Splunk Performance

Observations and Recommendations

Simeon Yep | AVP GSA
Brian Wooden | GSA Partner Integrations

2017-09-27 | Washington, DC

Forward-Looking Statements

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

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

Agenda

- Performance & Bottlenecks
  - The BBQ Analogy
- Indexing
  - Index-time Pipelines
  - Indexing Tests
- Searching
  - Without and With Indexing Load
  - Search Types
  - Mixed Workload Impacts
- Metric Store

Testing Caveats

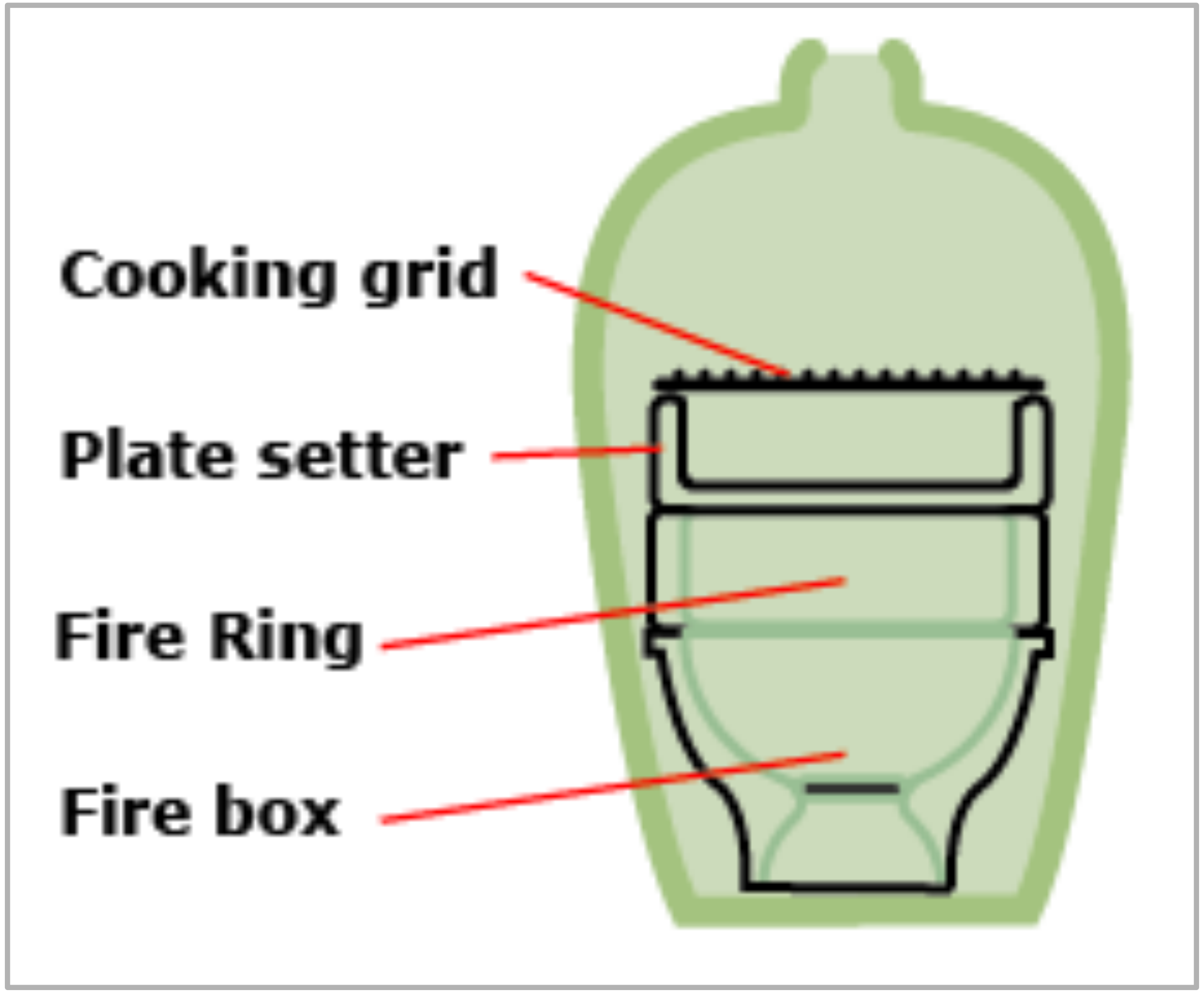
Do Not Take Results Out of Context

- Arbitrary Datasets Used
- “Dedicated/Isolated” Lab Testing

My Splunk is Slow

I knew I should have used SSD

- If we remove one bottleneck another will emerge
- Let's get cooking

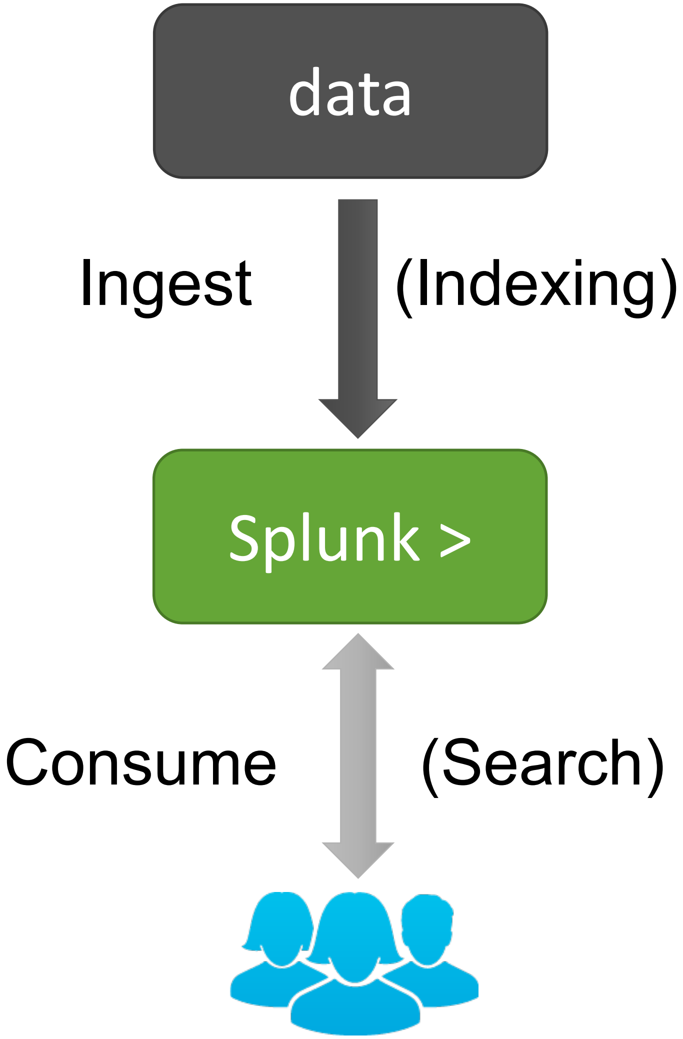


"Splunk, like all distributed computing systems, has various bottlenecks that manifest themselves differently depending on workloads being processed."

