Hadoop じゃなくて Dryad 考察

日本マイクロソフト株式会社
萩原 正義
@masayh

目的と Agenda

- Data Intensive Scalable Computing (DISC) における Dryad を理解する

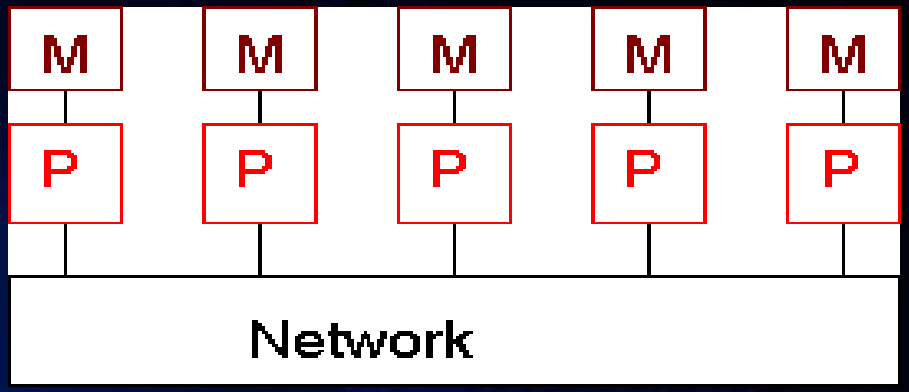
- DISC のアーキテクチャースタイル
- MapReduce の抽象化と最適化
- Dryad, DryadLINQ 概要
- 今後の方向性

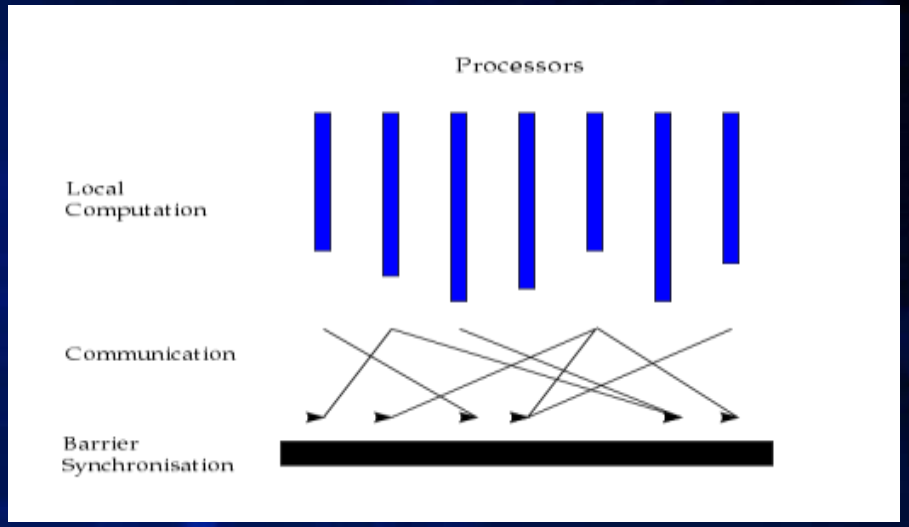
アーキテクチャスタイル

- Communication-oriented models
  - MPI, Parallel Virtual Machine, BSP
- Distributed shared-memory (DSM)
  - Tuple Spaces
- MapReduce and Dataflow models
  - Hadoop, Dryad, Spark
- Single-machine shared memory models
  - GPU: CUDA, OpenCL
  - Multicore CPUs: Cilk, OpenMP
- Distributed data structures
  - Memcached, RamCloud, KVS on DHT

BSP (Bulk Synchronous Parallel)

