Live de Python

#116 - Observer / PubSub / Dispatcher

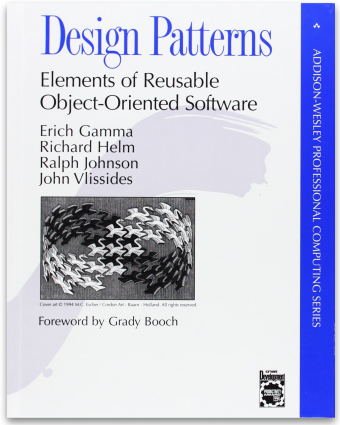
Obrigado!

apoia.se/livedepython

```
dunossauro at localhost in ~/git/apoiase on master*
$ python apoiadores.py
Alexandre Possebom   Alexandre Tsuno      Alysson Oliveira     Amaziles Carvalho
And Past             Andre Machado        Andre Rodrigues      Athayr Athayr
Bernardo Fontes      Bruno Oliveira       Bruno Gaffuri        Bruno Guizi
Bruno Rocha          Carlos Augusto       Cleber Santos        Cleiton Mittmann
David Reis           David Silva          Dayham Soares        Diego Ubirajara
Edimar Fardim        Eliabe Silva         Eliakim Morais       Elias Soares
Eliel Lima           Emerson Lara         Eugenio Mazzini      Fabiano Silos
Fabiano Teichmann    Fabiano Gomes        Fausto Caldeira      Fernando Furtado
Franklin Silva       Fábio Serrão         Gleison Oliveira     Guilherme Ramos
Hemilio Lauro        Humberto Rocha       Hélio Neto           JONATHAN DOMINGOS
Jean Vetorello       Johnny Tardin        Jonatas Oliveira     Jonatas Simões
José Prado           João Lugão           João Coelho          Juan Gutierrez
Jucélio Silva        Júlia Kastrup        Kauan Alves          Leon Teixeira
Lucas Nascimento     Magno Malkut         Marcello Benigno     Marcus Salgues
Maria Boladona       Matheus Francisco    Nilo Pereira         Nídio Dolfini
Pablo Henrique       Patrick Corrêa       Paulo Tadei          Pedro Alves
Rafael Galleani      Regis Santos         Renan Moura          Renato Santos
Rennan Almeida       Renne Rocha          Rhenan Bartels       Rodrigo Ferreira
Rodrigo Vaccari      Sérgio Passos        Thais Viana          Thiago Araujo
Tiago Cordeiro       Tyrone Damasceno     Vergil Valverde      Vicente Marcal
Wander Silva         Welington Carlos     Wellington Camargo   Welton Souza
William Oliveira     Willian Gl           Yros Aguiar          Falta você
```

Roteiro

- Recapitulando
- Observer
- PubSub
- Dispatcher

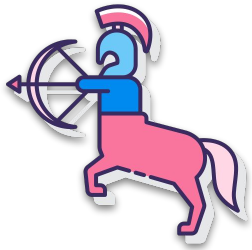


pg 275

Observer

Fausto conversou com todas as raças e percebeu que só poderia fazer alguma coisa quando os mortos realmente voltassem a vida

Assim decidiu chamar Centauros e Unicórnios para que rodeassem a terra e o avisassem quando os mortos voltassem



Fausto

Fausto

Fausto então fez um feitiço que deixava sua bola preparada para receber as atualizações dos unicórnios e centauros. Ele poderia estar na escola, então melhor se preparar para receber as mensagens.

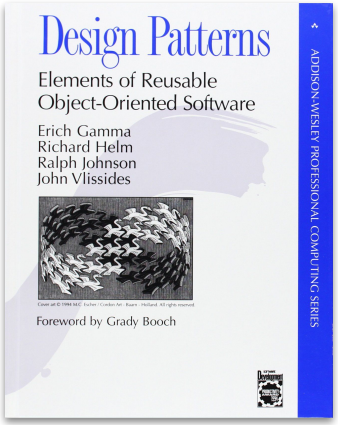
Por motivos de efeitos especiais vamos ter que usar código

```python
class Observador(ABC):
    """Feitiço para objetos que querem ser observados."""

    @abstractmethod
    def atualizar(self):
        ...
```

```python
class Observavel(ABC):
    """Feitiço observar objetos observáveis."""

    @abstractproperty
    def _observers() -> List:
        ...

    @abstractmethod
    def adicionar_observer(self, observer):
        ...

    @abstractmethod
    def notificar_observers(self, mensagem):
        ...
```

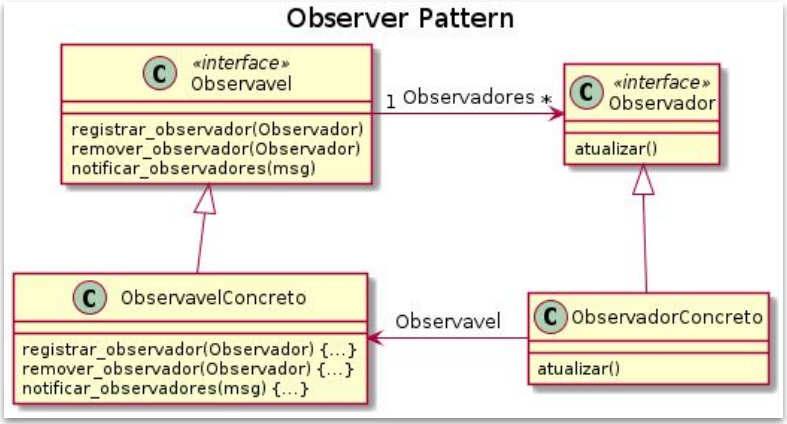
Abstrato quer dizer que é a ideia de como implementar

Padrão Observer [Observador]



pg 275

- Intenção
  - Permitir que um objeto seja capaz de notificar outro objeto
- Motivação
  - define uma dependência um-para-muitos entre objetos para que, quando um objeto mudar de estado, todos os seus dependentes sejam notificados e atualizados automaticamente



