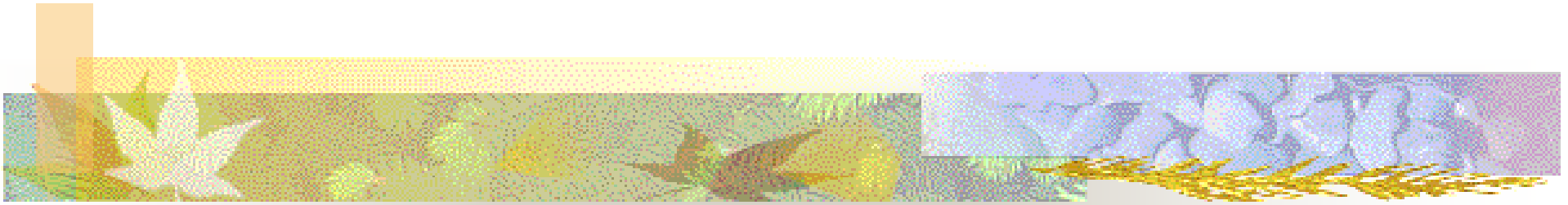
北海道大学
HOKKAIDO UNIVERSITY

コンピュータシステム
（アーキテクチャ　第7回）



工学部 情報エレクトロニクス学科
大学院 情報科学研究院 情報理工学部門
堀山 貴史

前回（アーキテクチャ第6回）の内容

並列処理アーキテクチャ

1. 計算機の処理能力の2つの要素
   - スループット、レイテンシ
2. 並列処理
   - 並行処理と並列処理、パイプライン並列とマルチプロセッシング並列
3. 並列処理の分類
   - SIMD と MIMD、粒度による分類
4. 並列アーキテクチャ
   - マルチプロセッサ・マルチコア(MIMD型)、SIMDプロセッサ
   - 命令パイプライン、演算パイプライン
5. ノイマン型計算機の限界
   - フォンノイマン・ボトルネック

5. ノイマン型計算機の限界

- ノイマン型計算機の特徴
  - プログラム内蔵方式
  - １次元に並んだメモリ空間
  - メモリ(主記憶)上のプログラムをCPUで逐次実行
  - メモリとCPUは、転送路を通して接続

ノイマン型計算機のボトルネック（フォンノイマン・ボトルネック）
命令もデータもすべてここを通るので、ここの性能がボトルネックになる

主記憶アクセスの高速化が、計算機の高速化のカギを握る

今回： メモリ アーキテクチャ

今回の内容

メモリ アーキテクチャ

1. 記憶装置の容量とアクセス速度
2. 参照の局所性とメモリ階層
   - 空間的局所性、時間的局所性
   - メモリの階層構造
3. 主記憶へのアクセスの高速化
   - メモリ インターリーブ
   - キャッシュ メモリ
4. ディスクキャッシュと仮想メモリ

1. 記憶装置の容量とアクセス速度

- プロセッサ内メモリ　プロセッサ
- DRAMモジュール　主記憶装置
- フラッシュメモリカード（SSD: Solid-State Drive）　補助記憶装置
- ハードディスク（HDD: Hard Disk Drive）　補助記憶装置
- 磁気テープ　バックアップ装置

記憶容量が大きくなるほど
アクセス速度が低下する

1. 記憶装置の容量とアクセス速度

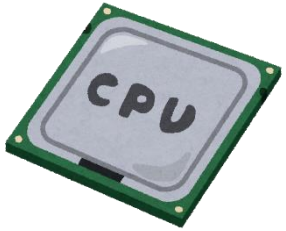
- プロセッサ内メモリ　プロセッサ
  - 容量：数KB～数MB、レイテンシ：数ns、高速ランダムアクセス
- DRAMモジュール　主記憶装置
  - 容量：数GB～数百GB、レイテンシ：数十ns、高速ランダムアクセス
- フラッシュメモリカード（SSD: Solid-State Drive）　補助記憶装置
  - 容量：数十～数百GB、レイテンシ：数百μs、
    ブロック単位でまとめて読み書き、ランダムアクセスはやや苦手
- ハードディスク（HDD: Hard Disk Drive）　補助記憶装置
  - 容量：数十TB、レイテンシ：数十ms、
    機械式ヘッド移動、ランダムアクセスは苦手

（参考）着脱可能ディスク：　FD（フロッピーディスク）、CD（コンパクトディスク）、
MO（光磁気ディスク）、DVD（ディジタルビデオディスク）, BD（ブルーレイディスク）

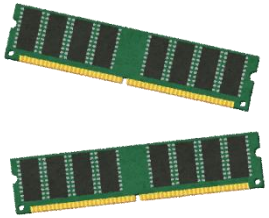
- 磁気テープ　バックアップ装置
  - PB以上、早送り/巻戻しが必要

**記憶容量が大きくなるほど
アクセス速度が低下する**

2. 参照の局所性とメモリ階層



プロセッサ内メモリ
- 速い
- 容量が小さい



主記憶
- プロセッサから見ると、遅い
- 容量は大きい

参照の局所性

ノイマン型計算機における2つの局所性：

- **空間的局所性**
  - 現在アクセスした場所の周囲の番地は、続けてアクセスされることが多い

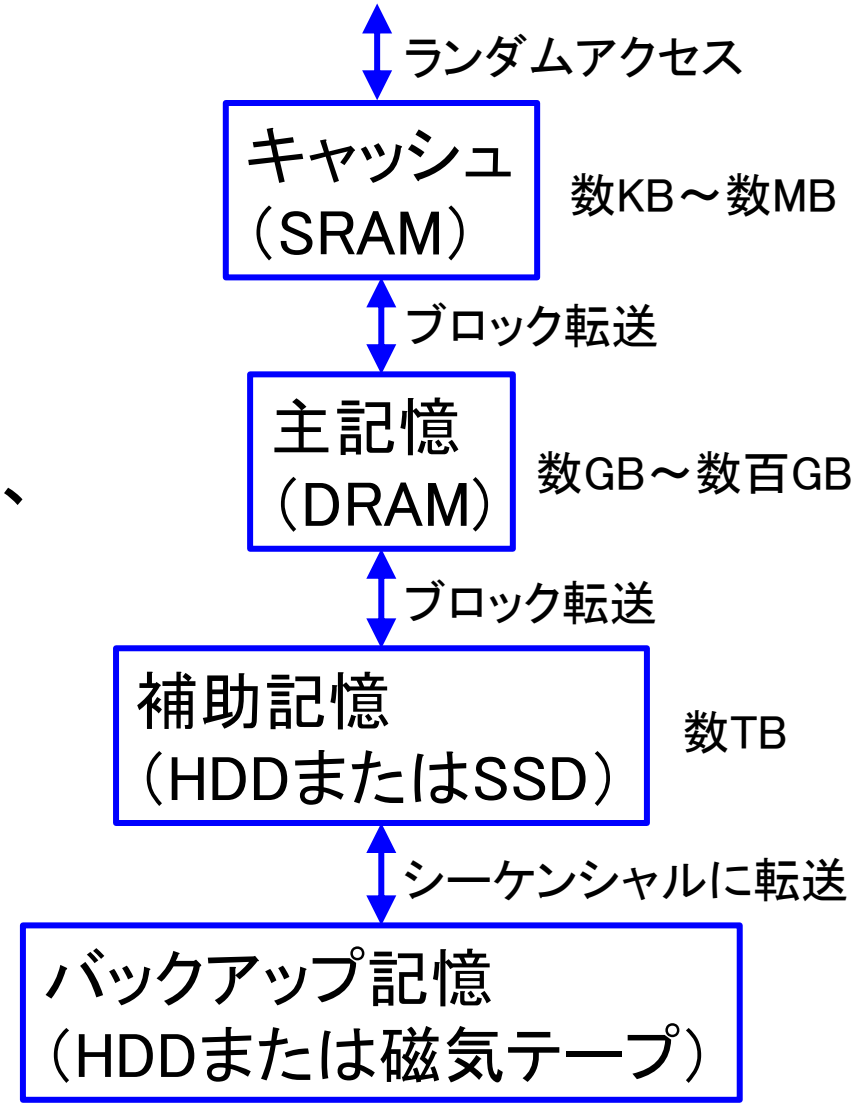
  例：機械語命令は順に読み出される
  配列データは順に読み出される
- **時間的局所性**
  - 比較的短い間隔で、同じ場所が何回もアクセスされることが多い

  例：ループで同じ機械語命令が何回も読み出される
  データを読み出して計算処理して、元の場所に上書き更新

主記憶アクセス高速化の原理は、参照の局所性に基づく

メモリの階層構造

- 高速なメモリを大量に並べることは、物理的または経済的に不可能
- **高速小容量**なメモリと**低速大容量**なメモリを組み合わせて階層構造を構成する
  → 見かけ上、**高速大容量**のメモリを、**現実的なコストで実現**できる
- 高速小容量なメモリをCPUの近くに、低速大容量なメモリを遠くに配置
- 階層間のデータはブロック転送
  - まとめて転送することにより、個別にランダムアクセスするよりもはるかに高速に転送できる
- **参照の局所性を利用**



3. 主記憶へのアクセスの高速化

参照の局所性を利用して
主記憶へのアクセスを高速化する工夫

- **メモリ インターリーブ**
  - 複数のメモリアクセスを一括処理
    - 機械語命令や配列データは順々に取り出されるので
      ブロック単位でまとめて読み出すと見かけ上高速化できる
  - 複数のメモリアクセスをパイプライン処理
    - 直前のメモリアクセスの完了を待たずに、次のメモリアクセスの準備を始めても良い場合がある
- **キャッシュ メモリ**
  - 主記憶とCPUの間に置かれる高速小容量のメモリ
  - アクセスされた番地の内容を**自動的に保存**しておき、
    同じところにアクセスしたら即座に返す

