Apache Sparkにおける ブルームフィルタを用いた SQL処理の高速化手法

早瀬大智[†]　稲垣英夫[†,††]　川島龍太[†]　松尾啓志[†]

第147回システムソフトウェアと
オペレーティング・システム研究発表会 (SWoPP2019)
2019/07/25

[†] 名古屋工業大学
[††]株式会社小松製作所

研究背景

- インメモリ処理による高速化
- 抽象化されたAPI
- 複数言語をサポート

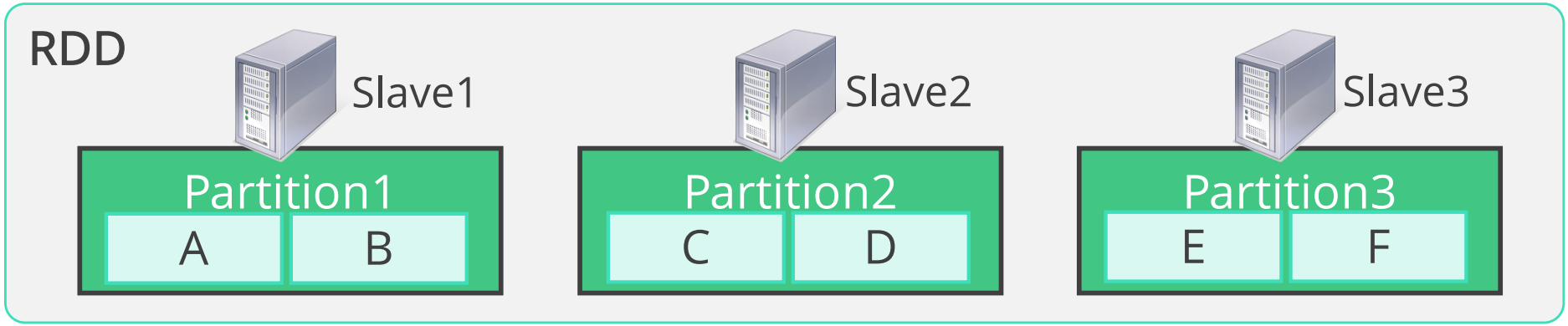
- データを明示的にキャッシュすることで、再構成を抑制
  - 冗長な計算の削減により処理の高速化を実現
  - キャッシュ管理手法にLRU(Least Recently Used)を採用
    - 直近に使用されたデータは必ずメモリに配置
    - **メモリディスク間の頻繁なデータ移動によって スラッシングが発生**

スラッシングを回避するキャッシュ管理手法を提案

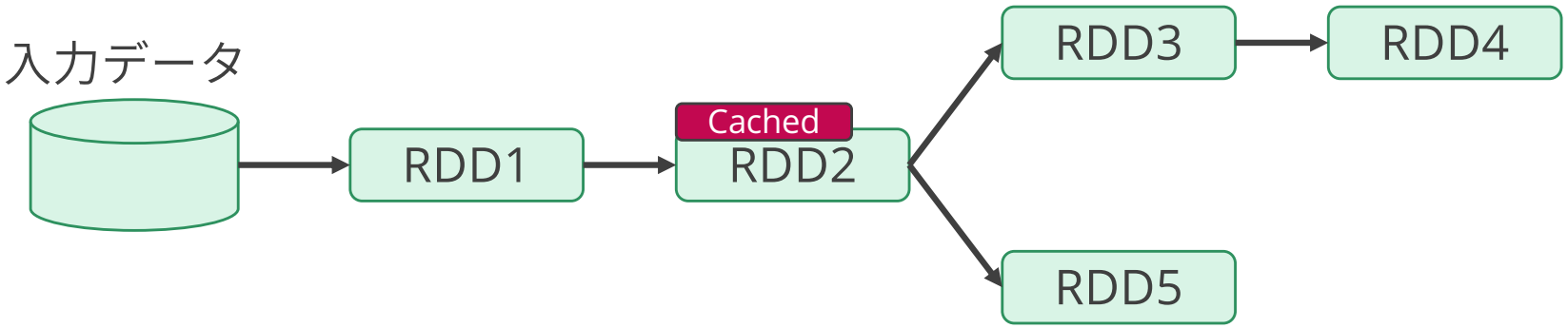
➢ 頻繁に利用されるデータをメモリに保持する手法の検討

RDD(Resilient Distributed Dataset)

