Forward-Looking Statements

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

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

In addition, any information about our roadmap outlines our general product direction and is subject to change at any time without notice. It is for informational purposes only and shall not be incorporated into any contract or other commitment. We undertake no obligation either to develop the features or functionalities described, in beta or in preview (used interchangeably), or to include any such feature or functionality in a future release.

splunk> .conf22

Master Joining Datasets Without Using Join

PLA1528B

Nick Mealy

Chief Mad Scientist | Sideview, LLC

Nick Mealy

CEO, Chief Mad Scientist | Sideview, LLC

splunk> .conf22

Why are we here?

You're in charge of some things! Searches that use join, append or appendcols.

You're hitting limits with them, and/or maybe seeing bad search performance.

You've heard that they are bad but…you feel like you're stuck with them.

You are hoping! That by rewriting these searches somehow you could greatly reduce resource load on your search heads (and make the searches faster, too).

You've seen this talk in prior years, but you're hoping it has something new. (It does!)

Why is this guy qualified to give this talk?

Former Splunk Mad Scientist and Principal UI Developer 2005-2010.

This is Rob Das and I in the booth at Linuxworld in 2005, the week we all launched Splunk 1.0.

Why is this guy qualified to give this talk?

Whenever there was any new search and reporting functionality in Splunk, the UI team was the first thrown into the pit.

The first people to hit the bottom learned how to welcome everyone else.

Why is this guy qualified to give this talk?

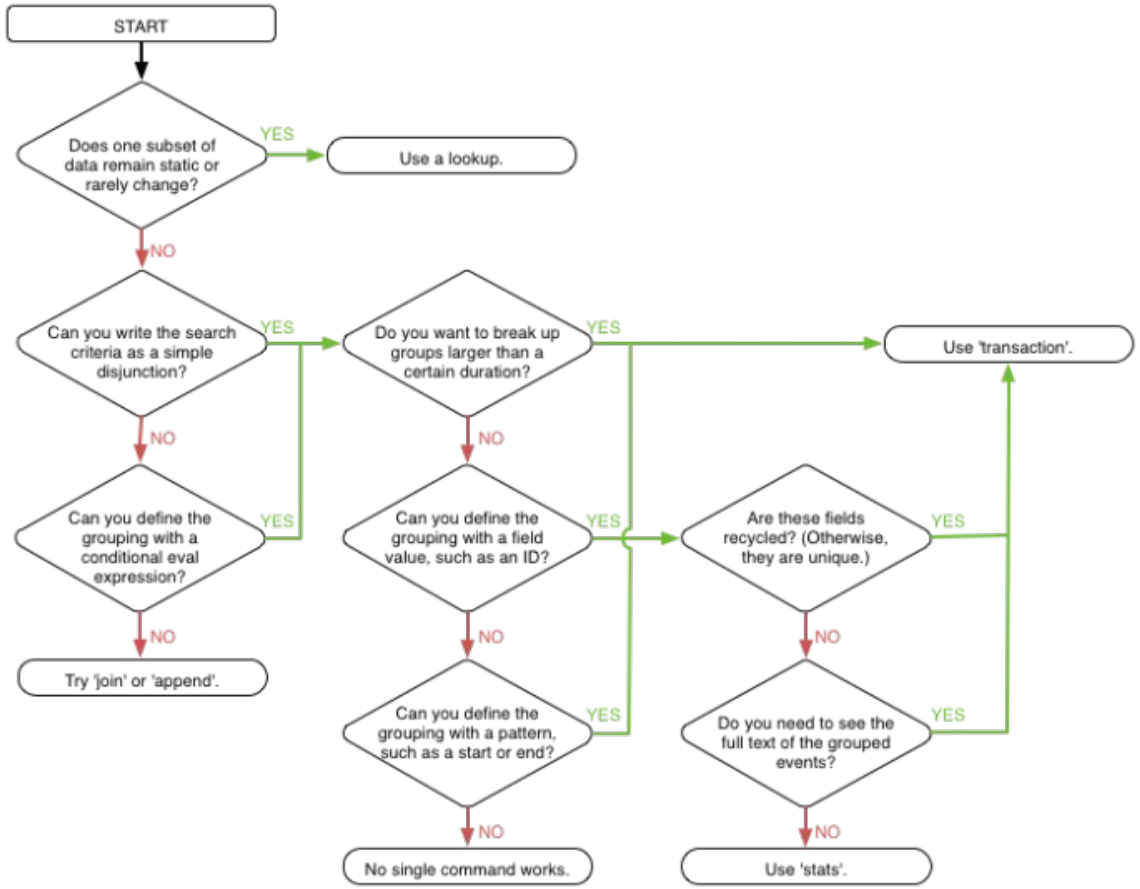
This is me gearing up when the whole company went spelunking together.

