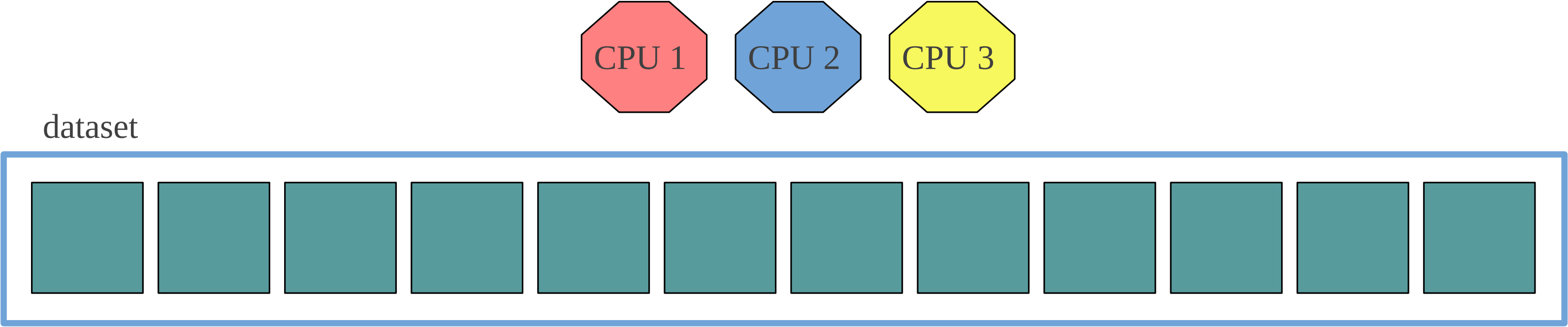


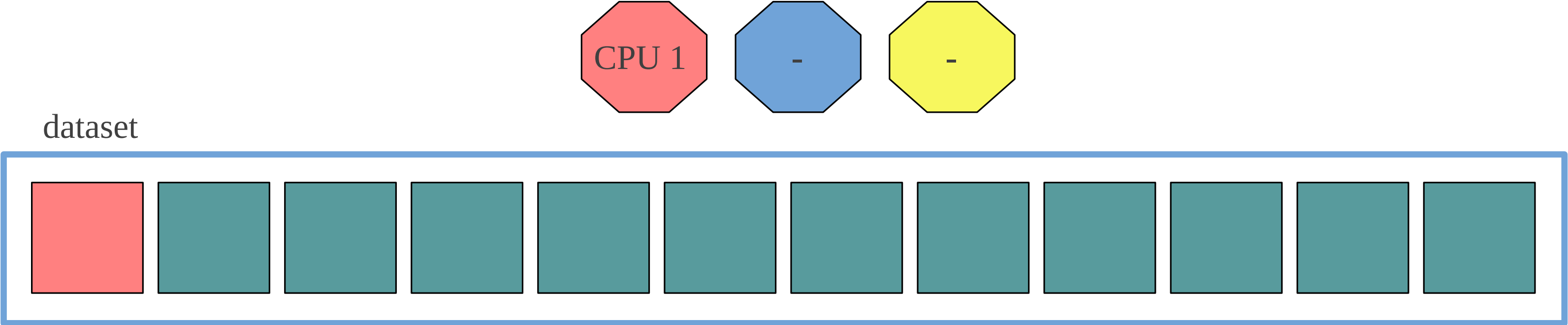


# ANALYSIS OF GLOBAL TEMPERATURE

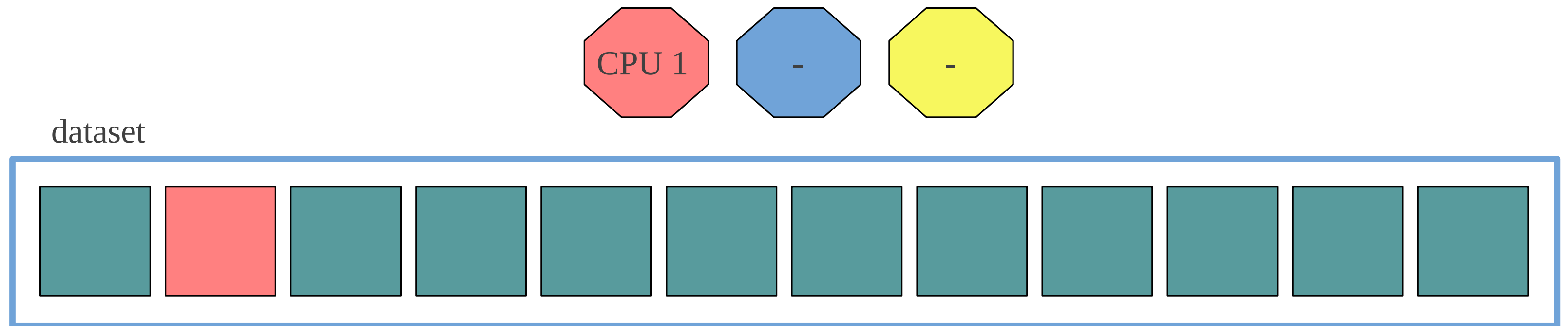
# Data processing



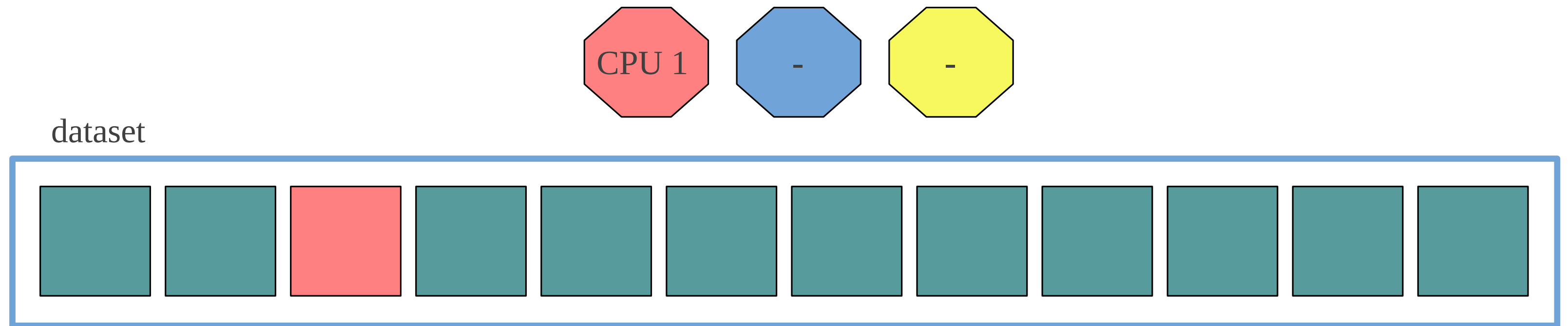
# Sequential iteration



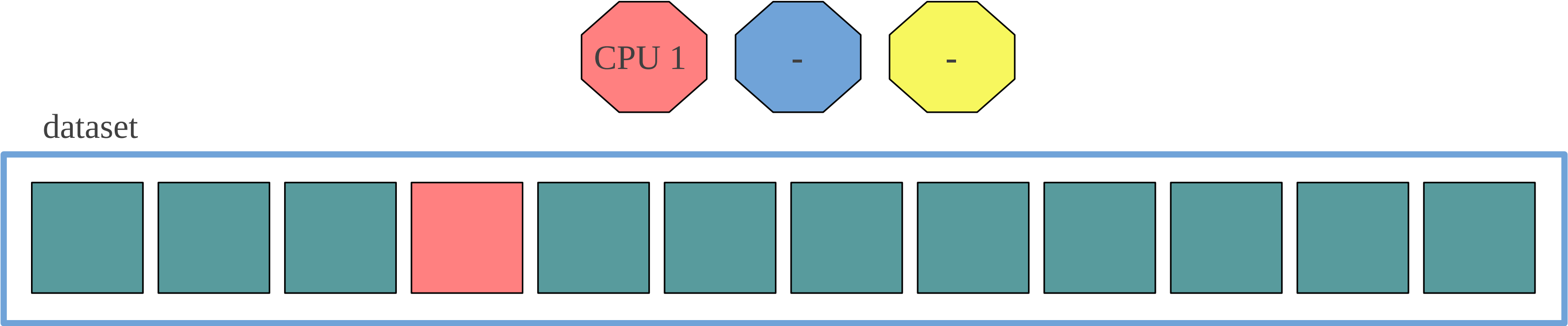
# Sequential iteration



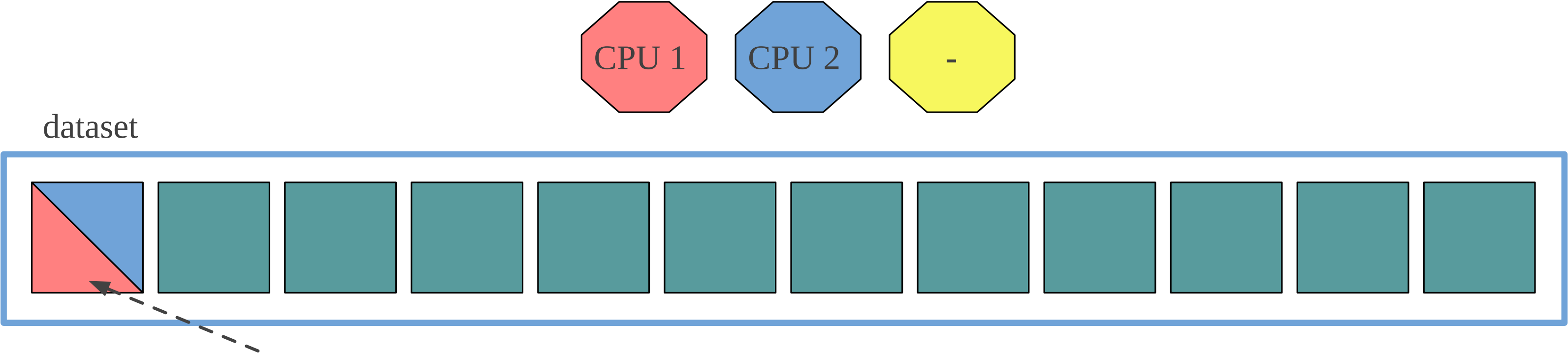
# Sequential iteration



# Sequential iteration

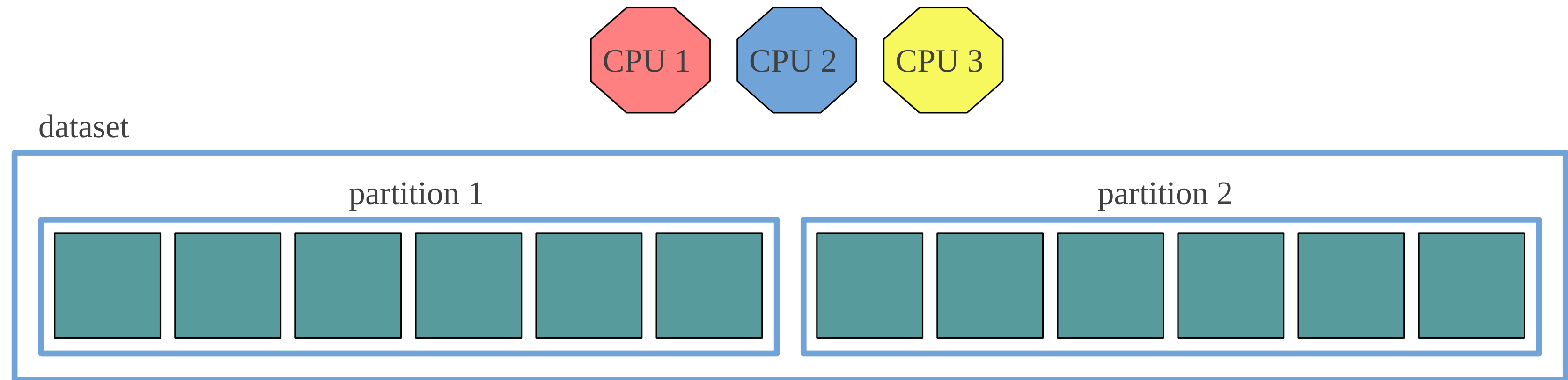


# Useful work



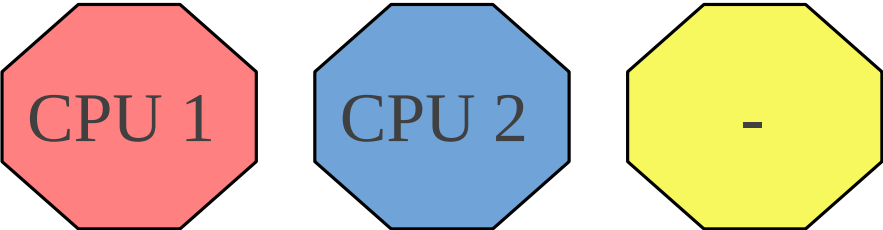
CPU 1 and 2 process the same data

# Partition the dataset

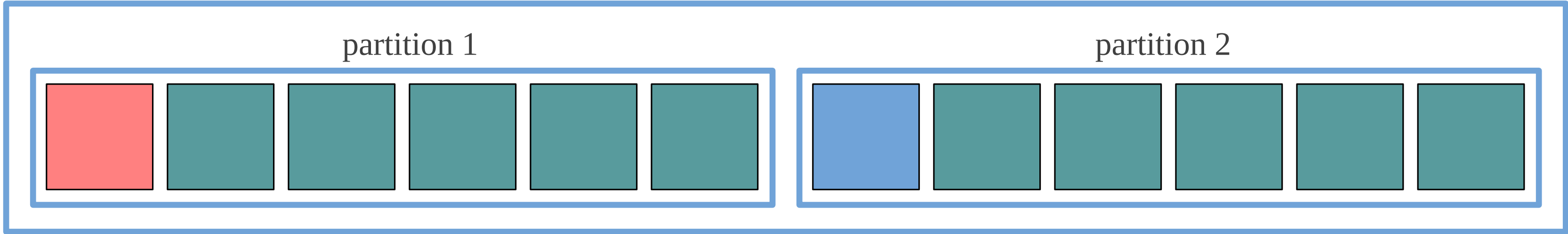




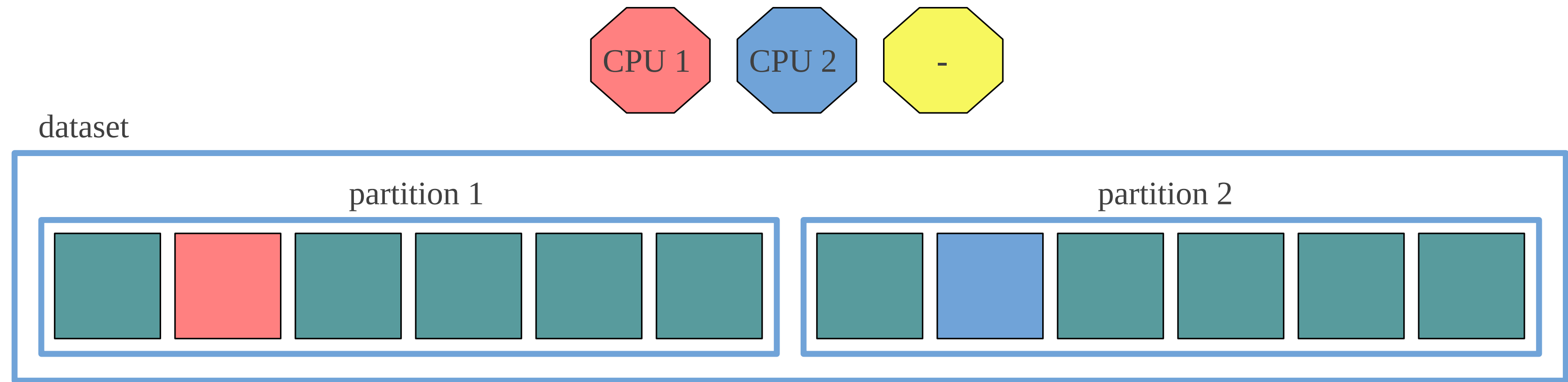
# Partition the dataset



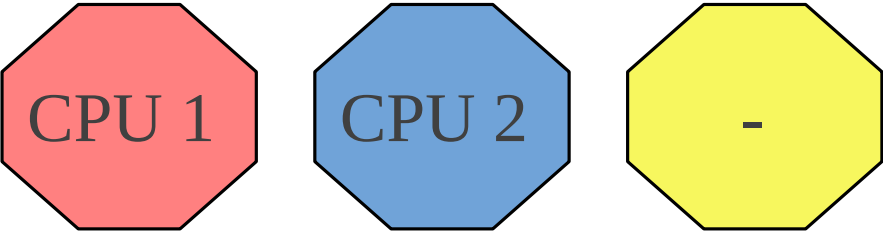
dataset



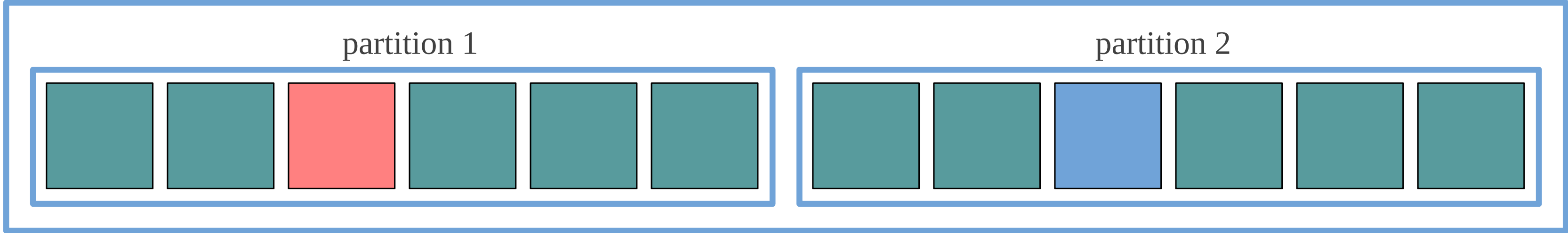
# Partition the dataset



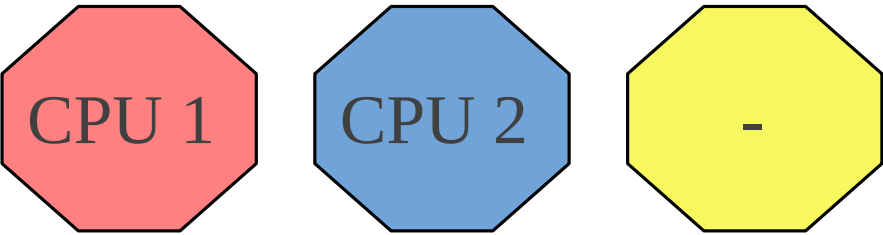
# Partition the dataset



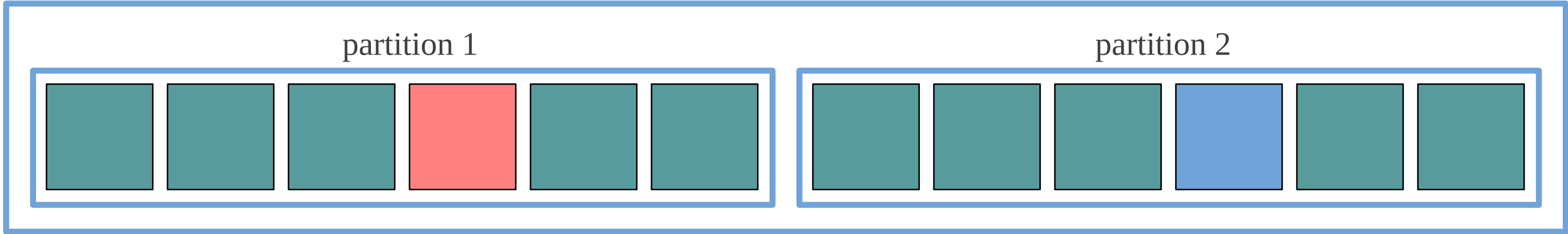
dataset



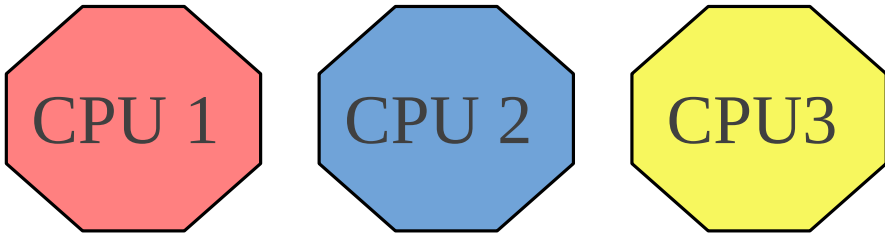
# Partition the dataset



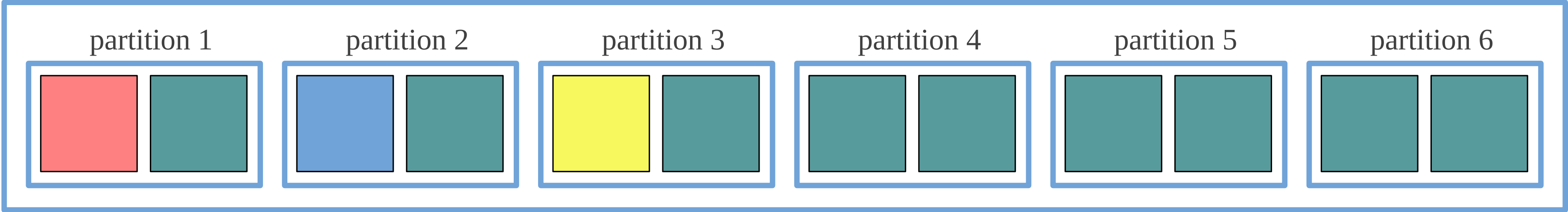
dataset



# Partition the dataset

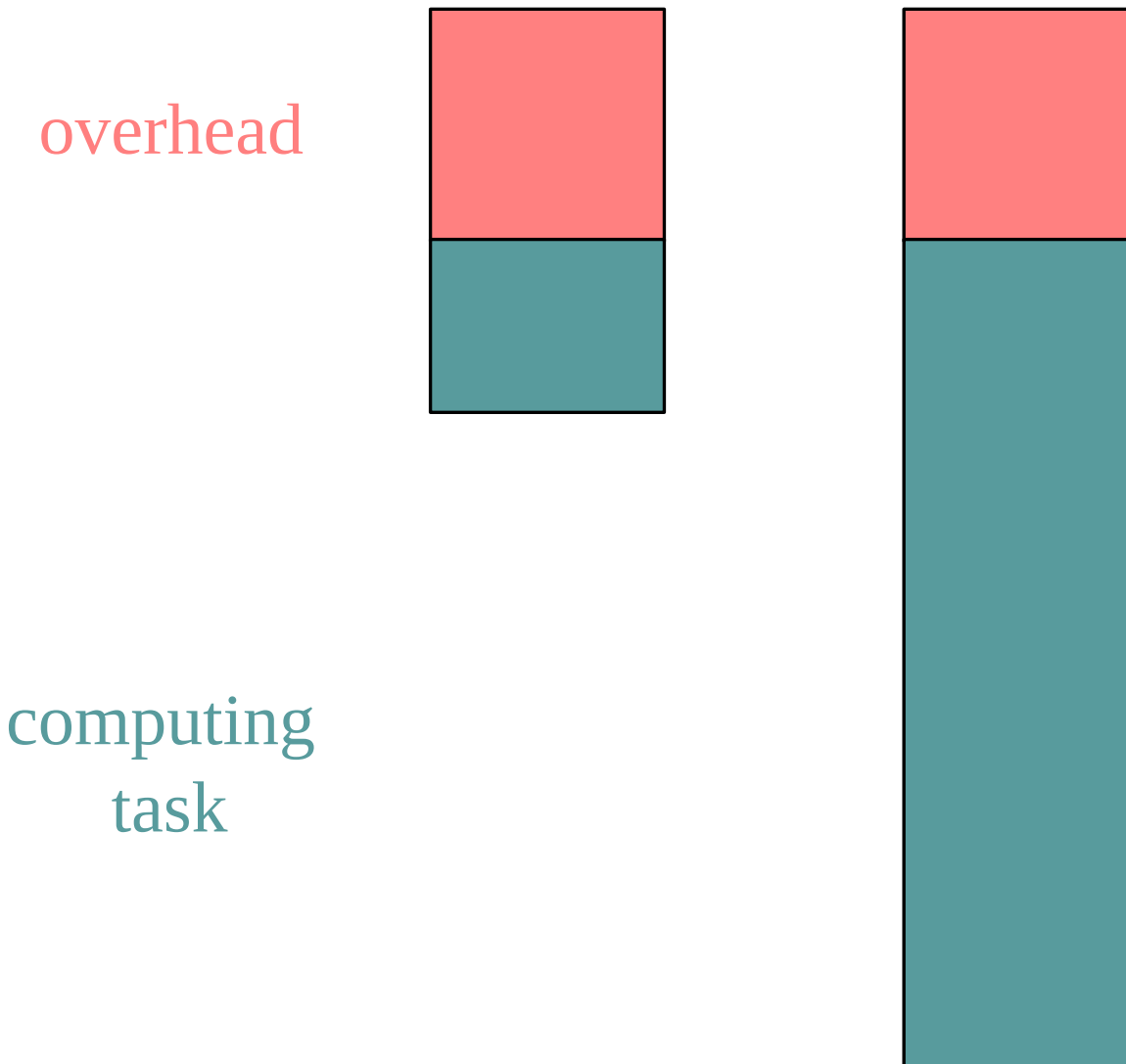


dataset

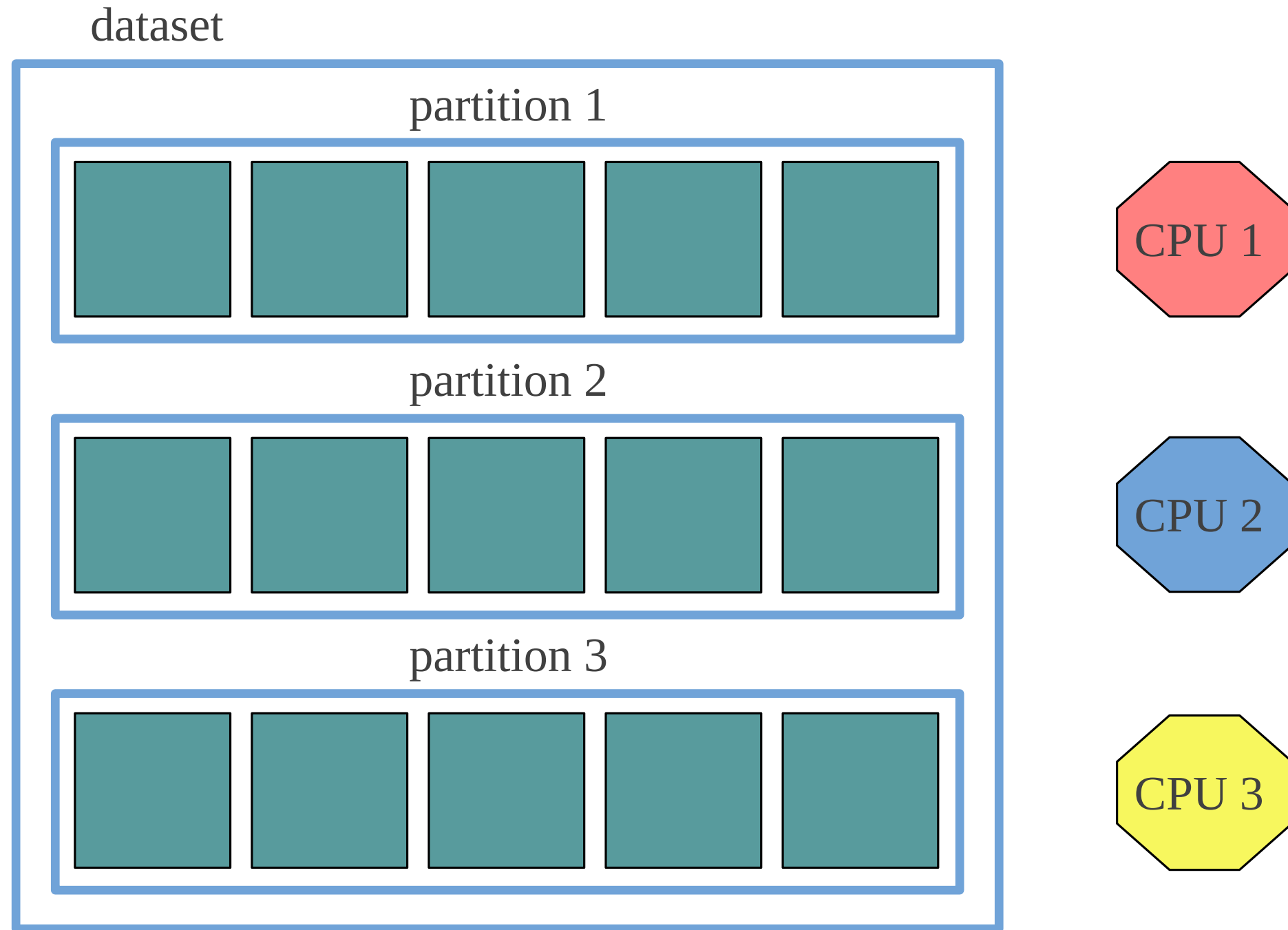


# Important points

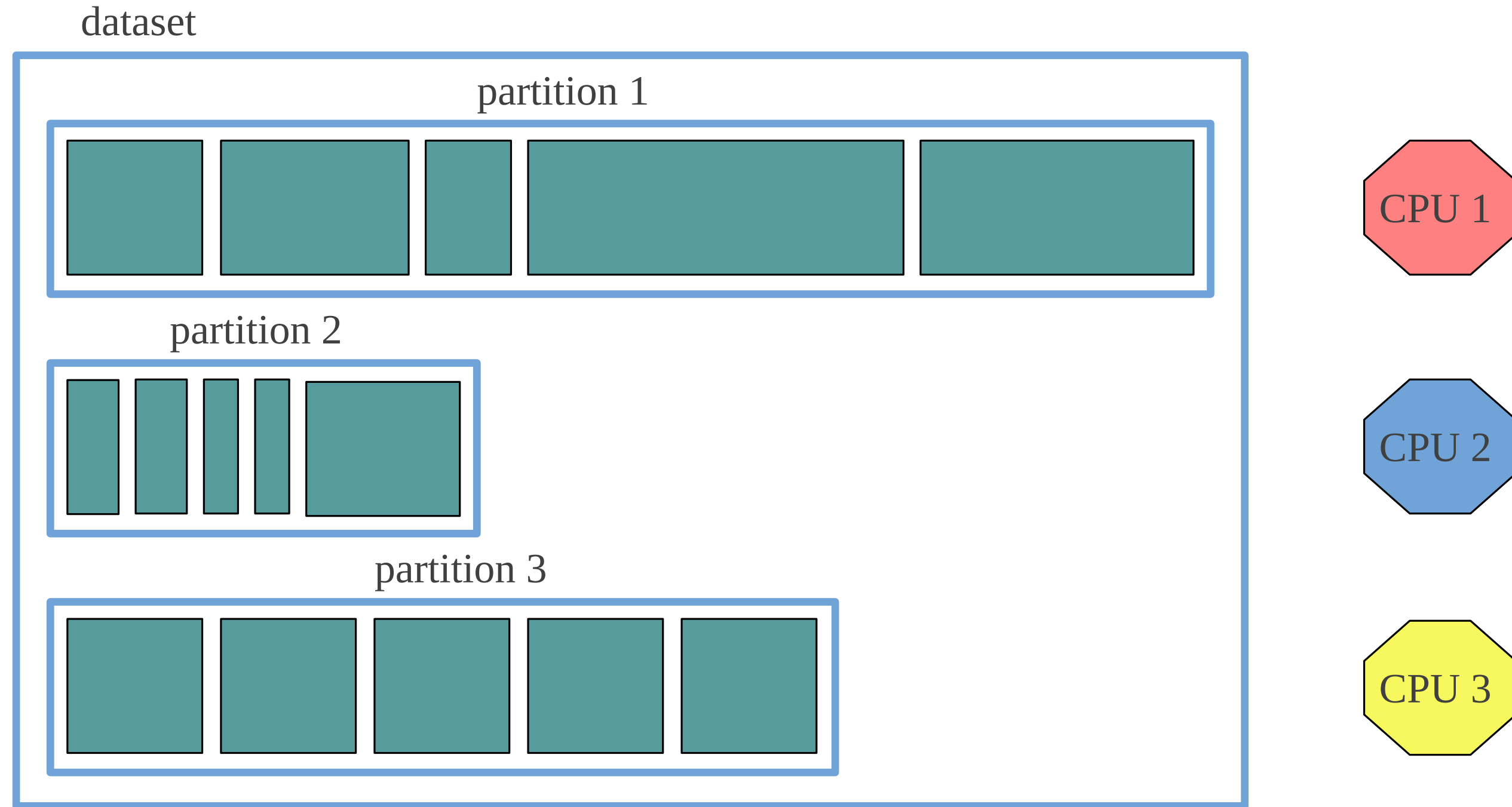
## 1. Size the partitions appropriately



# Tasks are not always as demanding



# Tasks are not always as demanding

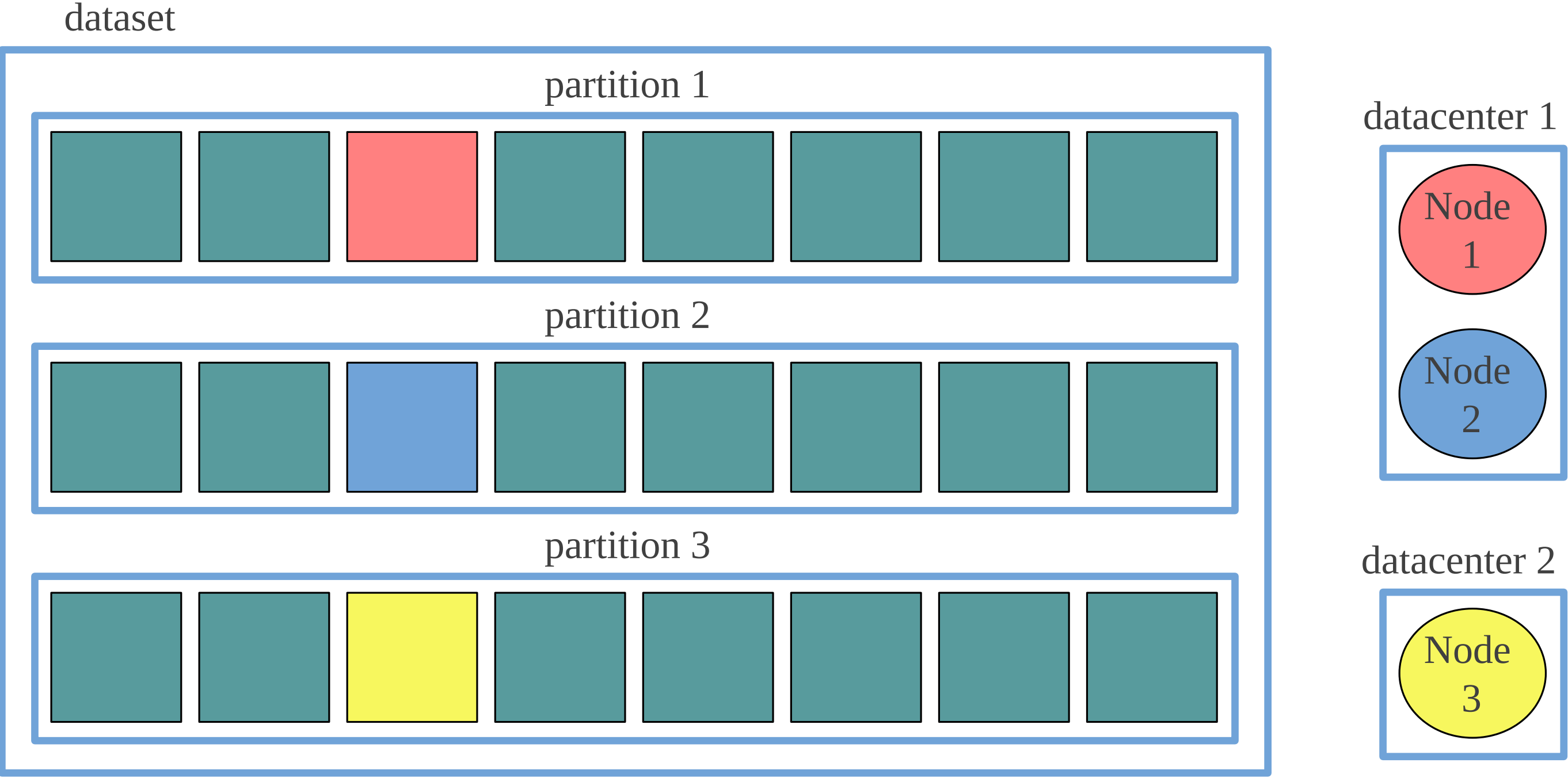




# Important points

1. Size the partitions appropriately
2. Benchmark and tweak configuration for the task at hand

# Scales to more than one computer



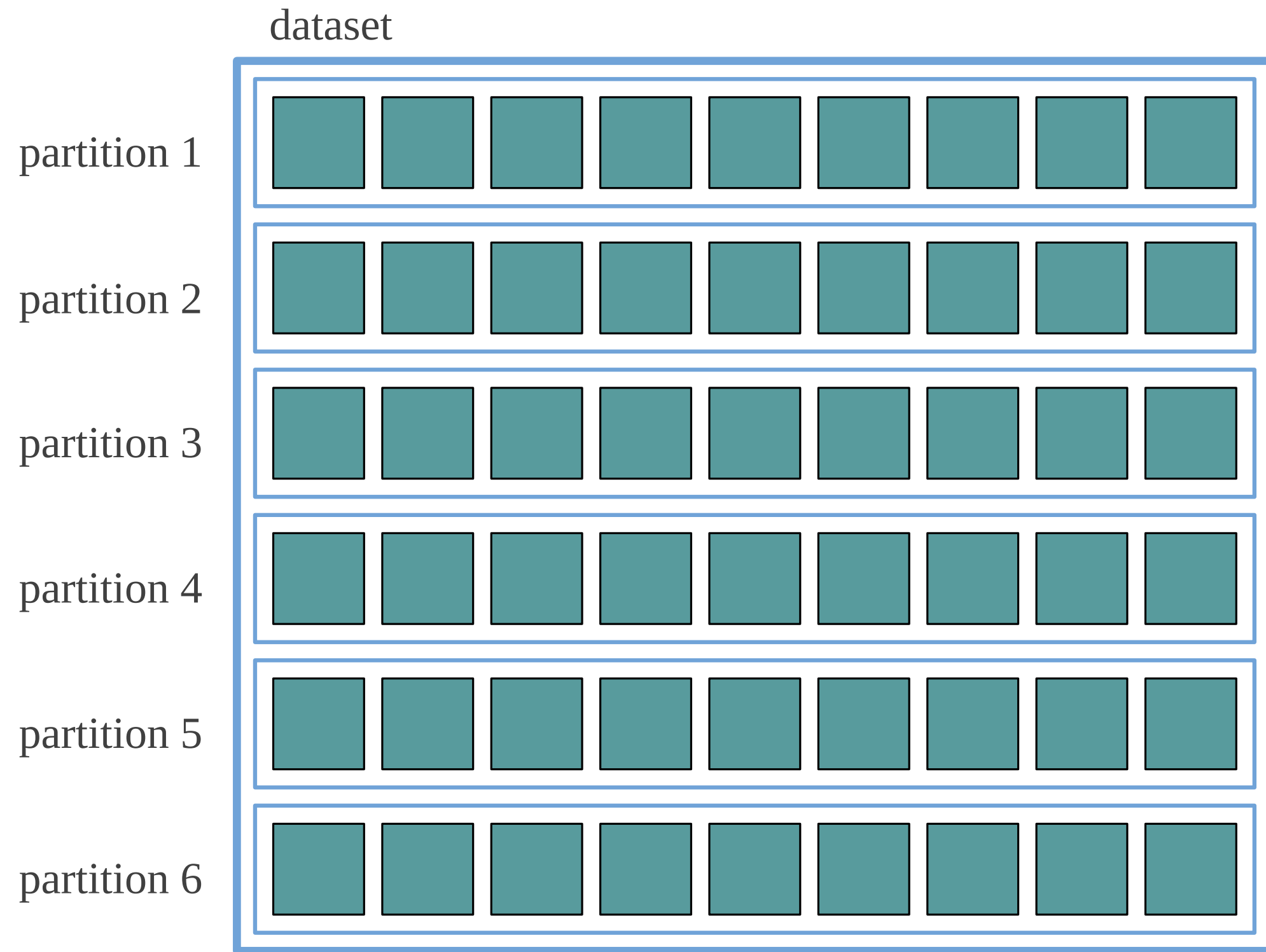
# Important points

1. Size the partitions appropriately
2. Benchmark and tweak configuration for the task at hand
3. Parallel process must produce the same result as the sequential one

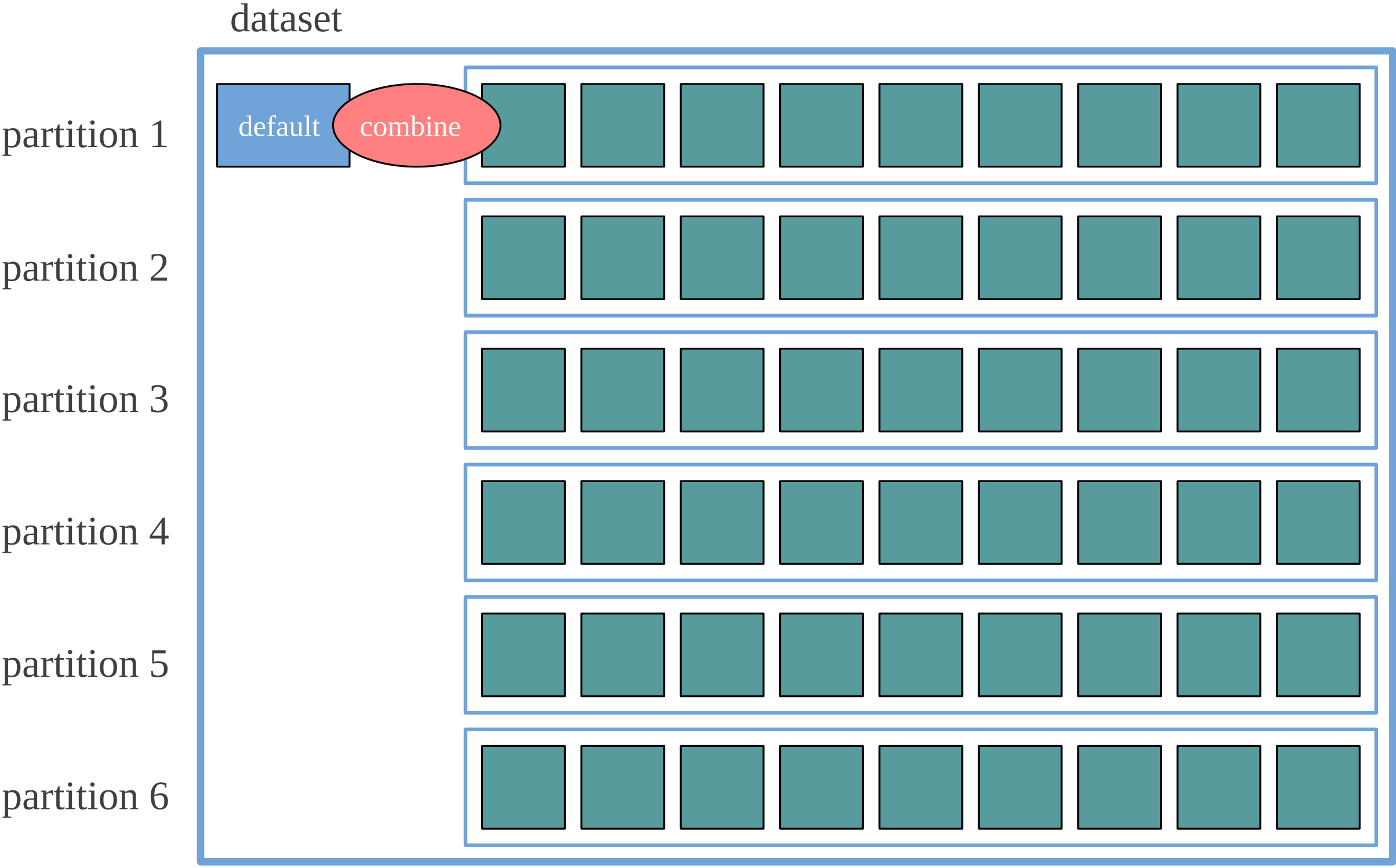
The background features a complex network of blue dots of varying sizes connected by thin, light blue lines. The dots are scattered across the frame, with some forming dense clusters and others standing alone. The lines create a web-like pattern that fills the entire background, giving it a technical or digital feel.

