Retrocompatibilidade

em desenvolvimento de software

Conceitos e definições

- **Retrocompatibilidade**: capacidade de um sistema de funcionar corretamente com versões anteriores de seus componentes;
- ***Fallback***: comportamento alternativo ou “reserva” para quando uma ou mais funcionalidades principais falharem;
- ***Deploy***: mover/enviar o código/versão de um sistema de um ambiente de desenvolvimento para um de produção onde os usuários finais podem utilizar;
- ***Rollback***: reverter uma alteração para um estado anterior considerado estável;
- ***Trigger*** (em banco de dados): procedimento armazenado no próprio banco de dados que é executado quando ocorre um evento;
- **EF (*Entity Framework*)**: conjunto de bibliotecas para mapeamento de objetos relacionais (ORM);
- **Migração**/*migration*: representação das mudanças no código para o banco de dados:
  - objeto/entidade → tabela;
  - propriedade → coluna;
  - *etc*.

Vantagens e desvantagens

Vantagens:

- “*Zero-downtime*” *deploy;*
- Riscos reduzidos no *deploy*;
- Melhor experiência para o usuário;
- etc.

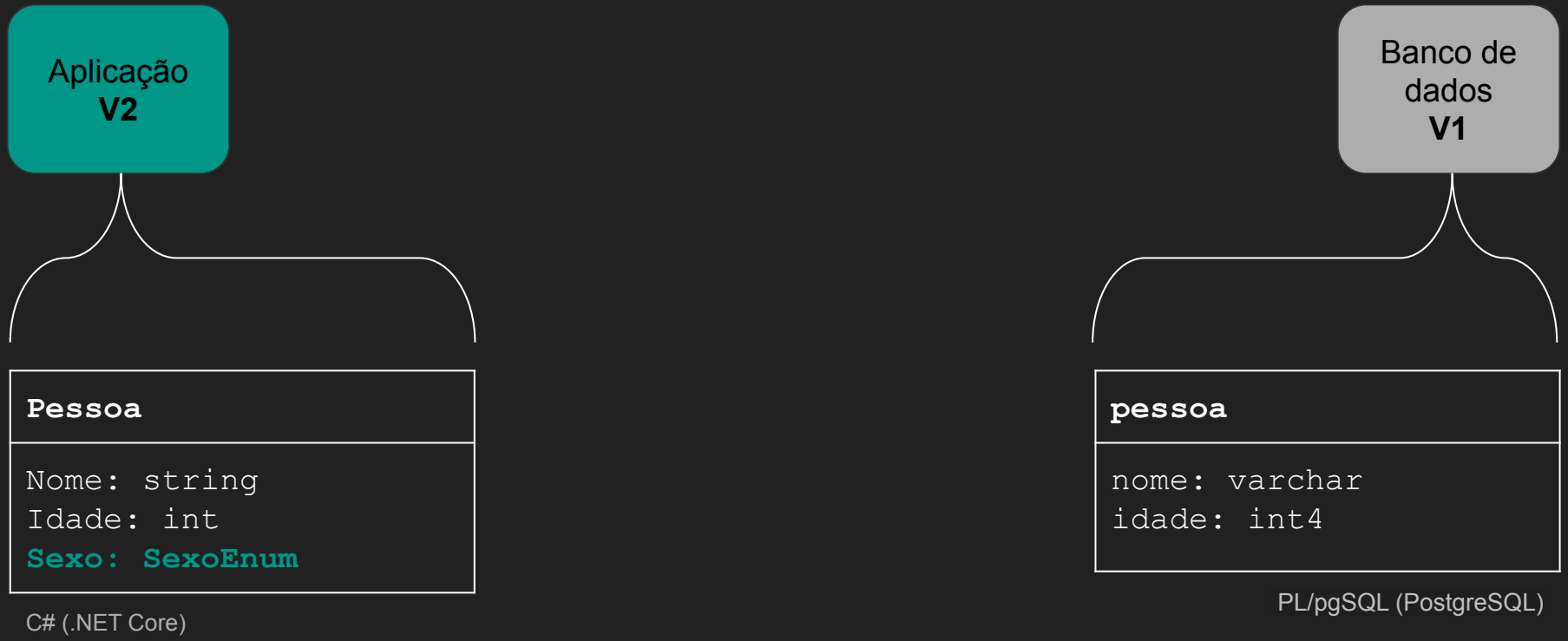
Desvantagens:

- Complexidade aumentada;
- Limpeza de código atrasada;
- Maior demanda de recursos;
- etc.

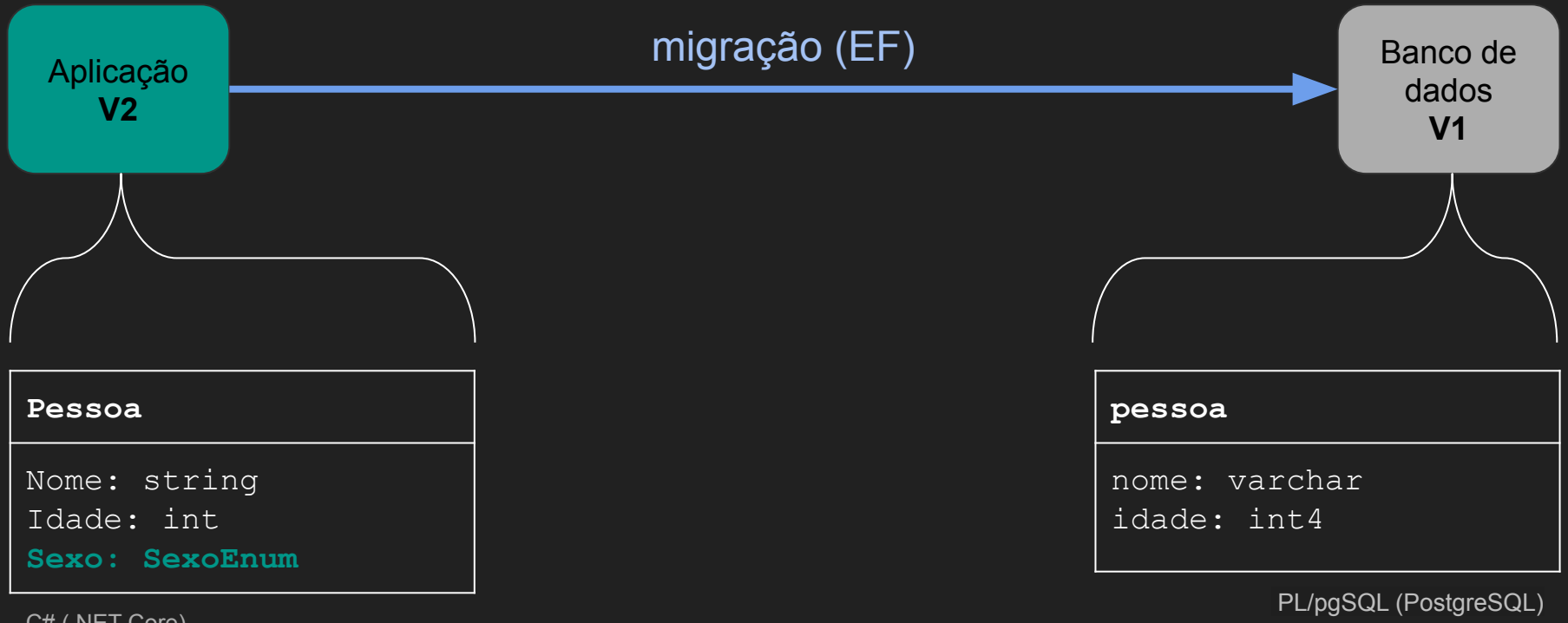
Backend: exemplo **SEM retrocompatibilidade**

