

# A Evolução do ABAP: Guia Essencial para o Desenvolvedor Moderno

Dos Reports Clássicos ao ABAP  
RESTful Model na Nuvem

```
REPORT z_classic_report  
REPORT z_classic_report  
WRITE: / 'Hello, World!'
```

```
WRITE: / 'Hello, World!'
```

ABAP RESTful Application Programming Model



API Gateway

Application Programming services



Cloud Services

Web APIs

Microservices

CDS Views

# A Nossa Jornada: Do “Porquê” ao “Como”



## A Nova Filosofia

Entendendo o problema do código legado e a solução estratégica do Clean Core.

## Ferramentas e Sintaxe

Dominando o novo ambiente de desenvolvimento (ADT) e a sintaxe ABAP que nos torna mais produtivos.

## Estruturas de Dados

Organizando e manipulando informações de forma eficiente em memória.

## Interação com o Banco

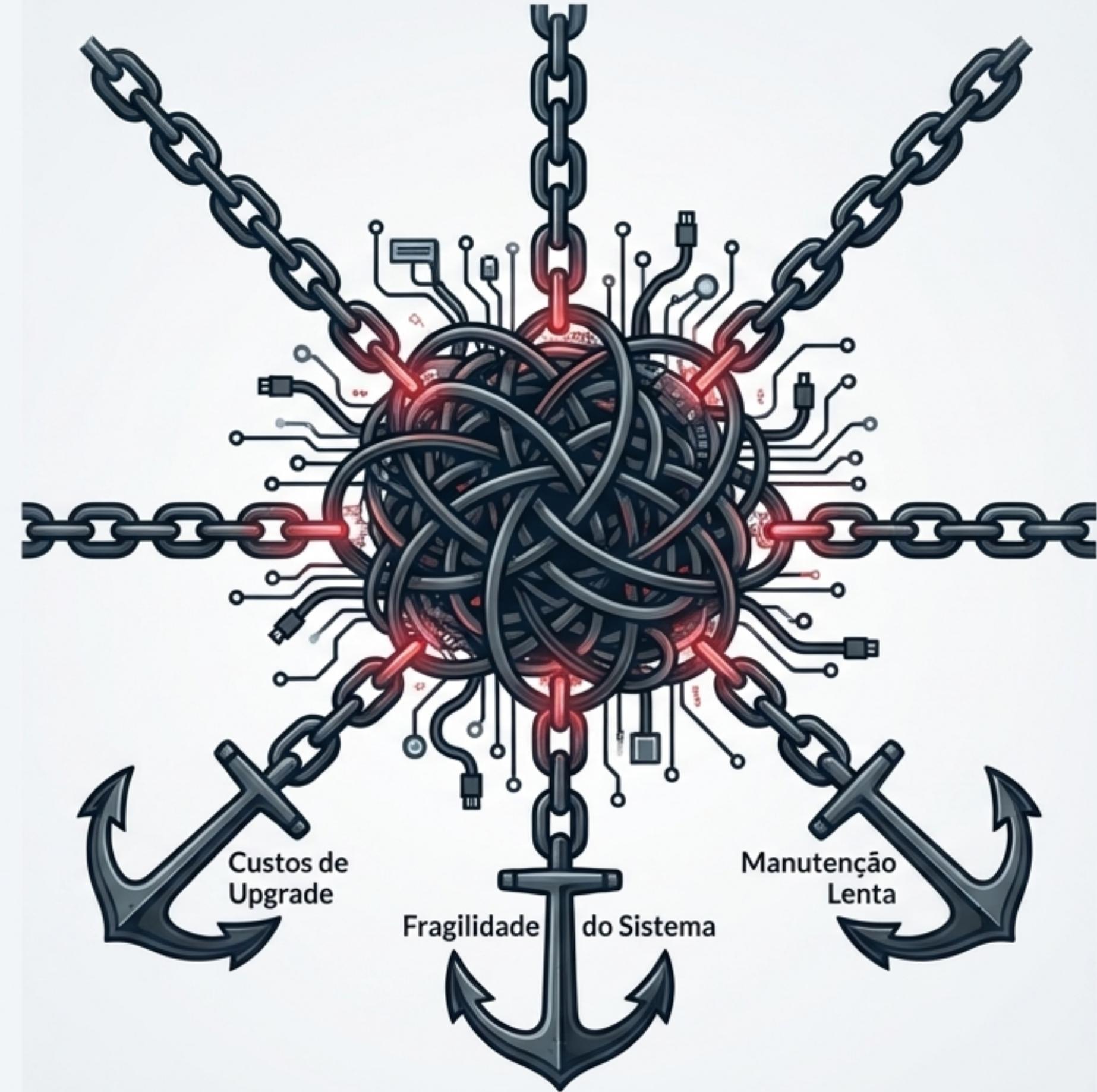
Lendo e, crucialmente, escrevendo dados da maneira correta e segura no modelo moderno.

