並列アプリケーション向けインテル® TBB スケーラブル・メモリー・アロケーターの活用

インテル® スレッディング・ビルディング・ブロック (インテル® TBB) 2019

インテル コーポレーション
ソフトウェア開発エンジニア
Nikita Ponomarev

アプリケーションの想定…

- 高速な malloc/free
  - クロススレッドはそこまで速くないかもしれないが忘れないようにする
- ローカルキャッシュでホットなオブジェクトを取得する
- キャッシュの連想度を乱用せず、フォルス・シェアリングを回避する
- スレッド競合を回避する
- メモリー消費を適度に保つ
  - システム・アロケーター +20% であれば OK
- API は標準、競争率は高い

アロケーター用 C インターフェイス

インテル® TBB アロケーター API	アナログ API
scalable_malloc	C 標準ライブラリー (scalable_ prefix なし)
scalable_calloc	
scalable_free	
scalable_realloc	
scalable_posix_memalign	POSIX*
scalable_aligned_malloc	Microsoft* C ランタイム・ライブラリー
scalable_aligned_realloc	
scalable_aligned_free	
scalable_msize	ptr が指し示すメモリーブロックの使用可能なサイズ

アロケーター用 C++ インターフェイス

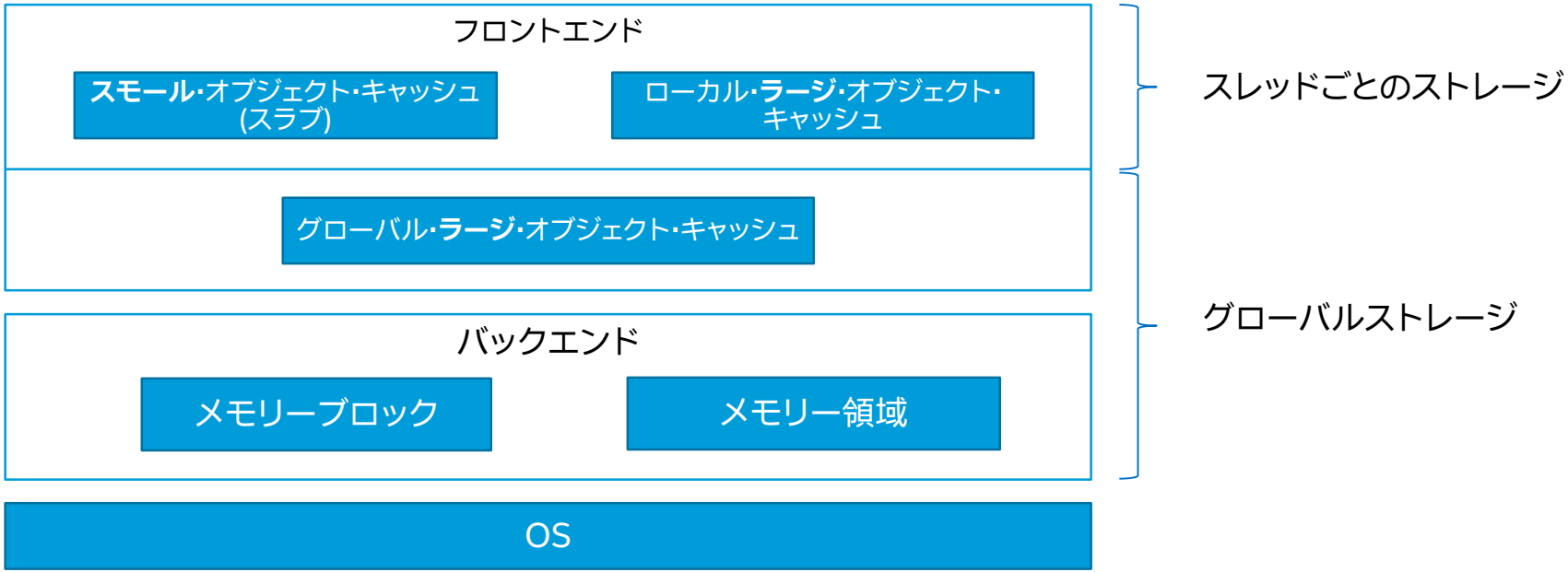
アロケーター・クラス

- tbb::scalable_allocator<T>
- tbb::cache_aligned_allocator<T>
- tbb::tbb_allocator<T>
- tbb::zero_allocator<T>

メモリー・リソース・クラス (C++17 以降)

