Multi-GPU Accelerated Processing of Time-Series Data of Huge Academic Backbone Network in ELK Stack

Usenix LISA19

October 28–30 2:00 pm–2:45 pm

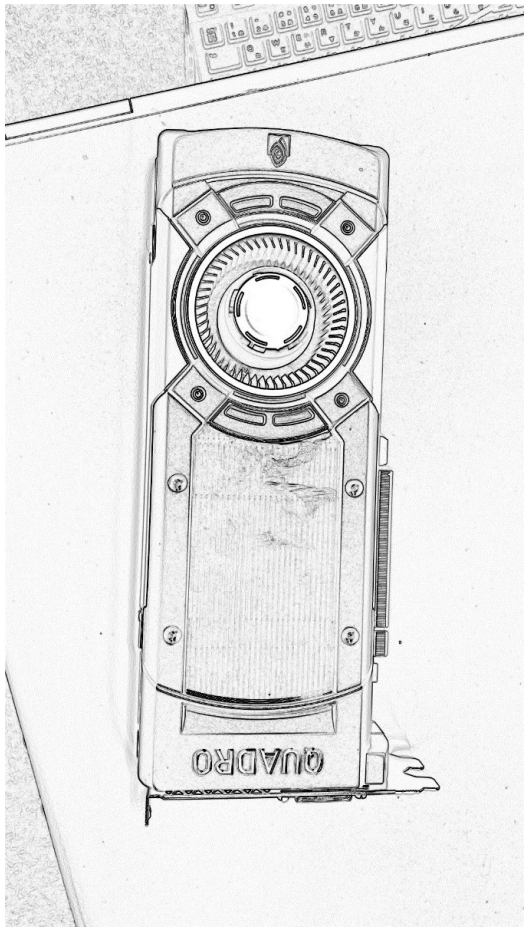
Portland, OR, USA

Ruo Ando
Center for Cybersecurity Research and Development,
National Institute of Informatics

Stand on huge academic backbone

- Three years (painful) operational experience in deploying multi-GPU accelerated monitoring system of huge academic backbone network.

- Science Information Network (SINET) is a Japanese academic backbone network for more than 800 research institutions and universities.



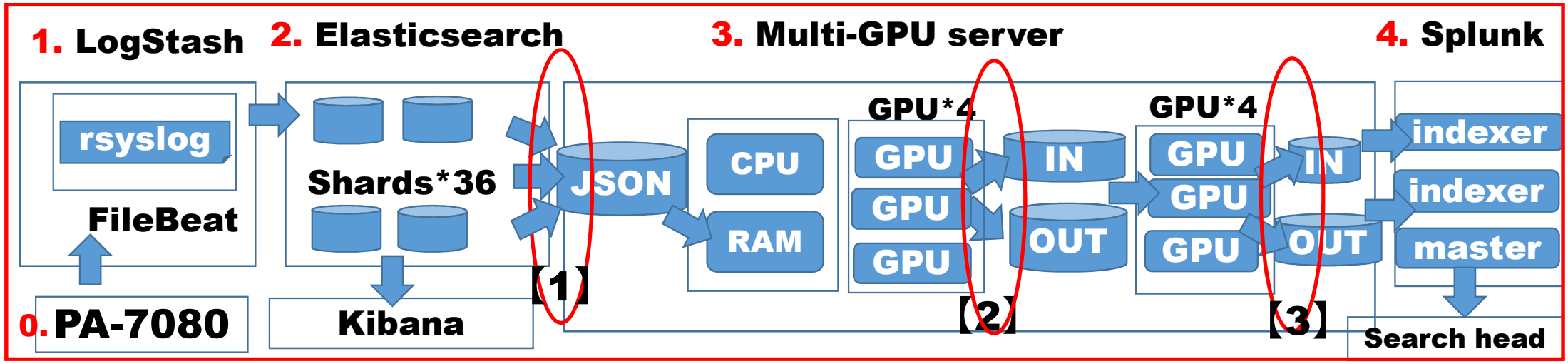
Outline

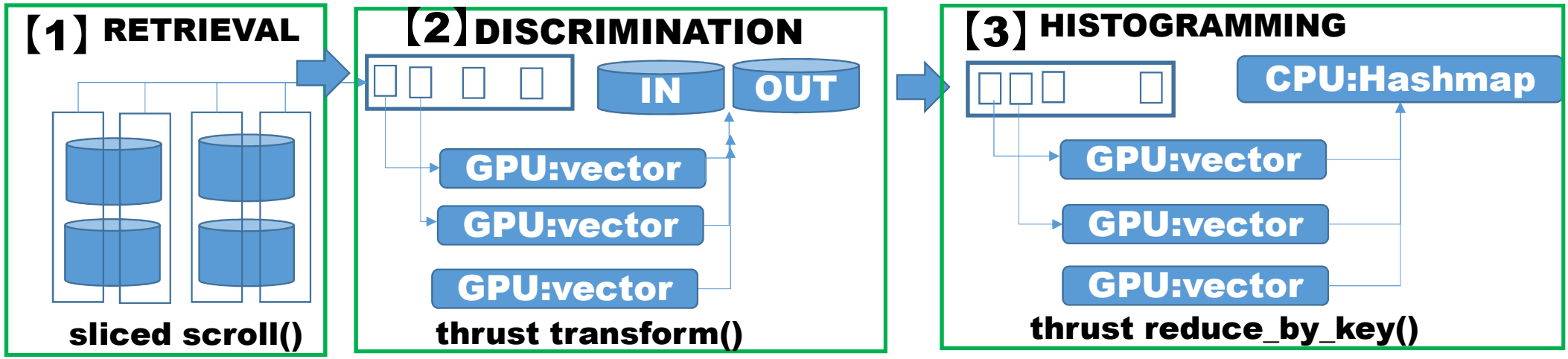
- ❒ Overview: Pipeline of Elastic stack, Multi GPU and Splunk
- ❒ Backgrounds and bottlenecks
- ❒ Strategy of parallelism for unpredictable
- ❒ Design philosophy
- ❒ Floorplan and configuration (Elastic stack and Splunk)

- ❒ Bottleneck 1 - Huge pagination without scoring
- ❒ Bottleneck 2 - Direction discrimination of big session data
- ❒ Bottleneck 3 - Histogramming with millisecond interval

- ❒ Conclusion and impressions

Pipeline: PA, ELK, GPU and Splunk





128 * 36 queries
Huge pagination

32-64 threads
Massive bitmasking

4-8 threads
Billions by millisecond

Background and bottlenecks

2016 Sep -2019 Oct

2010s - Universities under (massive) attack

- (Kind of) a top-down mission from Japanese government, ministry and universities.
- In 2016, the minister residence (a little similar to White House in US) worried so much about information breaches and assurance in Japanese universities.
- SINET connects various kinds of research facilities in such fields as space science, high-energy physics, nuclear fusion, computing science, and so on.
- (Perhaps)↑ These research facilities attracted APT people

Bottlenecks in 2016 - 2018: Large computing time

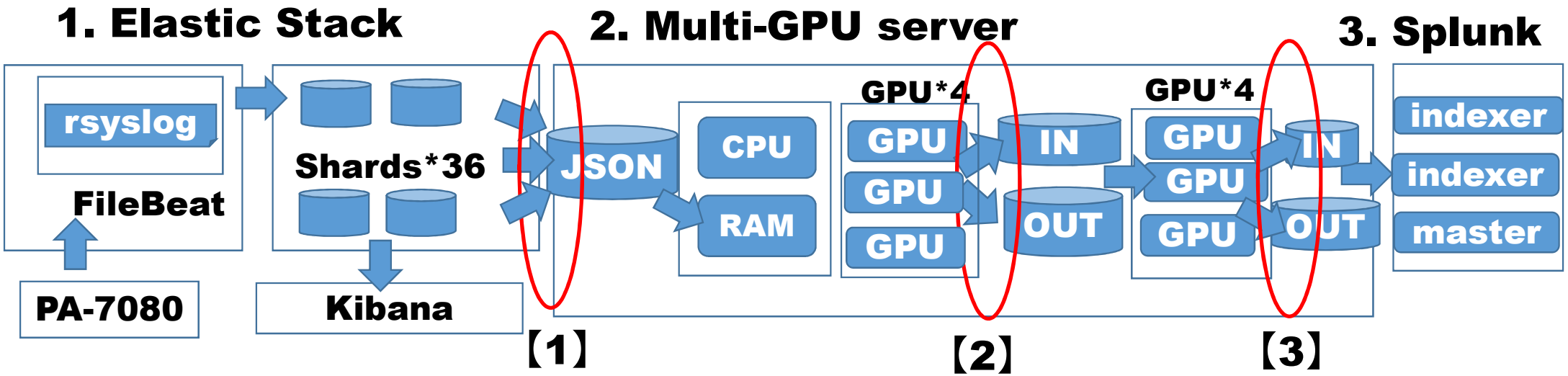
■ For example, to handle session data stream of **100GB** (for 24 hours, randomly generated in this case) ...

1. DATA DUMP: It took **1664 minutes** (about 26 hours) to retrieve all data from Elastic search.

2. DIRECTION DISCREMINATION: It took **704 minutes** (about 11.5 hours) for filtering ingress/egress traffic.

3. HISTOGRAMMING: It took **476 minutes** (about 8 hours) with the millisecond interval.

Three pitfalls on pipeline (2016-2018)



Huge pagination without scoring

1664m32.441s

Massive bitmasking

704m47.418s

Billions by millisecond

476m41.308s

1440m (24 hours) !

132m34.298s(12.6x)

174m7.888s(4.04x)

12m34.706s(39.6x)

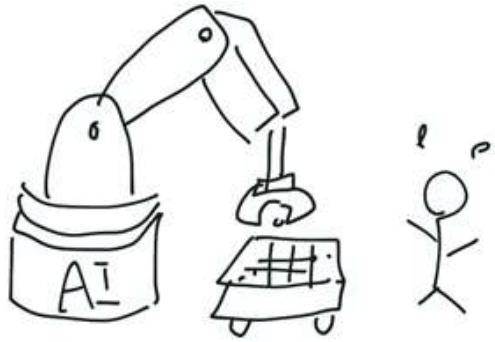
1664m32.441s

476m41.308s

How can we reduce unreasonable computation time by multi GPU acceleration?

704m47.418s

Philosophy: Data mining is plastic art



"Data mining is often considered in terms of location and extraction of nuggets of information from a sea of background noise. But this metaphor is entirely wrong. Data mining is essentially a plastic art, for it responds to the sculpture of the medium itself, to the background noise itself."
Gayatri Spivak, “Can the Subaltern Speak?”

In the world of 100 Gbps traffic stream in 2010s, the crafted filtering is the first priority.