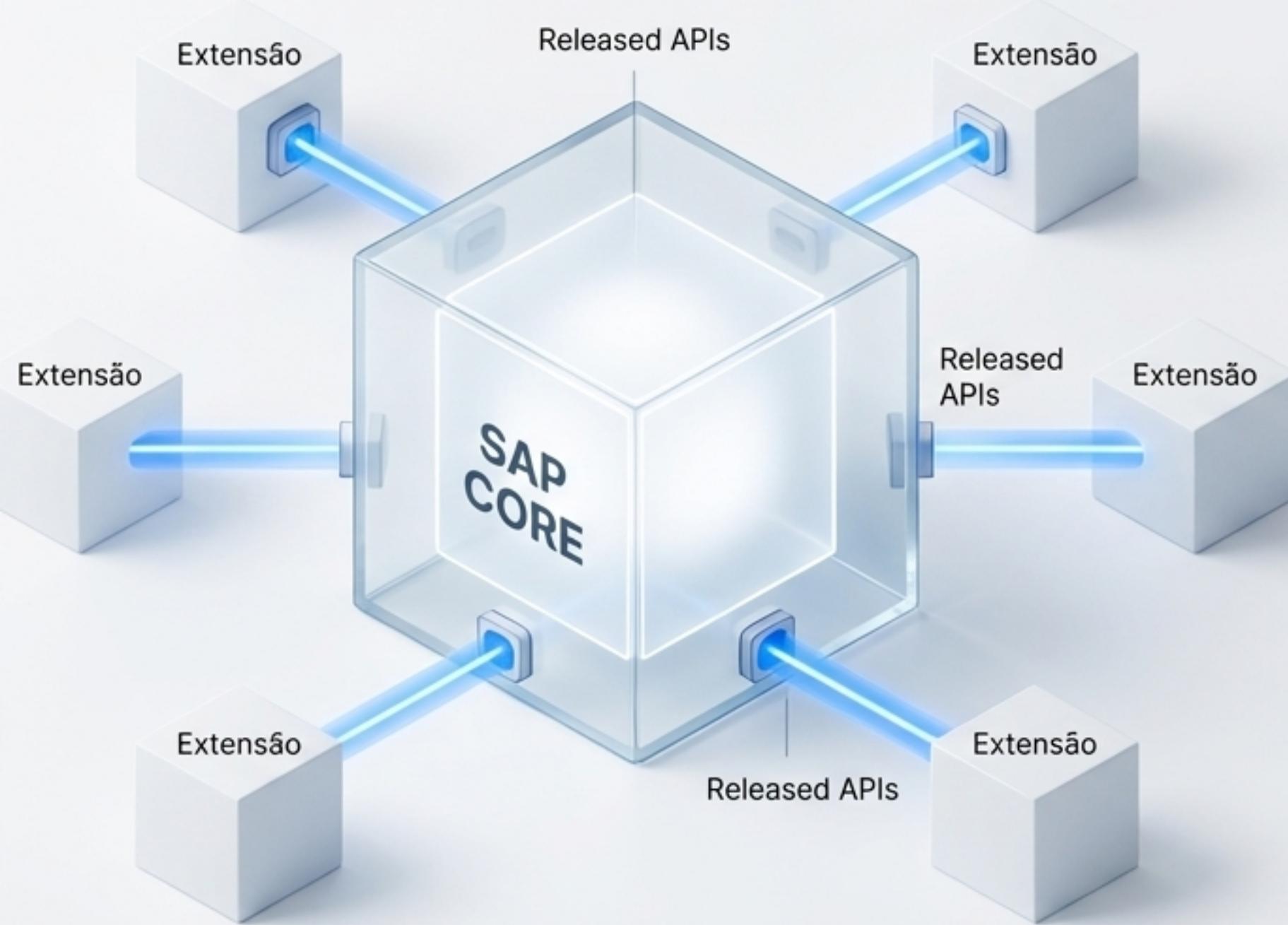
## O Modelo Completo

Juntando todas as peças na arquitetura ABAP RESTful (RAP).

# O Ponto de Partida: A Dívida Técnica do Legado

- **'Inferno do Upgrade'**: Modificações de núcleo e acesso direto a tabelas internas que quebravam o sistema em cada atualização da SAP (SPDD/SPAU).
- **Código Espaguete**: Acoplamento forte, onde customizações dependiam de estruturas internas da SAP, tornando o sistema frágil e caro de manter.
- **'Superpoderes' Perigosos**: Desenvolvedores podiam ler qualquer tabela e acessar o sistema operacional, criando riscos de segurança e instabilidade.





