ACH 2147 — Desenvolvimento de Sistemas de Informação Distribuídos

Aula 03: Arquiteturas (01)

Prof. Renan Alves

Escola de Artes, Ciências e Humanidades — EACH — USP

04/03/2024

Estilos arquiteturais

Ideia básica

Um estilo arquitetural é formulado em termos de

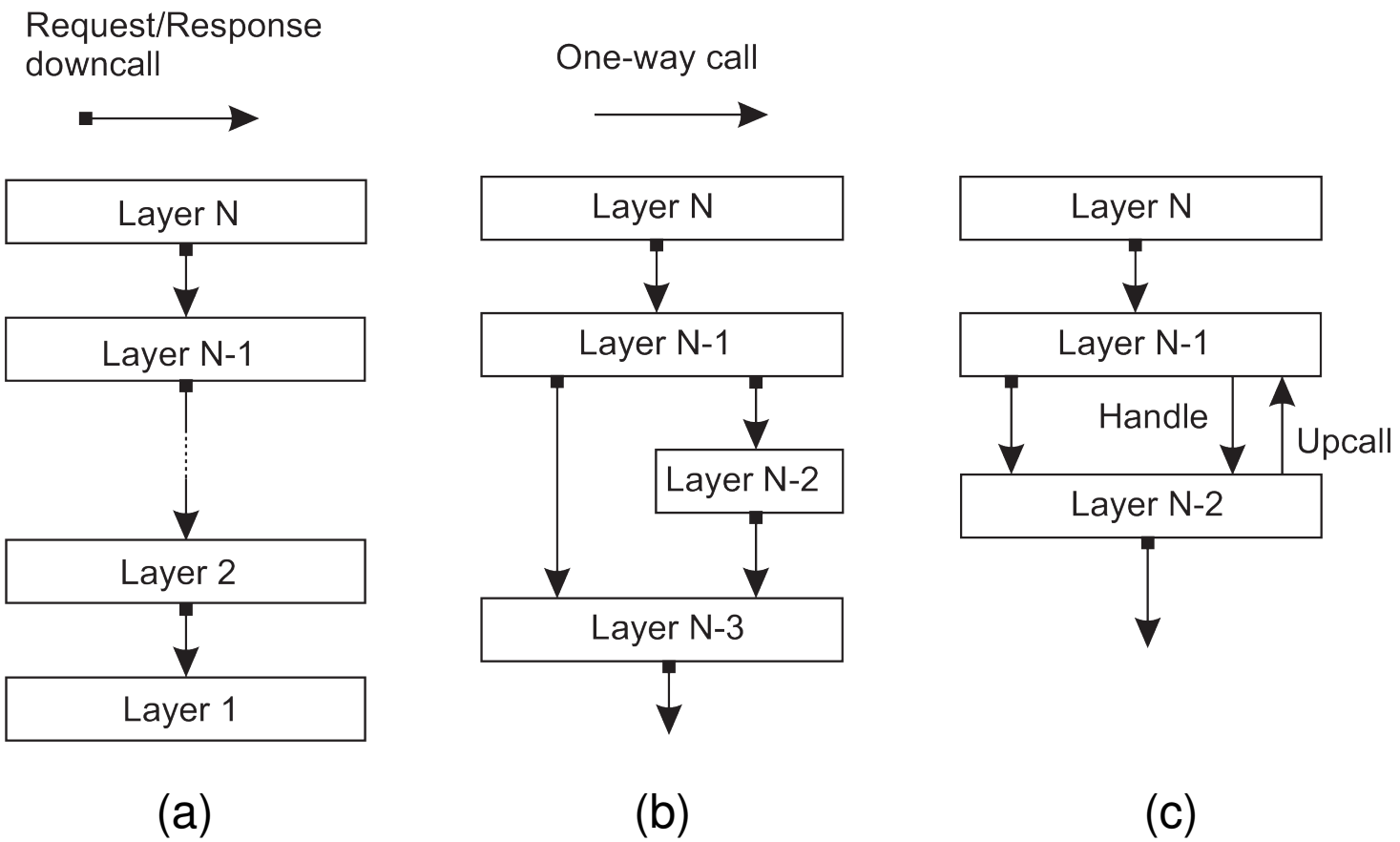
- componentes (substituíveis) com interfaces bem definidas
- a maneira como os componentes estão conectados entre si
- os dados trocados entre os componentes
- a forma como esses componentes e conectores são configurados em conjunto para formar um sistema.

