大規模マルチメディアを対象としたスクラップアンドビルド型DBMSの構築

田中俊顯† 中家路也†† 松田一章††
家富誠敏††† 富井尚志†††† 有澤博††††

† 横浜国立大学大学院 環境情報学府 情報メディア環境学専攻
†† 横浜国立大学大学院 工学研究科 電子情報工学専攻
††† 横浜国立大学 エコテクノロジーシステムラボラトリー
†††† 横浜国立大学大学院 環境情報研究院

発表の流れ

■DBの運用方針・蓄積されるデータ量に応じた拡張を可能とするDBMSとして、我々が提案している

スクラップアンドビルド型DBMSの概要について

■このシステムを実現する上で重要な役割を担う

共有メモリの排他制御機構

•新たな共有メモリの管理方式の提案と評価

スクラップアンドビルド型DBMSの概要について

共有メモリの排他制御機構

研究の背景

マルチメディアDBの、医療・生産技術分野などへの応用が期待されている。

例えば、生産技術分野の応用として、作業者の負荷の軽減や、作業効率の向上を図るために、数百・数千人の工場作業者が行う、数千・数万にも及ぶ作業工程に関する情報を図面・映像・管理データといった形でDBに蓄積しようとする場合など

DBMSに対する要求

大規模で長期間の運用を想定し、高い信頼性が求められるDBでは

- ユーザのニーズに合わせたDBMS機能の変更
- 運用方針に合わせたスキーマの変更
- データ量に合わせたハードウェア構成の変更

これらのオンライン拡張性が求められる。

関連する研究

- ●DBMSの機能の変更・・・ COMMON[1]、AXIS[2] 、Earth[3]
- ●ハードウェア構成の変更・・・（AXIS[2] ）
- ●スキーマの変更・・・[4]

上記の各項目ごとに、様々な研究が行われている。

[1]増永浩二、宝珍輝尚、都司達夫：”拡張可能DBMSにおける部品の管理と呼び出しの一方法”, 情報処理学会論文誌, Vol.40, No.SIG6(TOD3), pp.152-161, 1999.

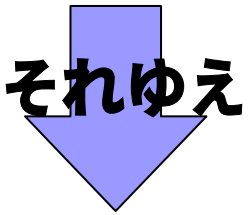
[2]Kojima, I., Tanuma, H., Sato, Y., Ebihara, I. and Okada, Y.: Design and Implementation of an Object-Oriented Database System Using an Extensible Database Toolkit, *Proc of 2nd Int'l Computer Science Conf., pp.588-594, 1992*

[3]早田宏、渡辺美樹、田中圭、山崎伸宏：“オブジェクト指向データベース・エンジンErath:キャッシュ共有のための設計空間”, 情報処理学会論文誌, Vol.39, No.7, pp.2240-2249, 1998.

[4]鬼塚真、山室雅司、石垣昭一郎：“オブジェクト進化を実現するクラスベースのオブジェクト指向データベース設計法”,電子情報通信学会論文誌, D-I, Vol.10 pp.803-810, 1996

