コンピュータシステム（第3回）

坂井　修一
東京大学大学院 情報理工学系研究科 電子情報学専攻

- はじめに
- 並列計算機の分類
- 共有メモリ型並列計算機
- コヒーレントキャッシュ
- スヌープバスによる共有メモリ型並列計算機
- スケーラブルな共有メモリ型並列計算機
- PCクラスタ、並列スパコン

はじめに

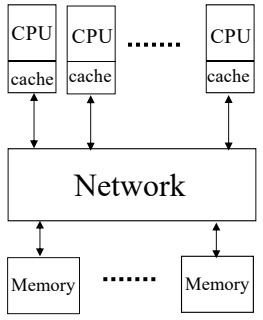
- 講義内容：　コンピュータシステム
  - 並列処理、再構成、OS、セキュリティ、スーパコンピュータ、コンピュータの未来
- 関連講義（電子・情報系）：
  - 論理回路基礎：入江、２年冬
  - コンピュータハードウェア：坂井、３年夏
  - アドバンスト・コンピュータアーキテクチャ：入江、大学院奇数年度夏
- 成績
  - 出席＋レポート

並列計算機アーキテクチャの構成上の分類

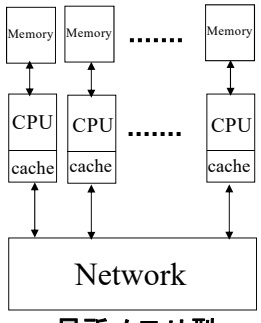
- 並列計算機
  - ＳＩＭＤ†
    - 細粒度：　CM-2, ICL-DAP
    - 粗粒度：　GF-11, Illiac-IV
  - ＭＩＭＤ‡
    - 大域メモリ型
      - バス結合型:　SUN Enterprize3000, Alliant
      - スイッチ結合型：VPP500, Cedar
    - 局所メモリ型：T3E, SP-2, SR2201, Cenju-4,　AP-1000, EM-4, PIM/m, J-Machine, RWC-1
    - 階層型（上位：局所、下位：大域）：SR8000, Origin2000, PIM/p,　DASH/FLASH,　Alewife,　Jump-1

† Single Instruction Stream Multiple Data Stream: 一個の命令を同時に複数のプロセッサで処理する。Data Parallelもこの範疇に入れることが多い。柔軟性はないが、問題を限れば有効。
‡ Multiple Instruction Stream Multiple Data Stream: 複数の命令を複数のプロセッサで処理する。

大域メモリ型と局所メモリ型

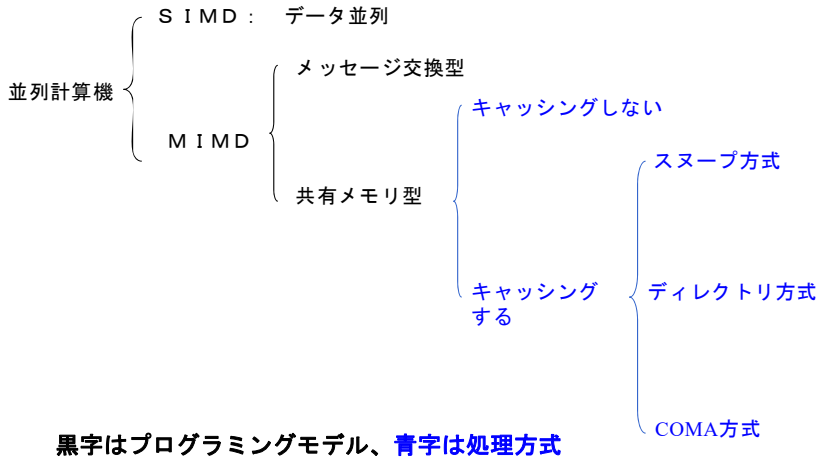


大域メモリ型



局所メモリ型

並列計算機のプログラミングモデル～処理方式上の分類



黒字はプログラミングモデル、青字は処理方式

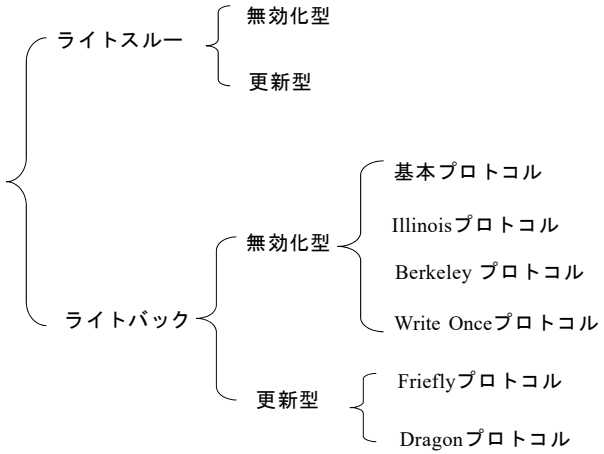
4. スヌープバスによる共有メモリ型並列計算機

- 「共有メモリ型並列計算機」の定義
  - 「各プロセッサで走るプロセスが互いに共有メモリを読み書きすることでデータ交換と同期を行う」というプログラミングモデルを（効率よく）実現する並列処理計算機
- 局所コピー
  - 目的
    1. メモリレーテンシ削減
       ※ レーテンシ：メモリ読み出しのターンアラウンド時間
    2. メモリアクセス競合の緩和
  - 方式
    - プログラムによる陽なコピー
    - コヒーレントキャッシュ

コヒーレントキャッシュ

- 定義
  - 複数のキャッシュ間でデータの一致(coherence)を保証するためのハードウェア機構をもったキャッシュシステム
- 利点
  - ユーザプログラムに意識させることなく(transparently)、データの局所コピーが実現される
    - レーテンシ削減＋アクセス競合緩和
    - 場合によってはプログラマが管理
- コヒーレンスの制御
  - 機構：　スヌープバス（小規模）、ディレクトリ（中・大規模）
  - コヒーレンスプロトコル：ライトスルー、ライトバック

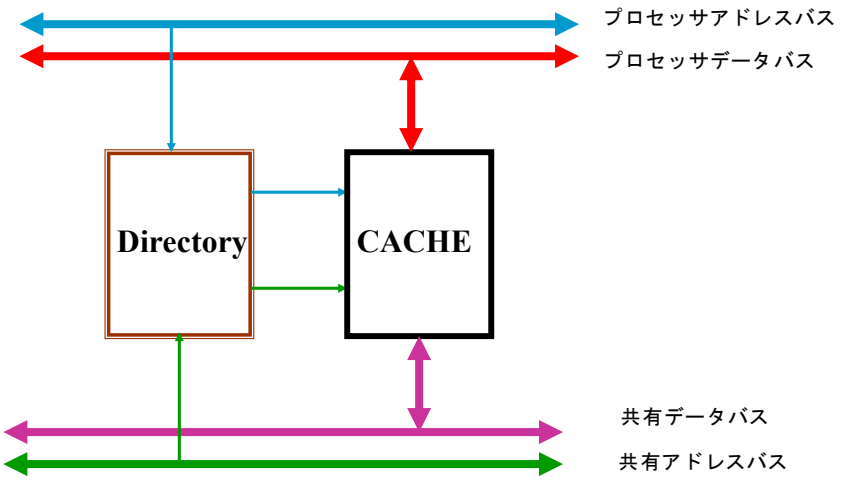
コヒーレンスプロトコル



スヌープバスによる共有メモリ型計算機

- スヌープバス方式
  - バスで結合されたプロセッサがそれぞれ常時バスを監視して、データのコヒーレンスを保証する方式
- スヌープバスの分類
  - 「時期」による分類
    - ライトスルー：I (Invalid), V (Valid)の2状態
      - データ書込のたびに主記憶を書き込むことで内容の一致をとる
    - ライトバック：3状態～4状態（以上）
      - データ書込時にも主記憶との一致をとらなくてよい
  - 「方法」による分類
    - 無効化型
      - 他のプロセッサが書込んだら当該キャッシュラインを無効にする
    - 更新型
      - 他のプロセッサが書込んだら当該キャッシュラインをその値にする