休 憩

- ここで、少し休憩しましょう。

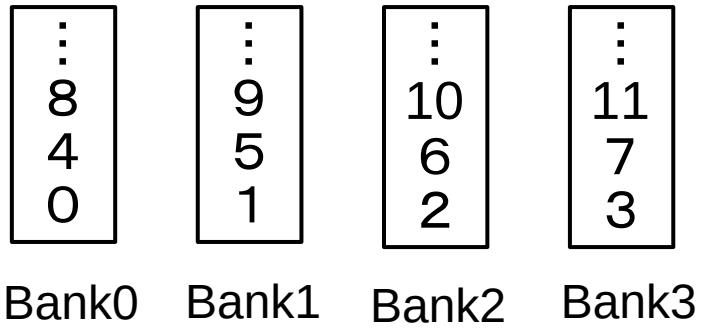
- 深呼吸したり、肩の力を抜いてから、
  次のビデオに進んでください。

3. 主記憶へのアクセスの高速化 (1)

- メモリ インターリーブ

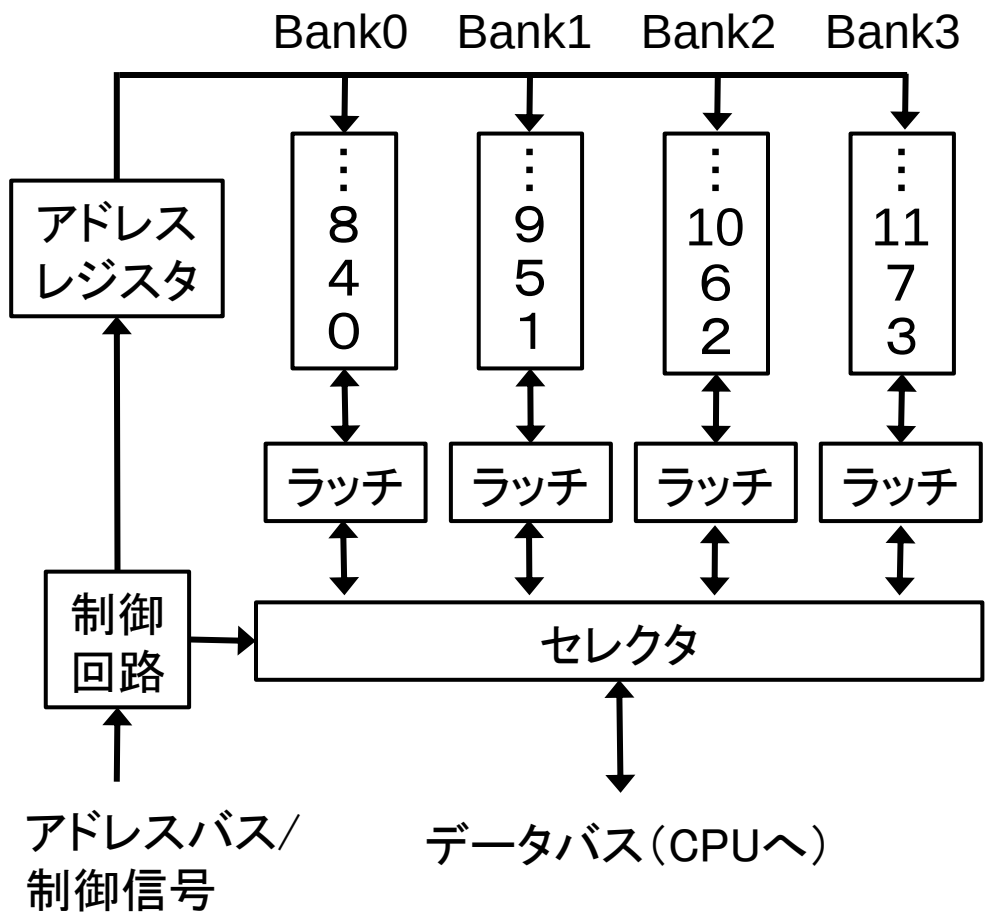
メモリインターリーブ

- メモリ領域をｍ個の**バンク（部分領域）に分割**し、**それぞれ別のメモリ素子**に担当させる
  - ｍ重インターリーブメモリ（m-way interleaved memory）と呼ぶ
- メモリの番地は、各バンクごとに連続的ではなく、1ワードずつバンクをローテーションする順序で割り振る
  - 異なるバンクのデータは並列にアクセス要求ができる
  - 同じバンク内のデータは1度に1ワードずつしかアクセスできない