- Sparkで利用される耐障害性に優れたデータセット
- データをパーティション単位で分割し、分散処理を実現

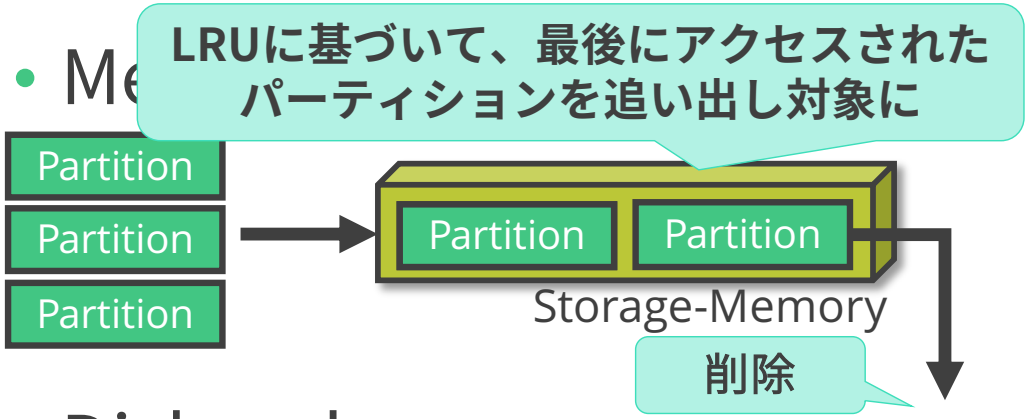


- 複数回利用するRDDをキャッシュすることで
冗長な処理を削減

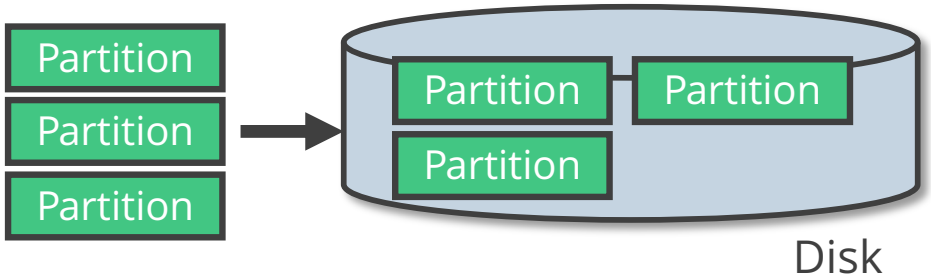


キャッシュデータの保存方式

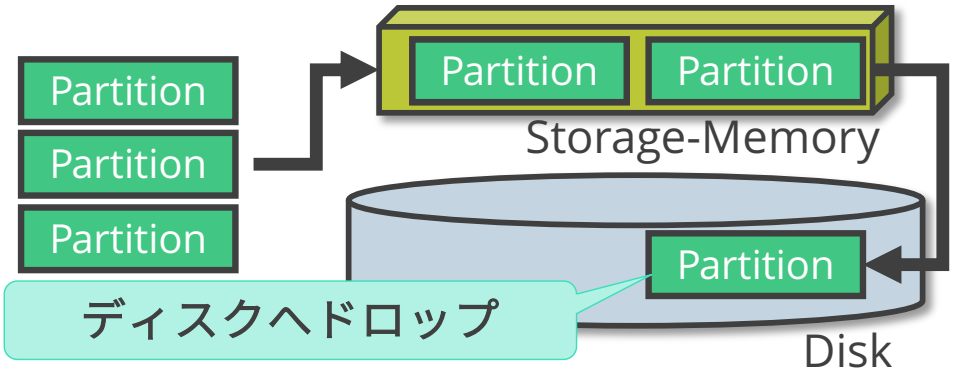
- Me



- Disk-only



- Memory-and-Disk

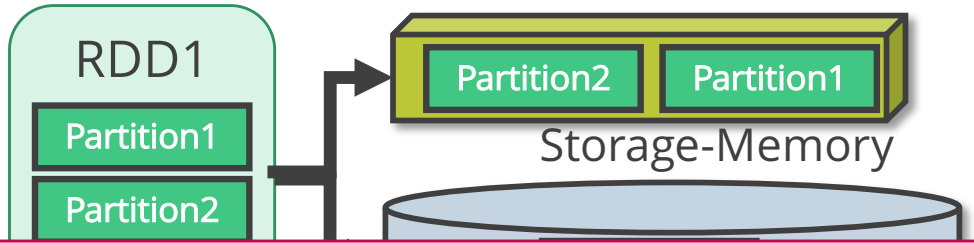


利点	欠点
高速なメモリI/O	パーティションの再生成が必要
パーティションの再生成が不要	低速なディスクI/O
高速なメモリI/O パーティションの再生成が不要	メモリ-ディスク間でスラッシングが発生

LRUの挙動と問題点

- キャッシュするRDDの数に応じて、挙動が変化
  - 単一RDDをキャッシュする場合

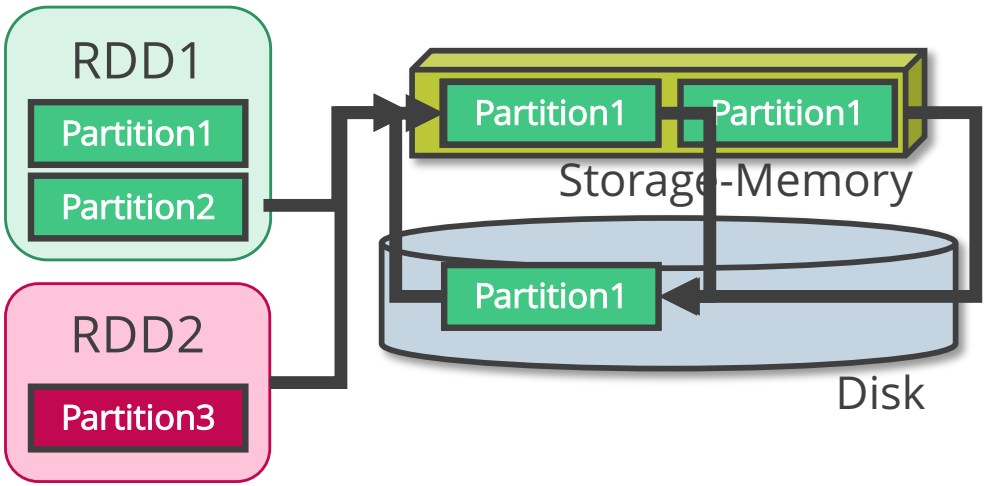
```
RDD1.cache()
RDD1.partition1.read()
```



