How to Upgrade Major Version of Your Production PostgreSQL

2018/12/12 PGConf.ASIA

Keisuke Suzuki
Software engineer

Who am I?

Keisuke Suzuki

- Backend Engineer @ Treasure Data K.K.
  - PlazmaDB: distributed storage
  - Datatank: data mart

    Both uses PostgreSQL internally

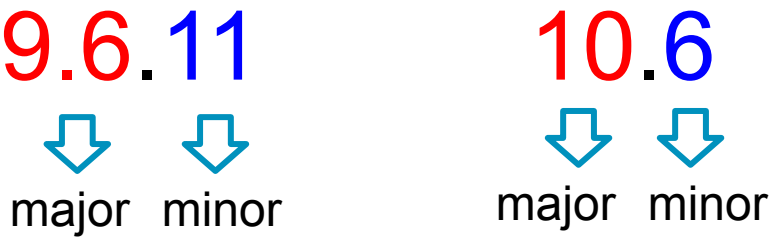
- Interest: DB / Distributed system / Performance optimization

- Twitter: @yajilobee

PostgreSQL Versions

PostgreSQL Versions

- Major version: released yearly - new features
  - DB data are not compatible between different major versions
- Minor version: every 3 months at least - bug & security fixes
  - DB data are compatible if major versions are the same



EOL of Major Versions

The PostgreSQL Global Development Group supports a major version for 5 years after its initial release.

Version	Current minor	Supported	First Release	Final Release
11	11.1	Yes	October 18, 2018	November 9, 2023
10	10.6	Yes	October 5, 2017	November 10, 2022
9.6	9.6.11	Yes	September 29, 2016	November 11, 2021
9.5	9.5.15	Yes	January 7, 2016	February 11, 2021
9.4	9.4.20	Yes	December 18, 2014	February 13, 2020
9.3	9.3.25	No	September 9, 2013	November 8, 2018

https://www.postgresql.org/support/versioning/

Q: Should we follow the latest Major Version?

A: Depends on the case
-> Upgrading requires downtime and may cause incompatibility

- Extended support is provided by venders
  - Mission critical systems
- SaaS venders may stop providing old major versions
  - e.g.) 9.3 retirement on Amazon RDS
    - ✓ Stop Deployment: Aug. 2018
    - ✓ Force Major Version Upgrade: Nov. 2018

Our Version

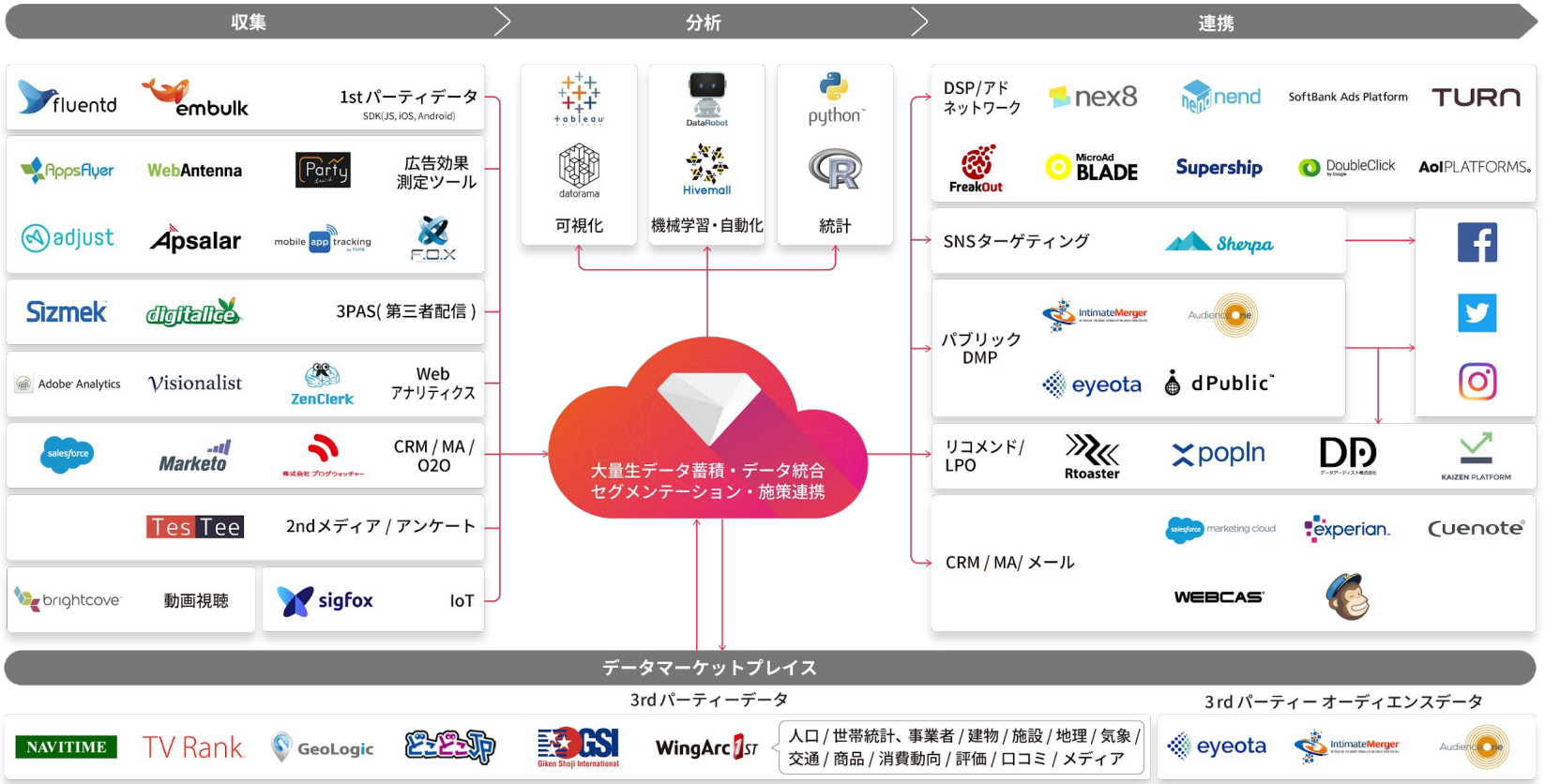
Version	Current minor	Supported	First Release	Final Release
11	11.1	Yes	October 18, 2018	November 9, 2023
10	10.6	Yes	October 5, 2017	November 10, 2022
9.6	9.6.11	Yes	September 29, 2016	November 11, 2021
9.5	9.5.15	Yes	January 7, 2016	February 11, 2021
9.4	9.4.20	Yes	December 18, 2014	February 13, 2020
9.3	9.3.25	No	September 9, 2013	November 8, 2018

We were here until May 2018...

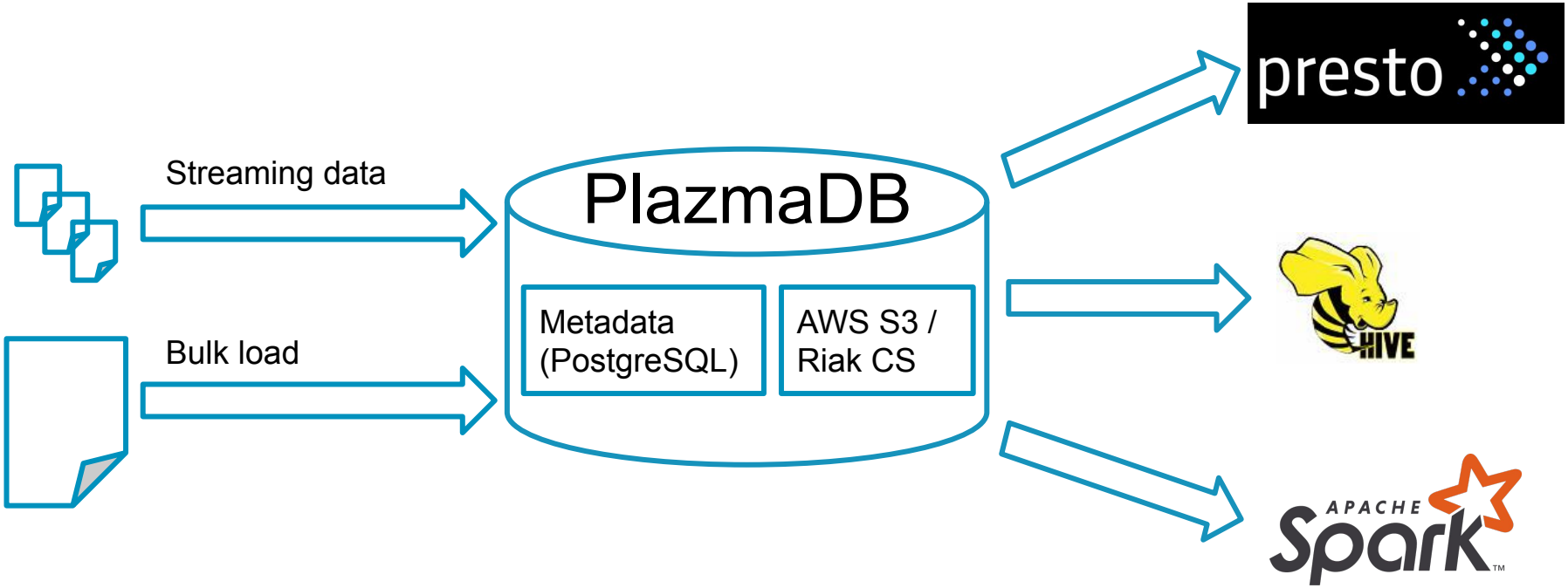
PostgreSQL Usage in Treasure Data

arm

Arm Treasure Data eCDP



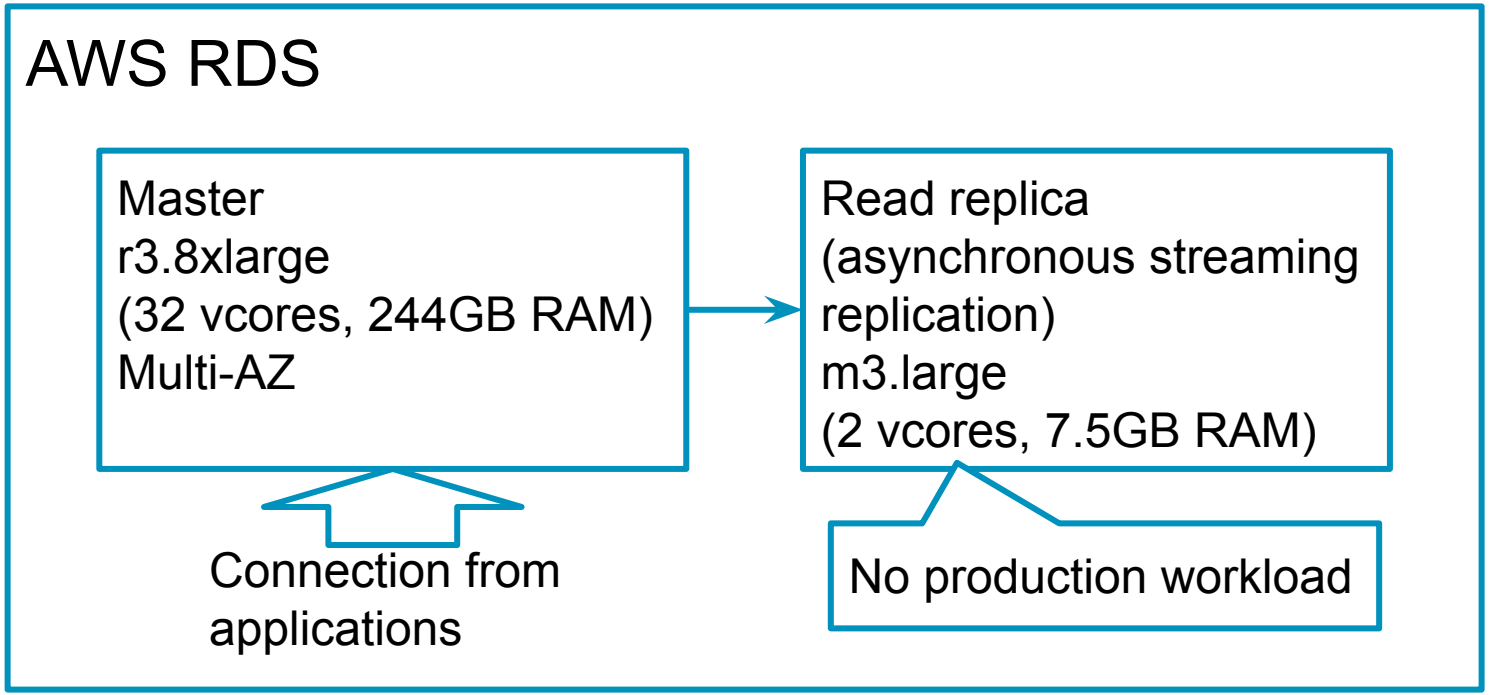
PlazmaDB: Core Storage Layer of TD



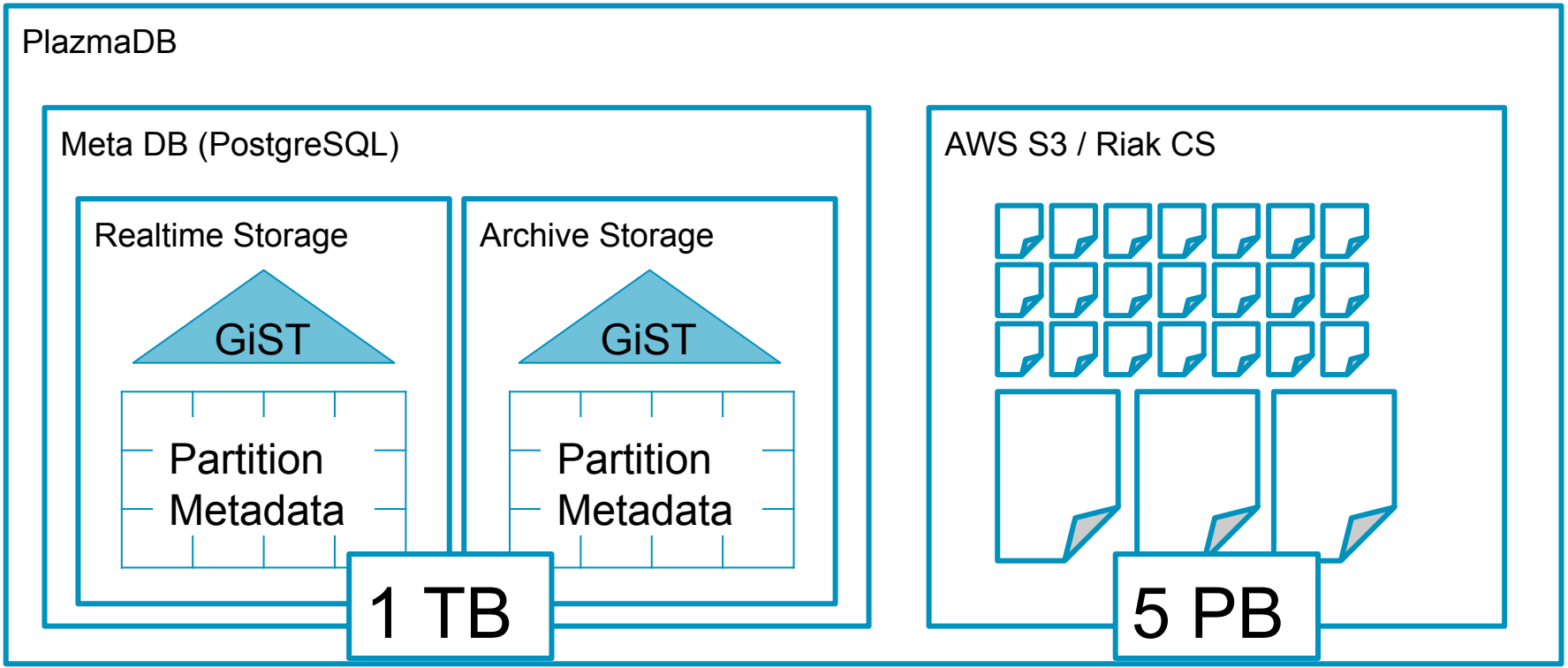
Daily Workload & Storage Size of PlazmaDB

Import	Analytical Query	Storage size
500 Billion Records / day ~ 5.8 Million Records / sec	600,000 Queries / day 15 Trillion Records / day	5 PB (+5~10 TB / day) 55 Trillion Records

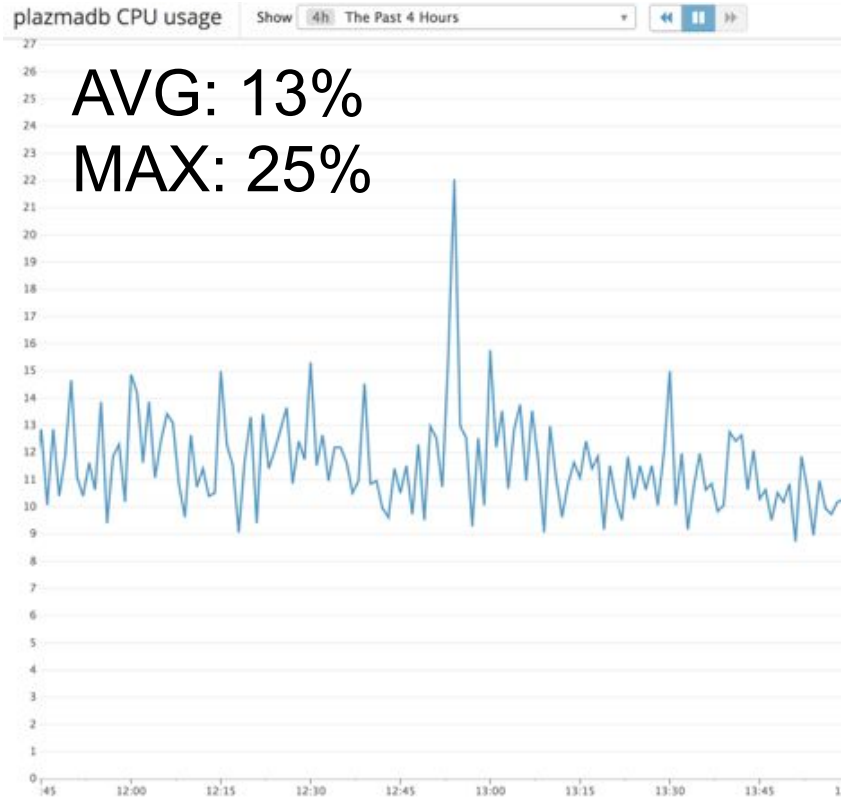
Server Structure of Meta DB

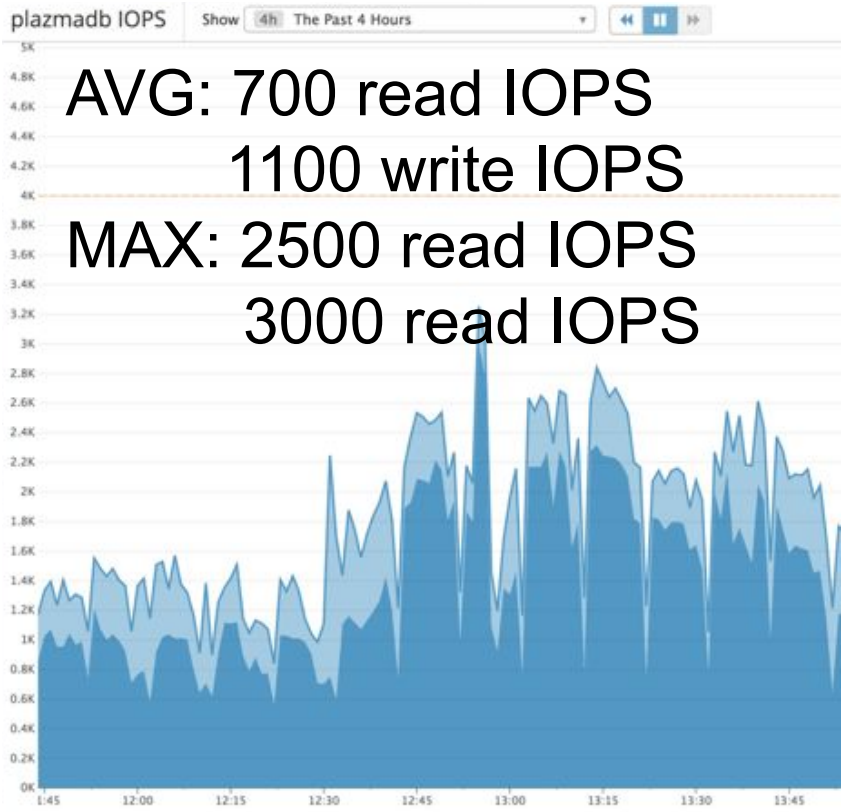


Data Volume

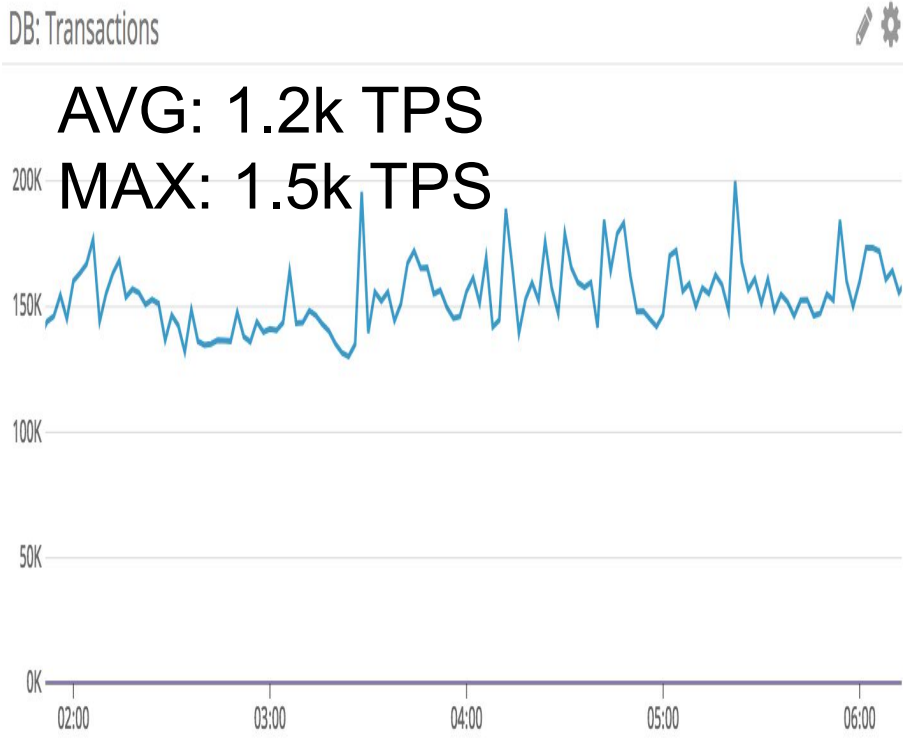


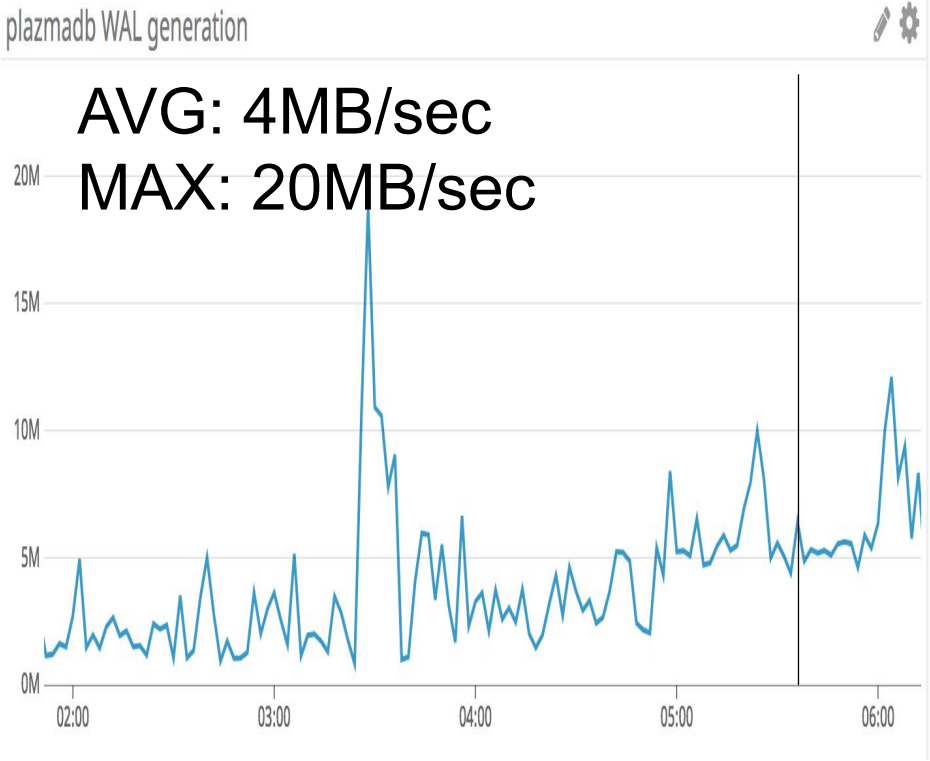
CPU & IO utilization of PostgreSQL (Meta DB)



