分散KVSに基づく MapReduce 処理系SSS

中田秀基、小川宏高、工藤知宏
独立行政法人産業技術総合研究所

背景

- MapReduce の普及
  - Apache Hadoopの普及による
  - 単発のMRではないアプリケーションの登場

- MapReduce 処理系
  - 大規模データ処理 - Hadoop
    - 繰り返し実行が低速
  - 共有メモリオンメモリデータ処理 – Phoenix, Metis
    - シングルノード、もしくはSMPが対象
    - メモリサイズが問題を制約

  - SSS [Ogawa, MapReduce11]: 両者の間をねらう
    - クラスタ上で動作
    - ディスクを利用 ― メモリ量の制限なし
    - 高速な実行

研究の目的

- 分散KVSをベースとしたMapReduce処理系 SSS の設計と実装

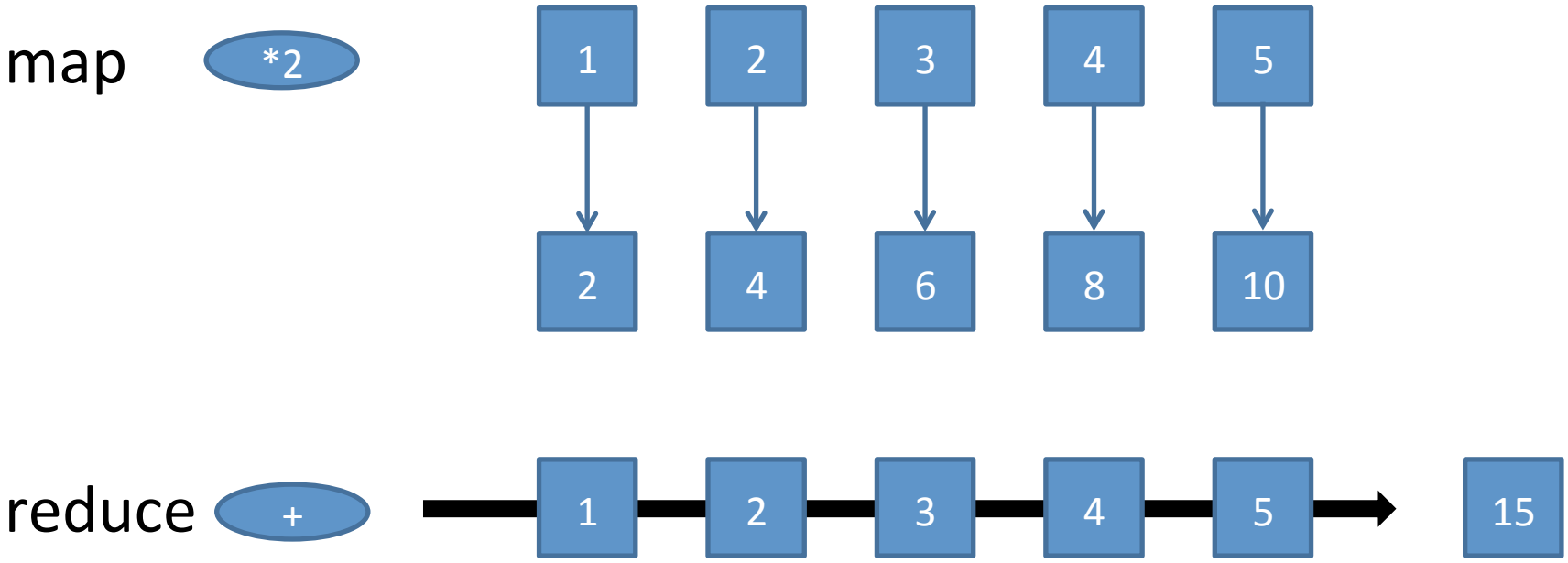
- SSS の評価
  - Hadoopと比較
  - 合成ベンチマーク[小川: HPC129] を利用
    - Read/Write/Shuffle 各フェイズでの動作特性
  - K-meansによる評価

発表の概要

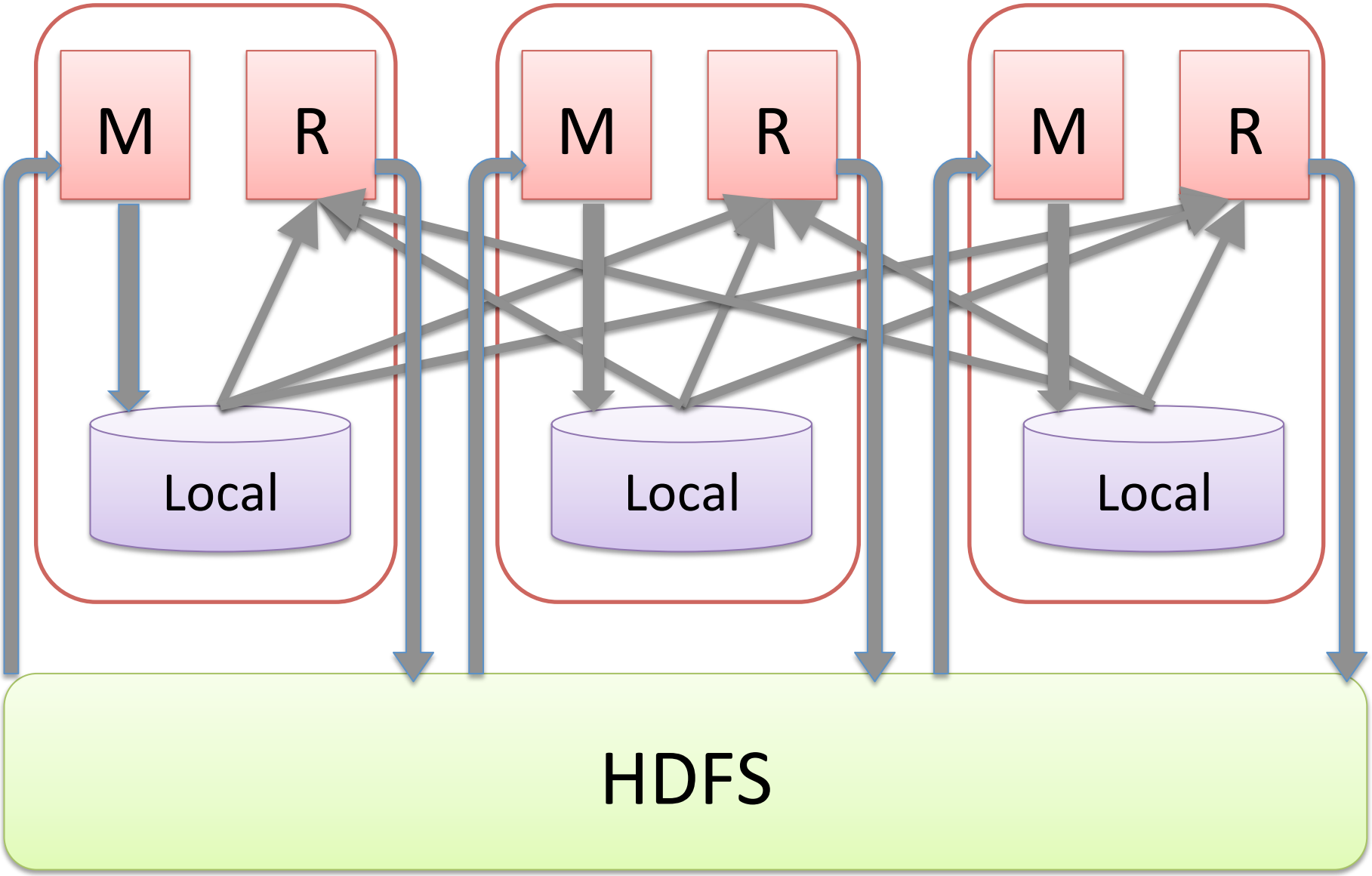
- MapReduce / Hadoop の概要
- SSSの設計と実装
- 合成ベンチマークによる評価
- K-meansによる評価
- まとめと今後の課題

MapReduceとは

高階関数を持つ言語に一般的なmapとreduce関数にヒント



Hadoopの概要



Hadoopの概要

- HDFSから読んでHDFSに書き込み
  - 繰り返し処理が遅い
- MapとReduceのデータフローが異なる
  - MapとReduceを対で使わなければならない