複数RDDをキャッシュする場合、データ移動が頻発

➢ スラッシングによって性能低下

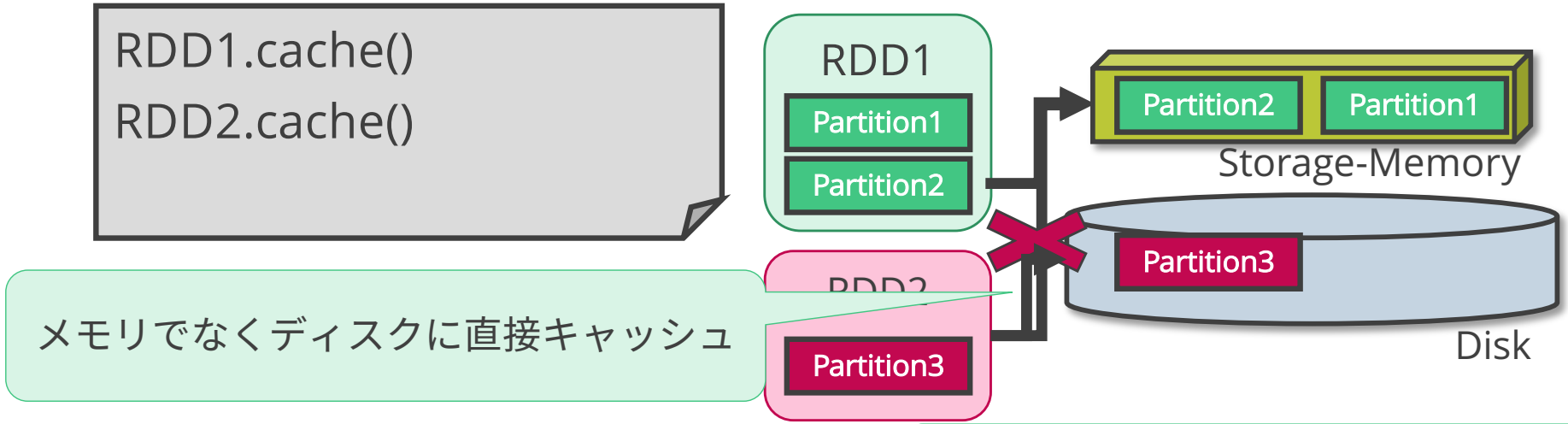
```
RDD1.cache()
RDD2.cache()

RDD1.partition1.read()
RDD1.partition2.read()
RDD2.partition3.read()
```



Modified cache I/O behaviors† [1/2]

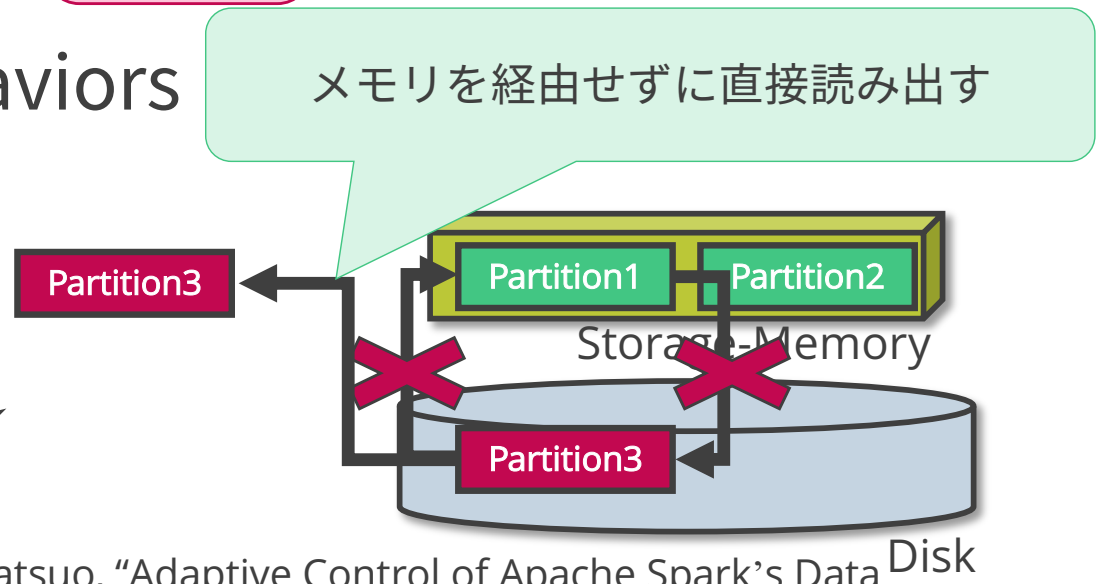
- Modified write I/O behaviors

```
RDD1.cache()
RDD2.cache()
```



- Modified read I/O behaviors

```
RDD1.partition1.read()
RDD1.partition2.read()
RDD2.partition3.read()
```

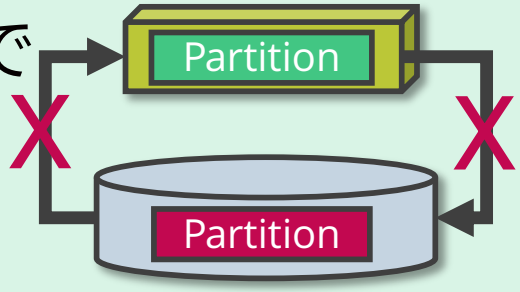


† H. Inagaki, T. Fujii, R. Kawashima, and H. Matsuo, "Adaptive Control of Apache Spark's Data Caching Mechanism Based on Workload Characteristics," *Proc. 5th International Symposium on Big Data Research and Innovation (BigR&I-2018)*, Barcelona, Spain, Aug. 2018.

Modified cache I/O behaviors [2/2]