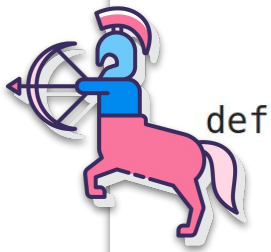
Live de Python

Live code

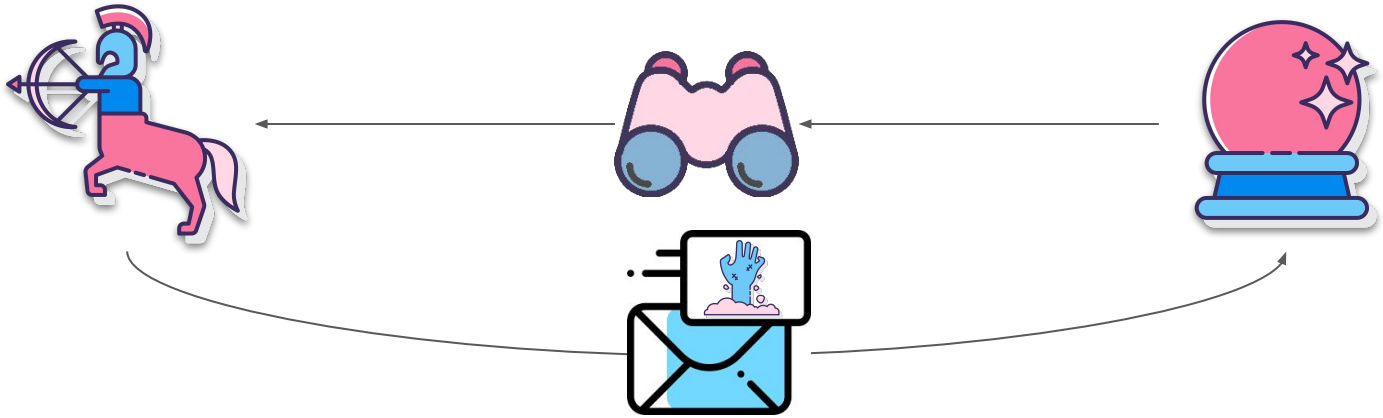
Vamos ajudar o Fausto a resolver seu problema

Resolução clássica

```
class BolaDeCristal:
    def atualizar(self, mensagem):
        print(f"Fausto está na escola, mas recebeu a mensagem: {mensagem}")

class Centauro:
    def __init__(self):
        self._observers = []

    def adicionar_observer(self, observador):
        self._observers.append(observador)

    def notificar_observers(self, mensagem):
        for observador in self._observers:
            observador.atualizar(mensagem)
```



Resolução clássica



```
centauro = Centauro()
centauro.adicionar_observer(BolaDeCristal())
centauro.notificar_observers('Os mortos chegaram #medo')
```

Push com funções

```
def observador_email(mensagem):
    print(f'observador_email recebeu a mensagem: {mensagem}')


def observador_impressora(mensagem):
    print(f'observador_impressora recebeu a mensagem: {mensagem}')


obs = Observavel()
obs.adicionar_observer(observador_email)
obs.adicionar_observer(observador_impressora)
obs.notificar_observers('A live de python é as 22')
```

Voltando a história

Fausto chegou em casa e ficou extremamente preocupado. Geraldo, o unicórnio, tinha enviado mais de 200 mensagens para sua bola de cristal e ele não sabia selecionar o que realmente eram informações relevantes.

Unicórnios eram spammers.

Fausto

Geraldo

```
$ tail -5 boladecristal.log
>>> Unicórnios notificam: GERALDO: Hora de comer, rapaize
>>> Unicórnios notificam:    LUANDA: Já chegou o disco voador
>>> Unicórnios notificam:    TYRONE: Django é melhor do que flask
>>> Unicórnios notificam:   ALLYTHY: Combina com Pipenv
>>> Unicórnios notificam:      DEBS: Esse livro do Pessoa é clássico
```

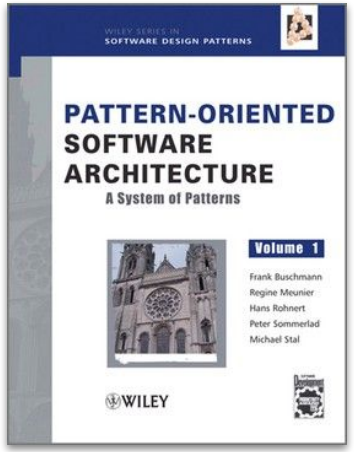
Logs da bola de cristal

Fausto chegou a conclusão, depois de algum tempo, que os unicórnios tinham um método bem eficiente de conversar. Afinal, eles usavam telepatia. O que quer dizer que todos ouviam todos ao mesmo tempo.