STRATEGY of parallelism

Dynamic allocation for unpredictables

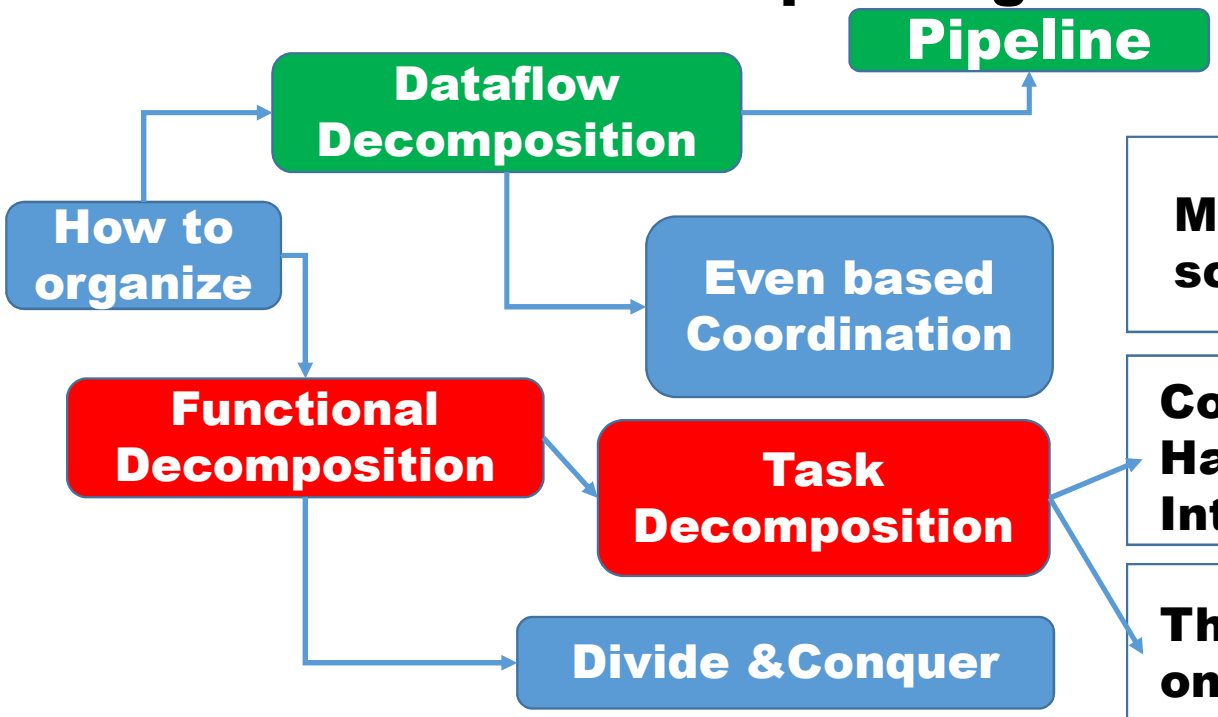
Sessions in the World of Extremistan

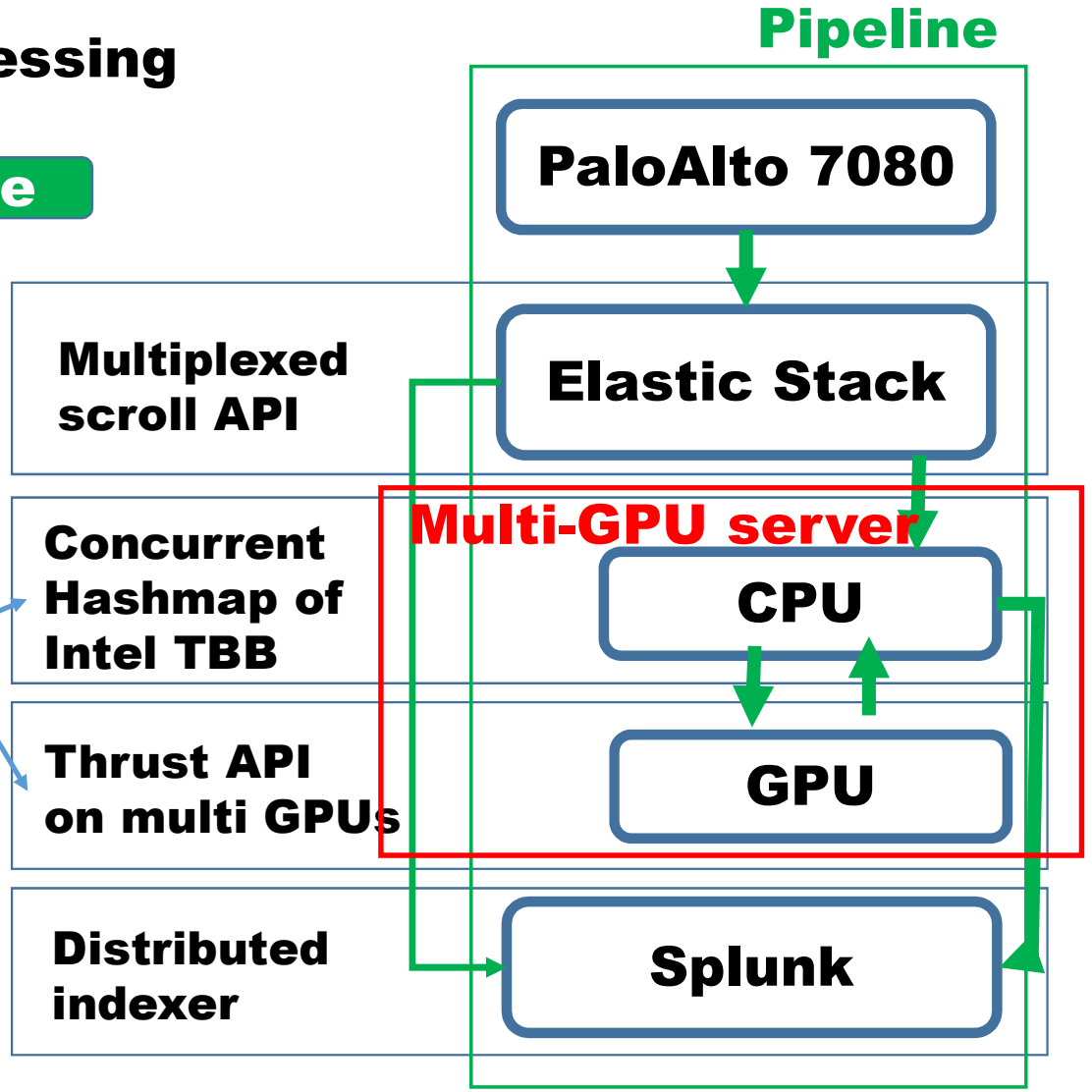
Sessions are anthropomorphized..

- Sessions are not homogeneous. Some are large, small, short, long...
- Sessions are unpredictable. We cannot use static scheduling.
- Task decomposition is useful when the amount of processing time for each piece is different or even unpredictable at the outset of computation.

Task decomposition and pipeline

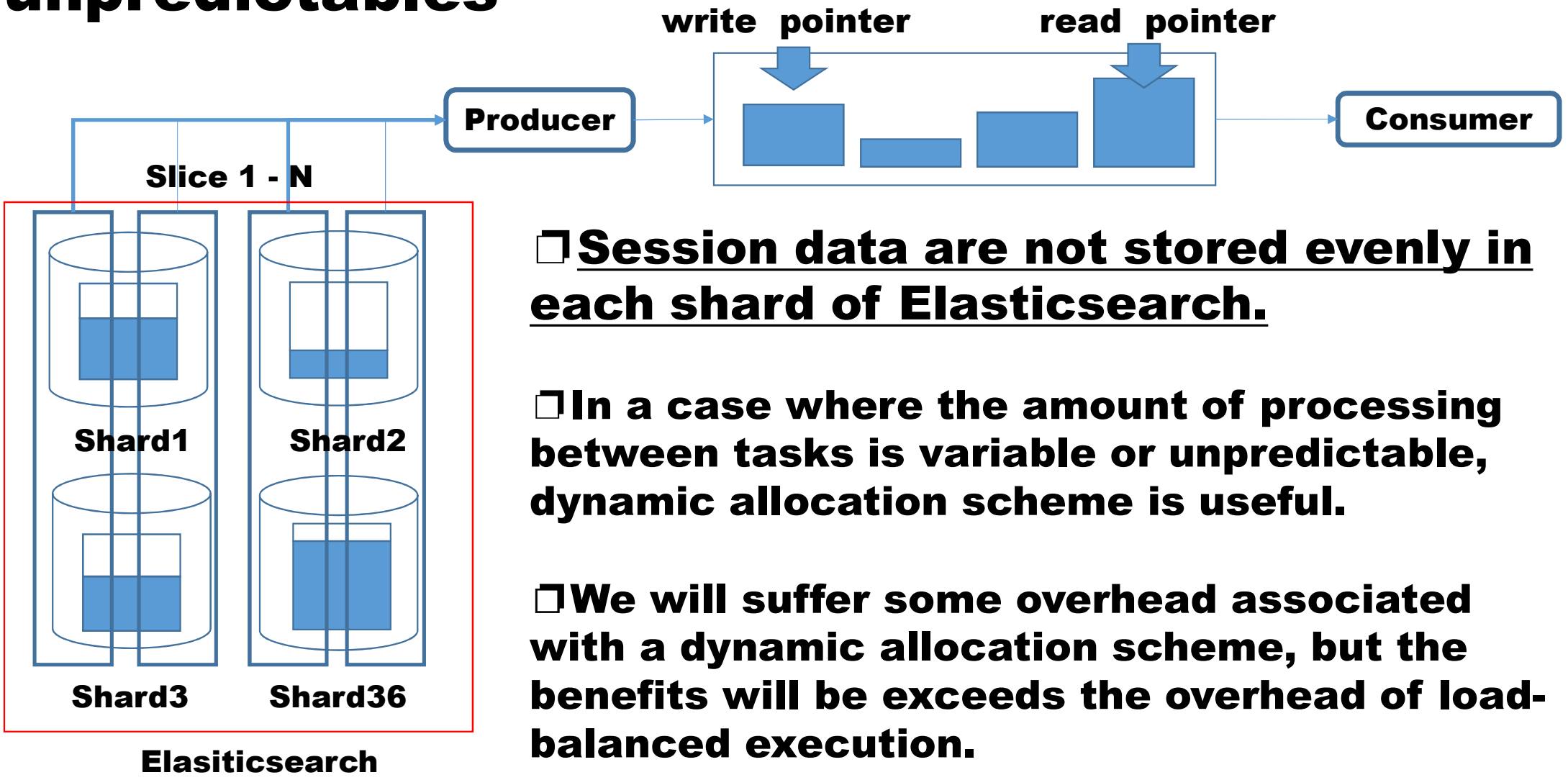
Dataflow decomposition is for processing a stream of data in multiple stages.





In **functional decomposition**, process is split into the different steps which take place.

Dynamic allocation (task decomposition) for unpredictables

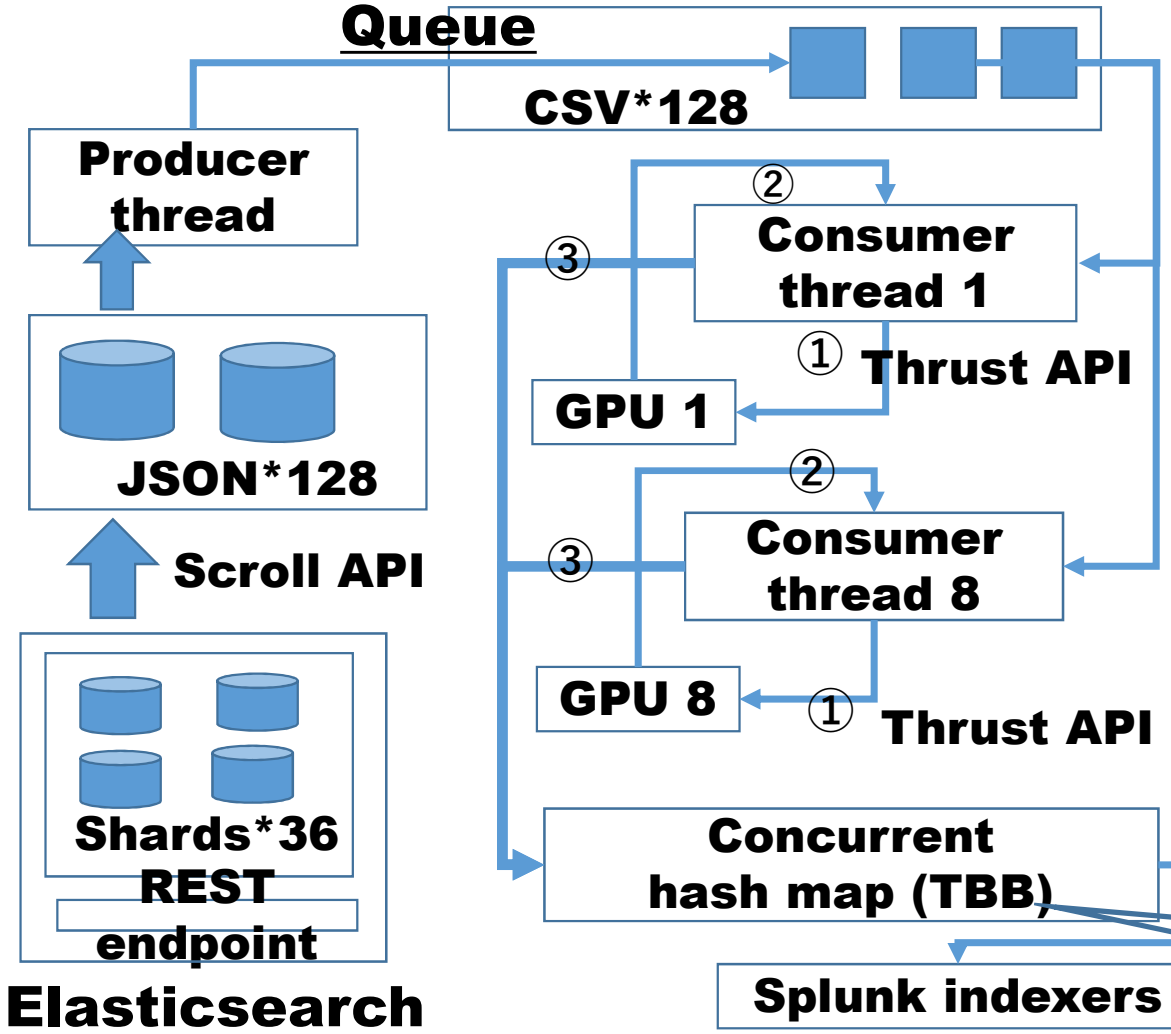


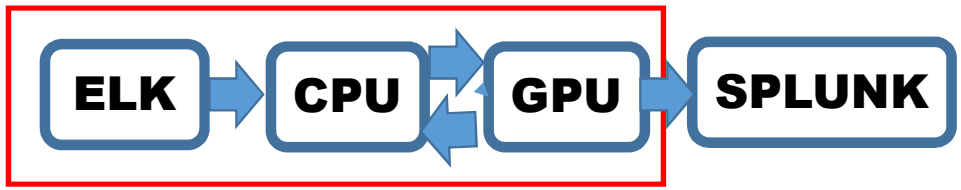
- **Session data are not stored evenly in each shard of Elasticsearch.**

- **In a case where the amount of processing between tasks is variable or unpredictable, dynamic allocation scheme is useful.**

- **We will suffer some overhead associated with a dynamic allocation scheme, but the benefits will be exceeds the overhead of load-balanced execution.**

Task decomposition (dynamic allocation) with multi-GPUs





# of shards: 36

# of JSON/CSV: 128

# of GPU: 8

GPU accelerates comsumer threads.

❑**Task decomposition is useful for cases when there are many more pieces of work (# of JSON/CSV:128) than threads (GPU:8).**

Defined as global static variable

```
1: typedef tbb::concurrent_hash_map<long, int> iTbb_Vec_timestamp;
2: static iTbb_Vec_timestamp TbbVec_timestamp;
```

0.Static global variables

```
3: void Pthread_comsumer()
4:  size_t kBytes = data.size() * sizeof(unsigned long long);
5:  unsigned long long *key;
6:  key = (unsigned long long *)malloc(kBytes);
8:  reduction(key, value, key_out, value_out, kBytes, vBytes, data.size(),
&new_size,thread_id);
10:  iTbb_Vec_timestamp::accessor tms;
11:  TbbVec_timestamp.insert(tms, key_out[i]);
12:  tms->second += value_out[i];
```

1. Calling GPU function

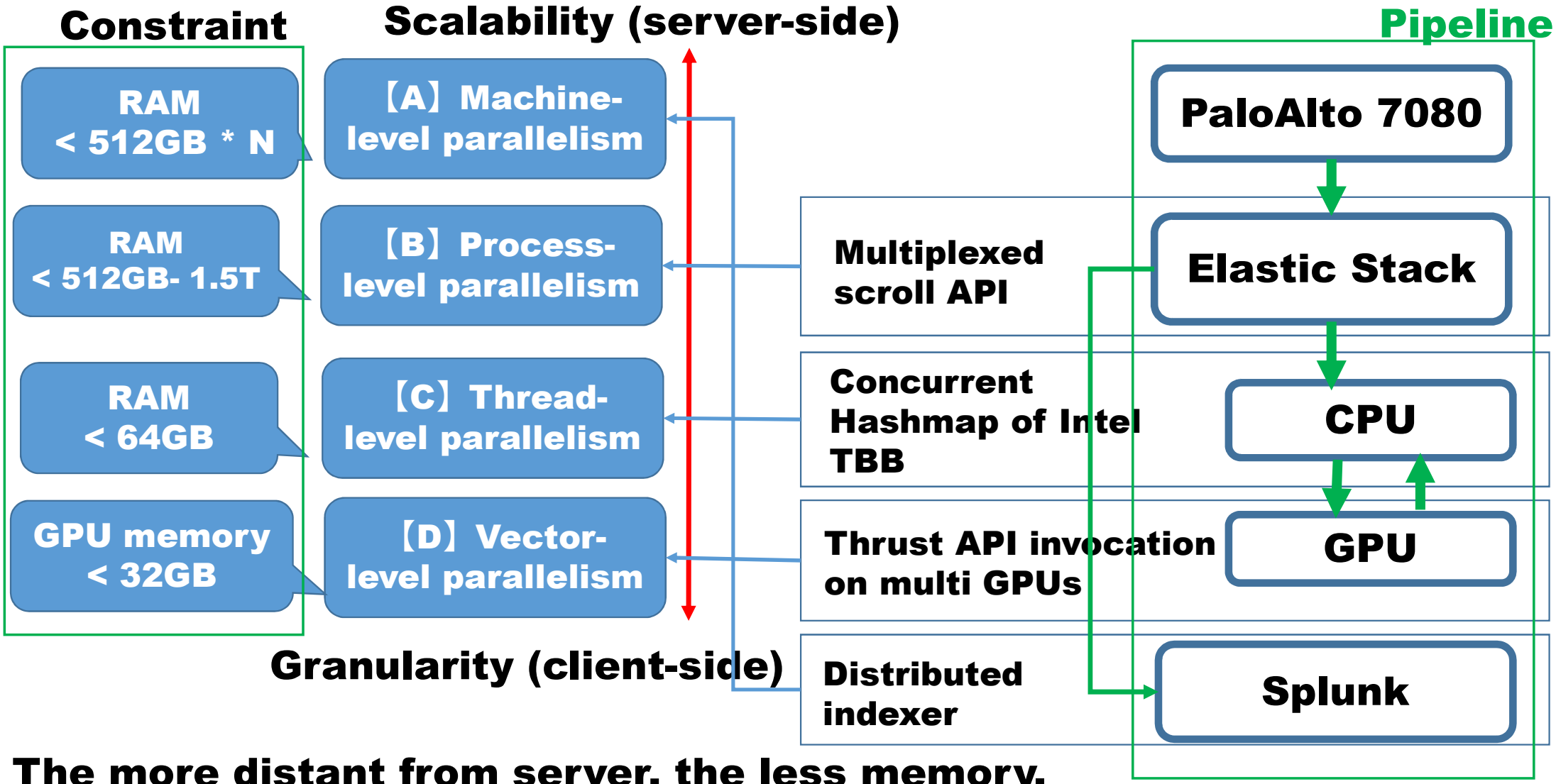
4.Hashmap insertion

```
13:void reduction(unsigned long long *key, long *value, unsigned long long *key_out, long
*value_out, int kBytes, int vBytes, size_t data_size, int *new_size, int thread_id)
14:
15:  cudaSetDevice(thread_id);
17: thrust::sort_by_key(d_vec_key.begin(), d_vec_key.end(), d_vec_value.begin());
18:
19:  auto new_end = thrust::reduce_by_key(d_vec_key.begin(), d_vec_key.end(),
  d_vec_value.begin(), d_vec_key_out.begin(), d_vec_value_out.begin());
```