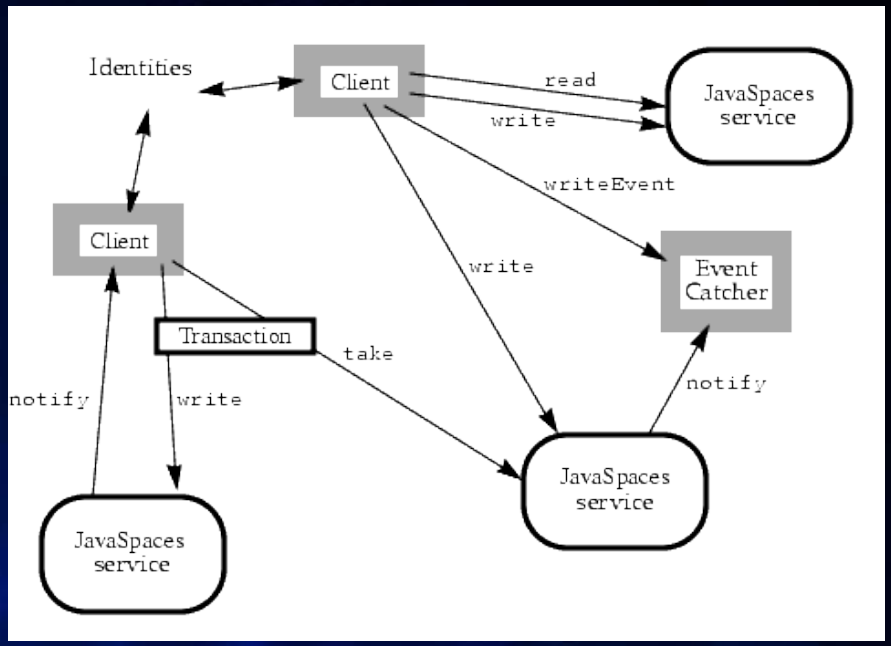
- プロセッサーとメモリの対がネットワーク通信で連結したシステム
  - ポイントツーポイント
  - バリアー同期を持つ
- プロセッサーはローカルデータでの実行、リモートメモリへの要求
- 各プロセッサーは実行終了時にバリアー機構により同期
- プロセッサー数、同期の遅延、通信スループットによるモデルの複雑さを持つ