# TemperatureNotebook.scala

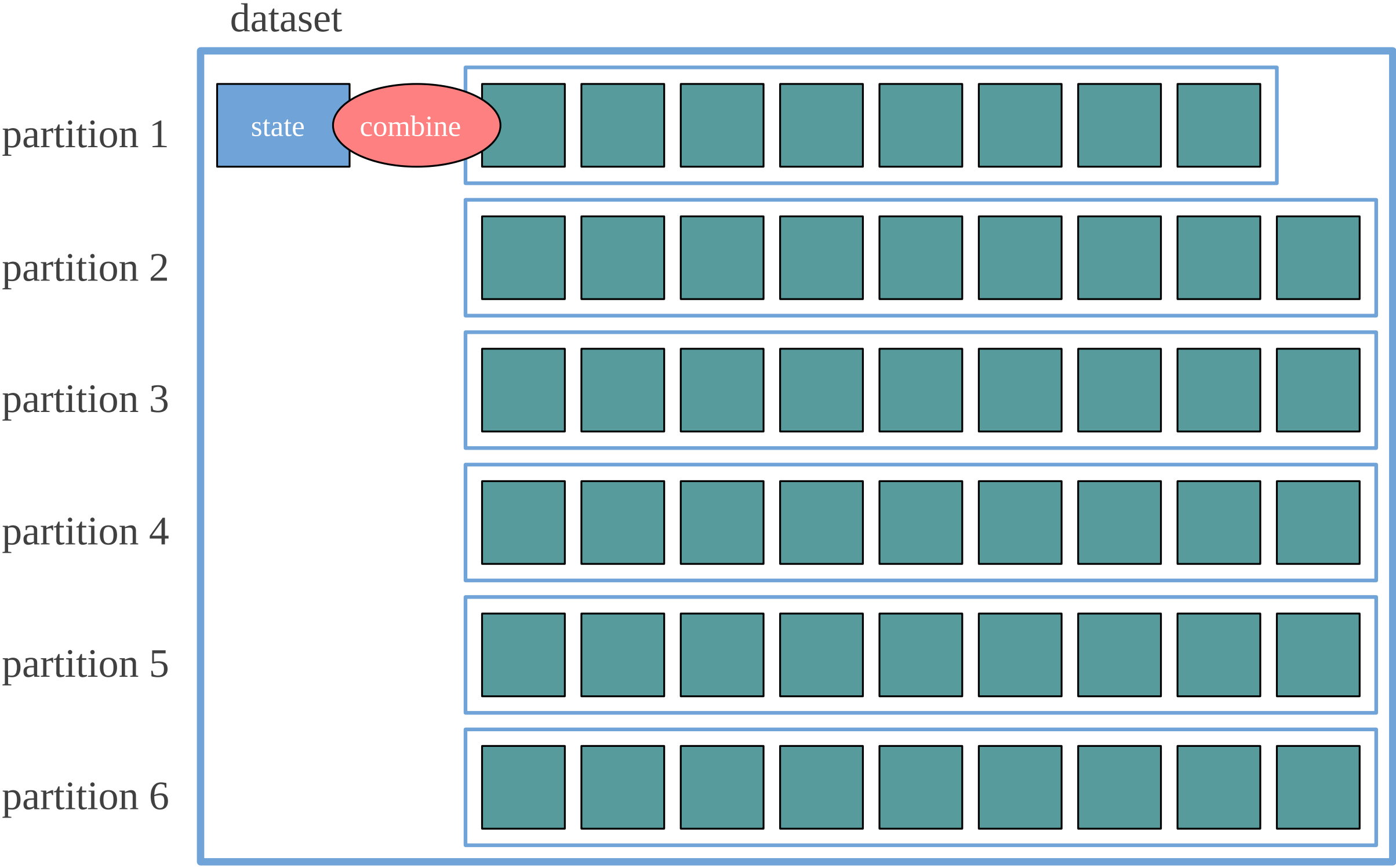
# foldLeft



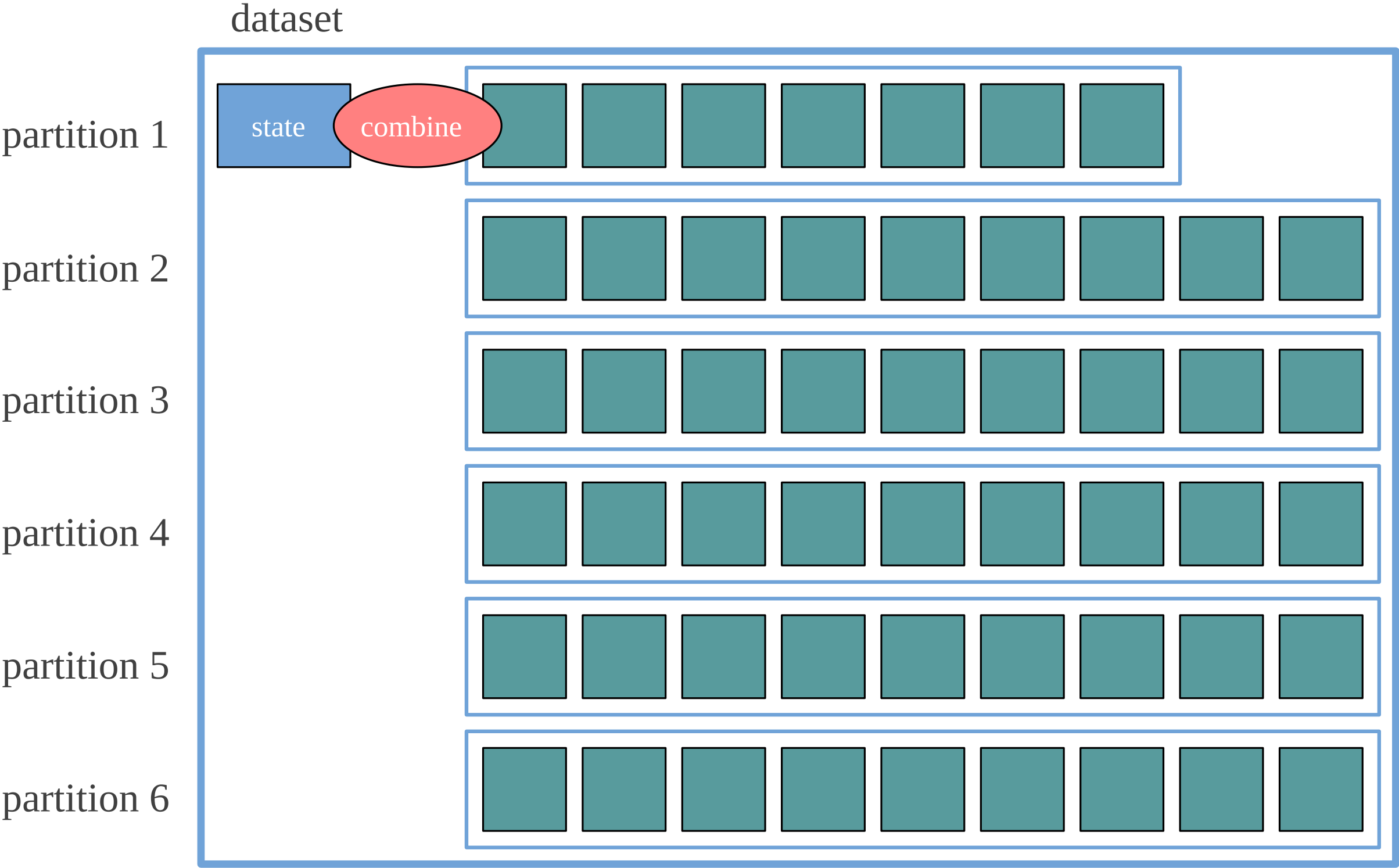
# foldLeft per partition



# foldLeft per partition

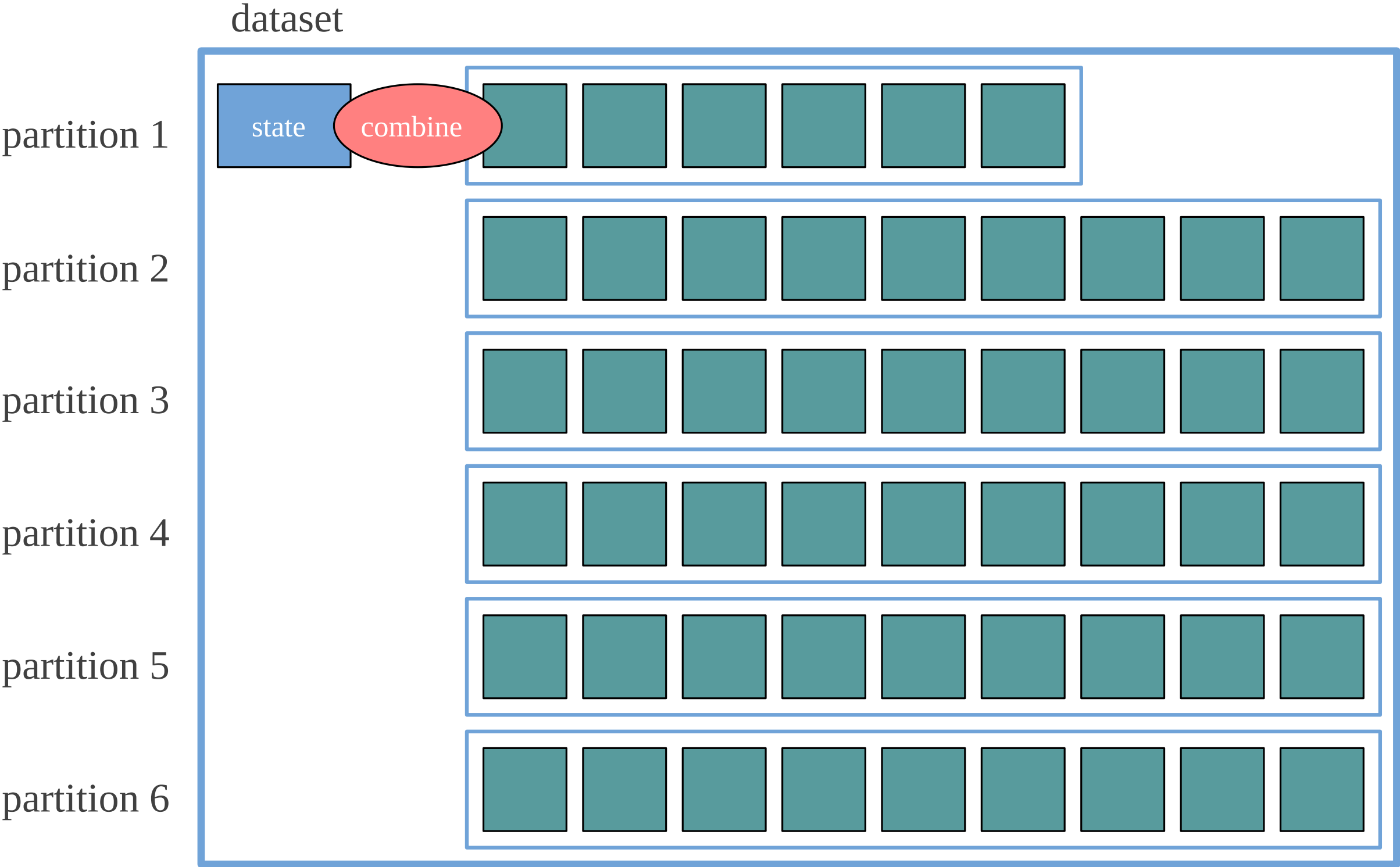


# foldLeft per partition

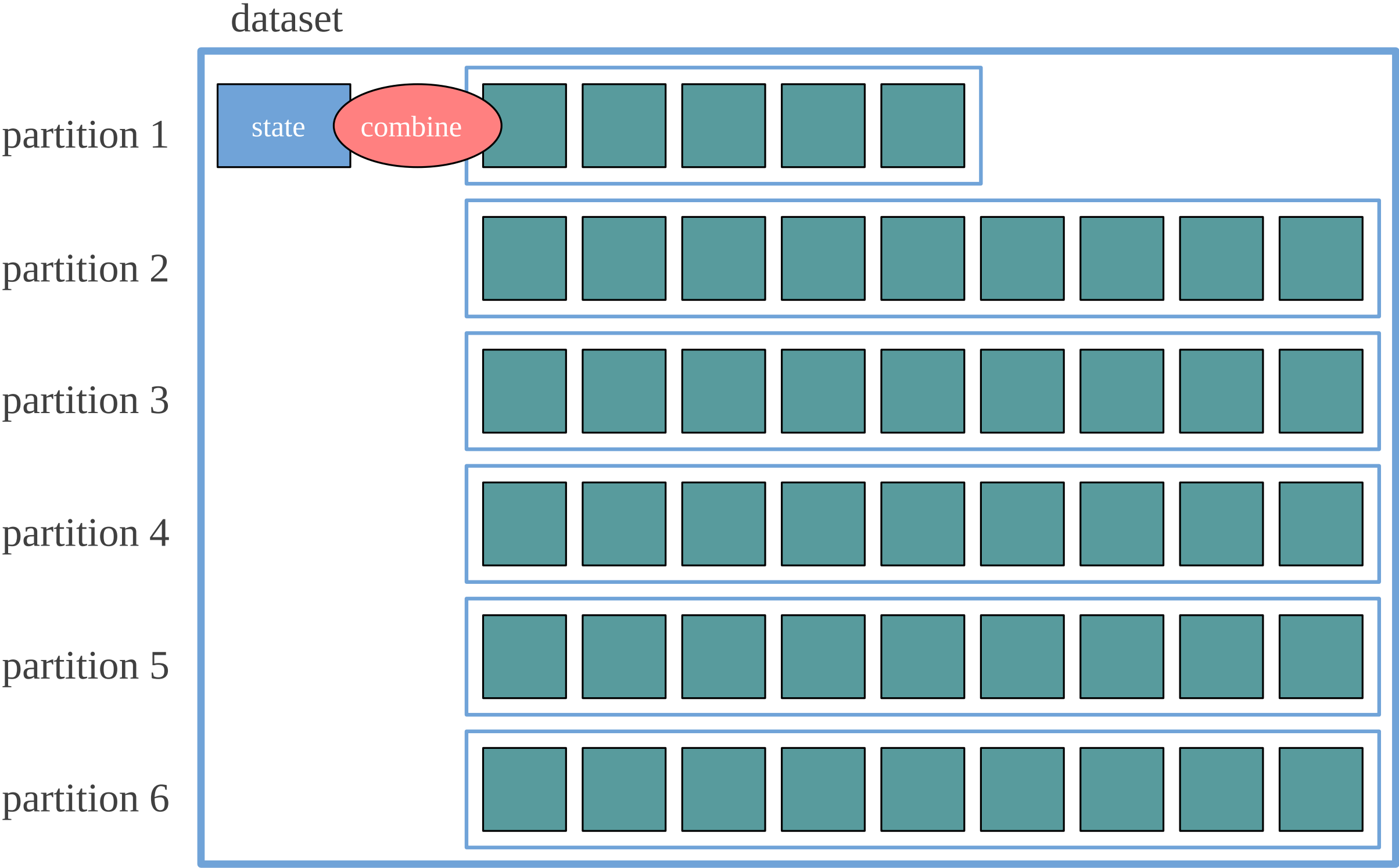




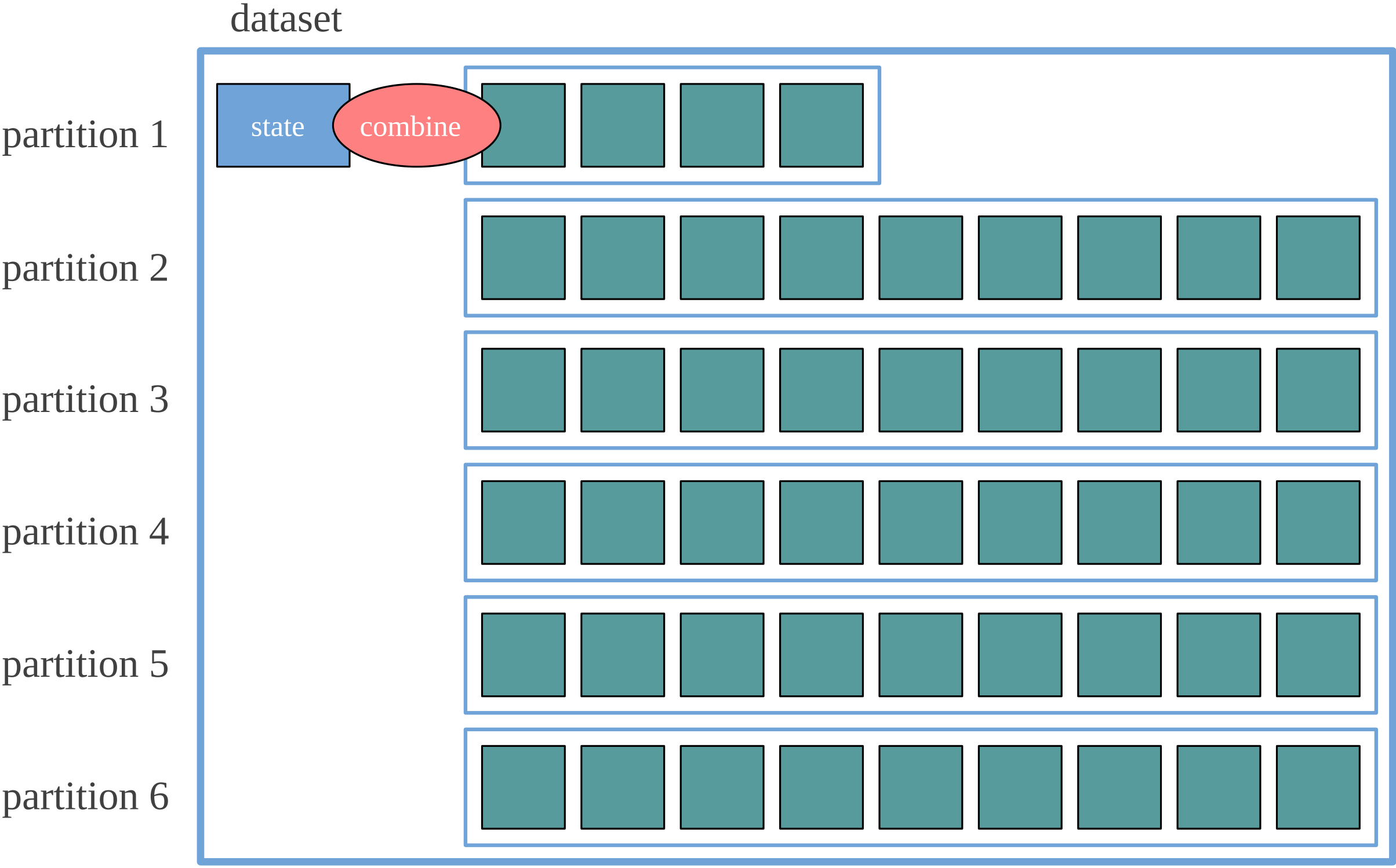
# foldLeft per partition



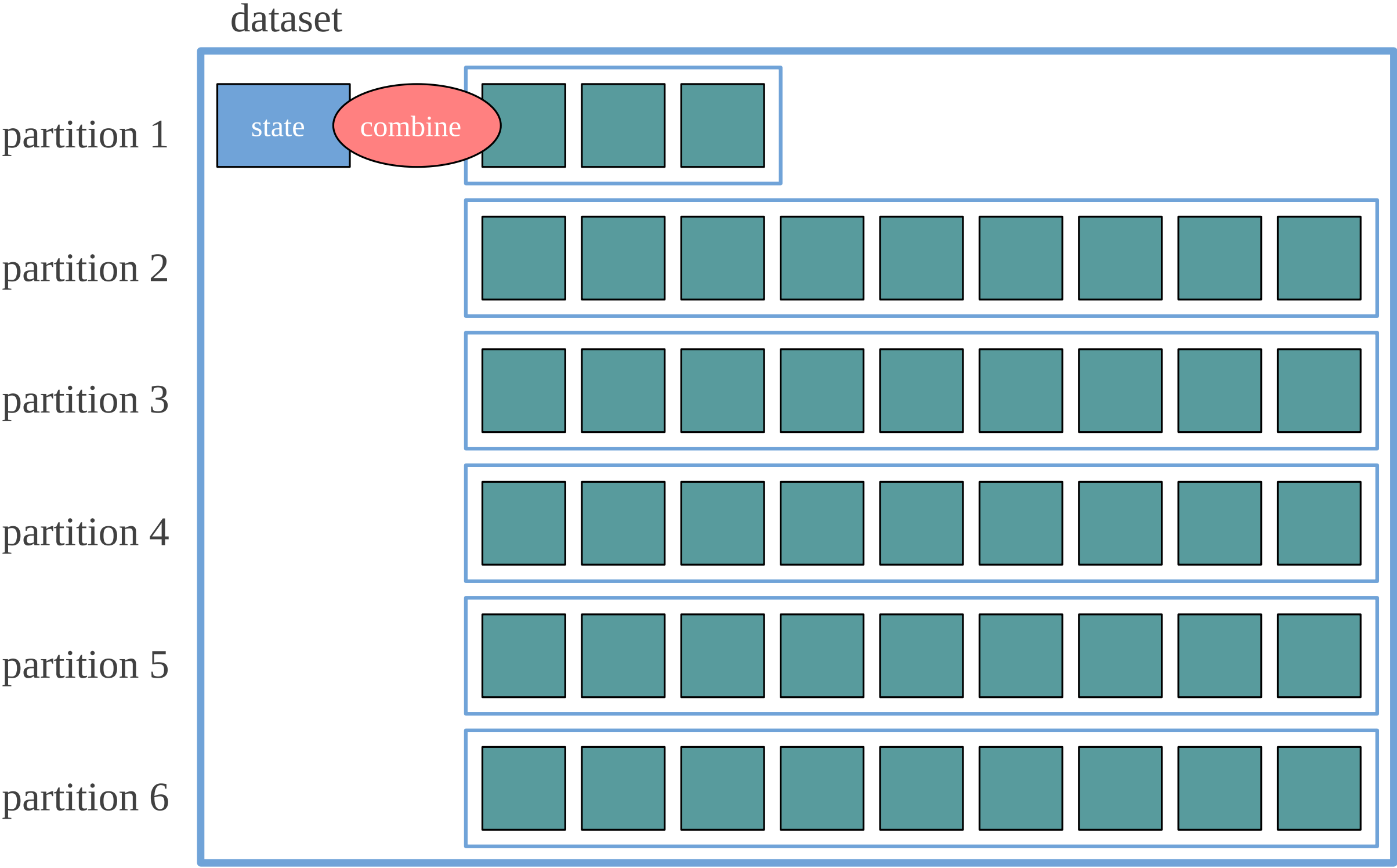
# foldLeft per partition



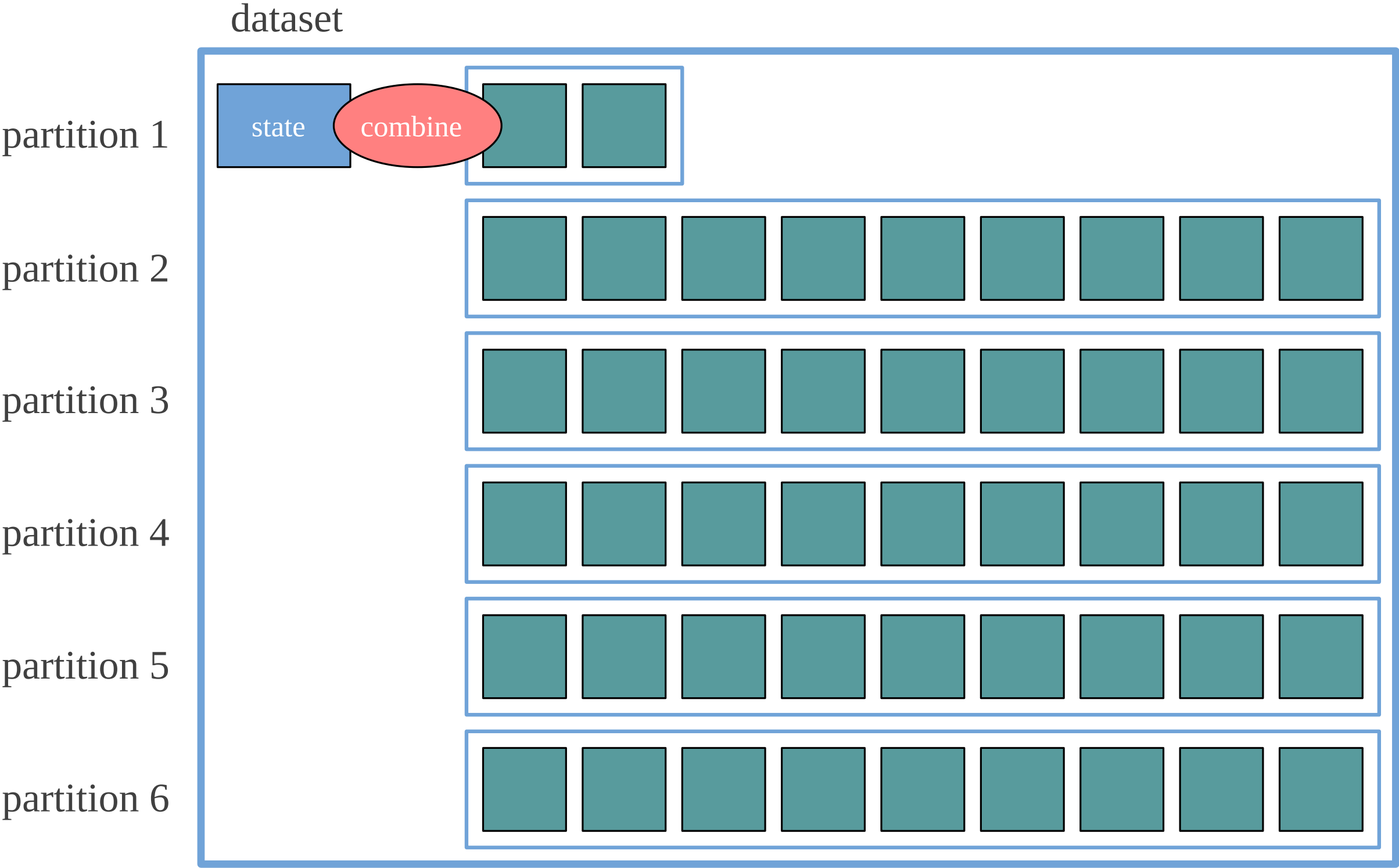
# foldLeft per partition



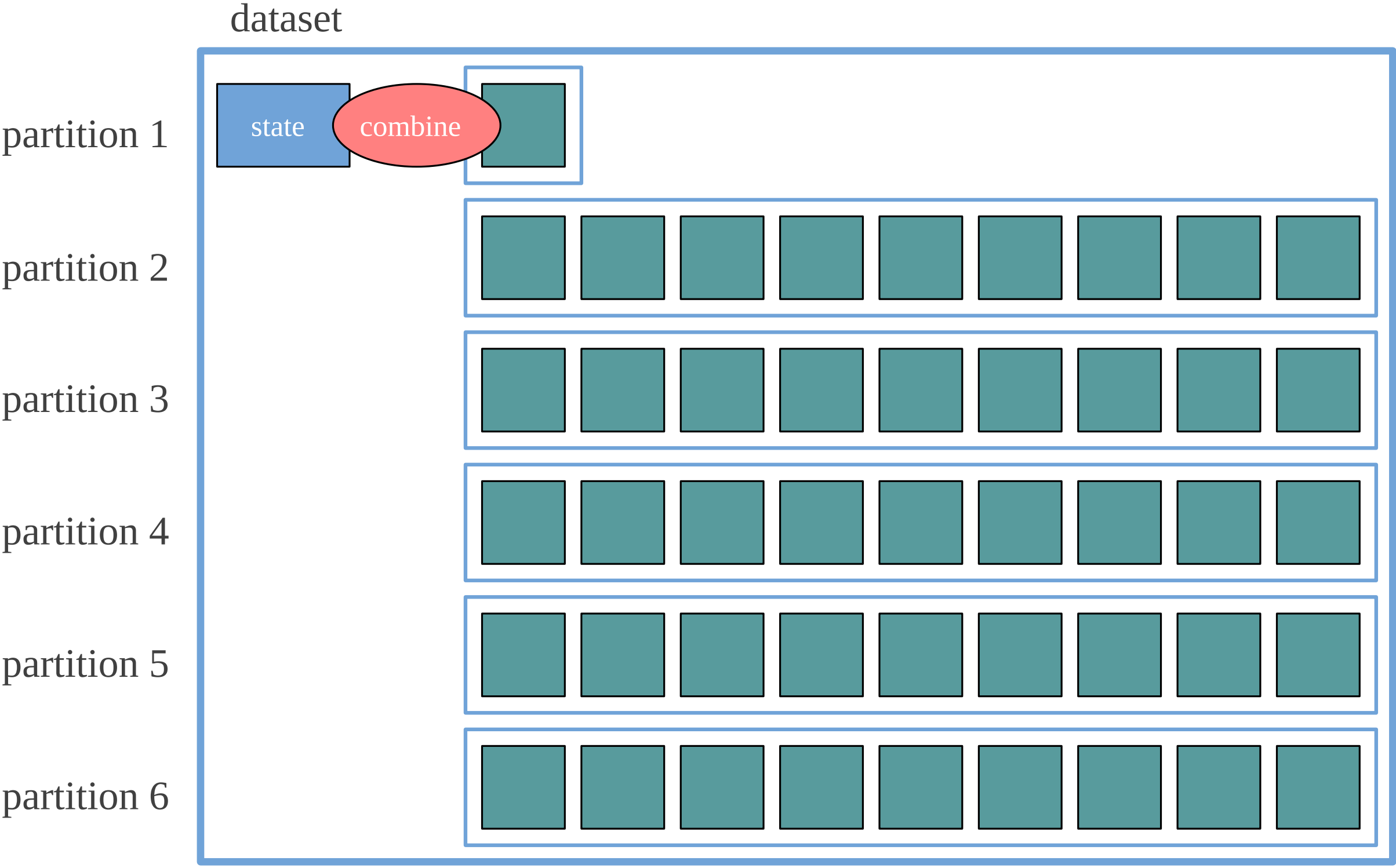
# foldLeft per partition



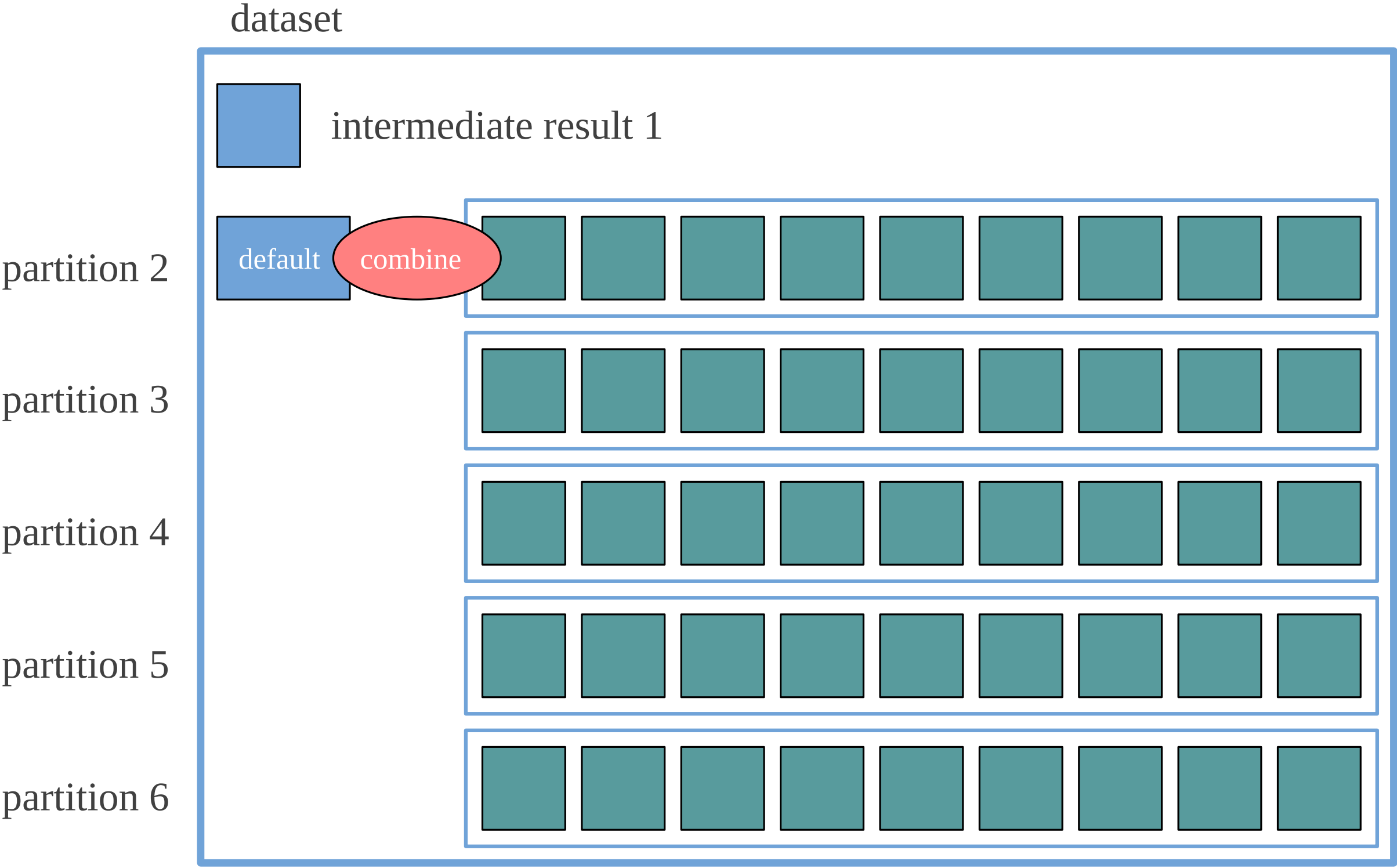
# foldLeft per partition



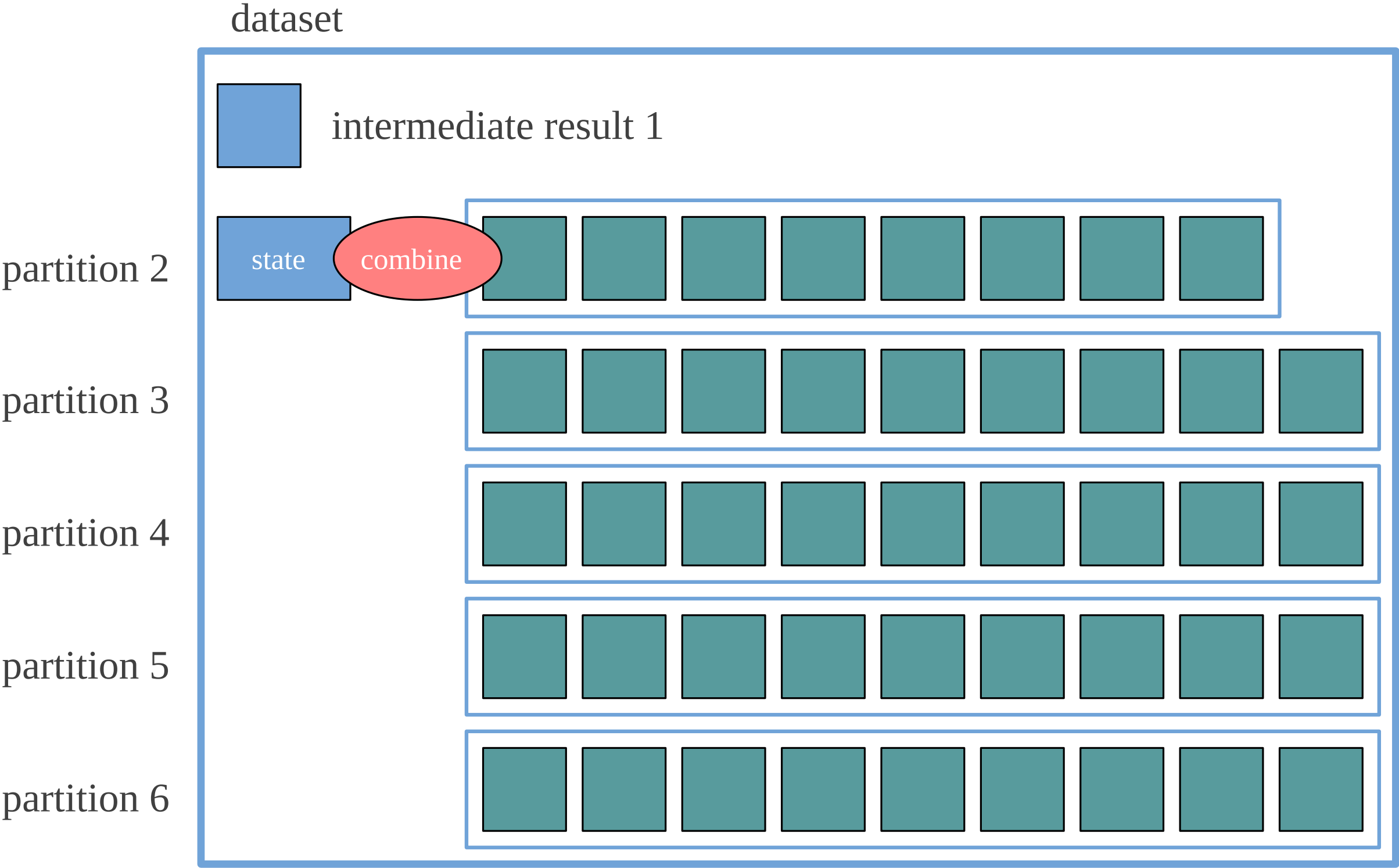
# foldLeft per partition



# foldLeft per partition

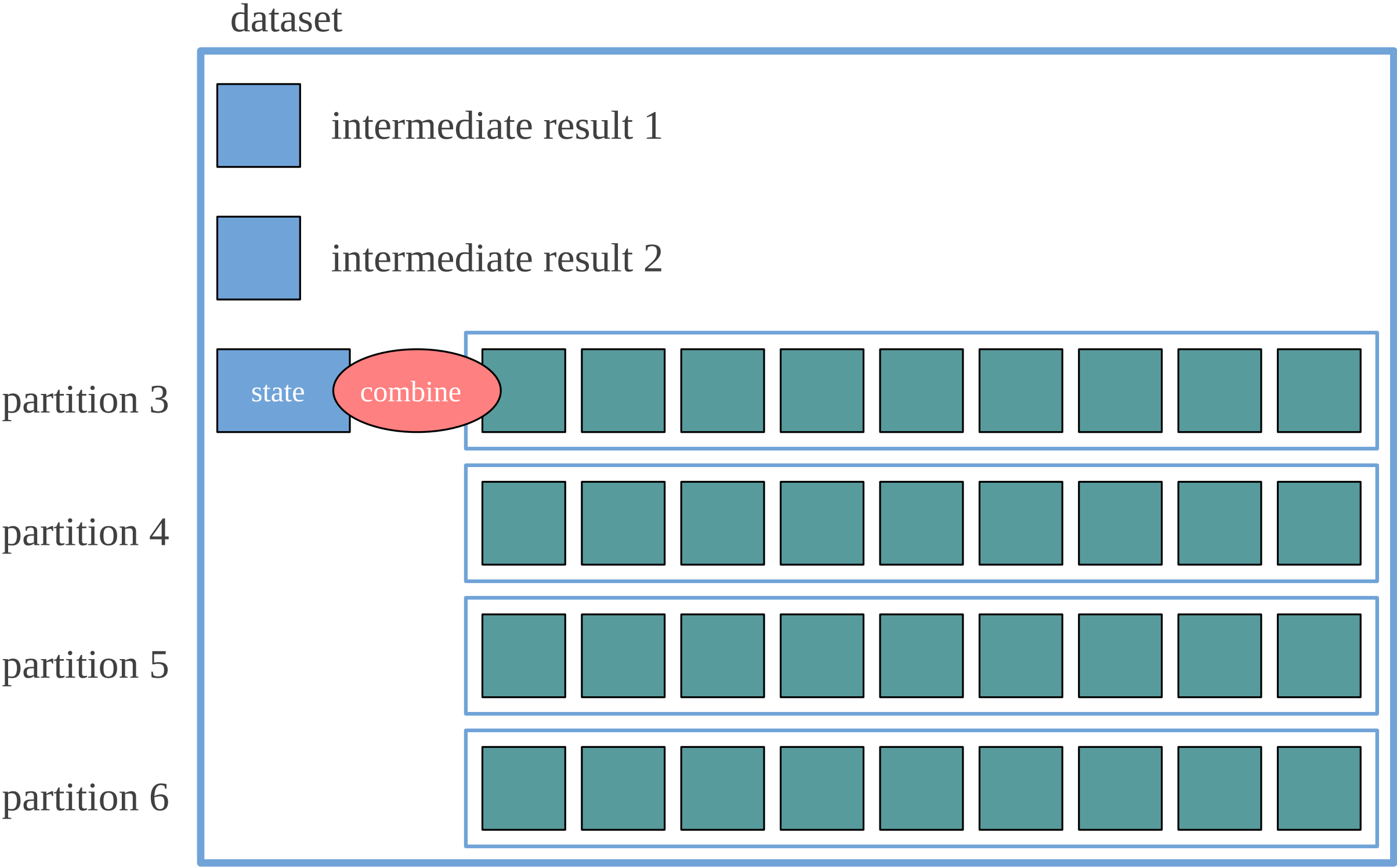


# foldLeft per partition



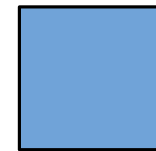


# foldLeft per partition

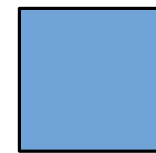


# All partitions folded

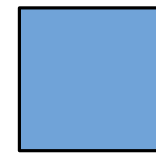
dataset



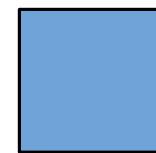
intermediate result 1



intermediate result 2



intermediate result 3



intermediate result 4

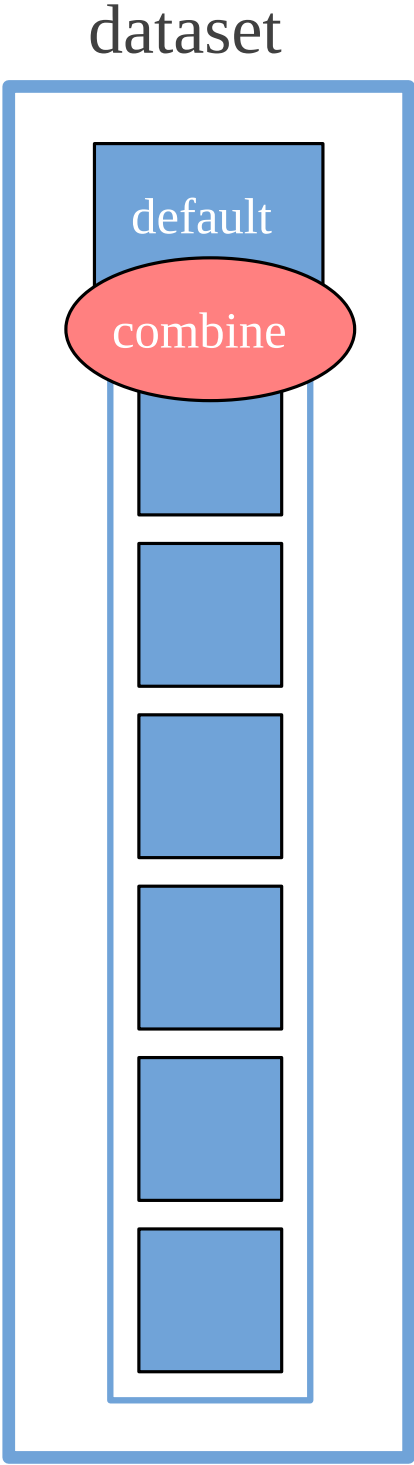


intermediate result 5

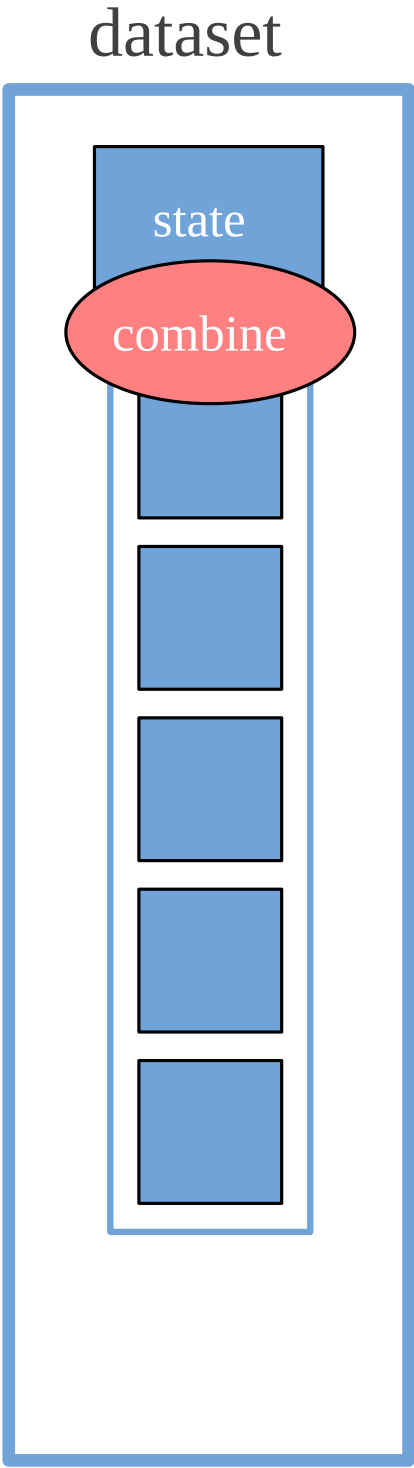


intermediate result 6

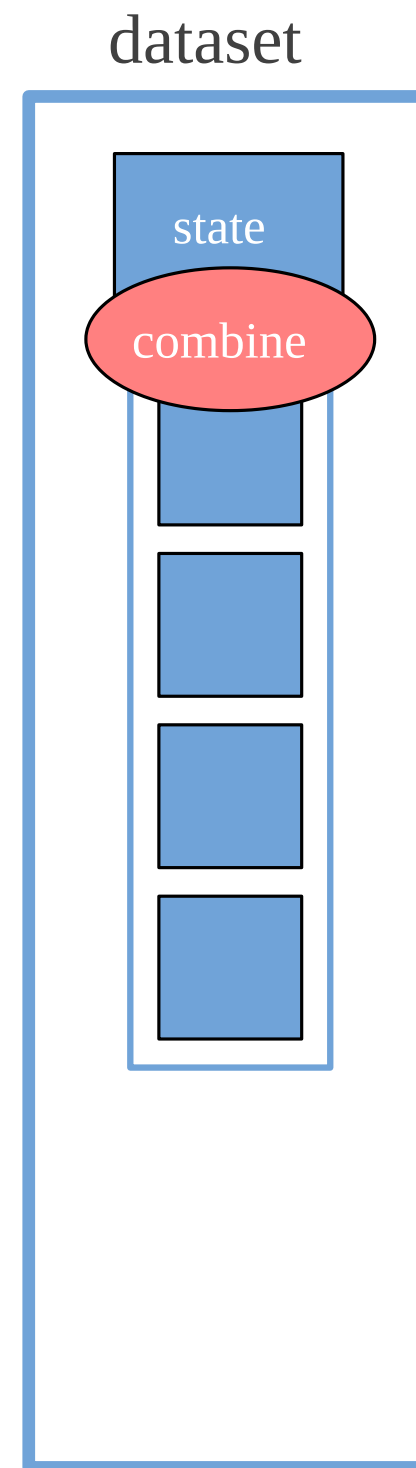
# foldLeft intermediate results



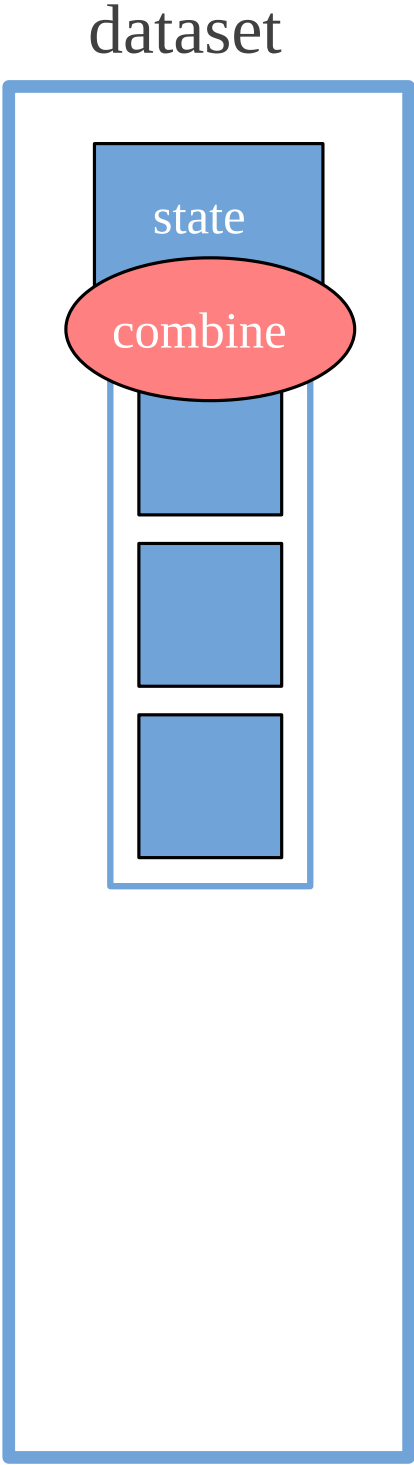
# foldLeft intermediate results



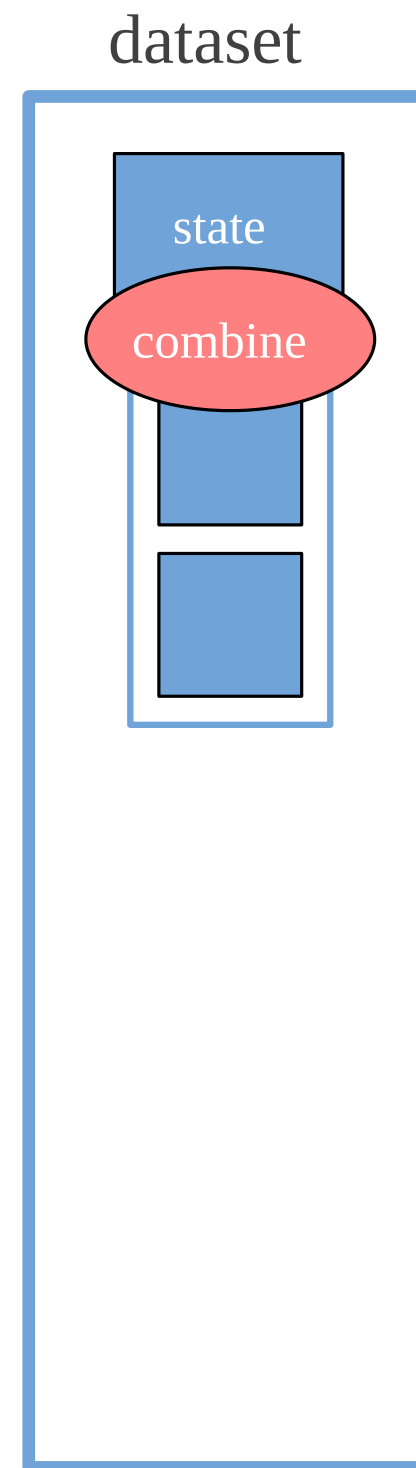
# foldLeft intermediate results



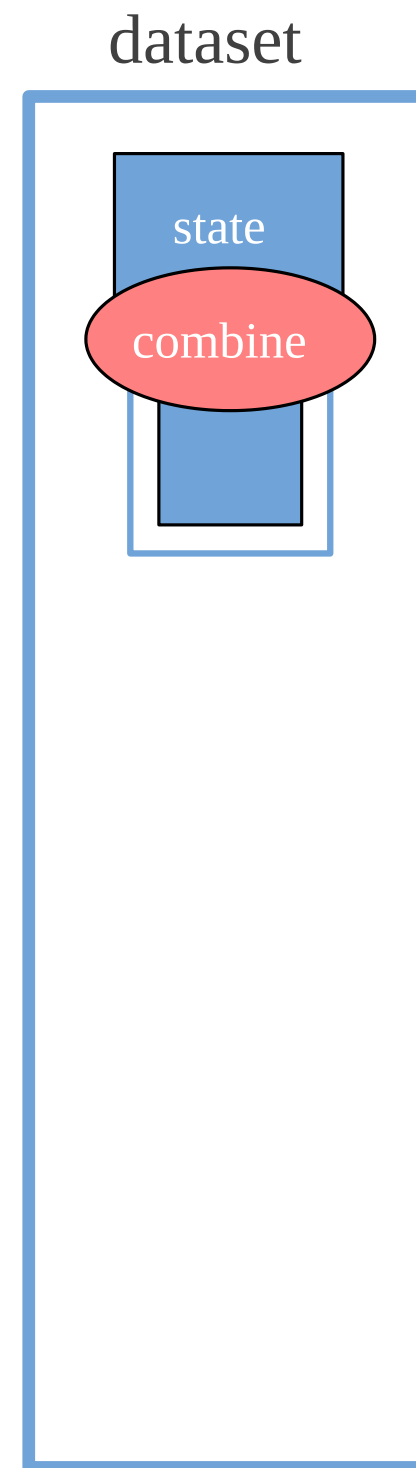
# foldLeft intermediate results



# foldLeft intermediate results



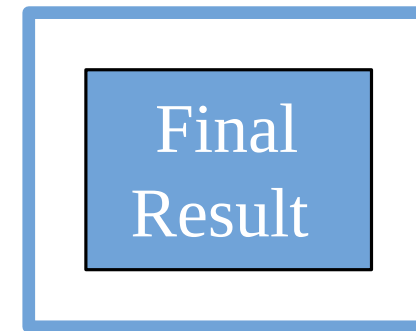
# foldLeft intermediate results





# foldLeft

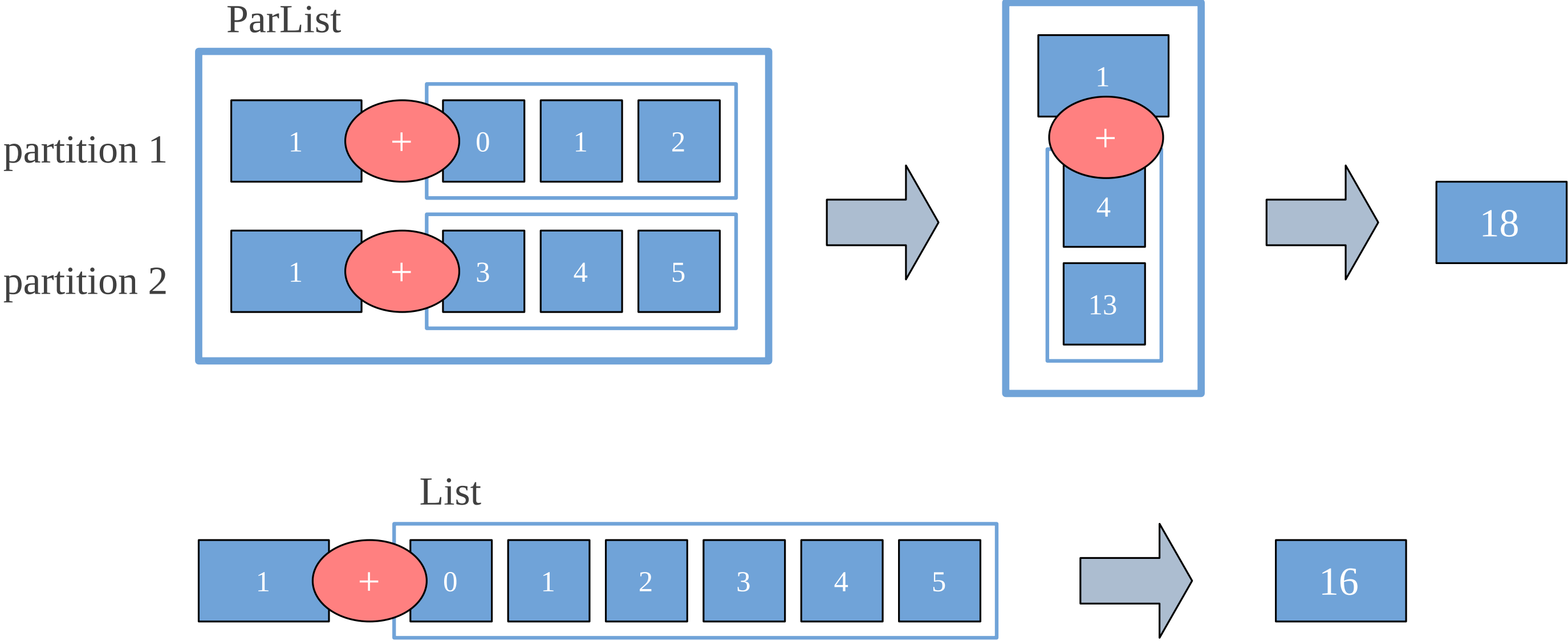
