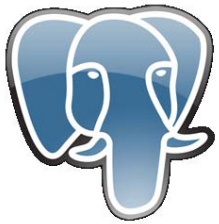


PostgreSQLのチューニング技法
- しくみを知って賢く使う -
PostgreSQL Tuning Technique
- Let's use PostgreSQL effectively, finding out its internals -

PostgreSQL Conference 2009 Japan
2009.11.21

PostgreSQLしくみ分科会
PostgreSQL Mechanism Research Working Group
笠原 辰仁. 坂田 哲夫. 桑村 潤.
Tatsuhito Kasahara. Tetsuo Sakata. Jun Kuwamura.

welcome !

- 今日は、PostgreSQLのパラメータについて、しくみを交えながら解説します
  - 各パラメータの働きのしくみと、設定方針を解説します
  - また、パラメータ依存で引き起こされる問題とそのシューティングについても少し触れていきます
  - どうしてこのパラメータが効くのか？なぜこの設定方針が良いのか？を知っていただき、PostgreSQLのチューニングに役立ててください

agenda

- Background PostgreSQL mechanism
  - SQL parse & planning & execution
  - memory & storage
  - maintenance (VACUUM)
  - checkpoint & WAL
- Preparing for tuning
  - setting for logging & statistics
- Parameters tuning
  - shared memory & checkpoint
    - shared_buffers
    - checkpoint_*
    - bgwriter_*
  - VACUUM
    - max_fsm_*
    - autovacuum_*
    - maintenance_work_mem
  - work_mem
  - wal_buffers
  - plan tuning
    - effective_cache_size
    - default_statistics_target

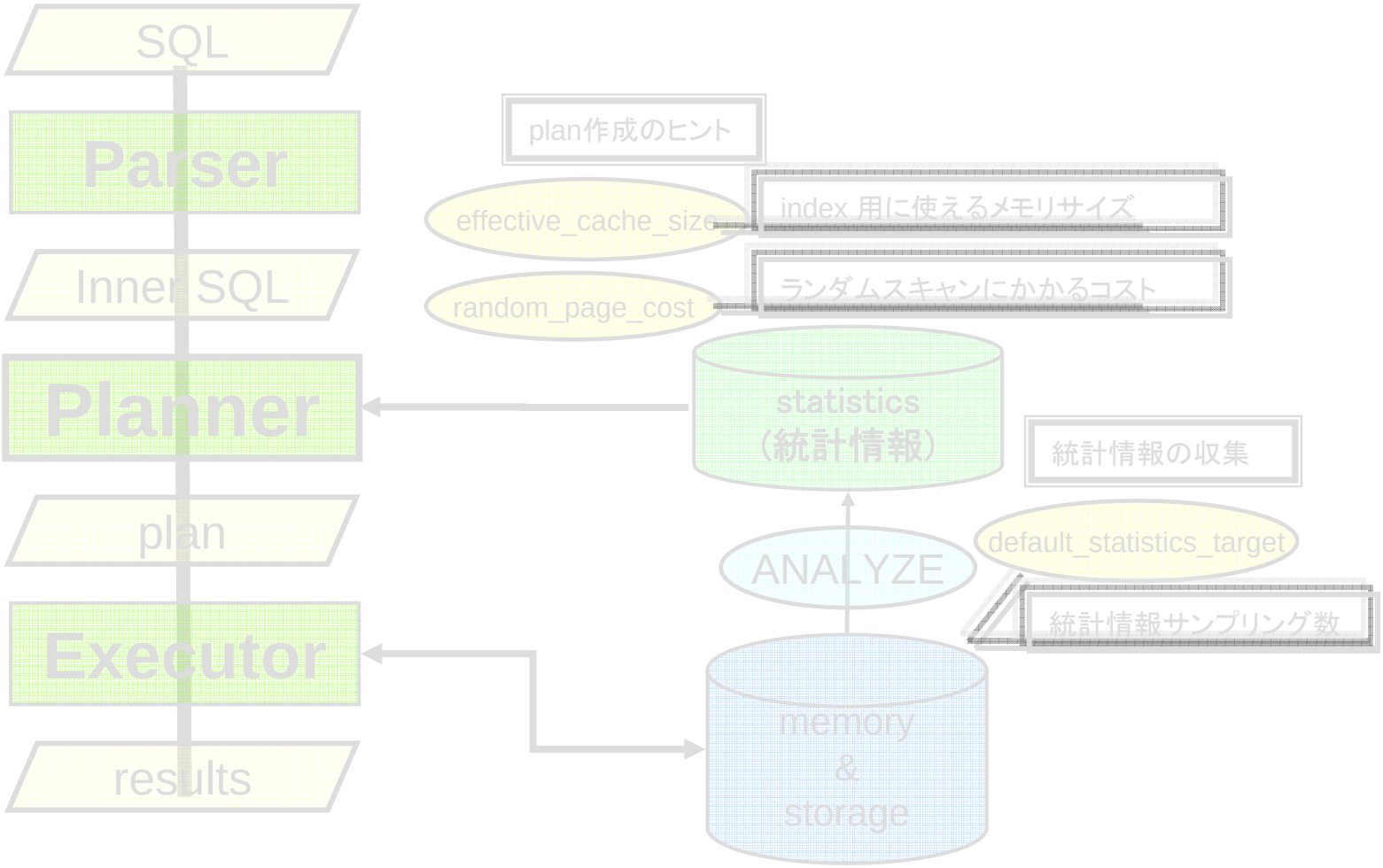
Background: PostgreSQL mechanism

- 解説対象のパラメータに関係するPostgreSQLのしくみを紹介します
  - まずはPostgreSQLのふるまいとパラメータの関係を見てみましょう

- 切り口は以下です
  - SQLが実行されるまで(parse, planning, execute)
  - メモリとストレージ(memory & storage)
  - VACUUM
  - CHECKPOINT と WAL

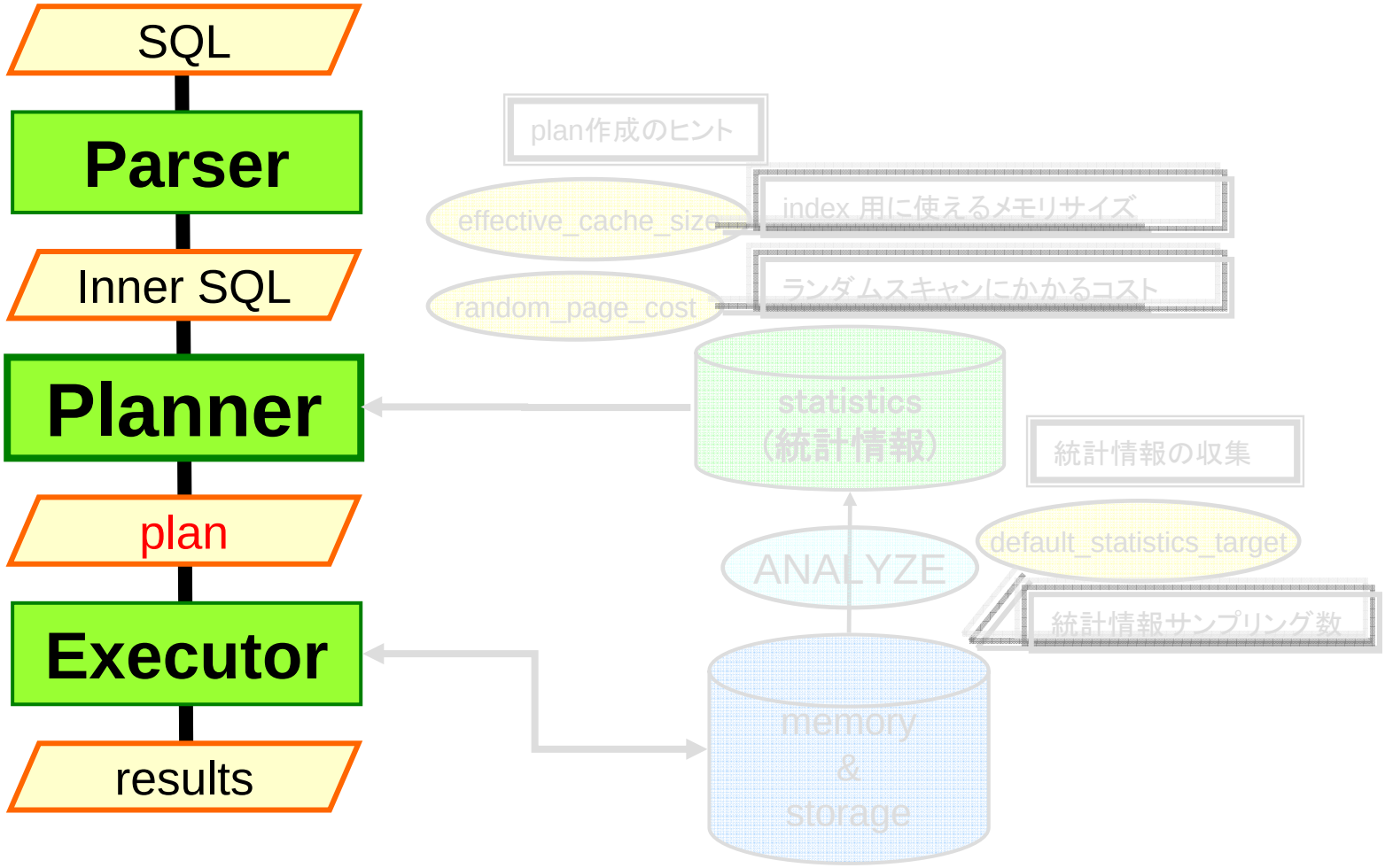
Background: PostgreSQL mechanism(SQLが実行されるまで)

- SQL parse & planning & execution
  - CHECK:適切な統計情報を取得できていますか？
  - CHECK:cost推定のヒントは適切に設定できていますか？



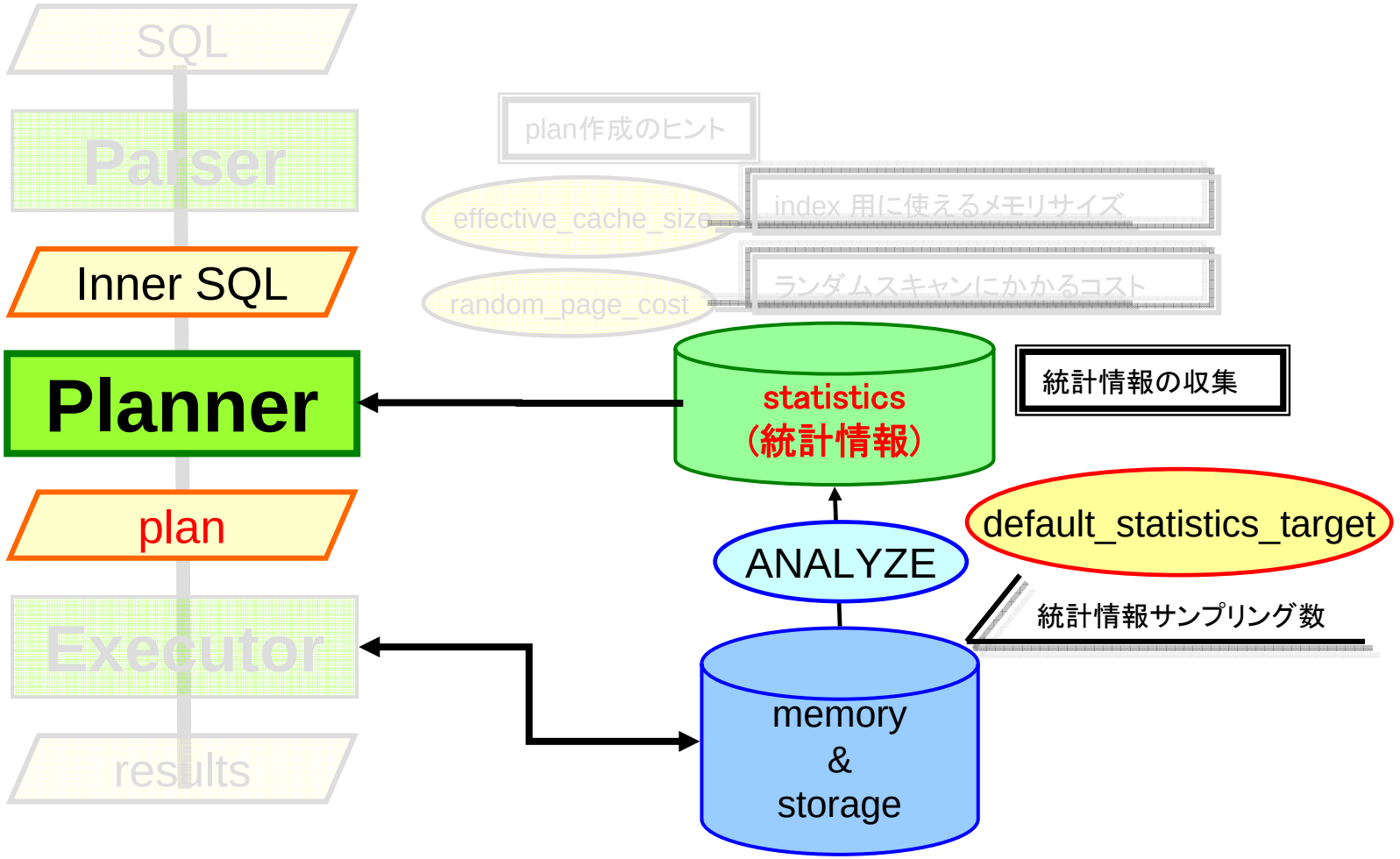
Background: PostgreSQL mechanism(SQLが実行されるまで)

- SQL parse & planning & execution
  - CHECK:適切な統計情報を取得できていますか？
  - CHECK:cost推定のヒントは適切に設定できていますか？



