ORACLE

オラクル・コンサルが語る！ 共有プール管理の極意

日本オラクル株式会社
テクノロジーコンサルティング統括本部
シニアプリンシパルコンサルタント
辰巳 昌紀
プリンシパルコンサルタント
池田 大地

#odddtky

日本オラクル、今年最大の技術トレーニング・イベント

Oracle DBA & Developer Day 2013

以下の事項は、弊社の一般的な製品の方向性に関する概要を説明するものです。また、情報提供を唯一の目的とするものであり、いかなる契約にも組み込むことはできません。以下の事項は、マテリアルやコード、機能を提供することをコミットメント（確約）するものではないため、購買決定を行う際の判断材料になさらないで下さい。オラクル製品に関して記載されている機能の開発、リリースおよび時期については、弊社の裁量により決定されます。

Program Agenda

- 共有プールの基本的理解
- 設計～テスト～運用フェーズでの検討事項
  - 管理方式選定
  - 一次サイジング（自動SGA前提）
  - 監視・チューニング（自動SGA前提）

本セミナーの内容に関する注意

- 本資料の内容
  - 製品仕様に関する正式な情報ではなく、プロジェクトで調査を行い把握した内容を記載しています。そのため、参考情報と考えてください。
  - プロジェクトで得られた経験を基に、具体的な一次サイジングの指針および値を示しています。実際には、必要領域は処理内容や処理タイミングに依存するため、テストによりサイズを検討、確定してください。

- 前提環境
  - Oracle Database 11g Release2
  - 自動SGA管理

本セミナーの趣旨（1／2）

背景

- 共有プールの性質
  - 自動SGAが主流となり、メモリの低コスト化で共有プール枯渇エラー（ORA-4031）は減ってきた感覚はあるが、発生するとクリティカルな問題になりやすい（DB全体に影響が波及するエラー）
- 設計・監視のポイント
  - どんなシステムでも、完全に障害を防げる、設計、監視というものは存在しないが、一歩踏み込んだアーキテクチャと、設計、監視のポイントを理解することによって、ORA-4031といった障害の発生確率をかなり減らすことが可能である
- 障害の予防
  - ポイントをおさえることで、少なくともテスト時や本番運用の序盤等に障害の予兆に気付き、エラーが発生し大問題になる前に対処できると考えている

本セミナーの趣旨（2／2）

本日お伝えしたいこと

ORA-4031撲滅

① 自動SGA管理の使用を推奨

② 大規模バッファキャッシュ環境（特にRAC環境）では、バッファキャッシュ依存領域を上乗せしてサイジングする

③ 1サブプールあたりのサイズを十分に大きくサイジングする

④ 共有プールの自動拡張余力を残して、バッファキャッシュの最低サイズを設定する

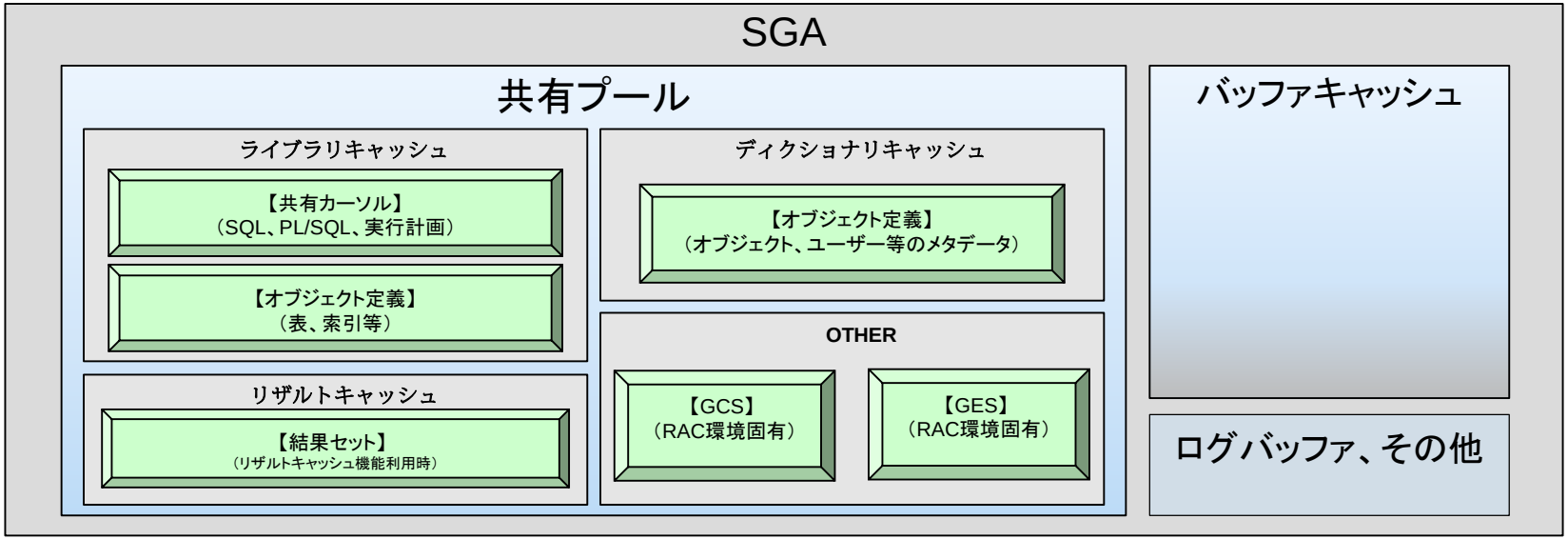
⑤ 共有プールの自動拡張余力が残っていることをリアルタイム監視する

共有プールの基本的理解

共有プールの基本的理解（1／18）

共有プールとは

- SQL、定義情報、実行計画等が格納される共有メモリ領域
- データベースで行われるほぼ全ての操作でアクセスされる領域

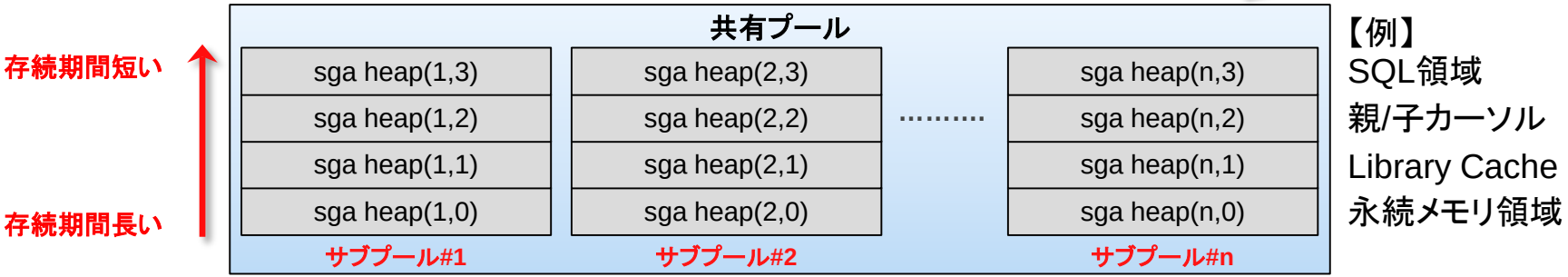


共有プールの基本的理解（2／18）

共有プールの内部構造

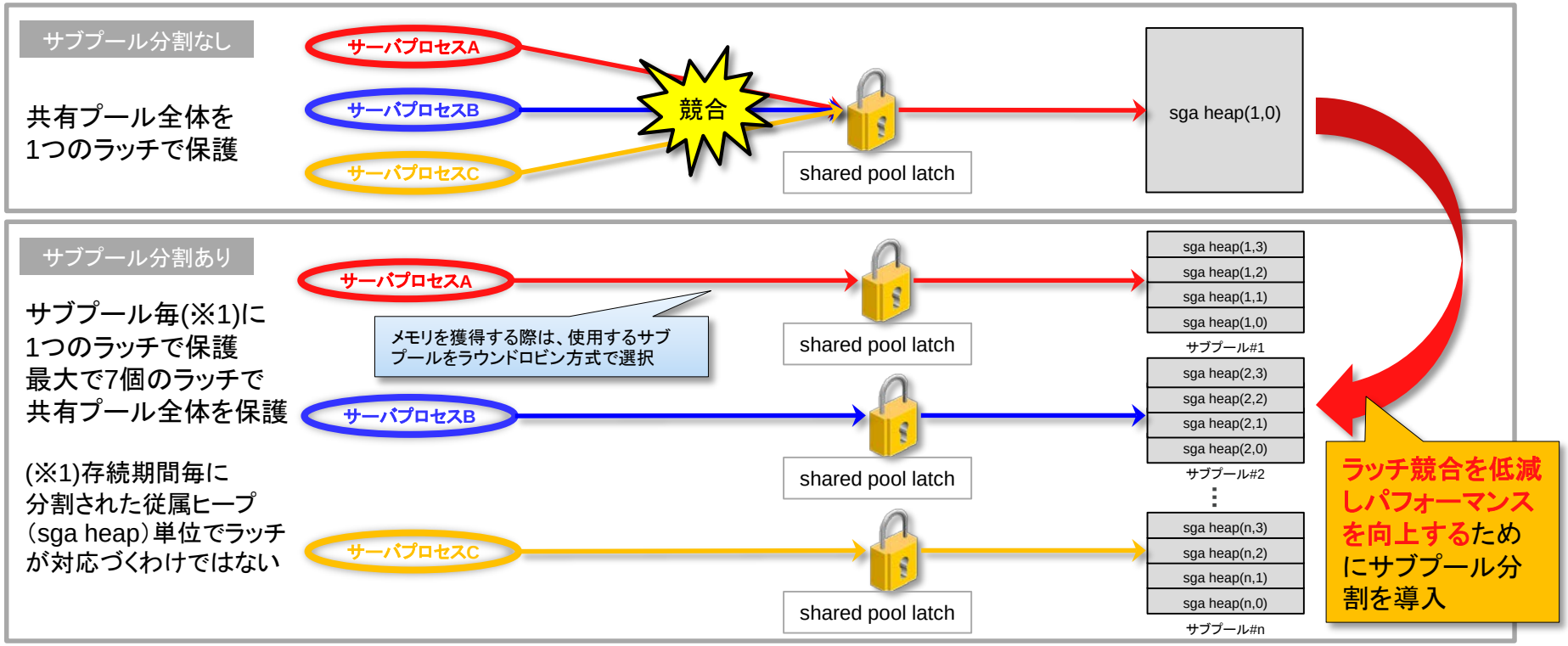
- サブプール分割
  - 目的：共有プールを保護するラッチ（shared pool latch）の競合分散のため
  - CPU数と共有プールのサイズに応じて最大7個に分割
- 存続期間による分割
  - 目的：断片化を予防するため
  - メモリの存続期間に応じてサブプール毎に4個に分割

