
facebook

Cosco: an efficient facebook-scale shuffle service

Brian Cho & Dmitry Borovsky

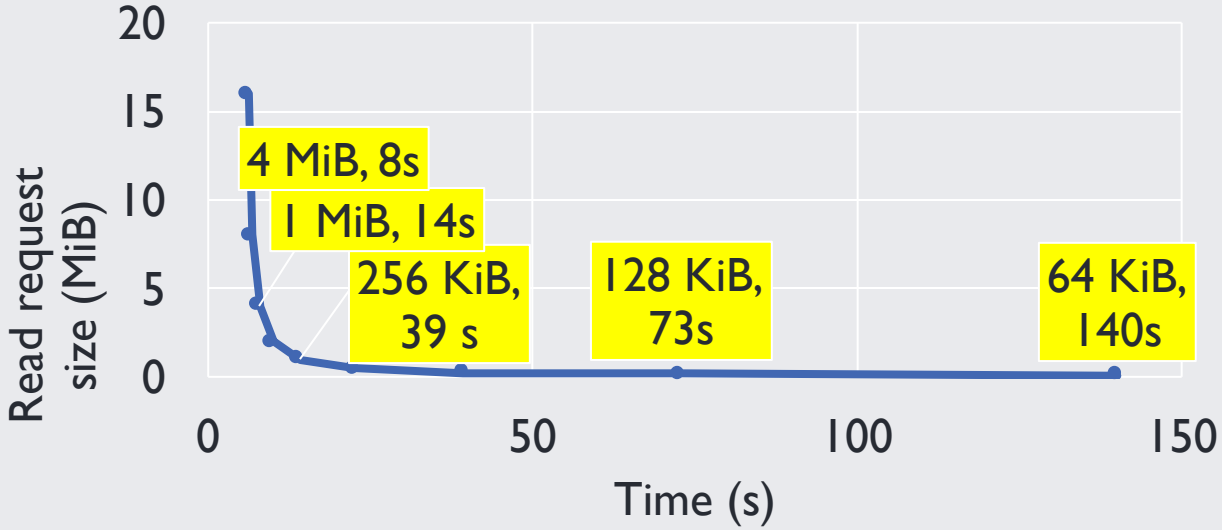
Spark + AI Summit 2019

Disaggregated compute and storage

- Advantages
  - Server types optimized for compute or storage
  - Separate capacity management and configuration
  - Different hardware cycles
- Compute clusters
  - CPU, RAM, no disks for data
  - Spark executors
- Storage clusters
  - Spindle disks
  - DFS (Warm Storage)
    - Permanent data: size dominant, uses less IO
    - Temporary data: IO dominant, uses less space

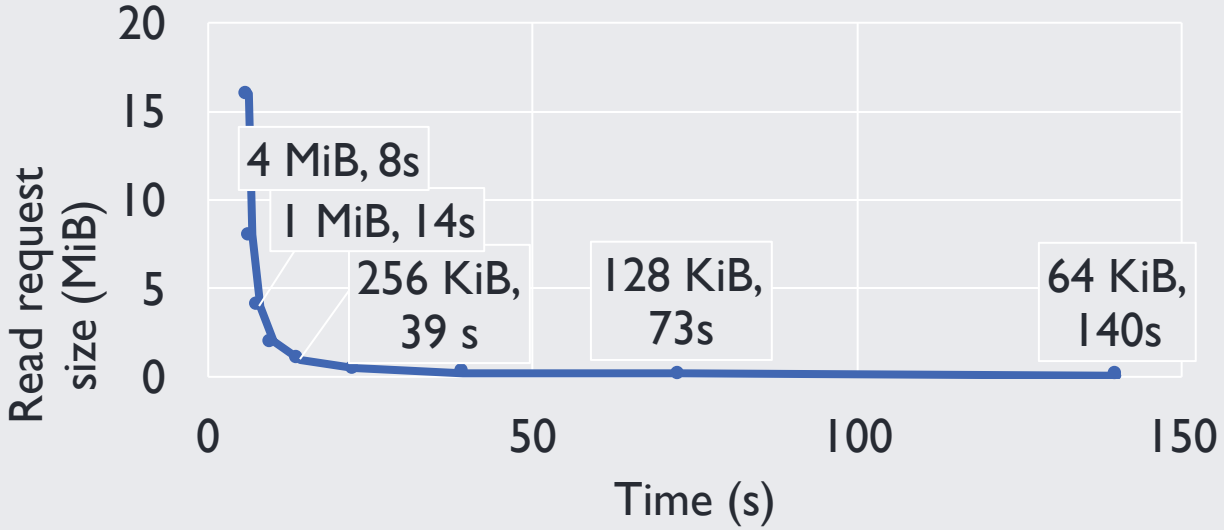
Spindle disk storage

- A single spindle is used to read/write data on the drive
  - Small IO sizes cause low throughput as seek times dominate

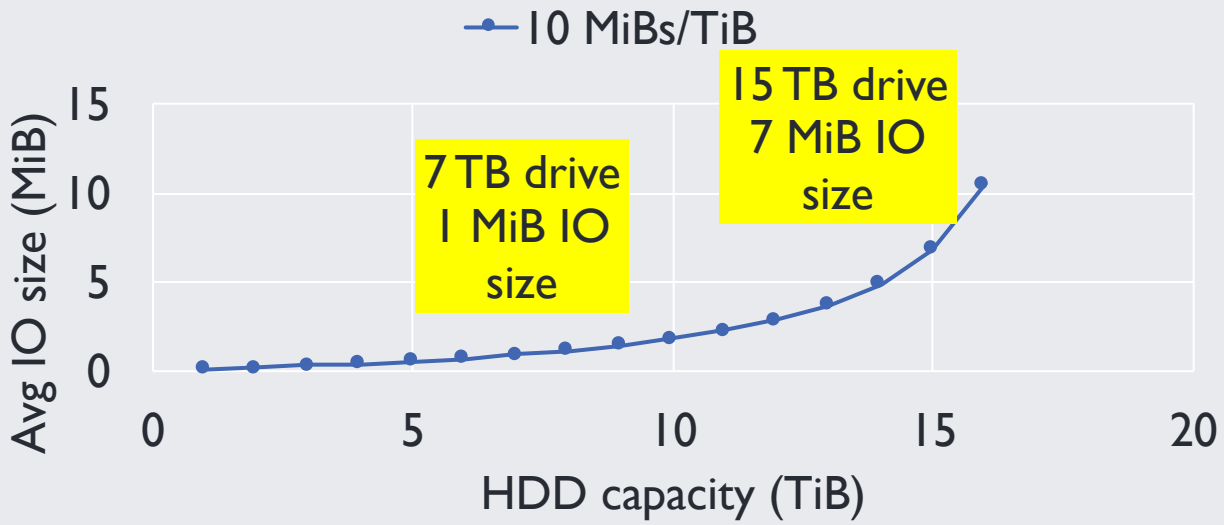


Spindle disk storage

- A single spindle is used to read/write data on the drive
  - Small IO sizes cause low throughput as seek times dominate

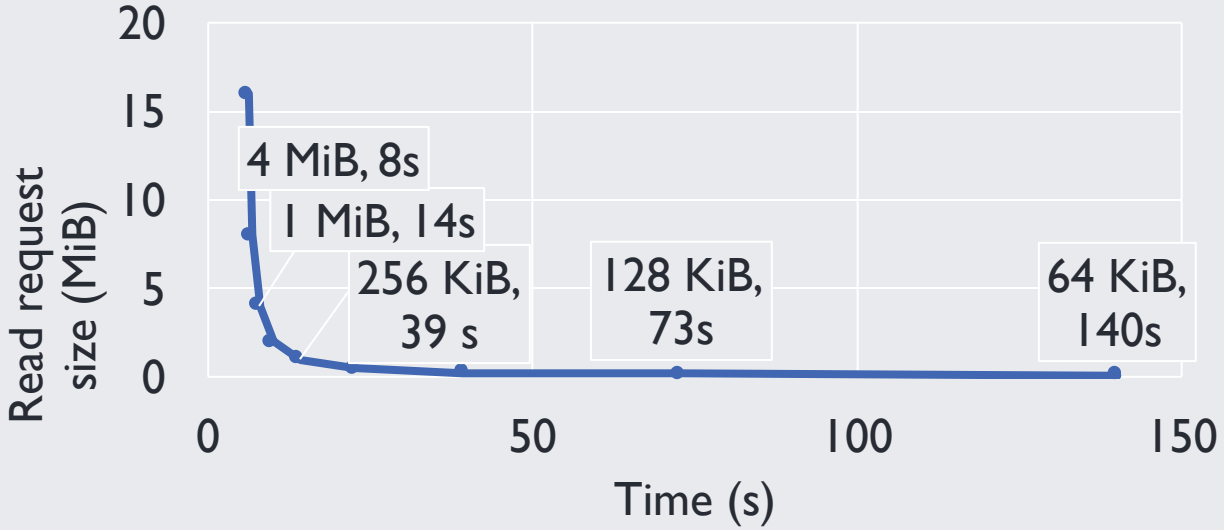


- Drive sizes increase over time
  - Must increase IO size to maintain the same throughput per TB

