Logical Replication of DDLs

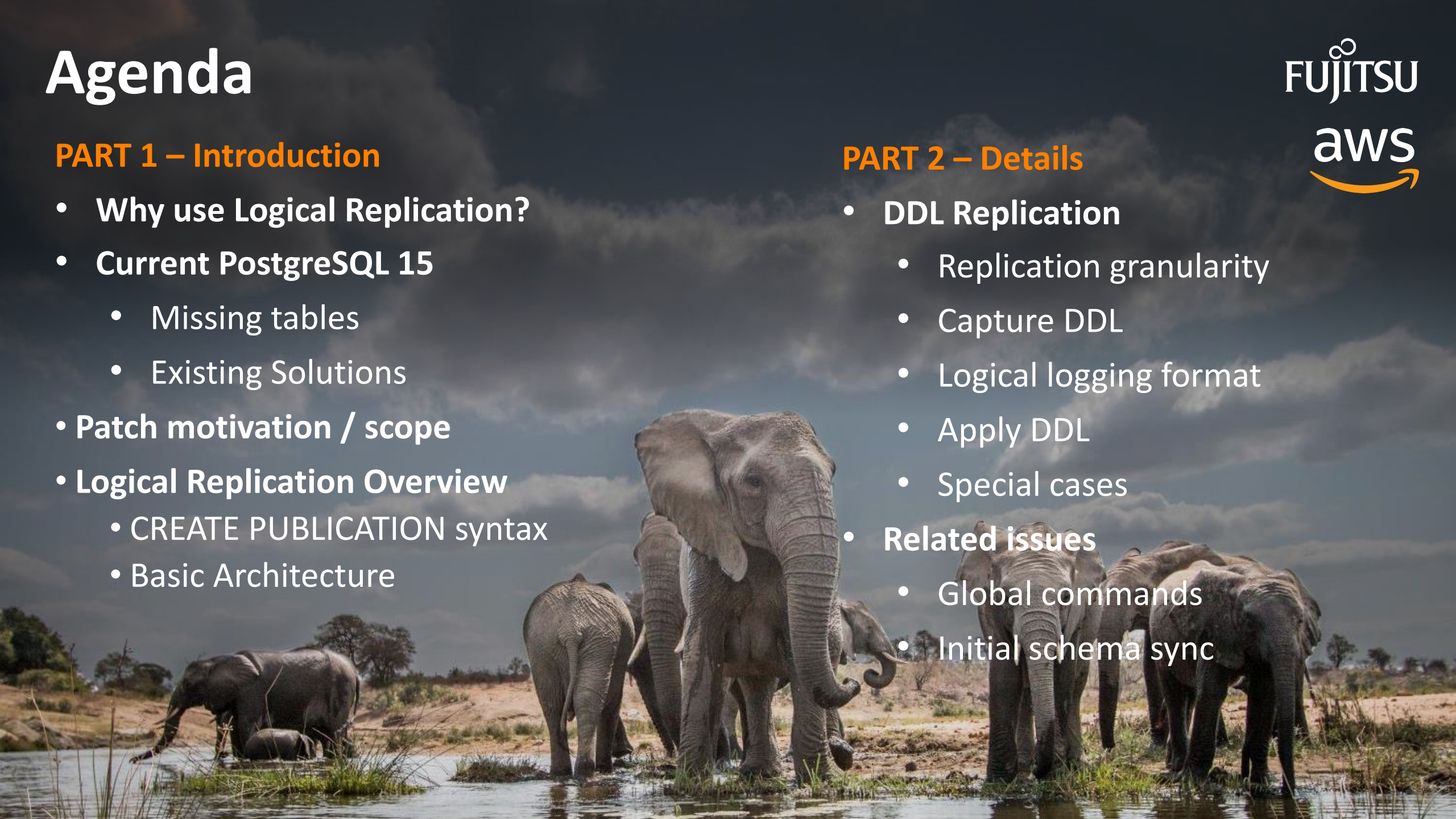
Peter Smith

Fujitsu

Zheng Li (Zane)

Amazon RDS Open Source

Agenda



FUJITSU

aws

PART 1 – Introduction

- **Why use Logical Replication?**
- **Current PostgreSQL 15**
  - Missing tables
  - Existing Solutions
- **Patch motivation / scope**
- **Logical Replication Overview**
  - CREATE PUBLICATION syntax
  - Basic Architecture

PART 2 – Details

- **DDL Replication**
  - Replication granularity
  - Capture DDL
  - Logical logging format
  - Apply DDL
  - Special cases
- **Related issues**
  - Global commands
  - Initial schema sync

Terms

FUJITSU

- **Logical Replication**
  - A method of *logically* replicating data changes from one node ("publisher") to another node ("subscriber").
  - See PostgreSQL CREATE PUBLICATION / SUBSCRIPTION

- **DDL -- Data Definition Language**
  - Subset of SQL, used for defining and managing the structure of a database
  - e.g. **CREATE / ALTER / DROP** a database object (TABLE, INDEX, etc.)

