❖ DMI：ノードの動的な増減に対応した大規模分散共有メモリインターフェース ❖

近山・田浦研究室　原健太朗

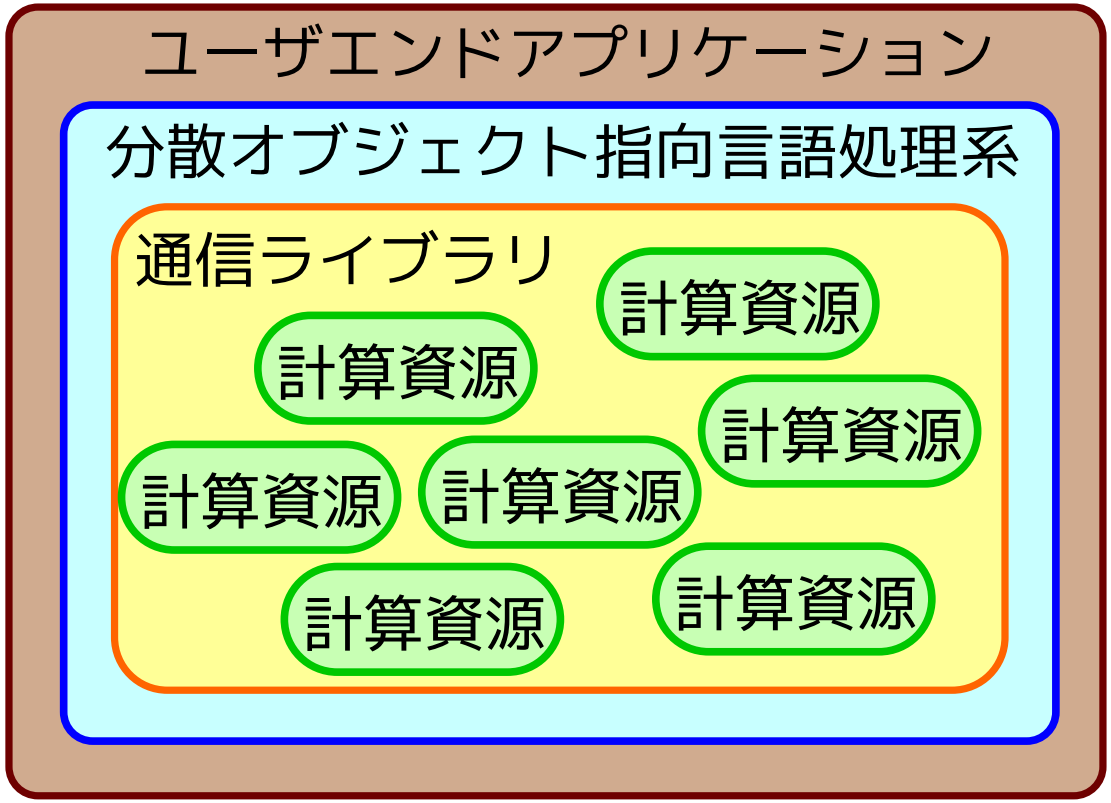
2008.10.2

本研究の背景

- グリッドコンピューティングの発展
  - → 並列分散プログラミングフレームワークが重要
- フレームワークへの要請
  - → 高い記述性
  - → スケーラビリティ
  - → 計算資源の動的な増減への対応

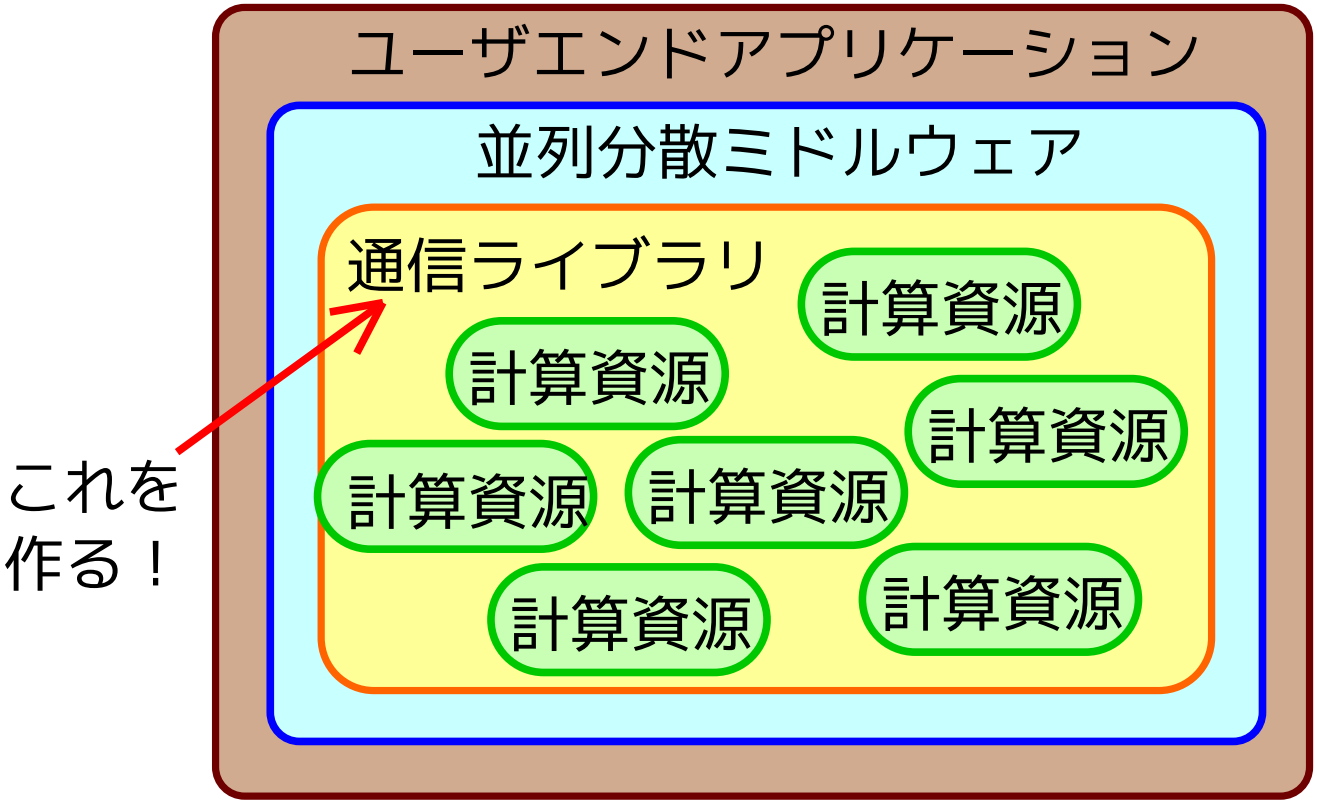
本研究の動機

- 将来的に分散オブジェクト指向言語処理系を開発したい
  - → オブジェクトのマイグレーション
  - → 計算環境のマイグレーション
- 適度な抽象度を持った通信ライブラリが必要