Conector

Um mecanismo que faz a mediação da comunicação, coordenação ou cooperação entre componentes. Exemplo: mecanismos para chamada de procedimento (remota), mensageria ou streaming.

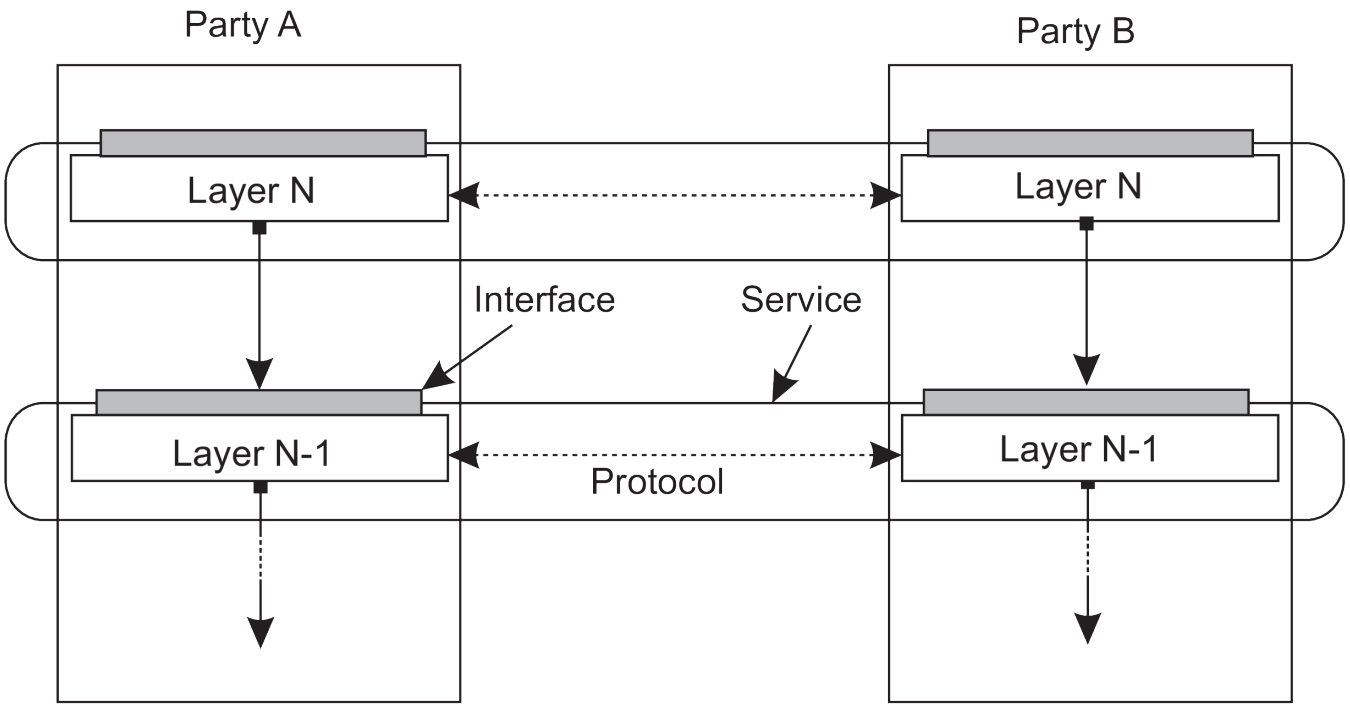
Arquitetura em camadas

Organizações em camadas diferentes



Exemplo: protocolos de comunicação

Protocolo, serviço, interface



Comunicação entre duas partes

Servidor

```python
from socket  import *

s = socket(AF_INET, SOCK_STREAM)
s.bind(("localhost", 5000))# atribui porta
s.listen()                 # deixa o socket pronto para receber conexoes
(conn, addr) = s.accept()  # retorna novo socket e endereco do cliente (bloqueia ate conectar)
while True:                # loop infinito
  data = conn.recv(1024)   # recebe dados do cliente
  if not data: break       # se o cliente parou, sai do loop
  msg = data.decode()+"*"  # processa dados recebidos
  conn.send(msg.encode())  # envia dados processados de volta para o cliente
conn.close()               # fecha a conexao
```

Cliente

```python
from socket  import *

s = socket(AF_INET, SOCK_STREAM)