-The one they call D

Identifying performance bottlenecks

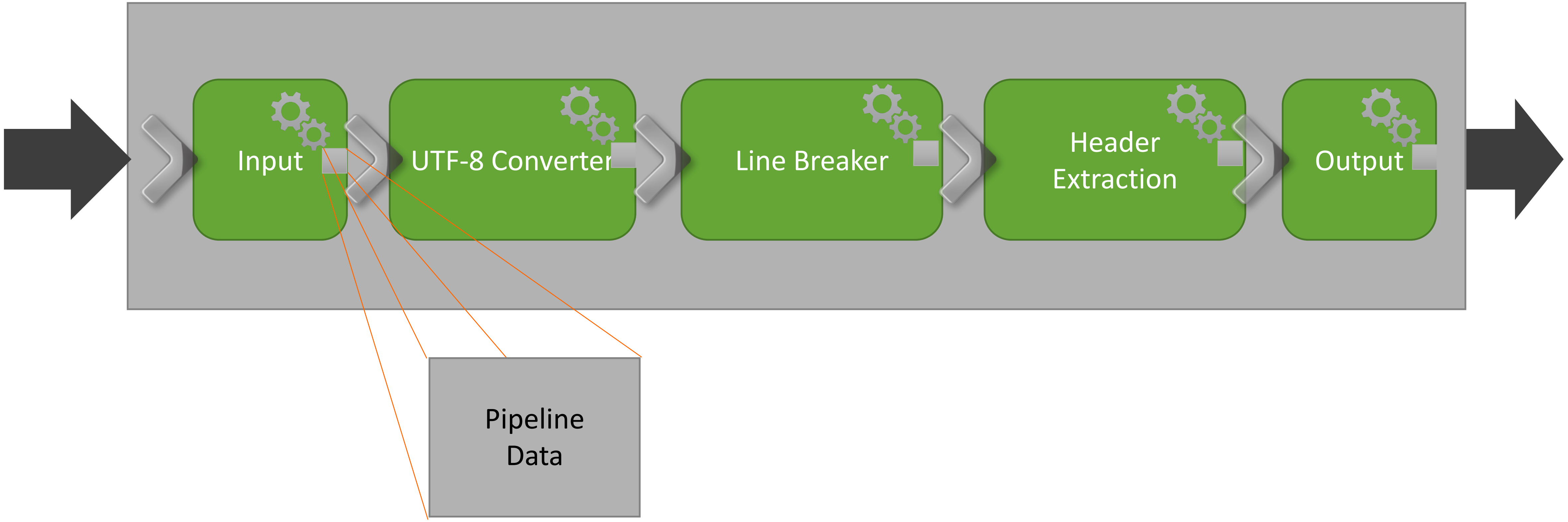
- Understand data flows
  - Splunk operations pipelines
- Instrument
  - Capture metrics for relevant operations
- Run tests
- Draw conclusions
  - Chart and table metrics, looks for emerging patterns
- **Make recommendations**



Indexing

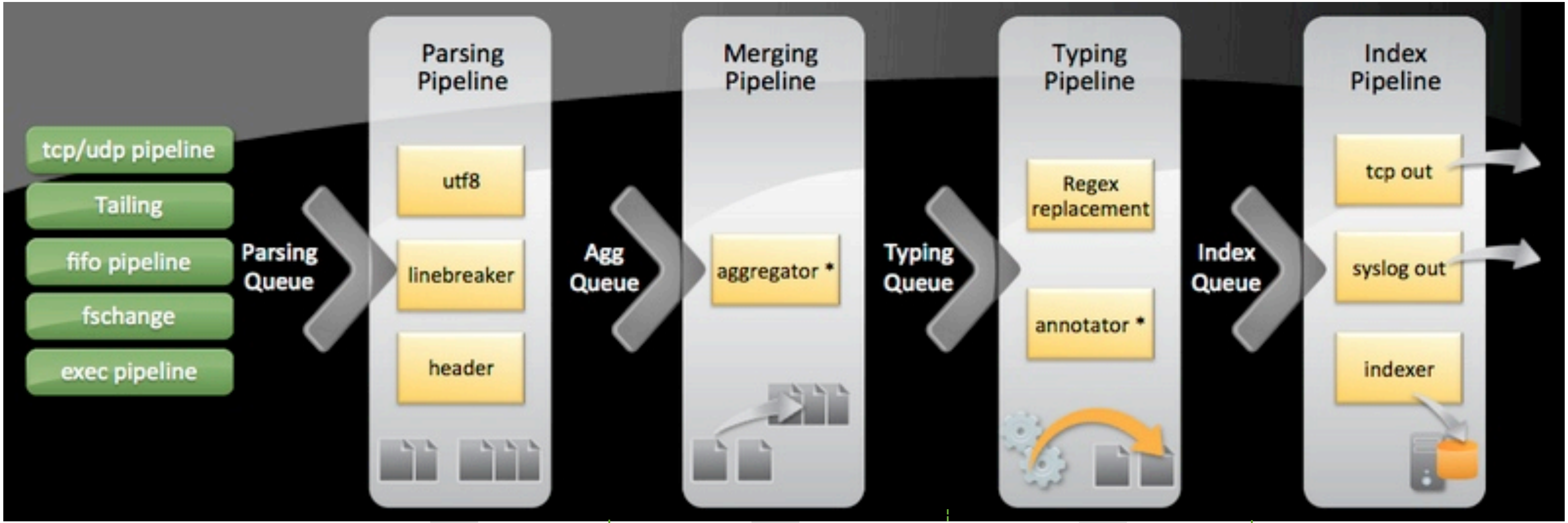
Pipelines, queues, and tests

Put that in your pipeline and process it



Splunk data flows thru several such pipelines before it gets indexed

Lots of pipelines



LINE_BREAKER
TRUNCATE

SHOULD_LINEMERGE
BREAK_ONLY_BEFORE
MUST_BREAK_AFTER
TIME_*

TRANSFORMS-xxx
SEDCMD
ANNOTATE_PUNCT

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> ANNOTATE_PUNCT <enable/disable punct:: extraction>
Typing	

Testing: dataset A

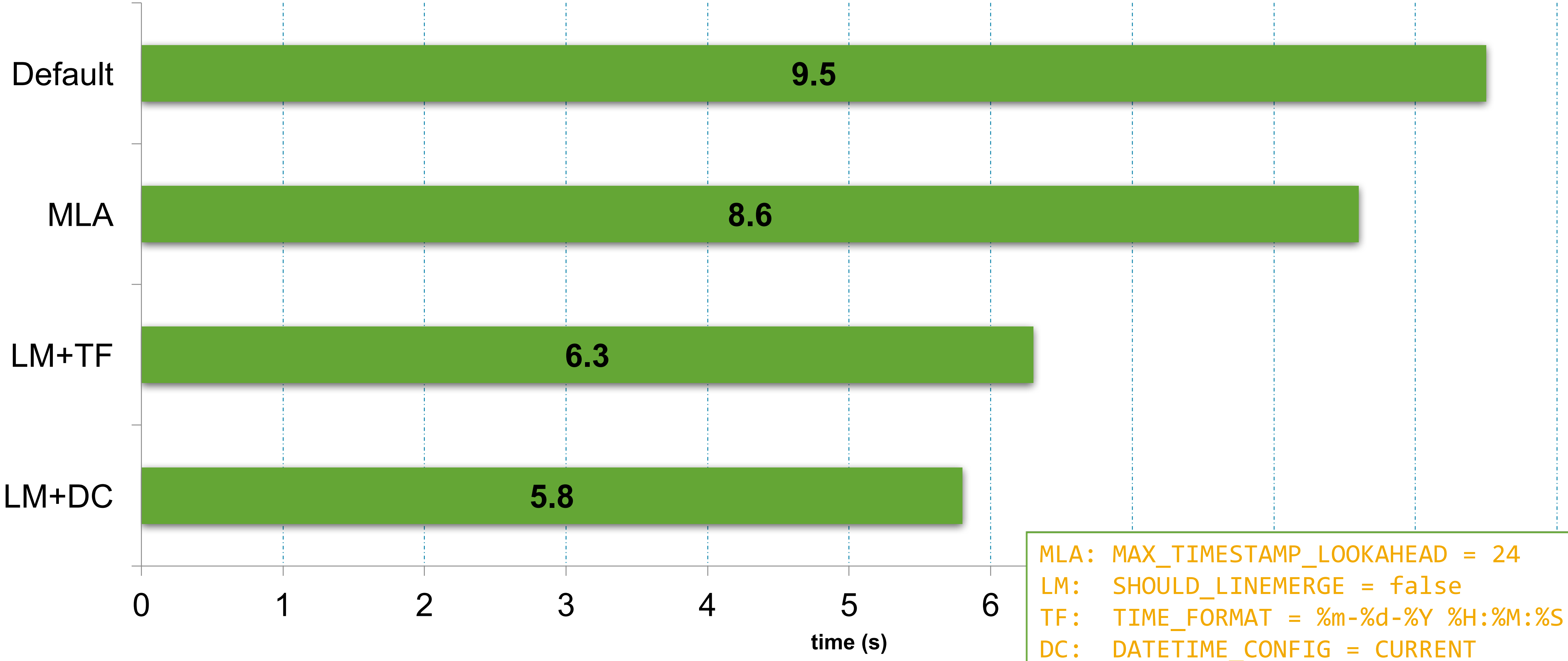
- 10M syslog-like events:

```
. . .
08-24-2016 15:55:39.534 <syslog message >
08-24-2016 15:55:40.921 <syslog message >
08-24-2016 15:55:41.210 <syslog message >
. . .
```