関連研究の問題点

しかし、これらの研究では、DBMSに求められる3つの変更を、それぞれ別のものと考えている。



1項目（又は2項目）については要求を満たしているものはあっても、 すべての要求を満たしているものはないという問題点がある。

我々の提案

そこで我々は、システムを抜本的に見直し、これらすべての変更要求を満たすような、まったく新しいDBMSを構築することを考えた。

これが我々の提案する

スクラップアンドビルド型DBMS

の概念である。

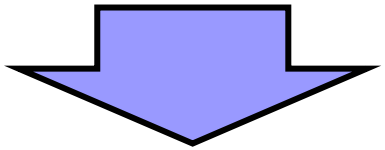
スクラップアンドビルド型DBMSの実現

すべての変更を可能にするDBMSを実現するために

スクラップアンドビルド型DBMSでは、マルチ・エージェントシステムを採用した。

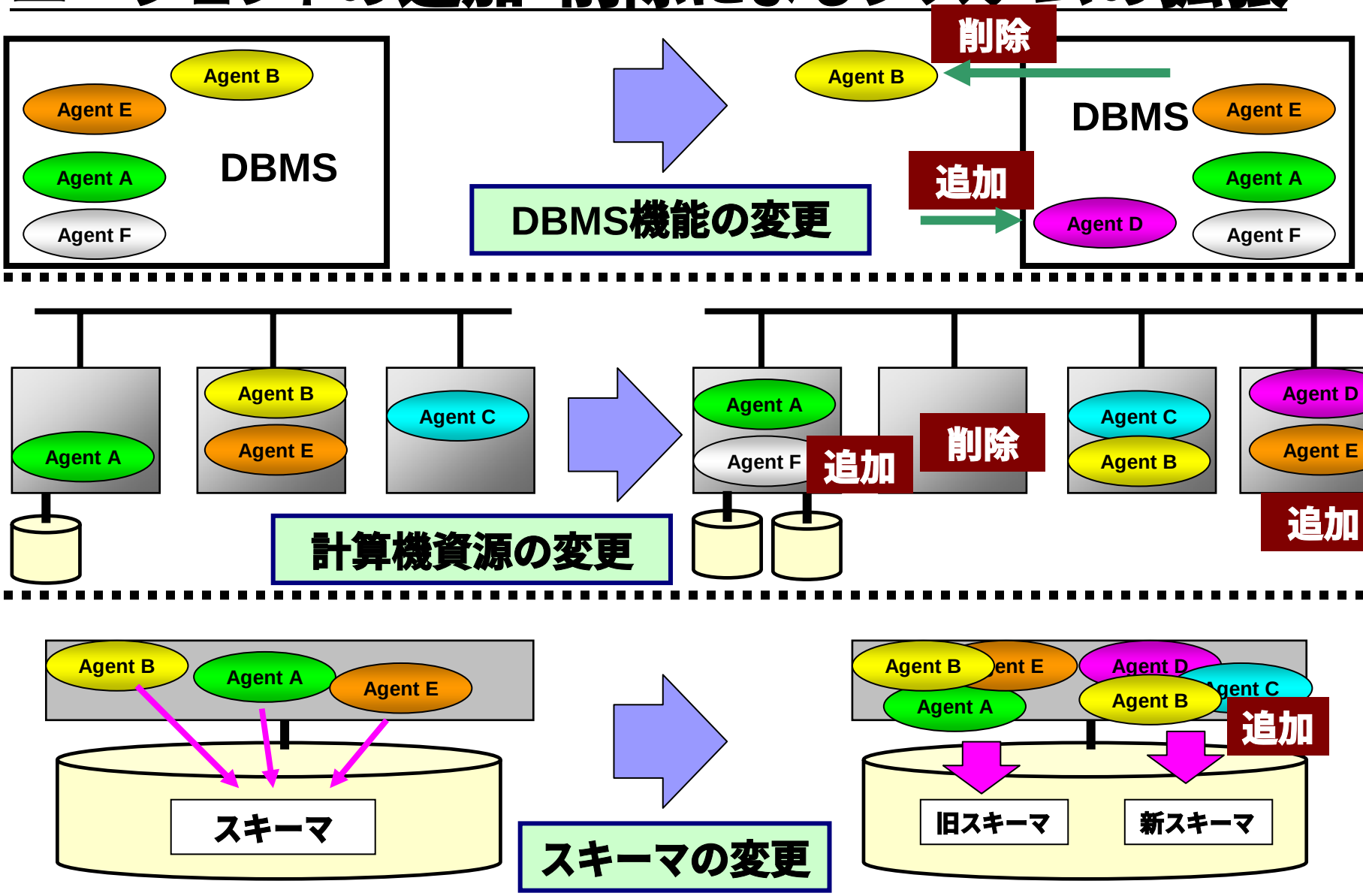
マルチエージェントシステムとは、

- 自立的に処理を行うエージェントが、互いに協調するシステムである。
- エージェントの追加・削除により、システムの自由な変更が可能である。

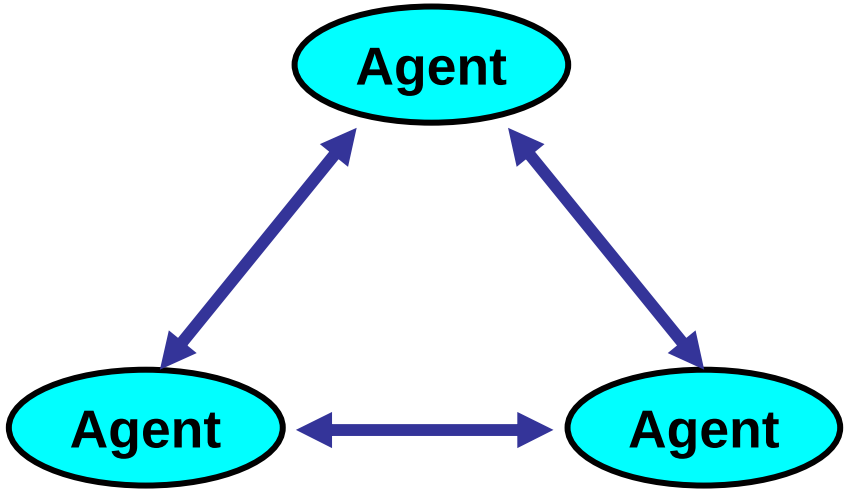


・エージェントを追加・削除することでDBMSに対する様々な変更を自由に行うことができる。

エージェントの追加･削除によるシステムの拡張

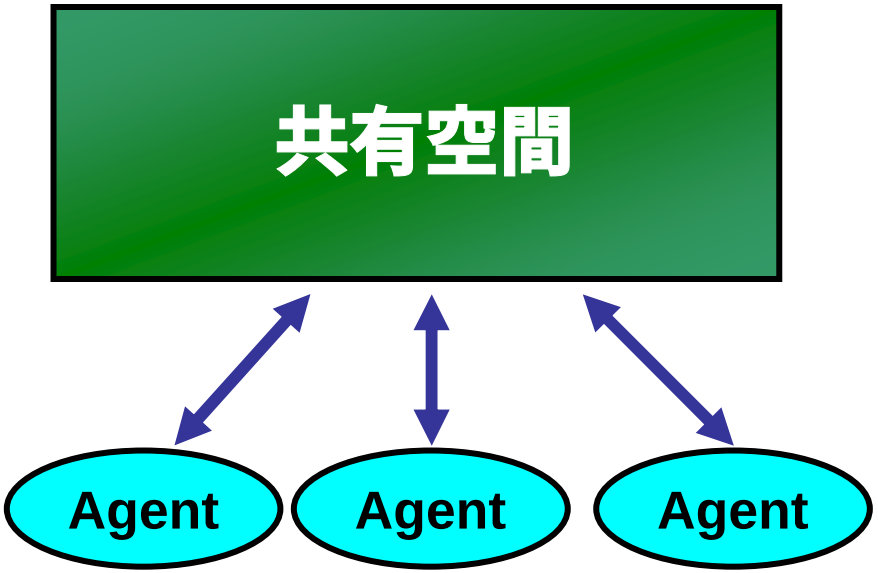


