再配置可能な大域アドレス空間システムの設計とRDMAを用いた実装

東京大学 大学院 情報理工学系研究科

遠藤 亘，田浦 健次朗

2015/08/04

SWoPP 2015

発表の概要

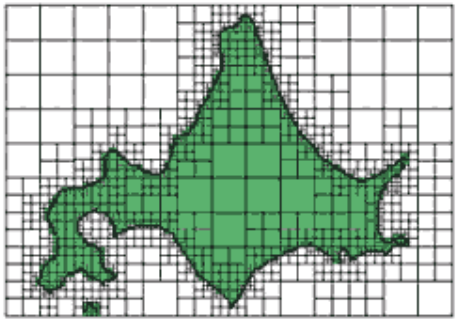
- データ再配置できるPGAS処理系の高速な設計
  - **データの再配置**
    - 共有データが配置されているノードを変更
    - ノードをまたいだ**動的負荷分散後の通信を削減**
  - **メタデータキャッシュ**
    - RDMA転送に必要なアドレス情報のキャッシュ
    - **最短でRDMA1回**でリモートの共有データにアクセス
  - 再配置とメタデータキャッシュを両立できるプロトコル
- 実際に処理系を実装し，評価を行った
  - メタデータキャッシュの導入によって
    get/put関数にかかるレイテンシが10倍以上高速化した

発表の流れ

- **研究背景**
  - 大域アドレス空間, PGAS
  - MassiveThreads/GAS
- 提案手法
- 現実装の評価
- 関連研究
- 今後の課題
- まとめ

研究背景

- 分散メモリ環境でのプログラミング
  - MPIのモデル (SPMD+メッセージパッシング)は生産性に課題
  - **大域アドレス空間**の活用による生産性向上
- 動的で非定型な計算構造を持ったアルゴリズム
  - 動的負荷分散の必要性
  - HPC分野で採用が進んでいる
  - 例. Adaptive Mesh Refinement (AMR)



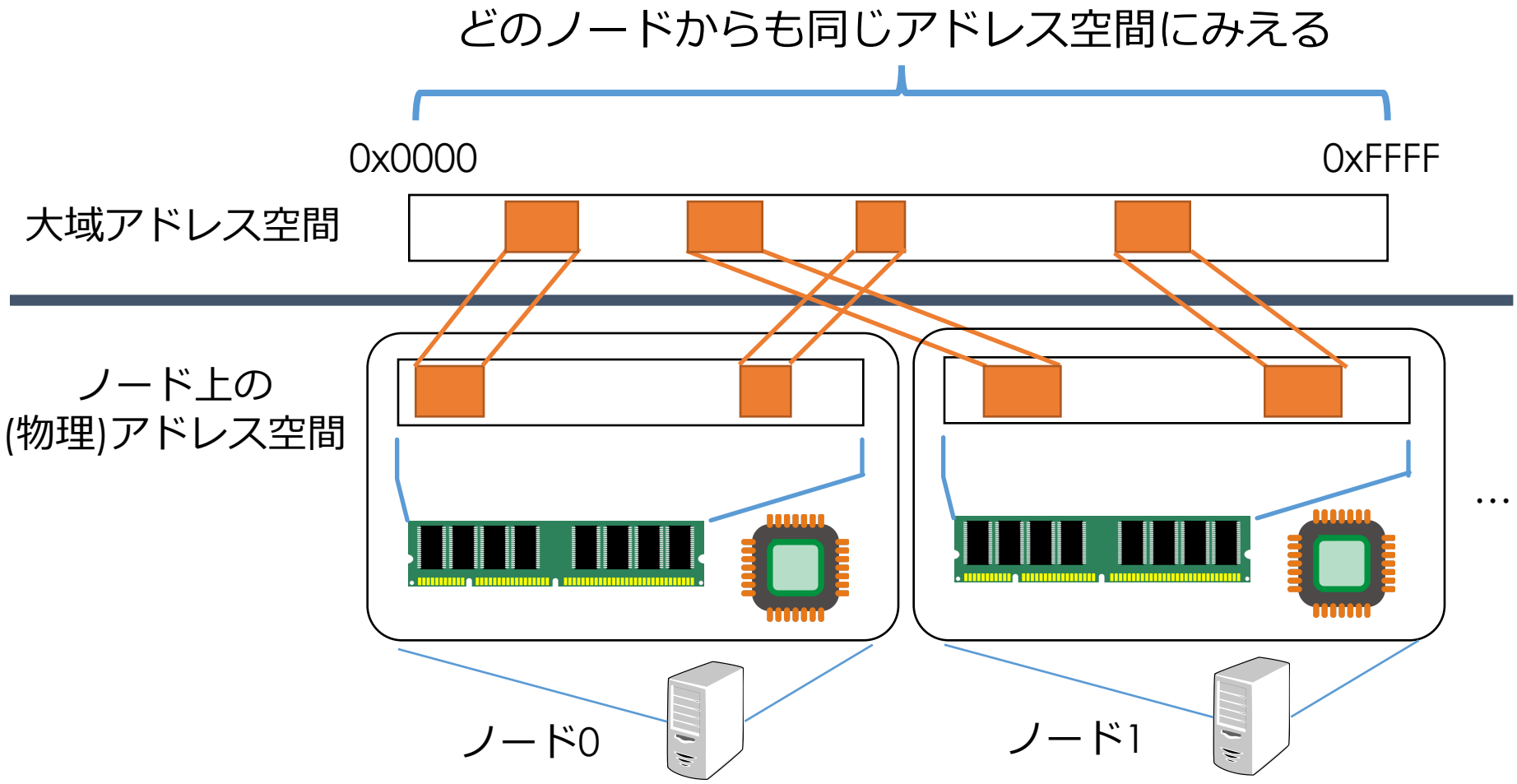
形状に合わせて
適応的に格子を生成し，
シミュレーションを高速化

東京工業大学 青木研究室のWebページより引用

http://www.sim.gsic.titech.ac.jp/Japanese/Research/amr.html

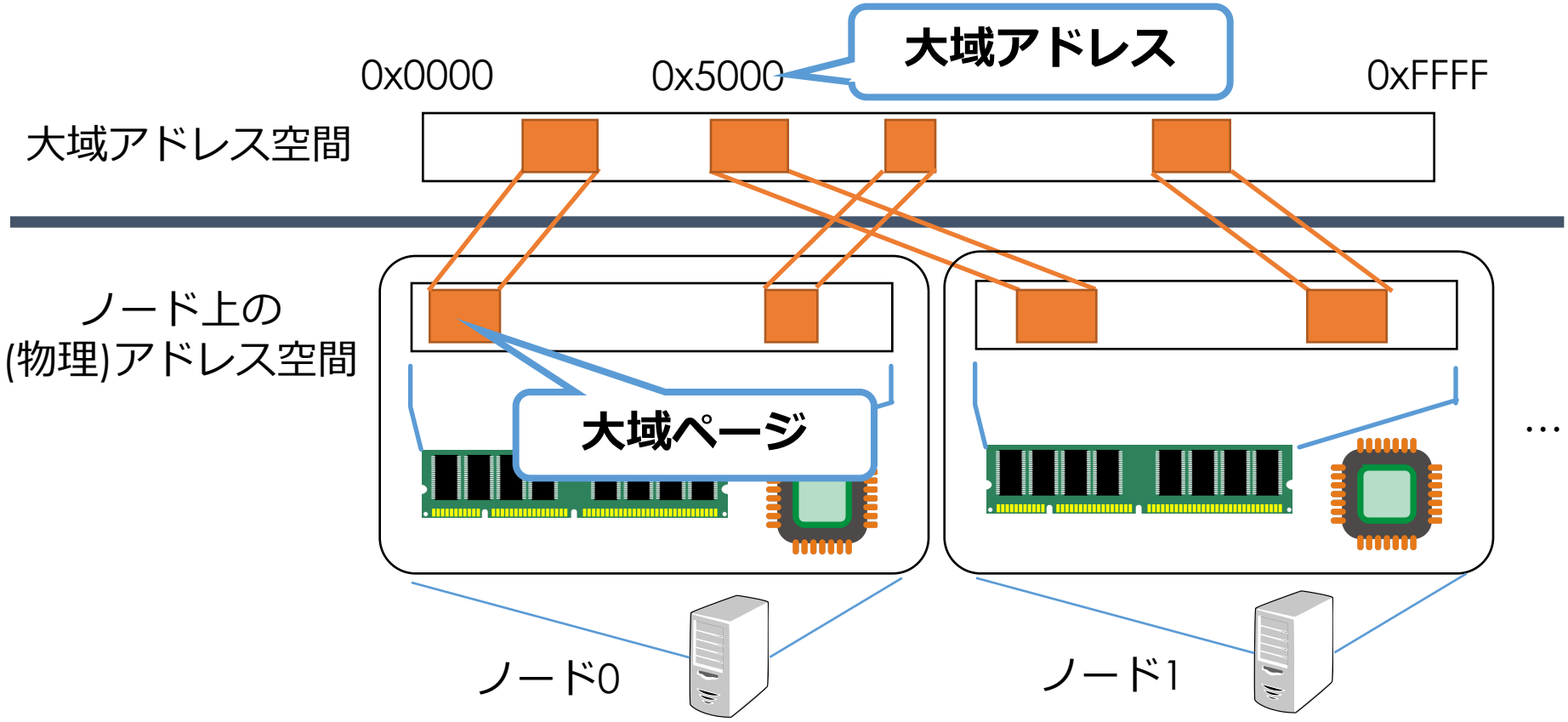
大域アドレス空間

- **仮想的に共有**されたアドレス空間
  - 分散メモリ環境で共有メモリに近い抽象化を提供



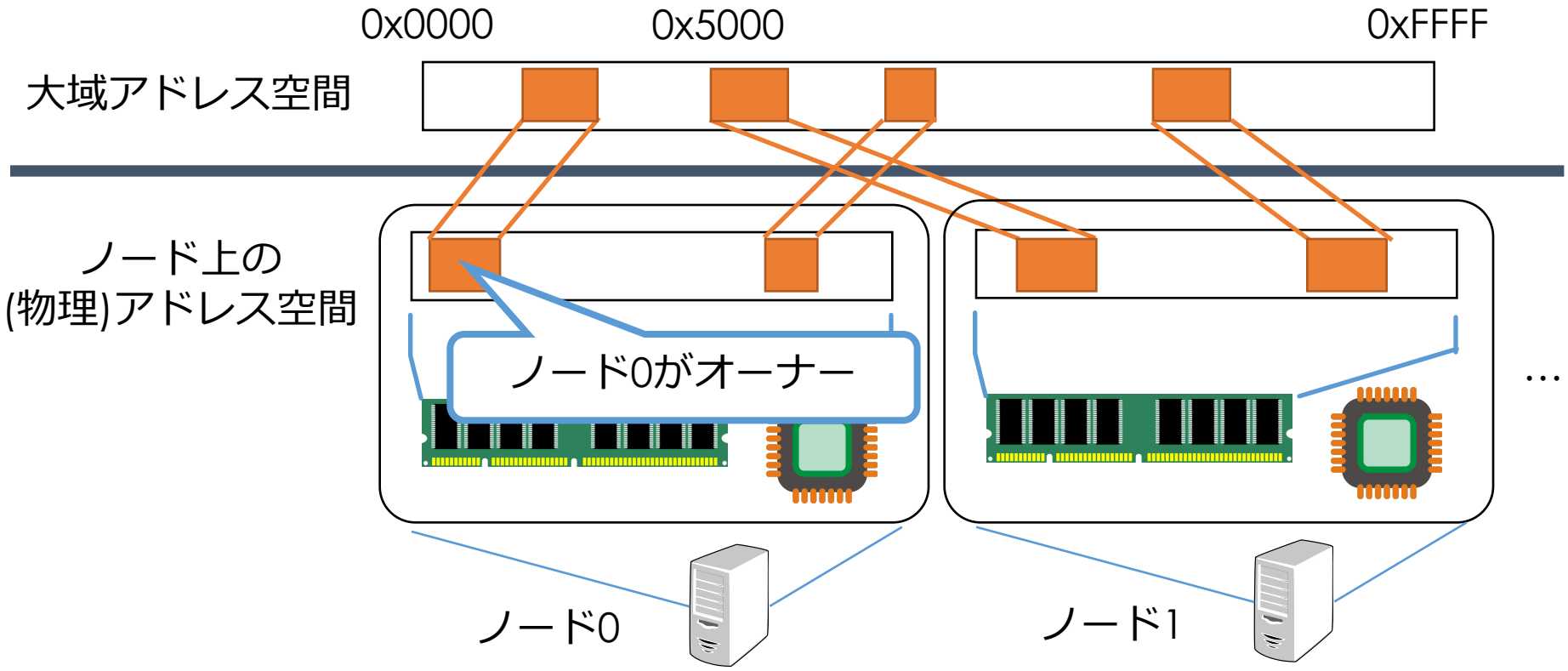
大域アドレス空間

- 共有データは”**大域ページ**“のような小さな単位に分割
  - “**大域ページID**”や”**大域アドレス**“といった値により，
    どのノードからも一意にデータを特定できる



