Zero downtime deployments with DB changes

@sebdehne

Systek

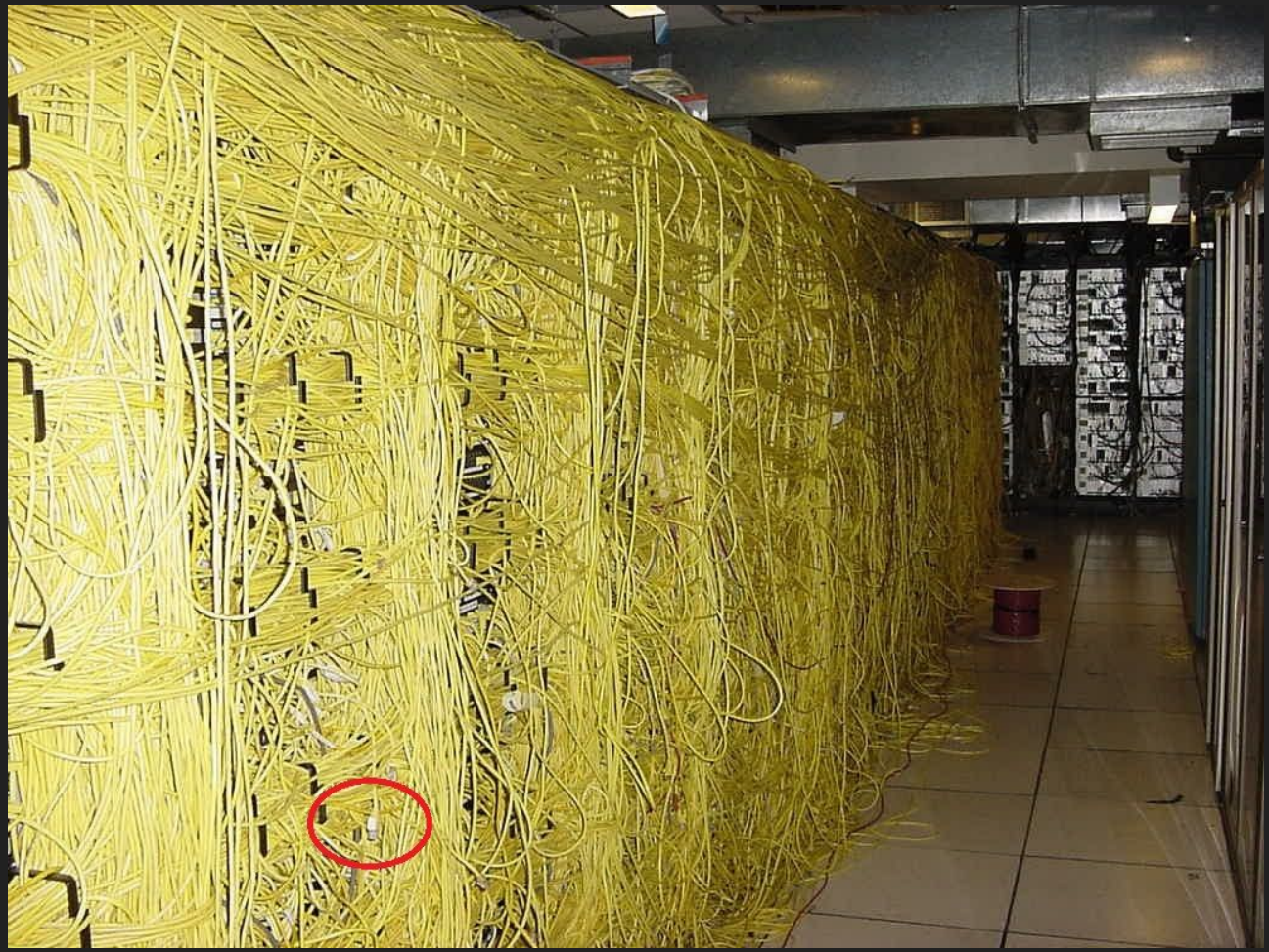
chalkthoughts.com

Systek

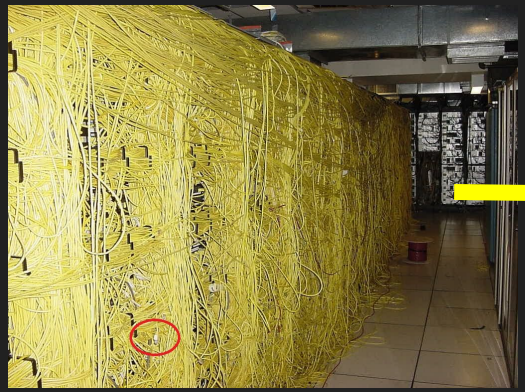
buildipedia.com

fastcodesign.com

Systek



Agile development



Refactor

Systek

Source: youtube

Daytime deployments

Opening hours: 24/7

Continuous delivery

New webshop for:

Agile development

Short time to market

Low maintenance costs

Requirements

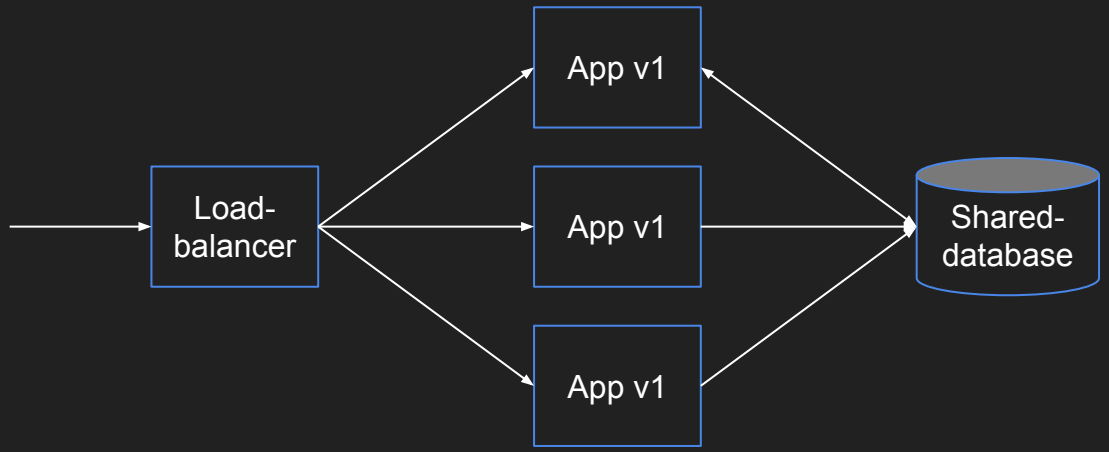
New webshop for:

Requirement #1 - able to deploy without interruptions

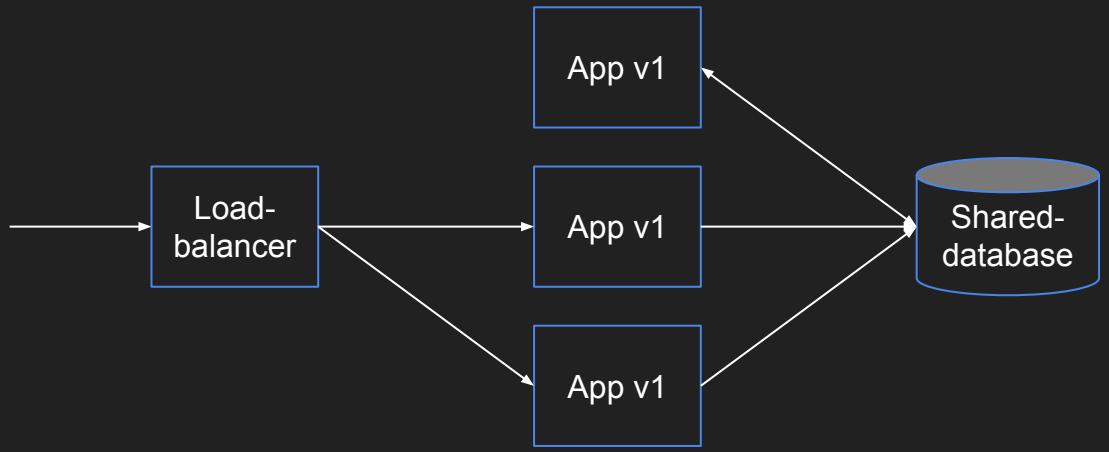
Requirement #2 - ability to refactor code and database

