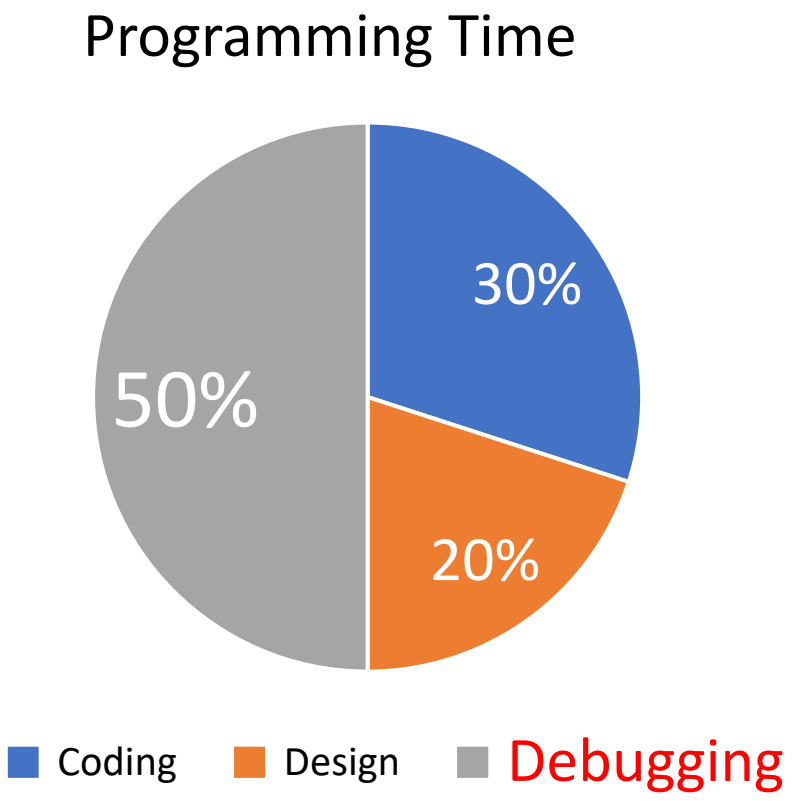
Resource Efficient Observability at Scale

Real-world Deployments in Data Center and Smartphones

Ding Yuan

YScope

The Cost of Debugging



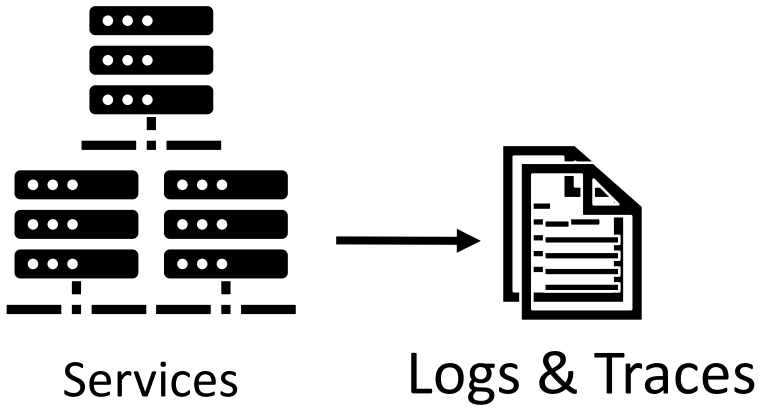
[Britton et al. 2013]

Debugging Production Failures Require Runtime Data

- No Data, Can’t Trouble-shoot
  - Debugging in the Dark

- Can’t use debuggers!

Engineers rely on Logs and Traces



	Who	What	Where
Logs	Programmers	Domain-specific	Source
Traces	Tools	Generic	External

Example log message Timestamp Variables Static text

2020-01-02T03:04:05.006 INFO Task task_12 assigned to container:
[NodeAddress:172.128.0.41, ContainerID:container_15], operation took 0.335 seconds

```
log.info(”Task” + task_id + “ assigned to container…”);
```

Challenge: *Resource Efficiency*

- Requires < 3% of overhead
  - Both CPU & memory

- Internet companies generate Petabytes of logs
  - Annual storage cost: $50 million

This Talk: Resource Efficient Observability

- CLP: Efficient and Scalable Search on Compressed Logs [OSDI'21]
  - Deployed on Uber's entire Big Data Platform
- Hubble: Performance Debugging with In-Production, Just-in-Time Method Tracing on Android [OSDI'22]
  - Shipped on all Huawei's Android devices in China

CLP : Efficient and Scalable Search on Compressed Text Logs

with Kirk Rodrigues and Yu Luo

https://github.com/y-scope/clp

The Log Management Pipeline

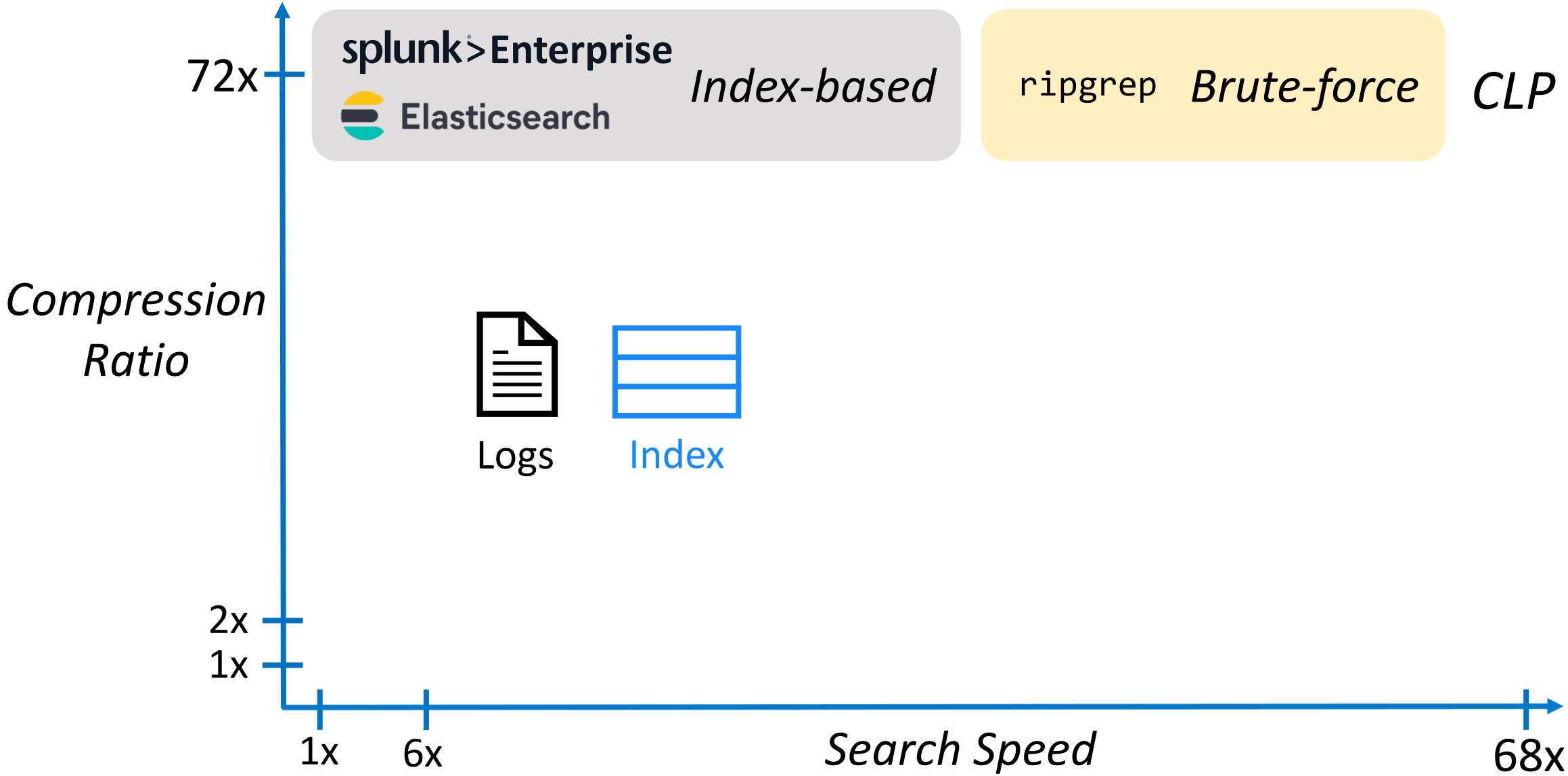
- Provide crucial runtime information
- Widely used for many purposes

