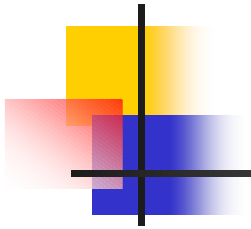
マルチコア時代の並列プログラミング ～ロックとメモリオーダリング～

中村 実

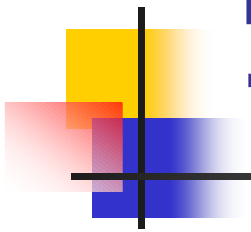
nminoru@nminoru.jp

http://www.nminoru.jp/~nminoru/



まずは自己紹介を

- 電機メーカー勤務のエンジニア
  - Java VM、特に並列GC・JITコンパイラの研究・開発
    - Java系雑誌にときどき寄稿
  - 最近はIA-64と戯れる日々
- 趣味で Web に細々とプログラミングのメモを綴る日々
  - 御縁がありまして Binary Hacks の著者の末席を汚すことに

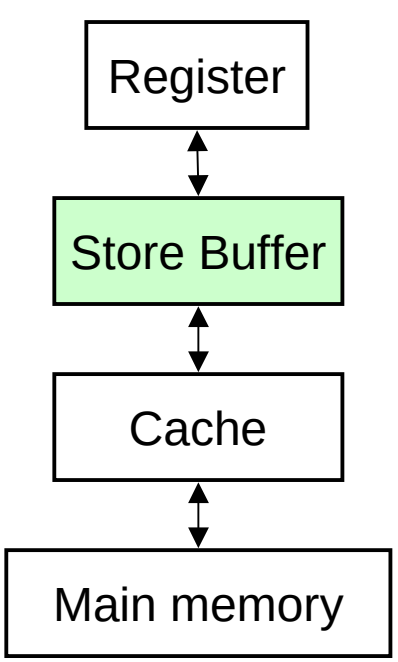


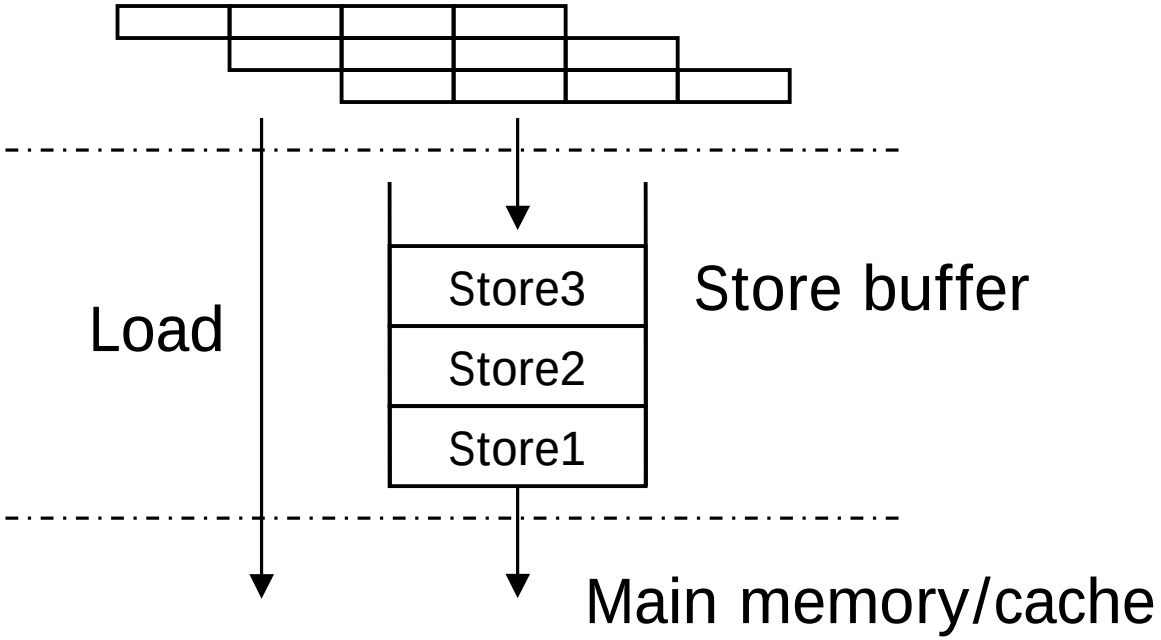
Binary Hacks #94
プロセッサのメモリオーダリングに注意

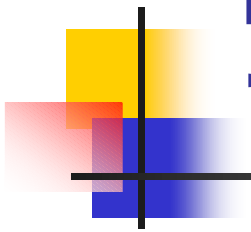
- CPU は Out-of-order 実行
  - 高速化のために load/store の順序を入れ替える
    - Load 命令は早く (投機実行)
    - Store 命令は遅く (Store buffering)
- メモリオーダリング(memory ordering)
  - CPU に許されているメモリ順序の規約
- 意図した通りの順序で実行させるにはメモリバリア(フェンス命令)が必要

