QUOI DE NEUF DANS POSTGRESQL 18 ?

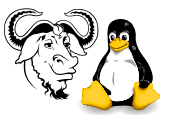
Jean-Christophe Arnu

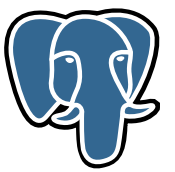
Meetup `pg_ctl restart -D Toulouse`
2025-10-09

ENCHANTÉ, JE ME PRÉSENTE !

JEAN-CHRISTOPHE ARNU

- Création d'une association autour des logiciels libres à Toulouse 1996
- PostgreSQL depuis 1998
- Un des membres fondateurs de PostgreSQLFr
- Organisateur du premier PGDay.fr à Toulouse en 2008
- Consultant PostgreSQL chez LOXODATA





QUOI ?

- 🎁 Ah! Voilà une nouvelle version! Que dois-je faire ?
  - 🗓️ Une version majeure par an ?
  - 🗞️ Faut vraiment lire les release notes ?
  - 🤔 Je mets à jour ou pas ?
- 📦 Venons en au fait : et la version 18 alors ?
  - 🏃‍♀️ Des nouveautés sous le capot ?
  - 👩‍💻 Et pour les developpeu·r·ses ?
  - ✍️ Et pour l'architecture ?

🎁 AH, VOILÀ UNE NOUVELLE VERSION! QUE DOIS-JE FAIRE ?

🗓️ UNE VERSION MAJEURE PAR AN ?

https://postgresql.org/support/versioning/

- Pour les majeures :
  - Vers septembre/octobre
  - 5 versions majeures en parallèle
  - EOL de version majeure à la première mineure de la nouvelle majeure
- Pour les mineures:
  - Versions planifiées tous les 3 mois voir la roadmap
    - 13 novembre 2025
    - 12 février 2026
    - 14 mai 2026
    - 13 août 2026
  - Versions « urgentes » lors de bugs critiques ou CVE

DOIT-ON LIRE LES RELEASE NOTES ?

RÉPONSE RAPIDE : OUI

🗞️ DOIT-ON LIRE LES RELEASE NOTES ?

- Autre réponse : **OUI** mais **PAS QUE**
- Il y a aussi:
  - Les release notes sont longues : Overview / Migration / Changes
  - Tout sur une page, sans exemple: https://www.postgresql.org/docs/18/release-18.html
  - La version « HPe » des nouveautés (environ 100 pages

🤔 DOIS-JE METTRE À JOUR OU PAS ?

> *« Je vais attendre la 18.1 »*

> *« Je vais attendre la version 19 pour installer la 18 »*

- La cycle de développement de PostgreSQL comprend une ou plusieurs Bêta et des Release Candidates.⚙️
- PostgreSQL est déjà bien testé en amont d'une GA 🧰

🤔 DOIS-JE L'UTILISER ?

Oui mais comment ?

Disclaimer:

J'enfonce les portes ouvertes !

Merci Captain Obvious©!

- Intégrer le plus tôt possible le test des bêtas, et RC dans votre CI 😎
- Tester et adapter vos applications maintenant plutôt que plus tard 👍

Et sinon ?

- À trop attendre, on crée :
  - De la dette technique : ne pas profiter des avancées ?😟
  - Des risques inutiles : se prémunier des brêches (CVE) et sécurité😱