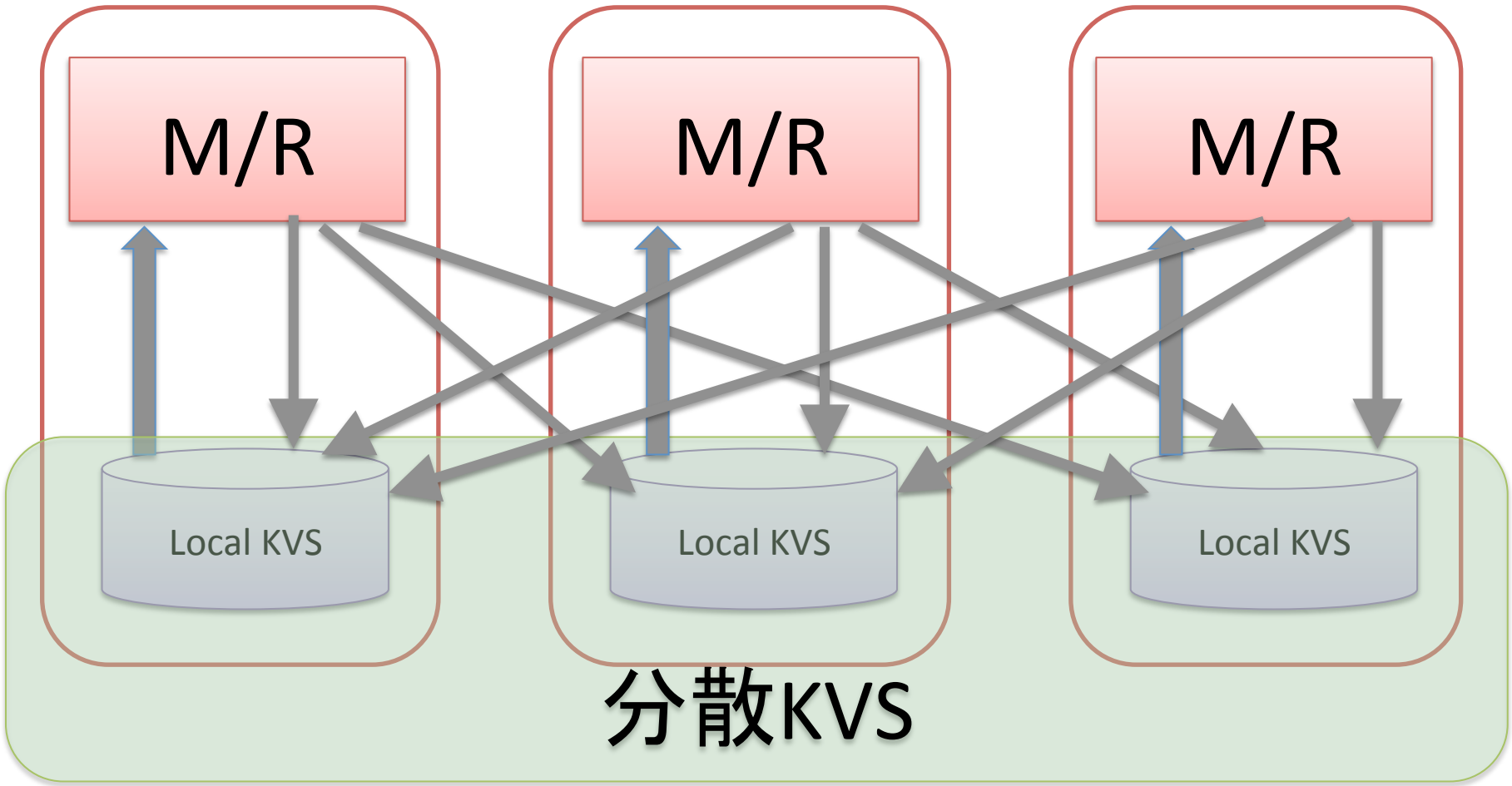
SSSの提案（1）

- より広範なアプリケーションへの対応
  - ジョブ起動オーバヘッドの低減
  - Map/Reduceの組み合わせを自由に
    - 複雑なワークフローを
- 分散ファイルシステムは必須なのか？
  - 処理対象はキーバリューペア
  - 毎回フラットなデータからパースするのは馬鹿らしい
  - とはいえ、シーケンシャル read/write でないと性能がでない

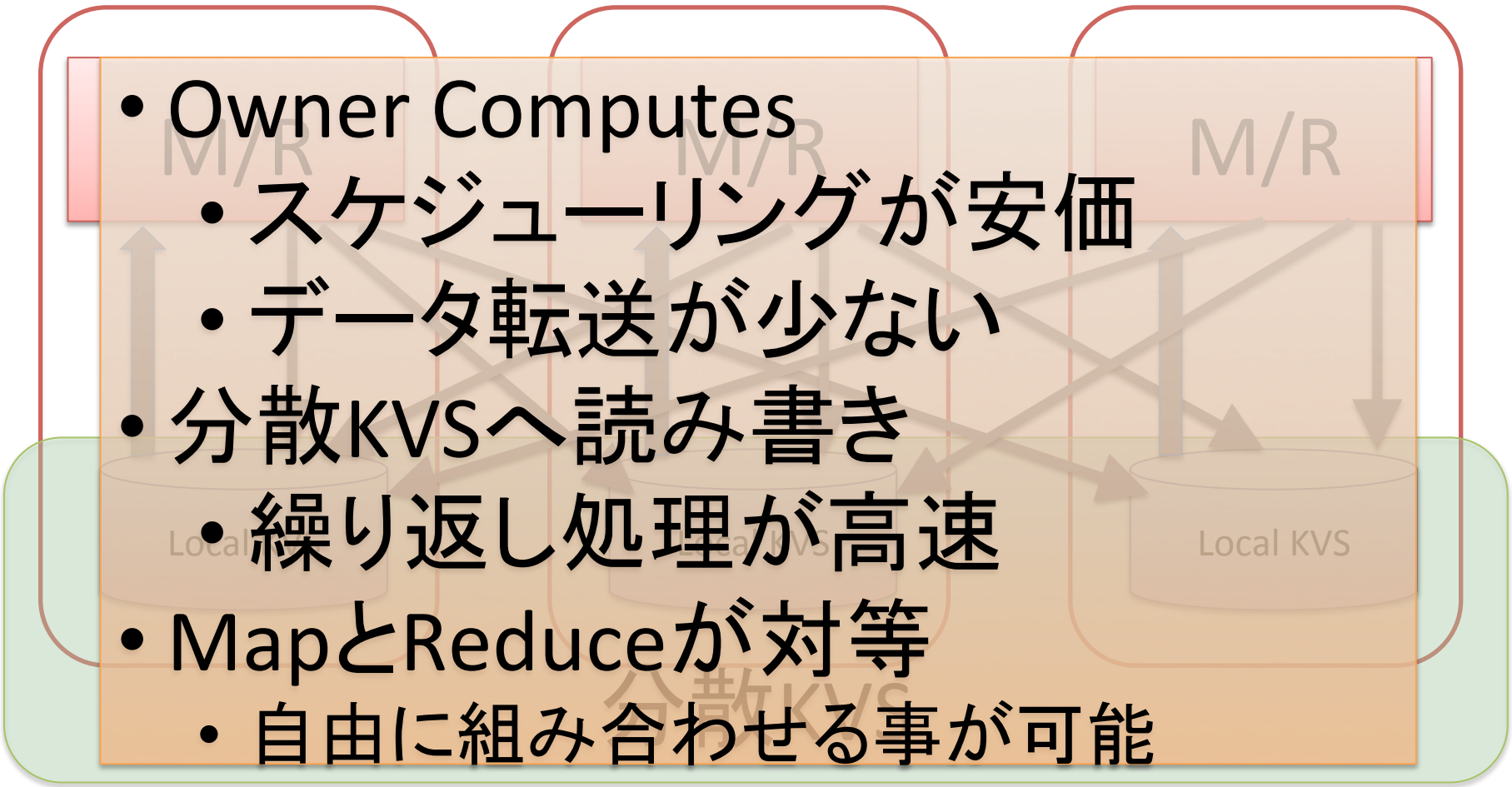
SSSの提案（2）

- SSDの普及
  - ioDriveなどのPCI-bus直結型は非常に高速
  - ランダムread/writeでも性能が低下しない
  - → 従来と異なる構成が可能
- 分散KVSを基盤としたMapReduceシステム
  - SSDを前提
  - 柔軟なMap/Reduceによるワークフローを実現