Oracle DatabaseではSGAやPGA等の多くのメモリ領域を**「ヒープ」**と呼ばれる共通の構造で管理している。
共有プールは複数の**従属ヒープ「sga heap(x,y)」**で構成される。最大で 28個に分割される。



共有プールの基本的理解（3／18）

サブプール分割

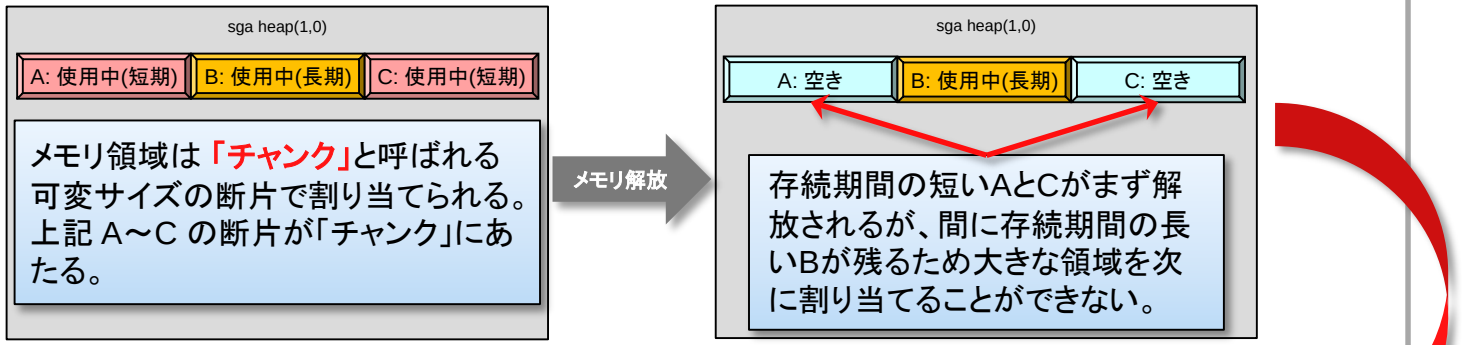


共有プールの基本的理解（4／18）

存続期間による分割

存続期間による分割なし

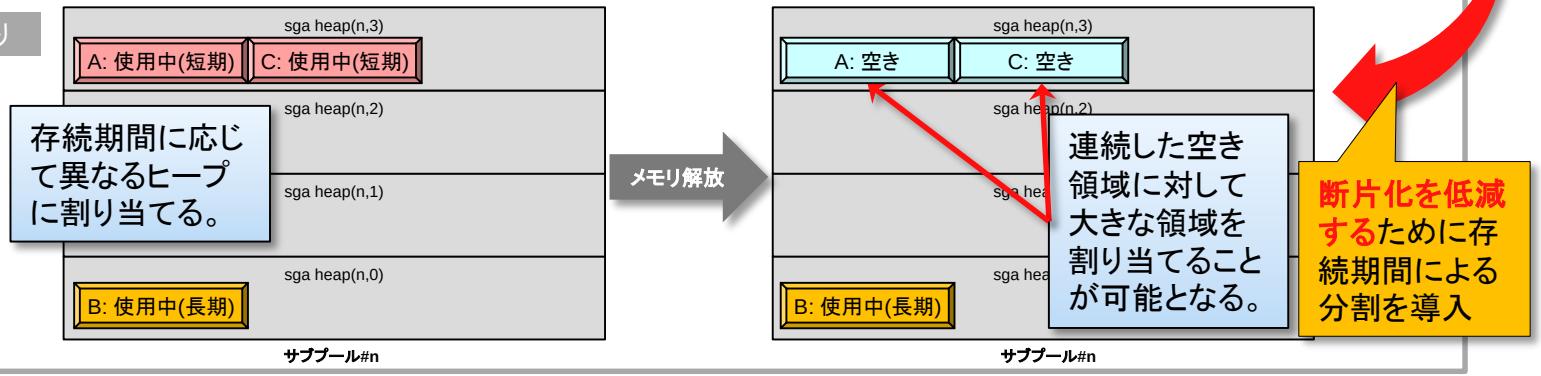
存続期間の異なるメモリ領域が一つの領域に混在すると、領域が解放されたときにメモリの断片化が発生しやすくなる。



存続期間による分割あり

同等の存続期間(※2)のメモリ領域を決まった領域に割り当てることで断片化が発生しにくくなる。

(※2)チャンクの獲得から解放までの期間



共有プールの基本的理解（5／18）

グラニュル（Granule）

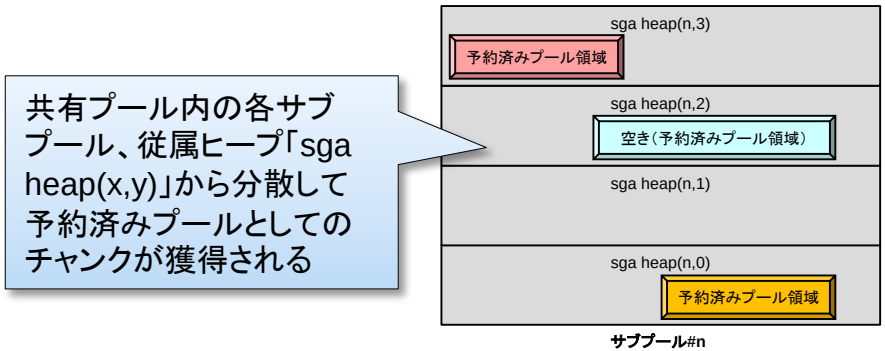
- SGAや各プールのメモリ割り当ての最小単位
  - 共有プールやバッファキャッシュ等の各領域のサイズはグラニュルサイズの倍数（グラニュルサイズ × グラニュル数) となる
  - SGAのサイズ（SGA_MAX_SIZEの値）に応じてグラニュルサイズは大きくなる
  - 自動SGA管理、自動メモリ管理における自動調整（拡張／縮小）はグラニュル単位で行われる

SGA_MAX_SIZE			グラニュルサイズ
	～	1GB以下	4MB
1GB超	～	8GB以下	16MB
8GB超	～	16GB以下	32MB
16GB超	～	32GB以下	64MB
32GB超	～	64GB以下	128MB
64GB超	～	128GB以下	256MB
128GB超	～		512MB

共有プールの基本的理解（6／18）

予約済みプール

- 予約済みプールは共有プールの一部
  - サイズ：shared_pool_reserved_size にて指定
    （デフォルト：共有プールのサイズの5%）
  - 要求サイズが閾値（デフォルト：4400バイト）を超えた場合にのみ使用されうるため、他の領域の使用状況が逼迫していても予約済み領域は使用されていないことがある（領域が有効活用されていない場合があるので注意）



共有プールの基本的理解（7／18）

共有プールへの新規メモリ割当ての流れ

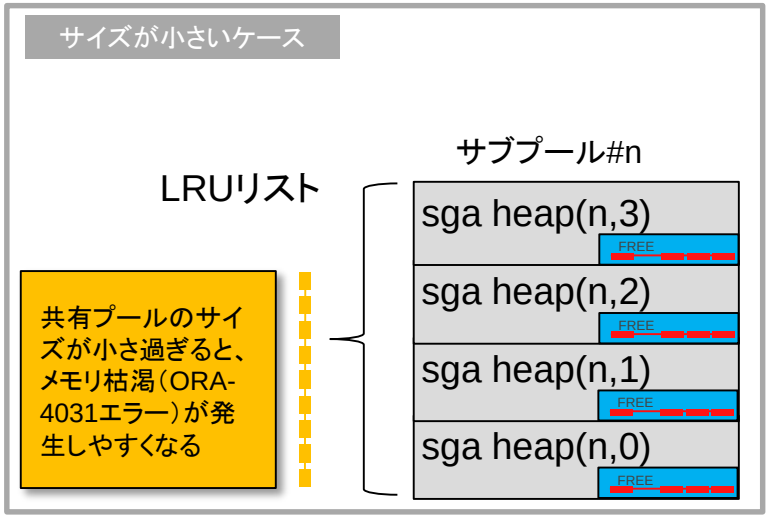
1. フリーリストから空きを探す
   フリーリストは各従属ヒープ sga heap(x,y) 毎に存在
2. Reserved Granule（未割当ての領域）から確保
3. 予約済みプールから確保
4. 使用済み領域を再利用（LRUリストをフラッシュ＆チャンクを連結して空きを作る）
5. 共有プールを拡張（IMMEDIATEモードでバッファキャッシュから空きを確保）
6. ORA-4031エラー（要求サイズのメモリが割り当てられない）

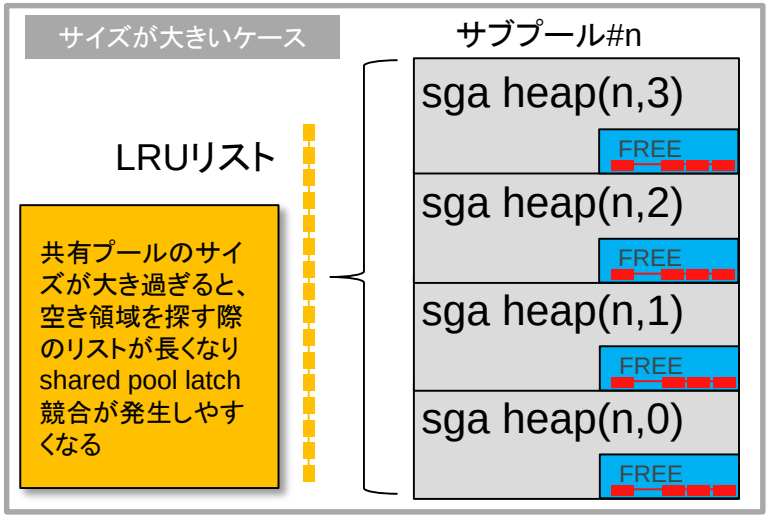
共有プールの基本的理解（8／18）

共有プールのサイズの考え方

- サイズ（shared_pool_size）の設定
  - サブプールあたり1G～4G程度を目安に初期設計を検討するとよい（経験則）
  - 大きければ大きいほどよいという領域ではない（shared pool latch 待ち注意）

