Hive Optimization

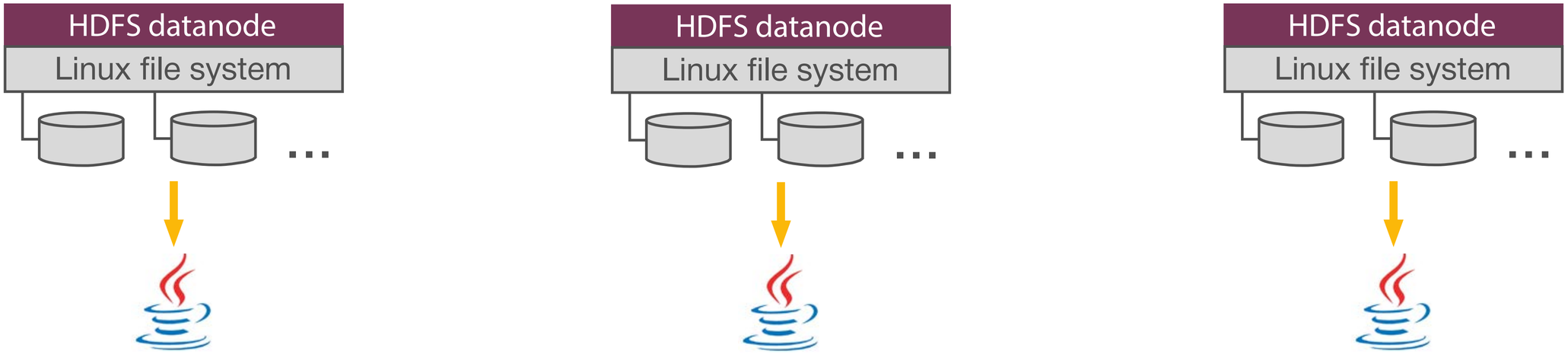
Bucketing, Sorting, Partitioning

???

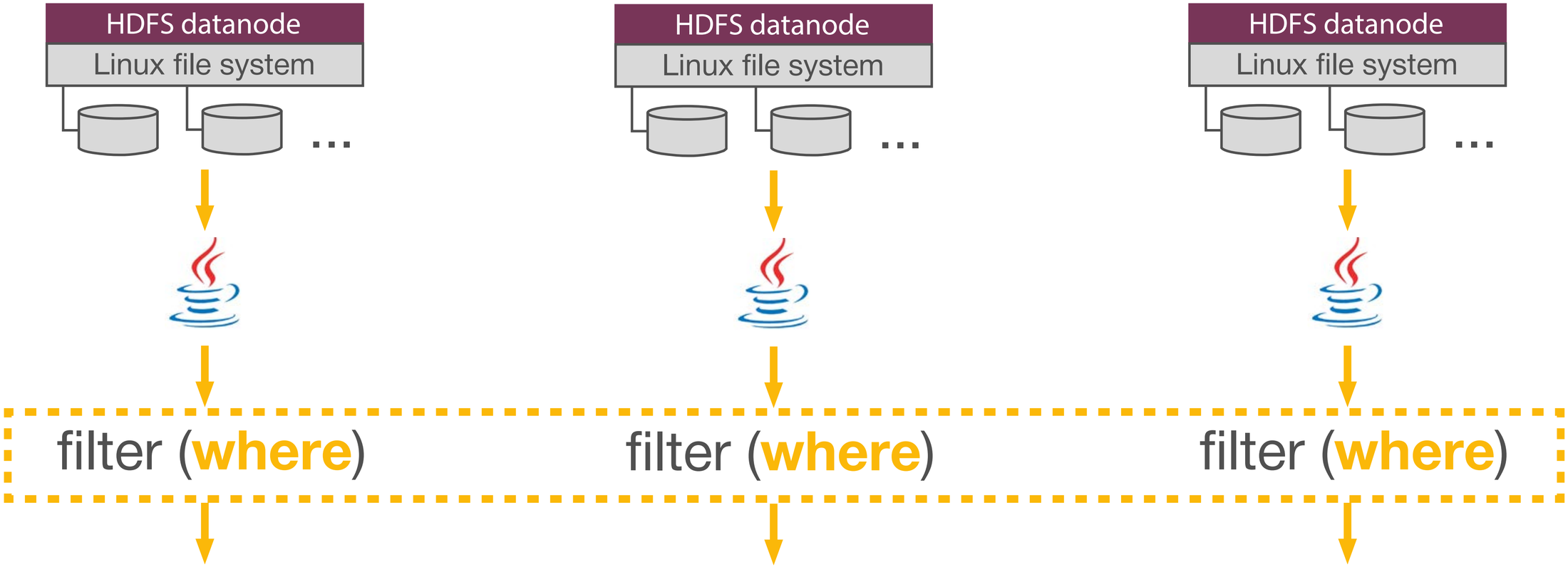

```
SELECT region_city, COUNT(1) AS hit_count
FROM access_log JOIN geo_base
ON (access_log.host = geo_base.host)
WHERE access_log.datetime BETWEEN "recent_date" AND "now"
GROUP BY region_city ORDER BY hit_count LIMIT 100
```

```
SELECT region_city, COUNT(1) AS hit_count
FROM access_log JOIN geo_base
ON (access_log.host = geo_base.host)
WHERE access_log.datetime BETWEEN "recent_date" AND "now"
GROUP BY region_city ORDER BY hit_count LIMIT 100
```

```
SELECT region_city, COUNT(1) AS hit_count
FROM access_log JOIN geo_base
ON (access_log.host = geo_base.host)
WHERE access_log.datetime BETWEEN "recent_date" AND "now"
GROUP BY region_city ORDER BY hit_count LIMIT 100
```



```
SELECT region_city, COUNT(1) AS hit_count
FROM access_log JOIN geo_base
ON (access_log.host = geo_base.host)
WHERE access_log.datetime BETWEEN "recent_date" AND "now"
GROUP BY region_city ORDER BY hit_count LIMIT 100
```