Binary Hacks #94
プロセッサのメモリオーダリングに注意

- Store Buffer
  - Store 命令をより早く完了させるための機構

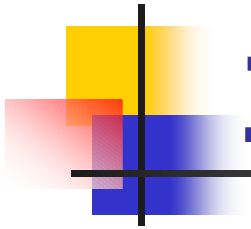






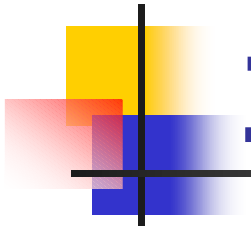
Binary Hacks #94
プロセッサのメモリオーダリングに注意

- 読者の方の感想
  - どういうプログラムでメモリオーダリングを気にする必要があるのかよくわからない
  - Pthread の mutex や IPC の semaphore ではダメなのか？
  - Cmpxchg 命令でいいのでは？
- そこで今日の発表
  - メモリオーダリングが問題になるような並列プログラムのテクニックとして Lock-free synchronization を紹介
  - 「マルチコア時代の並列プログラミング」というタイトルはオーバーだったかも orz



並列プログラムのモデル

- 今回のお話のターゲットとなるモデルは
  - スレッドがたくさん (Webアプリとか)
  - スレッドはコアにバインド
    - スレッド間の依存がない/少ない
      ⇒ メモリスループットがボトルネック
    - スレッド間通信が多い



並列プログラムのモデル

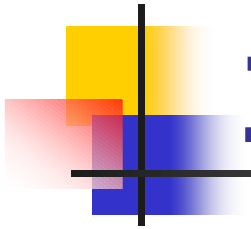
- 今回のお話のターゲットとなるモデルは
  - スレッドがたくさん (Webアプリとか) ×
  - スレッドはコアにバインド
    - スレッド間の依存がない/少ない
      ⇒ メモリスループットがボトルネック
    - スレッド間通信が多い

並列プログラムのモデル

- 今回のお話のターゲットとなるモデルは
  - スレッドがたくさん (Webアプリとか) ×
  - スレッドはコアにバインド
    - スレッド間の依存がない/少ない ×
      ⇒ メモリスループットがボトルネック
    - スレッド間通信が多い

並列プログラムのモデル

- 今回のお話のターゲットとなるモデルは
  - スレッドがたくさん (Webアプリとか) ×
  - スレッドはコアにバインド
    - スレッド間の依存がない/少ない ×
      ⇒ メモリスループットがボトルネック
    - スレッド間通信が多い ○



