



**AKADEMIA GÓRNICZO-HUTNICZA  
IM. STANISŁAWA STASZICA W KRAKOWIE**

# **MOM/RabbitMQ**

## Modele komunikacji

- Komunikacja synchroniczna
- Komunikacja asynchroniczna

## Modele komunikacji

- Komunikacja synchroniczna
  - Obie strony uczestniczące muszą być aktywne
  - Wywołania blokujące
- Komunikacja asynchroniczna
  - Obie strony uczestniczące nie muszą być aktywne jednocześnie
  - Wywołania nieblokujące
  - Potwierdzenia odbioru (opcjonalnie)

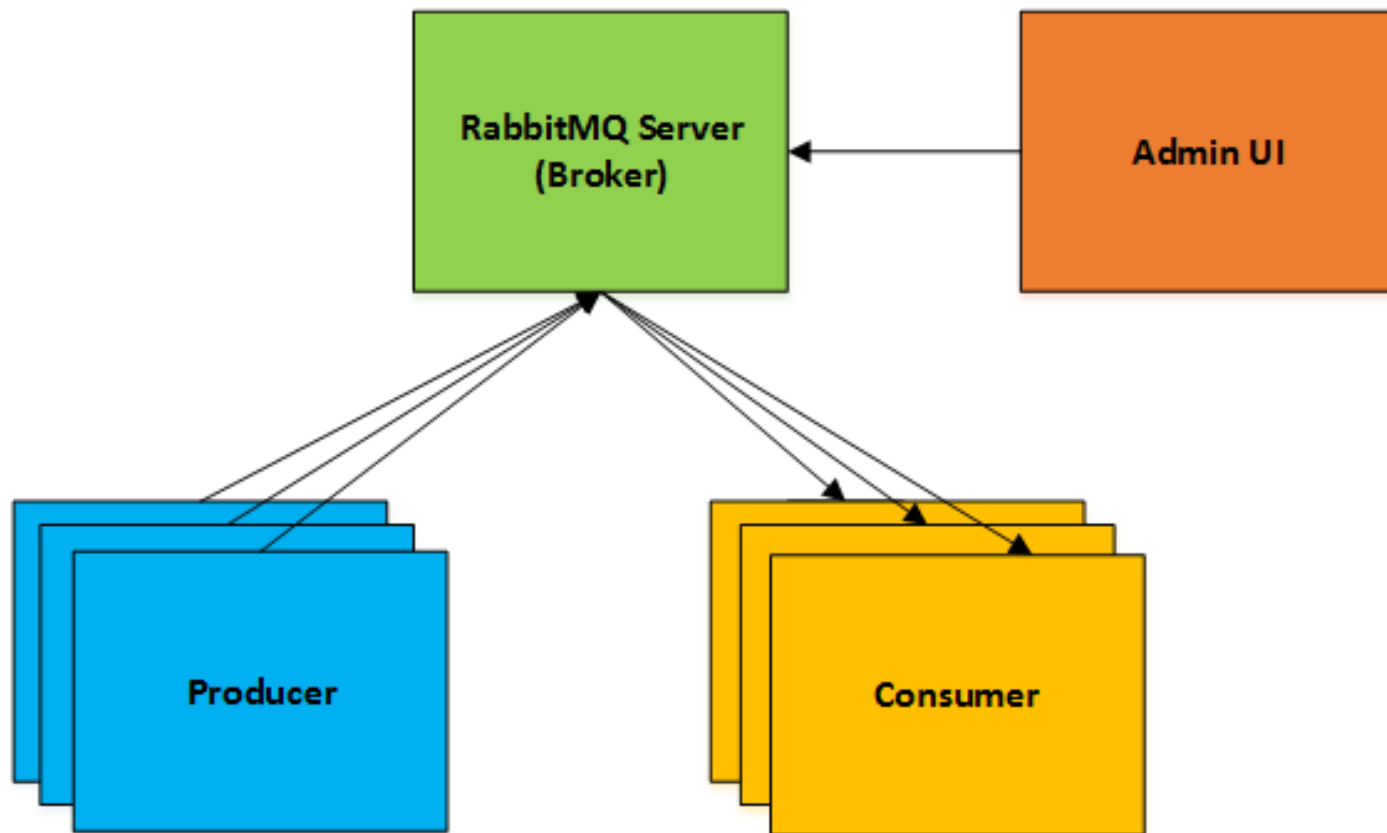
# Wiadomości

- Alternatywa dla wywołań metod
  - Format wiadomości zamiast interfejsu
  - Ukierunkowane na zdarzenia
  - Brak sztywnych zależności czasowych
- Luźne powiązania komponentów
- Message Oriented Middleware
  - Warstwa pośrednia dostarczająca mechanizmów obsługi wiadomości