dataset



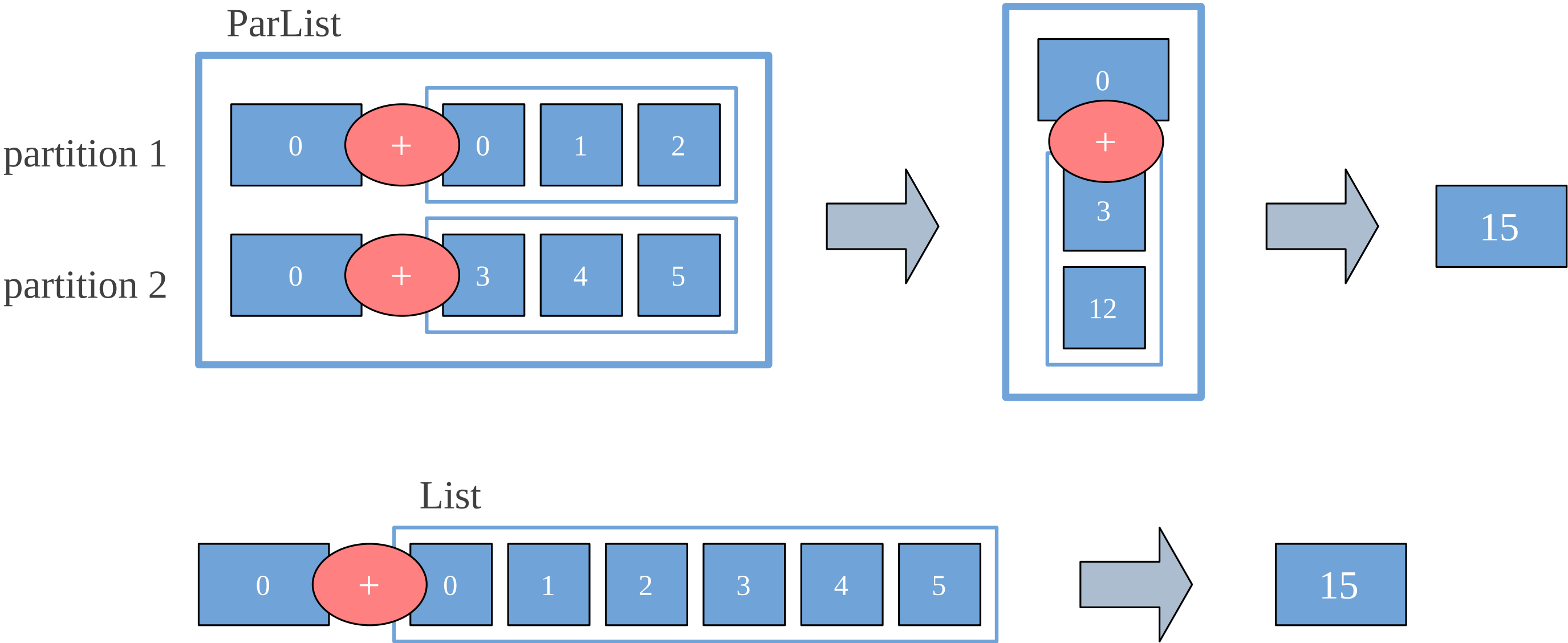
The background of the slide features a complex, abstract network of light blue lines and dots. The dots, representing nodes, vary in size and are connected by thin, intersecting lines that form a web-like structure across the entire frame. The overall aesthetic is clean and modern, with a light blue color palette.

# TemperatureExercises.scala

# monoFoldLeft vs List foldLeft



# monoFoldLeft vs List foldLeft



`combine(default, x) == x`

`combine(default, x) == x == combine(x, default)`

```
average(10, 12) = ???
```

```
average(10, 12) = (10 + 12) / 2  
                = 11
```



```
average(10, 12, 14) = (10 + 12 + 14) / 3  
                    = 12
```

$$\begin{aligned}\text{average}(\text{average}(10, 12), 14) &= \text{average}(11, 14) \\ &= (11 + 14) / 2 \\ &= 12.5\end{aligned}$$

$$\begin{aligned}\text{average}(10, \text{average}(12, 14)) &= \text{average}(10, 13) \\ &= (10 + 13) / 2 \\ &= 11.5\end{aligned}$$

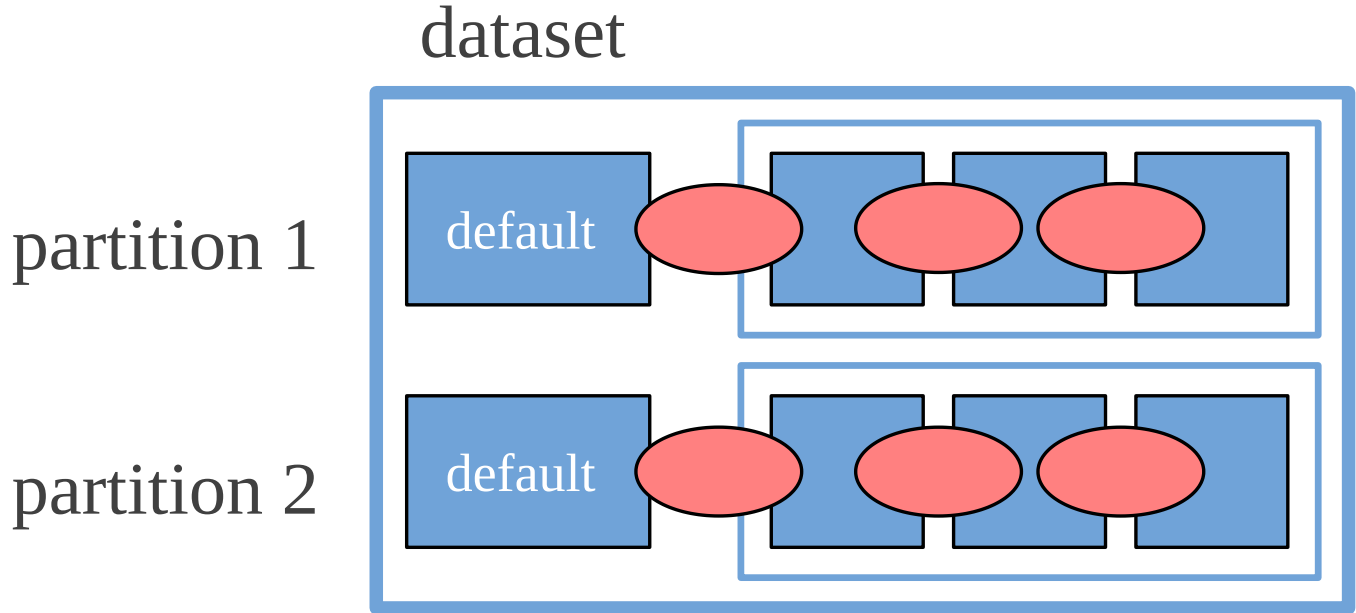
$$\text{average}(10, 12, 14) = 12$$

# Associative functions

```
(1 + (2 + 3)) == ((1 + 2) + 3)  
// res0: Boolean = true
```

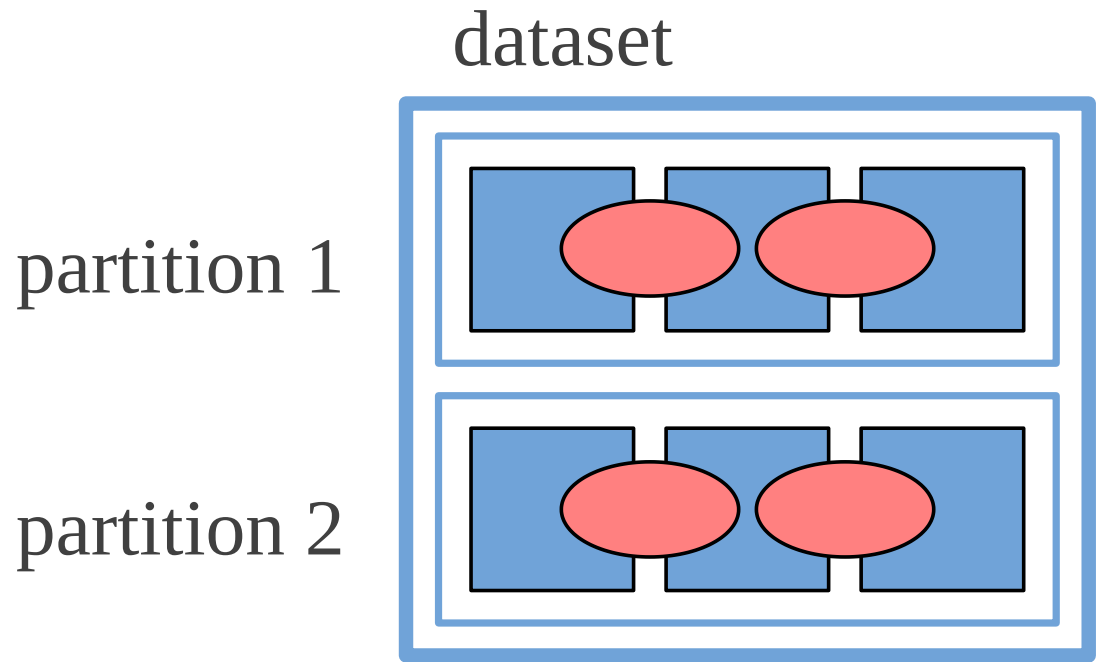
```
(1 min (2 min 3)) == ((1 min 2) min 3)  
// res1: Boolean = true
```

# monoFoldLeft



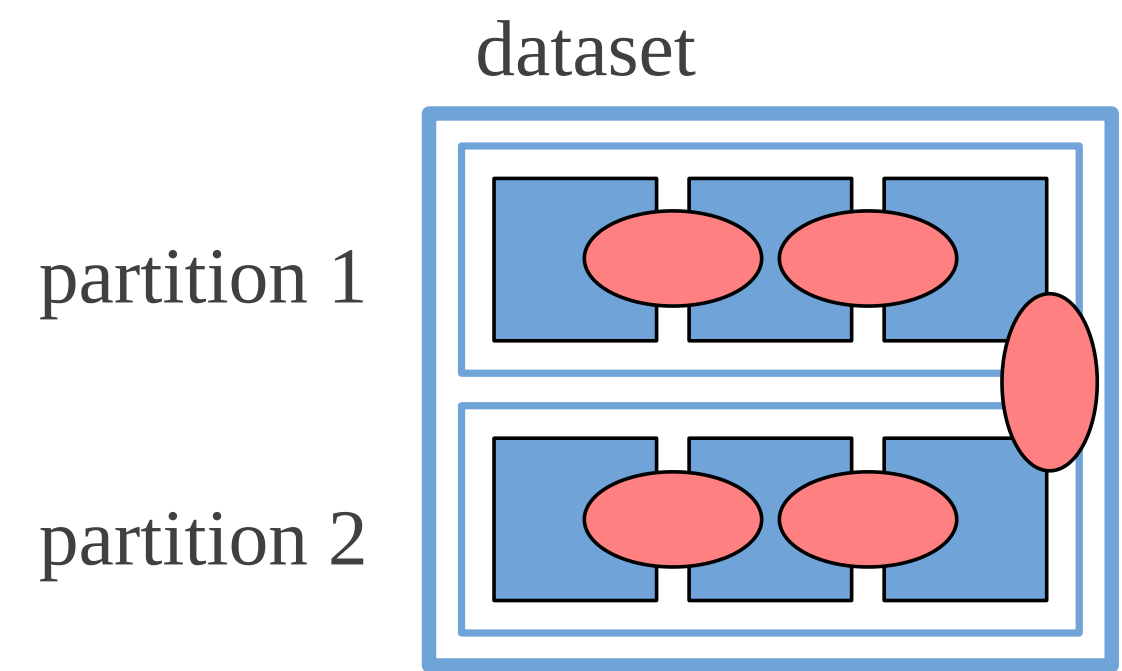
```
(default combine a1 combine a2 combine a3)  
(default combine b1 combine b2 combine b3)
```

# monoFoldLeft



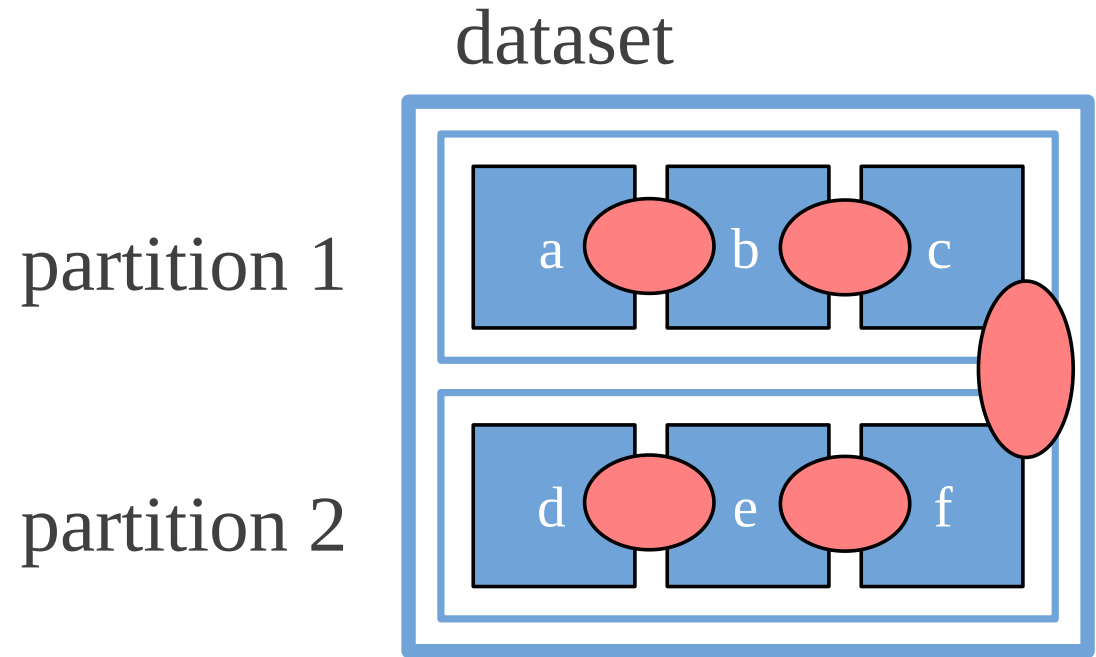
```
(a1 combine a2 combine a3)  
(b1 combine b2 combine b3)
```

# monoFoldLeft

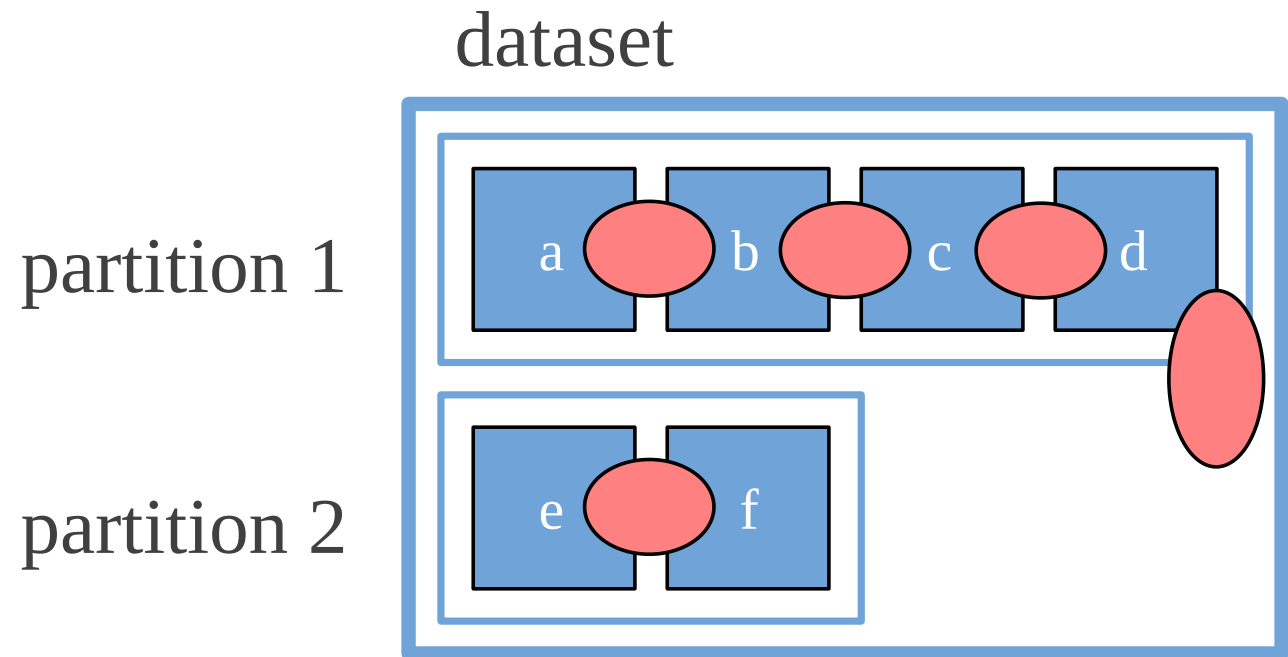


```
(a1 combine a2 combine a3) combine  
(b1 combine b2 combine b3)
```