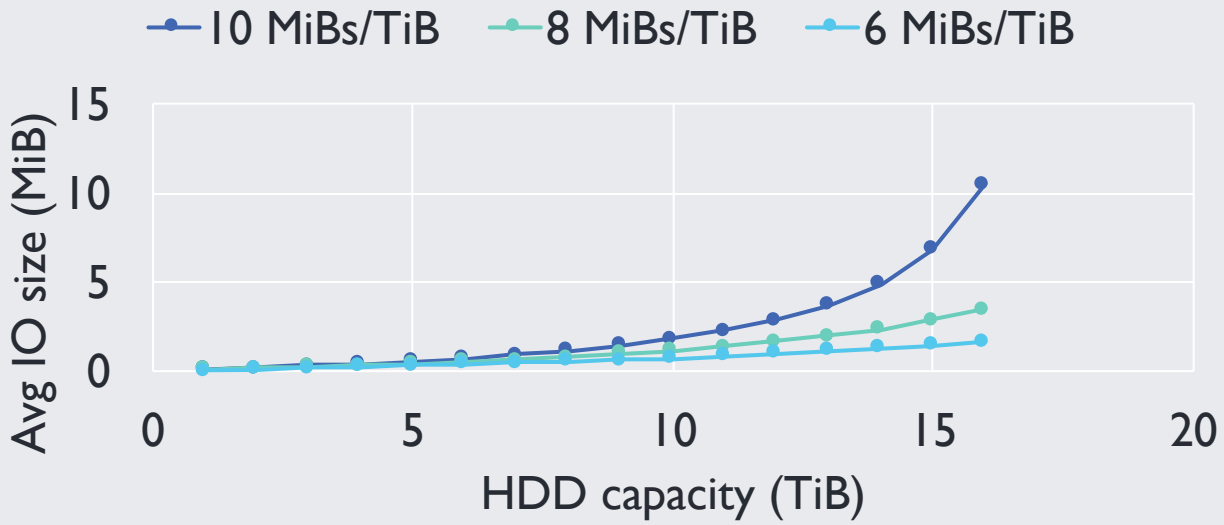


Spindle disk storage

- A single spindle is used to read/write data on the drive
  - Small IO sizes cause low throughput as seek times dominate



- Drive sizes increase over time
  - Must increase IO size to maintain the same throughput per TB, or
  - Read/write less data to reduce throughput demand



Spindle disk storage: key metrics

- A single spindle is used to read/write data on the drive
  - Small IO sizes cause low throughput as seek times dominate

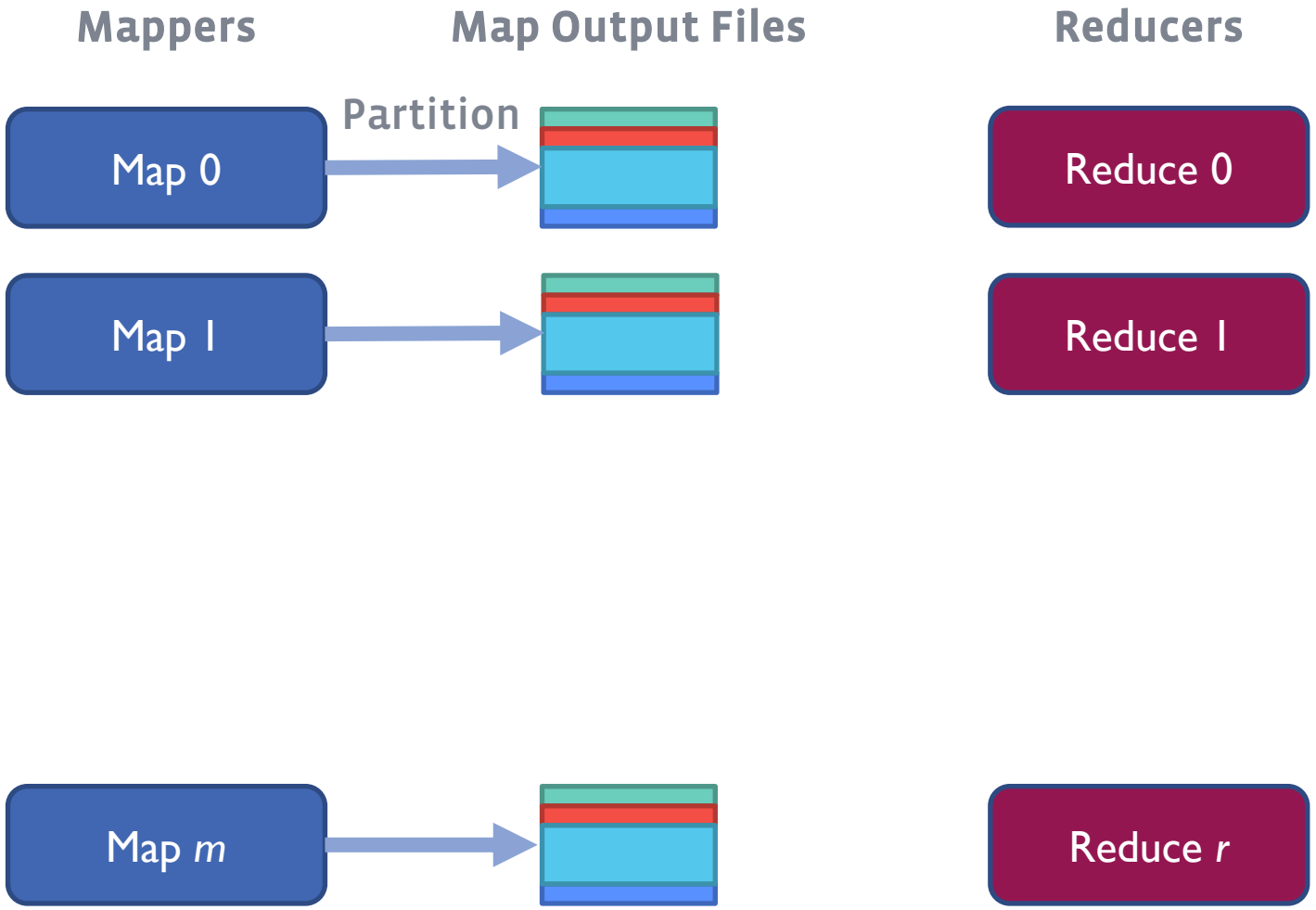
➢Disk service time

➢Average IO size

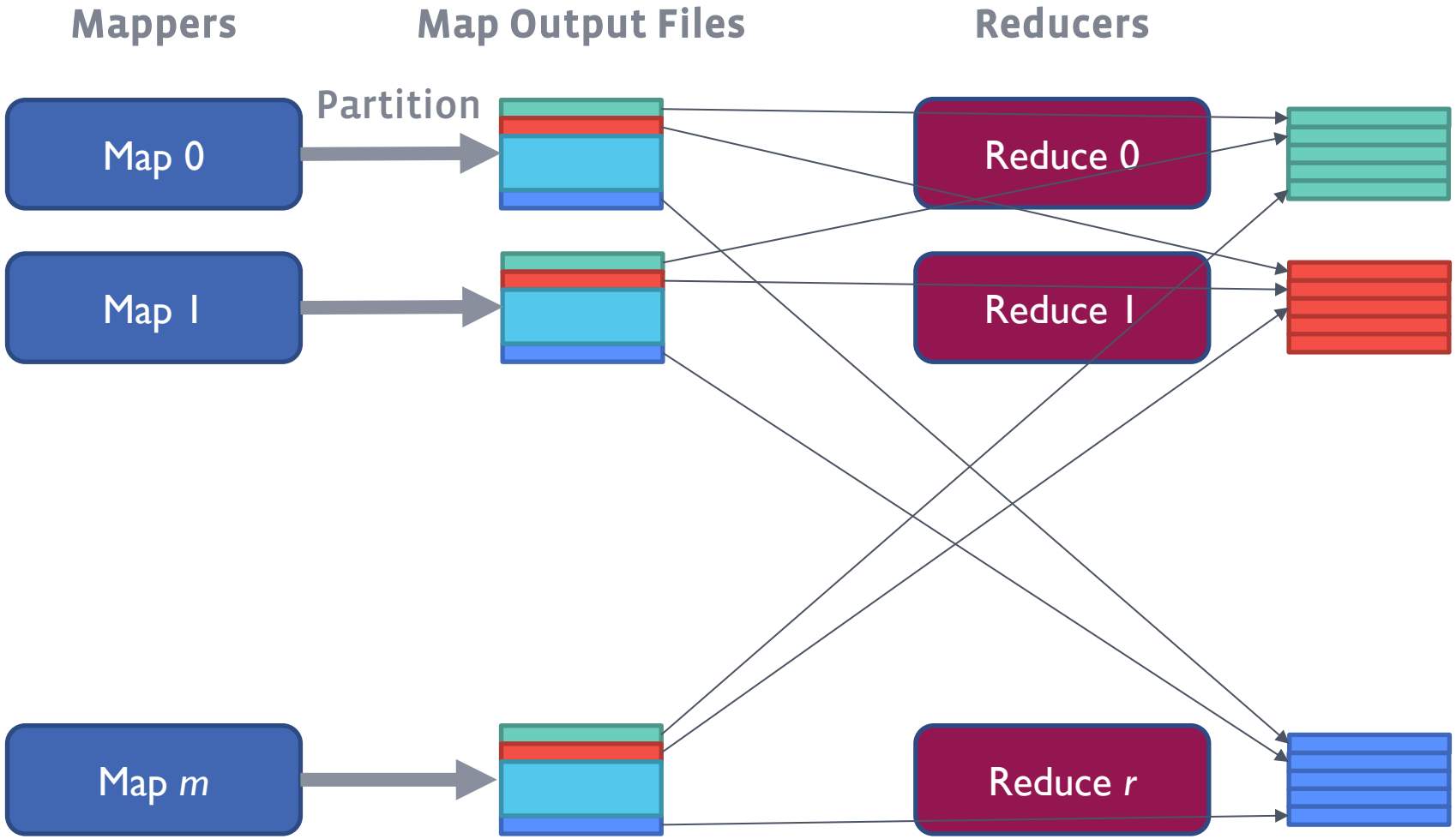
- Drive sizes increase over time
  - Must increase IO size to maintain the same throughput, or
  - Read/write less data to reduce throughput demand