2. Switch to GPU(N)

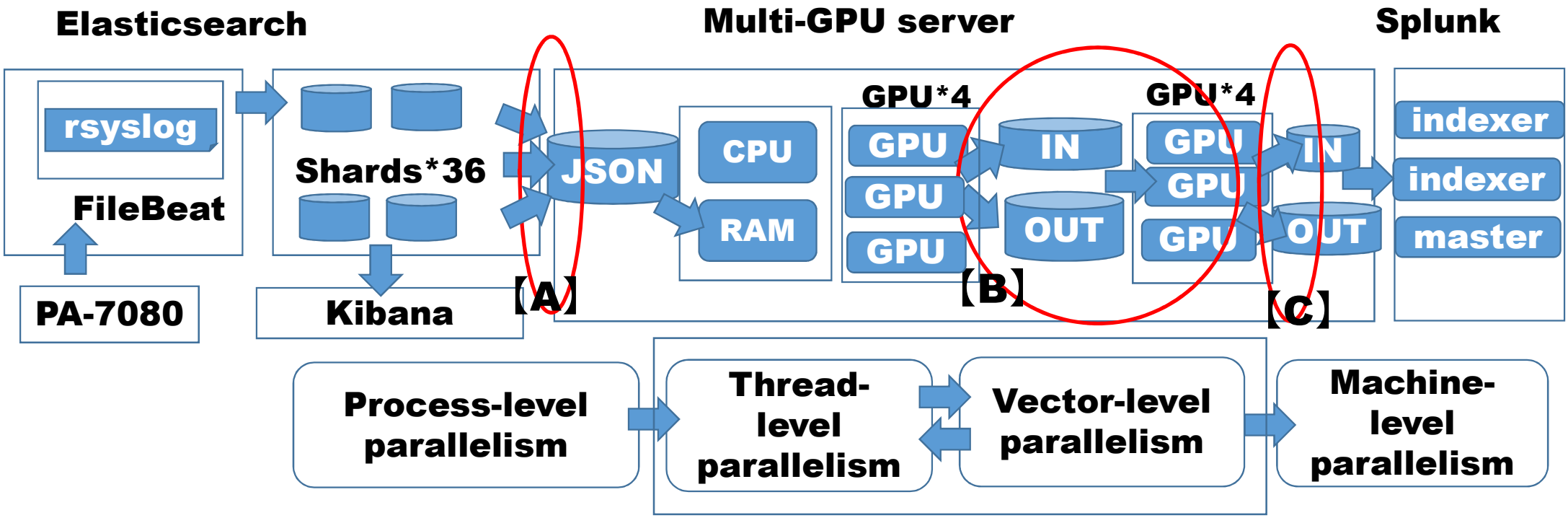
3. Calling ThrustAPI

Layers of parallelism and memory constraint



The more distant from server, the less memory.

Pipeline completed

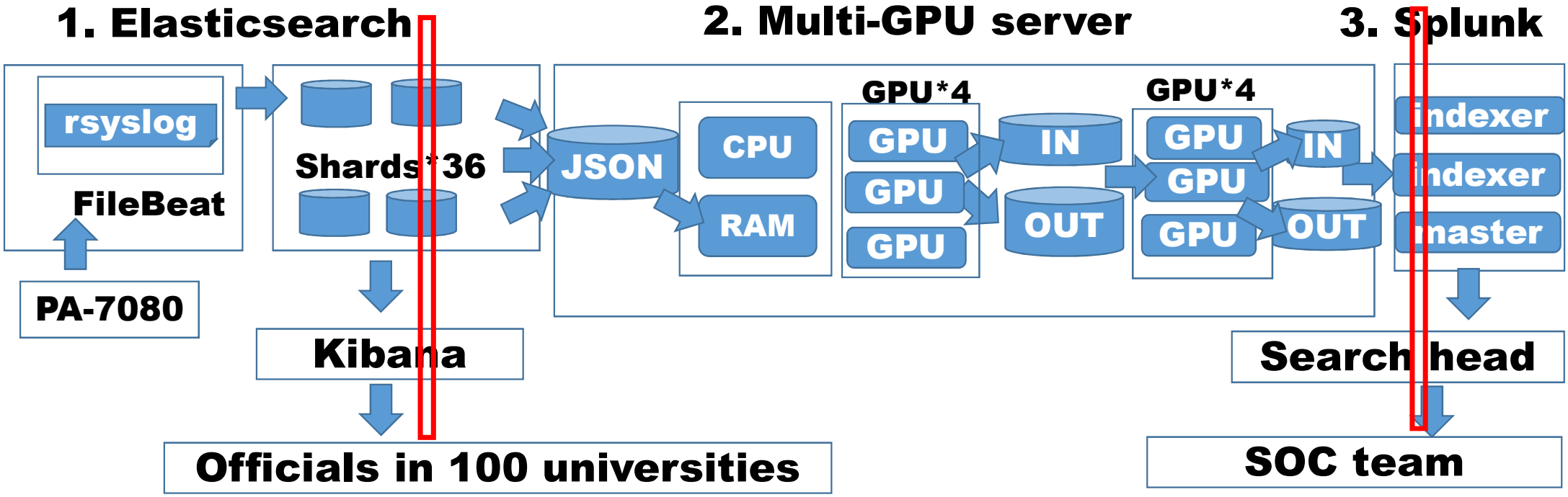


- ❑ Process-level parallelism – Huge pagination: 128 processes
- ❑ Thread-level parallelism – Dynamcic allocation: 64 threads
- ❑ Vector-level parallelism – Dynamic allocation: 8threads
- ❑ Machine-level parallelism – Distributed indexing: 6 indexers

FLOORPLAN and cluster CONFIGURATION

Elastic stack and Splunk

Defense line (2016 - 2019) - Two teams

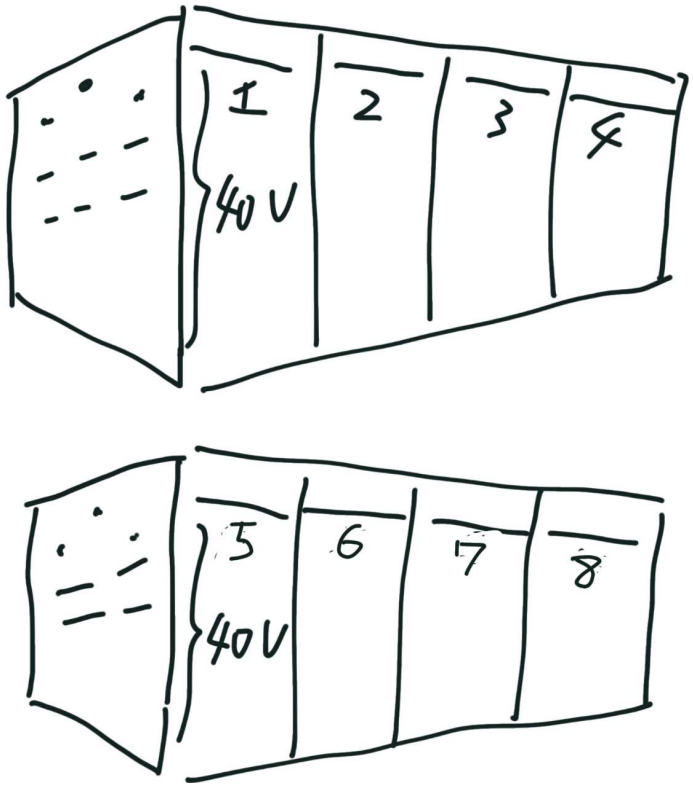


- Academic officials (government employees): In Japan, their term is often several years. Some are often newbies about IT security.
- SOC team: They are specialist with 3 years and more experience. They are responsible for prioritizing alerts, analyzing incidents.

Floorplan - Their requirements (and complaints)

- **University officials:**
  "Analysts, don't touch our ELK stack !
  (Aggregation is memory intensive !)"
  -> separating analytics from Elastic stack

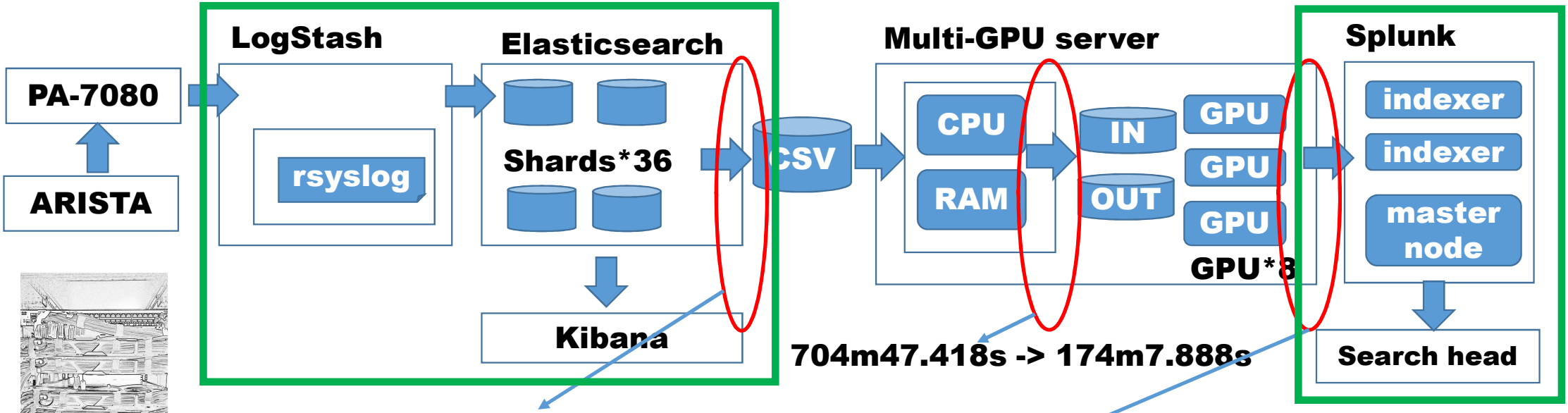
- **SOC team:**
  “Proritizing alerts is our first priority. For being proactive, chagnepoints should detected and reported in mechanized manner.”
  -> introducing Splunk with feature rich commands

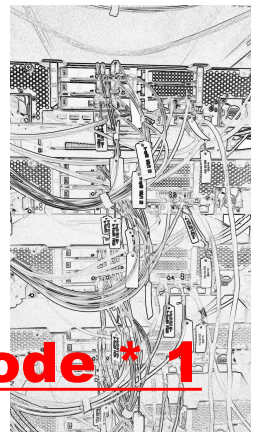


Reference: The Scream @ public domain

From purely financial perspective, GPU is impressive. It is estimated GPU's massively parallel processing could sometimes deliver performance up to a CPU-only configuration at one-tenth the hardware cost, and one-twentieth the power and cooling costs.