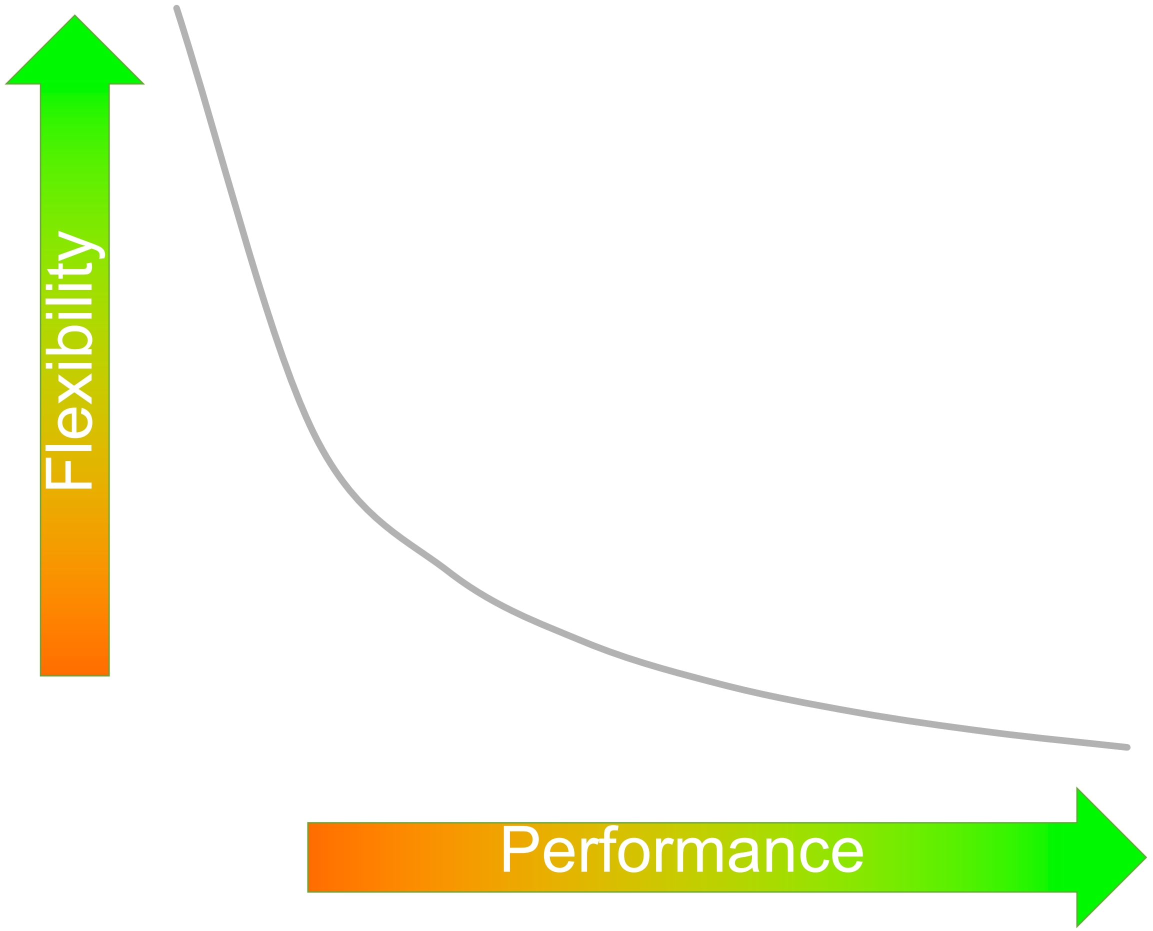
- Push data thru:
  - Parsing > Merging > Typing Pipelines
    - **Skip Indexing**
  - Tweak various props.conf settings
- **Measure**

```
MLA: MAX_TIMESTAMP_LOOKAHEAD = 24
LM:  SHOULD_LINEMERGE = false
TF:  TIME_FORMAT = %m-%d-%Y %H:%M:%S.%3N
DC:  DATETIME_CONFIG = CURRENT
```

Index-time pipeline results



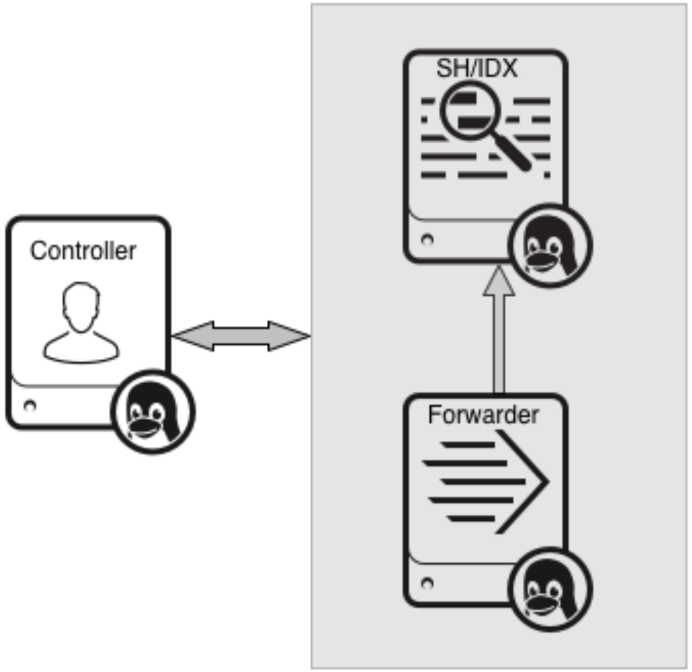

```
MLA: MAX_TIMESTAMP_LOOKAHEAD = 24
LM:  SHOULD_LINEMERGE = false
TF:  TIME_FORMAT = %m-%d-%Y %H:%M:%S.%3N
DC:  DATETIME_CONFIG = CURRENT
```



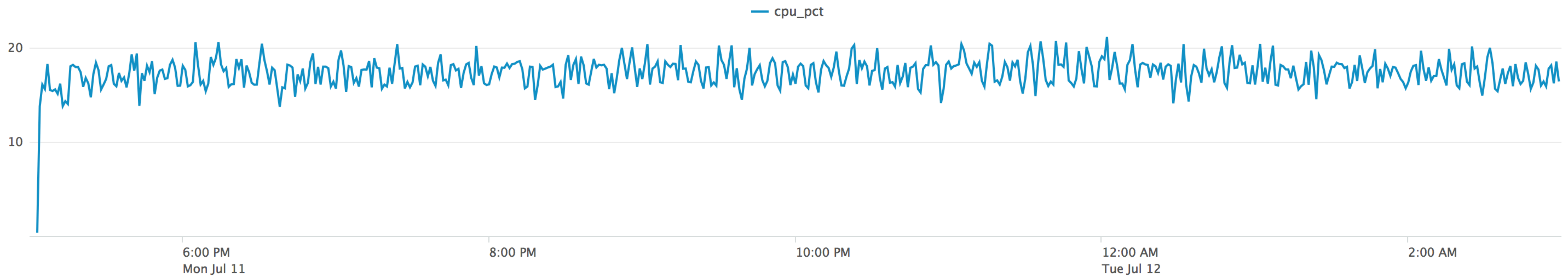
- All pre-indexing pipelines are expensive at default settings.
  - Price of flexibility
- If you're looking for performance, minimize generality
  - `LINE_BREAKER`
  - `SHOULD_LINEMERGE`
  - `MAX_TIMESTAMP_LOOKAHEAD`
  - `TIME_PREFIX`
  - `TIME_FORMAT`

