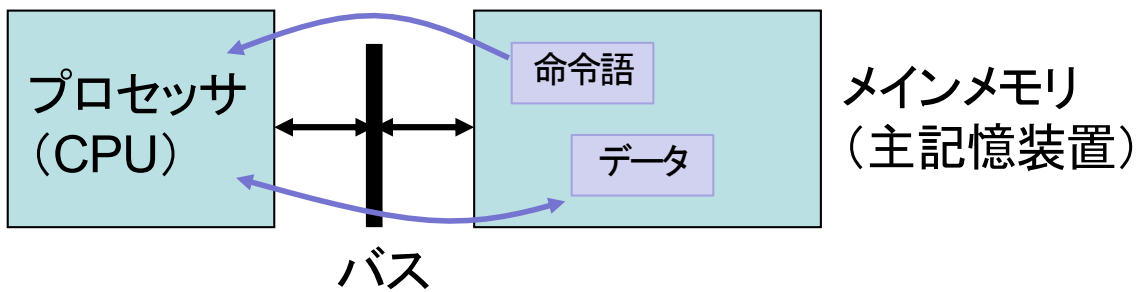
はじめに

- 今回はキャッシュメモリについて講義します。

- 教科書の対応範囲は、以下の通りです。
  - 7.3節はじめ～7.3.3項(g)（p.264～279）
    - この講義資料で解説します。
  - 7.3.3項(j)（p.282）
    - あえて解説はしません。最後に、読んでおいてください。

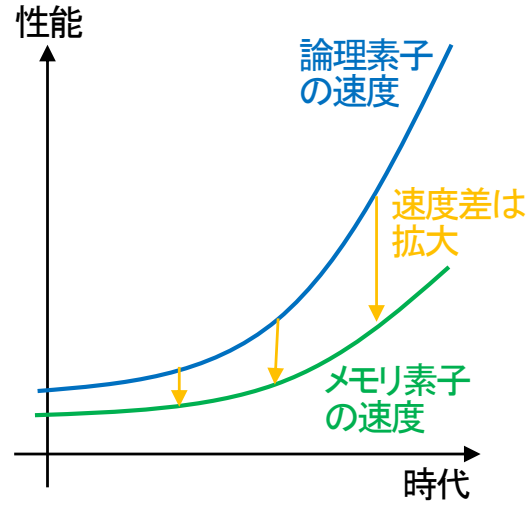
フォンノイマンボトルネック（再）

- プロセッサだけが速くても、コンピュータの性能は上がりません。マシン命令実行に必要な命令語やデータはメインメモリにあります。したがって、メインメモリのアクセス性能も含めて、プロセッサ-メインメモリ間のデータ転送性能がコンピュータの性能を決定する重要な要因となります。



- ところが、、、

- まず、使用している素子の性能として、論理素子の方がメモリ素子よりも速い。
- さらに、論理素子もメモリ素子も時代とともに高速化されてきているが、その性能向上には、右図のような違いがある。
  - その結果、プロセッサから相対的に見て、メインメモリは以前よりもどんどん遅くなってきています。
    - 現在、プロセッサからメインメモリにアクセスしようとすると、プロセッサは100サイクル以上待たなければなりません。



- これではコンピュータの性能を改善することなど無理です。なんとかして、プロセッサとメモリの速度差を埋めなければなりません。
  - そこで、考え出されたのがキャッシュ（cache）メモリです。

キャッシュとは？

- メインメモリの時間性能（速度）を改善するのが目的です。
  - とは言っても、速くならないものは速くなりません。
  - 見かけ上のアクセス時間を短縮します。
- プロセッサとメインメモリの中間に、メインメモリより高速で小容量のメモリを設置します。
  - その高速で小容量のメモリをキャッシュと呼びます。
  - 現在は、プロセッサ内のレジスタはフリップフロップで、キャッシュはSRAMで、メインメモリはDRAMで、それぞれ構成しています。
- 参照の局所性を利用して、キャッシュには、頻繁にアクセスする命令/データを格納します。

- 例えば、データを読み出すときの動作の概要は以下の通りです。
  1. プロセッサがデータのアドレスを出力して、読み出しを要求します。
  2. そのアドレスのデータが、キャッシュ内に格納されているかどうかをチェックします。
  3. キャッシュ内に格納されていれば（ヒット；hit）、そのデータをプロセッサに渡して、操作完了です。
     格納されていなければ（ミスヒット;miss-hitまたは単にミス）、メインメモリに読み出しを要求します。
  4. メインメモリから読み出したデータをキャッシュに格納し、プロセッサにも渡します。

上記2.～4.の動作は、キャッシュコントローラが行います。

キャッシュの性能指標

- キャッシュの性能指標として、以下の2つがあります。
  - ヒット率（hit ratio）： $R = H / A$
    - Aはプロセッサがメモリアクセスした回数（総数）
    - Hは、そのうちでキャッシュにヒットした回数

    - 当然、ヒット率が高い方が性能は良くなります。
      - プログラムによりますが、実際には、だいたい、
        » データアクセスの場合で70%以上、
        » 命令アクセスの場合で99%程度
        のヒット率があります。

– ミスペナルティ時間（miss penalty time/cycle）:

$$T_p = T_m - T_h$$

- T_hはヒット時のアクセス時間
- T_mはミス時のアクセス時間

- ヒット時に比べて、ミスした場合にアクセス時間がどれだけ余分にかかるか、を示す指標です。

- キャッシュがある場合のアクセス時間（実効アクセス時間）Tは、以下のようになります。
  - $T = T_h \times R + T_m \times (1 - R)$
    - Rが1に近くなるほど、TはT_hに近づきます。つまり、見かけ上、メインメモリのアクセス時間が小さくなります。

キャッシュのハードウェア構成

- キャッシュは単なるメモリではありません。
- 以下のようなハードウェアで構成します。
  - データアレイ（data array）
    - メインメモリのデータのコピーを格納するメモリ
  - タグアレイ（tag array）
    - キャッシュ（データアレイ）に格納しているデータの管理情報を格納するメモリ
  - キャッシュコントローラ（cache controller）
    - プロセッサからのアクセス要求を受け付け、ヒット/ミスを判定するとともに、ミス時にはメインメモリにアクセスします。