🤦 AVANT QUE J'OUBLIE

> *« Ok, je mettrai à jour ma mineure plus tard ! »*

Vraiment ?! 🤯

https://why-upgrade.depesz.com/show?from=17.5&to=17.6

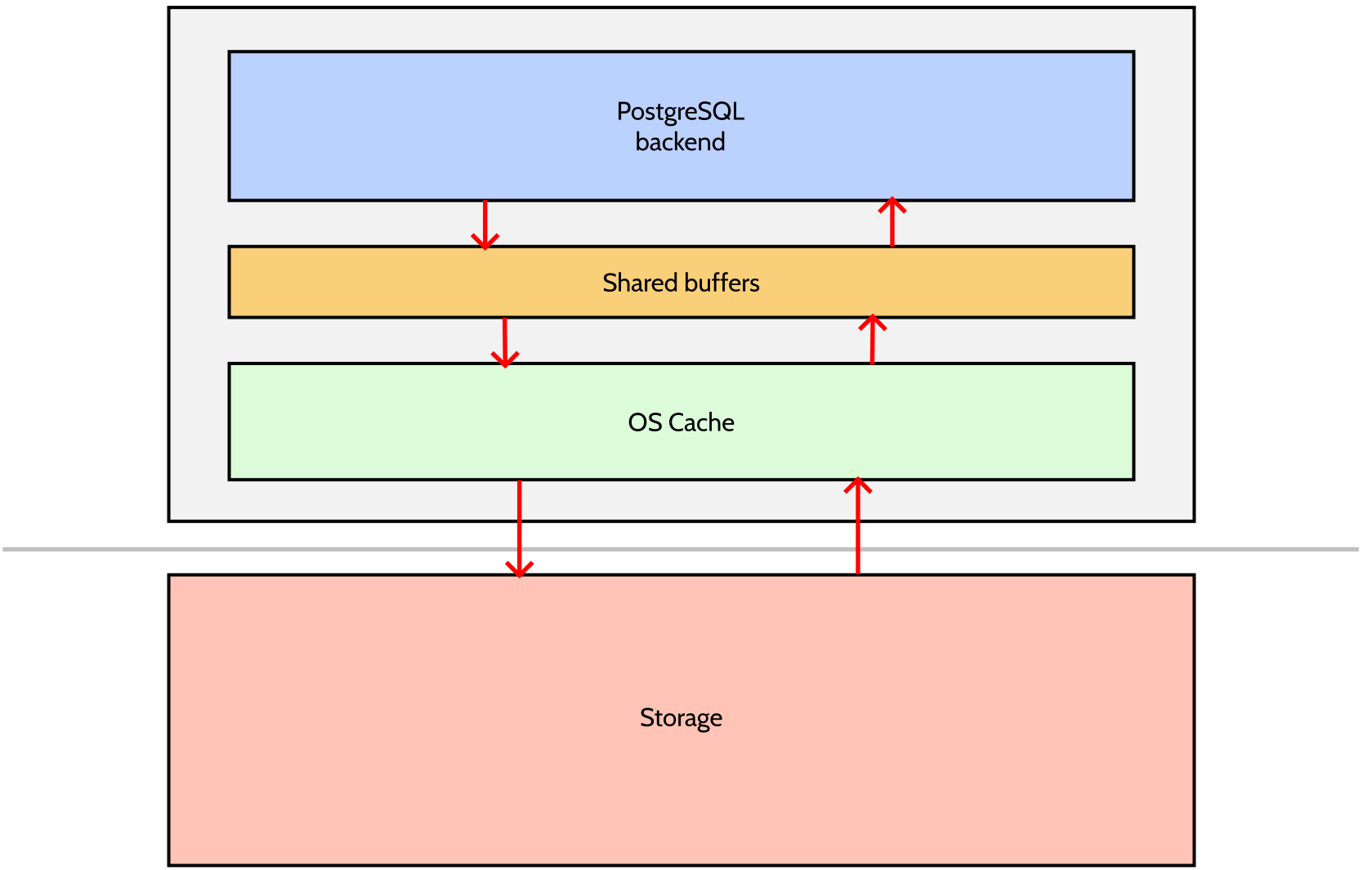
C'est juste un simple redémarrage 🎉🎉🎉

📦 VENONS EN AU FAIT : ET LA VERSION 18 ALORS ?!