SSSの概要



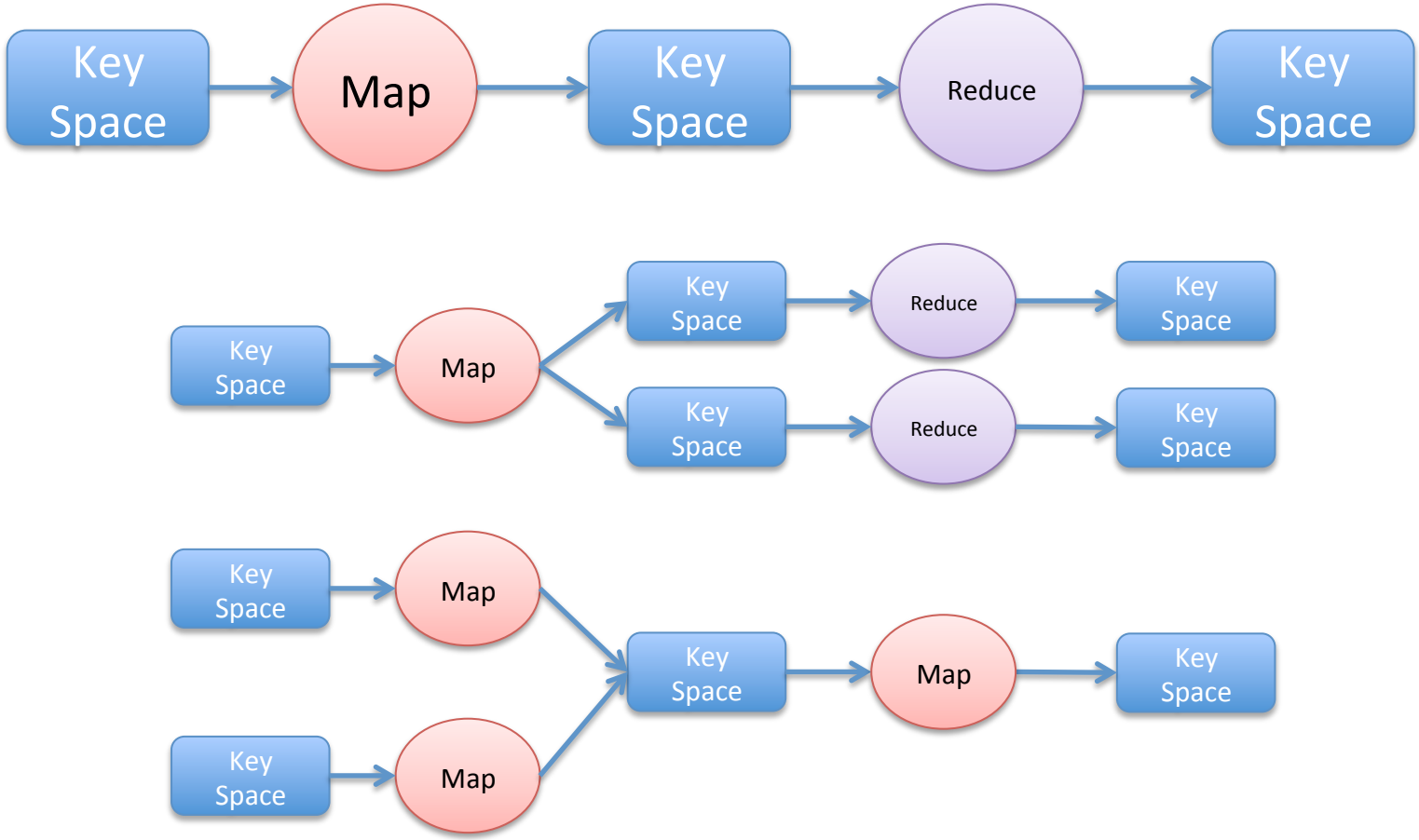
SSSの概要



- Owner Computes
  - スケジューリングが安価
  - データ転送が少ない
- 分散KVSへ読み書き
  - 繰り返し処理が高速
- MapとReduceが対等
  - 自由に組み合わせる事が可能

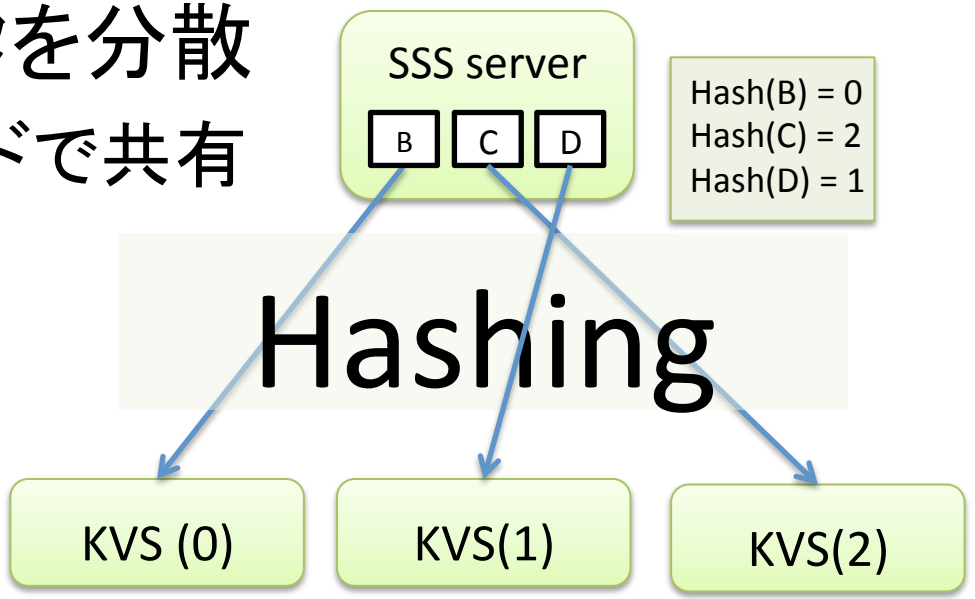
Key Space

- Key Space – 処理対象のデータ集合
- Map と Reduce の処理は殆ど同じ
- 自由に組み合わせることが可能



分散KVSの実装

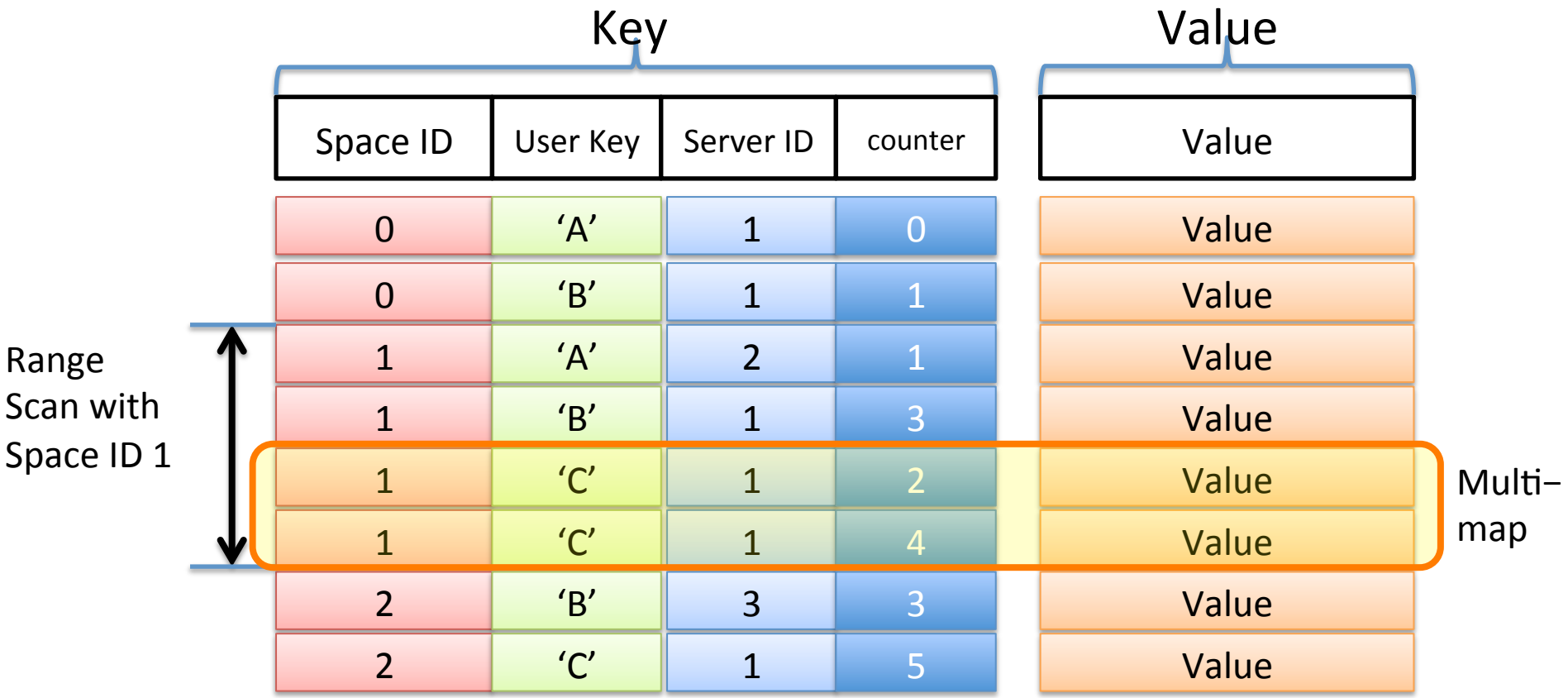
- キーのハッシュでデータを分散
  - ハッシュをすべてのノードで共有
  - 名前サーバはなし