s.connect(("localhost", 5000)) # conecta ao servidor (bloqueia ate aceitar)
msg = "Hello World"            # defina a mensagem
s.send(msg.encode())           # envia mensagem
data = s.recv(1024)            # recebe a resposta
print(data.decode())           # mostra a resposta
s.close()                      # fecha a conexao
```

Estratificação de Aplicações (layering)

Visão tradicional em três camadas

- Camada de interface de aplicação contém unidades para interface com usuários ou aplicações externas
- Camada de processamento contém as funções de uma aplicação, ou seja, sem dados específicos
- Camada de dados contém os dados que um cliente deseja manipular por meio dos componentes da aplicação

Estratificação de Aplicações (layering)

Visão tradicional em três camadas

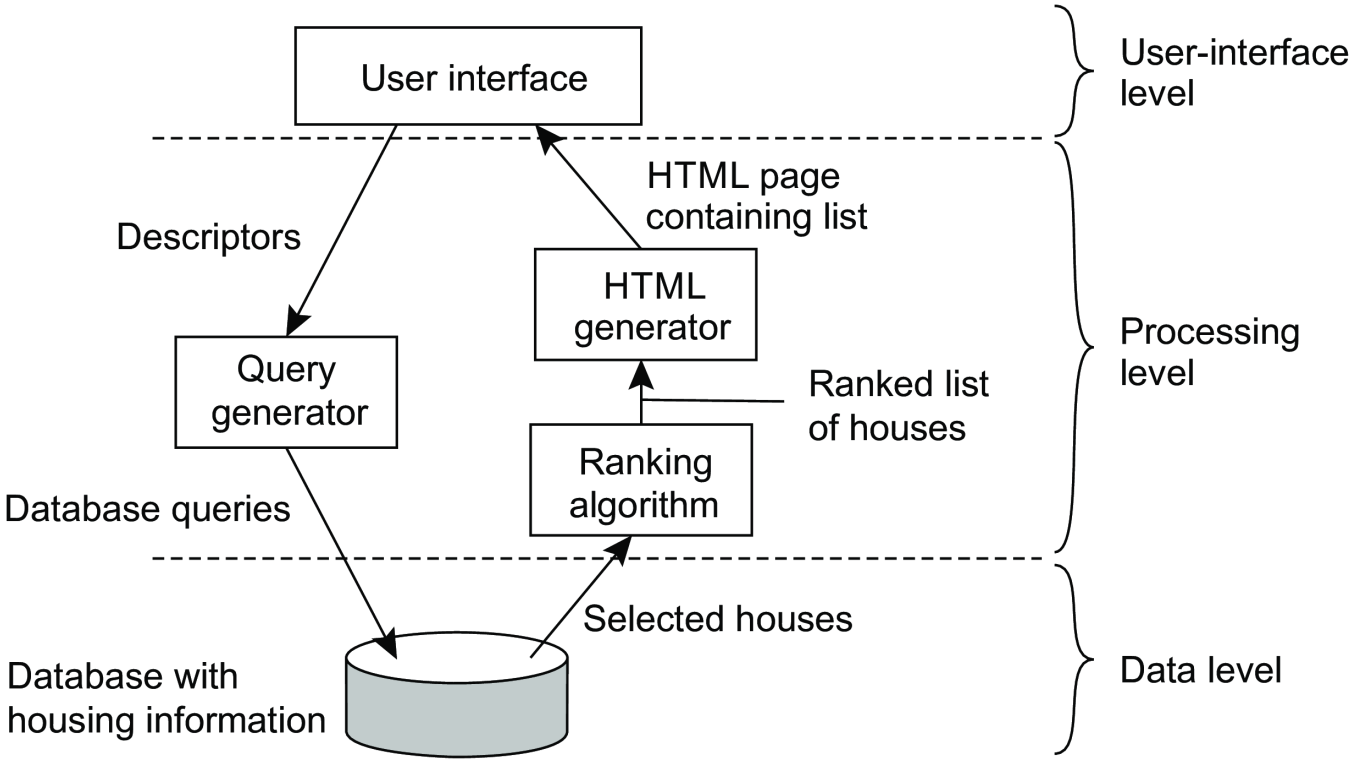
- Camada de interface de aplicação contém unidades para interface com usuários ou aplicações externas
- Camada de processamento contém as funções de uma aplicação, ou seja, sem dados específicos
- Camada de dados contém os dados que um cliente deseja manipular por meio dos componentes da aplicação