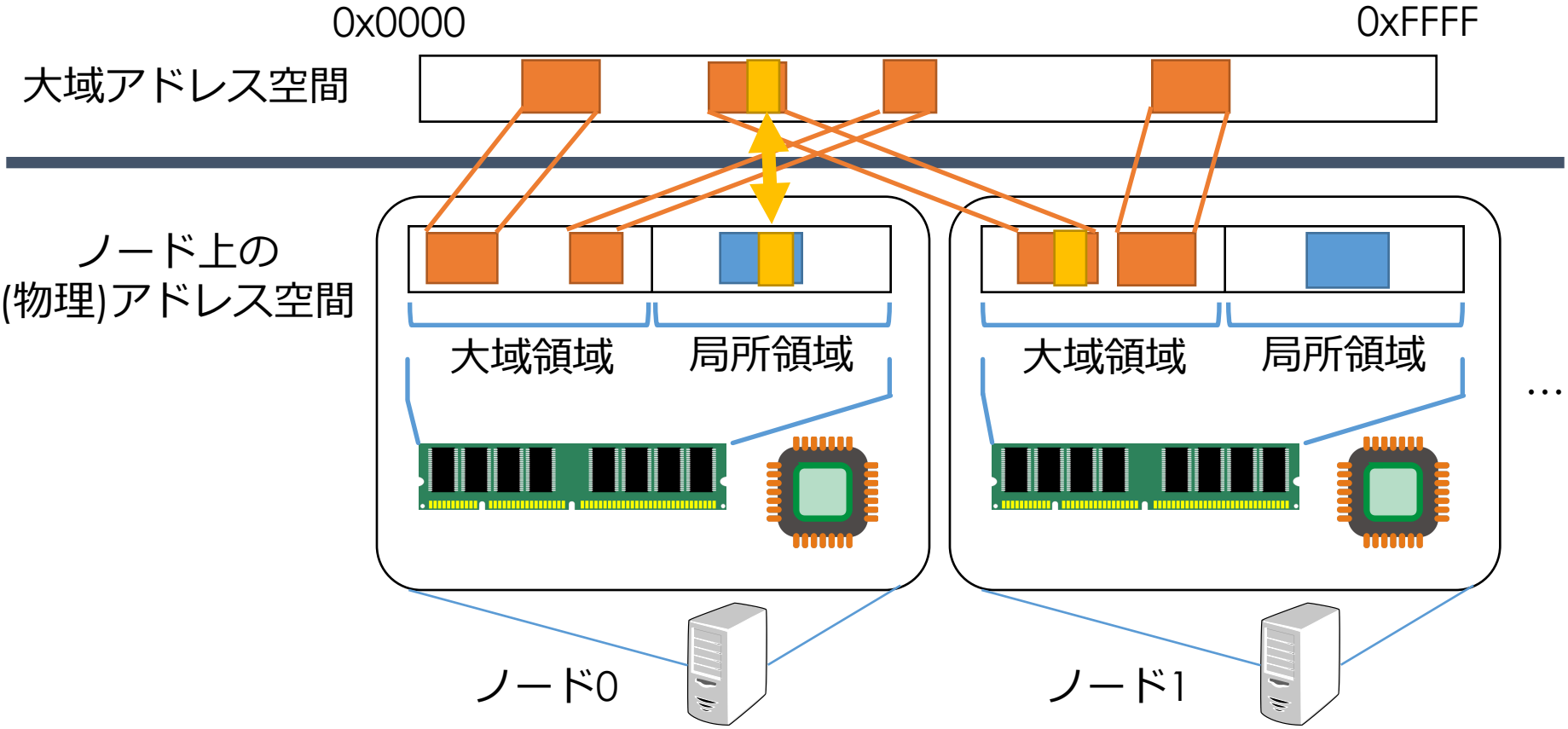
大域アドレス空間

- 各大域ページはいずれかのノード上に**“配置”**
  - 配置されているノードを**”オーナー“**と呼ぶ
  - 大域アドレスと実際のデータの対応関係を保ちながら
    オーナーを変更することを**”再配置”**と呼ぶ



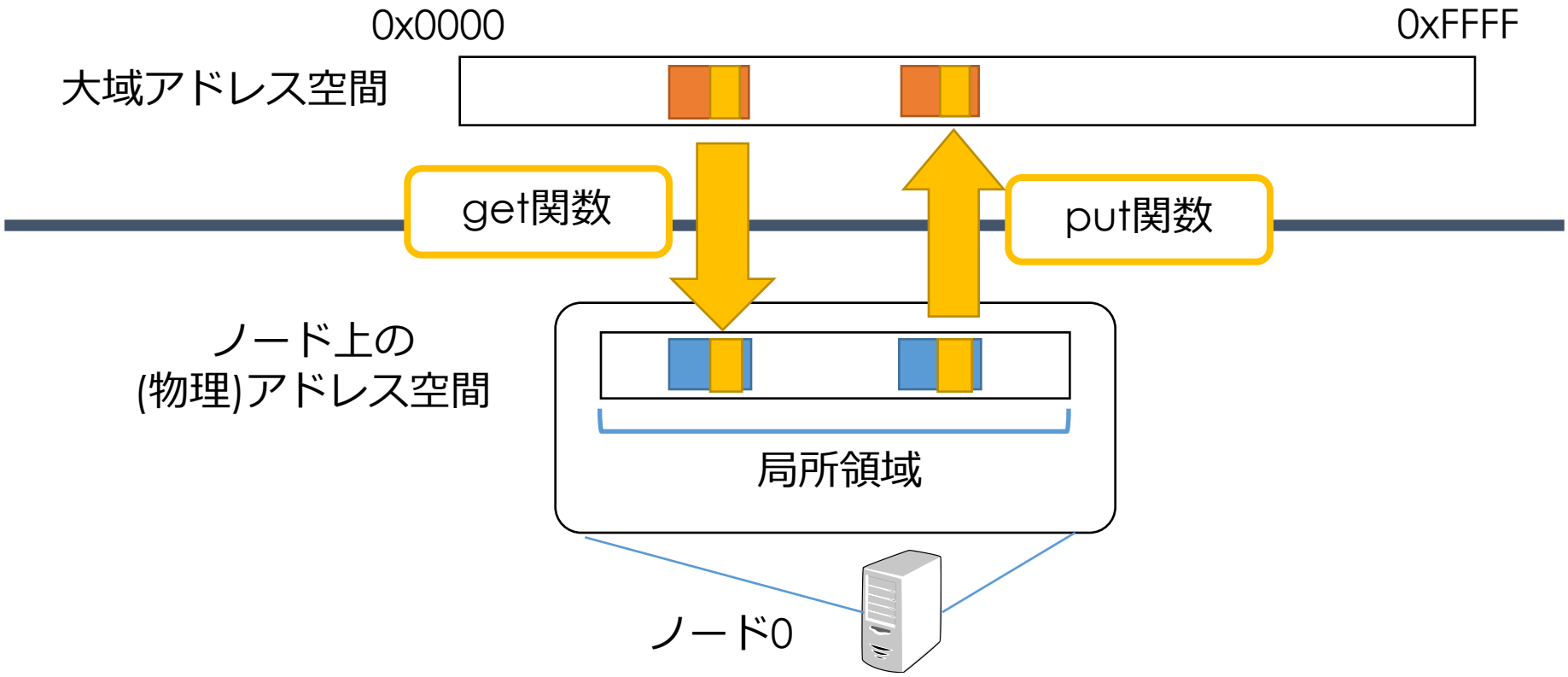
Partitioned Global Address Space (PGAS)

- **大域領域と局所領域を区別**する
  - 大域領域上のデータのみが共有される
  - 大域領域と局所領域間のデータコピーは，**get/put関数**によってのみ行える



Partitioned Global Address Space (PGAS)

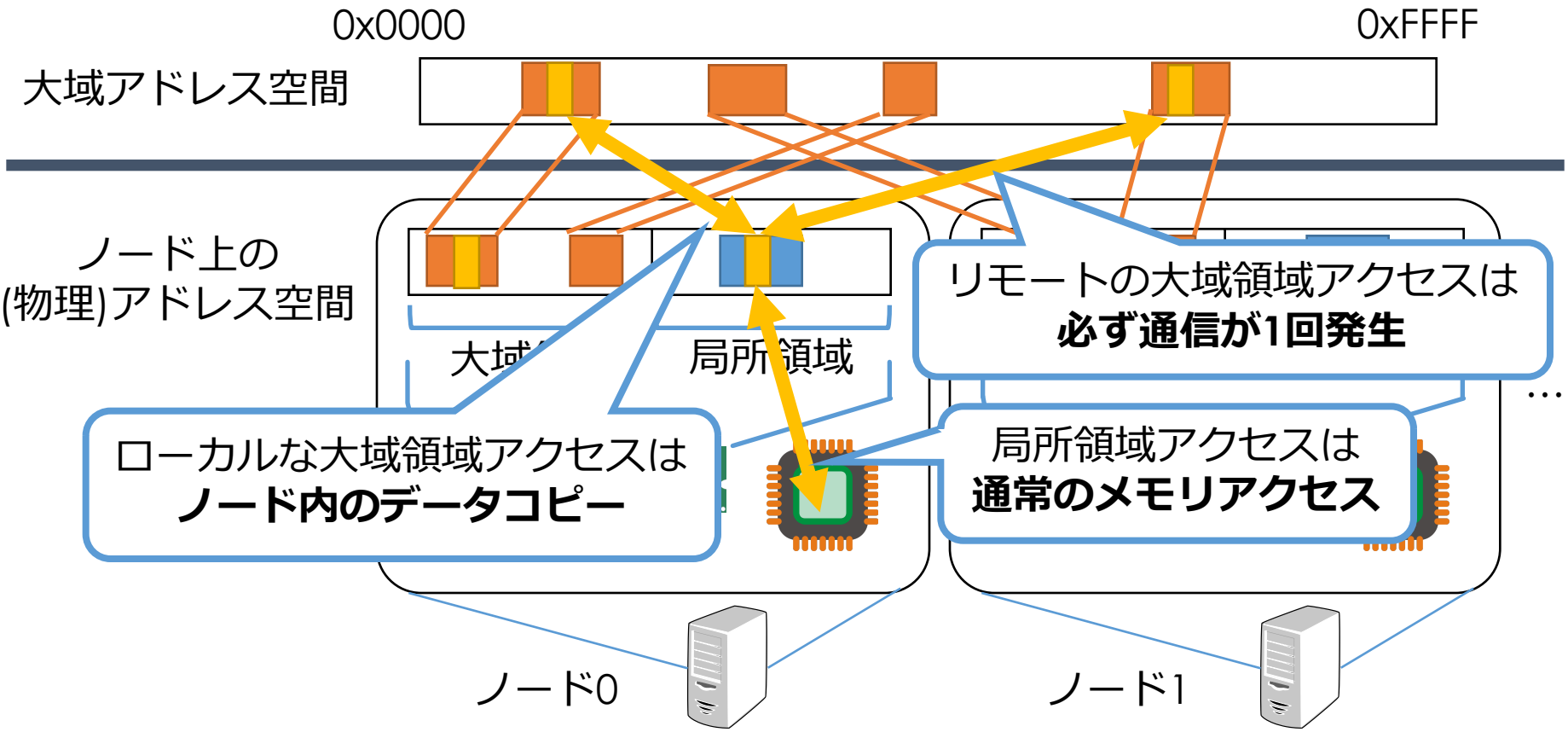
- **大域領域と局所領域を区別**する
  - 大域領域上のデータのみが共有される
  - 大域領域と局所領域間のデータコピーは，**get/put関数**によってのみ行える