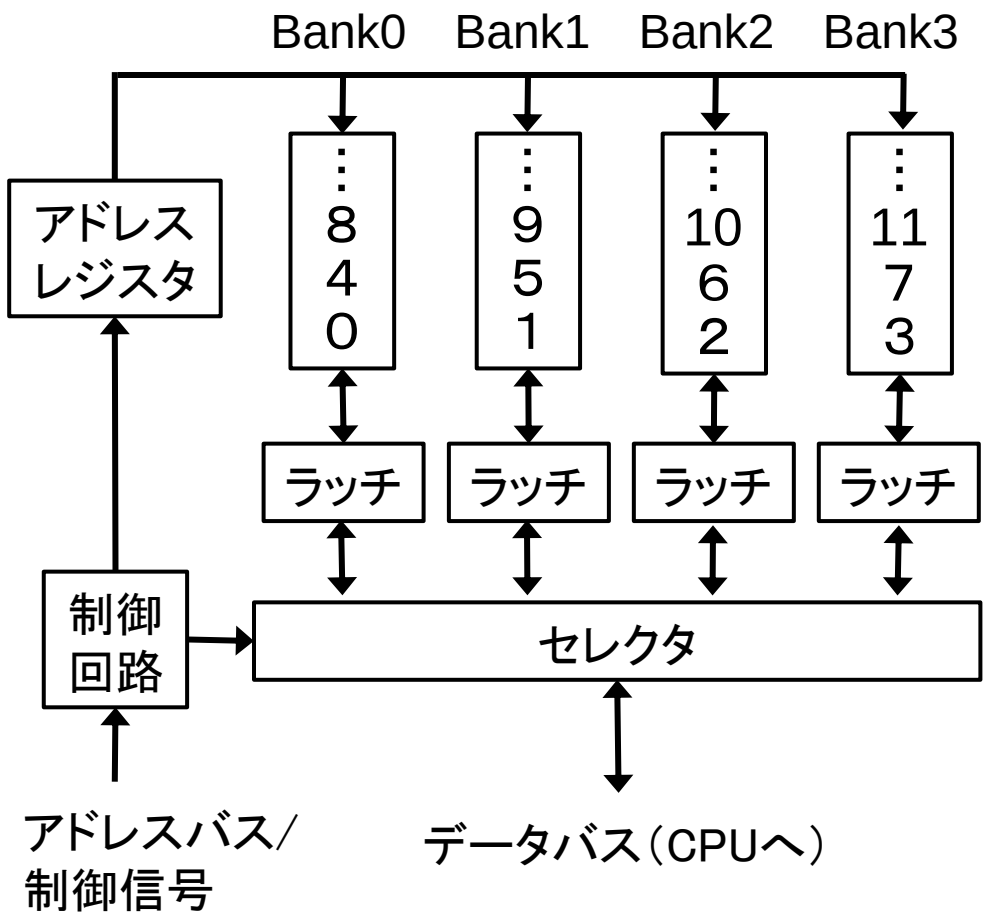
メモリインターリーブによる一括アクセス

- 連続するｍ番地をまとめて
  各バンクに読み出し要求する
  - メモリのデータ出力が安定
    したらラッチに一時記憶する



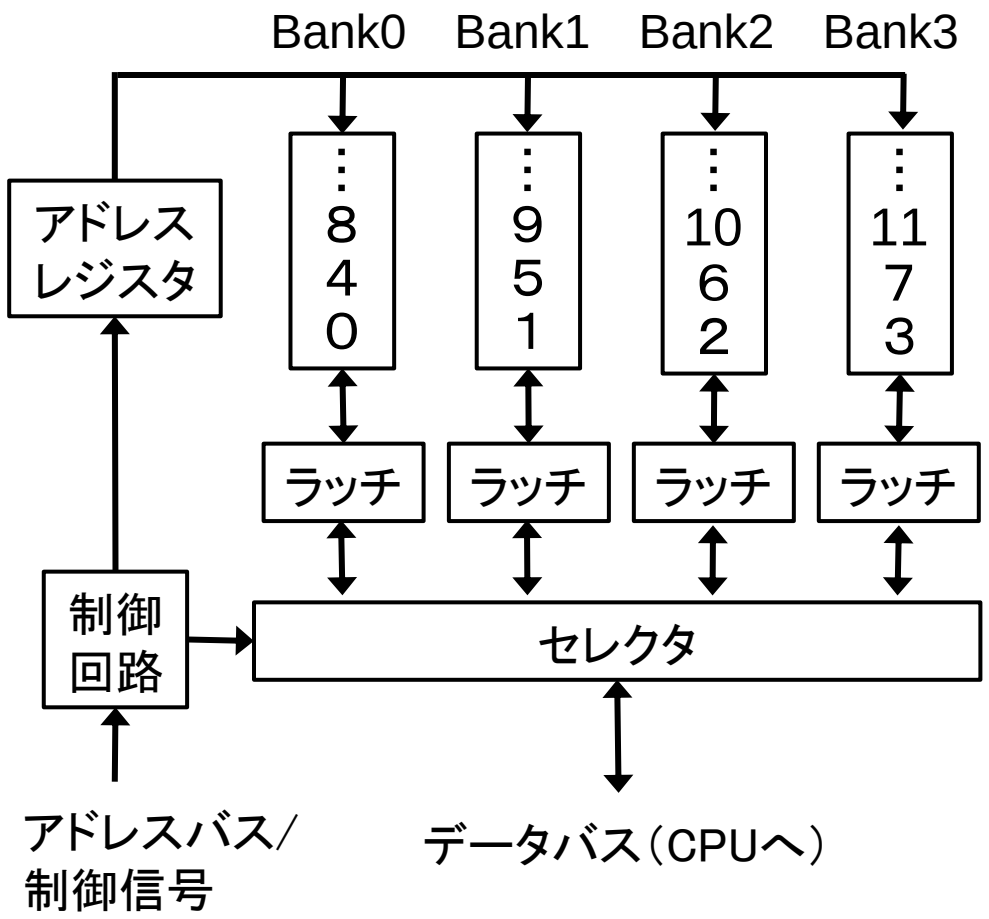
メモリインターリーブによる一括アクセス

- 連続するｍ番地をまとめて各バンクに読み出し要求する
  - メモリのデータ出力が安定したらラッチに一時記憶する
- セレクタを使って切り替えながらｍ個のデータを順にCPUに送り出す
  - ラッチやセレクタは、メモリに比べ、はるかに高速に動作
- ｍ個のデータを送っている間に、次のｍ番地のメモリ読み出し要求を行う
  - 以下同様に繰り返し



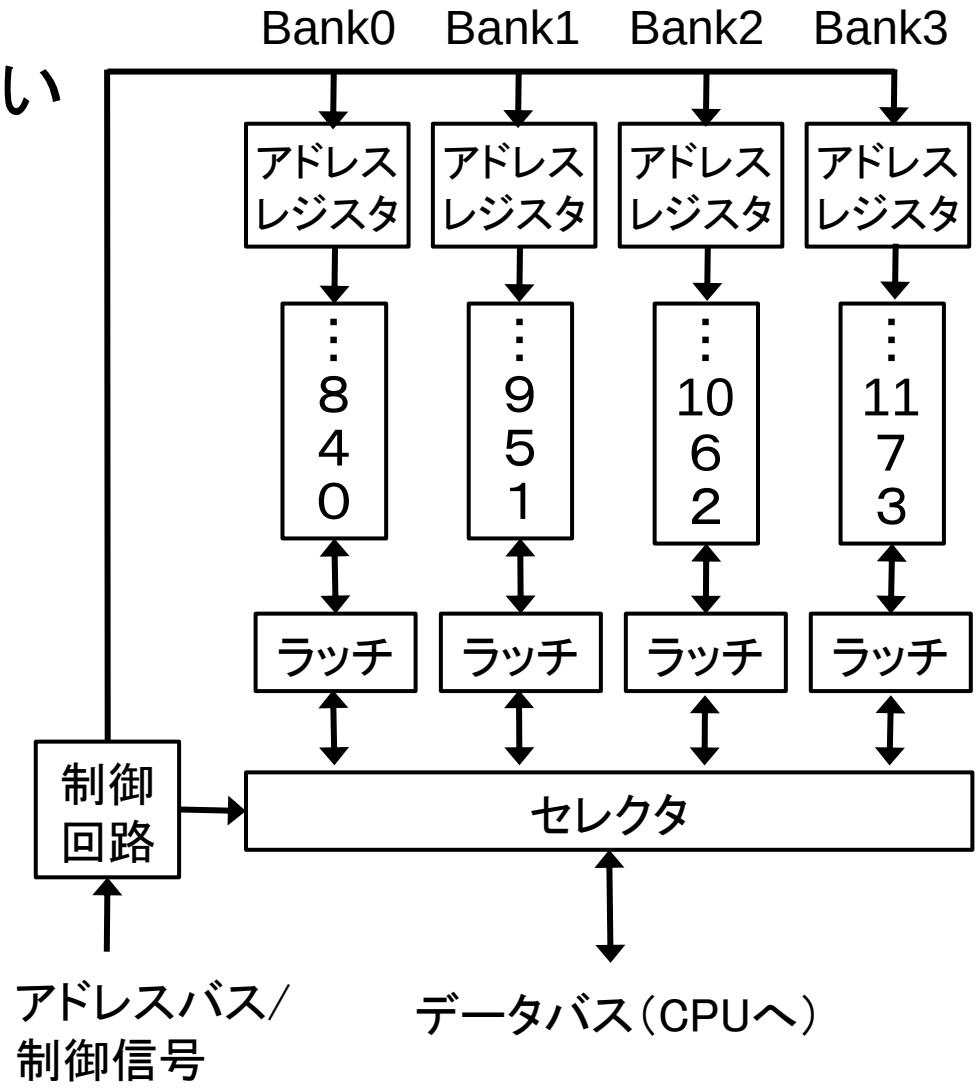
メモリインターリーブによる一括アクセスの効果

- m系統のバンクを用意すれば、最大m倍高速に処理できる
  - mを大きくするほど高速化できるが、ハードウェア量が増える
- 連続する番地を一括ブロック処理する場合に有効
  - 無関係の番地にランダムにアクセスする場合はほとんど効果がない
  - 順々にアクセスする場合でも、偶数番地だけ順に読むとか、kの倍数の番地だけ順に読む場合には、効果が低下する



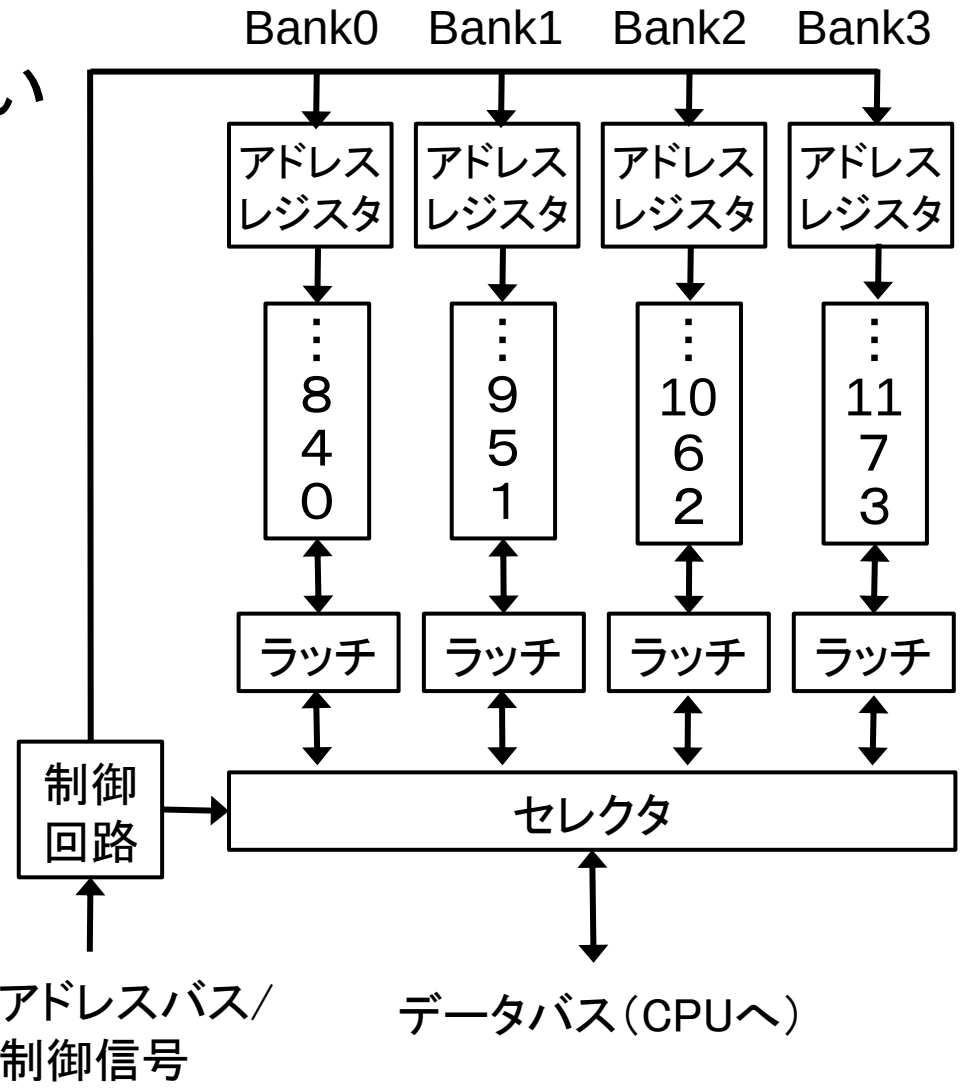
メモリインターリーブによるパイプラインアクセス

- 各バンクごとに個別のアドレスレジスタを用意すれば、連続しないアドレスでも並列処理できる



メモリインターリーブによるパイプラインアクセス

- 各バンクごとに個別のアドレスレジスタを用意すれば、連続しないアドレスでも並列処理できる
- 第ｋバンクのアクセス要求が完了する前に第(k+1)バンクへのアクセス要求を開始できる
- ラッチとセレクタで切り替えながら次々とデータを読み出せる
  - 最大ｍ倍の高速化
- ただし同じバンクの番地に続けてアクセスしようとすると、終了するまで待たされる（→アクセス衝突）



3. 主記憶へのアクセスの高速化 (2)