- **DML -- Data Manipulation Language**
  - Subset of SQL, used to manipulate and query data in a database
  - e.g. INSERT, UPDATE, DELETE

Why use Logical Replication?

FUJITSU

- **Physical Replication** - An exact binary copy from one node to another
- **Logical Replication** - A publish/subscribe model that sends “replication messages” to transfer incremental information from one node to another
  - Replicate between different major versions of PostgreSQL
  - Replicate between PostgreSQL instances running on different platforms
  - Share a subset of the database between multiple database servers
  - Distribute changes from a single publication to multiple subscribers
  - Built-in logical replication doesn’t replicate DDLs

    Schema changes need to be replicated manually on the subscription database, causing downtime

Logical Replication of DDLs in PostgreSQL 15

FUJITSU

- PG Documentation: 31.2
  - *The schema definitions are not replicated, and the published tables must exist on the subscriber.*
  - *The tables are matched between the publisher and the subscriber using the fully qualified table name. Replication to differently-named tables on the subscriber is not supported.*

- NOTE: Attempting to replicate to a missing subscriber-side table will cause a runtime error.

Example 1 – Missing table at subscription creation

FUJITSU

Subscriber-side table employee does not exist, when the CREATE SUBSCRIPTION is executed

T1

```
test_pub=# CREATE TABLE employee(id int, name text, PRIMARY KEY(id));
CREATE TABLE
test_pub=# CREATE PUBLICATION pub_all FOR ALL TABLES;
CREATE PUBLICATION
```

T2

```
test_sub=# CREATE SUBSCRIPTION mysub CONNECTION 'dbname=test_pub' PUBLICATION pub_all;
ERROR:  relation "public.employee" does not exist
```

Need initial schema sync!

Time

Example 2 – Replication error due to missing table

FUJITSU

Subscriber-side table employee does not exist, after the subscription is already created

T1

```
test_pub=# CREATE PUBLICATION pub_all FOR ALL TABLES;
CREATE PUBLICATION
```

T2

```
test_sub=# CREATE SUBSCRIPTION mysub CONNECTION 'dbname=test_pub' PUBLICATION pub_all;
NOTICE:  created replication slot "mysub" on publisher
CREATE SUBSCRIPTION
```

WITH (disable_on_error)

T3

```
test_pub=# CREATE TABLE employee(id int, name text, PRIMARY KEY(id));
CREATE TABLE
test_pub=# INSERT INTO employee VALUES (1, 'Fred'), (2, 'Barney');
INSERT 0 2
```

Need DDL replication!

Subscriber log file

T4

```
2023-05-02 11:36:16.977 AEST [15335] LOG:  logical replication apply worker for subscription "mysub" has started
2023-05-02 11:38:15.739 AEST [15335] ERROR:  logical replication target relation "public.employee" does not exist
2023-05-02 11:38:15.739 AEST [15335] CONTEXT:  processing remote data for replication origin "pg_16388" during message type
"INSERT" in transaction 744, finished at 0/1914DF0
2023-05-02 11:38:15.740 AEST [14725] LOG:  background worker "logical replication worker" (PID 15335) exited with exit code 1
2023-05-02 11:38:15.744 AEST [15354] LOG:  logical replication apply worker for subscription "mysub" has started
2023-05-02 11:38:15.753 AEST [15354] ERROR:  logical replication target relation "public.employee" does not exist
2023-05-02 11:38:15.753 AEST [15354] CONTEXT:  processing remote data for replication origin "pg_16388" during message type
"INSERT" in transaction 744, finished at 0/1914DF0
2023-05-02 11:38:15.754 AEST [14725] LOG:  background worker "logical replication worker" (PID 15354) exited with exit code 1
2023-05-02 11:38:20.752 AEST [15356] LOG:  logical replication apply worker for subscription "mysub" has started
2023-05-02 11:38:20.763 AEST [15356] ERROR:  logical replication target relation "public.employee" does not exist
```

Time

Existing solutions for missing tables

FUJITSU

- If there is no interest in the missing table, maybe use a different PUBLICATION

- If the mismatched table is due only to column differences, maybe use a PUBLICATION with Column Lists

```
test_pub=# CREATE PUBLICATION mypub FOR TABLE employee (id, name);
CREATE PUBLICATION
```

- Manually CREATE TABLE the missing tables

- Use the **pg_dump** tool to dump publisher table commands to a file, then execute on the subscriber-side

```
pg_dump --schema=myschema test_pub > db.sql

test_sub=# \i db.sql;
```

Maintaining publisher/subscriber table consistency

FUJITSU

NOTE: It is difficult to maintain consistency when the publisher-tables may be changing.

