超並列計算機における並列入出力システム

筑波大学

電子・情報工学系

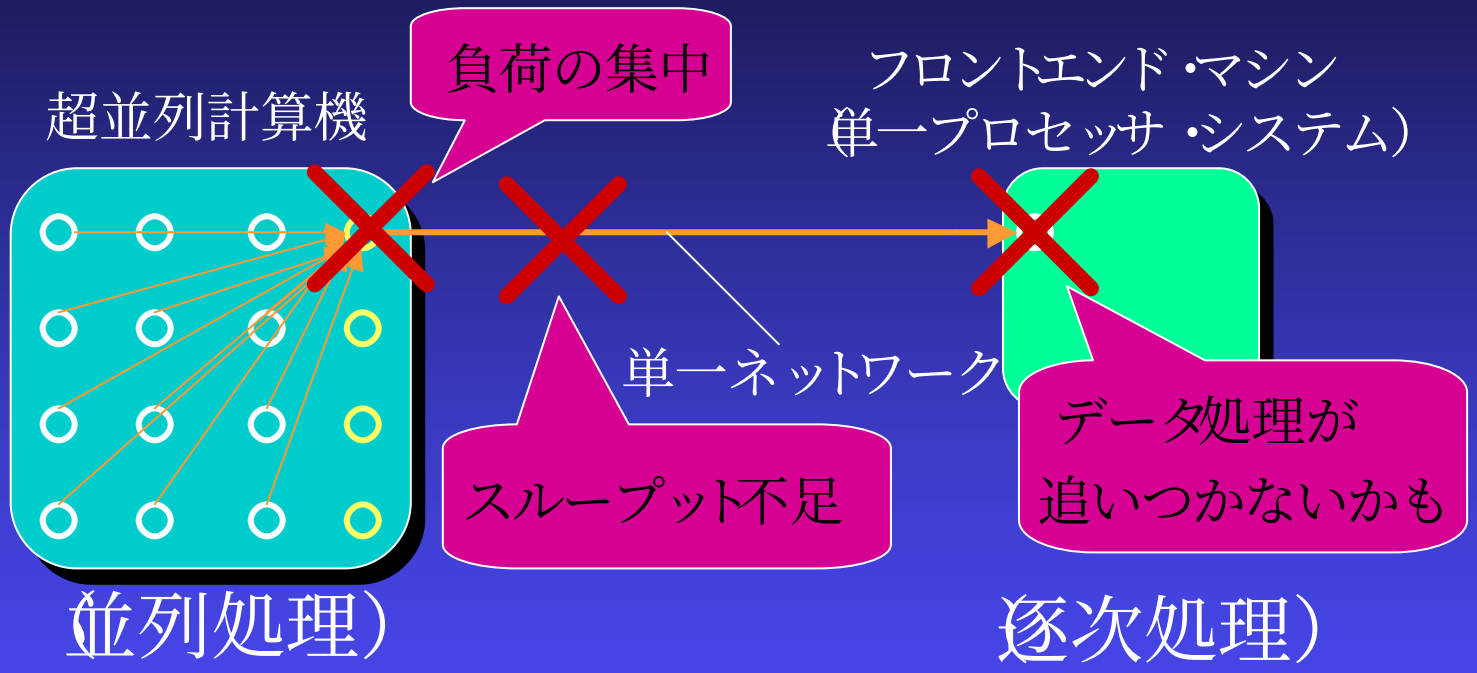
松原 正純

研究の背景

- ✓ 超並列計算機（MPP:Massively Parallel Processor）における科学技術計算により膨大な量の計算データが生成される
  - ⇒超並列計算機のデータ生成能力に見合った入出力システムが要求される
    - 計算と平行してデータ処理できるシステムが必要
      - （実時間的な利用が可能）

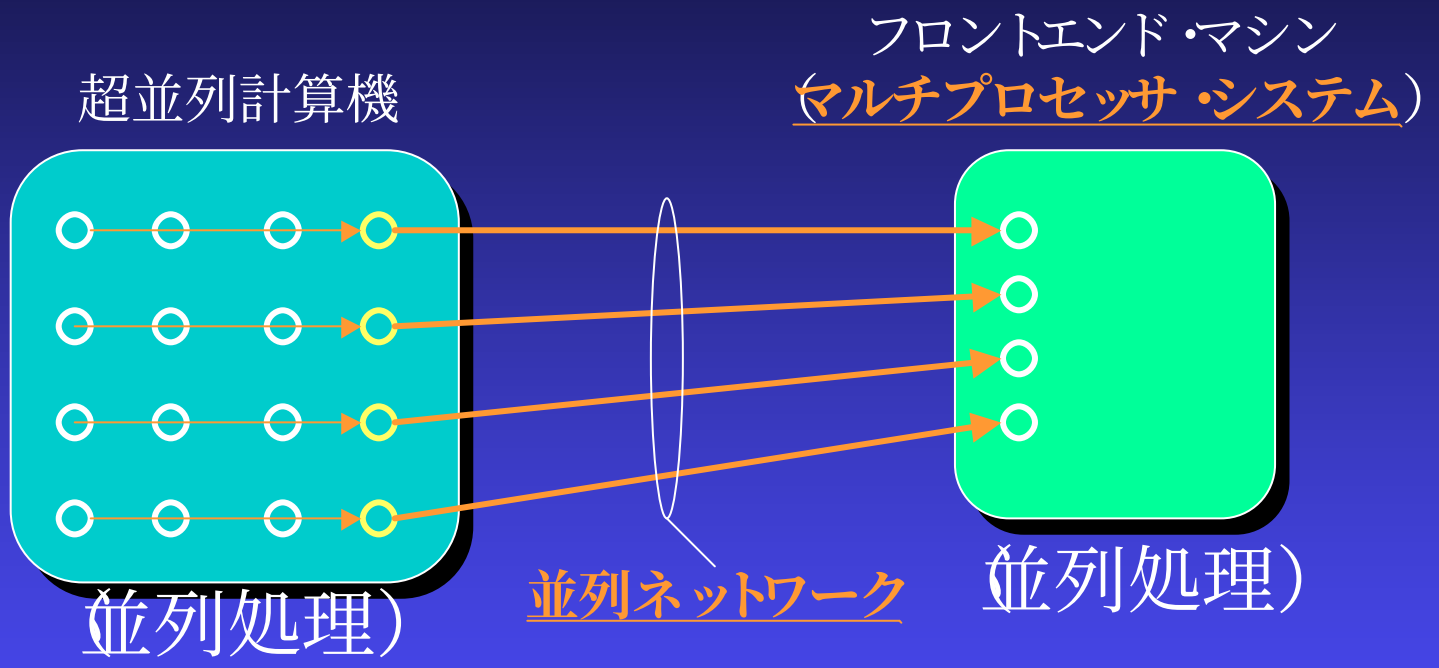
- ✓ 上記要件を満たす並列入出力システムを構築
  - なぜ並列入出力？
  - 従来のシステムのどこに問題があるのか？

従来の入出力環境



実時間的な利用を想定した場合、経路途中の逐次処理がボトルネックとなる

並列入出力環境



☆最初から最後まで並列に処理可能

既存の研究における問題点

- ✓ 主な用途として、ファイル転送を考えている
  - 可視化処理などでは、一旦中間ファイルを生成しなくてはならない
- ✓ クライアントが単一プロセスである
  - フロントエンド・マシンでのデータ処理が、超並列計算機のデータ生成能力に追いつかない場合が有り得る

途中で入る逐次処理がボトルネックとなる

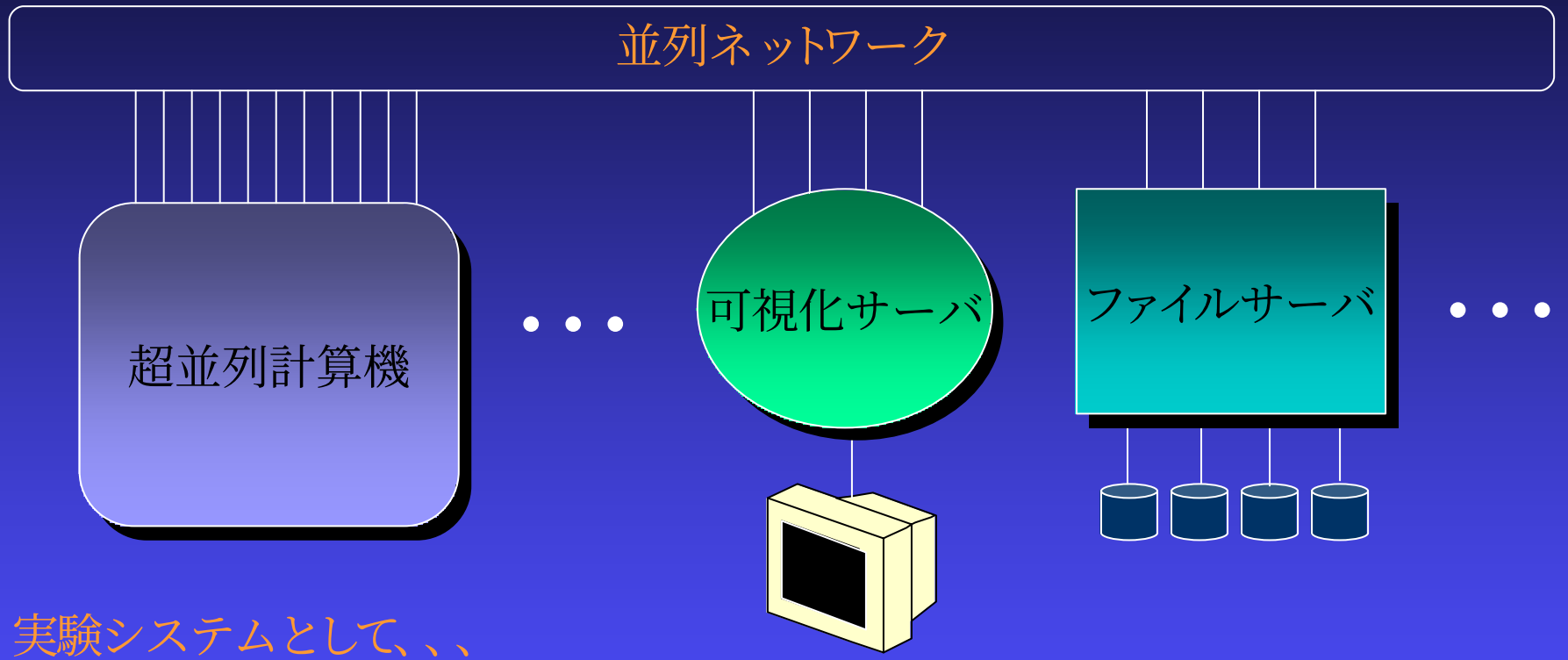
並列入出力システム設計の方針

- ネットワーク、フロントエンド・マシンとも並列化
  - 並列アプリケーション間での並列データ転送をサポート
    - 途中でボトルネックとなるような逐次処理がない

      ⇔ 並列ファイルI/O (HPSS, MPI-IO)
- commodityベースのネットワークを利用
  - コストパフォーマンスに優れている
  - 多重度を増すことができる

    ⇒ より多くの入出力プロセッサを活用できる

並列入出力システム環境



実験システムとして、、、

- ✓ 本学で稼動中の超並列計算機CP-PACSを利用
- ✓ フロントエンド・マシンとしては、共有メモリ型並列計算機Origin-2000, Onyx2、ALPHAクラスタを利用
- ✓ 最も入手が容易でコスト/パフォーマンスに優れた、100Base-TX Ethernetを使用

実験システム構成図

Massively Parallel Processor
CP-PACS
(2048 PUs, 128 IOUs)

8 links

Switching
HUB

100Base-TX
Ethernet
による並列接続

Parallel Visualization Server
SGI Onyx2 (4 Processors)

Alpha Cluster
HYADES (16 Nodes)

Parallel File Server
SGI Origin-2000 (8 Processors)

平成9年度

✓全体的な仕様の検討（特にハードウェア面）
Origin-2000の導入 (1997.9)

✓予備実験

10

✓並列入出力システムのプロトタイプ・バージョンの作成
Onyx2の導入 (1998.11)

11

✓入出力システムの仕様見直し、新バージョンの作成

12

✓分散メモリ型フロントエンドシステムへの対応
ALPHAクラスタの導入 (2000.11)

