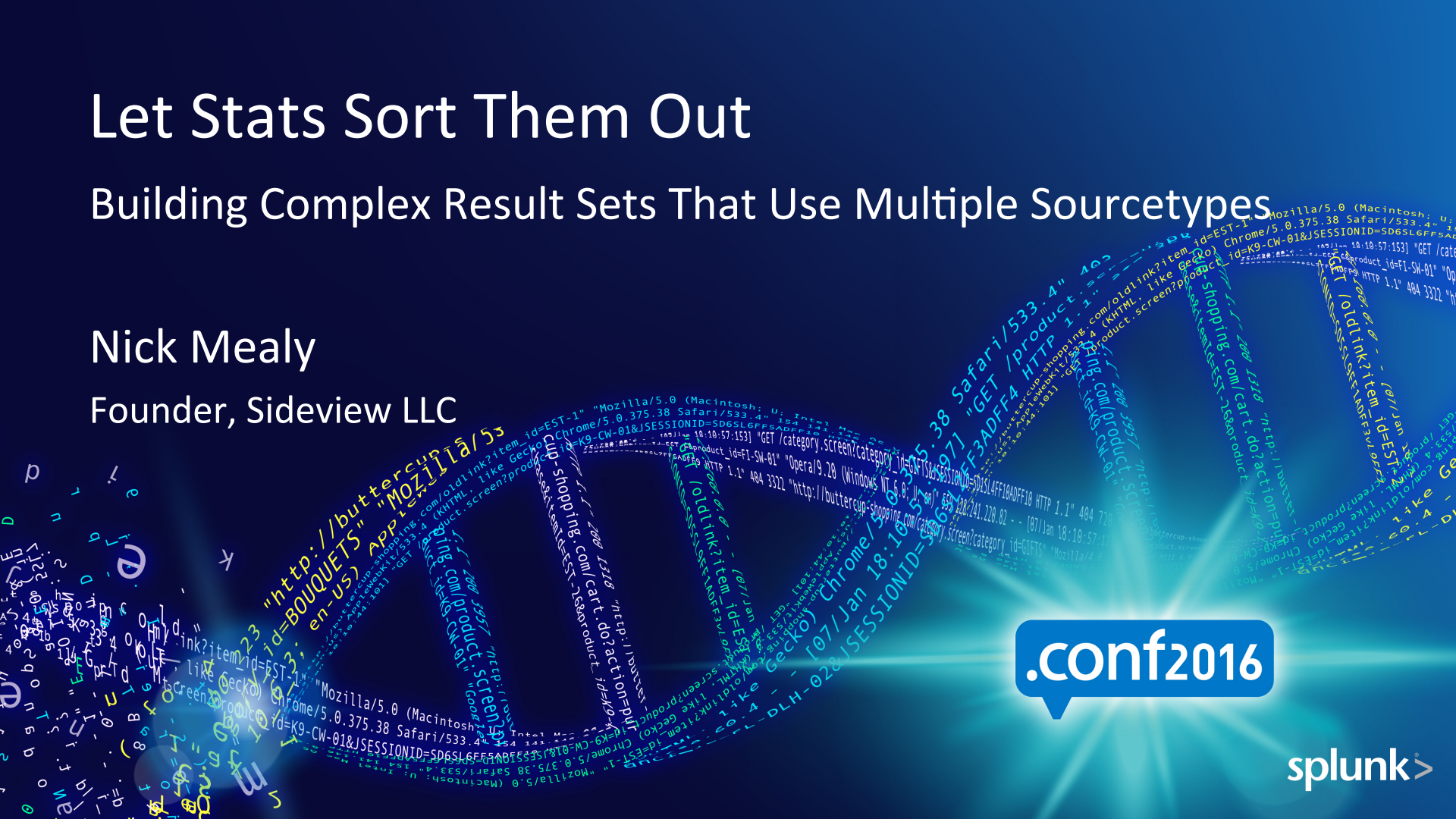
Let Stats Sort Them Out

Building Complex Result Sets That Use Multiple Sourcetypes

Nick Mealy

Founder, Sideview LLC



Who Is This Guy?

- Nick Mealy
- “sideview” On answers and slack, “madscient” on IRC
- SplunkTrust member
- Worked at Splunk in days of yore
  (Mad Scientist/Principal UI developer 2005-2010)
- Founded Sideview 2010. The Sideview Utils guy
- Search language expert

What's The Title Of This Talk?

Let stats sort them out - building complex result sets that use multiple sourcetypes.

We will talk about:

- Grouping!
- Why are the good things good and the bad things bad!
- Really? (yes. you can use stats to fold your laundry)
- Life after grouping – even more filtering and reporting!

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.