Only two people vomited that I know of.

Okay, so… I heard 2010. How about the last 12 years?

I was the karma leader on Answers once upon a time, and it was in large part from answering search language questions in this particular area.

Also the “grouping” docs still use a flowchart I made ages ago.



splunk> .conf22

Okay, so… 2016? Keep going…

I never stopped working with Splunk fulltime in a technical role and our main product for Cisco CallManager has to do some pretty hairy SPL™ under the hood of its' UI.

```
`custom_index` (sourcetype=cucm_cdr OR sourcetype=cucm_cmr OR sourcetype=cube_cdr)
  [search `cdr_events` callingPartyNumber::+1415* OR originalCalledPartyNumber::+1415* OR
     finalCalledPartyNumber::+1415* | streamstats dc(globalCallID_callId) as distinct_ids | head
     distinct_ids<1000 | stats count by globalCallID_callId globalCallID_callManagerId
     globalCallId_ClusterID | fields - count | append [| makeresults | fields - _time | eval sourcetype
     ="cube_cdr"]]
| `get_quality` | eval day=strftime(_time,"%m_%d_%Y") | eventstats values(clid) as clid by
   call_reference_id day | fields - day | `sort_call_legs` | eval durationStr=tostring(duration,"duration"
   )
| stats list(MLQK) as MLQK min(_time) as _time list(callingPartyNumber) as callingPartyNumber list
   (cause_description) as cause_description list(clid) as clid list(duration) as duration list(durationStr
   ) as durationStr list(eventtype) as eventtype list(finalCalledPartyNumber) as finalCalledPartyNumber
   list(jitter) as jitter list(latency) as latency list(numberPacketsLost) as numberPacketsLost list
   (numberPacketsSent) as numberPacketsSent list(originalCalledPartyNumber) as originalCalledPartyNumber
   list(quality) as quality list(sourcetype) as sourcetype list(type) as type by globalCallID_callId
   globalCallID_callManagerId globalCallId_ClusterID
| eval duration_gt_10=if(max(duration)>10,1,0) | search sourcetype="cucm_cdr" clid=* duration_gt_10=1 (
   eventtype="incoming_call" ) callingPartyNumber::+1415* OR originalCalledPartyNumber::+1415* OR
   finalCalledPartyNumber::+1415* | fields - duration_gt_10 | rename durationStr as duration
| sort 0 - _time | fields MLQK _time callingPartyNumber cause_description clid duration eventtype
   finalCalledPartyNumber jitter latency numberPacketsLost numberPacketsSent originalCalledPartyNumber
   quality sourcetype type  `id_fields`
```

The Talk

assuming we ever get to it

… what's it like?

- **Why are the bad things bad?**
  Why join and append are evil.
  (Why transaction is chaotic neutral.)

- **How to be good. How to 'use stats' even when it really looks like you can't.**
  You may get a tendency to say "I wonder if we can use some conditional eval to fix this."

- **Do I have to? Are my searches really bad?**
  I.e., is the SH stuck doing most of the work?
  Also, whose arms do I need to rip off.

The Talk

The TL;DR version

- Join and append really don't scale.
  They're designed to be last resorts for weird cases.
  They underuse the IDX tier and overload the SH.
  Raising limits.conf keys only helps a little.

- You use lots of OR and lots of eval in your SPL™.
  Sourcetype=A OR sourcetype=B
  You send a terrible ugly heap of different events down the pipeline and let eval/stats and friends sort them out for you. It's crazy unintuitive.

- Sometimes you don't have to.
  The Job Inspector is your friend.
  There are ways… of finding offenders…

Part 1
Why are the bad things bad?

splunk> .conf22

Naming things – it's hard.

The names are really unfortunate and send users off the wrong way.

What would SQL do? you will search the splunk docs for "join".
docs are using this word "transaction". "got it. there's a transaction command".
I need to like… tack on another column. "appendcols" ftw!!

vs

Stats? -- "Statistics? Nah I don't need that right now, I need to group things."