# RabbitMQ

- Framework do obsługi wiadomości
- Główne cechy:
  - Mechanizmy wyboru ścieżek do przesyłu wiadomości (routing)
  - Mechanizmy zapewnienia niezawodności (potwierdzenia, ponowne wysłanie)
  - Wsparcie dla różnych protokołów
  - Wsparcie dla wielu języków programowania
  - Interfejs do zarządzania
  - Pluginy

# RabbitMQ – Elementy składowe



# Hello World

- Producent
  - wysyła wiadomość do kolejki
- Konsument
  - odbiera wiadomości z kolejki



## Połączenie (Producer / Consumer)

```
ConnectionFactory factory = new ConnectionFactory();  
factory.setHost("localhost");
```

```
Connection connection = factory.newConnection();  
Channel channel = connection.createChannel();
```

```
(...)
```

```
// don't close while listening (consumer)  
channel.close();  
connection.close();
```



## Wysyłanie wiadomości (Producer)

```
String QUEUE_NAME = "queue1";  
channel.queueDeclare(QUEUE_NAME, false, false, false, null);  
  
String message = "Hello World!";  
channel.basicPublish("", QUEUE_NAME, null, message.getBytes());  
  
System.out.println("Sent: " + message);
```

## Odbieranie wiadomości (Consumer)

```
String QUEUE_NAME = "queue1";  
channel.queueDeclare(QUEUE_NAME, false, false, false, null);  
  
Consumer consumer = new DefaultConsumer(channel) {  
    @Override  
    public void handleDelivery(String consumerTag,  
        Envelope envelope, AMQP.BasicProperties properties,  
        byte[] body) throws IOException {  
        String message = new String(body, "UTF-8");  
        System.out.println("Received: " + message);  
    }  
};  
  
channel.basicConsume(QUEUE_NAME, true, consumer);
```

## Uruchomienie przykładu

- Należy wystartować serwer RabbitMQ
  - Menu start -> RabbitMQ Service - start
- Kod i biblioteki dostępne na UPEL
- Uruchomić konsumenta **Z1\_Consumer**
- Uruchomić producenta **Z1\_Producer**
- Przesłać wiadomość

# RabbitMQ

- Konsola administracyjna (web)
  - Uruchomić konsolę RabbitMQ Command Prompt (z menu start)
  - Wpisać:  
`rabbitmq-plugins enable rabbitmq_management`
  - Konsola dostępna pod adresem:  
<http://localhost:15672/>  
user: guest, password: guest
- Tutorial  
<https://www.rabbitmq.com/getstarted.html>

# Mechanizmy obsługi kolejek

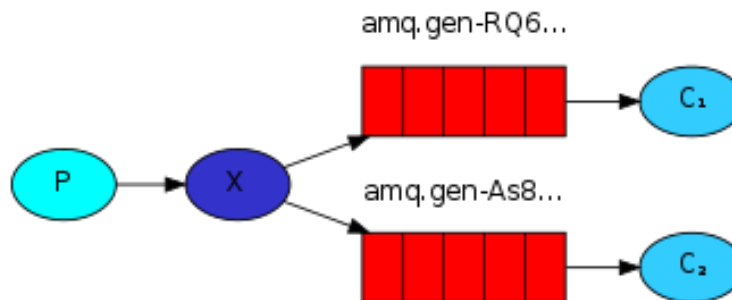
- Potwierdzenia
  - Potwierdzenie po otrzymaniu wiadomości
  - Potwierdzenie po przetworzeniu wiadomości
- Dystrybucja wiadomości do wielu konsumentów
  - Domyślnie round-robin
  - Możemy uzyskać load-balancing
- Trwałość
  - Możliwość zachowania wiadomości przy restarcie serwera

## **Zadanie 1 (2 pkt)**

- Instrukcja (niezawodność, load-balancing)

# Routing

- Exchange
  - Producent nie wysyła wiadomości bezpośrednio do kolejki, lecz do Exchange
  - Exchange decyduje gdzie wysłać wiadomość
  - Poprzednio korzystaliśmy z domyślnego Exchange (Nameless)



## Routing

- Exchange decyduje, do których kolejek wysłać wiadomość (może wysłać kopię wiadomości do więcej niż jednej kolejki)
- Wiadomości z kolejek trafiają do konsumentów (jedna wiadomość z kolejki trafia do jednego konsumenta, nawet jeśli do kolejki zapisanych jest ich więcej)



