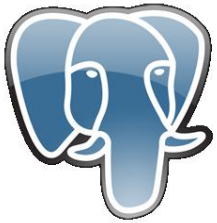
NTT Software Innovation Center

NTT



PgDay 2012 Japan

実践！PostgreSQL運用

2012.11.30

NTT OSS センタ

坂本　昌彦

アジェンダ

目標

○商用環境での運用ノウハウを持ち帰る

- 安定運用（正しい設計）のトピックを中心に
- 自身のシステムの運用で実践する

対象

○初級者，中級者

○運用する人，設計する人

トピック

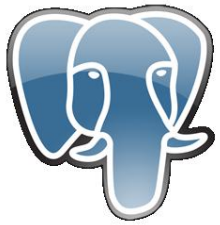
○ Part1: 接続数と安定運用

○ Part2: メモリ設定と安定運用

○ Part3: ディスク設定と安定運用

免責

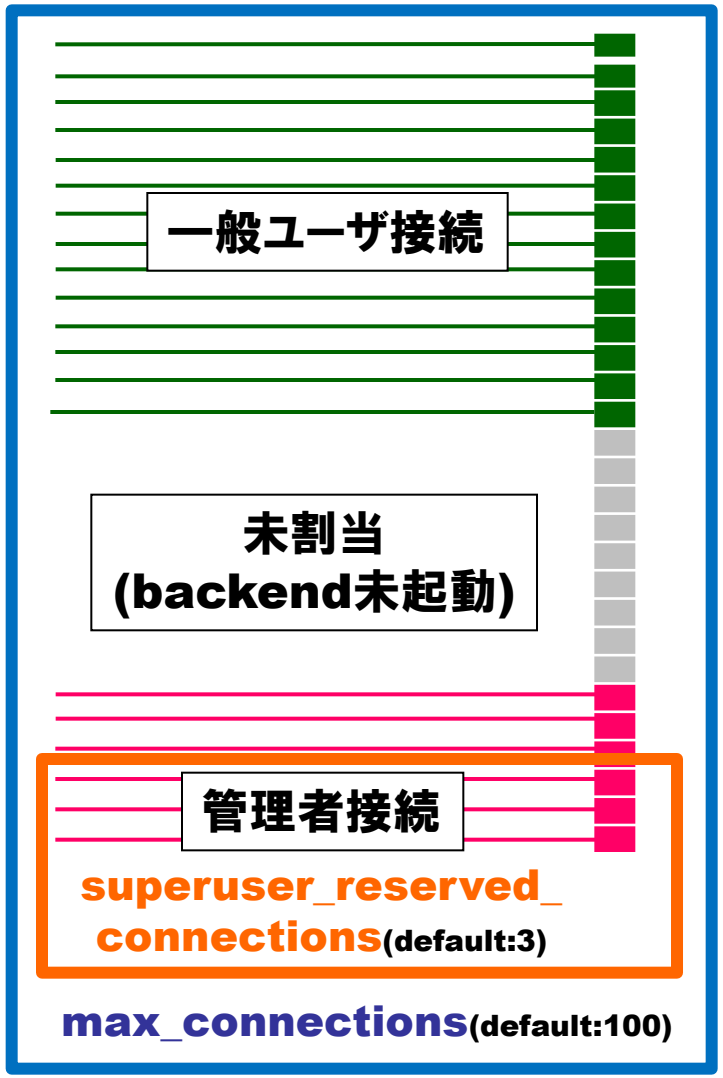
○トピックを詳細に掘り下げているため、運用全体を網羅的に触れていない



Part 1
接続数と安定運用

1-1. 接続数の考え方(1/5)

max_connectionsとsuperuser_reserved_connectionsの仕様をはっきりさせる



挙動を確認

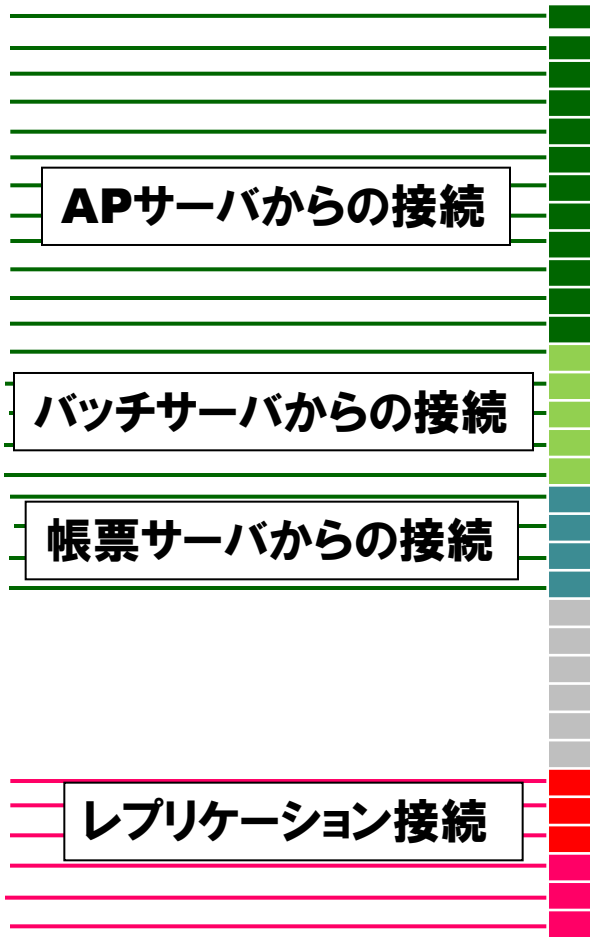
- 総接続数:　最大100接続まで
- 一般ユーザ:最大97接続まで
- 管理者:　最大100 接続まで
- 管理者:　最低3接続は確実に接続可能

運用時チェック

- ☑ APは一般ユーザで接続している
- ☑ 管理者の最大同時接続数が明確だ
- ☑ 一般ユーザの最大同時利用数が明確だ
- ☑ superuser_r.. に余裕がある
- ☑ max_connectionsに余裕がある

1-2. 接続数の考え方(2/5)

システムの挙動を把握して、PGを使用するプロセスを完全に列挙する

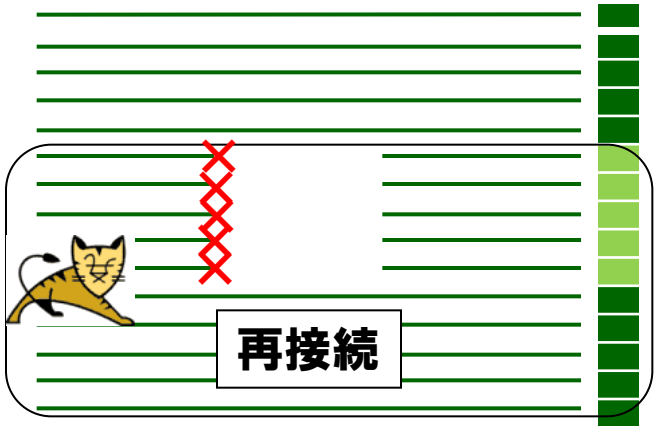


運用時チェック

- ☑ 下記の接続数を考慮に入れた
  - ☑ APからの接続数
  - ☑ バッチ/帳票/ジョブ管理サーバからの接続数
  - ☑ cronで呼び出されるジョブ
  - ☑ DBLINKによる接続
  - ☑ レプリケーションによる接続
  - ☑ HAソフトの死活監視 [管理者]
  - ☑ 情報取得用接続 [管理者]
  - ☑ 監視M/Wからの接続 [管理者]
  - ☑ 緊急メンテ用接続 [管理者]

1-3. 接続数の考え方(3/5)