NO. Stats eval and lookups should be your first tools.
Append/appendcols/join and even transaction should be last resorts.

What's wrong with the join and append commands?

- Fundamentally slow, and as soon as you push any real volume of data through them they break.
- truncation if you exceed 50,000 rows (and/or oom anxiety).
- Your job is quietly autofinalized (another kind of truncation) if you exceed execution time.
- 2 jobs instead of 1.
- Potentially getting the same data off disk twice.
- You might not even *realize* that you're hitting autofinalize and row-truncation limits, but they can nonetheless make your results wrong.
- **Breaking Map/Reduce, OR making it less efficient. I.e. forcing Splunk to pull more data and processing back to the search head.**
- **As a kind of "worst-practice", it proliferates quickly.**

Multisearch fixes this, doesn't it?

Not… really?

Kind of, in "high cardinality" scenarios where map/reduce wasn't doing much.

However in low cardinality the lack of distributed reporting is very bad.
E.g: Here, it's about 4 times slower than the equivalent "right way".

```
| multisearch
    [search index=cisco_cdr type=incoming]
    [search index=cisco_cdr type=outgoing]
    [search index=cisco_cdr type=internal]
    [search index=cisco_cdr type=tandem]
| stats count by type
```

Search head says it's tired – you do it.

```
Sourcetype=cdr type=outgoing | stats sum(duration) by device_type
```

YOU ARE NOW THE SEARCH HEAD!

You don't know the best way to do this.

But… what's the **worst** way?

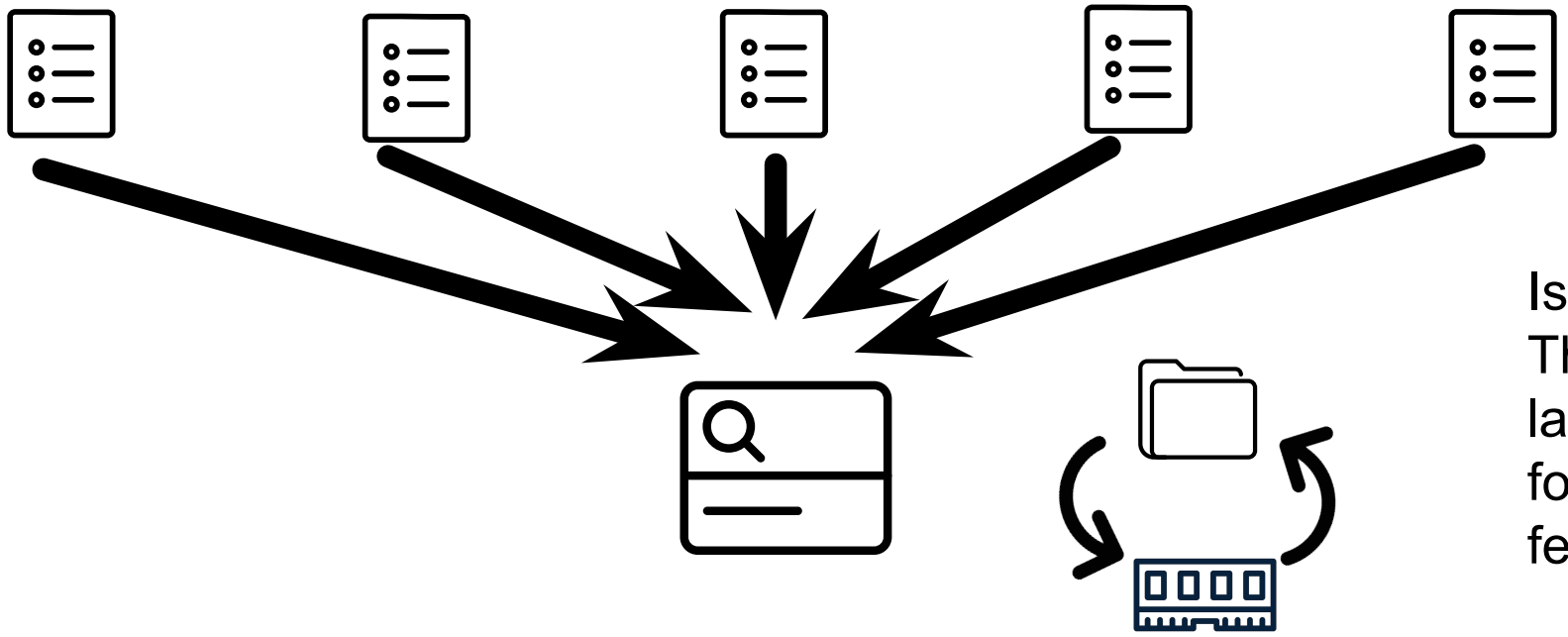
Let's have indexers send us **ALL** events.
We'll store them all.
We'll do all the search filtering locally.
We'll add up all the durations
!!!!