# monoFoldLeft requires combine to be associative

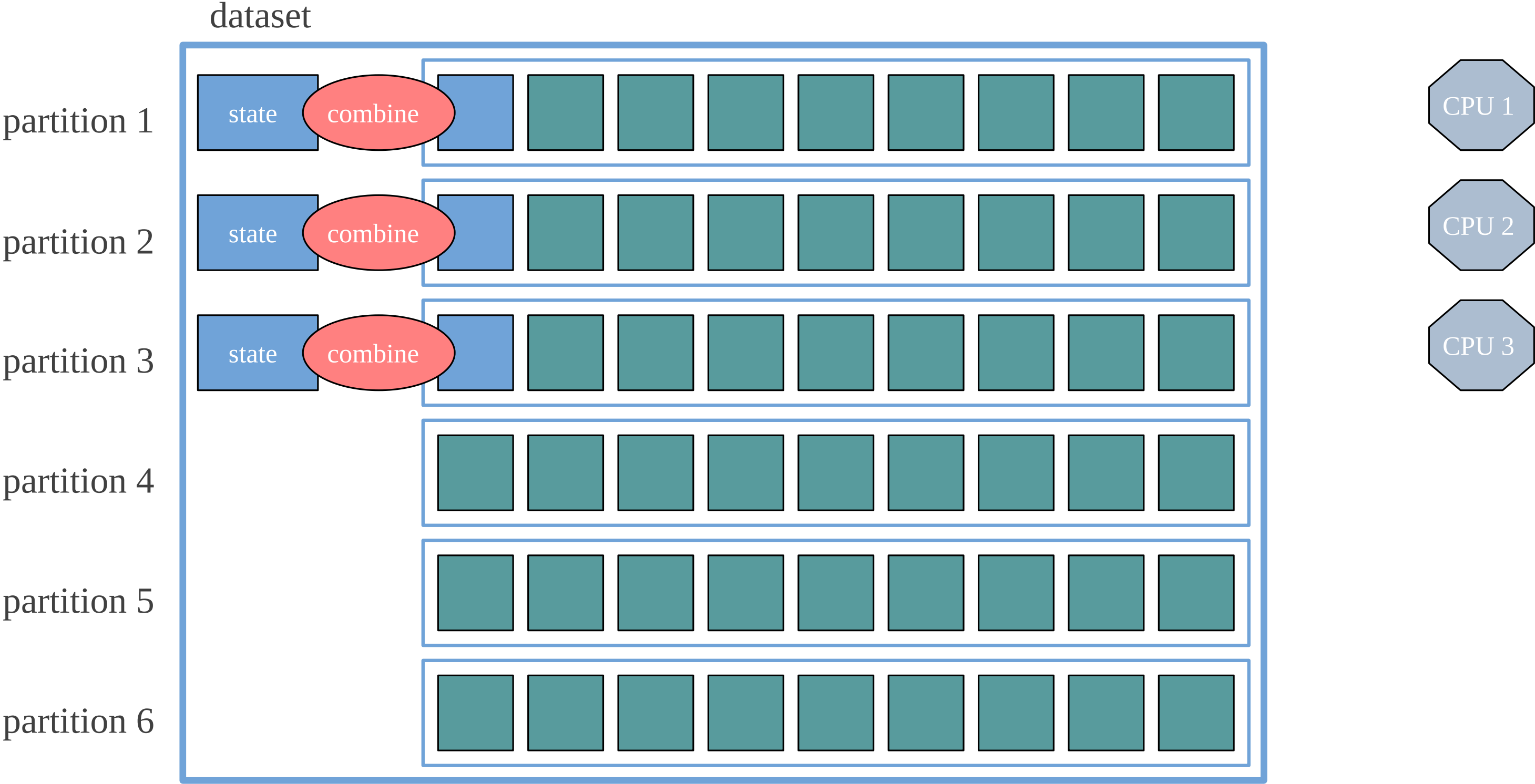


```
(a combine b combine c) combine  
(d combine e combine f)
```



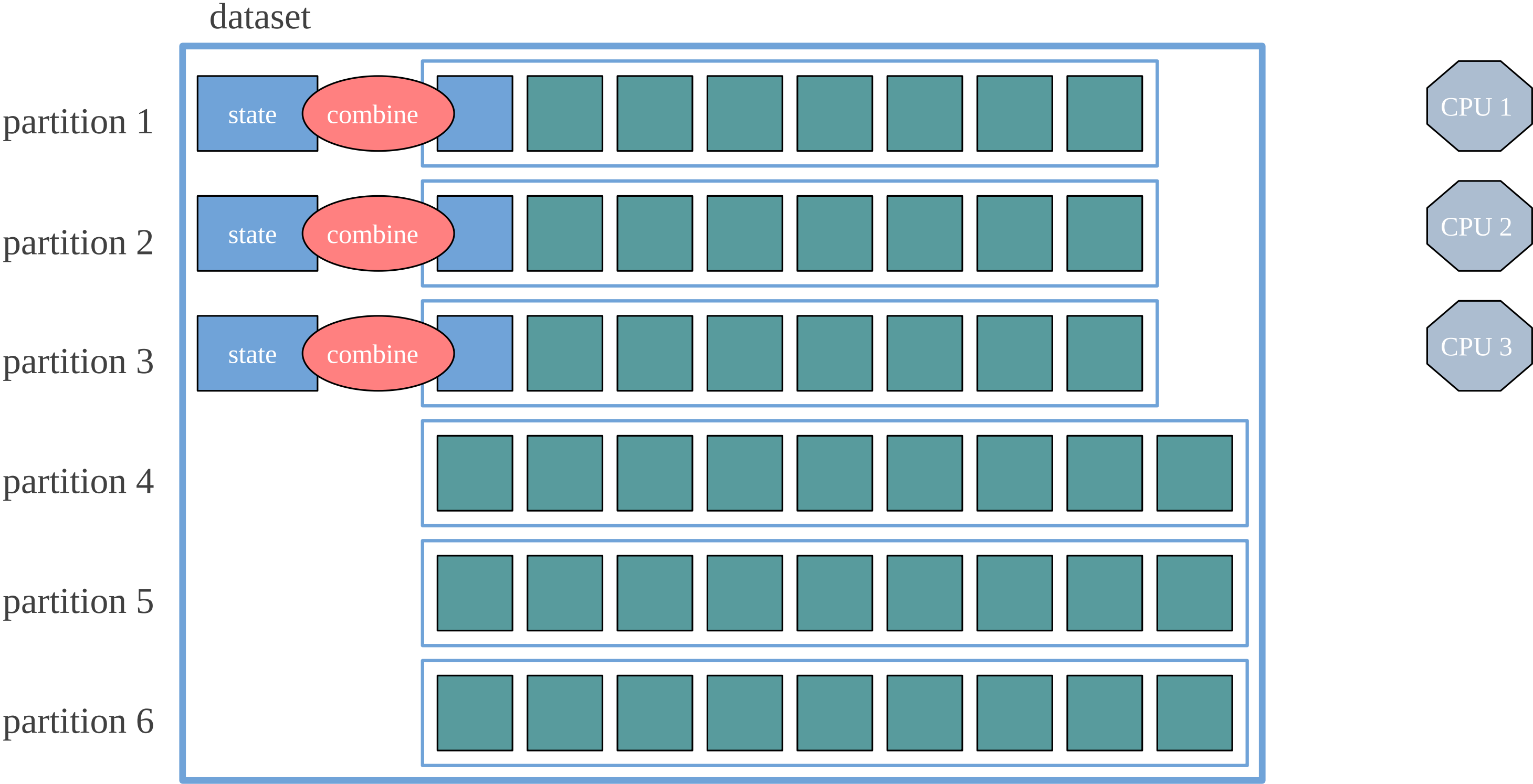
```
(a combine b combine c combine d) combine  
(e combine f)
```