splunk>Enterprise

Elasticsearch

ripgrep

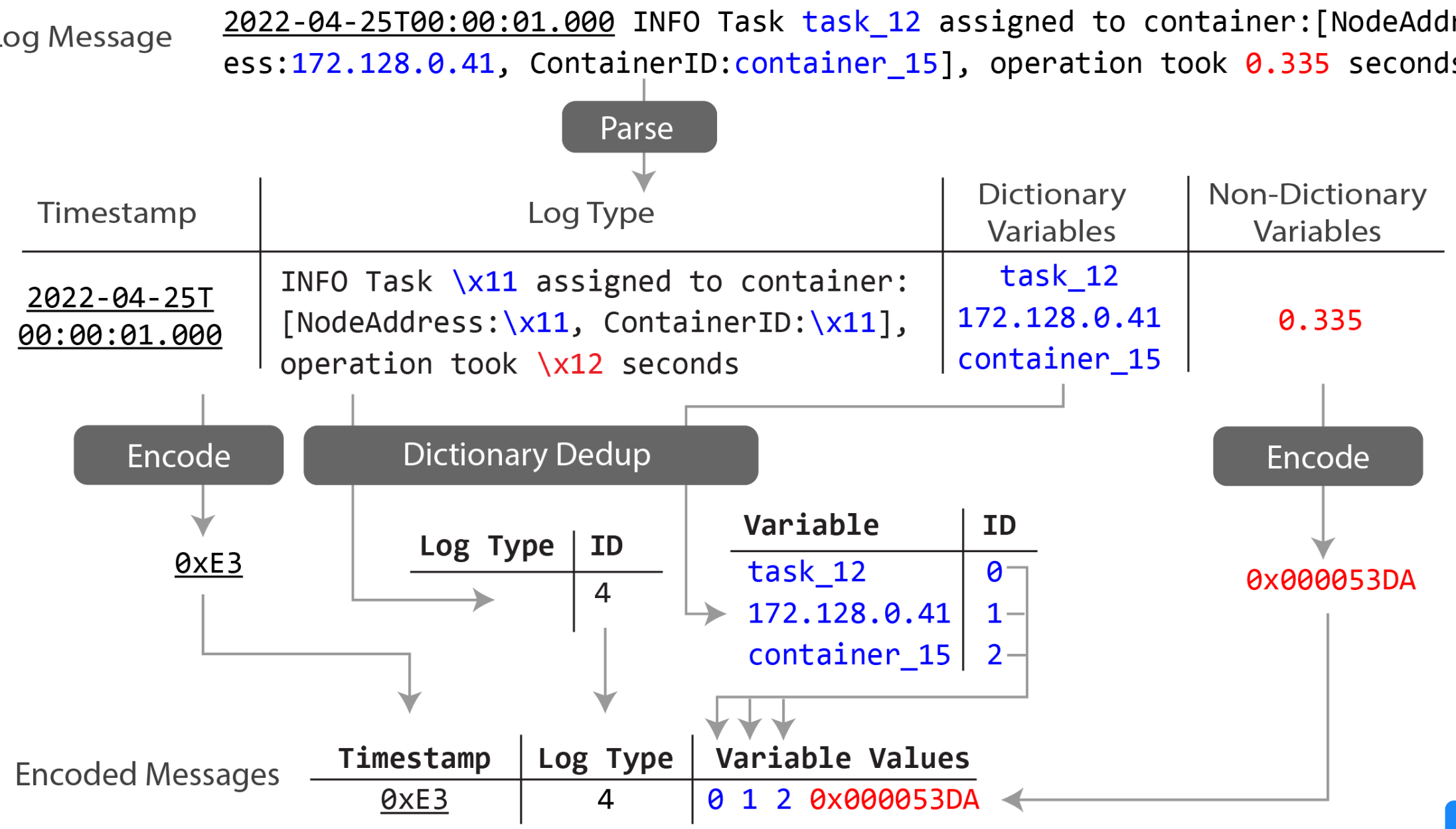
Design Space of Search Tools



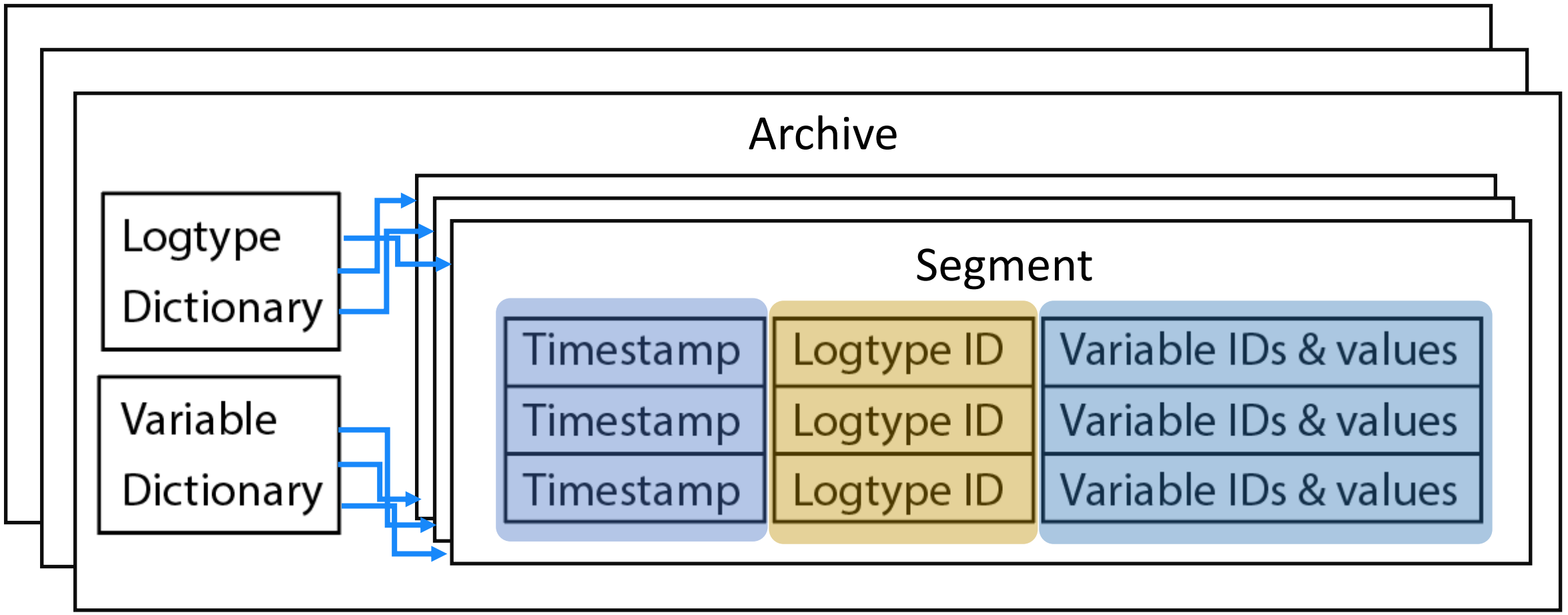
Explore the Repetitiveness of Logs

2020-01-02T03:04:05.006 INFO Task task_12 assigned to container: [NodeAddress:172.128.0.41, ContainerID:container_15], operation took 0.335 seconds

Lossless Compression

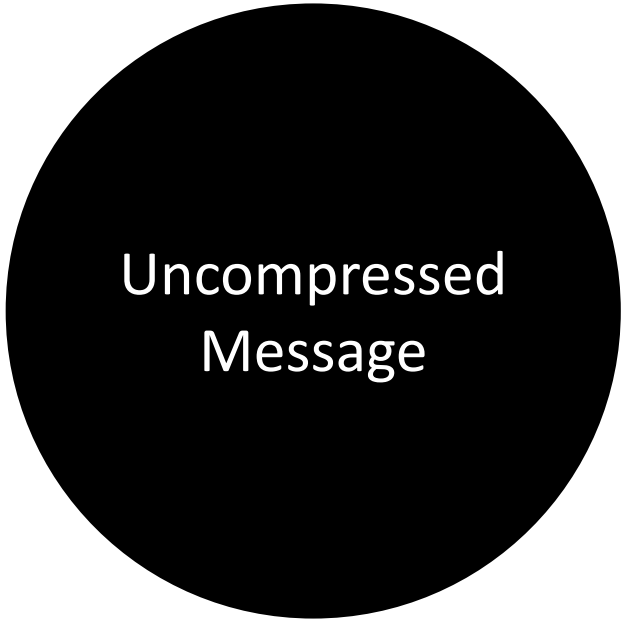


On-disk Format



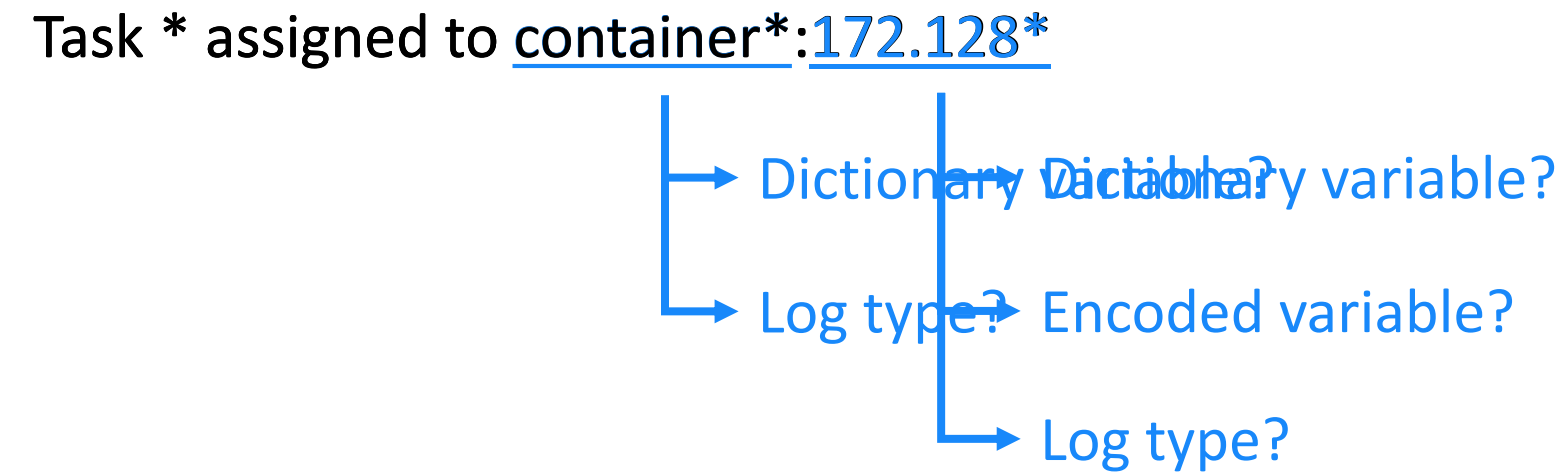
Further compress with zstandard

Search



Search