スヌープキャッシュの構造



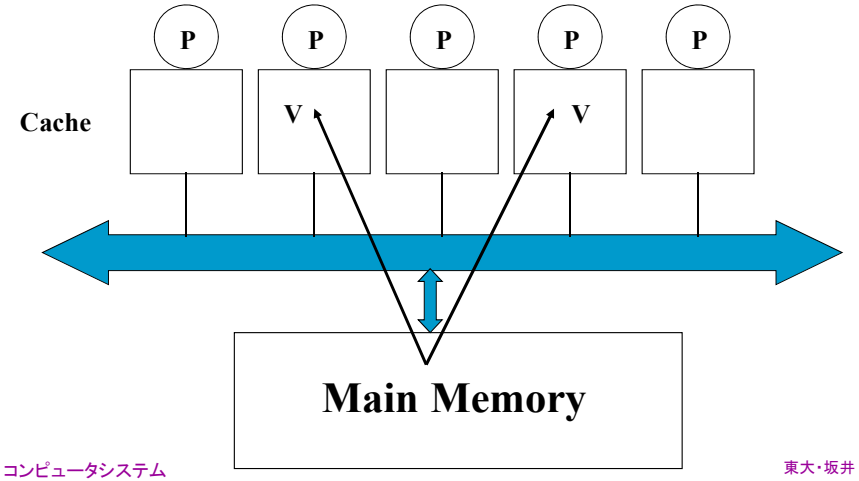
スヌープキャッシュの例

1. 無効化型ライトスルー
2. 更新型ライトスルー
3. 無効化型ライトバック
   - 基本
   - Illinois
   - Berkeley
4. 更新化型ライトバック
   - Friefly
   - Dragon

無効化型ライトスルー (1/3)

(1) 読み出し

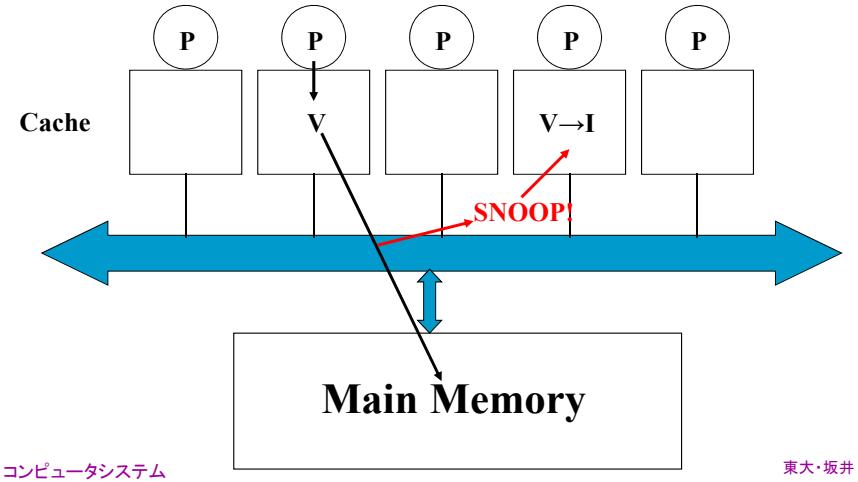
V: Valid, I: Invalid



無効化型ライトスルー (2/3)

(2) 書き込み
（自キャッシュにデータあり）

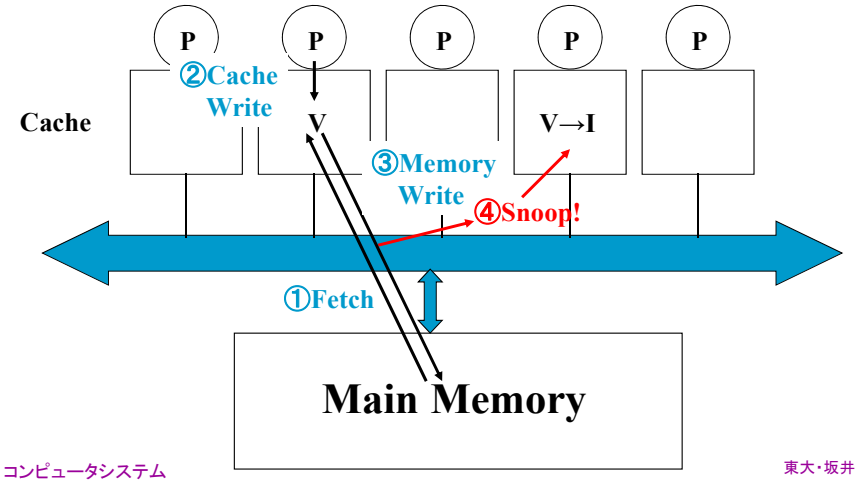
V: Valid, I: Invalid



無効化型ライトスルー (3/3)

(3) 書き込み
（自キャッシュにデータがない）

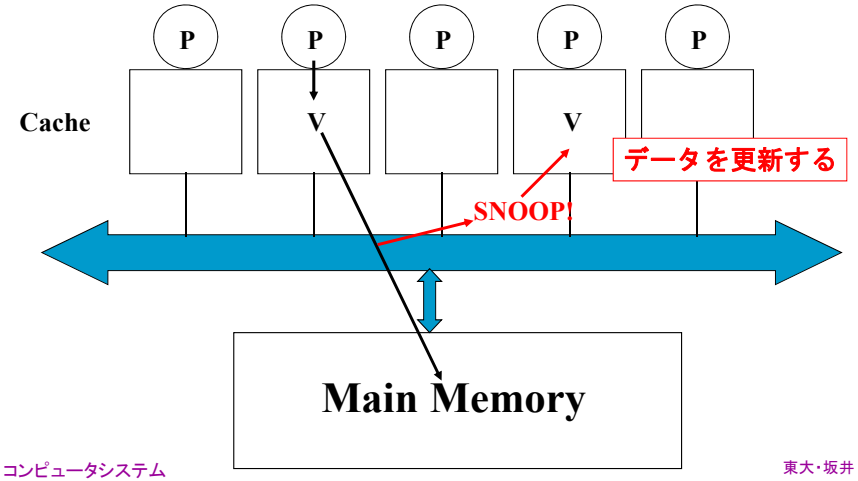
V: Valid, I: Invalid



更新型ライトスルー (1/2)

