

Splunk & Your Infrastructure

Getting to search results faster

Splunk Performance: Making hardware and platform choices



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.

Splunk, Splunk>, Listen to Your Data, The Engine for Machine Data, Splunk Cloud, Splunk Light and SPL are trademarks and registered trademarks of Splunk Inc. in the United States and other countries. All other brand names, product names, or trademarks belong to their respective owners. © 2019 Splunk Inc. All rights reserved.

Splunk as a Racecar

Design / Tune for Your Path



- ▶ Straight line fast
- ▶ Oval Track
- ▶ Rally Course





Simeon Yep

AVP GSA | Splunk, Inc.



Brian Wooden

Director GSA | Splunk, Inc.

Agenda

In This Session

- ✓ ~~Performance & Bottlenecks~~
 - ✓ ~~The Race Car Analogy~~
- ▶ Indexing
 - Index-time Pipelines
 - Indexing Tests
- ▶ Searching
 - Without and With Indexing Load
 - Search Types
 - Mixed Workload Impacts
- ▶ Real World Application

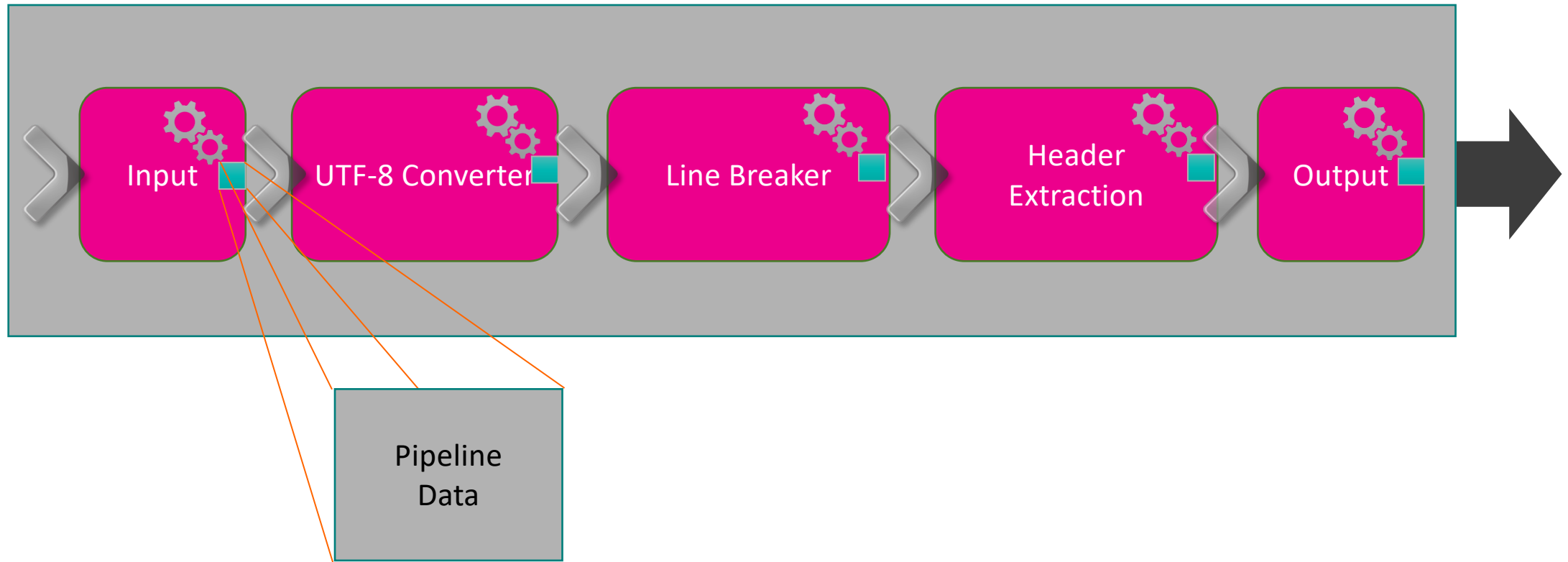
Indexing

- Review Index Tier
- Provide Recommendations



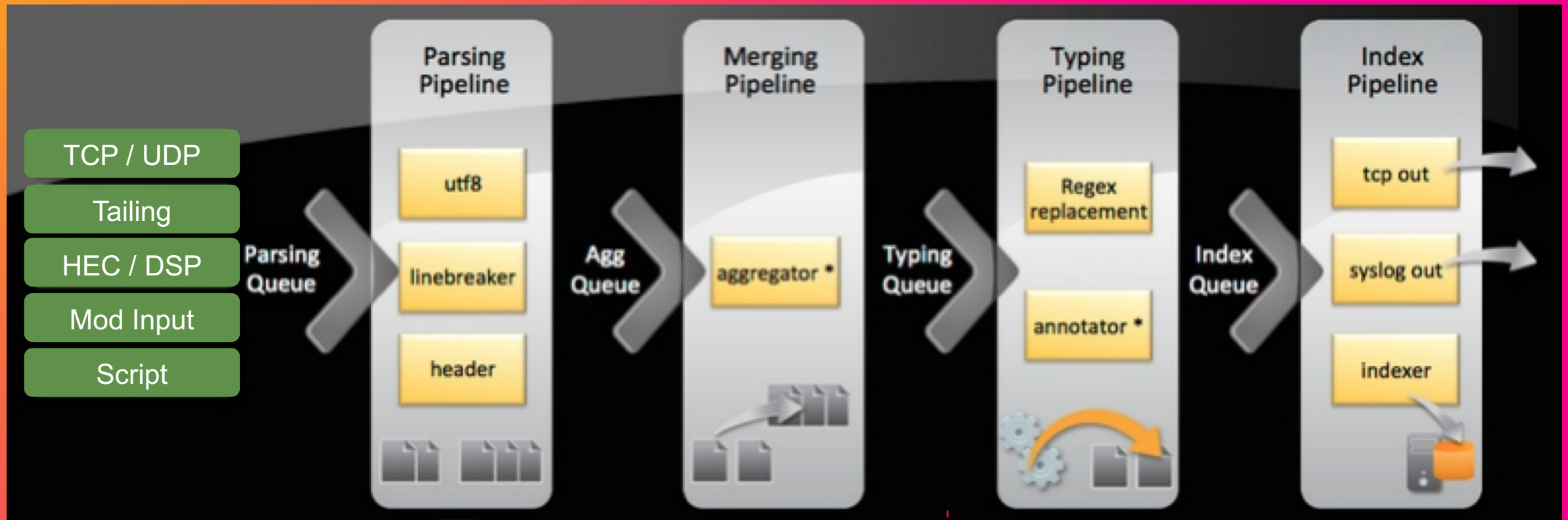
Pipeline Processing

Let it Flow



Splunk data flows thru several such pipelines before it gets indexed

Plethora of Pipelines



LINE_BREAKER
TRUNCATE

SHOULD_LINEMERGE
BREAK_ONLY_BEFORE
MUST_BREAK_AFTER
TIME_*

TRANSFORMS-xxx
SEDCMD
ANNOTATE_PUNCT

Parsing & Indexing

Flexibility vs Performance

Event Breaking

LINE_BREAKER

Where to separate events

SHOULD_LINEMERGE

Enable or disable merging

Timestamp Extraction

MAX_TIMESTAMP_LOOKAHEAD

How far from regex pattern