2020-01-02T03:04:05.006 INFO Task task_12 assigned to container: [NodeAddress:172.128.0.41, ContainerID:container_15], operation took 0.335 seconds



Search

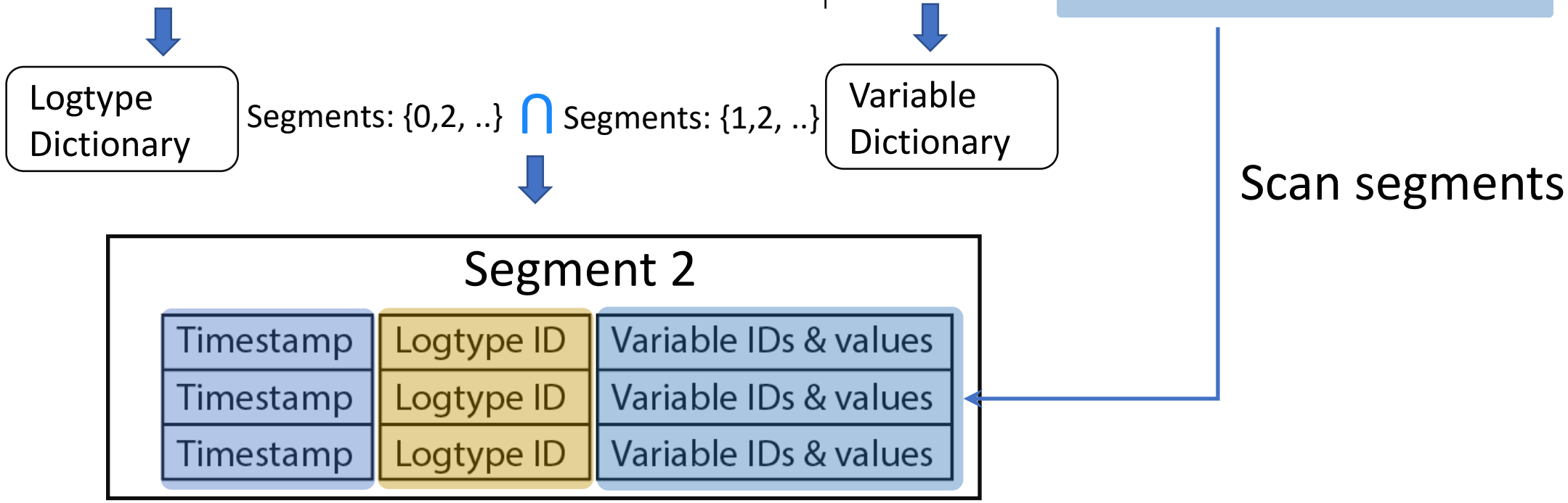
Task * assigned to container*:172.128*

#	Log type	Variables
1	Task * assigned to container*:172.128*	-
2	Task * assigned to container*:	172.128* (IP address)
3	Task * assigned to container*:	172.128* (float)
4	Task * assigned to :172.128*	container*
5	Task * assigned to :	container*, 172.128* (IP address)
6	Task * assigned to :	container*, 172.128* (floating point)