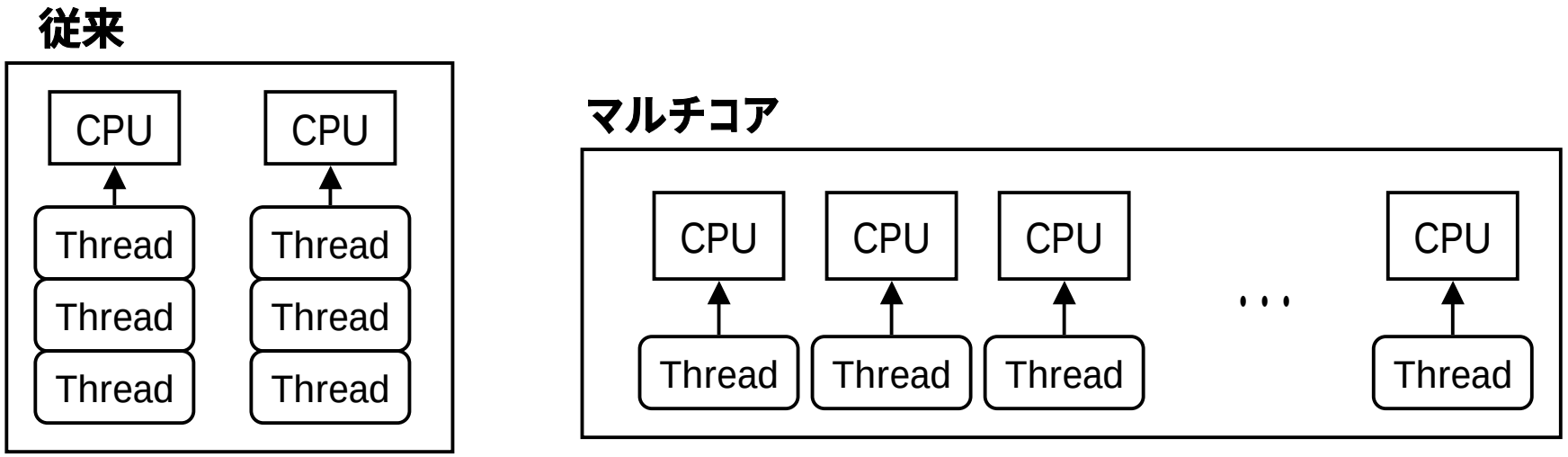
並列プログラムのモデル

- 今回のお話のターゲットとなるモデルは
  - スレッドがたくさん (Webアプリとか) ×
  - スレッドはコアにバインド
    - スレッド間の依存がない/少ない ×
      ⇒ メモリスループットがボトルネック
    - スレッド間通信が多い ○

例えば･･･ 並列GCとか

マルチコア時代の並列プログラム

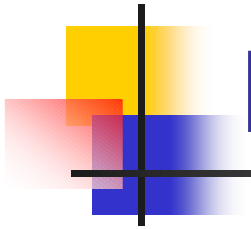
- マルチコアでは mutex がボトルネックになる(かも)
  - コアが増えると衝突(conflict)が増加
  - 衝突時にスレッドがサスペンドしてもうれしくない



Mutex や spin lock などに替わる
うまいスレッド同期処理はある？

Mutex や spin lock などに替わる
うまいスレッド同期処理はある？

ある!!
Lock-free synchronization

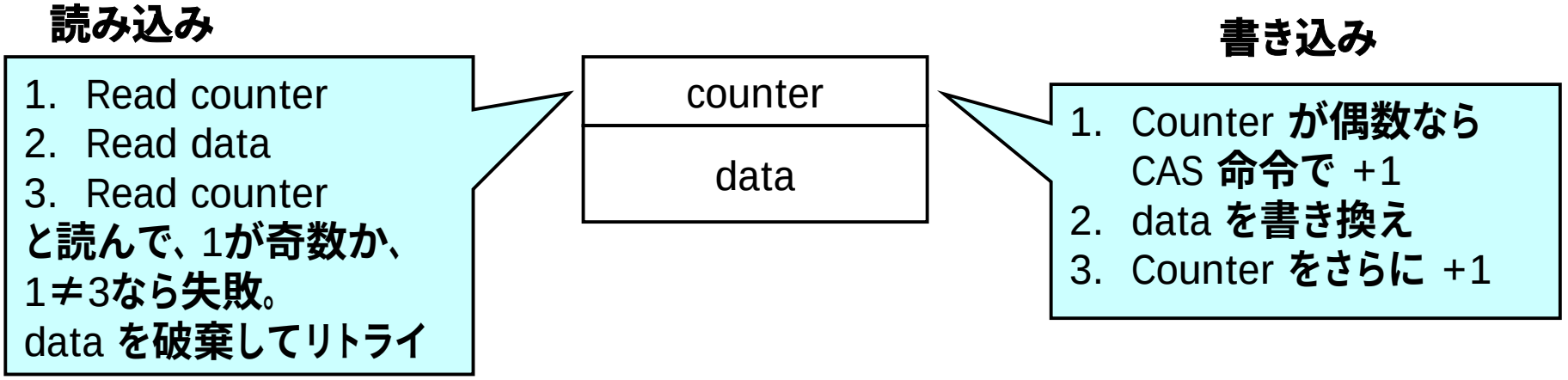


Lock-free synchronization

- 特徴
  - ロック状態がない。よって高速。
  - スレッドスケジューリングからの影響が小さい
- 実現方法
  - アトミック命令の組み合わせで実現
    - CAS (compare and swap) → x86 では cmpxchg命令
    - LL/SC (load linked/store conditional)
- どういうデータ構造があるの？
  - Deque, FIFO, LIFO
  - 単方向リスト,双方向リスト, Set, Hash
- 次から lock-free なアプローチのいくつかを紹介