Backend: exemplo **SEM retrocompatibilidade**

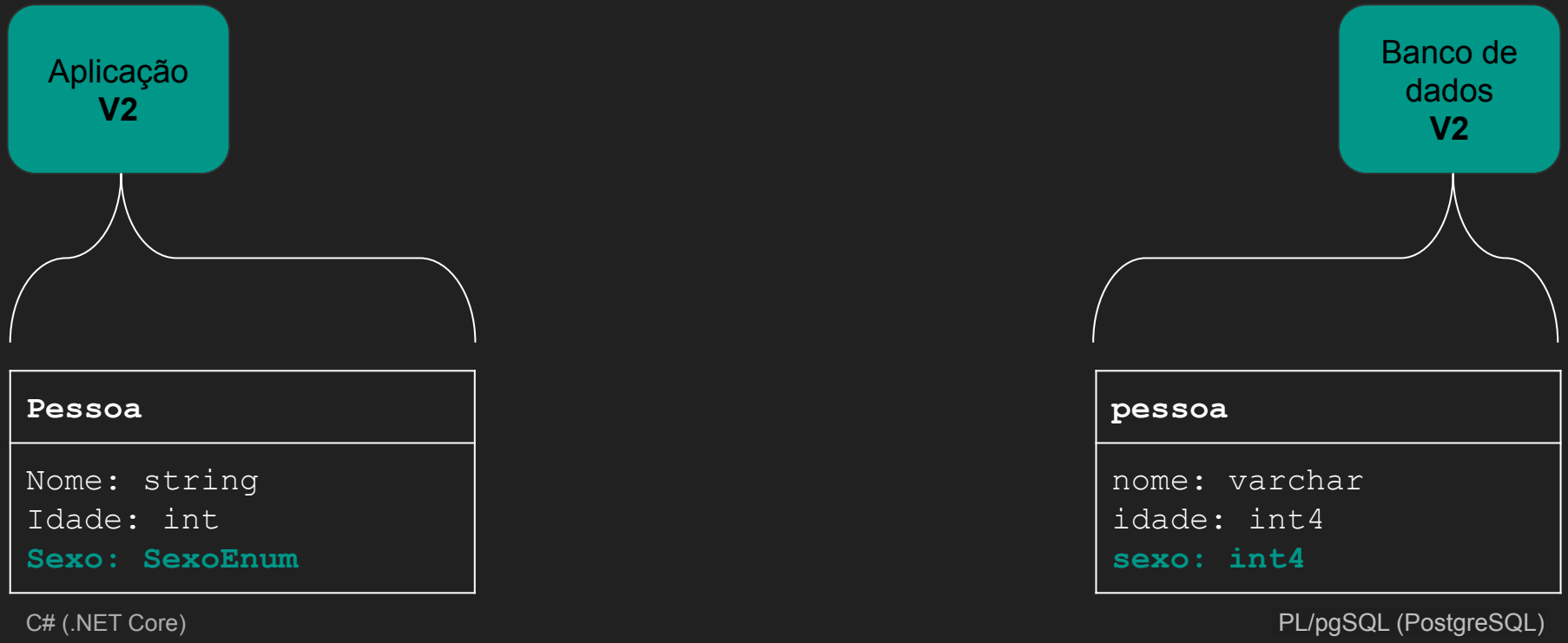
Backend: exemplo **SEM retrocompatibilidade**



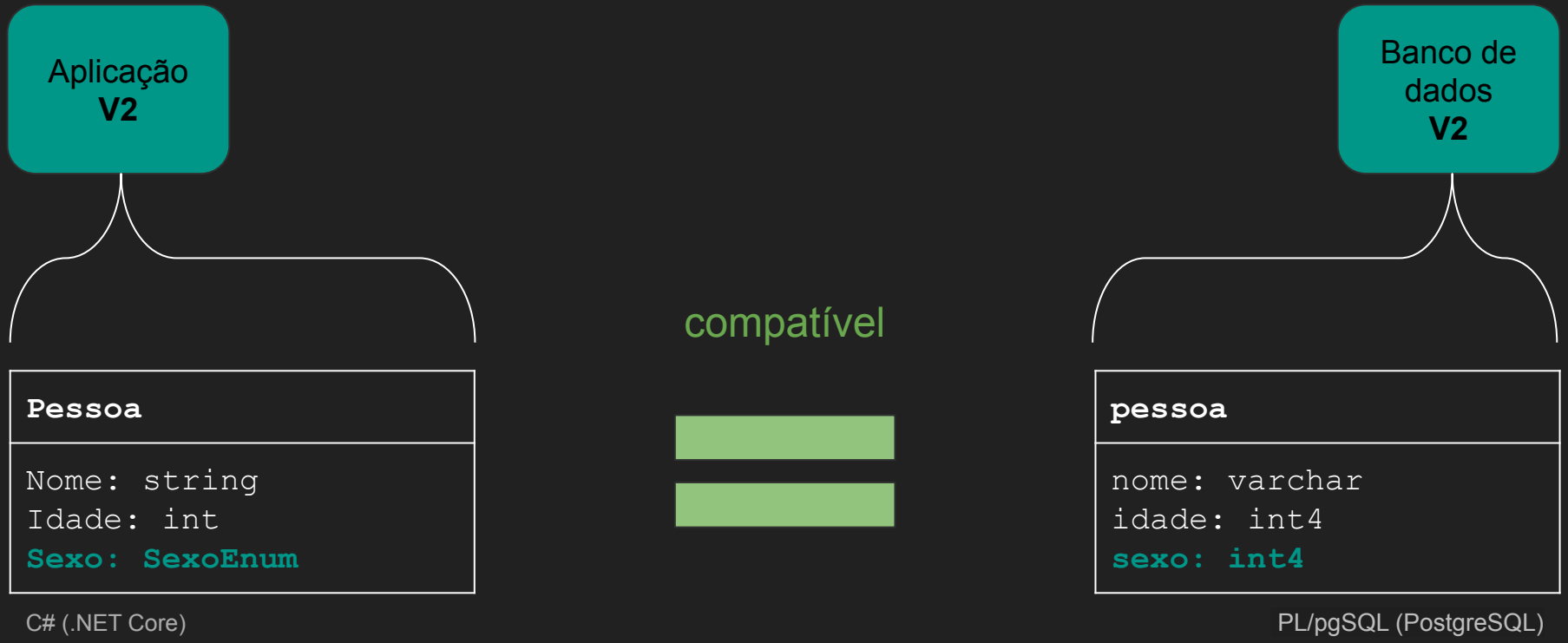
Backend: exemplo **SEM retrocompatibilidade**