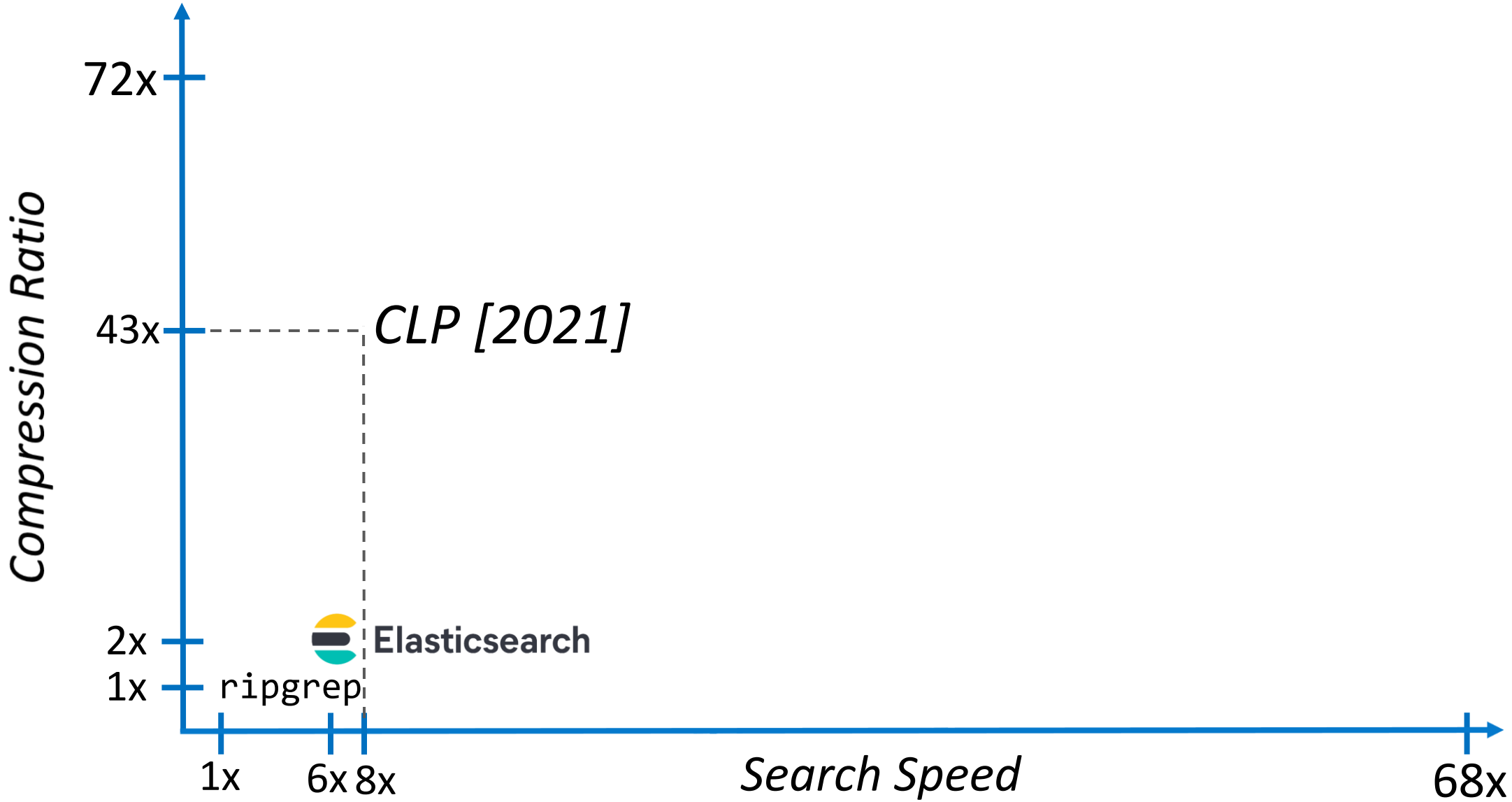
Search

Task * assigned to container*:172.128*

#	Log type	Variables
6	Task * assigned to :	container*, 172.128* (floating point)



Result


72x
43x
2x
1x
Compression Ratio
CLP [2021]
Elasticsearch
ripgrep
1x
6x
8x
Search Speed
68x

Optimization: Group-by Log Type (GLT)

Timestamp	Log Type	Variables
0xE3	4	0 1 2 0x000053DA
…	6	…
…	6	…
…	4	…
…	6	…

Log Type 4: Encoded Message Table

Timestamp	Task	IP	Container	Latency
0xE3	0	1	2	0x000053DA
..	..	..	..	..

Log Type 6: Encoded Message Table

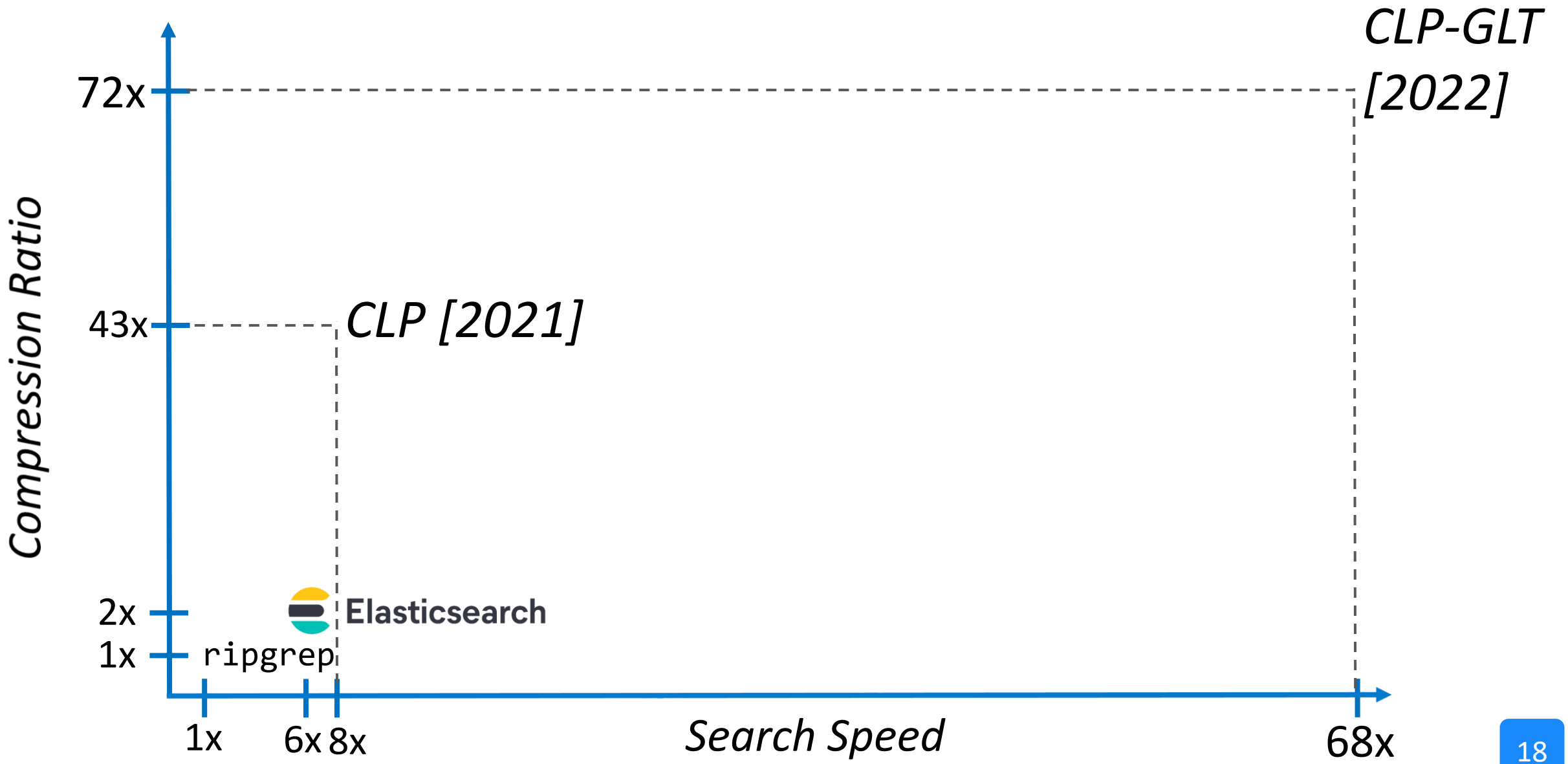
Timestamp	var1	var2	var3
..	..	..	..
..	..	..	..
..	..	..	..

