15 - Arquitetura Baseada em Espaço

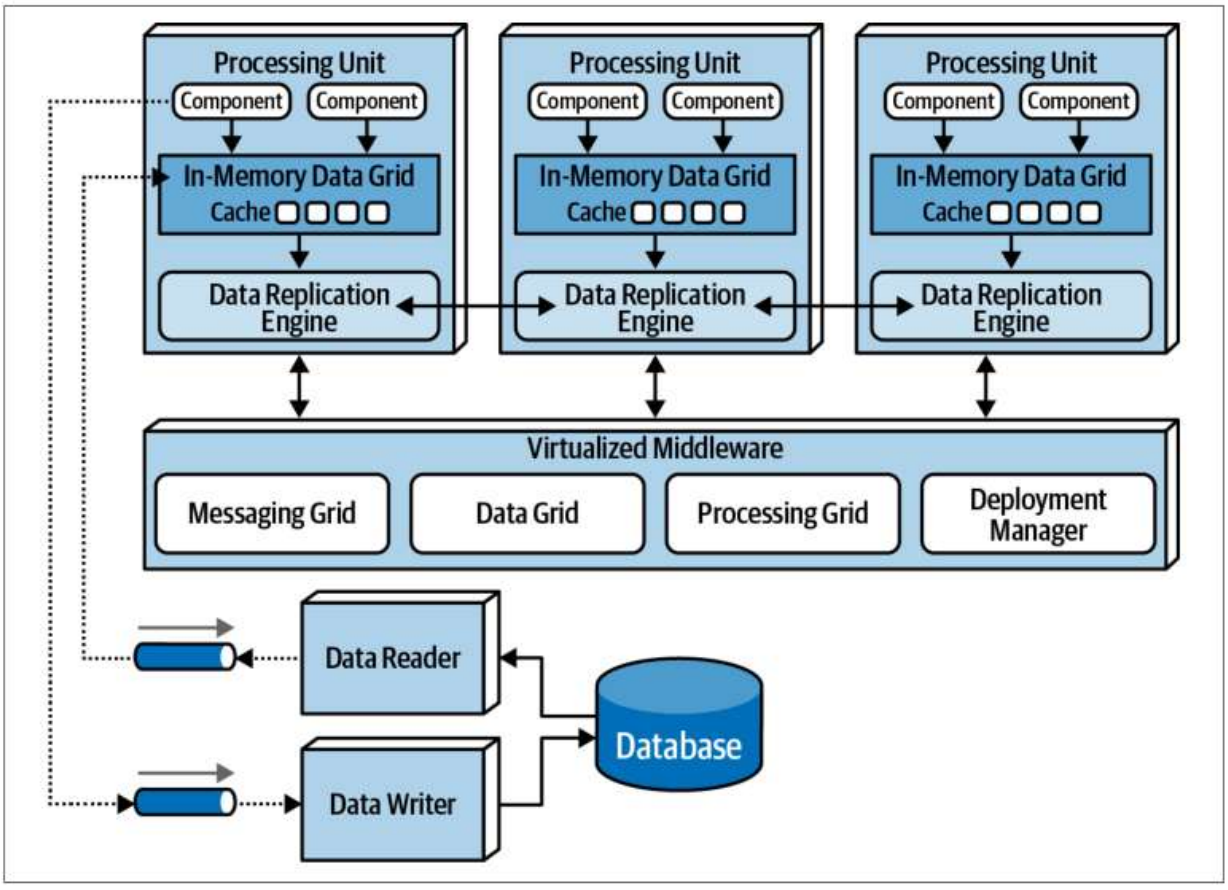
2025-12-22 08:03 am

Escalar uma aplicação web é desafiador:

- Webserver é mais fácil de escalar, que é mais fácil que a aplicação, que é mais fácil que escalar database.
- Acaba ficando escalado desigualmente.
- Baseado em espaço tenta consertar esse problema.
- Bom para aplicações de alto volume e de volume difícil de prever.

Topologia

- Não há gargalo de database central
- Dados são preservados em memória nas unidades de processamento.



