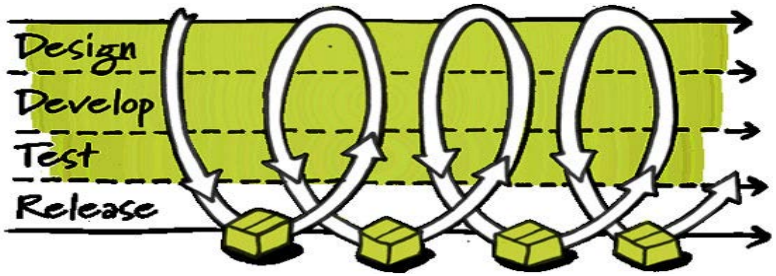
DATALUTION: A TOOL FOR CONTINUOUS SCHEMA EVOLUTION IN NOSQL-BACKED WEB APPLICATIONS

STEFANIE SCHERZINGER
STEPHANIE SOMBACH
KATHARINA WIECH
MEIKE KLETTKE
UTA STÖRL

OSTBAYERISCHE TECHNISCHE HOCHSCHULE REGENSBURG

datalution

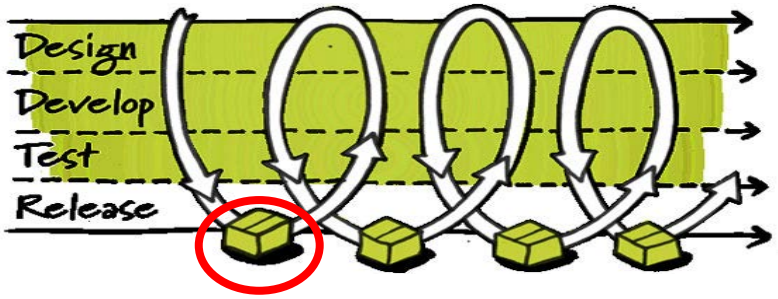


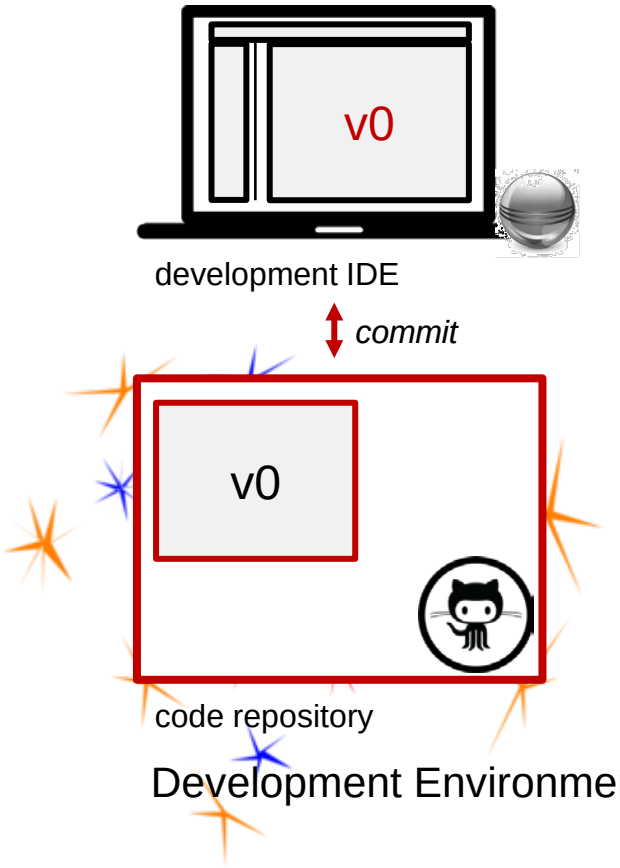
development IDE

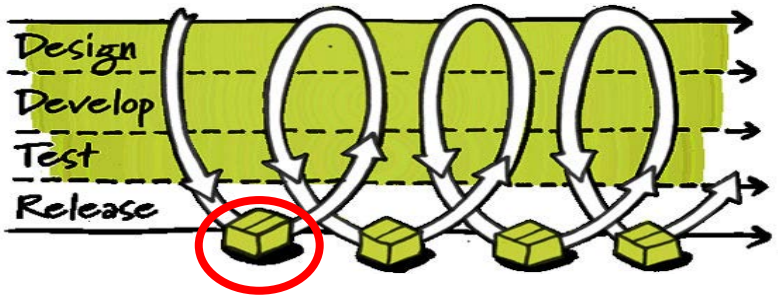
commit

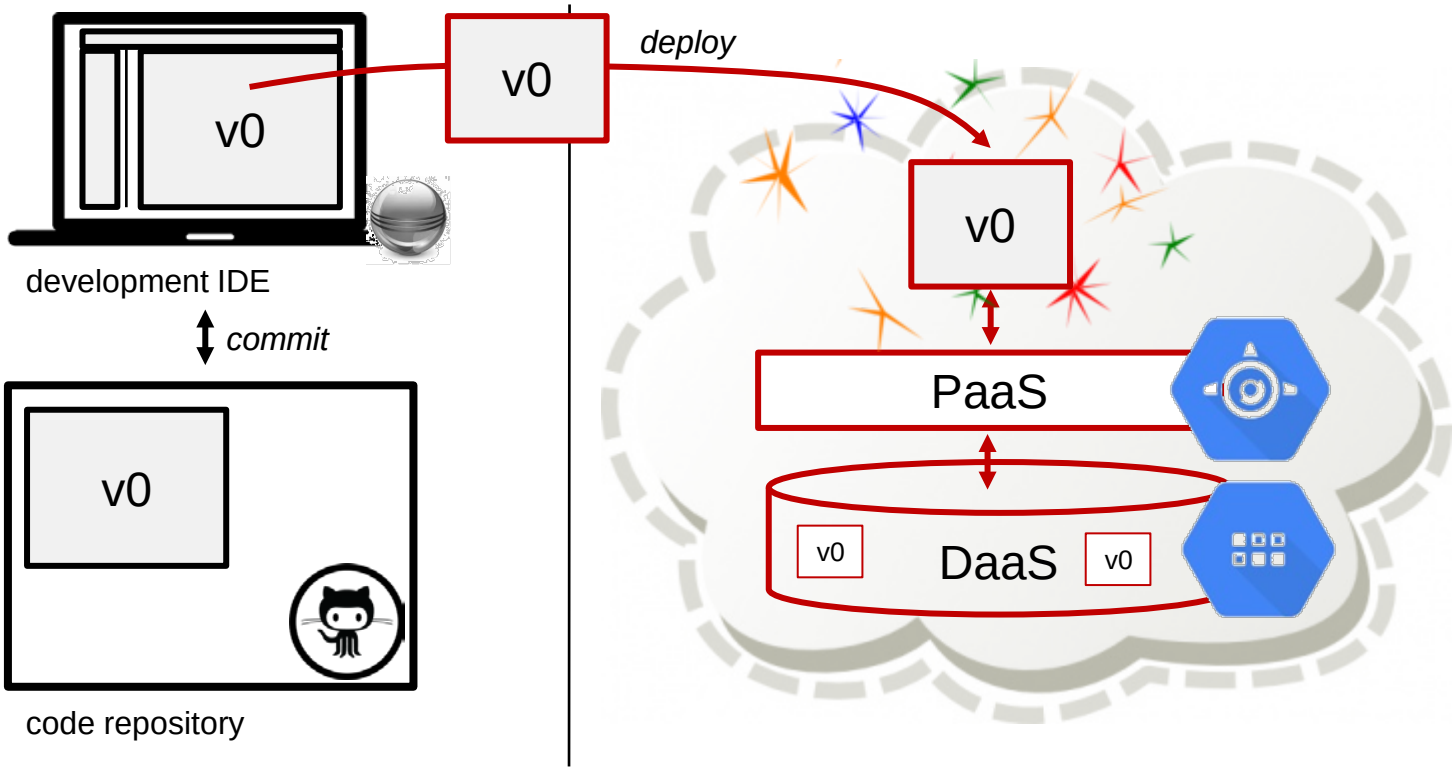
code repository

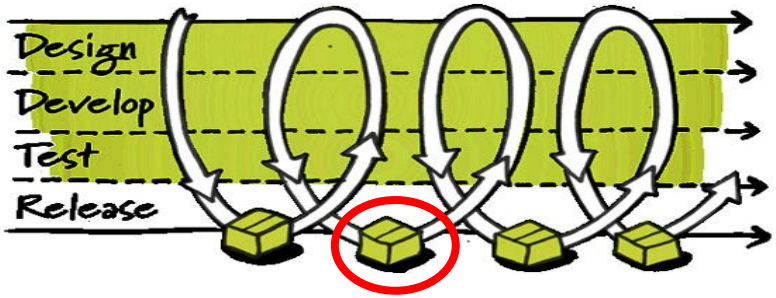
Development Environment

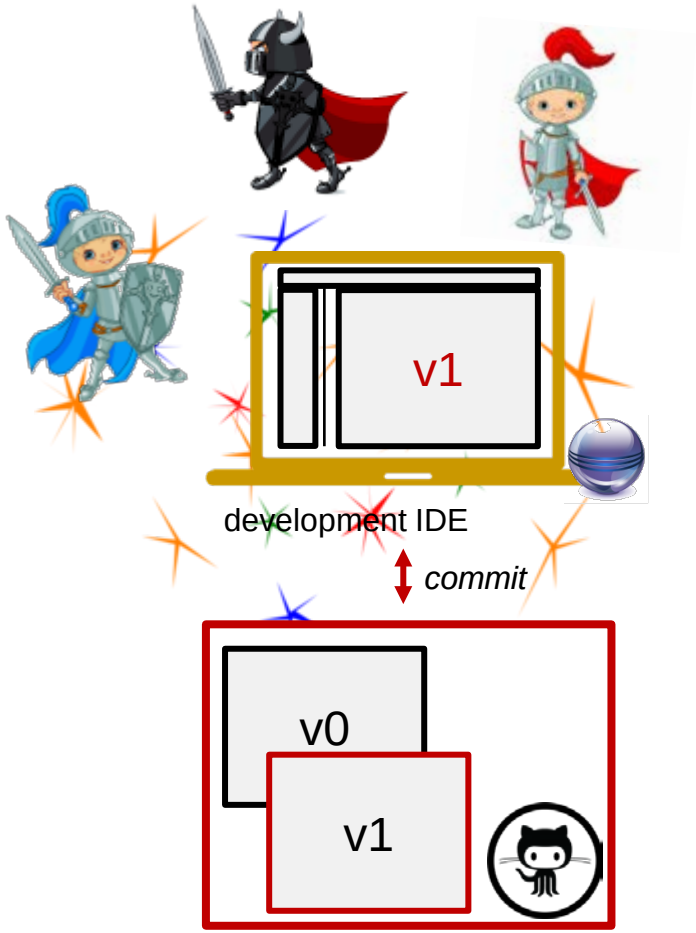

Design