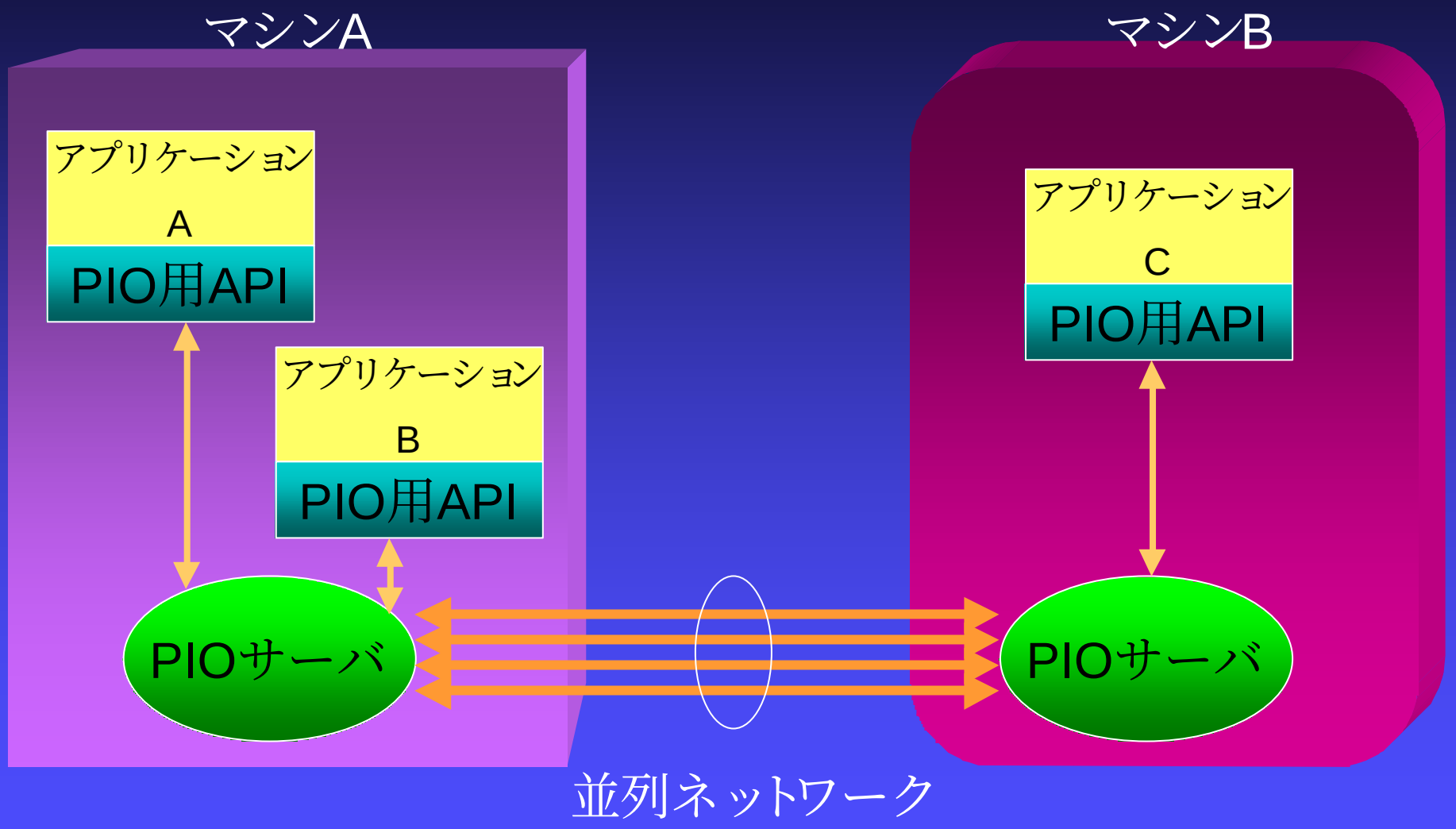
並列入出力システム (PIO System)

- ✓ 高性能で柔軟な入出力システム
  - 本システムではレイテンシよりもスループット重視
    ←大規模データ転送アプリケーションがターゲットだから
    – 並列ネットワークを有効に活用する機構が欲しい
  - 様々なマシン間で利用可能 ⇒ システムの移植性
  - プログラムの可搬性 ⇒ APIの提供

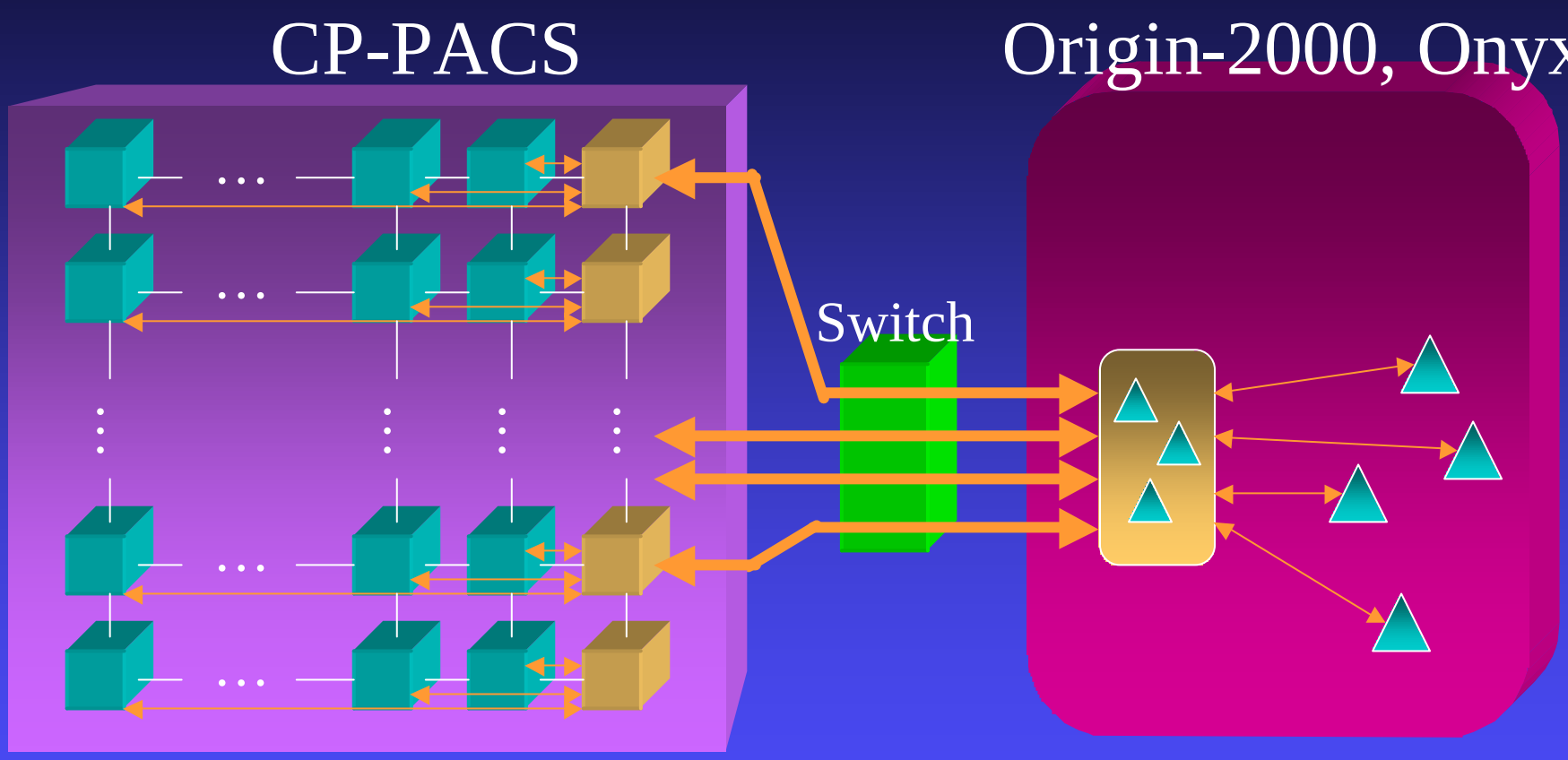
☆システム構成

- サーバ・プログラム（PIOサーバ）とAPIから構成
  – 簡便なAPIを用意することで、ユーザに並列ネットワークを意識させない
  – 複雑な制御はサーバ側で吸収

並列入出力システムの構成



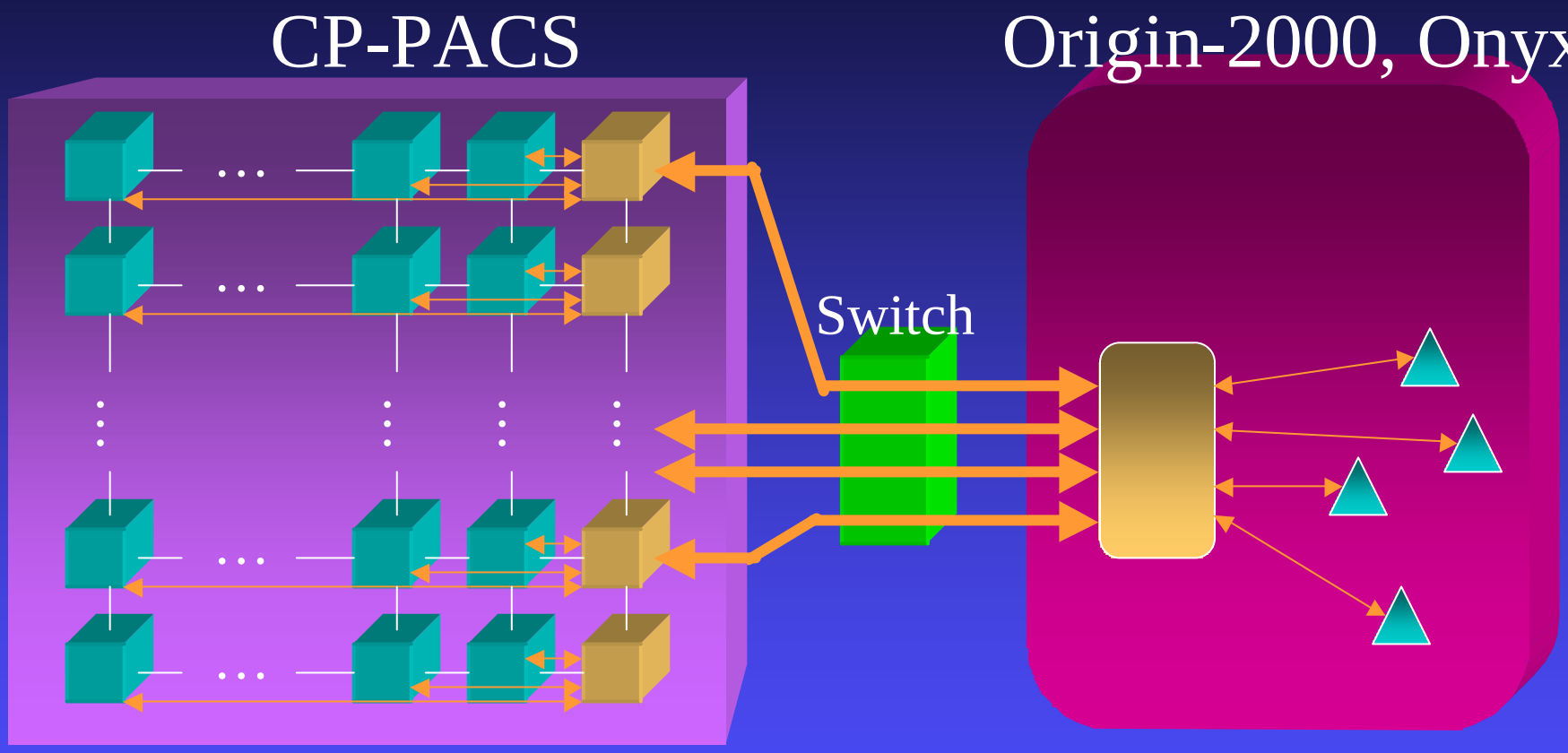
並列入出力システムの構成



並列入出力システム　— PIOサーバ —

- ✓ 並列アプリケーション間（マシン間）のデータ転送を管理
- ✓ 入出力データのバッファリング
- ✓ 余分なシステム情報の隠蔽
  - 複数あるIPアドレスなどをユーザが意識する必要がない
- ✓ 複数チャネル間での負荷分散
  - アプリケーションに応じた負荷分散方法を選択できるように、複数の負荷分散機能を用意
- ✓ 到着順序保証
- ✓ 様々な構成のシステムへの適応
- ✓ 異機種マシン間での通信における、外部データ表現の変換

