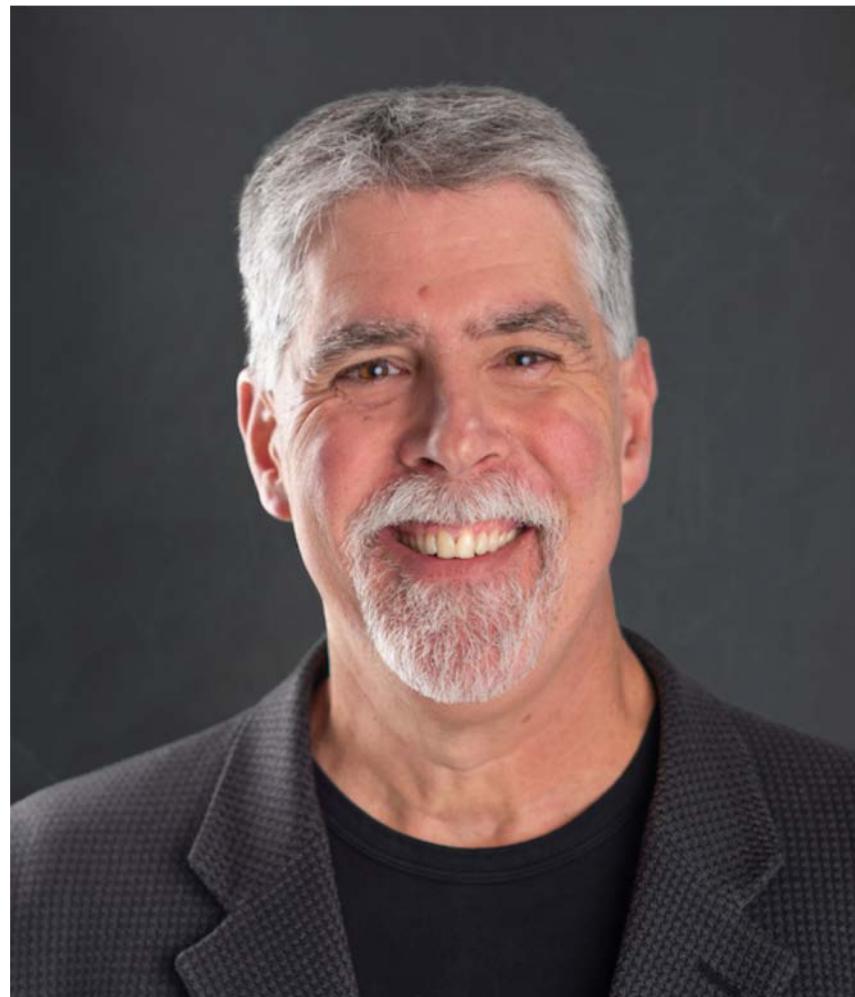




# Mastering Patterns in Event-Driven Architecture



**Mark Richards**

**Independent Consultant**

Hands-on Software Architect / Published Author

Founder, [DeveloperToArchitect.com](#)

<https://www.linkedin.com/in/markrichards3>

@markrichardssa

# the questions...

"how do I guarantee that no messages are lost when passing data from one service to another?"

"how do I increase performance and throughput while still maintaining message processing order?"

"how do I give some events a higher priority and still maintain responsiveness for standard events?"

"how do I send notification events to services without maintaining a persistent connection?"

"how do I handle varying request load and still maintain a consistent response time?"

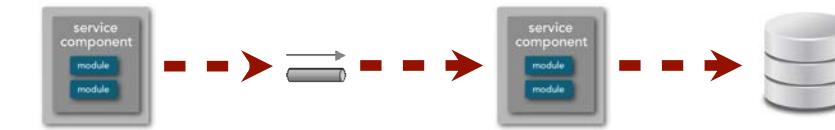
"how do I load balance heterogeneous events to ensure consistent response times?"

"how do I automatically fix error conditions when processing asynchronous events?"

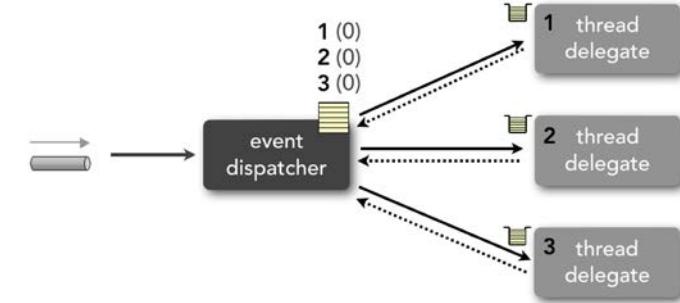
"how do I slow down message producers when the messaging system becomes overwhelmed?"

"how do I increase the throughput and capacity of events through the system?"

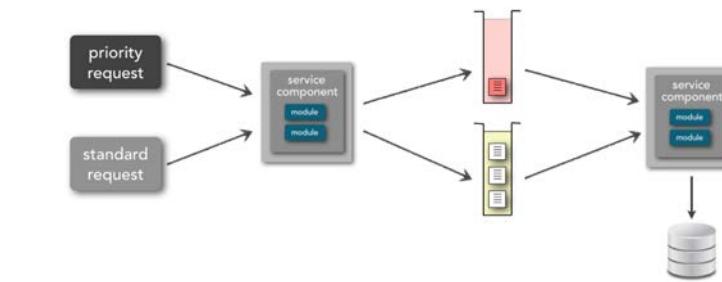
# the answers...



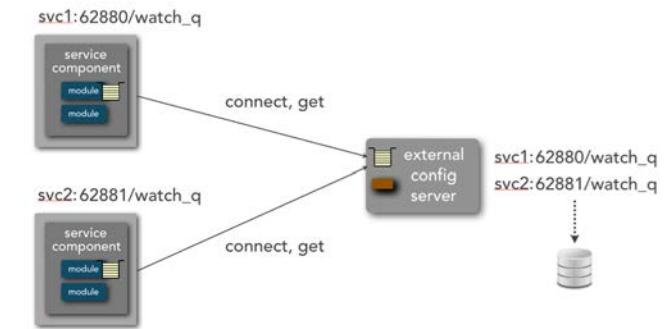
event forwarding pattern



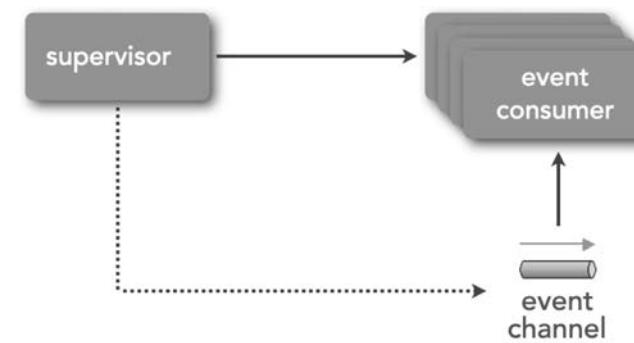
thread delegate pattern



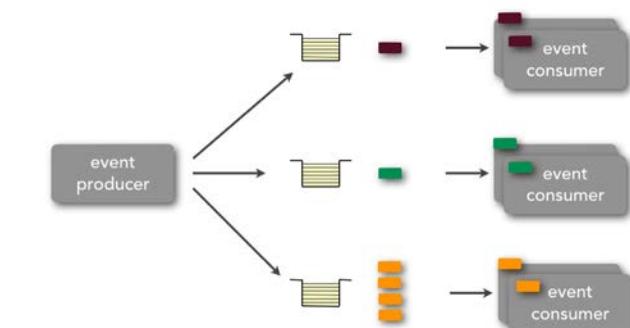
ambulance pattern



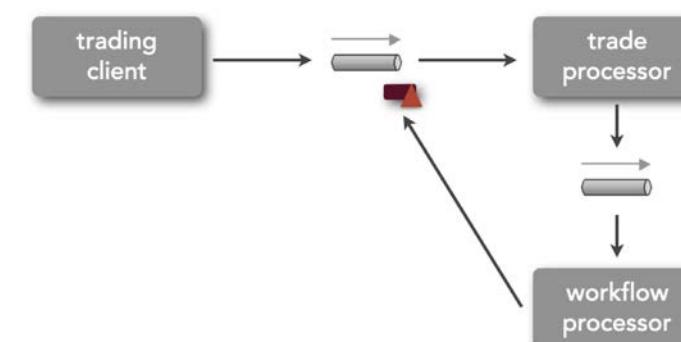
watch notification pattern



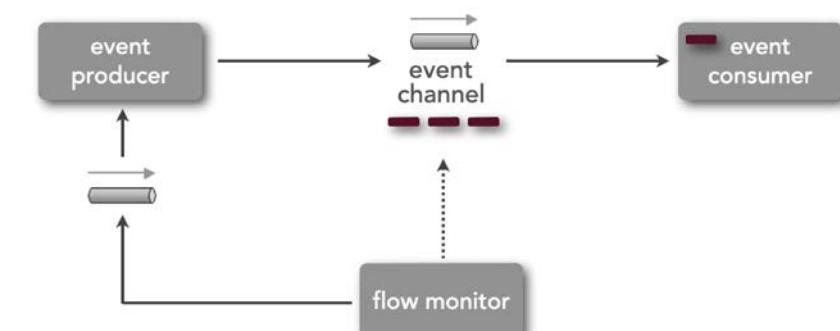
supervisor-consumer pattern



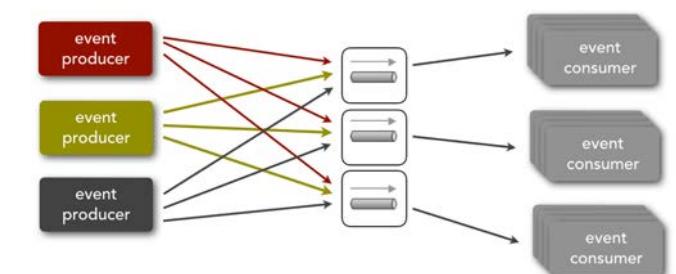
contextual queue pattern



workflow event pattern



producer control flow pattern



multi-broker pattern

# the tradeoffs...

the good things

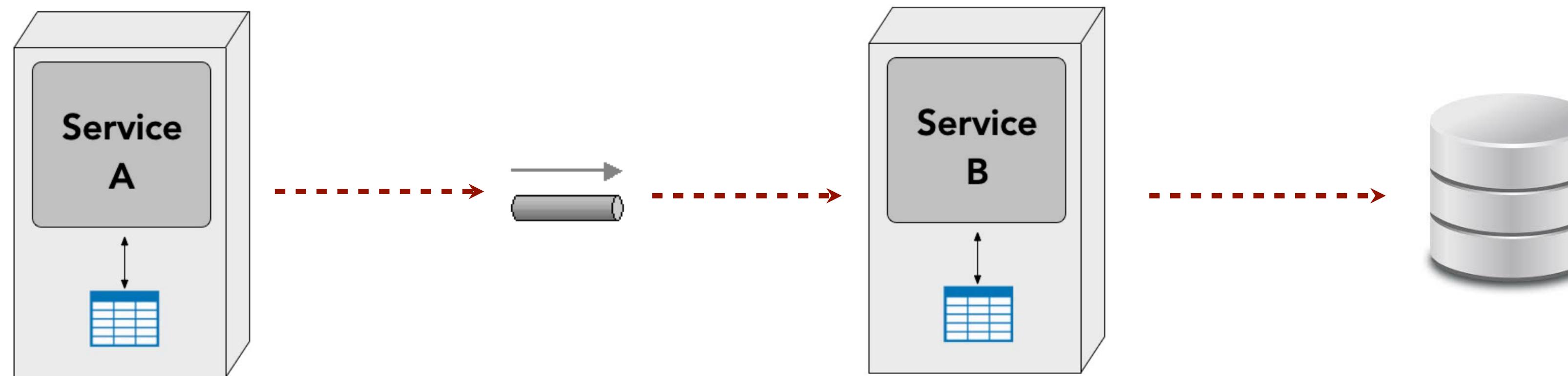


the bad things

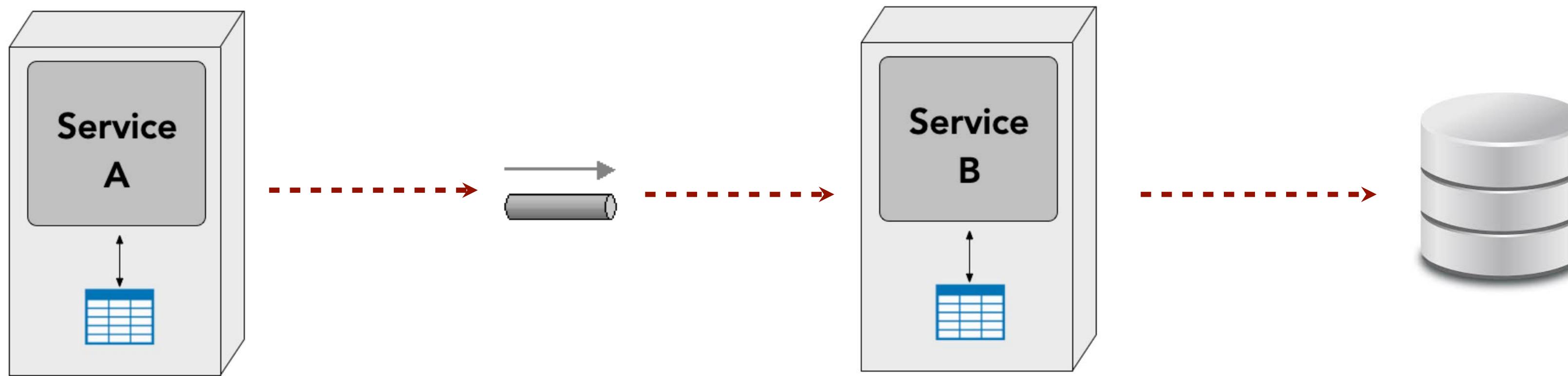
# Event Forwarding Pattern

# event forwarding pattern

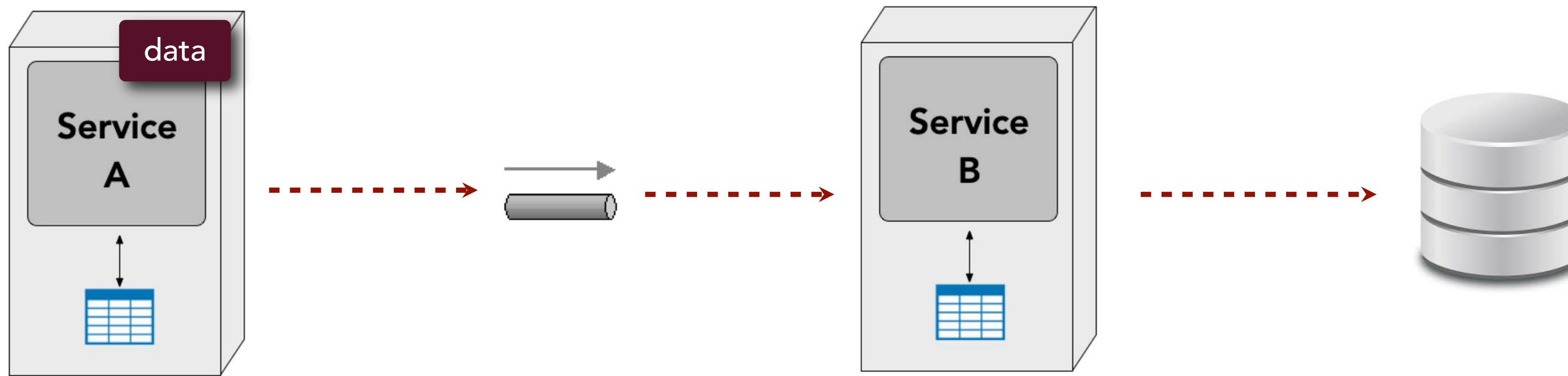
*“how do I guarantee that no messages are lost when passing data from one service to another?”*



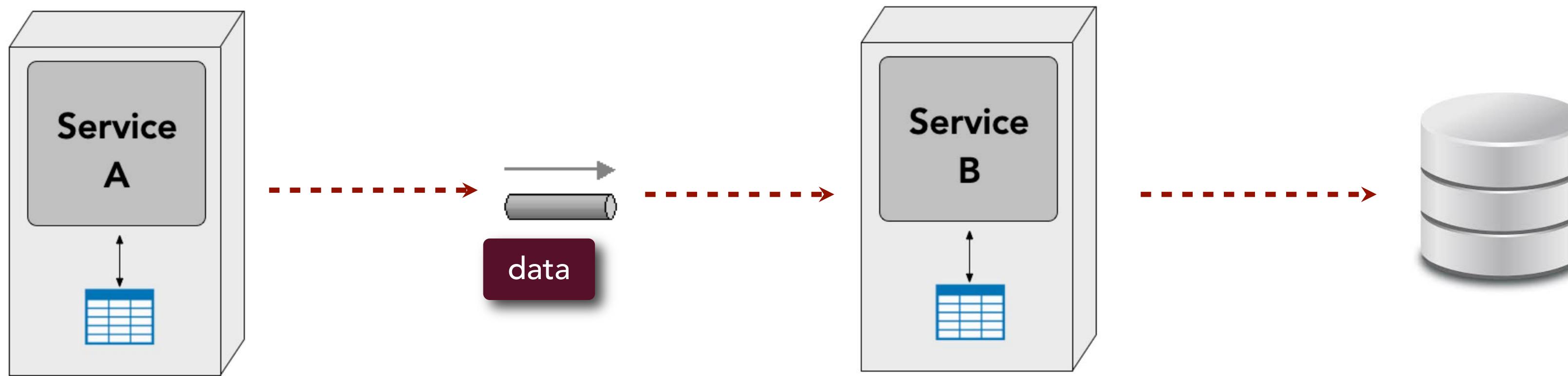
# event forwarding pattern



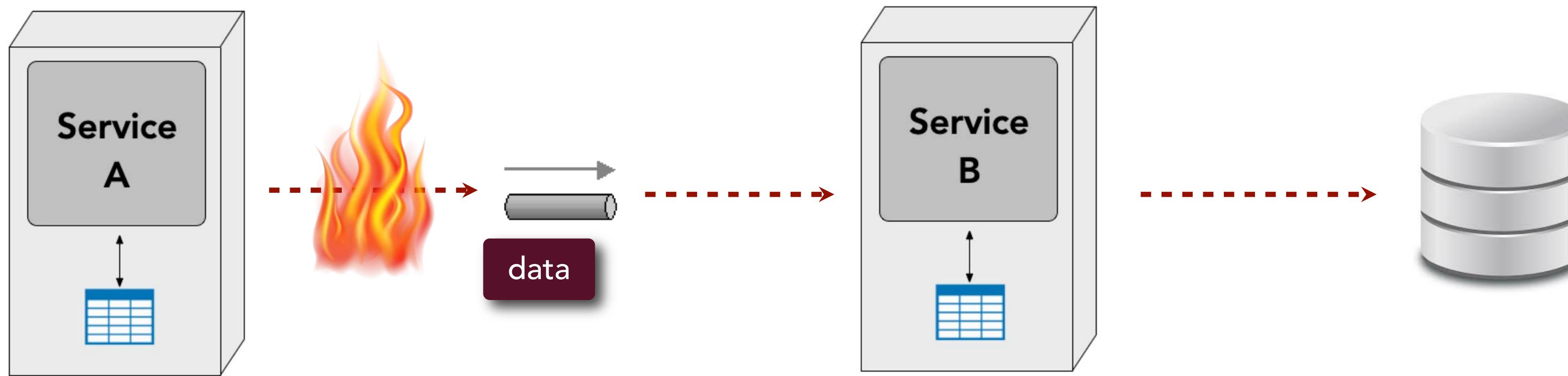
# event forwarding pattern



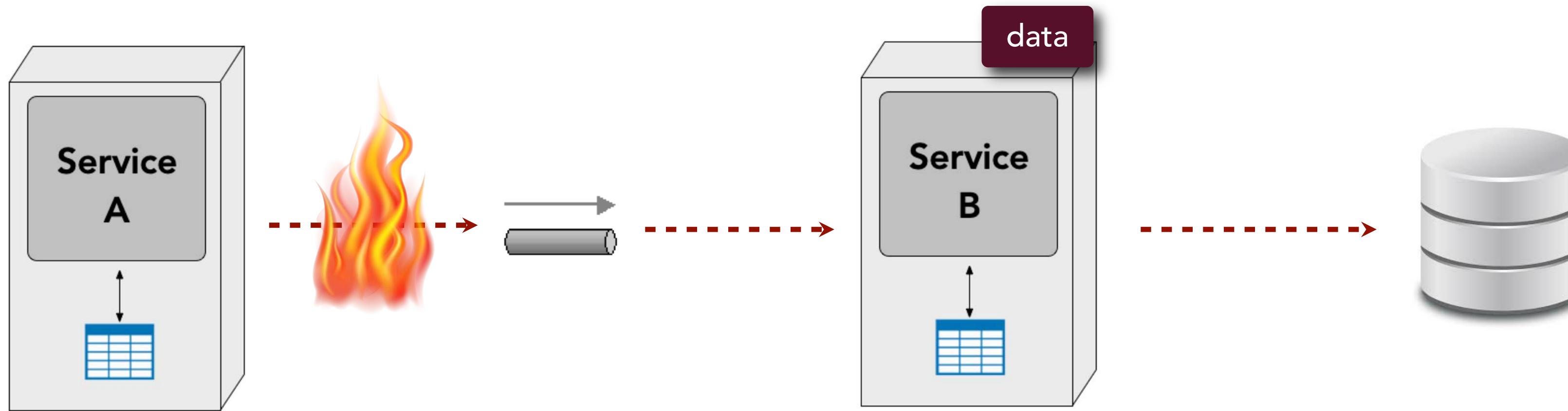
# event forwarding pattern



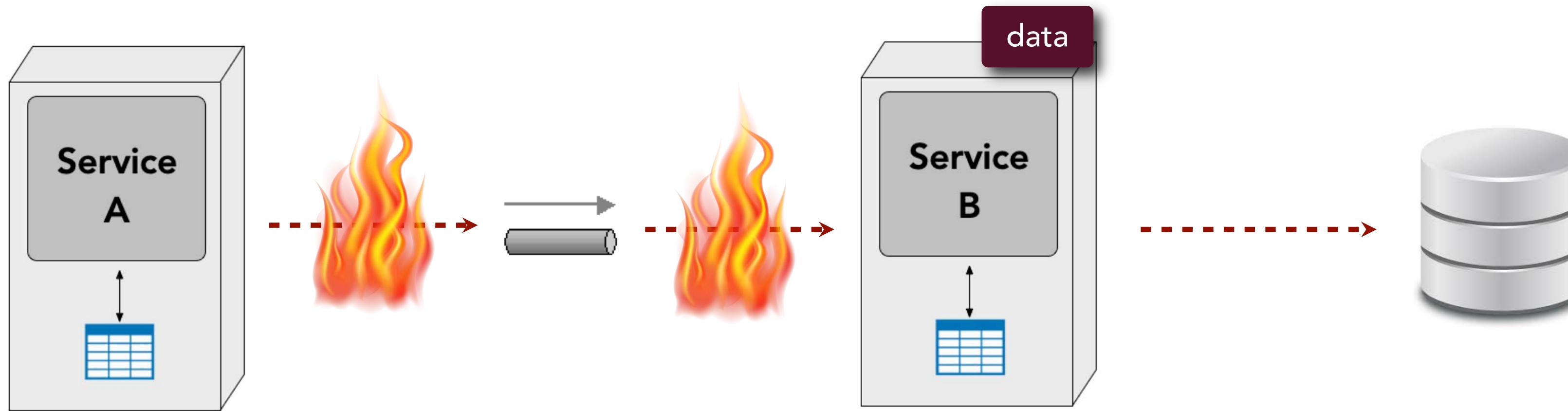
# event forwarding pattern



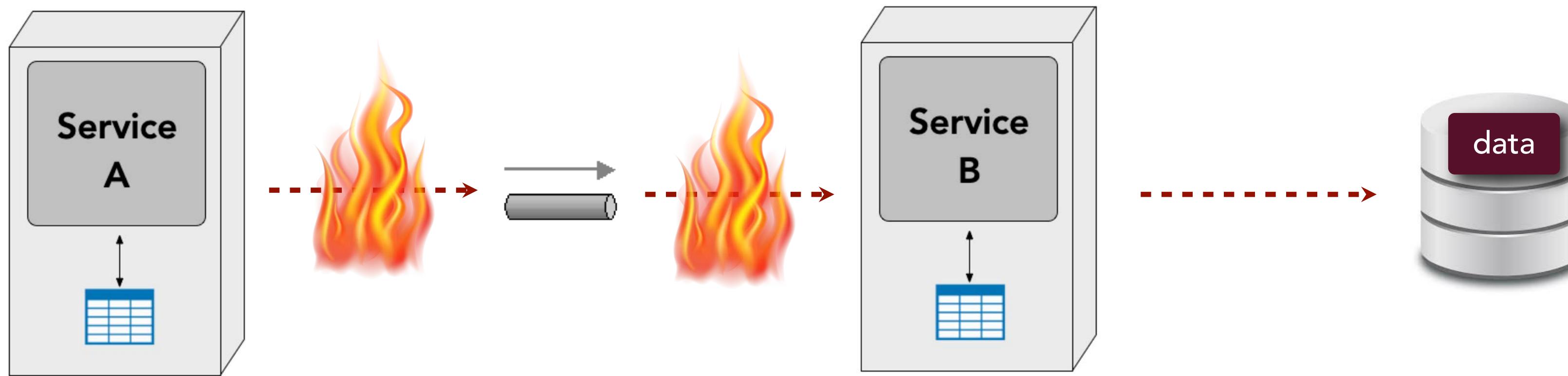
# event forwarding pattern



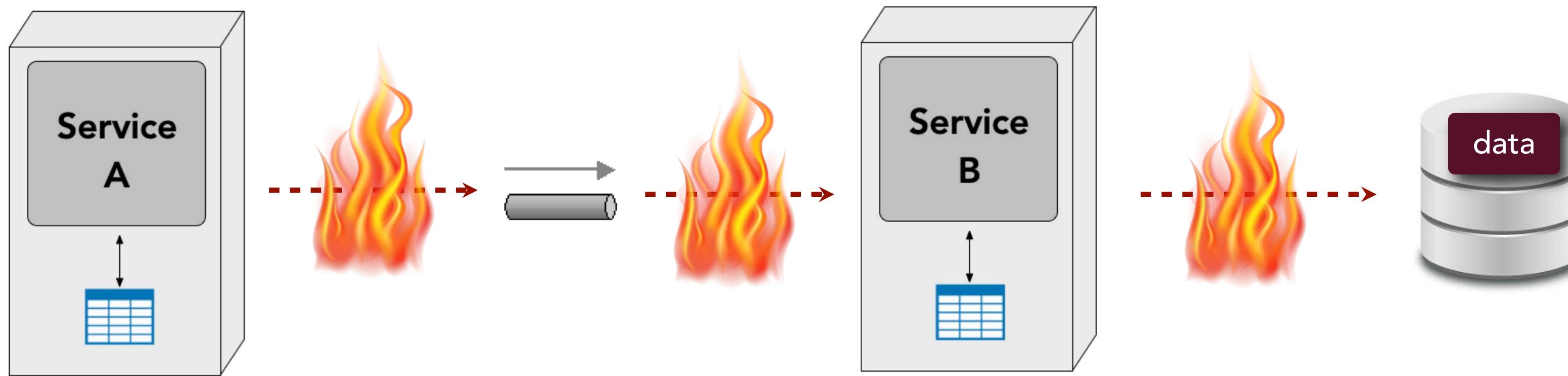
# event forwarding pattern



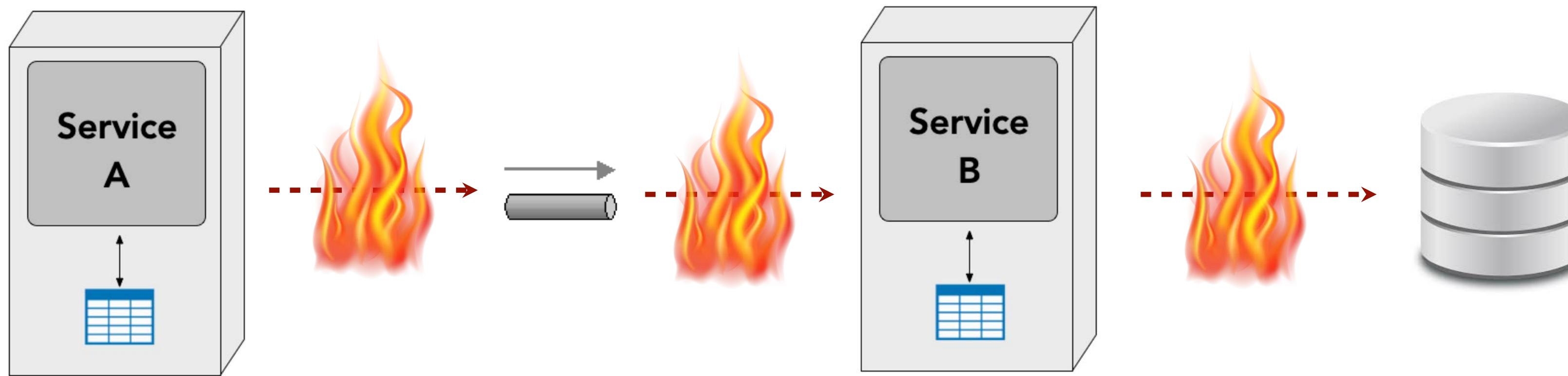
# event forwarding pattern



# event forwarding pattern



# event forwarding pattern



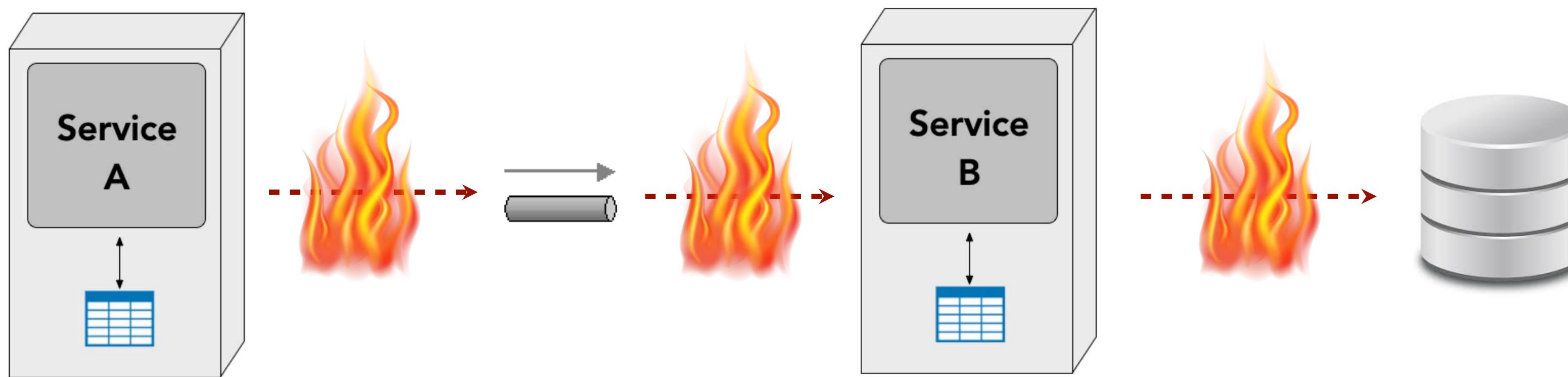
# event forwarding pattern



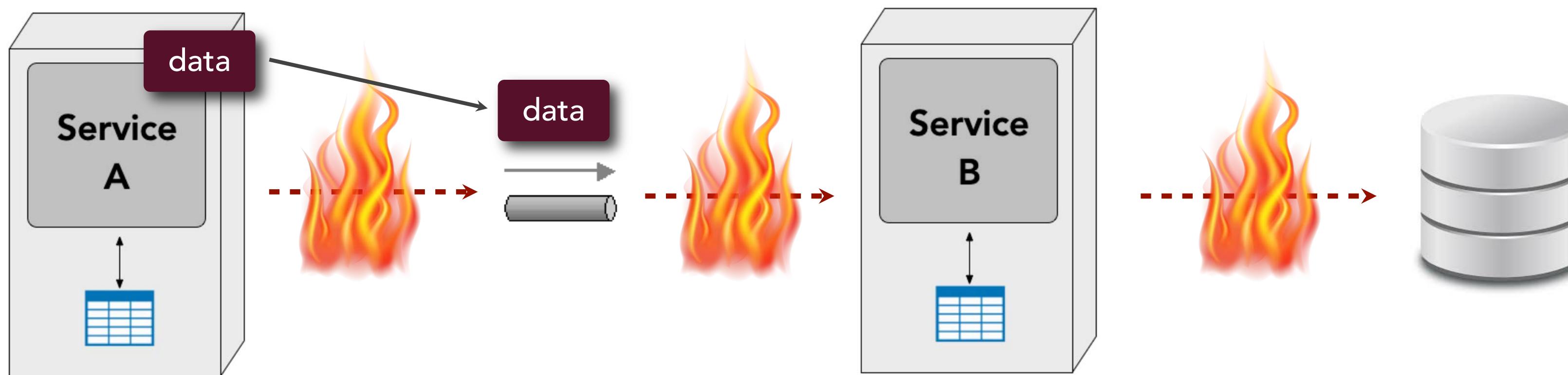
let's see the issue...

<https://github.com/wmr513/event-driven-patterns/tree/master/eventforwarding>

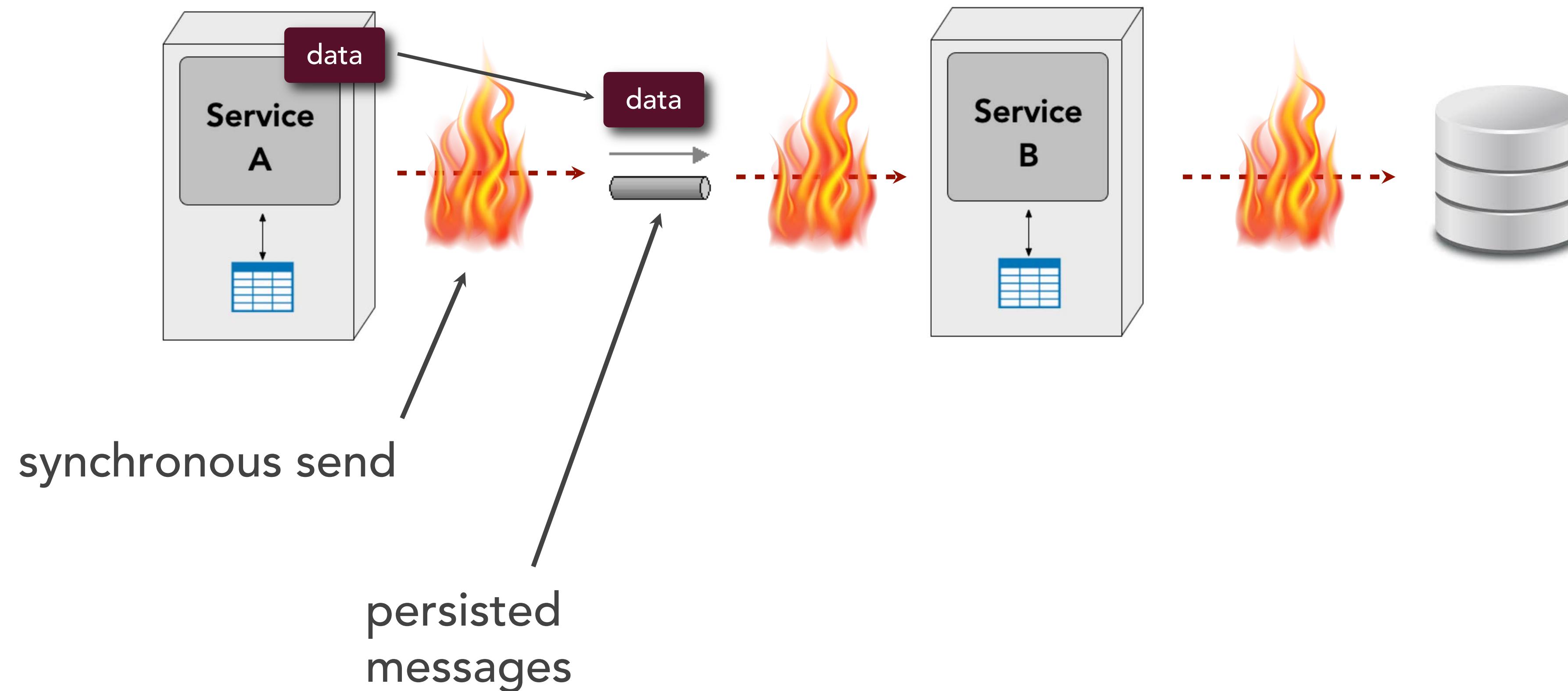
# event forwarding pattern



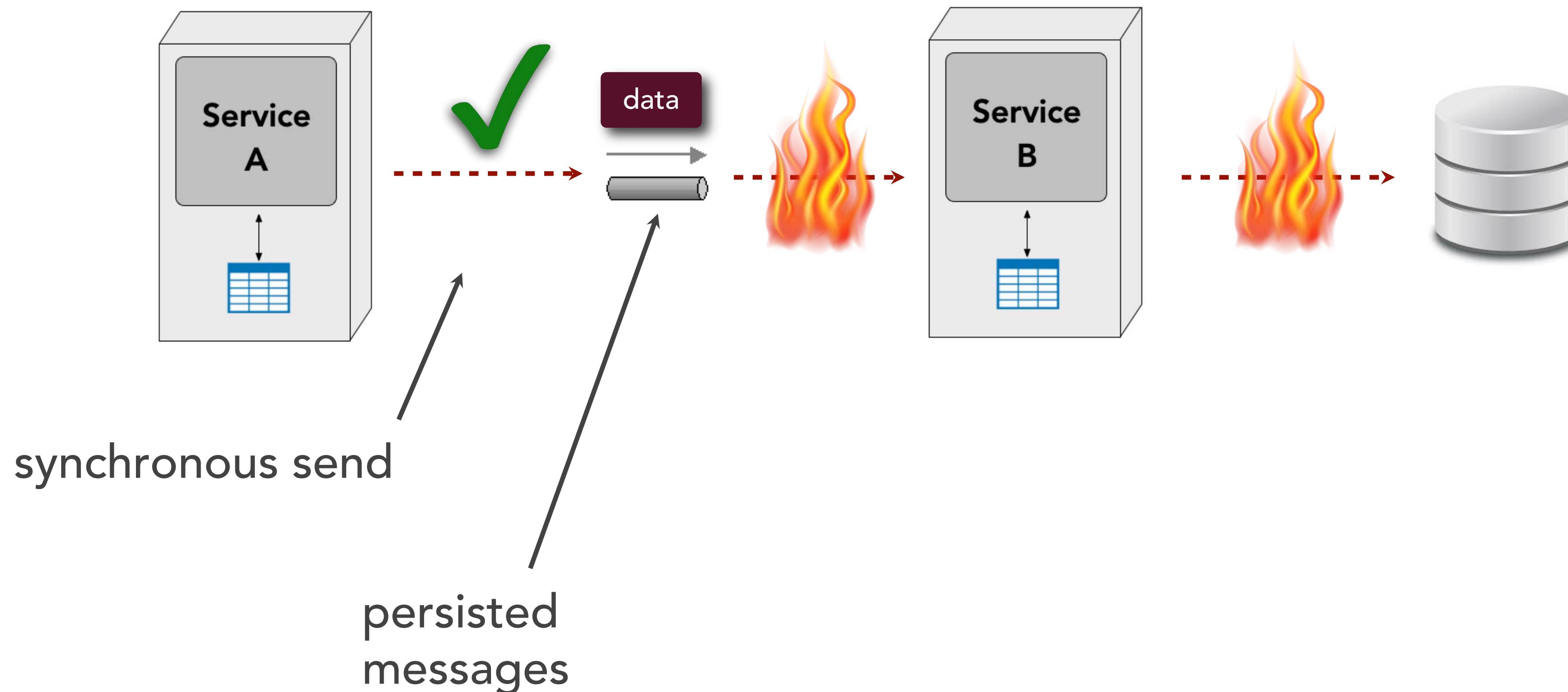
# event forwarding pattern



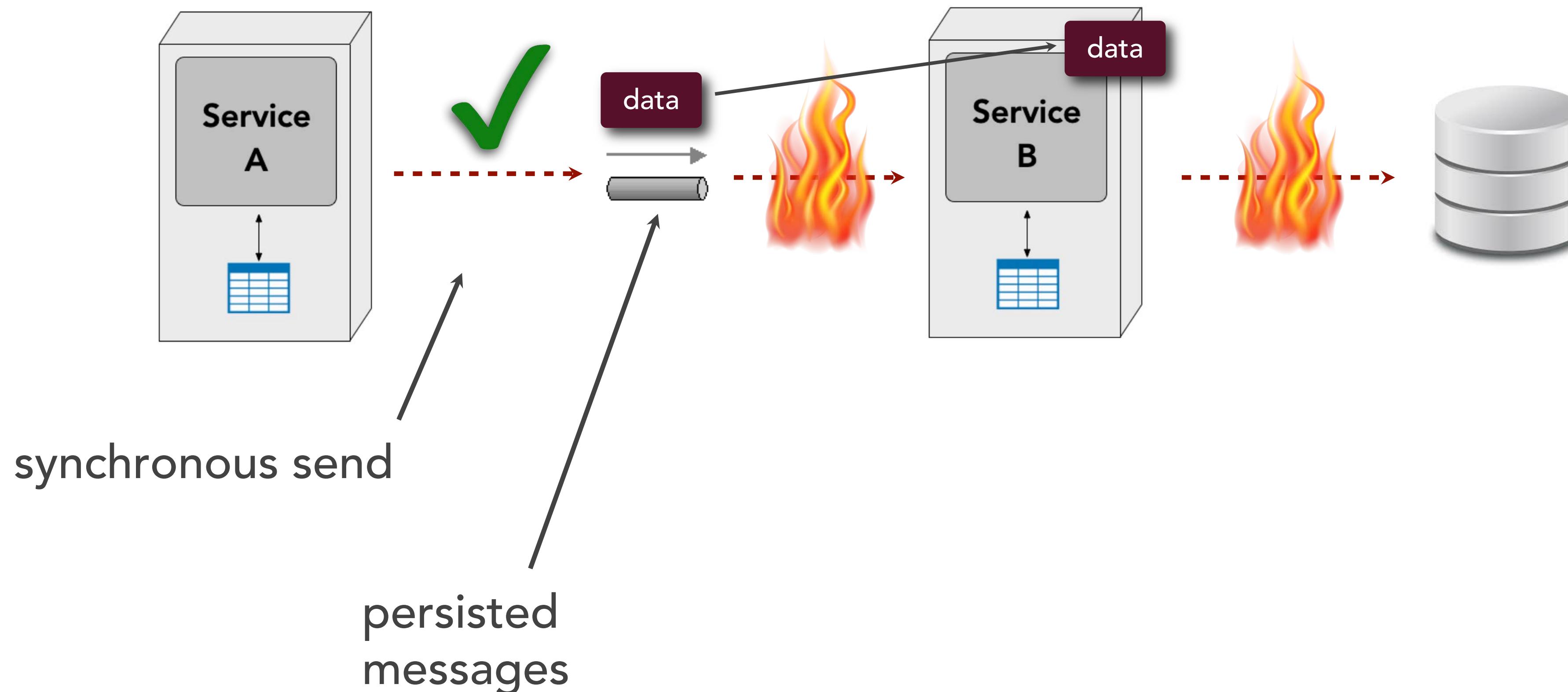
# event forwarding pattern



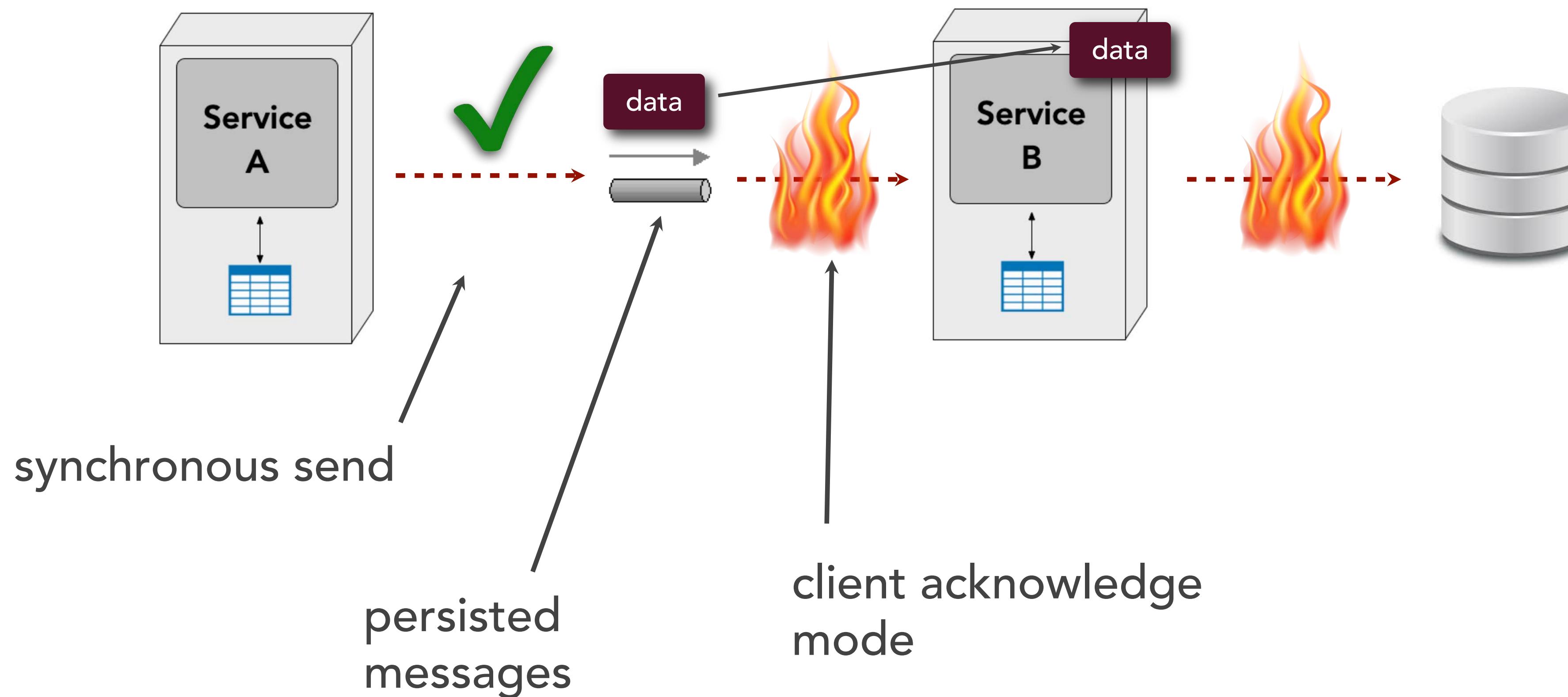
# event forwarding pattern



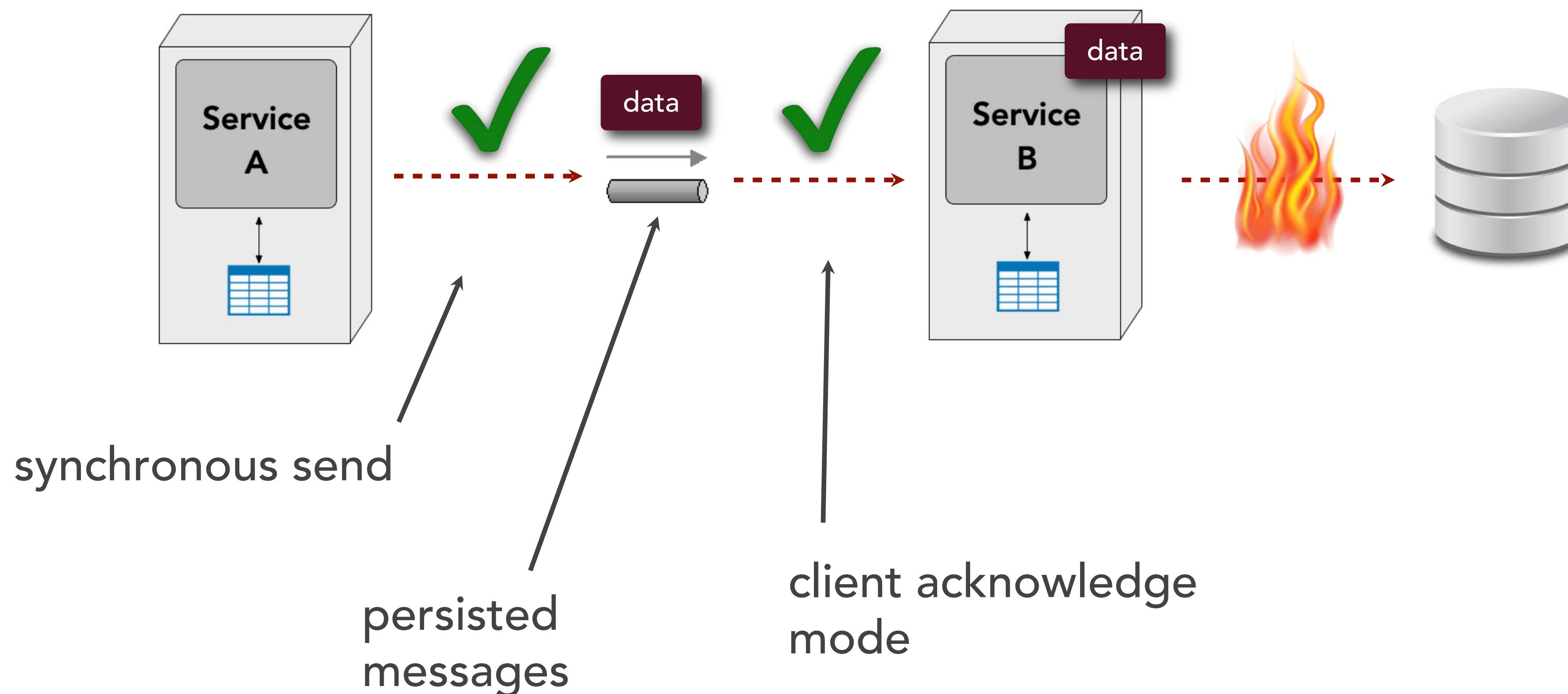
# event forwarding pattern



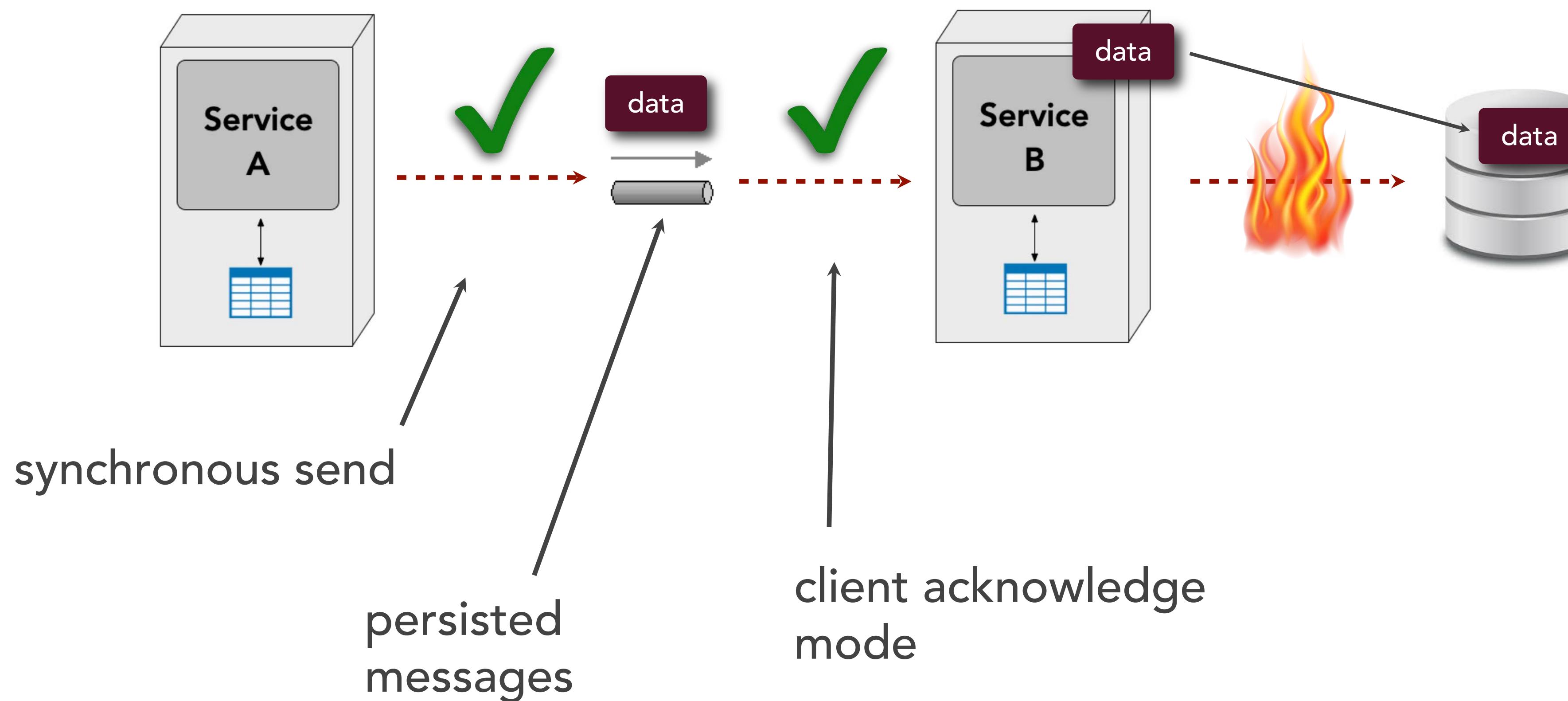
# event forwarding pattern



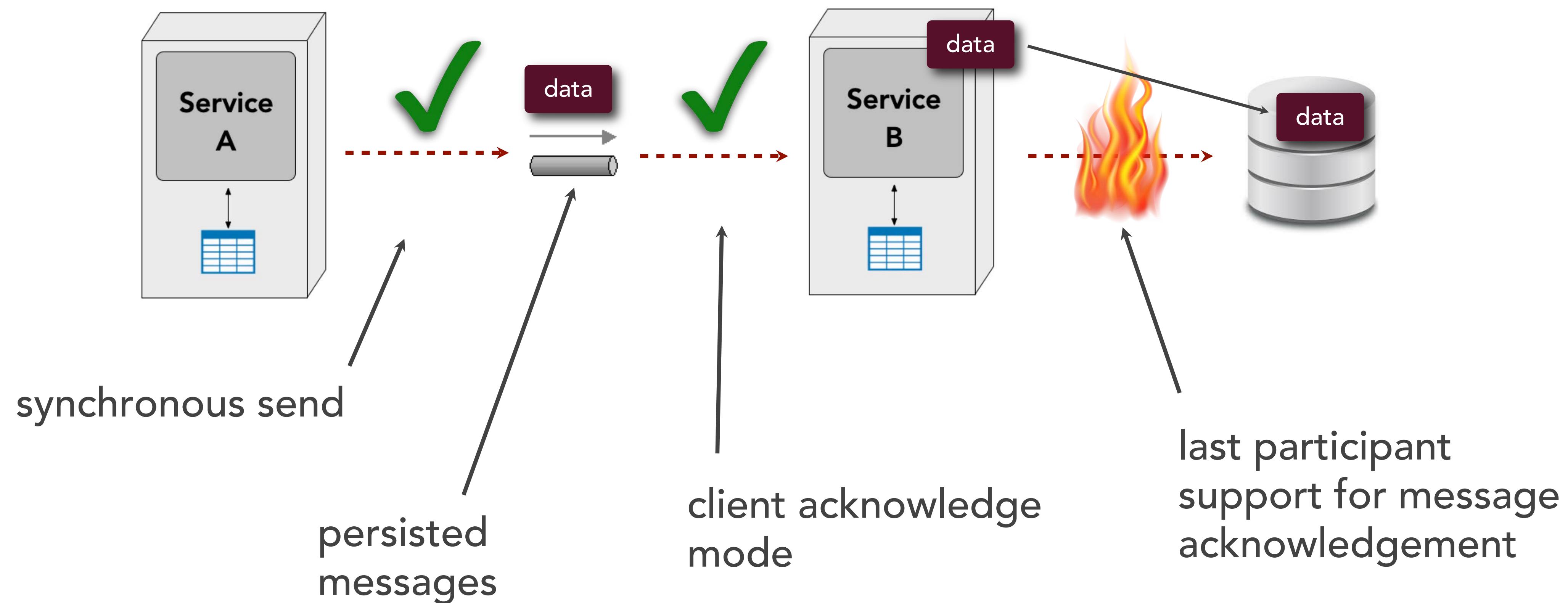
# event forwarding pattern



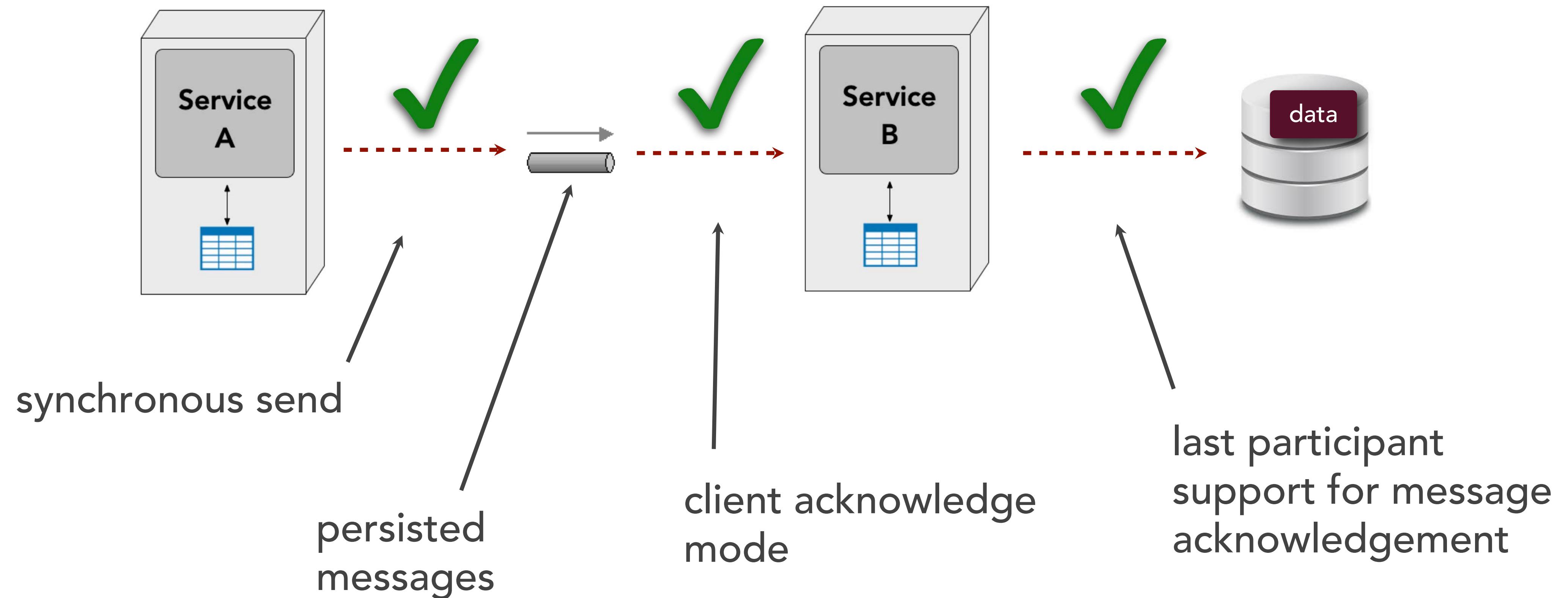
# event forwarding pattern



# event forwarding pattern



# event forwarding pattern



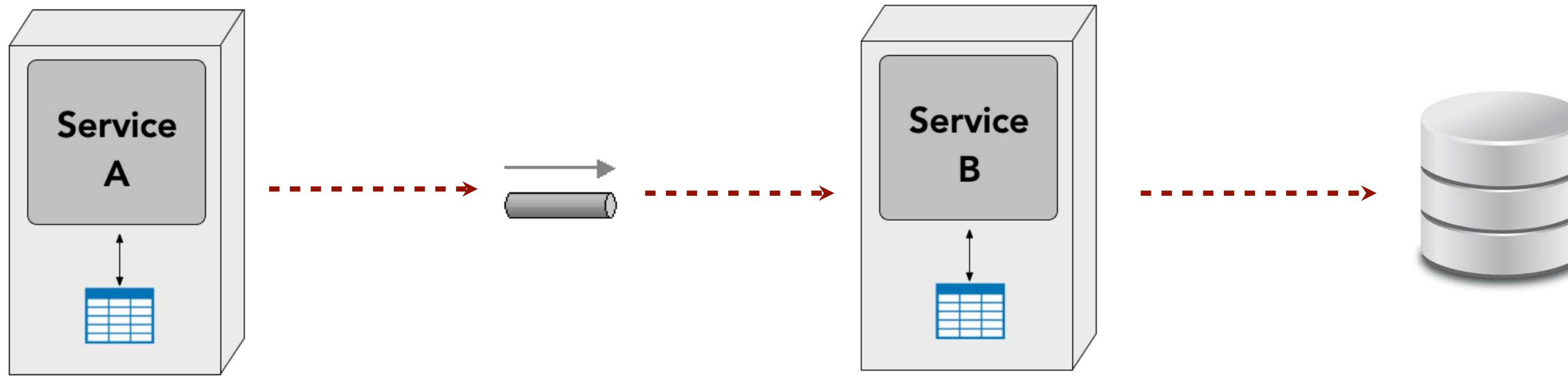
# event forwarding pattern



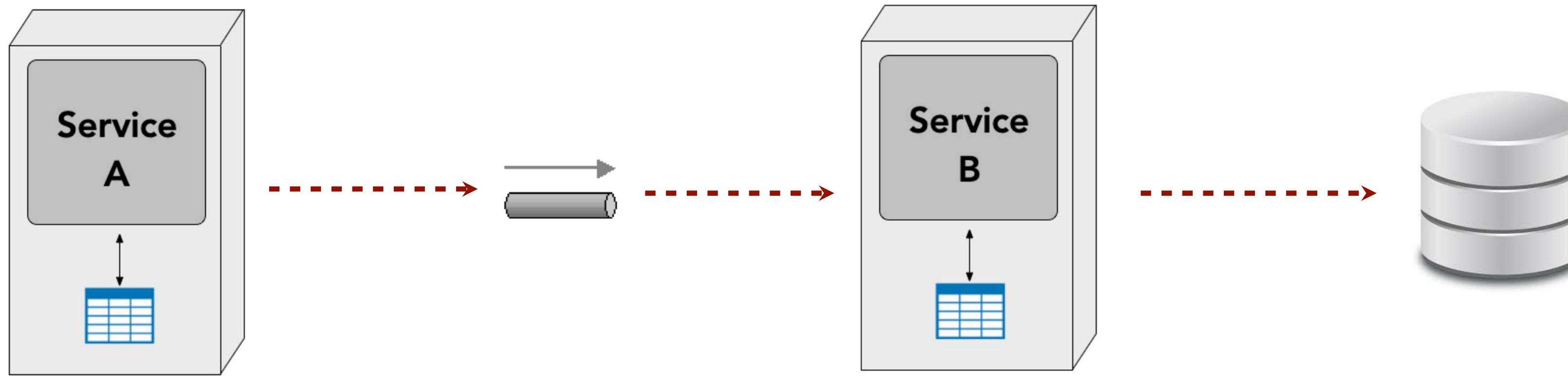
let's apply the pattern...

<https://github.com/wmr513/event-driven-patterns/tree/master/eventforwarding>

# event forwarding pattern



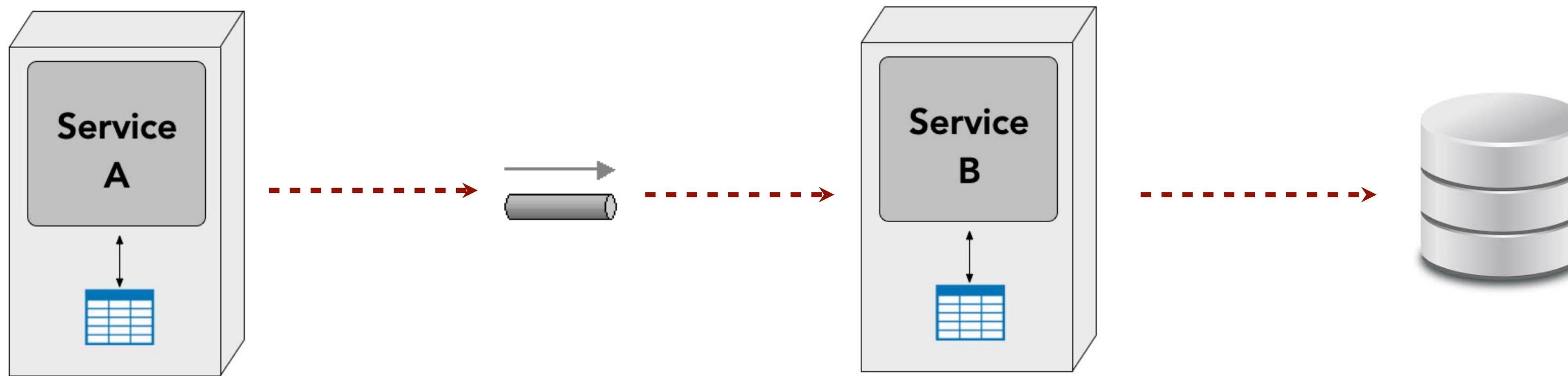
# event forwarding pattern



data integrity



# event forwarding pattern



data integrity

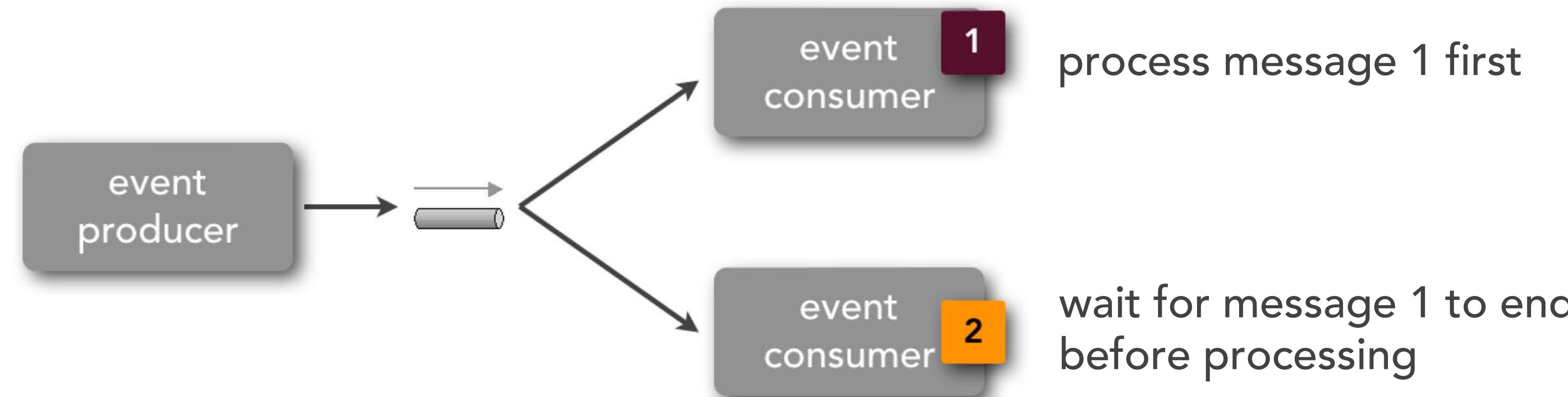


performance  
throughput  
possible duplicates

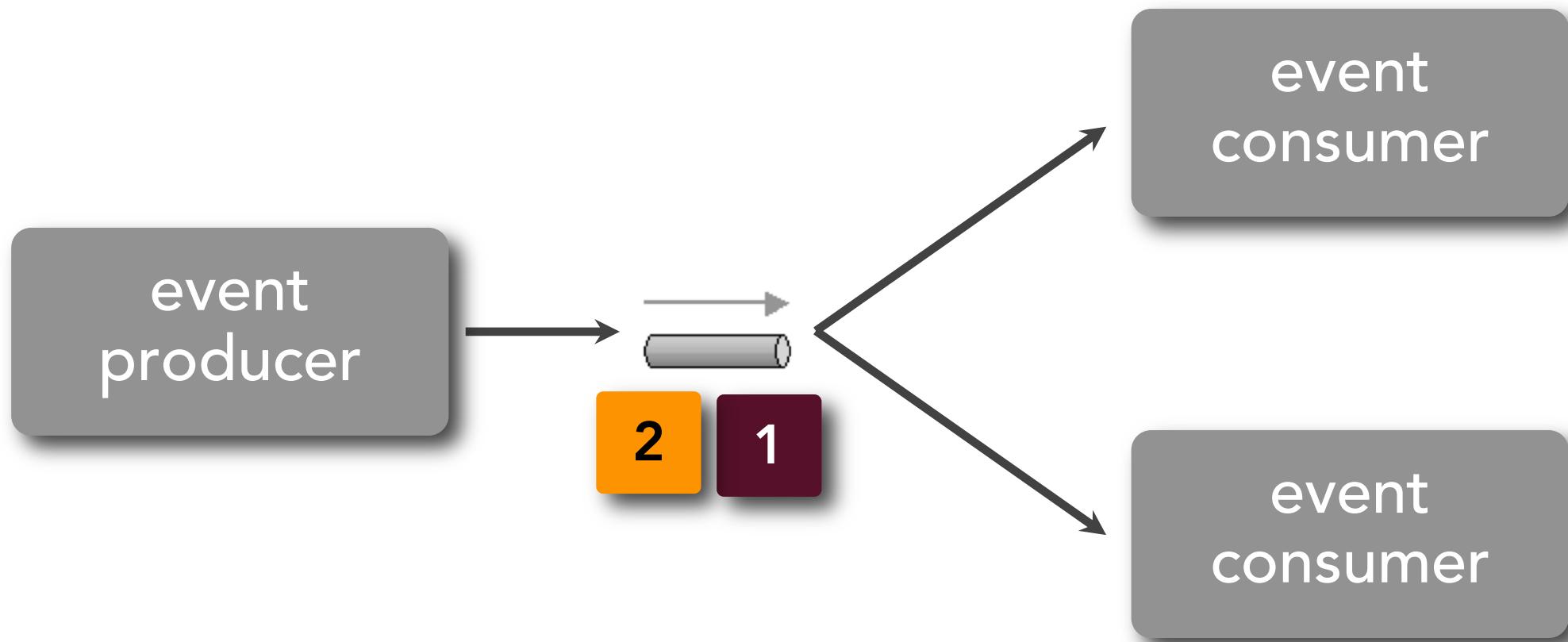
# Thread Delegate Pattern

# thread delegate pattern

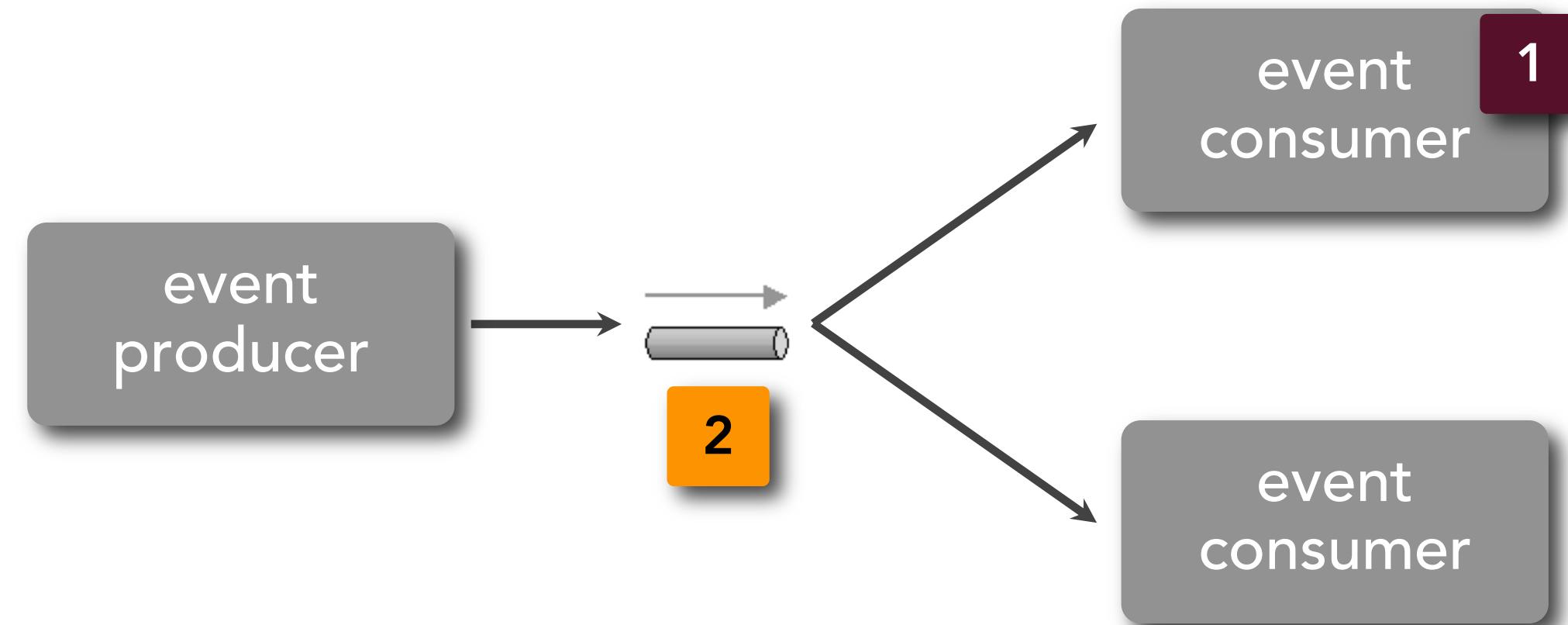
*“how can I increase performance and throughput while still maintaining message processing order?”*



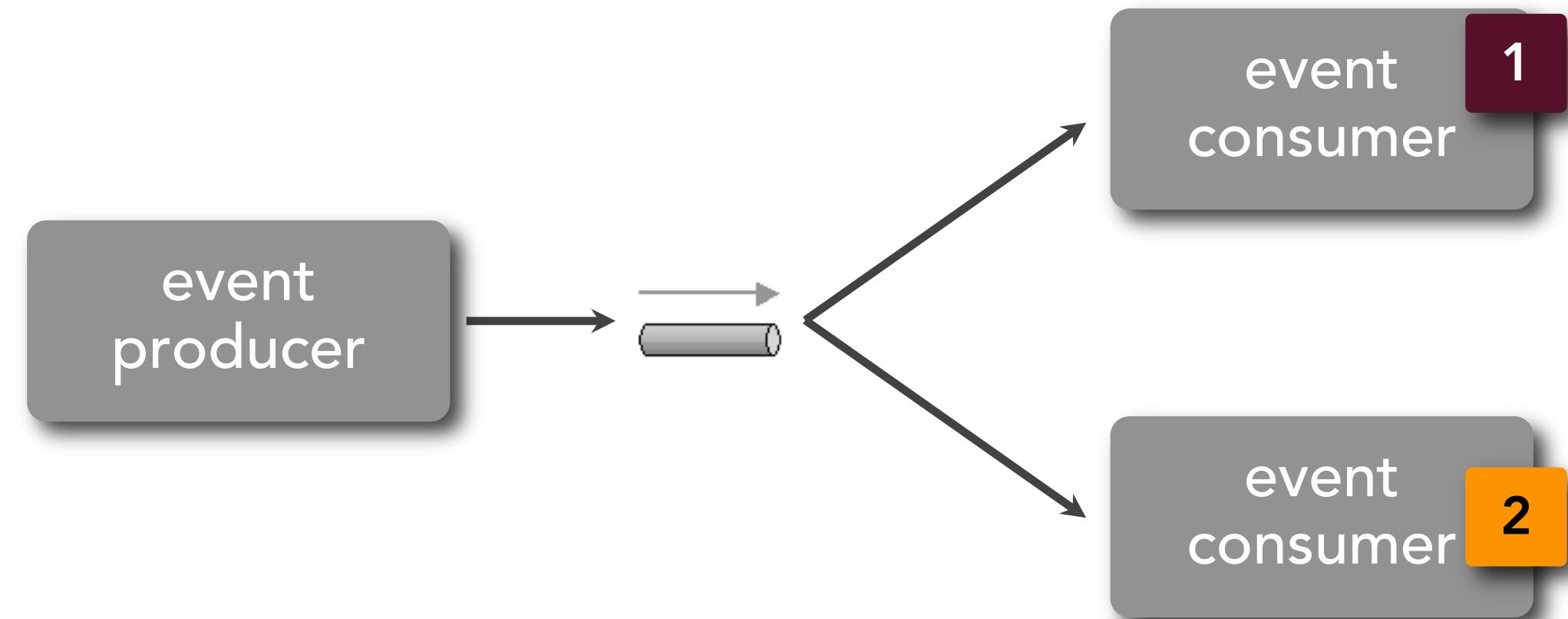
# thread delegate pattern



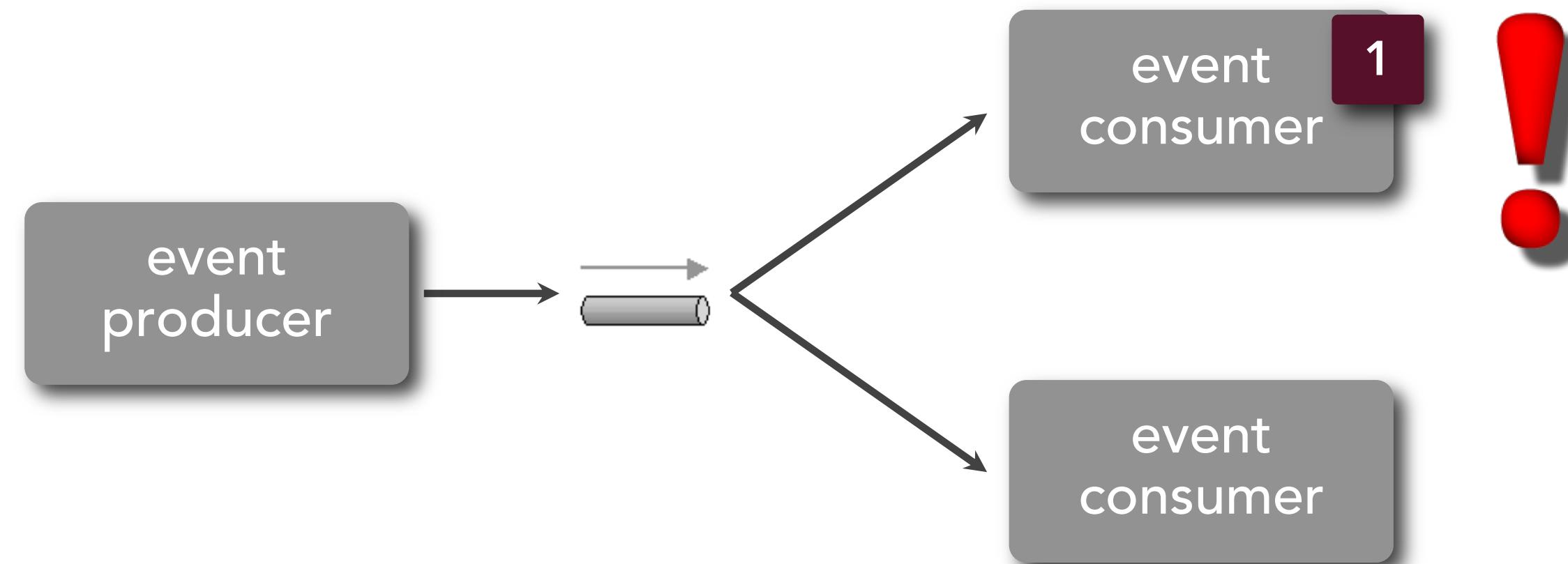
# thread delegate pattern



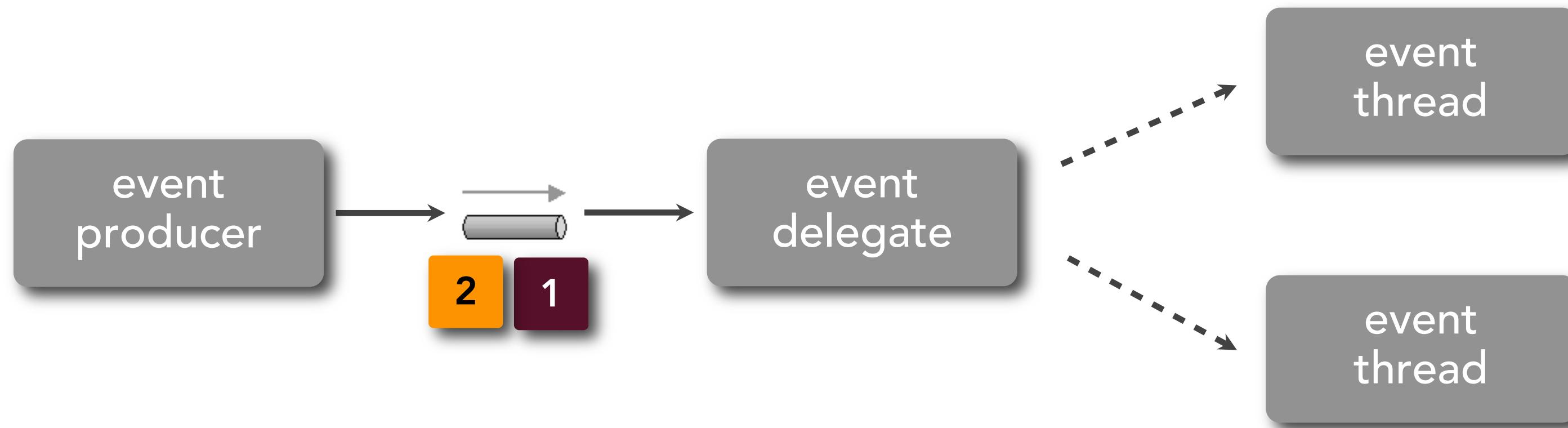
# thread delegate pattern



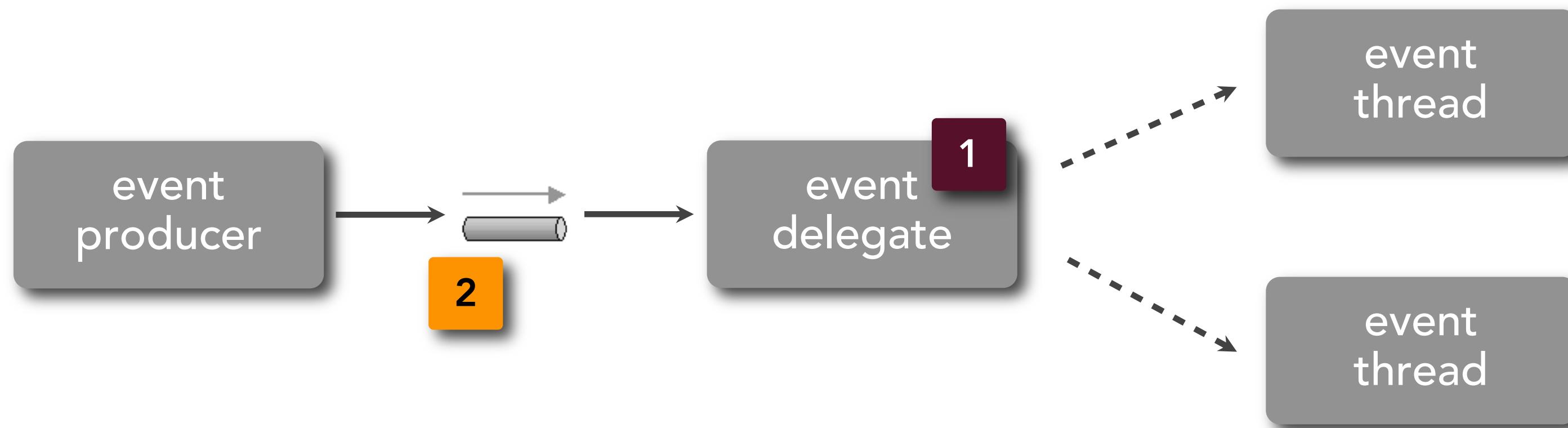
# thread delegate pattern



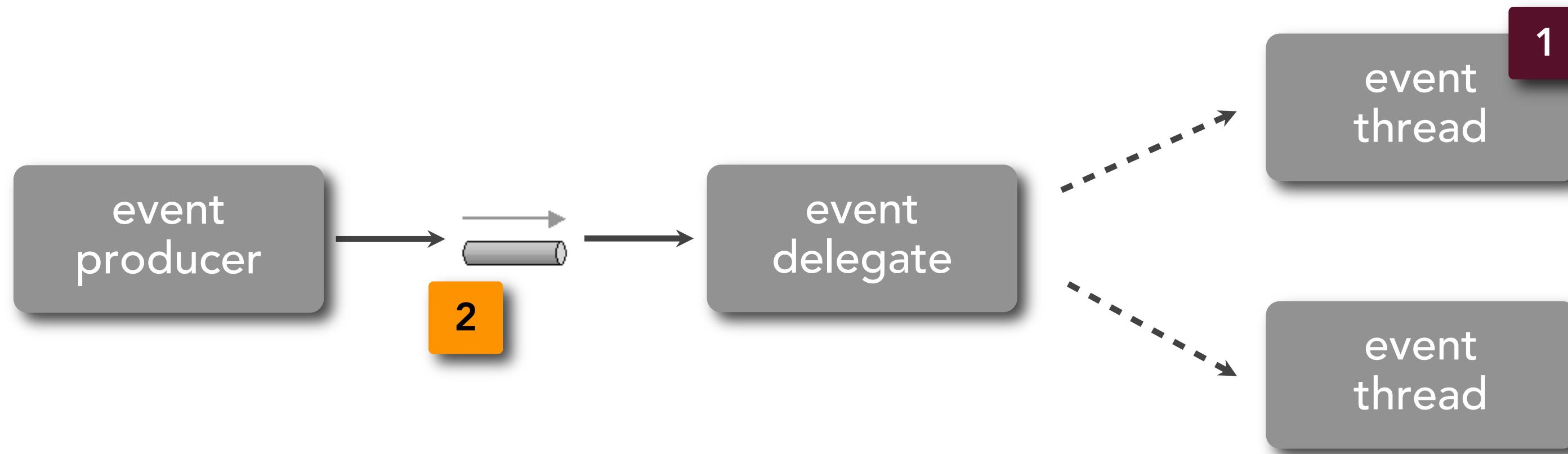
# thread delegate pattern



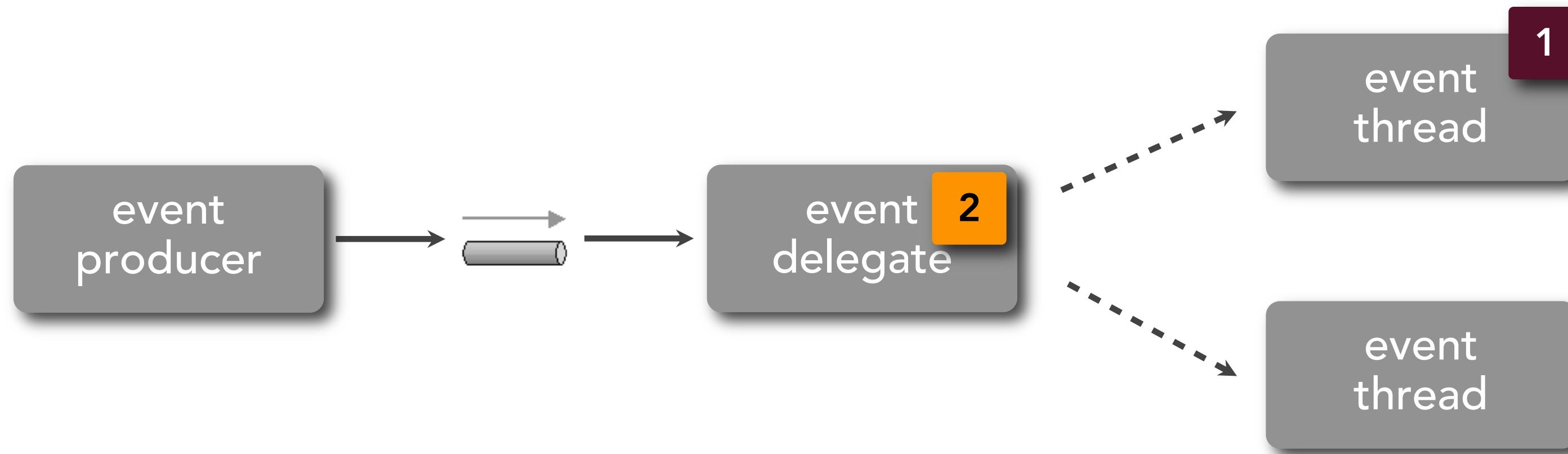
# thread delegate pattern



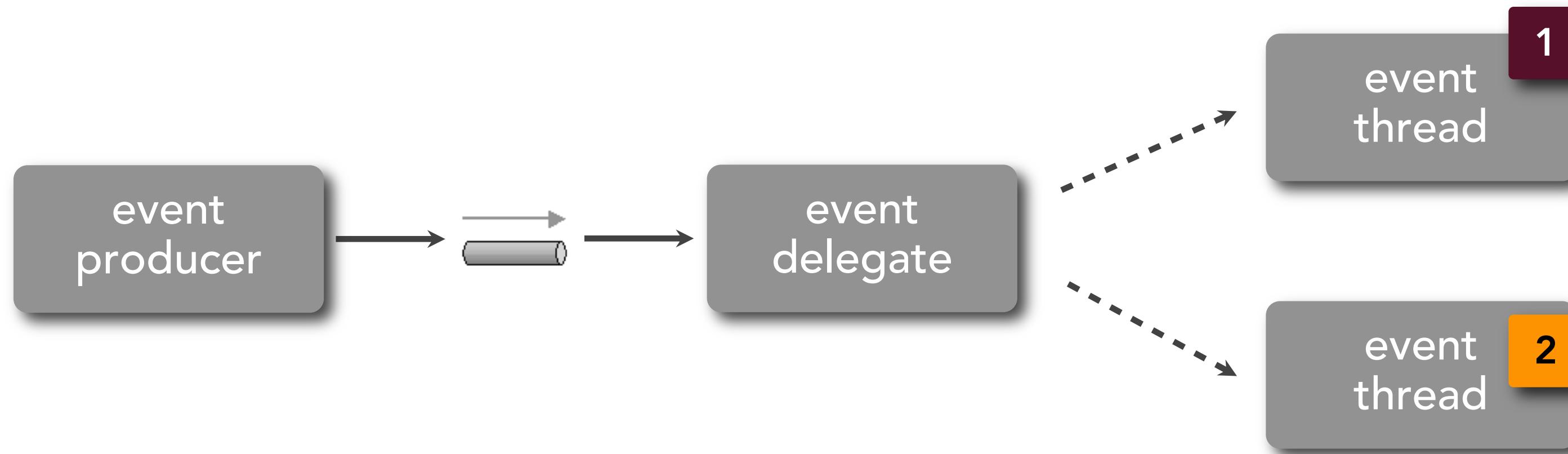
# thread delegate pattern



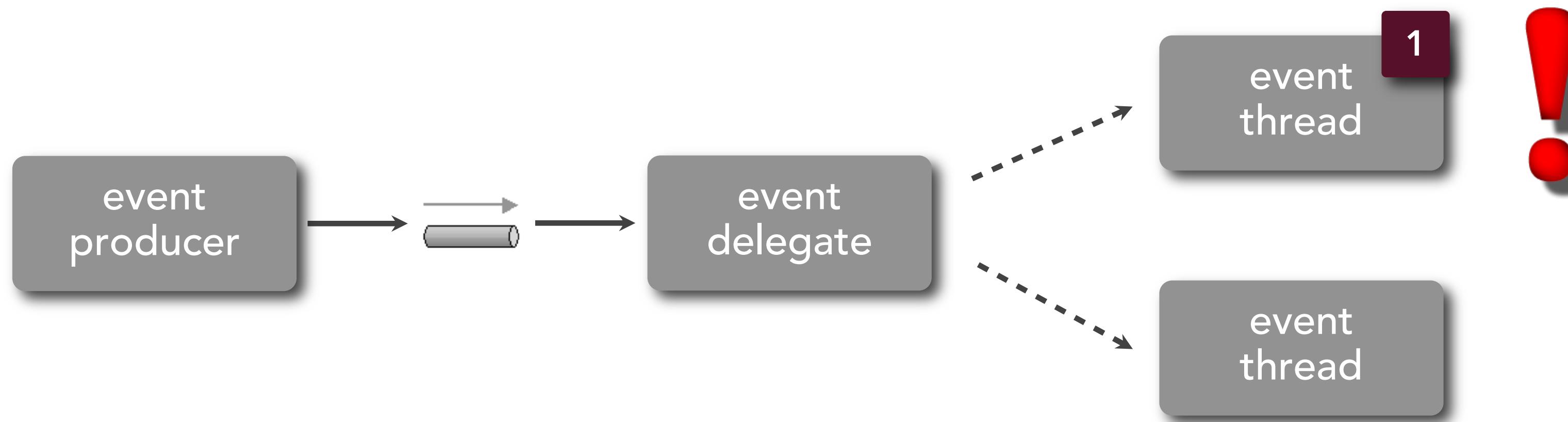
# thread delegate pattern



# thread delegate pattern



# thread delegate pattern



# thread delegate pattern

**premise:** not every message must be ordered, but rather messages  
*within a context* must be ordered

# thread delegate pattern

**premise:** not every message must be ordered, but rather messages  
*within a context* must be ordered

1. PLACE SELL GOOG A-136 2,000,000.00
2. PLACE BUY AAPL A-136 1,200,000.00
3. CANCEL BUY AAPL A-136 1,200,000.00

# thread delegate pattern

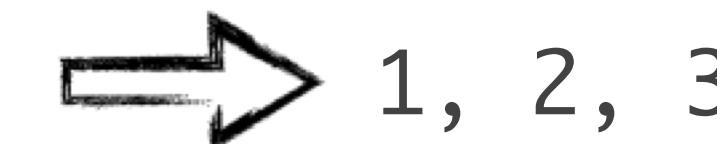
**premise:** not every message must be ordered, but rather messages  
*within a context* must be ordered

1. PLACE SELL GOOG A-136 2,000,000.00
  2. PLACE BUY AAPL A-136 1,200,000.00
  3. CANCEL BUY AAPL A-136 1,200,000.00
- 
- 1, 2, 3

# thread delegate pattern

**premise:** not every message must be ordered, but rather messages  
*within a context* must be ordered

1. PLACE SELL GOOG A-136 2,000,000.00
2. PLACE BUY AAPL A-136 1,200,000.00
3. CANCEL BUY AAPL A-136 1,200,000.00



1, 2, 3

1. PLACE SELL AAPL A-136 2,000,000.00
2. PLACE SELL GOOG V-976 650,000.00
3. PLACE BUY ATT V-976 900,000.00
4. PLACE BUY IBM A-136 1,300,000.00
5. CANCEL BUY ATT V-976 900,000.00

# thread delegate pattern

**premise:** not every message must be ordered, but rather messages  
*within a context* must be ordered

1. PLACE SELL GOOG A-136 2,000,000.00
2. PLACE BUY AAPL A-136 1,200,000.00
3. CANCEL BUY AAPL A-136 1,200,000.00

→ 1, 2, 3

1. PLACE SELL AAPL A-136 2,000,000.00
2. PLACE SELL GOOG V-976 650,000.00
3. PLACE BUY ATT V-976 900,000.00
4. PLACE BUY IBM A-136 1,300,000.00
5. CANCEL BUY ATT V-976 900,000.00