Blue/green deployments

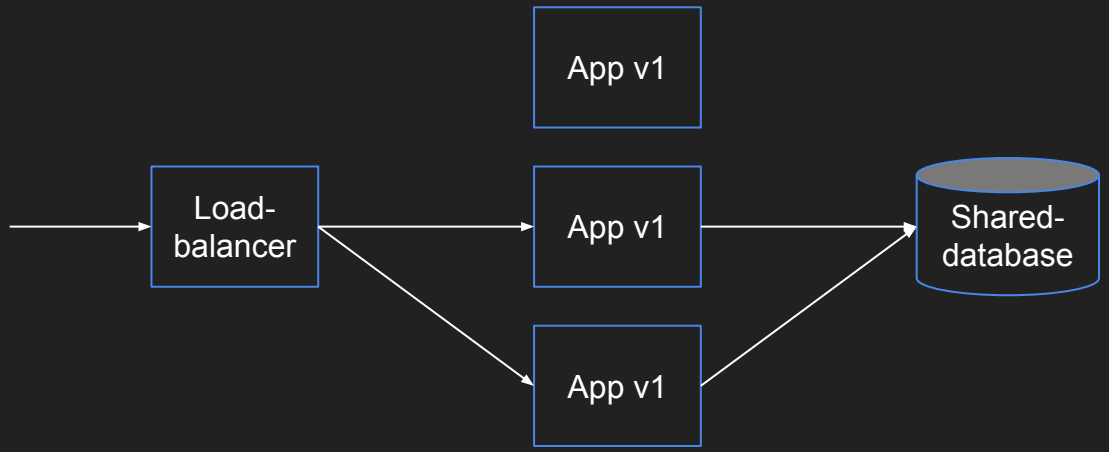
Blue/green deployments



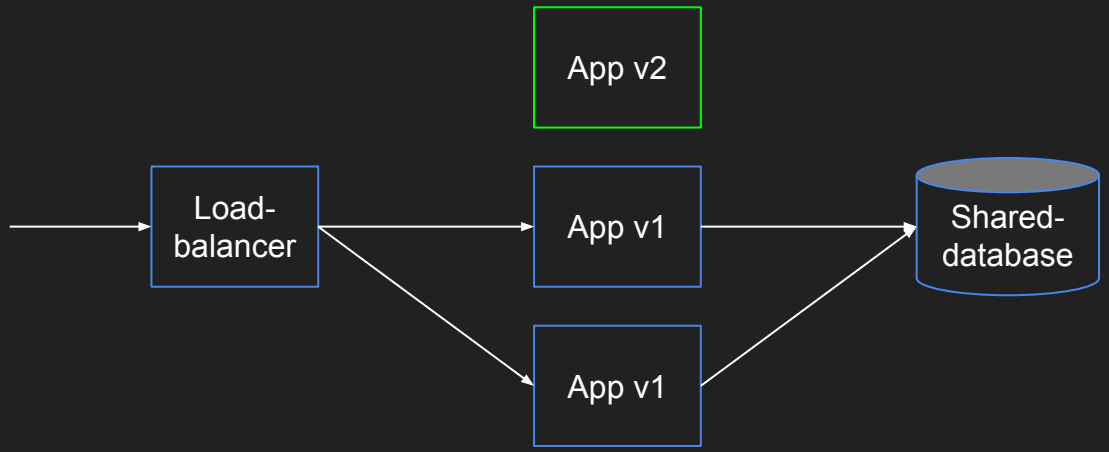
Blue/green deployments



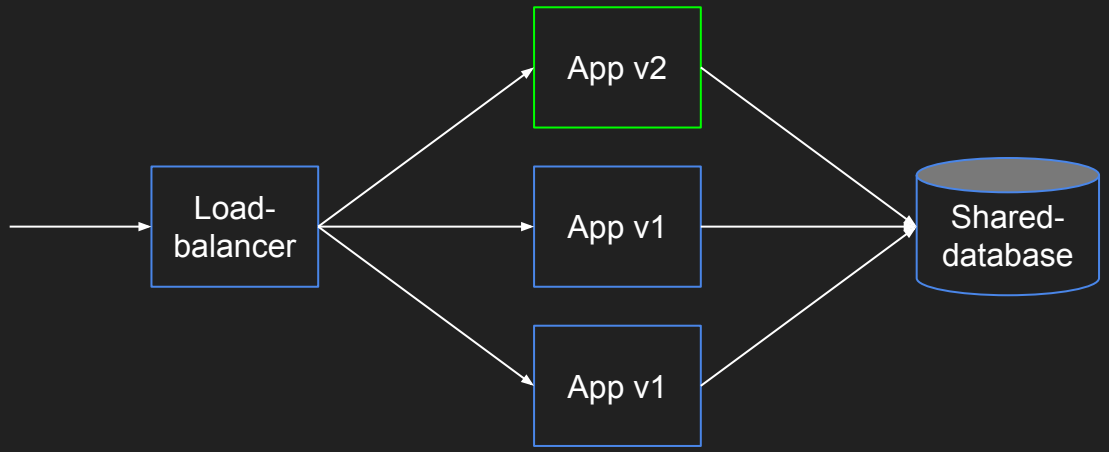
Blue/green deployments



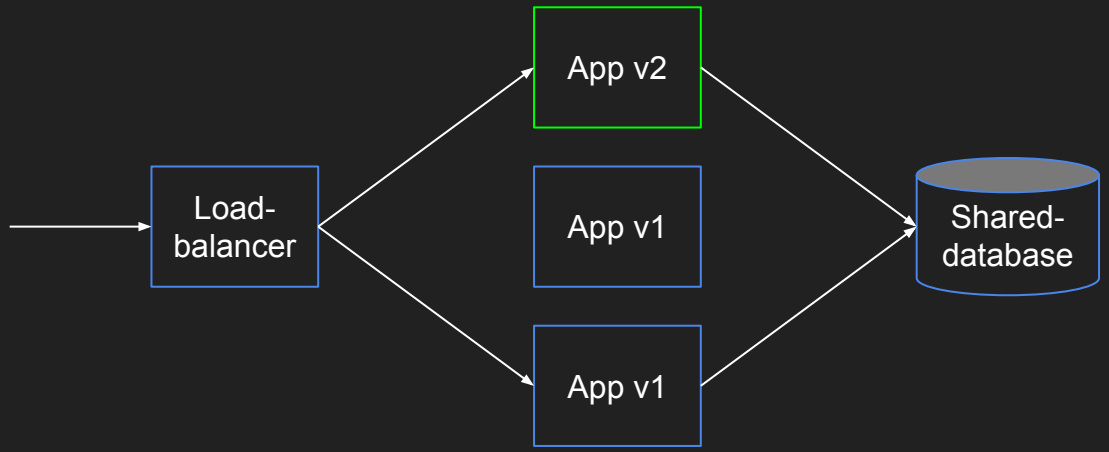
Blue/green deployments



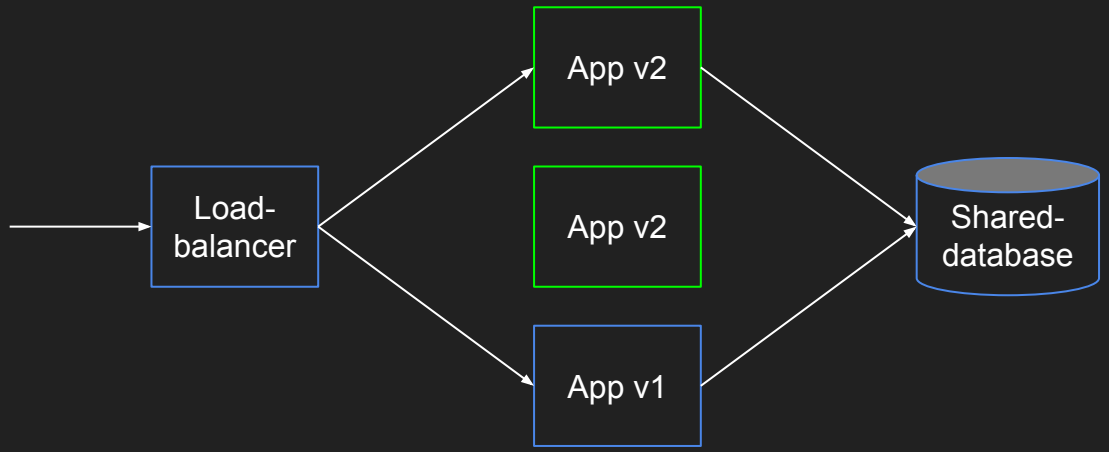
Blue/green deployments



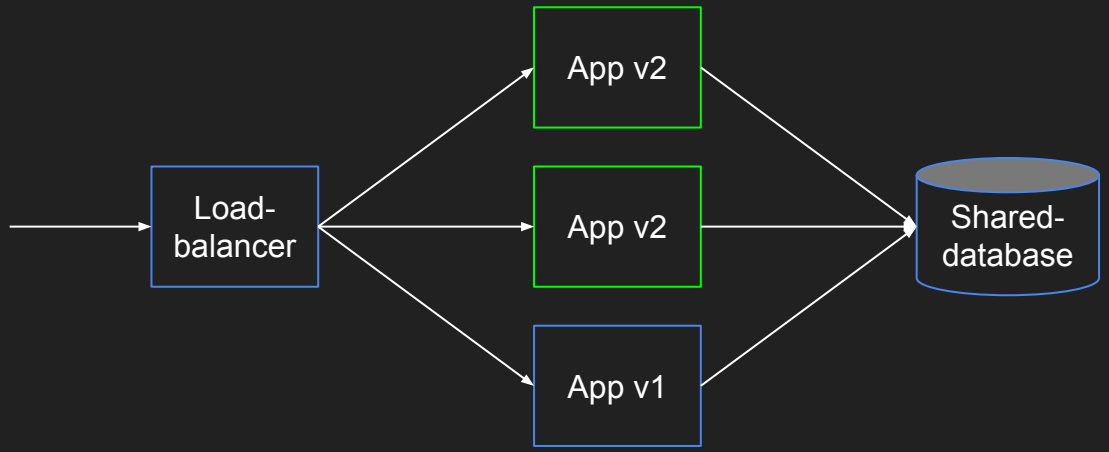
Blue/green deployments



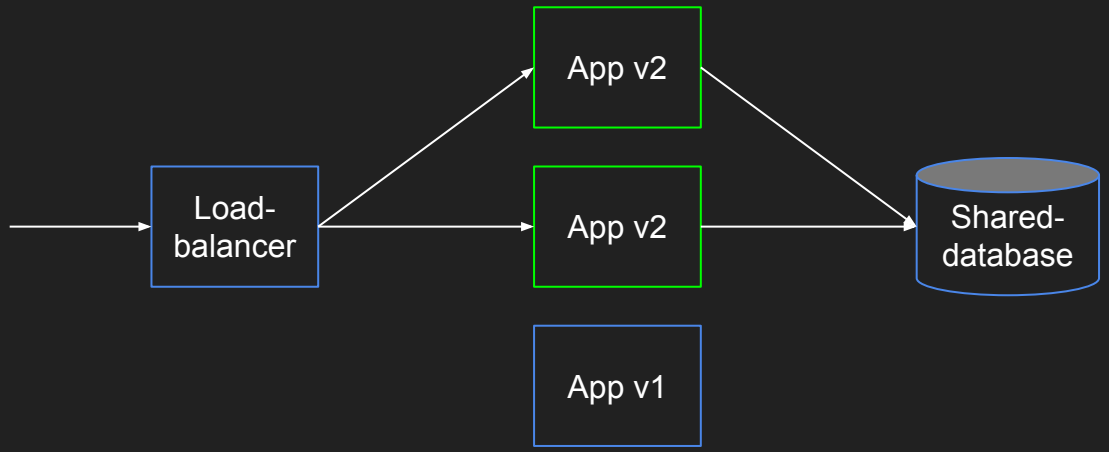
Blue/green deployments



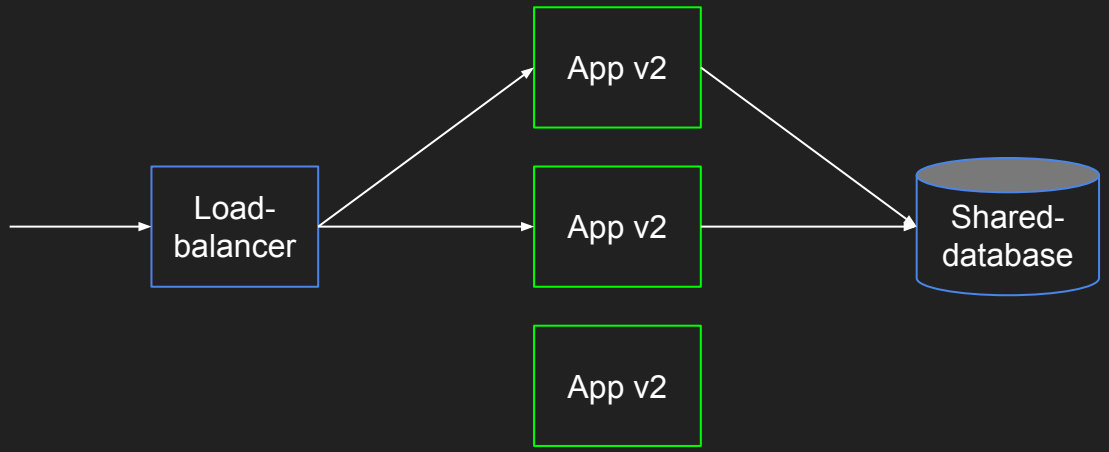
Blue/green deployments



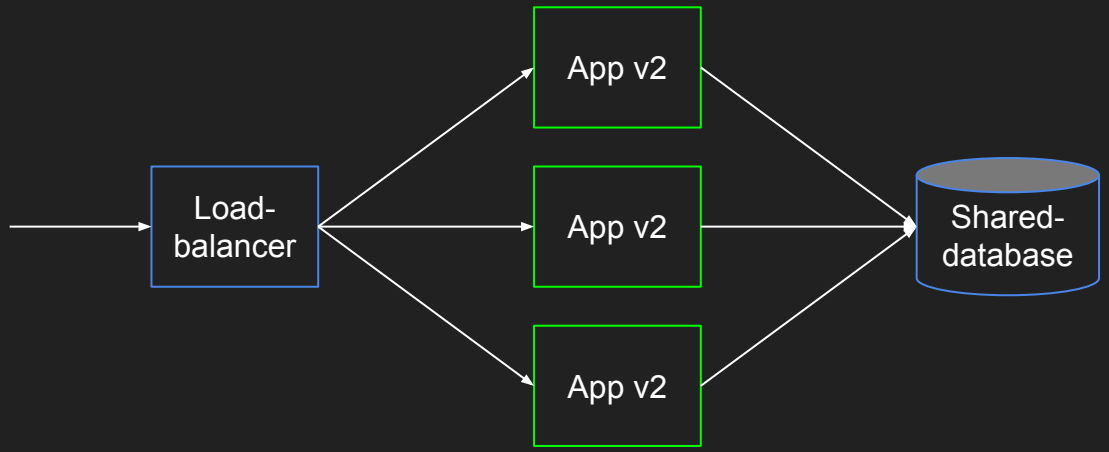
Blue/green deployments



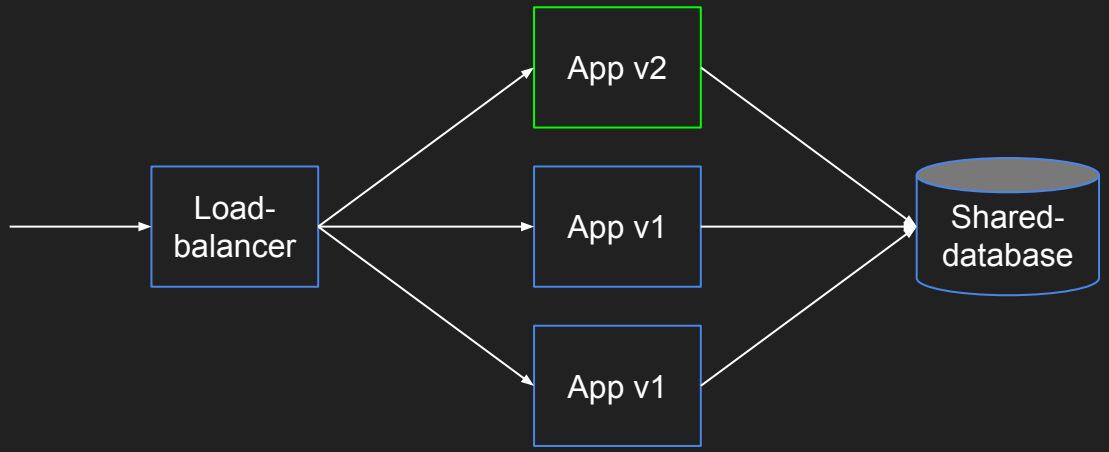
Blue/green deployments



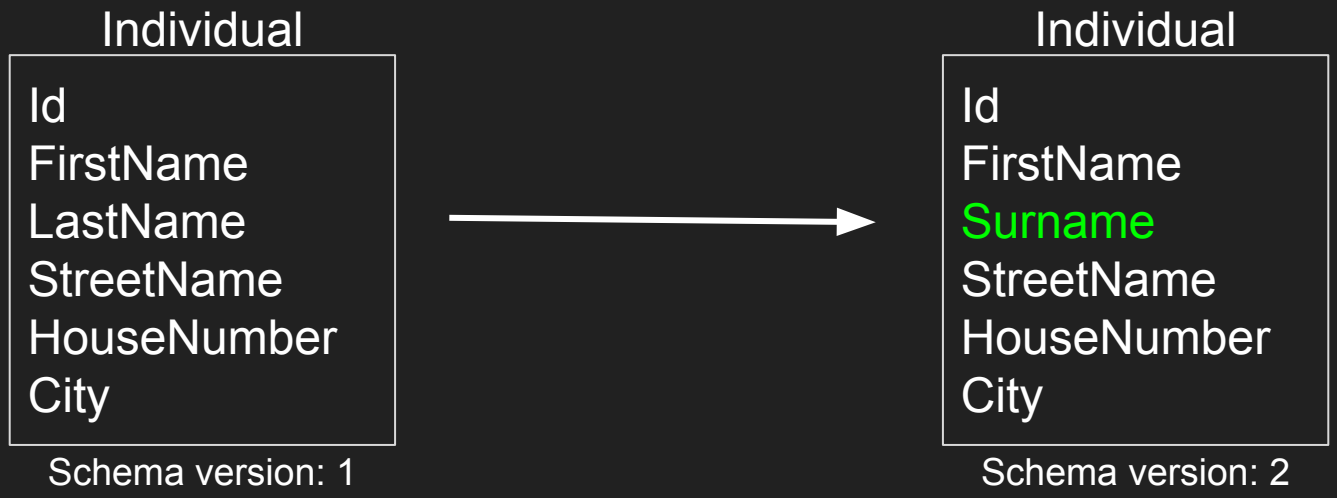
Blue/green deployments



And now with DB changes… oh yeah



Example case



The “naive approach”

App v1

```
setLastName(n) {
 lastName = n;
}
```

```
getLastName() {
 return lastName;
}
```

App v2

```
setSurname(n){
 surname = n;
 lastName = n;
}
```

```
getSurname(){
 return surname == null ?
   lastName :
   surname;
}
```

App v3

```
setSurname(n) {
 surname = n;
}
```

```
getSurname() {
 return surname;
}
```

The naive approach - rollout process

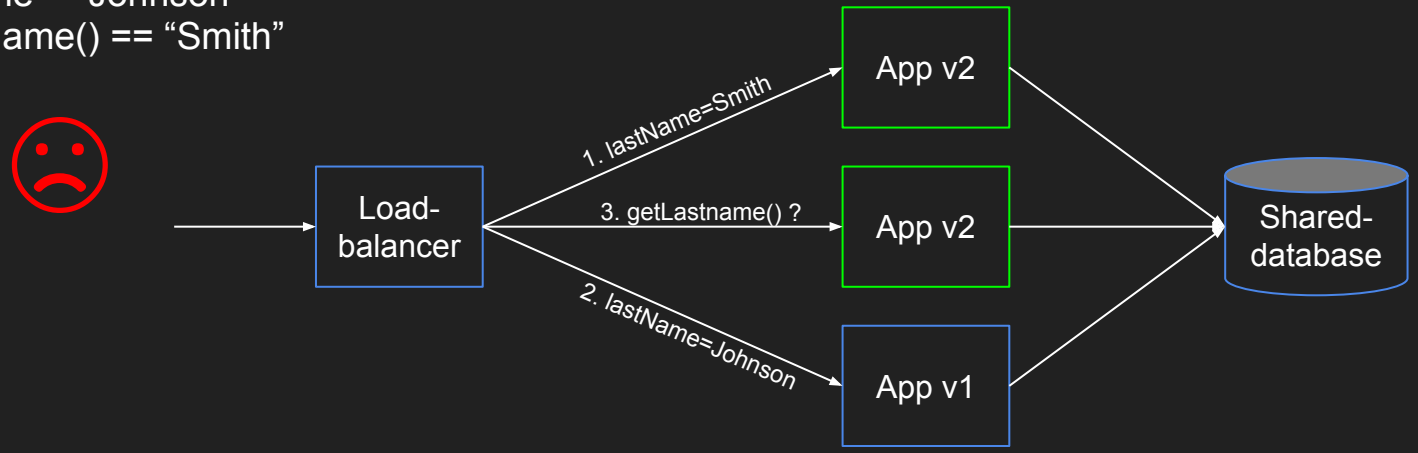
1. Add new field ***surname*** to schema
2. Deploy app version 2 using green/blue deployment
3. Copy all values from lastName -> surname