キャッシュ機構

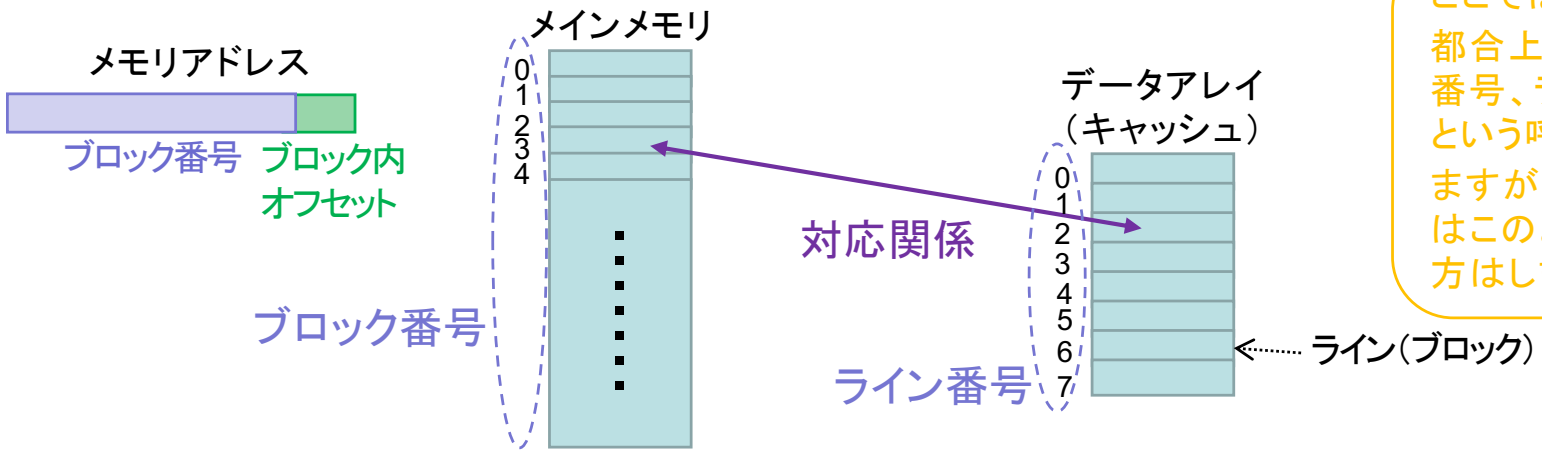
- キャッシュは、メインメモリアクセスの時間性能を改善するのが目的ですから、キャッシュにヒットしたかどうかを判定するのに時間をかけていたのでは話になりません。
- 即座に（例えば、1サイクルで）ヒットかミスかを判定する必要があります。
- この目的で、どのようにキャッシュデータを管理し、キャッシュを制御するかを考えていきます。

キャッシュデータの管理単位

- 固定長のブロック（キャッシュラインと呼ぶ）で管理します。
  - 処理に手間がかかるので、可変長にはしません。
  - 一般に、ラインサイズは16～128バイト（もちろん2のべき）程度が採用されます。
  - このライン単位で、キャッシュにデータを格納します。
    - 例えば、ラインサイズが8バイトの場合、12～13番地の2バイトのデータを読み出そうとしてミスすると、メインメモリからこの2バイトのデータだけを読み出すのではなく、8～15番地の8バイトのデータを読み出してキャッシュに格納します。
    - もちろん、キャッシュラインはメインメモリとキャッシュとの間でのデータの転送単位にもなります。

連想方式

- メインメモリとキャッシュのデータアレイを、キャッシュのラインサイズで分割して考えます。
  - 仮想メモリのページング方式の場合と同様の考え方です。
    - しかし、すべてをハードウェアで管理・制御しなければならないので、高速に処理できて、また、管理情報のサイズ（つまりタグアレイの容量）が小さくなるように設計しなければなりません。
- そして、分割したメインメモリのブロックを、キャッシュのデータアレイのどのライン（ブロック）に格納するのか、その対応関係を定めます。



ここでは、説明の都合上、ブロック番号、ライン番号という呼び方をしますが、一般にはこのような呼び方はしません。

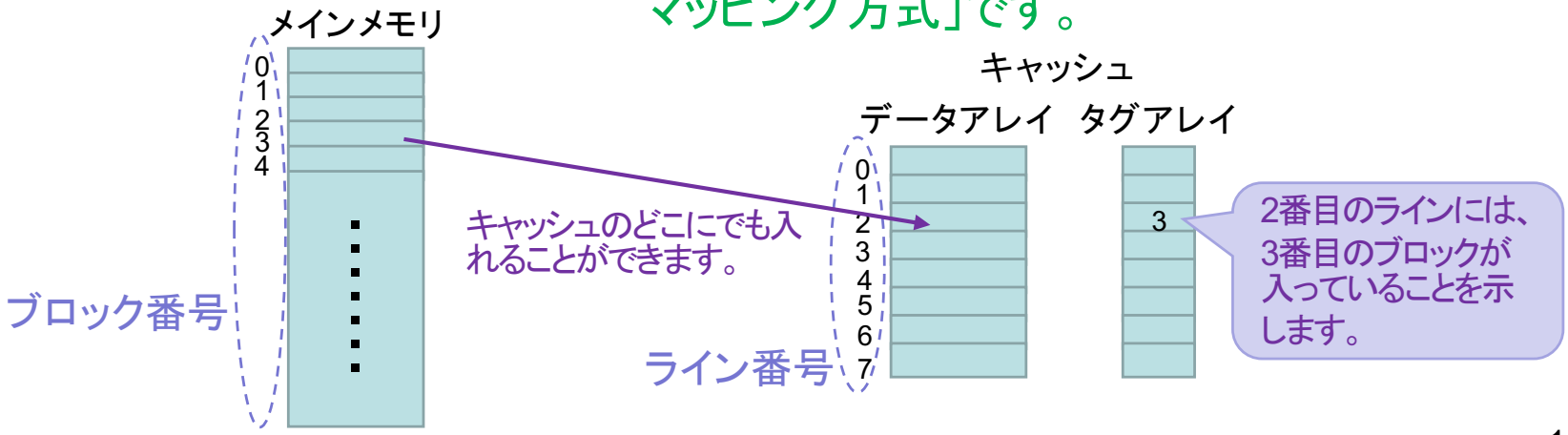
- では、具体的な連想方式について、次ページから順に見ていきます。
  - なお、簡単のため、以下の例で考えることにします。
    - »（実際のメインメモリやキャッシュの容量はこんなに小さくはありません）。
    - キャッシュのラインサイズは8バイト（ライン内オフセットのアドレスは3ビット）。
    - キャッシュ容量は64バイト。よって、ライン数は64÷8=8個。
    - メモリアドレスは10ビット。よって、メインメモリのサイズは1024(=1K)バイト。1024÷8=128ブロック（ライン）分のサイズです。

- 連想方式には、以下の3つがあります。
  - フルアソシアティブ（Full Associative）方式
  - ダイレクトマッピング（Direct Mapping）方式
  - セットアソシアティブ（Set Associative）方式