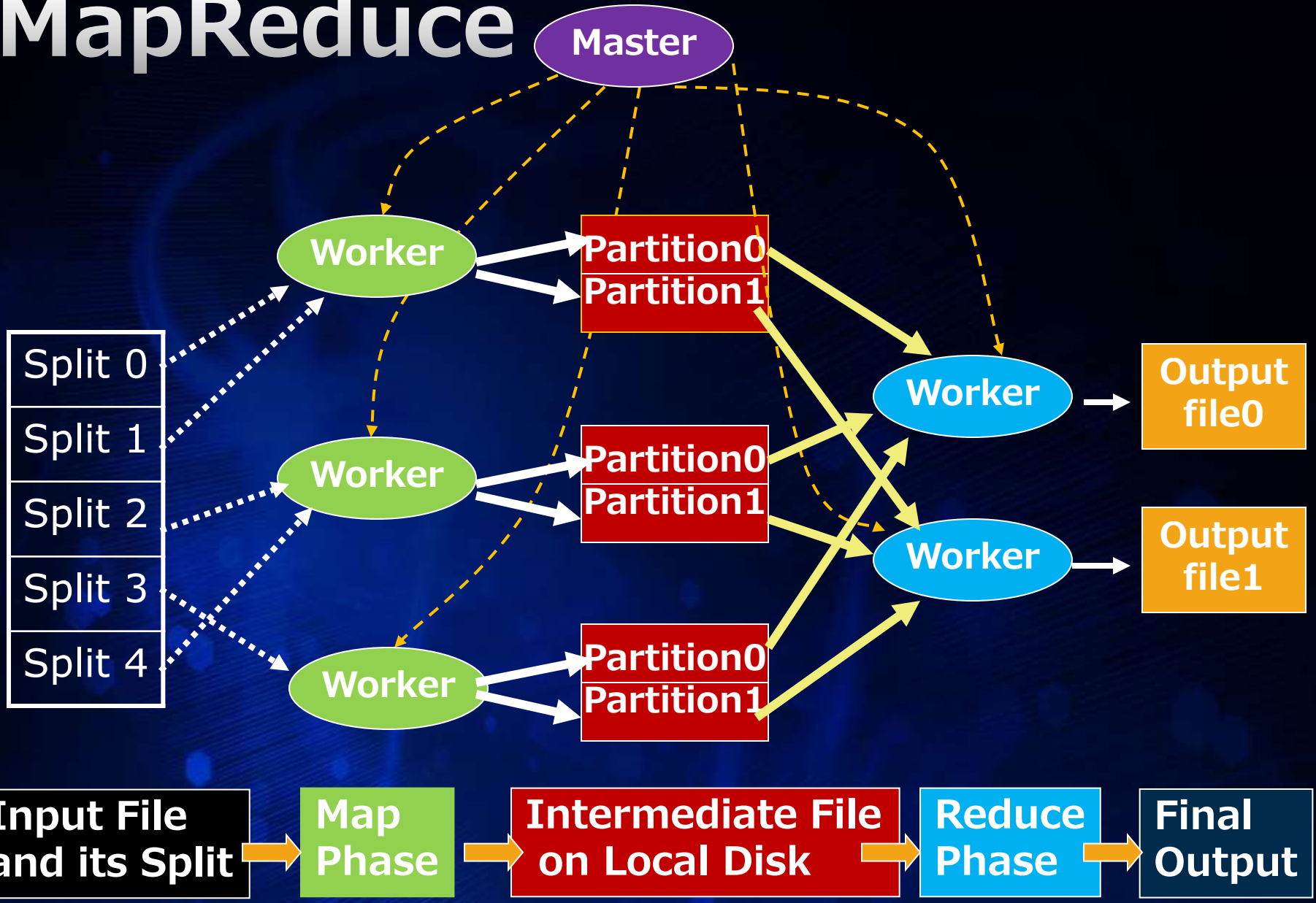


Tuple Spaces

- ネットワークサービス提供者と利用者が共有する仮想的リポジトリー Object Spaces
  - Object の属性を登録し識別する
  - 利用者は Object をローカルメモリの取り出し、そのサービスを利用し、状態を変更し、Object Space に戻す
  - 相互排他制御を保証
- 分散オブジェクト交換、協調機構による分散アルゴリズム、分散状態管理を実現
  - 並列処理によるスケーラビリティーと Object の分散冗長化による可用性
- Master-Worker パターン
  - Master が Space に UoW を挿入し、Worker が処理し結果を Space に戻す
- Linda, JavaSpaces, GigaSpaces など



MapReduce



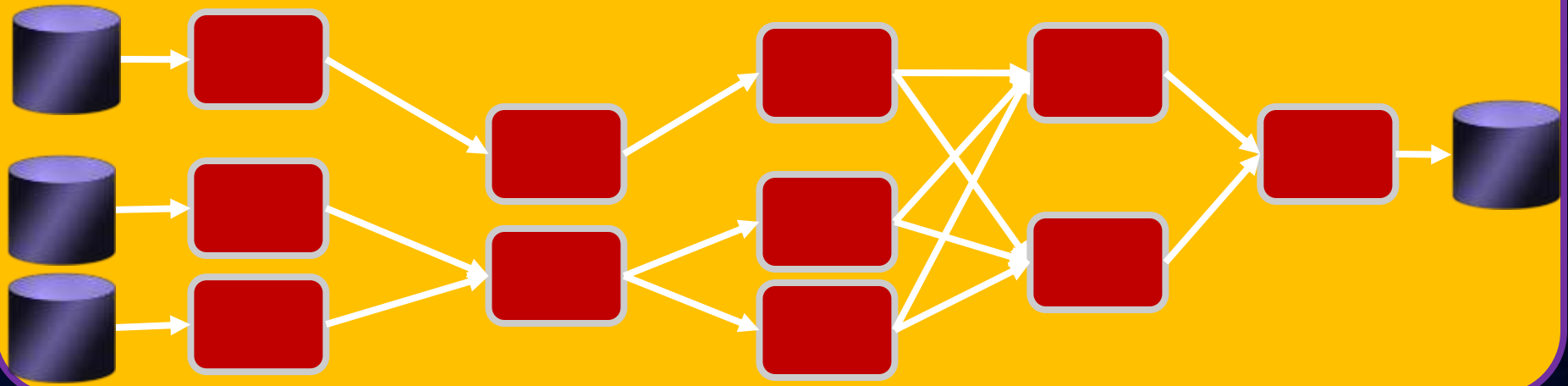
2-D Piping

- **Unix Pipes: 1-D**

grep | sed | sort | awk | perl

- **MapReduce: 2-D**

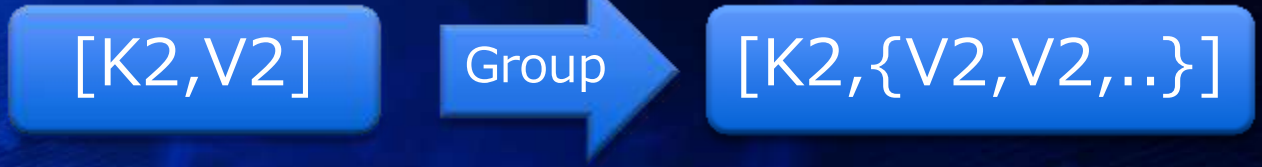
grep^{1000} | sed^{500} | sort^{1000} | awk^{500} | perl^{50}