T1

```
test_sub=# CREATE TABLE employee(id int, name text, PRIMARY KEY(id));
CREATE TABLE
```

T2

```
test_pub=# CREATE TABLE employee(id int, name text, PRIMARY KEY(id));
CREATE TABLE
test_pub=# INSERT INTO employee VALUES (1, 'Fred');
INSERT 0 1
```

T3

Subscriber-side table already exists, so replication is OK

T4

```
test_pub=# ALTER TABLE employee ADD age int;
ALTER TABLE
test_pub=# INSERT INTO employee VALUES (2, 'Barney', 27);
INSERT 0 1
```

T5

Subscriber log file

```
2023-05-12 12:43:11.921 AEST [32478]
ERROR:  logical replication target relation "public.employee" is
missing replicated column: "age"
```

Time

Patch – Motivation and Status

FUJITSU

- DDL replication can reduce the need for user-action
- DDL replication can provide a means for schema-mapping
- Patches
  - Please find the discussion and suite of patches in the **pgsql-hackers** thread
    -- ***Support logical replication of DDLs***
  - The scope of this work is currently limited to just DDL replication of TABLES and INDEXES, but in future more objects can be replicated
  - NOTE: This is ongoing development. Some details may already be outdated

CREATE PUBLICATION – new parameter

FUJITSU

The CREATE PUBLICATION syntax is unchanged but there is now a new parameter **'ddl'** to tell the PUBLICATION what kinds of objects will have their DDL published.

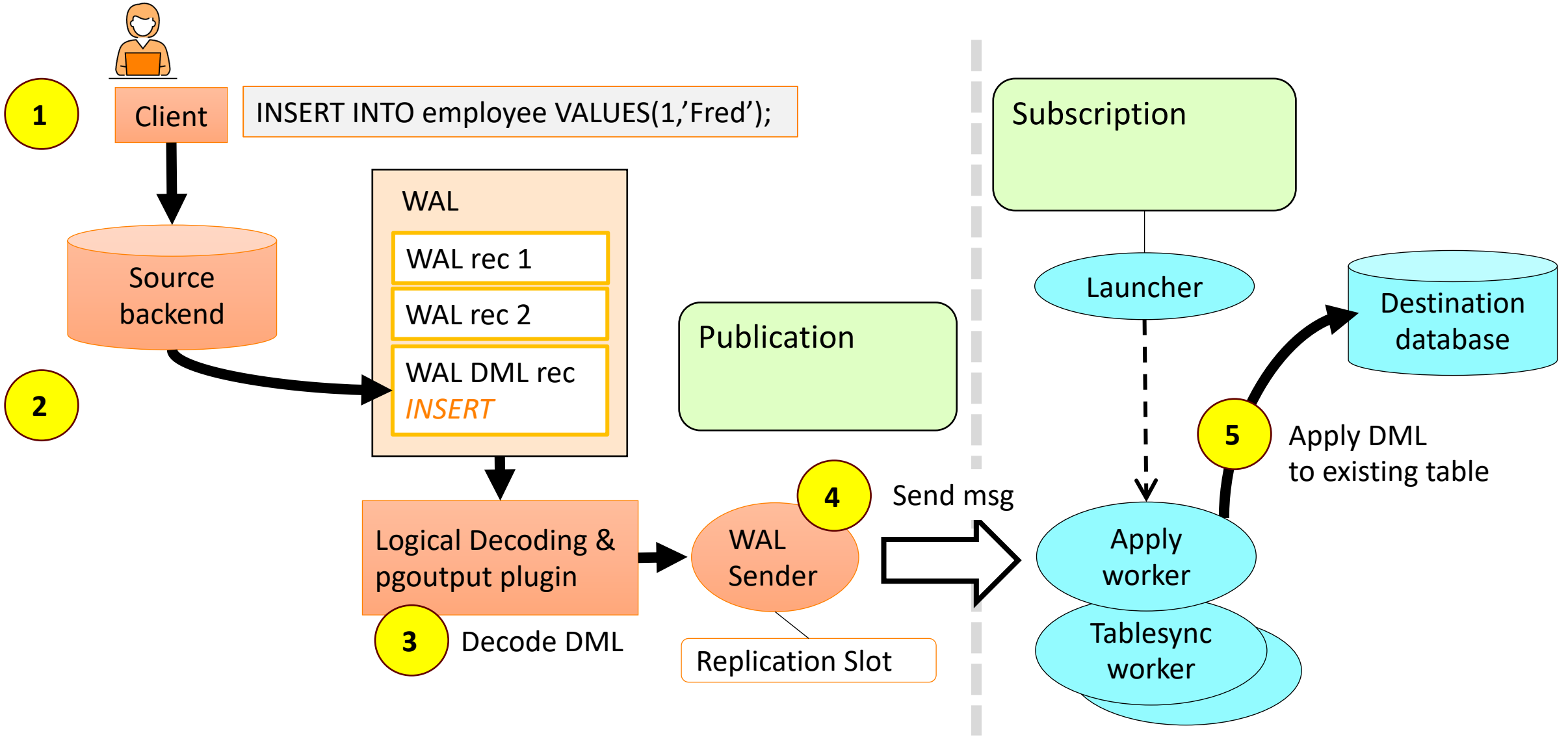
```
CREATE PUBLICATION mypub FOR ALL TABLES WITH (ddl = 'table');
```

```
CREATE PUBLICATION mypub FOR ALL TABLES WITH (ddl = 'table, index');
```

- This allows DDL publish operations CREATE/ALTER/DROP for the specified kinds of objects
- The **default** is no DDL replication, which is just same as PG15
- Various other parameter values are also being discussed. More details later.