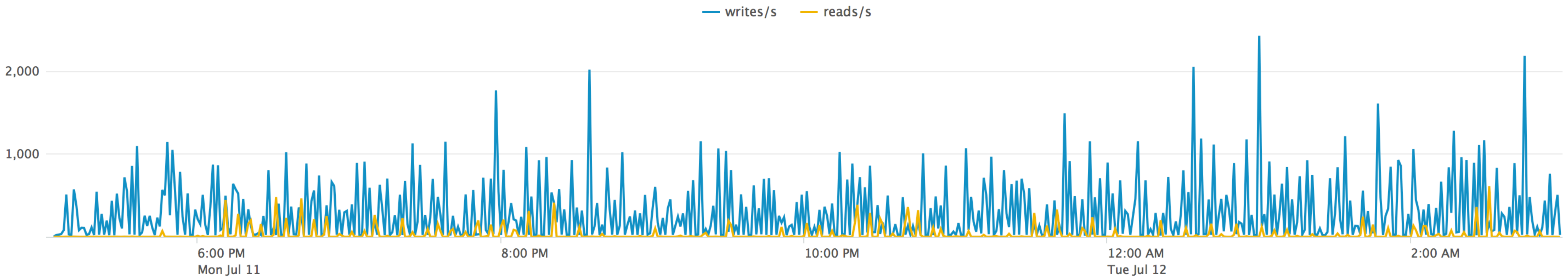
Next: let's index a dataset B

- Generate a much larger dataset (1TB)
  - High cardinality, ~380 Bytes/event, 2.9B events
- Forward to indexer as fast as possible
  - Indexer:
    - Linux 2.6.32 (CentOS);
    - 2x12 Xeon 2.30 GHz (HT enabled)
    - 8x300GB 15k RPM drives in RAID-0
  - No other load on the box
- **Measure**



Indexing: CPU and IO





splunk> .conf2017

Indexing Test Findings

- CPU Utilization
  - **~17.6%** 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
  - **30MB/s**
  - "Speed of Light" – no search load present on the server

Index Pipeline Parallelization

- Splunk 6.3+ introduced 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

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) will not generally improve indexing throughput by meaningful amount
- Faster (**not more**) CPUs would have improved indexing throughput
  - multiple pipelines would need more CPUs

Search

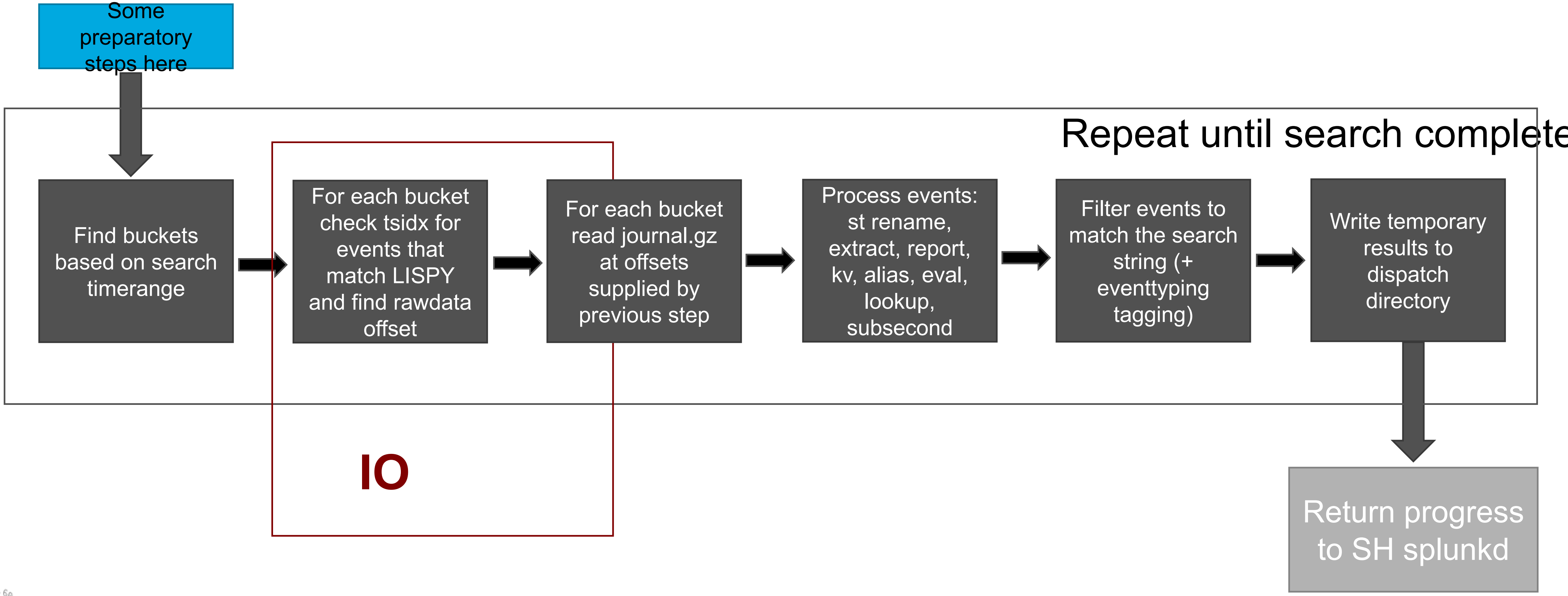
Types & Tests

Searching

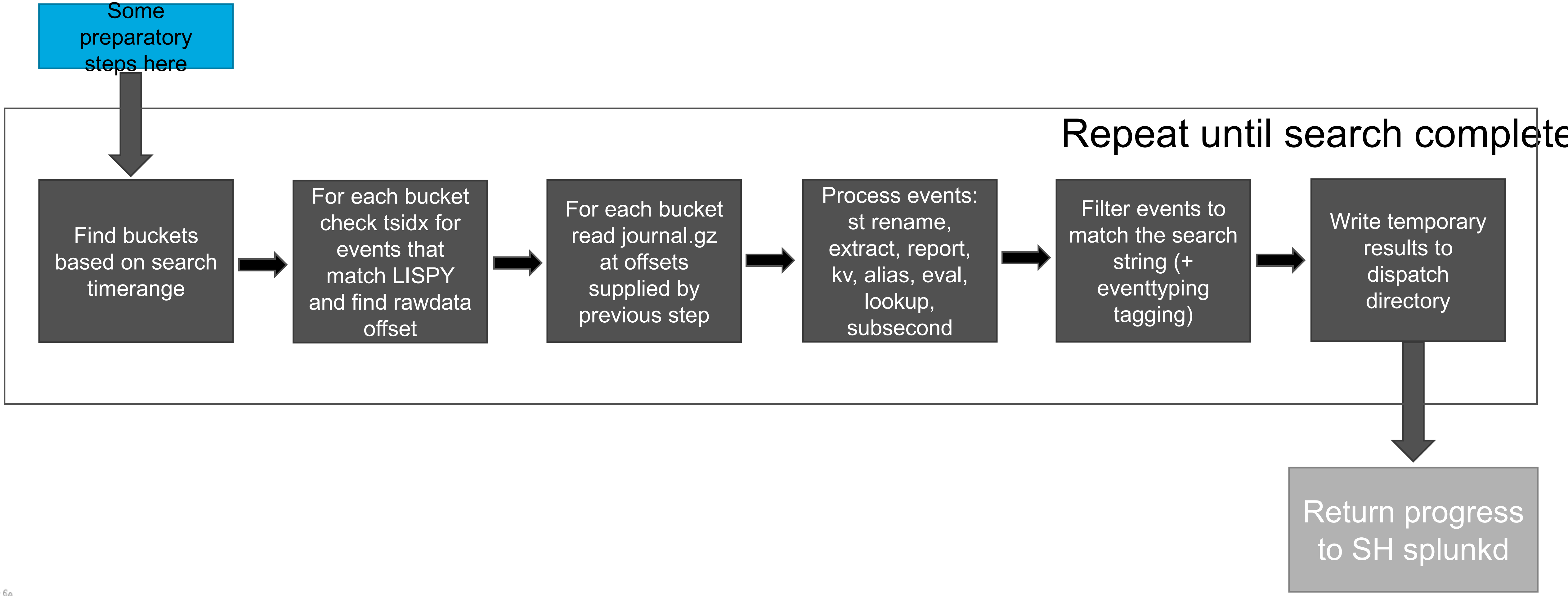
- Real-life search workloads are complex and varied
  - Difficult to encapsulate every organization’s needs into one neat profile

- Yet we can generate arbitrary workloads covering a wide range of resource utilization and profile those
  - Actual profile will fall somewhere in between.