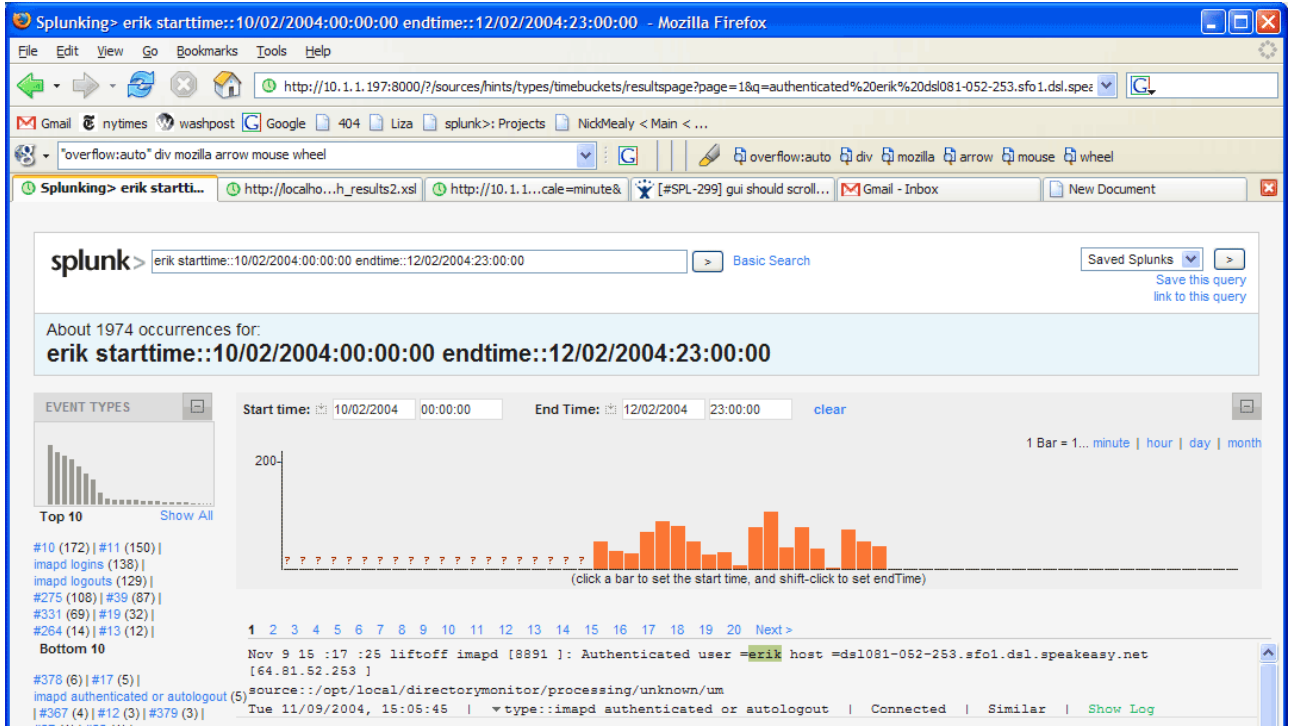
Splunk – The Short Short History

The following few slides contain backward looking statements regarding events in the past. We caution you that such statements reflect our imperfect memory and that actual events or results could differ materially. The backward-looking statements made in the next few slides are being made as of the time of the authoring of this presentation. If reviewed prior to such authoring, time and space may become unstable and the product, the conference and the known universe will become* subject to change without notice.

* (willan on becomen)

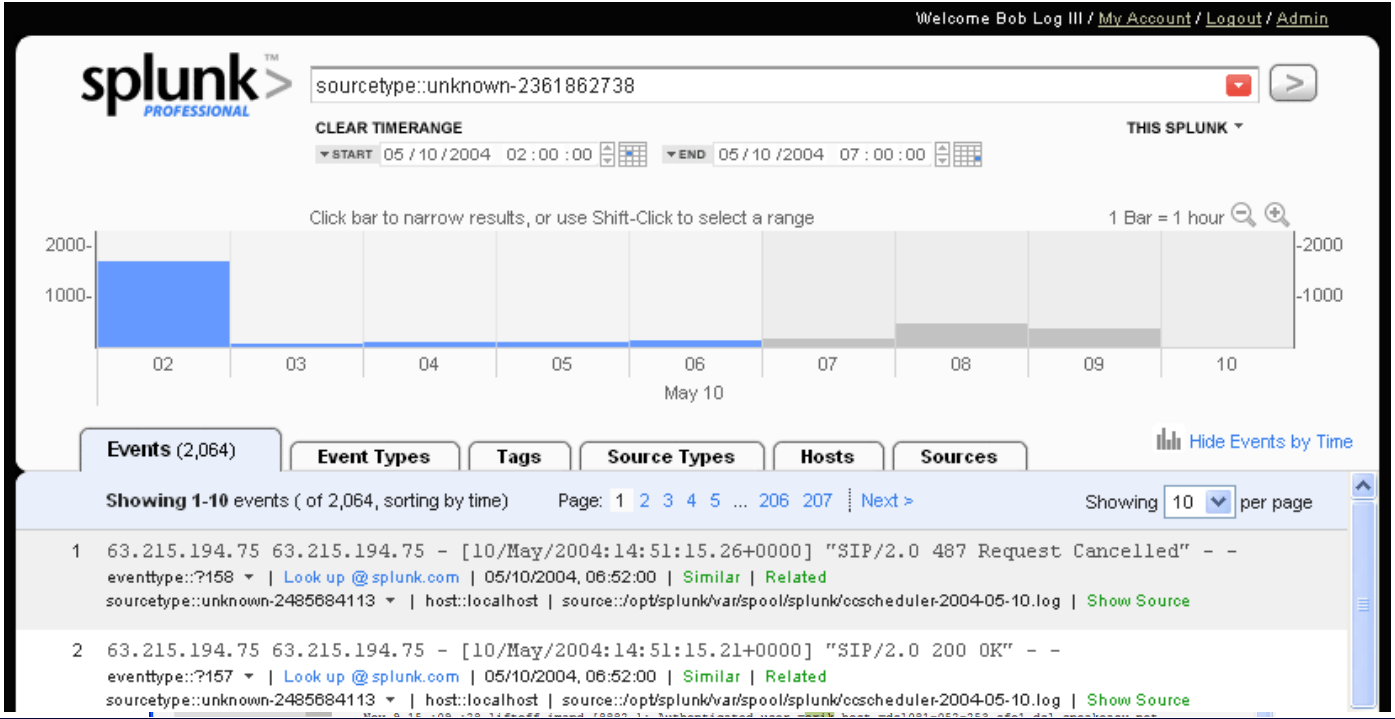
It’s 2004 And I Need To Search For Events!

Splunk 1.0 –Events are great!