Develop
Test
Release


v0
development IDE
commit
v0
code repository
Development Environment

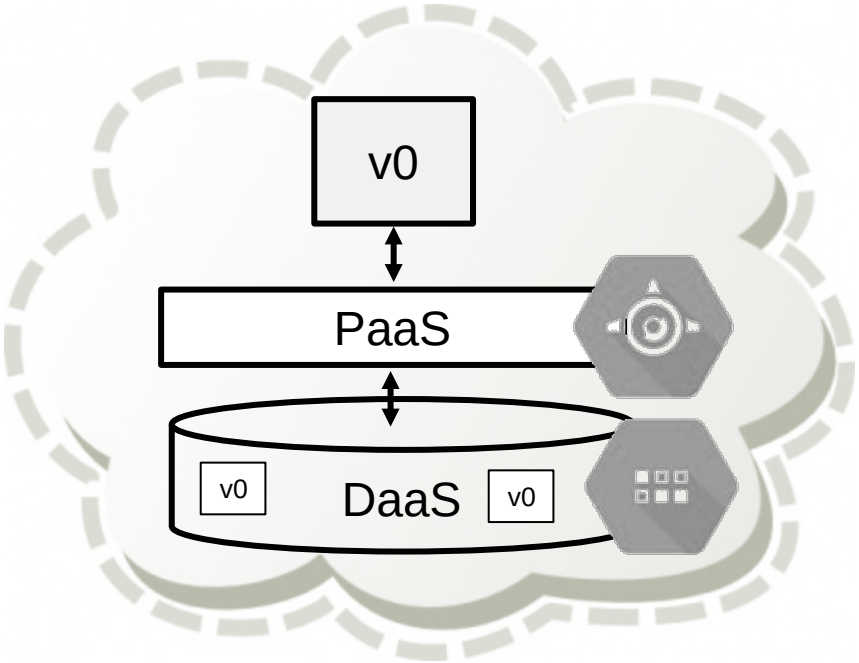

Design
Develop
Test
Release


v0
v0
deploy
development IDE
commit
v0
code repository
v0
PaaS
v0
DaaS
v0
Development Environment
Production Environment

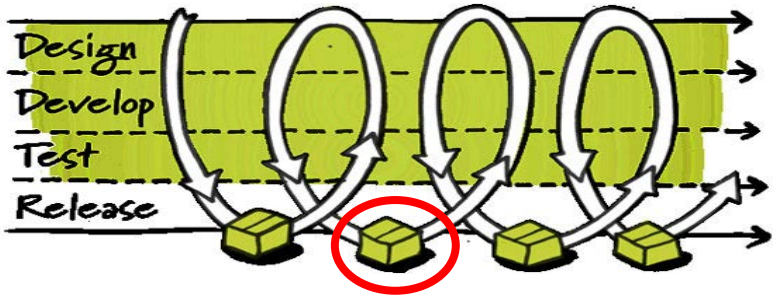


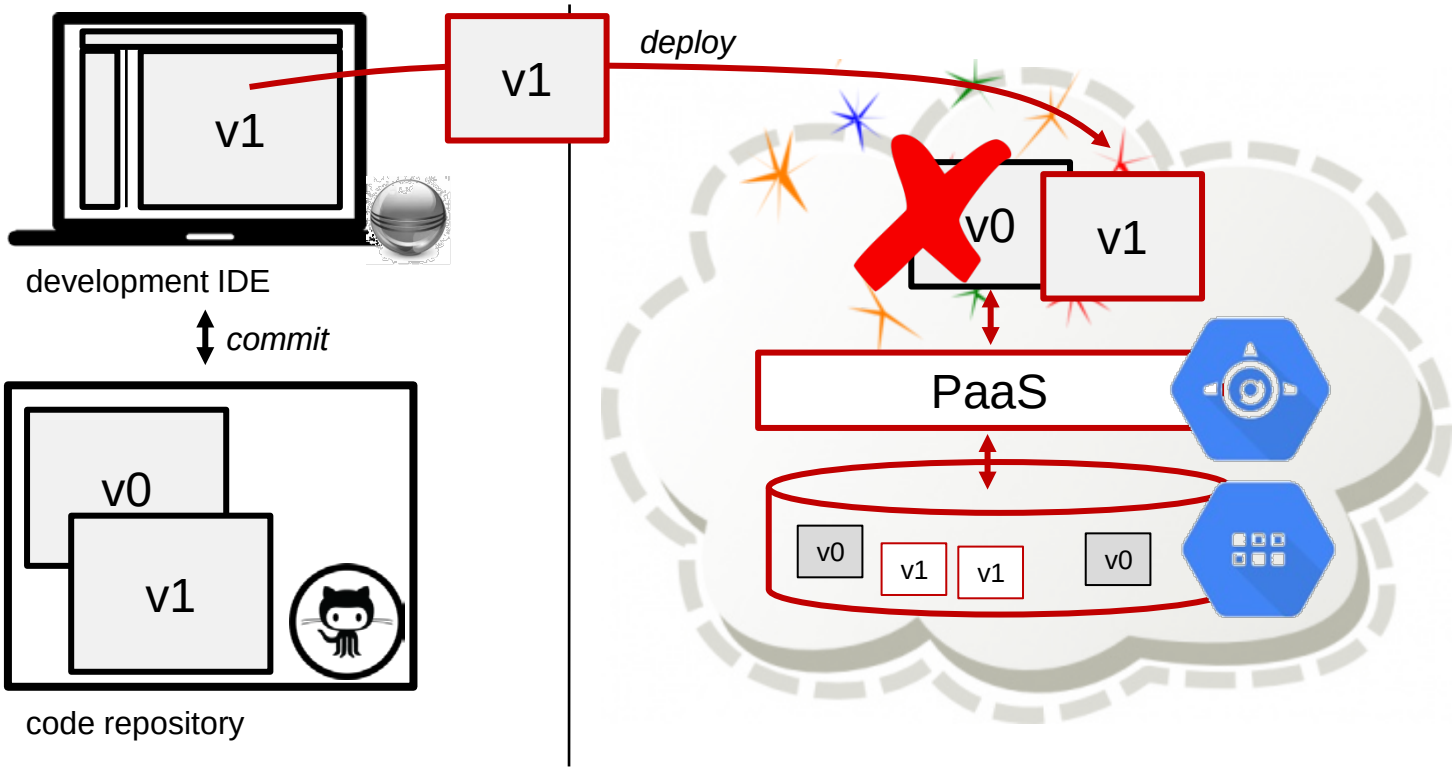


Development Environment



Production Environment


Design
Develop
Test
Release


v1
v1
deploy
development IDE
commit
v0
v1
code repository
v0
v1
PaaS
v0
v1
v1
v0
Development Environment
Production Environment