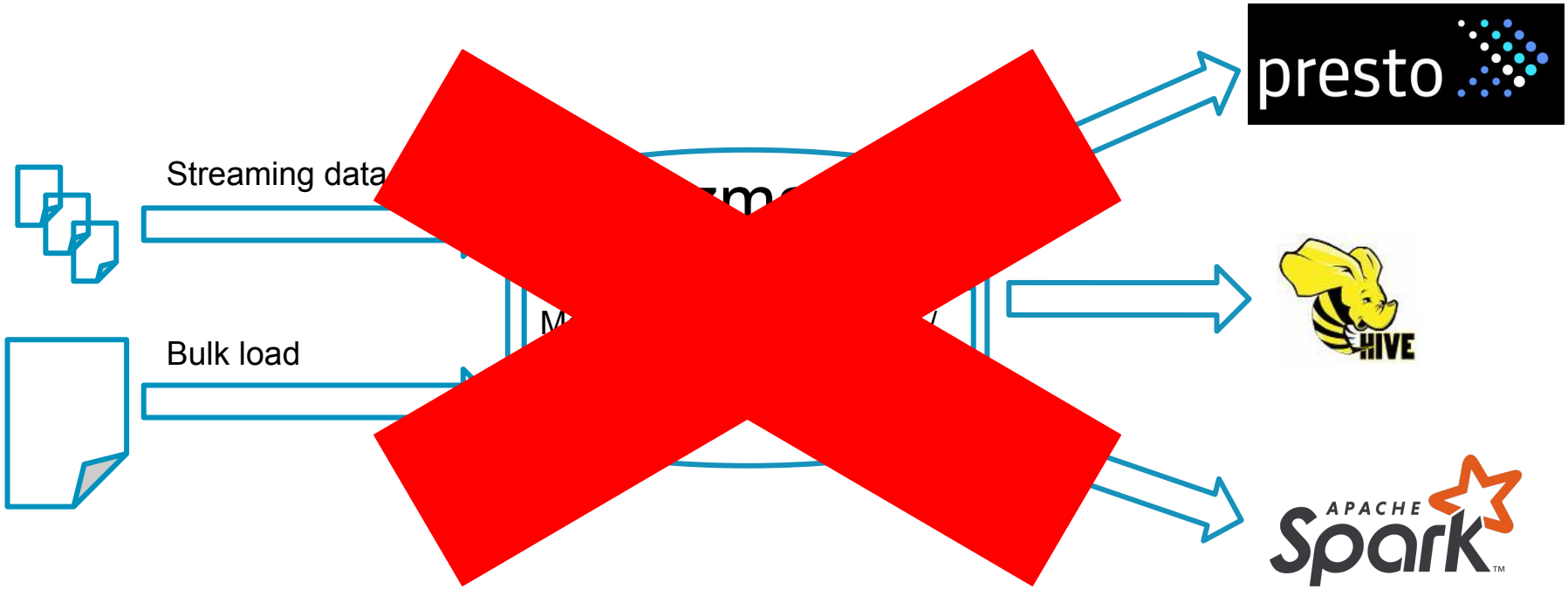


TPS & WAL Throughput





If PlazmaDB was down..



If PlazmaDB was down..

Planning Major Version Upgrade

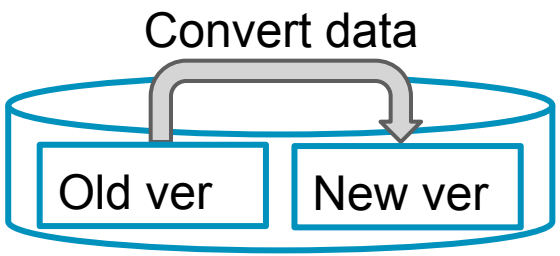
Long Way to Complete Upgrade

- Choose upgrade strategy
- Choose the next major version
- Plan operation and evaluation (include Rollback ops)
- Regression Test on the new major version

Major Version Upgrade Strategies

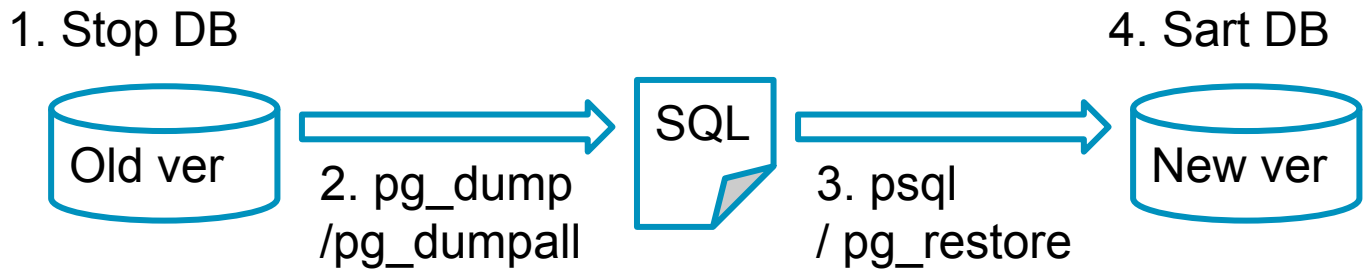
- pg_dump/pg_dumpall

- pg_upgrade



- Logical replication

Upgrade with pg_dump/pg_dumpall

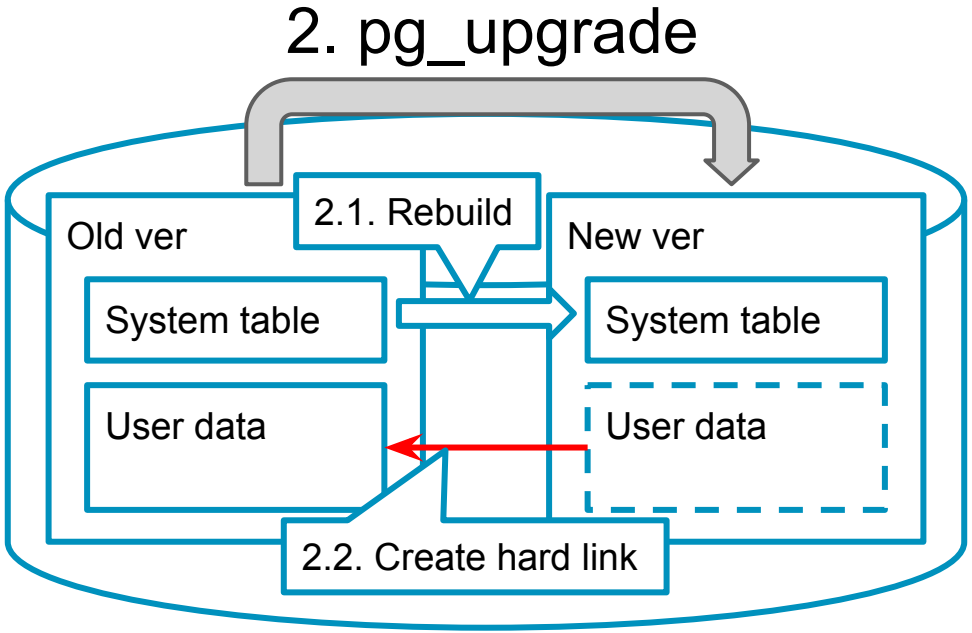


Downtime: Step 1 to 4

Include Full data copy

Upgrade with pg_upgrade Link mode

1. Stop DB

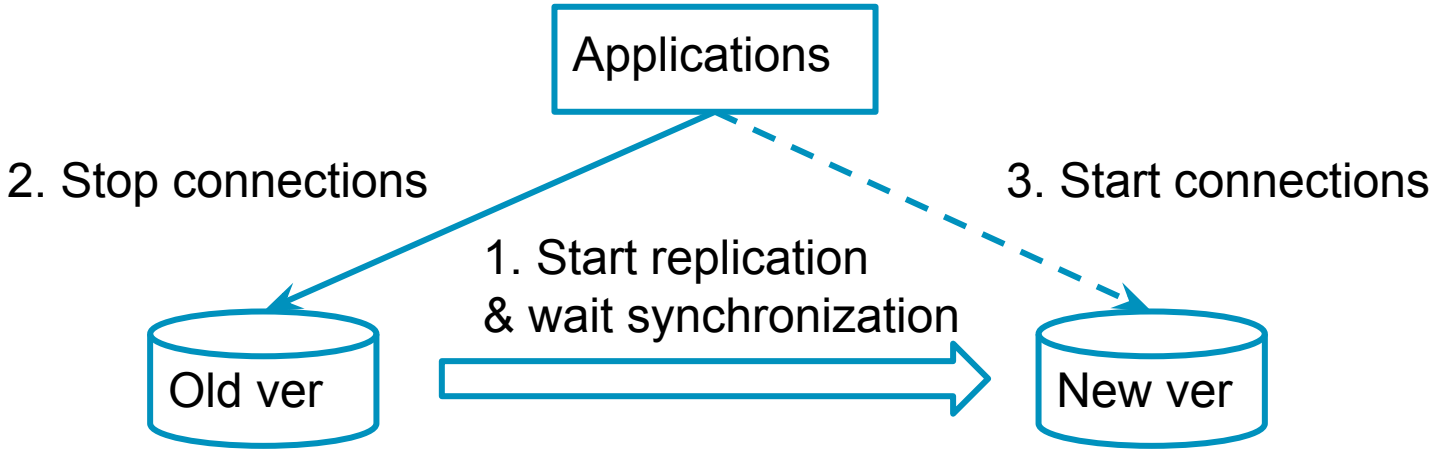


3. Start DB

Downtime: Step 1 to 3

Not include user data copy

Upgrade with Logical Replication

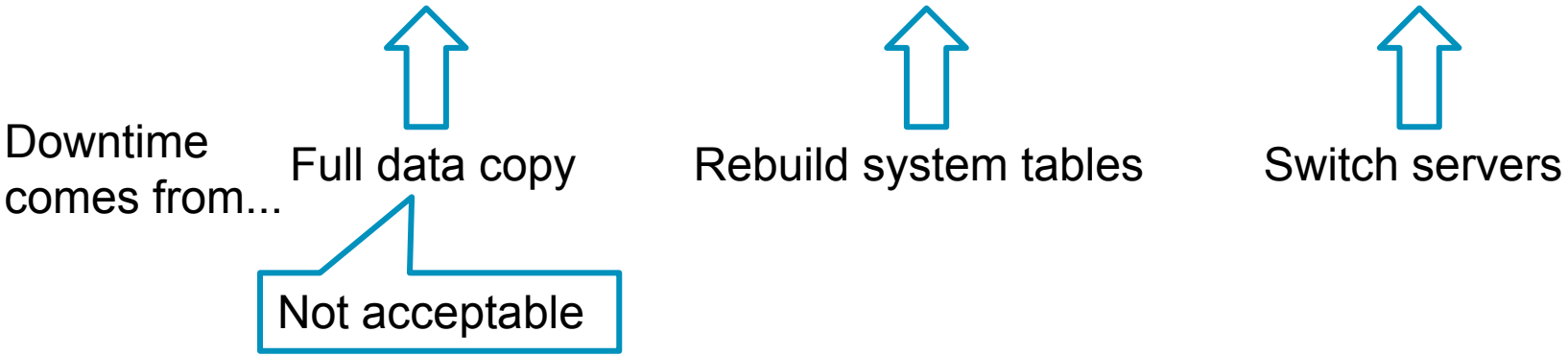


Downtime: Step 2 to 3

DBs don’t stop during upgrade

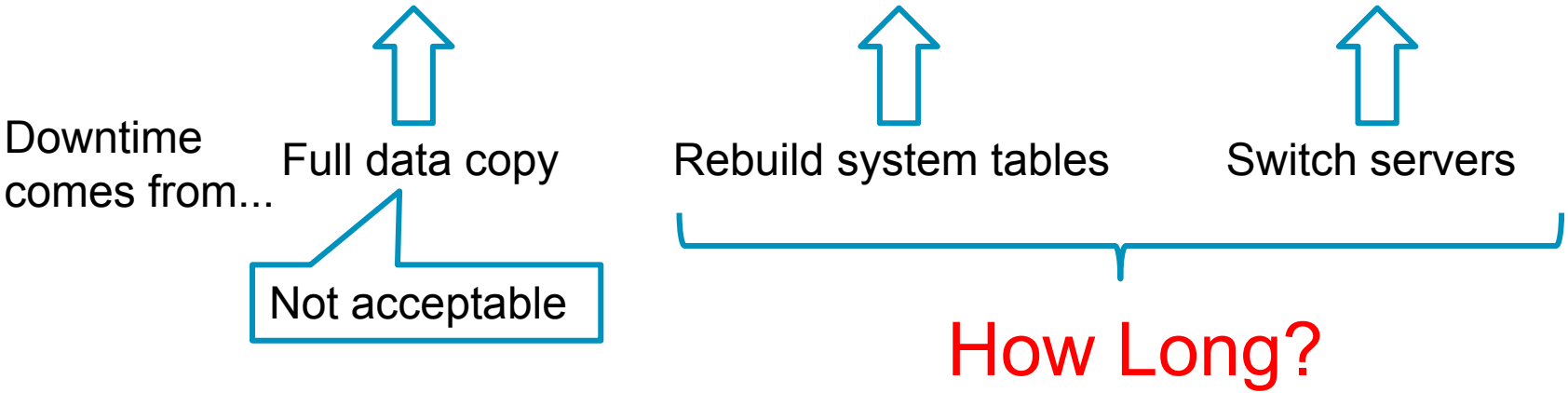
Downtime

pg_dump/pg_dumpall >> pg_upgrade > Logical Replication



Downtime

pg_dump/pg_dumpall >> pg_upgrade > Logical Replication

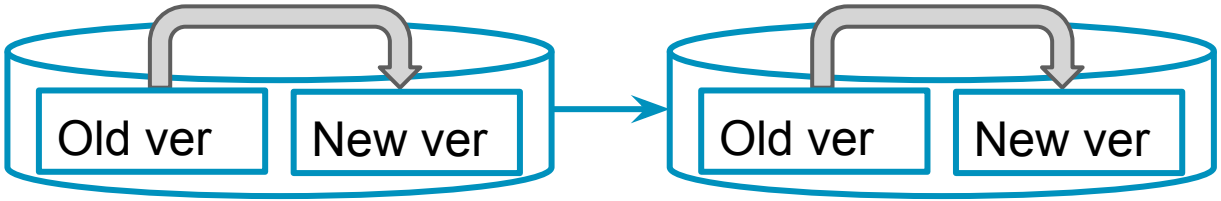


Downtime: pg_upgrade in RDS

- Operation
  - Upgrade major version by modify-db-instance API
    Perform pg-upgrade link mode internally
- Downtime -> 7-8 mins
- Note
  - Downtime happens several minutes after invoking API
    Exact time to happen cannot be specified.

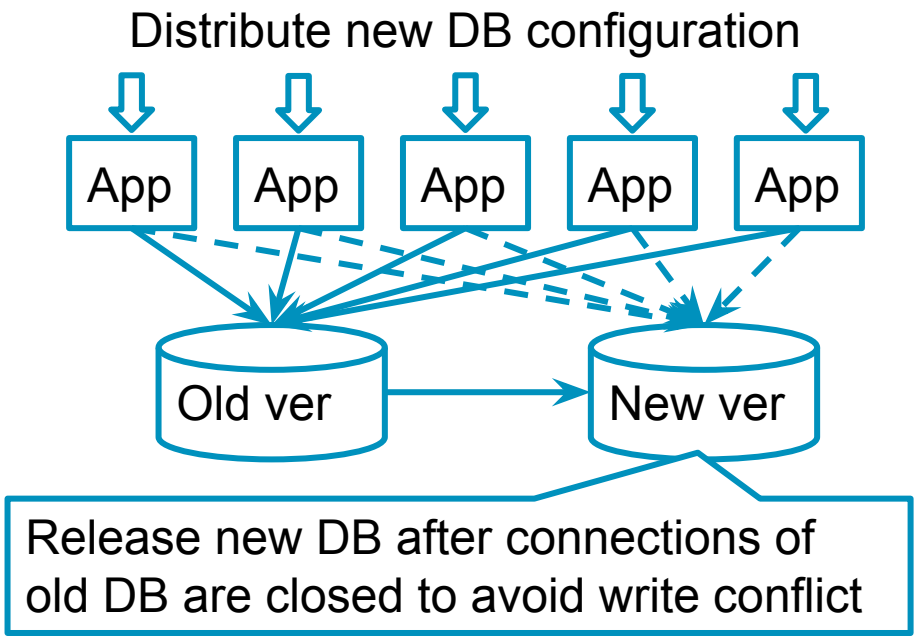
Upgrade of Replicas

- Downtime will be longer if upgrade of replicas is required
  - 1. Stop DBs
  - 2. Upgrade master
  - 3. Upgrade slave
  - 4. Start DBs

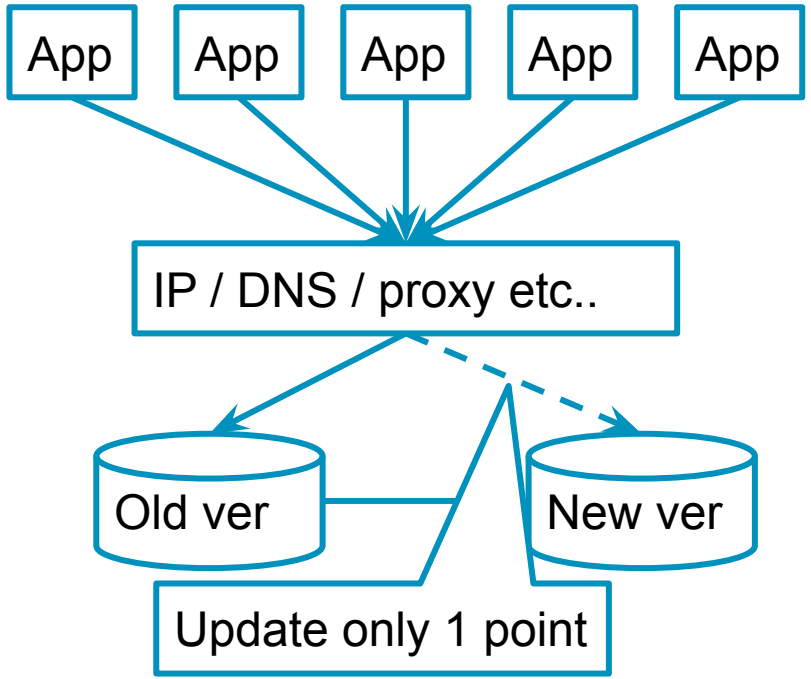


Logical Replication: How to switch servers?

Update application configuration

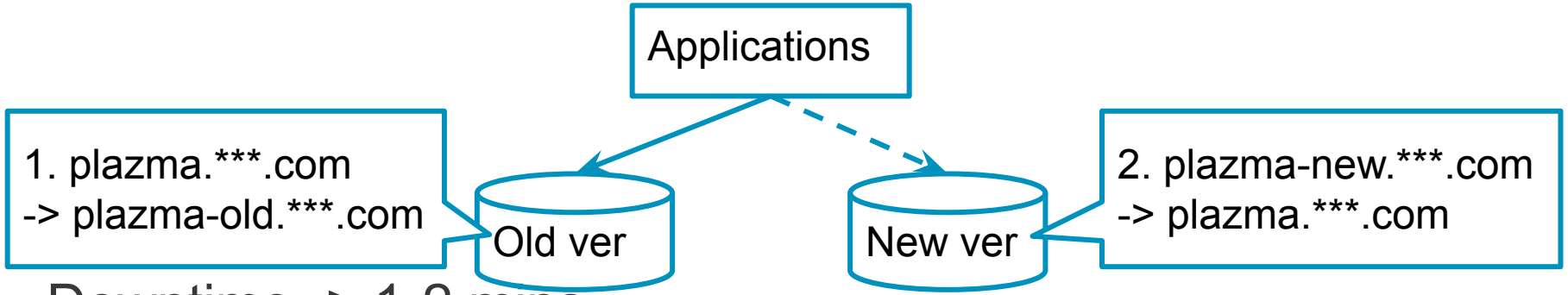


Overwrite DB endpoint



Downtime: Logical Replication

- Operation
  - Change DB endpoints (DNS) by modify-db-instance API



- Downtime -> 1-2 mins

Feasibility of each Strategy

pg_upgrade

- Pros
  - Easier operation
    Especially in AWS RDS
- Cons
  - Longer downtime
  - Upgrade only 1 major version at once (AWS RDS)

Logical Replication

- Pros
  - Shorter downtime
- Cons
  - PostgreSQL 9.3 doesn't have native logical replication

Logical Replication on PostgreSQL

- Before 9.3: 3rd party tool based on trigger
  - Bucardo, Londiste, Slony
  - Write amplification happens to record delta
  - For AWS RDS, replication server is required outside of DB servers
- 9.4 - 9.6: 3rd party tool based on logical decoding
  - pglogical, AWS Database Migration Service (DMS)
  - DMS provides managed replication server
- After 10: Native logical replication
  - No external process is required

Our Plan

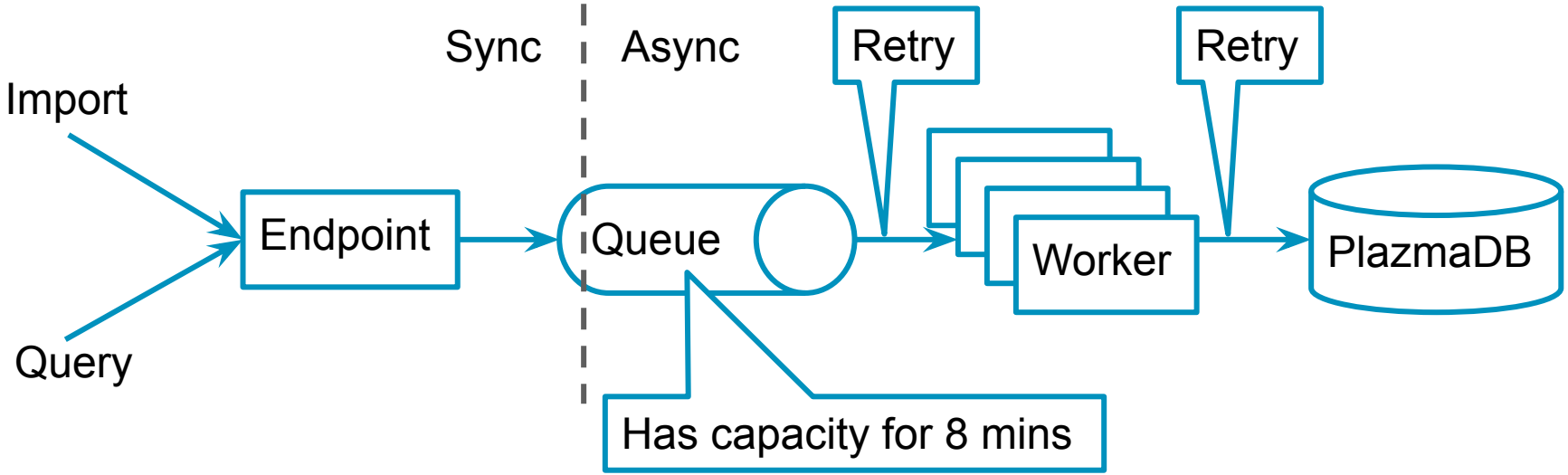
- Step 1: 9.3 -> 9.4 by pg_upgrade
  - Operation simplicity and manageable downtime
  - Done on May 2018

- Step 2: 9.4 -> latest by DMS
  - Less downtime and managed service is available
  - Coming soon..