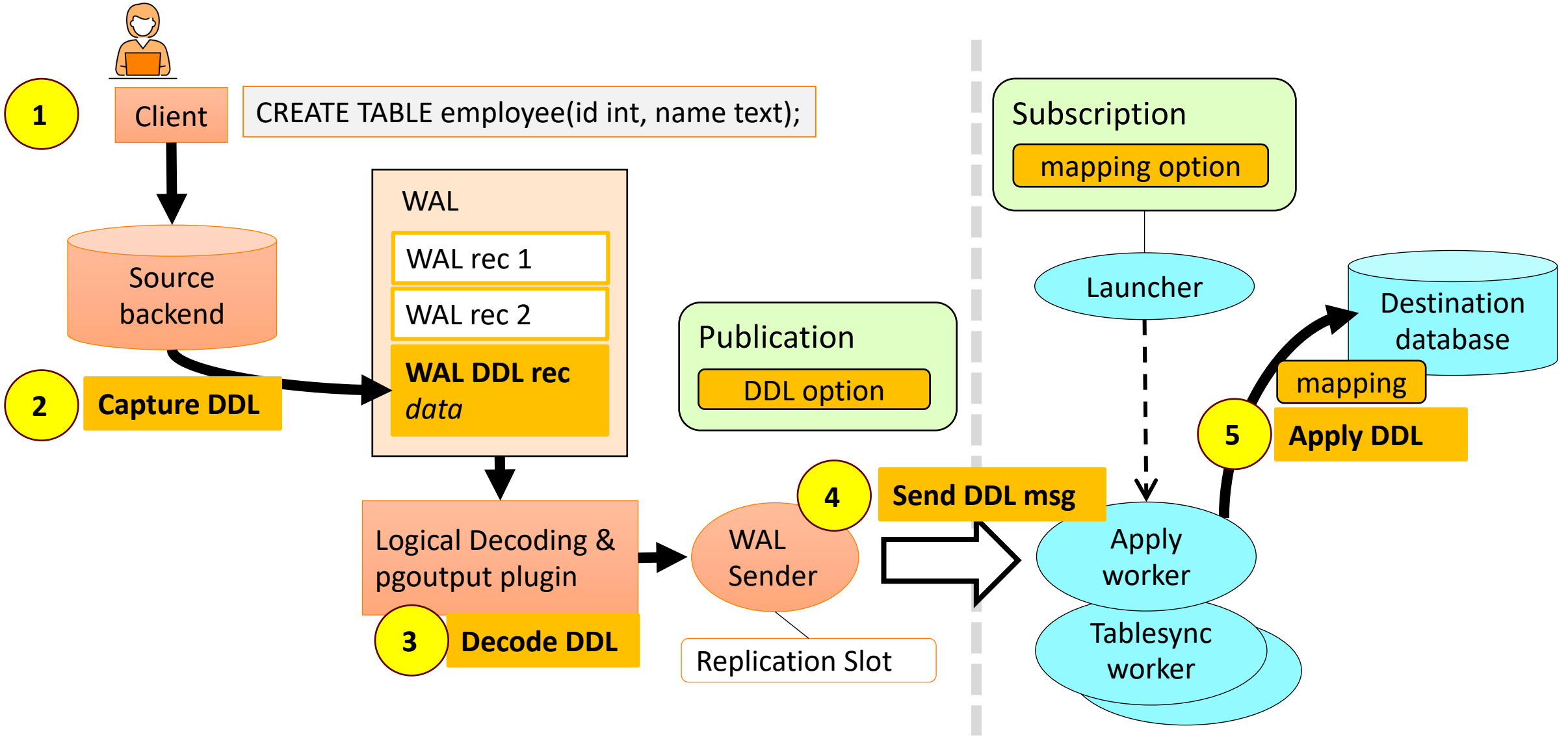
PostgreSQL Logical Replication

FUJITSU

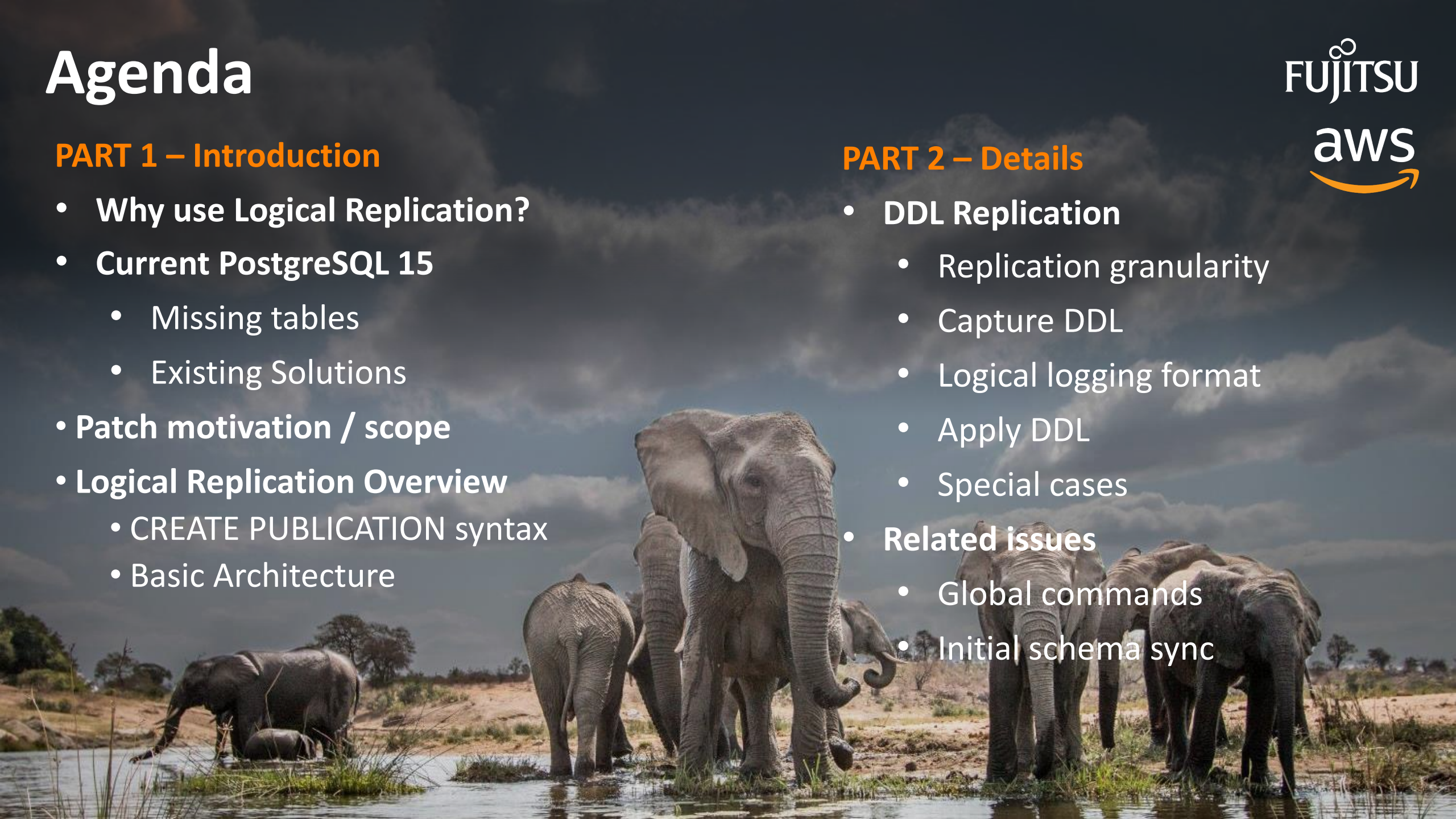


PostgreSQL Logical Replication + DDL support (overview)

FUJITSU



Agenda



FUJITSU

aws

PART 1 – Introduction

- **Why use Logical Replication?**
- **Current PostgreSQL 15**
  - Missing tables
  - Existing Solutions
- **Patch motivation / scope**
- **Logical Replication Overview**
  - CREATE PUBLICATION syntax
  - Basic Architecture

PART 2 – Details

- **DDL Replication**
  - Replication granularity
  - Capture DDL
  - Logical logging format
  - Apply DDL
  - Special cases
- **Related issues**
  - Global commands
  - Initial schema sync

Use Cases of Logical Replication of DDL

- Major version upgrade
  - Replicate all/most DDL
  - Auto-fix DDL syntax incompatibility
- Migrate multiple databases/subset of a database into one database
  - Only replicate certain DDLs
  - One desired feature is schema/name mapping
- Heterogeneous replication
  - OLTP -> OLAP
  - Structured representation facilitates heterogeneous replication

DDL Option Defines Replication Granularity

- Allow fine-grained DDL replication granularity
  - CREATE PUBLICATION mypub FOR ALL TABLES WITH (ddl = 'table, index’);
  - FOR pub_all_func WITH (ddl = 'function’);
  - FOR pub_create_trigger WITH (ddl = 'trigger’);