あくまで初期設計の目安であり、実システムにおいてより大きいサイズの構成実績あり

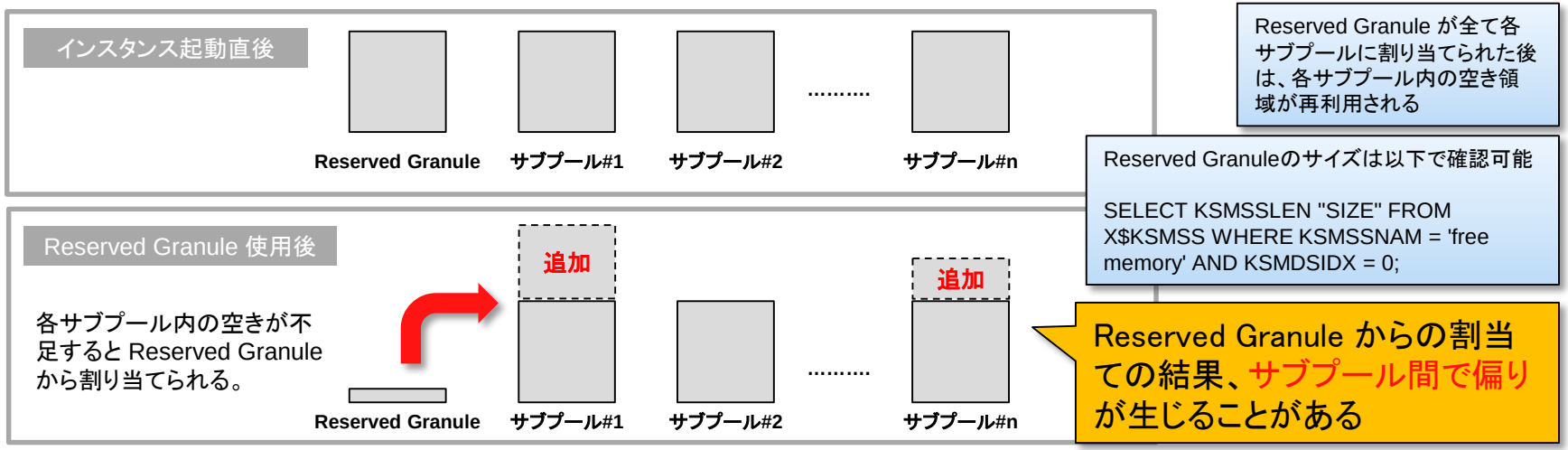




共有プールの基本的理解（9／18）

空き領域（1） Reserved Granule

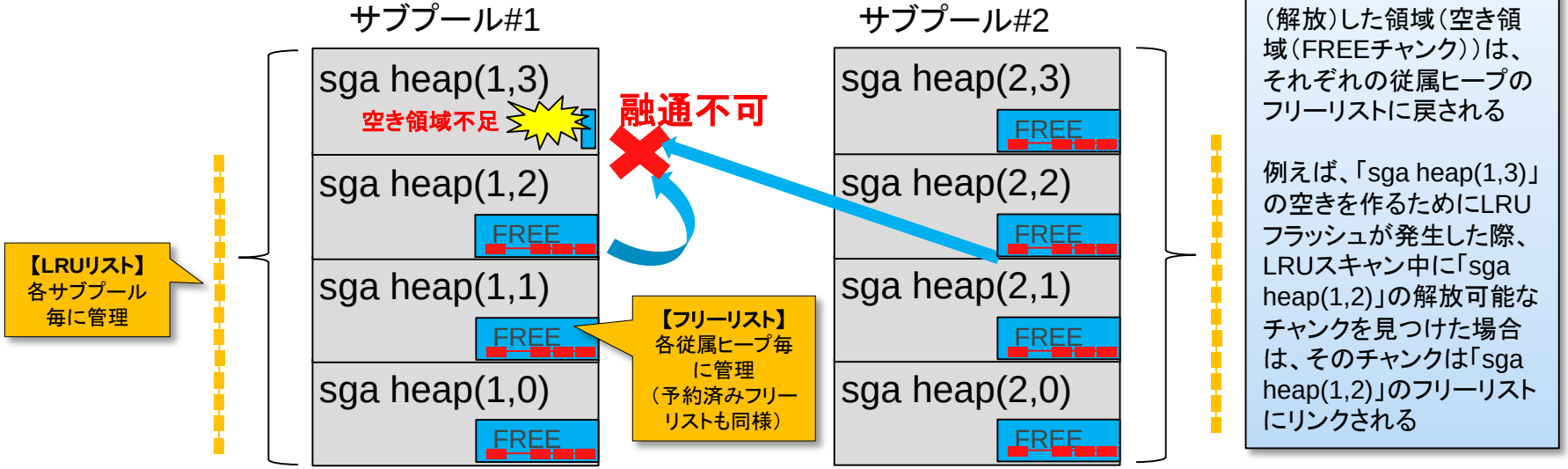
- Reserved Granule
  - インスタンス起動直後等に存在するグラニュル全体がまだ未使用の領域
  - V$SGASTAT の「free memory」は Reserved Granule を含む



共有プールの基本的理解（10／18）

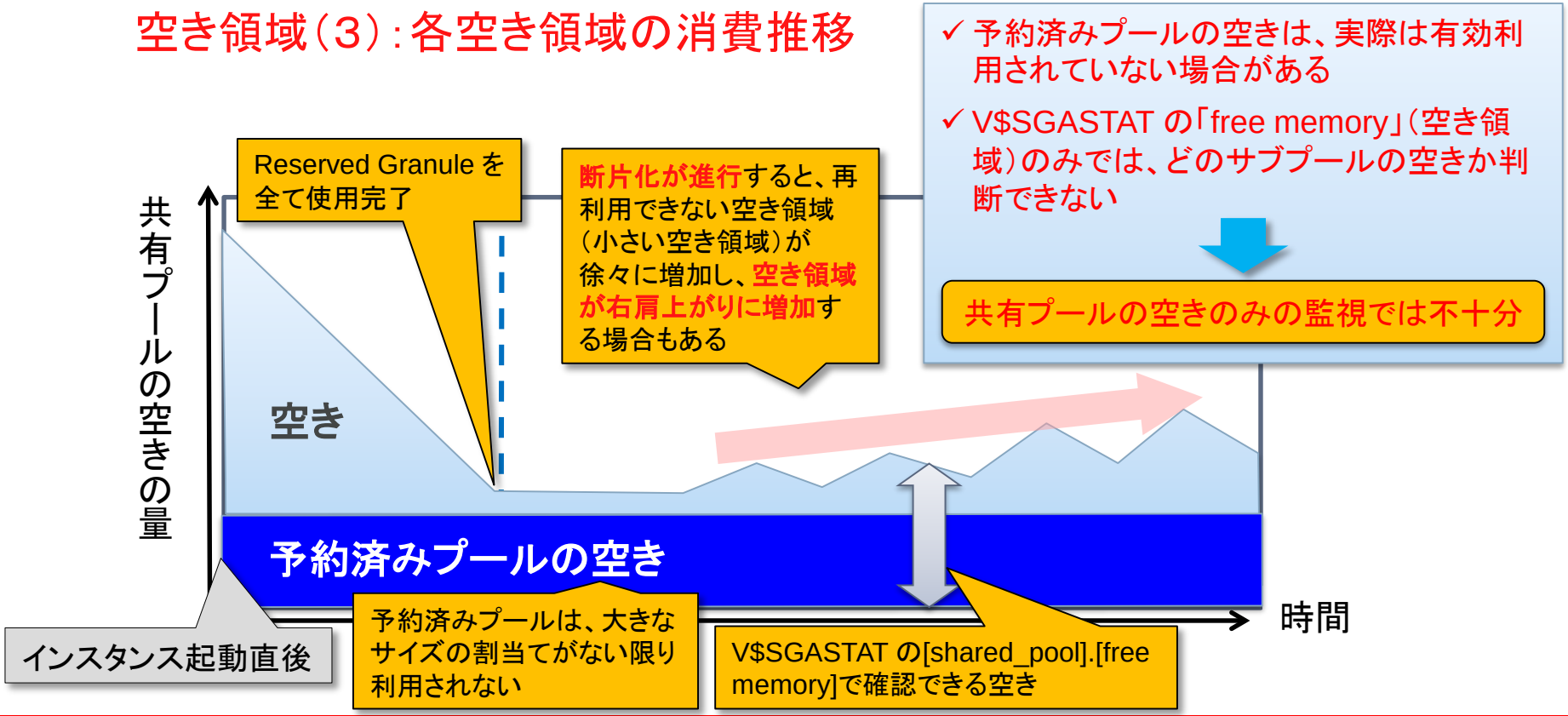
空き領域（2）：分割された各サブプール/従属ヒープ内の空き領域の管理

- 空き領域は各従属ヒープ毎に個別に管理/使用される
  - 特定のサブプールや従属ヒープにおいて空き領域が枯渇した場合でも他のサブプールや従属ヒープの空きメモリは使用されない



共有プールの基本的理解（11／18）

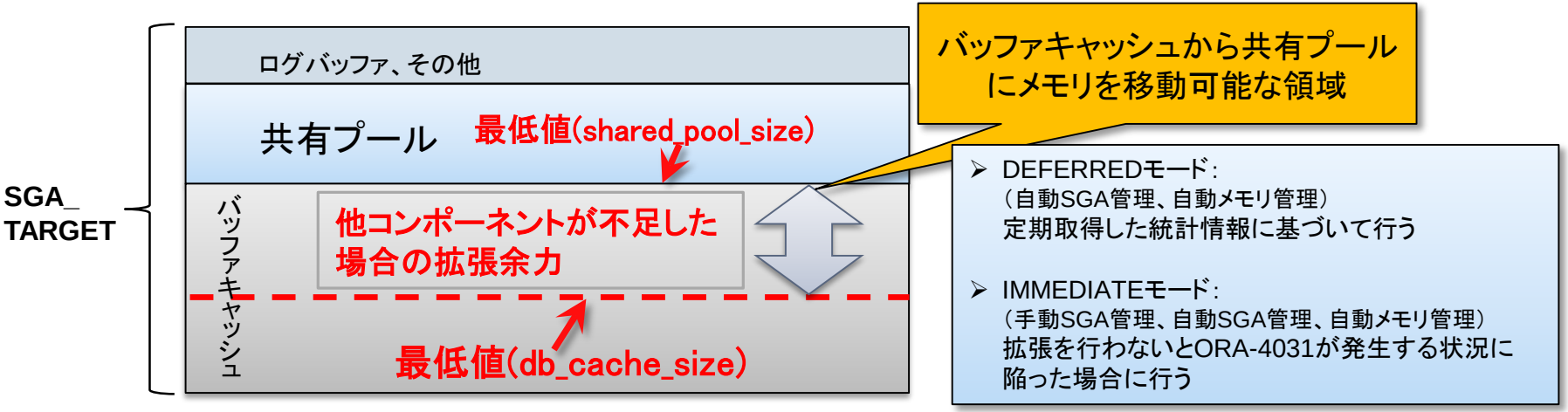
空き領域（3）：各空き領域の消費推移



共有プールの基本的理解（12／18）

空き領域（4）：共有プールの空き領域予備軍

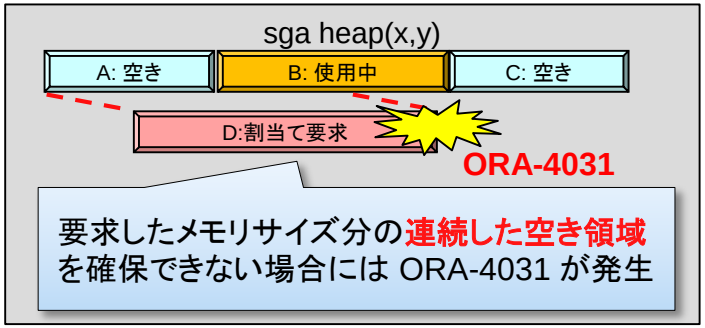
- バッファキャッシュ
  - 共有プールが不足すると、バッファキャッシュを減らして共有プールを拡張する（自動調整はグラニュル単位）
  - DEFERREDモード(遅延要求) と、IMMEDIATEモード(即時要求)がある