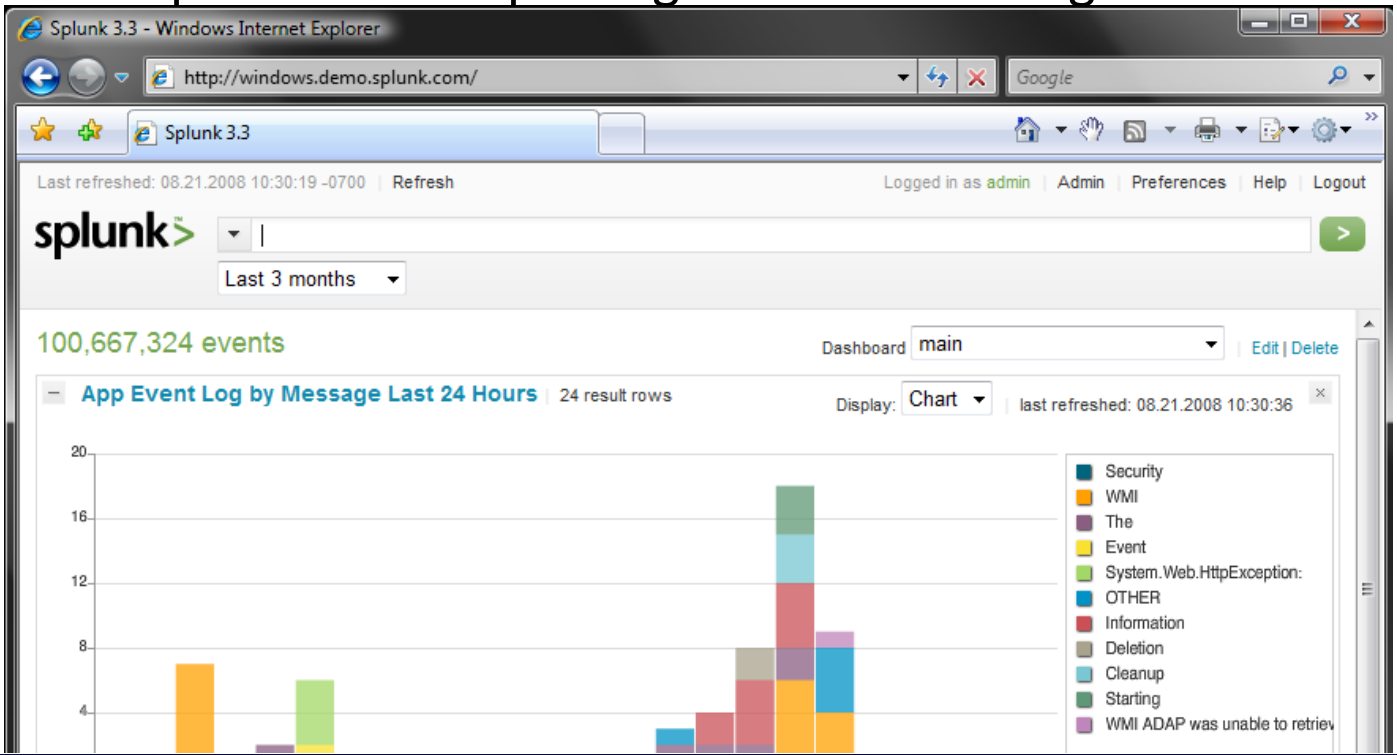
2006: OK, Except I Actually Need The Top 20 Hosts

Splunk 2.0 – Simple rollups are great!



2007: OK, But What I Actually Need Isn't A Simple Rollup

Splunk 3.0 – Reporting + dashboards = great!



2008: OK, But That Reporting Was Pretty Limited…

The report results I actually need to see look like this…

Splunk 4.0 – introduction of Search Language

(what the…)

The Splunk search language (SPL) is astonishing, but at first it looks both too complicated and also like it's just more "simple reporting".

It does simple things but it can do arbitrarily complicated, idiosyncratic and messy things.

(acknowledge the Splunk 5.x, 6.x – faster, better, easier, prettier, data-modelier, clusterier, not-appearing-in-this-talk-ier)

Why Is The Search Language So Weird?

The engineers who created the Splunk search language had to solve two different problems.

1) Give users a language to express what they need no matter how tweaky and insane it is.
2) Make it so as much of the work as possible can be farmed out to the indexers.

Arguably, weirdly, another name for #2 is…

“the stats command”

(with chart, timechart, top and all the si* commands just being stats wearing a funny hat.)

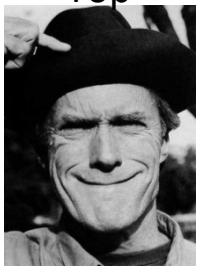
The Good

Stats

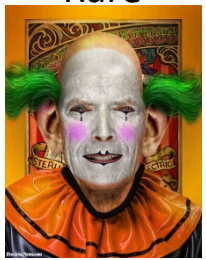
Chart

Timechart

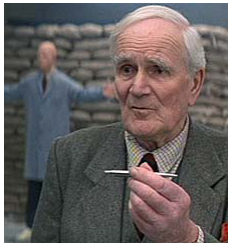
Top



Rare



Eval



Transaction

Get On With It Kid

This talk is for anyone who has ever said things like:

OK but in the results I actually need to see, each row is a <higher order thing> where the pieces come from different events.

I need to take two searches and mash them up!

I have to describe it in English and do you even want to hear it? it's pretty complicated.

You Know…Grouping!

Actually, This Talk Is A Little Non-trivial

(assuming we ever get to it)

This talk is for anyone who has ever said one of those things

But then also said

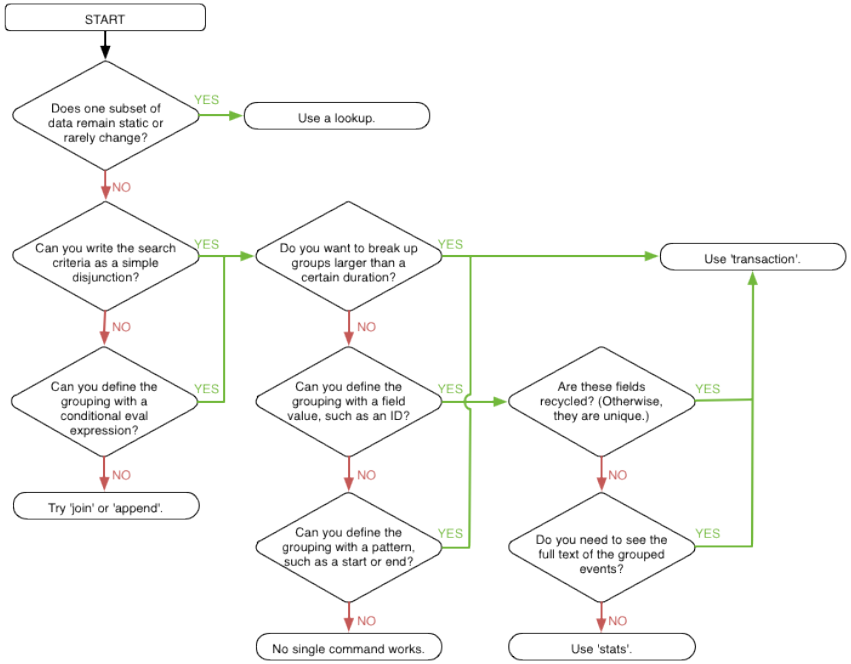
.. And I don't think I can just "use stats" here because <reasons>

“Use Stats”? Pls Explain

You may have seen this flowchart in the docs. It tries to get you to “use stats”, as if it were that simple.

If you have scoffed at this flowchart, you are in the right talk.

(I kind of wrote this flow chart, so the scoffing is OK by me).



So You Want To Group Some Things

Splunk has had amazing ability to do basically anything, since 4.0. Group anything by anything, turn it inside out and wash it through anything else. Rinse, repeat.

But it is a brutal learning curve.

One of the biggest pitfalls is how people go the wrong way, right away, away from core reporting functionality and into search commands designed for edge cases.