Cluster configuration





Logstash * 3, master node *3.
Data nodes * 9 – shards * 36 (9*4)

- **Data decomposition: indices are split into 36 shards.**

- **Functional decomposition. Process is divided into geographical locations.**

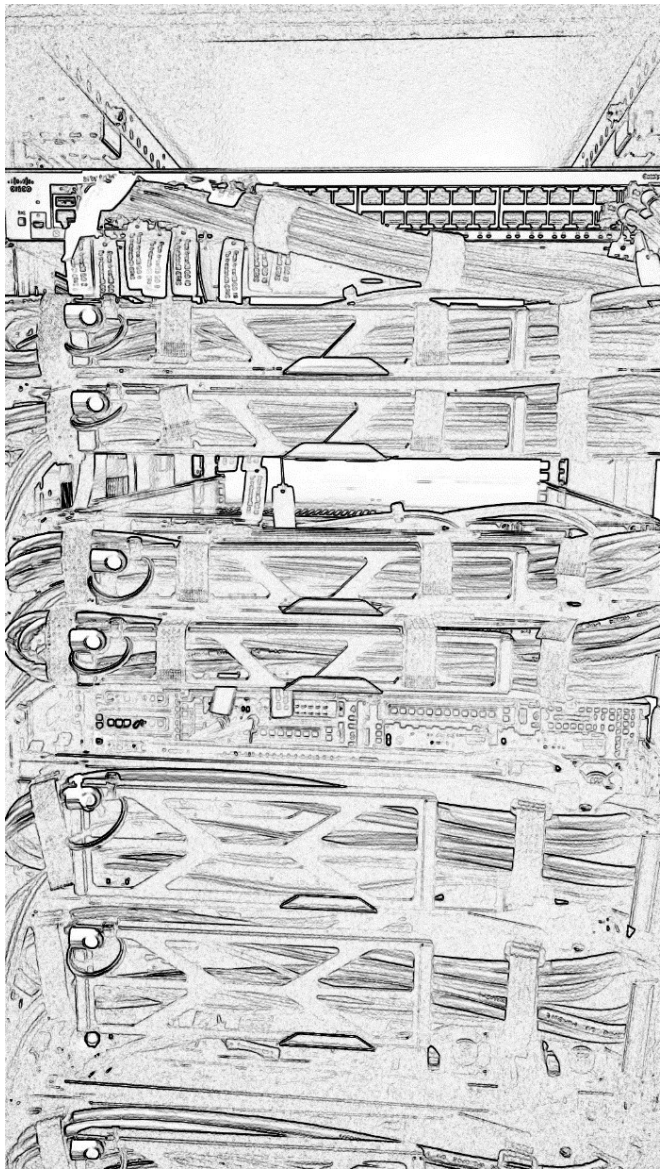
Search head / Master node * 1
indexer * 5

Elastic Stack

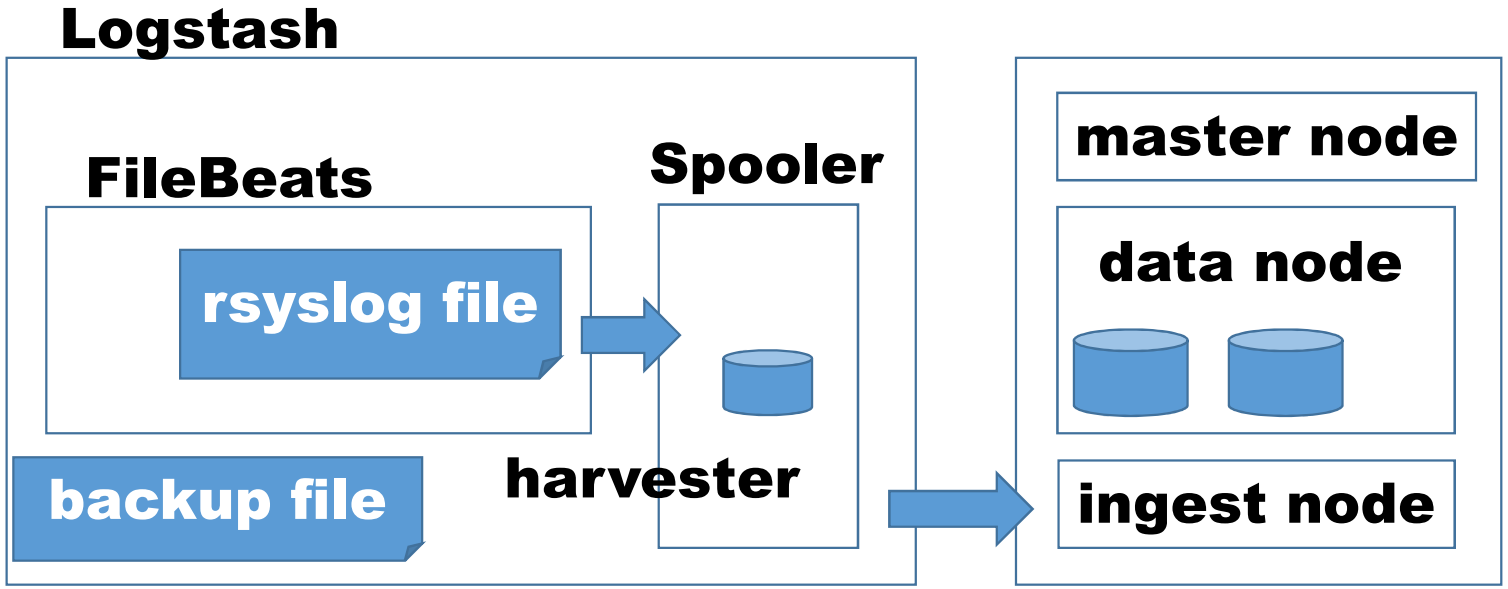
- **Distributed: Elasticsearch is a distributed in nature. Elastic stack has been designed for scaling horizontally, not vertically.**

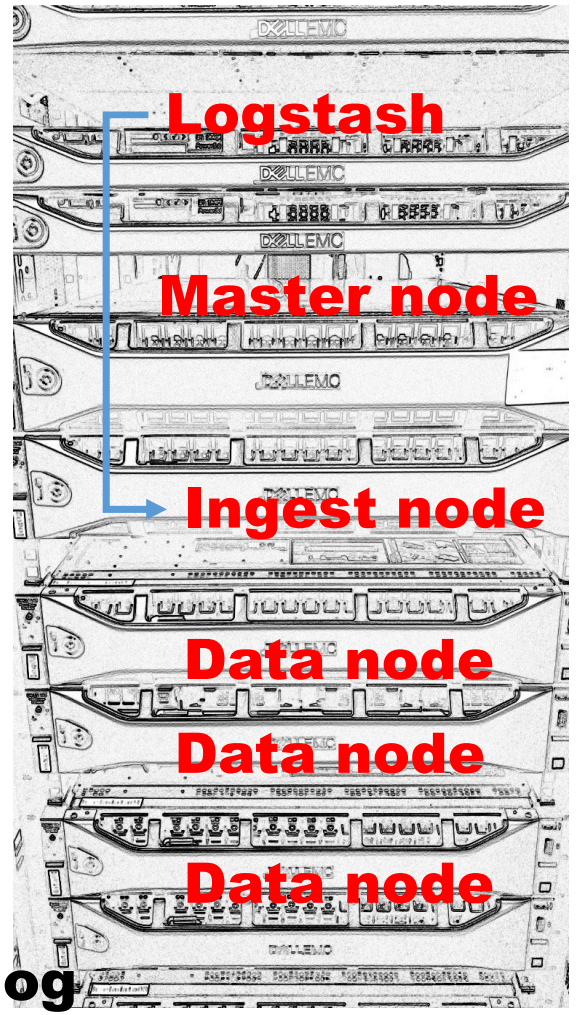
- **High availability**
- **REST-based**
- **Powerful query DSL**
- **Schemeless**

We are running 9 (servers) * 4 (data nodes) = 36 shards.



ELK Stack - From PA-7080 to Logstash via rsyslog





- The Beats is nput plugin which enables Logstash to handle events from the Elastic Beats framework.

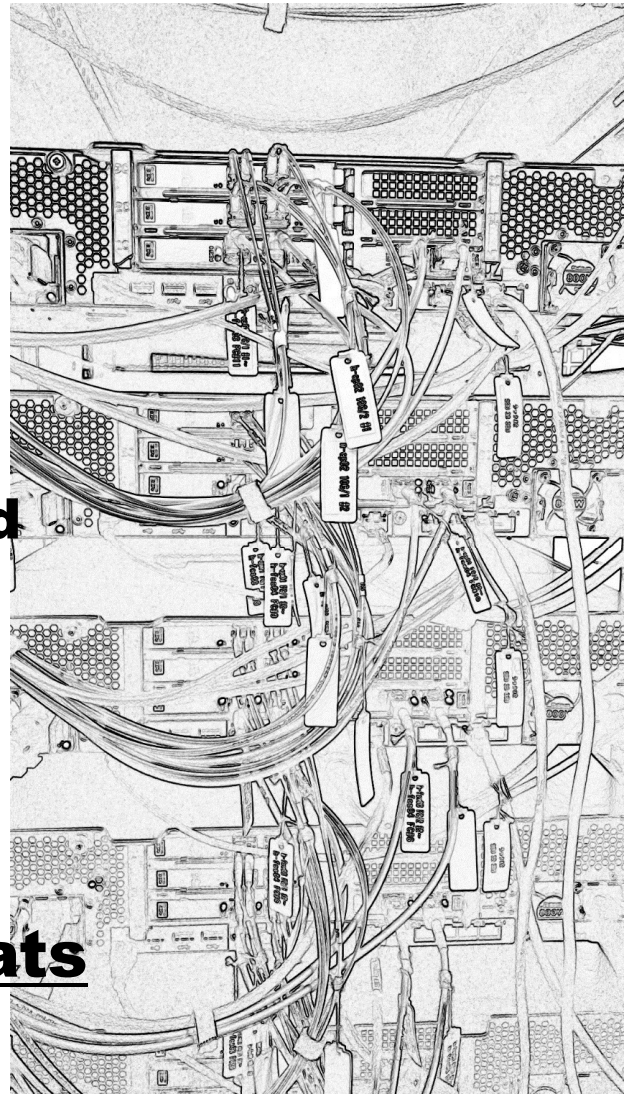
- FileBeat is a lightweight log shipping agent for shipping logs from local files. It is used to monitor log directories and files, and send them to Elasticsearch.

Splunk – Time series indexer

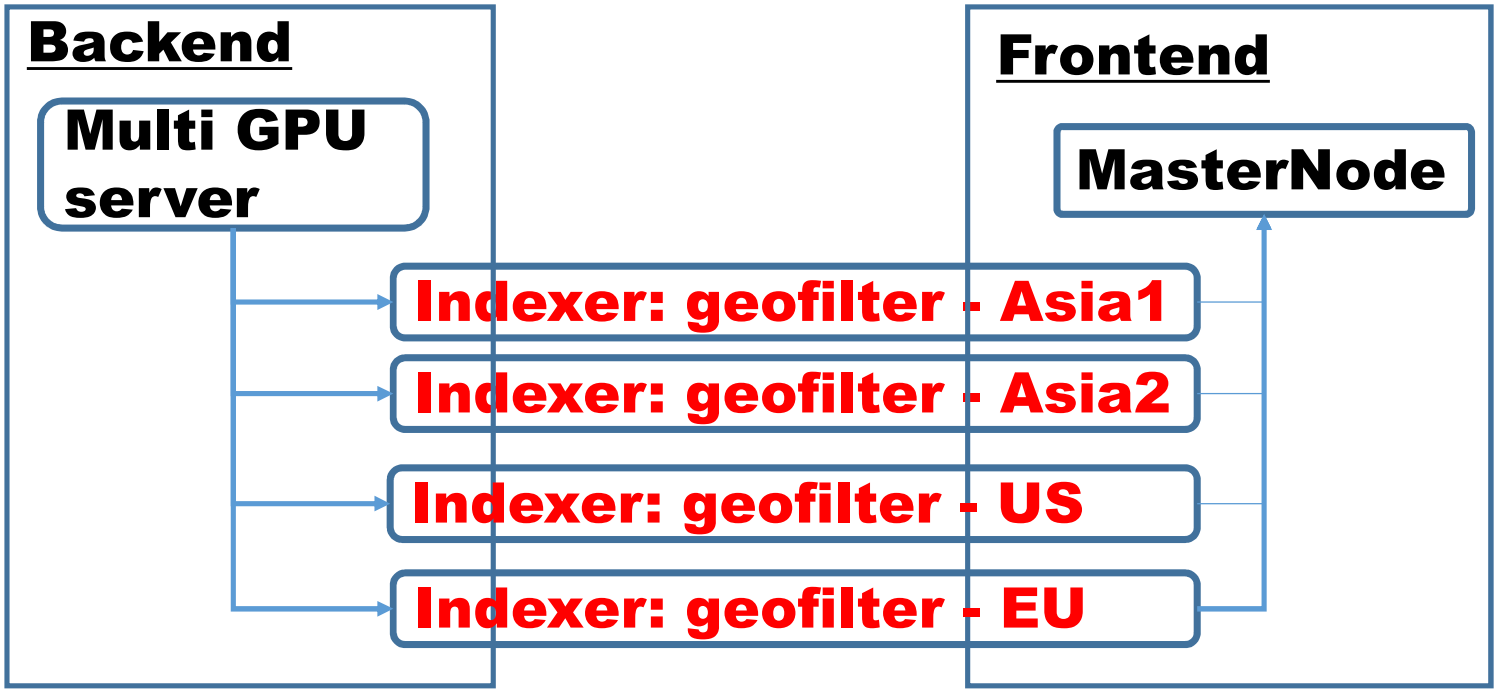
- Splunk has three main functionalities:
  - Data collection
  - Data indexing
  - Search and analysis

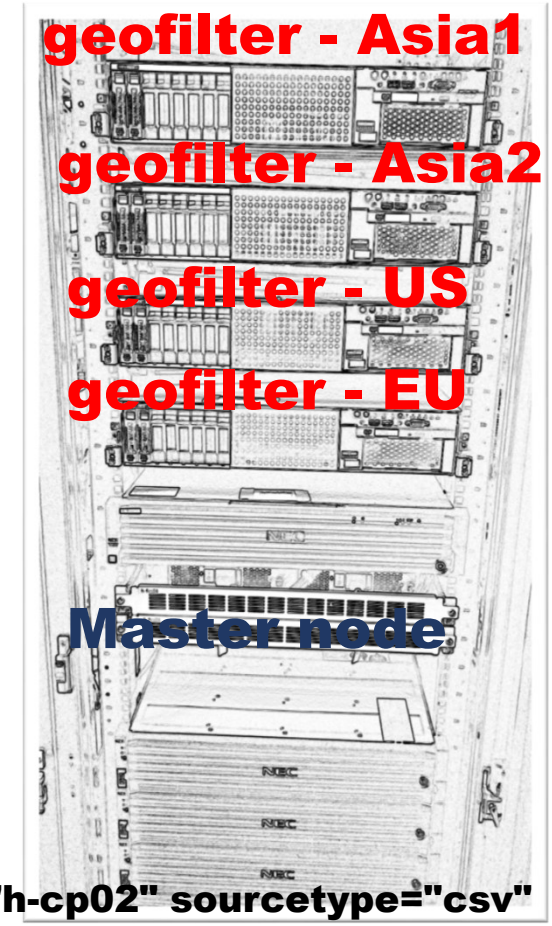
- Being proactive: Splunk's alert can be activated whenever a search is executed on or whenever certain conditions are met.

- Feature-rich SPL commands
  - transaction, concurrency, and streamstats



Splunk - From Multi GPU server to distributed indexer





Geofilter = geostats + streamstats

【geofilter】 source="/mnt/sdc/splunk_direction/dev02/msec-ingress/*" host="h-cp02" sourcetype="csv" earliest=-4d@d latest=-3d@d | iplocation sourceIP | table _time sourceIP City | where City == "CityName" | timechart count span=1m | streamstats window=12 avg(count) as avg stdev(count) as stdev | eval lower_bound = avg - stdev * 2 | eval upper_bound = avg + stdev * 2 | eval isOutlier = if(count>upper_bound OR count<lower_bound, 1, 0) | fields - avg stdev

Insight: The treaty of Westphalia is over

【geofilter】 source="/mnt/sdc/splunk_direction/dev02/msec-ingress/*" host="h-cp02" sourcetype="csv" earliest=-4d@d latest=-3d@d | iplocation sourceIP | table _time sourceIP City | where City == "CityName" | timechart count span=1m | streamstats window=12 avg(count) as avg stdev(count) as stdev | eval lower_bound = avg - stdev * 2 | eval upper_bound = avg + stdev * 2 | eval isOutlier = if(count>upper_bound OR count<lower_bound, 1, 0) | fields - avg stdev

Note: Data (in this talk) is randomly generated !