→ 1, 4

# thread delegate pattern

**premise:** not every message must be ordered, but rather messages  
*within a context* must be ordered

1. PLACE SELL GOOG A-136 2,000,000.00
2. PLACE BUY AAPL A-136 1,200,000.00
3. CANCEL BUY AAPL A-136 1,200,000.00

→ 1, 2, 3

1. PLACE SELL AAPL A-136 2,000,000.00
2. PLACE SELL GOOG V-976 650,000.00
3. PLACE BUY ATT V-976 900,000.00
4. PLACE BUY IBM A-136 1,300,000.00
5. CANCEL BUY ATT V-976 900,000.00

→ 2, 3, 5

# thread delegate pattern

**premise:** not every message must be ordered, but rather messages  
*within a context* must be ordered

1. PLACE SELL GOOG A-136 2,000,000.00
2. PLACE BUY AAPL A-136 1,200,000.00
3. CANCEL BUY AAPL A-136 1,200,000.00

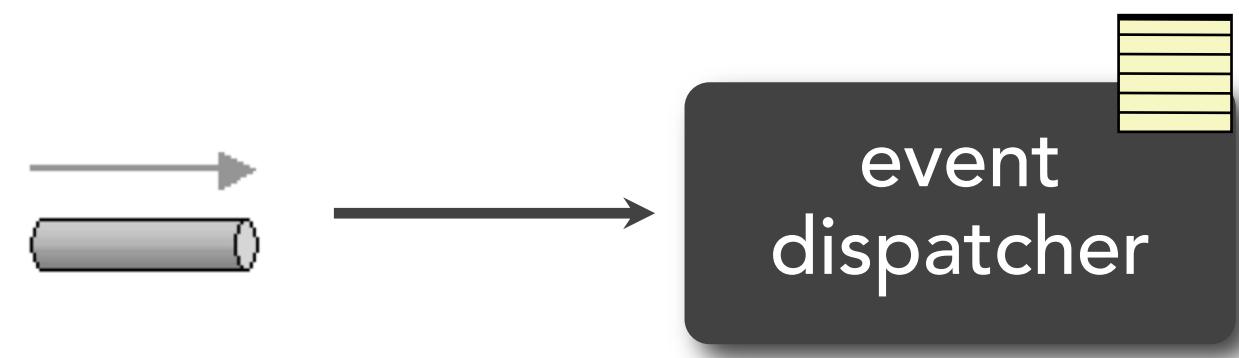
→ 1, 2, 3

1. PLACE SELL AAPL A-136 2,000,000.00
2. PLACE SELL GOOG V-976 650,000.00
3. PLACE BUY ATT V-976 900,000.00
4. PLACE BUY IBM A-136 1,300,000.00
5. CANCEL BUY ATT V-976 900,000.00

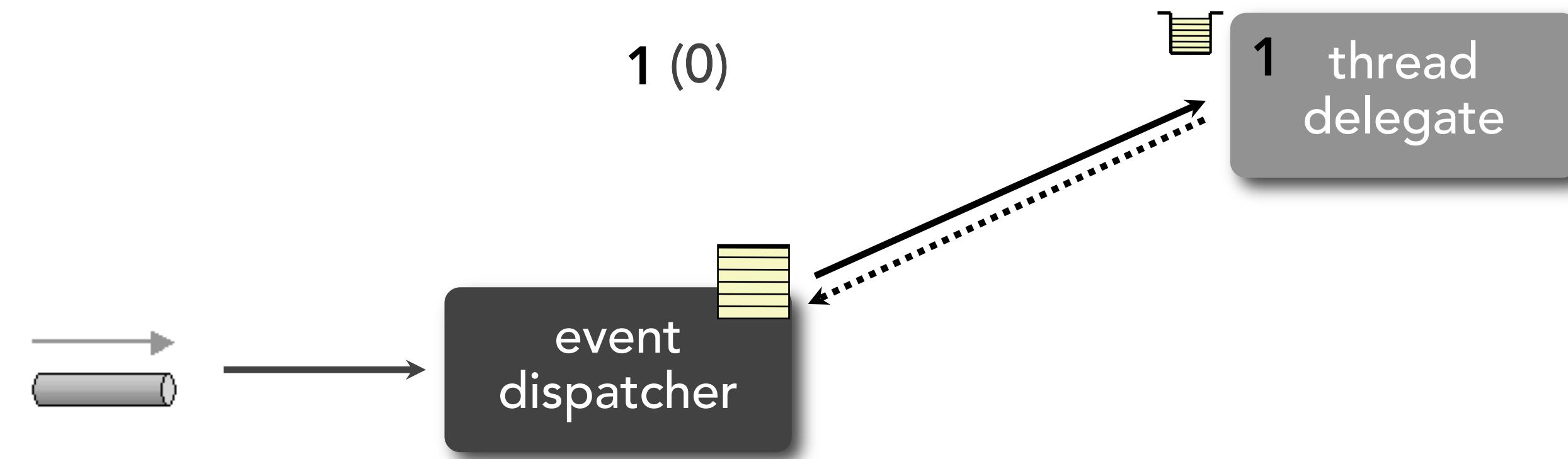
→ 1, 4

→ 2, 3, 5

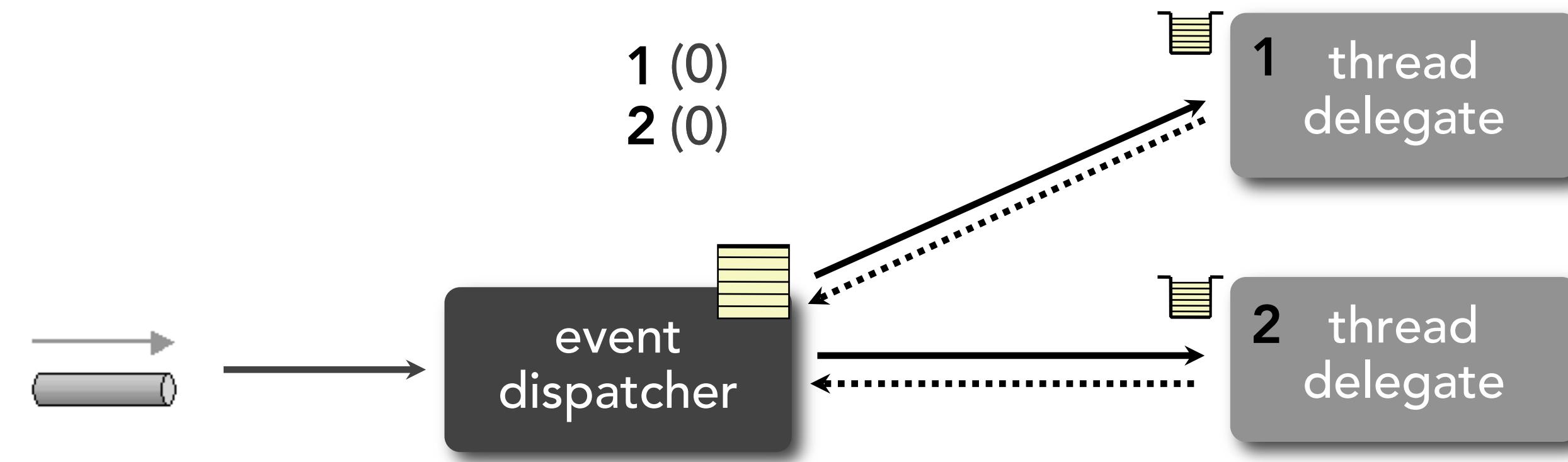
# thread delegate pattern



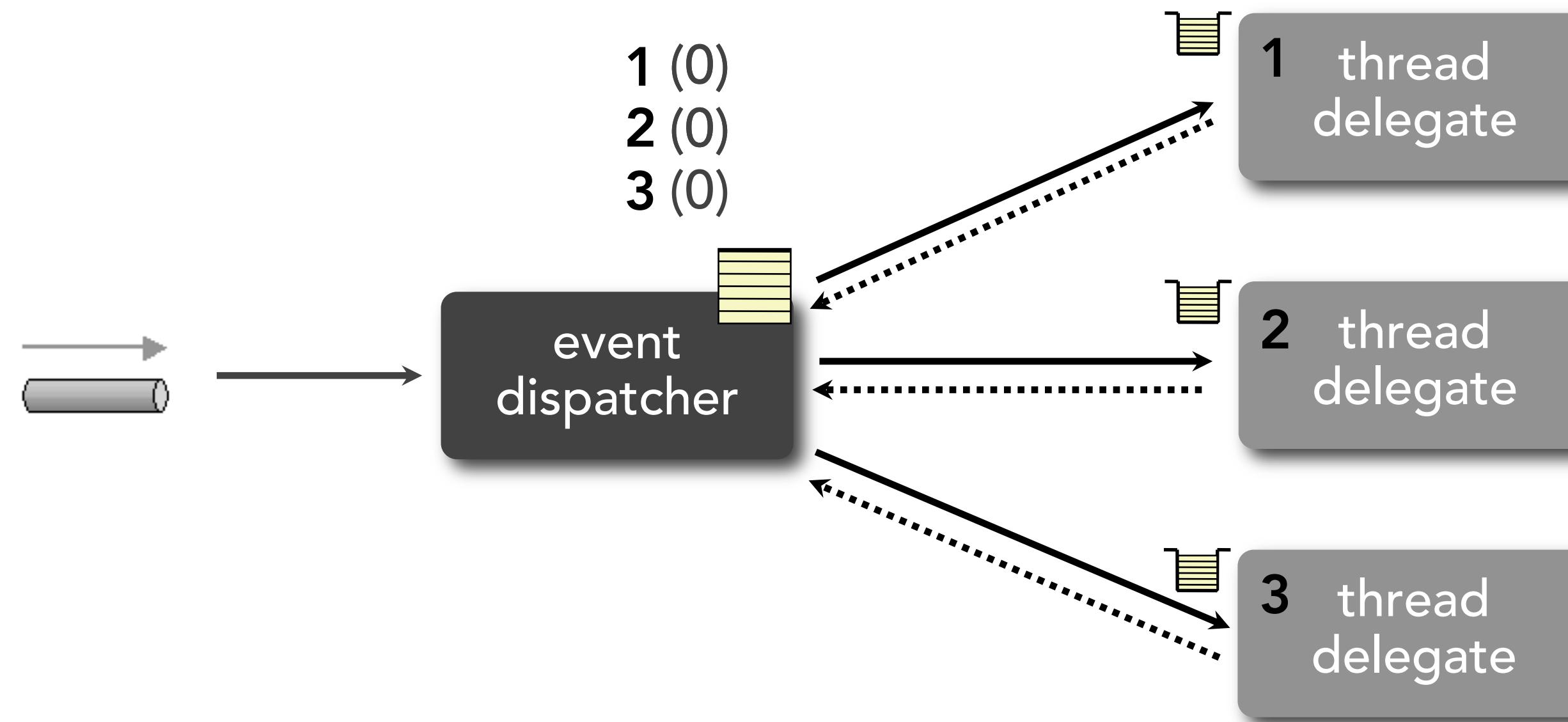
# thread delegate pattern



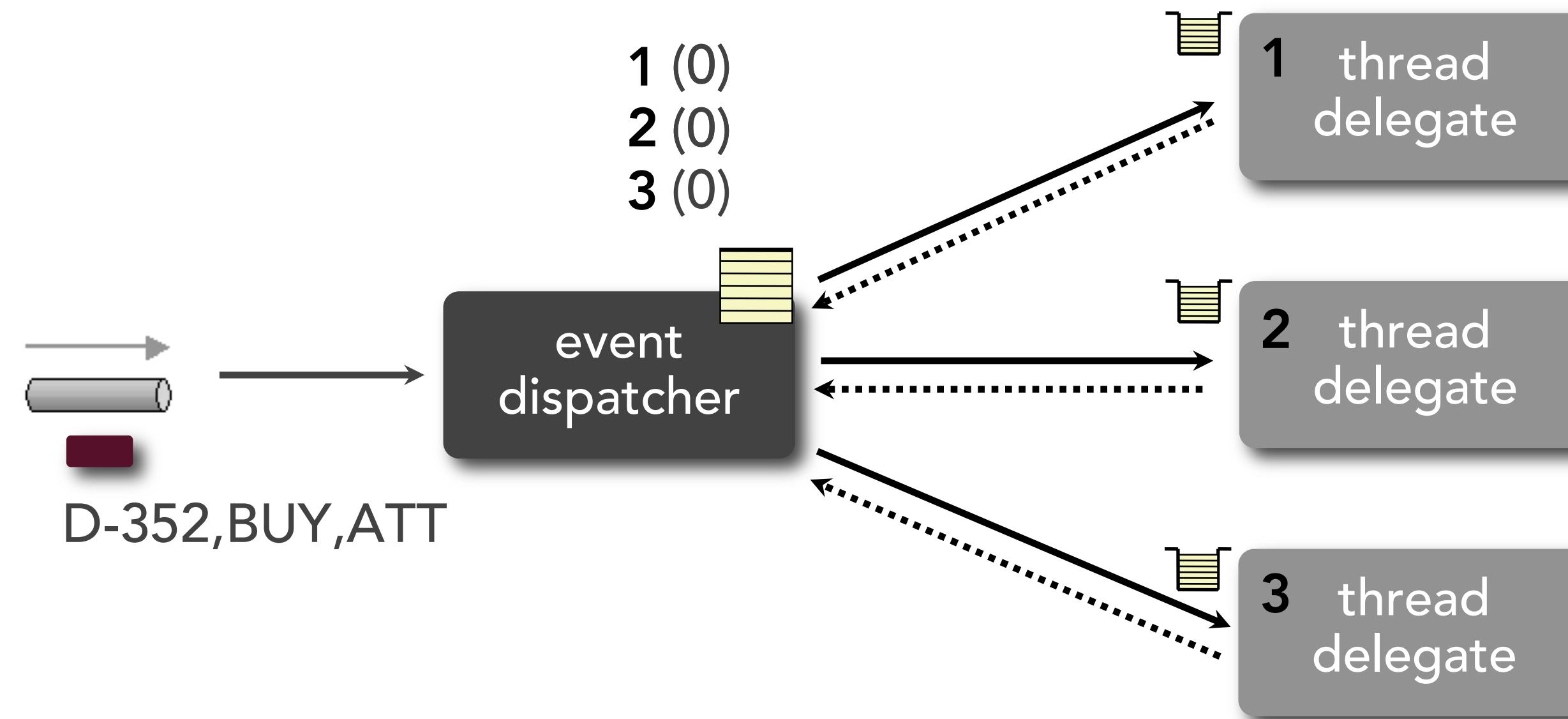
# thread delegate pattern



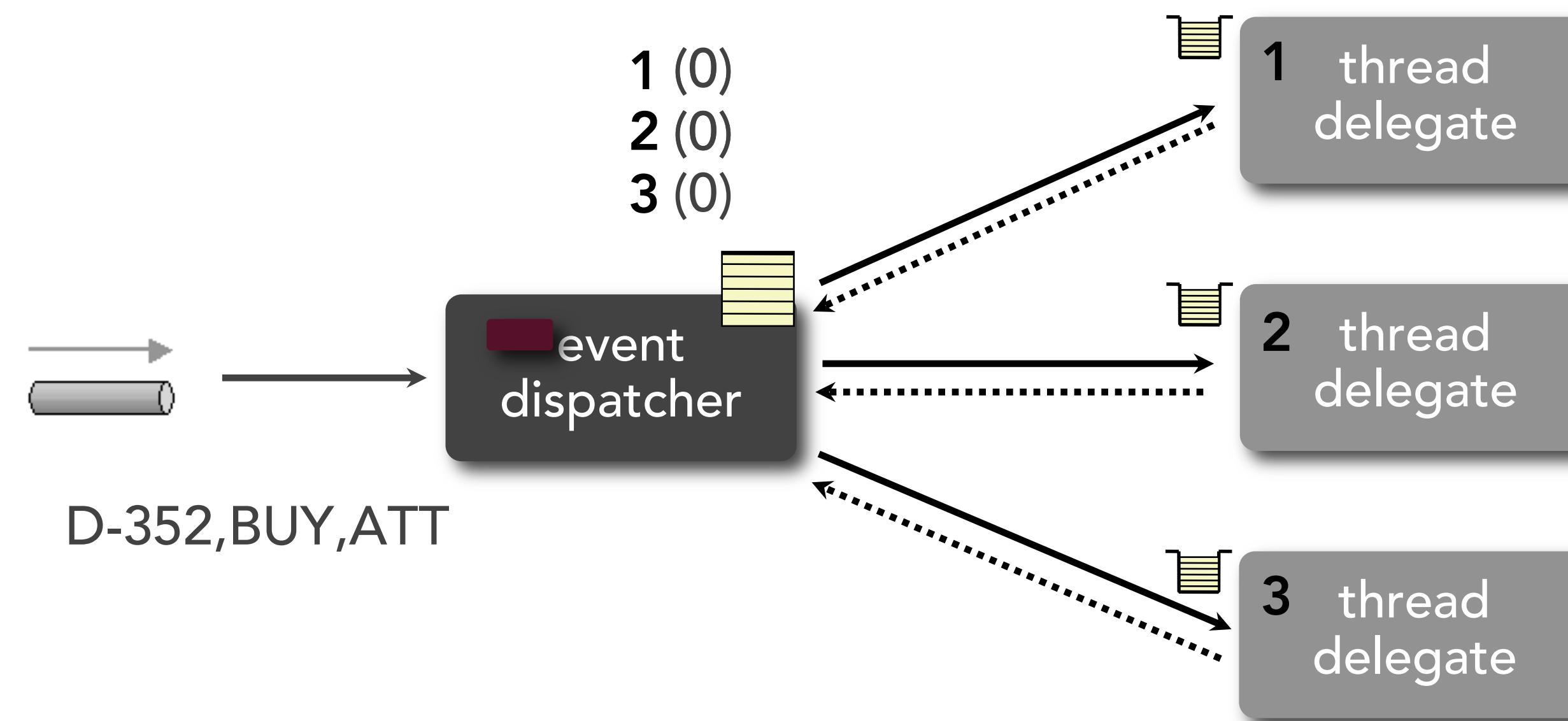
# thread delegate pattern



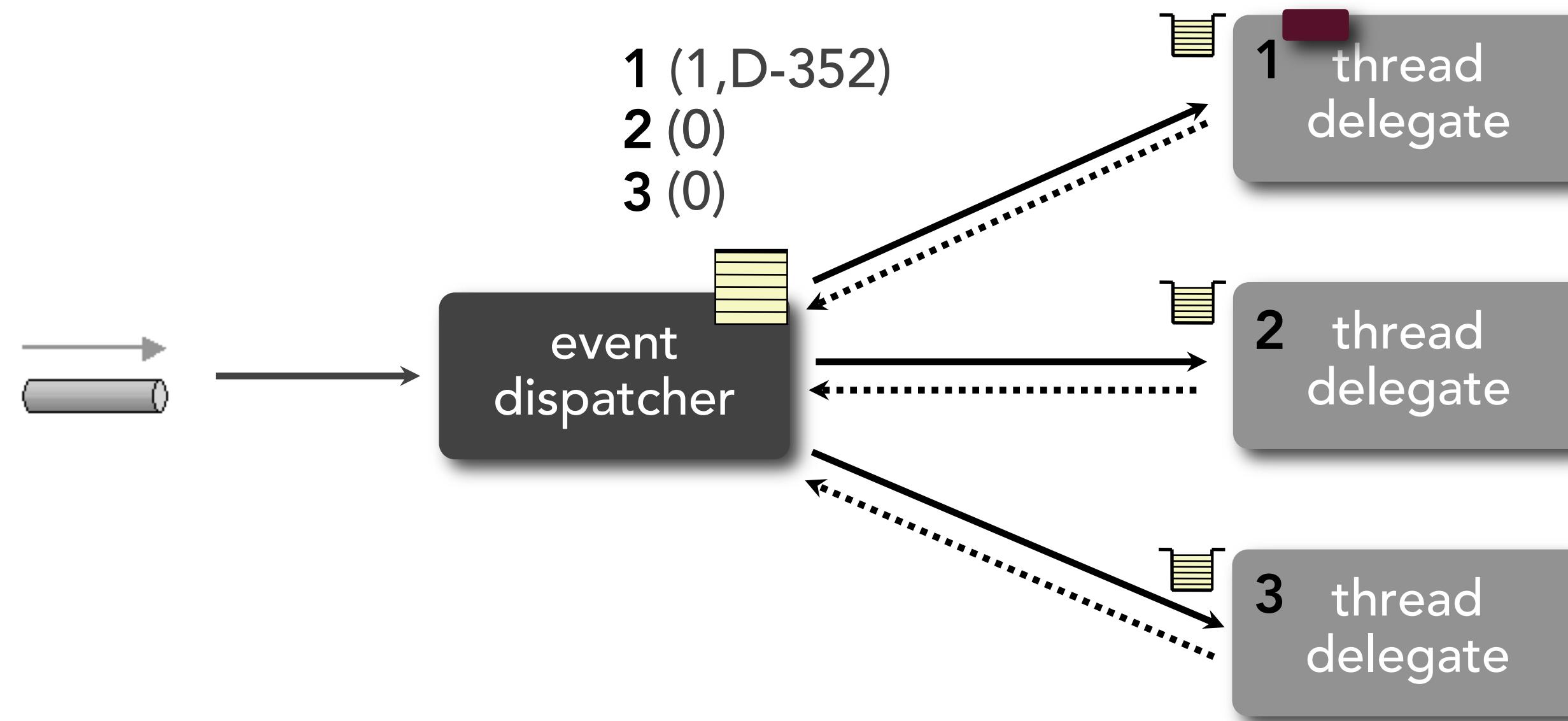
# thread delegate pattern



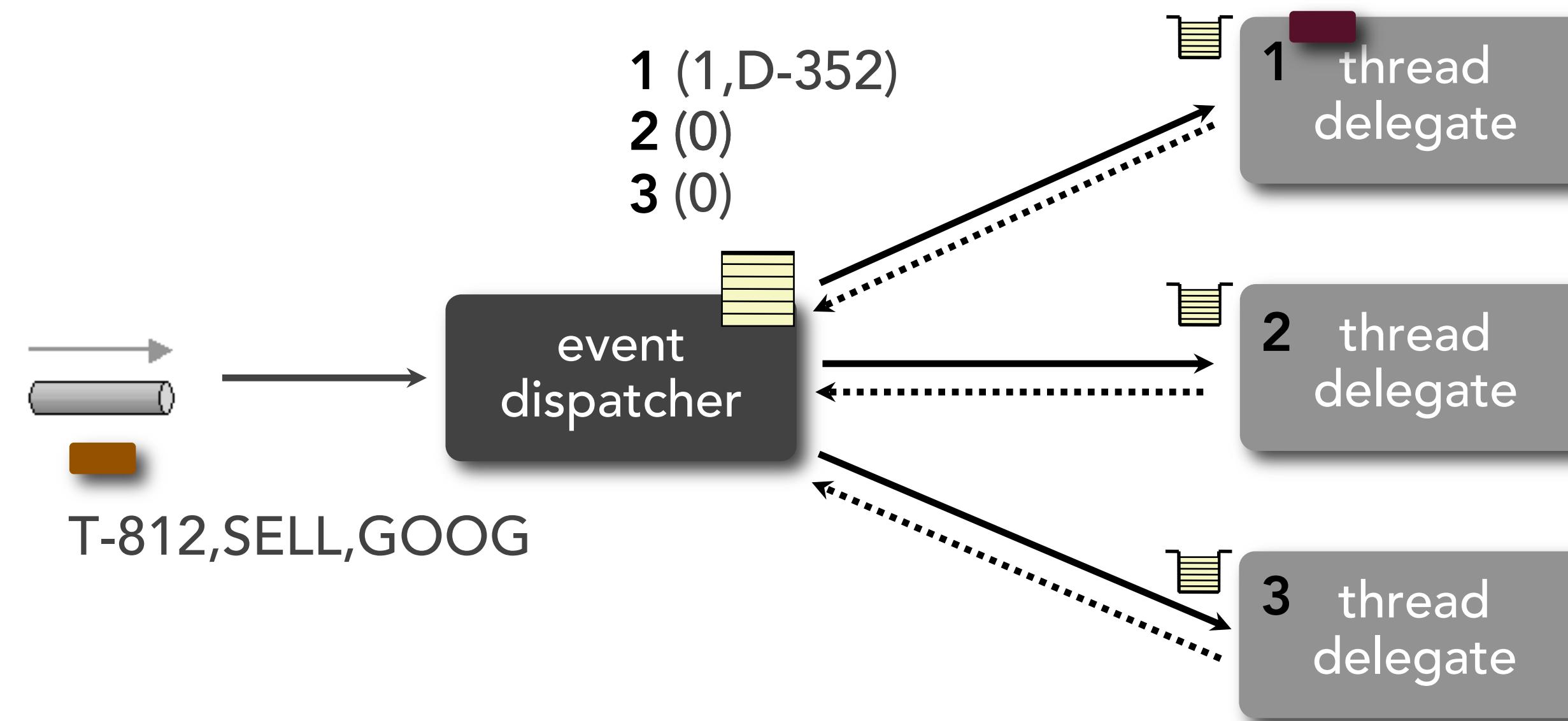
# thread delegate pattern



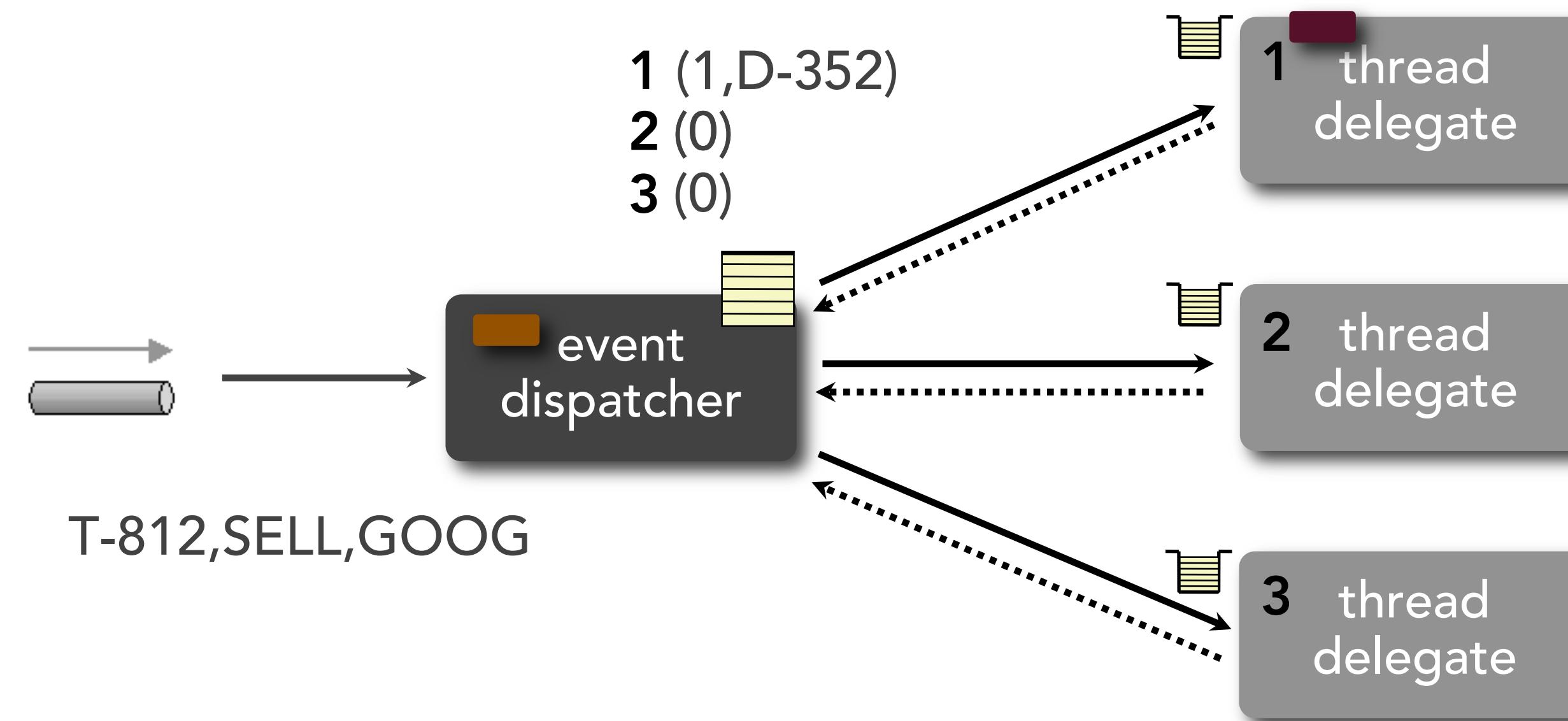
# thread delegate pattern



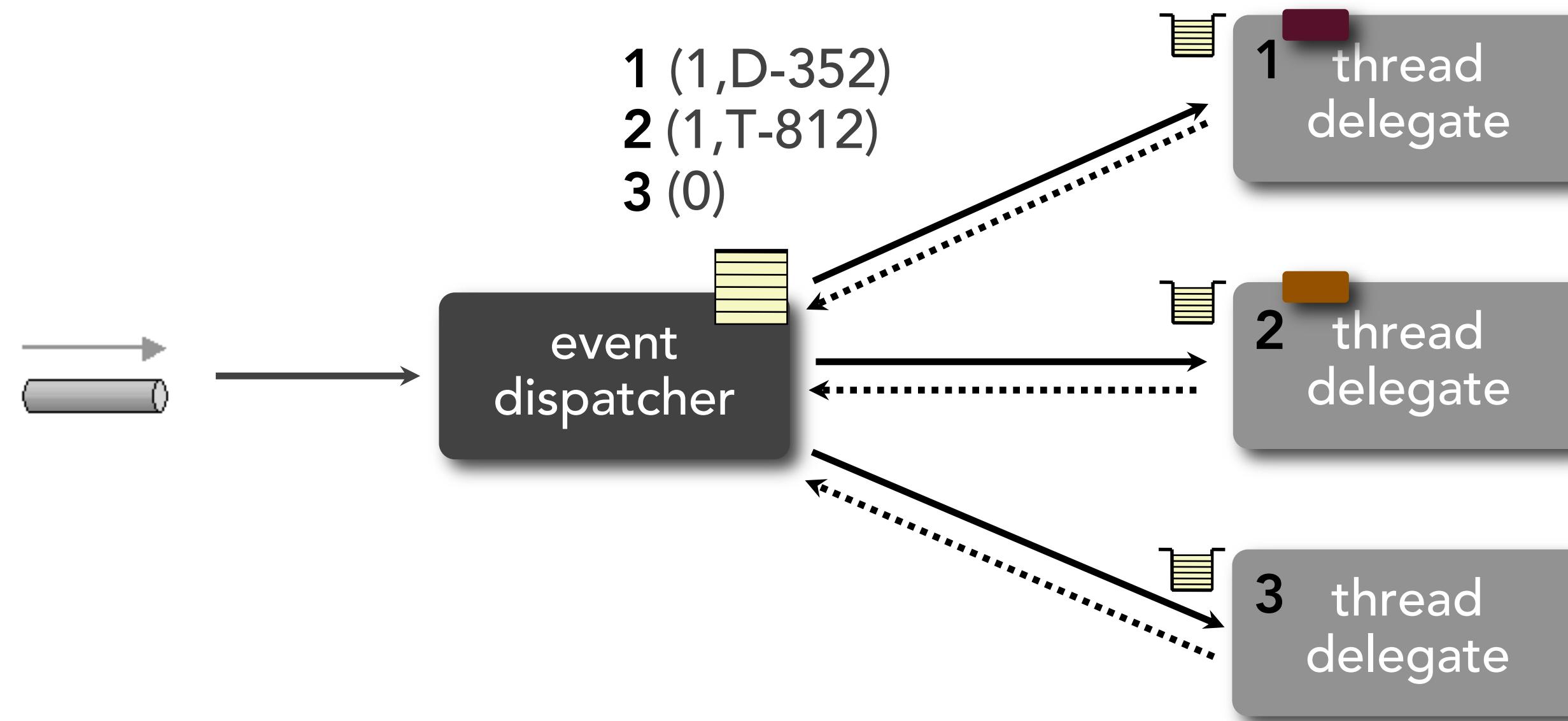
# thread delegate pattern



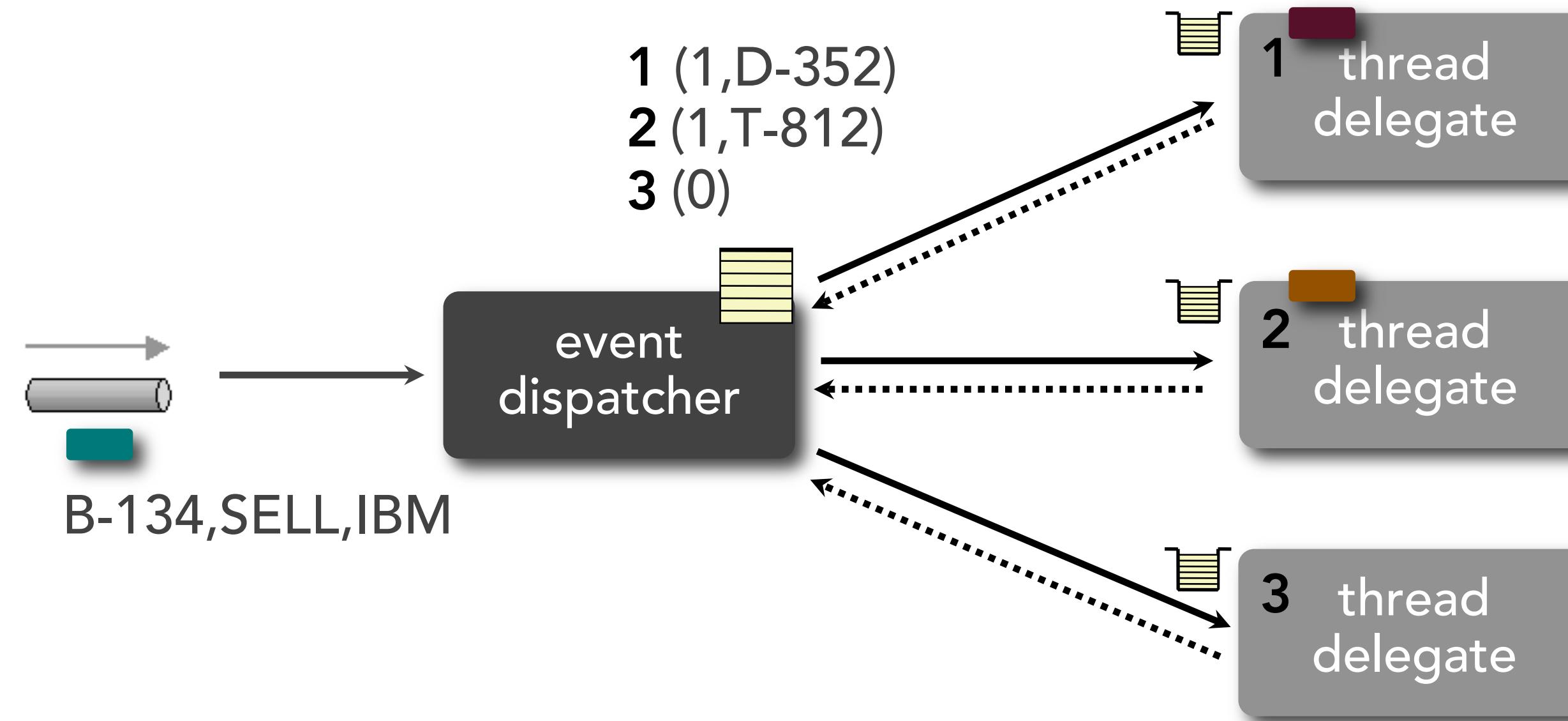
# thread delegate pattern



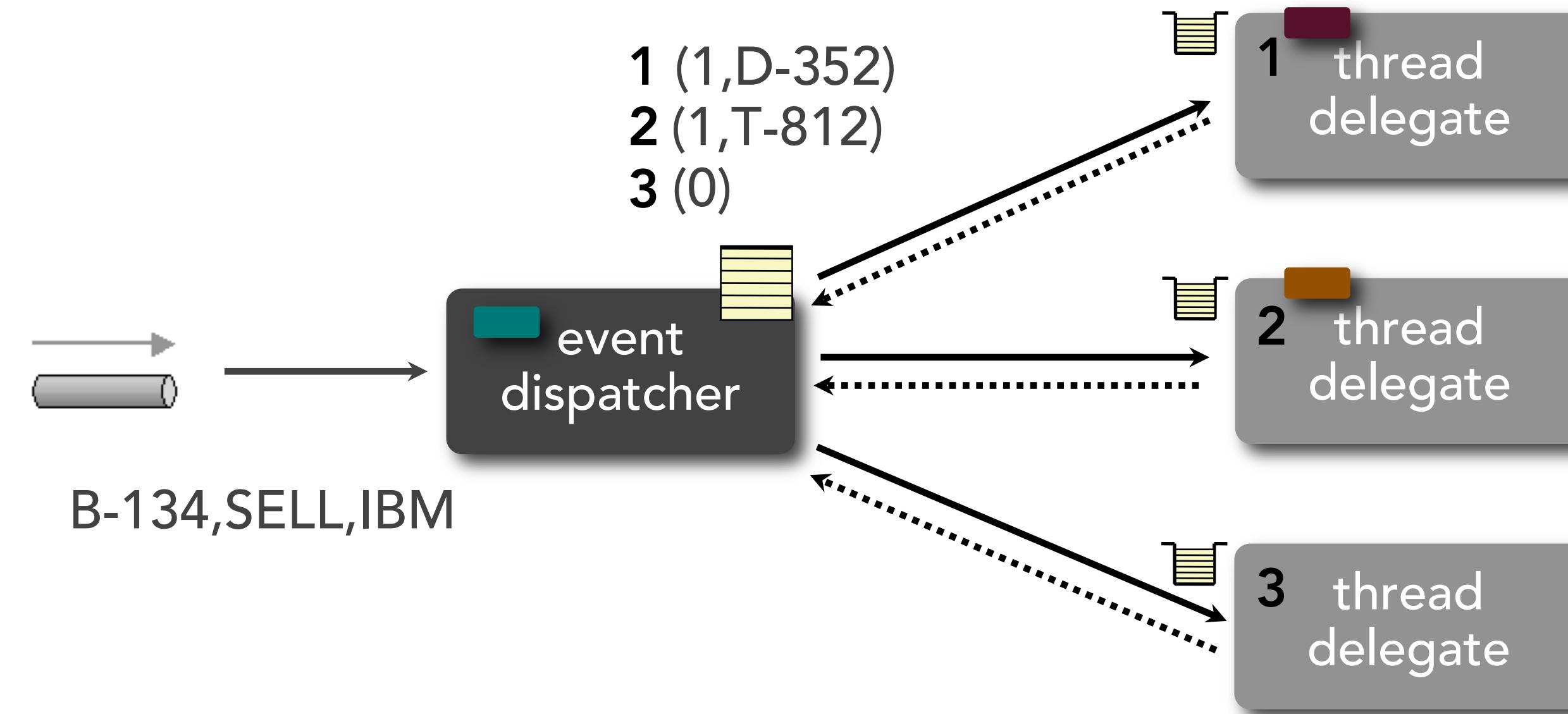
# thread delegate pattern



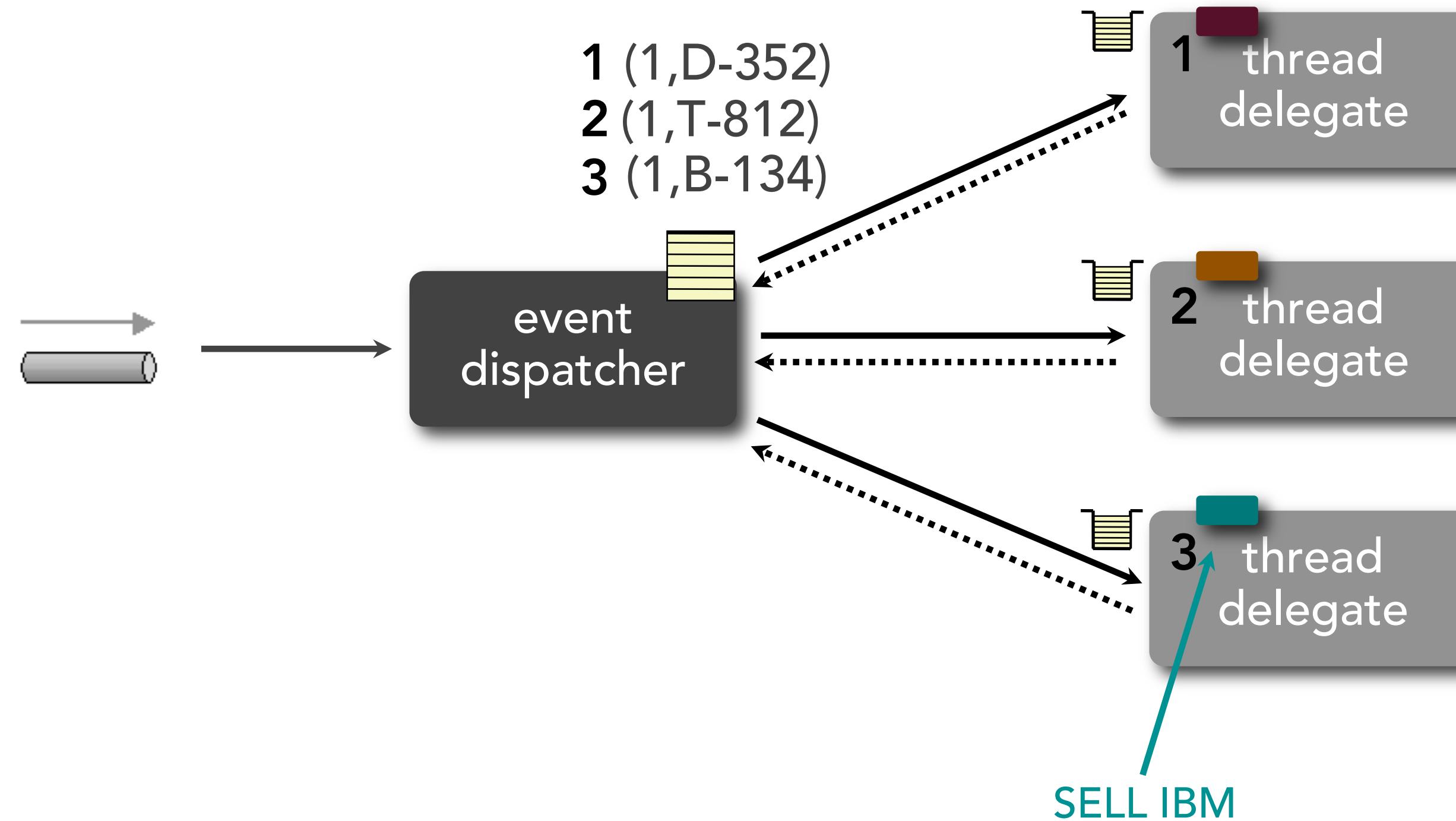
# thread delegate pattern



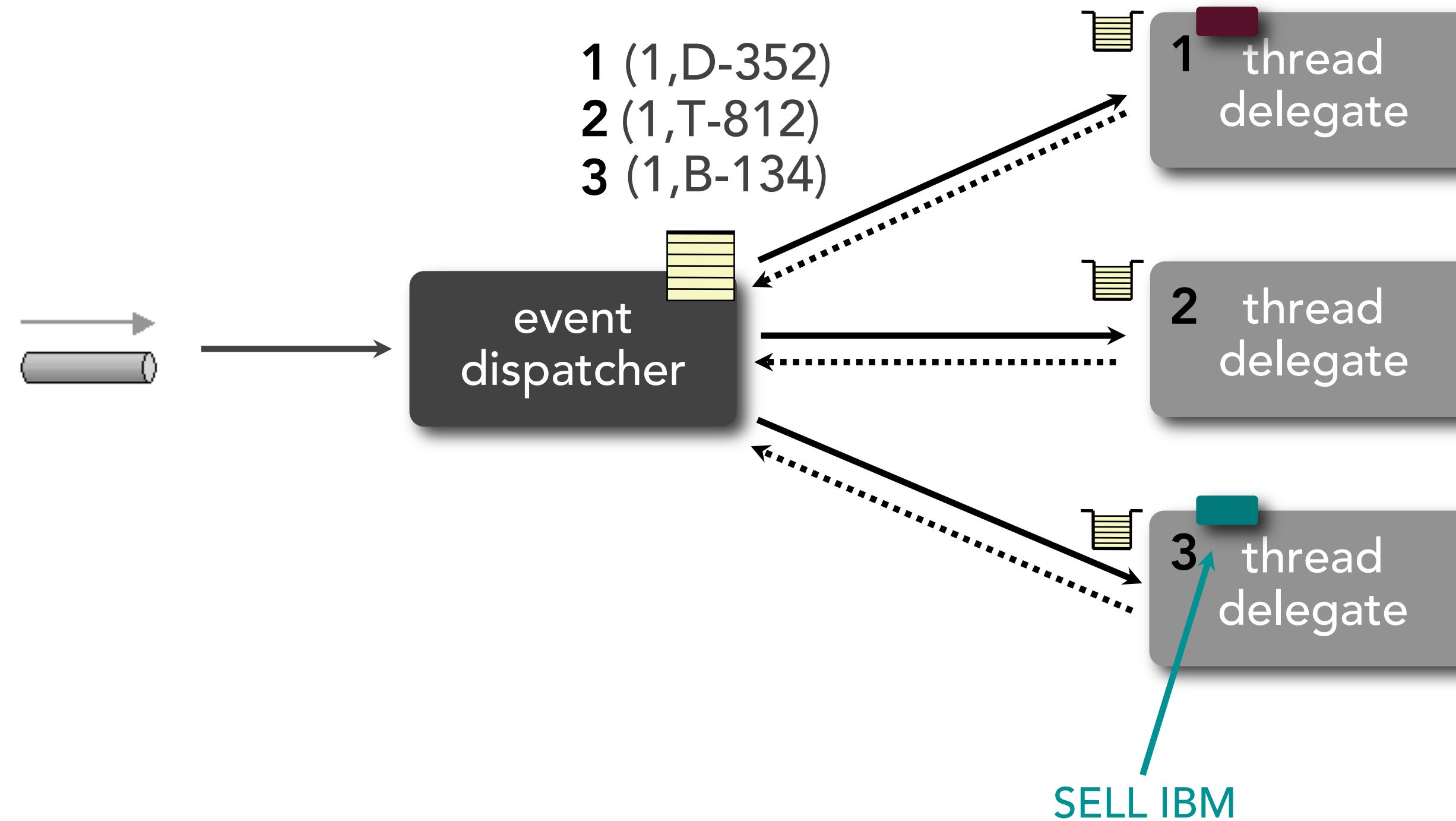
# thread delegate pattern



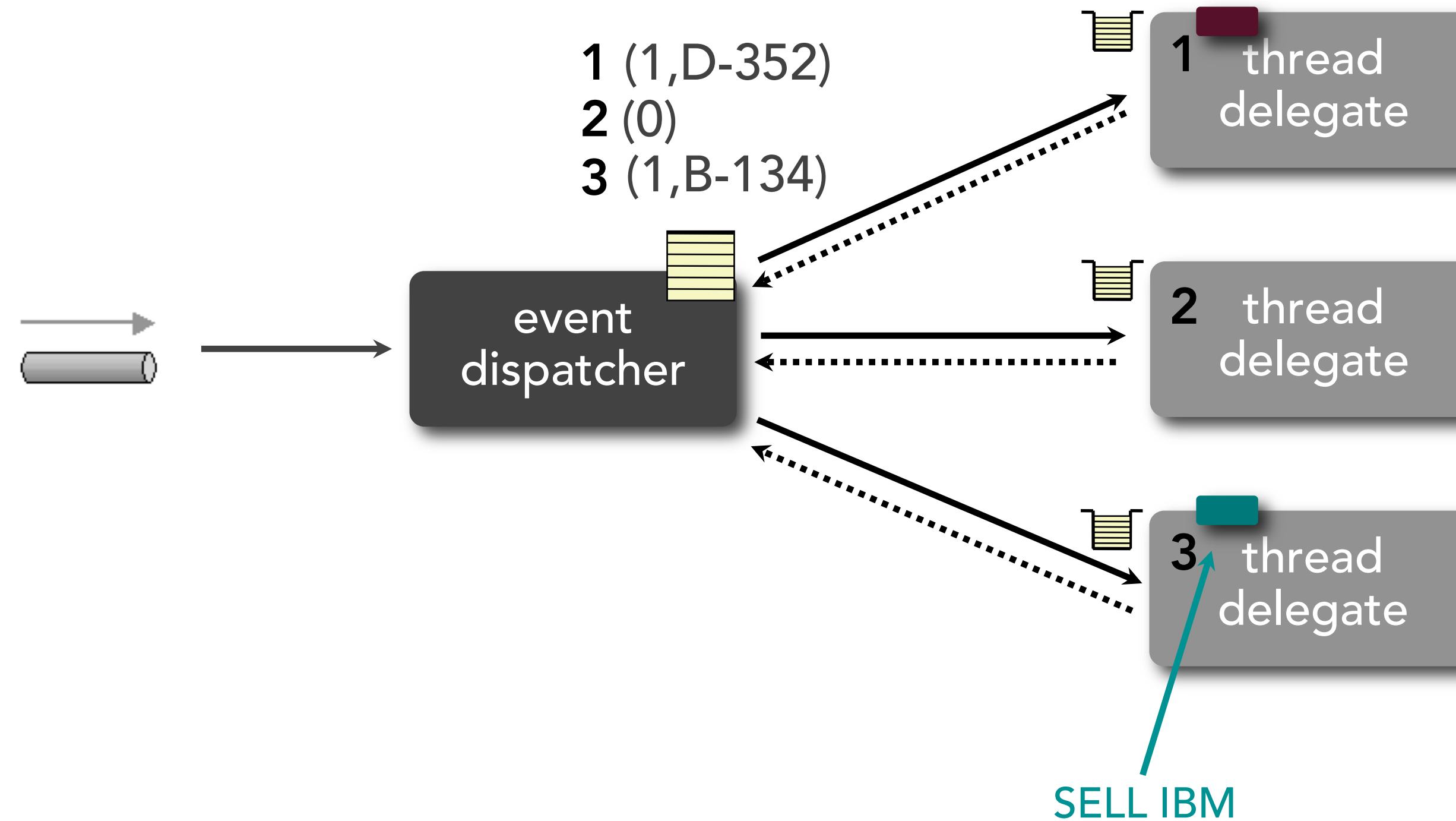
# thread delegate pattern



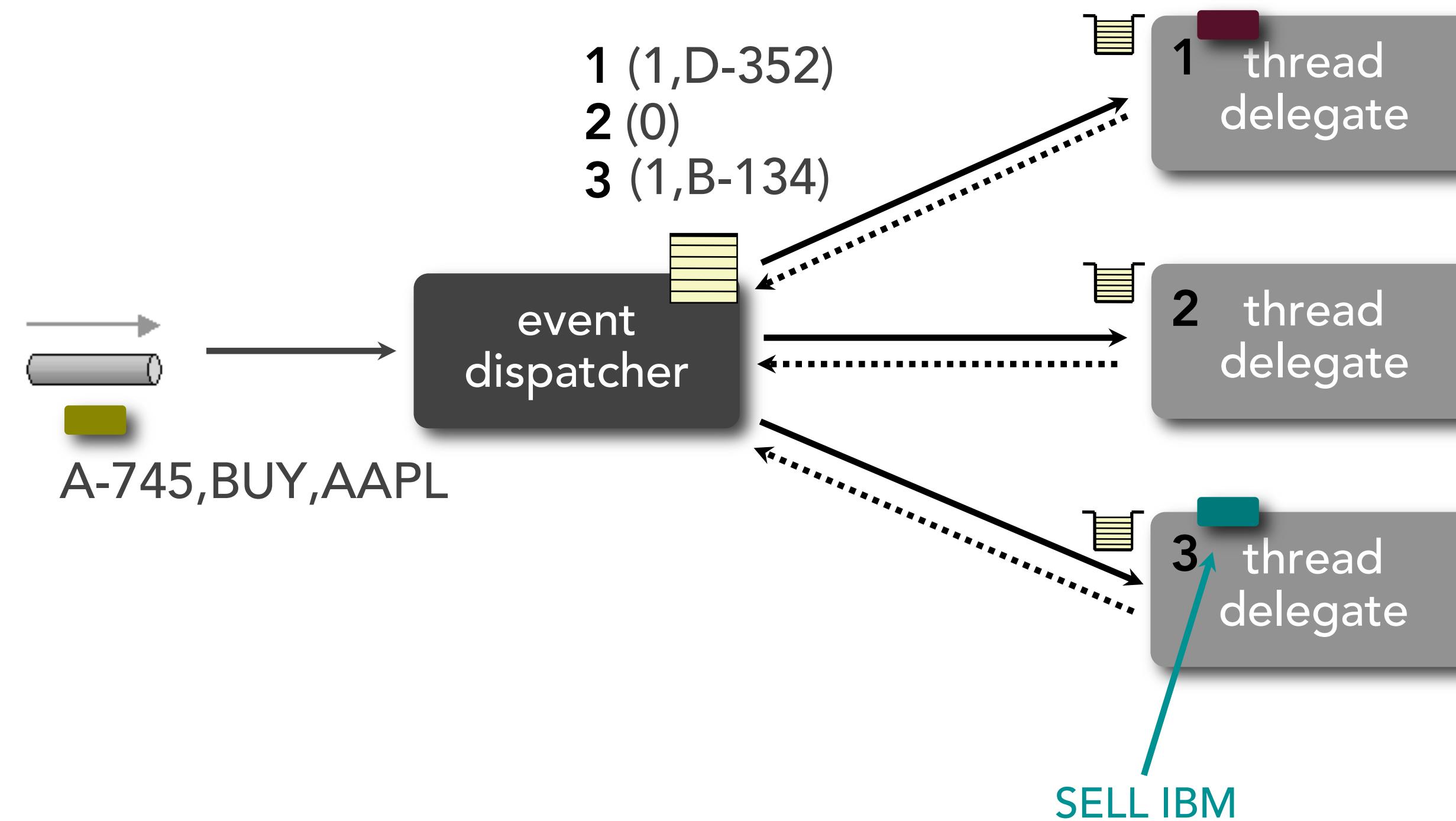
# thread delegate pattern



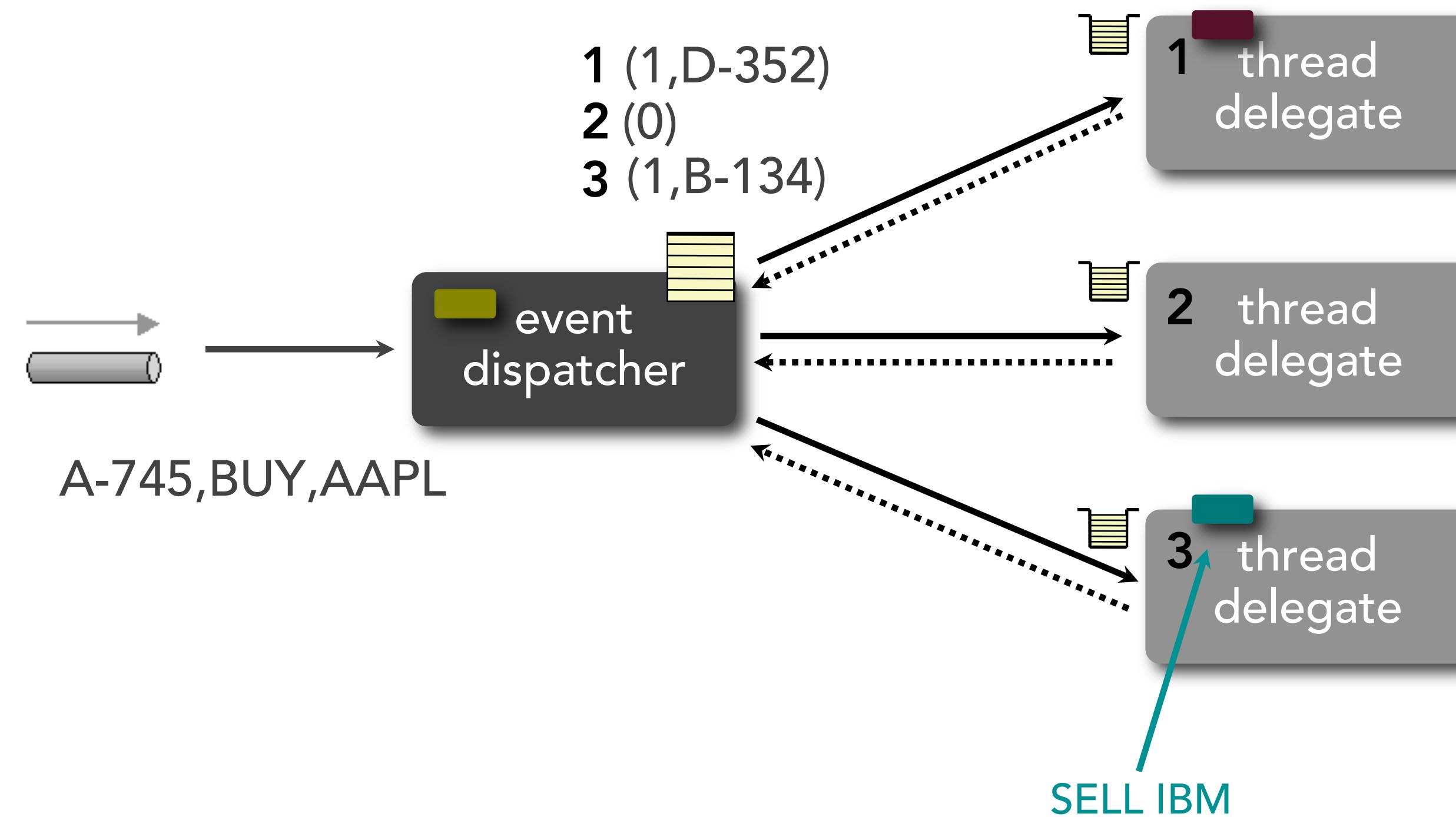
# thread delegate pattern



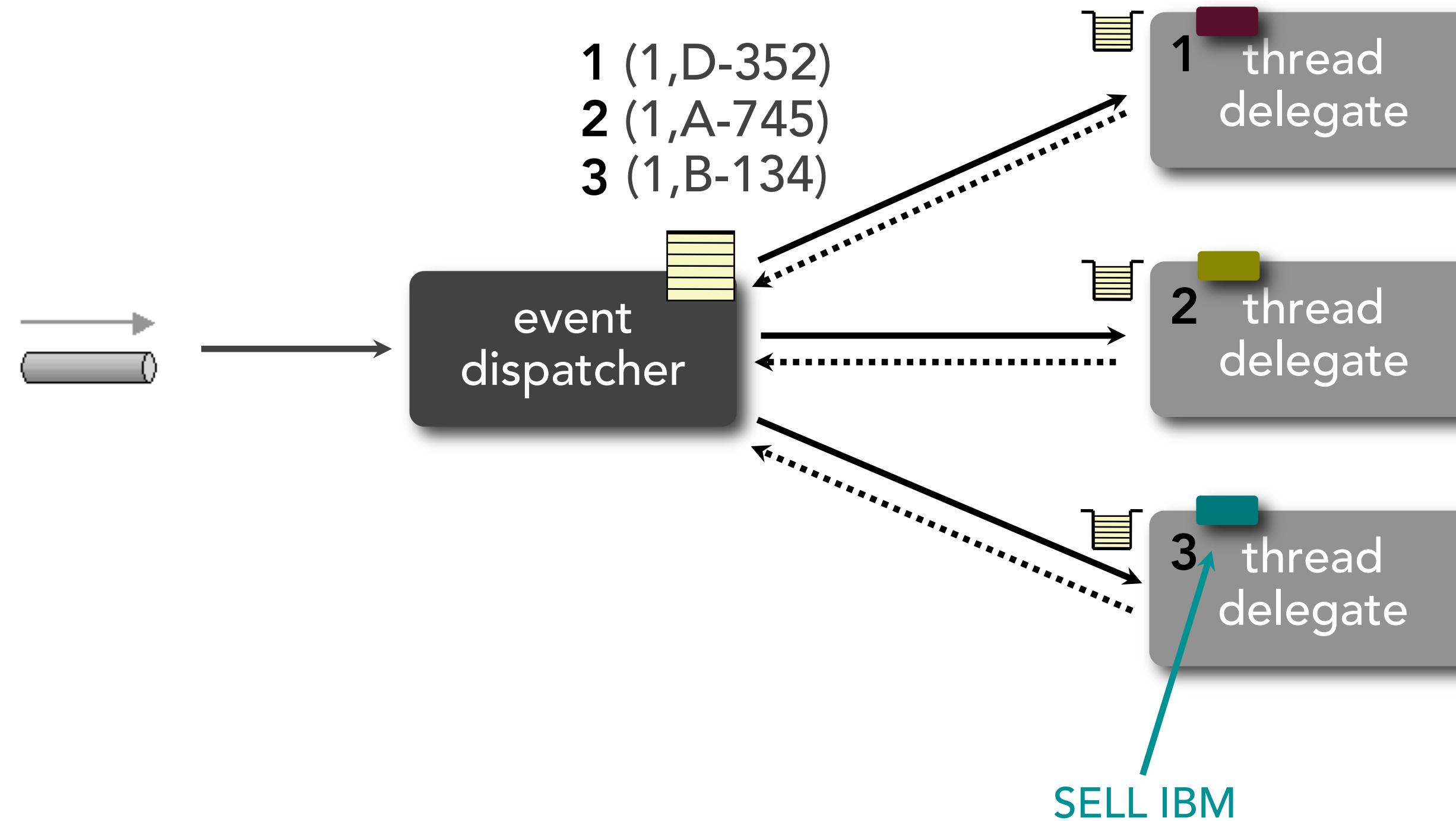
# thread delegate pattern



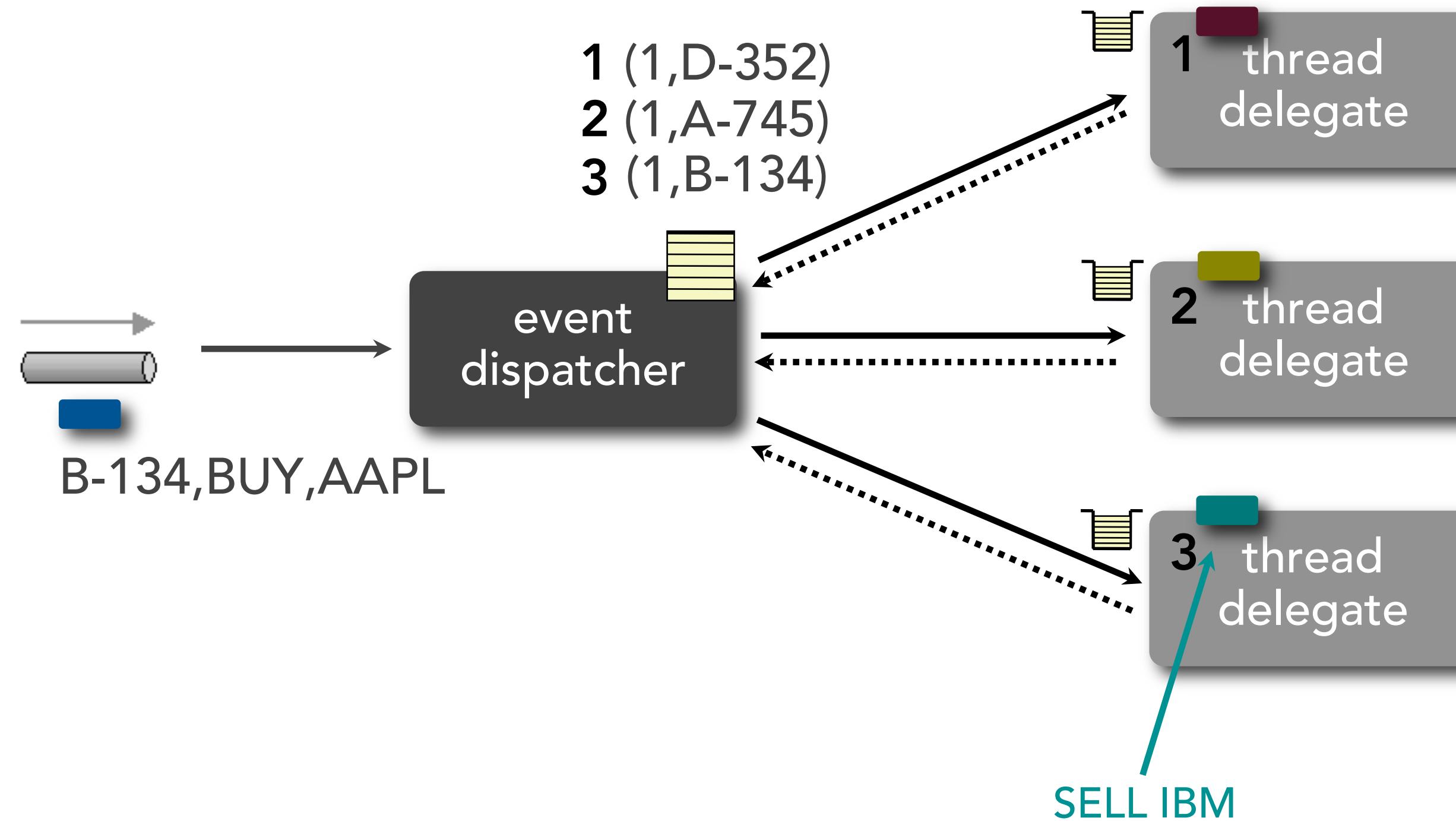
# thread delegate pattern



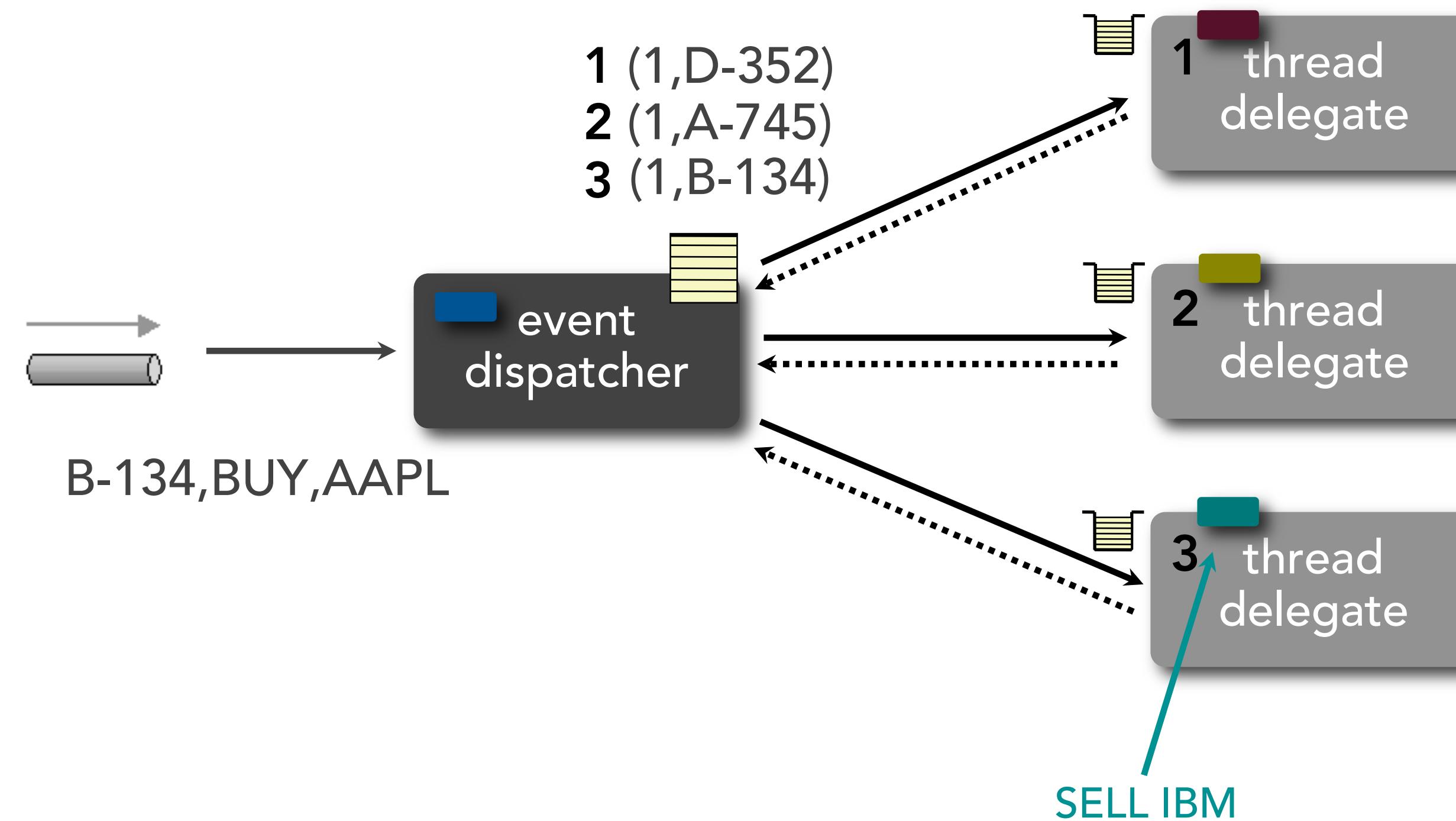
# thread delegate pattern



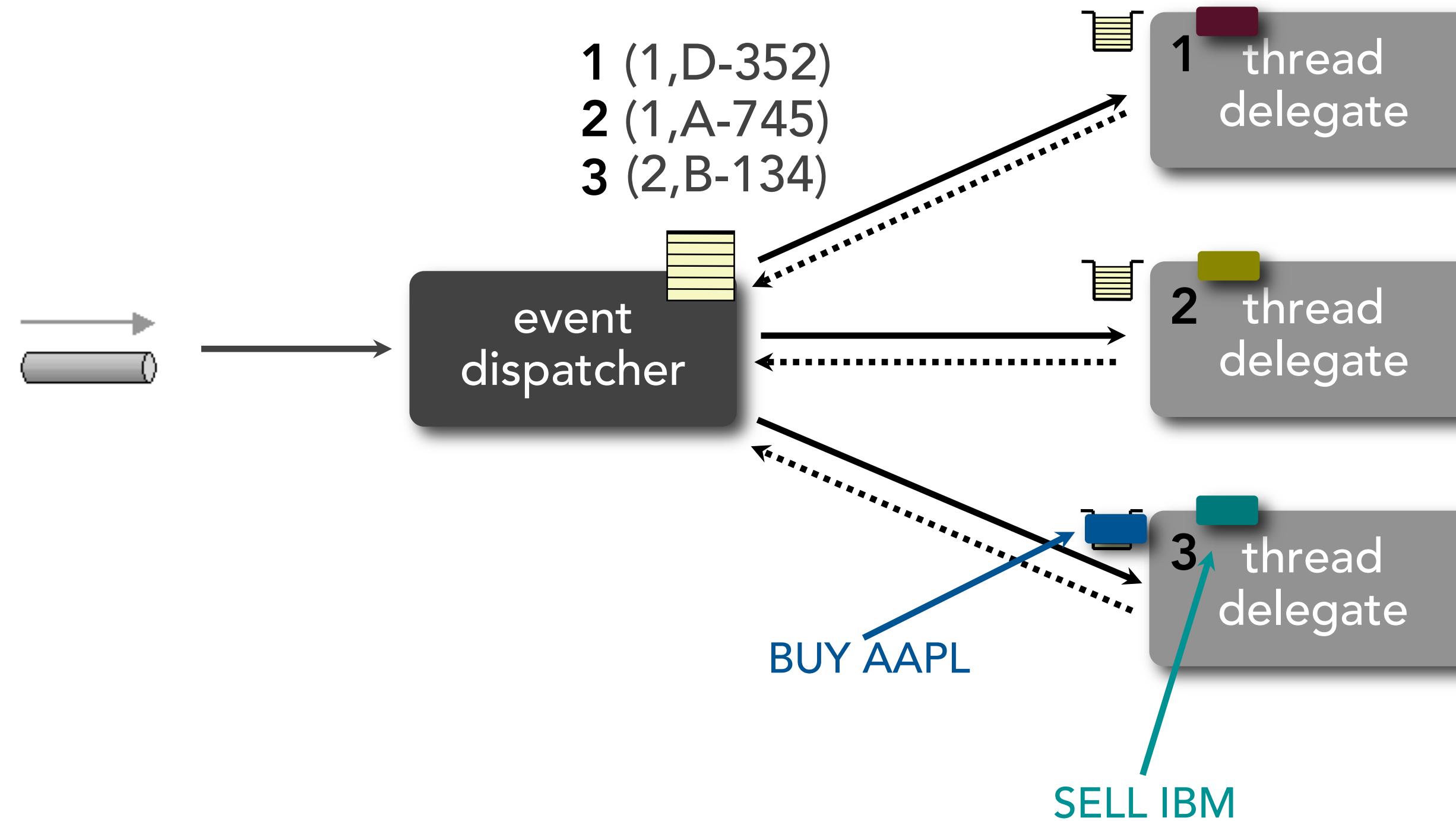
# thread delegate pattern



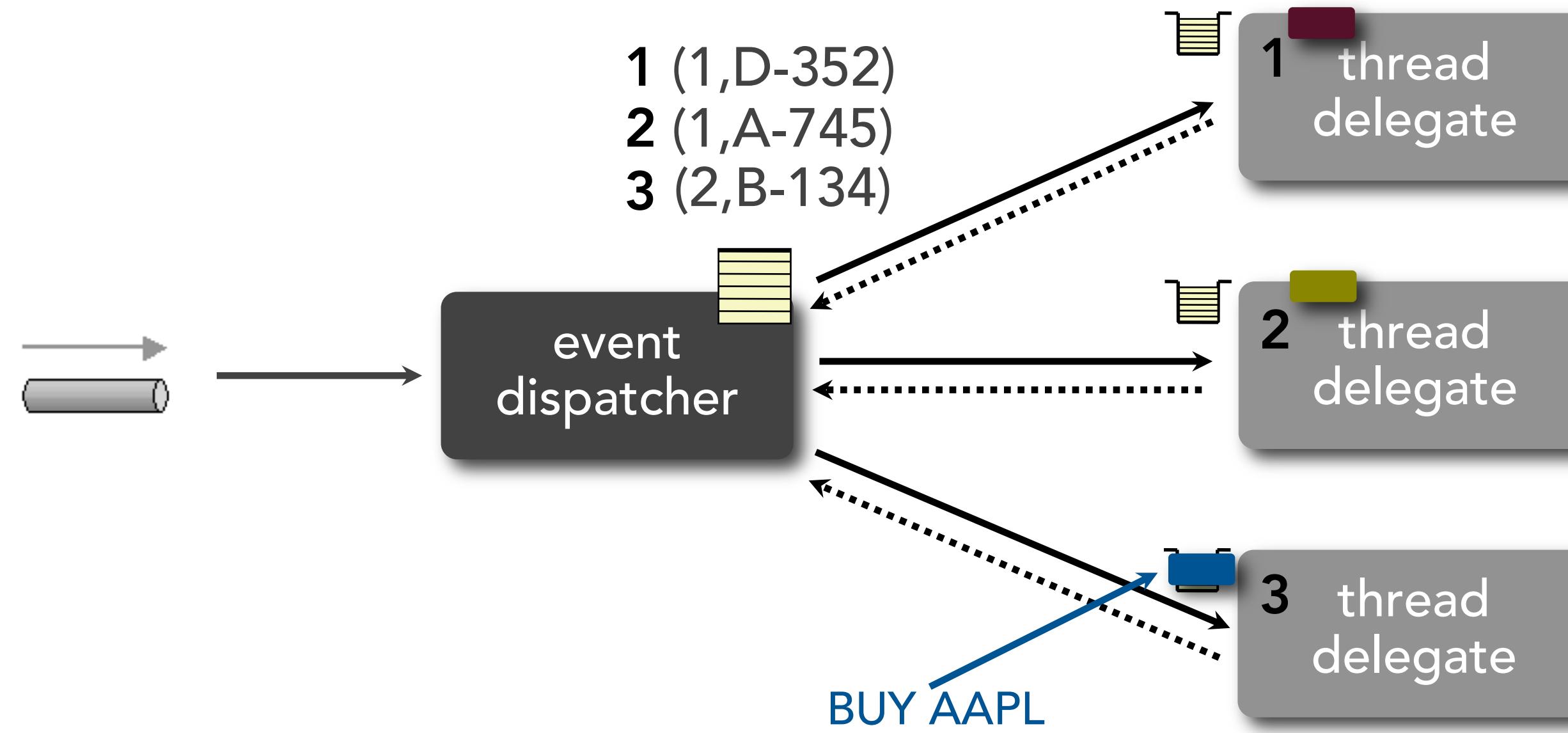
# thread delegate pattern



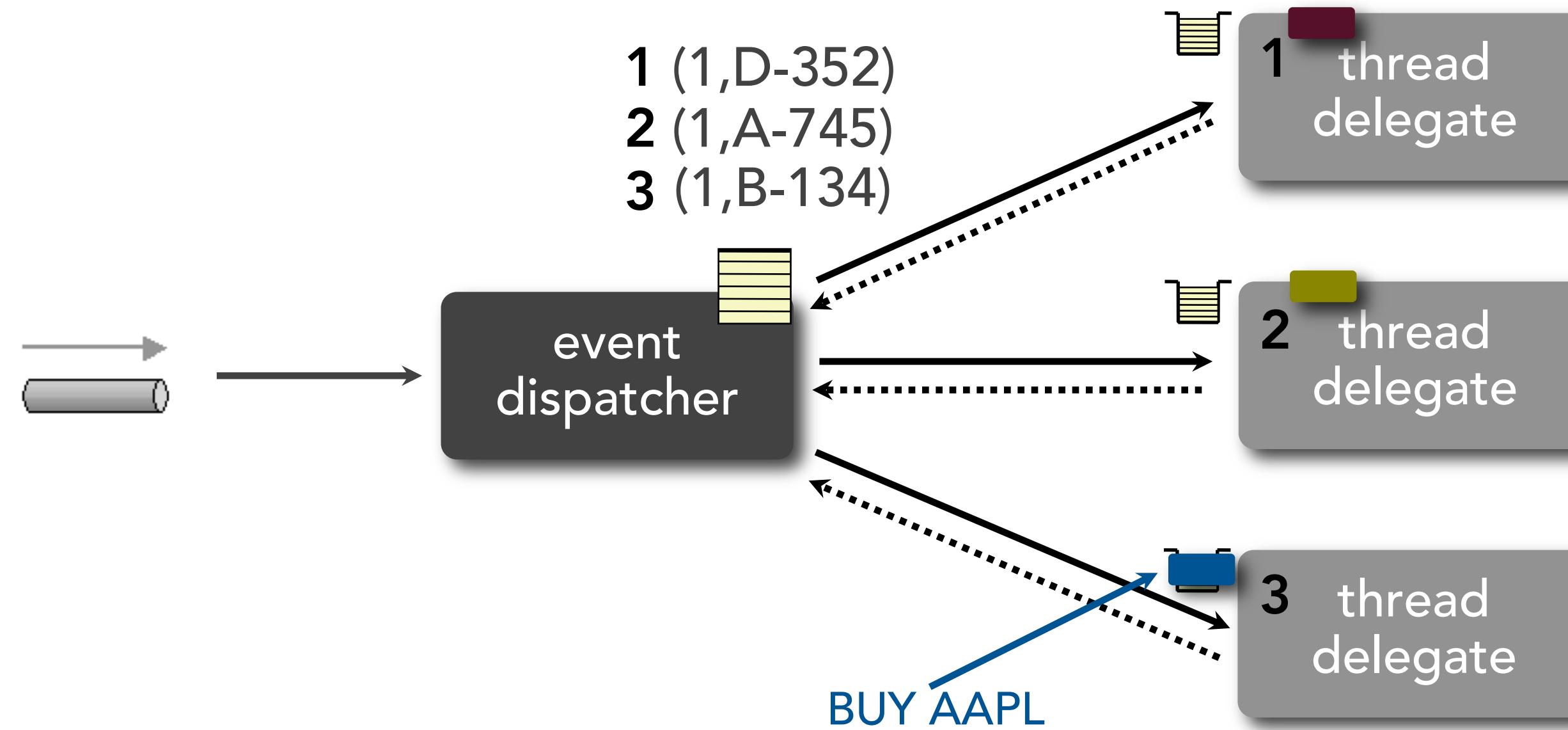
# thread delegate pattern



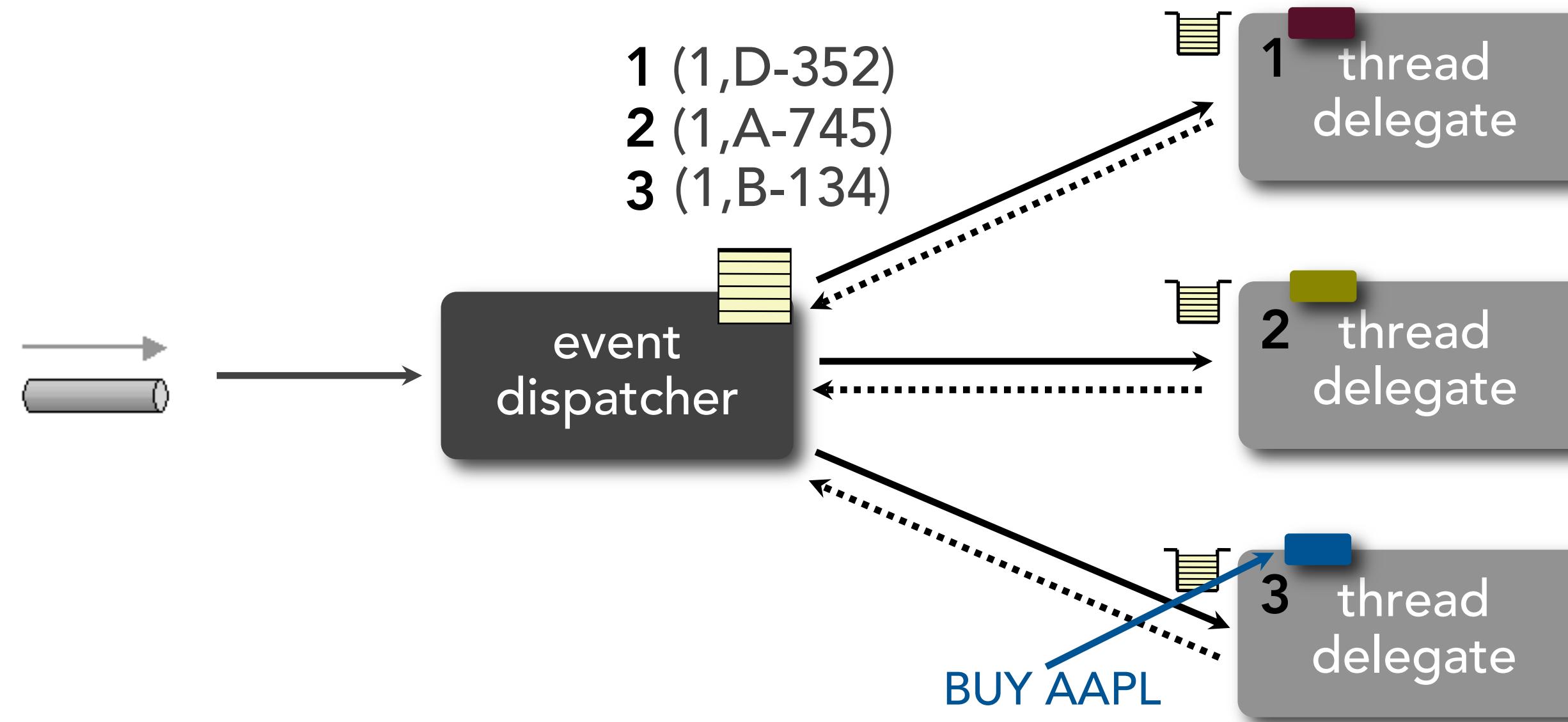
# thread delegate pattern



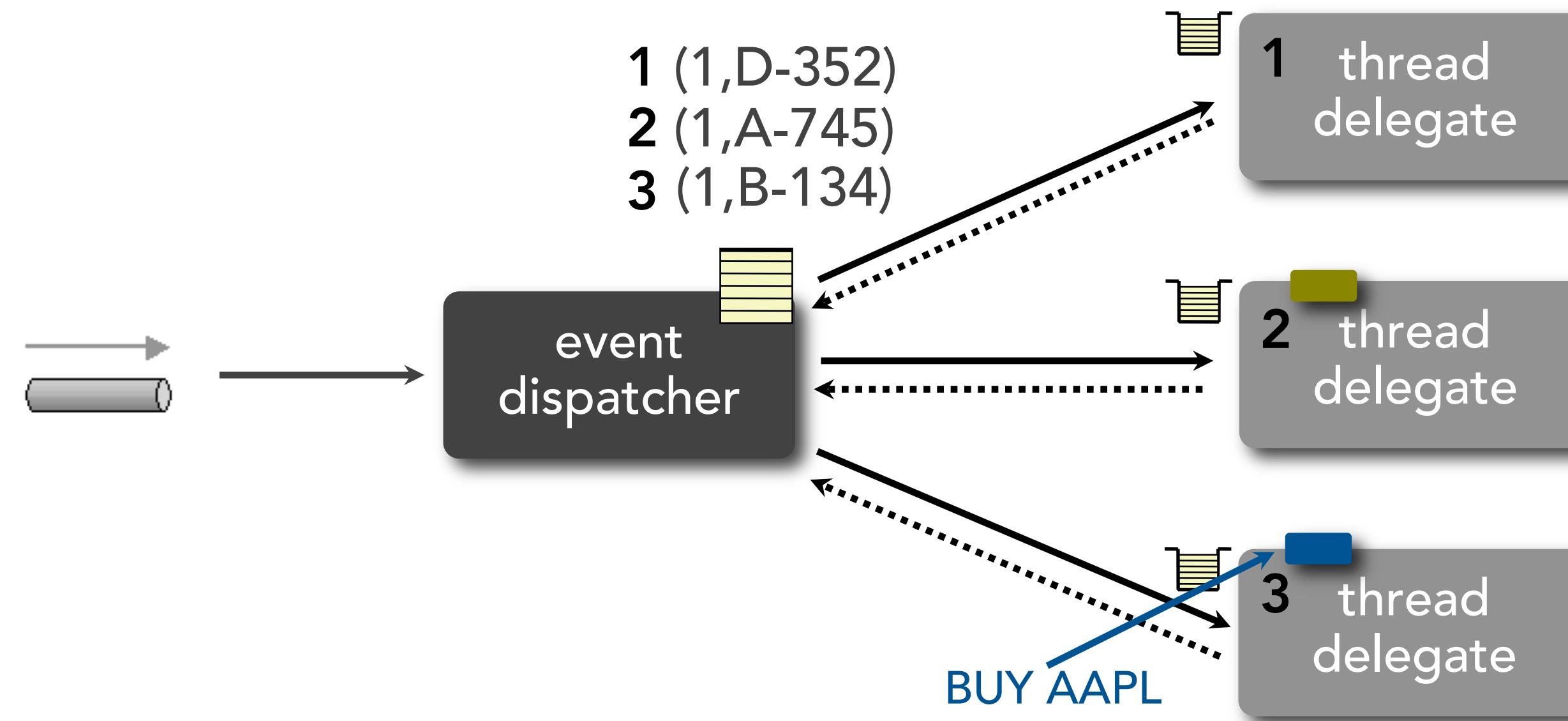
# thread delegate pattern



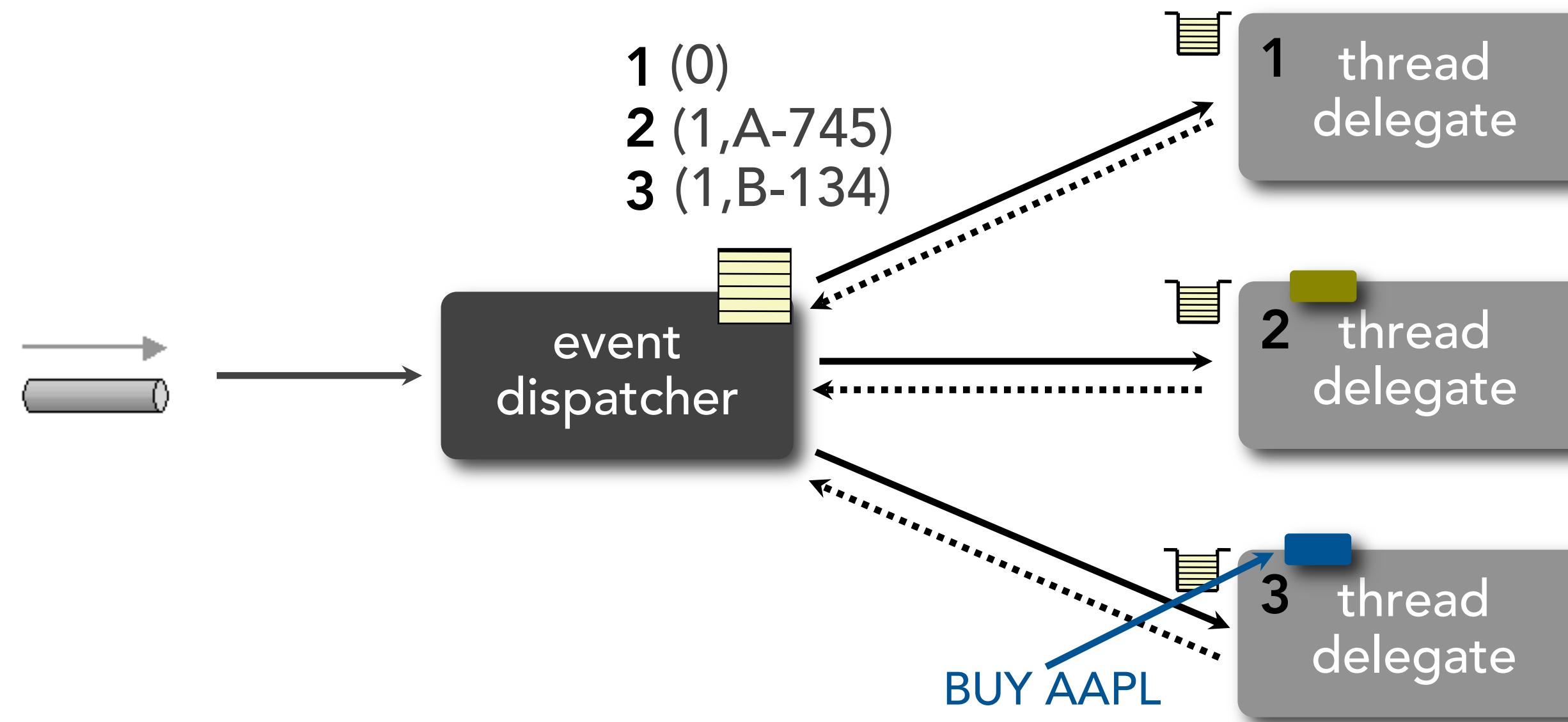
# thread delegate pattern



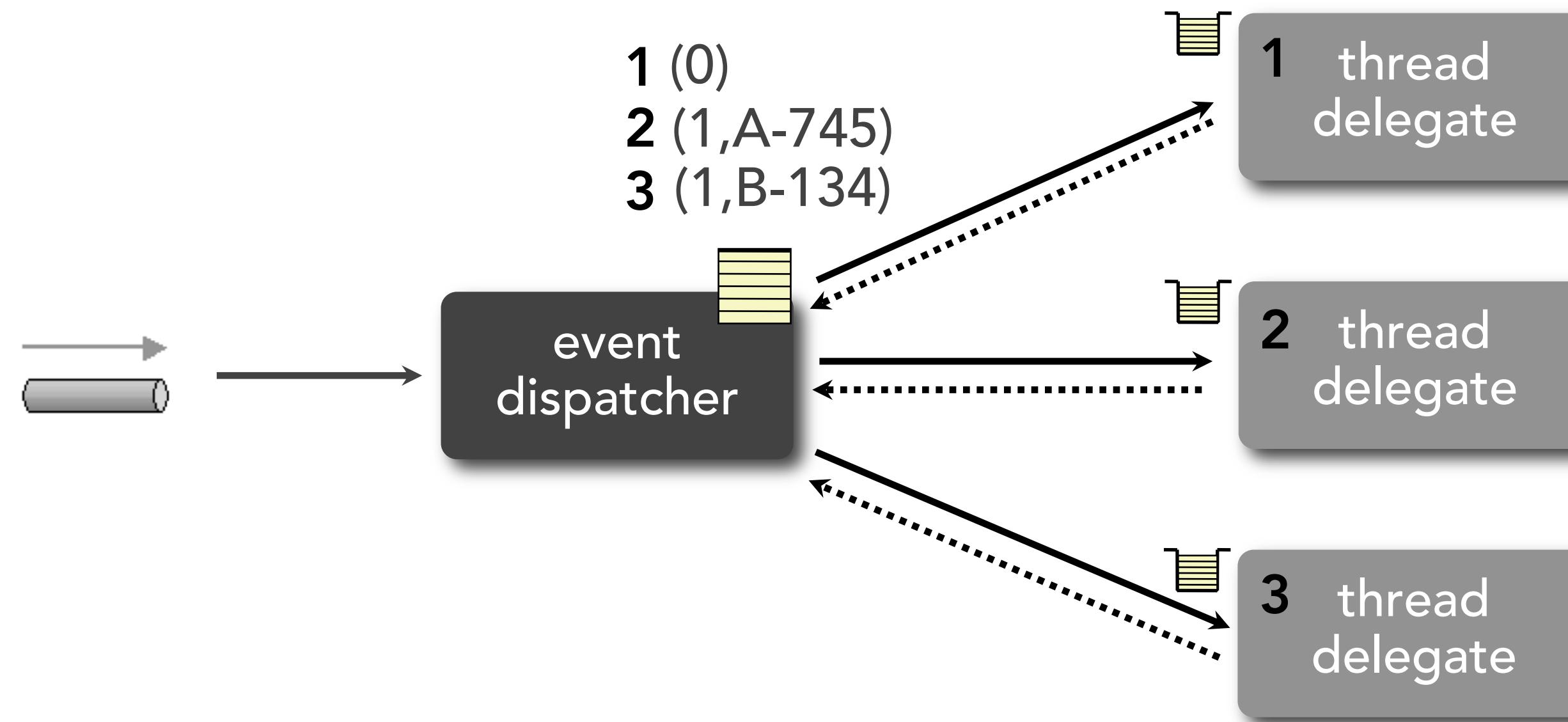
# thread delegate pattern



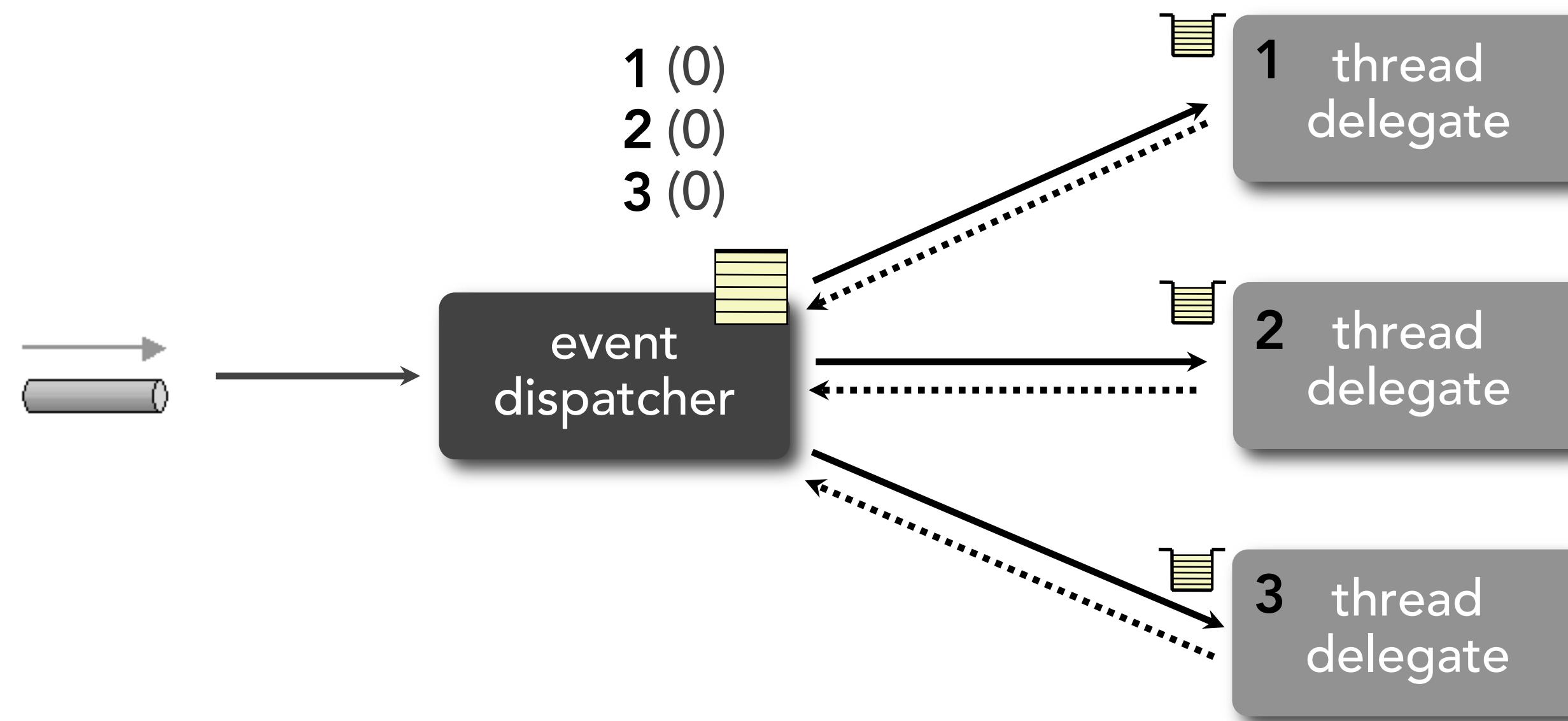
# thread delegate pattern



# thread delegate pattern



# thread delegate pattern



# thread delegate pattern

thread delegate

# thread delegate pattern

## thread delegate

```
private Map<Long, TradeProcessor> threadPool = new ConcurrentHashMap<Long, TradeProcessor>();
```