- Develop the full feature in multiple stages based on the replication granularity

Capture DDL

- Inline (ProcessUtilitySlow)
  - Captures all or any subset of DDLs
  - Small amount of code change

- Event Triggers
  - Existing mechanism to capture DDLs
  - Event trigger is only supported on a subset of DDLs, need to expand on the current event trigger support

Capture DDL with Event Triggers

```
source_db=# CREATE PUBLICATION mypub FOR ALL TABLES with (ddl = 'table');
CREATE PUBLICATION
source_db=# SELECT evtname, evtevent, evttags from pg_event_trigger;
                evtname                |     evtevent      |                  evttags
---------------------------------------+-------------------+-------------------------------------------
 pg_deparse_trig_table_init_write_16429 | table_init_write  | {"CREATE TABLE AS","SELECT INTO"}
 pg_deparse_trig_ddl_command_start_16429 | ddl_command_start | {"DROP TABLE"}
 pg_deparse_trig_table_rewrite_16429    | table_rewrite     | {"ALTER TABLE"}
 pg_deparse_trig_ddl_command_end_16429  | ddl_command_end   | {"CREATE TABLE","ALTER TABLE","DROP TABLE"}
(4 rows)

source_db=# DROP PUBLICATION mypub;
DROP PUBLICATION
source_db=# SELECT evtname, evtevent, evttags from pg_event_trigger;
 evtname | evtevent | evttags
---------+----------+---------
(0 rows)
```

A new WAL record for DDL messages

XLOG_LOGICAL_DDL_MESSAGE

```
/*
 * Generic logical decoding DDL message WAL record.
 */
typedef struct xl_logical_ddl_message
{
    Oid             dbId;         /* database Oid emitted from */
    Size            prefix_size;    /* length of prefix, including null terminator */
    Oid             relid;        /* id of the table */
    DeparsedCommandType cmdtype;      /* type of SQL command */
    Size            message_size;   /* size of the message */

    /* Payload, including null-terminated prefix of length prefix_size */
    char            message[FLEXIBLE_ARRAY_MEMBER];
} xl_logical_ddl_message;
```

Logical Logging Format

- Command string
  - Lightweight, easy to implement
  - Force search_path during apply
  - Doesn’t support schema mapping
  - Doesn’t allow straight machine editing of the command

- Structured format (JSON) generated by a deparsing utility
  - Fully qualifies DB objects - more secure
  - Allows support of schema mapping and command editing on the target – more flexible/robust
  - Allows command splitting on source
    - CREATE TABLE AS … SELECT … => CREATE TABLE
  - Development and maintenance burden, test coverage - more work

Logical Logging Format: DDL Deparsing

ALTER TABLE T1 ADD c3 int;

ALTER TABLE public.t1 ADD c3 int4;

```
{ "fmt":"ALTER TABLE %{identity}D %{subcmds:, }s",

  "identity":{
     "objname":"t1",
     "schemaname":"public"
   },

  "subcmds": [
     {
       "fmt":"ADD COLUMN %{definition}s",
       "definition":{
         "fmt":"%{name}I %{coltype}T %{default}s %{not_null}s %{collation}s",
         "name":"c3",
         "type":"column",
         "coltype":{
           "typmod":"",
           "typarray":false,
           "typename":"int4",
           "schemaname":"pg_catalog"
         },
         "default":{
           "fmt":"DEFAULT %{default}s",
           "present":false
         },
         "not_null":"",
         "collation": {
           "fmt":"COLLATE %{name}D",
           "present":false
         }
       }
     }
   ]
}
```

Logical Logging Format: DDL Deparsing with schema mapping

ALTER TABLE T1 ADD c3 int;

ALTER TABLE s1.t1 ADD c3 int4;

```
{ "fmt":"ALTER TABLE %{identity}D %{subcmds:, }s",

  "identity":{
     "objname":"t1",
     "schemaname":"s1"
   },

  "subcmds": [
     {
       "fmt":"ADD COLUMN %{definition}s",
       "definition":{
         "fmt":"%{name}I %{coltype}T %{default}s %{not_null}s %{collation}s",
         "name":"c3",
         "type":"column",
         "coltype":{
           "typmod":"",
           "typarray":false,
           "typename":"int4",
           "schemaname":"pg_catalog"
         },
         "default":{
           "fmt":"DEFAULT %{default}s",
           "present":false
         },
         "not_null":"",
         "collation": {
           "fmt":"COLLATE %{name}D",
           "present":false
         }
       }
     }
   ]
}
```

