PostgresPro

Advanced Queuing

Транзакционные очереди

Василий Бернштейн,

28 января 2025 года

PGProDay

Краткое введение в виды очередей

Log-based message broker

- Предназначены для передачи потока данных в строгом порядке следования
- Подходят для реализации синхронизации хранилищ данных
- Запись в очередь и чтение из неё идут строго в одном и том же порядке

Примеры:

- Apache Kafka
- Amazon Kinesis Steams
- Twitter DistributedLog

AMQP/JMS-style message broker

- Предназначены для передачи независимых блоков информации
- Подходят для реализации асинхронного RPC
- Запись в очередь и чтение из неё могут идти в разном порядке в зависимости от приоритетов сообщений, фильтров сообщений и других параметров

Примеры:

- RabbitMQ
- Java Message Service
- ActiveMQ / Amazon MQ
- Oracle AQ

Почему очередь в базе данных?

Администрирование

- Для Kafka / AMQP нужна отдельная инсталляция, отдельное администрирование
- Админы очередей в Enterprise обычно являются отдельной командой – усложняется поддержка приложений
- Сервер приложений должен держать соединения не только с DBMS, но и с сервером очередей – выше вероятность сетевых сбоев
- Для приложений на PL/pgSQL возникает заметный штраф на взаимодействие с очередями по сети

Транзакционность

- Надо гарантировать успешное завершение обработки сообщения
- Для любой внешней системы очередей это возможно только при использовании 2PC (Two-phase commit)
  - Требует ещё и дополнительного сервера-координатора 2PC
- Выполнение обработки сообщения (запросы в DBMS) в той же транзакции, что и чтение сообщения
  - Гарантирует, что сообщение будет удалено только при успешной его обработке
  - Это верно как для обработчиков на PL/pgSQL, так и любых других, действующих в рамках одной транзакции DBMS

Postgres – существующие реализации

Log-based message broker

PGQ (Open Source)

- https://github.com/pgq/pgq
- Высокая скорость
- Высокая надёжность
- Создавался в Skype для синхронизации баз данных разных регионов
- Проверенная технология уровня Enterprise

AMQP/JMS-style message broker

PGMQ (Open Source)

- https://github.com/tembo-io/pgmq
- Ограниченные возможности по фильтрации, dead letter queue и delayed processing
- Нет автоматического Retry on Rollback, Consume on Commit

Enterprise DB Advanced Queuing (Private Code)

- DBMS_AQ
- DBMS_AQADM
- Поддержка фильтрации, dead letter queue, delayed processing
- Автоматический Retry on Rollback, Consume on Commit