並列入出力システムの構成

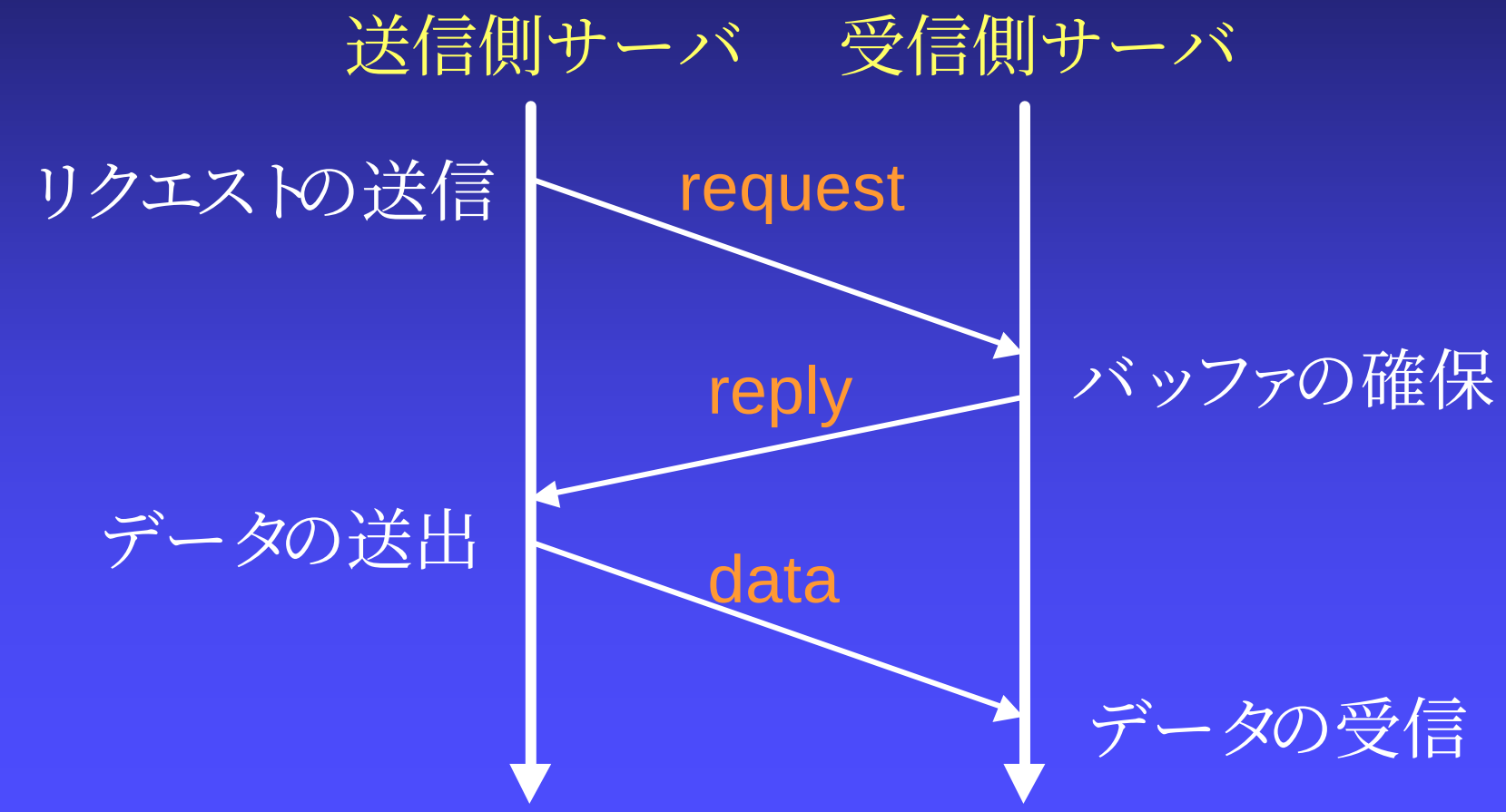


☆PIOサーバ間のデータ転送はどのようにして制御するのか？

通信プロトコル、データ転送方式、負荷分散

データ転送方式

- ✓ 3-way通信手順



並列入出力システム　— PIOサーバ —

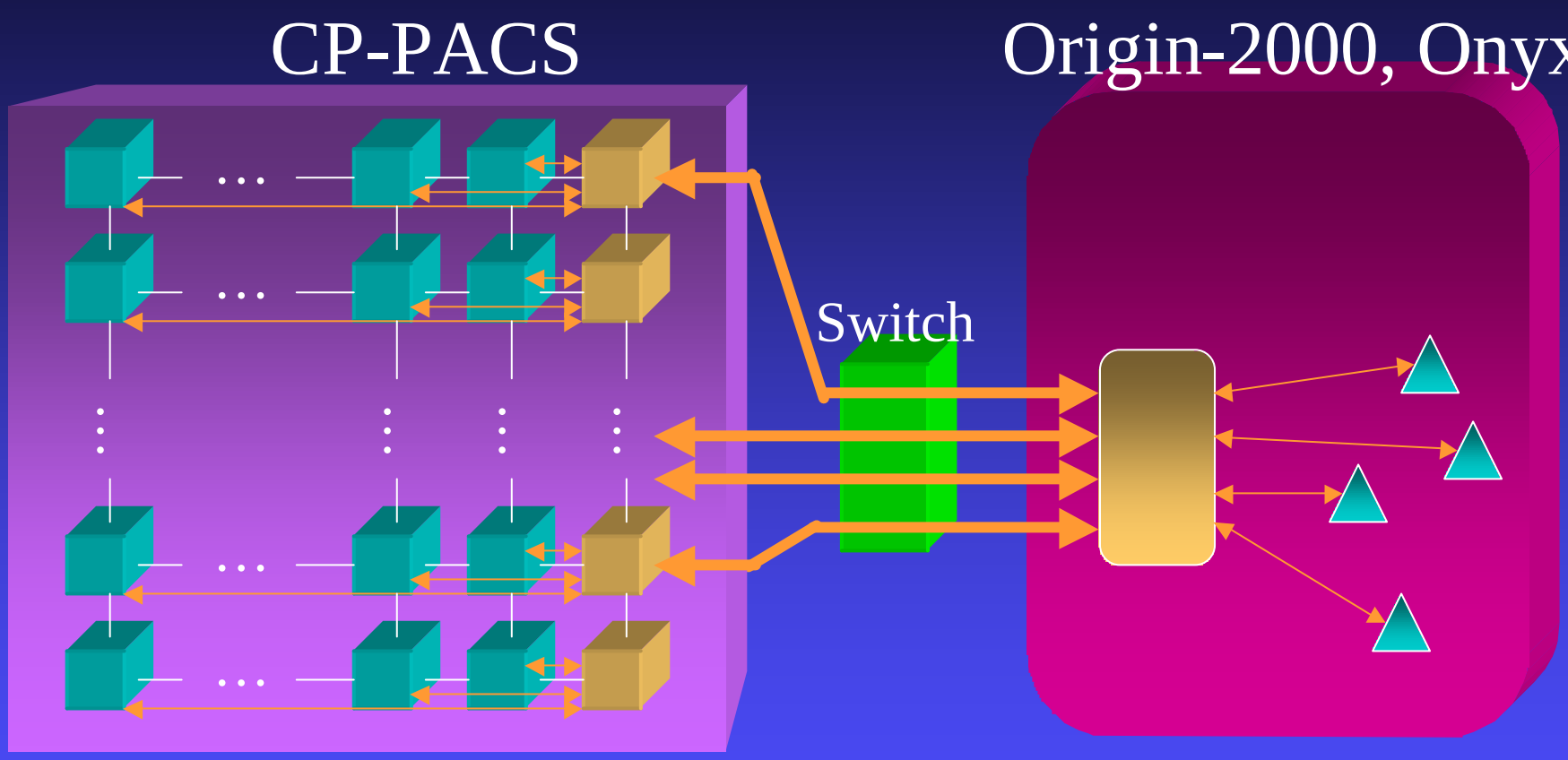
- ✓ 並列アプリケーション間（マシン間）のデータ転送を管理
- ✓ 入出力データのバッファリング
- ✓ 余分なシステム情報の隠蔽
  - 複数あるIPアドレスなどをユーザが意識する必要がない
- ✓ 複数チャネル間での負荷分散
  - アプリケーションに応じた負荷分散方法を選択できるように、複数の負荷分散機能を用意
- ✓ 到着順序保証
- ✓ 様々な構成のシステムへの適応
- ✓ 異機種マシン間での通信における、外部データ表現の変換

負荷分散方式

- ユーザによる負荷分散（USER）
  - 特定TCPコネクションを指定したい場合に有効
- システム（PIOサーバ）による静的負荷分散（STATIC）
  - 領域分割法等を扱うSPMD方式のアプリケーションに有効
- システム（PIOサーバ）による動的負荷分散（DYNAMIC）
  - 各コネクションの負荷に応じて適当なコネクションを選択する

負荷分散方式	長所	短所
USER	細かい制御が可能	プログラミングが複雑
STATIC	プログラミングが簡単	適用範囲が狭い
DYNAMIC	プログラミングが簡単 適用範囲が広い	オーバヘッドが増大

並列入出力システムの構成



動的負荷分散方式

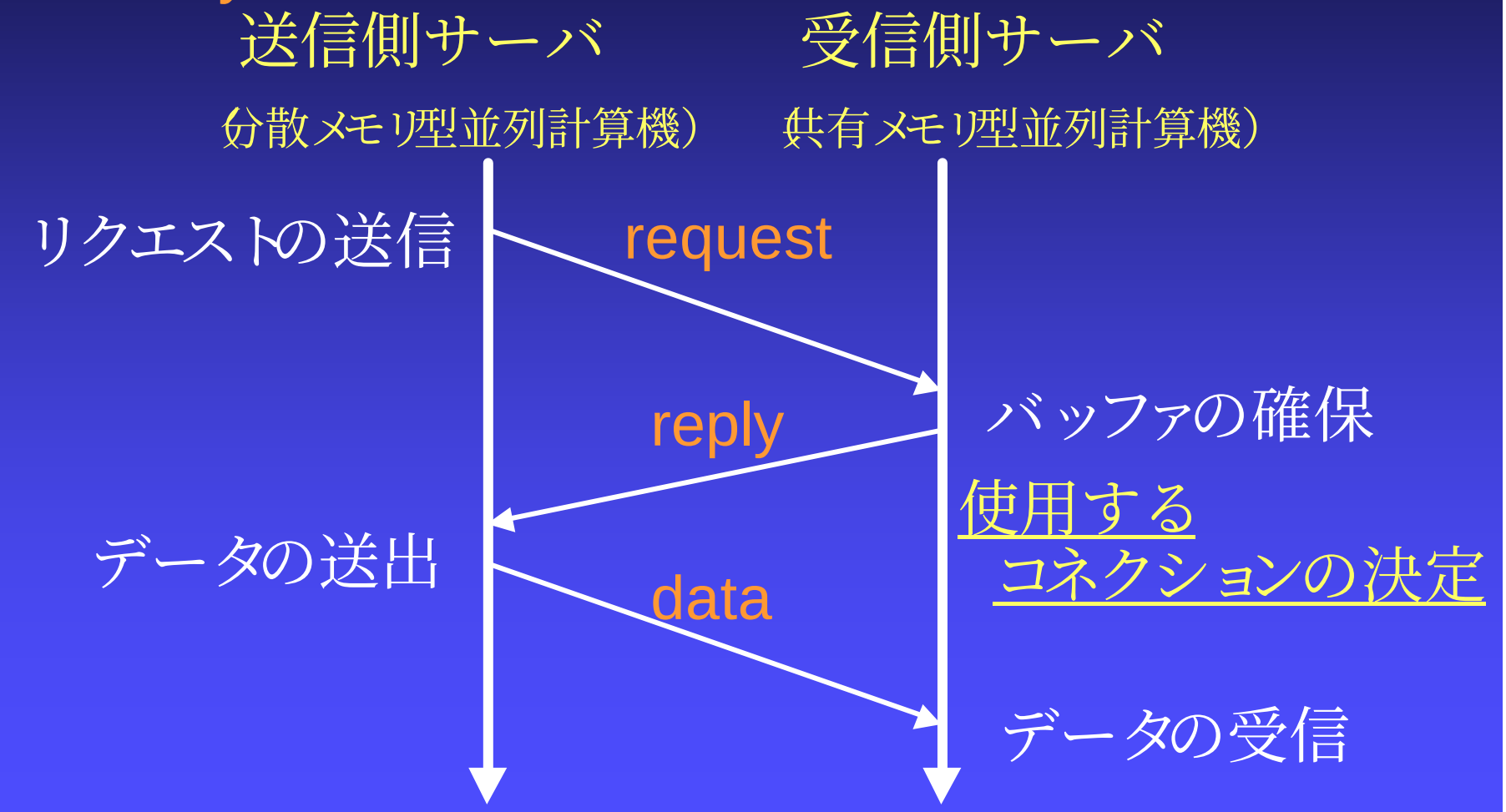
- ✓ 動的負荷分散を行うためには、データ転送毎に各ネットワークI/Fの負荷を調べる必要がある。
  - 分散メモリ型システム :複数サーバ間で負荷分散を図るのは大変。
  - 共有メモリ型システム :単一サーバなので、管理が楽。

分散メモリ型超並列計算機と共有メモリ型並列計算機が基本的な構成

フロントエンドマシン側で分散メモリ型の超並列計算機の負荷分散管理も行う

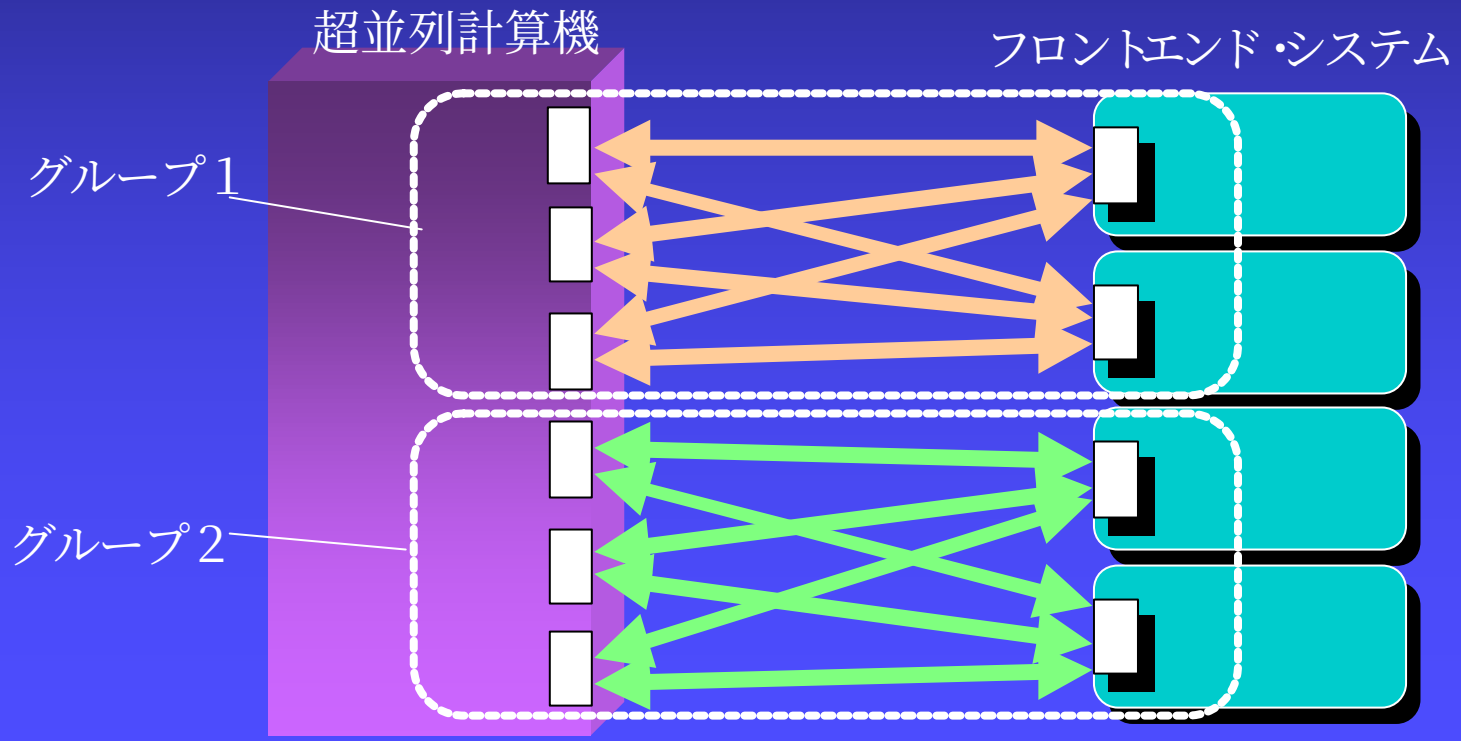
通信プロトコルの改良

- ✓ 3-wayプロトコル



動的負荷分散

- ✓ 分散メモリ型システム同士の場合はどうするか？
  - 同様にフロントエンド・マシン側で情報を管理
  - グルーピングを施し、そのグループ内でのみコネクション確立及び負荷分散情報交換を行う。



実験システム構成図

Massively Parallel Processor
CP-PACS
(2048 PUs, 128 IOUs)

