Sparkにおけるディスクを用いた RDDキャッシングの高速化と 効果的な利用に関する検討

張 凱輝　　（筑波大）
谷村 勇輔（産総研/筑波大）
中田 秀基（産総研/筑波大）
小川 宏高　　（産総研）

研究目的

- Apache Spark (以下、Spark) は並列データ処理フレームワークの1つ．中間データをメモリに保持することにより、優れた性能を提供

- 中間データをディスクに保持することも可能だが、実行性能の低下を招く恐れがある．ディスク利用の有無はユーザに委ねられ、適切に判断することが簡単ではない

- 中間データの保持にディスクを利用する際の問題点や注意点を明らかにし、高速化の可能性を探る

発表の構成

1. 研究目的
2. Sparkの概要
    1. Spark とは
    2. RDD (Resilient Distrbuted Datasets)
    3. ストレージレベル
3. 評価実験
    1. 実験項目と方法
    2. 実験環境
    3. 実験結果
4. まとめ
5. 今後の課題

Apache Sparkとは

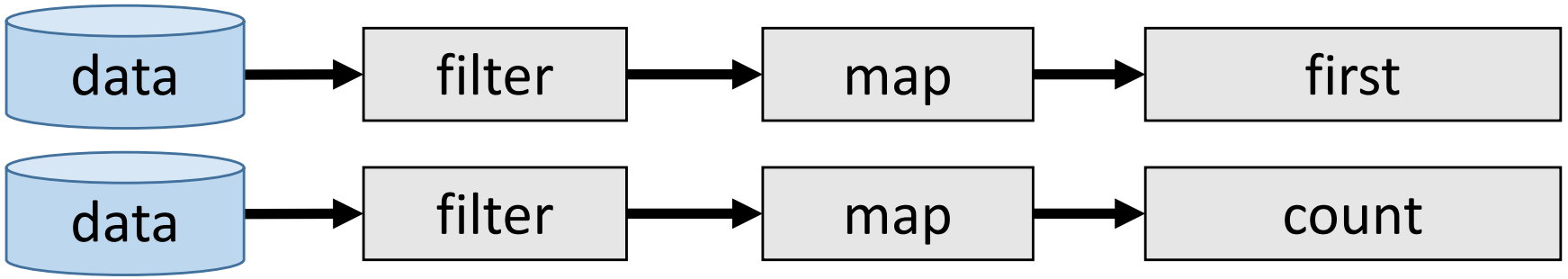
- オープンソースの並列データ処理フレームワーク

- Hadoop と同様に MapReduce のアルゴリズムに基づいた分散処理が可能
  - コンピューティングとストレージの両機能を持つノードからなるクラスタ上で実行

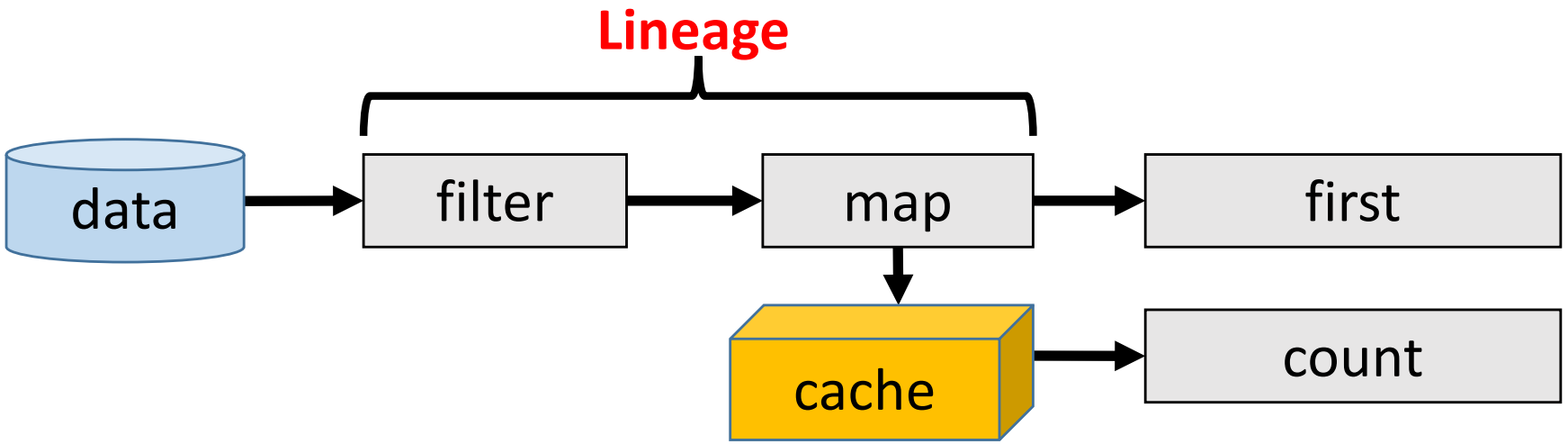
- 中間データをメモリに保持することで機械学習やデータマイニングなどの反復計算を効率良く実行
  - 反復計算の負荷が大きいほど、Spark を利用するメリットはより大きくなる

RDDキャッシング

キャッシュがない場合、同じ処理が2回行われる



キャッシュがある場合、map までの再処理が不要である

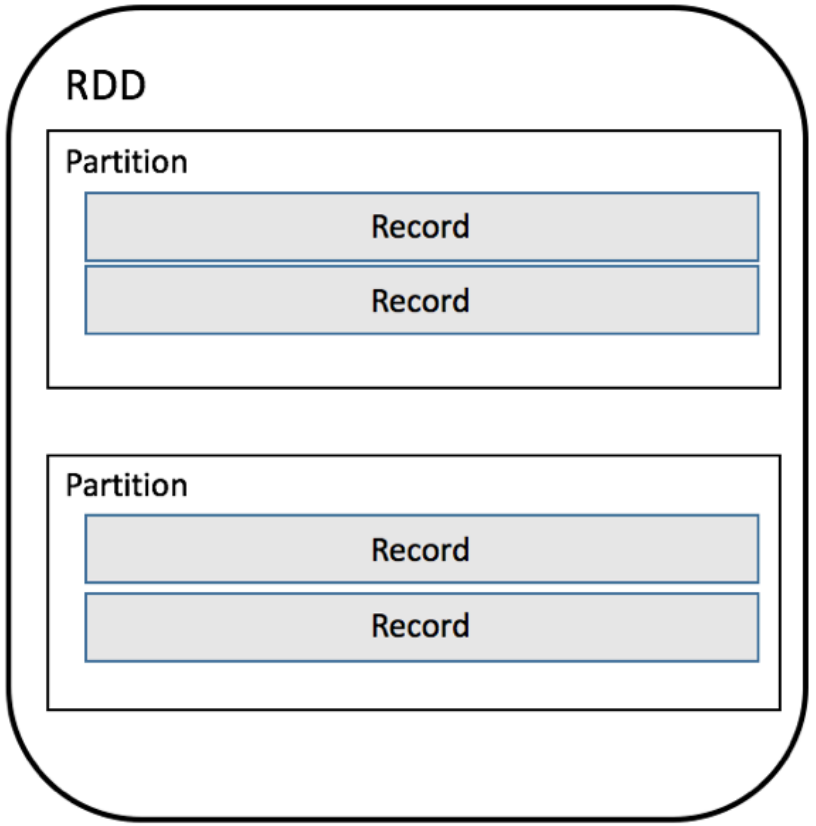


RDD キャッシング

- RDD を生成したノードは、自身が生成したパーティションを明示的にメモリあるいはディスクに保存することが可能
- RDD に対する反復操作や障害からの復旧を高速化する効果
- RDD に対して persist() メソッドを呼び出して生成
  - 引数にストレージレベルを指定

RDDの内部構造

- Read-only の分散データ構造
- 内部はパーティションに分割
- 各パーティションは複数のワーカノードに分散配置
  - データ処理の単位として分散並列実行が可能