異常時における再接続に関連する挙動をしっかり把握する



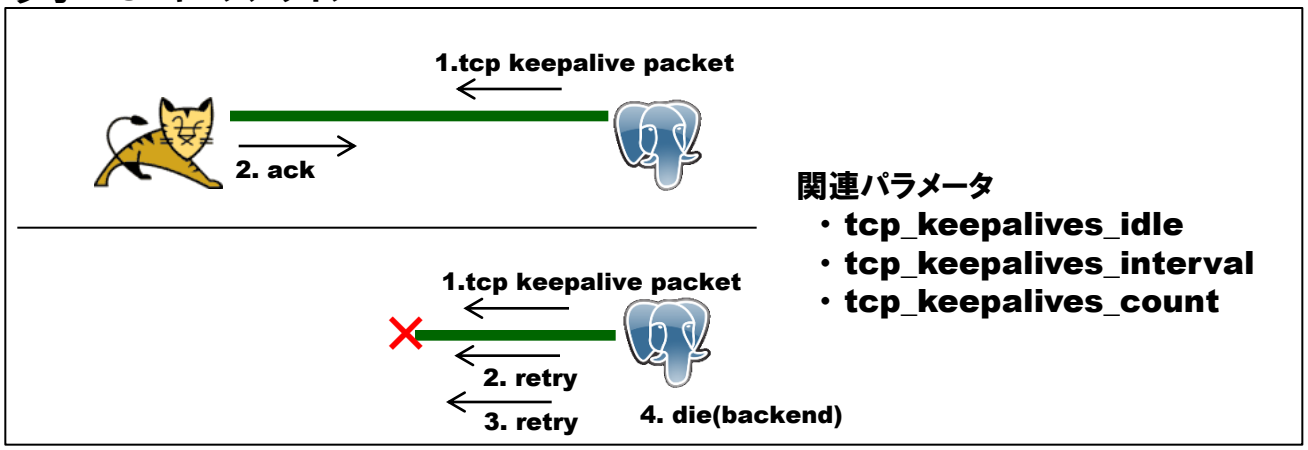
挙動を確認

- N/W断をPostgreSQL側で気づきにくい
- 気づくまでbackendが残存する

運用時チェック

- ☑ AP再接続に耐えうる接続数の余裕がある
- ☑ PG側で早期に検知する設定をしている

参考: TCPキープアライブ



1-4. 接続数の考え方(4/5)

常識的な同時接続数を知って、現行設定を見返す

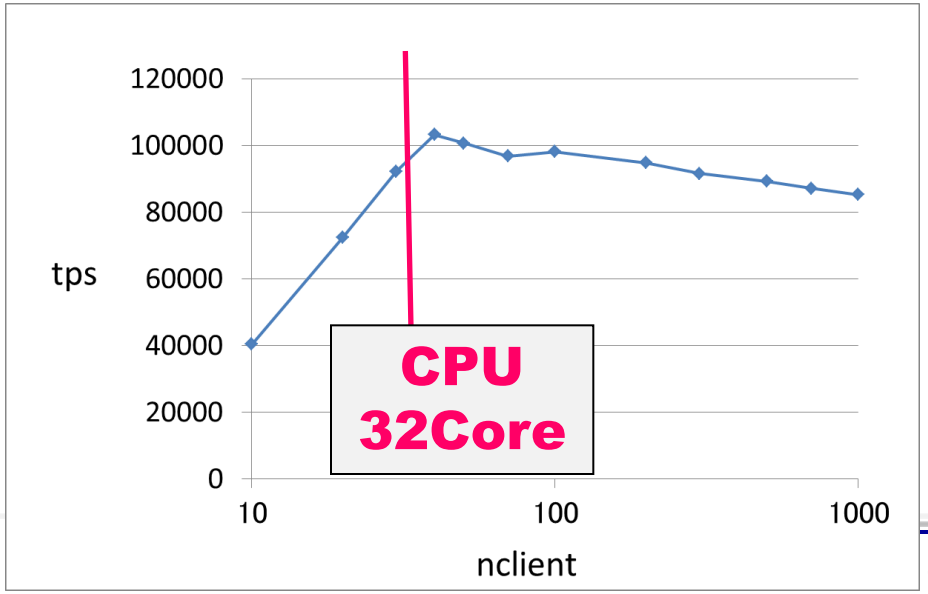
挙動を確認

- 多すぎるとスループットが落ちる
- 少なすぎるとレスポンスが安定しない

運用時チェック

☑ 概ね以下の目安を守っている

- ☑ 同時接続数 ～ 数百
- ☑ 同時実行数 ～ 数十



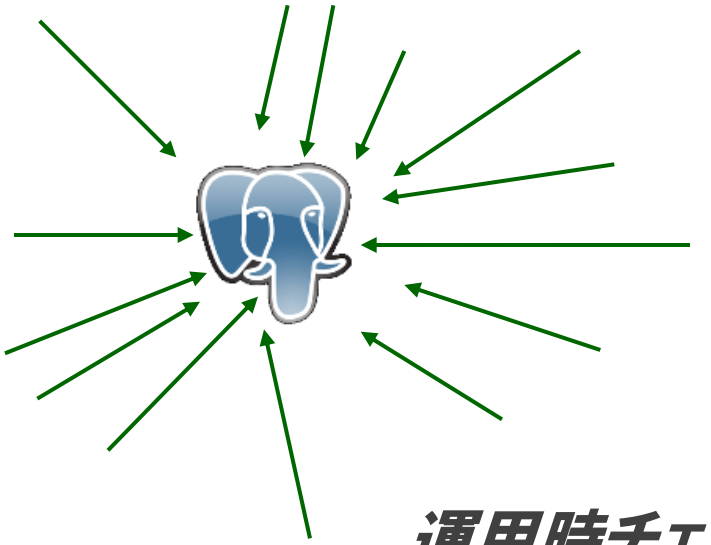
例: CPUネックのとき

同時実行数～ CPUコア数 × α(2～5)
くらいが目安

測定条件: pgbench –S –s 10 –T 600

1-5. 接続数の考え方(5/5)

同時接続数を制御できないときは、そのリスクをしっかり把握する

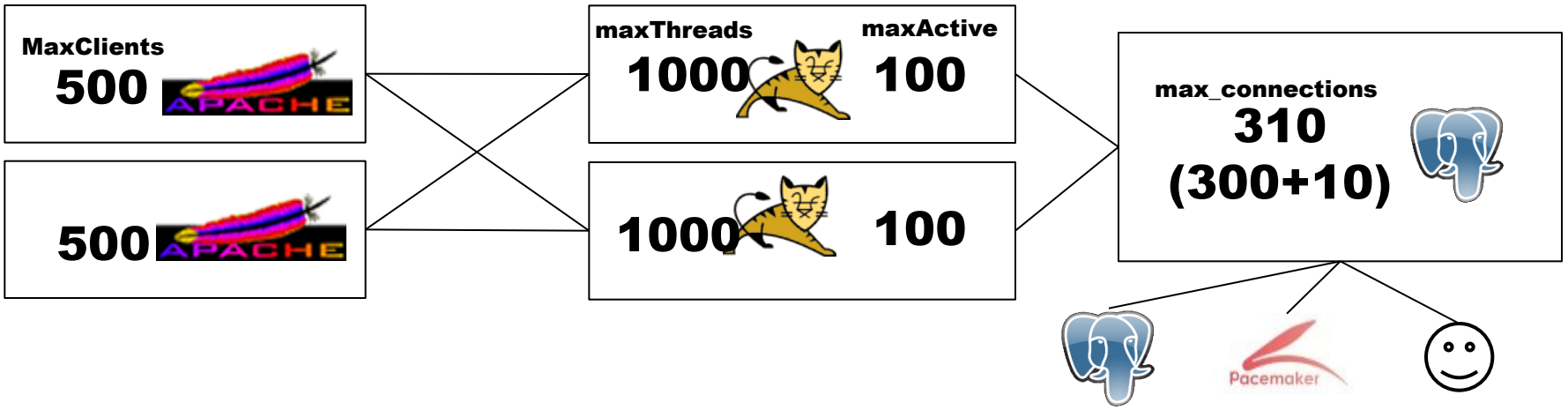


運用時チェック

- ☑ 接続数超過を想定しAP側でリトライ処理を実装した
- ☑ 接続エラーが生じうる負荷量をあらかじめ見積もった
- ☑ チェックポイント等の平滑化設計をした
- ☑ (万一の確率で)確保しているはずの管理者接続が一時的に確保できなくなりうる可能性を頭の片隅に入れた(後述)

1-6. 接続数の一つの理想例

一見して当たり前の設定でも、多くの検討項目があったことを知る



PostgreSQL設定項目	設定値
max_connections	310（接続）
superuser_reserved_connections	10（接続）
tcp_keepalives_idle	60（秒）
tcp_keepalives_interval	5（秒）
tcp_keepalives_count	5（秒）