8 links

Switching
HUB

100Base-TX
Ethernet
による並列接続

Parallel Visualization Server
SGI Onyx2 (4 Processors)

Alpha Cluster
HYADES (16 Nodes)

Parallel File Server
SGI Origin-2000 (8 Processors)

並列入出力システム　— PIOサーバ —

- 並列アプリケーション間（マシン間）のデータ転送を管理
- 入出力データのバッファリング
- 余分なシステム情報の隠蔽
  - 複数あるIPアドレスなどをユーザが意識する必要がない
- 複数チャネル間での負荷分散
  - アプリケーションに応じた負荷分散方法を選択できるように、複数の負荷分散機能を用意
- 到着順序保証
- 様々な構成のシステムへの適応
  - 複数のマシンをまとめて、シングルシステムと見なして利用することも可能
- 異機種マシン間での通信における、外部データ表現の変換

並列入出力システム　— API —

- 簡便で柔軟なインターフェースを提供
  - 各マシン上のPIO SystemのAPIを統一
    - プログラミング・デバッグが容易に、またプログラムの可搬性も向上
  - 余分な情報の隠蔽（必要に応じて適当な関数を呼ぶことで、各種情報を取得可能）
- 他のメッセージパッシング・ライブラリとの共存
- マルチスレッド環境下での動作保証（スレッド・セーフ）

並列入出力システム　— API —

APIとして用意する関数群

- ✓ PIOの初期化、終了
  - PIO_Init(), PIO_Exit()
- ✓ データの入出力
  - PIO_Send(), PIO_Recv()
- ✓ PIO Systemの各種情報取得
  - PIO_Getappinfo(), PIO_Gethostinfo(), PIO_Getconinfo()
- ✓ その他
  - PIO_Setpartner()

送受信関数

- PIO_Send(partner, process_ID, tag, connection, addr, datatype, count, mode)
- PIO_Recv(partner, process_ID, tag, addr, datatype, count, timeout, mode, status)
  - Partner : 通信相手となるアプリケーションの識別子
  - Process_ID : 並列アプリケーション内のプロセス識別子
  - Connection : 負荷分散方式の指定。ユーザによる負荷分散の場合はさらにどのコネクションを使用するかを示す

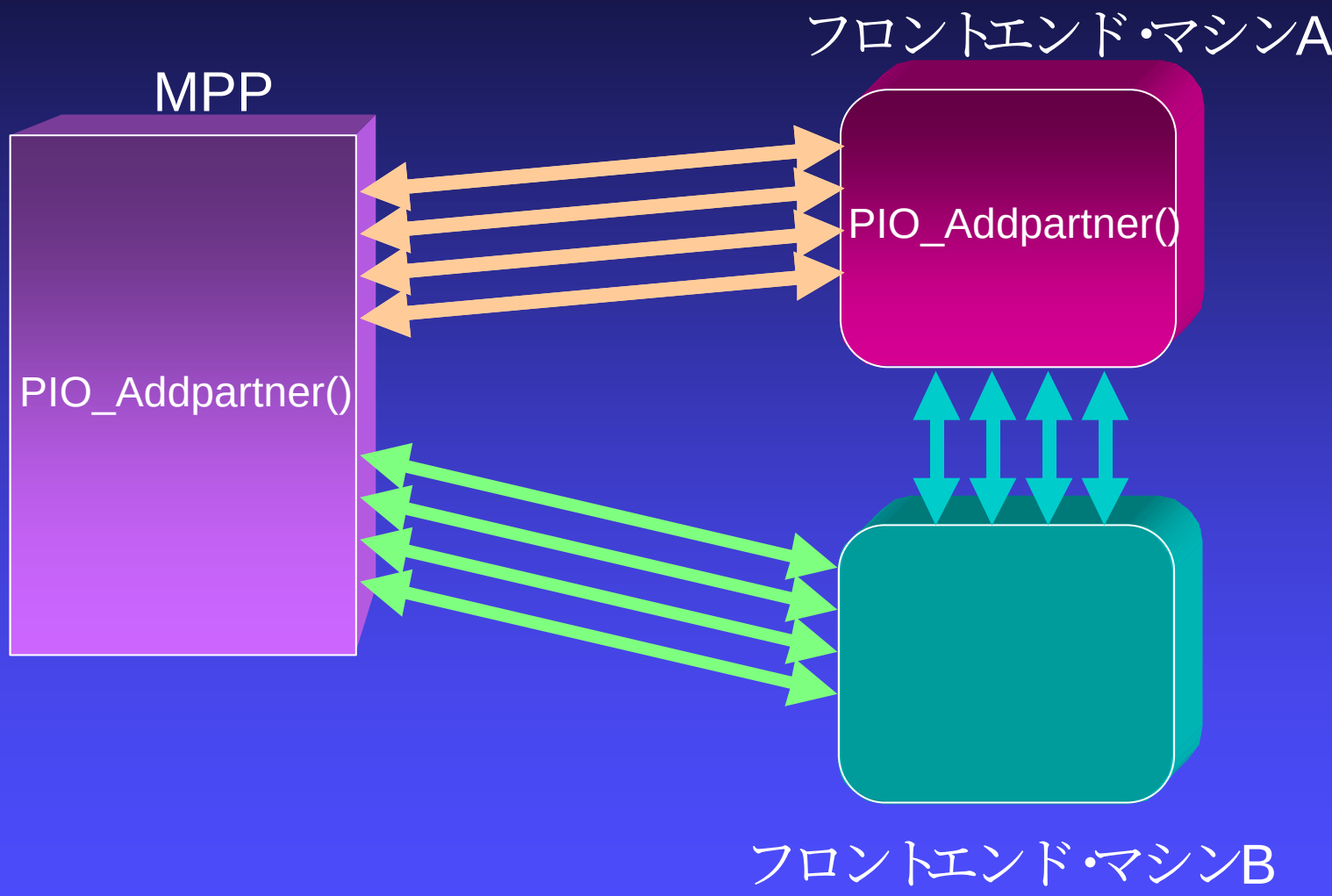
- ユーザは通信相手となるプロセスを指定すれば良い
  - システムによる負荷分散方式を指定することにより、並列ネットワークを意識することなくプログラミング可能

並列入出力システム　— API —

APIとして用意する関数群

- ✓ PIOの初期化、終了
  - PIO_Init(), PIO_Exit()
- ✓ データの入出力
  - PIO_Send(), PIO_Recv()
- ✓ PIO Systemの各種情報取得
  - PIO_Getappinfo(), PIO_Gethostinfo(), PIO_Getconinfo()
- ✓ その他
  - PIO_Addpartner()

3者間以上での利用


フロントエンド・マシンA
MPP
PIO_Addpartner()
PIO_Addpartner()
フロントエンド・マシンB

並列入出力システム　— API —

- 簡便で柔軟なインターフェースを提供
  - 各マシン上のPIO SystemのAPIを統一
    - プログラミング・デバッグが容易に、またプログラムの可搬性も向上
  - 余分な情報の隠蔽（必要に応じて適当な関数を呼ぶことで、各種情報を取得可能）
- 他のメッセージパッシング・ライブラリとの共存
- マルチスレッド環境下での動作保証（スレッド・セーフ）

各マシンに特化したチューニング

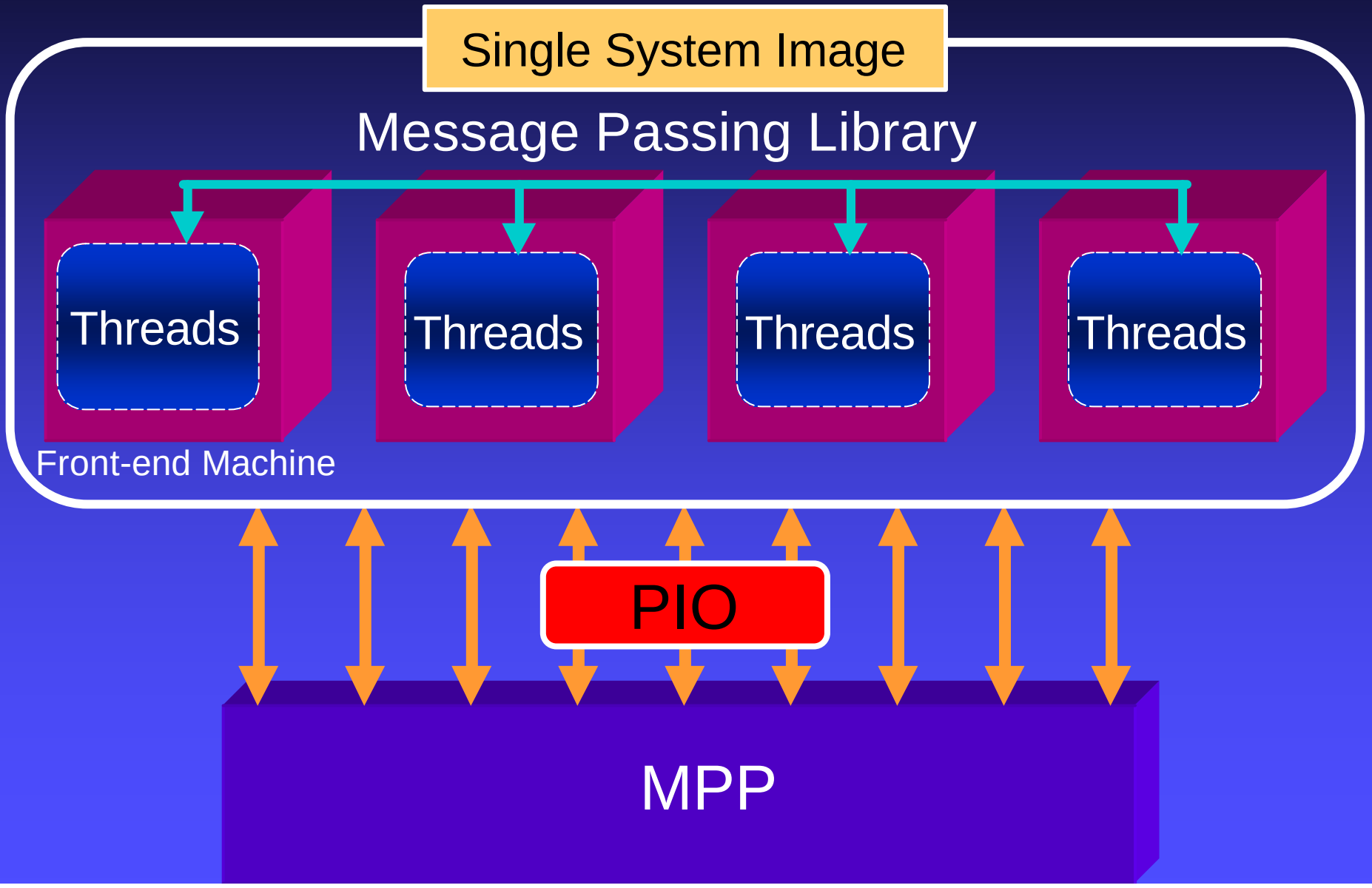
- ✓ 超並列計算機CP-PACS
  - 高速通信機構（リモートDMA転送）を用いた高速なデータ転送（PIOサーバ、ユーザ・プロセス間）
- ✓ 共有メモリ型並列計算機Origin-2000, Onyx2
  - マルチスレッド化
  - 共有メモリを介したデータ転送（PIOサーバ、ユーザ・プロセス間）

PIOシステムのまとめ

- ✓ 大規模データ転送アプリケーションをターゲット
  - レイテンシよりもスループット重視
- ✓ 並列システム間での利用を想定
  - ファイル転送以外にも様々なアプリケーションに適用可能
- ✓ サーバとAPIから構成
  - 簡便なAPI ⇒ ユーザに並列ネットワークを意識させない
  - サーバによる仲介 ⇒ 並列ネットワークの効率的な利用
- ✓ 多種多様なシステム上で利用可能

  ← サーバ、APIによるサポート
  - システムの移植性、汎用性が高い
  - ネットワークのみならず、フロントエンド・システムのスケーラビリティを確保することができる

PIOを利用したフロントエンド・マシン環境



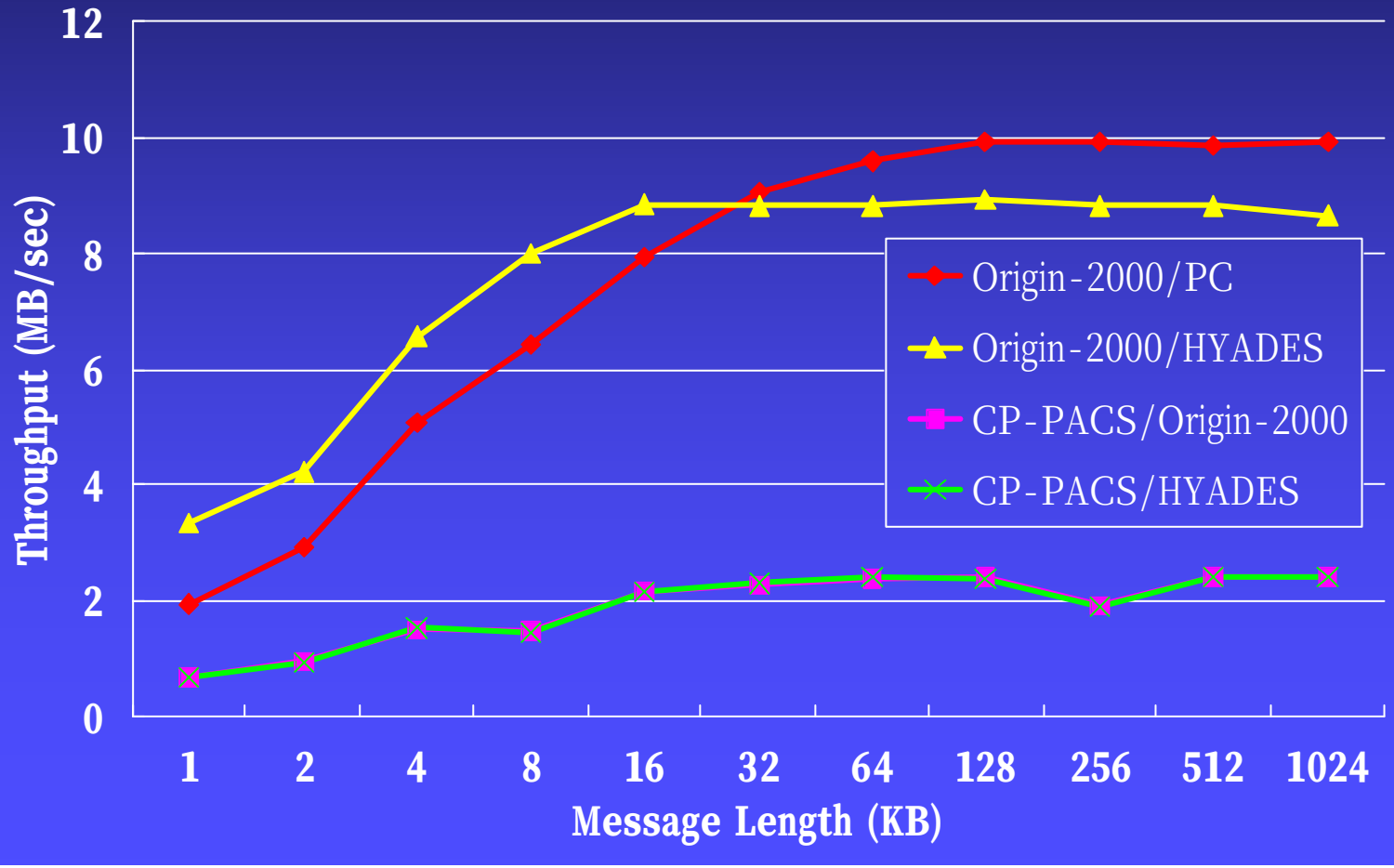
PIOシステムの性能評価

- ✓ CP-PACS, Origin-2000, HYADESを用いて測定
  - Ping-pong転送実験
    - 本システムのスケーラビリティ
  - 一方向転送実験
    - 実効性能
  - 転送データ量に偏りのある場合の実験
    - 負荷分散機能
  - リモートファイル転送実験