Query:

```
assigned to container: [*, ContainerID:container_15]
```

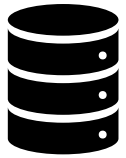
- Log type: 4
- Container: 2

Result

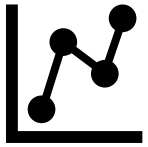

CLP-GLT
[2022]
72x
43x
CLP [2021]
Compression Ratio
2x
Elasticsearch
1x
ripgrep
1x
6x
8x
Search Speed
68x

Deployment at Uber

100,000s of workers **per** job



100 PB data analyzed **per** job



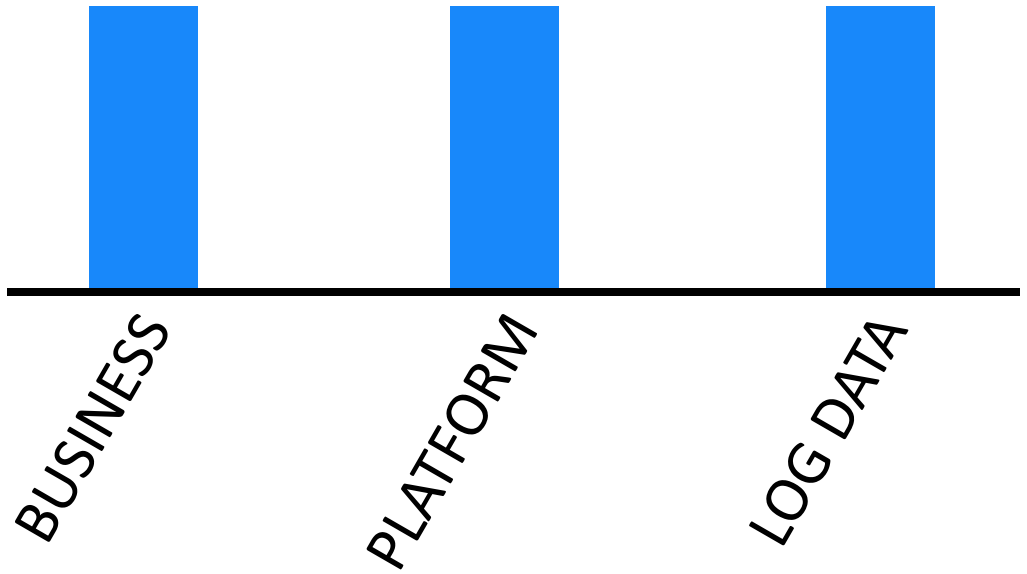
250,000 analytics jobs **per** day

200 TB logs **per** day

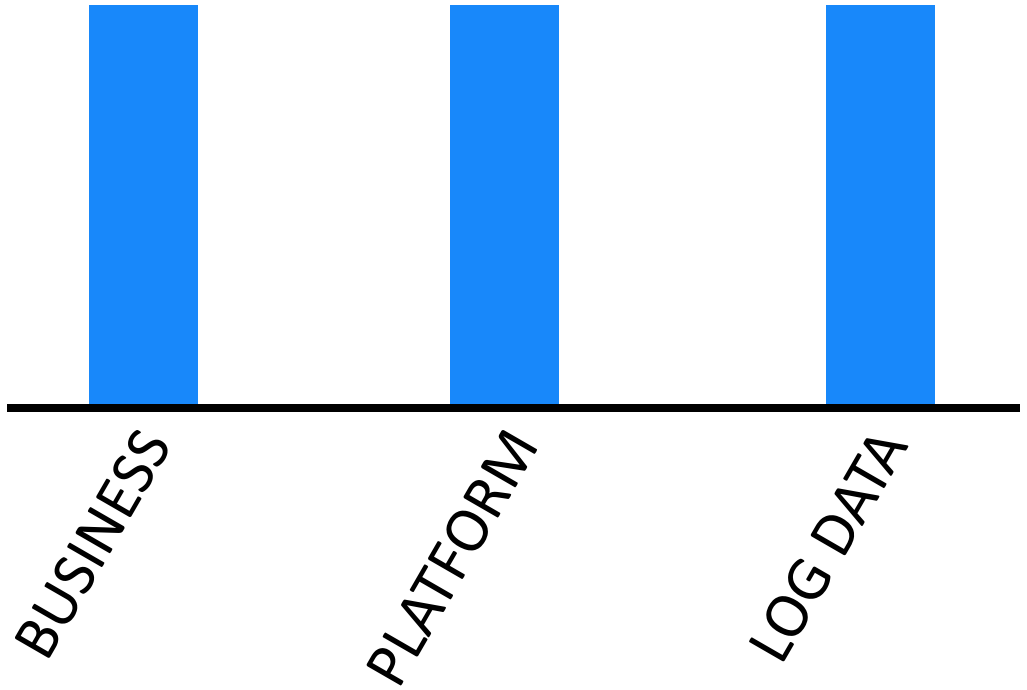
BIG DATA

Analytics Platform