利点

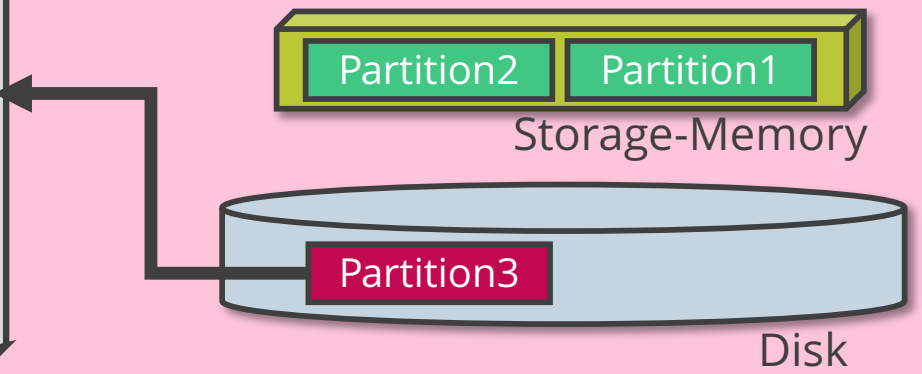
- データの移動が発生しないので
  スラッシングが発生しない



欠点

- アクセス履歴を考慮していない
  - ディスクアクセスが頻発

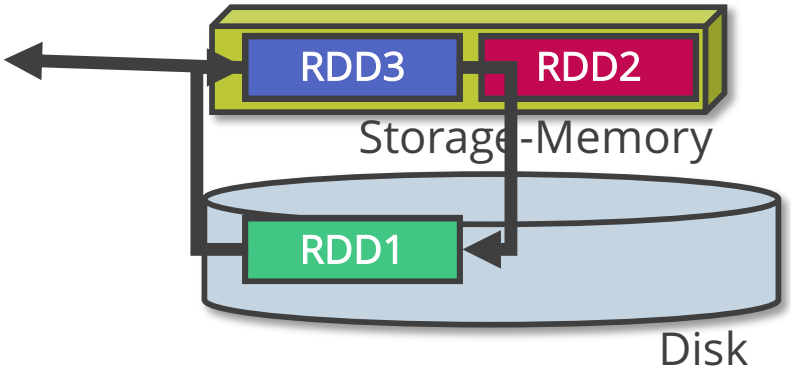
```
RDD1.cache()
RDD2.cache()

for(...){
  RDD2.partition3.read()
}
```



LRC(Least Reference Count)† [1/2]

- LRC(Least Reference Count)
  - RDDの参照カウントを基にドロップするRDDを決定

RDD_ID	参照カウント
RDD1	2
RDD2	3
RDD3	5



利点

- 頻繁に利用されるRDDをより正確に特定できる
  - LRUよりも長期的なアクセスパターンに対応できる

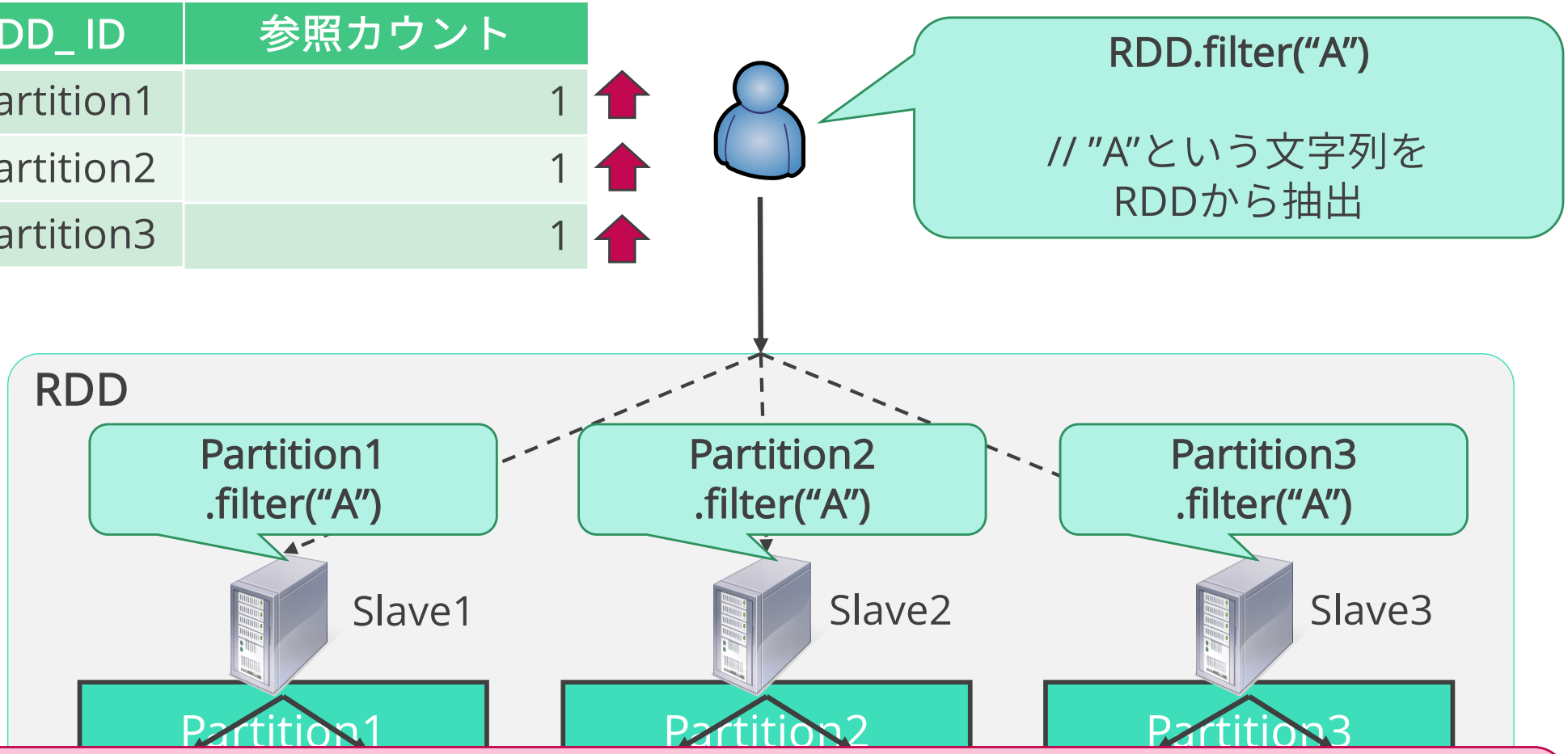
欠点

- 頻繁に利用されるパーティションの特定は不可能

† Y. Yu, W. Wang, J. Zhang, and K. B. Letaief, "LRC: Dependency aware cache management for data analytics clusters," *Proc. IEEE international Conference on Computer Communications (INFOCOM)*, pp. 1–9, Georgia, USA, May 2017.