各マシンのTCP/IP性能評価

○ ping-pong転送プログラムによる測定

CP-PACS 2.4MB/sec,　Origin-2000 9.9MB/sec
HYADES 8.8MB/sec,　PC　9.9MB/sec

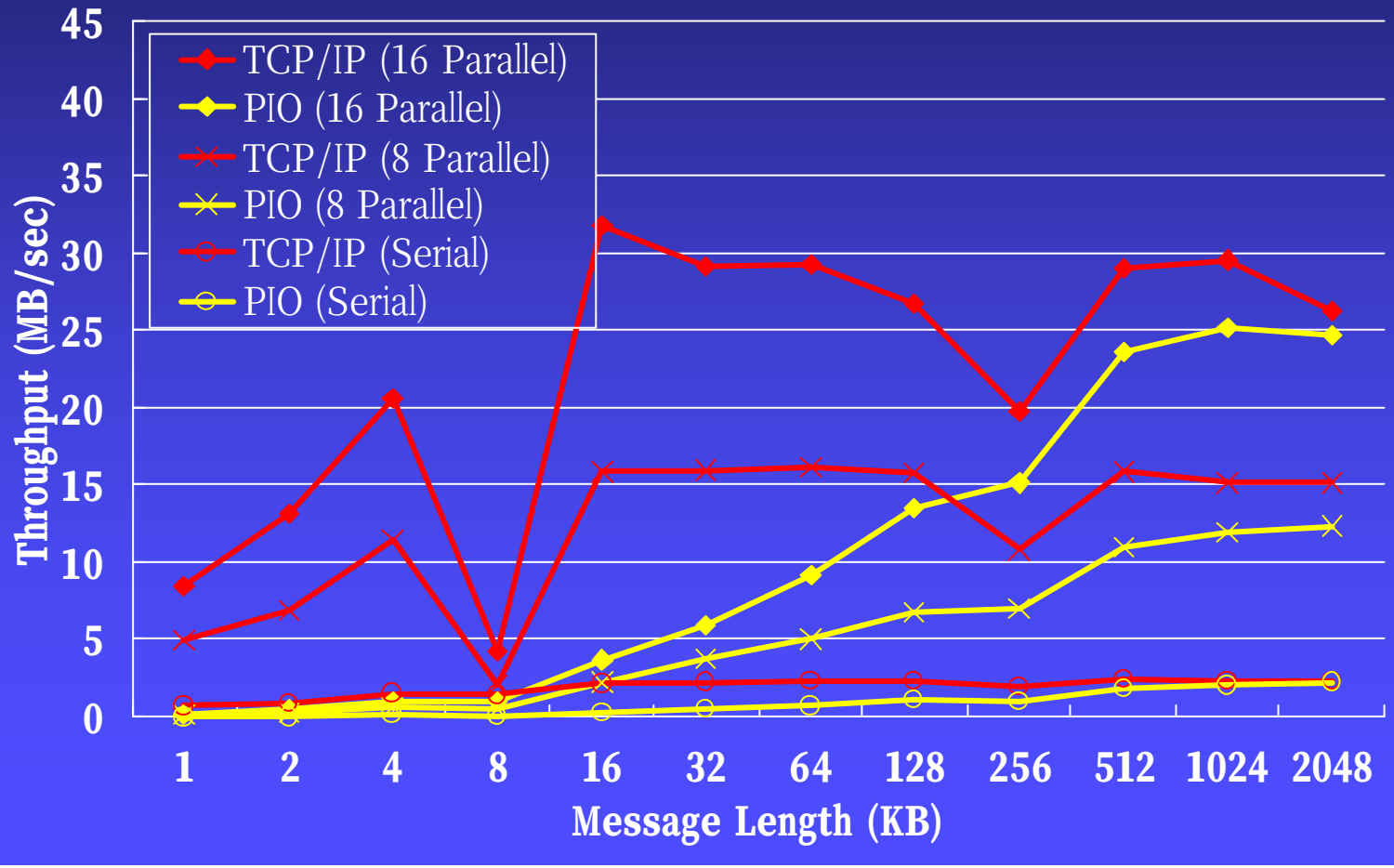


PIOの性能評価（1） (ping-pong転送)

○ CP-PACS, Origin-2000間での ping-pong転送実験 (1- 16 channel)

PIO - PIO Systemを利用したプログラム

TCP/IP - 通常のソケット通信関数を用いたプログラム

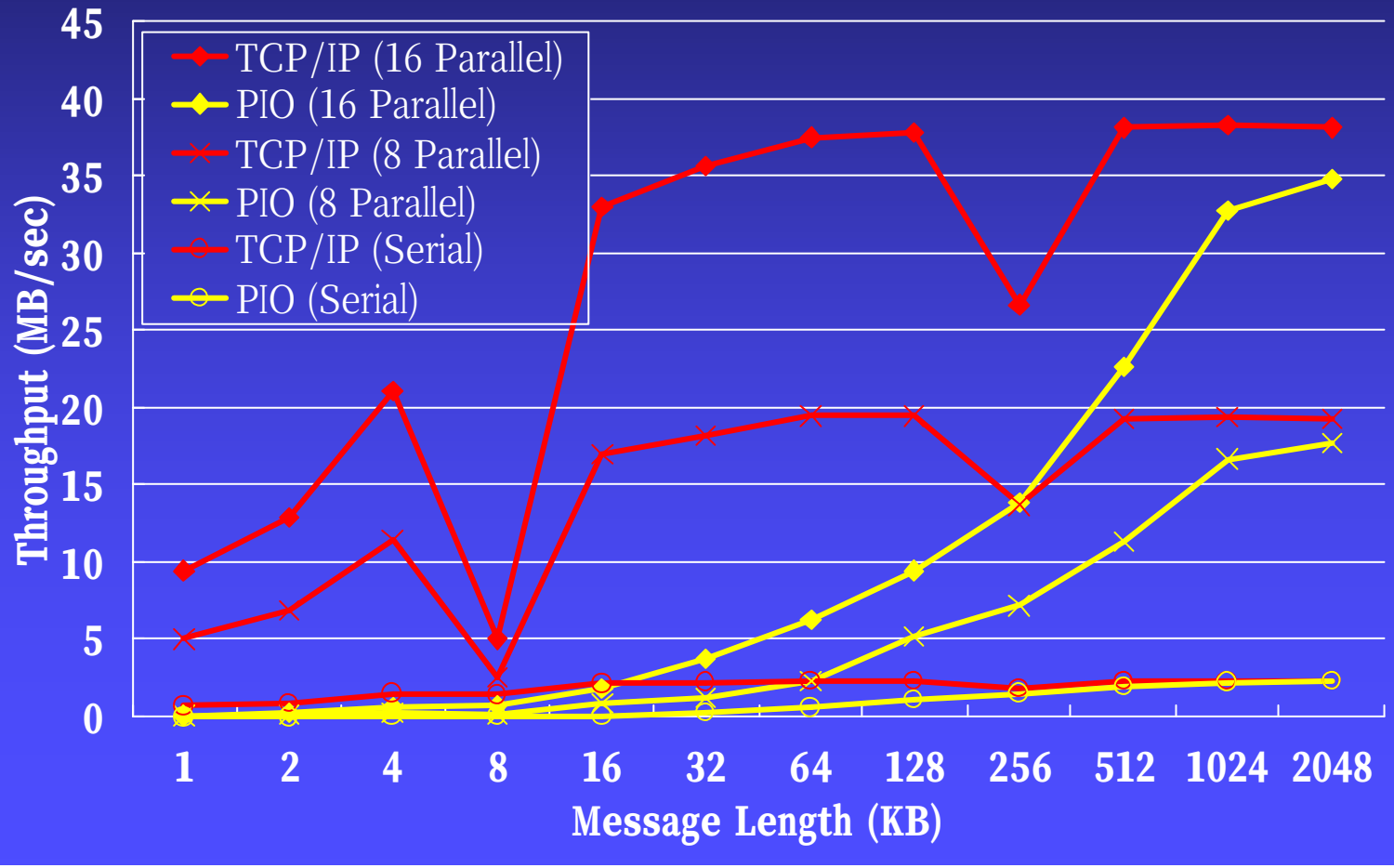


PIOの性能評価（1） (ping-pong転送)

○ CP-PACS, HYADES間での ping-pong転送実験 (1- 16 channel)