マルチエージェントシステムにおける エージェントの協調方法



エージェント同士が直接通信する方法

- エージェント間の情報の伝達がスムーズ



黒板システム[5]に代表される、
共有空間を介する方法

- 共有空間でエージェントの行動を管理できる
- 個々のエージェントの独立性が高い

[5]Hayes-Roeh,B.: “A Blackboard Architecture for Control”,Artificial Intelligence,Vol.26,pp.251-321,1985.

スクラップアンドビルド型DBMSにおけるエージェントの協調方式

共有空間を介した協調方法を採用

- 共有空間でエージェントの行動を把握できる。
- 個々のエージェントの独立性が高い。

- 障害時に備えて、エージェントの行動についてのログをとる。
- エージェントの追加･削除を自由に行いDBMSを柔軟に変更する。

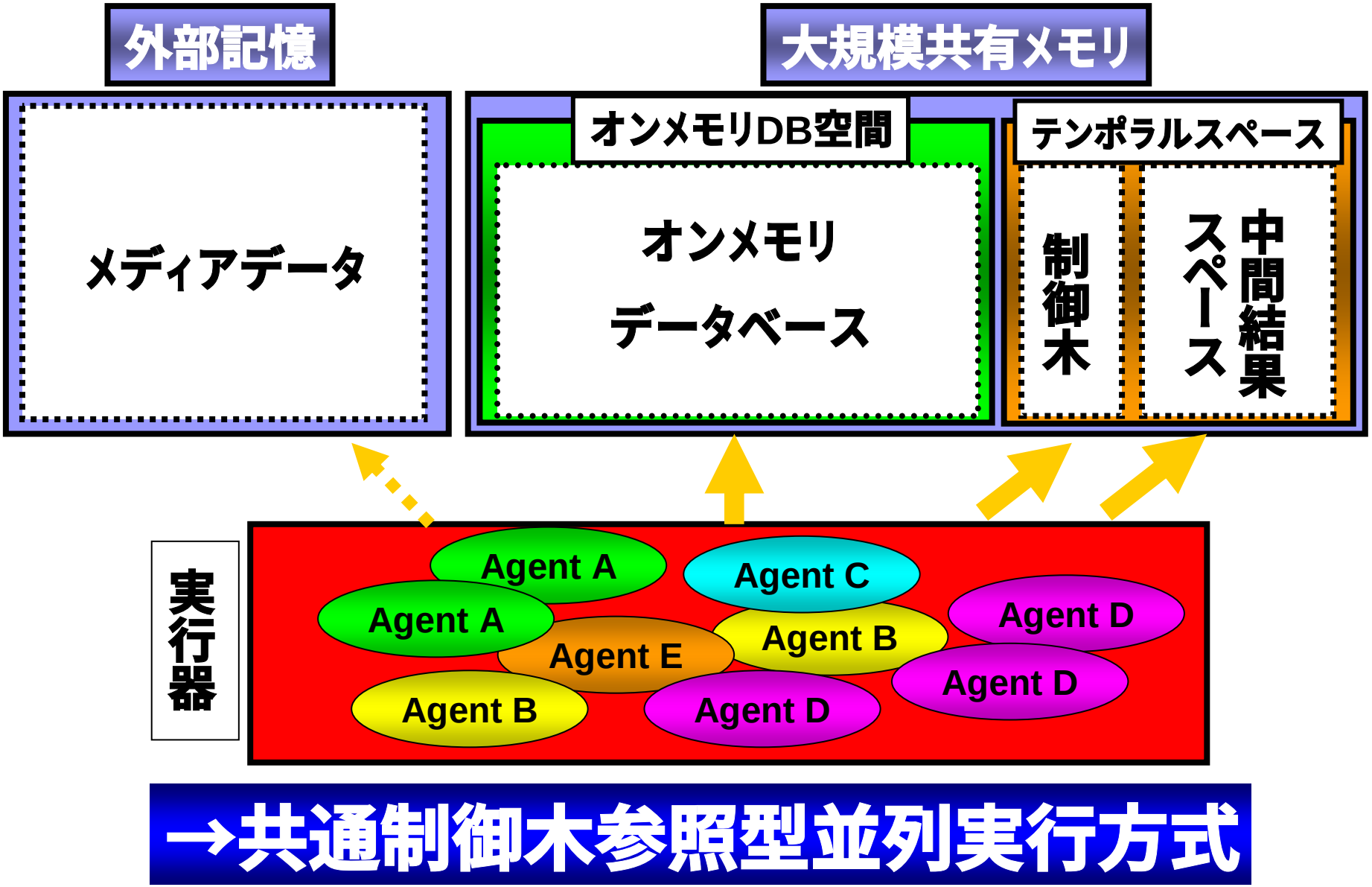
といったことが可能となる。

スクラップアンドビルド型DBMSの設計

このような「共有空間を介して協調するマルチエージェントシステム」を用いて

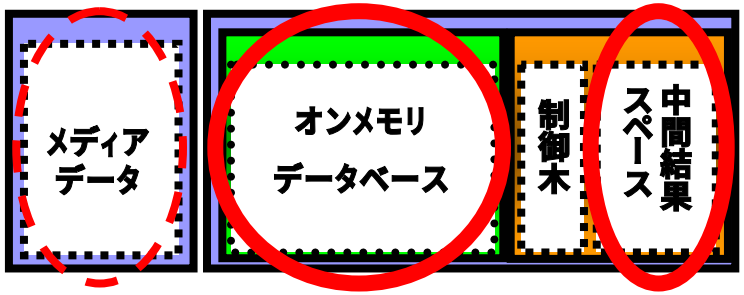
スクラップアンドビルド型DBMSを設計した。

スクラップアンドビルド型DBMSの概念図



→共通制御木参照型並列実行方式

共通制御木参照型 並列実行方式について(1)