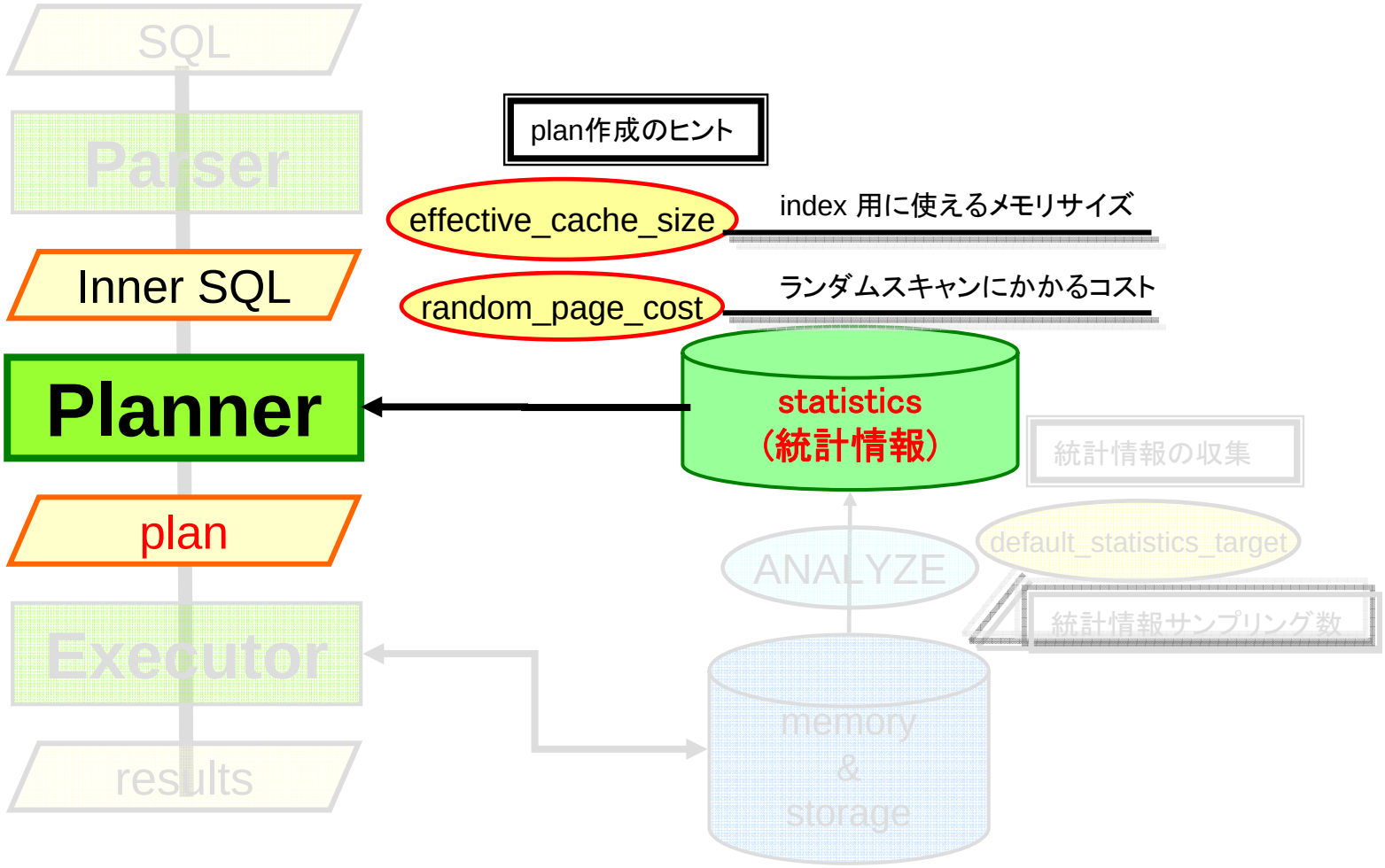
Background: PostgreSQL mechanism(SQLが実行されるまで)

- SQL parse & planning & execution
  - CHECK:適切な統計情報を取得できていますか？
  - CHECK:cost推定のヒントは適切に設定できていますか？



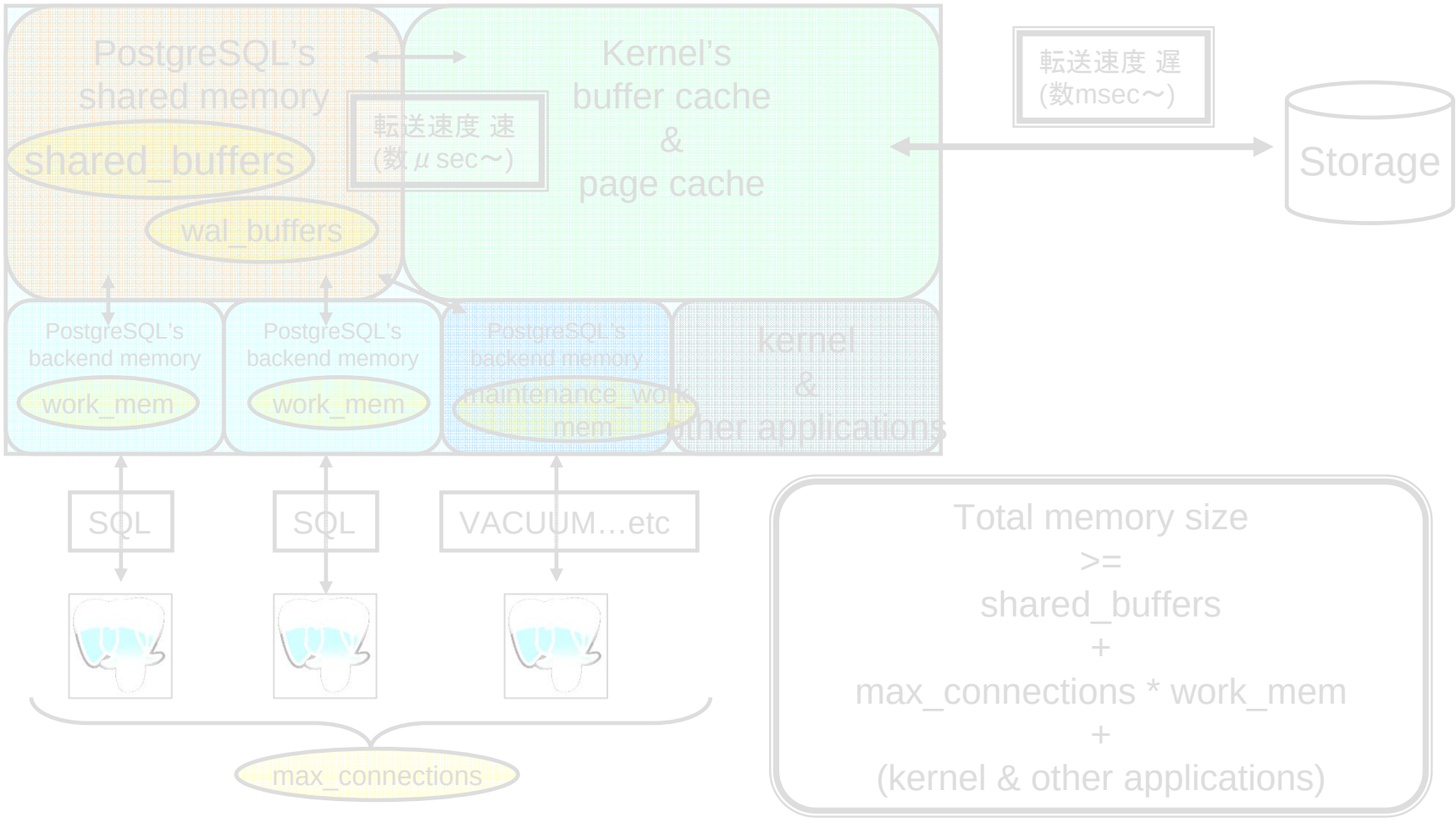
Background: PostgreSQL mechanism(SQLが実行されるまで)

- SQL parse & planning & execution
  - CHECK:適切な統計情報を取得できていますか？
  - CHECK:cost推定のヒントは適切に設定できていますか？



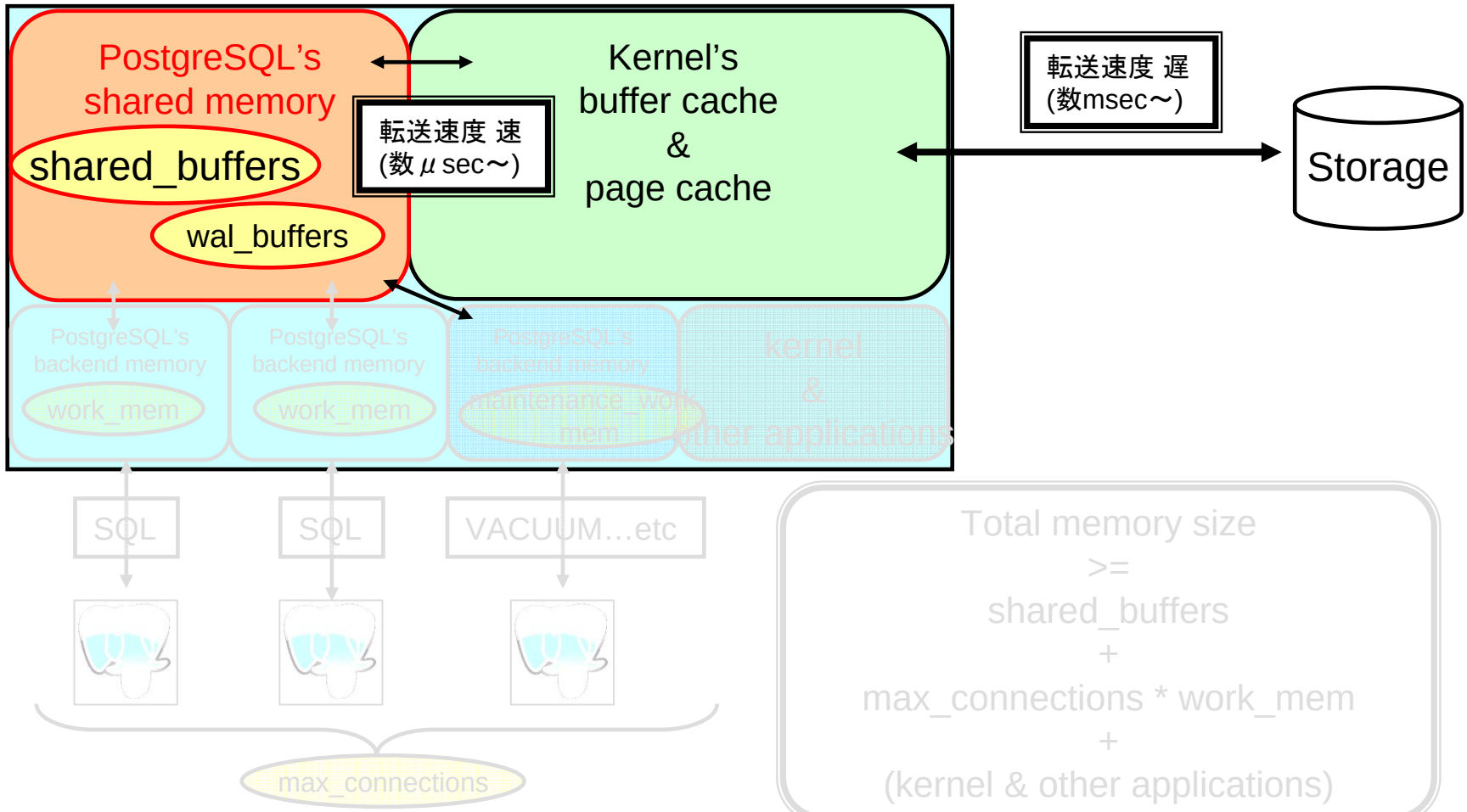
Background: PostgreSQL mechanism(メモリの使われかた)

- memory & storage
  - CHECK:メモリの利用効率は意識していますか？
  - CHECK:メモリ配分は意識していますか？



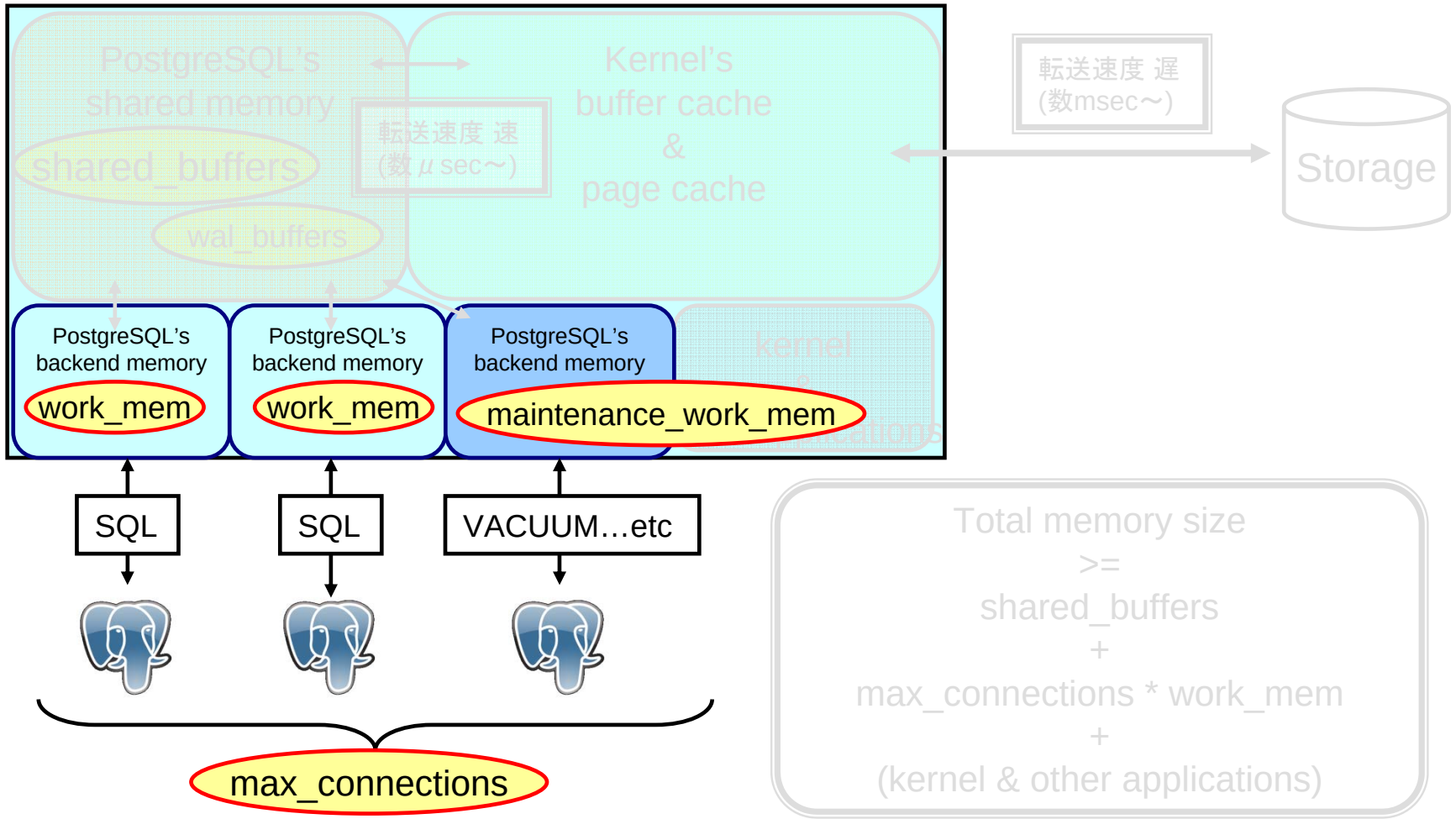
Background: PostgreSQL mechanism(メモリの使われかた)

