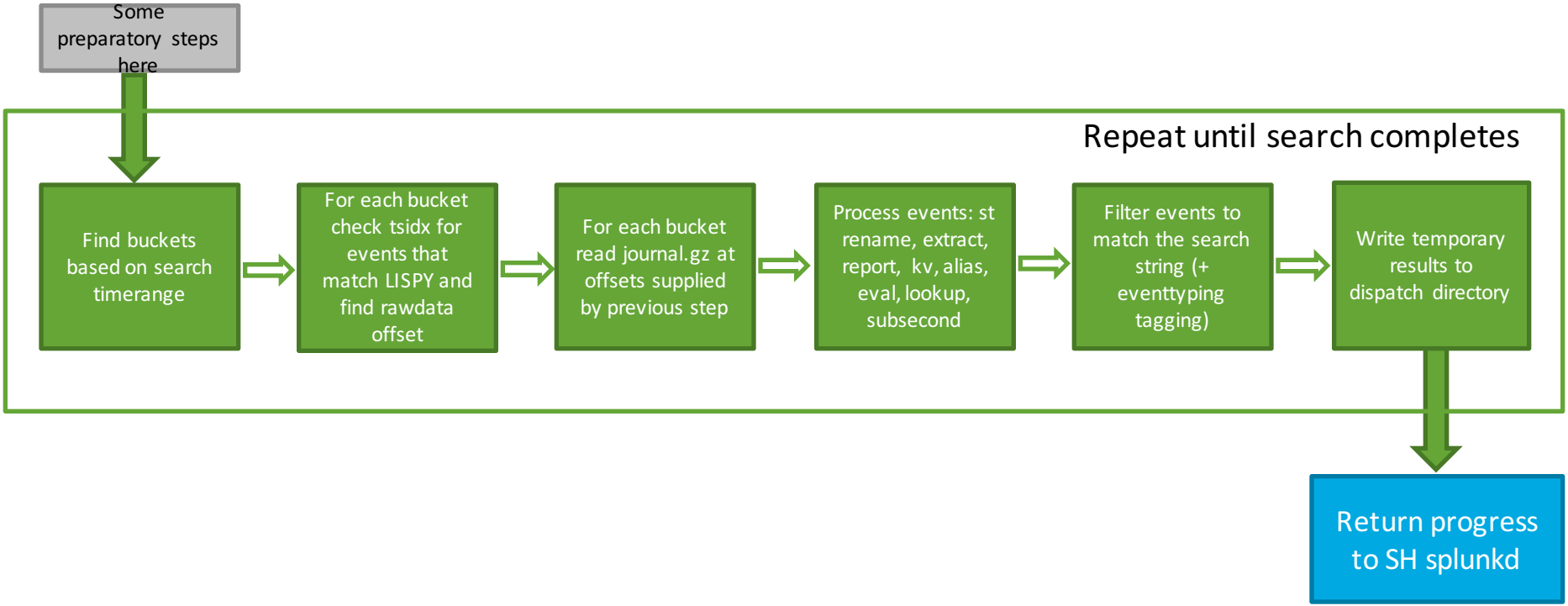
Next: Searching

- Real-life search workloads are extremely complex and very varied to be profiled correctly

- But, we can generate arbitrary workloads covering a wide spectrum of resource utilization and profile those instead. Actual profile will fall somewhere in between.