Things to Consider on Upgrade

How to handle downtime?

8 minutes downtime is expected



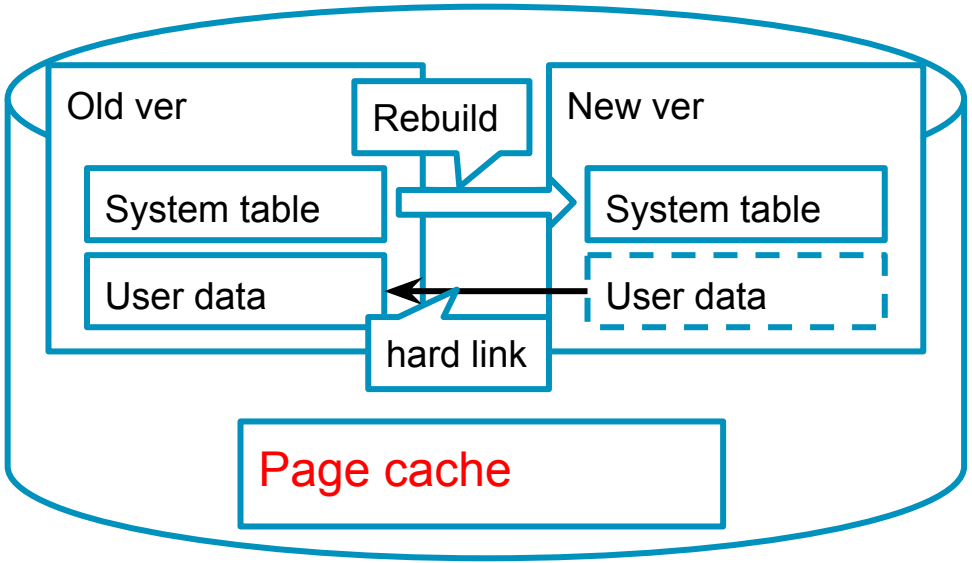
Requests were delayed but no error was returned

Impact of Cold Start

- Shared buffers is empty after booting DB
  - In our env, Shared buffer = 160GB
  - PostgreSQL page size = 8kB
  - Pages to load into buffer = 160GB / 8kB = 20M pages
- IOs are issued based on some factors
  - Number of queries
  - Number of pages read by a query
  - Locality

Bonus of pg_upgrade: Page Cache is Hot

- Page cache of OS remains during upgrade
  - Binary format of user data file is compatible

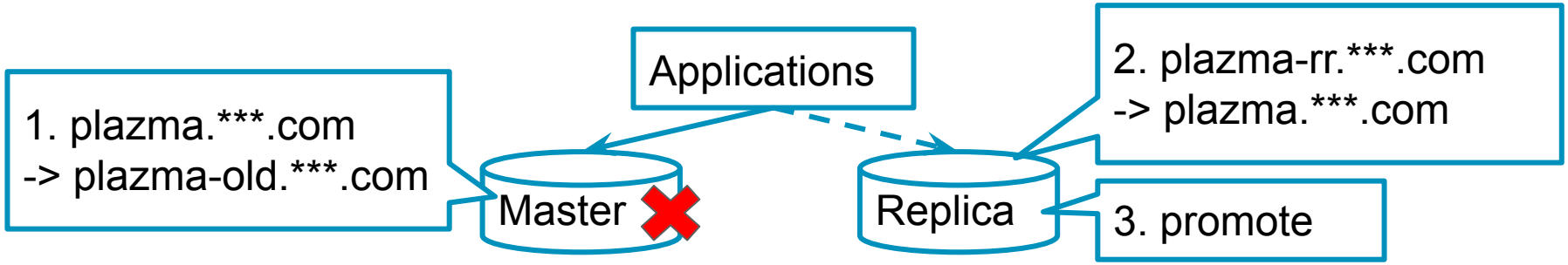


Mitigate Impact of Cold Start

- Adjust RDS Preserved IOPS to 10k
  - Our major select workload scans ~10k pages/sec
  - RAM = 244GB, Buffer = 160GB -> Page cache ~ 80GB
    - ✓ 50% is filled by page cache -> expected IOPS ~ 5k
    - ✓ +5k just in case
- Page cache isn't hot if you need to change servers
  - pg_prewarm

Rollback Plan

1. Promote Replica: 1-2 mins



2. Restore from Backup (if No.1 doesn't work): 30-60 mins

Evaluate New Major Version

Importance of Evaluation

- Rollback is the worst case
  - Longer downtime
  - Data created after upgrade should be recovered manually
- Detect degradation by test
  - Interface incompatibility
    - ✓ System test on staging environment
  - Performance degradation
    - ✓ production != staging in terms of server resources

Real World Application is Complicated..

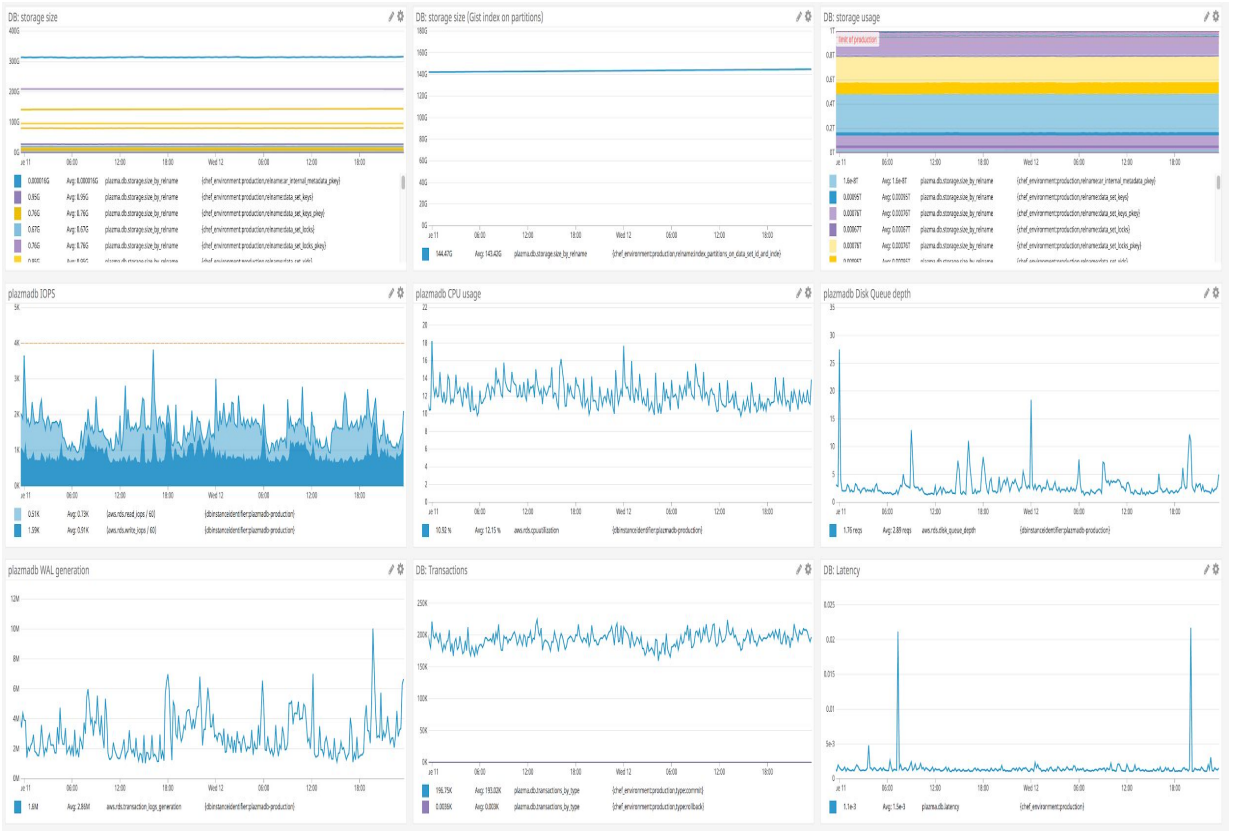
Many Performance Related Factors

- Type of Workload
- Users' behaviour
  - # of import requests / analytical query requests
  - Data skew
- Metadata Storage size
- Server Size (CPU cores, RAM, Storage, Network)
- etc..

Modeling workload and Define target performance

Use Metrics for Modeling Workload

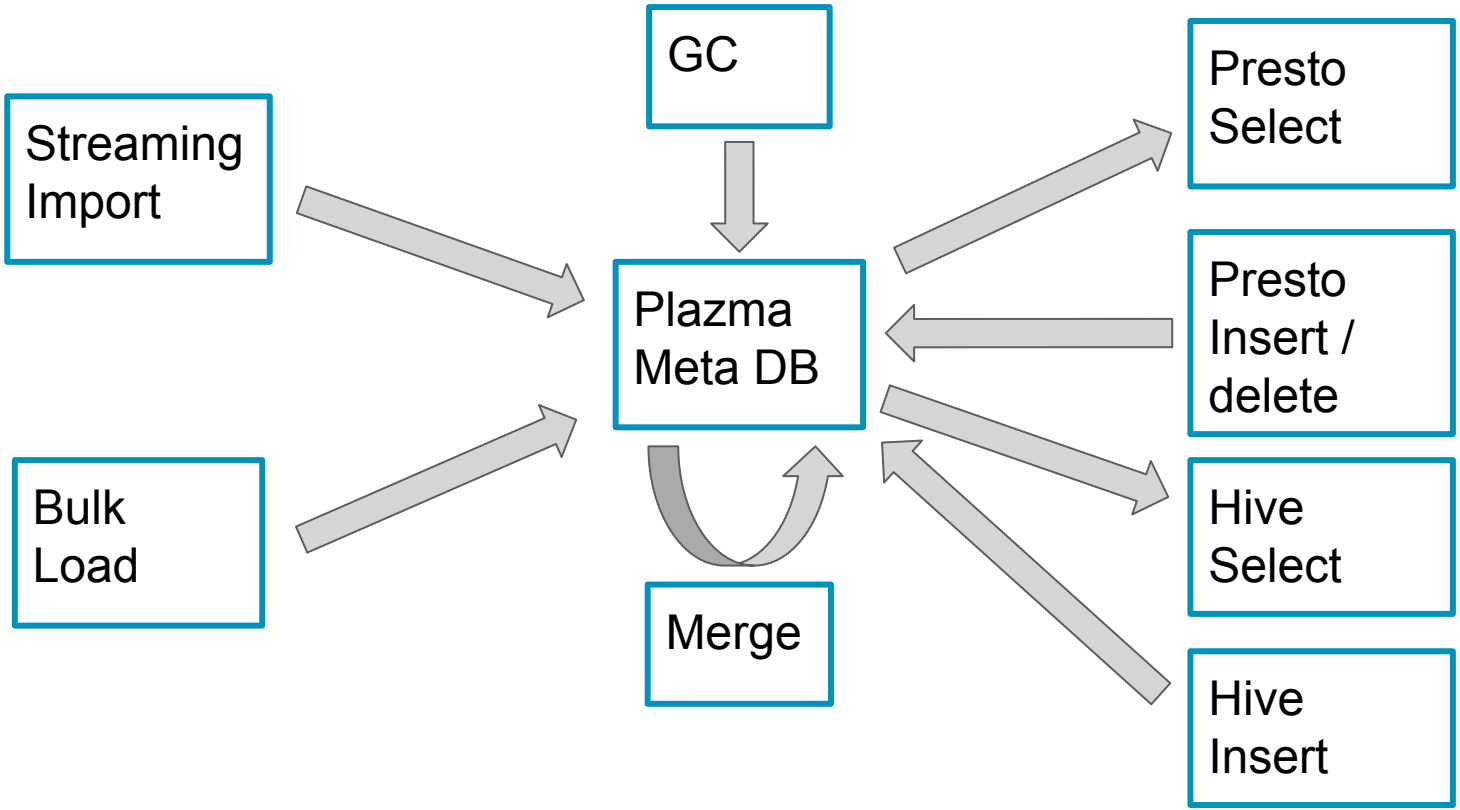
- Metrics Collectors
  - Arm Treasure Data: detailed analyze
  - DataDog: visualization
- Metrics
  - CPU / IOPS
  - Table size
  - # of Query / Query Types
  - Cache hit ratio
  - SEL / INS / DEL / UPD rows
  - … 50+ metrics



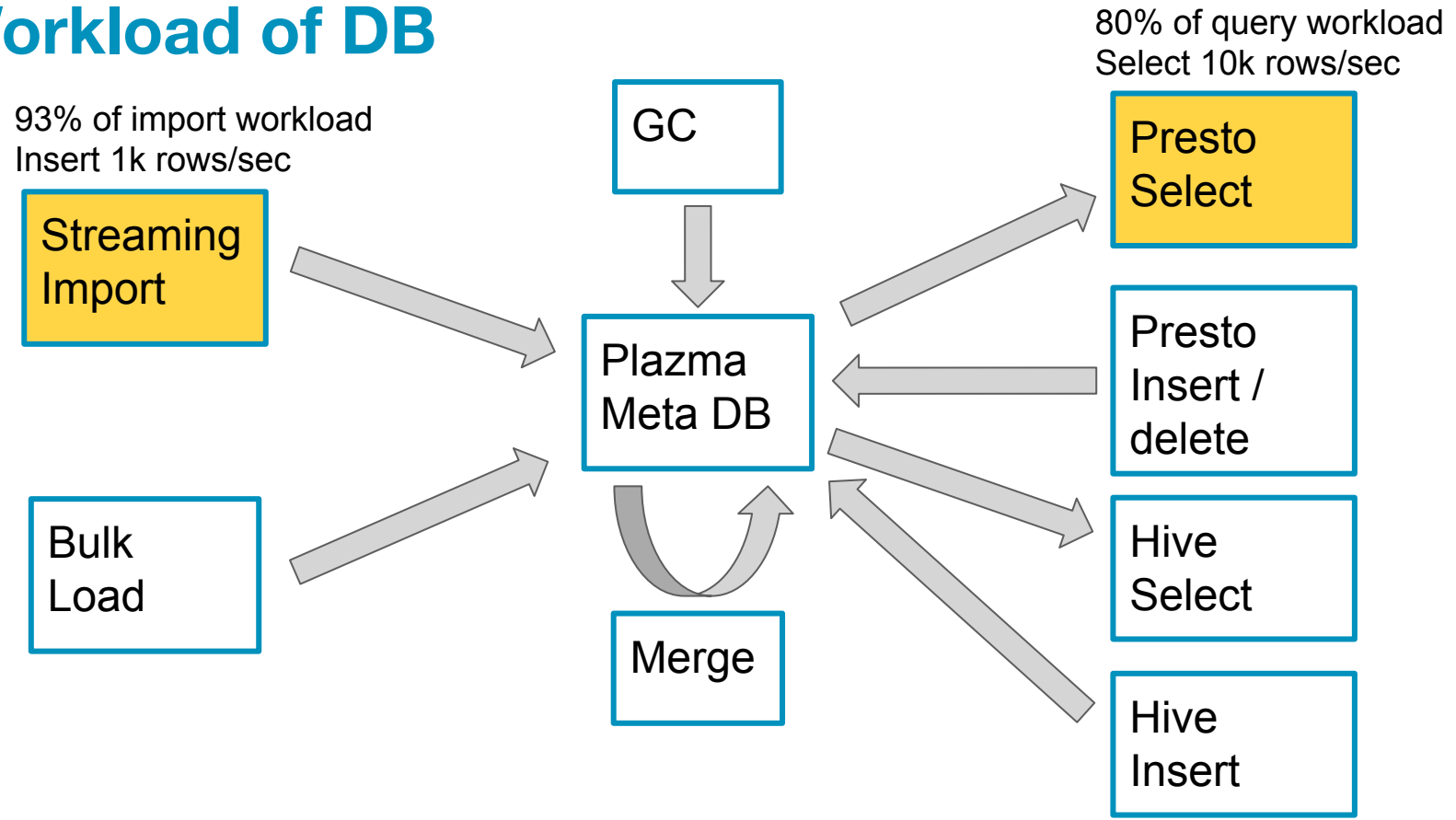
Created Benchmark Tool based on Metrics

- Enable production size perf measurement without integration
  - Iteratively refined benchmark workload by adding lacked parameters
    - Number of queries / Types of queries
    - DB size
    - Data access locality
    - Data skew
- Goal of Measurement
  - Check if pg 9.4 is capable of processing current production workload (+α)
  - Detect different behavior of metrics

Workload of DB

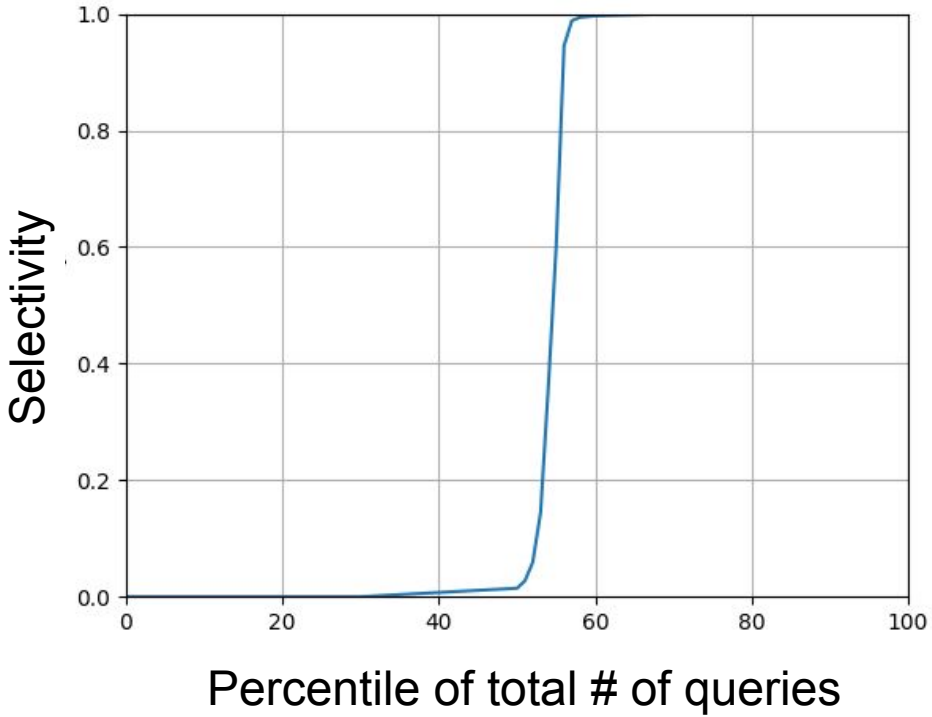


Workload of DB



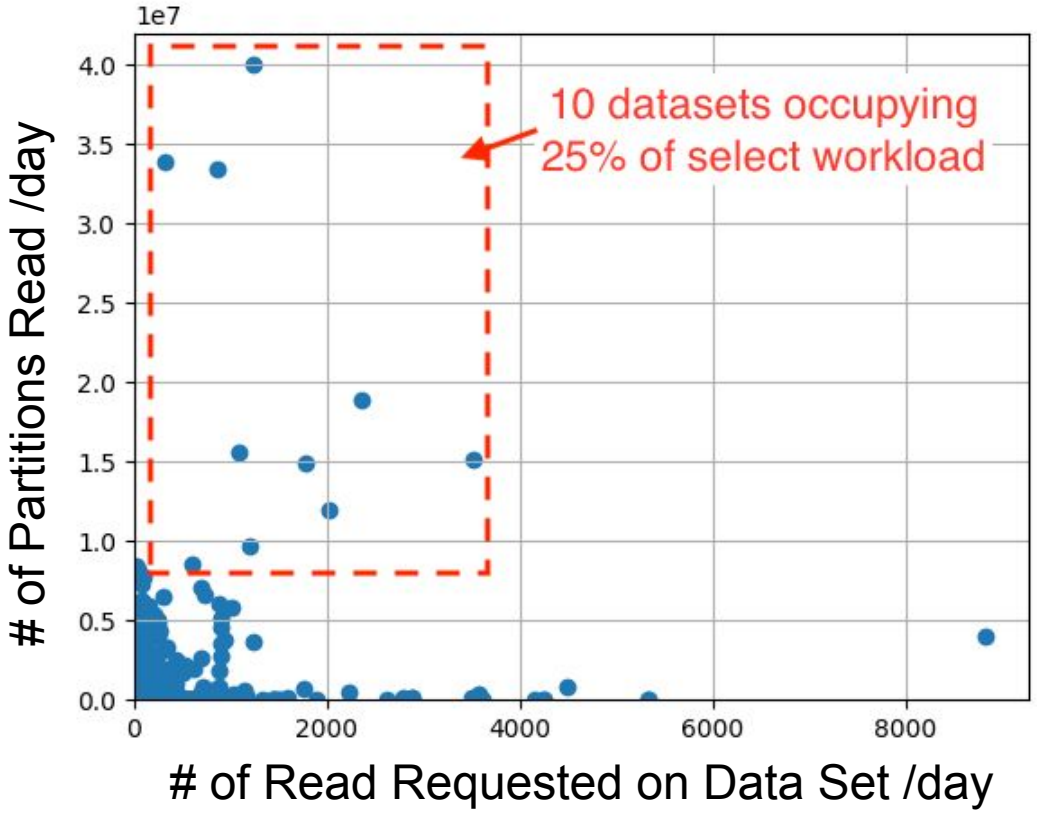
Selectivity

- Random sampling from actual selectivity distribution
  e.g.)
  - 40% queries: sl = 1
  - 5% queries: sl = [0.01, 0.5]
  - 5% queries: sl = [0.5, 0.99]