- キャッシュ メモリ

キャッシュメモリ（cache memory）

- CPUと主記憶の間に置かれる高速メモリ。**読み出した内容をコピー保存**しておき、**同じ番地を要求されたら即座に返す**
  - メモリ参照の時間局所性により高速化
  - アクセスした周囲の**ブロック単位で保存**する（→空間局所性）
  - 容量は主記憶の1/1000程度、速度は主記憶の10倍程度が目安
  - 2、3階層のキャッシュ（L1, L2, L3キャッシュ）が置かれることもある

（例：Intel Core i7：L1: 80 KB/core × 8 core, L2: 512 KB/core × 8 core, L3: 16 MB）

キャッシュメモリ（cache memory）

- CPUと主記憶の間に置かれる高速メモリ。**読み出した内容をコピー保存**しておき、**同じ番地を要求されたら即座に返す**
  - メモリ参照の時間局所性により高速化
  - アクセスした周囲の**ブロック単位で保存**する（→空間局所性）
  - 容量は主記憶の1/1000程度、速度は主記憶の10倍程度が目安
  - 2、3階層のキャッシュ（L1, L2, L3キャッシュ）が置かれることもある

  （例：Intel Core i7：L1: 80 KB/core × 8 core, L2: 512 KB/core × 8 core, L3: 16 MB）
- キャッシュ容量に限界があるので、古いブロックを追い出し、新しいブロックと置き換える（なるべくヒットしやすい方法は？）
- キャッシュの番地と本来のメモリの番地との変換機構が必要
- メモリ書き込みがあると、メモリの内容の更新と、キャッシュ上のコピーの更新が必要になる

キャッシュメモリ（cache memory）

- WT法（Write-Through）
  - データ書き込みは常に主記憶に対して行う。
    キャッシュがヒットすればそちらも更新する。
  - 常に主記憶とキャッシュの一致状態が保たれるが、
    書き込みについては高速化効果が出ない
- WB法（Write-Back）
  - キャッシュがヒットするときには、キャッシュのみに書き込み、
    主記憶は更新しない。
  - キャッシュが追い出される時に、主記憶に転送して反映させる
  - 書き込みも高速化されるが、機構が複雑になる

- メモリ書き込みがあると、メモリの内容の更新と、
  キャッシュ上のコピーの更新が必要になる

キャッシュのブロック置き換え法

- 古いブロックを追い出す:
  具体的にどのブロックを追い出すか

- LRU法 (Least-Recently-Used)
  - 最も長い間アクセスされなかったブロックを選ぶ
  - 効率が良いが、LRUスタックを管理する必要がある
- FIFO法 (First-In, First-Out)
  - そこそこ効率が良い
  - LRU法よりは機構が容易
- ランダム法
  - 機構は簡単だが効率は良くない

休 憩

- ここで、少し休憩しましょう。

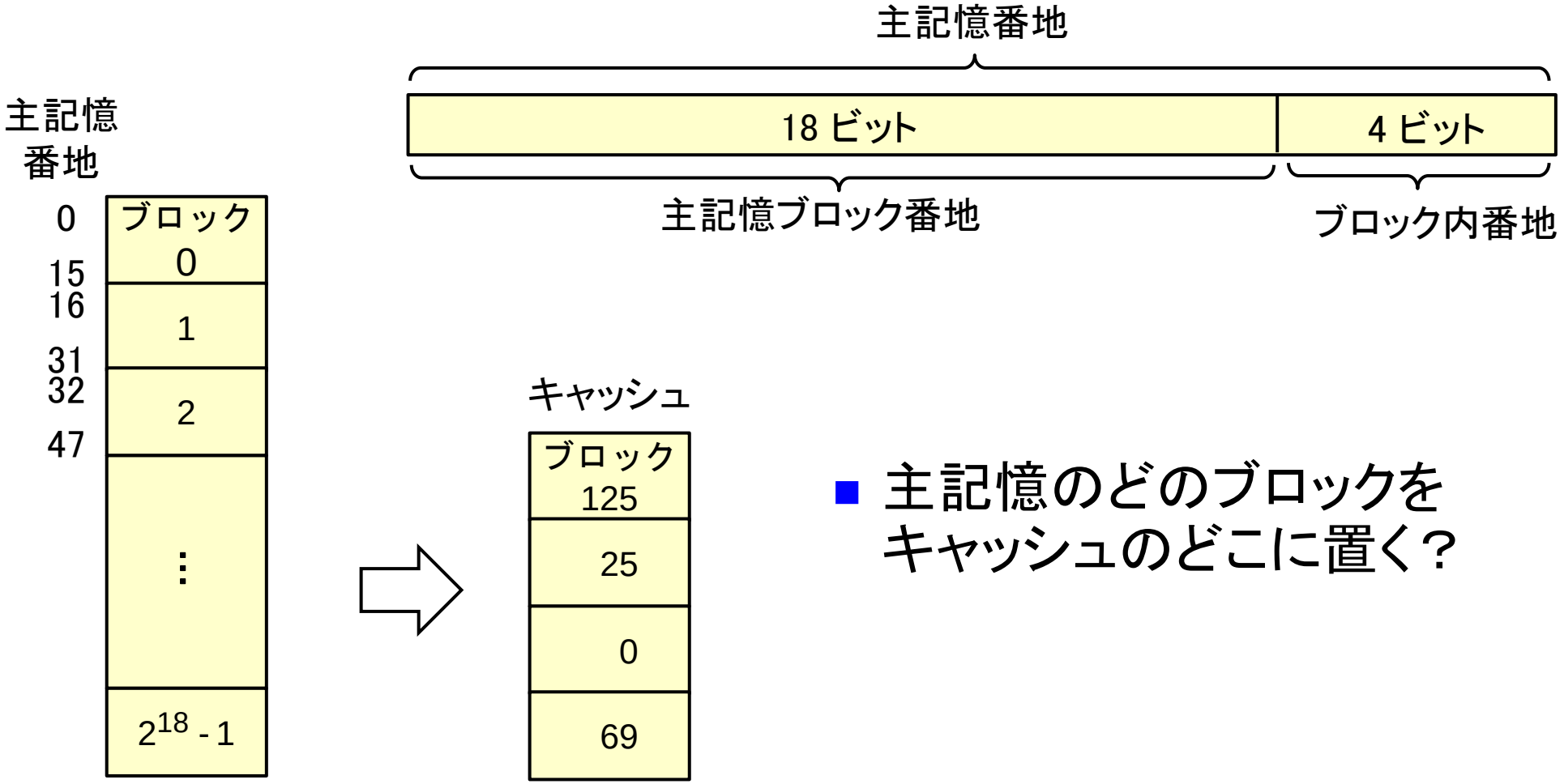
- 深呼吸したり、肩の力を抜いてから、
  次のビデオに進んでください。

3. 主記憶へのアクセスの高速化 (2)

- キャッシュ メモリ
  - キャッシュの配置法

主記憶をブロックに分ける

- 主記憶をブロックに分け、キャッシュにコピーを保存
  例）1ブロックは $2^4 = 16$ワード、主記憶を 2^{18} 個のブロックに分ける



- 主記憶のどのブロックを
  キャッシュのどこに置く？

キャッシュ配置法

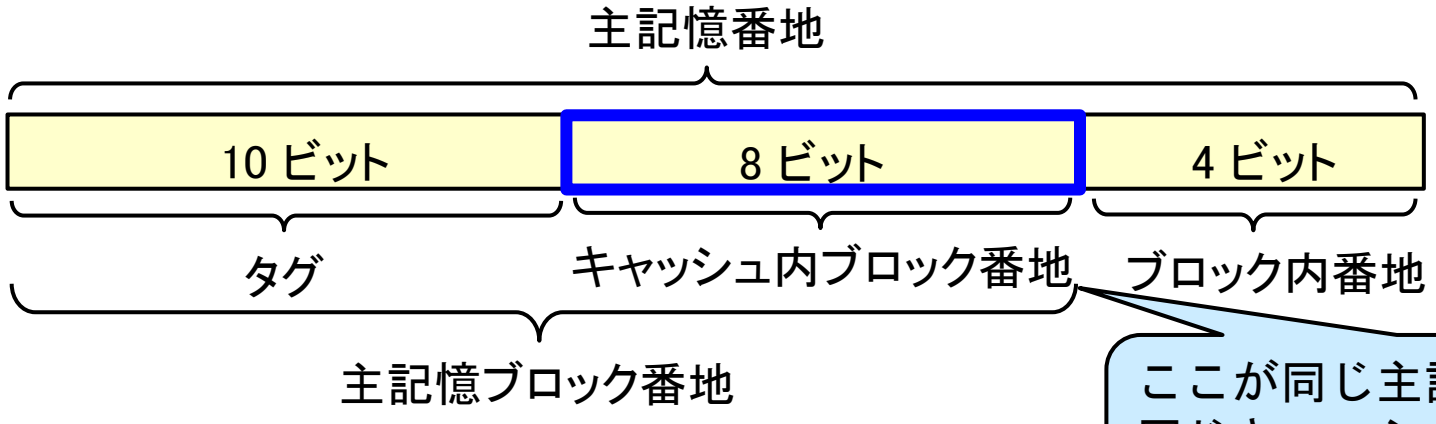
主記憶の各ブロックを、キャッシュのどの番地に割り当てて保存するかで、いくつかの方式がある。

- **直接法** (direct mapped)
  - 内部機構が簡単だが、効率が低下しやすい
- **完全連想配置法** (full associative)
  - 理想的だが、実装コストが高く実用的でない
- **部分連想配置法** (set associative)
  - 現実的で、比較的高性能な方式