4. Deploy app version 3 using green/blue deployment
5. Remove unused field ***lastName*** from schema

… Easy, or?

Why is the “*naive approach*” broken?

ID	lastName	surname
1	Johnson	Smith

1. App v2 writes: (lastName, surname) = “Smith”
2. App v1 writes: lastName = “Johnson”
3. App v2 reads: getSurname() == “Smith”



Systek

Conclusion: Cannot use the field value as an indication for which version an object is at

```
getSurname(){
 return surname == null ?
   lastName :
   surname;
}
```

Systek

Store the version of the object

explicitly

along with the object

Fixed

Individual

- Id
- **version**
- FirstName
- LastName
- Surname
- StreetName
- HouseNumber
- City

App v1

```
setLastName(n) {
  lastName = n;
  version = 1;
}
```

```
getLastName() {
  return lastName;
}
```

App v2

```
setSurname(n) {
  surname = n;
  lastName = n;
  version = 2;
}
```

```
getSurname() {
  return version == 1 ?
    lastName :
    surname;
}
```

App v3

```
setSurname(n) {
  surname = n;
  version = 2;
}
```

```
getSurname() {
  return surname;
}
```

Systek

Decouple DB version / app version

Individual

Id
Version
FirstName
LastName
Surname
StreetName
HouseNumber
City

App v1

```
setLastName(n) {
  version = 1;
  lastName = n;
}
```

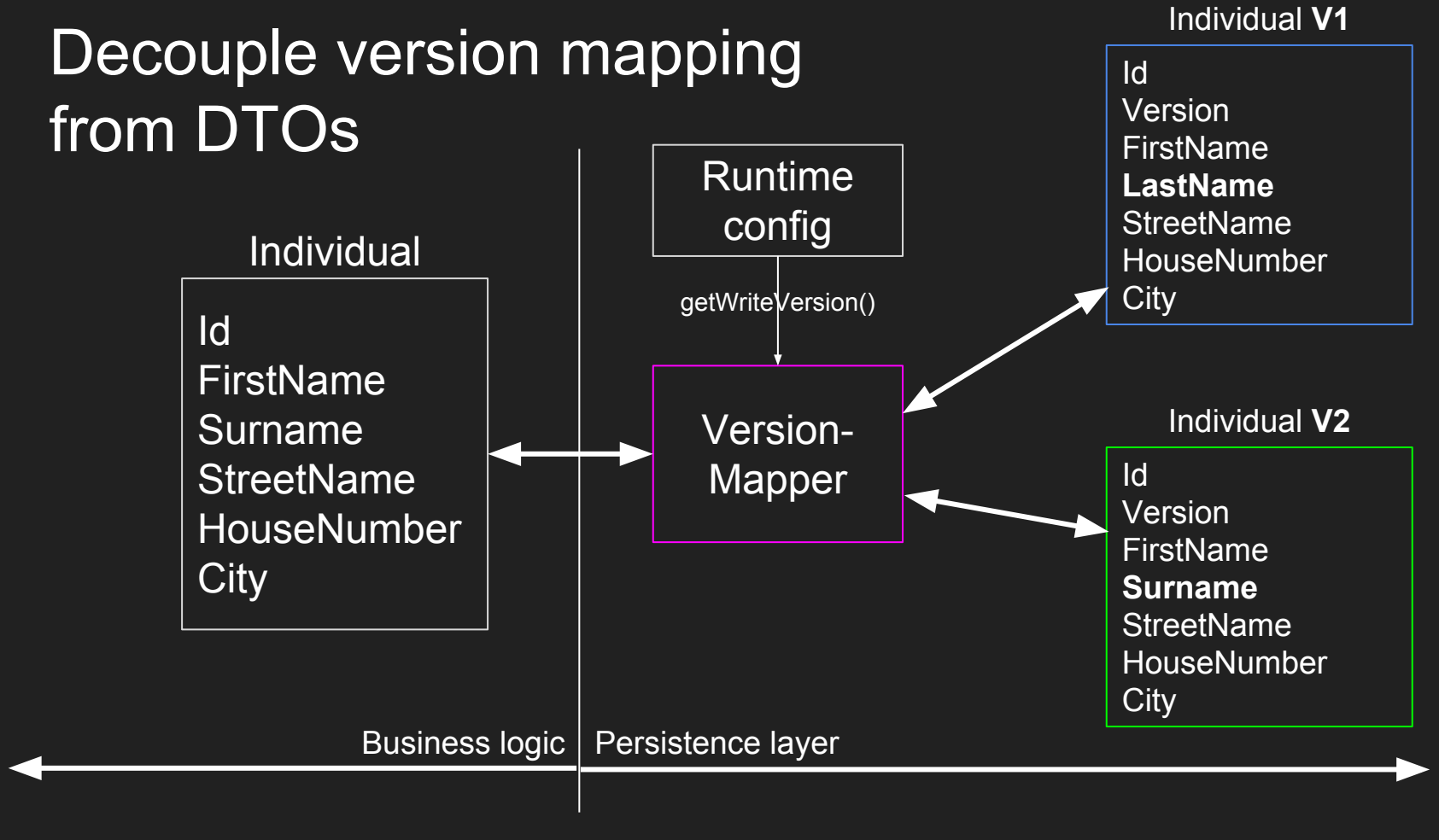
```
getLastName() {
  return lastName;
}
```