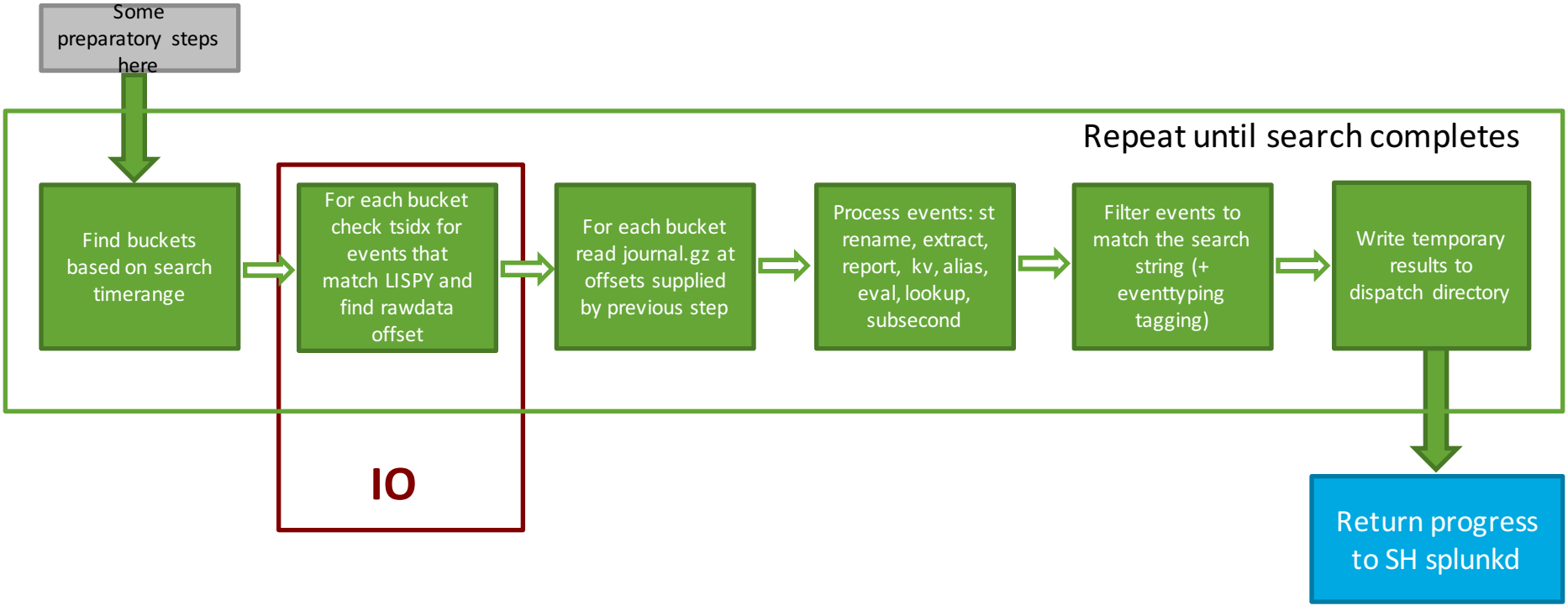
splunk> listen to your data™

Search pipeline (High Level)



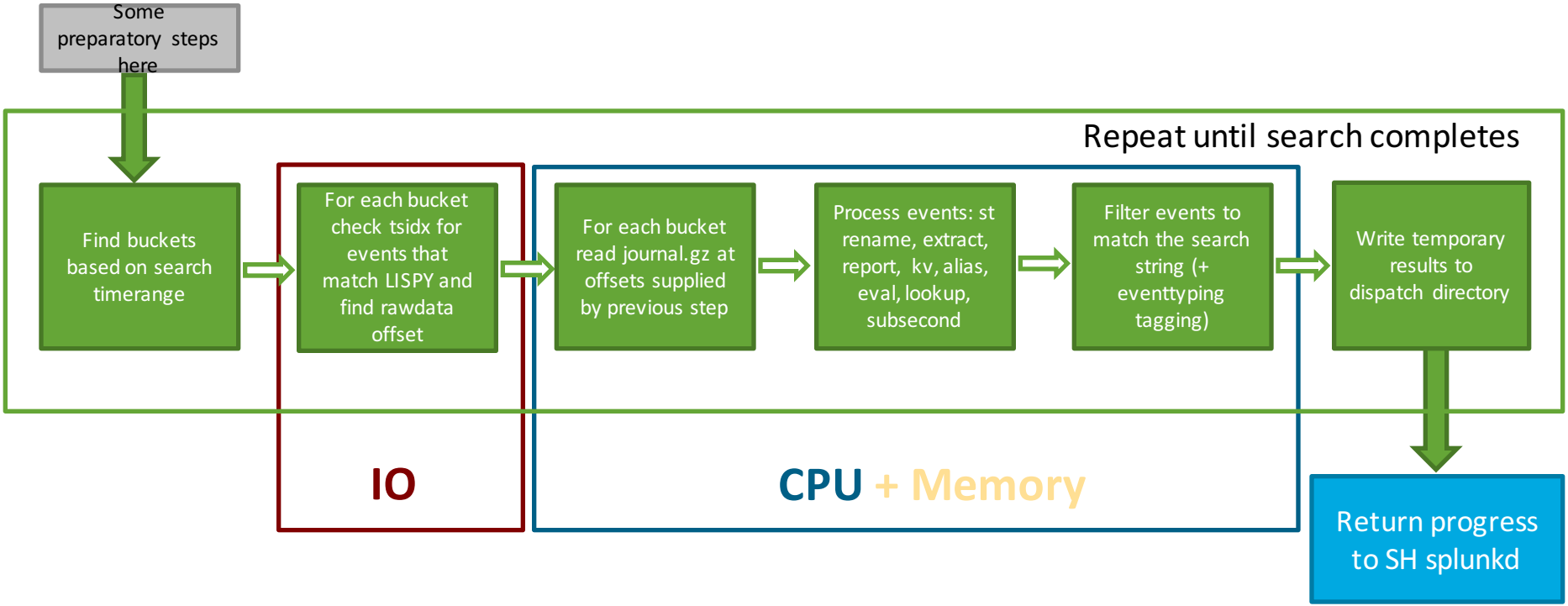
splunk> listen to your data™

Search pipeline boundedness



splunk> listen to your data™

Search pipeline boundedness



splunk> listen to your data™

Search Types

- **Dense**
  - Characterized predominantly by returning **many events** per bucket

```
index=web | stats count by clientip
```