- *Figure 15-2. Space-based architecture basic topology*

Unidade de Processamento

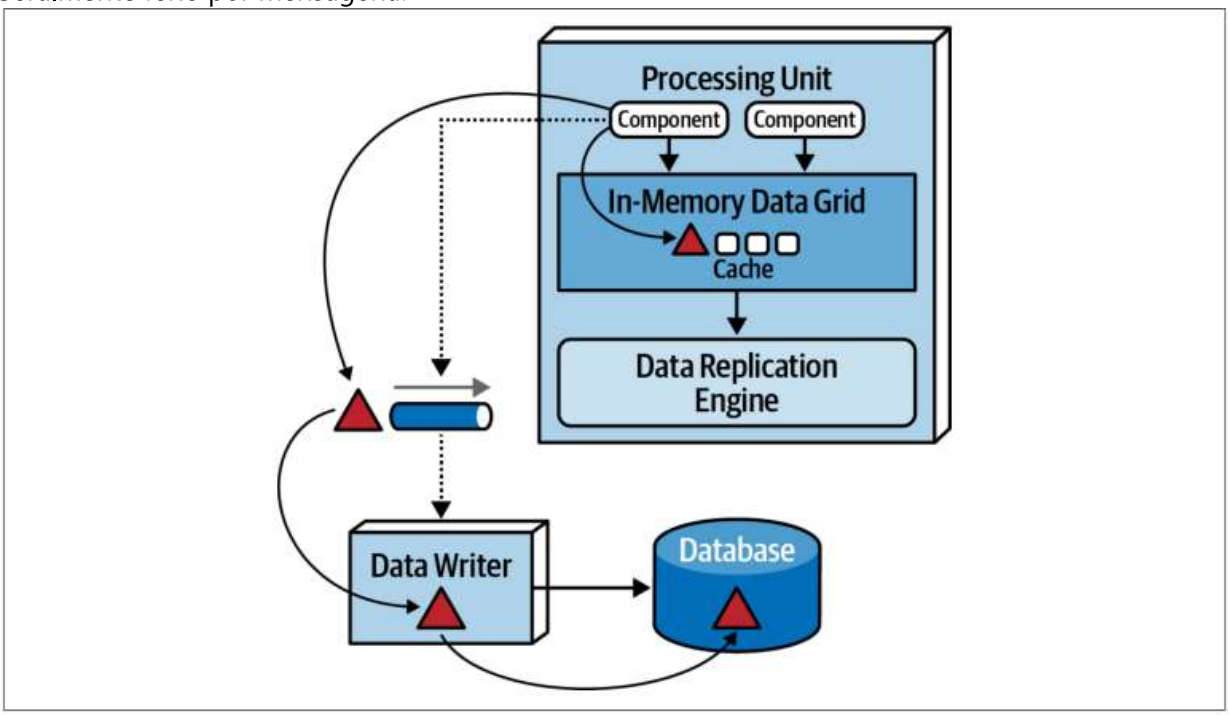
- Contém lógica da aplicação, inteira ou parcial.
  - Pode conter serviços em si.
- Também engine contém engine de dados-em-memória e de replicação.
  - Hazelcast, Apache Ignite ou Oracle Coherence.

Middleware Virtualizado

- Lida com a infraestrutura.
- Contém os seguintes componentes:
    - **Messaging Grid:**
        - Recebe a requisição para dentro do middleware virtualizado.
        - Determina quais unidades de processamento estão disponíveis para execução.
        - Geralmente feito com um webserver com load-balancing (e.g. Nginx)
    - **Data Grid:**
        - Coordena os data grids das unidades de processamento
        - Cada data grid em cada unidade de processamento tem que ter os mesmos dados, datagrid garante isso.
    - **Processing Grid:**
        - Componente opcional
        - Principalmente quando há várias unidades de processamento necessárias para uma única requisição.
        - Orquestra essa interação entre as unidades de processamento.
    - **Deployment Manager:**
        - Gerencia dinamicamente ligamento e desligamento de unidades de processamento a depender da carga.

Data Pumps

- Forma de mandar dados para outro processador.
- Unidades de processamento não leem ou escrevem diretamente a nenhum database.
- Unidade de processamento manda alterações que executou para serem sincronizadas pelas outras unidades.
    - Database será atualizado em algum momento.
- Geralmente feito por mensageria.



- *Figure 15-7. Data pump used to send data to a database*
- Costuma haver um data pump por domínio ou subdomínio.