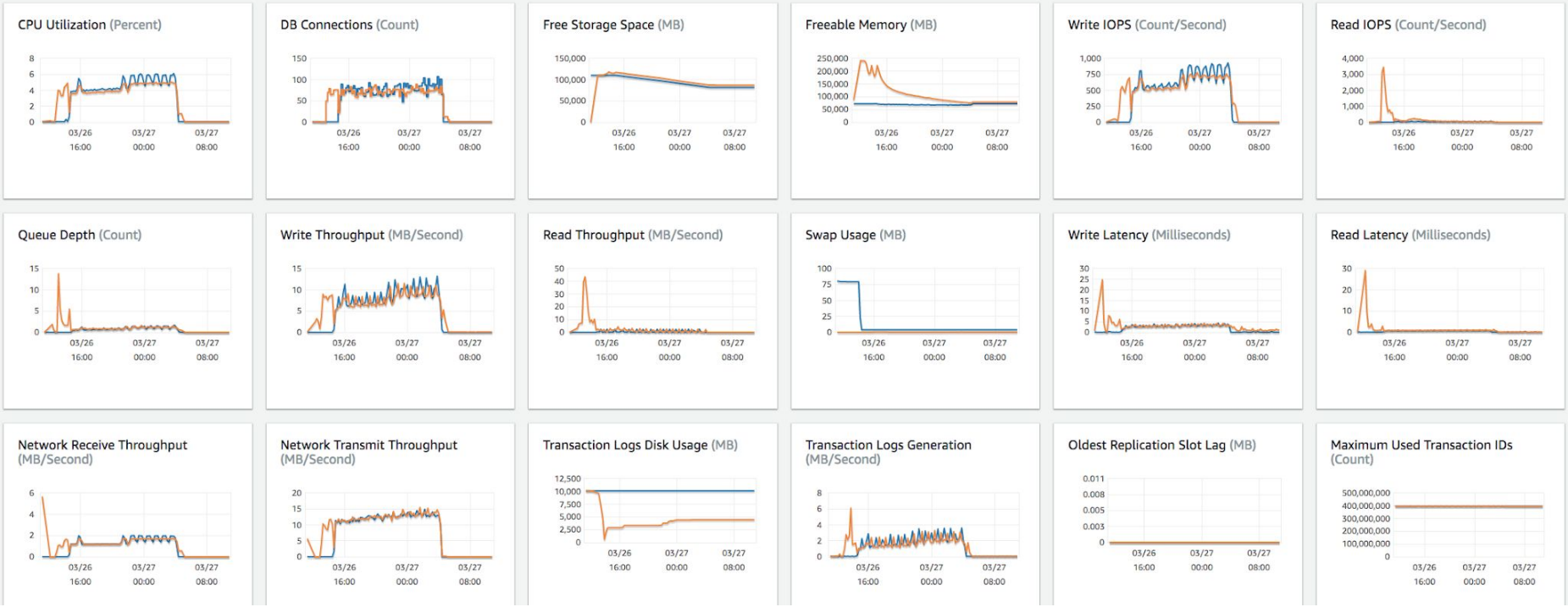


Data Access Locality

- Metadata size = 1TB
- Shared Buffer size = 160GB
- But, Hot Data size is smaller than Shared Buffer
  e.g.)
  - 85% of workload comes from 1% data sets
  - 95% of workload comes from 5% data sets



Result of Performance Measurement



Observed no performance degradation

Operation in Production

Final Upgrade Plan

- Preparation
  - Increase Preserved IOPS to 10k for cold start
  - Scale up replica server to the same as master for rollback
- Operation include downtime
  - Invoke pg_upgrade via modify-db-instance API in scheduled maintenance window
    - ✓ ~8 mins downtime starts several minutes after invoking API
    - ✓ Expected Read IOPS after upgrade ~5k

Actual Impact

Requests were delayed 9 minutes (~ DB downtime)

[US Region] Scheduled Metadata Database maintenance

Scheduled Maintenance Report for Treasure Data

Completed

Maintenance and system *recovery* processes have been fully executed.
Streaming import and job execution delay was around 9 minutes from 6:19 pm PDT.

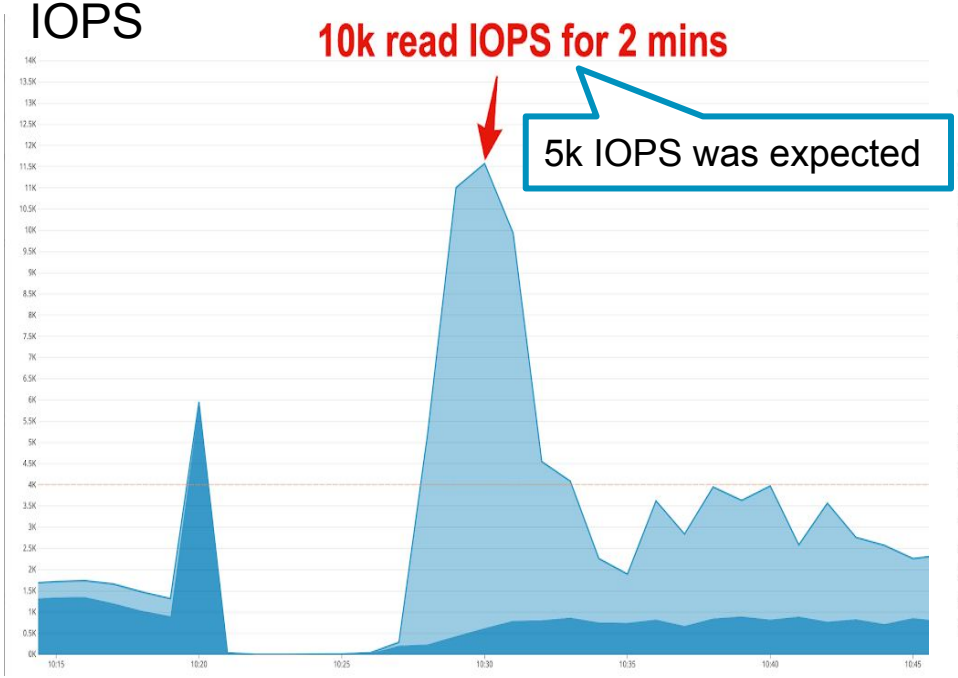
The scheduled Metadata Database maintenance is now complete.

Posted 7 months ago. May 08, 2018 - 18:45 PDT

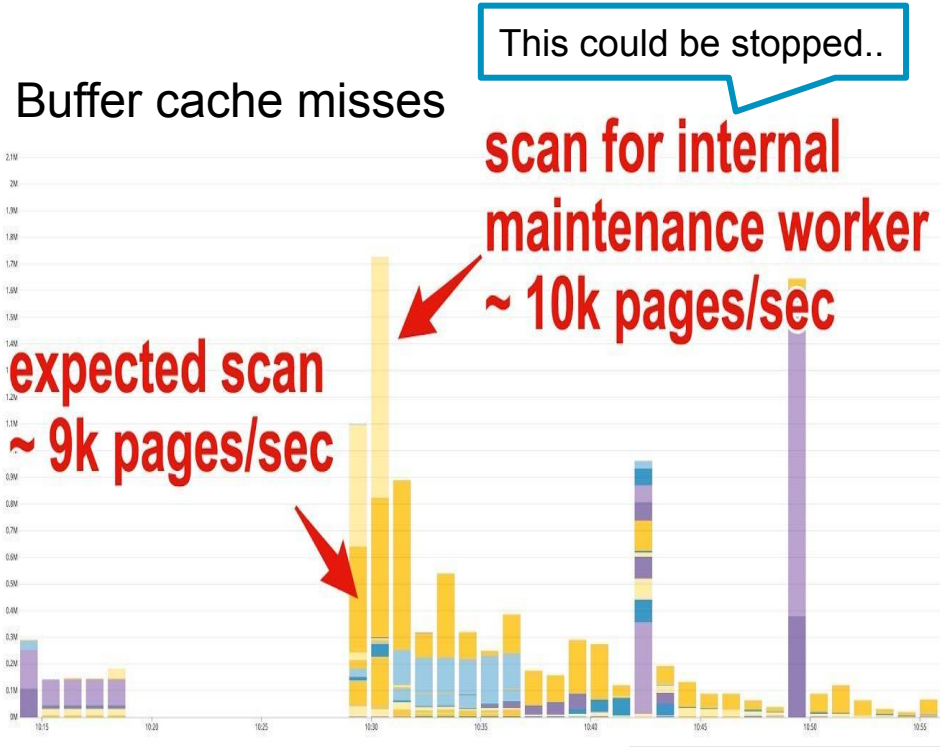
Request Queue Depth

Actual Impact of Cold Start

IOPS



Buffer cache misses



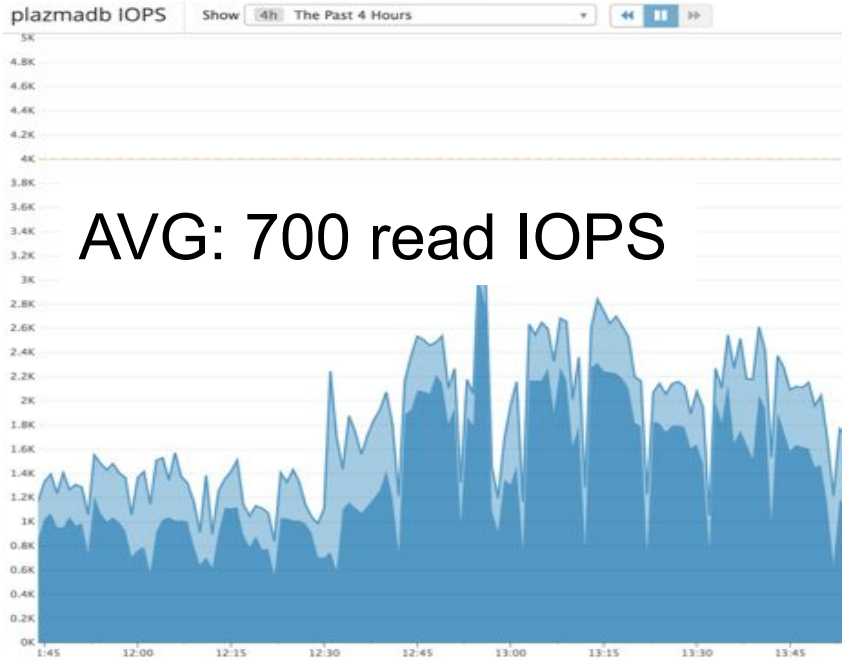
9k / 19k ~ 47% was hit on page cache

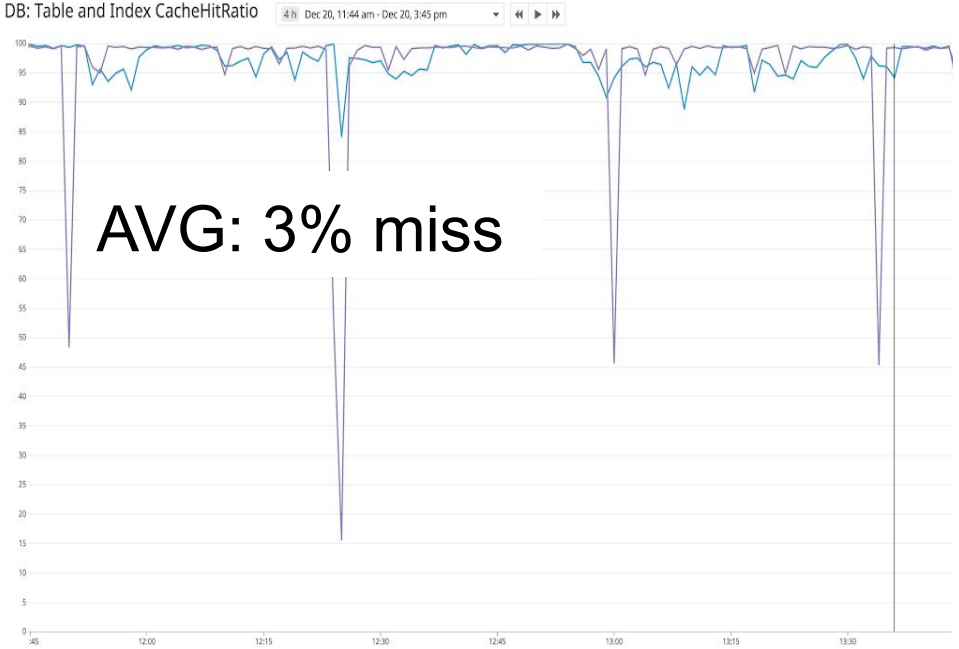
Summary

- Choose appropriate strategy for your environment
  - Downtime: pg_dump >> pg_upgrade > logical replication
  - Operation easiness: pg_dump < pg_upgrade < logical replication
  - Consider not only DB impact but also entire system impact
  - Think of Downtime, Cold Start, Rollback
- Metrics is very important
  - Estimate impact, Model workload, Evaluate performance, Measure system healthiness

Thank You!
Danke!
Merci!
谢谢!
Gracias!
Kiitos!

Impact of Cold Start

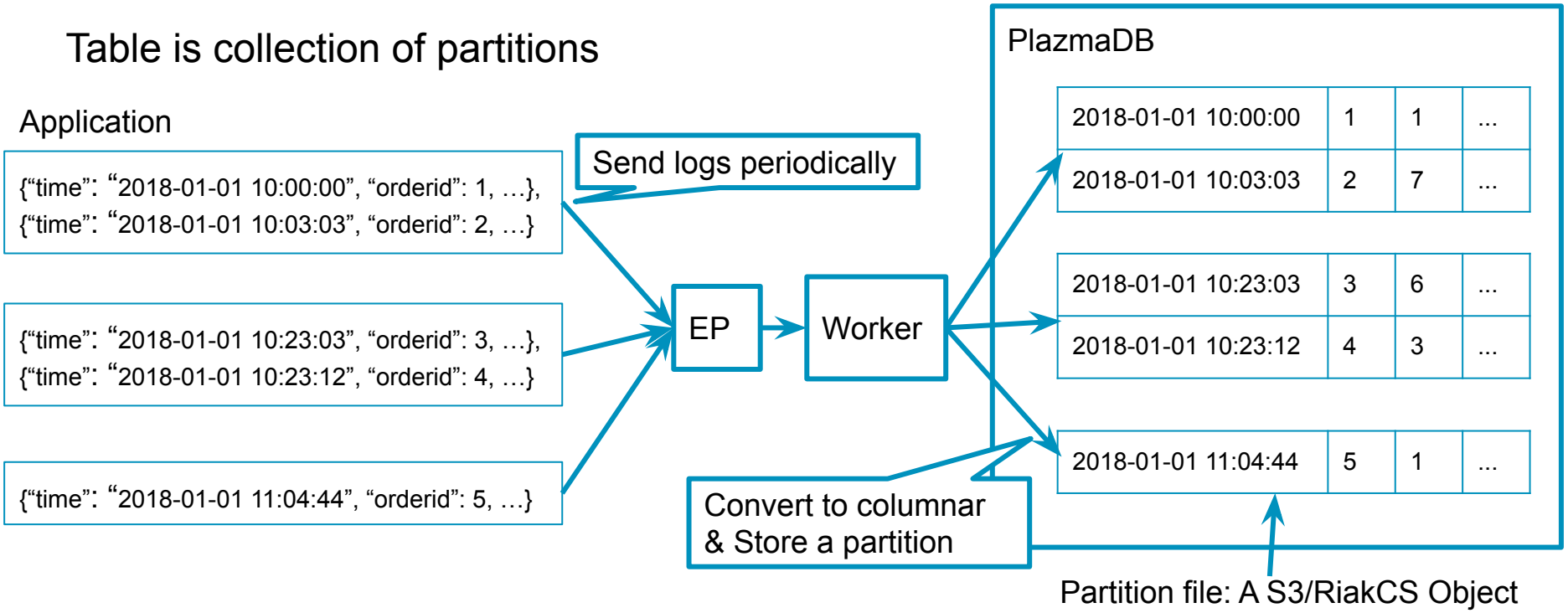




230 IOPS for 1% miss -> 23k IOPS for 100% miss

PlazmaDB Streaming Import

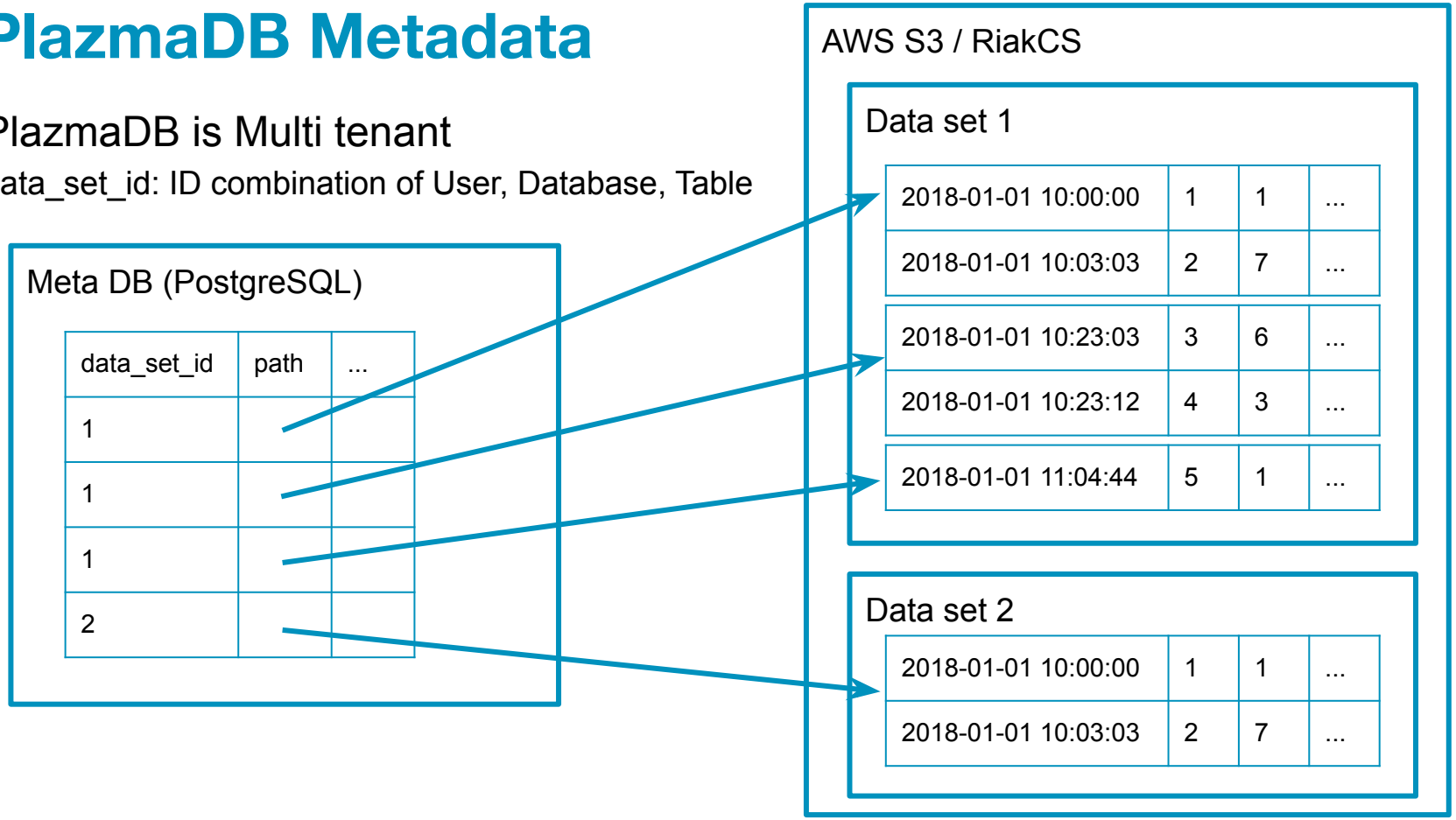
Table is collection of partitions



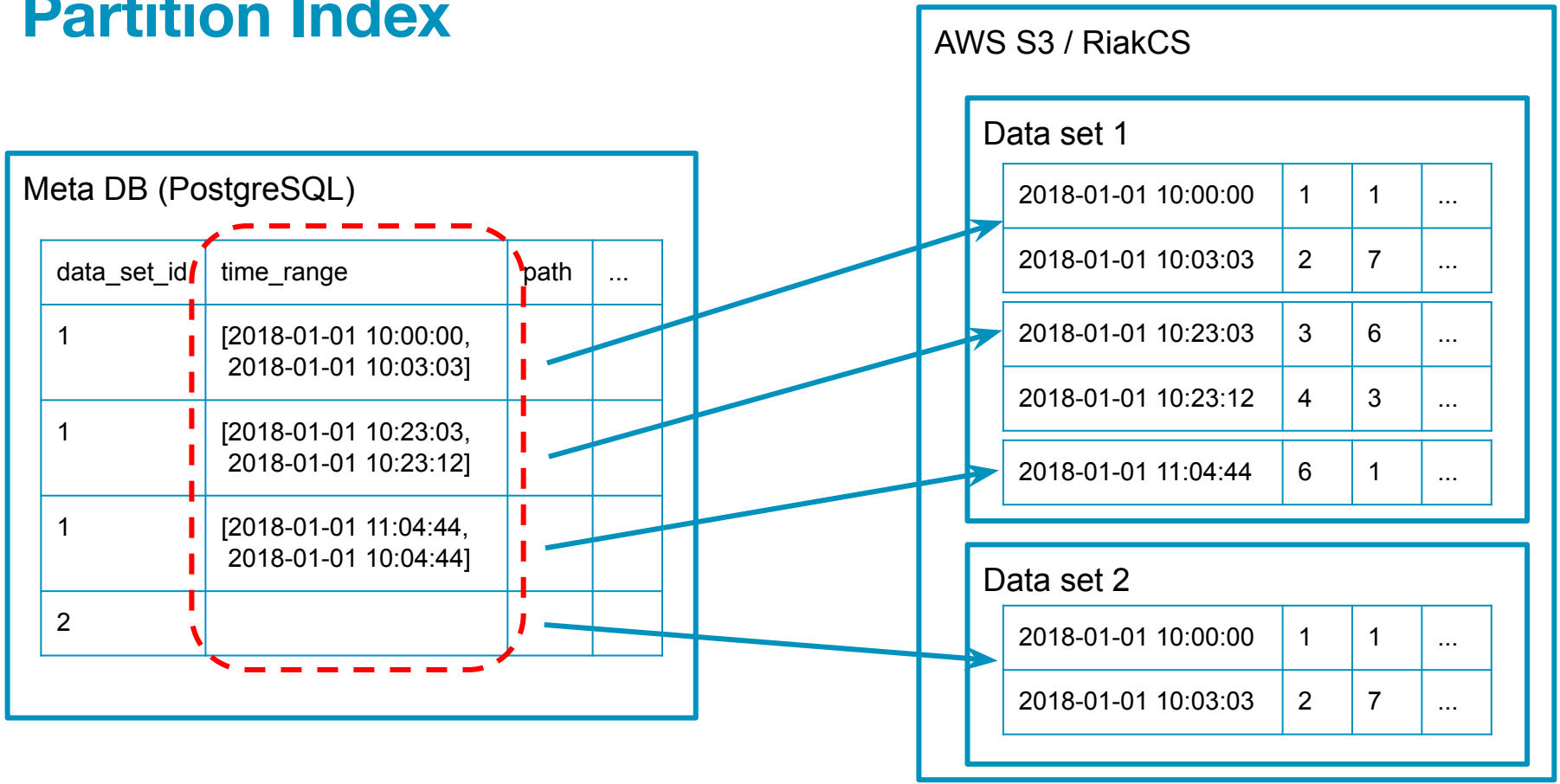
PlazmaDB Metadata

PlazmaDB is Multi tenant

data_set_id: ID combination of User, Database, Table



Partition Index



Partition Lookup on Analytical Query Processing

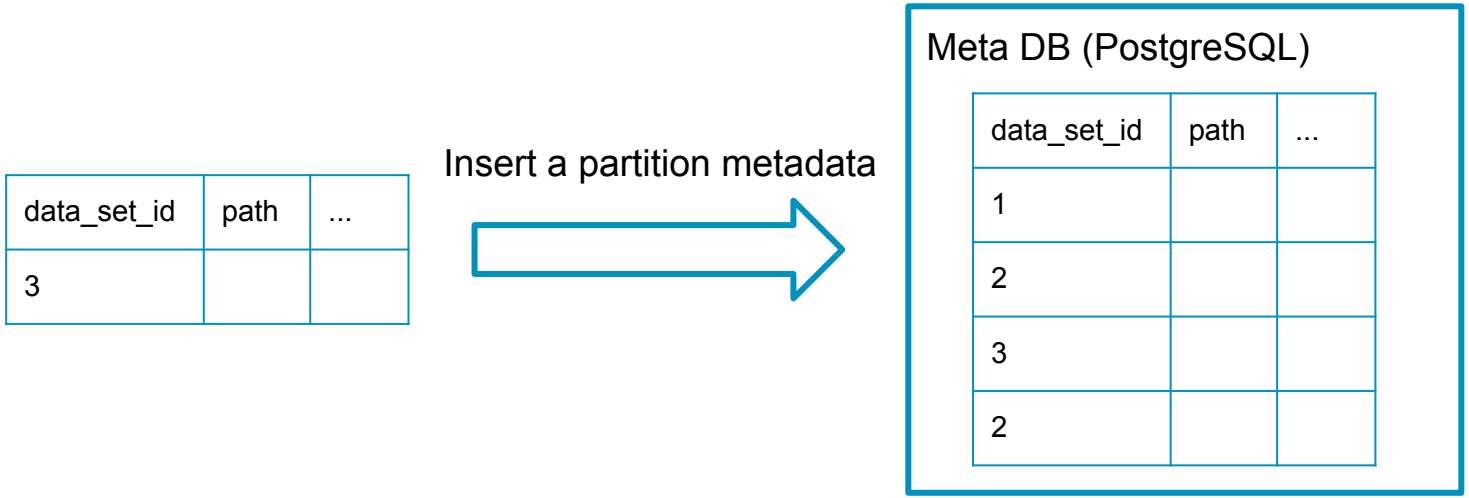
Meta DB (PostgreSQL)