App v2

```
setSurname(n) {
  version = config.dbVersion();
  if (version == 2)
    surname = n;
  else
    lastName = n;
}
```

```
getSurname() {
  return version == 1 ?
    lastName :
    surname;
}
```

Decouple version mapping from DTOs



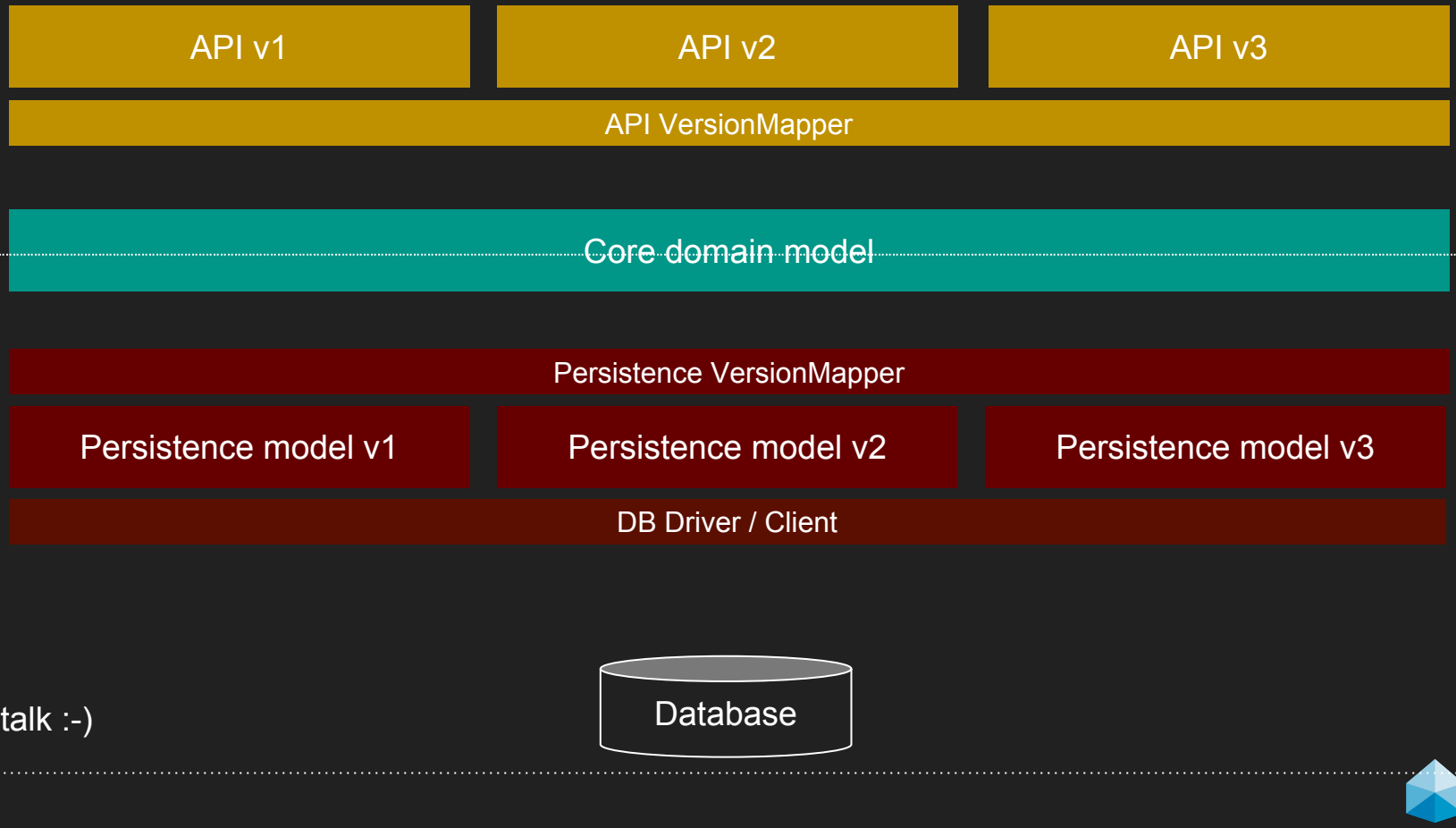
Systek

VersionMapper:

```
// upgrade
if (targetVersion > entity.getSchemaVersion()) {
    for (int v = entity.getSchemaVersion(); v < targetVersion; v++) {
        entity = translators.get(v).upgrade(entity);
    }
}

// downgrade
else {
    for (int v = entity.getSchemaVersion(); v > targetVersion; v--) {
        entity = translators.get(v).downgrade(entity);
    }
}
```