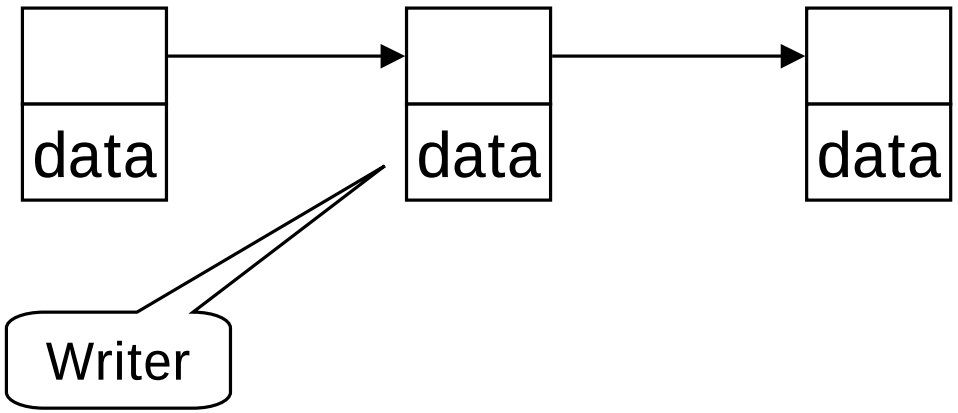
Sequence lock

- Optimistic lock (**楽観的なロック**)
- **任意のデータ** + counter
- **読み込みスレッドだけなら** lock-free
- **書き込みスレッドは** lock **が必要**
  - Counter **が偶数なら解放、奇数なら占有状態**



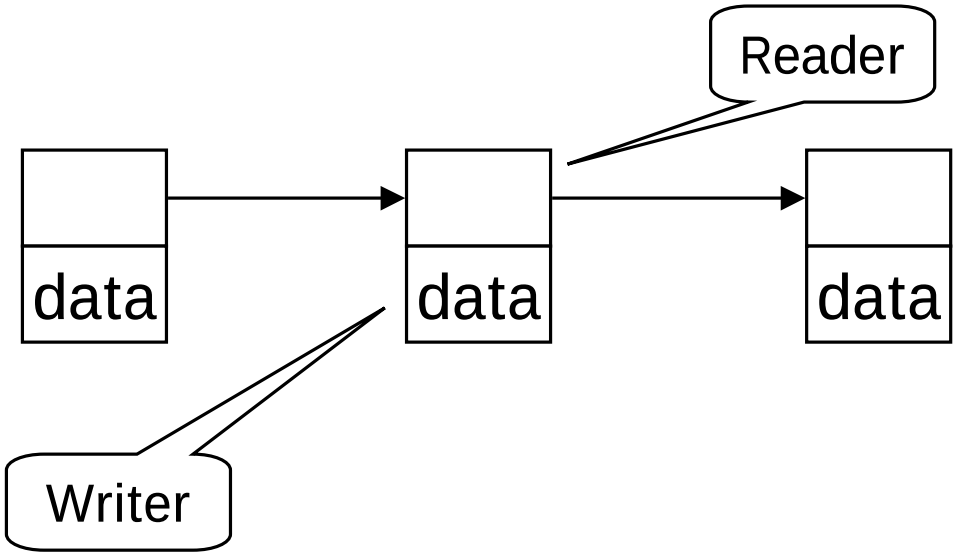
Read Copy Update (RCU)