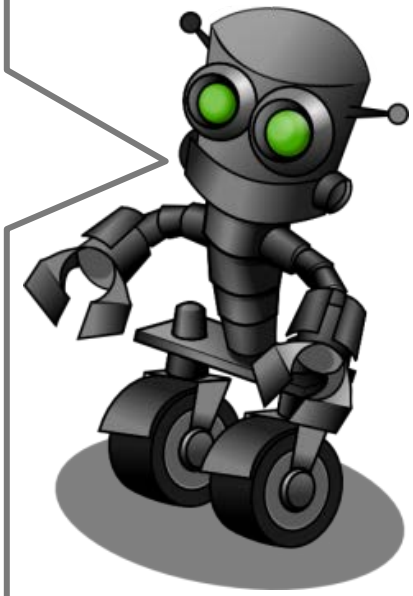
DESIDERATUM

The application code declares a schema.

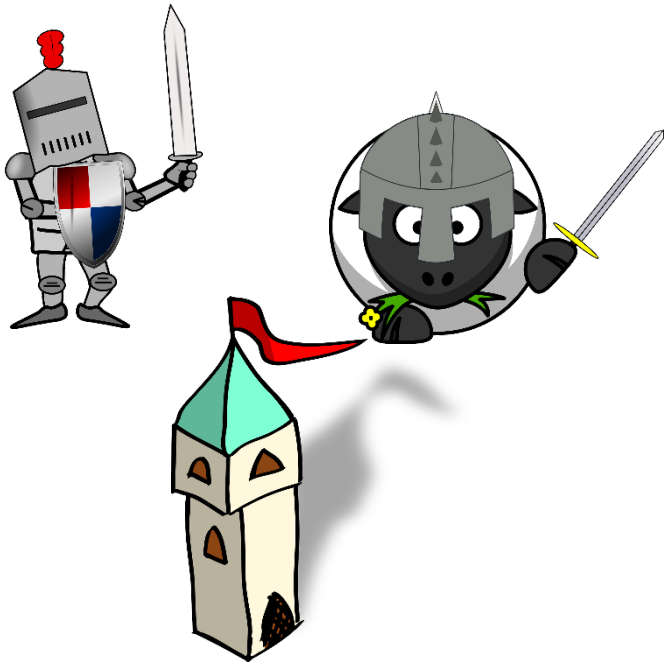
The application code evolves.

Thus, we need to address schema evolution:

- Eager
- Lazy with Object-NoSQL Mappers
- Lazy with Datalution



EXAMPLE: GAMING APPLICATION



Release 1

- `Player(ID, NAME)`
- `Mission(ID, TITLE, PID)`

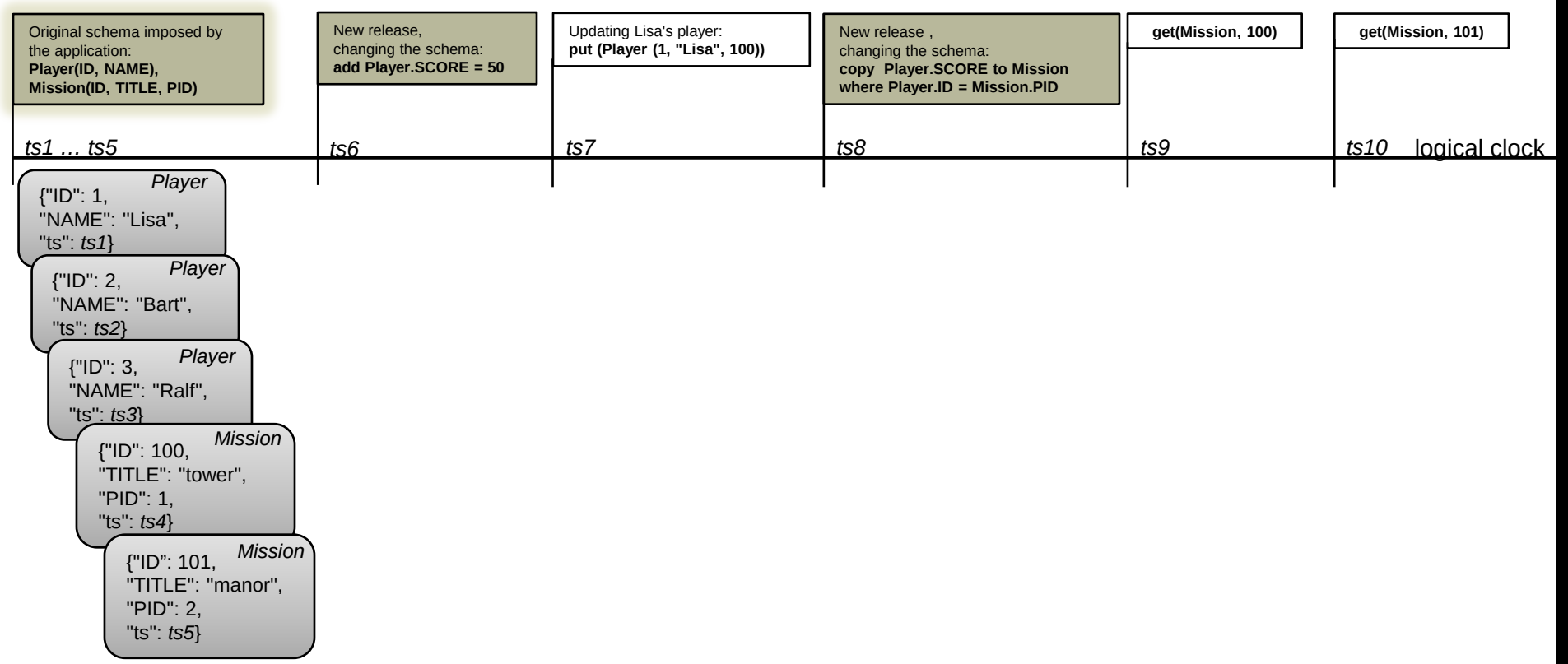
Release 2

- Players carry a property SCORE:
  `add Player.SCORE = 50`

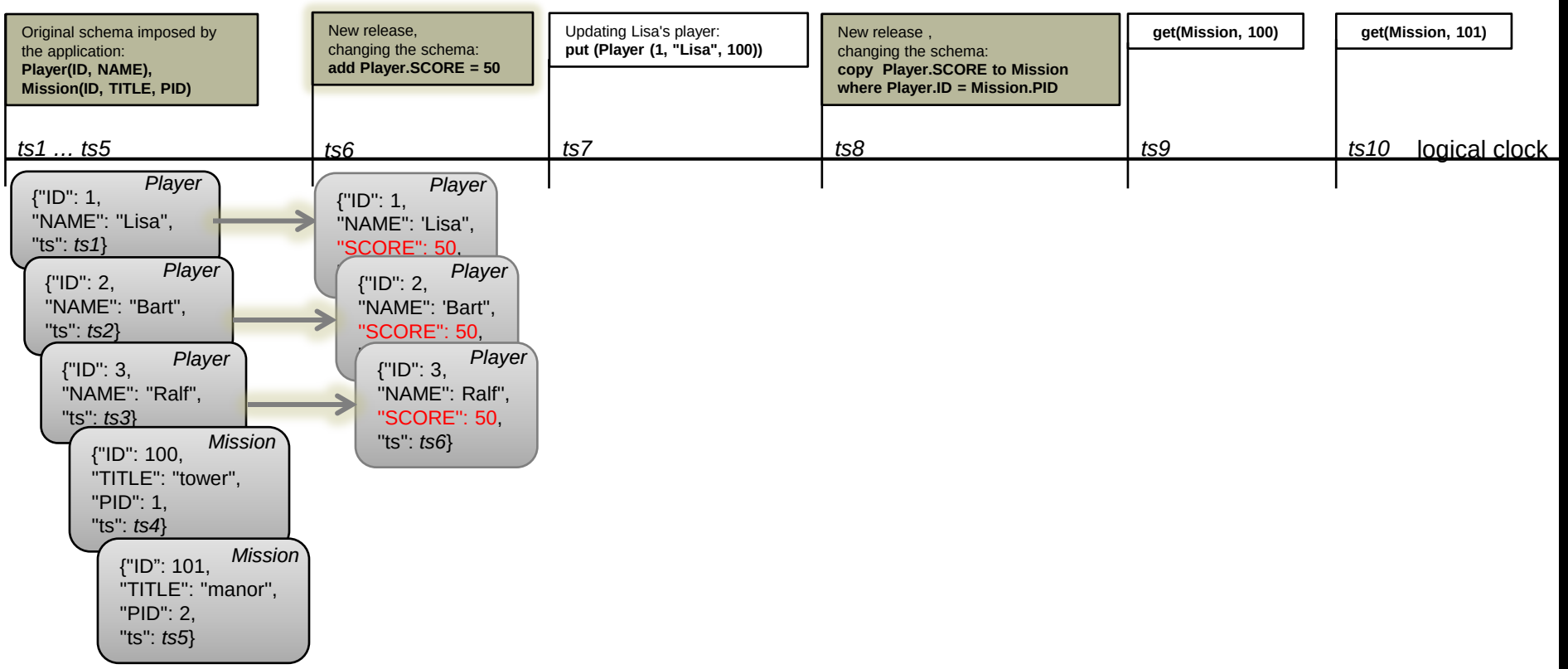
Release 3

- Missions carry their player‘s score
  `copy Player.SCORE to Mission`
  `where Player.ID = Mission.PID`