hdfs:///access_logs/

...

— 2017_01_20

— 2017_01_21

— 2017_01_22

...

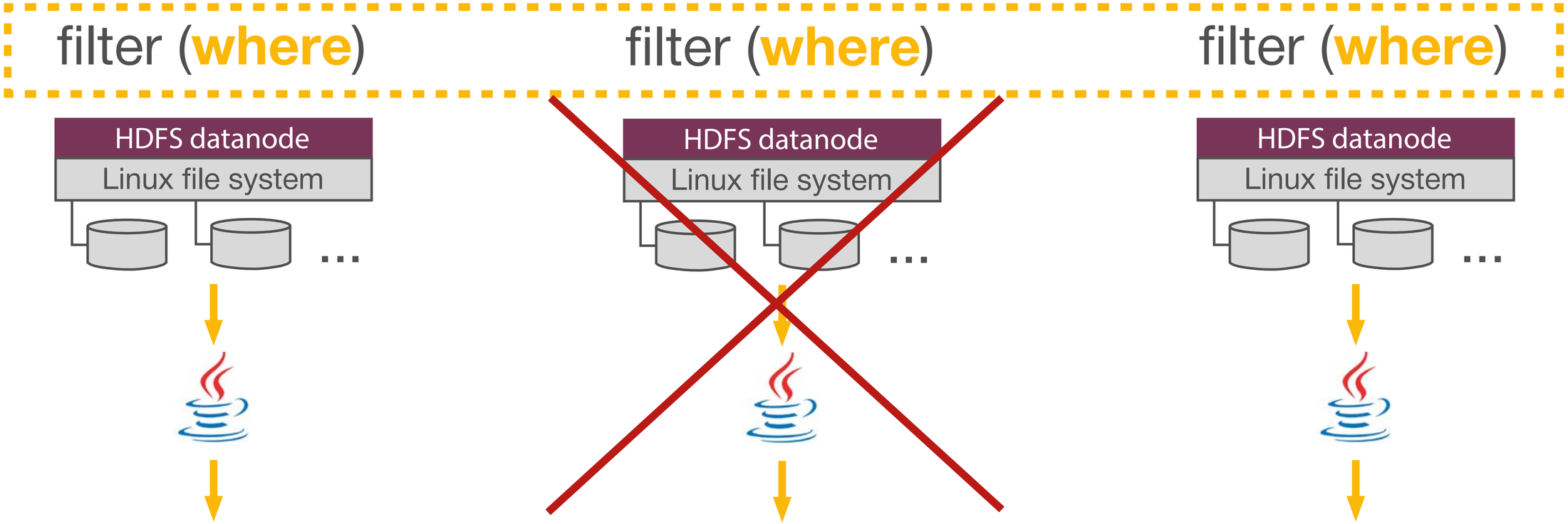
— "today"