- memory & storage
  - CHECK:メモリの利用効率は意識していますか？
  - CHECK:メモリ配分は意識していますか？



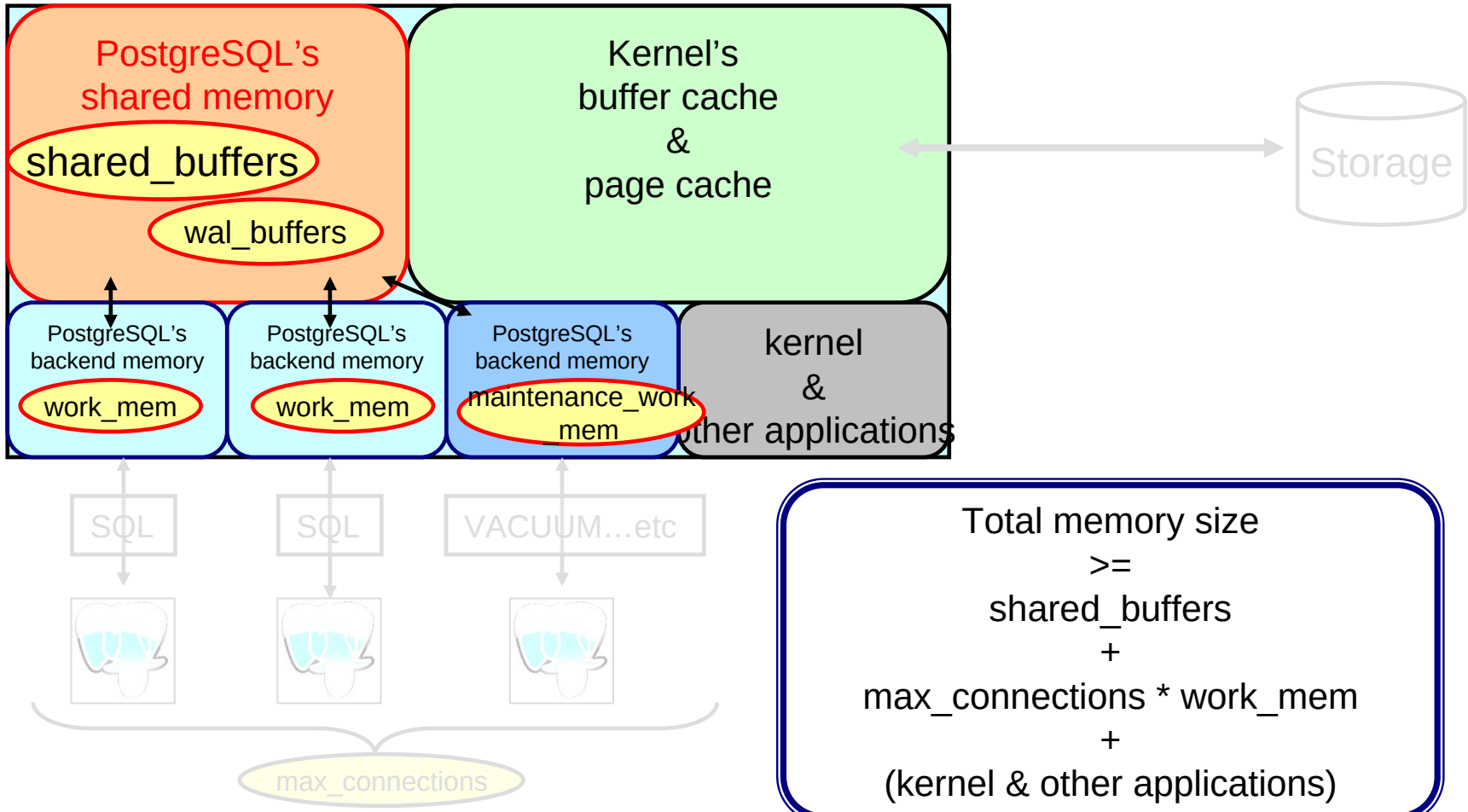
Background: PostgreSQL mechanism(メモリの使われかた)

- memory & storage
  - CHECK:メモリの利用効率は意識していますか？
  - CHECK:メモリ配分は意識していますか？



Background: PostgreSQL mechanism(メモリの使われかた)

- memory & storage
  - CHECK:メモリの利用効率は意識していますか？
  - CHECK:メモリ配分は意識していますか？



Total memory size
>=
shared_buffers
+
max_connections * work_mem
+
(kernel & other applications)

- VACUUM
  - の前に

まだ
VACUUM FULL
を使っていたりしませんか？

PostgreSQLはVACUUM(無印)だけで運用できます。
8.3からは、HOT&autovacuumのおかげでさらに
手動で行うことは少なくなります。

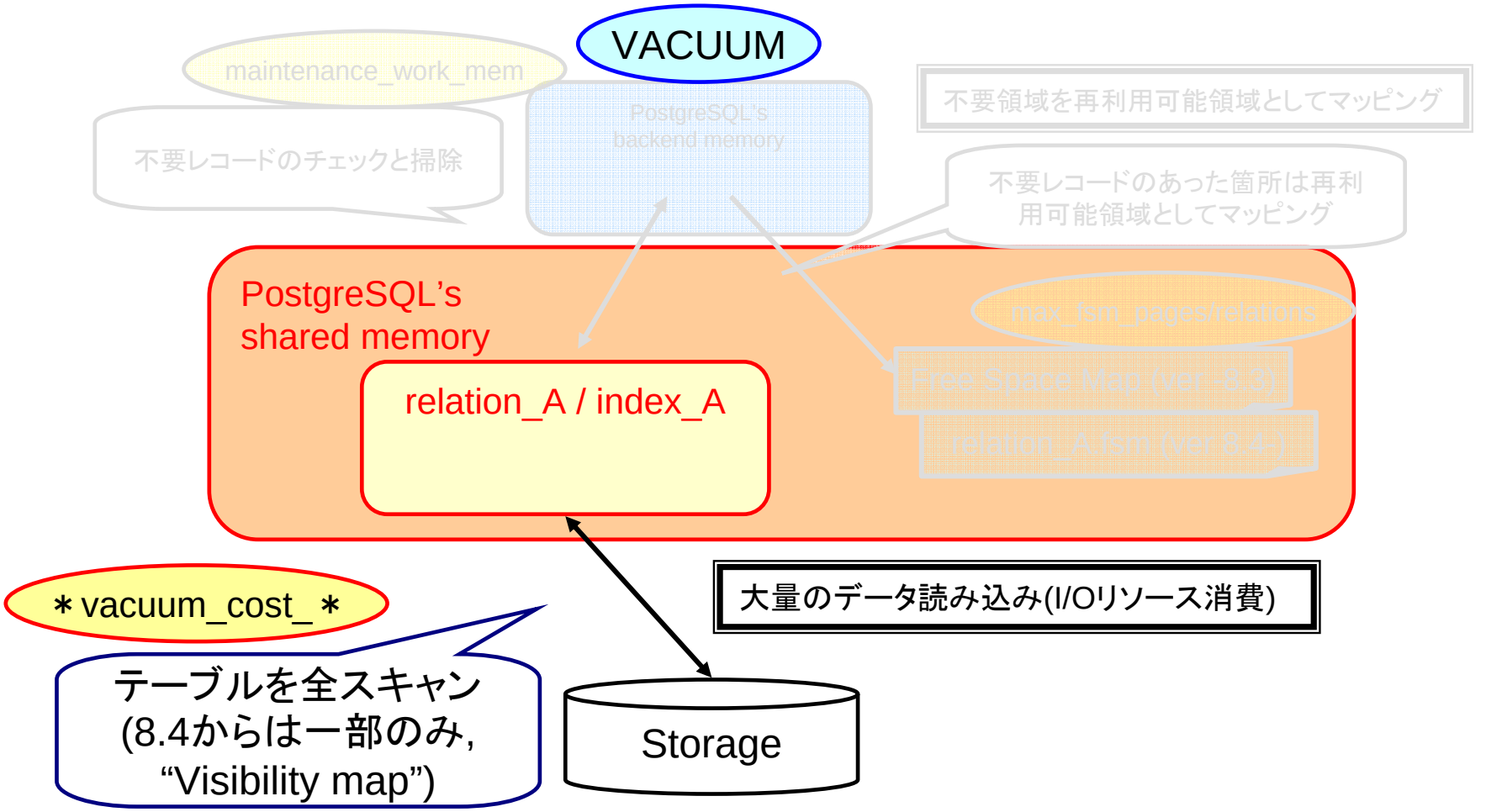
VACUUM FULL が必要になる運用は間違いです！
VACUUM(無印)を上手く使いこなしましょう！

Background: PostgreSQL mechanism(VACUUM処理の流れ)

- Maintenance(VACUUM)
  - CHECK:不要領域を全て回収できていますか？
  - CHECK:VACUUMのI/O負荷を制御できていますか？

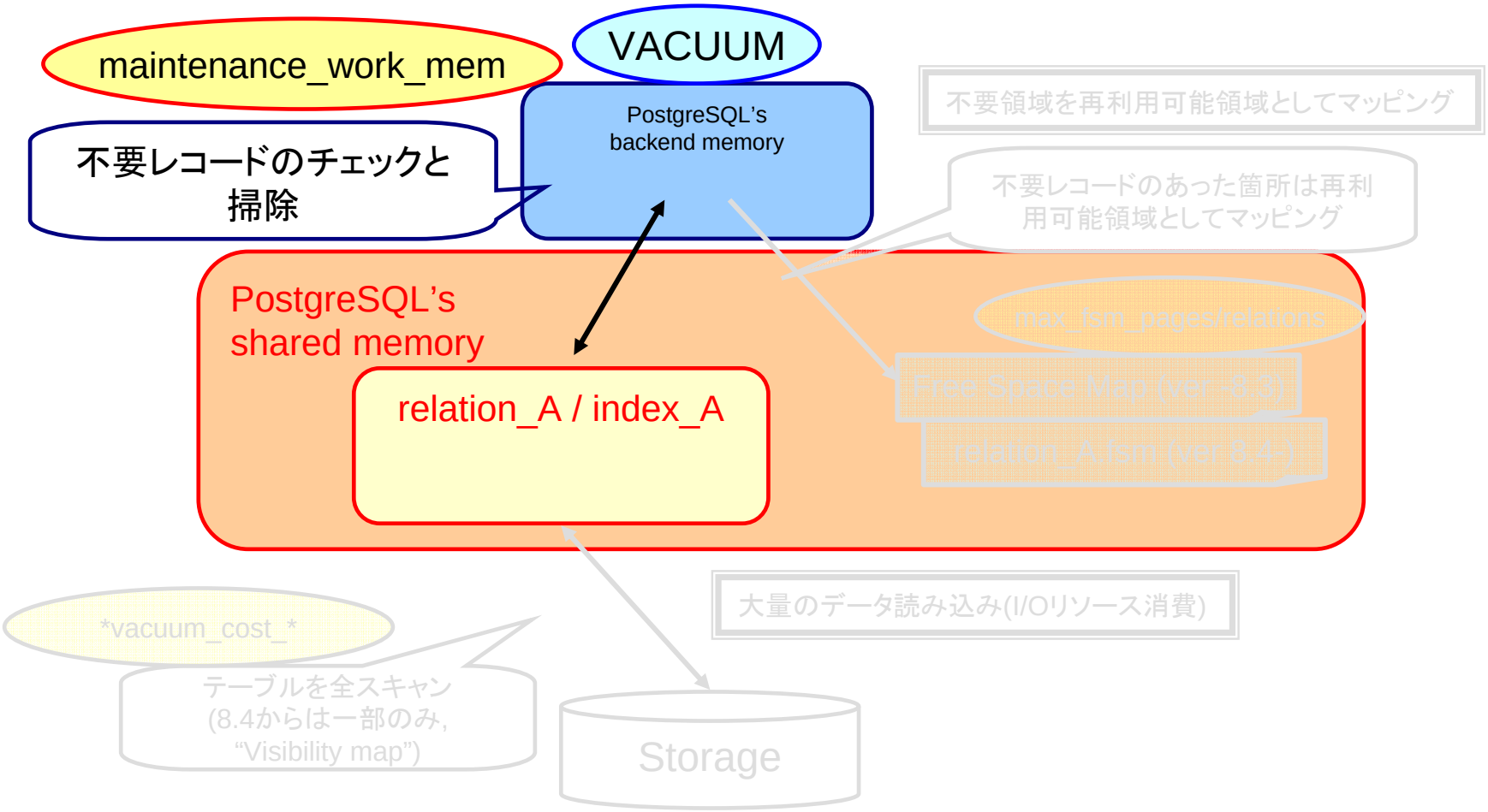
Background: PostgreSQL mechanism(VACUUM処理の流れ)

- Maintenance(VACUUM)
  - CHECK:不要領域を全て回収できていますか？
  - CHECK:VACUUMのI/O負荷を制御できていますか？