Okay, that was awful.

```
Sourcetype=cdr type=outgoing | stats sum(duration) by device_type
```

Network: So you like packets, eh?

Filesystem/RAM – Pulling everything to one host, there's just WAY too much data!



Is preview on?
Then the UI adds another layer of hellish load by forcing a finalize every few seconds.

Next – let's try something a little less terrible.

```
sourcetype=cdr type=outgoing | stats sum(duration) by device_type
```

So let's at least send this part out. Then the indexers can send only the matching events over the wire.

MUCH better. We are now at least doing **"distributed search"**.

Still a lot of resource usage on the SH though.

Search head says it's tired – you do it.

At least that was doing "distributed search. Let's do "distributed reporting", too.

```
sourcetype=cdr type=outgoing | stats sum(duration) by device_type
```

What if there was a way we could kind of… make the indexers do ALL the work.

Like, what if each indexer only sent back a tiny little table, literally like this:

device_type	sum(duration)
softphone	2422312
hardphone	858224
conference_bridge	582023
ip_communicator	590564
Jabber	18948

And that's what happens.

It's how Splunk does not just "distributed search" but "distributed reporting".

It's how you generally scale by adding indexers –the SH tier doesn't (shouldn't) have much work.

It's the "Reduce" part of Splunk's "Map Reduce" implementation. Or at least a really big part of it.

These little tables are sometimes called the "sufficient statistics" – half-baked cakes cooked by the indexers.

How it's actually implemented = "pre commands".

Send the "distributable streaming" commands, plus the relevant "pre" command.

```
sourcetype=cdr type=outgoing | stats sum(duration) by device_type
```

distributable streaming portion

Will include all distributable streaming commands (eval, where, rename etc..). Indexers run this part PLUS **"prestats".**

Transforming portion

Starts at the first non-"distributable streaming" command, goes all the way to the end.

SH runs just these commands at the end to tie it all together.

```
sourcetype=cdr type=outgoing
| prestats sum(duration) by
device_type
```

And this has all been happening automatically!

splunk> .conf22

Unless… you've been inadvertently writing suboptimal SPL™.

splunk> .conf22

…Like that ever happens.

splunk> .conf22

Part 2

How to be good.

How to ‘use stats’ even when it feels like you can’t.

splunk> .conf22

Example 1

(I can't use stats) …cause I don't want to and hey there's a join command

```
sourcetype=db | stats sum(rows) by pid
```

+

```
sourcetype=app | stats sum(cputime) by pid
```

=

```
sourcetype=db
| join pid [ search sourcetype=app ]
| stats sum(rows) sum(cputime) by pid
```

← **This is awful**

Example 1

(I can't use stats) …cause I don't want to and hey there's a join command

```
sourcetype=db
| join pid [ search sourcetype=app ]
| stats sum(rows) sum(cputime) by pid
```

← **This is awful**

```
sourcetype=db | stats sum(rows) as rows by pid
| join pid [
    search sourcetype=app
    | stats sum(cputime) as cputime by pid
]
| stats sum(rows) sum(cputime) by pid
```

← **Slightly better (but still bad)**

Example 1

Just send the whole mess along to stats/chart/timechart

```
sourcetype=db
| join pid [ search sourcetype=app ]
| stats sum(rows) sum(cputime) by pid
```

← This is awful

```
sourcetype=db OR sourcetype=app
| stats sum(rows) sum(cputime) by pid
```

← This is great

Example 2

… because I need to join first and THEN I need stats to do this other thing

```
sourcetype=cucm_cdr
| join callId [search sourcetype=cucm_cmr]
| stats perc5(MLQK) by type
```

← **This is bad**

```
sourcetype=cucm_cdr OR sourcetype=cucm_cmr
| stats values(MLQK) as MLQK
        last(type) as type
        by callId
| stats perc5(MLQK) by type
```

Nope – use stats.