data_set_id	time_range	path	...
1	[2018-01-01 10:00:00, 2018-01-01 10:03:03]		
1	[2018-01-01 10:23:03, 2018-01-01 10:23:12]		
1	[2018-01-01 11:04:44, 2018-01-01 10:04:44]		
2			

```
SELECT
  region,
  SUM(price)
FROM
  orders  -- assume this is data set 1
WHERE TD_TIME_RANGE(time,
'2018-01-01 10:00', '2018-01-01 11:00')
GROUP BY
  region
```

Benchmarking Streaming Import

Model of Streaming Import Workload



Performance related factors

Concurrency	(1, 2, 4, 8, 16, 32, 64, 128, 256)
Size of tuple	random based on normal distribution (185 byte on average)

Benchmarking Environment

- AWS RDS PostgreSQL

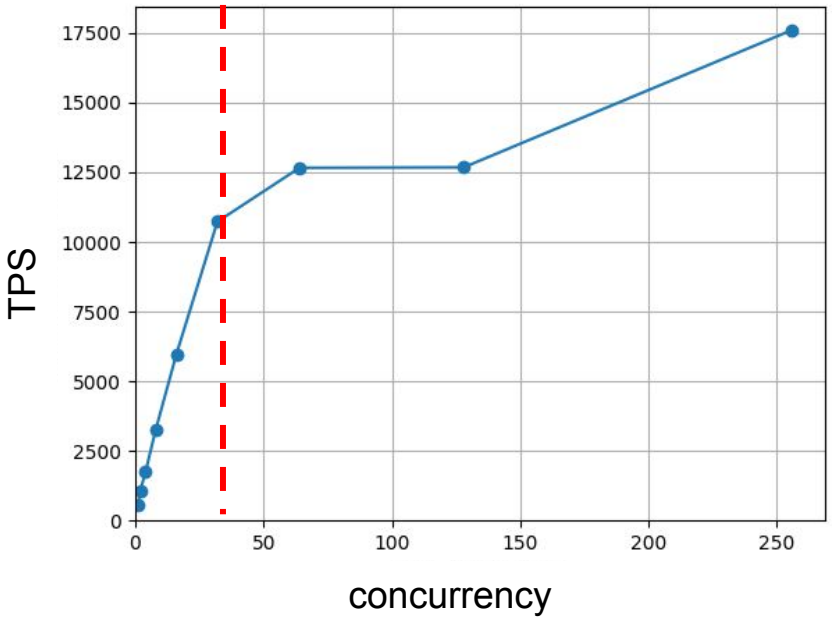
Instance type	db.r3.x8large (32 vcores, 244GB RAM)
Provisioned IOPS	4k
PostgreSQL version	9.4.17

- PostgreSQL parameters

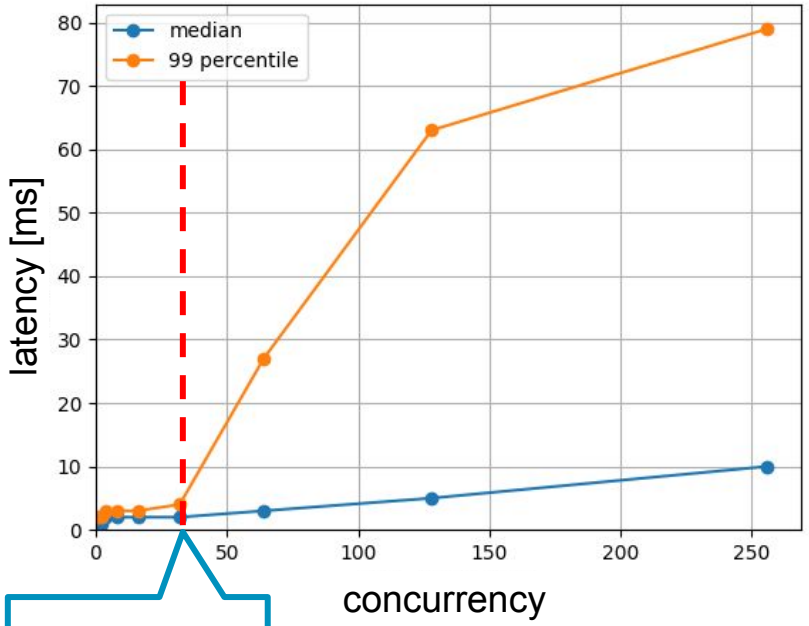
shared_buffers	160GB (~ 60% of RAM)
checkpoint_segments	1500 (24GB)

Scalability of Streaming Import Workload

Throughput



Latency



# of cores

Resource Consumption

CPU utilization

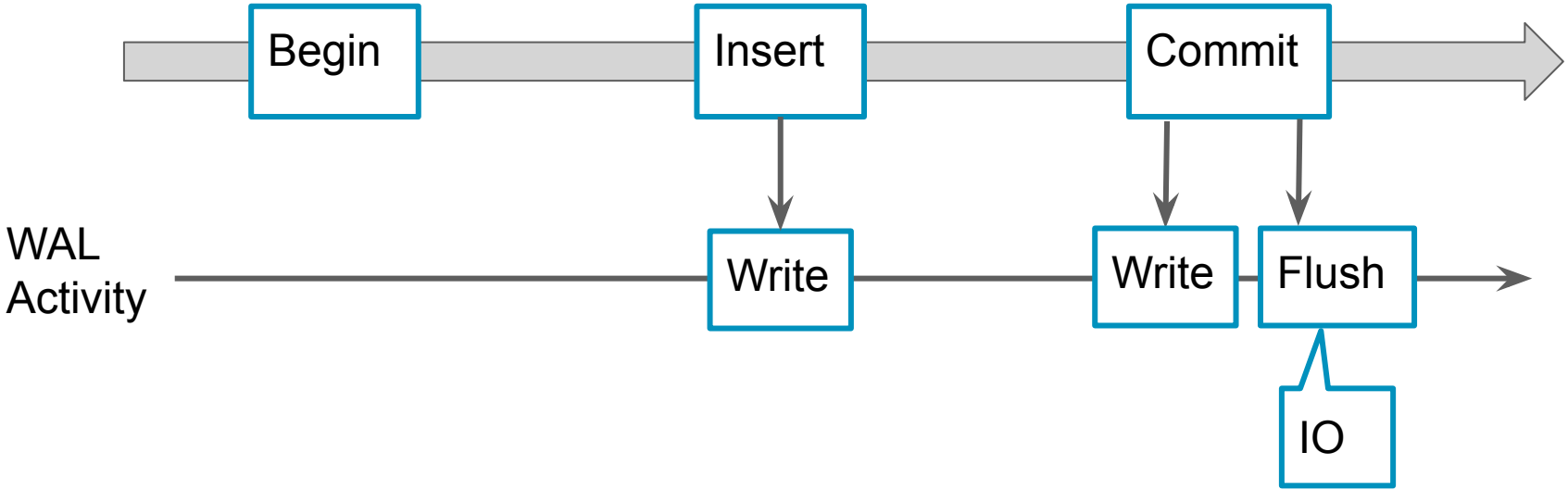
Write IO throughput

Write IO

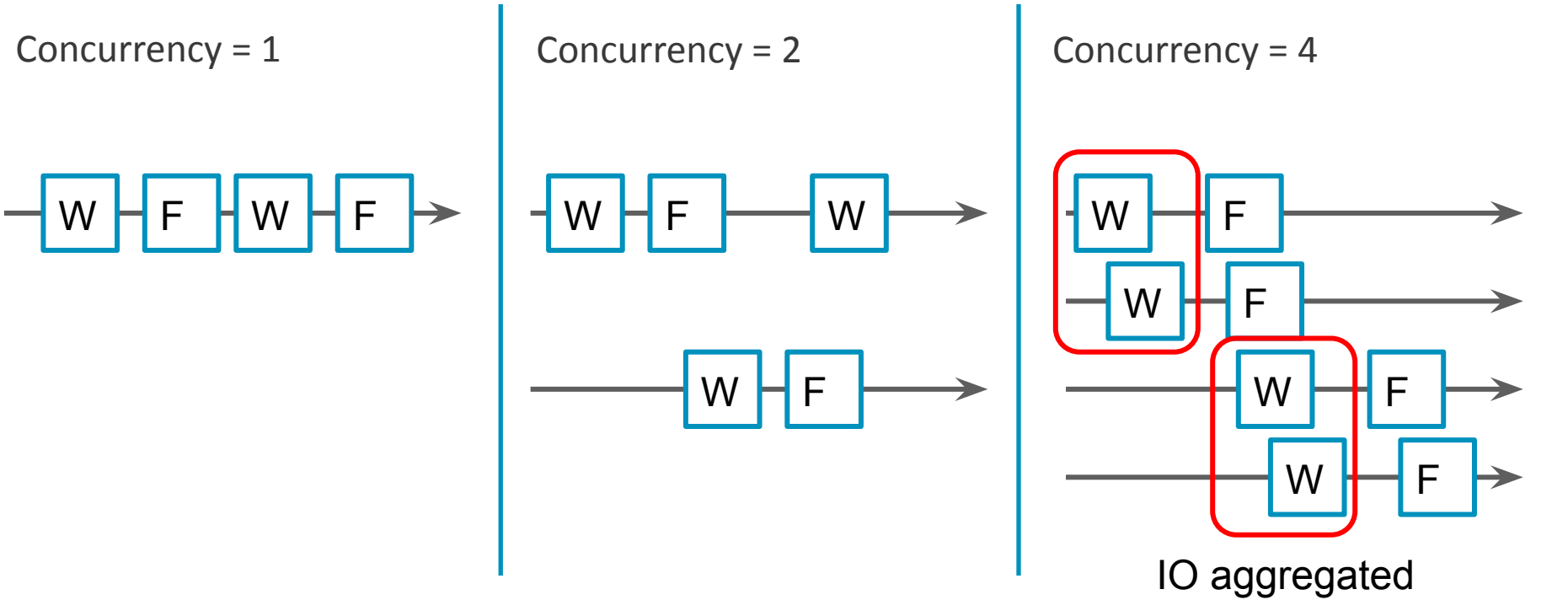
Write IO throughput

Write IOPS

When Write IO issued?



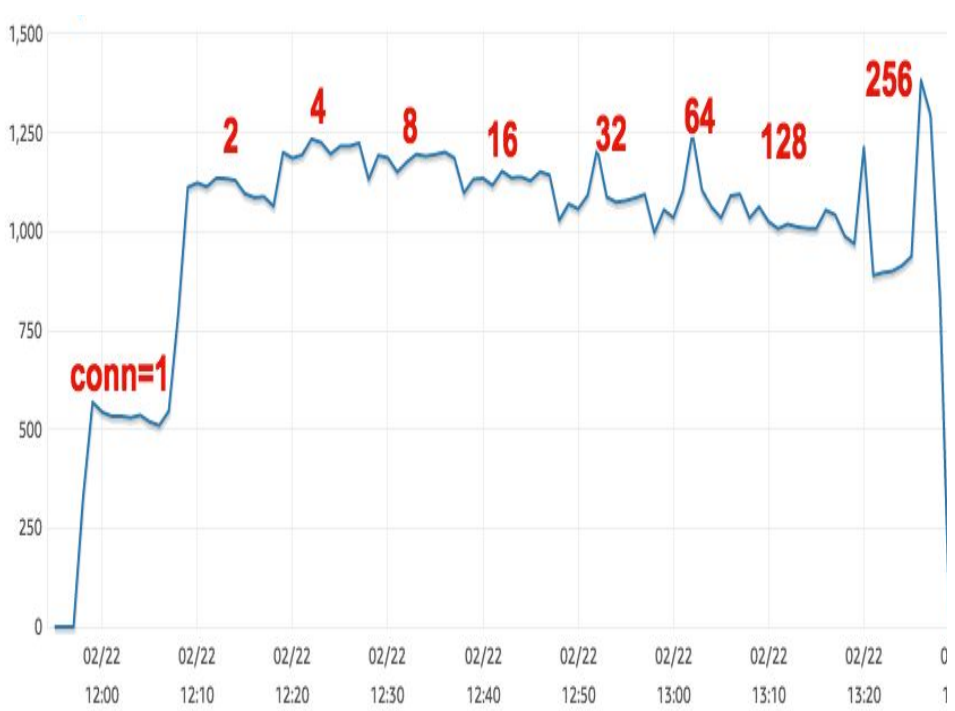
Concurrency and Write IO



Bottleneck

CPU utilization

Write IOPS



Summary of Streaming Import Workload

- CPU bottleneck
  - Scale almost linearly when concurrency is less than # of cores
  - Throughput can increase after that, but tail latency increases as well
  - Write IOPS doesn’t increase as increasing concurrency because of IO aggregation

Benchmarking Presto Select

arm

Model of Presto Select Workload

Meta DB (PostgreSQL)

data_set_id	time_range	path	...
1	[18-01-01 10:00, … 11:00]	a	
2	[18-01-01 10:00, … 11:00]	b	
1	[18-01-01 11:00, … 12:00]	c	
3	[18-01-01 13:00, … 14:00]	d	
2	[18-01-01 13:30, … 14:00]	e	
1	[18-01-01 16:00, … 17:00]	f	
...			
1	[18-01-02 03:00, … 04:00]	m	

data_set_id=1 and
time_range &&
[18-01-01 00:00, 18-01-02 00:00]

path
a
c
f

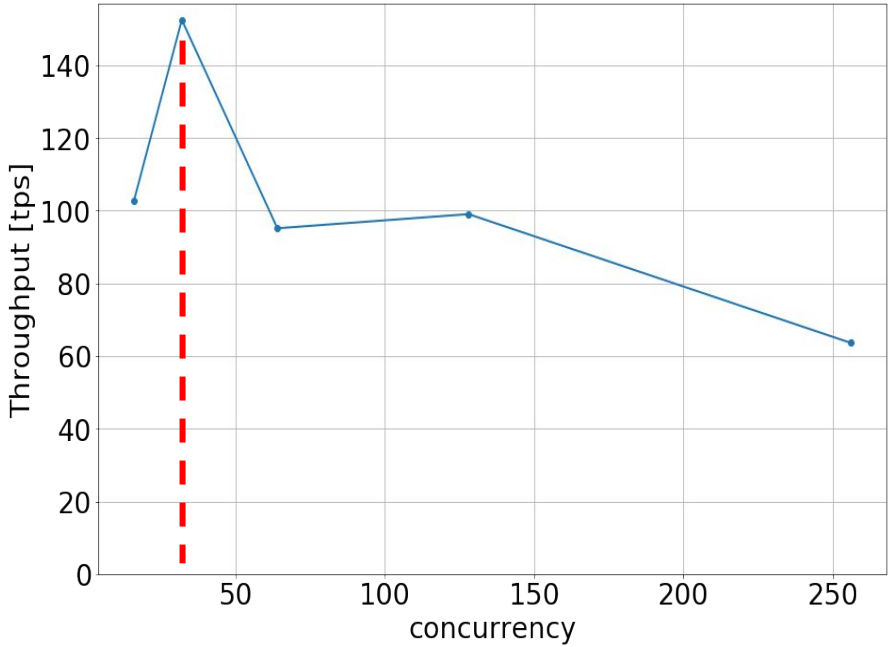
Index scan for data_set_id and time_range

Performance related factors

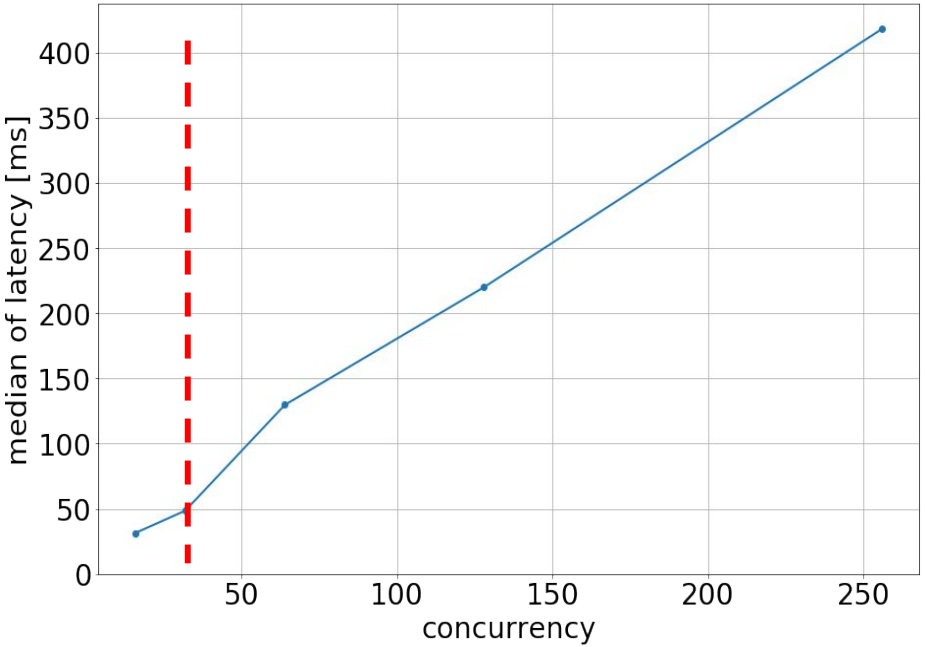
Concurrency	(16, 32, 64, 128, 256)
Metadata size	600GB (Dummy data based on actual trend)
# of data sets	30k
Time range to scan (selectivity)	(next slide)
Distribution of data set access frequency	(next slide)

Scalability of Presto Select Workload

Throughput



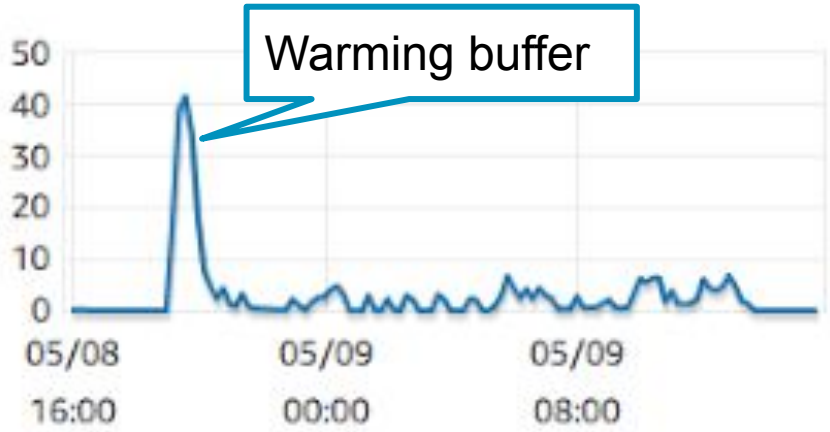
Latency



Resource Consumption (Concurrency=128)

CPU utilization [%]

Read IO throughput [MB/s]