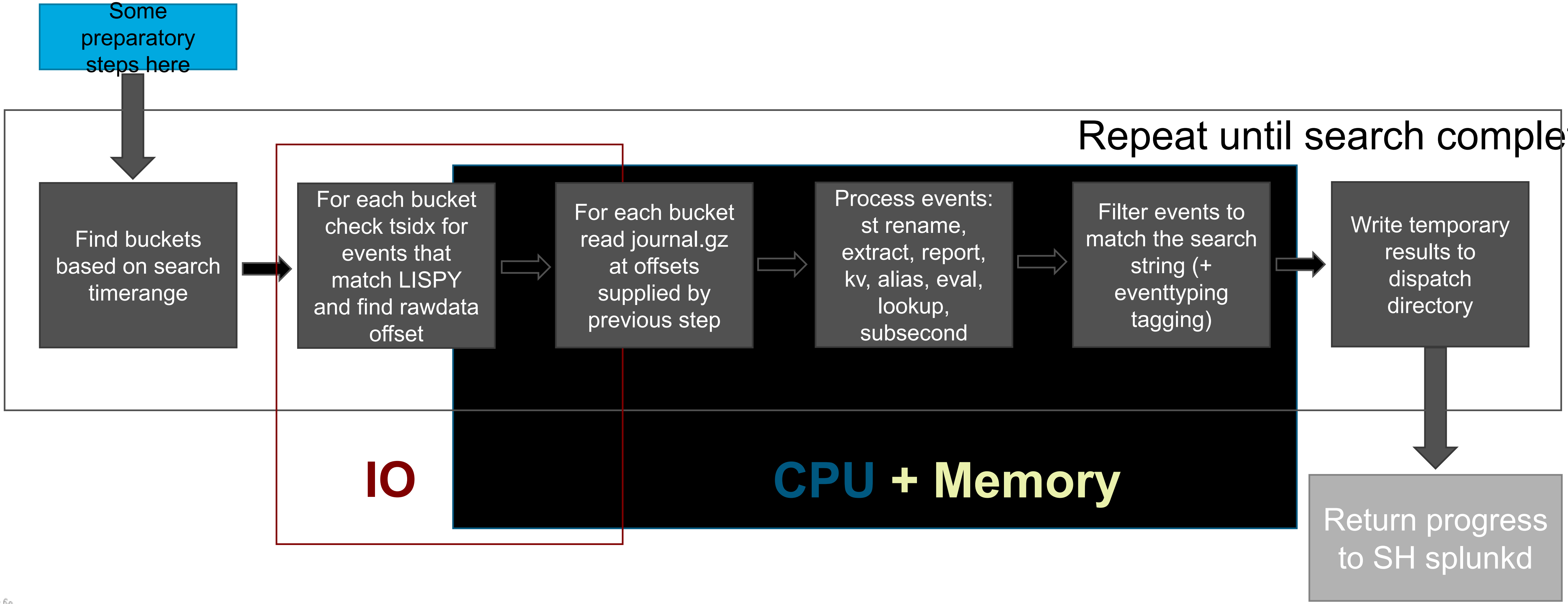
Search pipeline boundedness



Search pipeline (High Level)



Search pipeline boundedness



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**

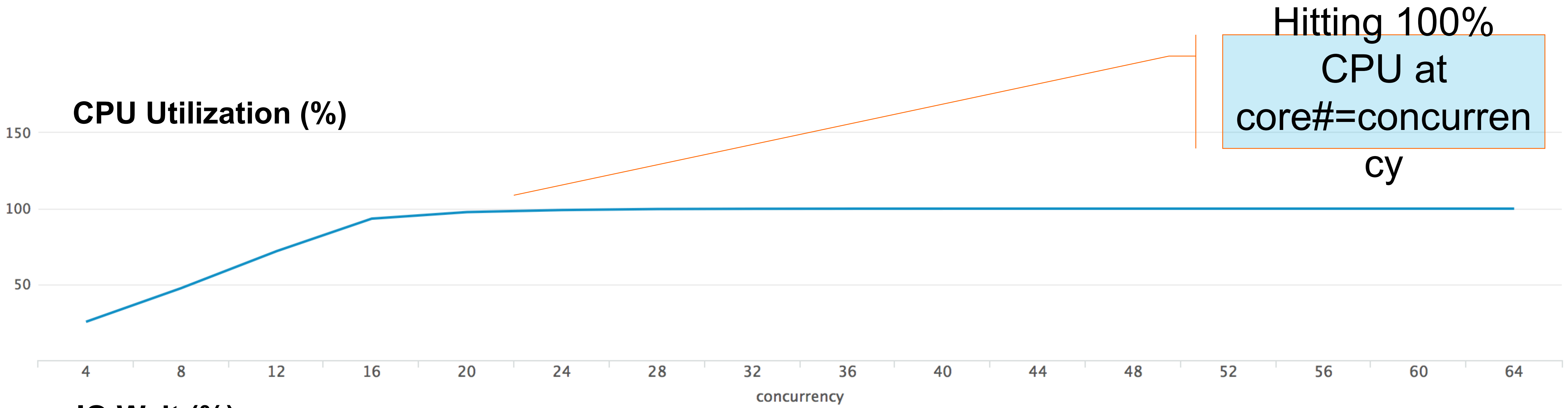
    `index=web some_term | stats count by clientip`
- **Rare**
  - Characterized predominantly by returning **only a few** events per index

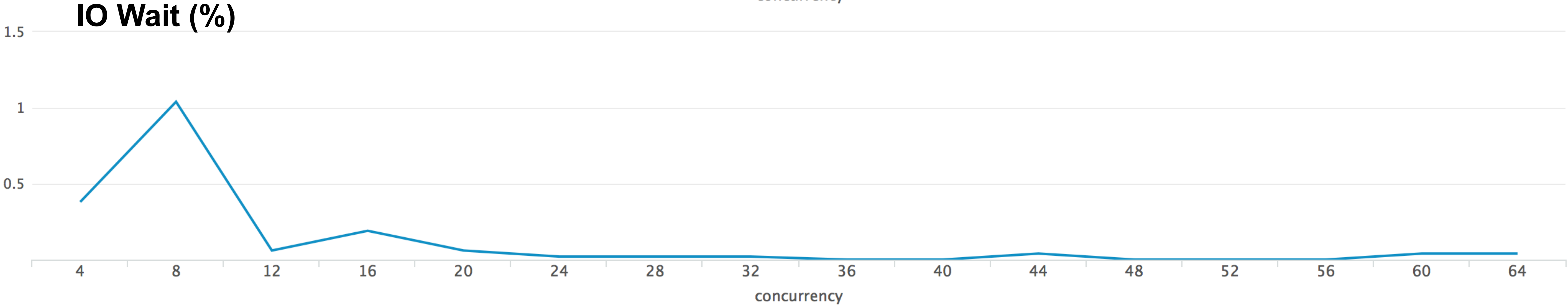
    `index=web url=onedomain* | stats count by clientip`