Data Writers

- Recebe mensagem do data pump e atualiza o banco de dados.

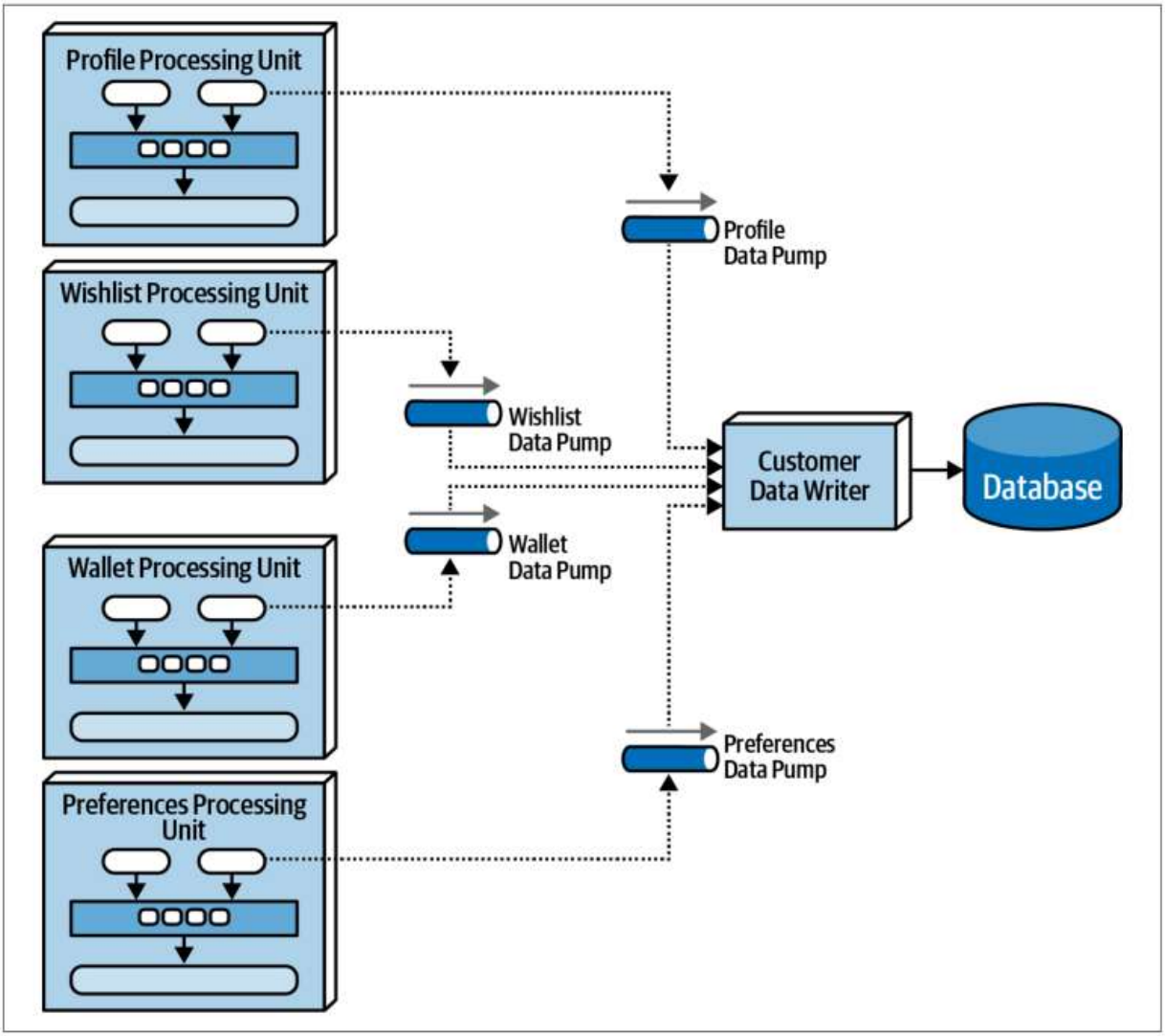


Figure 15-8. Domain-based data writer

- Podem ter granularidade variável
  - Vários pumps para um único writer, por exemplo.
  - Ou então um writer por data pump, também é possível.