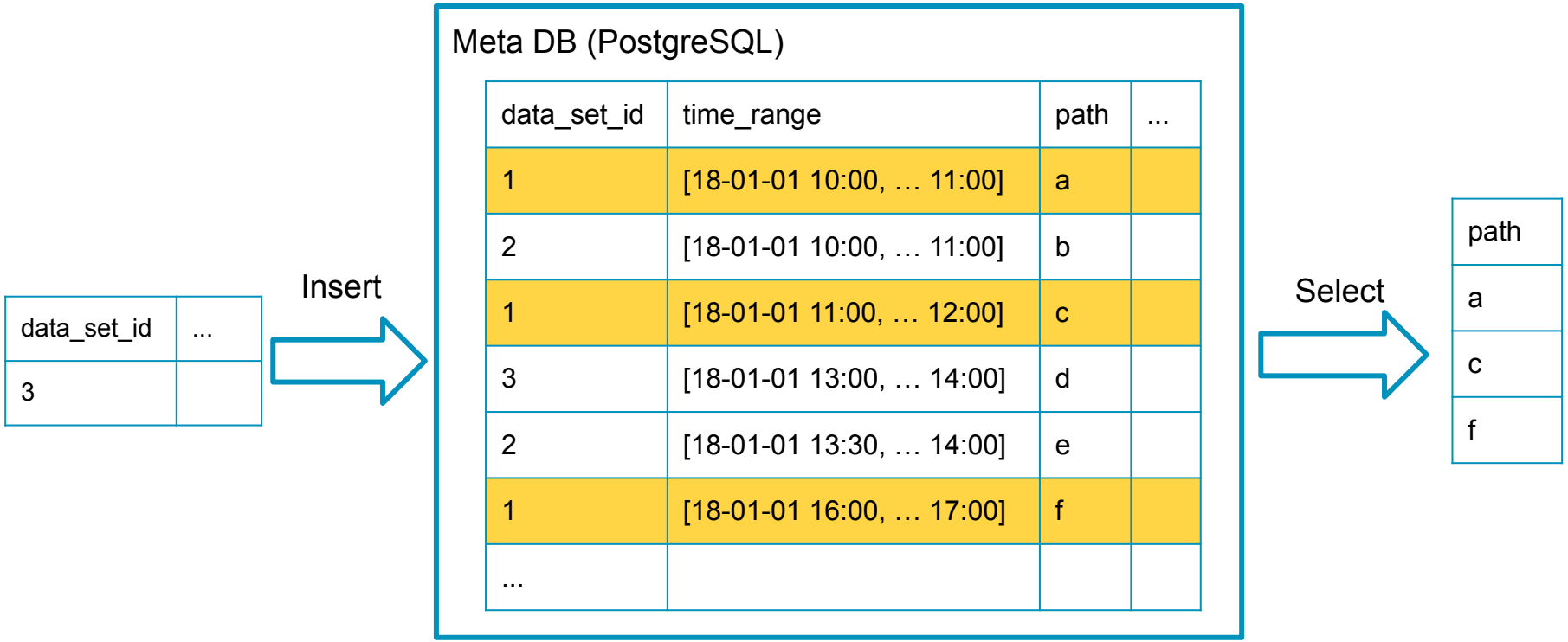


Summary of Presto Select Workload

- CPU bottleneck
  - Scale almost linearly when concurrency is less than # of cores
  - Throughput decreases when concurrency is higher than # of cores
  - Hot data is small enough to fit into DB shared buffer

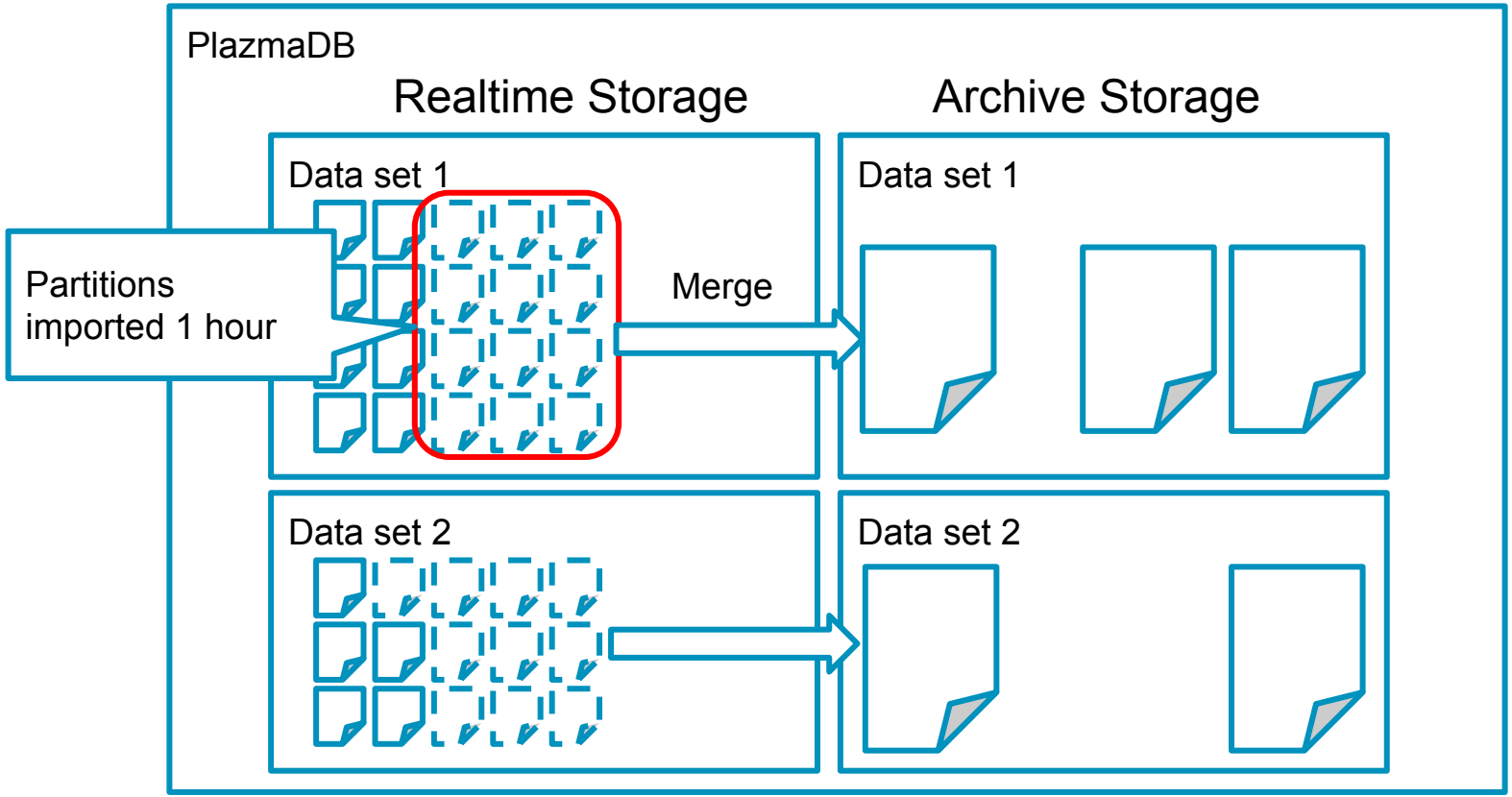
Benchmarking Mixed Workload

Mixing Streaming Import & Presto Select

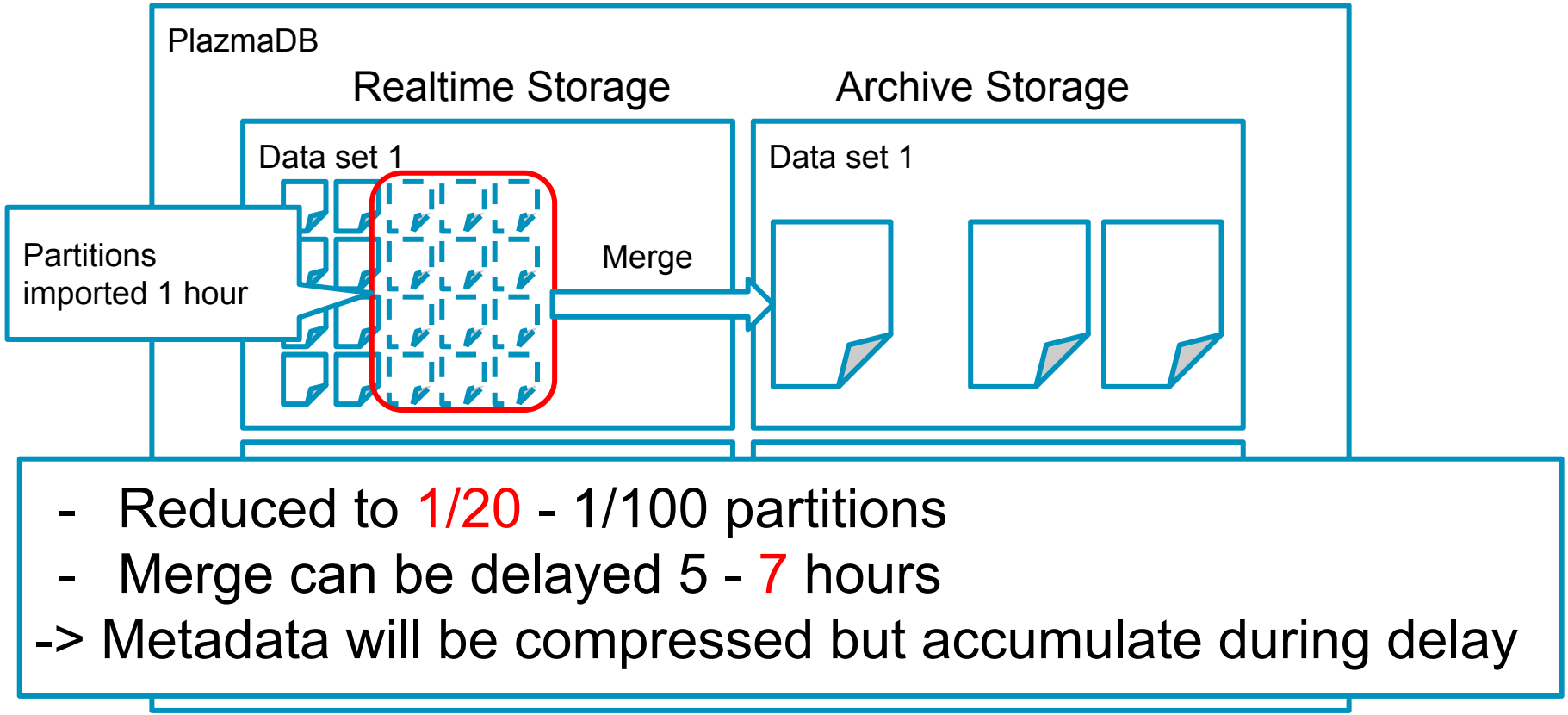


Insert : Select = 200 : 1

Realtime Storage & Archive Storage



Realtime Storage & Archive Storage



- Reduced to 1/20 - 1/100 partitions
- Merge can be delayed 5 - 7 hours

-> Metadata will be compressed but accumulate during delay

What is expectation?

- E.g. when concurrency = 64, what throughput will be?
  - Both Streaming Import and Presto Select were CPU bottleneck
- Ref: Single Workload Throughput
  - Streaming Import = 12500 tps
  - Presto Select = 95 tps
- Expectation
  - Streaming Import ~ 6000 tps ?
  - Presto Select ~ 45 tps ?

Result of mixed workload

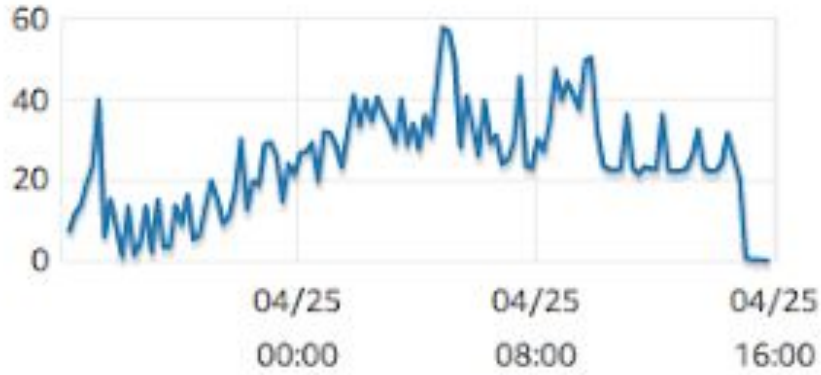
- Throughput when Concurrency = 64

	Streaming Import [tps]	Presto Select [tps]
mix concurrency=64	5547	27.6
Ref: single	12500	95

Resource Consumption

CPU utilization [%]

Read IO throughput [MB/s]



Bottleneck is changed to Disk IO

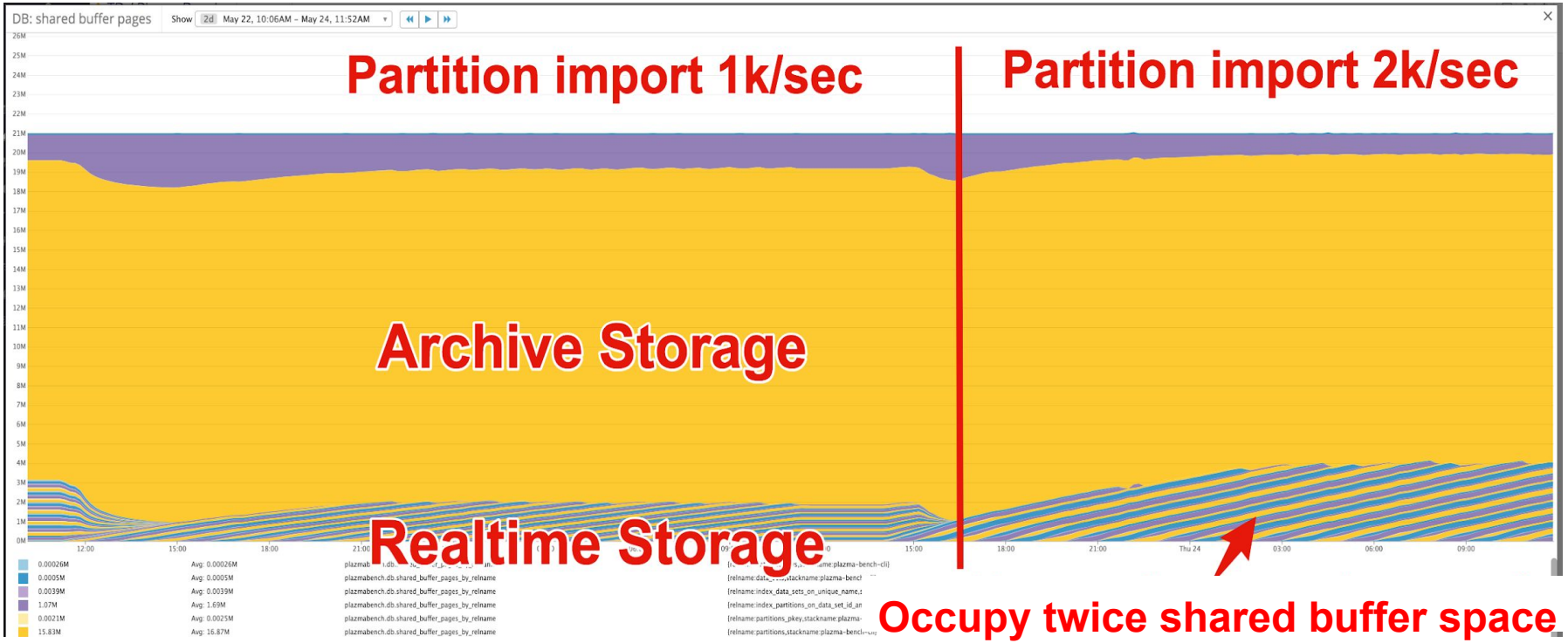
Increase PIOPS

	PIOPS	Streaming Import [tps]	Presto Select [tps]
mix concurrency=64	4k	5488	27.4
mix concurrency=64	20k	7179	35.9
Ref: single	4k	12500	95

Cache (DB Shared Buffer) Hit Ratio

	PIOPS	Streaming Import [tps]	Presto Select [tps]	RT storage size	Cache Hit Ratio
mix con=64	4k	5488	27.4	57GB	93%
mix con=64	20k	7179	35.9	75GB	89.5%
Ref: single	4k	12500	95		

Impact of Streaming Import increase



Impact of Cache Miss

- Avg # of selected rows per a Presto Select = 8000
- A postgres page mostly includes only 1 row for a data set
  - A page (8kB) has 20 - 30 rows, but different data sets' data are stored together
- # of pages to scan per a Presto Select = 8000 / 1 = 8000
- # of pages to scan per second = 8000 * TPS
  - E.g. TPS=35 -> 8000 * 35 = 280k pages/sec

    -> 1% cache miss causes 2800 IOPS

Another factor: Auto Vacuum

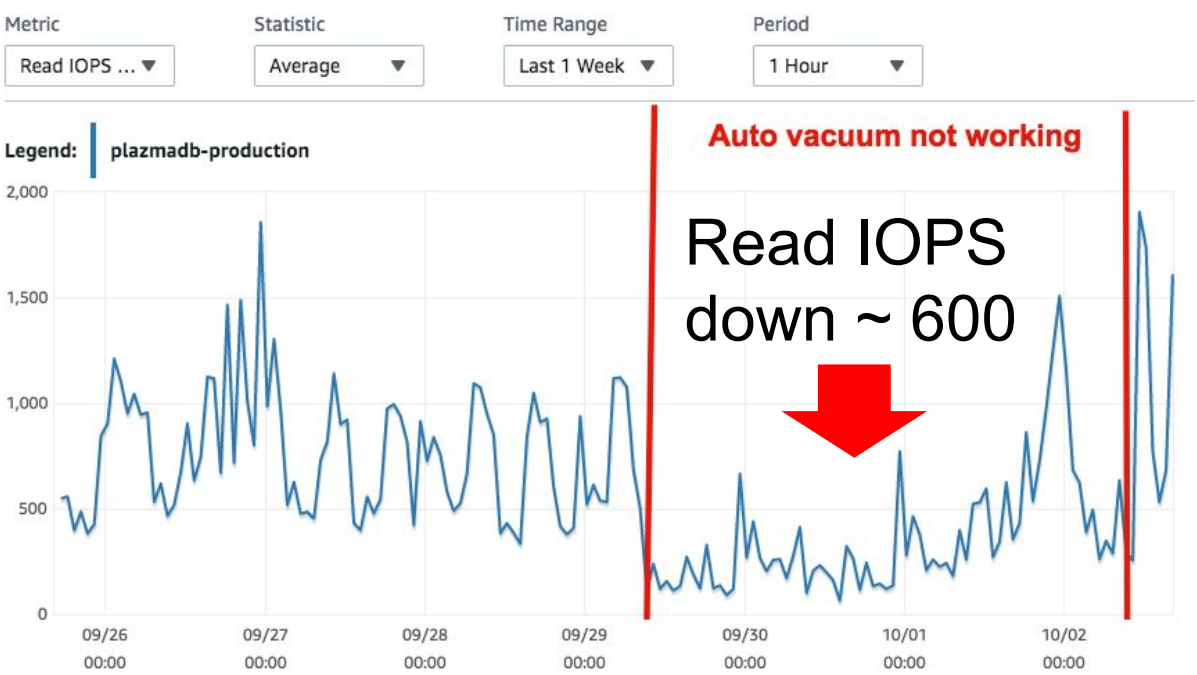
- Vacuum read cold data
  - Especially Vacuum Freeze does force full scan (before postgres 9.6)
- IO related Vacuum parameters

autovacuum_vacuum_cost_limit	400
autovacuum_vacuum_cost_delay	10ms
vacuum_cost_page_miss	10
vacuum_cost_page_dirty	20

- Max read IOPS = (max IO per vacuum) * (vacuum invoked per sec)
  = (400/10) * (1000/10) = 4000

Vacuum is tax of PostgreSQL

Someday auto vacuum stopped accidentally ..



To reduce IO

- Improve cache hit ratio
  - More RAM -> scale up / sharding
  - Improve locality
    -> Partitioning by data set ID: include more relevant rows in a (postgres) page
- Vacuum
  - Avoid full scan of vacuum freeze by using postgres 9.6 or newer

Q: What does Increase Cache Miss?

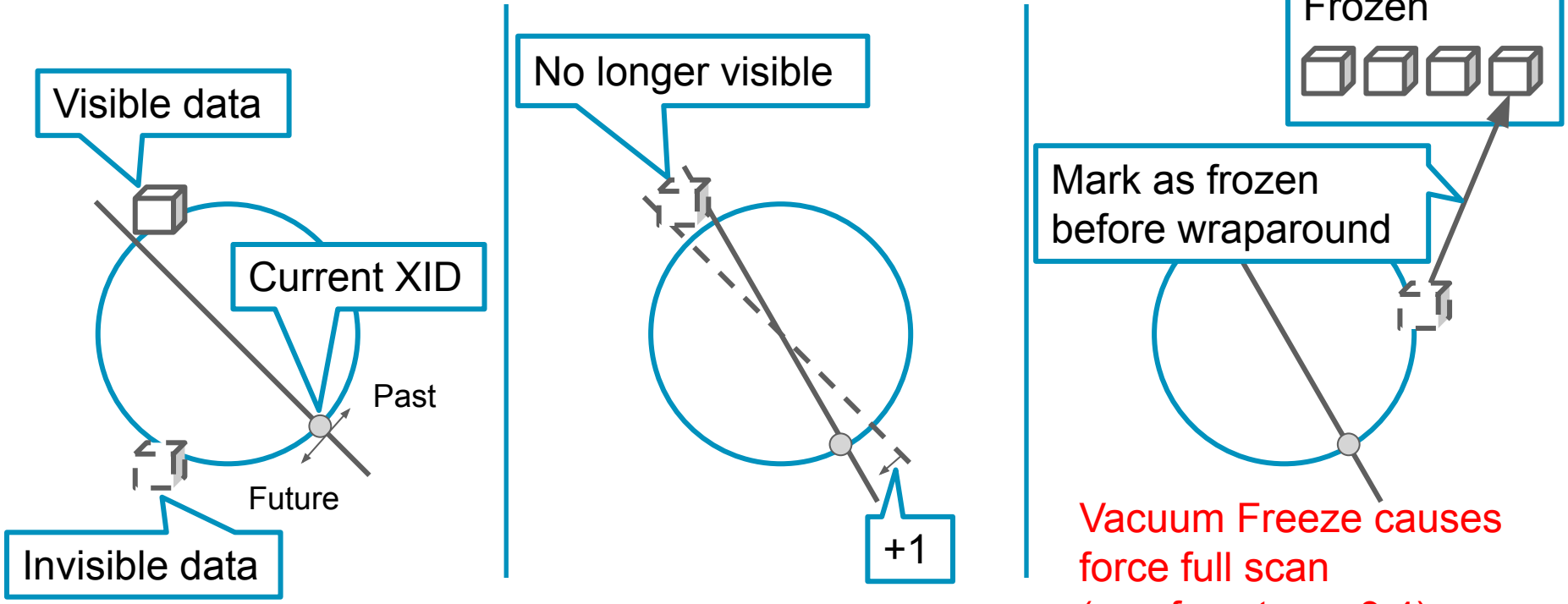
A: Vacuum Freeze

Vacuum to prevent transaction ID (XID) wraparound

https://www.postgresql.org/docs/current/static/routine-vacuuming.html

Vacuum Freeze

Reuse 32 bit XID space



What does Increase Cache Miss?

1. Streaming Import throughput is increased
2. More transaction IDs are consumed
3. More frequently Vacuum Freeze is invoked
4. More cold relations are entered in shared buffer by Vacuum
5. Hot data are more likely to be evicted
6. # of Cache misses is increased!