(1) 書き込み
（自キャッシュにデータあり）

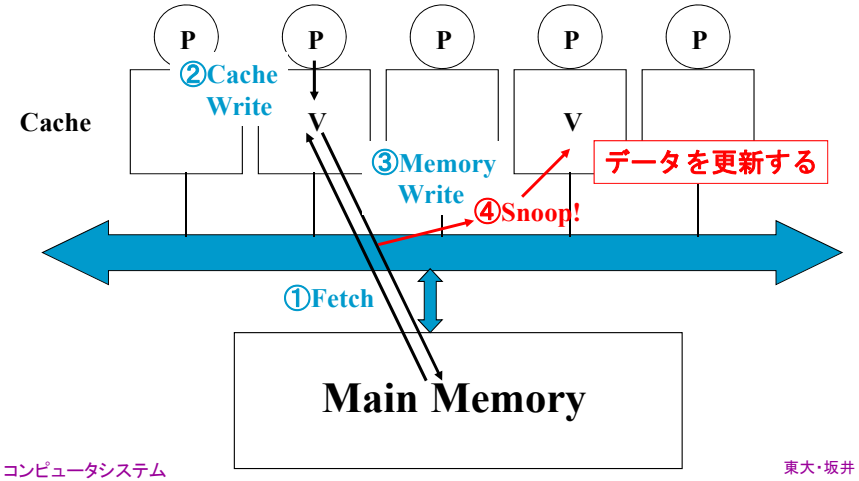
V: Valid, I: Invalid



更新型ライトスルー (2/2)

(2) 書き込み
（自キャッシュにデータがない）

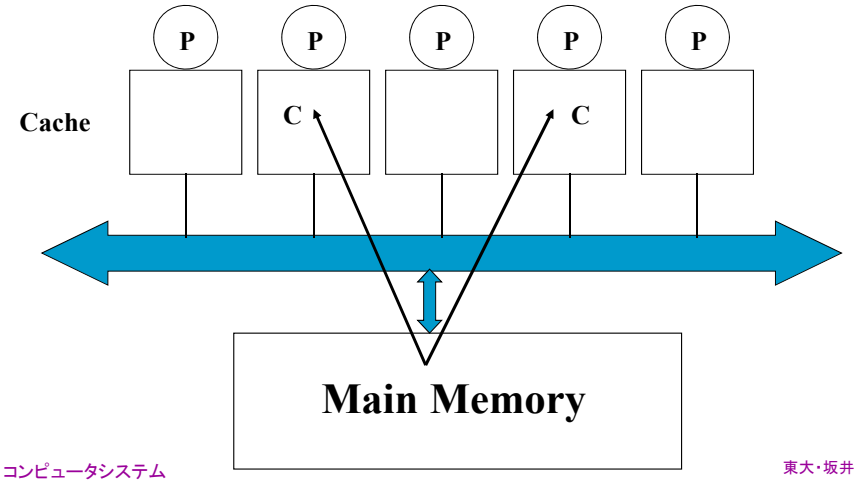
V: Valid, I: Invalid



無効化型ライトバック基本 (1/5)

(1) 読み出し

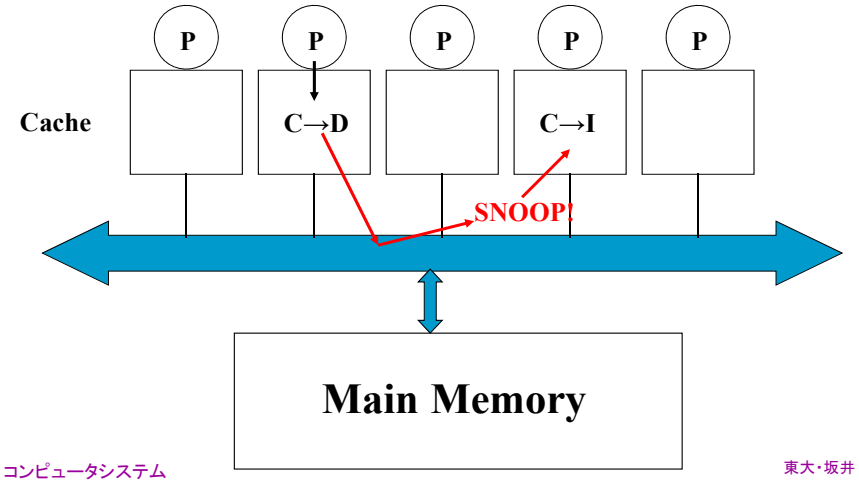
C: Clean （主記憶と一致）
D: Dirty （主記憶と不一致）
I: Invalid



無効化型ライトバック基本 (2/5)

(2) 書き込み
（自キャッシュにデータあり）

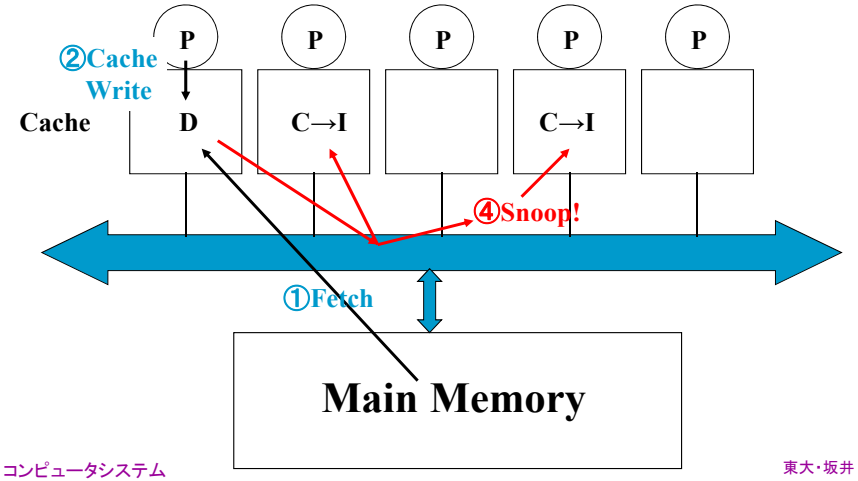
C: Clean （主記憶と一致）
D: Dirty （主記憶と不一致）
I: Invalid



無効化型ライトバック基本 (3/5)

(3) 書き込み
（自キャッシュにデータがない）

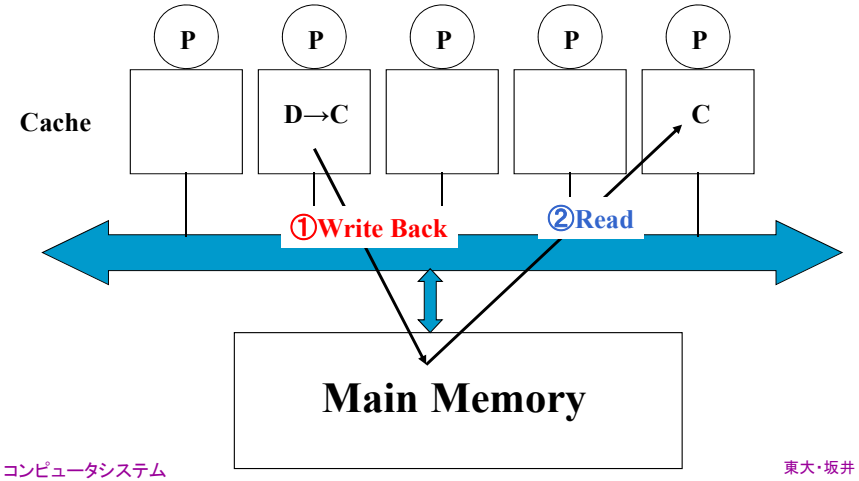
C: Clean （主記憶と一致）
D: Dirty （主記憶と不一致）
I: Invalid



無効化型ライトバック基本 (4/5)