LRC(Least Reference Count) [2/2]

RDD_ ID	参照カウント
Partition1	1
Partition2	1
Partition3	1

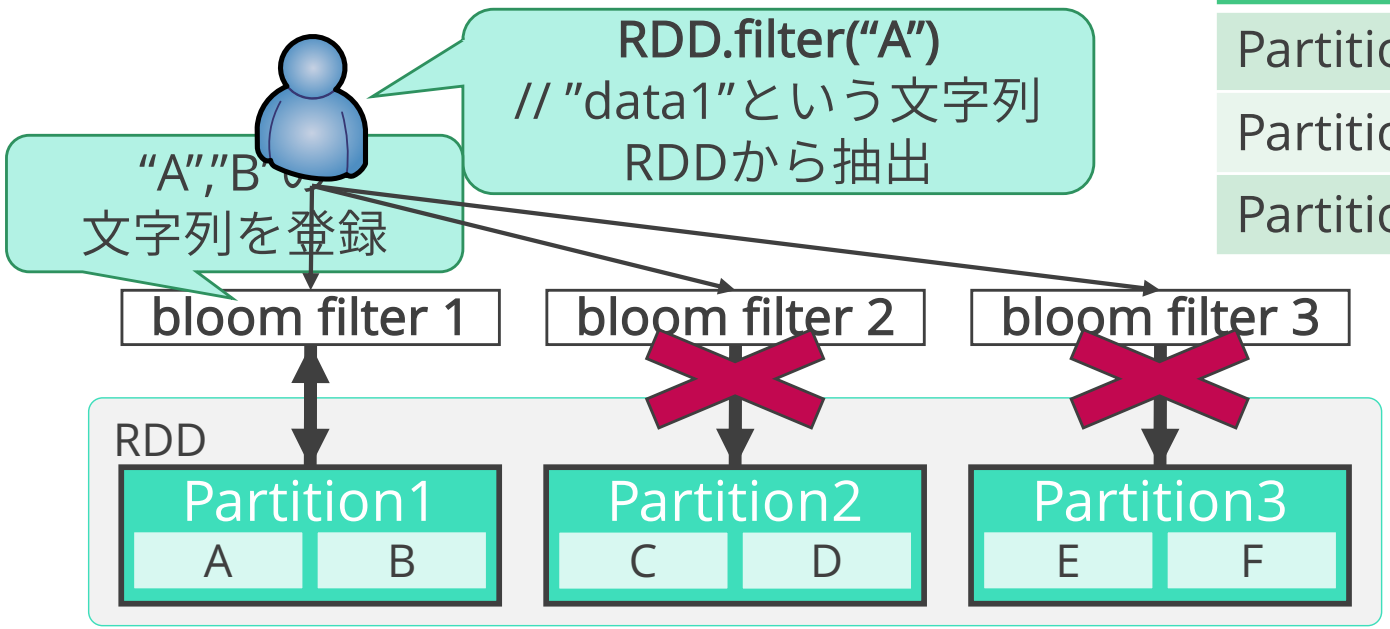


頻繁に利用されるパーティションの特定が困難

提案手法の概要

- **目的: 低オーバーヘッドで枝切りする手法の検討**
- **既存のキャッシュ管理手法**に**ブルームフィルタ**を導入
  1. 処理するパーティション数を削減
  2. より正確な参照カウントを記録することで
     頻繁に利用されるパーティションを特定

RDD_ ID	参照カウント
Partition1	1
Partition2	0
Partition3	0



評価環境

- 既存のキャッシュ管理手法にブルームフィルタを導入
  - ブルームフィルタの有効性を検証
  - 以下のキャッシュ管理手法において性能を評価
    - LRU
    - LRC
    - LRU + ブルームフィルタ(BFLRU)
    - LRC + ブルームフィルタ(BFLRC)

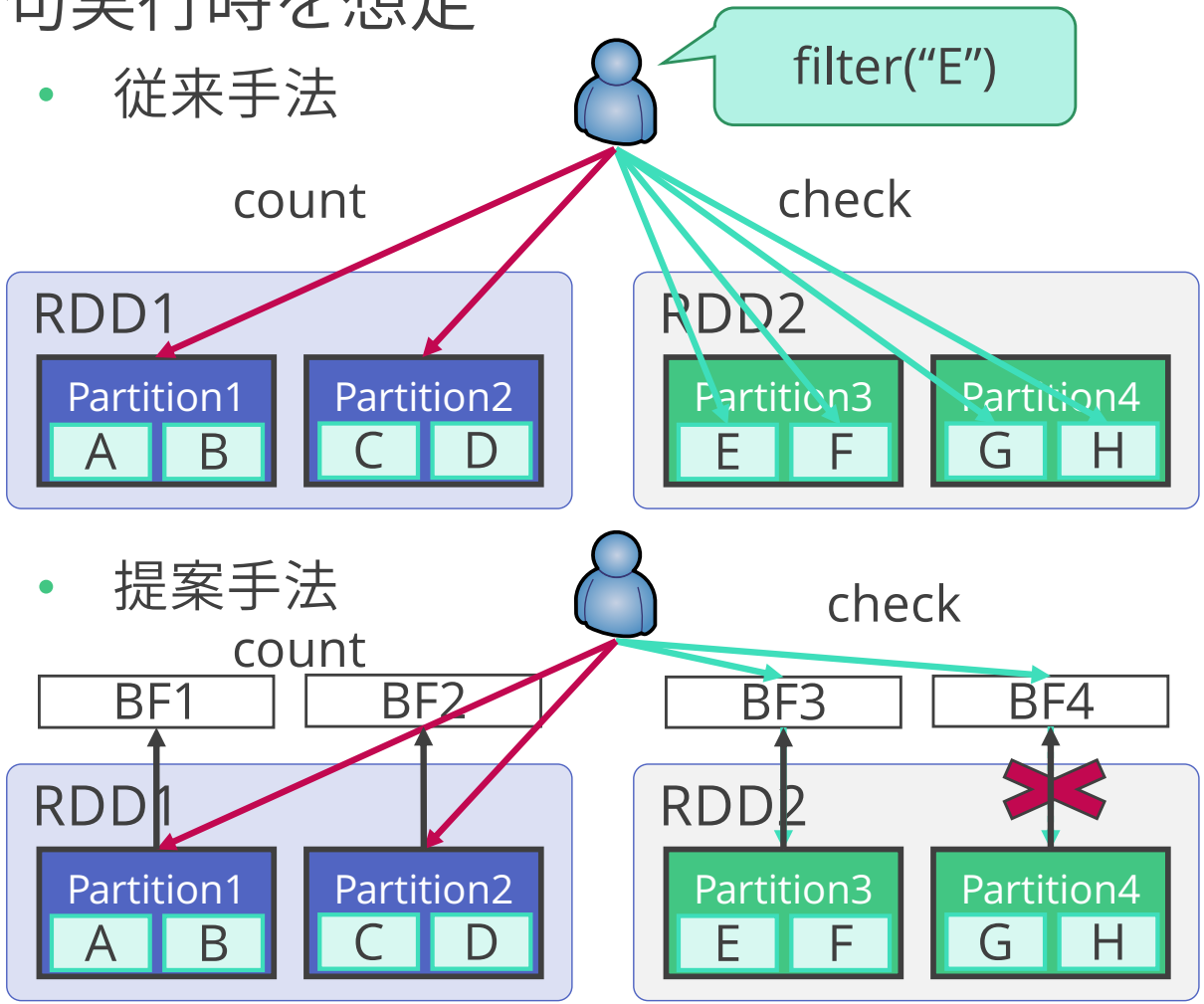
評価環境	
OS	Ubuntu 16.04
CPU	Core i5 4460 (3.20 GHz), 4 cores
メモリサイズ	1.0 GB size of storage-memory: 0.4 GB
HDD	1 TB
Spark バージョン	2.2.1
マシン構成	Master x1, Slave x1
ファイルシステム	HDFS (version 2.7.4)