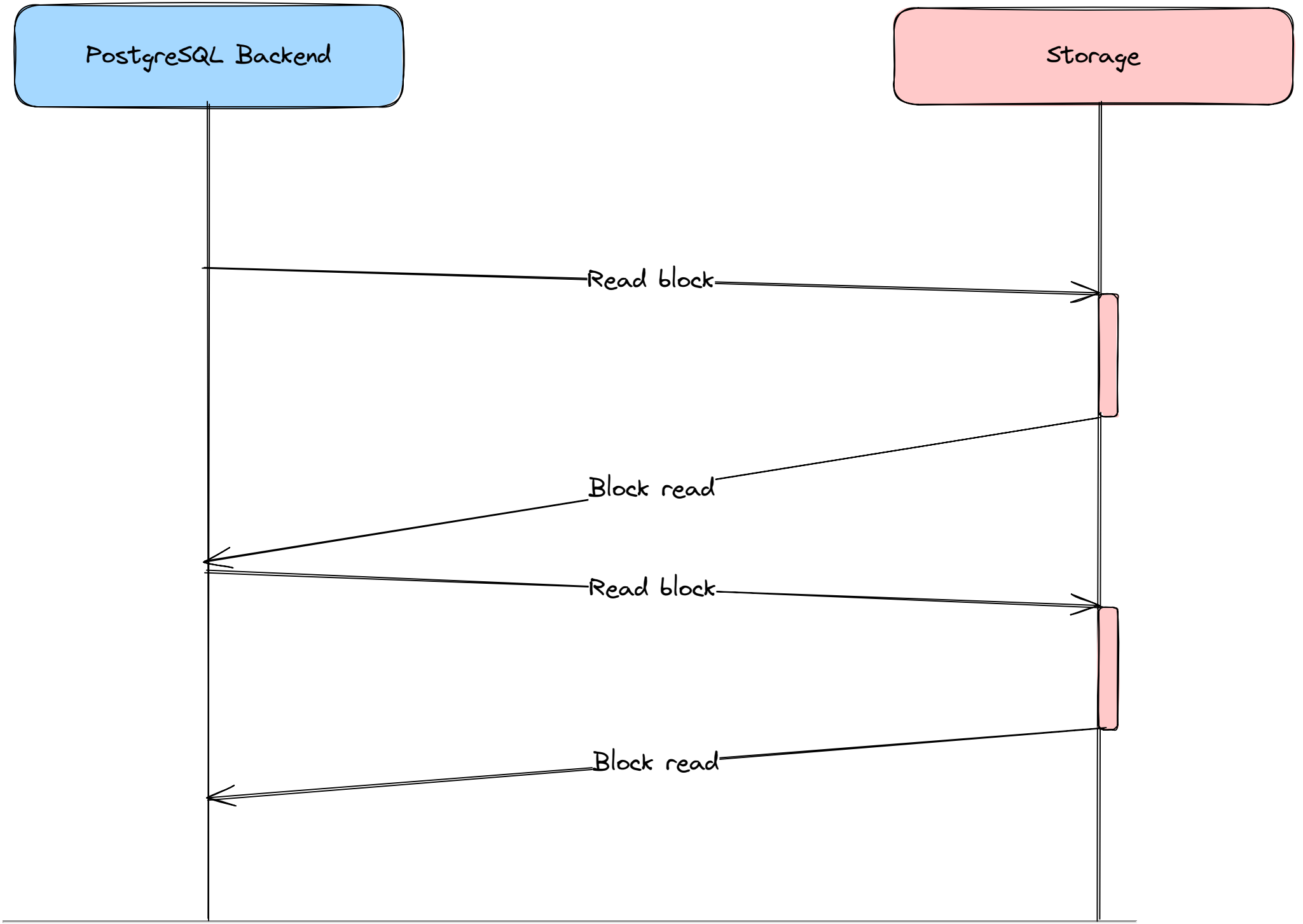
SOUS LE CAPOT ?

