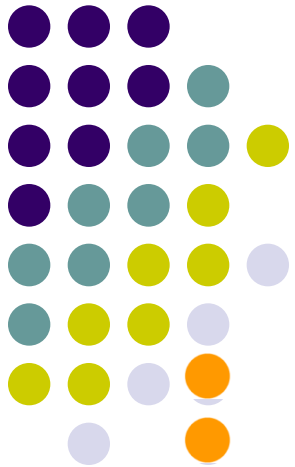
Arquitetura de Software

N-Tier, Clean & Reactive

1-Tier, 2-Tier, 3-Tier & N-Tier, Clean, Reactive

José Motta Lopes
josemotta@bampli.com



Agenda

- Objetivos
  - Separação de Conceitos
  - Responsabilidade Única
  - Não se Repita
  - Inversão da Dependência
  - Melhores Práticas
- Arquitetura 1-Tier, 2-Tier, 3-Tier & N-Tier
  - Benefícios e Desvantagens
  - Dependências Transitivas
- Arquitetura Clean
  - Regra de Dependência
  - Entidades
  - Casos de Uso
  - Adaptadores de Interface
  - Frameworks & Drivers
- Sistemas Reativos
  - Utilização

Separação de Conceitos

PRINCÍPIO DA SEPARAÇÃO DE CONCEITOS

Objetivo de evitar um emaranhado de código poluindo seu software.

- **Evitar misturar diferentes responsabilidades no mesmo:**
  - **Método**
  - **Classe**
  - **Projeto**

- **As principais responsabilidades:**
  - **Acesso de Dados**
  - **Regras do Negócio e Modelo do Domínio**
  - **Interface Usuário**

Responsabilidade Única

PRINCÍPIO DA RESPONSABILIDADE ÚNICA

Objetivo de evitar acoplar fortemente os seus recursos juntos.

- **Funciona em conjunto com a Separação de Conceitos**
- **Classes devem focar em uma responsabilidade única**
- **Uma única razão para mudar**

Não se Repita

PRINCÍPIO DRY - NÃO SE REPITA

A repetição de código é a raiz de todo software maligno.

- **Reduzir a duplicação de código e os problemas daí decorrentes**
- **Refatore código repetitivo em *funções***
- **Agrupe *funções* em *classes coesas***
- **Agrupe *classes* em *folders* e *namespaces* por:**
  - **Reponsabilidade**
  - **Nível de abstração, etc**
- **Agrupe *classes folders* em *projetos***

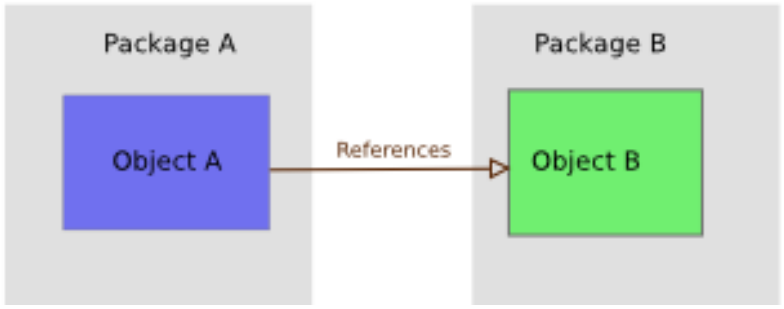
Inversão da Dependência

VOCÊ SOLDARIA O FIO DA LÂMPADA DIRETO NA TOMADA?

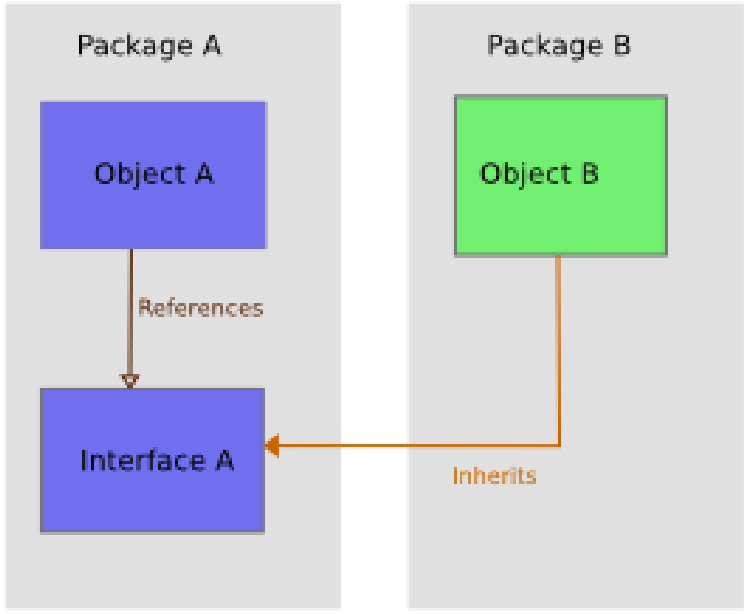
Inverter e injetar Dependências.

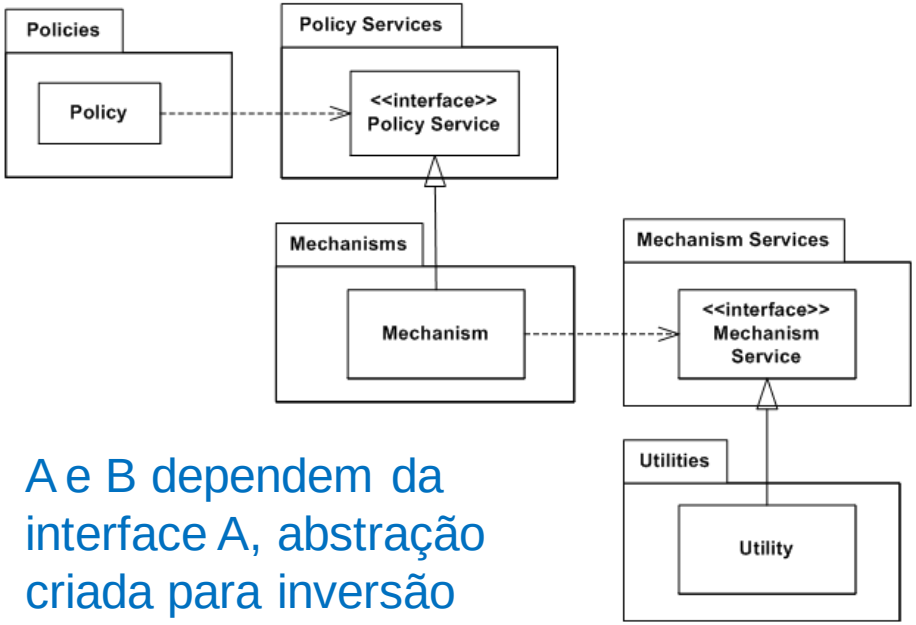
- **Tanto as classes de alto nível quanto as de implementação de detalhes devem depender de *abstrações (interfaces)*.**
- **Classes devem seguir o *Princípio das Dependências Explícitas*:**
  - **Solicite todas as dependências por meio de seu construtor.**
- **Corolário: Abstrações/interfaces definidas em local acessível a:**
  - **Serviços implementados em baixo nível**
  - **Serviços de negócio de alto nível**
  - **Entrypoints da interface do usuário**

Inversão da Dependência



A depende de B, limitando o reuso de A





A e B dependem da interface A, abstração criada para inversão

Melhores práticas

O projeto e a estrutura da solução devem ajudar a reforçar que:

- **Classes UI**
- **Classes de negócio/domínio**

Não devem depender diretamente de classes de infraestrutura

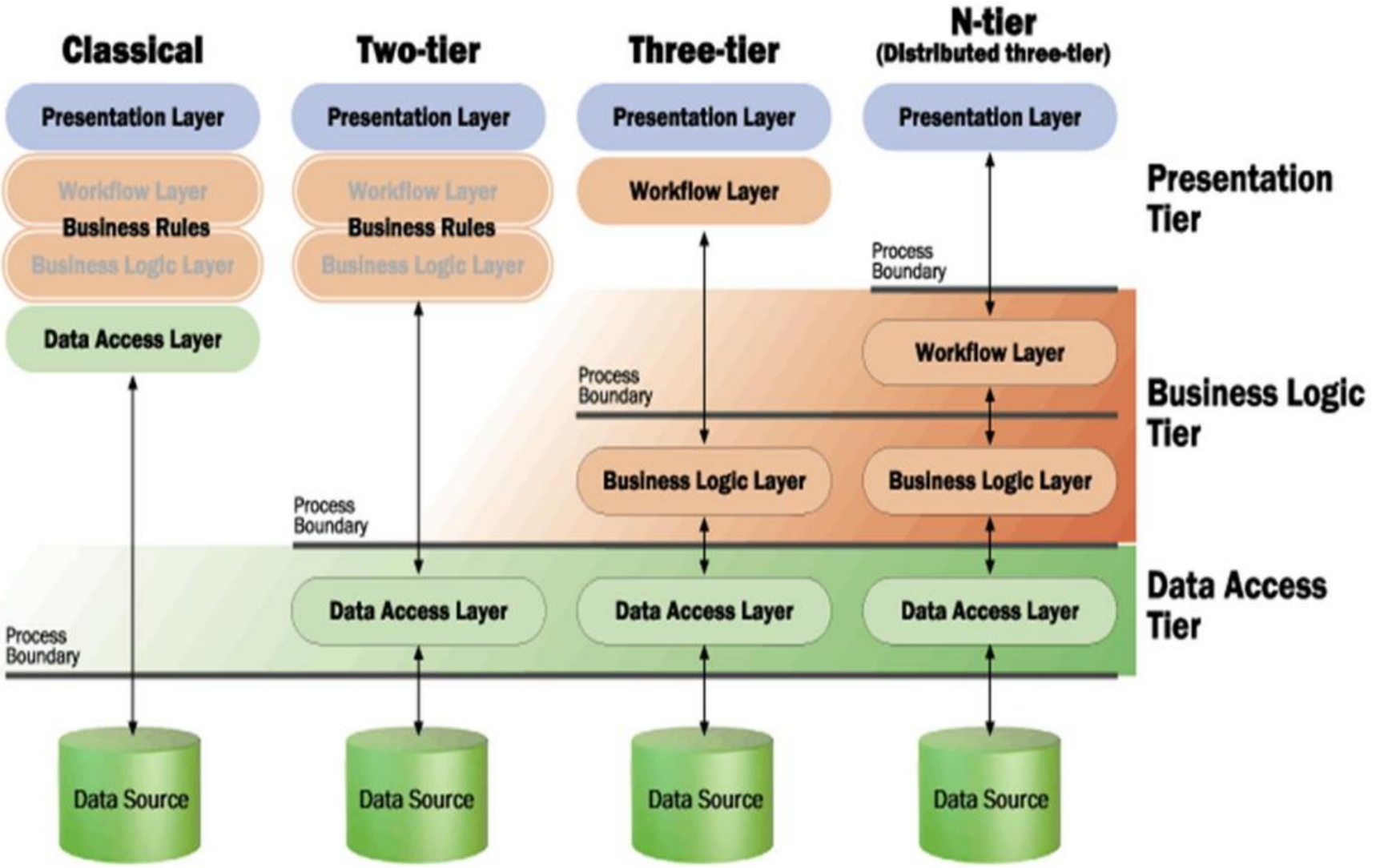
Melhores práticas

Repetição de qualquer coisa é um problema.

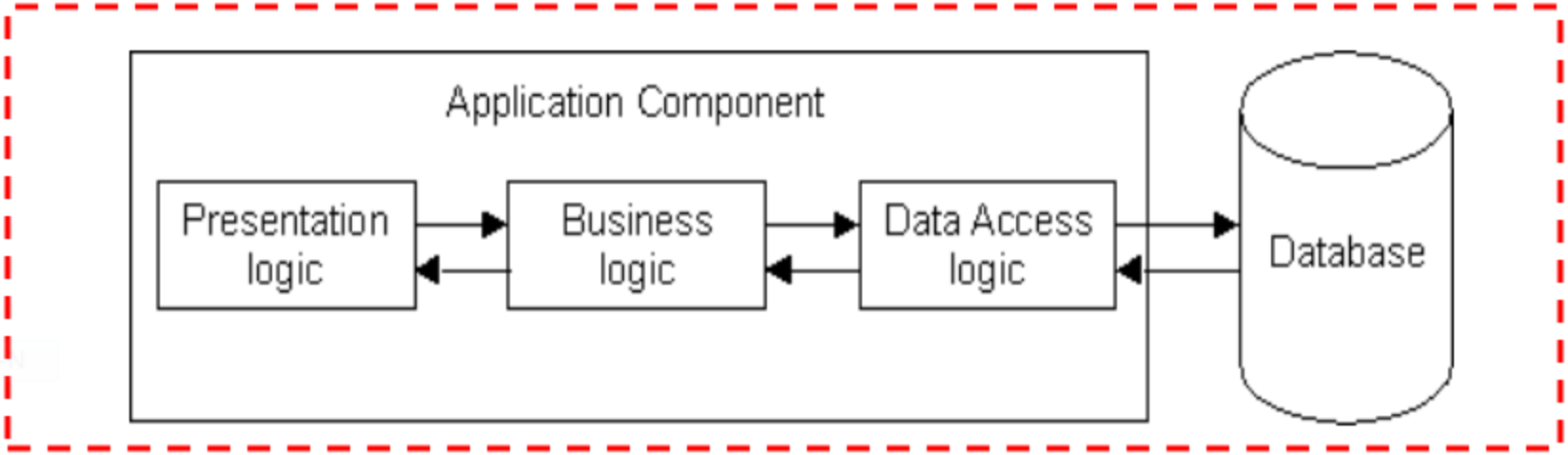
- **Lógica de query**
- **Lógica de validação**
- **Políticas**
- **Tratamento de erros**

Buscar padrões que evitem a repetição *copiar/colar*.