(4) Dのあと読み出し

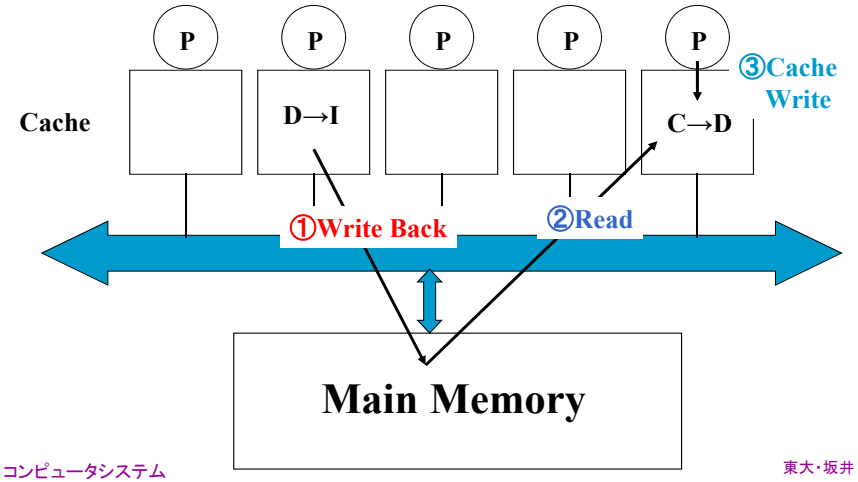
C: Clean （主記憶と一致）
D: Dirty （主記憶と不一致）
I: Invalid



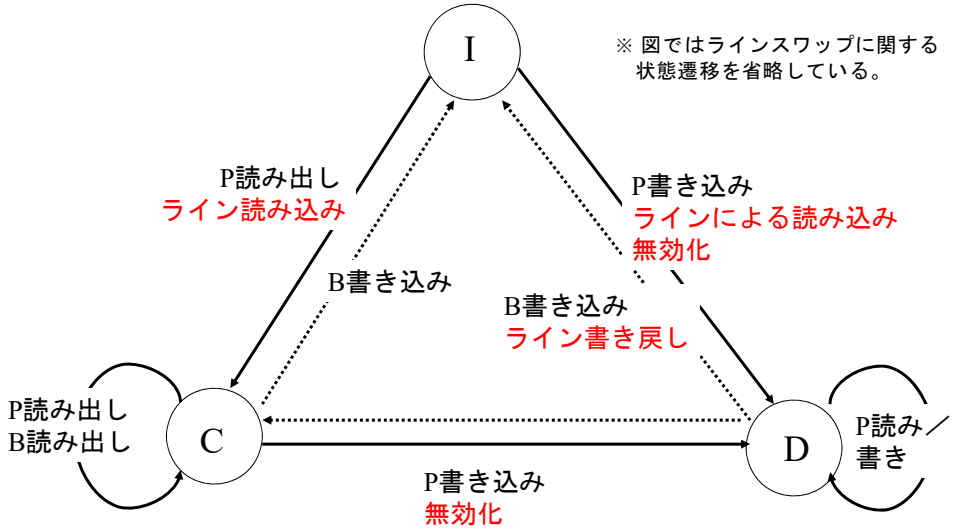
無効化型ライトバック基本 (5/5)

(5) Dのあと書き込み

C: Clean （主記憶と一致）
D: Dirty （主記憶と不一致）
I: Invalid



「無効化型ライトバック基本」の状態遷移



P: Processor, B: Snoop Bus

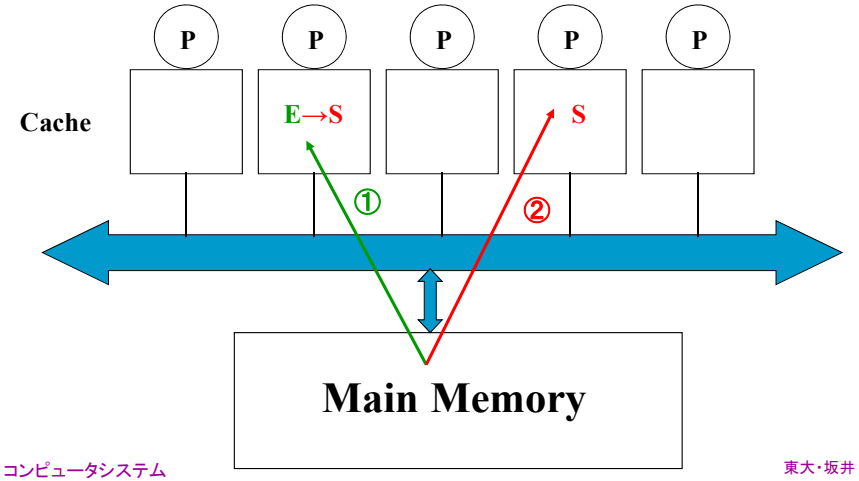
無効化型ライトバック基本の問題点

- 特定のプロセッサだけから参照・更新される変数
  → 更新するたびに「無効化」作業のためにバスが占有される
- 解決法
  - 状態をふやす → Illinoisプロトコル
    I: Invalid
    D: Dirty
    E: (Clean) Exclusive
    S: (Clean) Shared

Illinois（無効化型ライトバック）(1/2)

(1) 読み出し

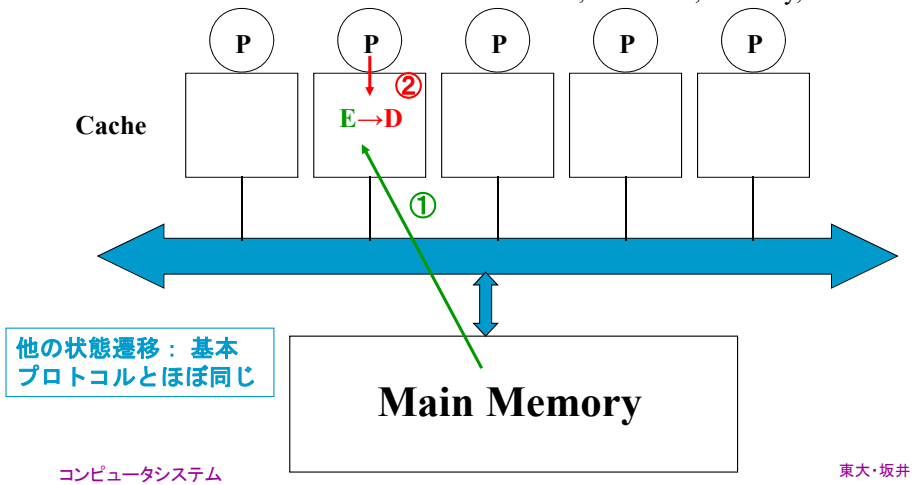
E: Exclusive, S: Shared, D: Dirty, I: Invalid



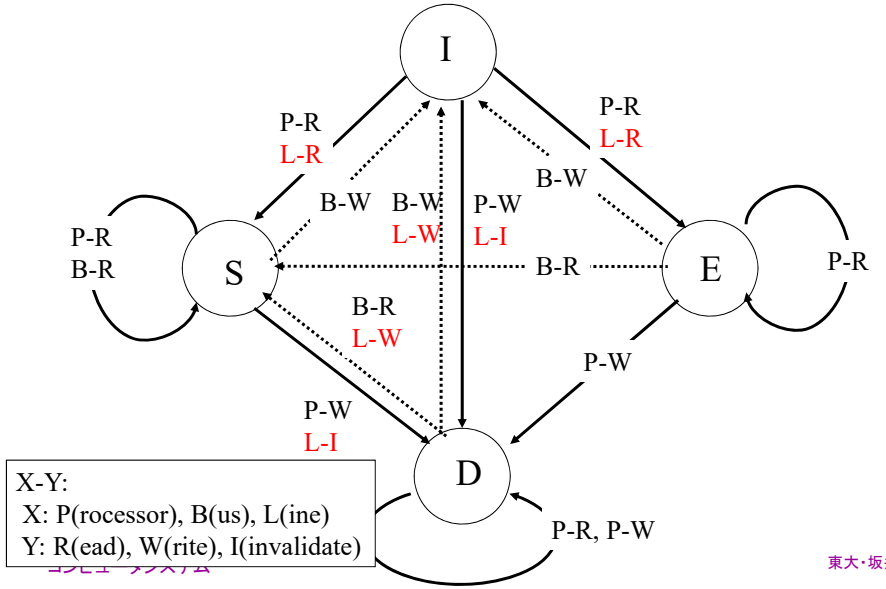
Illinois（無効化型ライトバック）(2/2)