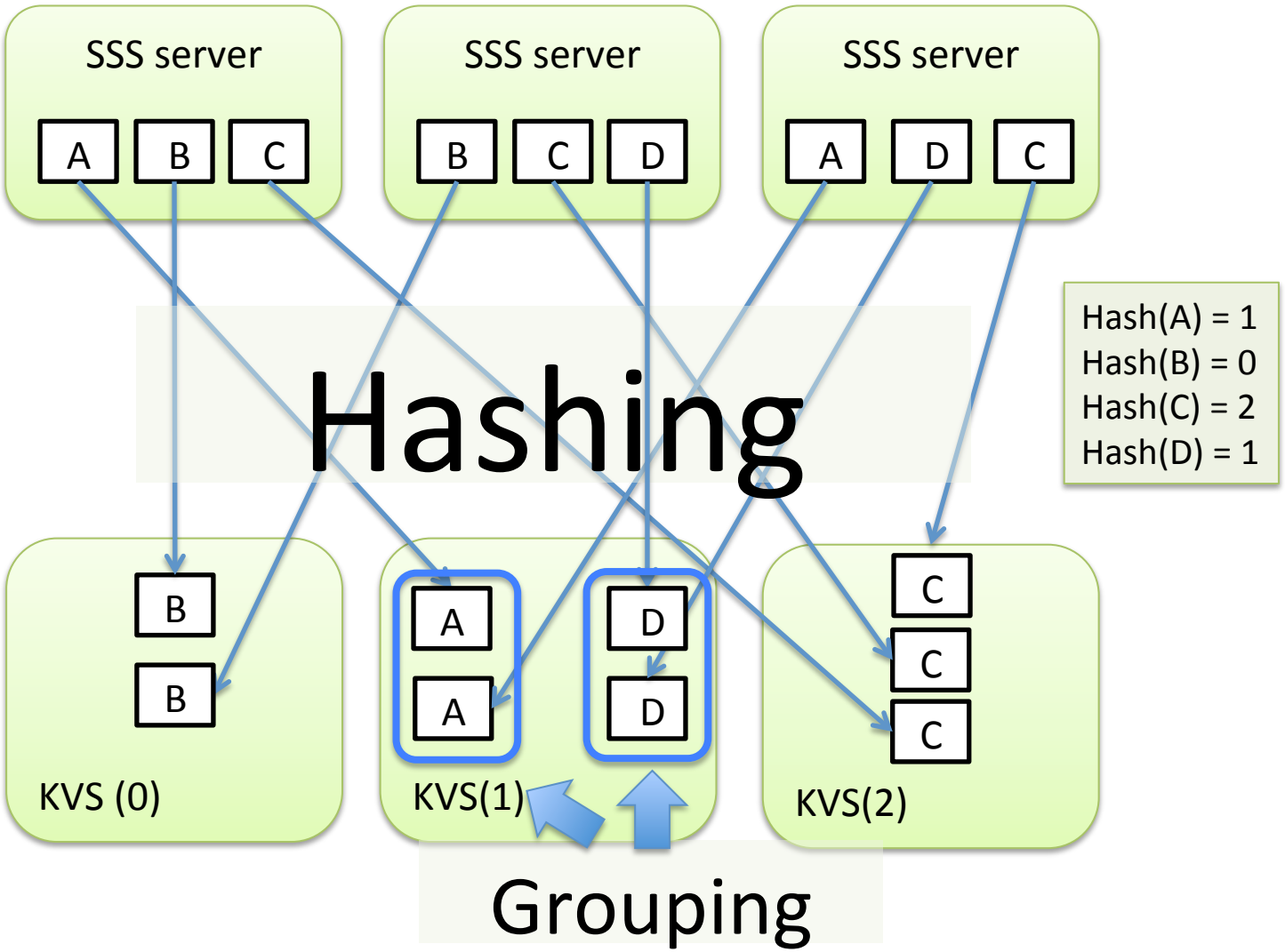
- 要素KVS
  - Tokyo Cabinetを利用
    - アクセスパターンに最適化
    - ソートされたデータのバルク書き込み
    - レンジ読み出し

分散KVSの実装（2）

- キーのエンコーディングによって実現
  - Key Space
  - Multi-map – 一つのKeyに対して複数の値



シャッフルの実現

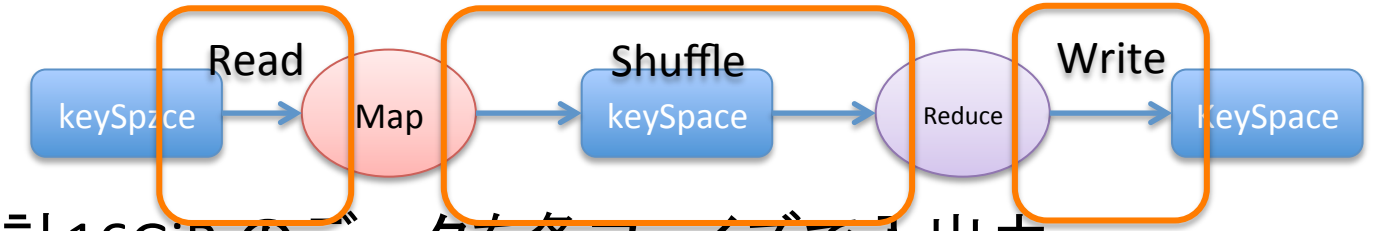

SSS server
A
B
C
SSS server
B
C
D
SSS server
A
D
C
Hash(A) = 1
Hash(B) = 0
Hash(C) = 2
Hash(D) = 1
Hashing
B
B
KVS (0)
A
A
D
D
KVS(1)
C
C
C
KVS(2)
Grouping

評価

- SSS の性能を計測
  - Hadoopと比較
  - 合成ベンチマーク[小川: HPC129] を利用
    - Read/Write/Shuffle 各フェイズでの動作特性
  - K-means
    - アプリケーションでの動作特性

合成ベンチマークの構成

- 読み込み、書き出し、シャッフルのデータ数、データ量を独立して制御することが可能



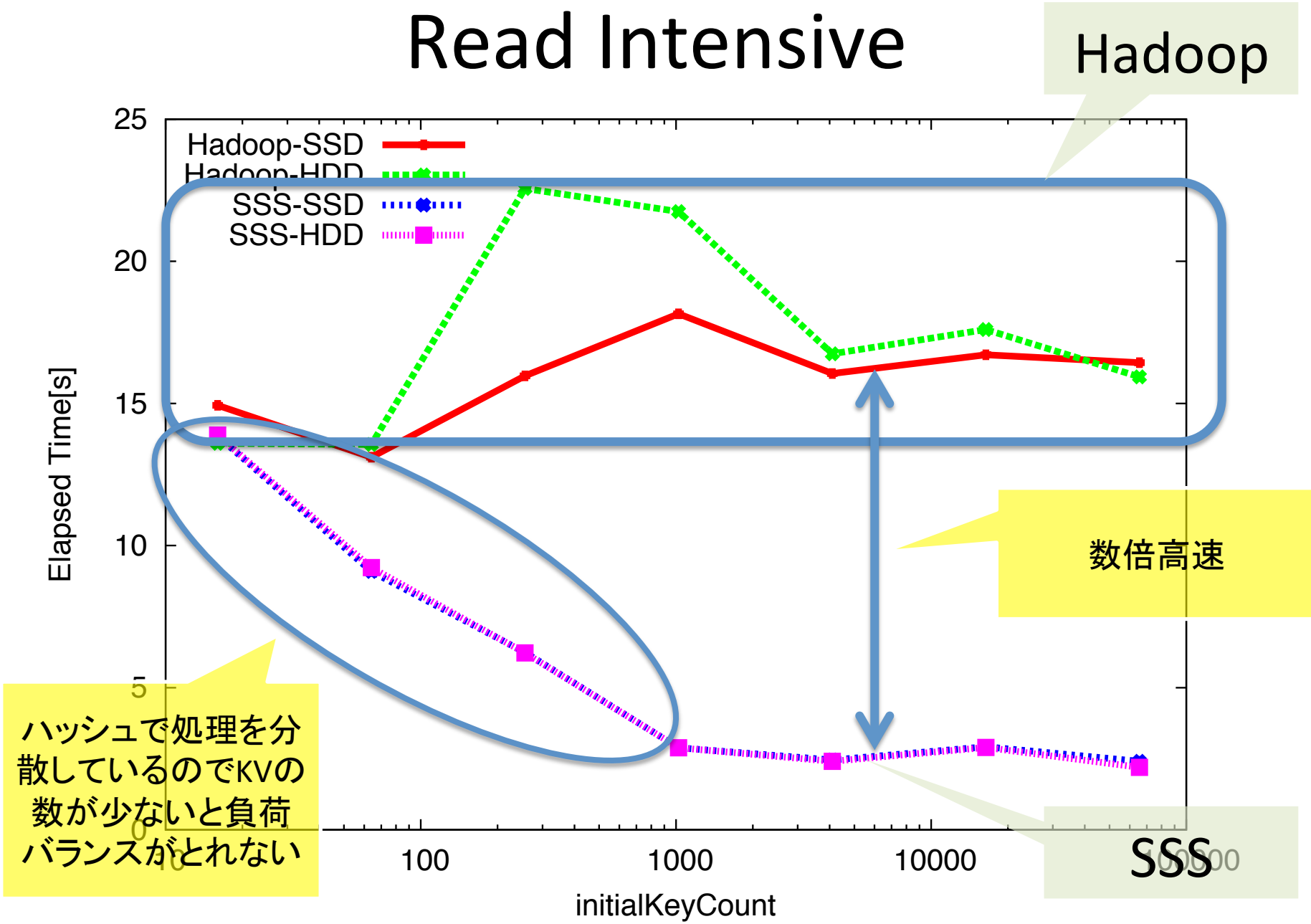
- 総計16GiB のデータを各フェイズで入出力
  - 個々のKVペアのサイズを変更、個数で総データ量を調整