The Names Do Us No Favors Here

What would SQL do?

-- you will search the Splunk docs for “join”.

Hey the Splunk docs are using the generic word “transaction”.

-- you will search for more about "transaction".

I need to like… tack on another column.

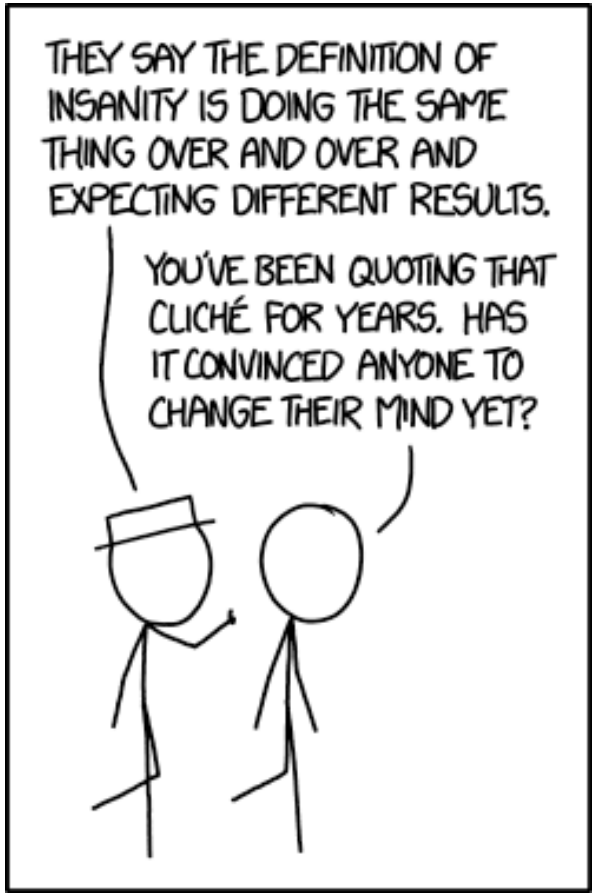
-- oh awesome there’s an "appendcols“ command!

The truth is that lookups and stats should be your first tools, and append/join and even transaction should be last resorts.

Great!

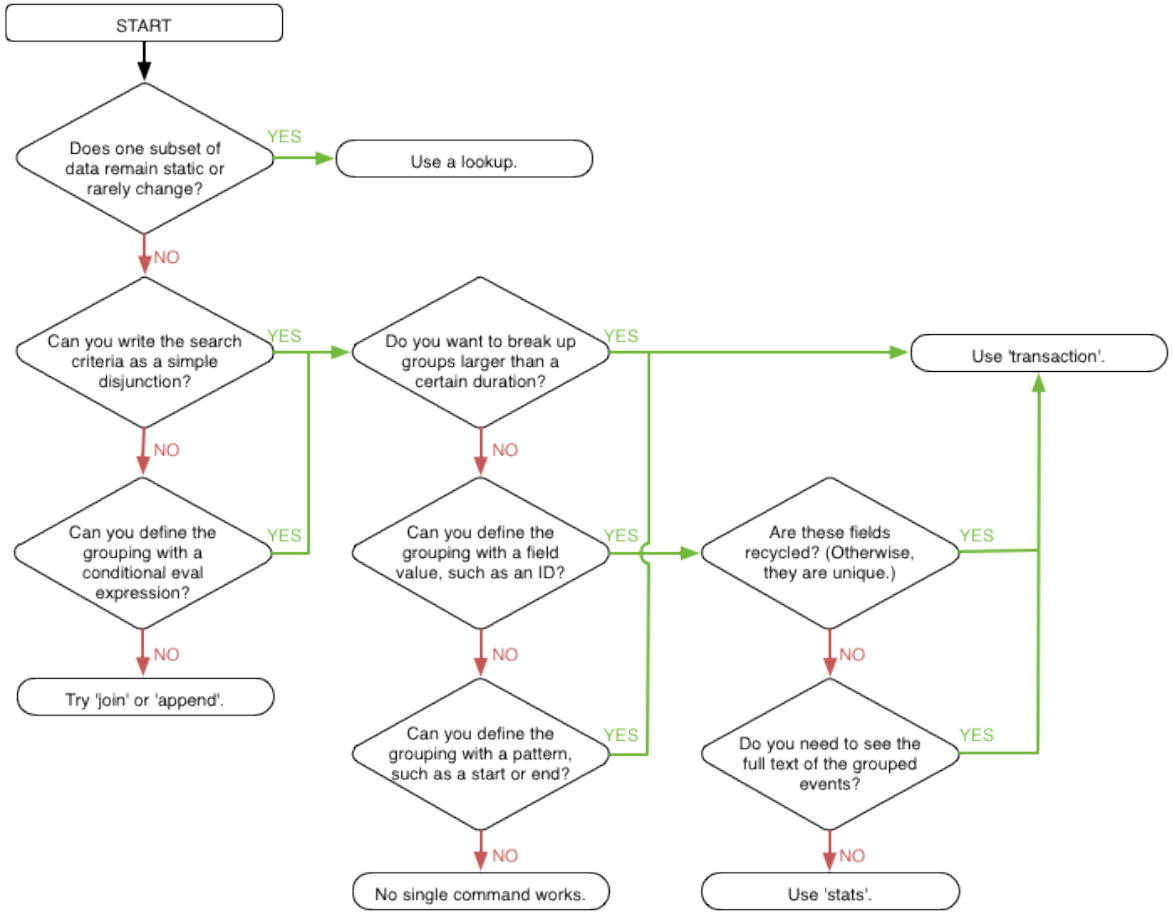
You convinced someone not to use join or append.