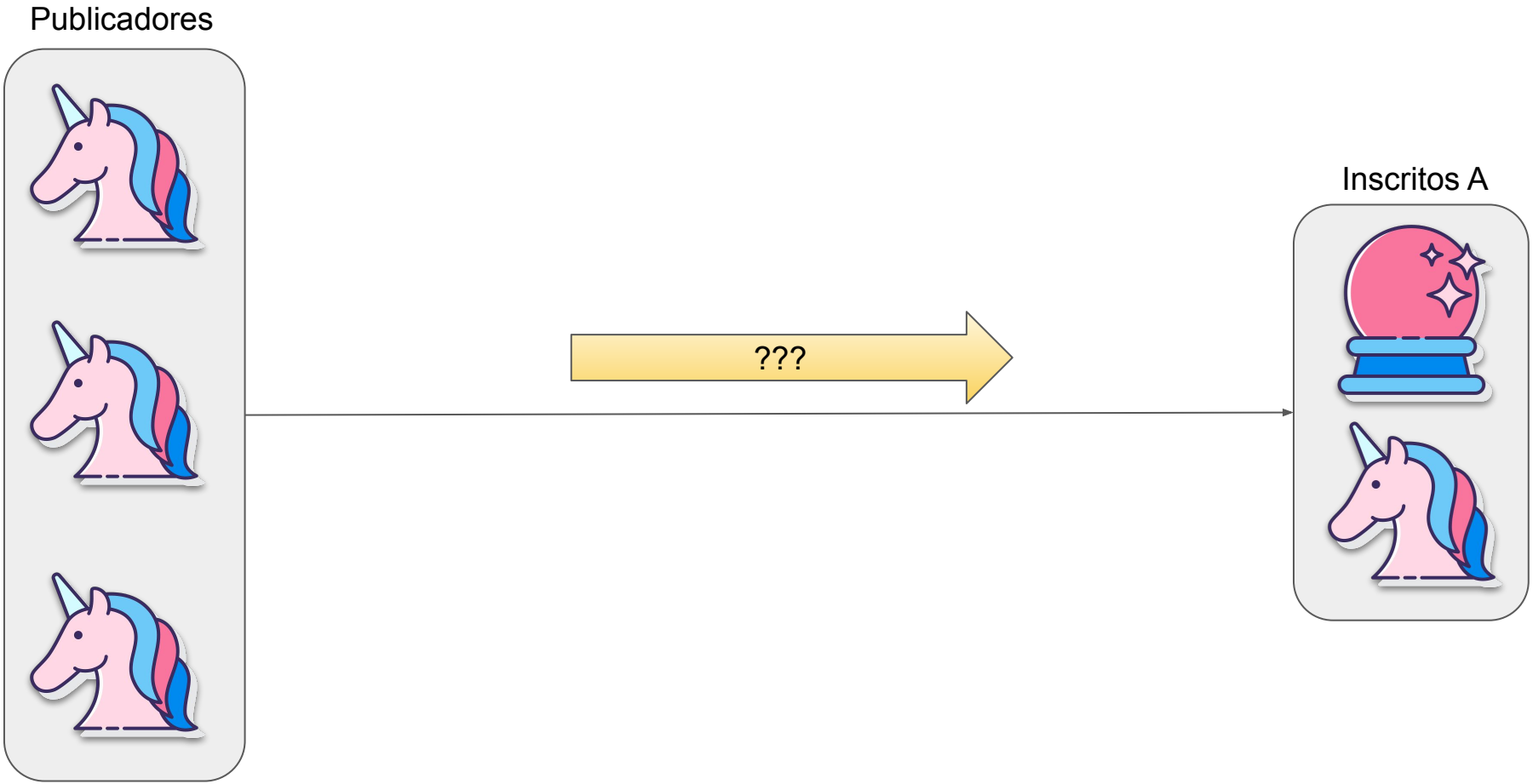
Esse modelo teria que ser adaptado.

Fausto

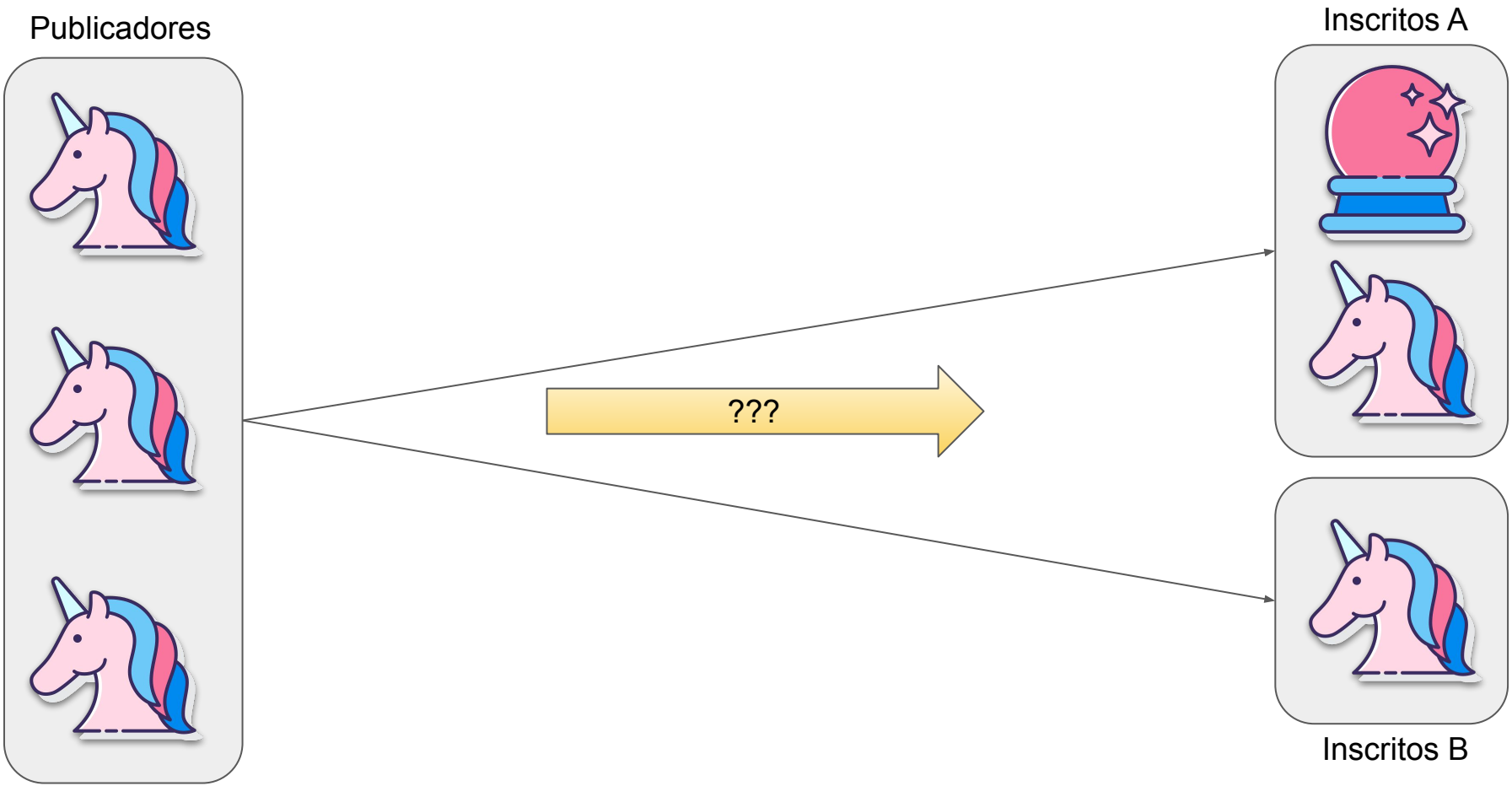
Geraldo


WILEY SERIES IN
SOFTWARE DESIGN PATTERNS
PATTERN-ORIENTED
SOFTWARE
ARCHITECTURE
A System of Patterns
Volume 1
Frank Buschmann
Regine Meunier
Hans Rohnert
Peter Sommerlad
Michael Stal
WILEY