1-7. 接続数設計にまつわる失敗例

穴があれば、すべからく落ちる

	事象	詳細
1	サービス停止 （フェイルオーバ）	APからの接続をすべてpostgresユーザで実施。特に接続数制限をかけていないため、負荷集中時にmax_connectionsを超過。このタイミングで HAミドルの死活監視 がエラーとなり強制断された。
2	性能劣化	max_connectionsをギリギリに取っていたために、N/W瞬断にともなうTomcatのコネクションプーリング再接続のうち一部が失敗し、部分的な性能劣化を引き起こした。
3	監視異常	backend数がmax_connections以下であることを監視していたが、backend数の大幅な超過を検知した。認証前であってもbackendは起動可能なためである。原理的に無制限にプロセスが立ち上がる可能性がある。

1-7. 落穂拾い

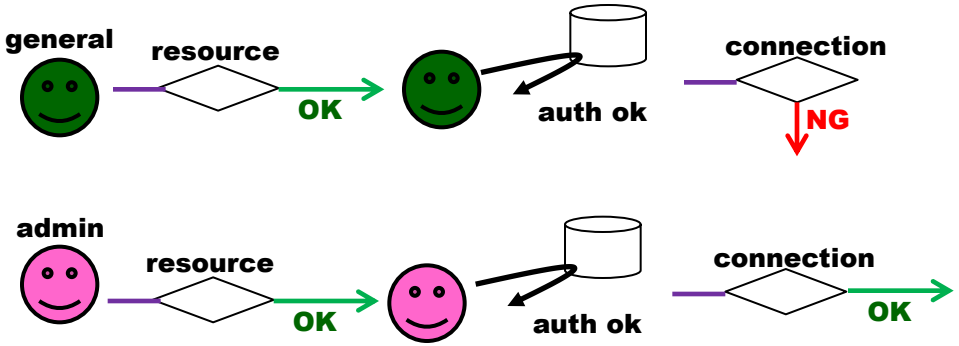
Q. ALTER DATABASE ... CONNECTION LIMIT は？

A. 必要に応じて。設計が煩雑になり、また設定箇所がばらけるため、設定時は文書化が必要。

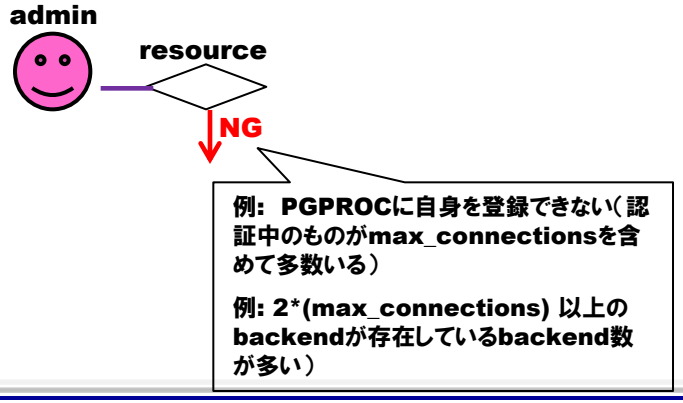
Q. 管理者用接続が確保できないことがある？

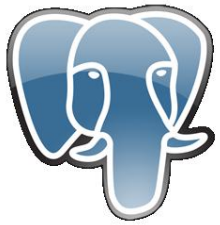
A. 非常にごくまれに。(一部の)最大接続数超過の判断が、認証よりも前に行われるため、最大接続数を大幅に超過してしまっている状況では、管理者でも接続を拒否される可能性がある。

典型的な接続数超過状況



接続数「超」超過状況



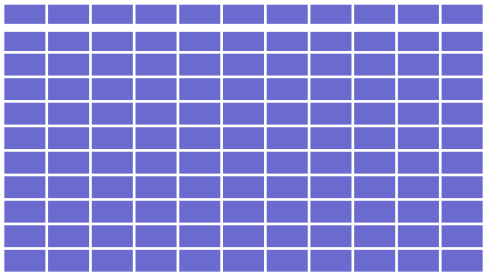


Part 2
メモリ設定と安定運用

2-1. メモリ設定の考え方(1/7)

PGが主に使う2種のメモリ領域と、密接に関連するファイルキャッシュを理解する

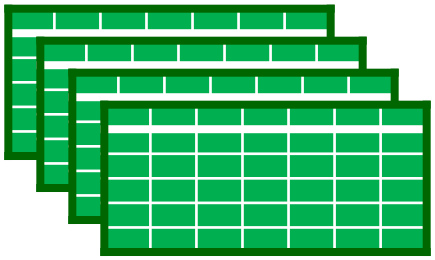
shared_buffers



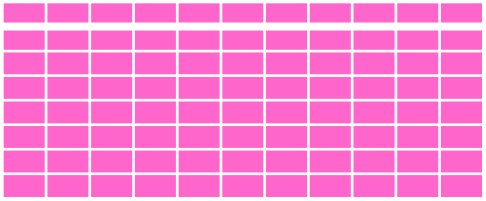
最大で(work_mem+α)×N

α: ～1MB

N: max_connections



File buffer/cache (Linux)



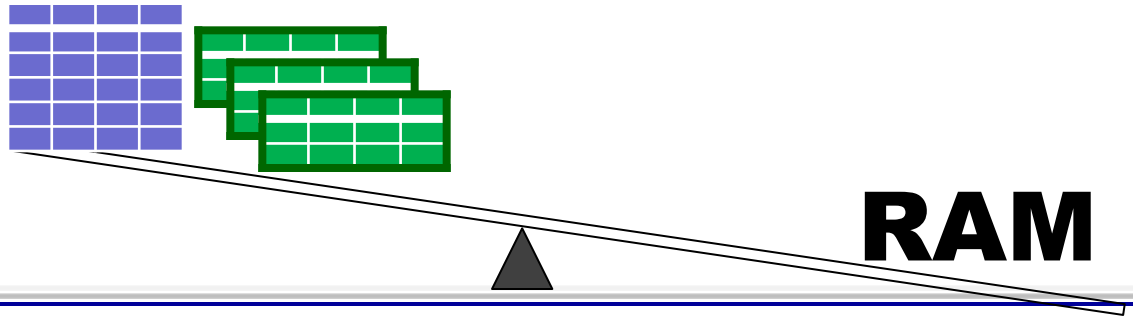
free コマンドで確認可能

挙動を確認

- shared_buffersは起動時にまとめて確保
- backendプロセスは接続時に動的に生成
- work_mem は必要に応じて動的に確保
- backendプロセス自体のメモリも存在

