ACH2147 - Desenvolvimento de Sistemas de Informação Distribuídos

Aula 05 – Arquitetura de Sistemas Distribuídos

Norton Trevisan Roman

4 de abril de 2022

Arquiteturas de sistemas distribuídos

Arquitetura de Software

- Organização lógica de um sistema distribuído em componentes de software
  - Nos diz como os vários componentes de software devem ser organizados e como devem interagir
- Projeto com a solução para um determinado problema, dado um conjunto de restrições

Arquiteturas de sistemas distribuídos

Estilos Arquiteturais

- Fornece os princípios gerais que guiam a criação e organização de uma arquitetura
  - Idéia básica: organize o sistema em componentes logicamente diferentes e os distribua entre as máquinas disponíveis
- Estilos arquiteturais informam e guiam a criação de arquiteturas

Arquiteturas de sistemas distribuídos

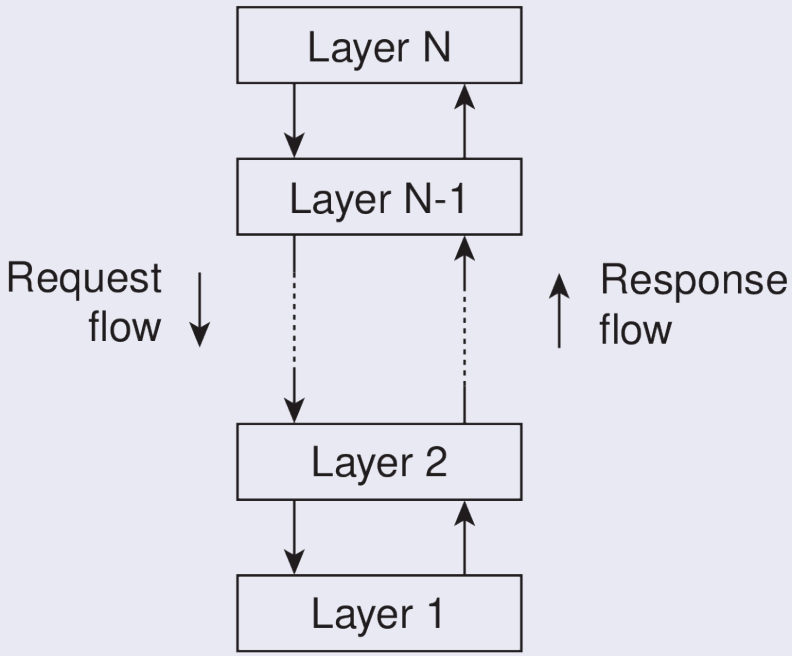
Estilos Arquiteturais

- Fornece os princípios gerais que guiam a criação e organização de uma arquitetura
  - Idéia básica: organize o sistema em componentes logicamente diferentes e os distribua entre as máquinas disponíveis
- Estilos arquiteturais informam e guiam a criação de arquiteturas

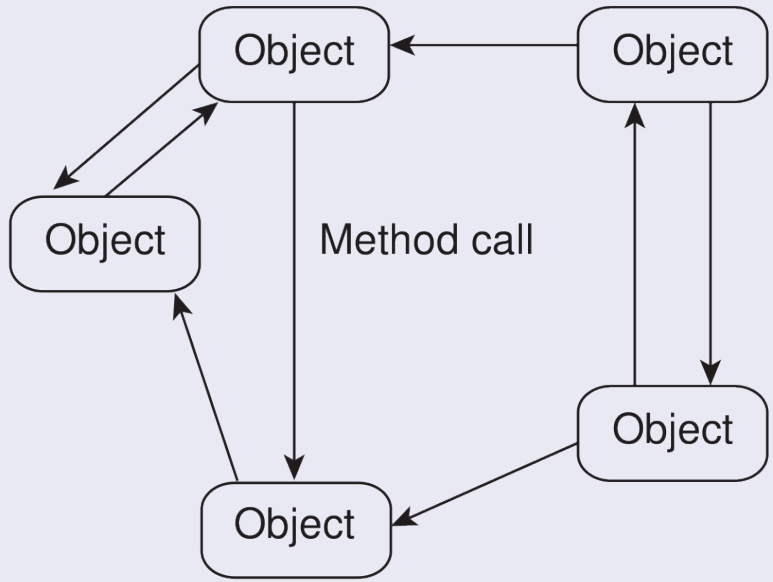
O **Louvre** é um *projeto* arquitetônico cujo *estilo arquitetural* é o **Barroco**

Arquiteturas de sistemas distribuídos

Estilos Arquiteturais: Exemplos



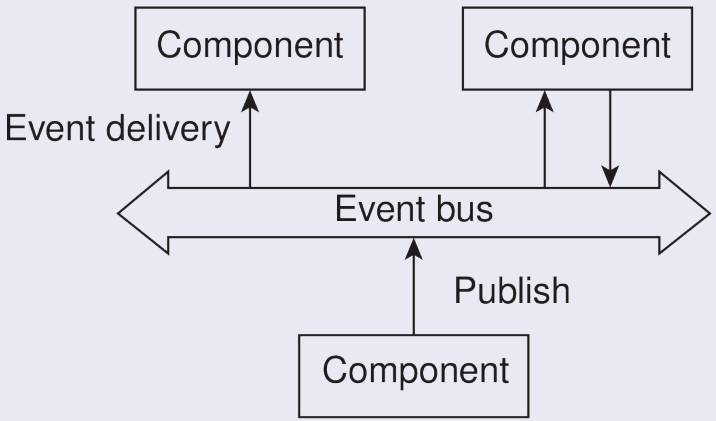
Multicamadas
(sistemas cliente-servidor)



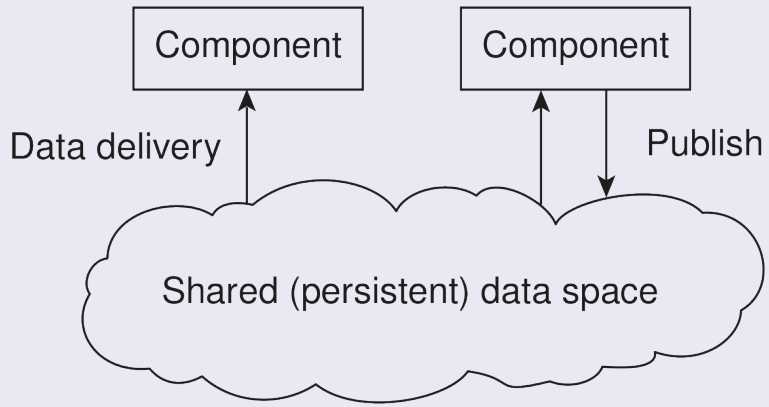
Orientada a objetos
(sistemas de objetos distribuídos)

Estilos Arquiteturais

Desacoplar processos no espaço (anônimos) e tempo (assíncronos) pode levar a estilos diferentes



Publish/subscribe
(desaclopado no espaço)



Espaço de dados compartilhados
(desacoplado no espaço e tempo)

Arquiteturas de sistemas distribuídos

Estilos Arquiteturais

- Um estilo é definido em termos de:
  - Componentes (substituíveis) com interfaces bem definidas
  - O modo como os componentes são conectados entre si
  - Como os dados são trocados entre componentes
  - Como esses componentes e conectores são configurados conjuntamente em um sistema

Arquiteturas de sistemas distribuídos

Conector?

- Mecanismo que intermedeia comunicação, coordenação ou cooperação entre componentes
  - Permite o fluxo de controle e dados entre os componentes
  - Ex: recursos para chamadas de procedimento (remotos), mensagens ou *streaming*

Estilos Arquiteturais

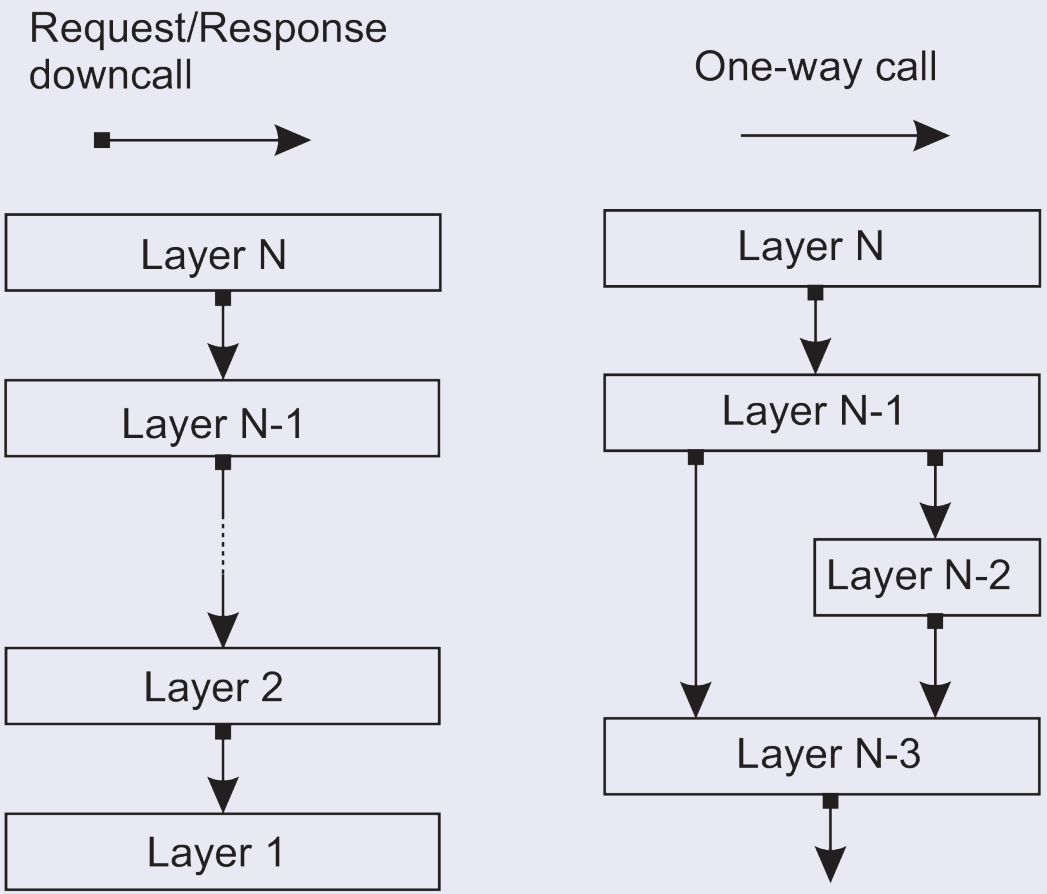
- Estilo Multicamadas
- Estilo Baseado em Objetos
- Estilo Baseado em Recursos
- Estilo Publish-Subscribe

Estilos Arquiteturais

- **Estilo Multicamadas**
- Estilo Baseado em Objetos
- Estilo Baseado em Recursos
- Estilo Publish-Subscribe