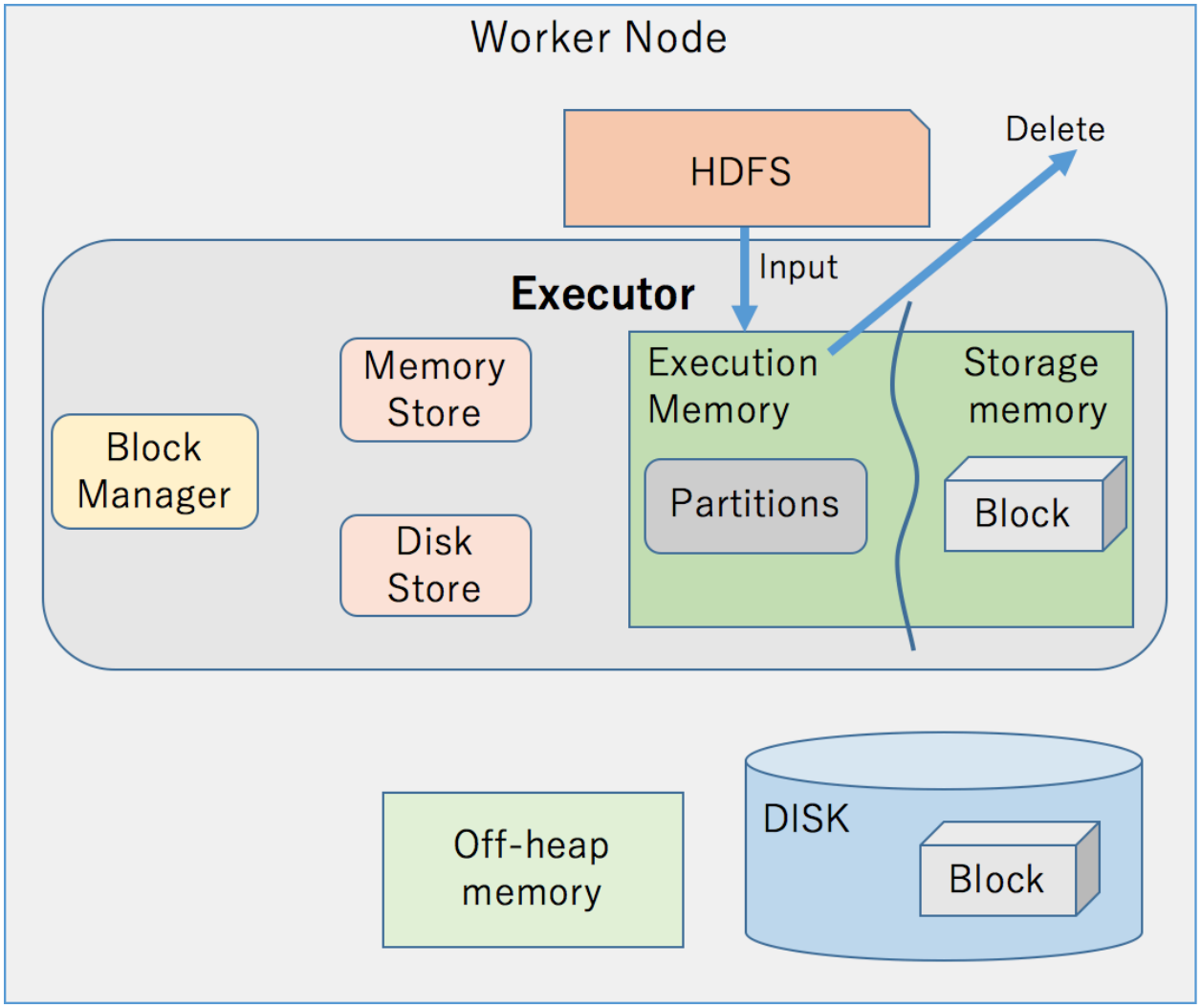


RDDキャッシュのストレージレベル

- メモリやディスク、あるいはそれらの併用が可能
- メモリを利用する場合はシリアライズの有無も指定可能
- 引数:
  - MEMORY_ONLY
  - MEMORY_ONLY_SER
  - MEMORY_AND_DISK
  - DISK_ONLY
  - OFF_HEAP
  - (NOCACHE)