- **オンメモリデータベース**

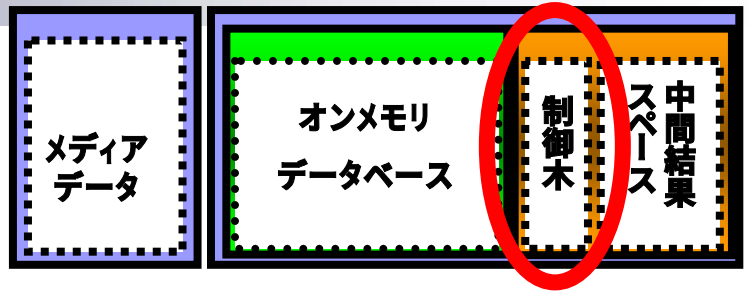
エージェントのDBへのアクセスが主に行われるところで、必要に応じて外部記憶からメディアデータを取り出す。

- **中間結果スペース**

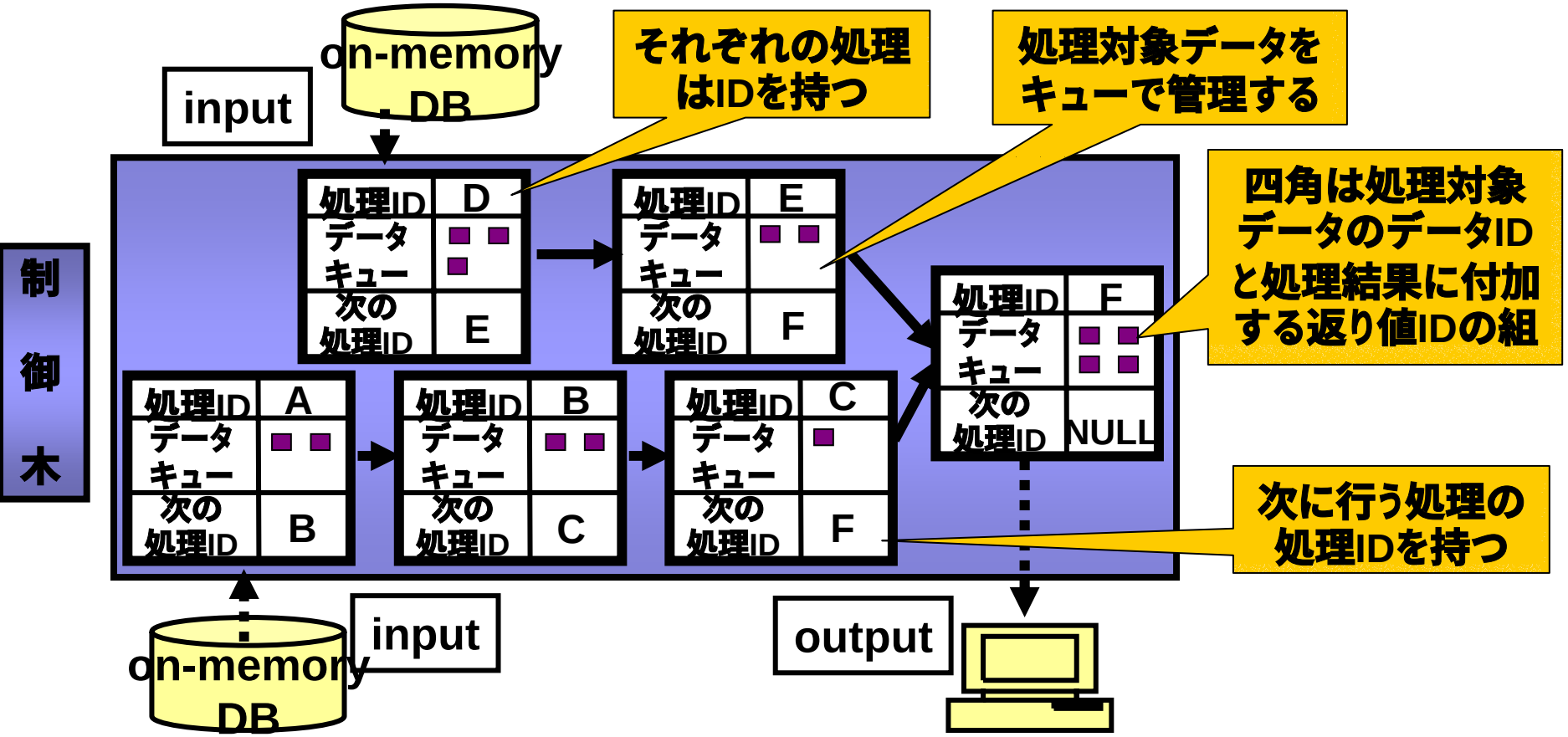
エージェントが処理結果を書き込むスペースで、最終的にはここに最終検索結果が生成される。

エージェントは処理結果に返り値IDを付加して書き込み、読み出しはIDを指定して行う。

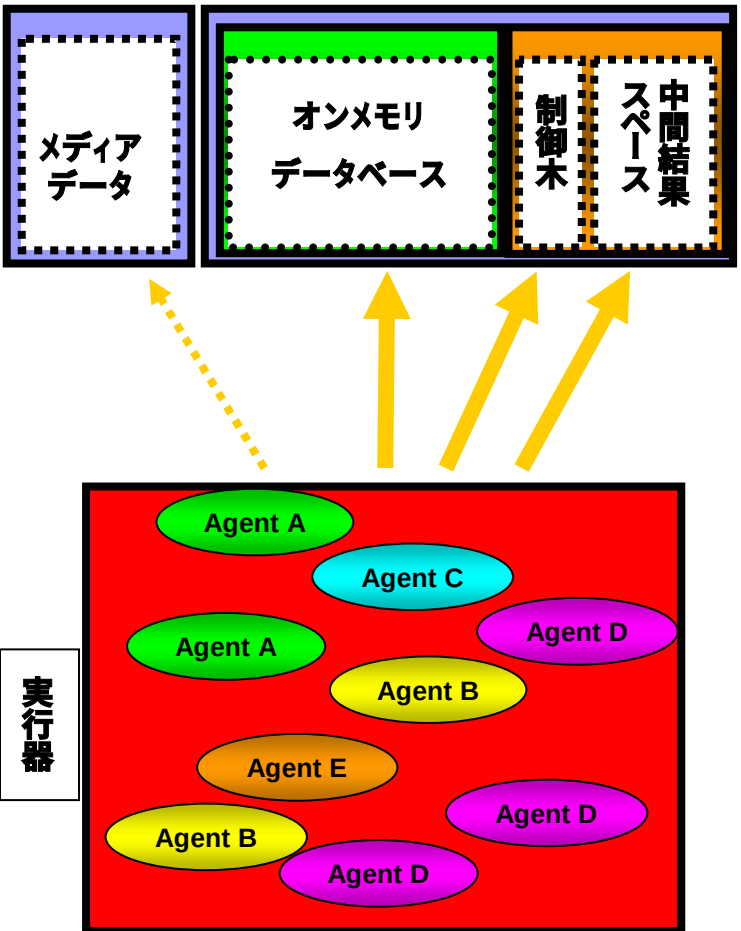
共通制御木参照型並列実行方式について(2)