"2019/07/02 00:00:32.030","2019/07/02 00:00:32","2019/07/02 00:00:32","986","X.X.X.X","5478","XW","Y.Y.Y.Y","3227", "Kj","YwW","n2uXvPwIn","Oo1","rcw5L","uiz9FUNg", "9","HfeQtkBXmuomcUojT6feWqNEtI","31","879","697", "545","199","542","rand-pa1"

3,600,000,000 (729GB)

❒ Random data generator
https://https://github.com/RuoAndo/Usenix_LISA19/generator

❒ Elasticsearch multi-process data exporter
https://https://github.com/RuoAndo/Usenix_LISA19

Bottleneck 1
Huge pagination without scoring

Parallel scroll API invocation

Scroll API - Pagination without scoring

- **Data pagination: We often needs more and more data either to render on a page or to analyze in the backend. Pagination makes it possible to retrieve a limited number of documents from Elasticsaerch.**

- **Scan-scroll: While running Elasticsearch, a functionality which is needed frequently is:**

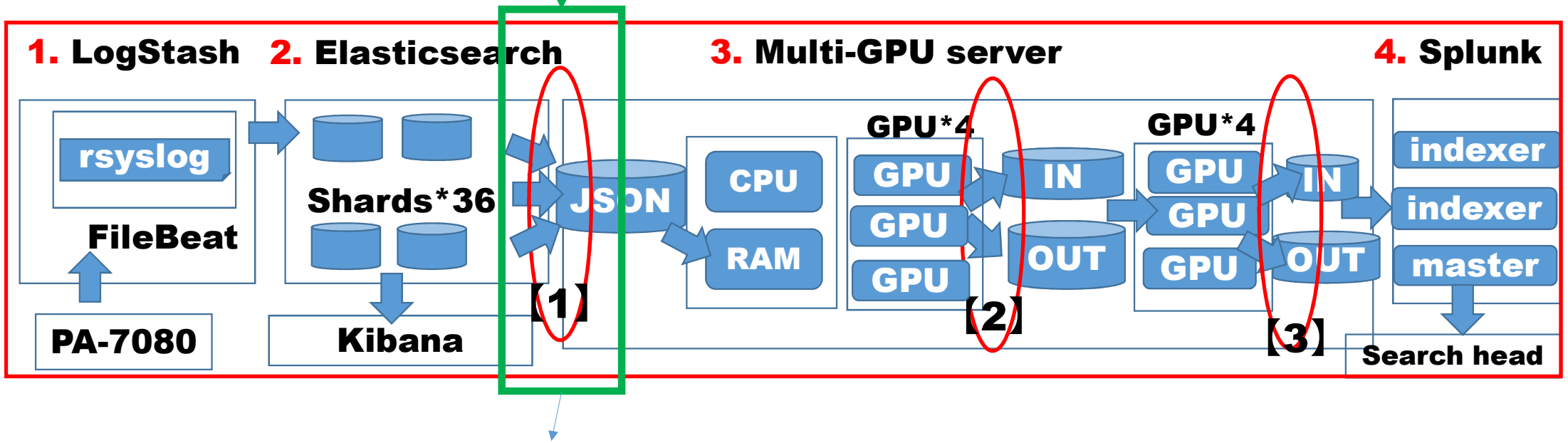
  - Returning a large set of data to analyze

  - To re-index from one index to another

These two types of data retrieval does not require any document scored or sorted.

Bottleneck 1

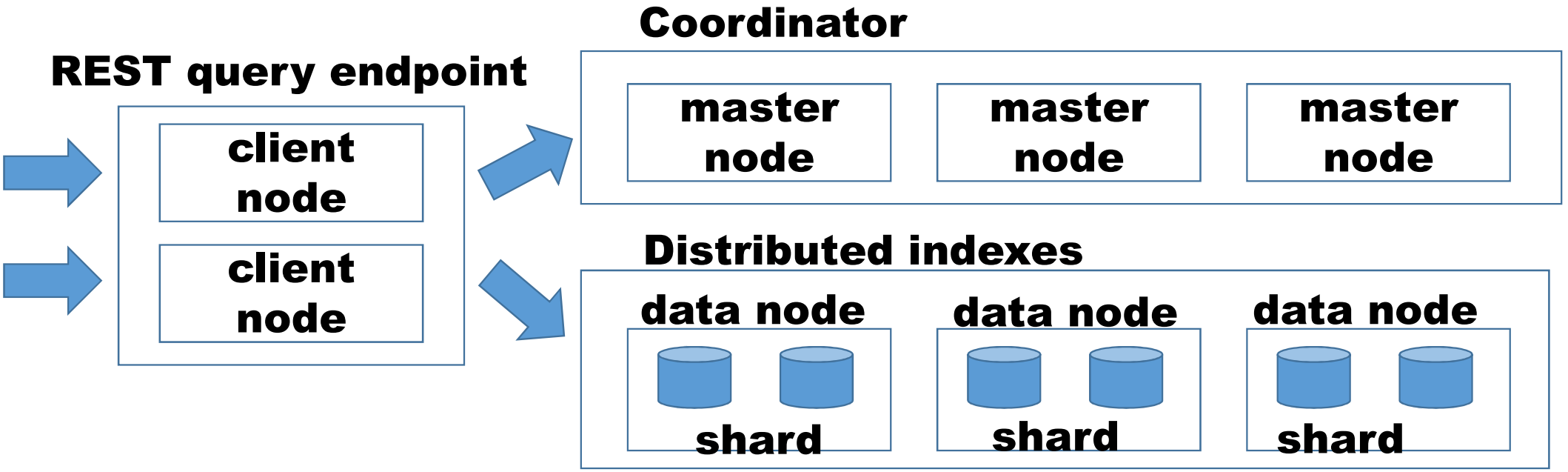
We are here



1664m32.441s -> 132m34.298s

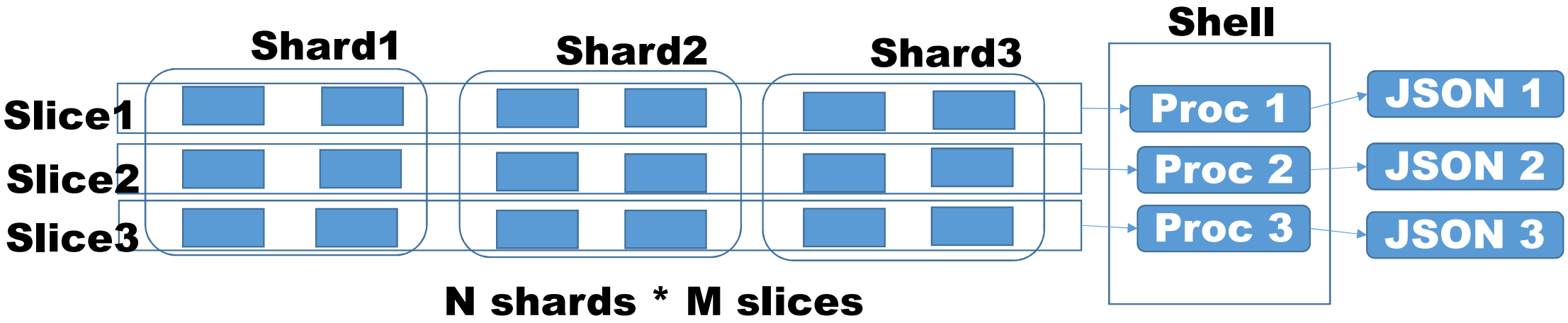
Pitfall 1: It took 1664 minutes to complete the retrieval of session data from Elasticsearch by single process of python script. Our ELK stack has 36 shards and therefore multiple process invocation of scroll API does not impact the resource utilization of Elasticsearch.

Three kinds of Elasticsearch nodes



- **Client node** acts as a query router and a load balancer. A client node can be used to query as well as index processes.
- **Data node** is responsible for holding the data, merging segments and executing queries.
- **Master node** is responsible for the management of the complete cluster.

Multiplexed Scroll API
- How to speed up the retrieval (shards * slices)



- Index and shards: an index is the logical place where data is stored. Each index can be scattered onto multiple Elasticsearch nodes and split into one or more smaller pieces called shards.
- Each process has a slice over multiple shards. If we have M processes and each process queries for N shards, which result in that N*M queries are invoked.

Multiplexed Scroll API - Numerical results

450,000,000 (100GB) with 36 shards

procs	elapsed time
2	1664m32.441s
4	879m58.432s
8	490m50.745s
16	318m43.851s
32	237m25.585s
64	157m3.422s
128	132m34.298s (x12.6)

3,600,000,000 (729GB) with 36 shards

procs	elapsed time
2	5680m59.927s
4	2963m59.851s
8	1718m54.688s
16	1149m11.902s
32	826m25.076s
64	589m55.432s
128	522m50.694s (x10.8)

The ratio of speeding up depends on the performance of the cluster of data nodes. With 36 shards, the speeding up is ranging from 10 to 12 times while the speeding up with 4 shards is from 4 to 5 times.

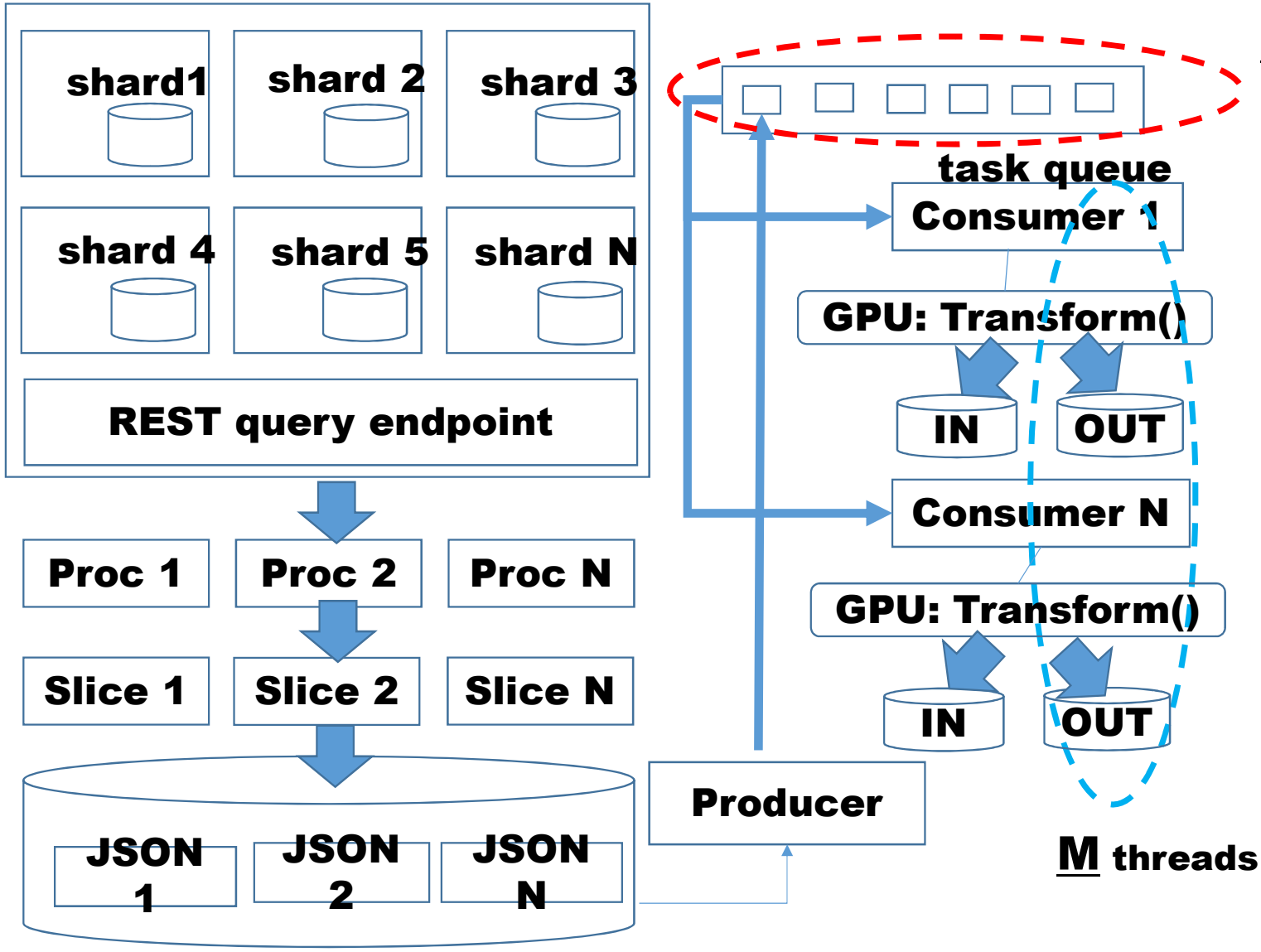
Optimized # of chunks

128 < 64 – 32 < 8

# of threads **# of GPUs**

- **Task decomposition is useful for cases when there are many more pieces of work (# chunks) than threads.**

Parallel exporter likes task decomposition



N chunks (CSV files)

N: 128

# of processes
# of JSON files
# of chunks

M: 8
# of GPU

Dynamic allocation (task decomp) is useful when N > M.