Estilo Multicamadas

Diferentes organizações



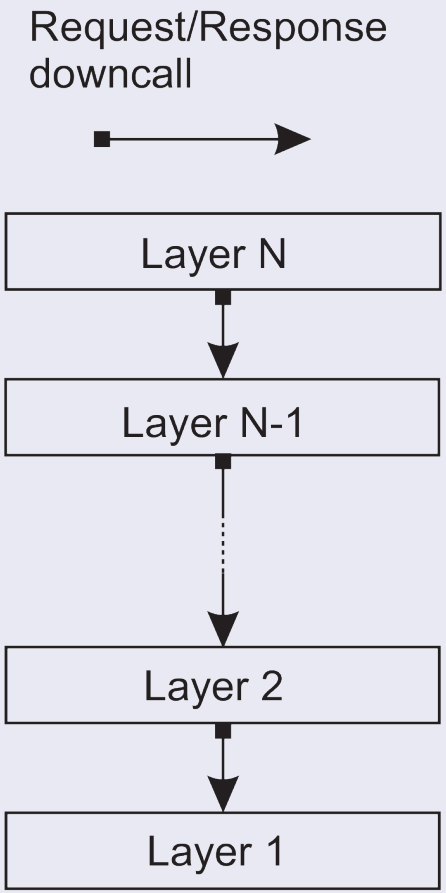
Componentes são organizados em camadas

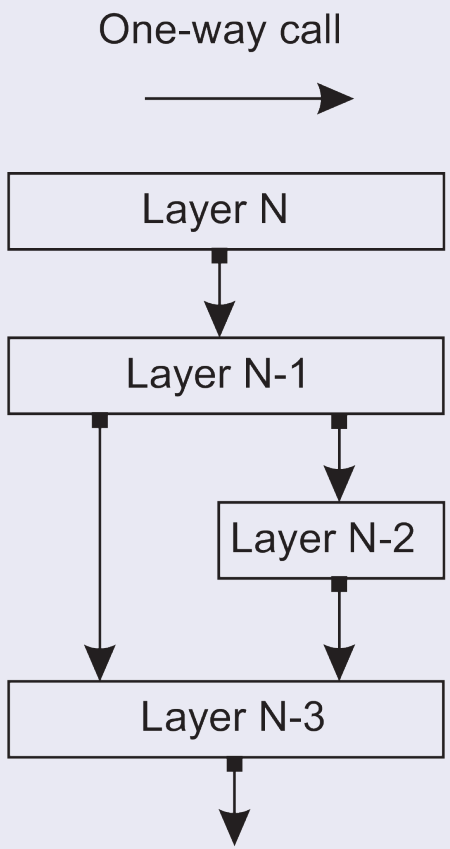
Componentes em uma camada podem chamar outros em camadas inferiores (*downcall*)

Podendo ou não esperar uma resposta

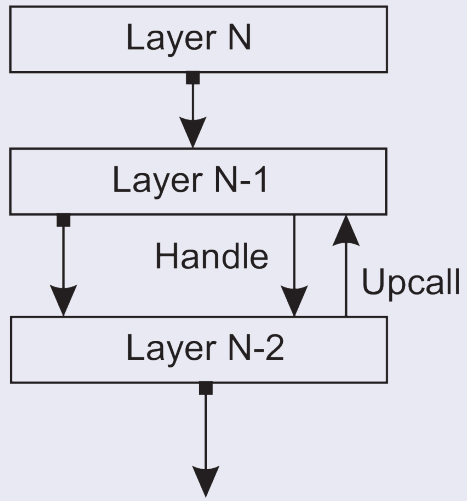
Estilo Multicamadas

Diferentes organizações



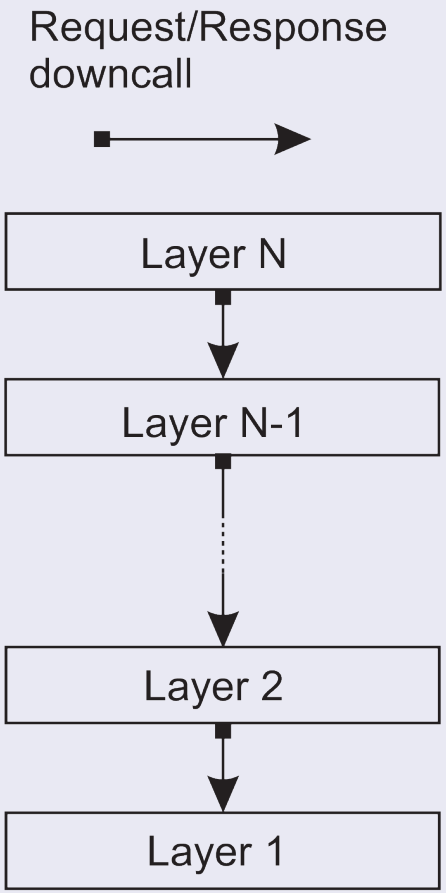


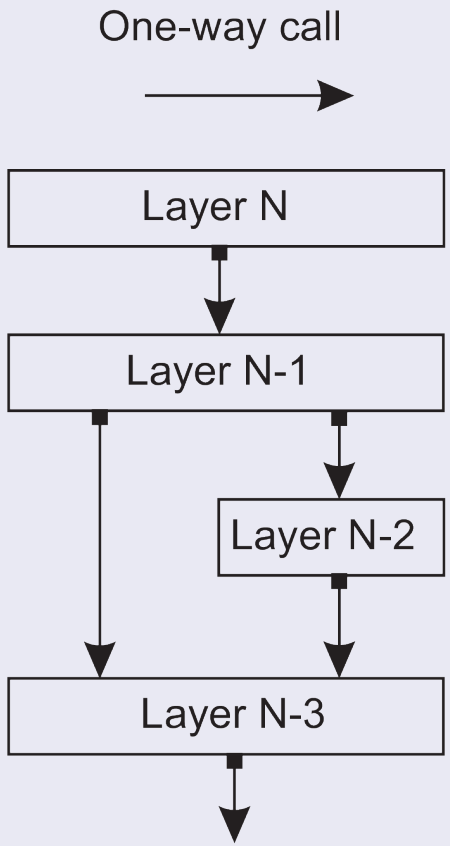
Excepcionalmente, componentes em camadas superiores podem ser chamados (*upcall*)