- ３つの方法の概略をまず把握する
- その後で、実装を見てみる

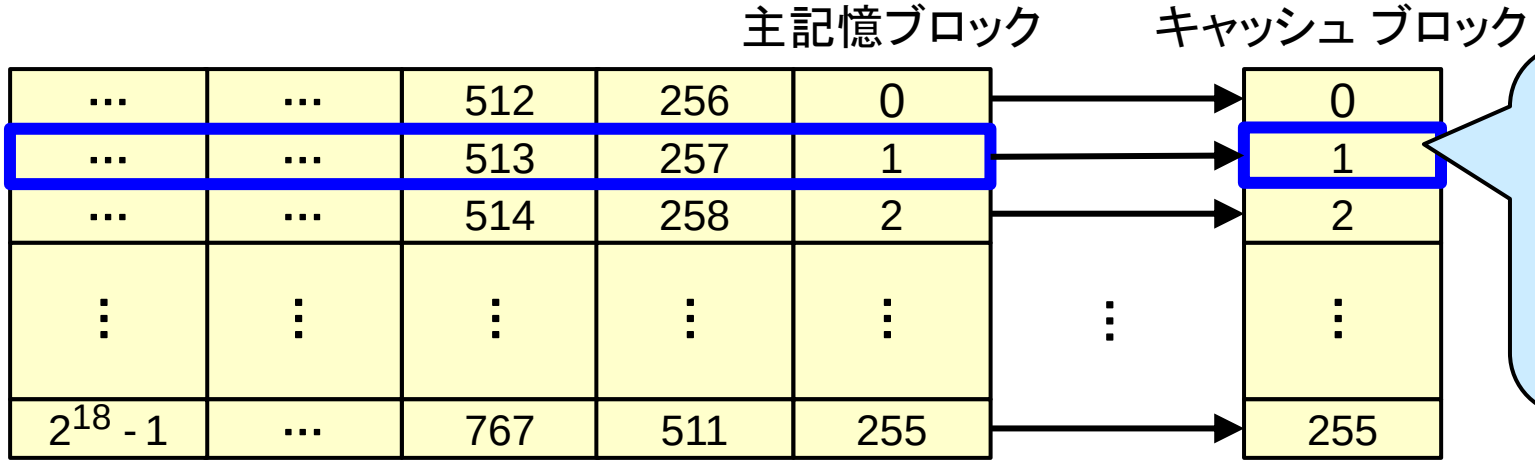
直接法 (direct mapped)

- 主記憶のブロック番地の下位ビットを、
  そのままキャッシュ内のブロック番地に割り当てる



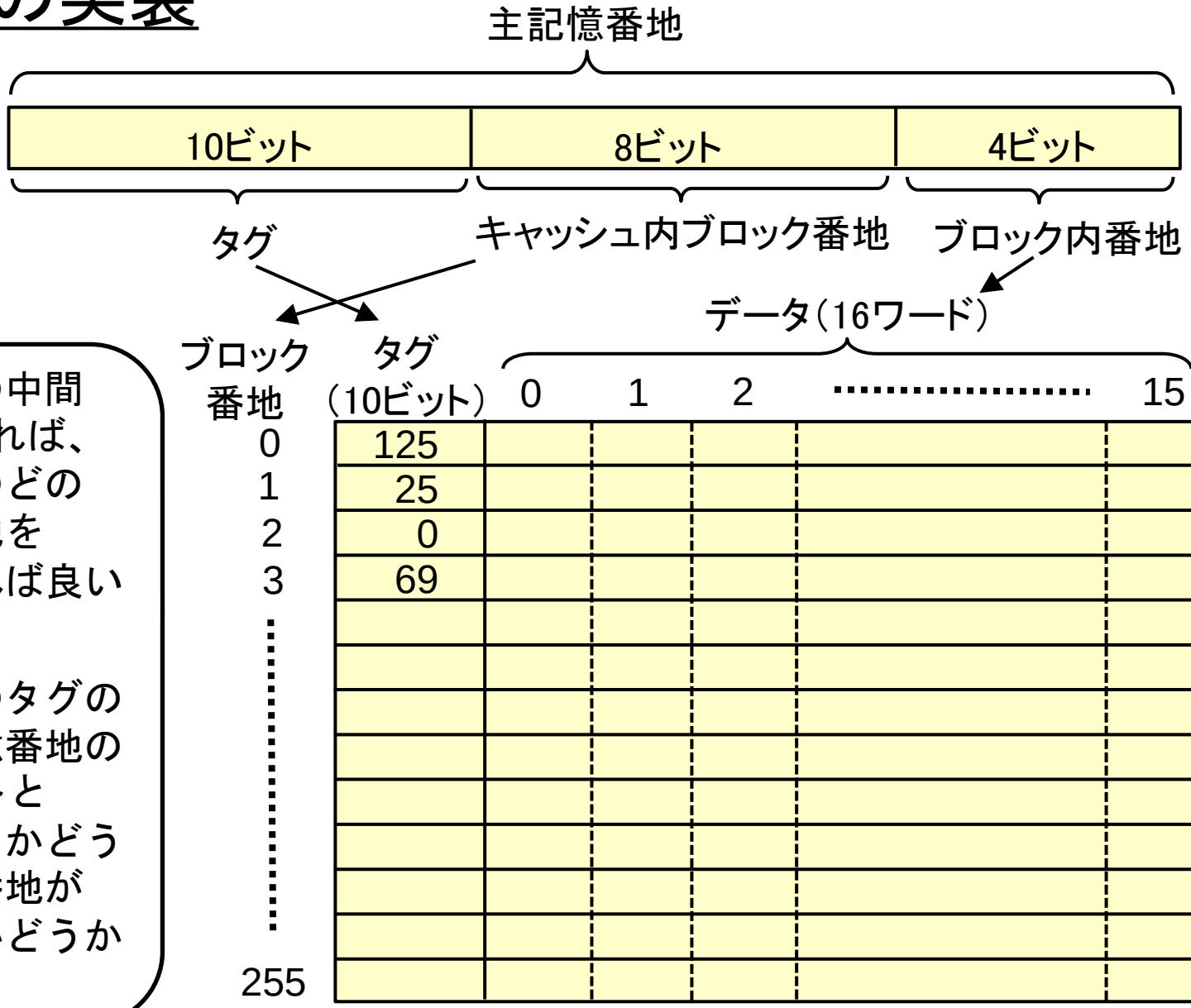
2^{10} = 1024 個の主記憶ブロックが1つのキャッシュブロックを共用

ここが同じ主記憶ブロックは、同じキャッシュブロックに割り当てられる



どの主記憶ブロックを保持しているか、タグの10ビットも記憶しておく
(アクセス時にタグが正しいかを確認する)