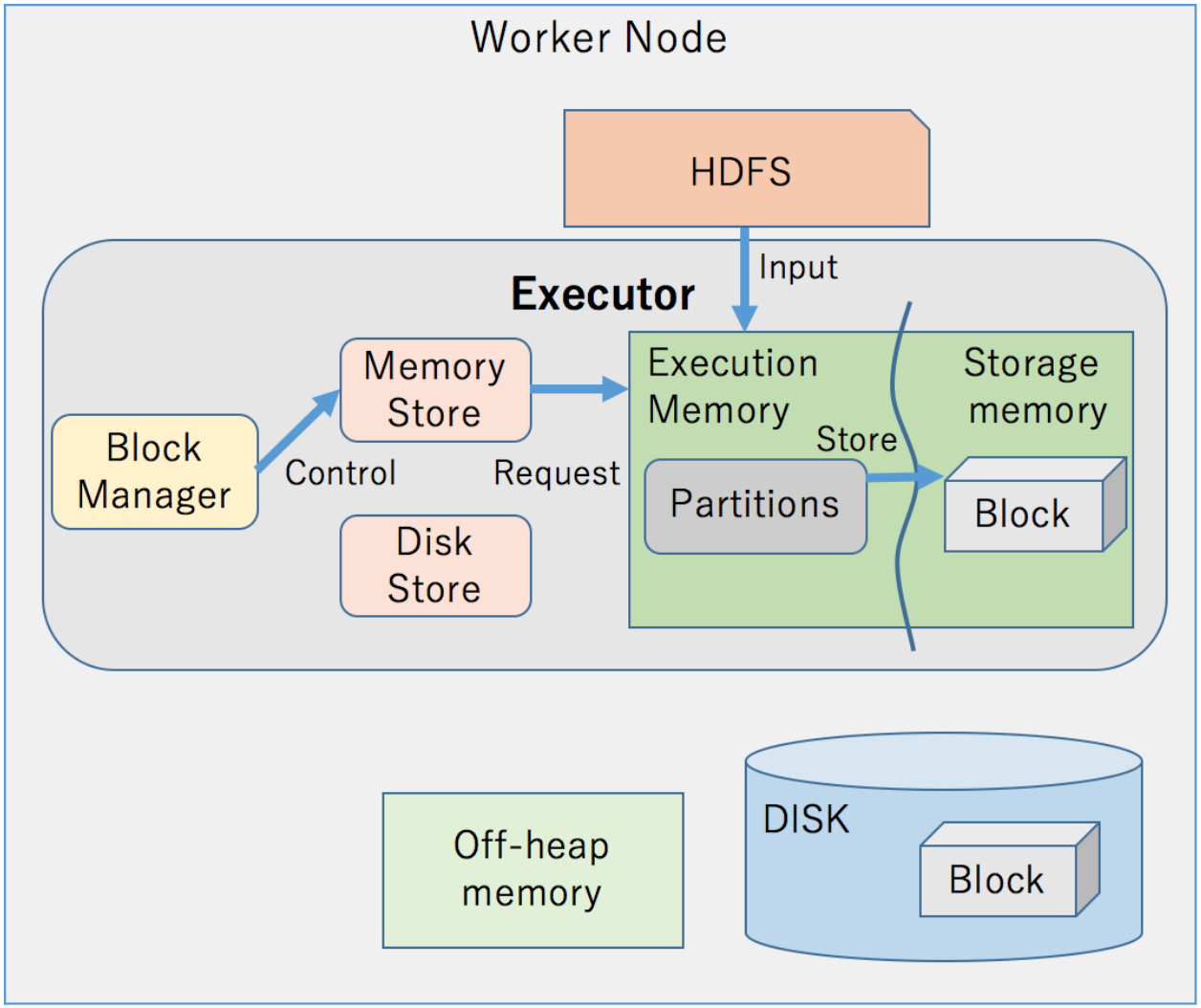
NOCACHE

- RDD キャッシュを保持しない
- RDD を使い捨てる



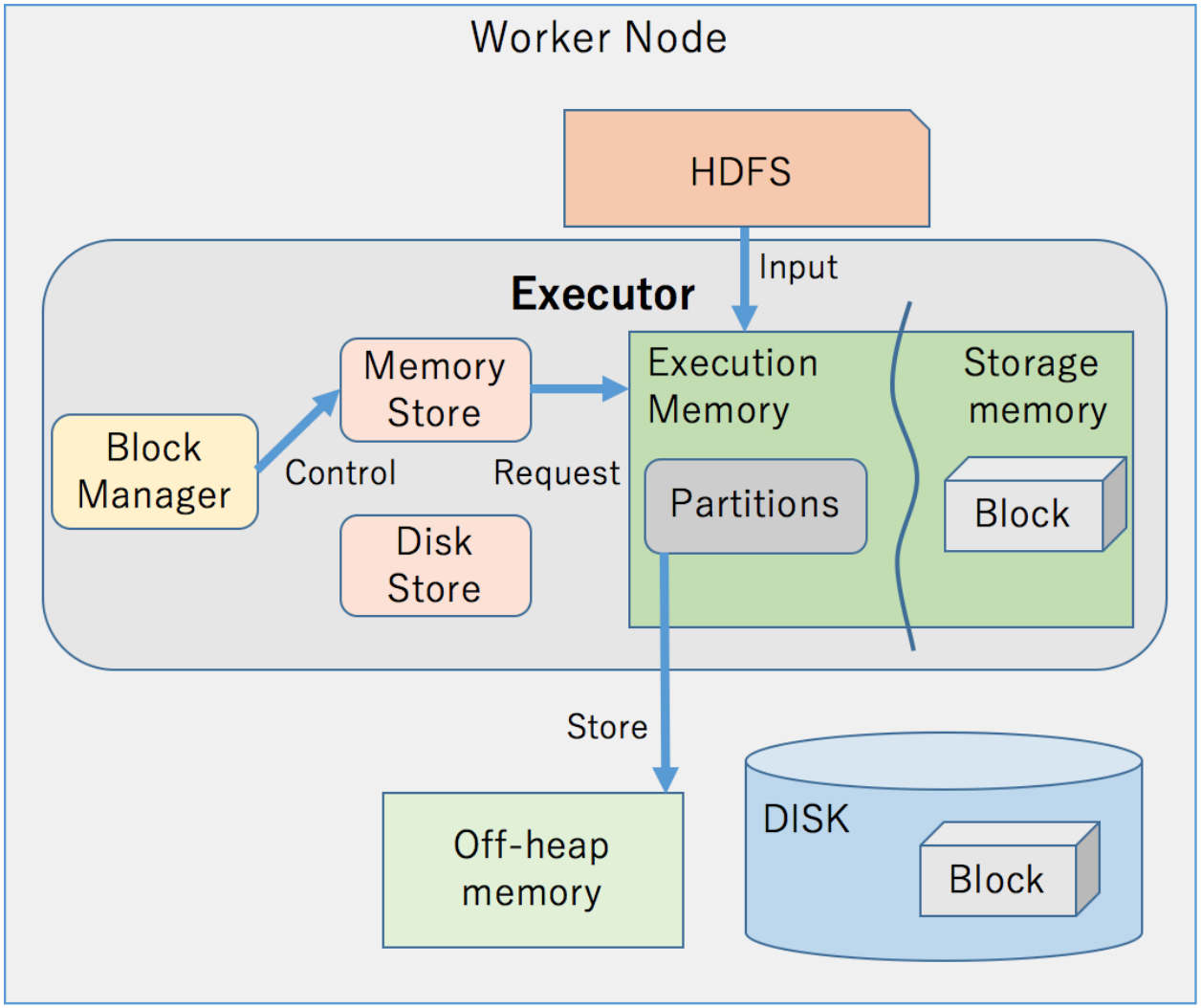
MEMORY_ONLY(default)