ENTRÉES SORTIE ASYNCHRONE

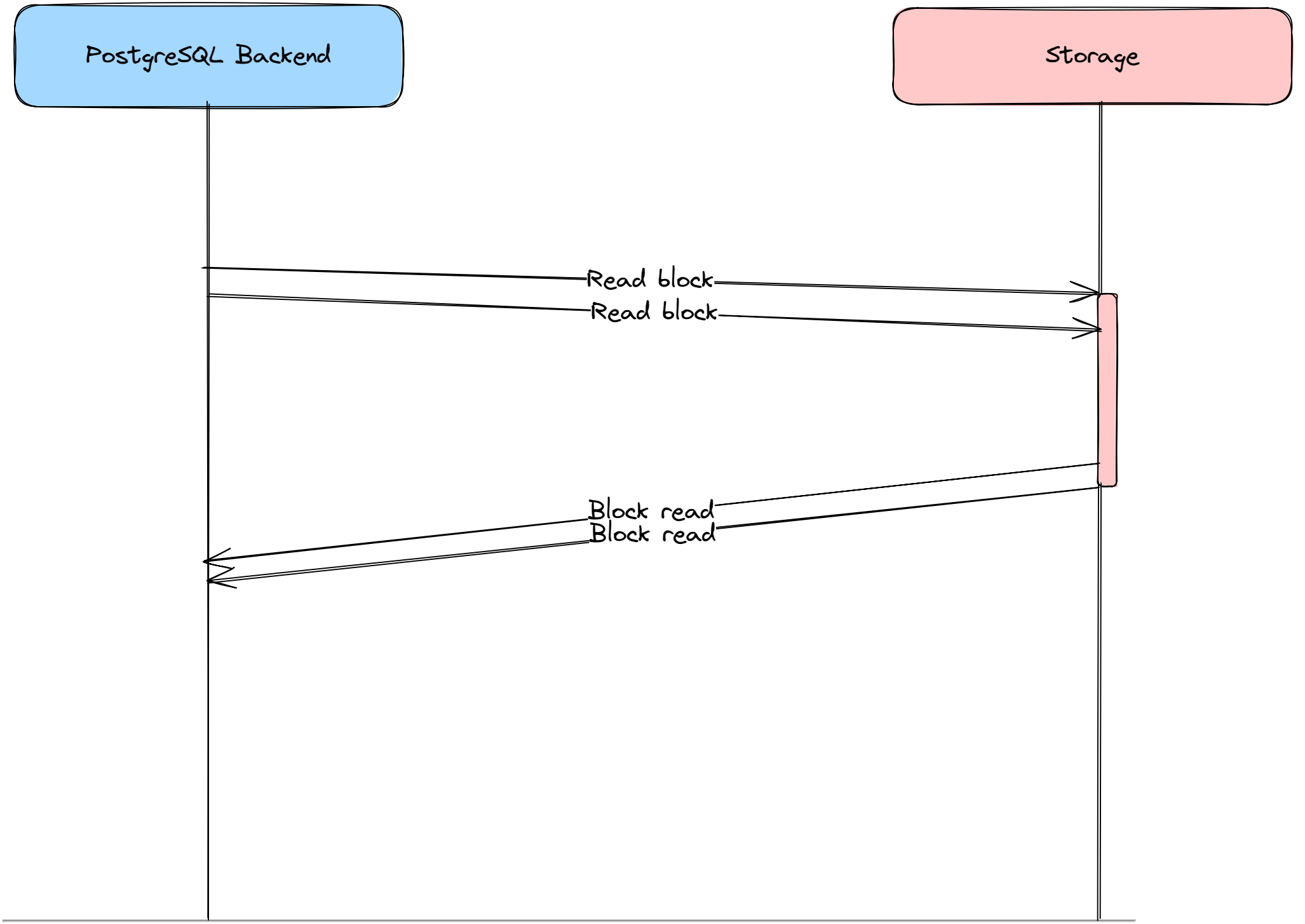
ACCÉDER AU DISQUE



ACCÉDER AU DISQUE : IO SYNCHRONES


PostgreSQL Backend
Storage
Read block
Block read
Read block
Block read

ACCÉDER AU DISQUE : IO ASYNCHRONES


PostgreSQL Backend
Storage
Read block
Read block
Block read
Block read

ACCÉDER AU DISQUE : IO ASYNCHRONES

- Fonctionne en lecture seulement (stay tuned)
- Pour les lectures séquentielles (SeqScan, bitmap, vacuum)
- Trois méthodes dans `io_method` (GUC):
  - `sync` (comme avant),
  - `worker` (défaut),
  - `io_uring` (le plus performant, Linux 5.1+)
- De nouvelles clés de conf pour le tuning (GUC) `io_*`

Bénéficie aux disques
à forte bande passante (locaux)
et/ou
ayant de la latence (network/☁️)

MISES À JOUR - TOUJOURS PLUS RAPIDES

- Sur les mises à jour majeures après `pg_upgrade`
- Sur les grosses bases, c'est l🐌🐌🐌ng de recontruire les statistiques, avec `ANALYZE` !
- Pendant ce temps, la base n'est pas très performante

C'est fini !

Les statistiques sont conservées 🫙
lors d'une mise à jour majeure

Enfin, pas toutes les stats!

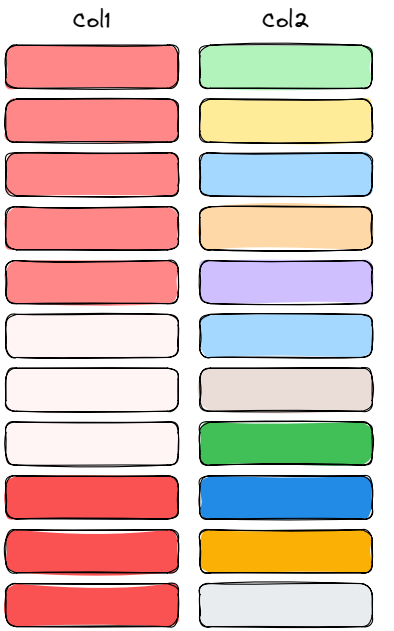
MISES À JOUR - ENCORE ET TOUJOURS PLUS RAPIDES

Quand on a une base contenant beaucoup d'objets :

- Les vérifications sont maintenant parallélisées `--jobs`
- Aller plus vite sur les transferts de fichiers :`--swap`
  - 🚨 `--sync-method=fsync`.
  - Méthode destructive : 🚨 pensez à vos backups !