(TIME_PREFIX) to look for timestamp

TIME_PREFIX

Pattern before timestamp

TIME_FORMAT

strptime format of timestamp to extract

Typing

ANNOTATE_PUNCT

Enable or disable punct:: field
creation

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 >  
. . .
```

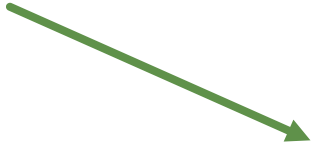
➤ Push data thru:

Parsing > Merging > Typing Pipelines

▪ Skip **Indexing**

Tweak various props.conf settings

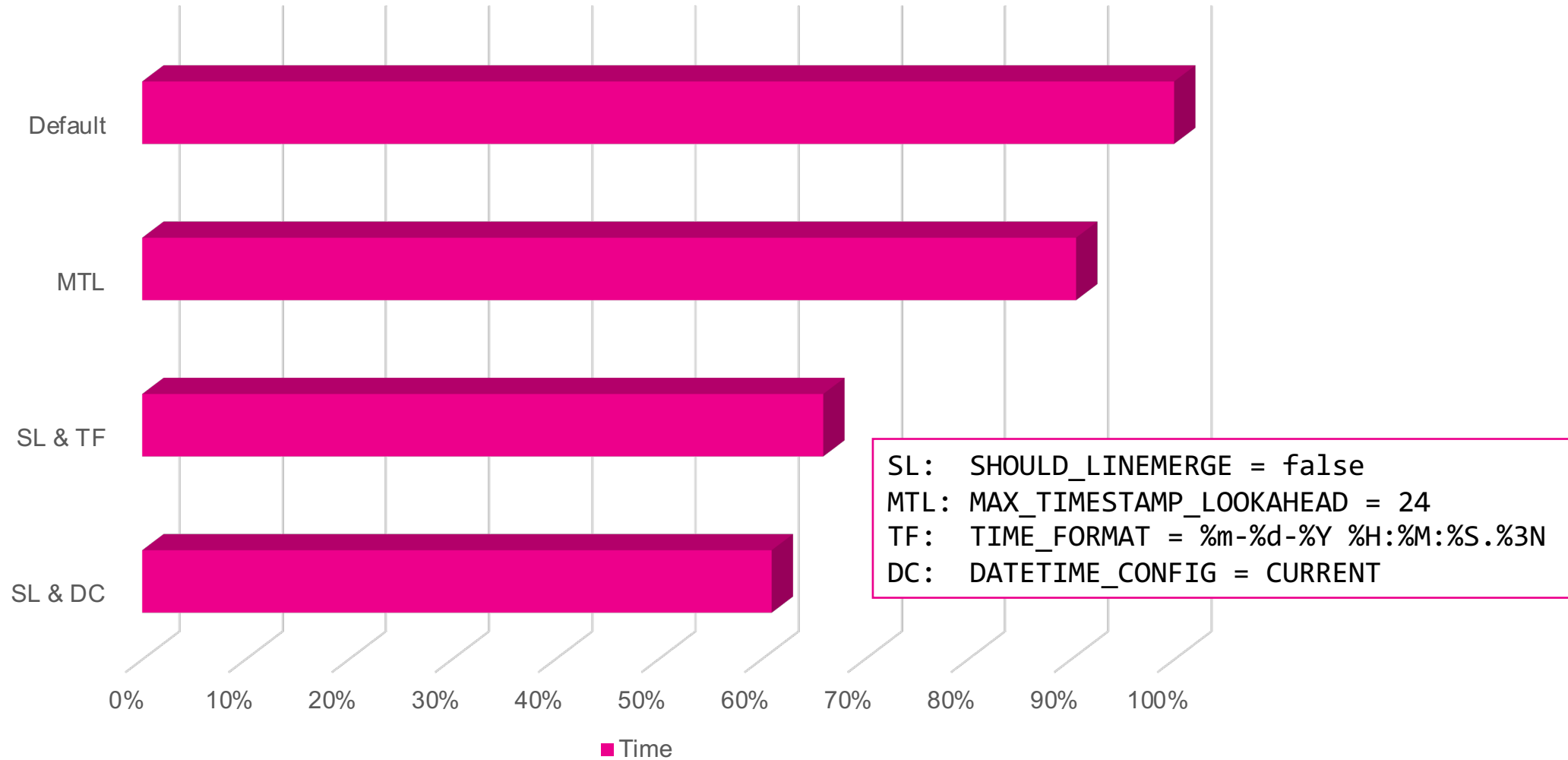
➤ **Measure**



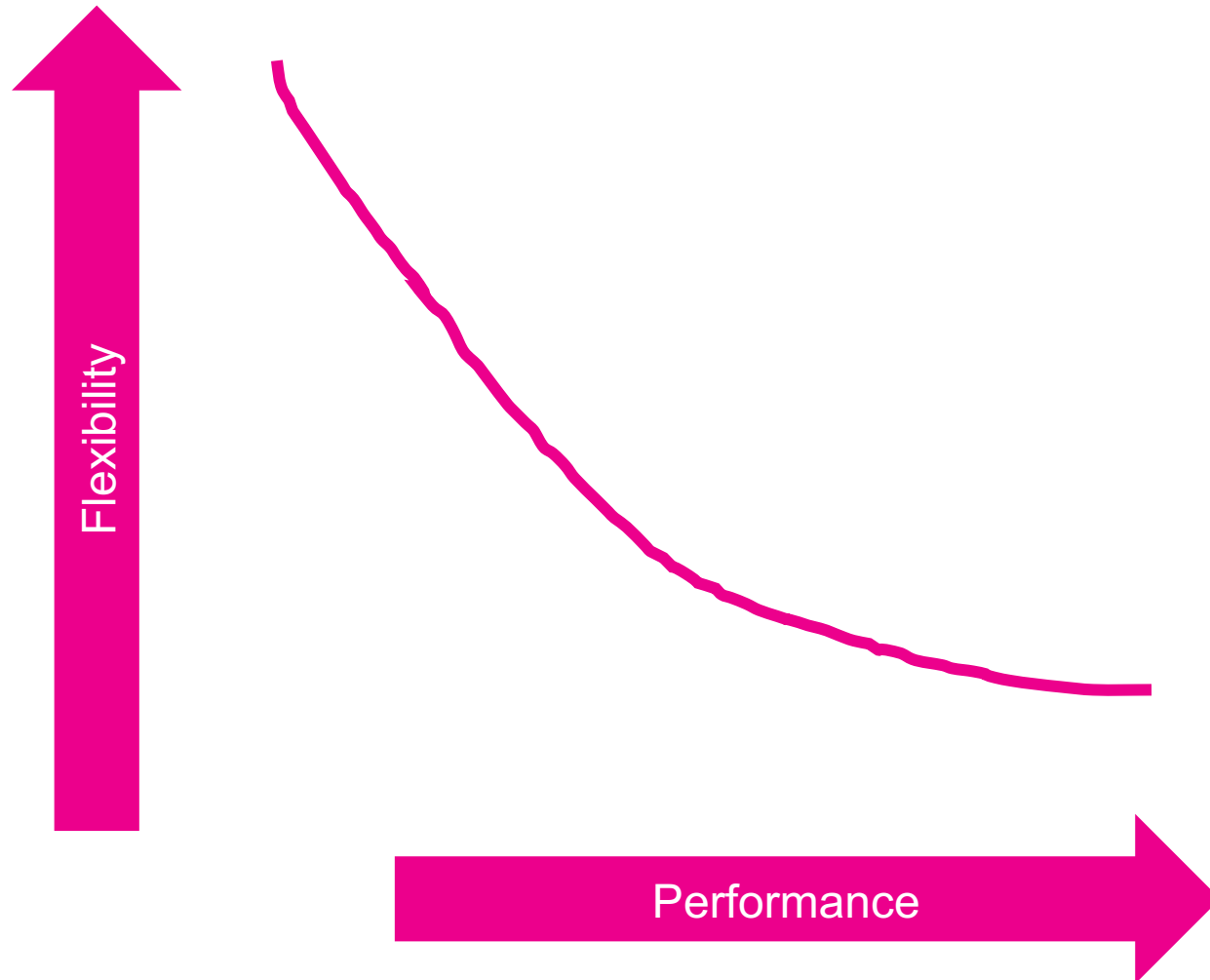
```
LM:  SHOULD_LINEMERGE = false  
MLA: MAX_TIMESTAMP_LOOKAHEAD = 24  
TF:  TIME_FORMAT = %m-%d-%Y %H:%M:%S.%3N  
DC:  DATETIME_CONFIG = CURRENT
```

Parsing Pipeline Workloads

Relative to Default



Flexibility Trade-Off

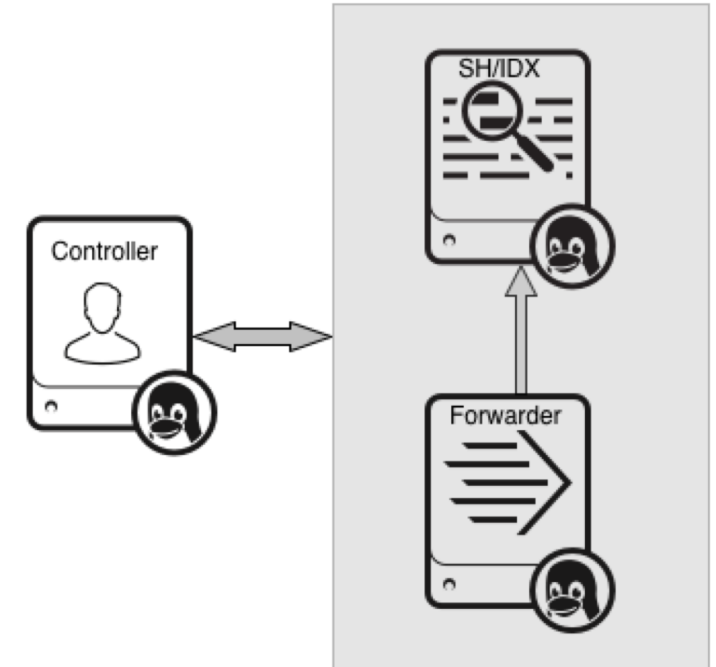