Okay, let's test some searches

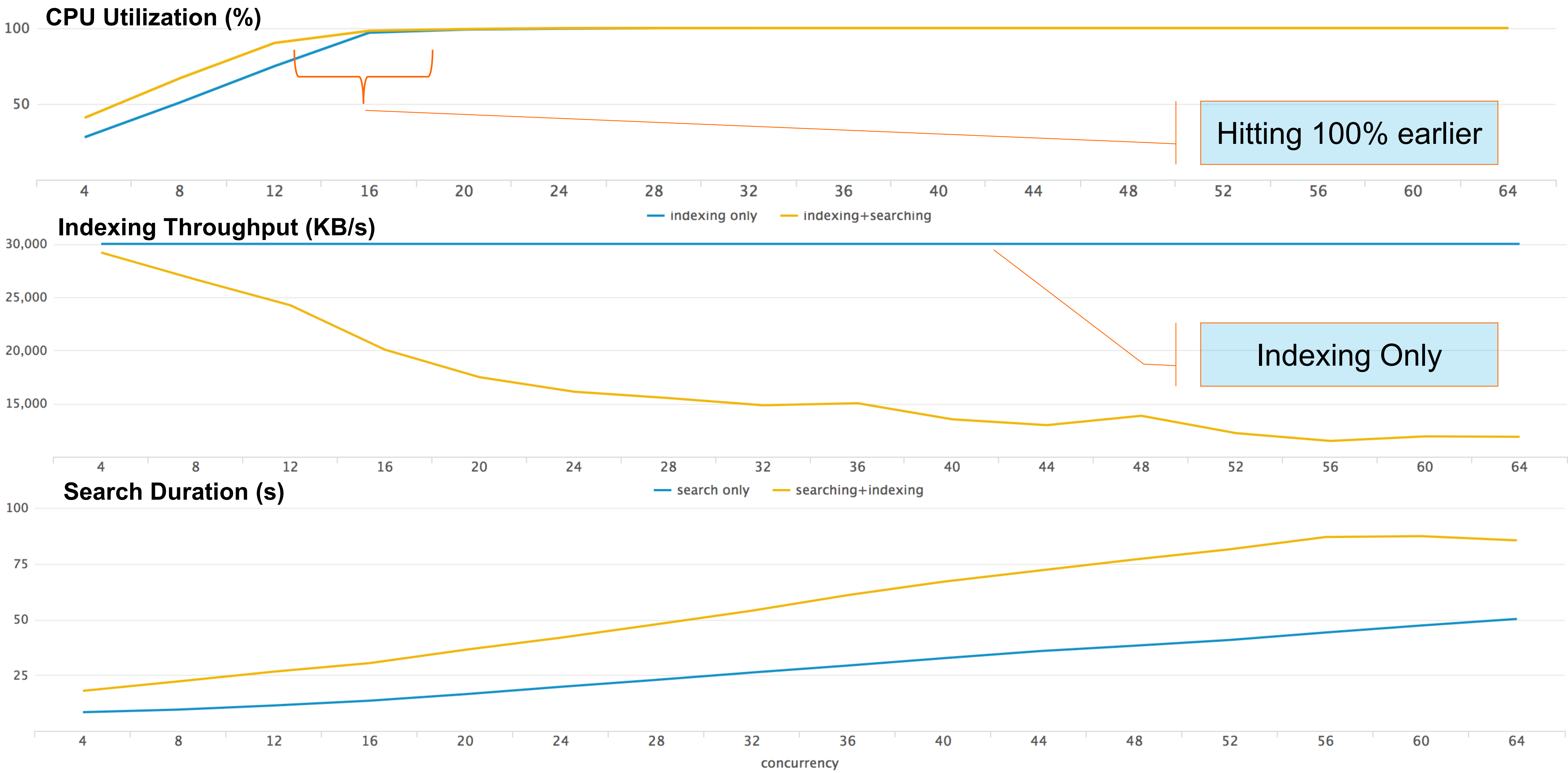
- Use our already indexed data
  - It contains **many** unique terms with predictable term density
- Search under several term densities and concurrencies
  - Term density: 1/100, 1/1M, 1/100M
  - Search Concurrency: 4 – 60
  - Searches:
    - **Rare: over all 1TB dataset**
    - **Dense: over a preselected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

Dense Searches





Indexing with Dense Searches

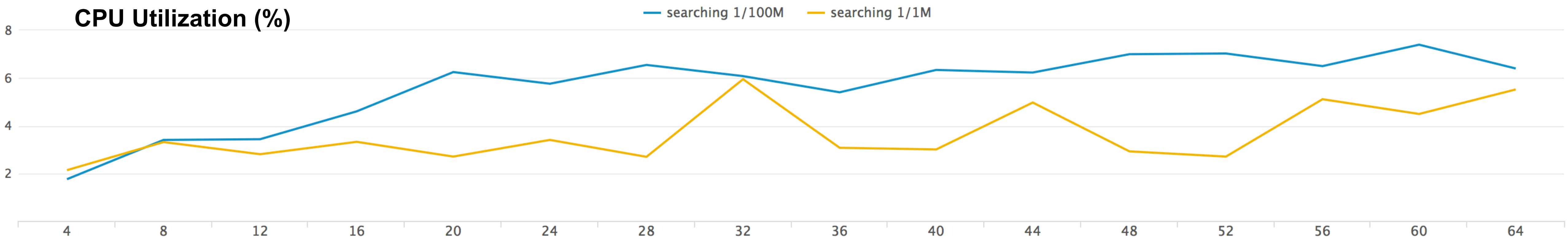


Dense Searches Summary

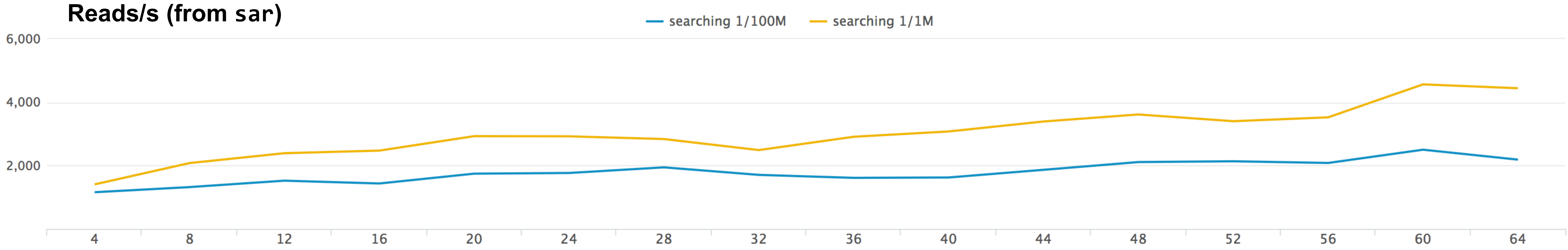
- Dense workloads are CPU bound
- Dense workload completion times and indexing throughput both negatively affected while running simultaneously
- **Faster disk wont necessarily help as much here**
  - Majority of time in dense searches is spent in CPU decompressing rawdata + other SPL processing
- **Faster and more CPUs would have improved overall performance**

Rare Searches

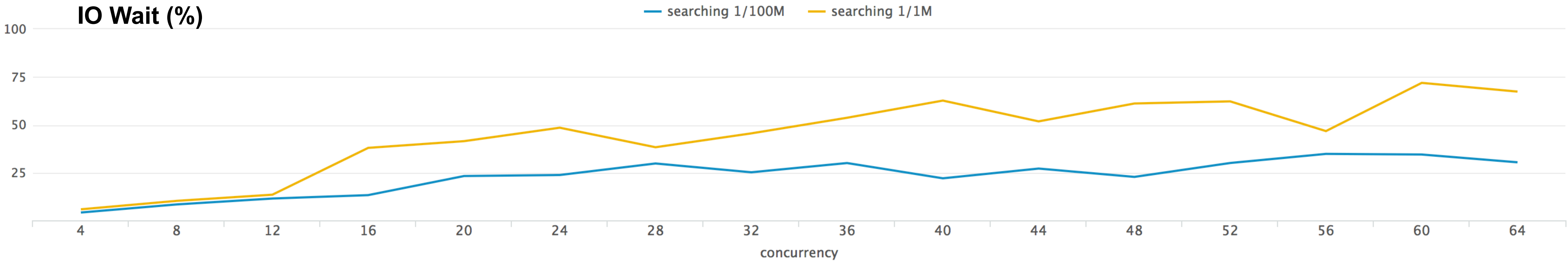
CPU Utilization (%)