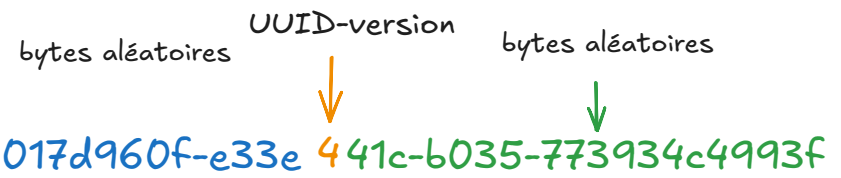
TOUJOURS PLUS VITE : OPTIMIZER !

- Le planner peut faire du **SkipScan** avec les index (btree) multicolonnes ?!
  - Dans l'exemple `WHERE col2=val`
  - Ne fonctionne qu'avec l'opérateur =
  - Se base sur les statistiques: 🚨 cardinalités !
- Amélioration des clauses OR ⇒ index
- Meilleure performance sur les jointures hachage, fusion + tri incrémental



ET POUR LES DÉVELOPPEU·R·SE·S ?

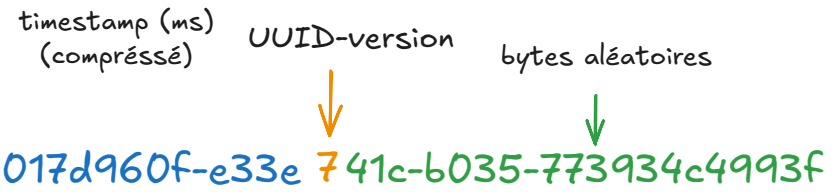
UUID AVANT (V4)



- Populaire pour les clés primaires
  - excellente randomisation
  - risque de collision extrêmement faible
- Indexation difficile
  - Position arbitraire dans l'arbre.
  - Chaque insertion devient une écriture aléatoire
  - Fragmentation de la données

🤔

UUID MAINTENANT (V7)



- Toujours Aléatoire
- Résoud les problèmes de tri et d'indexation.

😎

COLONNES GÉNÉRÉES VIRTUELLES

- Valeurs sont calculées à la requête
- Par défaut pour les colonnes générées >= V18

```
CREATE TABLE products (
    product_id SERIAL PRIMARY KEY,
    base_price DECIMAL(10,2) NOT NULL,
    tax_rate DECIMAL(5,4) DEFAULT 0.0875,
    discount_rate DECIMAL(5,4) DEFAULT 0.00,
    selling_price_v DECIMAL(10,2)
        GENERATED ALWAYS AS (
            base_price * (1 - discount_rate) * (1 + tax_rate)
        ) ,
    selling_price_s DECIMAL(10,2)
        GENERATED ALWAYS AS (
            base_price * (1 - discount_rate) * (1 + tax_rate)
        ) STORED
);
```

LEQUEL CHOISIR ?

VIRTUAL

- Préserver l'espace de stockage
- Insert plus rapide 🐇 (pas de calcul)
- Select plus lent 🐌 (calculs)
- Pour des calculs relativement simples
- Pour des valeurs qui changent fréquemment
- 🚨 Pas d'index sur les valeurs calculées

STORED

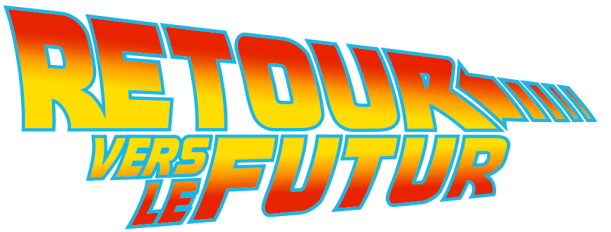
- Indexer les valeurs calculées
- Insert plus lent 🐌 (calculs)
- Select plus rapide 🐇 (pas de calcul)
- Réplication logique des valeurs calculées (>=V18).

Evidemment!

Ça dépend

Merci Captain Obvious©!