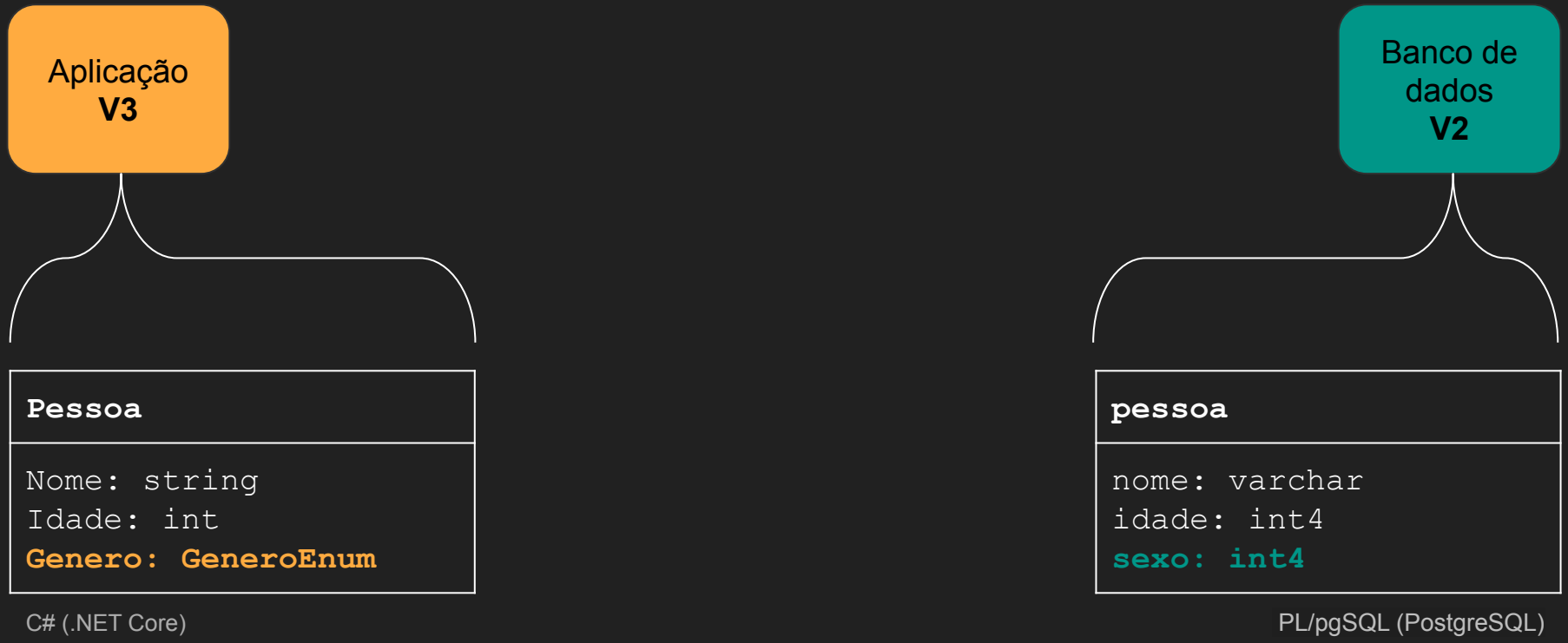
Backend: exemplo **SEM retrocompatibilidade**



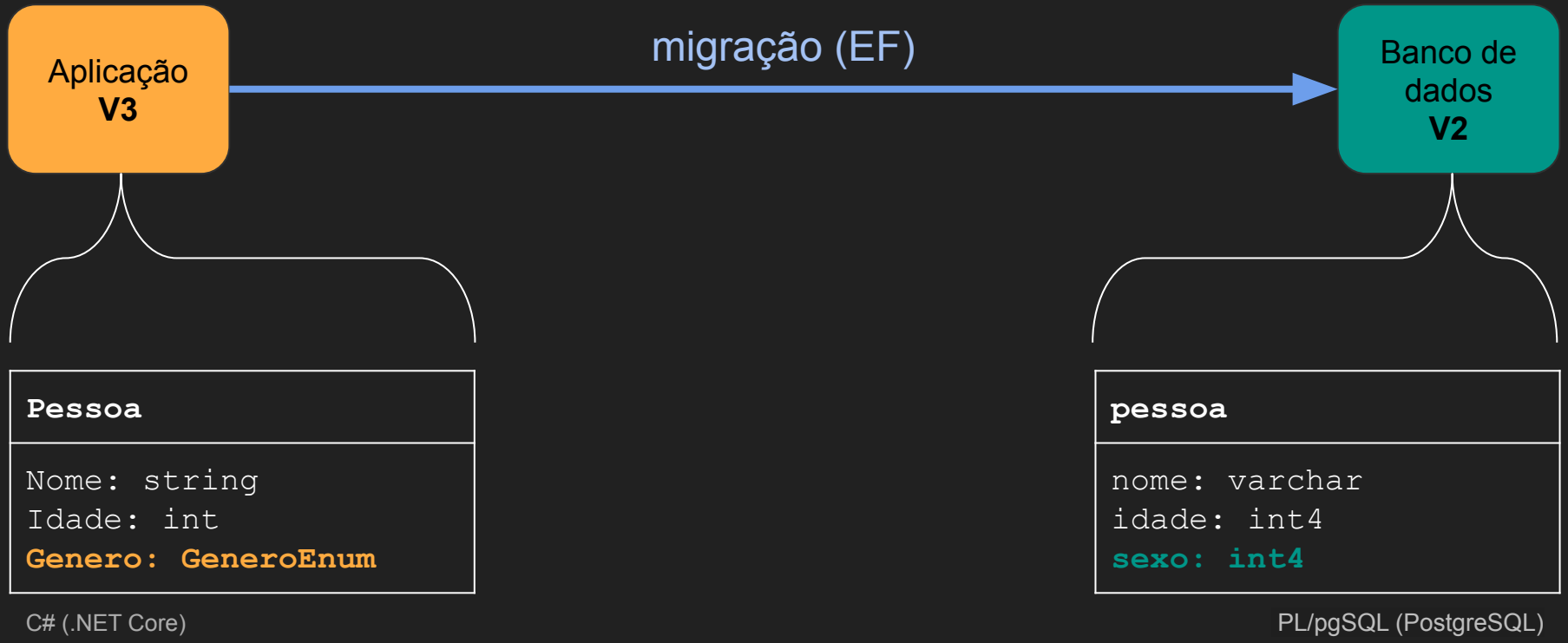
Backend: exemplo **SEM retrocompatibilidade**