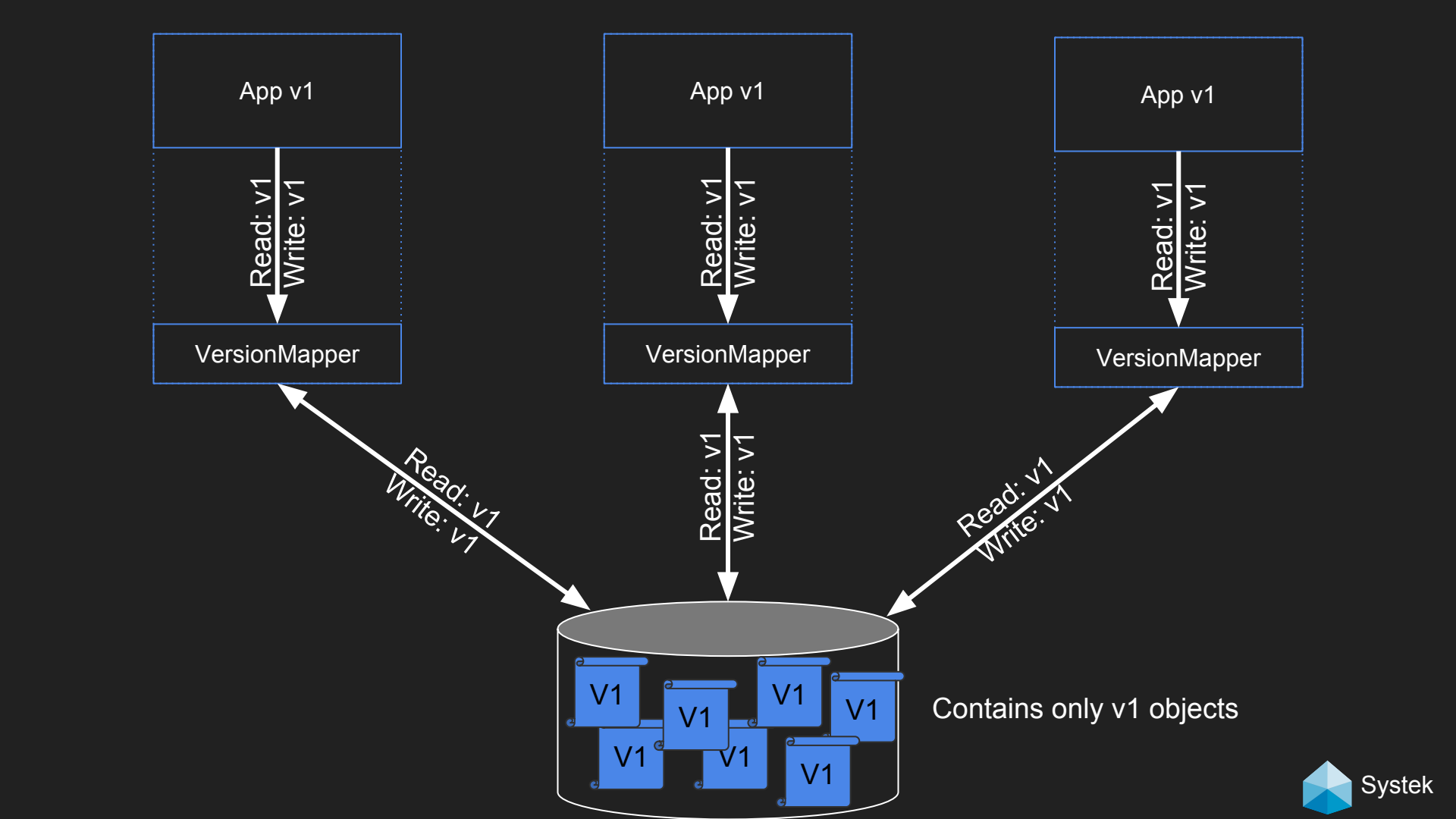
My talk @ JavaZone 2015

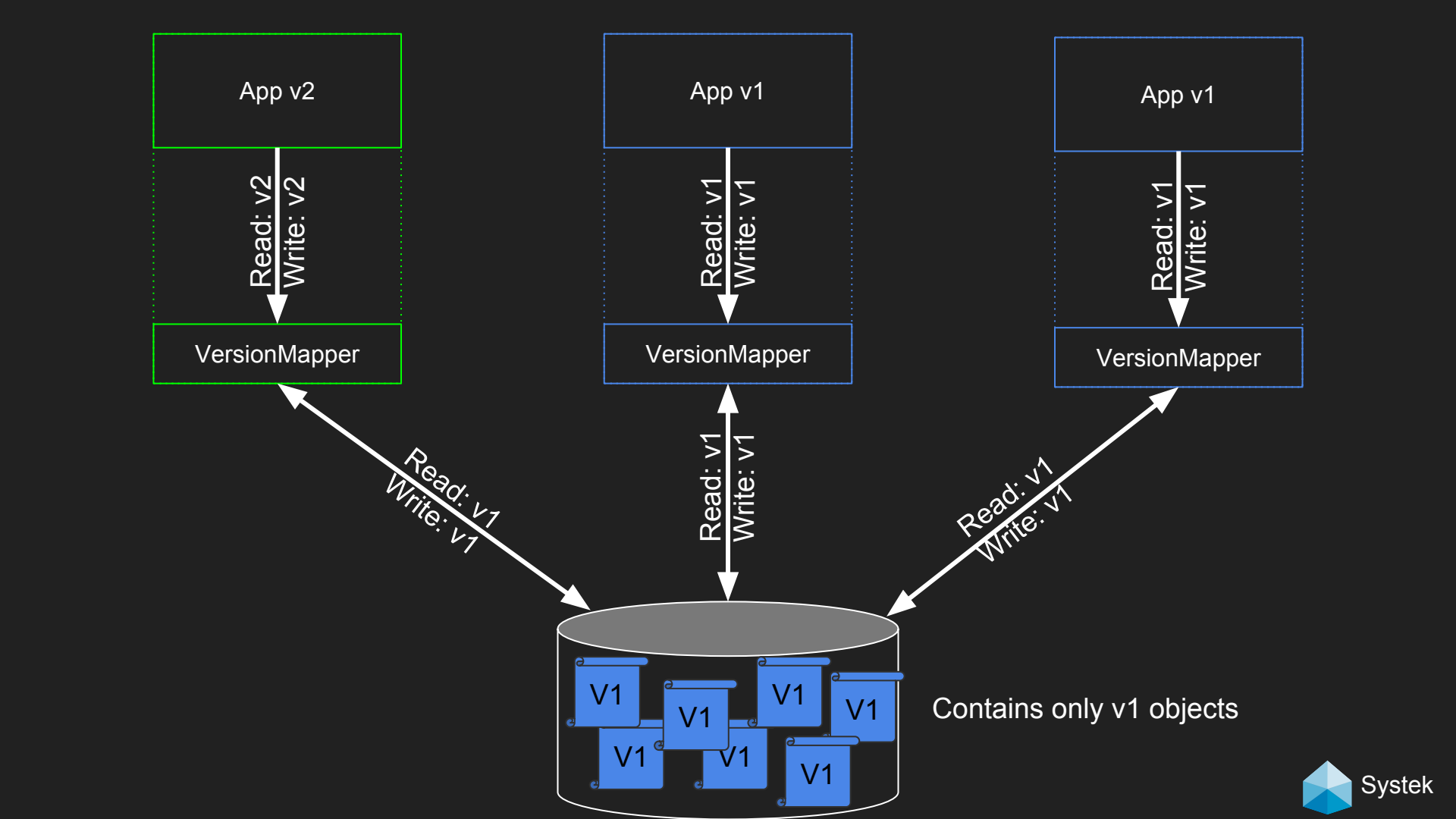


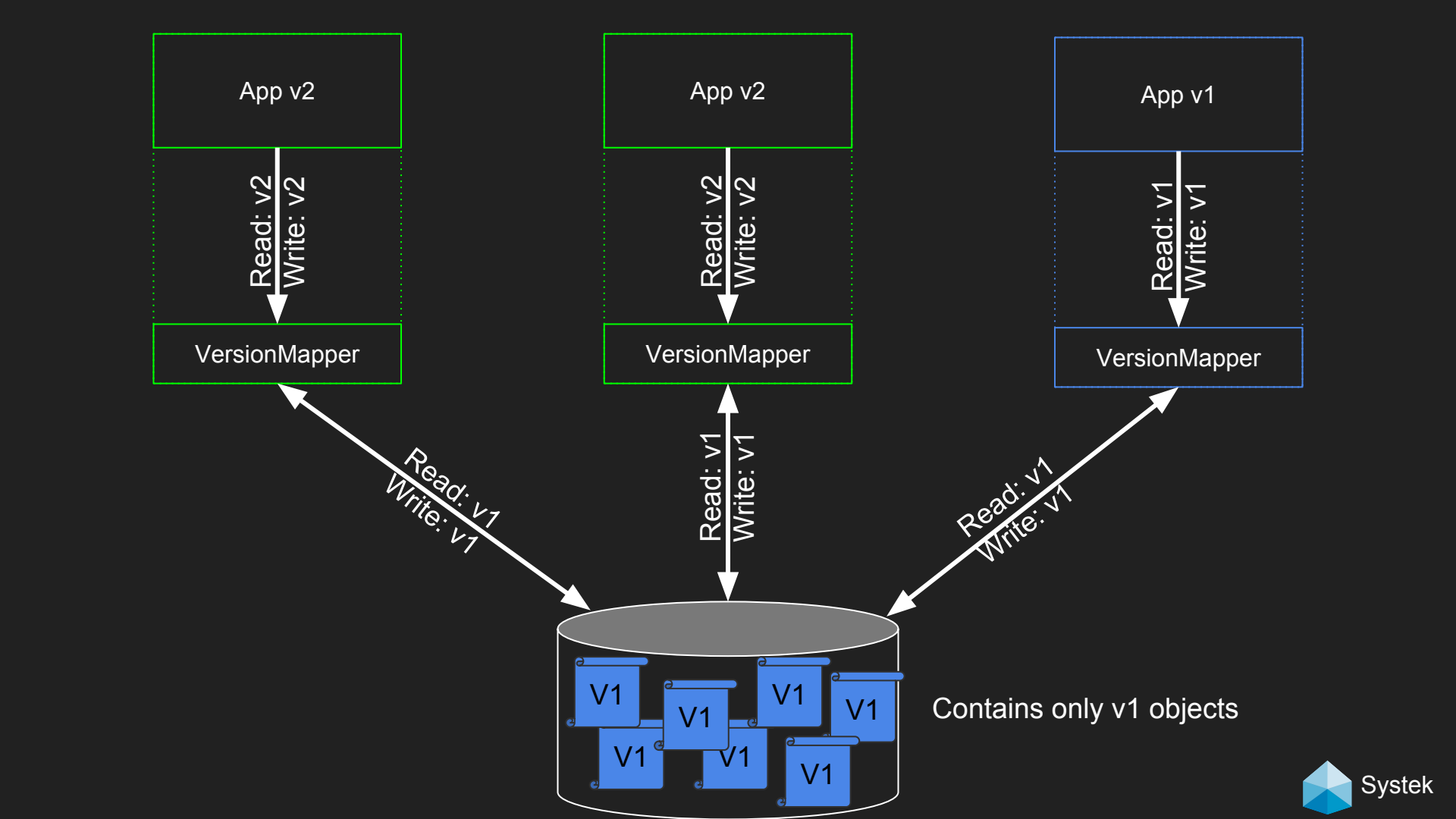
This talk :-)