Map-Group-Reduce

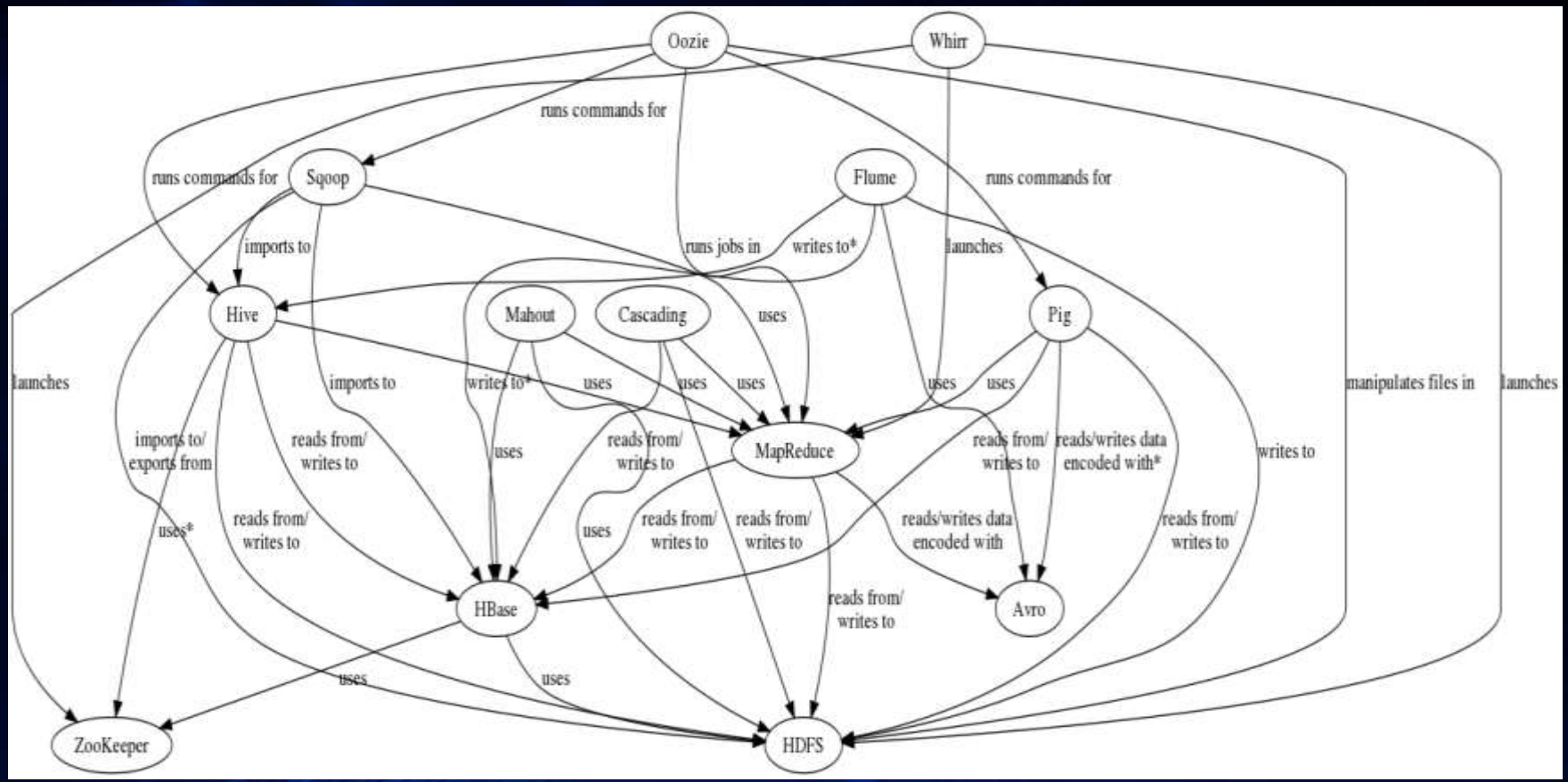
- Map と Reduce を定義する
- Map は入力 Key と Value を新しい Key と Value に変換

- システムは Key をソートして各 Key 毎に Value をグループ化



- Reduce は Key 毎にグループ化された Value を新しい Key と Value に変換

Hadoop エコシステム



抽象化

- データ型
  - 外部 DB、ファイル、インメモリ、ストリームデータを扱う
  - OOP ならコレクションで抽象化
    - Immutable bag で型 T を持つ
- 実行環境
  - ローカル、逐次でのテストからクラスター、並列化へ
  - Local loop、MapReduce での並列処理化
    - Map, Aggregation, Group： コレクションの項目単位
    - Shuffle, Reduce： コレクションの項目間
    - この2つに分割して記述するのが MapReduce
  - 全体をパイプラインで手順記述
    - 各手順を非同期化可能、しかも逐次的
    - 複雑化すると複数段、グラフ構造となる DAG による抽象化が必要
    - 同時に順序制御、エラー処理、中間ファイル管理も必要のため