サイズ	1GiB	256MiB	64MiB	16MiB	4MiB	1MiB	256KiB
個数	16	64	256	1Ki	4Ki	16Ki	64Ki

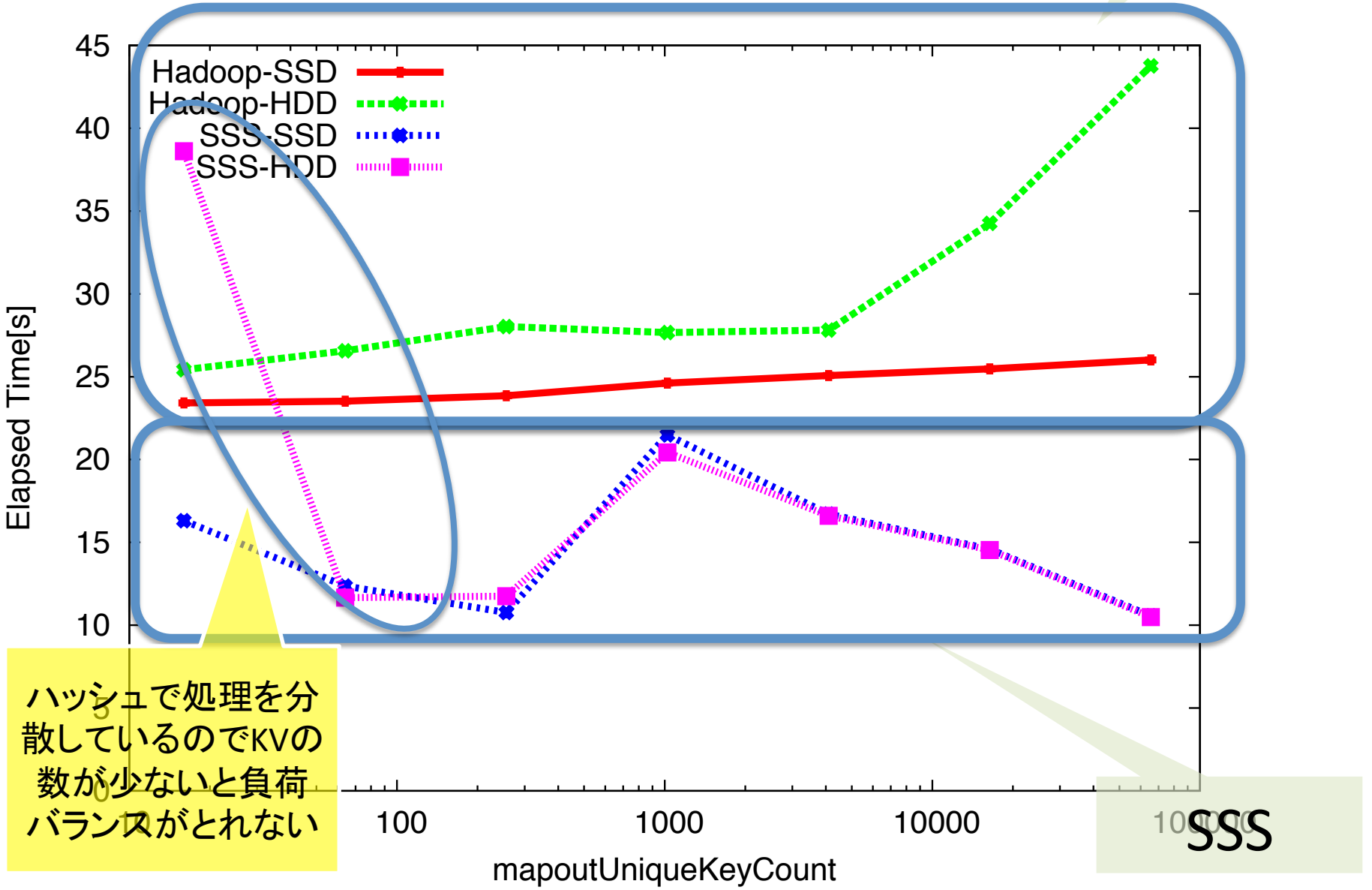
評価環境

- クラスタを使用
  - Number of nodes: 16 + 1 (master)
  - CPUs per node: Intel Xeon W5590 3.33GHz x 2
  - Memory per node: 48GB
  - OS: CentOS 5.5 x86_64
  - Storage: Fusion-io ioDrive Duo 320GB
  - NIC: Mellanox ConnectX-II 10G
- ソフトウェア
  - SSS
  - Hadoop 0.20.2
    - HDFSレプリカ数を1に設定
    - Nodeあたりのmapper数7