Bottleneck 2
Ingress/egress discrimination

Bulk masking by Thrust transform()

Direction discrimination - massive bitmasking

"2019/07/02 00:00:11.749", sourceIP1, destIP1, "841","25846
"2019/07/02 00:00:47.132", sourceIP2, destIP2, "784","52326
"2019/07/02 01:01:07.338", sourceIP3, destIP2, "912","12947
"2019/07/02 01:01:07.421", sourceIP4, destIP4, "336","50346
"2019/07/02 01:01:11.995", sourceIP5, destIP5, "278","36305
"2019/07/02 00:00:47.132", sourceIP6, destIP6, "784",“50000
"2019/07/02 01:01:17.073", sourceIP7, destIP7, "478","41214
"2019/07/02 01:01:18.987", sourceIP8, destIP8, "365","33646
"2019/07/02 01:01:29.376", sourceIP9, destIP9, "953","60043

X.X.X.X/xx
Y.Y.Y.Y/yy
Z.Z.Z.Z/zz

SINET address range

IF sourceIP1 << xx ∧ X.X.X.X << xx
THEN "2019/07/02 00:00:11.749", srcIP1, destIP1, INGRESS

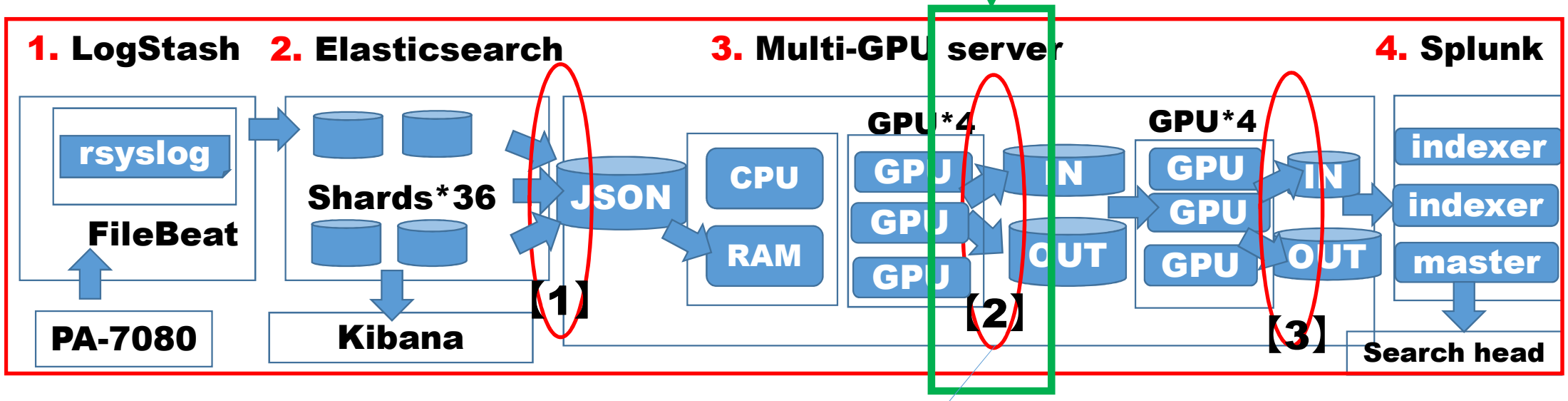
IF destIP2 << xx ∧ Y.Y.Y.Y << xx
THEN "2019/07/02 00:00:47.132", srcIP2, destIP2, EGRESS

<< : (bit) masking * 3,600,000,000 * 2* N

Huge amount of iteration should be processed (N > hundreds)

We are here

Bottleneck 2



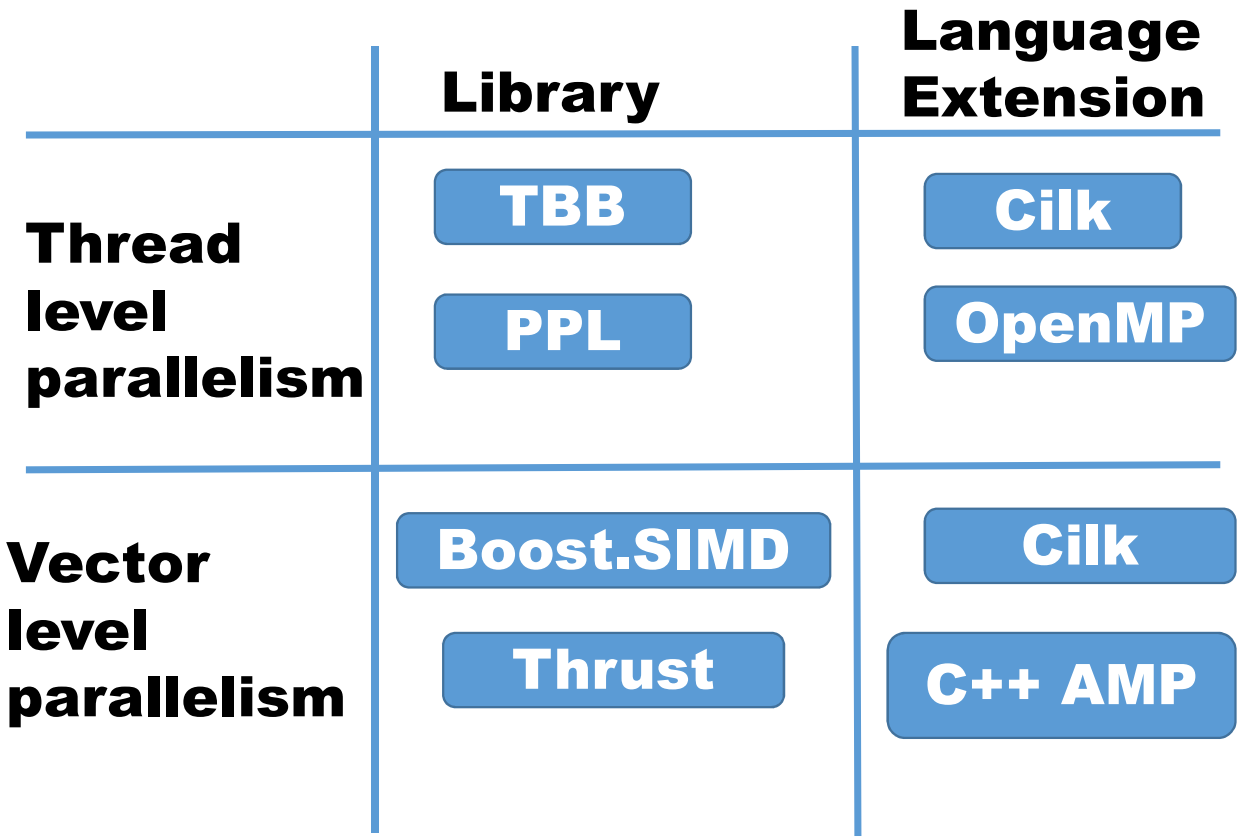
704m47.418s -> 174m7.888s

Bottleneck 2: Session direction discrimination takes unreasonable computing time (704 Minutes) with the bitwise and shift operation on multi-core CPU multithreading.

appended

OUTPUT: "2019/07/02 00:00:11.749", sourceIP1, destIP1, "841","25846", EGRESS

CUDA Thrust



- High Productivity
- High performance
- Interoperability with C/C++
- Generic programming

- **Functor of Bitwise operation with transform ()**

Thrust is a template library that enables a high-level approach to GPU instead of low-layer kernel implementation. Syntax is similar to the way of the standard STL library. Thrust effectively cuts the implementation cost associated with GPU computing.

Simple and rustic - Bulk masking with Thurst

```
1: void discern(unsigned long *Ipaddress,  unsigned long *netmask, unsigned 2:
long address_to_match, double *result, size_t data_size, int thread_id)
3: {
4:     int GPU_number = thread_id;
5:   cudaSetDevice(GPU_number);     //Swtich to GPU 1-8


6:    thrust::transform(Ipaddress_dv.begin(), Ipaddress_dv.end(),
7:    netmask_dv.begin(),   masked_Ipaddress_dv.begin(),
8:    thrust::bit_and<unsigned long>());    //bitmasking

 // Without tidy parameter selection in loop division or data decomp
9:     thrust::transform(masked_IPaddress_dv.begin(),masked_IPaddress_dv.end(),
10:   address_to_match_dv.begin(), result_dv.begin(), thrust::minus<double>());
}
```

(sourceIP[] << xx) - (X.X.X.X[] << xx) = result[] (0: ingress)

Insight: GPGPU is handaxe

Cut!

476m41.308s

1664m32.441s

Reduce!

Expose!

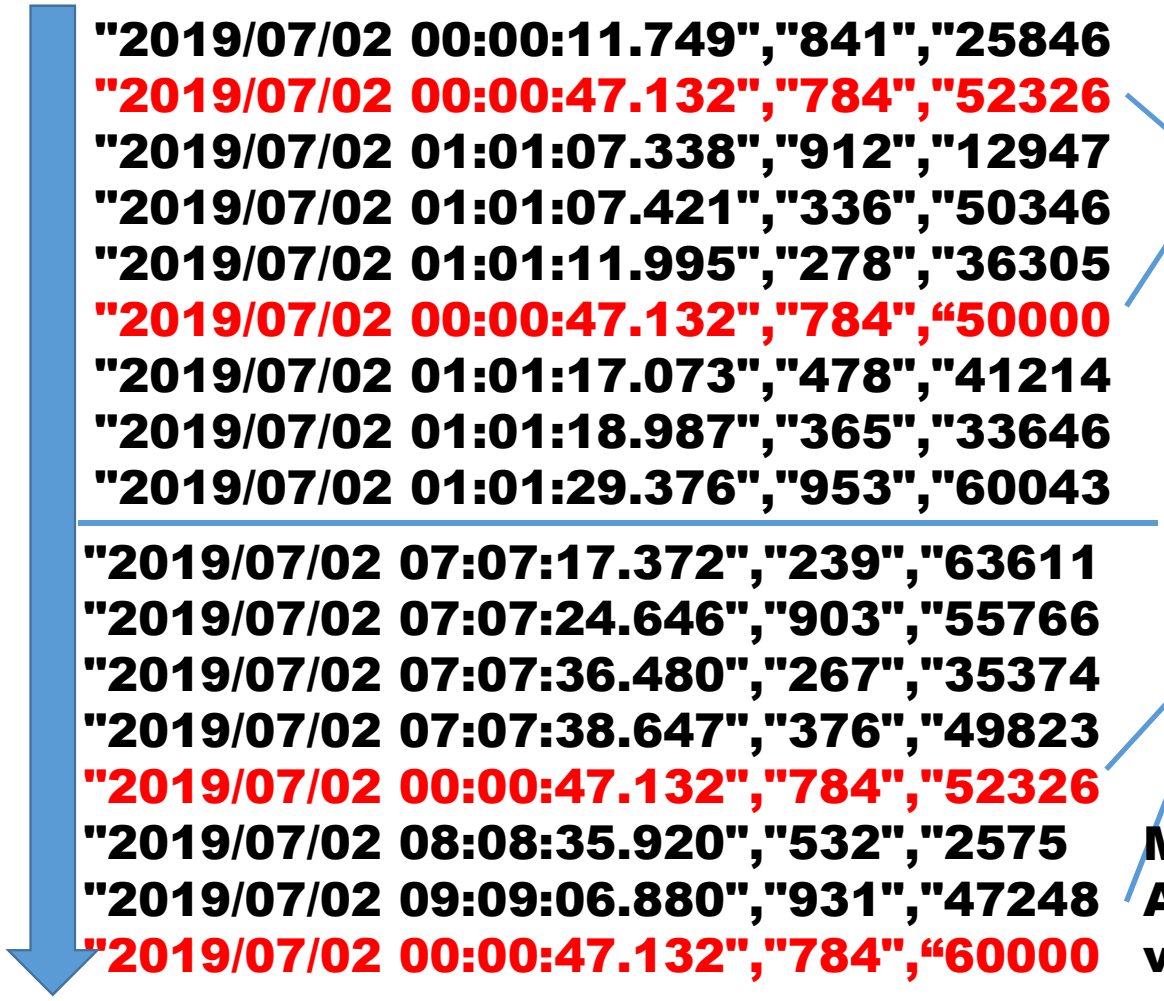
704m47.418s

For thousands of centuries, over much of the planet, a handaxe was de rigueur. --
The digital ape is a direct descendant of the axe-wielding hominin.

Nigel Shadbolt, Roger Hampson, "The Digital Ape: how to live (in peace) with smart machines"

Bottleneck 3
Histogramming in milliseconds
Map-Reduce and Tiling

Map-Reduce - Billions by millisecond



"2019/07/02 00:00:11.749","841","25846
"2019/07/02 00:00:47.132","784","52326
"2019/07/02 01:01:07.338","912","12947
"2019/07/02 01:01:07.421","336","50346
"2019/07/02 01:01:11.995","278","36305
"2019/07/02 00:00:47.132","784",“50000
"2019/07/02 01:01:17.073","478","41214
"2019/07/02 01:01:18.987","365","33646
"2019/07/02 01:01:29.376","953","60043

"2019/07/02 07:07:17.372","239","63611
"2019/07/02 07:07:24.646","903","55766
"2019/07/02 07:07:36.480","267","35374
"2019/07/02 07:07:38.647","376","49823
"2019/07/02 00:00:47.132","784","52326
"2019/07/02 08:08:35.920","532","2575
"2019/07/02 09:09:06.880","931","47248
"2019/07/02 00:00:47.132","784",“60000

3,600,000,000 (729GB)

①Fine reduction

"2019/07/02 00:00:47.132",“102326”

②Coarse reduction

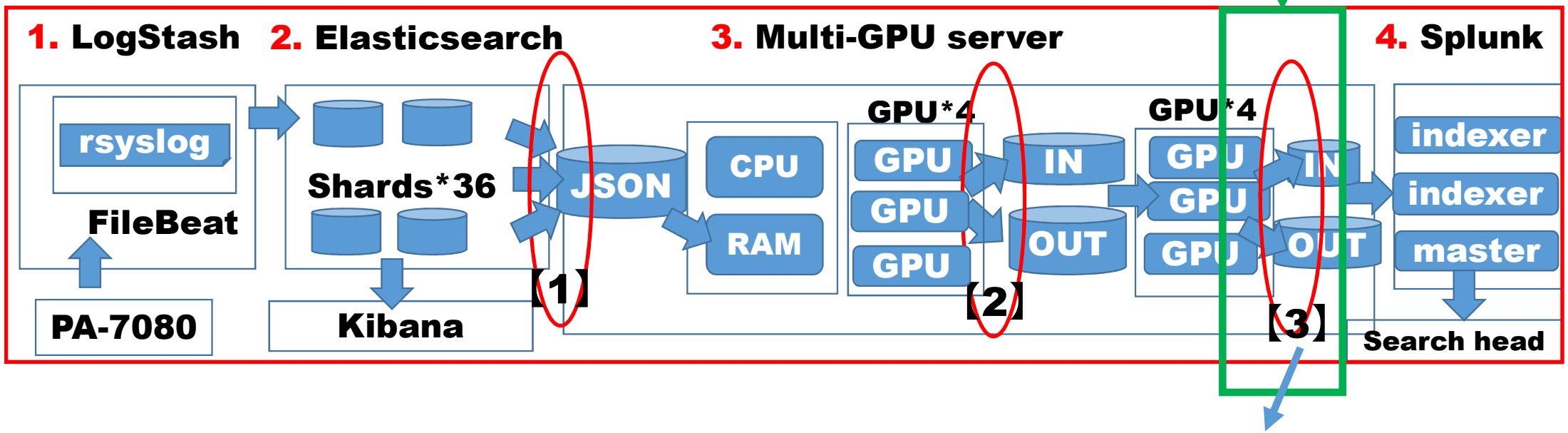
"2019/07/02 00:00:47.132",“214652”

"2019/07/02 00:00:47.132",“112326”

①Fine reduction

MAP: {timeseries, bytes}
A collection of data items and associated value with each item in the collection.