フルアソシアティブ方式

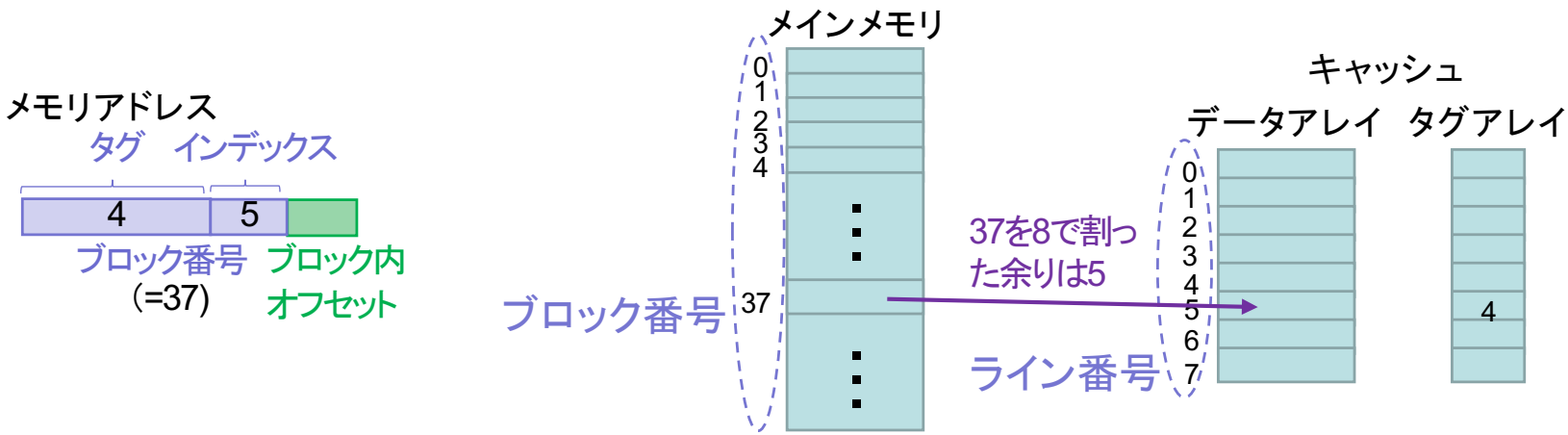
- メインメモリのブロックを、キャッシュのどのラインにも入れることができます。
  - 空いているところに入れられるので、データアレイの領域を無駄なく使うことができます。
  - ヒット/ミスの判定にはコスト（時間またはハードウェア量）がかかります。
    - ブロック番号をタグとして、タグアレイに記憶します。ヒット/ミスの判定には、ブロック番号をタグアレイ中の全てのタグと比較しなければなりません。

このコストを下げるのが、次の「ダイレクトマッピング方式」です。

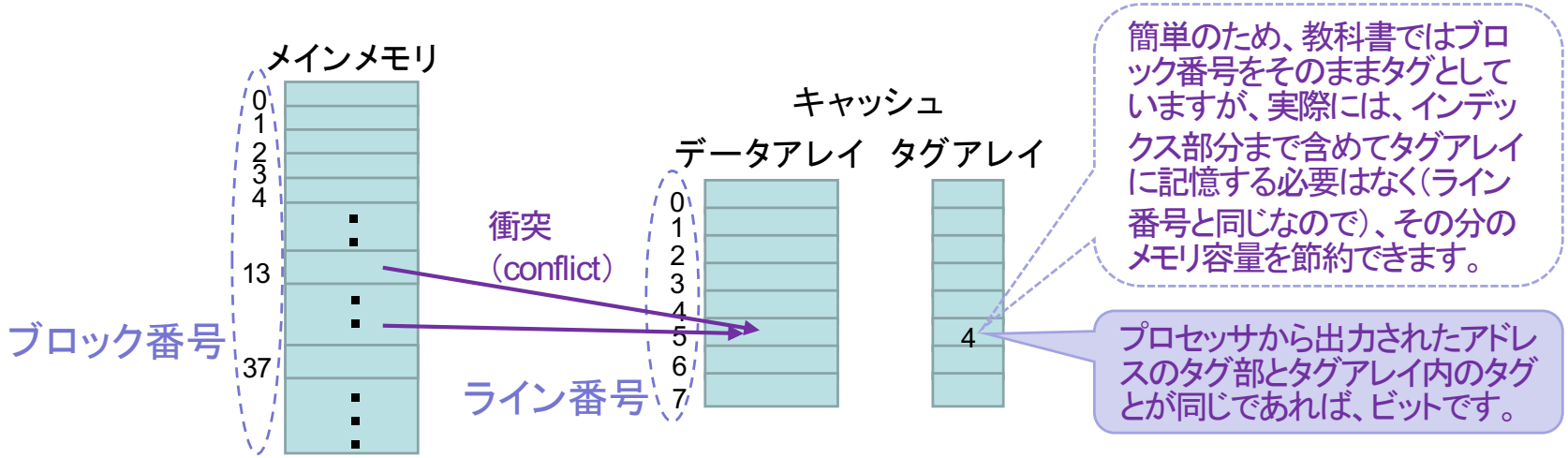


ダイレクトマッピング方式

- ブロック番号の下位ビットで、キャッシュのどのラインに入れかを決定します。
  - メインメモリのアドレスを、タグ、インデックス、ライン内オフセット、に分割します。
    - インデックスがライン番号を指します。
  - つまり、ブロック番号をライン数で割った余り（剰余）をライン番号として、そこにブロックデータを入れます。
    - 例えば、ブロック番号が37のブロックは5(=37%8)番目のラインに格納します。このとき、対応するタグアレイには4(=37/4)を記憶します。