PGASの特長

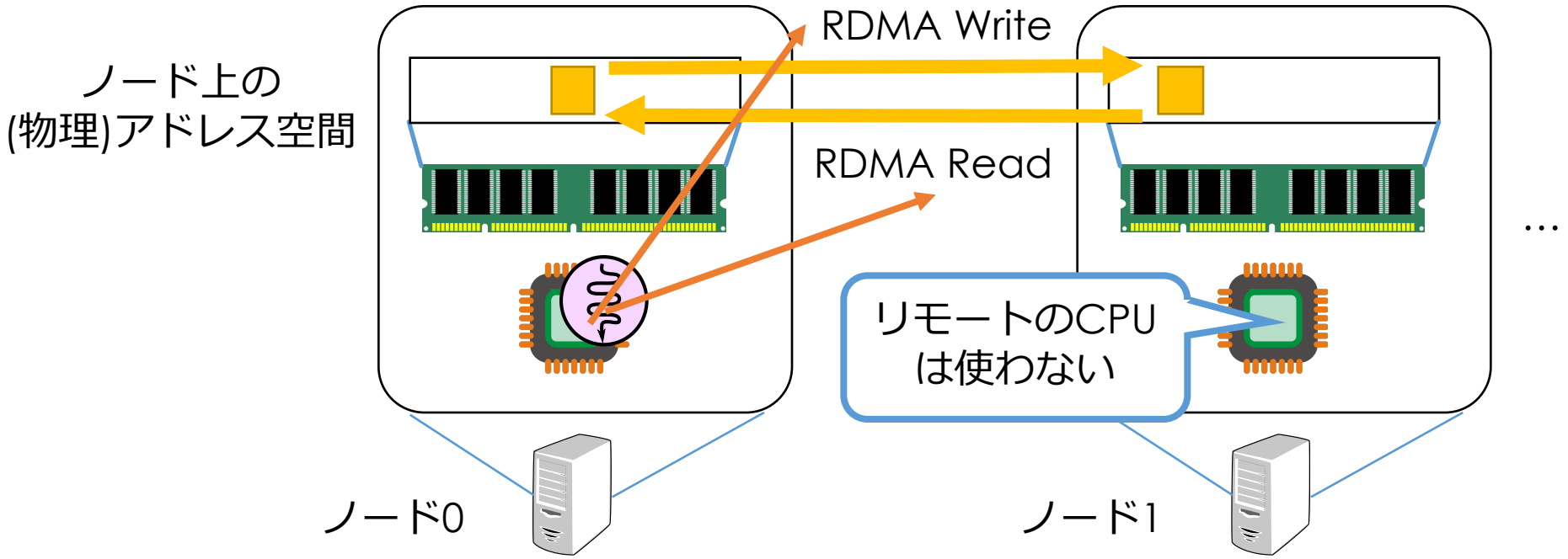
- **RDMA**との相性が良い (後述)
- メモリアクセスコストが明確
  - cf. Distributed Shared Memory (DSM)

アクセスコストが不明確なモデルで，スケーラビリティに難がある



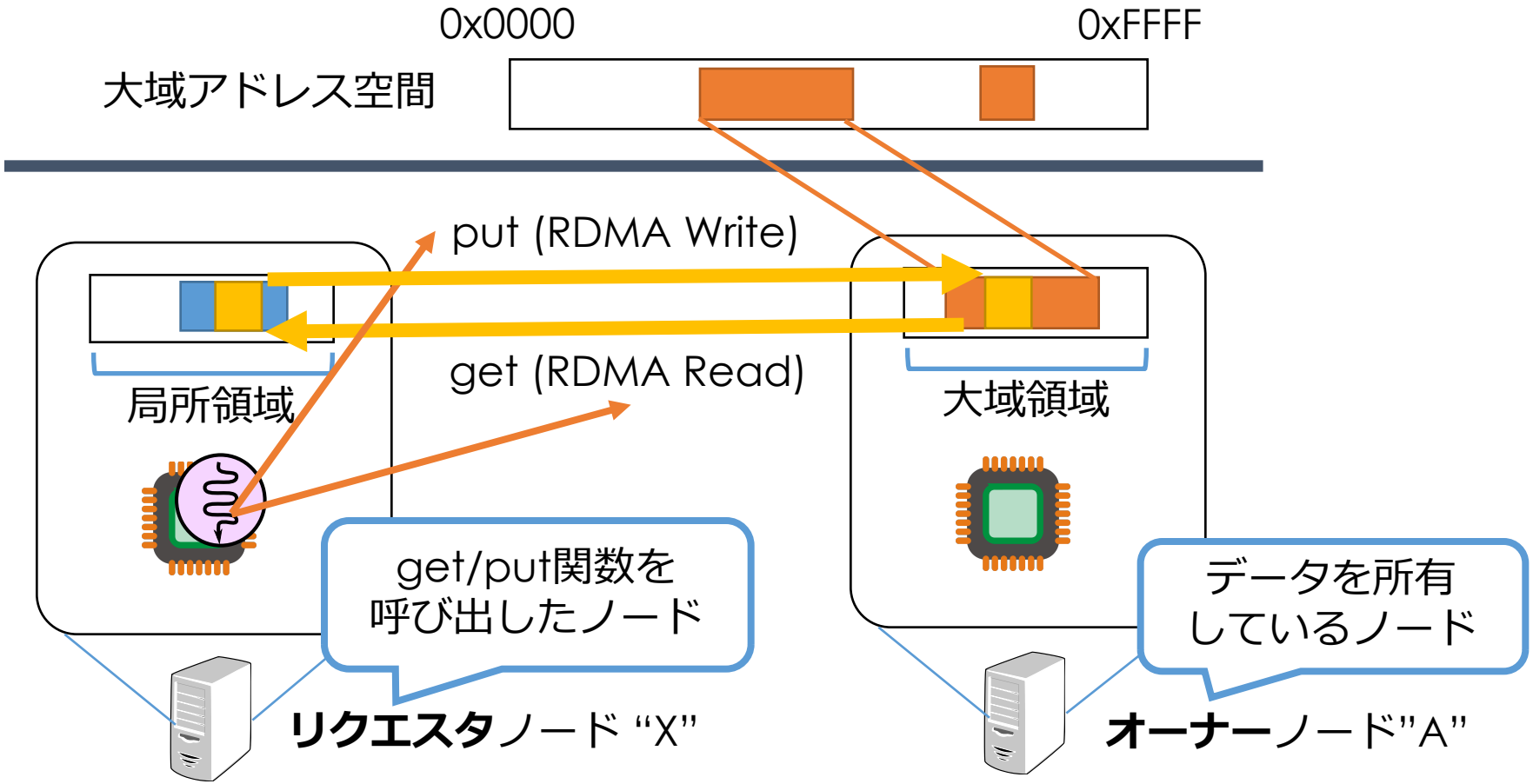
Remote Direct Memory Access (RDMA)

- ノード間のメモリ転送を行えるハードウェア機能
  - リモートノード上のCPUの介在なし，カーネルバイパス，ゼロコピーなどの利点
  - **低レイテンシ，高スループット**の通信



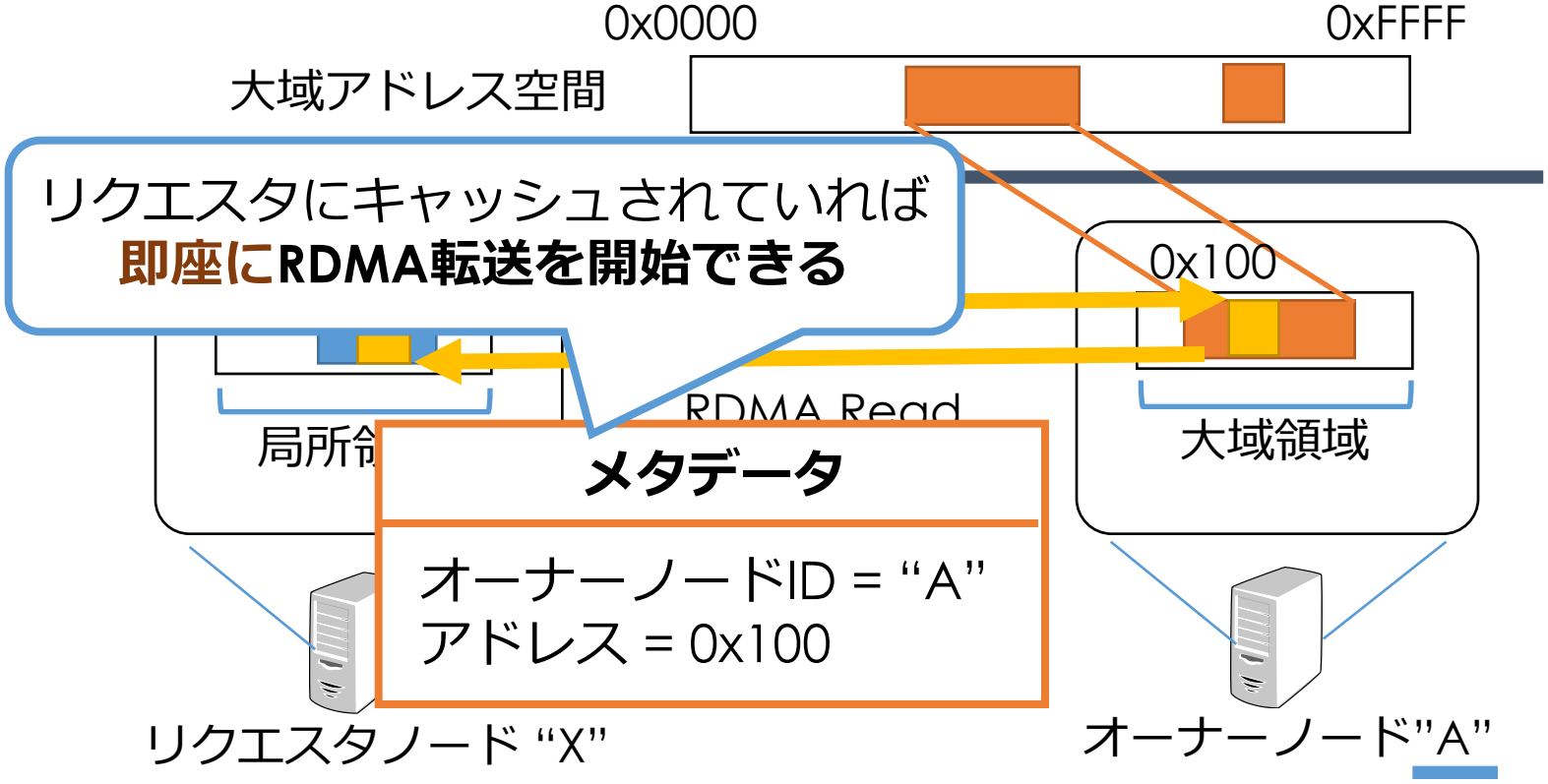
PGASとRDMA

- PGASのget/put関数は**容易にRDMA化**できる
  - RDMA READ/WRITEにそのまま対応
  - リモートノード上のCPUの介在は元々必要ない



PGASとメタデータキャッシュ

- PGAS上のデータは**静的**に配置
  - RDMAに必要なリモートノード上の大域領域に関する情報(**メタデータ**)も静的に決定 → キャッシュしやすい