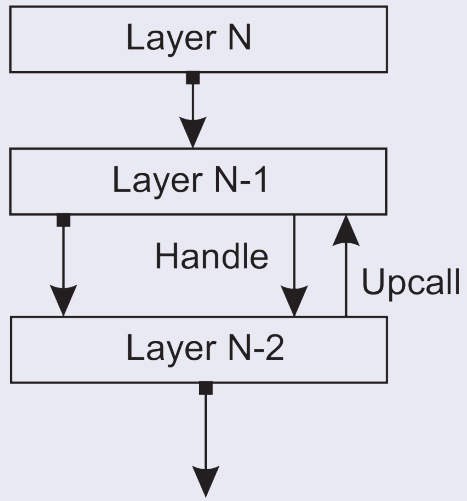
Estilo Multicamadas

Diferentes organizações



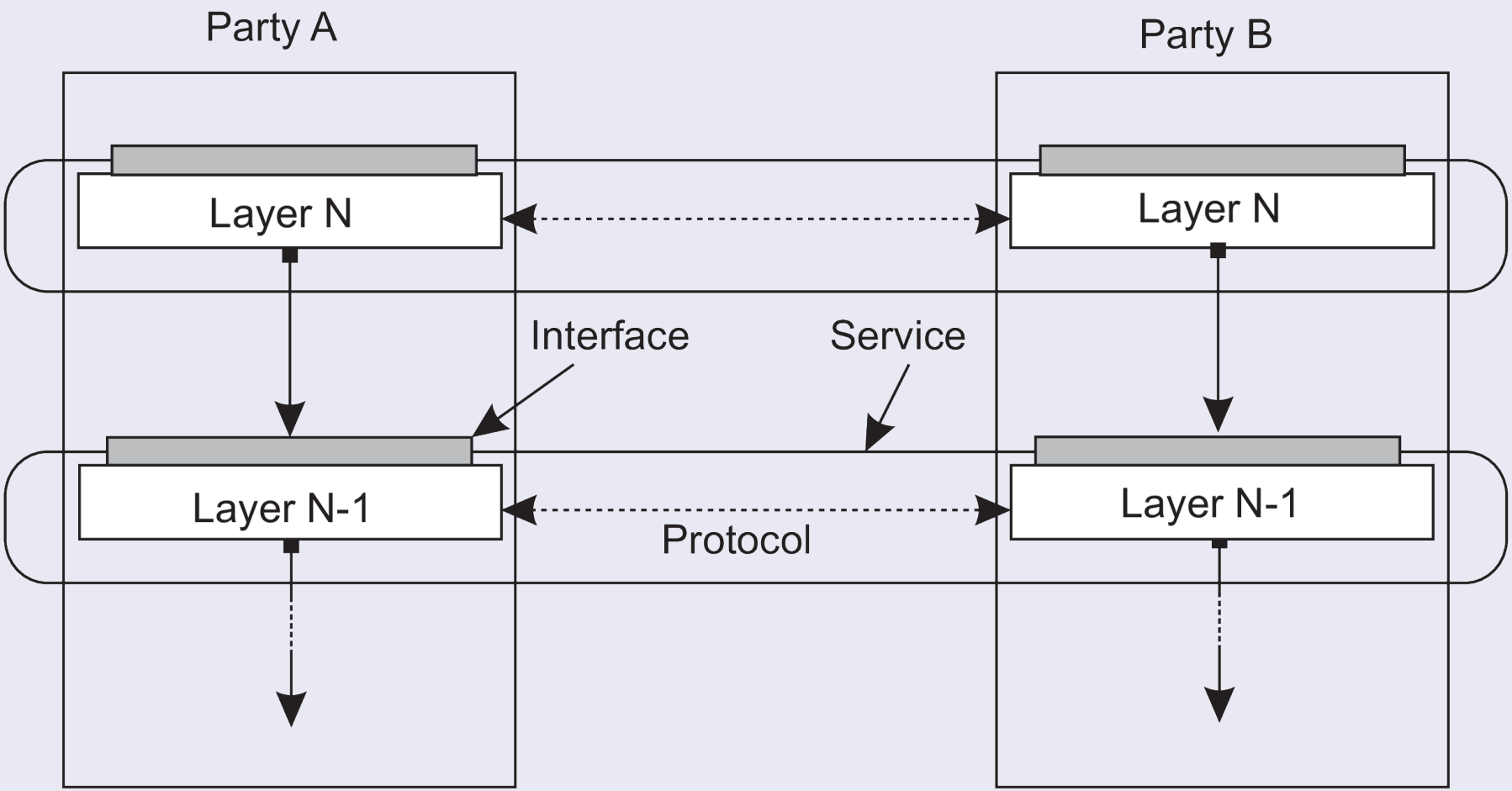


Como quando o SO avisa a aplicação de um evento, chamando seu *handler*



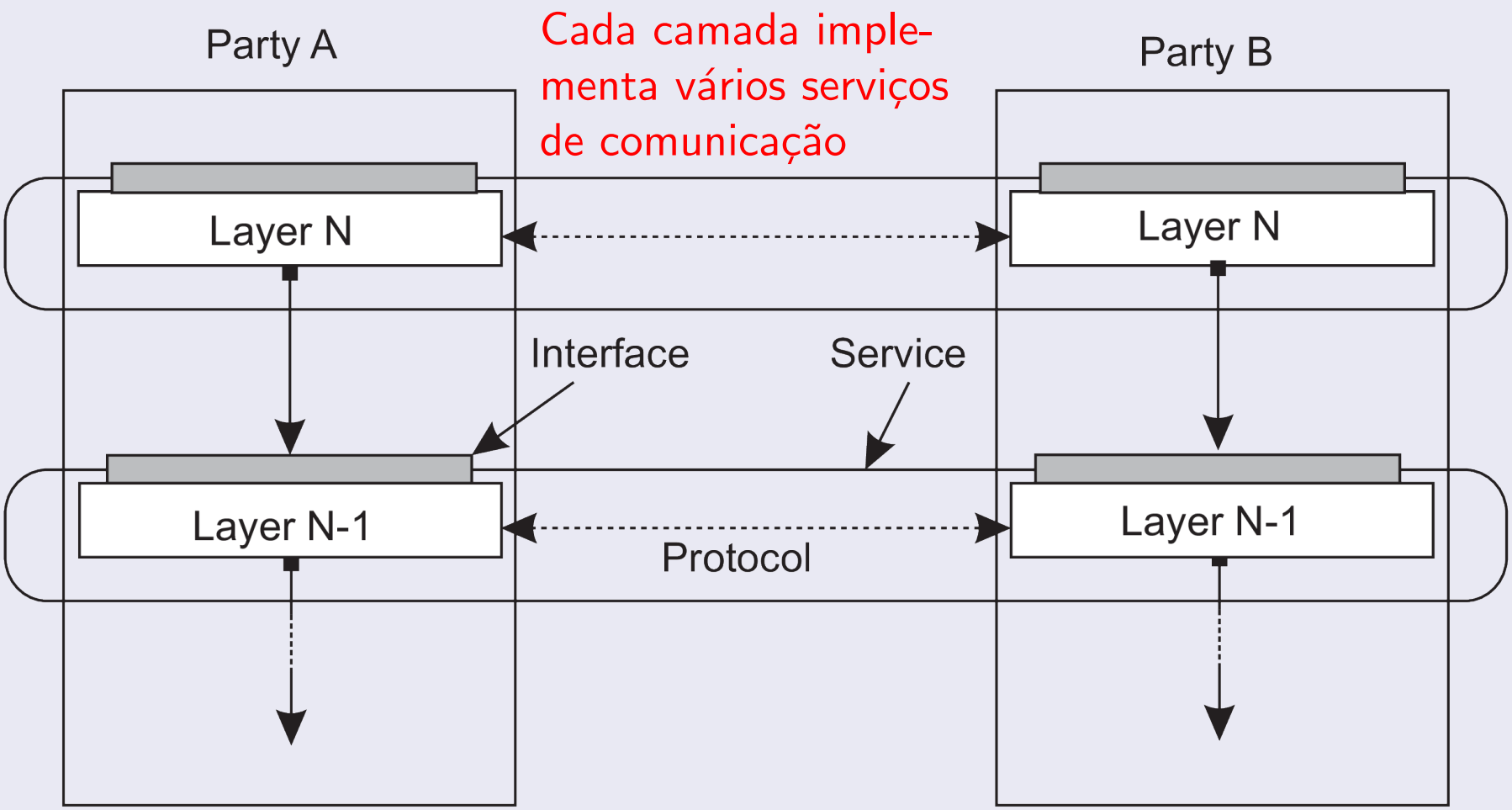
Estilo Multicamadas

Ex: Pilhas de protocolos de comunicação



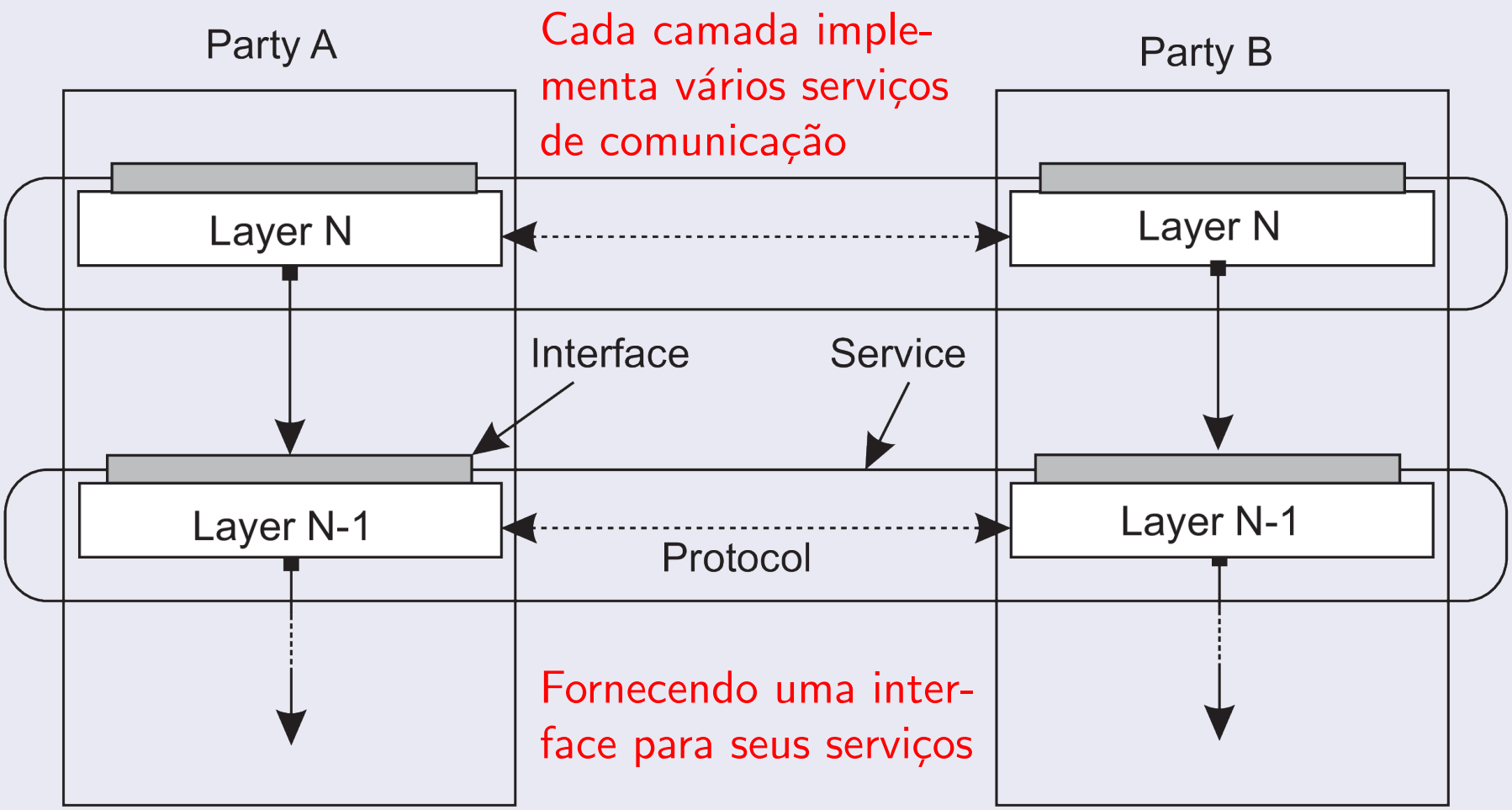
Estilo Multicamadas

Ex: Pilhas de protocolos de comunicação



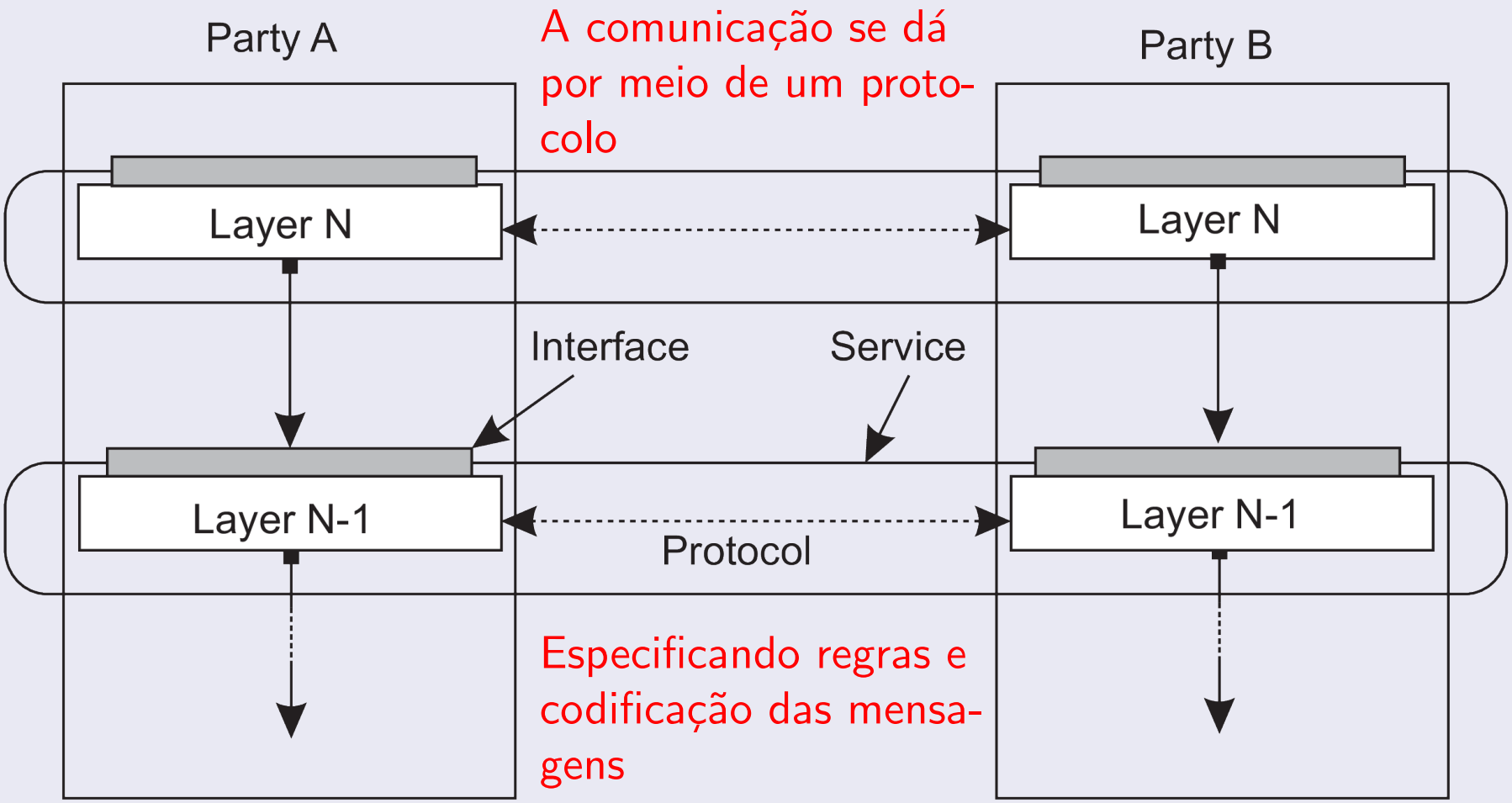
Estilo Multicamadas

Ex: Pilhas de protocolos de comunicação



Estilo Multicamadas

Ex: Pilhas de protocolos de comunicação



Serviço × Interface × Protocolo

Servidor

```
from socket import *
s = socket(AF_INET, SOCK_STREAM)
s.bind((HOST, PORT))
s.listen()
(conn, addr) = s.accept()  # returns new socket and addr. client
while True:                # forever
  data = conn.recv(1024)   # receive data from client
  if not data: break       # stop if client stopped
  conn.send(str(data)+"*") # return sent data plus an "*"
conn.close()               # close the connection
```

Serviço orientado à conexão

Serviço × Interface × Protocolo

Servidor

```
from socket import *
s = socket(AF_INET, SOCK_STREAM)
s.bind((HOST, PORT))
s.listen()
(conn, addr) = s.accept()  # returns new socket and addr. client
while True:                # forever
  data = conn.recv(1024)   # receive data from client
  if not data: break       # stop if client stopped
  conn.send(str(data)+"*") # return sent data plus an "*"
conn.close()               # close the connection
```

Funções da interface

Serviço × Interface × Protocolo

Servidor

```
from socket import *                    Protocolo de Comunicação
s = socket(AF_INET, SOCK_STREAM)        (TCP)
s.bind((HOST, PORT))
s.listen()
(conn, addr) = s.accept()  # returns new socket and addr. client
while True:                # forever
  data = conn.recv(1024)   # receive data from client
  if not data: break       # stop if client stopped
  conn.send(str(data)+"*") # return sent data plus an "*"
conn.close()               # close the connection
```

Serviço × Interface × Protocolo

Cliente

```
from socket import *
s = socket(AF_INET, SOCK_STREAM)
s.connect((HOST, PORT)) # connect to server (blocking)
s.send('Hello, world')  # send some data
data = s.recv(1024)     # receive the response
print data              # print the result
s.close()               # close the connection
```

Veja um exemplo completo em
https://docs.python.org/howto/sockets.html.