- **Sparse**
  - Characterized predominantly by returning **some events per bucket**

```
index=web some_term | stats count by clientip
```

- **Rare**
  - Characterized predominantly by returning **only a few** events per index

```
index=web url=onedomain* | stats count by clientip
```


splunk> listen to your data™

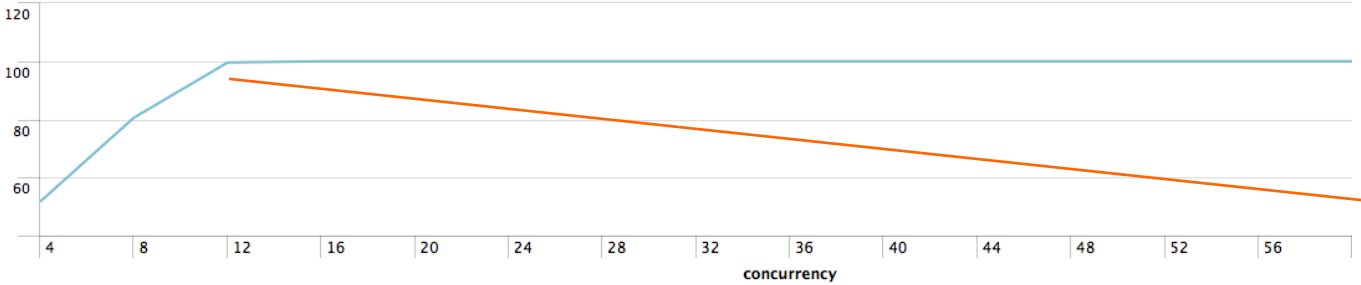
Okay, let's test some searches

- Use our already indexed data
  - It contains **many** unique terms with predictable term density
- Search under several term densities and concurrencies
  - Term density: 1/100, 1/1M, 1/100M
  - Search Concurrency: 4 – 60
  - Searches:
    - **Rare: over all 1TB dataset**
    - **Dense: over a preselected time range**
- Repeat all of the above while under an indexing workload
- **Measure**

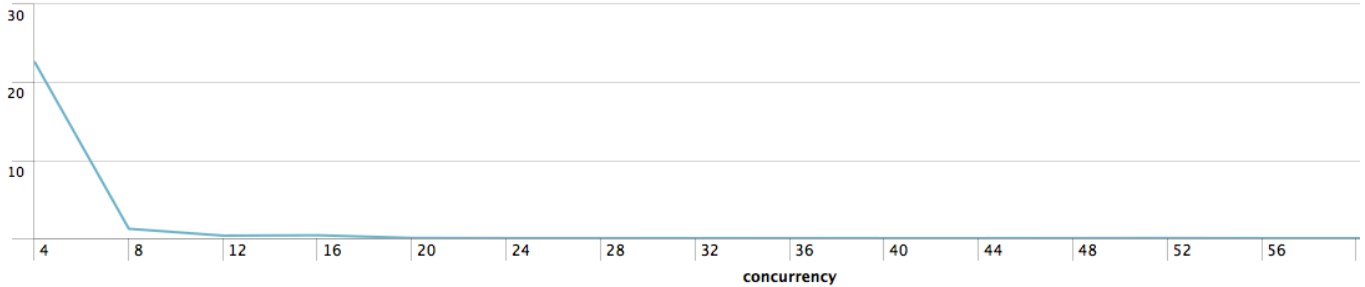
splunk> listen to your data™

Dense Searches

% CPU Util. vs. Concurrency (1/100)



% IO Wait vs. Concurrency (1/100)

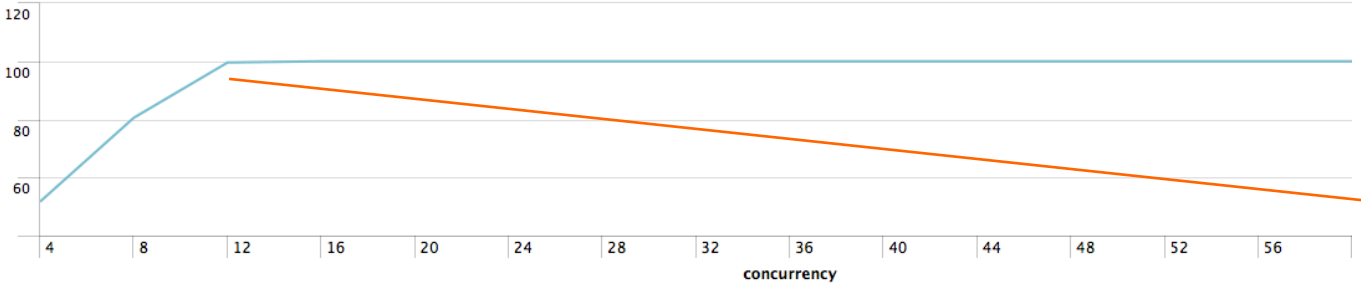


What's going on here?