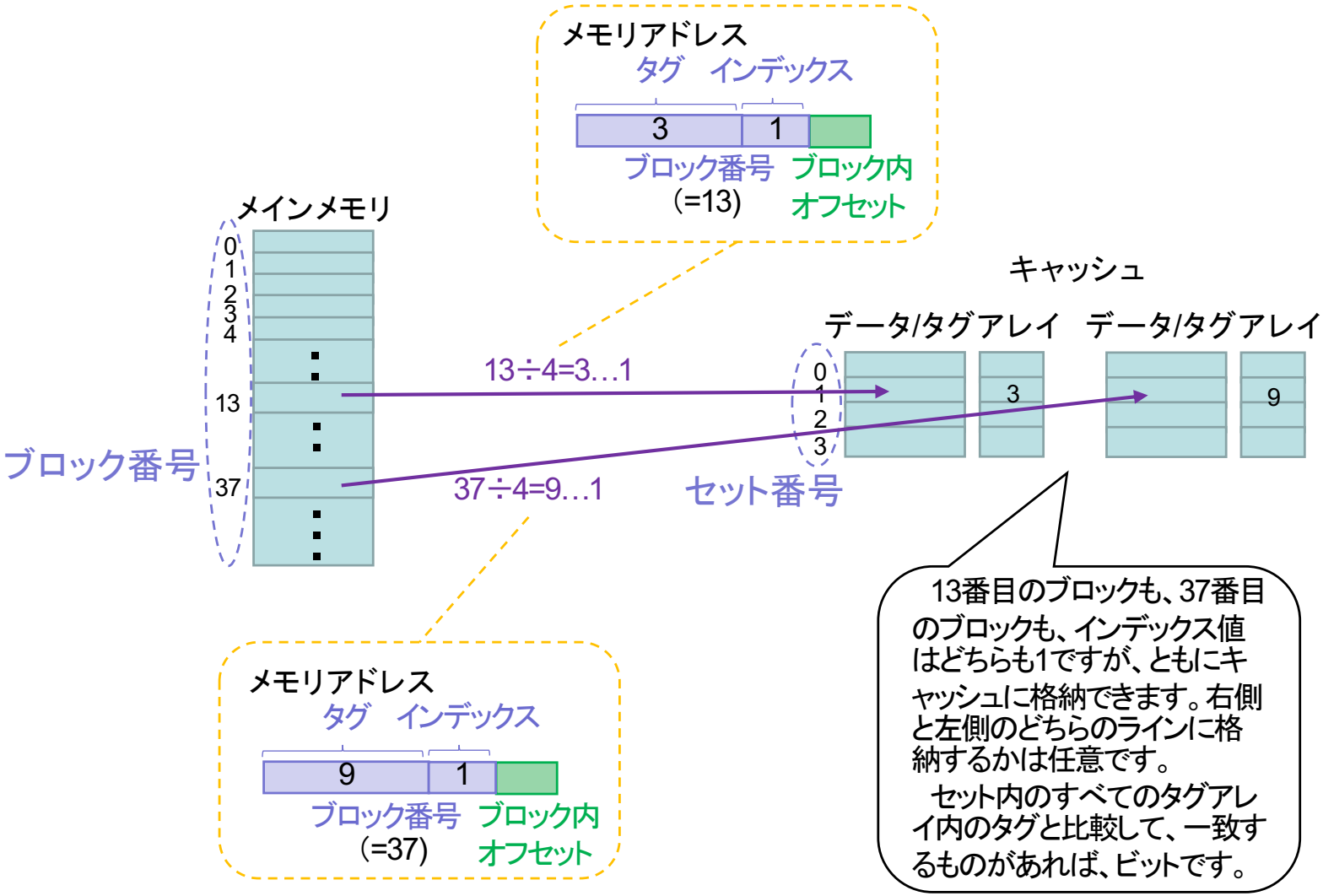
- ヒット/ミスの判定は、インデックスからライン番号を割り出し、そのタグ（1個だけ）を比較するだけです。
  - フルアソシアティブ方式に比べると、必要な比較器は1個だけで、また、高速に判定できます。
- しかし、マッピングコンフリクト（mapping conflict）が発生します。
  - 例えば、13番目のブロックと、37番目のブロックは、ともに5番目のラインに割り当てられるので、この2つのブロックを同時にキャッシュに格納することができません。（たとえ、隣の6番目のラインが空いていたとしても、そこには格納できません。）
  - したがって、データアレイの利用効率は下がってしまします。



セットアソシアティブ方式

- フルアソシアティブ方式とダイレクトマッピング方式の中間的な方式です。
  - 同じインデックス値を持つブロックをキャッシュに複数個格納します。
    - 複数のブロックをまとめて、1つのセットとして定義します。1つのセット内のブロック数がkのとき、kウェイ（k-way）のセットアソシアティブと言います。
    - インデックスでセットの番号を指定します。したがって、セット単位では、ダイレクトマッピングとなります。
    - セット内ではフルアソシアティブです。セット内のどのラインにでも格納できます。

- 以下は、2-wayの場合の例です。



ライン置換

- キャッシュミスが発生して新たなブロックをキャッシュにコピーする場合に、その格納場所が空いていない場合があります。この場合、どれかのラインのデータを消去しなければなりません。
  - ダイレクトマッピング方式なら格納場所は1つのラインに決まります。
  - セットアソシアティブ/フルアソシアティブの場合は複数のラインが対象となります。置き換え対象の(消去する)ラインの選択を誤るとヒット率が低下するので、参照の局所性が活かせるような方法でうまく制御する必要があります。

メインメモリ更新機能

- メインメモリとキャッシュのコヒーレンシ（coherency、一貫性/無矛盾性）について考えます。
  - メインメモリとキャッシュとの間で、記憶内容が一致しているか否か、という問題です。
  - メモリアクセスには、読み出しアクセスだけでなく、書き込みアクセスもあります。データをキャッシュだけに書き込むと、メモリのデータと値が一致しなくなって（一貫性がなくなって）しまいます。
- メインメモリの更新方式には、以下の2つがあります。
  - ライトスルー（write-through）方式
  - コピーバック（copy-back）/ライトバック（write-back）方式