Observação

Esta estratificação é encontrada em muitos sistemas de informação distribuídos, usando tecnologia de banco de dados tradicional e aplicativos correspondentes.

Estratificação de Aplicações

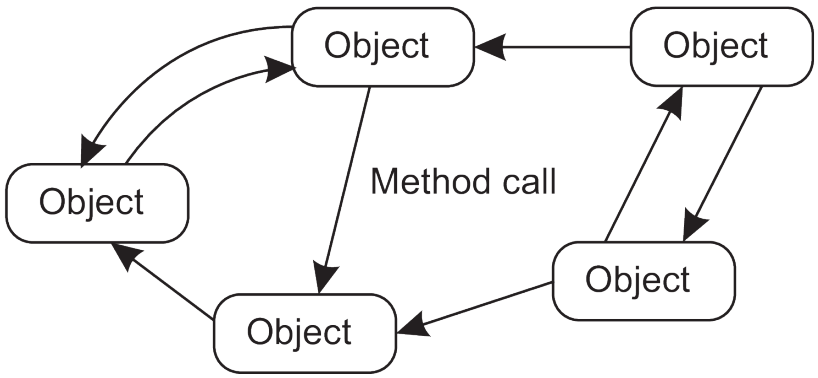
Exemplo: um buscador imobiliário simples

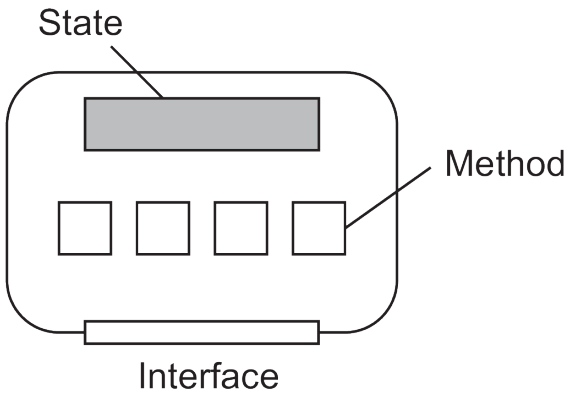


Arquiteturas orientadas a serviço

Arquitetura baseada em objetos

Componentes são objetos, conectados entre si por meio de chamadas de procedimento. Objetos podem ser colocados em máquinas diferentes; chamadas podem, portanto, ser executadas em uma rede.