Pub/Sub



https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern#Message_filtering



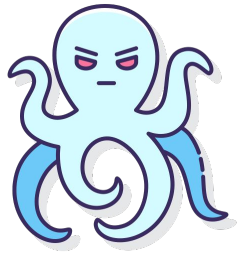
https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern#Message_filtering

Live de Python

Fausto ficou sabendo que na copa de 2010 um polvo chamado Paul foi o mensageiro das trevas e que foi mandado de volta ao inferno por fortalecer o sistema de apostas.

mas Fausto sabia que ele entendia de compartilhamento de mensagens. Então pediu uma ajuda a ele

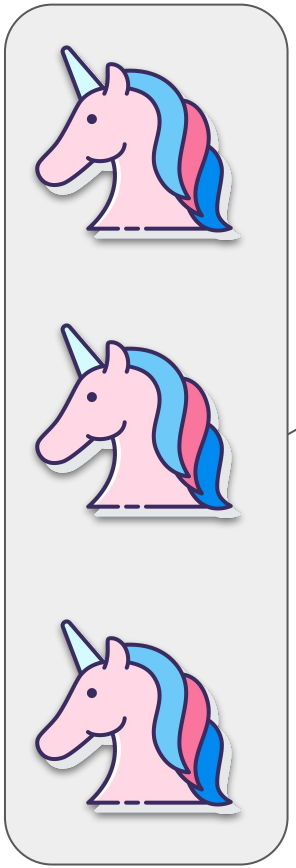
Fausto

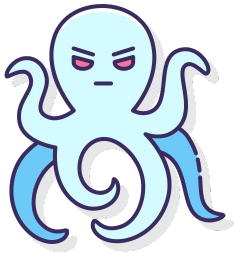


Paul

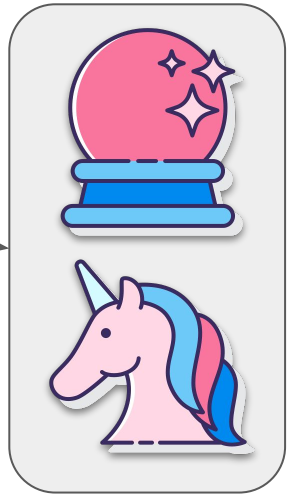
https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern#Message_filtering

Publicadores

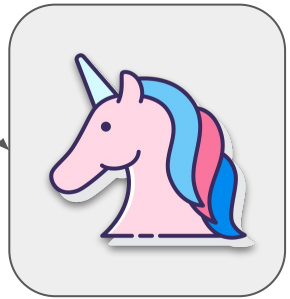




Inscritos A



Inscritos B

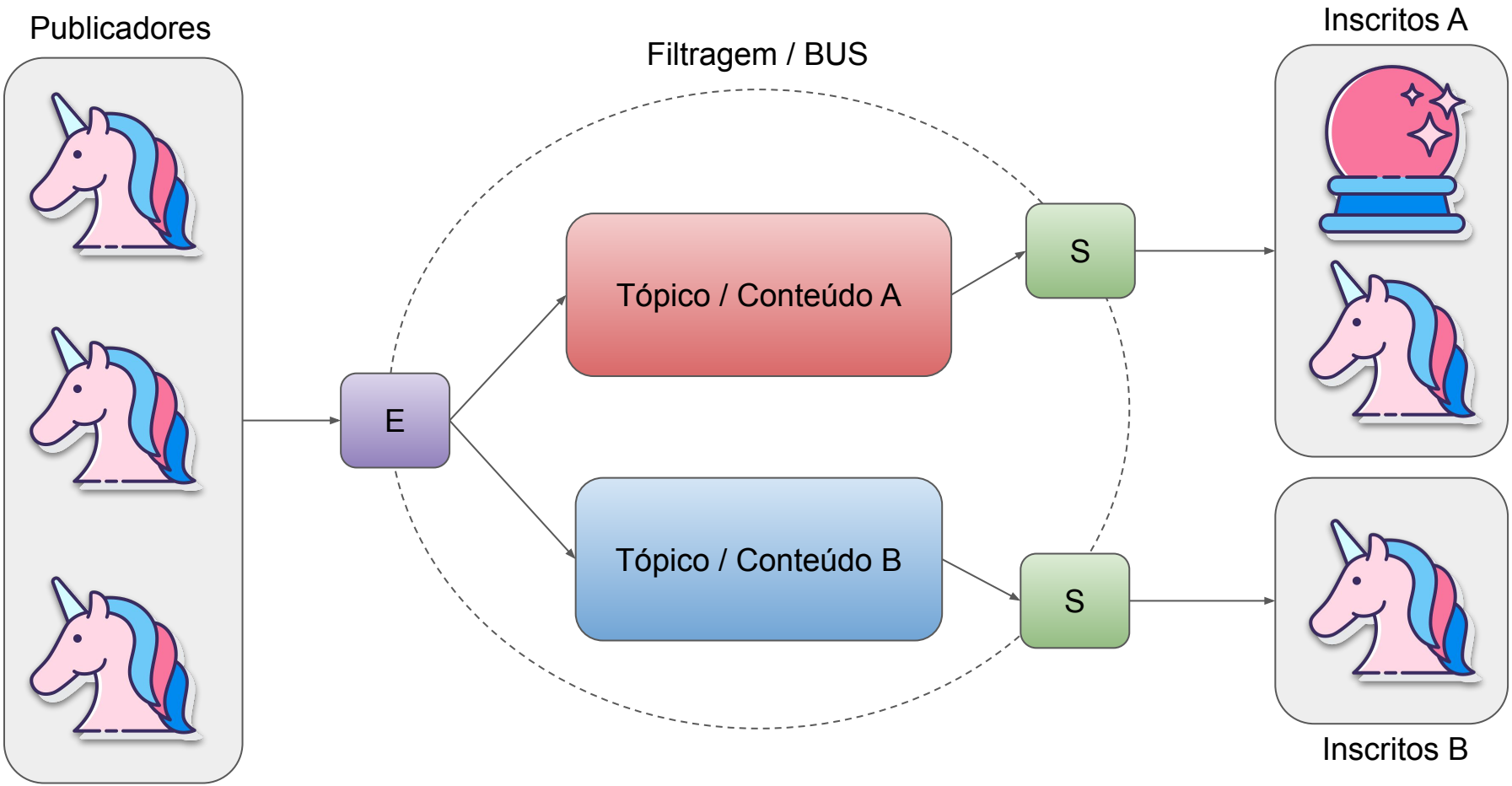


Paul explicou a Fausto que o demônio tinha muitos mensageiros e que todos eles gostavam de contar sobre pegadinhas que faziam o tempo todo.