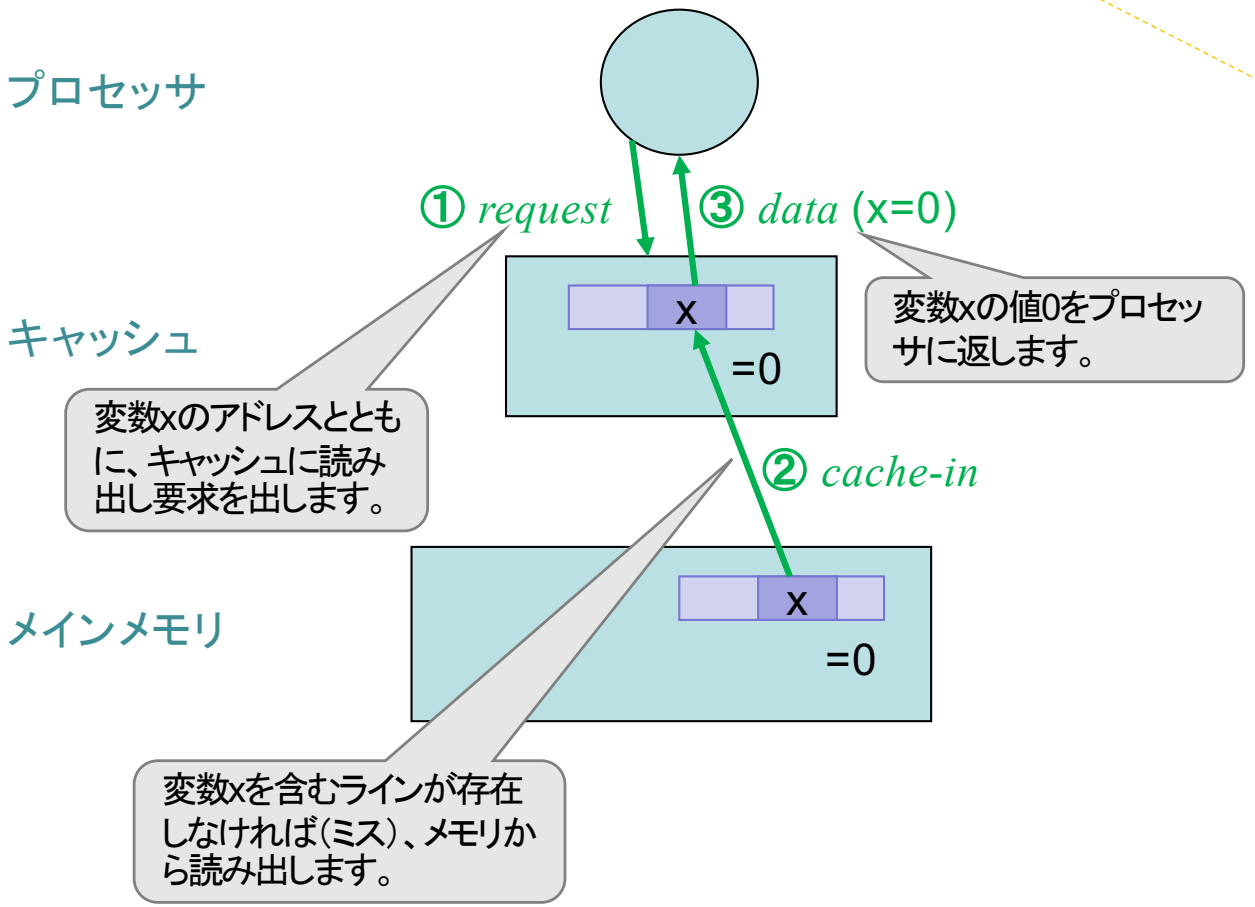
ライトスルー方式

- キャッシュに書き込むのと同時に、メインメモリも更新します。
  - したがって、常にコヒーレンシが保たれています。
- しかし、メインメモリアクセスにより、性能は低下してしまいます。
  - そもそも、メインメモリのアクセス時間を短縮するのがキャッシュの目的でしたから、毎回、メインメモリにアクセスするのでは、キャッシュを設ける意味がありません。
- そこで、ライトバッファ（write buffer）をキャッシュに併設します。
  - メインメモリに直接書くのではなく、キュー（待ち行列）構造のライトバッファに書き込みます。ライトバッファに書き込まれた値は、そのうちにメインメモリに書き込まれます。
  - ライトバッファに書き込んだ時点で、プロセッサはメインメモリに書き込んだものとして、次の動作に移ります。
    - メインメモリへの書き込みが完了するまで待つ必要がありませんので、性能低下は防げます。
  - ただし、メモリに書き込んだデータを再び読み出す場合は、ライトバッファ内にまだ残っていないか検索する必要があります。

コピーバック方式

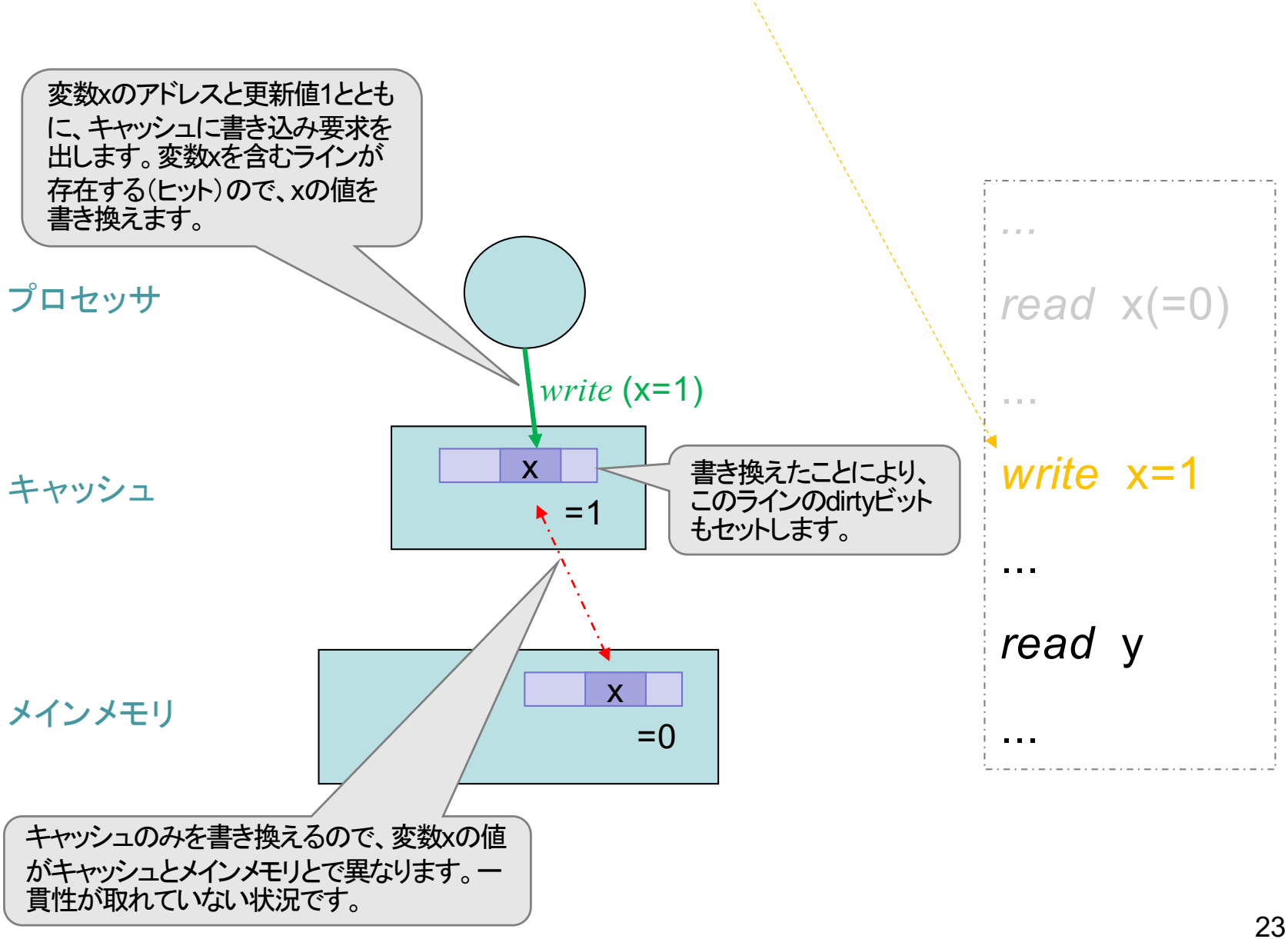
- キャッシュのみに書き込みます。
  - したがって、コヒーレンシが保たれていない期間が生じます。
- キャッシュから追い出す（置換する）ときに、そのラインのデータをメインメモリに書き戻します。
  - これによってコヒーレンシを保ちます。
  - ライン置換時に書き戻しが必要になるため、ミスペナルティ時間が大きくなります。
  - ただし、書き込んでいないクリーン（clean）な状態のラインまで書き戻す必要はありません（すでにコヒーレンシが保たれています）。
    - そのため、ライン単位で、書き込みを行ったか否かを記録します。
    - タグアレイ内にdirtyビットを設け、書き込みを行うときに、dirtyビットをセットします。
    - 置換対象のラインのdirtyビットがセットされている場合のみ、ラインデータをメインメモリに書き戻します。