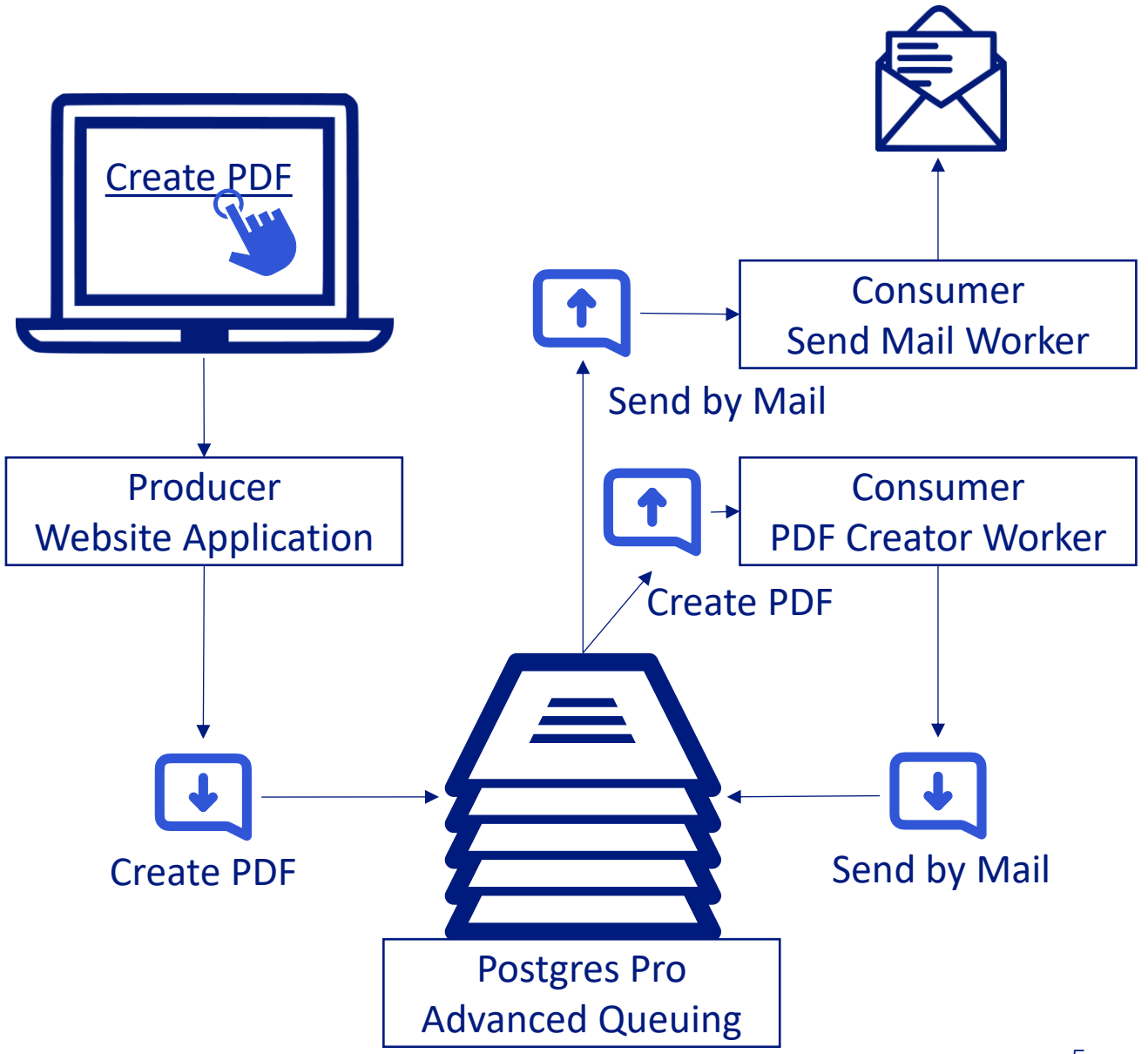
Чего не хватает?

AMQP/JMS-style message broker

- Поддержка фильтрации, dead letter queue, delayed processing
- Подходит для реализации асинхронной распределённой бизнес-логики
- Автоматический Retry on Rollback, Consume on Commit

Пример:

- Пользователь запрашивает создание отчёта с передачей по e-mail, исходные данные будут в 0:00 завтра
- Web App посылает сообщение с запросом на обработку не ранее 0:00 завтра
- PDF Creator Worker получает сообщение, когда данные готовы, создаёт отчёт, посылает сообщение с запросом на пересылку по e-mail
- Send Mail Worker получает сообщение, пересылает сгенерированный ранее отчёт по e-mail через SMTP сервер. Если сервер недоступен, повторяет попытку через определённое время



В чём отличие от pgpro_scheduler one-time jobs?

pgpro_scheduler

- Для выполнения регулярных операций
  - One-time jobs – редкий способ использования, служит для запуска регулярных операций в нестандартное время
  - Передача информации не является задачей one-time jobs
- Автоматический Retry после ошибки не является требованием
  - Предполагается анализ причин неуспешного завершения заданий перед их перезапуском

pgpro_queue

- Для передачи и запросов, событий и информации от одного процесса к другому
- Retry после ошибки является предусмотренным нормальным случаем
  - ошибки из-за рассинхронизации и неготовности данных
  - ошибки из-за недоступности других сервисов
  - и т.д.