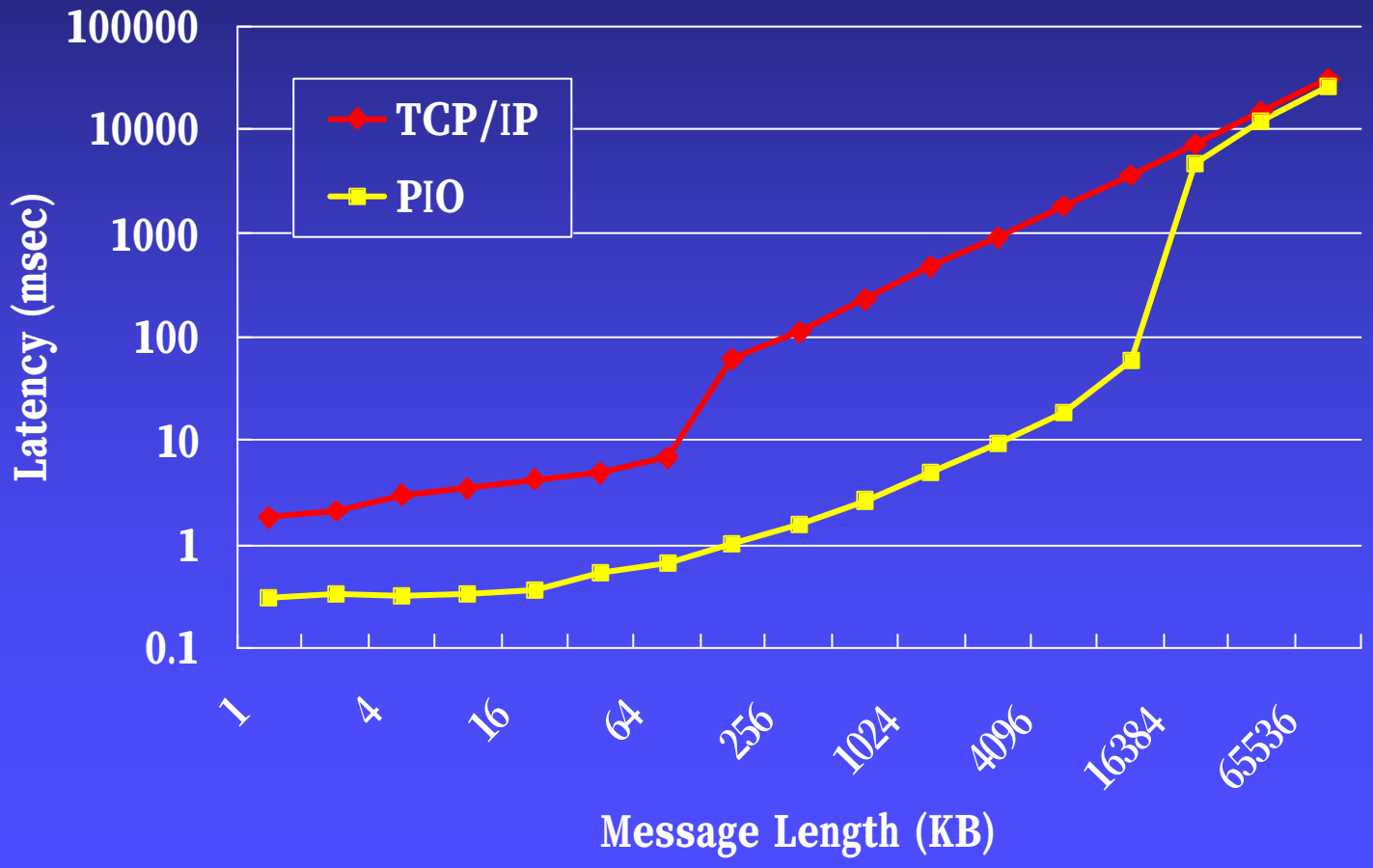
PIO - PIO Systemを利用したプログラム

TCP/IP - 通常のソケット通信関数を用いたプログラム

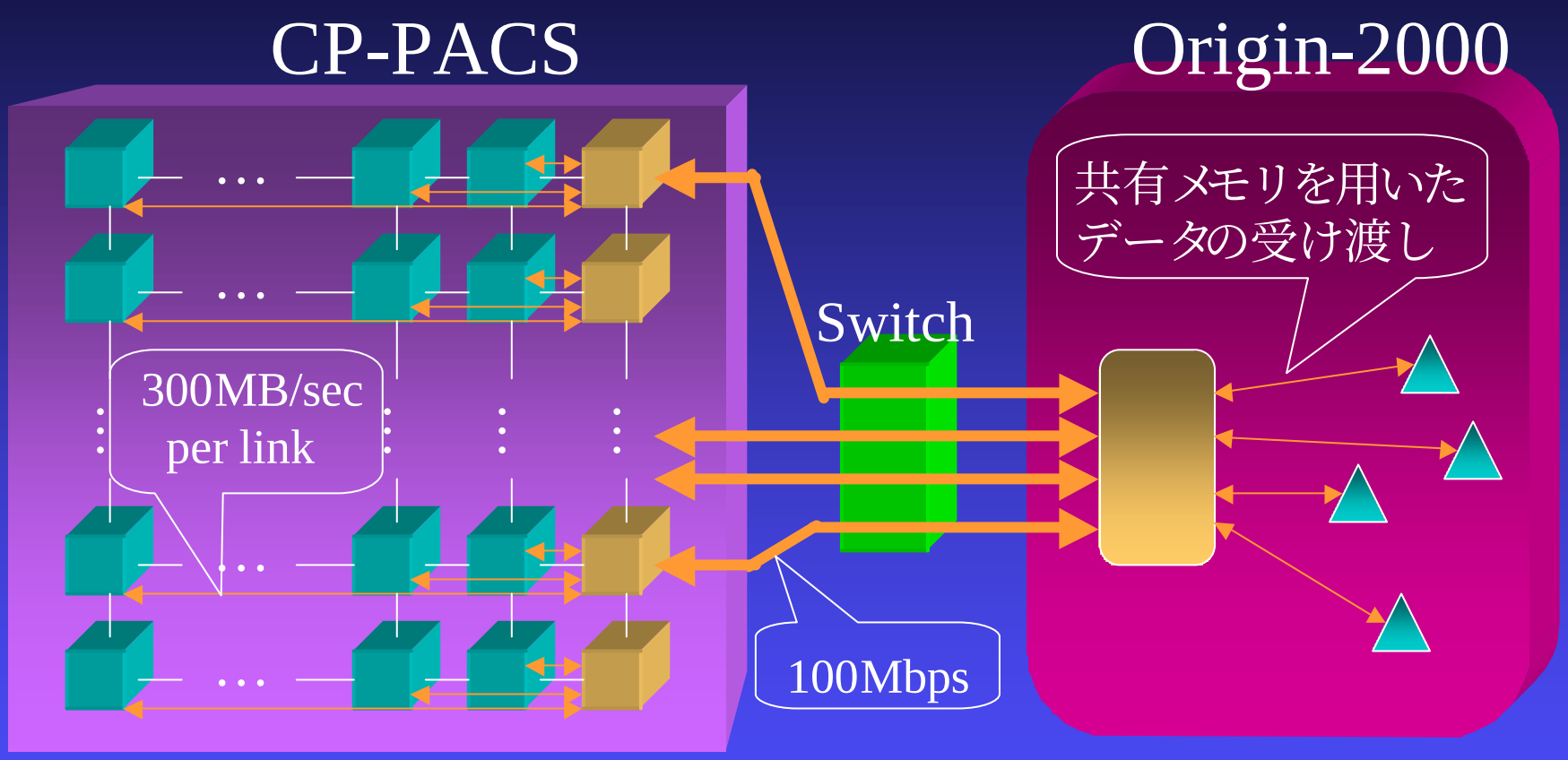


PIO の性能評価 (2) (一方向転送)

○ CP-PACSからOrigin-2000への一方向転送実験 (1 channel)
CP-PACS側のアプリケーションが、送信処理を開始してから次の処理に移れるようになるまでの時間を測定。

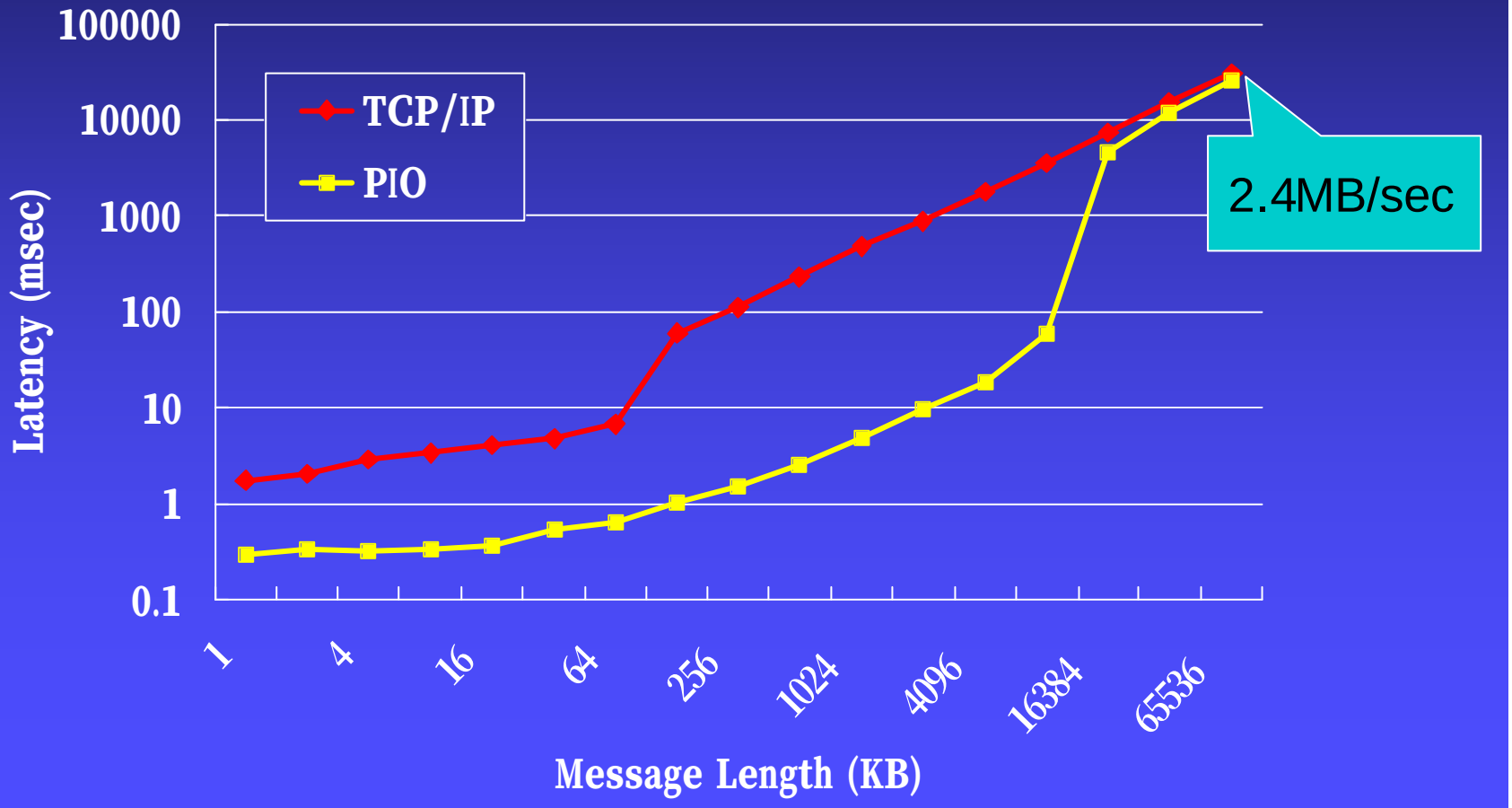


並列入出力システムの構成



PIO の性能評価 (2) (一方向転送)

○ CP-PACSからOrigin-2000への一方向転送実験 (1 channel)
CP-PACS側のアプリケーションが、送信処理を開始してから次の処理に移れるようになるまでの時間を測定。

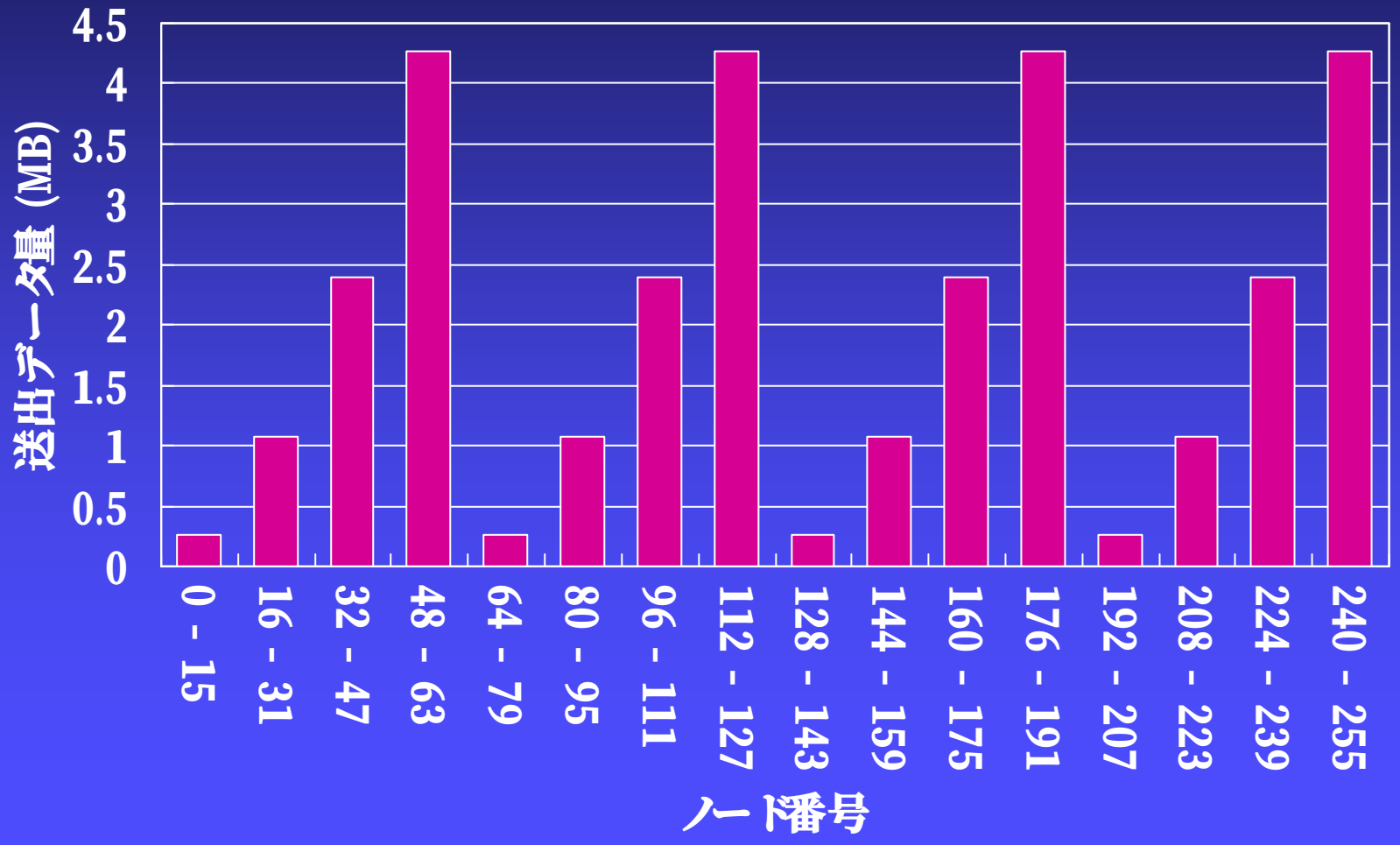


PIO の性能評価（3） 負荷分散）

- CP-PACS(256プロセス）からOrigin-2000（1プロセス（8スレッド））への総量512MBのデータ転送。
- 3通りの負荷分散方式
  - USER - ユーザによる負荷分散
  - STATIC - システムによる静的負荷分散
  - DYNAMIC - システムによる動的負荷分散
- 計16コネクションを使用し、各コネクションのデータ流量に偏りが生じるようにする。
  - 最大で16倍の差
- STATICの場合には意図的にうまく負荷分散が行えないよう、コネクションの張り方を設定している。