共有プールの基本的理解（13／18）

メモリ割当てエラー（ORA-4031）

- ORA-4031の発生ケース
  - 連続した空き領域を確保できない場合に発生するエラー
  - 従来からの代表的な発生例は、サイズが小さい、断片化によるもの
  - 昨今のメモリ低コスト化、大規模化によって、余裕を持った共有プールのサイジングが可能となり、共有プール全体のサイズが小さいことが原因でORA-4031が発生するという事例は比較的減少傾向（総量は足りているケースが多い）



共有プールの基本的理解（14／18）

最近のORA-4031の傾向

最近の発生パターン	実例
サブプール間の偏り	特定の機能に依存した処理の大量使用により、一部のサブプールが肥大化し、他のサブプールへの割り当てが失敗（他のサブプールのサイズ（空き領域）が小さくなったため）
従属ヒープ（存続期間による分割）間の偏り	リテラルSQLを多く使用した環境において、共有SQL領域（sga heap（n,3））が肥大化した結果、存続期間が長い従属ヒープ（sga heap（n,1））への割り当てが失敗
共有プールの拡張余力がない（共有プール拡張時の供給元のバッファキャッシュの余力がない）	バッファキャッシュ最低サイズ（db_cache_size）の設定不備（不必要に大きく設定されていた）により、共有プールの自動拡張ができず、割り当てが失敗

共有プールの空き領域監視のみでは検知できない

共有プールの基本的理解（15／18）

チャンクの種類

種類	用途
Free	・フリーリストで管理される未使用のチャンク ・獲得要求に対し、最初に使用される領域
Recreatable	・LRUリストで管理されるチャンク ・空きが十分にない場合、使用済みのチャンクのうち、使用頻度が低いチャンクから適宜解放され各従属ヒープ sga heap(x,y) のフリーリストに戻される ・「共有プールのフラッシュ」時に解放対象となるチャンク
Freeable	・明示的に解放命令を出すことで解放されフリーリストに戻されるチャンク
Permanent	・割当て先のヒープ自体が解放されるまで解放されないチャンク ・通常、インスタンス起動中は解放されない ・用途に応じてPermanentチャンク間での再利用動作がある

【共有プールのフラッシュ】

・共有プールのフラッシュは、使用済みの再利用可能なチャンクを解放し、使用中のチャンクは解放しない。
（用途によっては、Recreatableチャンクと関連付けられた Freeableチャンクも解放される。）

・ORA-4031 が発生する状況下では、既にLRUリストからのフラッシュを経てメモリの獲得が失敗しているため、共有プールのフラッシュと同等の処理を実施済みと言える。
よって、エラー発生後にフラッシュを実行しても、必ずしもエラーが解消するものではない。エラー発生後の効果的な対処は、アプリケーション終了後にフラッシュする方法。

共有プールの基本的理解（16／18）

構成変更時の注意事項

- 構成変更する際はサブプール毎のサイズに注意
  - CPUを増設する際
  - 共有プールのサイズを変更する際

【共有プールの分割数（11gR2の計算式）】

KROWN#147122 CPU 数、共有プールサイズによる共有プールの分割について

共有プールの分割数 = min(min(A, max(B , C)), 7)

A = trunc(((CPU_COUNT - 1) / 4) + 1)
B = trunc(SHARED_POOL_SIZE / 512M)
C = trunc((SGA_TARGET / 2) / 512M)

構成変更により、**サブプール1つあたりのサイズが変化**することがあるので注意
（特に小さくなった場合に、領域枯渇のリスクが高まるため注意）

前提：
MEMORY_TARGET を設定していない、または、0 に設定しており、かつ、SGA_TARGET > 0 を設定している場合

共有プールの基本的理解（17／18）

テストの網羅性について

理想的なテスト方針	テスト内容
実際の処理内容、運用形態に合わせたテスト	✓ 一週間で再起動するシステムであれば、一週間で実行されるすべての処理パターンを網羅した負荷を、実行される順序も考慮して、負荷をかけるのが理想 ✓ 長期間稼働してもORA-4031が発生しないことの確認はもちろん、共有プールの動的なサイズ変動や、サブプールの偏りなどを分析する

数か月～1年以上も再起動しないようなシステムで、断片化の進行までも考慮したロングランテスト（長期走行テスト）は一般的に困難

現実的なテスト方針	テスト内容
処理内容を疑似的にシミュレーションしたテスト	✓ 処理内容および負荷をできる限り疑似的にシミュレーション/短縮化して負荷テストを実施 ✓ 負荷をかけ続けことよりも、短時間でもいいので、「想定負荷」を、「本番想定の処理順序」で「網羅的」にテストすることを重視する ✓ 短期間で共有プールの拡張余力を使い尽くす処理や、特定のサブプールの偏りに関しては、短期のテストで検出できるはずであり、仮に偏りが生じた場合はテスト段階で原因を明らかにする

テストに完全な網羅性がないリスクは、拡張余力のリアルタイム監視で担保

共有プールの基本的理解（18／18）

まとめ

- 共有プールの構造
  - 最大で**28個に分割**される（サブプール分割/存続期間による分割）
  - サブプール間での**偏り**が生じる場合がある
  - 特定のサブプールや従属ヒープにおいて空き領域が枯渇した場合でも他のサブプールや従属ヒープの空きメモリは使用されない

共有プールの空き領域監視のみでは空きの枯渇は検知できない

テストに完全な網羅性がないリスクは、拡張余力のリアルタイム監視で担保

設計～テスト～運用フェーズでの検討事項

1. 管理方式選定
   - 手動SGA管理、自動SGA管理、自動メモリ管理の選択

2. 一次サイジング（自動SGA前提）
   - 主要チャンクのサイズを意識した一次サイジング
   - 共有プール自動拡張余力の確保

3. 監視・チューニング（自動SGA前提）
   - テスト・運用フェーズの監視
   - チューニング（二次サイジング）

設計

テスト・運用

1. 管理方式選定

共有プール(SGA)管理方式比較

比較項目	手動SGA管理	自動SGA管理（10gR1～）	自動メモリ管理（11gR1～）
概要	・SGAの各SGAコンポーネント（バッファキャッシュ、共有プール、ラージプールなど）のサイズを個別に設定する	・SGA全体のサイズを設定し、各SGAコンポーネント（バッファキャッシュ、共有プール、ラージプールなど）のサイズは、必要に応じて動的に調整される	・SGAとPGAの合計サイズを設定し、PGAと各SGAコンポーネント（バッファキャッシュ、共有プール、ラージプールなど）のサイズは、必要に応じて動的に調整される
特徴	・各コンポーネントの精緻なサイジングが必要 ・手動SGA管理でも、共有プールが不足した場合は、IMMEDIATEモードで、バッファキャッシュを縮小し、動的に共有プールが拡張される（11gR2～） （KROWN#151272）	・一次サイジングが比較的容易 ・各SGAコンポーネントの最低サイズを設定可能 ・11gR2では広く採用されている	・一次サイジングが比較的容易 ・PGA、各SGAコンポーネントの最低サイズを設定可能 ・LinuxではHugepageとの併用ができない ・PGAのサイズ変動により、同じ処理の性能が変わる可能性あり

1. 管理方式選定

手動SGA管理、自動SGA管理、自動メモリ管理の採用率
日本オラクルでコンサル支援することが多い中～大規模のシステムでの経験値

- **自動SGA管理 ⇒ 95%以上の採用率、推奨設定**
  - 11gR2では、経験則的に95%以上のシステムで、広く一般的に採用されており、日本オラクルコンサルが支援する場合に、初めに検討する方式
- 手動SGA管理 ⇒ 数%程度の採用率
  - 旧バージョンからの移行のお客様以外、新規に採用するケースはほとんどない
- 自動メモリ管理 ⇒ 数%程度の採用率
  - 中～大規模なシステムでもOSとしてLinuxが選定されることが多くなってきており、Hugepageと併用できないため、積極的に採用されるに至っていない