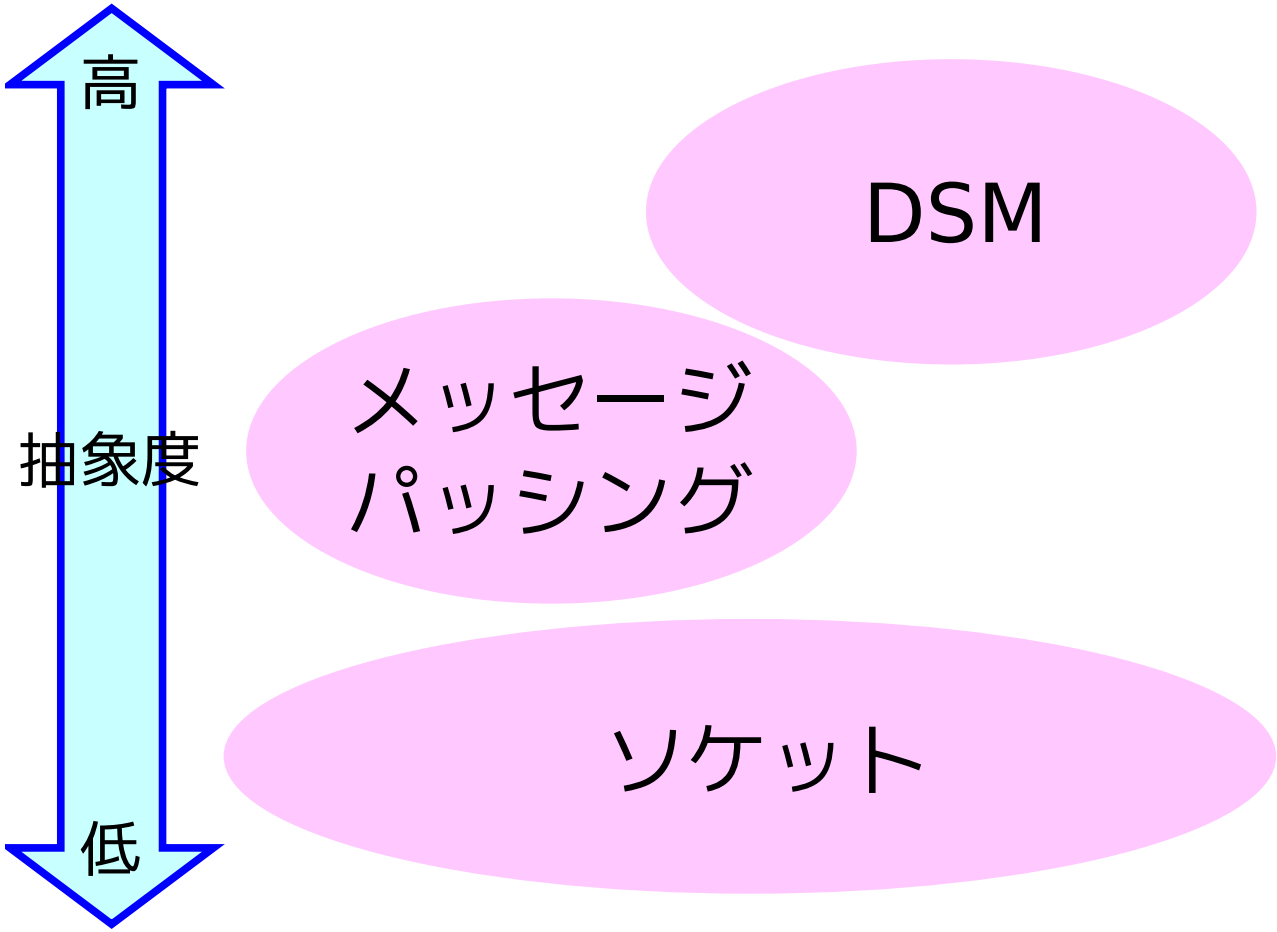


本研究の目的

- 並列分散ミドルウェアの基盤レイヤーとして有用な通信ライブラリの開発

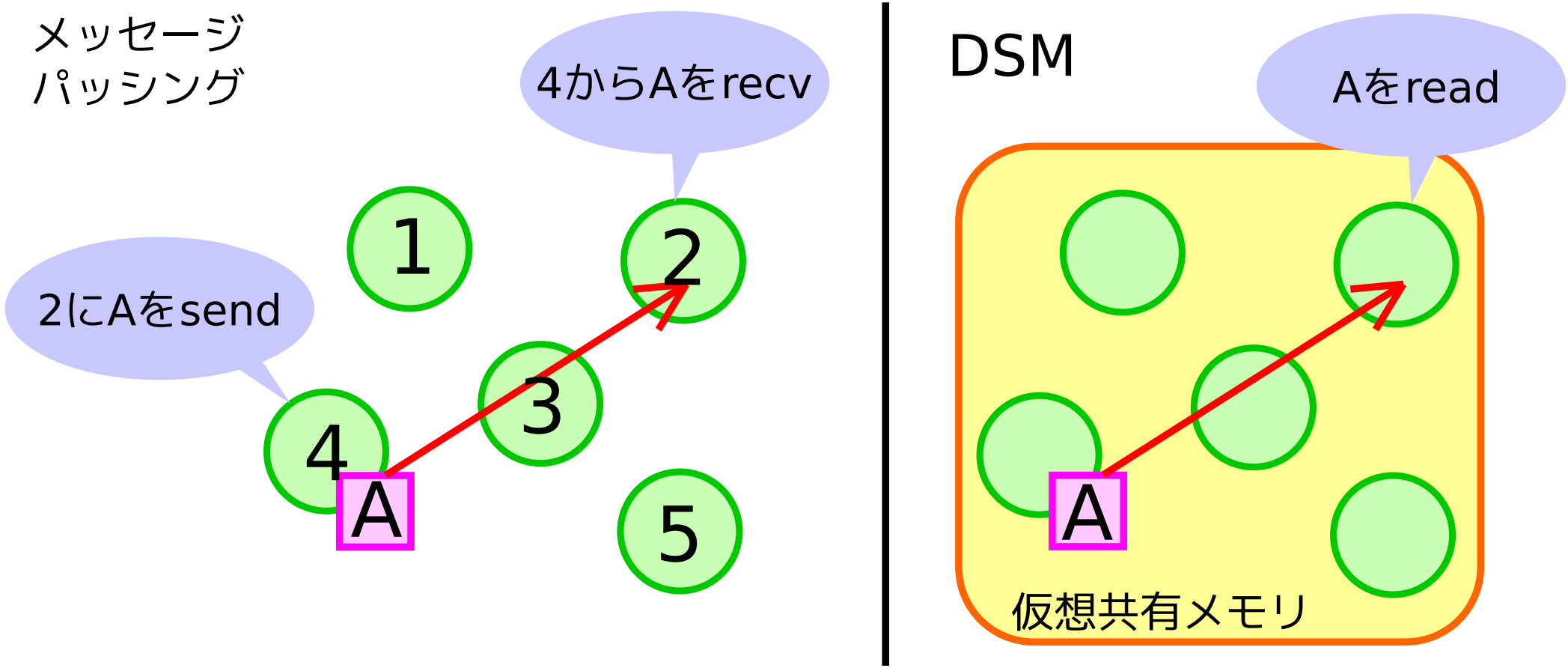


並列分散プログラミングモデル



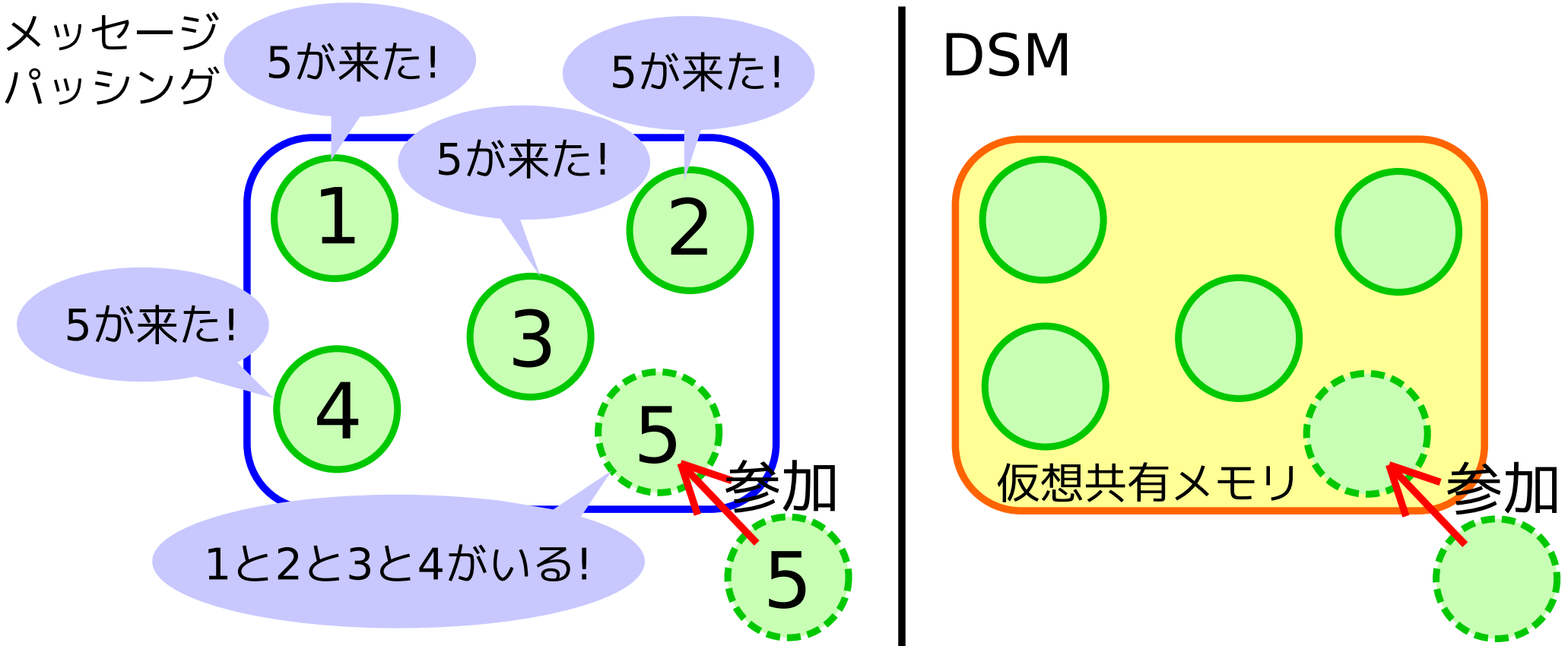
- 本研究では DSM に着眼

DSM の特徴（1）【記述力】



- DSM はプログラム記述が容易かつ論理的に明快
  - → 大規模で複雑なミドルウェアの信頼性，メンテナンス性にとって重要

DSM の特徴(2)【資源の動的な増減への適応力】

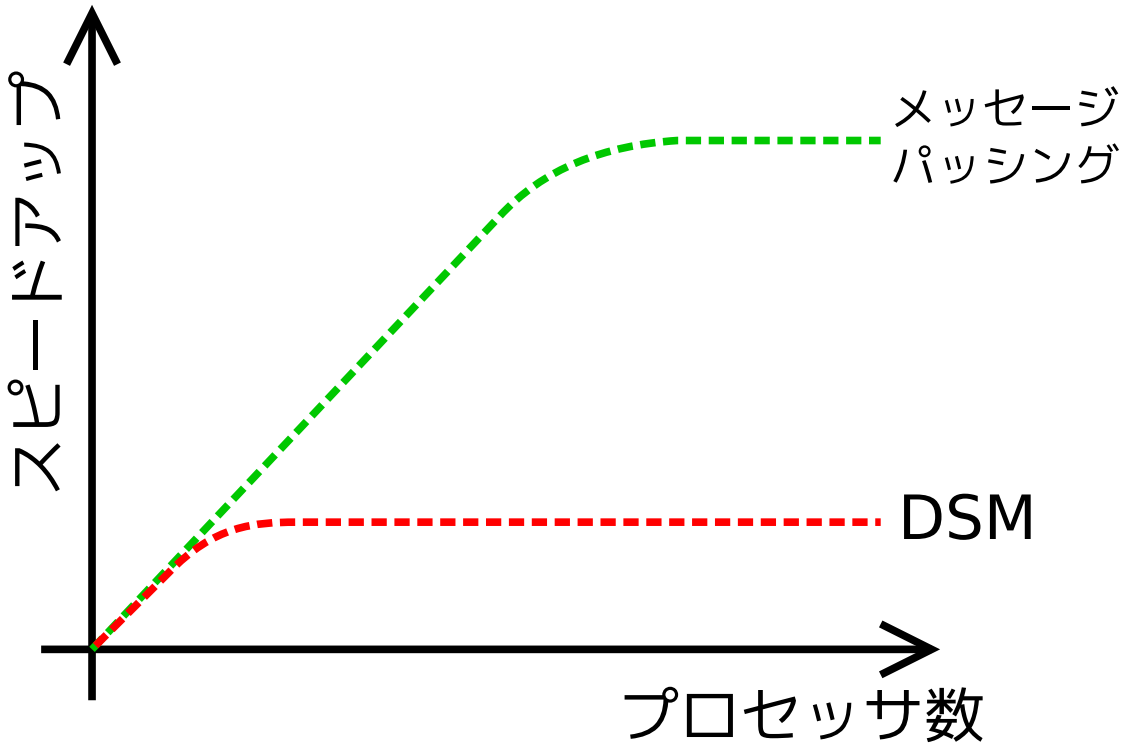


- DSM ではノードの動的な増減を自然に記述可能
- Phoenix[Taura et al,2003]：仮想ノード名空間という概念を導入し，メッセージパッシングで動的な資源の増減を表現可能にしたモデル

DSM の特徴（3）【応用の広さ】

- DSM は幅広い応用力を持つ
  - → ネットワークページング
    - ♦ DLM[Midorikawa et al,2007]：ローカルな swap アクセスと比較して 5~ 10 倍の性能を発揮
  - → プロセスマイグレーション
- DSM は柔軟なミドルウェア開発支援に適す

DSM の特徴（4）【スケーラビリティ】



- メッセージパッシングと比較して DSM はスケールしにくい
  - → DSM の方が抽象度が高いため
  - → DSM では通信パターンを把握できないため