➢**制御木**・・・処理の流れを表すもの。有向グラフによって処理の手順を示す。



エージェントによる検索処理の流れ



制御木←処理、処理対象のデータがあるか確認

見つかった

制御木←そのデータに対しての予約を要求

予約が認められた

オンメモリDB or 中間結果スペース←そのデータIDを持つデータを取り出し処理を行う

処理が終わった

中間結果スペース←処理結果を返り値IDと共に書き込む

制御木←次の処理ノードのデータキューに返り値IDを挿入する

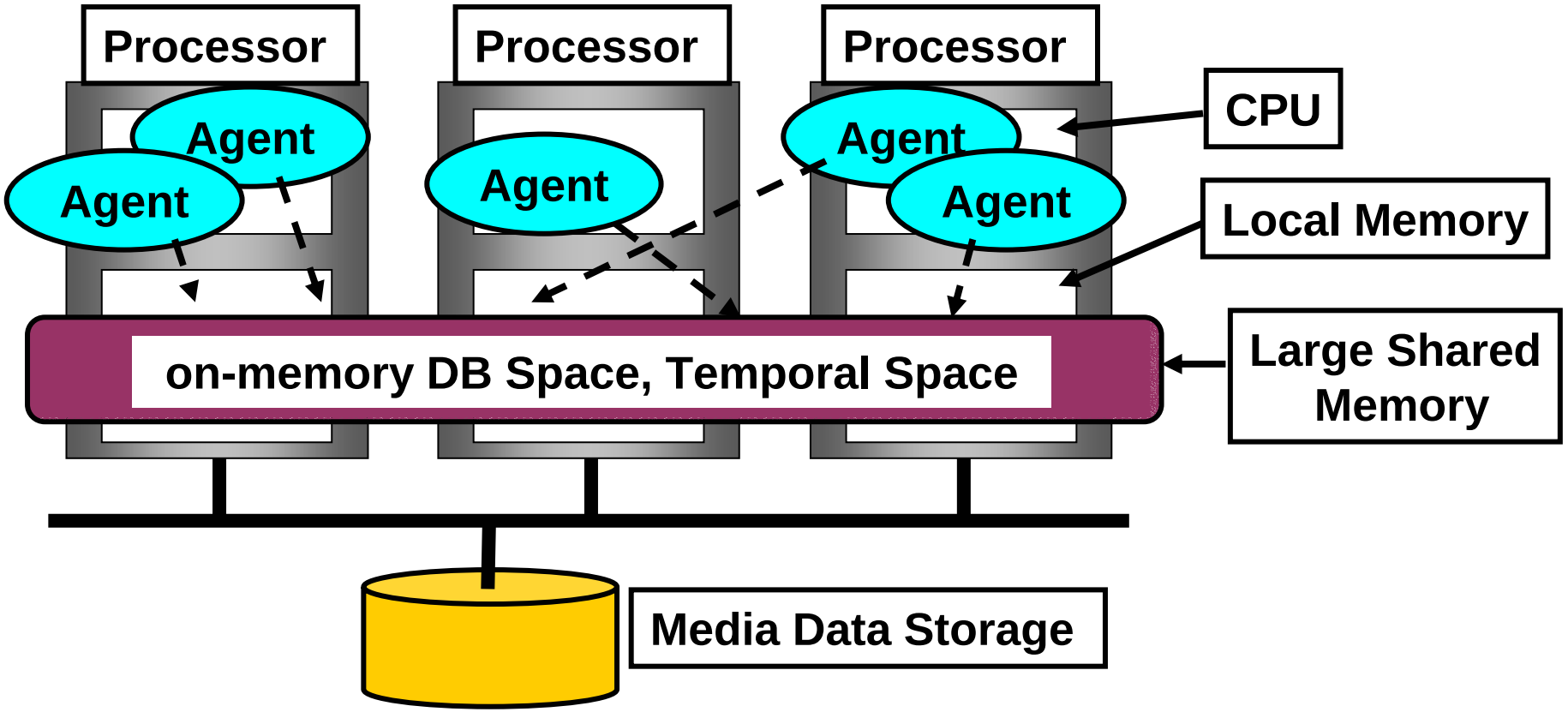
これまで述べてきたスクラップアンドビルド型
DBMSの概念設計に基づいて

並列計算機上でプロトタイプシステム

を構築した

並列計算機を用いて開発中のスクラップアンドビルド型DBMSの構成

大規模共有メモリに対する高速な読み書きが求められることから並列計算機上でシステム構築を行っている

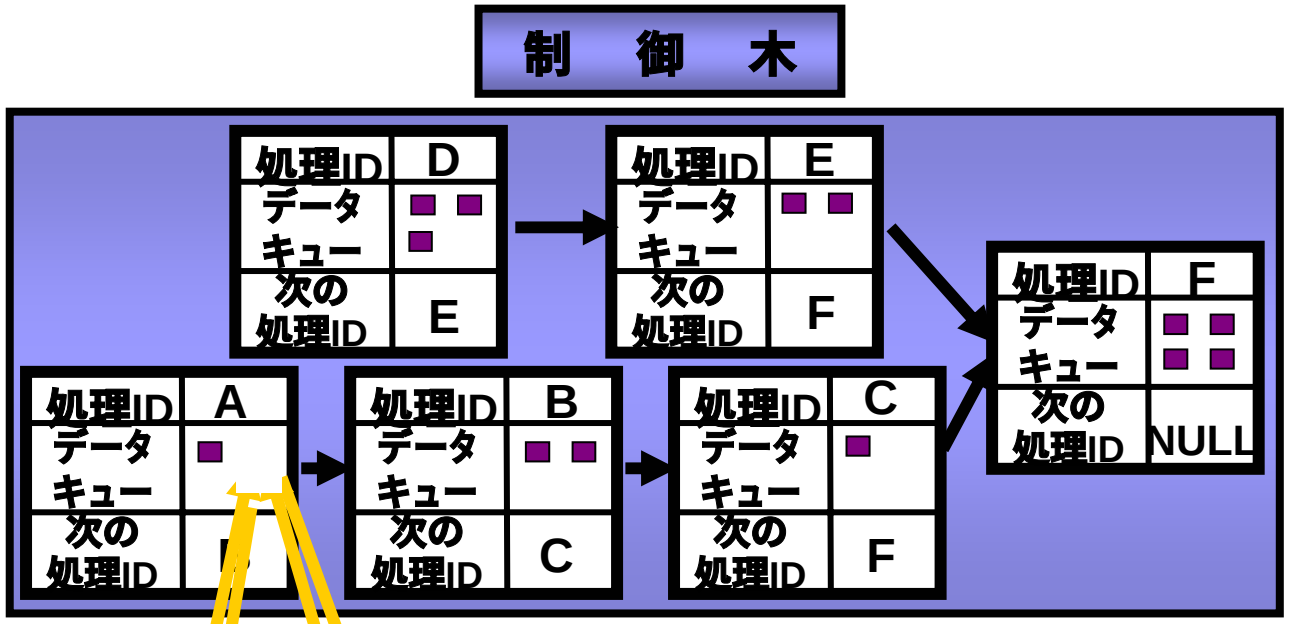


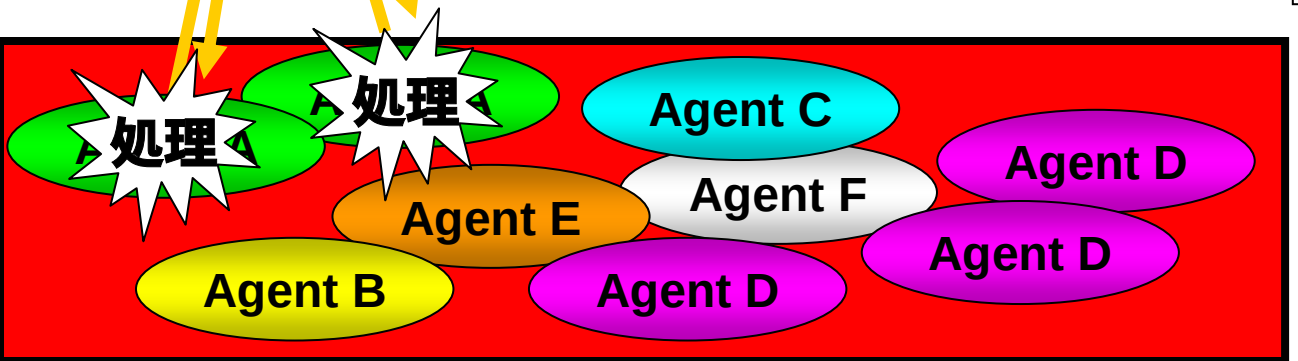
スクラップアンドビルド型DBMS

これ以降は、内容が大きく変わって、このプロトタイプシステムにおいて重要な役割を果たす、共有メモリの排他制御機構とその評価について述べる。

共有メモリの排他制御機構

共有メモリの排他制御機構の必要性





同一データに複数のエージェントから同時に予約要求がきた。

複数の予約を認める

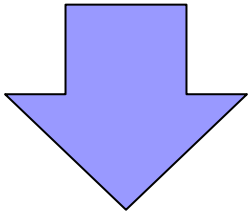
複数のエージェントがそれぞれ処理を行う

同一データに対して複数回処理が行われてしまう。

スクラップアンドビルド型DBMSを構築する上での課題

■複数プロセッサにまたがる大規模共有メモリ

■多数のエージェントからのアクセス



共有メモリにおける排他制御機構が重要な課題

エージェントからのアクセス・更新が頻繁に起こる制御木の排他制御機構に重点を置き、新たな管理方法を設計した。

また、一般的な管理方法と比較し評価を行った。

server-client　TYPE（一般的な管理方法）

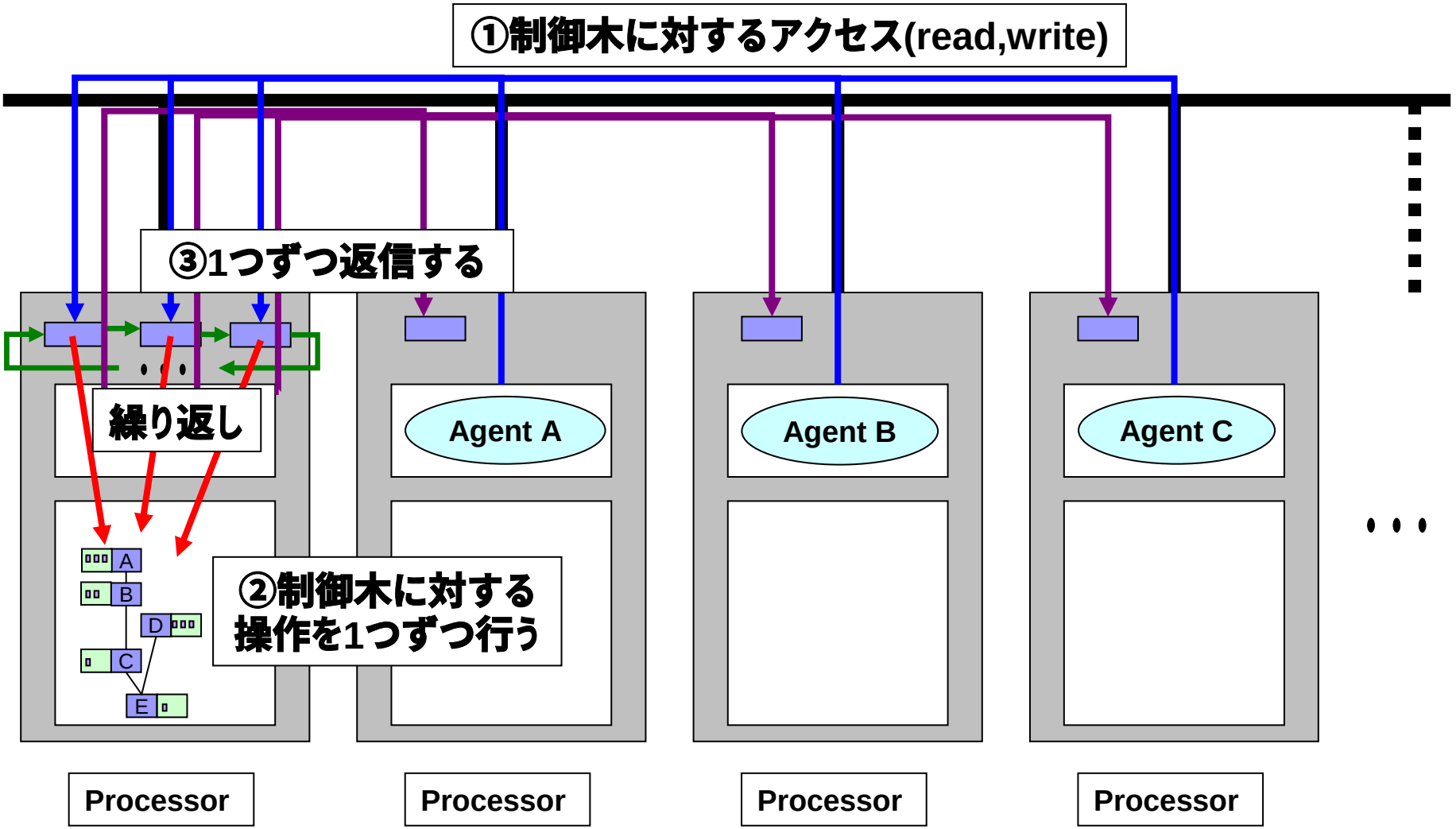
従来から用いられる一般的な管理方法[3]