- Filter-testベンチマーク
  - キャッシュするRDDのサイズ(RDD1,RDD2): 1GB
  - RDDのパーティション数:1,000

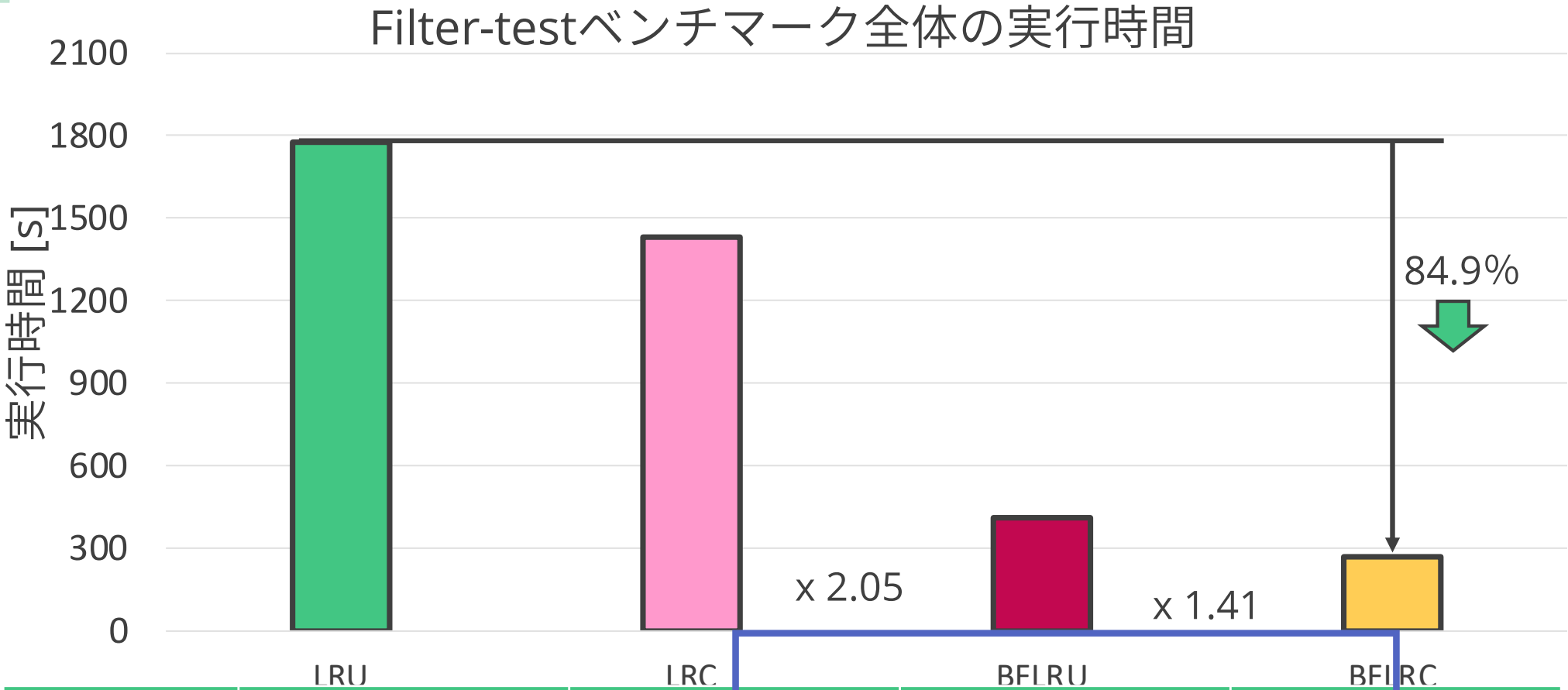
Filter-testベンチマーク

- 局所的なアクセスが頻繁に発生する状況下で性能評価
  - SQLにおけるWHERE句実行時を想定

```
RDD1.io.load(hdfs)
RDD1.cache()

RDD2 = RDD1.map{ ... }
RDD2.cache()

for (i <- 1 to 10) {
  for (j <- 1 to 10) {
    RDD2.filter(...).count
  }
  RDD1.count()
}
```



評価結果



	LRU	LRC	BFLRU	BFLRC
RDD2 cache	57 s	27 s	117 s	38 s
RDD2 filter	16 s	11 s	0.9 s	0.5 s