- では、コピーバック方式のキャッシュの動作を追いかけてみます。
  - 右は、プロセッサで実行するプログラムで、メモリアクセスについてのみ記しています。
  1. まず、変数xの値を読み出します。



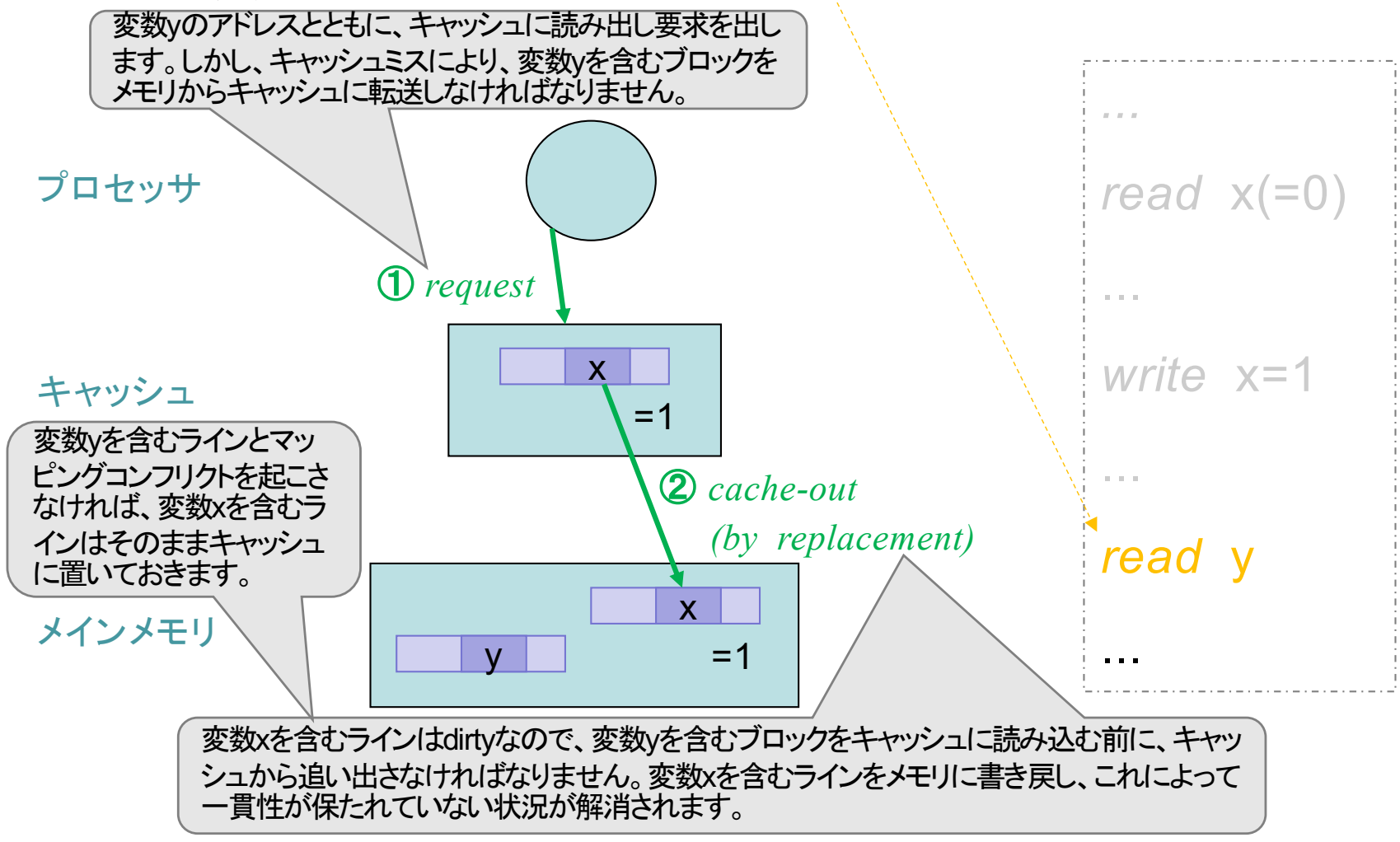

```
…
read  x(=0)
…
write  x=1
…
read  y
…
```