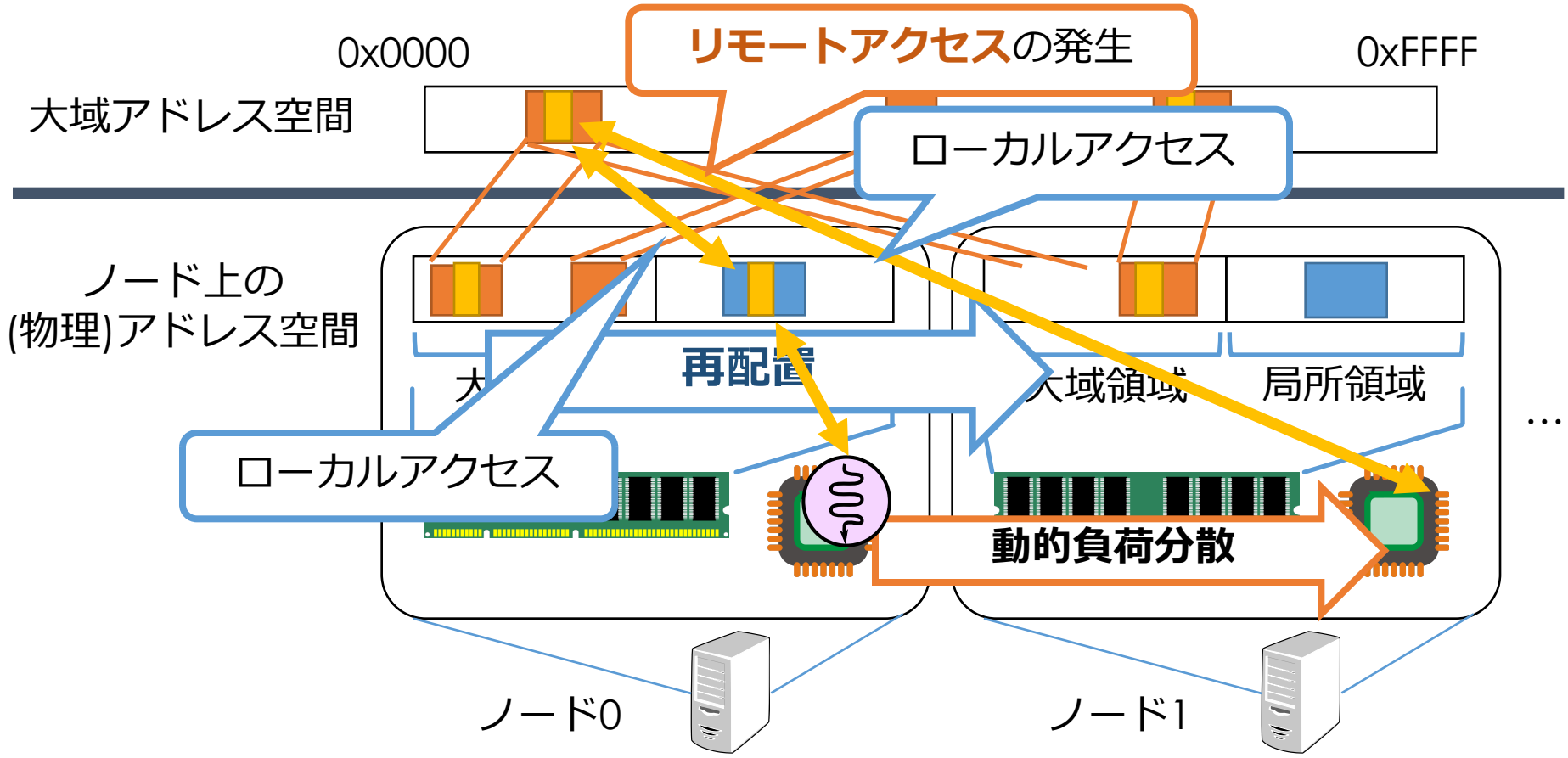
PGASの問題点

- データ配置が**柔軟でない**
  - 処理系が用意したパターンの配置のみ可能
    - 例えば，大域ページIDに対してノード番号をサイクリックに割り振るといったパターン
  - **非定型なデータ構造**に合わない
- データ配置を**動的に変更できない**
  - **動的負荷分散**後に通信が多発するおそれ

データ配置を手動で柔軟に調整できる機能が必要

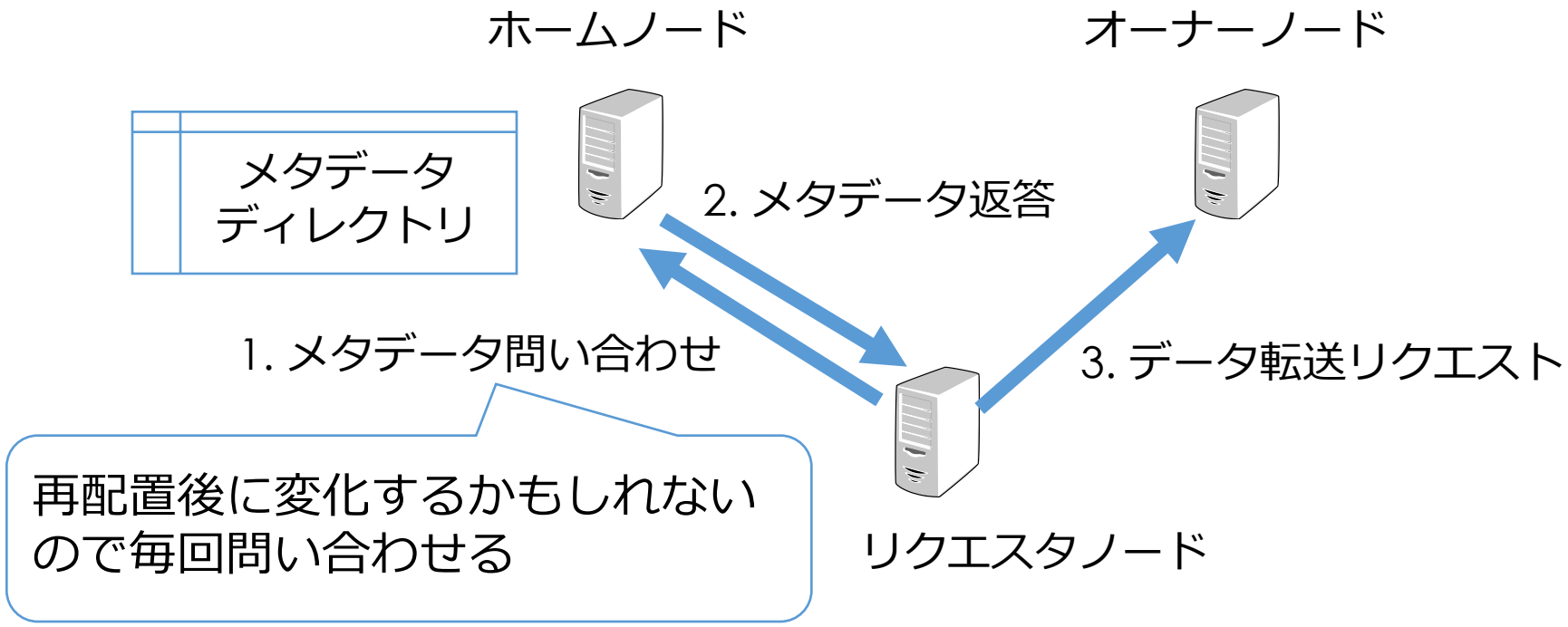
MassiveThreads/GAS

- 当研究室で開発した処理系 [秋山, 2012]
  - PGAS同様にget/put関数を提供
  - 必要な時に**手動再配置**を行える**own関数**を提供



MassiveThreads/GASの課題

- 本研究以前の実装
  - メタデータを集中管理するノード (ホームノード)を用意
  - get/put関数呼び出し毎にメタデータ問い合わせ
    →**get/put関数のレイテンシ増大**の原因



MassiveThreads/GASの課題

- get/put関数とown関数の呼び出し頻度
  - **own関数は負荷分散時以外には呼び出されない**
  - get/put関数はデータアクセス毎に頻繁に呼び出される
    →get/put関数を高速化することが重要
- **メタデータキャッシュ**を導入することが望ましい
  - get/put関数呼び出し時の，ホームへの問い合わせを減らす
  - メタデータがキャッシュされていれば，
    PGAS同様に**即座にRDMAを発行できるはず**
- **メタデータキャッシュは再配置との両立が難しい**
  - 再配置によって**メタデータが変化**する
  - get/put実行中に再配置が起こる可能性もあるので，
    適切に**排他制御**を行う必要がある

本発表ではこれを解決するための処理系の設計を提案

発表の流れ

- 研究背景
- **提案手法**
  - API
  - プロトコルと実装
- 現実装の評価
- 関連研究
- 今後の課題
- まとめ

提案手法

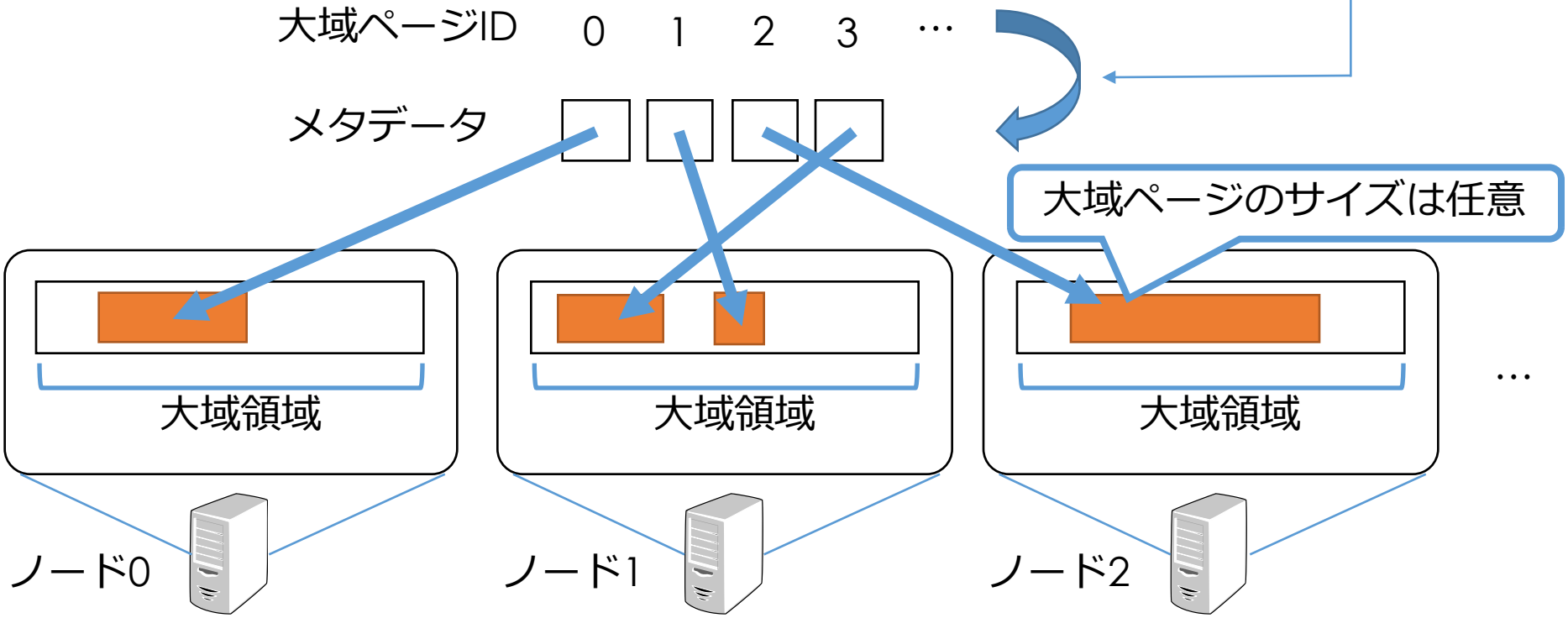
- 大域ページを**手動再配置**可能なPGASの設計
  - MassiveThreads/GASと同様のモデル
  - 通常のPGASよりも柔軟なデータ配置を実現
- **メタデータキャッシュ**によるget/put関数の高速化
  - 再配置と両立できるプロトコルを提案し，get/put関数による通信を最短でRDMA1回にできる