PIO の性能評価（3） 負荷分散）

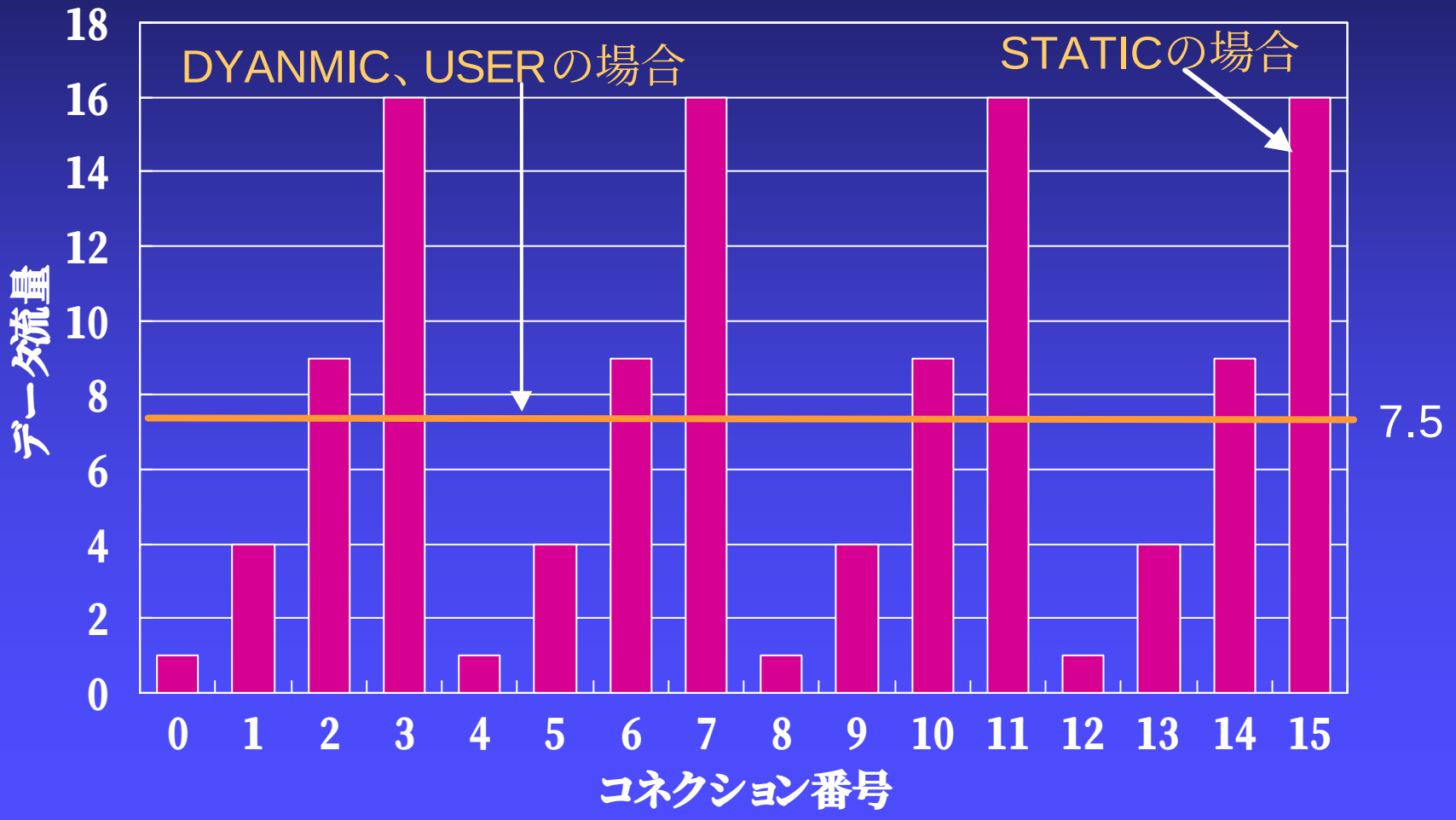
☆16ノードを1グループとして、各グループがそれぞれ異なるコネクションを通してデータを送信。



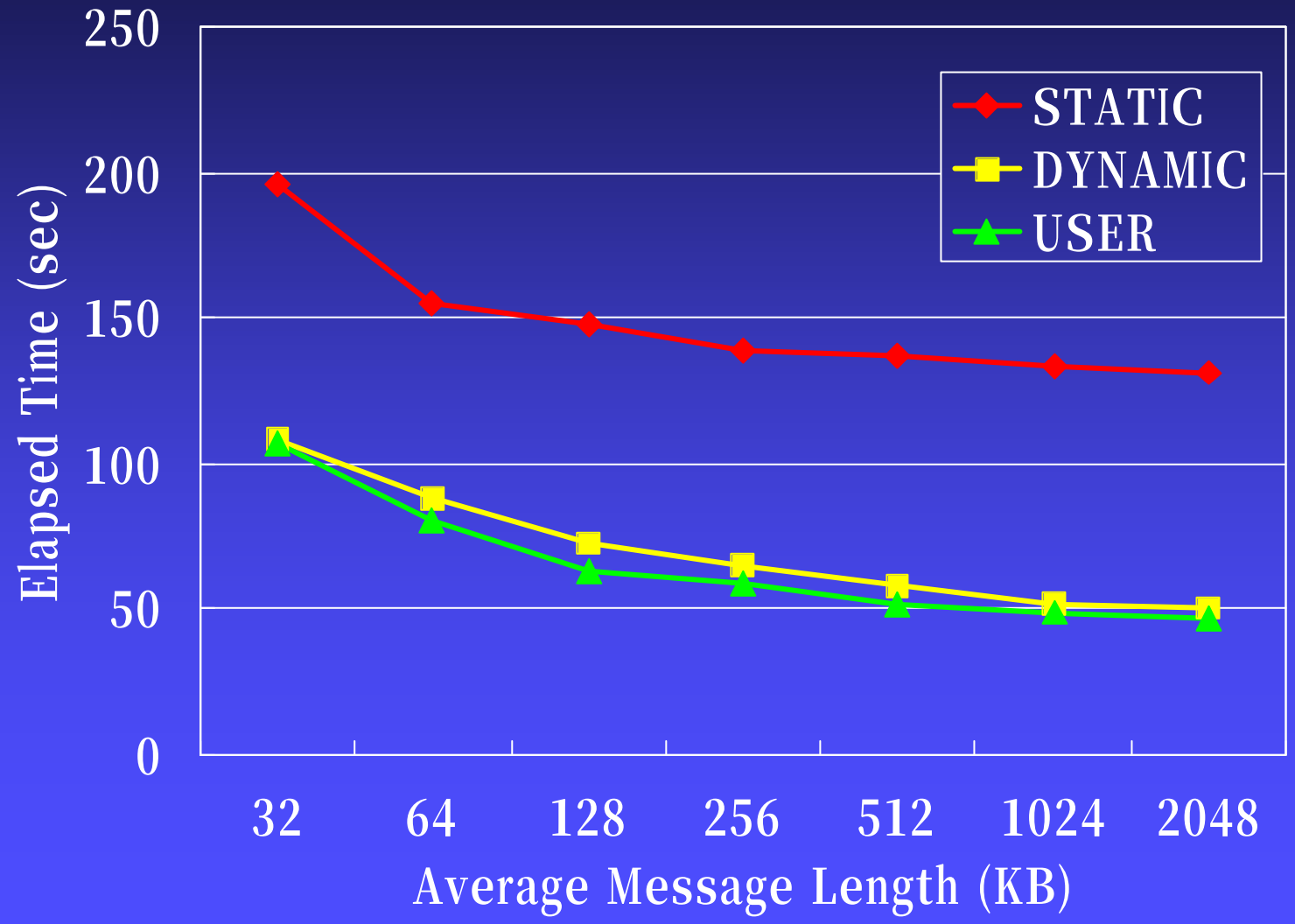
PIO の性能評価（3） 負荷分散）

STATICでの実行時間 :DYNAMICでの実行時間

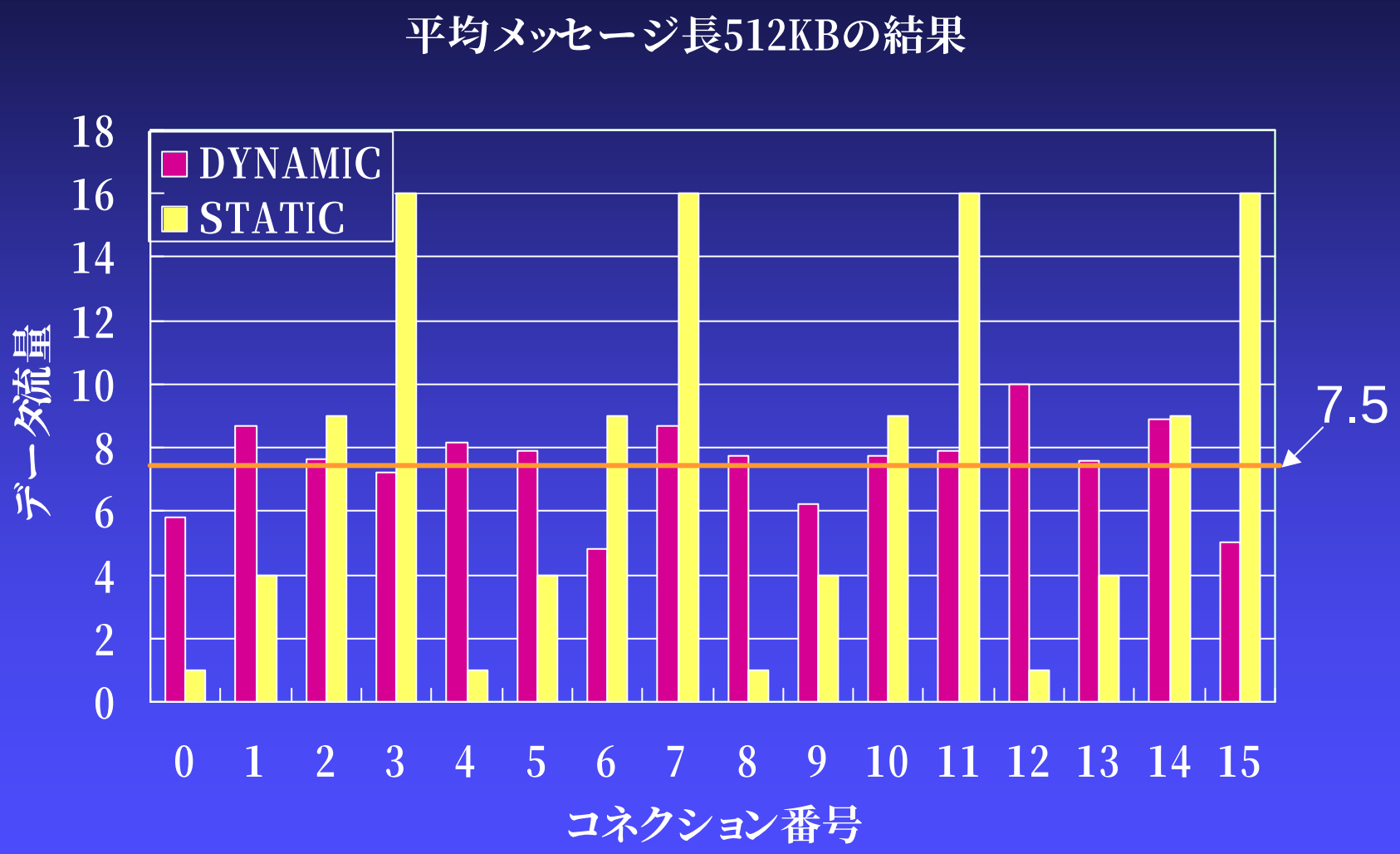
＝16 ：7.5 ＝ 2.13 ：1



PIO の性能評価 (3) 負荷分散)



PIO の性能評価（3） 負荷分散）



PIO の性能評価 (4)　(リモートファイル出力)

○ CP-PACSからOrigin-2000へのリモートファイル出力実験

4 PU で各PUが1MBのデータブロックを各々10回出力したものをリモートシステム上で1つのファイルにマージ

data size = 1(MB) × 10 (block) × 4 (PU) = 40MB

◎アプリケーションからの直接データ出力

NFS経由(1 channel): 63.39 Sec = 0.58 MB/sec

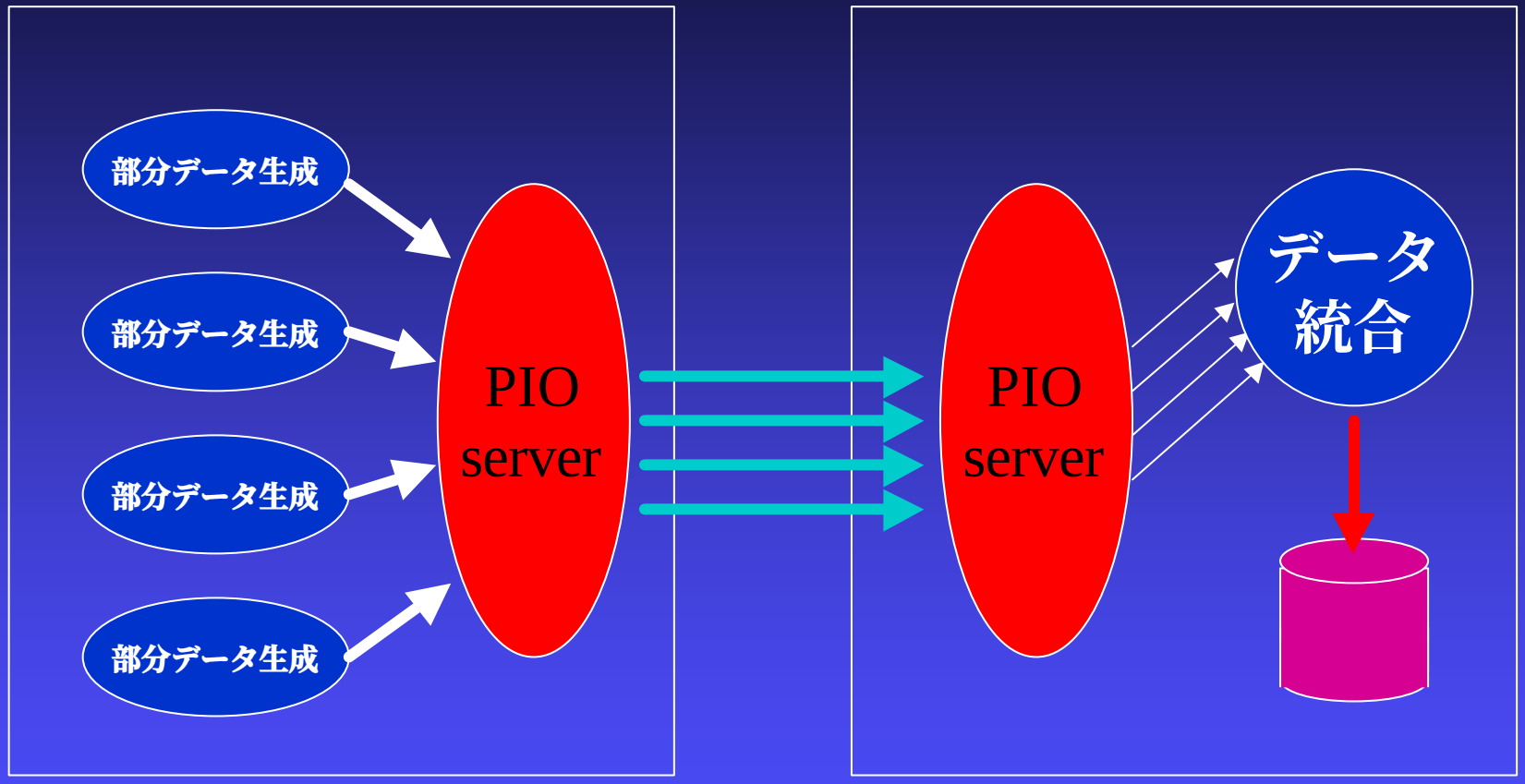
PIO経由(4 channel) : *5.95 Sec = 6.72 MB/sec*