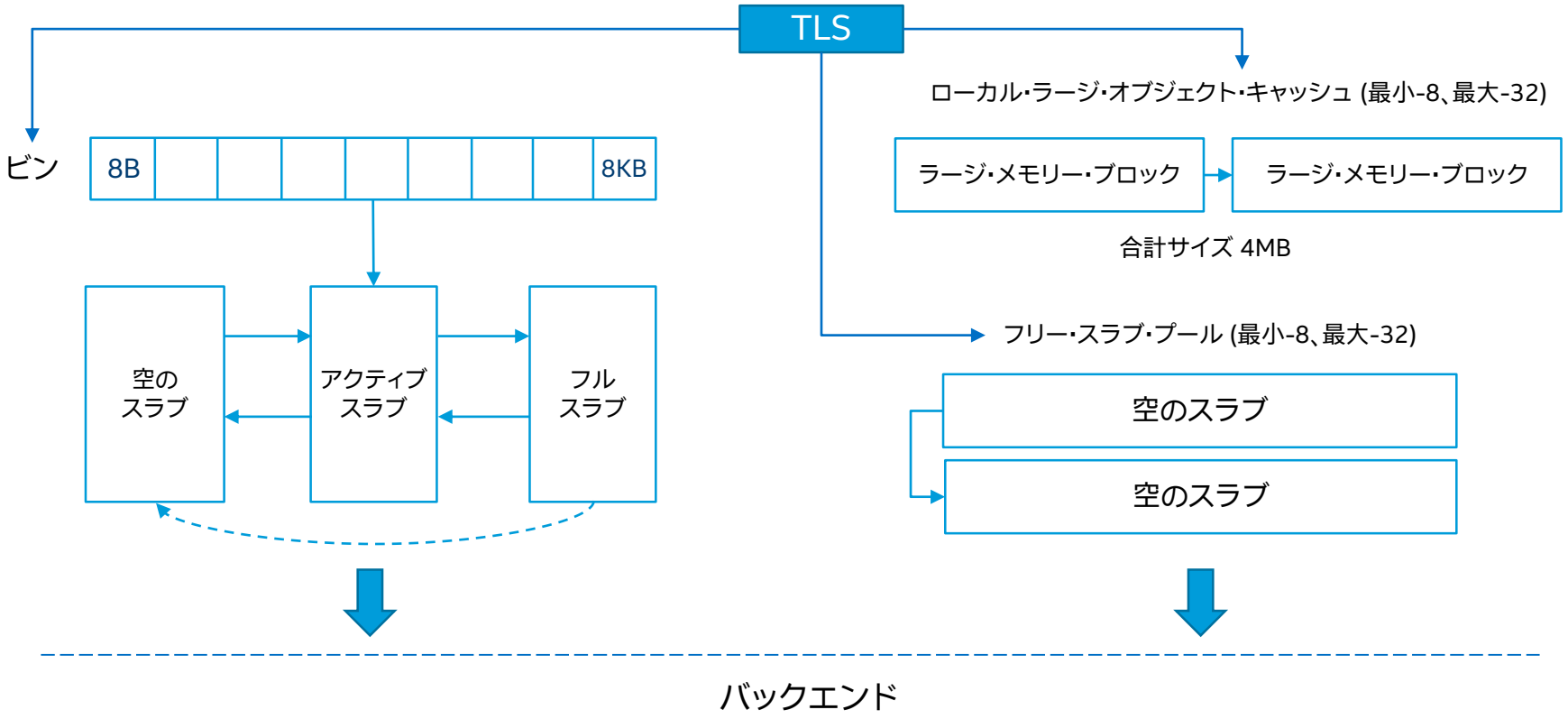
- tbb::scalable_memory_resource() グローバルアクセサー
- tbb::cache_aligned_resource

一般的なアーキテクチャー概要

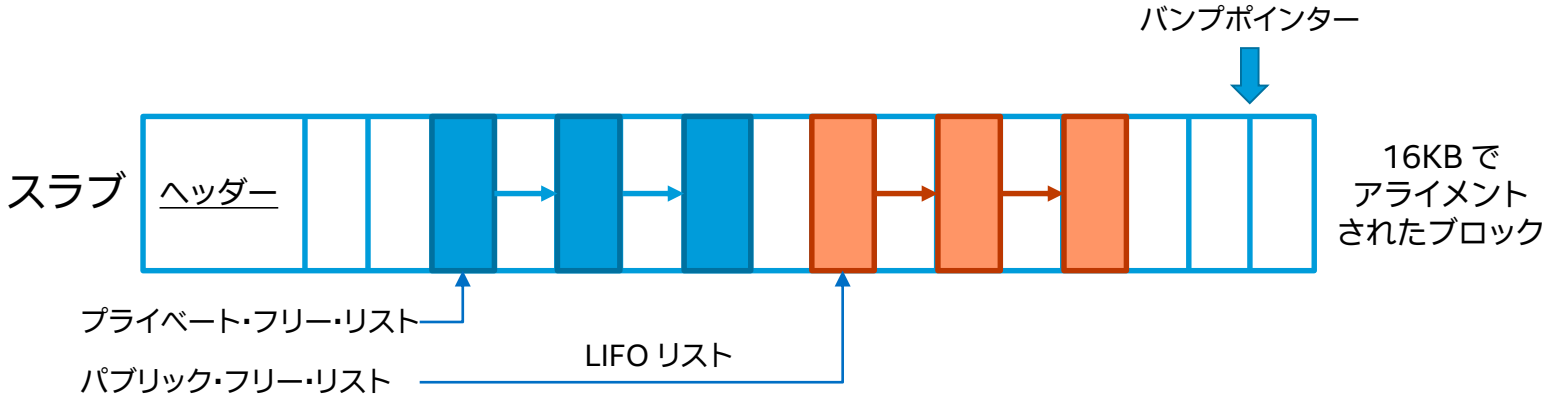


- スモール・オブジェクト (<=8KB) とラージ・オブジェクト (>8KB) で構造と割り当て手法は異なる

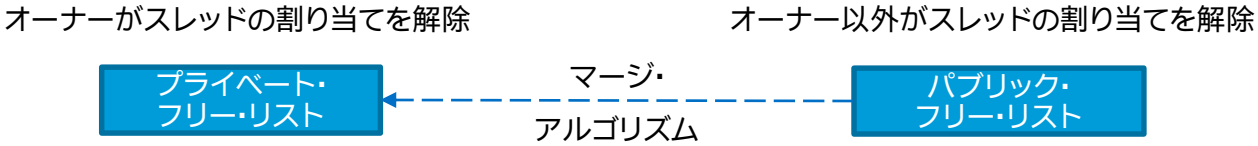
フロントエンド – スモール・オブジェクト



フロントエンド – スモール・オブジェクトの詳細



- スラブを同じサイズのオブジェクト (サポートしているサイズ以下の要求されたサイズでアライメント) とヘッダー (2 キャッシュライン) に保つ



割り当て/解放の手法

割り当て:

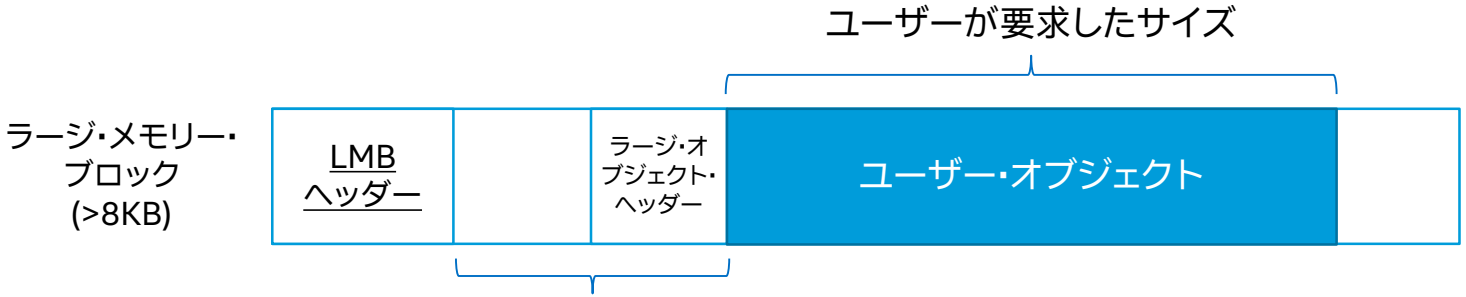
1. TLS に移動する
2. サイズでビンを見つける
3. アクティブなスラブを見つける
4. スラブのフリーリストでオブジェクトを見つける
5. オブジェクトを返す

解放:

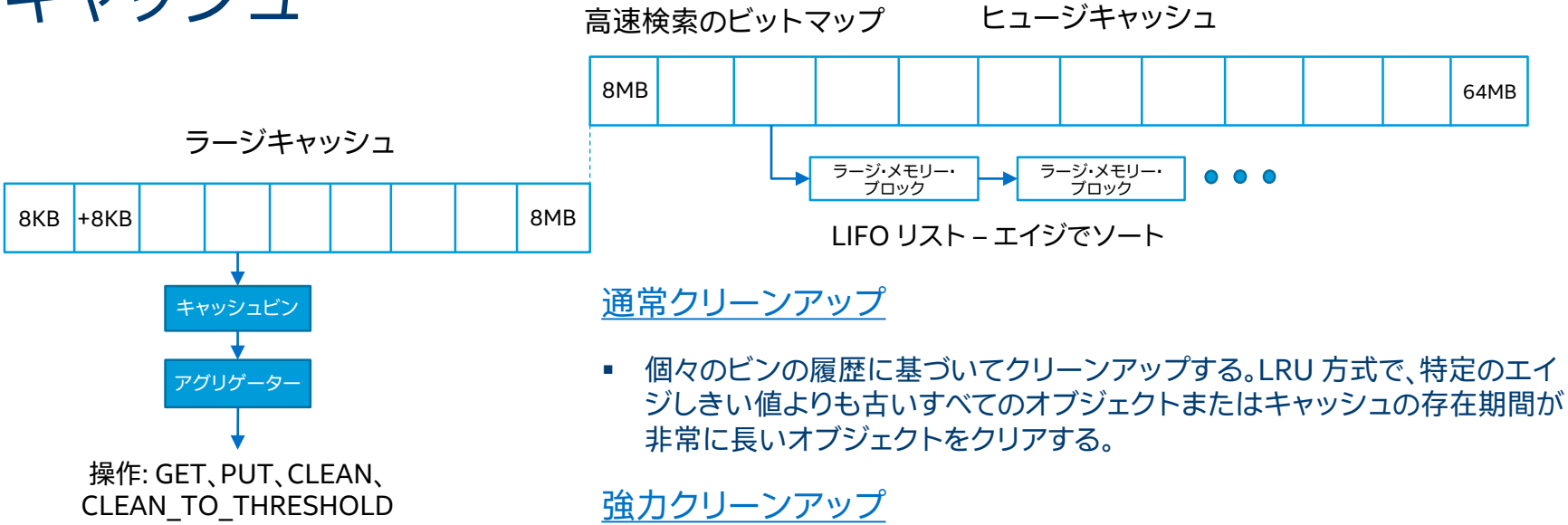
1. アドレスをアライメントする
2. スラブヘッダーを見つける
3. 自身の TID を見つける
4. スラブの TID と比較する
5. オブジェクトをスラブのフリーリストに入れる
6. スラブをアクティブに移動する

ホットなパスにアトミックがない!

フロントエンド – ラージ・オブジェクト



フロントエンド – グローバル・ラージ・オブジェクト・キャッシュ



通常クリーンアップ

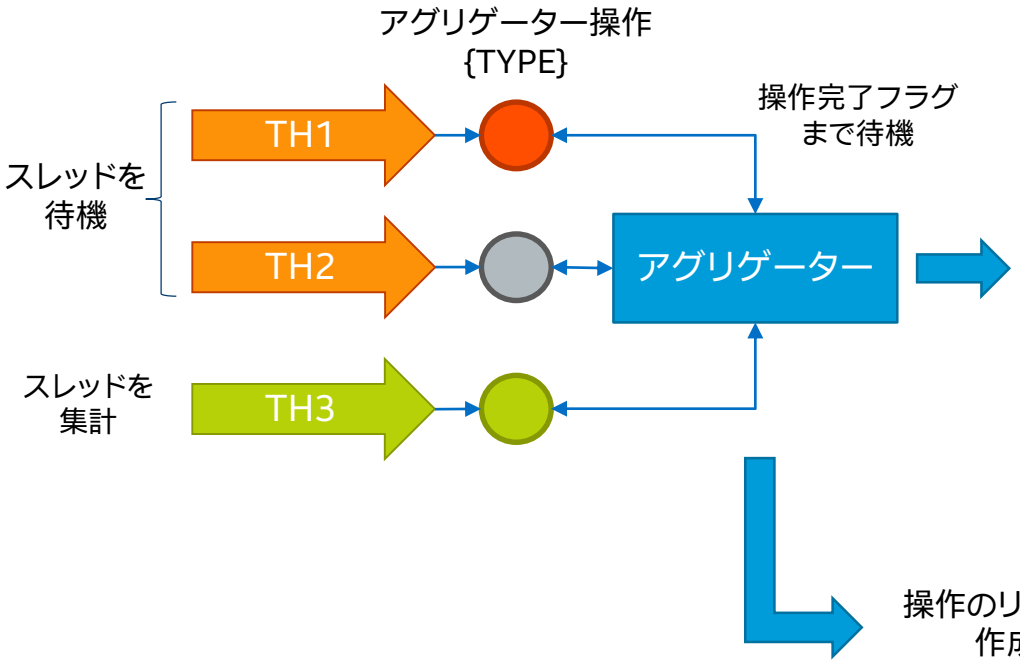
- 個々のビンの履歴に基づいてクリーンアップする。LRU 方式で、特定のエイジしきい値よりも古いすべてのオブジェクトまたはキャッシュの存在期間が非常に長いオブジェクトをクリアする。