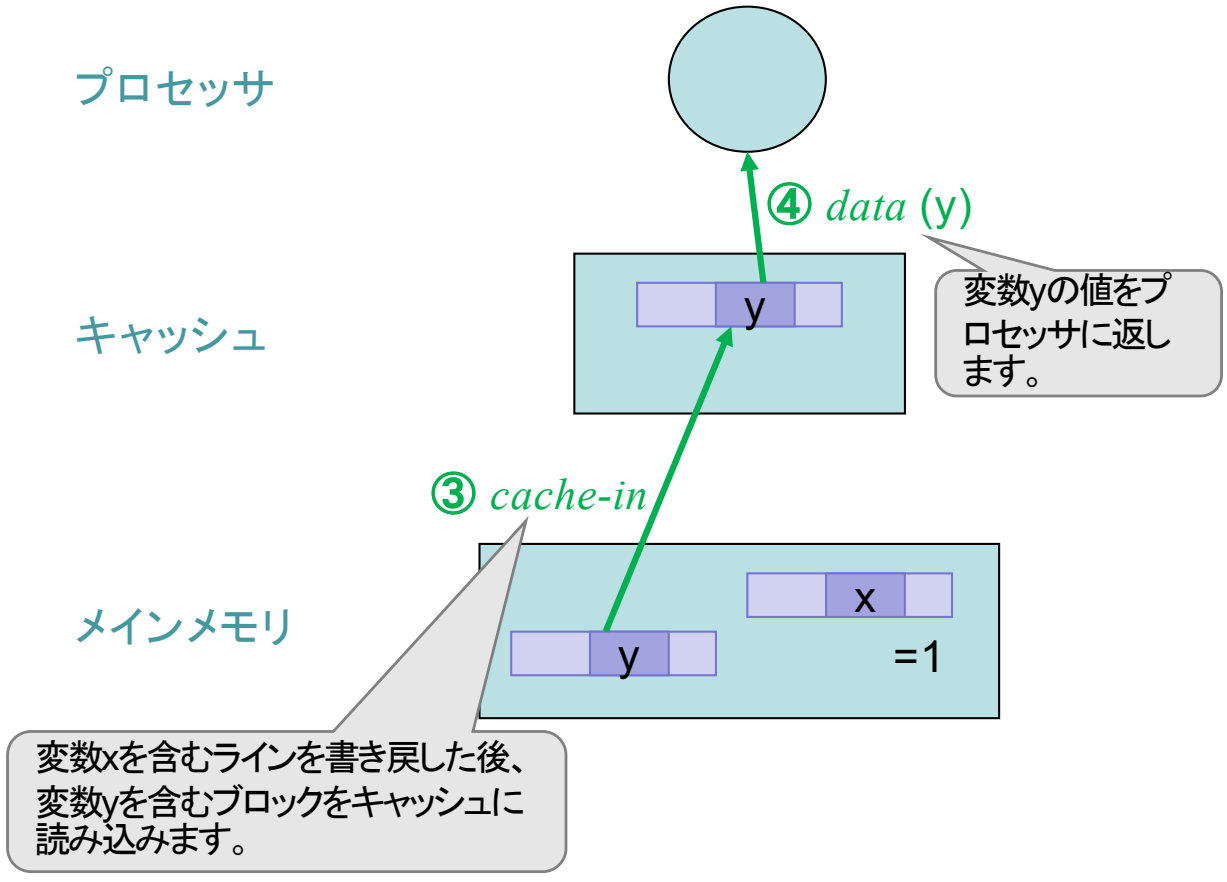
2. 次に、変数xの値を更新します。



3. 変数yの値を読み出します。

- 変数xと変数yは独立な変数ですが、ここでは、変数xを含むラインと変数yを含むラインとがマッピングコンフリクトを起こすものとします。



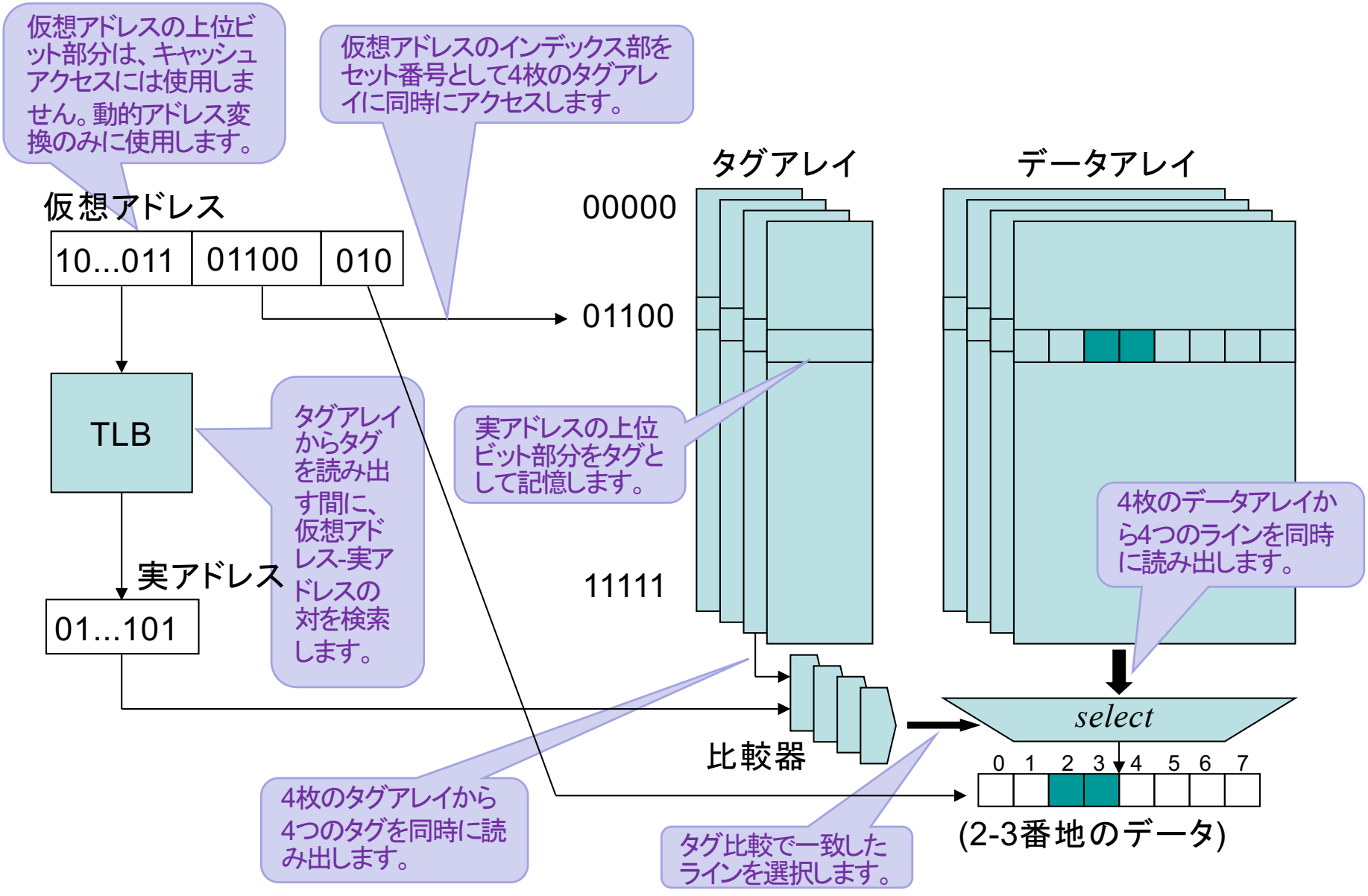



```
…
read  x(=0)
…
write  x=1
…
read  y
…
```

論理キャッシュと物理キャッシュ

- これまでの解説では、“メインメモリのアドレス”が仮想アドレスなのか実アドレスなのかを明らかにしていませんでした。
- 仮想アドレスを用いてキャッシュ検索を行うキャッシュを、論理キャッシュといいます。
  - キャッシュにヒットする場合の処理は高速に行えるので、キャッシュのアクセス時間が短い利点があります。
  - プロセスを切り替えるごとに、キャッシュしたラインのすべてを無効化（フラッシュ; flush）する必要があります。
  - そのほか、複数のプロセッサを搭載するマルチプロセッサや、仮想メモリとの関係で不都合な点がいくつかあり、現在は使用されません。

- 実アドレスを用いてキャッシュ検索を行うキャッシュを、物理キャッシュといいます。
  - キャッシュにアクセスする前に、仮想アドレスから実アドレスへのアドレス変換を行う必要があり、それらも含めた全体のキャッシュアクセス時間が長くなってしまいます。
- そこで、考え出されたのがLP（logically indexing and physically tag-comparison）キャッシュです。
  - 仮想アドレスでインデキシングを行い、実アドレスでタグ比較を行います。
  - キャッシュのタグアレイへのアクセス（読み出し）とTLBによる動的アドレス変換を同時に行うことで、物理キャッシュの問題点（長いアクセス時間）を解決します。
  - 次ページの図は、4-wayのセットアソシアティブ方式を採用したLPキャッシュのハードウェア構成です。
    - 2バイトのデータを読み出す場合を例示しています。