	PGAS	DSM	提案手法
明確なメモリアクセスコスト	○	×	○
データ再配置	×	○	○
メタデータキャッシュ	○	※	○

※：各処理系による

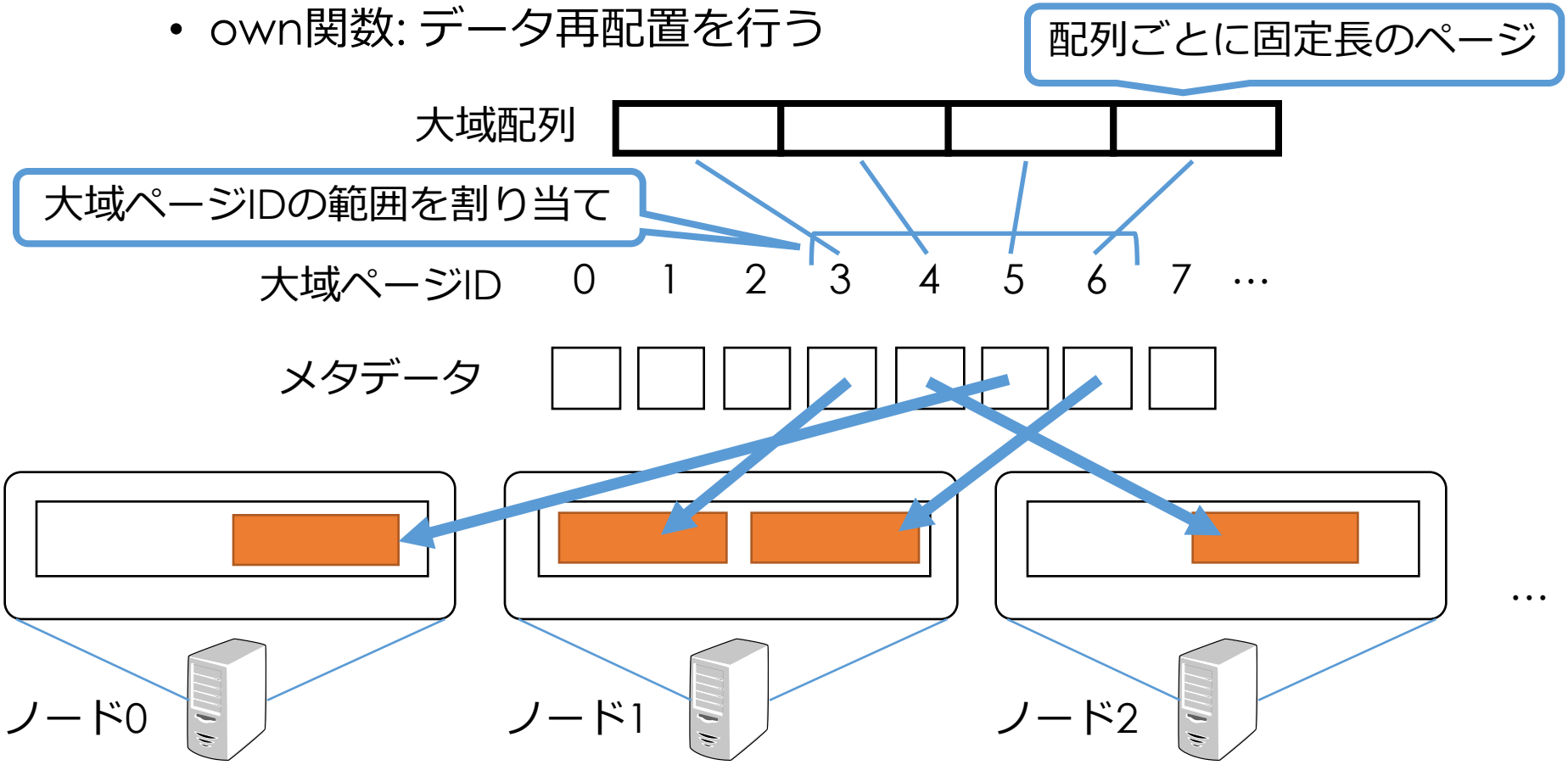
提案手法: API

- 低レベルAPI
  - 大域ページIDから対応するメタデータの取得
  - 大域ページを別ノードに再配置



提案手法: API

- 高レベルAPI: **大域配列**の機能
  - get/put関数: PGAS同様にデータアクセスを行う
  - own関数: データ再配置を行う



提案手法: API

- **get/put関数** (データアクセス)
  - 大域配列上のデータとローカルバッファの間でデータコピー
  - **大域配列上のデータはインデックスで指定**

void global_array::**get**(size_t src_index, void* dest_ptr, size_t size);

void global_array::**put**(void* src_ptr, size_t dest_index, size_t size);

- **own関数** (データ再配置)
  - インデックスに対応する大域ページを，自ノードに再配置

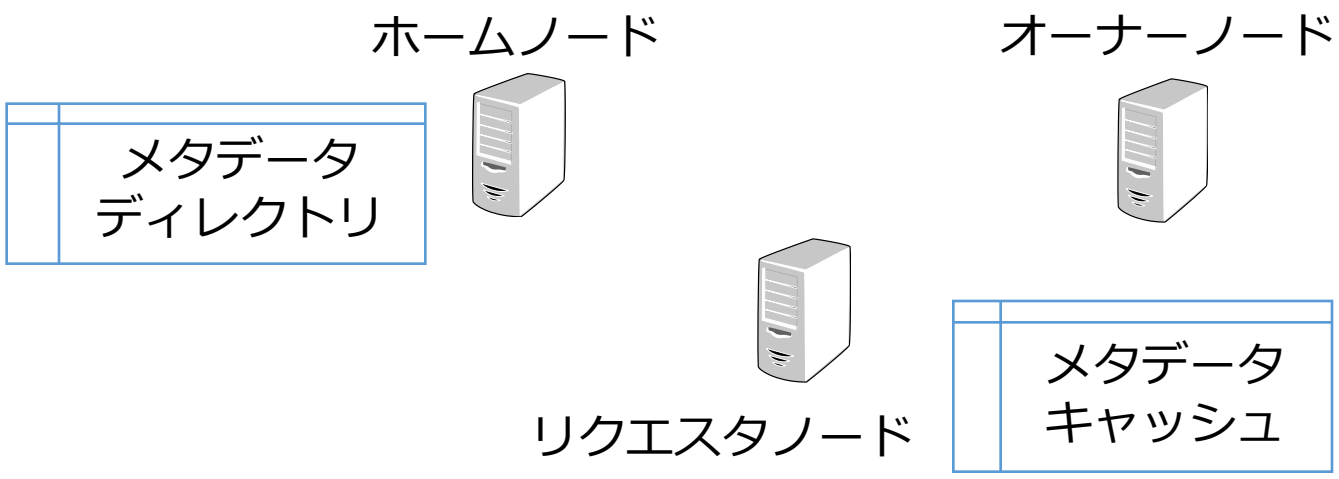
  void global_array::**own**(size_t index, size_t size);

提案手法: 実装

- get/put関数とown関数をどのように排他制御するか
- put (RDMA WRITE)とownの関係
  - 並列実行**できない**
  - 再配置実行中のputによる変更が失われるのを防ぐ
  - putとownの排他制御
    - RDMA WRITE実行中なら再配置を遅延
    - 再配置実行中ならRDMA WRITEを遅延
- get (RDMA READ)とownの関係
  - 並列実行**できる**
  - get呼び出し時点でのデータを再配置前の領域から読む (getの期待通りの動作と同じ)
  - ただし，再配置前の領域の解放にはgetも停止する必要あり (性能に大きく影響しないと考えられるため，詳細は割愛)

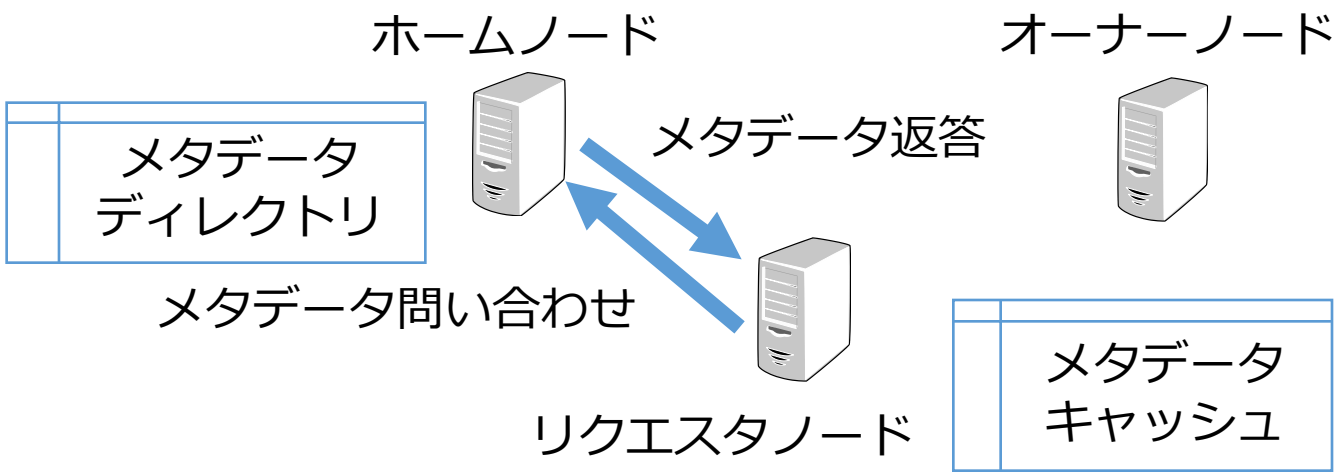
get/put関数の実装

- get/put関数実行の流れ



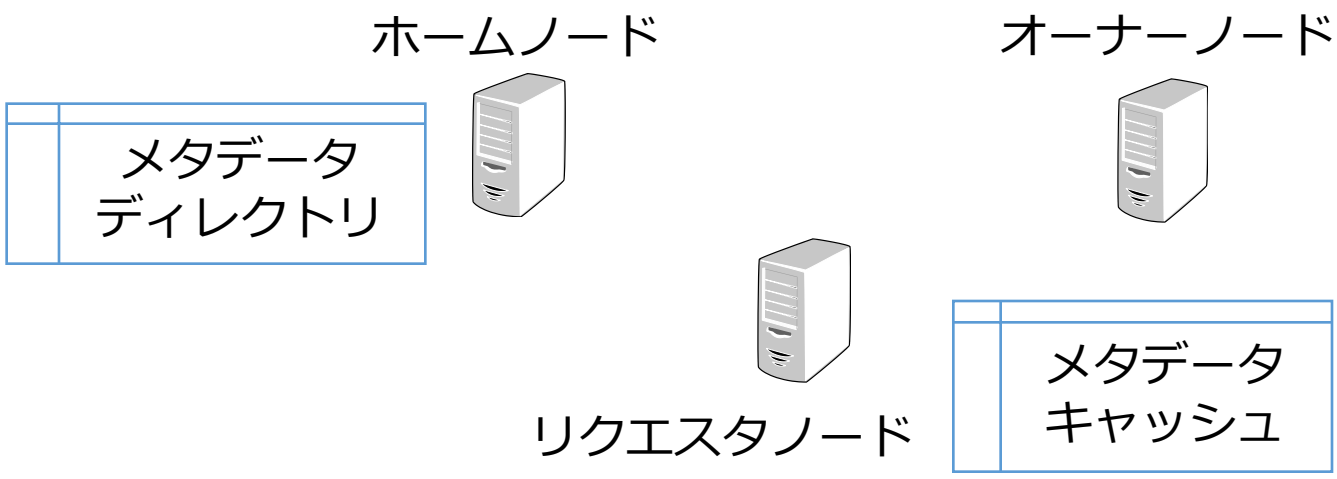
get/put関数の実装

- get/put関数実行の流れ
  1. **大域ページに対応するメタデータを取得**
     - メタデータがキャッシュされていればそれを使う



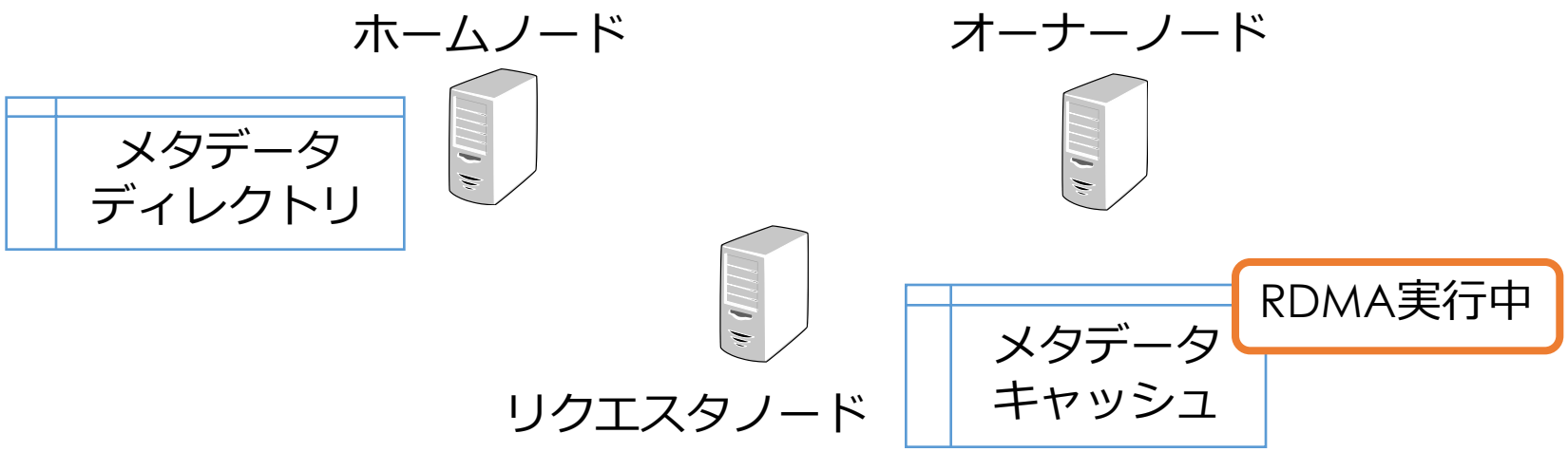
get/put関数の実装

- get/put関数実行の流れ
  1. 大域ページに対応するメタデータを取得
     - メタデータがキャッシュされていればそれを使う
  2. **(putの場合のみ)own実行中であれば終了まで待機**