従来の DSM の性能改善へのアプローチ

- 多数の実装例：IVY,Munin,TreadMarks,Midway,SMS,···

Approach1 : コンシステンシモデルの緩和

- → 論理的なわかりやすさが犠牲に
- → 特に，初期的なプログラミングの負担増

Approach2 : 高度で複雑な暗黙的機構による通信量の低減

- → データの差分転送，更新の通知タイミング，···
- → 明示的なチューニングが困難に

Approach3 : OS のメモリ保護機構を利用

- → コンシステンシ維持の単位は OS のページサイズ (の整数倍) に限定
- → ネットワークページングは 64bit OS が前提

本研究の提案

- DMI（Distributed Memory Interface）
  - → 大規模分散共有メモリインターフェース

DMI のコンセプトと利点

Concept1 : ノードの動的な増減に対応

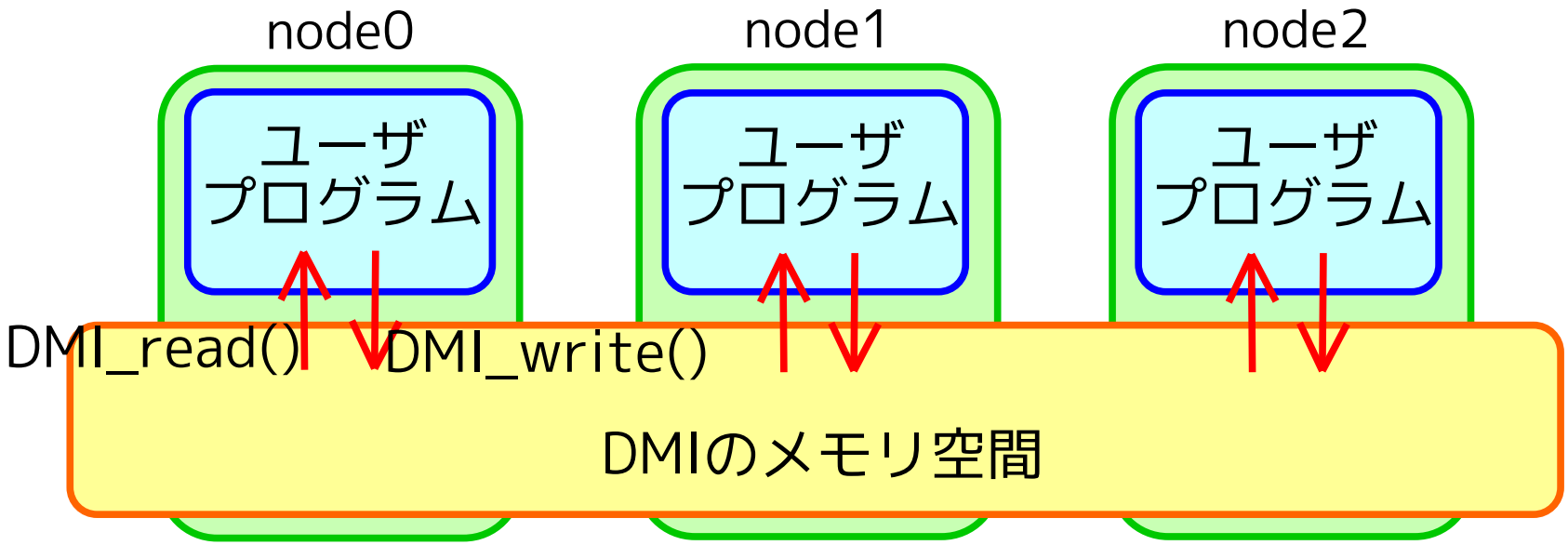
Concept2 : Sequential Consistency を採用

- ➔ 論理的にわかりやすい

Concept3 : メモリ管理機構をユーザレベルで実装

- ➔ 任意のページサイズが指定可能
  - ♦ ページフォルトを大幅に抑制可能
- ➔ OS のアドレッシング範囲を意識しないネットワークページング
- ➔ 非同期 read/write など柔軟なセマンティクスを提供
  - ♦ 性能チューニングの自由度が高い
  - ♦ インクリメンタルな開発が可能
- ➔ 機能拡張が自由

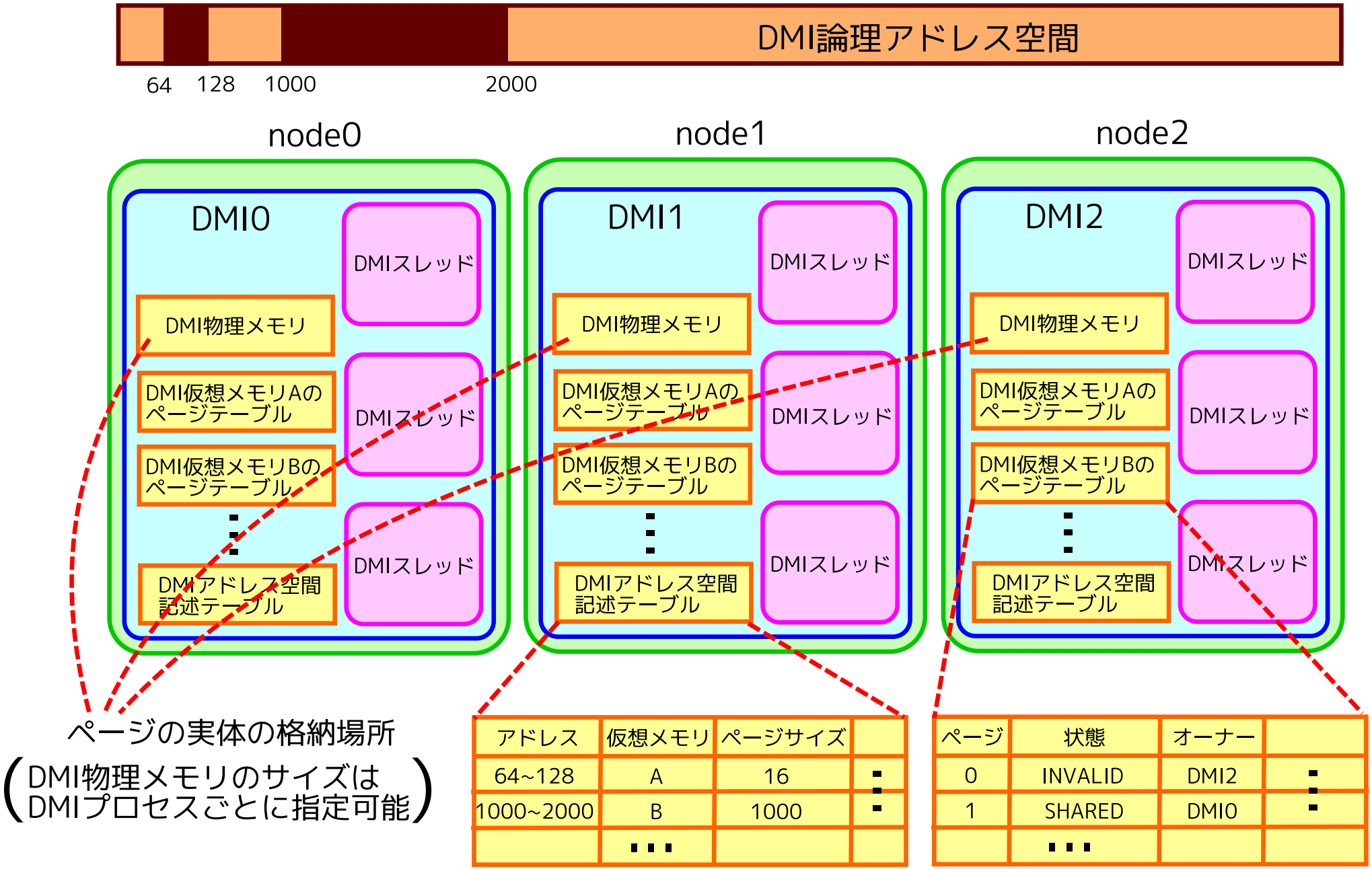
DMI とユーザプログラムの関係モデル




```
DMI_read(address, size, buffer);
DMI_write(address, size, buffer);
```