Arquitetura N-Tier

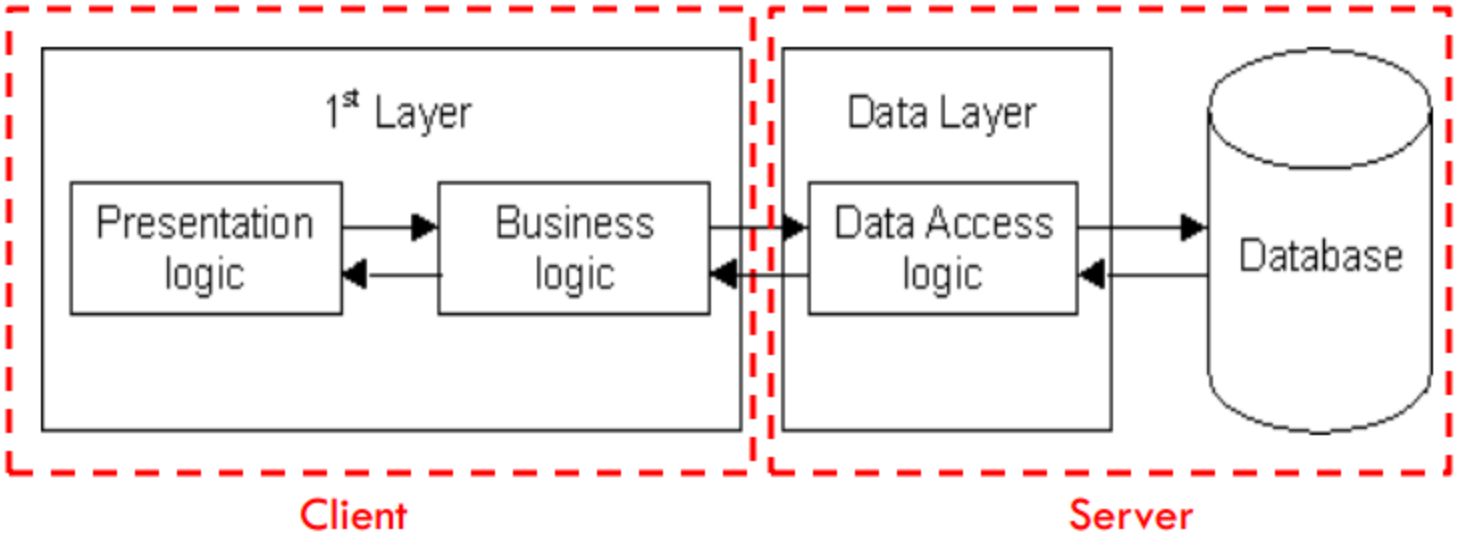


1-Tier



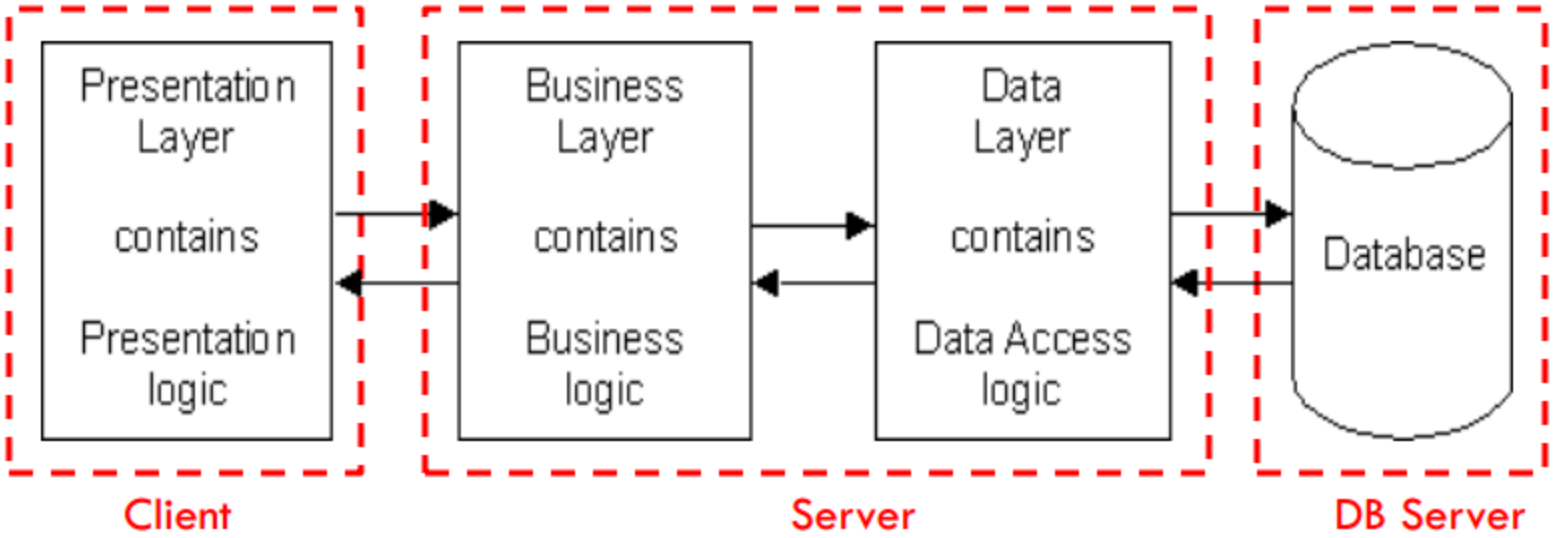
- **Todas as 3 camadas na mesma máquina**
  - Código e processamento mantidos em uma só máquina
- **Presentation, Logic, Data layers estão fortemente conectados**
  - Escalabilidade: Processador único torna difícil aumentar volume de processamento.
  - Portabilidade: Migrar de máquina significa reescrever tudo.
  - Manutenção: Mudança em uma camada afeta outras camadas.

2-Tier



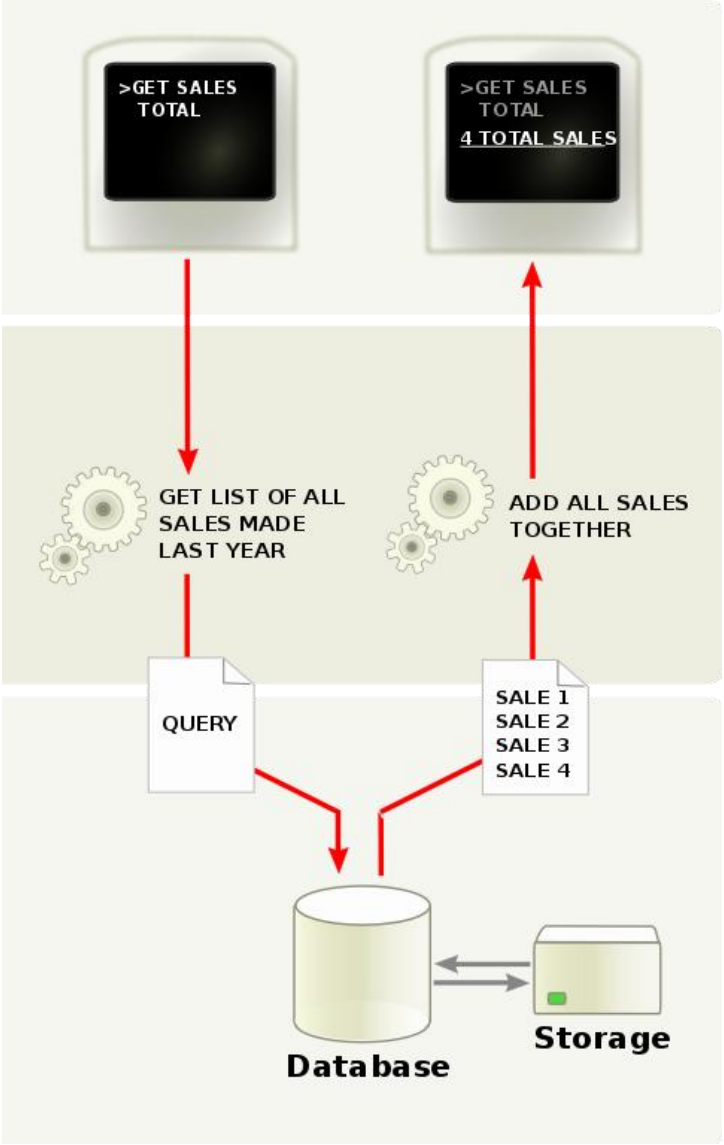
- **Banco de Dados roda em Servidor**
  - Separado do cliente
  - Fácil migrar para outro banco de dados
- **Presentation, Logic, Data layers ainda fortemente conectados**
  - Carga pesada no servidor
  - Potencial congestionamento na rede
  - Presentation ainda acoplado à lógica de negócios

3-Tier



- Camadas podem potencialmente rodar em máquinas diferentes
- Presentation, Logic, Data layers desacoplados

Arquitetura N-Tier



Presentation Tier:

Traduz tarefas e resultados para algo que o usuário possa entender.

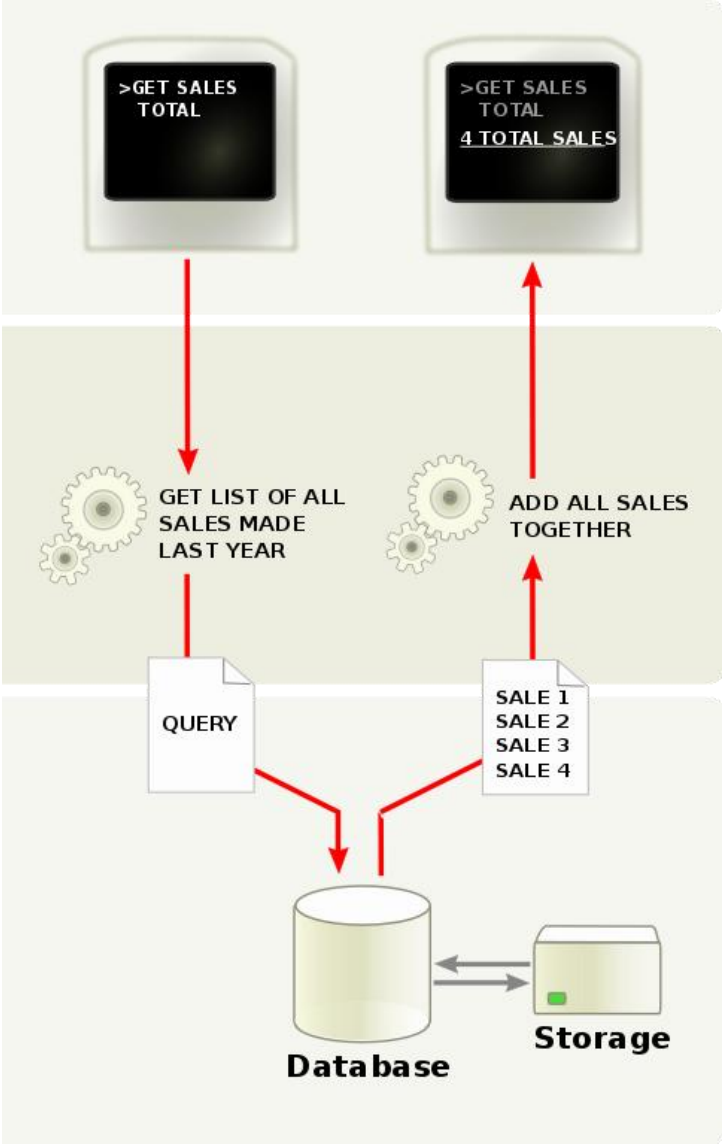
Logic Tier:

Coordena a aplicação, processa comandos, avalia, calcula e decide sobre a lógica. Move dados entre as duas camadas vizinhas.