設計～テスト～運用フェーズでの検討事項

1. 管理方式選定
   - 手動SGA管理、自動SGA管理、自動メモリ管理の選択

2. 一次サイジング（自動SGA前提）
   - 主要チャンクのサイズを意識した一次サイジング
   - 共有プール自動拡張余力の確保

3. 監視・チューニング（自動SGA前提）
   - テスト・運用フェーズの監視
   - チューニング（二次サイジング）

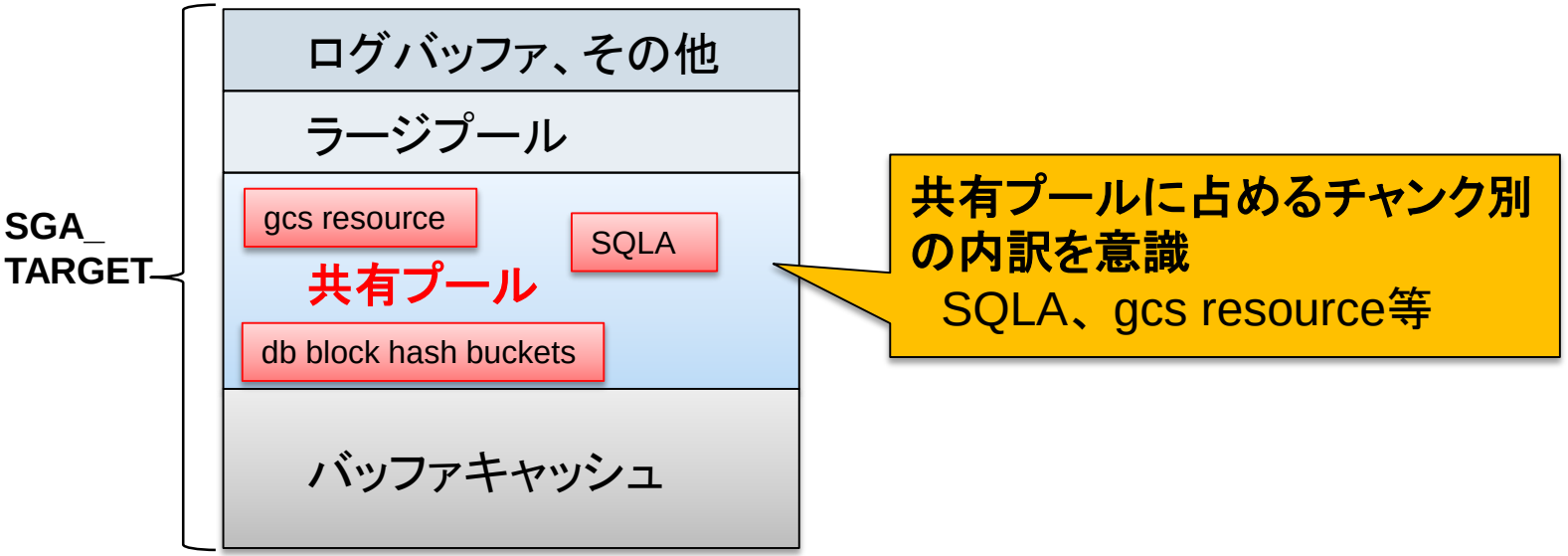
設計

テスト・運用

2. 一次サイジング

SGAを構成するコンポーネントで特に重要な共有プールの見積もり

- 共有プールに占める主要なチャンク領域を見積もる
  - 共有SQL領域、RACのGCS領域など、共有プールに大きな割合を占めるチャンクを意識して、一次サイジングを実施する

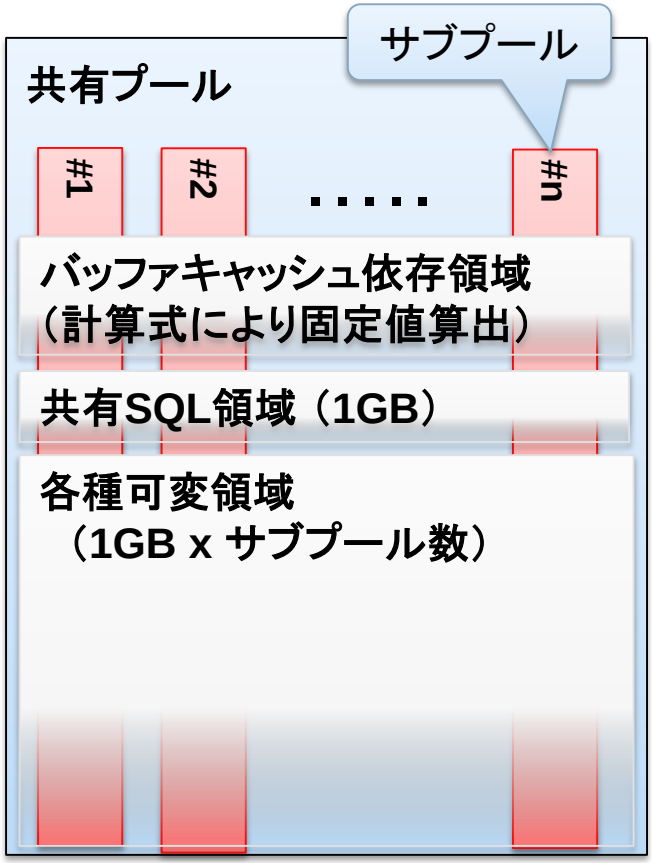


2. 一次サイジング

共有プールを3つの領域に分割して一次サイジング（1）

- バッファキャッシュ依存領域
  - バッファキャッシュのブロック数に応じて計算する
- 共有SQL領域（1GB）
  - 経験則的に十分なサイズを確保する
- 各種可変領域（1GB x サブプール数）
  - サブプール数に応じて一次サイジングする

完全な見積もりは困難であるため、3つの領域を、ある程度の前提を置いて一次サイジングし、テストの結果でチューニングする



2. 一次サイジング

共有プールを3つの領域に分割して一次サイジング（2）

- 初期設定値計算イメージ
  - 3つの領域の合計に余裕率を掛けて計算する

shared_pool_size初期設定値 ＝

バッファキャッシュ依存領域（固定領域）
＋ 共有SQL領域（可変領域）
＋ 各種可変領域（可変領域）

X

1.2倍（余裕率）

2. 一次サイジング

固定領域

バッファキャッシュ依存領域（RAC/Single共通）

- db_block_hash_buckets
  - バッファキャッシュを管理するハッシュ・バケットの領域がキャッシュされる
  - バッファキャッシュのブロック数に比例して大きくなり、大規模なバッファキャッシュを確保するシステムでは、共有プールのサイジングに注意が必要

    KROWN#129856
    バッファ・キャッシュのサイズ変更時の注意点

  - サイズ見積もり（11gR2の場合）
    - **バッファキャッシュ1GBにつき15MBで計算**

      ※ 11gR2の実績から導出した参考値。実機確認の上、調整する。

      ※ db_block_size=8KBの前提

2. 一次サイジング

固定領域

バッファキャッシュ依存領域（RACのみ）

- gcs resource/gcs shadow
  - RACノード間で、バッファキャッシュ上のデータブロックの整合性を管理するためのロック可能な実体
  - gcs resource/gcs shadowはバッファキャッシュのブロック数に比例して大きくなり、大規模なバッファキャッシュ環境では、共有プールを圧迫する可能性あり
  - サイズ見積もり（11gR2の場合）
    - **db_block_size=8Kでは、バッファキャッシュ1GBにつき45MBで計算**

    ※ 11gR2で2～3ノード環境の実績から導出した参考値。gcs shadowはノード数によって異なりますので、実機確認の上、調整する

    ※ 原則的には起動時のサイズで固定的に確保されるが、DRM（Dynamic Resource Mastering）などによるリソースマスタの偏りによって、変動する可能性あり

2. 一次サイジング

可変領域

共有SQL領域

- SQLA（共有SQL領域（共有カーソル領域））
  - SQLテキストや実行計画などがキャッシュされる
  - SQLが十分に共有化されていれば、大規模なシステムでも、**一般的に1GBあれば十分**であり、可変領域だが、一次サイジングでは、一旦、1GBと見積もる

【SQLが十分に共有された状態】

下記2点をいずれも満たす：

✓ SQLのヒット率（AWRの「Soft Parse %」）が95%以上

✓ 2回以上実行されたSQLの共有プール占有率（AWRの「% SQL with executions>1」）が95%以上

【高SQLヒット率でもリテラルSQLが多い例】

・数種類の共有されたSQLを100万回実行
・リテラルSQLを1万回実行

⇒ **SQLのヒット率99%**

  - リテラルSQLが多い、子カーソルが多く生成されるようなシステムで肥大し易いため、大きく見積もる。このケースでは、後述のKGLHxも肥大し易い

2. 一次サイジング

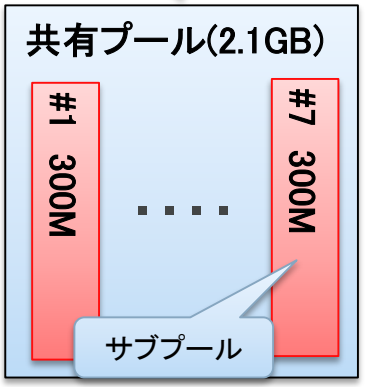
各種可変領域

可変領域

- 各種可変領域の机上見積もりは不可能
  - 共有SQL以外の可変領域で、そのサイズは、使用する機能、使い方、同時実行並列度に依存するため、見積もりは困難
  - 代表的なチャンク(KQR、GESなど)の注意点は後述する
- サブプール数から仮見積もりする
  - サブプールあたりのサイズを十分に確保する

    ⇒ **サブプール数 x 1GB**
    - CPU数が多い(サブプールが多い)システムでは、処理の多様性が高くなり、より多くの領域が必要
    - プール分割に起因する障害防止

【悪い例】
総量では2.1GBと大きいが、サブプールが小さいため、更に従属ヒープに分割され、ORA-4031が発生し易い



2. 一次サイジング

一次サイジングの計算式（自動SGA前提）

一次サイジングで暫定的な設定値を決めるための計算式です。テストの結果により調整してください。

shared_pool_size初期設定値 ＝

バッファキャッシュ依存領域（固定領域）
＋ 共有SQL領域（可変領域）
＋ 各種可変領域（可変領域）

X 1.2倍（余裕率）

shared_pool_size初期設定値(GB) ＝

db_cache_size(GB) x (15MB + 45MB) / 1024
（バッファキャッシュ依存領域）
＋ 1GB（共有SQL領域）
＋ 1GB x サブプール数（各種可変領域）

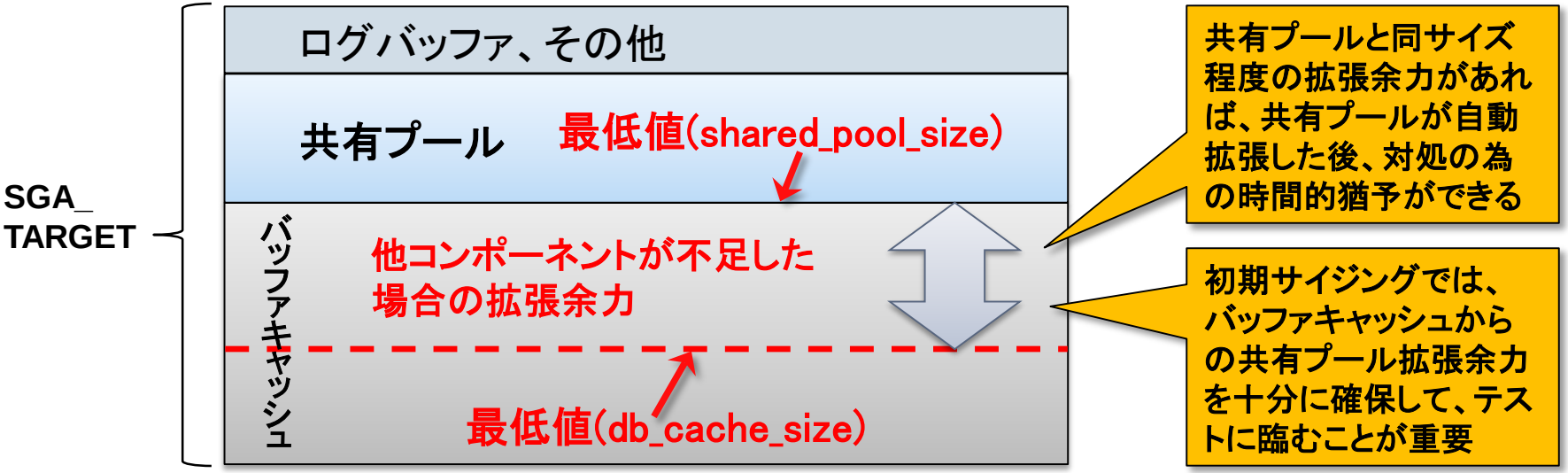
1GBあたりのdb block hash bucketsとGCS

X 1.2倍（余裕率）

2. 一次サイジング

共有プール自動拡張余力の確保

- db_cache_sizeとshared_pool_sizeには最低サイズを設定する
- [sga_target > 各SGAコンポーネント設定値の和]になるよう、**共有プールの拡張余力を十分に確保する**