# thread delegate pattern

## thread delegate

```
private Map<Long, TradeProcessor> threadPool = new ConcurrentHashMap<Long, TradeProcessor>();
```

thread id



# thread delegate pattern

## thread delegate

```
private Map<Long, TradeProcessor> threadPool = new ConcurrentHashMap<Long, TradeProcessor>();
```

thread id

thread object



# thread delegate pattern

## thread delegate

```
private Map<Long, TradeProcessor> threadPool = new ConcurrentHashMap<Long, TradeProcessor>();
```

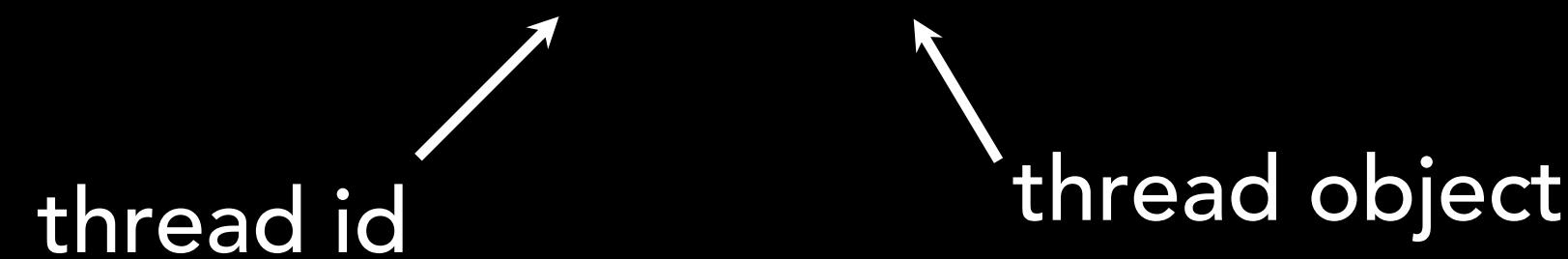


```
private Map<String, Long> allocationMap = new ConcurrentHashMap<String, Long>();
```

# thread delegate pattern

## thread delegate

```
private Map<Long, TradeProcessor> threadPool = new ConcurrentHashMap<Long, TradeProcessor>();
```



```
private Map<String, Long> allocationMap = new ConcurrentHashMap<String, Long>();
```



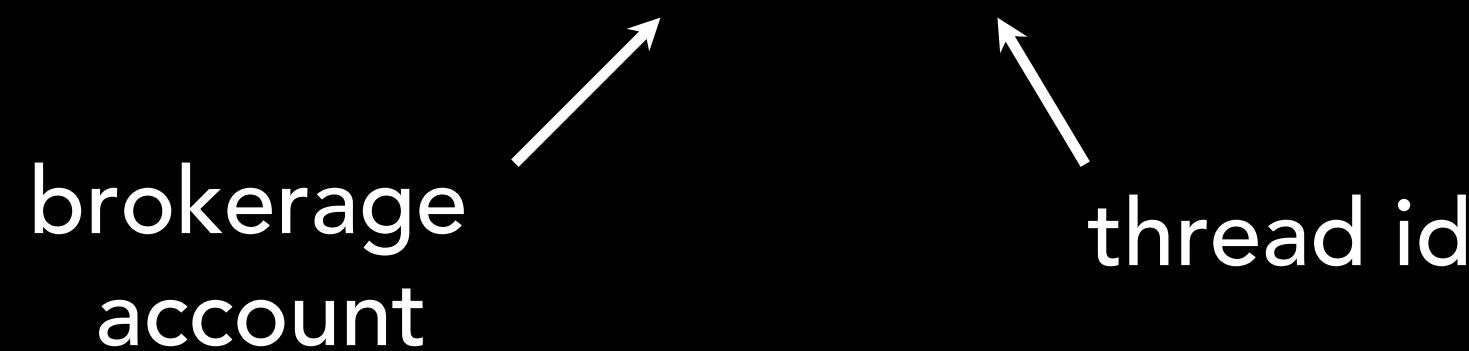
# thread delegate pattern

## thread delegate

```
private Map<Long, TradeProcessor> threadPool = new ConcurrentHashMap<Long, TradeProcessor>();
```



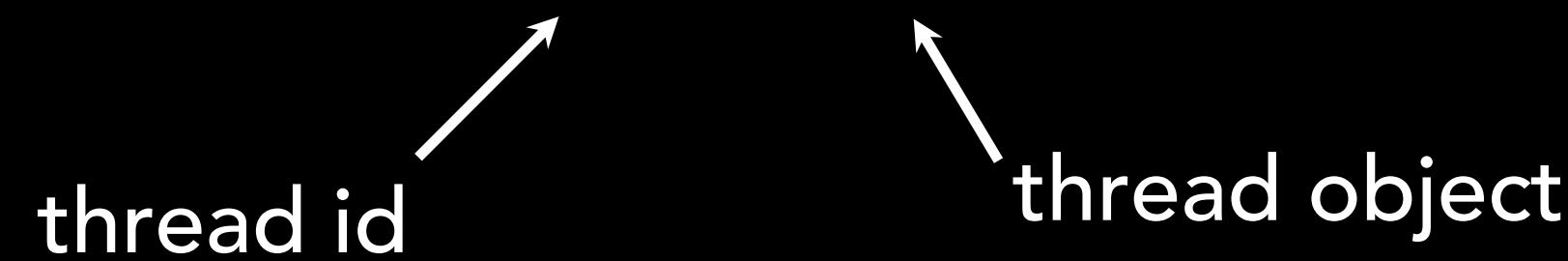
```
private Map<String, Long> allocationMap = new ConcurrentHashMap<String, Long>();
```



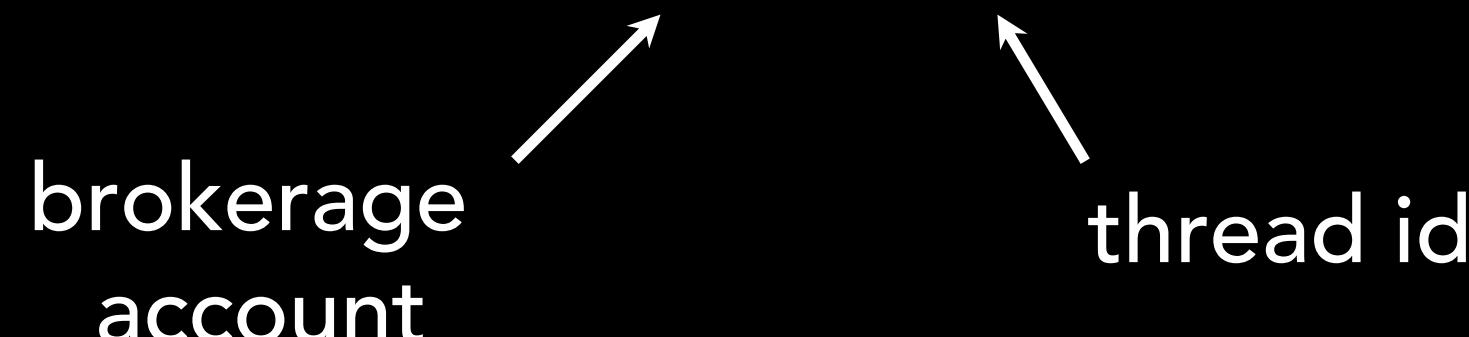
# thread delegate pattern

## thread delegate

```
private Map<Long, TradeProcessor> threadPool = new ConcurrentHashMap<Long, TradeProcessor>();
```



```
private Map<String, Long> allocationMap = new ConcurrentHashMap<String, Long>();
```

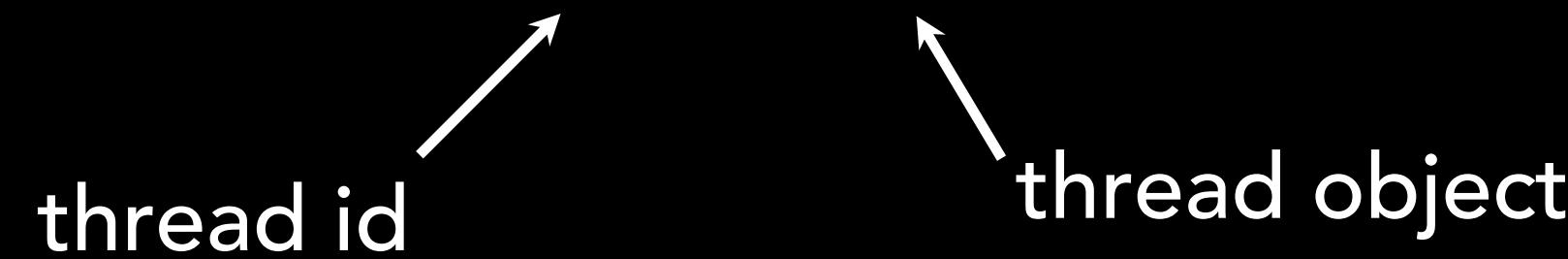


```
private Map<Long, Long> processingCountMap = new ConcurrentHashMap<Long, Long>();
```

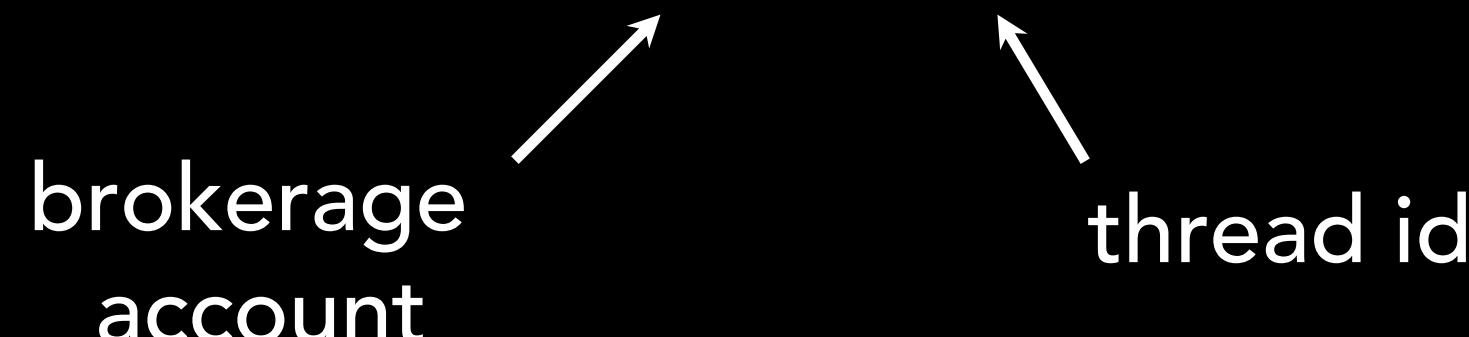
# thread delegate pattern

## thread delegate

```
private Map<Long, TradeProcessor> threadPool = new ConcurrentHashMap<Long, TradeProcessor>();
```



```
private Map<String, Long> allocationMap = new ConcurrentHashMap<String, Long>();
```



```
private Map<Long, Long> processingCountMap = new ConcurrentHashMap<Long, Long>();
```



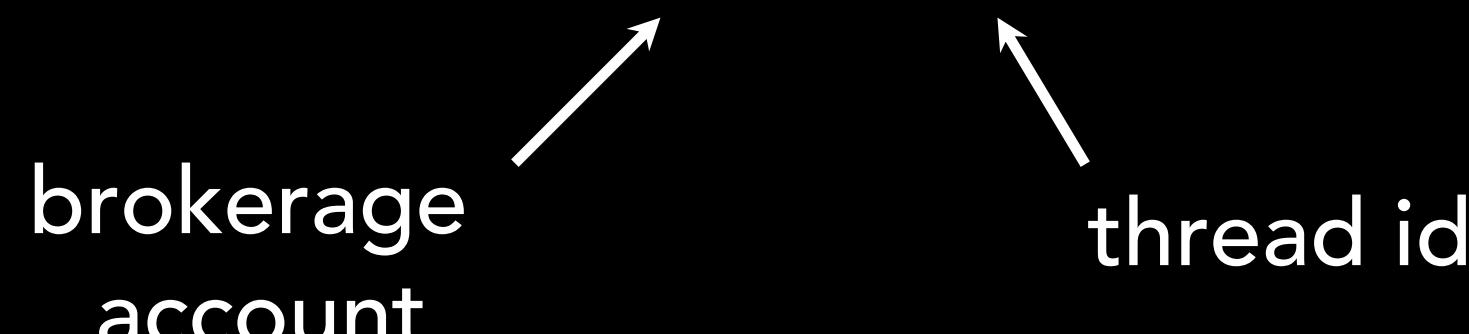
# thread delegate pattern

## thread delegate

```
private Map<Long, TradeProcessor> threadPool = new ConcurrentHashMap<Long, TradeProcessor>();
```



```
private Map<String, Long> allocationMap = new ConcurrentHashMap<String, Long>();
```



```
private Map<Long, Long> processingCountMap = new ConcurrentHashMap<Long, Long>();
```



# thread delegate pattern

## thread delegate

```
private Map<Long, TradeProcessor> threadPool
```

```
private Map<String, Long> allocationMap
```

```
private Map<Long, Long> processingCountMap
```

# thread delegate pattern

## thread delegate

```
private Map<Long, TradeProcessor> threadPool [1, @6d06d69c]  
[2, @7a24c42a]
```

```
private Map<String, Long> allocationMap
```

```
private Map<Long, Long> processingCountMap
```

# thread delegate pattern

## thread delegate

```
private Map<Long, TradeProcessor> threadPool [1, @6d06d69c]  
[2, @7a24c42a]
```

```
private Map<String, Long> allocationMap  
[B-387, 1]  
[G-186, 2]
```

```
private Map<Long, Long> processingCountMap
```

# thread delegate pattern

## thread delegate

```
private Map<Long, TradeProcessor> threadPool [1, @6d06d69c]  
[2, @7a24c42a]
```

```
private Map<String, Long> allocationMap  
[B-387, 1]  
[G-186, 2]
```

```
private Map<Long, Long> processingCountMap  
[1, 3]  
[2, 1]
```

# thread delegate pattern

## thread delegate

```
private Map<Long, TradeProcessor> threadPool [1, @6d06d69c]  
[2, @7a24c42a]
```

```
private Map<String, Long> allocationMap  
[B-387, 1]  
[G-186, 2]
```

```
private Map<Long, Long> processingCountMap = [  
    [1, 3],  
    [2, 1]
```

# thread delegate pattern

## thread delegate

```
private Map<Long, TradeProcessor> threadPool [1,@6d06d69c]  
[2, @7a24c42a]
```

```
private Map<String, Long> allocationMap
```

```
[B-387, 1]  
[G-186, 2]
```

```
private Map<Long, Long> processingCountMap
```

```
= [ ]
```

```
[1, 3]  
[2, 1]
```

# thread delegate pattern

## thread delegate

```
private Map<Long, TradeProcessor> threadPool [1,@6d06d69c]  
[2, @7a24c42a]
```

```
private Map<String, Long> allocationMap
```

```
[B-387, 1]  
[G-186, 2]
```

```
private Map<Long, Long> processingCountMap
```

```
= [1, ]
```

```
[1, 3]  
[2, 1]
```

# thread delegate pattern

## thread delegate

```
private Map<Long, TradeProcessor> threadPool  
    [1, @6d06d69c]  
    [2, @7a24c42a]
```

```
private Map<String, Long> allocationMap
```

```
[B-387, 1]  
[G-186, 2]
```

```
private Map<Long, Long> processingCountMap
```

```
[1, 3]  
[2, 1]
```

```
= [1, ]
```

# thread delegate pattern

## thread delegate

```
private Map<Long, TradeProcessor> threadPool  
[1, @6d06d69c]  
[2, @7a24c42a]
```

```
private Map<String, Long> allocationMap
```

```
[B-387, 1]  
[G-186, 2]
```

```
private Map<Long, Long> processingCountMap
```

```
[1, 3]  
[2, 1]
```

```
= [1, B-387, ]
```

# thread delegate pattern

## thread delegate

```
private Map<Long, TradeProcessor> threadPool  
    [1, @6d06d69c]  
    [2, @7a24c42a]
```

```
private Map<String, Long> allocationMap
```

```
[B-387, 1]  
[G-186, 2]
```

```
private Map<Long, Long> processingCountMap
```

```
[1, 3]  
[2, 1]
```

= [1, B-387, ]

# thread delegate pattern

## thread delegate

```
private Map<Long, TradeProcessor> threadPool  
    [1, @6d06d69c]  
    [2, @7a24c42a]
```

```
private Map<String, Long> allocationMap
```

```
[B-387, 1]  
[G-186, 2]
```

```
private Map<Long, Long> processingCountMap
```

```
[1, 3]  
[2, 1]
```

= [1, B-387, 3]

# thread delegate pattern

## thread delegate

```
private Map<Long, TradeProcessor> threadPool
```

[1, @6d06d69c]  
[2, @7a24c42a]

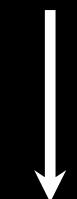
```
private Map<String, Long> allocationMap
```

[B-387, 1]  
[G-186, 2]

```
private Map<Long, Long> processingCountMap
```

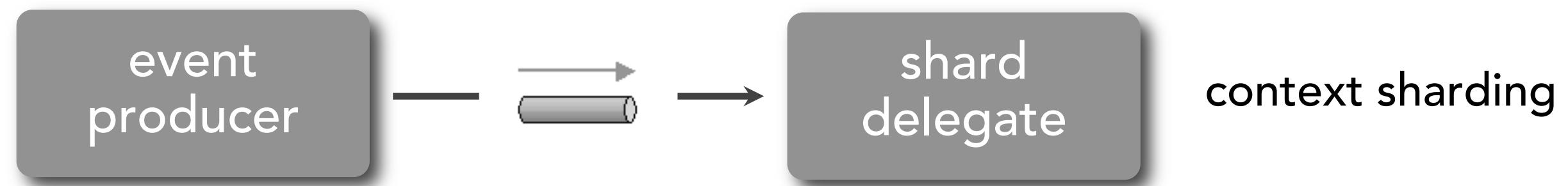
[1, 3]  
[2, 1]

thread 1 is currently  
processing 3 trades for  
brokerage account B-387

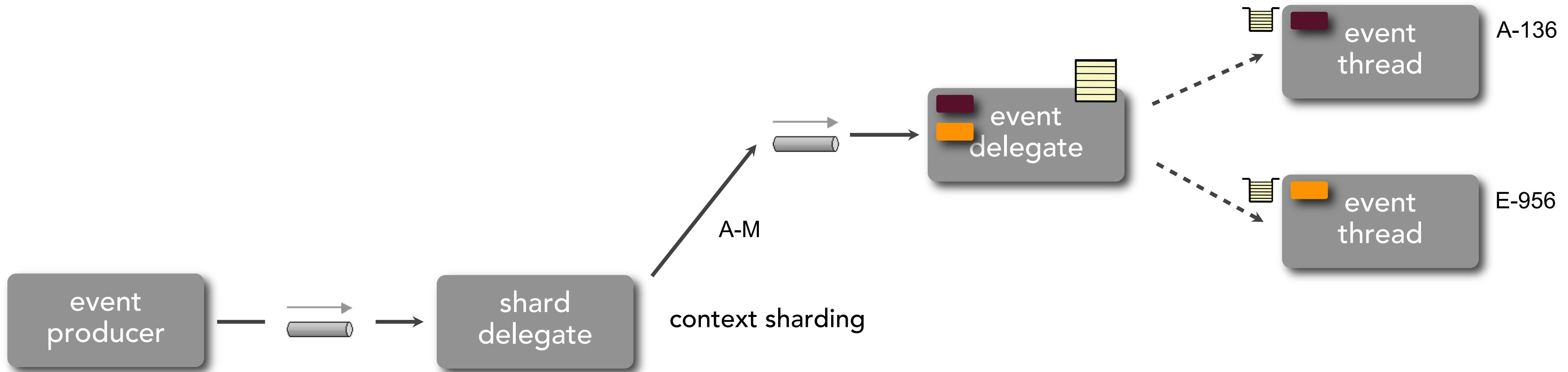


= [1, B-387, 3]

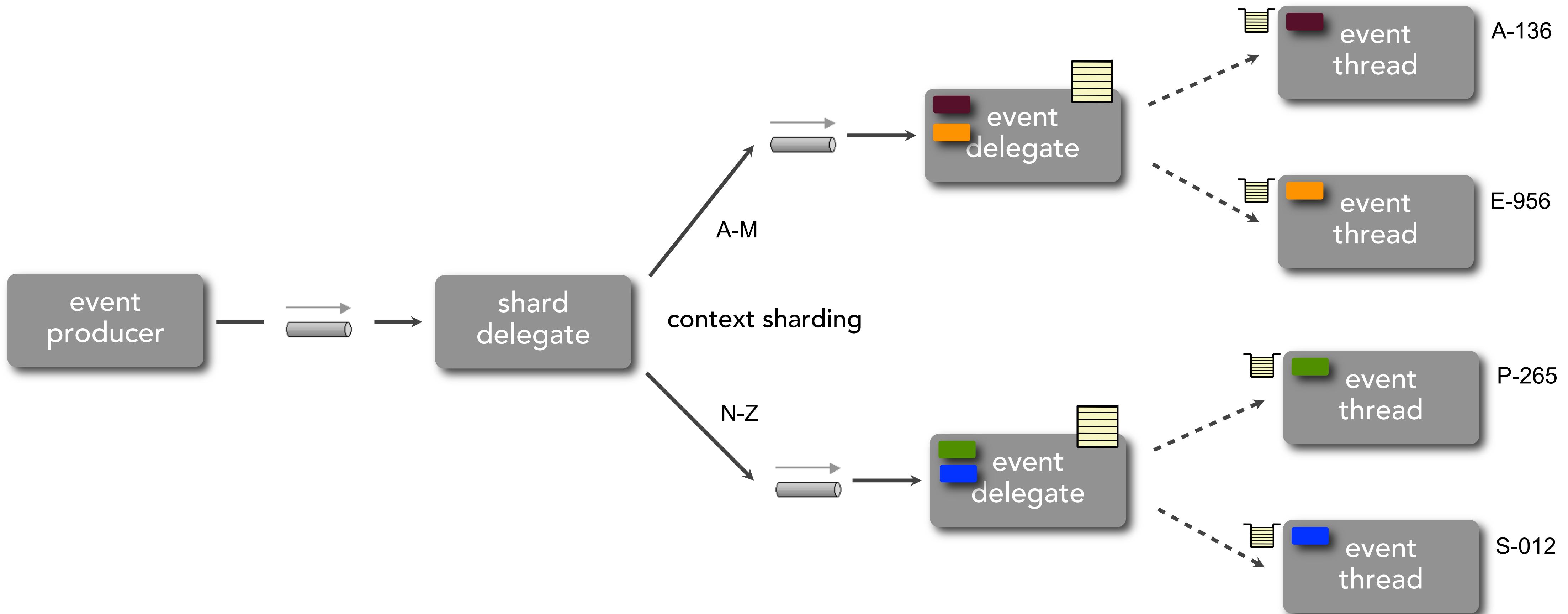
# thread delegate pattern



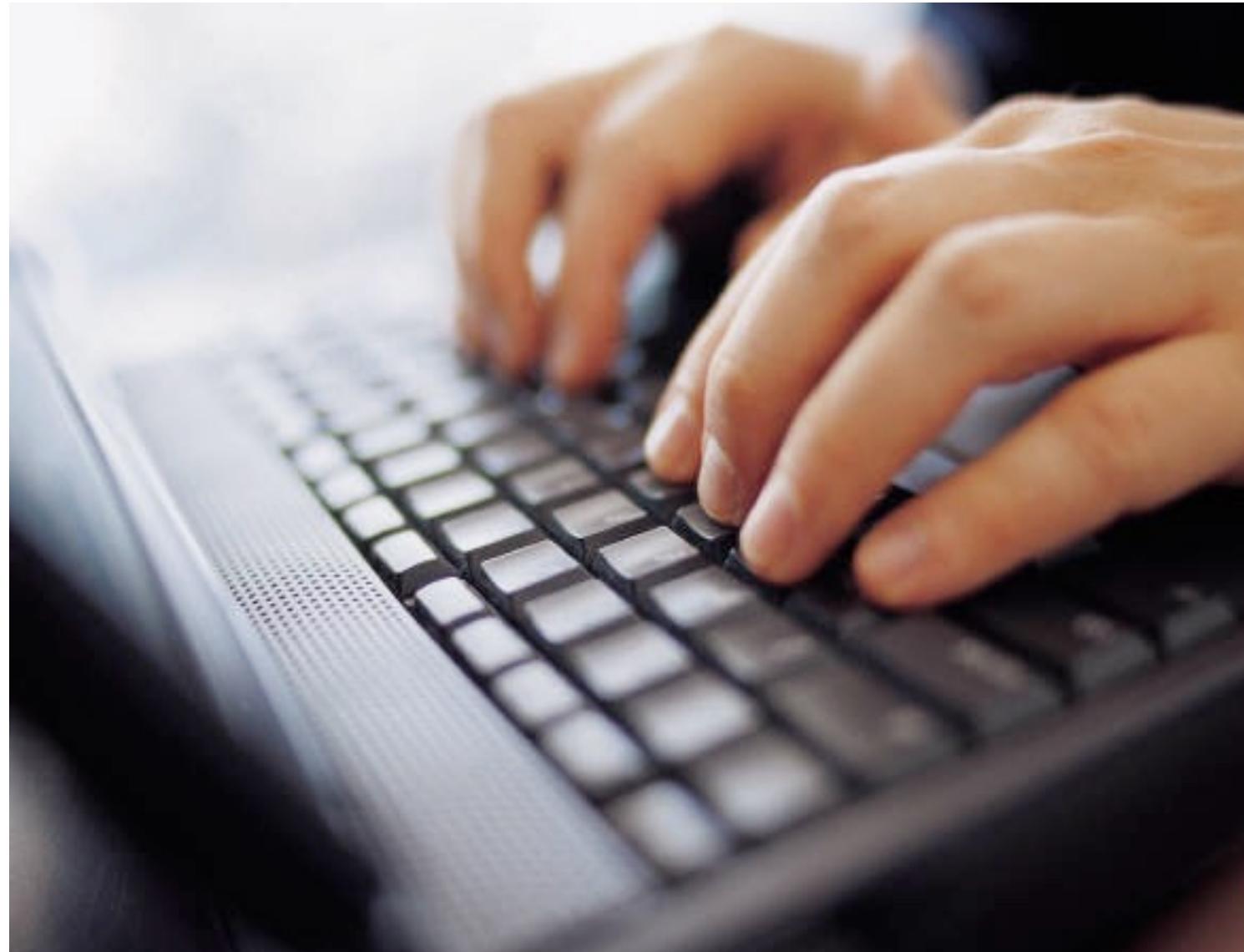
# thread delegate pattern



# thread delegate pattern



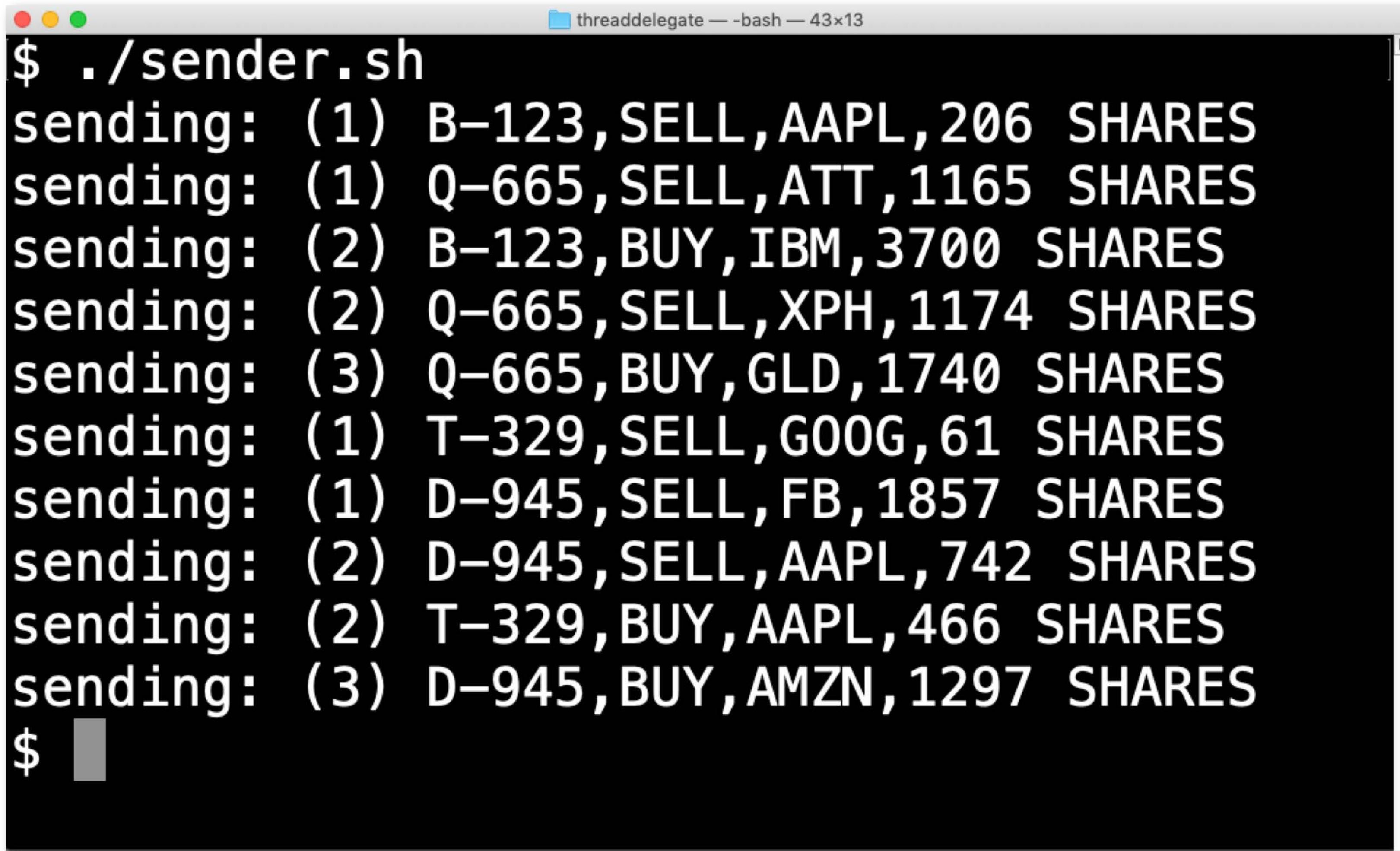
# thread delegate pattern



let's apply the pattern...

<https://github.com/wmr513/reactive/tree/master/threaddelegate>

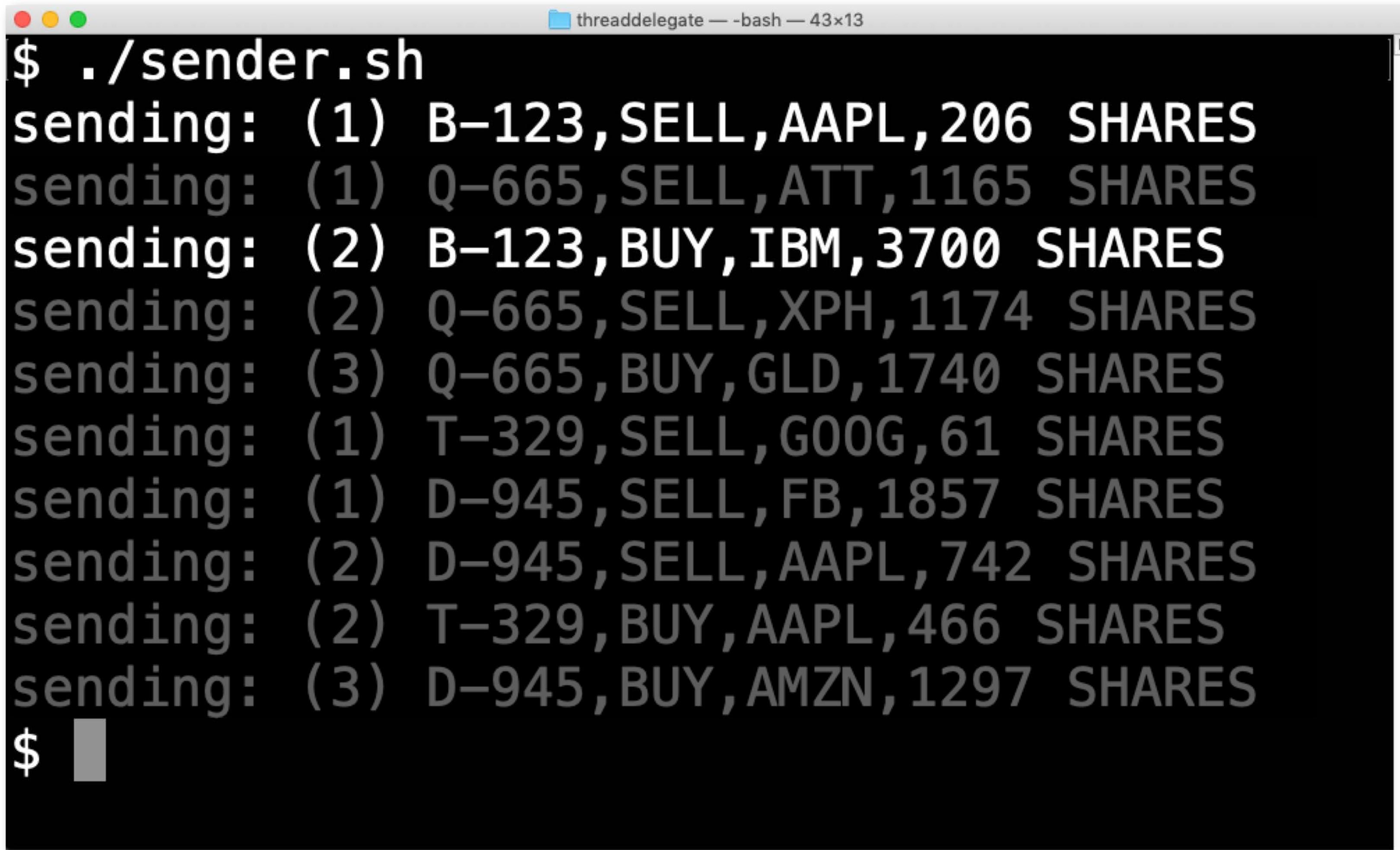
# thread delegate pattern



A terminal window titled "threaddelegate — bash — 43x13" displays the output of a shell script named "sender.sh". The script uses a thread delegate pattern to handle multiple concurrent tasks. The output shows the script sending various financial transactions (orders) to a system. The transactions include sell orders for AAPL, ATT, and FB, and buy orders for IBM, XPH, GLD, GOOG, and AMZN. The script sends three orders for each type of transaction.

```
$ ./sender.sh
sending: (1) B-123,SELL,AAPL,206 SHARES
sending: (1) Q-665,SELL,ATT,1165 SHARES
sending: (2) B-123,BUY,IBM,3700 SHARES
sending: (2) Q-665,SELL,XPH,1174 SHARES
sending: (3) Q-665,BUY,GLD,1740 SHARES
sending: (1) T-329,SELL,GOOG,61 SHARES
sending: (1) D-945,SELL,FB,1857 SHARES
sending: (2) D-945,SELL,AAPL,742 SHARES
sending: (2) T-329,BUY,AAPL,466 SHARES
sending: (3) D-945,BUY,AMZN,1297 SHARES
$
```

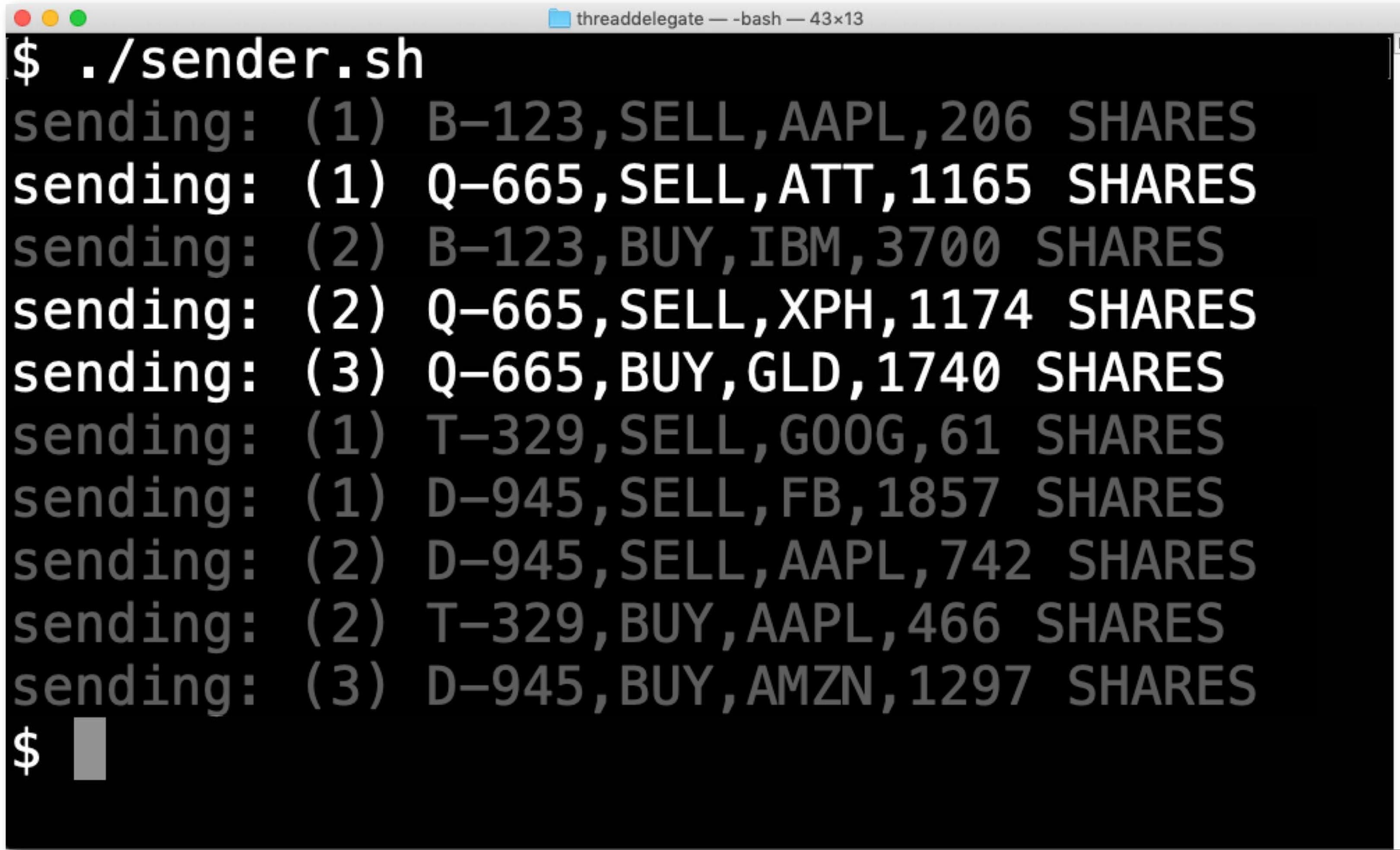
# thread delegate pattern



A terminal window titled "threaddelegate — bash — 43x13" is shown. The window has a black background and white text. It displays the command \$ ./sender.sh followed by ten lines of output starting with "sending:". Each line contains a transaction record with fields: identifier (B/Q/T/D), action (SELL or BUY), stock symbol (AAPL, ATT, IBM, XPH, GLD, GOOG, FB, AAPL, AMZN), and quantity (206, 1165, 3700, 1174, 1740, 61, 1857, 742, 466, 1297) SHARES.

```
$ ./sender.sh
sending: (1) B-123,SELL,AAPL,206 SHARES
sending: (1) Q-665,SELL,ATT,1165 SHARES
sending: (2) B-123,BUY,IBM,3700 SHARES
sending: (2) Q-665,SELL,XPH,1174 SHARES
sending: (3) Q-665,BUY,GLD,1740 SHARES
sending: (1) T-329,SELL,GOOG,61 SHARES
sending: (1) D-945,SELL,FB,1857 SHARES
sending: (2) D-945,SELL,AAPL,742 SHARES
sending: (2) T-329,BUY,AAPL,466 SHARES
sending: (3) D-945,BUY,AMZN,1297 SHARES
$ █
```

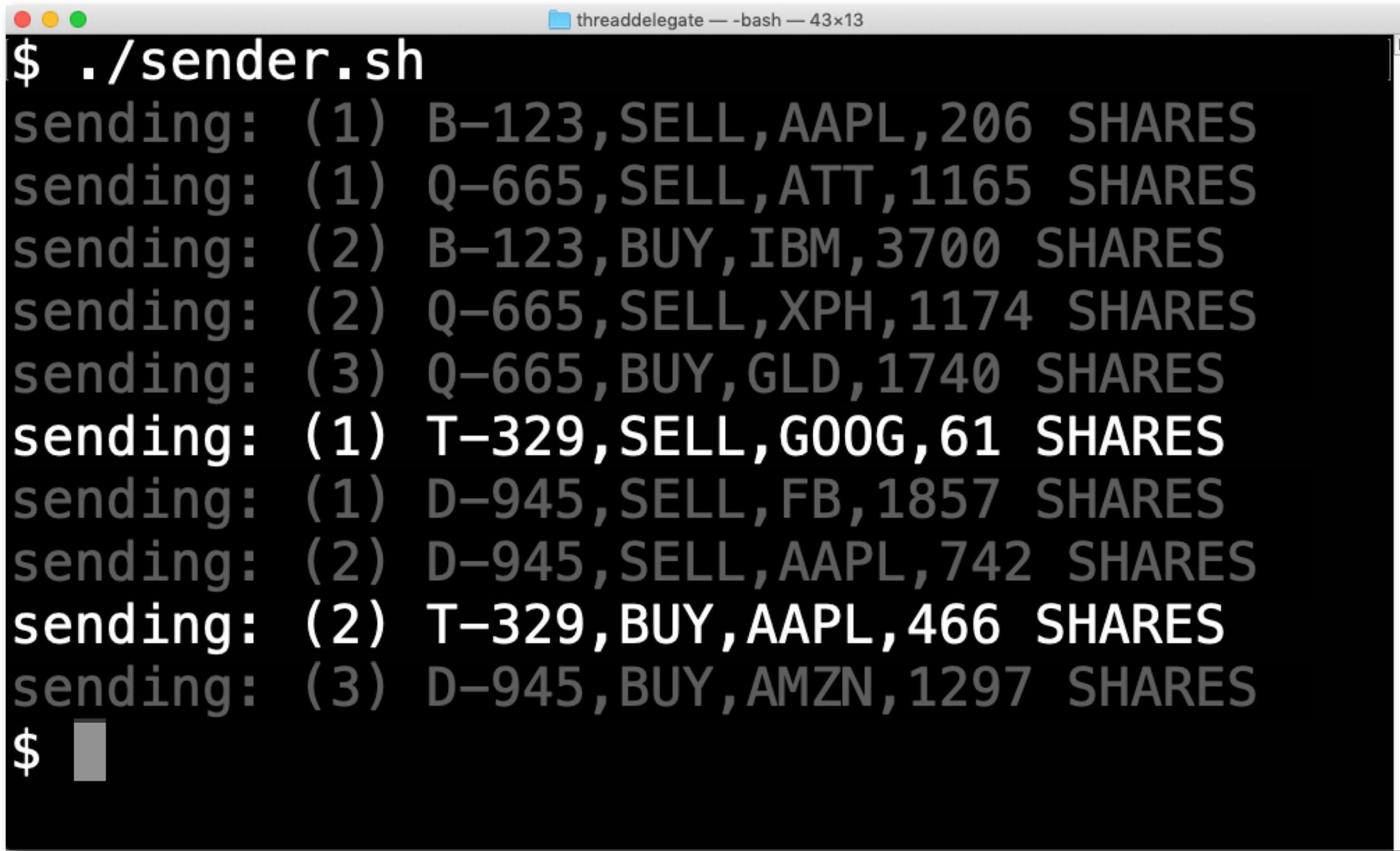
# thread delegate pattern



A terminal window titled "threaddelegate — bash — 43x13" is shown. The window contains the command \$ ./sender.sh followed by ten lines of output. Each line starts with the word "sending:" followed by a transaction identifier (e.g., 1, 2, 3, T, D) and a comma-separated list of stock symbols and quantities. The symbols include AAPL, ATT, IBM, XPH, GLD, GOOG, FB, AAPL, AMZN, and another AAPL entry.

```
$ ./sender.sh
sending: (1) B-123,SELL,AAPL,206 SHARES
sending: (1) Q-665,SELL,ATT,1165 SHARES
sending: (2) B-123,BUY,IBM,3700 SHARES
sending: (2) Q-665,SELL,XPH,1174 SHARES
sending: (3) Q-665,BUY,GLD,1740 SHARES
sending: (1) T-329,SELL,GOOG,61 SHARES
sending: (1) D-945,SELL,FB,1857 SHARES
sending: (2) D-945,SELL,AAPL,742 SHARES
sending: (2) T-329,BUY,AAPL,466 SHARES
sending: (3) D-945,BUY,AMZN,1297 SHARES
$ █
```

# thread delegate pattern

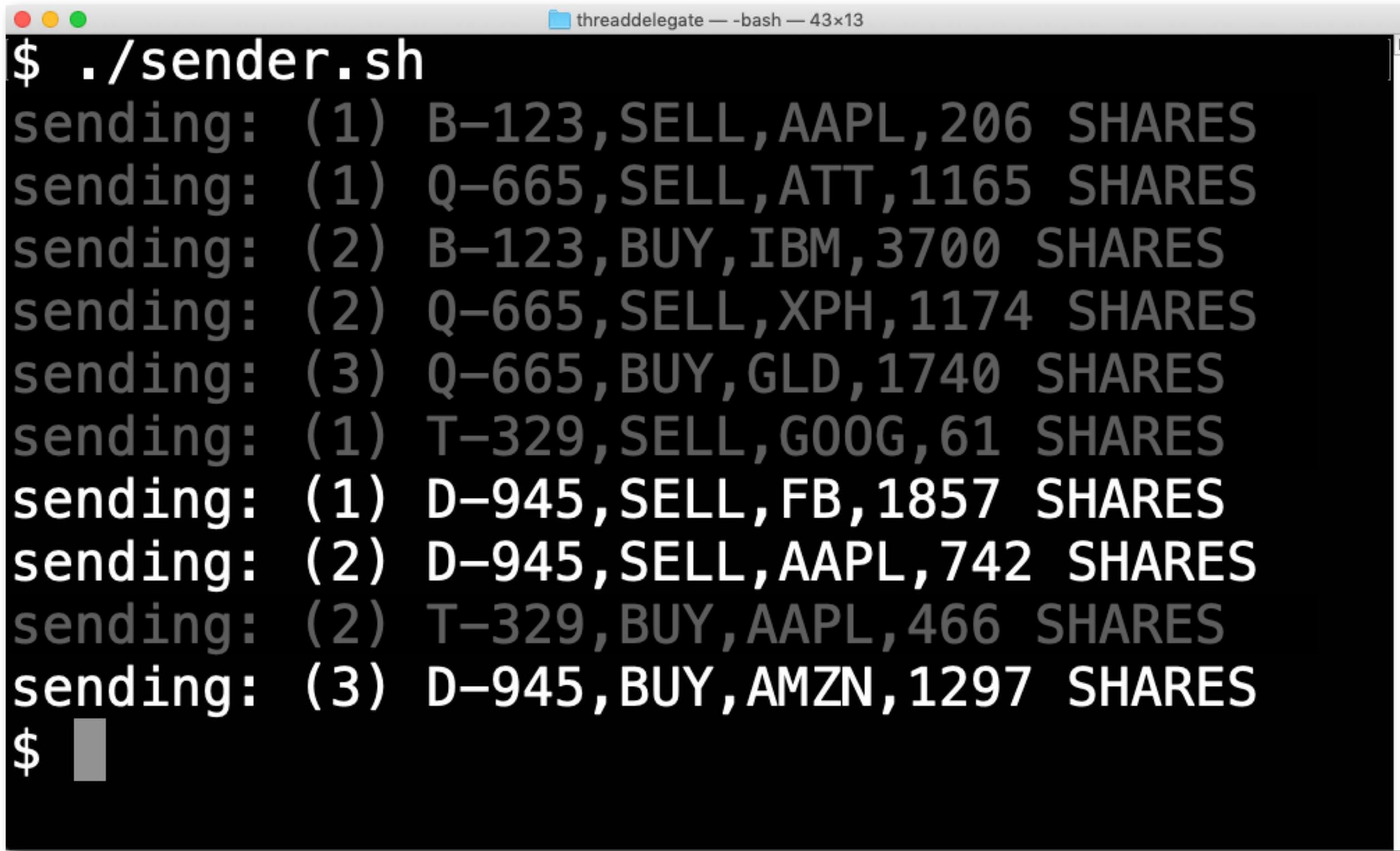


A terminal window titled "threaddelegate — bash — 43x13" is shown. The window contains the command \$ ./sender.sh followed by ten lines of output. Each line starts with the word "sending:" followed by a transaction record in parentheses. The records are:

- (1) B-123,SELL,AAPL,206 SHARES
- (1) Q-665,SELL,ATT,1165 SHARES
- (2) B-123,BUY,IBM,3700 SHARES
- (2) Q-665,SELL,XPH,1174 SHARES
- (3) Q-665,BUY,GLD,1740 SHARES
- (1) T-329,SELL,GOOG,61 SHARES
- (1) D-945,SELL,FB,1857 SHARES
- (2) D-945,SELL,AAPL,742 SHARES
- (2) T-329,BUY,AAPL,466 SHARES
- (3) D-945,BUY,AMZN,1297 SHARES

The terminal prompt \$ is visible at the bottom left.

# thread delegate pattern



A terminal window titled "threaddelegate — bash — 43x13" is shown. The window contains the command \$ ./sender.sh followed by ten lines of output. Each line starts with the word "sending:" followed by a transaction record in parentheses. The records are:

- (1) B-123,SELL,AAPL,206 SHARES
- (1) Q-665,SELL,ATT,1165 SHARES
- (2) B-123,BUY,IBM,3700 SHARES
- (2) Q-665,SELL,XPH,1174 SHARES
- (3) Q-665,BUY,GLD,1740 SHARES
- (1) T-329,SELL,GOOG,61 SHARES
- (1) D-945,SELL,FB,1857 SHARES
- (2) D-945,SELL,AAPL,742 SHARES
- (2) T-329,BUY,AAPL,466 SHARES
- (3) D-945,BUY,AMZN,1297 SHARES

The terminal prompt \$ is visible at the bottom left.

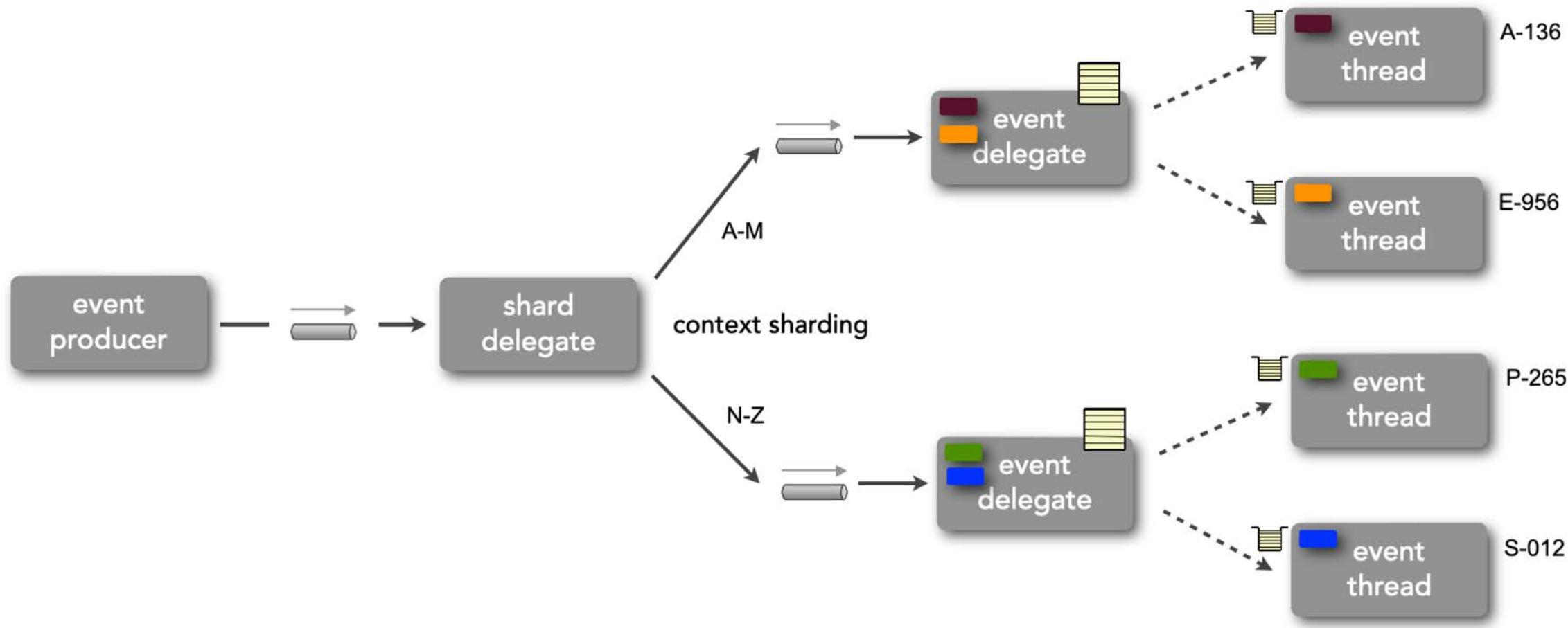
# thread delegate pattern



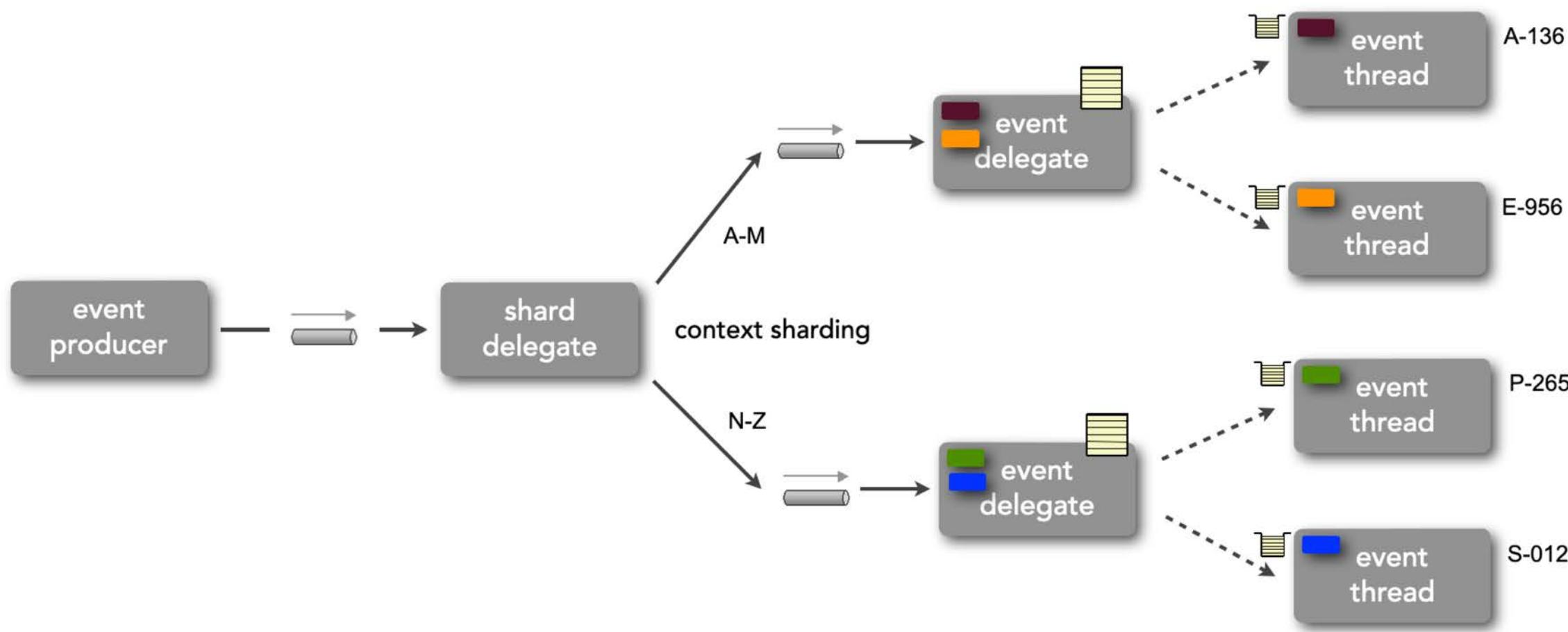
let's apply the pattern...

<https://github.com/wmr513/reactive/tree/master/threaddelegate>

# thread delegate pattern



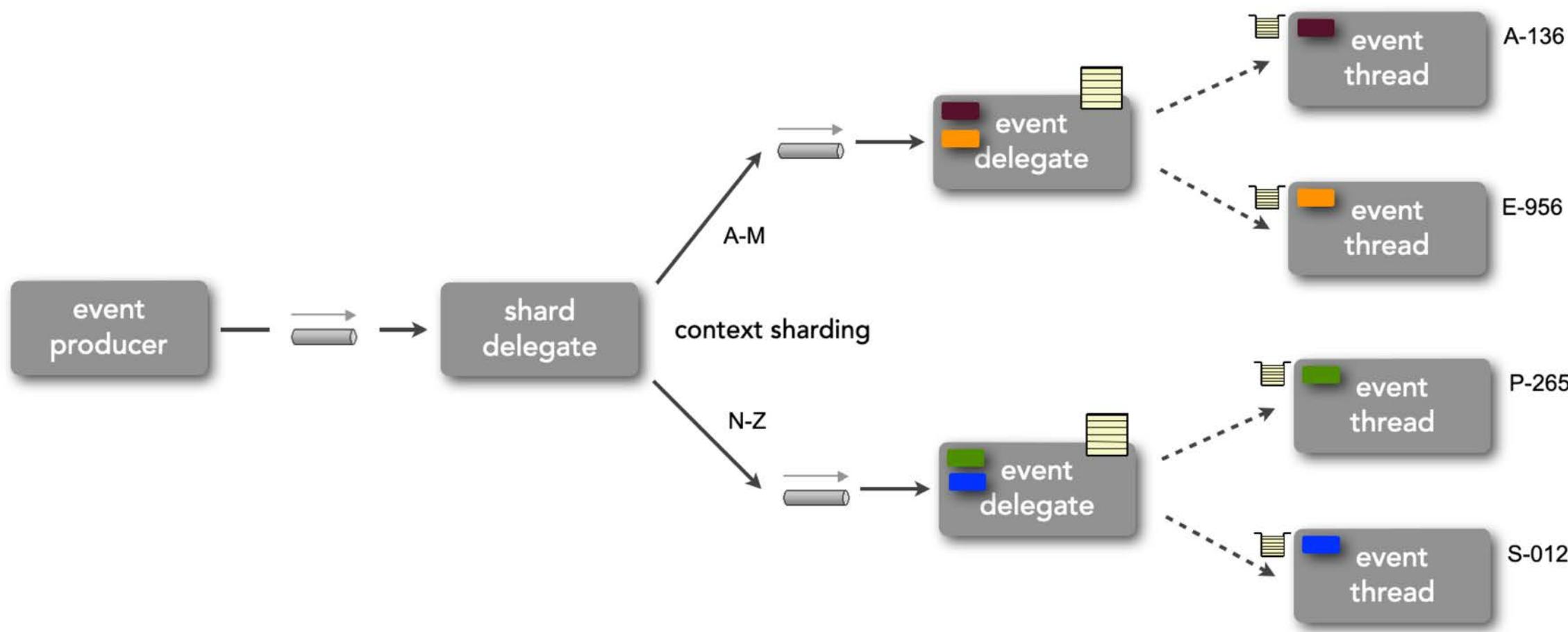
# thread delegate pattern



maintain processing order  
increase throughput  
increase performance  
increase scalability



# thread delegate pattern



- maintain processing order
- increase throughput
- increase performance
- increase scalability



- increased complexity
- possible thread saturation
- complex error handling
- increased cost

# Ambulance Pattern (Carpool)

# ambulance pattern

*“how can I give some events a higher priority and still maintain responsiveness for standard events?”*



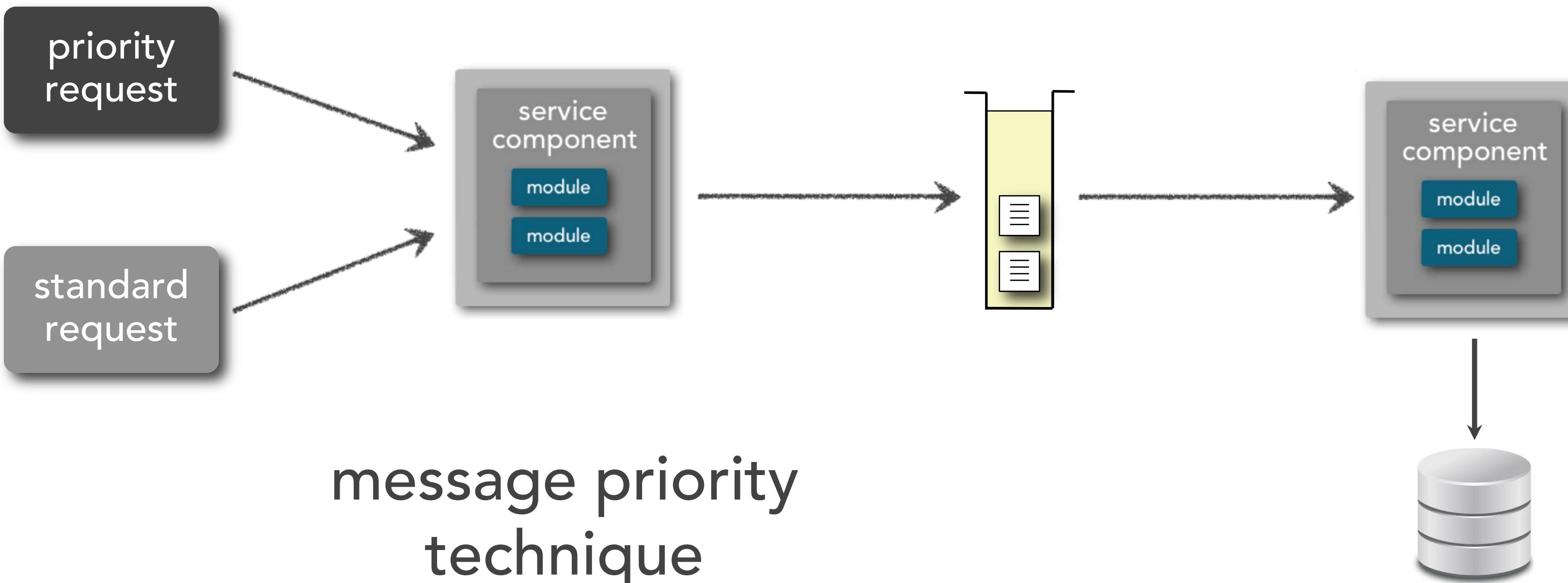
# ambulance pattern



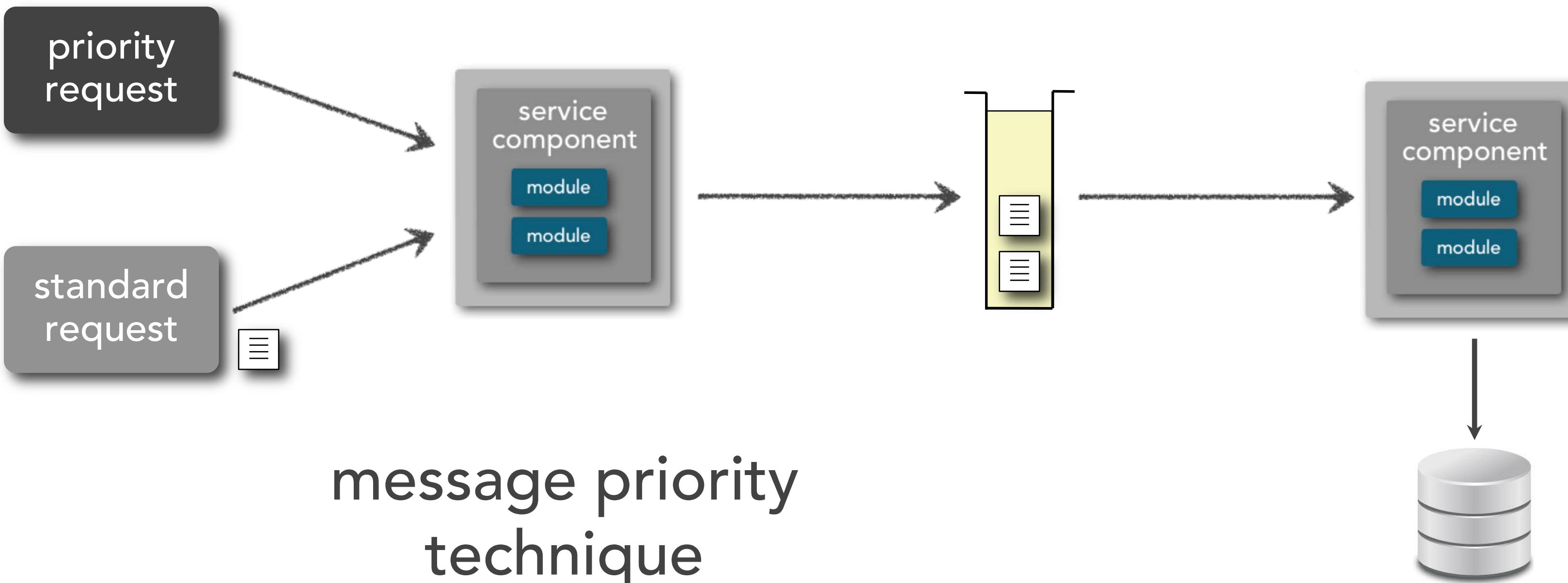
let's see the issue...

