Arquitetura de software

Professor: Eduardo Cizeski Meneghel

PADRÕES E ESTILOS DE ARQUITETURAS

PADRÕES E ESTILOS

- Padrões e estilos de arquitetura de software são abordagens e diretrizes para projetar a estrutura de um sistema de software;

ESTILOS ABORDADOS ANTERIORMENTE

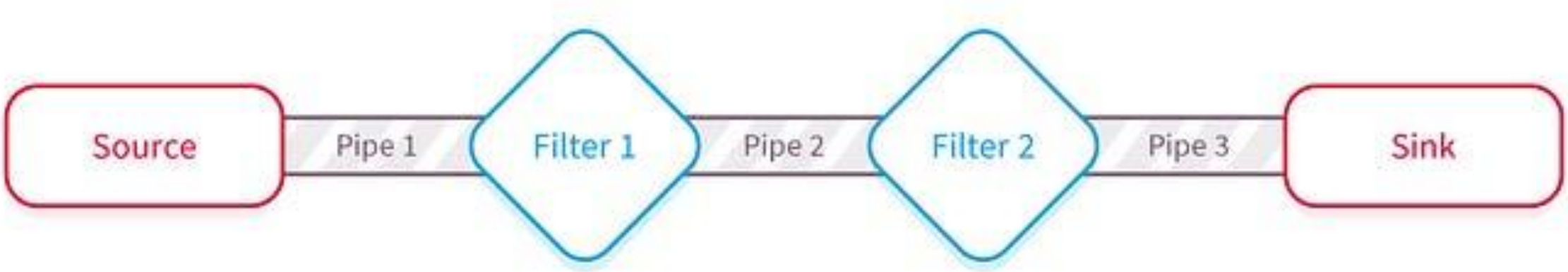
- Monolítica vs distribuída;
- Cliente-servidor;
- Arquitetura em camadas;
- Arquitetura orientada a serviços (SOA);
- Arquitetura baseada em microsserviços.

ESTILOS PIPES & FILTERS

PIPES & FILTER

- Sistemas projetados utilizando o estilo arquitetural pipes and filters fazem o tratamento de dados, obtidos através de algum mecanismo, movimentando-os através de unidades computacionais independentes (os filters), conectadas de maneira planejada (as pipes);

PIPES & FILTER



Pipe-filter pattern

PIPES & FILTER

- Imagine um sistema hidráulico responsável por fazer o tratamento de água imprópria para consumo. Internamente, a água é tratada sendo deslocada por meio de dutos e processada em “filtros” específicos, cada um com uma função singular determinada, como, por exemplo, remover impurezas, regular acidez, adicionar cloro e ajustar a temperatura.