- RDD キャッシュをメモリ上に保持



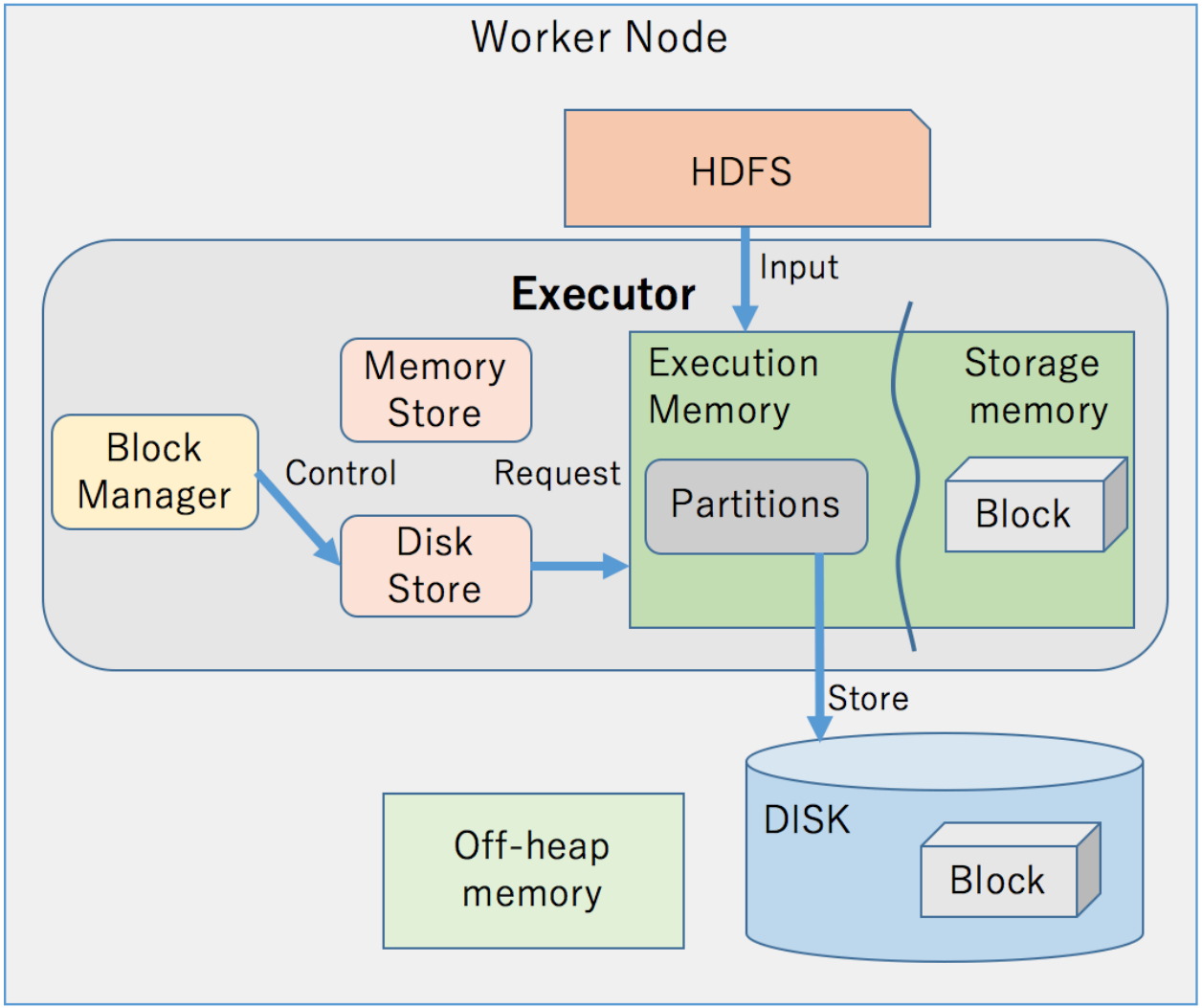
OFF_HEAP

- RDD キャッシュを off-heap メモリ上に保持
- GC の削減ができる



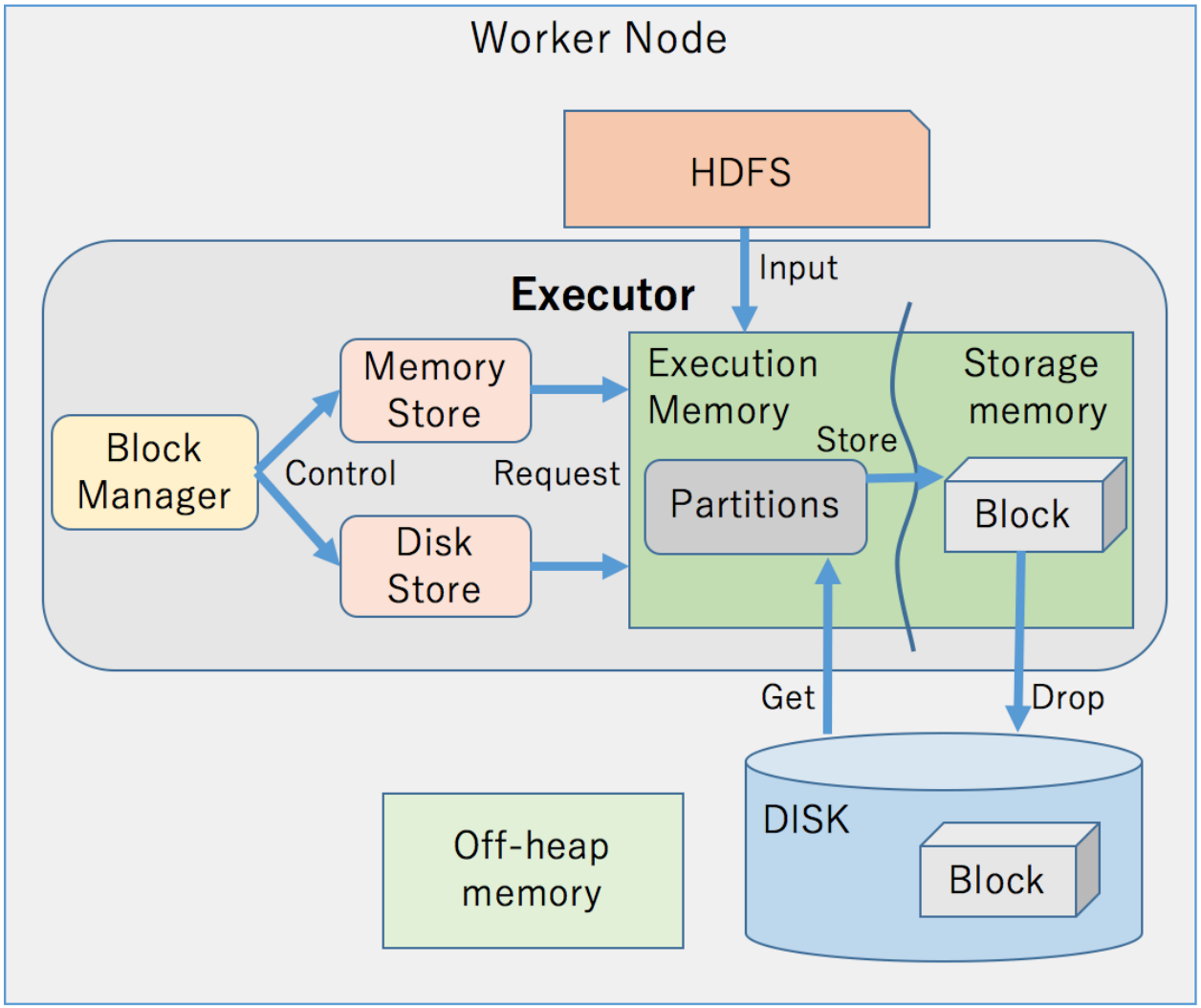
DISK_ONLY

- RDD キャッシュ をディスク上に保持
- メモリを消費しない



MEMORY_AND_DISK

- メモリとディスクを併用して RDDキャッシュを保持
- メモリを優先的に利用し、収まりきらない場合はディスクに移動



評価実験

- 調査項目
  1. ディスクを用いたRDDキャッシングの効果
  2. メモリとディスクの併用した場合の性能評価と改良
  3. ストレージデバイスによる性能の違い

- 調査方法
  - Sparkの機械学習ライブラリ(MLlib)に含まれるベンチマークと独自の測定ベンチマークを実行
  - RDDキャッシュの有無、ストレージレベル、RDDのサイズ、ドライバーのスレッド数、ストレージデバイスを変更し、性能を測定

実験環境

CPU	Intel Xeon CPU E5-2620v3 2.40GHz 6 cores x2
Memory	128 GB
NetWork	10 Gbps (for HDFS connection)
NVMe-SSD	Intel SSD DC P3700
SSD	OCZ Vertex3 (240GB, SATA6G I/F)
HDD	Hitachi Travelstar 7K320 (SATA3G I/F)
OS	Ubuntu 14.04 (Kernel v.3.13)
File system	ext4

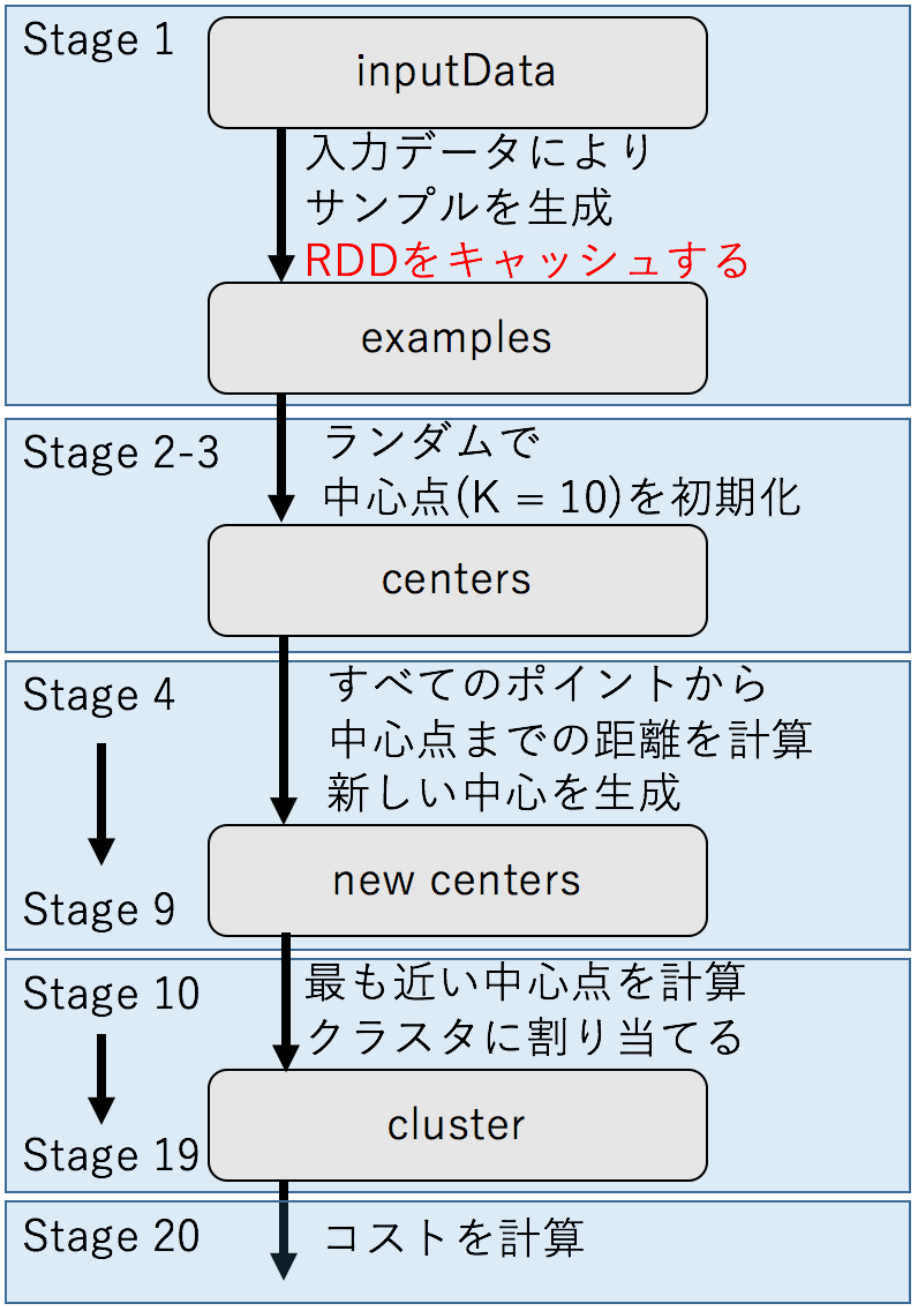
- ローカルモード（DriverMemory = ExecutorMemory）で各ベンチマーク (DenseKMeans、RDDTest) を実行
- RDD キャッシュにディスクを利用する場合には、RamDisk、NVMe-SSD、SATA-SSD および HDD の 4 種類を使用
- Spark v2.1.0、Scala v.2.10.6、Java v.1.8.0 66 を使用

機械学習プログラムによるベンチマーク

- DenseKMeans（k平均法）を利用
- 入力データは HiBench(6.0) を用いて生成、データサイズには large を指定
  - 入力データサイズ : 4 GB
- 異なるストレージレベルにおいて（各ステージ）実行時間の比較