Apply DDL

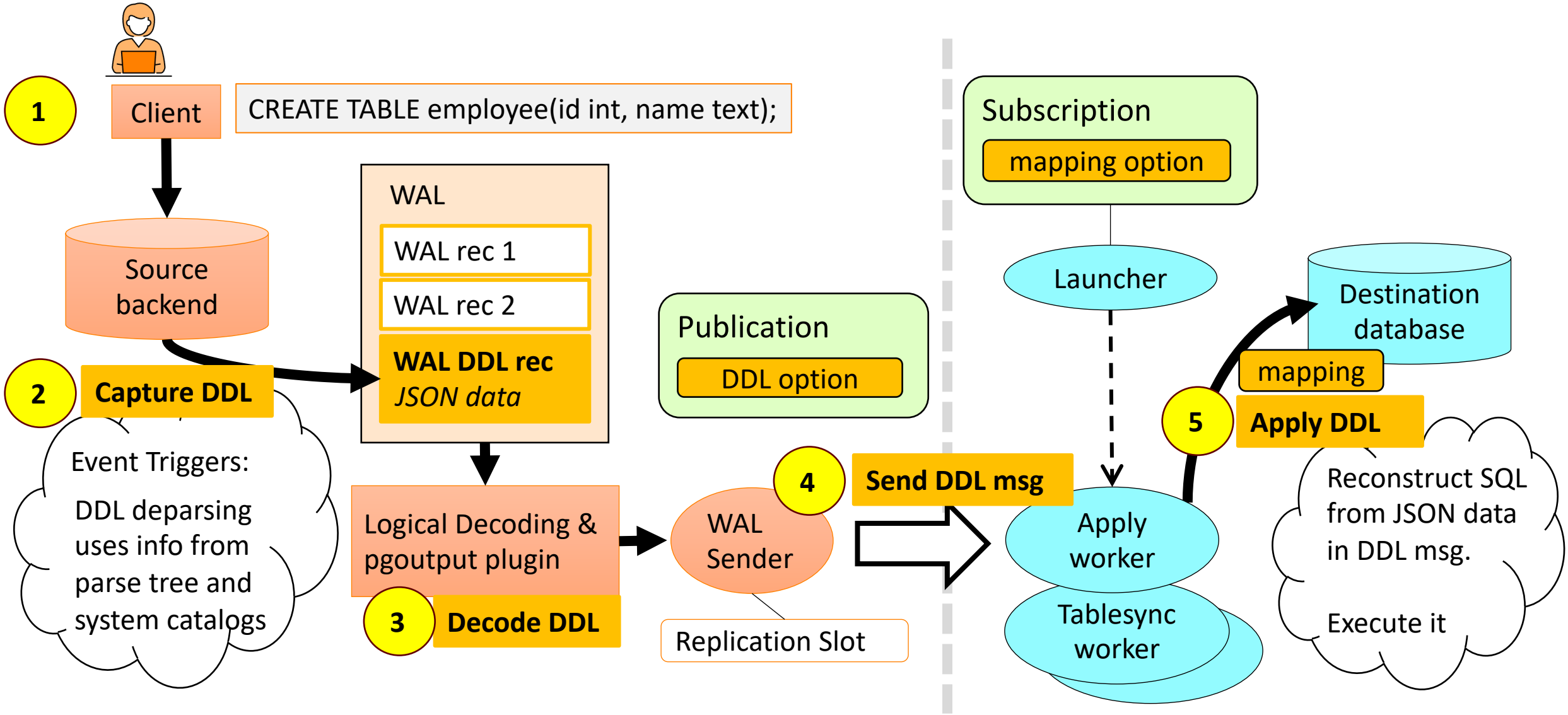
- Reconstruct the DDL commands from DDL messages
  - Perform schema mapping if configured (TODO)
  - Transform the command to auto-resolve syntax incompatibility if there is any (TODO)

- Automatically run ALTER SUBSCRIPTION … REFRESH PUBLICATION after CREATE TABLE

- Ownership mapping (new subscription option)

PostgreSQL Logical Replication + DDL support (details)

FUJITSU



Special Cases

- Non-replicated object
  - DROP TABLE replicated_foo, notreplicated_bar; => DROP TABLE IF EXISTS;

- Command performs both DDL and DML
  - CREATE TABLE foo AS SELECT field_1, field_2 FROM bar; / SELECT INTO;
  - ALTER TABLE ddl_test ADD COLUMN b int DEFAULT random();
  - Guarantee data consistency

- This is not a full list of special cases

Special Cases: CREATE TABLE AS SELECT / SELECT INTO

- WAL log and replicate the DDL part first without DML
  - CREATE TABLE t2 AS SELECT id, name from t1;