- ユーザプログラムのメモリ空間と DMI のメモリ空間は独立
  - → API を通じてのみアクセス可能

DMI のメモリ管理モデル



従来の DSM に対する DMI の欠点

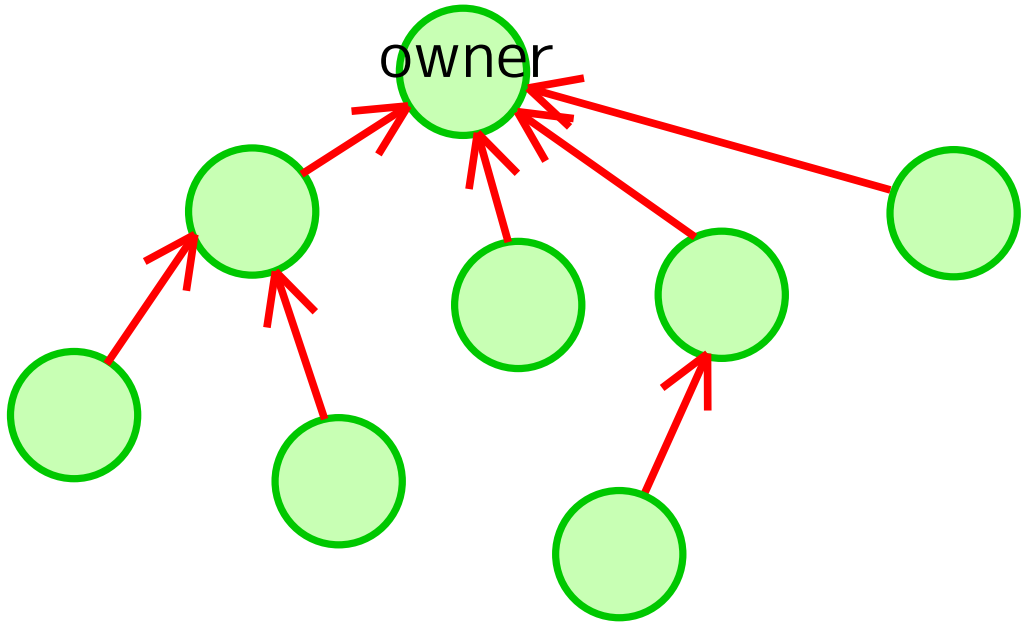
- プログラミングが作業的に面倒
  - → 論理的には容易
- アクセス時のオーバーヘッドが大きい
  - → 逐一ソフトウェア的な検査が入る
  - → ユーザプログラムのメモリ空間と DMI のメモリ空間の間でメモリコピーが発生

DMI におけるコンシステンシ管理（1）

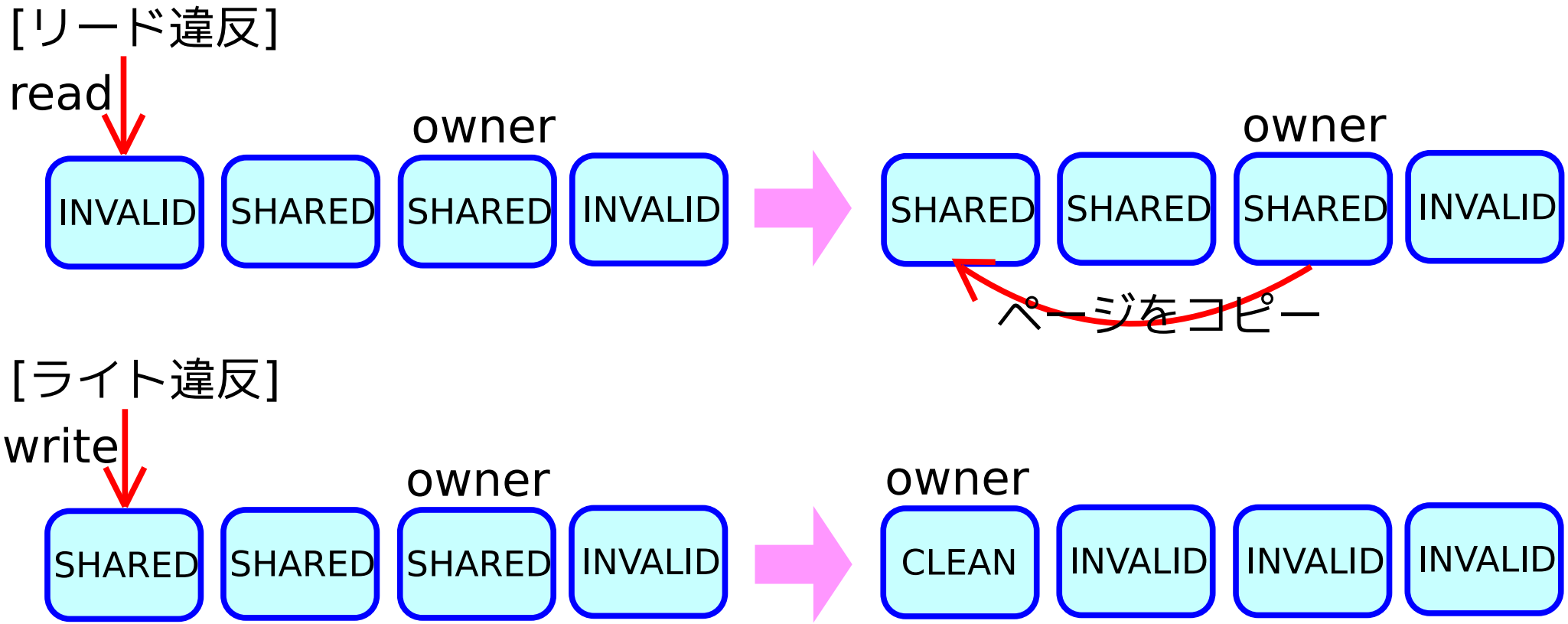
- Sequential Consistency
- Single Writer 型，Write Invalidation 型
- ページ単位で独立に管理
  - → 複数ページへの要求を並列に処理可能
- ページに関して管理する情報
  - → ページの状態
    - ♦ CLEAN(read：可,write：可)
    - ♦ SHARED(read：可,write：不可)
    - ♦ INVALID(read：不可,write：不可)
  - → オーナーの位置

DMI におけるコンシステンシ管理（2）

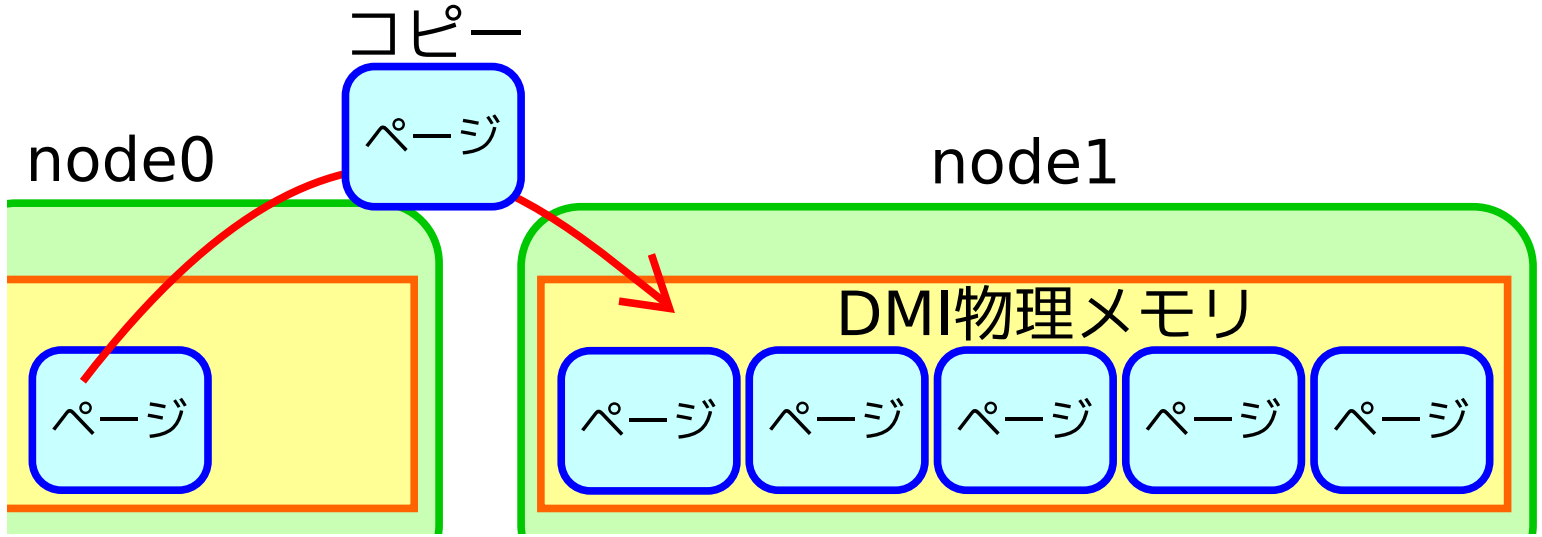
- オーナーの位置を常に把握するホームを設置しない
- オーナー追跡グラフを形成
  - → オーナーの参照関係を辿ることで真のオーナーに到達可能
  - → ページフォルト時にはリクエストをフォワーディング
- Li らのアルゴリズム [Li et al,1989] を改善した上で実装