“Meu lorde disse que eu teria que ser um gateway das mensagens do sub mundo”

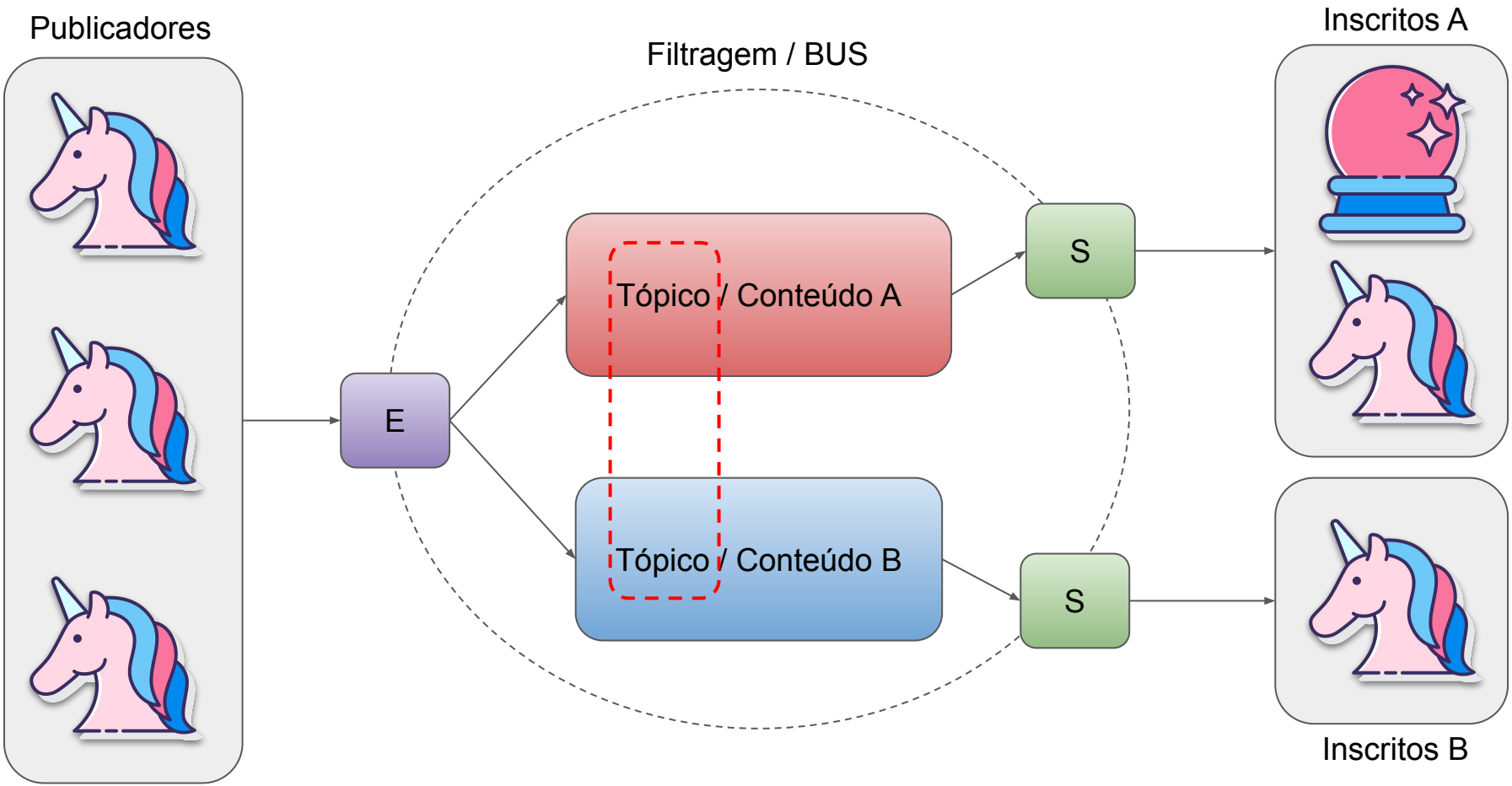
https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern#Message_filtering