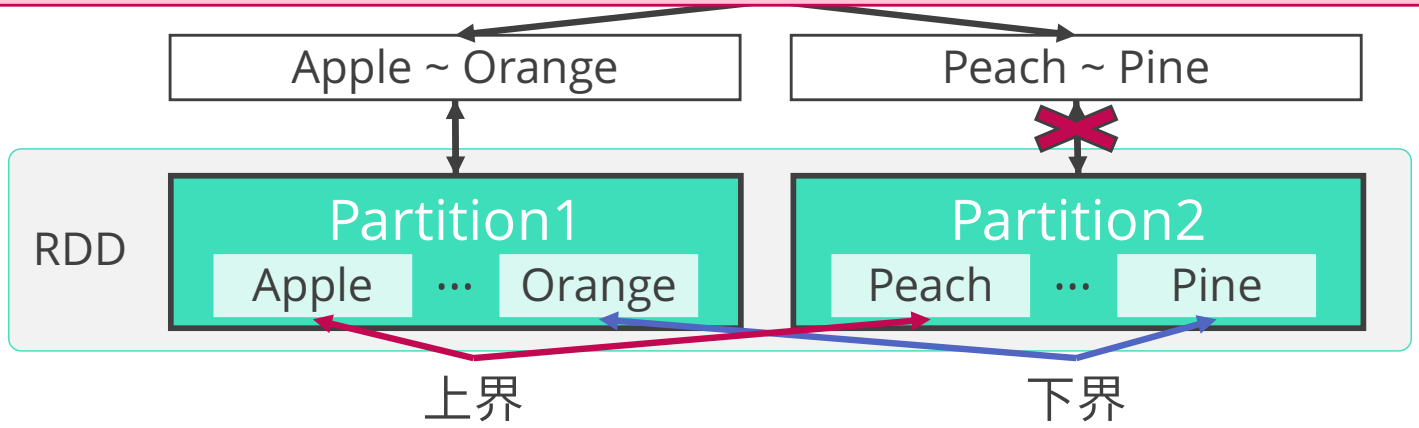
x 0.045

- filter-testベンチマーク
  - RDD1: cache x1 + count x10
  - RDD2: cache x1 + filter x100

SparkからSpark SQLへの移行

- Partition Pruning
  - 対象のデータが含まれないことが保証される場合のみ、処理対象から外す機能
- 上界/下界を用いた判定により処理するデータを剪定
  - Partition1,2からBananaという単語を検索

データセット内に含まれるデータに
極端な偏りが存在する場合にしか効果なし
➢ **選定基準をブルームフィルタを用いた手法に変更**



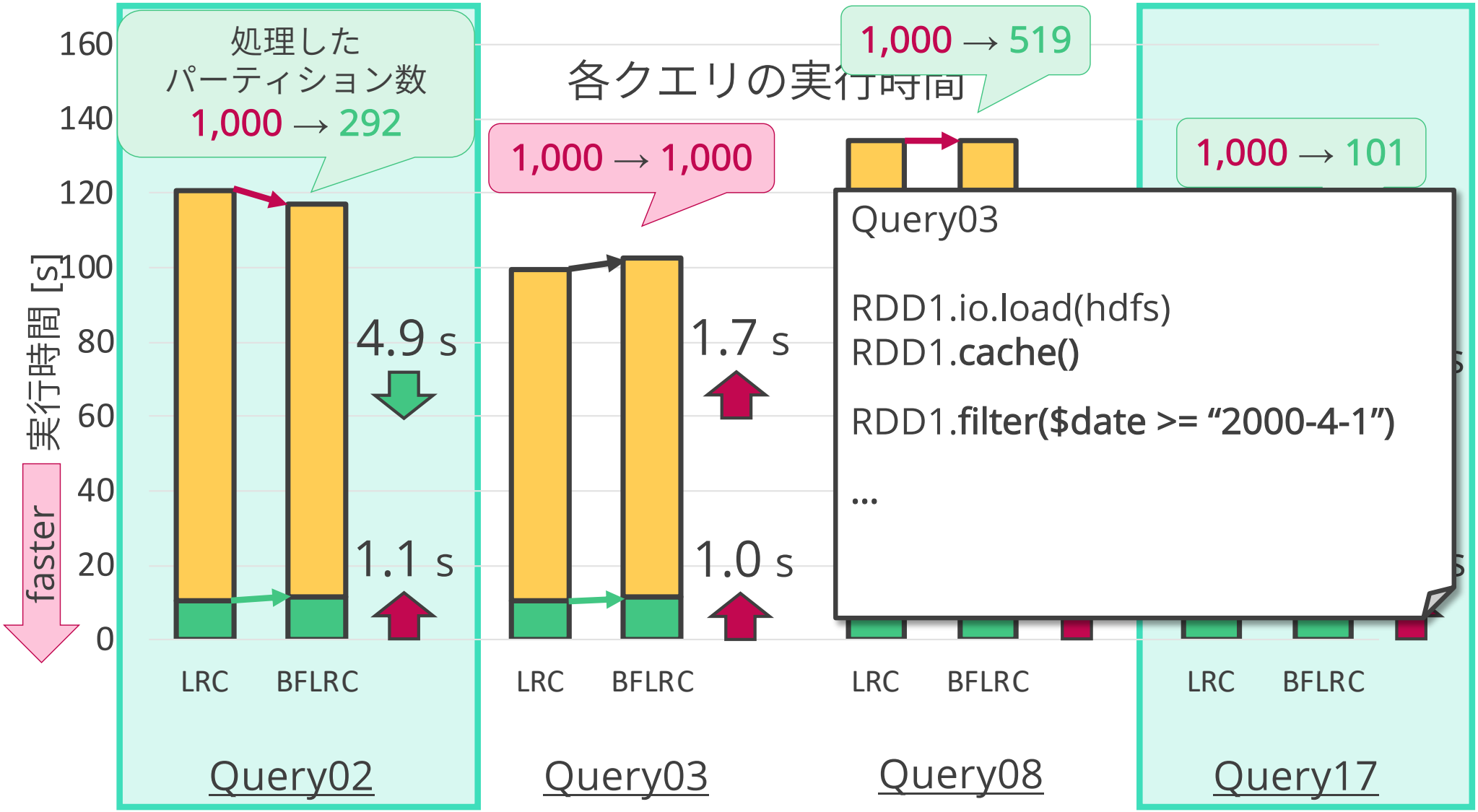
TPC-Hベンチマーク

- TPC-Hベンチマークを用いて提案手法の有効性を評価
  - filter関数時に処理したパーティション数の評価
  - 以下のキャッシュ管理手法において性能を評価
    - LRC + Partition Pruning
    - LRC + ブルームフィルタ(BFLRC)
- 評価に使用するデータセット
  - データ数
    - 100,000
  - パーティション数
    - 1,000
  - データの生成
    - random