- 単方向リスト
- 書き込みの遅延を許す
- アトミック命令が不要



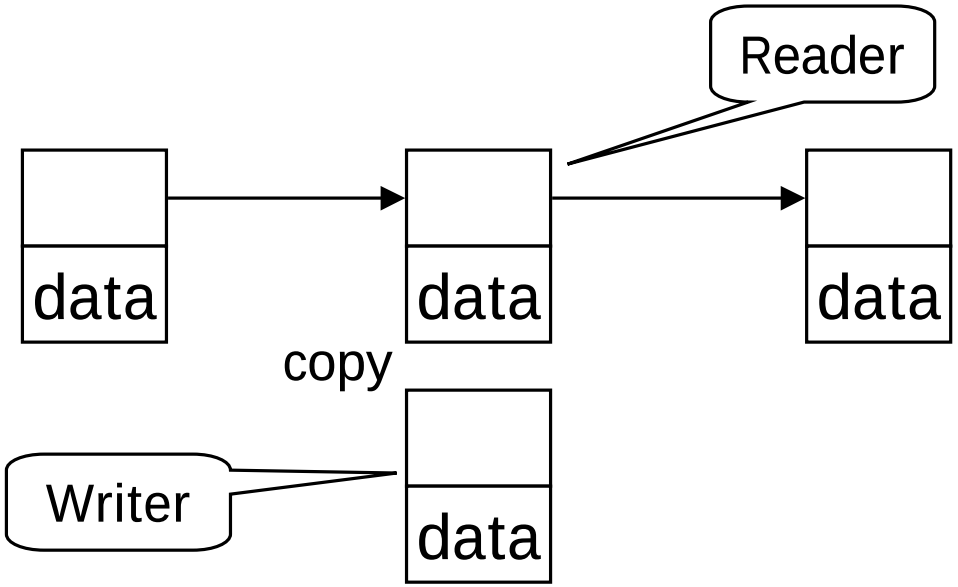
Read Copy Update (RCU)

- 単方向リスト
- 書き込みの遅延を許す
- アトミック命令が不要



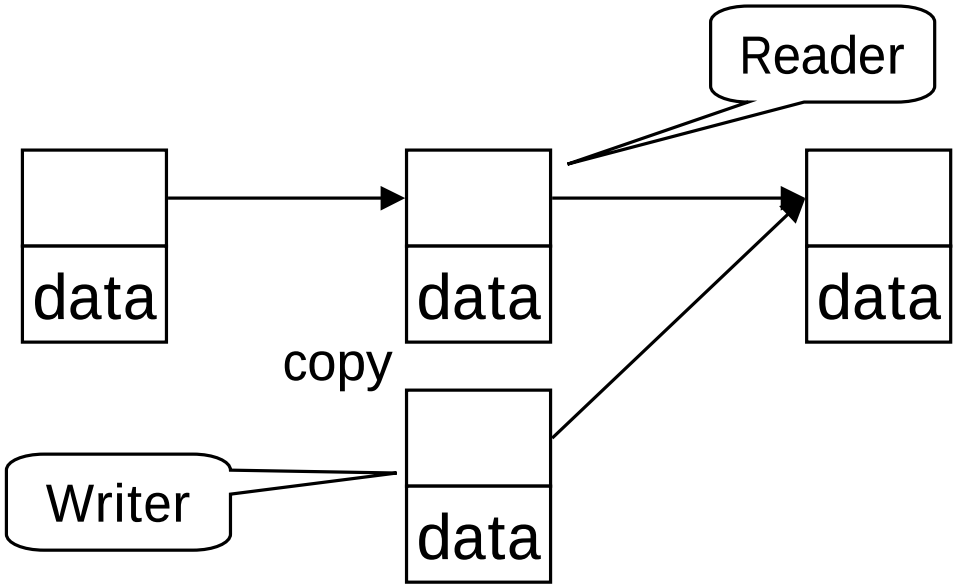
Read Copy Update (RCU)

