DMI：計算資源の動的な参加/脱退をサポートする大規模分散共有メモリインタフェース

近山・田浦研究室 B4 原健太朗

2009.2.20

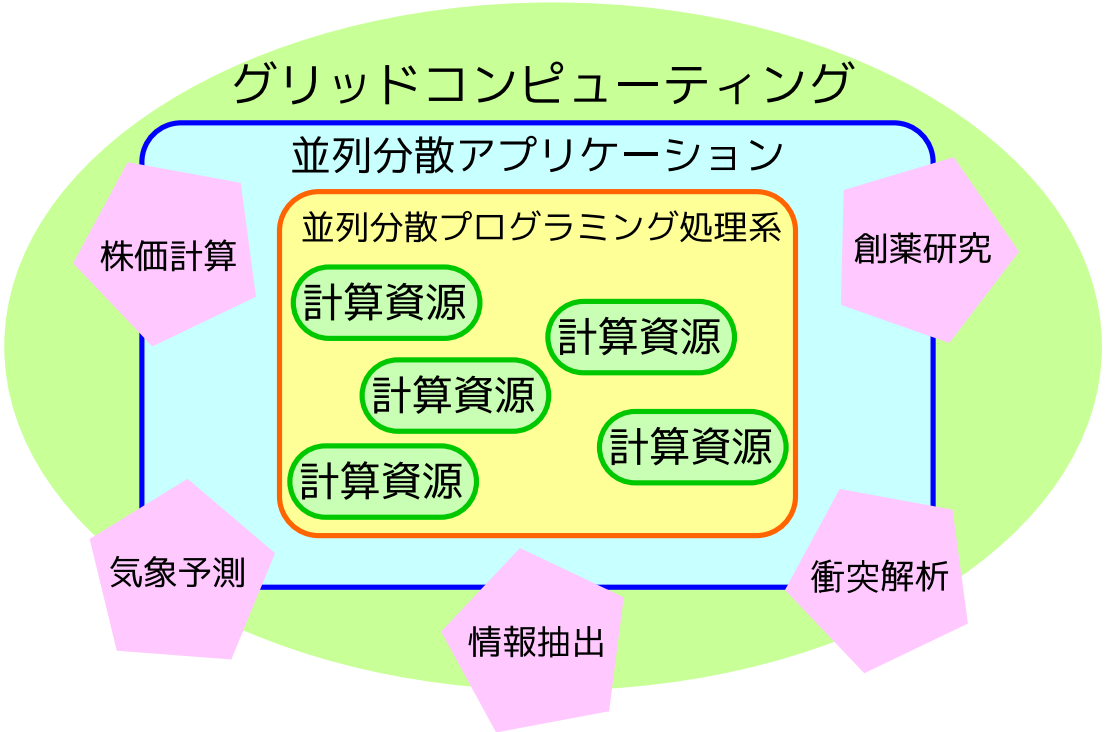
発表の流れ

- 序論
- コンセプト
- 実装
- 評価
- 結論

1. 序論

本研究の背景

- 並列分散コンピューティングの発展
  - → 産業界での応用分野 (金融，創薬，気象，…) での多様な並列分散アプリケーション
  - → 計算環境の高性能化と大規模化
- 基盤となる並列分散プログラミング処理系への要請が多様化

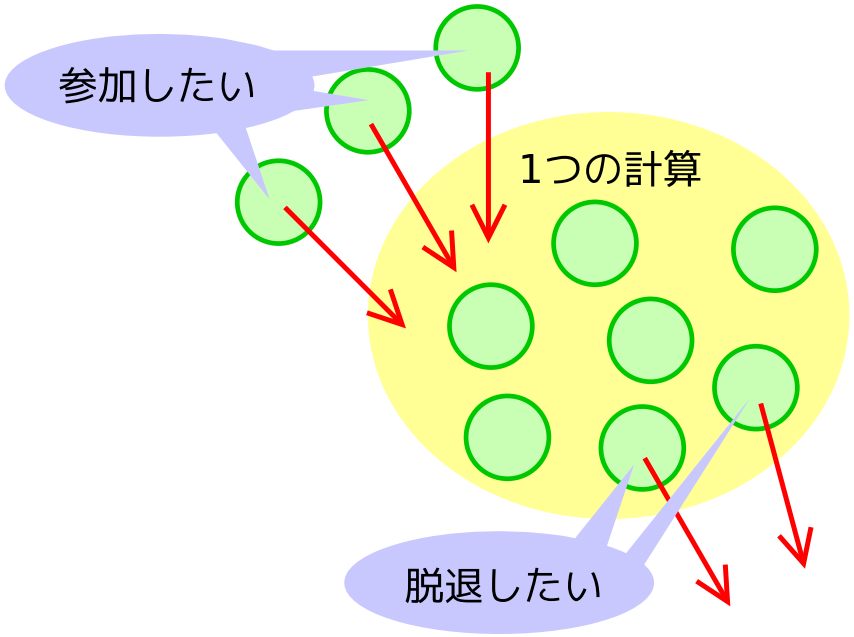


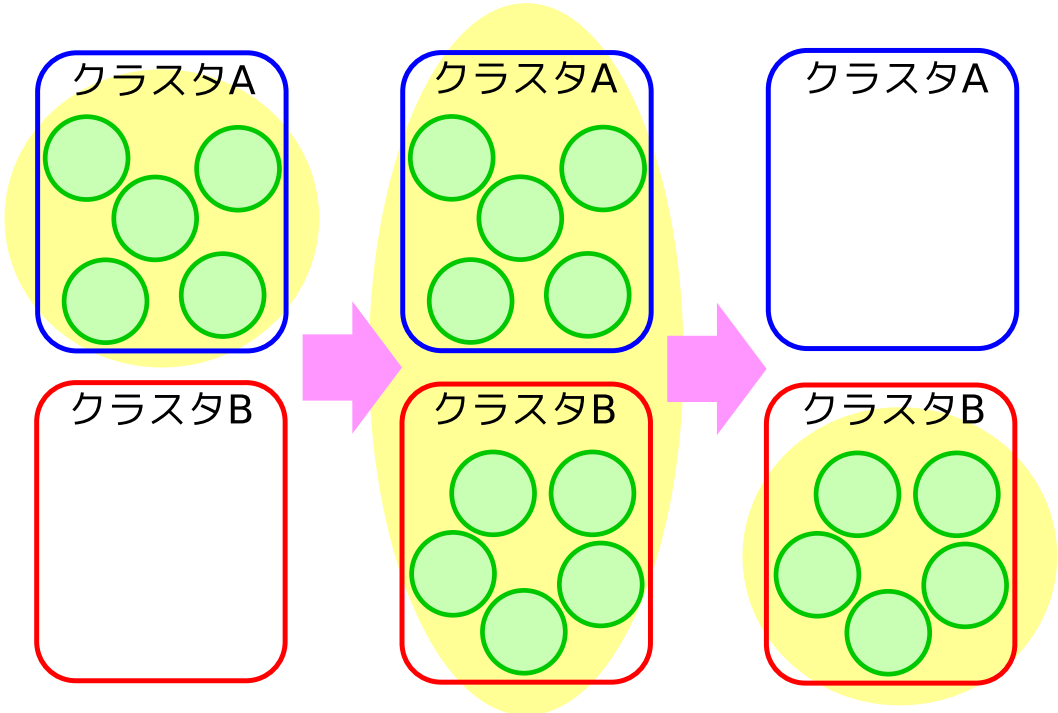
並列分散プログラミング処理系に対する要請

- 共通の要請：
  - → 記述力
  - → スケーラビリティ
- 機能面での要請：
  - → 複雑なネットワーク構成への対応
  - → 計算資源の動的な参加/脱退のサポート
  - → 耐故障
  - → …

計算資源の動的な参加/脱退のサポート (1)

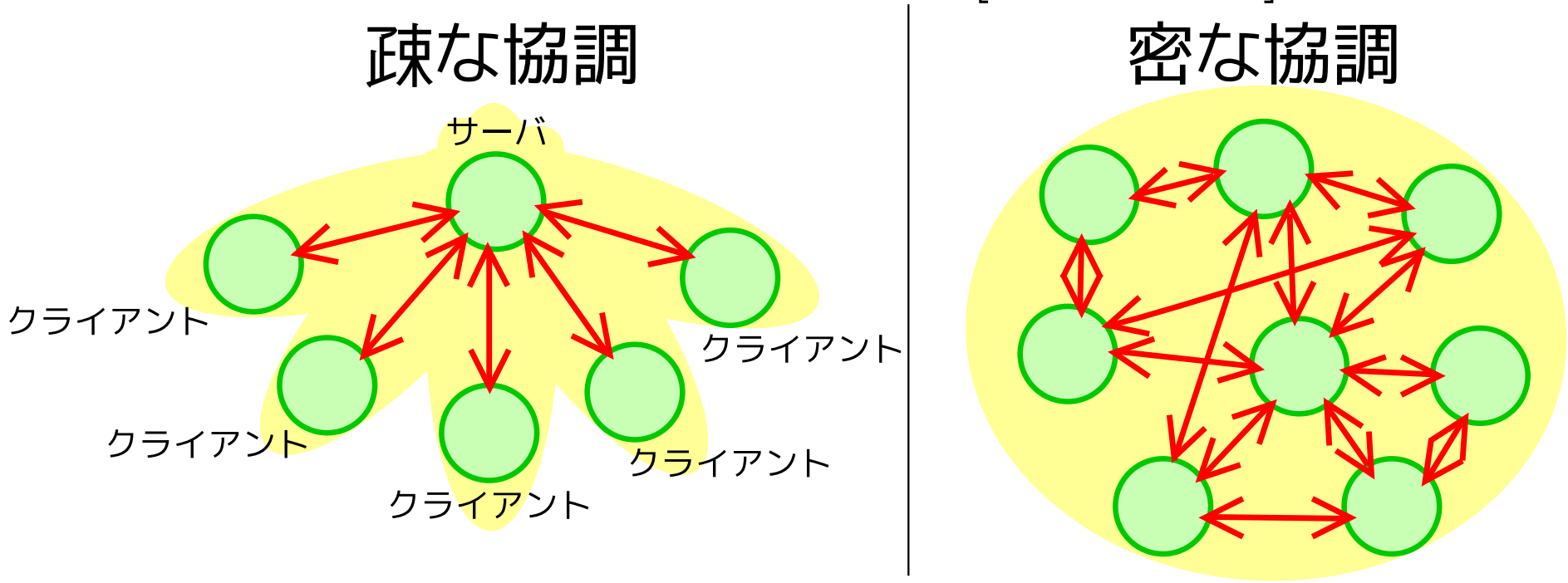
- 現状：計算資源は個人のものではない
  - → クラスタ環境の運用ポリシー，課金制度，…
- 要請：
  - → 計算資源の参加/脱退を越えて 1 つの計算の継続実行したい
  - → 計算環境のマイグレーションを実現したい





計算資源の動的な参加/脱退のサポート (2)