Live de Python



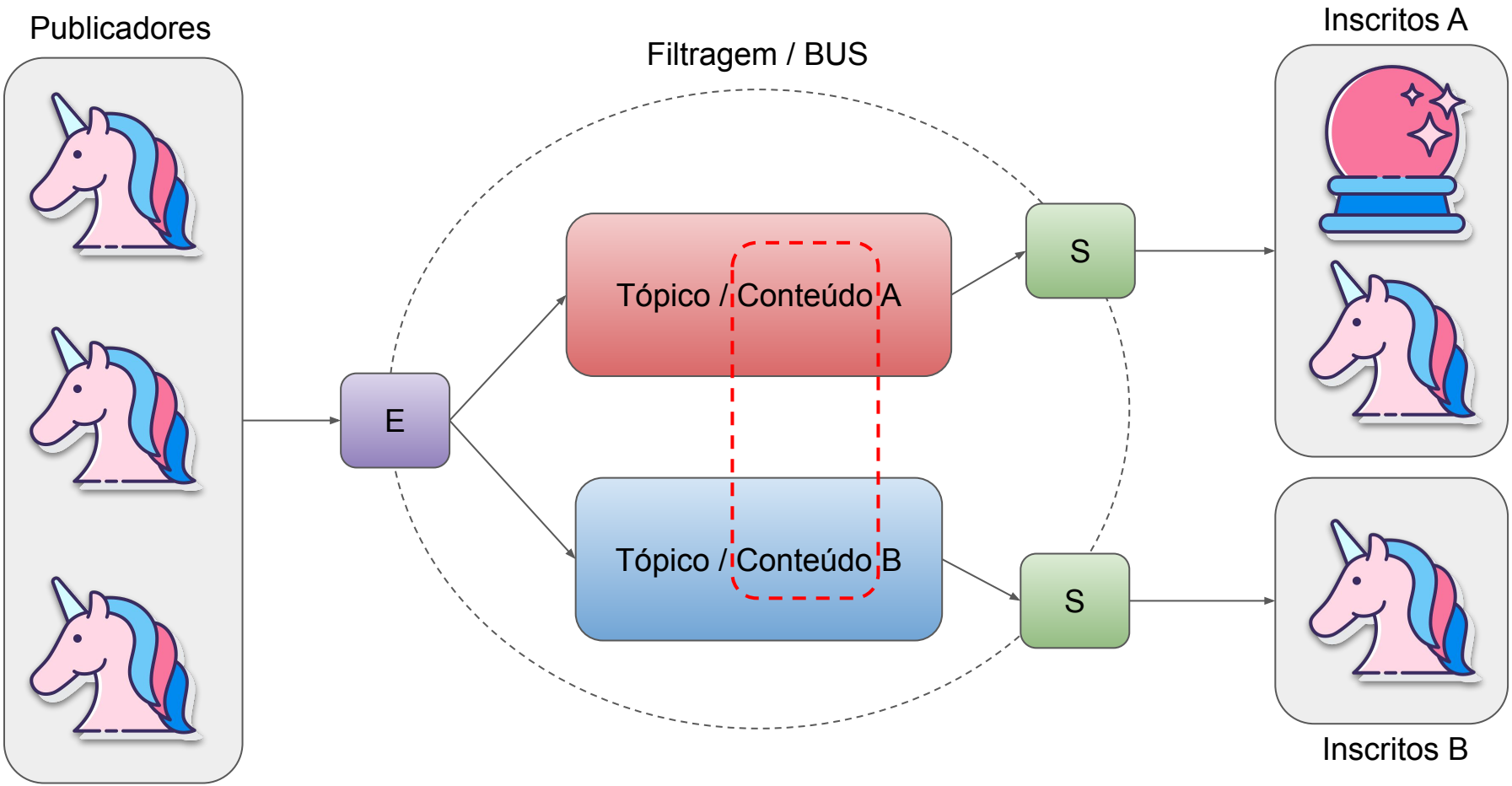
https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern#Message_filtering

Live de Python



https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern#Message_filtering

Live de Python



https://en.wikipedia.org/wiki/Publish%E2%80%93subscribe_pattern#Message_filtering

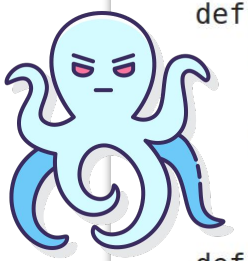
Live de Python

```python
class Publicador:
    def __init__(self, topico, pub_sub):
        self.topico = topico
        self.mensagens = []
        self.pub = pub_sub

    def publicar(self, mensagem):
        msg = {'topico': self.topico, 'mensagem': mensagem}
        self.pub.receber_mensagem(msg)
```

```python
class Inscrito:
    def __init__(self, nome):
        self.nome = nome

    def atualizar(self, topico, mensagem):
        print(f"|{topico}|\t{self.nome} recebeu: '{mensagem}'")
```

Inscritos B

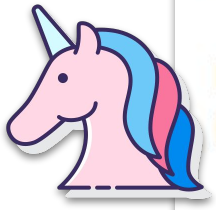
Live de Python

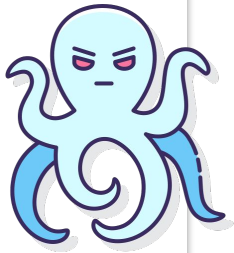


```python
class PubSub:
    def __init__(self):
        self.inscritos_por_topico: Dict[str, Set] = {}
        self.fila_de_mensagens: List[Dict[str, str]] = []

    def adicionar_inscrito(self, topico, inscrito):
        if topico in self.inscritos_por_topico:
            self.inscritos_por_topico[topico].add(inscrito)
        else:
            self.inscritos_por_topico[topico] = {inscrito}

    def receber_mensagem(self, mensagem: Dict[str, str]):
        """{'topico': xpto, 'mensagem': xpto}"""
        self.fila_de_mensagens.append(mensagem)

    def _enviar_mensagens_por_topico(self, topico, mensagem):
        for incrito in self.inscritos_por_topico[topico]:
            incrito.atualizar(topico, mensagem)

    def broadcast(self):
        for msg in self.fila_de_mensagens:
            self._enviar_mensagens_por_topico(msg['topico'], msg['mensagem'])

        self.fila_de_mensagens = []
```