hdfs:///access_logs/

...
— 2017_01_20
— 2017_01_21
— 2017_01_22
...
— "today"

```
CREATE TABLE partitioned_access_log (
   ip STRING,
   ...
)
PARTITIONED BY (request_date STRING)
...;
```

```
SELECT region_city, COUNT(1) AS hit_count
FROM access_log JOIN geo_base
ON (access_log.host = geo_base.host)
WHERE access_log.datetime BETWEEN "recent_date" AND "now"
GROUP BY region_city ORDER BY hit_count LIMIT 100
```



```
hdfs:///access_logs/
...
— 2017
——— 01
————— 20
————— 21
————— 22
...
```

```
hdfs:///access_logs/
...
— 2017
——— 01
————— 20
————— 21
————— 22
...
```

```
CREATE TABLE partitioned_access_log (
    ip STRING,
    ...
)
PARTITIONED BY (year STRING, month STRING, day STRING)
...;
```

```sql
CREATE TABLE partitioned_access_log (
    ip STRING,
    ...
)
PARTITIONED BY (year STRING, month STRING, day STRING)
...;
```

```sql
FROM raw_access_log
INSERT OVERWRITE TABLE partitioned_access_log
PARTITION (year="2017", month="03", day="25")
SELECT ip, ...
```

```
FROM raw_access_log
INSERT OVERWRITE TABLE partitioned_access_log
PARTITION (year=??, month=??, day=??)
SELECT ip, ...
```

```
FROM raw_access_log
INSERT OVERWRITE TABLE partitioned_access_log
PARTITION (year, month, day)
SELECT ip, ..., year, month, day
```

1. **partitioned columns go at the end**

```
FROM raw_access_log
INSERT OVERWRITE TABLE partitioned_access_log
PARTITION (year, month, day)
SELECT ip, ..., year, month, day
```

1. partitioned columns go at the end
2. **partitioned columns order is important**

```
SET hive.exec.max.dynamic.partitions=2048;
SET hive.exec.max.dynamic.partitions.pernode=256;
SET hive.exec.max.created.files=10000;
```

```
FROM raw_access_log
INSERT OVERWRITE TABLE partitioned_access_log
PARTITION (year, month, day)
SELECT ip, ..., year, month, day
```

1. partitioned columns go at the end
2. partitioned columns order is important
3. **use configuration parameters**

```
SET hive.exec.max.dynamic.partitions=2048;
SET hive.exec.max.dynamic.partitions.pernode=256;
SET hive.exec.max.created.files=10000;
SET hive.error.on.empty.partition=true;

FROM raw_access_log
INSERT OVERWRITE TABLE partitioned_access_log
PARTITION (year, month, day)
SELECT ip, ..., year, month, day
```

1. partitioned columns go at the end
2. partitioned columns order is important
3. use configuration parameters
4. **control empty partitions**

```
SET hive.exec.max.dynamic.partitions=2048;
SET hive.exec.max.dynamic.partitions.pernode=256;
SET hive.exec.max.created.files=10000;
SET hive.error.on.empty.partition=true;

FROM raw_access_log
INSERT OVERWRITE TABLE partitioned_access_log
PARTITION (year="2017", month, day)
SELECT ip, ..., month, day
```

1. partitioned columns go at the end
2. partitioned columns order is important
3. use configuration parameters
4. control empty partitions

```
$ hdfs dfs -ls /path/to/access_logs
part-00000
part-00001
...
part-00197
part-00198
part-00199
```

```
$ hdfs dfs -ls /path/to/access_logs
part-00000
part-00001
...
part-00197
part-00198
part-00199
```

```
CREATE TABLE granular_access_log (
    ip STRING,
    ...
)
PARTITIONED BY (request_date STRING)
CLUSTERED BY (column_name, ...) INTO 200 BUCKETS
...;
```