我々の開発しているプロトタイプでも用いられている方法

サーバ側で共有空間上のデータ（ここでは制御木）を一括管理し、クライアントはサーバを通して共有空間にアクセスする。

Earth[3]では、これをさらに細かく8種類に分類し、よりシステムアーキテクチャに合った共有方式を提供している。

server-client TYPE（一般的な管理方法）



server-client TYPEの特徴

○ メモリを使用する領域が小さい

○ 一ヶ所で管理するため、管理が容易

× serverとなるプロセッサに負荷が集中

× serverとclient間の通信が頻繁に起こる

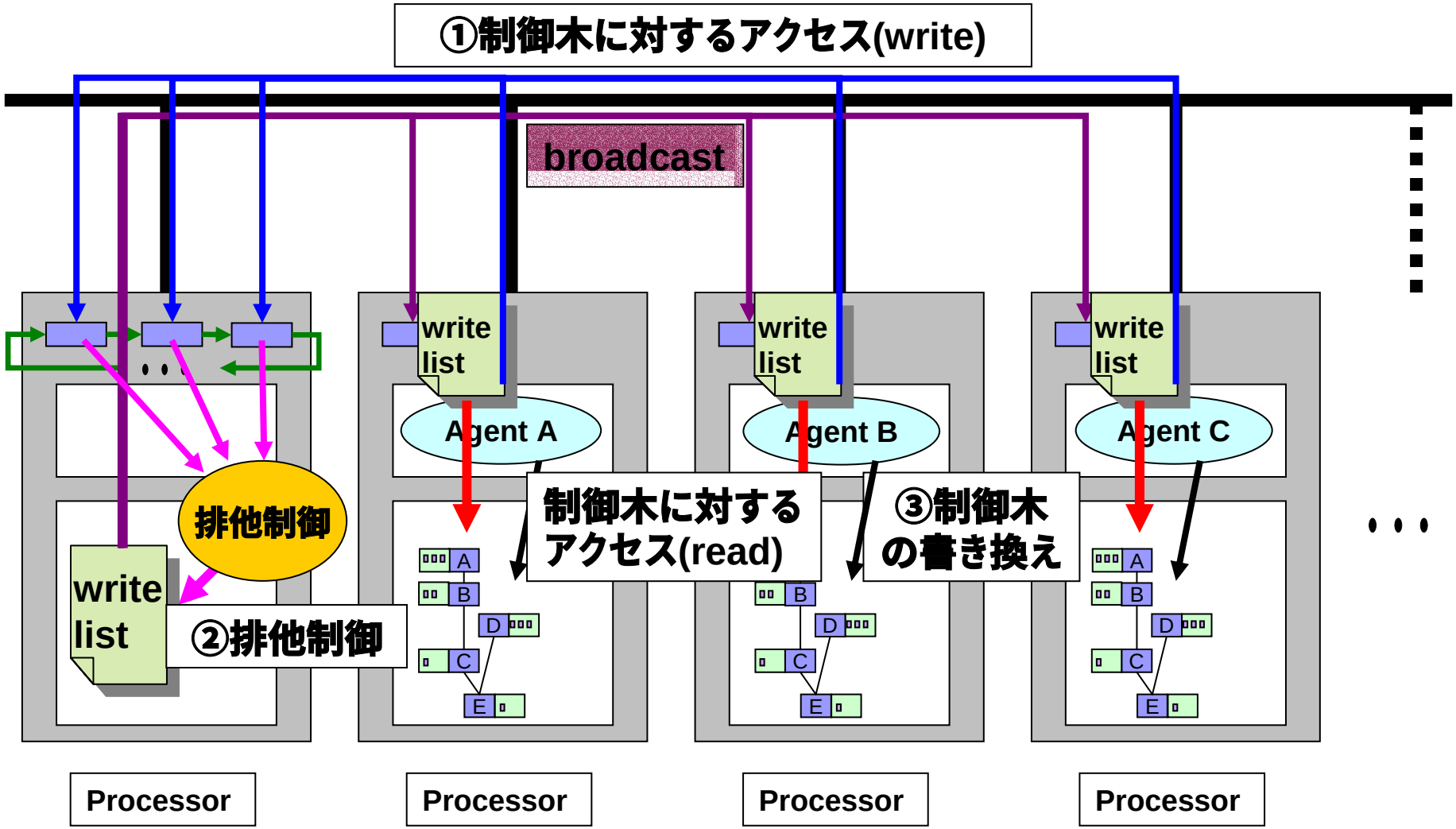
sever-client TYPE改良型

sever-client TYPE改良した管理方法で制御木をクライアント側に持たせる。

エージェントのreadアクセスは自メモリ上の制御木を参照する。

サーバ側ではwrite時の排他制御のみを行う。

sever-client TYPE改良型



server-client TYPE改良型の特徴

× メモリを使用する領域が大きい

○ read処理の高速化

○ serverの負荷軽減

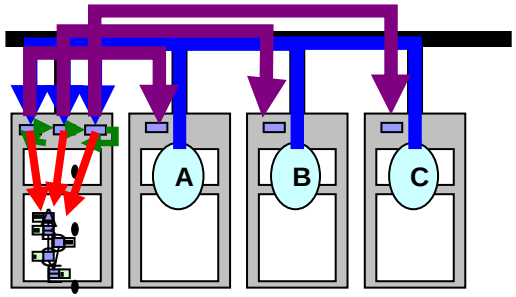
○ server-client間の通信回数が減る

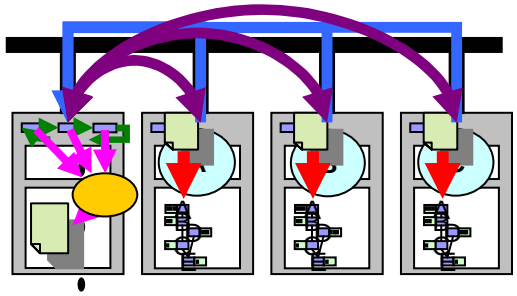
sever-client方式の問題点

serverとなるプロセッサが落ちてしまった場合

制御木が失われることや、排他制御が行えなくなることが考えられ、

検索システム全体が機能しなくなるといった問題がある。





全てのプロセッサが対等な関係にあり、

他に依存しないで排他制御を行う

管理方法を新たに提案する。

今回新たに提案する　broadcast　TYPE

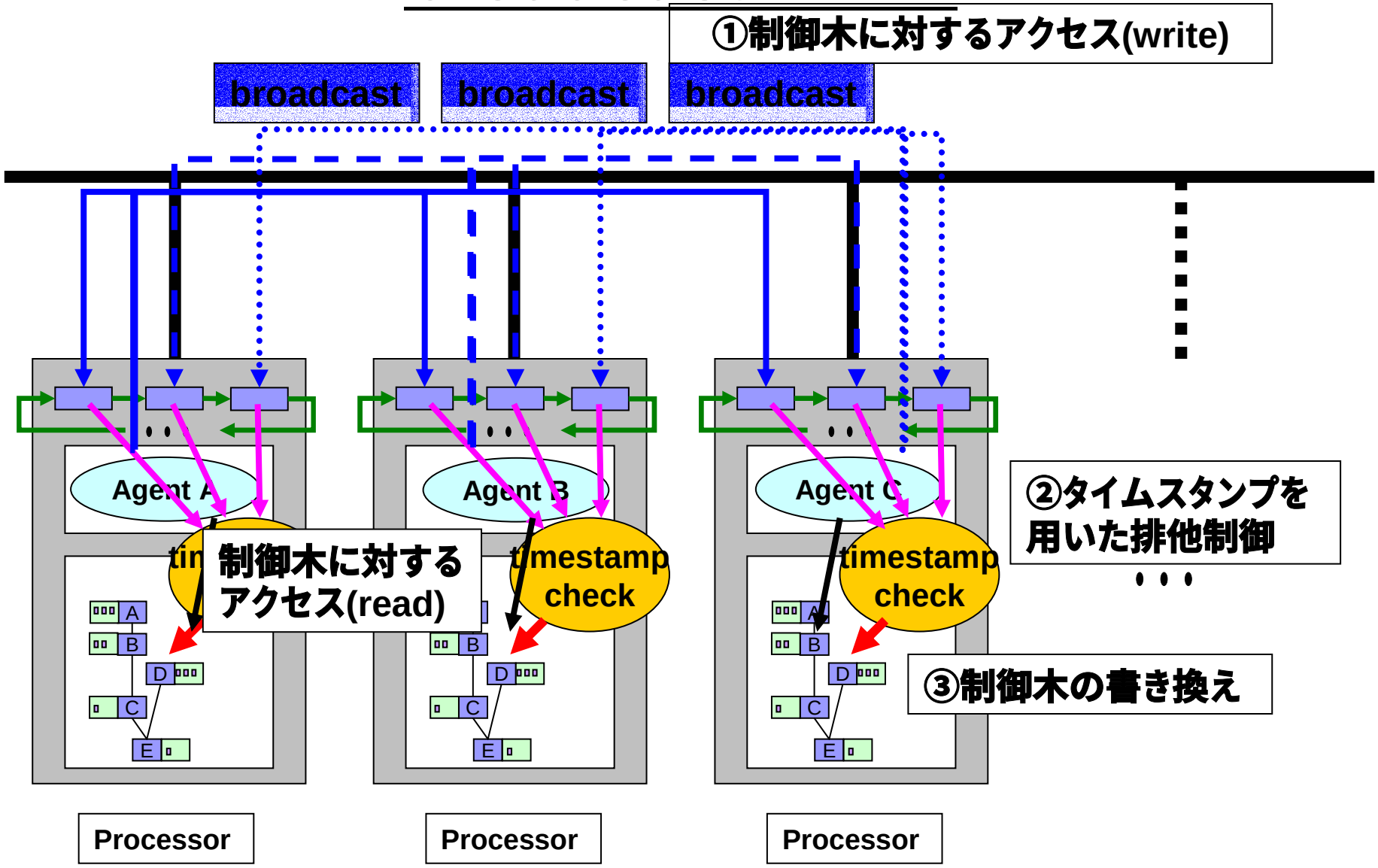
障害に強い制御木の管理方法として新たに提案するもの。

全てのプロセッサが対等な関係にあり、それぞれ制御木を持つ。

readは自メモリ上の制御木を参照し、write命令は全てのプロセッサにブロードキャストする。

排他制御はtimestampを用いて行う。

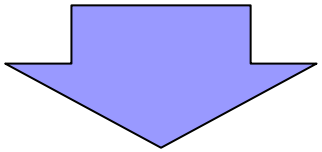
broadcast TYPE



broadcast TYPEの特徴

× timestampを用いた複雑な排他制御

○ 各プロセッサが対等であるため、部分的な障害が全体には大きく影響しないため信頼性が高い



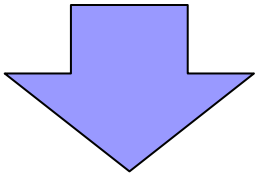
大規模で長期間運用するなど高い信頼性が求められるDB を対象とする本DBMSには適した管理方法であると考えられる。