# foldMap in parallel

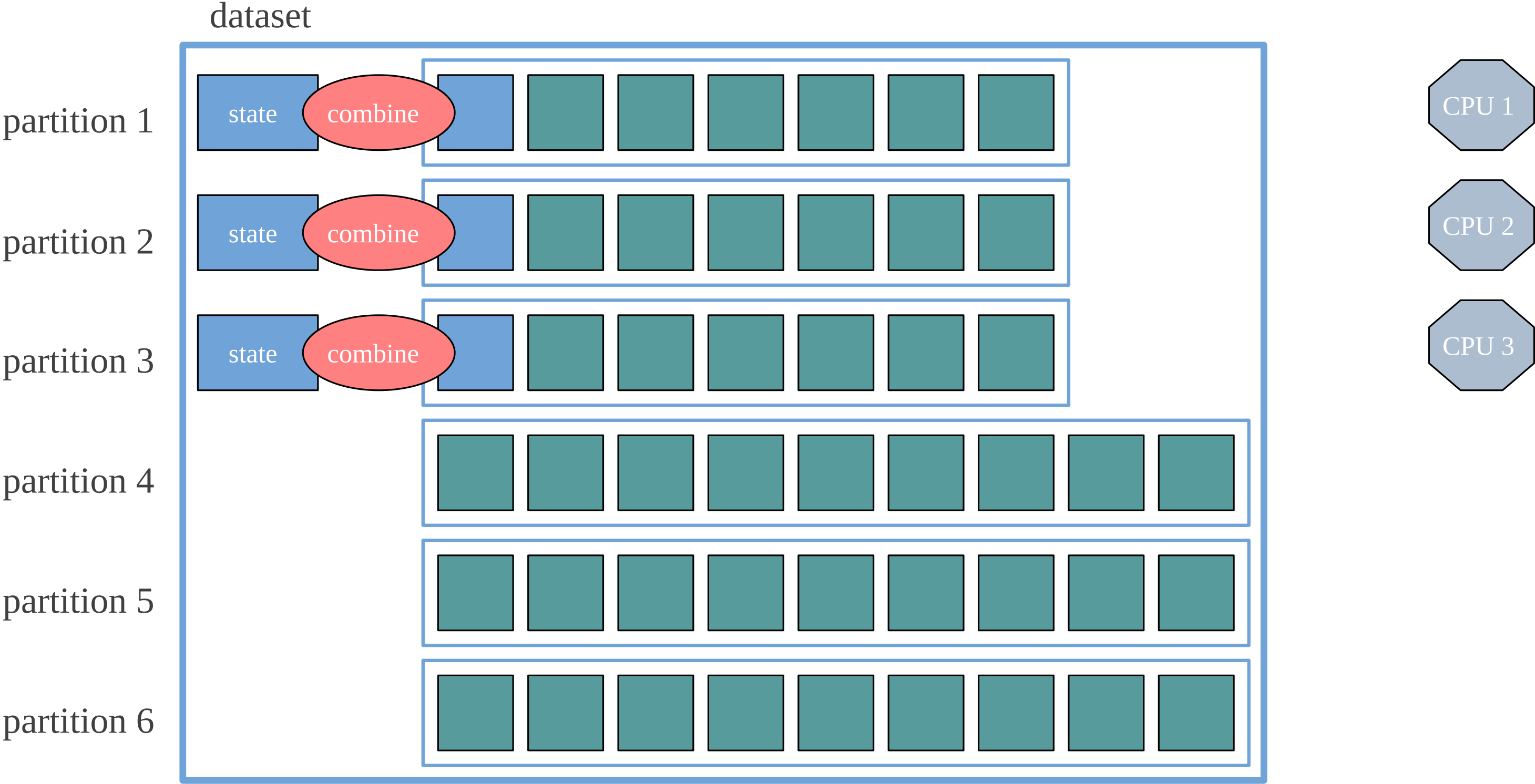




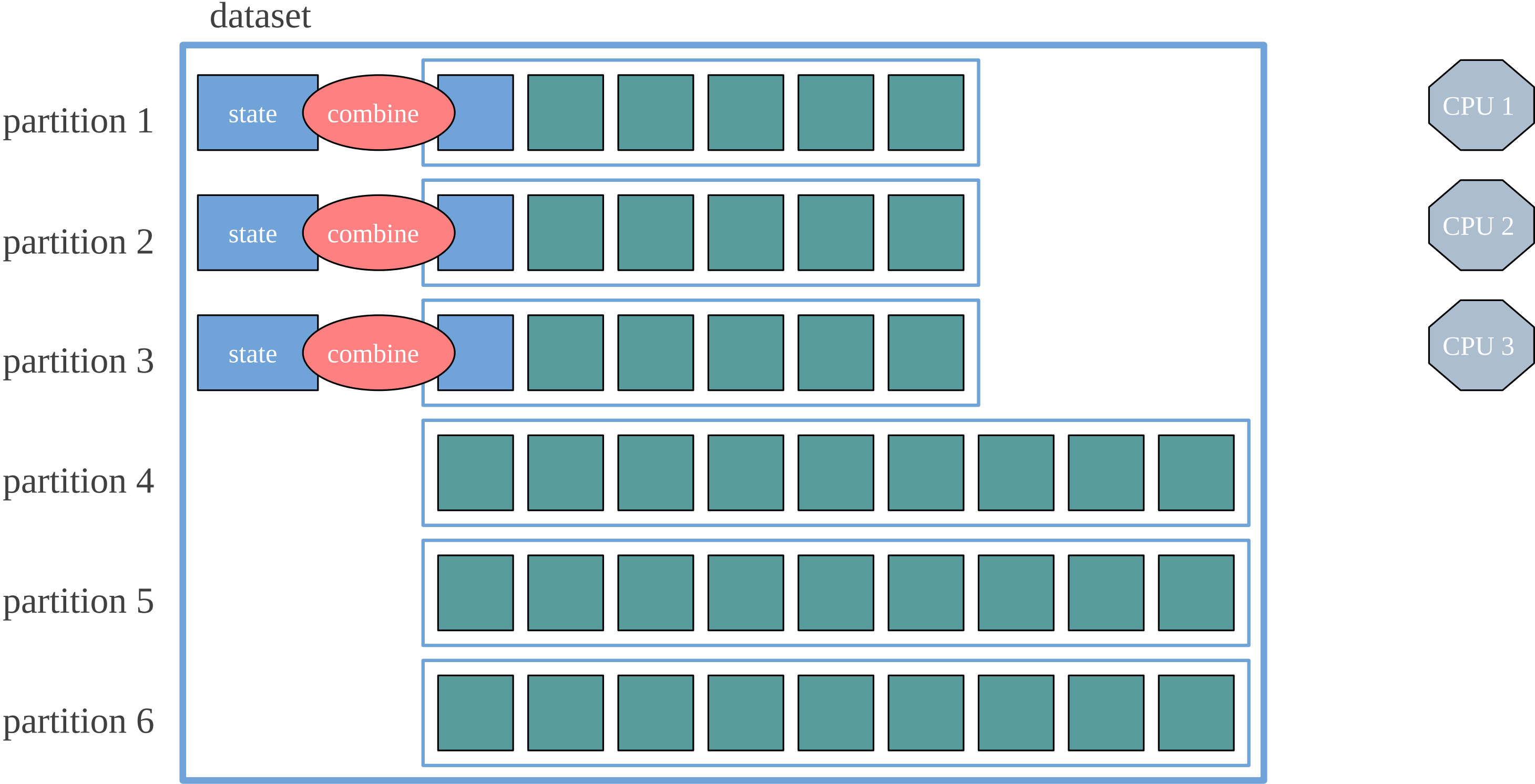
# foldMap in parallel



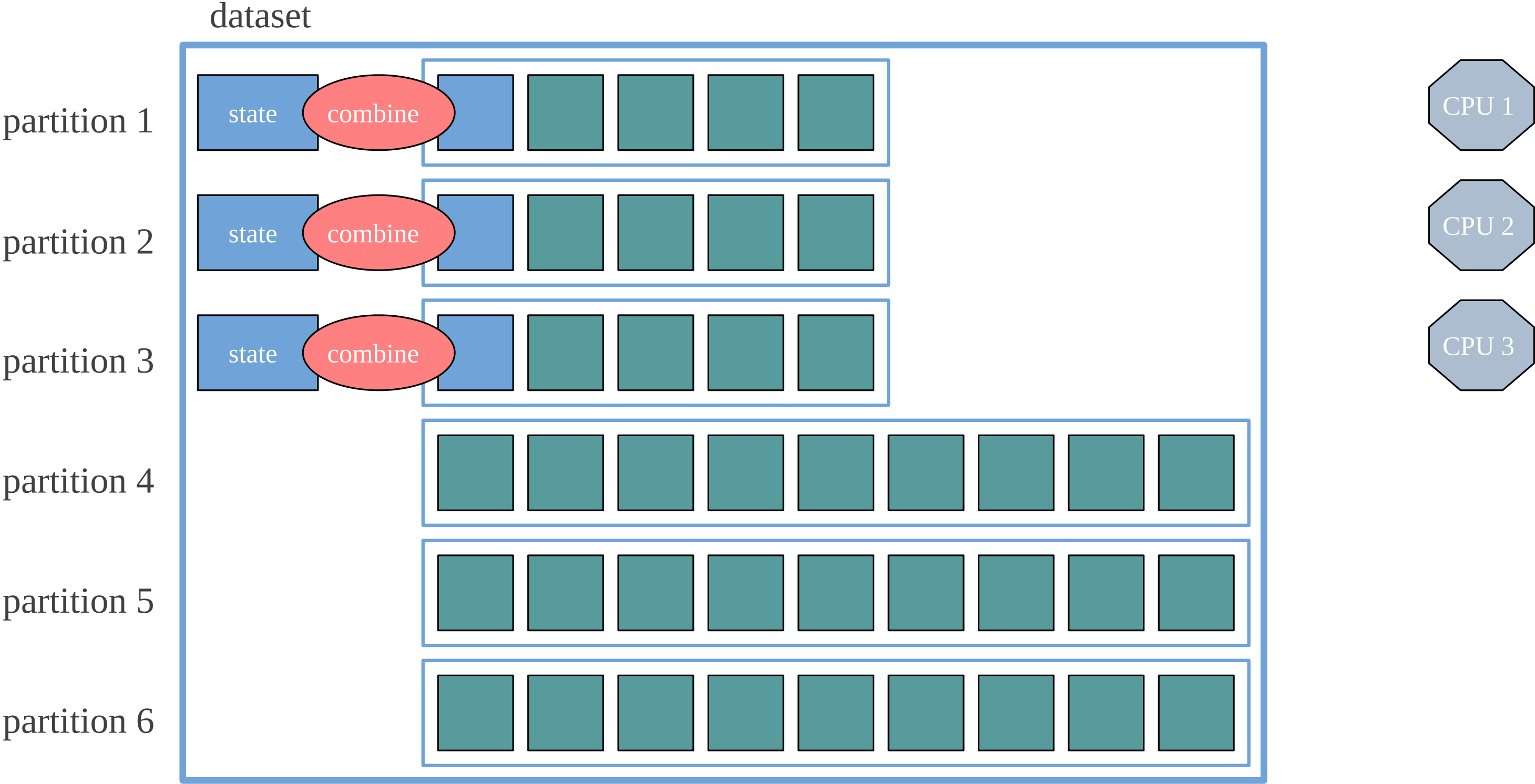
# foldMap in parallel



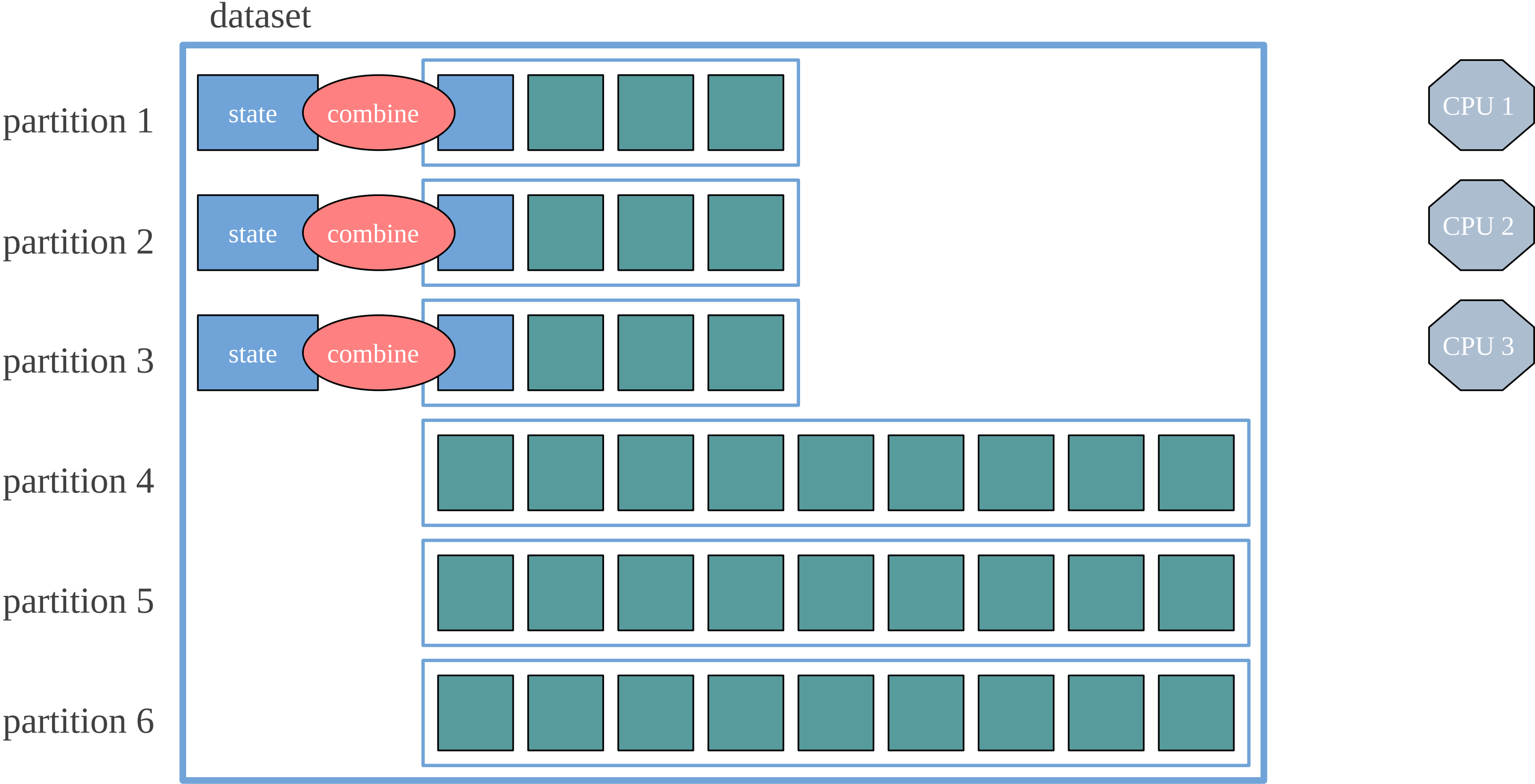
# foldMap in parallel



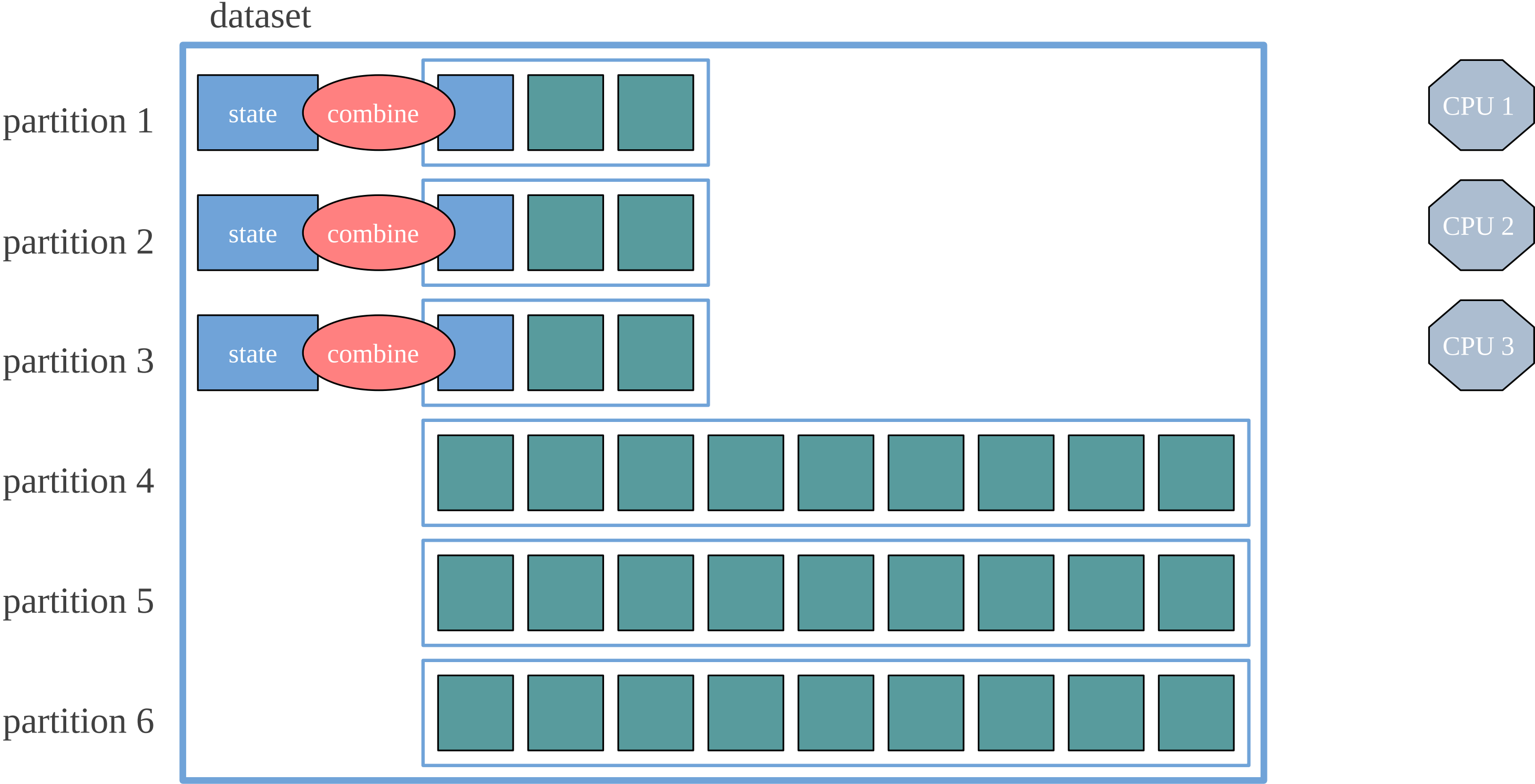
# foldMap in parallel



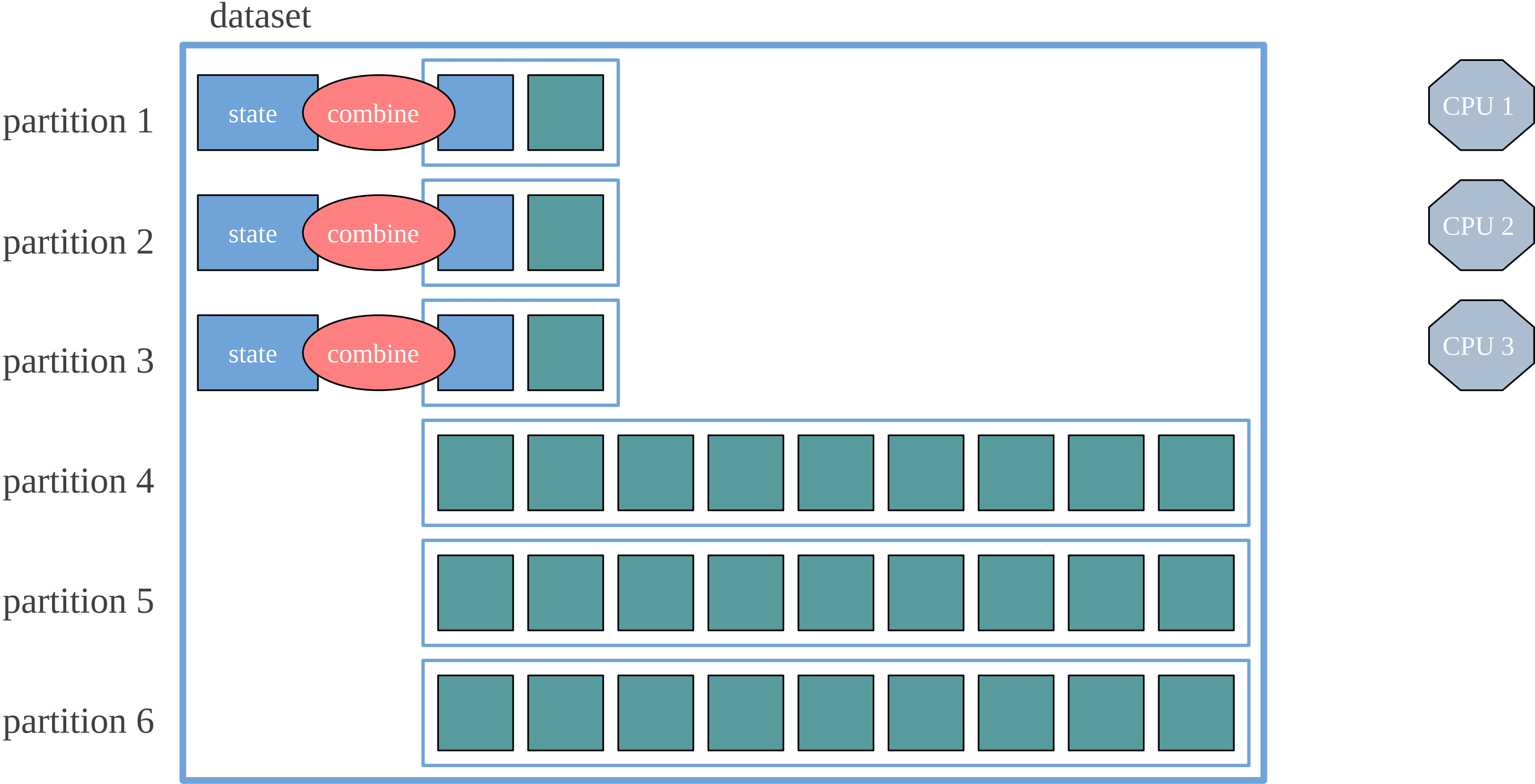
# foldMap in parallel



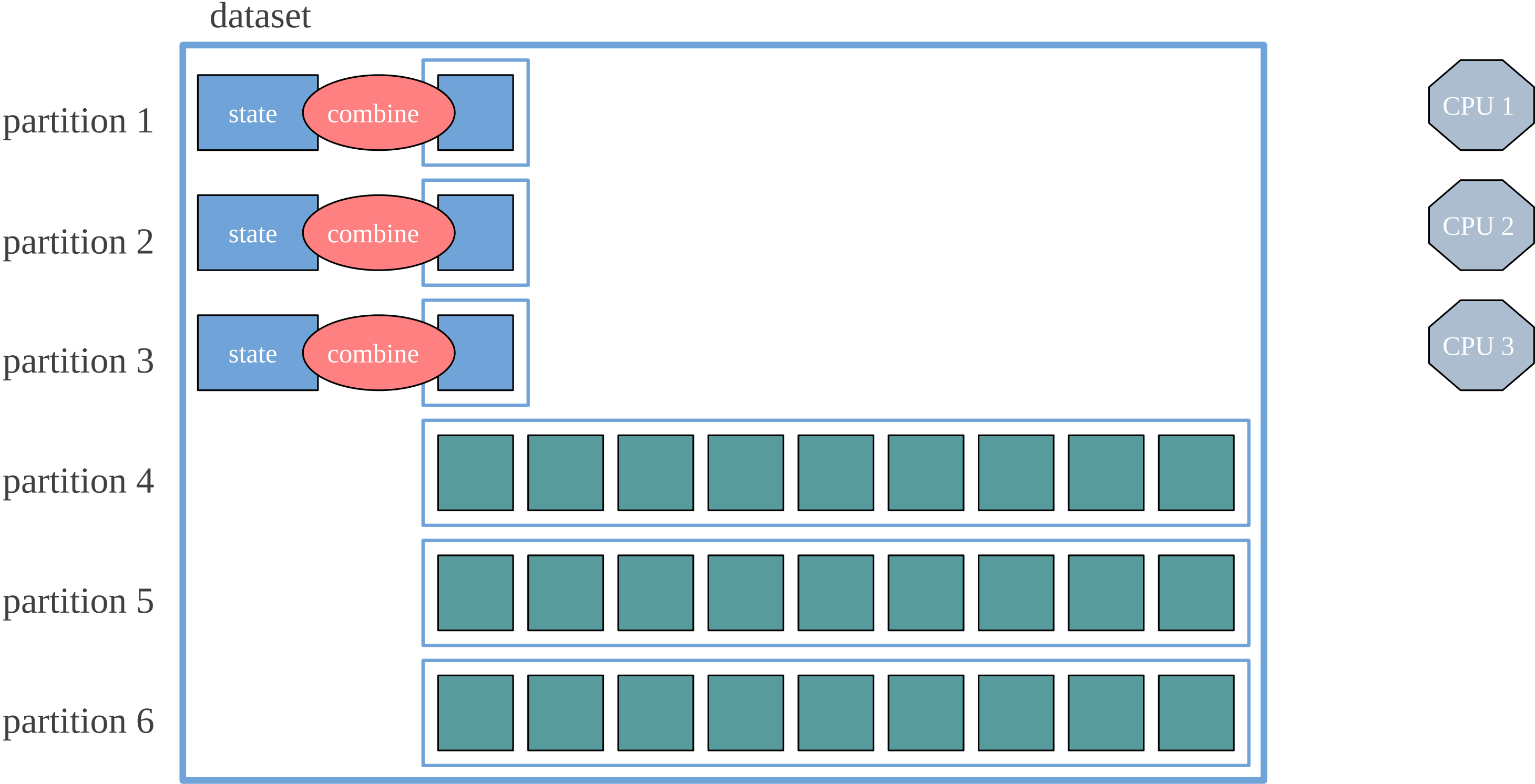
# foldMap in parallel



# foldMap in parallel

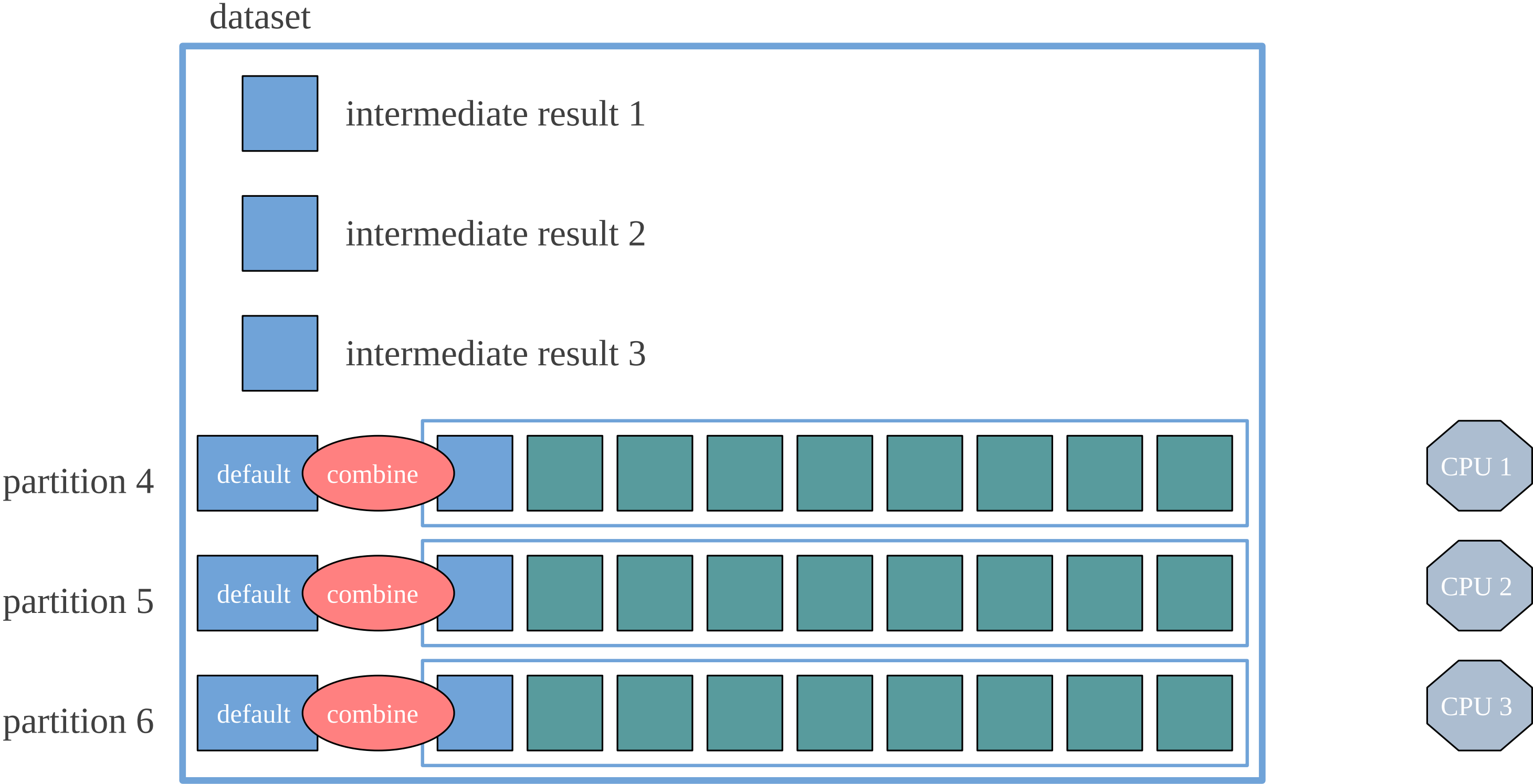


# foldMap in parallel

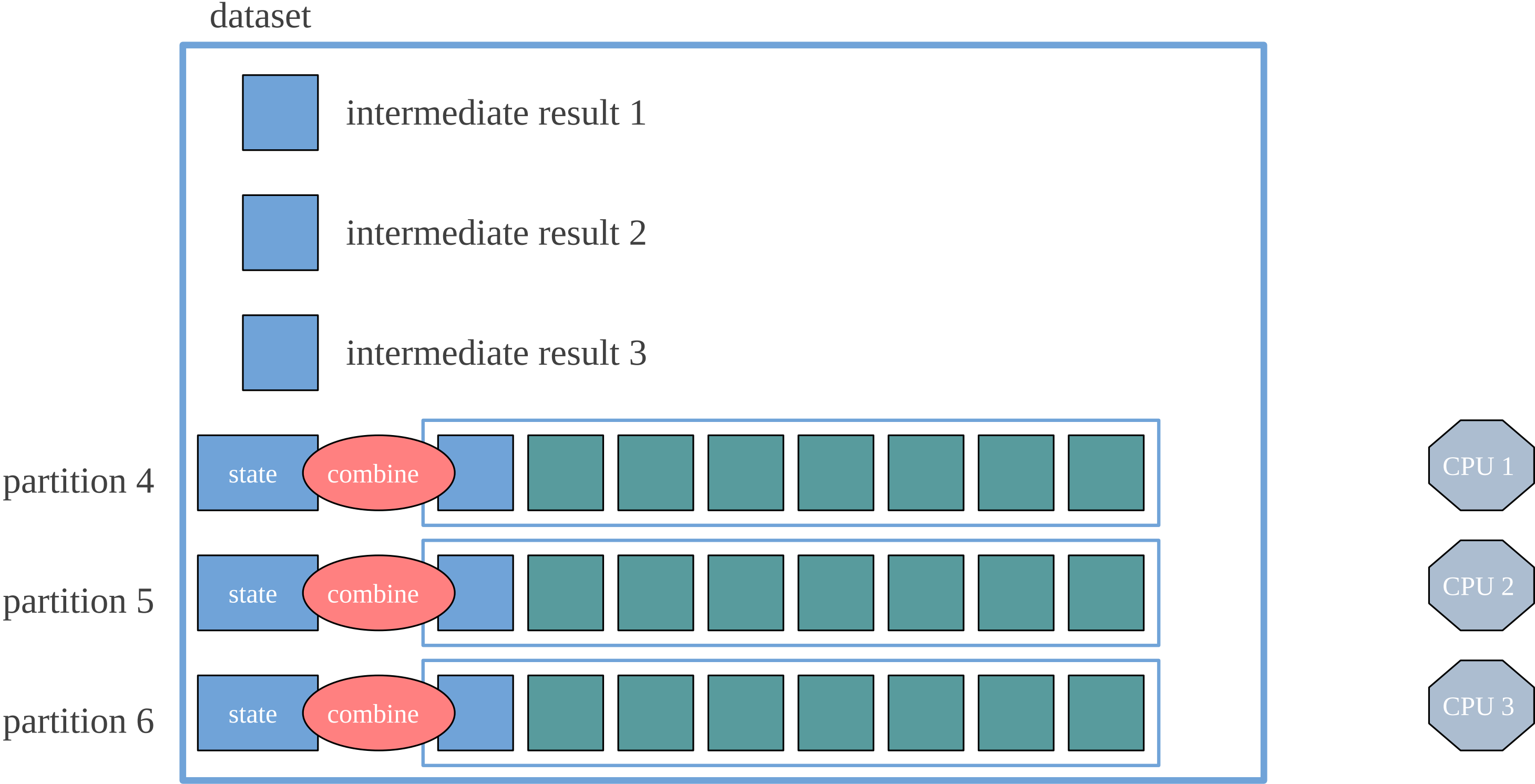




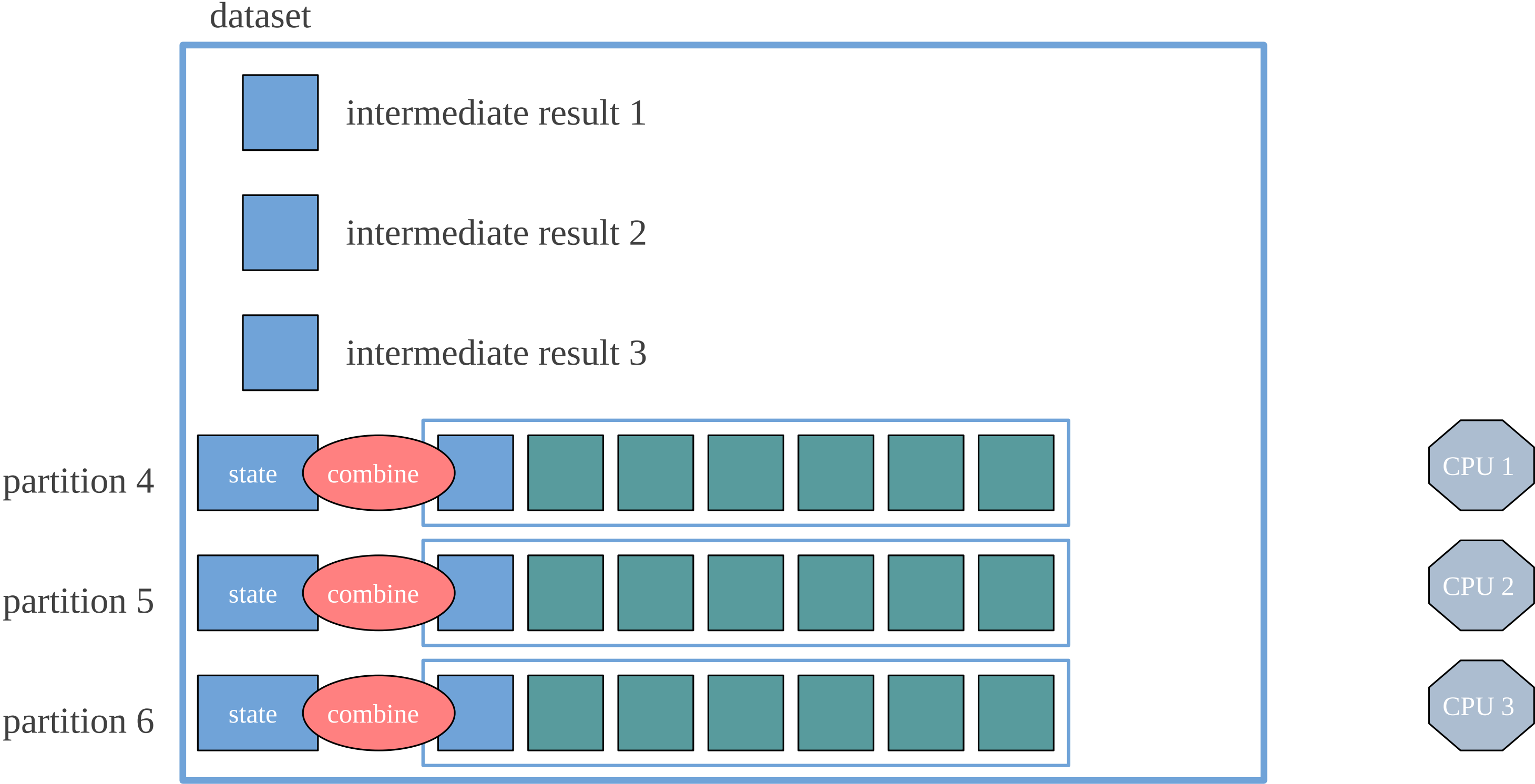