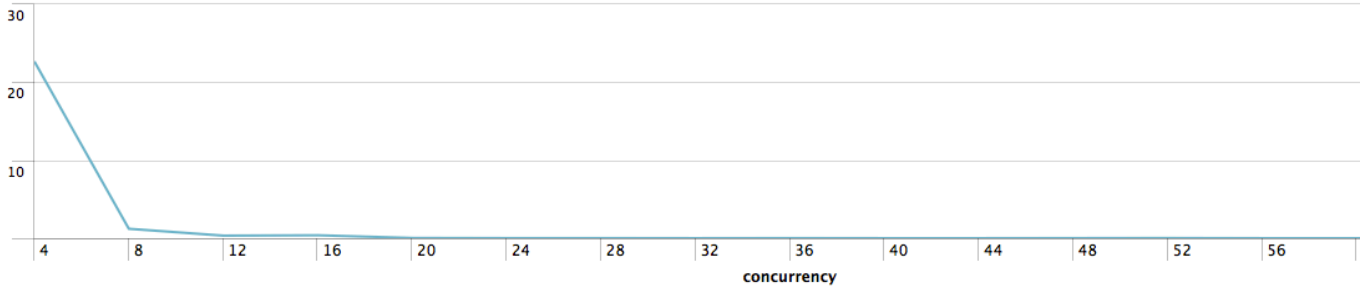
splunk> listen to your data™

Dense Searches

% CPU Util. vs. Concurrency (1/100)



% IO Wait vs. Concurrency (1/100)

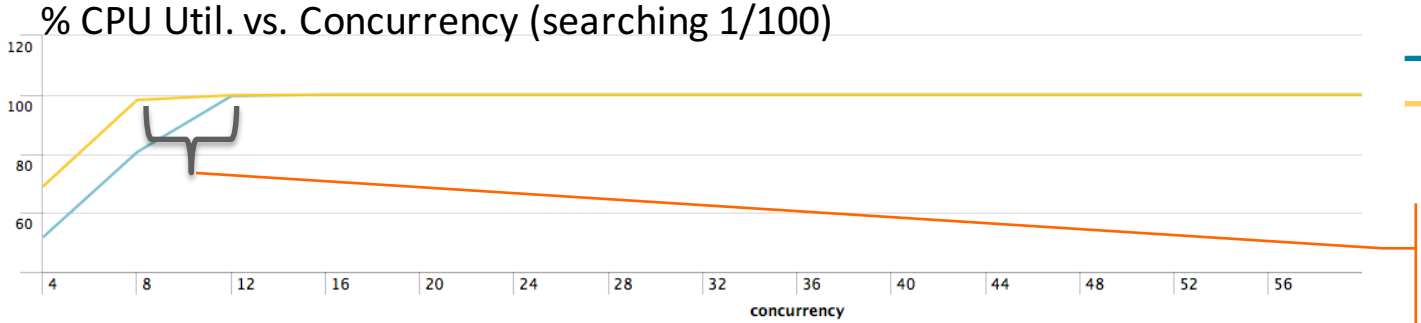


What's going on here?