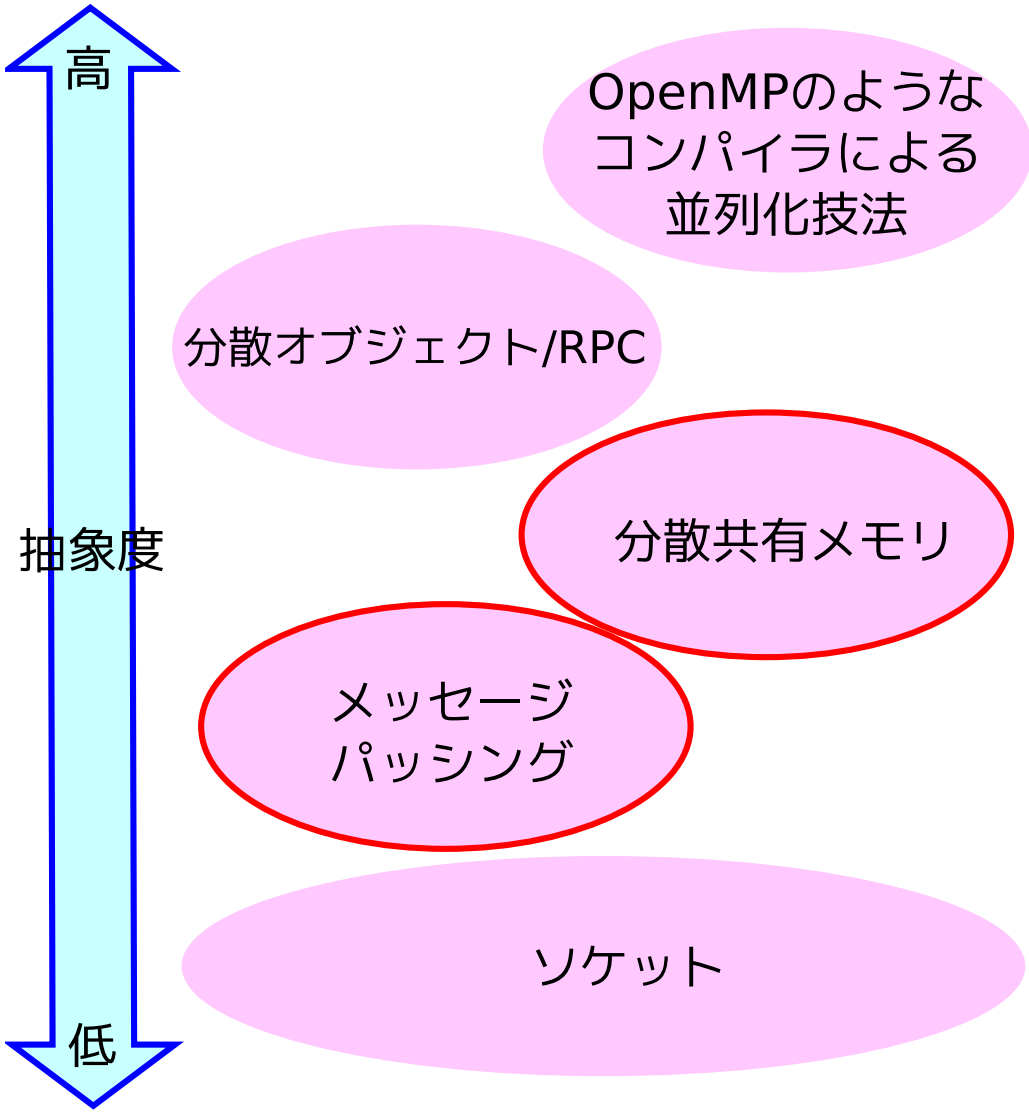
- 例：クライアント・サーバ方式
  - → 特定の計算資源に負荷が集中してスケーラブルでない
  - → 計算資源同士が疎に結び付くモデルの上で効率的に実行可能なアプリケーションは限られる [Taura,'01]



- 多数の計算資源がもっと密に協調するアプリケーション領域もサポートする枠組みが必要

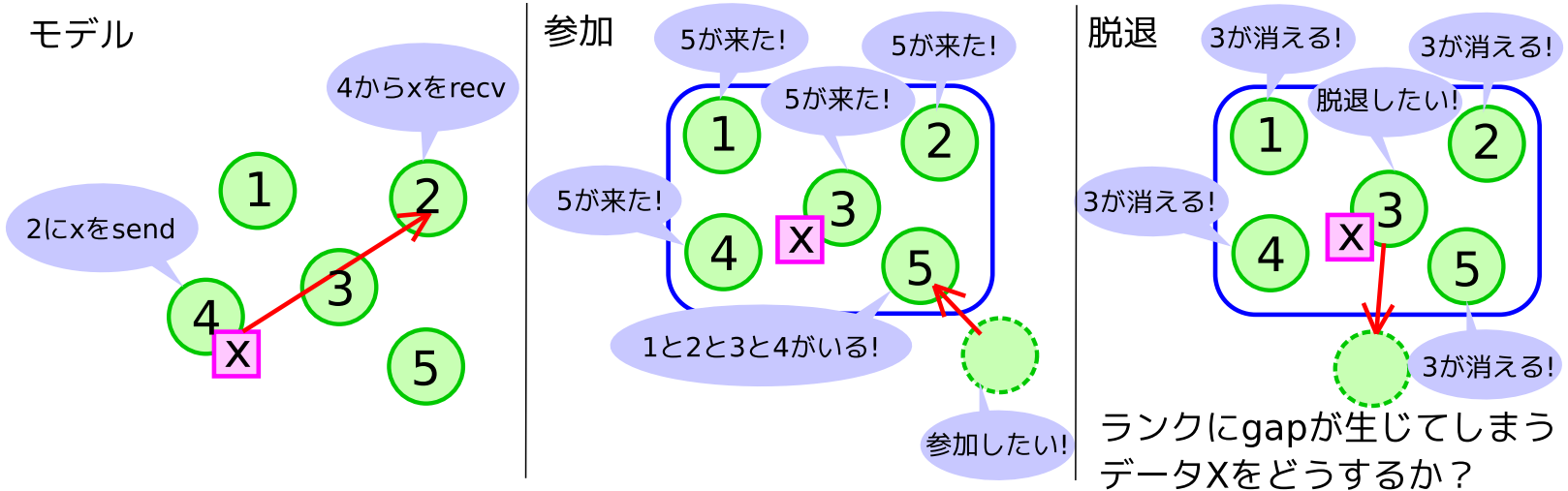
並列分散プログラミングモデル

- どのモデルをベースにすべきか?



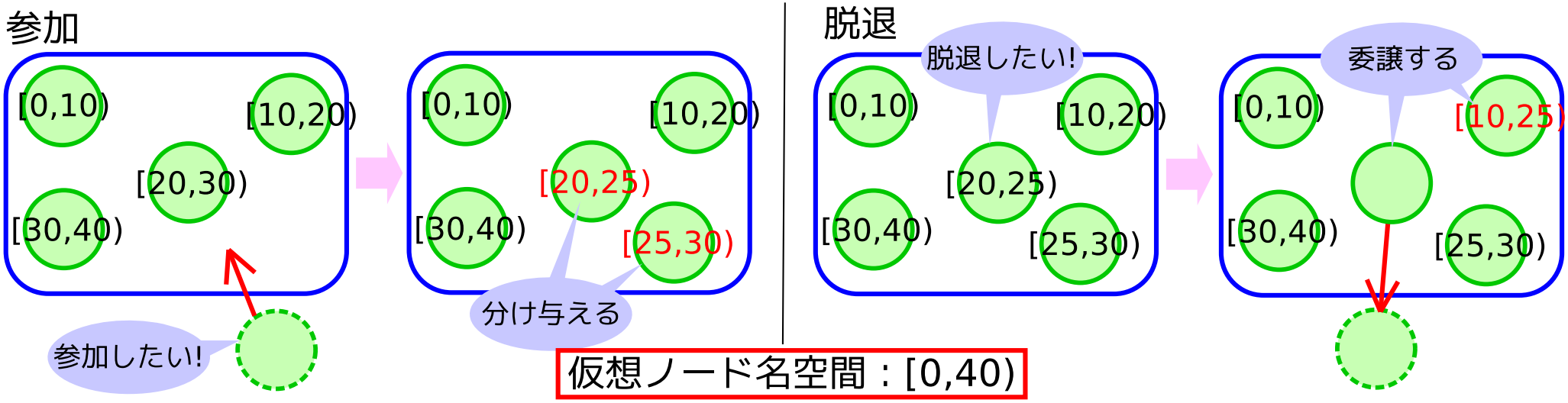
メッセージパッシング

- 一意的なランクを使用したデータの送受信 (send/recv) を明示的に記述
- データの所在管理はユーザプログラム側に任される
  - → ユーザプログラム側が「系内に誰がいて，誰がどのデータを持っているのか」を把握している必要がある
  - → 参加/脱退時には「系内に誰がいて，誰がどのデータを持っているのか」の更新作業が全ノード上で必要
    - ユーザプログラムの記述は相当に複雑化



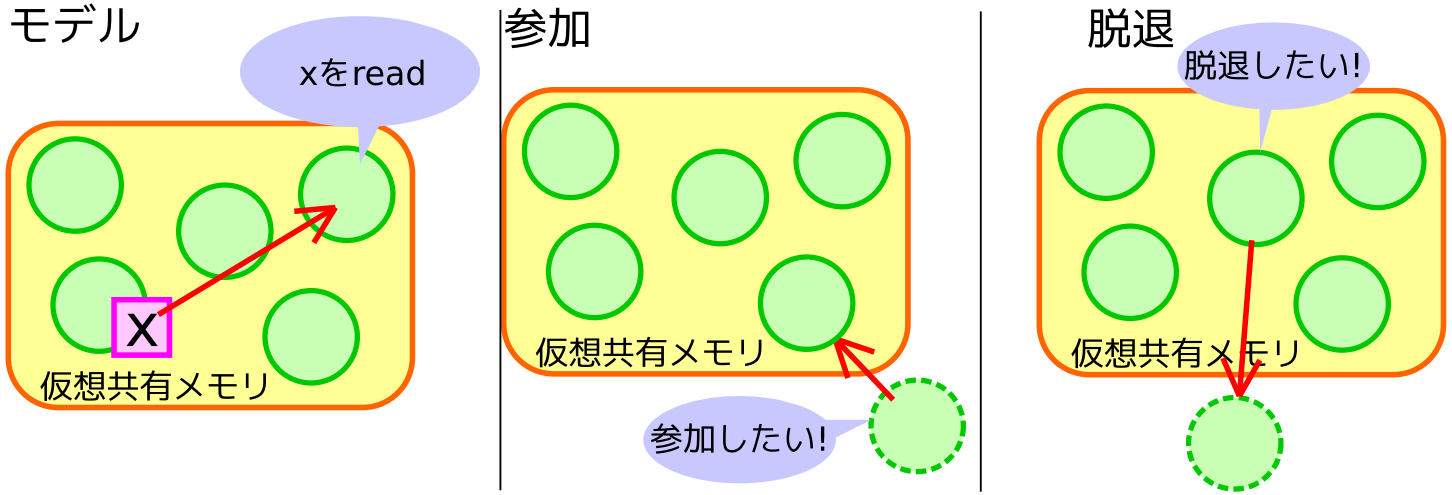
関連研究

- Phoenix[Taura et al,'03]：
  - → メッセージパッシングベースで計算資源の動的な参加/脱退に対応
  - → ユーザプログラムは，物理的なノード名とは別の「仮想ノード名」を使ってデータの送受信を記述
  - → スケーラブルだが**記述は複雑**



分散共有メモリ

- データ通信 (send/recv) を隠蔽して仮想的な共有メモリへのメモリアクセス (read/write) に抽象化
- データの所在管理は処理系側が行う
  - → ユーザプログラム側では「誰がどのデータを持っているか」はおろか「系内に誰がいるか」さえ把握している必要がない
  - → 参加/脱退時には本人が「参加/脱退したい」と言うだけで良い
    - ユーザプログラムの記述は容易