➢Write amplification

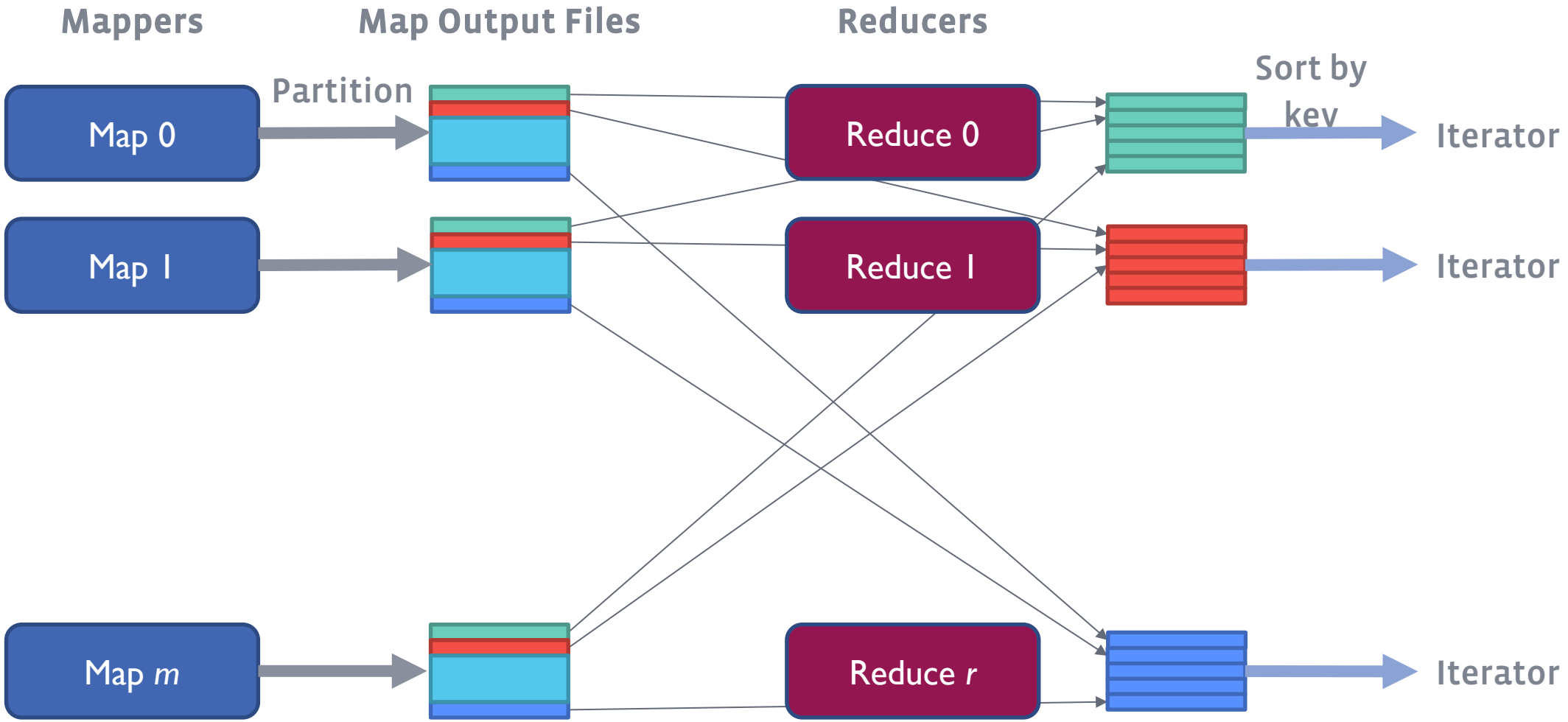
Spark shuffle recap



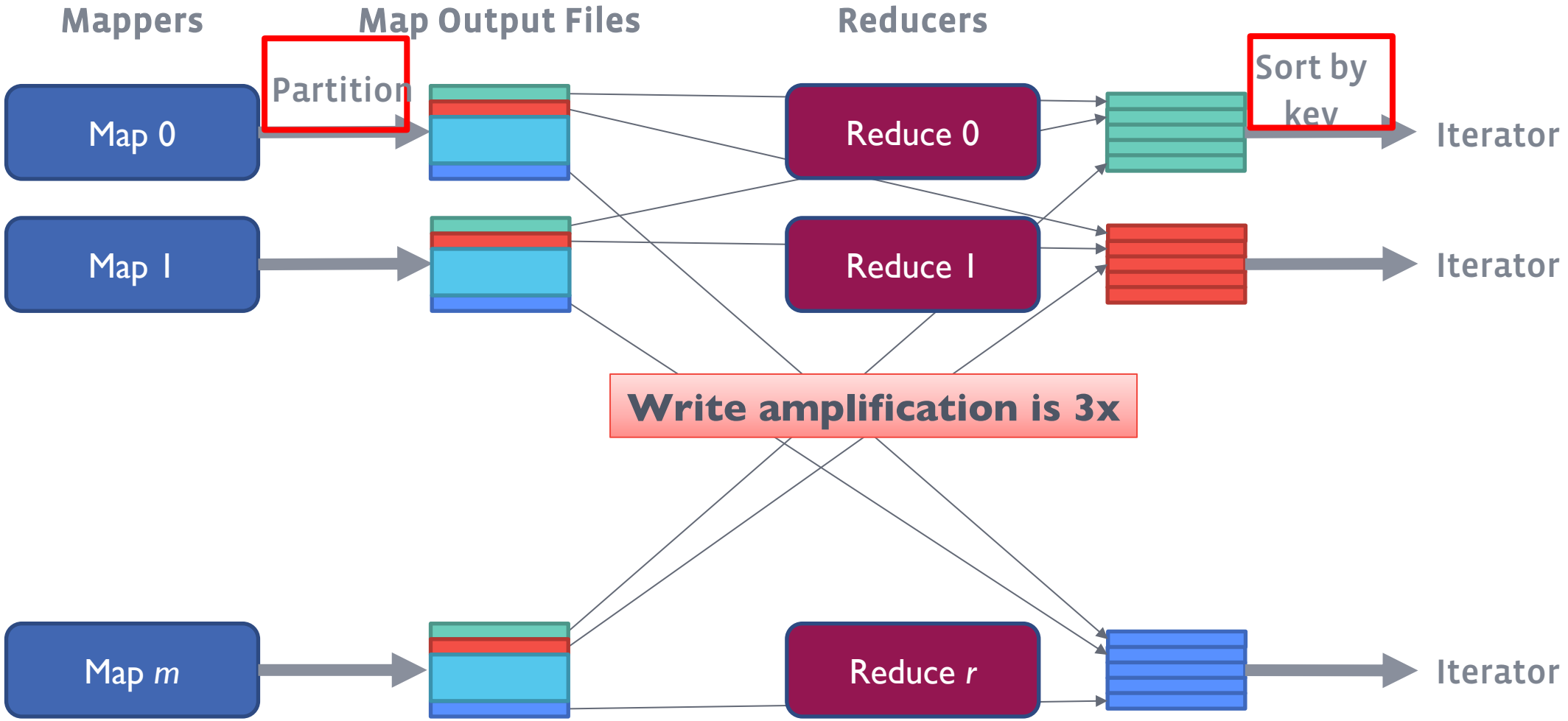
Spark shuffle recap



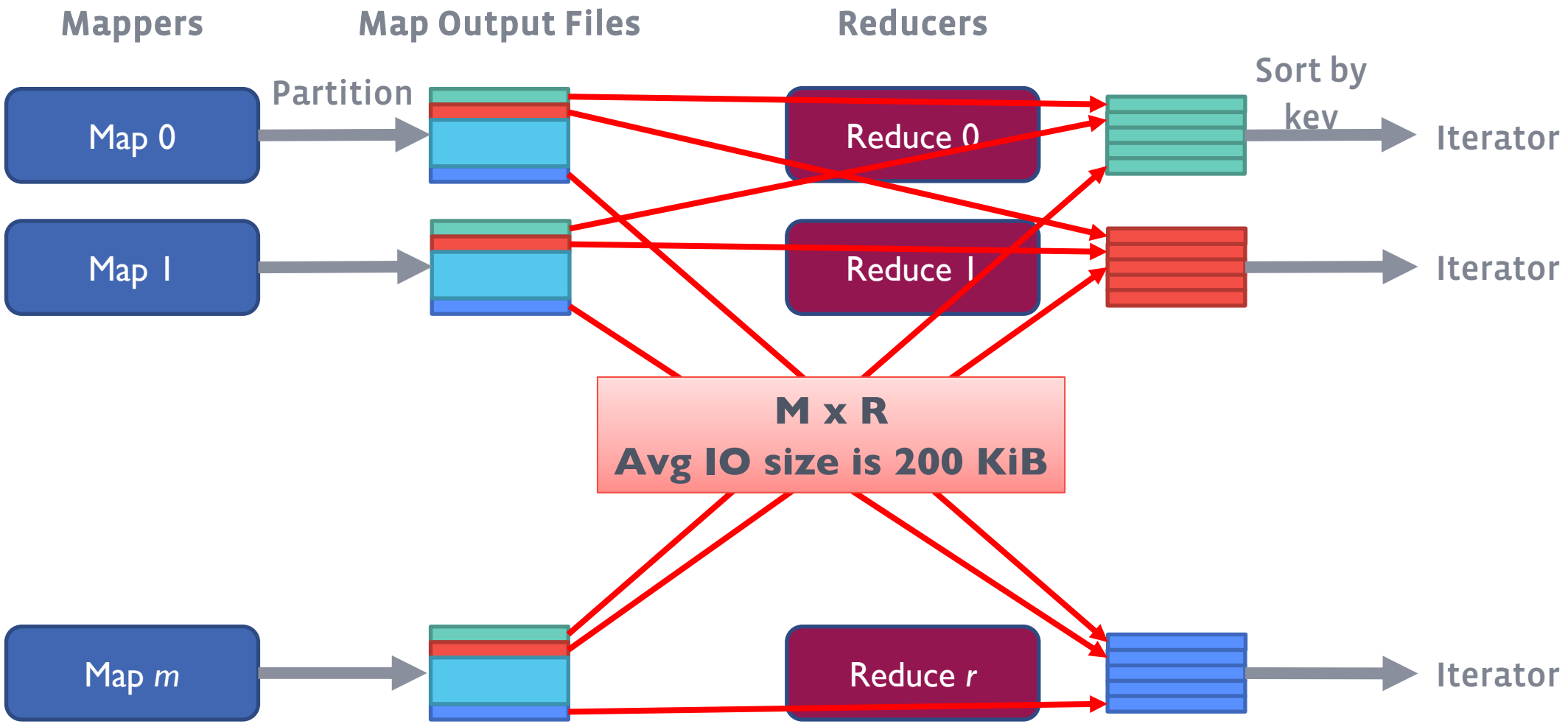
Spark shuffle recap



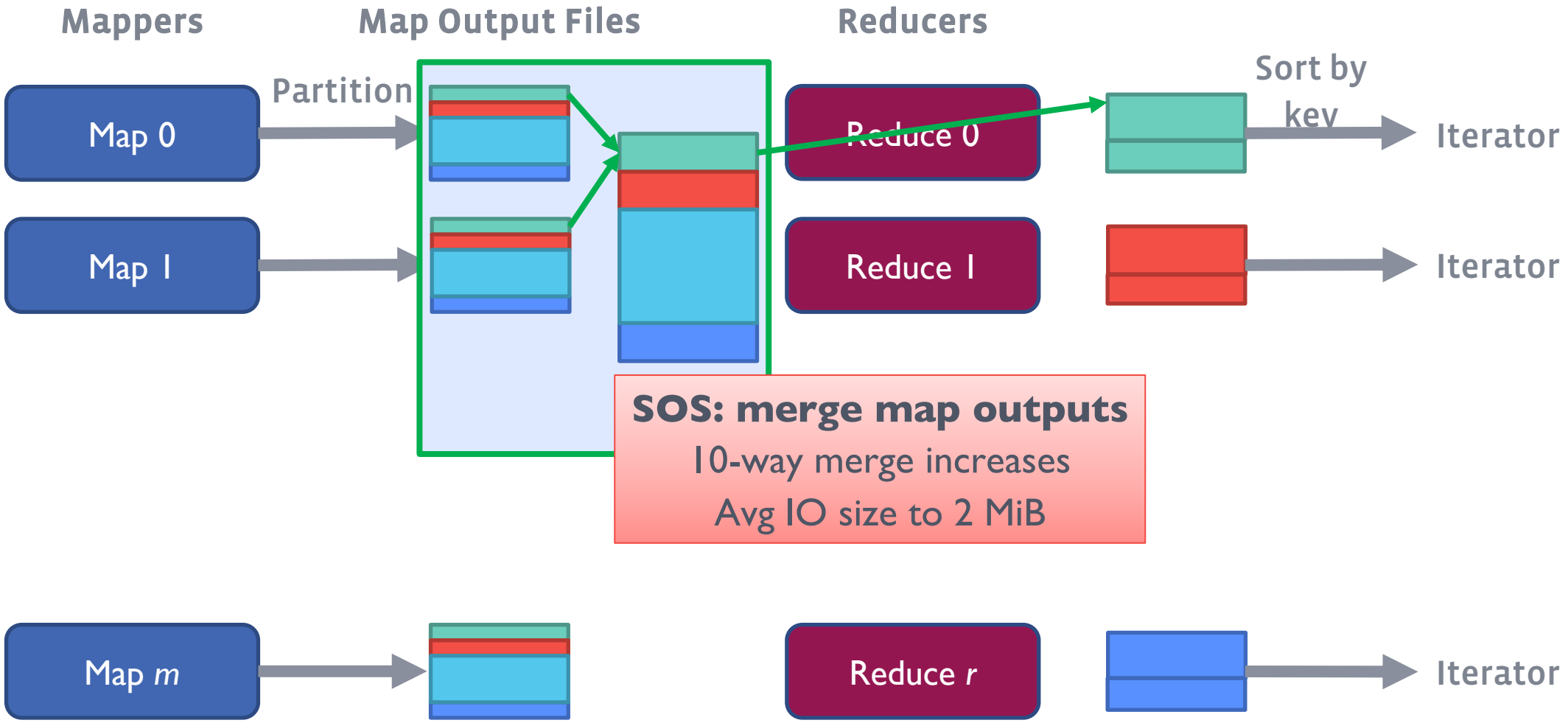
Spark shuffle recap: Write amplification



Spark shuffle recap: Small IOs problem



Spark shuffle recap: SOS



Spark shuffle using Cosco

- Mappers share a write-ahead buffer per reduce partition
- Reducers can read the written data sequentially

- Solves the small IOs problem
  - Sequential reads: Avg IO size 200 KiB → 2.5 MiB
- Solves the write amplification problem
  - Avoiding spills: Write amplification 3x → 1.2x

Results / Current status

- Hive