(Around the 30th time, you will create a flowchart)



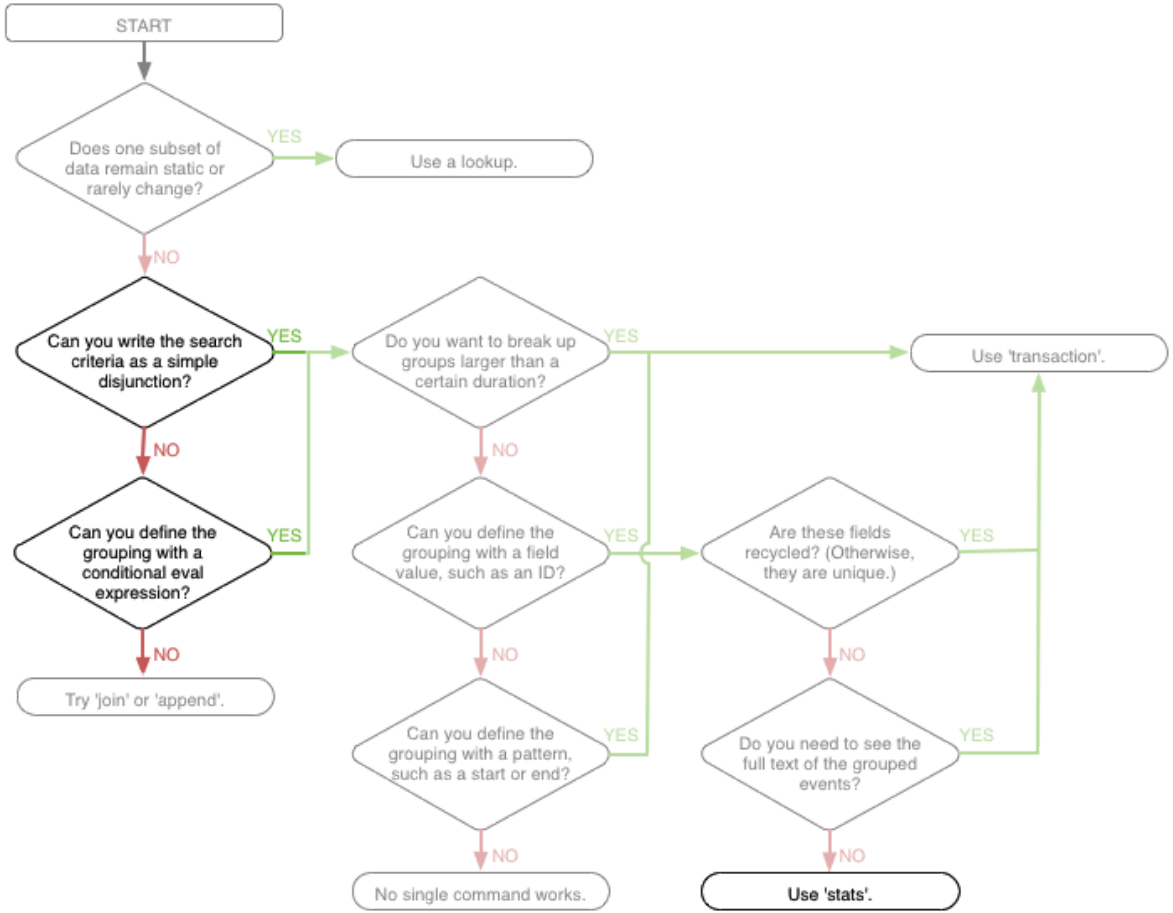
Flowchart

circa 2011



Flowchart

circa 2011



Flowchart Attack

Here is a more recent take on a flowchart.

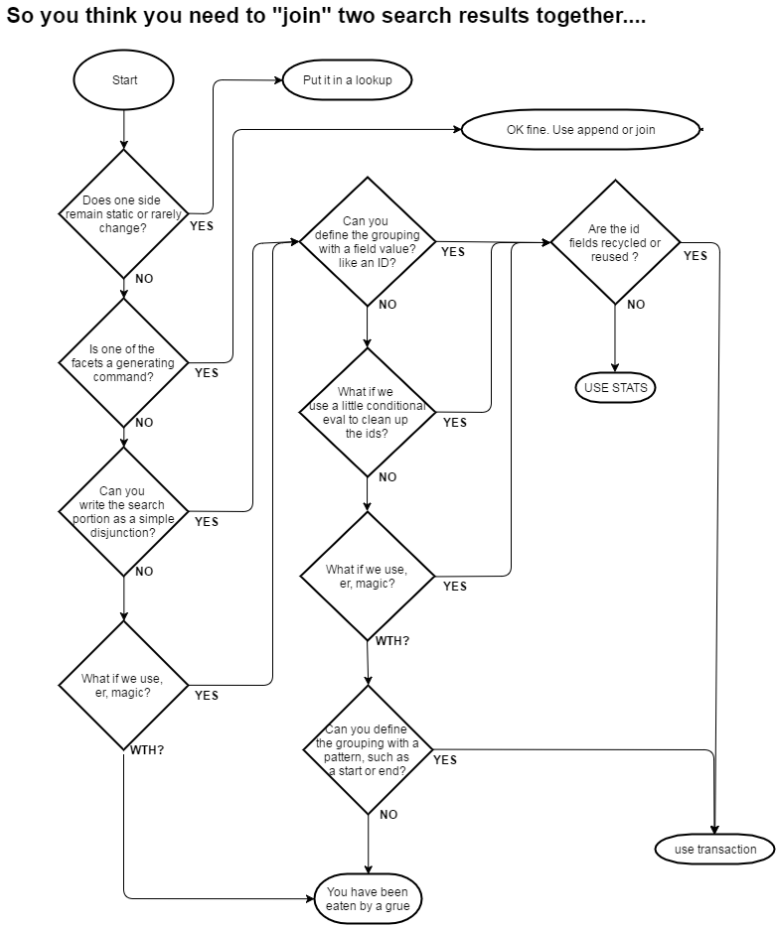
It still omits subsearches. =(

It still omits lots of advanced things,

And in place of jargon like “conditional eval” it now just refers to “magic”. Twice.

So let’s back away slowly.

Maybe flowcharts are bad.



What's Wrong With The Join And Append Commands?

Fundamentally slow, and as soon as you push any real volume of data through them, broken.

- Truncation if you exceed 50,000 rows. (and/or oom anxiety)
- Autofinalized if you exceed execution time.
- 2 Jobs instead of 1.
- Potentially getting the same data off disk twice.
- Extra overhead on job start.
- You might not even *realize* that you're hitting autofinalize and row-truncation limits, but they are nonetheless making 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.**

What's Wrong With The Transaction Command?

- Bristles with confusing edge-case logic. (what's this button do?)
- **Always breaks mapreduce, ie forces Splunk to pull raw rows back to the search head.**

If we're ever pulling raw event rows back to the search head…

…OK what are we even doing? For "dense" reports you may have accidentally created a big "grep deployment".

IOW: Surface Roads Are A Last Resort

What's Wrong With Map And Multisearch?

I don't know. What's wrong with nuclear weapons?