<https://github.com/wmr513/event-driven-patterns/tree/master/ambulance>

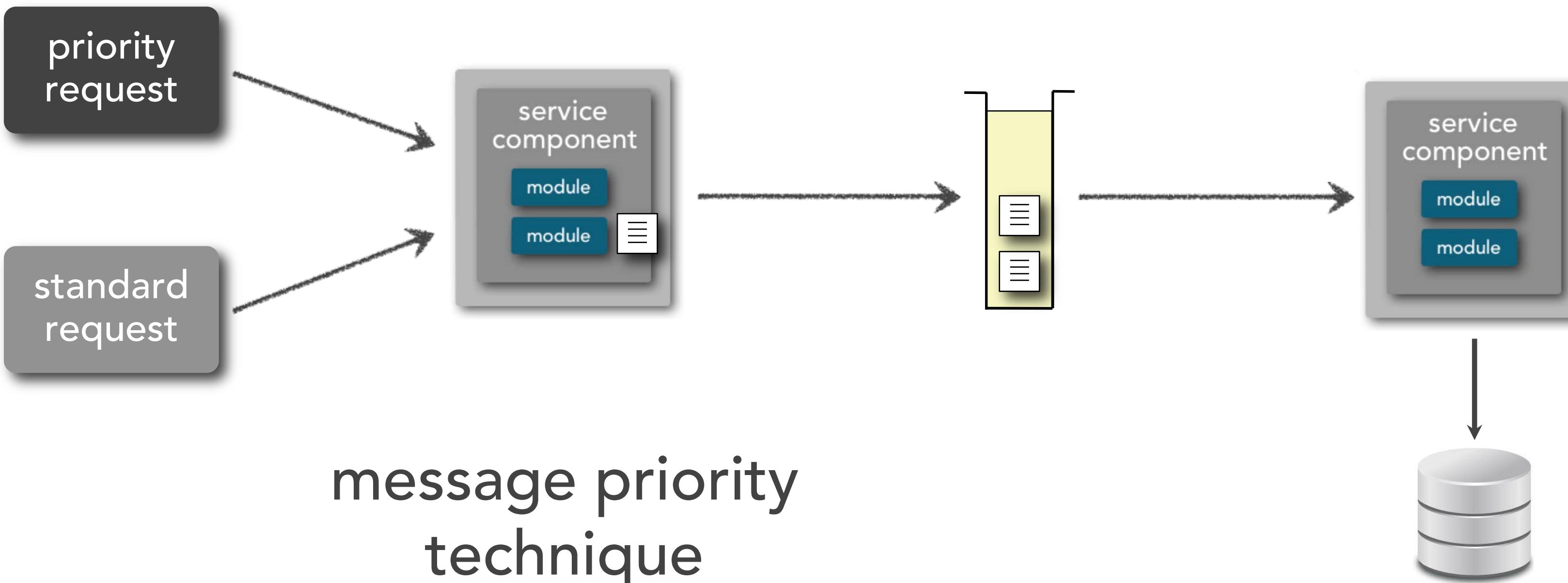
# ambulance pattern



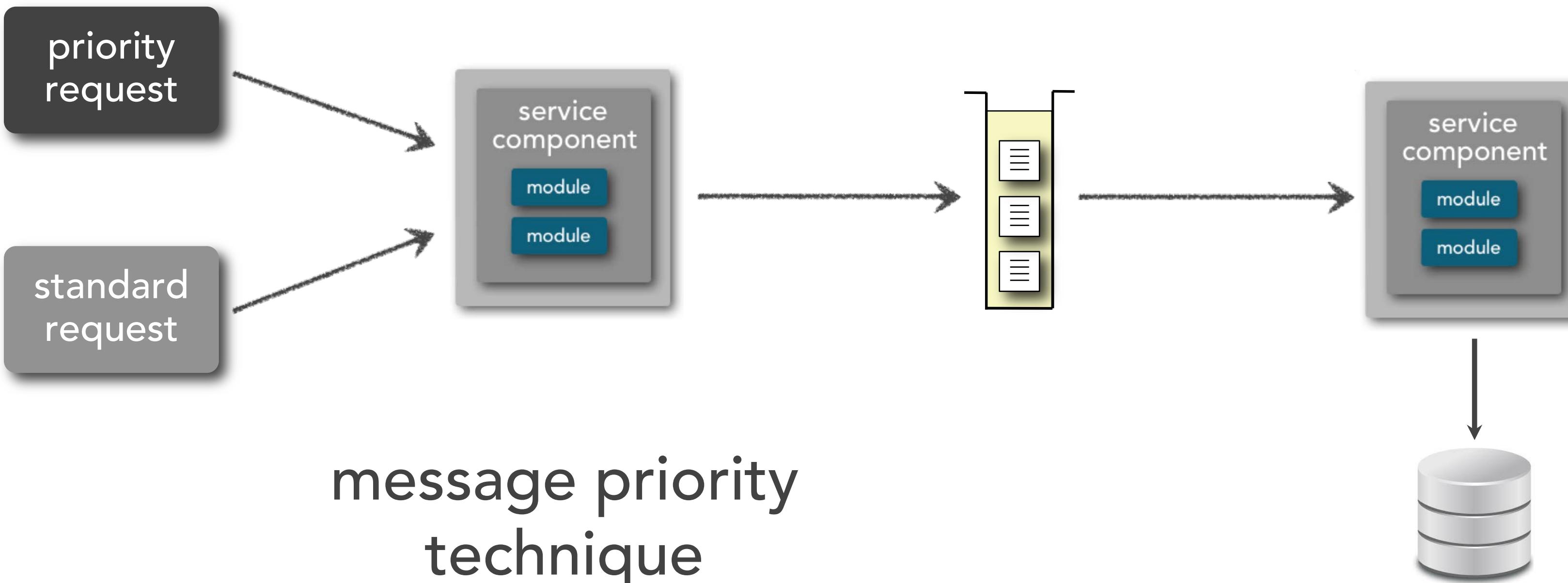
# ambulance pattern



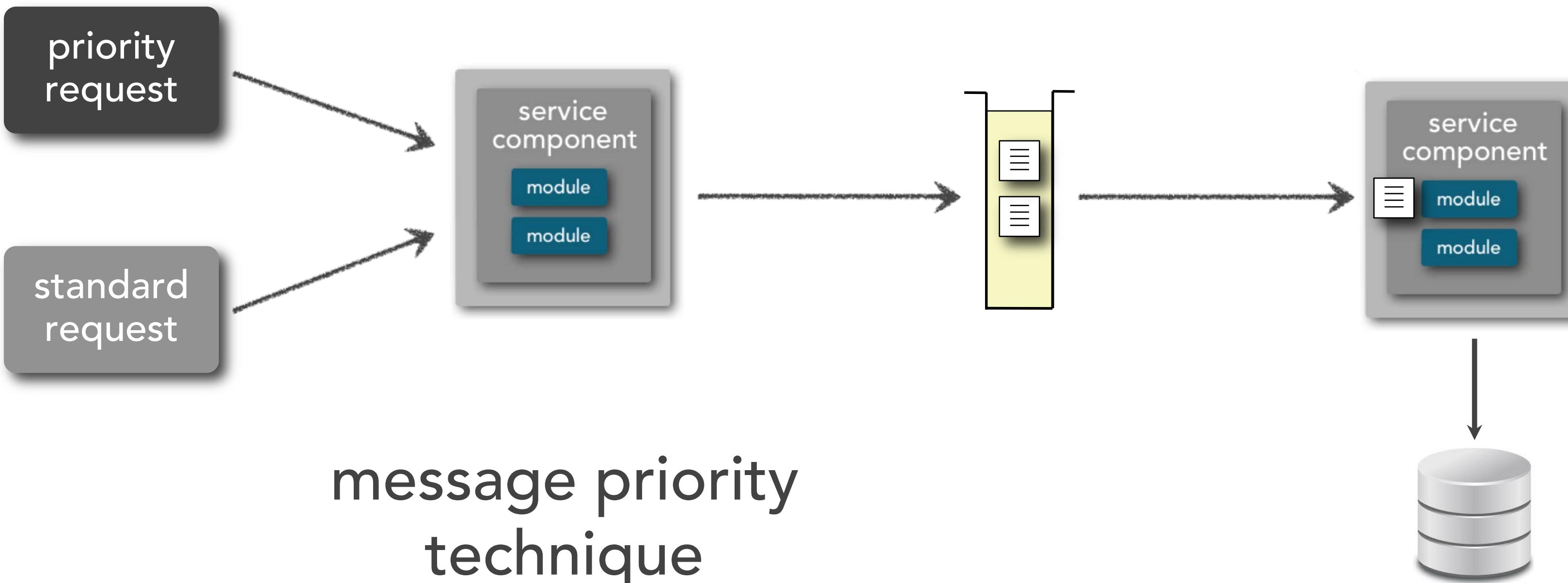
# ambulance pattern



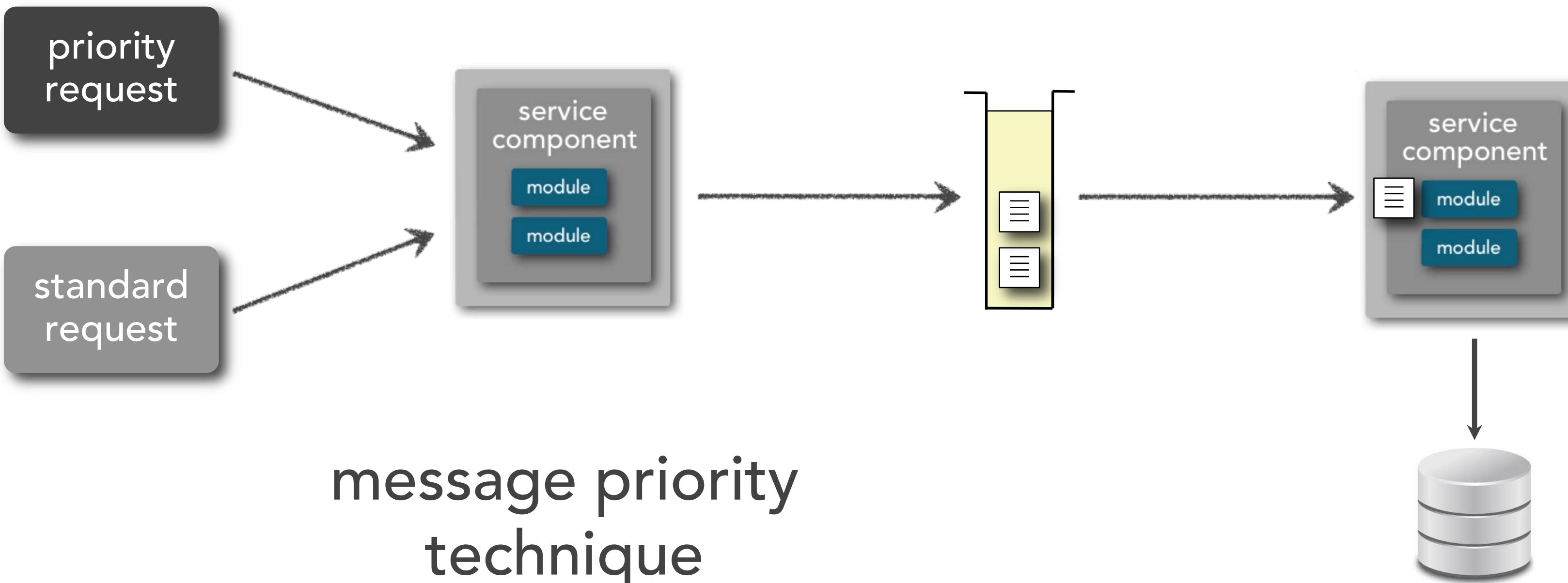
# ambulance pattern



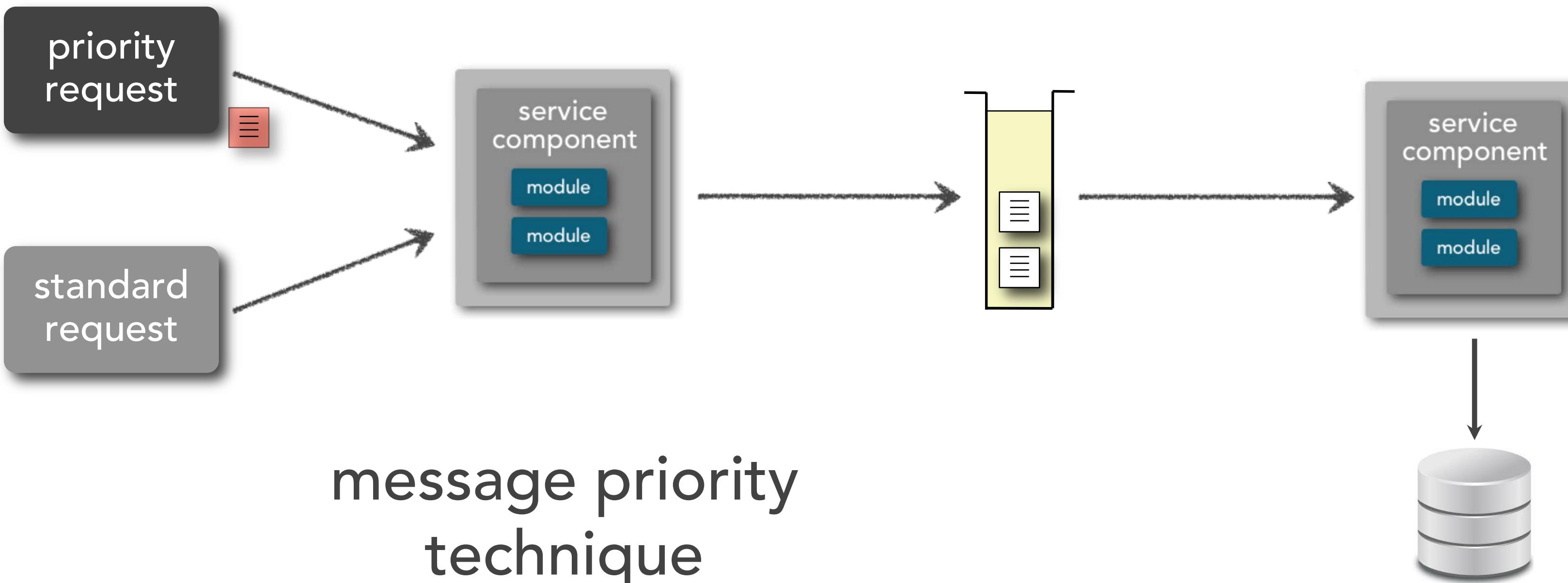
# ambulance pattern



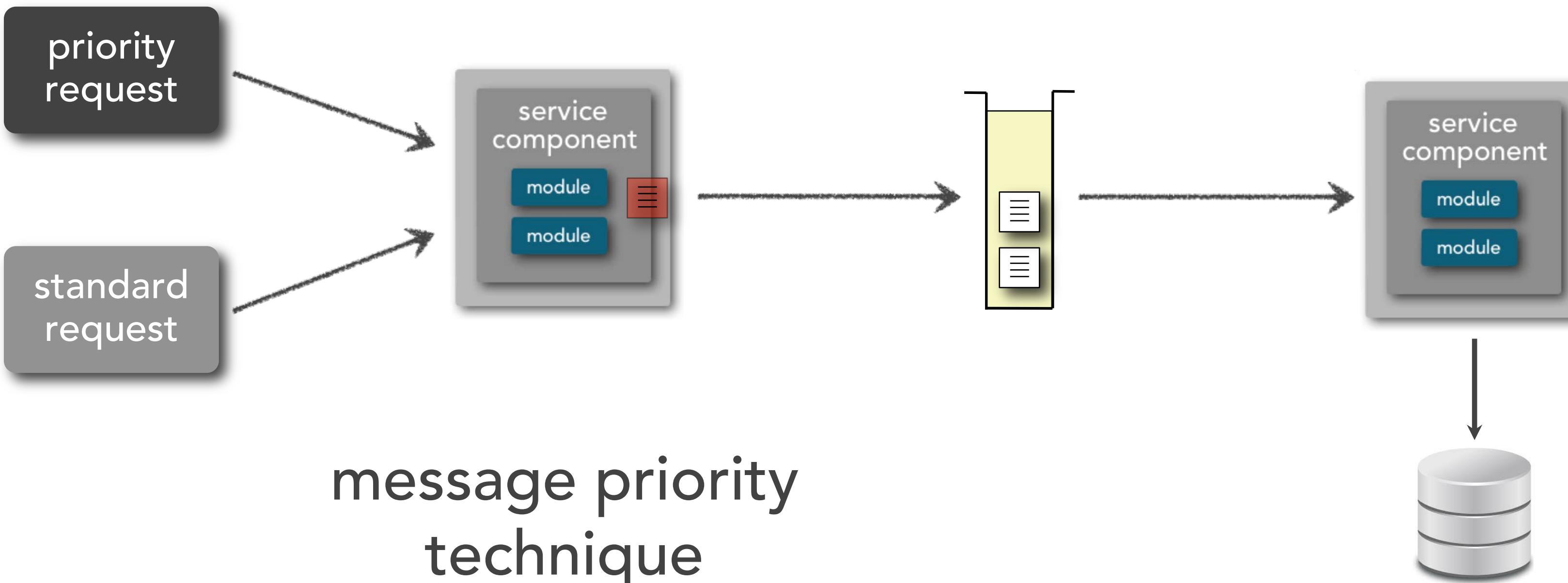
# ambulance pattern



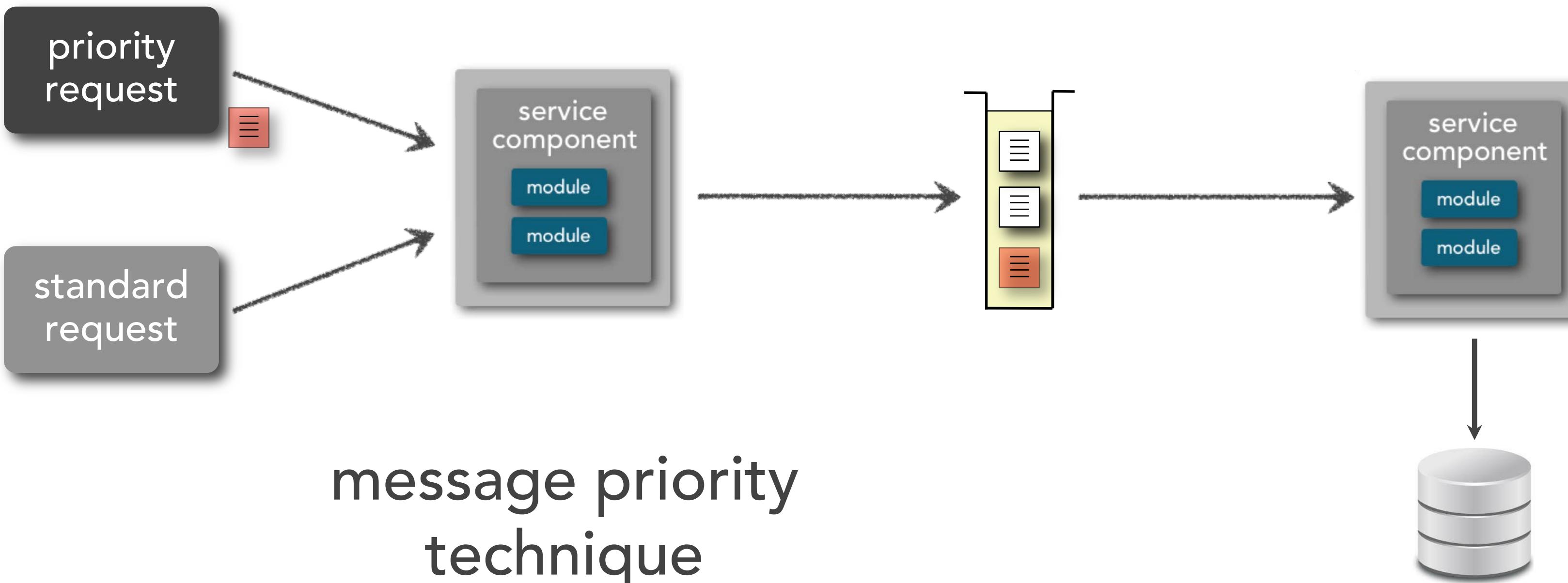
# ambulance pattern



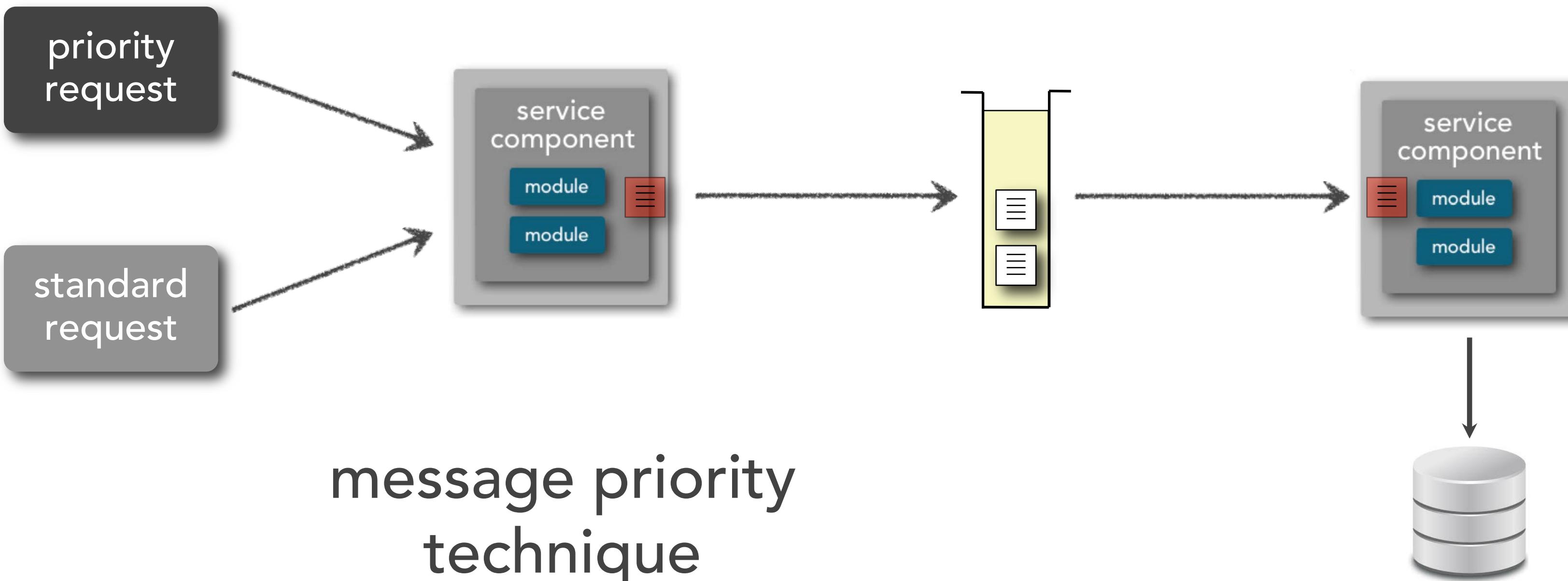
# ambulance pattern



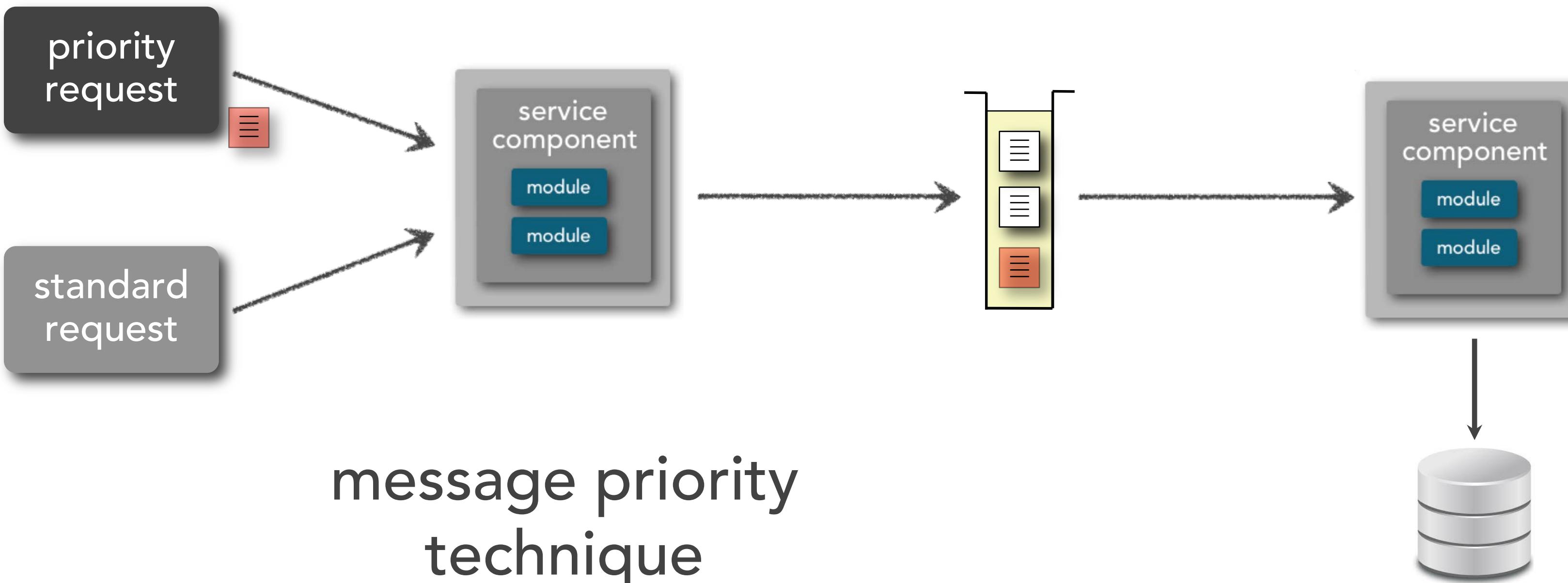
# ambulance pattern



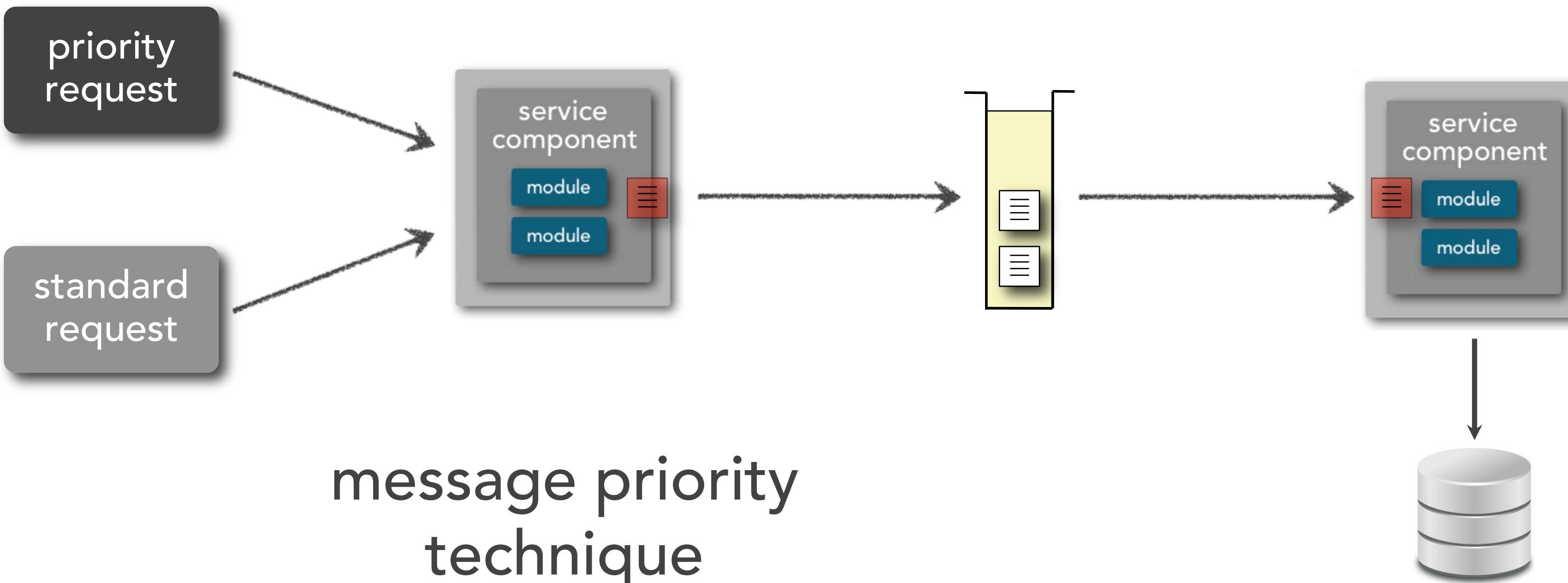
# ambulance pattern



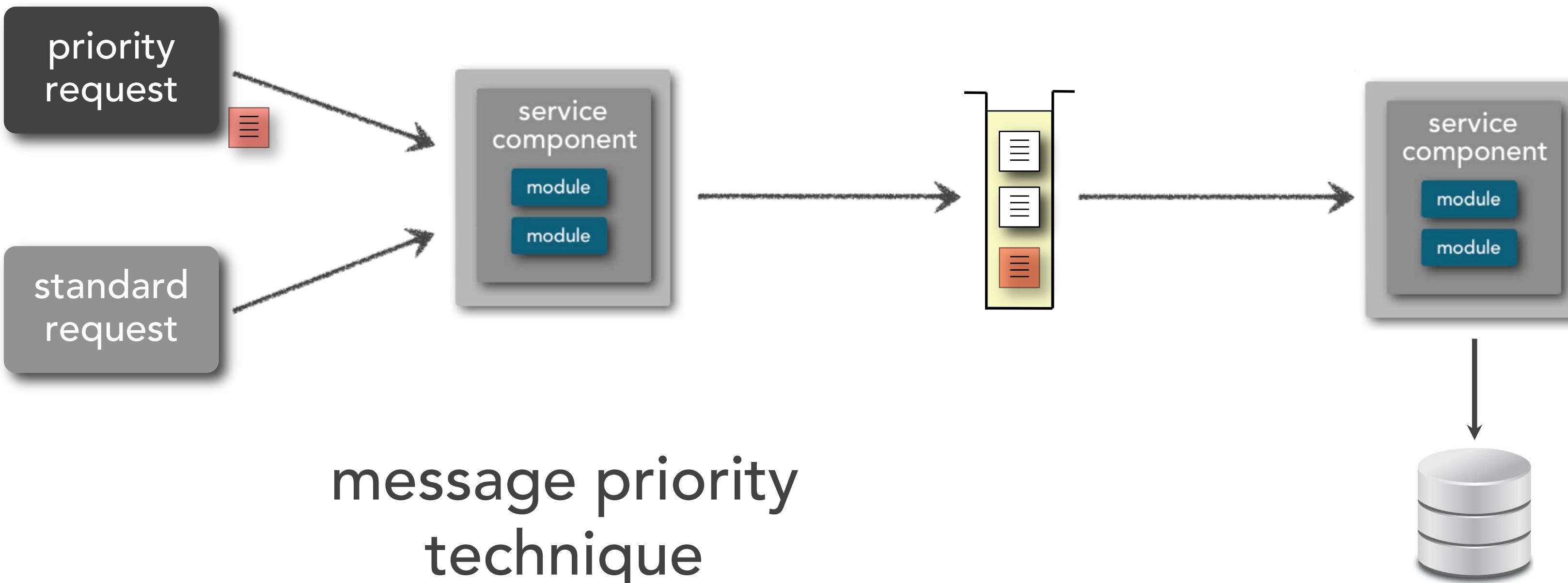
# ambulance pattern



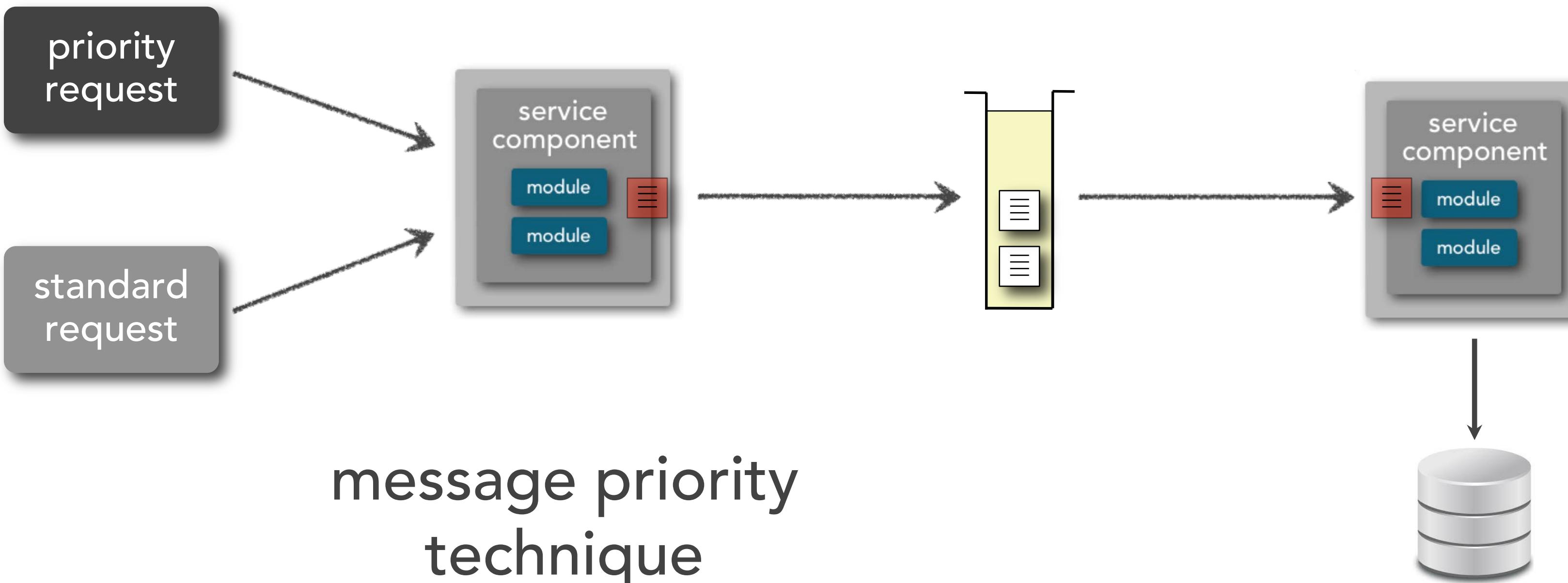
# ambulance pattern



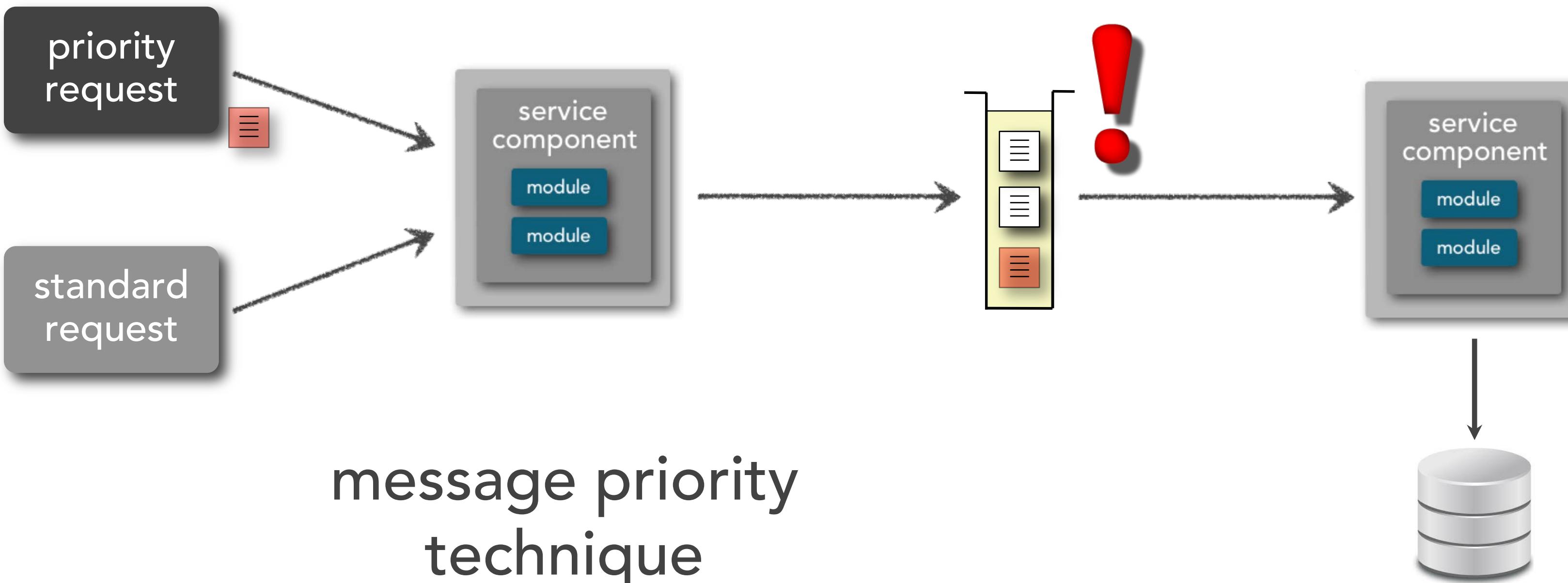
# ambulance pattern



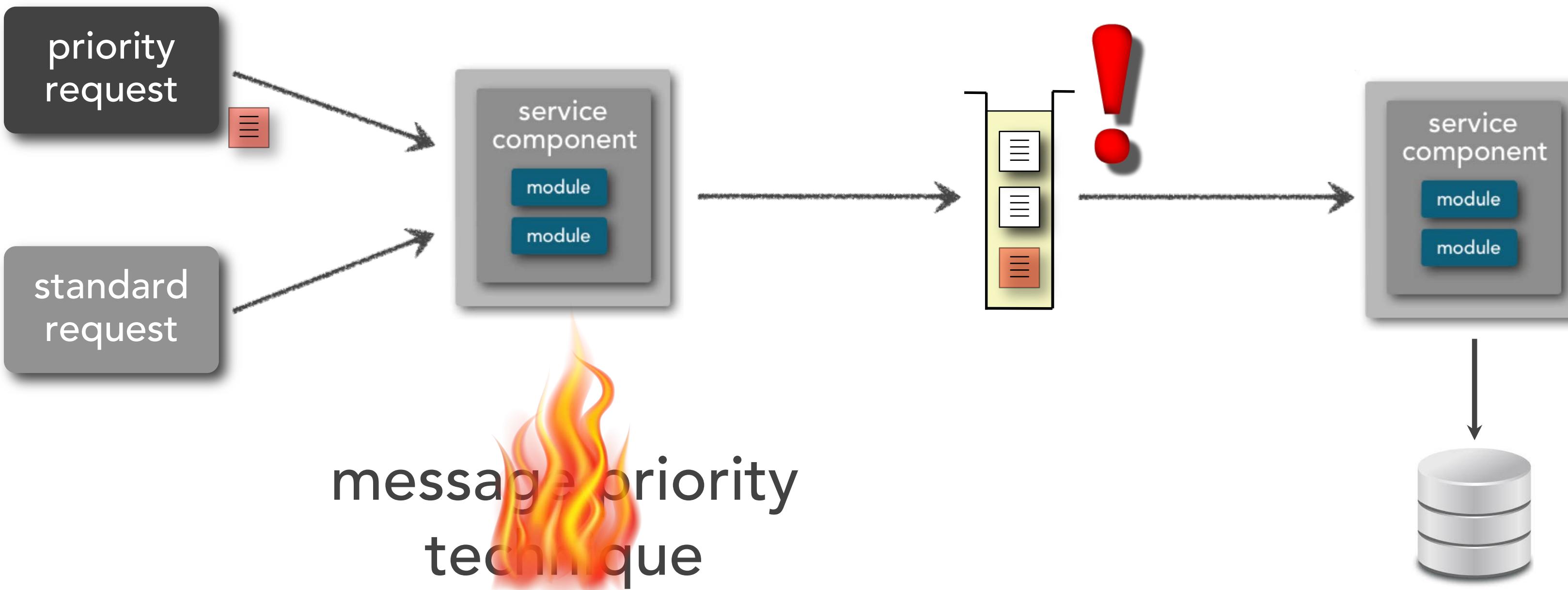
# ambulance pattern



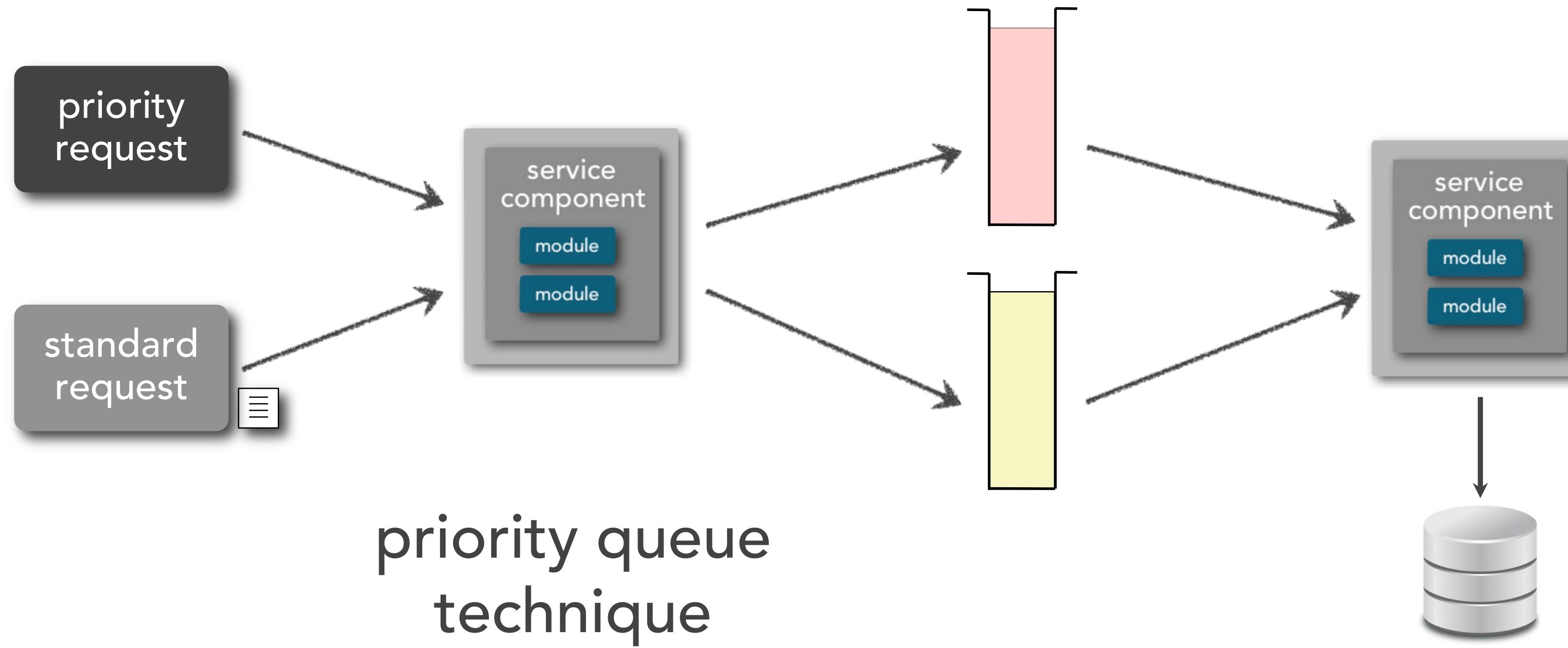
# ambulance pattern



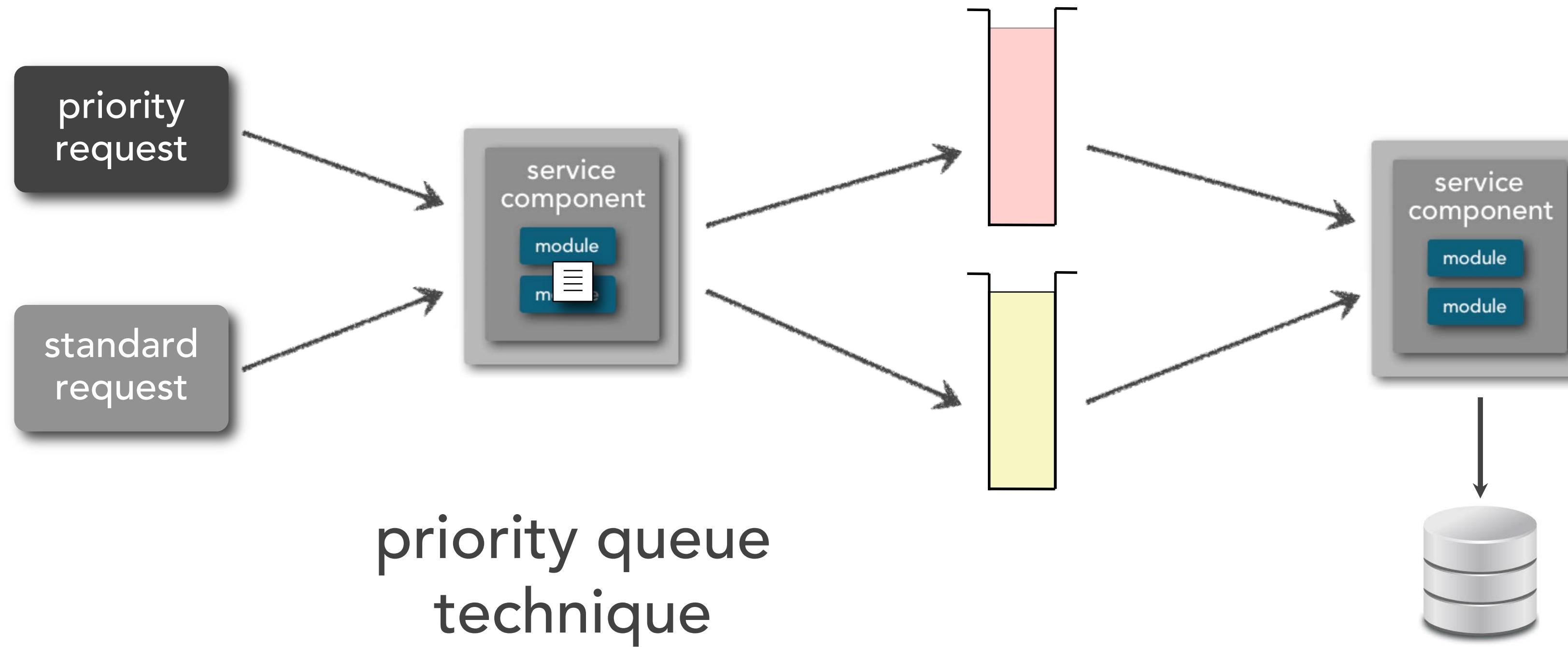
# ambulance pattern



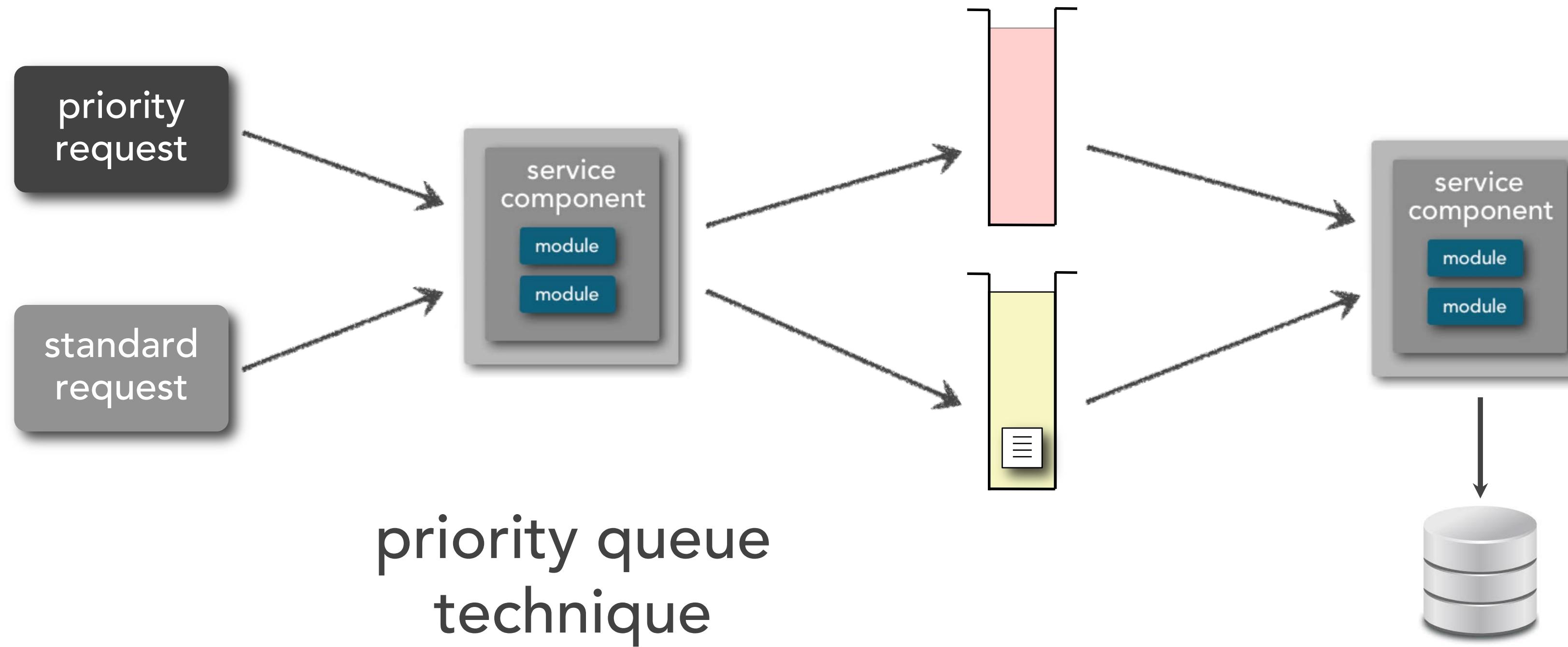
# ambulance pattern



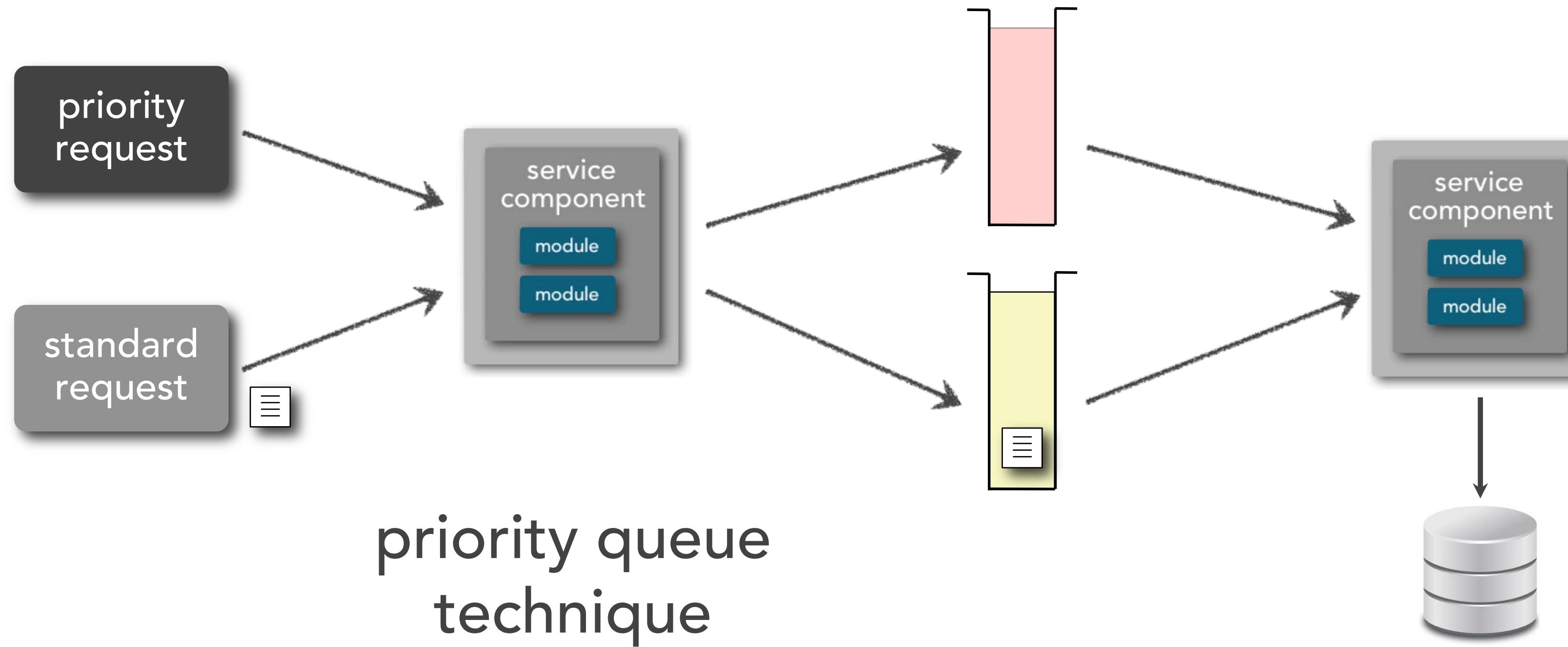
# ambulance pattern



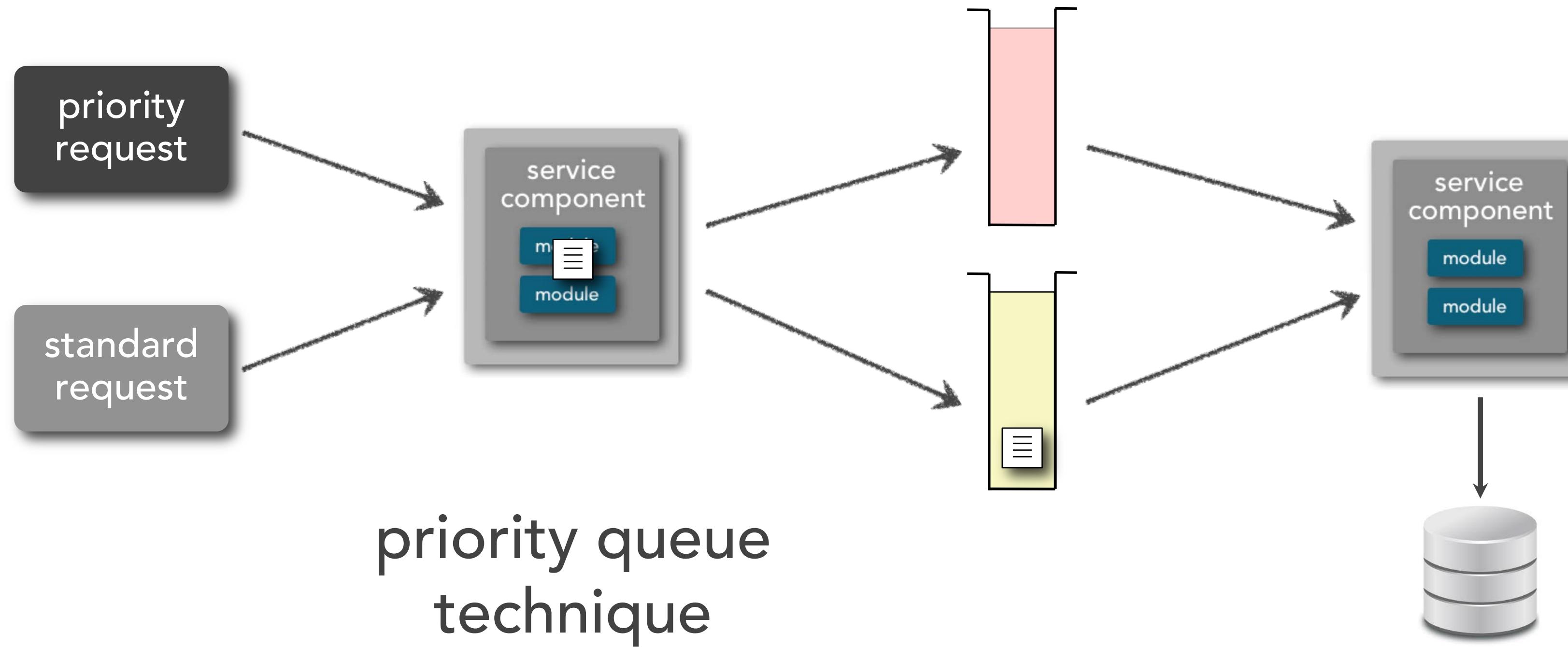
# ambulance pattern



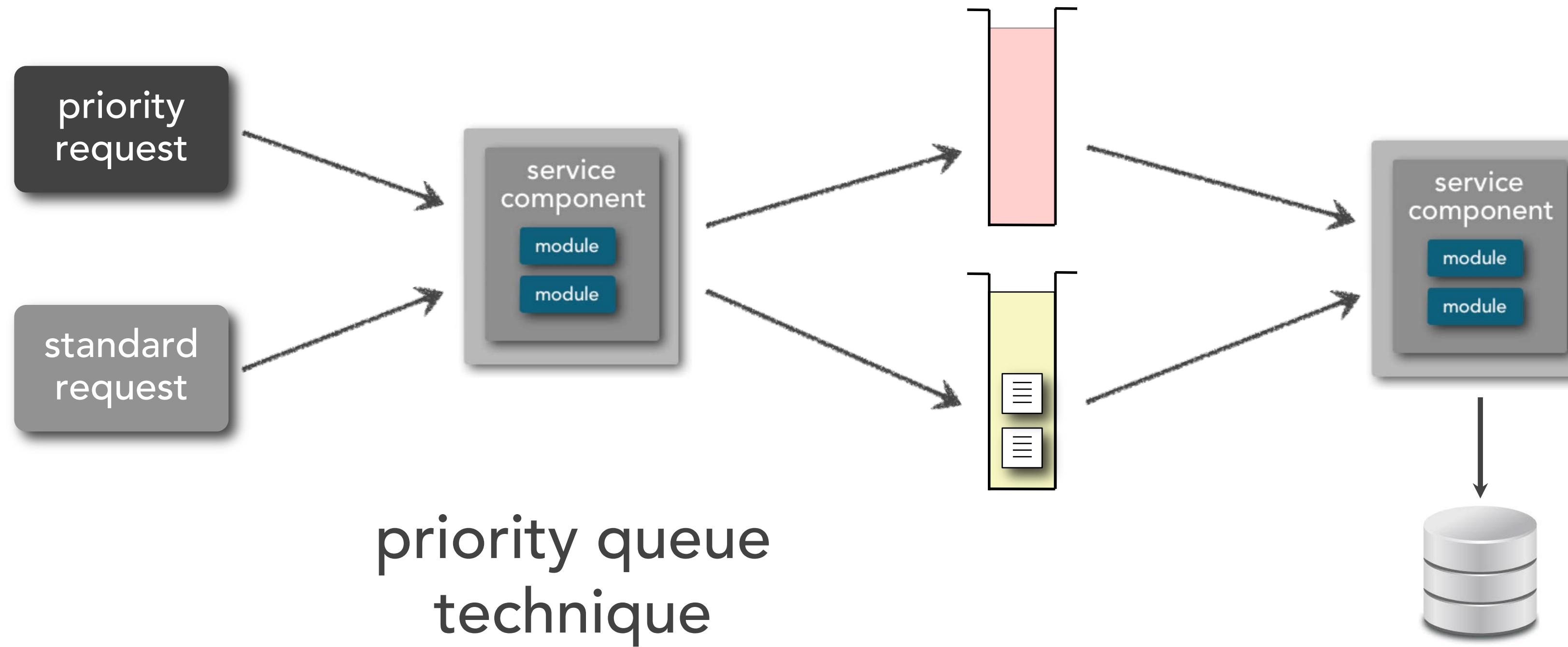
# ambulance pattern



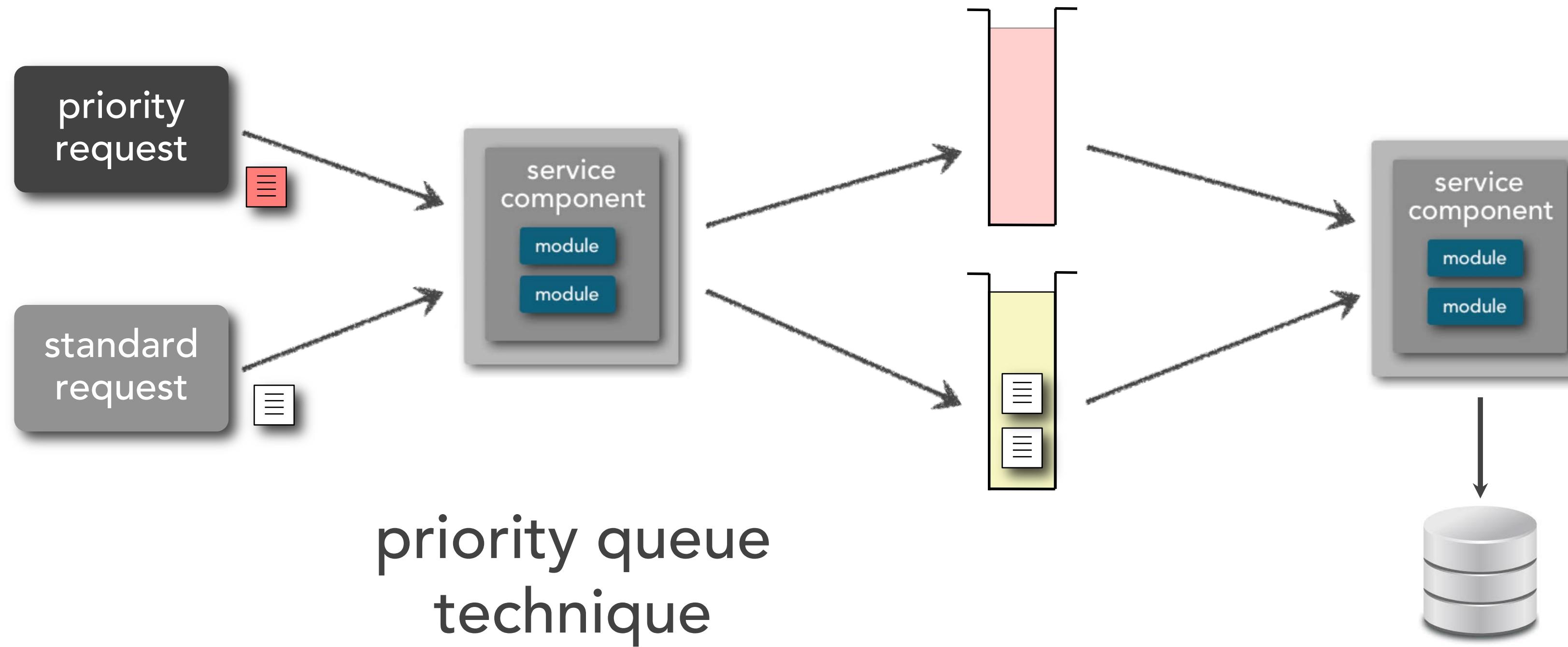
# ambulance pattern



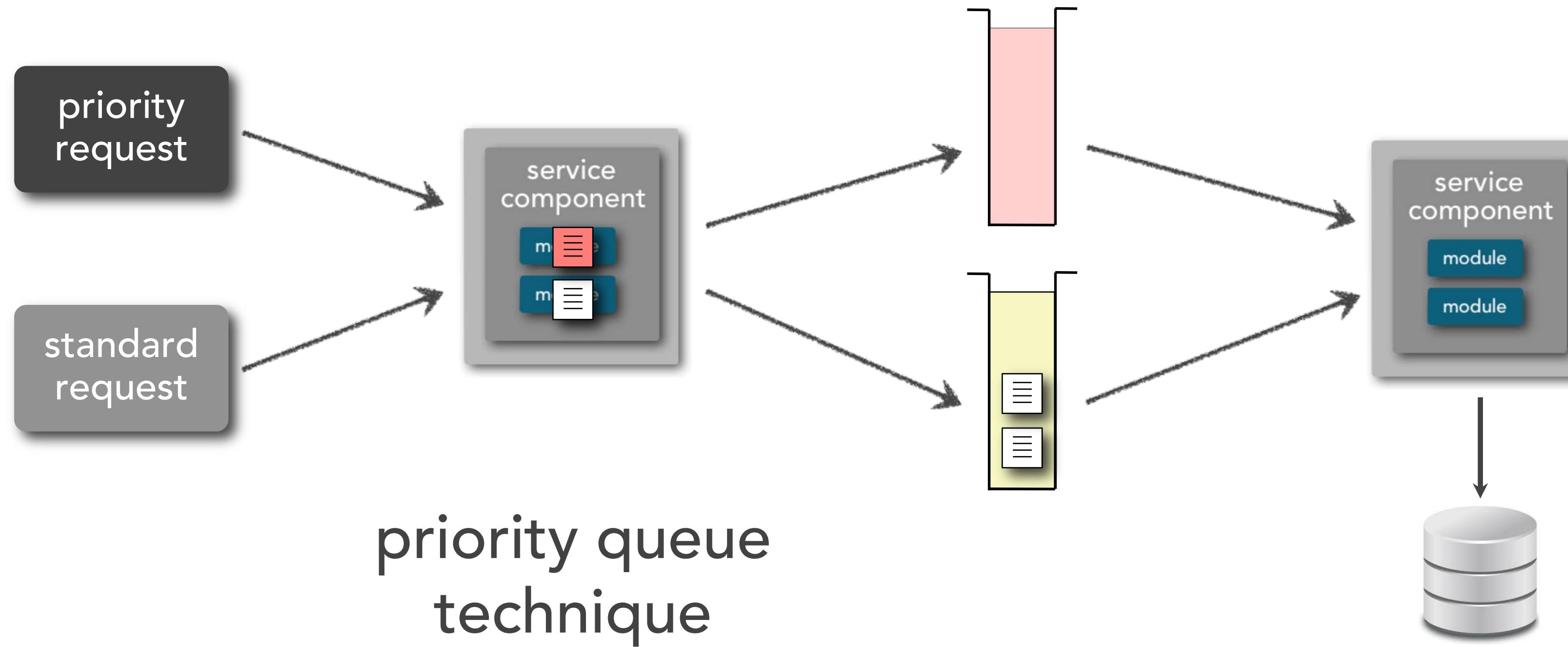
# ambulance pattern



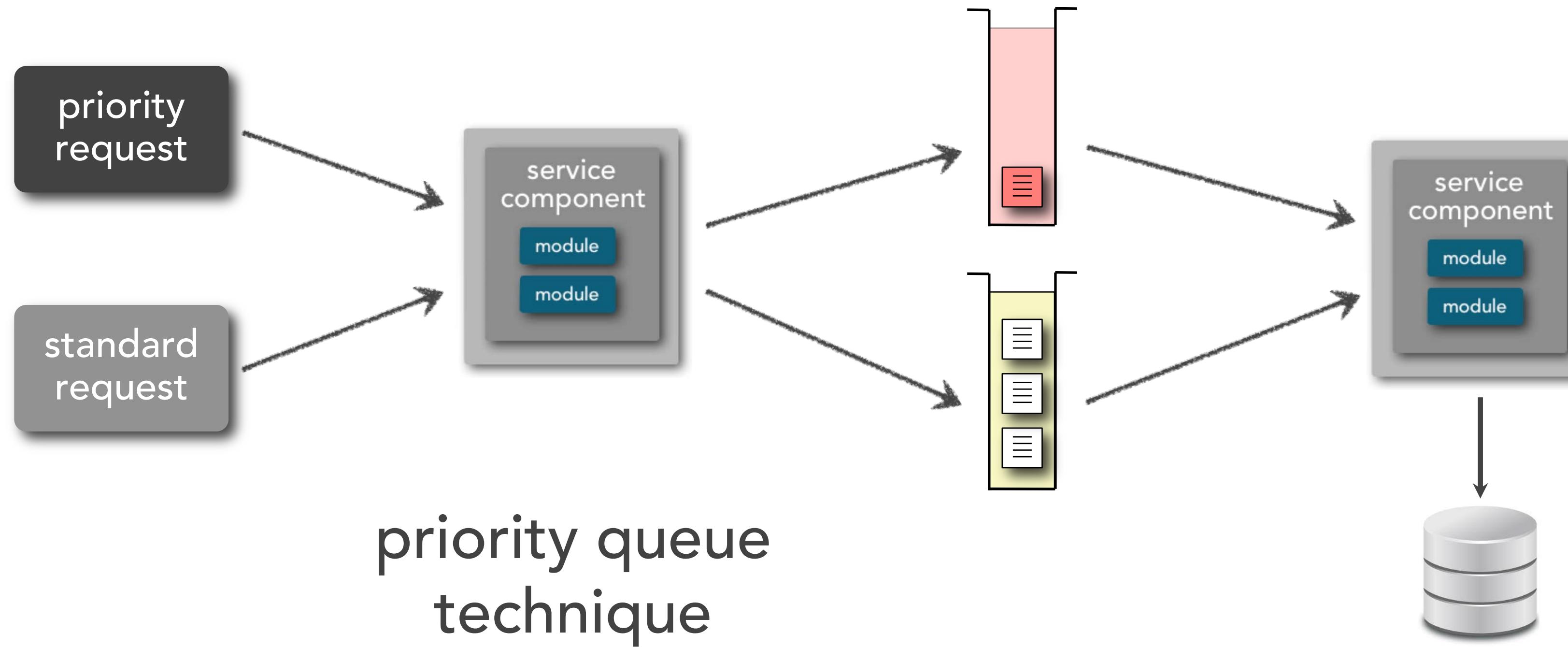
# ambulance pattern



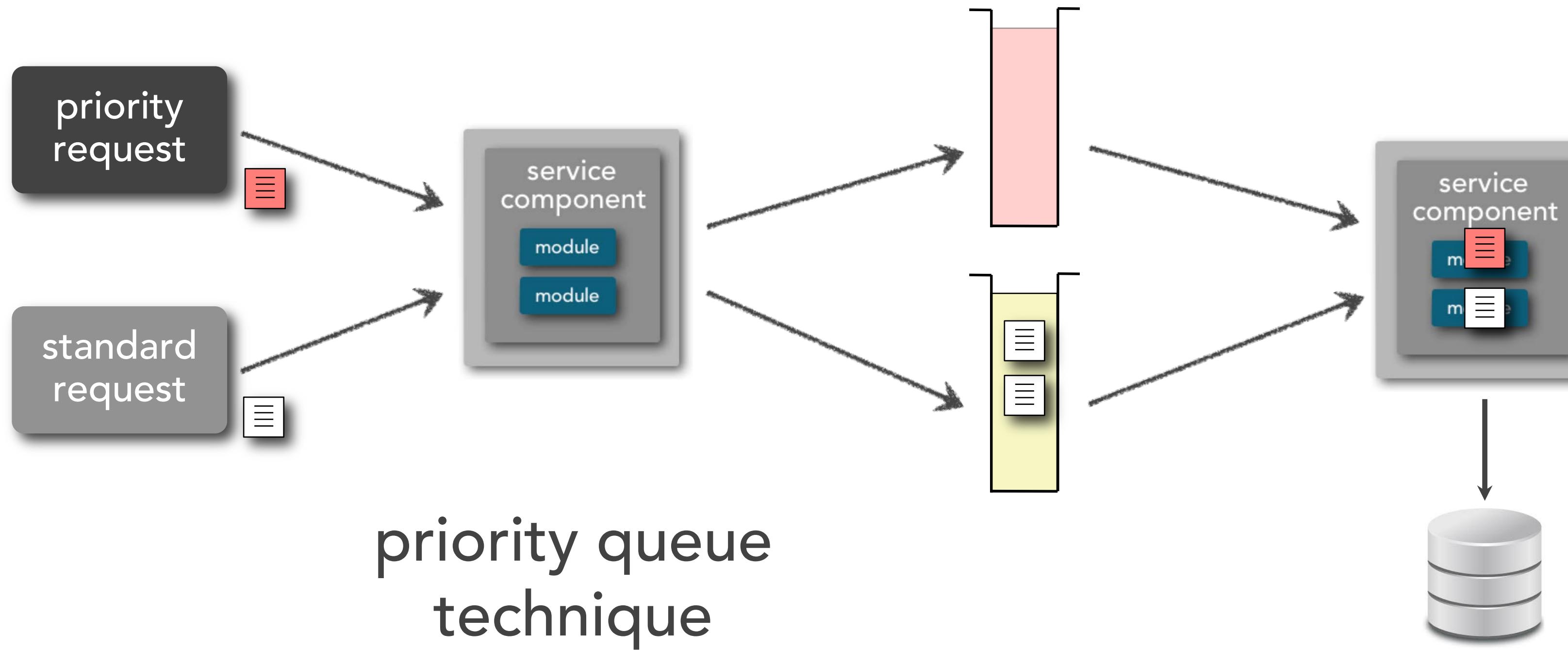
# ambulance pattern



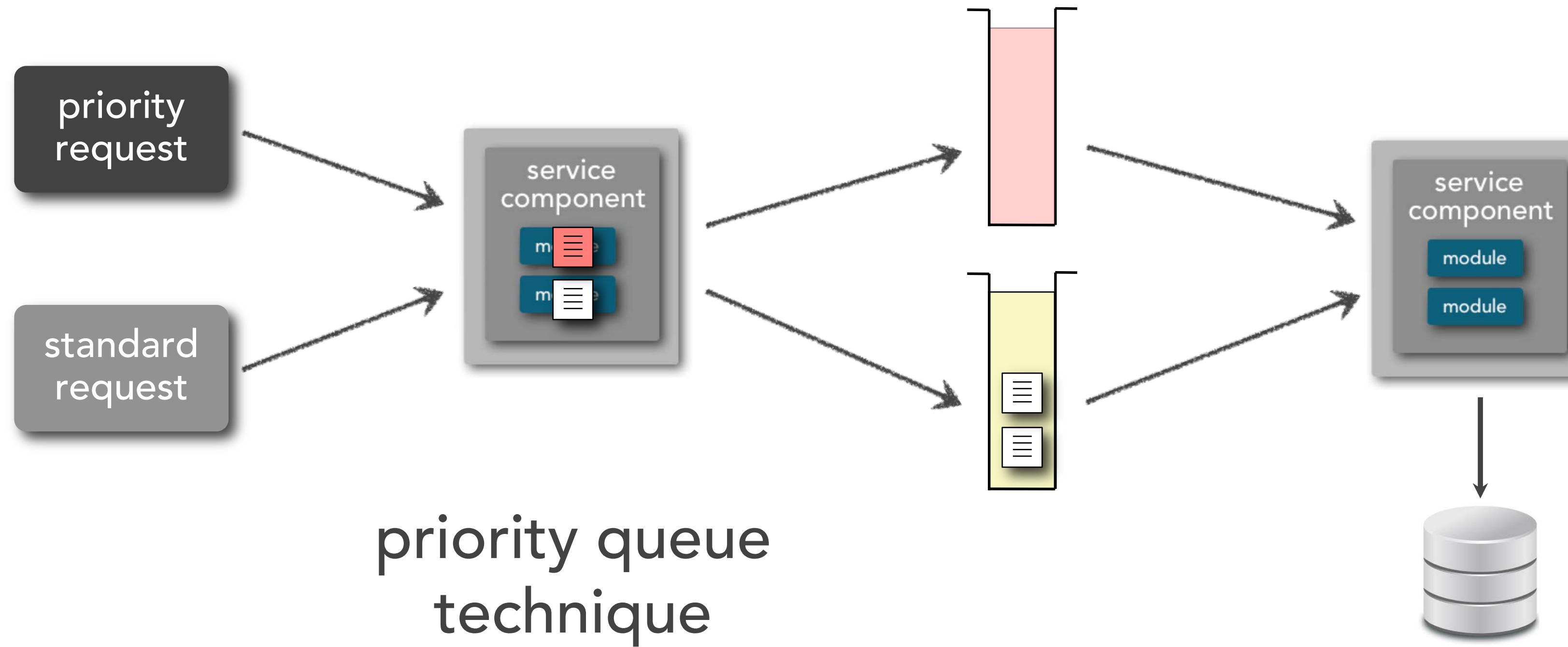
# ambulance pattern



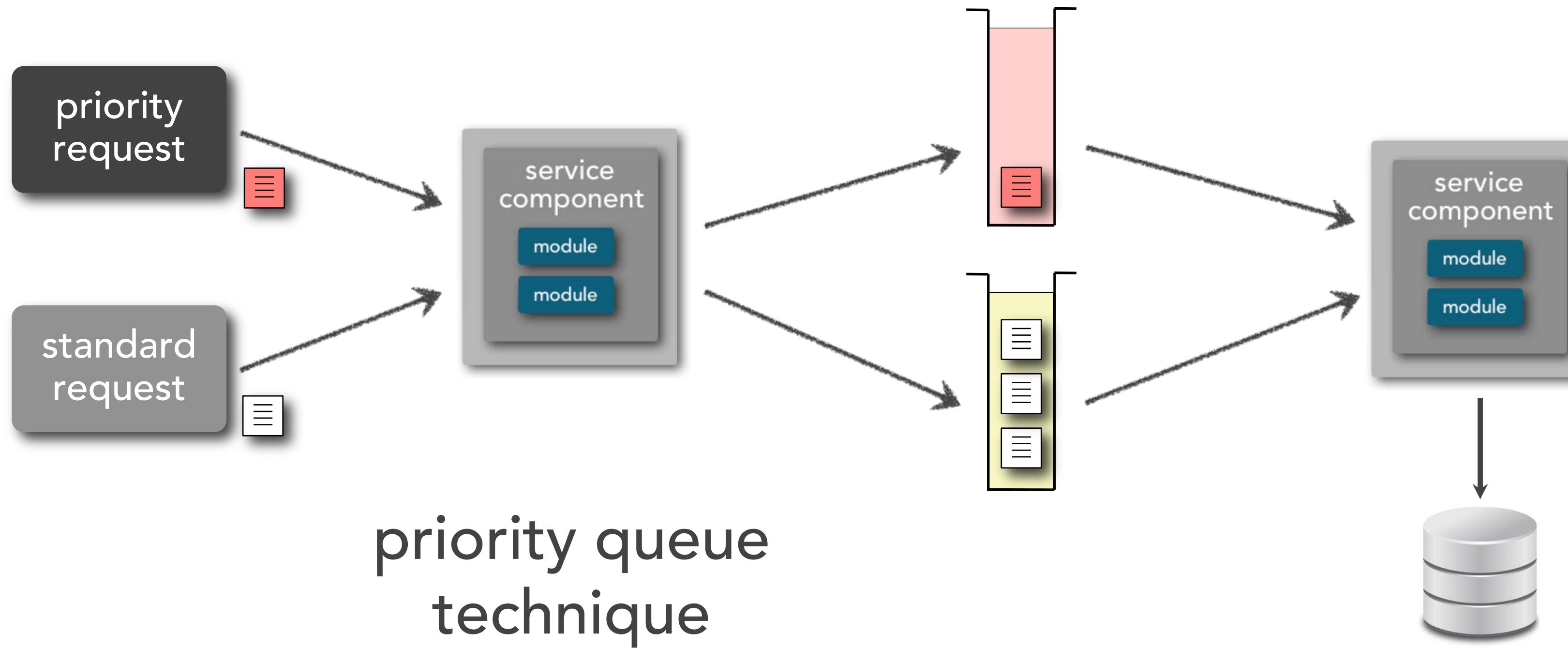
# ambulance pattern



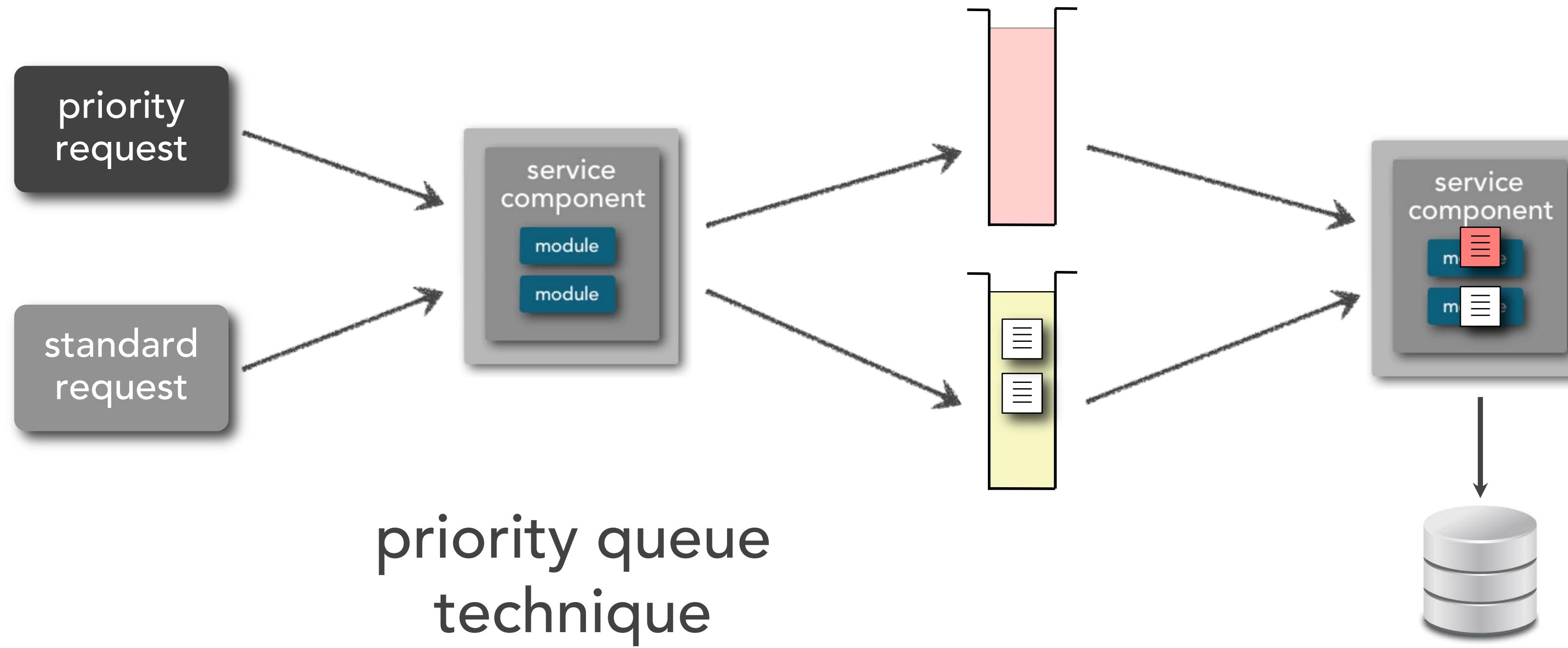
# ambulance pattern



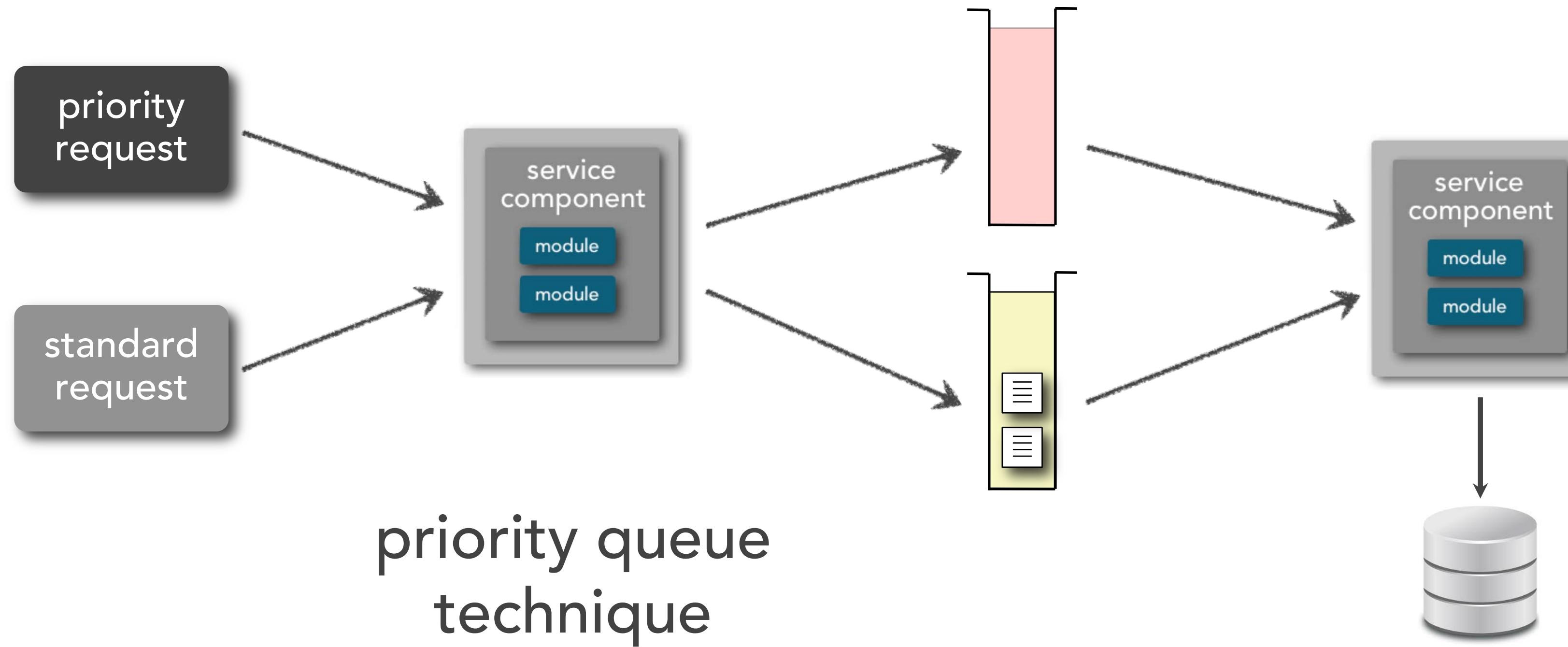
# ambulance pattern



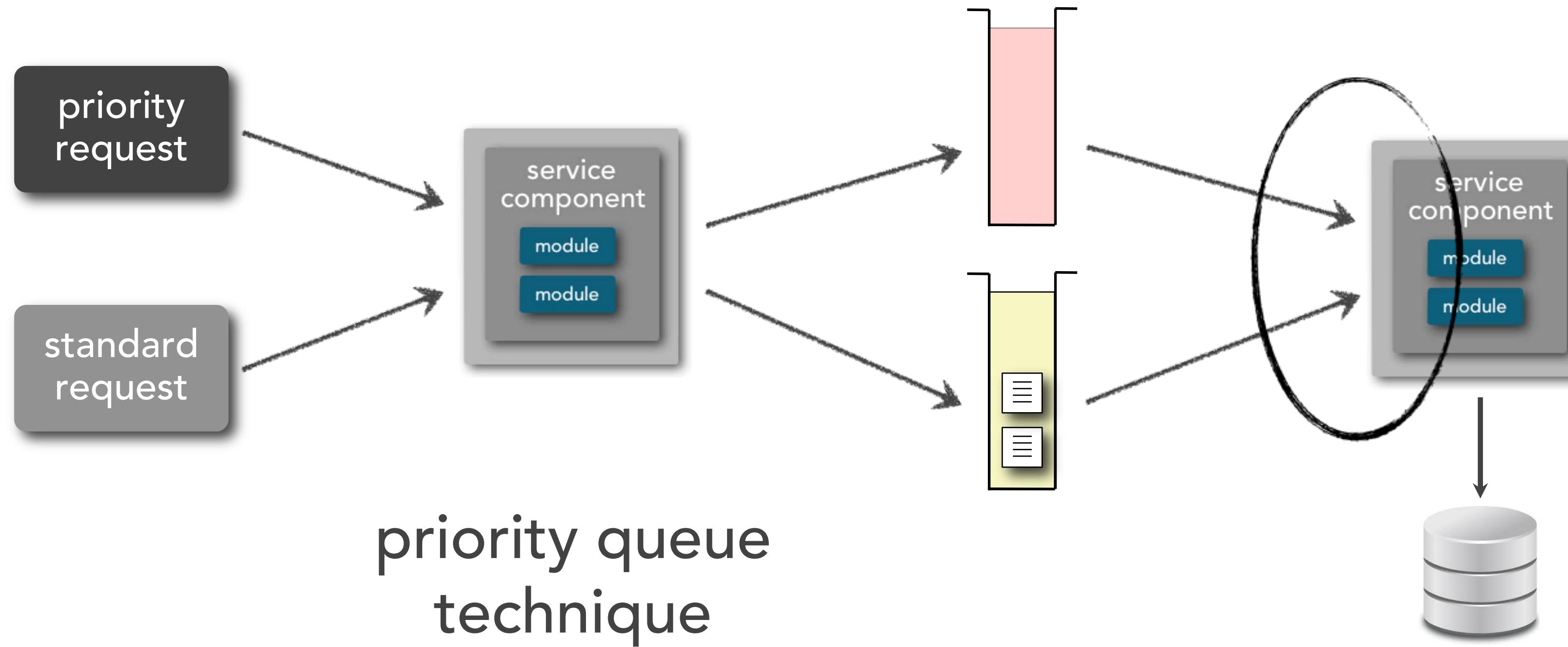
# ambulance pattern



# ambulance pattern

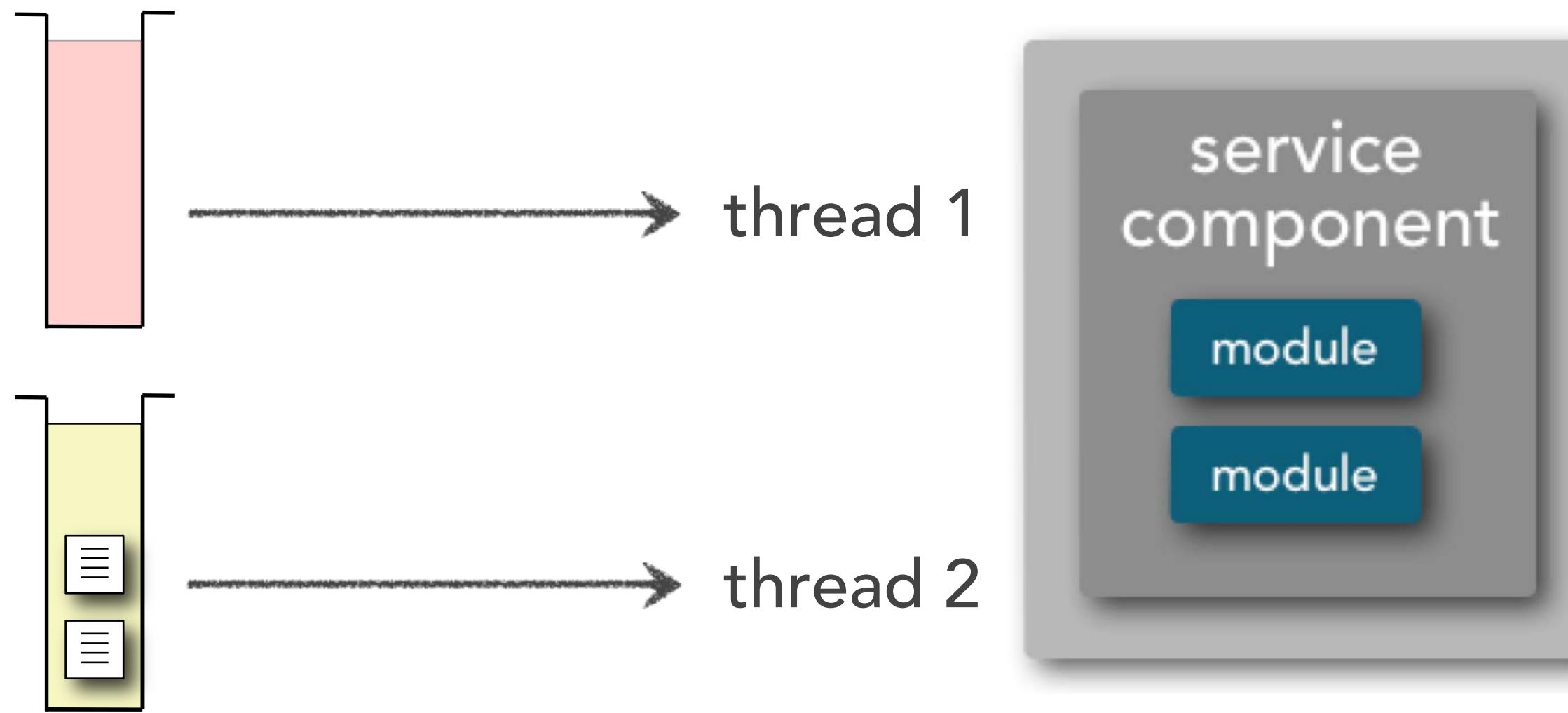


# ambulance pattern



# ambulance pattern

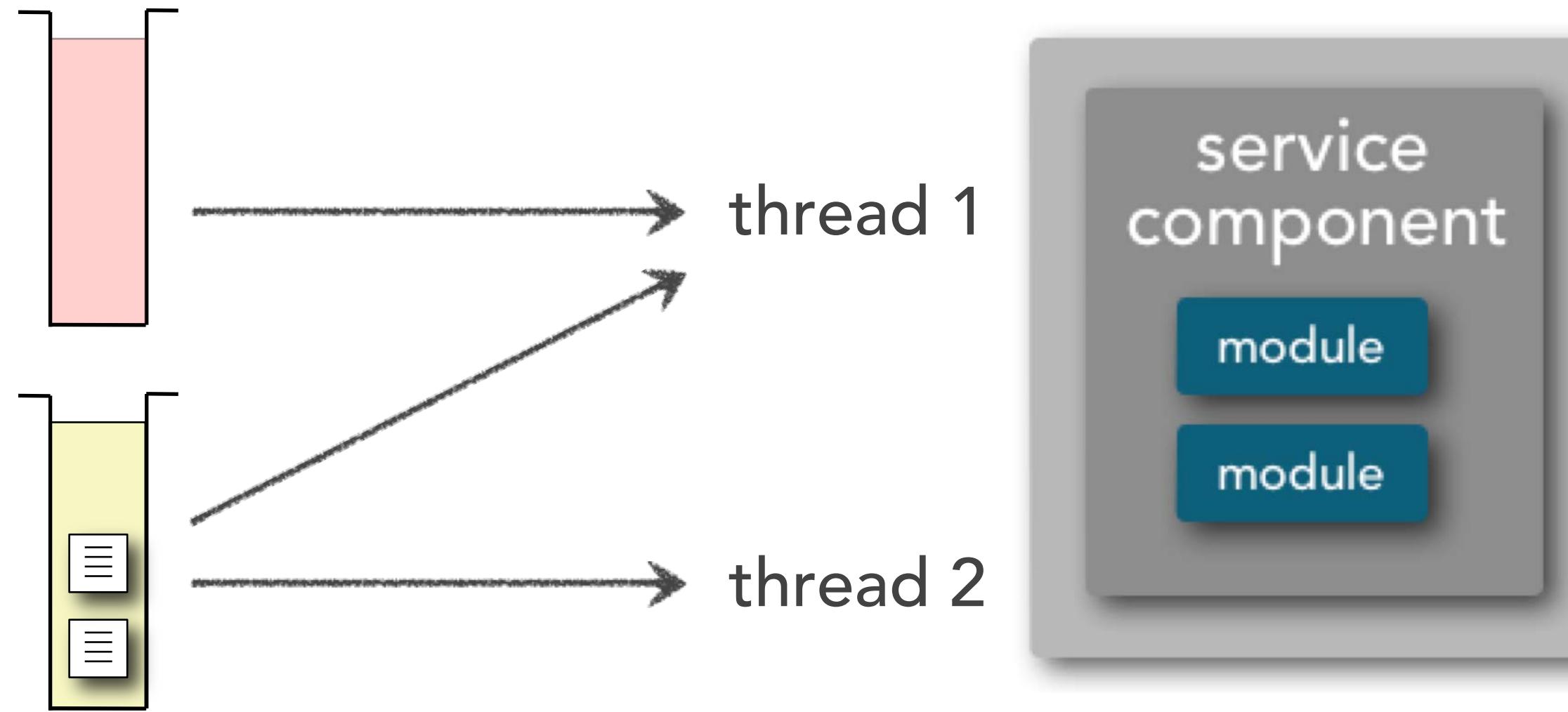
## carpool vs. ambulance



**carpool** - thread 1 is dedicated to priority messages only and remains idle if no priority messages exist

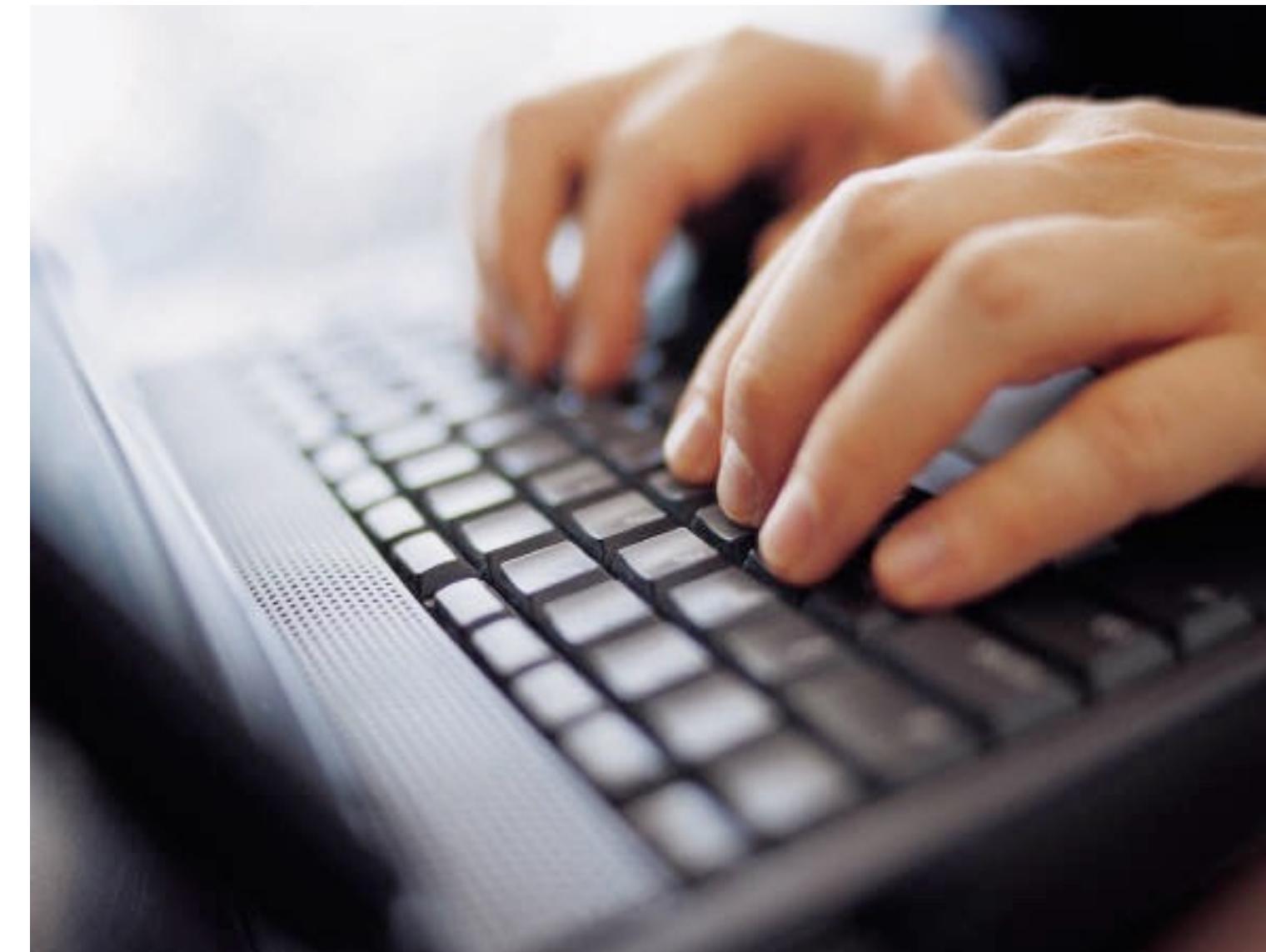
# ambulance pattern

## carpool vs. ambulance



**ambulance** - thread 1 can handle standard requests as well, but always checks priority queue first before getting the next standard message

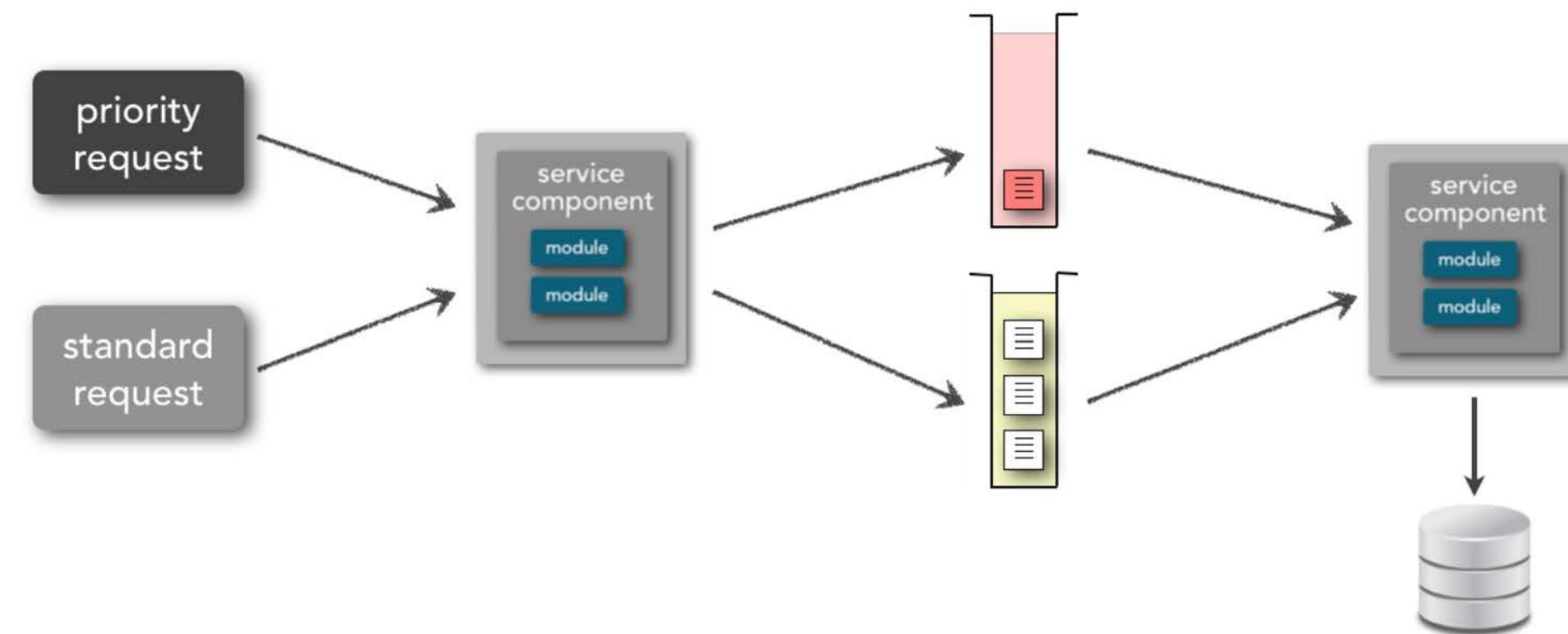
# ambulance pattern



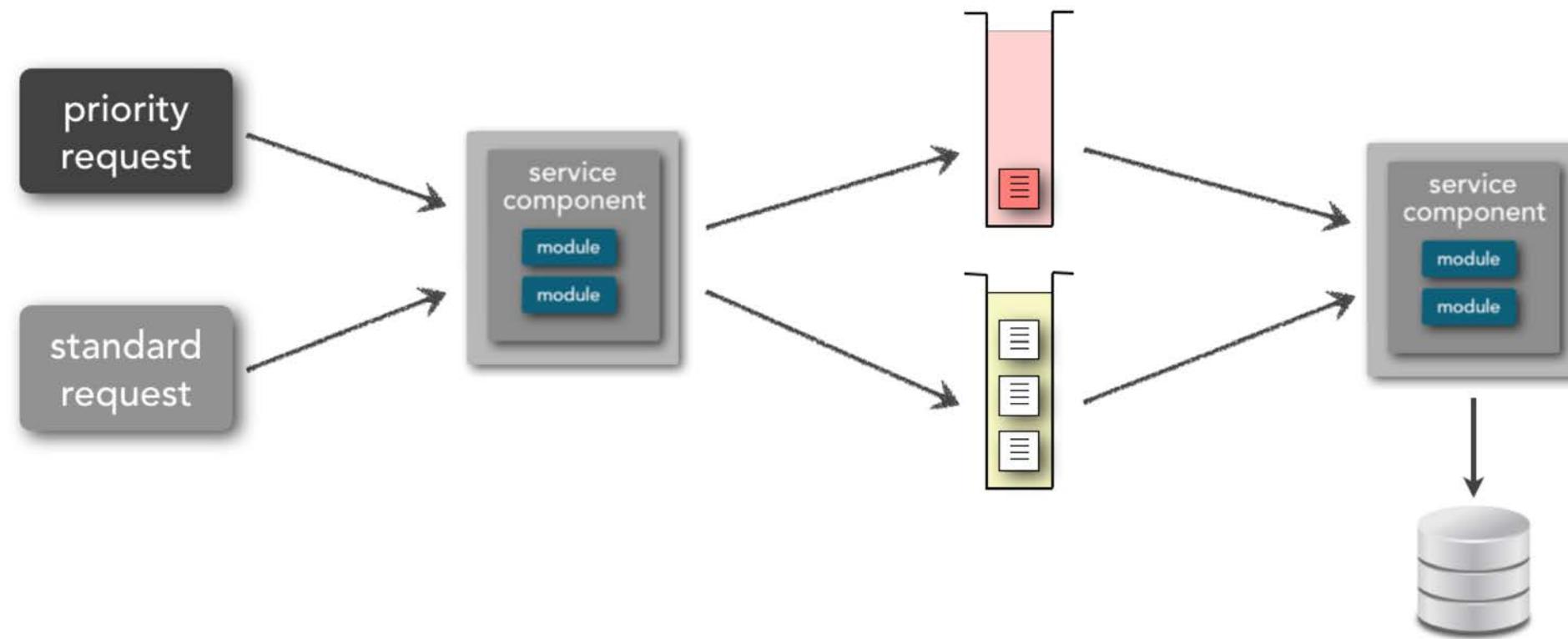
let's apply the pattern...

<https://github.com/wmr513/event-driven-patterns/tree/master/ambulance>

# ambulance pattern



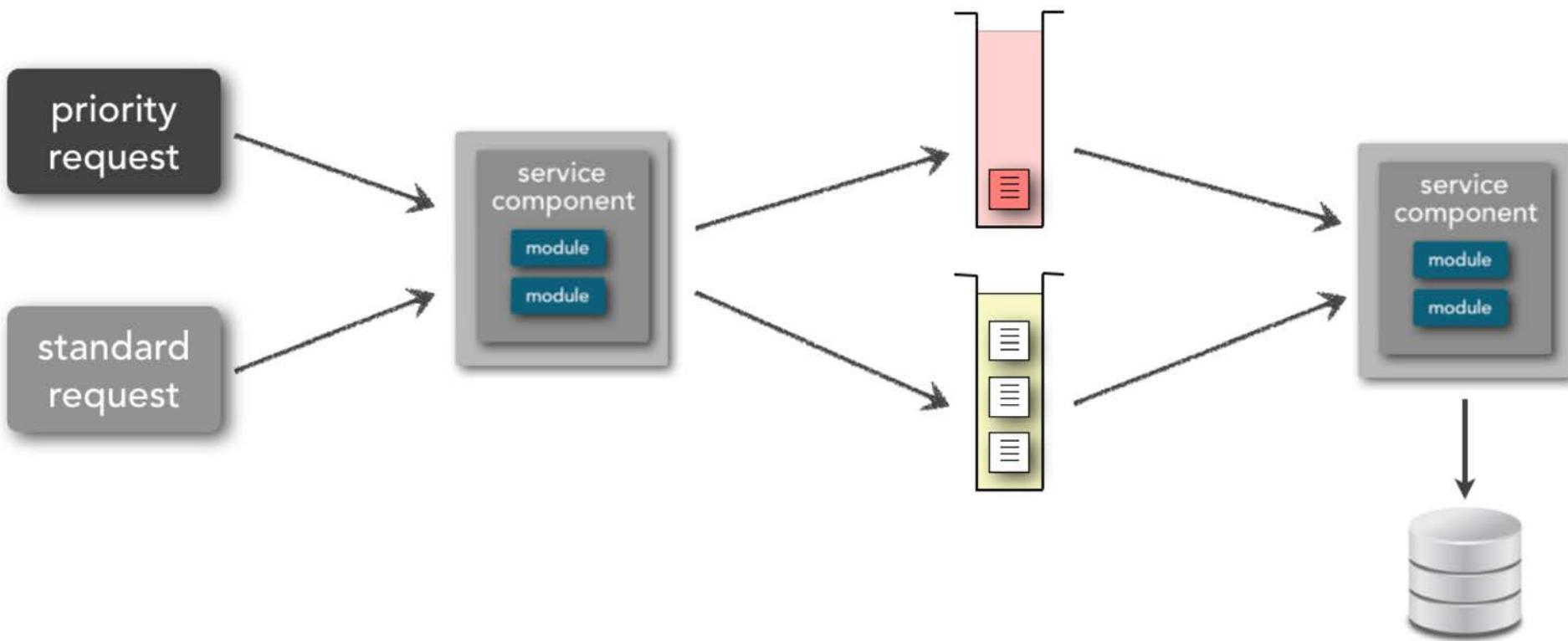
# ambulance pattern



message priority  
throughput  
performance



# ambulance pattern



message priority  
throughput  
performance

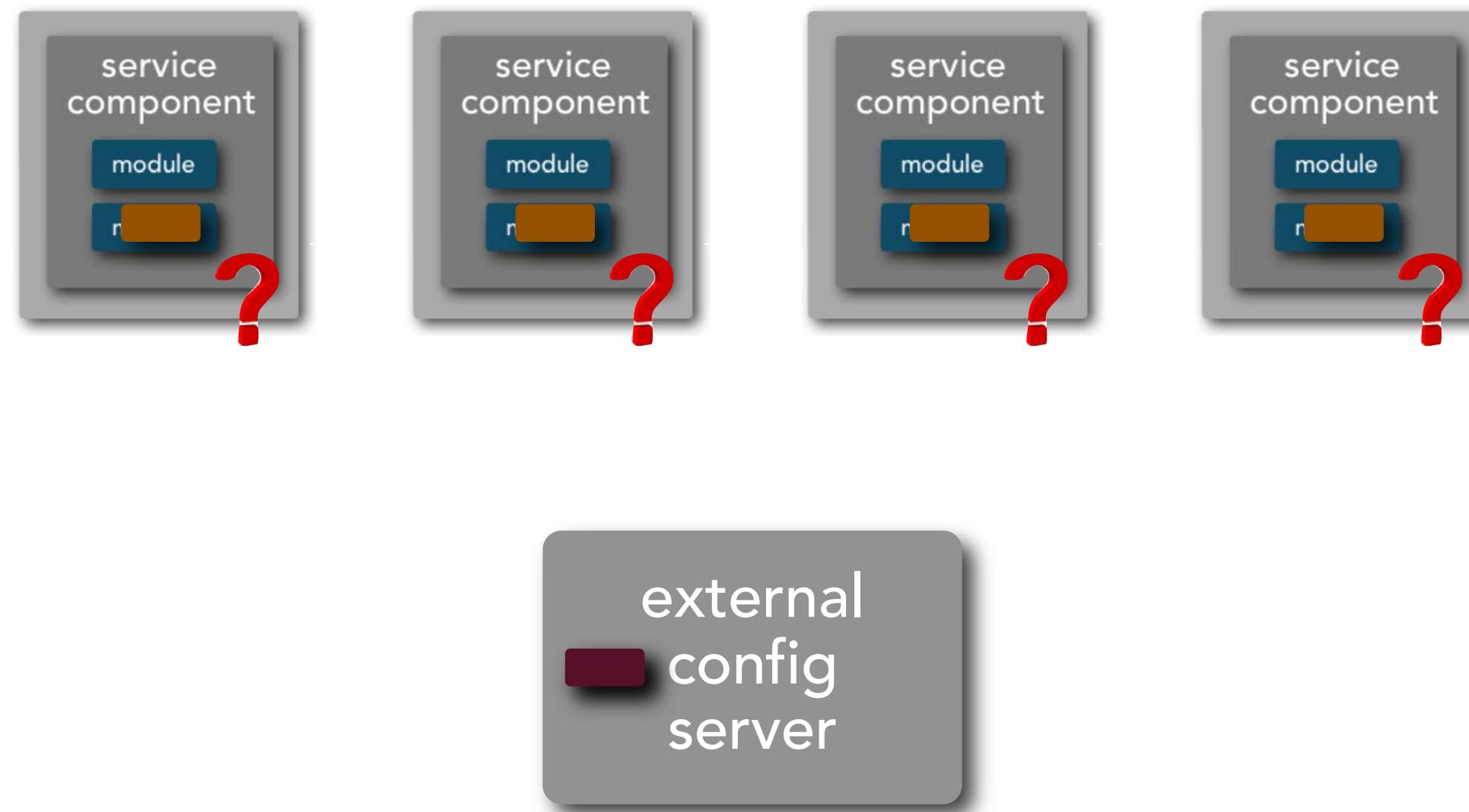


complexity

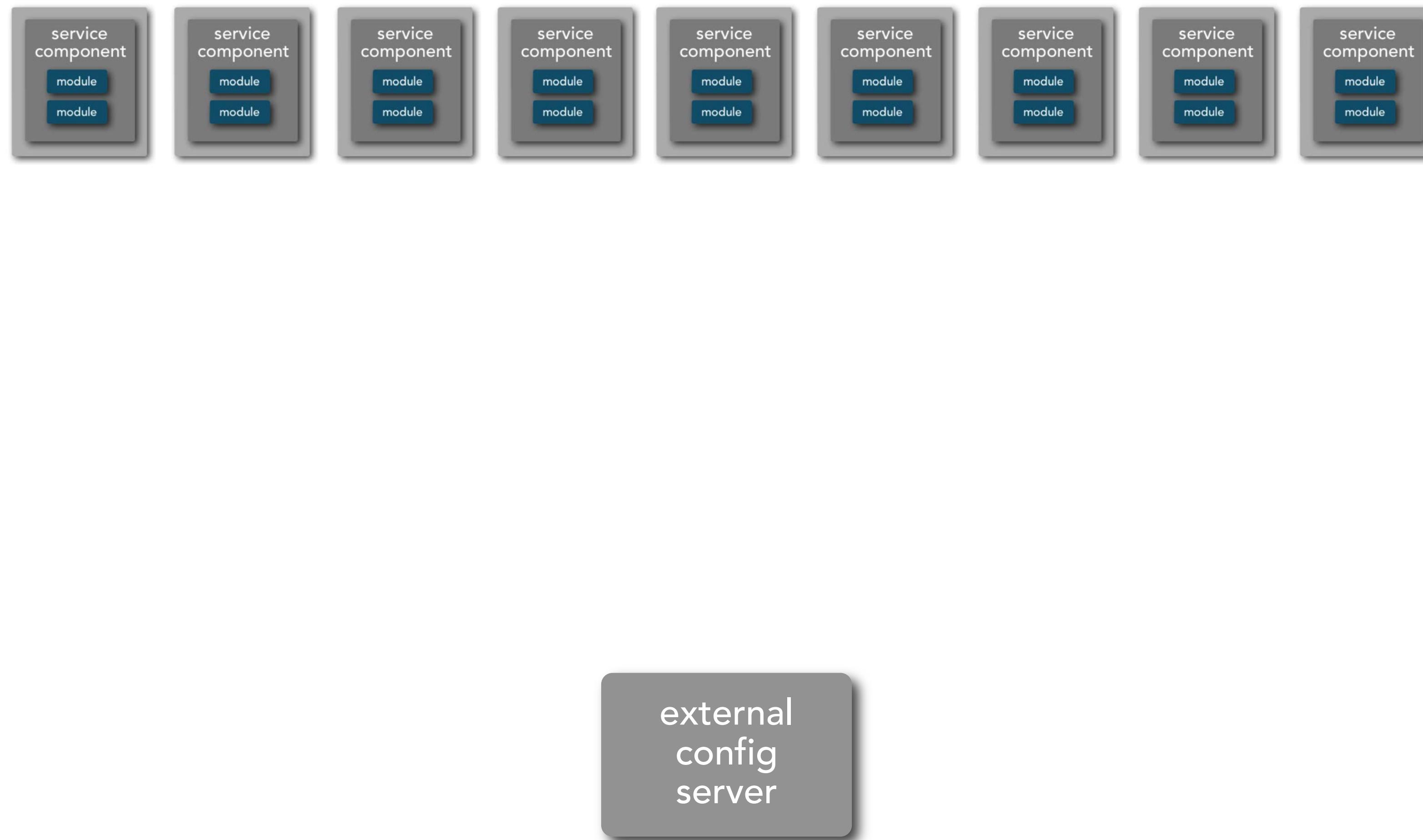
# Watch Notification Pattern

# watch notification pattern

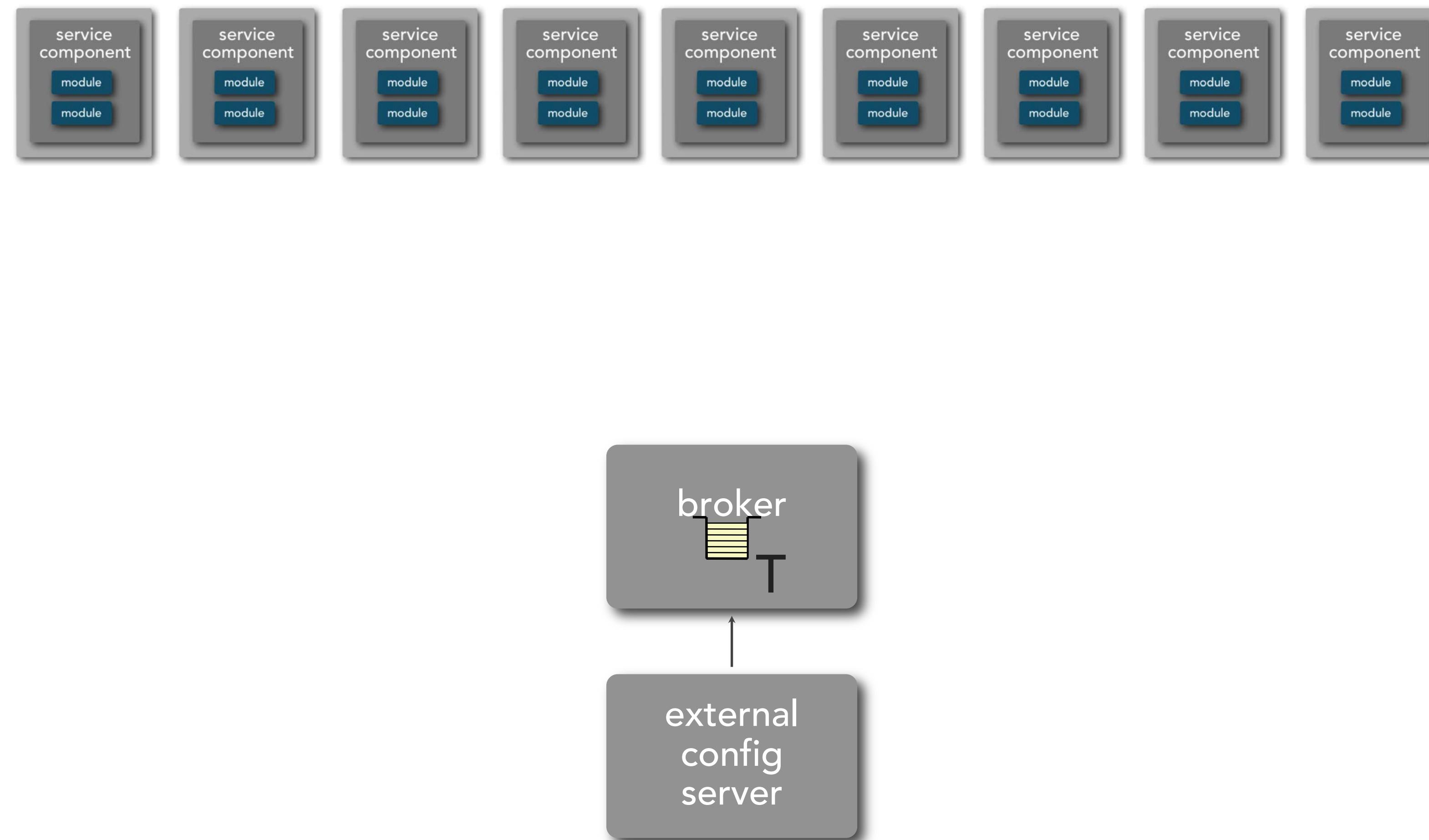
*“how can I send notification events to services without maintaining a persistent connection?”*