強力クリーンアップ

- すべてのビンのすべてのオブジェクトをクリーンアップする

バックエンド

アグリゲーター – 競合の軽減

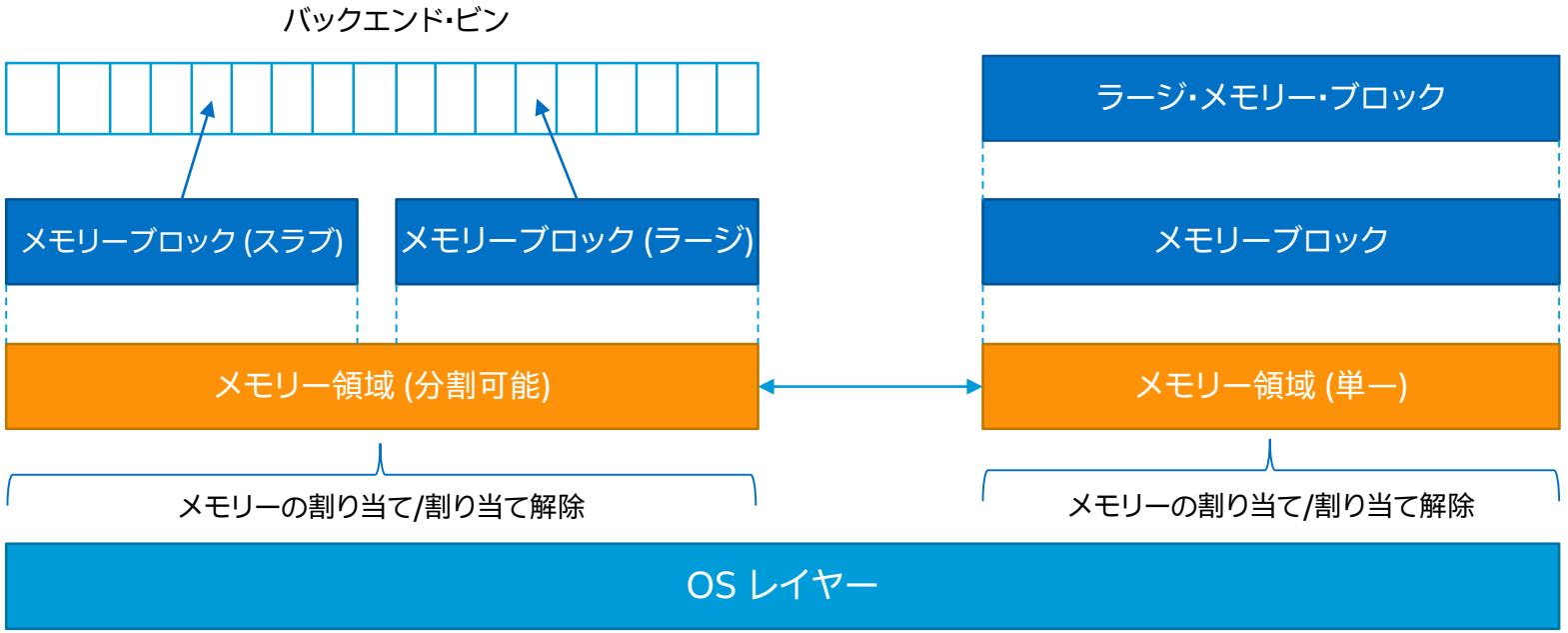


1 つのスレッドのみ実行操作を開始でき、
ほかのスレッドは待機する

handle_operations(operation_list_t* list)
{
- 操作リストを反復
- 各操作タイプ (GET、PUT、CLEAN) の特別な操作を行う
- 結果を各操作に戻す