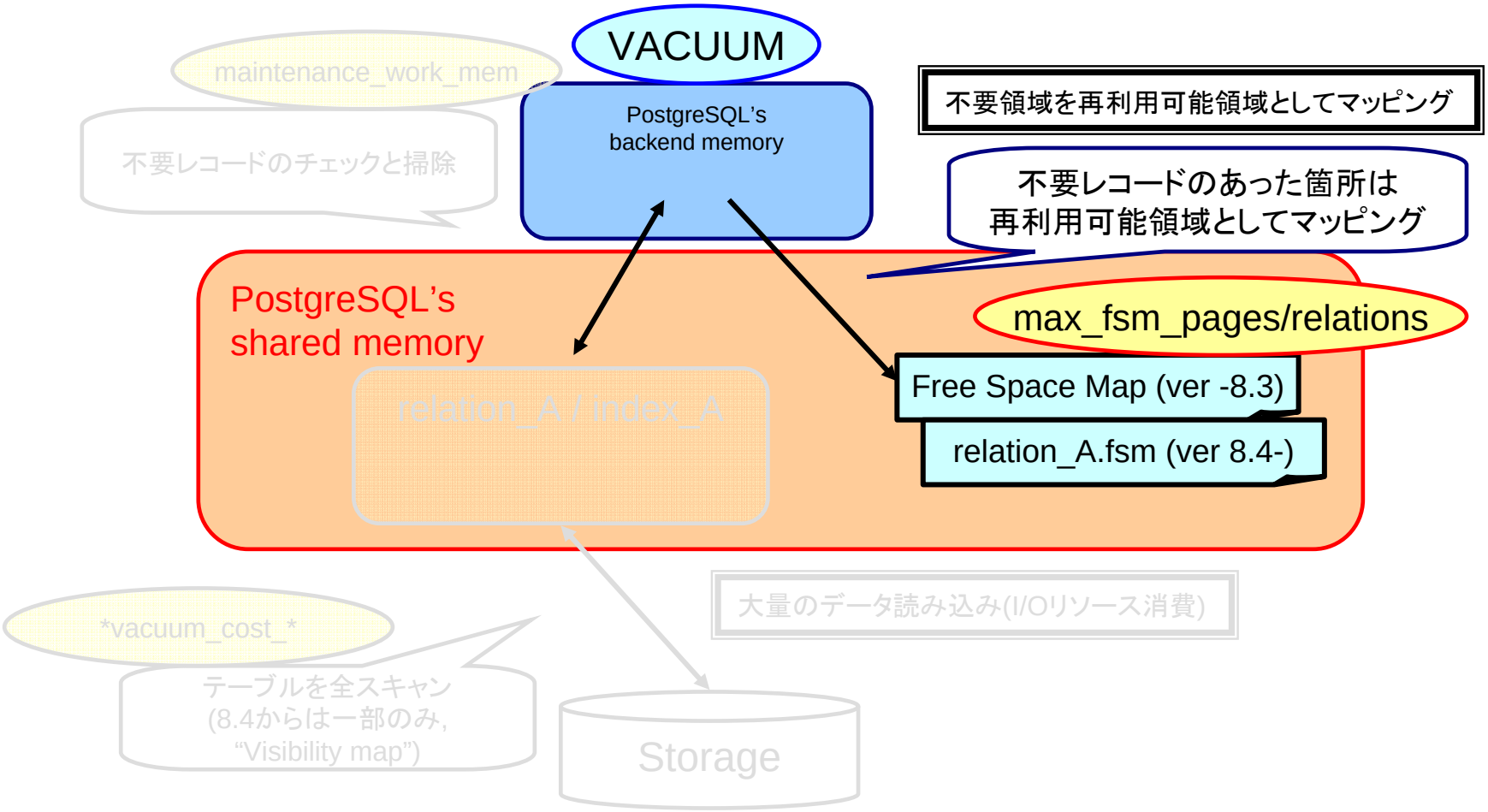
Background: PostgreSQL mechanism(VACUUM処理の流れ)

- Maintenance(VACUUM)
  - CHECK:不要領域を全て回収できていますか？
  - CHECK:VACUUMのI/O負荷を制御できていますか？



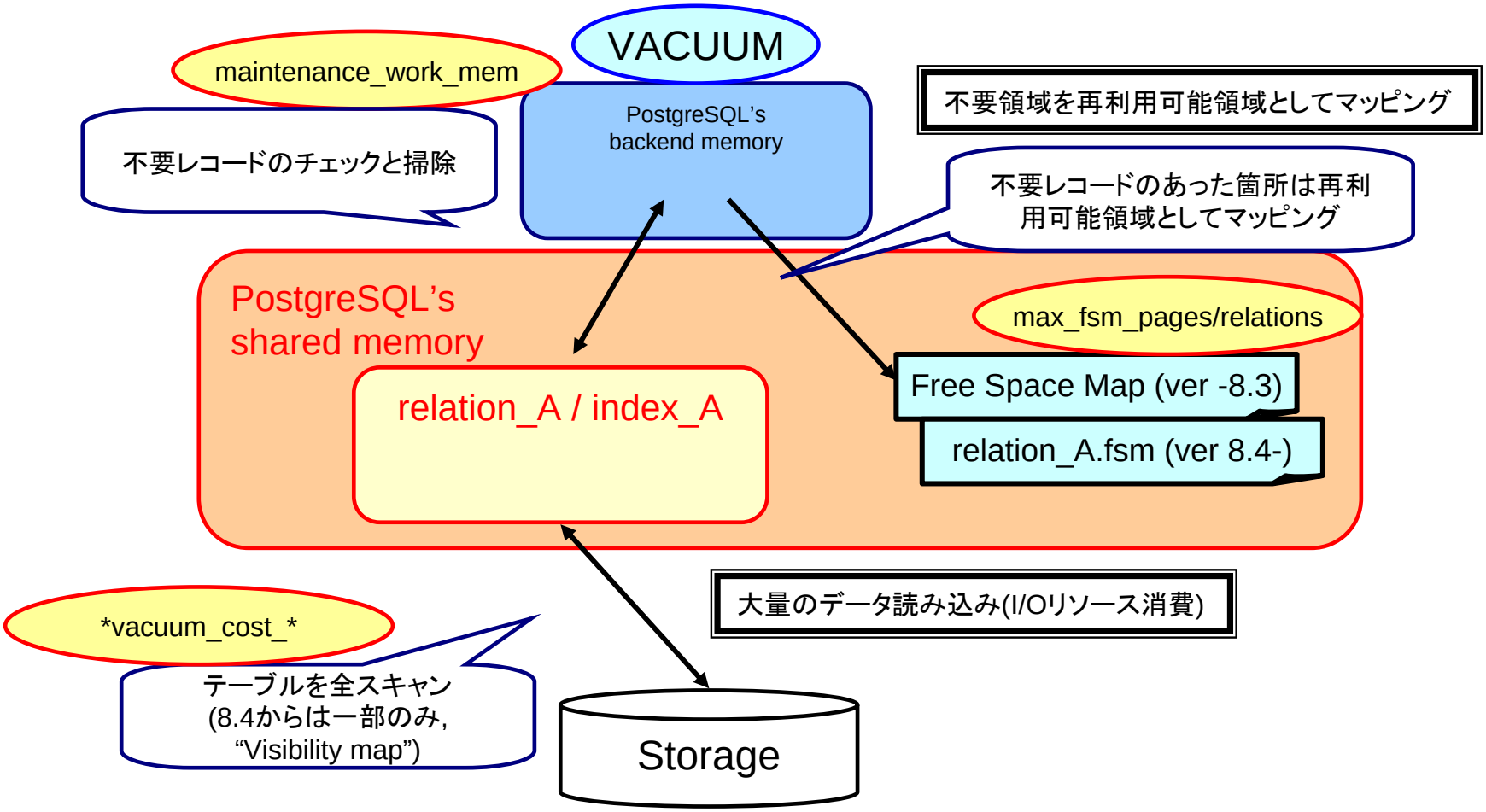
Background: PostgreSQL mechanism(VACUUM処理の流れ)

- Maintenance(VACUUM)
  - CHECK:不要領域を全て回収できていますか？
  - CHECK:VACUUMのI/O負荷を制御できていますか？



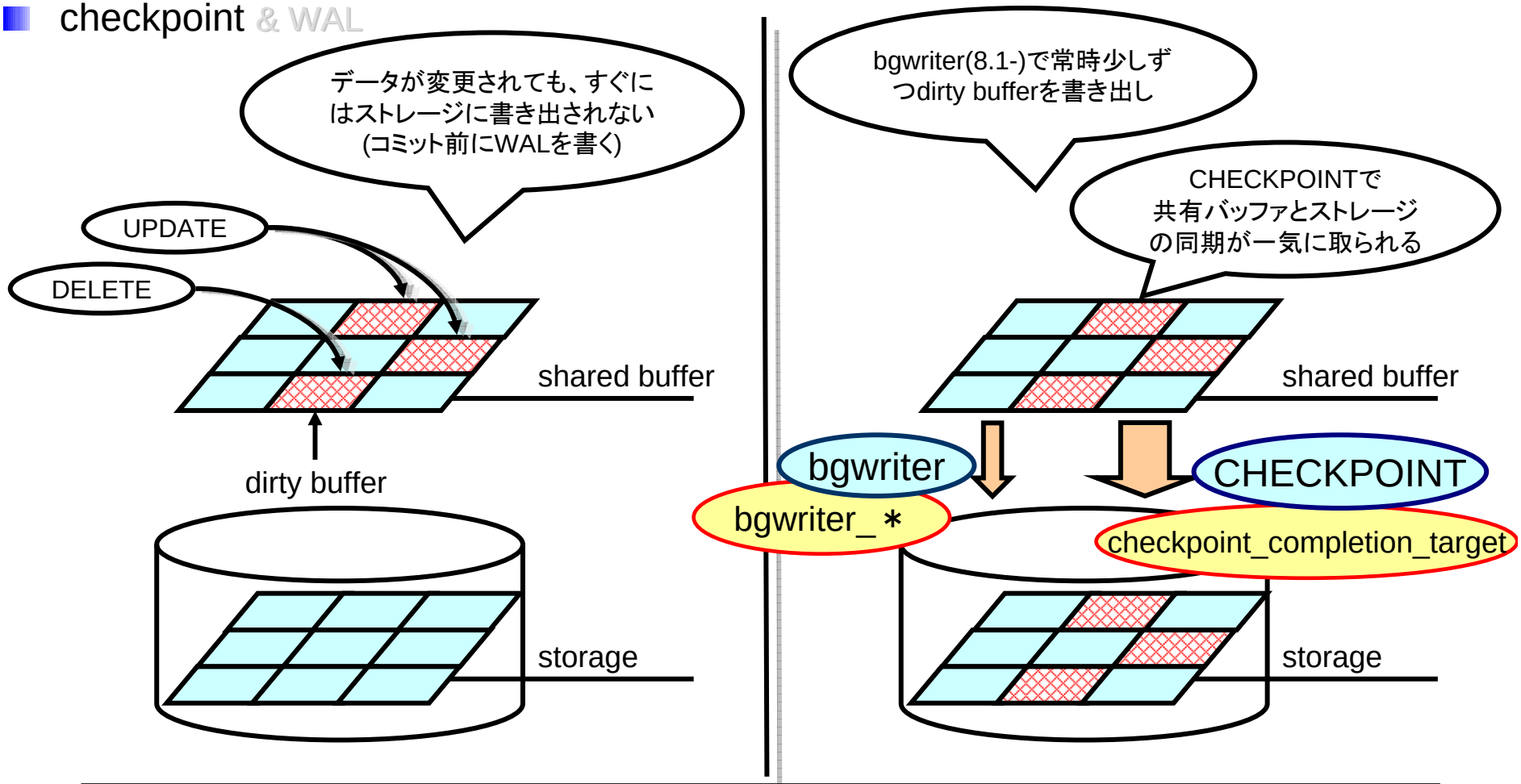
Background: PostgreSQL mechanism(VACUUM処理の流れ)

- Maintenance(VACUUM)
  - CHECK:不要領域を全て回収できていますか？
  - CHECK:VACUUMのI/O負荷を制御できていますか？



Background: PostgreSQL mechanism(checkpointとdirty bufferを知る)

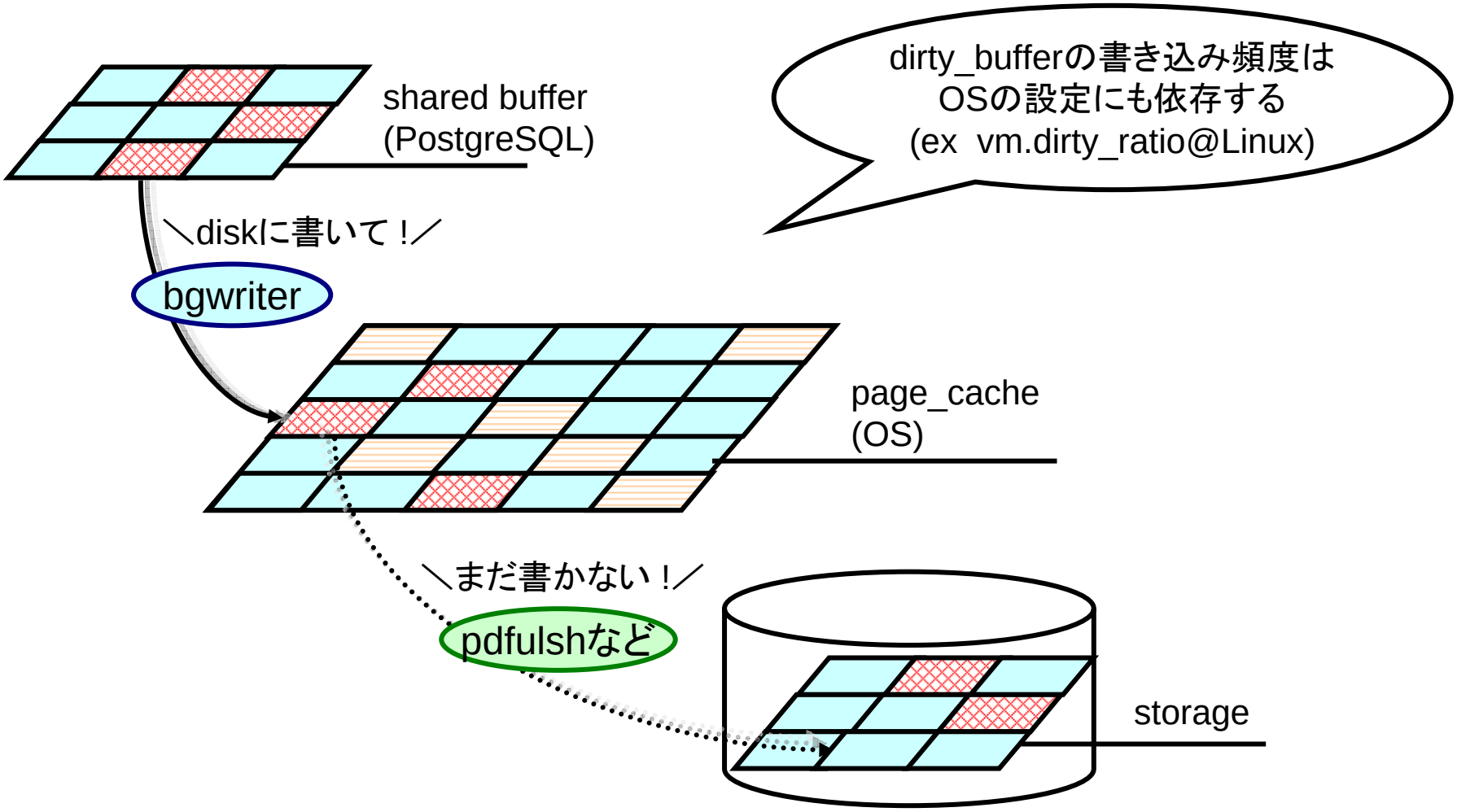
- checkpoint & WAL



- shared_buffersが大きいほど、checkpoint時に書き出すデータ量が多くなる。
  - 8.0 で実装された bgwriterによりcheckpoint時の書き出し量を低減できる。
  - 8.3 からは、「ゆっくり」checkpointを行う機能が入り、さらにI/O負荷が平滑化された。