2. 一次サイジング

見積もり例1（自動SGA前提）

【前提】
共有プールのサブプール数：3個
Physical Memory ： 64GB
db_cache_size ： 24GB
db_block_size=8K
2node RAC

shared_pool_size初期設定値(GB) ＝

db_cache_size(GB) x (15MB + 45MB) / 1024
（バッファキャッシュ依存領域）
+ 1GB（共有SQL領域）
+ 1GB x サブプール数（各種可変領域）

1GBあたりのdb block hash bucketsとGCS

X 1.2倍（余裕率）

shared_pool_size初期設定値 ＝

24 x (15+45)/1024 GB
+ 1GB
+ 1GB x 3

X 1.2倍 = 6.5GB

同サイズ

共有プールと同サイズの拡張余力

6.5GB

db_cache_size

17.5GB

24GB

2. 一次サイジング

見積もり例2（自動SGA前提）

【前提】
共有プールのサブプール数：7個
Physical Memory ： 640GB
db_cache_size ： 300GB
db_block_size=8K
2node RAC

shared_pool_size初期設定値(GB) ＝

db_cache_size(GB) x (15MB + 45MB) / 1024
（バッファキャッシュ依存領域）
+ 1GB（共有SQL領域）
+ 1GB x サブプール数（各種可変領域）

1GBあたりのdb block hash bucketsとGCS

X 1.2倍（余裕率）

shared_pool_size初期設定値 ＝

300 x (15+45)/1024 GB
+ 1GB
+ 1GB x 7

X 1.2倍 = 30.0GB

同サイズ

共有プールと同サイズの拡張余力

30GB

300GB

db_cache_size

270GB

2. 一次サイジング

初期サイジング見積もり式使用時の注意

- 本資料で紹介した初期サイジング見積もり式使用時の注意
  - 新規構築のシステムで、サイジング根拠がない場合の初期見積もりを目的にしたものです。
  - この見積もり結果は、ORA-4031などの障害抑止を保証するものではありません。
  - 11gR2のいくつかのシステムでの実機確認結果から導出した見積もり式であり、バージョンによって異なる可能性があります。
  - システム特性に応じた最適化が必要です。テストや運用中の監視、情報取得により、妥当性の確認、チューニングを実施してください。

2. 一次サイジング

可変領域

肥大化が問題になり易い各種可変領域（過去事例より）（1）

チャンク名	用途	肥大リスク
KGLH (xxx)	ライブラリキャッシュハンドル ライブラリキャッシュの管理情報（リロード回数など）	・使用するオブジェクト数が多いと肥大し易い。 ・SQL文もライブラリキャッシュとして管理されるため、リテラルSQLが多い、子カーソルが多い環境で、肥大し易い。
KQR (xxx)	ディクショナリキャッシュ （ROWCACHE）	・シーケンス、パーティション、ヒストグラム統計などがキャッシュされる。 ・キャッシュの種類によって、配置するサブプールが固定されるため、サブプール間のサイズバランスが崩れ易い。 ・パーティション数が多い環境、XMLDB(BinaryXML)使用環境などで著しい偏りの発生事例有り。
ges resource **ges enqueue**	GESリソース／GESロックをキャッシュ RAC環境のみ	・ノード間の処理の整合性を保つためのエンキューなど、グローバル管理が必要なロック数に依存するため、APの処理内容に依存して、肥大し易い。 ・LRUフラッシュや共有プールのフラッシュでは解放されない。 ・LCKプロセスが不要と判断すれば解放される。

2. 一次サイジング

可変領域

肥大化が問題になり易い各種可変領域（過去事例より）（2）

チャンク名	用途	肥大リスク
PLDIA **PLMCD**	PL/SQLのストアドオブジェクト	・PL/SQLのストアドオブジェクトを使用する環境で大きくなり易い。 ※　PL/SQLのストアドオブジェクトは4Kのチャンクに分割される
event statistics per sess **ksunfy : SSO free list** **dbktb: trace buffer**	セッションの統計など	・セッション数が多い環境では、管理するセッション情報が相対的に大きくなり易い。
Result Cache: (xxx)	リザルトキャッシュの結果セット	・リザルトキャッシュは、結果セット単位に、各サブプールにラウンドロビンに配置されるため、極端に大きな結果セットがあると、特定のサブプールが肥大する。 ・リザルトキャッシュは共有プールのフラッシュでも解放されない

2. 一次サイジング

RAC縮退運転時の考慮

- RAC環境の場合は、必要に応じて縮退運転時の余裕分を考慮する
  - 完全にアプリケーションパーティショニングされているRAC環境では、縮退運転時に、生存ノードでは全く異なるタイプの処理を受け付けることになる

【例】
・ノード間で実行されているSQLが異なるため、縮退時のSQLAが増加
・ノード間でアクセスするオブジェクトが異なるため、ディクショナリキャッシュが増加
・ノード間で発生するロックの種類が異なるため、GES関連領域が増加

- 縮退運転のテストにて増加分を検討する
  - 見積もりは困難であるため、縮退運転のテストにおいて、特定チャンクが過剰に増加していないか確認の上、共有プールのサイズ追加を検討する

設計～テスト～運用フェーズでの検討事項

1. 管理方式選定
   - 手動SGA管理、自動SGA管理、自動メモリ管理の選択

2. 一次サイジング（自動SGA前提）
   - 主要チャンクのサイズを意識した一次サイジング
   - 共有プール自動拡張余力の確保

3. 監視・チューニング（自動SGA前提）
   - テスト・運用フェーズの監視
   - チューニング（二次サイジング）

設計

テスト・運用

3. 監視・チューニング

共有プールの状態と監視

共有プールの状態は刻々と変化する

- 共有プールの状態は、テストフェーズ、運用フェーズで刻々と変化している
- 共有プールの状態は、単体テストと総合テストで異なることはもちろん、負荷の掛け方によっても異なる
- 運用フェーズでも、共有プールの状態は、オンラインとバッチの違い、負荷の増大、新規アプリのリリースなど様々な要因で変化する

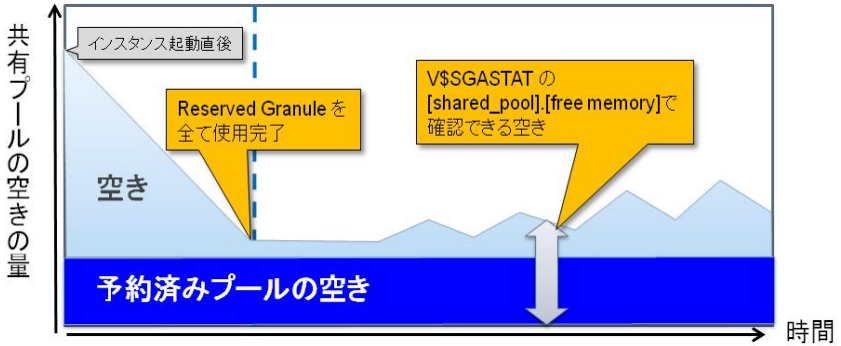
共有プールの監視・情報収集が重要

テストフェーズ、運用フェーズで、共有プールの状態遷移を適宜、検出できるような監視および情報収集を準備することが重要

3. 監視・チューニング

一般的に採用されている空き率分析の問題点

- 一般的な共有プールの監視
  - 「共有プールの空き率監視」、「チャンク別のサイズ遷移分析」などが一般的
- 共有プールの空き率分析の問題点



  - V$SGASTATの[shared pool].[free memory]には、予約済プールが含まれるため、空きが枯渇することはほぼない
  - 空きが少なくても、使用頻度の低いチャンクを再利用できていれば問題ない
  - サブプール別、従属ヒープ別の空きは確認できない
  - 断片化が進行すると、再利用可能な連続領域が減少し、逆に空き率が上昇傾向を示すことがある

3. 監視・チューニング

推奨する共有プールの監視

- 共有プールに関するリアルタイム監視は①で実施する
- ②～⑥と高負荷の⑦は必要に応じて情報収集を検討する

◎:必須
○:推奨
△:必要に応じて検討

No.	監視／分析の内容	監視／情報採取の方法	種別	テストフェーズ	運用フェーズ
①	共有プール拡張余力の監視	V$SGA_DYNAMIC_COMPONENTSによる自動拡張余力の監視	リアル監視	○	◎
②	共有プールのサイズ遷移の傾向分析	AWRレポートの「Memory Dynamic Components」セクション参照	情報取得	◎	◎
③	チャンク別サイズ遷移の傾向分析	AWRレポートの「SGA breakdown difference」セクション参照	情報取得	◎	○
④	サブプール別の偏り傾向分析	X$KSMSSのロギング	情報取得	◎	○
⑤	予約済みプールの傾向分析	V$SHARED_POOL_RESERVEDのロギング	情報取得	◎	○
⑥	巨大SQLを検知する	Memory Notificationの閾値変更(50MB⇒2MB、等)	情報取得	○	△
⑦	従属ヒープ別の偏り傾向分析	X$KSMSPのロギング(※latchを掴むため高負荷)	情報取得	△	△