NOTE: values() is underrated for single valued fields – just because your field is single valued today doesn't mean it will be tomorrow.

Example 3

… because the field names are different, but when I use rename it damages the “good” one

Let's say one side calls it “pid” and the other calls it “processId”

```
sourcetype=db
| rename processId as pid
| join pid [search sourcetype=app]
| stats sum(rows) sum(cputime) by pid
```

← **Bad**

```
sourcetype=db OR sourcetype=app
| eval pid=if(sourcetype=="db",processId,pid)
| stats sum(rows) sum(cputime) by pid
```

← **Good!**

Why not coalesce()?

– Assumptions change,
– Copy and paste pitfalls.
– Case() and if() are explicit and any assumptions they make are clearer to future readers.

“Conditional eval” is a powerful ally.

Example 4

I need some extra SPL to clean up one side, but if it touches the other rows it damages them

```
sourcetype=db
| rex field=pid mode=sed "s/cruft//g"
| join pid [search sourcetype=app]
| stats sum(rows) sum(cputime) by pid
```

Nope! It just needs more conditional eval

```
sourcetype=db OR sourcetype=app
| eval pid=if(sourcetype=="db",replace(pid,"cruft",""),pid)
| stats sum(rows) sum(cputime) by pid
```

ALSO you can sometimes use this to hide field(s) from the bad thing.

And yes, these examples are toys. More “real” ones don’t fit in slides.

Part 3 – Do I have to? How bad are my searches, really?

splunk> .conf22

How to know how you're doing (review)

"Pre" commands are in "remoteSearch".

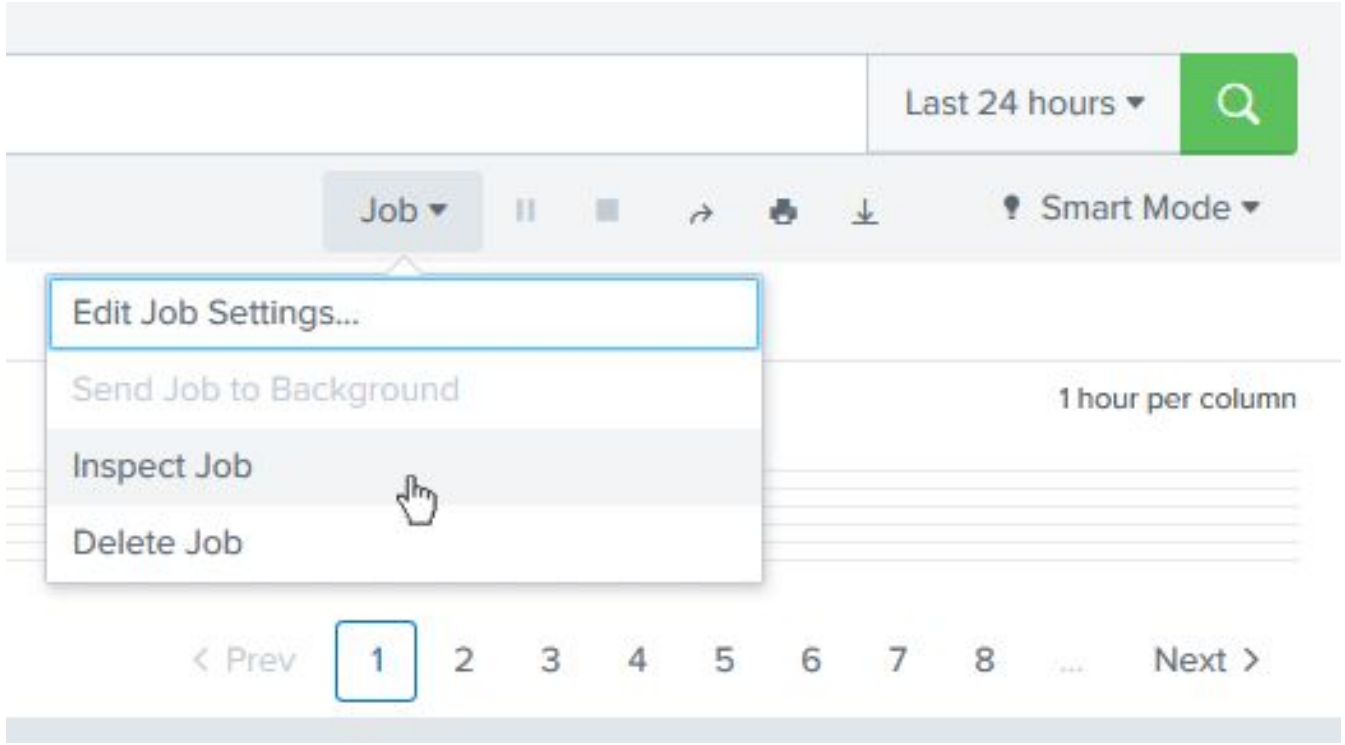
Click "Job" then "Inspect Job".