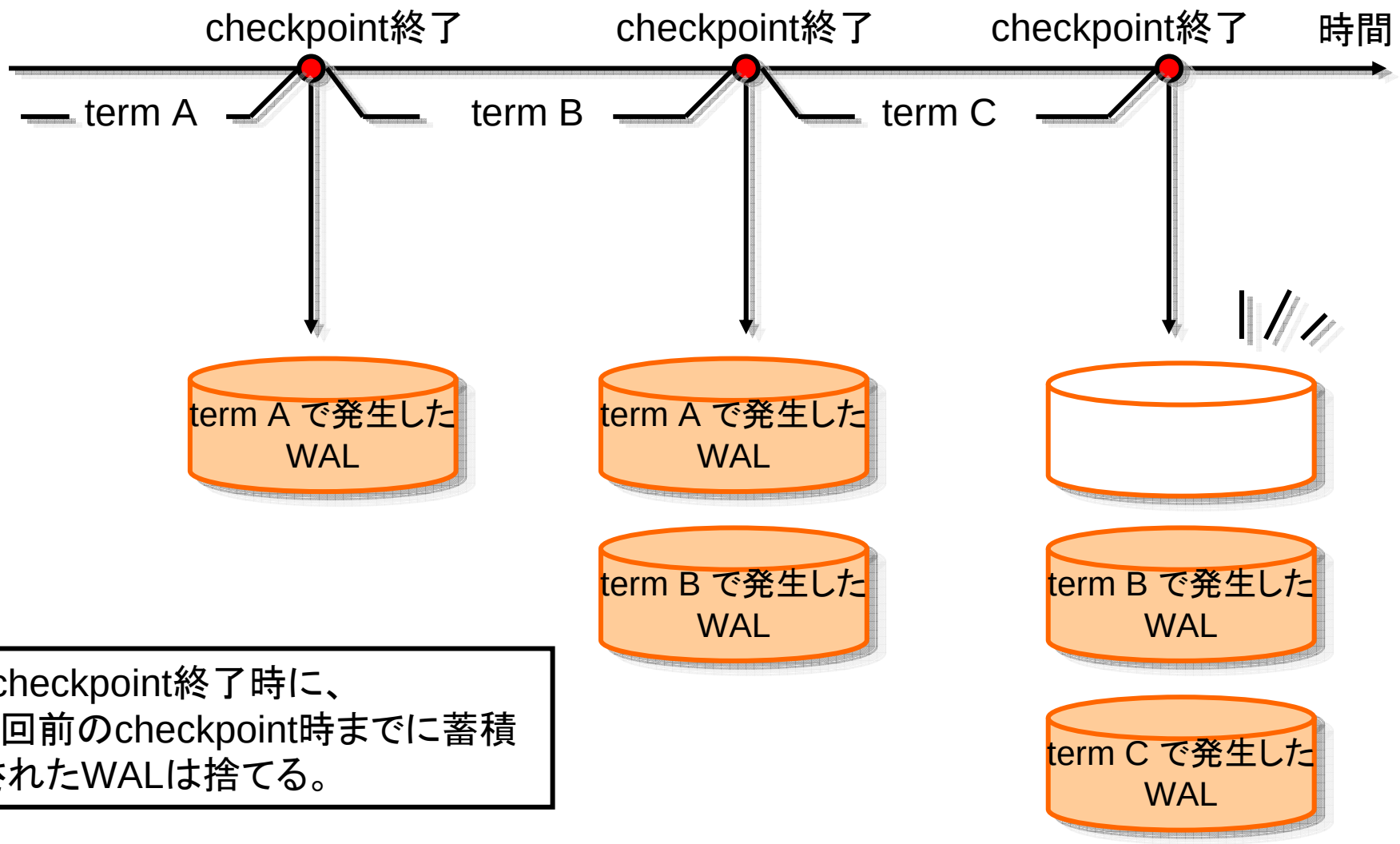
Background: PostgreSQL mechanism(checkpointとdirty bufferを知る)

- checkpoint & WAL



Background: PostgreSQL mechanism(checkpointとWALの関係)

- checkpoint & WAL



checkpoint終了時に、
2回前のcheckpoint時までに蓄積されたWALは捨てる。

Preparing for tuning

- logging and runtime statistics
  - PostgreSQLのログと稼動統計情報は、問題把握・効果確認に便利！
- OS resource monitoring
  - sar, vmstat, iostat, ps …
- benchmark tool
  - pgbench …
- profiler
  - Oprofile、Dtrace、systemtap ...
- (reference) Let's postgres
  - ログ関連の設定
    - http://lets.postgresql.jp/documents/technical/log_setting
  - 稼動統計情報を活用しよう
    - http://lets.postgresql.jp/documents/technical/statistics/1

Parameters tuning

- 性能に深く関係するパラメータについて
  - 何を設定するのか？
  - 設定方針はどうするか？
  - パラメータいかんでどのような影響があるのか？

を、パラメータの具体的な働きと共に解説します

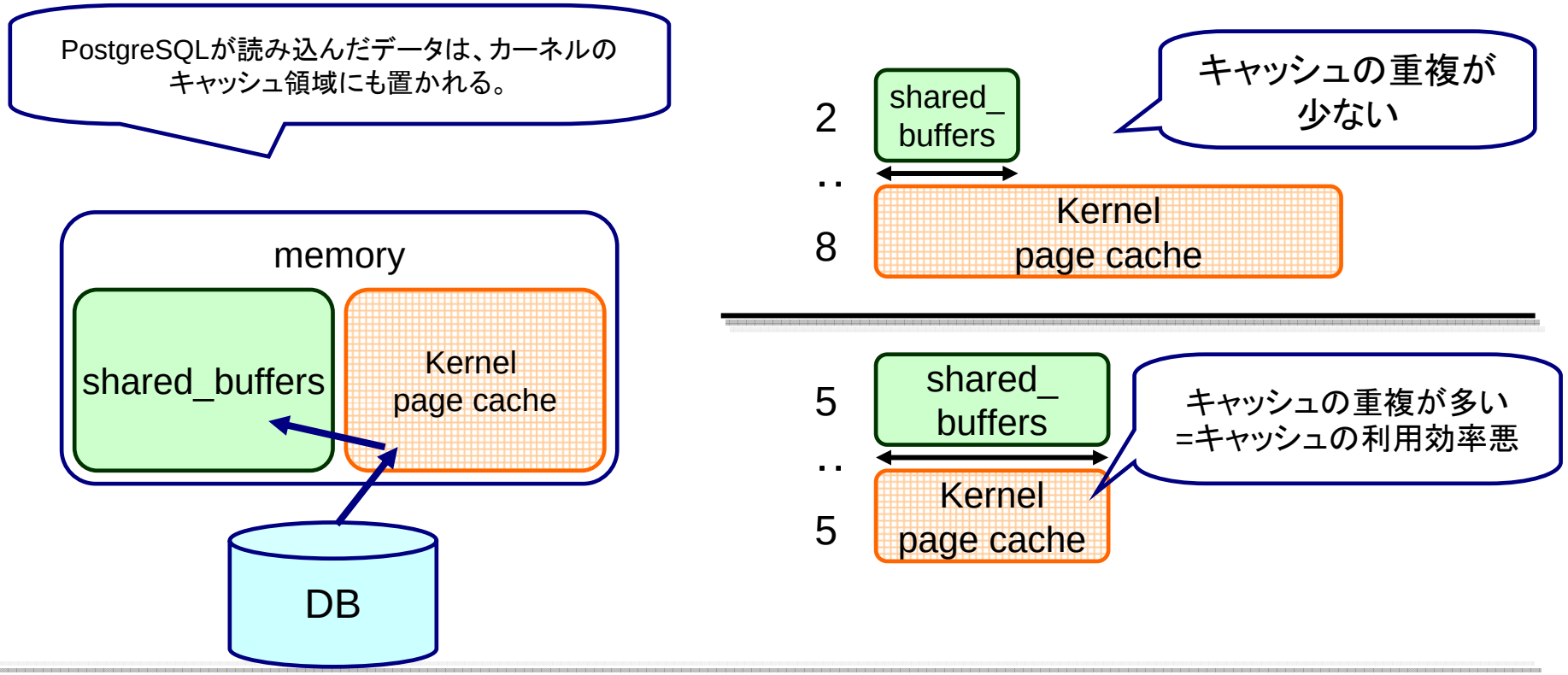
shared memory & checkpoint (summary)

relevant parameters

parameters	ver	description	setting guideline
shared_buffers	all	共有バッファのサイズ調整 adjustment of shared buffer size	物理メモリの10 - 20 % 10% - 20% of physical memory
checkpoint_segments	all	チェックポイントの実施間隔調整(WAL数) adjustment of checkpoint interval (WAL segments)	[参照メイン (Web)] - 16 [更新メイン (OLTP)] 32 - 64
checkpoint_completion_target	8.3 -	チェックポイント処理の速度調整 control checkpoint speed	[参照メイン(Web)] default [更新メイン(OLTP)] 0.6 - 0.9
bgwriter_all_maxpages	- 8.2	常時書き出すダーティページ数の調節 adjustment of dirty pages to write regularly	[参照メイン(Web)] - 40 [更新メイン(OLTP)] 100 -
bgwriter_all_percent			[参照メイン(Web)] - 5 [更新メイン(OLTP)] 5 - 10

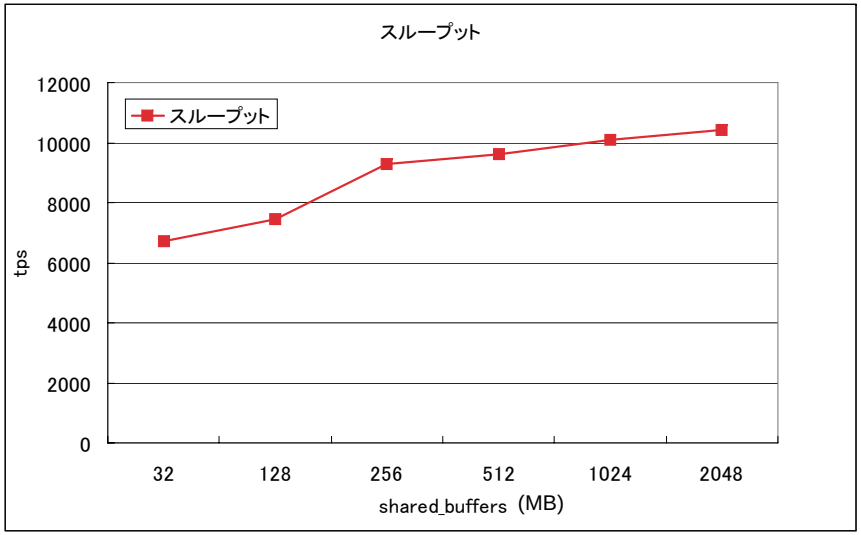
shared_buffers (version：ALL)

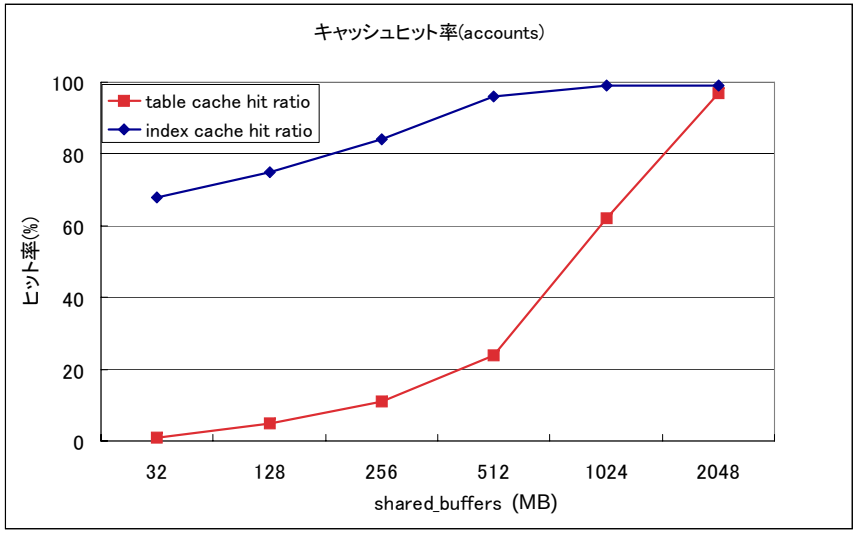
- 何を設定する？
  - PostgreSQLの共有メモリサイズ
- 設定方針は？
  - 物理メモリの10 - 20% or DB サイズ (DBがメモリに乗り切る場合)
- 影響は？
  - (小さいと) 共有メモリとページキャッシュでのデータのやり取りが頻発
  - (大きいと) checkpoint時に高いI/O負荷が発生 or スワップが起きやすい