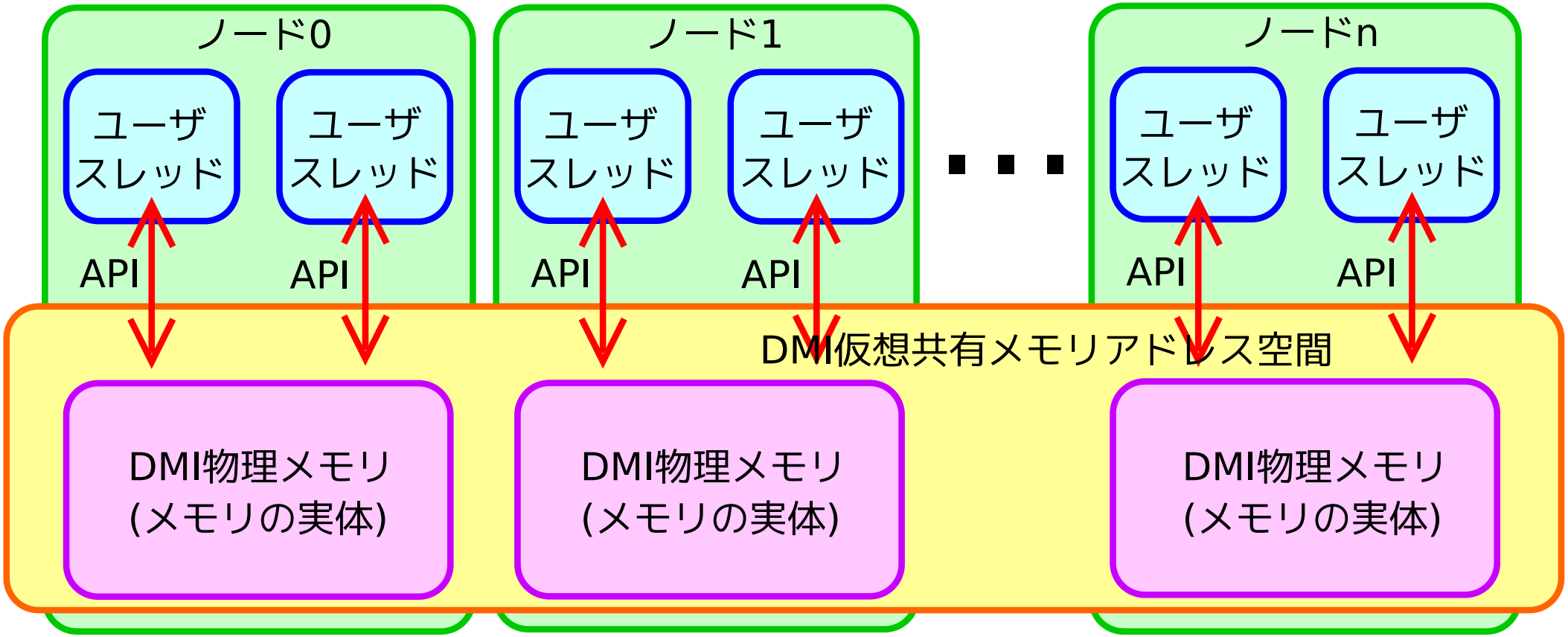


本研究の提案

- DMI：Distributed Memory Interface
  - → **分散共有メモリをベースとして，多数の計算資源が密に協調するアプリケーション領域に対しても計算資源の参加/脱退をサポートする並列分散ミドルウェア基盤**
  - → **ノードの参加/脱退に対応したコンシステンシプロトコルを提案**
  - → **独自の設計コンセプトに基づき分散共有メモリとしての機能と性能を追求**

2. コンセプト

(1) 大規模分散共有メモリの構成



- **大規模メモリ**
- **分散レベルの並列性とともにマルチコアレベルの並列性を活用**

(2) ミドルウェア基盤としてのインタフェース

DMI_read(addr, size, buf); addrからsizeバイトをbufにread
DMI_write(addr, size, buf); addrにbufからsizeバイトをwrite

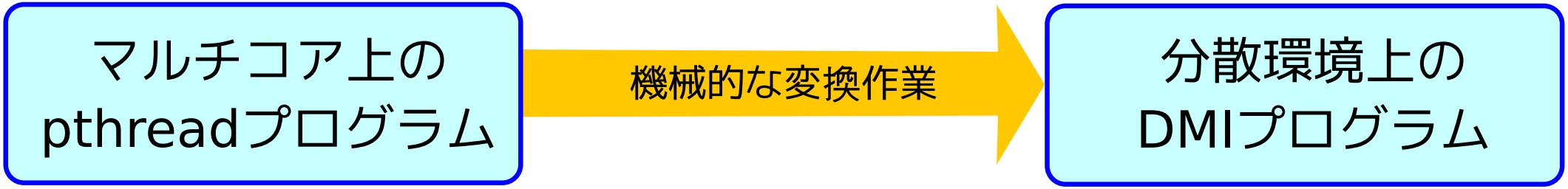
- API によるメモリアクセスとし，メモリ管理は全てユーザレベルで実装：
  - → 任意のページサイズを指定可能
    - ♦ ページフォルト回数を大幅に削減可能
  - → OS のアドレッシング範囲に囚われないネットワークページング
  - → 非同期モードの read/write などを整備
    - ♦ 性能チューニングの機会を提供してインクリメンタルな開発を支援
- 柔軟性，汎用性，機能拡張性に富んだインタフェース設計

(3) 計算資源の動的な参加/脱退をサポート

- SPMD 型ではなく pthread 型のスタイルを採用：
  - → ×：SPMD 型は「全員」の時系列的な挙動が明確な処理の記述に特化
    - ♦ 参加/脱退に対応させる上では「全員」の概念を隠す必要がある
  - → ○：動的なスレッド生成/破棄が可能な pthread 型では，参加/脱退に応じた動的な並列度変化をプログラムとして表現可能
- 便利な API：
  - → 現在参加中のノードの状態を取得する API
  - → ノードの参加/脱退イベントをポーリングする API
  - → 参加ノードの総コア数が目標値になるまで待機する API
  - → …

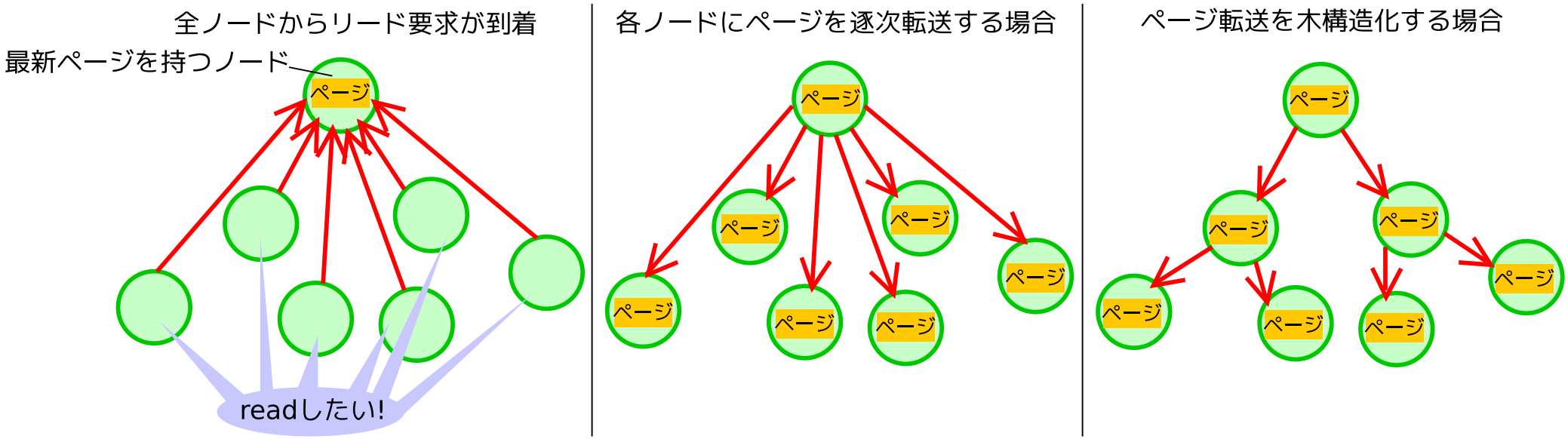
(4) マルチコア並列プログラムとの類似性

- **要件：マルチコア並列プログラムを分散化させる敷居を下げたい**
- **対策：pthread プログラムのシンタックスやセマンティクスを忠実に模擬**



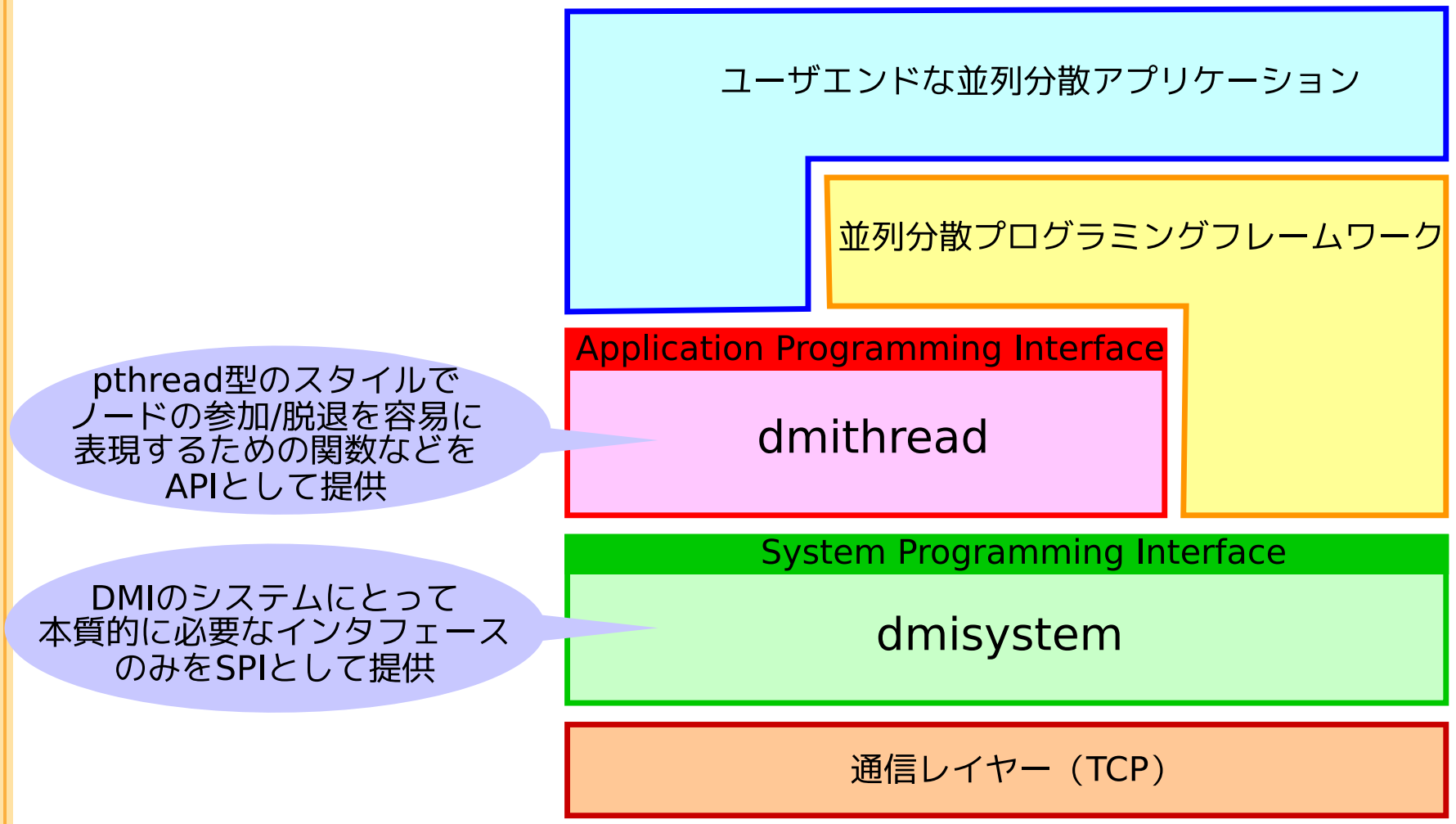
(5) ページ転送の動的負荷分散

- 要件：性能上，集合通信の最適化は必須
  - → しかし「全員」の概念を隠す pthread 型のスタイルでは集合通信の導入は不可能
- 対策：ページ転送を動的に木構造化させる