Close "execution costs", open "search job properties", Ctrl-F search for:

"remoteSearch" is what goes to the indexers.

"reportSearch" stays on the SH.

Also check out the "Sideview UI" app on Splunkbase.



Do I have to?

Maybe not!

With small data, fast searches, rows<<50K then none of this really matters. Pick your battles.

High cardinality “by id” searches– because of the nature of the query, sometimes even prestats is outputting a huge number of rows too. This cuts down on potential resources saved at the SH.

I only have a couple indexers anyway. (??)

Search Heads are free to me because someone else maintains them WAT no don’t be evil!

What about streamstats and eventstats?

Yes. Super powerful. Super useful. They deserve a whole talk to themselves.

However, despite what you might guess from their names, neither streamstats nor eventstats are "distributable streaming" commands. And they do not have “pre” commands. Therefore if they are the FIRST transforming command, you’ve bypassed distributed reporting.

So although many or possibly most transaction use cases could be refactored to use a combination of eval, streamstats, stats….
You'd still be breaking MapReduce and therefore you might stick with transaction.

Test it both ways !!

Trick – walk softly and carry a big transforming command

When you have a big expression with 2 or more transforming commands, try and rework it so that the first stats command will do most of the work reducing the weight involved.

Sometimes you can even eliminate the second one entirely.

Sometimes you can "set the table" really well with eval and streaming commands such that one big stats command can work a miracle in one pass, and thus do it out at the indexers, too.

```
eval {type}_duration=duration
eval {type}_callId=callId
`crazypants_macro_to_calculate_and_mvexpand_name_and_number_fields`
stats dc(incoming_callId) as incoming dc(outgoing_callId) as outgoing
dc(internal_callId)  as internal dc(callId) as total
sum(incoming_duration) as incoming_duration sum(outgoing_duration) as
sum(duration) as total_duration values(name) as name by number
```

Trick – break it into two problems

Sometimes when there's just way too much going on.
Look at what pieces only change rarely. Imagine if THOSE get baked out daily as a lookup, does the rest of the problem get easier?

Simple example -- I want to know the phones that have NOT made a call in the last week (and have thus generated no data). I could do a search over all time, then join with the same search over the last week.

Better - make a lookup that represents "all phones that have ever been seen" (i.e. with one hugely expensive all time search). Then bake that out as a lookup, then:

```
<terms> | eval present=1
| inputlookup all_phones_ever append=t
| stats values(present) as present by extension
| search NOT present=1
```

If there's a way, you can find it!

Or at least…. you can find crazy people in the Community who will help you

Even in super complex reporting situations, even after you've beaten it to death and given up, there are strange helpful people on Slack/Answers who have arcane knowledge and can totally help you. Many have strange hats.

The #search-help and #tinfoilstats channels on Slack are there to help.
(community slack sign up is at: http://splk.it/slack)

Make sure you give sufficient details and this generally means posting the SPL (yes, sometimes the raw actual SPL).

Thanks to everyone who made .conf happen this year!

And thanks for watching/reading/listening!