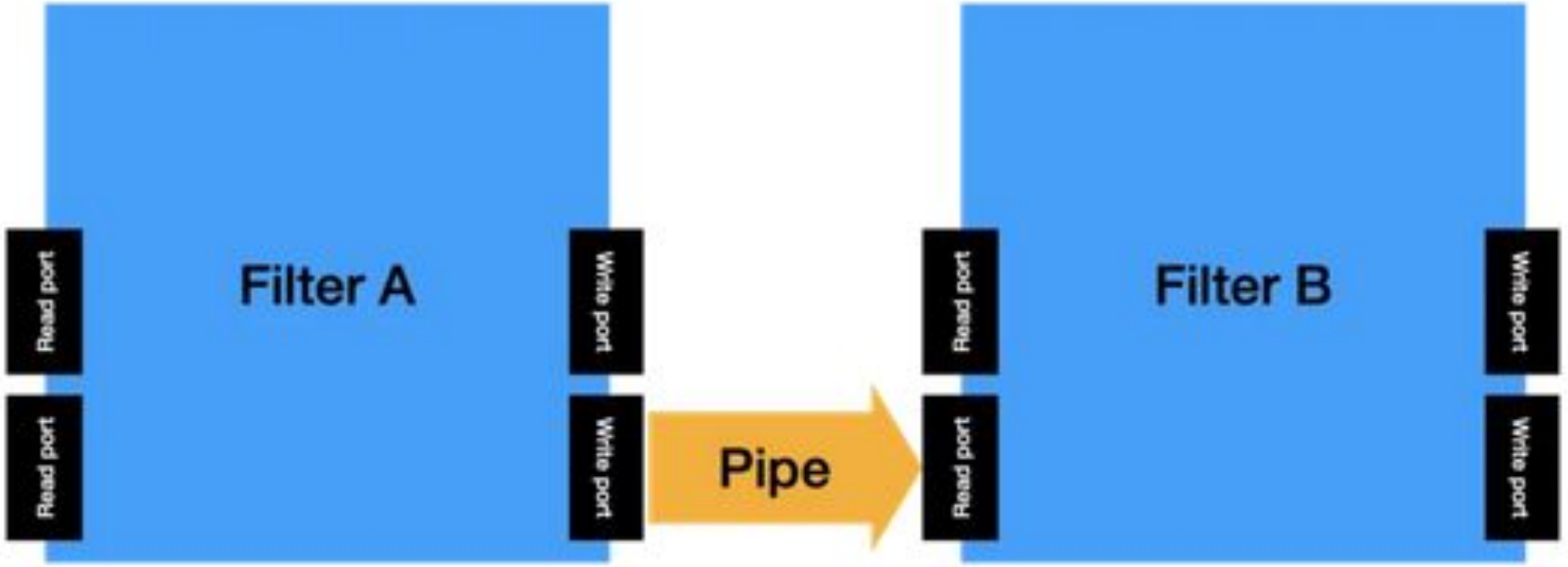
PIPES & FILTER

- Em sistemas implementados utilizando o estilo pipes and filters, a “água” é substituída por dados, os dutos são substituídos por algum mecanismo de comunicação e, finalmente, os filtros, são substituídos por unidades computacionais.

PIPES & FILTER

- **Filters**: Lêem dados a partir de uma ou mais read ports, executam algum tipo de processamento específico, e escrevem os resultados em uma ou mais write ports.
- **Pipes**: Conectam dois filters, recebendo input de uma write port e escrevendo dados em uma read port;
- **Read ports**: São pontos de entrada para filters, sendo responsáveis por converter dados em formatos diversos para uma representação interna compatível;
- **Write ports**: Que são pontos de saída dos filters, convertendo resultados do processamento para representações em formatos específicos.

PIPES & FILTER

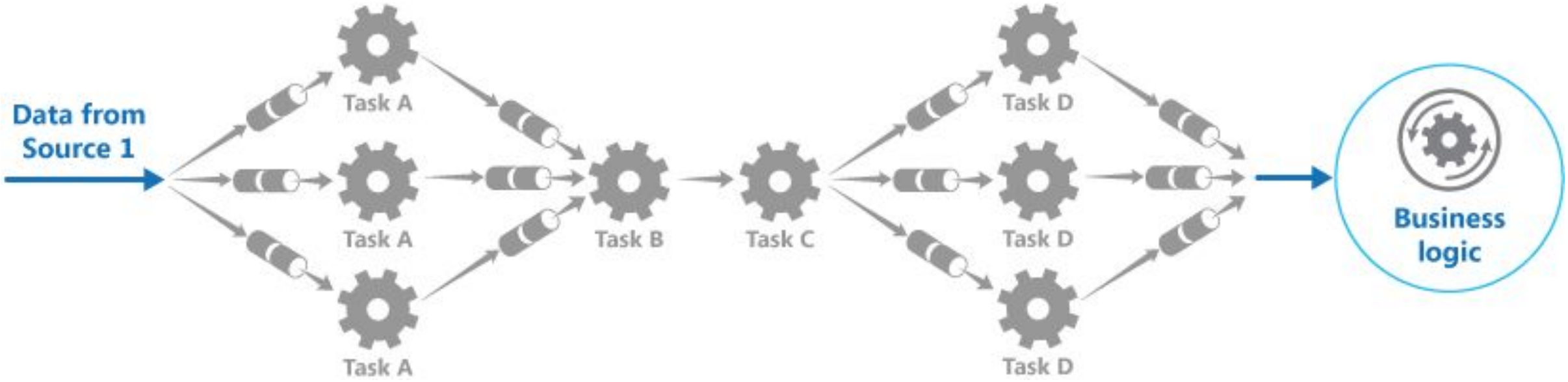


PIPES & FILTER

- **Flexibilidade (adição de novos filters e reordenação);**
- **Utilização de vários “sources”;**

PIPES & FILTER

- A lentidão em um filter afeta toda a cadeira;
- Existe a possibilidade de provisionar vários filters;



PIPES & FILTER

- É muito importante que a arquitetura defina uma estratégia comum para reportar **erros em produção**. Entretanto, a possibilidade de implantar mecanismos de recuperação pode ser difícil, ou até mesmo impossível, dependendo da natureza do problema.

ARQUITETURA ORIENTADA A EVENTOS (EDA)