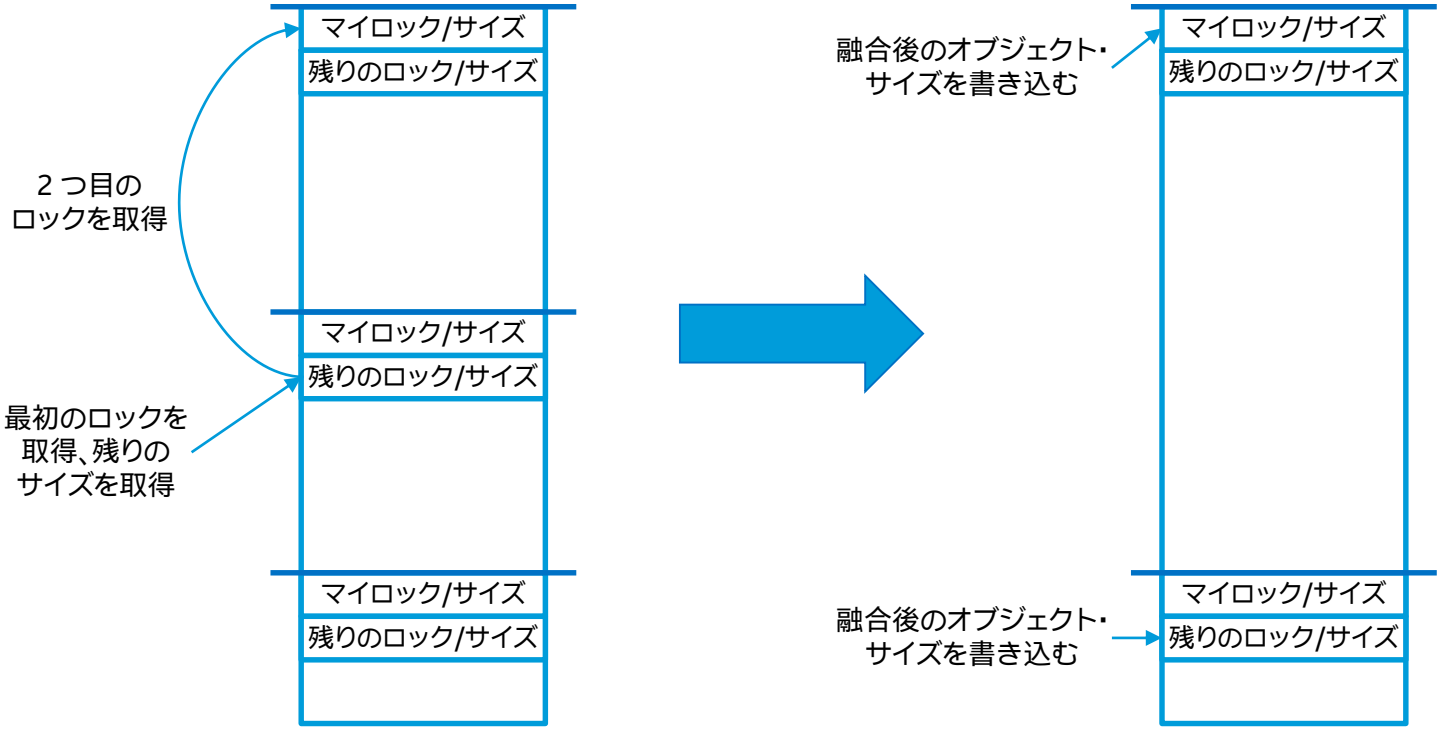
}

バックエンド – 全体的な構造



- 3 スレッドまで同時に OS からメモリーを追加できる
- バックエンドが分割/融合を行い、キャッシングは利用せず、バッファリングは利用する
- 共有状態なし: 領域およびブロックは互いについて何も知らない

バックエンド – ブロックの融合



アロケーター・チューニング API – 構成

int scalable_allocation_mode(int mode, intptr_t value):

- TBBMALLOC_USE_HUGE_PAGE
  - ヒュージページを使用 (トランスペアレント・ヒュージ・ページをサポート)。値は 1 または 0 (デフォルト)。
- TBBMALLOC_SET_SOFT_HEAP_LIMIT
  - 全体的なキャッシング制限を定義。値はサイズ (バイト)。
- TBBMALLOC_SET_HUGE_SIZE_THRESHOLD (インテル® TBB 2019 Update 6 以降)
  - クリーンアップが明示的に要求されない限り OS に解放されない割り当ての下限しきい値を定義。値はサイズ (バイト)。

類似環境変数: TBB_MALLOC_USE_HUGE_PAGE および TBB_MALLOC_SET_HUGE_SIZE_THRESHOLD

アロケーター・チューニング API – コマンド

int scalable_allocation_command(int cmd, void *reserved):

- TBBMALLOC_CLEAN_THREAD_BUFFERS
  - スレッドのメモリーバッファー (スモール・オブジェクト、フリー・スラブ・プール、LLOC) をクリーンアップする
- TBBMALLOC_CLEAN_ALL_BUFFERS
  - アロケーターのグローバル・メモリー・バッファー (および呼び出しスレッドのバッファー) をクリーンアップする

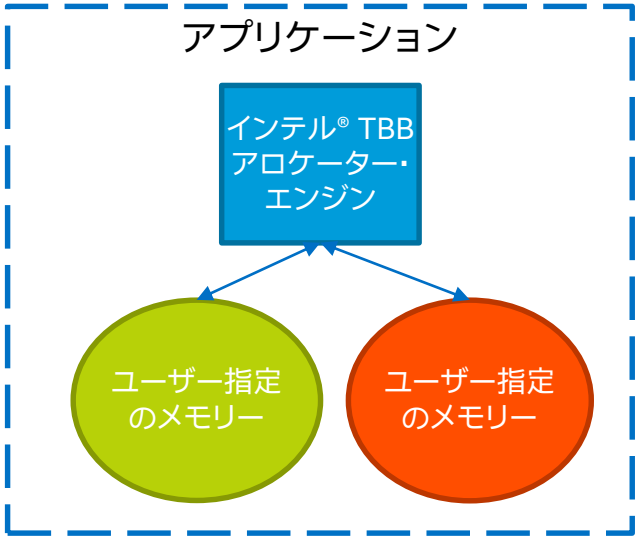
正しいクリーンアップ・プロシージャー:

- 利用可能なメモリーをすべて解放 -> すべての「割り当て」スレッドで TBBMALLOC_CLEAN_THREAD_BUFFERS を呼び出す -> 「メイン」スレッドで TBBMALLOC_CLEAN_ALL_BUFFERS を呼び出す

メモリープール – プレビュー機能

すべてのアロケーター構造、ユーザー指定のメモリーで次が可能

- すべてのメモリーの高速割り当て解除
- メモリー・フラグメントと個々のグループ間の同期を抑える
- ユーザー指定のメモリーのソースを含む



メモリープール – 例

```
// 固定サイズのバッファーからのメモリープール
char buffer[1024 * 1024];
tbb::fixed_pool my_fixed_pool((void*)buffer, 1024 * 1024);

// ユーザー指定のアロケーターからのメモリープール
tbb::memory_pool< std::allocator<char> > my_pool; // メモリープールを作成
void *ptr = my_pool.malloc(8); // 8 バイト割り当て
ptr = my_pool.realloc(ptr, 24); // 割り当てを 24 バイトに拡張
my_pool.free(ptr); // 割り当てを解除
my_pool.recycle(); // 再利用のためプールのメモリーをすべて解放

// コンテナーの使用法
typedef tbb::memory_pool_allocator<int> pool_allocator_t;
std::list<int, pool_allocator_t> my_list(pool_allocator_t(my_pool));
```