Encapsulamento

Diz-se que os objetos encapsulam dados e oferecem métodos sobre esses dados sem revelar a implementação interna.

Microsserviços

Arquiteturas RESTful

Essência

Enxerga um sistema distribuído como uma coleção de recursos, gerenciados individualmente por componentes. Recursos podem ser adicionados, removidos, recuperados e modificados por aplicativos (remotos).

1. Recursos são identificados por meio de um único esquema de nomeação
2. Todos os serviços oferecem a mesma interface
3. Mensagens enviadas para um serviço ou de um serviço são totalmente auto-descritivas
4. Após executar uma operação em um serviço, esse componente esquece tudo sobre o chamador (stateless)

Operações básicas

Operação	**Descrição**
`POST`	Criar um novo recurso
`GET`	Recuperar o estado de um recurso em alguma representação
`DELETE`	Excluir um recurso
`PUT`	Modificar um recurso transferindo um novo estado

Exemplo: Amazon S3 (Simple Storage Service)

Essência

Objetos (ou seja, arquivos) são colocados em buckets (ou seja, diretórios). Buckets não podem ser colocados em buckets. Operações em um dado objeto `ObjectName` no bucket `BucketName` requerem o seguinte identificador:

`http://BucketName.s3.amazonaws.com/ObjectName`

Operações típicas

Todas as operações são realizadas enviando solicitações HTTP:

- Criar um bucket/objeto: `PUT`, junto com o URI
- Listar objetos: `GET` em um nome de bucket
- Ler um objeto: `GET` em um URI completo

Sobre interfaces

Problema

Muitas pessoas gostam de abordagens RESTful porque as interface dos serviços são simples. O problema é que muito precisa ser feito no espaço de parâmetros.

Interface SOAP (Simple Object Access Protocol) do Amazon S3

Bucket operations	**Object operations**
ListAllMyBuckets	PutObjectInline
CreateBucket	PutObject
DeleteBucket	CopyObject
ListBucket	GetObject
GetBucketAccessControlPolicy	GetObjectExtended
SetBucketAccessControlPolicy	DeleteObject
GetBucketLoggingStatus	GetObjectAccessControlPolicy
SetBucketLoggingStatus	SetObjectAccessControlPolicy

Sobre interfaces

Simplificações

Suponha uma interface `bucket` oferecendo uma operação `create`, exigindo a string de entrada `mybucket`, para criar um bucket ”mybucket”.

Sobre interfaces

Simplificações

Suponha uma interface `bucket` oferecendo uma operação `create`, exigindo a string de entrada `mybucket`, para criar um bucket ”mybucket”.

SOAP

```
import bucket
bucket.create("mybucket")
```

Sobre interfaces

Simplificações

Suponha uma interface `bucket` oferecendo uma operação `create`, exigindo a string de entrada `mybucket`, para criar um bucket ”mybucket”.

SOAP

```
import bucket
bucket.create("mybucket")
```

RESTful

```
PUT "https://mybucket.s3.amazonsws.com/"
```

Sobre interfaces

Simplificações

Suponha uma interface `bucket` oferecendo uma operação `create`, exigindo a string de entrada `mybucket`, para criar um bucket ”mybucket”.