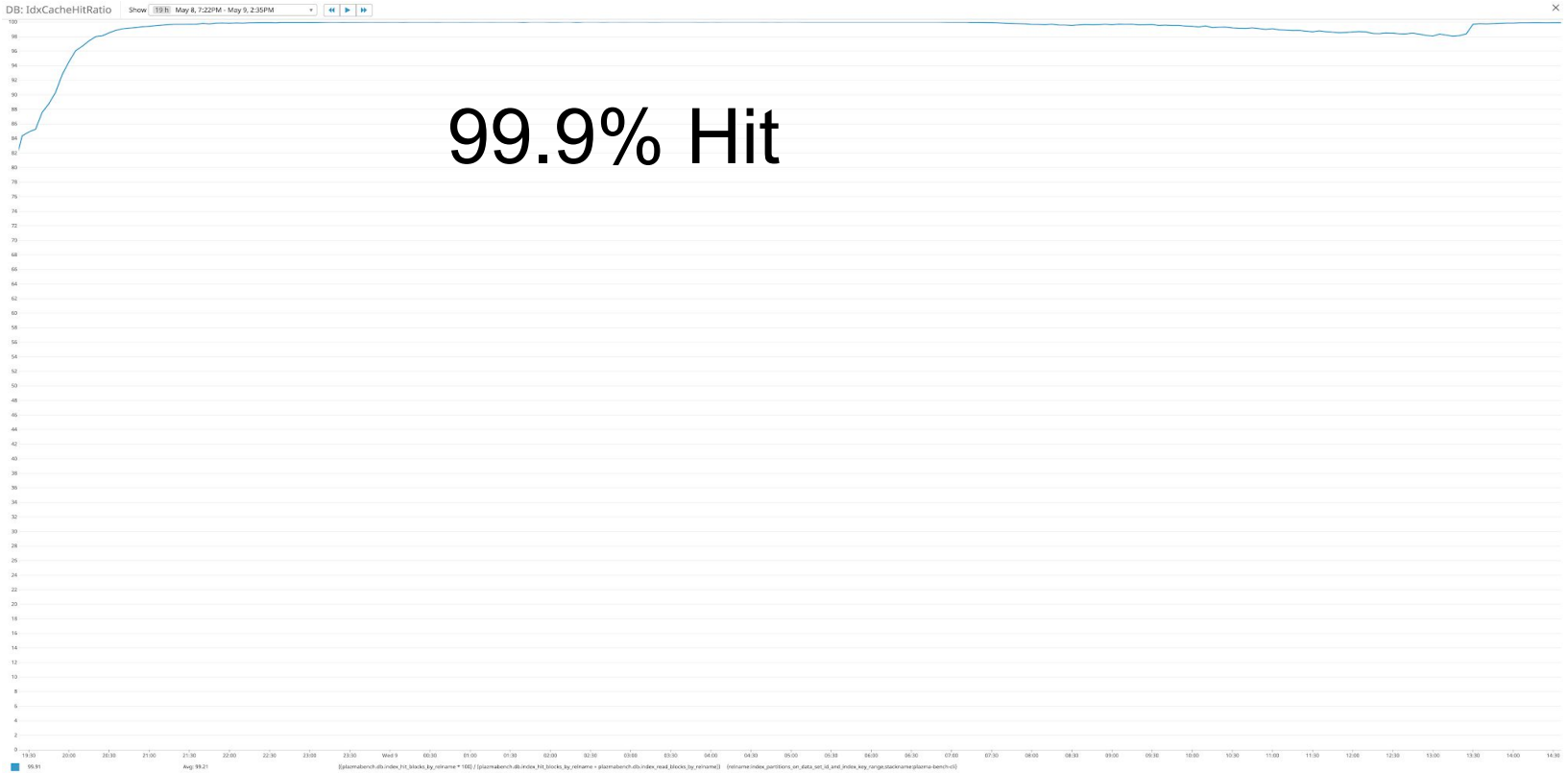
If streaming import throughput becomes 2x, vacuum run 2x frequently for 2x larger relations -> 4x larger cold data are entered in shared buffer

How Frequently does Vacuum Freeze occur?

Vacuum Freeze occurs every 350M txs (configurable)

Streaming Import TPS	Time to vacuum freeze
1k /sec	350k sec ~ 4 day
2k /sec	2 day
5k /sec	0.8 day
10k /sec	0.4 day

Cache (DB Shared Buffer) Hit Ratio



PlazmaDB Features

- Columnar format
  - Partitioned by time and optionally user defined column
- Schema less
- Partition index
- Partition optimization
  - Merge partitions
  - Realtime Storage & Archive Storage
- Transaction
  - Read committed isolation

Log data

time	orderid	user	region	price	...
2018-01-01 10:00:00	1	1	‘A’	10000	
2018-01-01 10:03:03	2	7	‘C’	40000	
2018-01-01 10:23:03	3	6	‘B’	3000	
2018-01-01 10:23:12	4	3	‘A’	5500	
2018-01-01 11:04:44	5	1	‘A’	20000	
2018-01-01 11:30:00	6	8	‘C’	3000	
...					

Accumulate over time

Many columns (attributes)

Analytical Query

time	orderid	user	region	price	...
2018-01-01 10:00:00	1	1	‘A’	10000	
2018-01-01 10:03:03	2	7	‘C’	40000	
2018-01-01 10:23:03	3	6	‘B’	3000	
2018-01-01 10:23:12	4	3	‘A’	5500	
2018-01-01 11:04:44	5	1	‘A’	20000	
2018-01-01 11:30:00	6	8	‘C’	3000	
...					

```
SELECT
  region,
  SUM(price)
FROM
  orders
WHERE time >= ‘2018-01-01 10:00’
    AND time <= ‘2018-01-01 11:00’
GROUP BY
  region
```

Few part of columns

Filter by time window

Inefficiency of Row Based Format

time	orderid	user	region	price	...
2018-01-01 10:00:00	1	1	‘A’	10000	
2018-01-01 10:03:03	2	7	‘C’	40000	
2018-01-01 10:23:03	3	6	‘B’	3000	
2018-01-01 10:23:12	4	3	‘A’	5500	
2018-01-01 11:04:44	5	1	‘A’	20000	
2018-01-01 11:30:00	6	8	‘C’	3000	
...					


```
SELECT
  region,
  SUM(price)
FROM
  orders
WHERE time >= ‘2018-01-01 10:00’
      AND time <= ‘2018-01-01 11:00’
GROUP BY
  region
```

Columnar Format

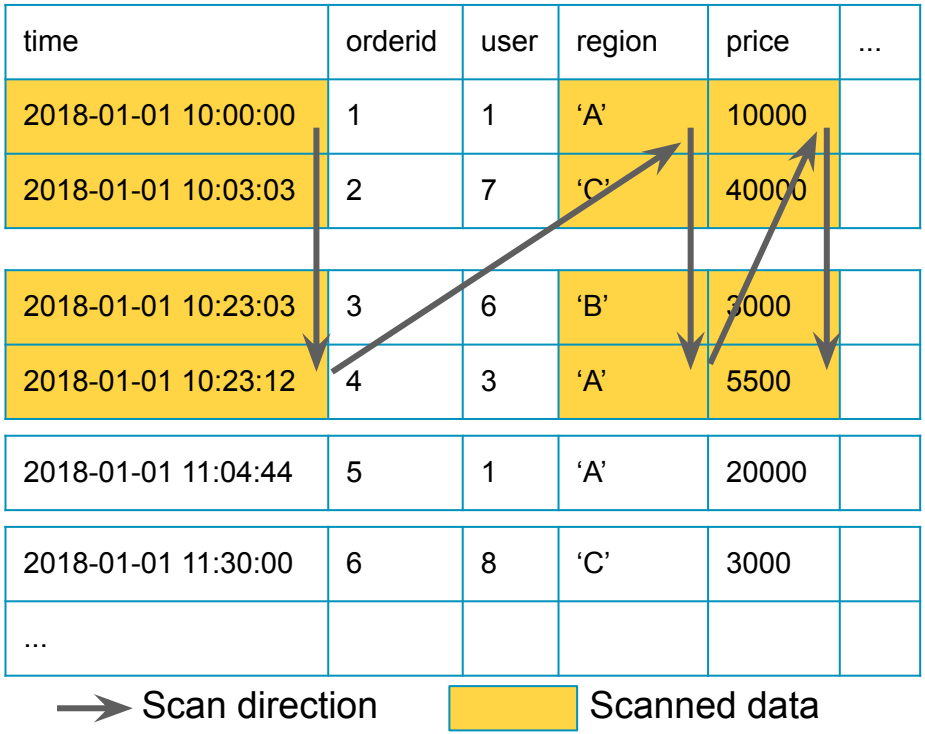
time	orderid	user	region	price	...
2018-01-01 10:00:00	1	1	‘A’	10000	
2018-01-01 10:03:03	2	7	‘C’	40000	
2018-01-01 10:23:03	3	6	B’	3000	
2018-01-01 10:23:12	4	3	‘A’	5500	
2018-01-01 11:04:44	5	1	‘A’	20000	
2018-01-01 11:30:00	6	8	‘C’	3000	
...					


```
SELECT
  region,
  SUM(price)
FROM
  orders
WHERE time >= ‘2018-01-01 10:00’
    AND time <= ‘2018-01-01 11:00’
GROUP BY
  region
```

Few part of columns

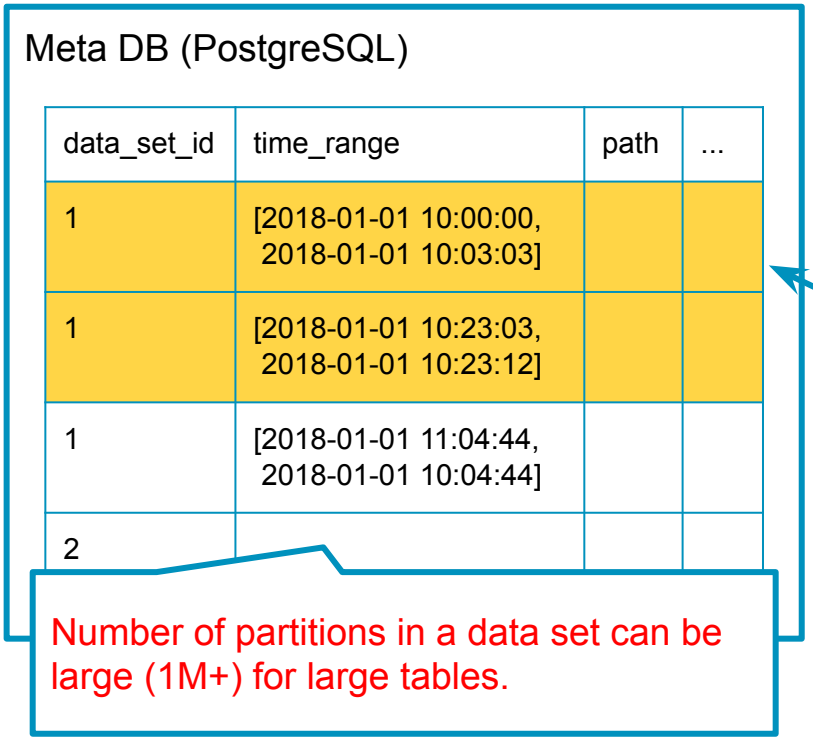
Filter by time window

Skip Partition Scan




```
SELECT
  region,
  SUM(price)
FROM
  orders
WHERE time >= '2018-01-01 10:00'
      AND time <= '2018-01-01 11:00'
GROUP BY
  region
```

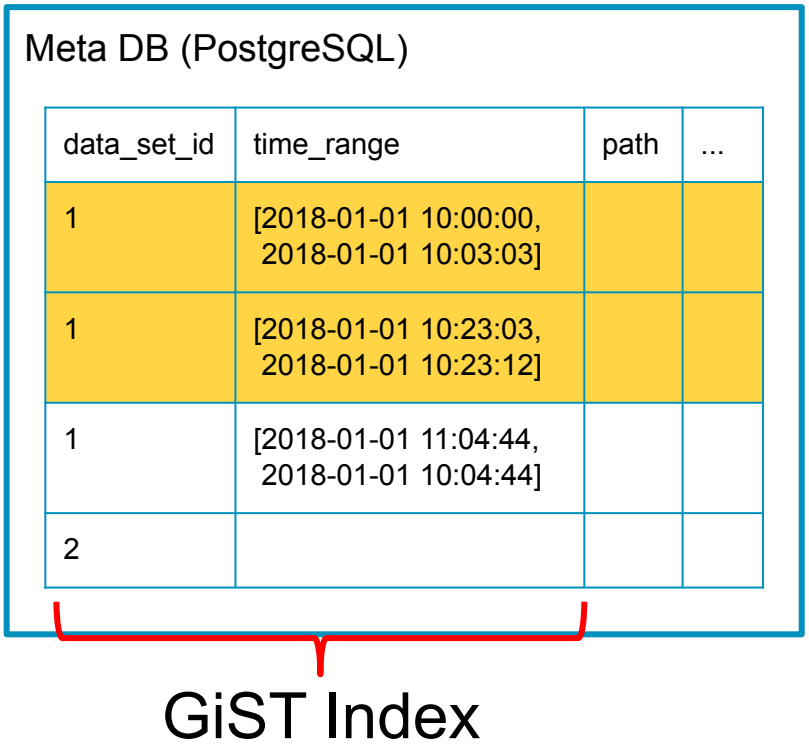
How to find Partitions?



?

```
SELECT
  region,
  SUM(price)
FROM
  orders  -- assume this is data set 1
WHERE time >= '2018-01-01 10:00'
    AND time <= '2018-01-01 11:00'
GROUP BY
  region
```

Range Type and GiST Index of PostgreSQL



Range type

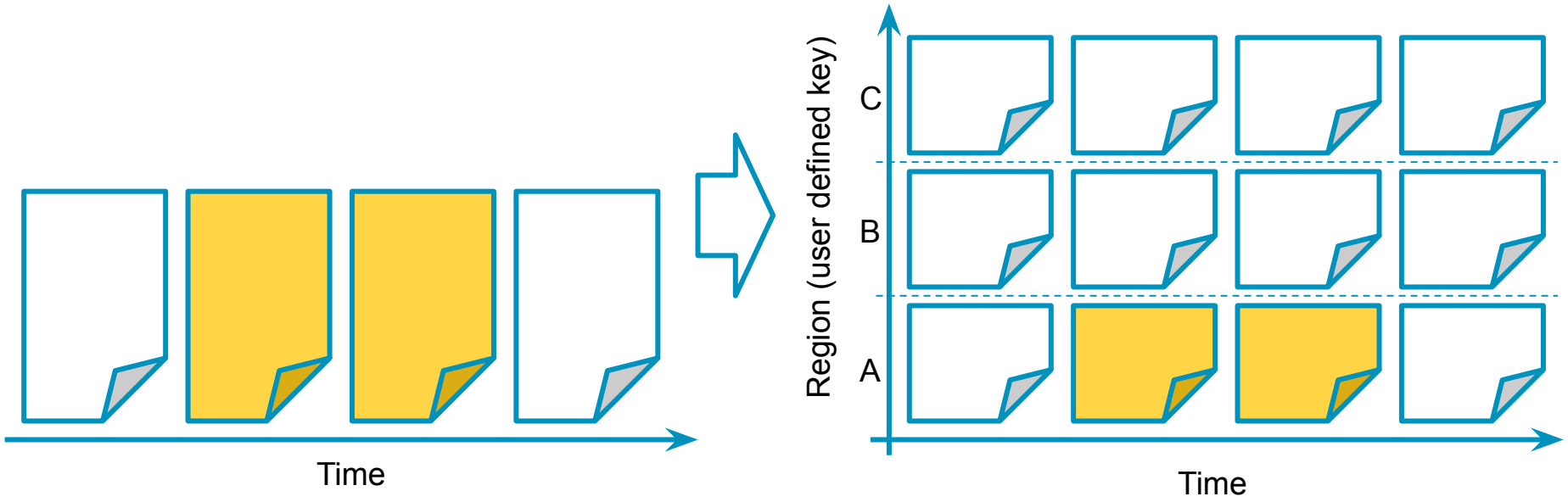
- Overlap operator

  time_range && [2018-01-01 10:00, 2018-01-01 11:00]

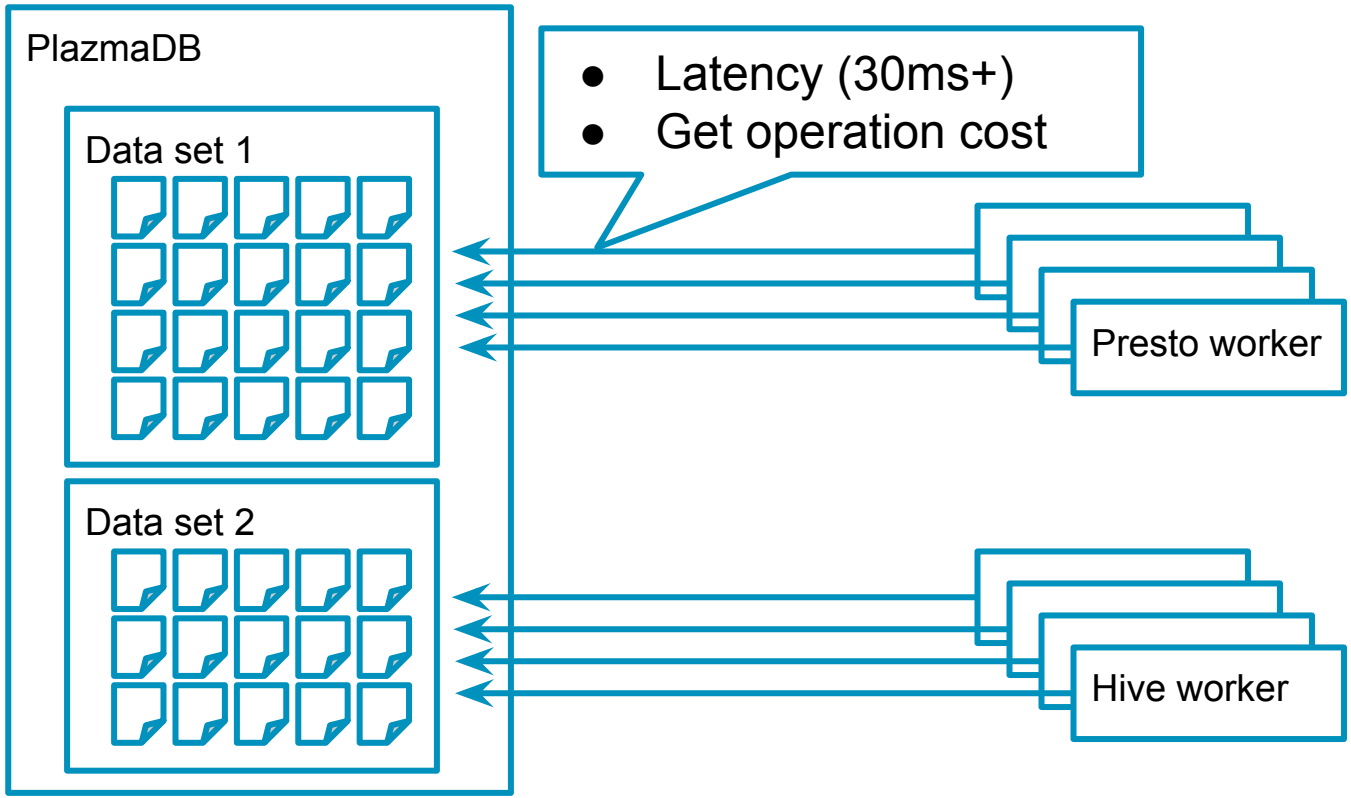
Overlap is checked by index scan

User Defined Partition (Beta)

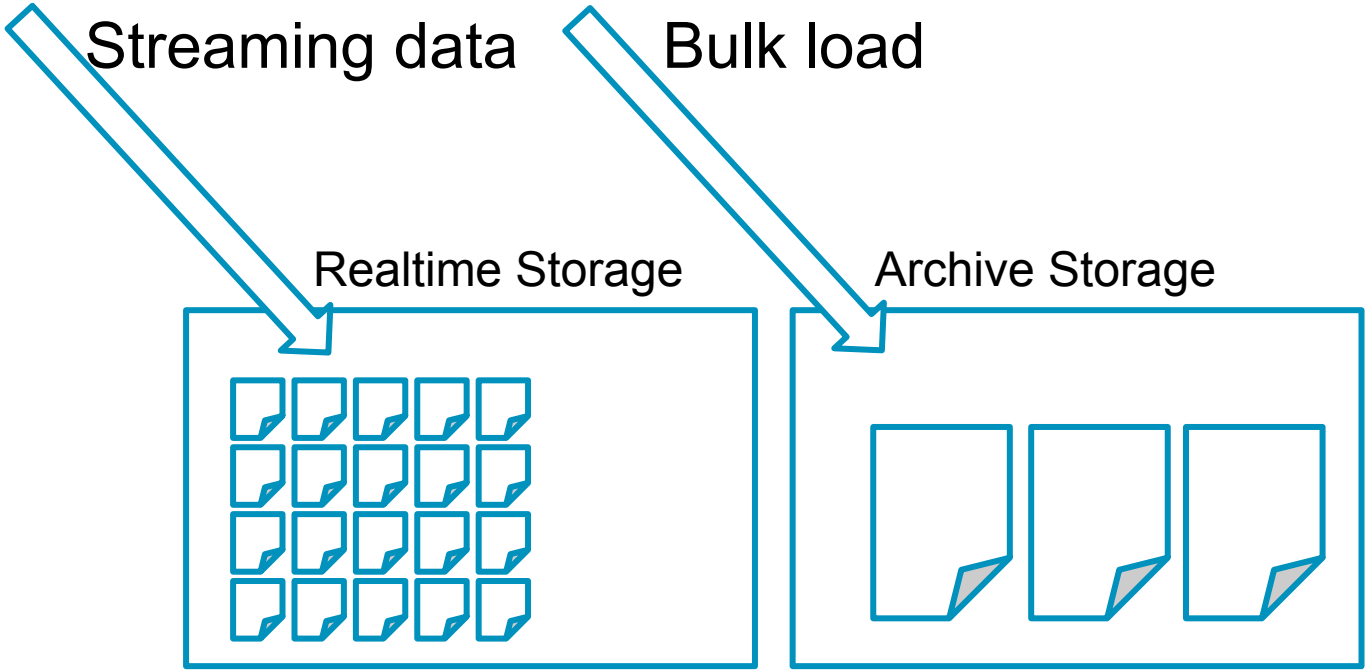
SELECT … FROM … WHERE time > … AND time <... AND region = ‘A’