直接法の実装

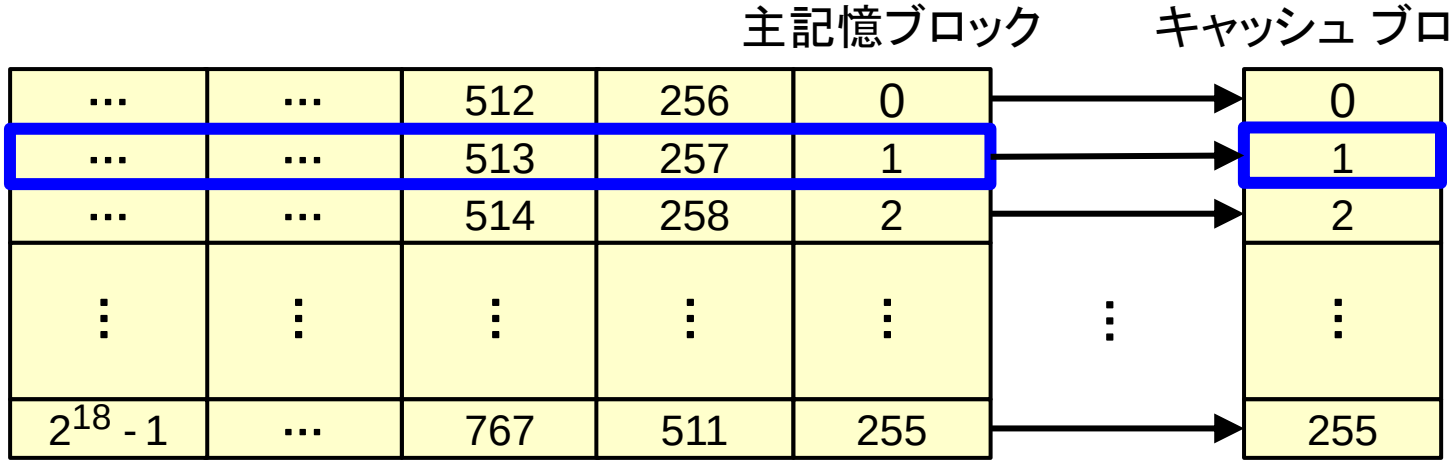


主記憶番地の中間8ビットを見れば、キャッシュのどのブロック番地をチェックすれば良いかが分かる。

次にその行のタグの数値が主記憶番地の上位10ビットと一致しているかどうかで、同じ番地がヒットしたかどうか判定できる。

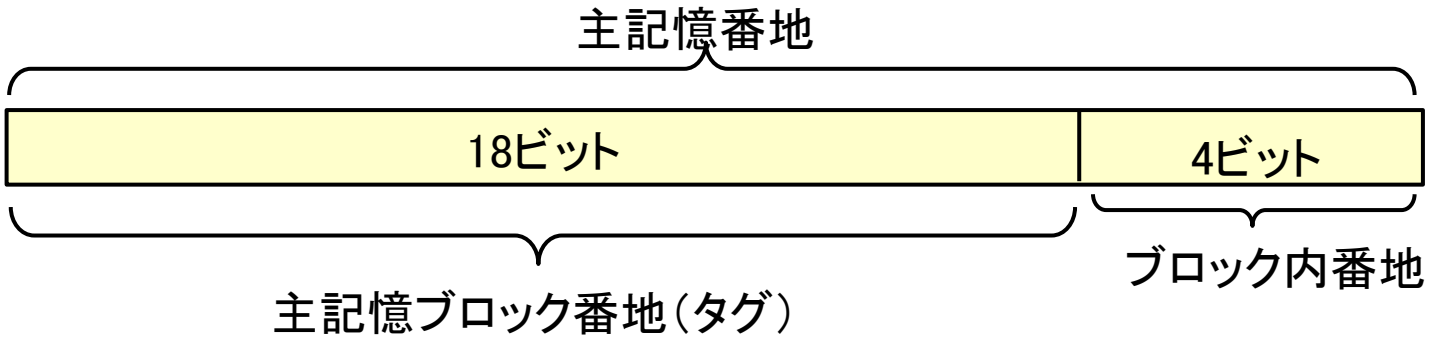
直接法の利点と欠点

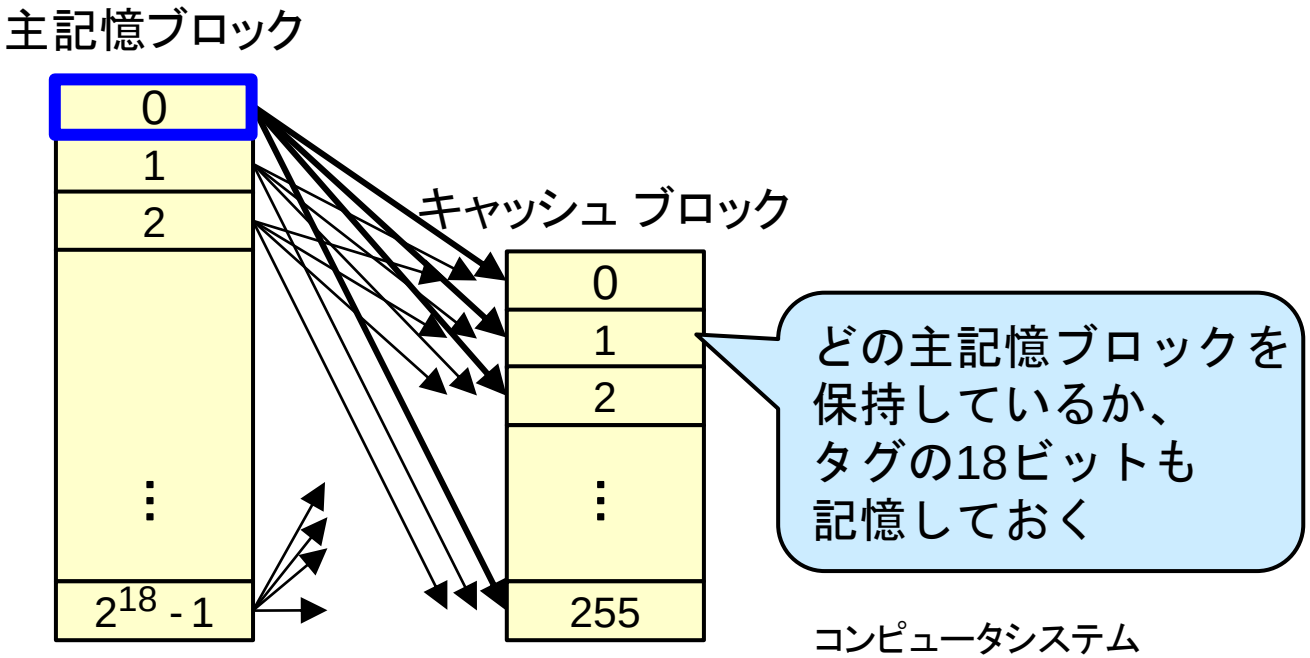
- 利点
  機構が単純なので、ハードウェアだけで容易に実現できる
- 欠点
  主記憶ブロック番地の下位ビットが同じだと、必ず同じキャッシュブロックを共用するので、運が悪いと特定のキャッシュブロックに集中的に上書きが発生する可能性がある
  - キャッシュはヒットすれば大幅に時間を節約できるが、ヒットせずに上書きが頻繁に発生すると、かえって遅くなることもある



完全連想配置法 (full associative)

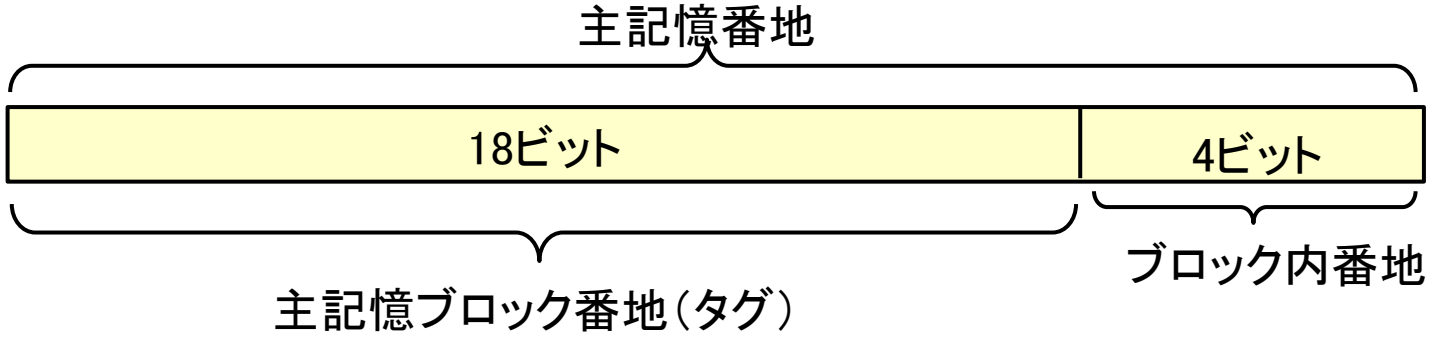
- 主記憶のブロックを、キャッシュの任意のブロックに保持





完全連想配置法 (full associative)