  - Rolled out to 90%+ of Hive workloads, in production for 1+ year
  - **3.2x** more efficient disk service time
- Spark
  - Analysis shows potential **3.5x** more efficient disk service time
  - Analysis shows CPU neutral
  - Integration is complete, rollout planned during next few months

Cosco deep dive

Dmitry Borovsky

Spark + AI Summit 2019

Problem

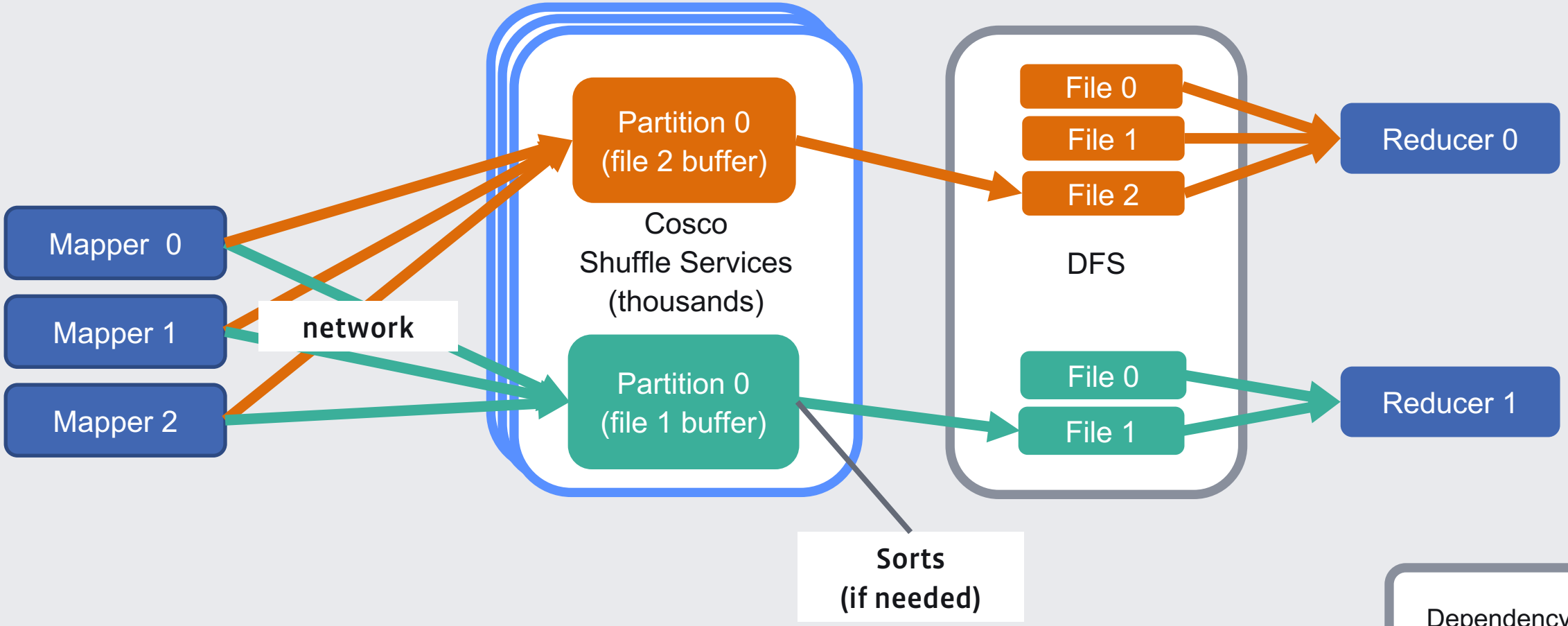
- Shuffle exchange on spinning disks (disaggregated compute and storage)
- Single shuffle exchange scale: PiBs size, 100Ks of mappers, 10Ks reducers
- Write amplification is ~3x (1PiB shuffle does 3PiB writes to disk)
- Small Average IO size: ~200KiB (at least MxR reads)
- IO is spiky (all readers may start at the same time and do MxR reads)
- Cosco is shared between users