Data Tier:

Informações são armazenadas e consultadas em um banco de dados ou sistema de arquivo.

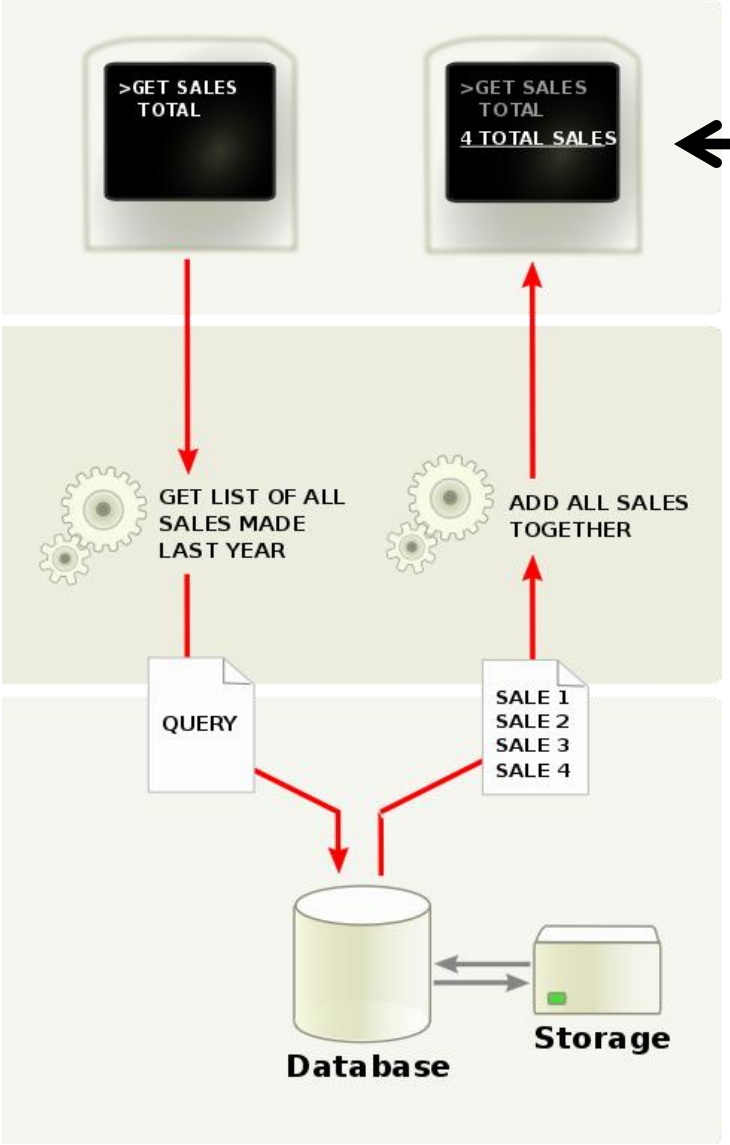
Arquitetura N-Tier



Princípios Arquitetura

- Arquitetura Cliente-Servidor
- Cada tier (Presentation, Logic, Data) deve ser independente e não expor dependências relativas à implementação
- Tiers desconectados não devem se comunicar
- Mudanças de plataforma afetam somente a camada que a utiliza

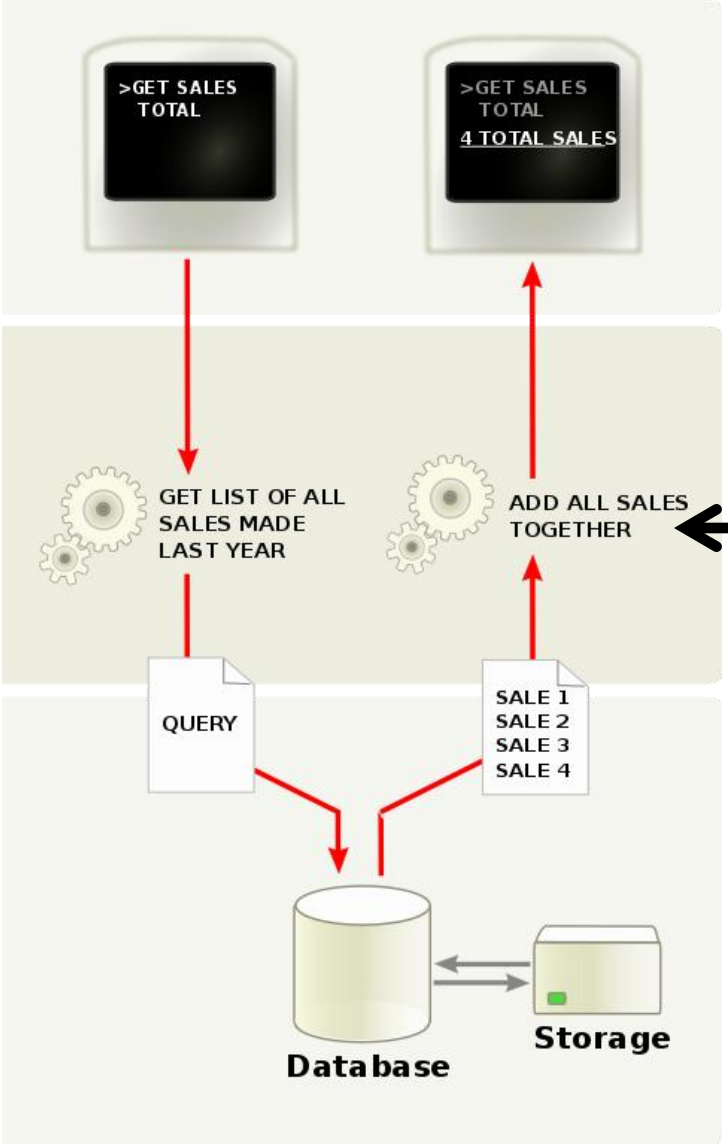
Arquitetura N-Tier



Presentation Layer

- Interface com Usuário
- Cuida da interação com o usuário
- Também chamada de GUI, client view ou front-end
- Não deve conter lógica de negócio ou código de acesso a dados

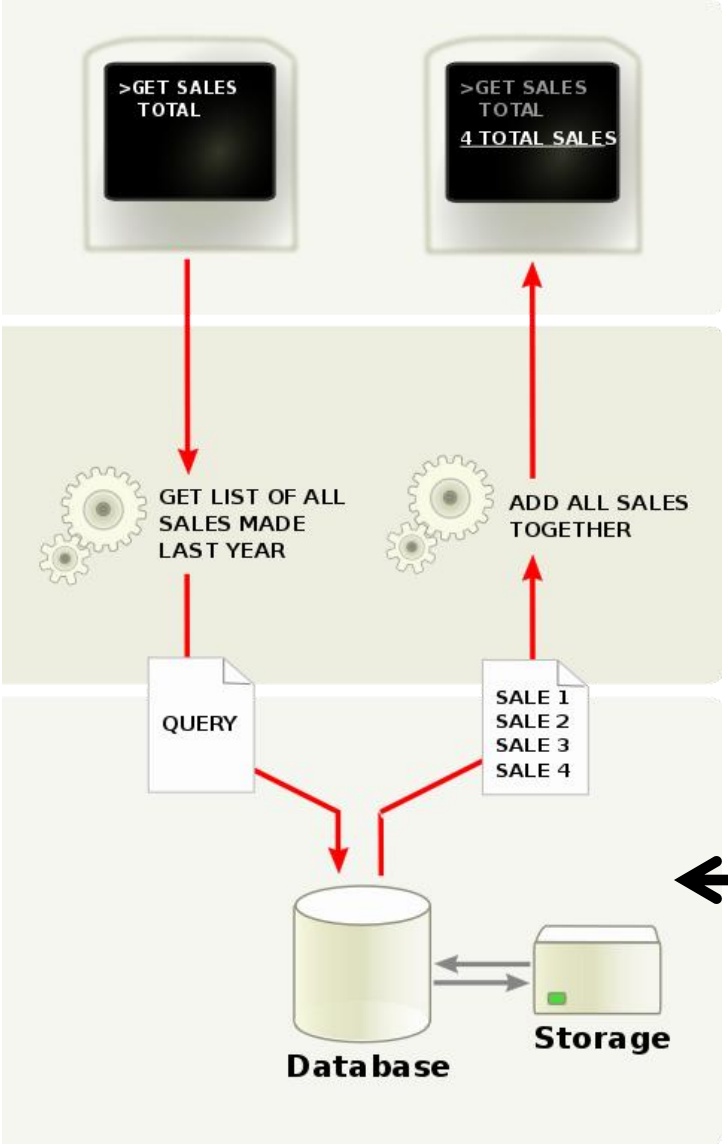
Arquitetura N-Tier



Logic Layer

- Conjunto de regras para processar a informação
- Pode acomodar muitos usuários
- Também chamada de middleware ou back-end
- Não deve conter apresentação ou código de acesso a dados