評価環境	
OS	Ubuntu 16.04
CPU	Core i5 4460 (3.20 GHz), 4 cores
メモリサイズ	1.0 GB size of storage-memory: 0.4 GB
HDD	1 TB
Spark バージョン	2.2.1
マシン構成	Master x1, Slave x1
ファイルシステム	HDFS (version 2.7.4)

評価結果

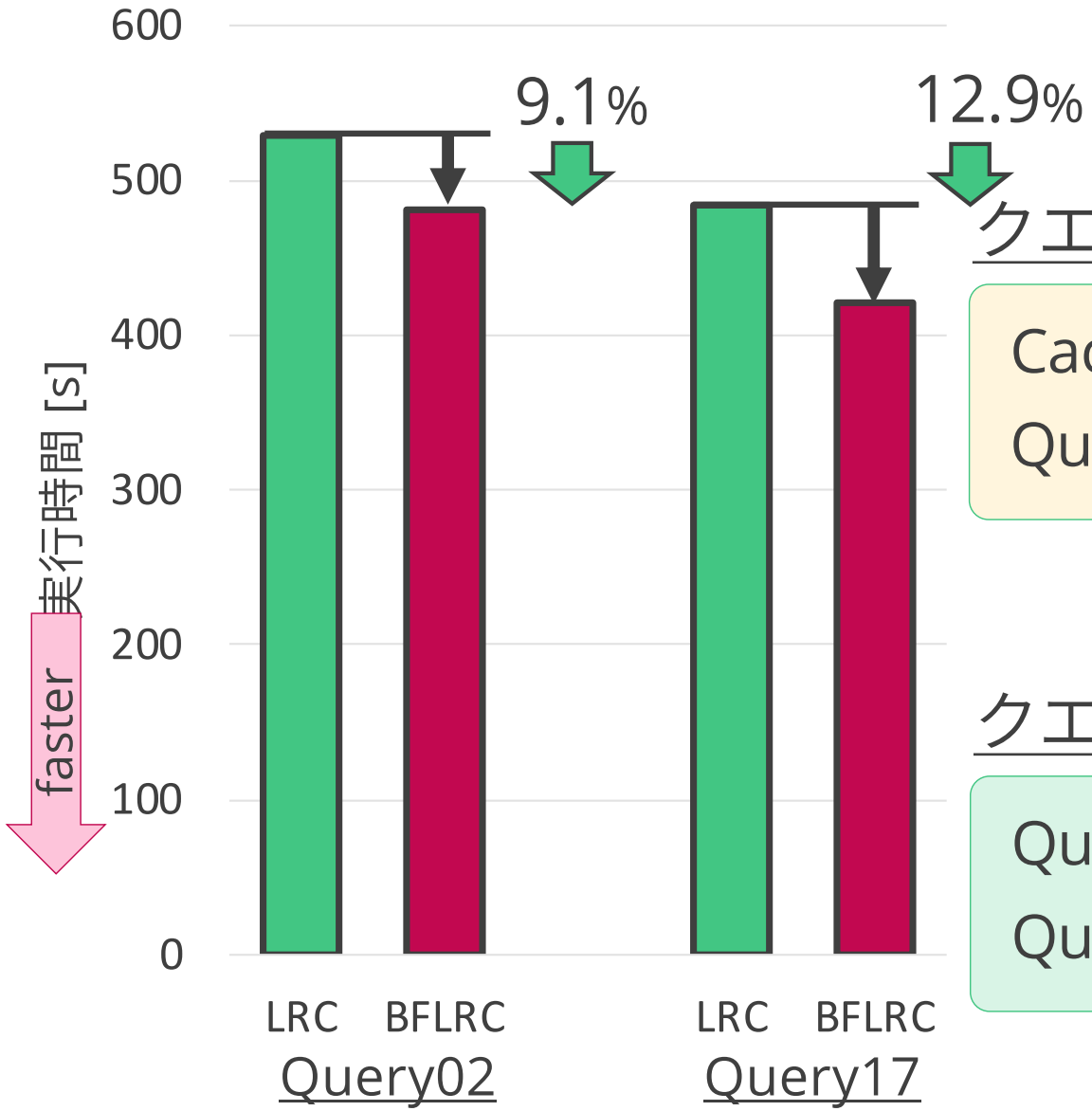
- Filter関数を含む4つのクエリで比較



Cache処理　本処理

評価結果 (2) 複数回クエリ実行

- Query02, 17を10回実行した時の時間



クエリを1回実行

Cache: 19.2% 実行時間**増加**

Query: 6.5% 実行時間**削減**

クエリを10回実行

Query02: 9.1%

Query17: 12.9%

実行時間**削減**

まとめ

- まとめ
  - ブルームフィルタを用いた、処理データの剪定手法を提案
    - 処理するパーティション数を削減
    - 頻繁に利用されるパーティションを検出可能
  - 実用的な環境を想定したTPC-Hベンチマークでは、実行時間を最大12.9%削減
- 今後の課題
  - ユーザが設定したパーティション数によっては、性能を十分に出し切れない可能性がある
    - ➢ユーザの設定したパーティション数に依存しない手法の検討