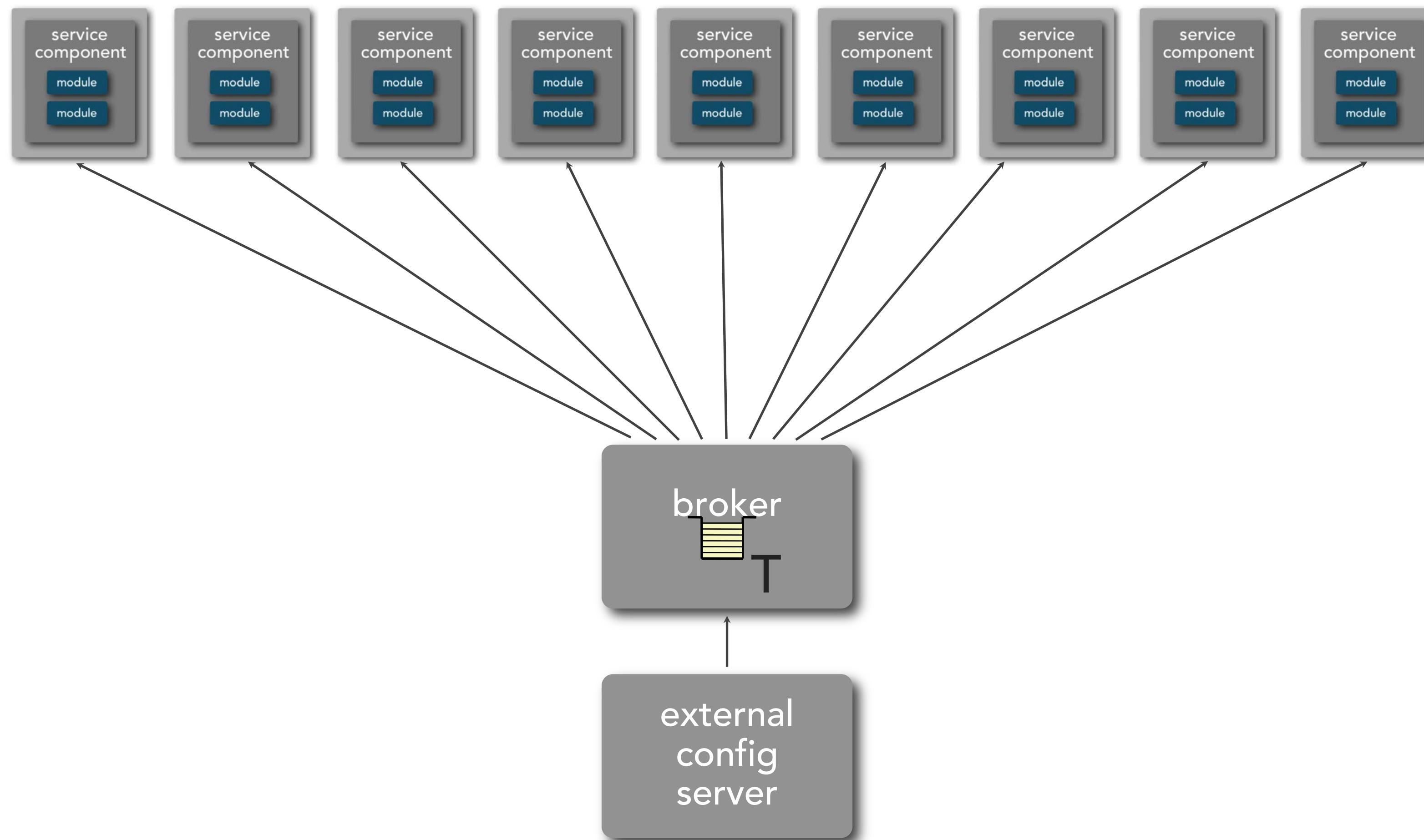
# watch notification pattern



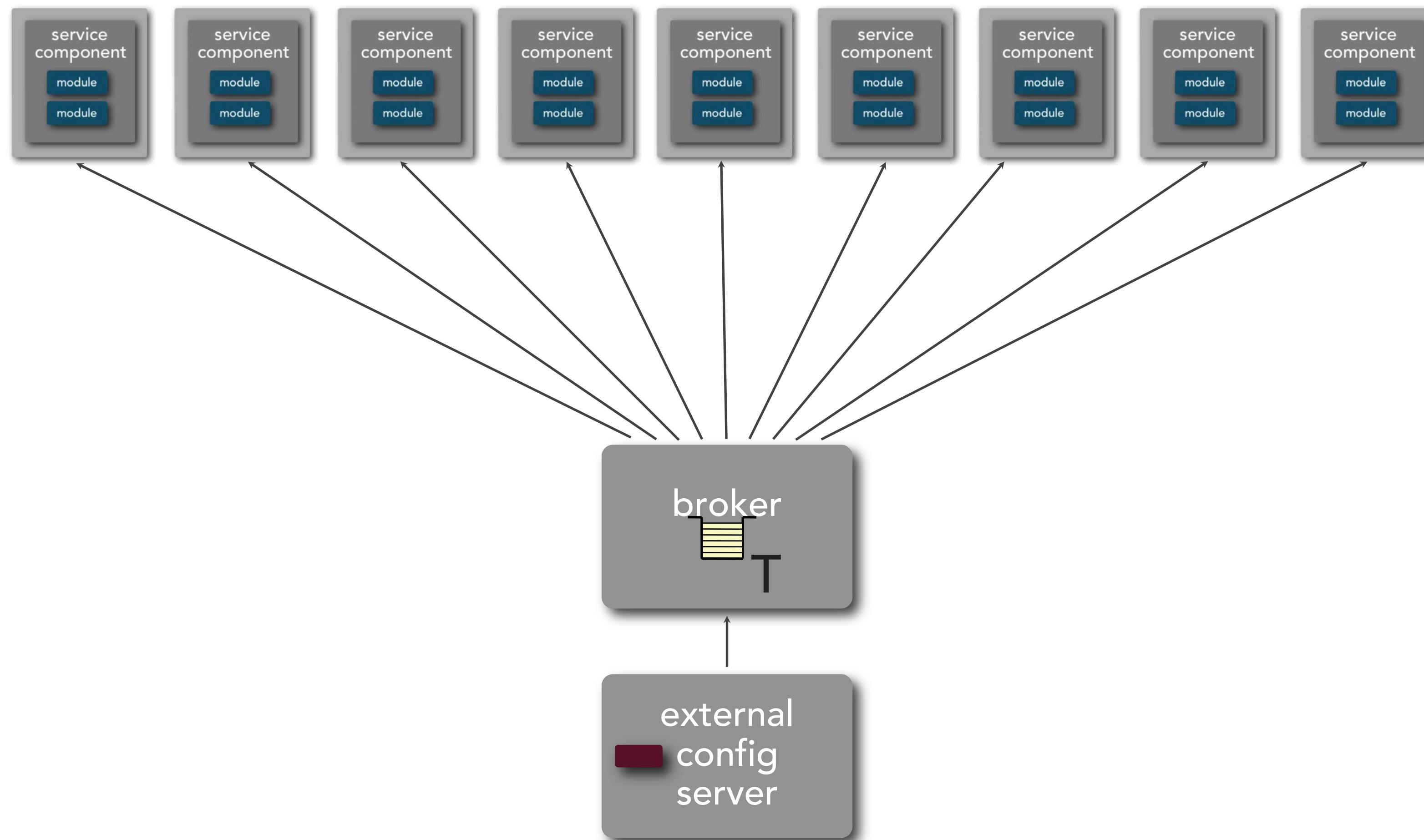
# watch notification pattern



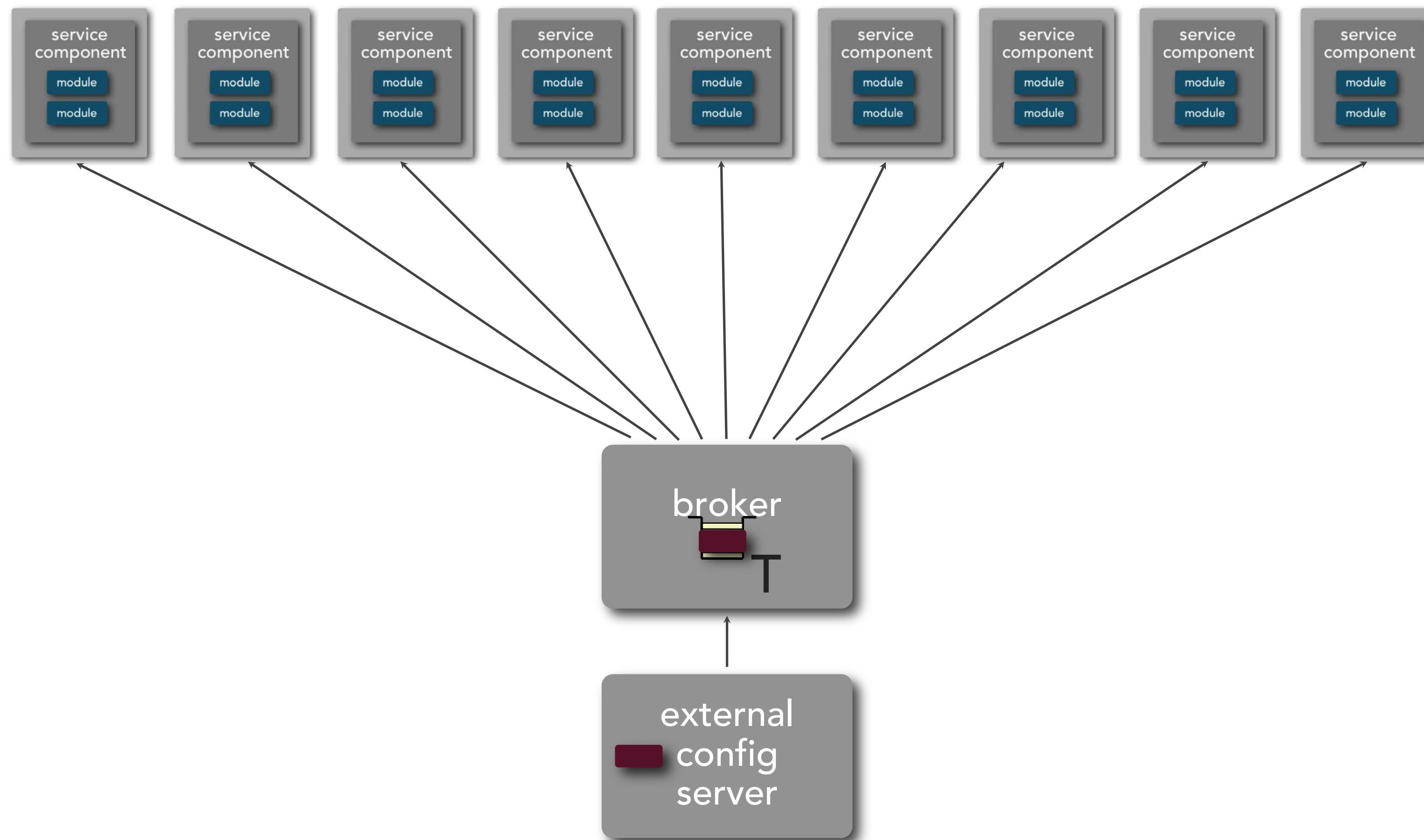
# watch notification pattern



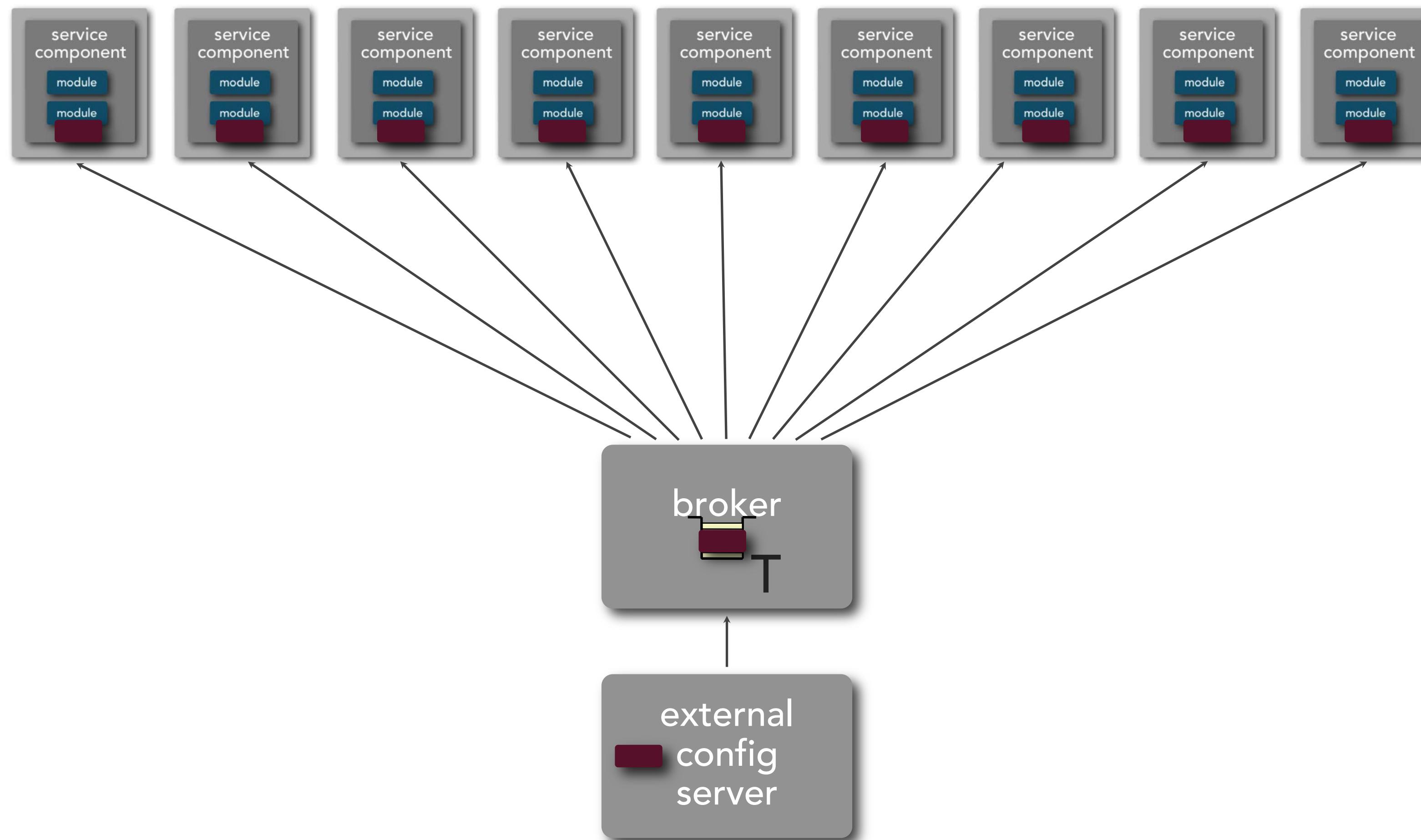
# watch notification pattern



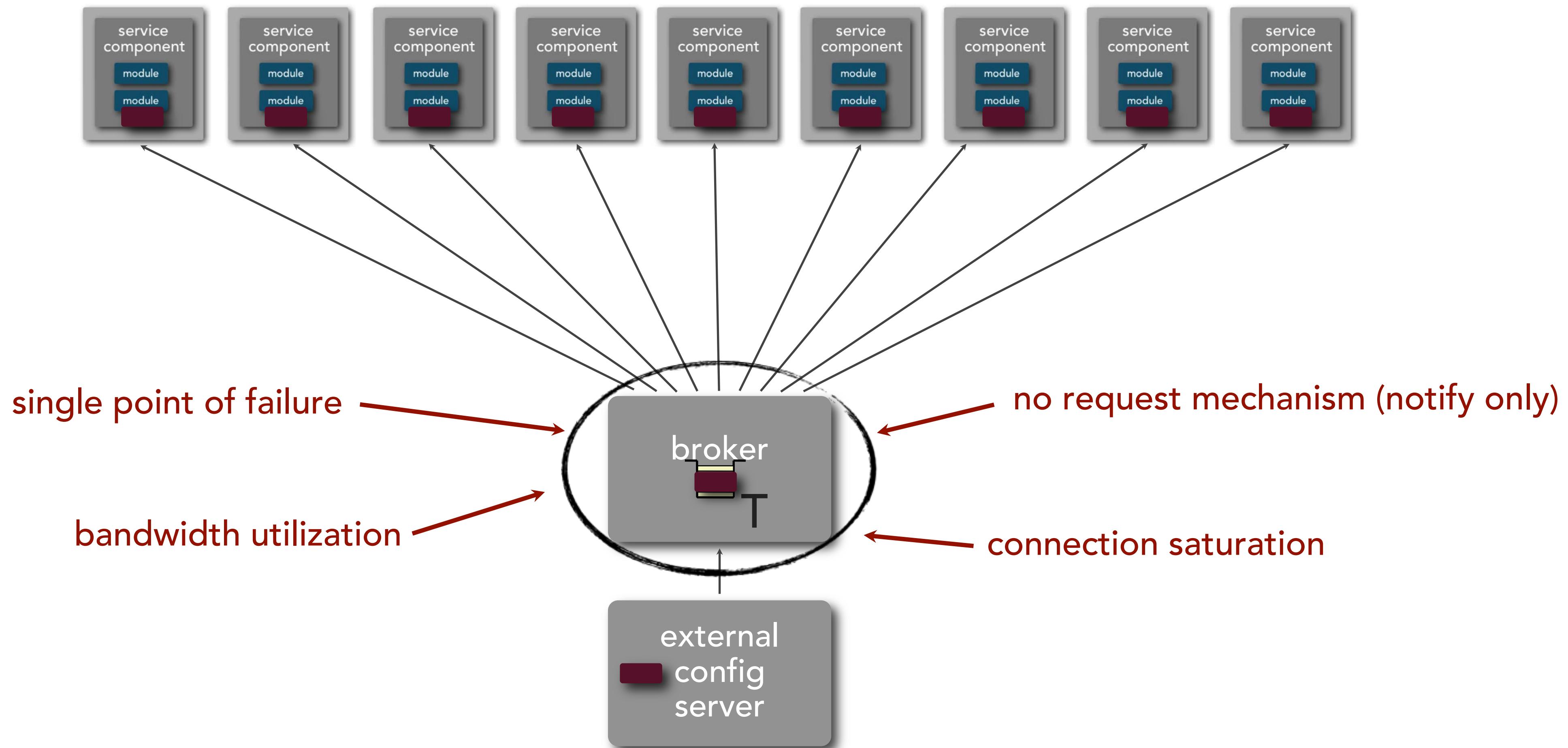
# watch notification pattern



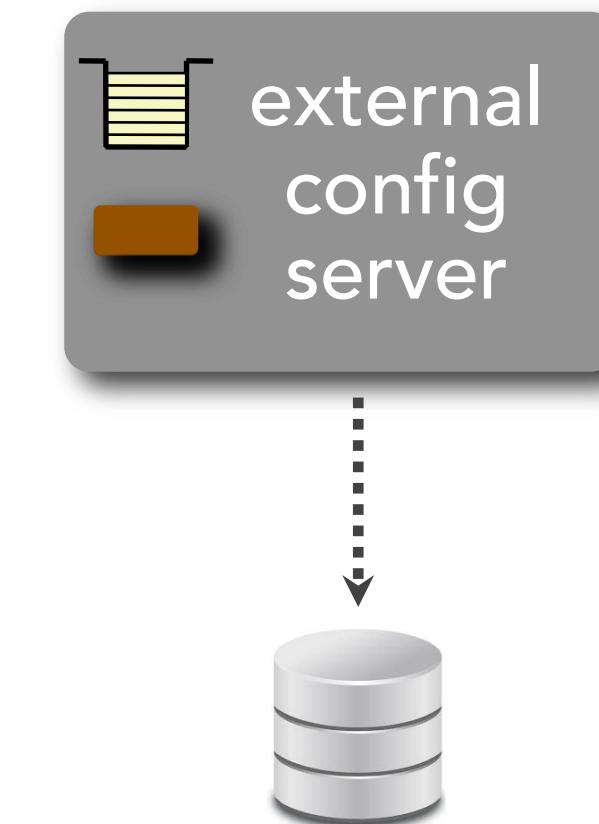
# watch notification pattern



# watch notification pattern

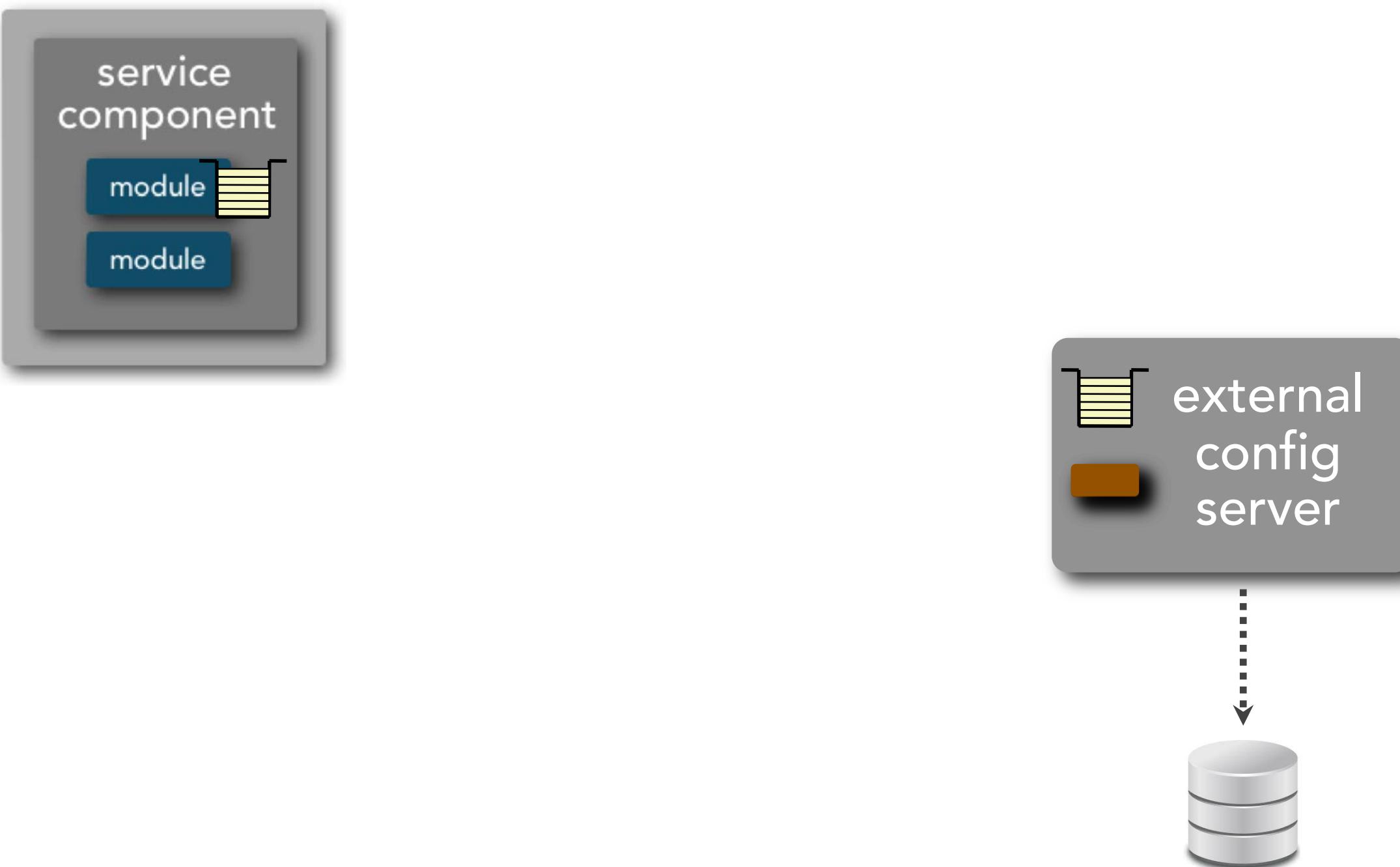


# watch notification pattern



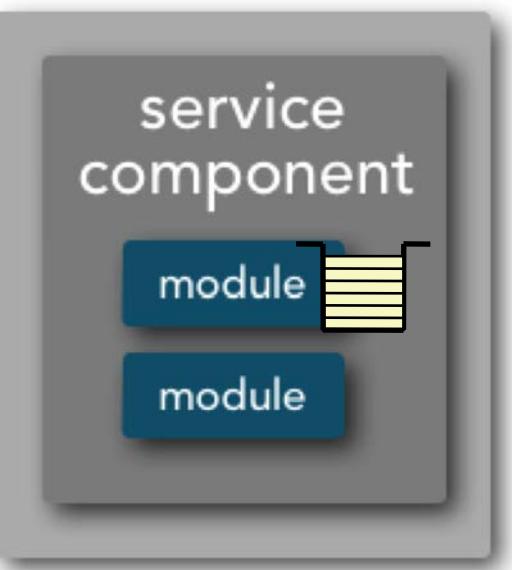
# watch notification pattern

svc1:62880/watch\_q

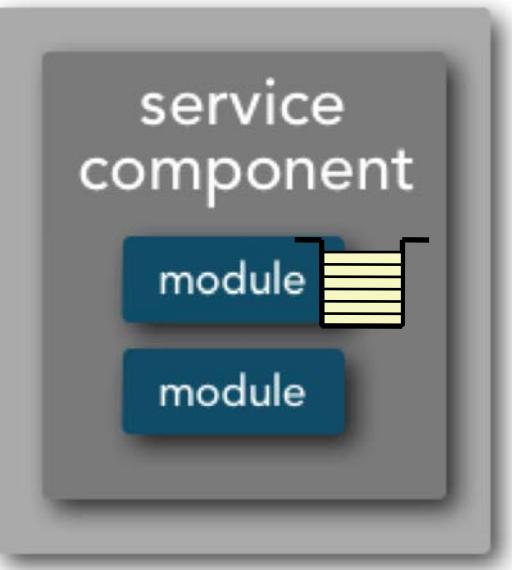


# watch notification pattern

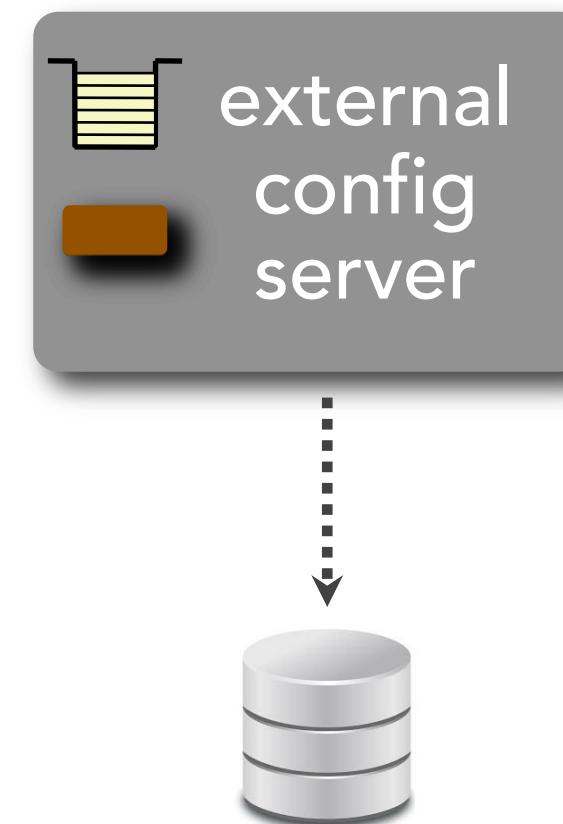
svc1:62880/watch\_q



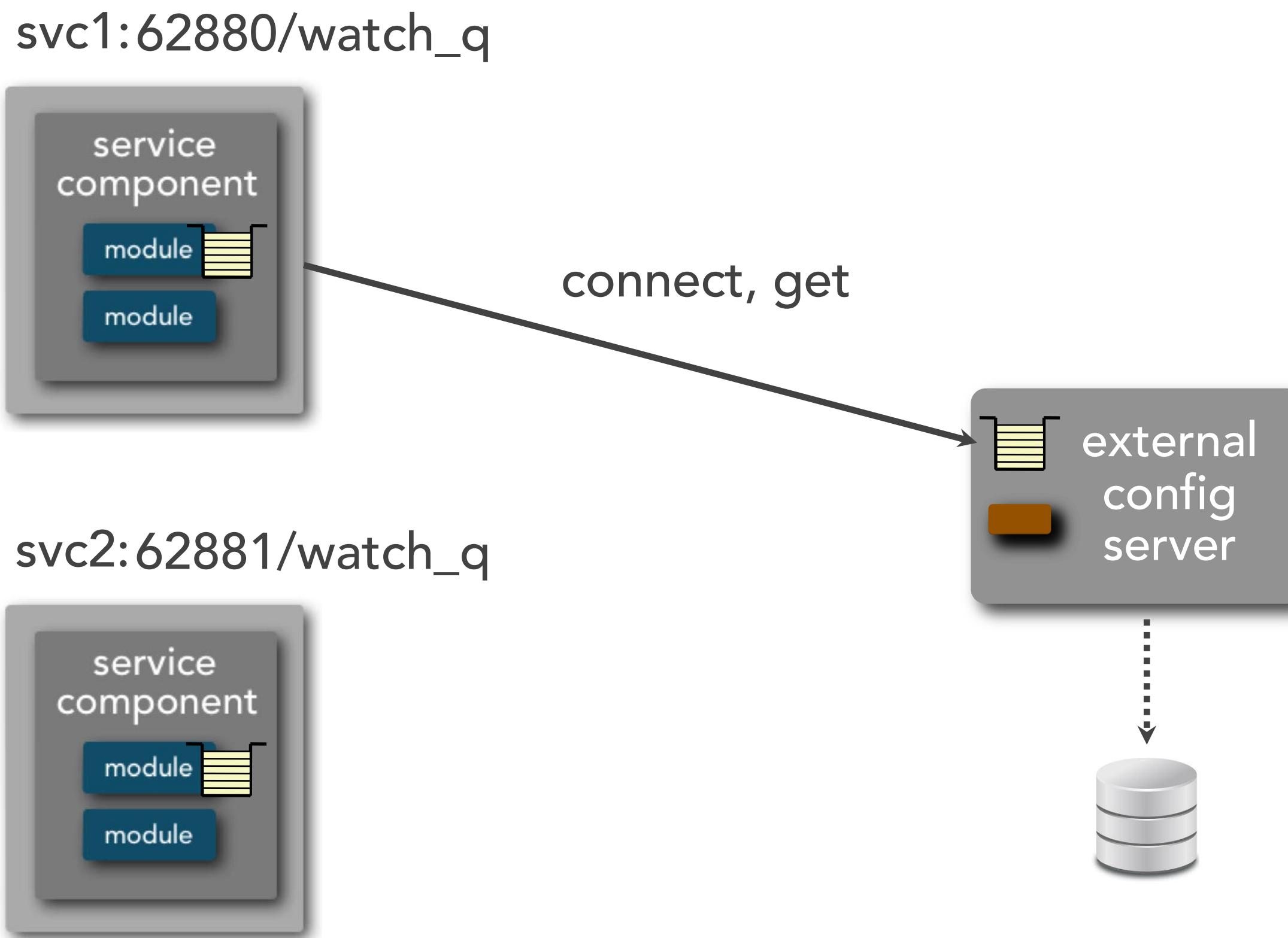
svc2:62881/watch\_q



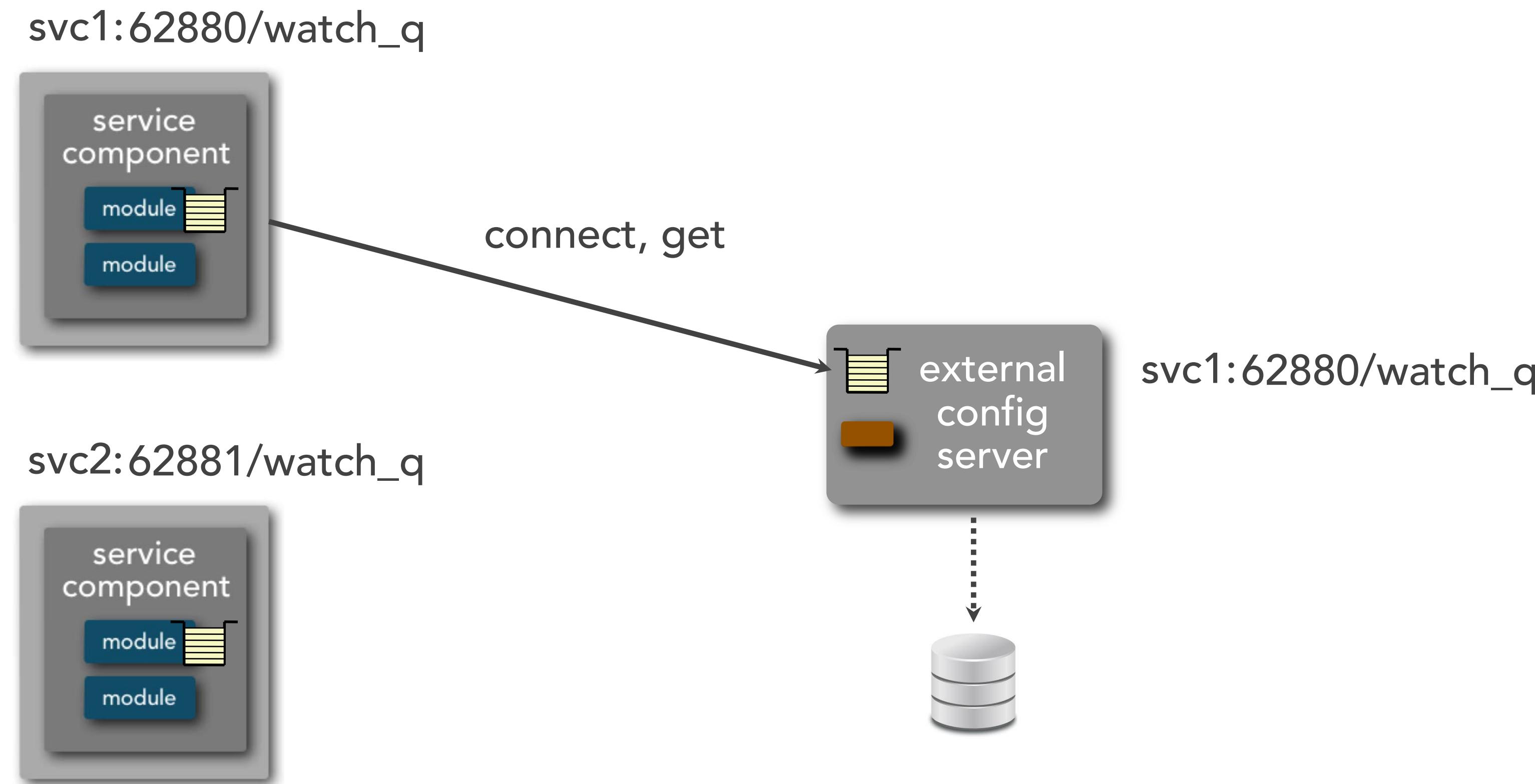
external config server



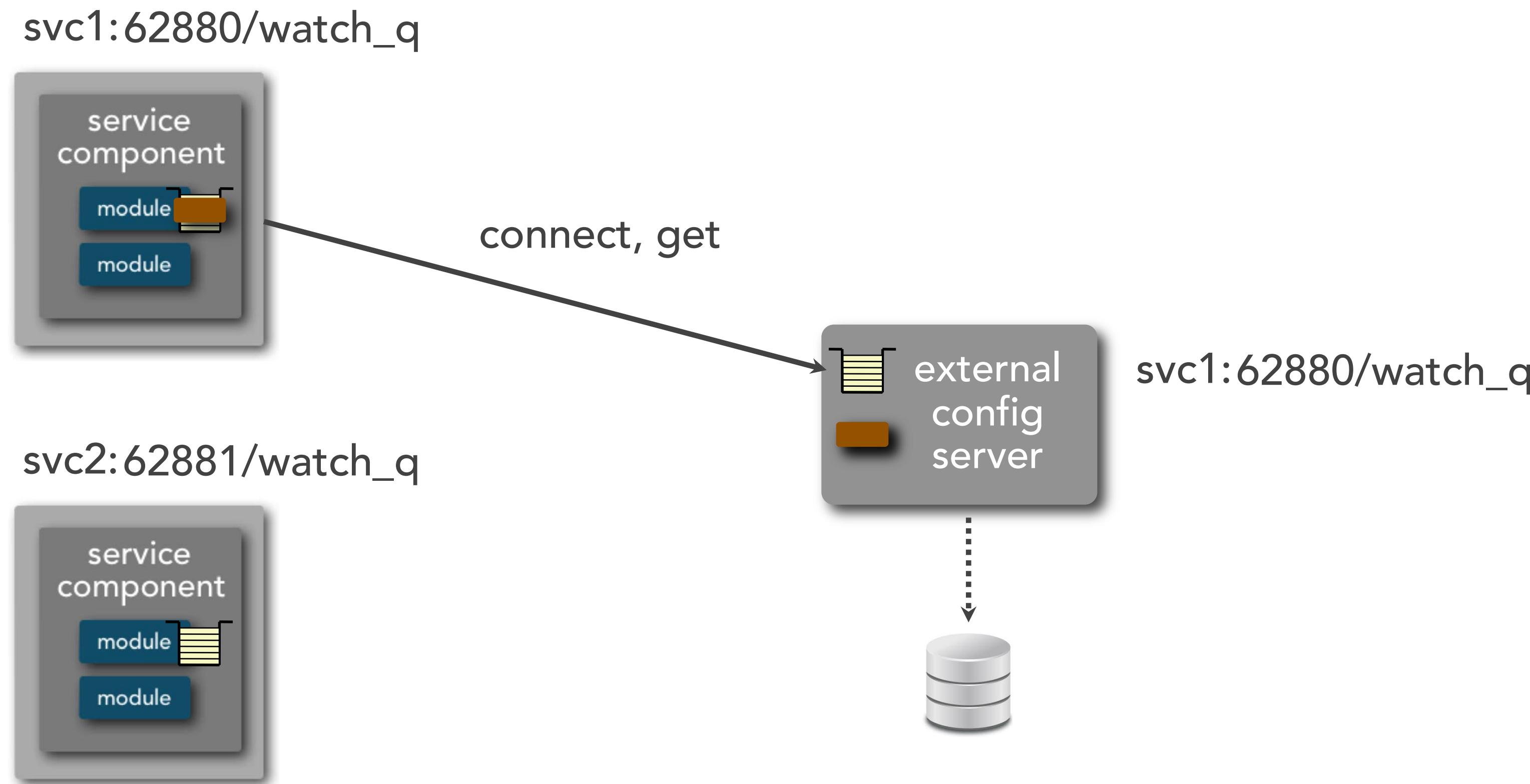
# watch notification pattern



# watch notification pattern

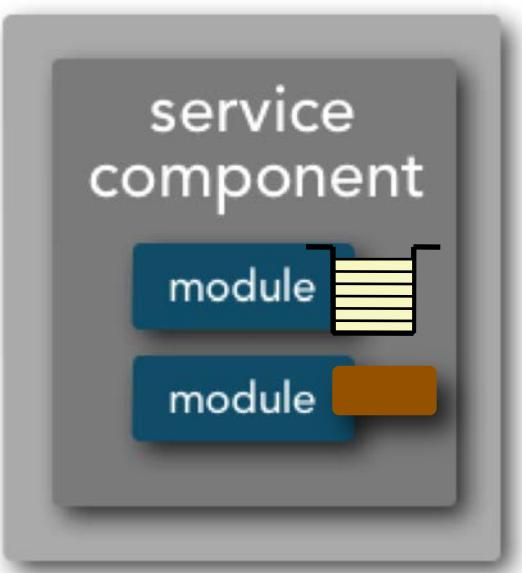


# watch notification pattern

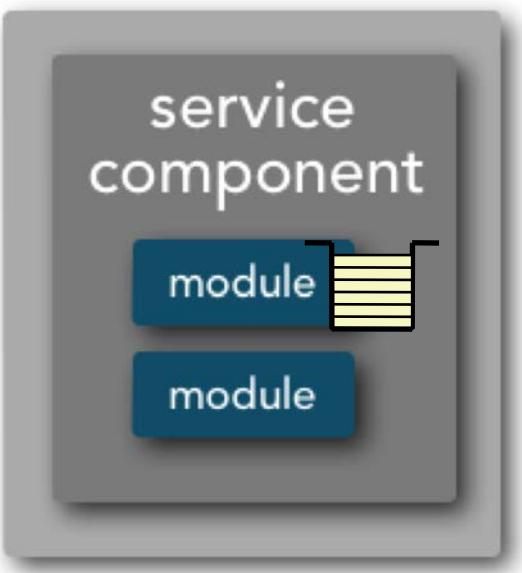


# watch notification pattern

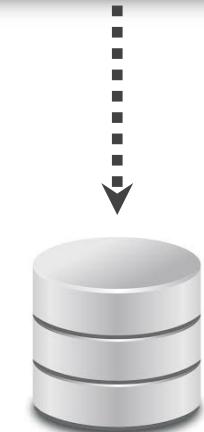
svc1:62880/watch\_q



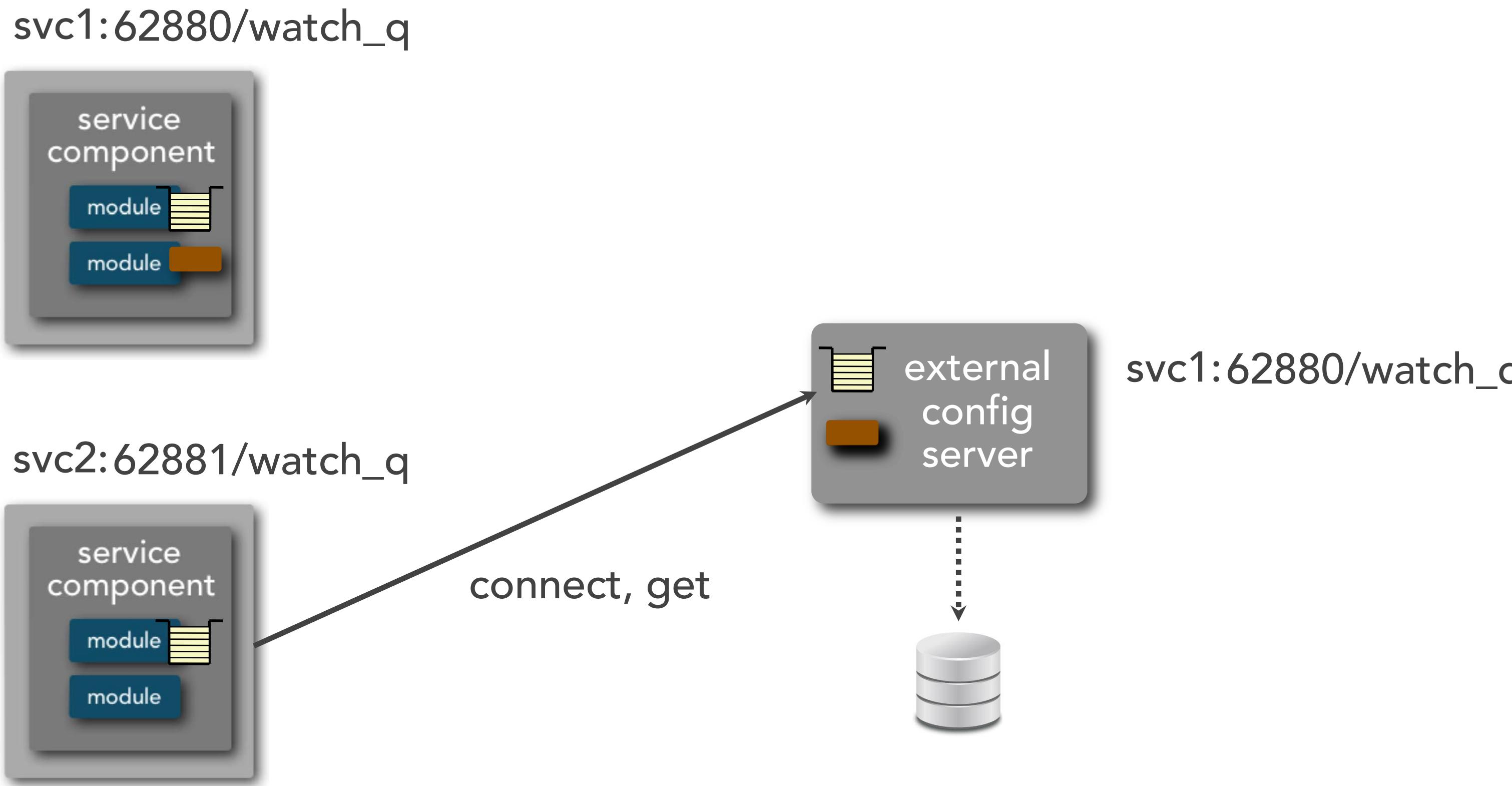
svc2:62881/watch\_q



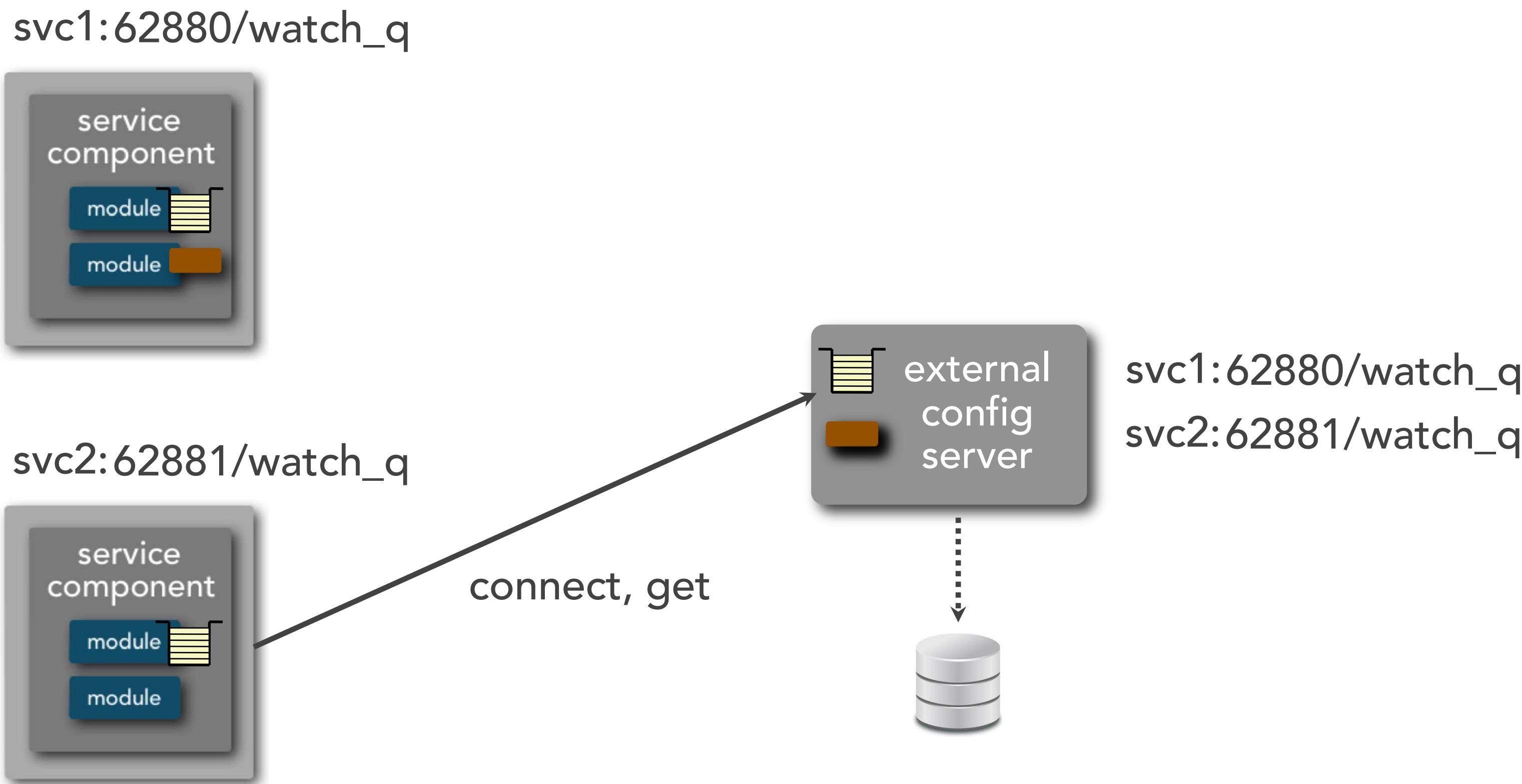
svc1:62880/watch\_q



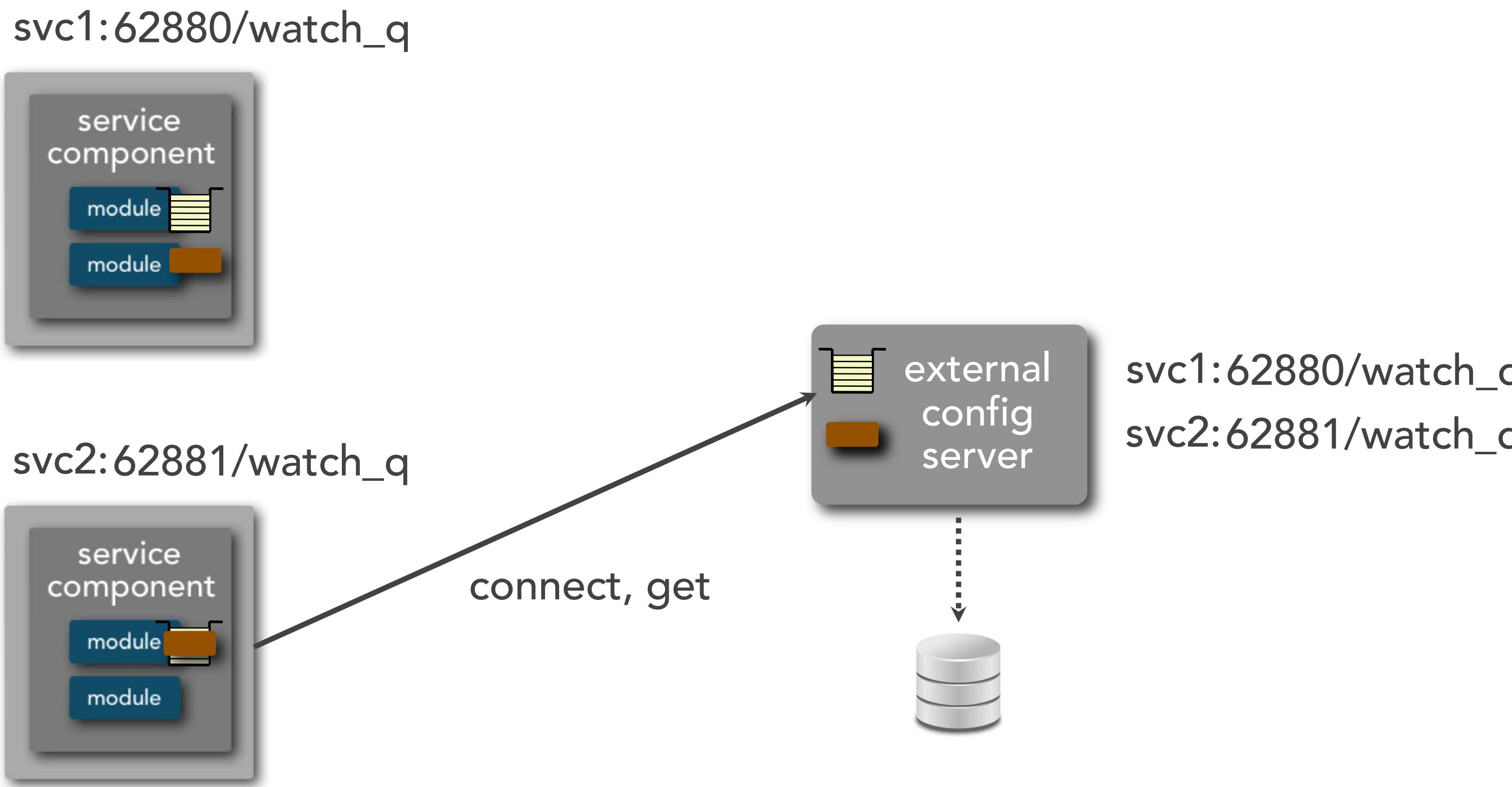
# watch notification pattern



# watch notification pattern

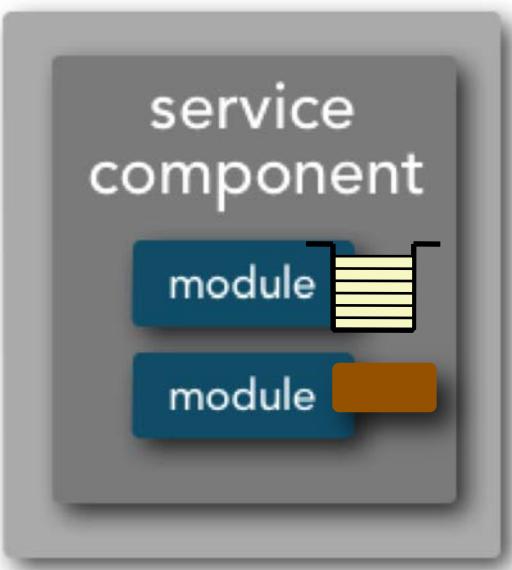


# watch notification pattern

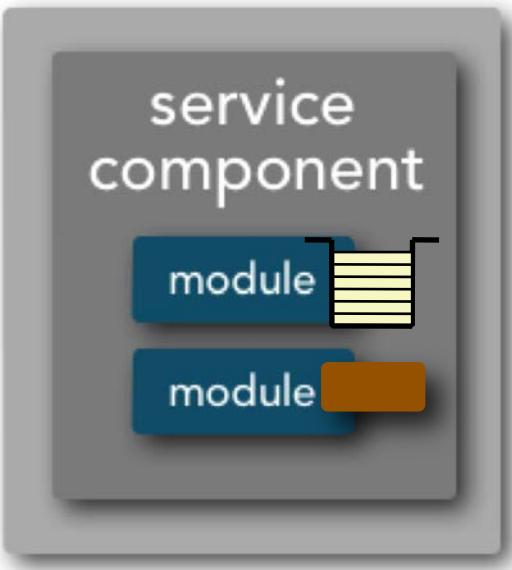


# watch notification pattern

svc1:62880/watch\_q



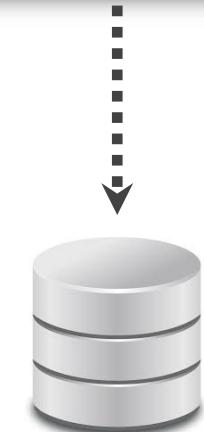
svc2:62881/watch\_q



external config server

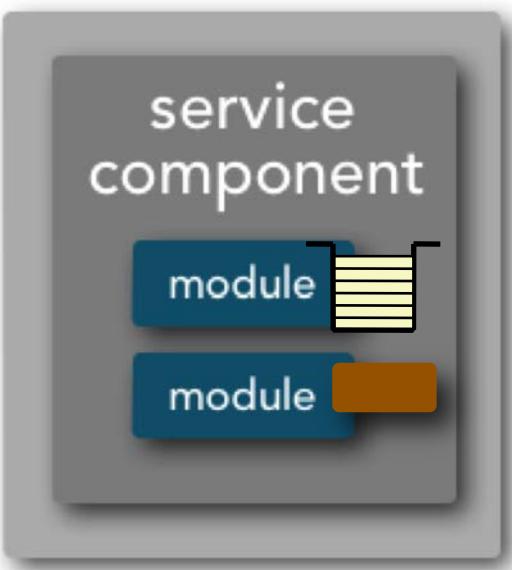
svc1:62880/watch\_q

svc2:62881/watch\_q

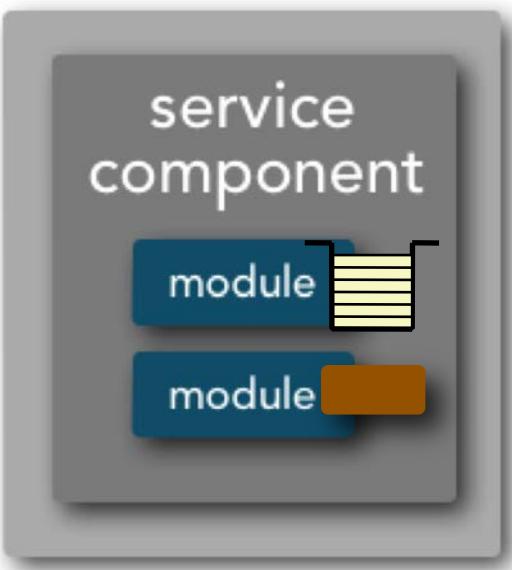


# watch notification pattern

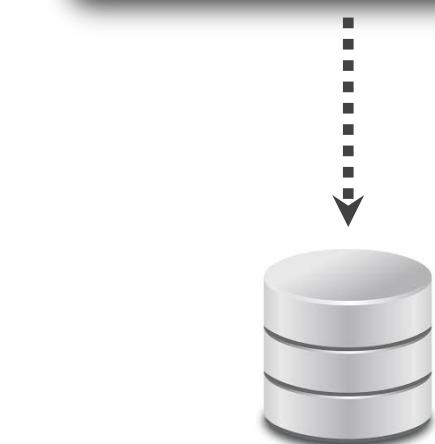
svc1:62880/watch\_q



svc2:62881/watch\_q



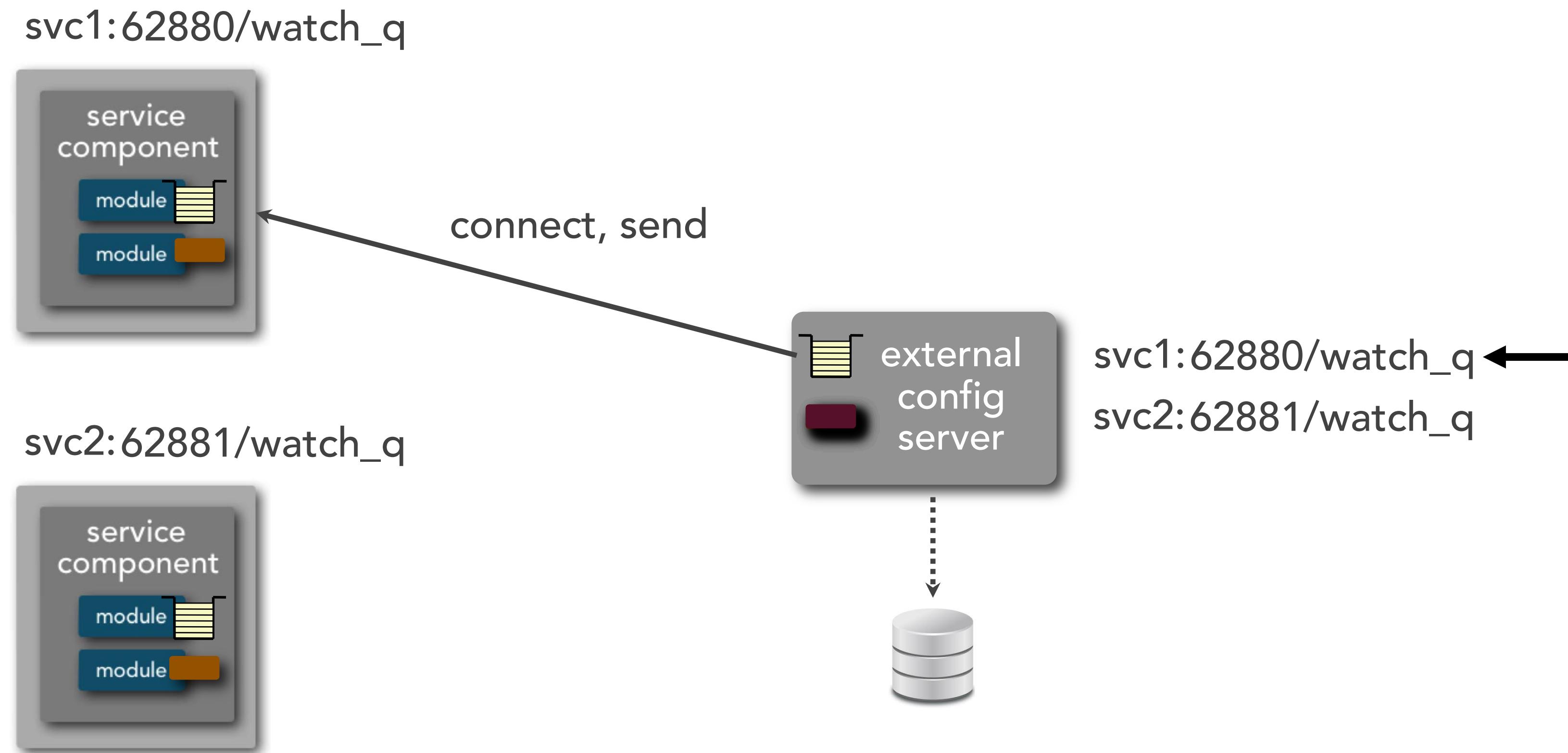
external config server



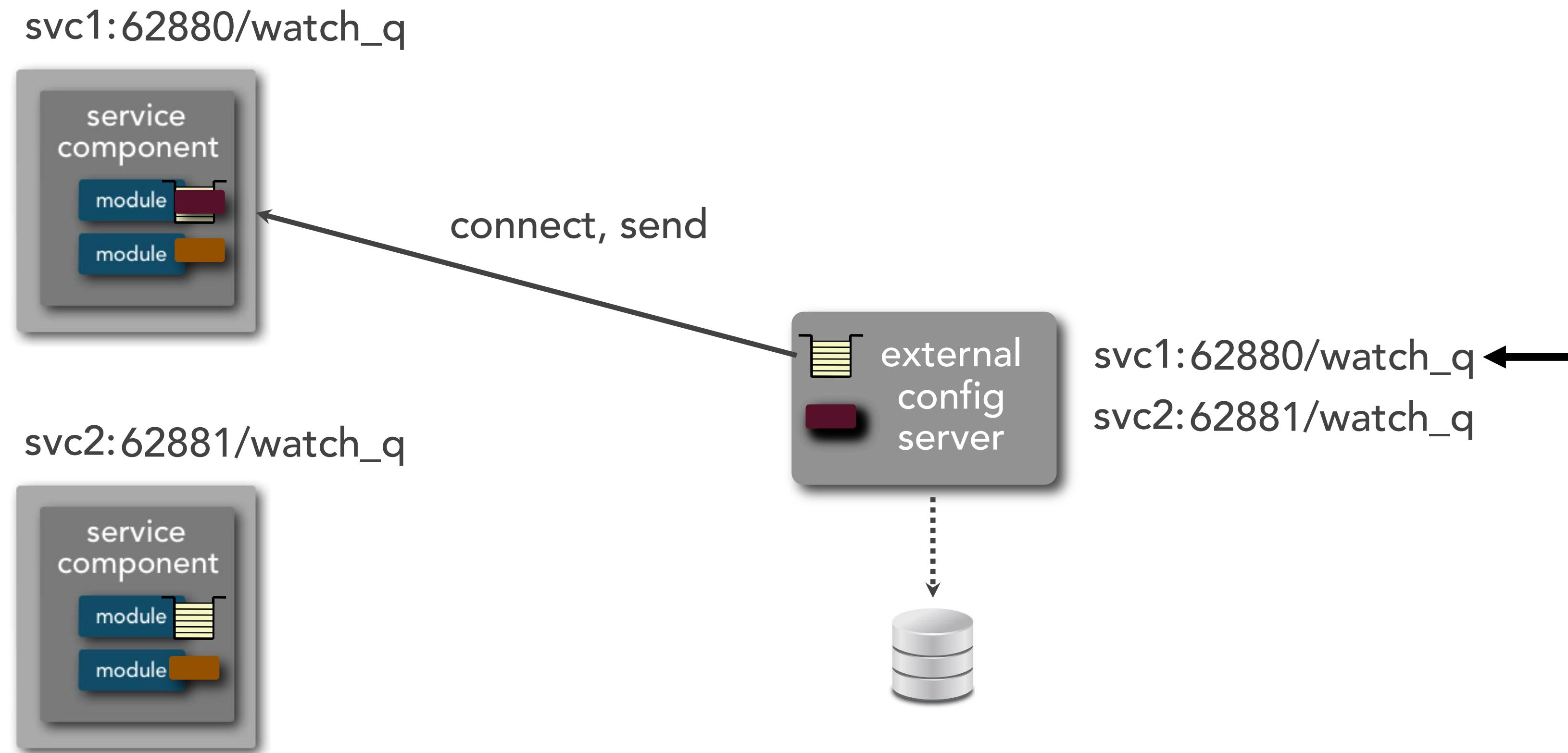
svc1:62880/watch\_q

svc2:62881/watch\_q

# watch notification pattern

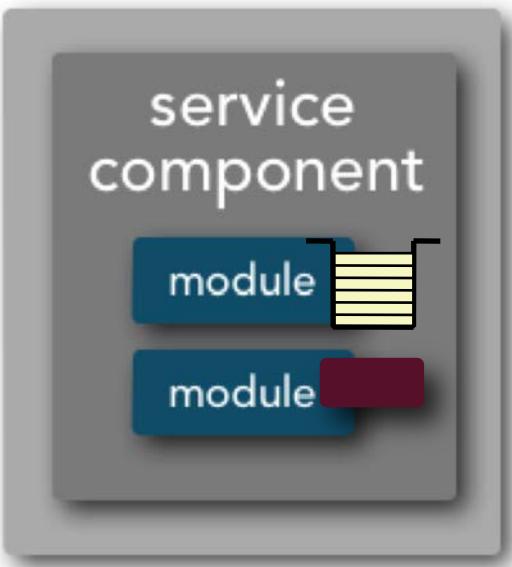


# watch notification pattern

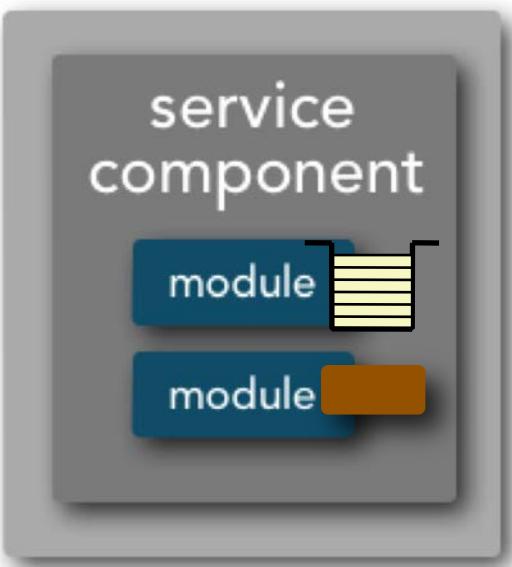


# watch notification pattern

svc1:62880/watch\_q

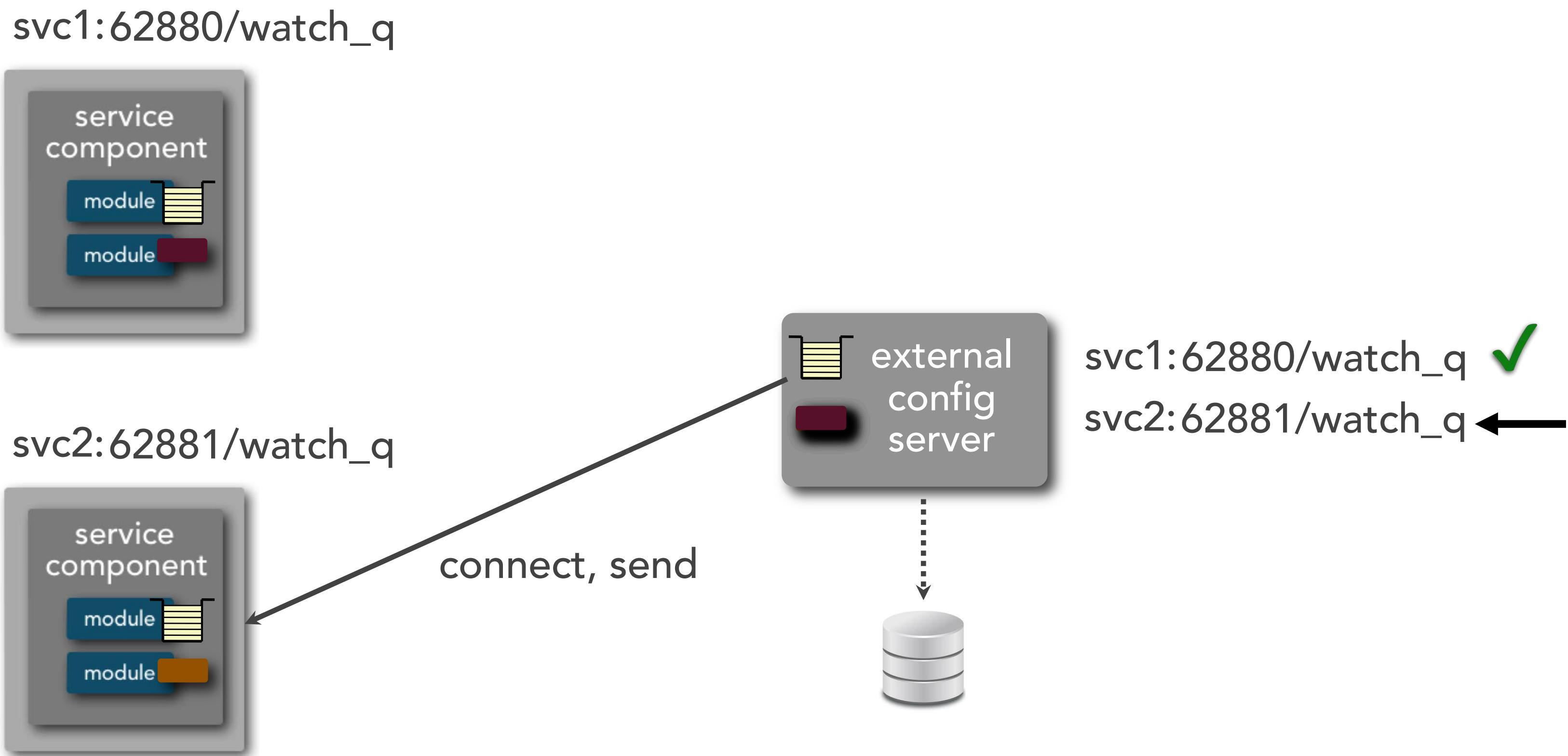


svc2:62881/watch\_q

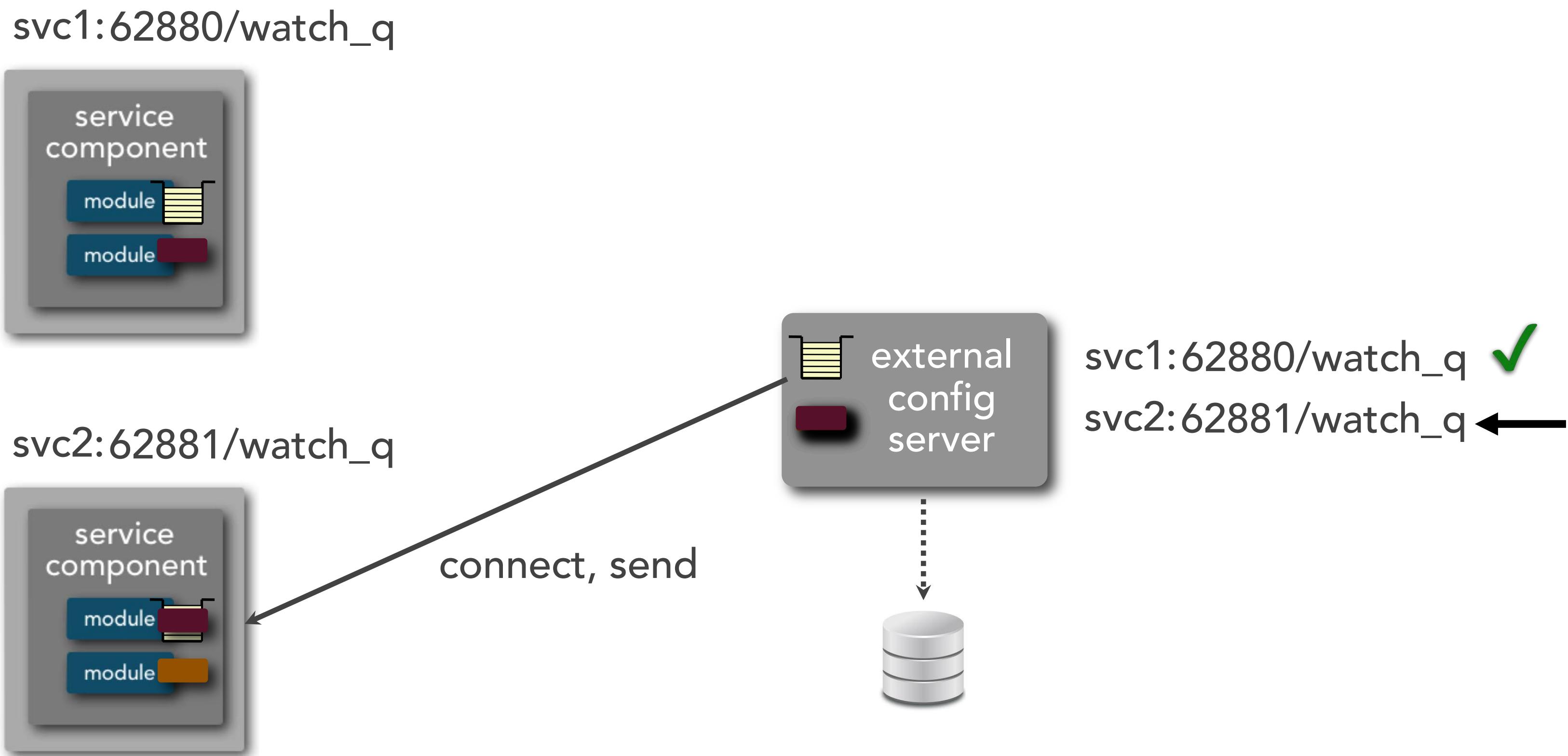


svc1:62880/watch\_q ✓  
svc2:62881/watch\_q

# watch notification pattern

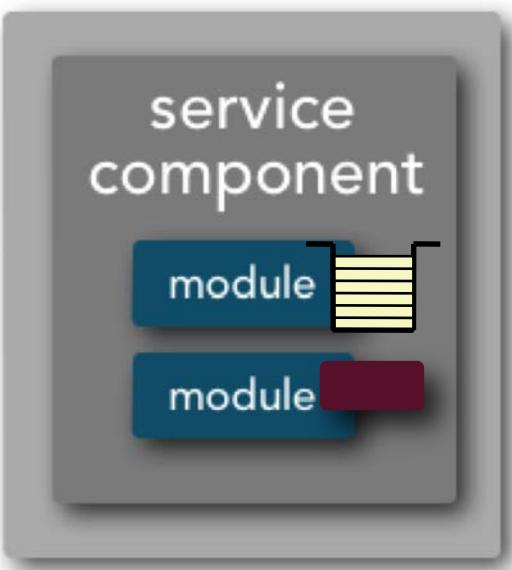


# watch notification pattern

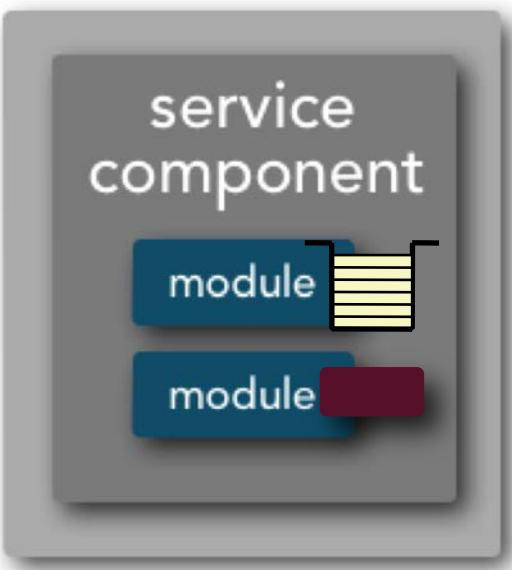


# watch notification pattern

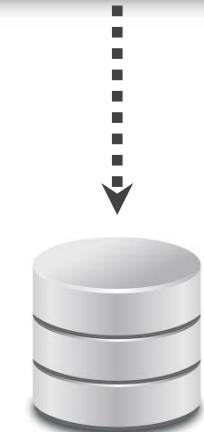
svc1:62880/watch\_q



svc2:62881/watch\_q



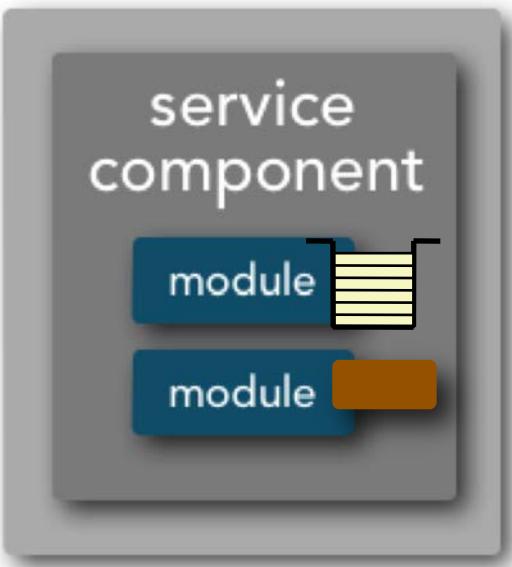
external config server



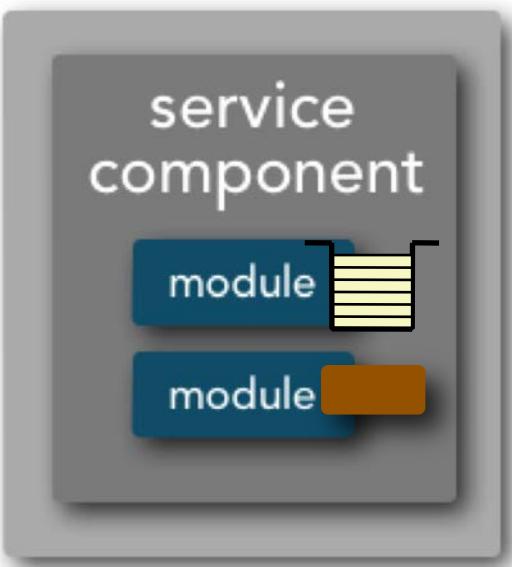
svc1:62880/watch\_q ✓  
svc2:62881/watch\_q ✓

# watch notification pattern

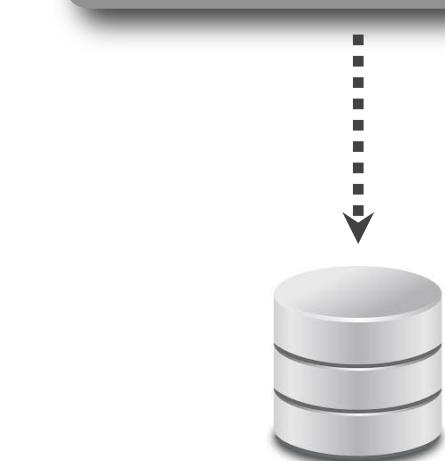
svc1:62880/watch\_q



svc2:62881/watch\_q



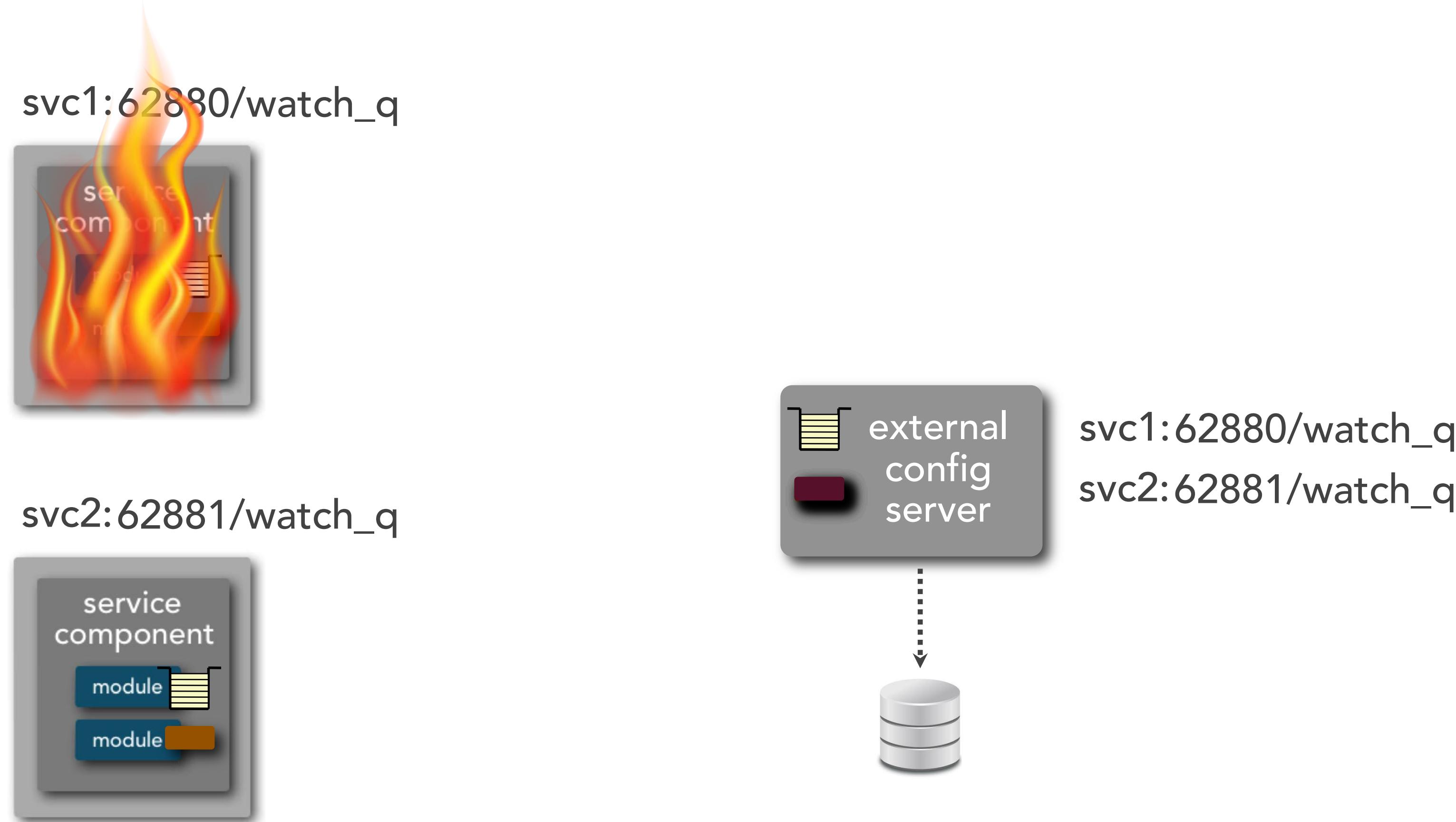
external config server



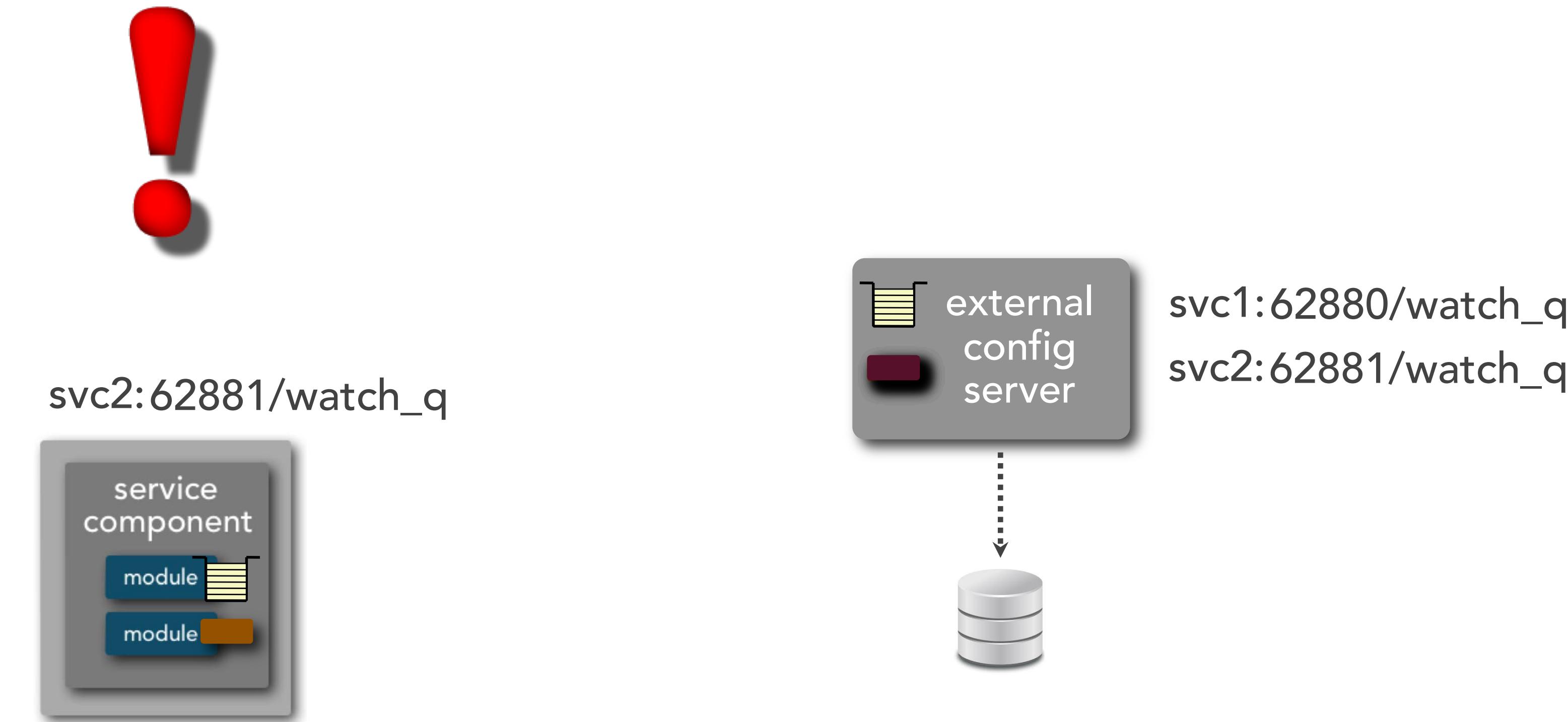
svc1:62880/watch\_q

svc2:62881/watch\_q

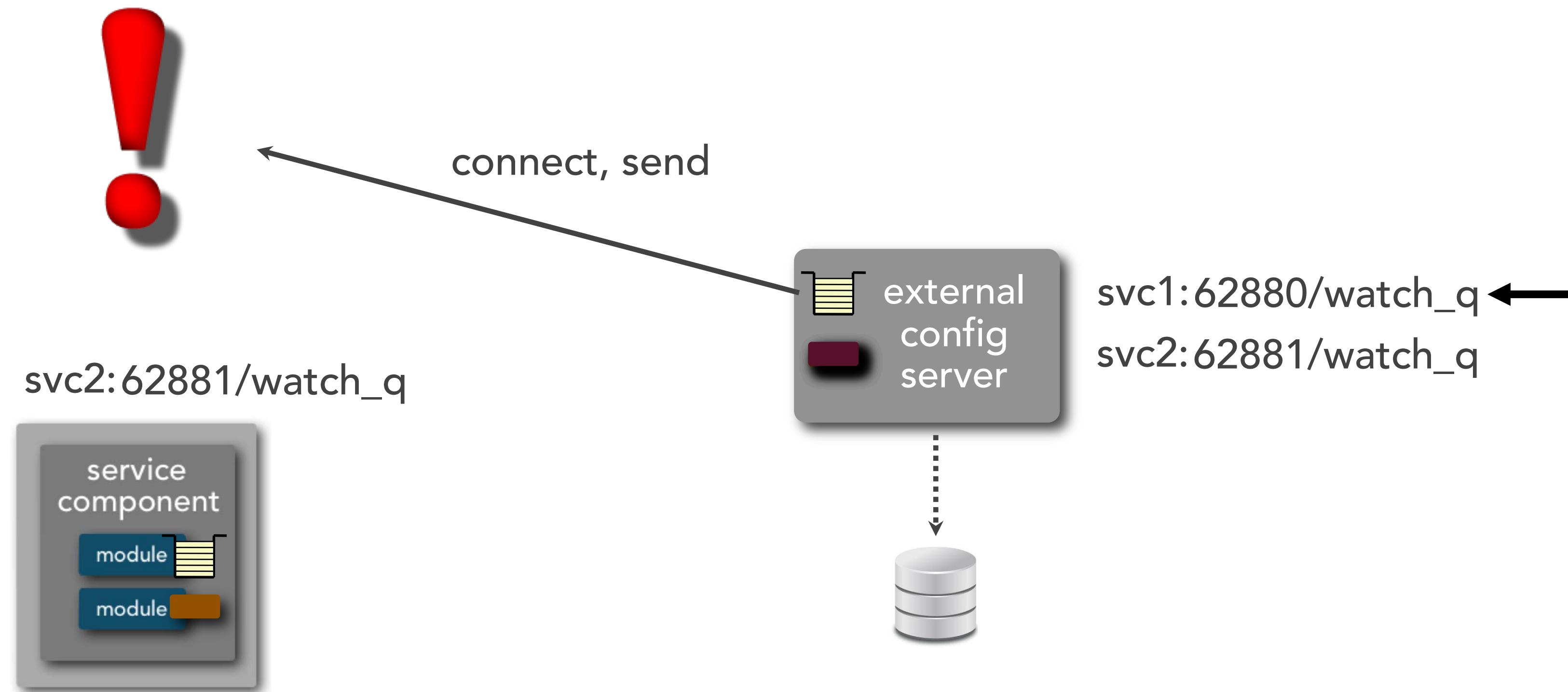
# watch notification pattern



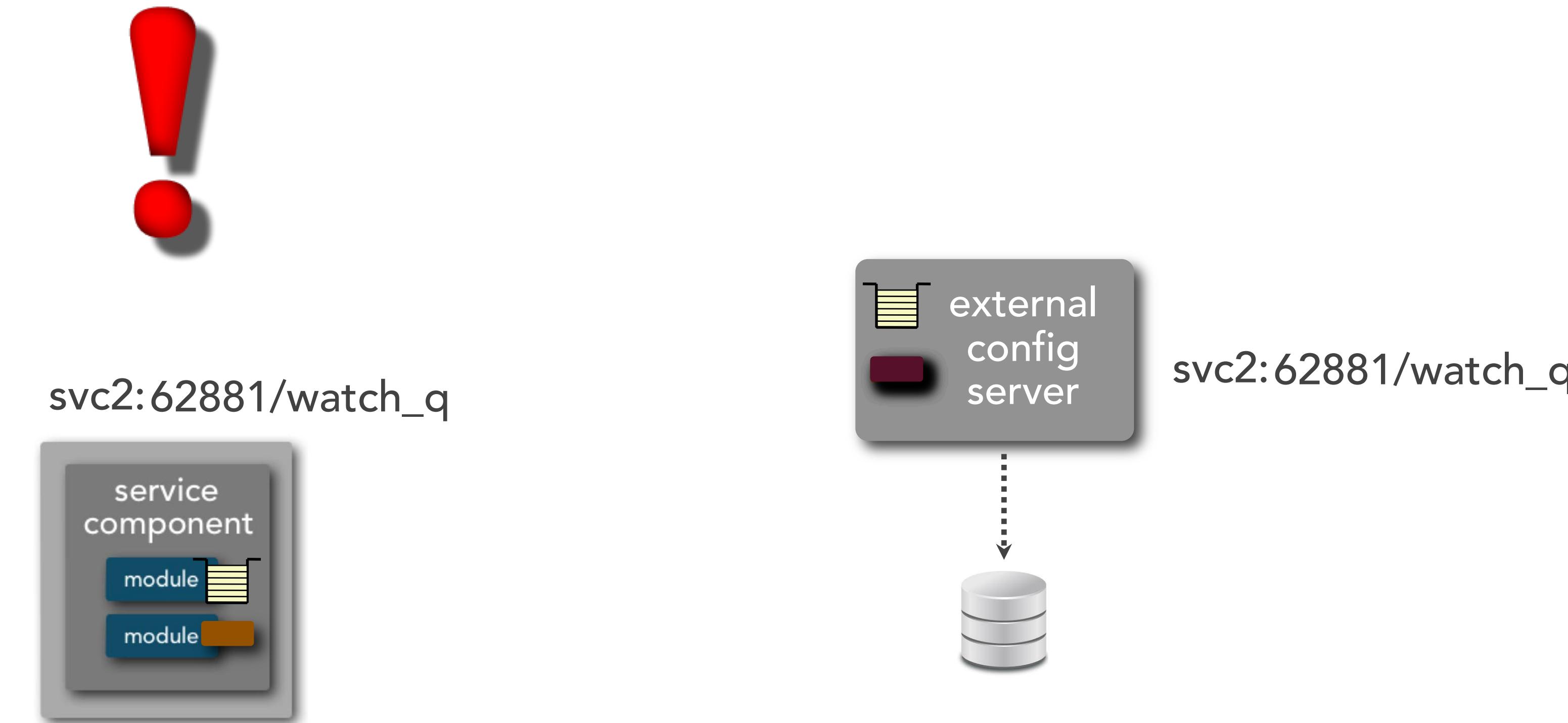
# watch notification pattern



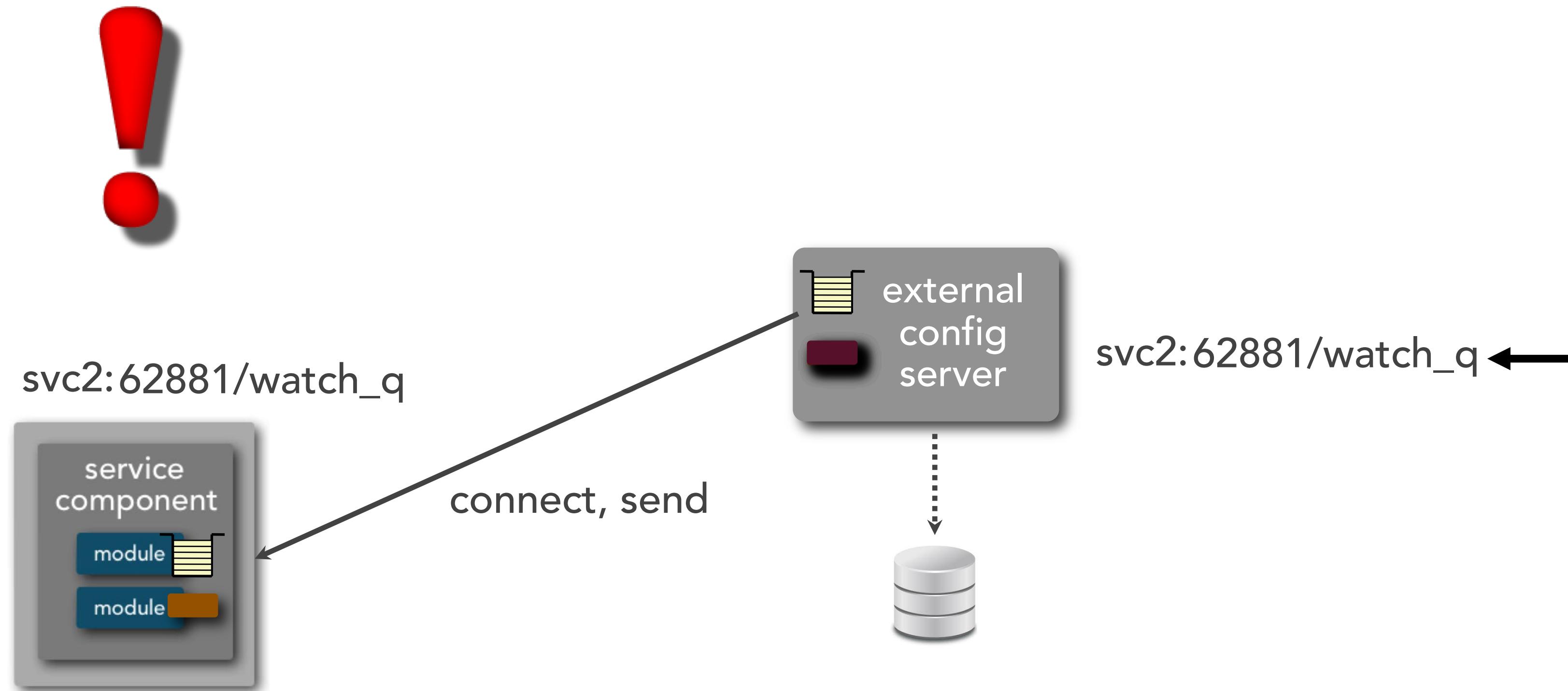
# watch notification pattern



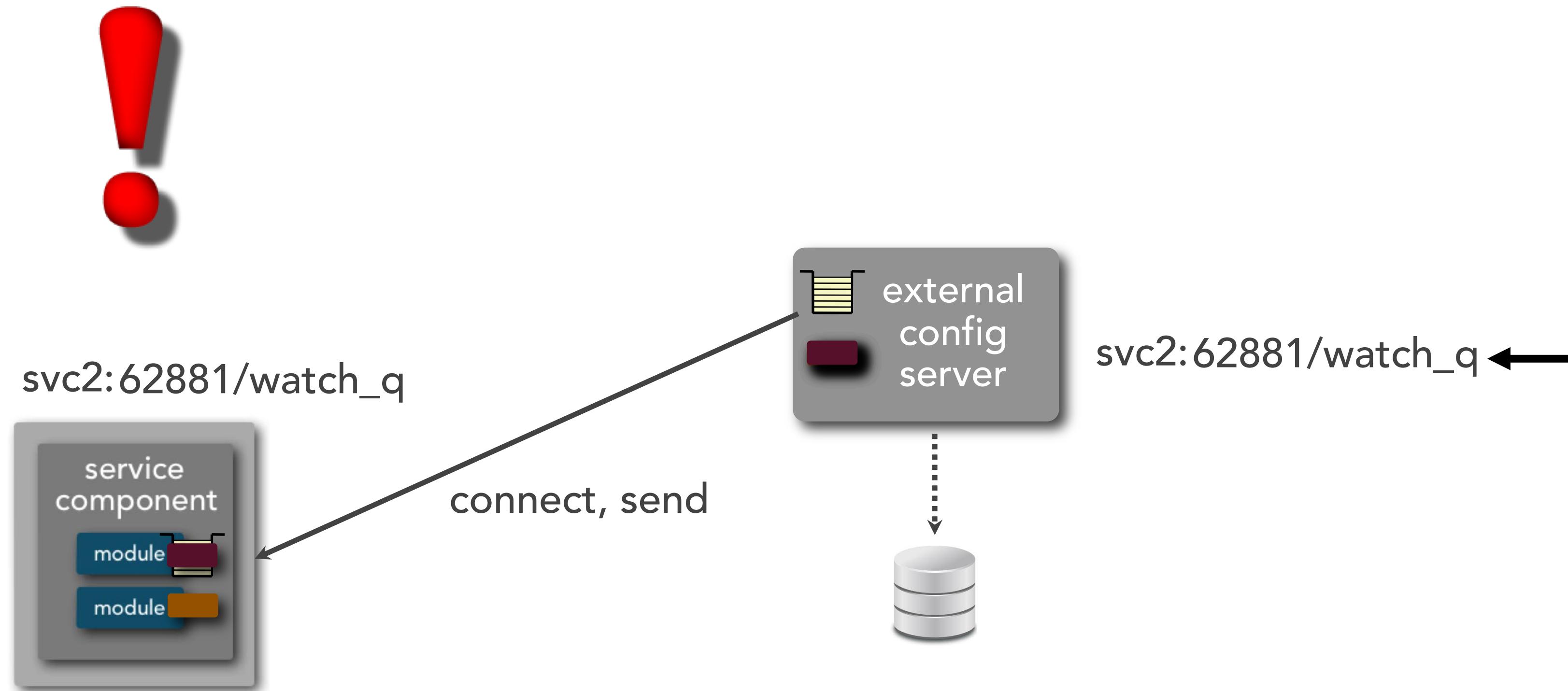
# watch notification pattern



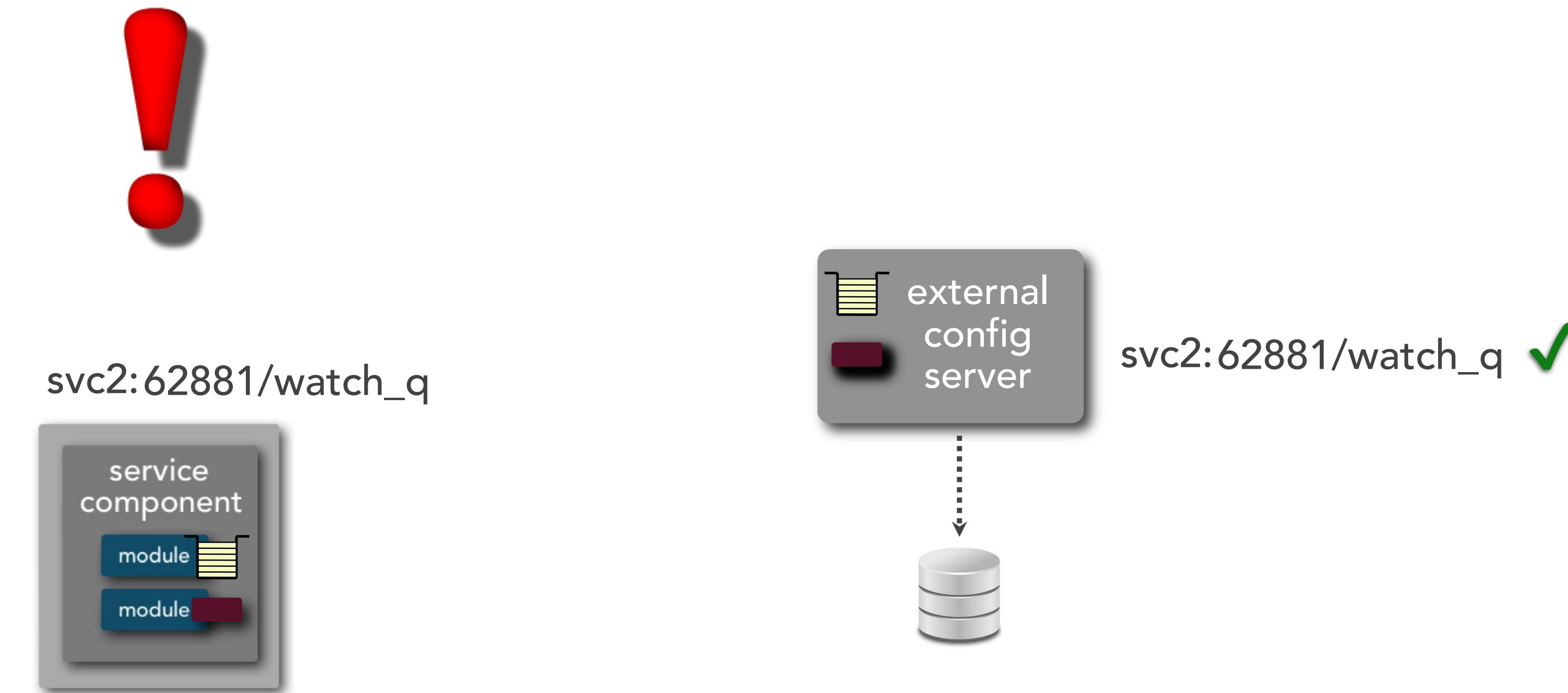
# watch notification pattern



# watch notification pattern



# watch notification pattern



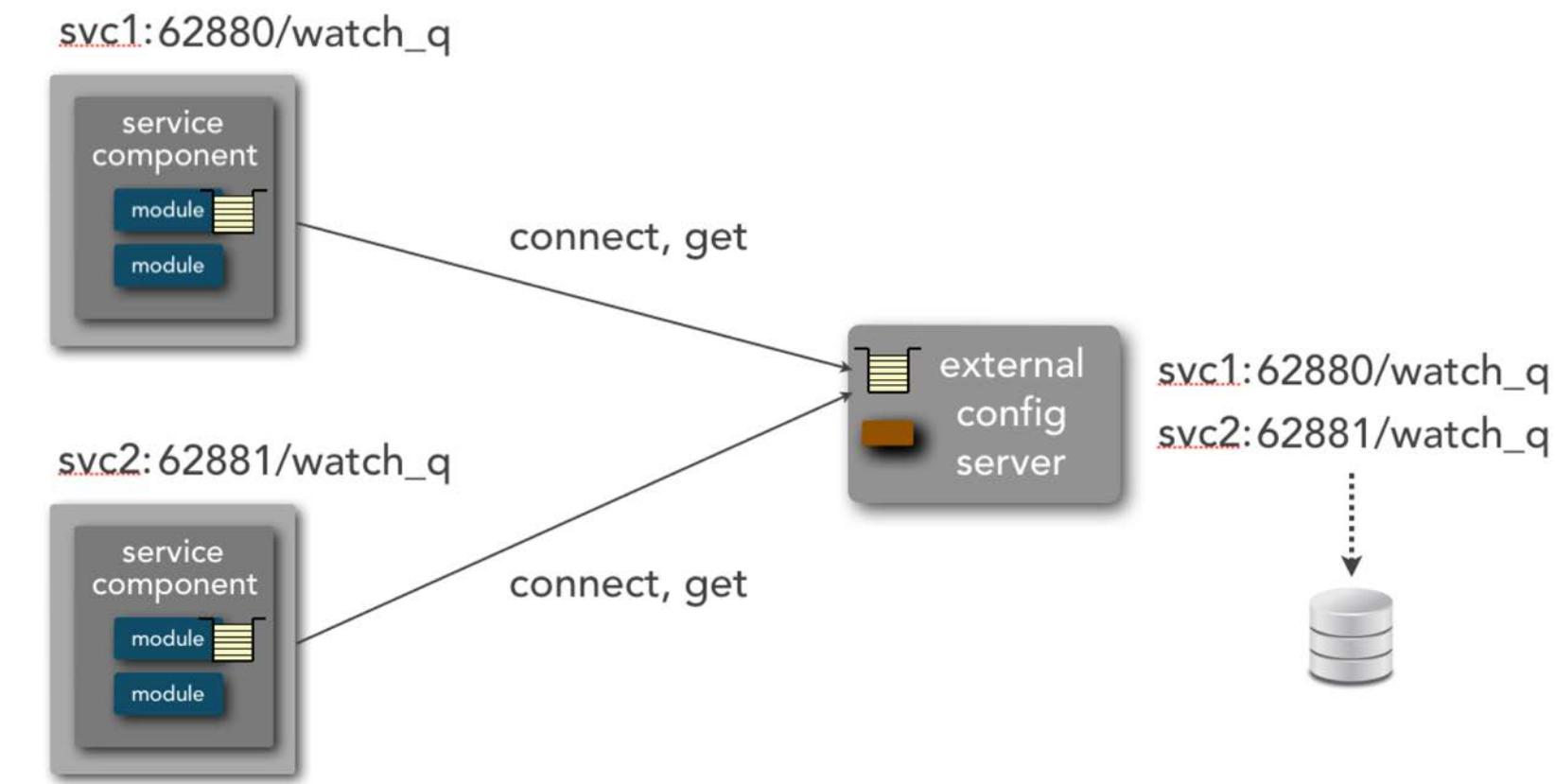
# watch notification pattern



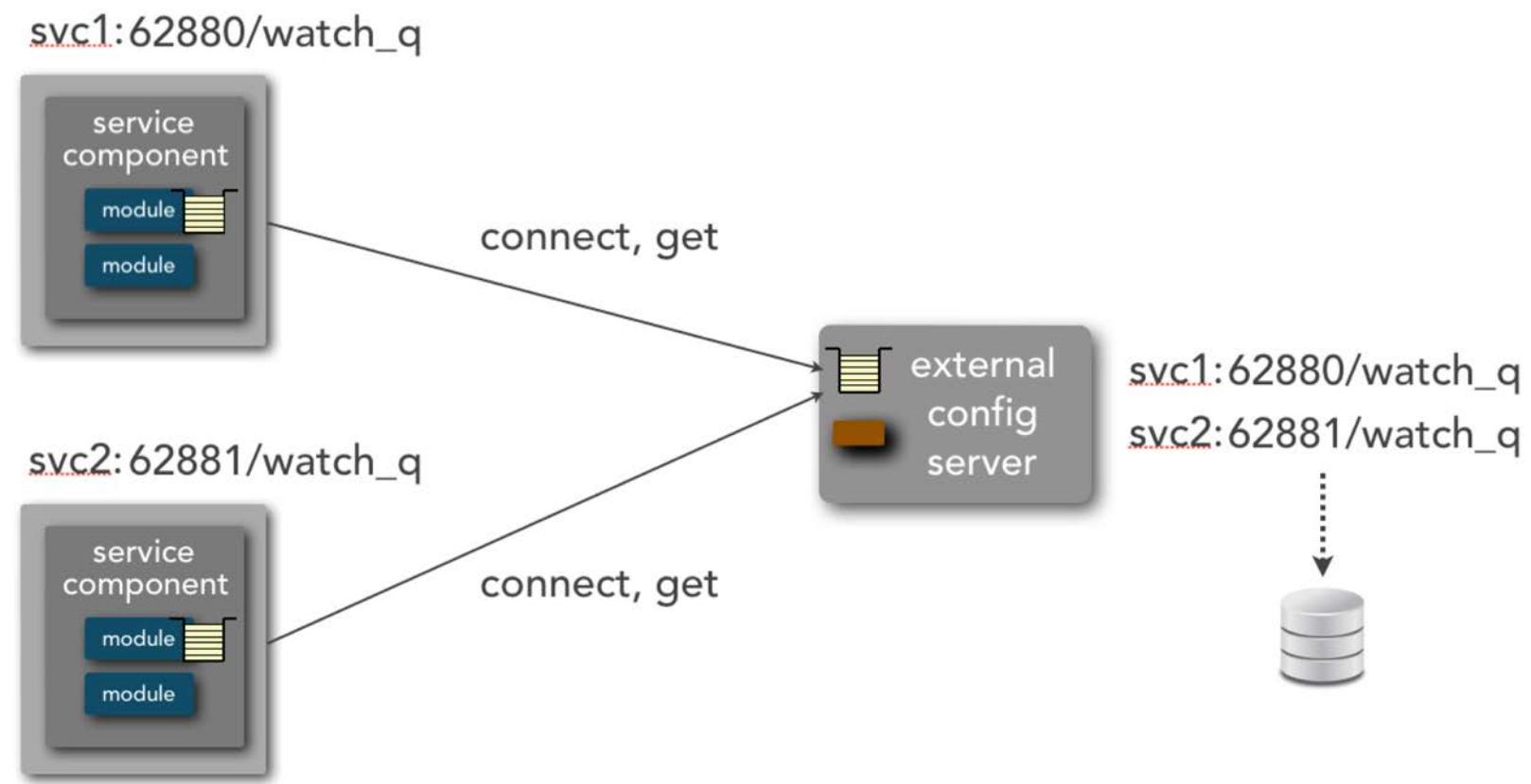
let's apply the pattern...