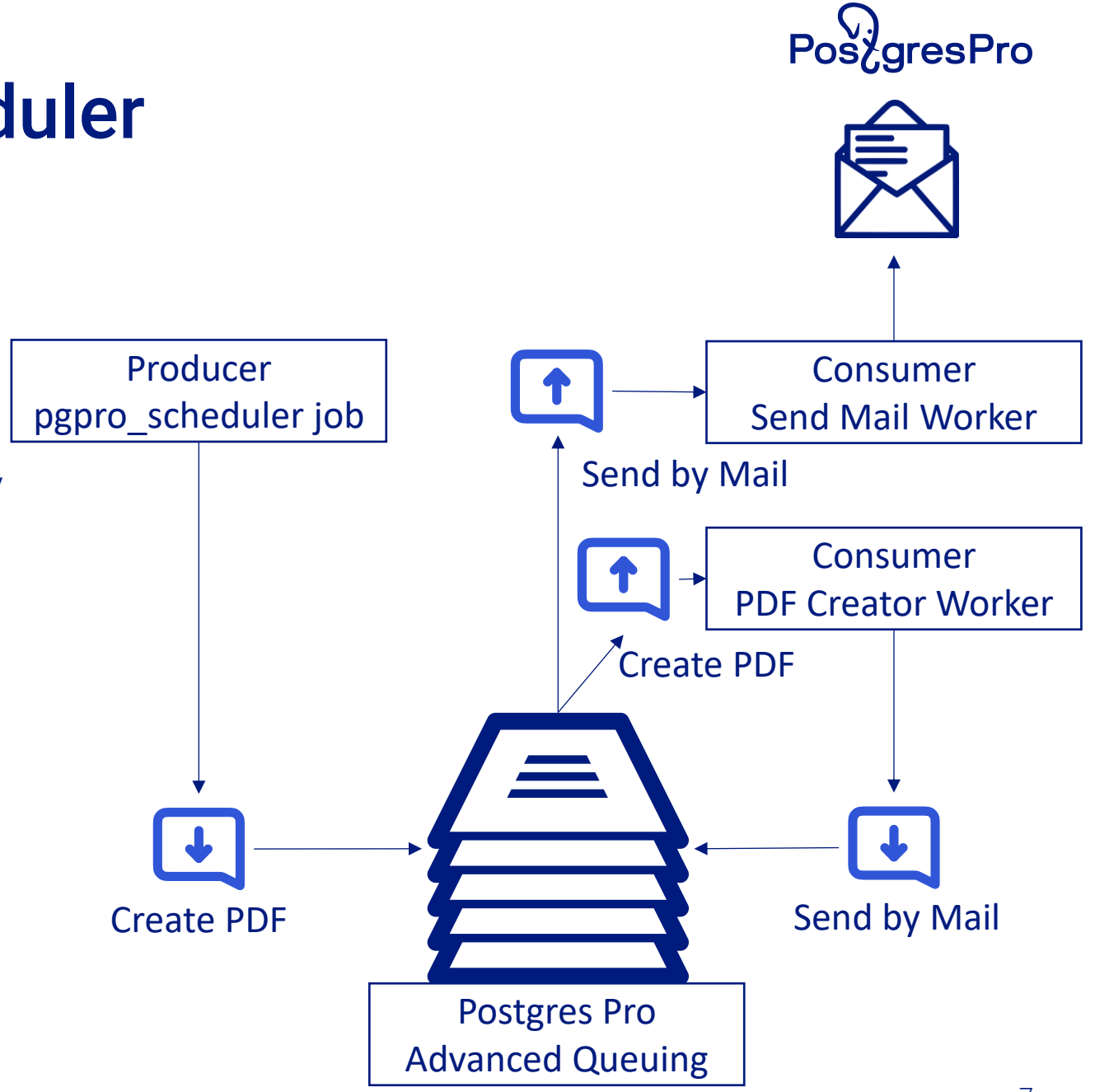
pgpro_queue + pgpro_scheduler

pgpro_scheduler

- Запускает регулярные задачи
- Пример: генерация выписок для клиентов
  - Для каждого из отобранных задачей клиентов через pgpro_queue требует сгенерировать выписку
  - Сбой может быть вызван серьёзной ошибкой в DB, требует изучения перед перезапуском