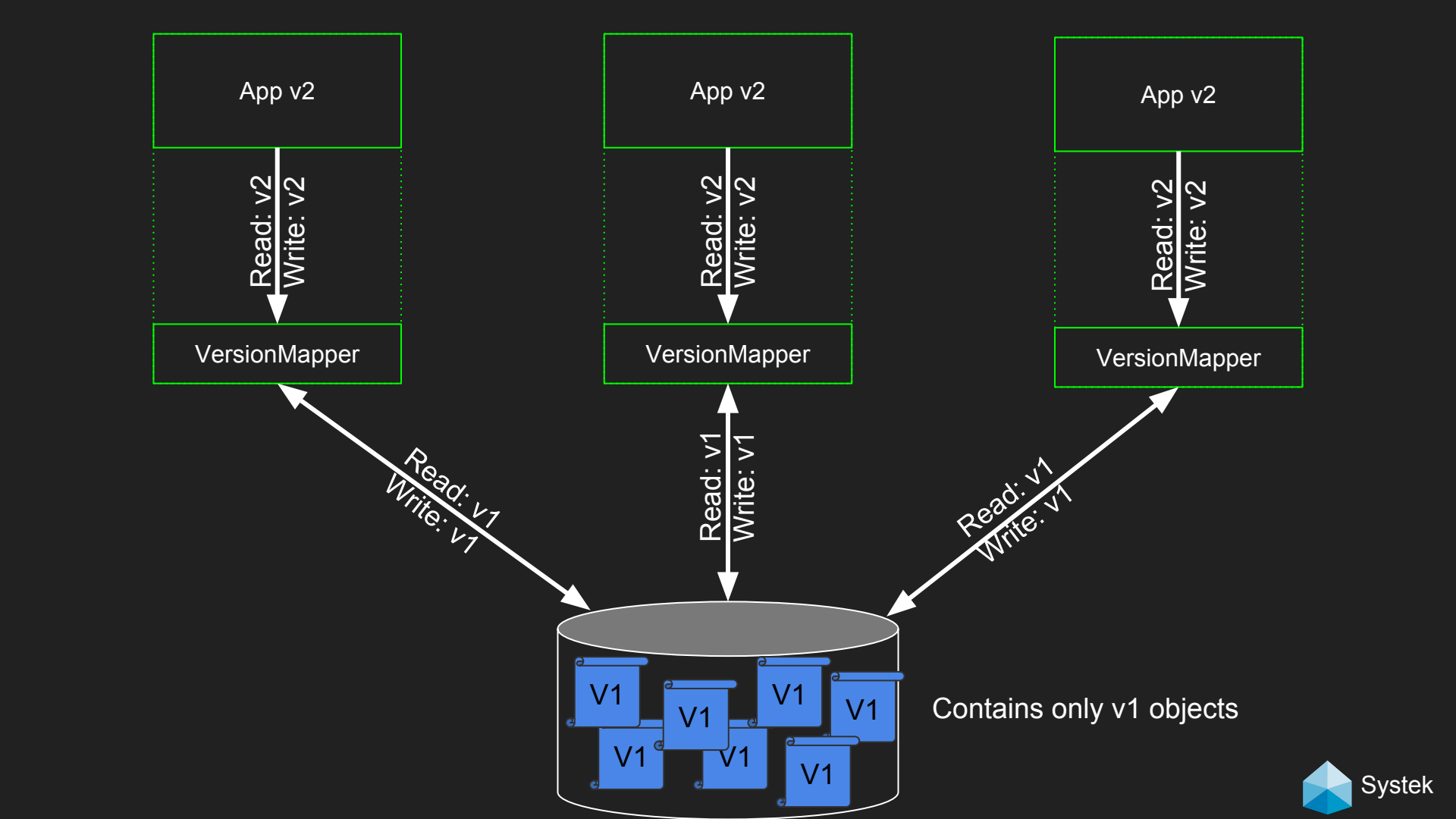
Systek

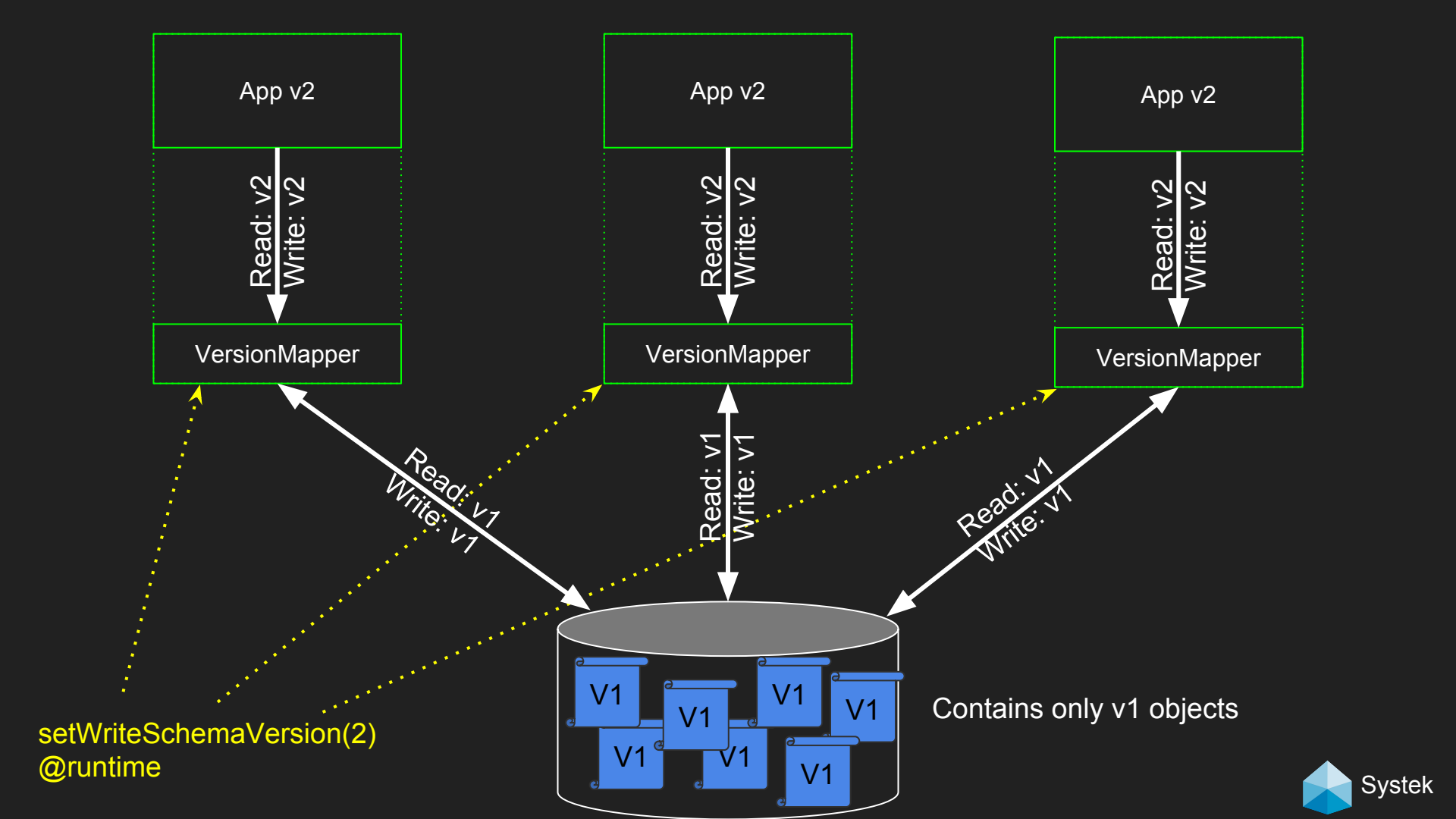
Improved rollout process

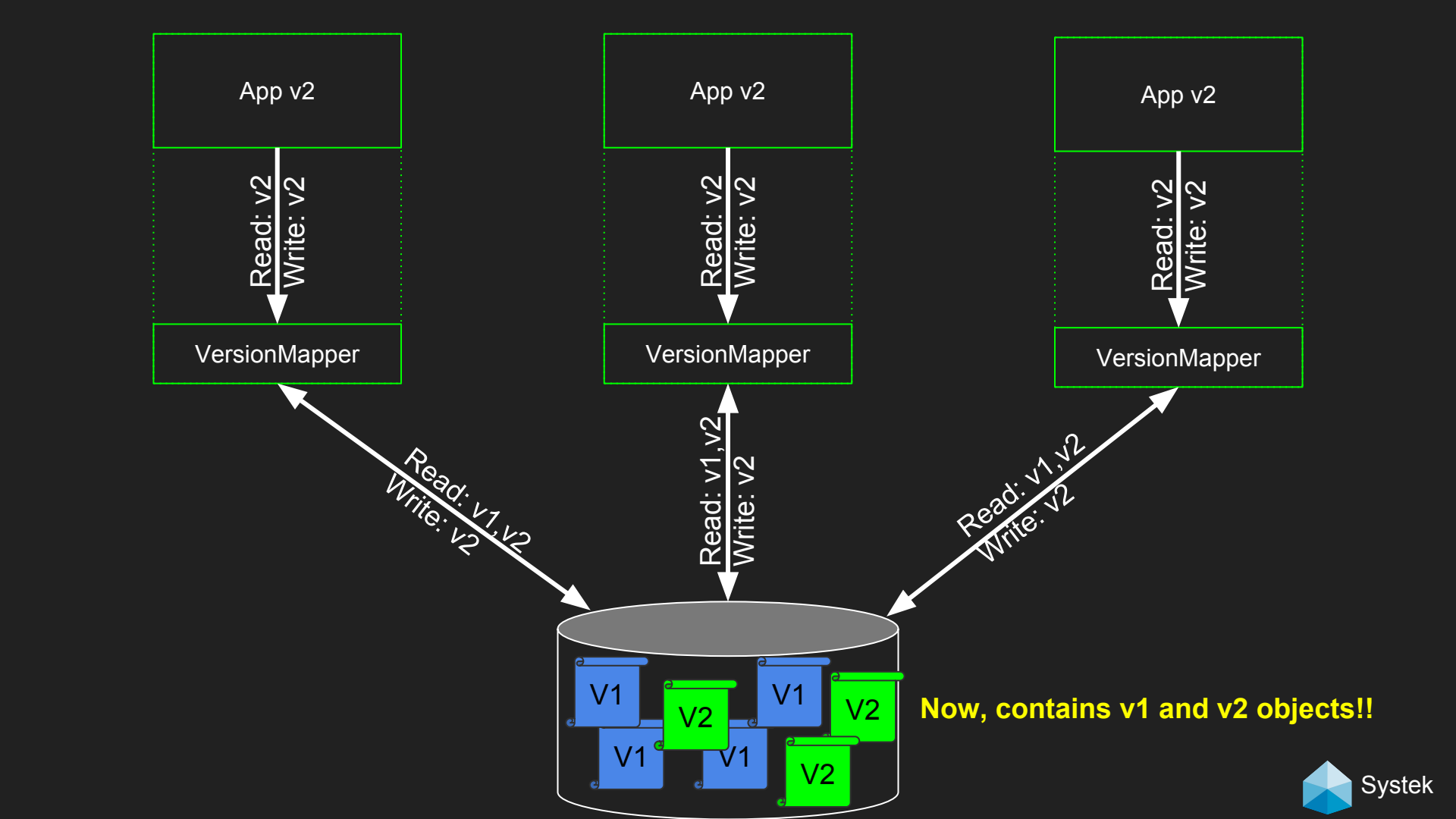

App v1
App v1
App v1
Read: v1
Write: v1
Read: v1
Write: v1
Read: v1
Write: v1
VersionMapper
VersionMapper
VersionMapper
Read: v1
Write: v1
Read: v1
Write: v1
Read: v1
Write: v1
V1
V1
V1
V1
V1
V1
V1
Contains only v1 objects
Systek

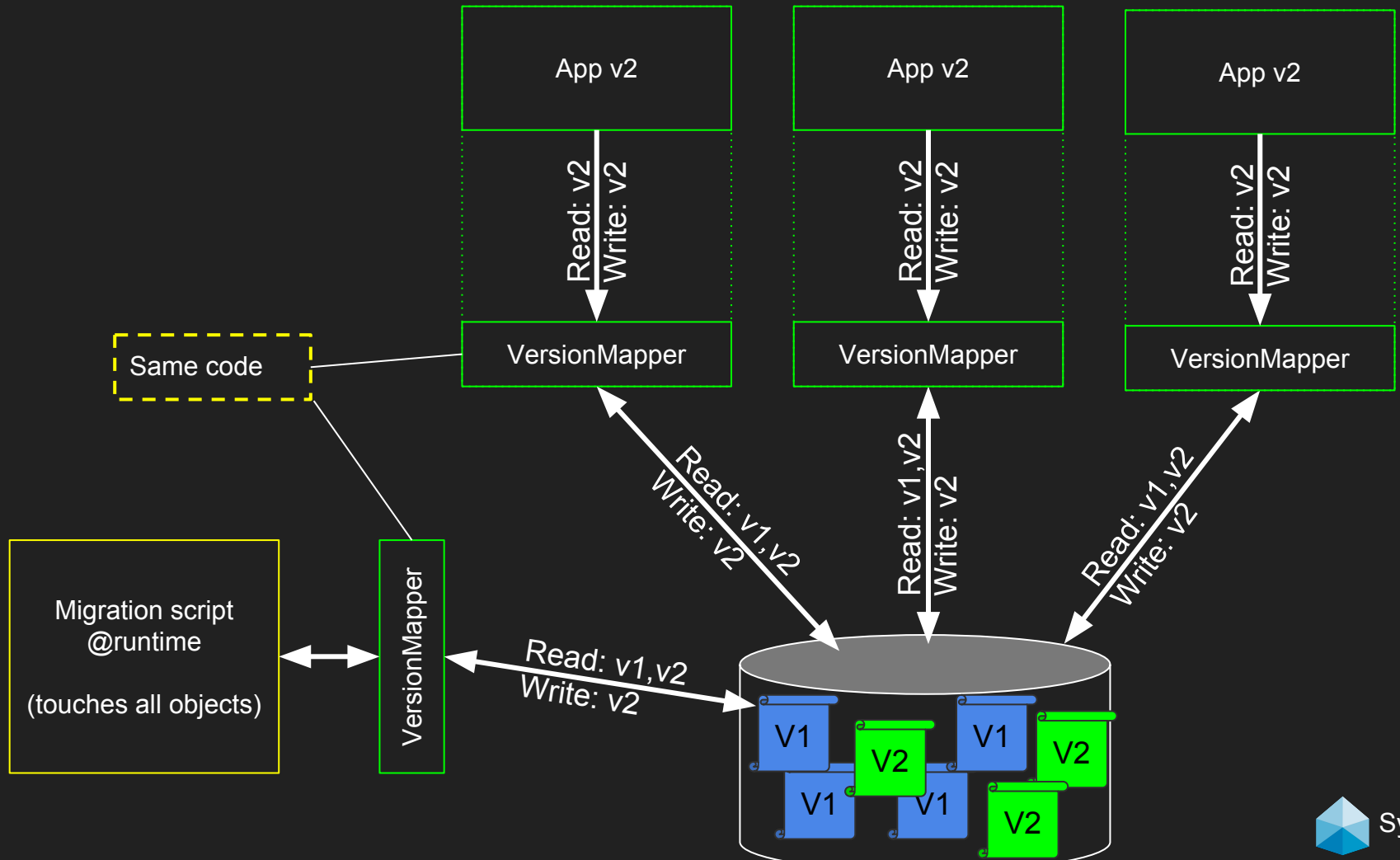

App v2
App v1
App v1
Read: v2
Write: v2
Read: v1
Write: v1
Read: v1
Write: v1
VersionMapper
VersionMapper
VersionMapper
Read: v1
Write: v1
Read: v1
Write: v1
Read: v1
Write: v1
V1
V1
V1
V1
V1
V1
V1
Contains only v1 objects
Systek