<https://github.com/wmr513/event-driven-patterns/tree/master/watch>

# watch notification pattern



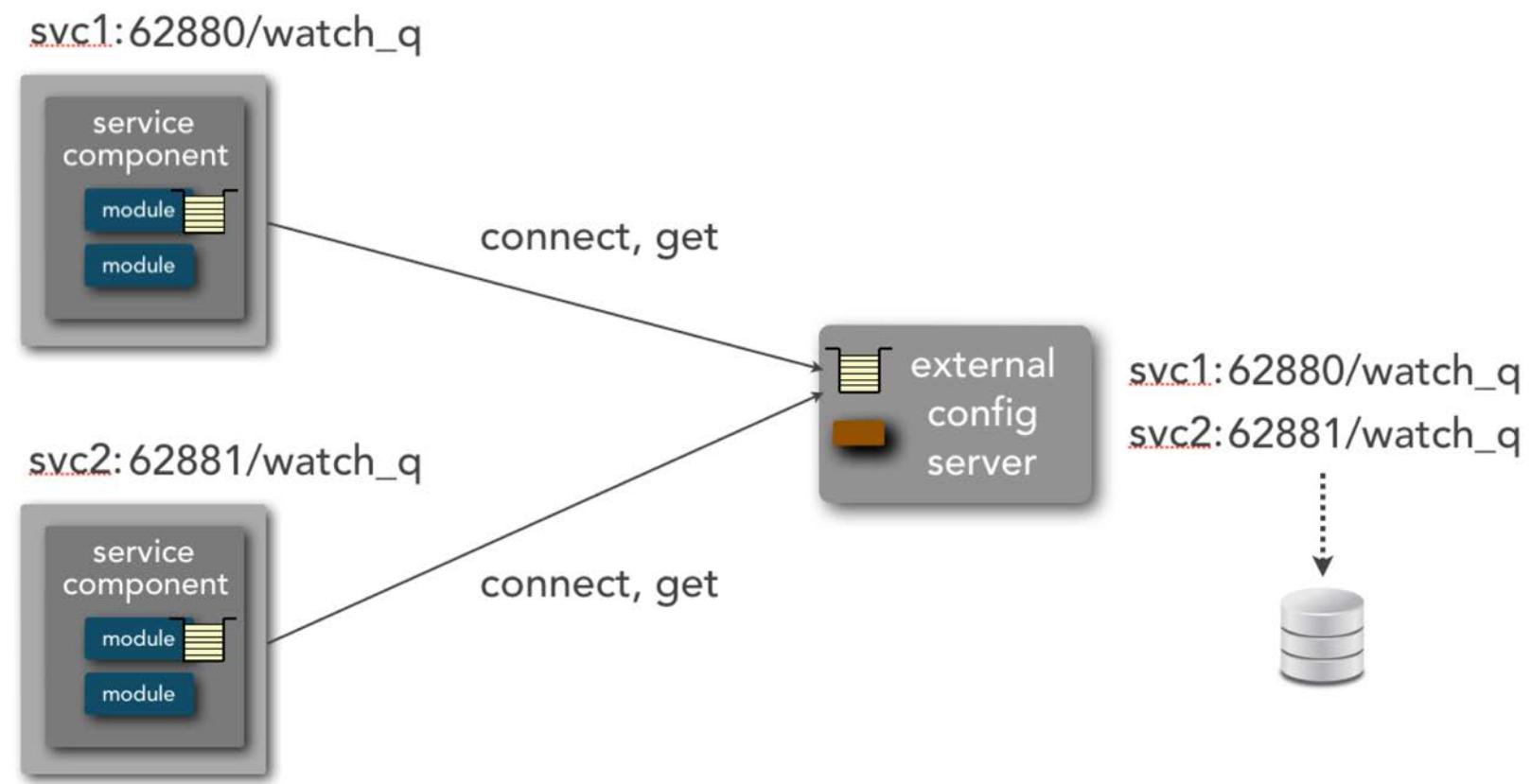
# watch notification pattern



lower bandwidth  
fewer connections  
fault tolerance



# watch notification pattern



lower bandwidth  
fewer connections  
fault tolerance



performance  
complexity  
error handing

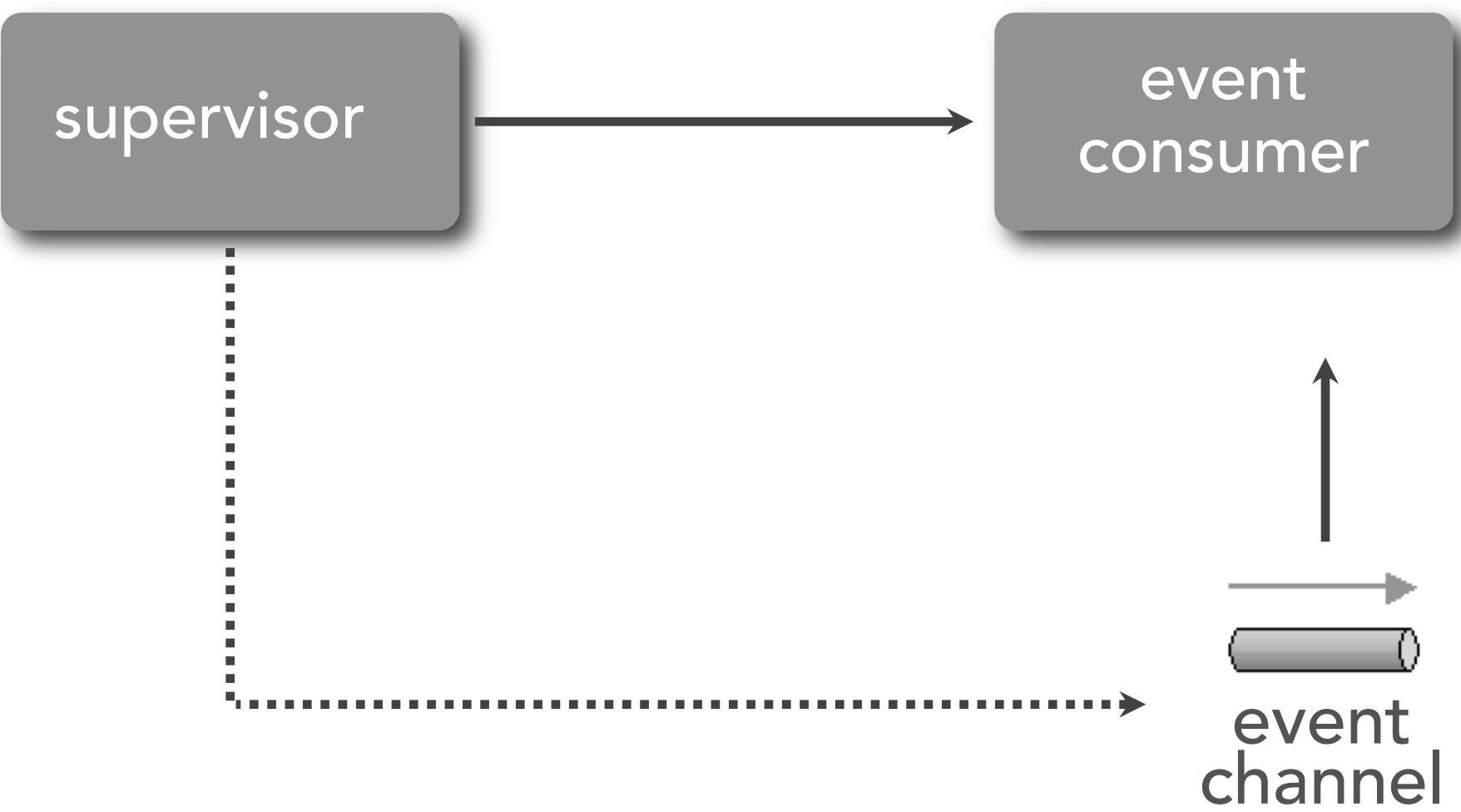
# Supervisor Consumer Pattern

# supervisor consumer pattern

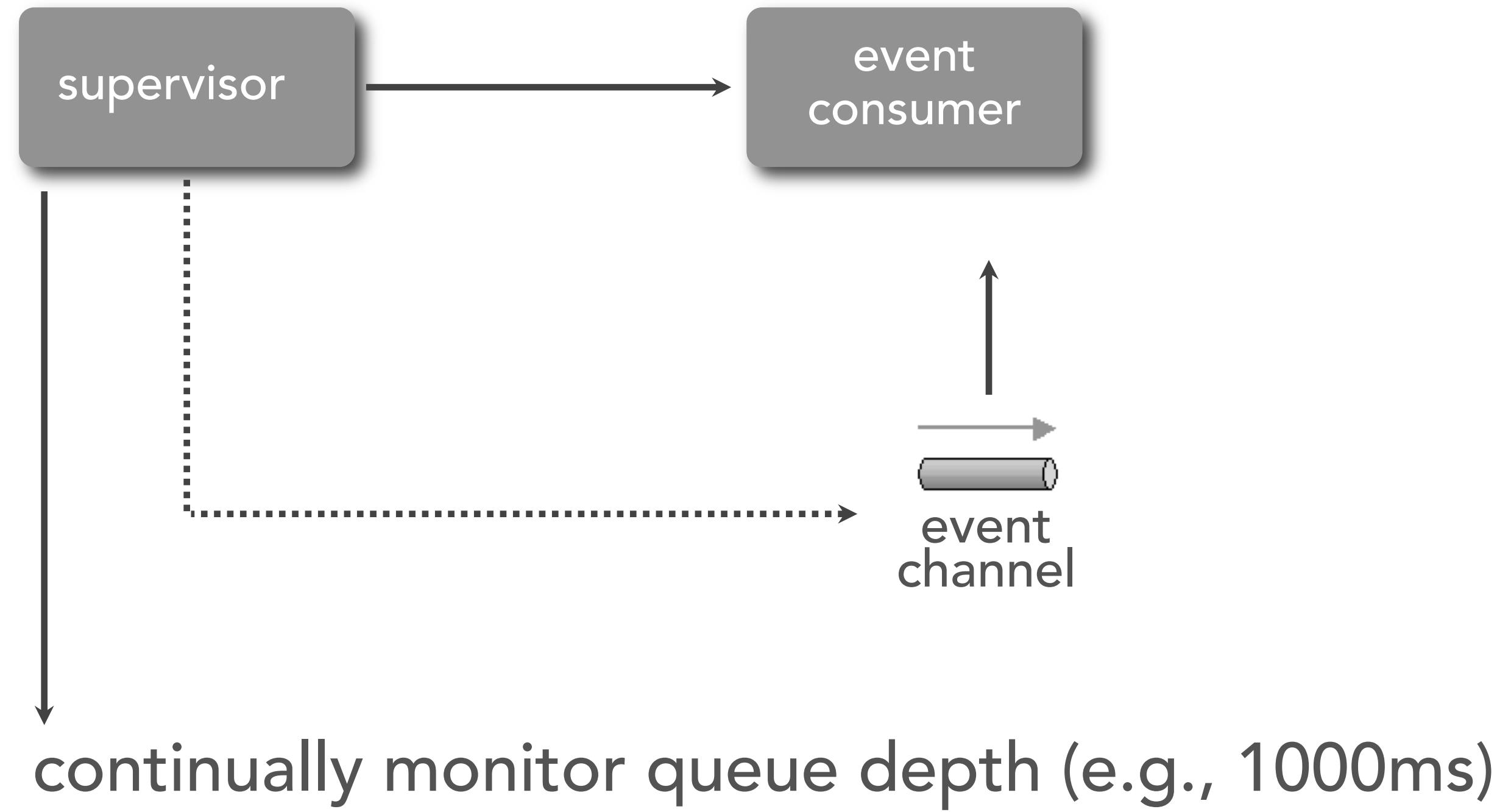
*“how do I handle varying request load and still maintain a consistent response time?”*



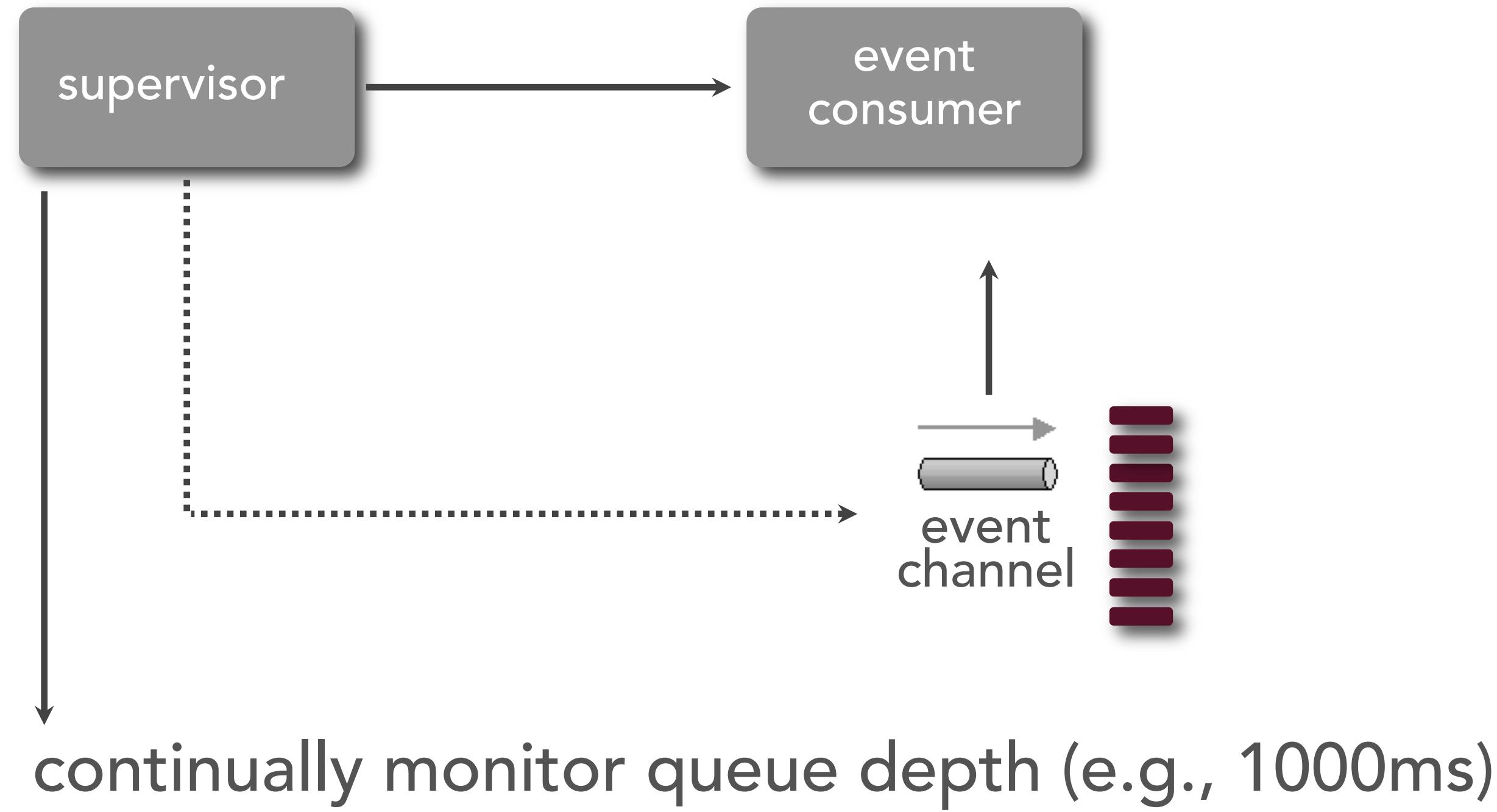
# supervisor consumer pattern



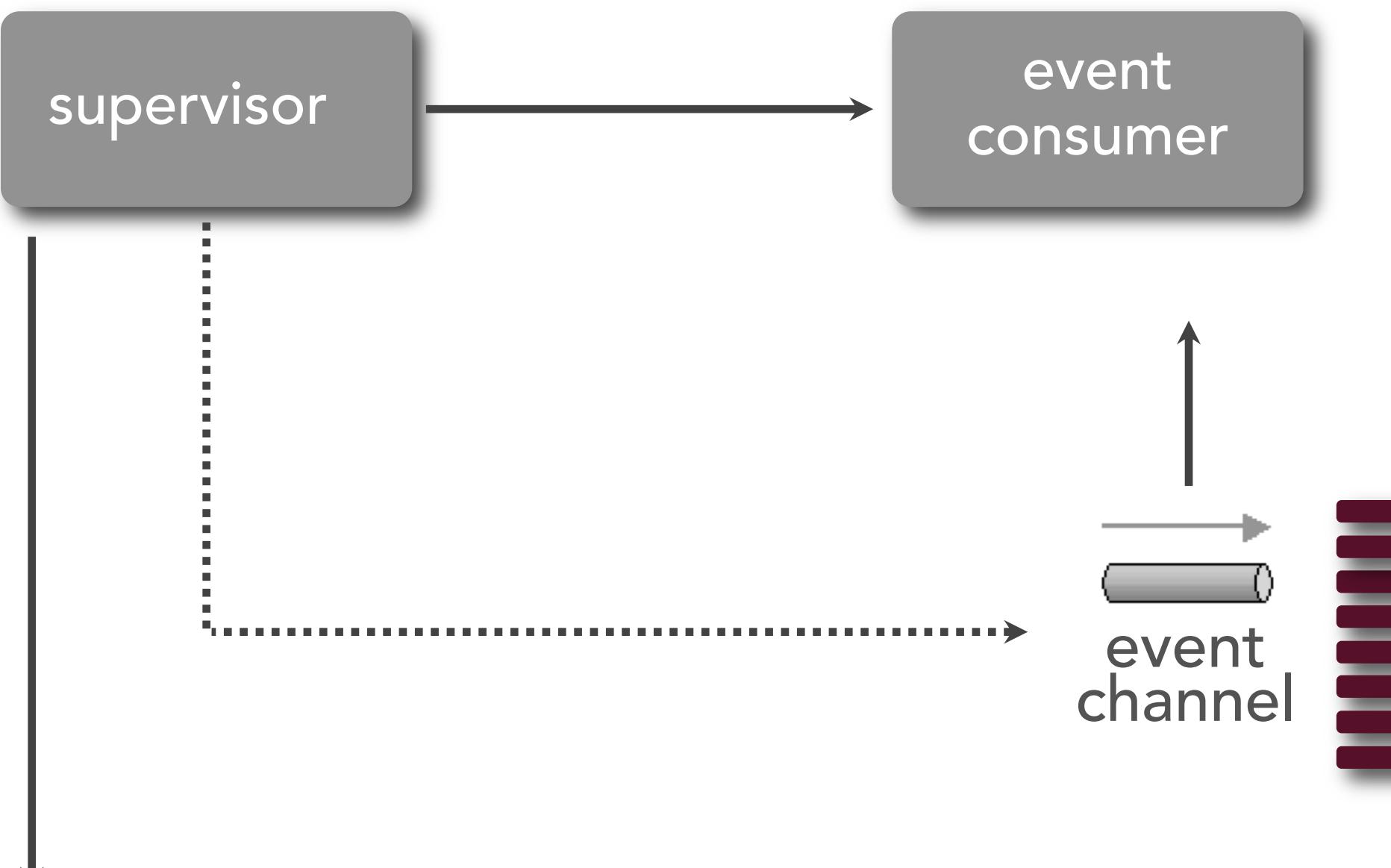
# supervisor consumer pattern



# supervisor consumer pattern

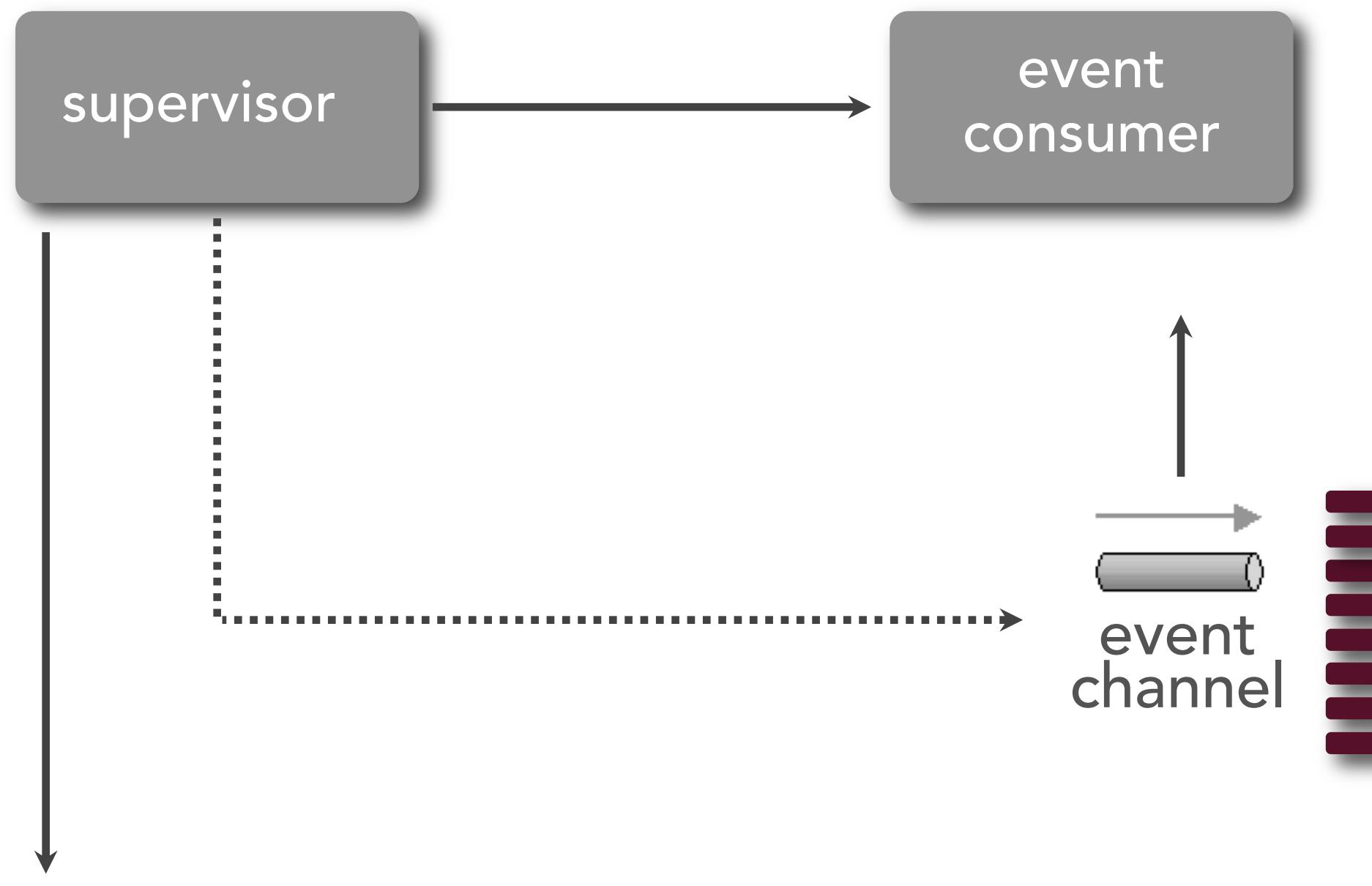


# supervisor consumer pattern



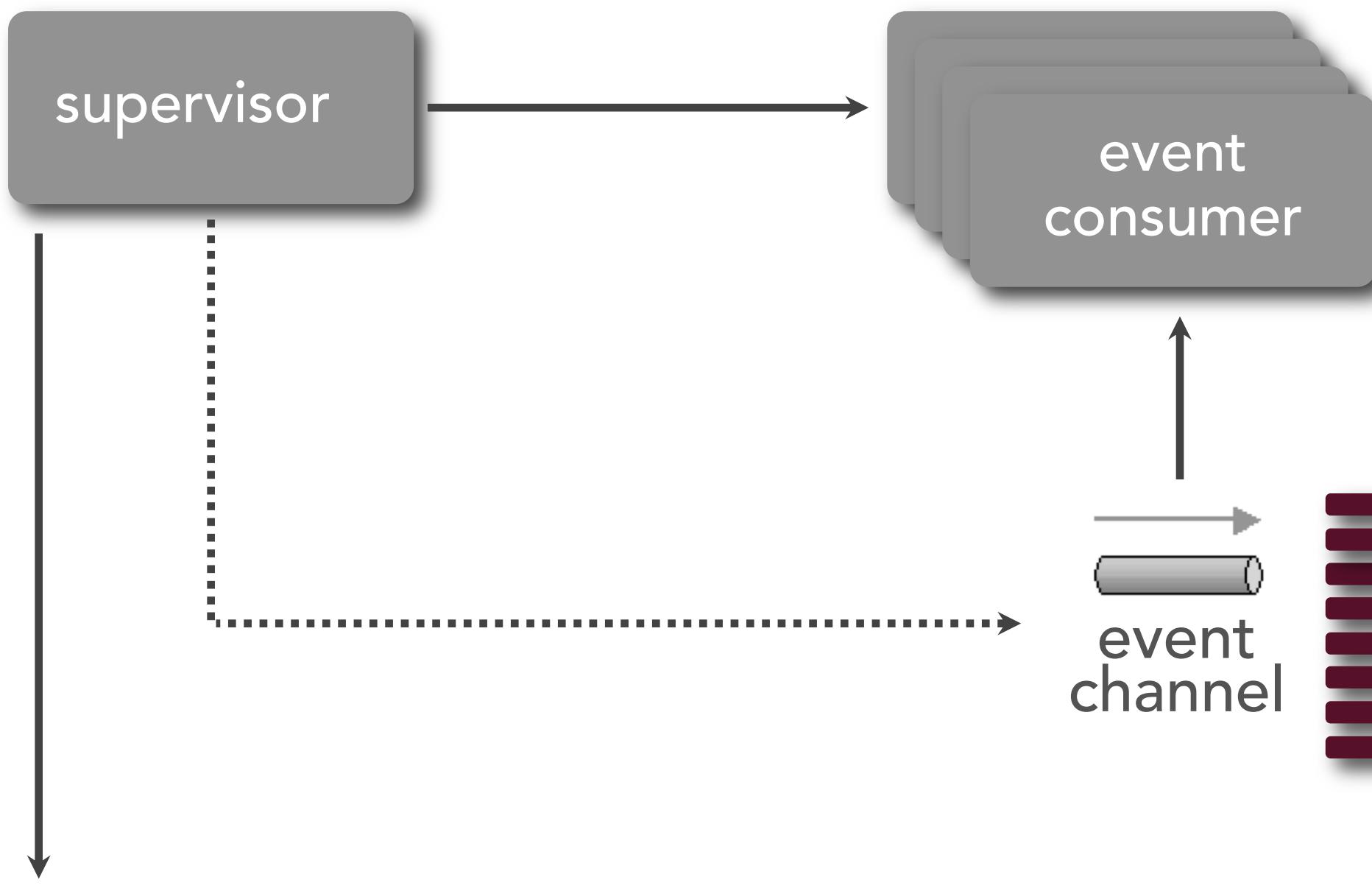
continually monitor queue depth (e.g., 1000ms)  
determine consumers needed ( $\text{depth}/n$ )

# supervisor consumer pattern



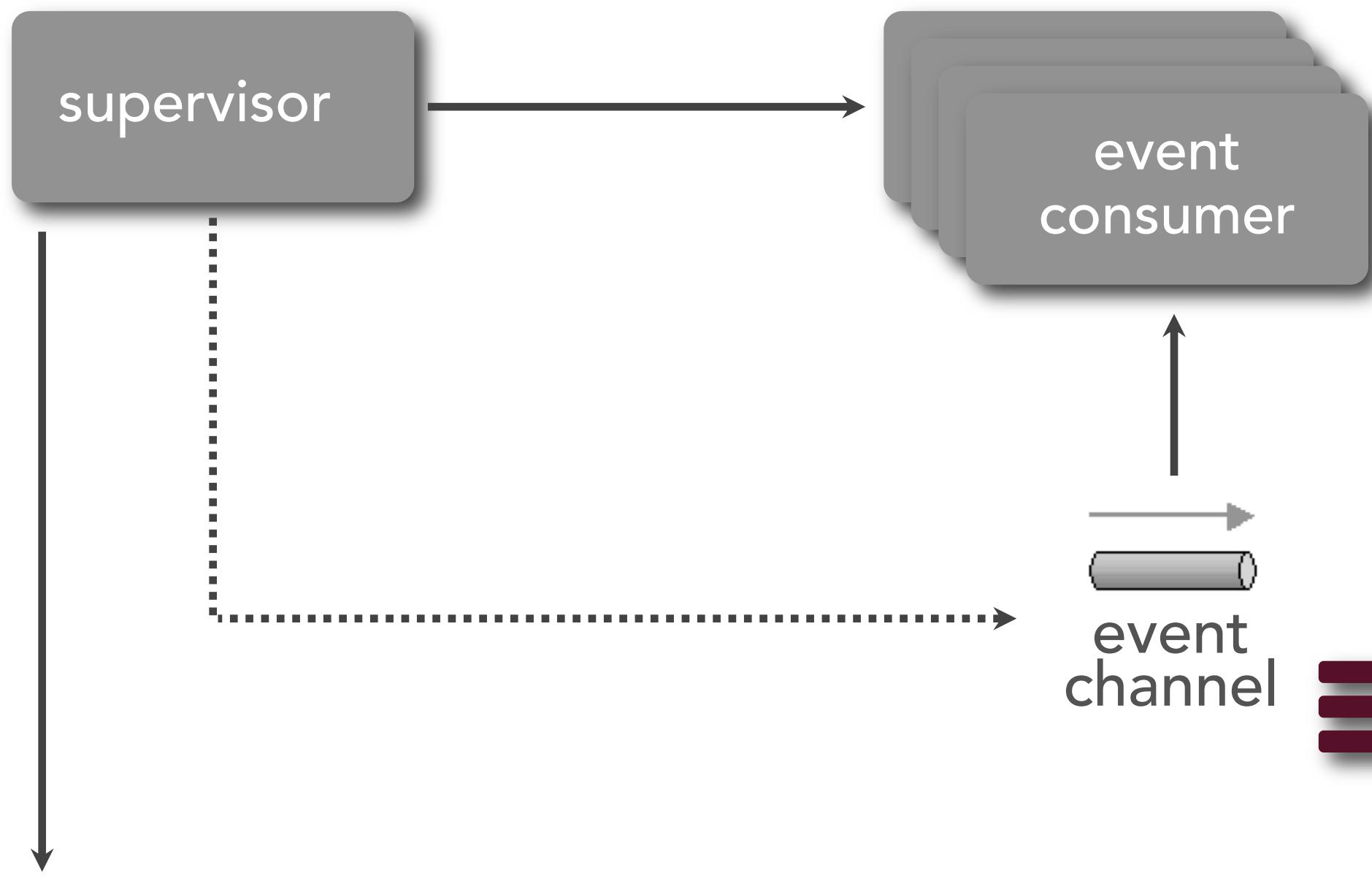
continually monitor queue depth (e.g., 1000ms)  
determine consumers needed ( $\text{depth}/n$ )  
apply max threshold (e.g., 500 consumers)

# supervisor consumer pattern



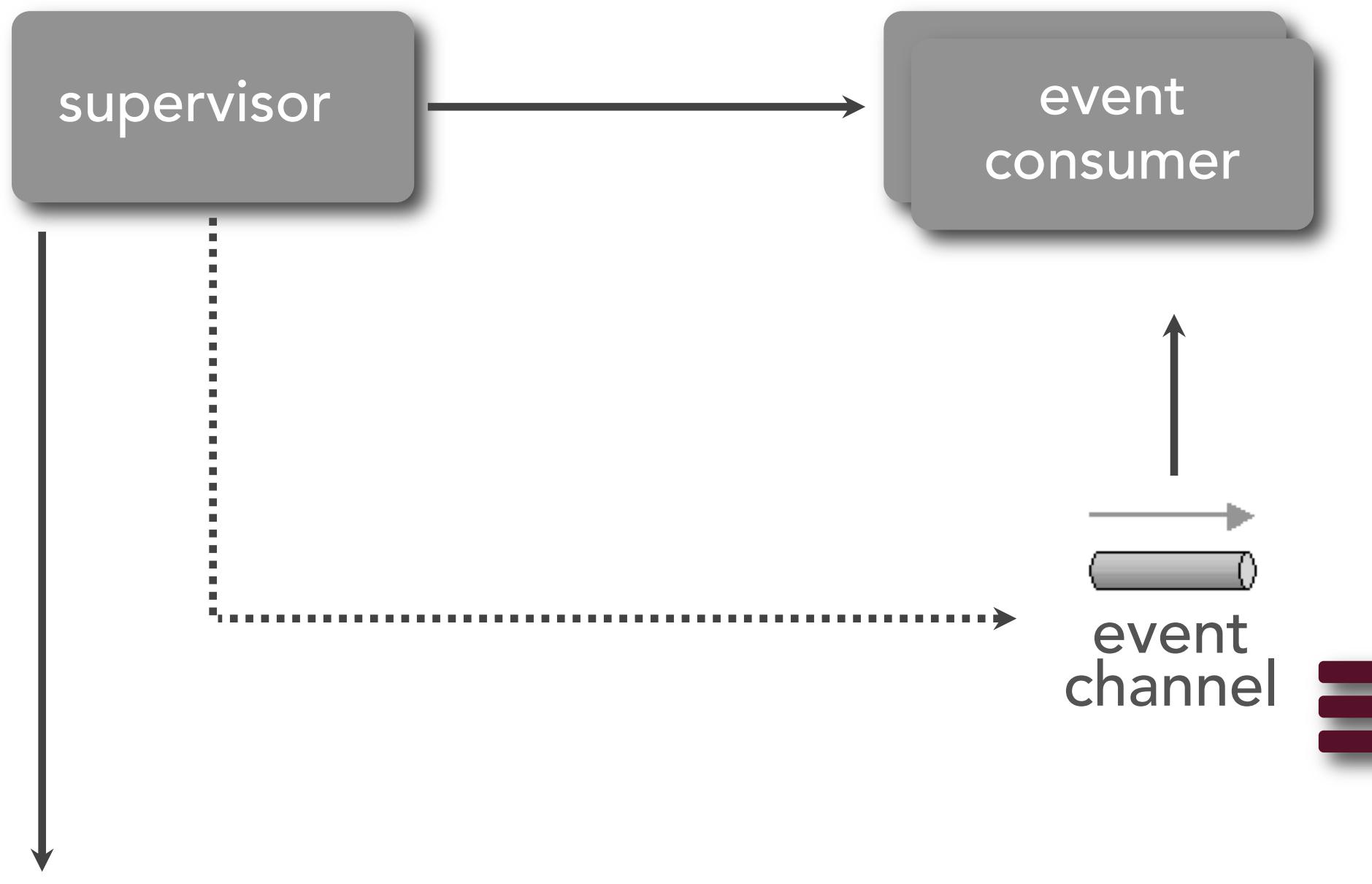
continually monitor queue depth (e.g., 1000ms)  
determine consumers needed ( $\text{depth}/n$ )  
apply max threshold (e.g., 500 consumers)  
add or remove consumers as needed

# supervisor consumer pattern



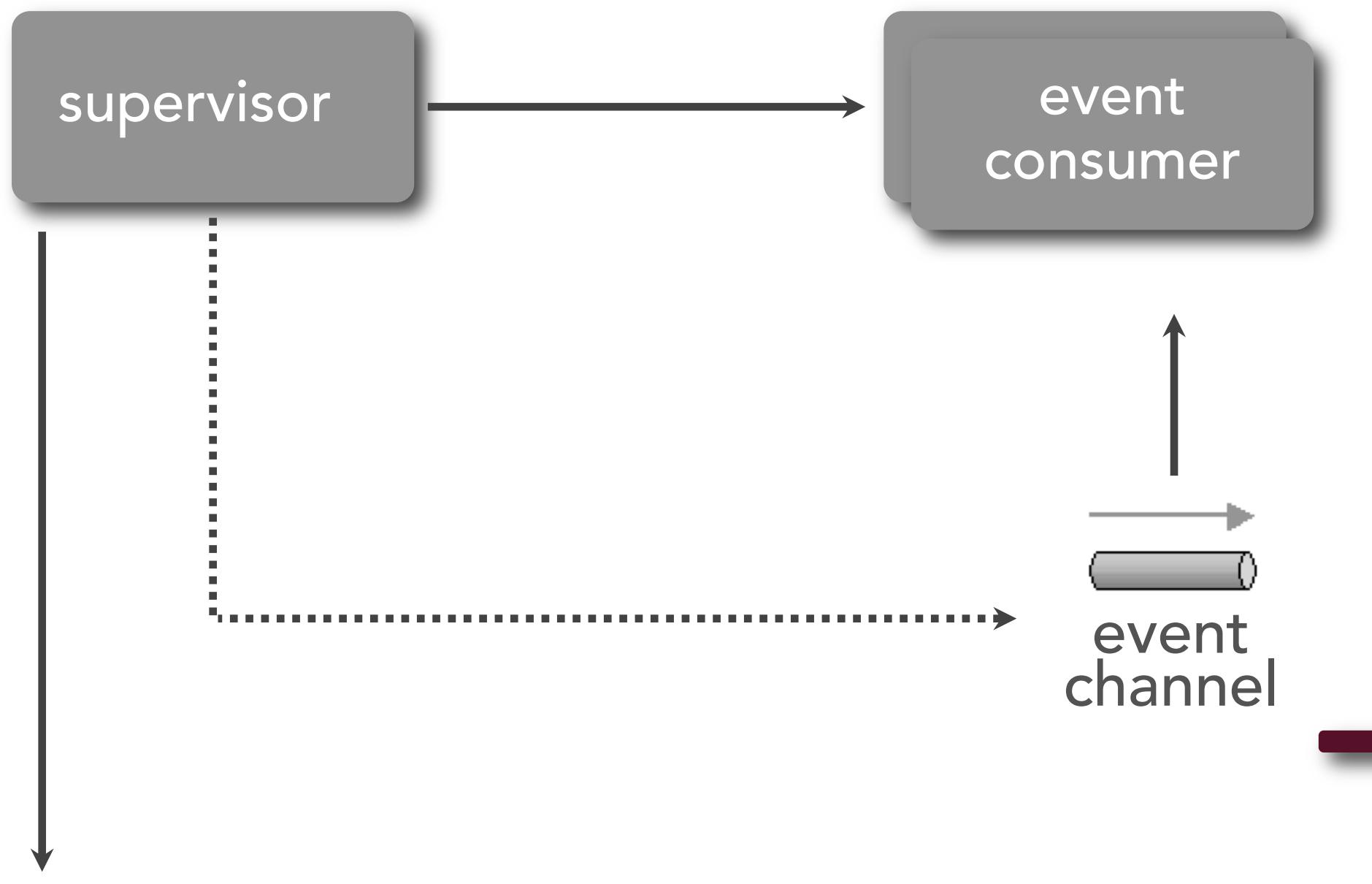
continually monitor queue depth (e.g., 1000ms)  
determine consumers needed ( $\text{depth}/n$ )  
apply max threshold (e.g., 500 consumers)  
add or remove consumers as needed

# supervisor consumer pattern



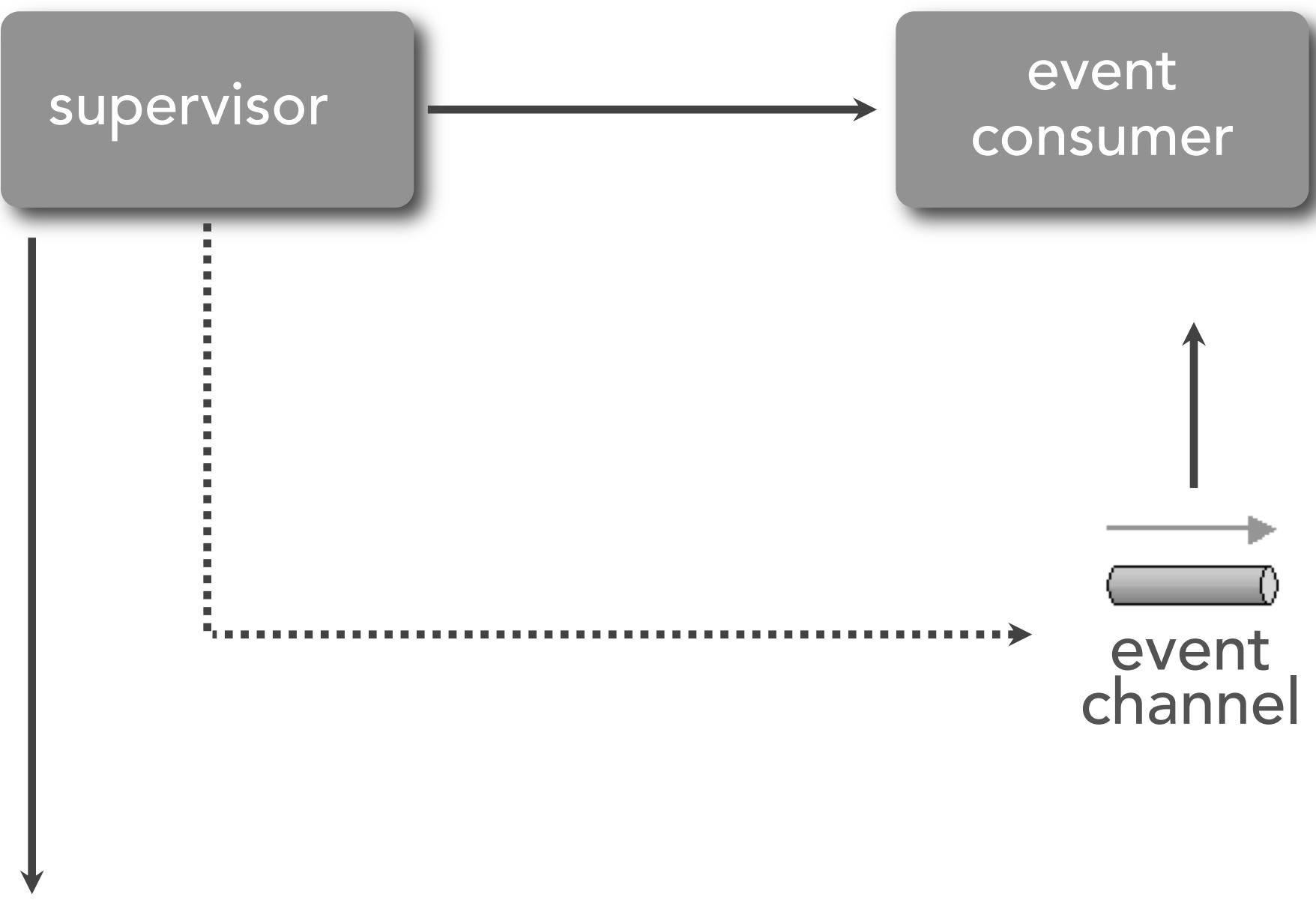
continually monitor queue depth (e.g., 1000ms)  
determine consumers needed ( $\text{depth}/n$ )  
apply max threshold (e.g., 500 consumers)  
add or remove consumers as needed

# supervisor consumer pattern



continually monitor queue depth (e.g., 1000ms)  
determine consumers needed ( $\text{depth}/n$ )  
apply max threshold (e.g., 500 consumers)  
add or remove consumers as needed

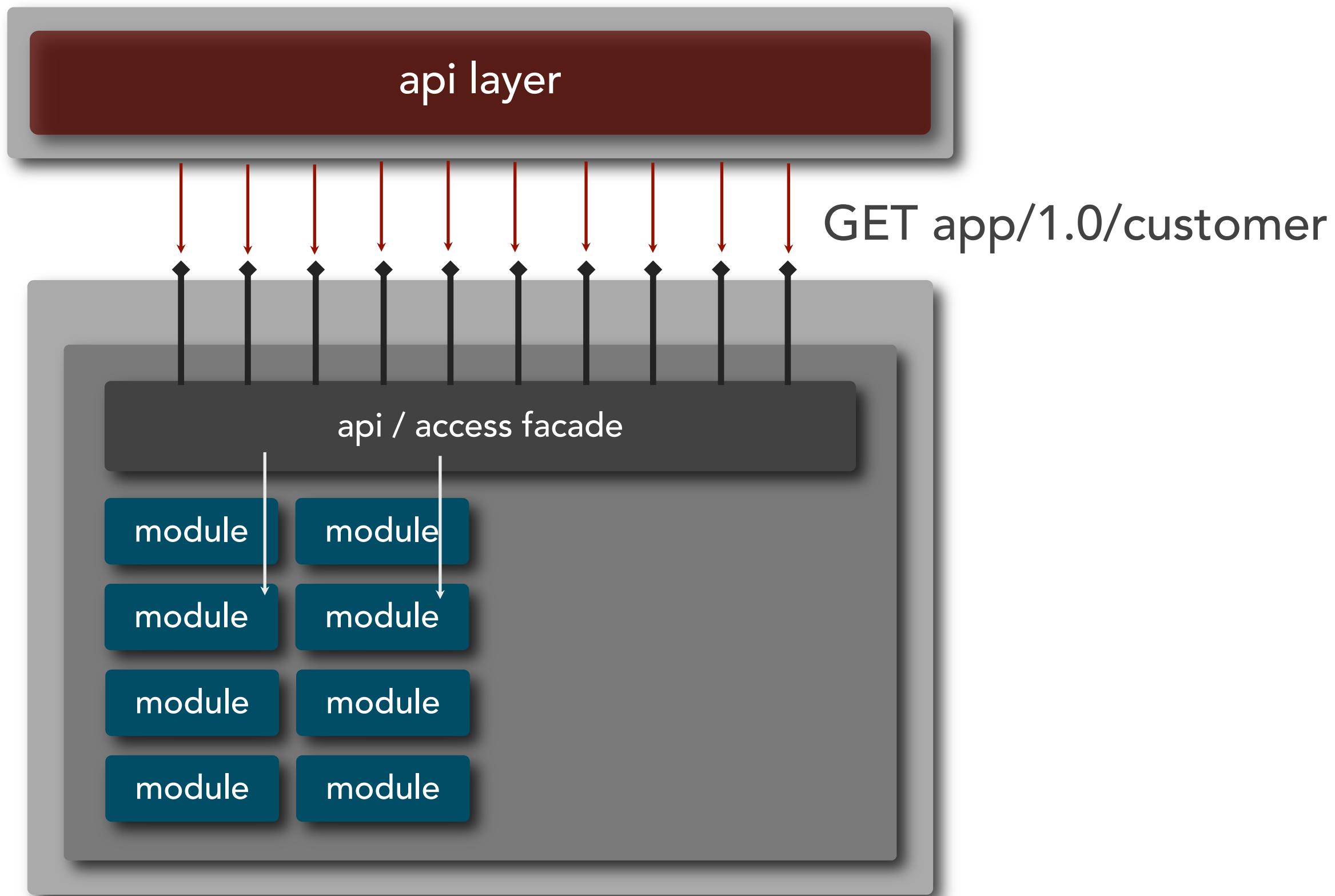
# supervisor consumer pattern



continually monitor queue depth (e.g., 1000ms)  
determine consumers needed ( $\text{depth}/n$ )  
apply max threshold (e.g., 500 consumers)  
add or remove consumers as needed

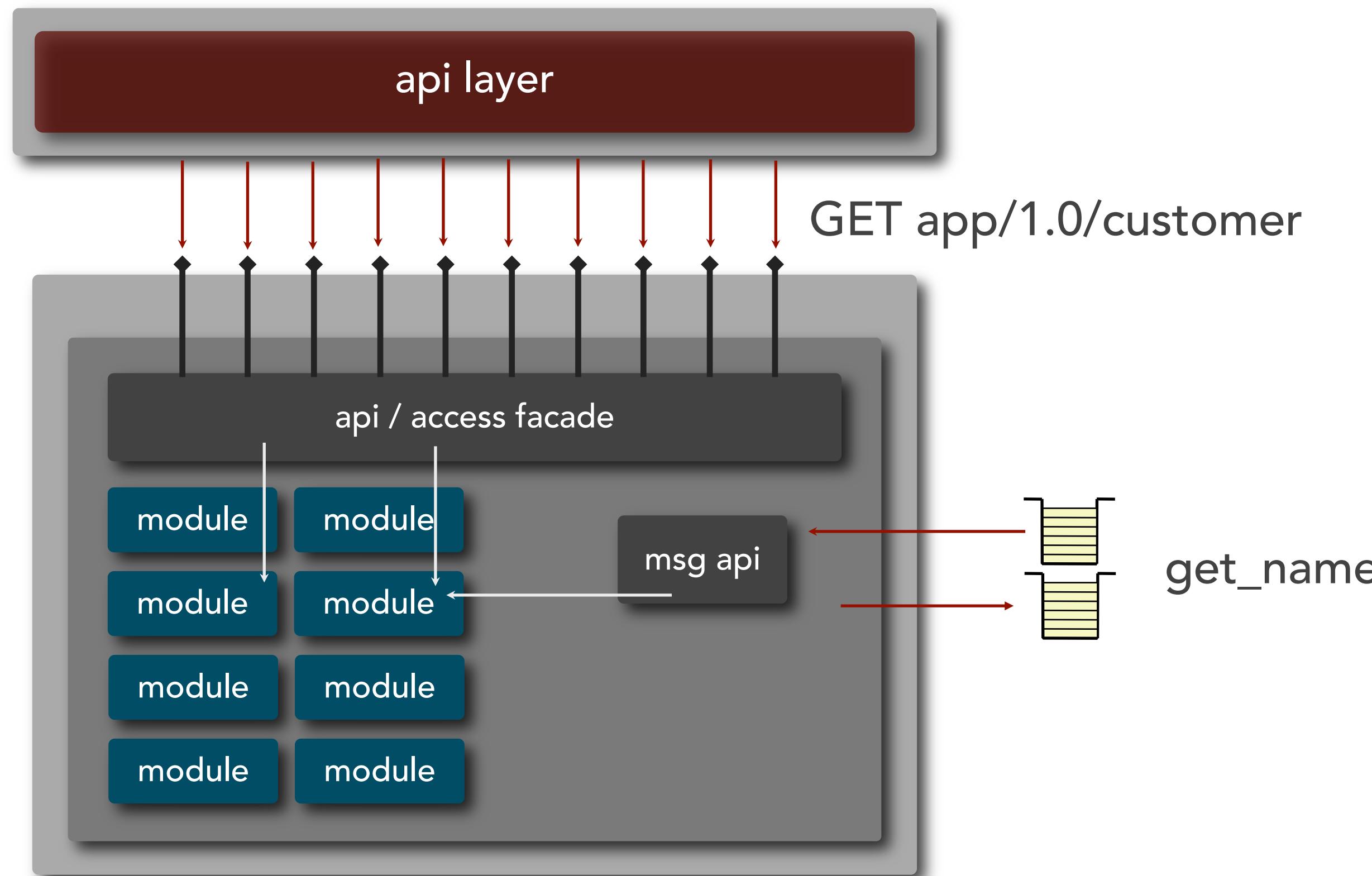
# supervisor consumer pattern

## inter-service communication



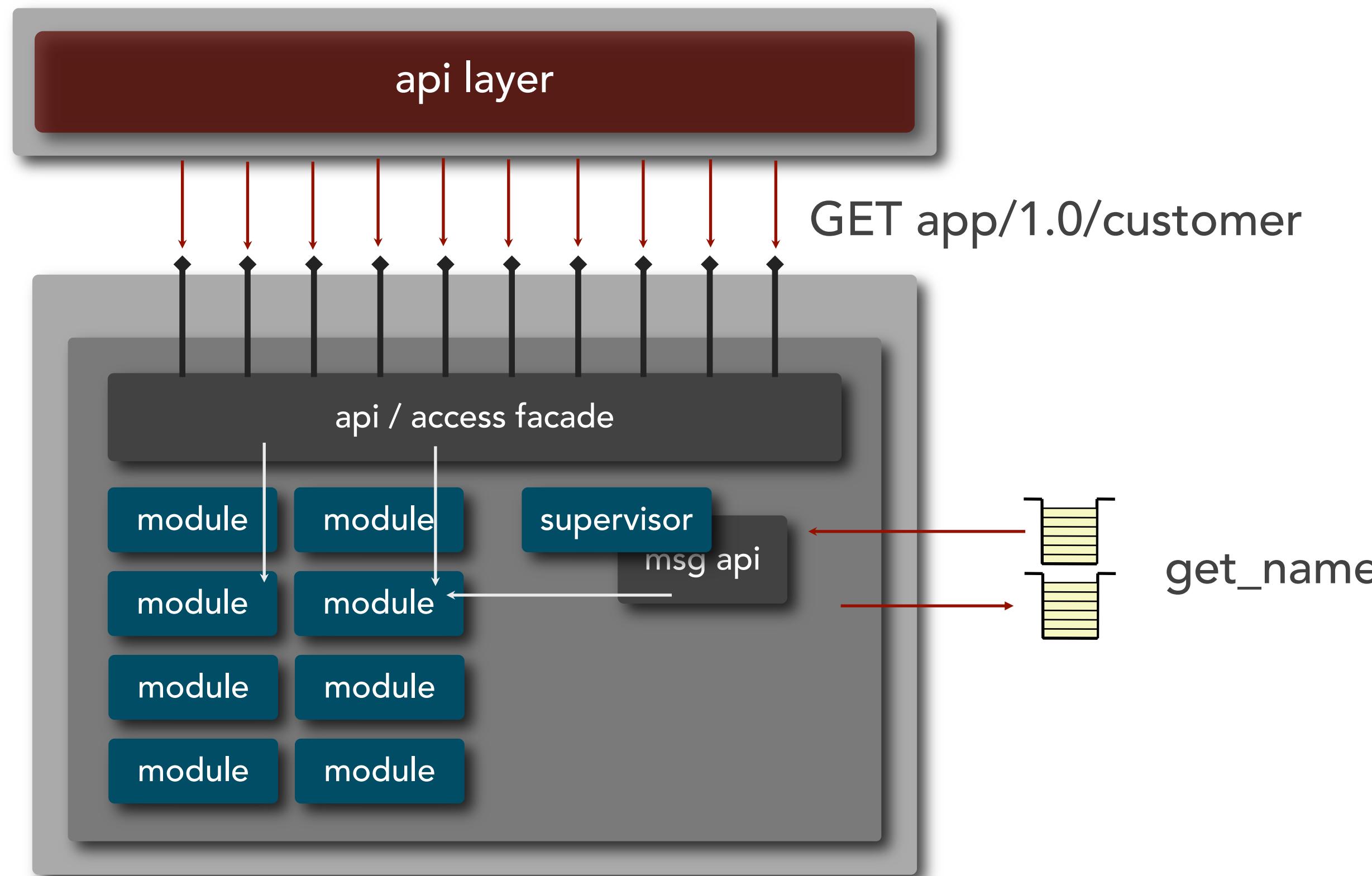
# supervisor consumer pattern

## inter-service communication



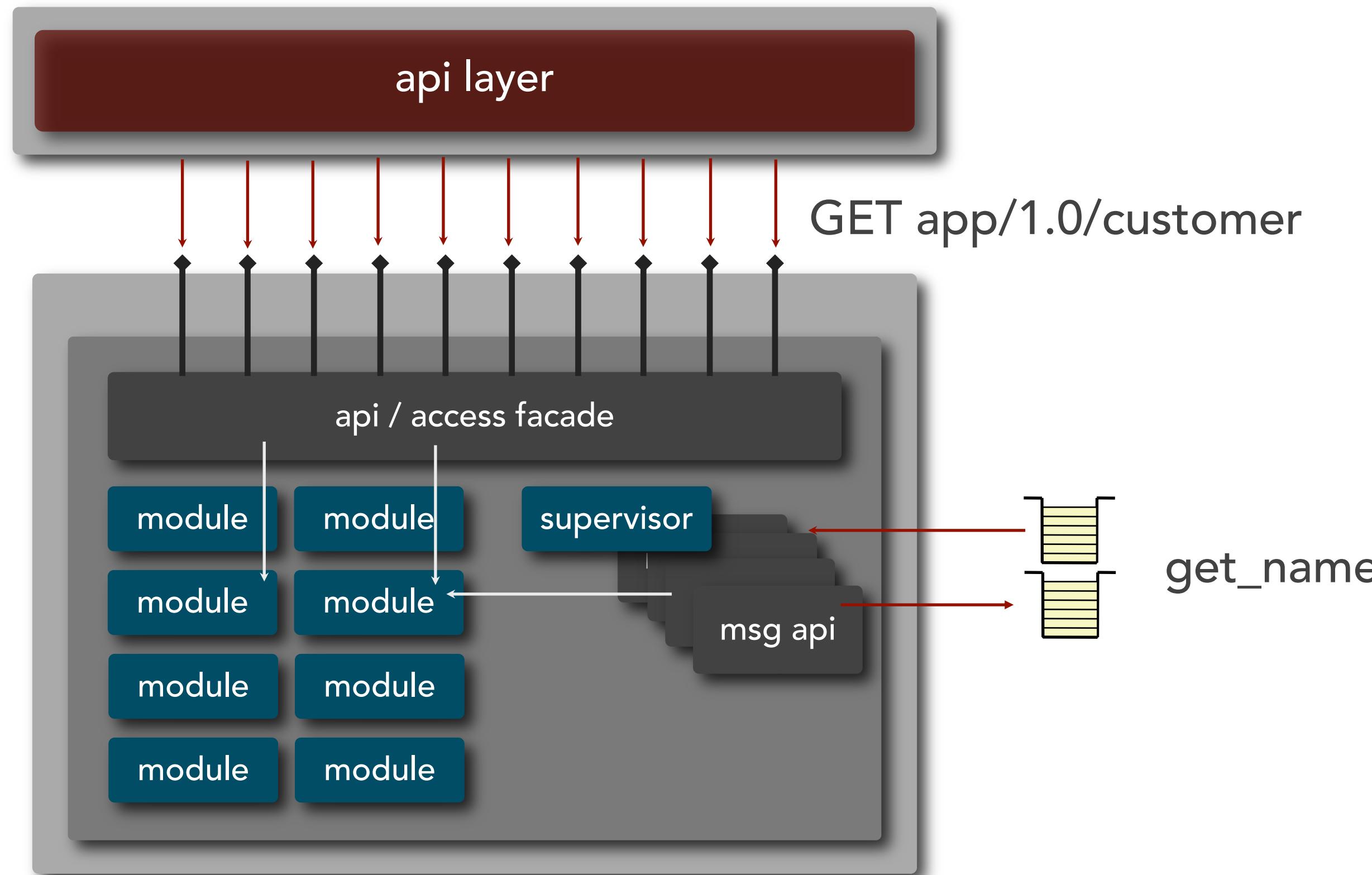
# supervisor consumer pattern

## inter-service communication



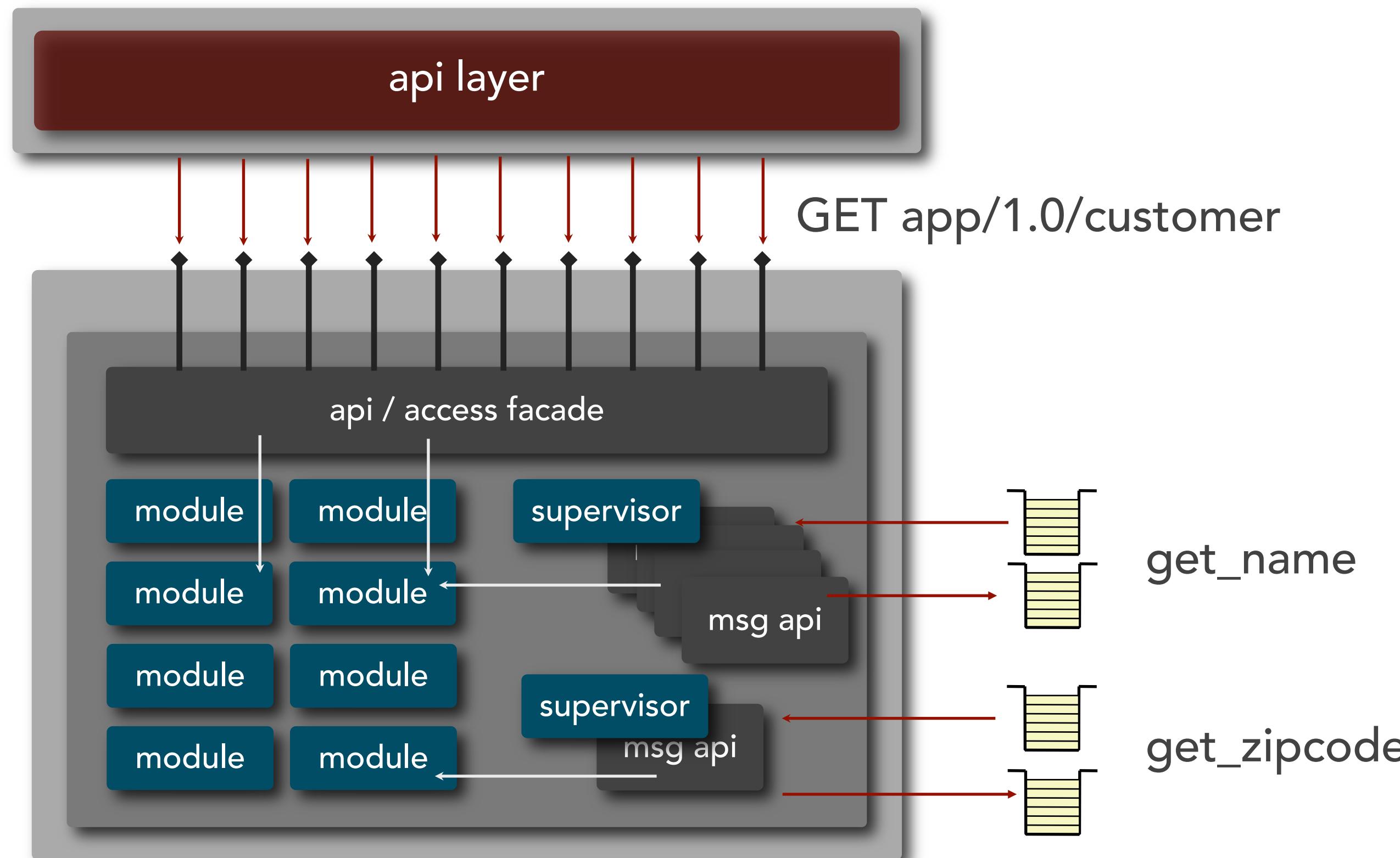
# supervisor consumer pattern

## inter-service communication



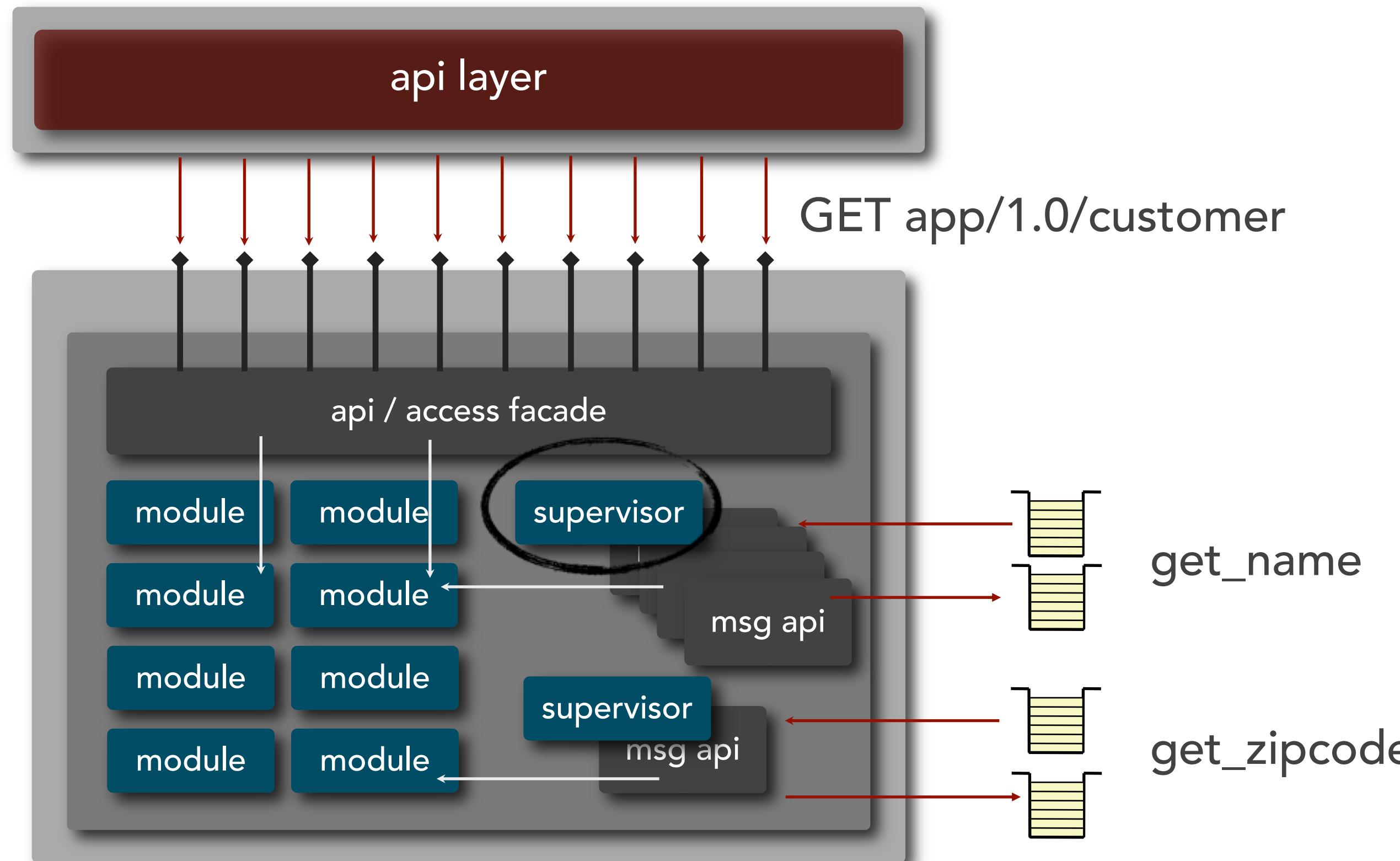
# supervisor consumer pattern

## inter-service communication



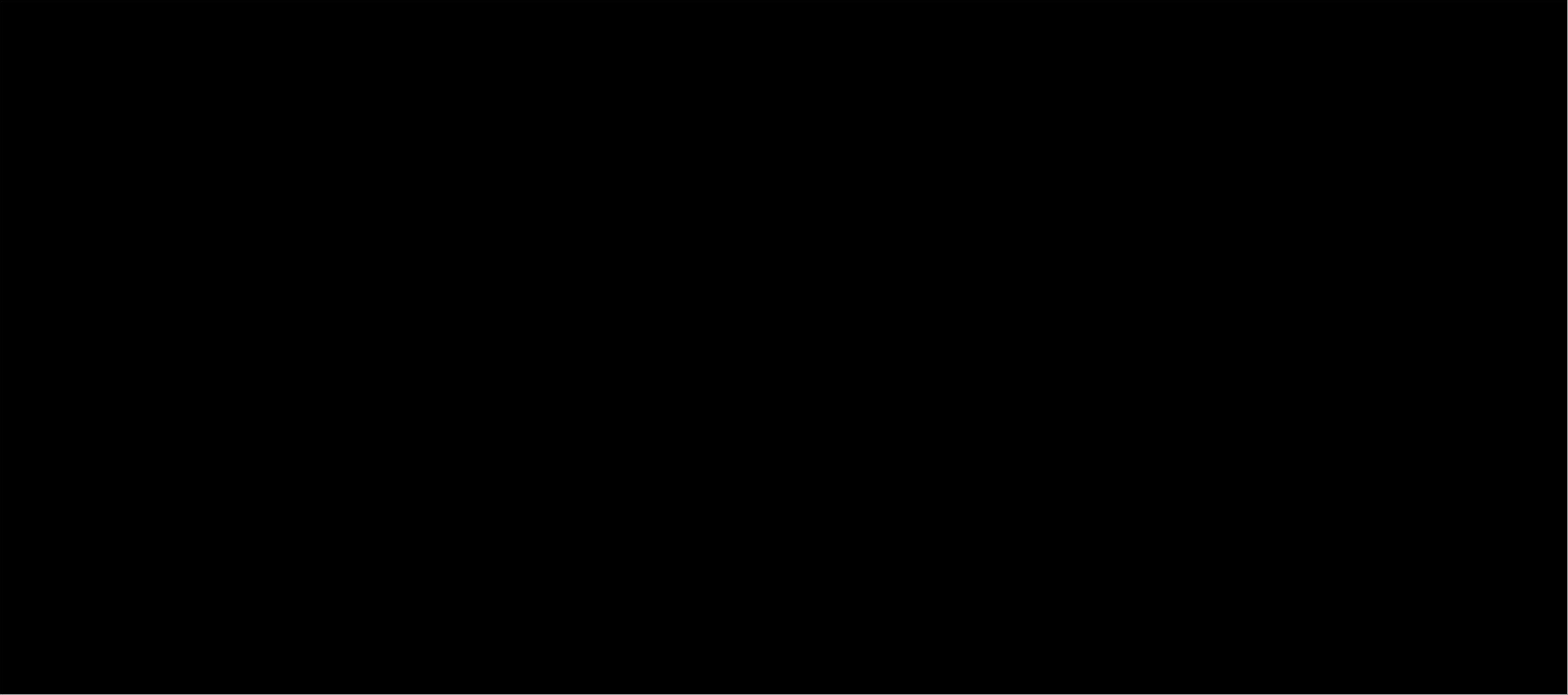
# supervisor consumer pattern

## inter-service communication



# supervisor consumer pattern

supervisor



# supervisor consumer pattern

supervisor

```
private List<NameAPI> consumers = new ArrayList<NameAPI>();
```

# supervisor consumer pattern

supervisor

```
private List<NameAPI> consumers = new ArrayList<NameAPI>();  
  
public void execute() throws Exception {  
    //connect to message broker  
    startConsumer(1);
```

# supervisor consumer pattern

supervisor

```
private List<NameAPI> consumers = new ArrayList<NameAPI>();  
  
public void execute() throws Exception {  
    //connect to message broker  
    startConsumer(1);  
    long inst = getInstanceCount(serviceName);
```

# supervisor consumer pattern

supervisor

```
private List<NameAPI> consumers = new ArrayList<NameAPI>();  
  
public void execute() throws Exception {  
    //connect to message broker  
    startConsumer(1);  
    long inst = getInstanceCount(serviceName);  
    while (true) {  
        long consumersNeeded = getMsgCount("get_name.q")/inst;
```

# supervisor consumer pattern

## supervisor

```
private List<NameAPI> consumers = new ArrayList<NameAPI>();  
  
public void execute() throws Exception {  
    //connect to message broker  
    startConsumer(1);  
    long inst = getInstanceCount(serviceName);  
    while (true) {  
        long consumersNeeded = getMsgCount("get_name.q")/inst;  
        if (consumersNeeded > 2000) consumersNeeded = 2000;
```

# supervisor consumer pattern

## supervisor

```
private List<NameAPI> consumers = new ArrayList<NameAPI>();

public void execute() throws Exception {
    //connect to message broker
    startConsumer(1);
    long inst = getInstanceCount(serviceName);
    while (true) {
        long consumersNeeded = getMsgCount("get_name.q")/inst;
        if (consumersNeeded > 2000) consumersNeeded = 2000;
        long diff = Math.abs(consumersNeeded - consumers.size());
```

# supervisor consumer pattern

## supervisor

```
private List<NameAPI> consumers = new ArrayList<NameAPI>();

public void execute() throws Exception {
    //connect to message broker
    startConsumer(1);
    long inst = getInstanceCount(serviceName);
    while (true) {
        long consumersNeeded = getMsgCount("get_name.q")/inst;
        if (consumersNeeded > 2000) consumersNeeded = 2000;
        long diff = Math.abs(consumersNeeded - consumers.size());
        if (consumersNeeded > consumers.size()) { startConsumer(diff);
```

# supervisor consumer pattern

## supervisor

```
private List<NameAPI> consumers = new ArrayList<NameAPI>();

public void execute() throws Exception {
    //connect to message broker
    startConsumer(1);
    long inst = getInstanceCount(serviceName);
    while (true) {
        long consumersNeeded = getMsgCount("get_name.q")/inst;
        if (consumersNeeded > 2000) consumersNeeded = 2000;
        long diff = Math.abs(consumersNeeded - consumers.size());
        if (consumersNeeded > consumers.size()) { startConsumer(diff);
        else { stopConsumer(diff); }}
```

# supervisor consumer pattern

supervisor

```
private List<NameAPI> consumers = new ArrayList<NameAPI>();

public void execute() throws Exception {
    //connect to message broker
    startConsumer(1);
    long inst = getInstanceCount(serviceName);
    while (true) {
        long consumersNeeded = getMsgCount("get_name.q")/inst;
        if (consumersNeeded > 2000) consumersNeeded = 2000;
        long diff = Math.abs(consumersNeeded - consumers.size());
        if (consumersNeeded > consumers.size()) { startConsumer(diff);
        else { stopConsumer(diff); }
        Thread.sleep(1000);
    }
}
```

# supervisor consumer pattern

supervisor

```
private List<NameAPI> consumers = new ArrayList<NameAPI>();

public void execute() throws Exception {
    //connect to message broker
    startConsumer(1);
    long inst = getInstanceCount(serviceName);
    while (true) {
        long consumersNeeded = getMsgCount("get_name.q")/inst;
        if (consumersNeeded > 2000) consumersNeeded = 2000;
        long diff = Math.abs(consumersNeeded - consumers.size());
        if (consumersNeeded > consumers.size()) { startConsumer(diff);
        else { stopConsumer(diff); }
        Thread.sleep(1000);
    }}
```

# supervisor consumer pattern

supervisor

```
private long getMessageCount(String queueName)
    throws Exception {
    return channel.messageCount(queueName);
}
```

# supervisor consumer pattern

## supervisor

```
private long getMessageCount(String queueName) throws Exception {  
    Queue queue = session.createQueue("trade.eq.q");  
    QueueBrowser browser = session.createBrowser(queue);  
    int msgCount =  
        Collections.list(browser.getEnumeration()).size();  
    browser.close();  
    return msgCount;  
}
```

# supervisor consumer pattern

supervisor

```
private List<NameAPI> consumers = new ArrayList<NameAPI>();

public void execute() throws Exception {
    //connect to message broker
    startConsumer(1);
    long inst = getInstanceCount(serviceName);
    while (true) {
        long consumersNeeded = getMsgCount("get_name.q")/inst;
        if (consumersNeeded > 2000) consumersNeeded = 2000;
        long diff = Math.abs(consumersNeeded - consumers.size());
        if (consumersNeeded > consumers.size()) { startConsumer(diff);
        else { stopConsumer(diff); }
        Thread.sleep(1000);
    }
}
```

# supervisor consumer pattern

supervisor

```
private void startConsumer(long count) {  
    for (long i=0;i<count;i++) {  
        NameAPI consumer = new NameAPI();  
        consumers.add(consumer);  
        new Thread(()->consumer.startup(connection)).start();  
    }  
}
```

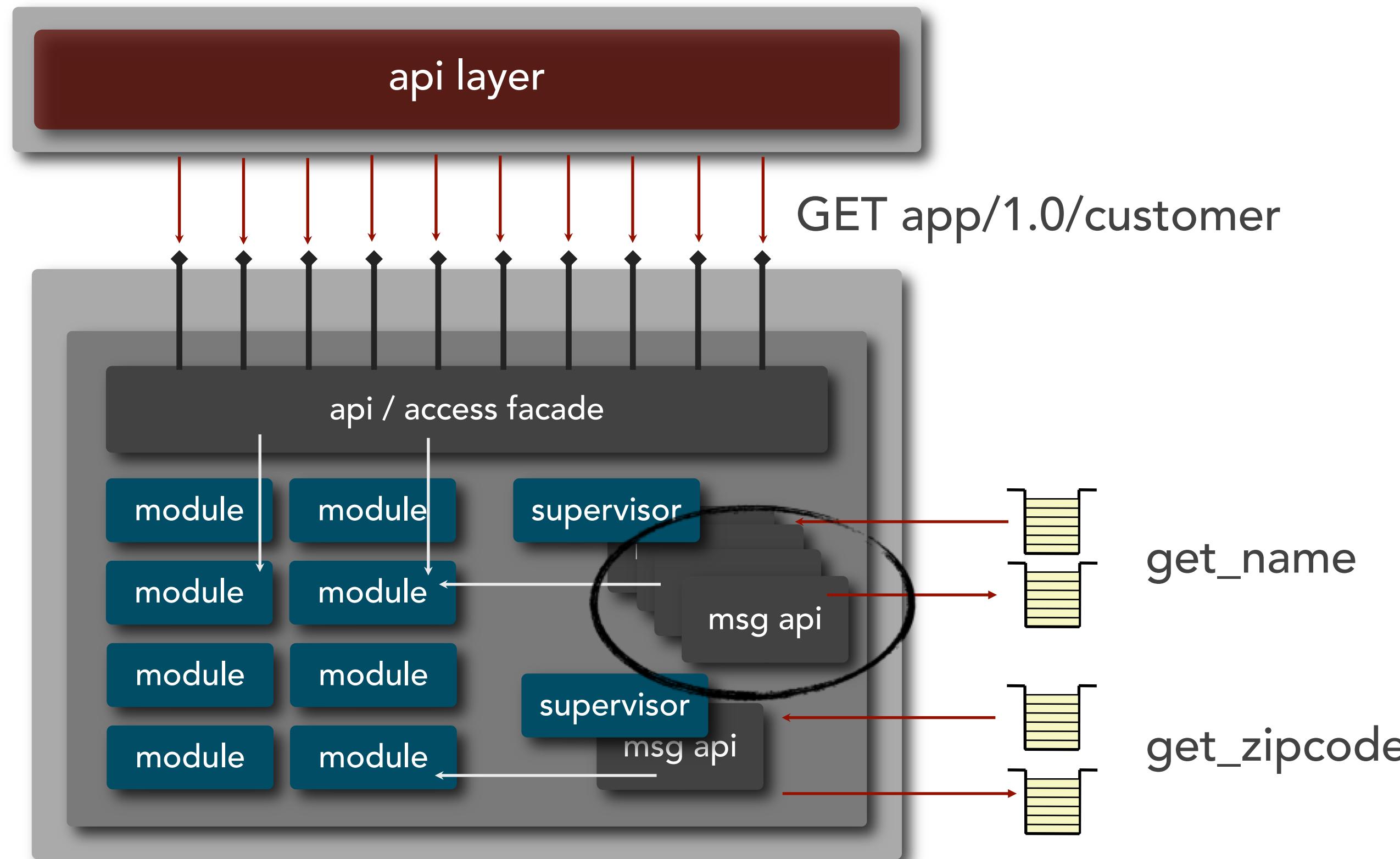
# supervisor consumer pattern

supervisor

```
private void stopConsumer(long count) {  
    for (long i=0;i<count;i++) {  
        if (consumers.size() > 1) {  
            consumers.get(0).shutdown();  
            consumers.remove(consumer);  
        }  
    }  
}
```

# supervisor consumer pattern

## inter-service communication



# consumer supervisor pattern

event consumer



# consumer supervisor pattern

event consumer

```
private Boolean active = true;
```

# consumer supervisor pattern

## event consumer

```
private Boolean active = true;  
  
public void startup(Connection connection) {  
    //get broker session or channel and create consumer  
    while (active) {  
        msg = getNextMessageFromQueue(1000);  
        if (msg != null) //process message  
    }  
}
```

# consumer supervisor pattern

## event consumer

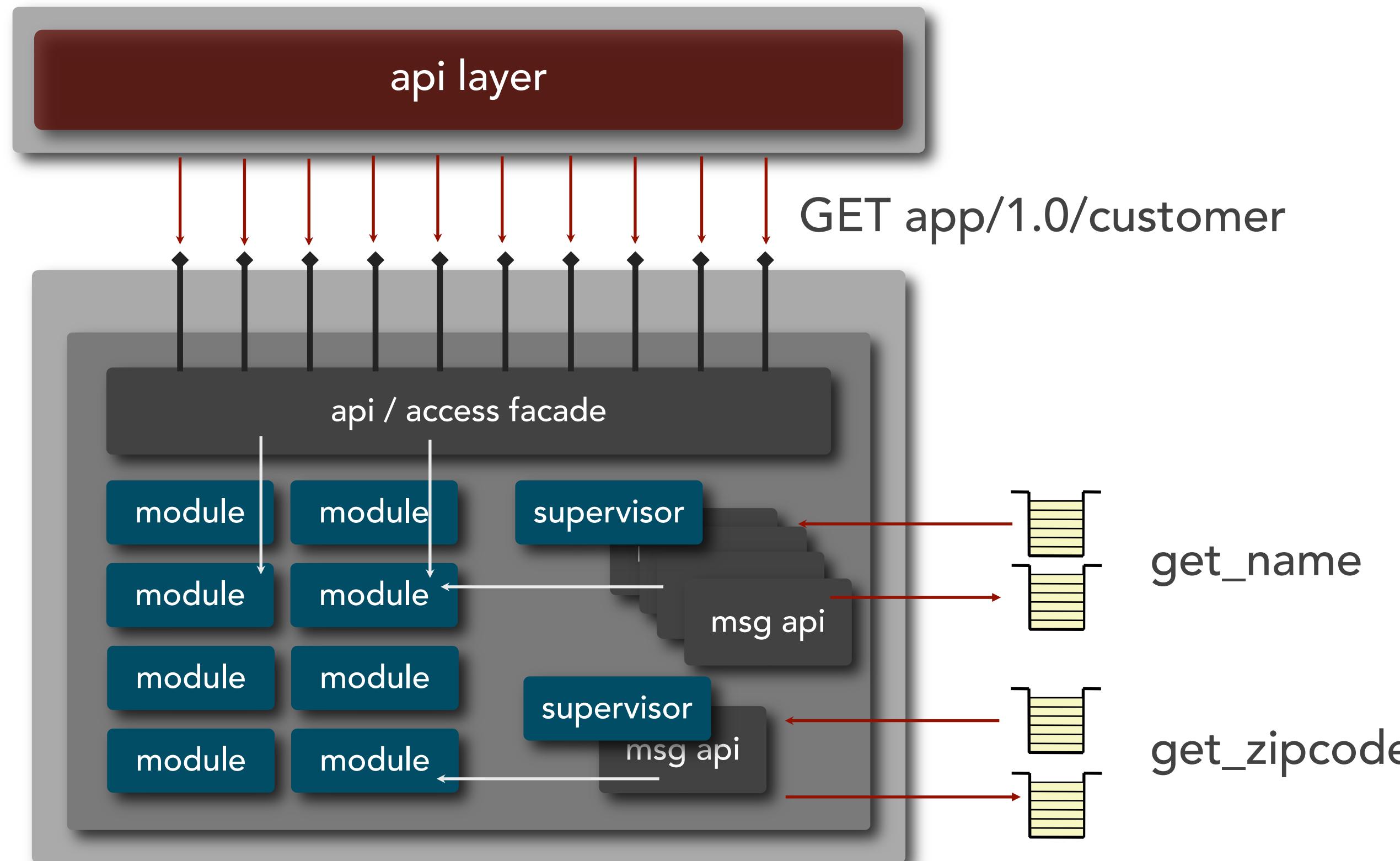
```
private Boolean active = true;

public void startup(Connection connection) {
    //get broker session or channel and create consumer
    while (active) {
        msg = getNextMessageFromQueue(1000);
        if (msg != null) //process message
    }
}
```

```
public void shutdown() {
    synchronized(active) { active = false; }
}
```

# supervisor consumer pattern

## inter-service communication



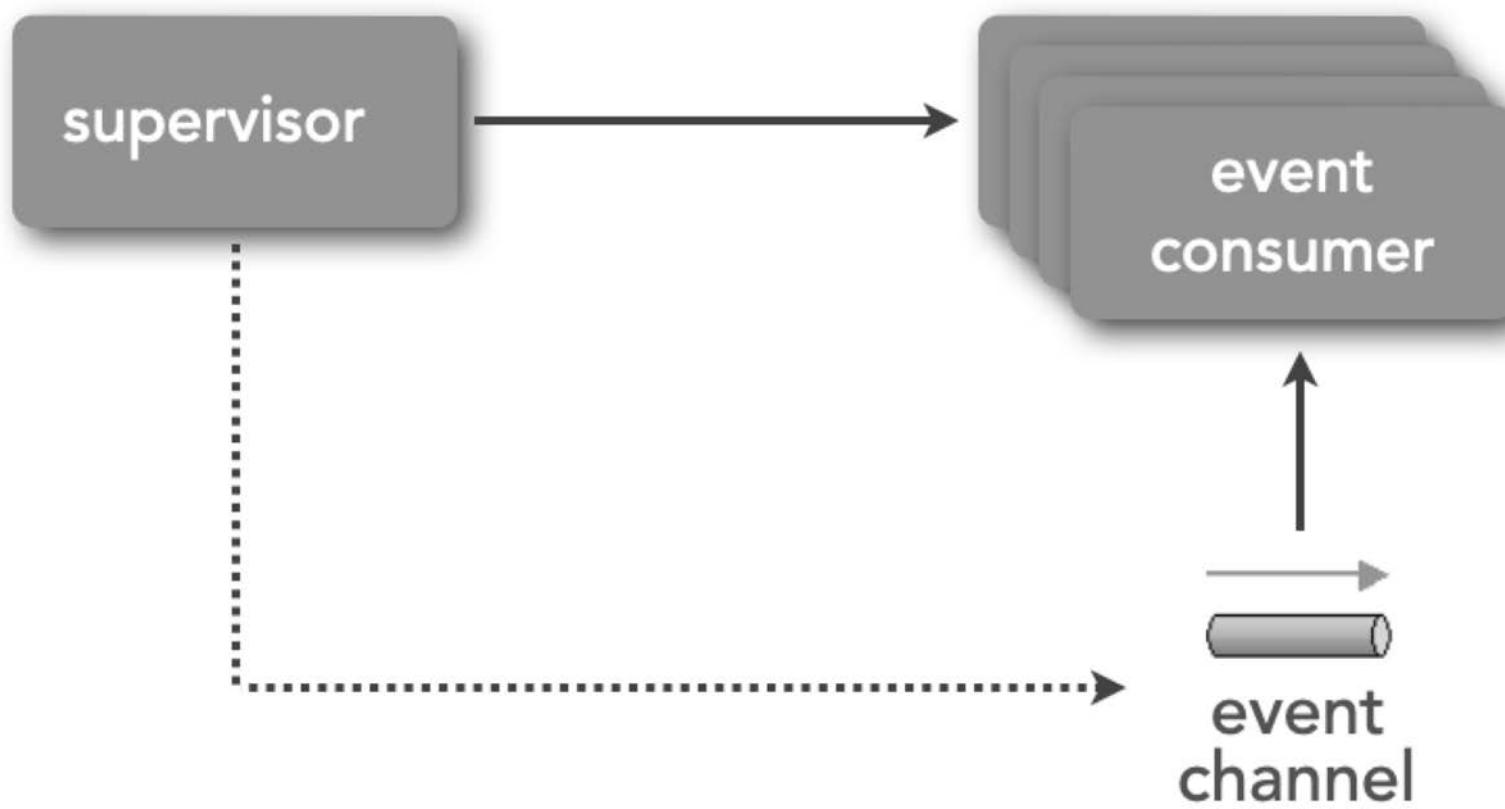
# supervisor consumer pattern



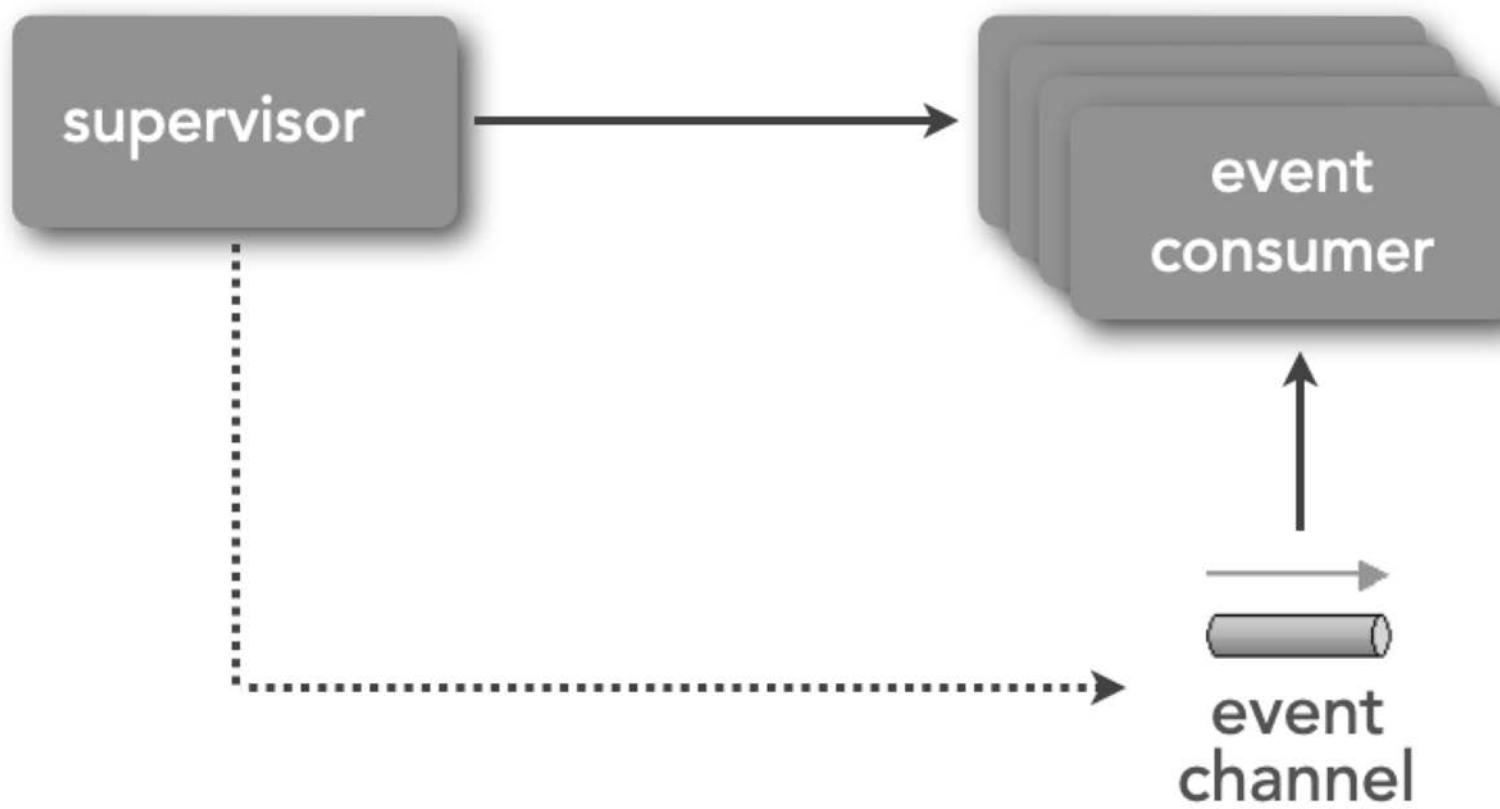
let's see this in action...