抽象化の例

- 代表的アプローチ
  - 宣言的 （SQL-like）
  - 関数的 パイプライン/プロセス記述
- 事例
  - SCOPE, DryadLINQ
  - Pig Latin（Yahoo）
  - Hive
  - Cascading
  - Jaql（IBM）
  - Sawzall, FlumeJava（Google）
  - SQL/MapReduce（Aster Data）
  - Map-Reduce-Merge（SIGMOD 07）
  - Ciel
  - Cascalog
  - BOOM

SQL（関係代数）による基本操作

- 基本操作
  - Selection σ： （複数）行の取り出し
  - Projection π： 不要な列の削除
  - Cross-product X： 2つの関係の結合
  - Set-difference − ： 1つの関係に存在し、もう１つの関係に存在しない tuple
  - Union ∪： 2つの関係に存在する tuple
- 追加操作
  - Intersection, Join, Division, Renaming
- 操作（代数）は閉じている

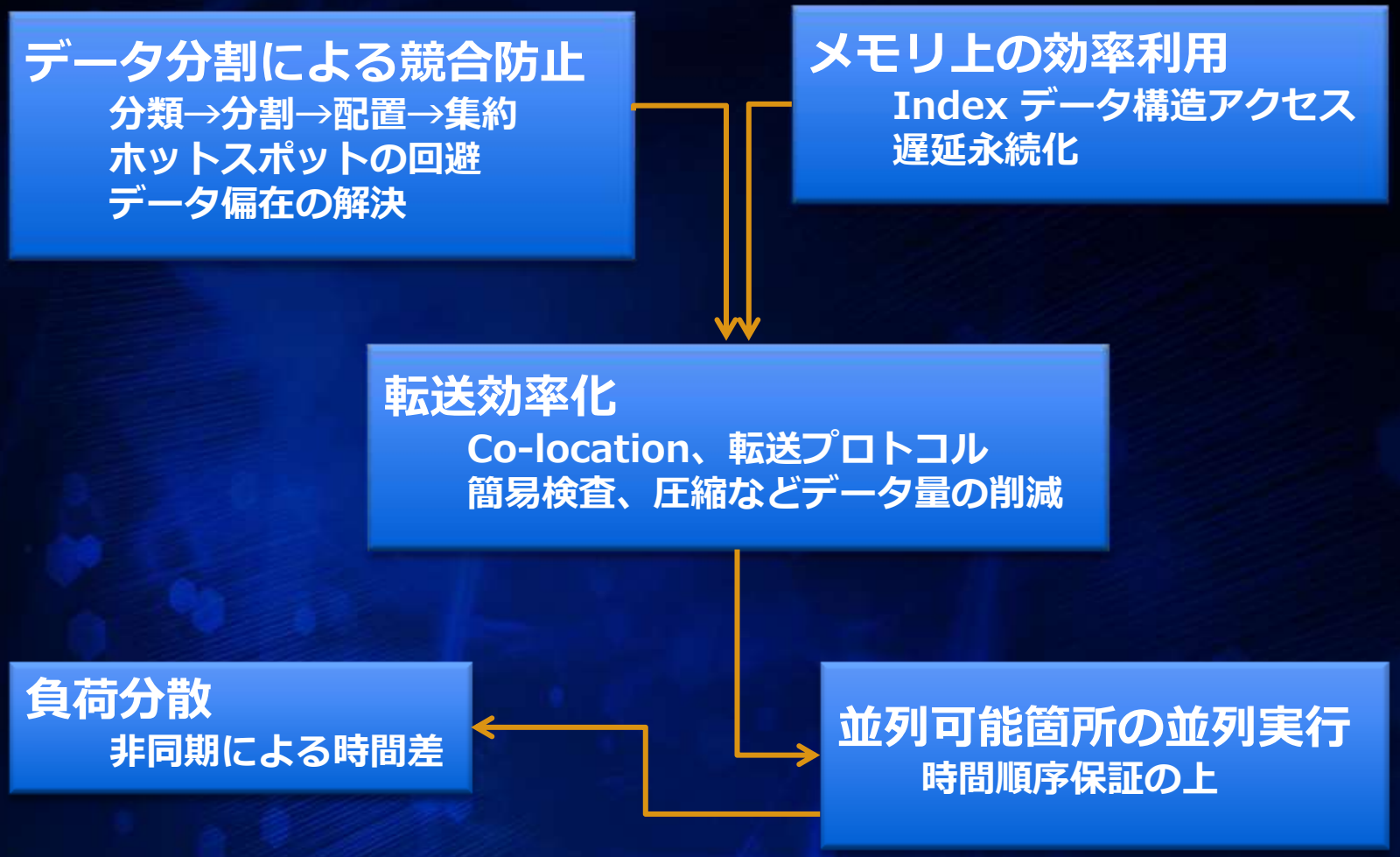
最適化

- DISC は事前最適化が困難
  - データ形式の決定
  - UDF のコスト、データ量の見積もり
  - 異機種、低信頼性のクラスターの性能評価
- 論理最適化
  - データ操作、並列化のためのタスク定義
  - Projection、filtering の早期化、operator rewrite
- 物理最適化
  - MapReduce 固有
  - 単一ジョブ内最適化、複数ジョブ間の最適化
- 動的最適化