メモリー API 置換ライブラリー

動的メモリー割り当ての標準関数に対するすべての呼び出しをインテル® TBB 関数に自動的に置換

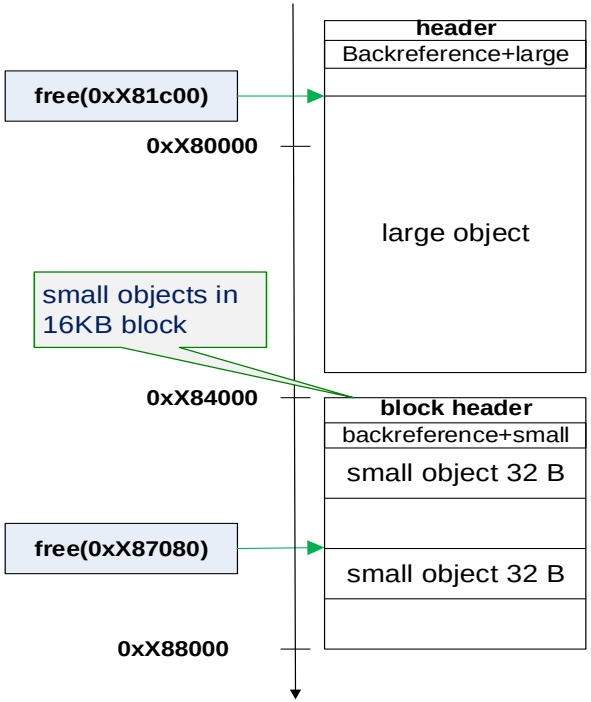
異なる DLL に分離 – tbbmalloc_proxy

- Windows*
  - トランポリンで MSVC ランタイムをフック
- Linux*/macOS*
  - 単純なシンボル置換
  - リンカーは最初に見つけたシンボル (malloc、calloc、free、その他) を使用する
    - LD_PRELOAD 環境変数 (Linux*)
    - DYLD_INSERT_LIBRARIES (macOS*)

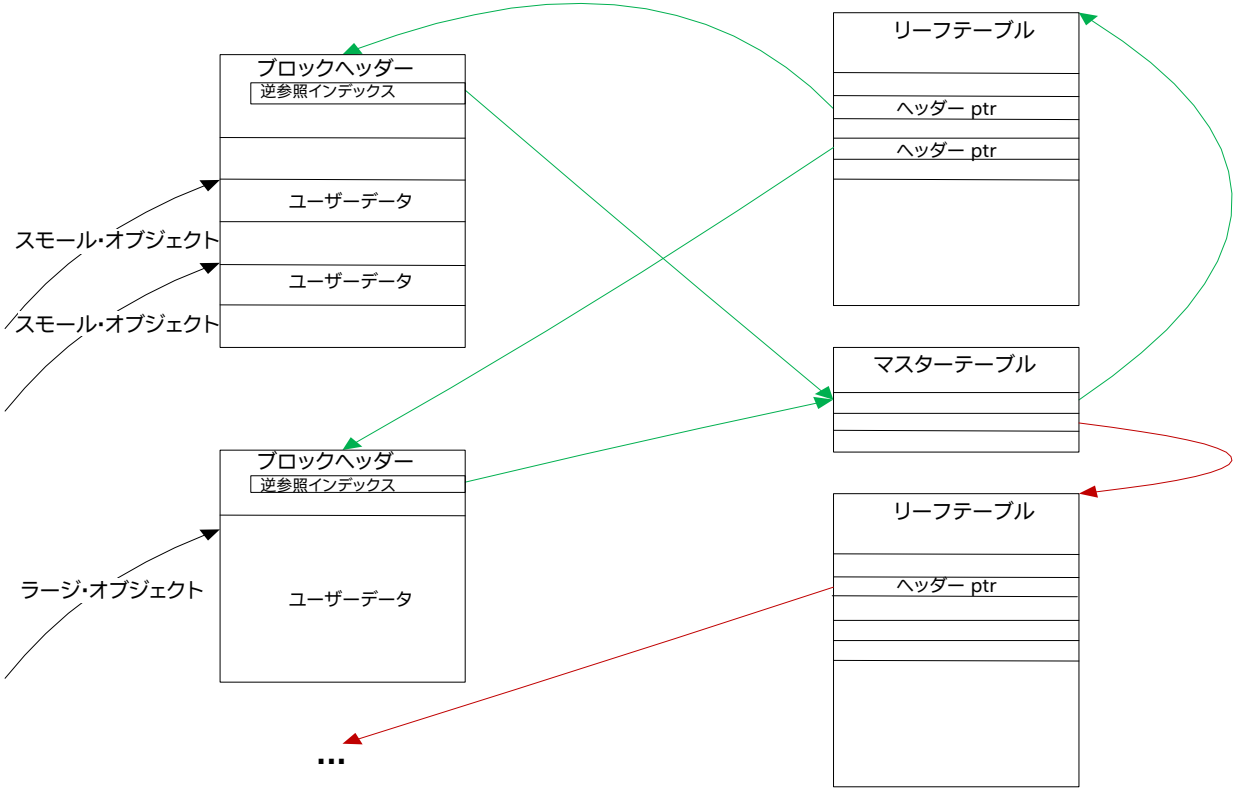
補足資料

オブジェクトを特定する方法

- free() には長さがない。確実に見つけるには?
- システム・アロケーターに負荷をかけると所有権を検出できる
- 解決策は逆参照:
  - フリー・ホット・パスの最大 2 つのコールドリード