- All pre-indexing pipelines are expensive at default settings
Price of flexibility
- If you're looking for performance, minimize generality via props.conf's

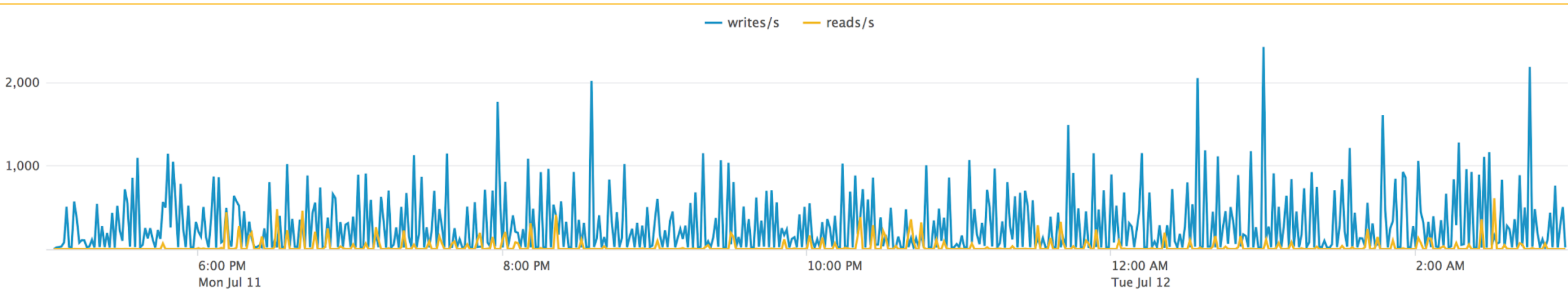
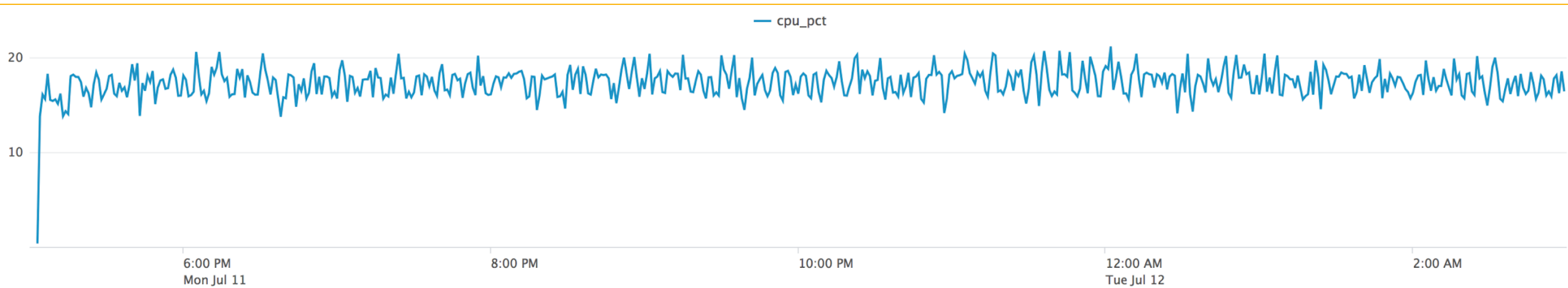
LINE_BREAKER
SHOULD_LINEMERGE
MAX_TIMESTAMP_LOOKAHEAD
TIME_PREFIX
TIME_FORMAT

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-0No other load on the box
- **Measure**



Indexing: CPU and IO



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

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
- **S2 behaves the same for INDEXING**

Searching

Under the Hood



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**

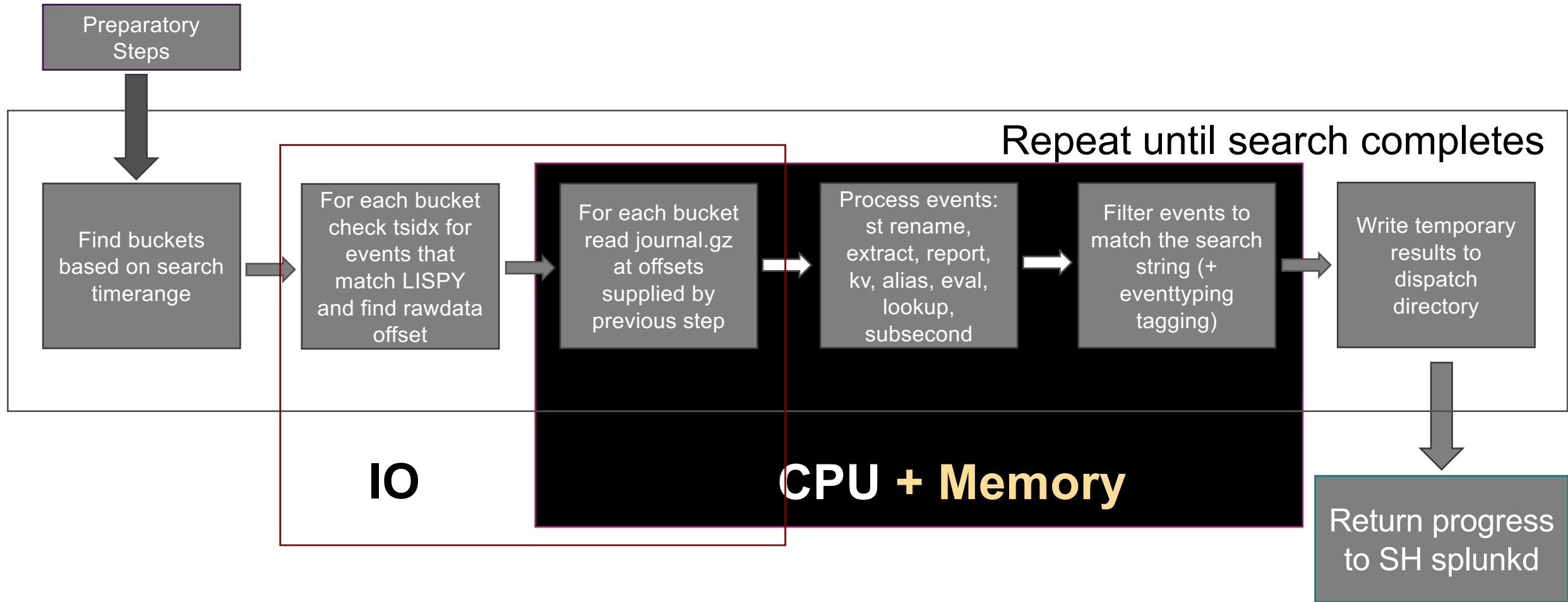
- Characterized predominantly by returning **only a few** events per index