＊ NaiveBayes の実験データは省略

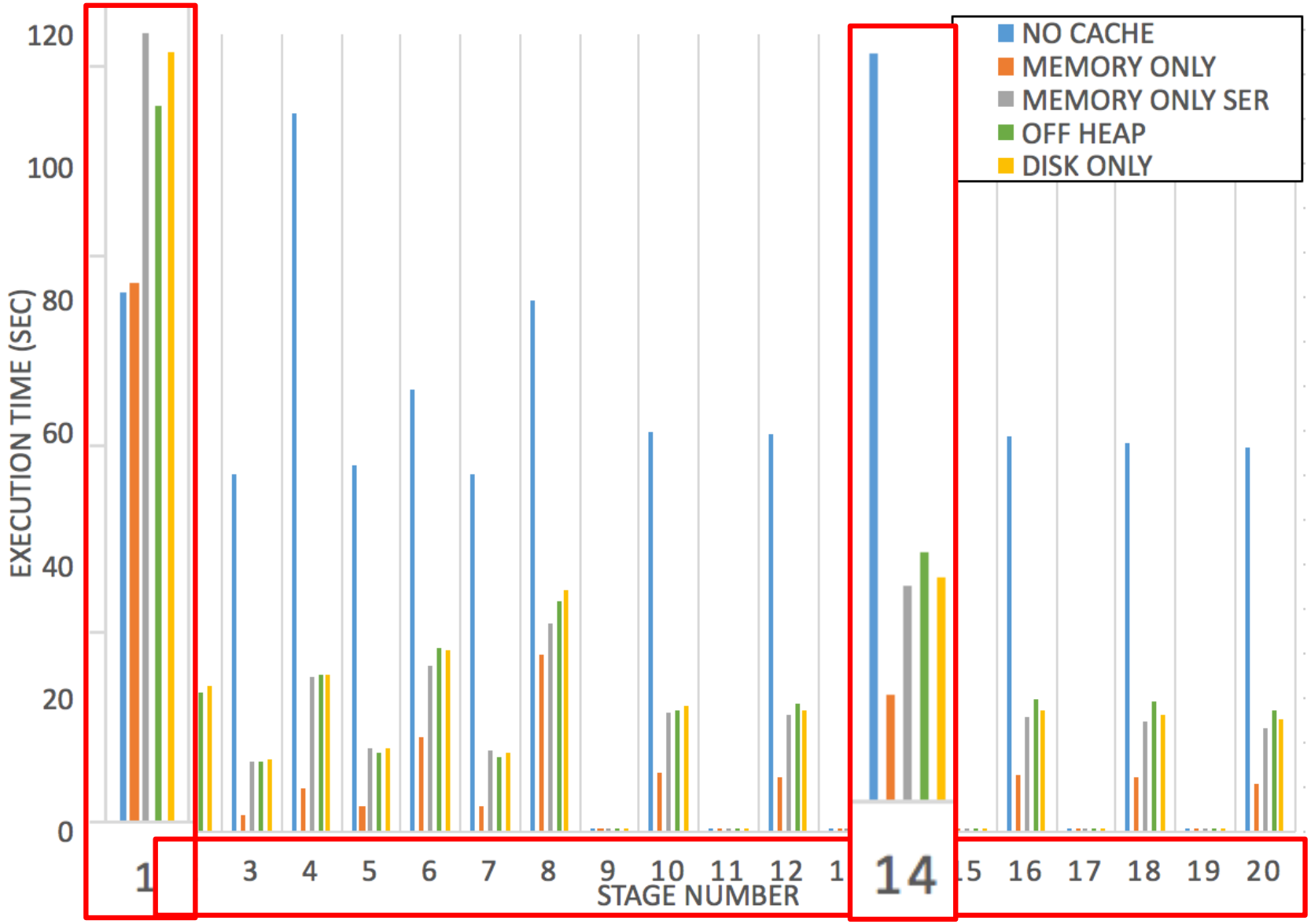
DenseKMeans のデータフロー



調査項目１: ディスクを用いたRDDキャッシングの効果

- 設定
  - スレッド数: １
  - ドライバーのメモリ: 60GB
  - ストレージレベル: NOCACHE
    MEMORY_ONLY
    MEMORY_ONLY_SER
    DISK_ONLY
  - ベンチマーク: DenseKMeans

ディスクを用いた RDD キャッシングの効果



異なるストレージレベルのステージ毎の実行時間

調査項目２: メモリとディスクの併用による影響

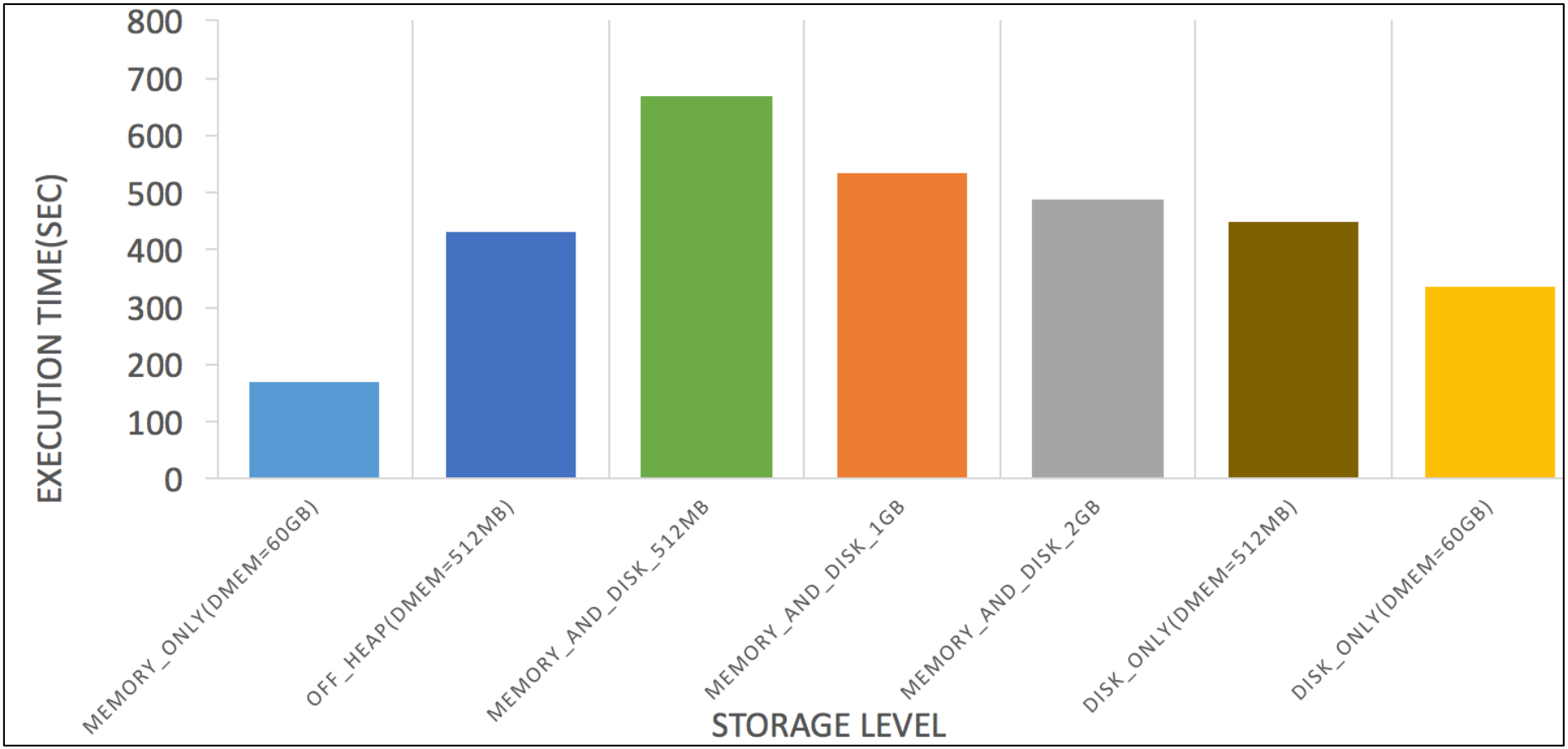
- 設定
  - スレッド数 : 1

  - ストレー ジレベル : MEMORY_AND_DISK
  - ドライバーメモリ量 : 512MB、1GB、2GB
  - ストレー ジレベル : OFF_HEAP
  - ドライバーメモリ量 : 512MB
  - ストレー ジレベル : MEMORY_ONLY、DISK_ONLY
    ドライバーメモリ量 : 60GB