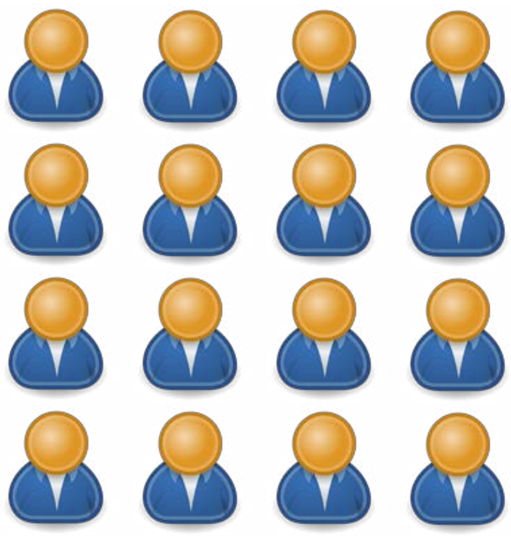
```
$ hdfs dfs -ls /path/to/access_logs
part-00000
part-00001
...
part-00197
part-00198
part-00199
```

```
CREATE TABLE granular_access_log (
    ip STRING,
    ...
)
PARTITIONED BY (request_date STRING)
CLUSTERED BY (column_name, ...)
    SORTED BY (some_column_name, ...)
    INTO 200 BUCKETS
...;
```

```
$ hdfs dfs -ls /path/to/access_logs
part-00000
part-00001
...
part-00197
part-00198
part-00199
```

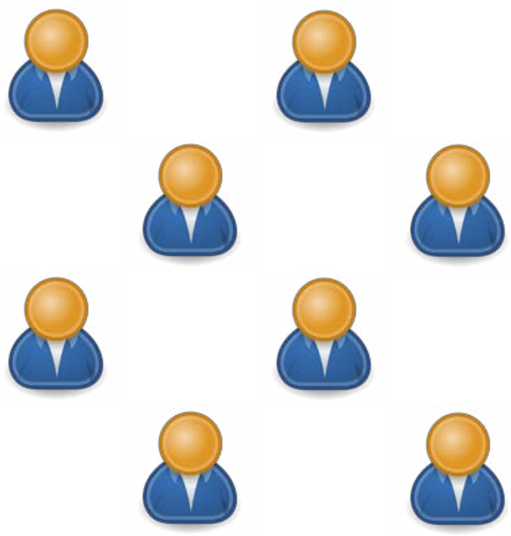
```
CREATE TABLE granular_access_log (
    ip STRING,
    ...
)
PARTITIONED BY (request_date STRING)
CLUSTERED BY (user_id)
    INTO 200 BUCKETS
...;
```

```
$ hdfs dfs -ls /path/to/access_logs
part-00000
part-00001
...
part-00197
part-00198
part-00199
```



```
CREATE TABLE granular_access_log (
    ip STRING,
    ...
)
PARTITIONED BY (request_date STRING)
CLUSTERED BY (user_id)
    INTO 200 BUCKETS
...;
```

```
$ hdfs dfs -ls /path/to/access_logs
part-00000
part-00001
...
part-00197
part-00198
part-00199
```



```
CREATE TABLE granular_access_log (
    ip STRING,
    ...
)
PARTITIONED BY (request_date STRING)
CLUSTERED BY (user_id)
    INTO 200 BUCKETS
...;
```