Please send any and all feedback or thoughts to nick@sideviewapps.com

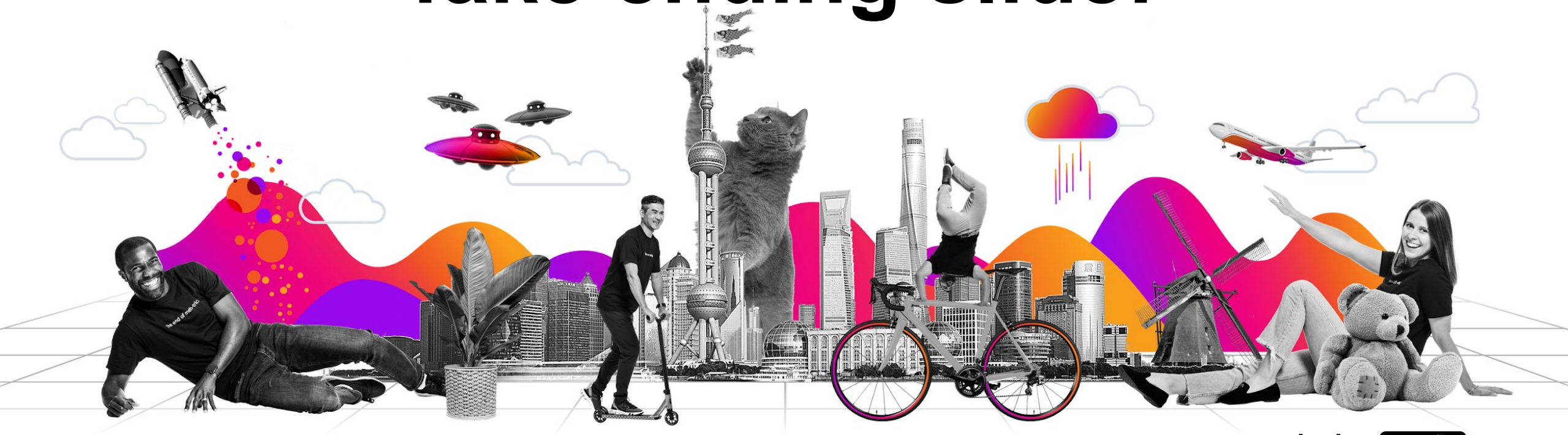
And please do take the session survey.

Thank You

splunk> .conf22

Hey!

You clicked past the fake ending slide!



splunk> .conf22

Glossary, page 1

Autofinalizing - See ‘finalizing’. This is when a search or subsearch is automatically finalized. In the UI there’s a message displayed. When it happens in a scheduled job it happens silently.

Disjunction - Is just a fancy word for using "OR" to join two sets of searchterms together.

Finalizing - Is basically when you click the "stop" button on a running search. Splunk stops getting new events off disk and kind of "wraps up" and returns the partial results. It might look complete but it's not.

Grouping: The official Splunk docs: (ie the page with the flow chart).
https://docs.splunk.com/Documentation/Splunk/8.2.6/Search/Abouteventcorrelation
https://docs.splunk.com/Documentation/Splunk/8.2.6/Search/Abouttransactions

Job Inspector: If you’re first learning about this now that’s scary but no worries. Run a search. Click the word “Job” and then “Inspect Job”...
Also, https://www.splunk.com/en_us/blog/tips-and-tricks/splunk-clara-fication-job-inspector.html?locale=en_us

Glossary, page 2

Join/append – You can read this but… in many key places where it tells you join or append is OK, it's wrong.
https://docs.splunk.com/Documentation/Splunk/8.2.6/SearchReference/SQLtoSplunk
So remember I said that. Note I avoided saying "inner left" join or any such sqlism here in this talk. They can all be done. If you were hoping for a mapping of how to do each one with stats and eval, I am sorry to disappoint.

Map/Reduce – This is a technical term for a system that divides up a chunk of work into smaller work items (the ‘map’ phase), and that specifies for how the output of the smaller items is combined into the final answer (the ‘reduce’ phase)**.**
https://en.wikipedia.org/wiki/MapReduce

Subsearch – Technically this refers to SPL expressions in square brackets, in a plain old search clause.
https://docs.splunk.com/Documentation/Splunk/8.0.5/Search/Aboutsubsearches
And conversely the term, imo, should NOT refer to how square brackets are used by the join or append commands.

Problem – I need more “distinctness”

In this example, we have “OrderNo”, and then “start” and “end” that are both times. The need was to calculate for each Service, the average time elapsed per order. The trick was that there were often numerous transactions per OrderNo and we had to treat each separately when calculating averages.

We relied on an assumption – that the Orders would never have interleaved transactions, and we used streamstats to supply the extra “distinctness”.

```
<search string>
| streamstats dc(start_time) as transaction_count by OrderNo
| stats earliest(start_time) as start_time earliest(stop_time) as
stop_time by OrderNo, transaction_count, Service
| eval duration=tostring(stop_time-start_time)
| stats mean(duration) as avg_duration by Service
| table Service, avg_duration
```

Problem – I have gaps in my ids

I don't really have one good id – instead I have two bad ones. Each side of the data has its own.
Luckily, there are some events that have both.
This feels a lot like a transaction use case and it might be.
(It might even be a searchtxn use case).