DMI の位置付け

- 並列分散プログラミングフレームワークにおけるミドルウェア基盤



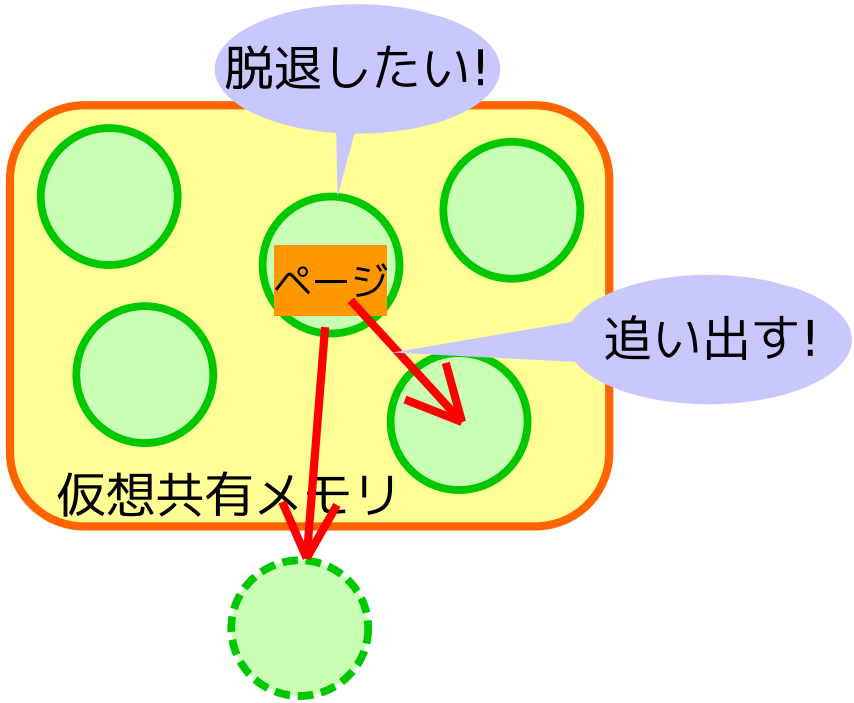
関連研究

- **大規模分散共有メモリ**
  - → DLM[Midorikawa et al,'07]
  - → Cashmere-VLM[Dwarkadas et al,'99]
- pthread **の分散系への拡張**
  - → DSM-Threads[Mueller,'97]
- **任意のページサイズを実現** (region-based)
  - → CRL[Johnson et al,'95]
  - → HIVE[Baiardi et al,'00]
- **マルチコアレベルの並列性の有効活用**
  - → UPC[Carlson et al,'05]

3. 実装

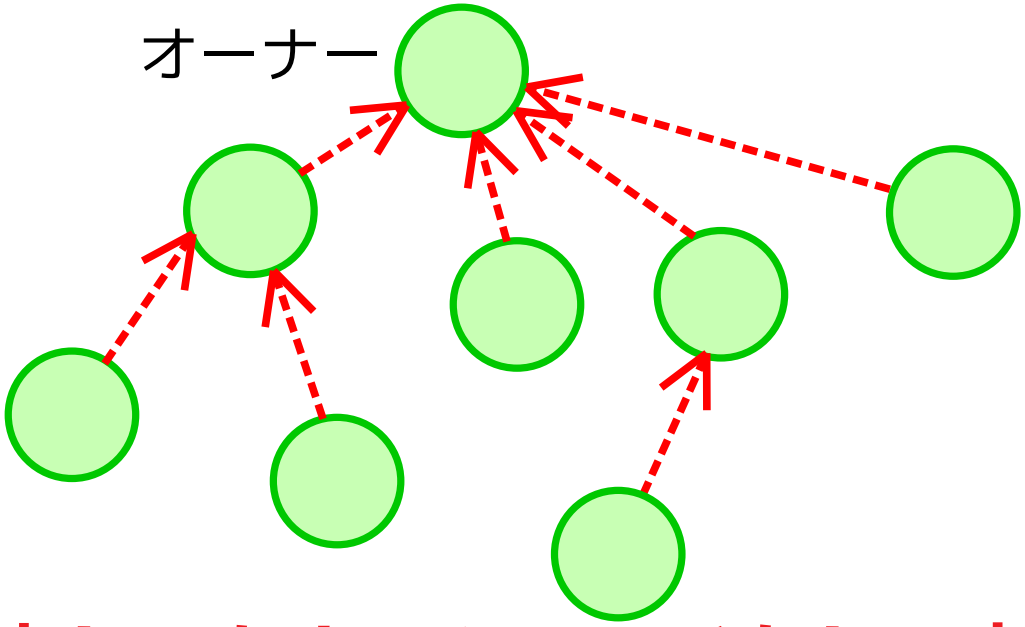
DMI のプロトコルへの要件

- 動的な参加/脱退を実現するには…
  - → ホームノードのような固定的なノードの設置は不可
  - → 脱退前には持っているページを追い出す必要がある
- プロトコルへの要件：
  - → 固定的なノードを設けることなく「ページフォルト」と「追い出し」をハンドリング



既存のプロトコルを観察

- 「ページフォルト」に関しては...
  - → オーナー追跡グラフ [Li et al,'89] で解決可能
    - ♦ オーナーの参照関係を辿ることで真のオーナーに到達可能
    - ♦ ページフォルト時にはリクエストをフォワーディング

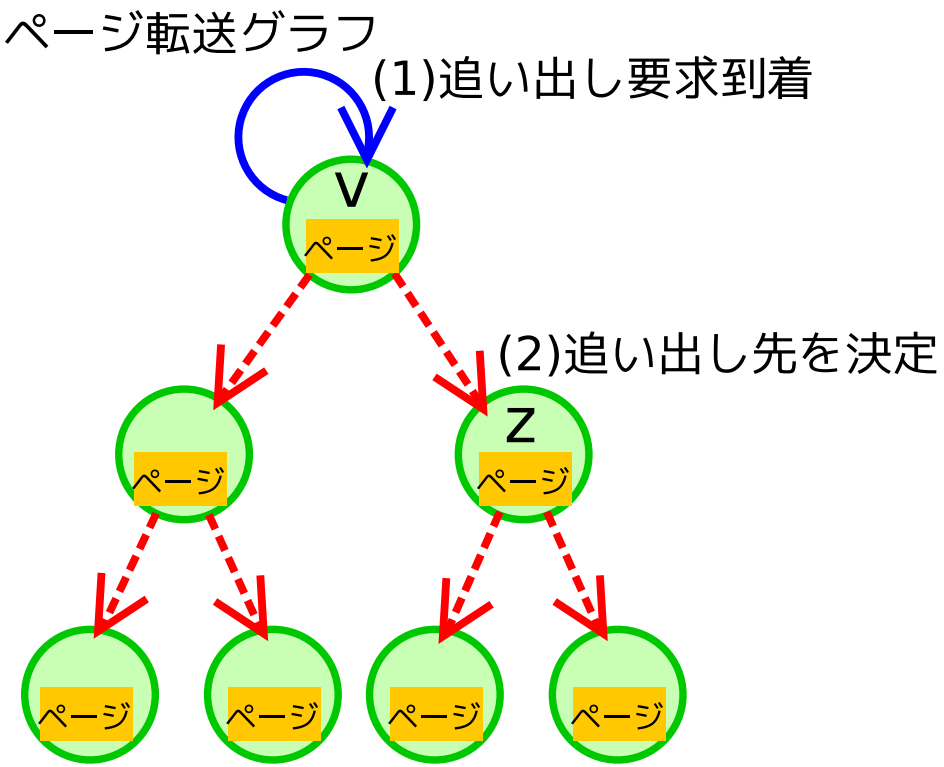


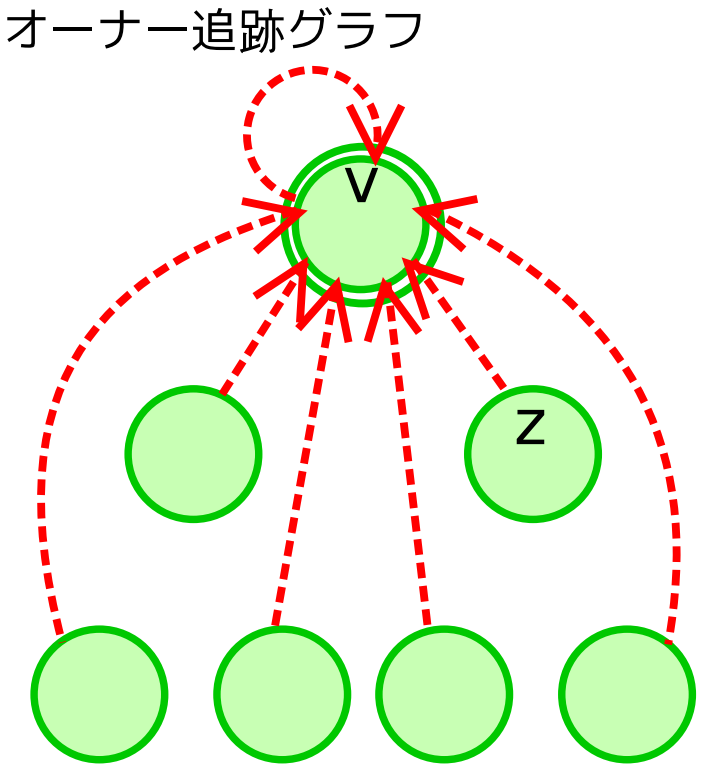
- しかし「追い出し」をホームノードなしで実現するプロトコルはない (ようだ)