<https://github.com/wmr513/reactive/tree/master/supervisor>

# supervisor consumer pattern



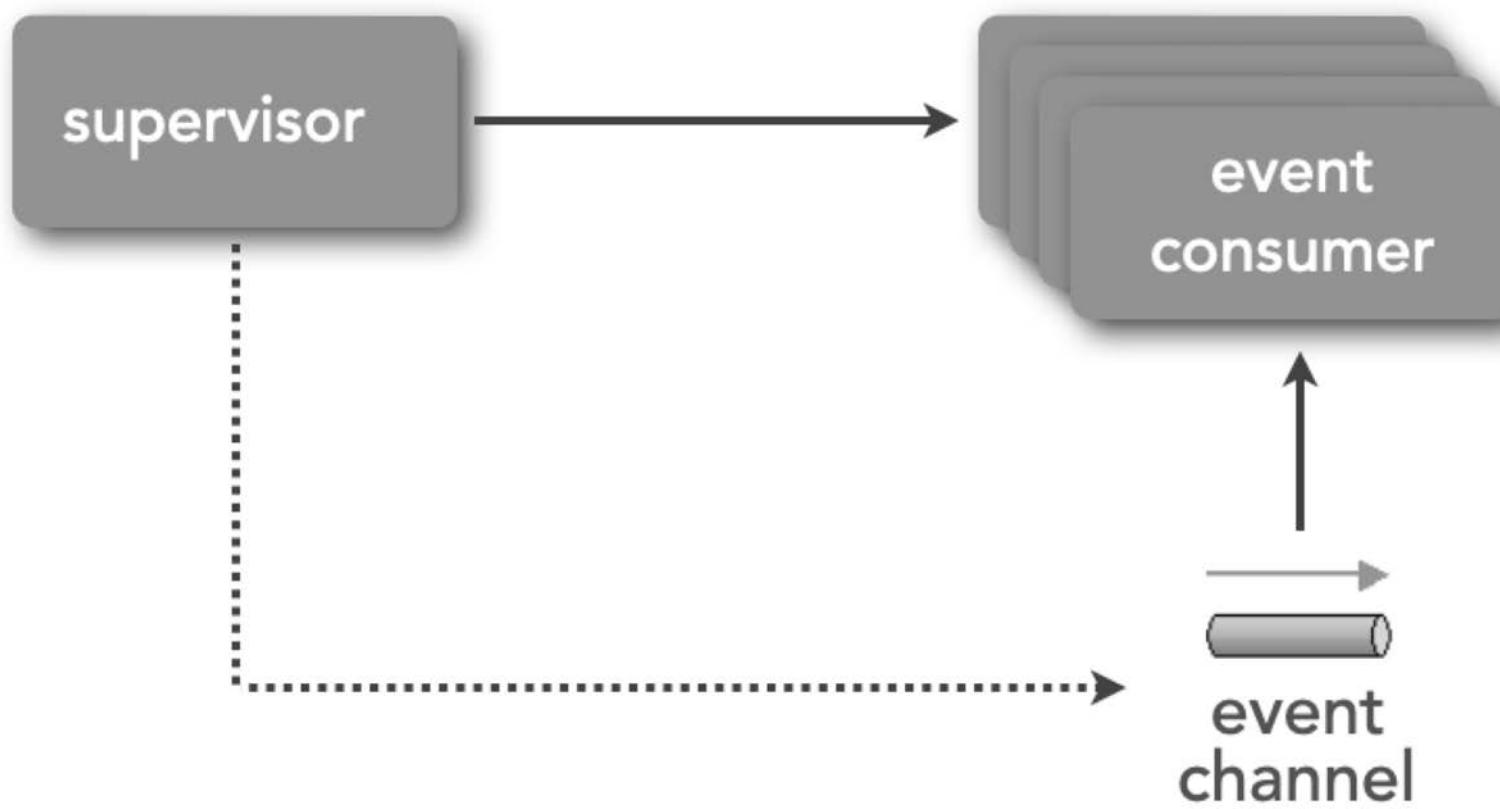
# supervisor consumer pattern



programmatic elasticity  
consistent responsiveness  
optimized thread allocation  
optimized memory usage



# supervisor consumer pattern



- programmatic elasticity
- consistent responsiveness
- optimized thread allocation
- optimized memory usage

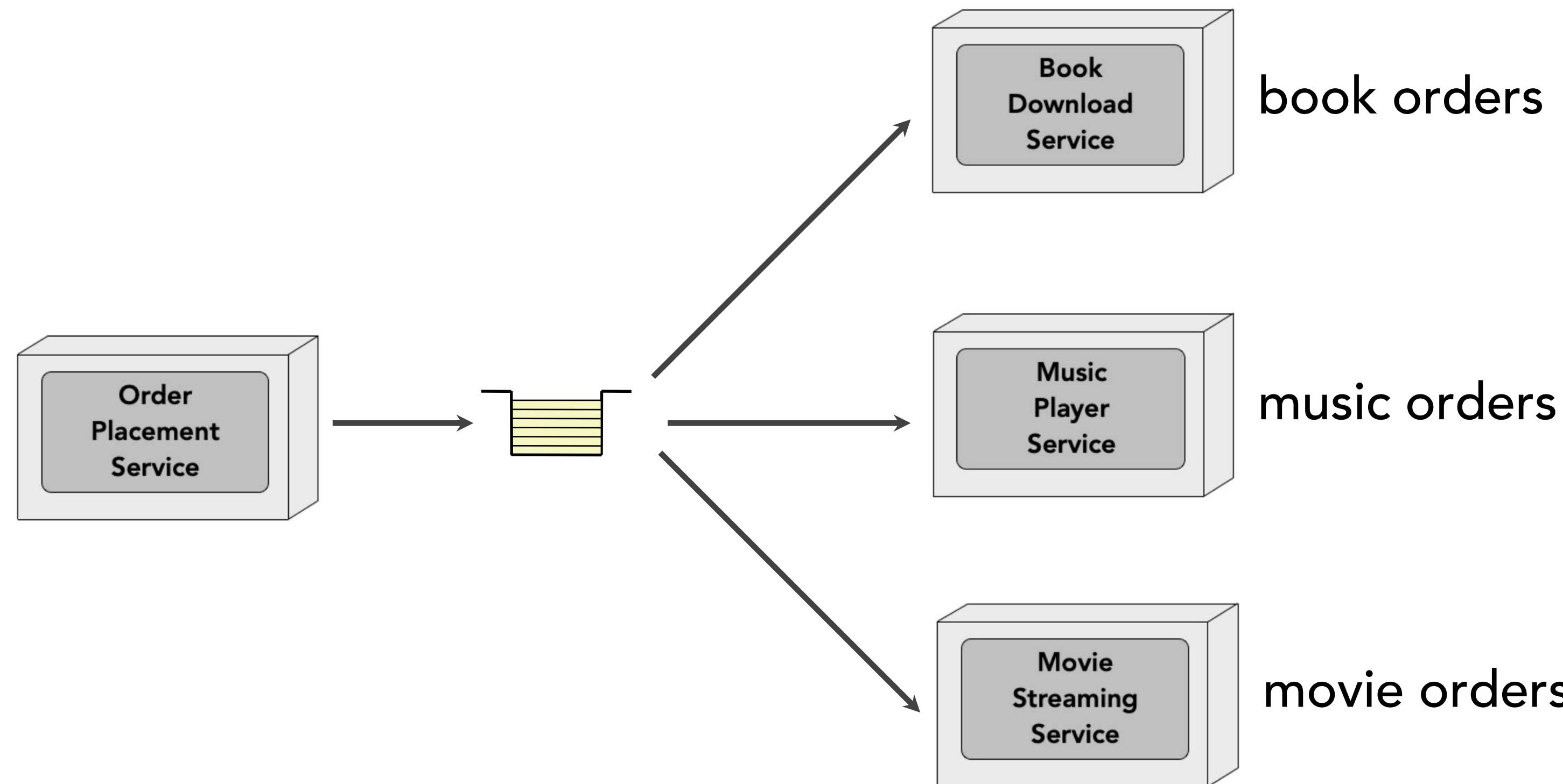


- increased complexity
- possible thread saturation
- possible memory errors
- impact on other requests

# Contextual Queue Pattern

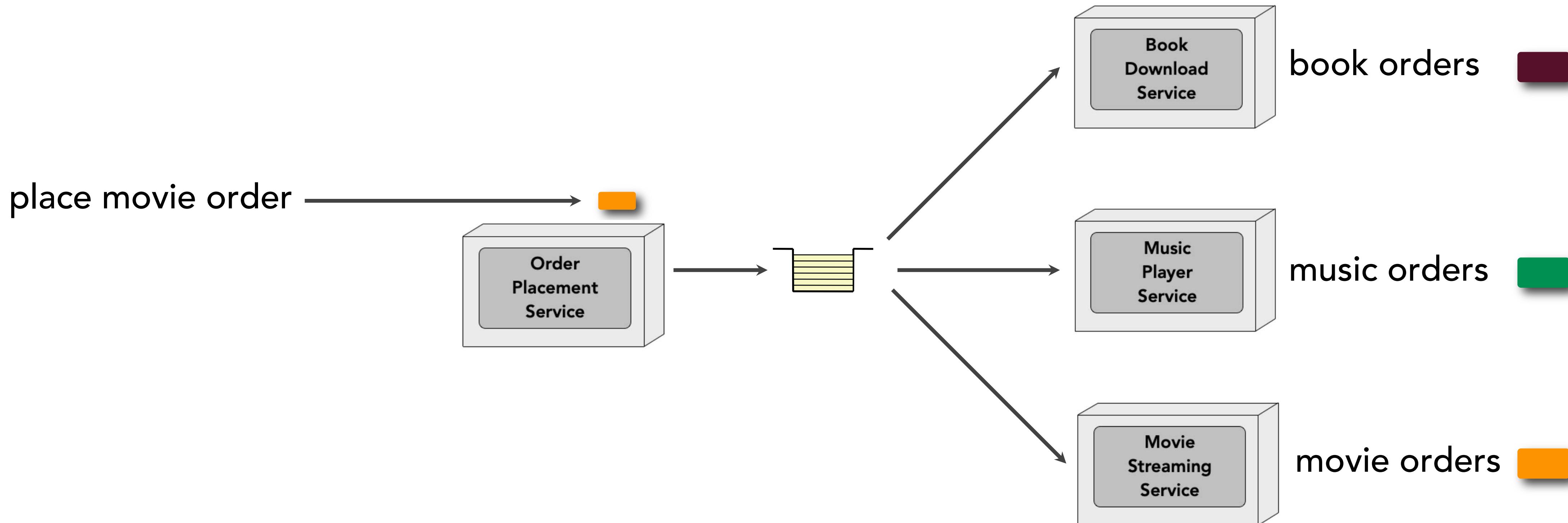
# contextual queue pattern

*“how can I load balance heterogeneous events to ensure consistent response times?”*



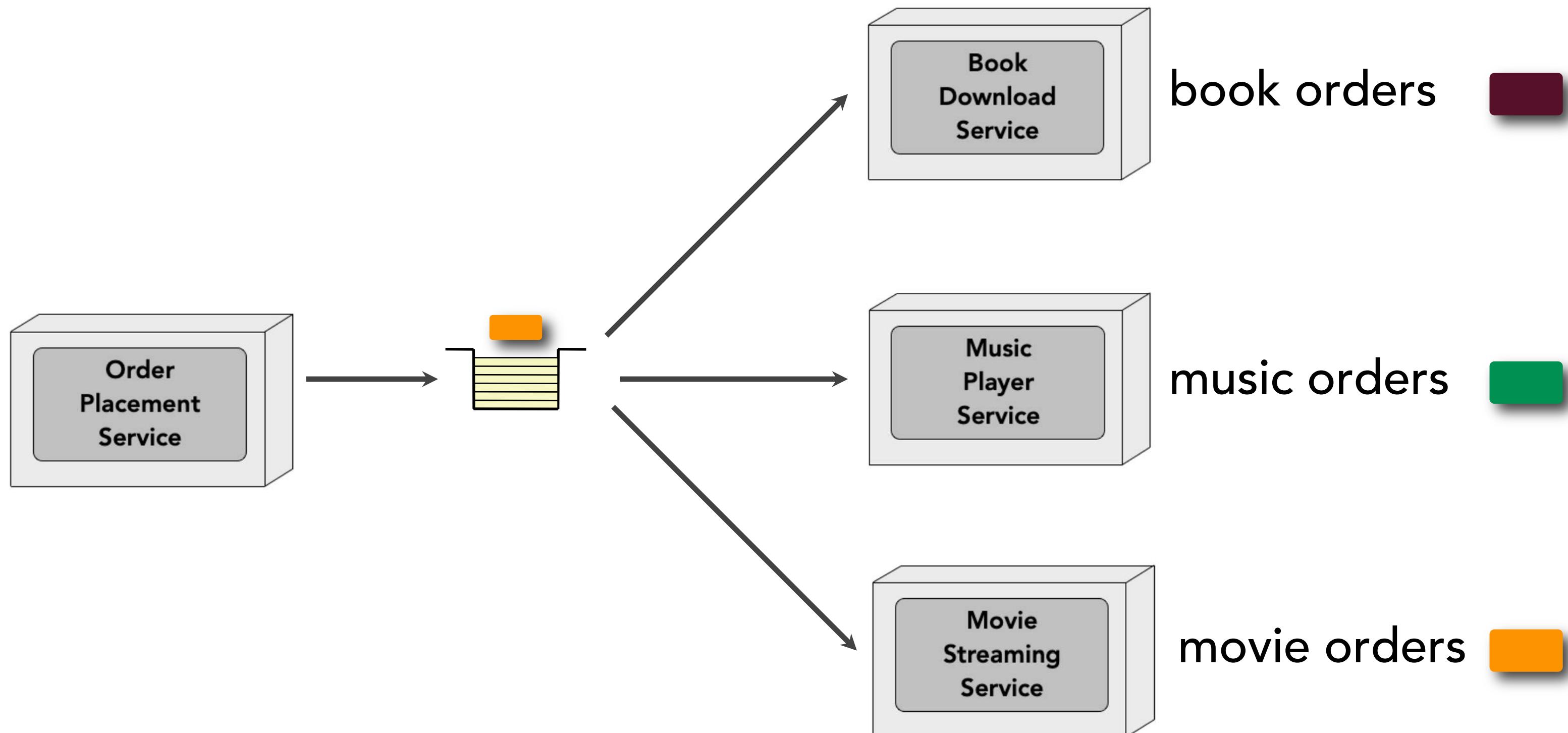
# contextual queue pattern

book orders, music orders, and movie orders are processed by different services



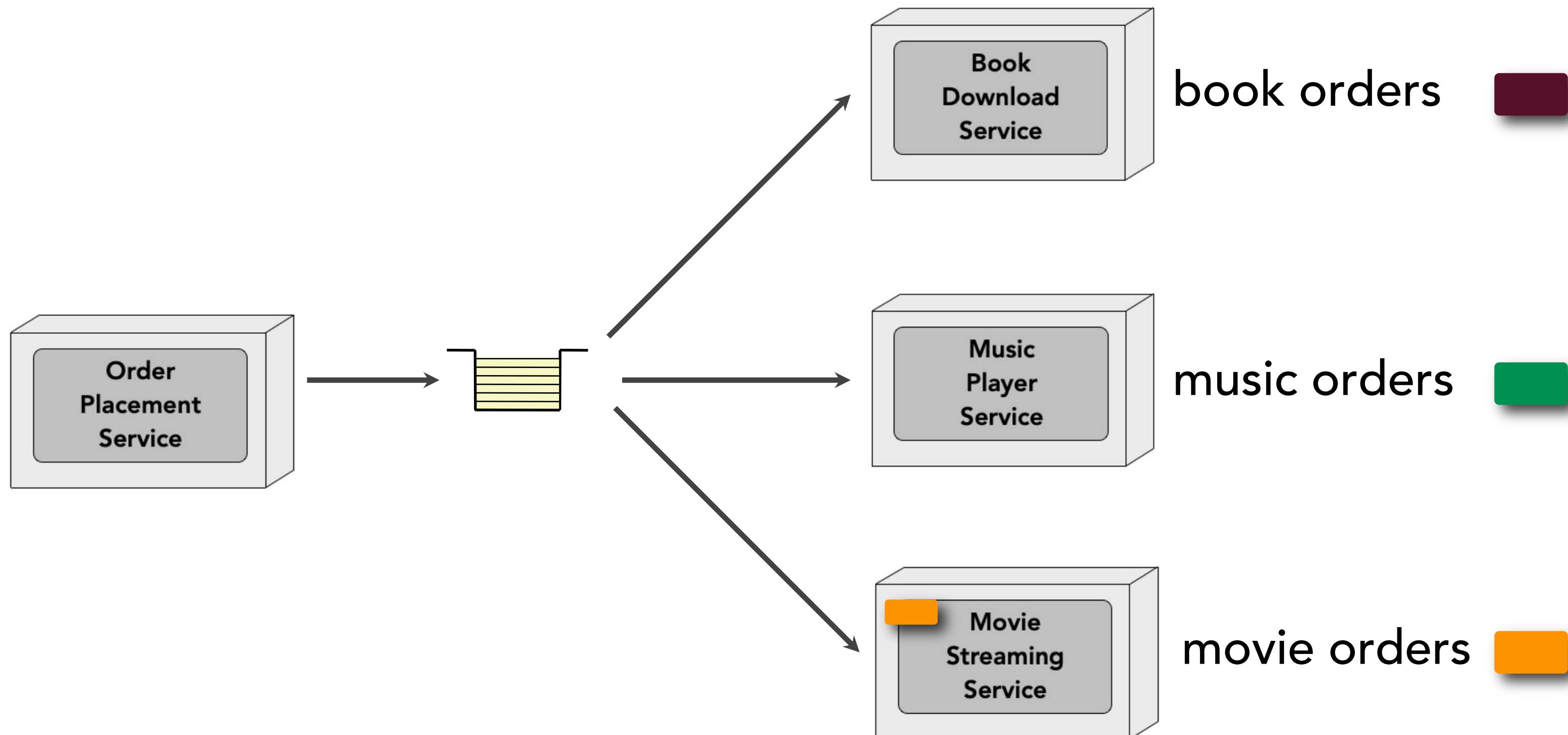
# contextual queue pattern

book orders, music orders, and movie orders are processed by different services



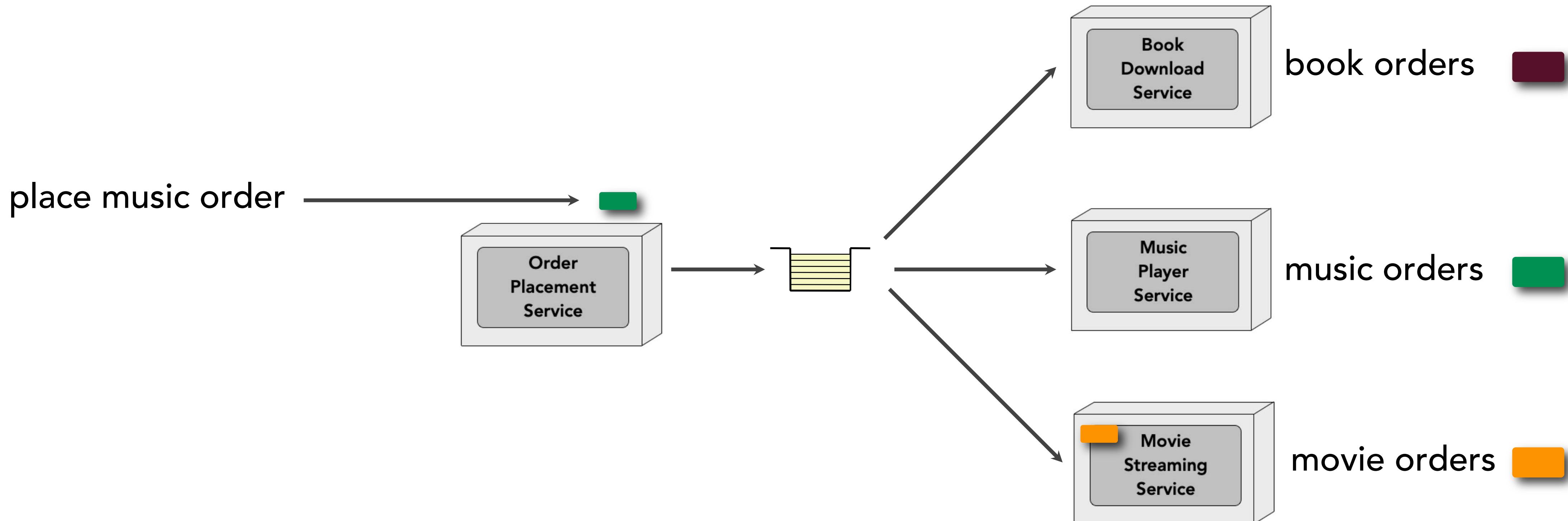
# contextual queue pattern

book orders, music orders, and movie orders are processed by different services



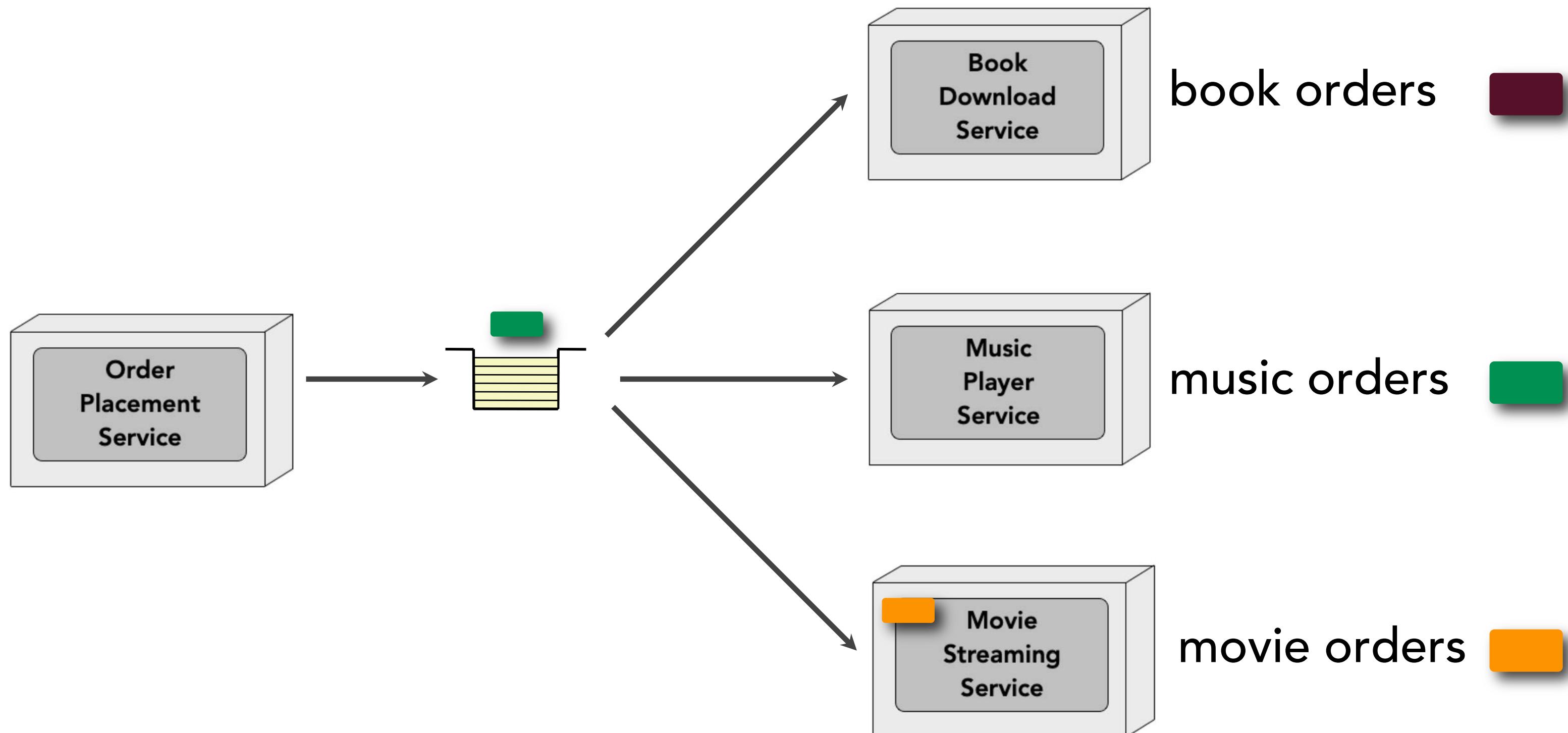
# contextual queue pattern

book orders, music orders, and movie orders are processed by different services



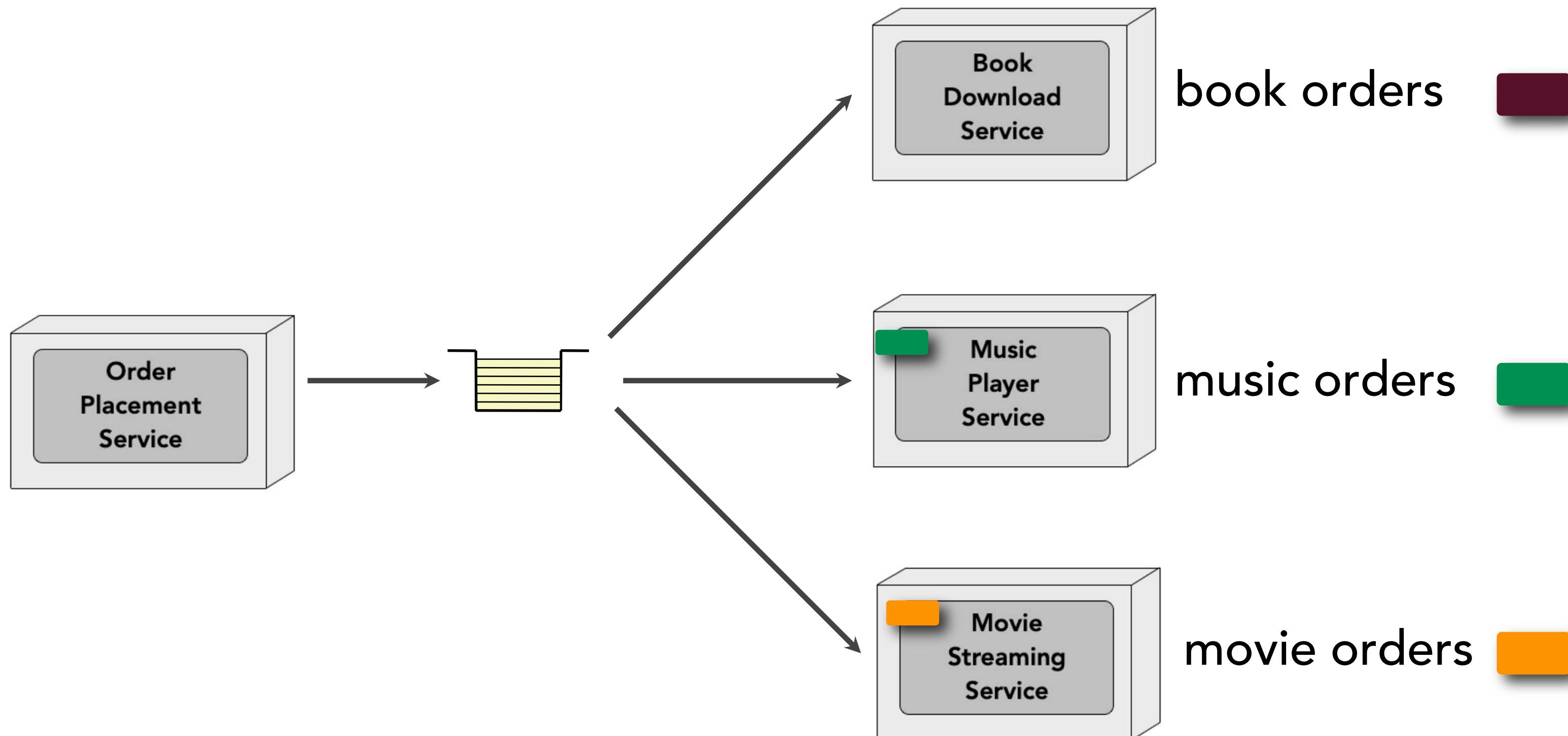
# contextual queue pattern

book orders, music orders, and movie orders are processed by different services



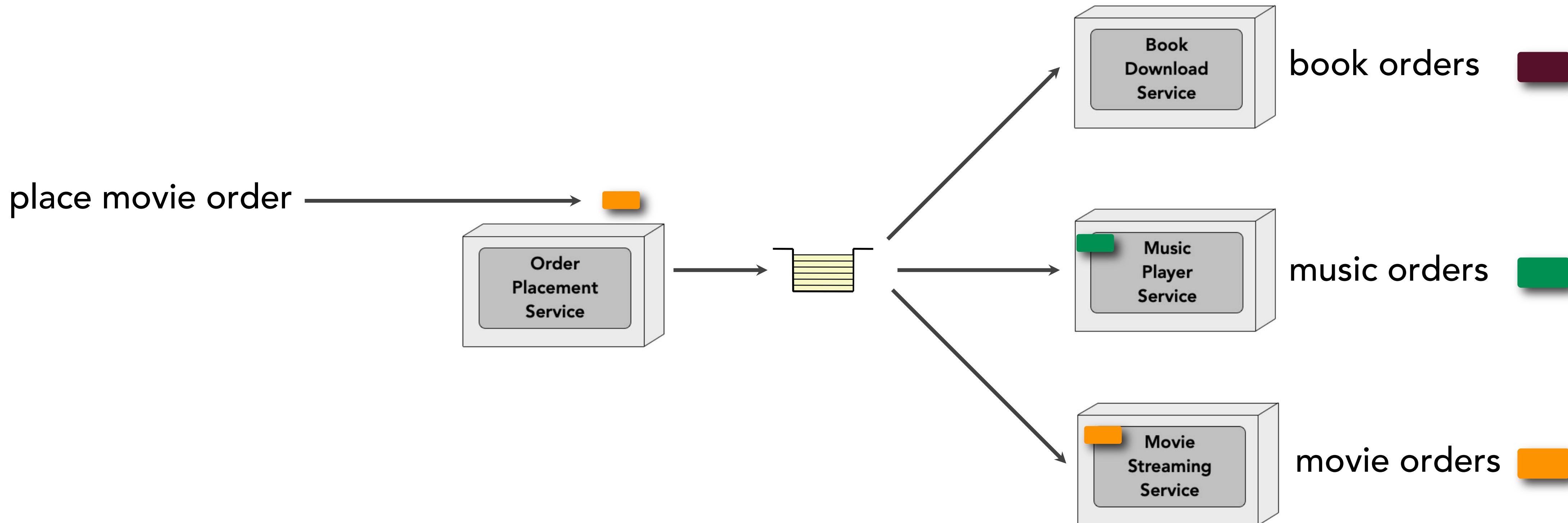
# contextual queue pattern

book orders, music orders, and movie orders are processed by different services



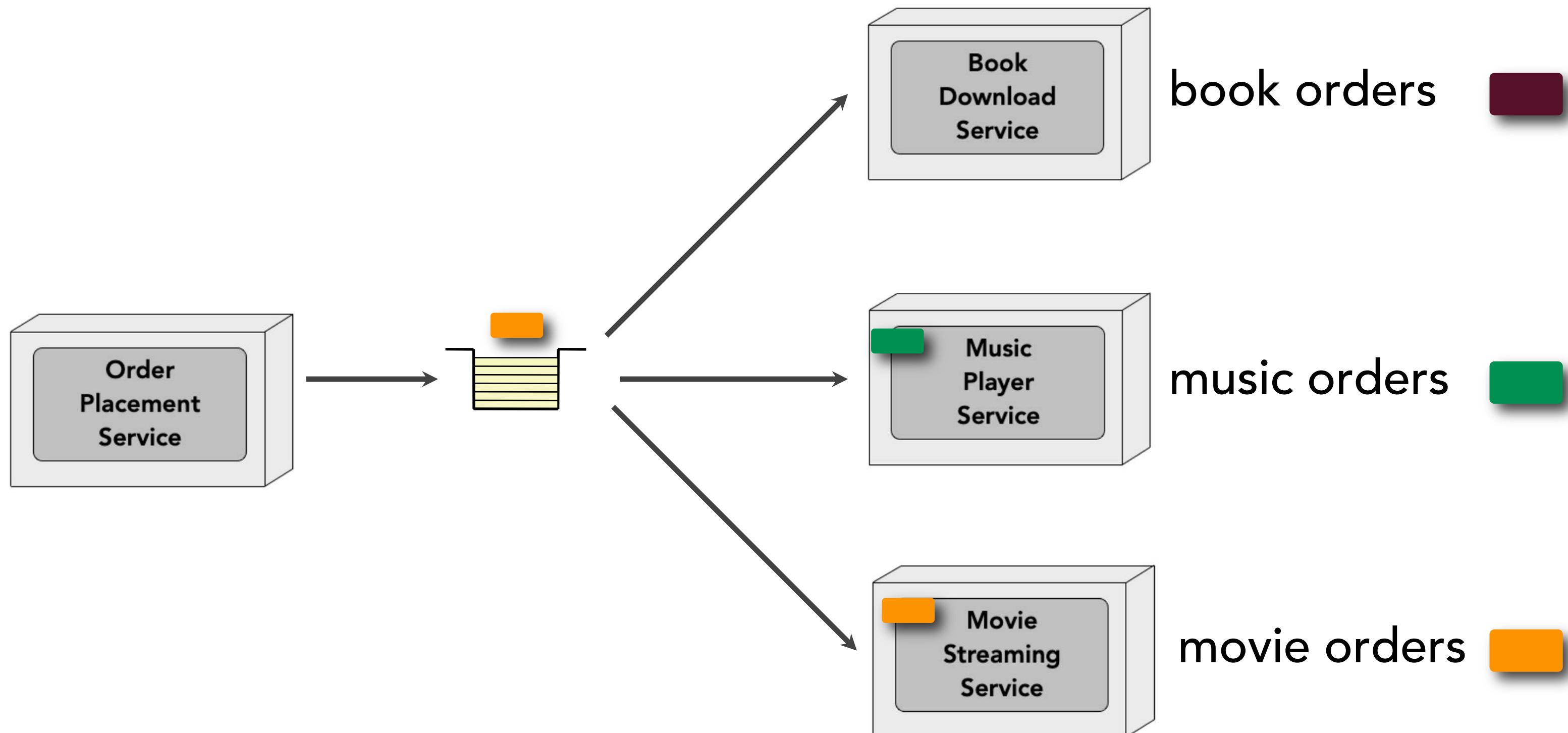
# contextual queue pattern

book orders, music orders, and movie orders are processed by different services



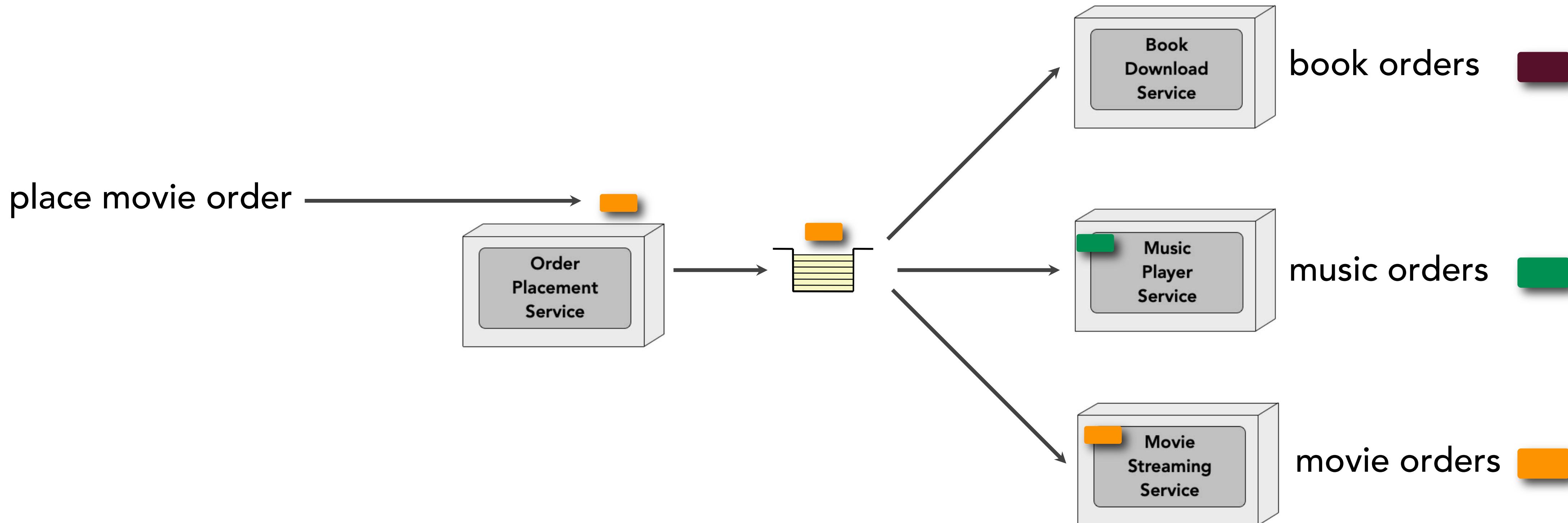
# contextual queue pattern

book orders, music orders, and movie orders are processed by different services



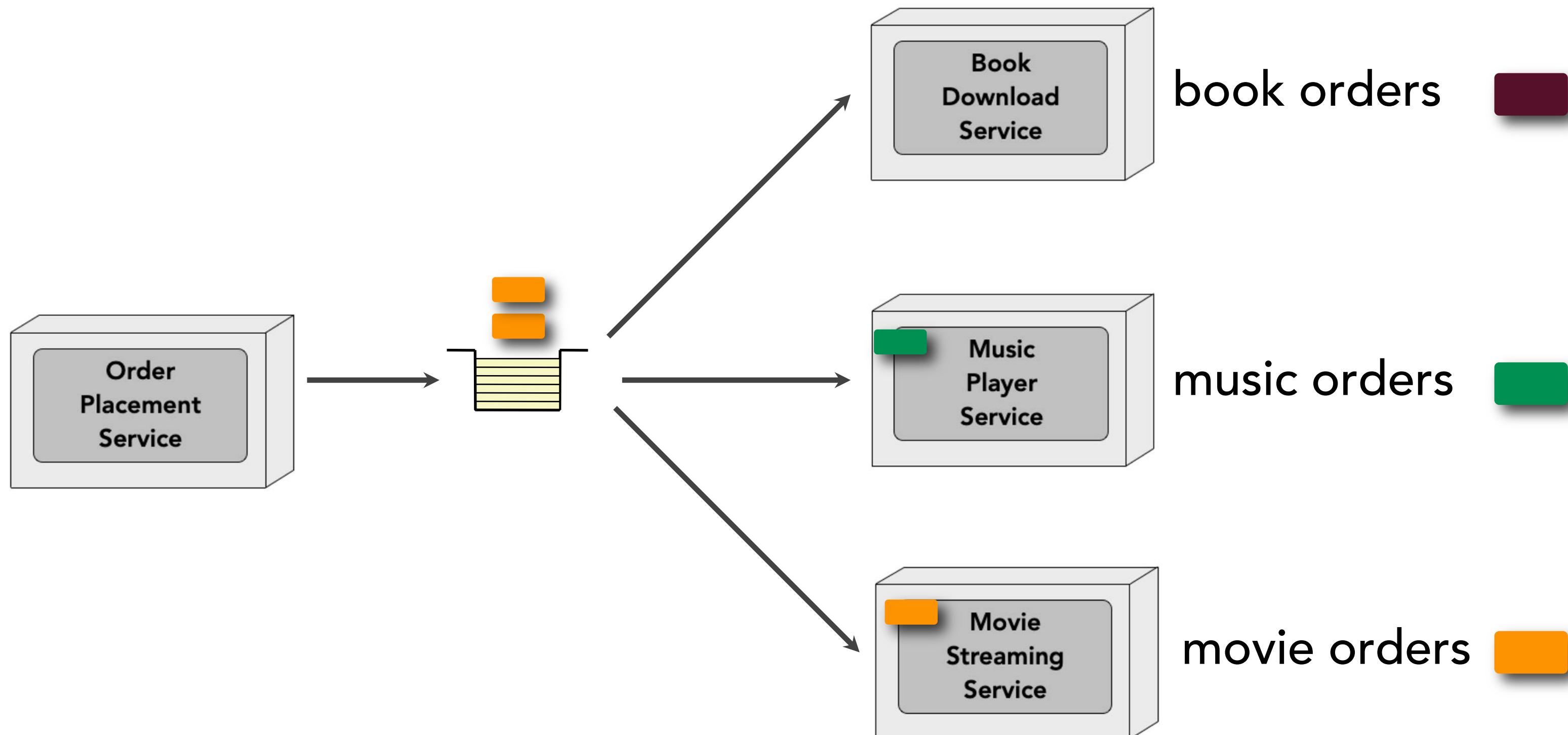
# contextual queue pattern

book orders, music orders, and movie orders are processed by different services



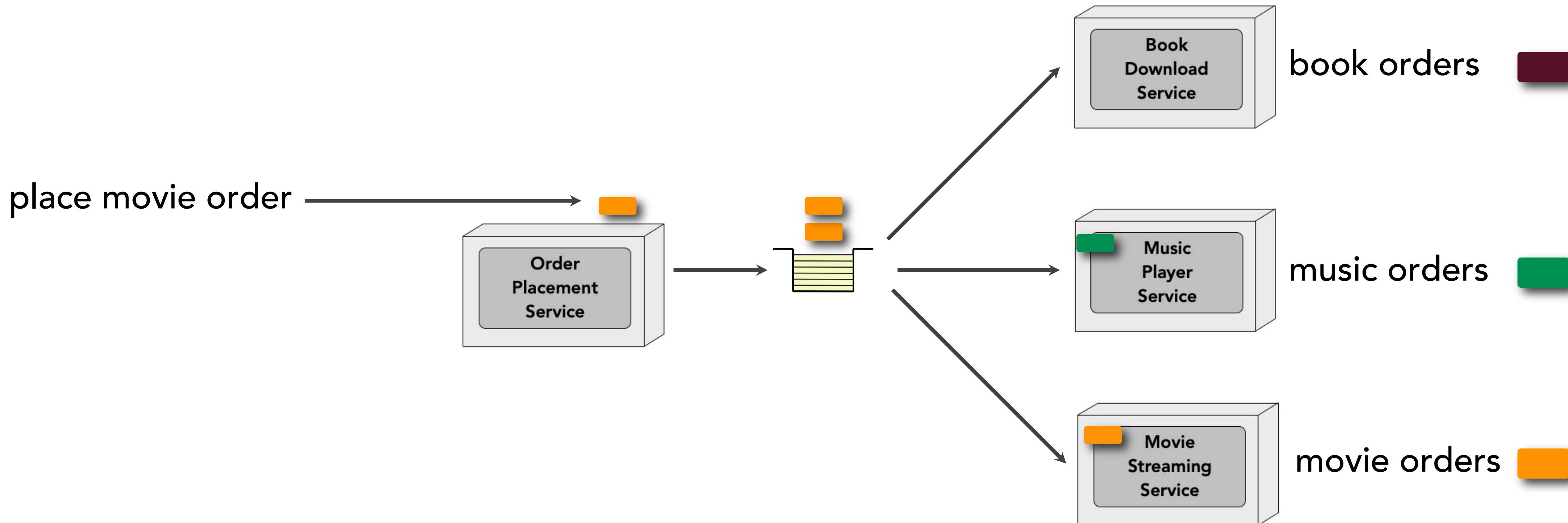
# contextual queue pattern

book orders, music orders, and movie orders are processed by different services



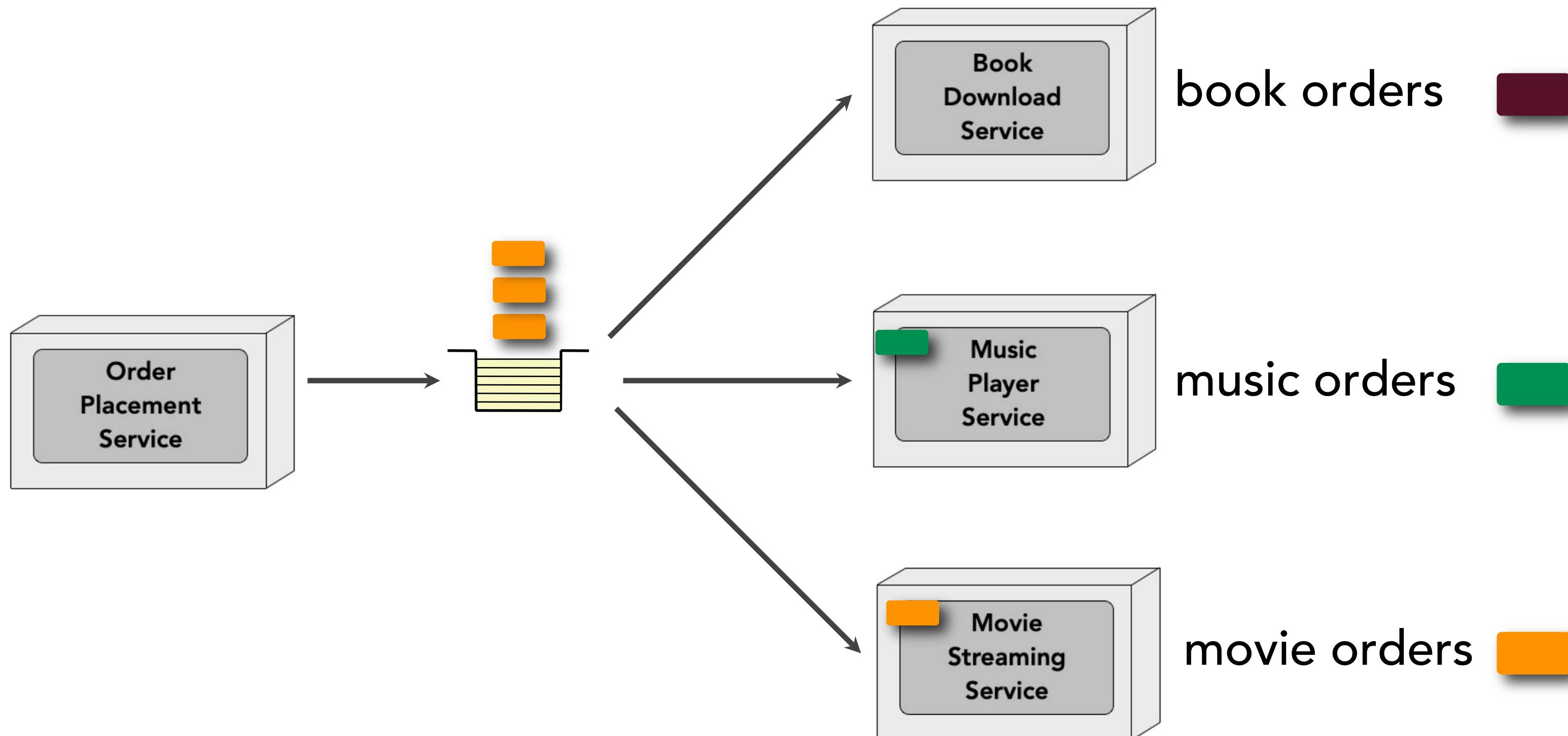
# contextual queue pattern

book orders, music orders, and movie orders are processed by different services



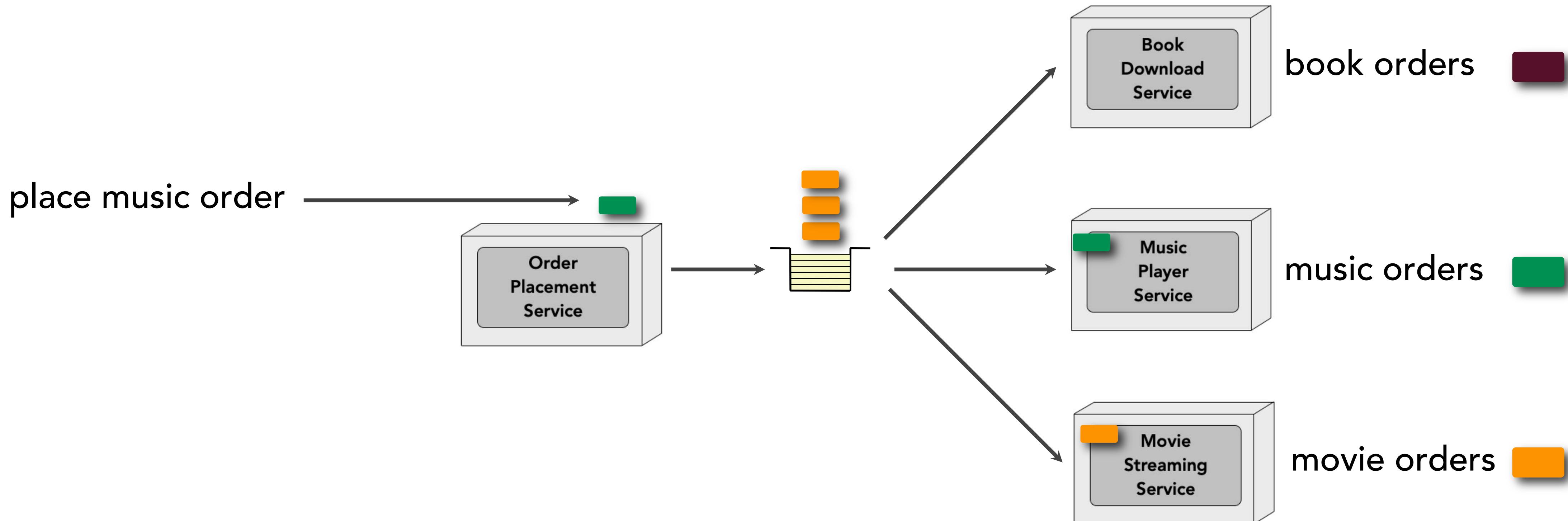
# contextual queue pattern

book orders, music orders, and movie orders are processed by different services



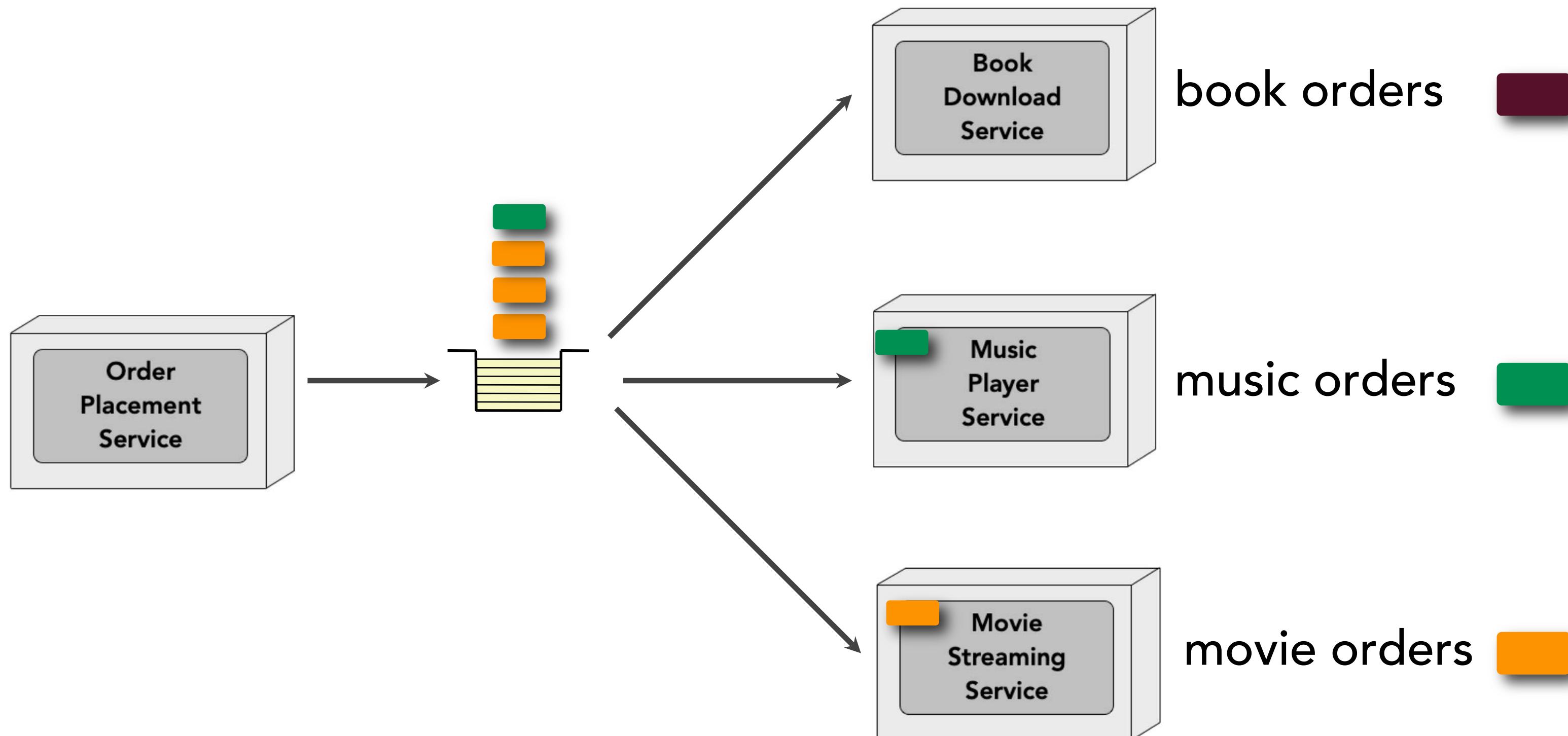
# contextual queue pattern

book orders, music orders, and movie orders are processed by different services



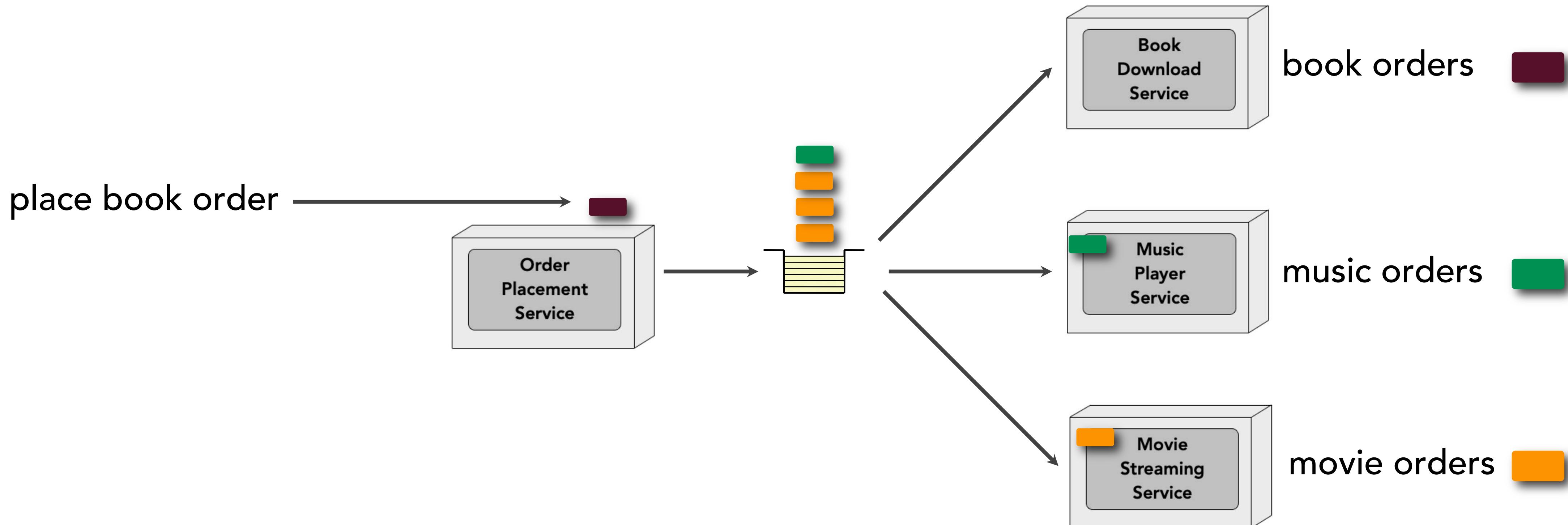
# contextual queue pattern

book orders, music orders, and movie orders are processed by different services



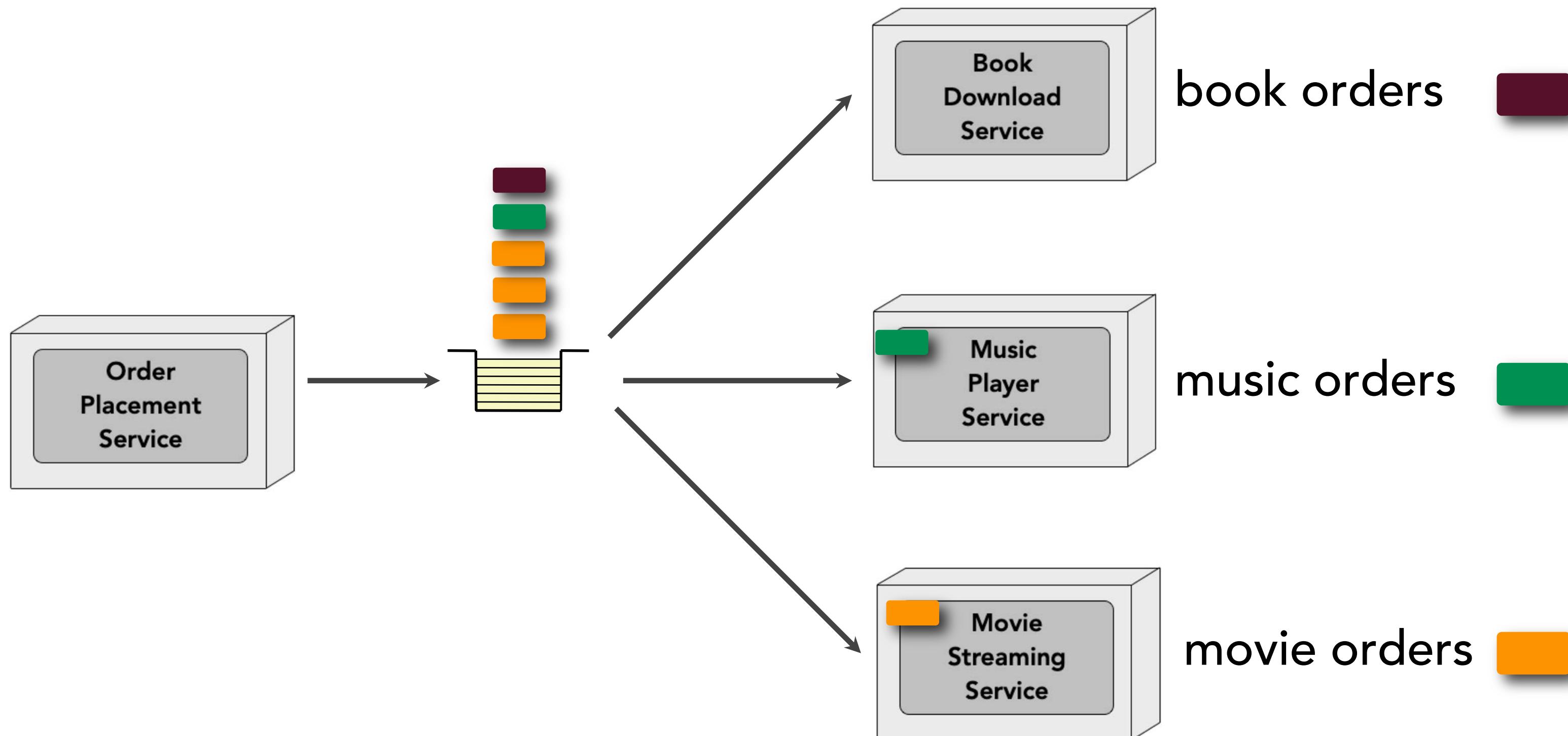
# contextual queue pattern

book orders, music orders, and movie orders are processed by different services



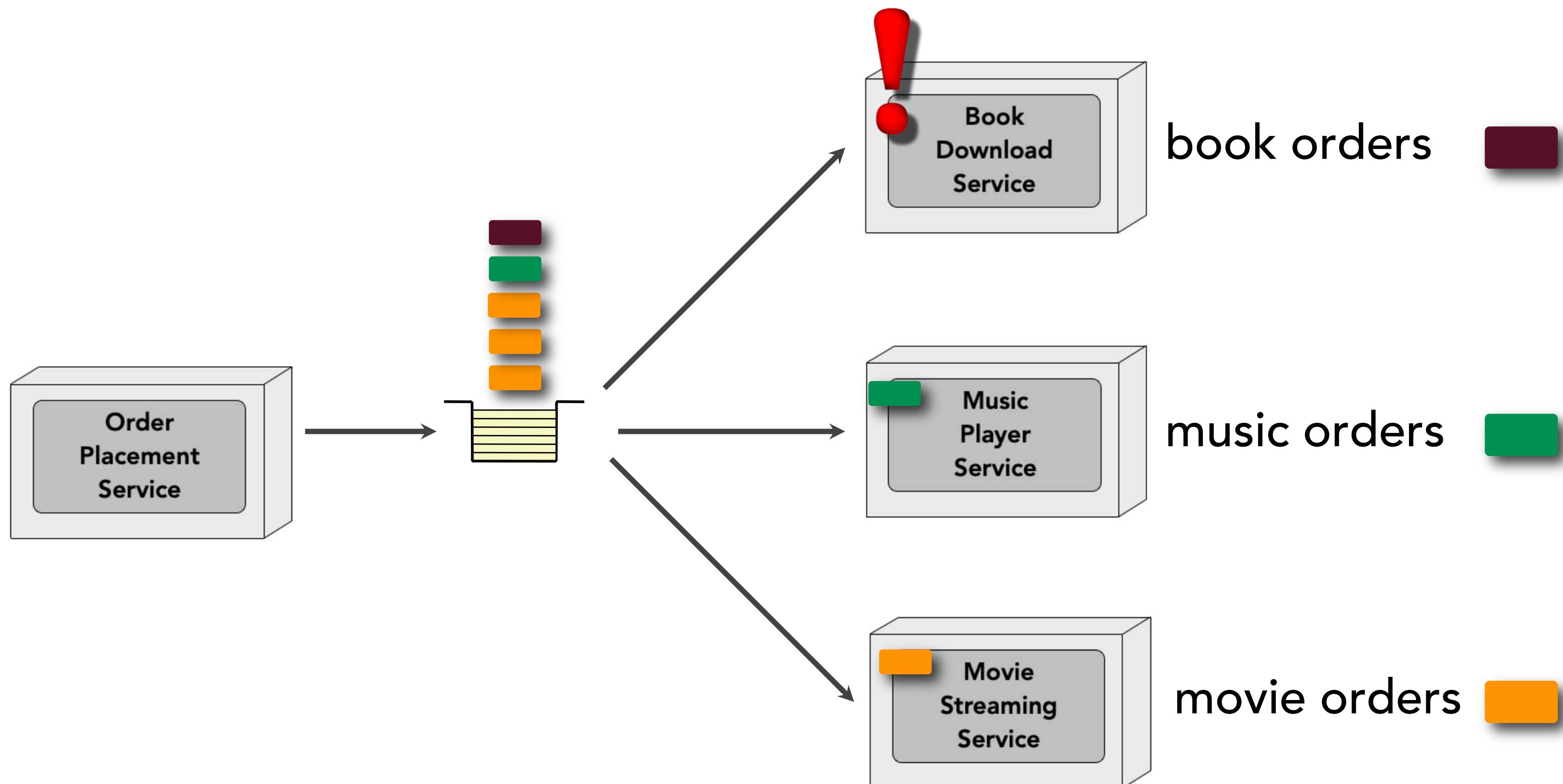
# contextual queue pattern

book orders, music orders, and movie orders are processed by different services

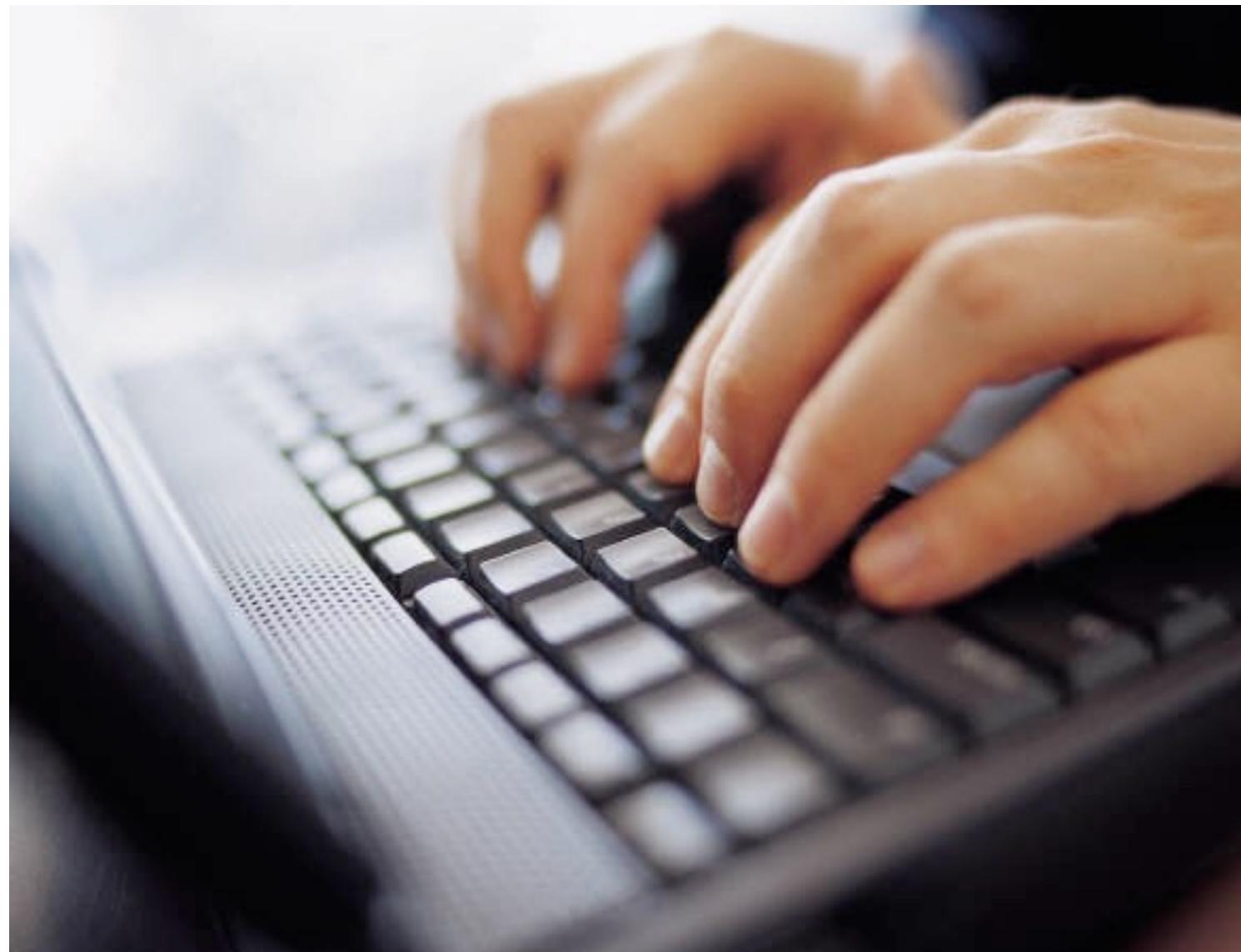


# contextual queue pattern

book orders, music orders, and movie orders are processed by different services



# contextual queue pattern

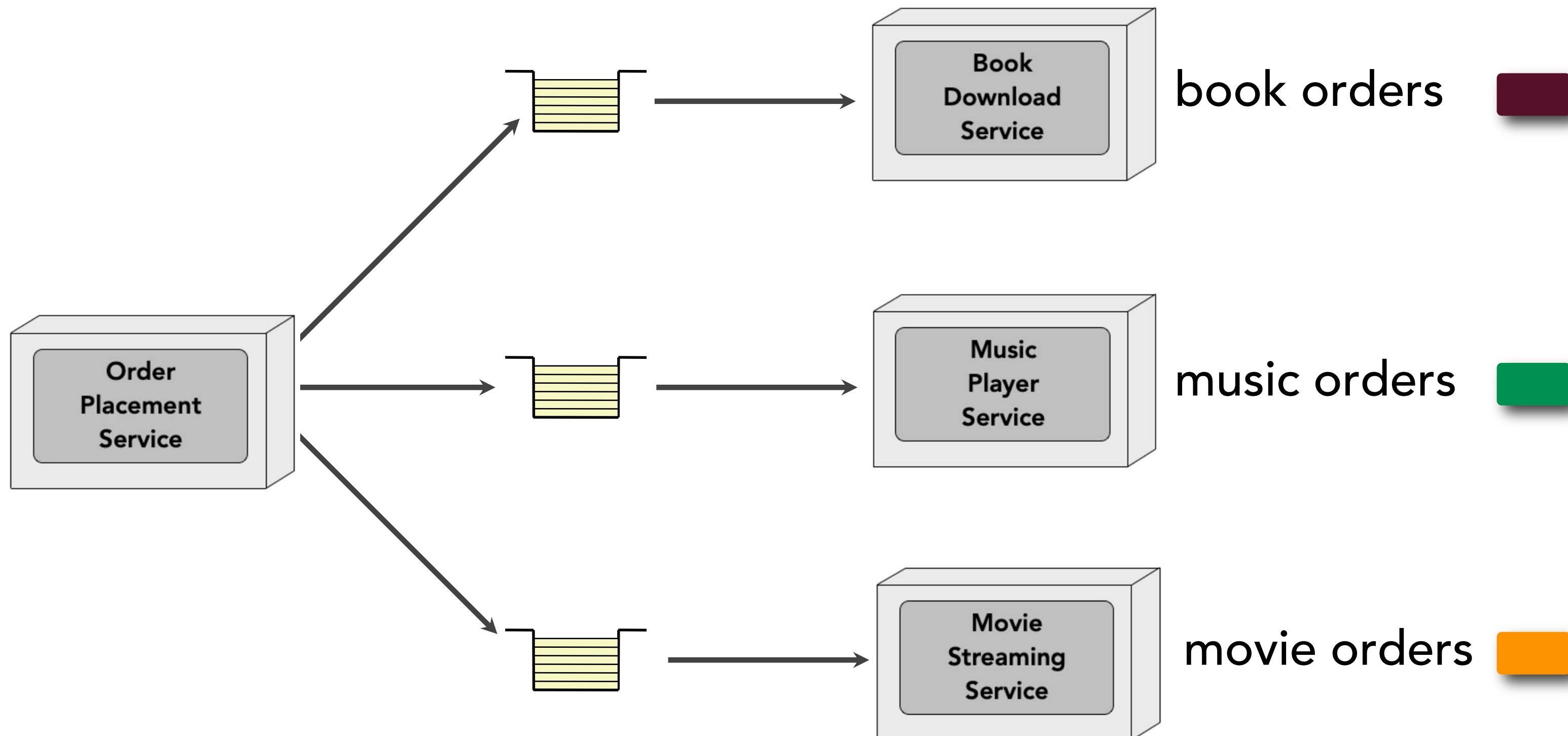


let's see the issue...