Arquitetura N-Tier



Data Layer

- A camada de armazenamento físico para a persistência de dados
- Gerencia o acesso ao DB ou sistema de arquivos
- Algumas vezes chamado de back-end
- Não deve conter apresentação ou lógica de negócios

Benefícios N-Tier

- Independência das camadas
- Reutilização do código dos componentes
- Segmentação do trabalho da equipe agiliza o desenvolvimento
- Facilidade de Manutenção
- Acoplamento (levemente) enfraquecido

No código altamente acoplado:

- as dependências entre as partes são densas
- muitas coisas dependem de muitas outras
- é mais difícil de se entender e manter
- tende a ser frágil, falhando facilmente em caso de mudança

Desvantagens N-Tier

- **Dependências Transitivas**
- **Alguma complexidade**
- **Ainda um acoplamento forte**

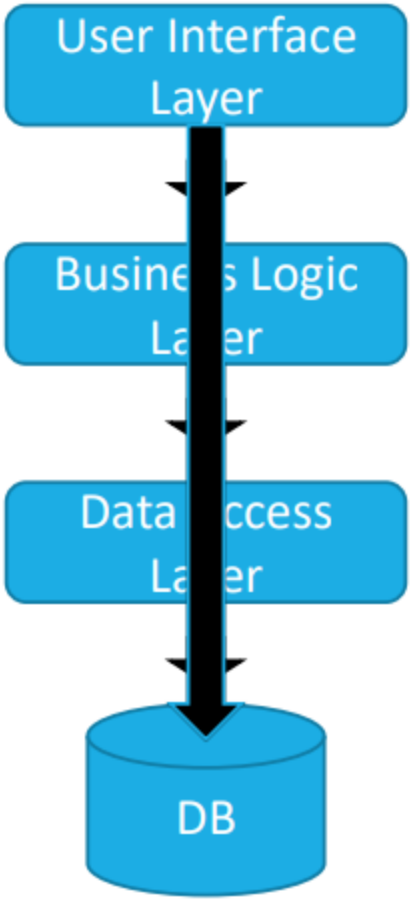
Acoplamento estático:

- **Pode-se dizer que a classe A é estaticamente acoplada à classe B, se o compilador precisa da definição de B para compilar A.**

Dependência transitiva:

- **Se A depende de B e B depende de C, então A depende de B e C.**

Dependências Transitivas



TUDO

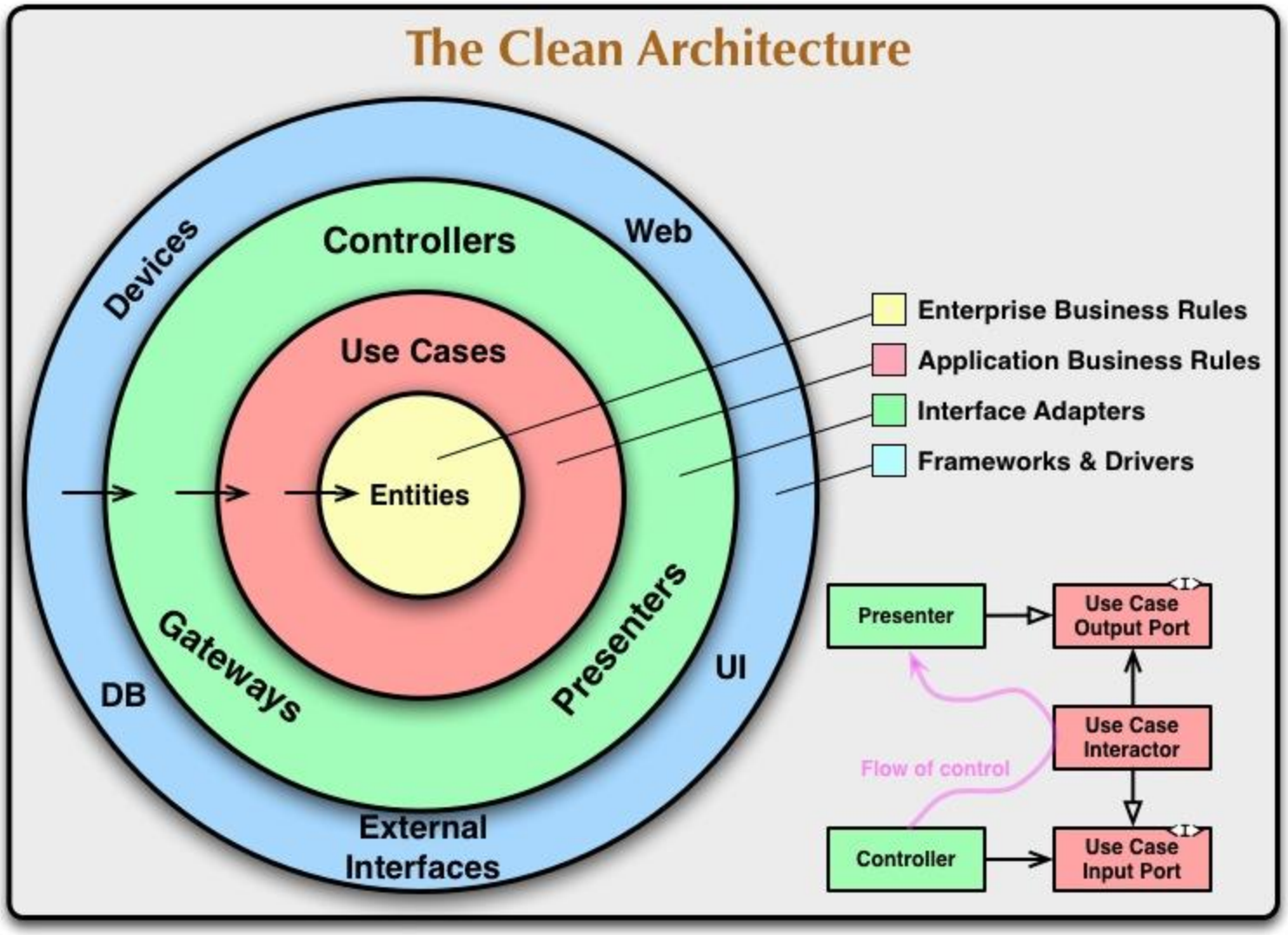
depende do banco de dados

Arquitetura Clean

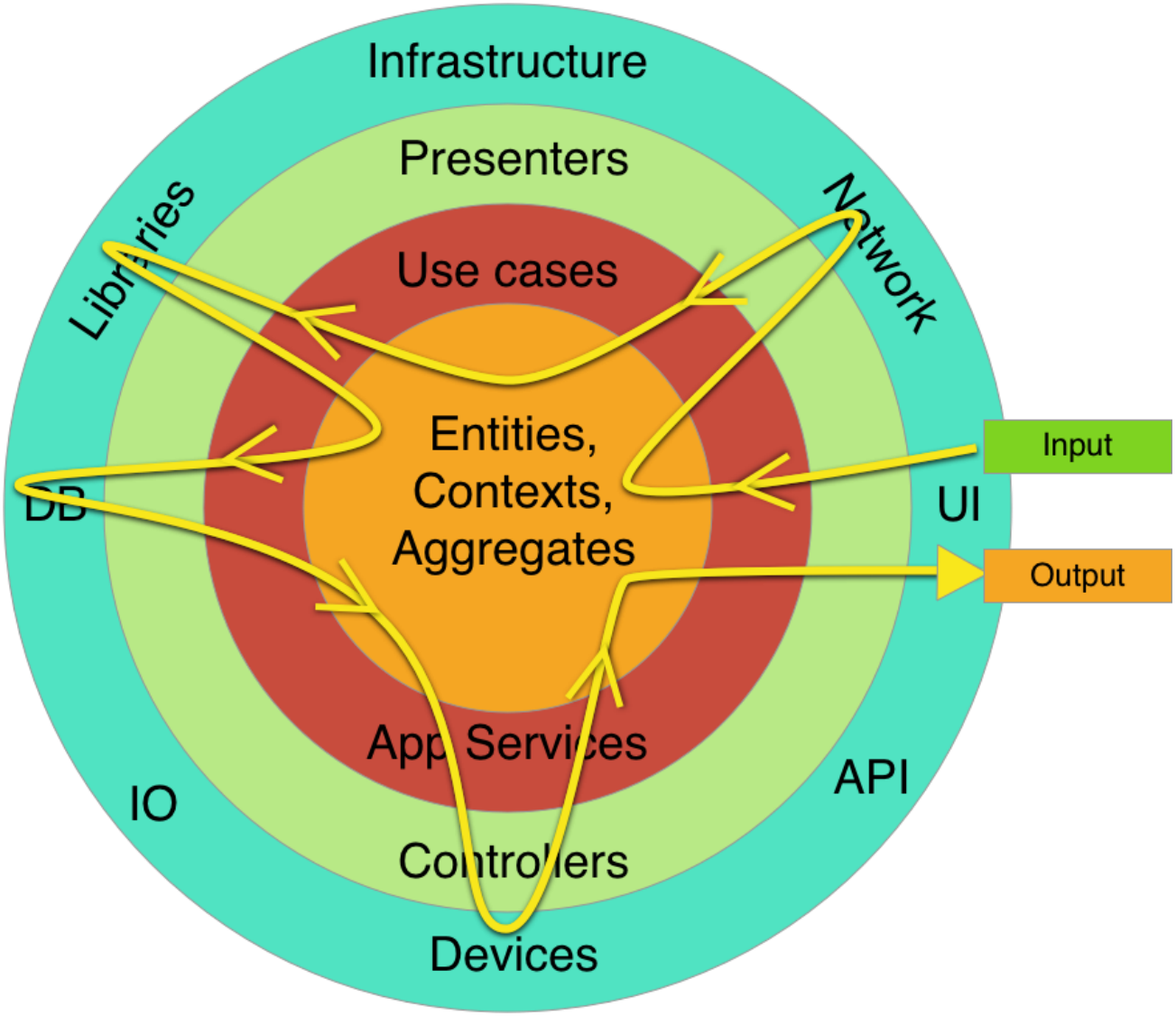
ARQUITETURAS PARA PRODUÇÃO DE SISTEMAS

- **Independentes de Frameworks**
- **Testáveis**
- **Independentes de interface do usuário**
- **Independente do banco de dados**
- **Independente de qualquer agência externa**