(2) E状態での書き込み

E: Exclusive, S: Shared, D: Dirty, I: Invalid



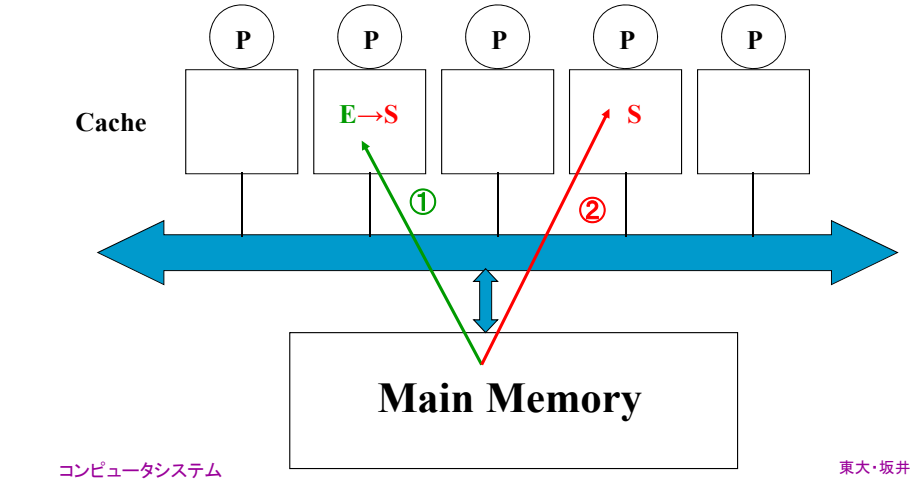
Illinois の状態遷移



Friefly（更新型ライトバック）(1/3)

(1) 読み出し

E: Exclusive, S: Shared, D: Dirty

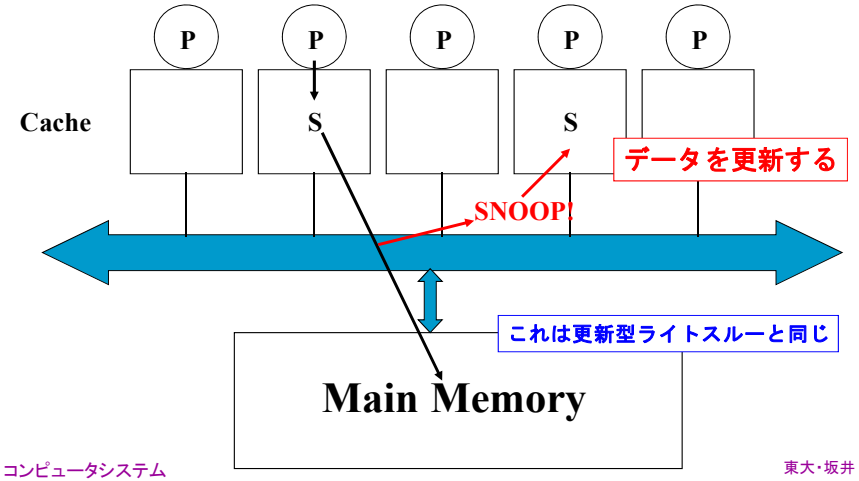


Friefly（更新型ライトバック）(2/3)

(2) 書き込み

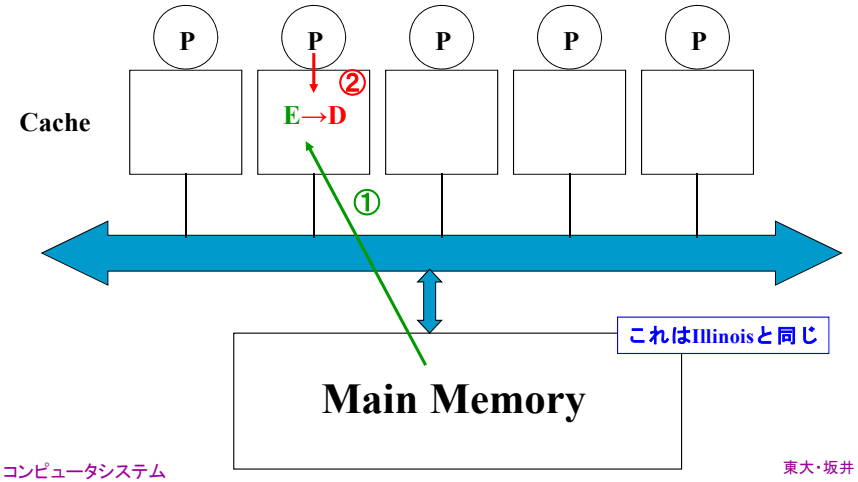
（自キャッシュにデータあり）

E: Exclusive, S: Shared, D: Dirty



Friefly（更新型ライトバック）(3/3)

(3) E状態での書き込み　E: Exclusive, S: Shared, D: Dirty



キャッシュの方式比較

- ライトスルー（WT）とライトバック（WB）
  - ハードウェア
    - WTが簡単
  - 性能
    - 1プロセッサ：　ライトバッファをつけたWTはWBに劣らない
    - マルチプロセッサ：WTはバストラフィックが多く不利
- 無効化型と更新型
  - 共有データへの読み書きが多いとき：
    - 無効化型が交通量が多く不利
  - 共有データが特定プロセッサ以外であまり使われないとき：
    - 更新型は余分な更新をやり続けるので不利

4. スケーラブルな共有メモリ型並列計算機

- スヌープバスによる共有メモリ計算機
  - キャッシュによる局所化
    - アクセスレーテンシの短縮
    - バストラフィックの軽減
    - トランスペアレンシの維持
  - 問題点
    - Scalability（プロセッサ数に比例した性能を出す性質）がない
      - プロセッサ数が増えるとバスネックとなる
        - » cf. Sun StarFire: **コヒーレンス制御はバス、データ転送はスイッチ**
        - → それでもコヒーレンス制御はO(1)でしか処理できない