# A Solução Estratégica: ABAP Cloud e a Filosofia Clean Core

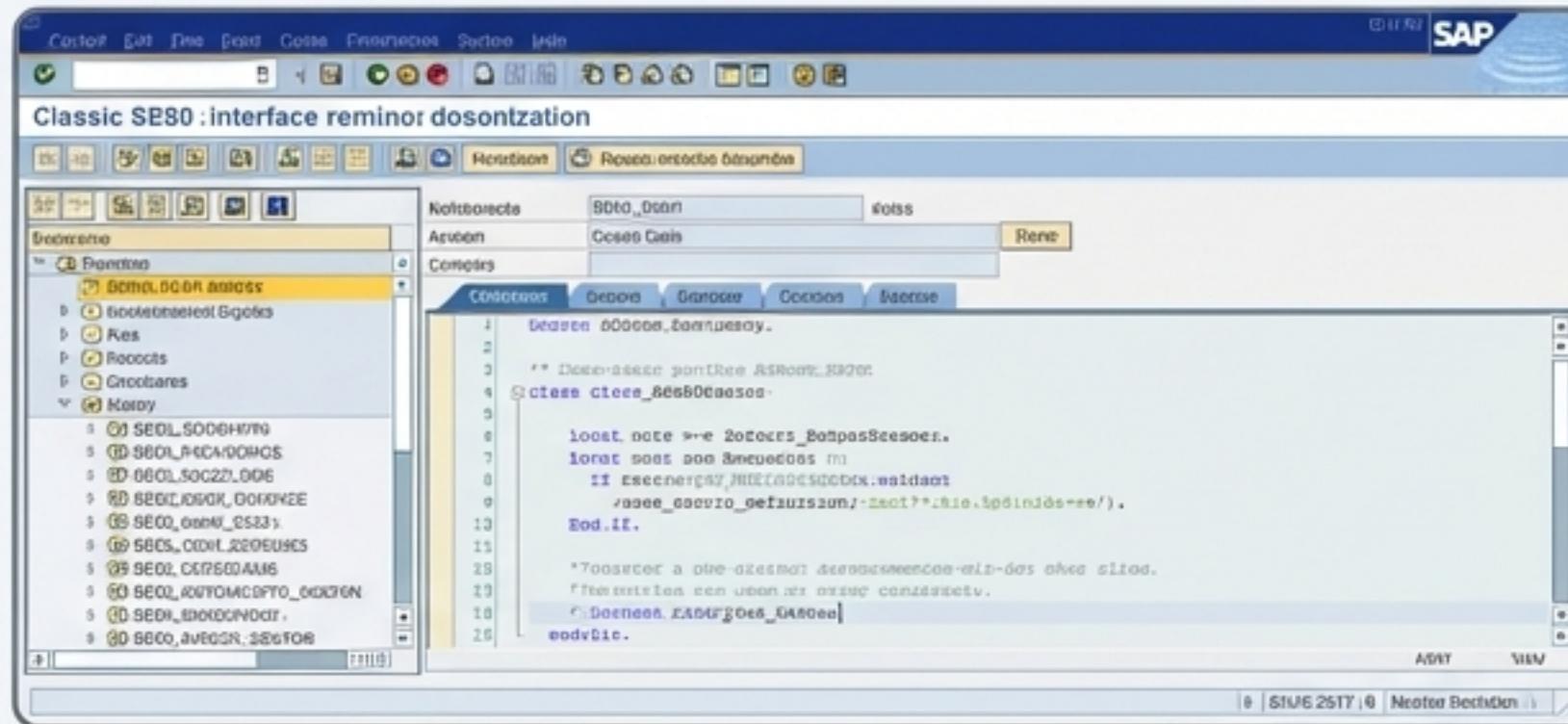
## A Regra de Ouro: Estenda, Não Modifique.

O princípio do Clean Core é manter o núcleo do ERP intacto para permitir atualizações automáticas e seguras.

## Restrições Técnicas do ABAP Cloud:

- **Language Version 5:** O compilador bloqueia comandos obsoletos como `CALL SCREEN` e `WRITE`.
- **Released Objects (APIs Públicas):** Só é permitido referenciar objetos SAP explicitamente liberados (Whitelisted), como a CDS View `I_Product` em vez da tabela `MARA`. Acesso direto a tabelas internas é proibido.

# O Cockpit Moderno: Por Que a SE80 se Tornou Obsoleta



SE80 (SAP GUI)

A screenshot of the ADT (Advanced Development Tool) in Eclipse. The title bar says "adt - SAP - Eclipse Platform - Eclipse IDE - Open". The main area shows a code editor with ABAP code for renaming objects, with syntax highlighting and code completion. The code includes annotations like "@sapui5", "class class ASAP\_Connect", and "class name = Connecter;". The interface is modern and integrated with other Eclipse tools.