- They kill a few too many things
- Someone else has to clean up the strontium-90
- (Almost) never the simplest tool for the job

Lookup

- Is one side of the data essentially static, or changing so slowly that you could run a search once a day to take a new snapshot and that would be good enough?
- YES == Use a lookup

(Do think a bit about whether you'll need to search older data using the old mappings though.)

No = No lookup for you.

Subsearches

True actual subsearches are not fundamentally evil, although they certainly do have limits.

(Other things like join and append use square-bracket syntax, but these are only called “subsearches” because nobody ever came up with any other name.)

- Is one side always going to be a relatively sparse subset relative to the whole sourcetype?
- AND do you need this sparser side pretty much only because you need some set of field values from it in order to narrow down the results on other side?
- YES == subsearch!
  NO == no subsearch for you.

Join And Append

Is one side coming from a generating command, ie does it begin with a leading pipe? Eg, dbquery, ldapsearch, tstats.
YES == You need to use join or append*

NO == Believe it or not, you probably don't need join or append.

Read on.

* Unless that command has its own "append=t" argument, in which case you might be better off using that. eg tstats and inputlookup

Transaction

Is the only reliable way you can define the boundaries from one group to the next, defined by some kind of "start event" or "end event"?

YES = You can use transaction, but stats might be better anyway.

No = OK so you have id(s). Are they reused ever?

YES = you might as well use transaction

No = OK, the Id's aren't reused but are there some transitive shenanigans going on with these id's?

YES = you might as well use transaction
NO = o_O. This was never a transaction use case in the first place. Stats!

Map Or Multisearch

Do you totally feel like you need a map command or a multisearch command?

- YES. OK, er, wait here a minute. Breathe deeply. You might be right but lets get you some help on answers.splunk.com

OK, But I Don't Think I Can Use Stats Here Because Y

(The rest of this talk will be Nick trying to solve for Y)

Example #1 - Newbie

… because I found join and join does exactly what I need.

```
sourcetype=A | join myIdField [search sourcetype=B]
```

This is magnificently, marvelously wrong.

So wrong it's hard sometimes to even remember how often you can find people doing it.

```
Sourcetype=A OR sourcetype=B
| stats <various things> by myIdField
```

Example #2 – (Still Too Easy)

I don't think I can use stats here because one side calls it "id" and the other calls it "userId".

Piece of cake. use coalesce()

```
  sourcetype=A OR sourcetype=B
| eval normalizedId=coalesce(id, userId)
| stats count sum(foo) by normalizedId
```

Example #3 - Newbie

Actually I can't use coalesce() because… <reasons>.

Yeah, I lied a little there. You can use coalesce but don't. Use case() use eval().

Coalesce is great until you have a day where autokv accidentally throws in a field by the same name in an unexpected place so your coalesce() grabs the wrong one and now your data is wrong and.... Best of all you might not even find out about this problem for a while.

And the overall logic to get the right id often gets one or two little wrinkly nuances in it.

-- Coalesce is a hammer. You might hit the right nail.
-- If and Case are nailguns. The nail they're hitting is listed right there.

```
| eval id=case(sourcetype="A",id,sourcetype="B",otherId)
| stats count sum(foo) by id
```

If() And Case() – Be Explicit. Be, Be Explicit

Sourcetype A has `device=EEC0B84221E2B31`
Sourcetype B has `macAddr=0B84221E2B31`

```
| eval macAddr=case(sourcetype="A",replace(device,"^EEC",""),
sourcetype="B",macAddr)
```

In theory you could run the regex replacement on both sides and then coalesce(), but what if your repair accidentally damages the other side? Oh yea look at that, it could.

This represents the first glimpse of a whole world of "the thing I need to do one side seems to damage the other side". We haven't seen the last of this.
And "conditional eval" stuff is often the solution.

Sidebar – Trying Things Yourself

Stats can also be a little Swiss army knife to help you test out search language behavior.

```
| stats count | fields – count
```

Will make a single row, with no fields at all…. ?

Let's say you're wondering about that mac address scenario. Here's a search that generates it.

```
| stats count
| eval foo=split("EEC0B84221E2B31,0B84221E2B31")
| mvexpand foo | streamstats count
| eval device=if(count==0,foo)   | eval sourcetype=if(count==0,"A")
| eval macAddr=if(count==1,foo) | eval macAddr=if(count==1,"B")
| fields – foo count
```

(OK Fine, there's a new command called "makeresults" too that does the same thing. :P)

Example #4 – Gluing Things To Other Things

I don’t think stats is right because I just need to just glue some result rows together.

Say you want to calculate one thing from one set of fields in one set of events, and at the same time calculate another thing from another set of fields in some other events.

I don't really need to "join" them, I just want to... what's the word.... **APPEND!**

```
sourcetype=A | stats avg(session_length) as length
+
sourcetype=B | stats dc(sessions) as sessions dc(users) as users
=
sourcetype=A | stats avg(session_length) as length
| append [sourcetype=B | stats dc(sessions) as sessions dc(users) as
users]
```

Example #4 – Cont.

I don't think stats is right because I just need to just glue some result rows together.

No, you can still use stats. It's OK- stats doesn't care. It will effectively calculate your two things separately, handle the gaps just fine, then glue them together at the end.

```
sourcetype=A | stats avg(session_length) as length
+
sourcetype=B | stats dc(sessions) as sessions dc(users) as
users
=
sourcetype=A OR sourcetype=B
| stats avg(session_length) as length dc(sessions) as sessions
dc(users)
```

Example #5 – Gluing + Joinery

I want to calculate one thing from one set of fields in one set of events, and at the same time calculate something else for which I have to kinda… "join" things from both sides.

```
sourcetype=A | stats sum(kb) by ip

sourcetype=B | stats dc(sessionid) by ip
```

AND I like join because I need to be careful -- sometimes sourcetype B has another field also called "kb"!

(or maybe sourcetype=A has a field called sessionid.)

Example #5 – Gluing + Joinery, Cont.