- 単方向リスト
- 書き込みの遅延を許す
- アトミック命令が不要



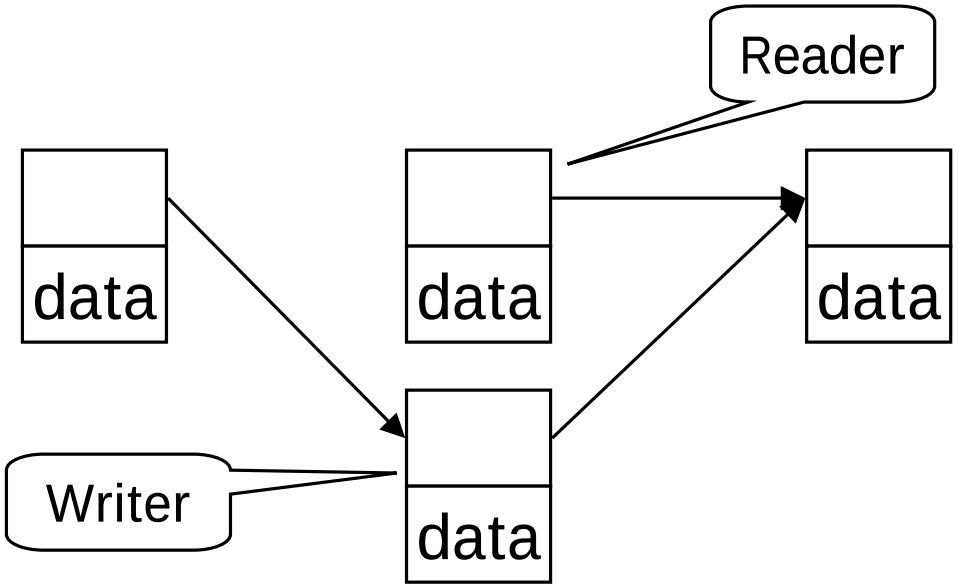
Read Copy Update (RCU)

- 単方向リスト
- 書き込みの遅延を許す
- アトミック命令が不要



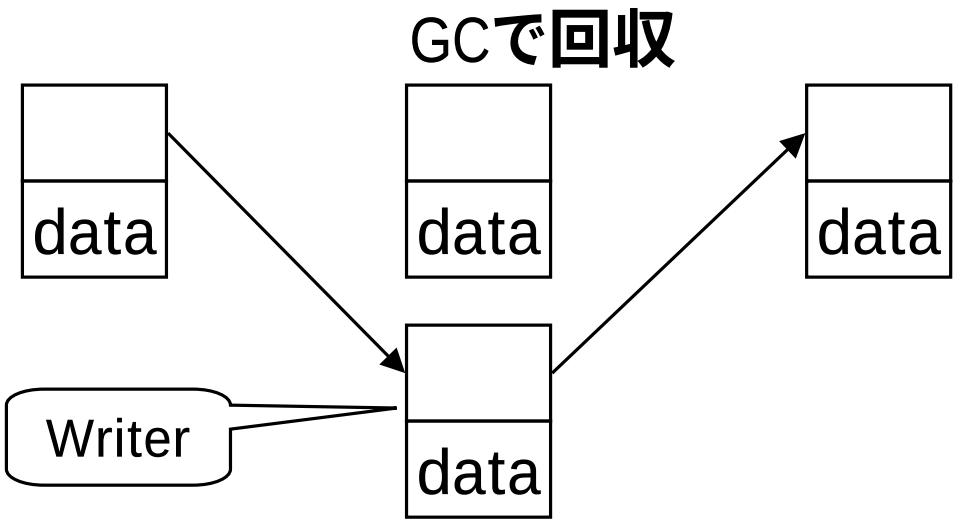
Read Copy Update (RCU)

- 単方向リスト
- 書き込みの遅延を許す
- アトミック命令が不要



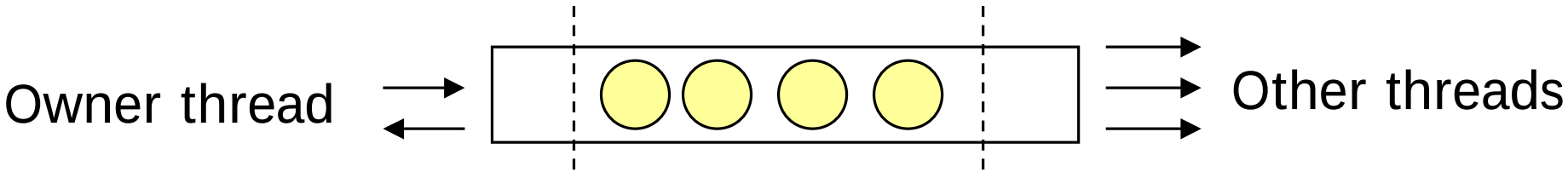
Read Copy Update (RCU)

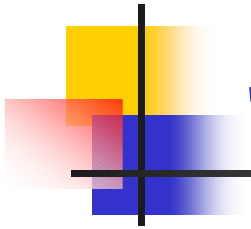
- 単方向リスト
- 書き込みの遅延を許す
- アトミック命令が不要



Double-ended Queue (Deque)