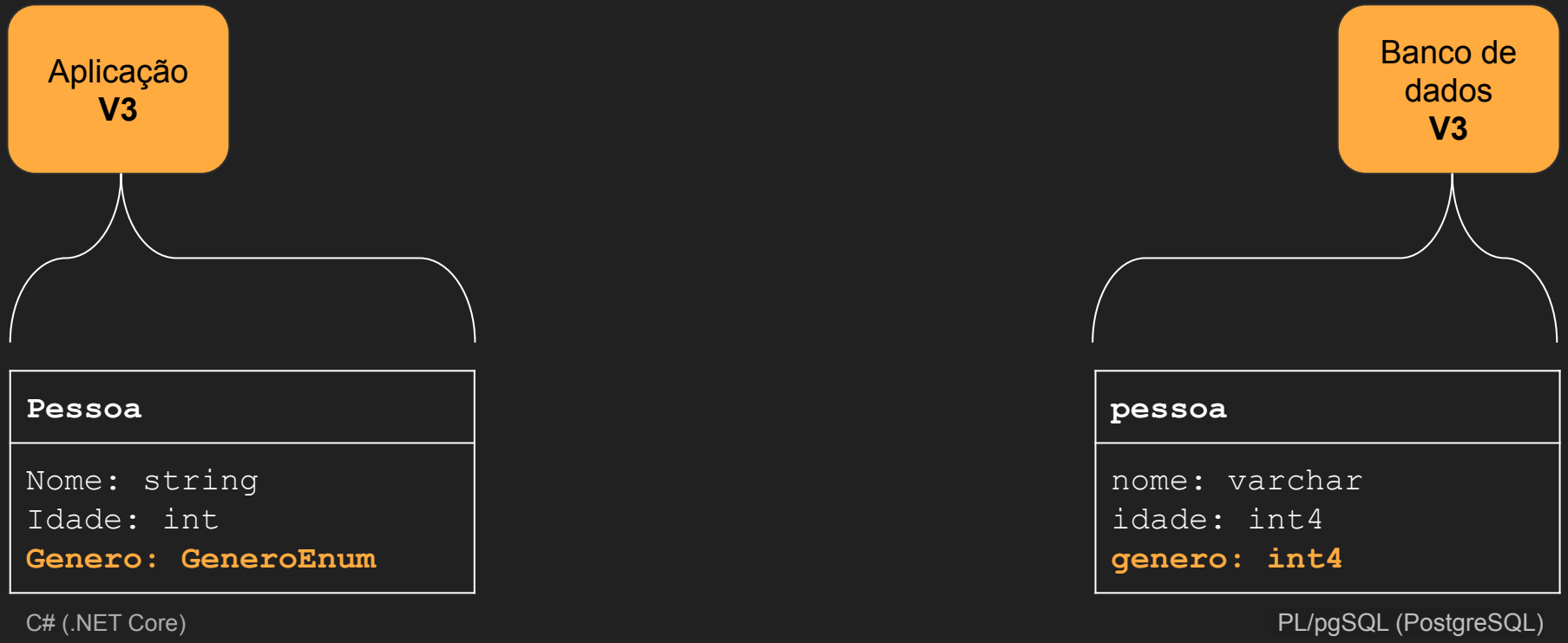
Backend: exemplo **SEM retrocompatibilidade**



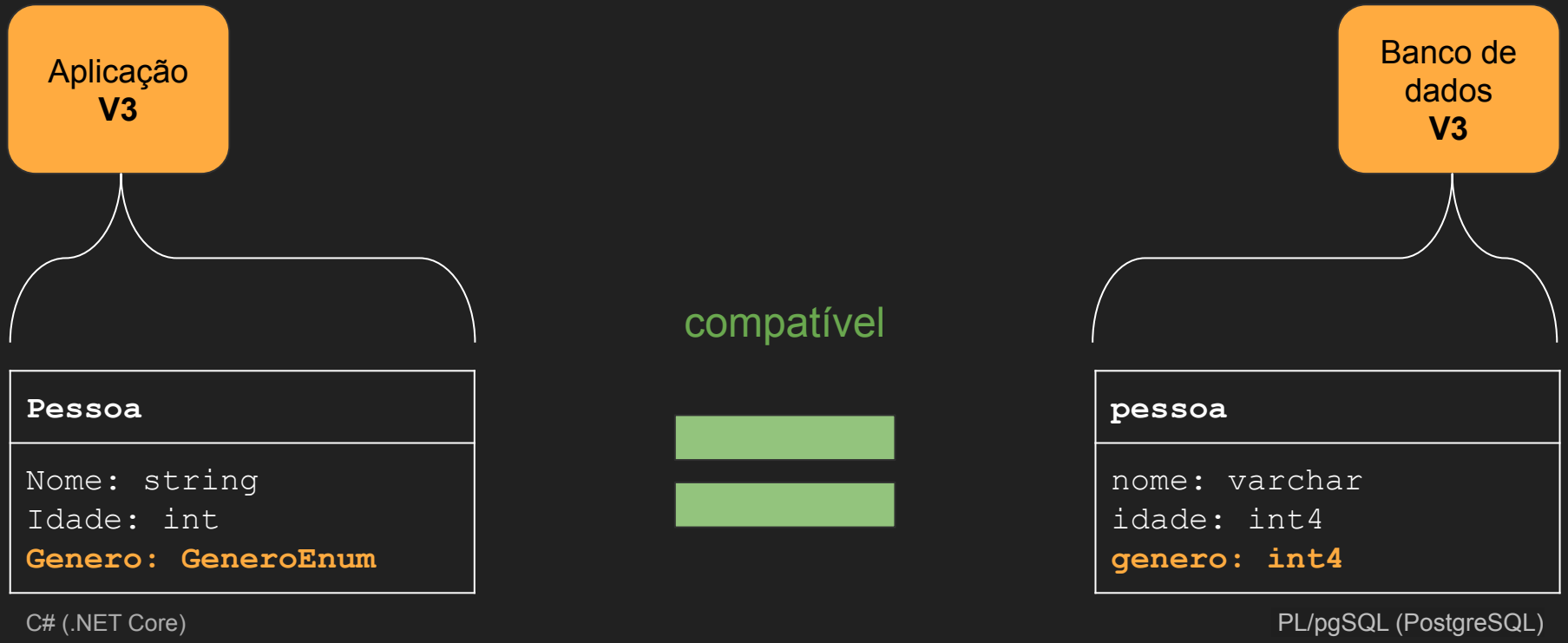
Backend: exemplo **SEM retrocompatibilidade**



Backend: exemplo **SEM retrocompatibilidade**



Backend: exemplo **SEM retrocompatibilidade**



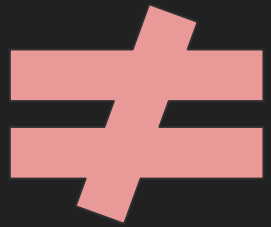
Backend: exemplo **SEM retrocompatibilidade**