Arquitetura Clean



Arquitetura Clean

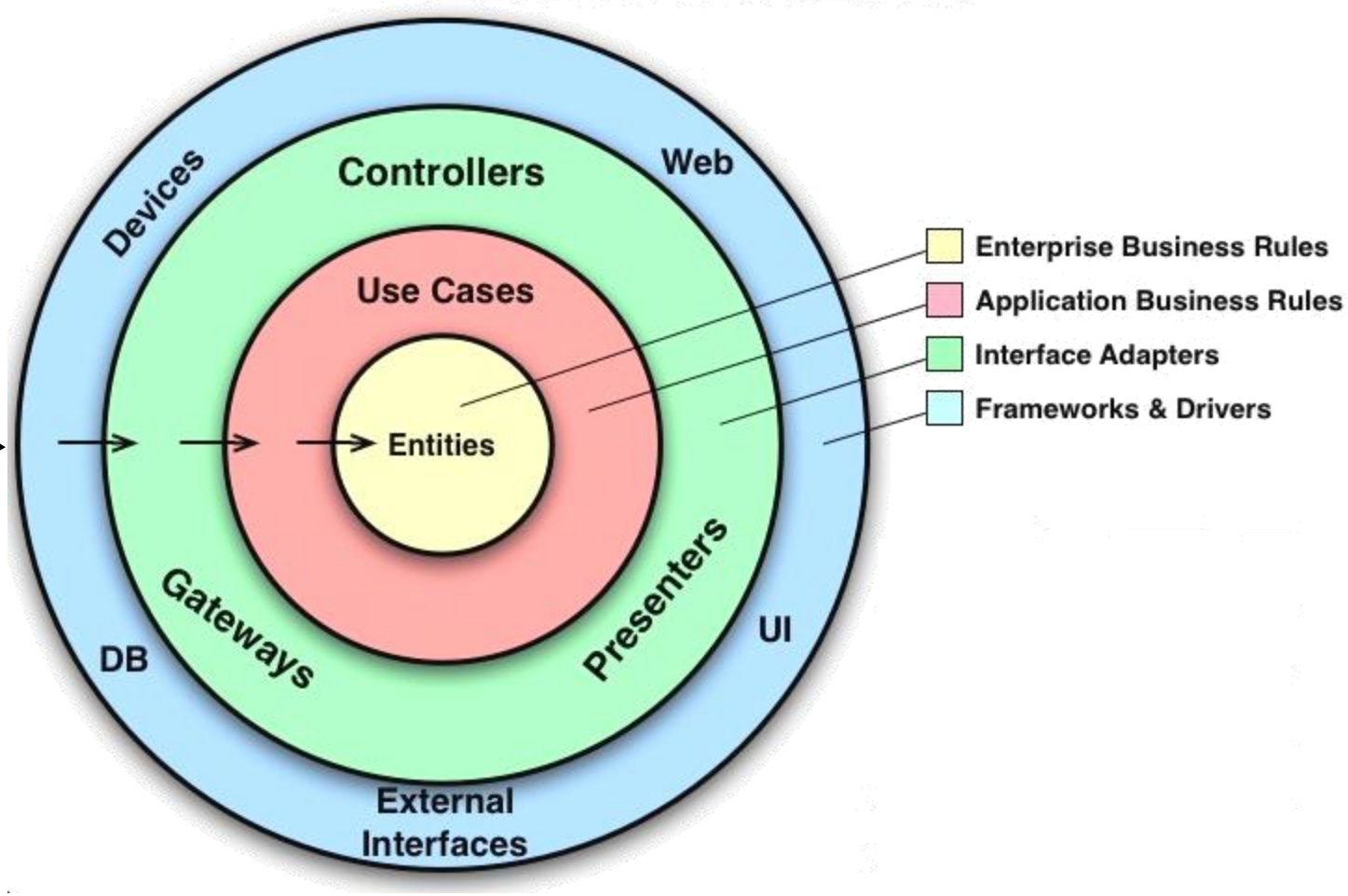


Arquitetura Clean

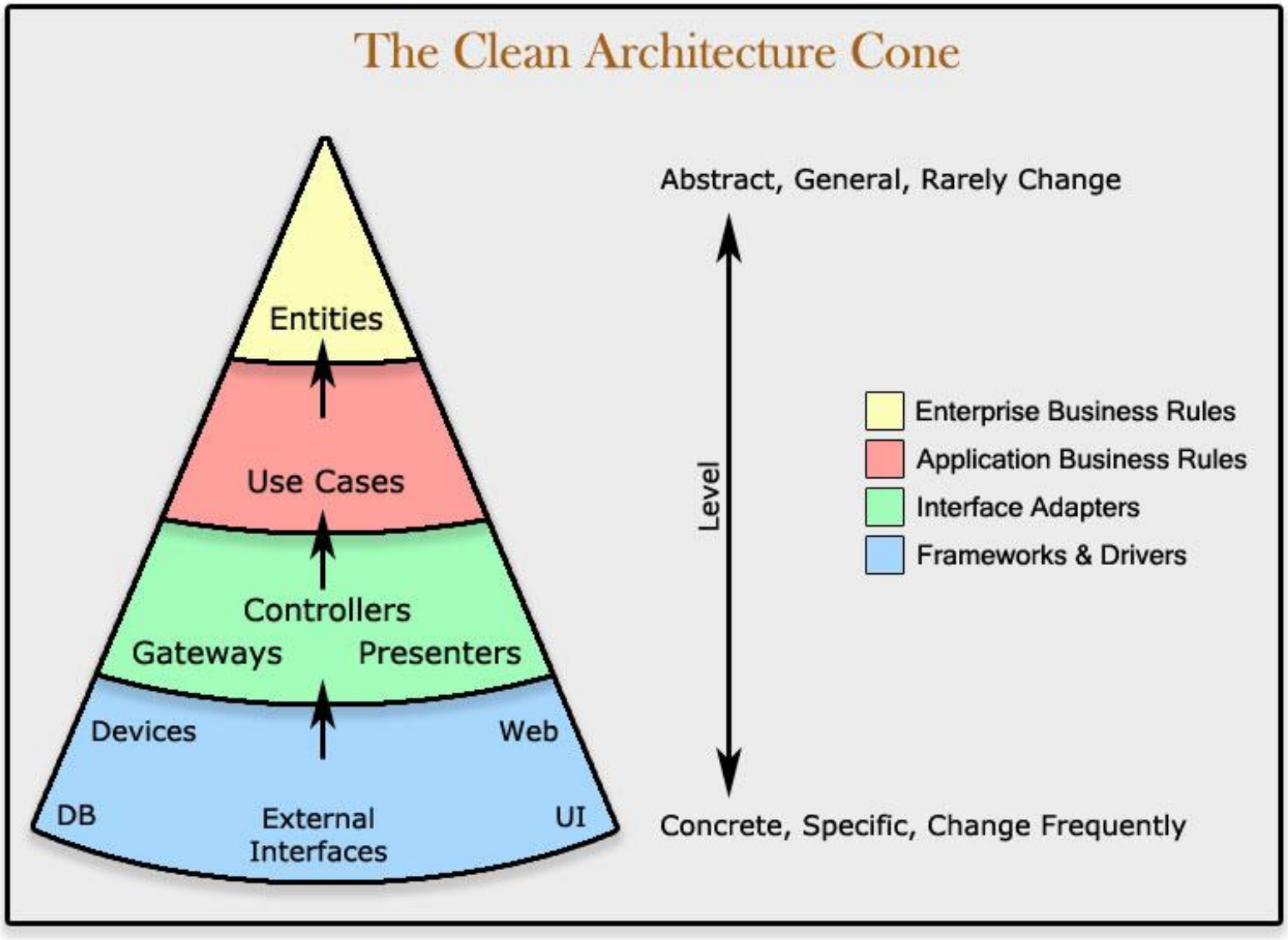
Regra de Dependência

Camadas internas não sabem nada sobre camadas externas.

Inclui funções, classes, variáveis ou quaisquer outras entidades de software.



Arquitetura Clean



Entidades

ENTERPRISE WIDE BUSINESS RULES

- **Encapsulam regras de negócio corporativos.**
  - **Objeto com métodos**
  - **Conjunto de estruturas de dados e funções**
- **Em aplicações simples, são os objetos do negócio.**
- **Encapsulam regras mais gerais e de mais alto nível.**
- **São as menos afetadas com as mudanças externas.**
- **Mudanças operacionais não devem afetar as entidades.**