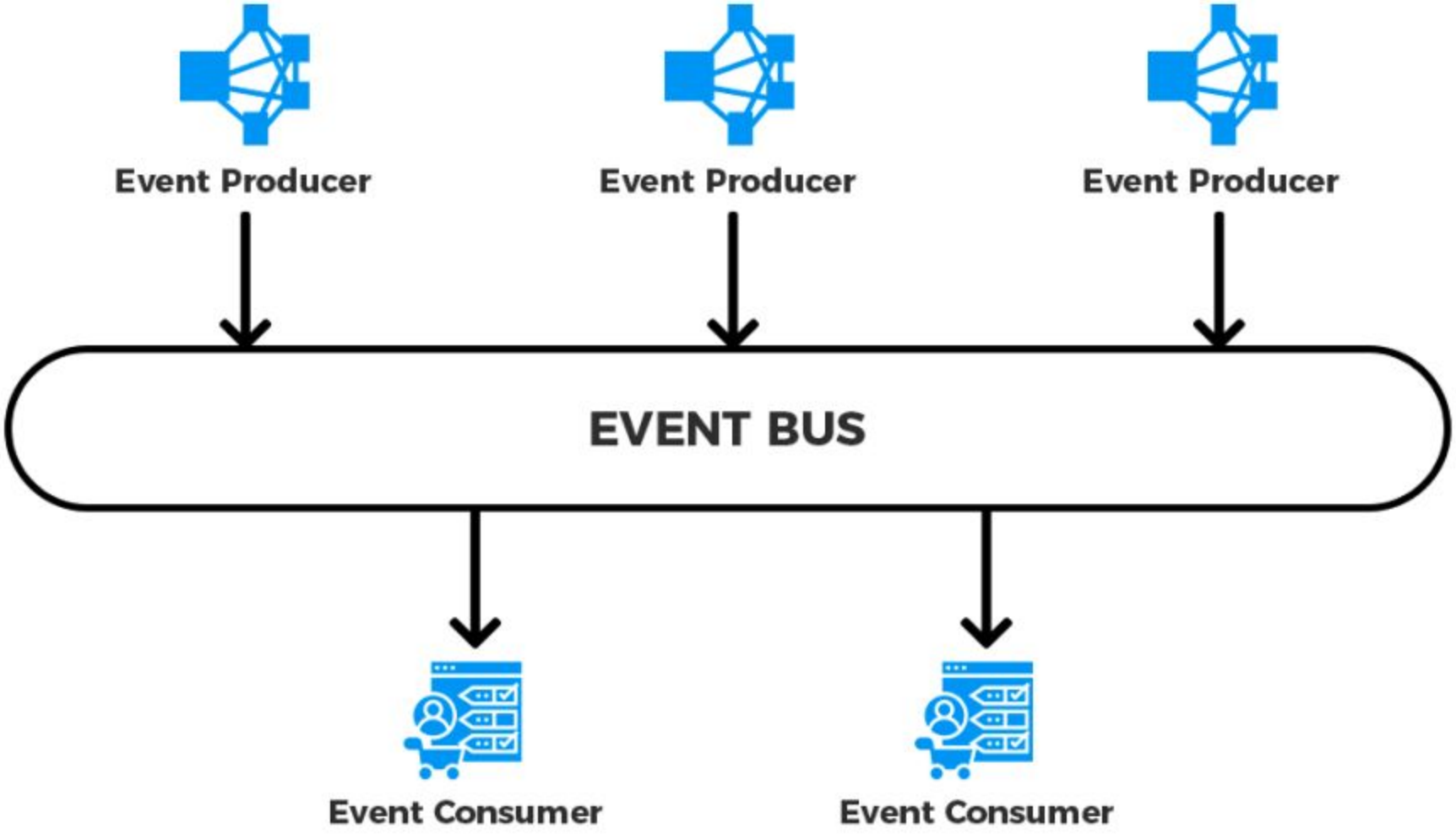
ARQUITETURA ORIENTADA A EVENTOS

- A arquitetura orientada a eventos (EDA) é um estilo que permite a uma organização detectar "eventos" ou momentos de negócios importantes e agir sobre eles em tempo real ou quase real;

ARQUITETURA ORIENTADA A EVENTOS

- Este padrão substitui a arquitetura tradicional de “solicitação / resposta”, em que os serviços teriam que esperar por uma resposta antes de poderem passar para a próxima tarefa. O fluxo da arquitetura orientada a eventos é executado por eventos e é projetado para responder a eles ou realizar alguma ação em resposta a um evento.

ARQUITETURA ORIENTADA A EVENTOS



ARQUITETURA ORIENTADA A EVENTOS

- O **evento** é algo relevante para o negócio da empresa, podendo ser: a ocorrência de uma venda, o cadastro de um novo cliente, a desistência de uma assinatura de revista, saques em locais geograficamente distantes do mesmo correntista, ou o valor acumulado de pedidos realizados dentro de um período.

ARQUITETURA ORIENTADA A EVENTOS

Produtores (Producers)

- Um produtor de eventos é um componente de software responsável por detectar eventos no ambiente. Considere o seguinte exemplo: Um leitor RFID identifica uma mudança em seu ambiente como um evento de negócio, criando um objeto de evento e colocando-o em uma mensagem de notificação, emitindo-a através de um canal. O produtor, neste exemplo, é um software distribuído junto ao leitor RFID.