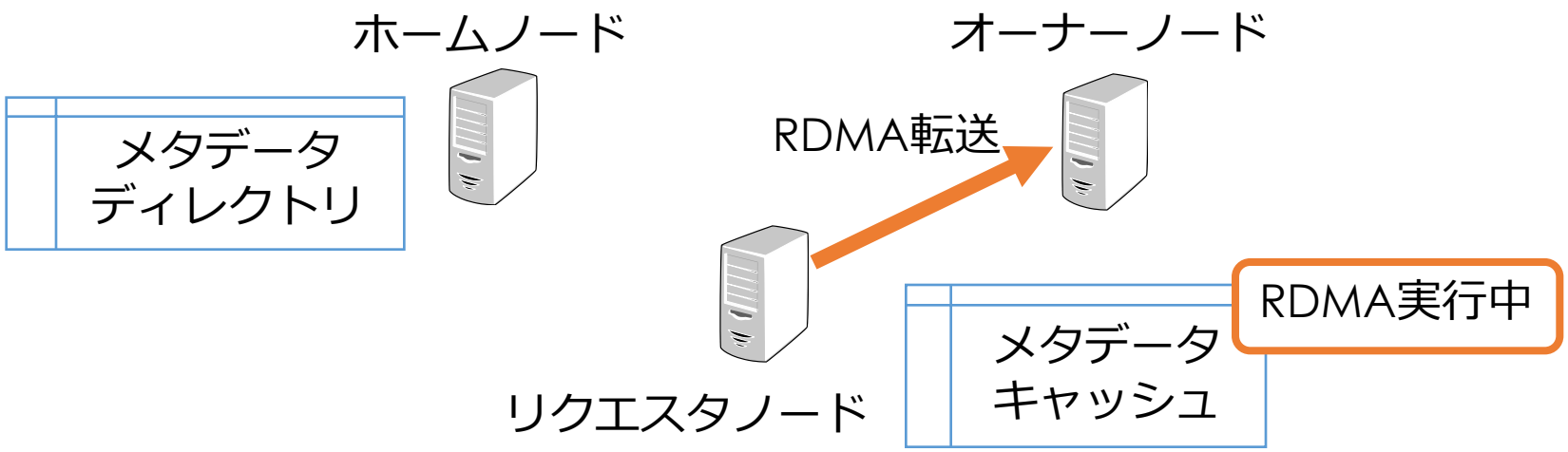
get/put関数の実装

- get/put関数実行の流れ
  1. 大域ページに対応するメタデータを取得
     - メタデータがキャッシュされていればそれを使う
  2. (putの場合のみ)own実行中であれば終了まで待機
  3. **RDMA実行中のフラグをセット**
     - put中に再配置が発生することを防ぐ



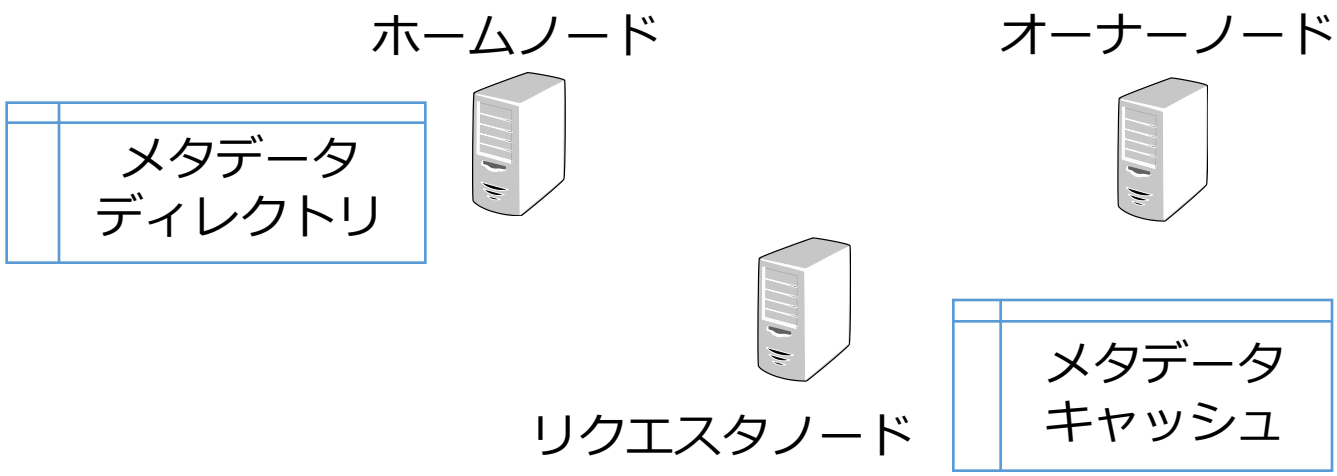
get/put関数の実装

- get/put関数実行の流れ
  1. 大域ページに対応するメタデータを取得
     - メタデータがキャッシュされていればそれを使う
  2. (putの場合のみ)own実行中であれば終了まで待機
  3. RDMA実行中のフラグをセット
     - put中に再配置が発生することを防ぐ
  4. **RDMAによってデータを転送**

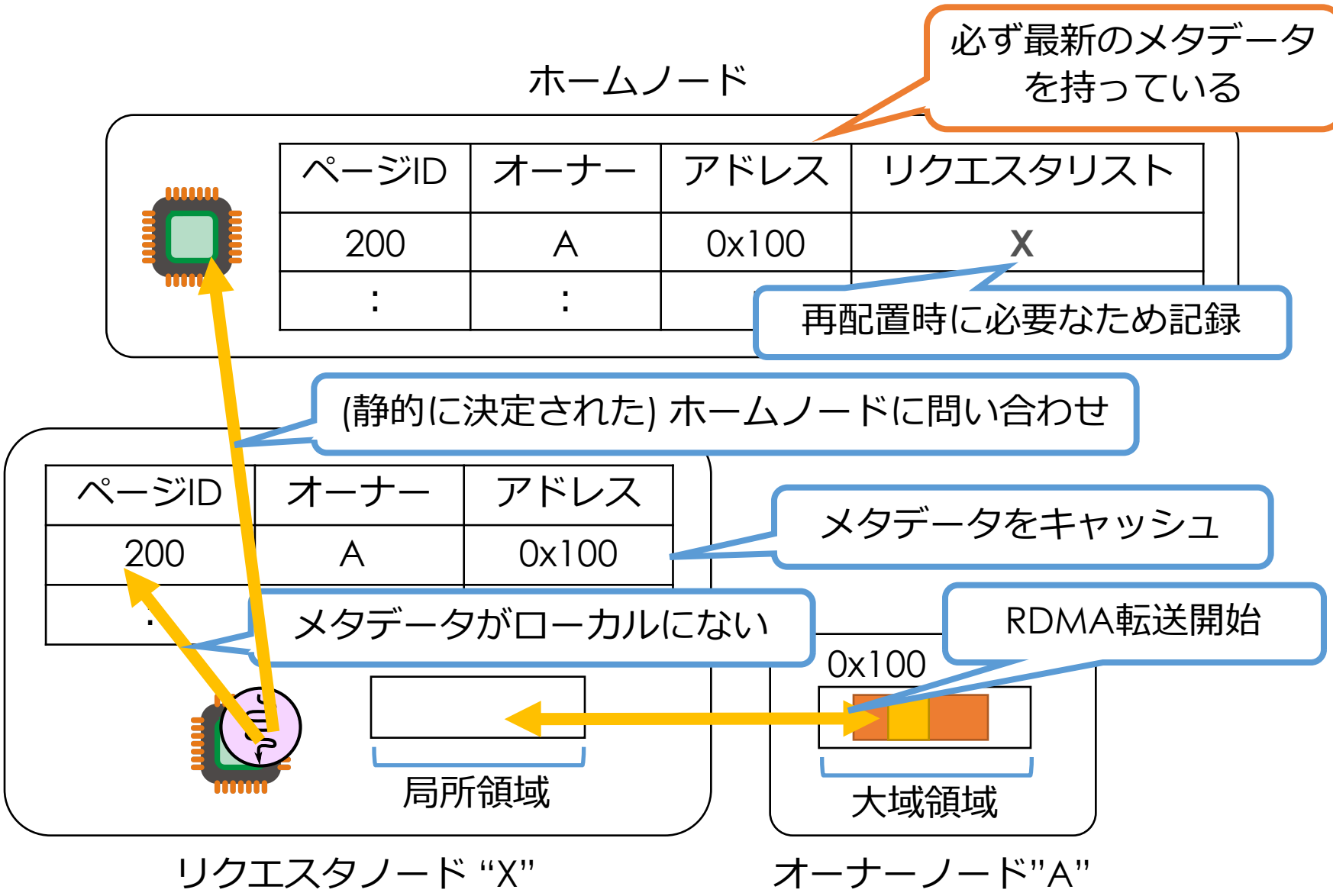


get/put関数の実装

- get/put関数実行の流れ
  1. 大域ページに対応するメタデータを取得
     - メタデータがキャッシュされていればそれを使う
  2. (putの場合のみ)own実行中であれば終了まで待機
  3. RDMA実行中のフラグをセット
     - put中に再配置が発生することを防ぐ
  4. RDMAによってデータを転送
  5. **RDMA実行中のフラグをクリア**