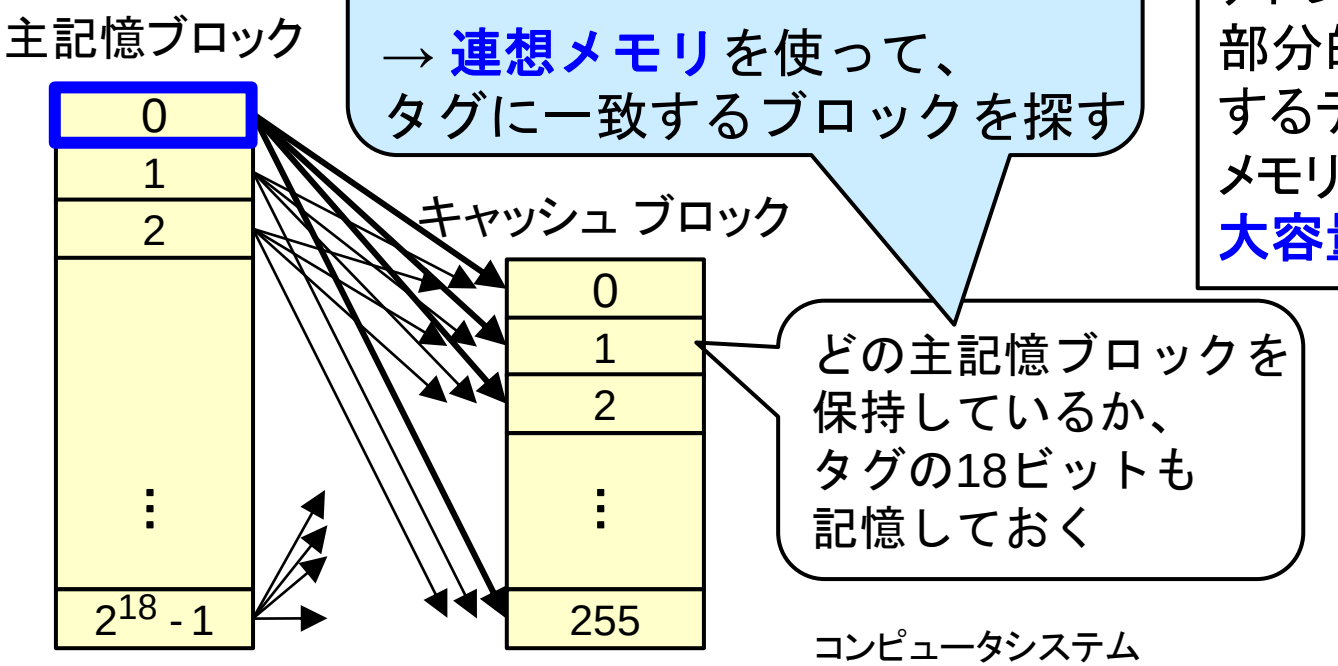
- 主記憶のブロックを、キャッシュの任意のブロックに保持



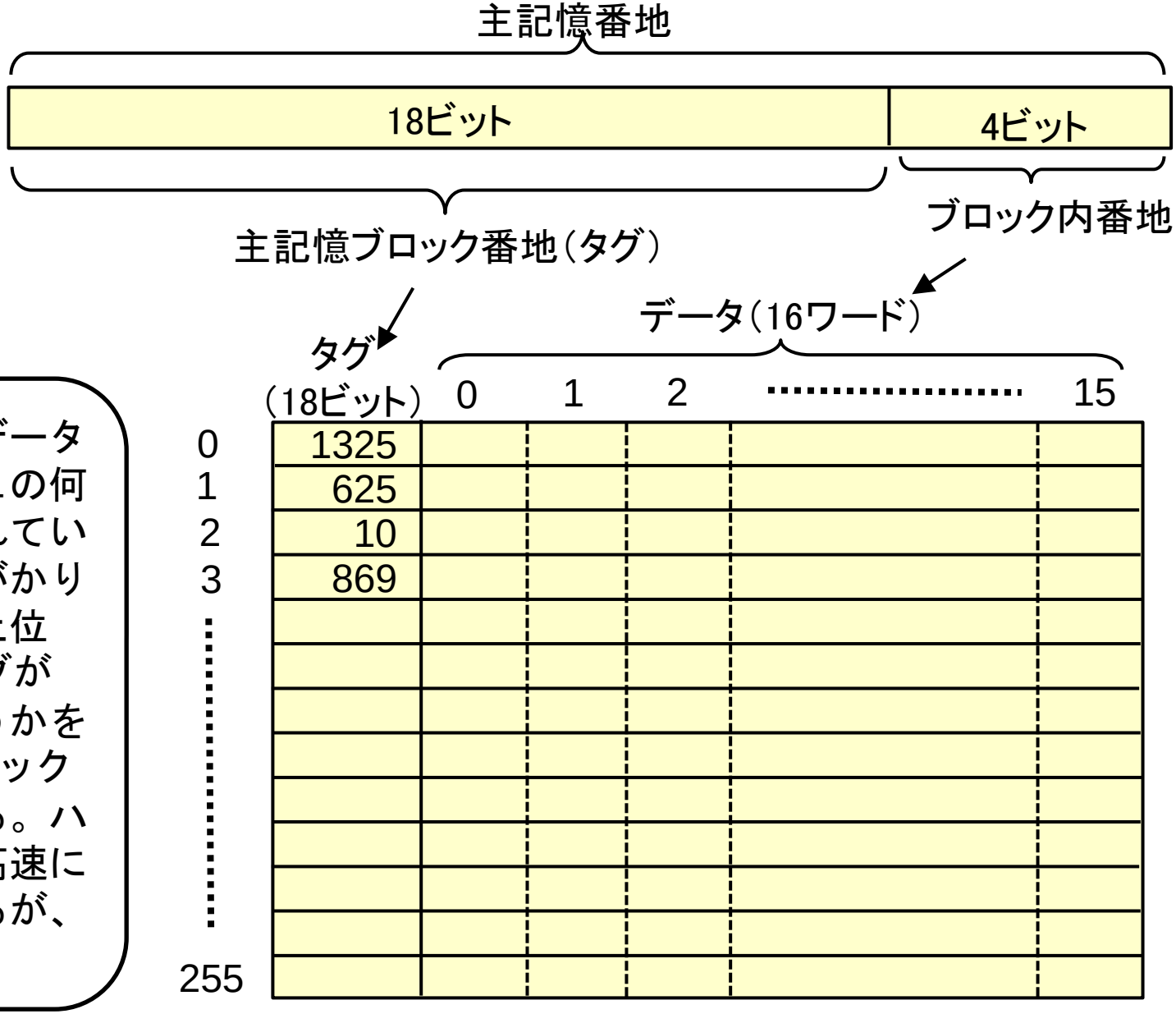
どのキャッシュブロックに
保持しているか、分からない

→ **連想メモリ**を使って、
タグに一致するブロックを探す

連想メモリ
アドレスで参照するのではなく、
部分的なビットパタンを与え、一致
するデータを探し出して出力する
メモリ装置。技術的に難しく、
大容量のものは製造困難・高価



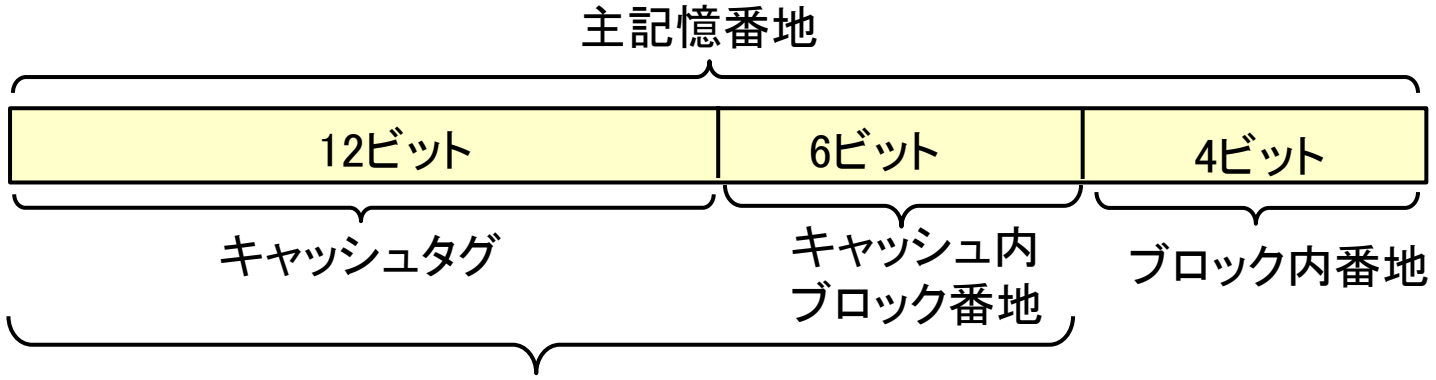
完全連想配置法の実装

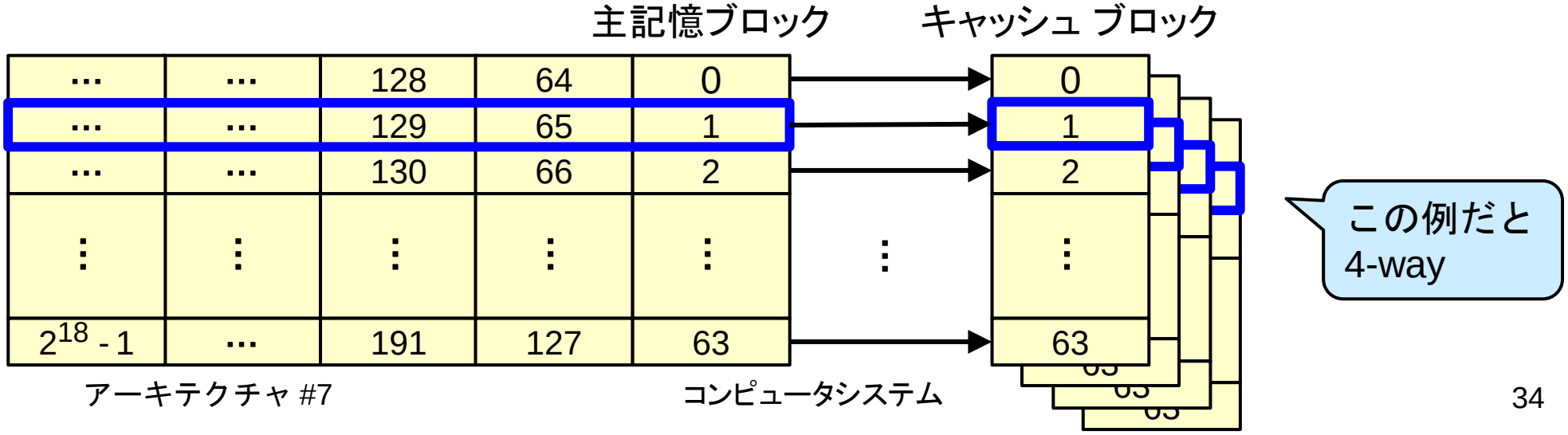


主記憶番地のデータが、キャッシュの何行目に保存されているか、全く手がかりがないので、上位18ビットのタグが一致するかどうかを全256行でチェックする必要がある。ハードウェアで高速に行う必要があるが、難易度が高い。

部分連想配置法 (set associative)

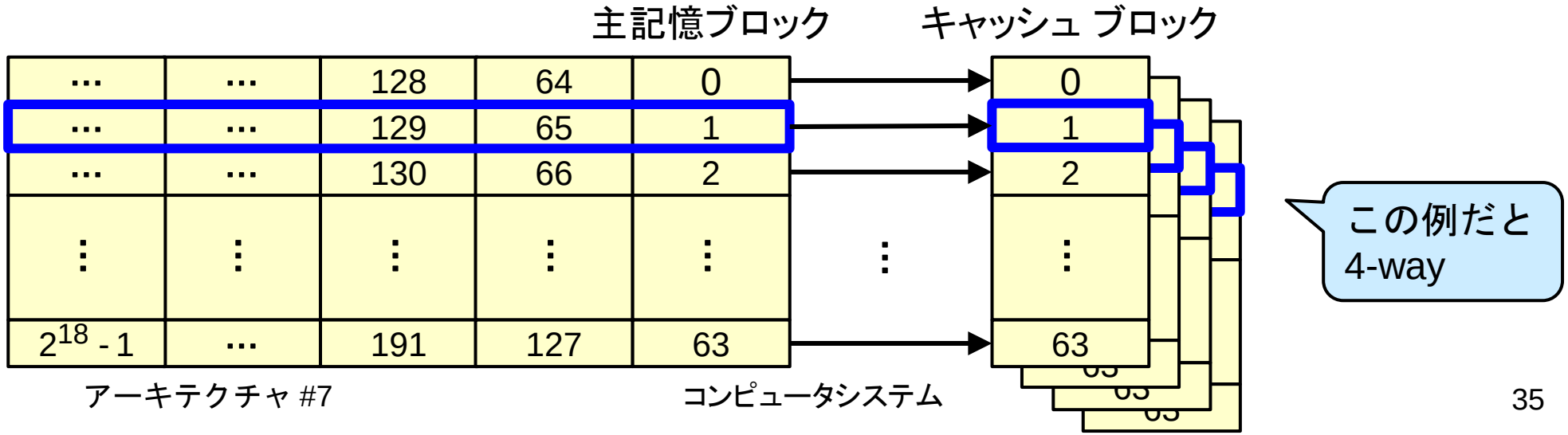
- 直接法と連想配置法を組合せた現実的な方法
  - キャッシュメモリをいくつかのバンク(wayとも呼ばれる)に分割する