CPU Bound

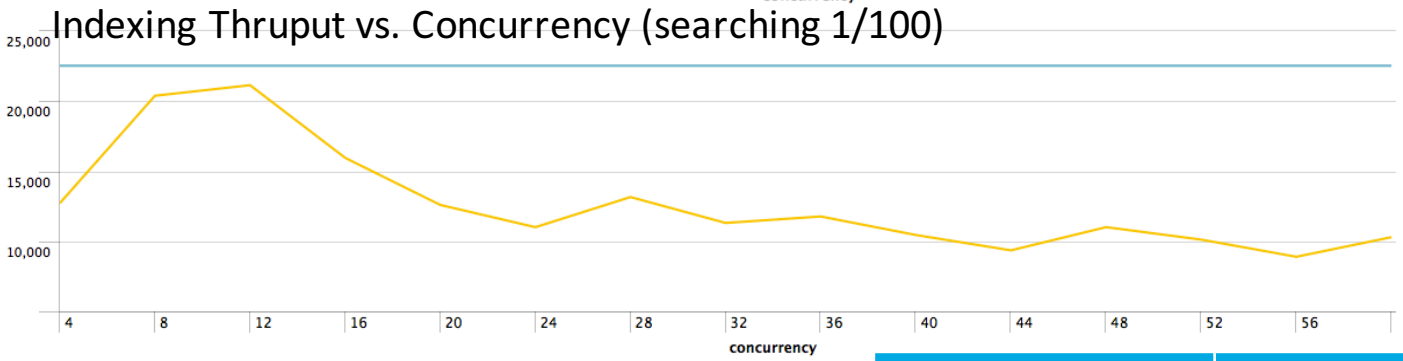
splunk> listen to your data™

Dense Searches with Indexing



Indexing off
Indexing on

Hitting 100% earlier



SpeedOfLight 1/100

Test Type	Indexing off	Indexing on	diff
Avg. Search Time	40.58s	43.85s	**+8.6%**

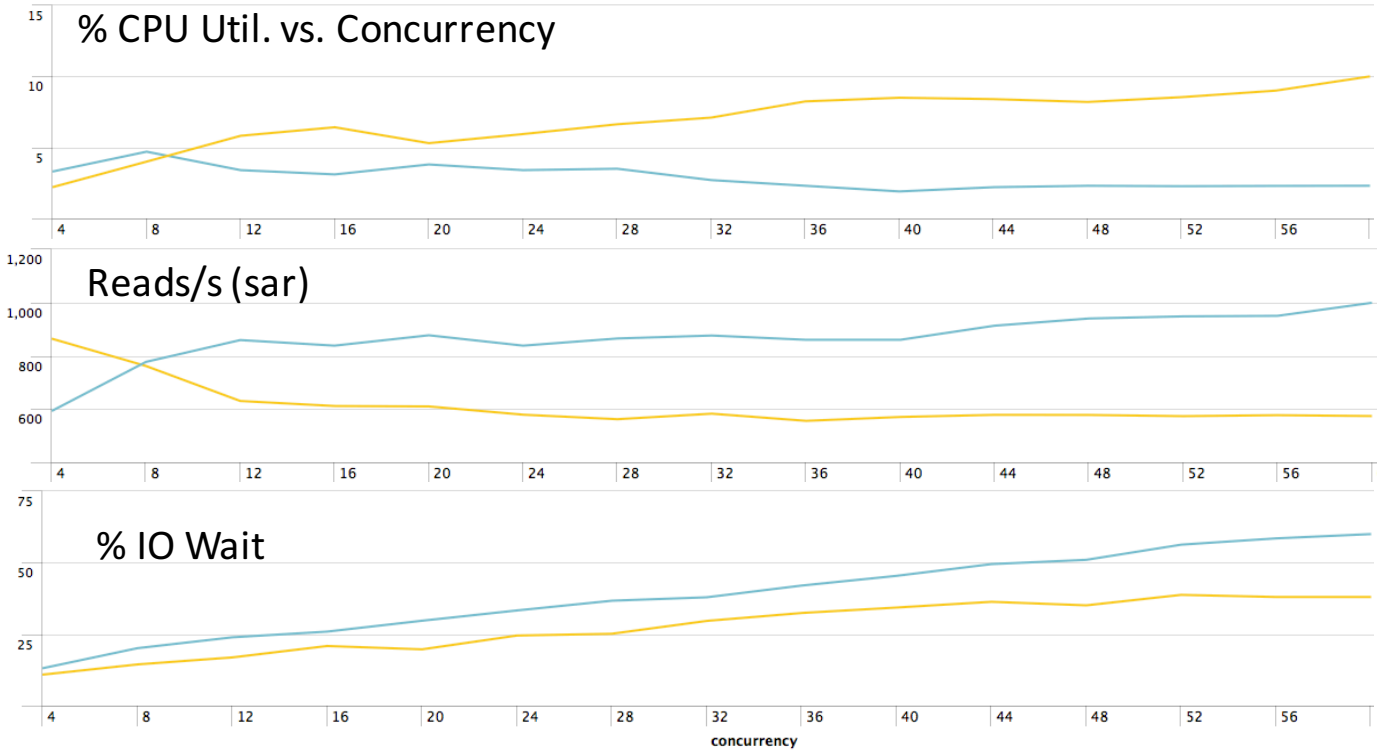
splunk> listen to your data™

Dense Search Test Conclusions

- Dense workloads are CPU bound
- Dense workloads (reporting, trending etc.) play relatively “okay” with indexing
  - While indexing throughput decreases by ~40% search time increases only marginally
- **Faster disk wont necessarily help as much here**
  - Majority of time in dense searches is spent in CPU decompressing rawdata + other SPL processing
- **Faster and more CPUs would have improved overall performance**