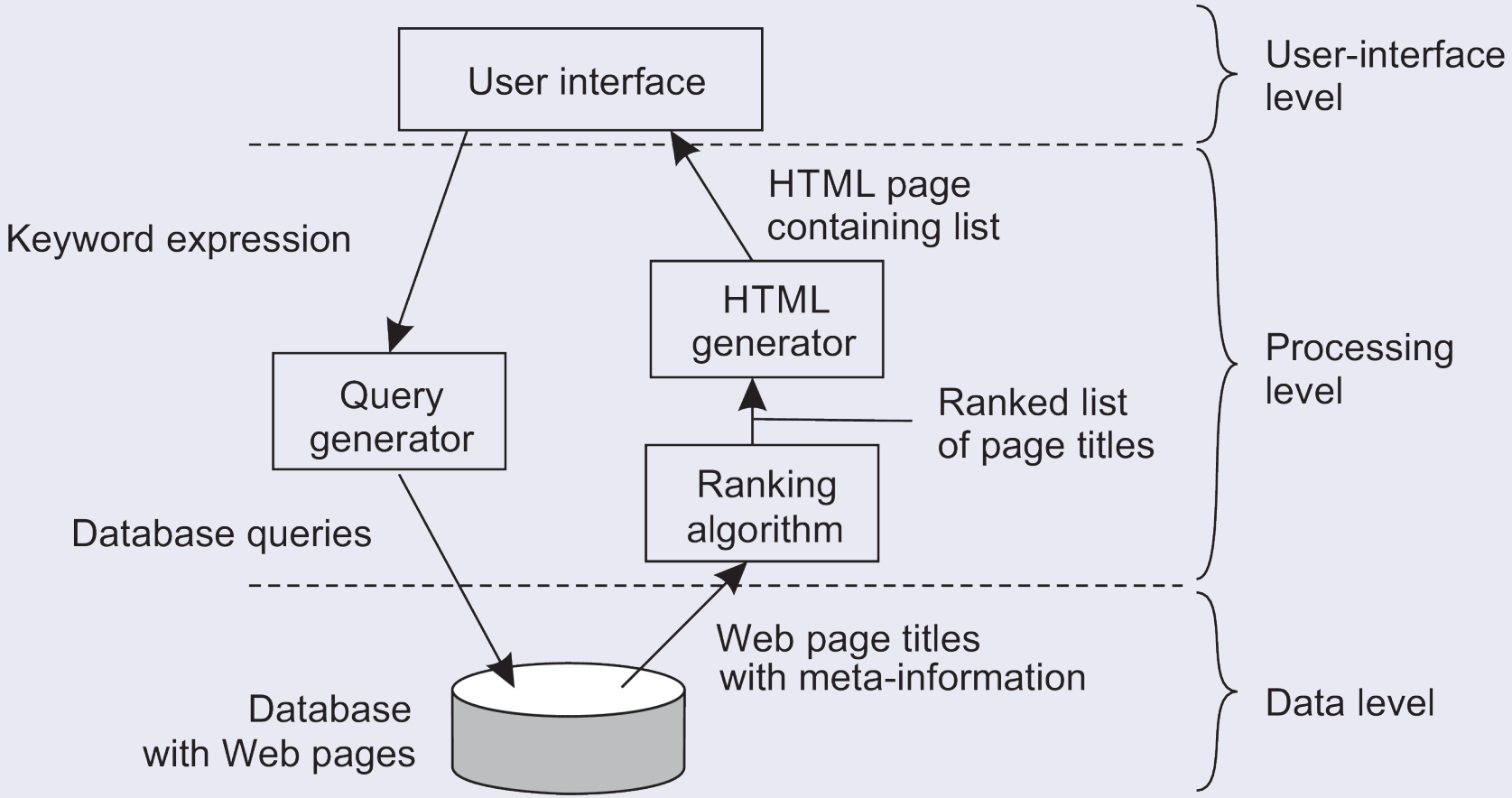
Estilo Multicamadas

Visão tradicional em três camadas

- A camada de apresentação contém o necessário para a aplicação poder interagir com o usuário ou aplicações externas
- A camada de negócio (ou processamento) contém as funções de uma aplicação
- A camada de dados contém os dados que o cliente quer manipular através dos componentes da aplicação

Estilo Multicamadas

Ex: Um *search engine* simples



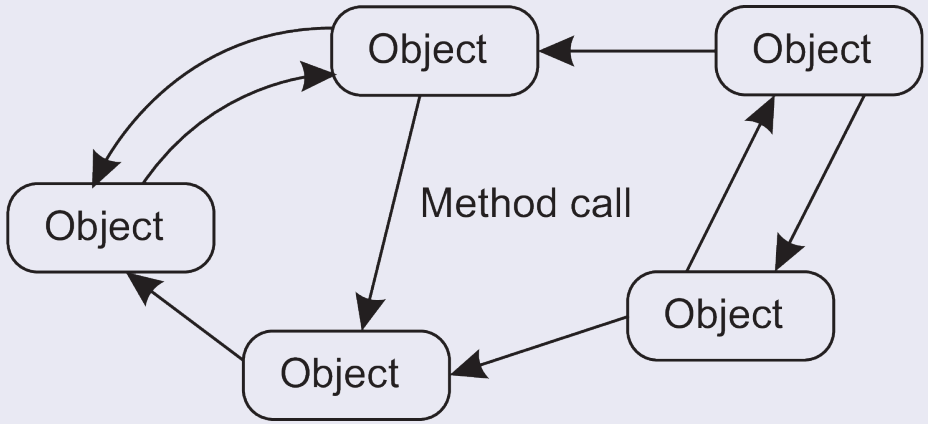
Estilos Arquiteturais

- Estilo Multicamadas
- **Estilo Baseado em Objetos**
- Estilo Baseado em Recursos
- Estilo Publish-Subscribe

Estilo Baseado em Objetos

Essência

- Componentes são objetos
  - Conectados entre si por meio de chamadas a procedimentos

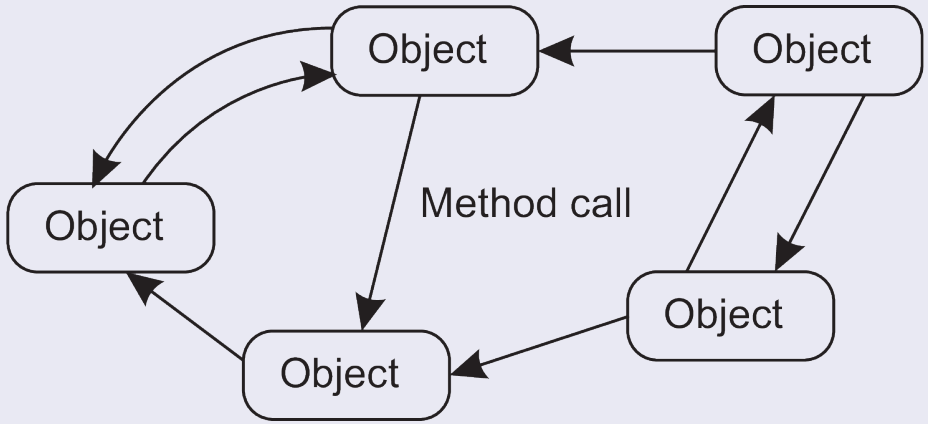


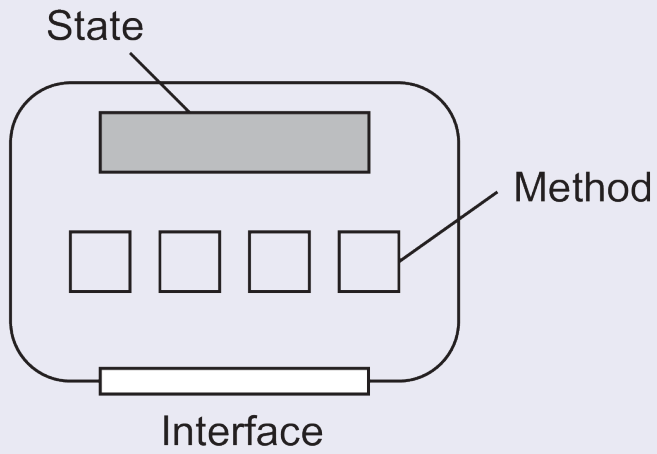
- Objetos podem ser colocados em máquinas diferentes
  - Chamadas a eles são então executadas via rede

Estilo Baseado em Objetos

Encapsulamento

- Fornecem um modo natural de encapsular dados
  - Chamados de **estado** do objeto
- E de encapsular as operações executadas sobre esses dados
  - Métodos do objeto, acessados por meio de uma interface