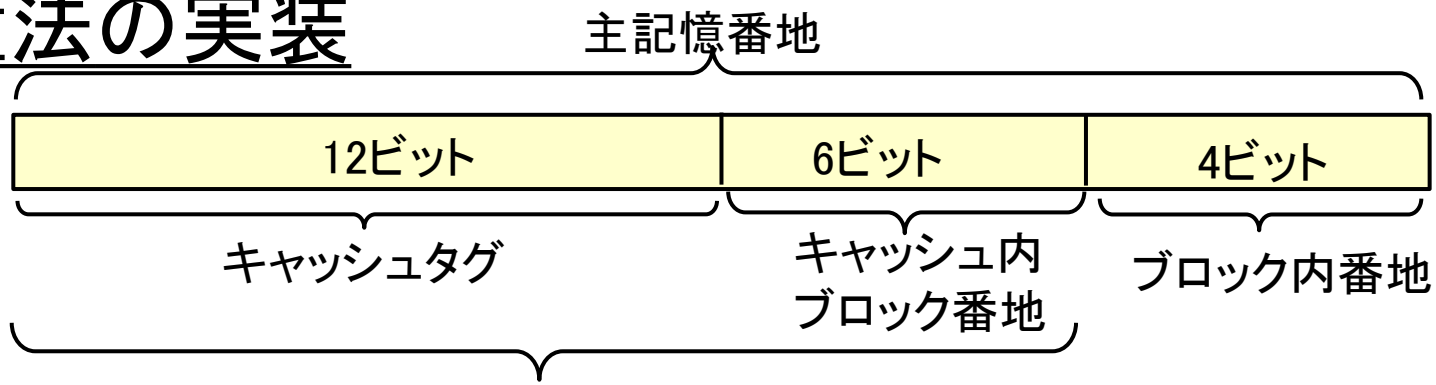


部分連想配置法 (set associative)

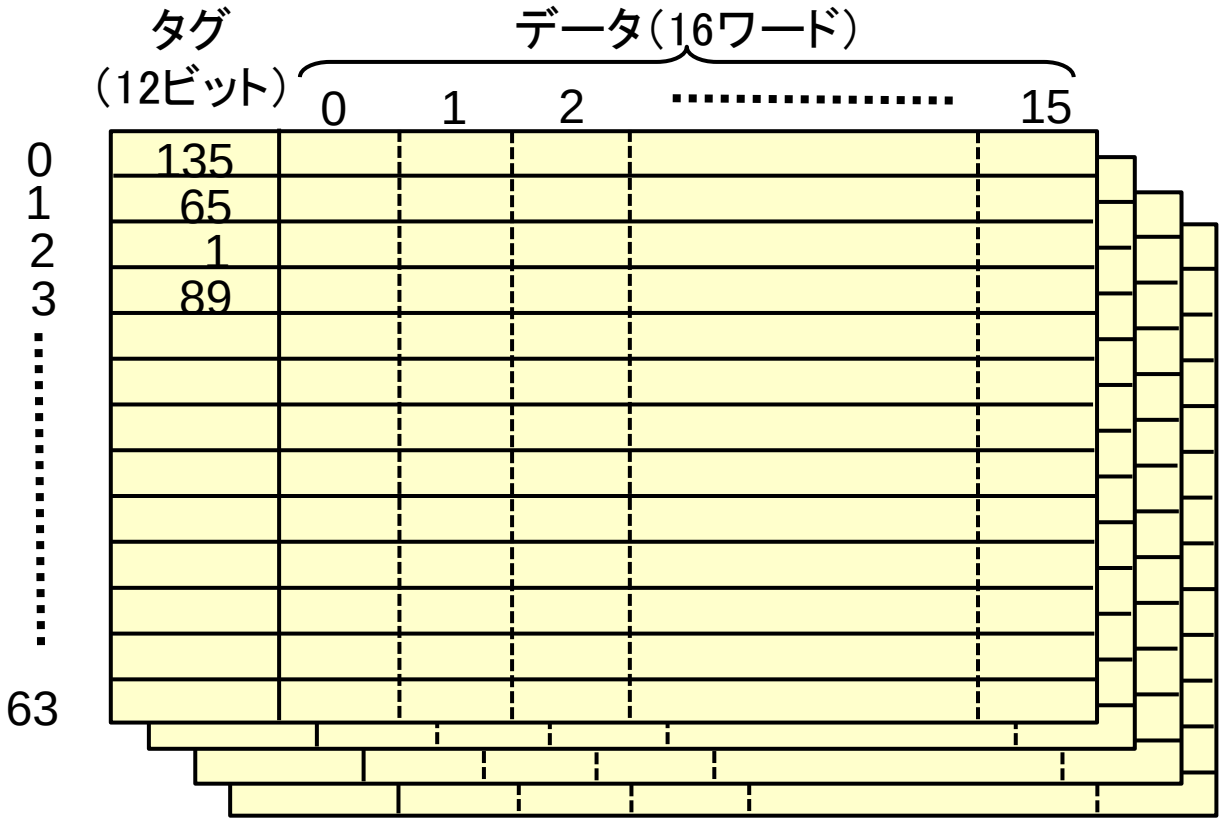
- 直接法と連想配置法を組合せた現実的な方法
  - キャッシュメモリをいくつかのバンク(wayとも呼ばれる)に分割する
  - 主記憶番地が与えられると、各バンク独立に直接法を実施し、該当するキャッシュ内ブロックに保持されているタグを取り出す
  - 各バンクごとにタグの一致判定を行い、いずれかのバンクでタグが一致すれば、そのバンクのキャッシュメモリがヒットしたことになる（バンク数の分だけ連想記憶配置を行ったことになる）
  - どのバンクでもヒットしない → どれかのバンクを選んでブロックを配置



部分連想配置法の実装



全256行のキャッシュを64行×４バンクに分け、それぞれのバンクで直接法で記憶する。どこかのバンクでヒットすればそれを採用し、ヒットしなければどこかの1バンクを選んで上書きする。4バンクの中からヒットしているバンクを探し出すのはハードウェアでも比較的作りやすい



4. ディスク キャッシュ と 仮想メモリ

- キャッシュと主記憶の関係と同様に、
主記憶と補助記憶の間でも階層構造が構成できる
  - **高速小容量と低速大容量の組合せ**で、見かけ上、**高速大容量**
- 補助記憶（ハードディスク）の手前に半導体メモリを置いて、
キャッシュと同様に、補助記憶へのアクセスを高速化する
技法を「**ディスクキャッシュ**」と呼ぶ
  - CPUからは、補助記憶装置上のファイルに高速にアクセスしているように見える

4. ディスク キャッシュ と 仮想メモリ

- キャッシュと主記憶の関係と同様に、
主記憶と補助記憶の間でも階層構造が構成できる
  - **高速小容量と低速大容量の組合せ**で、見かけ上、**高速大容量**
- 補助記憶（ハードディスク）の手前に半導体メモリを置いて、
キャッシュと同様に、補助記憶へのアクセスを高速化する
技法を「**ディスクキャッシュ**」と呼ぶ
  - CPUからは、補助記憶装置上のファイルに高速にアクセスしているように見える
- 主記憶の後ろにハードディスクを置いて、見かけ上、主記憶の容量を増大させる技法を、「**仮想メモリ**」と呼ぶ
  - CPUからは、主記憶（1次元のアドレス空間）として見える

仮想メモリの目的と基本構造

(1) 十分に大きな空間 (論理空間) をプログラマに提供する

(2) 論理的には連続領域としてプログラマに見えているが、
物理的には一部分を切り出して主記憶上に置いている

仮想メモリの目的と基本構造

(1) 十分に大きな空間 (論理空間) をプログラマに提供する

(2) 論理的には連続領域としてプログラマに見えているが、
物理的には一部分を切り出して主記憶上に置いている

- 記憶領域は同じ大きさのページと呼ばれる単位に区切られる
- 補助記憶と主記憶の間はページ単位で転送される
- 主記憶上にないページへのアクセスは、ページフォールトと呼ばれ、
主記憶のどこかのページを補助記憶に退避した上で、
必要なページを補助記憶からロードする
- アドレス変換と、どのページを置き換えるかを選ぶ技法が、
それぞれ開発されている
  - ハードウェア実現する場合と、OSでソフトウェア的に実現する場合がある

今回のまとめ

メモリ アーキテクチャ

1. 記憶装置の容量とアクセス速度
2. 参照の局所性とメモリ階層
   - 空間的局所性、時間的局所性
   - メモリの階層構造
3. 主記憶へのアクセスの高速化
   - メモリ インターリーブ
   - キャッシュ メモリ
4. ディスクキャッシュと仮想メモリ

アーキテクチャ講義内容のまとめ

- 第１回　計算機アーキテクチャとは
- 第２回　機械語命令と内部動作
- 第３回　アーキテクチャの基本知識（１）
- 第４回　アーキテクチャの基本知識（２）
- 第５回　アーキテクチャの基本知識（３）
- 第６回　並列処理アーキテクチャ
- 第７回　メモリ アーキテクチャ

計算機アーキテクチャ技術（主にハードウェア）を概観

計算機はどのように工夫され、今日の形になったのか