性能評価の必要性

従来の管理方法に比べ信頼性の高いbroadcast Typeだが、
ブロードキャストの多発した場合、

通信にかかるオーバヘッドが増し、

処理に時間がかかることも考えられる。



従来の管理方法との比較・評価を行う。

各管理方法の性能評価実験

各管理方法の処理性能を計るためそれぞれ

制御木に対する同時に起こる処理要求の数（同時アクセス数）
と
read・write処理時間

の関係を調べた。

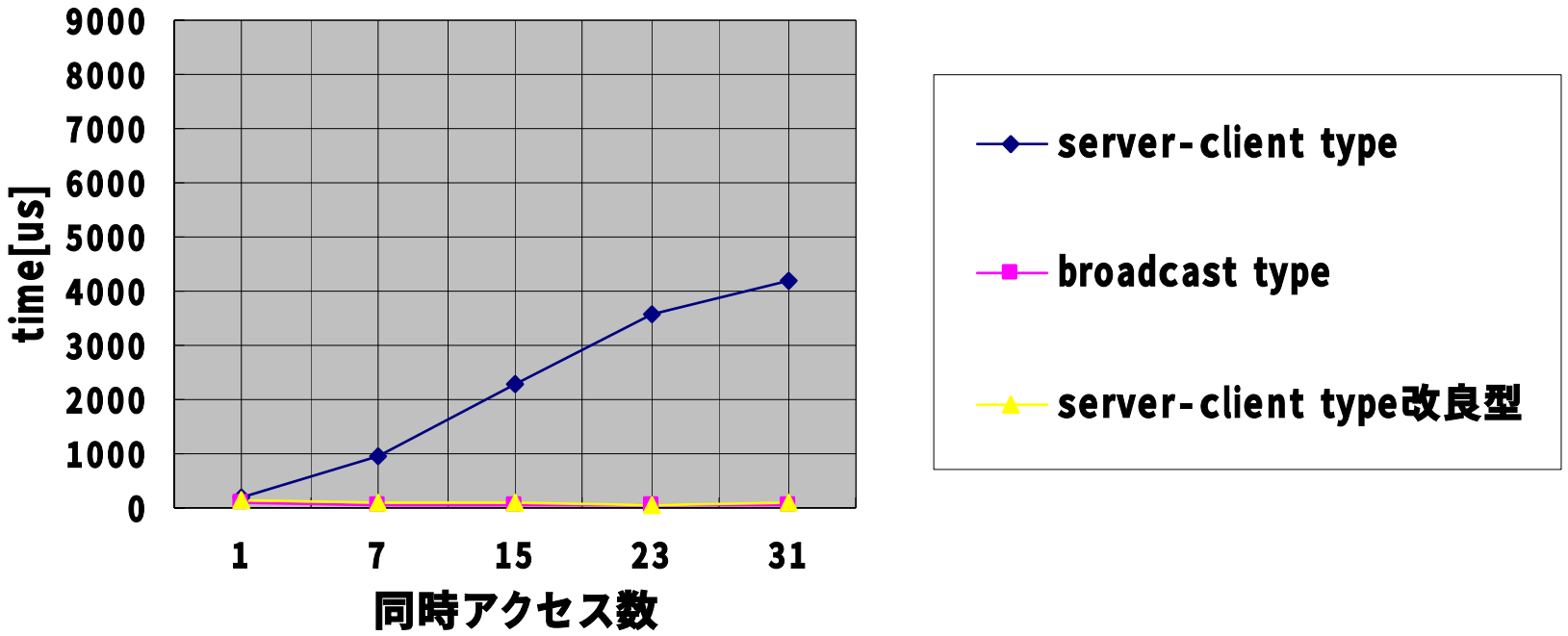
※各実験は、それぞれの処理を10回ずつ測定し、その処理にかかった最小時間と最大時間を除き、平均をとったものを測定値とした。

実験環境

•計算機	HITACHI SR2201
	プロセッサ数：32
	メモリ:512MB×9，256MB×23，　計10.5GB
•OS	HI–UX / MPP
•開発環境	C言語
•使用コンパイラ	Cコンパイラ（CC）
•通信方式	リモートDMA

実験結果(1/4)

同時アクセス数とread処理時間の関係

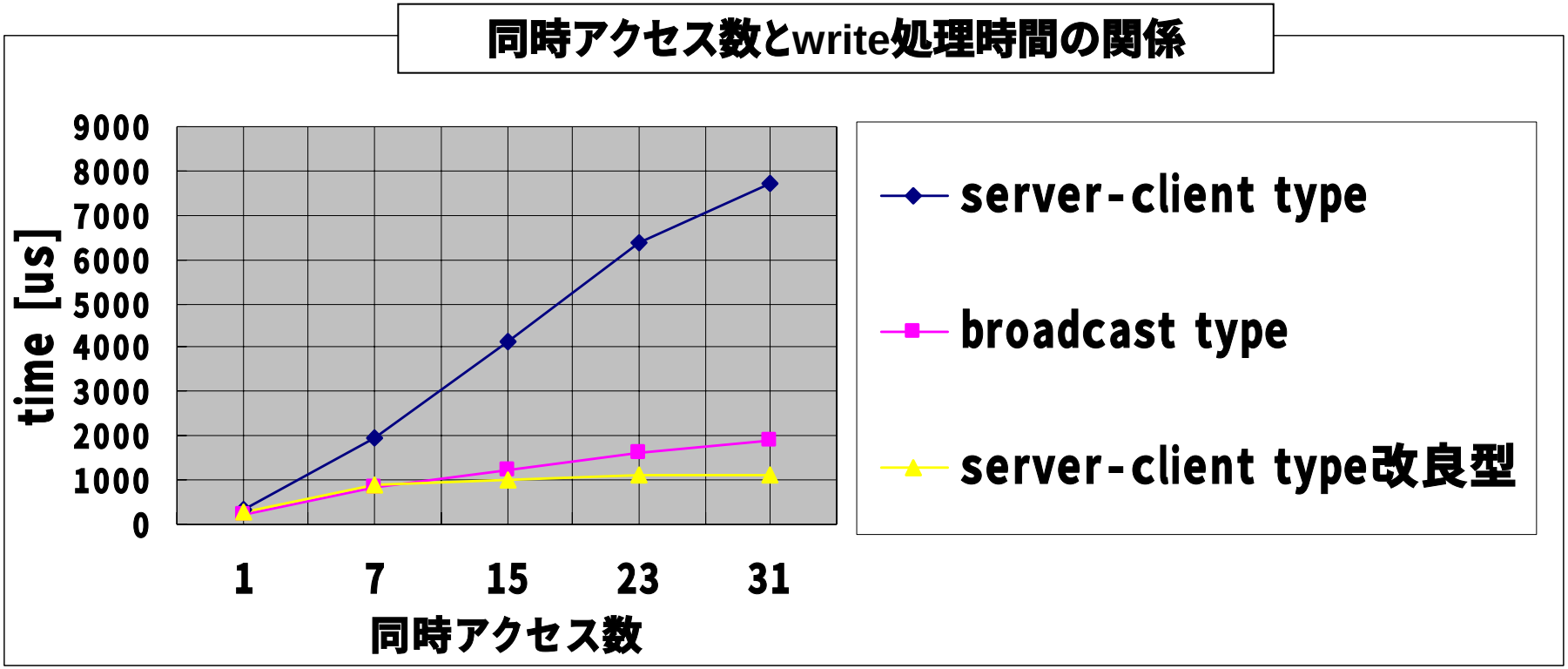


- broadcast TYPEのread処理は、自メモリ上の制御木にアクセスするため

  同時アクセス数に寄らず

  非常に高速に処理が行えることが確認できる。

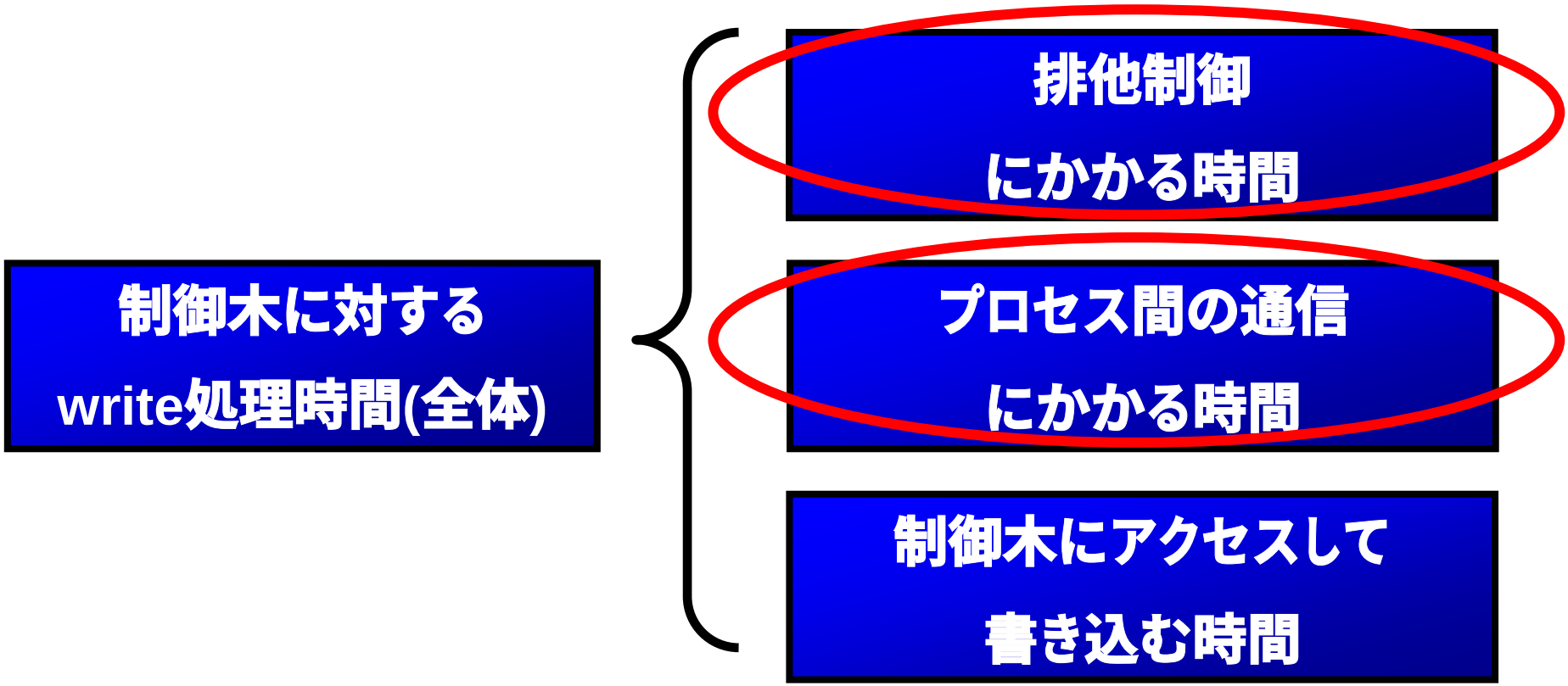
実験結果(2/4)