splunk> listen to your data™

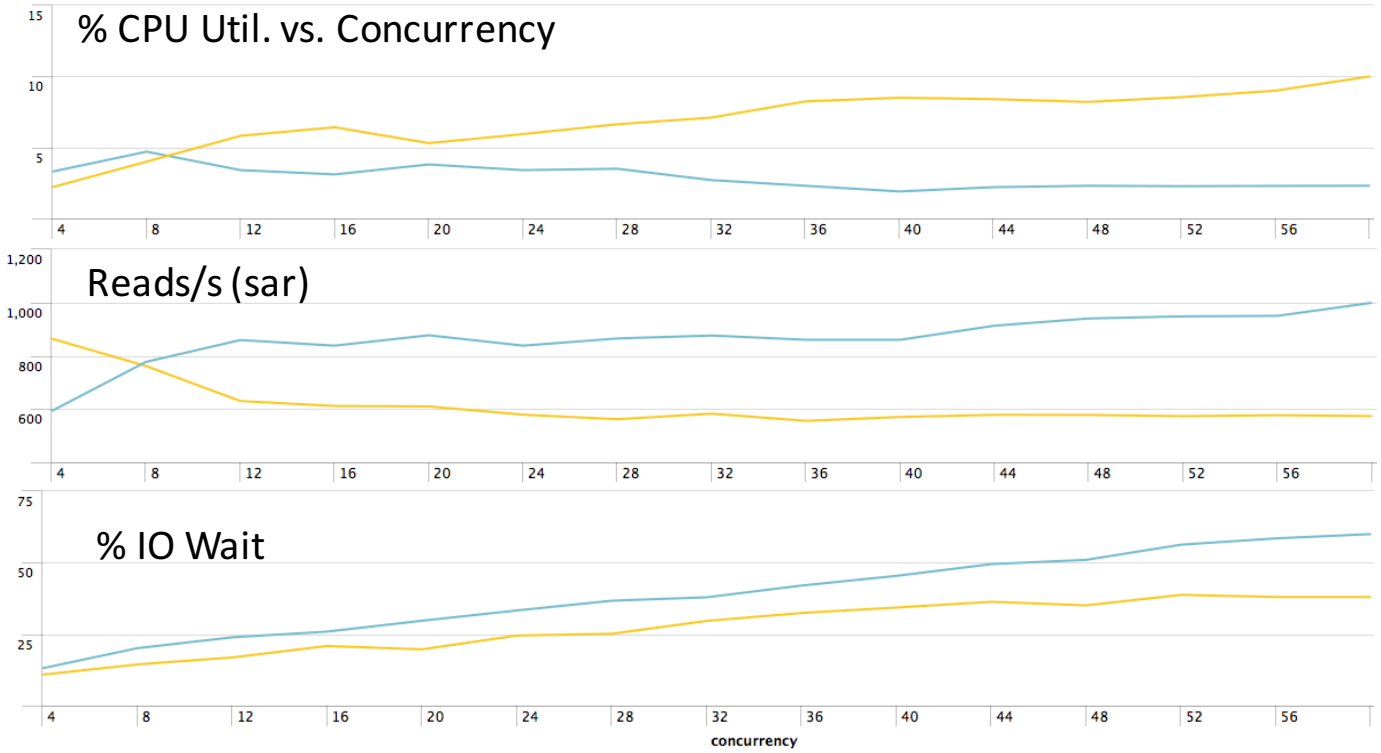
Rare Searches



1/M
1/100M

splunk> listen to your data™

Rare Searches

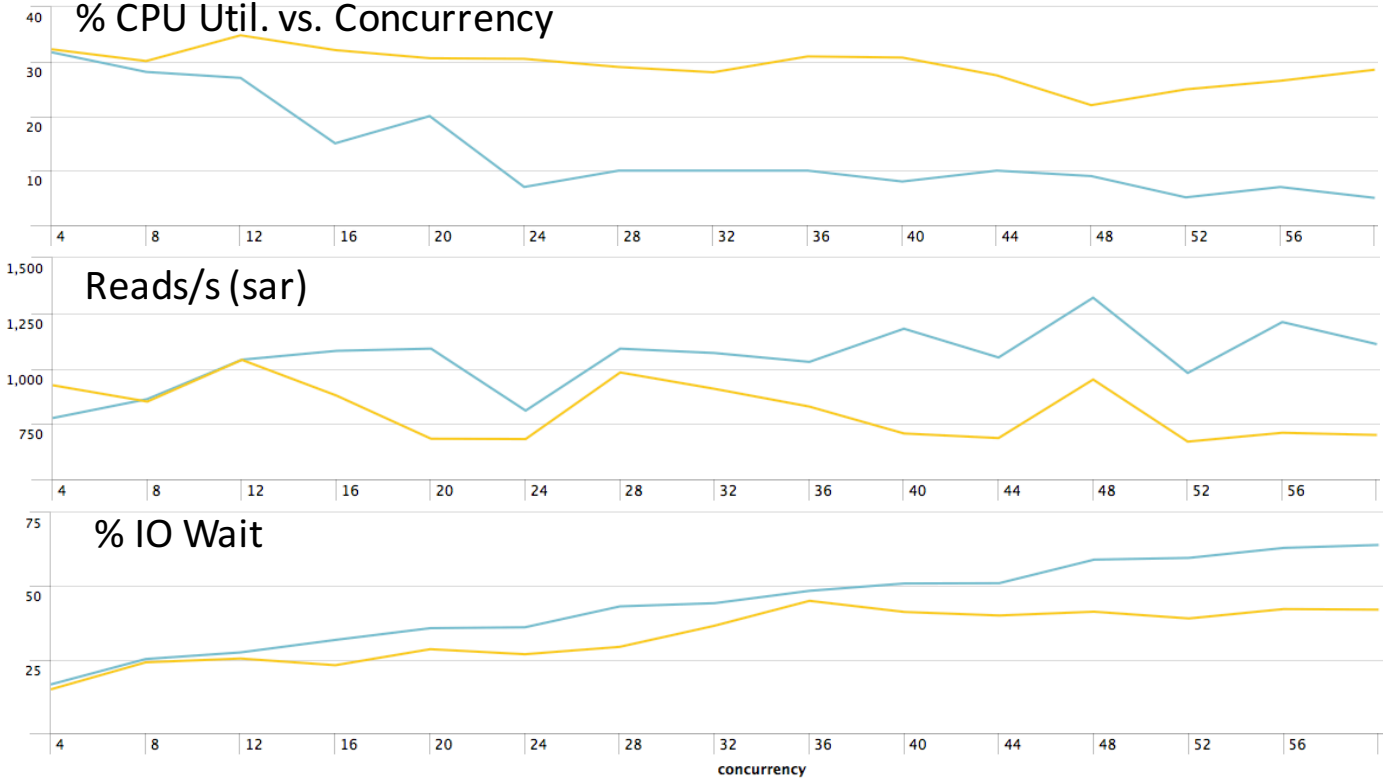


1/M
1/100M

IO Bound

splunk> listen to your data™

Rare Searches with Indexing

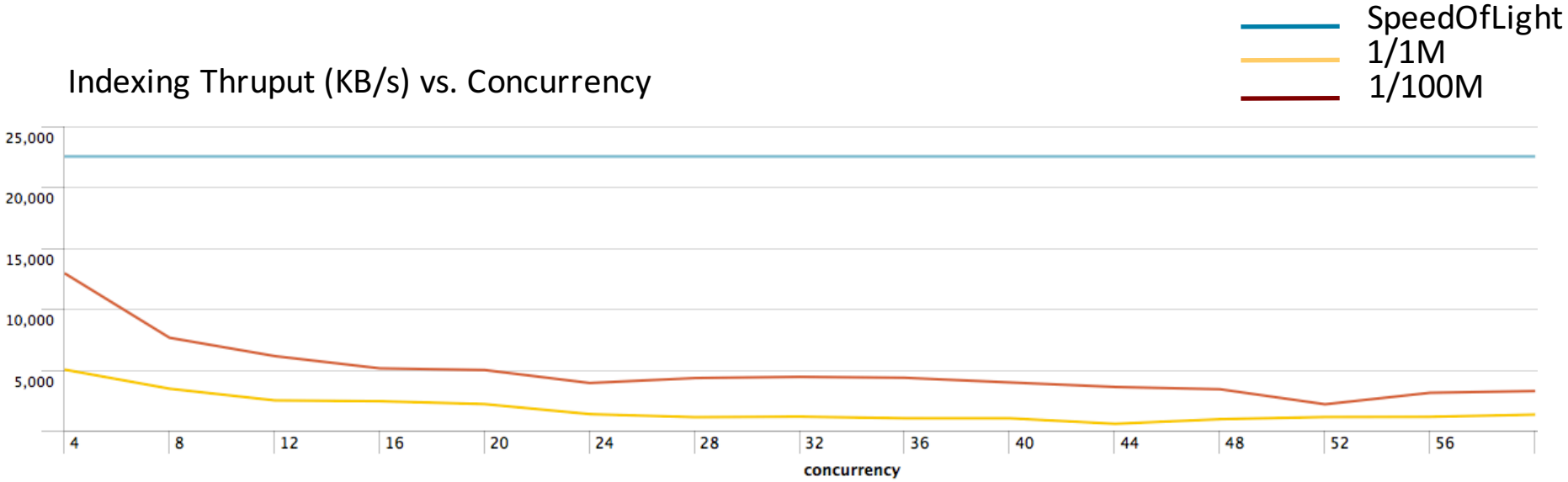


1/M
1/100M

splunk> listen to your data™

More numbers

Indexing Thruput (KB/s) vs. Concurrency



Density	1/1M			1/100M		
Test Type	**Indexing off**	**Indexing on**	**diff**	**Indexing off**	**Indexing on**	**diff**
Avg Search Time	1041s	1806s	**+73.5%**	231s	304s	**+31.6%**