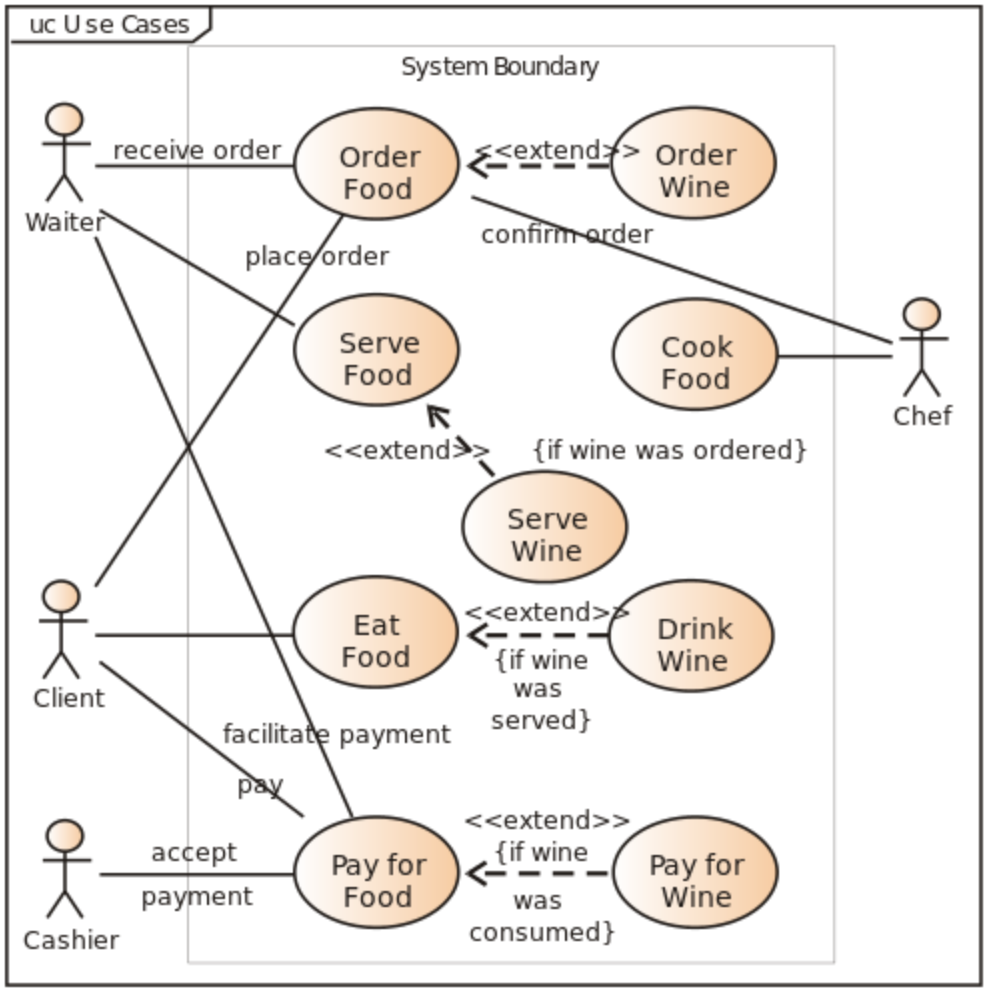
Casos de Uso

APPLICATION SPECIFIC BUSINESS RULES

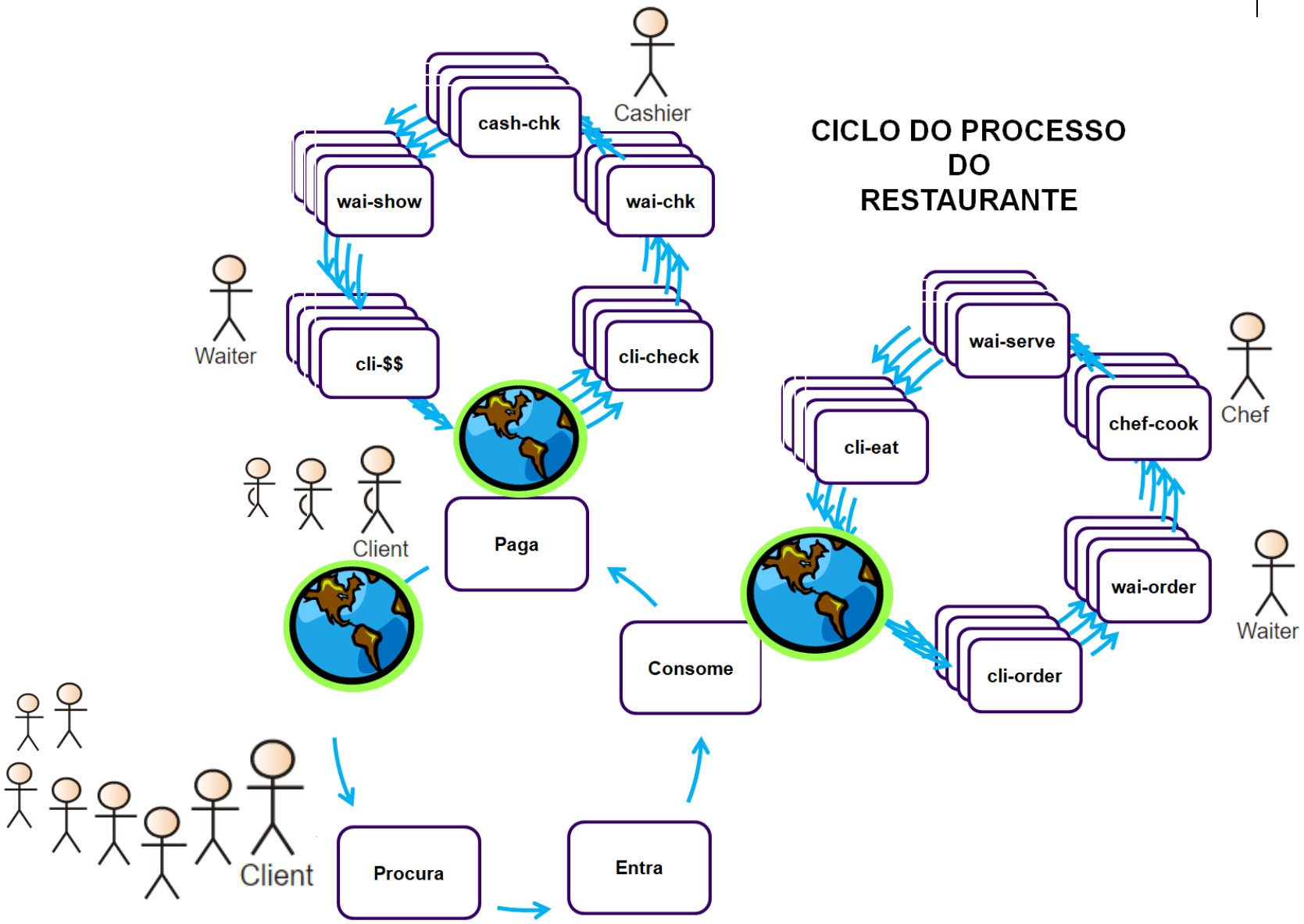
- **Encapsulam e implementam os casos de uso do sistema.**
- **Orquestram o fluxo de dados de/para entidades.**
- **Direcionam as entidades a usar as regras gerais do negócio para alcançar as metas do caso de uso.**
- **Não se espera que mudanças nesta camada afetem as entidades.**
- **Também não se espera que essa camada seja afetada por mudanças externas, como banco de dados, UI ou framework.**
- **Espera-se que mudanças na operação da aplicação afetem os casos de uso e em consequência, o software dessa camada.**

Casos de Uso

INTERAÇÕES DE UM RESTAURANTE COM ATORES DIVERSOS



Ciclo do Processo



Adaptadores de Interface

INTERFACE ADAPTERS

- **Convertem dados:**
  - **do formato mais conveniente para casos de uso e entidades;**
  - **para formatos convenientes aos agentes externos, DB e Web.**
- **Inclui MVC de uma GUI, com presenters, views e controllers.**
- **Os modelos são estruturas de dados:**
  - **passadas dos controladores para os casos de uso,**
  - **e de volta dos casos de uso para os presenters e views.**
- **Se o banco de dados é um SQL server, todo o SQL fica aqui.**

Frameworks & Drivers

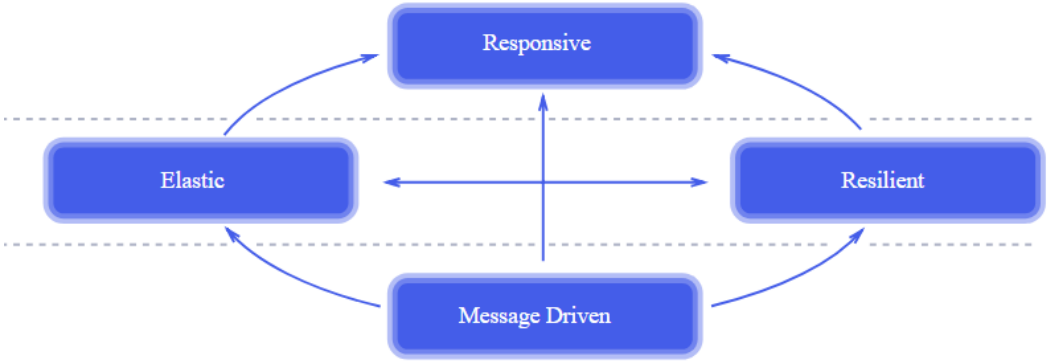
FRAMEWORKS & DRIVERS

- **Composto de frameworks e ferramentas:**
  - **Banco de Dados**
  - **Web Framework, etc**
- **Não possui muito código, só para se ligar às camadas internas.**
- **Esta camada contem os detalhes:**
  - **A Web é um detalhe;**
  - **O Banco de Dados é um detalhe;**

Sistemas Reativos

O MANIFESTO REATIVO

- **RESPONSIVO**
- **RESILIENTE**
- **ELÁSTICO**
- **ORIENTADO POR MENSAGEM**



Responsivo

O SISTEMA RESPONDE DE MANEIRA OPORTUNA, SE POSSÍVEL.

- **Fornece tempos de resposta rápidos e consistentes**
- **Estabelece limites superiores confiáveis**
- **Qualidade de serviço consistente**
- **Desta forma, a consistência:**
  - **Simplifica o tratamento de erros**
  - **Injeta confiança no usuário final**
  - **Incentiva interação adicional**

Resiliente

O SISTEMA PERMANECE RESPONSIVO EM CASO DE FALHA.

- **Isso se aplica não apenas a sistemas de missão crítica**
- **Sistemas não resilientes não respondem após uma falha**
- **Não sobrecarrega o cliente com tratamento de falhas**
- **A resiliência é alcançada por:**
  - **Replicação: garante a alta disponibilidade, quando necessário**
  - **Contenção: falhas são contidas dentro de cada componente**
  - **Isolamento: partes falham e se recuperam sem comprometer o sistema**
  - **Delegação: a recuperação é delegada a outro componente externo**

Elástico

O SISTEMA PERMANECE RESPONSIVO SOB CARGA VARIÁVEL.

- **Reaje a mudanças na taxa de entrada**
- **Aumenta ou diminui recursos alocados para servir entrada**
- **Projeto não deve ter pontos de contenção**
- **Projeto não deve ter restrições centrais**
- **Projeto com capacidade de:**
  - **Fragmentar ou replicar componentes**
  - **Distribuir a entrada entre eles**
- **Algoritmos preditivos e reativos, com desempenho ao vivo**
- **Alcançar elasticidade econômica em plataformas commodity**

Orientado por Mensagem

PASSAGEM ASSÍNCRONA DE MENSAGENS ENTRE COMPONENTES

- **Baixo acoplamento, isolamento e transparência de localização**
- **Todos os componentes suportam mobilidade**
- **Modelagem e monitoração de filas de mensagens no sistema**
- **Gerenciamento de carga, elasticidade e controle de fluxo**
- **Aplicar contrapressão, quando necessário**
- **Mesma construção e semântica através de cluster ou host simples**
- **Comunicação sem bloqueio**
- **Destinatários consomem recursos enquanto estão ativos**

Contrapressão

BACK-PRESSURE

- **Componente sob stress está lutando para se manter**
- **Para o sistema é inaceitável que o componente:**
  - **Falhe catastroficamente**
  - **Perca mensagens de forma descontrolada**
- **Se não pode lidar e nem pode falhar, comunica “chefes” upstream**
- **“Chefes” aliviam carga de trabalho**
- **Componente responde normalmente ao stress, sem colapsar**
- **Sistema aplica outros recursos para ajudar a distribuir a carga**

Sem bloqueio

NON-BLOCKING

- **Programação concorrente com threads competindo por recurso**
- **Há mútua exclusão protegendo o recurso**
- **Algoritmo non-blocking não adia indefinidamente a execução**
- **API permite acesso ao recurso, se disponível**
- **Senão, retorna informado que operação ainda está incompleta**
- **Permite que cliente vá fazer outra coisa enquanto isso**
- **Permite cliente se registrar e ser notificado quando recurso liberar**

Utilização Sistemas Reativos

OS MAIORES SISTEMAS DO MUNDO SÃO REATIVOS

- **Atendem a bilhões de pessoas diariamente**
- **Sistemas grandes compostos por sistemas menores**
- **Dependem das propriedades reativas de seus constituintes**
- **Propriedades que se aplicam em todos os níveis de escala**
- **Propriedades que se tornam componíveis (*composable*)**