Data Readers

- Lê do database e manda pras unidades de processamento via data pump reversa.
- Só são invocados se:
  1. Todas as unidades de processamento com um cache estão derrubadas
  2. Um redeploy das unidade de processamento.
  3. Leitura de dados arquivados que não estão no cache
- Popula o cache das unidades processamento até não precisar mais
- Podem existir por domínio, assim como os writers.
- Junto com os data writers, formam a *camada de abstração de dados*
  - Unidades de processamento não acessam database diretamente, apensa essa camada.
  - Essa camada facilita lidar com mudanças no database.

Colisões de Dados

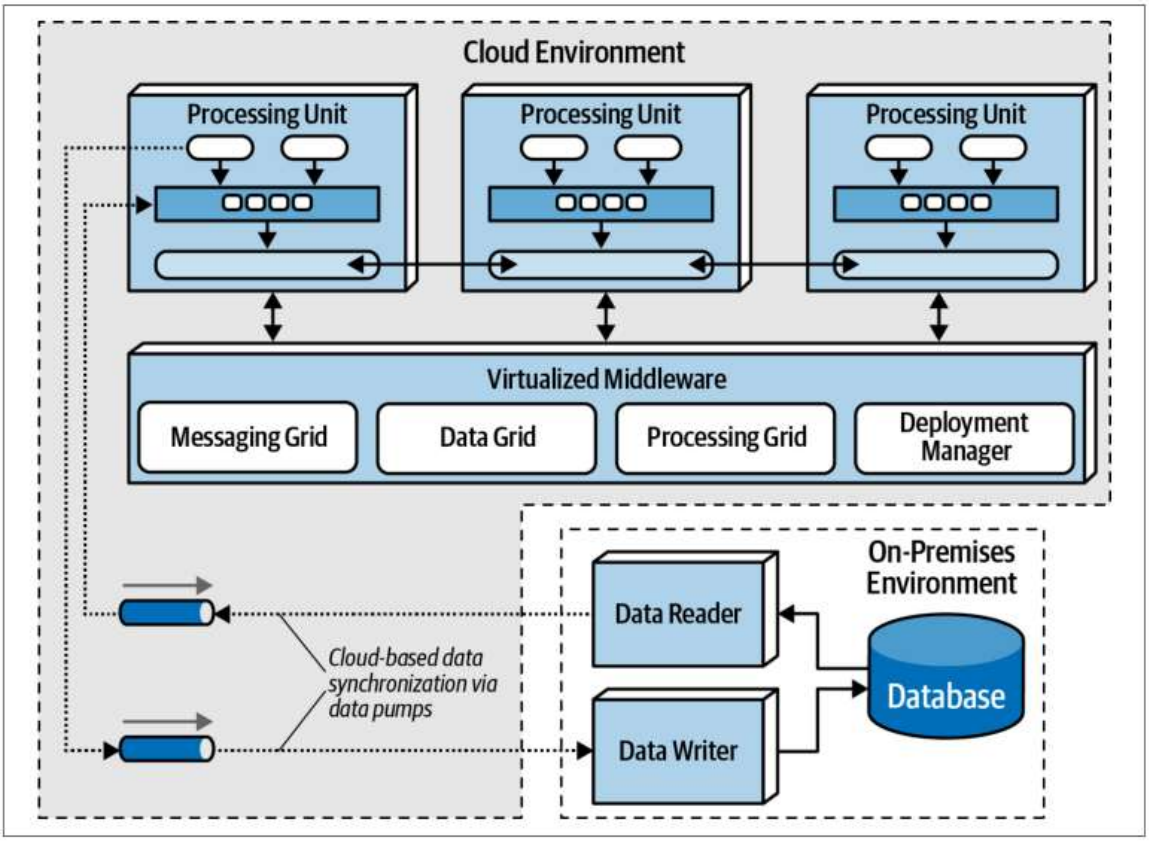
- Cache sofre atualização de um dado pela própria unidade de processamento e por replicação de outra unidade ao mesmo tempo devido a latência na replicação.
    - Replicação vinda externamente sobreescreveria o update local.
- Gera inconsistências.
- Taxa de colisão (`Colisões/unidade de tempo`).

$$CollisionRate = N * \frac{UR^2}{S} * RL$$

    - `N=número de serviços, UR=taxa de updates/ms, S=tamanho do cache, RL=Latência de Replicação`
    - Importante verificar a taxa para diferente momentos de carga sobre o sistema

Implementações Cloud Vs On-Premise

- Permite mesclar cloud e on-premise de um jeito que as outras arquiteturas não permitem.
    - E.g. Manter database on-prem e o resto na cloud.