DMI におけるコンシステンシ管理（3）

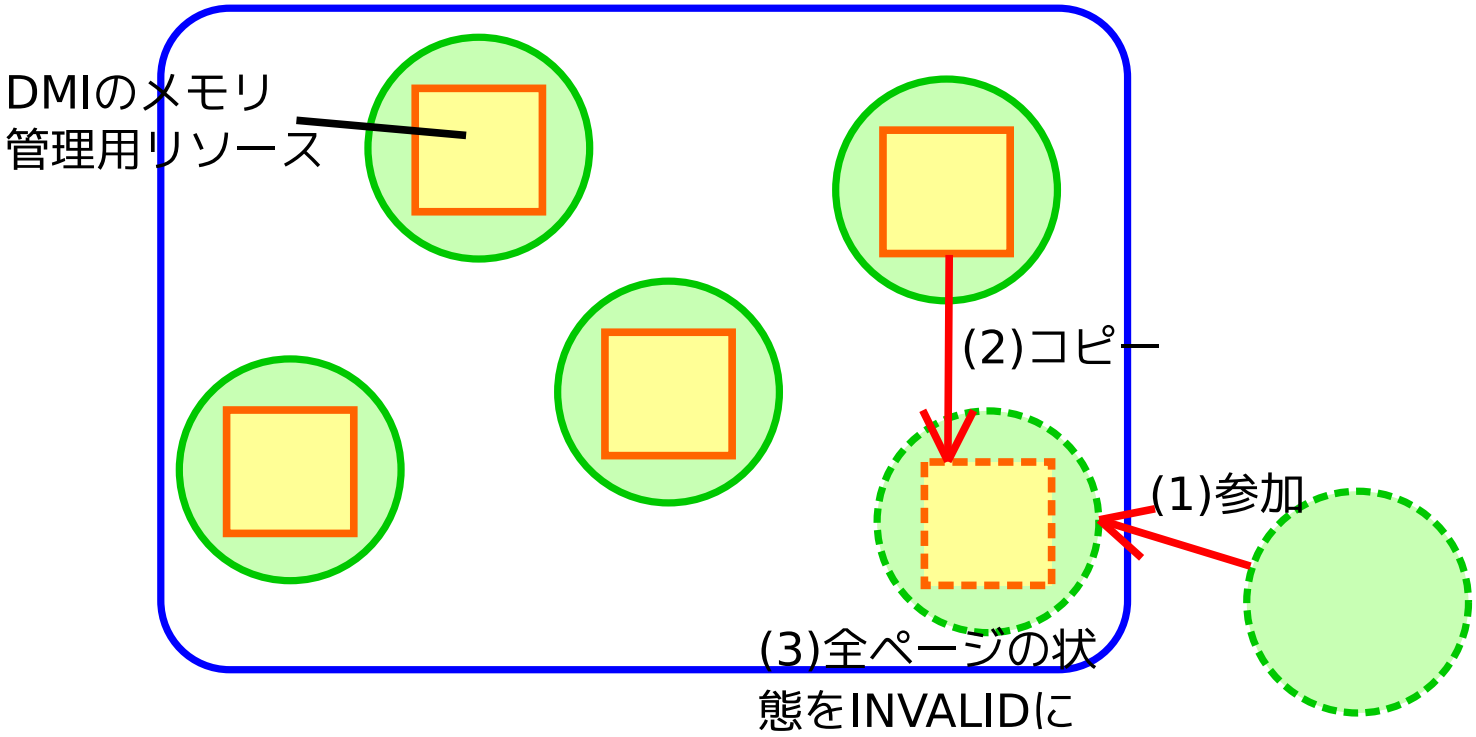


DMI におけるページ置換

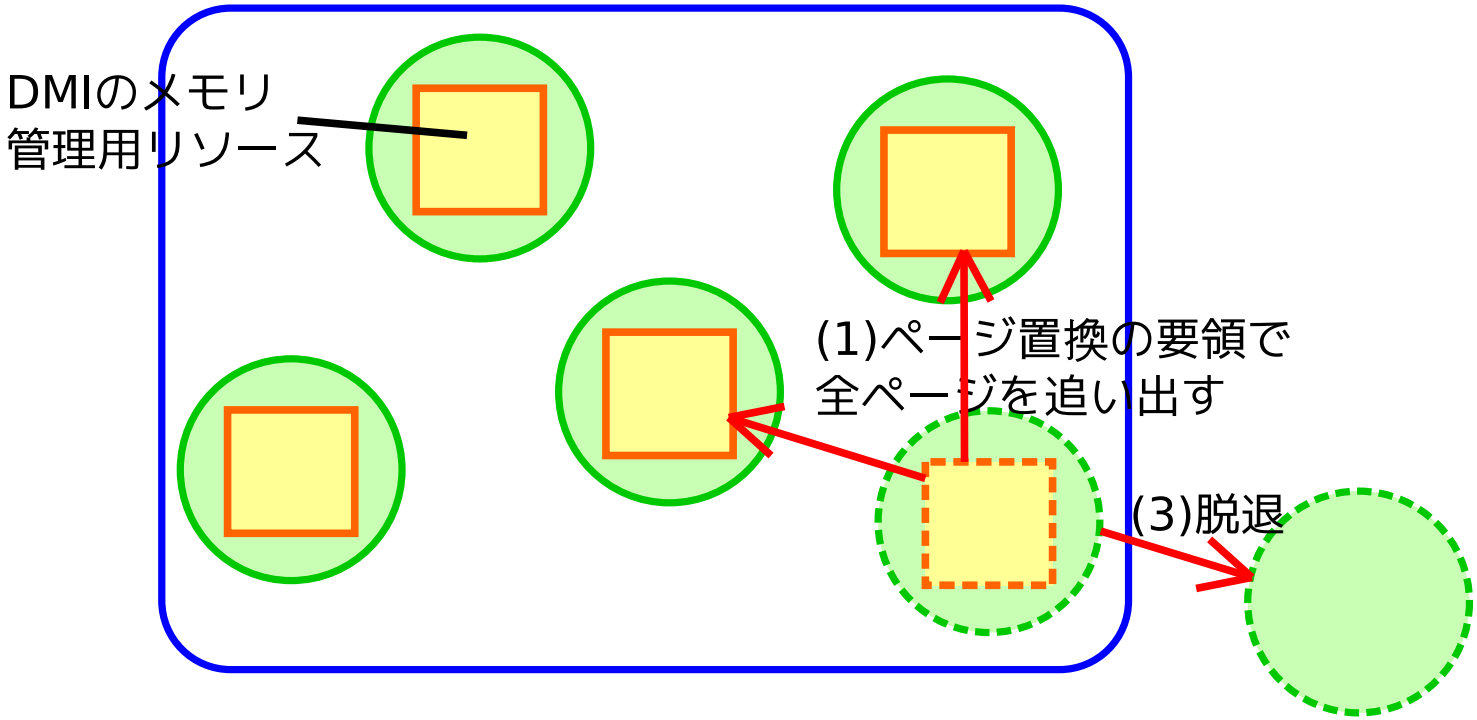


- INVALID に変化したページはすぐに解放
- それでも空き領域がない場合，
  - → オーナー権を伴わない SHARED，オーナー権を伴う SHARED，CLEAN の優先度順に追い出す [Sinha,1996]
  - → 追い出し先のノードを決定し，そのノードにライト違反要求を起こすよう指示