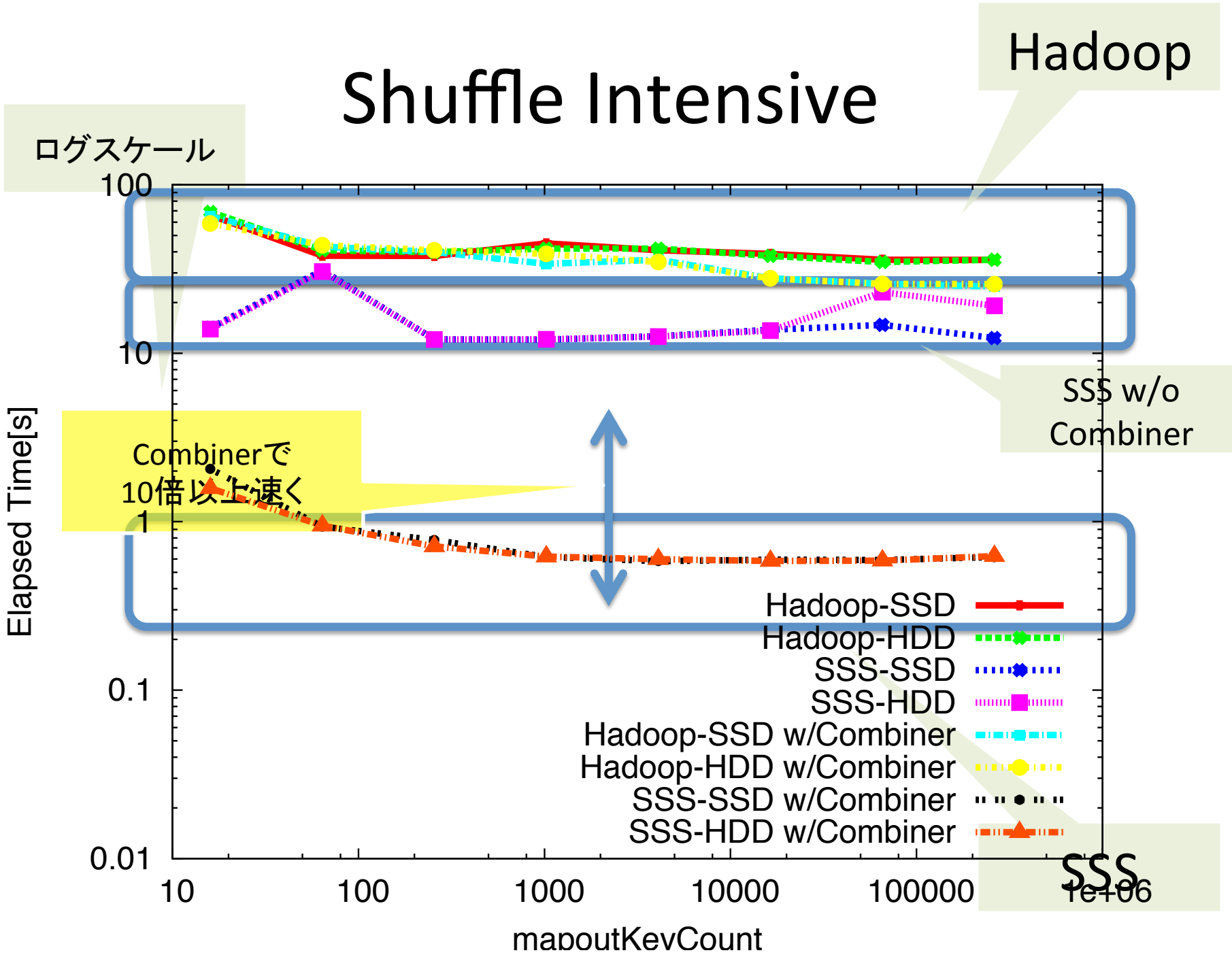
Read Intensive



Write Intensive



Shuffle Intensive



議論

- 殆ど常にSSSのほうが高速
  - SSDは効果なし
- キーの数が少ない場合には、データがうまくハッシュで分散されないためSSSが低速になる場合がある

- データ量が小さすぎる？
  - 16GiB -> ノードあたり1G
  - ディスクキャッシュに乗ってしまう

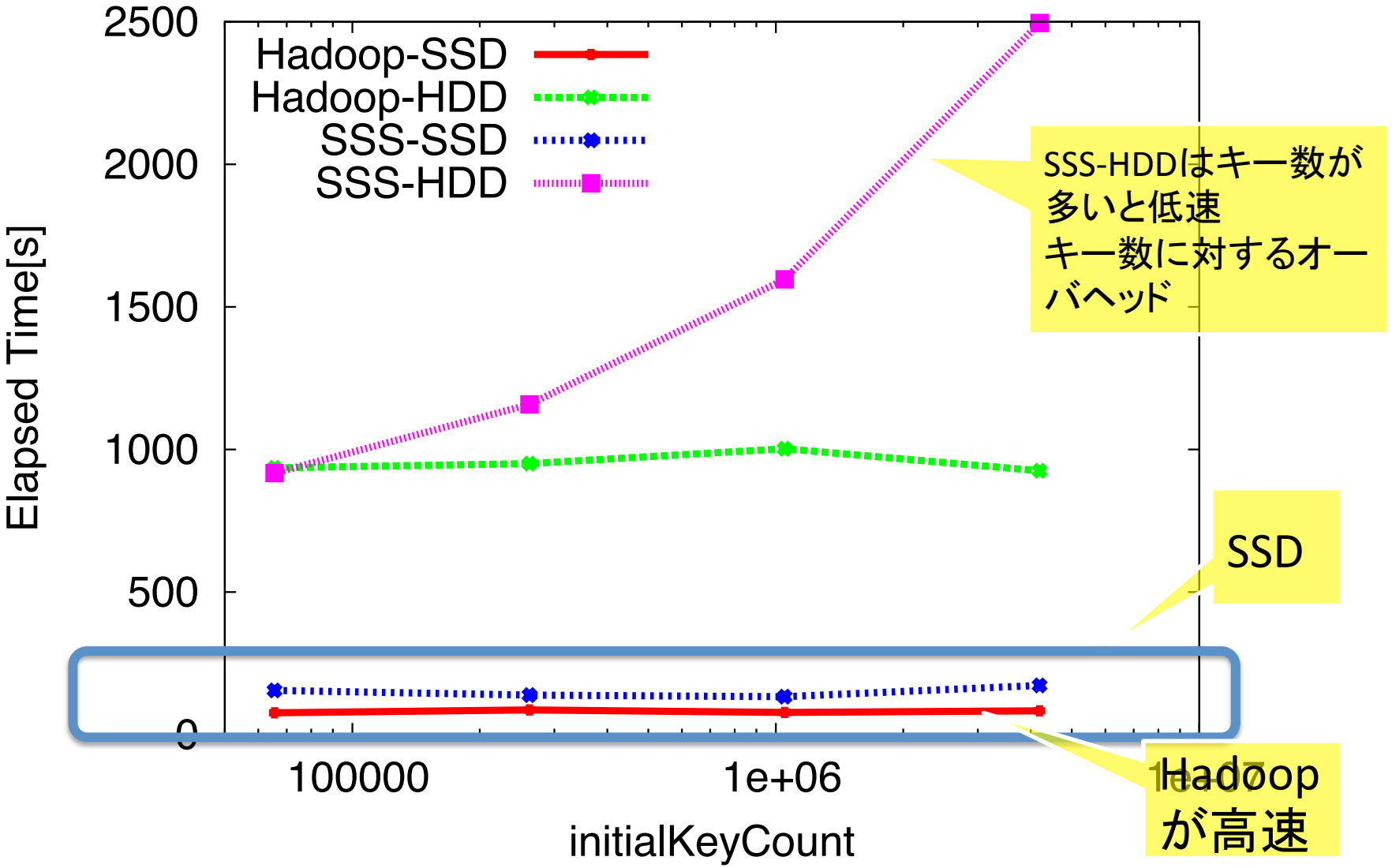
合成ベンチマークの構成 (2)

- 総計1TiB のデータを各フェイズで入出力
  - ノードあたり64GiB – メモリに乗らない

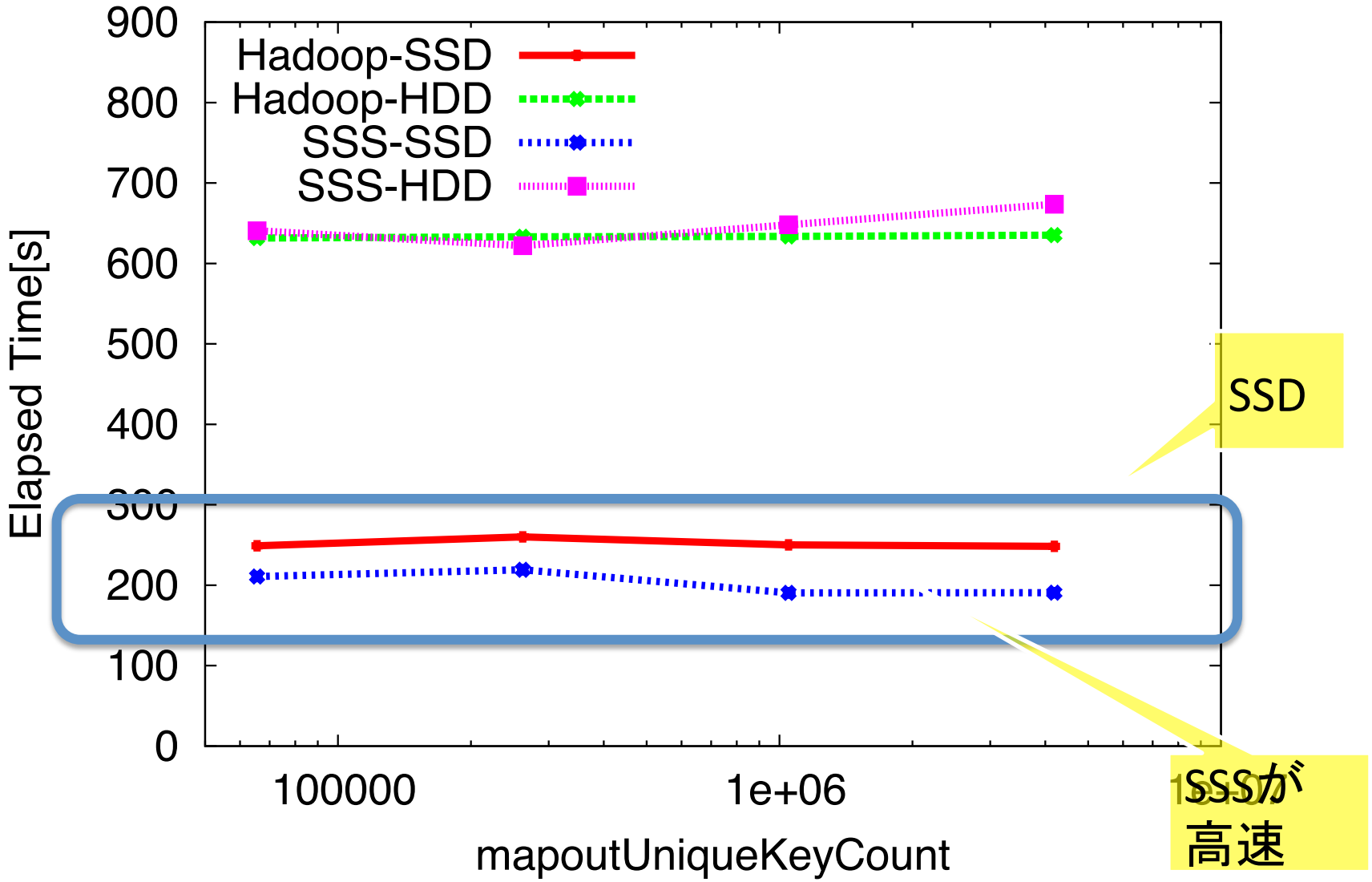
サイズ	1GiB	256MiB	64MiB	16MiB	4MiB	1MiB	256KiB
個数	16	64	256	1Ki	4Ki	16Ki	64Ki