# Routing

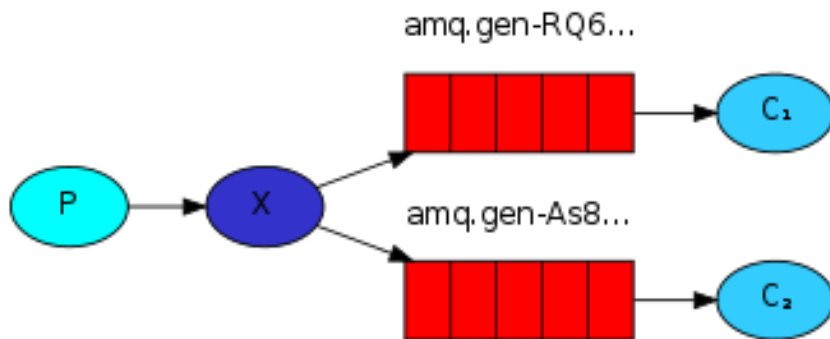
- Typy Exchange:
  - Fanout (do wszystkich zapisanych)
  - Direct (bezpośrednio wg. klucza)
  - Topic (dopasowanie wg. wzorca)
  - Headers
- Uwaga:
  - Kolejki muszą zostać związane (bind) z danym Exchange, aby otrzymywać z niego wiadomości

## Routing Fanout

- Wiązanie kolejki z Exchange (model publish/subscribe)

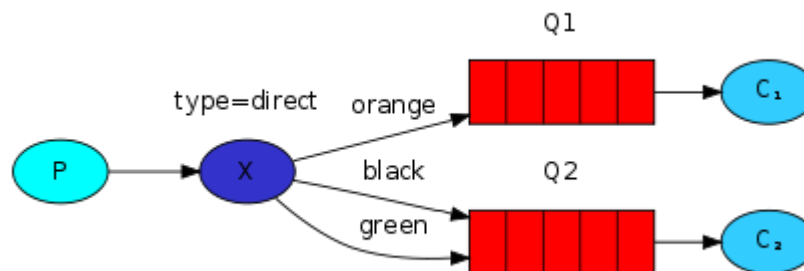
```
String queueName = channel.queueDeclare().getQueue();  
channel.queueBind(queueName, EXCHANGE_NAME, "");
```

- Każdy kto jest zapisany do danego Exchange dostaje wiadomości

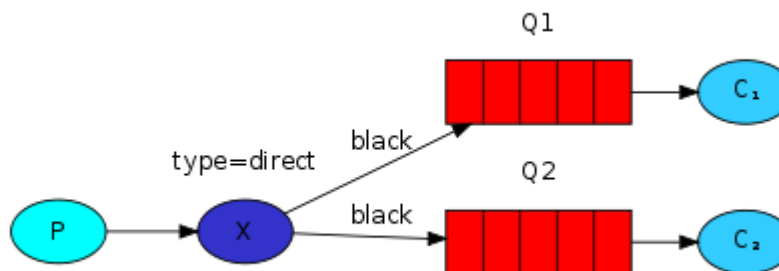


# Routing Direct

- Wiązanie kolejki z Exchange wg klucza  
`channel.queueBind(queueName, EXCHANGE_NAME, routingKey);`
- Możliwe wiele kluczy dla danej kolejki