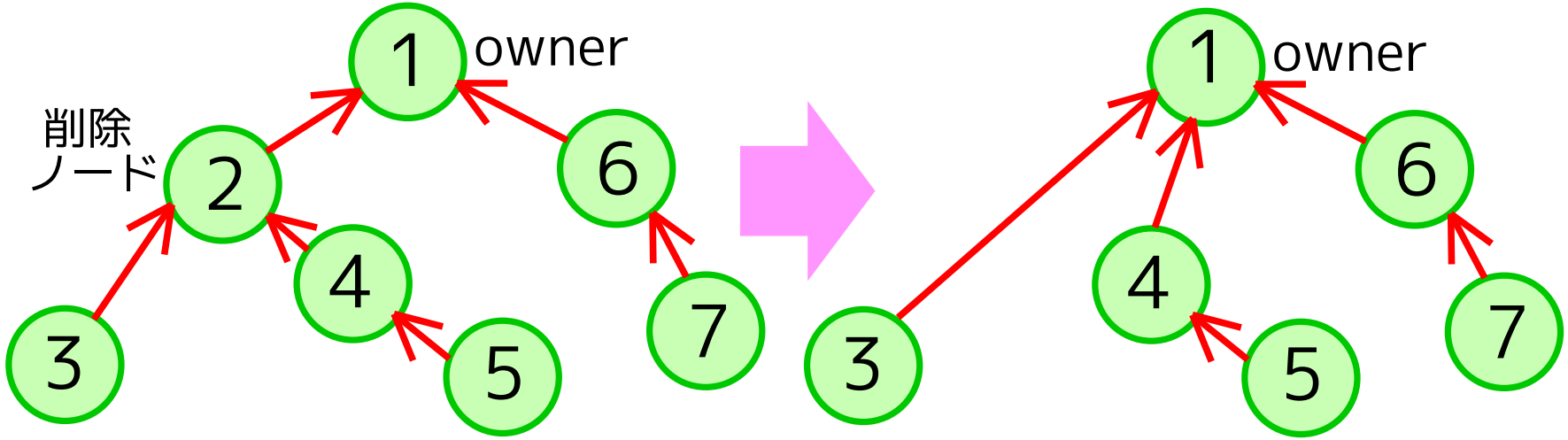
DMI におけるノードの参加



DMI におけるノードの脱退



(2)オーナー追跡グラフの再形成

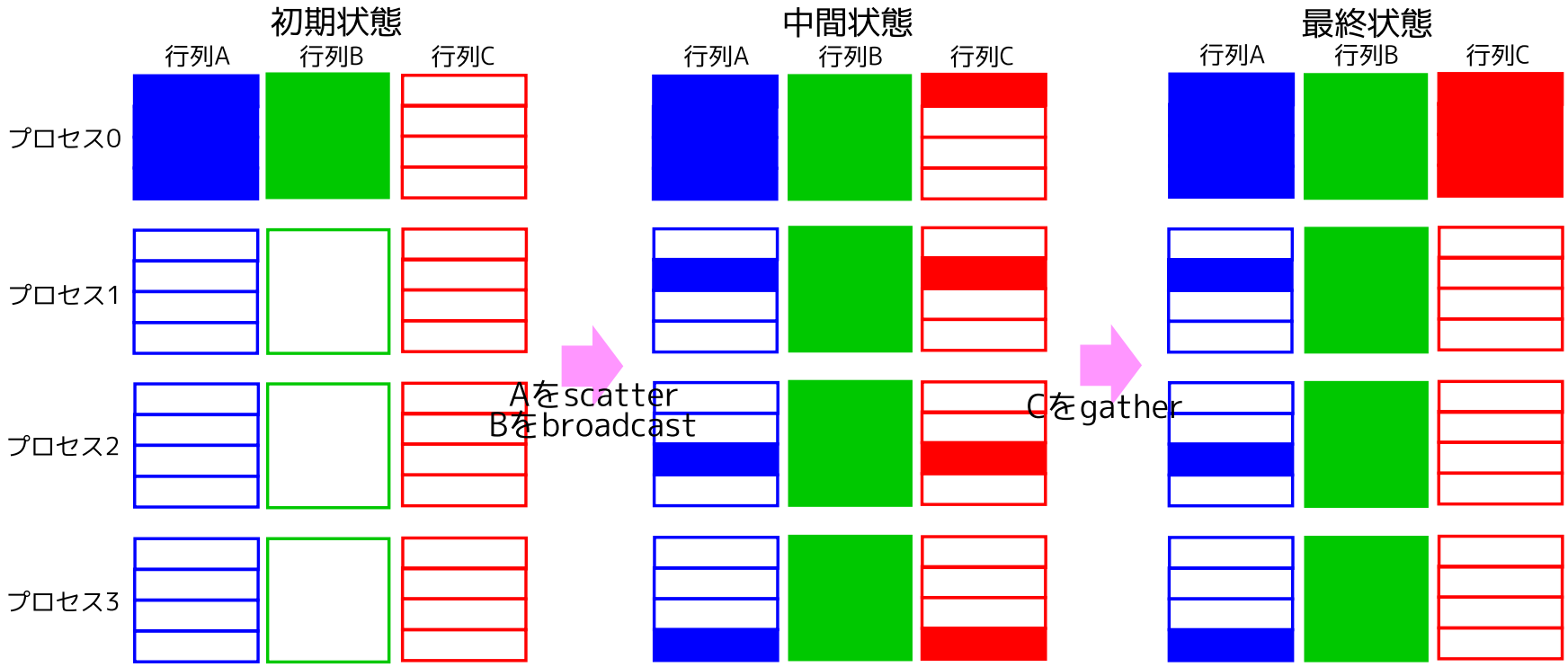


DMI の API

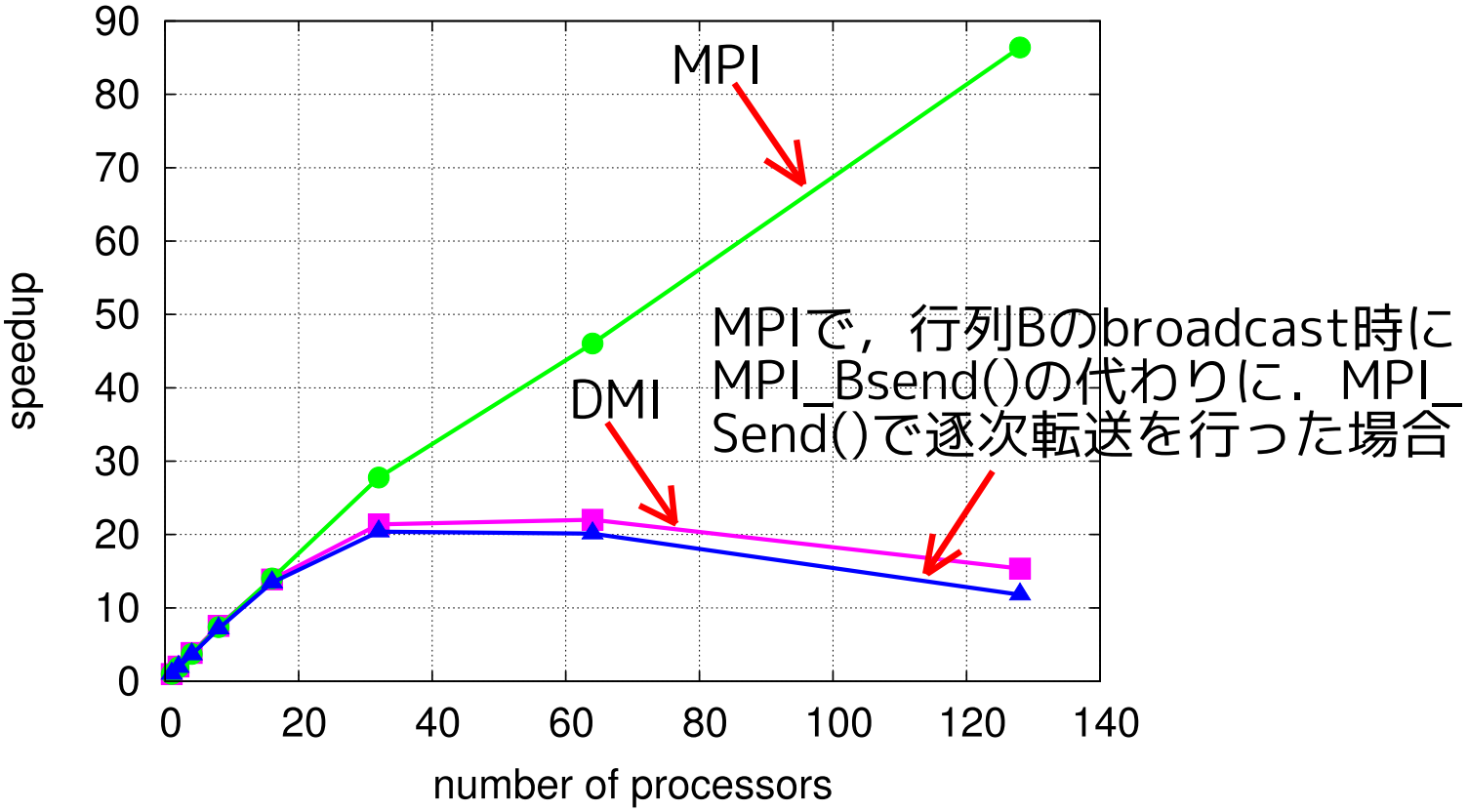
- 提供予定の API
  - → 初期化/終了
  - → メモリ確保/解放
  - → DMI スレッドの参加/脱退
  - → 通常の read/write
  - → 非同期 read/write
  - → ロック機構
  - → 条件変数
  - → メモリフェンス
- その他，ニーズに応じた機能拡張が自由

予備的性能評価（1）

- 現状：ノードの動的な増減，一部の API，ページ置換は未実装
- 実験：2048×2048 の行列行列積 $AB = C$ を MPI と DMI で性能比較