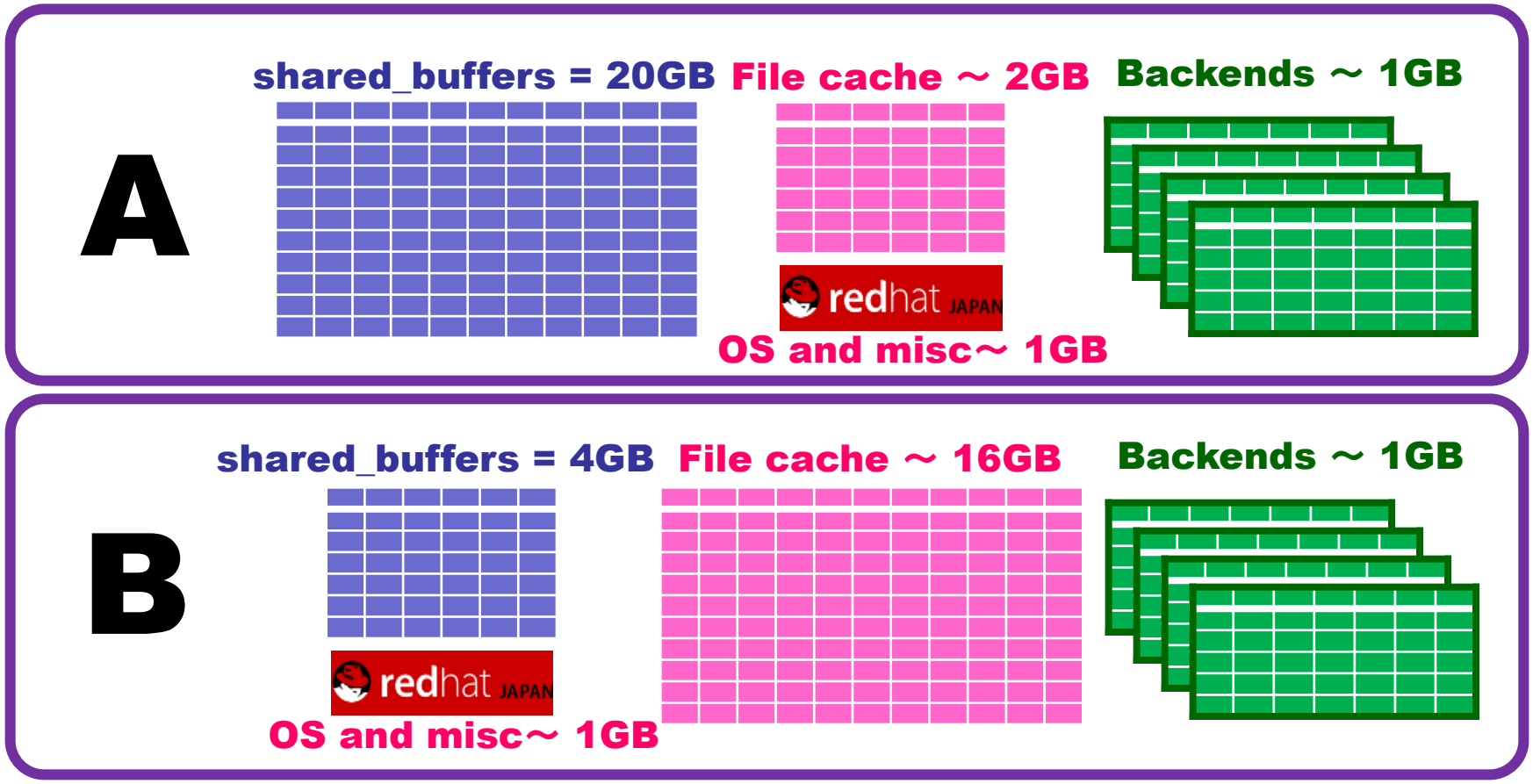
運用時チェック

- ☑ スワップが起きないことを確かめた
  - ☑ (RAM量)＞(PGメモリ最大使用量)+(OSメモリ使用量)
  - ☑ 繁忙時にsi/so がない. freeコマンドで buffer/cache残



2-2. メモリ設定の考え方(2/7)

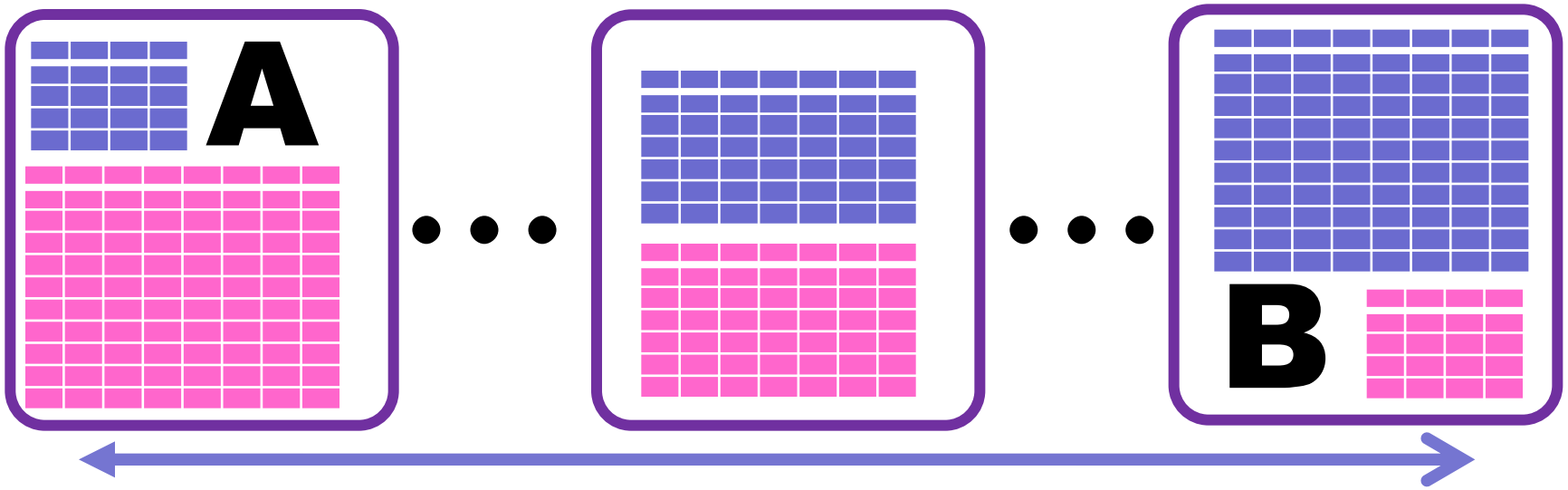
搭載物理メモリ24GBだった。これをどう割り当てるか？それが問題だ。



（総DBサイズ）～ 20GB：Type Aを選択
（総DBサイズ）～ 数百GB：Type Bを選択

2-3. メモリ設定の考え方(3/7)

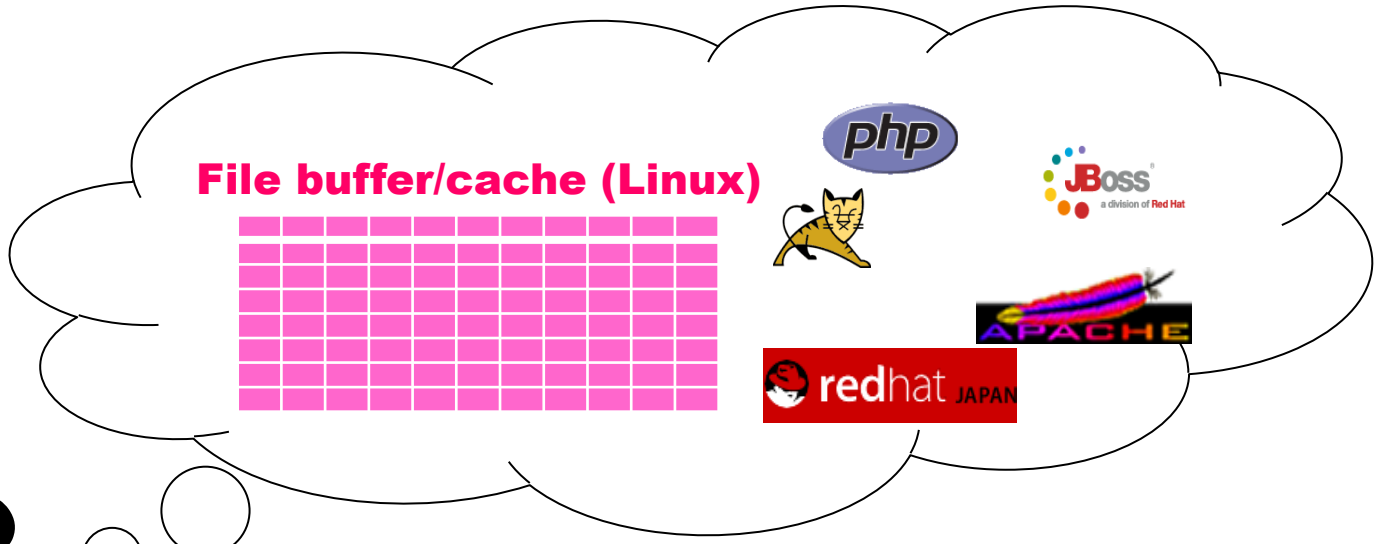
一長一短。決定打はないものの…目安はある(前ページ参照)



メモリ効率(重複)	○ ・・・・・△・・・・・×・・・・・△・・・・・・・○
スループット	×?(やや低下) ・・・・・・△・・・・・・・・・・・・・○?(高)
応答時間のばらつき	○(安定) ・・・・・・・・・△・・・・・・・・・・・・・×?(ばらつく)
設計の柔軟度	○(柔軟) ・・・・・・・・・△・・・・・・・・・・・・・△?(きめうち)