# foldMap in parallel



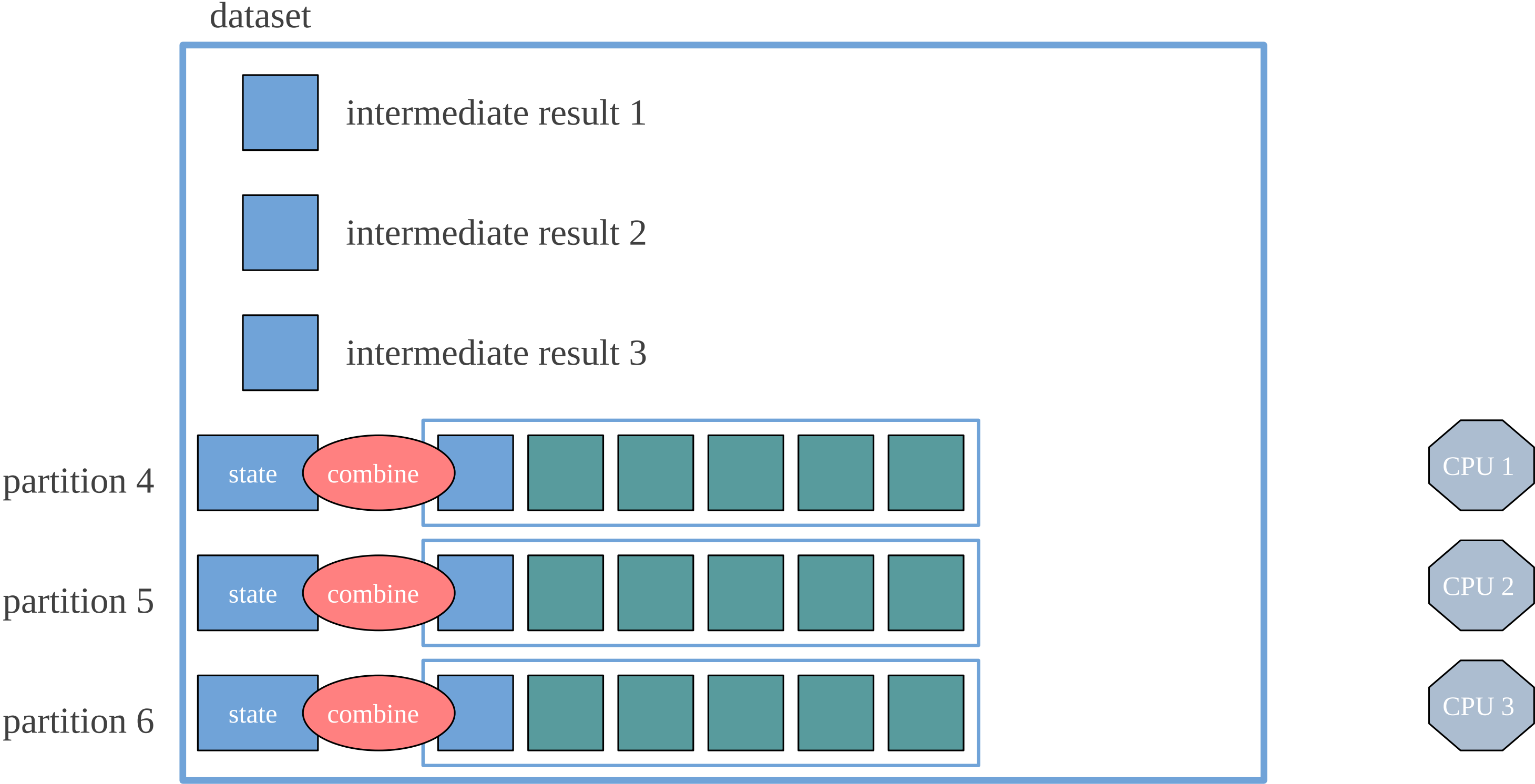
# foldMap in parallel



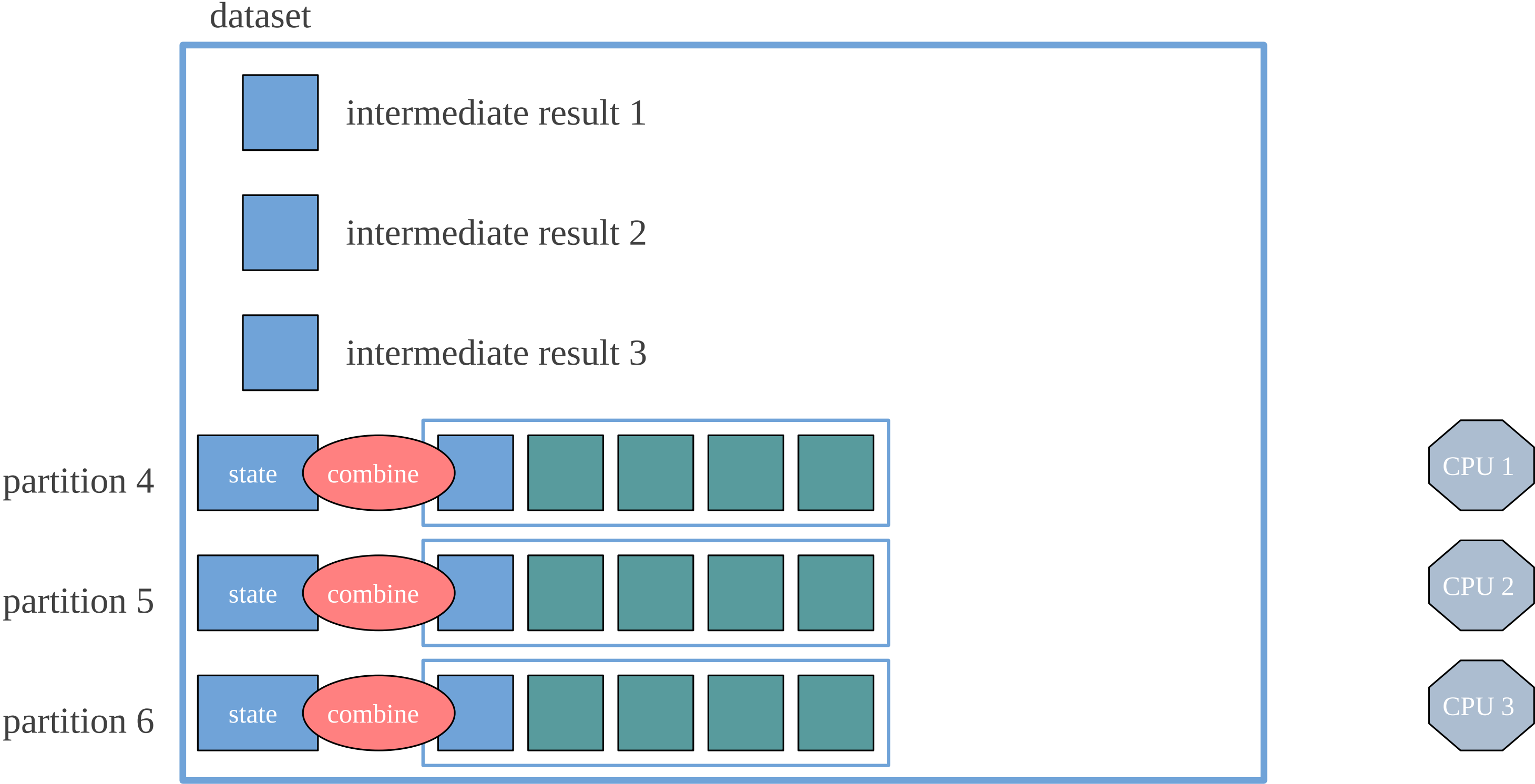
# foldMap in parallel



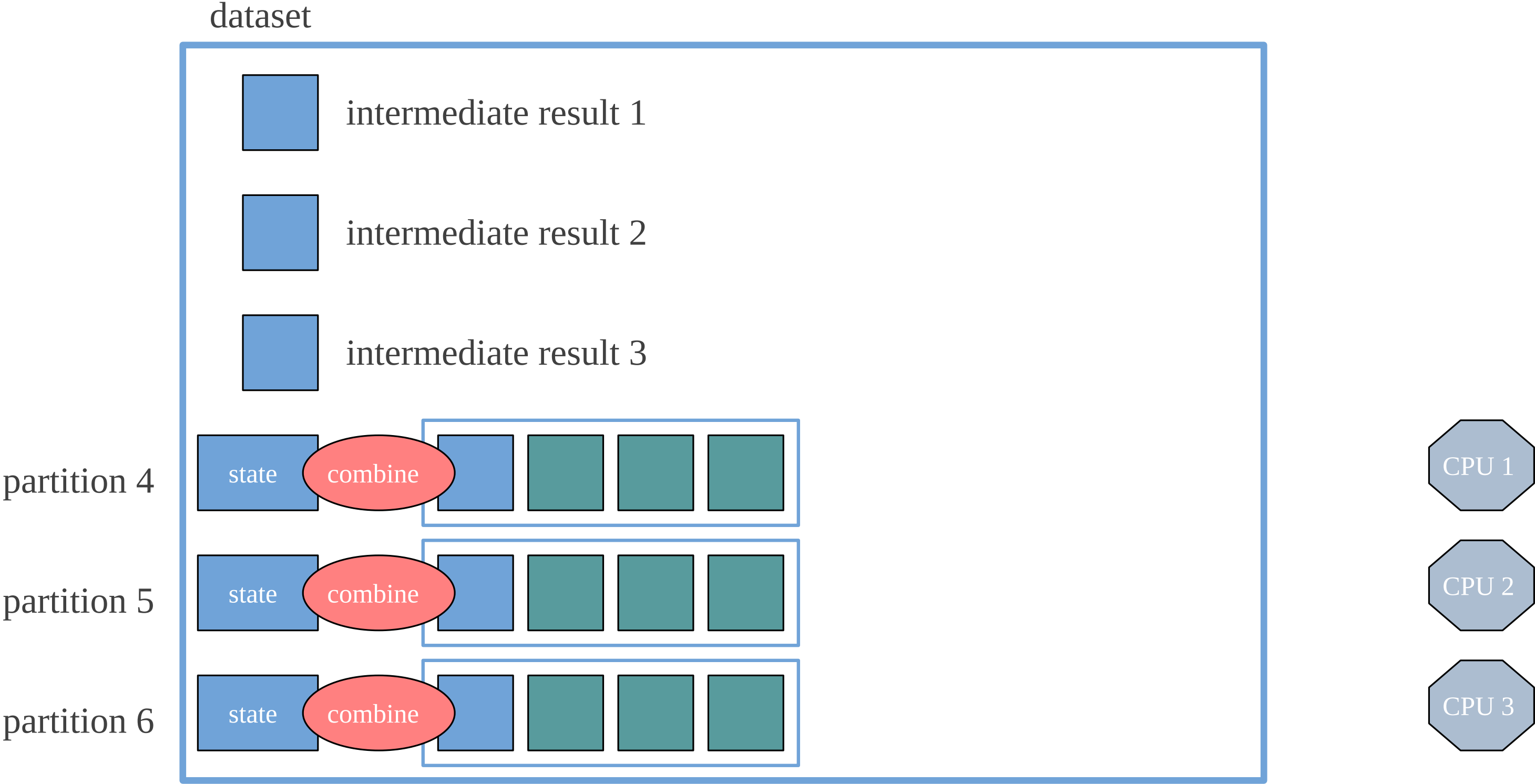
# foldMap in parallel



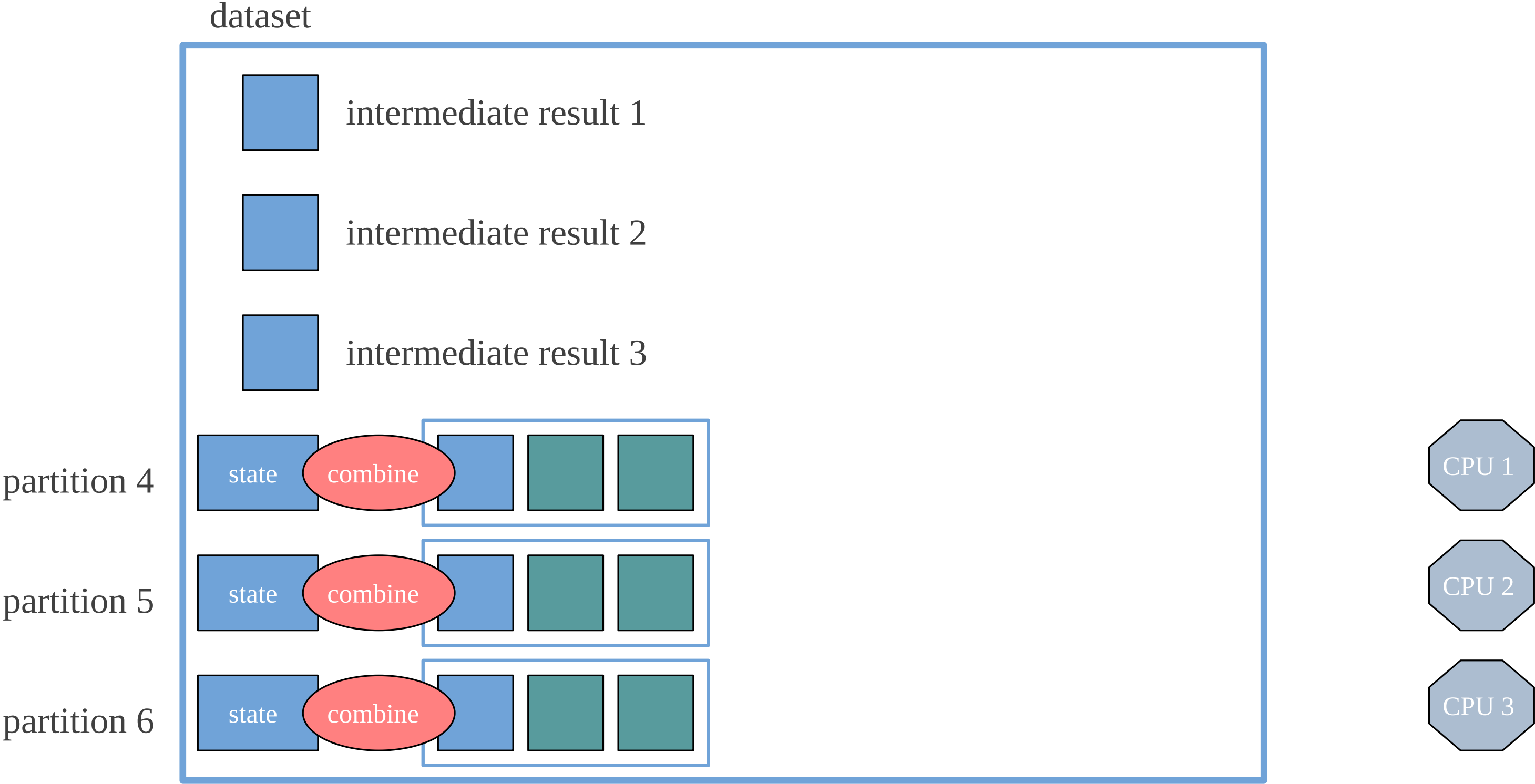
# foldMap in parallel



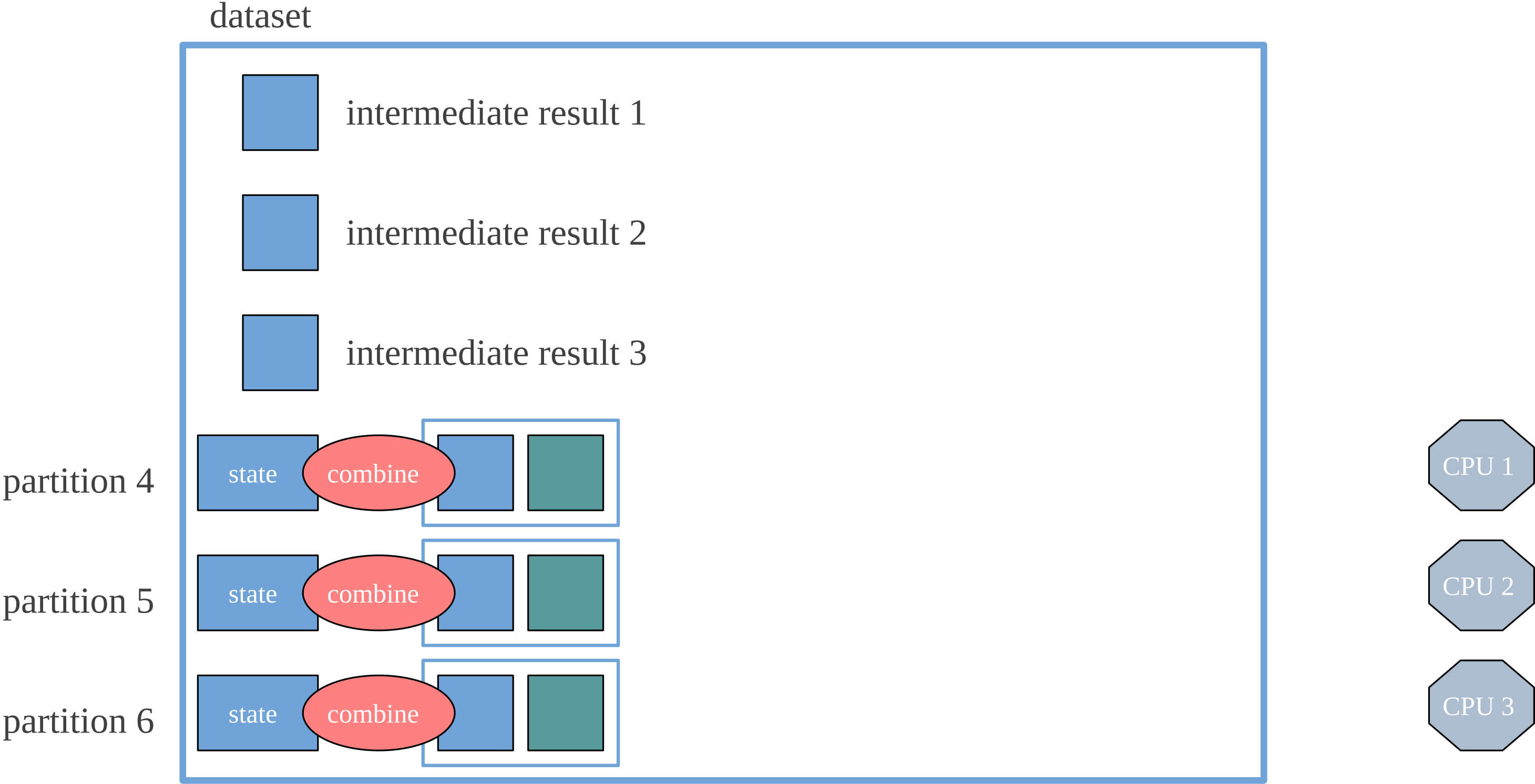
# foldMap in parallel



# foldMap in parallel

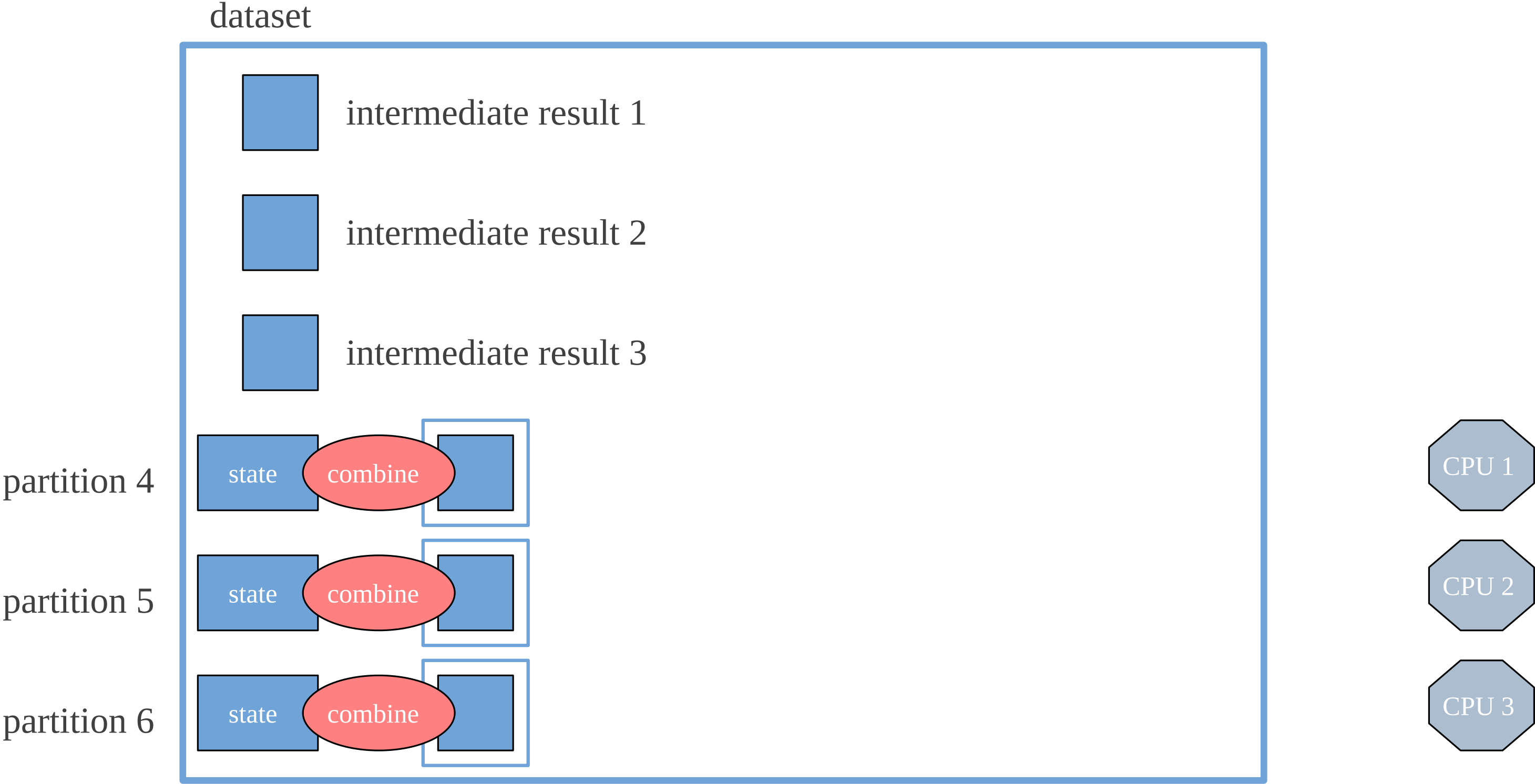


# foldMap in parallel





# foldMap in parallel

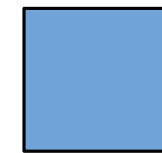


# foldMap in parallel

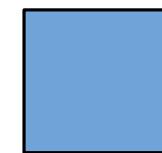
dataset



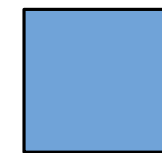
intermediate result 1



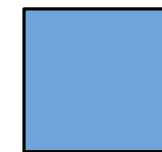