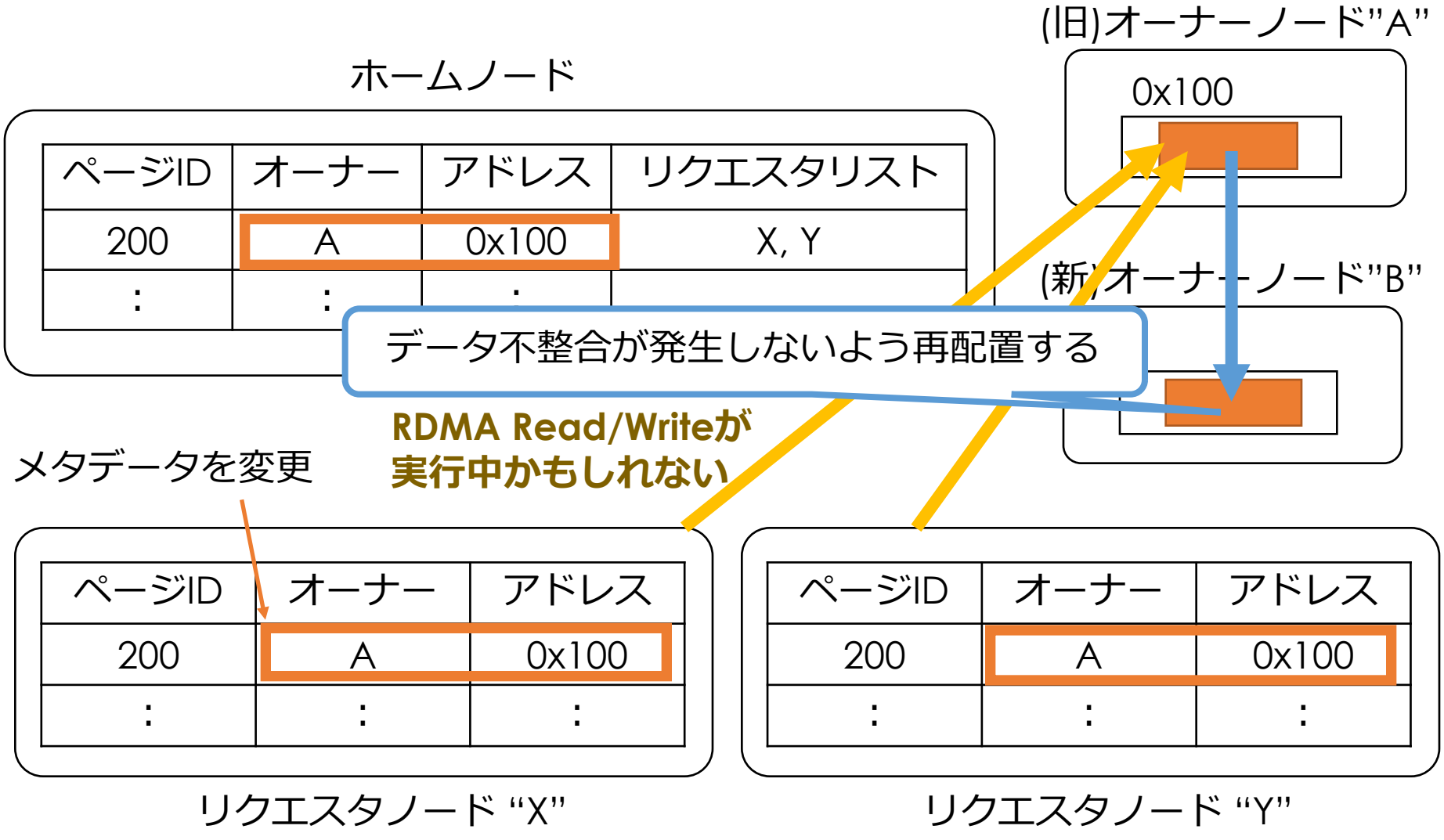


get/put関数の実装

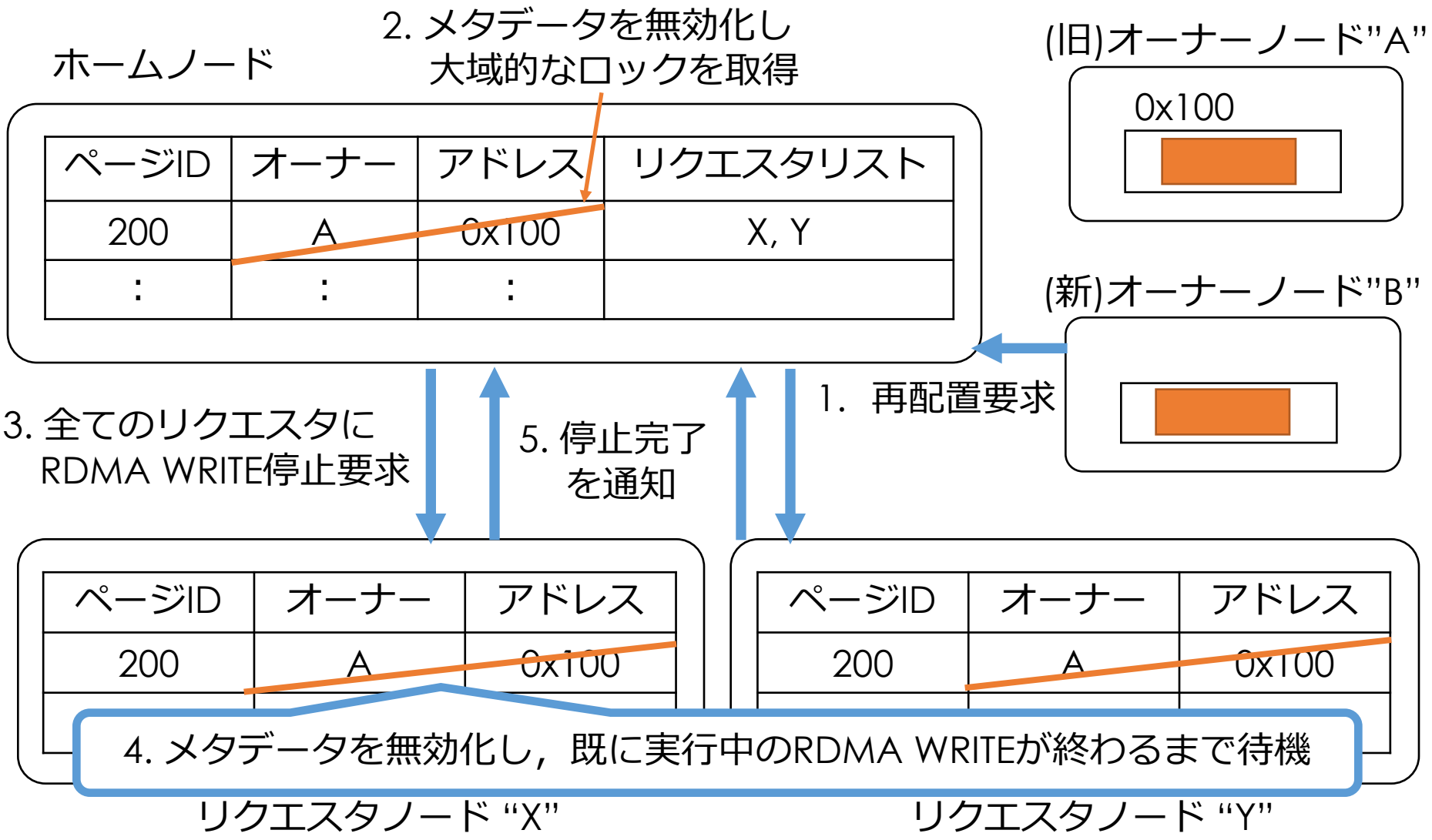


own関数の実装

- データ不整合が発生しないようにown関数を実行する



own関数の実装

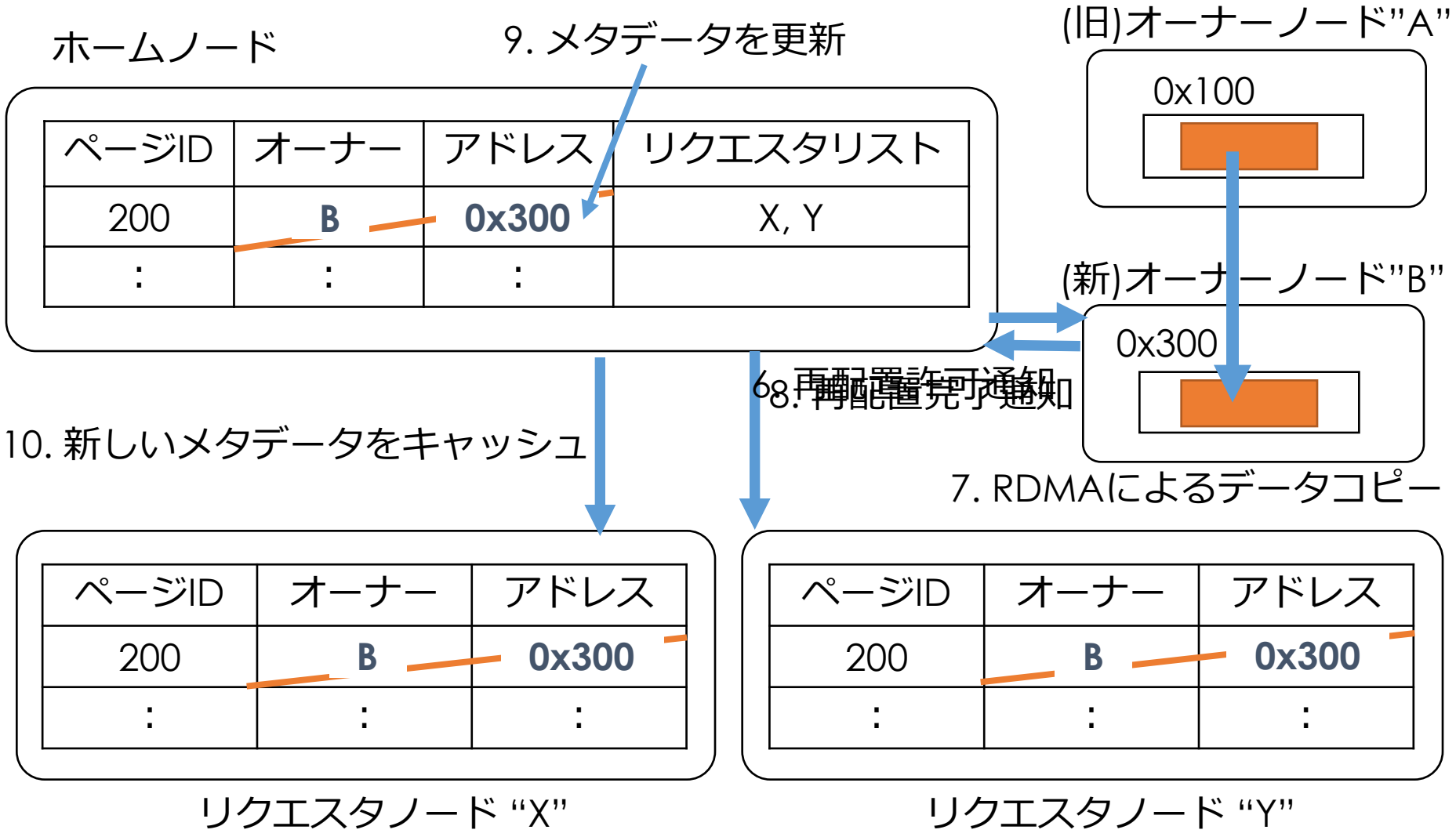


ページID	オーナー	アドレス	リクエスタリスト
200	A	0x100	X, Y
:	:	:	

ページID	オーナー	アドレス
200	A	0x100

ページID	オーナー	アドレス
200	A	0x100

own関数の実装


ホームノード
9. メタデータを更新
(旧)オーナーノード"A"
0x100
(新)オーナーノード"B"
0x300
8. 再配置許可通知
7. RDMAによるデータコピー
10. 新しいメタデータをキャッシュ
ページID
オーナー
アドレス
リクエスタリスト
200
B
0x300
X, Y
:
リクエスタノード "X"
リクエスタノード "Y"

発表の流れ

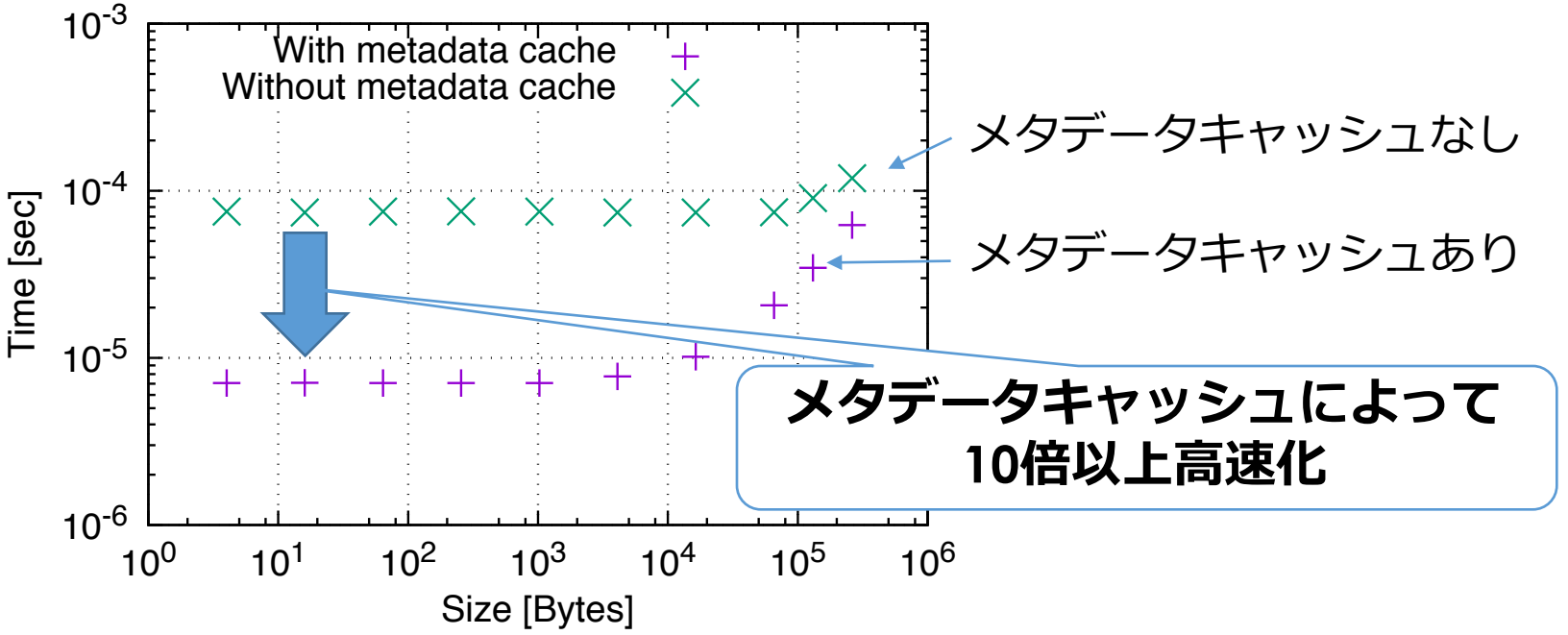
- 研究背景
- 提案手法
- **現実装の評価**
  - 評価手法，評価結果
- 関連研究
- 今後の課題
- まとめ