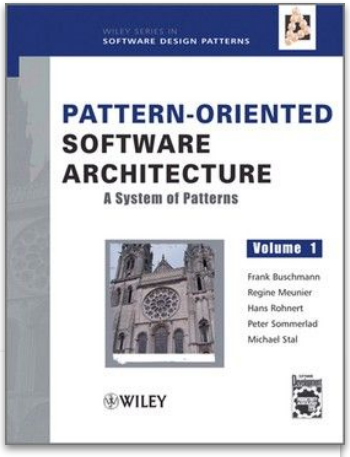
Live de Python

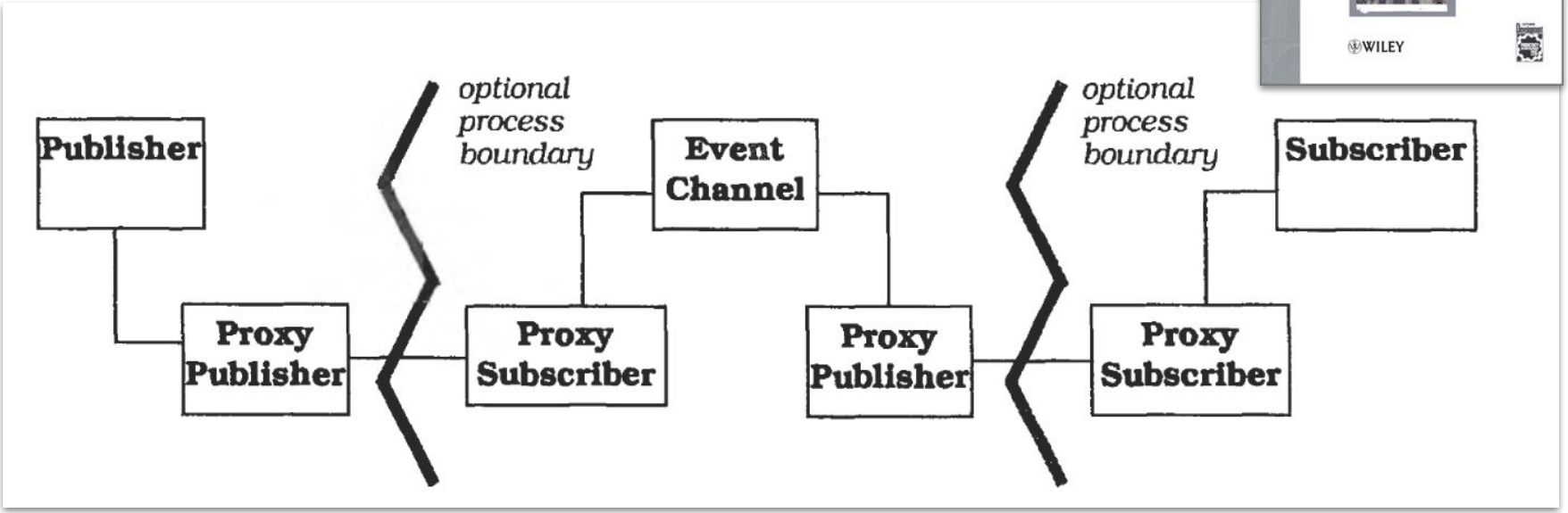
```
eduardo = Inscrito('Eduardo')
maria = Inscrito('Maria')
jose = Inscrito('José')

bus = PubSub()

blog_1 = Publicador('Blog do zé', bus)
blog_2 = Publicador('PSF', bus)

bus.adicionar_inscrito('PSF', eduardo)
bus.adicionar_inscrito('PSF', maria)
bus.adicionar_inscrito('Blog do zé', jose)

blog_1.publicar('Zé foi a feira hoje')
blog_2.publicar('Zé, membro da PSF foi a feira hoje.')


bus.broadcast()
```





Publisher-Subscriber Pattern





I was once asked in an interview, *"what is the difference between the Observer pattern and Pub-Sub pattern?"* I immediately figured that Pub-Sub means **'Publisher-Subscriber'** and I then vividly recalled from the book "Head first Design Pattern":

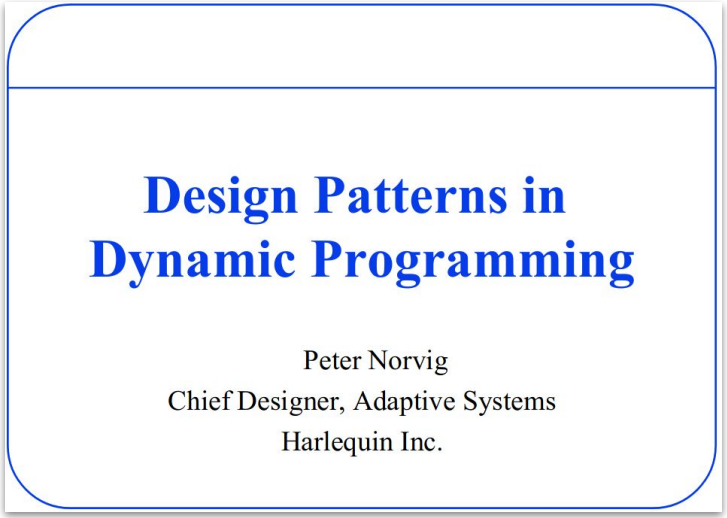
> Publishers + Subscribers = Observer Pattern

"I got it, I got it. You can't trick me Mr."- I thought.

I am adding this irrelevant GIF, like everybody do

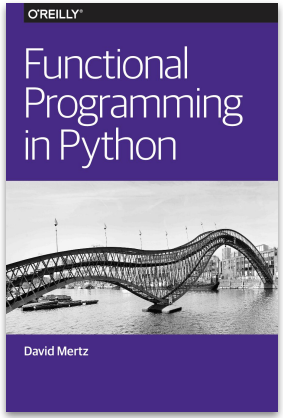
https://hackernoon.com/observer-vs-pub-sub-pattern-50d3b27f838c

Pontos de atenção



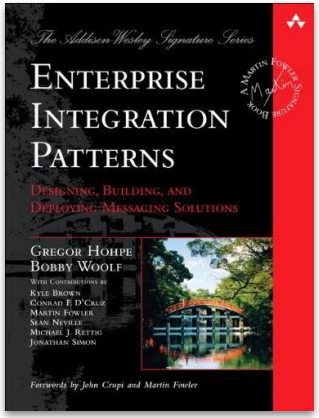
http://www.norvig.com/design-patterns/design-patterns.pdf