Deployment Experience: Uber



Deployment Experience: Uber



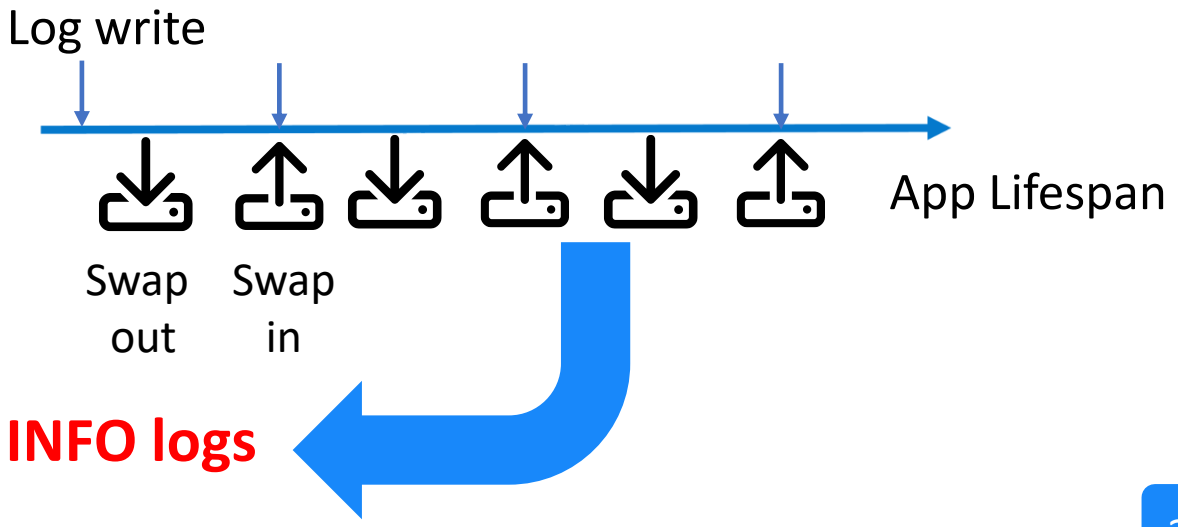
The Problems at Uber

Need 10x longer retention

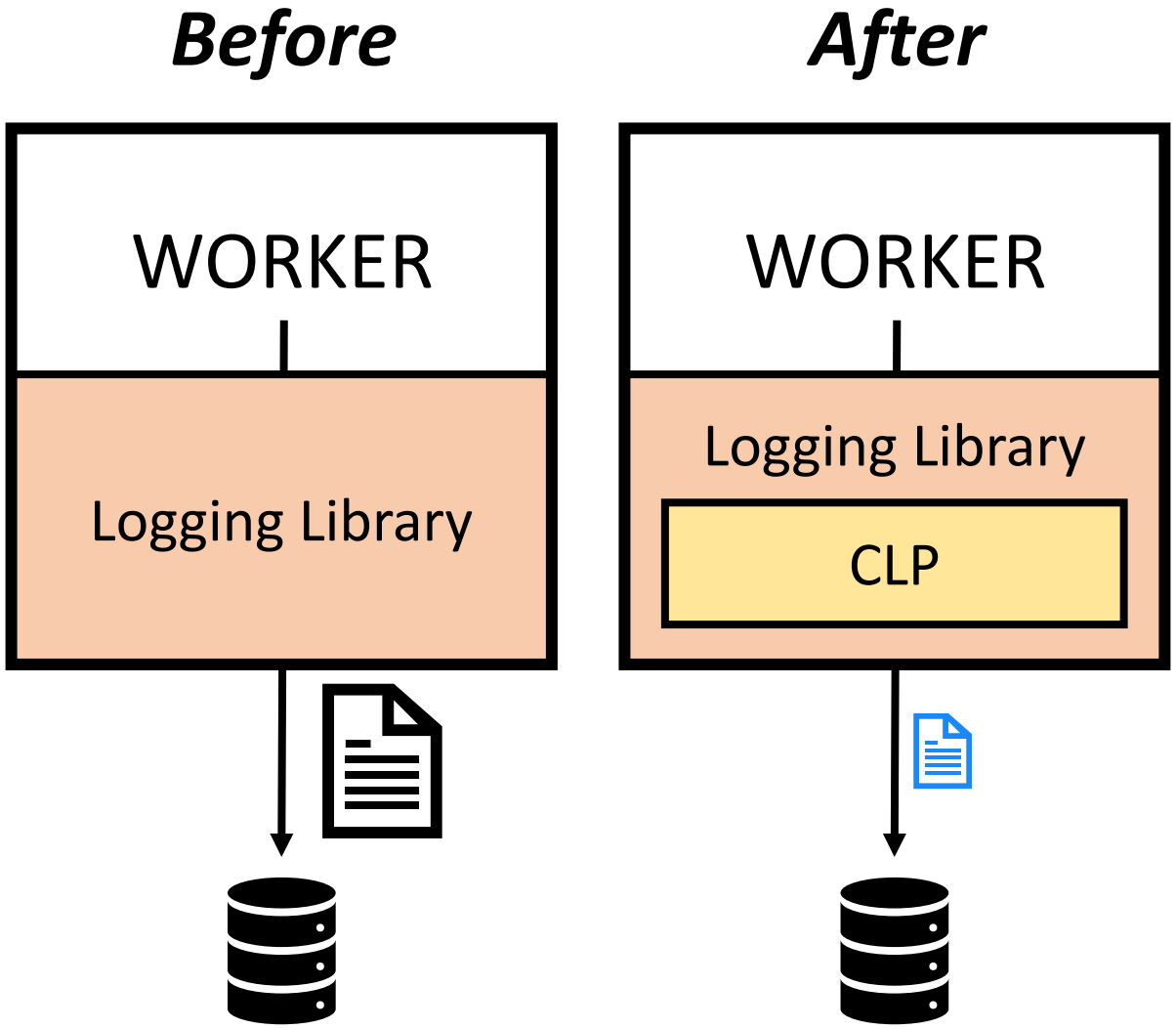
SSDs Burning Out

4 year life-span used up in < 1 year

Various use cases



Integrating CLP at Uber



Integrating CLP at Uber: Challenges

Uber Worker

Single log file per worker

Workers are memory constrained

CLP

Designed to compress many files in batches

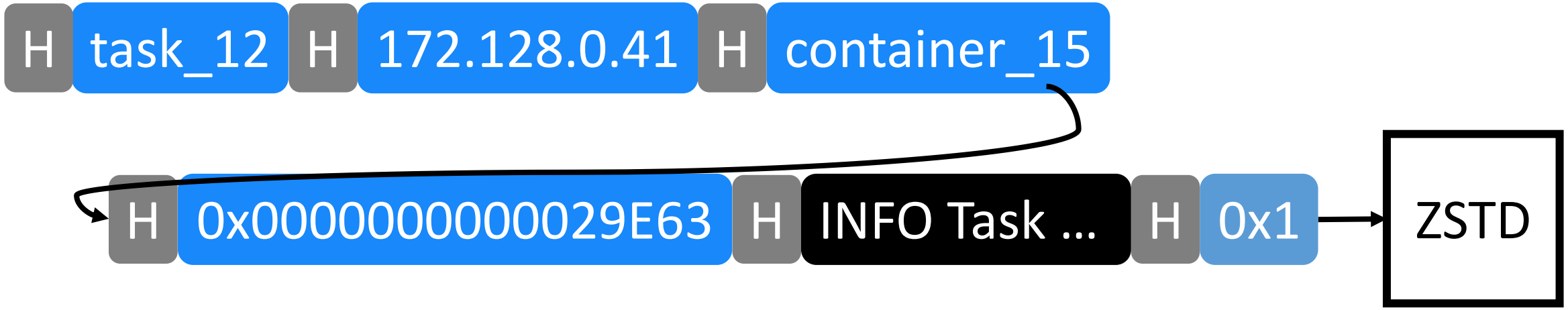
Uses hundreds of MB of memory to store dictionaries and columns