ARQUITETURA ORIENTADA A EVENTOS

Canal (Channel)

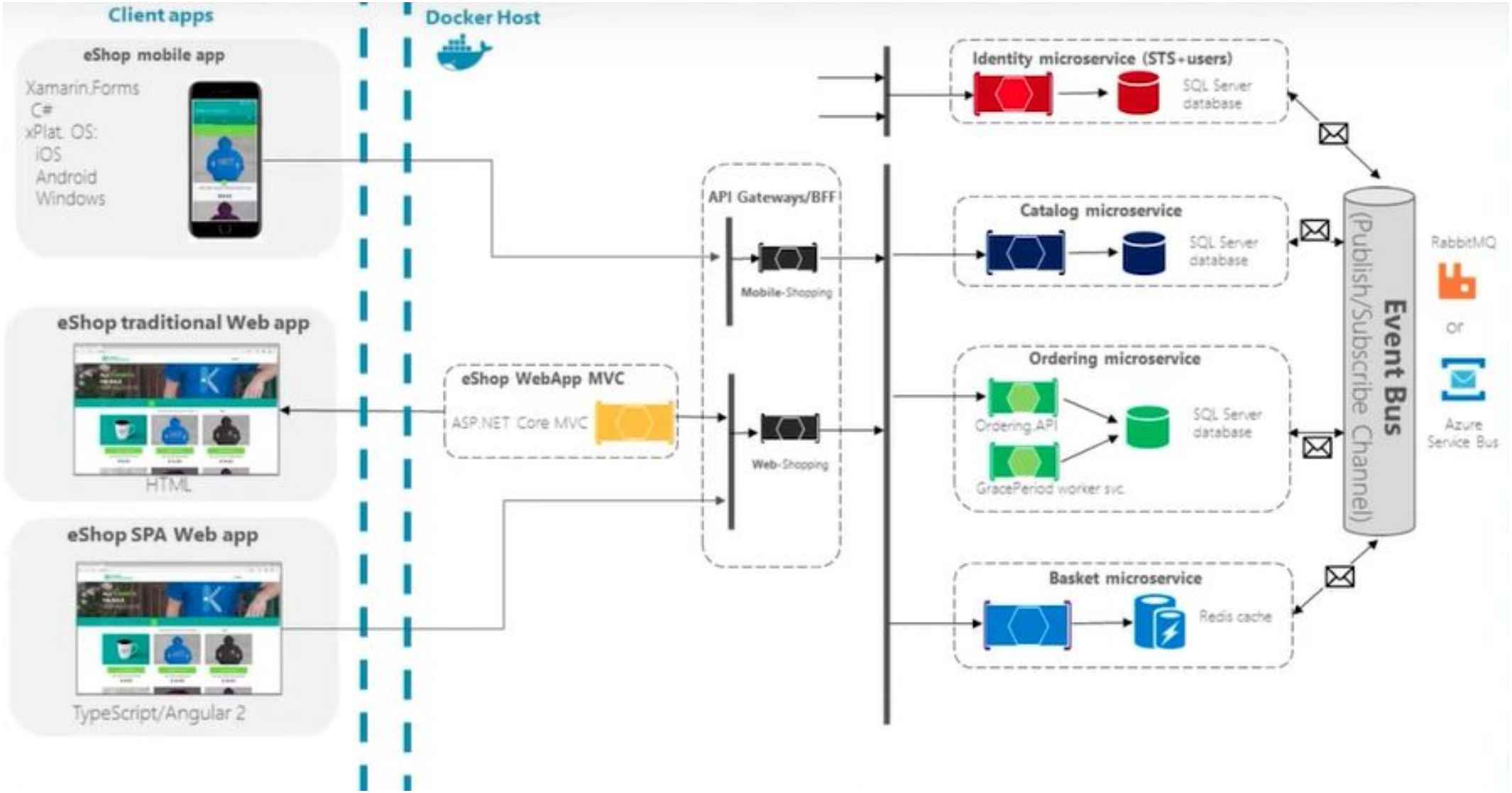
- Um canal pode ser entendido como um meio de transmissão de uma notificação de um componente da rede de processamento de eventos para o outro.
- Canais podem funcionar enviando mensagens, colocando objetos de eventos em um local compartilhado por produtores e consumidores ou ainda tendo o produtor chamando diretamente o consumidor.

ARQUITETURA ORIENTADA A EVENTOS

Consumidor (Consumer)

- Um consumidor é um agente da rede de processamento de eventos que recebe objetos de eventos.
- Dado o recebimento de um evento, um consumidor pode processar o evento localmente, pode acionar um serviço ou um processo de negócio, emitir uma mensagem, salvar o objeto para uso futuro ou ainda descartar o evento sem fazer nada com ele. Consumidores também são conhecidos como “event handlers”, “listeners” ou ainda “responders”.