サイズ	16MiB	4MiB	1MiB	256KiB
個数	64Ki	256Ki	1Mi	4Mi

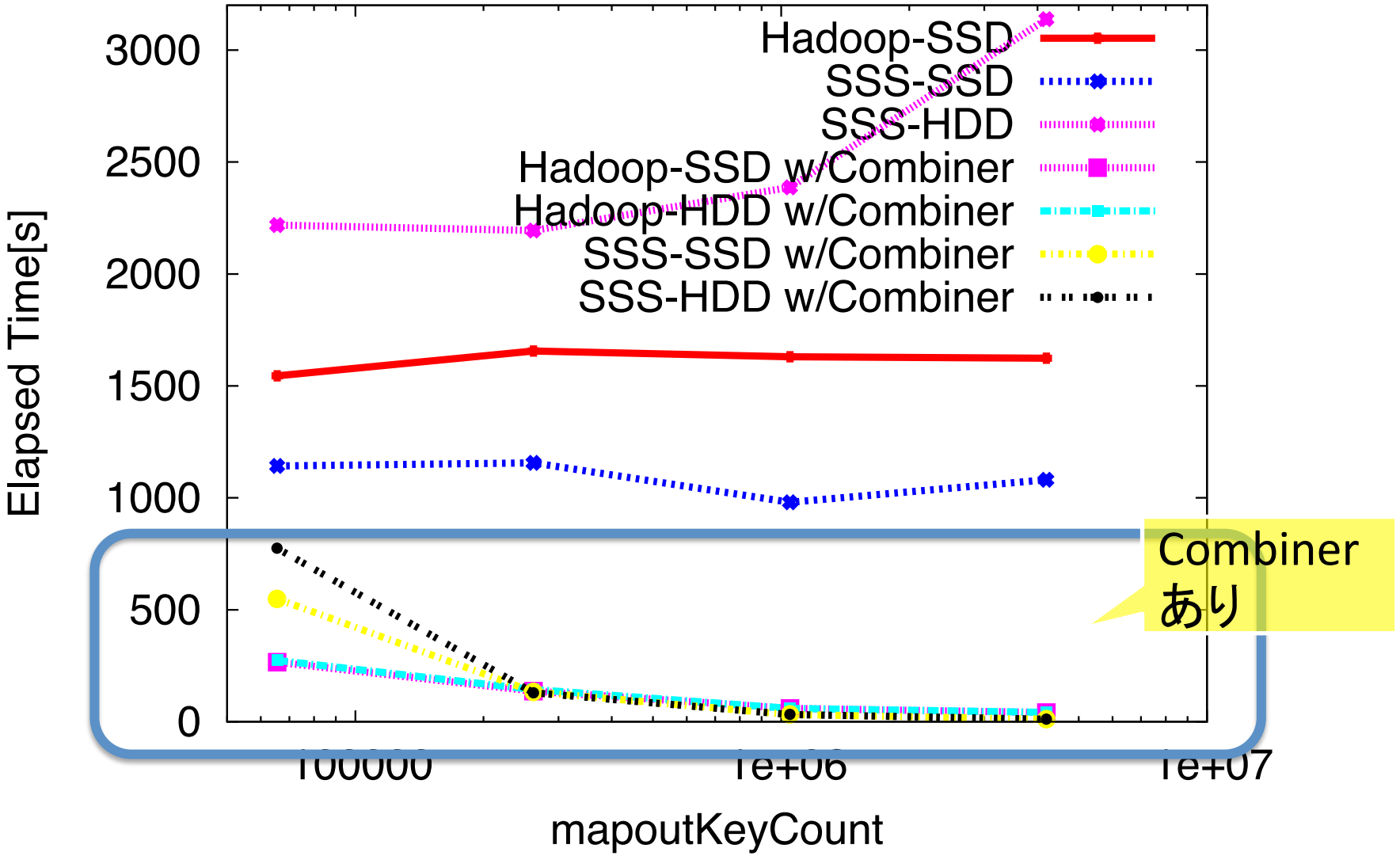
Read Intensive



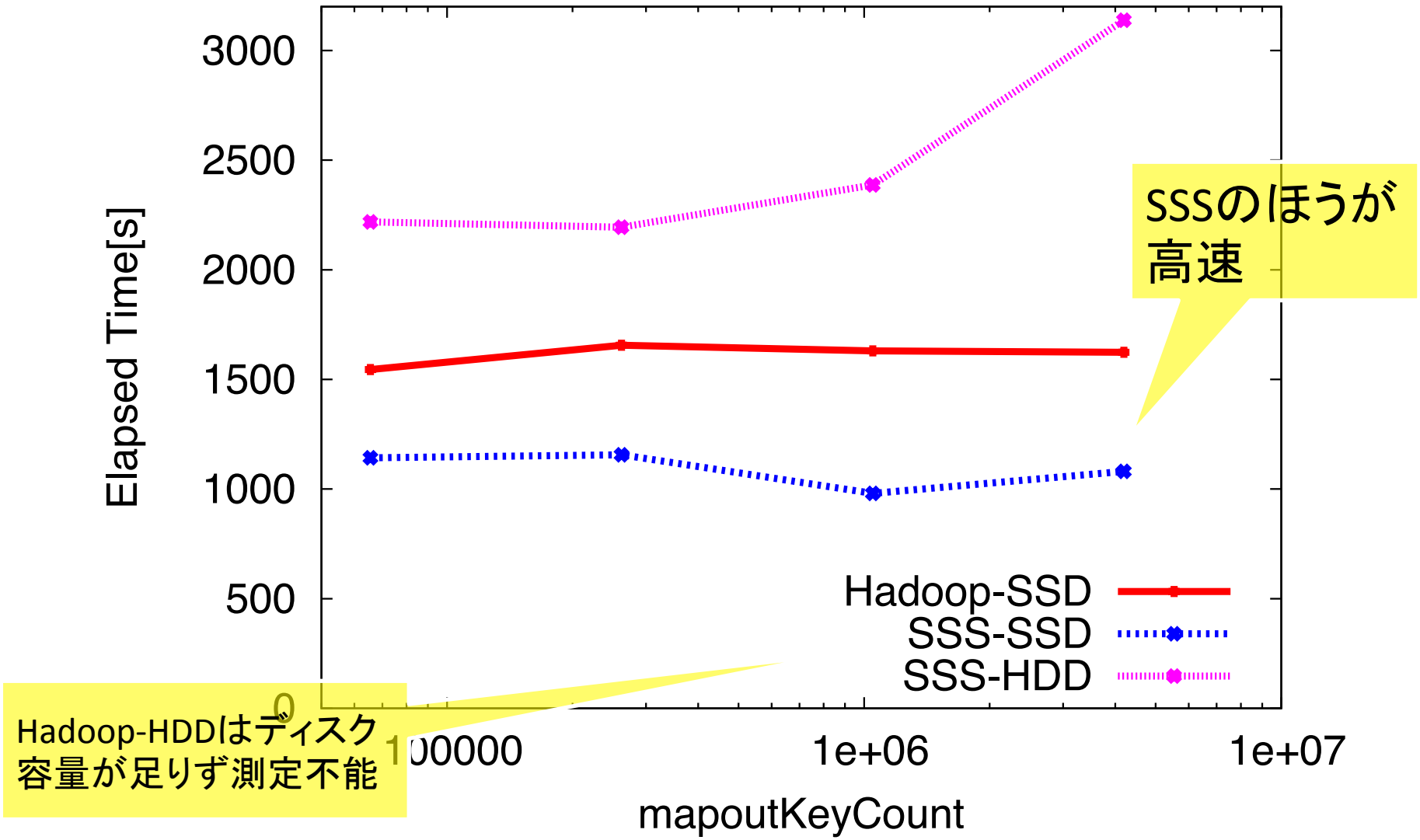
Write Intensive



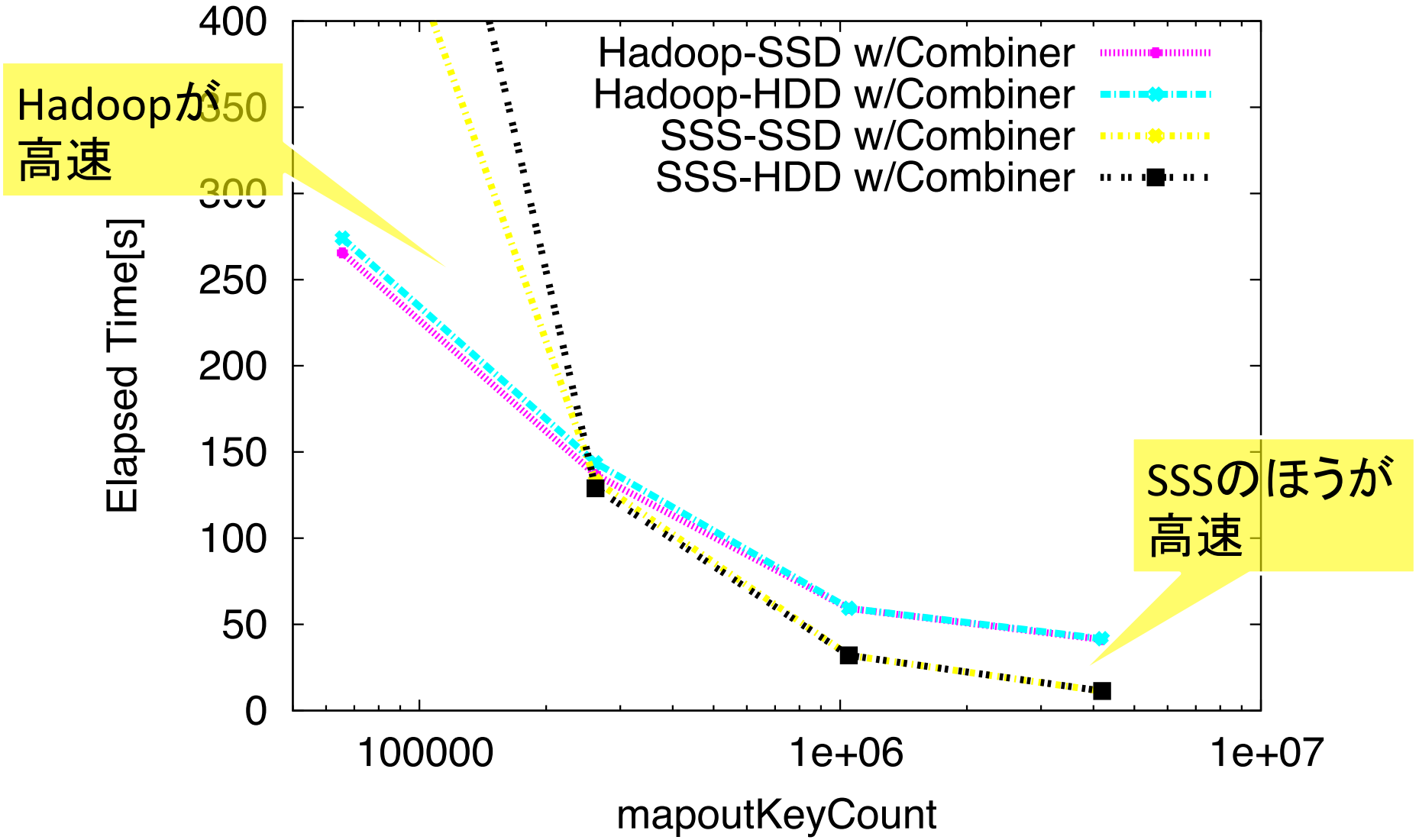
Shuffle Intensive



Shuffle Intensive (combiner 無し)



Shuffle Intensive



考察

- データ量が大きくなるとSSDの効果大
  - SSSだけでなくHadoopにも

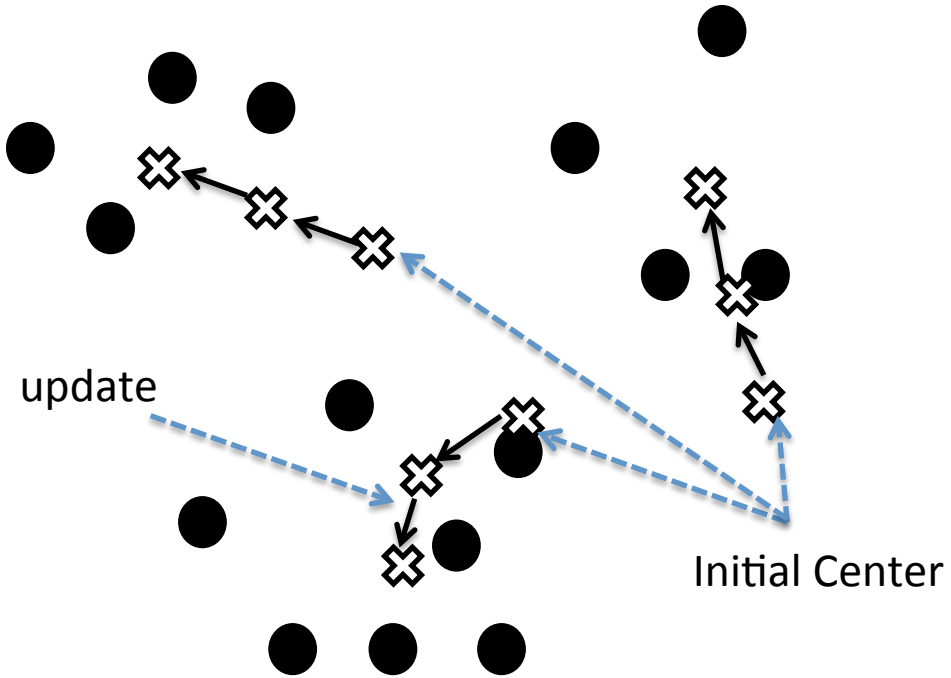
- Read はHadoop が優越
  - KVSの管理オーバヘッド
- WriteはSSSが若干優越
- Shuffle はSSSが優越
  - Combinerなしで特に

K-meansによるクラスタリング

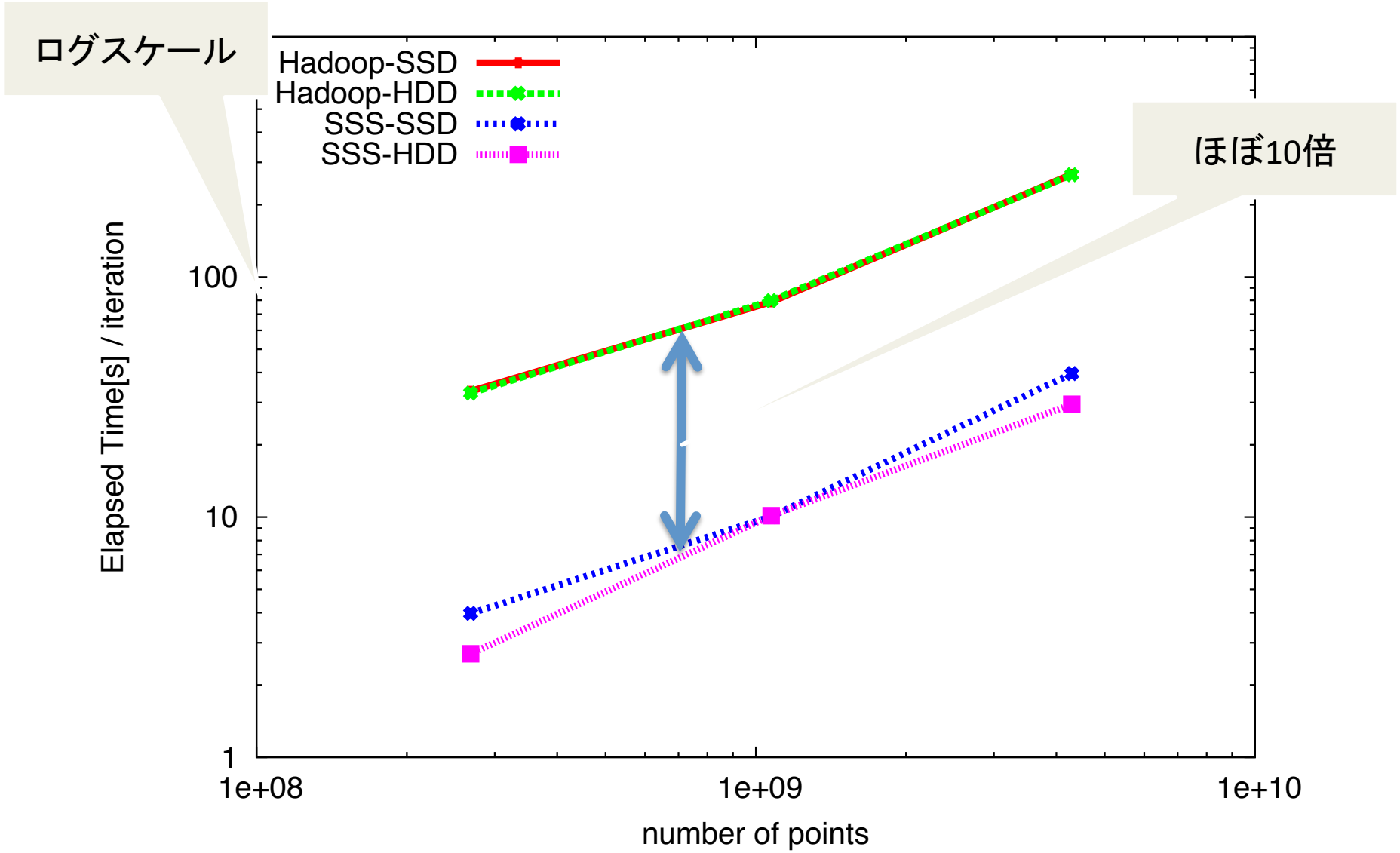
- 大容量データを繰り返しスキャン
- 重心を繰り返し更新
- 収束するまで実行

- Read Intensiveに近い

- 256Mi点、1Gi点、4Gi点を処理
- データ総量は 1GiB, 16GiB, 64GiB



K-meansの結果 (iteration あたり)



まとめ

- SSSの実装を詳説
- SSSの評価
  - 合成ベンチマークによるHadoopとの比較
    - データ量小では大幅に高速
    - データ量大でもReadを除いて高速
  - K-meansによる評価
    - 10倍の高速化 ― データ量小

今後の課題

- 他の実アプリケーションでの評価
  - PrefixSpanでの評価 [CPSY 中田]
- 実行モデルの拡張
  - 追加されたデータを随時処理するストリーミングコンピューティング的な処理など
- オープンソースでの公開
  - 2011年度末を目処
  - http://sss.apgrid.org/

謝辞

- 本研究の一部は，独立行政法人新エネルギー・産業技術総合開発機構（NEDO）の委託業務「グリーンネットワーク・システム技術研究開発プロジェクト（グリーンITプロジェクト）」の成果を活用している.

ありがとうございました