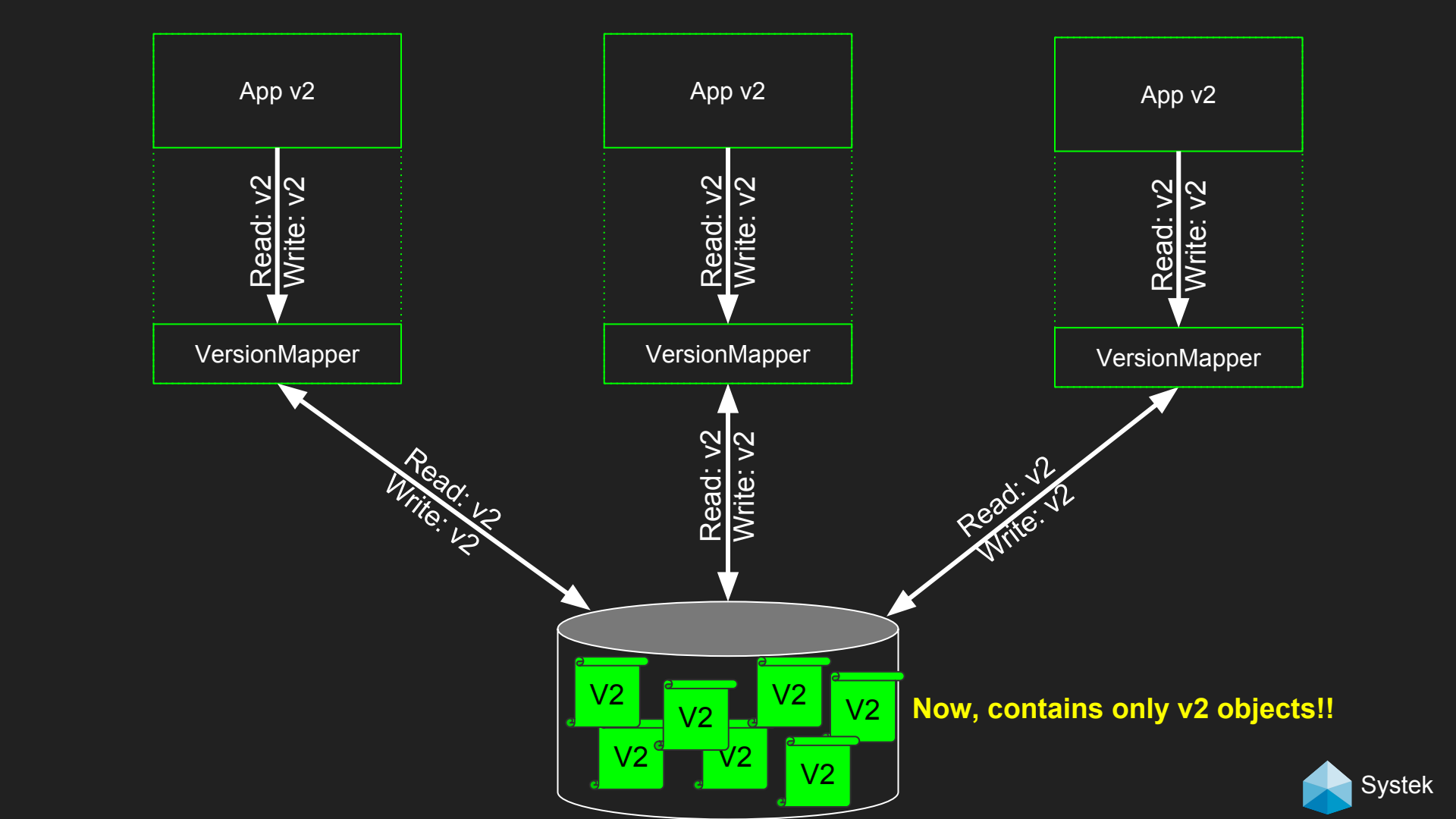

App v2
Read: v2
Write: v2
VersionMapper
App v2
Read: v2
Write: v2
VersionMapper
App v1
Read: v1
Write: v1
VersionMapper
Read: v1
Write: v1
Read: v1
Write: v1
Read: v1
Write: v1
V1
V1
V1
V1
V1
V1
V1
Contains only v1 objects
Systek


App v2
App v2
App v2
Read: v2
Write: v2
Read: v2
Write: v2
Read: v2
Write: v2
VersionMapper
VersionMapper
VersionMapper
Read: v1
Write: v1
Read: v1
Write: v1
Read: v1
Write: v1
V1
V1
V1
V1
V1
V1
V1
Contains only v1 objects
Systek


App v2
App v2
App v2
Read: v2
Write: v2
Read: v2
Write: v2
Read: v2
Write: v2
VersionMapper
VersionMapper
VersionMapper
Read: v1
Write: v1
Read: v1
Write: v1
Read: v1
Write: v1
V1
V1
V1
V1
V1
V1
V1
setWriteSchemaVersion(2)
@runtime
Contains only v1 objects
Systek


App v2
App v2
App v2
Read: v2
Write: v2
Read: v2
Write: v2
Read: v2
Write: v2
VersionMapper
VersionMapper
VersionMapper
Read: v1,v2
Write: v2
Read: v1,v2
Write: v2
Read: v1,v2
Write: v2
V1
V2
V1
V2
V1
V1
V2
Now, contains v1 and v2 objects!!
Systek


App v2
App v2
App v2
Read: v2
Write: v2
Read: v2
Write: v2
Read: v2
Write: v2
Same code
VersionMapper
VersionMapper
VersionMapper
Read: v1,v2
Write: v2
Read: v1,v2
Write: v2
Read: v1,v2
Write: v2
Migration script
@runtime
(touches all objects)
VersionMapper
Read: v1,v2
Write: v2
V1
V2
V1
V2
V1
V1
V2
Systek


App v2
App v2
App v2
Read: v2
Write: v2
Read: v2
Write: v2
Read: v2
Write: v2
VersionMapper
VersionMapper
VersionMapper
Read: v2
Write: v2
Read: v2
Write: v2
Read: v2
Write: v2
V2
V2
V2
V2
V2
V2
V2
Now, contains only v2 objects!!
Systek

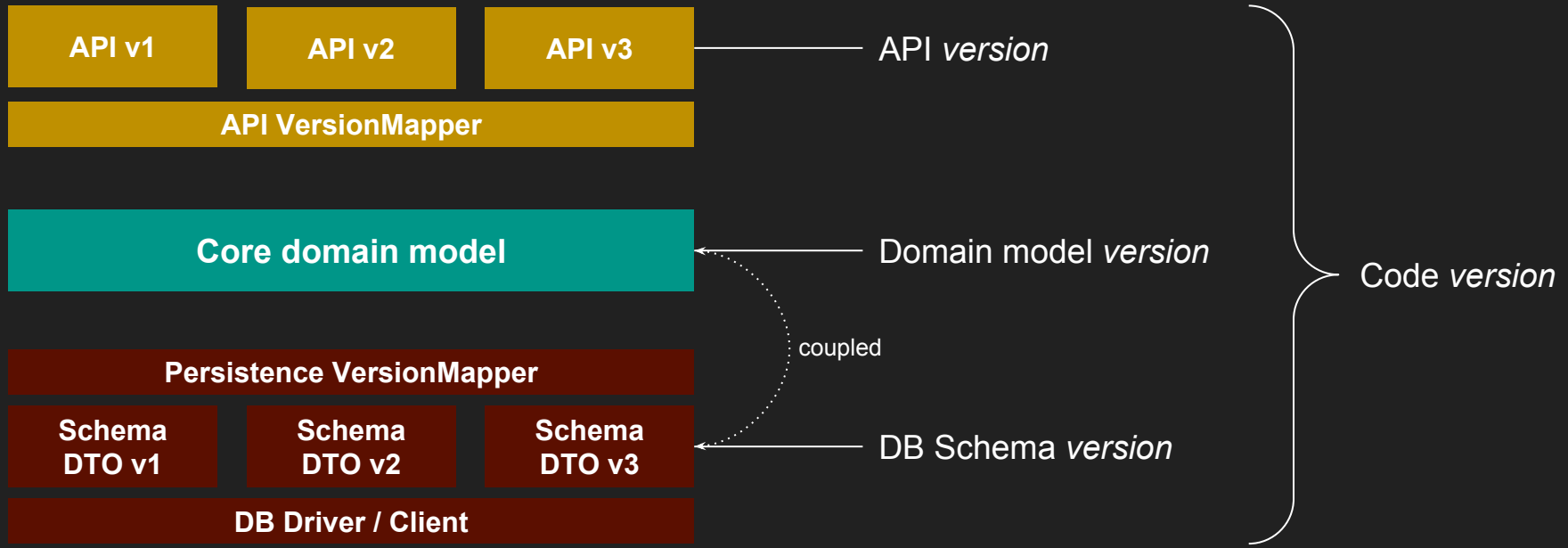
Summary rollout

1. Deploy app v2 using green/blue deployment
2. Add new field ***surname*** to schema
3. Set *config.dbVersion=2* @runtime
4. Run migration script to upgrade all objects to schema *version=2*
5. Remove unused field ***lastName*** from schema

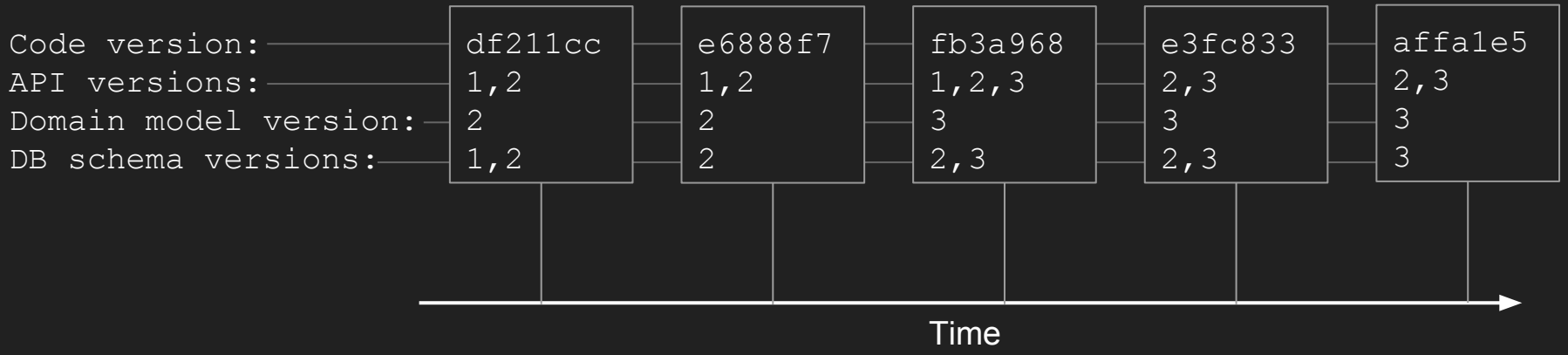
- No second round deployment needed
- DB Schema changes done independently from app deploy

Versioning schemes

A word about versioning schemes



Versioning example



Which database technology?