SOAP

```
import bucket
bucket.create("mybucket")
```

RESTful

```
PUT "https://mybucket.s3.amazonsws.com/"
```

Conclusões

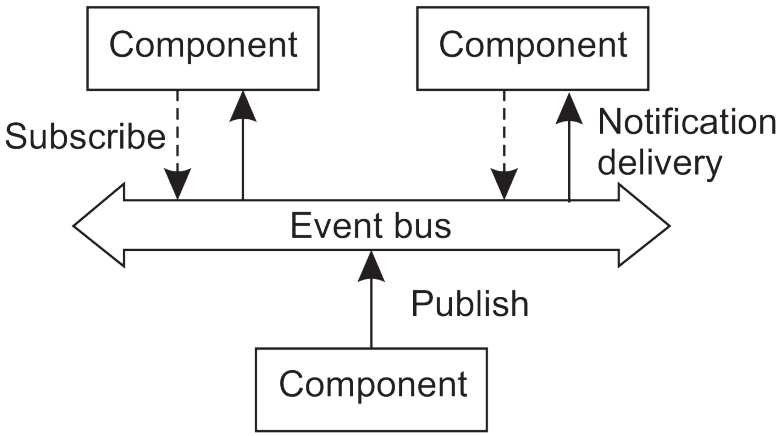
- Verificação de erros
- Semântica da operação

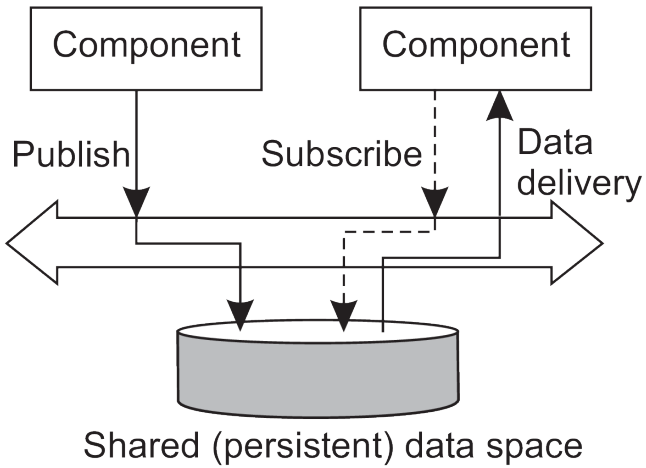
Arquiteturas de produtor/consumidor

Coordenação: Acoplamento temporal e referencial

	Acoplamento temporal	**Desacoplamento temporal**
Acoplamento referencial	Comunicação direta	Caixa de correio
Desacoplamento referencial	Baseado em evento	Espaço de dados compartilhado

Baseado em evento e espaço de dados compartilhado





Exemplo: Espaço de tuplas do tipo Linda

Três operações simples

- `in(t)`: remover uma tupla que corresponde ao modelo `t`
- `rd(t)`: obter cópia de uma tupla que corresponde ao modelo `t`
- `out(t)`: adicionar a tupla `t` ao espaço de tuplas

Mais detalhes

- Chamar `out(t)` duas vezes seguidas leva a armazenar duas cópias da tupla `t` $\Rightarrow$ um espaço de tuplas é modelado como um multiconjunto (multiset).
- Tanto `in` quanto `rd` são operações bloqueantes: o chamador será bloqueado até uma tupla correspondente ser encontrada ou estar disponível.

Exemplo: Espaço de tuplas do tipo Linda

Bob:

```
1  import linda
2  linda.connect()
3
4  blog = linda.TupleSpace()
5  linda.universe._out(("MicroBlog",blog))
6
7  blog = linda.universe._rd(("MicroBlog",linda.TupleSpace))[1]
8
9  blog._out(("bob","distsys","I am studying chap 2"))
10 blog._out(("bob","distsys","The linda example's pretty simple"))
11 blog._out(("bob","gtcn","Cool book!"))
```

Alice:

```
1  import linda
2  linda.connect()
3
4  blog = linda.universe._rd(("MicroBlog",linda.TupleSpace))[1]
5
6  blog._out(("alice","gtcn","This graph theory stuff is not easy"))
7  blog._out(("alice","distsys","I like systems more than graphs"))
```

Chuck:

```
1  import linda
2  linda.connect()
3
4  blog = linda.universe._rd(("MicroBlog",linda.TupleSpace))[1]
5
6  t1 = blog._rd(("bob","distsys",str))
7  t2 = blog._rd(("alice","gtcn",str))
8  t3 = blog._rd(("bob","gtcn",str))
9
10 print t1
11 print t2
12 print t3
```