Integrating CLP at Uber: 2 Phases

2022-04-25T00:00:01.000 INFO Task task_12 assigned to container:
[NodeAddress:172.128.0.41, ContainerID:container_15], operation took 0.335 seconds

Timestamp	Log Type	Dictionary Variables	Non-Dictionary Variables
2022-04-25T 00:00:01.000	INFO Task \x11 assigned to container: [NodeAddress:\x11, ContainerID:\x11], operation took \x12 seconds	task_12 172.128.0.41 container_15	0.335

PHASE 1: Parse and stream out



PHASE 2: Aggregate into archives

Integrating CLP at Uber: Results

PHASE 1

169x

compression ratio

PHASE 2

2x

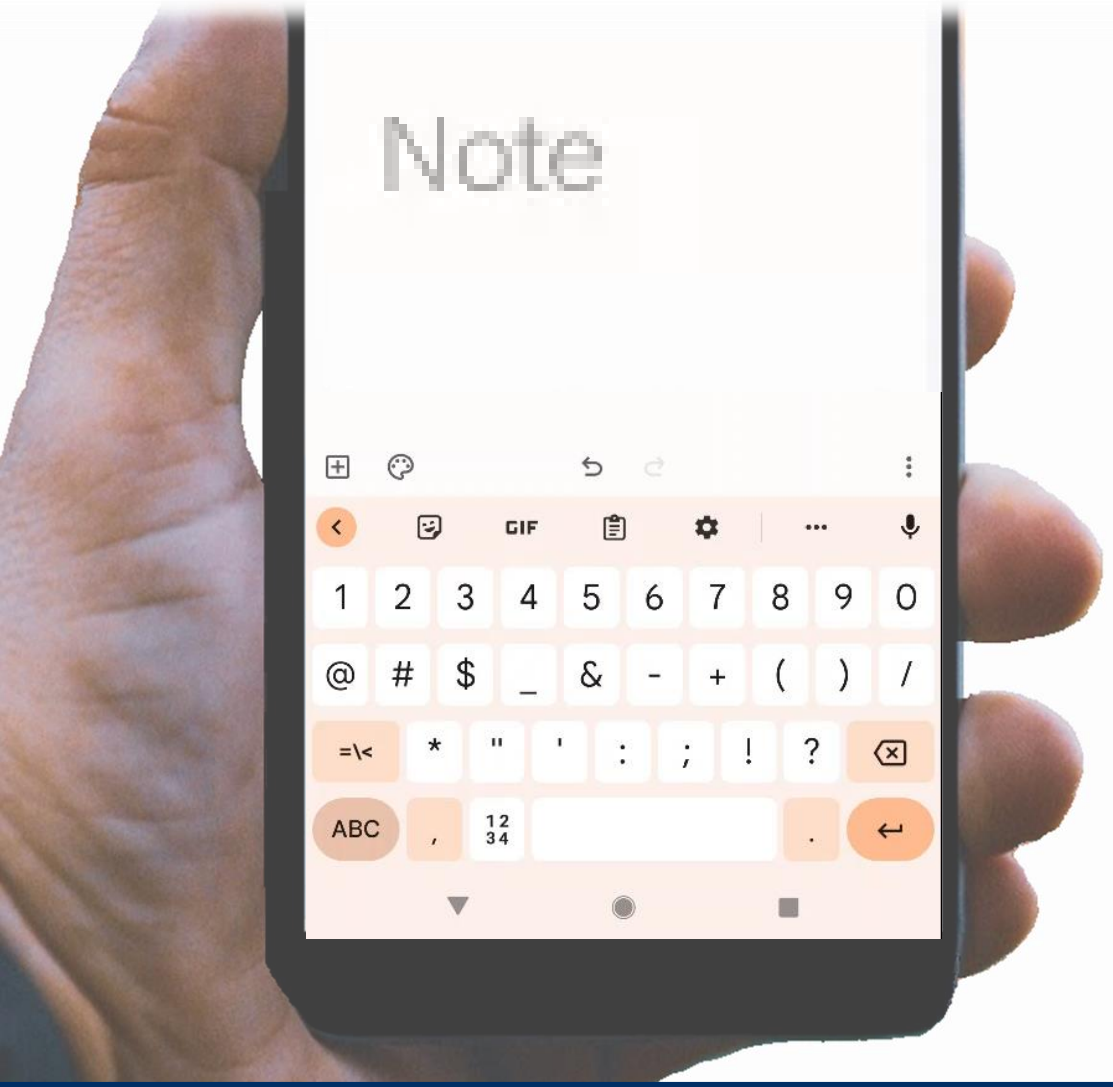
more

Hubble: Performance Debugging with In-Production, Just-In-Time Method Tracing on Android

Yu Luo, Kirk Rodrigues,
Cuiqin Li, Fen Zhang, Lijin Jiang, Bing Xia,
David Lion, Ding Yuan

Published in OSDI'22

What is an Intermittent Performance Bug?



Example: While typing

1. Keyboard becomes unresponsive
2. A second later…
3. Problem resolves itself, but typed an extra “S” … ☹

Why Focus on Intermittent Performance Bugs?

Elusive: Hardest to catch during testing

- Requires rare combination of events or environment factors

Painful to Debug: Lack of diagnostic data from production

- Sparse trace from important runtime methods only
- Low quality application log messages
- Periodic system metrics