2-4. メモリ設定の考え方(4/7)

PGに適切なメモリのヒントを与える



運用時チェック

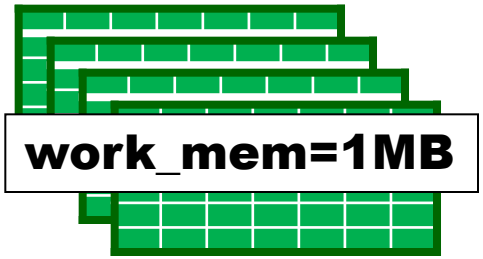
- ☑ 適切な effective_cache_sizeを与えている
  - ☑ DBサーバに同居している他のM/Wの使用メモリを考慮した
  - ☑ 実際の値(cache/buffers)と大きな乖離がないことを確認した
- ☑ H/W特性を生かした設定をしている(オプション)
  - ☑ SSDドライブなのでrandom_page_costを小さくした

2-5. メモリ設定の考え方(5/7)

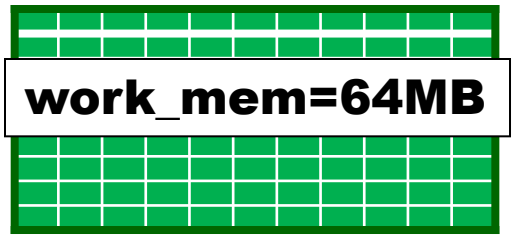
work_memを柔軟に運用すると、設計・運用に幅がでる

よい例

オンライン系TX

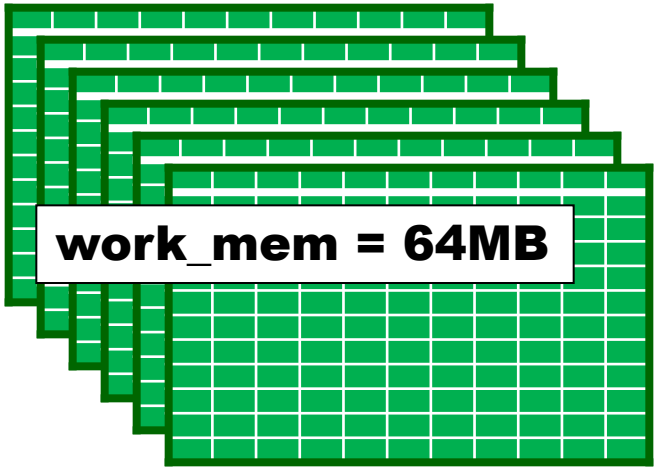


バッチ系TX



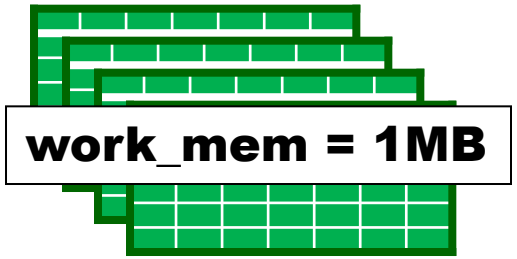

```
BEGIN;
SET LOCAL work_mem  to '64MB';
-- SQLs that needs memory;
COMMIT;
```

悪い例: 無駄が多すぎ



work_memは必要に応じて確保されるので、全接続が64MB使い切ることはないかもしれないが、メモリオーバーコミットの設定を容認できないため(2-1参照) 設計が窮屈になる。

悪い例？: 改善の余地あり



特定のクエリ(特にバッチ系トランザクション)に関しては、work_memを大きくすることで性能向上できるはず(上記例)なのにそれができていない。

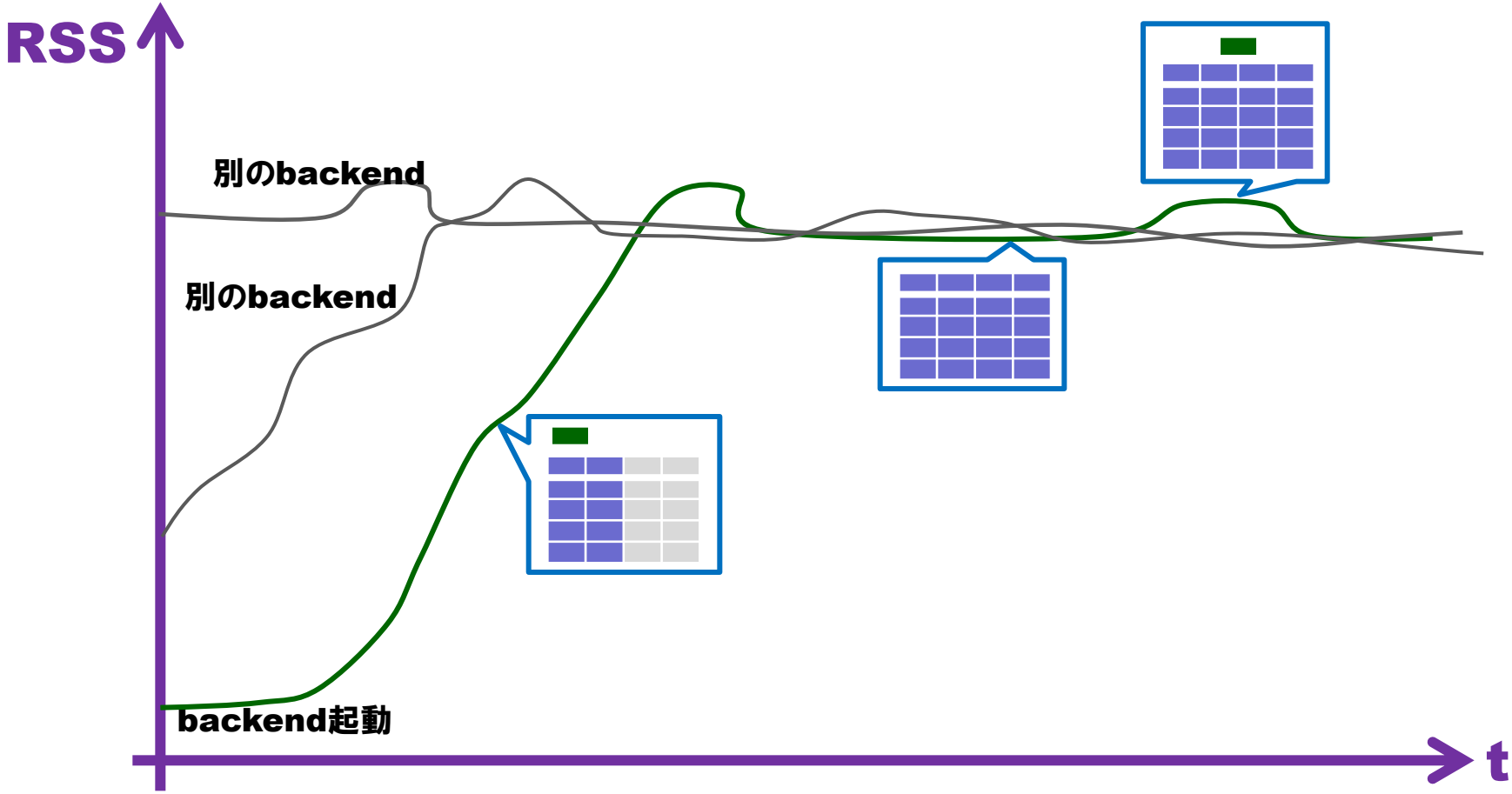
2-6. メモリ設定の考え方(6/7)

そもそも work_memはどんな時に使われるのかを理解する

		work_memが大きくなるとどうなるか？
1	ソート	ディスクを使わずにインメモリでソートできる
2	JOIN	ハッシュがインメモリでできる
3	Bitmapスキャン	Bitmap Indexがメモリに収まると、lossyな BitmapScanにならなくなる
4	集約	HashAggregateが選ばれやすくなる
5	一時表	ディスクアクセスを減らすことができる

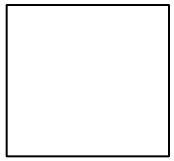
2-7. メモリ設定の考え方(7/7)