  - ベンチマーク : DenseKMeans

メモリとディスクの併用による影響

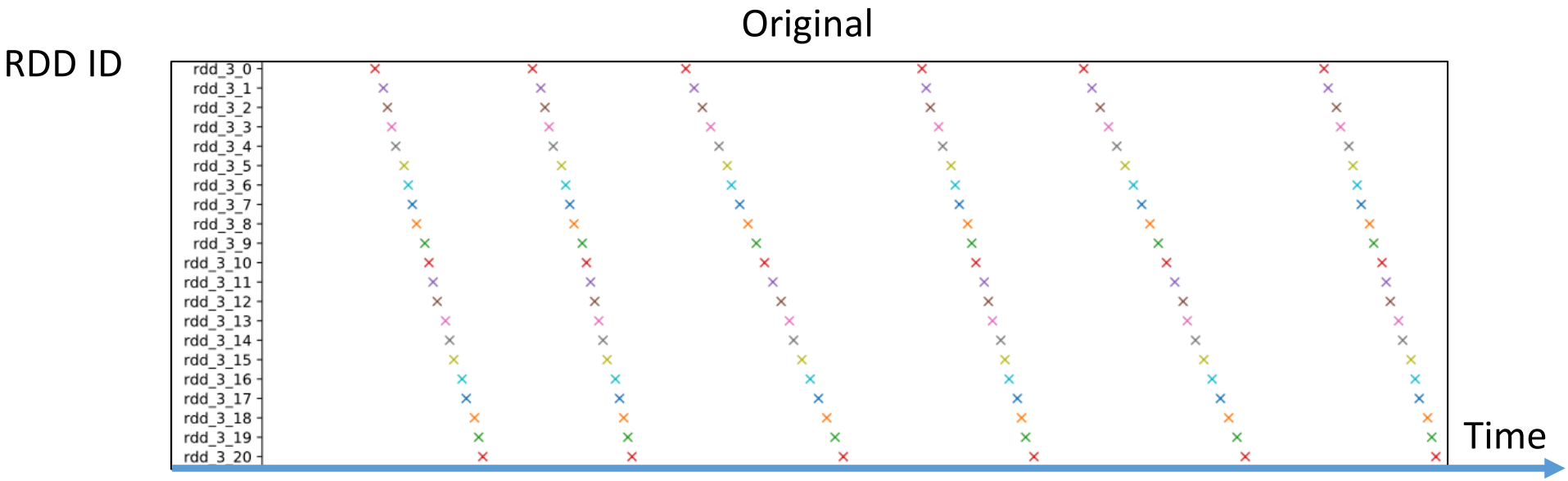
各ストレージレベルにおける実行時間

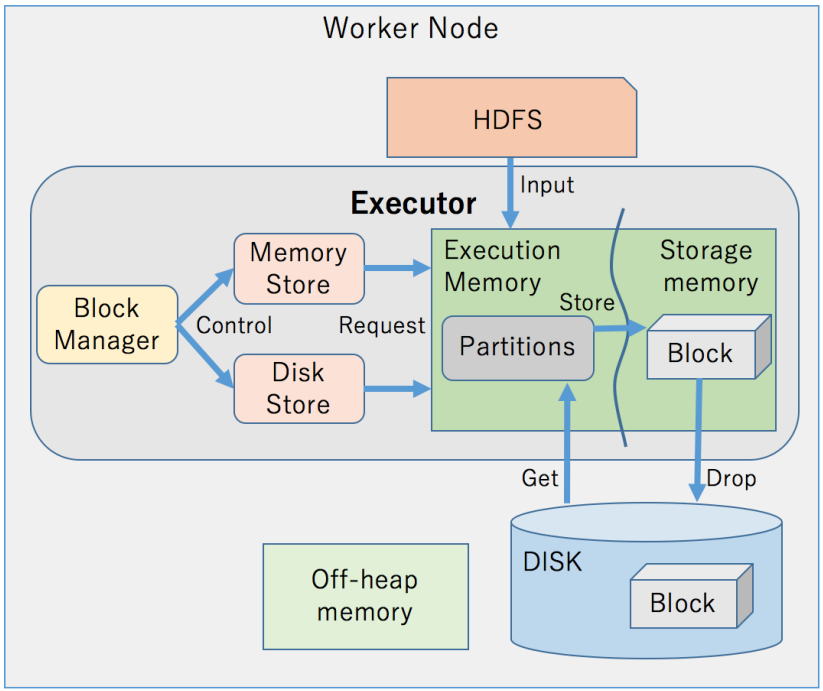


MemoryStore からのドロップの発生頻度と量

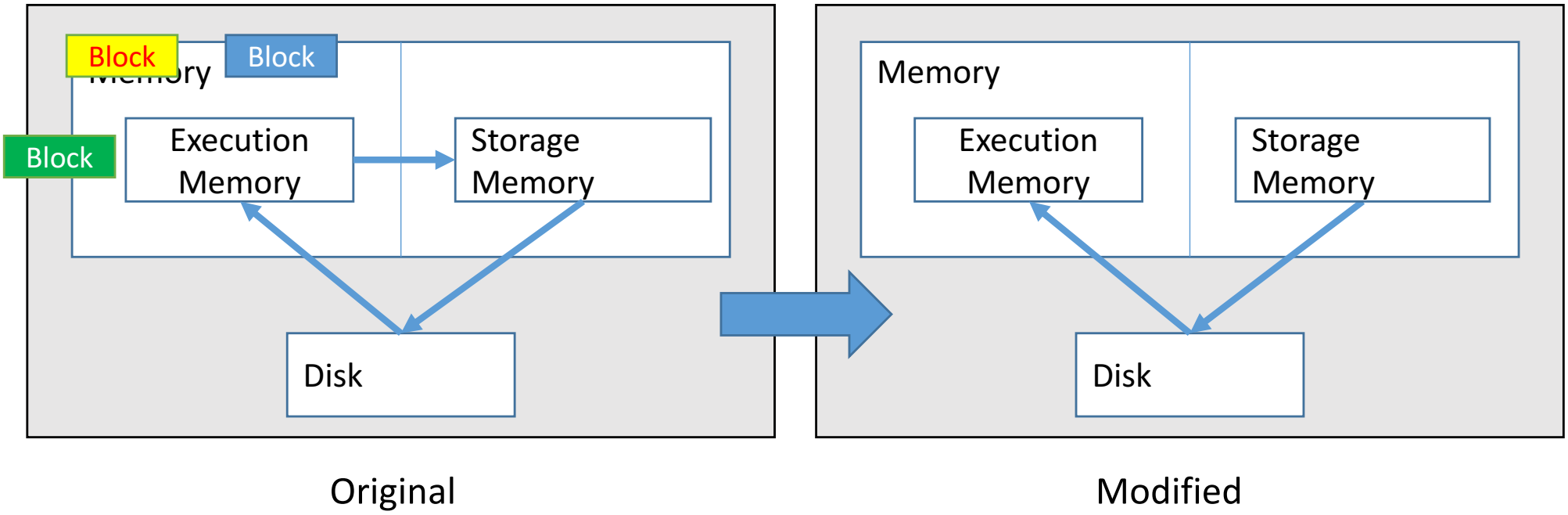
	#Blocks	Size (MB)
MEMORY_AND_DISK_512MB	600	2,577
MEMORY_AND_DISK_1GB	746	26,793
MEMORY_AND_DISK_2GB	715	20,936
OFF_HEAP_512MB	305	1,332

ドロップの様子


Original
RDD ID
rdd_3_0
rdd_3_1
rdd_3_2
rdd_3_3
rdd_3_4
rdd_3_5
rdd_3_6
rdd_3_7
rdd_3_8
rdd_3_9
rdd_3_10
rdd_3_11
rdd_3_12
rdd_3_13
rdd_3_14
rdd_3_15
rdd_3_16
rdd_3_17
rdd_3_18
rdd_3_19
rdd_3_20
Time


Worker Node
HDFS
Input
Executor
Memory Store
Execution Memory
Storage memory
Store
Block Manager
Control
Request
Partitions
Block
Disk Store
Get
Drop
DISK
Block
Off-heap memory