pgpro_queue

- Гарантирует обработку индивидуального задания некоей задачи или процесса
- Пример: генерация выписки для одного клиента
  - Для указанного клиента создаёт выписку, затем посылает её по e-mail
  - Сбой может быть вызван временной недоступностью сервера e-mail, нужен автоматический повтор



Postgres Pro Advanced Queuing

Реализованы в первой версии

- **Автоматический Retry on Rollback**
- Передача сообщений point-to-point
- Поддержка в сообщениях форматов JSON и XML
- Именованные очереди
- Приоритеты сообщений
- Expiry Date сообщений
- Фильтрация сообщений

Дальнейшее развитие

- Dead Letter Queue (Exception Queue)
- Передача сообщений списку подписчиков
- Система общих или закрытых подписок
- Expiry Date подписок
- Вызов Callback у получателей сообщений

Автоматический Retry on Rollback

Значительное упрощение кода обработчиков сообщений

- Транзакционность обработки сообщений
- Автоматический повтор обработки сообщения при возникновении ошибки (rollback)
- Не нужна сложная логика обработки Exceptions внутри обработчиков сообщений
- Поддержка вложенных транзакций
- Поддержка автономных транзакций

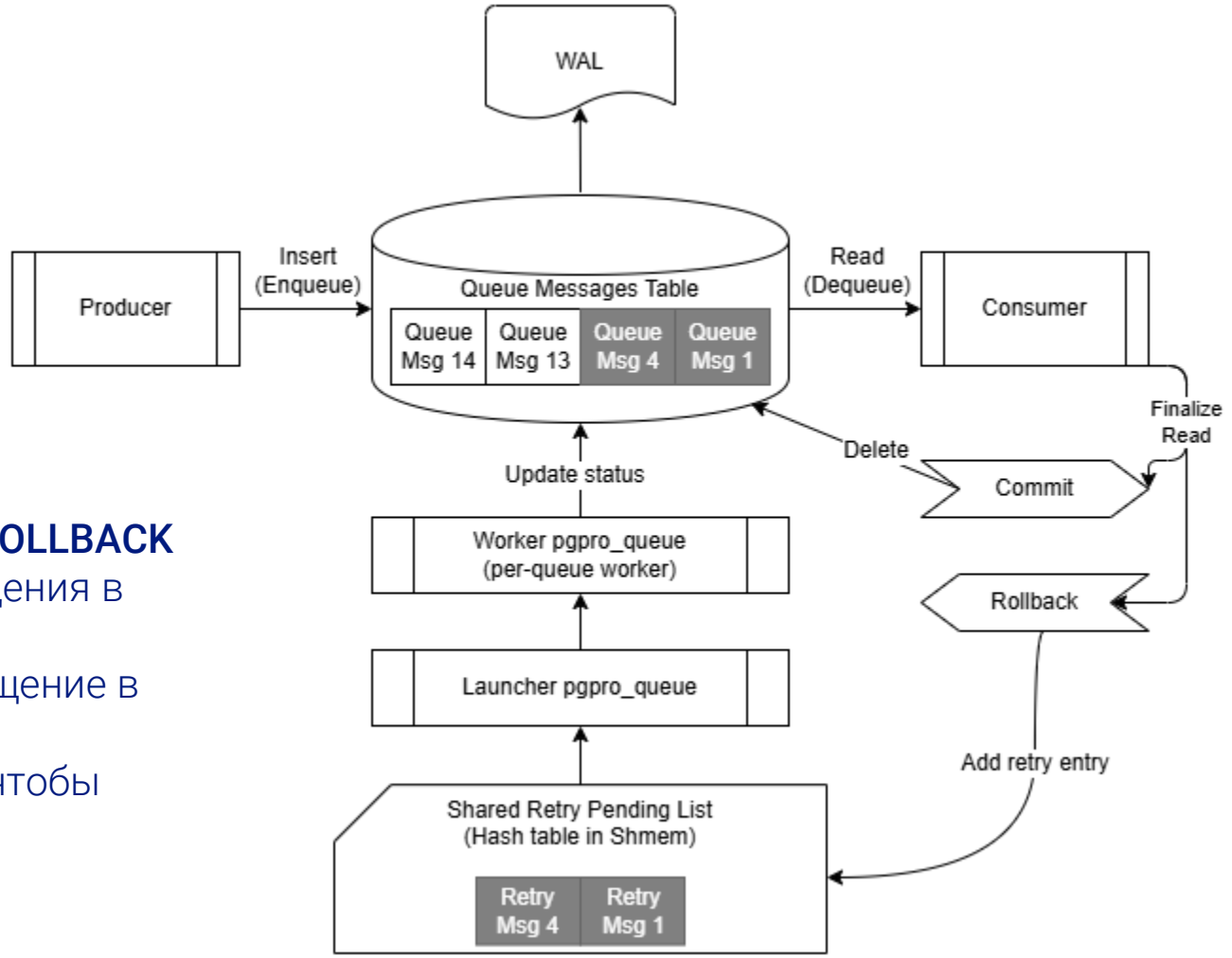
```
CREATE PROCEDURE send_PDF_by_email(
  PDF sfile,
  sendto text)
AS $$
BEGIN
  BEGIN AUTONOMOUS TRANSACTION;
  perform log_writer(concat('Sending PDF via email for ', sendto);
  COMMIT AUTONOMOUS TRANSACTION;
  conn := utl_smtp.open_connection('smtp.mail.ru', 25, 10);
  perform utl_smtp.auth(conn, 'test_email@example.com', 'super-secret-password');
  perform utl_smtp.mail(conn, 'sender@example.com');
  perform utl_smtp.rcpt(conn, sendto);
  perform utl_smtp.open_data(conn);
  perform utl_smtp.write_data(conn, E'From: Sender <sender@example.com>\n');
  perform utl_smtp.write_data(conn, E'To: Recipient <recipient@example.com>\n');
  perform utl_smtp.write_data(conn, E'Subject: PDF Report\n');

  perform utl_smtp.write_data(conn, encode_binary( PDF));

  perform utl_smtp.write_data(conn, E'Sent using utl_smtp\n');
  perform utl_smtp.close_data(conn);
  perform utl_smtp.quit(conn);
END$$;
```