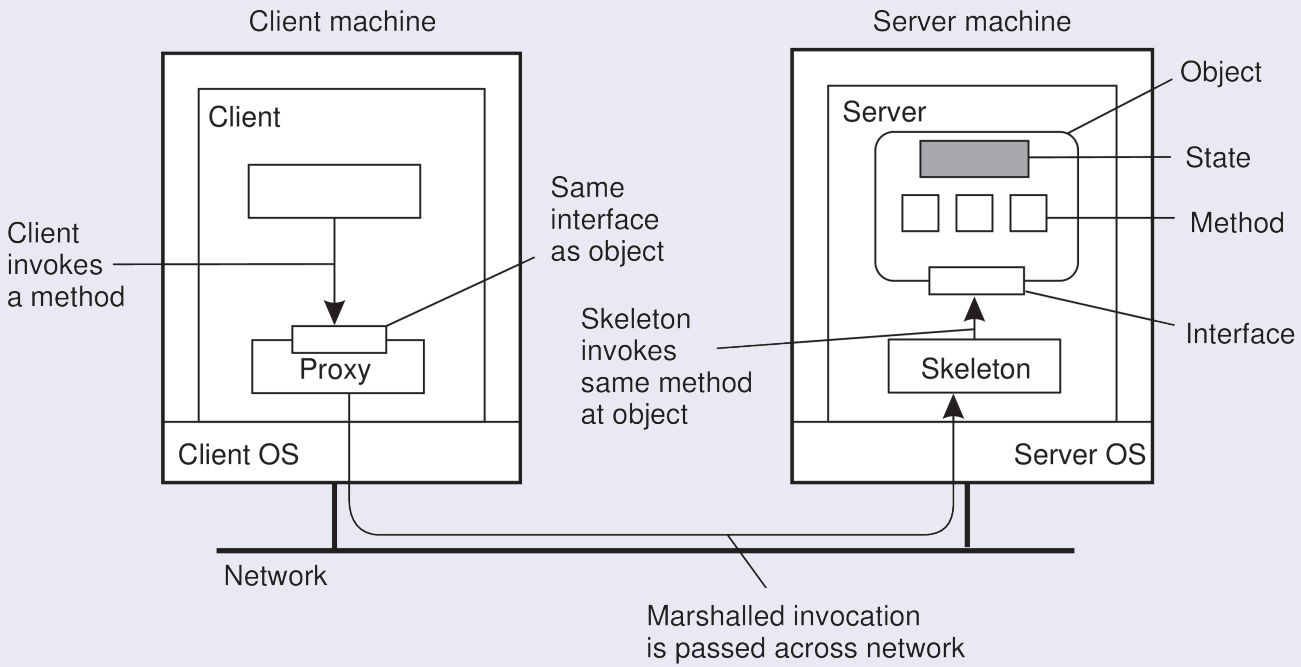




Estilo Baseado em Objetos

Objetos distribuídos

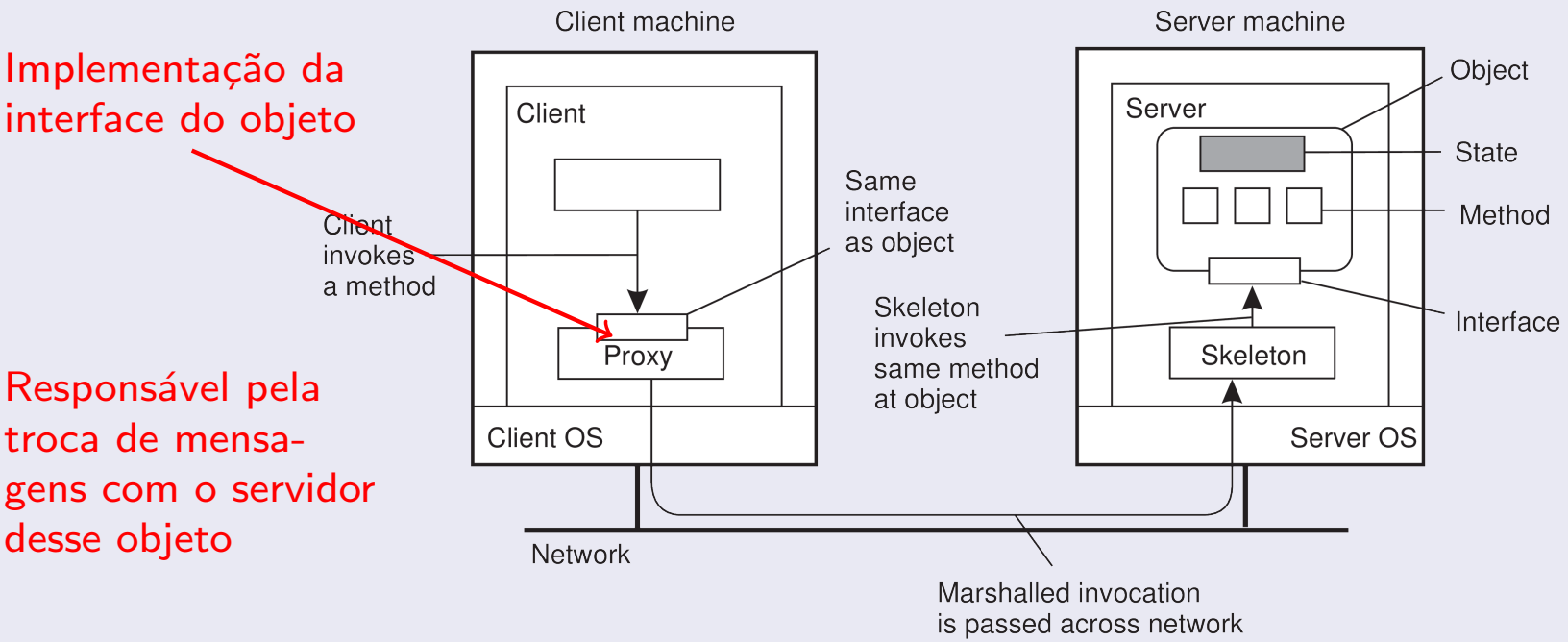
- Podemos então colocar a interface em uma máquina e o objeto em outra
  - Criando assim um objeto distribuído (análogo ao RPC)



Estilo Baseado em Objetos

Objetos distribuídos

- Podemos então colocar a interface em uma máquina e o objeto em outra
  - Criando assim um objeto distribuído (análogo ao RPC)



Estilos Arquiteturais

- Estilo Multicamadas
- Estilo Baseado em Objetos
- **Estilo Baseado em Recursos**
- Estilo Publish-Subscribe

Estilo Baseado em Recursos

Arquiteturas RESTful

- Conectar e integrar vários componentes pode se tornar um pesadelo
  - Solução: olhar o sistema distribuído como uma coleção de recursos, individualmente gerenciados por componentes
  - Recursos podem ser adicionados, removidos, recuperados e modificador por aplicações (remotas)

Estilo Baseado em Recursos

Arquiteturas RESTful

- Conectar e integrar vários componentes pode se tornar um pesadelo
  - Solução: olhar o sistema distribuído como uma coleção de recursos, individualmente gerenciados por componentes
  - Recursos podem ser adicionados, removidos, recuperados e modificador por aplicações (remotas)
- Abordagem conhecida como **Representational State Transfer** (REST)
  - REST não é uma arquitetura, mas sim um estilo arquitetural

Estilo Baseado em Recursos

Arquiteturas RESTful: Características

- Identificação de recursos
  - **Recursos** são identificados usando um único esquema de nomeação
  - Dê nome a qualquer coisa que você queira mencionar
    - Uniform Resource Identifier (URI)

Estilo Baseado em Recursos

Arquiteturas RESTful: Características

- Identificação de recursos
  - **Recursos** são identificados usando um único esquema de nomeação
  - Dê nome a qualquer coisa que você queira mencionar
    - Uniform Resource Identifier (URI)
- Interface uniforme
  - Todos os serviços oferecem a mesma interface
  - Um mesmo conjunto pequeno de operações se aplica a *tudo*
  - A Web RESTful usa métodos HTTP como sua interface uniforme

Estilo Baseado em Recursos

Arquiteturas RESTful: Características

- Mensagens enviadas de ou para um serviço são auto-descritivas
  - Recursos são entidades abstratas (elas não podem ser utilizadas *per se*)
  - Recursos são acessados via *representações de recursos*
    - Representações devem ser suficientes para distinguir um certo recurso
    - Deve-se comunicar qual tipo de representação é usada
    - O formato para a representação pode ser negociado entre os envolvidos

Estilo Baseado em Recursos

Arquiteturas RESTful: Características

- Execução sem estado
  - Isso não significa "**Aplicações** sem Estado"!
    - Em muitas aplicações RESTful, estado é uma parte essencial
    - A ideia de RESTful é evitar transações longas *nas aplicações*

Estilo Baseado em Recursos

Arquiteturas RESTful: Características

- Execução sem estado
  - Isso não significa "**Aplicações** sem Estado"!
    - Em muitas aplicações RESTful, estado é uma parte essencial
    - A ideia de RESTful é evitar transações longas *nas aplicações*
  - "Sem estado" significa mover o estado para os clientes ou recursos

Estilo Baseado em Recursos

Arquiteturas RESTful: Características

- Execução sem estado
  - Isso não significa "**Aplicações** sem Estado"!
    - Em muitas aplicações RESTful, estado é uma parte essencial
    - A ideia de RESTful é evitar transações longas *nas aplicações*
  - "Sem estado" significa mover o estado para os clientes ou recursos
  - Após a execução de uma operação em um serviço, o componente esquece tudo sobre quem chamou a operação
    - Evita manter o estado da aplicação no lado do servidor
    - Apenas o estado do recurso é gerenciado no servidor – é o mesmo para todos os clientes que interagem com o serviço

Estilo Baseado em Recursos

Arquiteturas RESTful: Operações básicas

Operação	Descrição
`GET`	Recupera o estado de um recurso em alguma representação
`POST`	Modifica um recurso, pela transferência de um novo estado
`DELETE`	Apaga um recurso
`PUT`	Cria um novo recurso

(veja `https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods`)

Estilo Baseado em Recursos

Arquiteturas RESTful: Exemplo

- Amazon Simple Storage Service (Amazon S3)

Estilo Baseado em Recursos

Arquiteturas RESTful: Exemplo

- Amazon Simple Storage Service (Amazon S3)
- Objetos (arquivos) são armazenados em buckets (diretórios)

Estilo Baseado em Recursos

Arquiteturas RESTful: Exemplo

- Amazon Simple Storage Service (Amazon S3)
- Objetos (arquivos) são armazenados em buckets (diretórios)
  - Buckets não podem ser colocados dentro de outros buckets

Estilo Baseado em Recursos

Arquiteturas RESTful: Exemplo

- Amazon Simple Storage Service (Amazon S3)
- Objetos (arquivos) são armazenados em buckets (diretórios)
  - Buckets não podem ser colocados dentro de outros buckets
- Operações em `ObjectName` em `BucketName` requerem o seguinte identificador:
  - `http://BucketName.s3.amazonaws.com/ObjectName`

Estilo Baseado em Recursos

Amazon S3: Operações típicas

Todas as operações são realizadas com requisições HTTP:

- Criar um bucket/objeto: `PUT` + URI
- Listar objetos: `GET` em um nome de bucket
- Ler um objeto: `GET` em uma URI completa

Arquiteturas de sistemas distribuídos

Formas de Coordenação

- Sistemas podem estar acoplados no tempo ou referencialmente

Formas de Coordenação

- Sistemas podem estar acoplados no tempo ou referencialmente
- Acoplamento temporal
  - Todos os processos envolvidos na comunicação precisam estar rodando

Arquiteturas de sistemas distribuídos

Formas de Coordenação

- Sistemas podem estar acoplados no tempo ou referencialmente
- Acoplamento temporal
  - Todos os processos envolvidos na comunicação precisam estar rodando
- Acoplamento referencial
  - Processos precisam se "conhecer" explicitamente para se comunicar
  - Precisam de uma referência um do outro

Formas de Coordenação

- Acoplamento temporal e referencial
  - A coordenação é direta
  - Ex: Falar em um celular

	Temporally coupled	**Temporally decoupled**
Referentially coupled	Direct	Mailbox
Referentially decoupled	Event-based	Shared data space

Formas de Coordenação

- Acoplamento temporal e referencial
  - A coordenação é direta
  - Ex: Falar em um celular

	Temporally coupled	**Temporally decoupled**
Referentially coupled	Direct	Mailbox
Referentially decoupled	Event-based	Shared data space

- Acoplamento referencial mas não temporal
  - A coordenação é feita via mailbox
  - Os processos não precisam executar ao mesmo tempo, mas ambos precisam referenciar a mailbox

Formas de Coordenação

- Desacoplamento referencial
  - Os processos não têm uma referencia explícita um ao outro

	Temporally coupled	**Temporally decoupled**
Referentially coupled	Direct	Mailbox
Referentially decoupled	Event-based	Shared data space