Backend: exemplo **SEM retrocompatibilidade**

Aplicação
V2

Banco de dados
V3

incompatível



Pessoa

```
Nome: string
Idade: int
Sexo: SexoEnum
```

C# (.NET Core)

pessoa

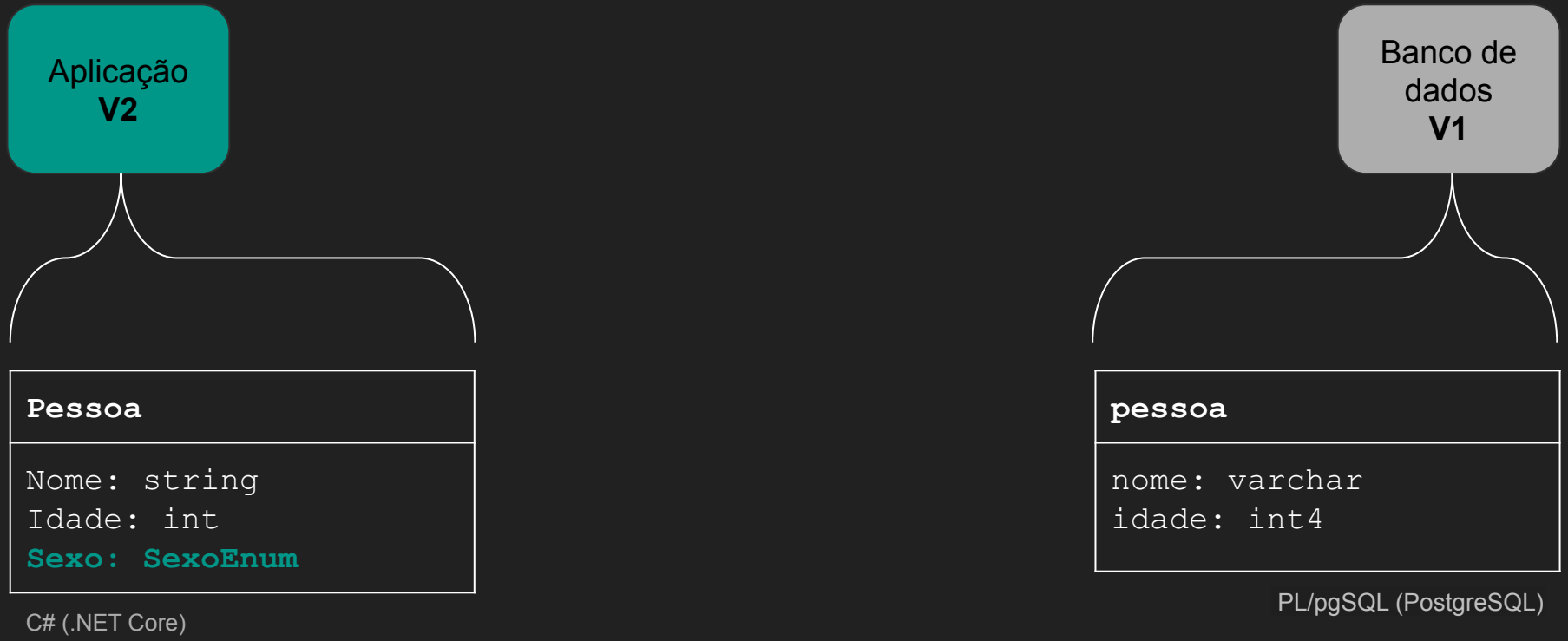
```
nome: varchar
idade: int4
genero: int4
```

PL/pgSQL (PostgreSQL)

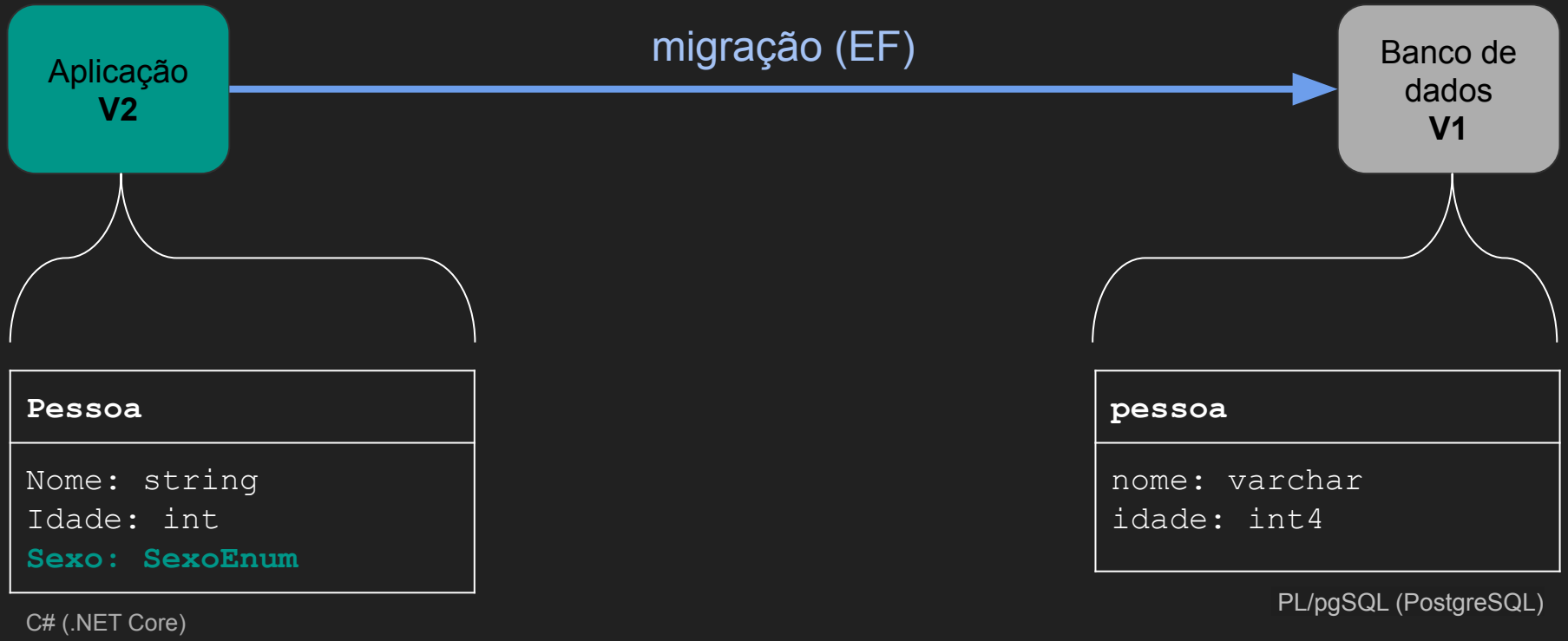
Backend: exemplo **COM retrocompatibilidade**