実際のところ、総使用メモリを正確に測定するのはあきらめた方がよい

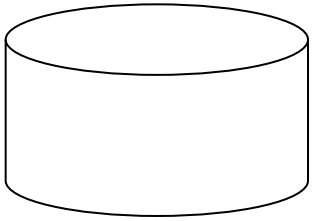


psコマンドの RSS/VSZ はあまり参考にならないことが多い。
見るべきは vmstat/sar の si/so や cache/buffers

2-8. メモリ設定の一つの理想例

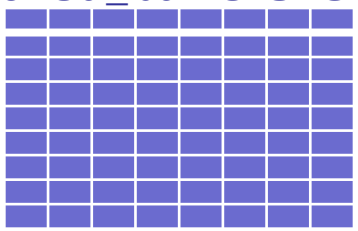


RAM: 32GB

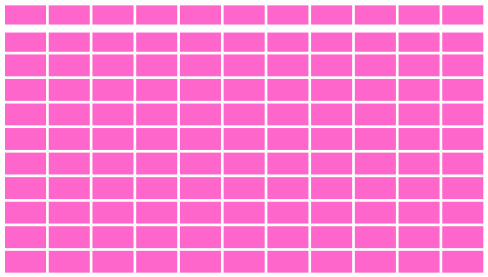


DB: 300GB

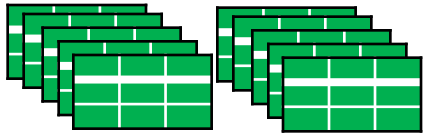
shared_buffers=8GB



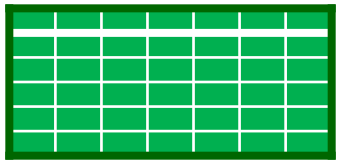
effective_cache_size=20GB

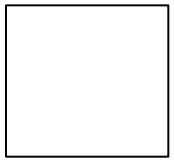


work_mem=1MB
max_connections=300

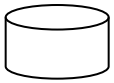


work_mem = 64MB
(業務Aのみ・動的に設定）



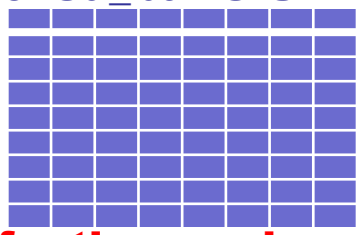


RAM: 32GB

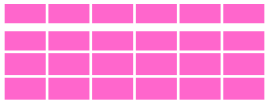


DB: 20GB

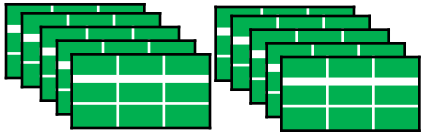
shared_buffers=20GB



effective_cache_size=10GB



work_mem=1MB
max_connections=300



2-9. メモリ設定にまつわる失敗例

運用を重ねるうちにいつのまにかはまることも…

1	遅い！	shared_buffers、effective_cache_sizeはデフォルトで小さすぎる。デフォルト設定だと本来の実力の半分以下のスループットしかでない。
2	特定のSQLが遅くなったり早くなったりプランが安定しない	その業務に必要な work_mem が十分でなく、実行計画が安定しない。
3	負荷集中時メモリ不足	work_memを大きく設定しすぎており、同時実行数を上げるとメモリを大量に消費するようになりスワップが発生した。

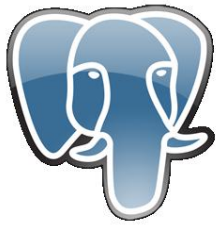
2-10. 落穂ひろい

Q. maintainance_work_mem は？

A. VACUUM などの運用系SQL文を発行するときはこのパラメータが使われる。この分も一応考慮する（一般的には前項までの設計をしていれば、十分バッファが取れているはず）。自動VACUUMは、workerがそれぞれメモリを食うので気を付ける。

Q. 共有メモリに確保されるのは shared_buffers だけ？

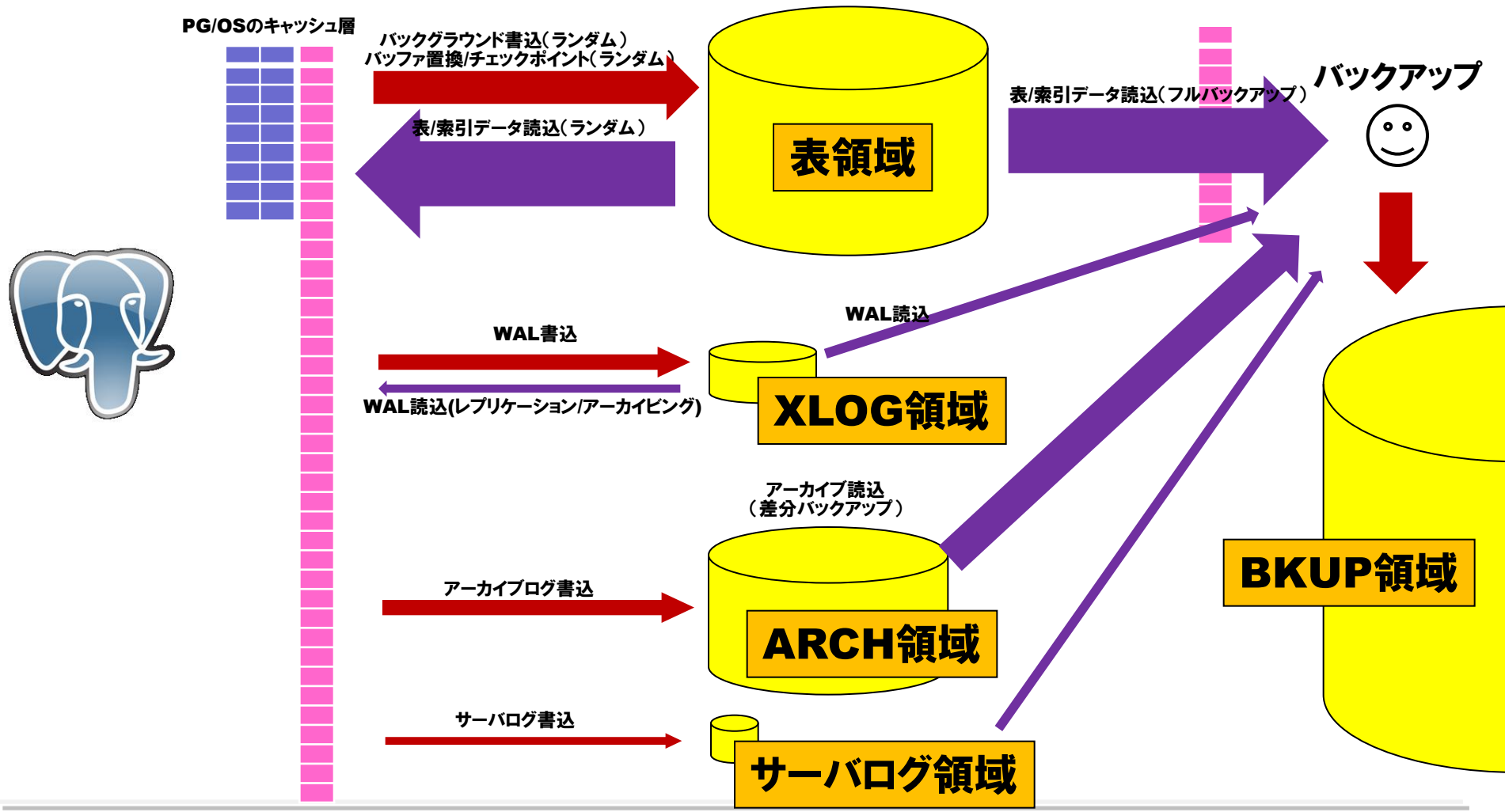
A. そんなことはない。各種ロック管理、プロセス間通信や、状態管理などのための領域がある。容量は大したことがないけれど。