wiki.c2.com/?AreDesignPatternsMissingLanguageFeatures

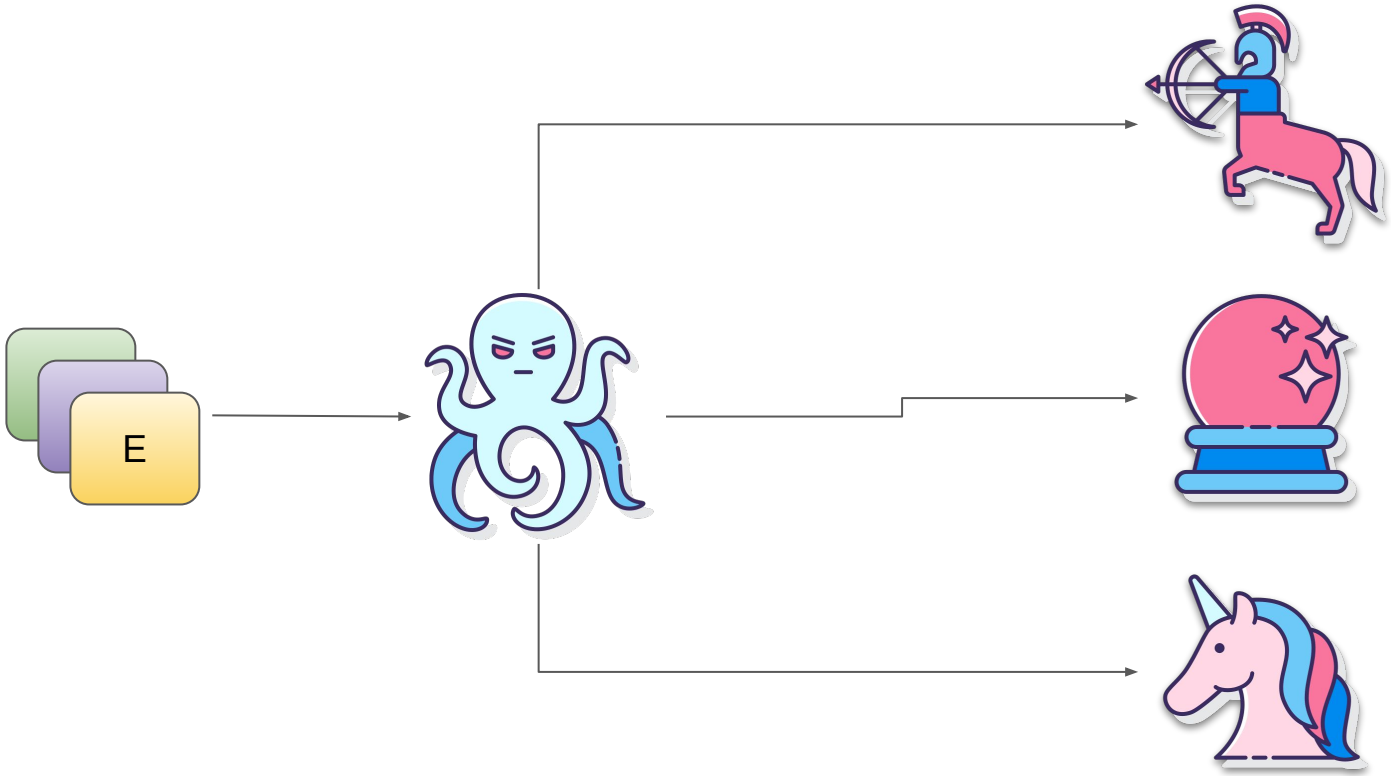


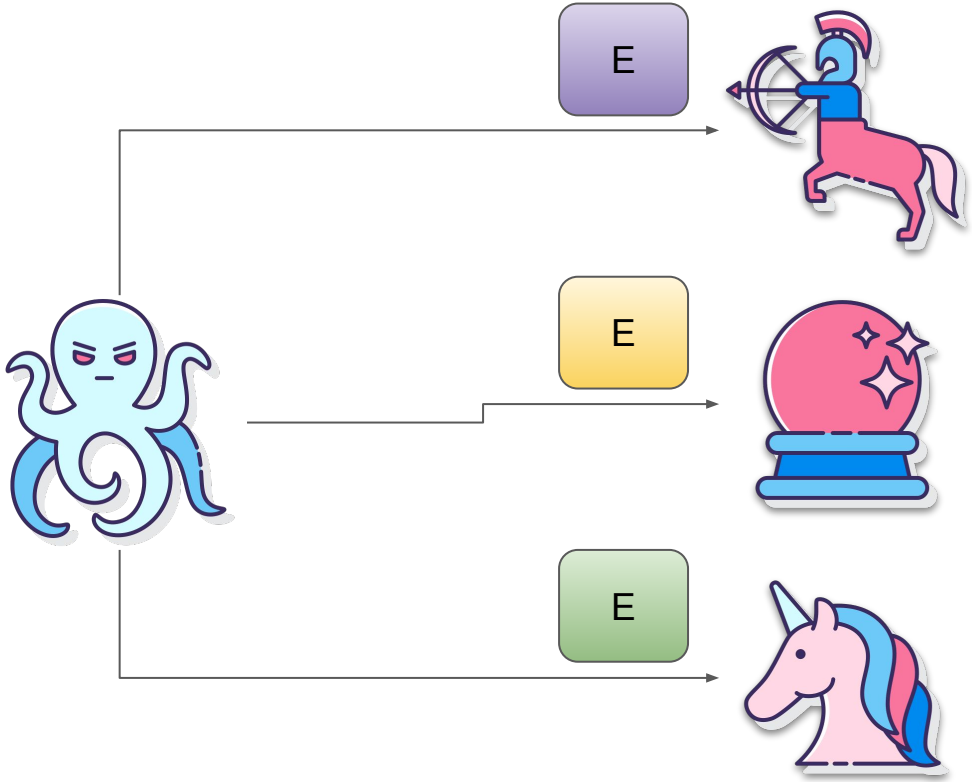
Multimethods / Dispatcher

PubSub Channel



Live de Python


E


E
E
E

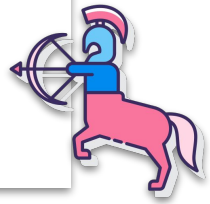
```
class Cor:
    ...


class Verde(Cor):
    ...

class Roxo(Cor):
    ...

class Amarelo(Cor):
    ...
```

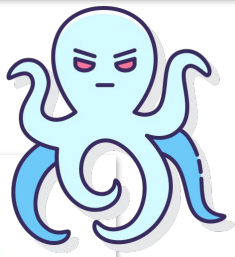
E

E

E

```
@paul.register(Verde)
def entregar_unicornio(evento):
    return 'Enviado para unicório'


@paul.register(Amarelo)
def entregar_centauro(evento):
    return 'Enviado para centauro'


@paul.register(Roxo)
def entregar_fausto(evento):
    return 'Enviado para Fausto'
```





```
@singledispatch
def paul(evento):
    return 'default'
```

Live de Python