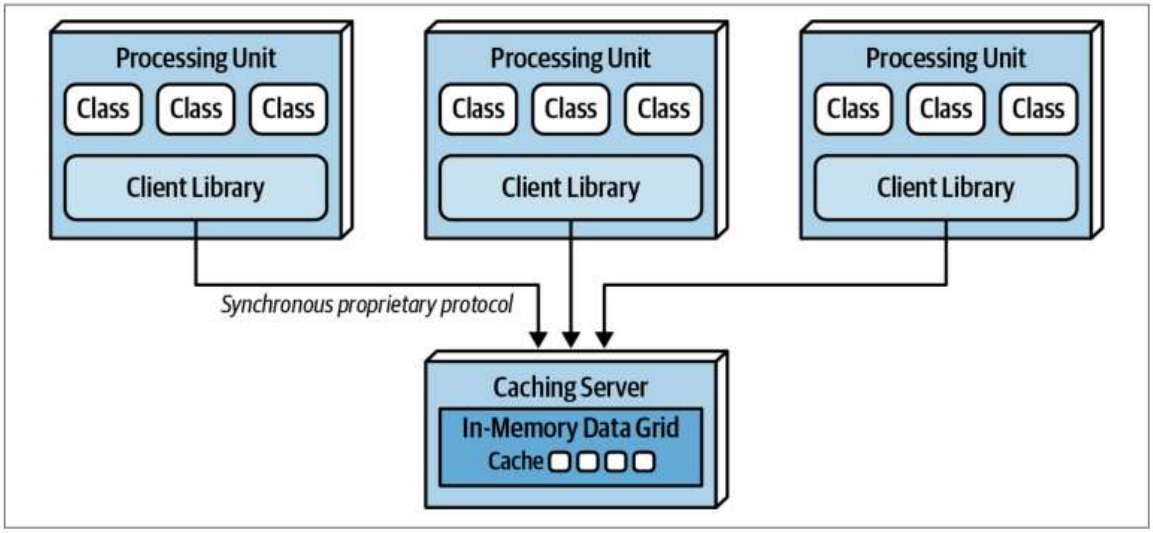


    - *Figure 15-11. Hybrid cloud-based and on-prem topology*

Cache Distribuído Vs Replicado

- Geralmente requer que os cache entre as unidades sejam replicados (i.e. todas as unidades têm todas as informações em um namespace)
    - Mas cache distribuído também é possível (i.e. cada cache tem um pedaço das informações em um namespace)

- Replicação é rápido e tolerante a falhas.
- Mas se o volume do cache for muito grande (> 100MB) isso começa a se tornar um problema.
    - É o caso de usar caching distribuído.
- Caching distribuído requer um servidor externo que tem um cache centralizado.
    - As unidades de processamento então não têm um cache interno mas acessam este servidor central.



    - *Figure 15-13. Distributed caching between processing units*
    - Menor desempenho que cache replicado simples, há mais latência nessa comunicação.
    - Adiciona um ponto de falha central também.
        - Pode ser mitigado replicando o servidor central, mas daí surgem problemas de consistência entre as réplicas.
    - Cache distribuído oferece melhor consistência de dados sobre cache replicado.
        - Dados ficam centralizados.
    - Trade-off *consistência* VS *desempenho/tolerância a falhas*