逆参照



グローバル･ラージ･オブジェクト･キャッシング手法

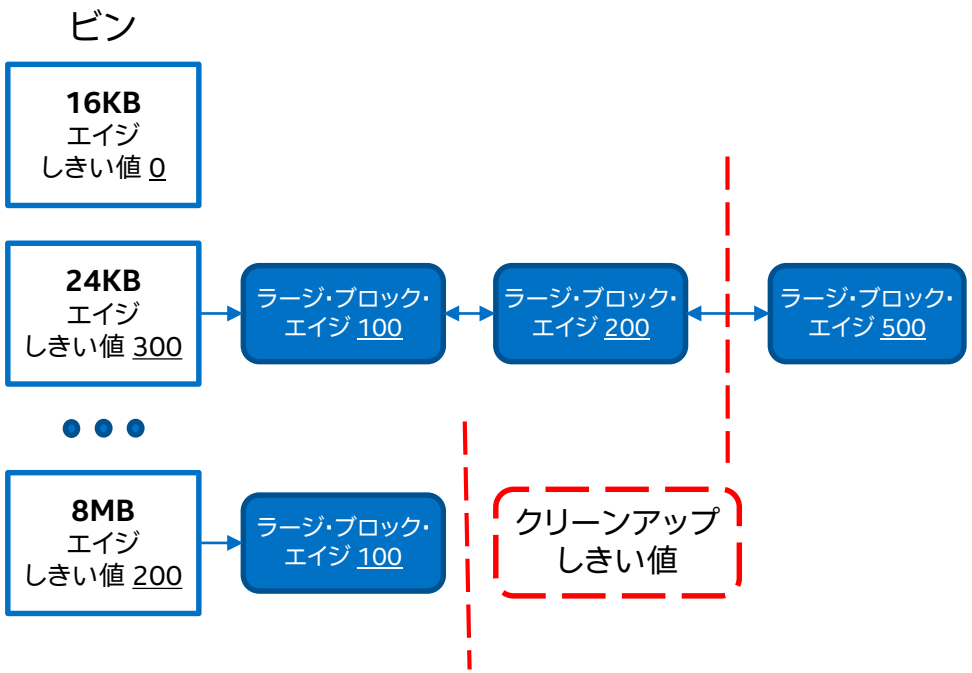
エイジベースのクリーンアップ、エイジはプログラム開始後の put/get の数
- エイジはグローバルステート

しきい値:

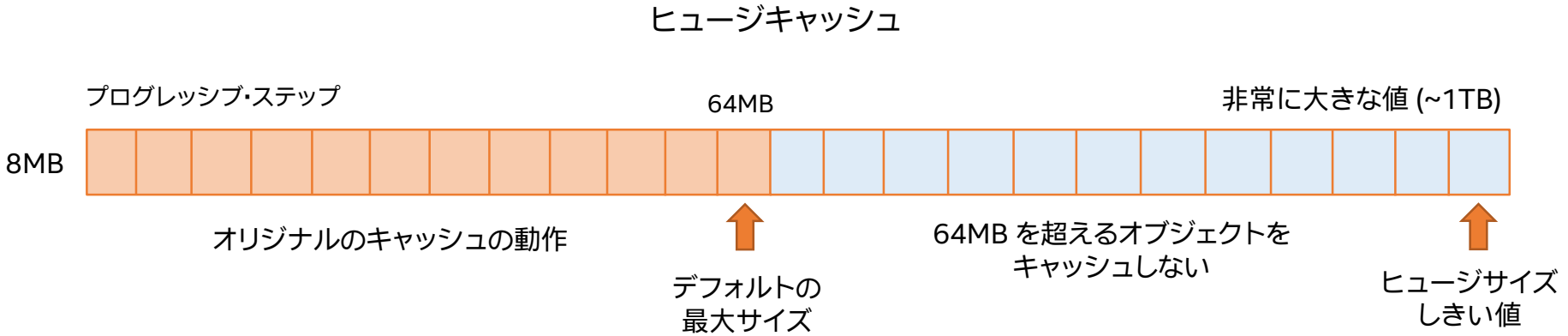
1. ミスする場合は増やす
2. 「非常に長い」で合計キャッシュが「非常に大きい」場合は減らす
3. 「非常に長い」を使用していない場合はすべて忘れて、長時間実行しているプログラムで減らさない