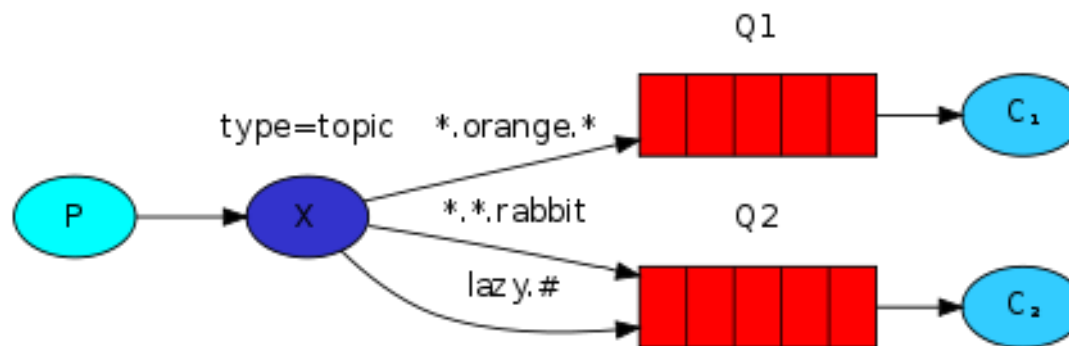


- Możliwe wiele kolejek z tym samym kluczem

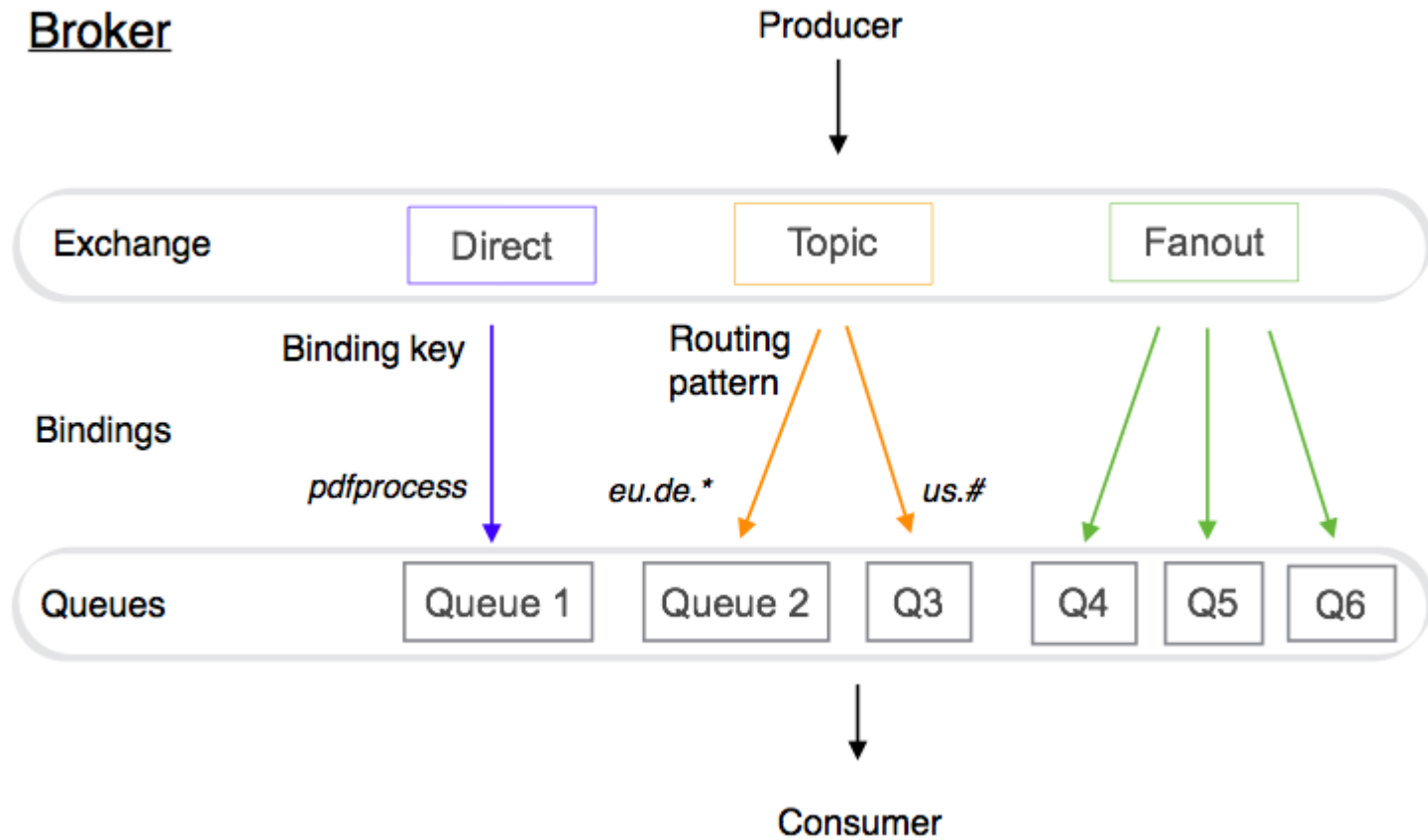


## Routing Topic

- Dopasowanie do wzorca
  - np. blue.fast.sedan
  - \* to dokładnie jedno dowolne słowo
  - # to zero lub więcej dowolnych słów



# Routing



## Uruchomienie przykładu (Fanout)

- Uruchomić producenta **Z2\_Producer**
- Uruchomić dwóch konsumentów **Z2\_Consumer**
- Przesłać wiadomość
- Każdy konsument powinien dostać wiadomość

## **Zadanie 2 (2 pkt)**

- Instrukcja (Direct, Topic)

## Zadanie domowe

- Treść na UPEL
- Pytania do zadania domowego proszę kierować przez forum na UPEL
- Termin wysłania zadania:  
wtorek, 13 czerwca godz. 8:00  
(wszystkie grupy)