DMI のプロトコル

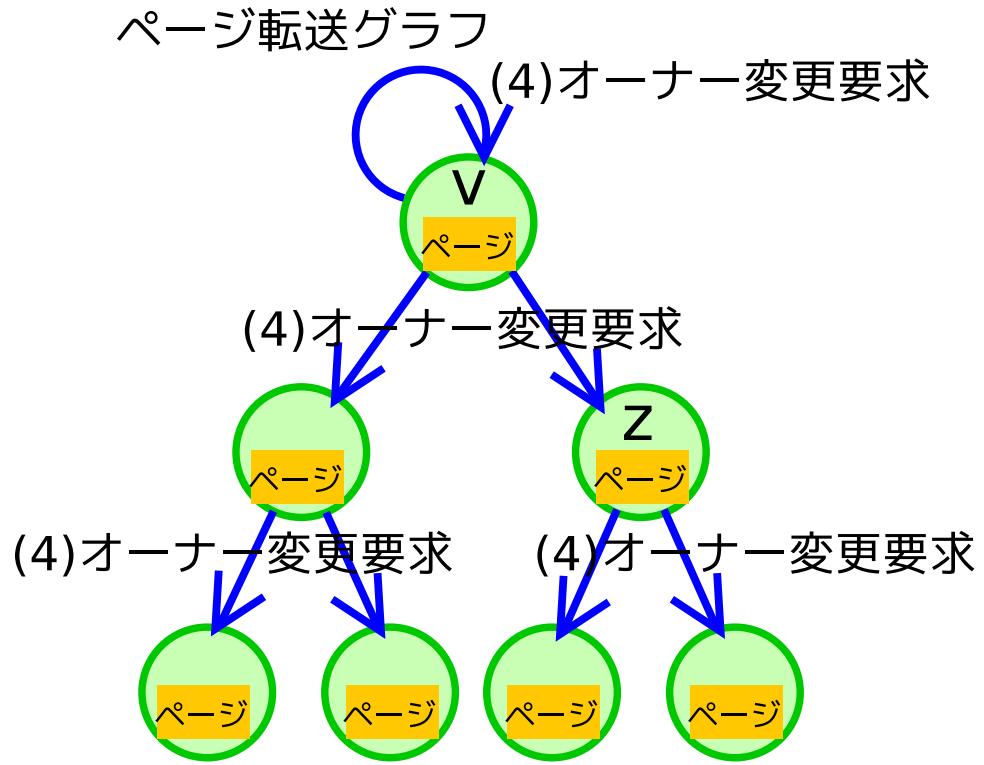
- 「追い出し」に対応すべく，Dynamic Distributed Manager Algorithm[Li et al,'89] を修正・拡張
  - → リードフォルト (修正)
  - → ライトフォルト (修正)
  - → オーナーの追い出し (拡張)
  - → オーナー以外の追い出し (拡張)
- DMI のプロトコルの特徴
  - → ページ単位での Sequential Consistency
  - → Single Writer/Multiple Reader 型
  - → Invalidate 型
  - → オーナー追跡グラフを形成

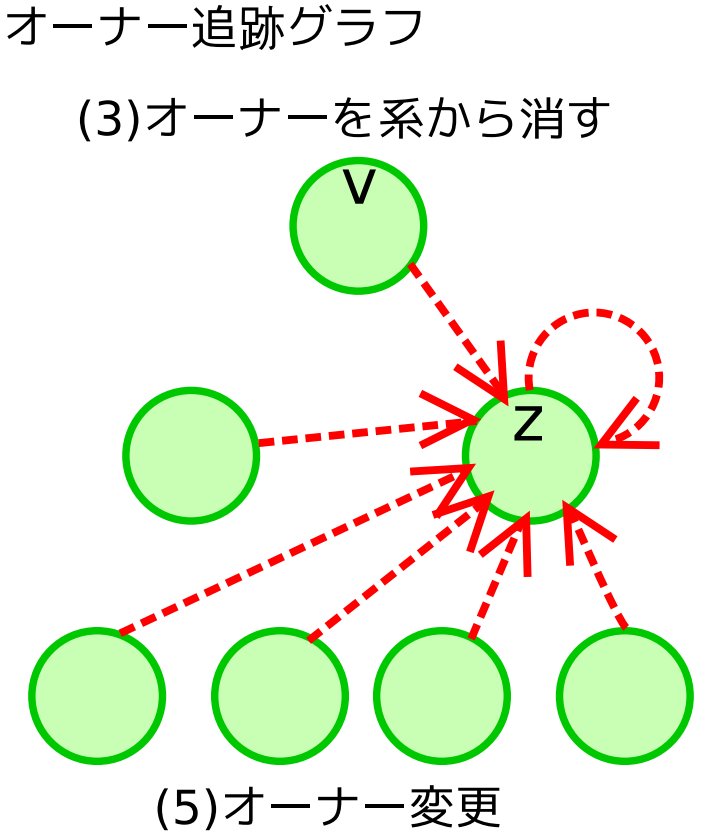
具体例：オーナーの追い出し (1)





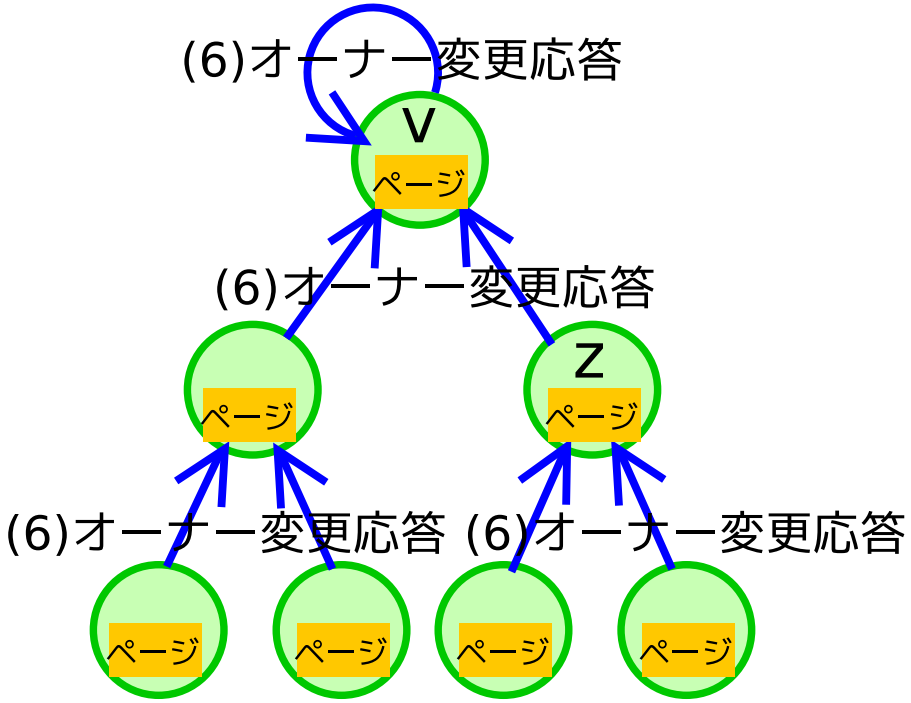
具体例：オーナーの追い出し (2)



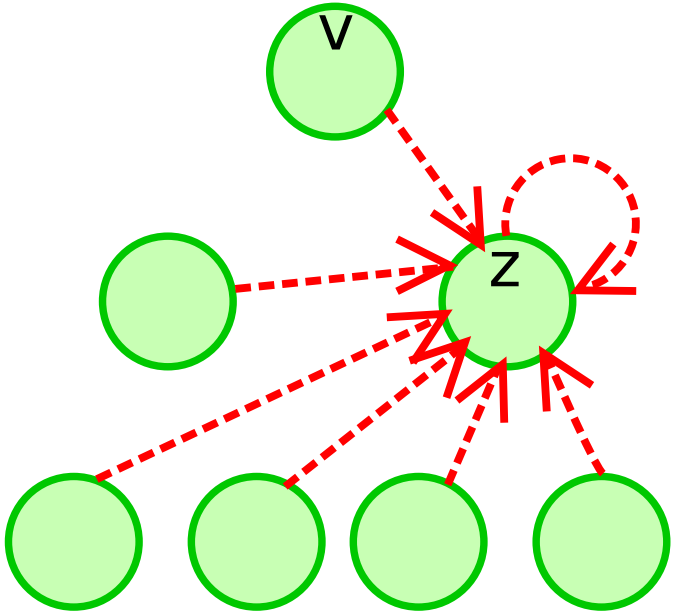


具体例：オーナーの追い出し (3)

ページ転送グラフ

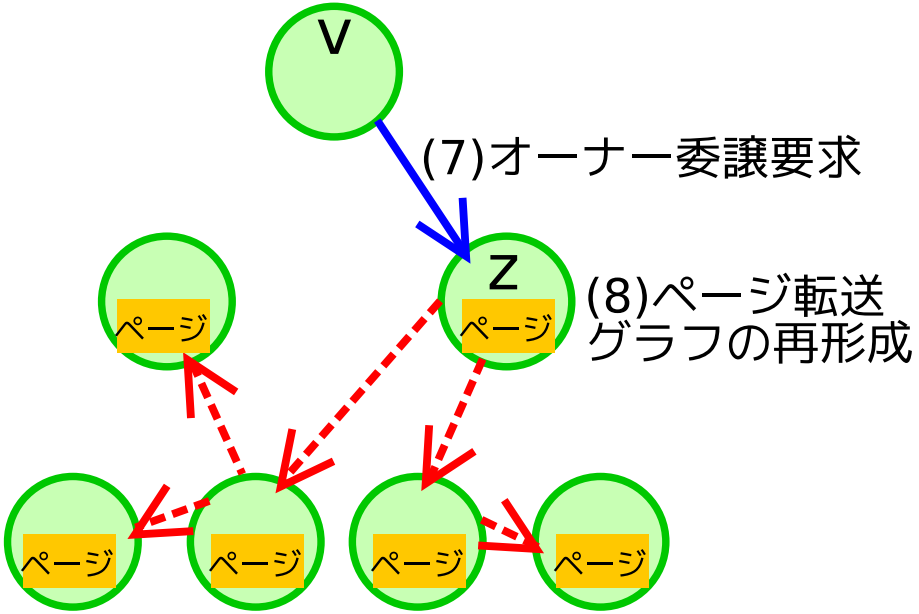


オーナー追跡グラフ

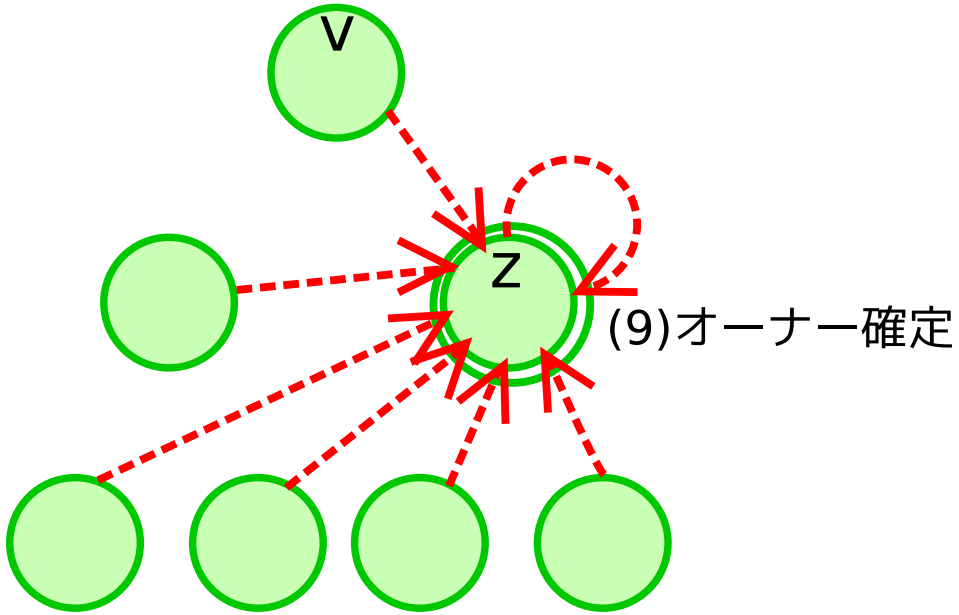


具体例：オーナーの追い出し (4)