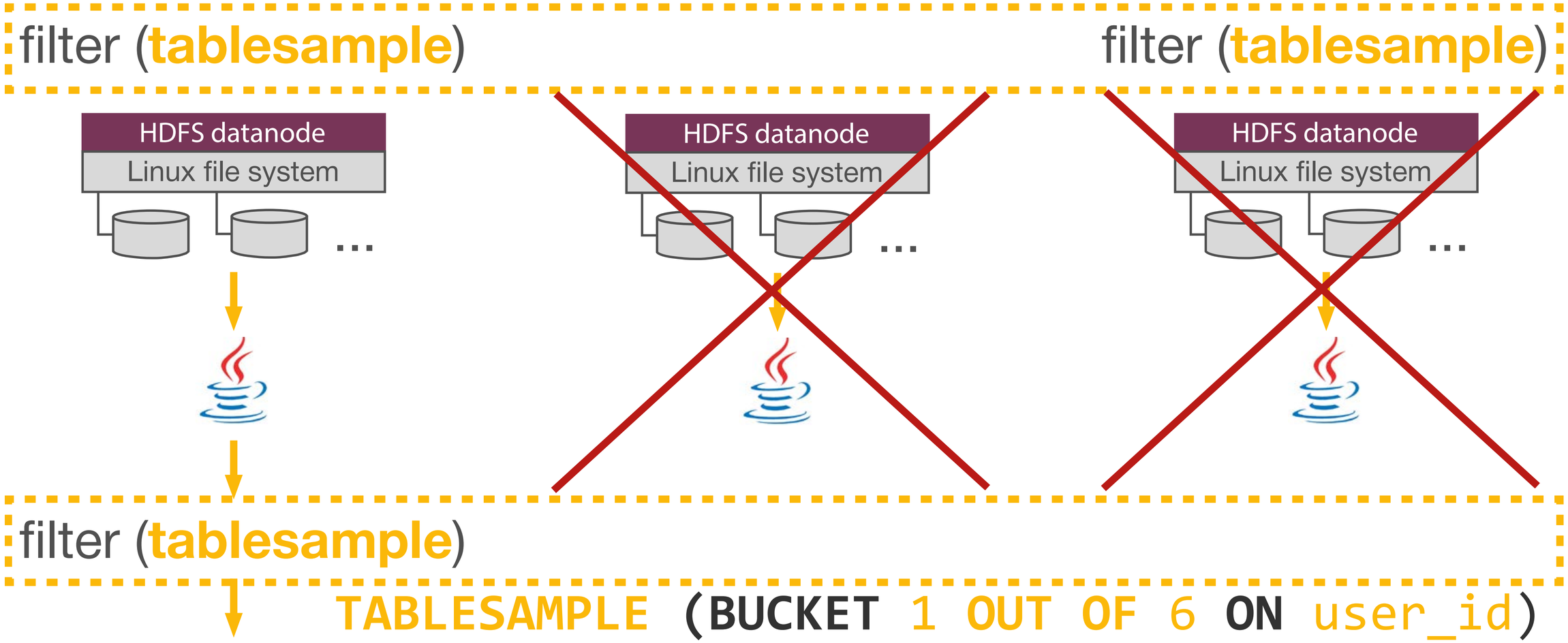
```
$ hdfs dfs -ls /path/to/access_logs
part-00000
part-00001
...
part-00197
part-00198
part-00199
```

```
CREATE TABLE granular_access_log (
    ip STRING,
    ...
)
PARTITIONED BY (request_date STRING)
CLUSTERED BY (user_id)
    INTO 200 BUCKETS
...;
```

see: https://en.wikipedia.org/wiki/Sampling_(statistics)

25%

```
SELECT ip, ...
FROM granular_access_log
    TABLESAMPLE (BUCKET 1 OUT OF 4 ON user_id)
...
```


filter (tablesample)
filter (tablesample)
HDFS datanode
Linux file system
...
HDFS datanode
Linux file system
...
HDFS datanode
Linux file system
...
filter (tablesample)
TABLESAMPLE (BUCKET 1 OUT OF 6 ON user_id)

```
FROM raw_access_log
INSERT OVERWRITE TABLE granular_access_log
PARTITION BY (request_date)
SELECT ..., request_date
WHERE ...
;
```

```
SET mapred.reduce.tasks = 200;

FROM raw_access_log
INSERT OVERWRITE TABLE granular_access_log
PARTITION BY (request_date)
SELECT ..., request_date
WHERE ...
;
```

```
SET mapred.reduce.tasks = 200;

FROM raw_access_log
INSERT OVERWRITE TABLE granular_access_log
PARTITION BY (request_date)
SELECT ..., request_date
WHERE ...
DISTRIBUTE BY user_id
;
```