ARQUITETURA ORIENTADA A EVENTOS



ARQUITETURA ORIENTADA A EVENTOS

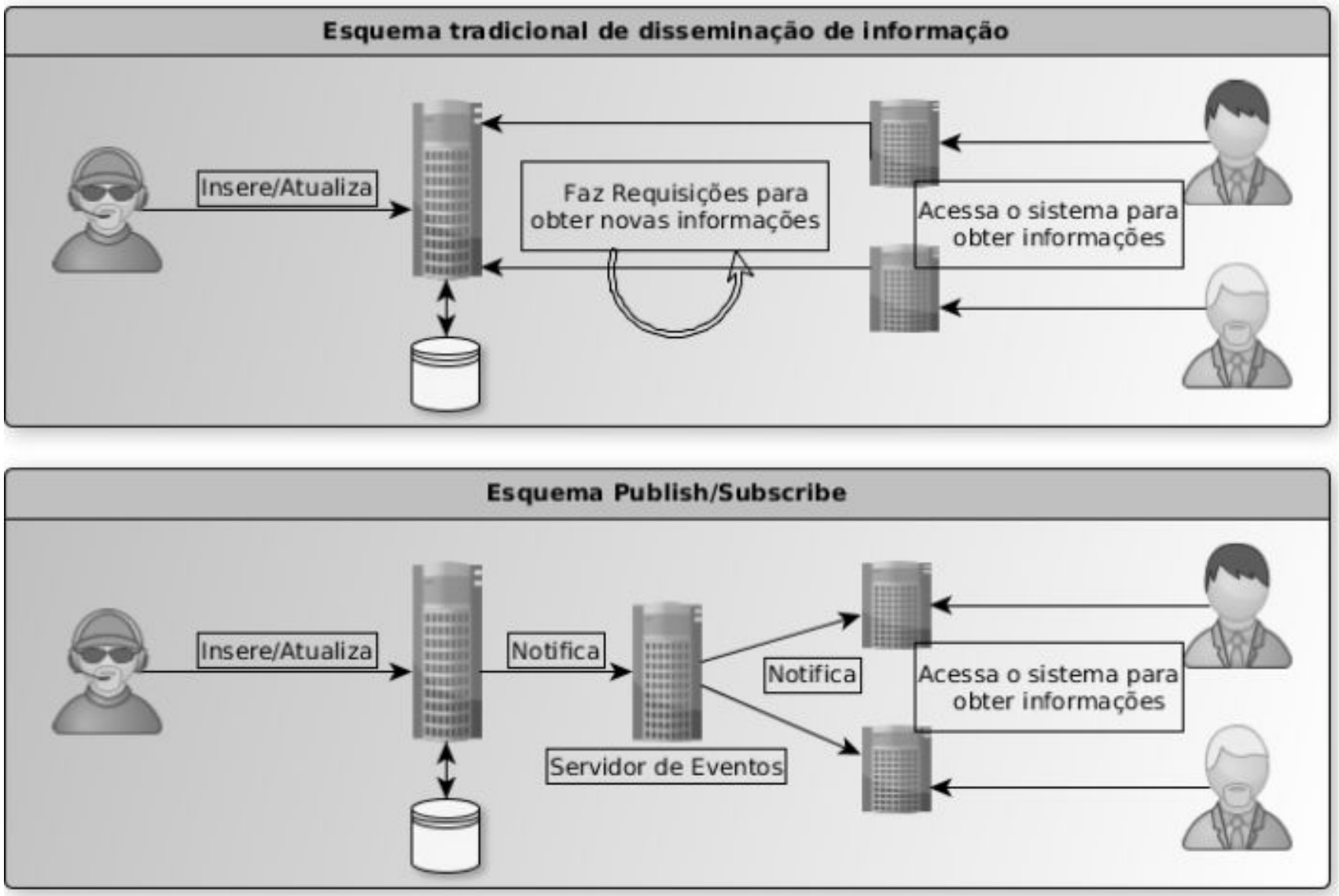
- LISTA DE EXERCÍCIOS 1

ARQUITETURA EM ESTILO PUBLICADOR-SUBSCRITOR

PUBLICADOR-SUBSCRITOR

- O Pub/Sub é que um padrão de troca de mensagens entre serviços. Com ele, é possível fazer com que serviços se comuniquem de forma assíncrona e independente.