仮想アドレスの上位ビット部分は、キャッシュアクセスには使用しません。動的アドレス変換のみに使用します。
仮想アドレスのインデックス部をセット番号として4枚のタグアレイに同時にアクセスします。
タグアレイ
データアレイ
仮想アドレス
10...011
01100
010
00000
01100
11111
TLB
タグアレイからタグを読み出す間に、仮想アドレス-実アドレスの対を検索します。
実アドレスの上位ビット部分をタグとして記憶します。
4枚のデータアレイから4つのラインを同時に読み出します。
実アドレス
01...101
比較器
select
0 1 2 3 4 5 6 7
4枚のタグアレイから4つのタグを同時に読み出します。
タグ比較で一致したラインを選択します。
(2-3番地のデータ)

キャッシュ設計におけるトレードオフ

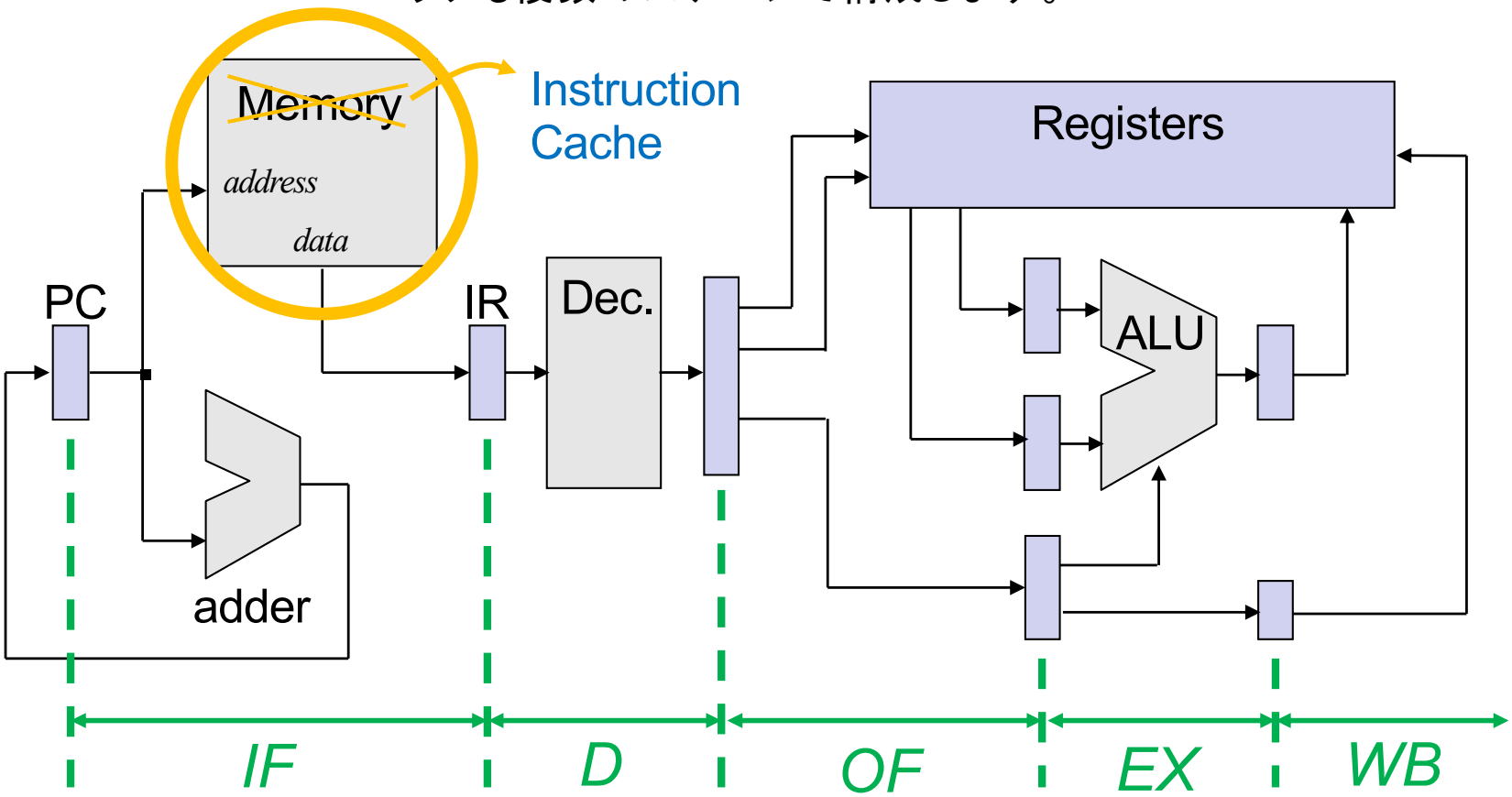
- キャッシュ容量と性能
  - キャッシュ容量が大きいほど、ヒット率は上がります。
  - キャッシュ容量が大きいほど、キャッシュのアクセス時間が長くなります。
- 連想度と性能
  - 連想度が高いほど、ヒット率は上がります。
  - 連想度が高いほど、ハードウェア機構は複雑になり、また、キャッシュのアクセス時間も長くなります。

ハーバードアーキテクチャ

- キャッシュにおけるハーバードアーキテクチャとは？
  - » 名称は、ハーバード大学とIBMが共同開発したコンピュータの名前に由来します。命令用とデータ用にそれぞれ独立した記憶装置とアドレス空間を持っていました。
  - 命令アクセスとデータアクセスとでは特性が異なるので、その特性に応じて最適化できるように、キャッシュを命令用（命令キャッシュ; instruction cache）とデータ用（データキャッシュ; data cache）に分割したアーキテクチャを指します。

- 命令パイプライン制御方式との関係
  - 命令フェッチとデータアクセスとで競合を避けるためには、命令キャッシュとデータキャッシュとを分離することが必須です。
  - このため、RISCコンピュータでは、必ずハーバードアーキテクチャを採用します。

- さて、以前の講義で、命令パイプライン制御のプロセッサのハードウェア構成（下図）を示しましたが、実は、この図には“ウソ”があります。（メイン）メモリと示されているのは、本当は、命令キャッシュです。
  - 図の通りに（メイン）メモリなら、1サイクルで命令をフェッチできません。
    - » なお、現在は、キャッシュアクセスにも複数サイクル必要で、命令フェッチも複数のステージで構成します。



キャッシュの多重レベル化

- 現在は、キャッシュのメモリ階層も多層化しています。
  - 1次キャッシュ、2次キャッシュ、3次キャッシュ、などと呼びます。
    - ただし、命令用とデータ用に分割するのは1次キャッシュです。命令キャッシュ、データキャッシュという名称を使うので、あまり1次キャッシュとは言わなくなりました。
  - 原則として、1次キャッシュのデータは必ず2次キャッシュにも存在し、2次キャッシュのデータは必ず3次キャッシュにも存在し、3次キャッシュのデータは必ずメインメモリに存在しますが、最近は、この包含原理を満たさない3次キャッシュも登場しています。
    - 犠牲キャッシュ（victim cache）と言って、2次キャッシュから追い出されたラインデータのみを3次キャッシュに格納します。