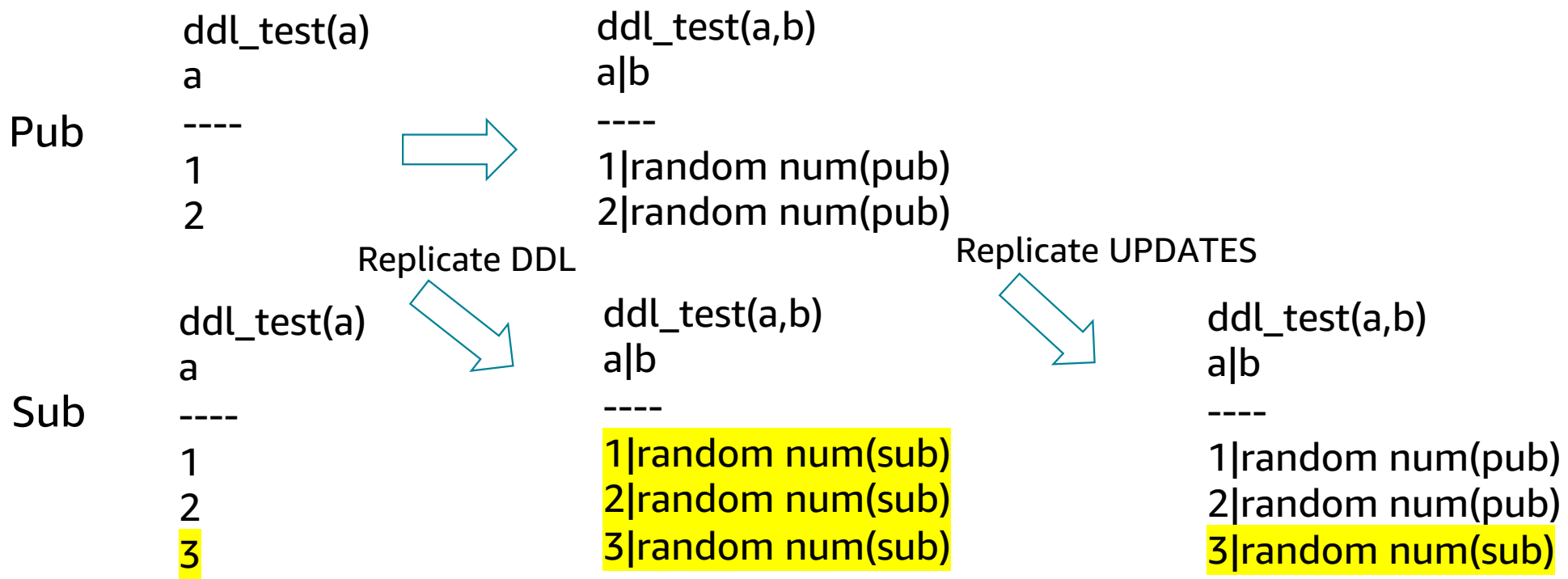
    =>

    CREATE TABLE t2 (id serial, name text);

- Let the data population replicate to the subscriber by the subsequent DML replication

Special Cases: table rewrite with volatile function

- ALTER TABLE ddl_test ADD COLUMN b int DEFAULT random();
  - don't replicate such commands
  - if the rewrite function is replication safe, can separate the DDL change and table rewrite (UPDATES) and replicate each.



Testing

- TAP tests for DDL replication
- A new testing module for the DDL deparsing utility
  - Test the deparsed JSON output of a DDL is expected
  - Test that the reconstructed DDL command is expected
  - Test the reconstructed command from JSON can be executed and has the same effect as the original command by comparing the results from pg_dump

Related Issues

- Global commands
- Initial Schema Sync

Global Commands

- Commands that manage global objects
  - DATABASE Commands
  - ROLE Commands
  - TABLESPACE Commands
  - GRANT ROLE (GRANT privilege to rolex)
  - GRANT/REVOKE on global objects (GRANT ALL ON DATABASE)
- Not captured by event triggers
- Global objects are not schema qualified
- Per-DB replication model (per-db pg_publication) isn’t ideal for global objects replication

Initial Schema Sync

- Today initial schema has to be manually setup on the subscriber

- Automate initial schema sync
  - How to get the schema definition on the subscriber
    - Use pg_dump with new options to dump table with dependencies
    - Provide more ruleutils functions like pg_get_viewdef
    - Build a pg_dump_library that can be referenced by pg_dump and the backend

  - Properly handle concurrent DDLs during initial sync

- It's being discussed in a different pgsql-hackers thread Initial schema sync for logical replication

Summary

- Motivation

- Support DDL replication on the existing logical replication architecture
  - Replication granularity
  - Capture DDL
  - Logical logging format
  - Apply DDL
  - Special cases

- Related issues
  - Global commands
  - Initial schema sync

Thank you for attending

Peter Smith
Fujitsu
peter.b.smith@fujitsu.com

Zheng Li (Zane)
Amazon RDS Open Source
zhelli@amazon.com

References

- wiki https://wiki.postgresql.org/wiki/Logical_replication_of_DDLs
- pgsql-hackers thread - Support logical replication of DDLs
- pgsql-hackers thread - Deparsing utility commands
- pgsql-hackers thread - Support logical replication of global object commands
- pgsql-hackers thread - Initial schema sync for logical replication
- PG documentation for Logical Replication
- PG documentation for CREATE PUBLICATION