`index=web url=onedomain* | stats count by clientip`

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.

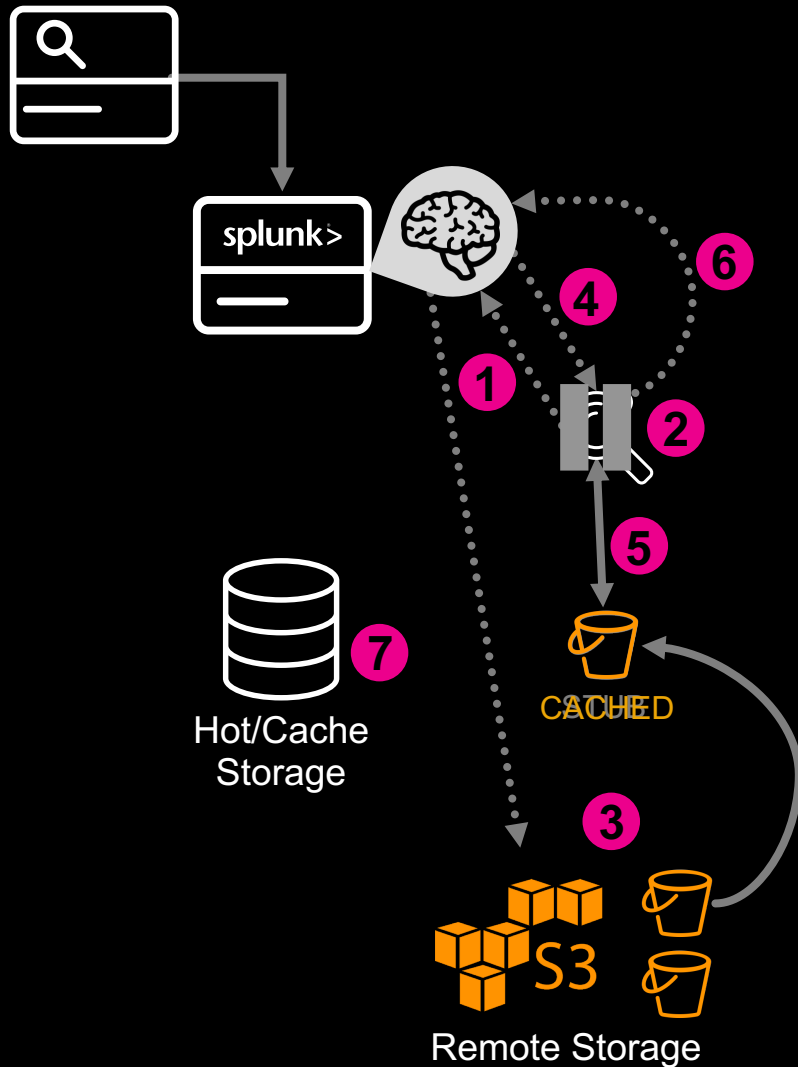


Search pipeline boundedness



Searching with S2

Remote Buckets



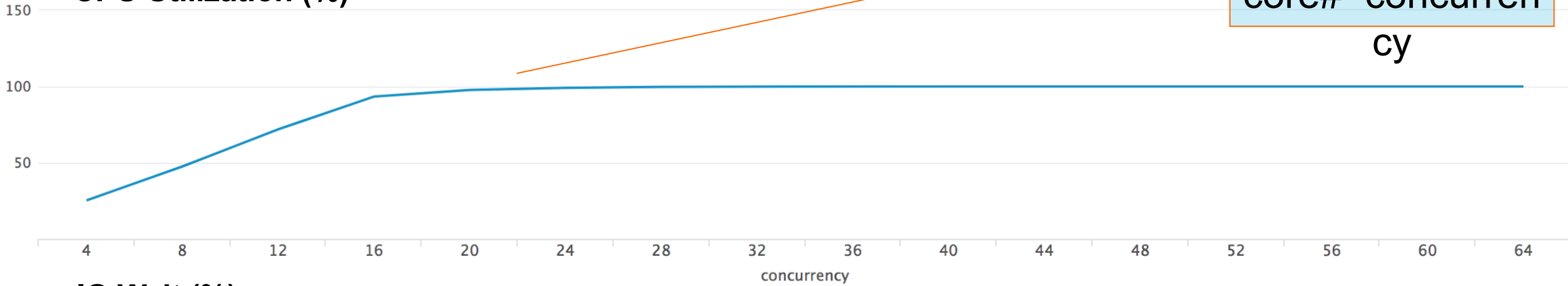
1. Search process "opens" the bucket with the Cache manager, but it isn't in cache
2. Search process waits
3. Cache manager fetches the bucket from the remote store
4. Cache manager tells the search process that the bucket is local and available for search
5. Search process searches the bucket
6. Search process "closes" the bucket with the cache manager
7. Bucket remains in cache until evicted by the cache manager

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**

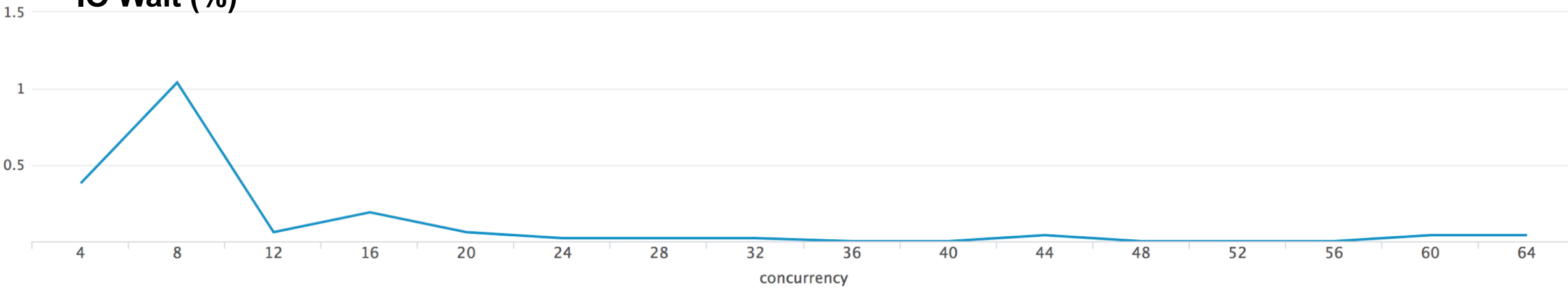