PUBLICADOR-SUBSCRITOR



PUBLICADOR-SUBSCRITOR

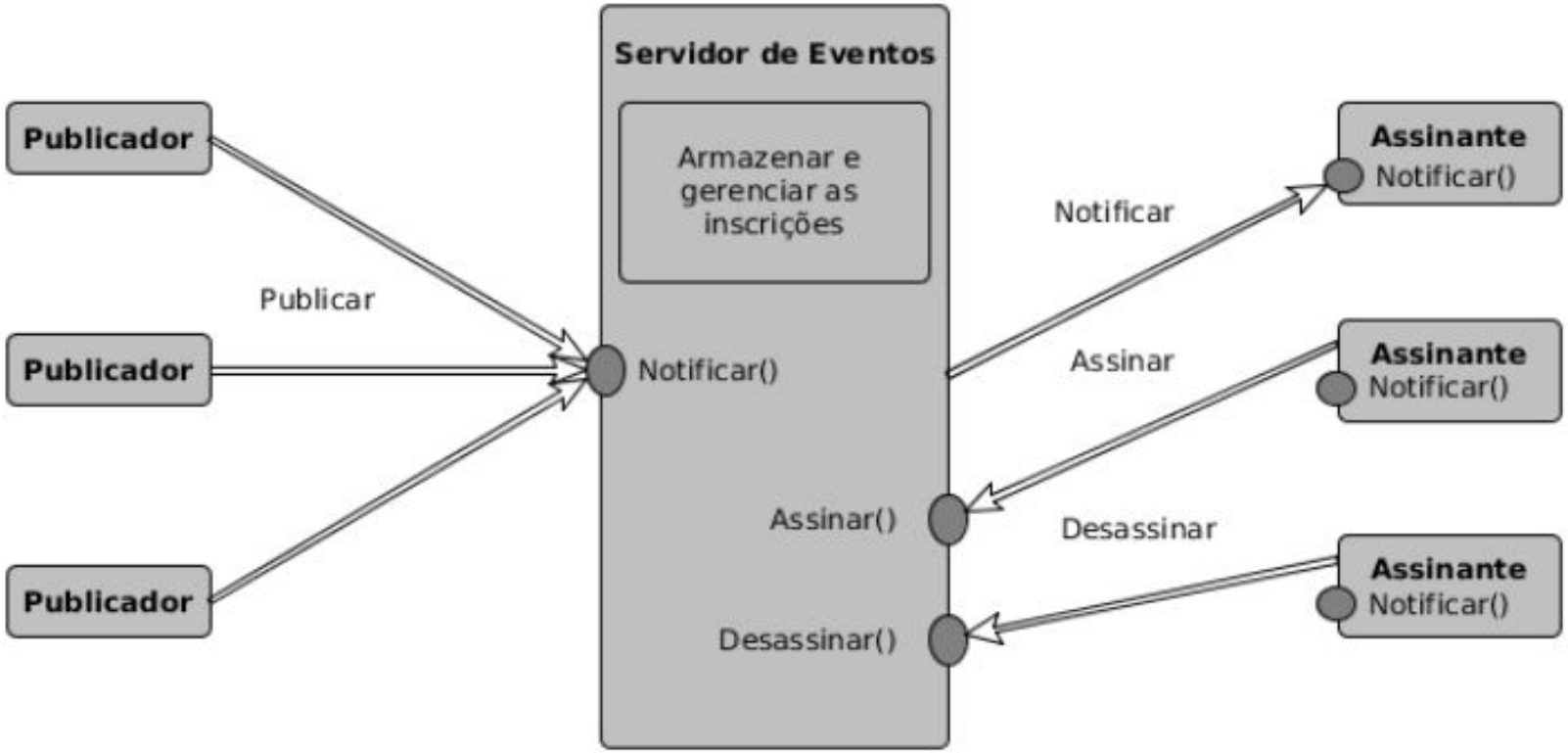


Figura 2.3: Interações do padrão *Pub/Sub* (adaptado de Eugster *et al.* [7])

PUBLICADOR-SUBSCRITOR

Benefícios:

- Ciclo de vida independente: A responsabilidade do Publisher termina quando entregar as mensagens no Channel. Este por sua vez ao começar a entregar as mensagens aos Subscribers, será uma cópia da mensagem para cada um, iniciando também terá seu próprio ciclo de vida.

PUBLICADOR-SUBSCRITOR

Benefícios:

- Assíncrono: Quando você delega o processamento a outro software, ao enviar essa mensagem via pub/sub você não sabe quando vai receber a resposta. Isso também facilita para não ficar prendendo processamento aguardando uma resposta que foi feita a outro servidor.

PUBLICADOR-SUBSCRITOR

Benefícios:

- Escalabilidade: Através de ferramentas gráficas oferecidos por cada fornecedor de Cloud, é possível acompanhar o volume de mensagens, identificar pontos de gargalo, necessidades de escalar ou não os consumidores.

PUBLICADOR-SUBSCRITOR

Benefícios:

- Armazenamento de Mensagens: Caso o consumidor esteja sobrecarregado, o channel se encarrega de enviar as mensagens conforme fluxo definido. Haverá uma dosagem controlada, evitando assim que o software não fique offline devido a sobrecarga. Caso o consumidor seja atualizado, haverá um tempo mínimo de indisponibilidade, isso não será um problema pois o channel aguardará que o consumidor fique online.

PUBLICADOR-SUBSCRITOR

- Publicador - Subscritor X Fila

PUBLICADOR-SUBSCRITOR

- Exemplo TryRabbitMq.

ARQUITETURA MVC
MODEL, O CONTROLLER E A VIEW

ARQUITETURA MVC

- O MVC é utilizado em muitos projetos devido a arquitetura que possui, o que possibilita a divisão do projeto em camadas muito bem definidas. Cada uma delas, o Model, o Controller e a View, executa o que lhe é definido e nada mais do que isso.

ARQUITETURA MVC

- O MVC é utilizado em muitos projetos devido a arquitetura que possui, o que possibilita a divisão do projeto em camadas muito bem definidas. Cada uma delas, o Model, o Controller e a View, executa o que lhe é definido e nada mais do que isso.