intermediate result 2



intermediate result 3



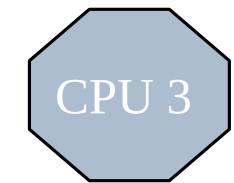
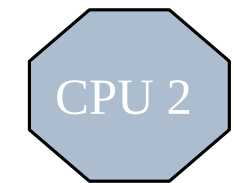
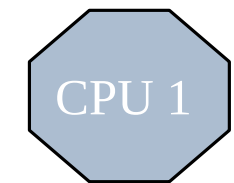
intermediate result 4



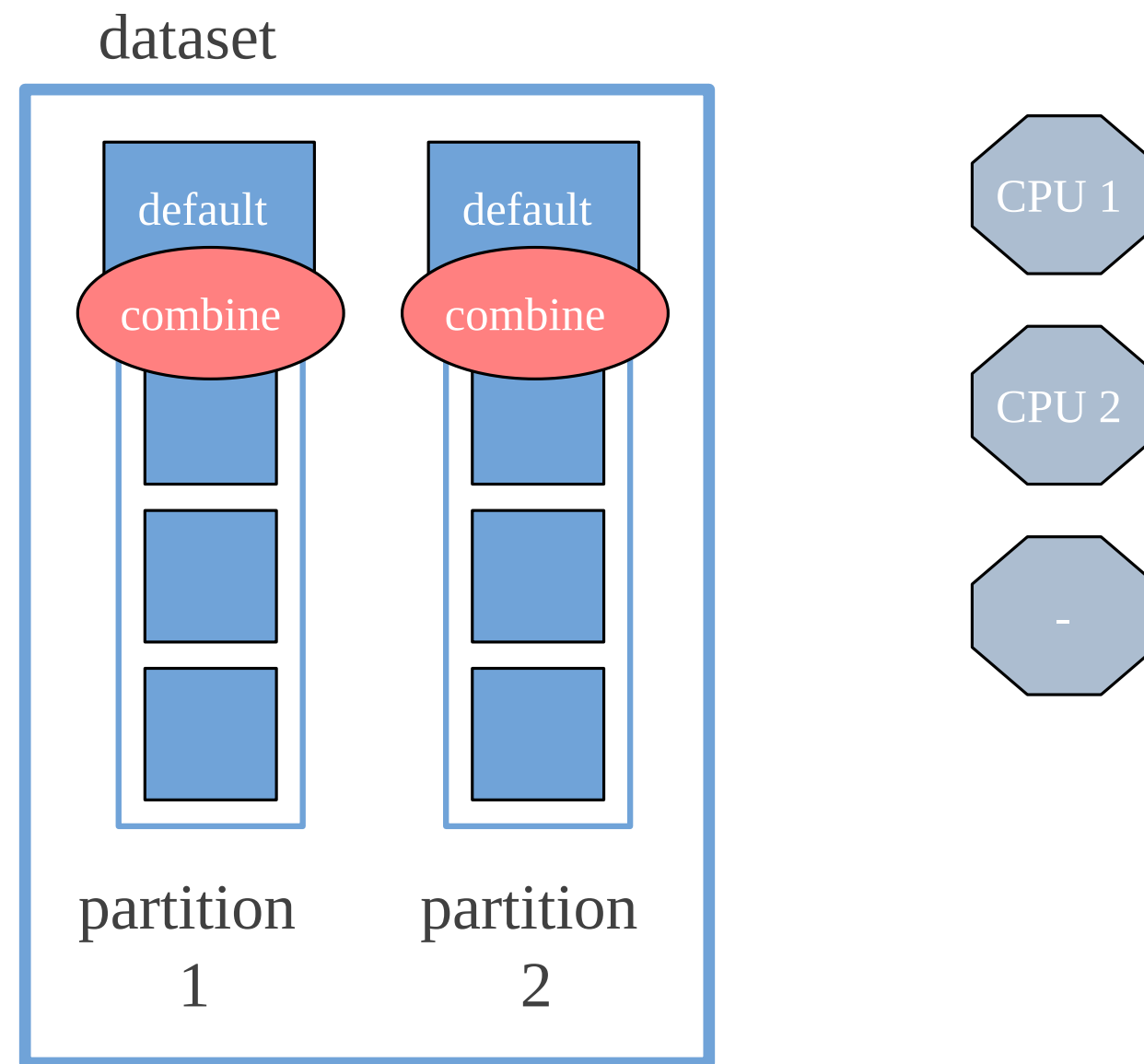
intermediate result 5



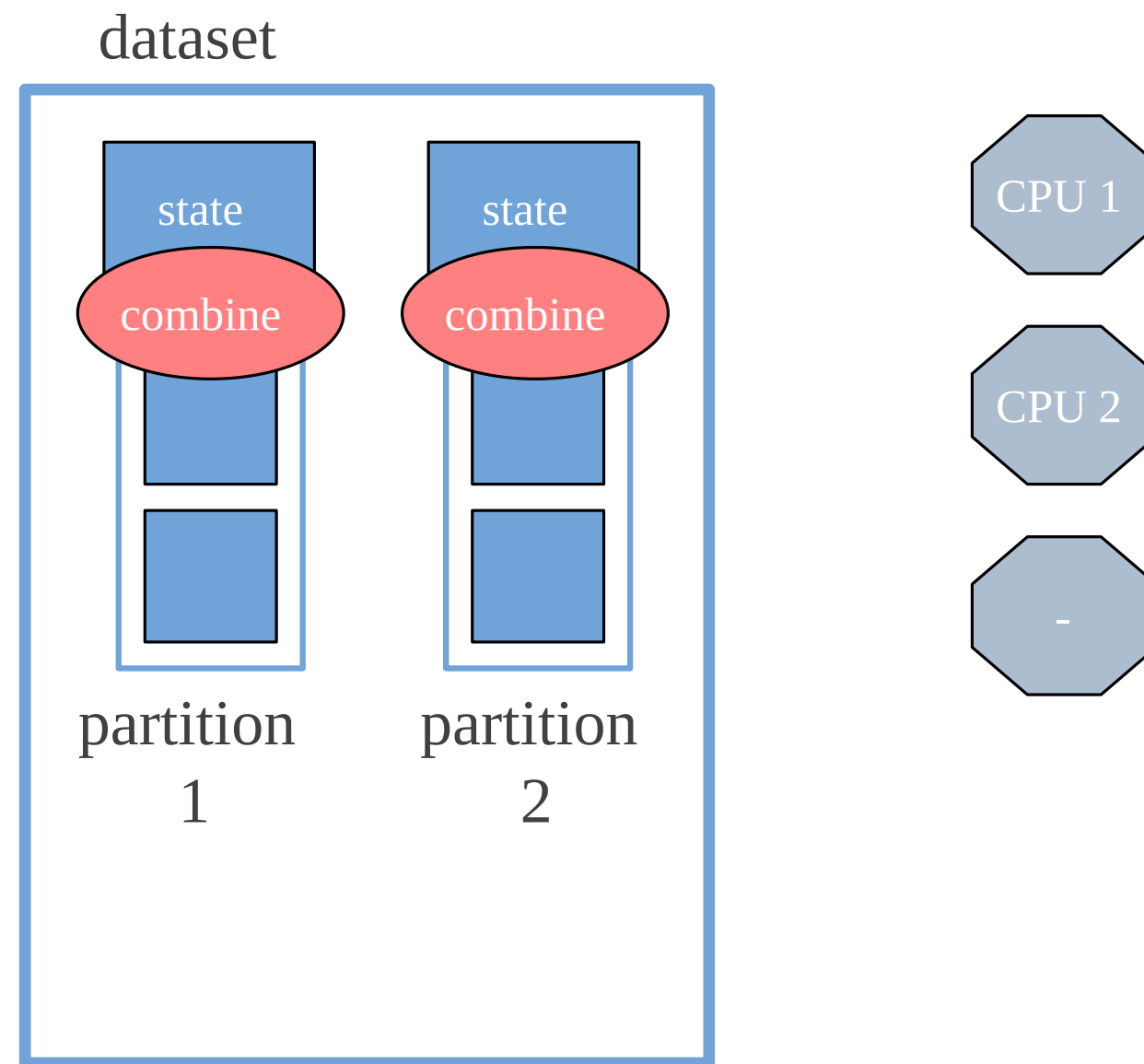
intermediate result 6



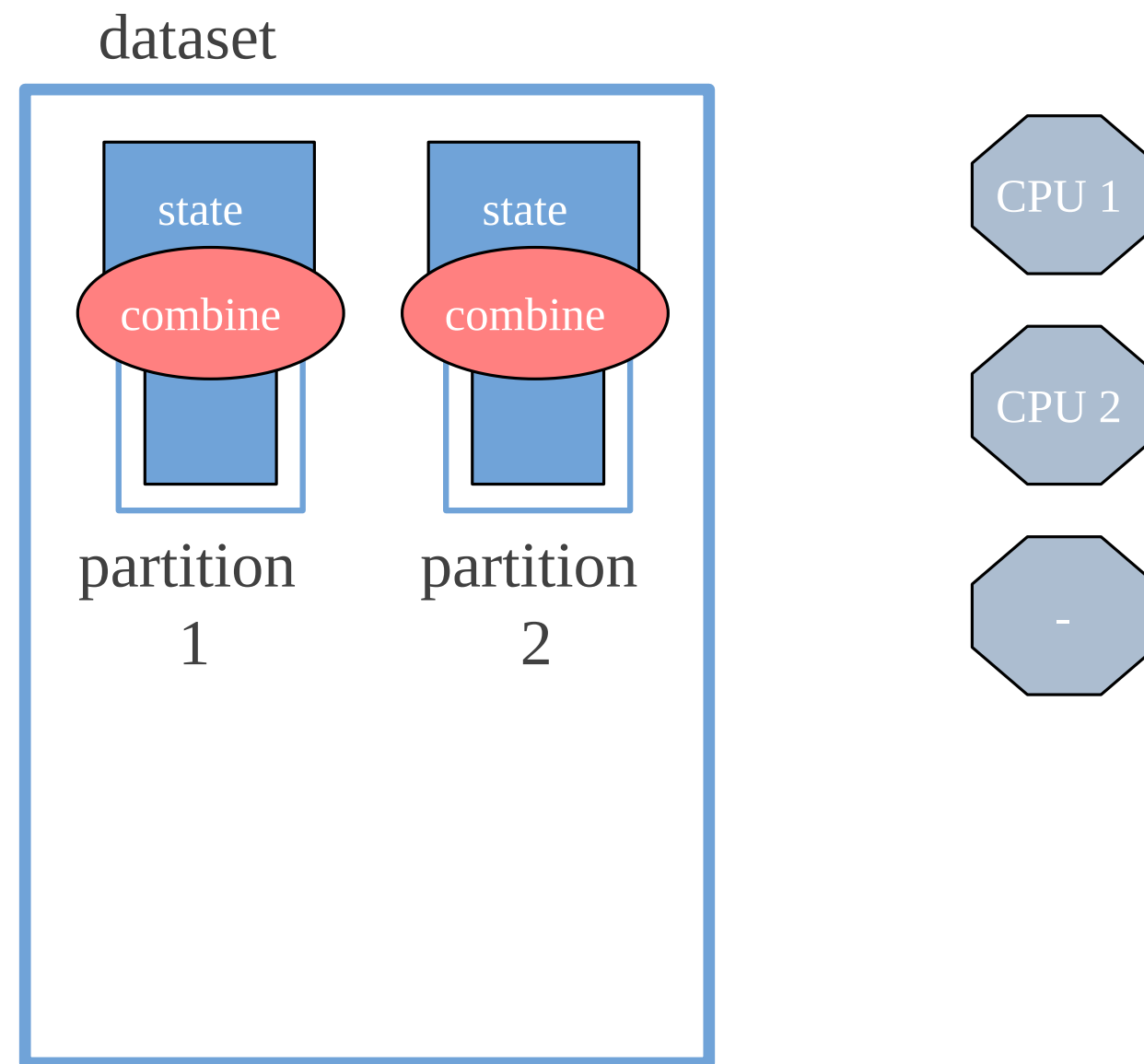
# foldMap intermediate results in parallel



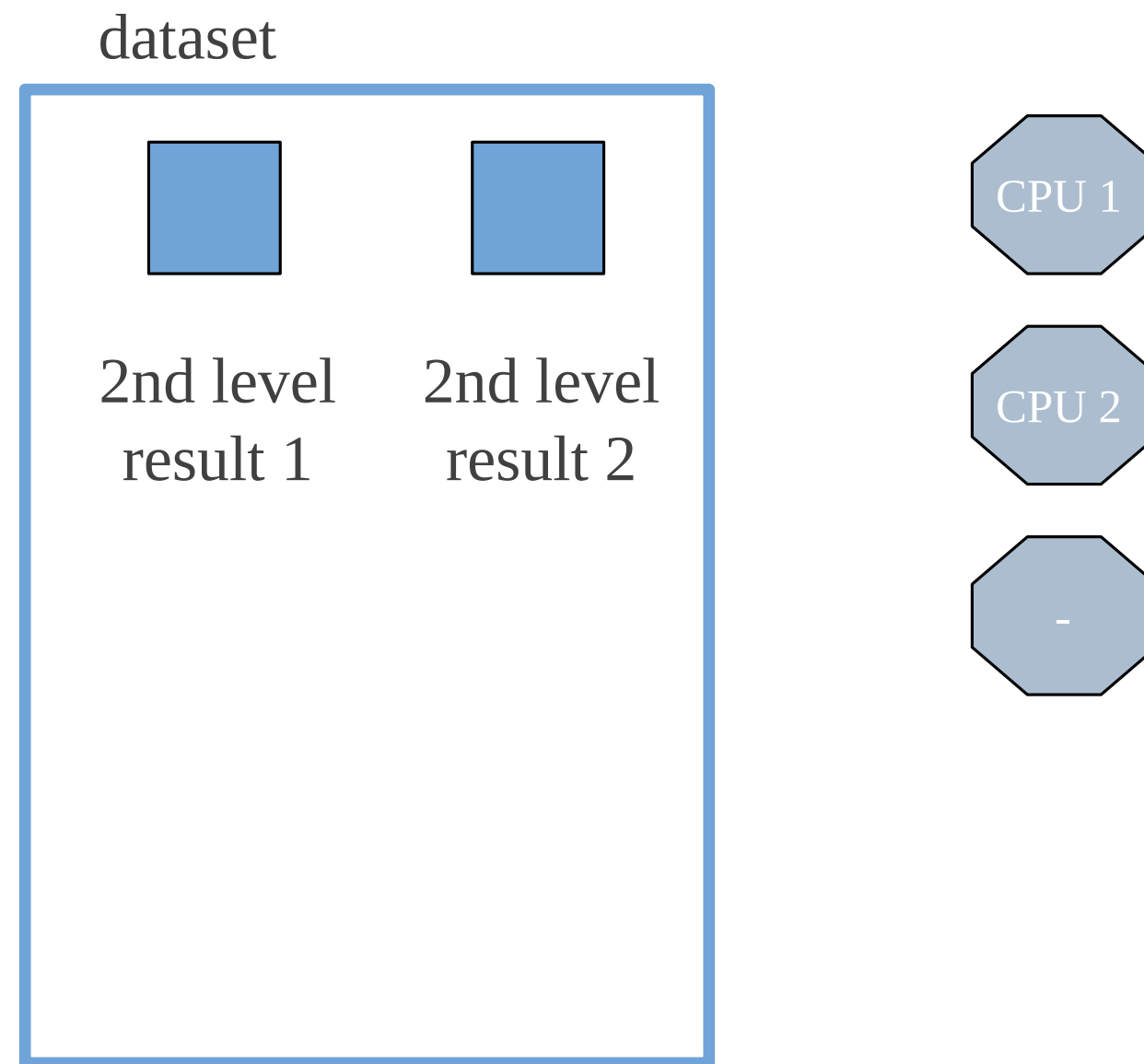
# foldMap intermediate results in parallel



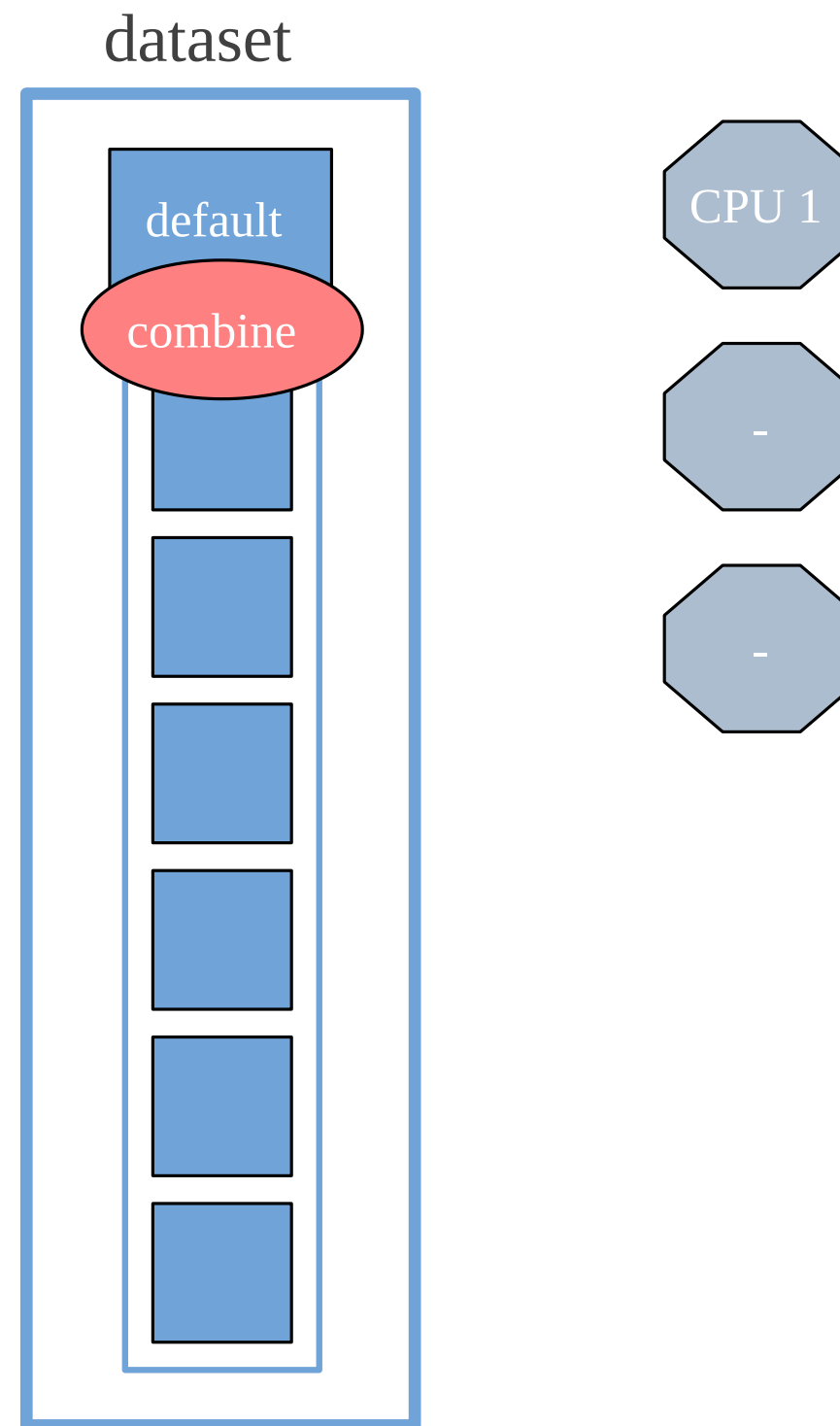
# foldMap intermediate results in parallel



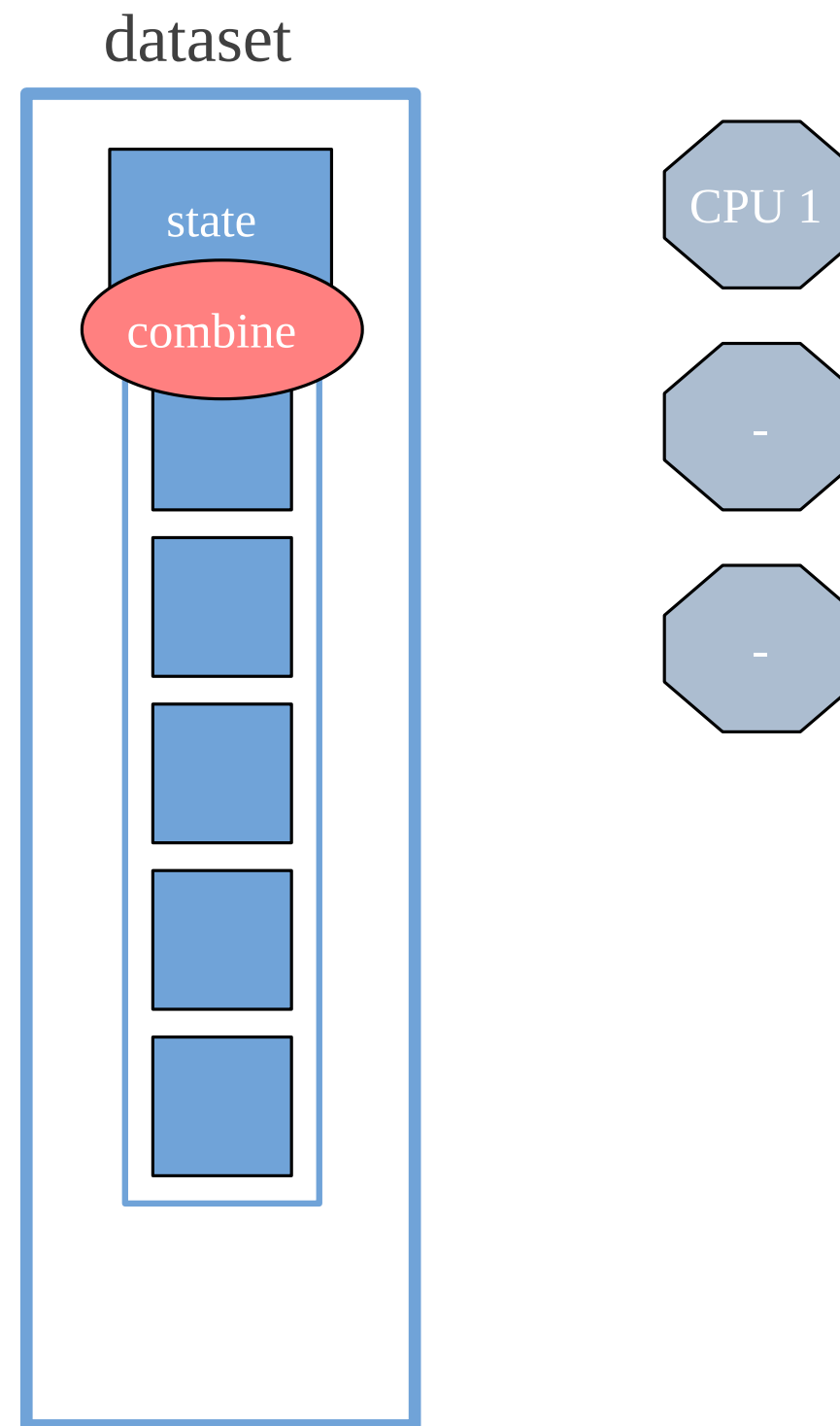
# foldMap intermediate results in parallel