Sailfish: a framework for large scale data processing

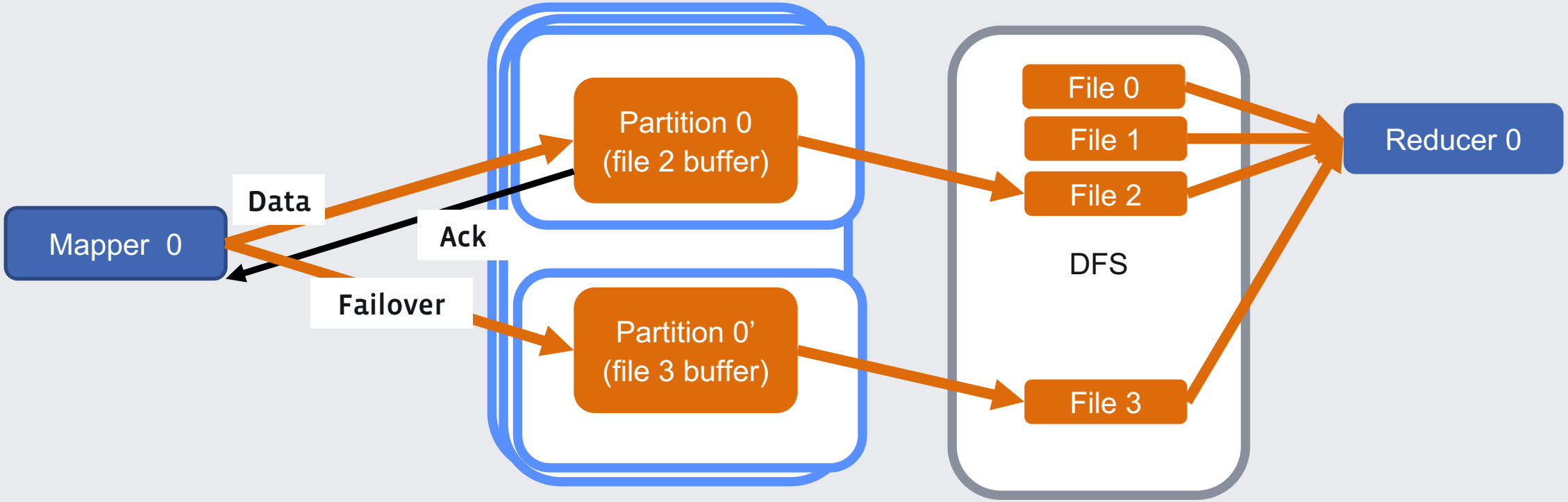
SoCC '12 Proceedings of the Third ACM Symposium on Cloud Computing, Article No. 4, San Jose, California — October 14 - 17, 2012