ARQUITETURA MVC

MODEL

- Essa classe também é conhecida como Business Object Model (objeto modelo de negócio). Sua responsabilidade é gerenciar e controlar a forma como os dados se comportam por meio das funções, lógica e regras de negócios estabelecidas;
- Ele é o detentor dos dados que recebe as informações do Controller, válida se ela está correta ou não e envia a resposta mais adequada.

ARQUITETURA MVC

CONTROLLER

- A camada de controle é responsável por intermediar as requisições enviadas pelo View com as respostas fornecidas pelo Model, processando os dados que o usuário informou e repassando para outras camadas;
- Numa analogia bem simplista, o controller operaria como o ‘’maestro de uma orquestra’’ que permite a comunicação entre o detentor dos dados e a pessoa com vários questionamentos no MVC.

ARQUITETURA MVC

VIEW

- Essa camada é responsável por apresentar as informações de forma visual ao usuário. Em seu desenvolvimento devem ser aplicados apenas recursos ligados a aparência como mensagens, botões ou telas.

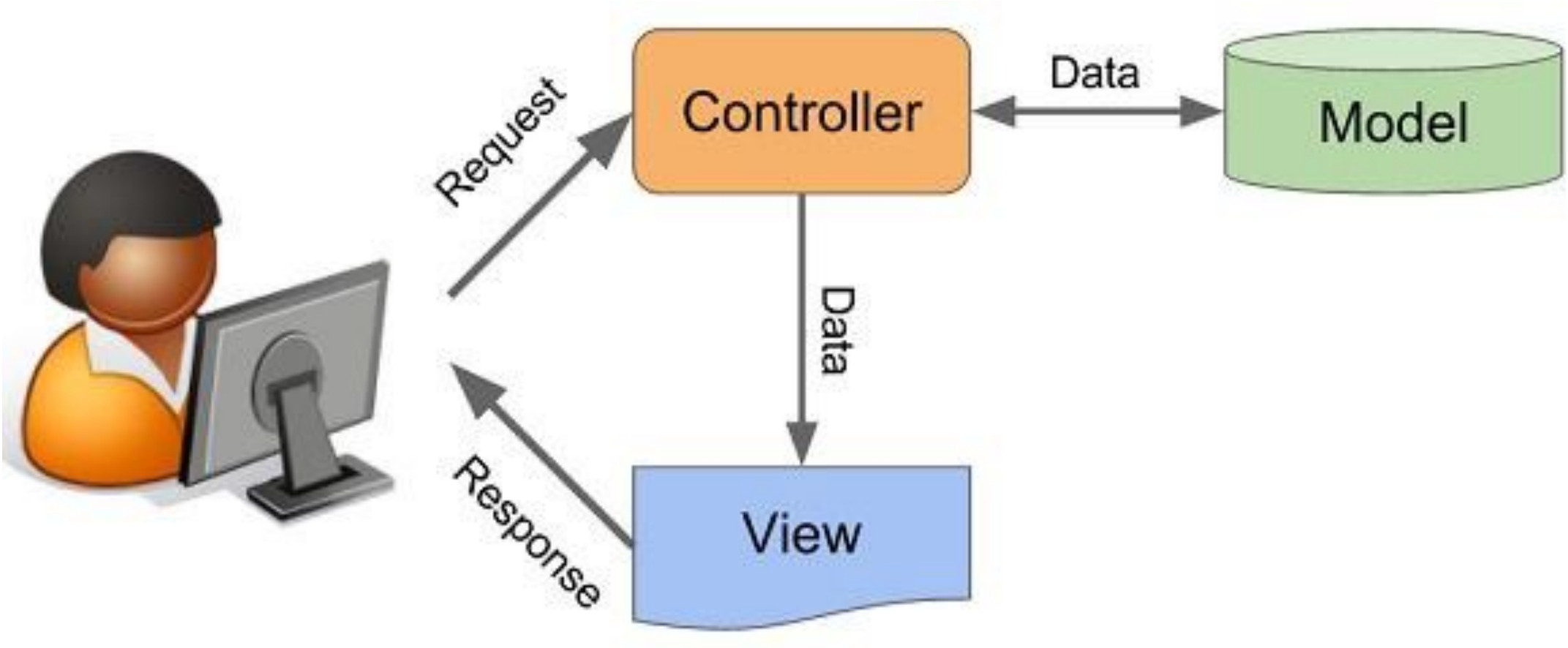
- Seguindo nosso processo de comparação o View está na linha de frente da comunicação com usuário e é responsável transmitir questionamentos ao controller e entregar as respostas obtidas ao usuário. É a parte da interface que se comunica, disponibilizando e capturando todas as informação do usuário.

ARQUITETURA MVC

- Tudo começa com a interação do usuário na camada View. A partir daí o controlador pega essa informações e envia para o Model que fica responsável por avaliar aqueles dados e transmitir uma resposta.

- O controlador recebe essas respostas e envia uma notificação de validação daquela informação para a camada visão, fazendo com a mesma apresente o resultado de maneira gráfica e visual.