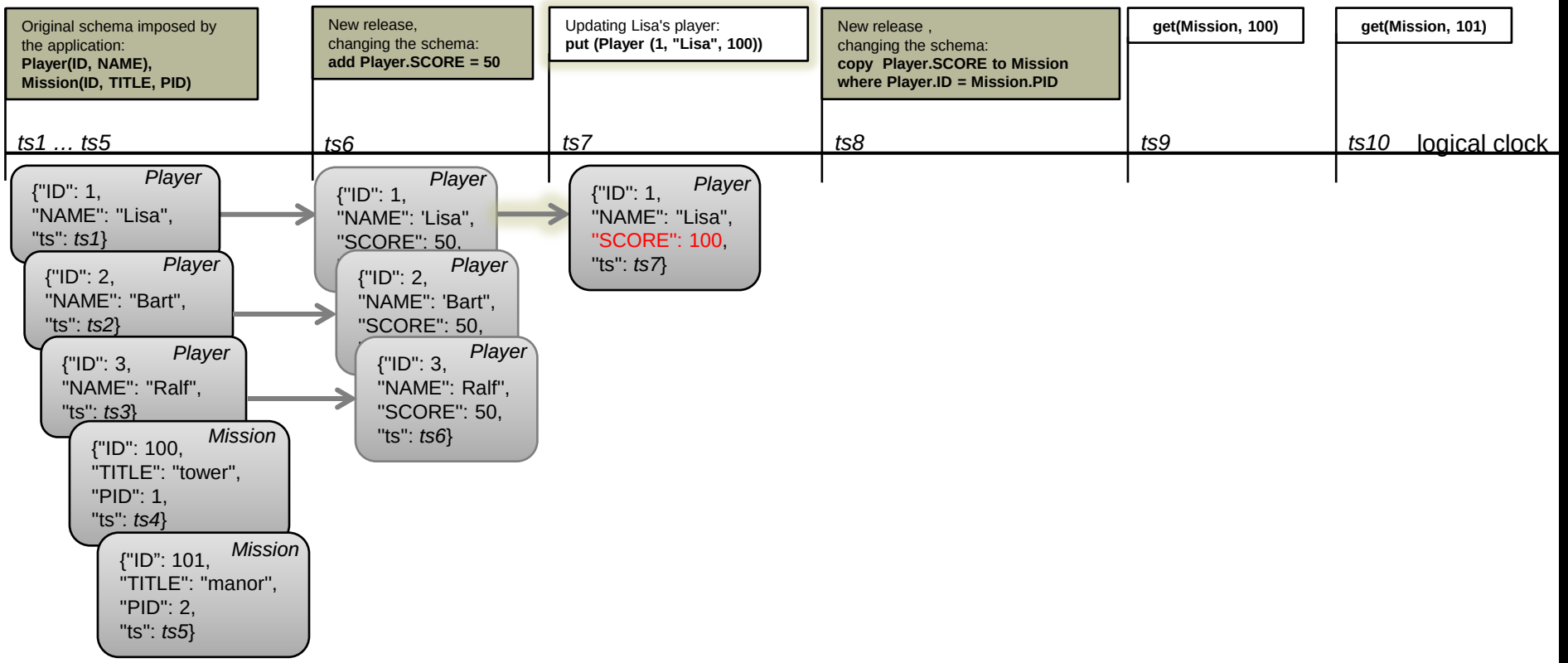
EAGER MIGRATION



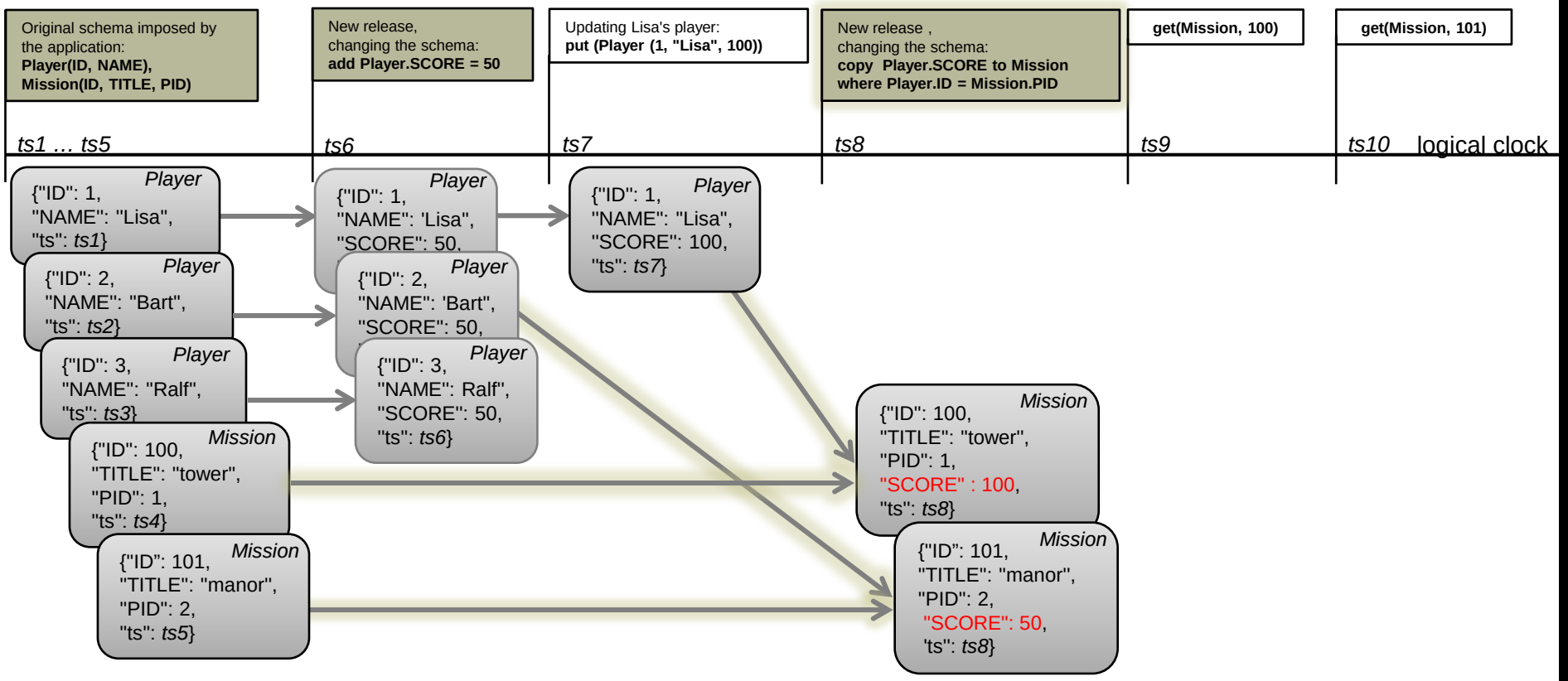
EAGER MIGRATION



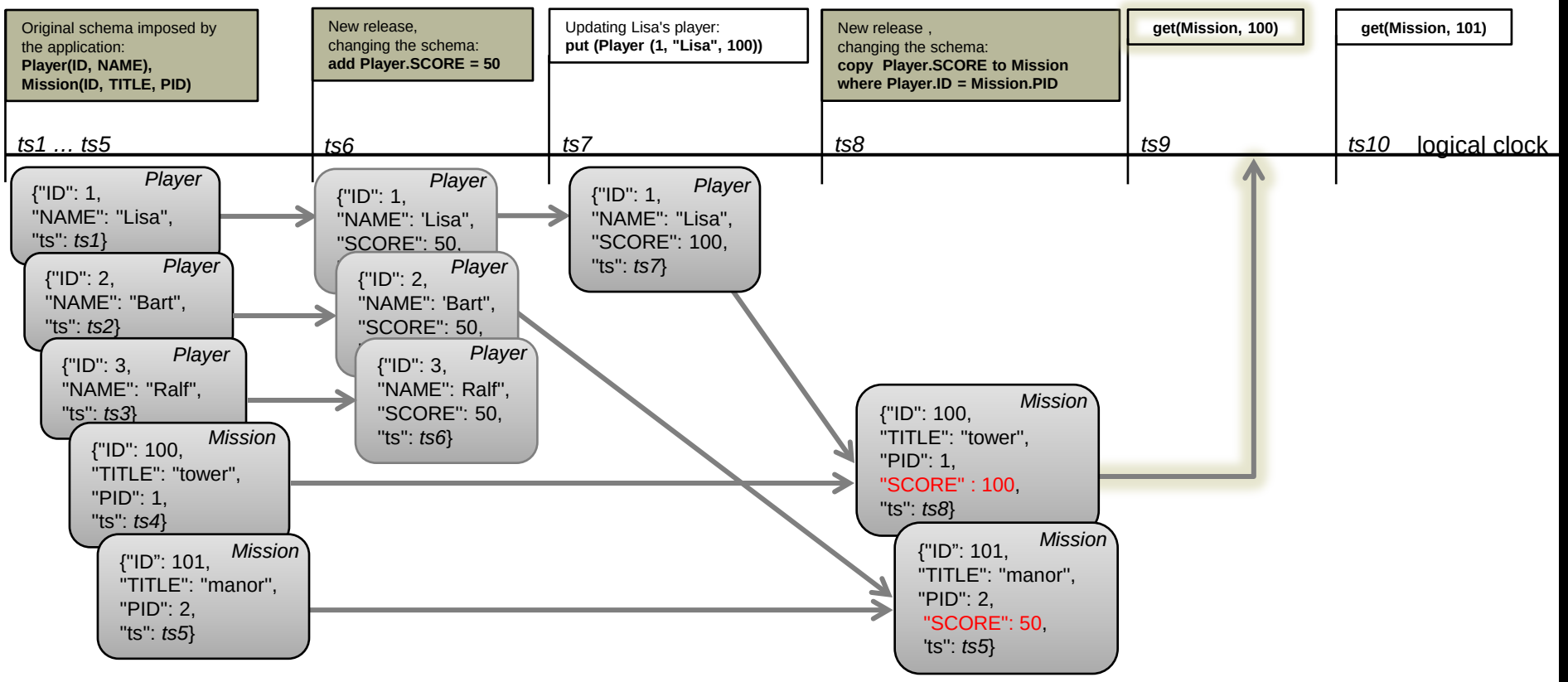
EAGER MIGRATION



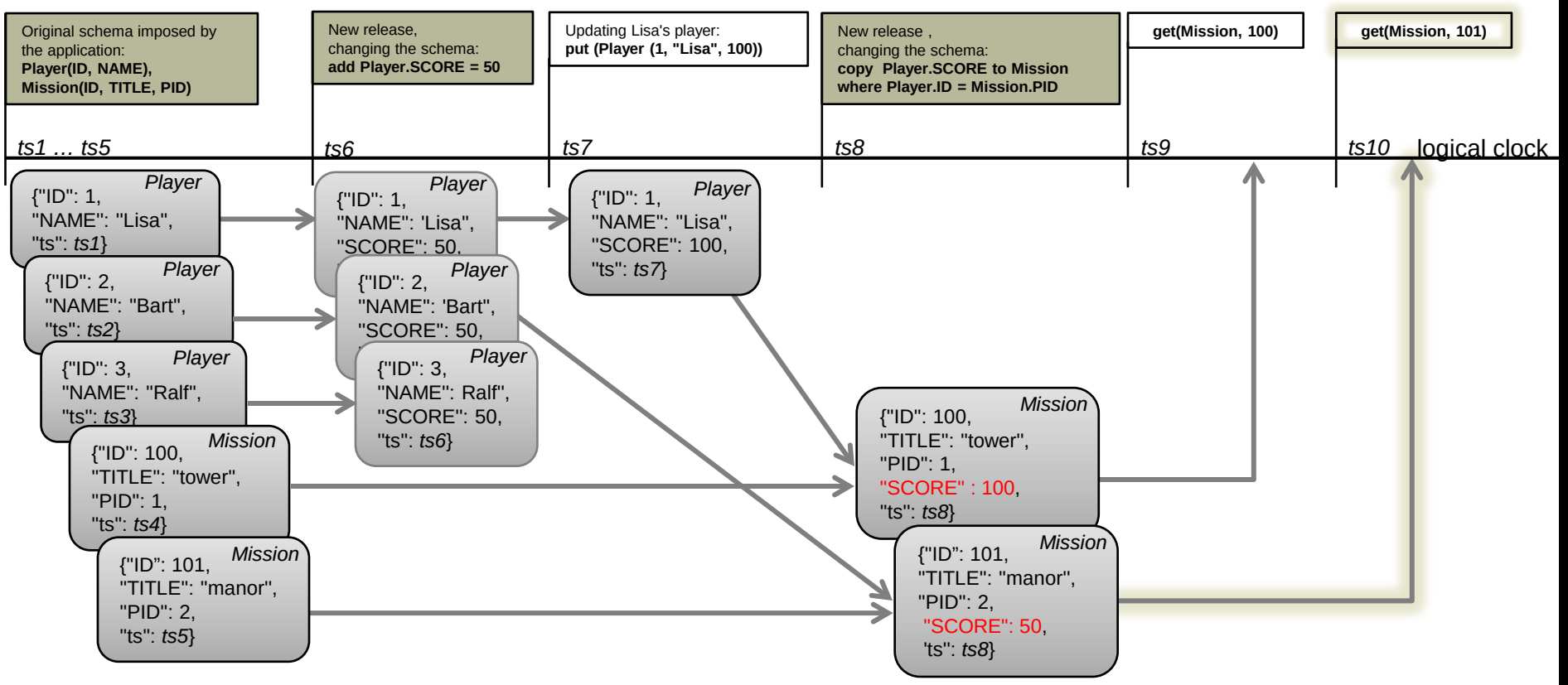