Backend: exemplo **COM retrocompatibilidade**

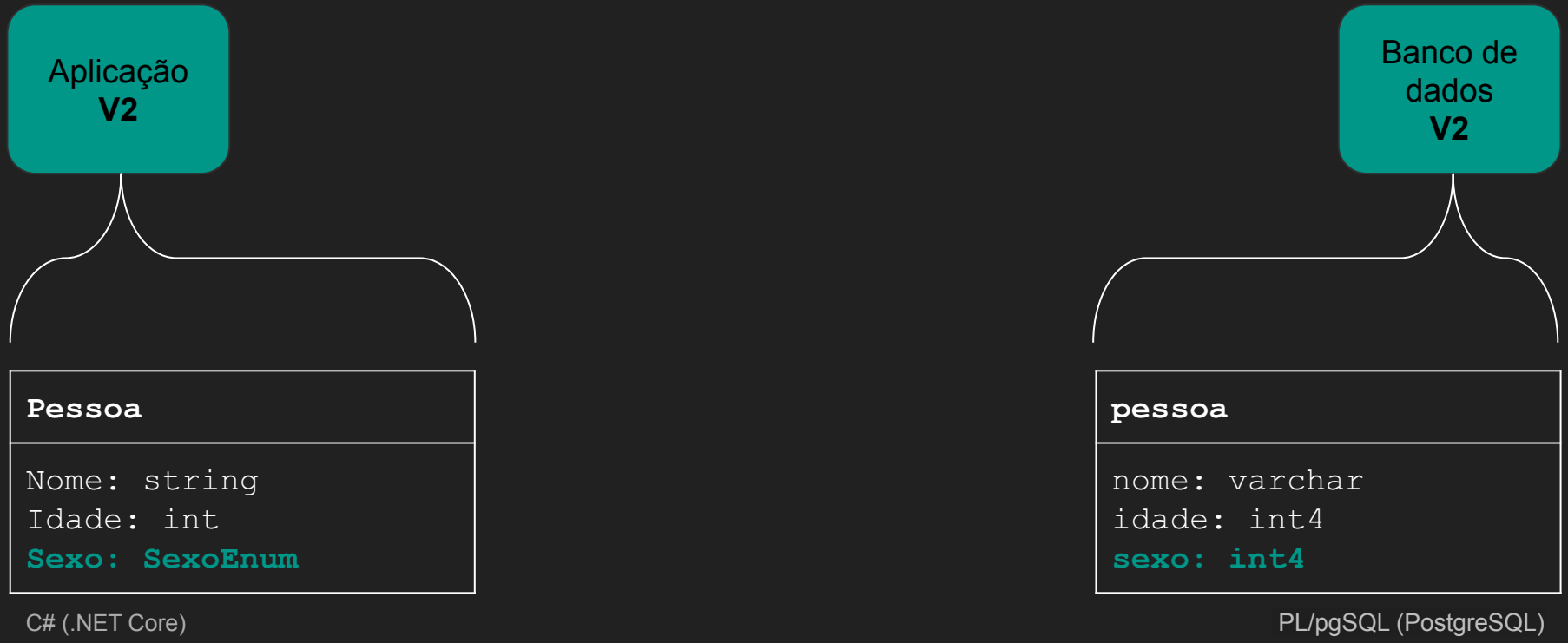
Backend: exemplo **COM retrocompatibilidade**



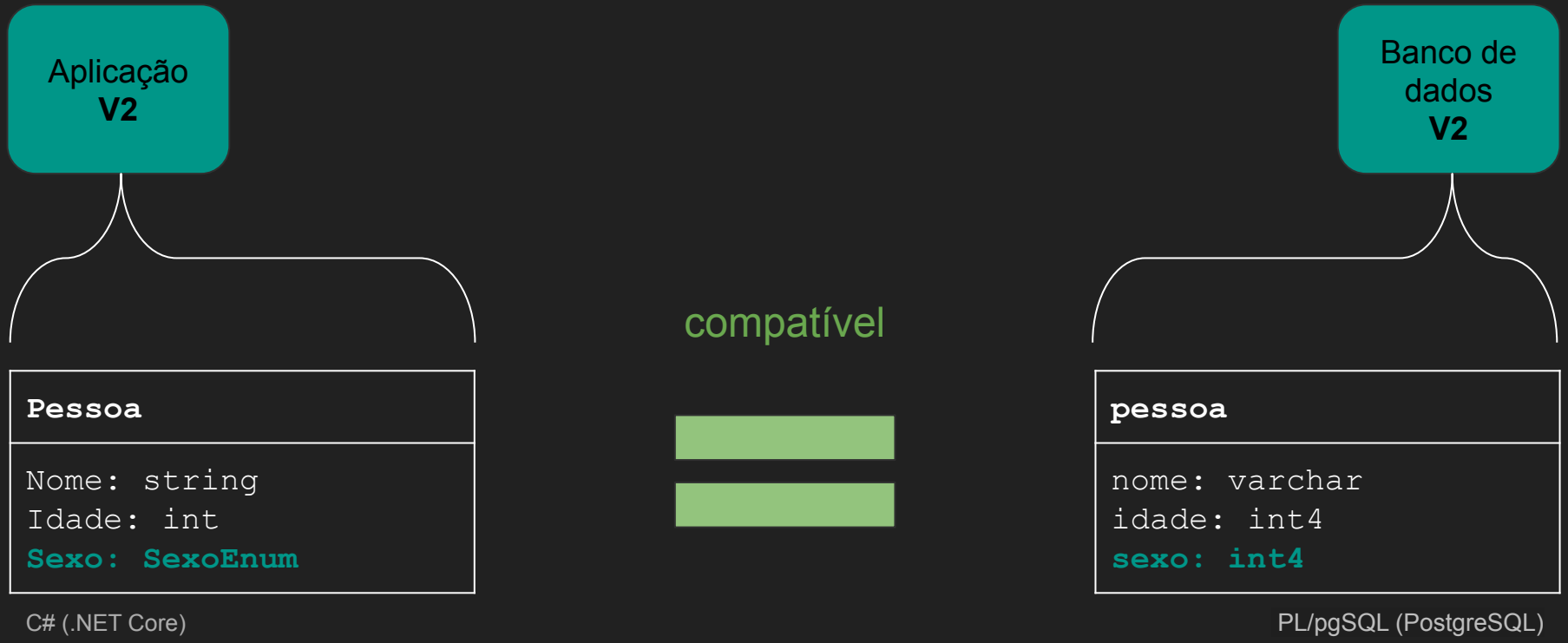
Backend: exemplo **COM retrocompatibilidade**