Dense Searches

CPU Utilization (%)

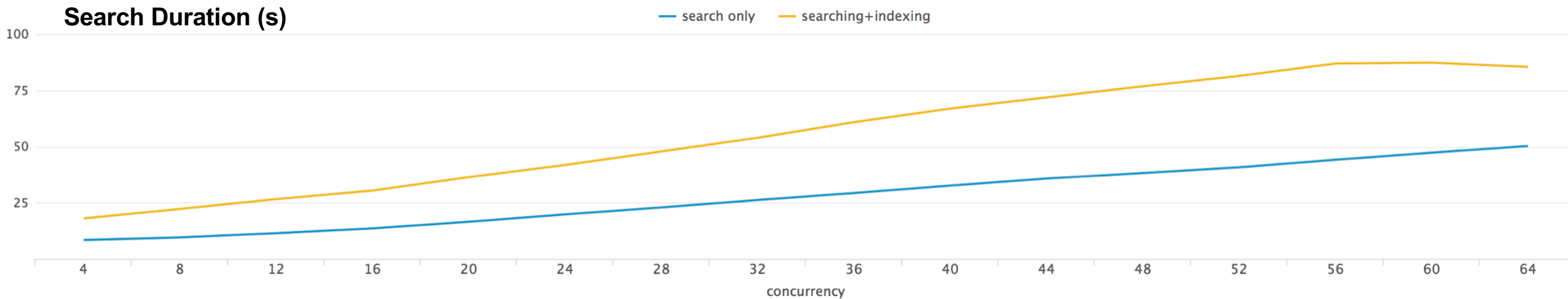
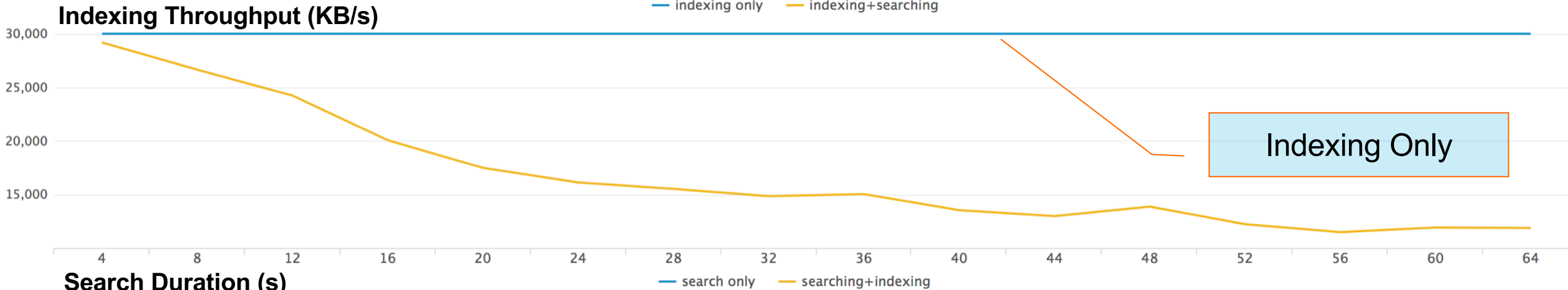
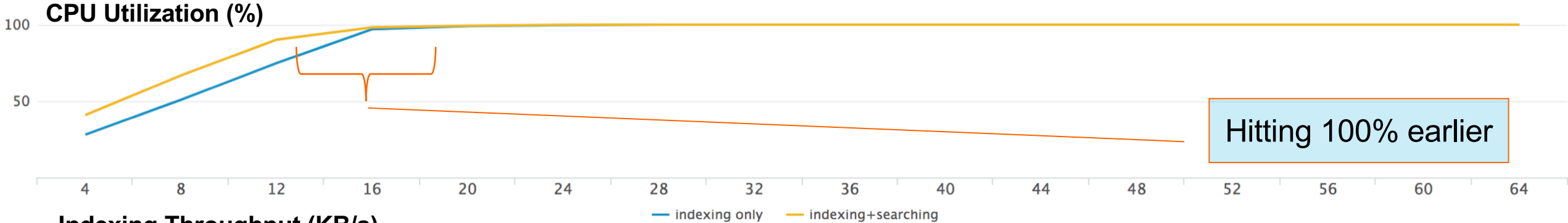


Hitting 100%
CPU at
core#=concurrent
cy

IO Wait (%)



Indexing with Dense Searches



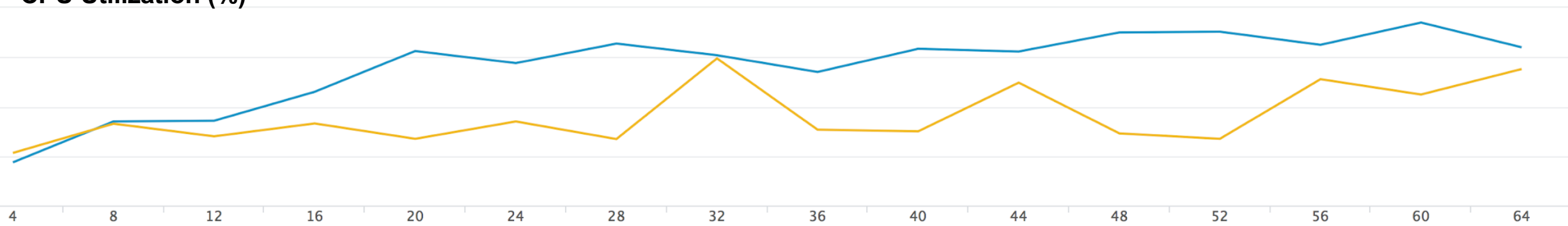
Dense Searches Summary

- 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 and Sparse Searches

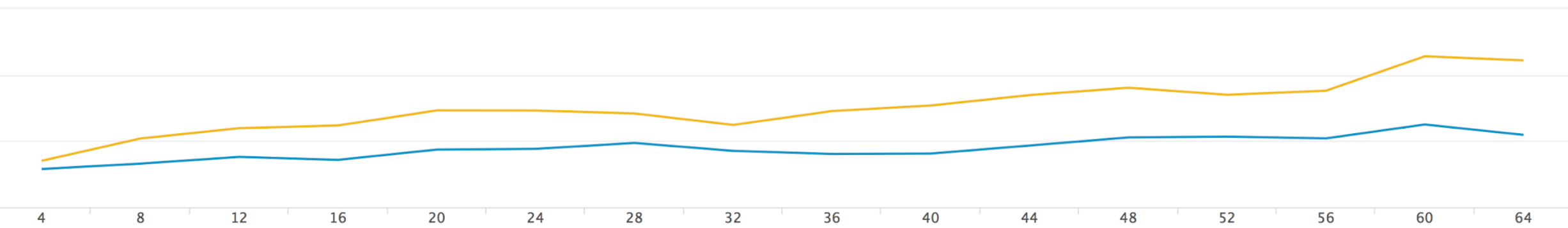
CPU Utilization (%)

searching 1/100M searching 1/1M



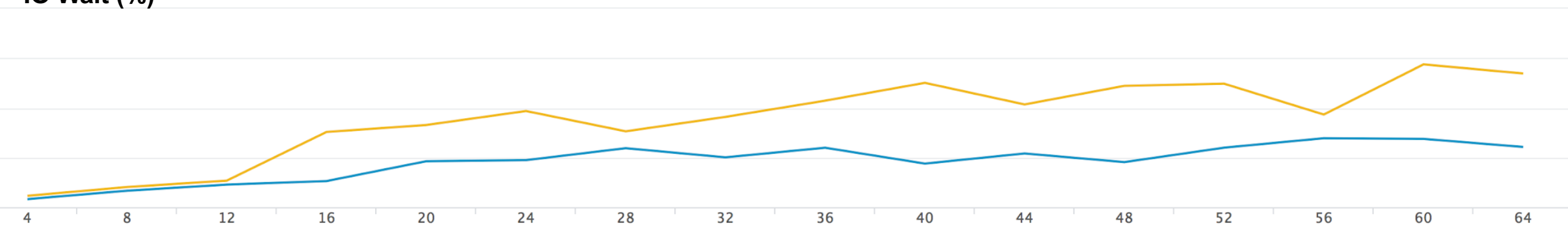
Reads/s (from sar)

searching 1/100M searching 1/1M



IO Wait (%)

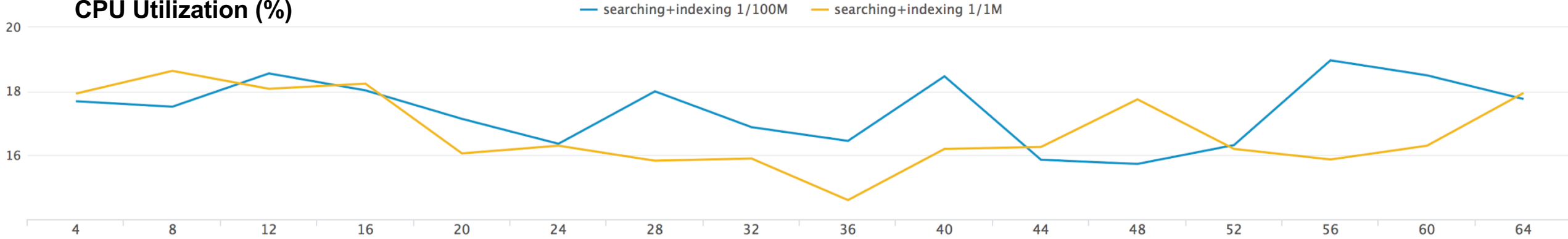
searching 1/100M searching 1/1M



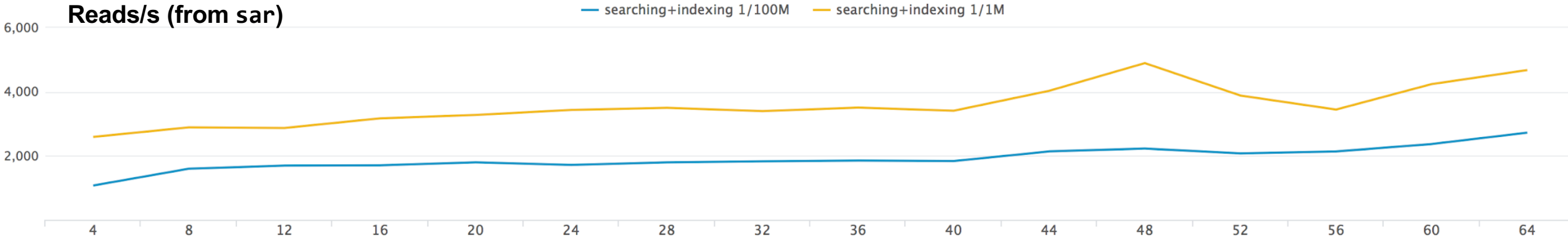
concurrency

Indexing with Rare and Sparse Searches