EAGER MIGRATION



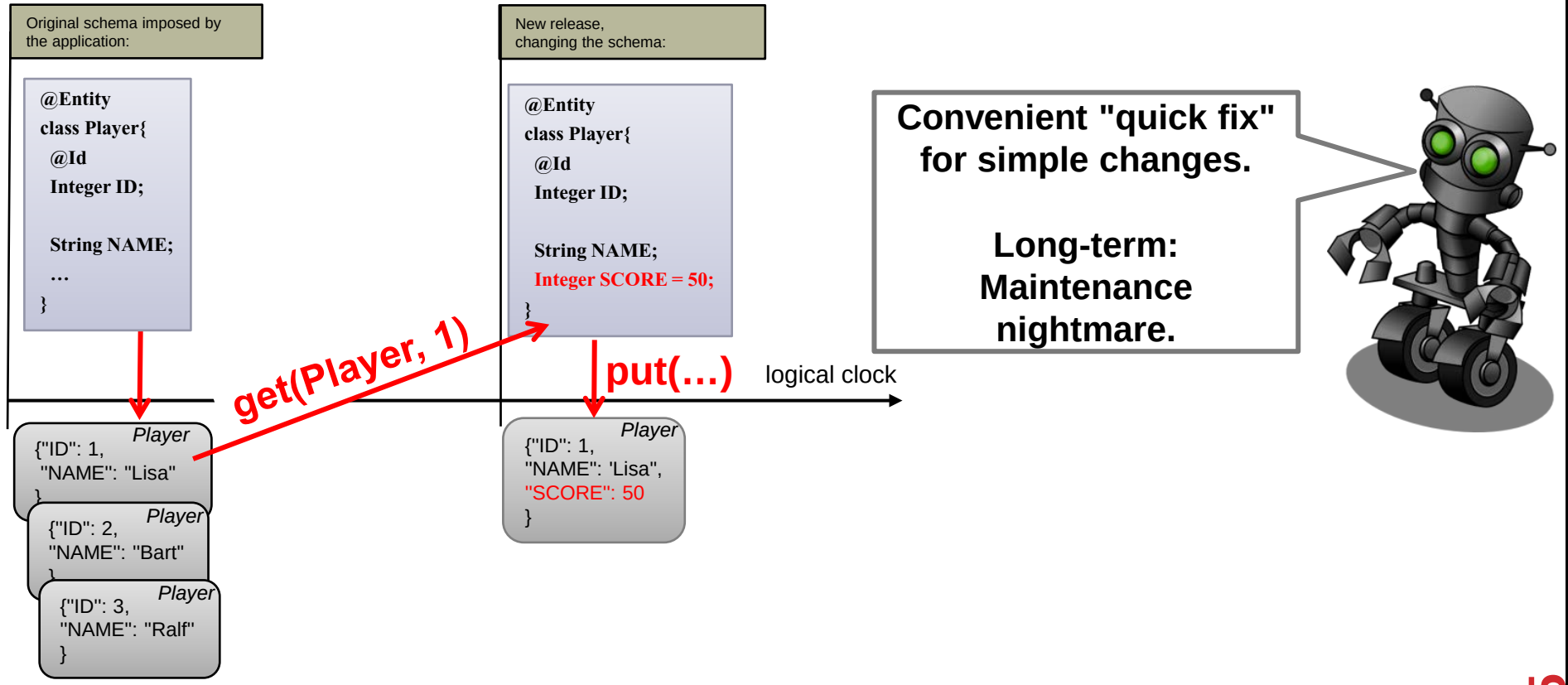
EAGER MIGRATION



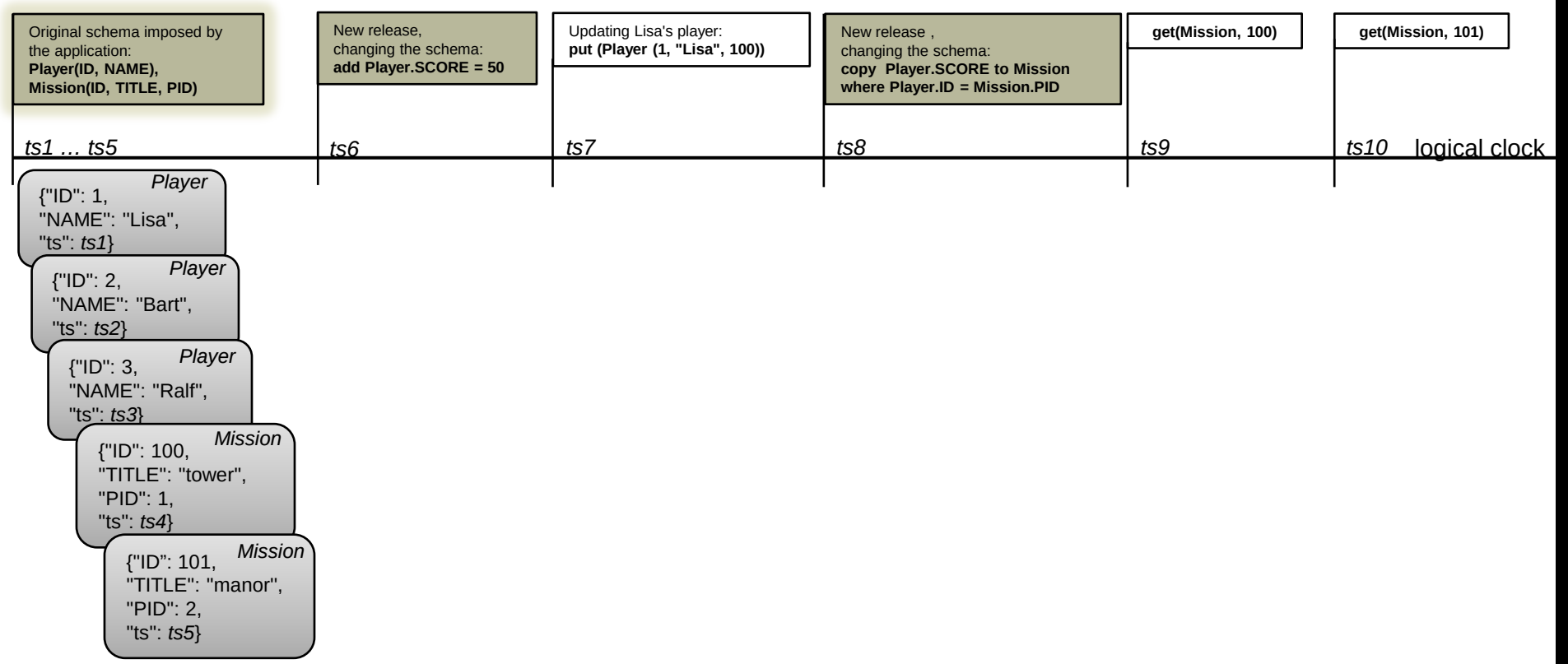
EAGER MIGRATION



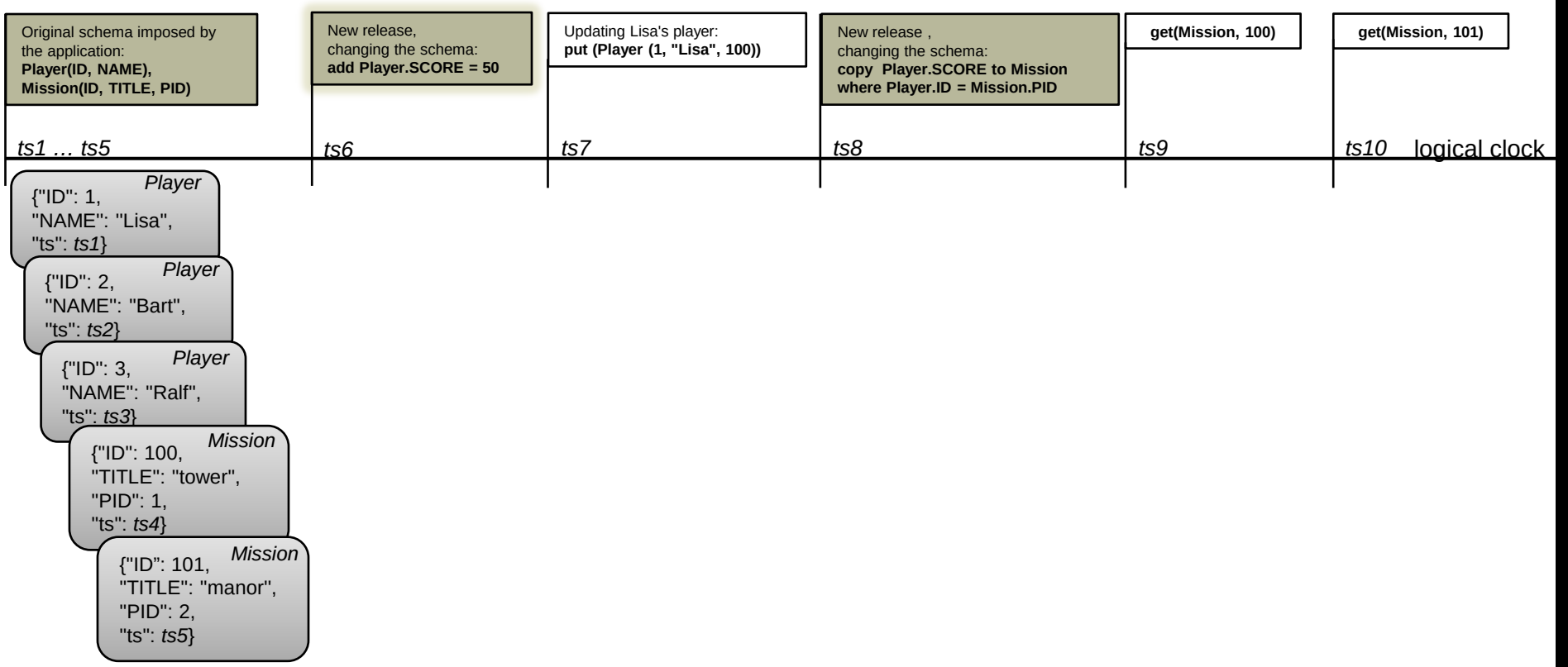
LAZY EVOLUTION WITH OBJECT-NOSQL MAPPERS