3. 監視・チューニング

運用 ◎

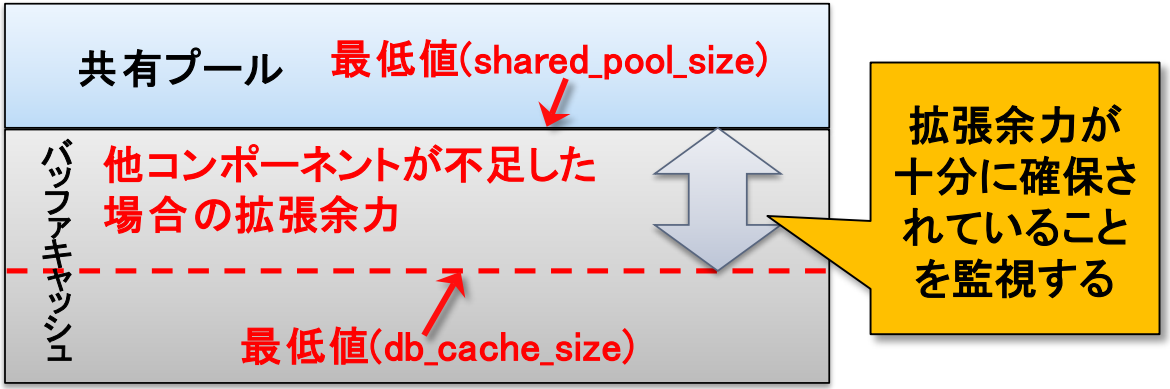
テスト ○

①共有プール拡張余力の監視（リアルタイム監視）（1）

- 自動SGA管理の設計指針として重要なのは、「共有プールの拡張余力を十分に残す」こと

共有プールの拡張余力を監視する

- インスタンスの再起動を頻繁にできない環境では、後述の対処策を実施するための時間的猶予を確保するため、早めに検知できる閾値を設定する
- 監視間隔は30分～60分間隔程度で十分



3. 監視・チューニング

運用 ◎

テスト ○

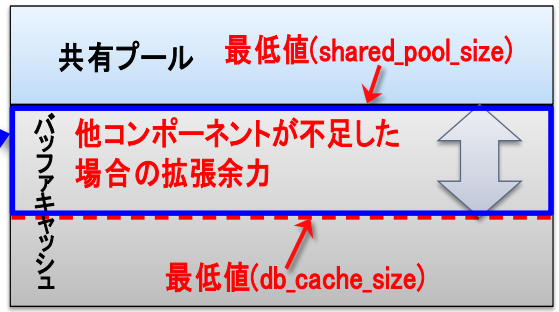
①共有プール拡張余力の監視（リアルタイム監視）（2）

- V$SGA_DYNAMIC_COMPONENTSによる拡張余力の監視
  - バッファキャッシュの【現在値(CURRENT_SIZE)】－【初期設定値(USER_SPECIFIED_SIZE)】に十分余裕があることを確認
  - バッファキャッシュの「現在値」から「初期設定値」を引いた数値が共有プールの自動拡張余力と言える

【例】「2GB」のdb_cache_sizeを設定したが、現在は「4G程」（4,385,875,968byte）であり、共有プールが不足した場合、「2G程」（2,238,392,320byte）の拡張余力がある

```
SELECT USER_SPECIFIED_SIZE,CURRENT_SIZE,
       CURRENT_SIZE - USER_SPECIFIED_SIZE SIZE_DIFFERENCE
 FROM V$SGA_DYNAMIC_COMPONENTS
 WHERE COMPONENT = 'DEFAULT buffer cache';

USER_SPECIFIED_SIZE     CURRENT_SIZE  SIZE_DIFFERENCE
------------------- ---------------- ---------------
      2,147,483,648    4,385,875,968    2,238,392,320
```



3. 監視・チューニング

運用 ◎

テスト ○

①共有プール拡張余力の監視(リアルタイム監視)(3)

【拡張後サイズの記録】

- 各領域の変動後サイズは(二重アンダースコア)“_ _<領域名>_size”に記録されている。_ _shared_pool_size、_ _db_cache_sizeなど。
- spfileを使用している場合、この設定値はインスタンスが停止しても、spfileに記録されているため、インスタンスを起動すると停止前のサイズに戻る

【監視閾値を越えた場合の対処】

- sga_target/shared_pool_sizeを見直す
- **二重アンダスコアパラメータを変更・初期化してインスタンスを再起動する**
- インスタンスを再起動できない場合、暫定対処としてdb_cache_sizeを小さくする

【_ _shared_pool_sizeの設定変更例】

```
ALTER SYSTEM SET "__shared_pool_size"=1024M SCOPE=SPFILE SID= 'ORCL1';
```

【_ _shared_pool_sizeの初期化例】

```
ALTER SYSTEM RESET "__shared_pool_size" SCOPE=SPFILE SID= 'ORCL1' ;
```

【db_cache_sizeの初期値縮小例】

```
ALTER SYSTEM RESET  "db_cache_size" = 100G SCOPE=MEMORY SID= 'ORCL1' ;
```

3. 監視・チューニング

運用 ◎

テスト ◎

②共有プールのサイズ遷移の傾向分析（情報取得）（1）

- 共有プールが自動SGAの自動拡張機能で拡張を続けていないか確認
  - IMMEDIATEモードや、DEFERREDモードによる自動拡張が頻発していない（**各コンポーネントのサイズが安定推移している**）ことを確認する
  - 共有プールが拡張し続けるような傾向にあれば、①の監視閾値を越える前に、対策をする必要がある
  - 処理特性が同じ時間帯に、共有プールとバッファキャッシュの間で、短時間に頻繁に奪い合いが発生（増減を繰り返す）していたら、SGAが不足していると考えられる

【対処】
・③の分析で特定チャンクの過剰消費があれば原因を分析する
・sga_target/shared_pool_sizeを見直す

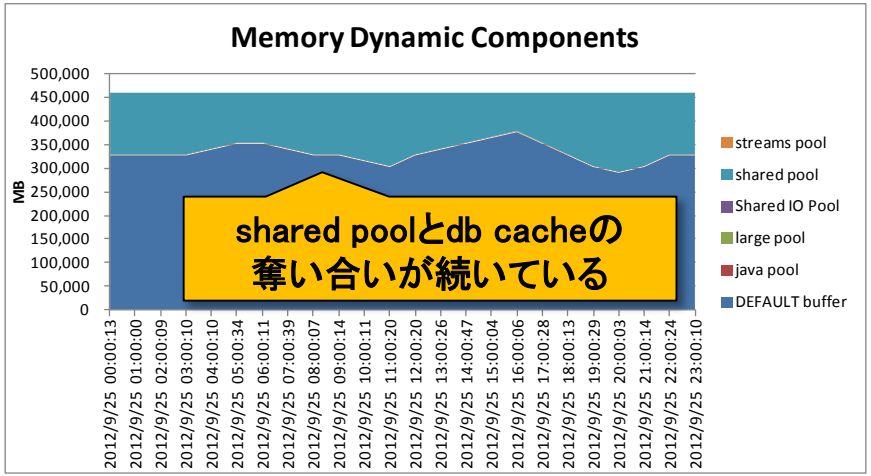
3. 監視・チューニング

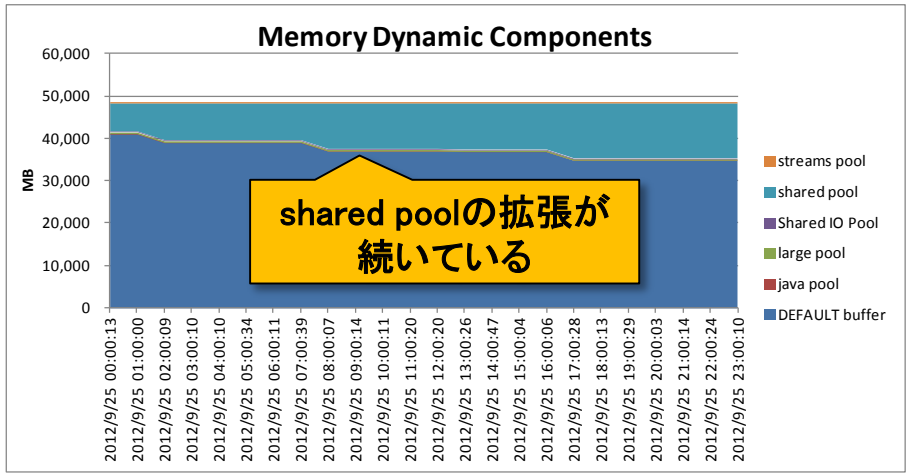
運用 ◎

テスト ◎

②共有プールのサイズ遷移の傾向分析（情報取得）（2）

- SGAコンポーネントのサイズ遷移を確認する
  - AWRレポートの「Memory Dynamic Components」セクションの時系列データ
  - 定期取得したV$SGA_DYNAMIC_COMPONENTS
    （①のリアルタイム監視と情報ソースは同じ）





3. 監視・チューニング

運用 ◎

テスト ◎

②共有プールのサイズ遷移の傾向分析（情報取得）（3）

- SGAコンポーネントの変動履歴情報を確認する
  - AWRレポートの「Memory Resize Ops」セクション
  - V$SGA_RESIZE_OPS

Memory Resize Ops DB/Inst: ORCL/orcl1 Snaps: 1002-1003
-> Oper Types/Modes: INItializing,GROw,SHRink,STAtic/IMMediate,DEFerred

Memory Resize Ops DB/Inst: ORCL/orcl1 Snaps: 1001-1002
-> Oper Types/Modes: INItializing,GROw,SHRink,STAtic/IMMediate,DEFerred
Delta : change in size of the component
Target Delta: displayed only if final size <> target_size
-> Status: COMplete/CANcelled/INActive/PENding/ERRor
-> ordered by start_time desc,component

Start	Ela (s)	Component	Oper Typ/Mod	Init Size (M	Delta	Target Delta	Final (M)	Sta
02/16 22:20:11	0	bufcache	SHR/DEF	28,596	-128	N/A	28,464	COM
02/16 22:20:11	0	shared	GRO/DEF	5,632	128	N/A	5,760	COM
02/17 10:15:23	0	bufcache	SHR/IMM	28,464	-128	N/A	28,336	COM
02/17 10:15:23	0	shared	GRO/IMM	5,760	128	N/A	5,888	COM
02/17 21:35:01	0	bufcache	SHR/DEF	28,336	-128	N/A	28,208	COM
02/17 21:35:01	0	shared	GRO/DEF	5,888	128	N/A	6,016	COM

3. 監視・チューニング

運用 ◎

テスト ◎

②共有プールのサイズ遷移の傾向分析（情報取得）（4）

- 性能テスト・本番運用開始時の注意
  - **テスト、運用開始時点で、共有プールの拡張余力が十分残っていることを、AWRやV$SGASTATなどで確認すること**
  - テストデータの作成、大量オブジェクトのコンパイル、移行データの作成などで、実運用時と異なる用途の領域により共有プールが自動拡張している場合がある
  - このような状態でテスト、運用を開始することは、設計意図と異なっているため、二重アンダースコアパラメータを変更、初期化すること