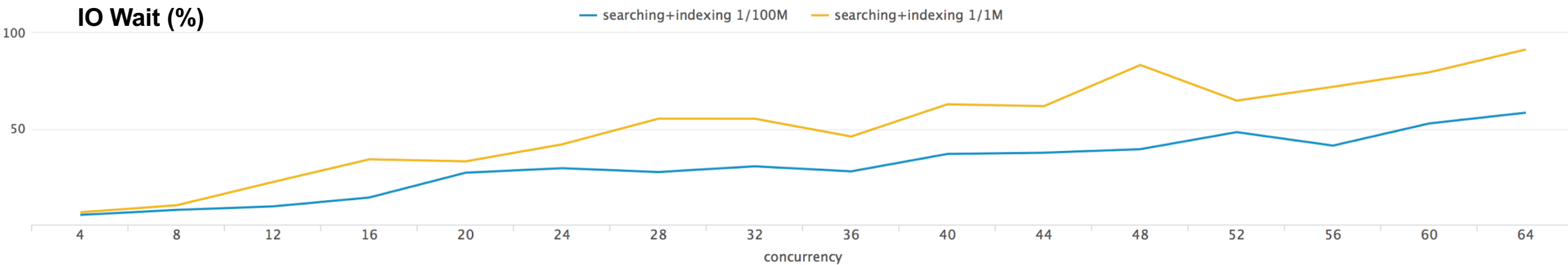
CPU Utilization (%)



Reads/s (from sar)



IO Wait (%)

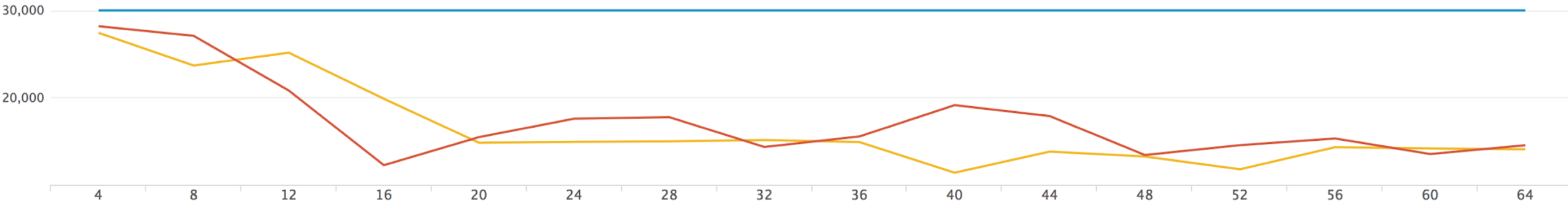


Functional Comparison Rare + Sparse

© 2019 SPLUNK INC.

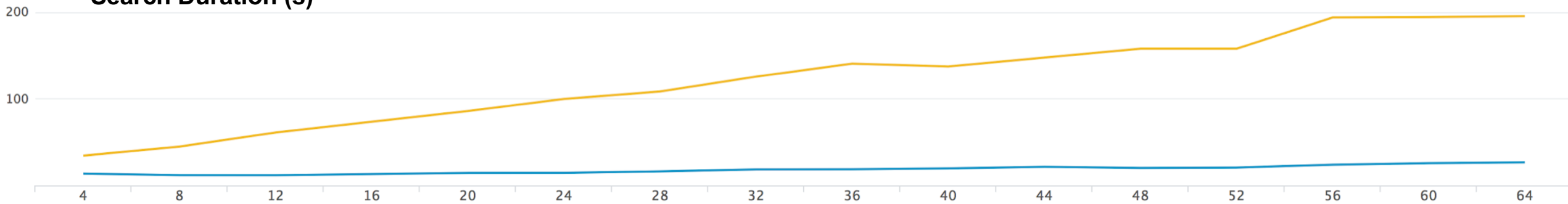
Indexing Throughput (KB/s)

indexing only indexing+searching 1/100M indexing+searching 1/1M



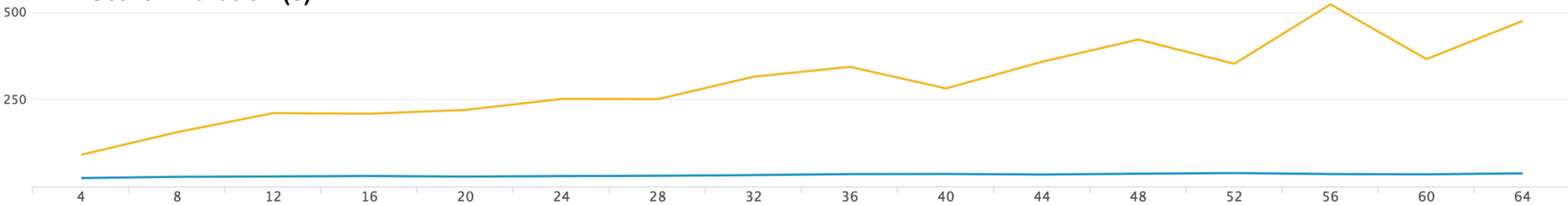
Search Duration (s)

searching 1/100M searching 1/1M



Search Duration (s)

searching+indexing 1/100M searching+indexing 1/1M

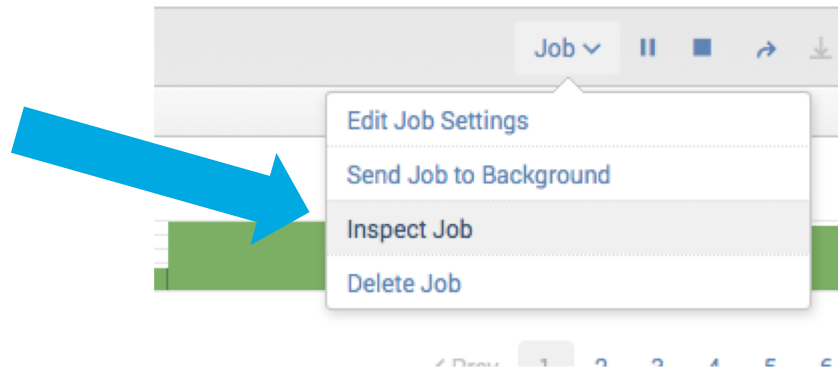


concurrency

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?



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 lisp: [ AND index::internal ]
DEBUG: search context: user="admin", app="aws_app", bs-pathname="/opt/splunk61/etc"
```