➢処理性能が最も高いserver-client TYPE改良型と比べて、broadcast Typeが特定のケースにおいては多少劣るもののほぼ同等の性能を示していることが実験の結果よりわかった。

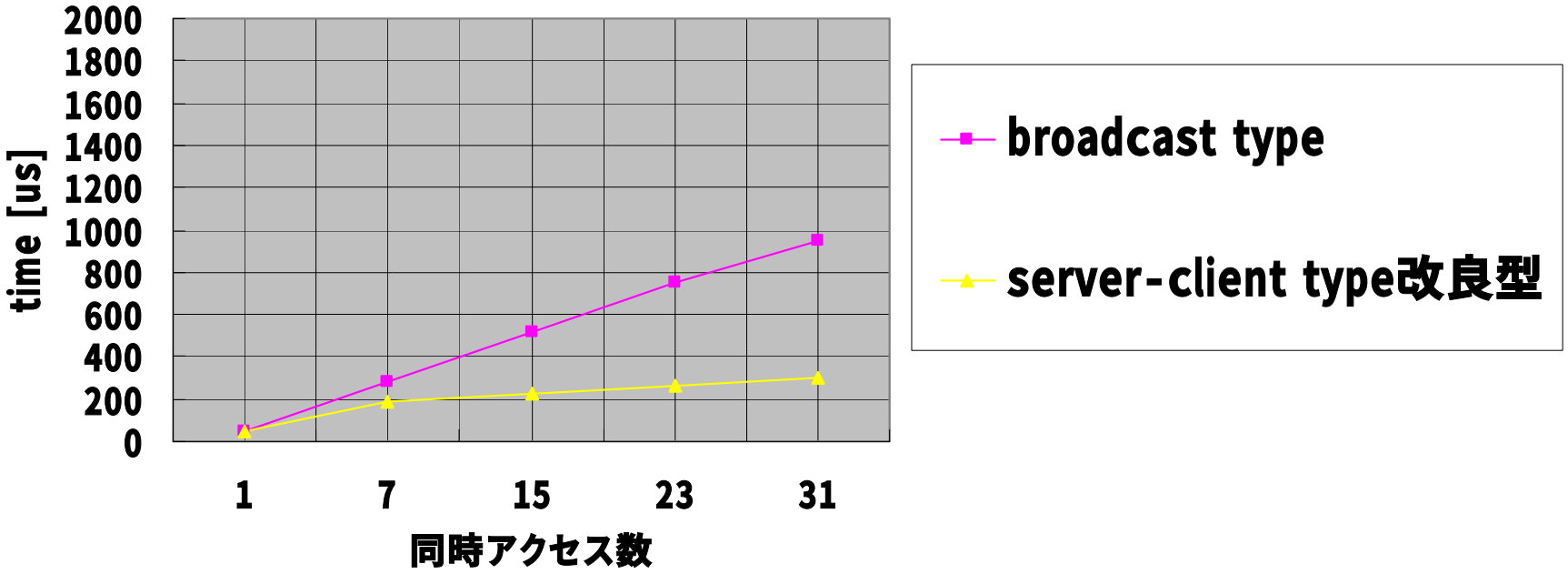
write処理時間(内訳)

同時アクセス数とwrite処理時間の関係をさらに検証する。



実験結果(3/4)

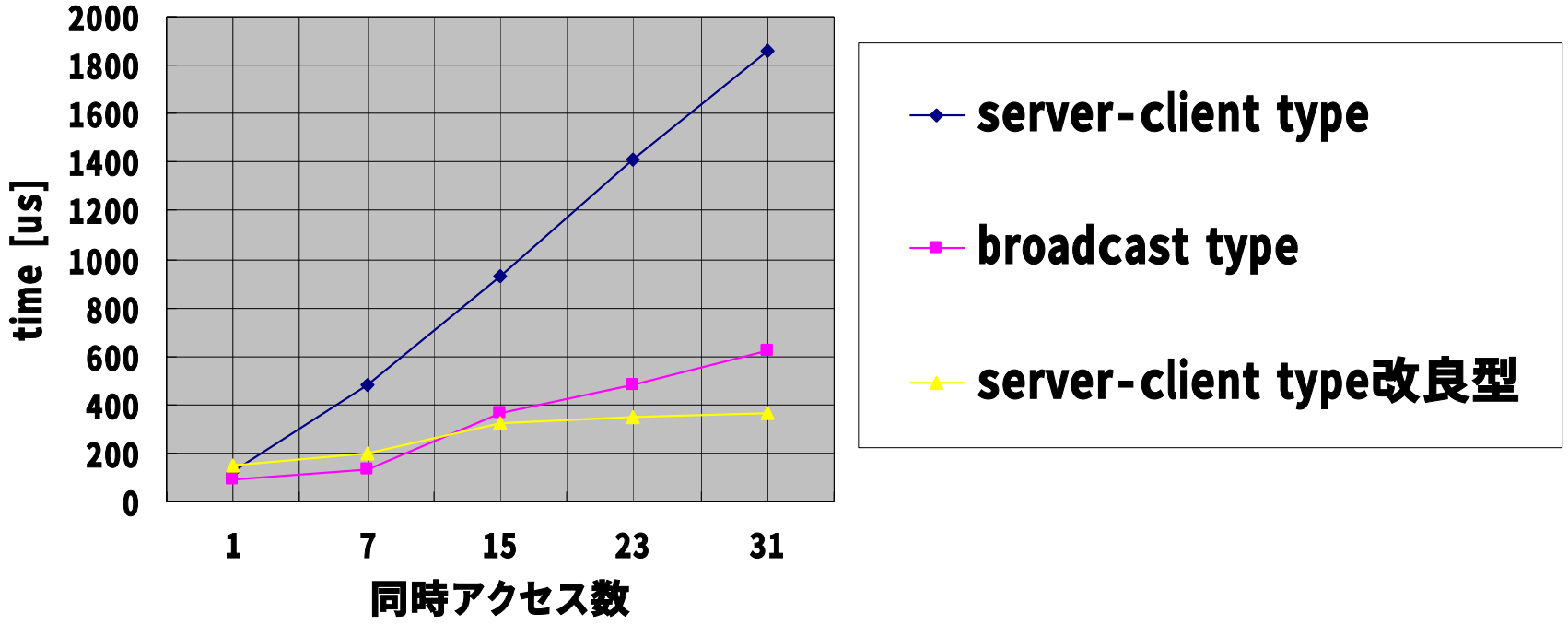
同時アクセス数と排他制御にかかる時間の関係



➢排他制御の方法の違いから同時アクセス数の増加と共に、改良型との差が開く。

実験結果(4/4)

同時アクセス数とプロセス間の通信時間の関係



➢同時アクセス数が少ないうちはbroadcast TYPEの方がプロセス間通信時間が短い。

考察

◆処理性能が最も高いserver-client TYPE改良型と比べて、broadcast　Typeが特定のケースにおいては多少劣るもののほぼ同等の性能を示していることが実験の結果わかった。

◆制御木を一括管理するserver-client Typeやその改良型に比べて、broadcast Typeは信頼性が高く、同時アクセス数がそれほど多くない場合には、制御木の管理方法として最も有用であると考えられる。

まとめと今後の課題

◆DBMSに対する様々な変更を自由に行うことのできるものとして、スクラップアンドビルド型DBMSの概念を提案し、以下の手法を用いてこの実現を行った。

1. マルチエージェントシステムを用いたシステム設計

2. 共通制御木参照型並列実行方式によるエージェントの協調

3. 排他制御を実現する、新しい共有空間の管理方法

◆実験により、各種管理方法の比較をし、最適な管理方法の分析を行った。

- 実際のアプリケーションに応用した場合の、本システムの有効性の検証
- 更なるパフォーマンスの向上