Exemplo: Espaço de tuplas do tipo Linda

Bob:

```
1  import linda
2  linda.connect()
3
4  blog = linda.TupleSpace()
5  linda.universe._out(("MicroBlog",blog))
6
7  blog = linda.universe._rd(("MicroBlog",linda.TupleSpace))[1]
8
9  blog._out(("bob","distsys","I am studying chap 2"))
10 blog._out(("bob","distsys","The linda example's pretty simple"))
11 blog._out(("bob","gtcn","Cool book!"))
```

Alice:

```
1  import linda
2  linda.connect()
3
4  blog = linda.universe._rd(("MicroBlog",linda.TupleSpace))[1]
5
6  blog._out(("alice","gtcn","This graph theory stuff is not easy"))
7  blog._out(("alice","distsys","I like systems more than graphs"))
```

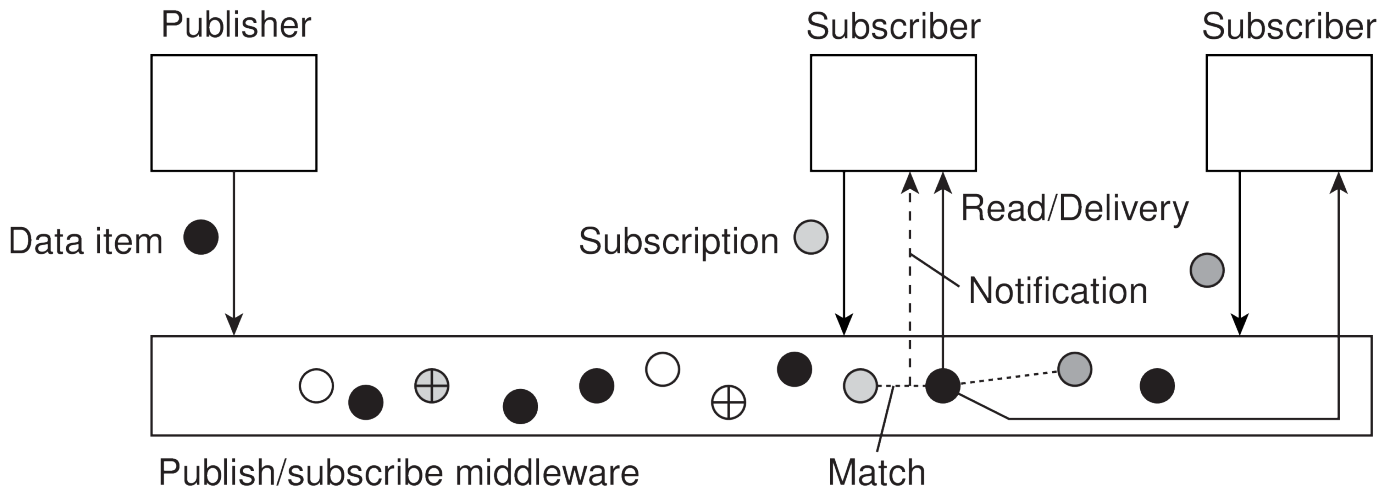
Chuck:

```
1  import linda
2  linda.connect()
3
4  blog = linda.universe._rd(("MicroBlog",linda.TupleSpace))[1]
5
6  # nao eh necessario conhecer ID do publicador
7  t1 = blog._rd((str,"distsys",str))
8  t2 = blog._rd((str,"gtcn",str))
9
10 print t1
11 print t2
```

Produtor e consumidor

Questão: como encontrar eventos correspondentes?

- Suponha que eventos sejam descritos por pares (atributo,valor)
- assinatura baseada em tópico: especificar uma série de "atributo = valor"
- assinatura baseada em conteúdo: "atributo $\in$ intervalo"
- eventos compostos: "atributo1 = valor AND atributo2 = valor"



Observação

Assinaturas baseadas em conteúdo podem facilmente ter problemas sérios de escalabilidade (por quê?)