ADT (Eclipse)

Vantagem	ADT no Eclipse	SE80 no SAP GUI
Refatoração	Ferramentas avançadas (renomear, extrair) com um clique.	Manual e propenso a erros.
Objetos Cloud	Suporte <b>exclusivo</b> para CDS Views, Behavior Definitions (RAP) e Service Bindings.	<b>Impossível</b> editar ou criar objetos RAP.
Conexões	Múltiplos sistemas (DEV, QAS) visíveis lado a lado na mesma interface.	Uma janela por mandante.
Integração	Nativa com <b>abapGit</b> para versionamento e pipelines de CI/CD.	Limitada ou inexistente.

# Escreva Menos, Faça Mais: A Sintaxe Expressiva do ABAP Moderno

## Antes (ABAP Clássico) (Lato)

```
DATA: lv_text_a TYPE string,  
      lv_text_b TYPE string,  
      lv_result TYPE string,  
      lv_fase   TYPE string.  
  
lv_text_a = 'Olá'.  
lv_text_b = 'Mundo'.  
CONCATENATE lv_text_a lv_text_b INTO  
lv_result SEPARATED BY space.  
  
IF lv_idade < 18.  
  lv_fase = 'Menor'.  
ELSE.  
  lv_fase = 'Adulto'.  
ENDIF.
```

## Depois (ABAP Moderno) (Lato)

```
" Declaração Inline (DATA(...))  
DATA(lv_text_a) = 'Olá'.  
DATA(lv_text_b) = 'Mundo'.  
  
" String Templates (|...|)  
DATA(lv_result) = |{ lv_text_a } {  
DATA(lv_result) = |{ lv_text_a } {  
lv_text_b }|.  
  
" Operador Construtor (COND)  
DATA(lv_fase) = COND #( WHEN lv_idade < 18  
                           THEN 'Menor'  
                           ELSE 'Adulto' ).
```

# Classes: O Contêiner da Lógica de Negócio

## Anatomia Essencial

Toda classe ABAP é dividida em `DEFINITION` (o 'contrato', descreve o que a classe faz) e `IMPLEMENTATION` (a 'lógica', o código real).

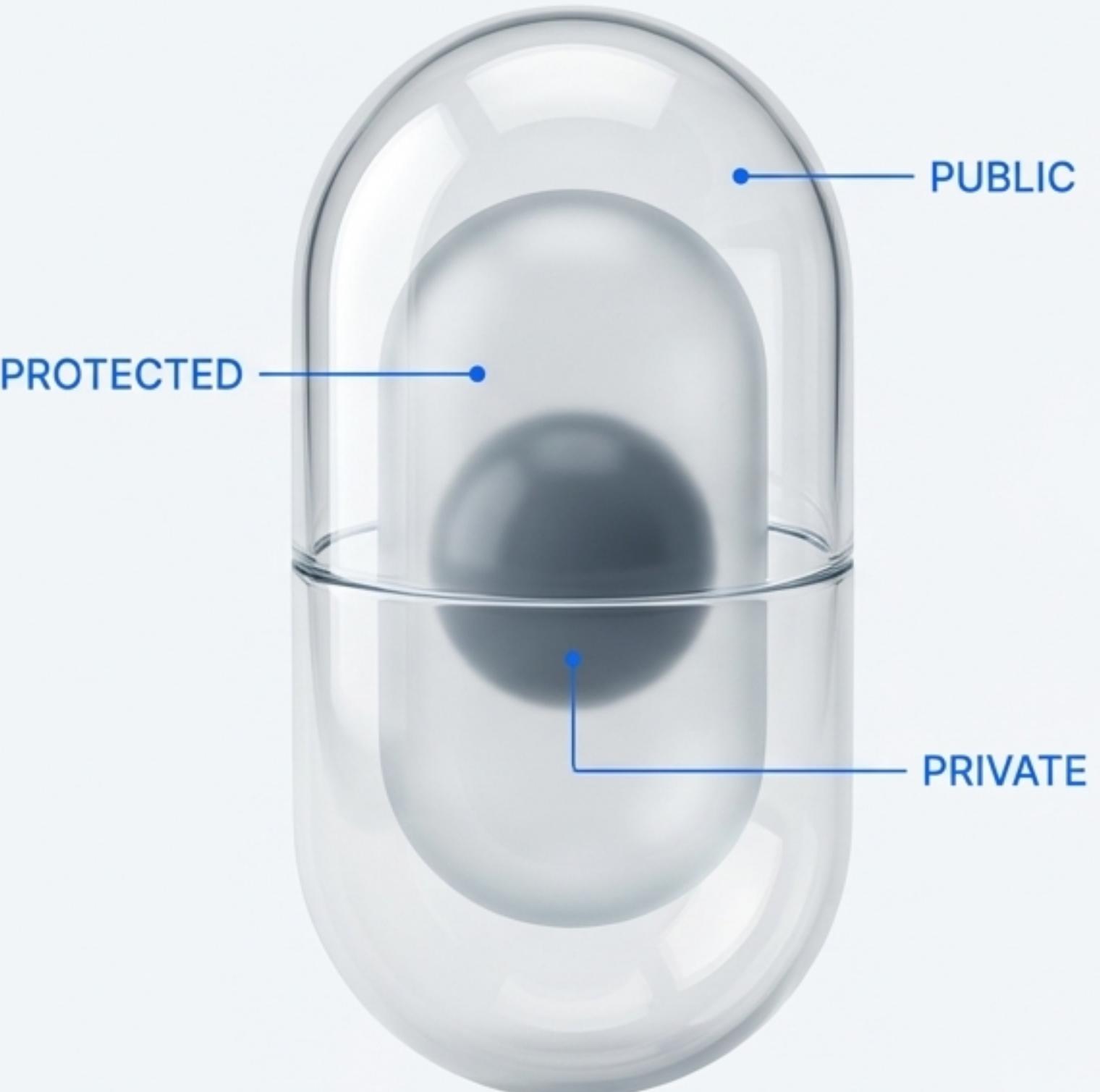
## Encapsulamento (O Cofre)