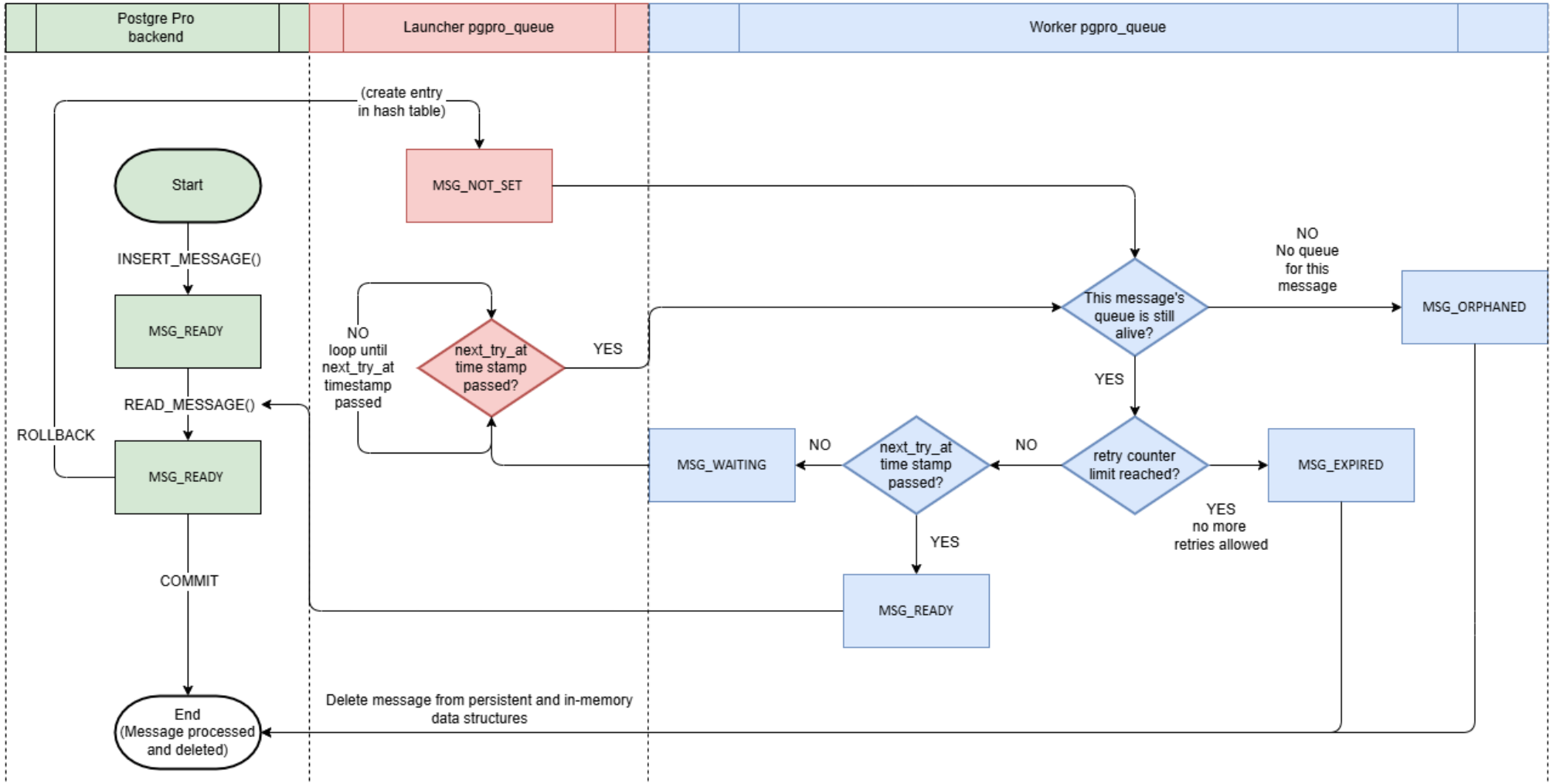
Архитектура – общий вид

- **Хранение очереди в таблице heap**
  - Persistence в relations
  - Синхронизация со Standby через WAL
  - Гарантия восстановления при сбое



- **Launcher & Workers, следящие за COMMIT/ROLLBACK**
  - Автоматически «замораживают» сообщения в случае Rollback
  - Отслеживают, когда пора вернуть сообщение в обработку
  - Хранят список retry сообщений в кэше, чтобы обеспечить транзакционность при ROLLBACK

Архитектура – Workflow сообщений



Полезные ссылки

- pgpro_queue – управление очередями сообщений
  https://postgrespro.ru/docs/enterprise/17/pgpro-queue

- Высоконагруженные приложения. Программирование, масштабирование, поддержка. Клеппман Мартин.
  https://www.ozon.ru/product/vysokonagruzhennye-prilozheniya-programmirovanie-masshtabirovanie-podderzhka-kleppman-martin-1346808746/

PostgresPro

Спасибо за внимание!