◎CP-PACSのローカルファイルをOrigin-2000にコピー

rcpによる転送(1 channel): 22.43 sec = 1.78 MB/sec

PIOによる転送(4 channel): *7.34 sec = 5.45 MB/sec*

リモートファイル出力の流れ



PIO の性能評価 (4)　(リモートファイル出力)

○ CP-PACSからOrigin-2000へのリモートファイル出力実験

4 PU で各PUが1MBのデータブロックを各々10回出力したものをリモートシステム上で1つのファイルにマージ

data size = 1(MB) × 10 (block) × 4 (PU) = 40MB

◎アプリケーションからの直接データ出力

NFS経由(1 channel): 63.39 Sec = 0.58 MB/sec

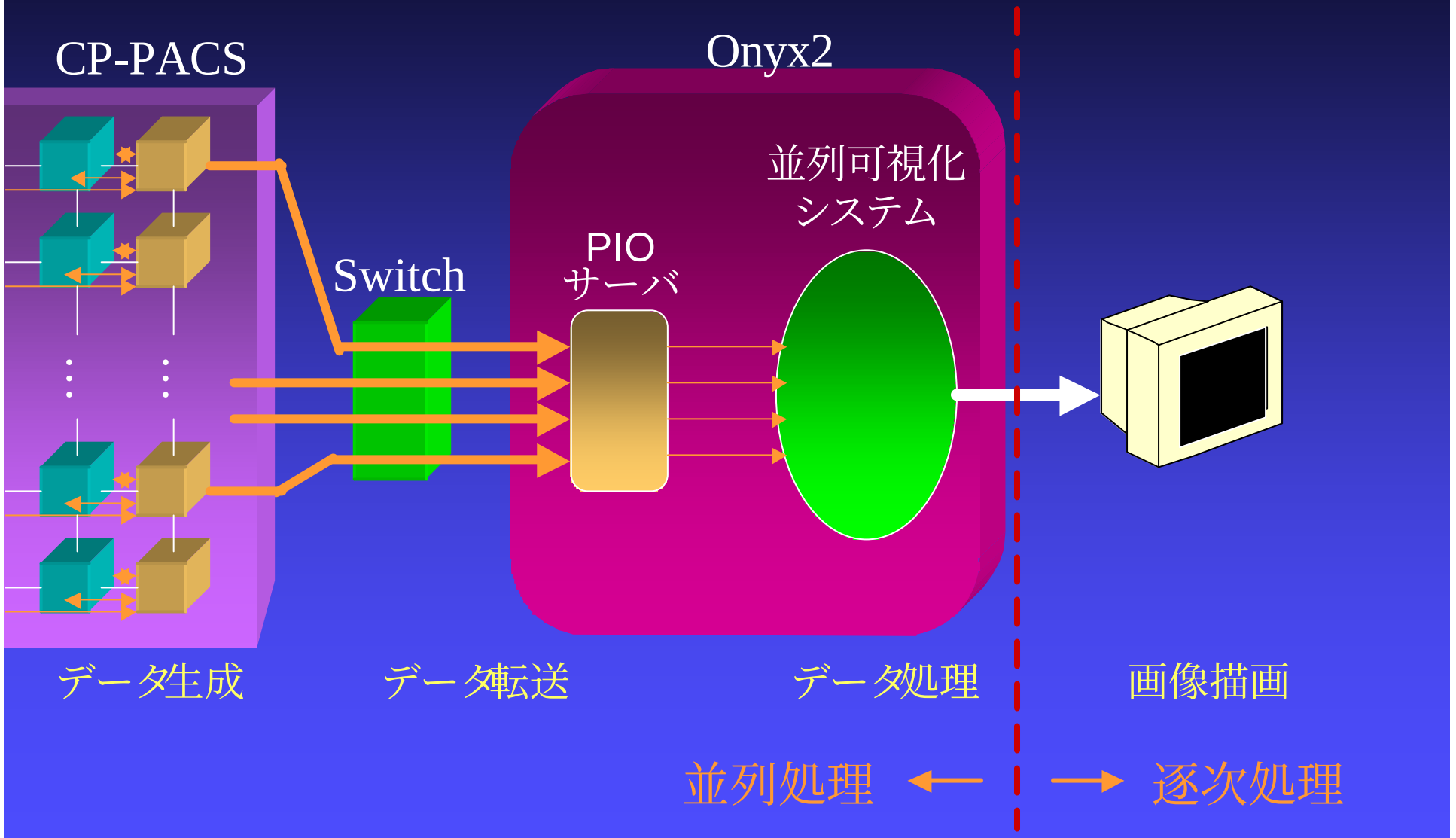
PIO経由(4 channel) : *5.95 Sec = 6.72 MB/sec*

◎CP-PACSのローカルファイルをOrigin-2000にコピー

rcpによる転送(1 channel): 22.43 sec = 1.78 MB/sec

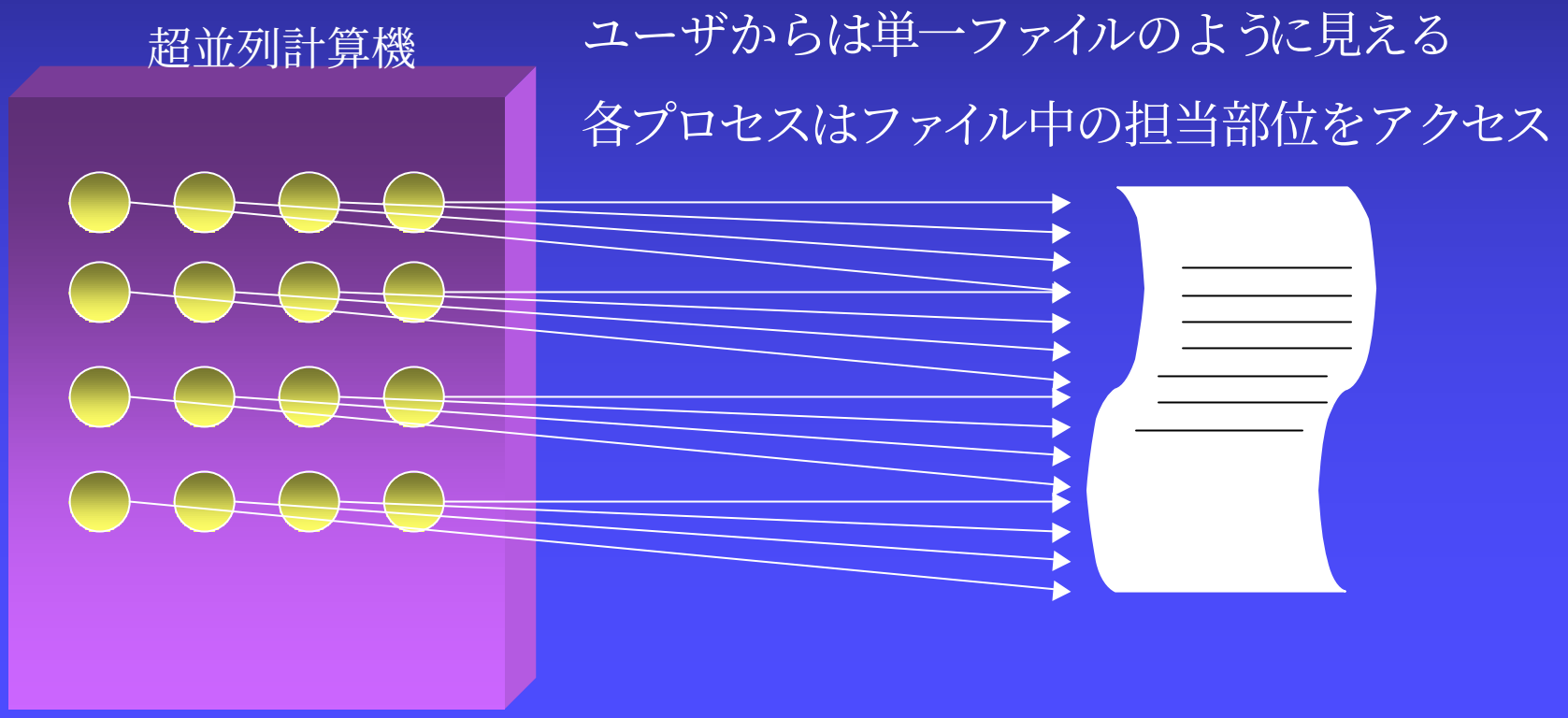
PIOによる転送(4 channel): *7.34 sec = 5.45 MB/sec*

PIOの実用例（1）－並列可視化システム－



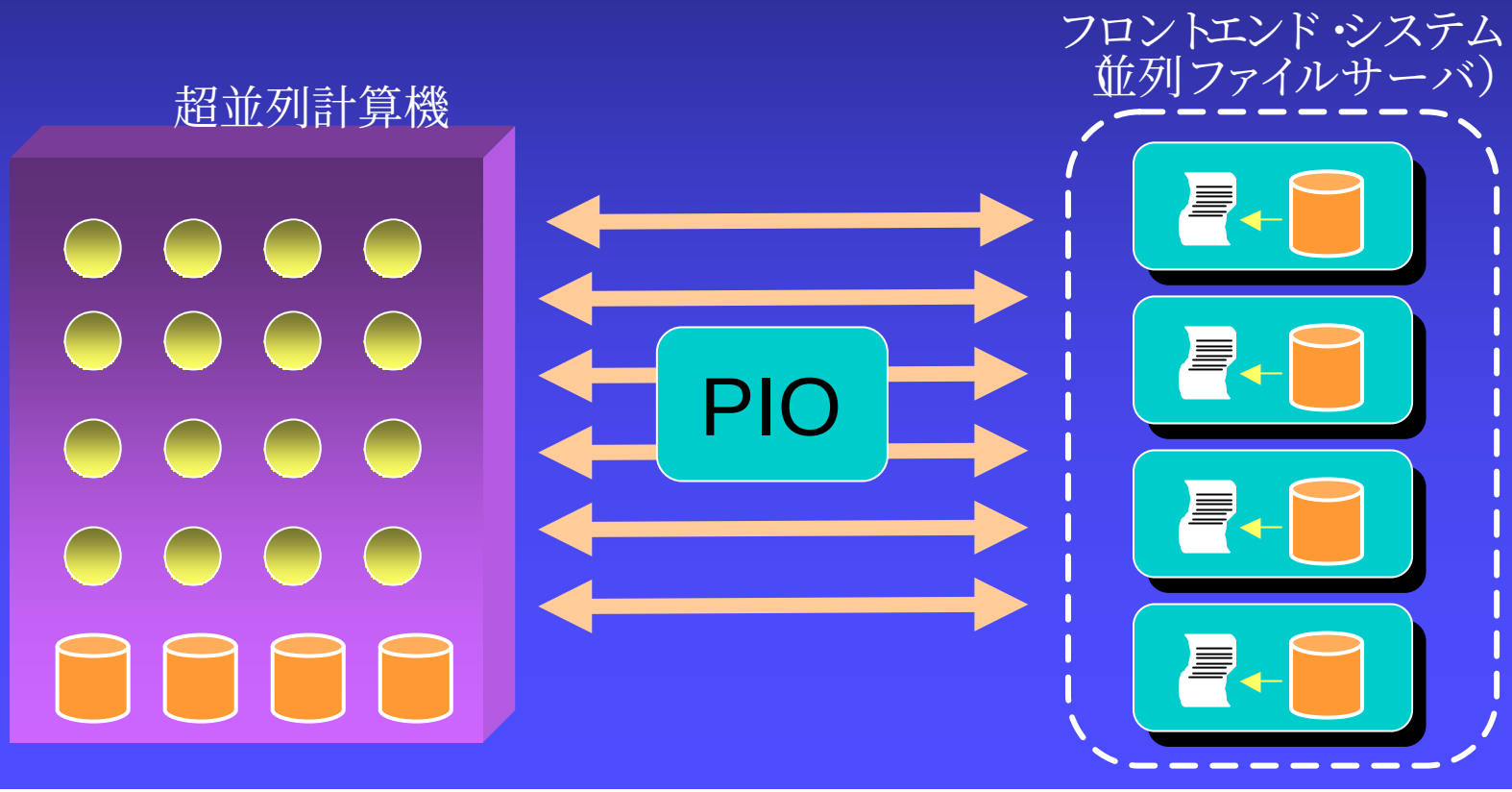
PIOの実用例(2) ―並列ファイルシステム―

- ✓ ユーザに物理的なファイル構造ではなく、より簡単な論理的構造を見せるインタフェースを提供
  - ● 物理⇔論理ファイル構造のマッピングが必要



PIOの実用例(2) ―並列ファイルシステム―

- ✓ 物理的には複数のストレージに分散配置（ローカルorリモート）
  - PIOがデータ転送の最適化を行う（必要ならばデータ変換も）ので、並列ファイルシステムは上記マッピング問題にのみ集中



今後の課題

- PIO Systemを利用したアプリケーションの開発
  - 並列ファイルシステム等
- MPI等の標準APIへの適用
- 入出力機構のチューニング

まとめ

- ✓ 超並列計算機の外部に対する入出力機構を検討した結果、並列ネットワークとマルチプロセッサ・システムというハードウェア構成を選択した。
- ✓ commodity化しているネットワークを複数本束ねることにより、安価で高性能な並列ネットワークが構築可能であることを示した。
- ✓ 超並列計算機向けの並列入出力システムを作成した。
  - 並列ネットワークを効率良く利用するためのサーバを用意。
  - 並列アプリケーション間での利用を想定した、簡便で柔軟なAPIを提供。
- ✓ データ生成からデータ処理に至るまですべて並列に処理可能な並列入出力システムを構築した。