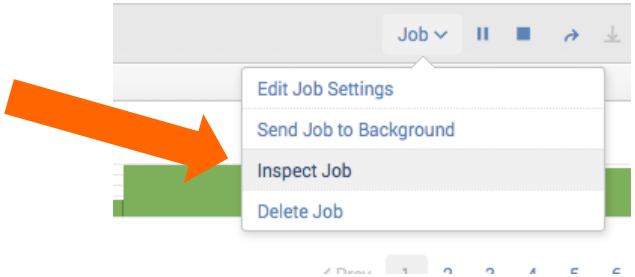
splunk> listen to your data™

Rare Search Conclusions

- Rare workloads (investigative, ad-hoc) are IO bound
- Rare workloads (high IO wait) do not play well with indexing
  - Search time increases significantly when indexing is on. Also, indexing throughput takes an equal hit on the opposite direction.
- 1/100M searches have a lesser impact on IO than 1/1M.
- When indexing is on, in 1/1M case search time increases more substantially vs. 1/100M. Search and indexing are both contenting for IO.
- In case of 1/100M, **bloomfilters** save precious IO which, in turn, allows for a better indexing throughput.
  - **Bloomfilters** are special data structures that indicate with 100% certainty that a term **does not exist** in a bucket (effectively telling the search process to skip that bucket).
  - Note the higher CPU consumption during 1/100M searches with indexing on
- **Faster disks would have definitely helped here**
- **More CPUs would not have improved performance by much**

splunk> listen to your data™

Is my search CPU or IO bound?



Guideline in absence of full instrumentation

- **command.search.rawdata** ~ CPU Bound
  - Others: .kv, .typer, .calcfields,
- **command.search.index** ~ IO Bound

Search job inspector

This search has completed and has returned **1** result by scanning **4,159,473** events in **20.706** seconds.

The following messages were returned by the search subsystem:

DEBUG: **Disabling timeline and fields picker for reporting search due to adhoc_search_level=smart**
DEBUG: **base lispy: [AND index::_internal]**
DEBUG: **search context: user="admin", app="aws_app", bs-pathname="/opt/splunk61/etc"**

(SID: 1410010633.156)

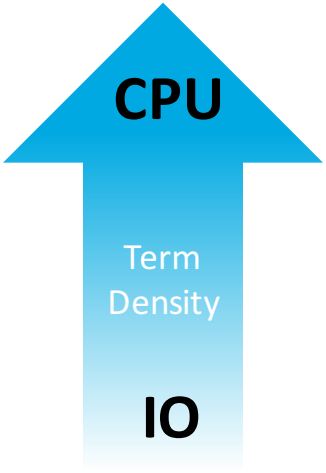
Execution costs

Duration (seconds)	Component	Invocations	Input count	Output count
0.344	command.addinfo	344	4,159,473	4,159,473
0.343	command.fields	344	4,159,473	4,159,473
7.133	command.prestats	344	4,159,473	343
13.247	command.search	344	-	4,159,473
10.254	command.search.rawdata			
0.363	command.search.kv	343	-	-
0.344	command.search.tags	344	4,159,473	4,159,473
0.344	command.search.typer	344	4,159,473	4,159,473
0.343	command.search.calcfields	343	4,159,473	4,159,473
0.343	command.search.fieldalias	343	4,159,473	4,159,473
0.343	command.search.lookups	343	4,159,473	4,159,473
0.11	command.search.summary	344	-	-
0	command.search.index.usec_1_8	22	-	-
0	command.search.index.usec_512_4096	84	-	-
0	command.search.index.usec_64_512	314	-	-
0	command.search.index.usec_8_64	116	-	-
0.345	command.stats.execute_input	345	-	-

splunk> listen to your data™

Top Takeways/Re-Cap

- **Indexing**
  - **Distribute** – Splunk scales horizontally
  - **Tune** event breaking and timestamp extraction
  - **Faster** CPUs will help with indexing performance
- **Searching**
  - **Distribute – Splunk scales horizontally**
  - **Dense Search Workloads**
    - CPU Bound, better with indexing than rare workloads
    - Faster and more CPUs will help
  - **Rare Search Workloads**
    - IO Bound, not that great with indexing
    - Bloomfilters help significantly
    - Faster disks will help
- **Performance**
  - Avoid generality, optimize for expected case and add hardware whenever you can



Use case	What Helps?
Trending, reporting over long term etc.	More distribution Faster, more CPUs
Ad-hoc analysis, investigative type	More distribution Faster Disks, SSDs

splunk> listen to your data™