Part 3
ディスク設定と安定運用

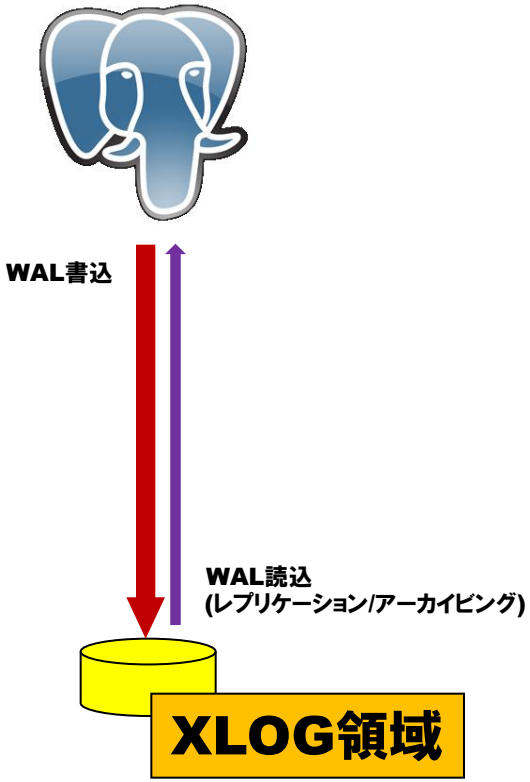
3-1. ディスク設計の考え方(1/7)

DBサーバのI/Oを理解する。考慮するべき対象を明確化する。



3-2. ディスク設計の考え方(2/7)

XLOG領域には最も高いレベルの安全性が求められる

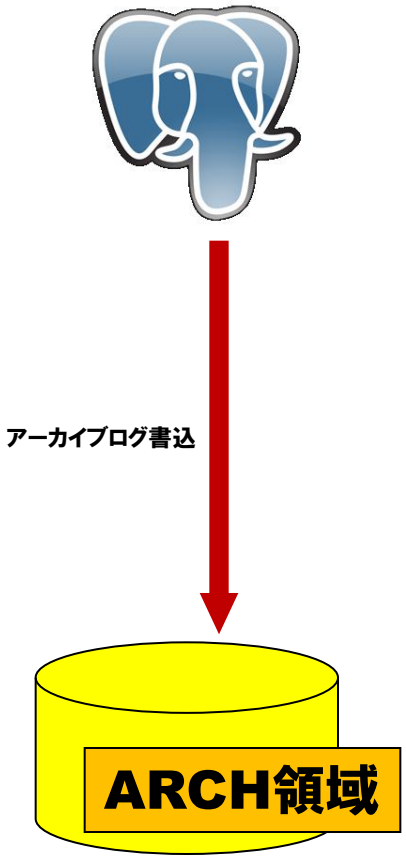


運用時チェック

- ☑ 物理的に安全性が高いことを確認した
  - ☑ RAID(1+0)が組まれている
  - ☑ ホットスペアがついている
  - ☑ バッテリーキャッシュがついている
  - ☑ ディスク故障＋リビルド実施をすぐに検知できる
- ☑ 論理的に安全性が高いことを確認した
  - ☑ 別パーティションとして切り出されている
  - ☑ (システム休止時などのタイミングで)fsckを実施している
  - ☑ wal_sync_methodがfsync(fdatasync)である
  - ☑ full_page_writes/synchronous_commitが設定されている
- ☑ 十分な容量があることを確認した
  - ☑ 一定時間アーカイブ失敗しても問題ない容量を有している
- ☑ 十分な応答性能があることを確認した
  - ☑ writeback設定がなされている(バッテリーキャッシュ必須！)

3-3. ディスク設計の考え方(3/7)

ARCH領域には高い独立性・拡張性が求められる



運用時チェック

- ☑ 独立性が高いことを確認した
  - ☑ NFSマウントされた領域である(最善)
  - ☑ 書込み先を複数登録している(次善)
  - ☑ 表領域とは別に切り出している(最低限)

- ☑ 容量に十分余裕があることを確認した
  - ☑ 必要に応じて容量を増加させられるH/W構成である
  - ☑ 設計時に1日当たりの量を見積もるのは難しいことを理解した
  - ☑ 今後の1日当たりの量を見積もるのは難しいことを理解した

3-4. ディスク設計の考え方(4/7)

ARCH領域とXLOG領域には特徴的なI/O特性がある

Q. ARCH領域の await が高い。ボトルネックなのか？

A. 違う。awaitが高いのは、アーカイブ対象をcpなどでまとめてコピーするため。

Q. でもアーカイブやめたら性能があがった。やっぱりボトルネックでは？

A. アーカイブにともなって表データが載っているOSのバッファを飛ばしたから。

Q. XLOG領域の書込量がARCH領域への書込量よりかなり多い。何故？

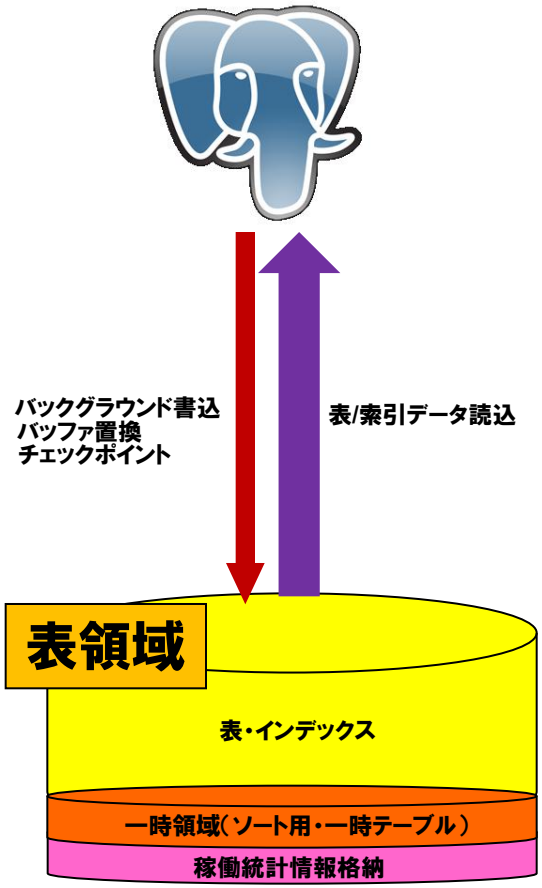
A. XLOG領域へはこまごまと書き込みをするので、どうしてもARCH領域よりも書き込み量が多くなる。

Q. XLOG領域の読込量がほぼゼロだが、どのようにアーカイブしている？

A. OSのバッファに残っており、実I/Oが発生しない。

3-5. ディスク設計の考え方(5/7)

表領域にはランダムI/O性能が求められる。容量にも注意。



挙動を確認

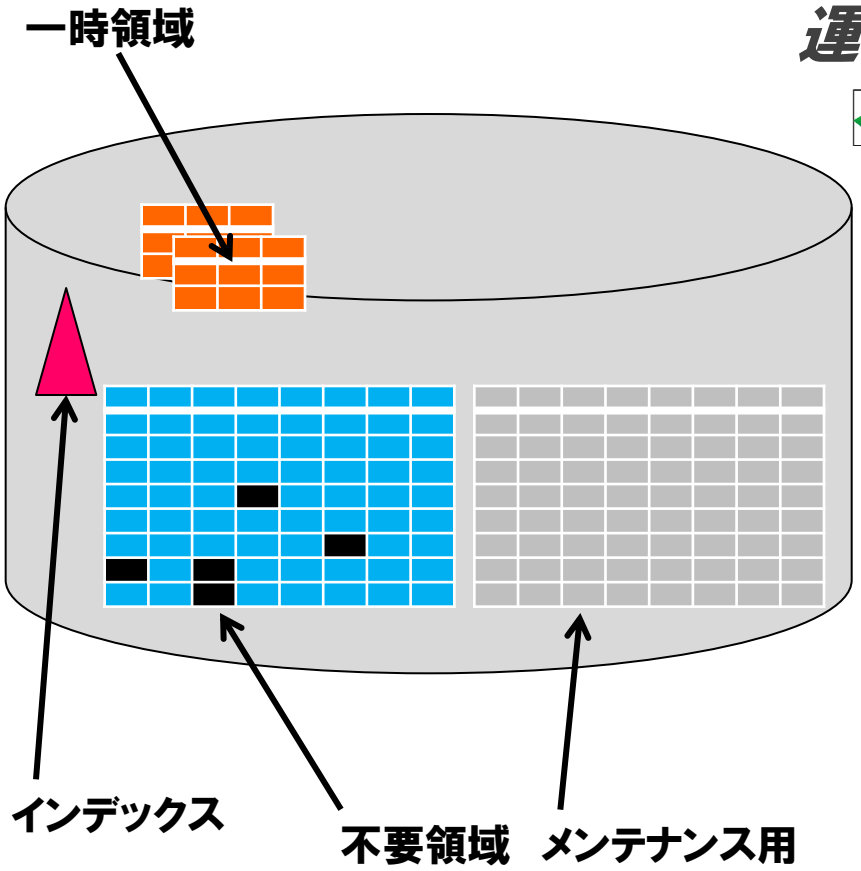
- 一時領域や統計情報格納領域を含む(分離可)
- 基本的にランダムI/Oが支配的
- 一般的にディスクネックになるならココ