各ビンには個別のしきい値がある
- 異なる使用モードの近似は不適切

魔法の定数はない

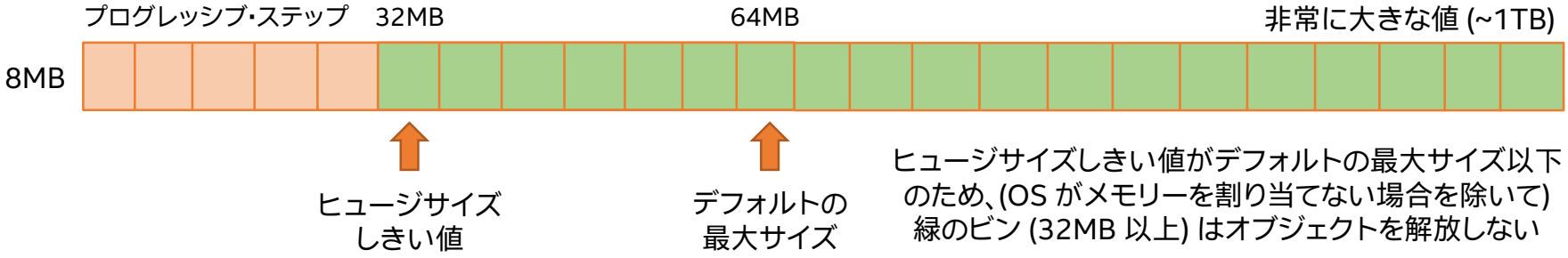


ヒュージサイズしきい値 – デフォルト

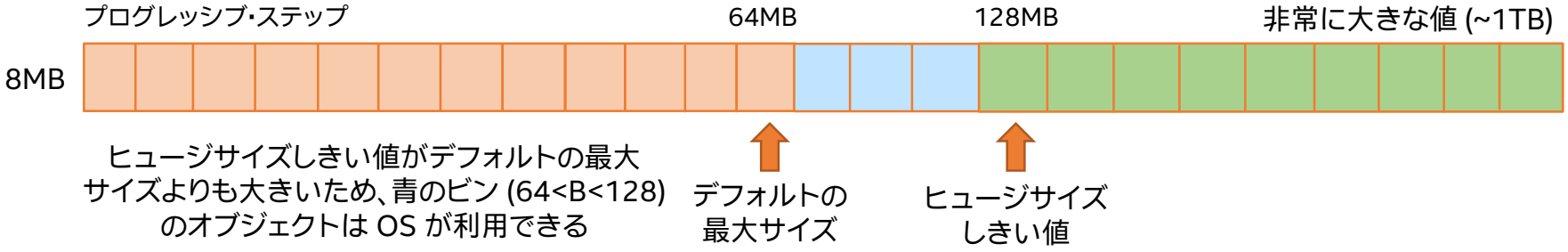


ヒュージサイズしきい値 – 定義

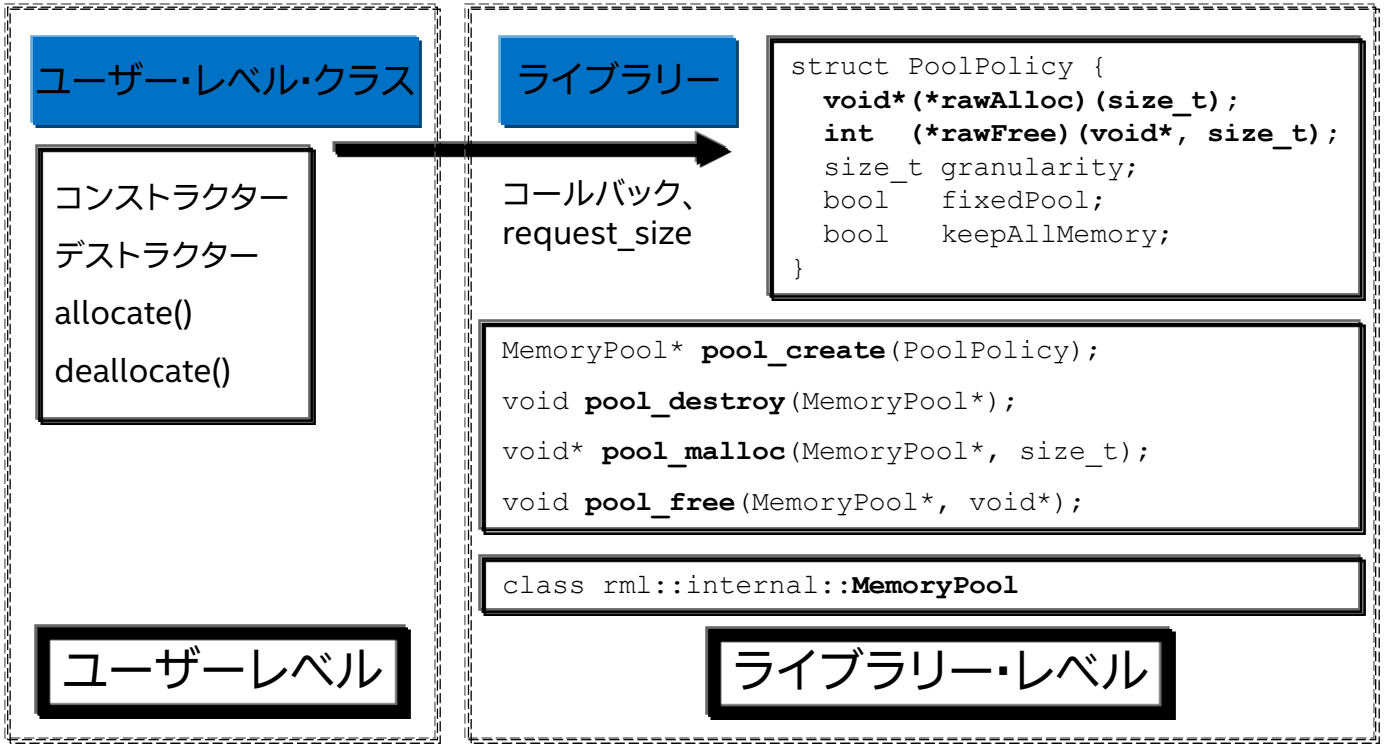
TBB_MALLOC_SET_HUGE_SIZE_THRESHOLD=32



TBB_MALLOC_SET_HUGE_SIZE_THRESHOLD=128



メモリー・プール・ライブラリー・アーキテクチャー



ありがとうございました

法務上の注意書きと最適化に関する注意事項

性能に関するテストに使用されるソフトウェアとワークロードは、性能がインテル® マイクロプロセッサー用に最適化されていることがあります。SYSmark* や MobileMark* などの性能テストは、特定のコンピューター・システム、コンポーネント、ソフトウェア、操作、機能に基づいて行ったものです。結果はこれらの要因によって異なります。製品の購入を検討される場合は、他の製品と組み合わせた場合の本製品の性能など、ほかの情報や性能テストも参考にして、パフォーマンスを総合的に評価することをお勧めします。 詳細については、www.intel.com/benchmarks (英語) を参照してください。

本資料の情報は、現状のまま提供され、本資料は、明示されているか否かにかかわらず、また禁反言によるとよらずにかかわらず、いかなる知的財産権のライセンスも許諾するものではありません。製品に付属の売買契約書『Intel's Terms and Conditions of Sale』に規定されている場合を除き、インテルはいかなる責任を負うものではなく、またインテル製品の販売や使用に関する明示または黙示の保証 (特定目的への適合性、商品性に関する保証、第三者の特許権、著作権、その他、知的財産権の侵害への保証を含む) をするものではありません。

最適化に関する注意事項

インテル® コンパイラーでは、インテル® マイクロプロセッサーに限定されない最適化に関して、他社製マイクロプロセッサー用に同等の最適化を行えないことがあります。これには、インテル® ストリーミング SIMD 拡張命令 2、インテル® ストリーミング SIMD 拡張命令 3、インテル® ストリーミング SIMD 拡張命令 3 補足命令などの最適化が該当します。インテルは、他社製マイクロプロセッサーに関して、いかなる最適化の利用、機能、または効果も保証いたしません。本製品のマイクロプロセッサー依存の最適化は、インテル® マイクロプロセッサーでの使用を前提としています。インテル® マイクロアーキテクチャーに限定されない最適化のなかにも、インテル® マイクロプロセッサー用のものがあります。この注意事項で言及した命令セットの詳細については、該当する製品のユーザー・リファレンス・ガイドを参照してください。

注意事項の改訂 #20110804

intel
Software