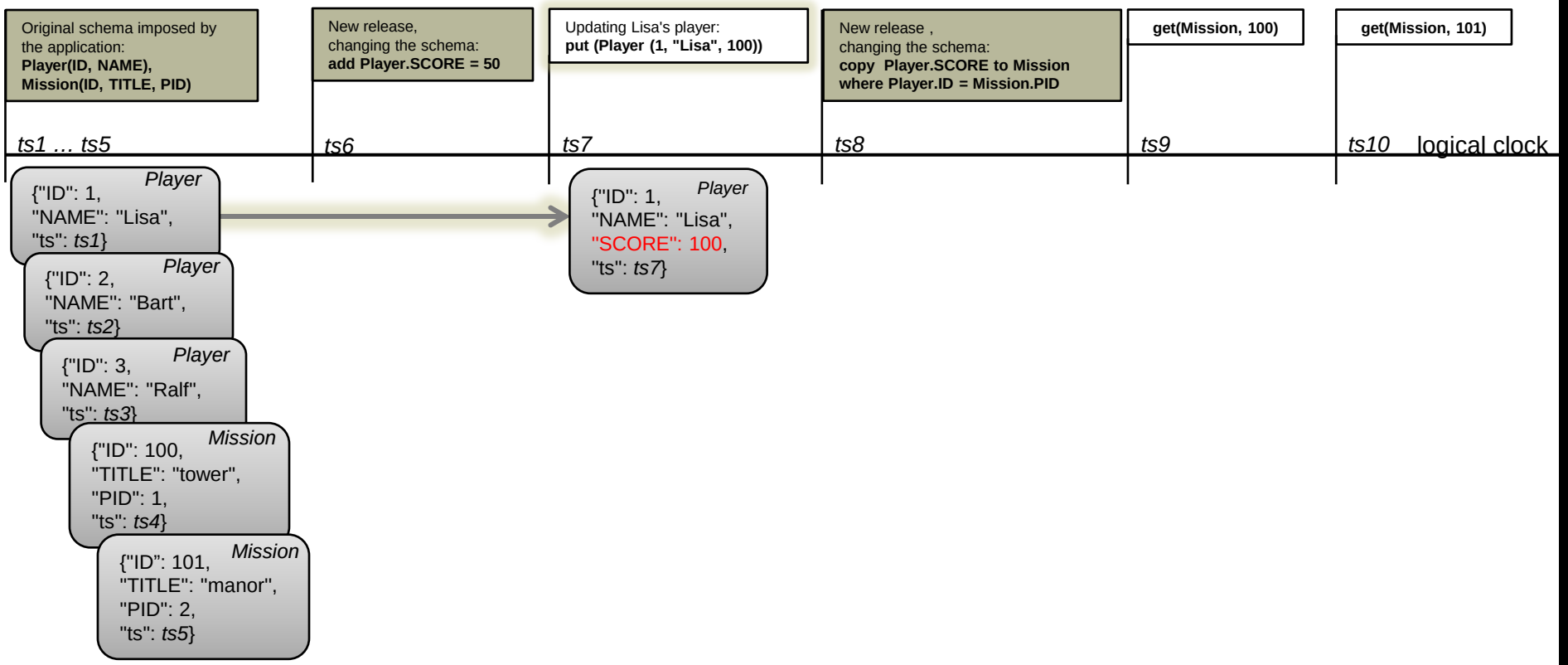
LAZY MIGRATION IN DATALUTION



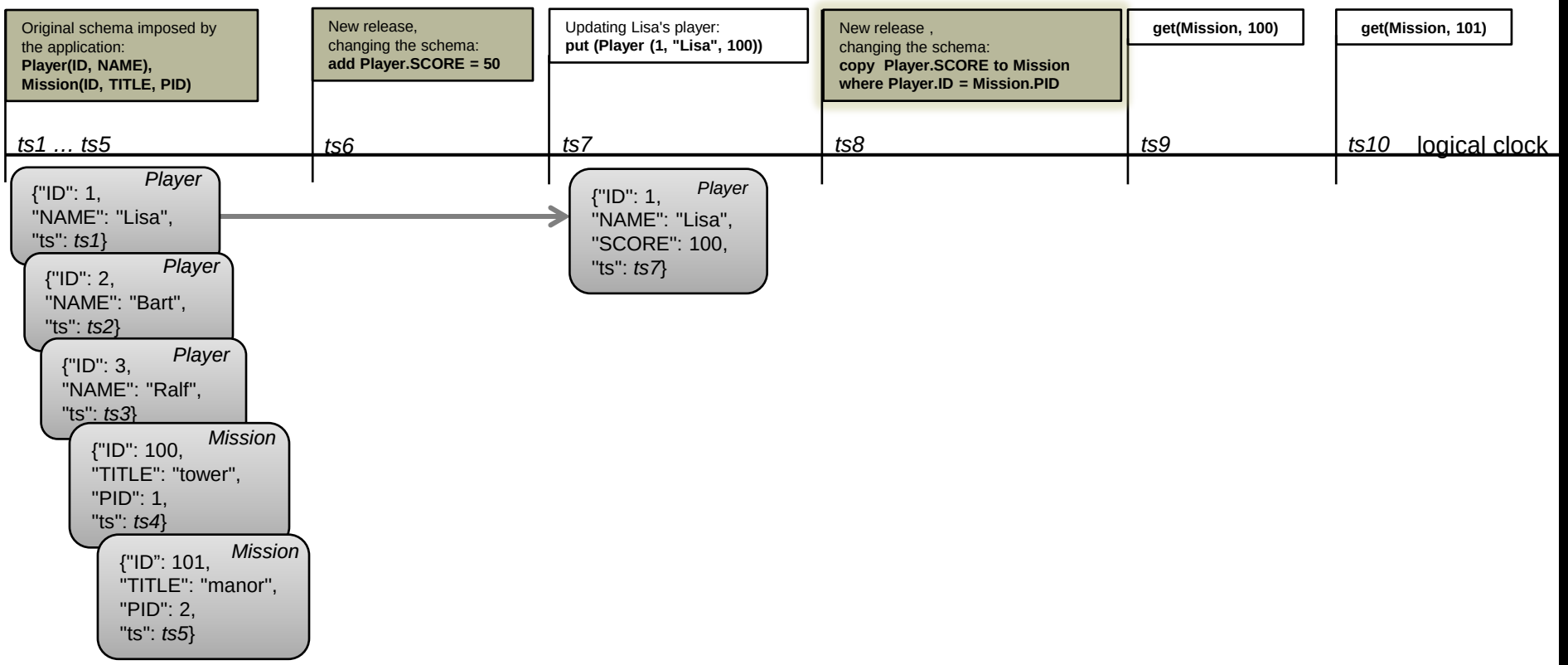
LAZY MIGRATION IN DATALUTION



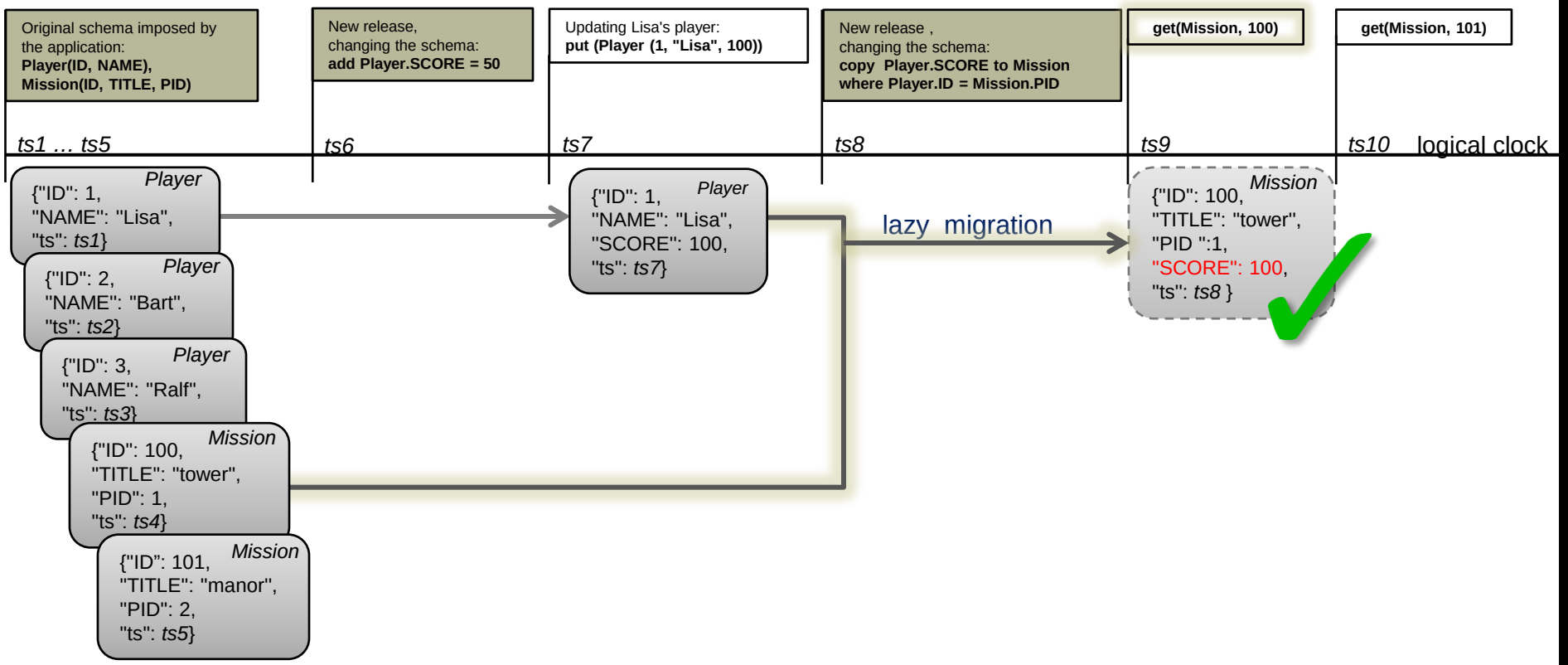
LAZY MIGRATION IN DATALUTION



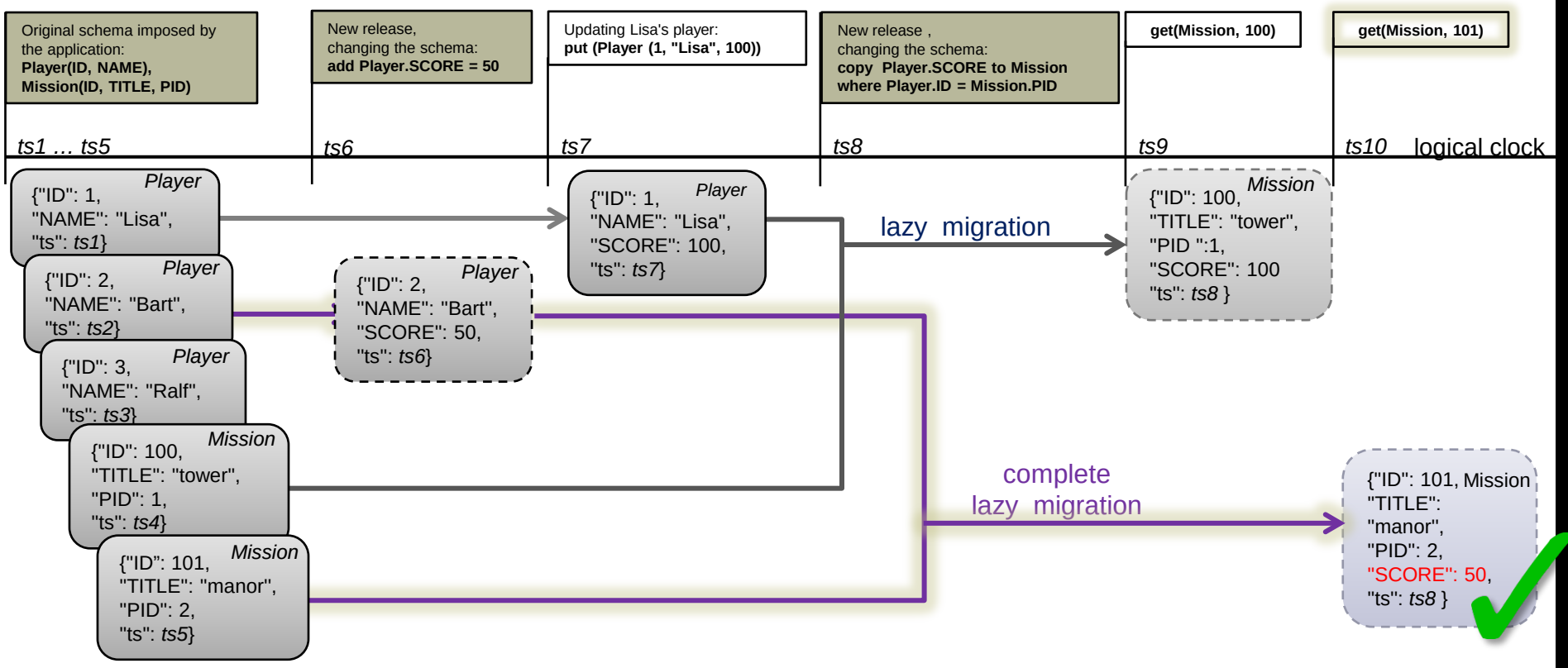
LAZY MIGRATION IN DATALUTION



LAZY MIGRATION IN DATALUTION



LAZY MIGRATION IN DATALUTION



DATALOG MODEL (NONRECURSIVE, STRATIFIED)

```
a1: put(Player(1, "Lisa"));
a2: put(Player(1, "Lisa S."));
```

```
r1: Player(1, "Lisa", ts1).
r2: Player(1, "Lisa S.", ts2).
```

```
a3: get("Player", 1);
```

```
r3: legacyPlayer(ID, TS) :-
      Player(ID, _, TS), Player(ID, _, NTS), TS < NTS.
r4: latestPlayer(ID, TS) :-
      Player(ID, _, TS), not legacyPlayer(ID, TS).
r5: getPlayer(ID, NAME, TS) :-
      Player(ID, NAME, TS), latestPlayer(ID, TS).
```

transient rule – derived facts not kept around for incremental evaluation

Let *kind*$[r](ID, P_1, ..., P_n)$ be the schema imposed by the current application release. *ts* denotes a fresh timestamp associated with release r.

i) **add** *kind*.$P_{n+1} = v$, where P_{n+1} is a new property name and v is a default value (in the new version of the entity, P_{n+1} has value v):

kind$[r+1]$(ID, P1,...,Pn, v, *ts*) :- *kind*$[r]$(ID, P1,...,Pn, OTS), latest*kind*$[r]$(ID, OTS).

ii) **delete** *kind*.P_i

kind$[r+1]$(ID, P1,...,P(i-1),P(i+1),...,Pn, *ts*) :- *kind*$[r]$(ID, P1,...,Pn, OTS), latest*kind*$[r]$(ID, OTS).