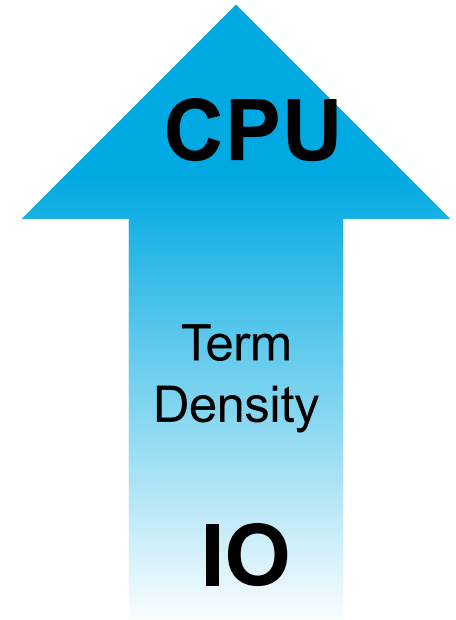
(SID: 1410010633.156)

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	-	-

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

Real World Application

Where The Rubber Meets The Road

- ▶ Thought Process
- ▶ Apply Thought Process to Examples
 - New Deployment
 - Expansion / Upgrade
 - Cloud Providers (BYOL)
- ▶ Making it Easy

Thought Process

Building Is Not Always Best

Theory

- ▶ Understand all use-cases and workloads to calculate requirements
- ▶ Use workload formula to convert workloads/use-cases into resource requirements
- ▶ Follow Splunk guidance for reference hardware
- ▶ Follow Splunk guidance for using premium apps

Reality

- ▶ Difficult to account for everything
 - ▶ Splunk typically grows it's footprint and use-cases
- ▶ Resource Requirements don't convert into what you actually buy
- ▶ Should have followed guidance
- ▶ Should have followed guidance again

New Deployment

Step 1: Prioritize requirements (retention, performance, high availability, disaster recovery, and costs)

Step 2: Consider priorities & leverage [Splunk Validated Architecture](#) (SVA) designs to determine topology

Step 3: Create Hardware to Topology mappings that support business priorities



SVA

Rules of Thumb

Start with indexing workload hardware guidelines (125-325 GB/Day)

- Search/Use-Case workload is your primary concern
- Indexing is probably not going to be your bottleneck
- Because Racecar >>> Because Moore's Law
- Don't compare apples and oranges

New Deployment

Example

Requirements:

1 TB/day, High Availability (HA), 1 yr retention, Security Use Case with high search performance requirements

Step 1 Prioritize requirements (retention, performance, high availability, disaster recovery, and costs st)

1st = HA, 2nd = performance, 3rd = retention, 4th cost

Step 2 Considering Priorities, leverage Splunk Validated Architecture designs to determine topology...

Need multi-site cluster to satisfy HA (double planned amount of indexers)

New Deployment

Example Continued

Step 3 Create hardware to topology mappings that fit priorities

16 indexers (125 gb/day/indexer), 6 search heads, 2 admin servers, object storage

NB: Maximize SSDs on Indexers (this will directly drive search speed for recent searches)

NB: Purchase highest compute available (ignore hyperthreading as additional compute capacity) for both indexers and search heads

Example

Indexer: 36 core; 192 GB RAM; 12 TB

Search Head: 36 core; 192 GB RAM; 1 TB

Admin Server: 36 core; 192 GB RAM; 1 TB

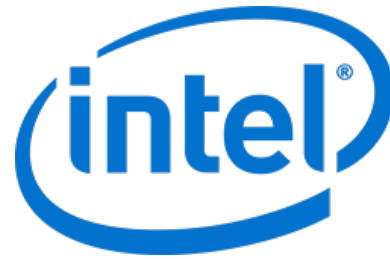
16 Indexers

6 Search Heads

2 Admin Servers

Partner Contributions

Making it Easy.....



Key Takeaways

Failing to Plan
is Plan for Failing

“Plan to Plan”
—The Bobs

1. Performance > Flexibility
2. Plan for Use Cases
3. Leverage Splunk Validated Architectures and Capacity Planning Guides



**Thank
You!**

Go to the .conf19 mobile app to

RATE THIS SESSION