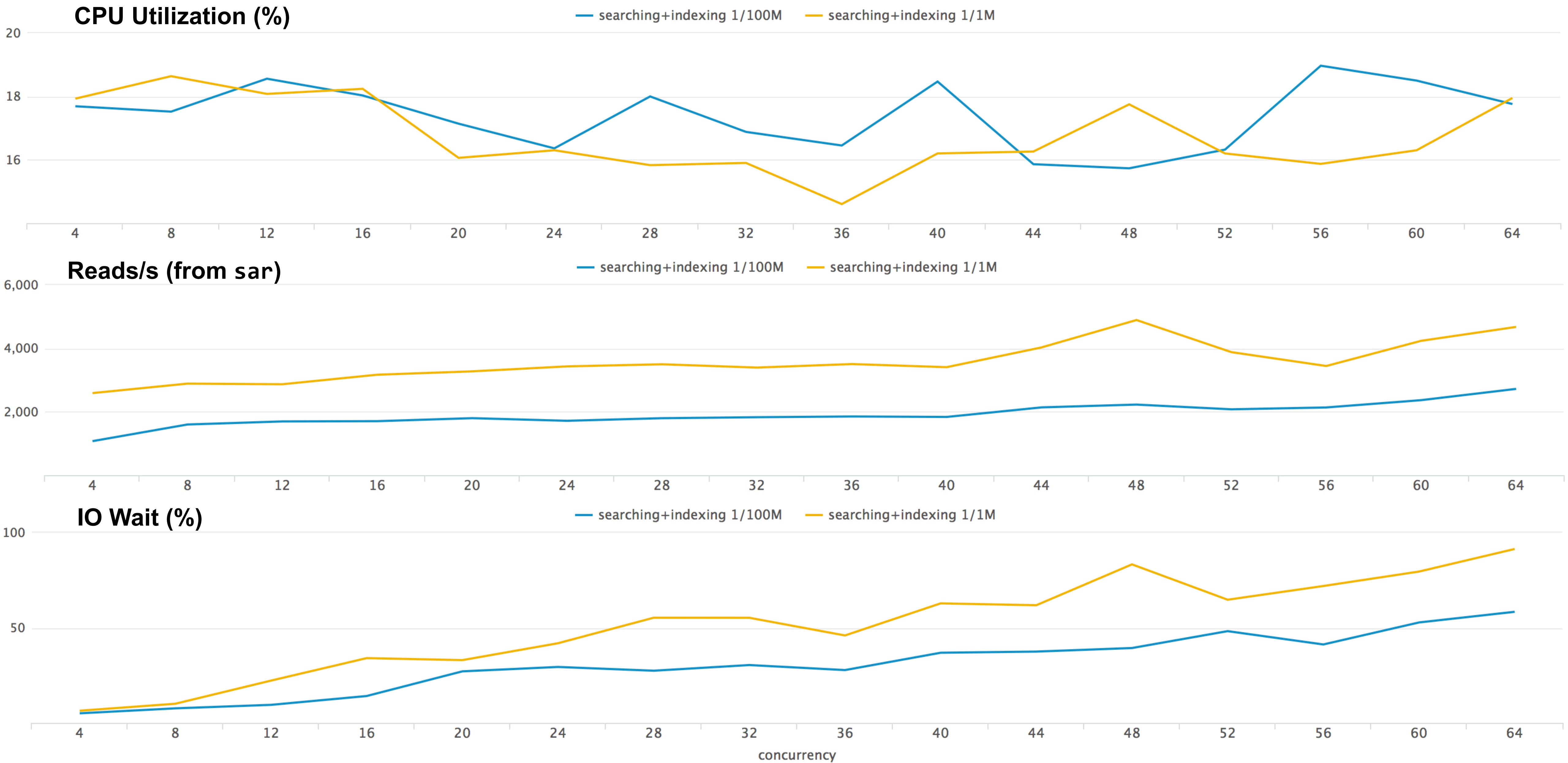
Reads/s (from sar)



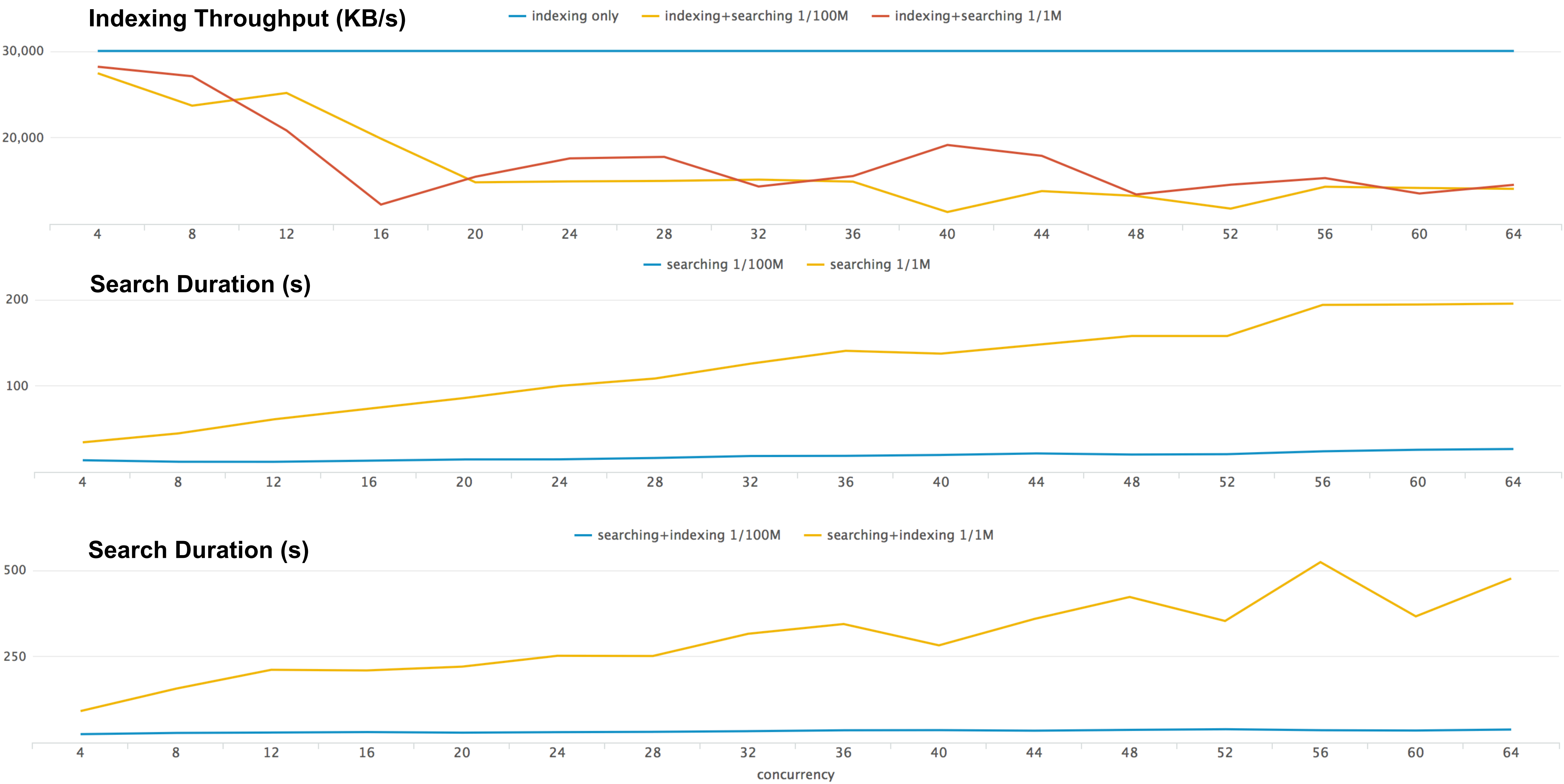
IO Wait (%)