スケーラブルな共有メモリ型並列計算機

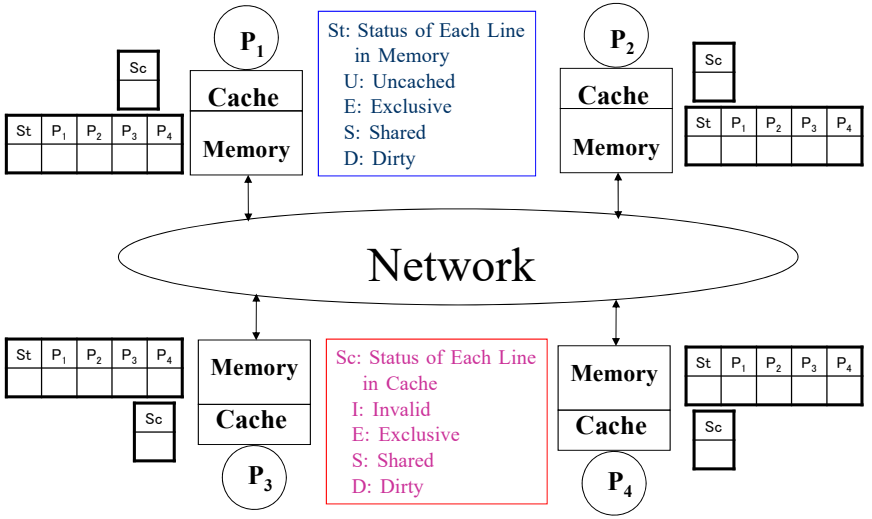
- 網の形態とコヒーレンス制御方式が課題
  - 結合網：　Interconnection Network
    - 本講義の後の回で
    - ほぼスケーラブルな通信を保証することが可能
  - コヒーレンス制御方式
    - ディレクトリ法
      - ディレクトリによって、キャッシュコヒーレンスを取る手法のこと
      - ディレクトリ
        どのノードのキャッシュがどのメモリのどのラインをもっているか、という情報[を入れたもの(テーブル)]
      - 通常、ディレクトリは共有メモリのコントローラが管理している

ディレクトリ法の分類

- ディレクトリ法の分類
  - ディレクトリ全コピー法
  - フルマップディレクトリ法
  - リミテッドポインタ法
  - チェインドディレクトリ法

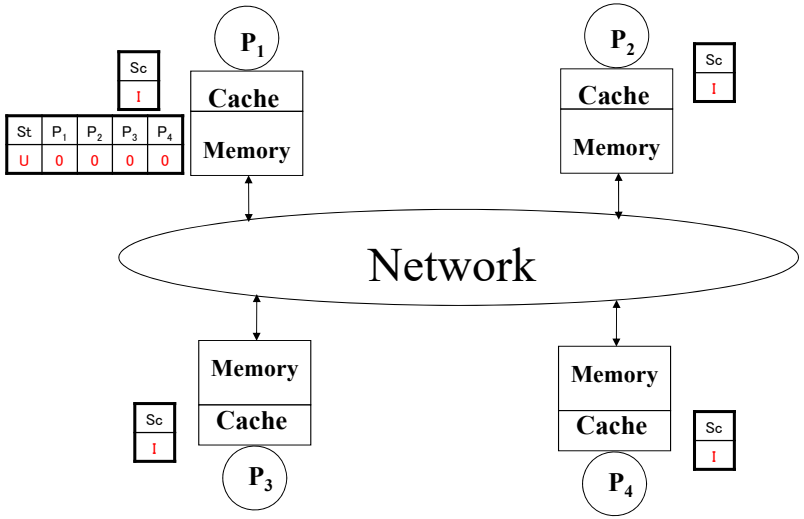
- ディレクトリ全コピー法
  - 各キャッシュのキャッシュディレクトリの複製を、共有メモリのコントローラ上に置いて集中管理する方式
  - ディレクトリが大きくなり、検索に時間がかかる
    → 非現実的

フルマップディレクトリ法(1/7):構成法

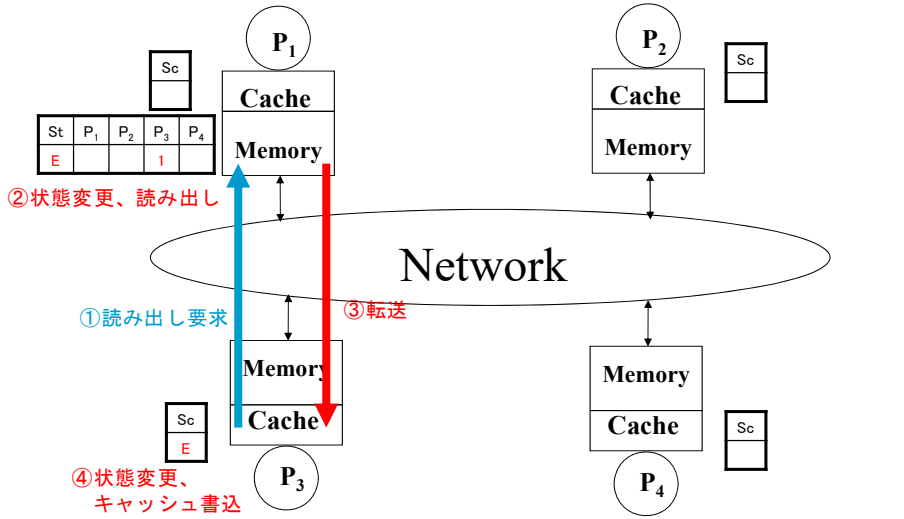


＊ ここではIllinois Protocolを採用している

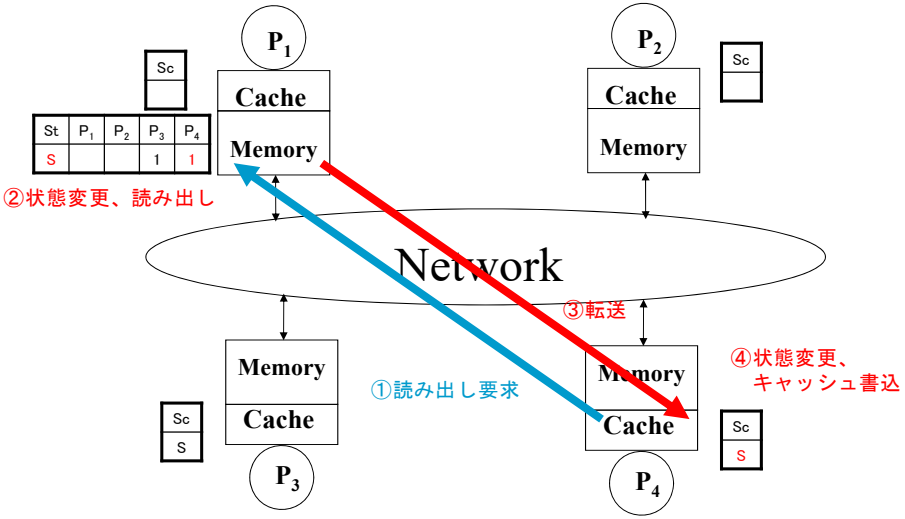
フルマップディレクトリ法(2/7): 初期状態



フルマップディレクトリ法(3/7): 読み出し

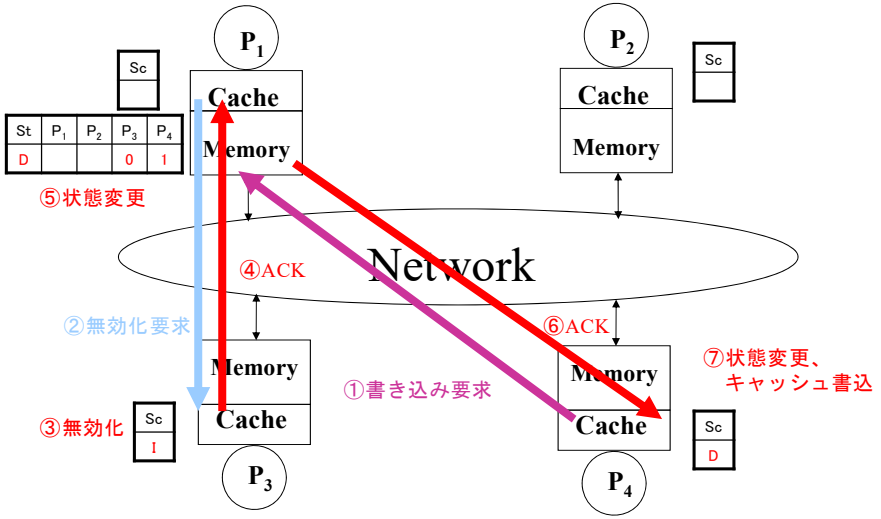


フルマップディレクトリ法(4/7): 読み出し(その2)