Backend: exemplo **COM retrocompatibilidade**

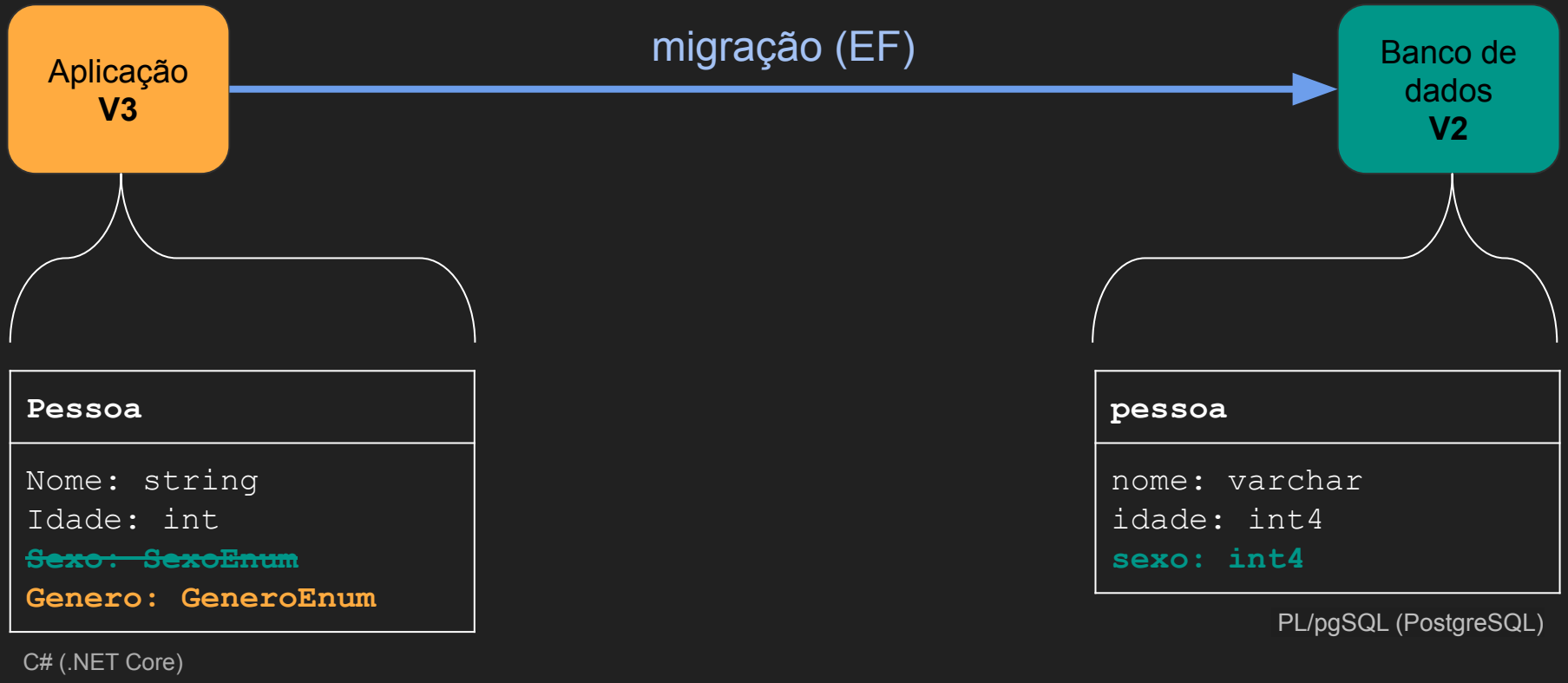


Backend: exemplo **COM retrocompatibilidade**



Backend: exemplo **COM retrocompatibilidade**

Backend: exemplo **COM retrocompatibilidade**



Backend: exemplo **COM retrocompatibilidade**

Backend: exemplo **COM retrocompatibilidade**

Aplicação
V3

Banco de dados
V3

compatível

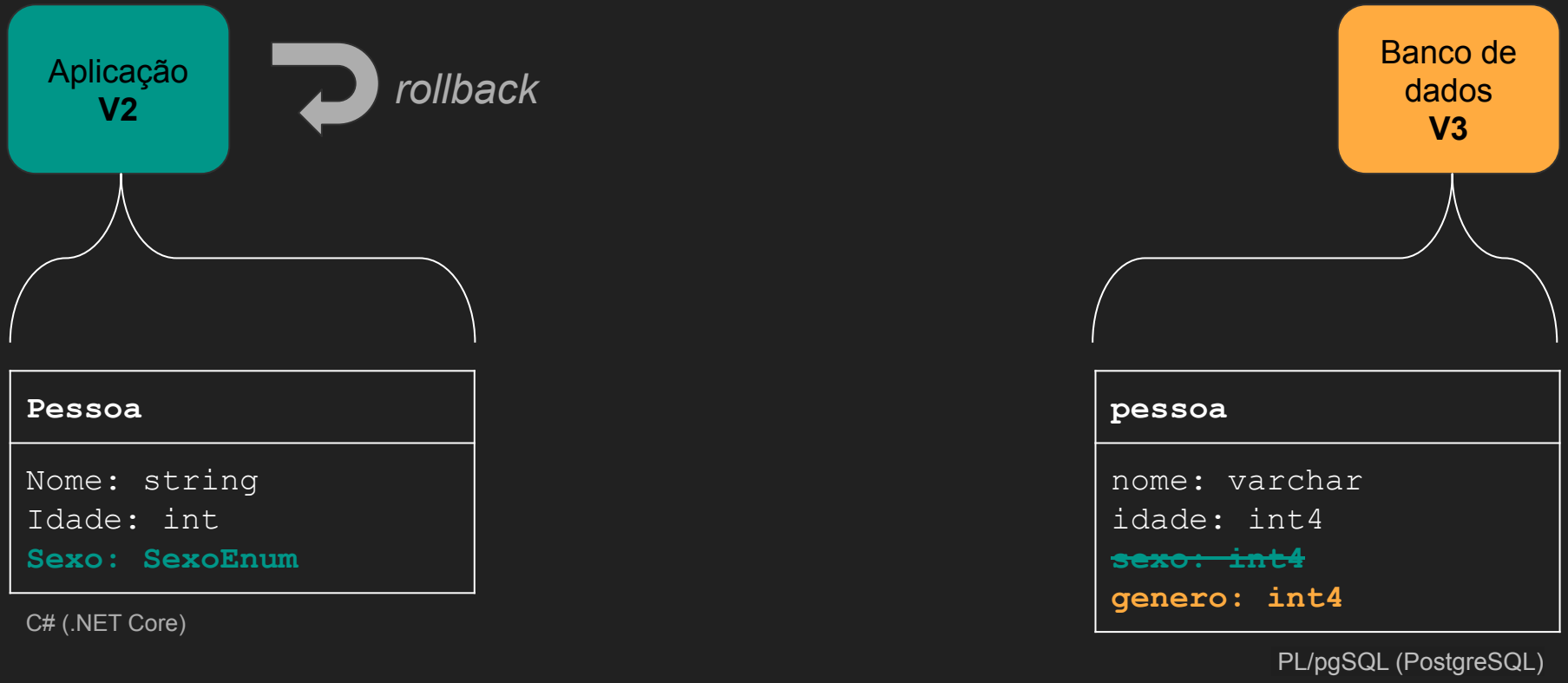
```
Pessoa
Nome: string
Idade: int
~~Sexo: SexoEnum~~
Genero: GeneroEnum
```

```
pessoa
nome: varchar
idade: int4
~~sexo: int4~~
genero: int4
```