- Tudo que podem fazer é publicar uma notificação descrevendo a ocorrência de um evento
- Os demais processo podem então subscrever para receberem determinados tipos de notificações
- Formam a base para as **arquiteturas publish-subscribe**

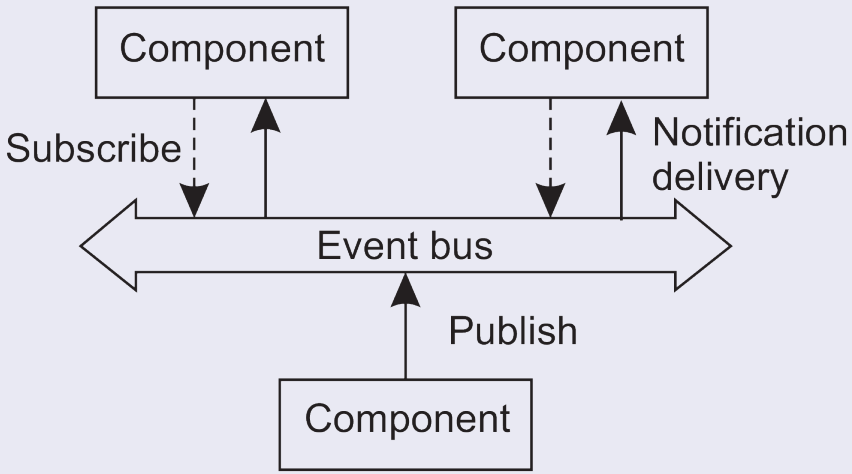
Estilos Arquiteturais

- Estilo Multicamadas
- Estilo Baseado em Objetos
- Estilo Baseado em Recursos
- **Estilo Publish-Subscribe**

Estilo Publish-Subscribe

Modelos Básicos

- A combinação de desacoplamento referencial e acoplamento temporal leva ao modelo de coordenação baseada em eventos

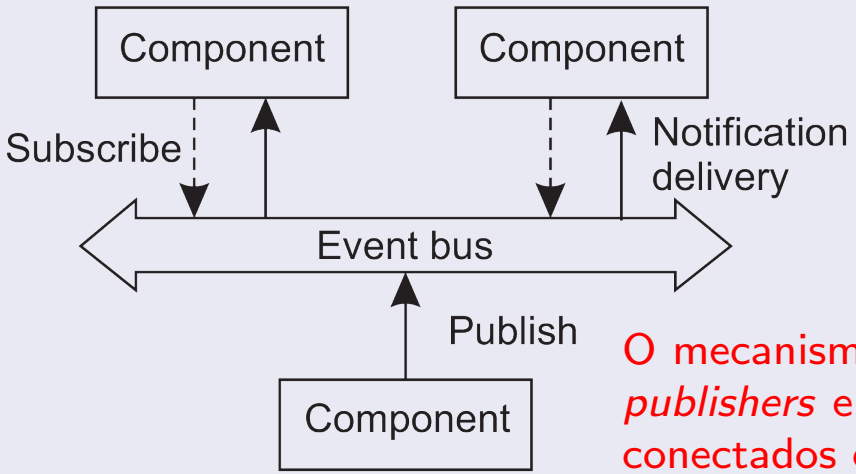


Baseado em eventos

Estilo Publish-Subscribe

Modelos Básicos

- A combinação de desacoplamento referencial e acoplamento temporal leva ao modelo de coordenação baseada em eventos



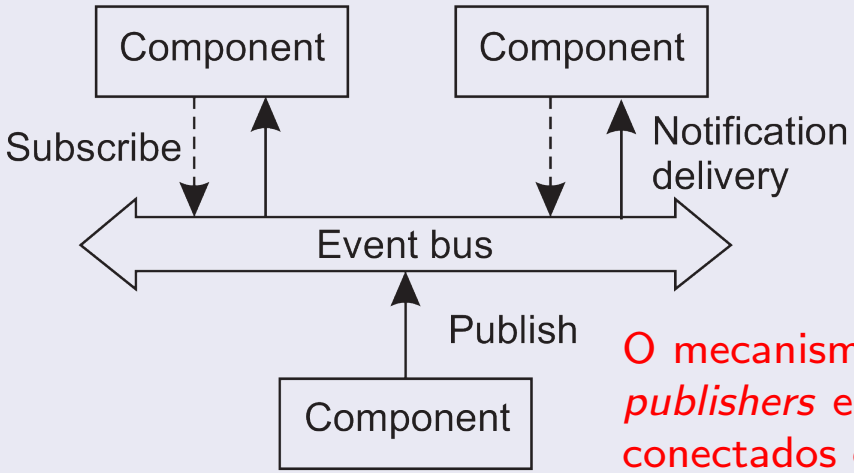
O mecanismo pelo qual *publishers* e *subscribers* são conectados é conhecido como *event bus*

Baseado em eventos

Estilo Publish-Subscribe

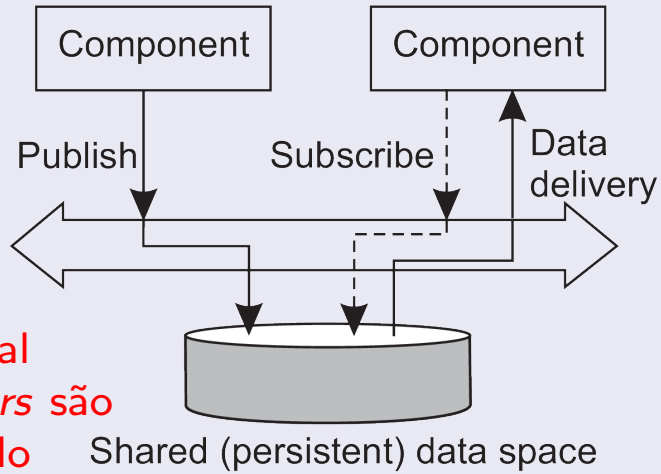
Modelos Básicos

- A combinação de desacoplamento referencial e temporal leva ao modelo de coordenação de espaço de dados compartilhado



Baseado em eventos

O mecanismo pelo qual *publishers* e *subscribers* são conectados é conhecido como *event bus*

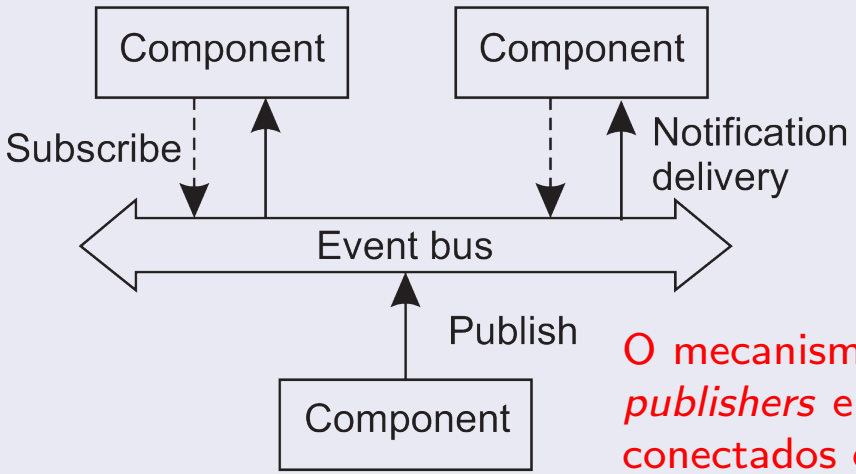


Espaço de dados compartilhado

Estilo Publish-Subscribe

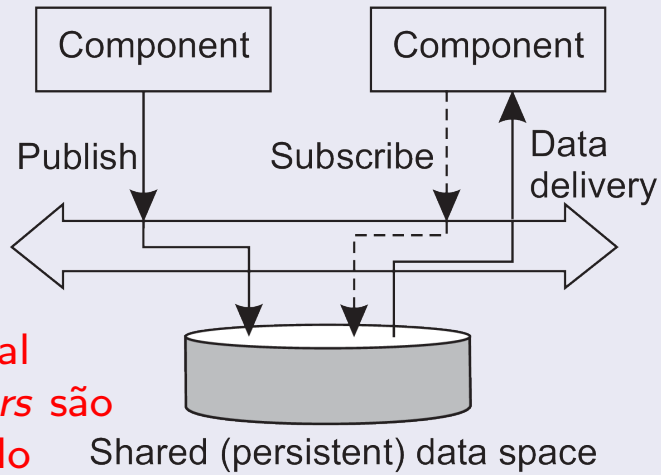
Modelos Básicos

- Modelos mistos também podem existir



Baseado em eventos

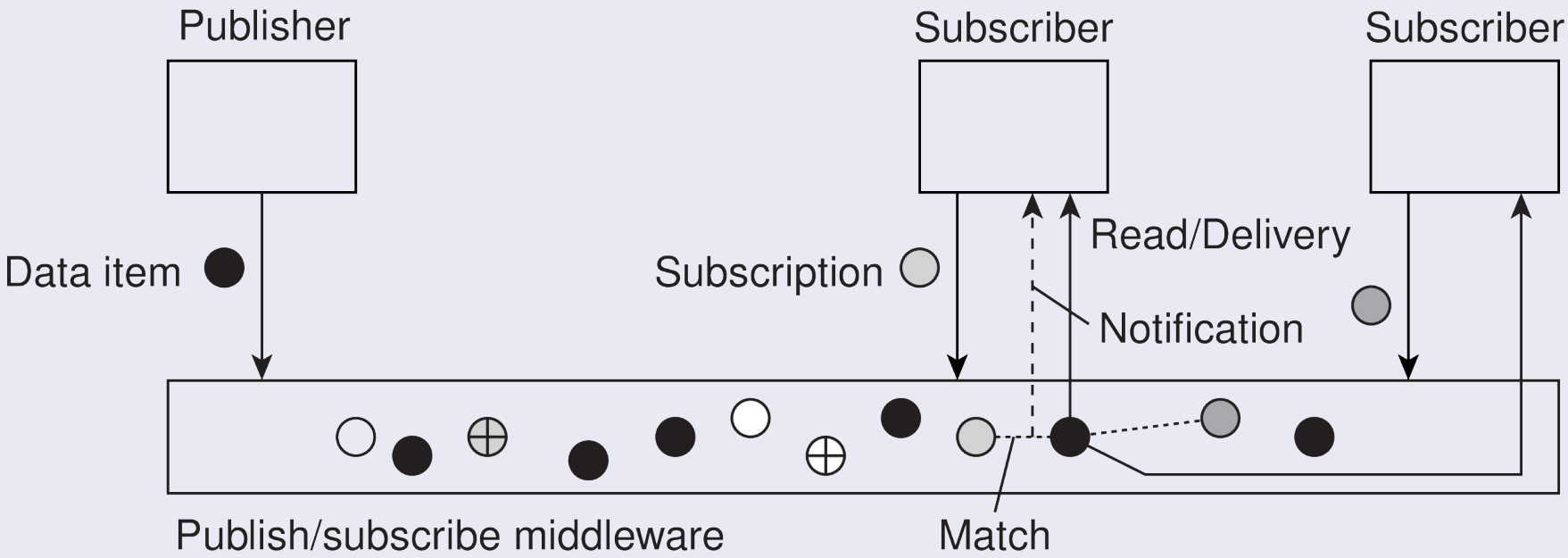
O mecanismo pelo qual *publishers* e *subscribers* são conectados é conhecido como *event bus*