ページ転送グラフ

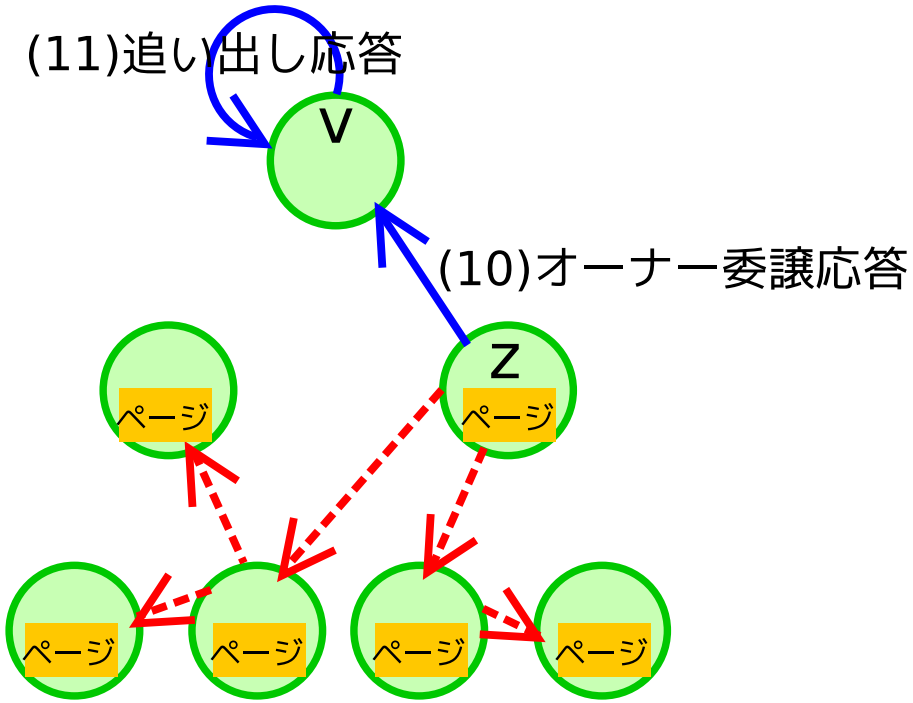


オーナー追跡グラフ

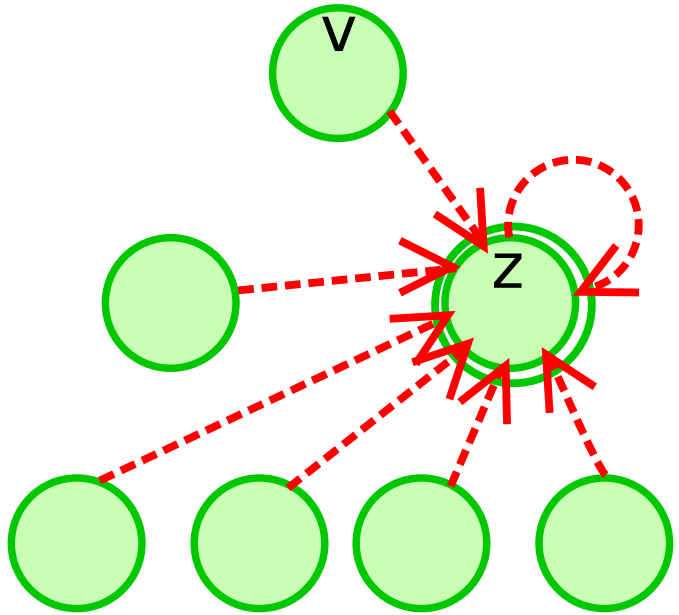


具体例：オーナーの追い出し (5)