Hubble: Continuous Method Tracing In-Production

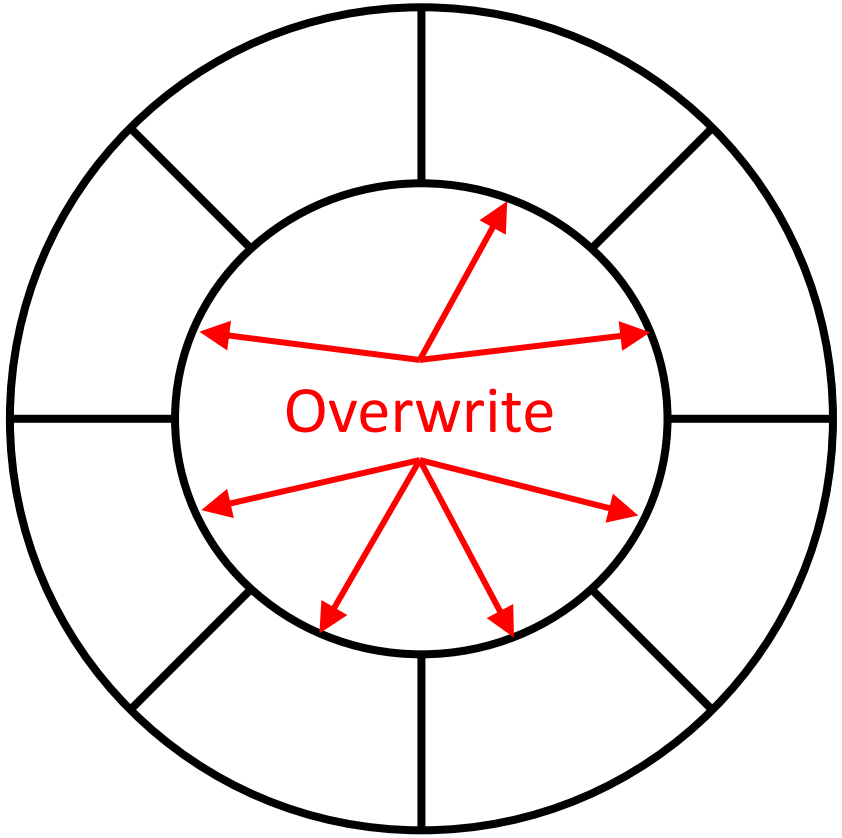
Application Method

System Method

Hubble: Continuous Method Tracing In-Production

Application Method

System Method



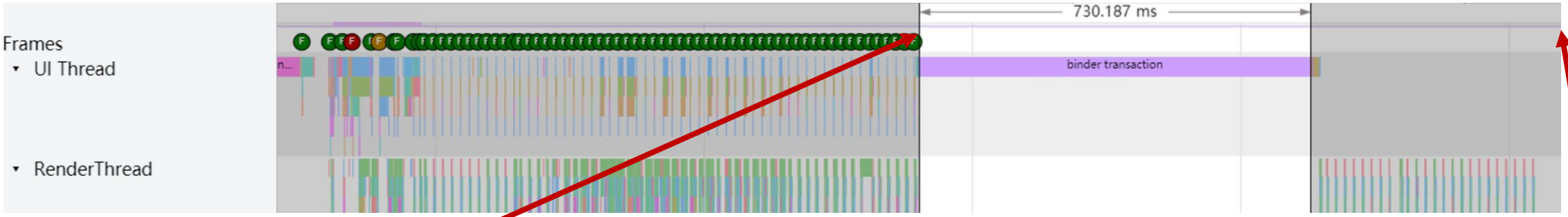
In-Memory Ring Buffer

Minimal Privacy Concerns

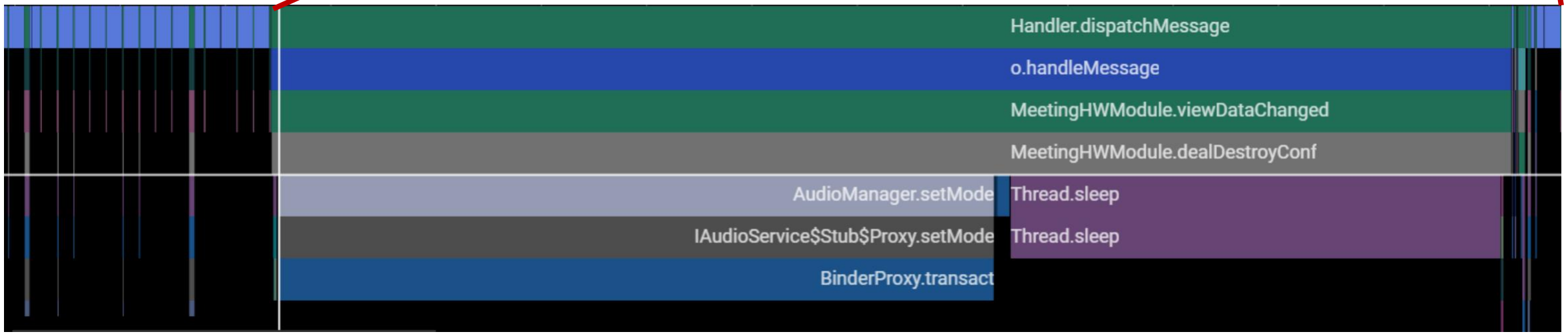
- Does not collect personally identifiable information
- Trace data only contain method names and timestamp
  - WER, MacOS or Mozilla crash report collect minidumps

Case Study – One Second Freeze after Video Call

Without Hubble



With Hubble



Real World Requirements

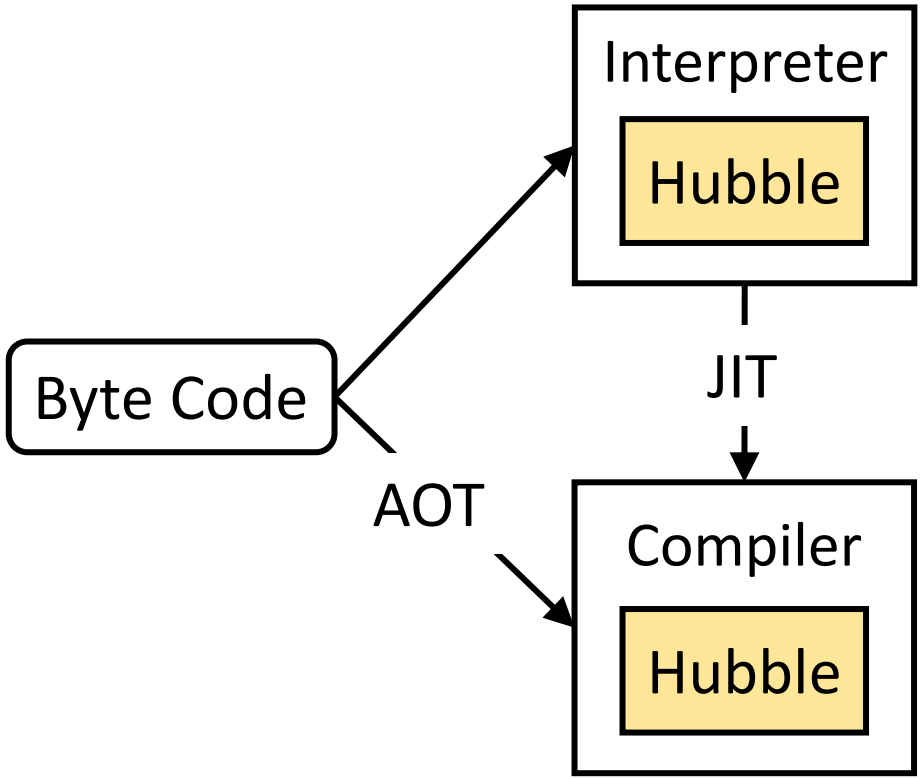
Performance & Memory overhead:

- 2 - 3% worst case

Android specific requirements:

- No source code dependency
- Easy to maintain and rebase
- Optimize for big.LITTLE CPU cores

Opportunity: Leverage JIT in Runtime Environment



Ensure correctness

No data dependency with method

- compiler free to optimize out trace point

Inlined Trace Point

Avoid saving register to stack (slow)

Hand-Optimized Assembly

Efficient Access to Trace Data

- Method pointer at a specific CPU register upon method entry
- Direct access to clock cycle counter (RDTSC in x86) register value

Specific Optimizations for big.LITTLE CPU's LITTLE Cores

- Worked SoC chip designer to perform uArch specific optimizations
- Data prefetching, manual instruction re-ordering, avoid pipeline stalls

Deployment Experiences

Status:

- Dev branch – 2019 Production branch - 2020

Size:

- Tens of thousands of user in beta groups
- Thousands of devices in automated regression test
- Can be enabled on other users with consent

Use Case:

- Triage and diagnose intermittent performance bugs in production
- Equally helpful for bugs discovered via automated testing

Concluding Remarks

- Resource Efficiency: Key challenge on Log and Trace management
  - CLP: Log compression & search
  - Hubble: Method tracing on Android
- Exciting opportunities for resource-efficient *observability* support