But whenever you have to kind of "fill in" or "smear out" data across a bunch of rows, also think of eventstats and streamstats.

Here we use eventstats to smear the JSESSIONID values over onto the other side.

```
sourcetype=access_combined OR sourcetype=appserver_log
| eventstats values(JSESSIONID) as JSESSIONID by ecid
| stats avg(foo) sum(bar) values(baz) by JSESSIONID
```

Problem – I need the raw event data

And with transaction I get the _raw for free

Do you really? I mean both "need" it and get it "for free" ?

But for debugging, yes sometimes you need _raw to make sense of things.
However you can get some debugging mileage out of:

```
... | stats list(_raw) as events by someId
```

Or even this trick, which forces ANY transforming result back into the "events" tab.

```
foo NOT foo
| append [
  search SEARCHTERMS | stats count sum(kb) as kb list(_raw) as
_raw by clientip host]
```

Problem – I need to search two different timeranges

But the two sides have different timeranges so I need join/append.
I need to see, out of the users active in the last 24 hours, the one with the highest number of incidents over the last 30 days.

```
  sourcetype=A | stats count by userid                (last 24 hours)
  sourcetype=B | stats dc(incidentId) by userid       (Last 7 days)
```

First back up – is the big one so static it could be a lookup?

```
  sourcetype=B | stats dc(incidentId) by userid | outputlookup user_incidents_7d.csv
```

OR Is the second one so small and cheap that it could be a simple subsearch?

```
  sourcetype=B [search sourcetype=A earliest=-24h@h | stats count by userid | fields
userid ]
  | stats dc(incidentId) by userid
```

Problem – I need to search two different timeranges

Nice try but that won't work…

No and the final results need to end up with values from that "inner" search, so I can't use a subsearch.

```
sourcetype=A | stats count values(host) by userid (-24h)
sourcetype=B | stats dc(incidentId) by userid      (-7d)
```

No problem. Still stats.

```
  sourcetype=A OR sourcetype=B
| eval isRecentA=
   if(sourcetype=A AND _time>relative_time(now(),
"-24h@h"),1,0)
| where sourcetype=B OR isRecentA=1
| eval hostFromA=if(sourcetype=A,host,null())
| stats dc(incidentId) values(hostFromA) as hosts by userid
```

Problem – I need to search two different timeranges

But

But that search…

```
  sourcetype=A OR sourcetype=B
| eval isRecentA=
 if(sourcetype=A AND _time>relative_time(now(), "-24h@h"),1,0)
| where sourcetype=B OR isRecentA=1
| eval hostFromA=if(sourcetype=A,host,null())
| stats dc(incidentId) values(hostFromA) as hosts by userid
```

… it gets lots of A off disk and then throws it away.

True! It's out at the indexers at least! "At least make the hot air go far away?"

But yes, the corresponding join may indeed be less evil here. Test it!

Problem – I need chart to mix split-by columns with normal columns

Aka “clown car techniques”

I need to have rows that are users, and then some columns that are from a “split by” field expression, and some other columns that are NOT from that splitby.

Sometimes this is expressed as needing to join the result output of two *different* transforming commands, ie:

```
sourcetype=A | chart count over userid by app

sourcetype=B | stats sum(kb) by userid
```

For each user I need the eventcounts across the 5 apps, PLUS the total KB added up from sourcetype B. This totally feels like what join is made for.

Clown car, continued

Nope. Stats. You may have heard that "chart is just stats wearing a funny hat". Or… if you haven't heard that before, well you've heard it now.

It turns out that you can always refactor chart into a stats and an xyseries.

```
| chart count over userid by application
```

Is equivalent to

```
| stats count by userid application
| xyseries userid application count
```

Clown car, continued

It's a little awful – you mash all the values into one clown-car field, and group by that, then unpack them all from the clown car afterward.
Here is a SIMPLE example.

```
sourcetype=A OR sourcetype=B
| stats sum(kb) as kb count by userid application
| eval clown_car = userid + ":::" + kb
| chart count over clown_car by application
| eval clown_car=mvsplit(clown_car, ":::")
| eval userid=mvindex(clown_car, 0)
| eval kb=mvindex(clown_car, 1)
| table userid kb *
```