Solution: needs more nailgun

```
sourcetype=A | stats sum(kb) by ip
+
sourcetype=B | stats dc(sessionid) by ip
=
sourcetype=A OR sourcetype=B
| eval kb=if(sourcetype="B",null(),kb)
| eval sessionId=if(sourcetype="A",null(),sessionId)
| stats sum(kb) dc(sessionid) by ip
```

Example #5 –About Those Inline Expressions

```
sourcetype=A OR sourcetype=B
| eval kb=if(sourcetype="B",null(),kb)
| eval sessionId=if(sourcetype="A",null(),sessionId)
| stats sum(kb) dc(sessionid) by ip
```

To be fair, that sourcetype="X" is standing in for what might be a more complex expression. Often you'll want to pull it out as its own "marker field". Clarity above all things.

```
| eval isFromDataSet1=if(<ugly expression>,1,0)
| eval kb=if(isFromDataSet1,null(),kb)
| …
```

Example #6 – Timerange Shenanigans

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)
```

If it's a join you're leaning towards, this may well be a subsearch use case hiding in plain sight.

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

Example #7 – More timerange Shenanigans

I have different timeranges but I can't use a subsearch because my outer results need some values from that "inner" search. Specifically, I need the hosts that the users have been on in the last 24 hours.

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

No problem. Stats.

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

It’s a little ugly and yes, a fair bit of events from “A” get thrown away. But no OOM risk! No autofinalizing!

Example #8 – IDK. Still Pretty Sure I Need Join

I have:

```
sourcetype = a | table _time, id
sourcetype = b | table bid, cid
sourcetype = c | table cid
```

sourcetype=a

Then left join with:

sourcetype=b, with a.id = b.bid

Then left join with:

sourcetype=c with b.cid = c.cid

I want to end up with: a._time, a.id, b.bid, c.cid, so clearly join right?

Nope! Conditional eval + Stats.

```
sourcetype=a OR sourcetype=b OR
sourcetype=c
| eval id=if(sourcetype="b",bid,id)
| eval
a_time=if(sourcetype="a",_time,null(
))
| stats values(cid) as cid
values(a_time) as a_time by id
```

Sidebar – Values(foo), First(foo), “By Foo”

It takes a while to learn how to choose between

```
| stats sum(kb) by field1 field2 field3
vs
| stats sum(kb) values(field3) as field3 by field1 field2
vs
| stats sum(kb) first(field3) as field3 by field1 field2
```

- Do I need it as a group-by field?
  YES - Great. Just make sure it’s never null or you’ll be losing rows unexpectedly.(iow you may need a fillnull)
- **Actually it seems kinda wrong that way -** it probably is. Try values()

Avoid first() and last() and earliest() and latest() unless there are other values in there and you specifically want to ignore them.

If you’re confident other values will never exist…. Even then I say use values() anyway.

Example #9

But I need the raw event data rolled up so this means I need the transaction command.

Well... when pressed the person may admit that they don't actually care about the raw text, they just like seeing it for debugging.

If that's true, meet a quick and dirty trick:

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

Example #10

I just want to group things by this ID, and transaction seems like the tool for the job.

It does feel right, but it's very wrong!

If you're using transaction but you're NOT using any of its fancy arguments like maxevents or maxspan or startswith or endswith, AND you don't need to see the raw events,

and then after the transaction you're doing something like stats, chart, timechart, you can and should use stats instead.

Transaction forces Splunk to bring all these events back to the server, and in a lot of simple cases it's just not necessary.

Example #10 – Cont.

```
| transaction field1 field2
| stats sum(foo) count by field1 field2
```

… just dissolves away and should just be

```
| stats sum(foo) count by field1 field2
```

Which is so weird it seems like I'm cheating, so lets look at more complicated examples.

Example #10 – Cont.

Aha! I need the “duration” field!

```
| transaction field1 field2
| stats sum(duration) as duration by field1 field2
```

Nope. Use stats.

```
| stats min(_time) as start max(_time) as end by
field1 field2
| eval duration=end-start
```

Sidebar – Stats And _time

In case that

```
| stats min(_time) as start max(_time) as end
```

Looked funny to you.

You might say use first() and last() because they're very slightly faster. You'd be right.

And you might say that the event rows coming into stats will always be reverse-time ordered and you'd be…. **WRONG. WAT.**

Outside of "verbose mode", the event rows are mostly in reverse-time-order but it's not guaranteed.

So use first() and last()… pretty much when you have an explicit sort beforehand. <cue minor uproar>

If I'm positive there's only one anyway I still use values() (so if that assumption breaks it shows up fast).

Earliest() and latest() – they make my head hurt. I worry about them getting pasted into places where _time is not on every row. So… this is why I use max() and min() kind of a lot.

Sidebar In A Sidebar

Beware earliest() and latest() when _time isn't present on every row.

```
| stats count
| eval name=split("agnes,mildred,elihu,ruth",",")
| mvexpand name
| streamstats count
| eval _time=if(count=="3",1423231321,null()) | stats count earliest(name)
```

_time is of course always present on raw events, but stats often processes rows that have already been through some reporting command… And the more you use stats in the way I'm talking about in this talk, the more likely you'll give it some rows with no _time's.

Example #12

I think I need transaction because

… I have a start event. … I have an end event,

… my supposedly unique id's actually get reused

If you need these things, you most likely *could* use clever constructions with eval, streamstats and stats to do the same thing.

The catch – even though streamstats is streaming non-transforming, the rows coming into it always have to get brought back to the search head.

If you're going to break map-reduce anyway, go ahead and use transaction if you prefer it.

Example #13

But.. I have to do this thing to one side to make it how I want, and that thing involves one or more search commands that would irreparably damage the other side.

FIRST -- subsearch use cases can hide here. Especially if one side is sparse/fast/short.

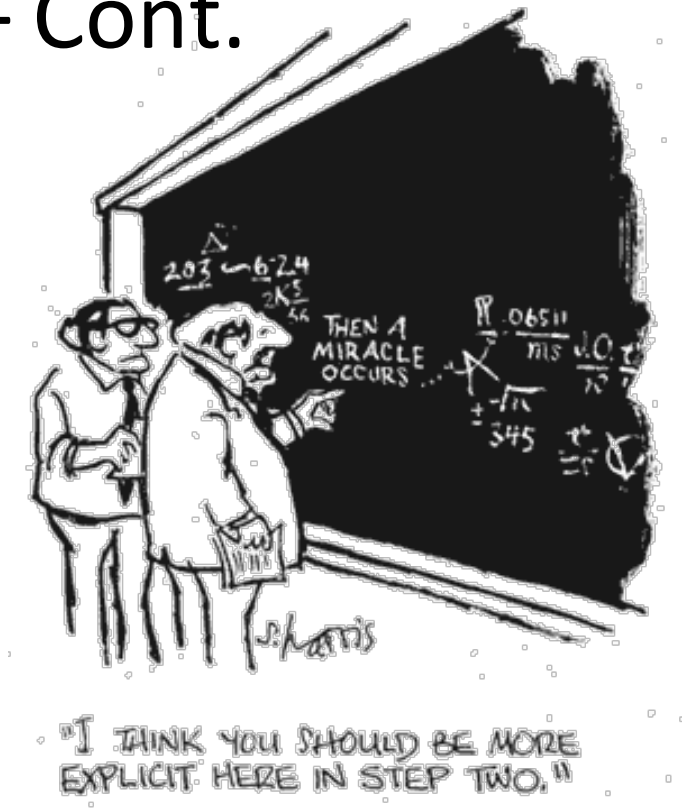
but failing that sometimes this sort of thing does sometimes force you over to join/append.

...but not always.

Example #13 – Cont.

The smaller side needs some search language that is just very expensive or scary (eg xmlkv) and we don't want to run that on the other side.

Sometimes you can conditionally eval your fields to hide them from the scary thing, then put them back after.



Example #14

Sorry smart guy, I literally need to join the result output of two *different* transforming commands.

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

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

I need to end up with the eventcounts across the 5 applications, plus the total KB added up from sourcetype B. I need stats behavior AND I need chart behavior!

So I need appendcols! QED!

Example #14 – Cont.

Nope. Stats! Remember that most transforming commands are just stats wearing a funny hat. In other words with a little eval, a little xyseries and/or untable we can often refactor both searches to have stats commands.

Refactor the chart search into a stats search plus... some other stuff to make the rows look the same. (In this case an xyseries)

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

Is equivalent to

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

Example #14 – Cont.

Now lets forget about xyseries for the moment. Let's try and get one stats command to do the work of both the ones we have.