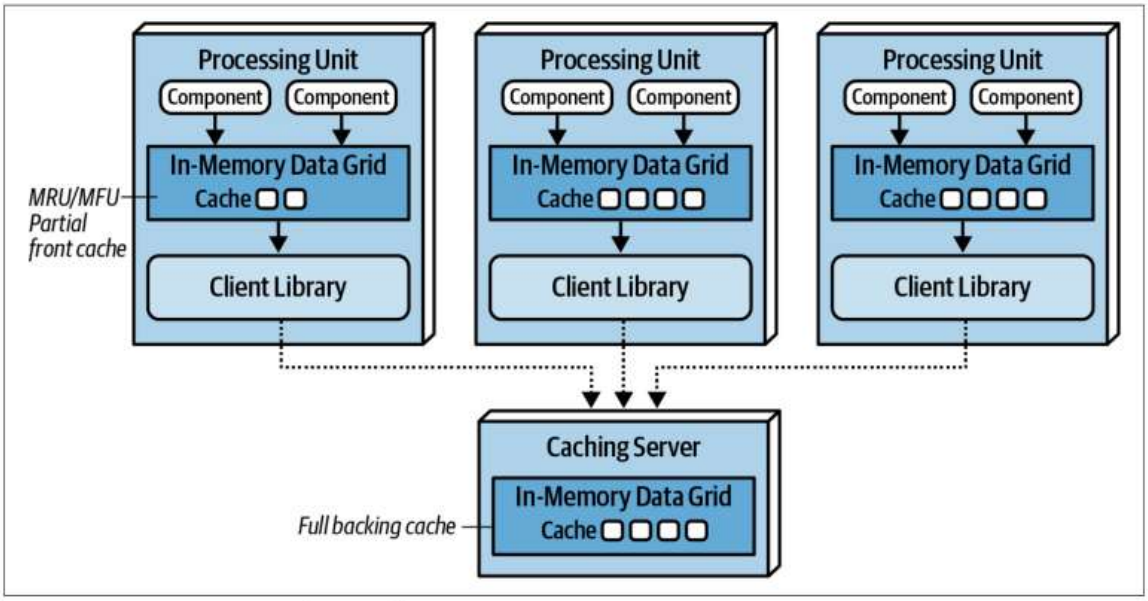
Table 15-1. Distributed versus replicated caching

Decision criteria	Replicated cache	Distributed cache
Optimization	Performance	Consistency
Cache size	Small (<100 MB)	Large (>500 MB)
Type of data	Relatively static	Highly dynamic
Update frequency	Relatively low	High update rate
Fault tolerance	High	Low

    - Pode ser bom usar os dois tipos de caching em pedaços diferentes da aplicação.
        - e.g. *inventário* requer alta consistência, *perfil do usuário* maximiza desempenho

Considerações de Near-Cache

- Near-cache é um caching híbrido.
    - In-memory com caching híbrido.
        - Caching distribuído --> *full backing cache*
        - Data Grid em memória --> *front cache*
- Front cache contém um subconjunto do backing caching
    - Vai renovando o conteúdo do front cache usando uma *eviction policy*
        - Por frequência, importância, recência, dos dados ou aleatório.



- *Figure 15-14. Near-cache topology*

- Front caches não são sincronizados entre si.
  - Isso cria inconsistências de desesmpenho e responsividade entre as unidades de processamento.
- Por isso, Near-Cache, é **NÃO RECOMENDADO** para Arquitetura Baseada em Espaço.

Exemplos de Implementação

- Adequado para aplicações com alta variação de carga, com no mínimo 10.000 usuários simultâneos.
- **Sistema de Ingresso para Shows**
  - Volume de carga é baixo até ser altíssimo em instantes. Alta elasticidade.
  - Disponibilidade dos assentos deve ser altamente consistente e atualizar rapidamente.
    - Updates síncronos a um database seria impossível
  - Deployment manager lida com elasticidade da carga
    - De preferência antes das vendas começarem de fato.
- **Sistema de Leilão Online**
  - Também requer alto desempenho e elasticidade.

Características

Architecture characteristic	Star rating
Partitioning type	Domain and technical
Number of quanta	1 to many
Deployability	★★★
Elasticity	★★★★★
Evolutionary	★★★
Fault tolerance	★★★
Modularity	★★★
Overall cost	★★
Performance	★★★★★
Reliability	★★★★
Scalability	★★★★★
Simplicity	★
Testability	★

- *Figure 15-15. Space-based architecture characteristics ratings*
- Maximiza elasticidade e desempenho.
- Mas complexo e difícil de testar.
    - Difícil garantir que não há perda de dados caso algum elemento falhe.
    - Difícil testar volume de carga de uma situação extrema real.
- Tanto particionada tecnicamente quanto por domínio.
    - Unidade de processamento pode ser separada por domínio
    - Mas responsabilidades também tem camadas (como a abstração de dados) estritamente técnicas.
- Permite vários quanta.
    - Database não é um gargalo de características.