同じブロックに対する再ドロップの抑制

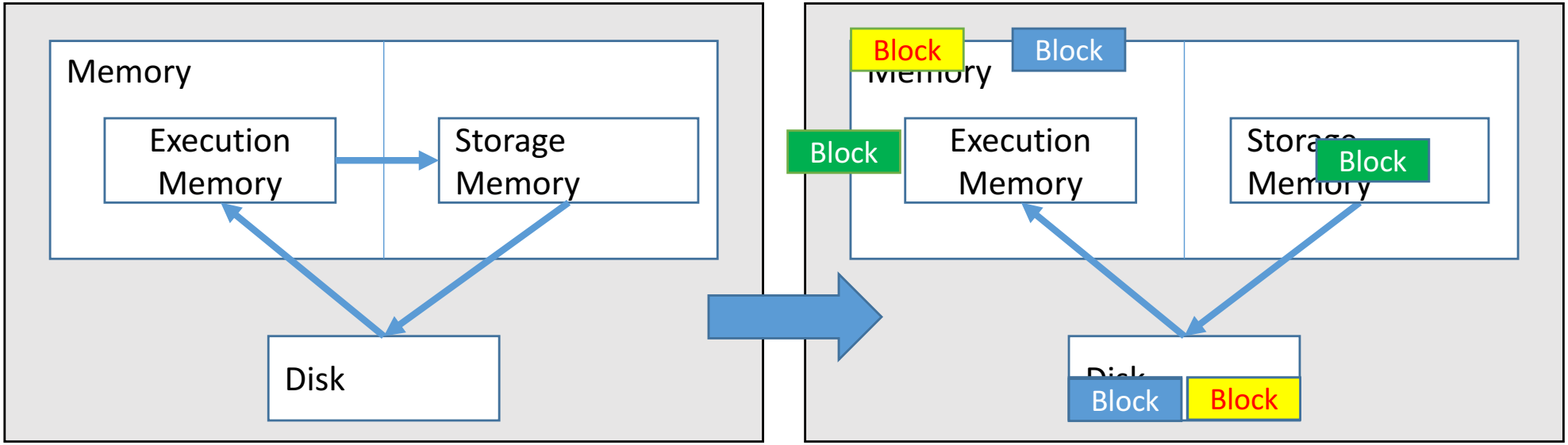


Original

Modified

- ドライバーメモリへの割り当て量が不十分な場合、RDD キャッシュを構成するブロックが繰り返し、StorageMemory に入る状況が発生していることを確認
- 提案手法：一旦ディスクにドロップされたブロックを StorageMemory に戻さず、そのままディスクに保持し、ExecutionMemory に読み込む

同じブロックに対する再ドロップの抑制

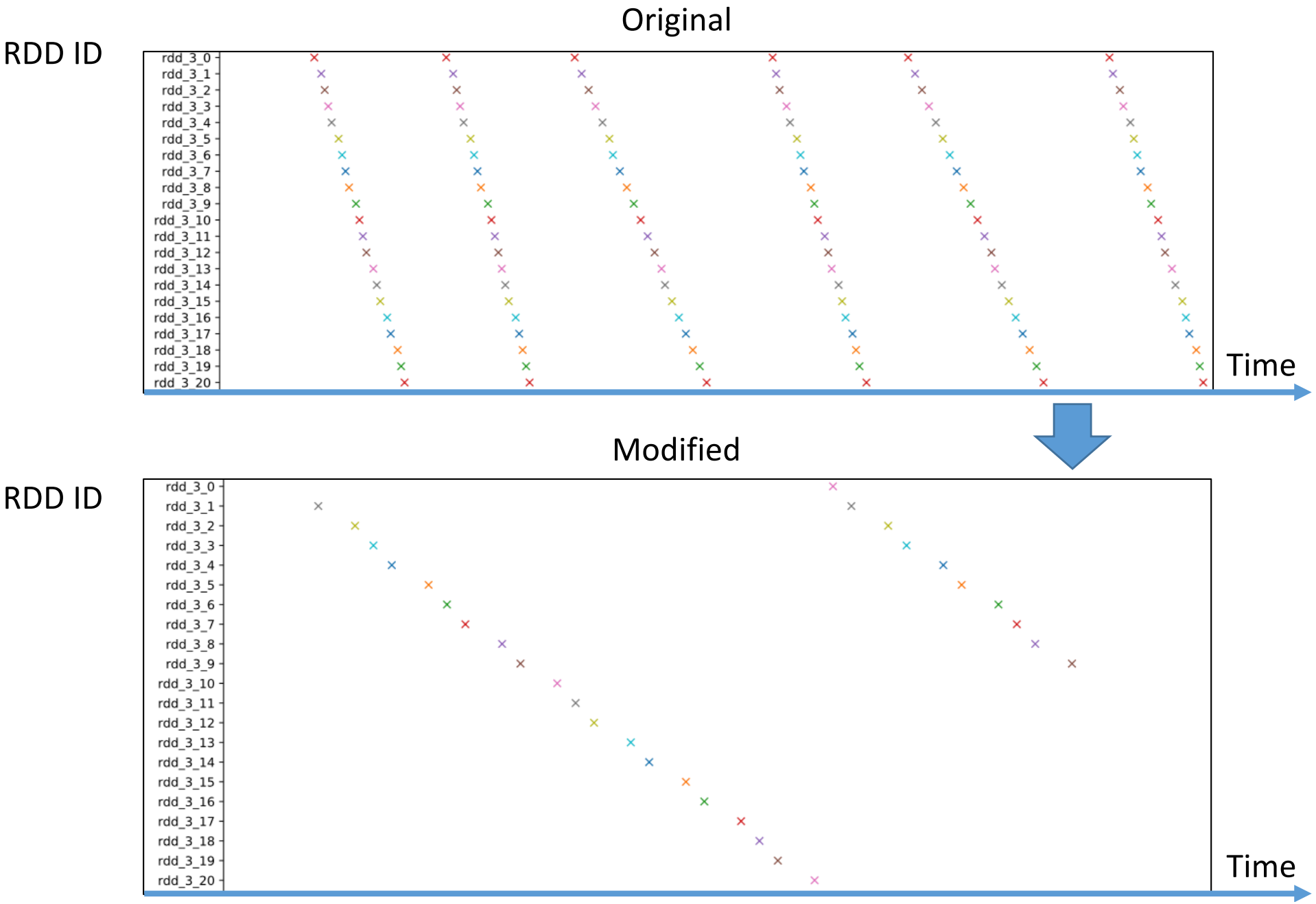


Original

Modified

- ドライバーメモリ割り当てが不十分な場合、RDD キャッシュを構成するブロックが繰り返し、StorageMemory に入る状況が発生していることを確認
- 提案手法：一旦ディスクにドロップされたブロックを StorageMemory に戻さず、そのままディスクに保持し、ExecutionMemory に読み込む

再ドロップ抑制後のドロップの様子



再ドロップ抑制後の実行結果

ドロップの発生回数と合計サイズの比較

	Original		Modified	
	#Blocks	Size(MB)	#Blocks	Size(MB)
MEMORY_AND_DISK_512MB	600	2,577	133	610
MEMORY_AND_DISK_1GB	746	26,793	33	486
MEMORY_AND_DISK_2GB	715	20,936	58	473