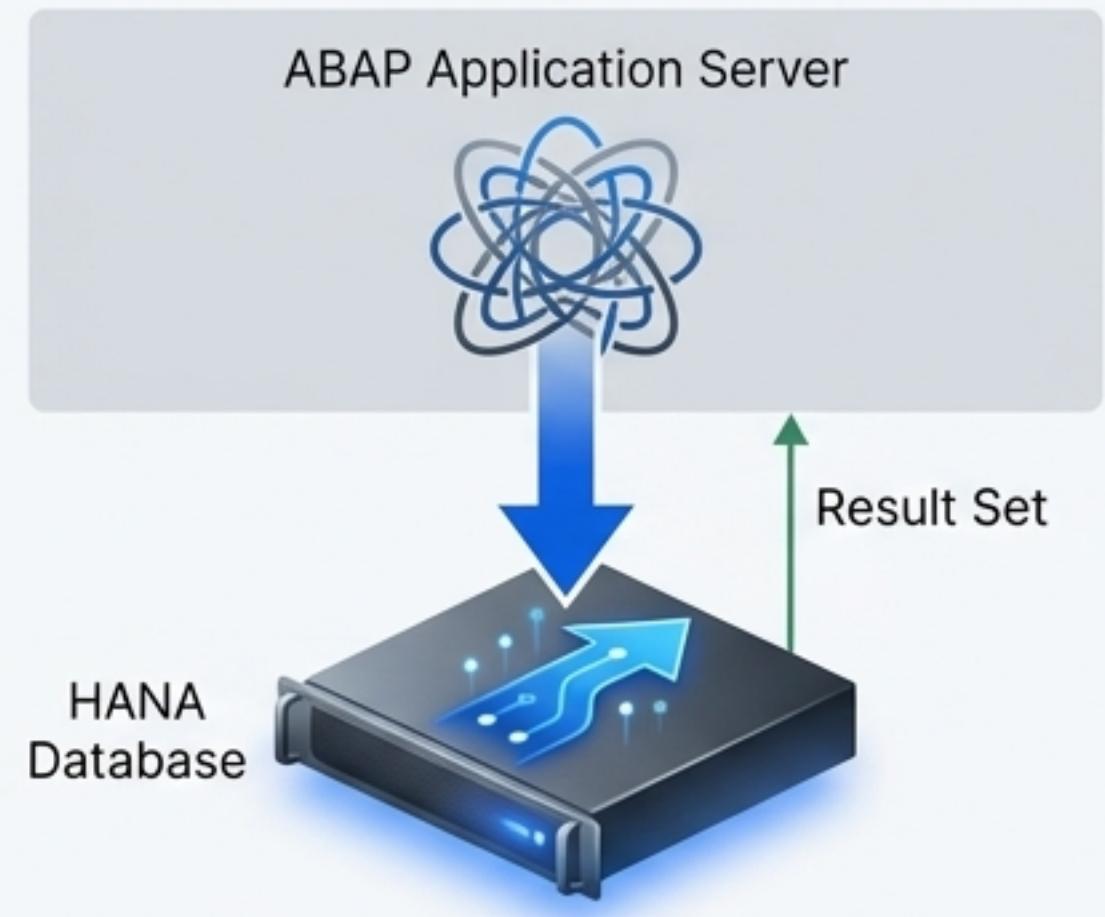
As seções de visibilidade protegem a integridade do código:

- **PUBLIC**: A 'vitrine', acessível por qualquer um. Define a API da classe.
- **PROTECTED**: Acessível pela classe e suas filhas (herança).
- **PRIVATE**: Acessível **apenas** pela própria classe. Oculta a complexidade interna.