Espaço de dados compartilhado

Estilo Publish-Subscribe

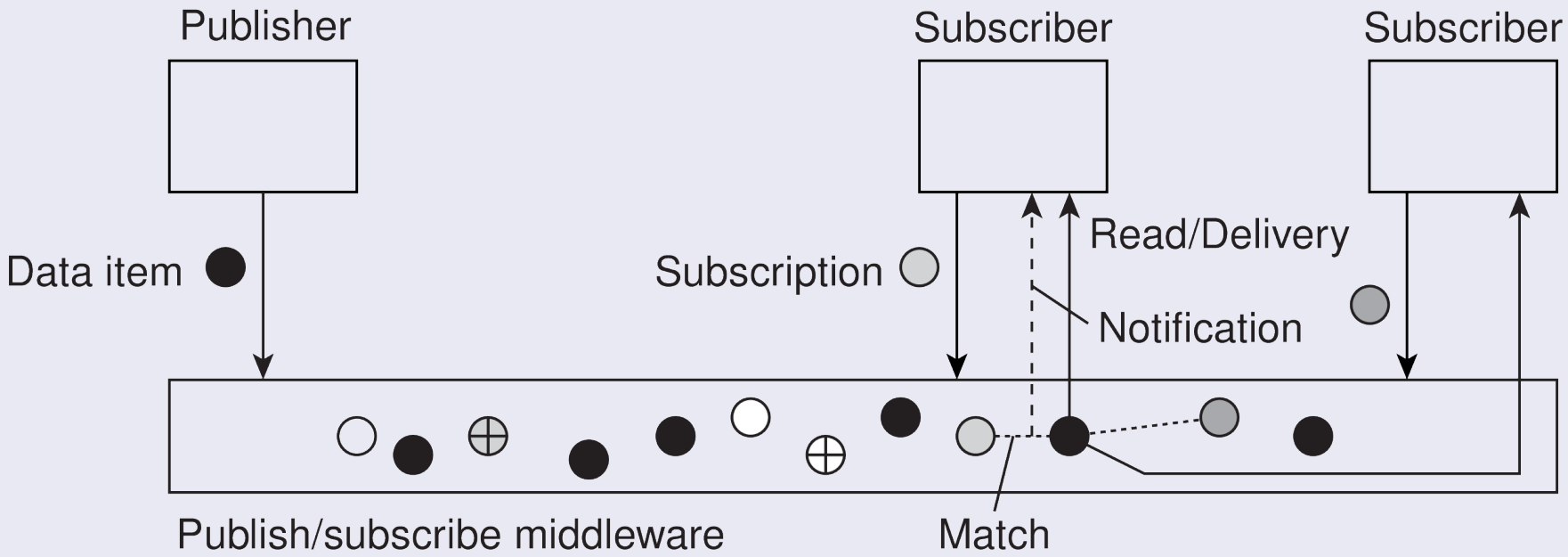
Funcionamento



Estilo Publish-Subscribe

Funcionamento

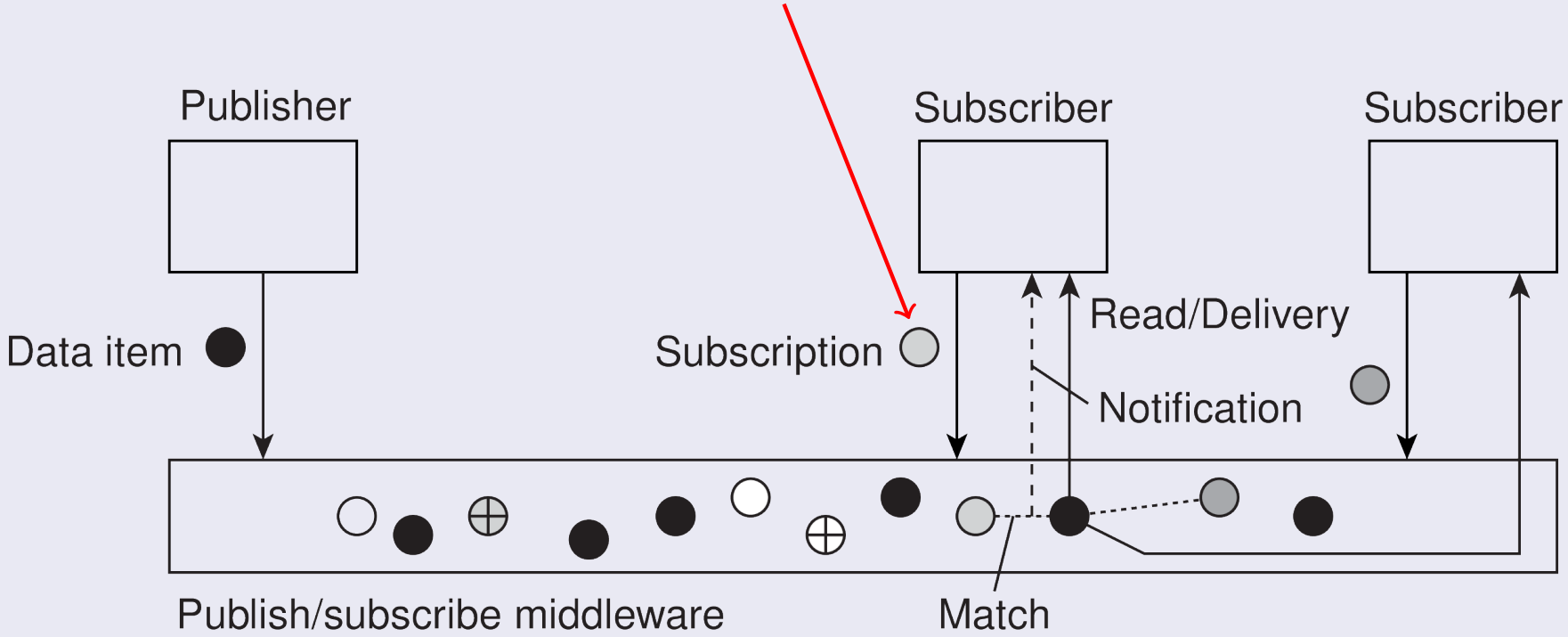
A comunicação começa com a descrição dos eventos de interesse pelo *subscriber*



Estilo Publish-Subscribe

Funcionamento

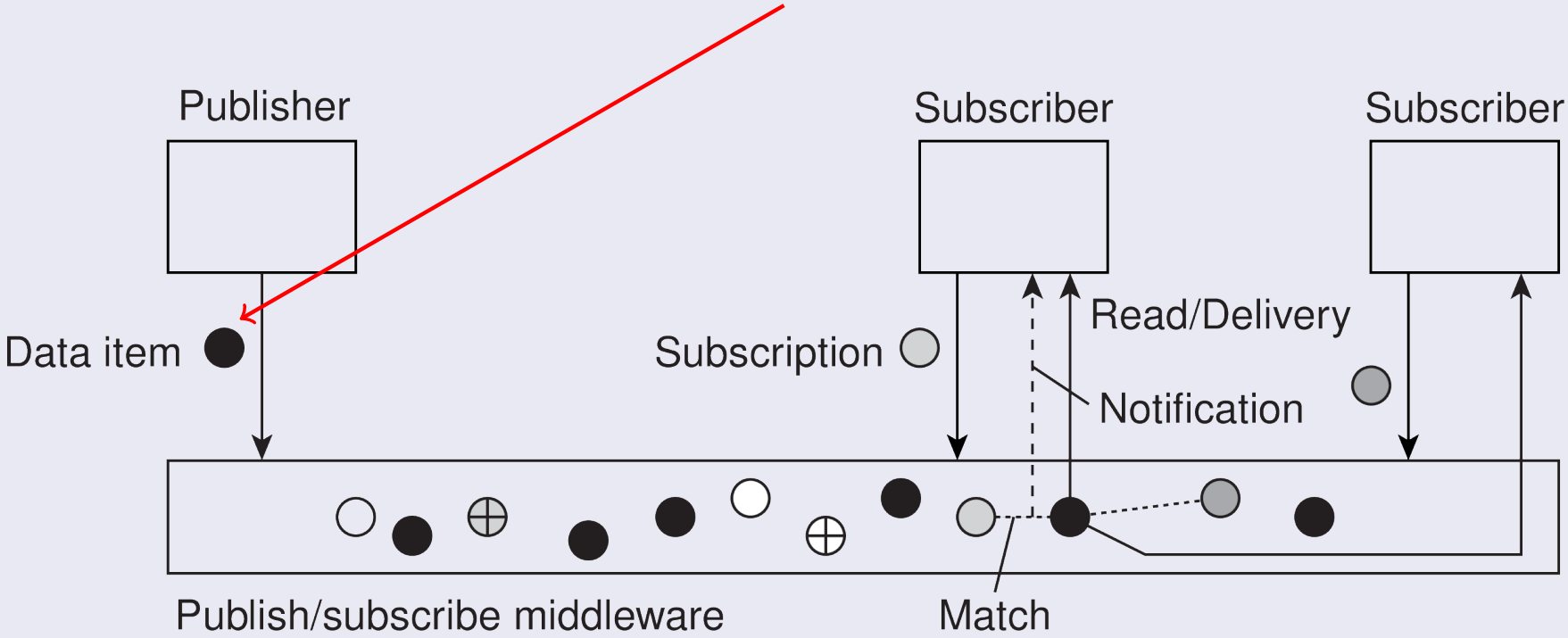
Uma subscrição é feita então ao *middleware*, contendo a descrição dos eventos nos quais o *subscriber* está interessado



Estilo Publish-Subscribe

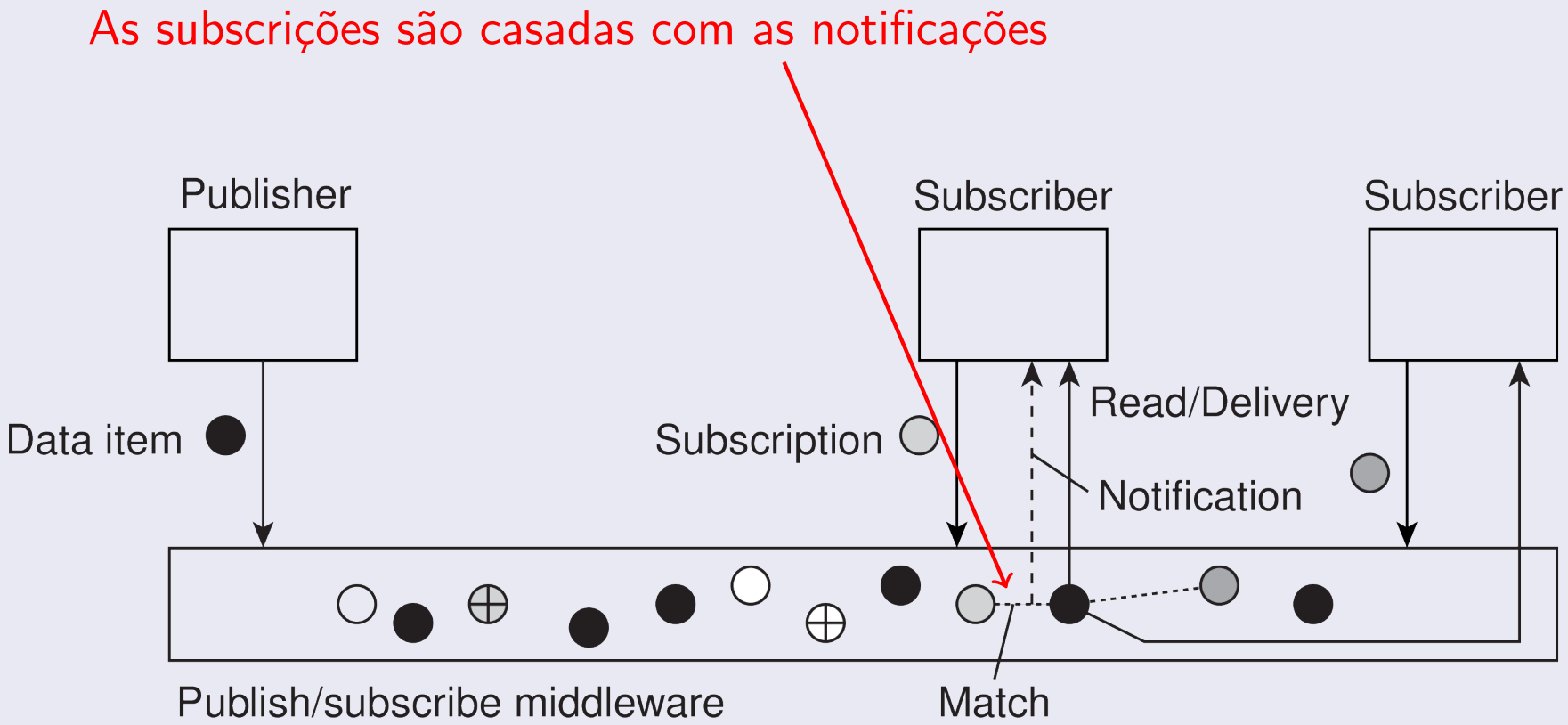
Funcionamento

O *publisher* disponibliza aos demais processos uma notificação descrevendo um evento (a publica)