Source code: https://code.google.com/archive/p/sailfish/

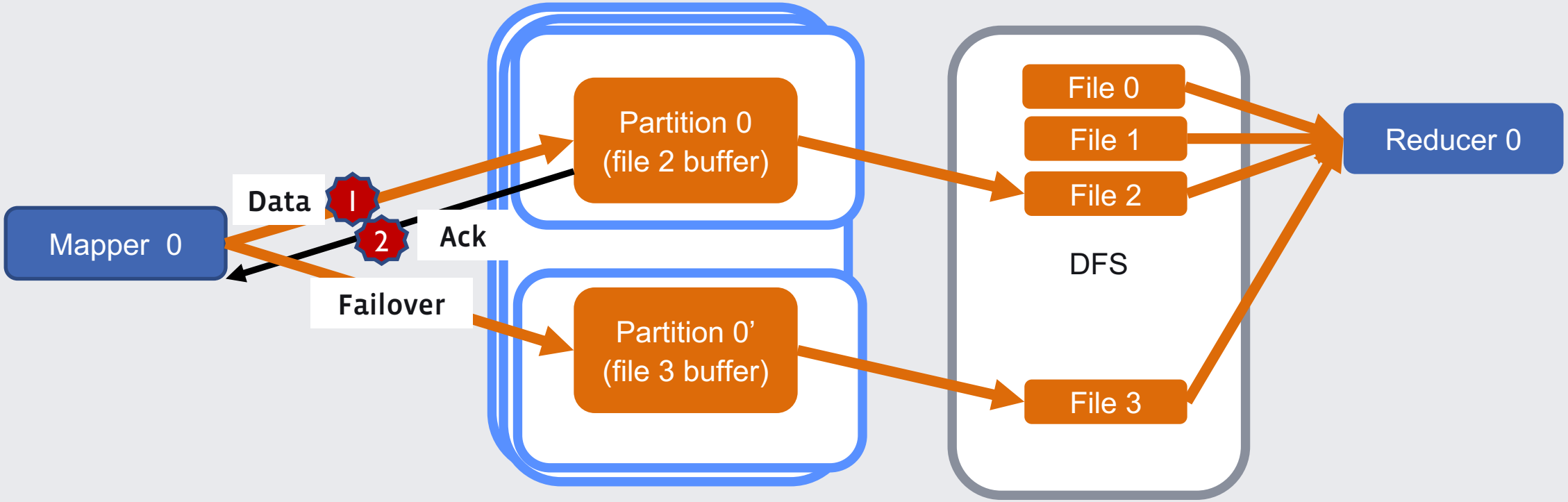
Write-ahead buffers



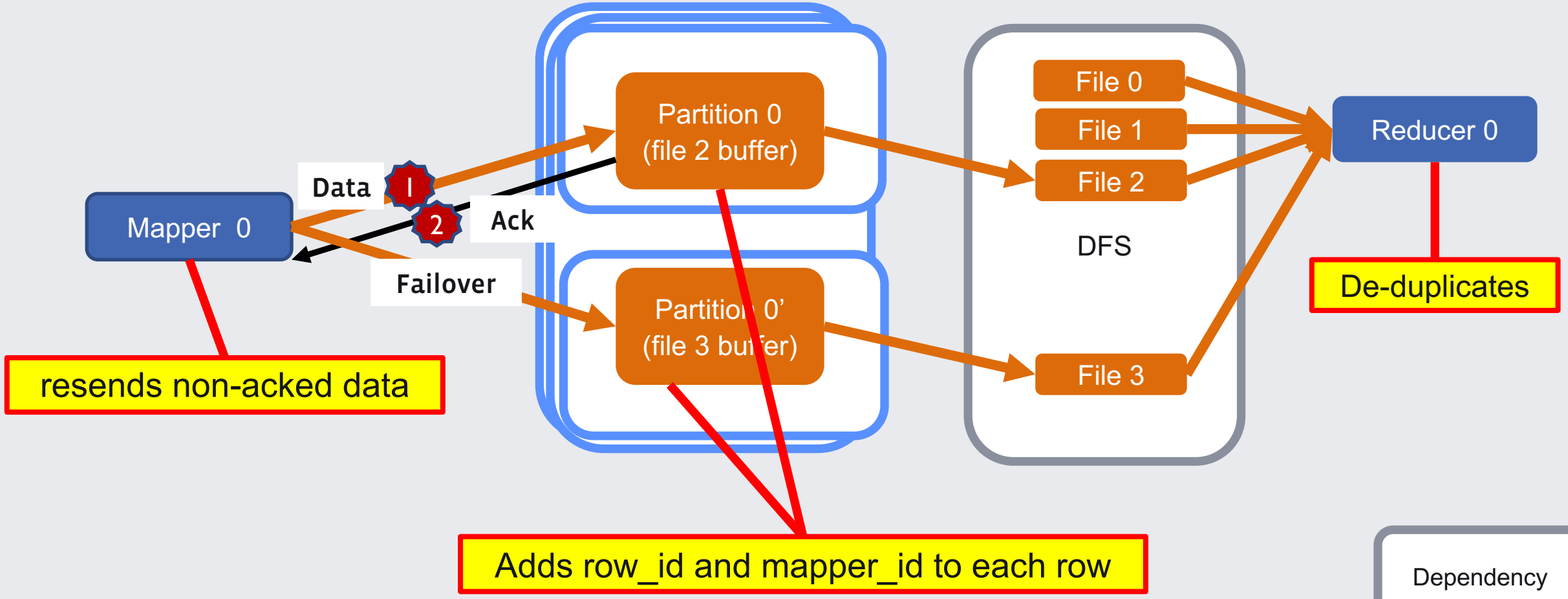
Exactly once delivery



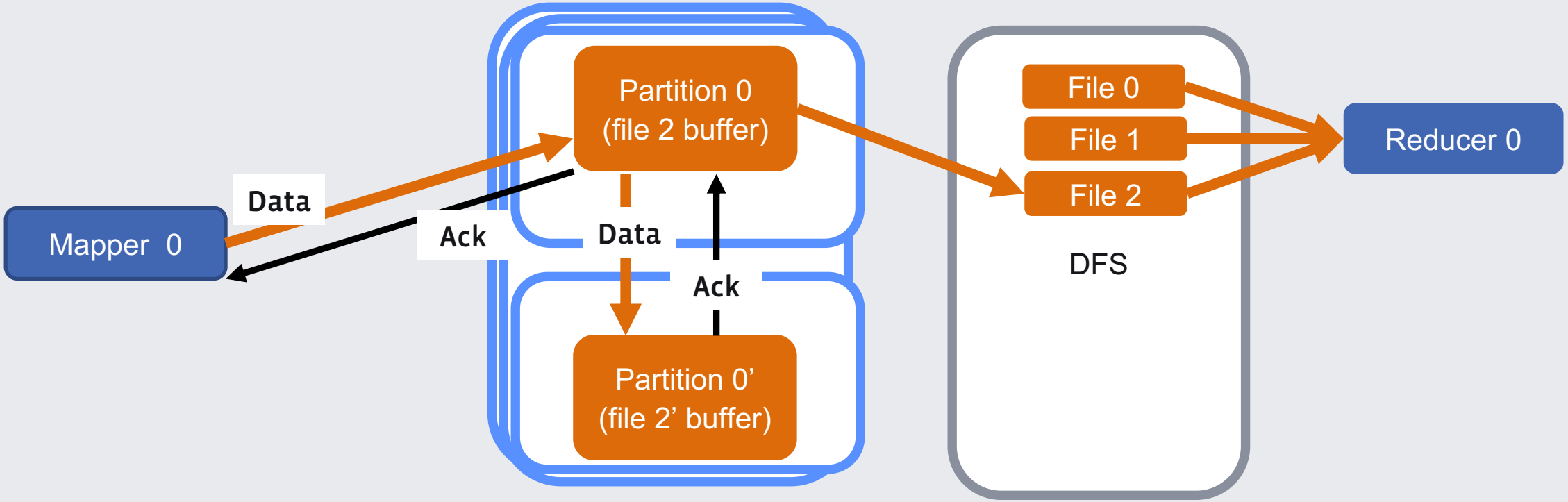
Exactly once delivery



At least once delivery and deduplication



Replication

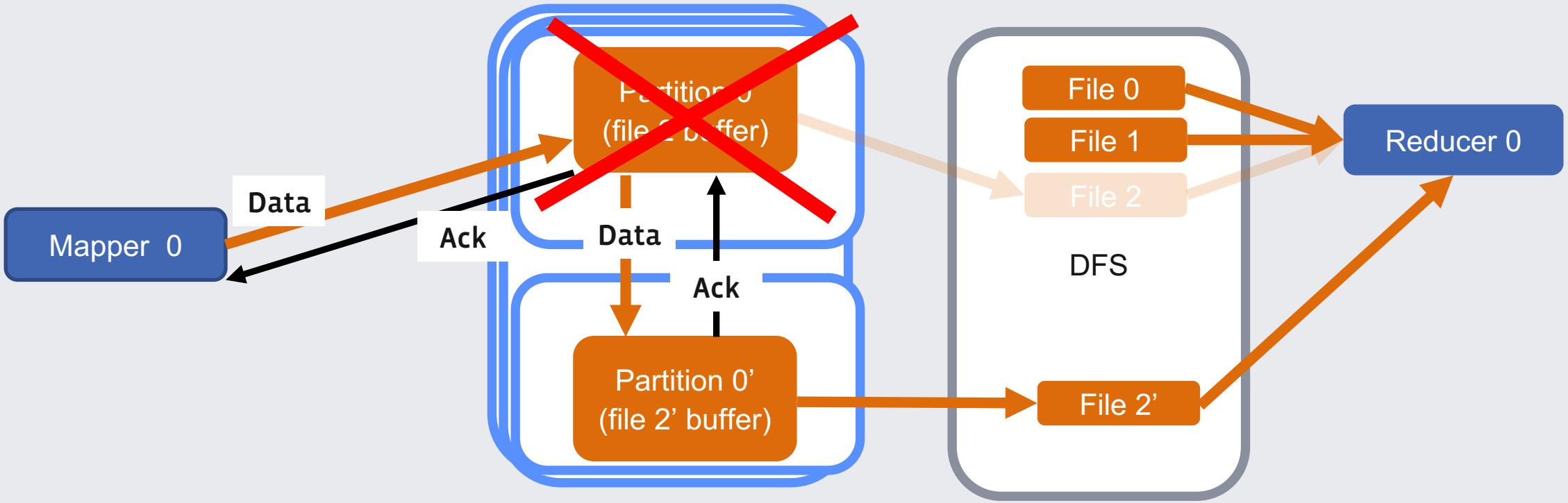


Dependency

Cosco process
(shared between apps)

Spark process
(cosco client)

Replication



```
// Driver
shuffle = new CoscoExchange(
  shuffleId: String,
  partitions: int,
  recomputeWritersCallback: (mappers: long[], reason: String) -> void);
// end of exchange
shuffle.close();
```

```
// Mapper
writer = new CoscoWriter(
  shuffleId: String,
  mapper: long);

writer.collect(
  partition: int, row: byte[]);
// ...
writer.collect(
  partition: int, row: byte[]);

writer.close();
```

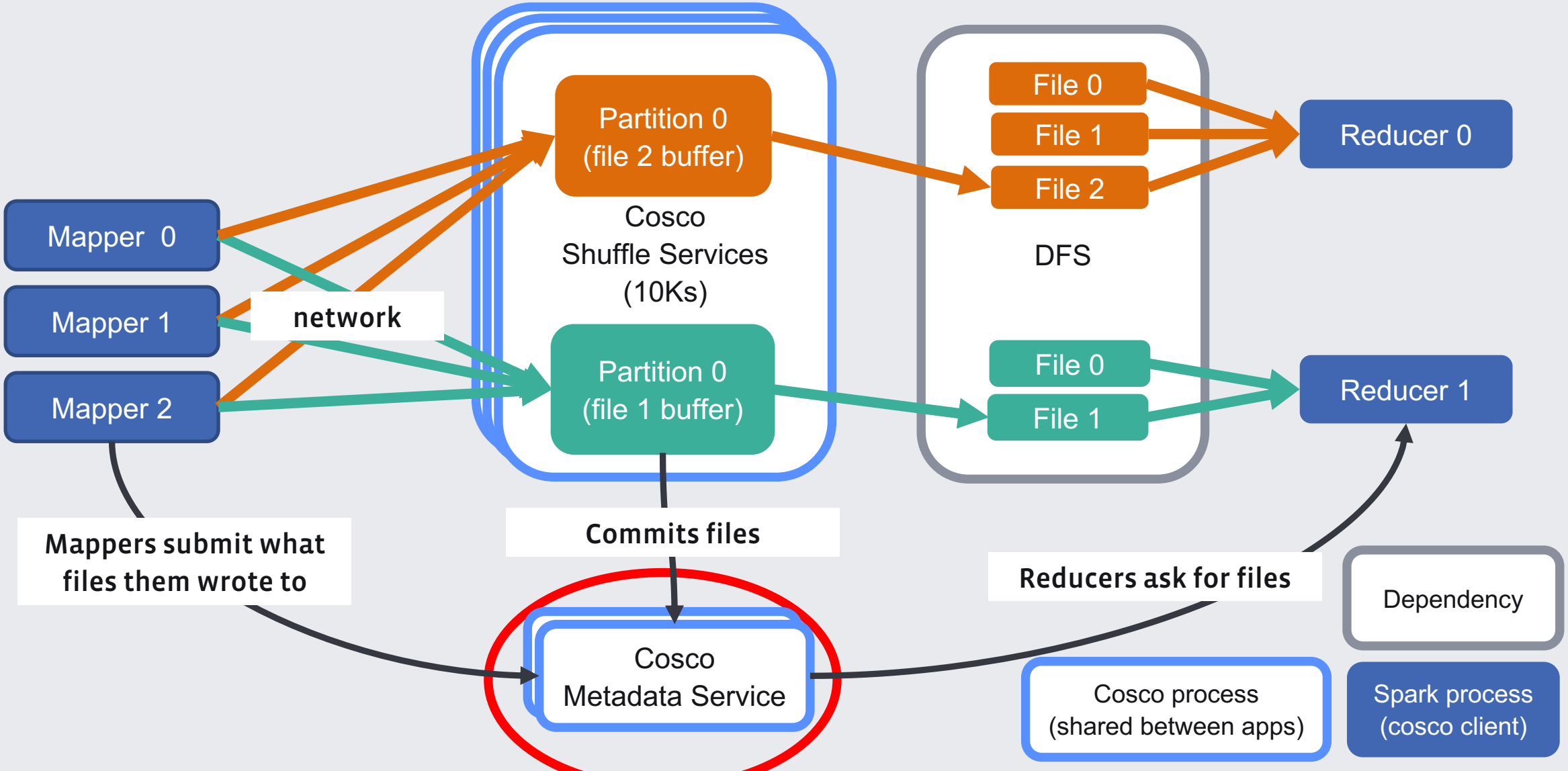
```
// Reducer
reader = new CoscoReader(
  shuffleId: String,
  mappers: long[],
  partition: int);

while (reader.next()) {
  // using row
  row = reader.row();
}

reader.close();
```

```
// Driver
shuffle = new CoscoExchange(
  shuffleId: String,
  partitions: int,
  recomputeWritersCallback: (mappers: long[], reason: String) -> void);
// end of exchange
shuffle.close();
```