Doing this in SQL is hard(er)

- Problem: objects have different schema versions
- Need two queries:
  - Read the object's version
  - Create a version-specific query to fetch the target object
- Both queries need to be atomic
- Requires transaction isolation level "SERIALIZABLE" and "SELECT … FOR UPDATE"

Doing this in SQL is hard(er) - 2

- ALTER TABLE require table lock
- ALTER TABLE cause pauses (=unavailability)
- Disable constraints/indexes
- What about triggers and stored procedures?

Doing this in SQL is hard(er) - 3

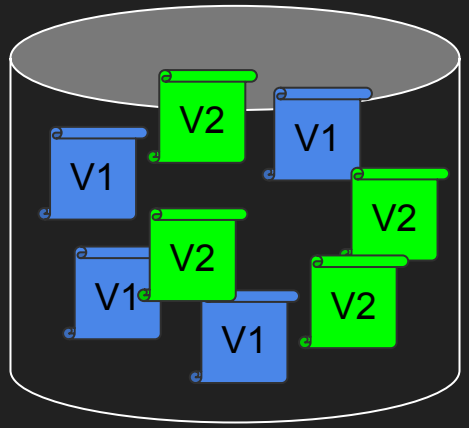
- SQL schemas are not meant to be used for a *per-object* schema
- Hard to change datatype: String birthdate -> Int birthdate
- Need to use temporarily fields/tables which result is a more complex migration process
- .. which limits the number of versions you can roll back to!

noSQL is more suited for the job

- Supports per-object schema
- If you can, just use a key/value store
- Use CAS (compare-and-swap) as the transaction primitive
- Optimistic locking for all transactions
- Better performance without compromising consistency
- Schema deploy easier, no need to run ALTER TABLE - just start writing in the new format

noSQL is more suited for the job

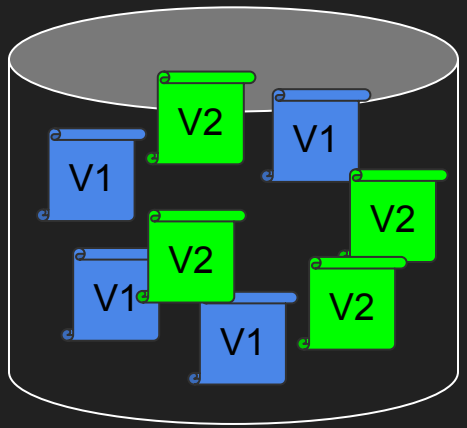
- Need to fetch the object only once
- The version can be stored inside the same object in a header
- Before unmarshalling the entire object, parse only the header to learn the version
- Keep as many historic schema versions “around” as desired for the rollback strategy
- Remove old versions from code and DB as suited



How to query across objects?

How to query across objects?

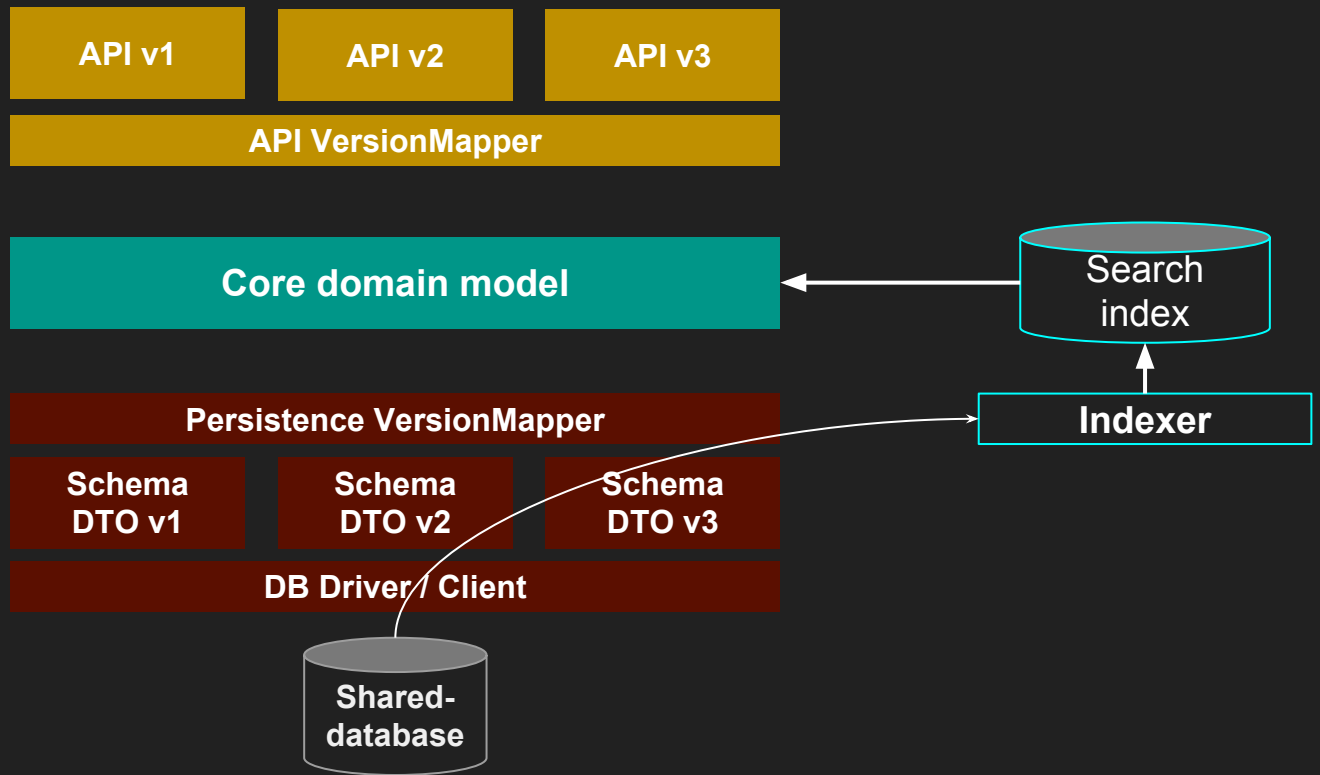
- One query per schema version and merge the results?
- What about paging and sorting
- Best to avoid queries across object at all
- Or….



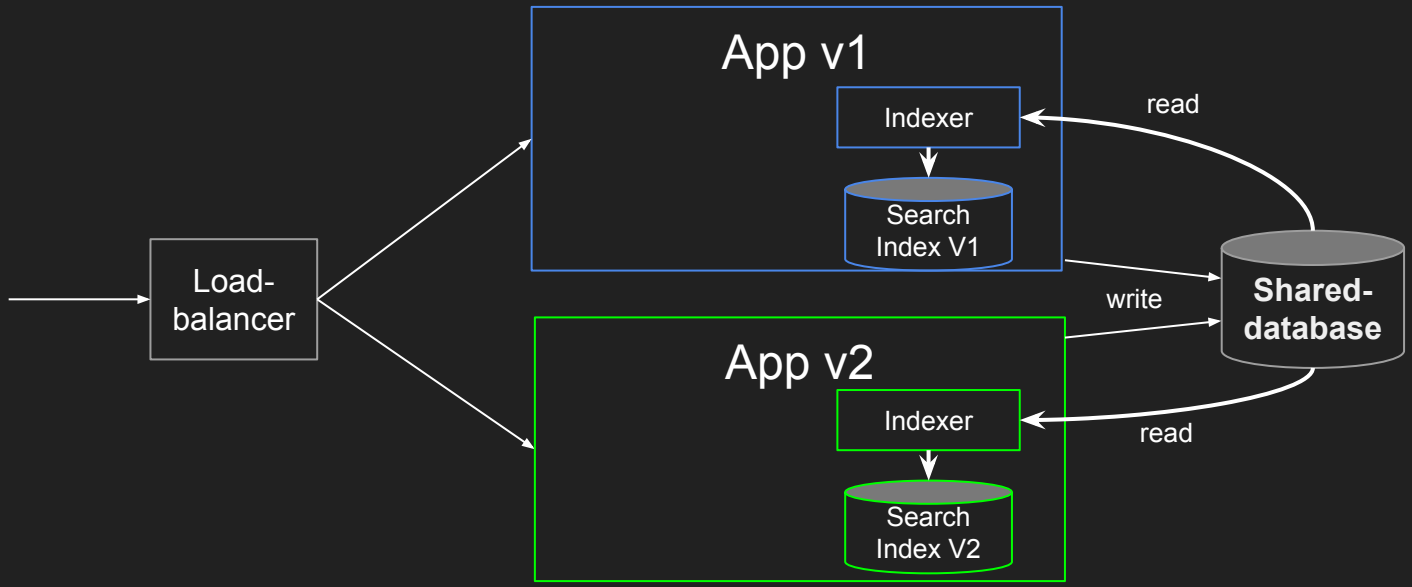
Setup a search index **per**

“*domain model version*”

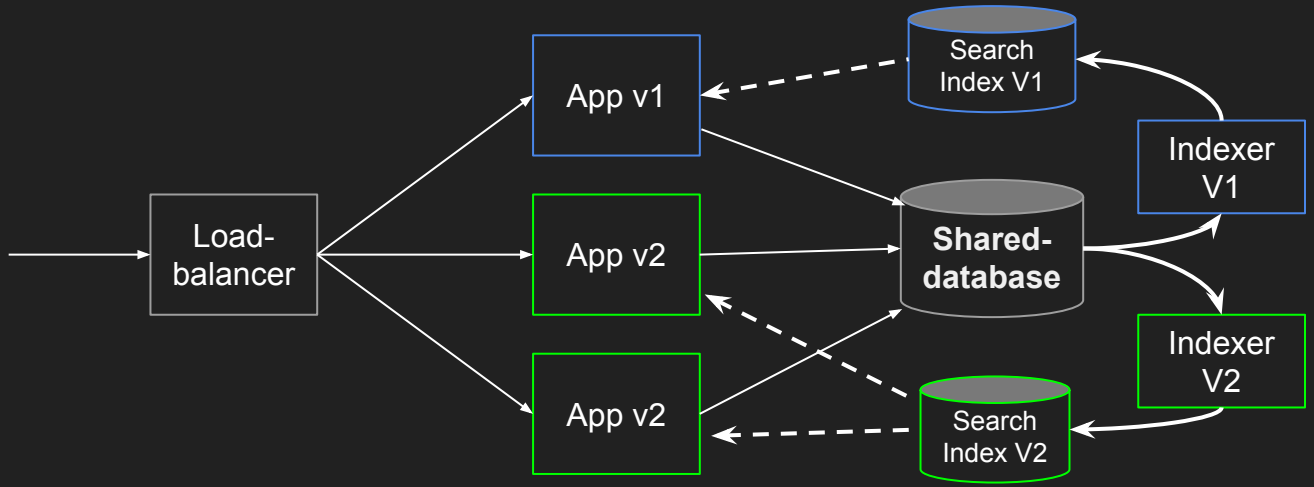
Index coupled to domain model version



Embedded index



External index



Summary

- Can rollout new code without interruptions
- Can rollout db schema changes without interruptions
- Both production- and test environments have high availability
- Can deploy during daytime and sleep at night; in theory :-)
- Correctness during the transition phases
- Can leave the system in an intermediate state if needed
- Can rollback to many versions, not just 1
- Supports business with short time to market
- Embraces refactoring to keep maintenance costs low

Questions?

Thanks

Systek