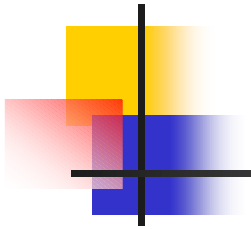
- N.Arora et al.,″Thread scheduling for multiprogrammed multiprocessors″,SPAA 1998
  - OS **内部のタスクキューのために考えられた** deque
  - **片側が所有スレッド用、もう片側は他スレッド用**
  - **所有スレッドだけがデータを** push **できる**
  - Push**も**pop**も**lock-free **かつ、通常時はアトミック命令も不要**
  - Sun HotSpot VM **の並列**GC**などで利用されている。**





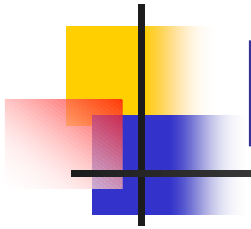
その他

- Deque
  - M.Micheal, “CAS-based lock-free algorithm for shared dequeues”, EuroPar 2003
- **双方向リスト**
  - H.Sundell, “ Lock-free and practical doubly linked list-based deques using single-word compare-and-swap”, OPODIS 2004
    - NOBLE - a library of non-blocking synchronization protocols
      http://www.cs.chalmers.se/~noble/



どうやってプログラムするの？

- 基本は論文を読んで実装!!
  - Lock-free synchronizationは、アプリケーションに合わせてデータ構造を調整して使う必要あり
- ライブラリもあるよ
  - Ross Bencina 氏のページ
    - http://www.audiomulch.com/~rossb/code/lockfree
    - Lock-free & wait-free アルゴリズムの実装のリストがある
  - Lock-free library
    - http://www.cl.cam.ac.uk/research/srg/netos/lock-free/
    - Alpha, MIPS, IA-64, x86, PPC, SPARC で動作
    - GPL 下でソース公開
- 情報はどこに？
  - 意外にも wikipedia が充実
    - http://en.wikipedia.org/wiki/Lock-free_and_wait-free_algorithms

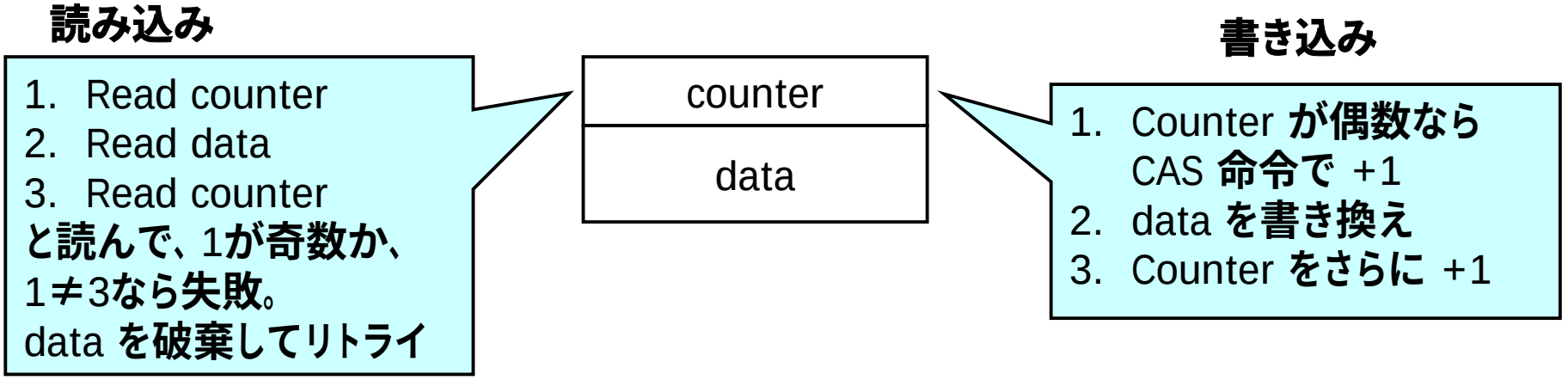


問題点もある

- 衝突(conflict)が少ないプログラムでは、mutexと性能に差がない
- アルゴリズムが複雑
  - 複雑なものはバグの源
  - 実装が正しいことの論理検証が困難
- 実装が難しい
  - CPU の out-of-order 実行によるメモリ順序の逆転が問題に
    - ようやく話がメモリ・オーダリングに繋がった (^_^)/