運用時チェック

- ☑ H/W・運用要件にあった形になっている
  - ☑ ストレージスナップショットの上限を意識している
  - ☑ fsckしても問題ない時間で終わる
  - ☑ 運用コストは高くない
- ☑ 容量に十分余裕がある
  - ☑ ストレージ容量に余裕がある(後述)
  - ☑ 容量を追加に耐えうる構造である(例:パーティショニング)

3-6. ディスク設計の考え方(6/7)

表領域における容量見積もりの勘所をつかむ

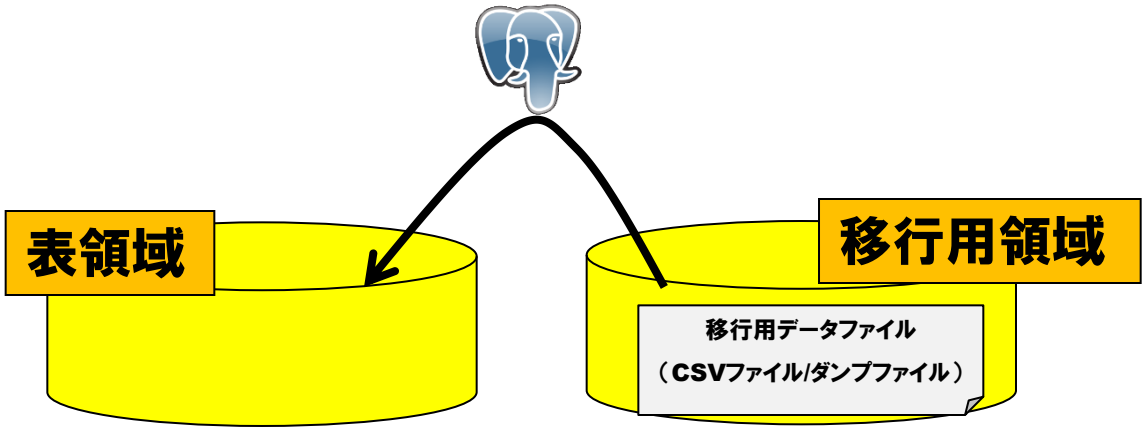


運用時チェック

- ☑ **容量設計時に以下を考慮した**
  - ☑ 論理レコード⇒物理レコードのオーバーヘッド
  - ☑ インデックスサイズ
  - ☑ 不要領域分
  - ☑ メンテナンスに必要な一時領域
  - ☑ 一時テーブル領域
  - ☑ ソート等で利用する一時領域
  - ☑ 想定外のデータ増加にも耐えられるようなバッファ

3-7. ディスク設計の考え方(7/7)

その他、システム運用上必要な容量見積もりの勘所をつかむ

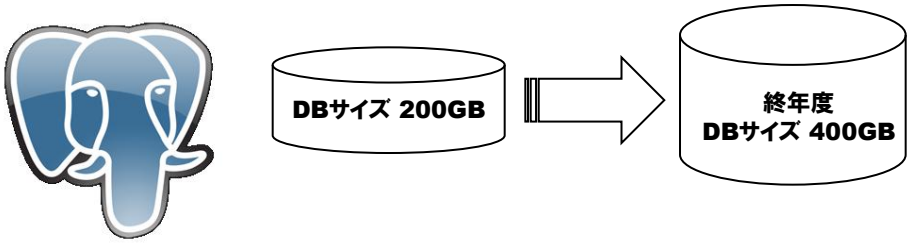


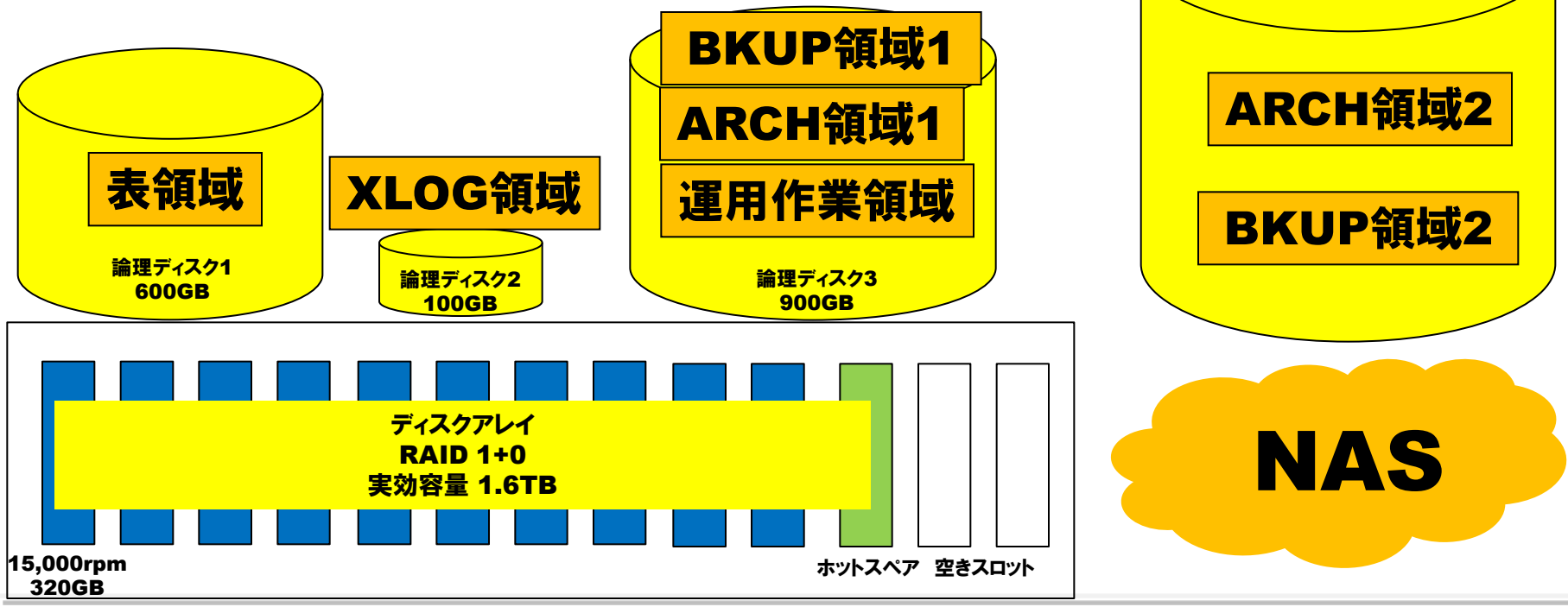
運用時チェック

- ☑ 容量設計時に以下を考慮した
  - ☑ BKUP領域にN+1世代分のフルバックアップ領域がある
  - ☑ 上記世代に対応するアーカイブログ領域がある
  - ☑ 移行データ配置用の領域がある
  - ☑ リカバリデータ配置用の領域がある

3-8. ディスク設計の一つの理想例

正解を見つけるよりも、妥当かどうかの判断が重要視される





3-9. ディスク設計にまつわる失敗例

	失敗例	詳細
1	XLOG領域あふれてサービス停止	NFSマウントが失敗して、アーカイブが失敗し続けていたため
2	メンテナンスできないどうしよう	空き容量がなくなってきたのでVACUUM FULLしようとしたが、そのための一時領域が取れない。
3	めちゃくちゃ遅い	ストレージの設定として writethrough設定がされていた。Writeback設定にする。 注: linuxのmountオプションで writeback設定をしてはいけない。これはバッテリーキャッシュのついたストレージの設定の話。