3. 監視・チューニング

運用　○

テスト　◎

③チャンク別サイズ遷移の傾向分析（情報取得）（1）

- 共有プールのチャンクサイズの推移を確認する
  - 空き領域と、先に上げた、共有SQL領域、RAC依存領域など重要なチャンクを中心に、サイズの推移を確認する
  - 各チャンクサイズが安定的に推移しているか、また、特定チャンクが増加し続ける傾向がないか確認する
  - **空き領域の大小をそれほど注目して見る必要はない**

3. 監視・チューニング

運用　○

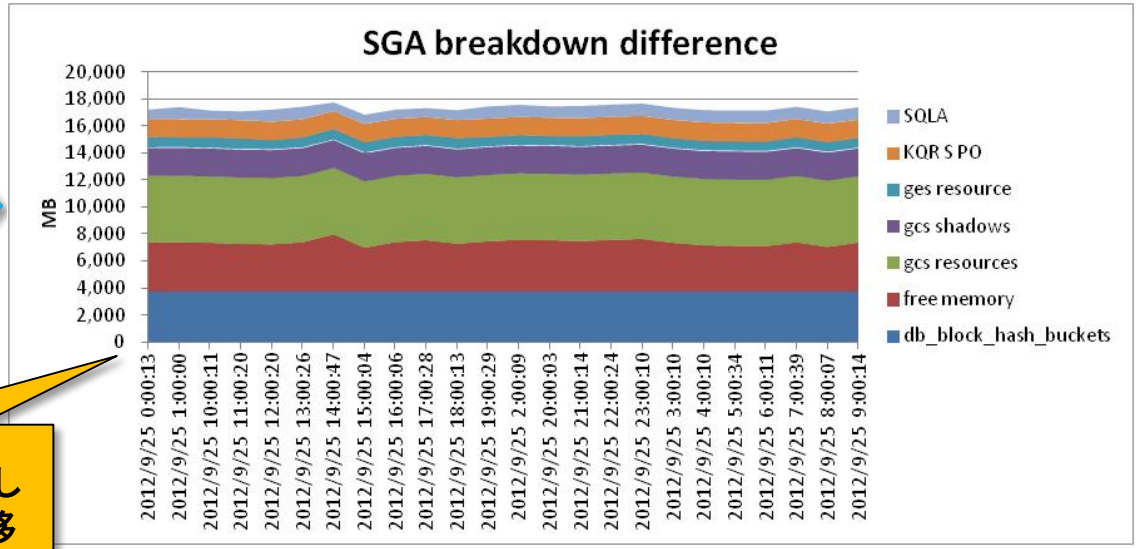
テスト　◎

③チャンク別サイズ遷移の傾向分析（情報取得）（2）

- AWRレポートの「SGA breakdown difference」より時系列にグラフ化
- 定期的取得したV$SGASTATより時系列にグラフ化

SGA breakdown difference　DB/Inst: ORCL/orcl1　Snaps: 1003-1004
-> ordered by Pool, Name
-> N/A value for Begin MB or End MB indicates the size of that Pool/Name was insignificant, or zero in that snapshot

Pool	Name	Begin MB	End MB	% Diff
java	free memory	512.0	512.0	0.00
large	free memory	1,015.8	1,015.8	0.00
shared	KQR S PO	2,839.0	2,839.0	0.00
shared	ges resource	758.8	758.8	0.00
shared	SQLA	902.0	902.0	0.00
shared	db_block_hash_buckets	4,696.0	4,696.0	0.00
shared	free memory	5,320.2	5,320.2	0.00
shared	gcs resources	5,964.5	5,964.5	0.00
shared	gcs shadows	3,082.6	3,082.6	0.00
stream	free memory	1,024.0	1,024.0	0.00
	buffer_cache	296,480.0	296,480.0	0.00
	fixed_sga	2.1	2.1	0.00
	log_buffer	656.1	656.1	0.00
	shared_io_pool	512.0	512.0	0.00
log_buffer		656.1	656.1	0.00
shared_io_pool		512.0	512.0	0.00



特定のチャンクが肥大し続けることなく安定推移

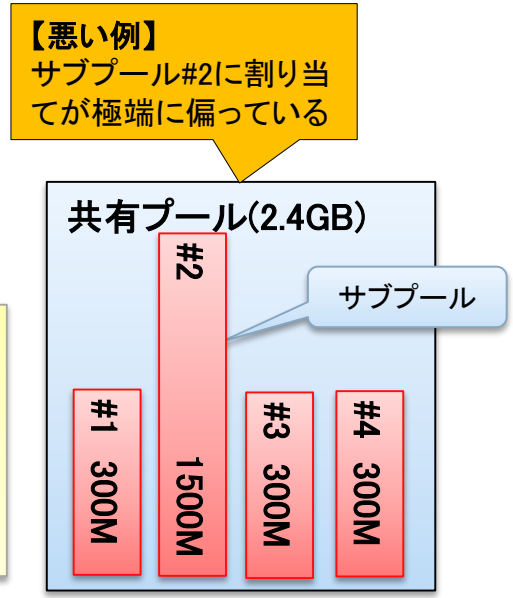
3. 監視・チューニング

運用　○
テスト　◎

④サブプール別の偏り傾向分析（情報取得）（1）

- サブプール間の偏りを確認する
  - 共有プール全体として空きがあっても、特定のサブプールの肥大によりORA-4031が発生することがある
  - リアルタイム監視は不要だが、テストの時や、新規アプリリリース後の本番環境などで、特定のサブプールに偏りがないか確認し、極端な偏りは、原因を明らかにする

【偏りが見つかった場合の対処】
- 偏りの原因を分析する
- sga_target/shared_pool_sizeを見直し、**各サブプールが十分大きくなるように調整する**
  （サブプール別の割り当てサイズは手動制御できないため）



3. 監視・チューニング

運用 ○

テスト ◎

④サブプール別の偏り傾向分析(情報取得)(2)

- X$KSMSS(V$SGASTATの元表)で、サブプール別のサイズと空きを確認

【例】

```
SELECT
  KSMDSIDX SUBPOOL#,DECODE(KSMSSNAM,'free memory','free memory','used memory') TYPE,SUM(KSMSSLEN) BYTES
FROM X$KSMSS
GROUP BY KSMDSIDX,decode(KSMSSNAM,'free memory','free memory','used memory')
ORDER BY KSMDSIDX,decode(KSMSSNAM,'free memory','free memory','used memory');
```

SUBPOOL#	TYPE	BYTES
0	free memory	0
0	used memory	0
1	free memory	52,393,584
1	used memory	232,819,088
2	free memory	53,940,240
2	used memory	231,272,432
3	free memory	46,664,144
3	used memory	221,771,312
4	free memory	60,816,984
4	used memory	190,841,256

Reserved Granuleの空きは
すでに枯渇

4つのサブプールに分割されており、
ほぼ均等に割り当てられている

偏りがある場合は、DECODEとSUM()を外して、
何のチャンクが偏っているか確認する

3. 監視・チューニング

運用　○

テスト　◎

⑤予約済みプールの傾向分析

- 予約済みプールの空き領域確認
  - V$SGASTAT.[shared_pool].[free memory]の値には、予約済みプールの空きが含まれており、4400byte以下の要求で使える領域ではない
  - V$SHARED_POOL_RESERVED.USED_SPACEとFREE_SPACEで、共有プール全体に占める、予約済みプールの使用サイズ、空きサイズを確認する
  - 予約済みプールとして割り当てられる、デフォルトのshared_pool_size x 5%がどの程度消費されているか確認する

【対処】

- 予約済みプールがほとんど使用されていない場合は、予約済みプールのサイズ（SHARED_POOL_RESERVED_SIZE）を小さくし、非予約済みプールに割り当てたほうが有効
- 予約済みプールの空きが枯渇している場合は、4400byte以上のチャンクが頻繁に割り当てられる原因を調査した上で、必要に応じて、共有プールの拡張を検討する

3. 監視・チューニング

運用 △

テスト ○

⑥巨大SQLを検知する(情報取得)

- Memory Notificationで巨大SQLを検知する
  - Memory Notification(_kgl_large_heap_warning_threshold)は一定サイズ以上のSQLが実行されたことを、アラート・ログにメッセージ出力する機能
  - テストで閾値を小さめに設定しておくことにより、極端に大きなSQLを検知し、事前に対処することが可能
  - 10.2.0.2以降は50MB以上のSQLが実行されると、メッセージ出力

KROWN#109941
「アラート・ログにライブラリ・キャッシュ・オブジェクトの SGA への割り当てに関するメッセージが出力される」

【アラート・ログメッセージ例】

```
Wed Dec 26 12:38:02 2012
Memory Notification: Library Cache Object loaded into SGAHeap size 1018K exceeds notification threshold (70K)
Details in trace file D:¥APP¥diag¥rdbms¥orcl¥trace¥orcl_ora_2352.trc
KGL object name :SELECT /* Test SQL */ COL01 , COL02 , COL03FROM TAB01 , TAB02 , TAB03 WHERE
```

3. 監視・チューニング

運用 △

テスト △

⑦従属ヒープ別の偏り傾向分析（情報取得）

- X$KSMSPで、従属ヒープ別のサイズと空きを確認
  - 検索時にshared pool latchを取得し負荷が高いため、本番環境や性能テスト中の採取は推奨しない
  - 従属ヒープの肥大や、断片化状態を分析するなど、個別の障害分析目的で取得することが多い
  - 性能・負荷テスト終了後や、本番のオンライン終了後、バッチ終了後のような処理の切れ目で情報取得しておくと、ORA-4031の分析に役立つ場合がある

KROWN#19607
ライブラリ・キャッシュのメモリ使用状況確認スクリプト
KROWN#132267
X$ 表から分割された共有プールの各プールごとの使用状況を確認する方法

まとめ

共有プールのサイジング、テスト、運用の重要ポイント

ORA-4031撲滅

① 自動SGA管理の使用を推奨

② 大規模バッファキャッシュ環境（特にRAC環境）では、バッファキャッシュ依存領域を上乗せしてサイジングする

③ 1サブプールあたりのサイズを十分に大きくサイジングする

④ 共有プールの自動拡張余力を残して、バッファキャッシュの最低サイズを設定する

⑤ 共有プールの自動拡張余力が残っていることをリアルタイム監視する

Hardware and Software

ORACLE

Engineered to Work Together


ORACLE®