```
A:    | stats sum(kb) as kb by userid
B:    | stats count by userid application | <xyseries magic>
```

We make a disjunction. OK but stats will throw away rows with null application values so we have to workaround that. Ick.

```
sourcetype=A OR sourcetype=B
| fillnull application value="SO MUCH NULL"
| stats sum(kb) as kb count by userid application
| eval application=if(application="SO MUCH NULL",null(),application)
```

Example #14 – Cont...

```
sourcetype=A OR sourcetype=B
 | fillnull application value="NULL"
 | stats sum(kb) as kb count by userid application
 | eval application=if(application="NULL",null(),application)
```

ok now we have fields of `userid application kb count`

and we need fields that are `userid kb app1count app2count app3count app4count`

if only we could do two "group by" fields in the chart command!!

```
chart count over userid kb by application
```

OMG We can't!!! <sad trombone>

Example #14 – Cont....

Oh no wait we can. It's just a bit, er, hideous.

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

Example #15 – The Handwave

Complicated thing that seems to need a transforming command on **only one** of the 2 sides, but that can be rewritten to use eventstats and/or streamstats and eval in some mind-bending way.

These exist.

Whether the end result is worth the warping of your mind is perhaps a different question.

OK, Now We Have The Results. What Then?

Why, filter and report on them of course!

Once we get these “higher order” rows, your users will get used to them and want to start “doing things” with them. What will emerge:

- One class of searchterms users intuitively expect to use, that apply only to the higher-order entity.
- Another class of… filtering / categorizing / tagging that users need to apply at the raw event level, and/or that need to bubble up to the final report level.
- Even a third level of filtering if you include even higher-level rollup reports at the top.

```
< raw event search>                                    (level 1)
| stats sum(foo) as foo values(bar) as bar … by id1 id2
| search <more filtering at the higher level> (level 2)
| chart count over id1 by bar
| search <omg even more filtering> (level 3)
```

Some Advice

Rule 1 – Pretty much all event-level filtering has to be in a subsearch.

Rule 2 - If there's more than one event-level subsearch term, they have to be separated by OR's. And then at the end you have to filter again with the same terms (without the OR's).

Why? **No row left behind!**

You have to beat this into your brain a little bit because there's a strong temptation later to sneak simple search terms into the event-search, or to put things in the subsearch as AND'ed terms instead of OR'ed terms.

Oh and eventtypes can come in handy. Let them ride up on raw event rows then use them as filters at the grouping level.

Appendix

There are more slides, but you'll have to read them yourself.

Go forth and do great things.

If you have questions, track me down. If you want to hear endless stories about the early or middle history of Splunk, buy me a drink.

THANK YOU

splunk>

Oh Hey

You clicked past the fake ending slide.

Nice work.

There are some more slides with more examples.

Example #16 – Complex Call Flow

UK Customers are reporting that they call in from overseas and get transferred around and get hung up on.

Q1: Where do they get disconnected?

```
`cdr_events` [search `cdr_events` callingPartyCountryCode="44" | fields
`id_fields`]
| sort 0 - dateTimeConnect
| stats first(finalCalledPartyNumber) as terminatingCalledPartyNumber
values(eventtype) as eventtype values(originalCalledPartyNumber) as
originalCalledPartyNumber values(callingPartyNumber) as callingPartyNumber
sum(transfers) as transfers by globalCallID_callId
globalCallID_callManagerId globalCallId_ClusterID
| search transfers>2 eventtype="incoming_call" callingPartyCountryCode="44"
| chart count as calls over terminatingCalledPartyNumber | sort 0 count
desc
```

Example #17 – Simplify

Sometimes when there's just a whole lot going on, you can break it into two things and bake one of them out as a lookup.

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 ever" (i.e. with that expensive all time search). Then:

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

Example #18 – Call Concurrency

How long is this long tail? I have no idea.

Let's leap out to something pretty far out. Concurrency.

Splunk has a concurrency command. It's neat.

But you usually end up needing concurrency by someField.

I need to calculate the concurrency of two different things, in one chart. But concurrency has no splitby so I need to append these and then re-timechart them

```
No silly, you can use eval, mvexpand, fillnull, streamstats,
timechart, filldown and foreach.
```

```
=/   (see next slide)
```

Example #18 – Call Concurrency, Cont.

```
`cdr_events`
| eval increment = mvappend("1","-1")
| mvexpand increment
| eval _time = if(increment==1, _time, _time + duration)
| sort 0 + _time
| fillnull SPLITBYFIELD value="NULL"
| streamstats sum(increment) as post_concurrency by SPLITBYFIELD
| eval concurrency = if(increment==-1, post_concurrency+1, post_concurrency)
| timechart bins=400 max(concurrency) as max_concurrency last(post_concurrency)
as last_concurrency by SPLITBYFIELD limit=30
| filldown last_concurrency*
| foreach "max_concurrency: *" [eval <<MATCHSTR>>=coalesce('max_concurrency:
<<MATCHSTR>>','last_concurrency: <<MATCHSTR>>')]
| fields - last_concurrency* max_concurrency*
```