Indexing with Rare Searches



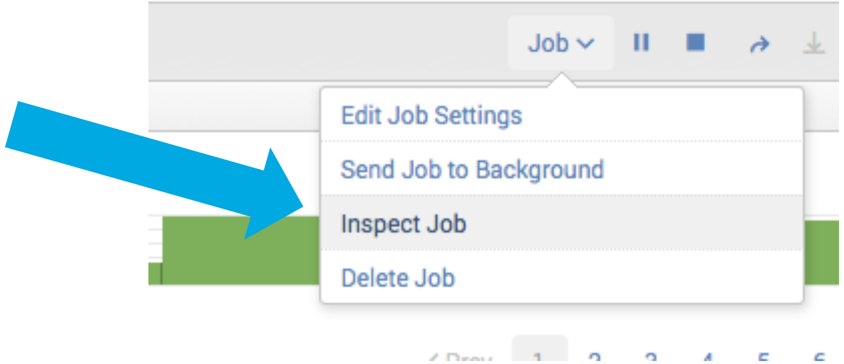
Indexing & Searching Rare

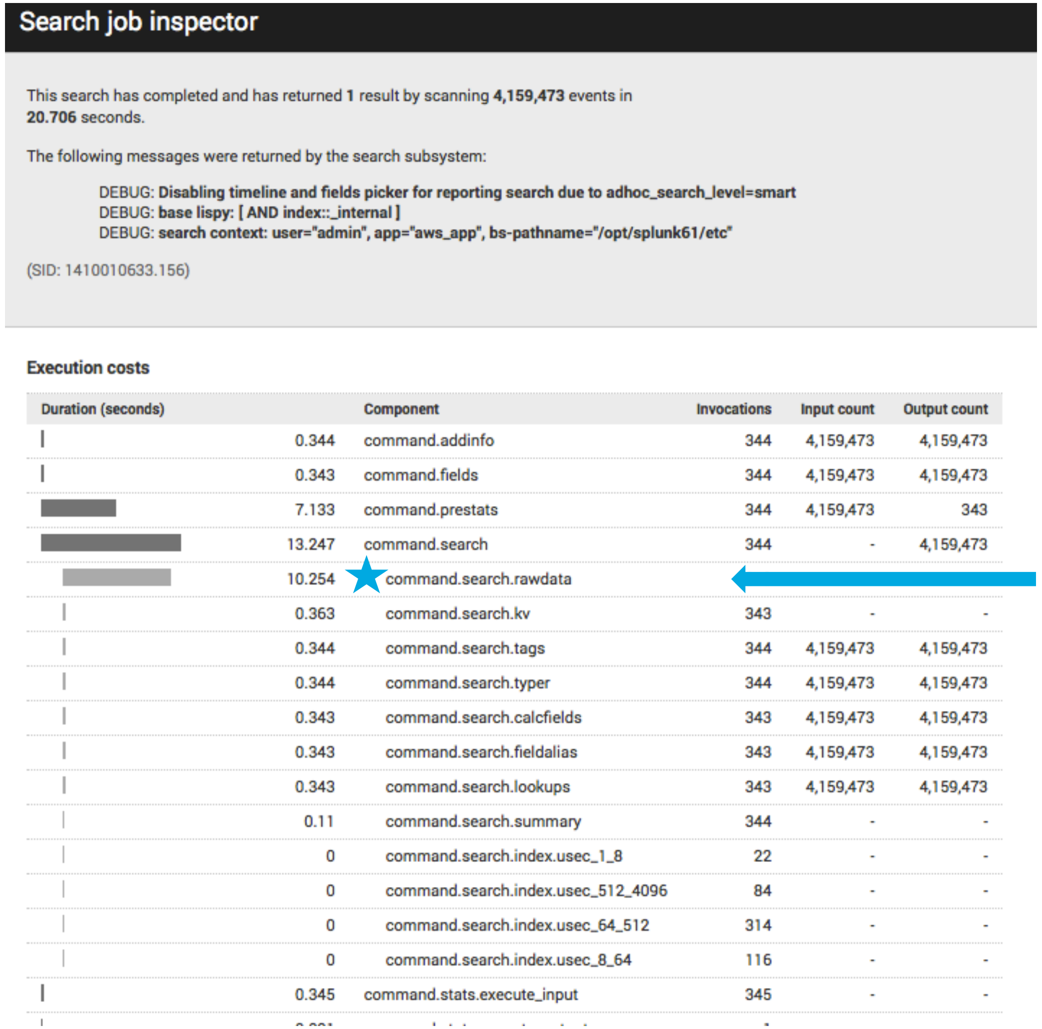


Rare Searches Summary

- Rare workloads (investigative, ad-hoc) are IO bound
- Rare workload completion times and indexing throughput both negatively affected while running simultaneously
- 1/100M searches have a lesser impact on IO than 1/1M.
- When indexing is on, in 1/1M case search duration increases substantially more vs. 1/100M. Search and indexing are both contenting for IO.
- In case of 1/100M, **bloomfilters** help improve search performance
  - ***Bloomfilters*** *are special data structures that indicate with 100% certainty that a term* ***does not exist*** *in a bucket (indicating to the search process to skip that bucket).*
- **Faster disks would have definitely helped here**
- **More CPUs would not have improved performance by much**

Is my search CPU or 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	-	-

Guideline in absence of full instrumentation

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

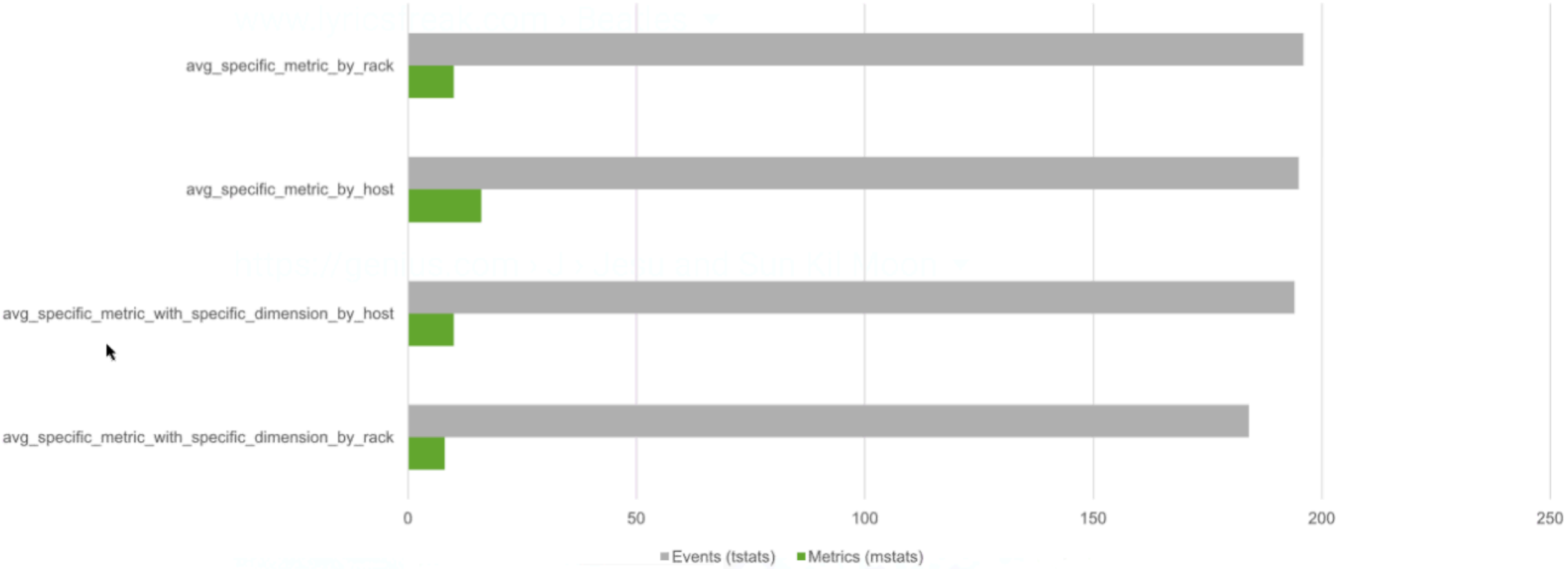
Metric Store

Types & Tests

Metric Store Performance

Query Response Times Metrics vs Events

360M events, 10 hosts, 87 distinct metrics



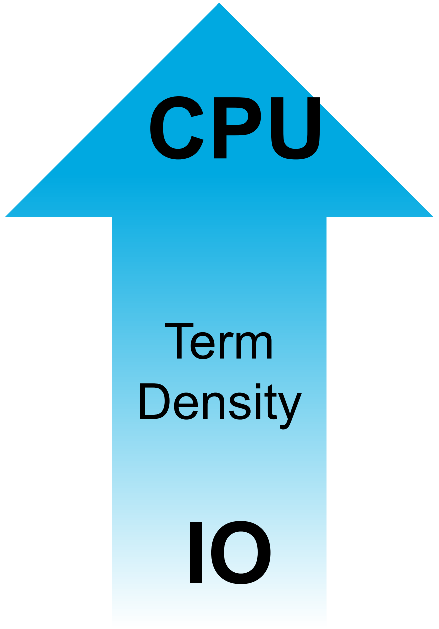
Metric Store Performance

Ingestion

- HTTP Endpoint (AKA HTTP Event Collector, HEC)
  - ~55,000 EPS / indexer sans search load
  - Scales nearly linearly
- UDP
  - Varies
  - 33% packet loss at 10,000 EPS

Top Takeaways

- **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

Testing Disclaimer Reminder

1. Testing conducted on arbitrary datasets
2. “closed course” (lab) environment
3. Not to be interpreted out of context

Q&A

Simeon Yep | AVP GSA

Brian Wooden | Partner Integrations

Thank You

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