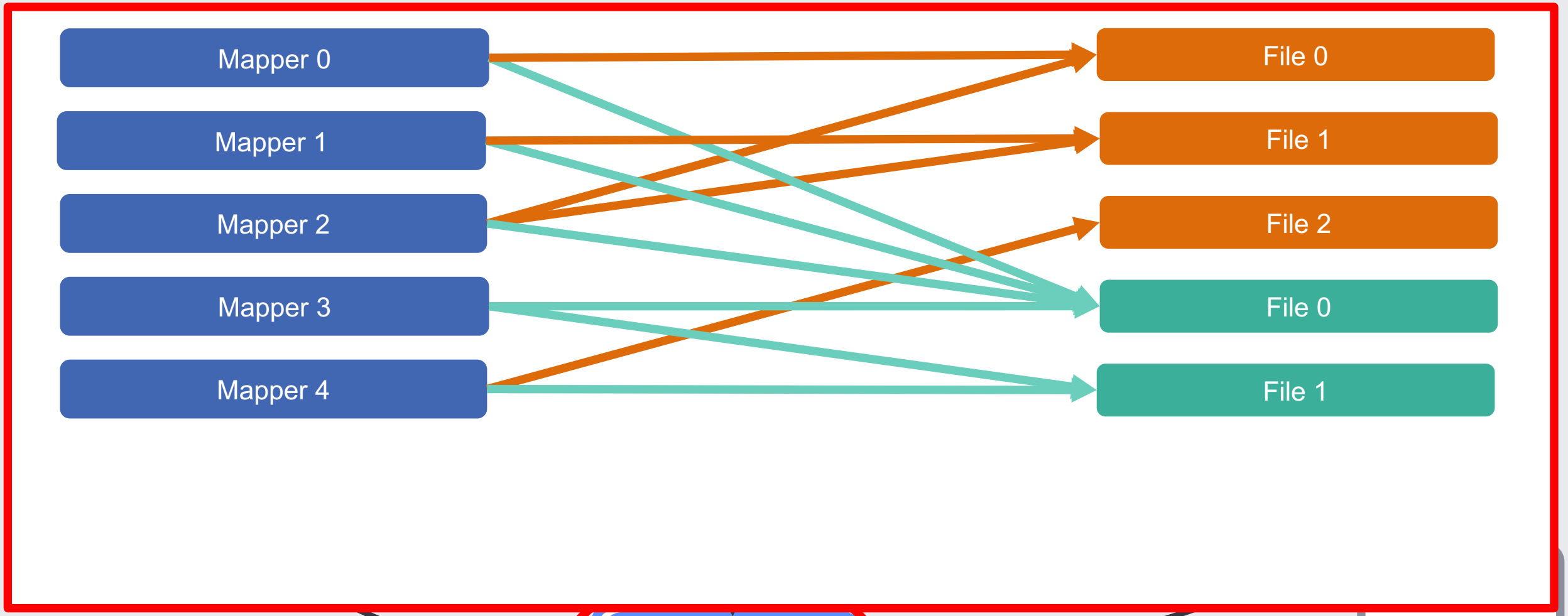
```
// Mapper
writer = new CoscoWriter(
  shuffleId: String,
  mapper: long);

writer.collect(
  partition: int, row: byte[]);
// ...
writer.collect(
  partition: int, row: byte[]);

writer.close();
```

```
// Reducer
reader = new CoscoReader(
  shuffleId: String,
  mappers: long[],
  partition: int);

while (reader.next()) {
  // using row
  row = reader.row();
}

reader.close();
```

Metadata


Partition 0
(file 2 buffer)
Cosco
Shuffle Services
(10Ks)
Partition 0
(file 1 buffer)
File 0
File 1
File 2
DFS
File 0
File 1
Reducer 0
Reducer 1
Mapper 0
Mapper 1
Mapper 2
network
Mappers submit what files them wrote to
Commits files
Reducers ask for files
Cosco
Metadata Service
Dependency
Cosco process
(shared between apps)
Spark process
(cosco client)

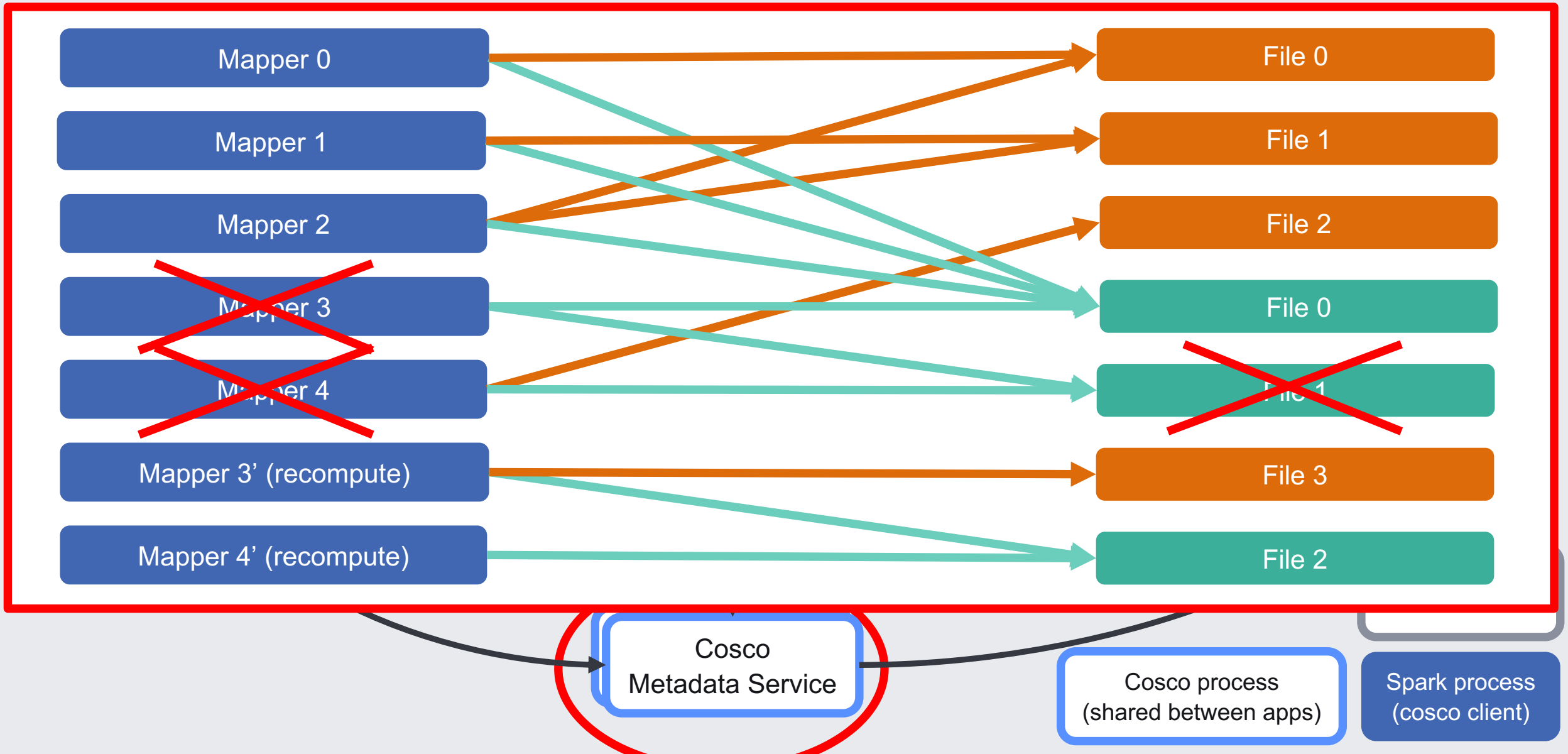
Metadata



Cosco process
(shared between apps)

Spark process
(cosco client)