Sequence lock

- Optimistic lock (**楽観的なロック**)
- **任意のデータ** + counter
- **読み込みスレッドだけなら** lock-free
- **書き込みスレッドは** lock **が必要**
  - Counter **が偶数なら解放、奇数なら占有状態**



Sequence lock

- Optimistic lock (**楽観的なロック**)
- **任意のデー**
- **読み込み**
- **書き込み**
  - Counter **有状態**

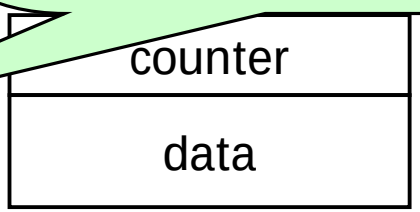
3つの read の順番が守られていないとアルゴリズムが破綻

メモリバリアが必要

読み込み

1. Read counter
2. Read data
3. Read counter

と読んで、1が奇数か、1≠3なら失敗。data を破棄してリトライ



書き込み

1. Counter **が偶数なら CAS 命令で** +1
2. data **を書き換え**
3. Counter **をさらに** +1

x86のメモリオーダリングの復習

x86 CPU のメモリオーダリング

- X86 は同じ命令セットでも、メモリオーダリングは CPU によって違う･･･

	RAR	WAR	WAW	RAW
i386				
i486,Pentium				×
P6 〜	×	×		×
Opteron	×			×

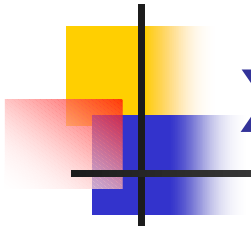
?A? = ? After ?　　× 順序の逆転が起きる

x86 CPU のメモリオーダリング

- X86 は同じ命令セットでも、メモリオーダリングは CPU によって違う･･･

	RAR	WAR	WAW	RAW
i386				
i486,Pentium				×
P6 〜	×	×	×	×
Opteron	×			×

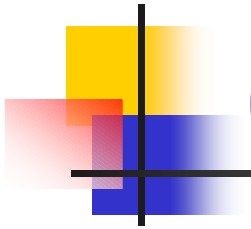
?A? = ? After ?　　× 順序の逆転が起きる



x86 CPUのメモリの順序化

- 副作用のある命令
  - CPUID, Lock# プレフィックス
    - パイプライン・Store buffer をいったんクリアするため、メモリの順序化の副作用がある。
    - 重い。
- フェンス専用命令
  - SFENCE 命令 (Pentium3以降)
    - Store → Store を順序化
  - LFENCE 命令 (Pentium4以降)
    - Load → Load を順序化
  - MFENCE 命令 (Pentium4以降)
    - 全てのメモリ操作を順序化

C/C++ でメモリバリアを使うには？

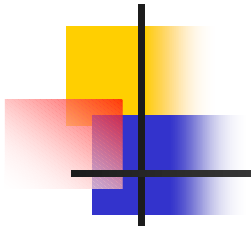


- インラインアセンブラ

```
#define mb()  asm volatile ("mfence“:::”memory”);
#define rmb() asm volatile (“lfence”:::”memory”);
#define wmb() asm volatile (“sfence”:::”memory”);
```

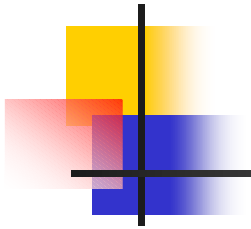
- volatile を付けると順序化される ABI もある
  - ex. IA-64 ABI
- C++0x には入るかも!

> Evolution working group issue list
>
> ES066. Support for parallel programming
> For example, locks, threading, memory barrier, static local initialization.



まとめ

- **マルチコア時代到来**
  - Mutex/spin lock **などの替わりに** (**使えるときは**) Lock-free synchronization **を使おう**
  - Memory ordering
    - **コア数が多いほどメモリ順序の逆転は起き易い**
    - **今からメモリフェンスを入れた正しいプログラムを書こう**



まとめ

- **マルチコア時代到来**
  - Mutex/spin lock **などの替わりに (使えるときは)** Lock-free synchronization **を使おう**
  - Memory ordering
    - **コア数が多いほどメモリ順序の逆転は起き易い**
    - **今からメモリフェンスを入れた正しいプログラムを書こう**

ご静聴ありがとうございました