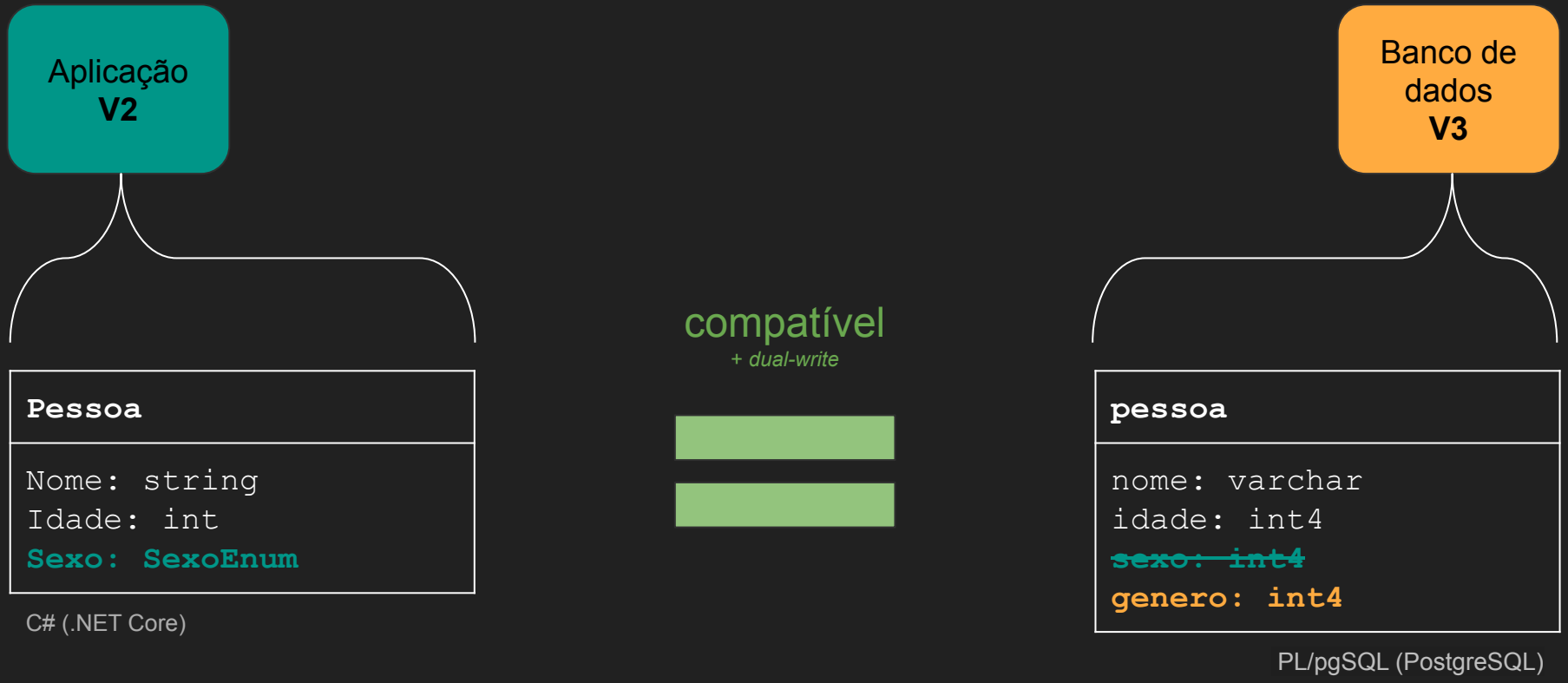
C# (.NET Core)

PL/pgSQL (PostgreSQL)

Backend: exemplo **COM retrocompatibilidade**



Backend: exemplo **COM retrocompatibilidade**



Matriz de compatibilidade (operações x fases)

Operação	***Expand***	***Backfill***	***Compat / dual-write***	***Cutover***	***Contract / kill legacy***
Renomear / substituir coluna	✅	✅	✅	✅	✅
Alterar tipo	✅	✅	✅	✅	✅
…					

Fonte: https://ricioli.bitbucket.io/docs/retrocompatilidade-db

Definições:

- ***Expand***: adiciona estruturas novas sem remover/alterar destrutivamente o legado;
- ***Backfill***: preenche novas propriedades/estruturas a partir de dados antigos;
- ***Compat*** *(dual-write)*: versões antigas continuam funcionando (*triggers*, *views*, camada de app);
- ***Cutover***: aplicação/algoritmos utilizam a propriedade/estrutura nova e para de ler as antigas;
- ***Contract***: remove legado após a janela de tempo com métricas confirmando desuso.

Frontend: conceitos de retrocompatibilidade

- **Manter suporte a formatos antigos de API** (ex: *backend* muda o formato do JSON de retorno, mas o *frontend* ainda entende o antigo);
- **Controle de versão em contratos de API** (ex: `/v1/usuarios`, `/v2/usuarios` no consumo);
- *Flags*/**condicionais para ativar novas funcionalidades** sem quebrar o fluxo anterior;
- ***Fallbacks* visuais** (ex: comportamento padrão quando o novo dado ainda não existe);
- **Persistência local** (*localStorage*, *caches*, etc.) **compatível com versões antigas** de dados;
- **Componente tolerante a mudanças**: não depender rigidamente de propriedades novas ou obrigatórias;
- *etc*.

Quality assurance/tests: conceitos de retrocompatibilidade

- **Testes de regressão** funcionais: validar que as funções antigas continuam operando normalmente;
- **Cenários com dados antigos** / legados: reutilizar *payloads* e registros de versões anteriores;
- **Testes manuais exploratórios** focados em compatibilidade: verificar impactos sutis em fluxos antigos;
- **Verificação de compatibilidade *frontend-backend* (Teste *Matrix*)**: testar o novo *frontend* com API antiga e vice-versa;
- **Validação de exibição e comportamento de elementos obsoletos**: garantir que campos removidos ou alterados não causem falhas;
- *etc*.

Resumo

- Evite a complexidade: **faça o possível para NÃO PRECISAR da retrocompatibilidade e/ou migrações**:
  - exemplo válido: mudar a tipagem da propriedade `Valor` de `int` para `decimal`;
  - exemplo inválido: mudar o nome da propriedade `UserId` para `UsuarioId`.

- Caso sua implementação/correção haja alterações no banco de dados se questione:
  - ***funciona com versões (da aplicação) antigas/anteriores?***
  - *funciona durante o deploy?*
  - *funciona após um rollback?*