Let *kindS*$[r](ID, S_1, ..., S_n)$ and *kindT*$[r](ID, T_1, ..., T_m)$ be the current source and target schema imposed by the application.

iii) **copy** *kindS*.S_i **to** *kindT* where *kindS*.*ID* = *kindT*.T_j

kindT$[r+1]$(ID_T, T1,...,Tm, Si, *ts*) :- *kindT*$[r]$(ID_T, T1,...,Tm, TS_T), latest*kindT*$[r]$(ID_T, TS_T),
kindS$[r]$(ID_S, S1,...,Sn, TS_S), latest*kindS*$[r]$(ID_S, TS_S), ID_S = Tj.

kindT$[r+1]$(ID_T, T1,...,Tm, *null*, *ts*) :- *kindT*$[r]$(ID_T, T1,...,Tm, TS_T), latest*kindT*$[r]$(ID_T, TS_T),
not *kindS*$[r]$(ID_S, S1,...,Sn, TS_S), ID_S = Tj.

kindS$[r+1]$(ID, S1, ..., Sn, *ts*) :- *kindS*$[r]$(ID, S1, ..., Sn, OTS), latest*kind*$[r]$(ID, OTS).

iv) **move** *kindS*.S_i **to** *kindT* where *kindS*.*ID* = *kindT*.T_j, with the same first two rules as for copy, as well as the following rule:

kindS$[r+1]$(ID, S1,...,S(i-1),S(i+1),...,Sn, *ts*) :- *kindS*$[r]$(ID, S1,...,Sn, OTS), latest*kind*$[r]$(ID, OTS).

DATALUTION: DATALOG-BASED

- **Eager migration: Incremental bottom-up evaluation**
- **Lazy migration: Incremental top-down evaluation**
  - Employing sideways information passing strategies
  - Exploiting uniqueness of identifiers

- **Both strategies always yield the same result**

- **Progress:**
  - Theory in DBPL@SPLASH'15 paper
  - Demo of PoC Datalution at QUDOS'16
  - Ongoing: Integration with NoSQL data store