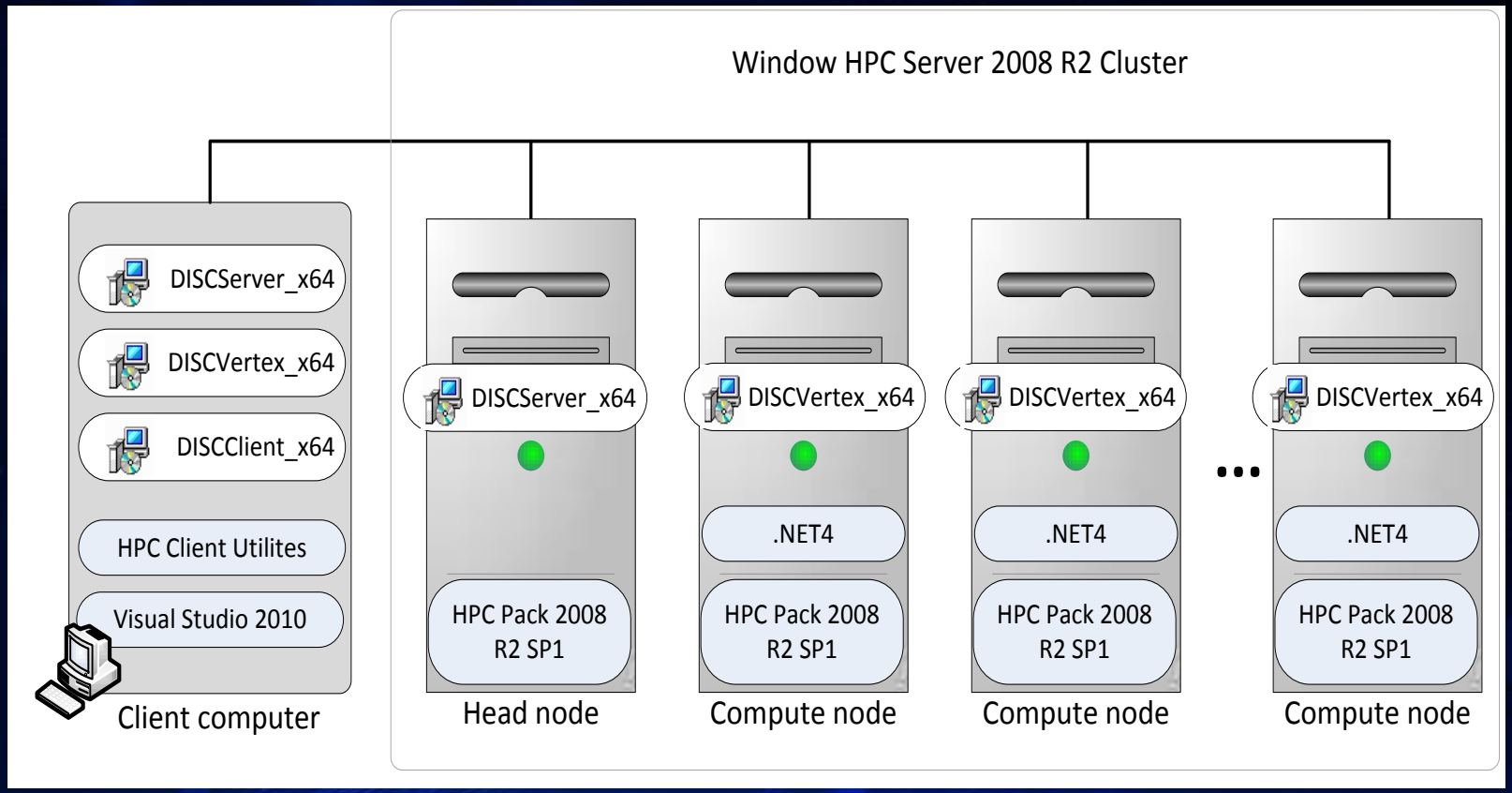
MapReduce の物理最適化

- ジョブ段数を減らし shuffle 数を削減
- 前段に寄せることでデータ転送量を減らす
  - push-down： join－select－projection を select－join－projection とする
    - sum(2,3,1) = sum(sum(2,3),1)
    - avg(2,3,1) != avg(avg(2,3),1)
    - インクリメンタルな計算に置き換え
  - semi-join(⋉) , bloom filter で転送量を減らす
  - Combiner
- データ偏在の解決、分割法
- データ構造の最適化（カラム指向 DB など）
- コード/データの共有、キャッシュ
- 動的最適化