評価手法

- 東大のスーパーコンピュータ (FX10)で実験
  - CPU : SPARC64 Ixfx, 1.848 GHz
  - 1ノードあたり1コアのみ使用
- 現行の実装について
  - C++によるライブラリ
  - 提案したPGAS処理系 + (独自実装の)通信レイヤ
- 評価対象
  - get関数のレイテンシ

評価結果：get関数のレイテンシ

- get関数によるレイテンシを測定
  - 3ノードを使用 (ホーム，リクエスタ，オーナーをそれぞれ別々のノード上で実行)
  - 同じ領域にget関数を呼び続けた際のレイテンシ
  - 縦軸は時間，横軸は大域ページのサイズ



発表の流れ

- 研究背景
- 提案手法
- 現実装の評価
- **関連研究**
  - PGASにおけるメタデータキャッシュの導入
  - DSMにおけるRDMAの活用
- 今後の課題
- まとめ

関連研究

- PGASにメタデータキャッシュを導入した研究
  - IBM XLUPC [M. Farreras, 2009]
  - Global Arrays [D. Chavarría-Miranda, 2013]
- PGASはメタデータキャッシュが容易
  - メタデータが静的に決定されるから
  - 提案手法では再配置を認めるのでより複雑
- PGASのメタデータキャッシュにおける問題点
  - **問題点**：全てのメタデータを全ノードでキャッシュすることは**メモリ消費量**の観点からスケーラブルでない
  - **解決策**：必要になったメタデータだけをキャッシュする

関連研究

- Argo [S. Kaxiras, 2015]
  - Distributed Shared Memory (DSM)の処理系の一つ
  - RDMAのみで全ての操作を行う
    - データ転送だけでなく，**ディレクトリ操作もRDMA化**
  - データキャッシュの無効化/更新を各ノードごと独立に判断
    - 各ノードごとに独立したディレクトリを持つ
    - 読み込み中/書き込み中のノード数に応じて，
      キャッシュ管理の方法を自動的に判断

発表の流れ

- 研究背景
- 提案手法
- 現実装の評価
- 関連研究
- **今後の課題**
- **まとめ**

今後の課題

- 実装上のオーバーヘッドを低減
- 全てのディレクトリ操作をRDMA化
  - リモートアトミック命令が必要
- データキャッシュ機能の追加
  - プログラマが手動でチューニングできるデータキャッシュ機能が必要

まとめ

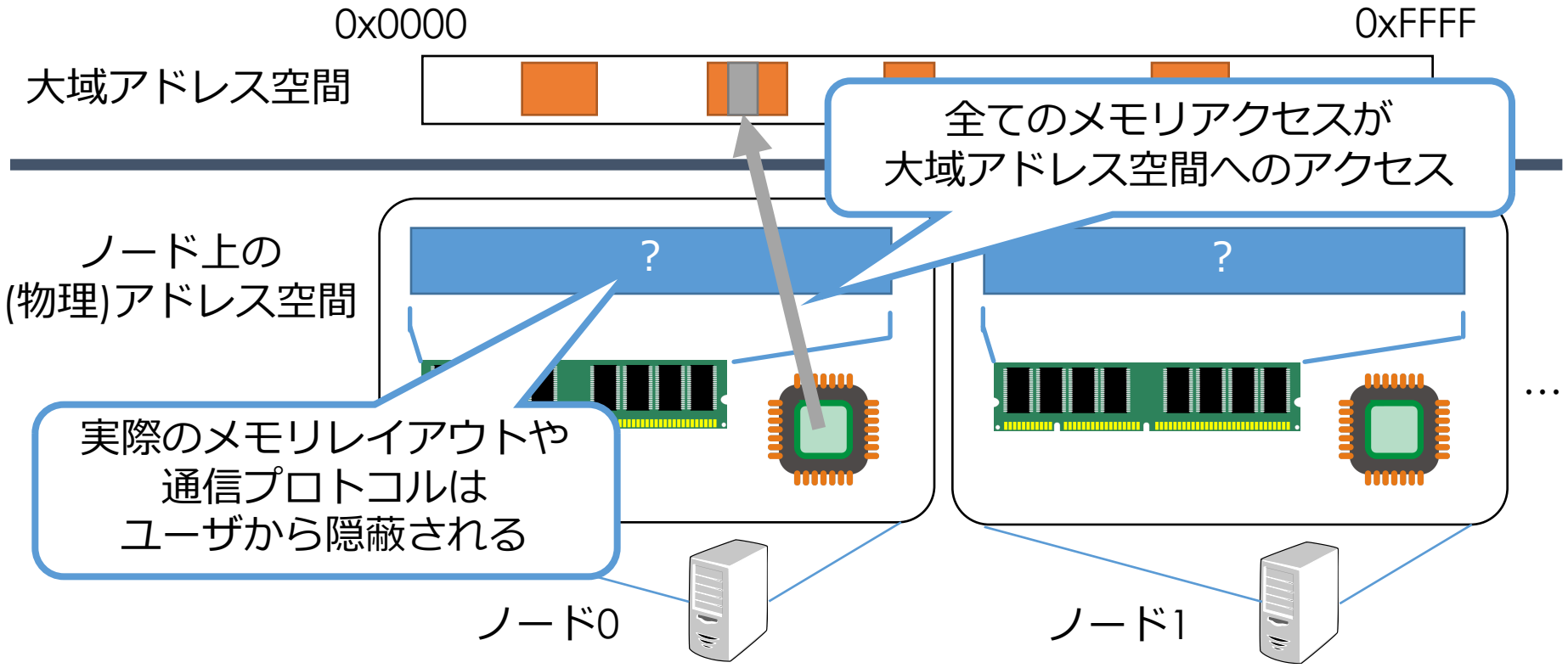
- 再配置可能なPGASの設計と実装
  - **データ再配置**によって動的負荷分散後の通信を
  - **メタデータキャッシュ**によって
    get/put関数が最短でRDMA1回で済む
  - 再配置とメタデータキャッシュを両立するための
    ディレクトリ構造と通信プロトコル
- 現実装の評価
  - メタデータキャッシュの導入によって，
    get関数によるレイテンシが10倍以上高速化された

DSMの特徴

- **自動的なデータキャッシュ**機能を持つ
  - リモートノード上のデータを自動的にキャッシュ
  - 通信回数・サイズ共に削減できる
- 緩和型のキャッシュコヒーレンシ
  - キャッシュプロトコルの代表例としては Release Consistency [Gharachorloo, 1990] など
- **自動的なデータ再配置**を行う
  - オーナーノードを自動的に変更する

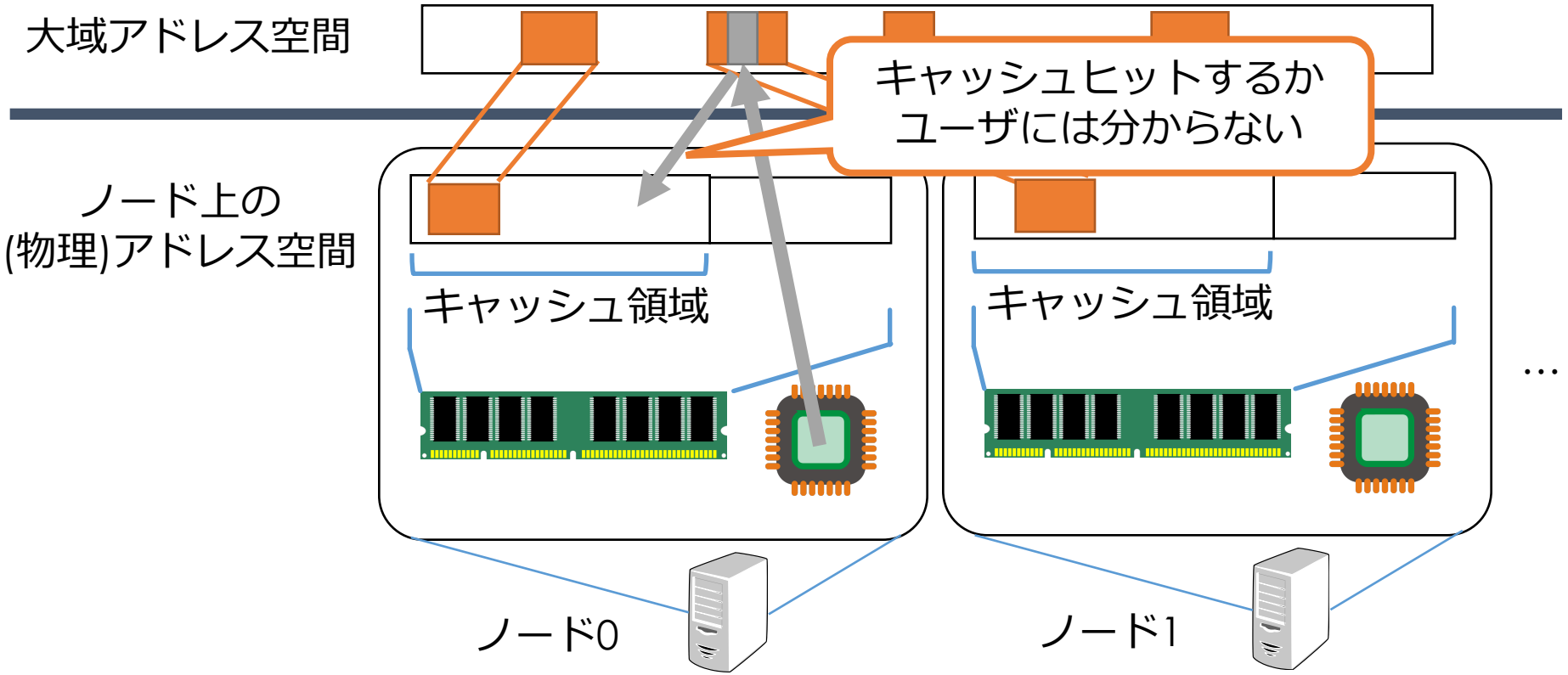
Distributed Shared Memory (DSM)

- ユーザが扱うメモリ領域全てが共有される
  - システムはOSのメモリ保護機構を活用して通信を隠蔽
- 自動的なデータキャッシュ・**自動的なデータ再配置**



DSMの問題点

- メモリアクセスコストが不明確
  - 通信の発生時点が分かりにくく，手動チューニングが難しい
  - 通信粒度の調整も難しい



Partitioned Global Address Space (PGAS)

- 大域ページのキャッシュは行わない
  - リモートノード上の大域ページへのアクセスは **必ず通信を発生させる**
  - プログラマが手動で局所領域にデータキャッシュするようなコードを記述してくれることを期待する
- 大域ページの配置は静的に固定
  - 再配置は起きない (オーナーは変化しない)
- PGAS処理系の例
  - UPC, Global Arrays, X10, Chapel, Co-array Fortranなど