実行時間の比較

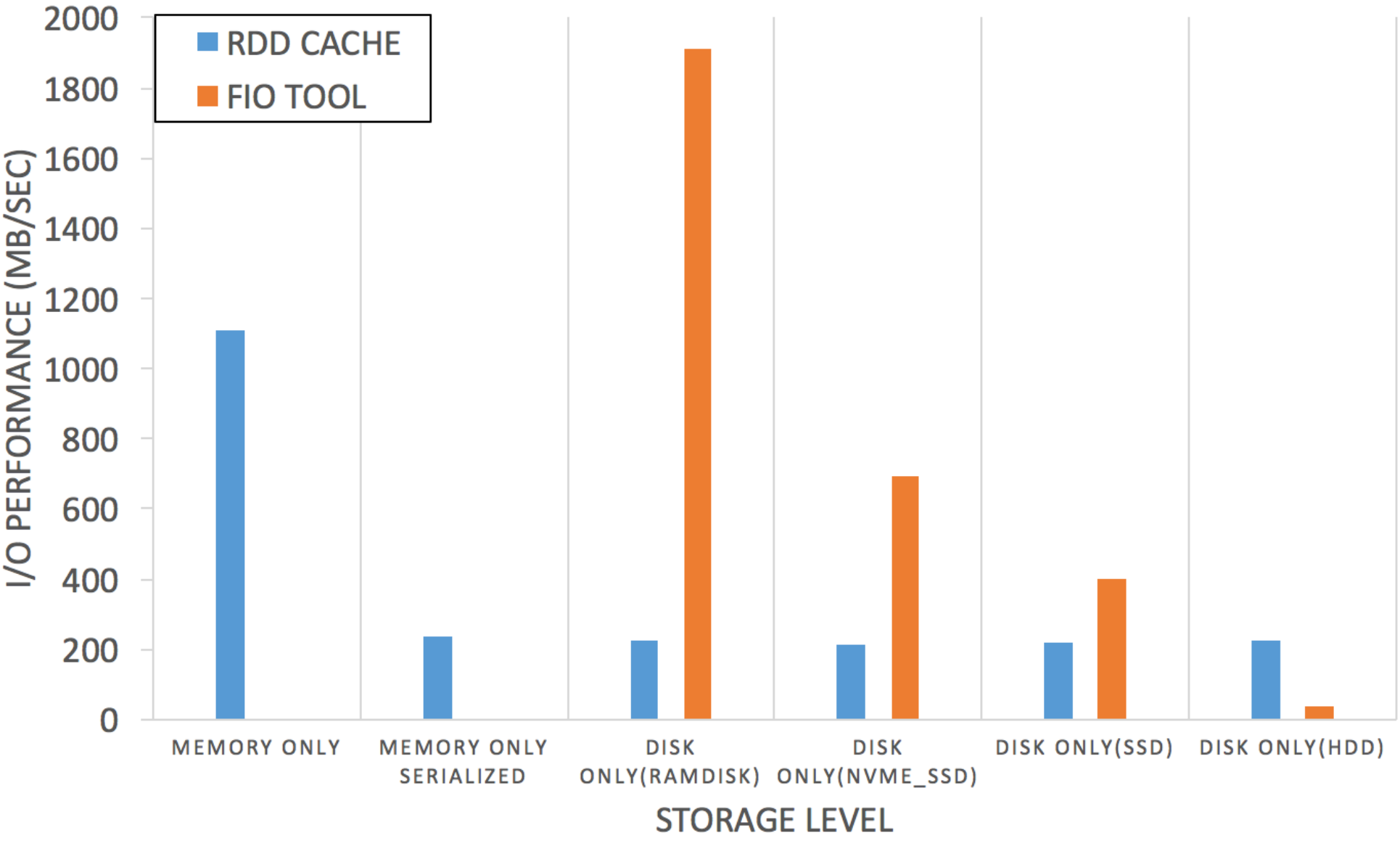
	Original	Modified
MEMORY_AND_DISK_512MB	670 s	422 s
MEMORY_AND_DISK_1GB	533 s	331 s
MEMORY_AND_DISK_2GB	489 s	318 s

調査項目３: ストレージデバイスによる違い

- 設定１
  - スレッド数は **1**
  - ドライバーメモリ量 : **64GB**
  - ストレー ジレベル : **DISK_ONLY**
  - RDDのサイズ : **1000MB**

  - ベンチマーク : **RDDTest、Fioベンチマーク**

RDDキャッシングの実行速度の比較 (1 スレッド)

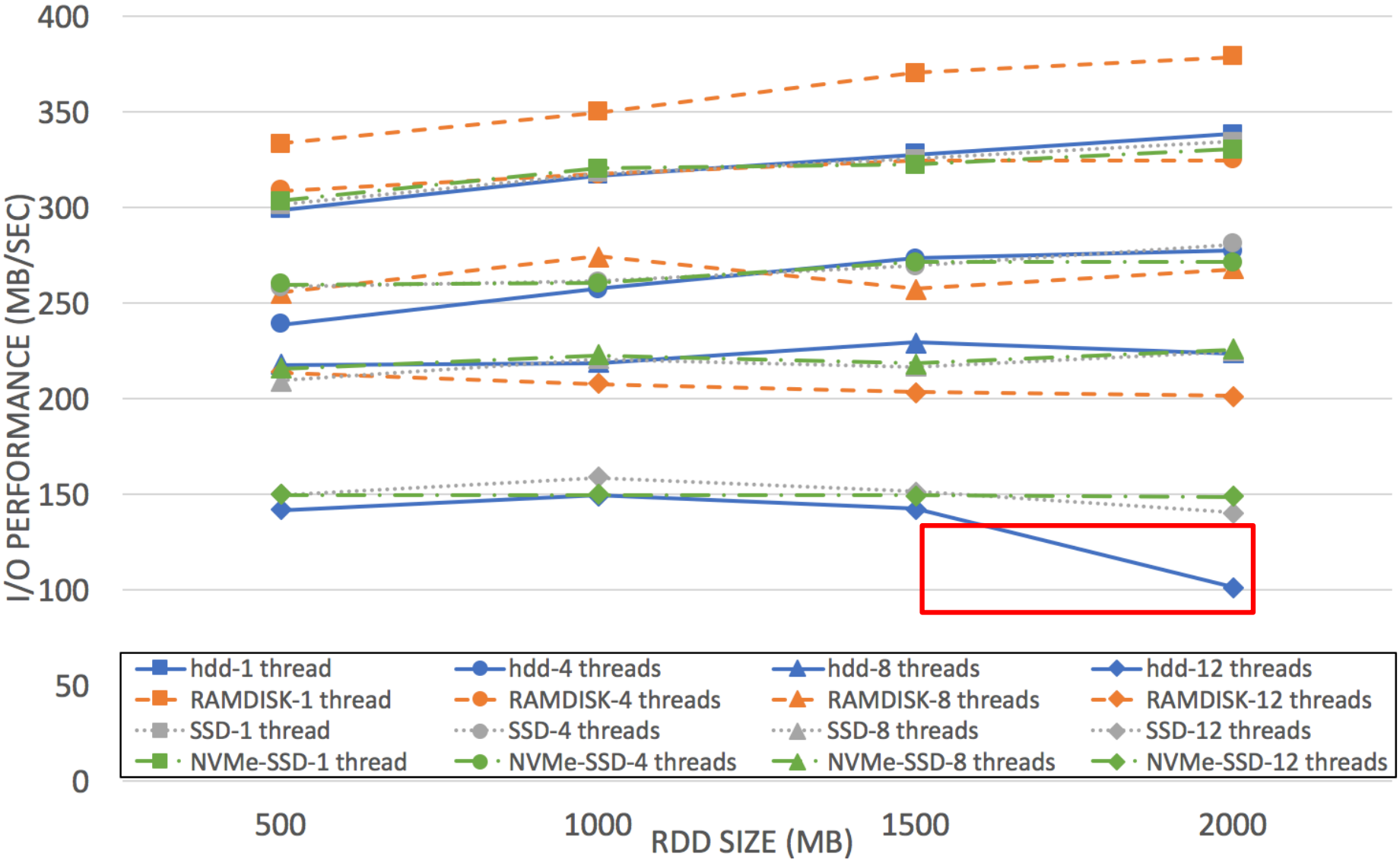


調査項目３: ストレージデバイスによる違い

- 設定２
  - スレッド数 : **4、8、12**
  - ドライバーのメモリ量 : **64GB**
  - ストレージレベル : **DISK_ONLY**
  - RDDのサイズ : **500MB、1000MB、1500MB、2000MB**

  - ベンチマーク : **RDDTest**

RDDキャッシュの実行速度の比較 (複数スレッド)



まとめ（１）

- RDD キャッシングにディスクを利用した場合、MEMORY_ONLY（Default）よりは確実に遅くなるが、それ以外とは性能差がほとんどない
- RDD キャッシングにメモリとディスクを併用する場合、ディスクのみを利用する場合より性能が低い場合がある
  - メモリ不足の場合にディスクへのドロップが多発し、ガベージコレクションの影響も受けるため
  - この場合はディスクのみを利用するのが有効
  - 一方、再ドロップを抑制する修正を施すことにより、この問題が解決できることも確認

まとめ（２）

- RDDキャッシングにおいては、OS のバッファキャッシュがあるためにディスクの性能はボトルネックになりにくい
- ただし、RDD サイズとスレッド数が増えるとバッファキャッシュの効果が低減し、ストレージデバイスの性能が重要になる

今後の課題

- シリアライズの高速化など Spark 内部の仕組みの改善が必要

- 今回提示した指針が他の GC アルゴリズムでも有効であるかを検証

謝辞

- この成果の一部は、国立研究開発法人新エネルギー・産業技術総合開発機構（NEDO）の委託業務の結果得られたものです。
- 本研究はJSPS科研費 JP16K00116の助成を受けたものです。