We are here — Bottleneck 3



476m41.308s -> 12m34.706s

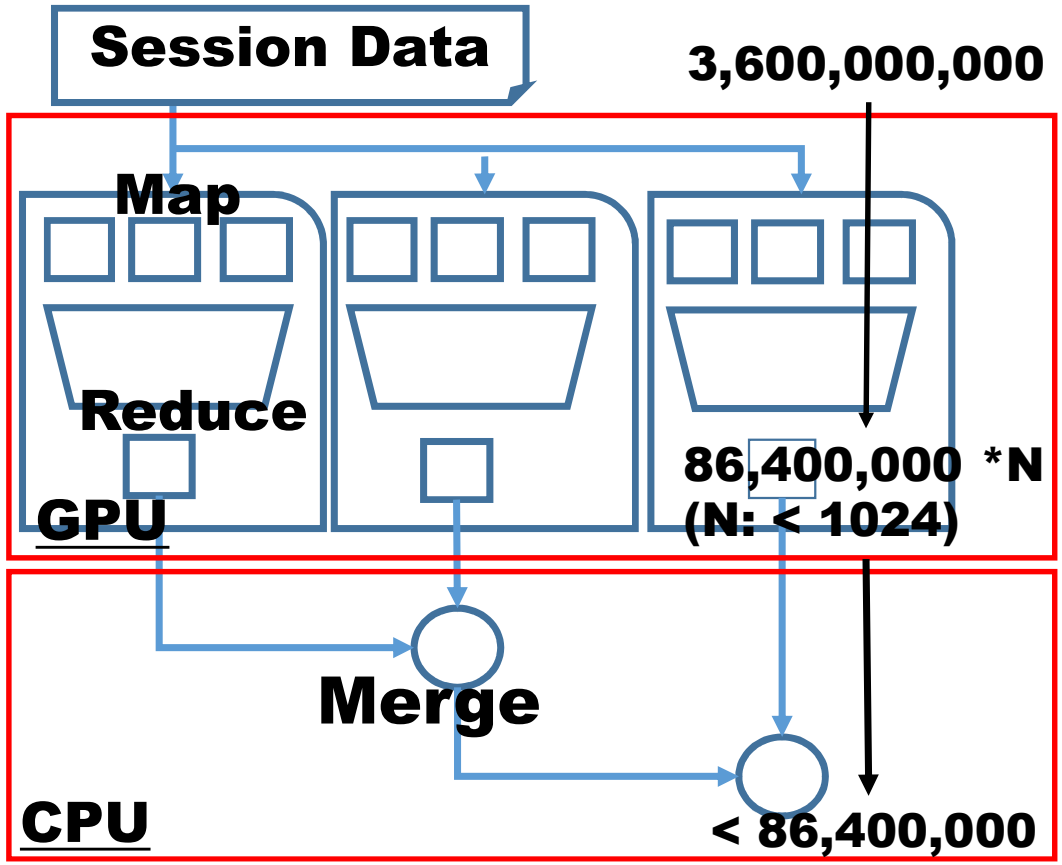
Bottleneck 3: Time histogramming on server side took unreasonable computing time (476 minutes).

"2019/07/02 00:00:47.132", sourceIP1, destIP2, "784", ingress
"2019/07/02 00:00:47.132", sourceIP3, destIP4, "700", ingress

↓

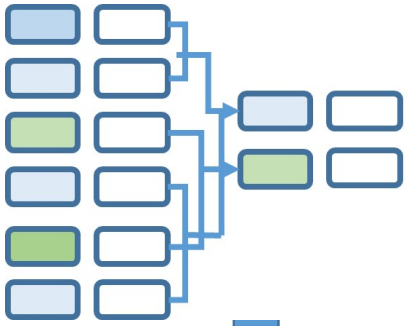
"2019/07/02 00:00:47.132", “1484", INGRESS

Map reduce + Tiling (Blocking)



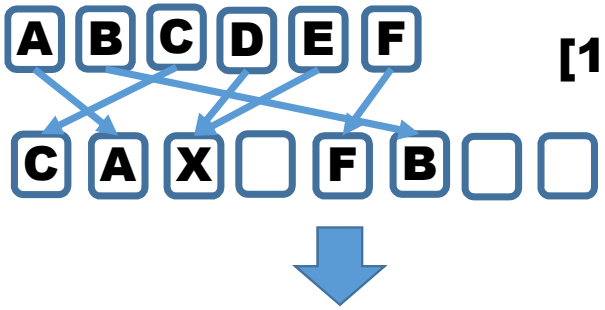
Tiling is dividing a process into a set of parallel tasks of a suitable granularity.

Pairwise reduction pattern
CUDA Thrust (reduce_by_key())



Binary operation to reduce an input sequence to a single value.

Merge scatter pattern
Intel TBB(Concurrent Hashmap)

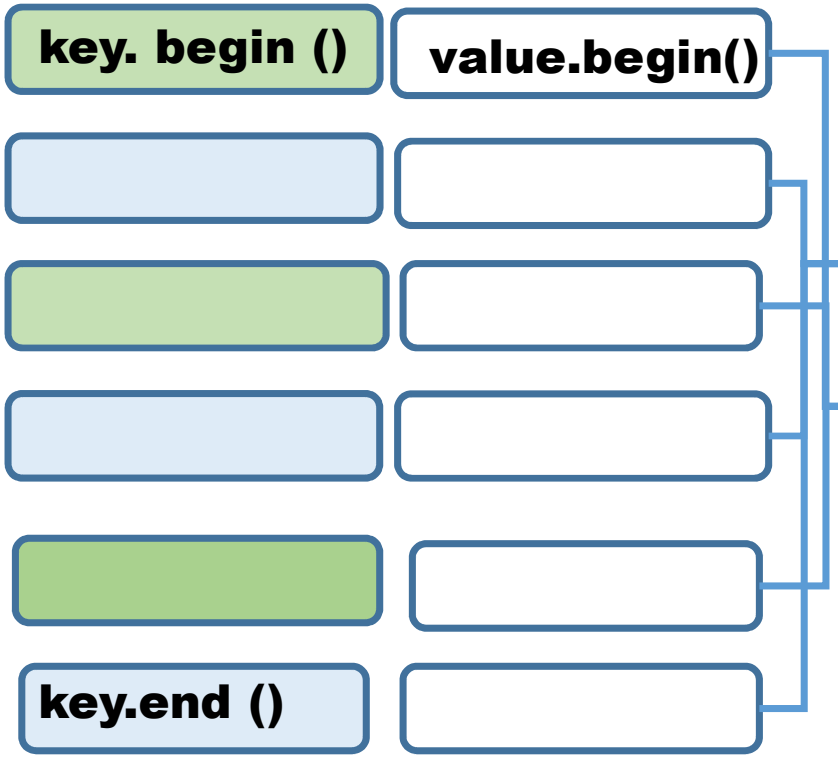


[1, 5, 6, 2, 2, 4]

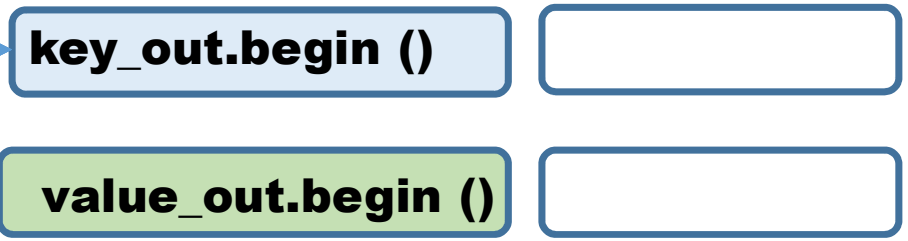
"2019/07/02 00:00:47.132",“214652”

Pairwise reduction pattern - CUDA Thrust

KEY[0]="2019/07/02 00:00:11.749" / VALUE[0]="742"



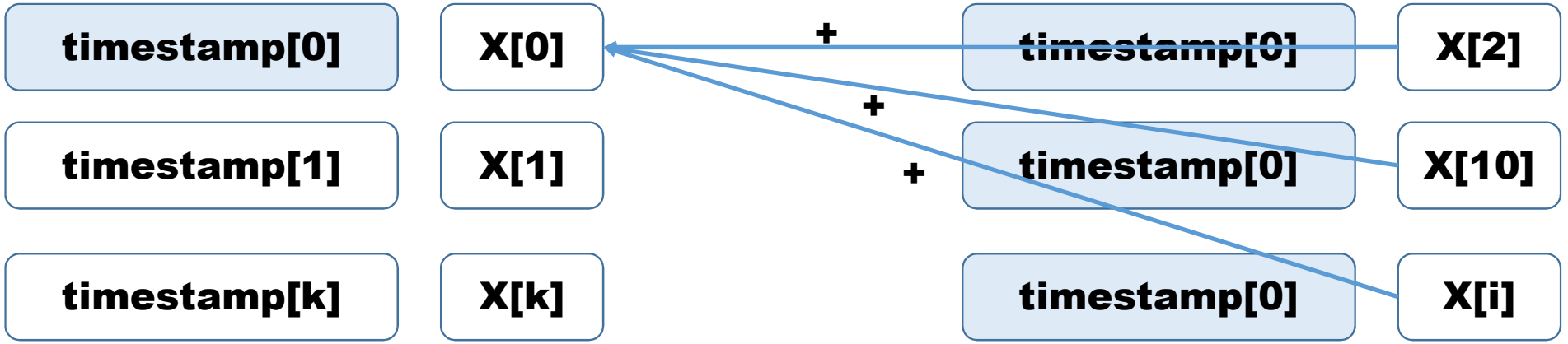
Reduce_by_key() is a generalization of reduce to key-value pairs.




```
auto new_end = thrust::reduce_by_key(
d_vec_key.begin(),
d_vec_key.end(),
d_vec_value.begin(), d_vec_key_out.begin(),
d_vec_value_out.begin());
```

Merge scatter pattern - Intel TBB

```
1: typedef tbb::concurrent_hash_map<unsigned long, long>
2: iTbb_timestamp;
3: static iTbb_timeseries Tbb_timeseries;
```



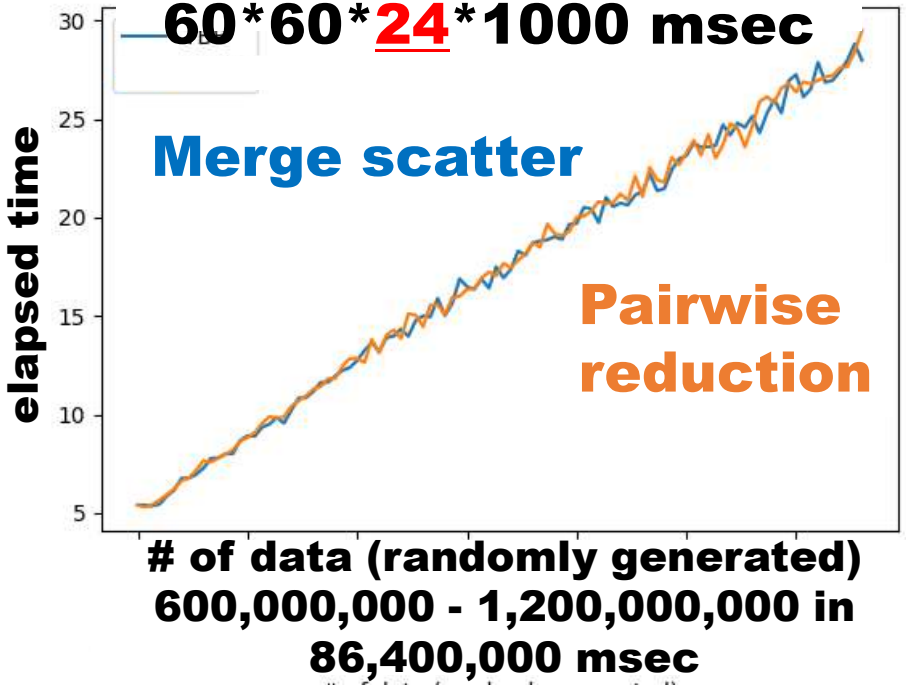

```
1: iTbb_timestamp :: accessor tms ;
2: Tbb_timestamp. insert (tms , key out [ i ]);
3: tms->second += value out [ i ];
```

We have {key[i], value[i]} and if key[i] (timestamp[i]) is identical to key[k], then value[i] is added by the value of value[k].

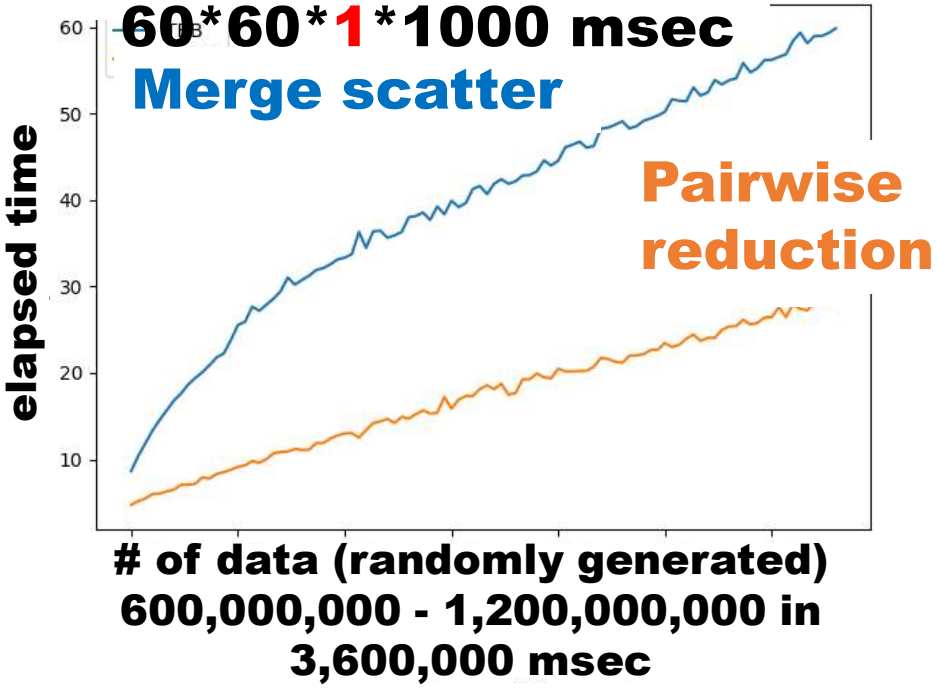
In a merge scatter, outputs which collide while implementing a scatter pattern are combined with an associative operator.