Estilo Publish-Subscribe

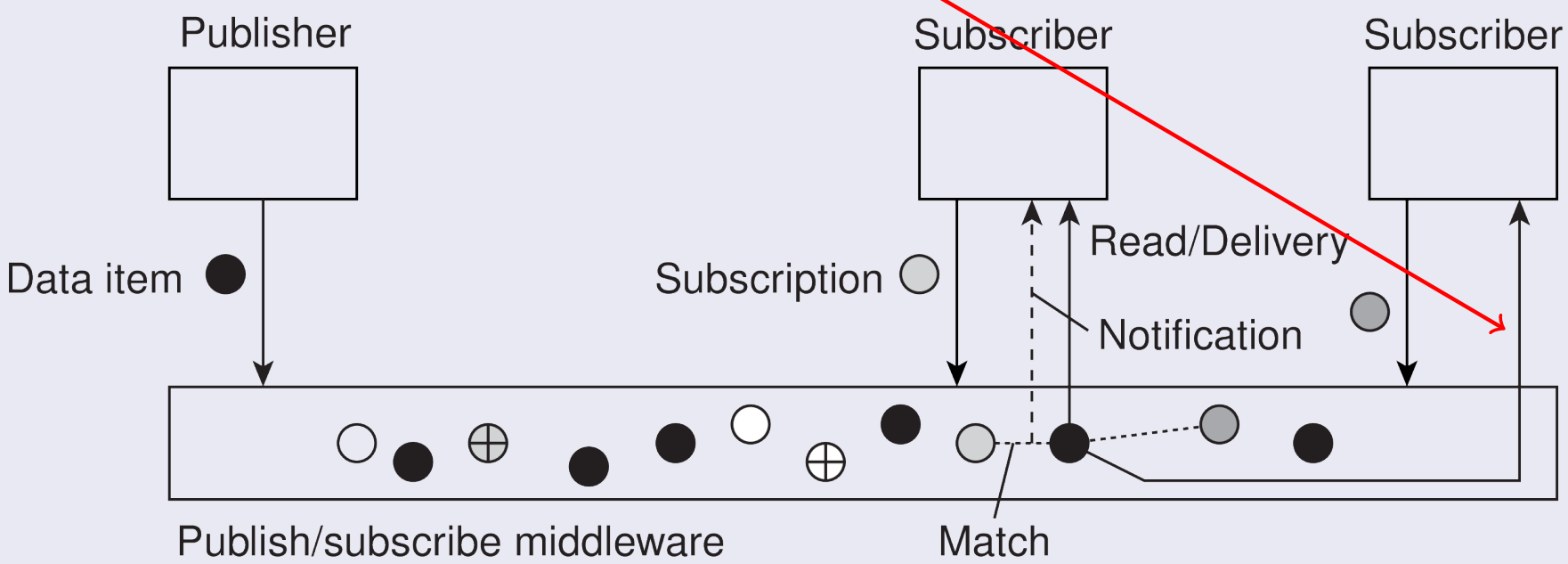
Funcionamento



Estilo Publish-Subscribe

Funcionamento

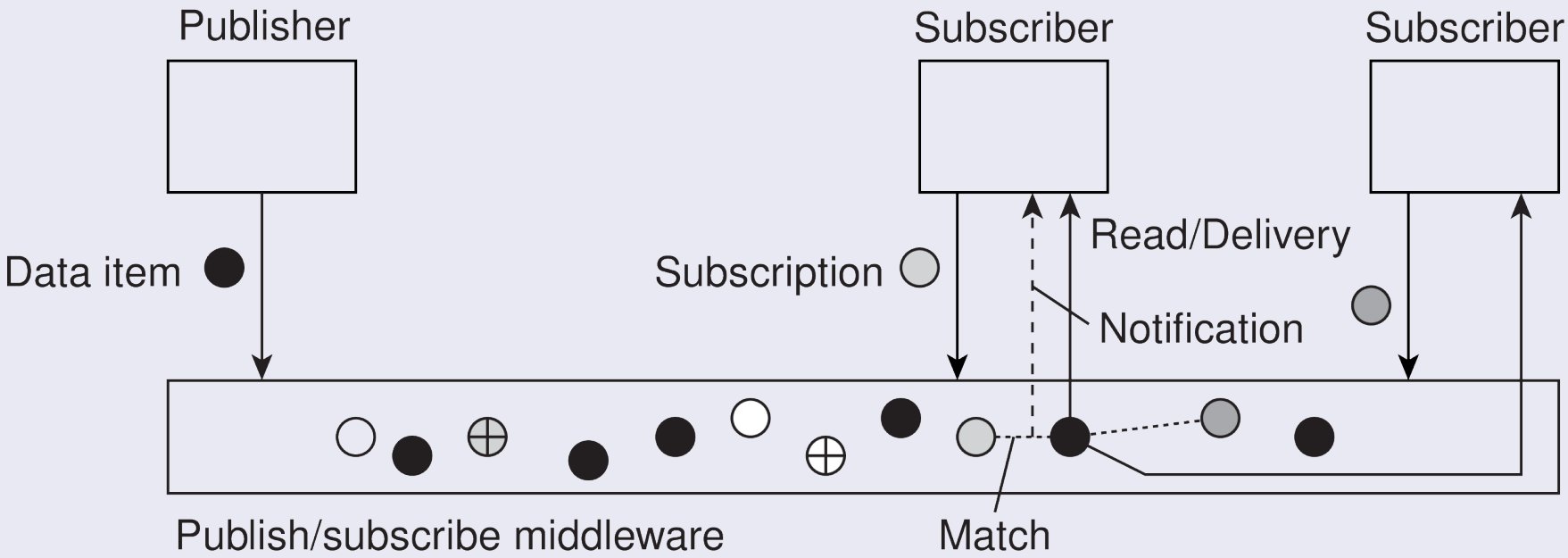
Temos então 2 possibilidades: (1) o *middleware* envia a notificação, juntamente com os dados associados a ela, a todos os *subscribers* que a buscavam



Estilo Publish-Subscribe

Funcionamento

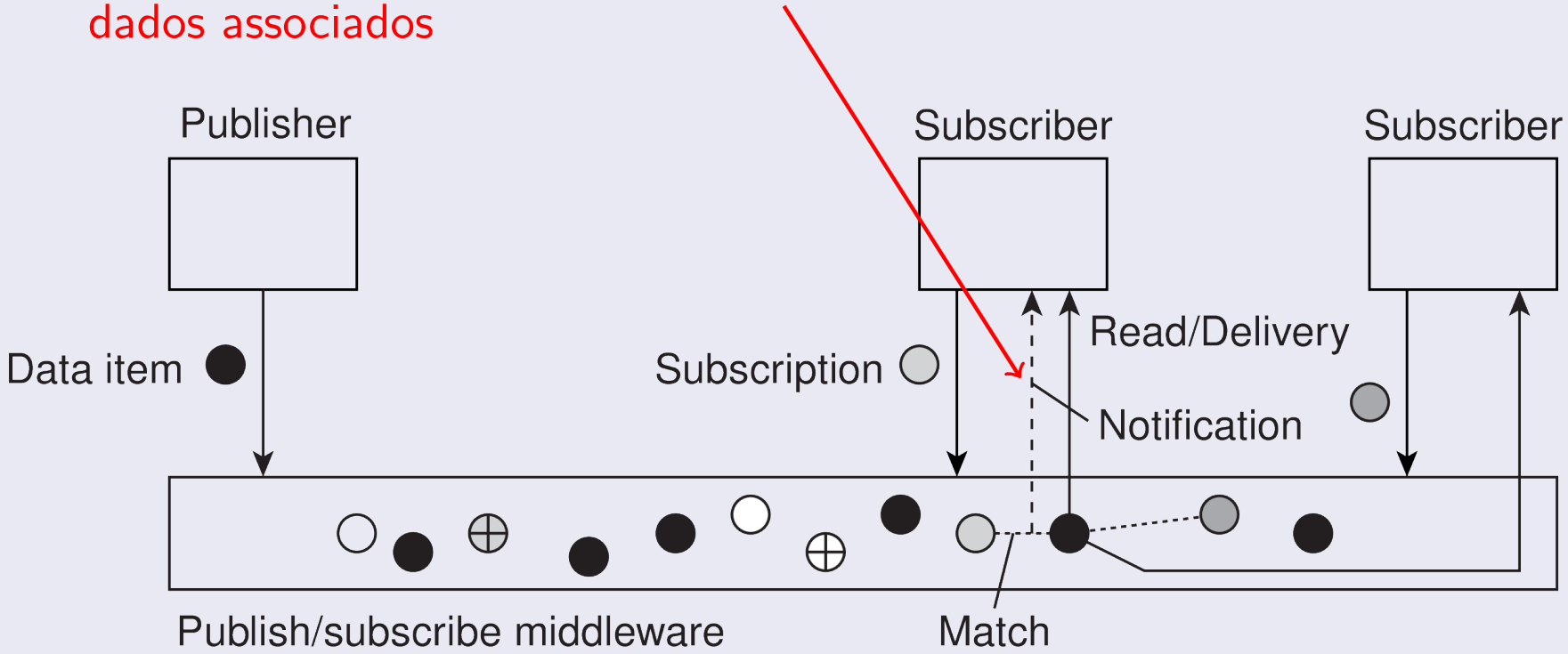
Nesse caso, em geral o *middleware* não armazena os dados – temos um sistema desacoplado referencialmente mas acoplado no tempo



Estilo Publish-Subscribe

Funcionamento

Ou (2) o *middleware* envia apenas a notificação, caso em que os assinantes devem executar uma operação de leitura para obter os dados associados



Estilo Publish-Subscribe

Funcionamento

Forçando assim o middleware a armazenar os dados – temos um sistema desacoplado referencialmente e no tempo