Check & Trouble Shoot

- pgbenchの結果
  - DBサイズ：約1.5GB
  - -S オプションによる参照のみの負荷を10分間実施

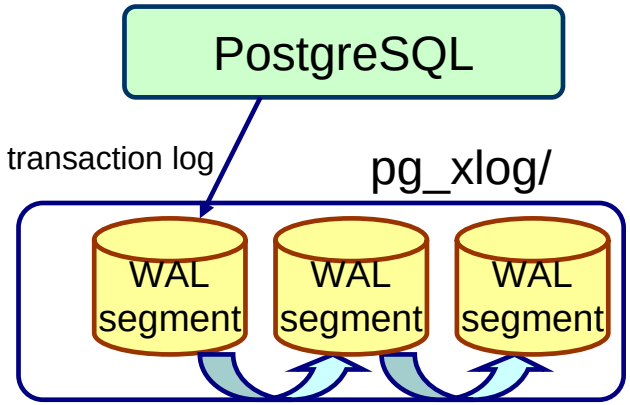




・キャッシュヒット率(shared_buffersにヒット)が低くても、ページキャッシュで救われているので極端にスループットは悪くない
・インデックスが乗り切ってしまえばおよそ問題なし

checkpoint_segments(version:ALL)

- 何を設定する？
  - WALセグメントの数を調節
- 設定方針は？
  - 16 ～ 64
- 影響は？
  - (小さいと) checkpoint が頻発する
  - (大きいと) 蓄積されるWALの量が膨大になる



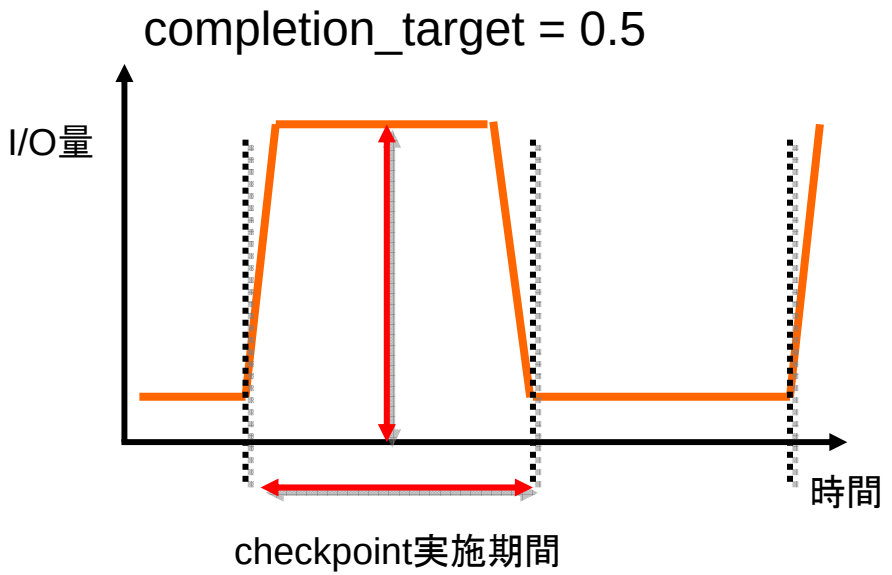
トランザクションログをWALセグメントにどんどん書き込み、セグメントが一杯になると次のセグメントへ書き出す。
「checkpoint_segments分のセグメントが一杯になる」
or
「checkpoint_timeout時間分を経過する」
とcheckpointが発生

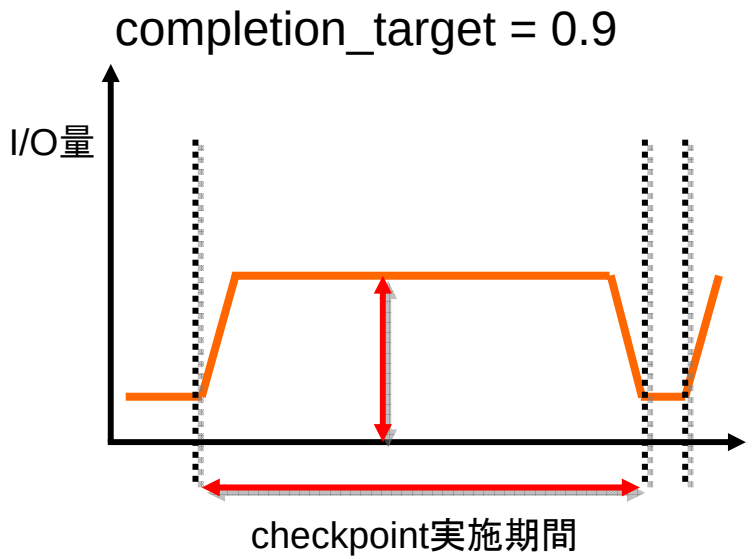
checkpoint_segmentsの値が小さいとcheckpointが頻発する。デフォルトの3では小さすぎることが多いので、増やしておくと吉。

なお、WALセグメント(1個16MB)は最大で「checkpoint_segments * 3 + 1」のサイズになるので、その分のストレージは確保しましょう。

checkpoint_completion_target(version: 8.3 -)

- 何を設定する？
  - チェックポイントのスピード
- 設定方針は？
  - 更新処理が多い場合は 0.6 - 0.9
- 影響は？
  - (小さいと) checkpoint 時のI/O負荷が性能に影響を与える
  - (大きいと) クラッシュリカバリ時の復旧が遅くなる





クラッシュリカバリにかかる時間 ≒
checkpoint_completion_target × checkpoint_timeout

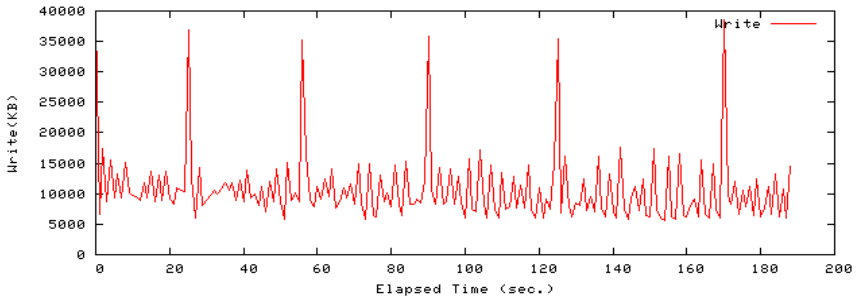
Check & Trouble Shoot

- checkpoint の頻発チェック
  - サーバログ に以下のメッセージが出ていませんか？

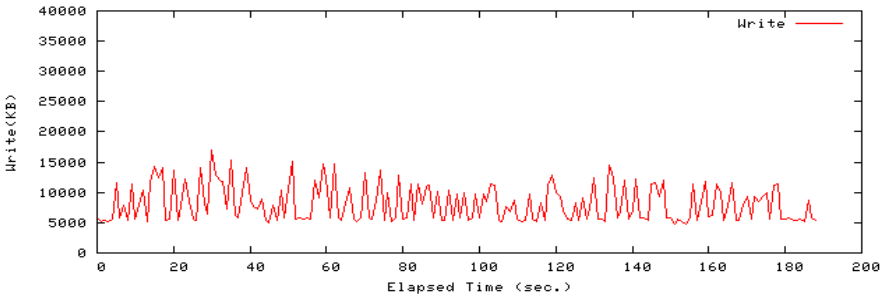
checkpoints are occurring too frequently…

- checkpointの負荷
  - checkpoint時のwrite量がストレージのI/O性能を上回っていませんか？
  - iostatのwrite量やvmstatのiowaitに着目しましょう

iostat でみたpgbench実施中のwrite量の遷移(8.2)



bgwriter 関連 normal



bgwriter_max_percent=5, bgwriter_max_allpages=40

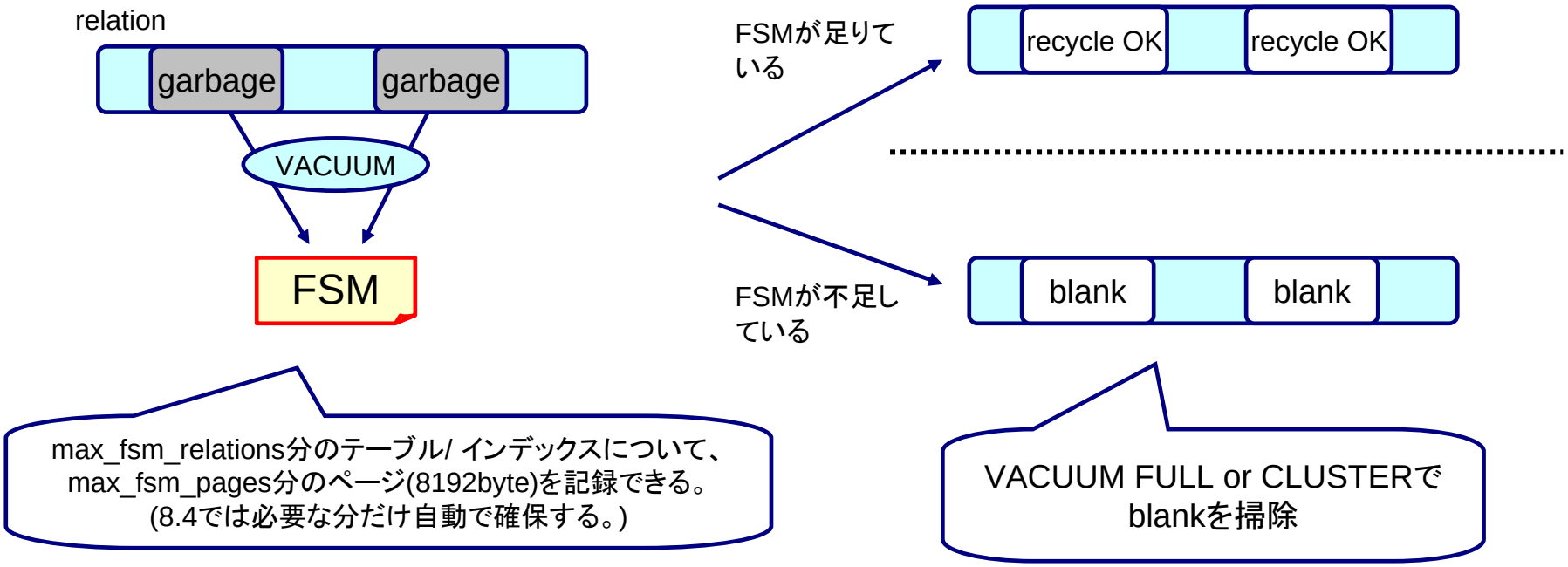
VACUUM(autovacuum) (summary)

relevant parameters

parameters	ver	description	setting guideline
max_fsm_relations	- 8.3	FSM(Free Space Map)で追跡可能なテーブル/インデックスの調節 Adjustment of tables/indexes number that trace for FSM	DBクラスタ内のユーザテーブル+インデックス数 + DB個数 × 160 All user tables + indexes + 160 × # of DB.
max_fsm_pages	- 8.3	FSMで追跡可能なページ数の調節 Adjustment of pages number that trace for FSM	DB size(byte) × 0.2 / 8192
autovacuum_max_workers	8.3 -	同時にVACUUMを行うautovacuumプロセスの最大数 Adjustment number of autovacuum worker process	DB sizeの20%を超えるくらいの大きなテーブル数 + 1 # of large size table (over 20% of DB size) + 1
autovacuum_vacuum_cost_limit	8.2 -	autovacuum処理の遅延契機の調整 Control autovacuum speed.	200
log_autovacuum_min_duration	8.3 -	ログ出力対象とするautovacuumの所要時間の指定 logging autovacuum requires long time	1min -
maintenance_work_mem	all	VACUUM、REINDEX用の作業メモリサイズ Adjustment memory size for VACUUM/REINDEX	最も行数の多いテーブルの行数 × 0.2 × 6 # of records (the table with much records) × 0.2 × 0.6

max_fsm_pages / relations (version: - 8.3)

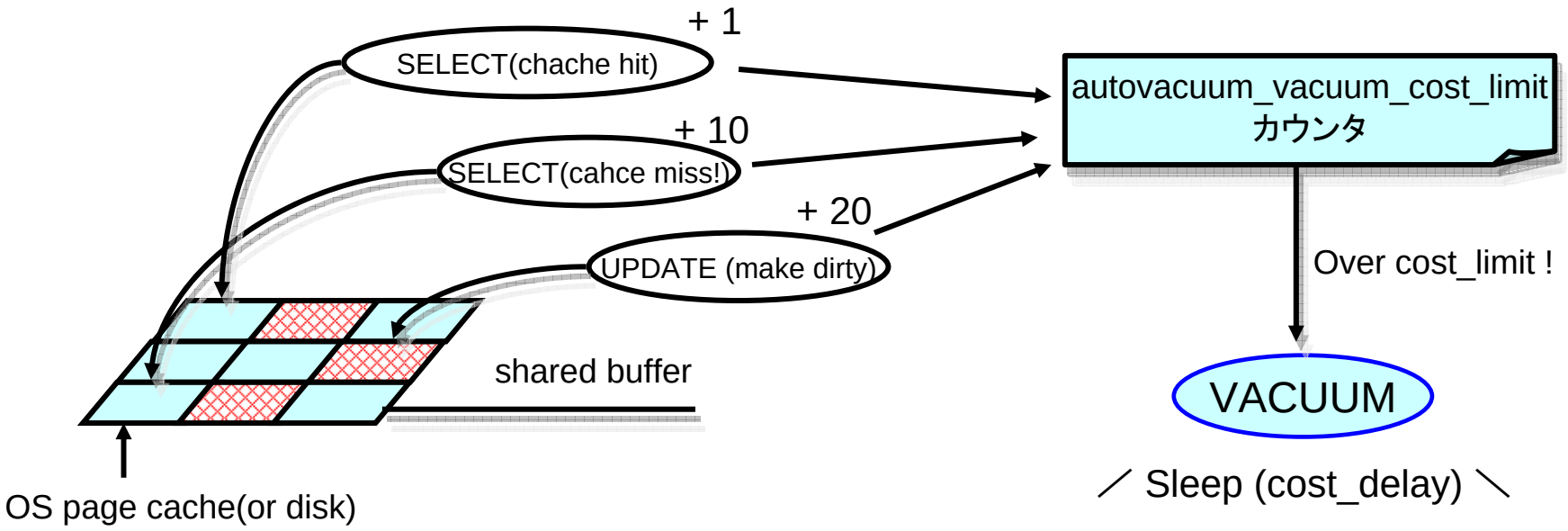
- 何を設定する？
  - FSM (Free Space Map) のサイズ
- 設定方針は？
  - pages : DB size(byte) * 0.2 / 8192 (vacuum実施間隔でDBの2割が更新されると仮定))
  - relations : number of user tables + indexes + DB * 160(← system tables)
- 影響は？
  - (小さいと) VACUUMをしてもDBが肥大化してしまう
  - (大きいと) 更新系処理がやや低速になる



autovacuum_vacuum_cost_limit (version: 8.2 -)