Merge scatter and pairwise reduction

Sparse: x / 60*60*24*1000 msec



Dense: x / 60*60*1*1000 msec



As the density of session data (points/interval) is increasing, pairwise reduction is getting faster. Run this command:

https://github.com/RuoAndo/Usenix_LISA19/blob/master/misc/test.sh

Merge scatter and pairwise reduction

- **GPU and CPU are complementary.**

- **Resolving collisions in merge scatter pattern comes at a cost in some cases.**

- **As data density in time-series is getting high, pairwise reduction gains an advantage (faster).**

- **On the other hand, highly concurrent container is only effective and reasonable way to share data among multiple threads.**

```
1: typedef tbb::concurrent_hash_map<long, int> iTbb_Vec_timestamp;
2: static iTbb_Vec_timestamp TbbVec_timestamp;
```

0.Static global variables

```
3: void Pthread_comsumer()
4:  size_t kBytes = data.size() * sizeof(unsigned long long);
5:  unsigned long long *key;
6:  key = (unsigned long long *)malloc(kBytes);
8:  reduction(key, value, key_out, value_out, kBytes, vBytes, data.size(),
&new_size,thread_id);
10:  iTbb_Vec_timestamp::accessor tms;
11:  TbbVec_timestamp.insert(tms, key_out[i]);
12:  tms->second += value_out[i];
```

1. Calling GPU function

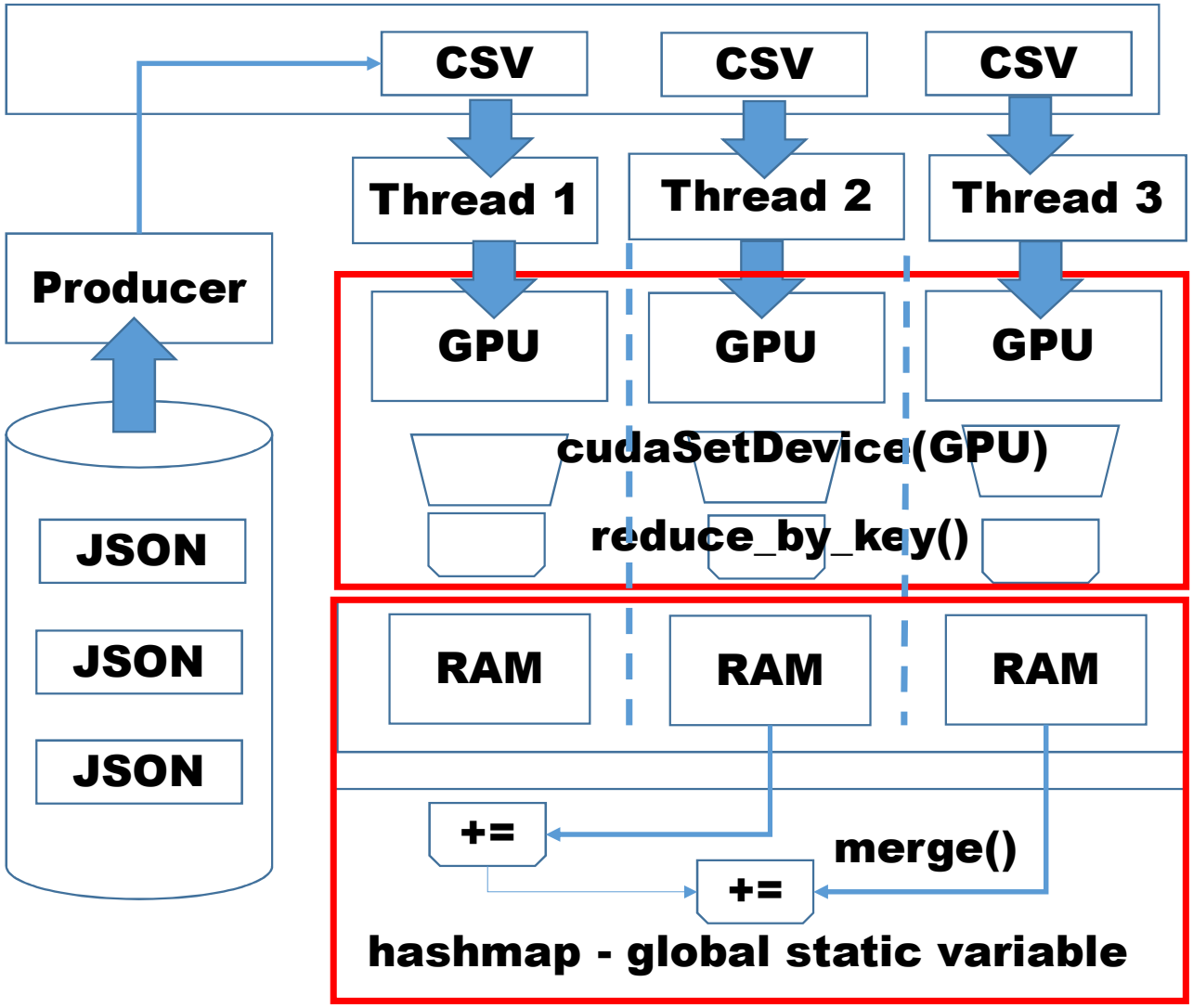
4.Hashmap insertion

```
13:void reduction(unsigned long long *key, long *value, unsigned long long *key_out, long
*value_out, int kBytes, int vBytes, size_t data_size, int *new_size, int thread_id)
14:
15:  cudaSetDevice(thread_id);
17: thrust::sort_by_key(d_vec_key.begin(), d_vec_key.end(), d_vec_value.begin());
18:
19:  auto new_end = thrust::reduce_by_key(d_vec_key.begin(), d_vec_key.end(),
  d_vec_value.begin(), d_vec_key_out.begin(), d_vec_value_out.begin());
```

2. Switch to GPU(N)

3. Calling ThrustAPI

Histogramming - MultiGPU acceleration (Tesla V100)



single GPU and 4 threads - 9GB (400,000,000) *4 =36GB		
	memory transfer (sec)	reduction (sec)
GPU0	52.76937812	0.334865648
GPU0	48.60224942	out of memory
GPU0	50.43252043	out of memory
GPU0	49.95634021	out of memory

multi GPU and 4 threads - 9GB (400,000,000) * 4 =36GB		
	memory transfer (sec)	reduction (sec)
GPU0	52.85998047	0.344755277
GPU1	50.19232587	0.337464371
GPU2	49.0351076	0.335592131
GPU3	51.06696428	0.33519815

4 GPUs vs 1 GPUs - Tesla V100 (32510MiB)

1 GPU	4 GPU	Slowdown
4GB (20,000,000) * 4 = 16GB		
0.603212162	0.169472176	
2.017158832	0.169368701	
1.266480964	0.169198073	x7.8
4.328394731	0.172328982	x25.4
6GB (30,000,000) * 4 = 24GB		
0.25455562	0.266662563	
1.09780012	0.254225929	
3.405360669	0.263795581	x13.07
2.618192345	0.254559077	x10.44
9GB (40,000,000) * 4 = 36GB		
0.334865648	0.344755277	
out of memory	0.337464371	
out of memory	0.335592131	
out of memory	0.33519815	

- In worst case, single GPU is 25.4 times slower.
- Thrust API is fully asynchronous and the contention may have been occurred when we don't apply multi GPU.

*Streams are not adopted.

- Single GPU can HARDLY handle no more than 36GB session data.

Speed up with multi GPU (more than 36GB)

120GB

30GB (100,000,000) * 4 = 120GB (single thread)		
GPU0 (thread0)	memory transfer hostToDevice	121.516296313 sec
GPU0 (thread0)	reduce_by_key	1.224606337 sec
GPU0 (thread0)	memory transfer deviceToHost	39.981257910 sec
GPU0 (thread0)	hashmap insertion	0.048758897 sec
total elapsed time: 177m42.066s (multi threads)		
GPU0 (thread0)	memory transfer hostToDevice	121.275596159 sec
GPU1 (thread1)	reduce_by_key	1.223066522 sec
GPU2 (thread2)	memory transfer deviceToHost	44.010355884 sec
GPU3 (thread3)	hashmap insertion	0.051888012
total elapsed time: 103m19.728s (x1.71)		

Total: 720GB: (177-103) * 6 = 384minutes speedup

It is estimated that for coping with 720GB (3600,000,000 session data), Multi CPU can speed up the histogramming by ranging from 450 to 520 minutes.

Insights: 86,400,000 and 5–7 Billion

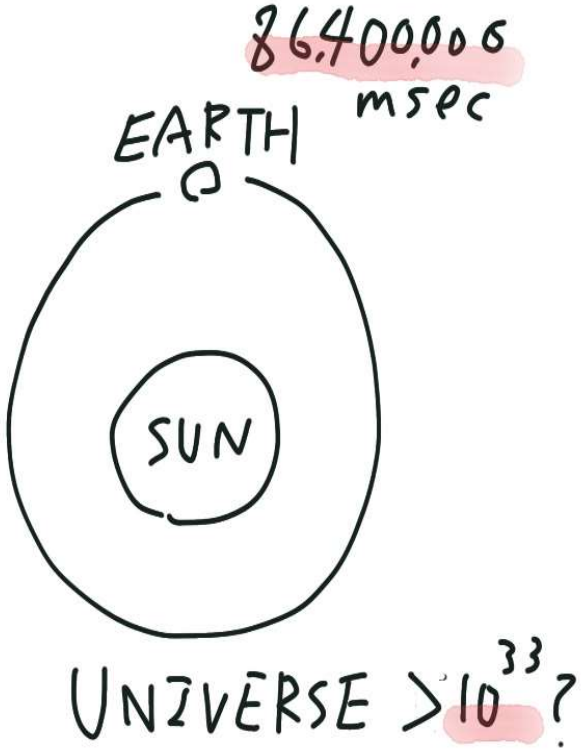
❒ **GPU fits the exploding universe of data.**

❒ For the next decade, the universe of data will be exploding.

❒ Until after 5–7 Billion, 1 day is 86,400,000 msec.

→ Data density per 24 hour will be increasing.

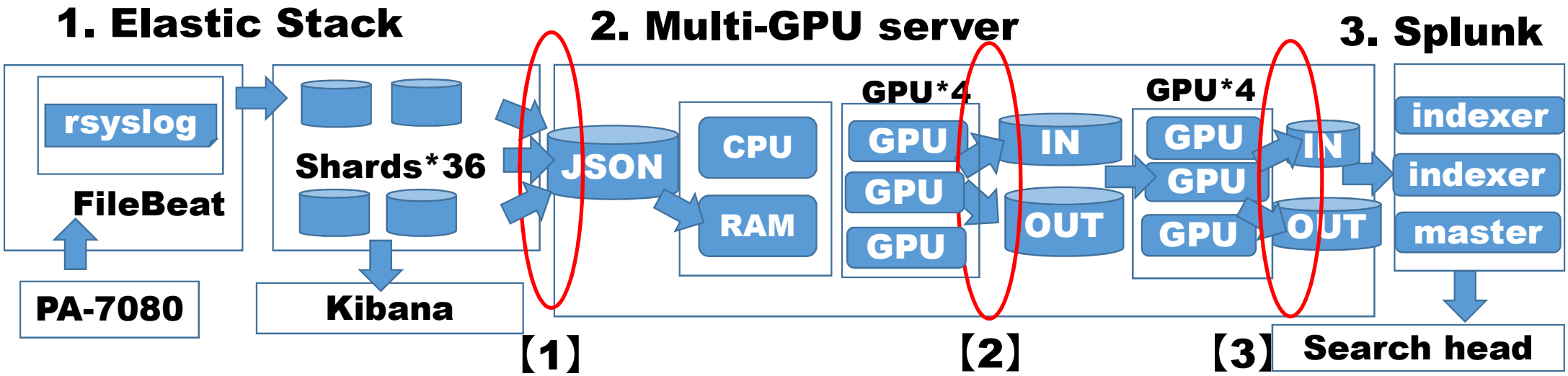
❒ Reduction on CUDA GPUs will be more and more effective as the density of time-series data is increasing.



* "~ 5–7 Billion - End of the Sun" - The Space Book: From the Beginning to the End of Time: 250 Milestones in the History of Space & Astronomy (Sterling Milestones) , Jim Bell

Conclusion and impressions

Multi GPU acceleration on the go



1664m32.441s *704m47.418s* *476m41.308s*

1440m (24 hours) !

132m34.298s(12.6x) 174m7.888s(4.04x) 12m34.706s(39.6x)

128 processes * 36 shards
< 512GB * 9

32-64 threads
< 64GB (RAM)

4-8 threads
< 32GB (GPU)

3374m (= 1634 + 704 + 976) / 318m (132 + 174 + 12) = 10.6 (x)

Explosion of the digital universe

- "The universe is not complicated, there's just a lot of it."
  - Richard Feynman

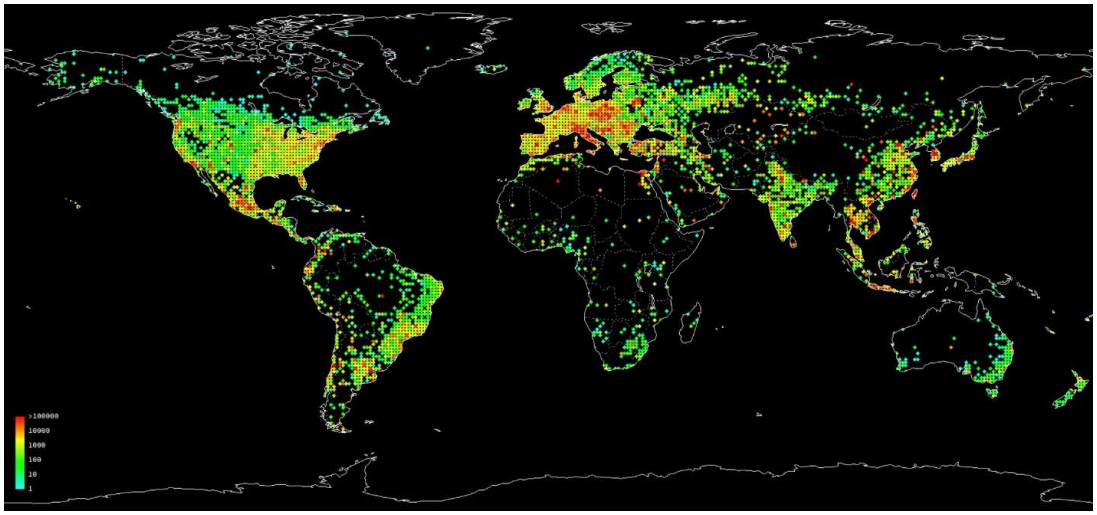
- **Choosing appropriate level of parallelism**

  If a algorithm cannot work with a training of a million examples, then the intuitive conclusion follows that it cannot work at all. However, it has become obvious that the algorithm using a huge dataset with a trillion items can be highly effective in tasks for which the algorithm using a sanitized (clean) dataset with a only million items is NOT useful.

REFERENCE[1]: 1. A. Halevy, P. Norvig, and F. Pereira (2009), “The Unreasonable Effectiveness of Data,” IEEE Intelligent Systems (March–April): 8–12.
RERERENCE[2]: Murray Shanahan, "The Technological Singularity"

GPU for human being – Via negativa

"And then is heard no more: it is a tale told by an idiot, full of sound and fury, signifying nothing"
Macbeth - William Shakespeare



- ❒ **Machine scales. But human does not.**
- ❒ **Don't analyze the ghosts. Just filetr it out!**

Conclusion

- I have reported our three years (painful) operational experience in deploying multi-GPU accelerated monitoring system of huge academic backbone network.

- Some arts of concurrency (huge pagination, bulk bitmasking, and tiled map-reduce) have been introduced.

- It turned out that multi GPU acceleration are essential for the sophistication of the workflow between Elastic stack and Splunk.

Thank you for listening !

https://github.com/RuoAndo/Usenix_LISA19