  - → 各ページに対して 1 回しかページフォルトが起きない



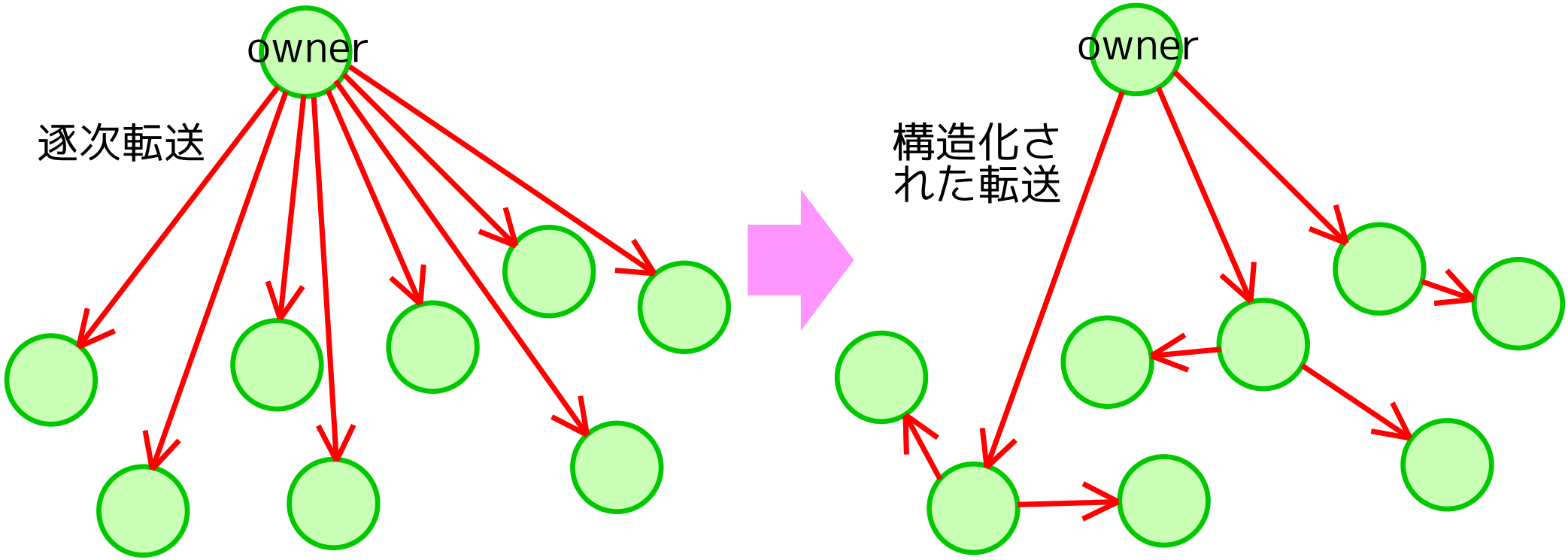
予備的性能評価（2）



- DMI は 30 台弱までしかスケールせず
- 性能劣化の原因のほぼ全てが，行列 B に関するページ転送方式に起因

ページ転送の動的負荷分散

- ページ転送処理を構造化
  - → 各ノードが自分に届いたページ要求の一部を，すでにページ転送したノードにフォワーディング



まとめ

- DMI
  - 提案手法
    - メモリ管理機構をユーザレベルで実装
    - ノードの動的な増減を実現
    - ページ転送の動的負荷分散
  - 高い記述性，柔軟性，拡張性
  - 並列分散ミドルウェアの基盤レイヤーとしての応用を期待
- 今後の予定
  - 以上で述べた機能の実装
  - NAS Parallel Benchmark などで性能評価