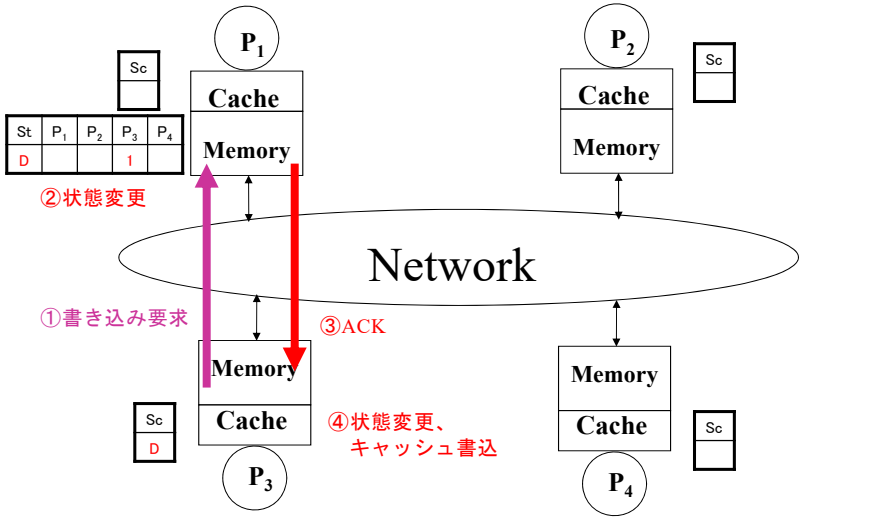
フルマップディレクトリ法(5/7): 書き込み(その1)

S状態からの書き込み



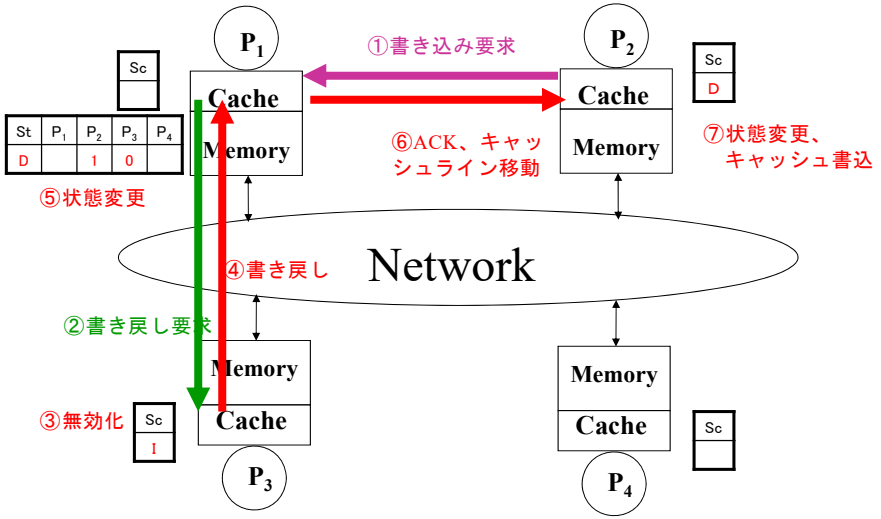
フルマップディレクトリ法(6/7): 書き込み(その2)

E状態からの書き込み



フルマップディレクトリ法(7/7): 書き込み(その3)

メモリがD状態のラインへの書き込み

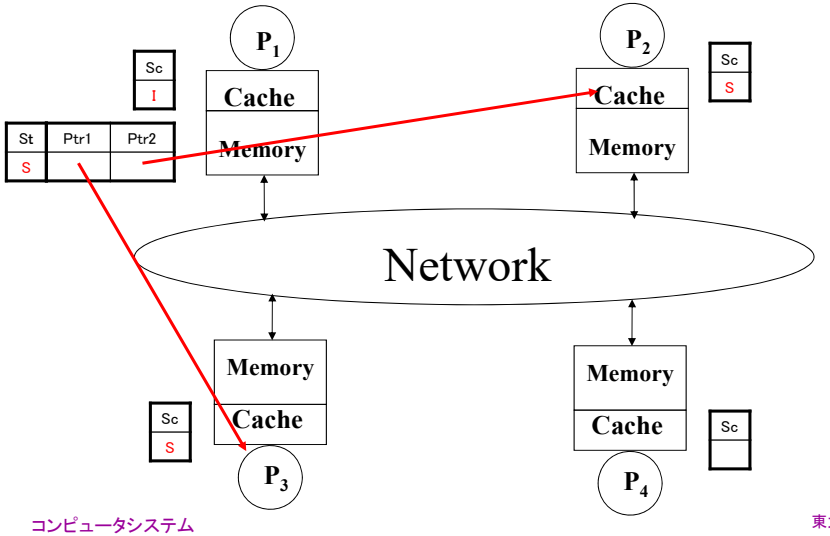


フルマップディレクトリ法の利点と欠点

- 利点
  - 1プロセッサあたりのディレクトリの大きさ
    - （プロセッサ台数）×（ライン数） ビット
    - ディレクトリ全コピー法よりははるかに小さい
  - 制御が簡単
- 欠点
  - プロセッサ数が大きいとディレクトリの大きさも大きくなる
    - スケーラブルでない！！

リミテッドポインタ法

少数のポインタを持たせて当該ラインをキャッシュしている場所を指させる

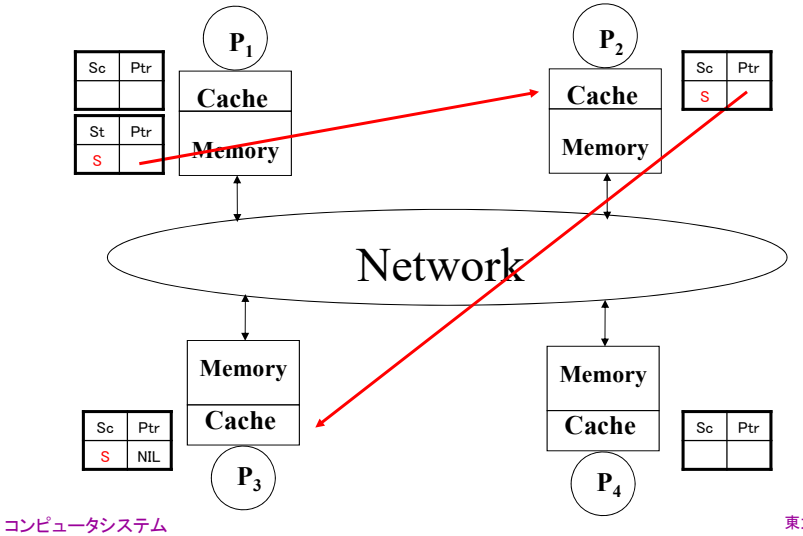


リミテッドポインタ法の利点と欠点

- 利点
  - スケーラブル
    - プロセッサが増えてもディレクトリを大きくしなくてよい
- 欠点
  - 複雑。特にポインタ領域が足りなくなったとき
    - ソフトウェアによる処理　→　共通領域の利用など
    - 強制的な無効化