Metadata

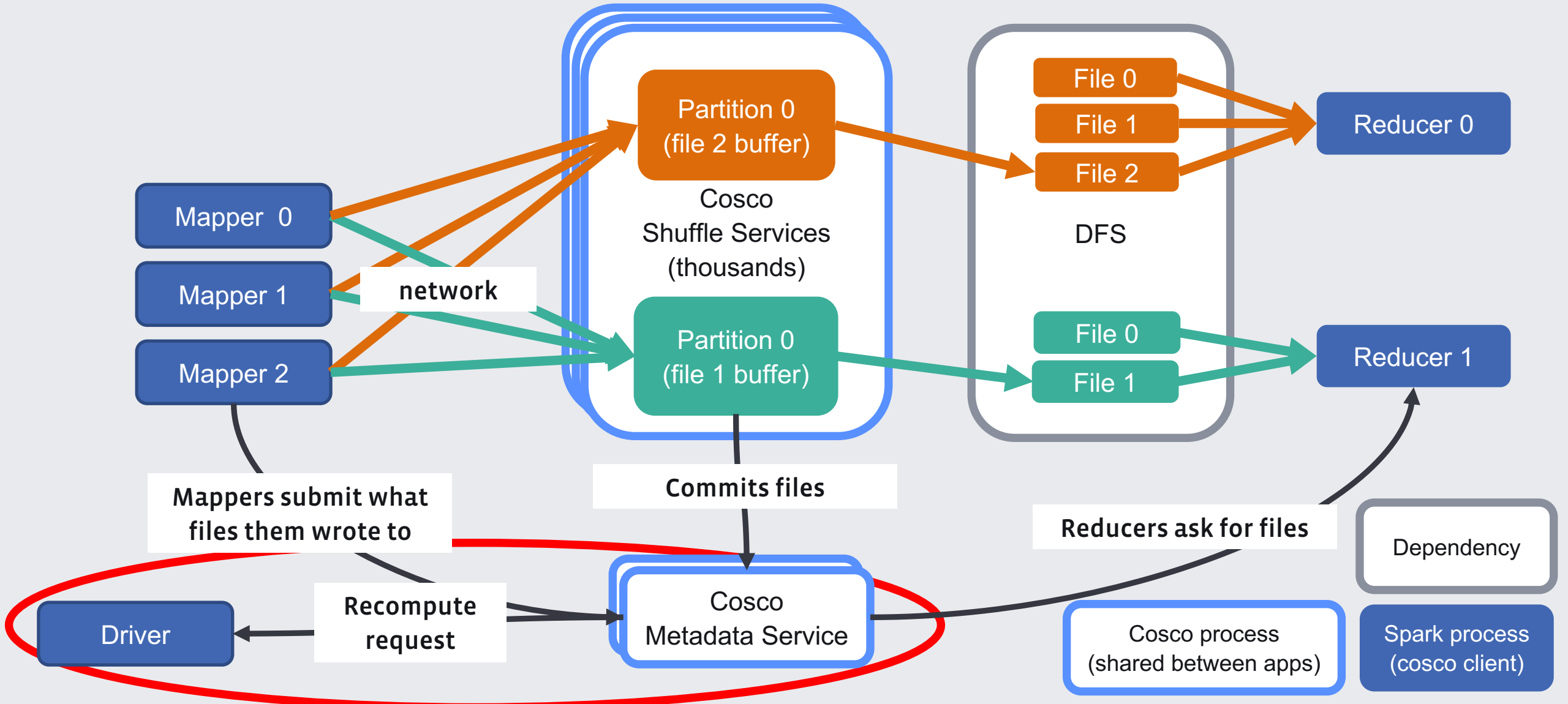

Mapper 0
Mapper 1
Mapper 2
Mapper 3
Mapper 4
Mapper 3' (recompute)
Mapper 4' (recompute)
File 0
File 1
File 2
File 0
File 1
File 3
File 2
Cosco
Metadata Service
Cosco process
(shared between apps)
Spark process
(cosco client)

```
// Driver
shuffle = new CoscoExchange(
  shuffleId: String,
  partitions: int,
  recomputeWritersCallback: (mappers: long[], reason: String) -> void);
// end of exchange
shuffle.close();
```

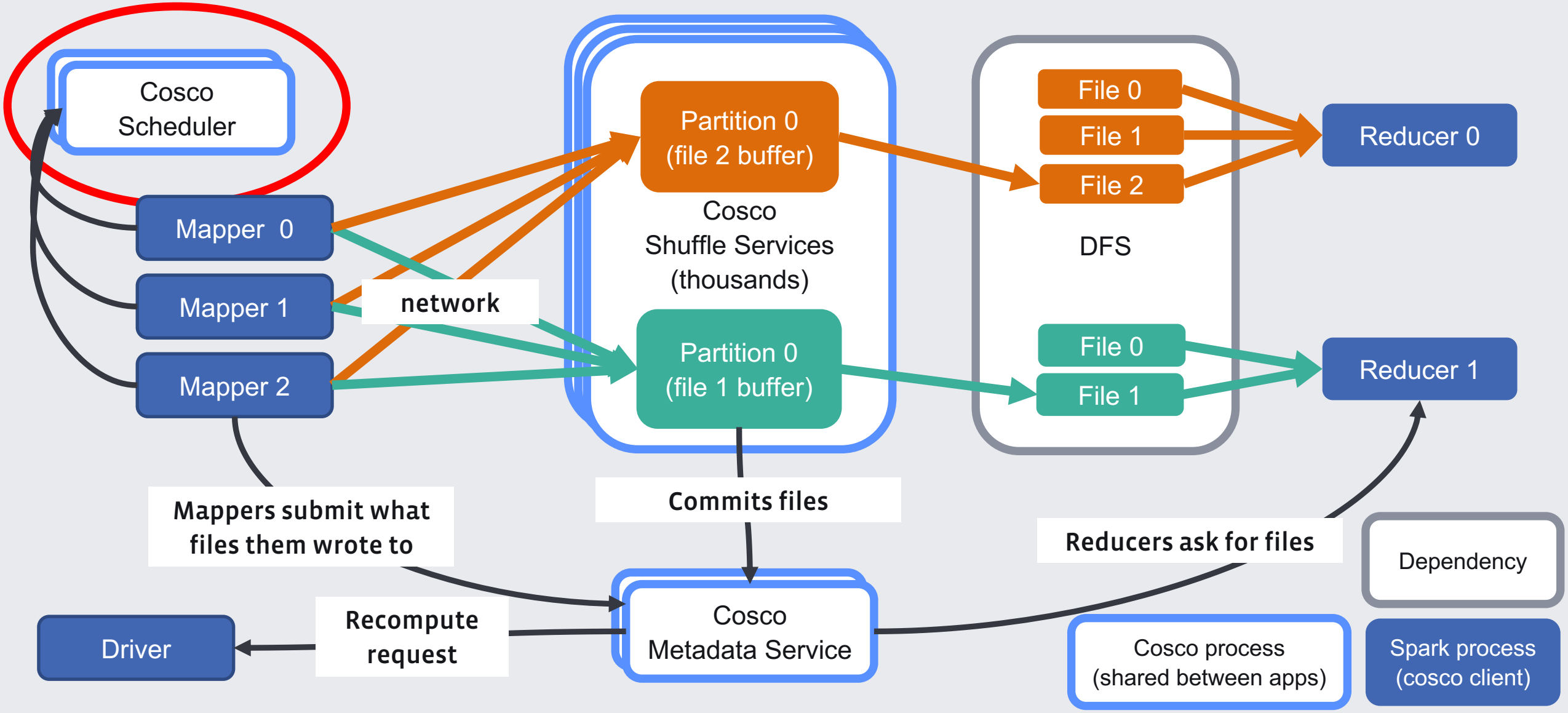
```
// Mapper
writer = new CoscoWriter(
  shuffleId: String,
  mapper: long);

writer.collect(
  partition: int, row: byte[]);
// ...
writer.collect(
  partition: int, row: byte[]);

writer.close();
```

```
// Reducer
reader = new CoscoReader(
  shuffleId: String,
  mappers: long[],
  partition: int);

while (reader.next()) {
  // using row
  row = reader.row();
}

reader.close();
```

Metadata


Partition 0
(file 2 buffer)
Cosco
Shuffle Services
(thousands)
Partition 0
(file 1 buffer)
File 0
File 1
File 2
DFS
File 0
File 1
Reducer 0
Reducer 1
Mapper 0
Mapper 1
Mapper 2
network
Mappers submit what files them wrote to
Commits files
Reducers ask for files
Recompute request
Driver
Cosco
Metadata Service
Dependency
Cosco process
(shared between apps)
Spark process
(cosco client)

Scheduler


Cosco
Scheduler
Mapper 0
Mapper 1
Mapper 2
network
Partition 0
(file 2 buffer)
Cosco
Shuffle Services
(thousands)
Partition 0
(file 1 buffer)
File 0
File 1
File 2
DFS
File 0
File 1
Reducer 0
Reducer 1
Mappers submit what
files them wrote to
Commits files
Reducers ask for files
Driver
Recompute
request
Cosco
Metadata Service
Dependency
Cosco process
(shared between apps)
Spark process
(cosco client)

Limits

- Cosco doesn’t support large rows (<4MiB)
- Capacity: shuffle services memory, number of write-ahead buffers

Future work

- “Unlimited” shuffle exchange:
  - millions of splits/partitions
  - 10s of PiBs
- Streaming

Questions?

Brian Cho (bcho@fb.com)
Dmitry Borovsky (borovsky@fb.com)