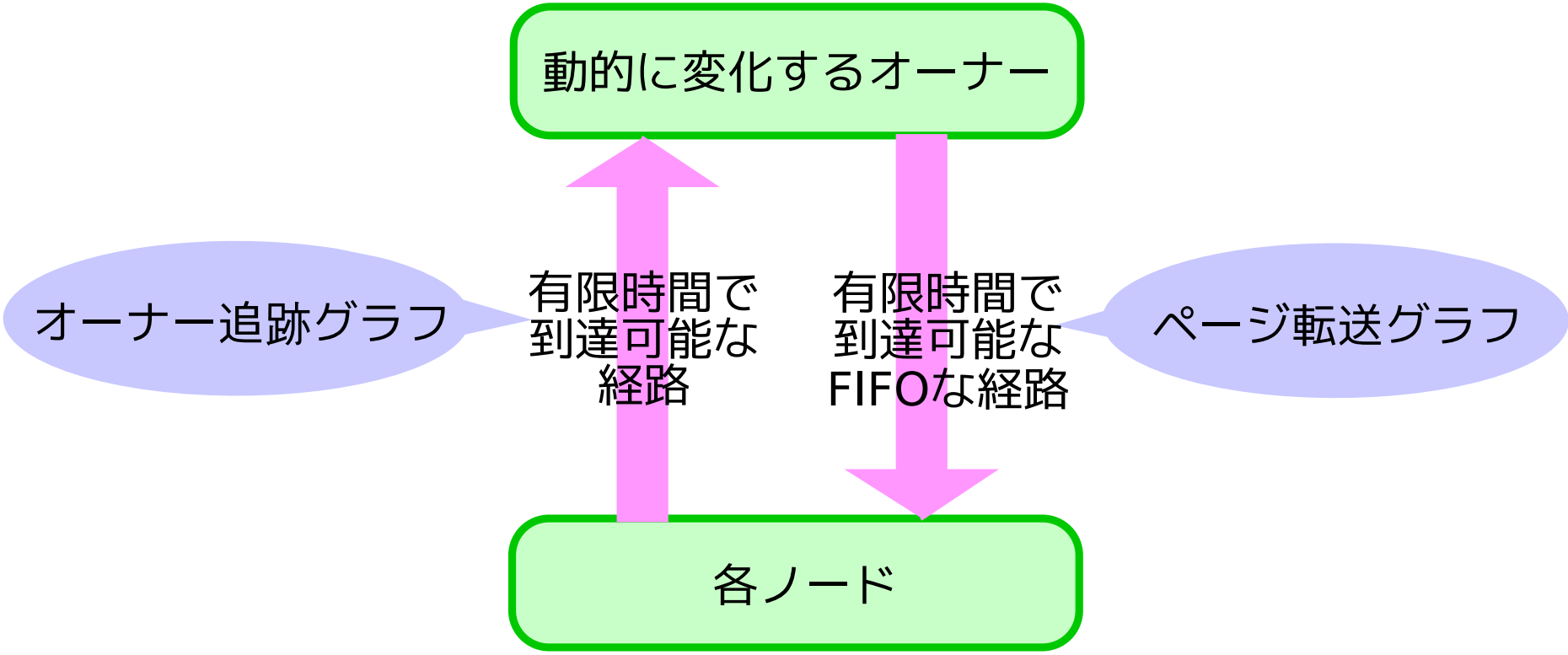
ページ転送グラフ



オーナー追跡グラフ

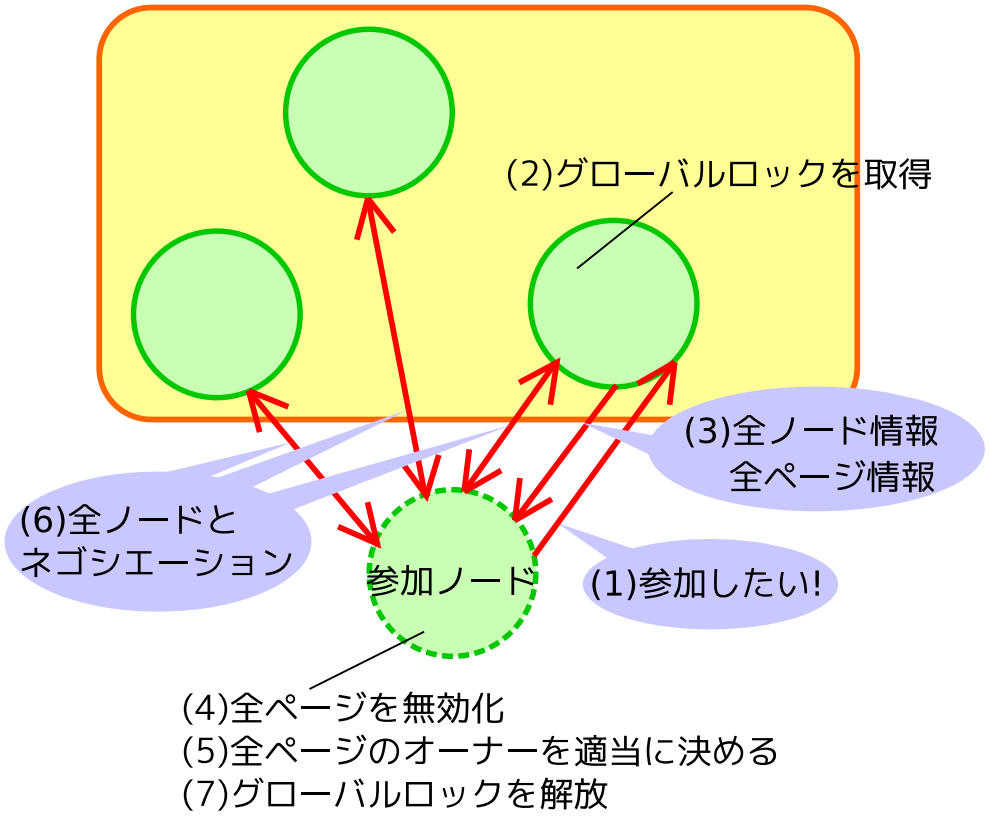


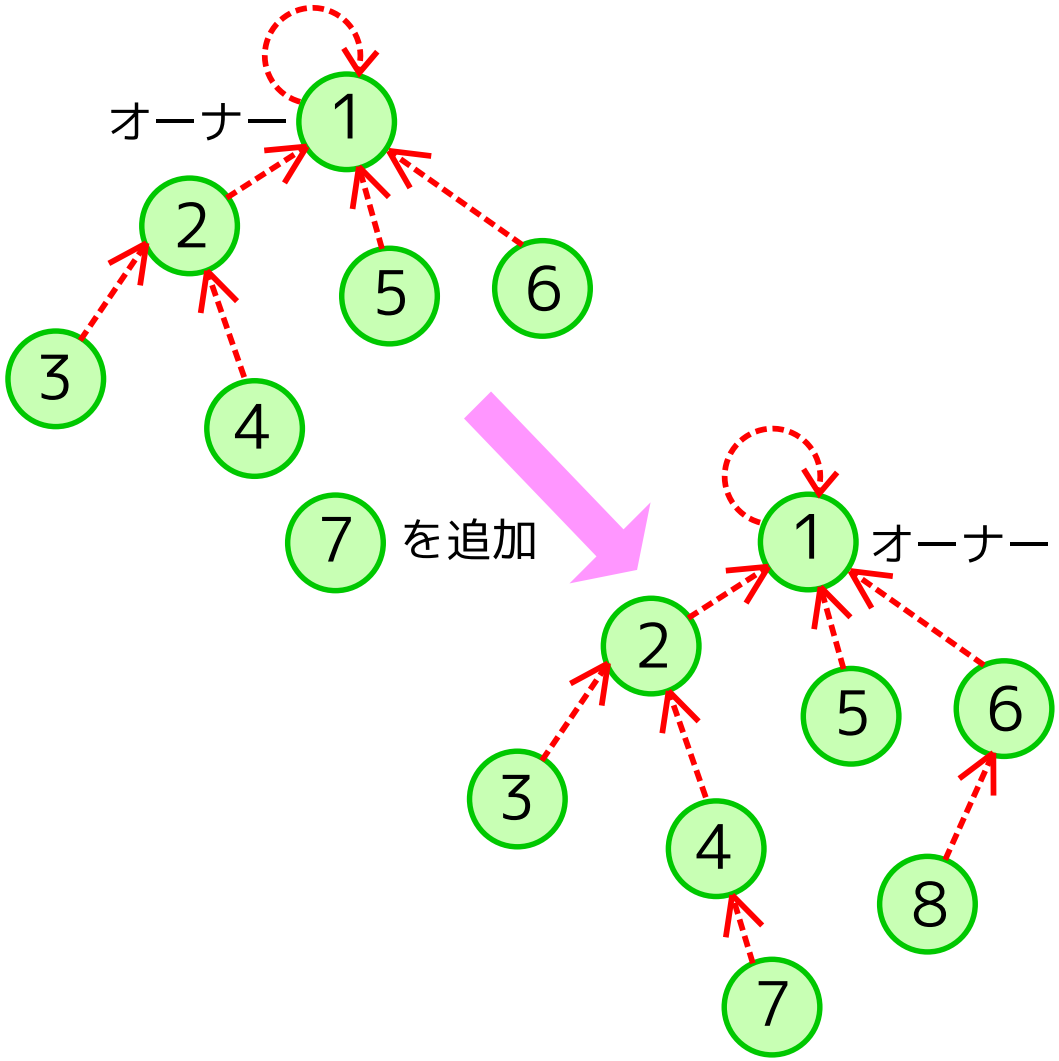
プロトコルの要約



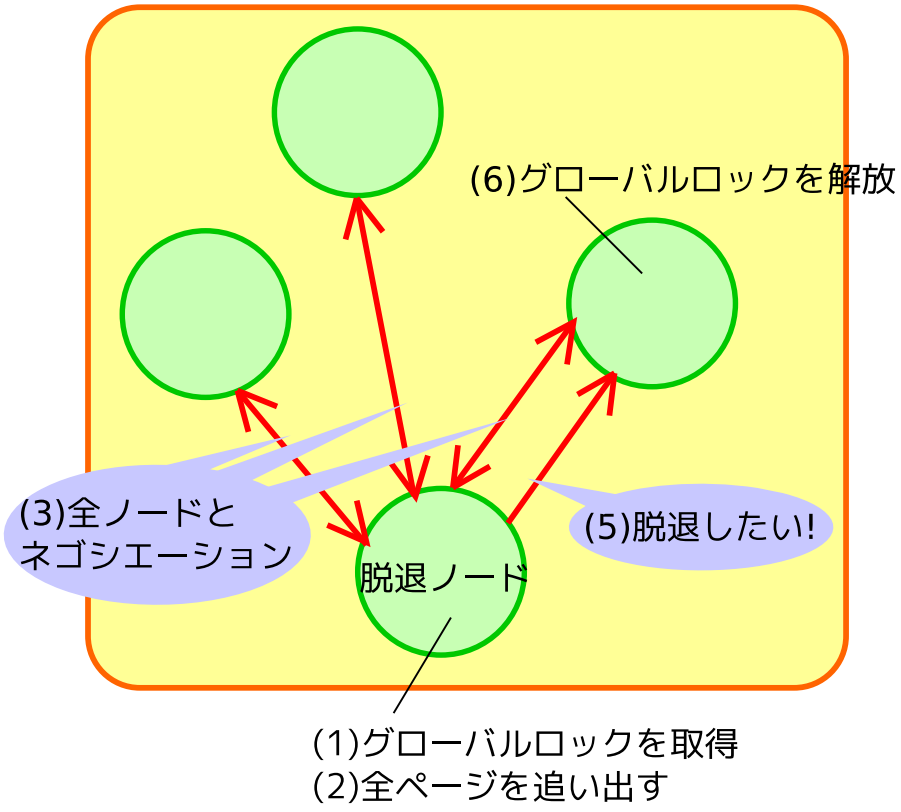
- **単独のオーナーがシリアライズした** read/write **の順序で実際の** read/write **が発生することを保証することで，コンシステンシを保証**
- **数学的に証明済**

ノードの参加

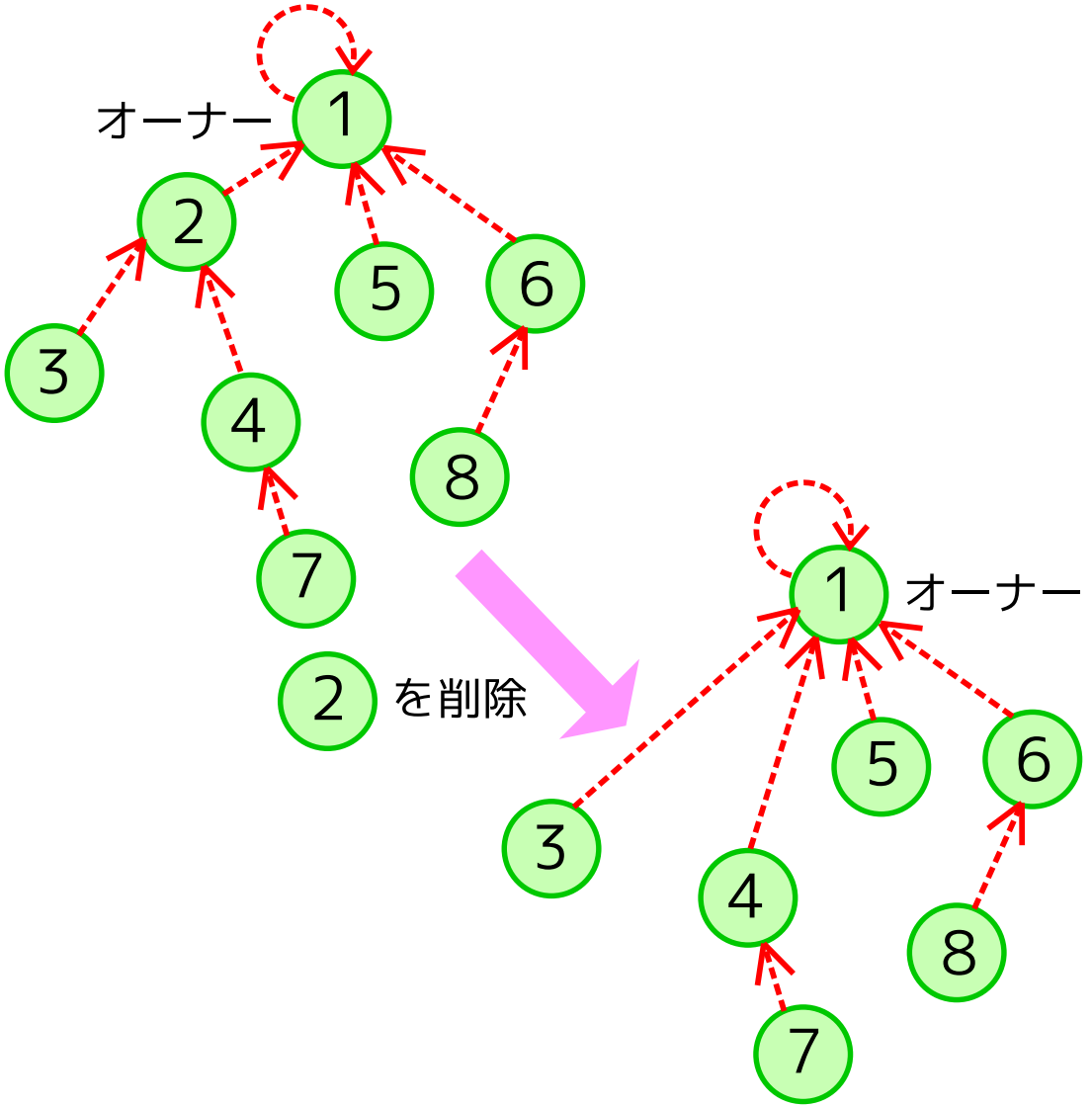

(2)グローバルロックを取得
(3)全ノード情報
全ページ情報
(6)全ノードと
ネゴシエーション
参加ノード
(1)参加したい!
(4)全ページを無効化
(5)全ページのオーナーを適当に決める
(7)グローバルロックを解放


オーナー
1
2
3
4
5
6
7
を追加
1
オーナー
2
3
4
5
6
7
8

ノードの脱退



(4)全ノード上でオーナー追跡グラフの再形成



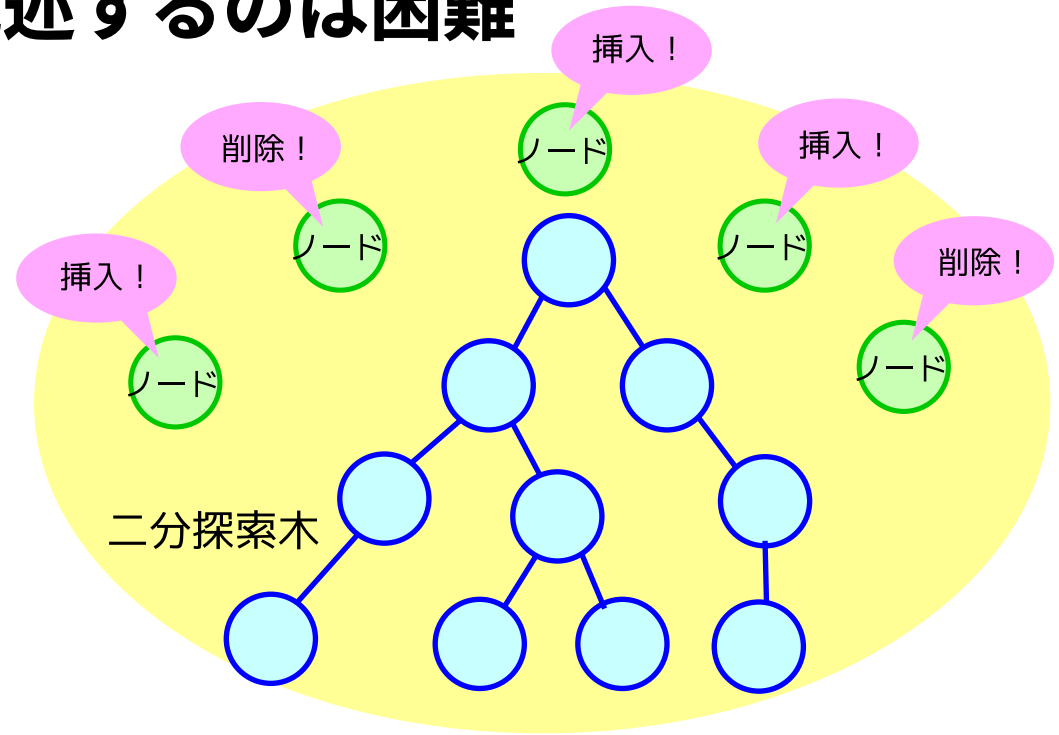
その他の実装

- ページ置換
  - → ページのサイズと状態で優先度付けした置換アルゴリズム
- token[Naimi et al,'96] を用いた同期機構
  - → 排他制御変数，条件変数
- スレッド管理
  - → create，destroy，detach
- 非同期モードのインタフェース
  - → 非同期モードの read，write，malloc，free...
- トランザクション管理
  - → 全インタフェースに対して原子性を保証

4. 評価

二分探索木への並列なデータの挿入/削除 (実験)

- 問題：
  - → 二分探索木に対して，動的に参加/脱退する多数のノードが，適切に排他制御を行いながらデータを挿入/削除
- 種別：共有メモリベースだからこそ記述できるアプリ
  - → 動的で複雑なグラフ構造を扱う処理であり，メッセージパッシングで記述するのは困難