チェインドディレクトリ法

キャッシュ側にもポインタを持たせて共有関係のチェインを作る



チェインドディレクトリ法の利点と欠点

- 利点
  - ディレクトリの大きさが小さくてすむ
  - ポインタ不足の問題がない
- 欠点
  - チェインを辿るのに時間がかかる

大規模共有メモリ型並列計算機の実例

- CC-NUMA
  - Cache Coherent Non Uniform Memory Architecture の略。本日扱った共有メモリ型アーキテクチャはすべてここに分類される
- 実例
  - DASH （Stanford U.）
    - 1ノード = SGI Power Challenge (Snoop Bus)
    - メッシュ型ネットワーク
    - ディレクトリによるノード間コヒーレンス制御
  - Origin-2000 （SGI）
  - Jump-1 (U.Tokyo, Kyoto U., Keio U., etc.)
    - ページ単位の管理
    - ライン単位の転送
    - 10000PE規模を目指す → 128 PE版試作

共有メモリ型並列計算機の展開

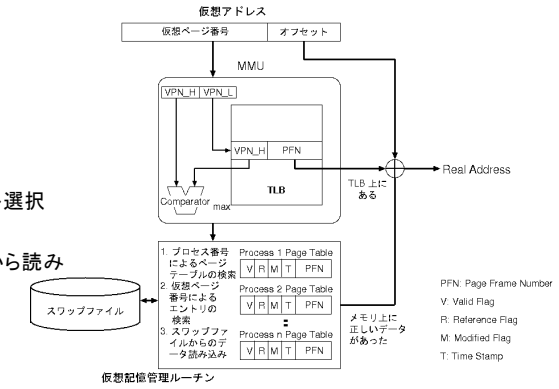
- 既存の並列計算機・コモディティの利用
  特別なコヒーレンス機構がなくても高性能が出せる！
  - Shrimp (Princeton U.)
    - Intel Paragon Network + Pentium PC
  - Tempest/Typhoon （UWM）
    - CM-5 + Virtual Shared Memory
- WS/PC Clusterとの融合

共有仮記憶(SVM)

仮想記憶機構を改変することで DSM を実現

MMU : 仮想アドレス→実アドレス

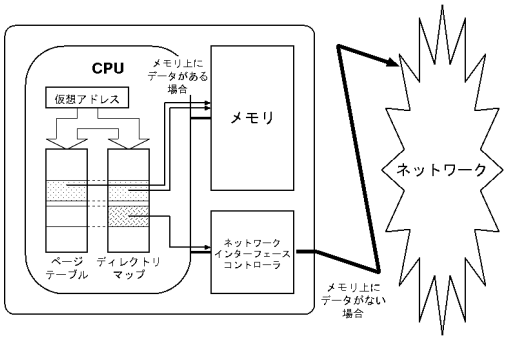
- if (TLB に該当アドレスが存在)
  実アドレスでアクセス
- TLB 検索に失敗
  → 仮想記憶管理ルーチンへ
  - プロセス番号からページテーブル選択
  - 仮想ページ番号から所在の検索
  - スワップしていた場合、ディスクから読み込む



共有仮想メモリ(SVM)(その2)

仮想記憶管理ルーチンの書き換え

- ページ単位のデータ共有を実現
- ディレクトリマップ
  - ページテーブルと独立して設定
- メモリ上に正しいデータがある
  - → ページテーブルに実アドレスを入れる
- データが無い場合
  - → ディレクトリマップを基にデータを要求
- 通信には一般的な LAN を使用
- 共有アドレスはキャッシュ不可
- Sequential Consistency を採用



ナイーブなSVM の問題点と対策

- メモリコンシステンシ
  - Sequential consistency → Relaxed consistencyへ
- 通信オーバーヘッド
  - Ethernet ではプロトコル処理や I/F までのデータ転送に CPU が使われてしまう → オーバラップさせる
- アクセス検出粒度

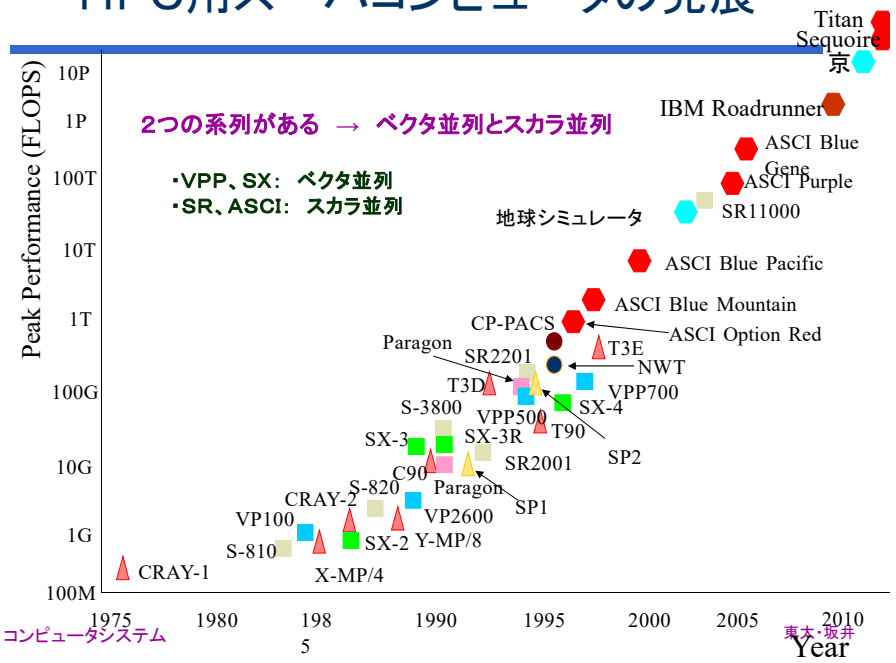
  ページ単位： 大きすぎる
  - False Sharing がおき、無駄な通信で性能を落とす

False sharing: dirtyになったブロックでは、dirtyにならなかったデータ(shareされていないデータ)までinvalidになる！

PCクラスタ

- コモディティを最高度に活用
  - プロセッサ： Pentium
  - ネットワーク: [Myrinet,] Gbit Ether
- 90年代までの並列処理技術の転化・応用
  - 共有メモリの技術
    - レーテンシ削減(キャッシュ)
    - コヒーレンス制御
    - レーテンシ隠蔽(プリフェッチ)
  - メッセージ交換の技術/データフローの技術
    - アクティブメッセージによる通信オーバヘッド削減
    - レーテンシ隠蔽(スレッド切り替え)
  - 同期の技術
  - ネットワークの技術 → 最終回
    - トポロジー最適化
    - IPパケット通信
- 例
  - (たくさん市販されている！)

HPC用スーパコンピュータの発展



まとめ

- 新世代の並列計算機
  - Convergence（収束）とDivergence（発散）
  - コモディティ化＋ソフトウェア化の進展
    - CPU
    - ネットワーク
  - 新技術の研究開発
    - マルチスレッド
    - 投機処理: データ投機、マルチパス
    - チップマルチプロセッサ
    - メモリ混載アーキテクチャ

Wintelをワッチしてこれにアドオンする技術を開発すると
いうことだけではなく、「新しいコモディティを作ること」が第一