## Instanciação Moderna

```
DATA(lo_objeto) = NEW zcl_minha_classe( ).
```





## 'Code Pushdown': Deixando o Banco de Dados HANA Trabalhar

O ABAP SQL moderno move o processamento para o banco, trazendo apenas os resultados. Para isso, a sintaxe mudou para o 'Strict Mode'.

### As 3 Regras de Ouro do Strict Mode:

1. **Virgulas Obrigatórias:** Campos na lista FIELDS devem ser separados por vírgula.
2. **INTO no Final:** A cláusula de destino (INTO TABLE ...) deve ser a última.
3. **@ Obrigatório:** Toda variável ABAP usada na query deve ser prefixada com @.

```
DATA(lv_airline) = 'LH'.

SELECT FROM /dmo/connection
  FIELDS carrier_id, connection_id
  WHERE carrier_id = @lv_airline    " <— Regra 3
  ORDER BY connection_id
  INTO TABLE @DATA(lt_flights).    " <— Regra 2
```

# Da Linha Única à Transferência de Dados: O Papel das Estruturas

Uma estrutura agrupa campos relacionados. No RAP, é essencial mover dados entre estruturas de camadas diferentes (ex: do Banco para a UI).

## O Problema:

Copiar campo a campo é tedioso e propenso a erros.

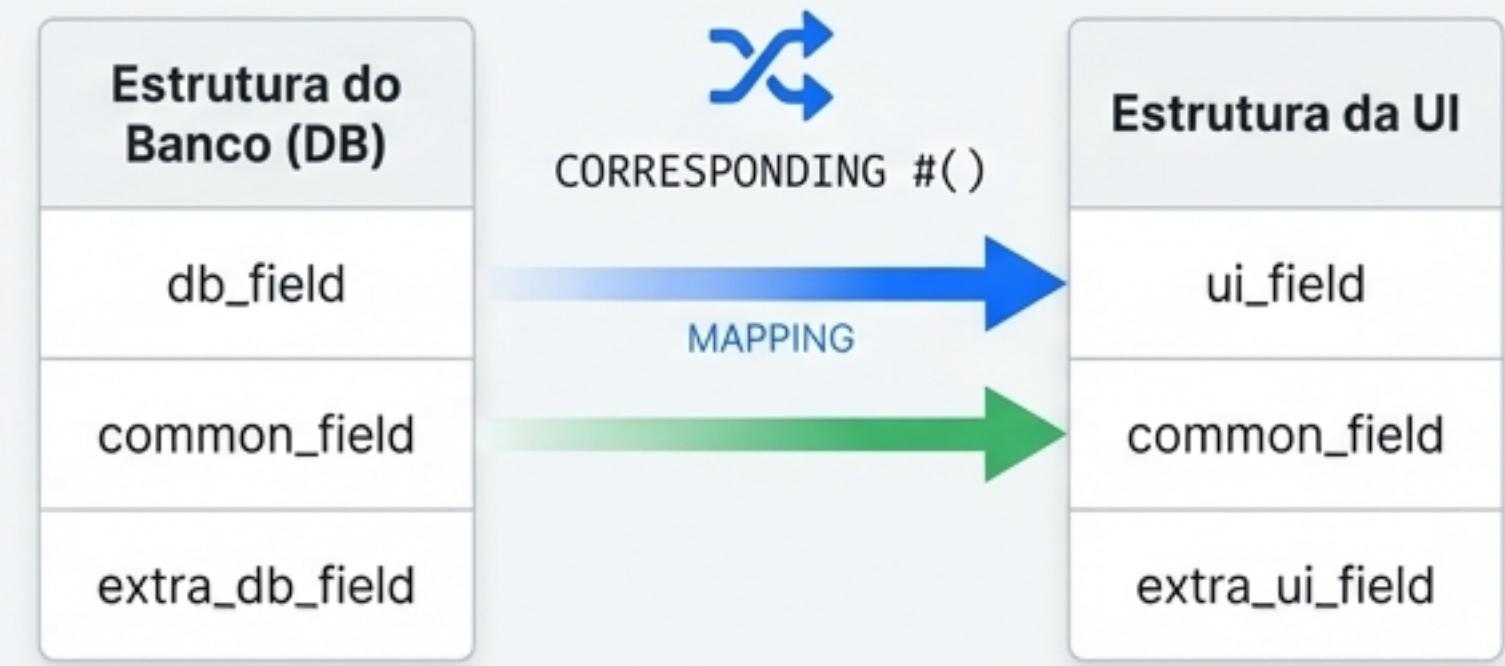
## A Solução: `CORRESPONDING`

Este operador moderno automatiza a cópia de campos com nomes iguais e oferece controle total.