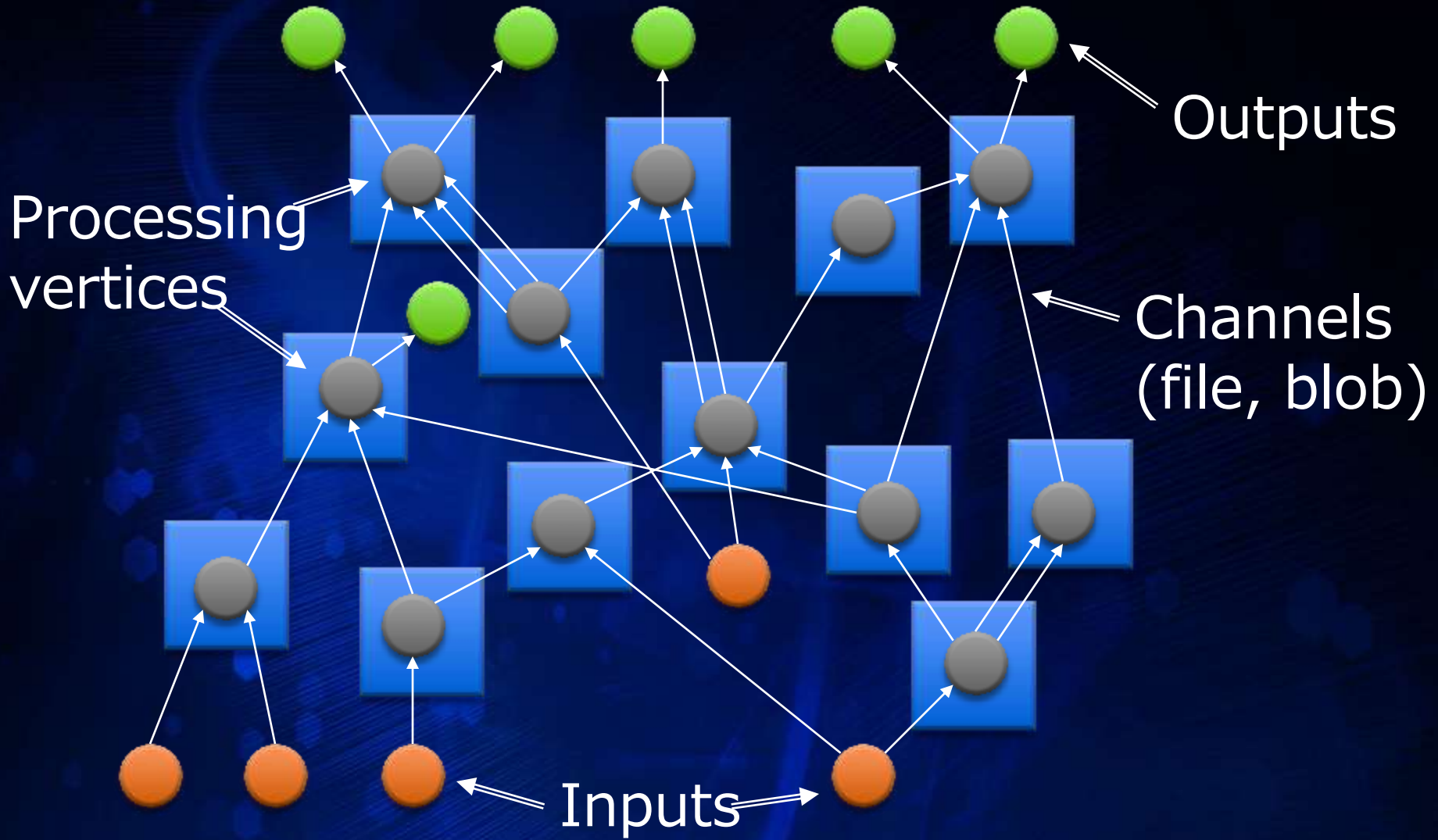
アーキテクチャー設計の原則



Windows HPC Server 2008 R2 Cluster



DAG (Directed Acyclic Graph)



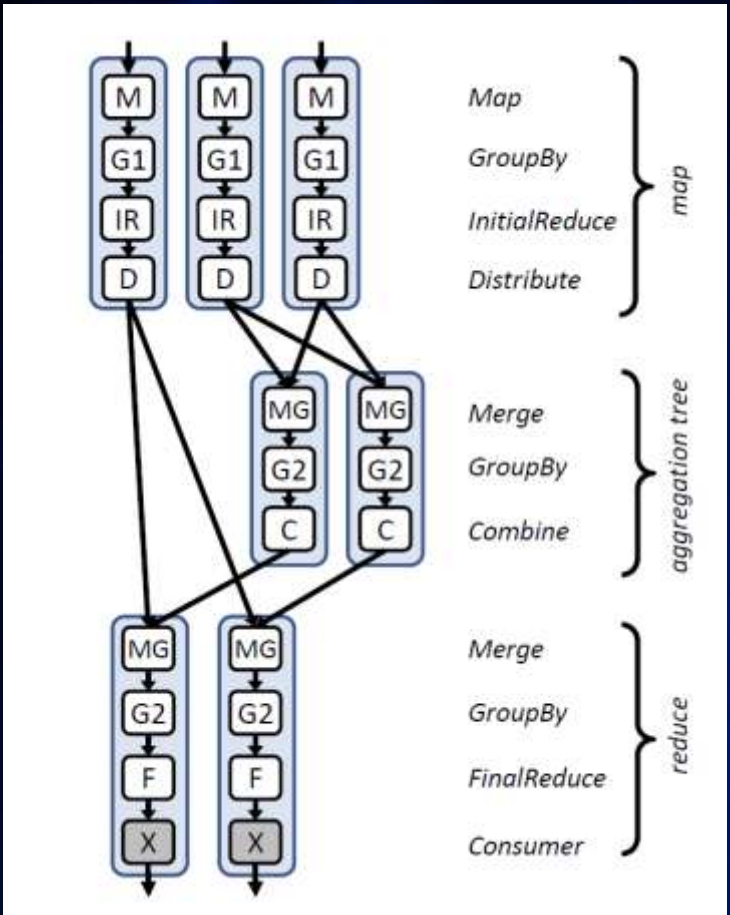
DryadLINQ 例

```
var duplicatedFiles =
        Directory.GetFiles(directoryName,"*.jpg",
            SearchOption.AllDirectories)
        .Select(filename => new {
            hash = GetChecksum(filename),
            name = filename })
        .GroupBy(record => record.hash)
        .Where(group => group.Count() > 1)
        .SelectMany(group =>
            group.Select(record => record.name));
```

Dryad の特長

- パイプラインを指定しない、DAG の抽象化
- クライアントサイドの最適化
- コード配置、Co-location
- ラムダ式、データ型の抽象化、LINQ プロバイダー拡張
- HPC との統合
  - 他の DISC アーキテクチャスタイル
  - Incremental updates、iterative/recursive applications
- グラフ言語

分散実行プランの例



Partial Aggregation
を実行している例

今後の方向性

- Dremel、Piccolo、Ciel など他アーキテクチャスタイルとの融合
  - Communication-oriented、複数との組み合わせ
- ジョブ内、ジョブ間最適化
  - 計算、I/Oの再利用
  - 業務系（帳票バリエーション）、データ分析（パラメータ組み合わせ）
- Asakusa など問題領域対応
  - DSL、コンパイラ最適化と MapReduce 最適化、中間コード、トランザクション、監査
  - Push と Pull （Co-Relational Model）
- Nectar, Data cube など代数的手法
  - Incremental query plan, decomposable functions