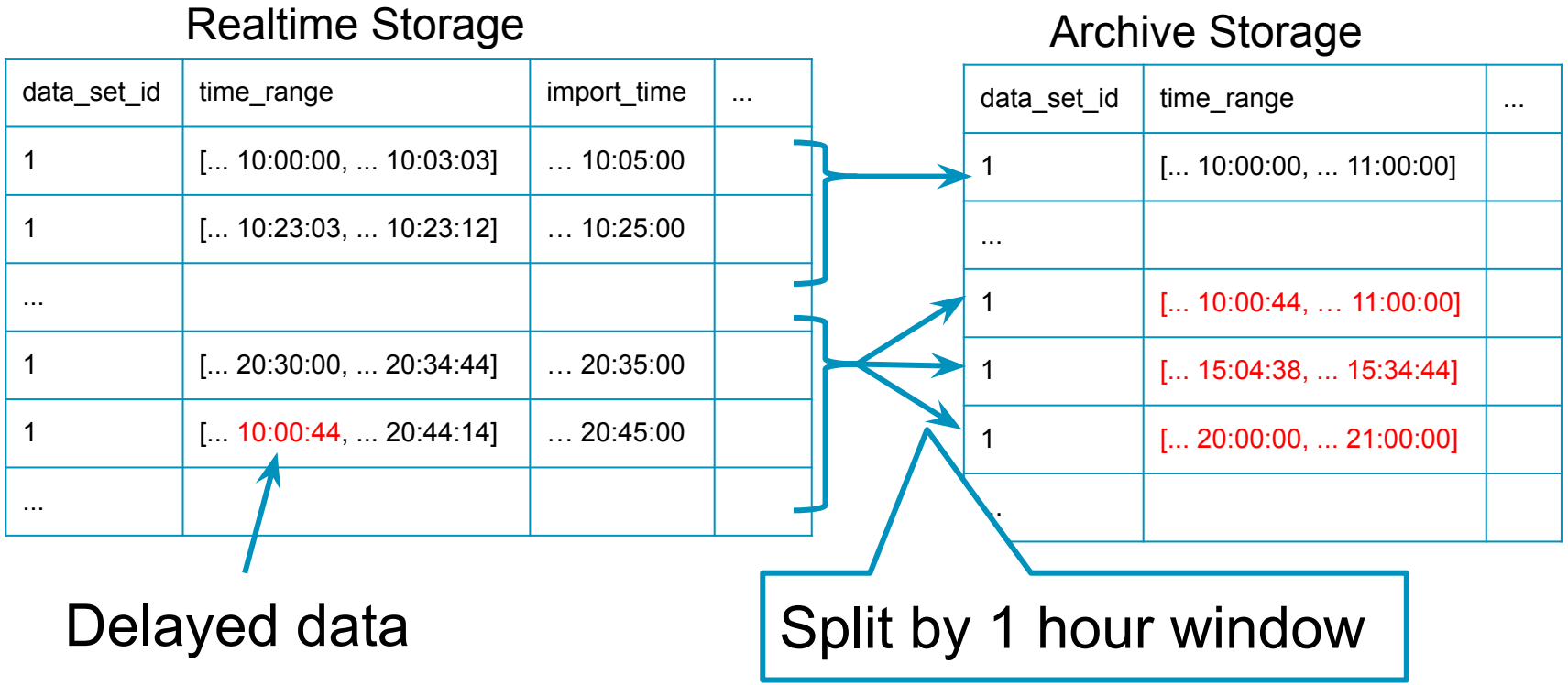
Partition Fragmentation



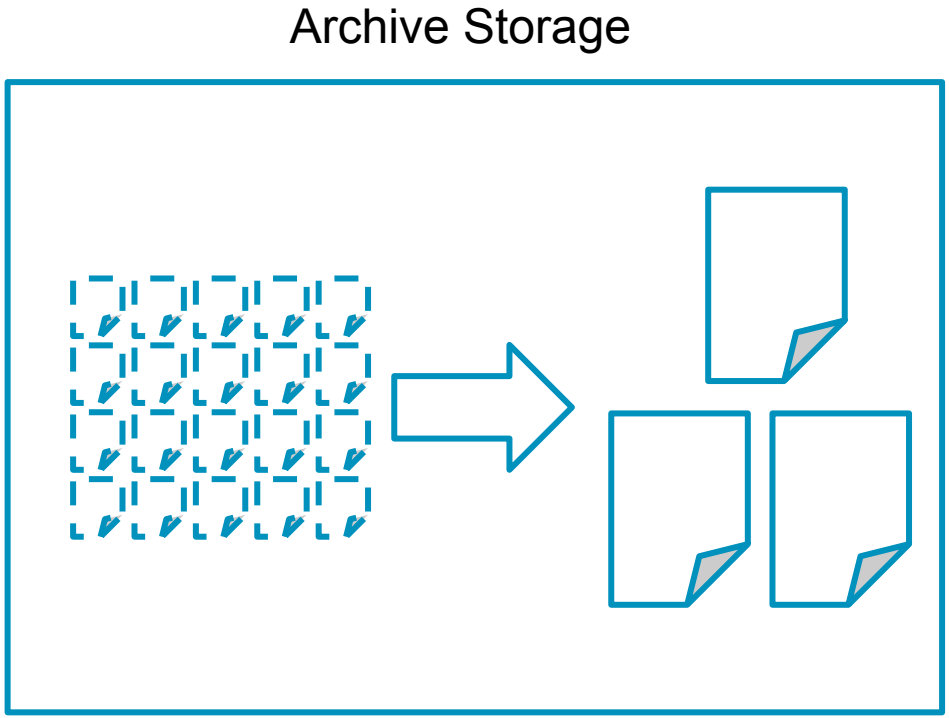
Streaming & Bulk data upload



Fragmentation of Archive Storage



Remerge Partitions

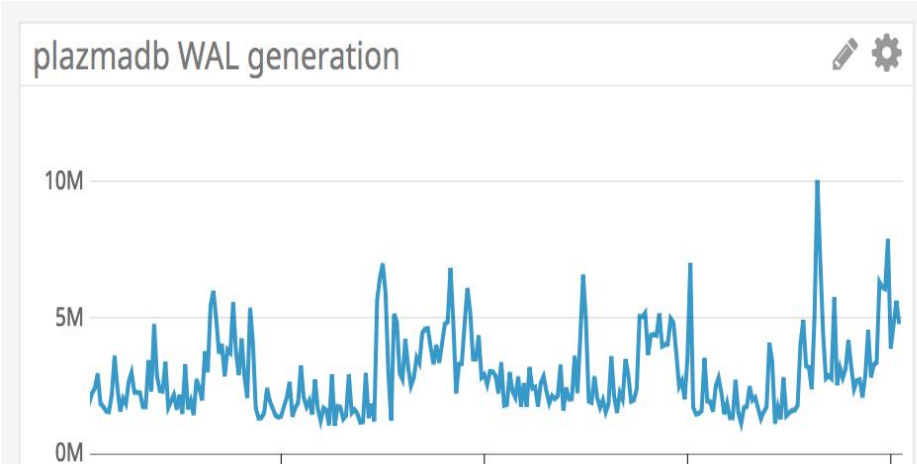


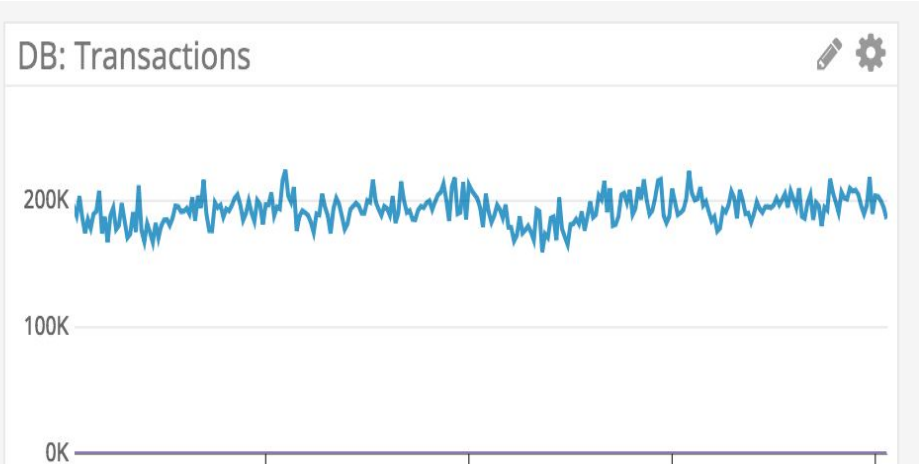
Re: PlazmaDB features

- Columnar format
  - Partitioned by time and optionally user defined column
- Schema less
- Partition index
- Partition optimization
  - Merge partitions
  - Realtime Storage & Archive Storage
- Transaction
  - Read committed isolation

Current PlazmaDB & Future Challenges

Write Workload on Meta DB





3 MB / sec

200k transaction / min
~ 3k transaction / sec

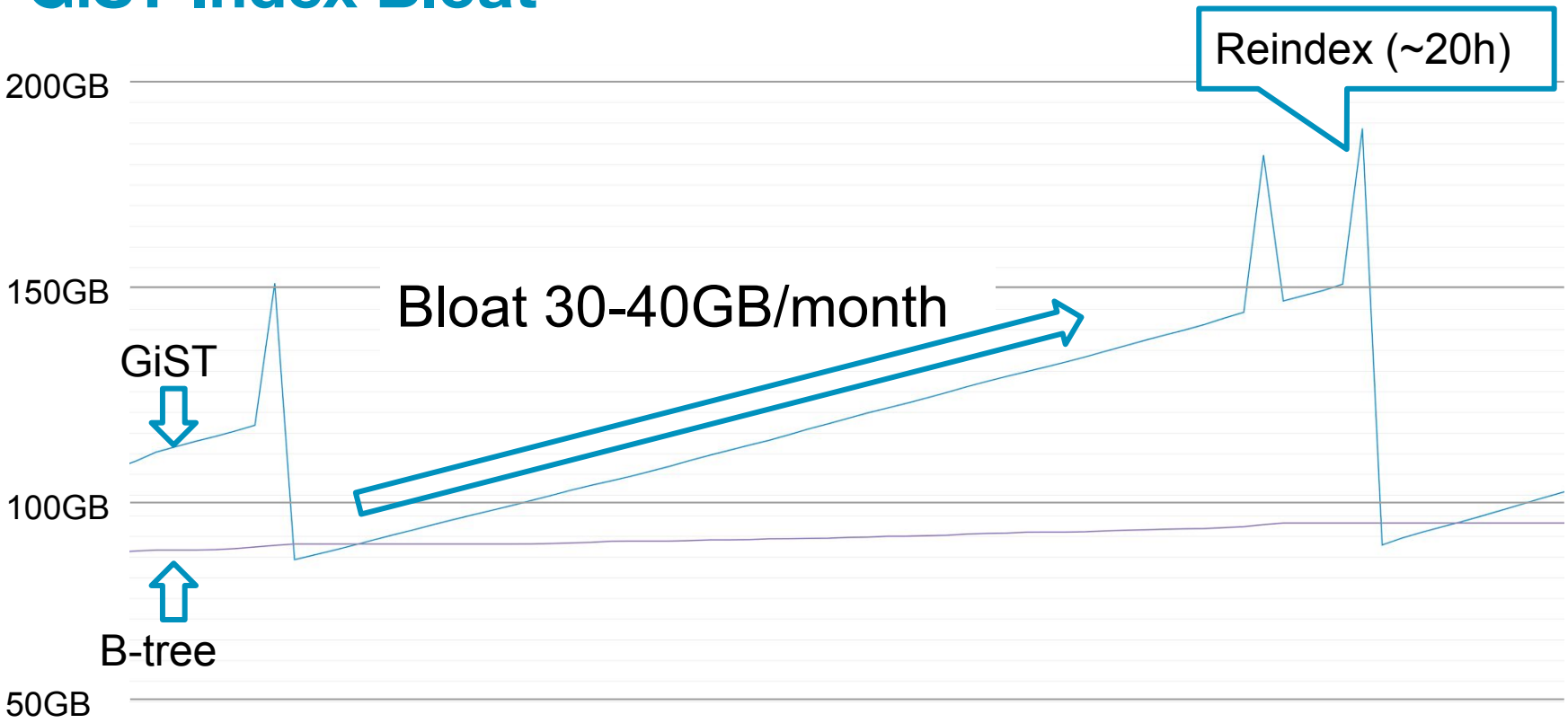
PostgreSQL Auto VACUUM (FREEZE)

VACUUM FREEZE: Vacuum to prevent transaction ID wraparound failures

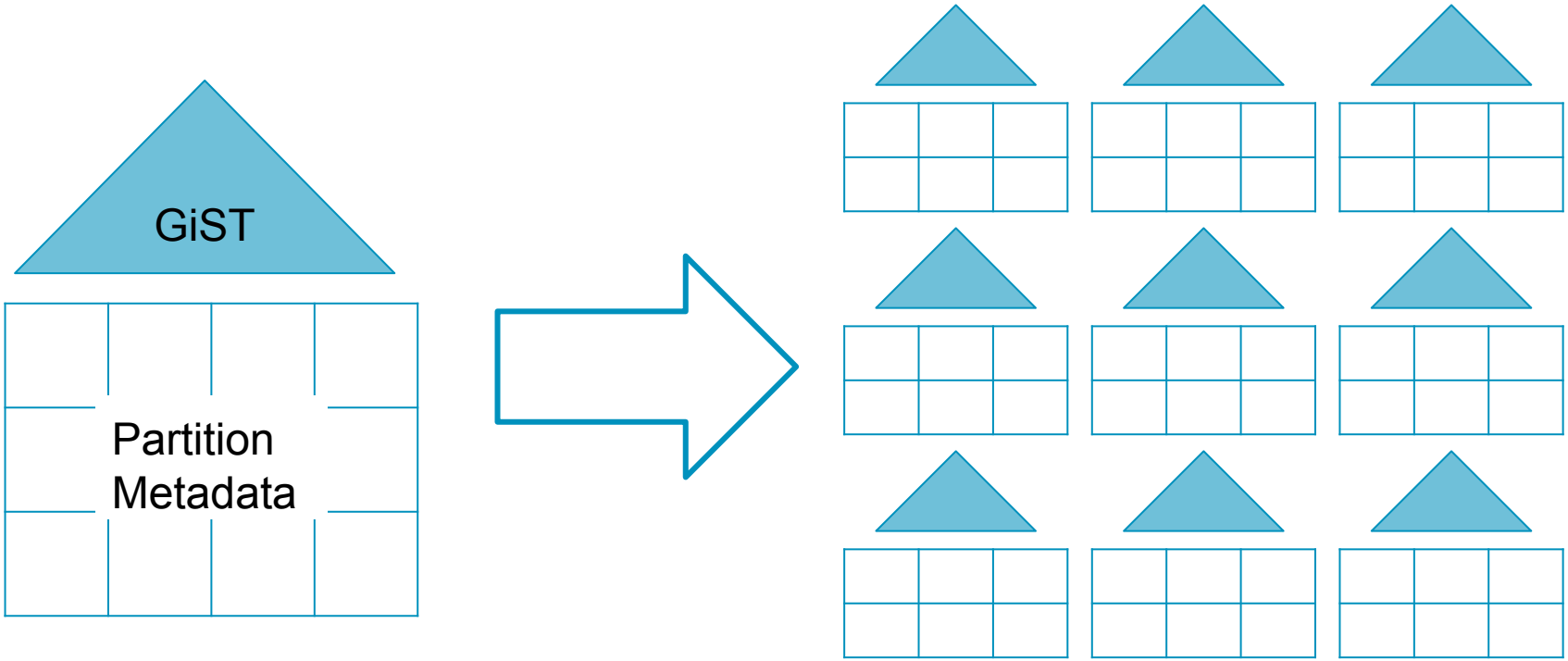
- Force full scan on relation (as of PostgreSQL 9.4)
  - Hot data may be evicted to scan relations for vacuum
    => Read workload can be affected
  - PostgreSQL 9.6 or later mitigate the problem

- The more transaction IDs are consumed, the more vacuum can be happened
  - In our case, it happens every 2-3 day

GiST Index Bloat

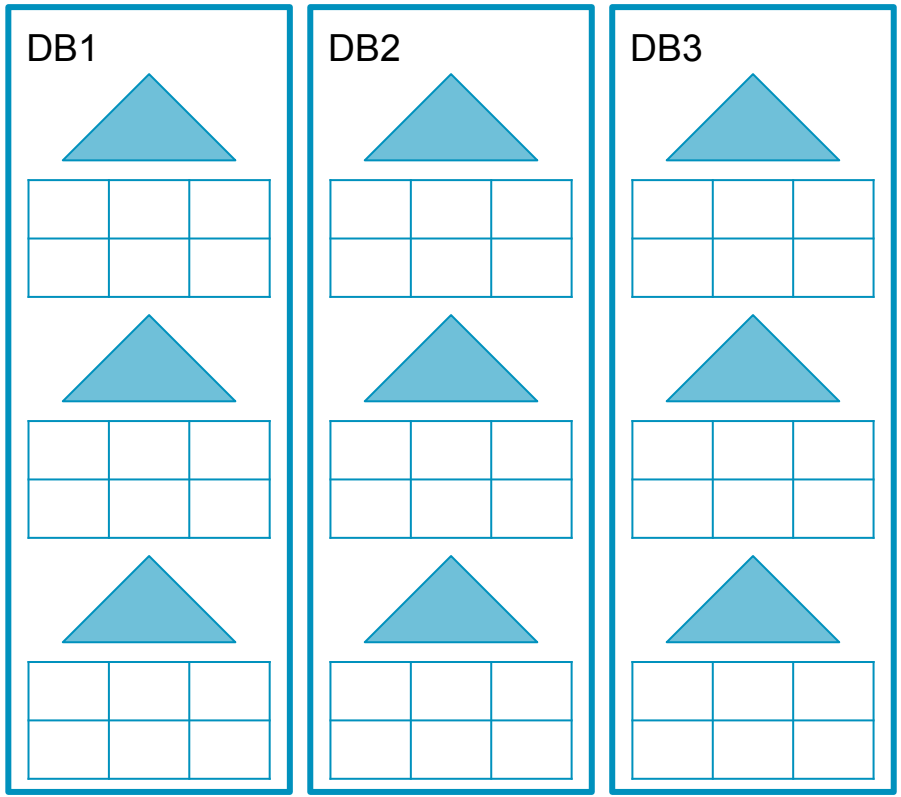


Metadata Table Partitioning

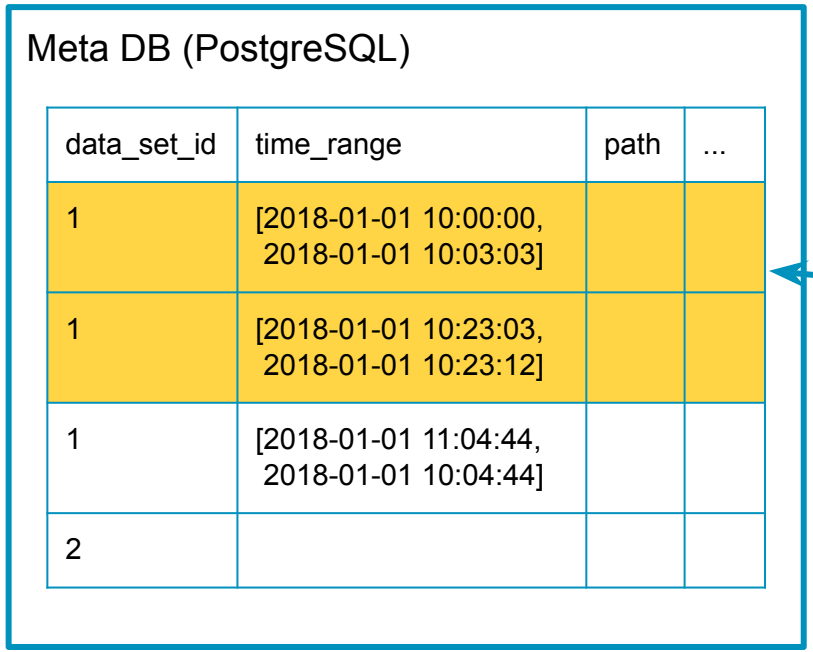


One relation’s reindex becomes shorter and space saving

Meta DB Scale out



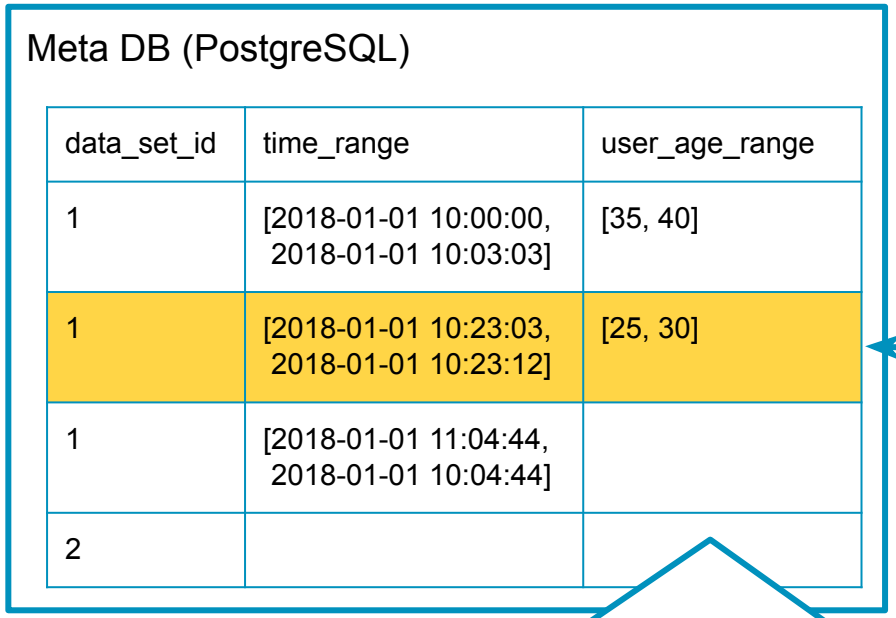
More Partition Skip




```
SELECT
  region,
  SUM(price)
FROM
  orders
WHERE time >= '2018-01-01 10:00'
    AND time <= '2018-01-01 11:00'
    AND user_age >= 20
    AND user_age <= 30
GROUP BY
  region
```

Scan partition & Filter

More Partition Skip



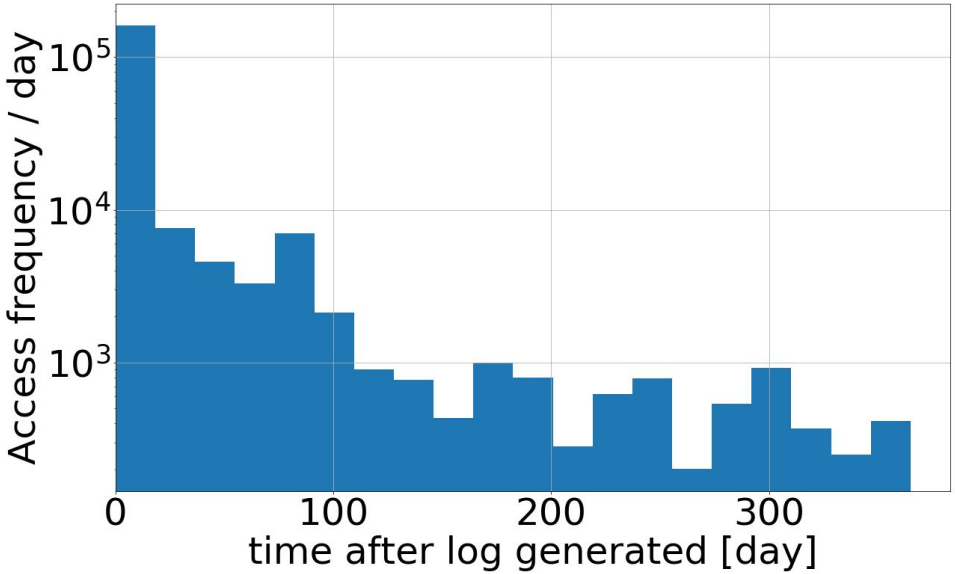

```
SELECT
  region,
  SUM(price)
FROM
  orders
WHERE time >= '2018-01-01 10:00'
      AND time <= '2018-01-01 11:00'
      AND user_age >= 20
      AND user_age <= 30
GROUP BY
  region
```

Store metadata on frequently accessed columns

Smart Partition Selection for Remerge

Remerge is resource consuming

- Current
  - Data set size
  - # of Partitions
- Idea
  - Access frequency
  - Data freshness
    Fresh data is likely to be hot



Summary

- PlazmaDB: Storage Layer of Arm Treasure Data Analytics Platform
  - Optimization for Analytical Queries
    - Columnar + Time Partitioning + Partition Index
  - Optimization for Streaming data
    - Realtime & Archive Storage + Merge Partition
- Challenges
  - Reduce impact of PostgreSQL VACUUM FREEZE
  - GiST index management
  - More Partition Optimization
    - Enrich Metadata
    - Smart Remerge