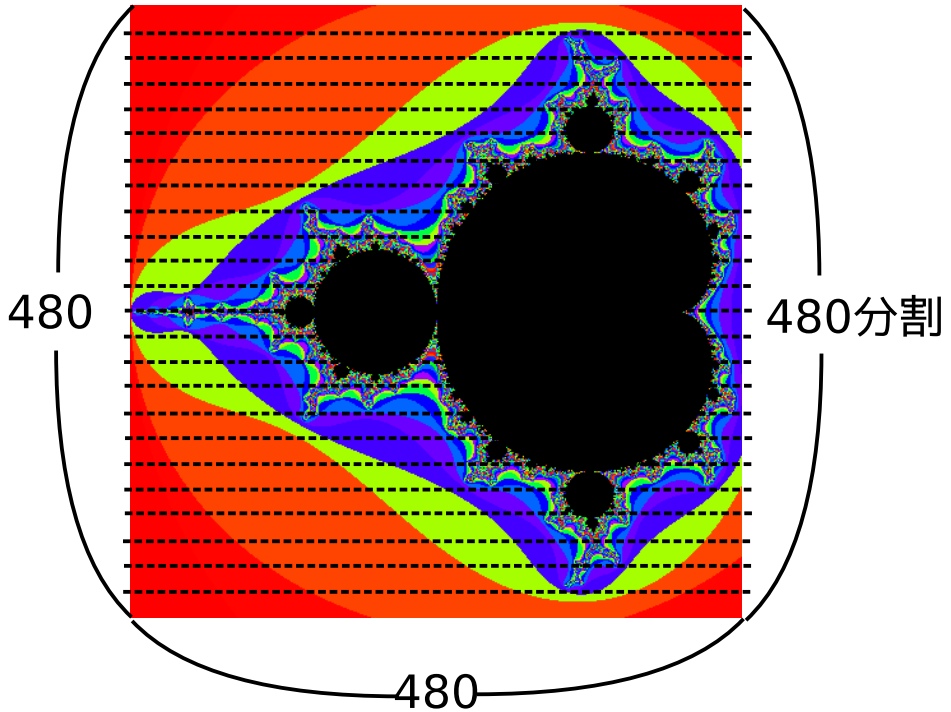


二分探索木への並列なデータの挿入/削除 (結果)

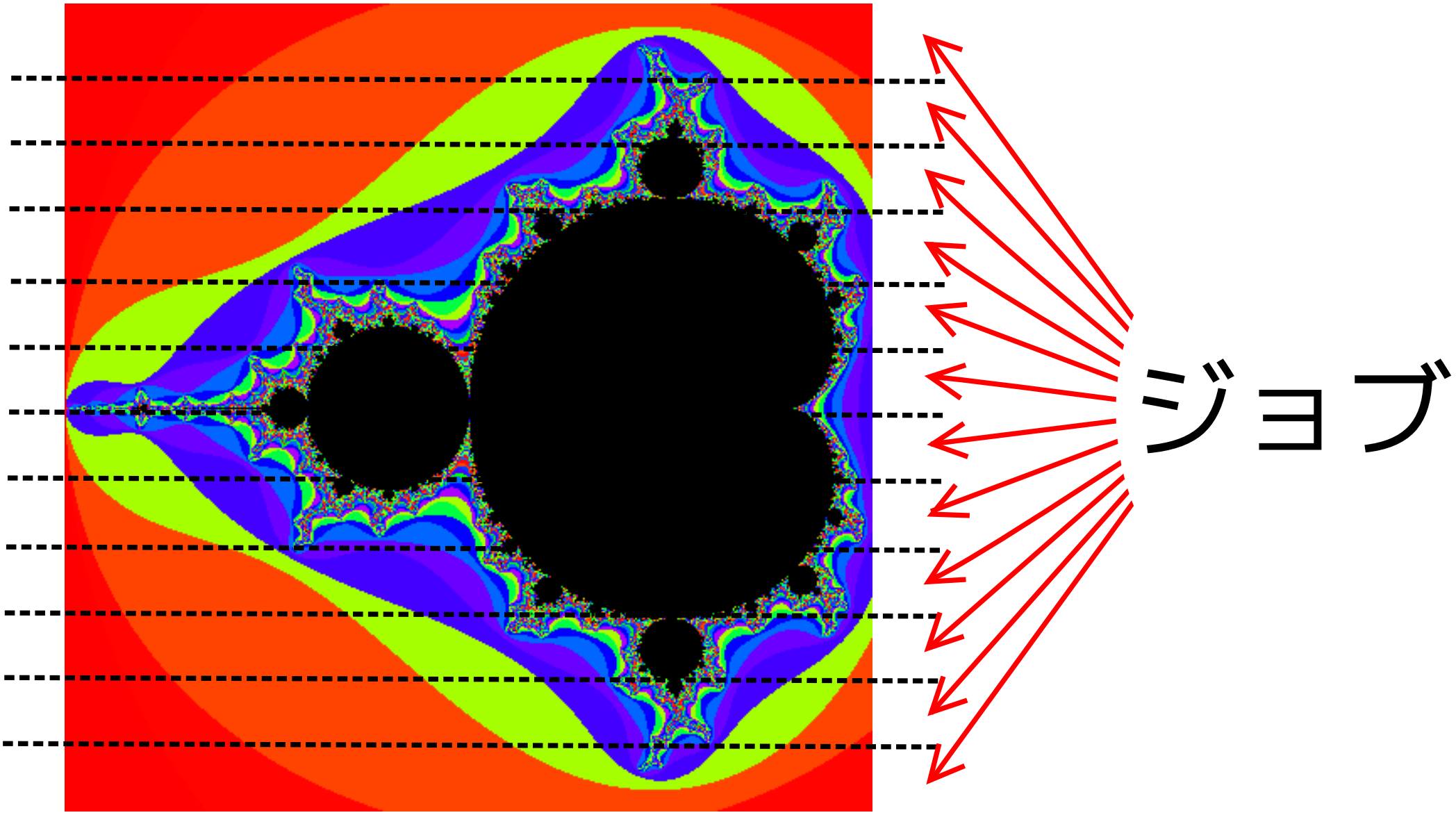
- 結果：
  - → 22 ノード 88 スレッドを動的に参加/脱退させても，二分探索木の中身が正しくソートされた状態で計算が継続されることを確認
  - → pthread プログラムに対するほぼ機械的な翻訳作業でコードが得られることを確認
- 考察：
  - → 従来の処理系では参加/脱退をサポートできなかった，多数の計算資源が密に協調して動作する共有メモリベースのアプリケーション領域に対しても，DMI によるアプローチが応用できる可能性を示唆

マンデルブロ集合の並列描画 (実験)

- **問題：**
  - → $z_0 = 0, z_{n+1} = z_n^2 + c$ **なる複素数列** $\{z_n\}$ **が** $n \to \infty$ **で発散しない** c **の範囲を描画**
- **種別：**Embarassingly Parallel **なアプリ**
  - → 480×480 **の描画領域を横に** 480 **分割したマスタ・ワーカモデルで記述**



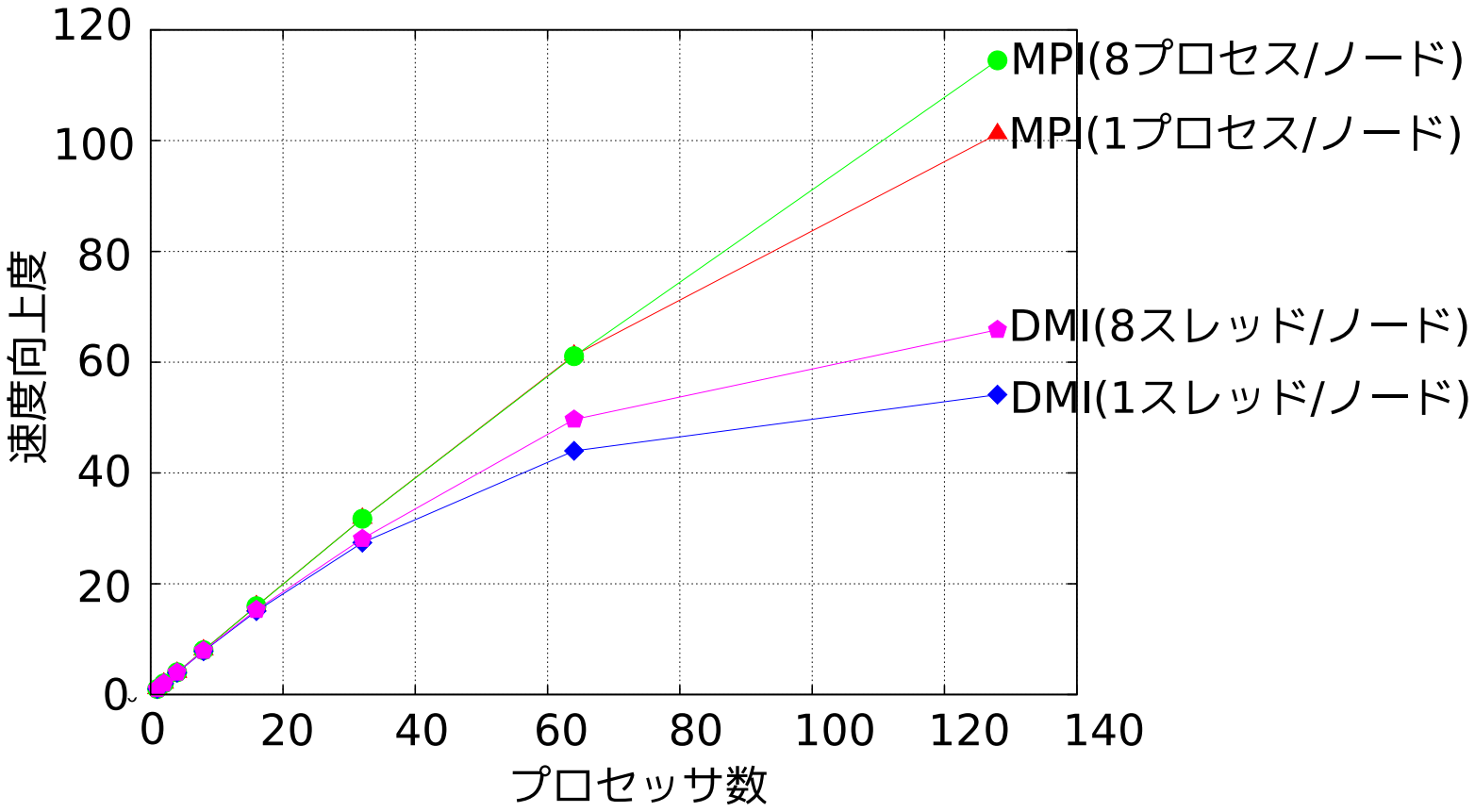
マンデルブロ集合の並列描画 (デモ)



ジョブごとに計算量はまちまち

マンデルブロ集合の並列描画 (結果)

- 結果：



- 考察：
  - → 32 プロセッサ程度までは MPI 並にスケール
  - → マルチコアレベルの並列性を活用できている

5. 結論

まとめ

- DMI：Distributed Memory Interface
  - → 計算資源の動的な参加/脱退をサポートする大規模分散共有メモリ
    - ♦ 参加/脱退対応のコンシステンシプロトコルを開発
  - → pthread プログラムとの類似性を重視
  - → 並列分散ミドルウェア基盤としての，柔軟性，汎用性，機能拡張性に富んだインタフェース設計
- 従来の処理系では計算資源の動的な参加/脱退に対応できなかったアプリケーション領域に対しても，DMI のアプローチが応用できる可能性あり
- Embarassingly Parallel なアプリに対しては台数効果が出ることも確認

今後の課題 (処理系の改良)

- ページ転送の動的負荷分散
- より pthread の実装事情に近い同期機構
- 効率的な malloc アルゴリズム
- 他ノードのメモリ使用状況や参照局所性を考慮したページ置換アルゴリズム
- read/write 粒度での Invalidate 型と Update 型のハイブリッド型プロトコル

今後の課題 (処理系の上に)

- DMI を基盤レイヤーとした並列分散プログラミング言語処理系の開発
  - → 大規模分散共有メモリがあるからこそできること
  - → 計算資源の参加/脱退がサポートされるからこそできること
  - → 最初から分散処理を前提として言語を設計するからこそできること