ARQUITETURA MVC



ARQUITETURA MVC

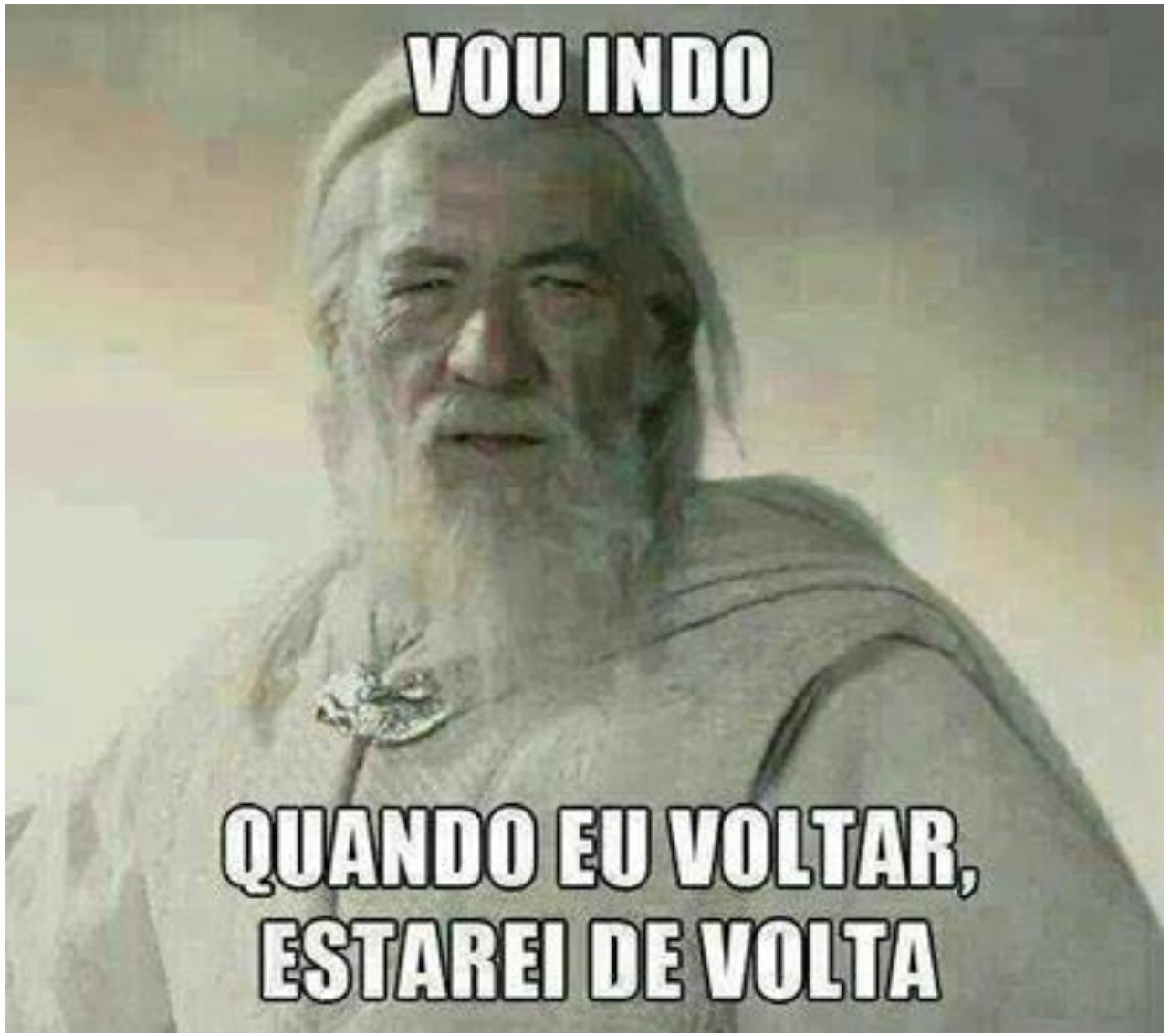
- **Segurança**: O controller funciona como uma espécie de filtro capaz de impedir que qualquer dado incorreto chegue até a camada modelo.
- **Organização**: Esse método de programação permite que um novo desenvolvedor tenha muito mais facilidade em entender o que foi construído, assim como os erros se tornam mais fácil de serem encontrados e corrigidos.

ARQUITETURA MVC

- **Eficiência**: Como a arquitetura de software é dividida em 3 componentes, sua aplicação fica muito mais leve, permitindo que vários desenvolvedores trabalhem no projeto de forma independente.
- **Tempo**: Com a dinâmica facilitada pela colaboração entre os profissionais de desenvolvimento, o projeto pode ser concluído com muito mais rapidez, tornando o projeto escalável.
- **Transformação**: As mudanças que forem necessárias também são mais fluidas, já que não será essencial trabalhar nas regras de negócio e correção de bugs.

ARQUITETURA MVC

- LISTA DE EXERCÍCIOS 2


VOU INDO
QUANDO EU VOLTAR,
ESTAREI DE VOLTA