- 何を設定する？
  - autovacuumの遅延処理の調節
- 設定方針は？
  - ディスク本数×30程度を目安
- 影響は？
  - (小さい) autovacuumにかかる時間が延びる
  - (大きい or 無効) autovacuumがI/O帯域を消費しやすい

cost_limit 大 ⇔ 小
VACUUM 全力 ⇔ ゆっくり



実施されるSQL(特にI/Oが発生しやすいもの)が多くなるほどVACUUMがsleepしやすくなります。VACUUMの処理自体もカウンタに累積されることに注意！

autovacuum_vacuum_cost_limit (version: 8.2 -)

- （参考）
  - cost_limitで、
    - VACUUMのI/O帯域はどれくらいになる？

VACUUMが消費するI/O帯域(byte/s)の目安 <=
　cost_limit * 8192 * 1000 / {10 ～ 40} / cost_delay

cost_limit = 200 の場合
 200 * 8192 * 1000 / 10 / 20 = 8MB/s がVACUUMの最大消費I/O帯域

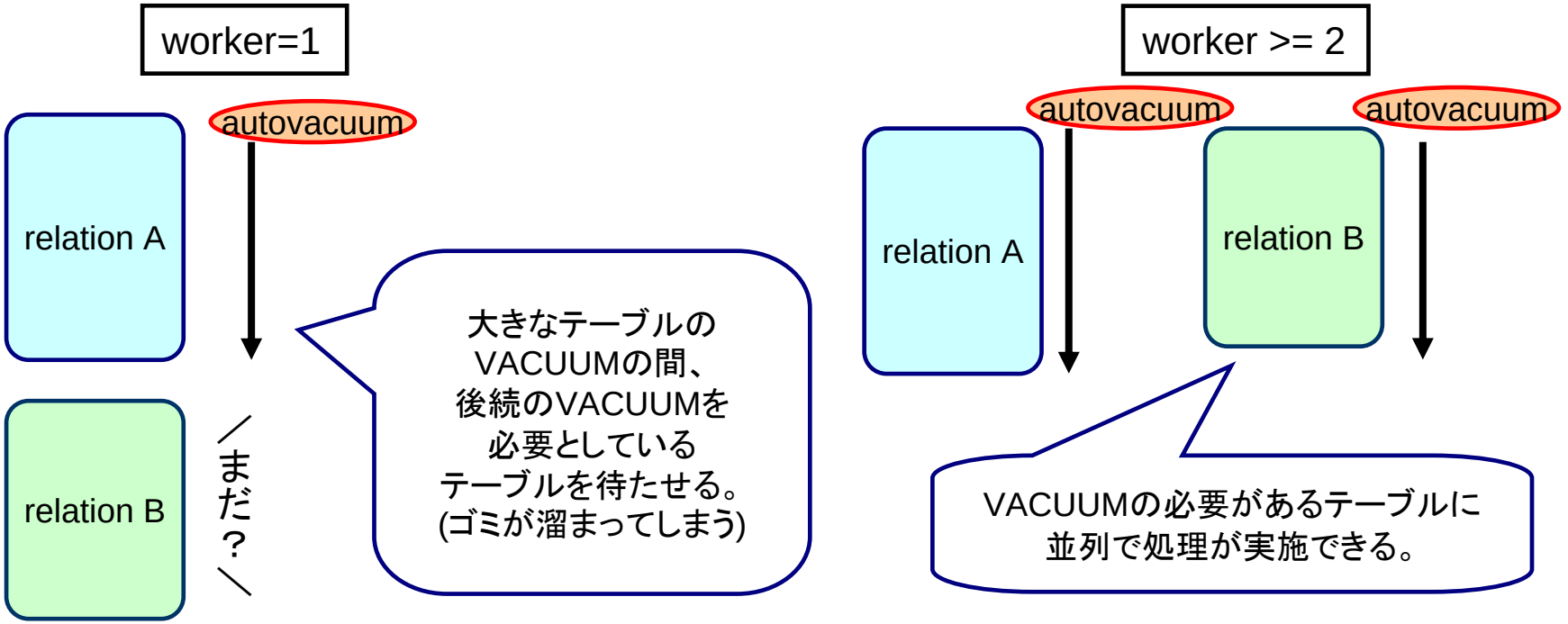
    - VACUUMはどれくらい遅延される？

延びる時間の目安(ms) =>
{10 ～ 40} * テーブルサイズ(byte) / 8192 / cost_limit * cost_delay

cost_limit = 200 & テーブルサイズ1GBの場合
40 * 1024^3 / 8192 / 200 * 20 ≒ 8.7分 程度、素のVACUUMより
時間がかかる

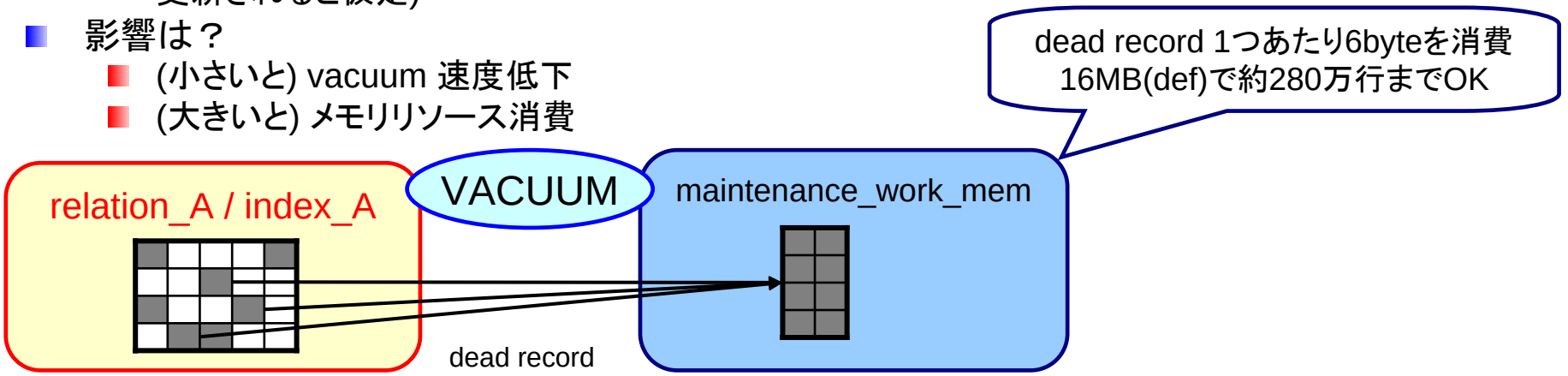
autovacuum_max_workers(version:8.3 -)

- 何を設定する？
  - 同時に実行されるautovacuumプロセスの数
- 設定方針は？
  - 大きなテーブル (over 20% of DB size)の数 + 1
- 影響は？
  - (小さいと) VACUUMが必要なテーブルに長時間VACUUMされない



maintenace_work_mem(version:all)

- 何を設定する ？
  - VACUUMやREINDEX用の作業メモリサイズ
- 設定方針は？
  - 最も行数の多いテーブルの行数 * 0.2 * 6 (vacuum実施間隔でテーブルの2割が更新されると仮定)
- 影響は？
  - (小さいと) vacuum 速度低下
  - (大きいと) メモリリソース消費



VACUUM処理の流れ
1. dead record を読み取り maintenance_work_memへ列挙
2. dead record に対応する index を掃除
3. dead record を掃除

！maintenace_work_mem が小さいと、1-3を何度も繰り返すので遅くなる
VACUUM実施間隔で発生するガベージの分を保持できるのが理想

Check & Trouble Shoot

- FSMの不足確認
  - サーバログ に以下のメッセージが出ていませんか？
    - FSMが不足している状況なので、FSMを増やしましょう

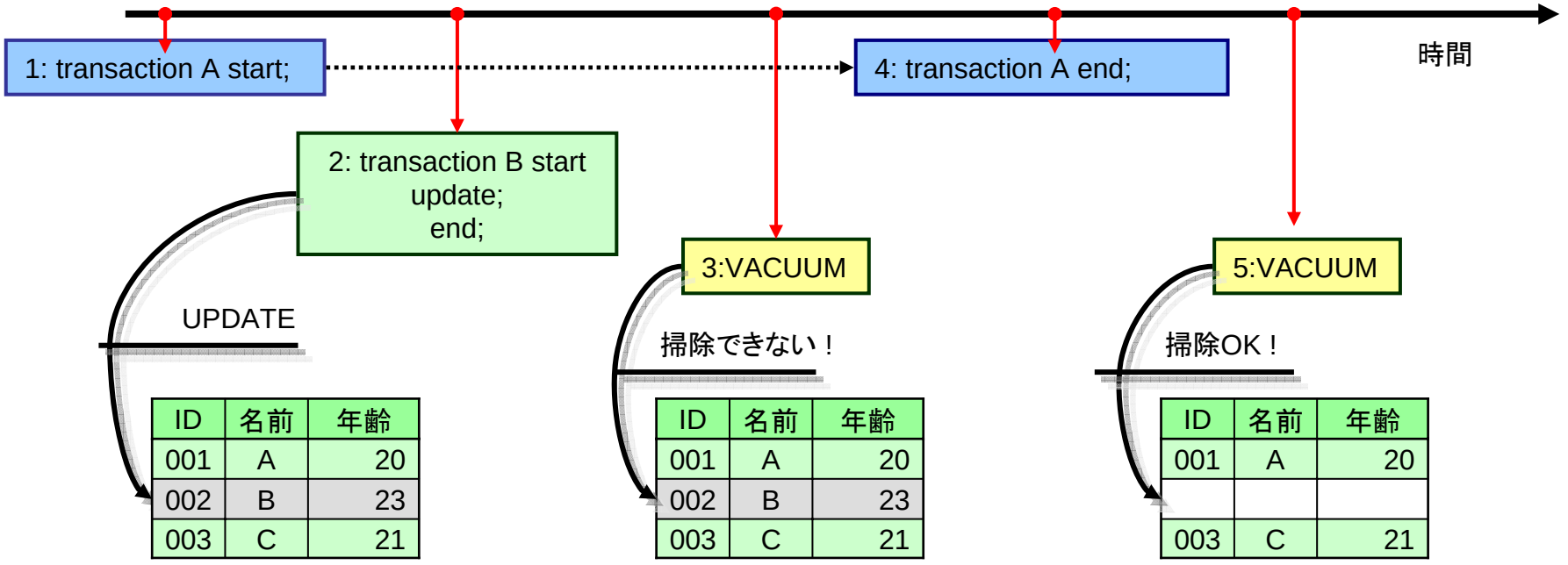
```
NOTICE:  number of page slots needed (9280) exceeds max_fsm_pages (2048)
HINT:  Consider increasing the configuration parameter "max_fsm_pages" to a value over 9280.
```

(注意_1)DB全体へのVACUUMでのみ上記のログは出力されます
(注意_2)上記はmax_fsm_pages=9280まで増やす必要があったことを示唆するものです。

  - 長期FSMが不足していた場合、あるいは極端にFSMが不足していた場合は、テーブルの充填率を回復させましょう
    - 一度肥大化したテーブルは、VACUUM FULL、あるいはCLUSTERで物理的な圧縮を実施すると良いでしょう
    - ストレージに空き容量がある(対象テーブルの2倍以上)場合は、CLUSTER + ANALYZEがお勧めです
    - ストレージに空き容量は無い場合はVACUUM FULL＋REINDEX + ANALYZEがお勧めです
      - 時間がかかるので注意して実施して下さい！

Check & Trouble Shoot

- ロングトランザクションの確認
  - 長時間実施されているトランザクションはありませんか？
    - 「最古のトランザクション」以降に開始されたトランザクションにより発生したガベージは、VACUUMやHOTでは除去できません



Check & Trouble Shoot

- ロングトランザクションの確認
  - 疑わしい場合は下記をチェックしてみましょう
    - 下線部の数値が多い場合はガベージの掃除が阻害されている可能性あがります
  - VACUUM VERBOSEログ

```
INFO:  "test": found 0 removable, 1099999 nonremovable row versions in 9167 pages
DETAIL:  99999 dead row versions cannot be removed yet.
```

  - autovacuumログ(log_autovacuum_min_duration を調節して出力できます)

```
LOG:  00000: automatic vacuum of table "postgres.public.test": index scans: 1
    pages: 0 removed, 12500 remain
    tuples: 0 removed, 1399999 remain
    system usage: CPU 0.01s/0.08u sec elapsed 38.71 s
```

remain(残った)レコード数が
多すぎませんか？
逆にremoved(掃除済み)の
レコード数が少なすぎませんか？

Check & Trouble Shoot

ロングトランザクションそのものがいないかを確認

```
-- 8.3 以降で実施可能
=# SELECT procpid, waiting, (current_timestamp - xact_start)::interval(3)
AS duration, current_query FROM pg_stat_activity
WHERE procpid <> pg_backend_pid();

procpid  | waiting |   duration   | current_query
---------+---------+--------------+-----------------------------
10434    |       f | 00:04:52.846 | <IDLE> in transaction
10440    |       t | 00:04:48.974 | SELECT * FROM t FOR UPDATE;
```

duration(トランザクション開始からの経過時間)が長いものはいませんか？

```
-- 8.2以前はこちらの方法で実施可能
=# SELECT procpid, age(transaction), current_query
FROM pg_stat_activity a, pg_locks l WHERE a.procpid = l.pid
AND locktype = 'transactionid' AND pid <> pg_backend_pid();

procpid  |  age  | current_query
---------+-------+-----------------------------
10434    | 10016 | <IDLE> in transaction
10440    | 10016 | SELECT * FROM t FOR UPDATE;
```

age(どれくらい昔にトランザクションが開始されたか)が大きなものはいませんか？

左記の場合、100TPMのシステムであれば100分間継続されていると推測できる。

Check & Trouble Shoot

- maintenace_work_memが不足していないかチェック
  - VACUUM VERBOSE ログ

```
(中略)
INFO:  scanned index "test_idx" to remove 174480 row versions
DETAIL:  CPU 0.17s/0.39u sec elapsed 1.73 sec.

INFO:  "test": removed 174480 row versions in 1454 pages
DETAIL:  CPU 0.02s/0.01u sec elapsed 0.05 sec.

INFO:  scanned index "test_idx" to remove 174480 row versions
DETAIL:  CPU 0.19s/0.38u sec elapsed 2.90 sec.

INFO:  "test": removed 174480 row versions in 1454 pages
DETAIL:  CPU 0.02s/0.01u sec elapsed 0.05 sec.

INFO:  scanned index "test_idx" to remove 174480 row versions
DETAIL:  CPU 0.20s/0.33u sec elapsed 2.82 sec.

INFO:  "test": removed 174480 row versions in 1454 pages
DETAIL:  CPU 0.02s/0.01u sec elapsed 0.05 sec.
(中略)
```