# foldMap intermediate results sequentially

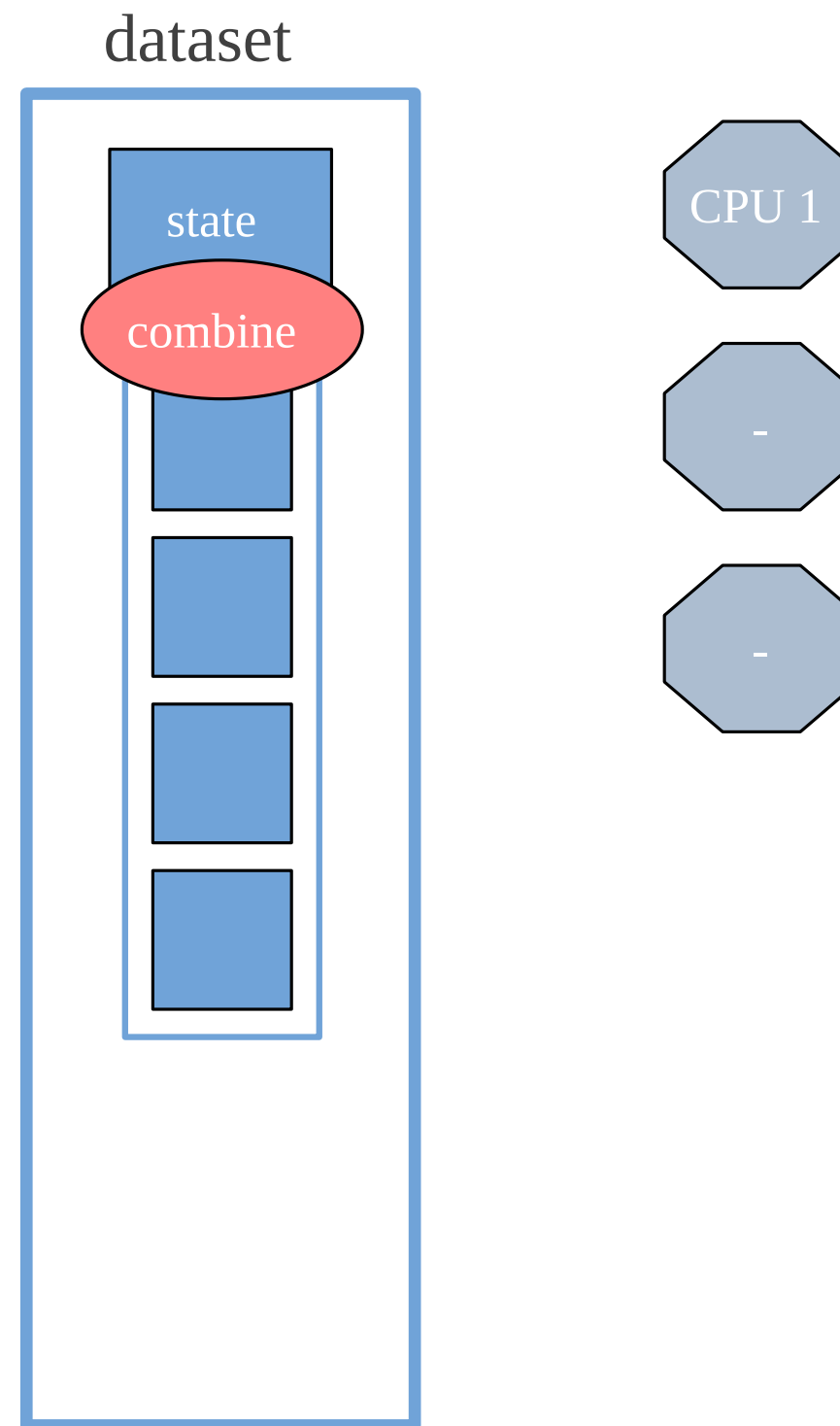


# foldMap intermediate results sequentially

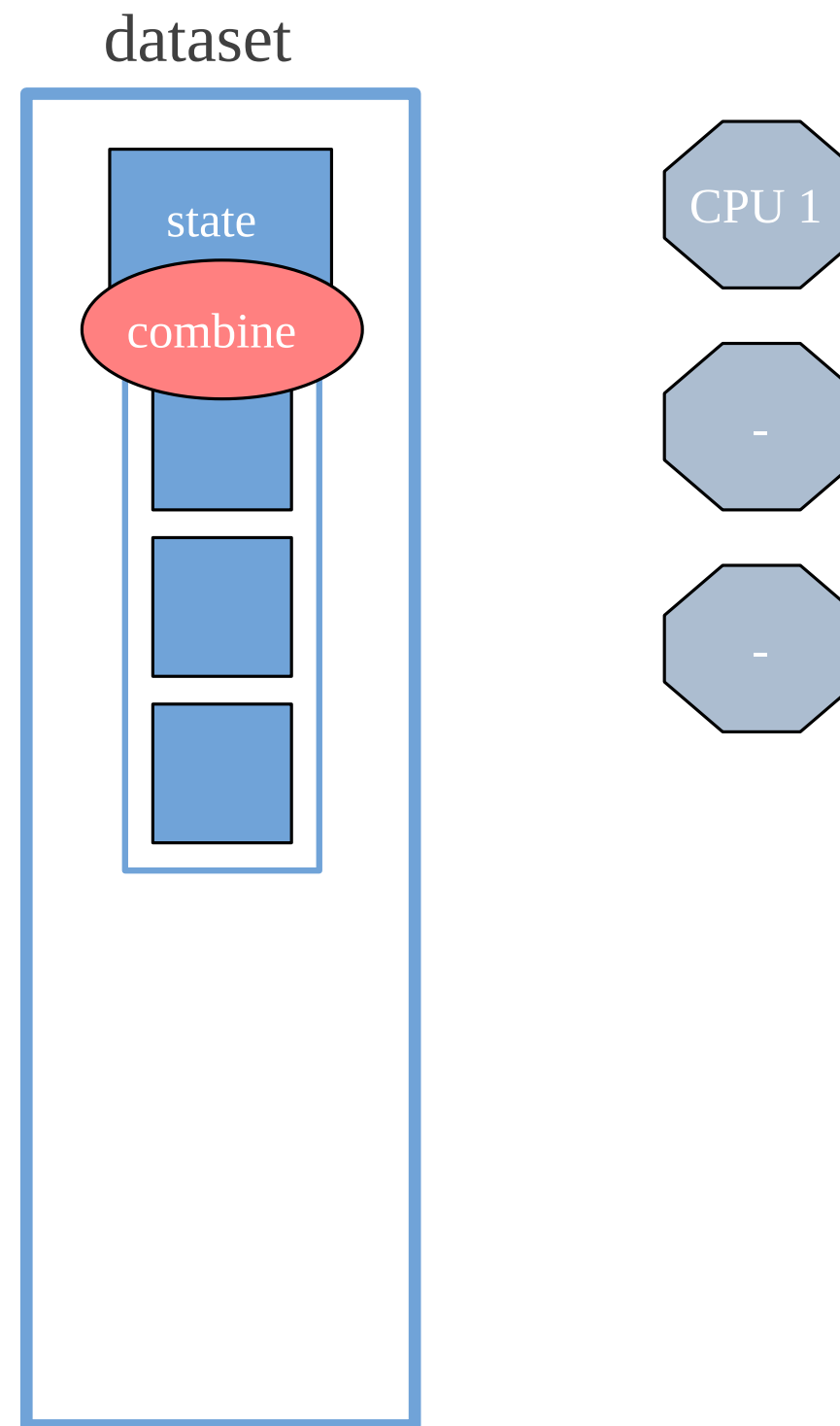




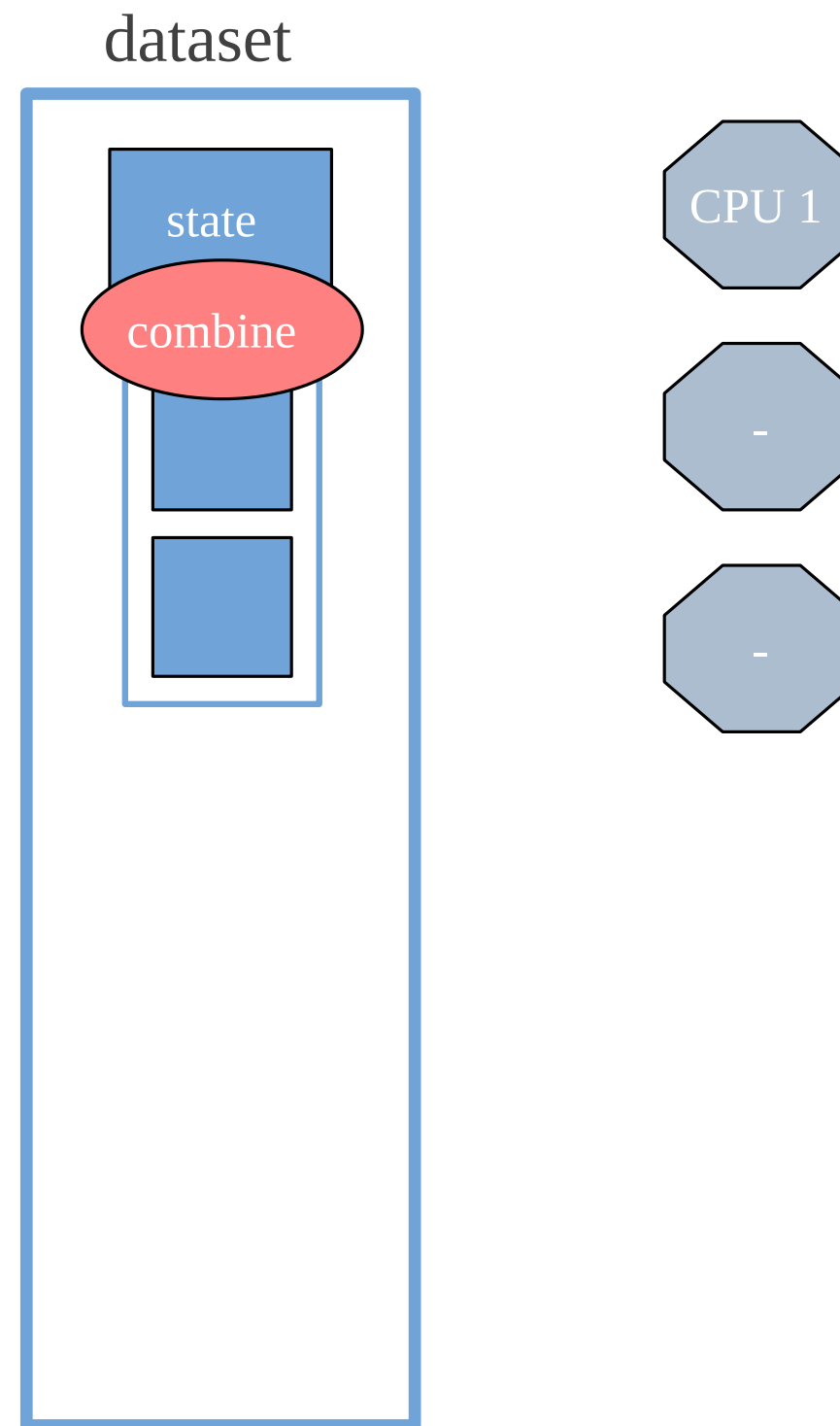
# foldMap intermediate results sequentially



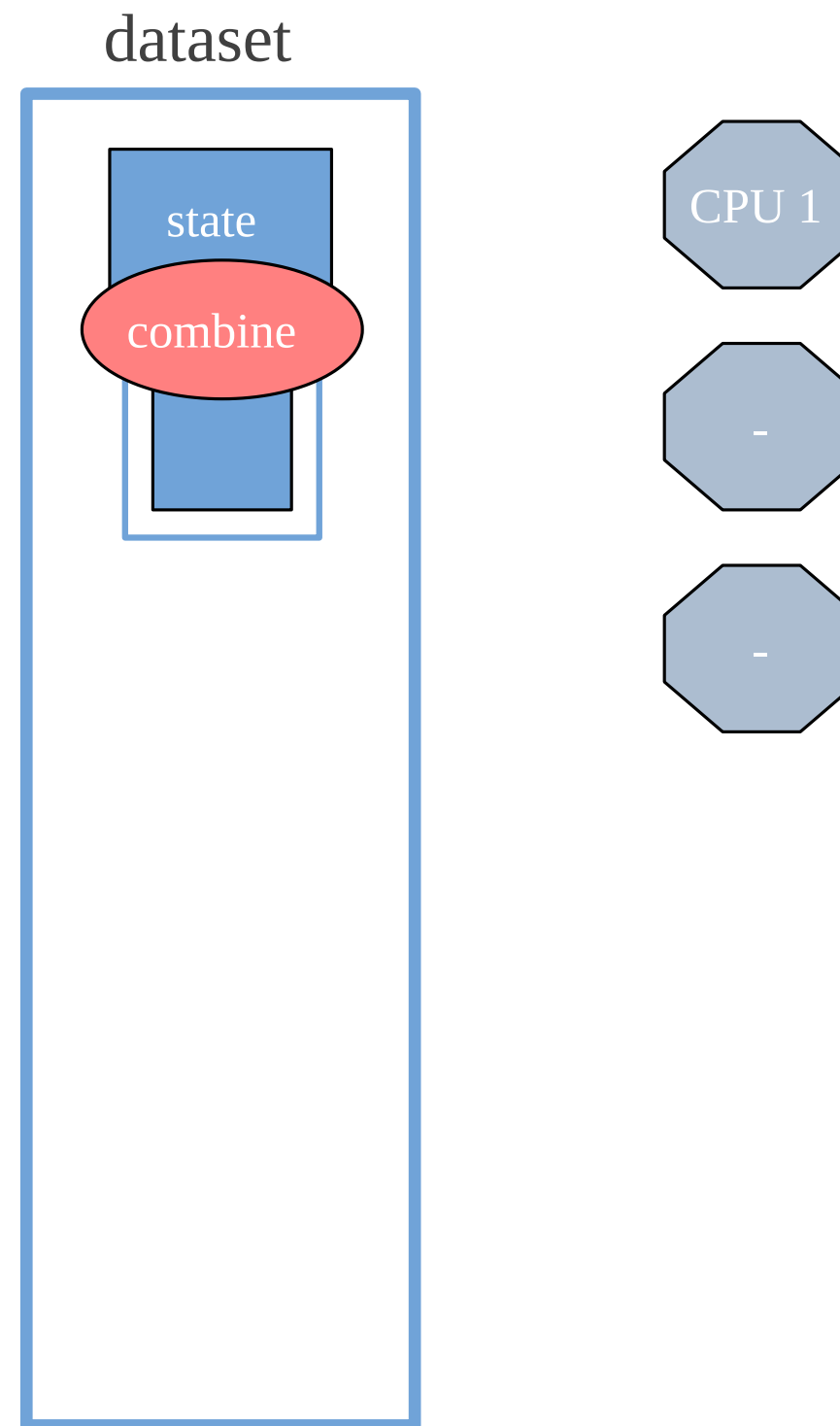
# foldMap intermediate results sequentially



# foldMap intermediate results sequentially

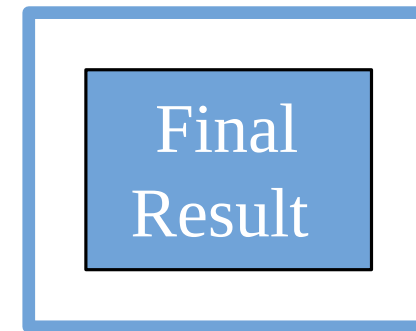


# foldMap intermediate results sequentially



# foldMap

dataset



# Thread

```
def createThread(n: Int): Thread = new Thread {  
  override def run(): Unit =  
    println(s"Thread ${n}")  
}  
  
val threads = 1.to(4).map(createThread)
```

```
threads.foreach(_.start())  
// Thread 1  
// Thread 3  
// Thread 2  
// Thread 4
```

# Executor and Runnable

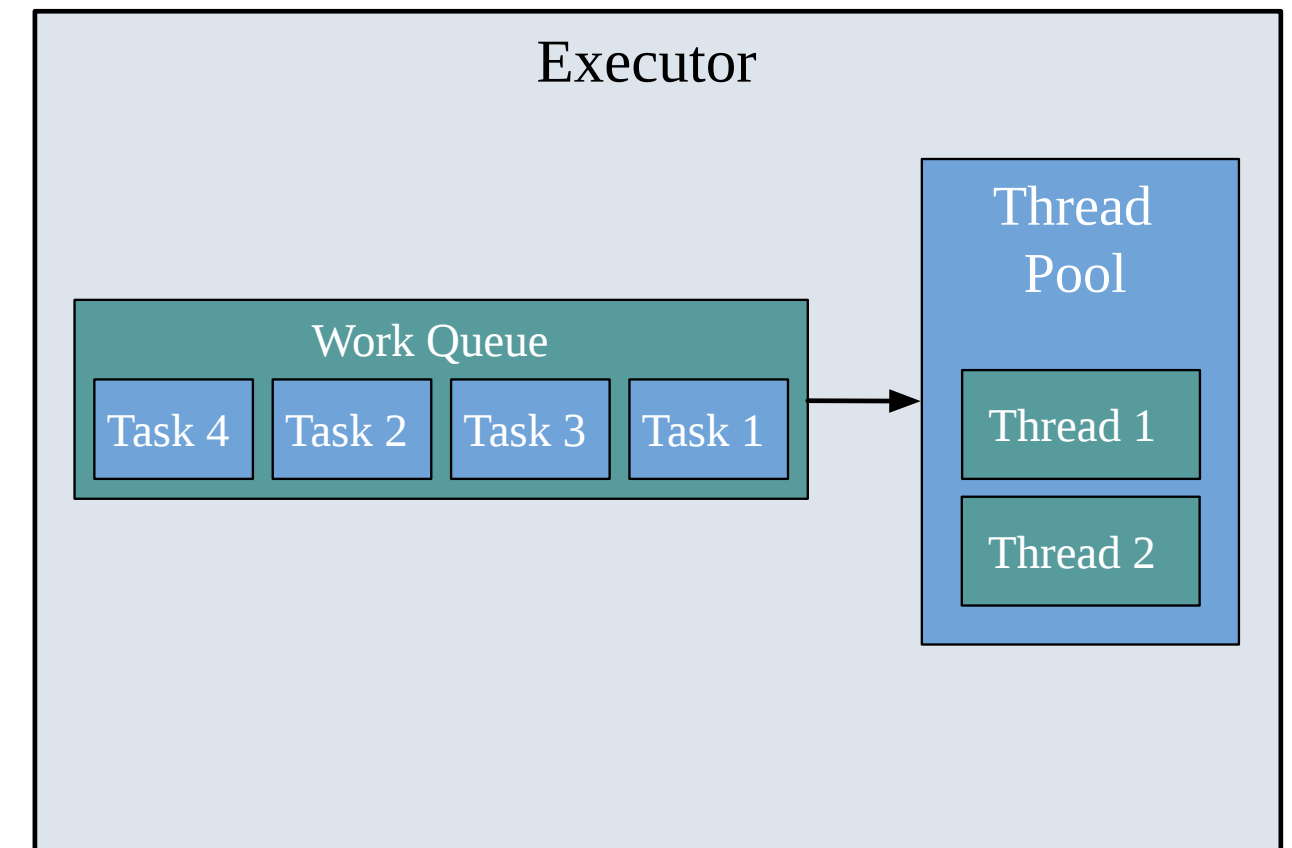
```
import java.util.concurrent.Executors

val fixedPool = Executors.newFixedThreadPool(2)

def createRunnable(n: Int): Runnable =
  new Runnable {
    def run(): Unit =
      println(s"Runnable ${n}")
  }

val runnables = 1.to(4).map(createRunnable)
```

```
runnables.foreach(fixedPool.submit)
// Runnable 1
// Runnable 3
// Runnable 2
// Runnable 4
```



# Executor and Runnable

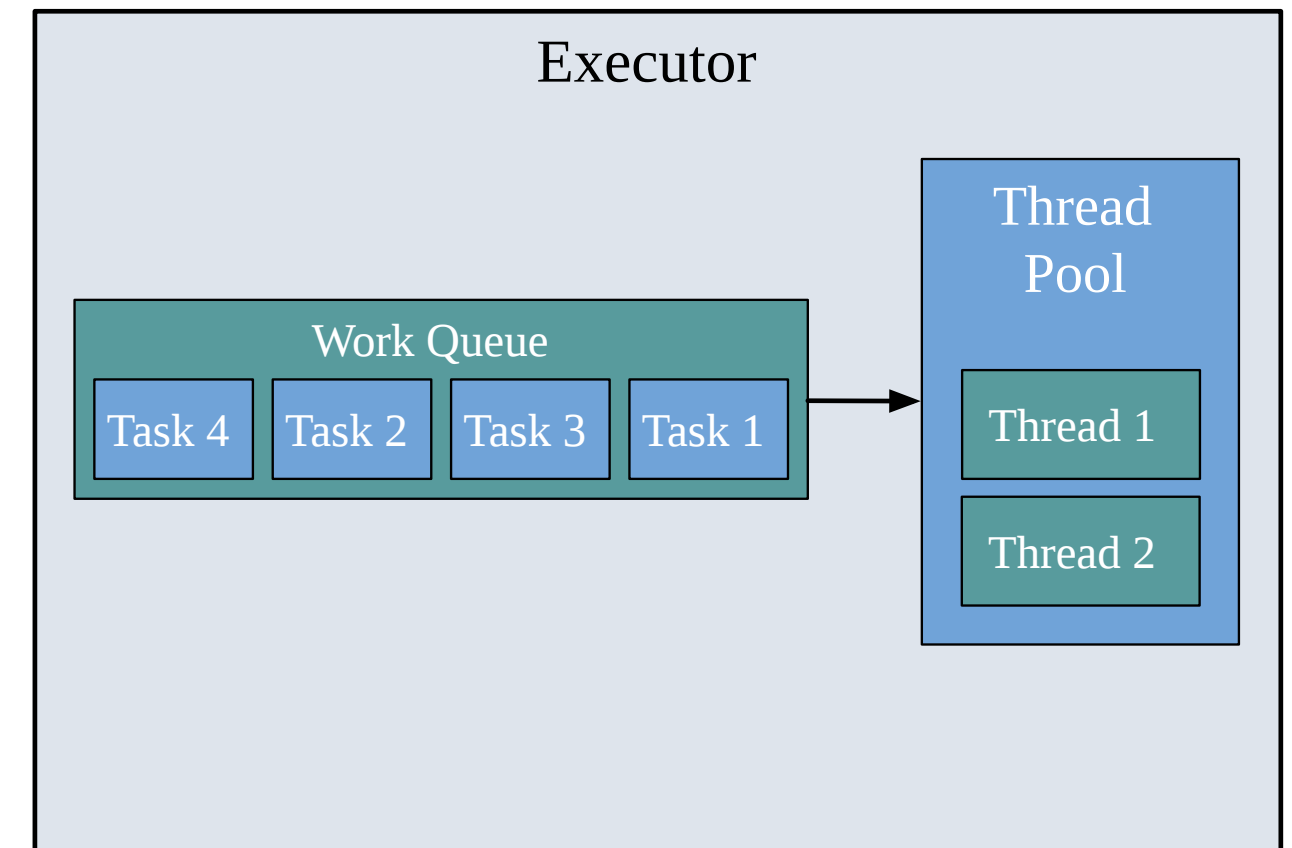
```
import java.util.concurrent.Executors

val fixedPool = Executors.newFixedThreadPool(2)

def createRunnable(n: Int): Runnable =
  new Runnable {
    def run(): Unit = {
      val thread = Thread.currentThread
      println(s"[${thread.getName}] Runnable ${n}")
    }
  }

val runnables = 1.to(4).map(createRunnable)
```

```
runnables.foreach(fixedPool.submit)
// [pool-19-thread-1] Runnable 1
// [pool-19-thread-2] Runnable 3
// [pool-19-thread-1] Runnable 2
// [pool-19-thread-2] Runnable 4
```





# ExecutionContext and Future

```
import java.util.concurrent.Executors
import scala.concurrent.duration._
import scala.concurrent.{ Await, ExecutionContext, Future }

val fixedPool      = Executors.newFixedThreadPool(2)
val executionContext = ExecutionContext.fromExecutor(fixedPool)
```

```
val future = Future {
  Thread.sleep(1000) // sleep 1 second
  1
}(executionContext)
// future: Future[Int] = Future(<not completed>)
```

# ExecutionContext and Future

```
import java.util.concurrent.Executors
import scala.concurrent.duration._
import scala.concurrent.{ Await, ExecutionContext, Future }

val fixedPool = Executors.newFixedThreadPool(2)
implicit val executionContext = ExecutionContext.fromExecutor(fixedPool)
```

```
val task = Future {
  Thread.sleep(1000) // sleep 1 second
  3
}
// task: Future[Int] = Future(<not completed>)
```

# ExecutionContext and Future

```
import java.util.concurrent.Executors
import scala.concurrent.duration._
import scala.concurrent.{ Await, ExecutionContext, Future }

val fixedPool = Executors.newFixedThreadPool(2)
implicit val executionContext = ExecutionContext.fromExecutor(fixedPool)
```

```
val task = Future {
  Thread.sleep(1000) // sleep 1 second
  3
}
// task: Future[Int] = Future(<not completed>)
```

```
Await.result(task, 2.minutes)
// res: Int = 3
```

# ExecutionContext and Future

```
import java.util.concurrent.Executors
import scala.concurrent.duration._
import scala.concurrent.{ Await, ExecutionContext, Future }

val fixedPool = Executors.newFixedThreadPool(2)
implicit val executionContext = ExecutionContext.fromExecutor(fixedPool)
```

```
val task = Future {
  Thread.sleep(1000 * 60 * 5) // sleep 5 minutes
  3
}
// task: Future[Int] = Future(<not completed>)
```

```
Await.result(task, 2.minutes)
// java.util.concurrent.TimeoutException: Future timed out after [2 minutes]
```

# ExecutionContext and Future

```
import java.util.concurrent.Executors
import scala.concurrent.duration._
import scala.concurrent.{ Await, ExecutionContext, Future }

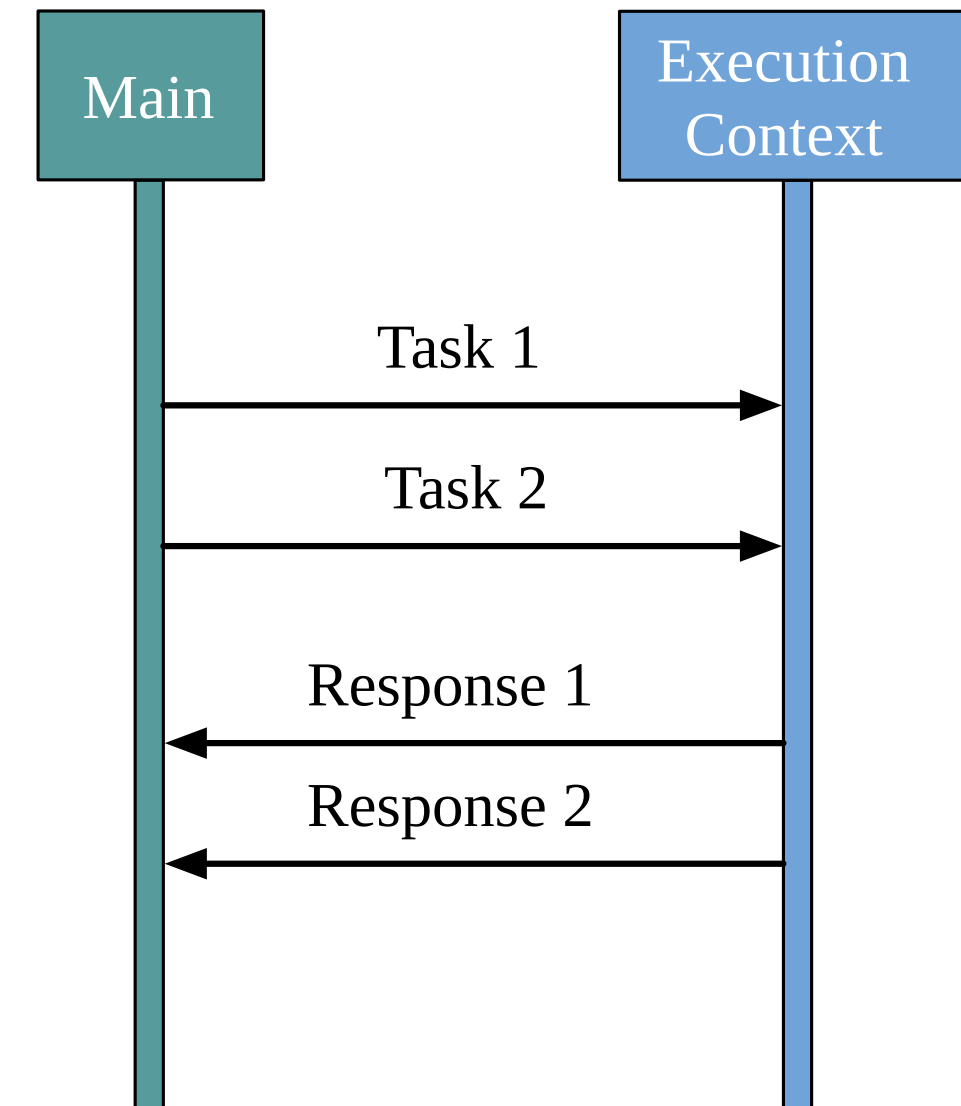
val fixedPool = Executors.newFixedThreadPool(2)
implicit val executionContext = ExecutionContext.fromExecutor(fixedPool)
```

```
val task = Future {
  Thread.sleep(1000 * 60 * 5) // sleep 5 minutes
  3
}
// task: Future[Int] = Future(<not completed>)
```

```
Await.result(task, Duration.Inf)
// res: Int = 3
```

# ExecutionContext and Future

```
val future1  = Future { task(1) }  
val future2  = Future { task(2) }  
  
val response1 = Await.result(future1, Duration.Inf)  
val response2 = Await.result(future2, Duration.Inf)
```



# ExecutionContext and Future

```
val future1    = Future { task(1) }  
val response1 = Await.result(future1, Duration.Inf)  
  
val future2    = Future { task(2) }  
val response2 = Await.result(future2, Duration.Inf)
```