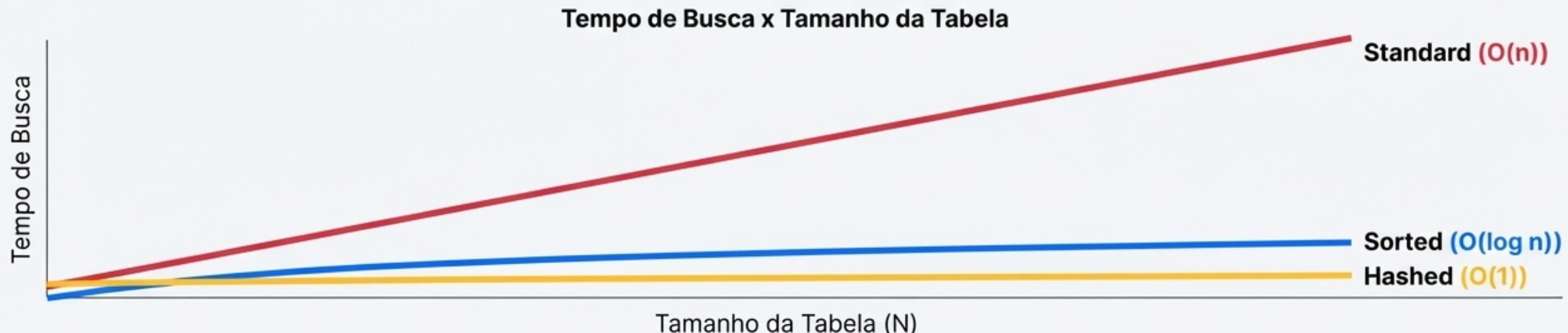
## Recursos Avançados Essenciais para RAP:

```
ls_b = CORRESPONDING #( ls_a MAPPING ui_field = db_field ).
```

```
ls_b = CORRESPONDING #( BASE (ls_b) ls_a ).
```



# O Custo de Uma Busca: Escolhendo a Tabela Interna Certa



Tipo de Tabela	Acesso por Chave	Complexidade	Caso de Uso Ideal
Standard	Busca Linear (percorre tudo)	$O(n)$	Listas pequenas, onde a ordem de inserção importa.
Sorted	Busca Binária (divide ao meio)	$O(\log n)$	Tabelas com muitas leituras e poucas escritas.
Hashed	Acesso Direto (cálculo de hash)	$O(1)$	Grandes tabelas de cache, acesso sempre por chave única.

**Otimização Adicional:** Chaves Secundárias (SECONDARY KEY) permitem otimizar buscas em outros campos sem alterar a chave primária da tabela.

# Por Que `INSERT`, `UPDATE` e `DELETE` Diretos São Proibidos

No paradigma RAP, a tabela do banco é privada. Acessá-la diretamente com SQL é um 'Bypass' que ignora ('atropela') toda a inteligência da aplicação.

## O que você ignora com um `UPDATE` direto:

- **Validações:** Regras de negócio como 'data de entrega não pode ser no passado'.
- **Determinações:** Cálculos automáticos de impostos ou totais.
- **Autorizações:** Verificações de permissão do usuário.
- **Logs de Auditoria:** Geração de documentos de modificação.

## A Solução:

Para modificar dados, você deve 'pedir' ao **Business Object (BO)**. Ele atua como um guardião que garante que todas as regras sejam cumpridas antes de persistir a alteração.