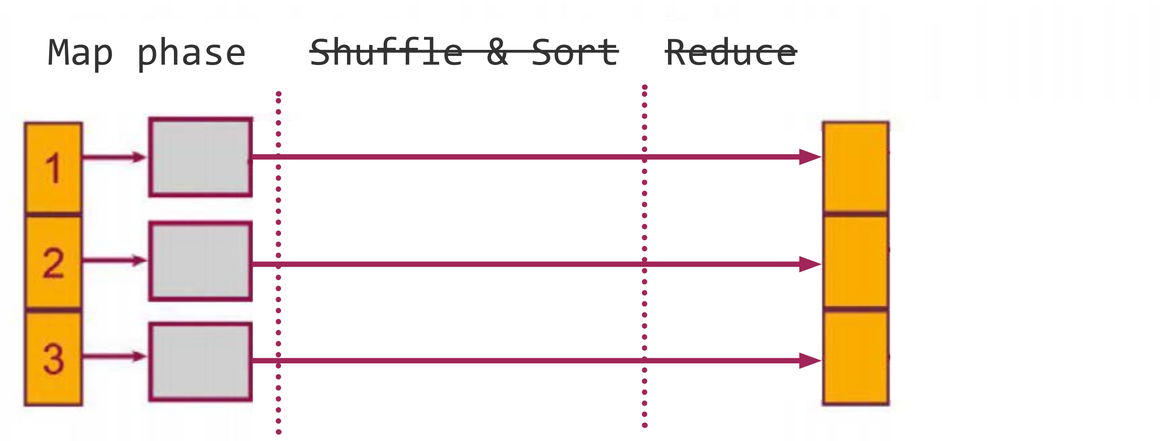
```
SET mapred.reduce.tasks = 200;

FROM raw_access_log
INSERT OVERWRITE TABLE granular_access_log
PARTITION BY (request_date)
SELECT ..., request_date
WHERE ...
DISTRIBUTE BY user_id
[SORT BY user_id]
;
```

```
SET mapred.reduce.tasks = 200;

FROM raw_access_log
INSERT OVERWRITE TABLE granular_access_log
PARTITION BY (request_date)
SELECT ..., request_date
WHERE ...
DISTRIBUTE BY user_id
[SORT BY user_id]
;
```

```
SET hive.enforce.bucketing=true;

FROM raw_access_log
INSERT OVERWRITE TABLE granular_access_log
PARTITION BY (request_date)
SELECT ..., request_date
WHERE ...;
```

```sql
CREATE TABLE granular_access_log (
    ip STRING,
    ...
)
PARTITIONED BY (request_date STRING)
CLUSTERED BY (user_id)
    SORTED BY (user_id)
    INTO 200 BUCKETS
...;
```

```
CREATE TABLE granular_access_log (
    ip STRING,
    ...
)
PARTITIONED BY (request_date STRING)
CLUSTERED BY (user_id)
    SORTED BY (user_id)
    INTO 200 BUCKETS
...;
```




```
SELECT user_id, COUNT(1)
FROM granular_access_log
GROUP BY user_id;
```

Summary

Summary

- You can **optimise** data processing with the help of **partitioning**

Summary

- You can **optimise** data processing with the help of **partitioning**
- You can **organise** data in Hive warehouse to **efficiently execute** sampling queries (see: **bucketing**)

Summary

- You can **optimise** data processing with the help of **partitioning**
- You can **organise** data in Hive warehouse to **efficiently execute** sampling queries (see: **bucketing**)
- You can **read from / write into** Hive tables taking into account partitions, buckets and **sorting**

Summary

- You can **optimise** data processing with the help of **partitioning**
- You can **organise** data in Hive warehouse to **efficiently execute** sampling queries (see: **bucketing**)
- You can **read from / write into** Hive tables taking into account partitions, buckets and **sorting**

see: https://cwiki.apache.org/confluence/display/Hive/LanguageManual+Sampling
see: https://cwiki.apache.org/confluence/display/Hive/LanguageManual+DDL+BucketedTables