一つのテーブルについて、このように複数回 index & table 掃除が繰り返されていませんか？

work_mem (version: all)

- 何を設定する？
  - ソートやハッシュ用の作業メモリサイズの調節
- 設定方針は？
  - ……
- 影響は？
  - (小さいと) ソートやハッシュ処理の速度低下
  - (大きいと) メモリの枯渇とスワップ発生

ソートやハッシュ処理に必要となるメモリ量はAPに依存します。
そのため、下記のパラメータを利用して不足しているかどうかのチェックを行いつつ値を調整すると良いでしょう。
ver -8.2 : trace_sortパラメータを有効にし、ソート処理結果をログで確認
ver 8.3 - : EXPLAIN ANALYZE 句を該当SQLに付与し、実行計画を確認
(次ページ参照)

work_memを引き上げる場合はスワップに注意しましょう。最大で
max_connections * work_mem + shared_buffers がPostgreSQLで消費される可能性があります。

(参考)EXPLAIN ANALYZE

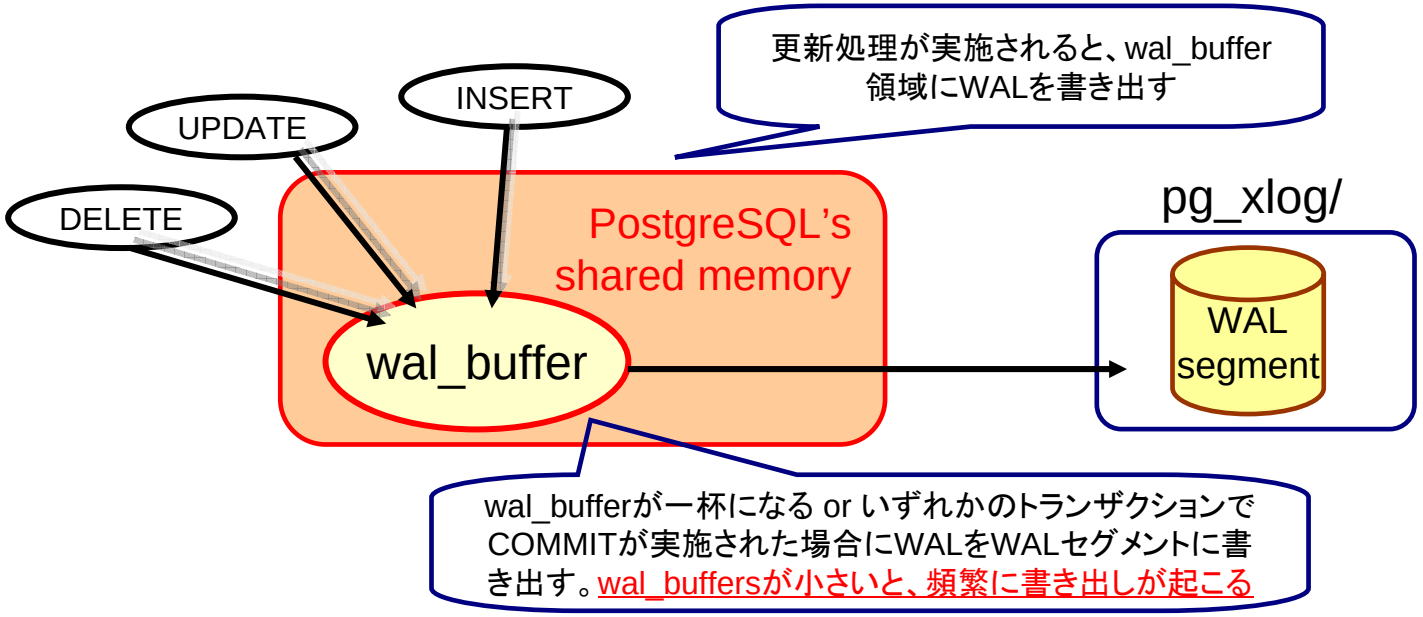
8.4はEXPLAIN ANALYZEでソート処理内容を確認可能

```
=# EXPLAIN ANALYZE SELECT * FROM test2 ORDER BY 1;
                                    QUERY PLAN

-----------------------------------------------------------------------------------------------
 Sort  (cost=1.27..1.29 rows=10 width=4) (actual time=0.112..0.115 rows=10 loops=1)
   Sort Key: id
   Sort Method:  quicksort  Memory: 25kB (メモリ不足の時はexternal sort  Disk: nnnnnnkB)
   ->  Seq Scan on test2  (cost=0.00..1.10 rows=10 width=4)
                                        (actual time=0.010..0.013 rows=10 loops=1)
 Total runtime: 0.155 ms
(5 rows)
```

wal_buffers (version: all)

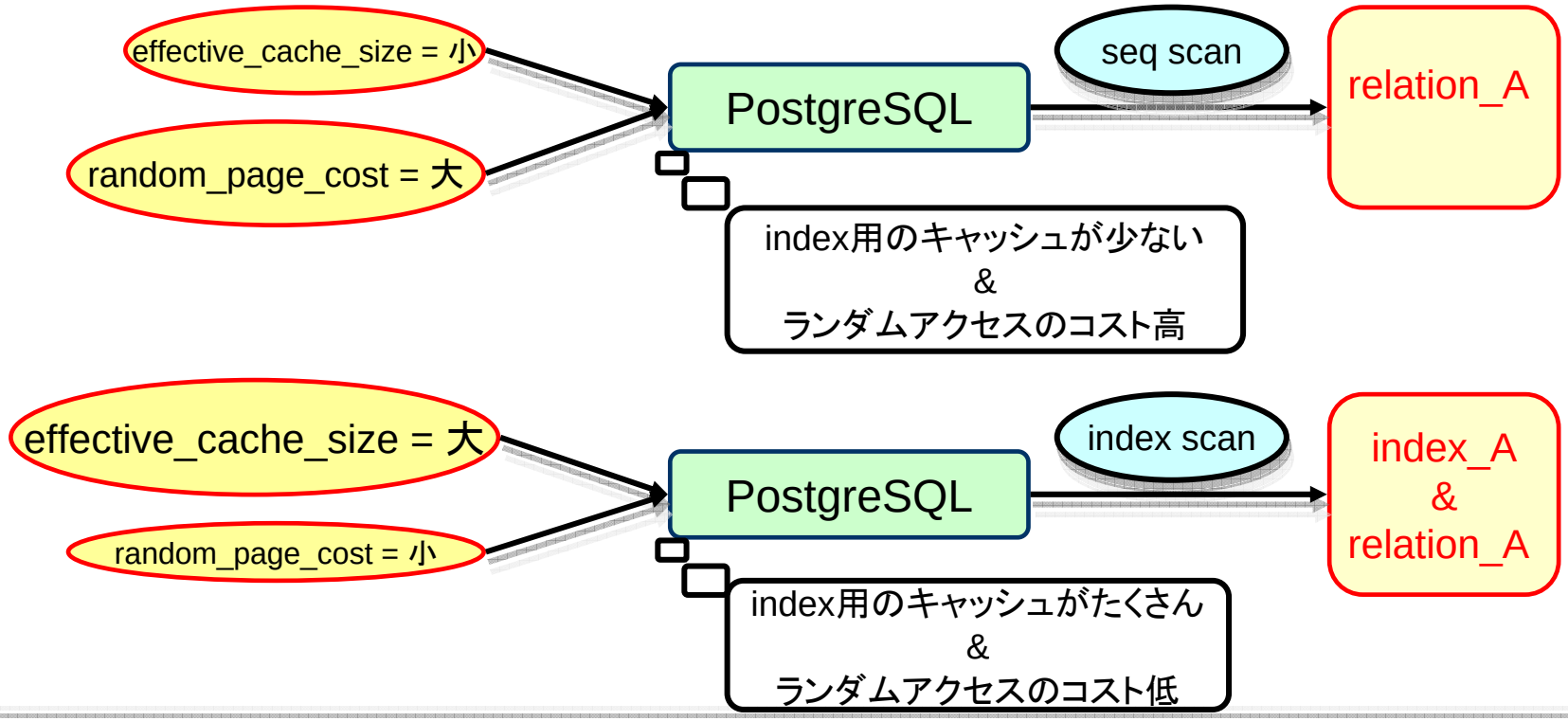
- 何を設定する？
  - WALバッファのサイズ
- 設定指針は？
  - 4MB -
- 影響は？
  - (小さいと) 更新処理の速度低下



以下のケースでは、wal_buffersを増やしておくと良いでしょう
・1トランザクションで巨大なレコード、あるいは大量の行を更新する場合
・多数のクライアントが同時に更新トランザクションを実施している場合

Plan tuning - effective_cache_size & random_page_cost(version: all)

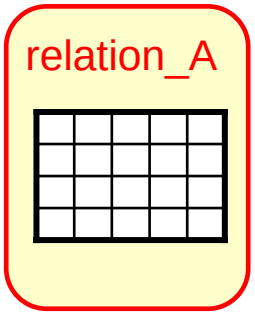
- 何を設定する？
  - effective_cache_size : index用に使えるキャッシュサイズ(内部変数)の調整
  - random_page_cost : 1回のランダムアクセスにかかる推定コストの調整
- 設定方針は？
  - effective_cache_size : 50 - 60% of physical memory
  - ranodm_page_cost : 2 - 3
- 影響は？
  - (特にeffective_cache_sizeが小さいと) index scan がされづらい

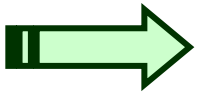


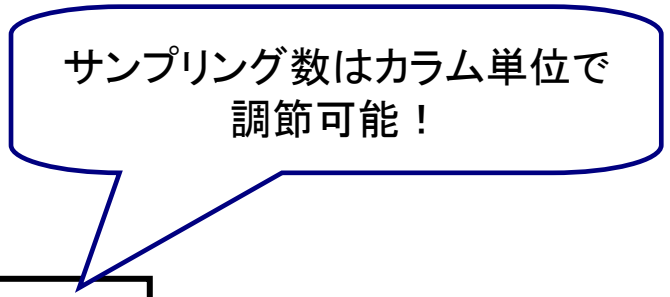
Plan tuning - default_statistics_target (version: all)

- 何を設定する？
  - ANALYZEでのサンプリング数
- 設定方針は？
  - LIKE検索があるカラムは100
- 影響は？
  - (小さいと) 適切な実行計画が作成されない
  - (大きいと) ANALYZEが長時間に及ぶ(ロングトランザクション問題)

SELECT…. FROM relation_A WHERE col LIKE ‘…..’;
→ LIKE検索性能が☹ or 巨大なテーブルへの検索性能が☹







ALTER TABLE relation_A ALTER COLUMN col
SET STATISTICS 100;
  or
SET default_statistics_target TO ‘100’;

others

- TIPS
  - パラメータの一部はユーザやDB毎に設定値を与えることが可能です
    - 例1:あるユーザのみ、実施したSQLを全て記録したい
      - ALTER USER some_user SET log_statement TO 'all‘;
    - 例2:あるDBのみ、1分以上かかったSQLを強制的に終了させたい
      - ALTER DATABASE some_db SET statement_timeout TO '60s';
    - 例3:あるDBのみ、最大接続数を20にしたい
      - ALTER DATABASE some_db CONNECTION LIMIT 20;

```
-- 変更の確認例
=# SELECT usename, useconfig FROM pg_user;
 usename   |      useconfig
-----------+---------------------
 some_user | {log_statement=all}

=# SELECT datname, datconnlimit, datconfig FROM pg_database
WHERE datname = 'some_db';
 datname | datconnlimit |        datconfig
---------+--------------+-------------------------
 some_db |           20 | {statement_timeout=60s}
```

others

- TIPS
  - パラメータの一部はSET文で動的に変更することもできます
    - 例1: 管理者ユーザでのセッション時のみmaintenance_work_memを大きくしたい
      - psql some_db
      - SET maintenance_work_mem TO ‘128MB’;
    - 例2: あるトランザクションの時だけ、work_memを大きくしたい
      - psql some_db
      - BEGIN;
      - SET LOCAL work_mem TO ‘128MB’;

- 本日説明できなかった分は、いずれ Let's Postgresやしくみ勉強会で補完していこうと思っています

- ご清聴ありがとうございました！

  Thank you for your attention！

(参考)trace_sort

- trace_sortは、隠し？設定パラメータ
  - postgresql.confに”trace_sort = on”と記述
    - sortの処理内容をログに出力できます

```
LOG:  begin tuple sort: nkeys = 1, workMem = 1024, randomAccess = t
LOG:  switching to external sort: CPU 0.00s/0.00u sec elapsed 0.00 sec
LOG:  performsort starting: CPU 0.01s/0.01u sec elapsed 0.05 sec
LOG:  finished writing run 1: CPU 0.01s/0.01u sec elapsed 0.06 sec
LOG:  finished writing final run 2: CPU 0.02s/0.02u sec elapsed 0.09 sec
LOG:  finished merge step: CPU 0.02s/0.02u sec elapsed 0.10 sec
LOG:  performsort done: CPU 0.02s/0.02u sec elapsed 0.10 sec
LOG:  external sort ended, 103 disk blocks used: CPU 0.05s/0.03u sec elapsed 0.16 sec
```

ディスクの103ブロックを使用：ソート処理のためにwork_memを増やすと良さそう