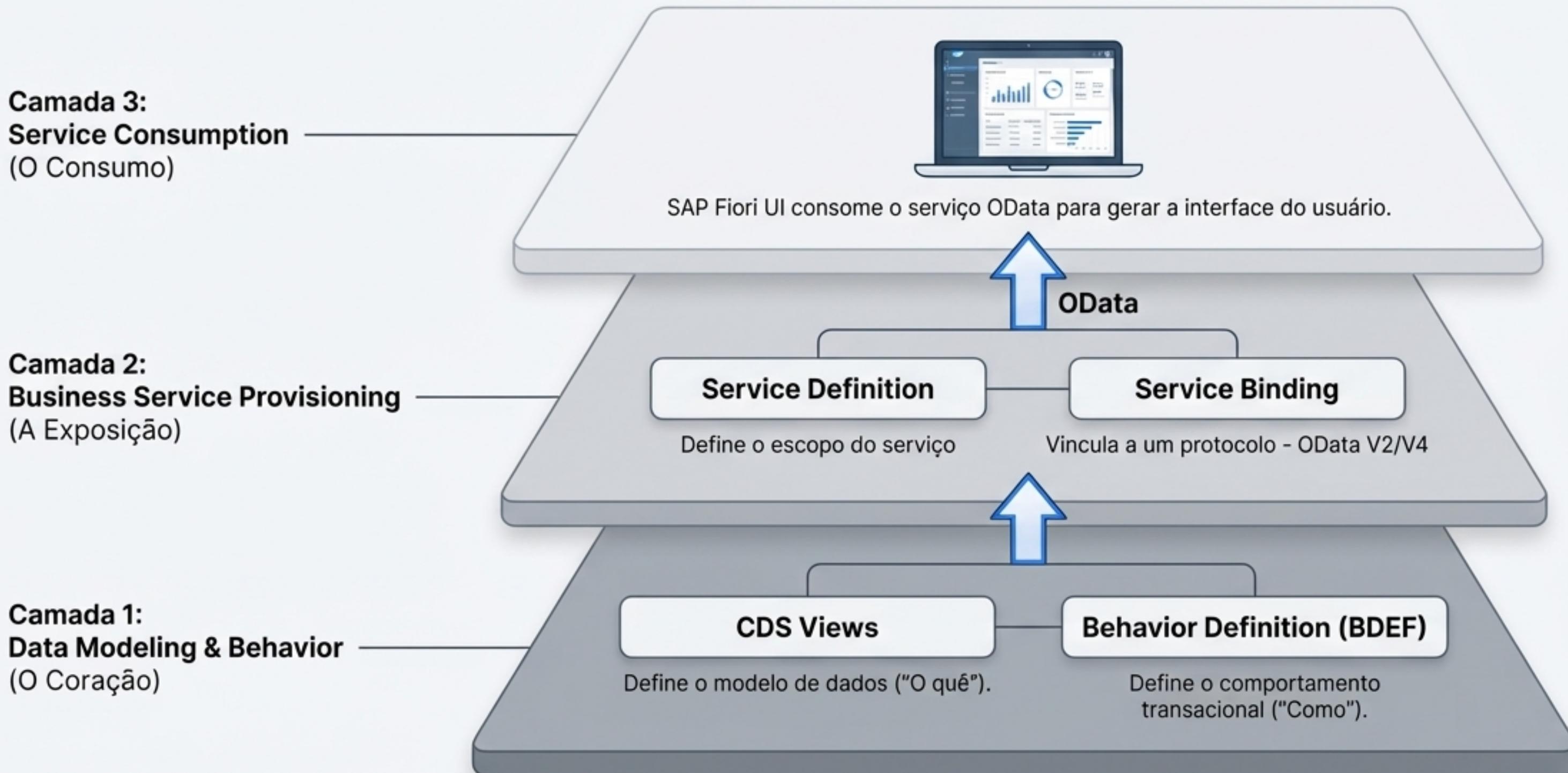
# A Linguagem da Transação: Falando com o Business Object via EML

EML (Entity Manipulation Language) é a extensão da sintaxe ABAP para interagir com Business Objects do RAP. O comando central é `MODIFY ENTITIES`, que orquestra todas as operações transacionais.



**Processo:** As modificações ocorrem em um *buffer transacional* em memória. A gravação no banco só acontece com o comando `COMMIT ENTITIES`.

# Juntando Todas as Peças: A Arquitetura do ABAP RESTful Model (RAP)



# Piloto Automático ou Câmbio Manual? A Decisão Entre Managed e Unmanaged

Ao criar um BDEF, você escolhe o cenário de implementação, que define o nível de responsabilidade do desenvolvedor.

Cenário	Managed (Gerenciado)	Unmanaged (Não Gerenciado)
Visual		
Como Funciona	O framework RAP assume o controle total das operações CRUD (INSERT, UPDATE, DELETE).	O desenvolvedor implementa manualmente a lógica de persistência (CREATE, UPDATE, SAVE).
Seu Papel	Focar 100% na lógica de negócio (validações, ações, cálculos).	Chamar BAPIs, Function Modules ou classes legadas para salvar os dados.
Caso de Uso Ideal	<b>Greenfield:</b> Novos desenvolvimentos, onde não há código legado a ser reaproveitado.	<b>Brownfield:</b> Sistemas legados, para expor uma lógica de negócio existente e complexa como um serviço moderno.

# Você Está Pronto: Seu Novo Fluxo de Trabalho do Zero à Aplicação



O domínio destes conceitos transforma você de um mantenedor de código legado em um arquiteto de soluções nativas da nuvem na plataforma SAP.