RETURING OLD, NEW (INSERT,UPDATE, DELETE, MERGE)




```
1 UPDATE products_stored SET base_price = 1299.00
2 WHERE product_id = 1
3 RETURNING NEW.base_price as new_b_p , NEW.selling_price as new_s_p,
4           OLD.base_price as old_b_p,OLD.selling_price old_s_p ;
5
6  new_b_p | new_s_p | old_b_p | old_s_p
7 ---------+---------+---------+---------
8  1299.00 | 1376.94 | 1298.00 | 1375.88
9 (1 row)
```

EXEMPLES DE CAS D'USAGE,

Detecter si insert or update sur un upsert.

```
INSERT INTO products_stored (product_id, base_price, tax_rate, discount_rate)
VALUES (1,1297.00,0.0600,0.0000)
ON CONFLICT (product_id)
    DO UPDATE SET base_price = excluded.base_price
RETURNING products_stored.*,
    (OLD IS NULL)::boolean AS is_new;

 product_id | base_price | tax_rate | discount_rate | selling_price | is_new
------------+------------+----------+---------------+---------------+----------
          1 |       1297 |   0.0600 |        0.0000 |       1376.94 | f
(1 row)
```

CONTRAINTES SUR INTERVALLES

- `PRIMARY KEY` et `UNIQUE` avec la clause `WITHOUT OVERLAPS`
- `FOREIGN KEY` en utilisant la clause `PERIOD`.
- Nécessite l'extension `btree_gist` si d'autres types qu'un range
- Exemple

```
      Table "public.rents"
   Column   |   Type    | Collation | Nullable |             Default
------------+-----------+-----------+----------+-------------------------------------
 id         | integer   |           | not null | generated by default as identity
 rent_dates | tstzrange |           |          |
 bike_id    | integer   |           |          |
Indexes:
    "rents_pkey" PRIMARY KEY, btree (id)
    "rents_bike_id_rent_dates_key" UNIQUE (bike_id, rent_dates WITHOUT OVERLAPS)

INSERT INTO rents (rent_dates , bike_id)
VALUES ('(2025-10-02 08:00,2025-10-03 08:00]':: tstzrange, 1);
INSERT 0 1

INSERT INTO rents (rent_dates , bike_id)
VALUES ('(2025-10-02 17:00,2025-10-02 19:00]':: tstzrange, 1);

ERROR:  conflicting key value violates exclusion constraint "rents_bike_id_rent_dates_key"

DETAIL:  Key (bike_id, rent_dates)=(1, ("2025-10-02 17:00:00+02","2025-10-02 19:00:00+02"])
conflicts with existing key
(bike_id, rent_dates)=(1, ("2025-10-02 08:00:00+02","2025-10-03 08:00:00+02"]).
```

✍️ ET POUR L'ARCHITECTURE ?

S'INTÉGRER AU SI ?

- S'interfacer avec service de gestion de l'authentification :
  - Nouvelle méthode OAuth v2.0 SASL dans `pg_hba.conf`
  - Plus de mot de passe dans la base
  - Autorise la MFA (dépend du gestionnaire)
  - Extension: un validateur (GUC `oauth_validator_libraries`)

SÉCURITÉ

- 🚨 Les mots de passe `md5` sont dépréciés ⇒ passer à `scram-sha-256`
- 🔐 Authentification SCRAM passthrough sur serveurs distants (`FOREIGN SERVER`) pour les Foreign Data Wrappers
- 🔏 Support de la configuration des ciphers acceptés pour TLS v1.3
- 🔐 Support de `sha-2` dans pg_crypto

RÉPLICATION

- Les erreurs d'écriture en **réplication logique**
  - dans le log de PostgreSQL 📰
  - dans la vue `pg_stat_subscription_stats` 🔍
- Le streaming logique est maintenant `parallel` par défaut
- Créer un souscripteur pour toutes les bases : `pg_createsubscriber --all`
- Détecter et supprimer les slots idle : GUC `idle_replication_slot_timeout` 🚨 `checkpoint_timeout`

MAINTENANCE ET OBSERVABILITÉ

- Éviter les `VACUUM FREEZE` : `VACUUM` plus agressif sur le gel des pages (tunnel de gel)
- `EXPLAIN ANALYZE` affiche les buffers par défaut, combien d'index lookup.
- `EXPLAIN ANALYZE VERBOSE` affiche maintenant les informations de temps CPU, de WAL et les statistiques de lecture
- Vue `pg_stat_all_tables : ajout de`
  - `total_(vacuum|analyze)_time en ms`
  - `total_auto_(vacuum|analyze)_time en ms`

👋 MERCI !

Il y aurait encore beaucoup de choses à dire sur cette version...

⌛

mais nous n'avions que 30 minutes!

Merci pour votre attention!

🙏 Merci à Geoffrey, Akram, Florian et toute l'équipe Pictarine 🙏

(l'organisation, le sponsoring, le lieu, c'est eux!)