<https://github.com/wmr513/event-driven-patterns/tree/master/contextualqueue>

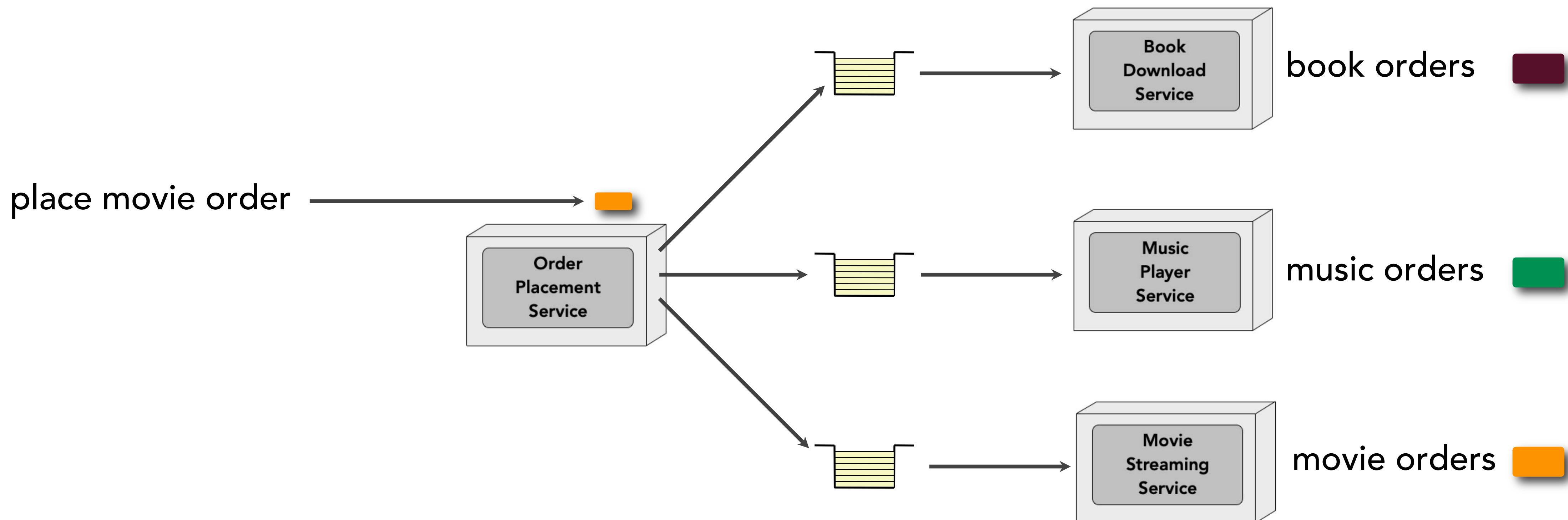
# contextual queue pattern

book orders, music orders, and movie orders are processed by different services



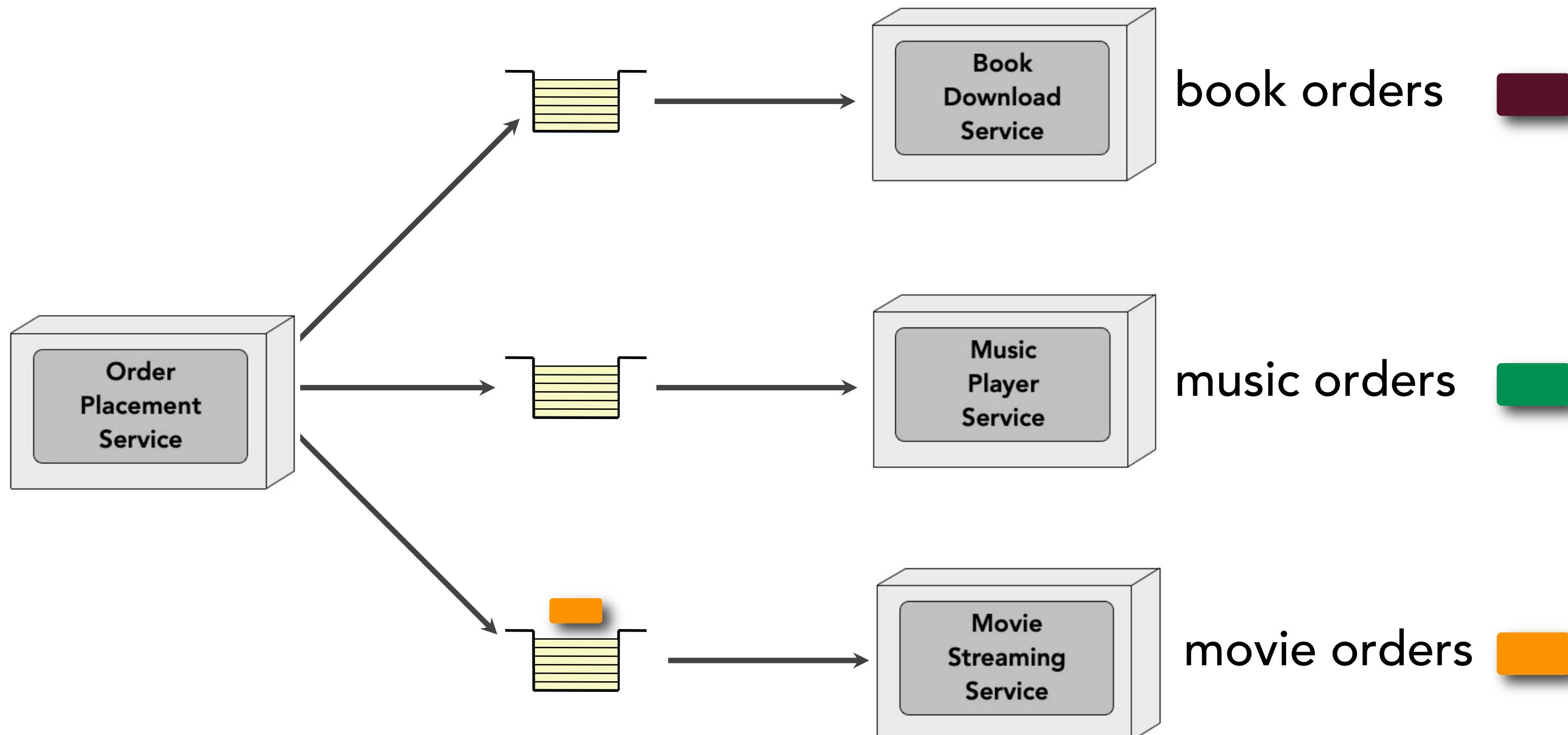
# contextual queue pattern

book orders, music orders, and movie orders are processed by different services



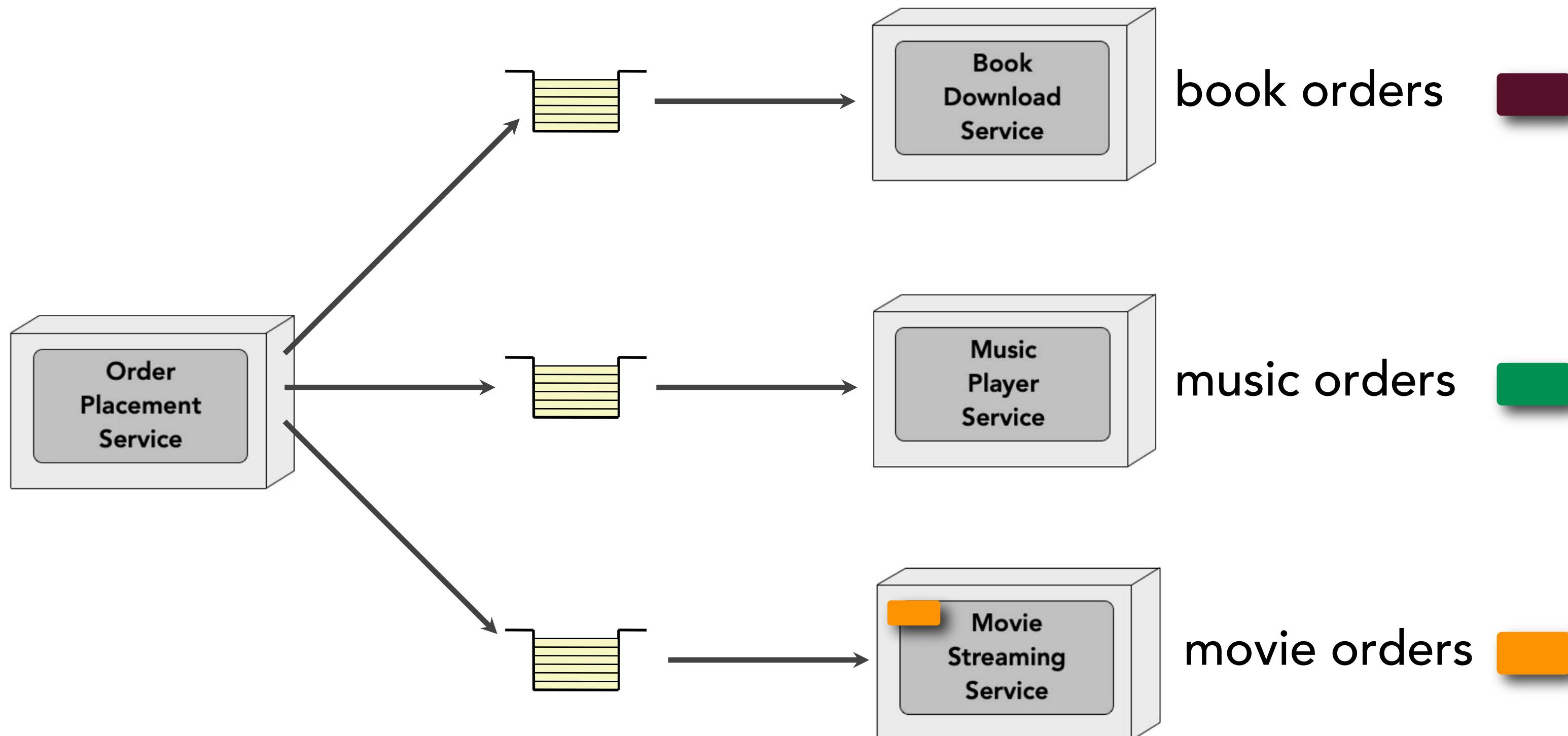
# contextual queue pattern

book orders, music orders, and movie orders are processed by different services



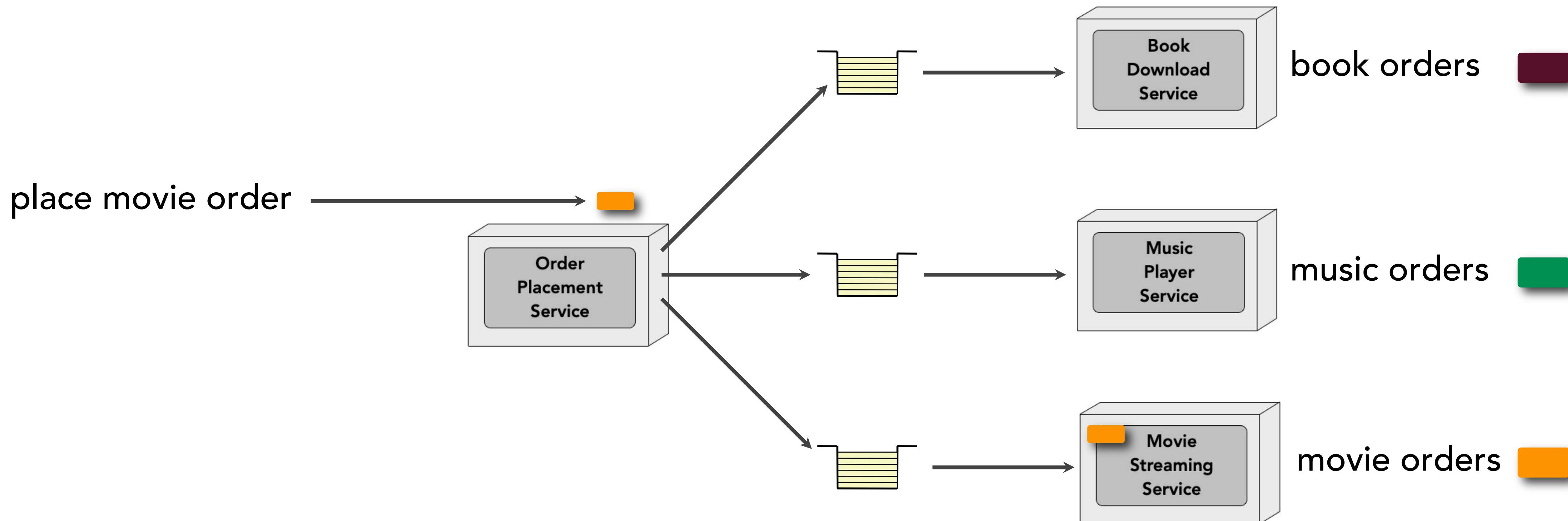
# contextual queue pattern

book orders, music orders, and movie orders are processed by different services



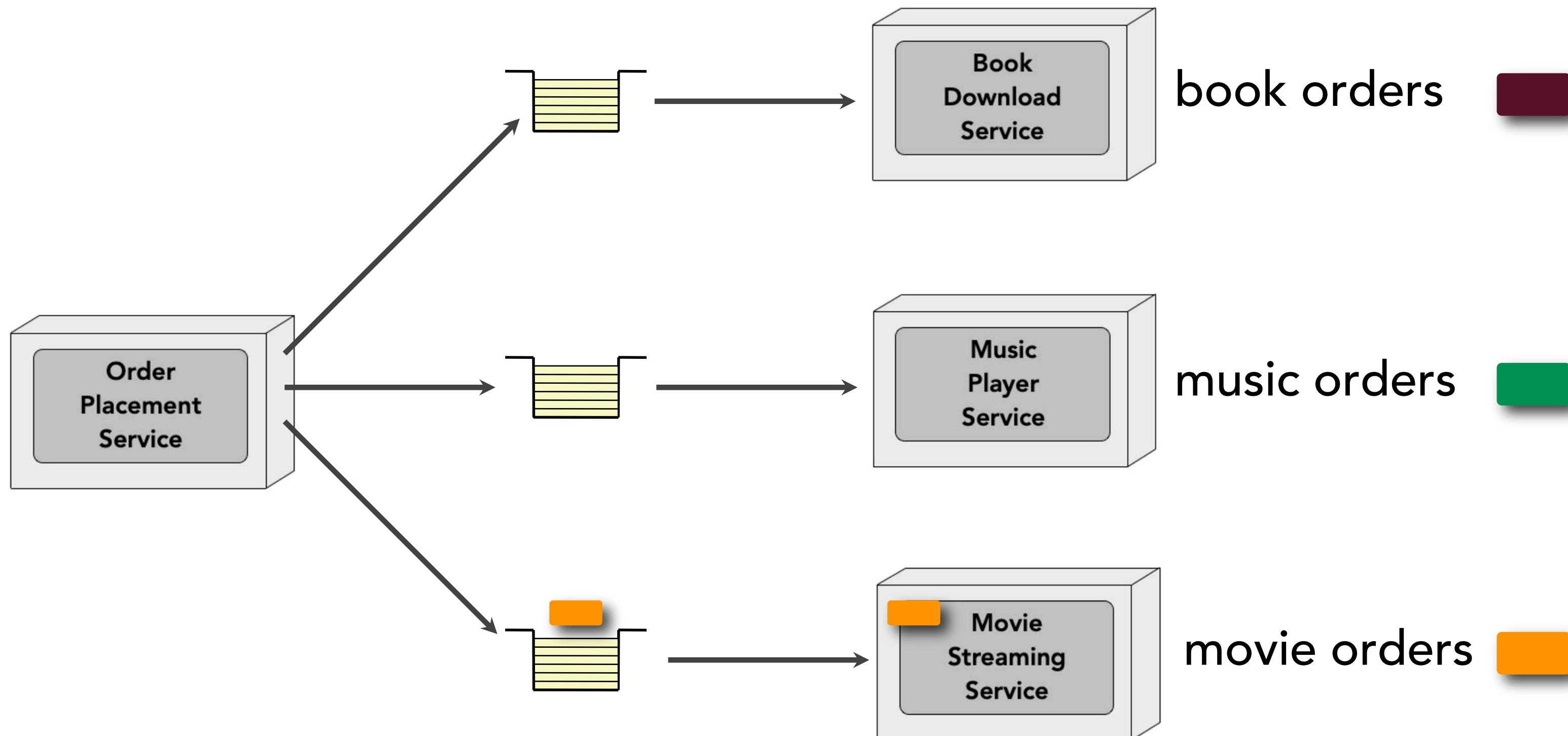
# contextual queue pattern

book orders, music orders, and movie orders are processed by different services



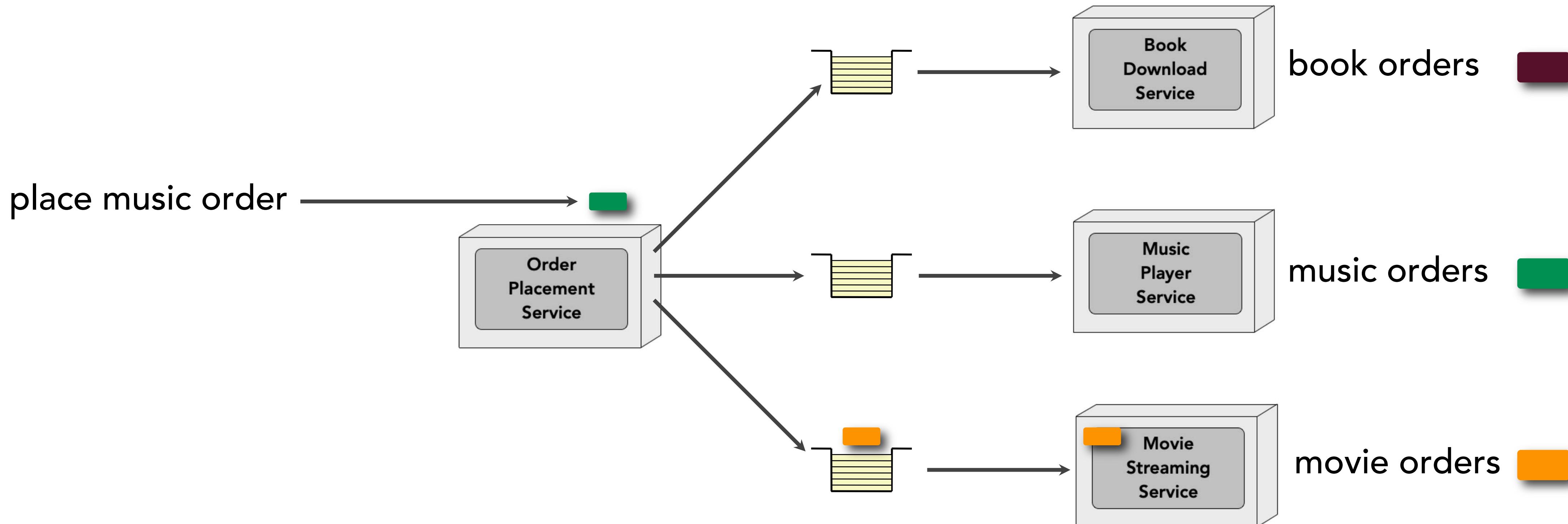
# contextual queue pattern

book orders, music orders, and movie orders are processed by different services



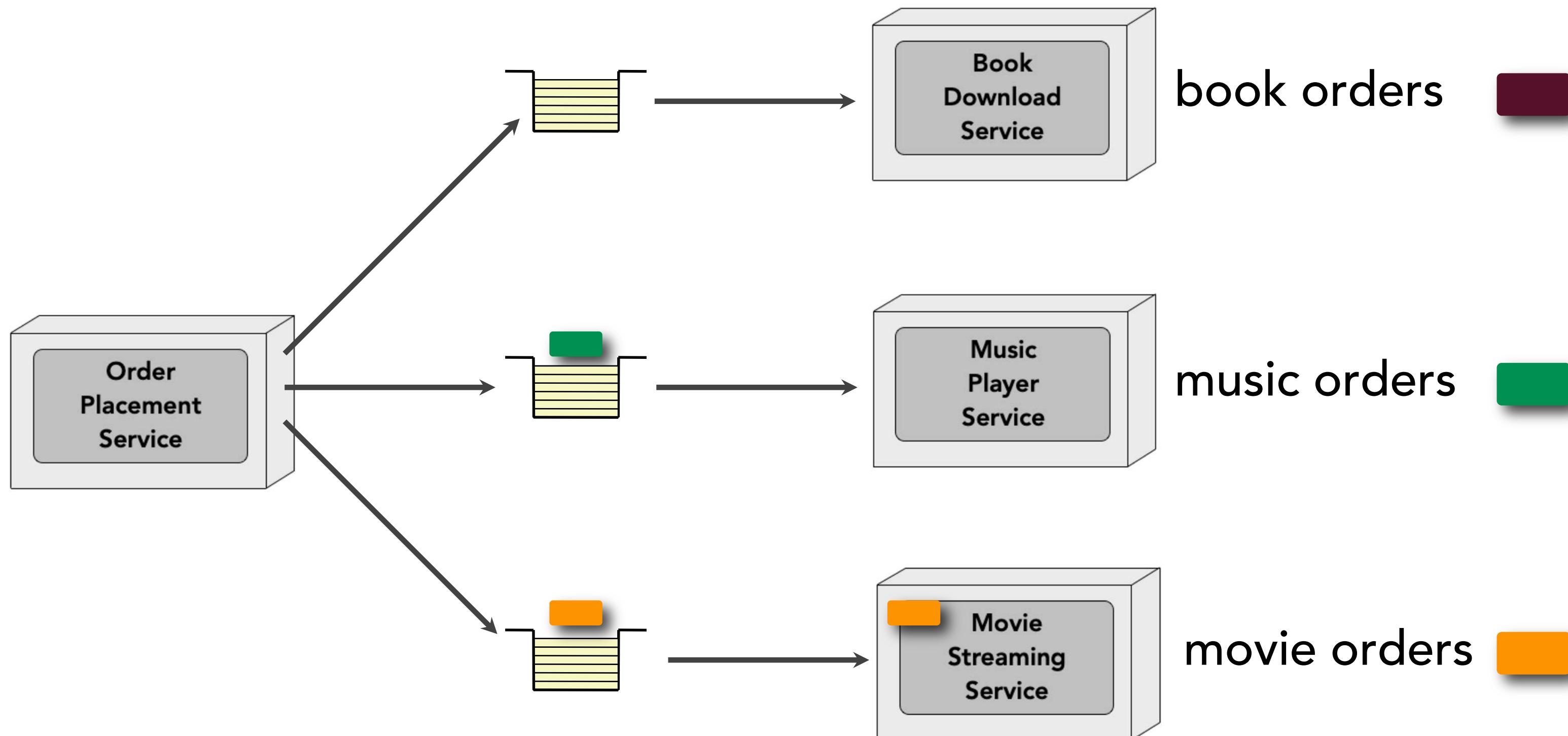
# contextual queue pattern

book orders, music orders, and movie orders are processed by different services



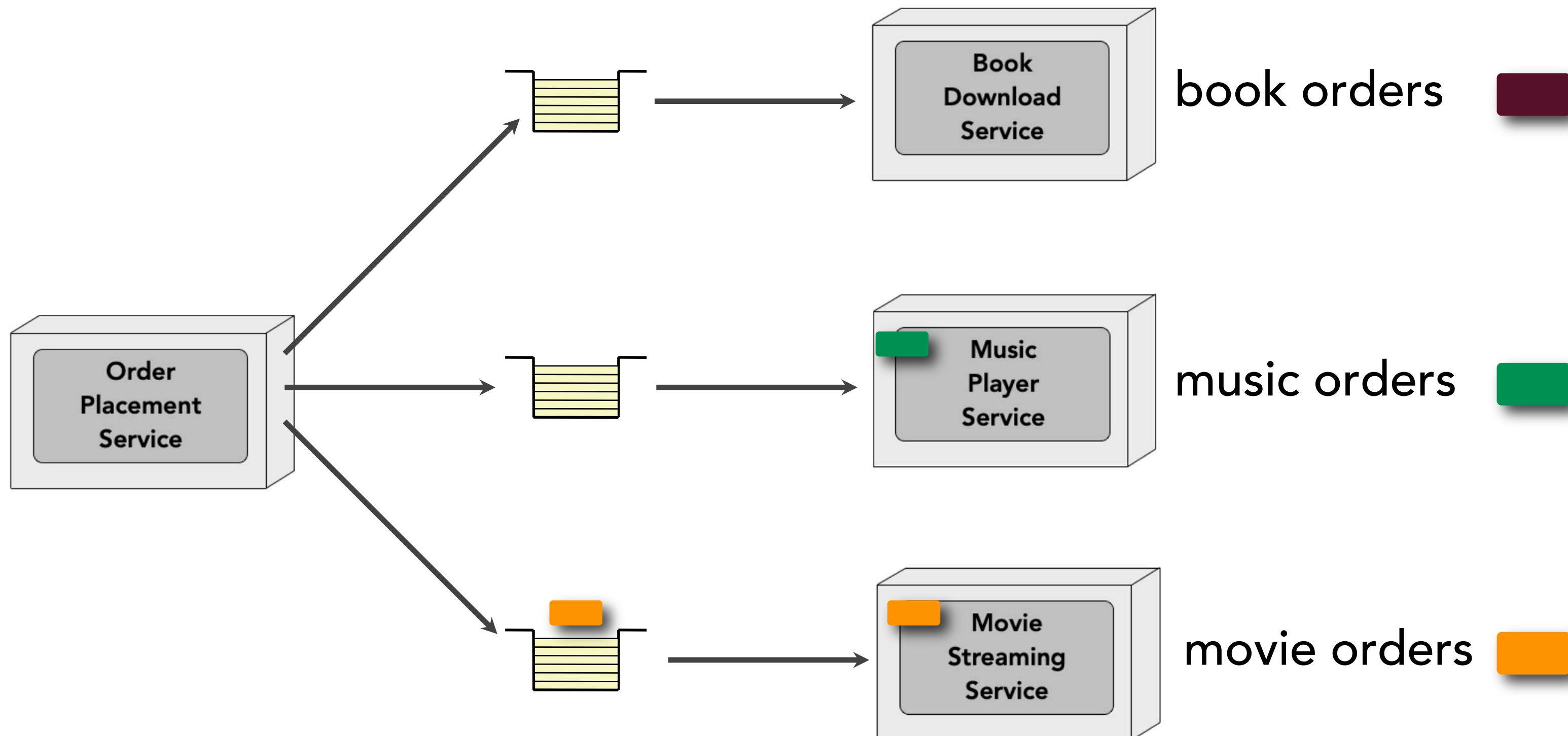
# contextual queue pattern

book orders, music orders, and movie orders are processed by different services



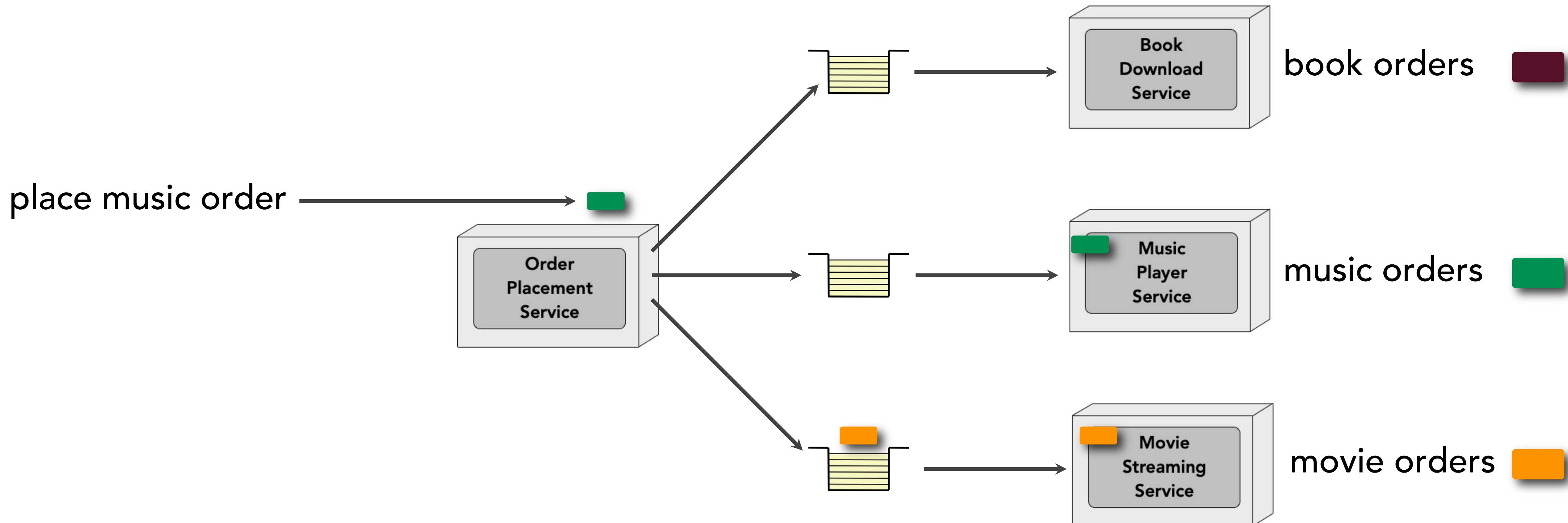
# contextual queue pattern

book orders, music orders, and movie orders are processed by different services



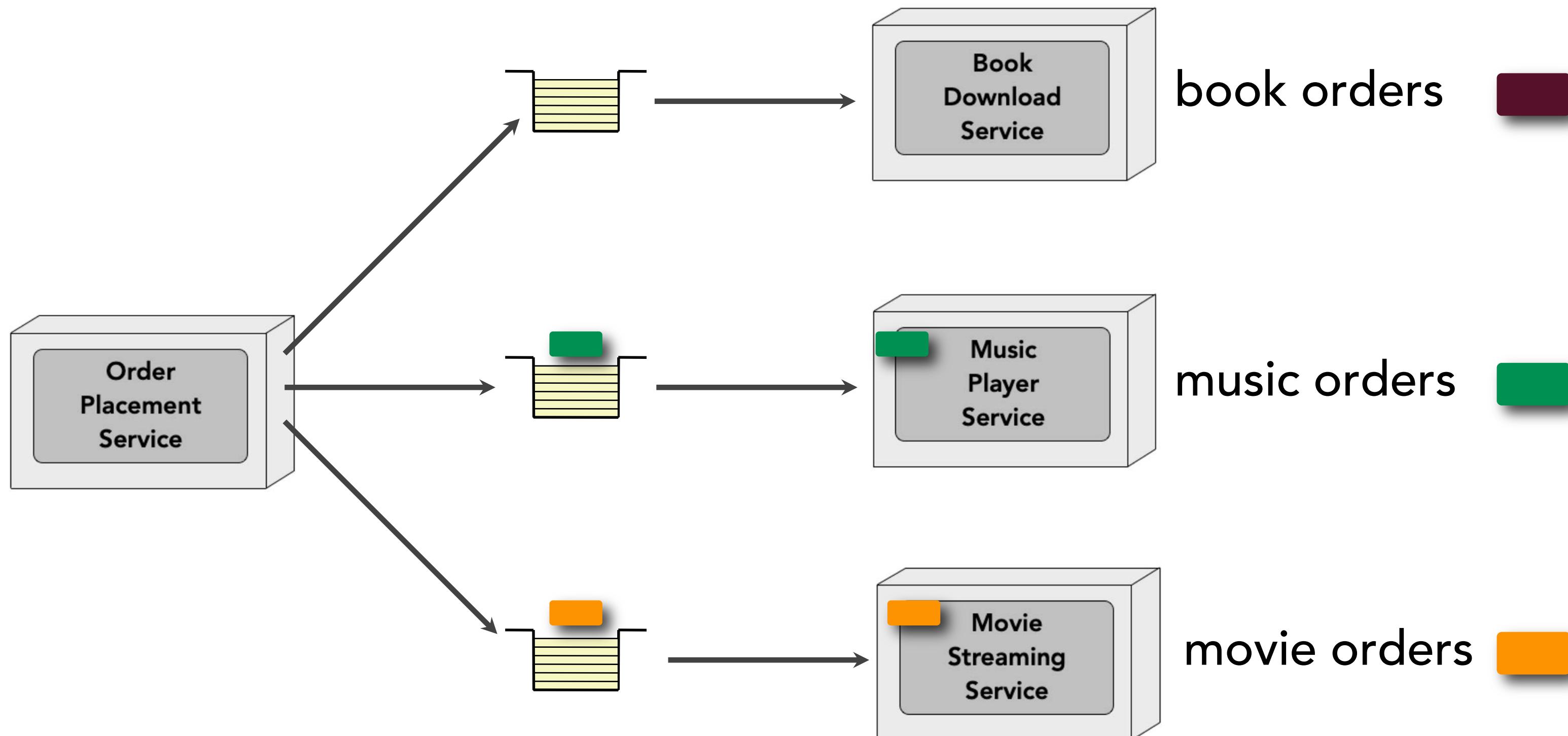
# contextual queue pattern

book orders, music orders, and movie orders are processed by different services



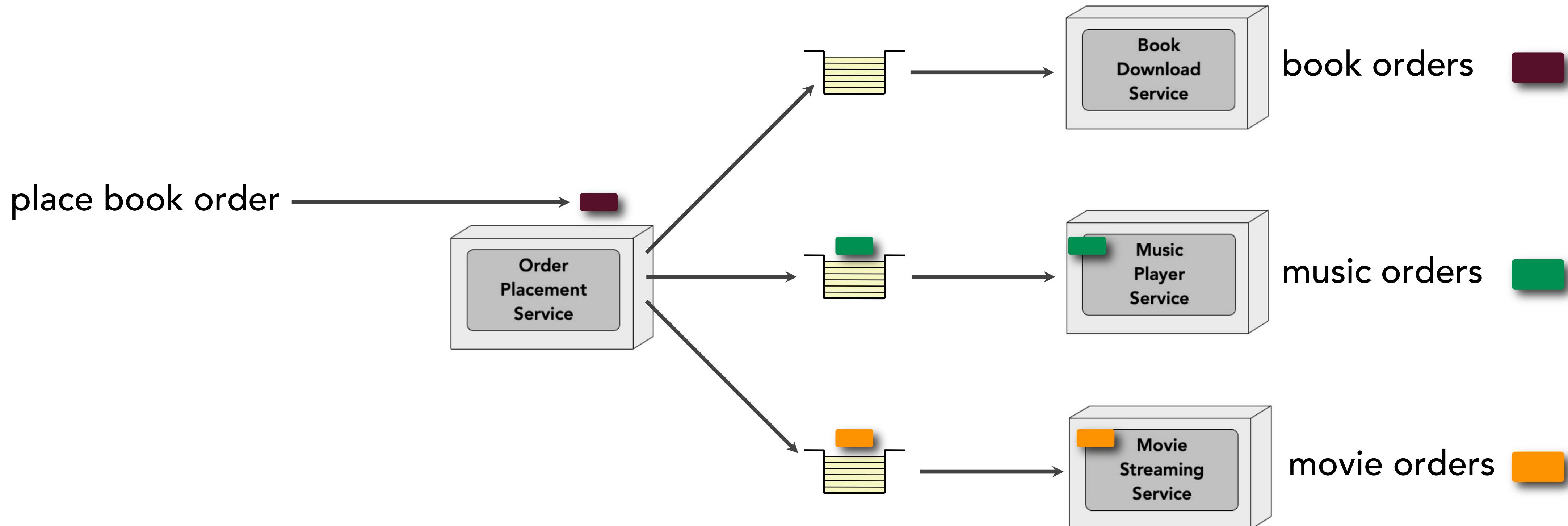
# contextual queue pattern

book orders, music orders, and movie orders are processed by different services



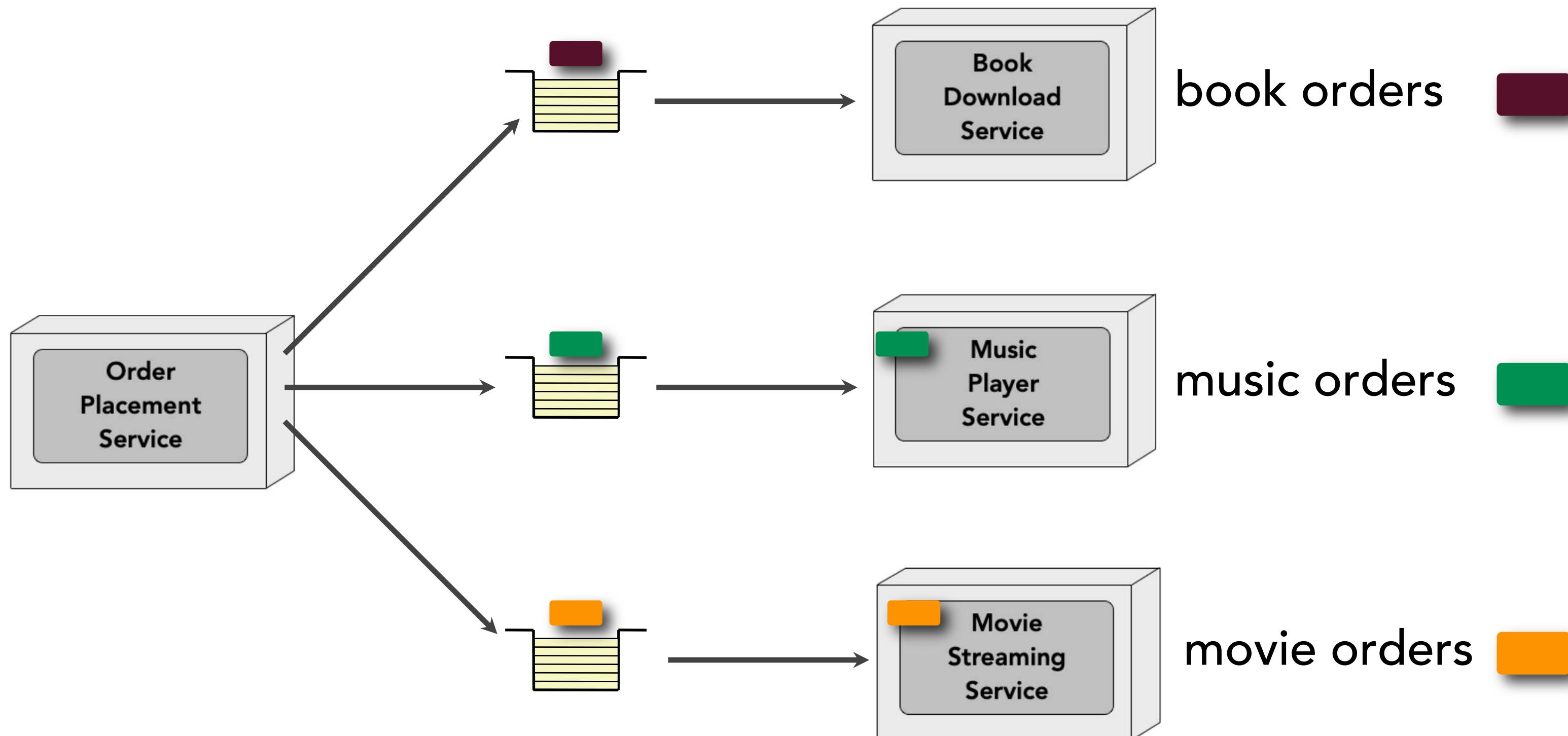
# contextual queue pattern

book orders, music orders, and movie orders are processed by different services



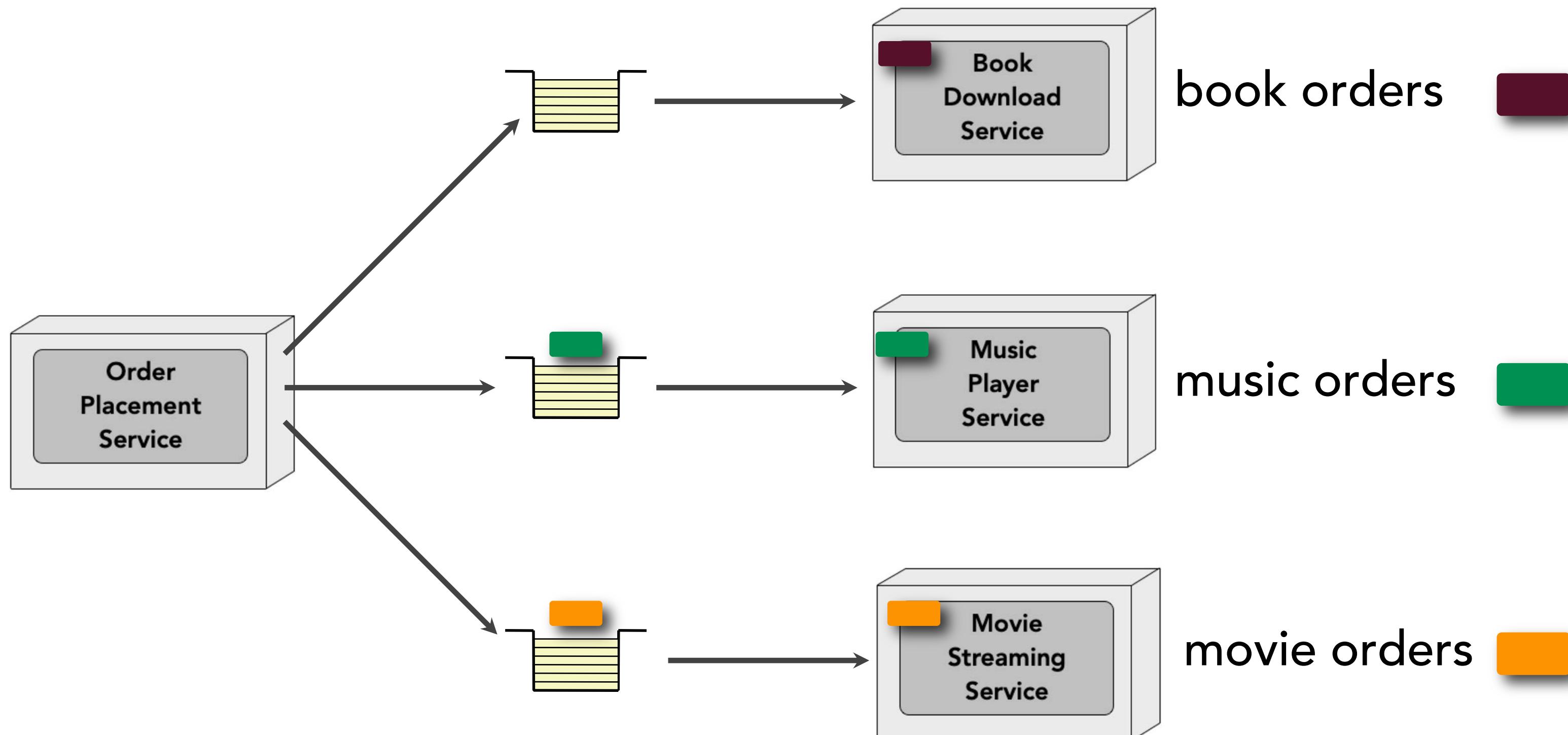
# contextual queue pattern

book orders, music orders, and movie orders are processed by different services



# contextual queue pattern

book orders, music orders, and movie orders are processed by different services



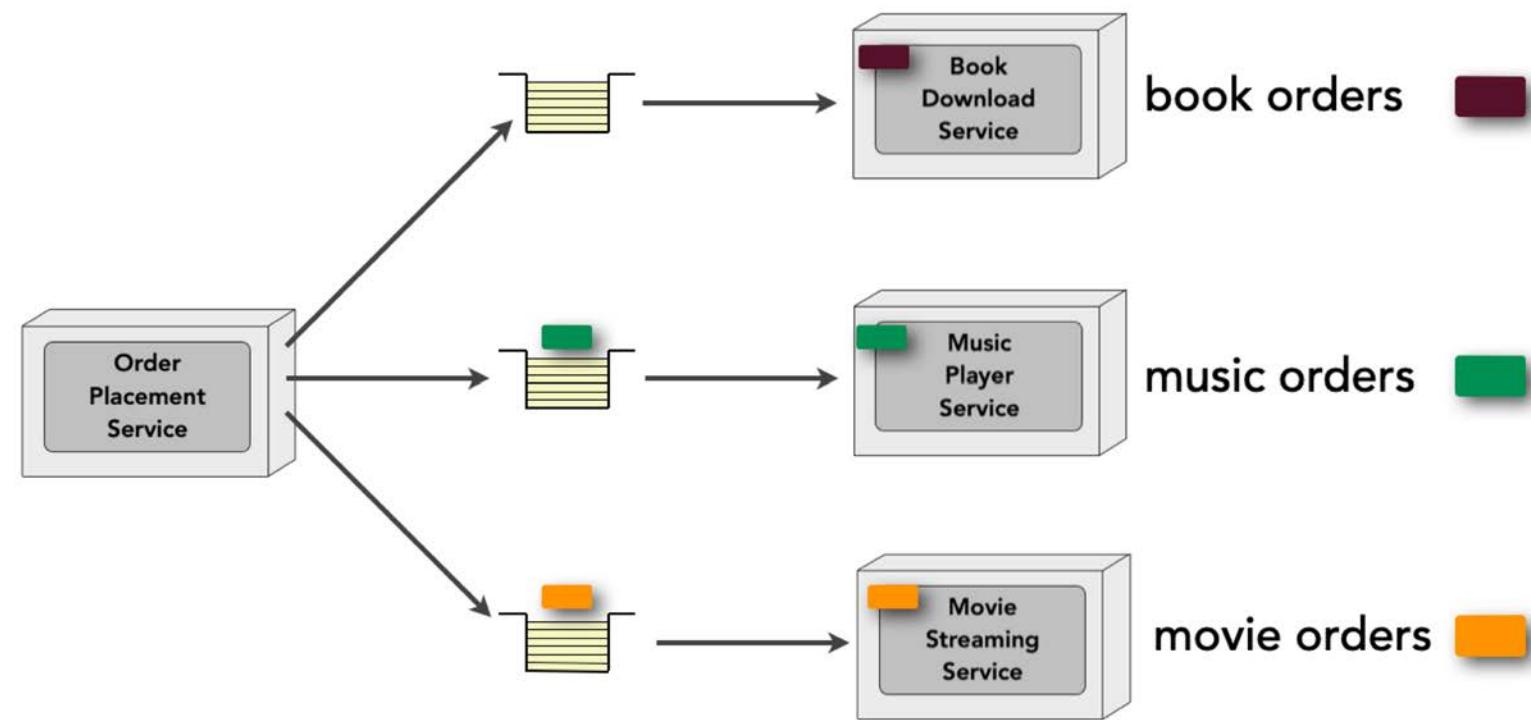
# contextual queue pattern



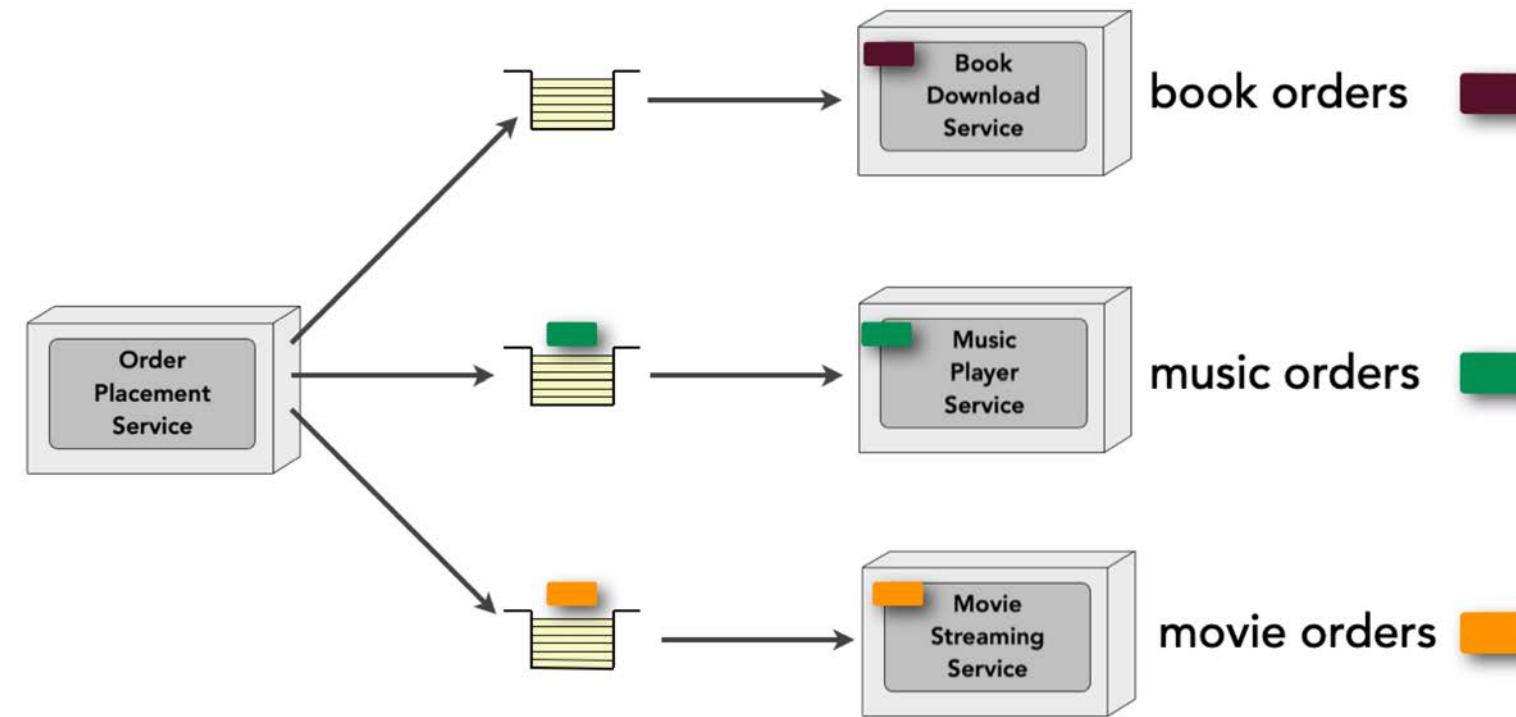
let's apply the pattern...

<https://github.com/wmr513/event-driven-patterns/tree/master/contextualqueue>

# contextual queue pattern



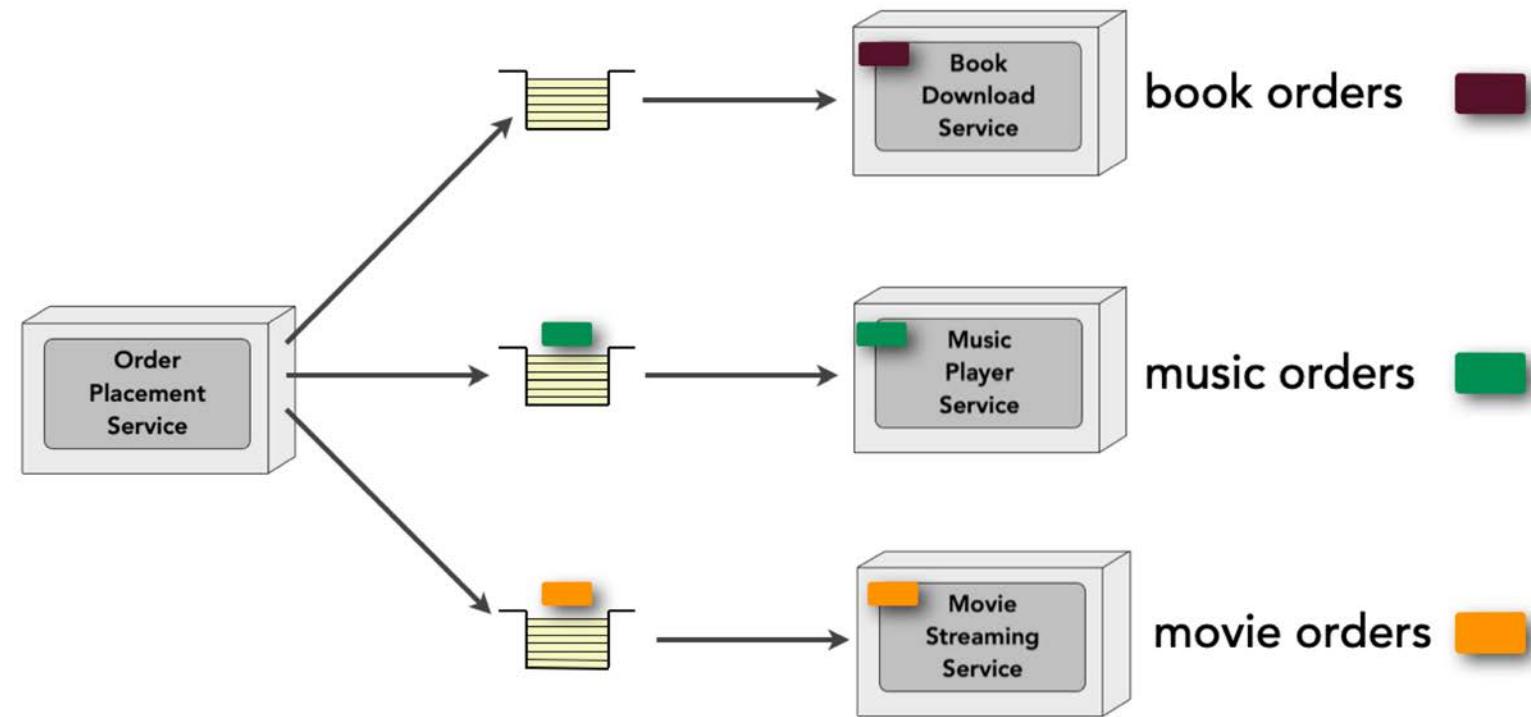
# contextual queue pattern



consistent performance  
increased throughput



# contextual queue pattern



consistent performance  
increased throughput

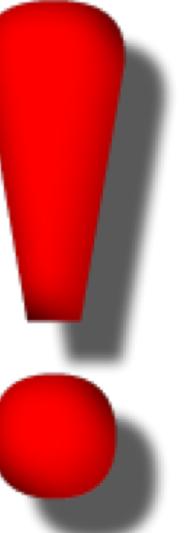
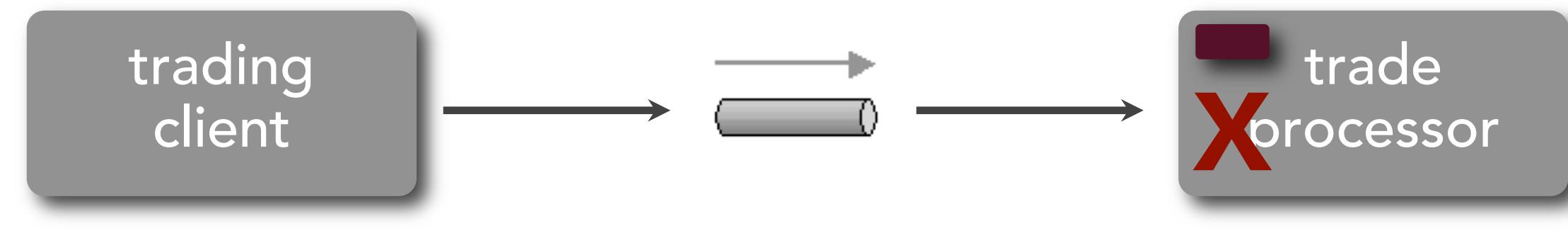


increased complexity  
producer knowledge

# Workflow Event Pattern

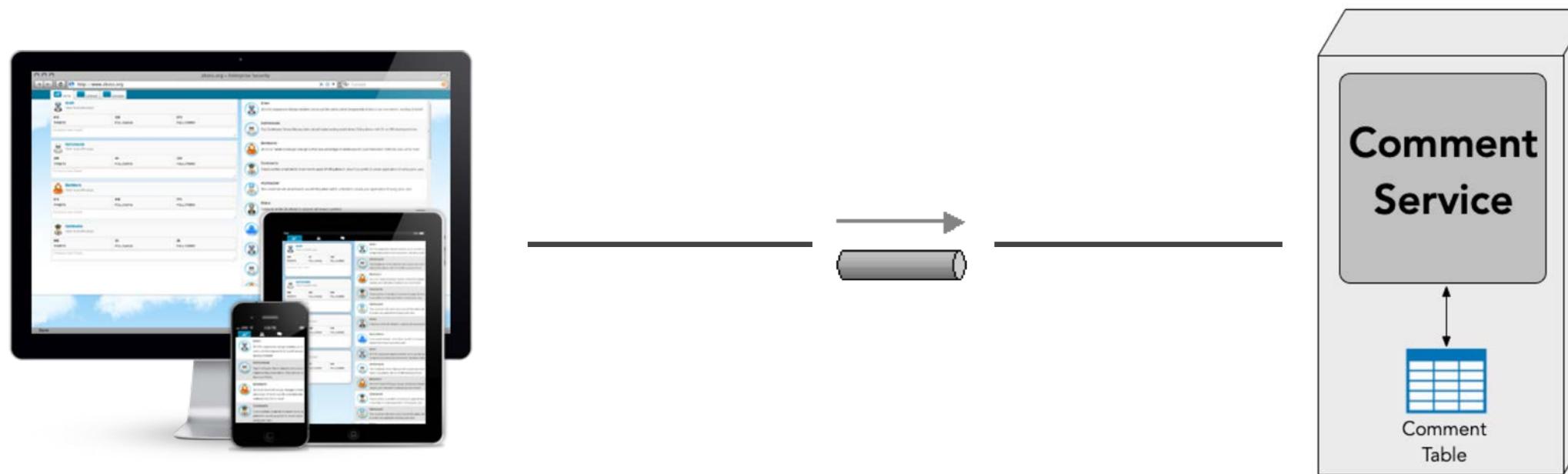
# workflow event pattern

*“how can I automatically fix error conditions when processing asynchronous events?”*



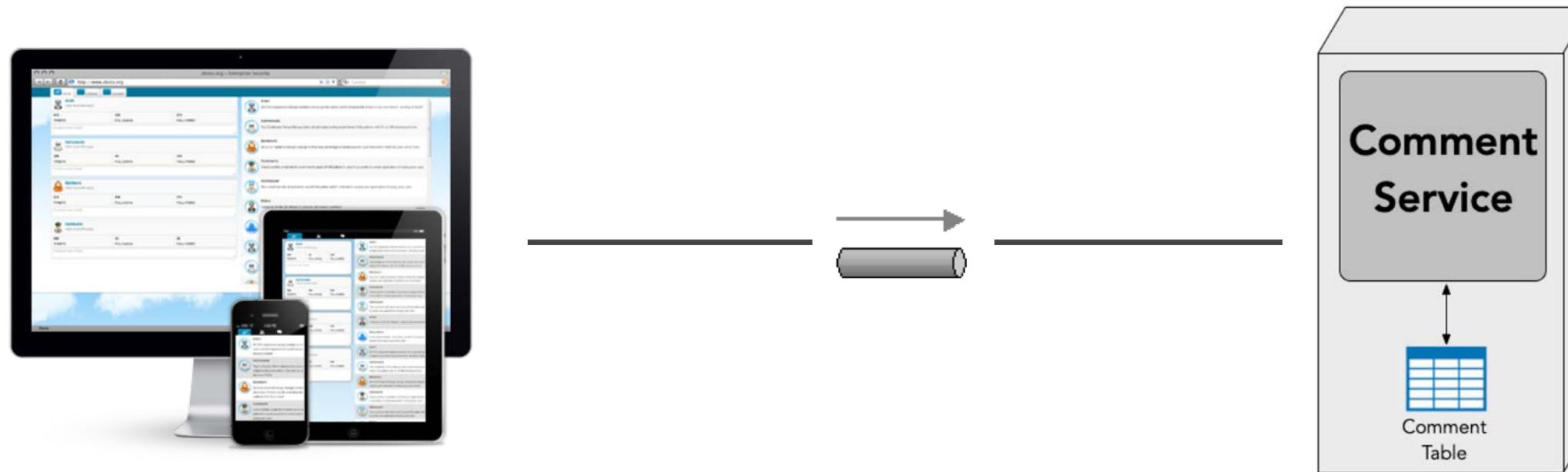
# workflow event pattern

## asynchronous vs. synchronous protocol



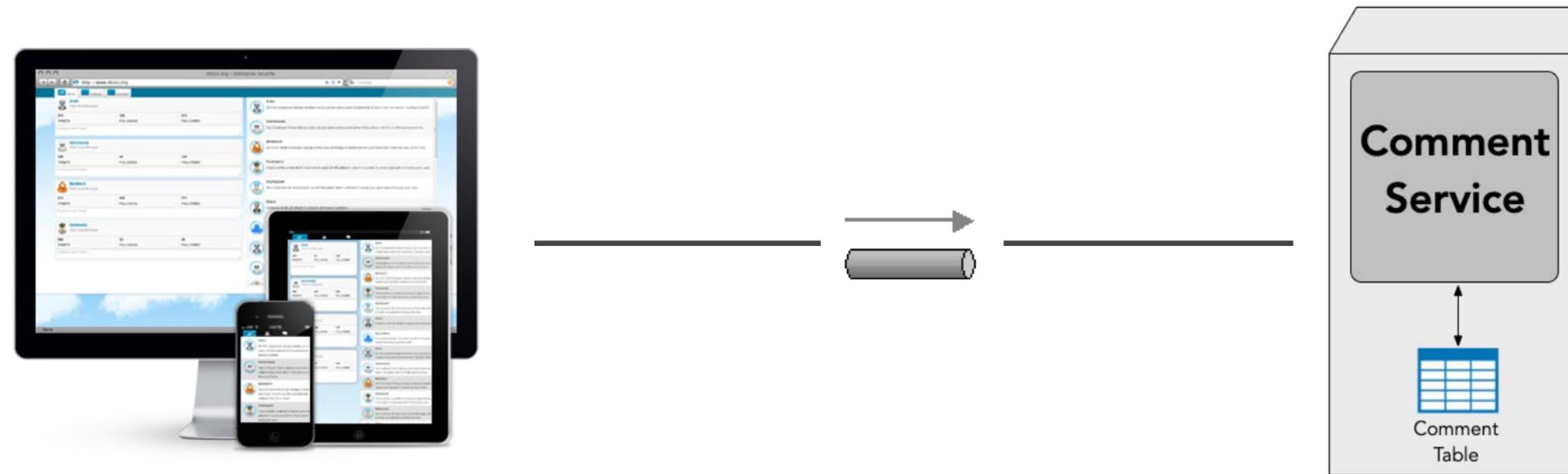
# workflow event pattern

## asynchronous vs. synchronous protocol



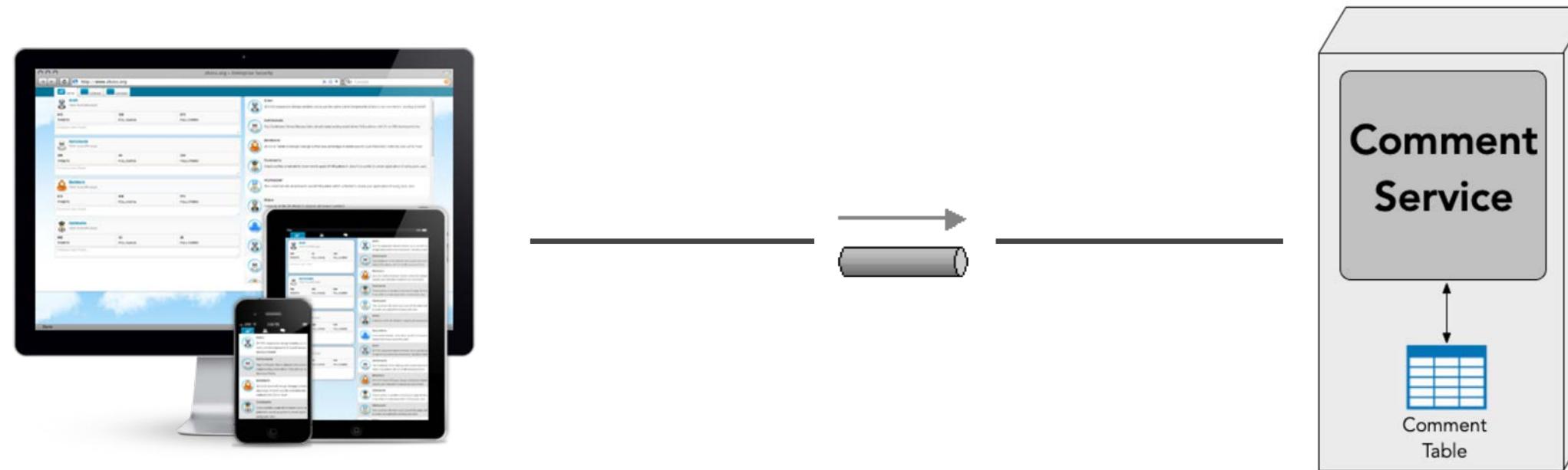
# workflow event pattern

## asynchronous vs. synchronous protocol



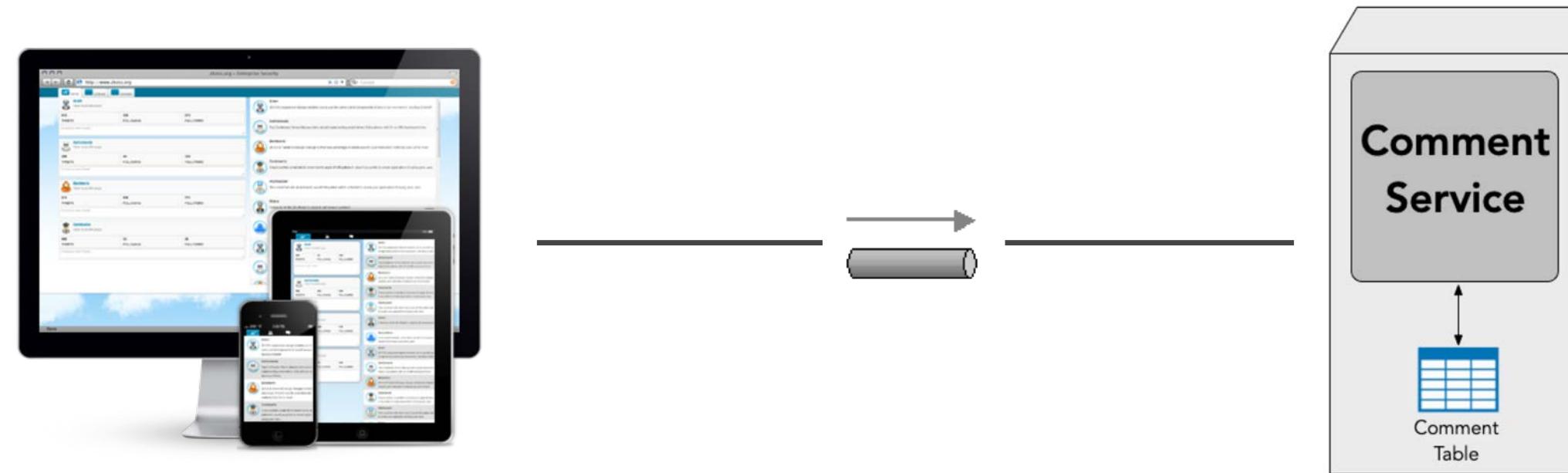
# workflow event pattern

## asynchronous vs. synchronous protocol



# workflow event pattern

## asynchronous vs. synchronous protocol



# workflow event pattern

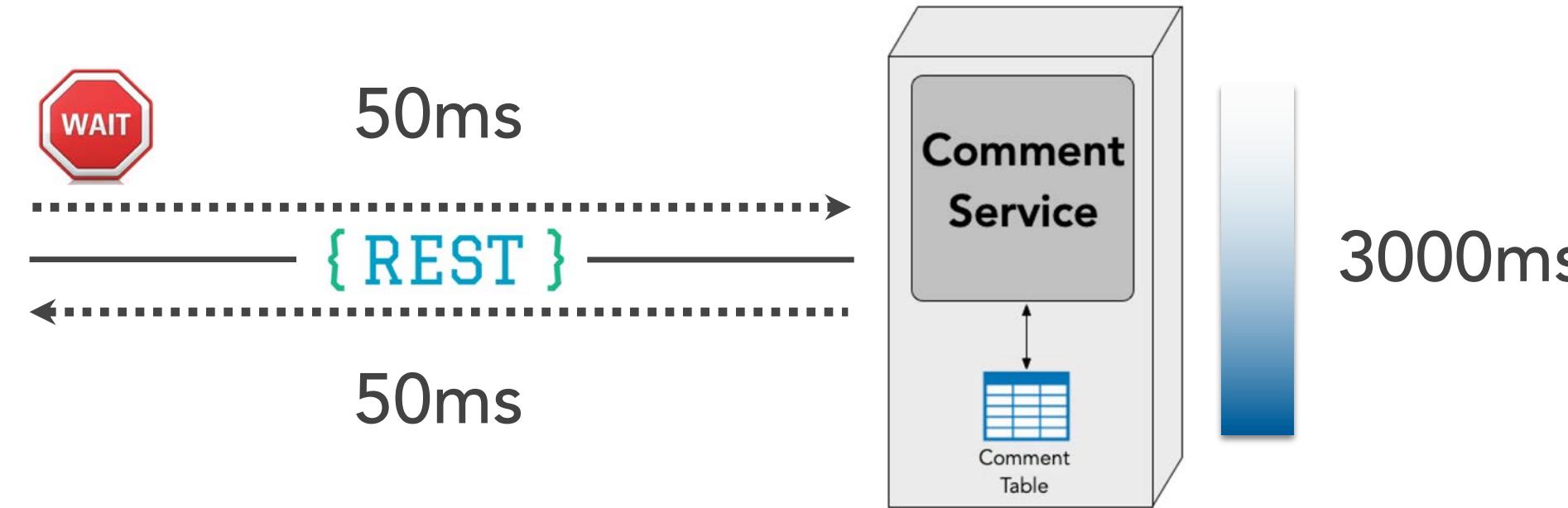
## asynchronous vs. synchronous protocol



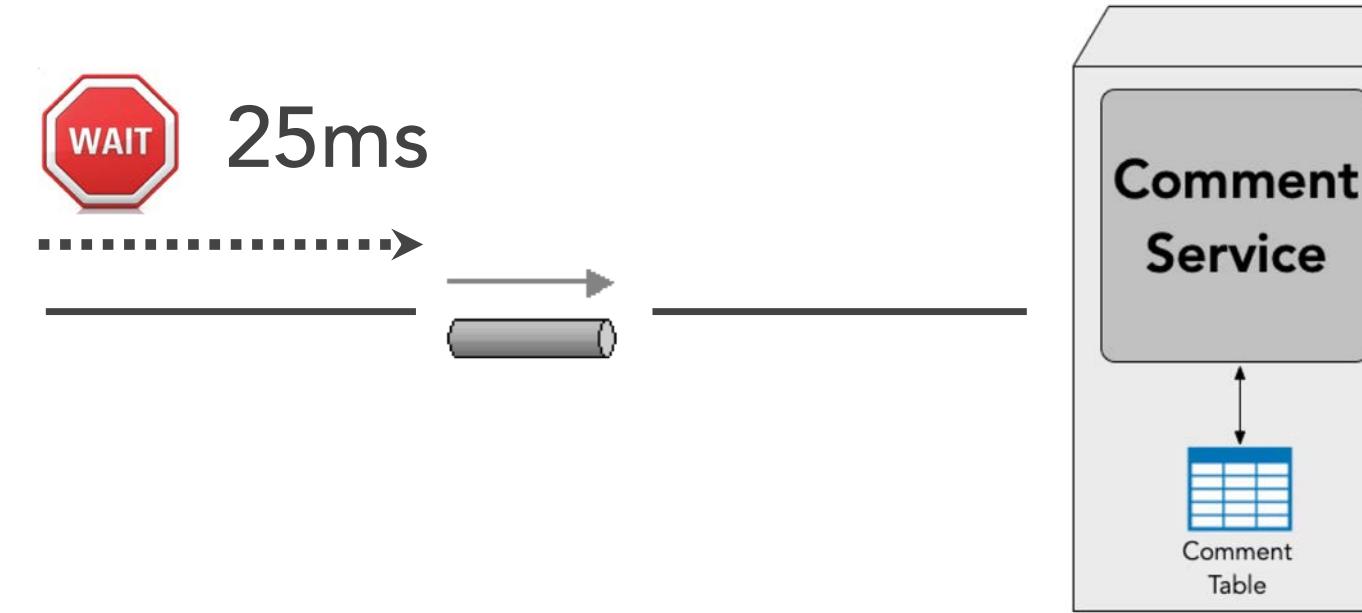
# workflow event pattern

## asynchronous vs. synchronous protocol

“post a comment”  
total = 3100ms



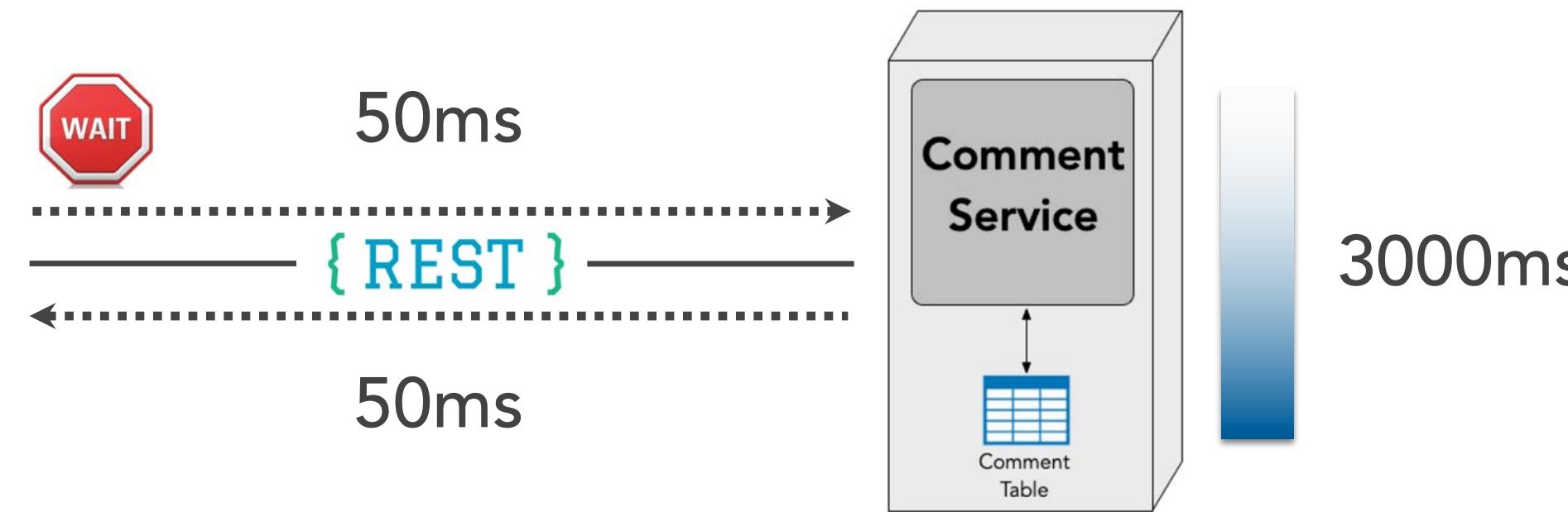
“post a comment”



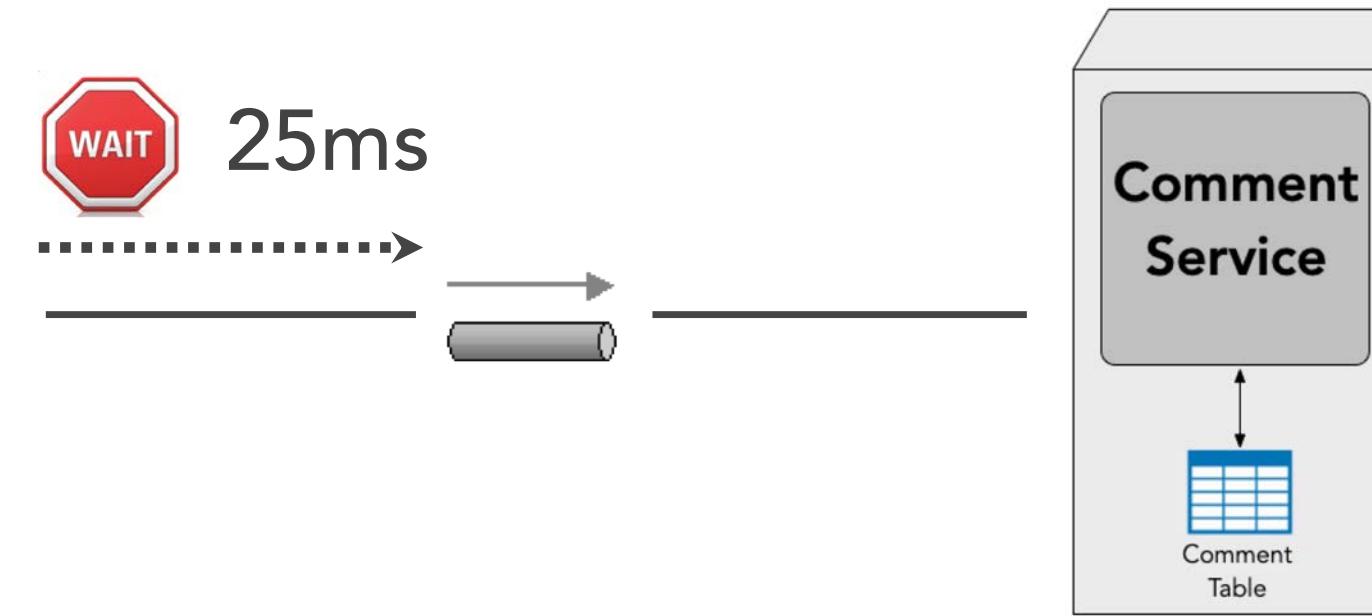
# workflow event pattern

## asynchronous vs. synchronous protocol

“post a comment”  
total = 3100ms



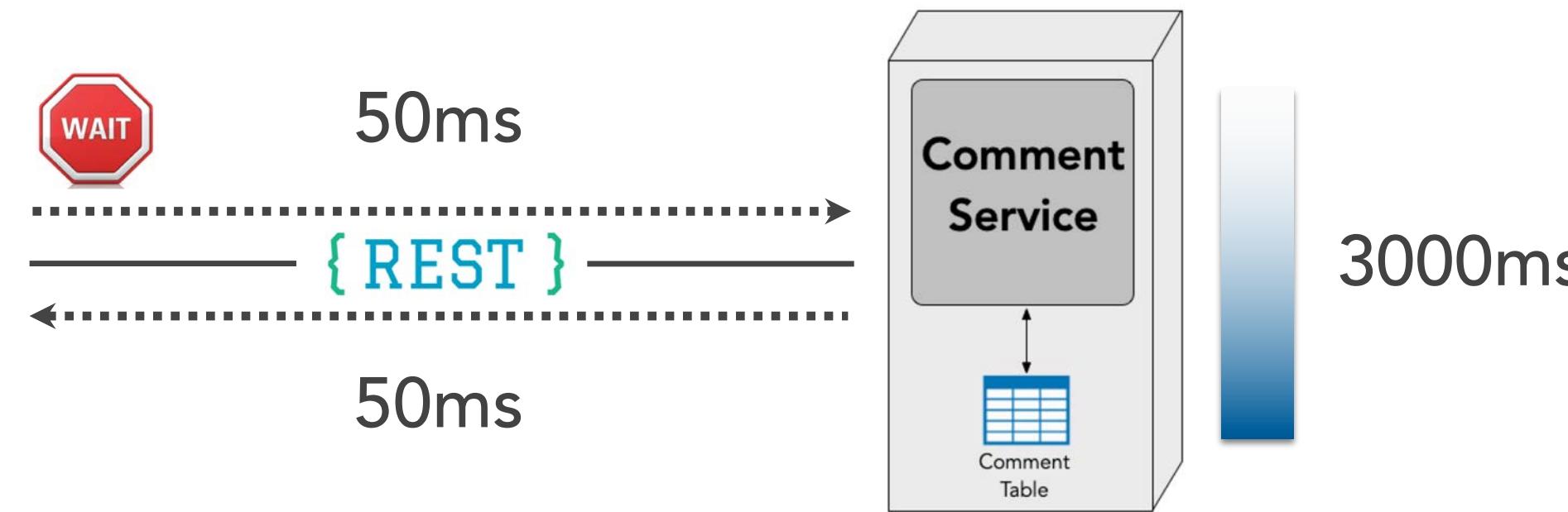
“post a comment”  
total = 25ms



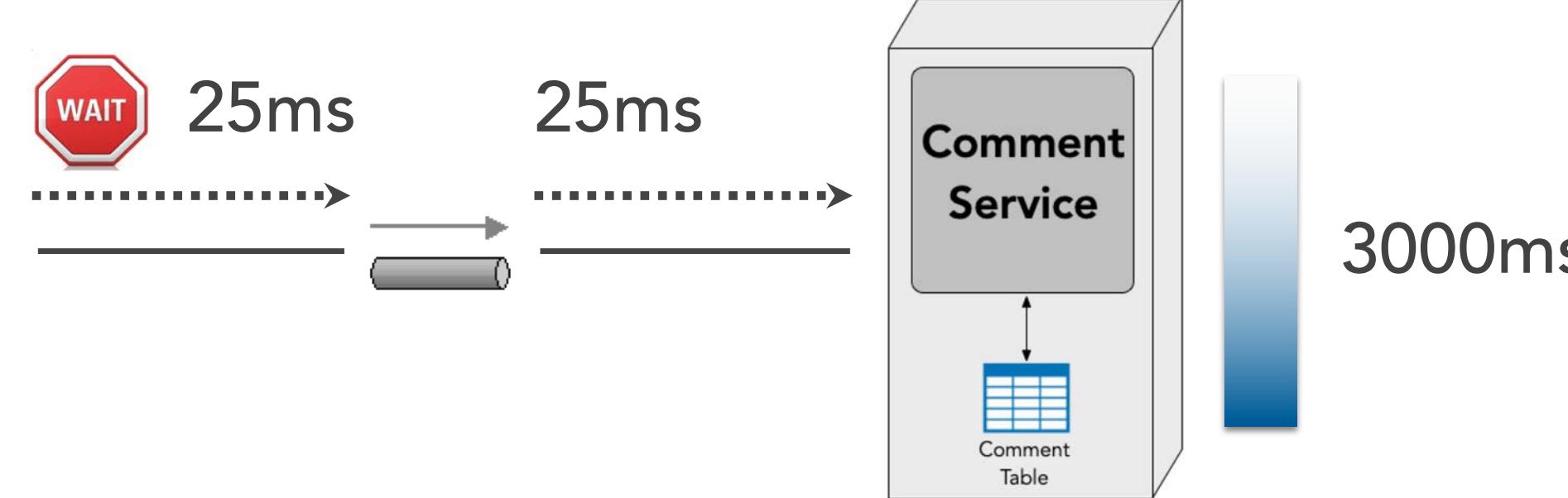
# workflow event pattern

## asynchronous vs. synchronous protocol

“post a comment”  
total = 3100ms

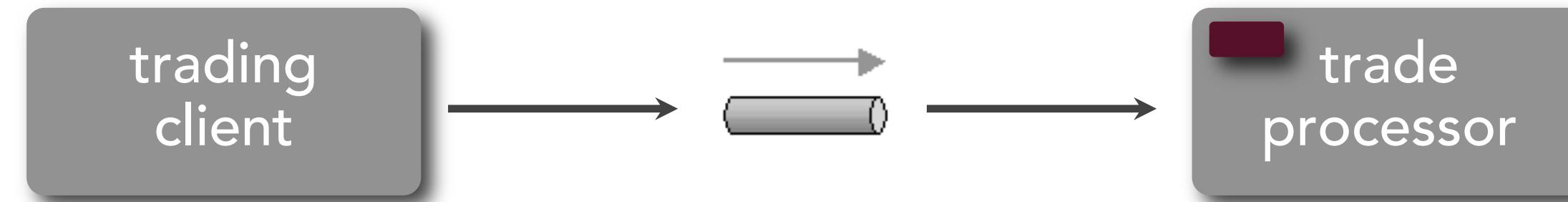


“post a comment”  
total = 25ms



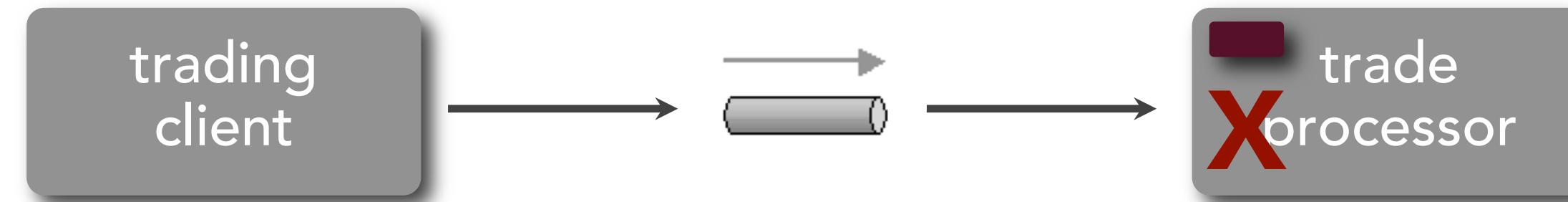
# workflow event pattern

while asynchronously processing trades an error occurs with one of the trade orders



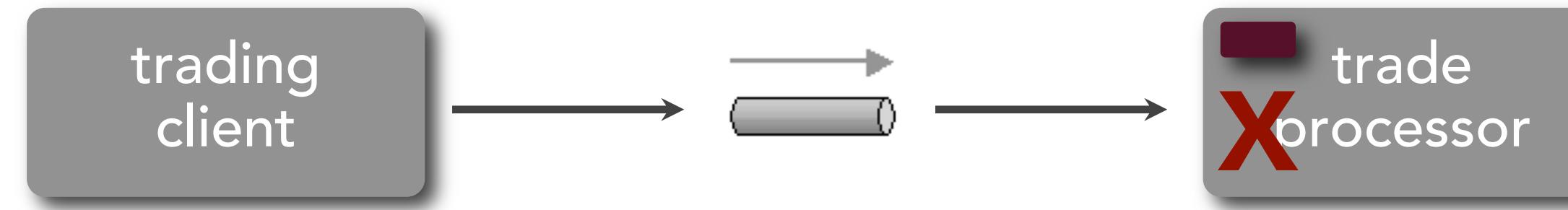
# workflow event pattern

while asynchronously processing trades an error occurs with one of the trade orders



# workflow event pattern

while asynchronously processing trades an error occurs with one of the trade orders



# workflow event pattern

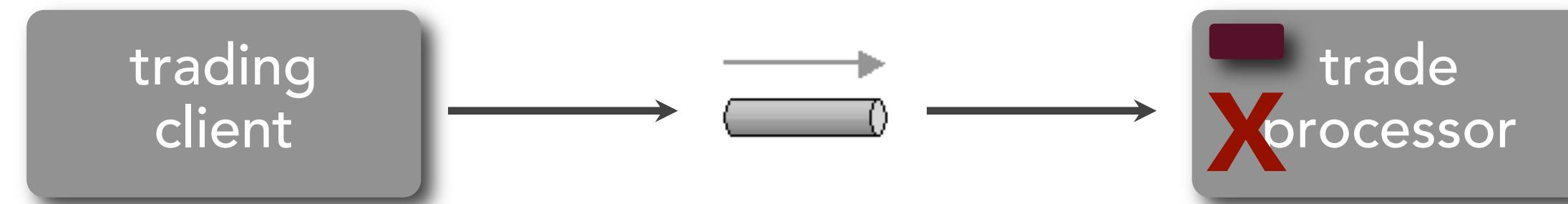


let's see the issue...

<https://github.com/wmr513/reactive/tree/master/workflow>

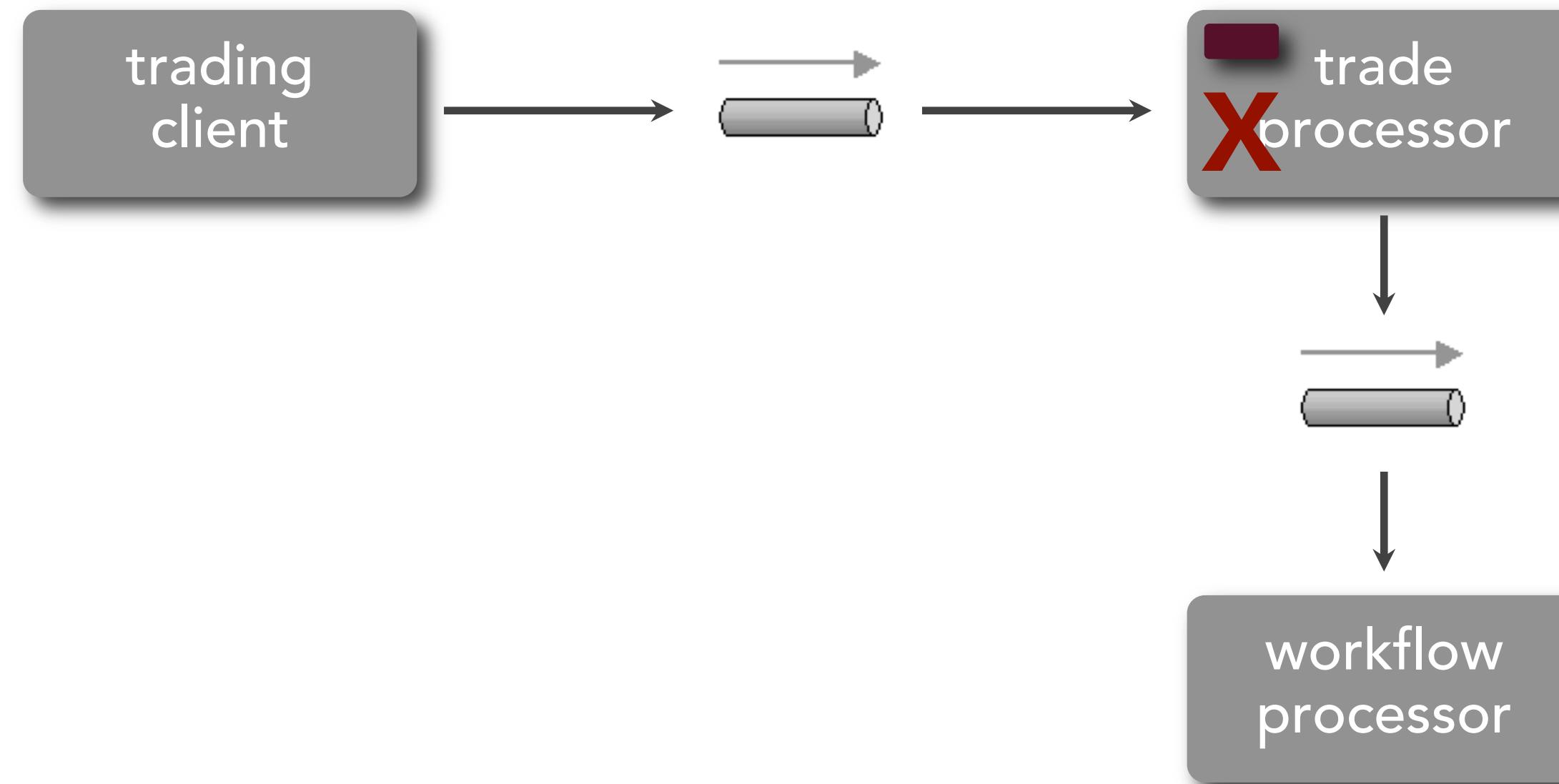
# workflow event pattern

while asynchronously processing trades an error occurs with one of the trade orders



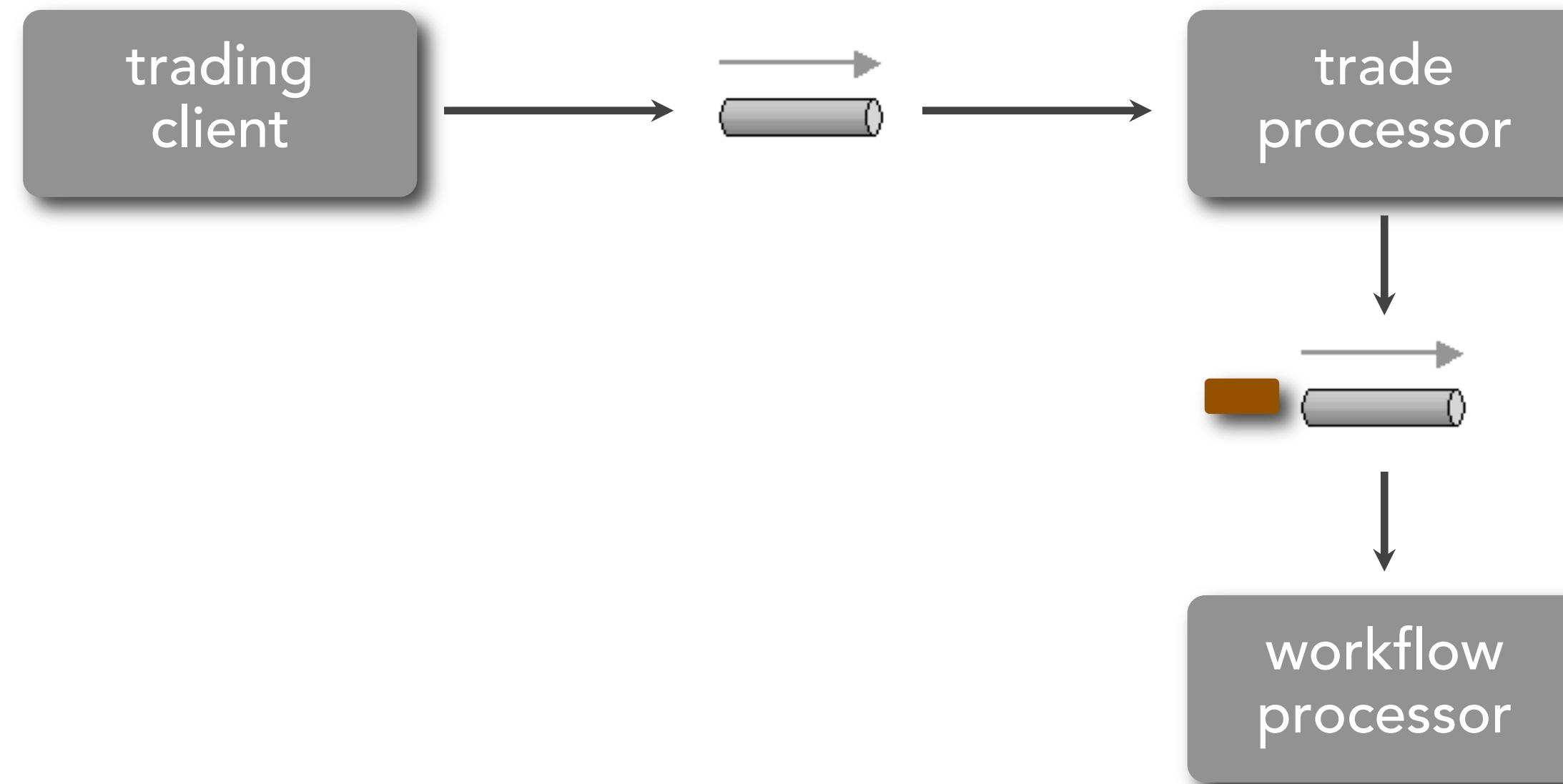
# workflow event pattern

while asynchronously processing trades an error occurs with one of the trade orders



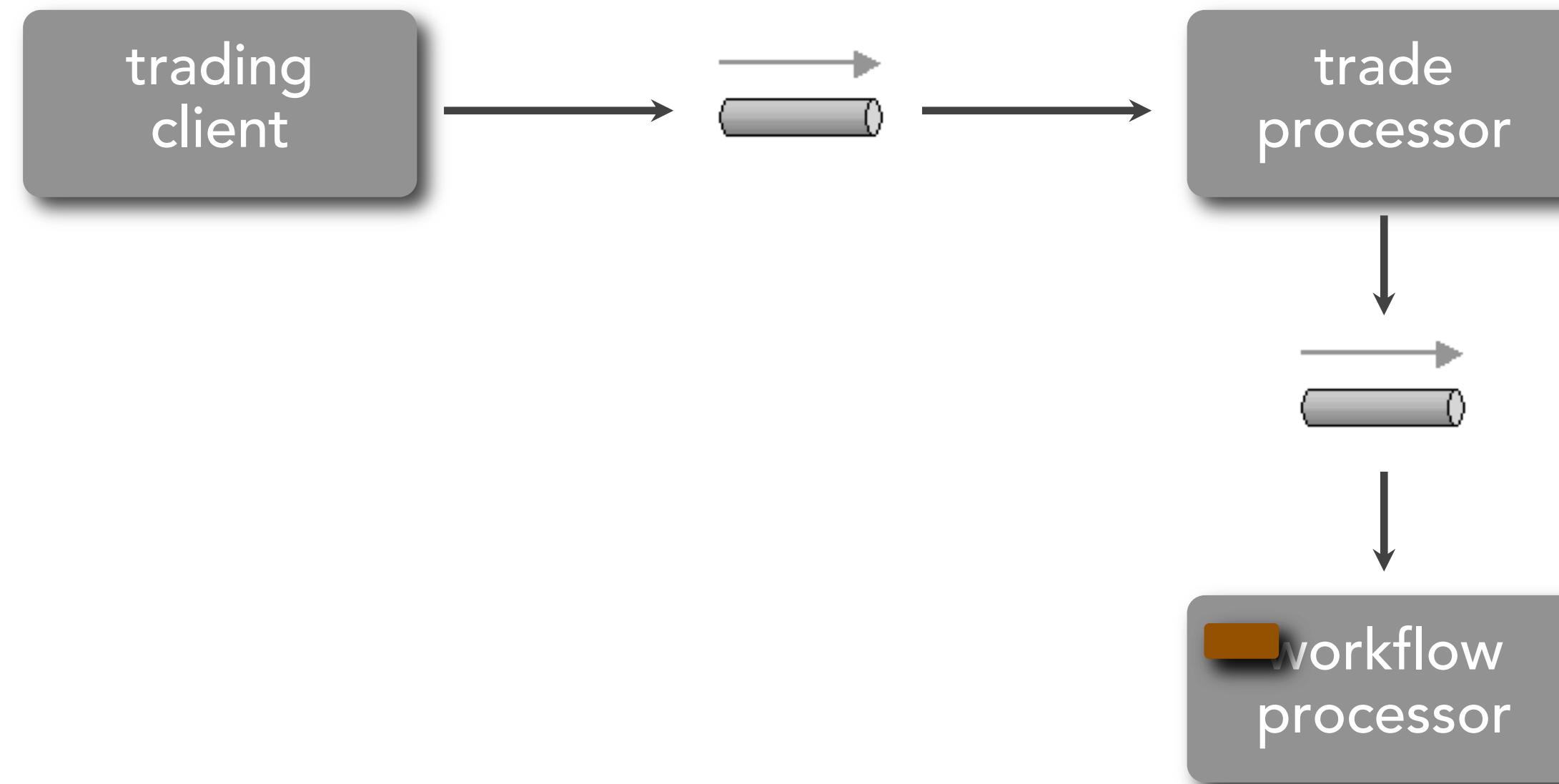
# workflow event pattern

while asynchronously processing trades an error occurs with one of the trade orders



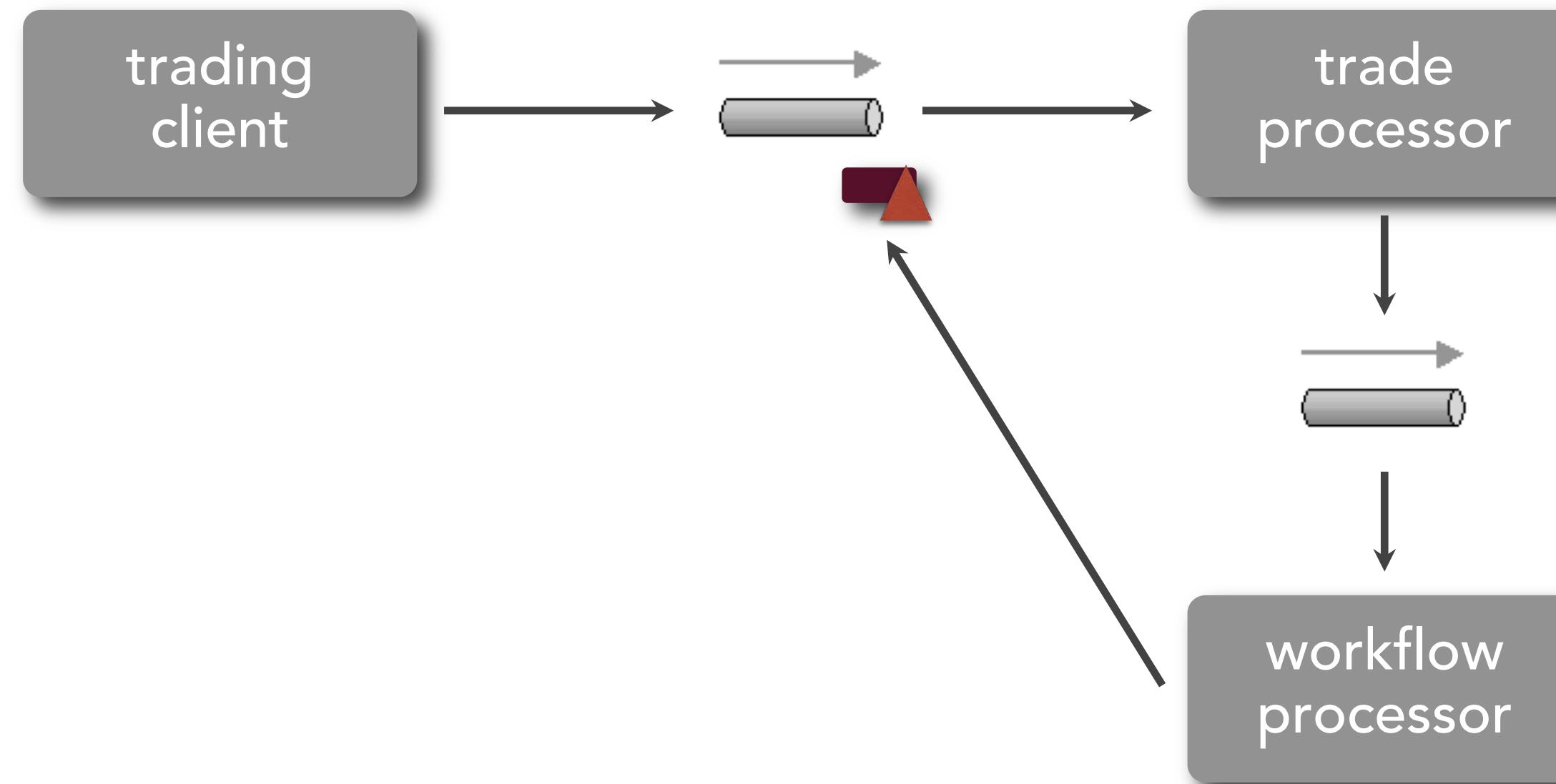
# workflow event pattern

while asynchronously processing trades an error occurs with one of the trade orders



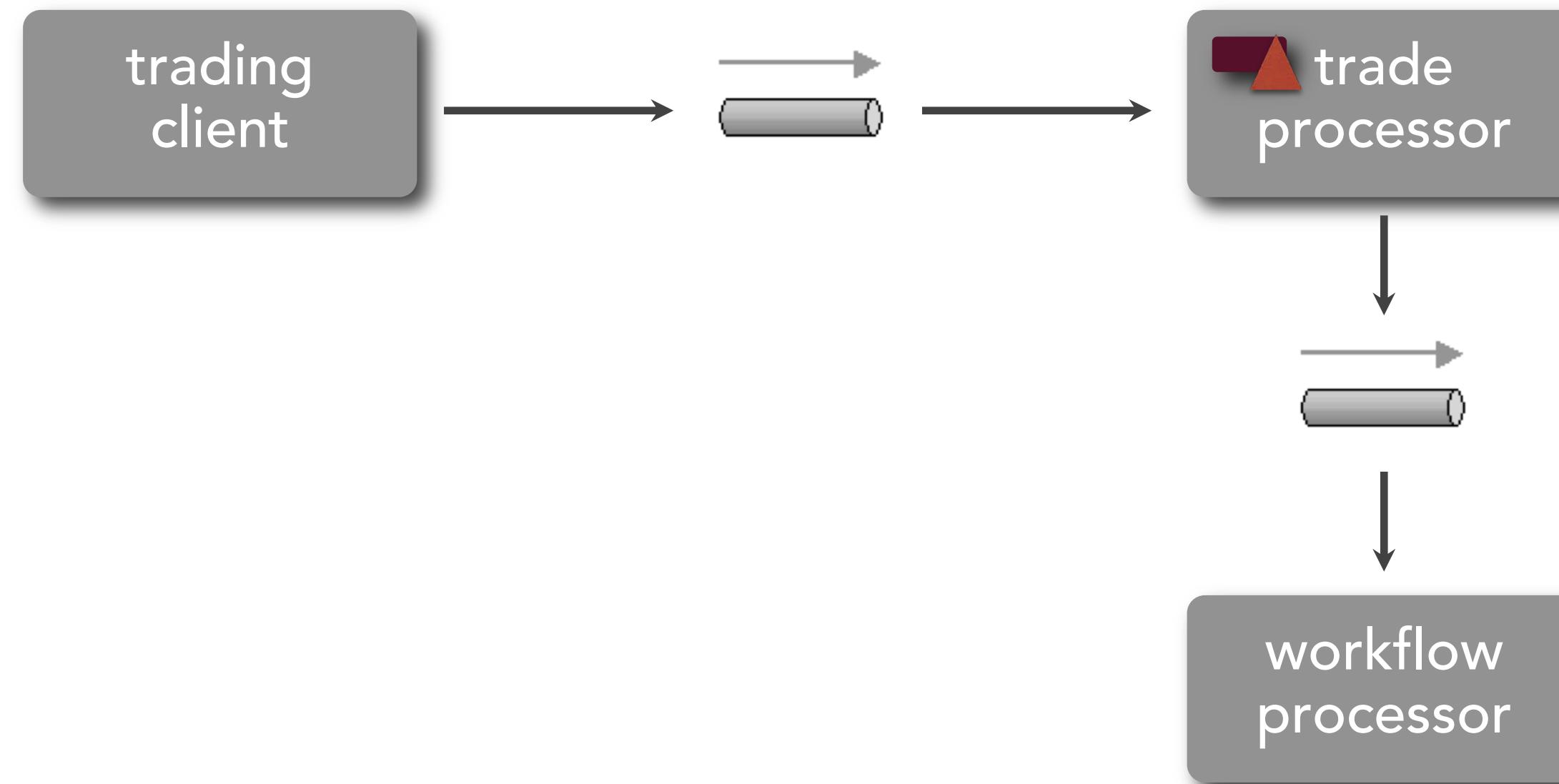
# workflow event pattern

while asynchronously processing trades an error occurs with one of the trade orders



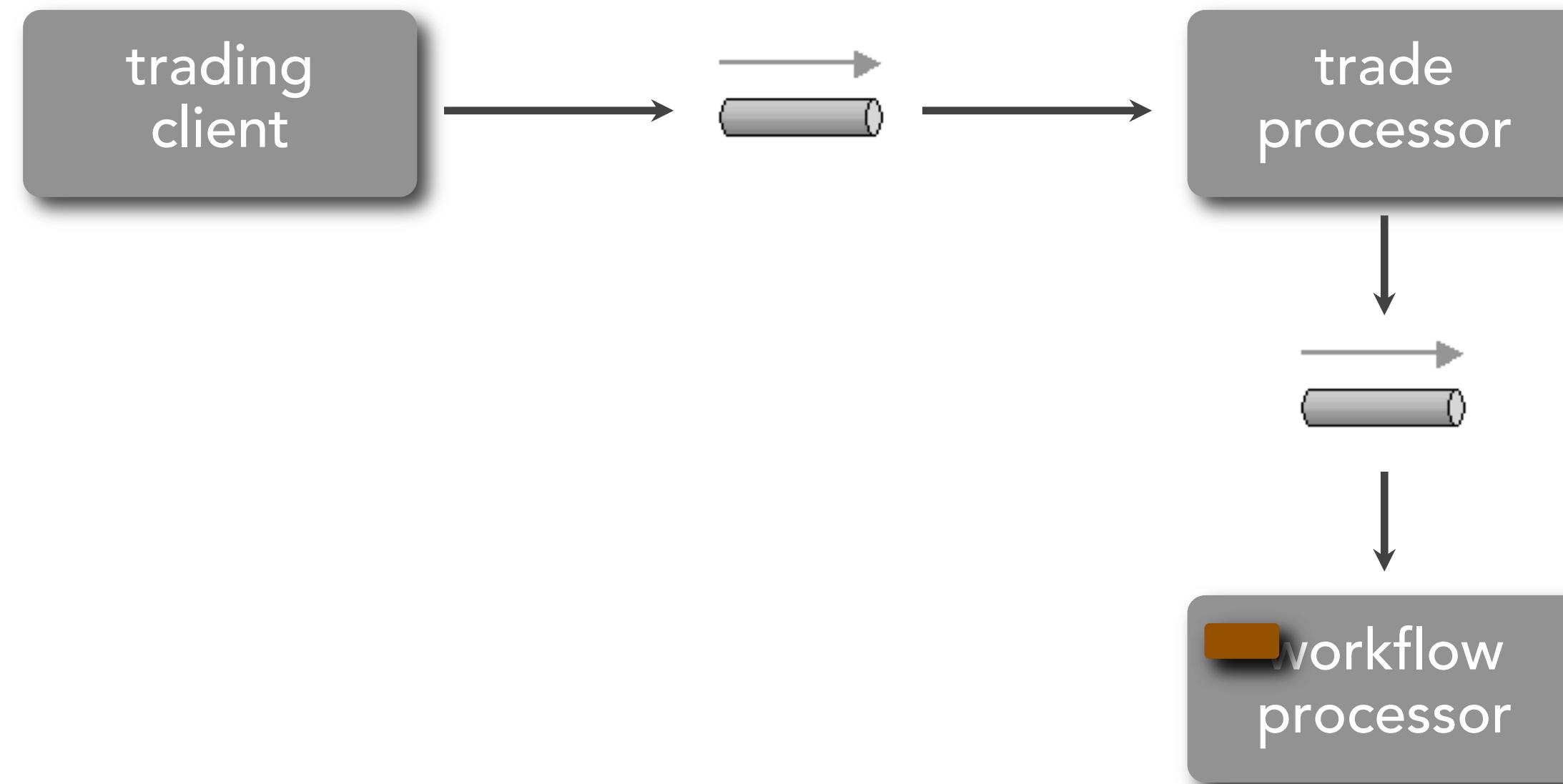
# workflow event pattern

while asynchronously processing trades an error occurs with one of the trade orders



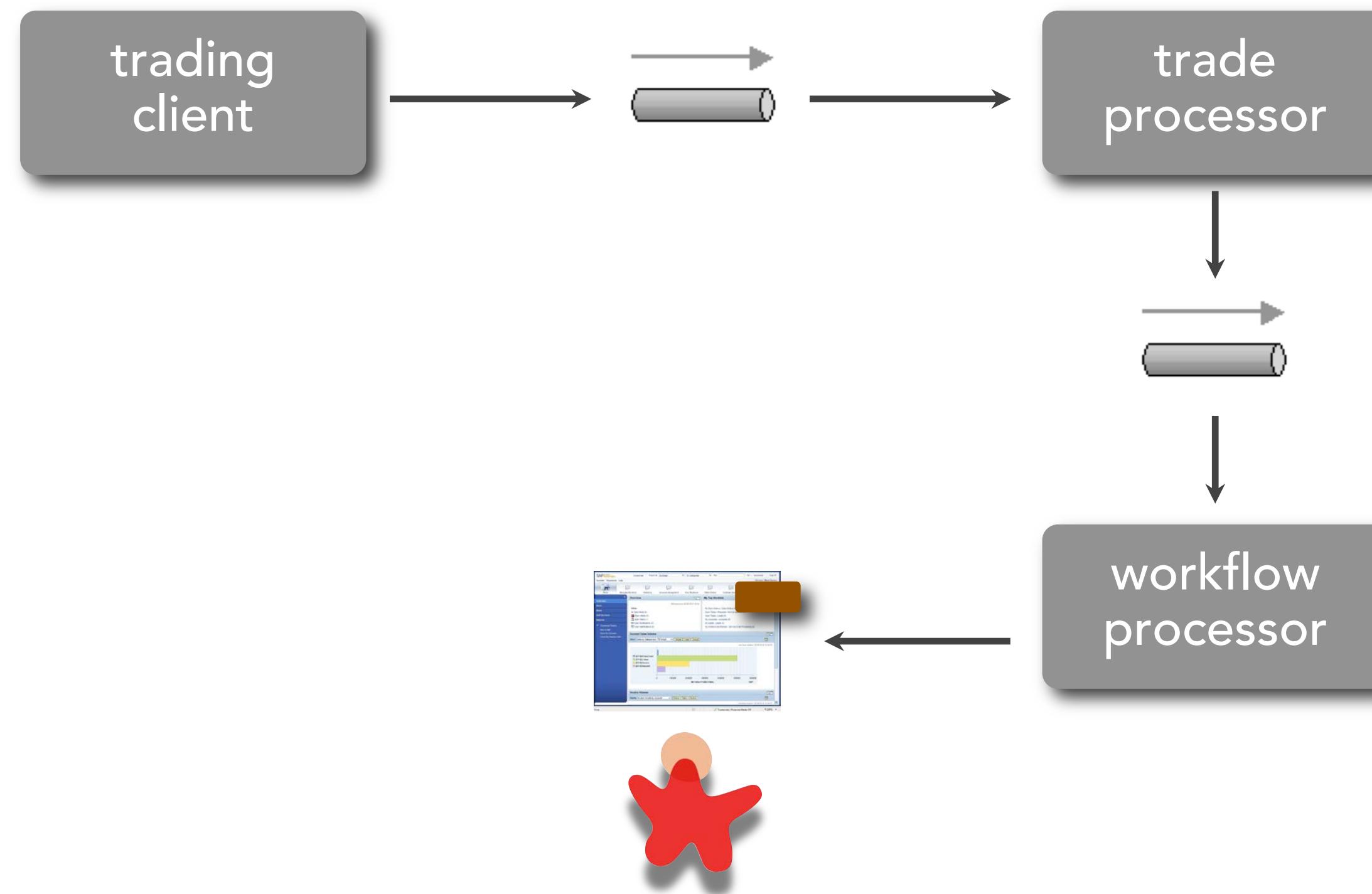
# workflow event pattern

while asynchronously processing trades an error occurs with one of the trade orders



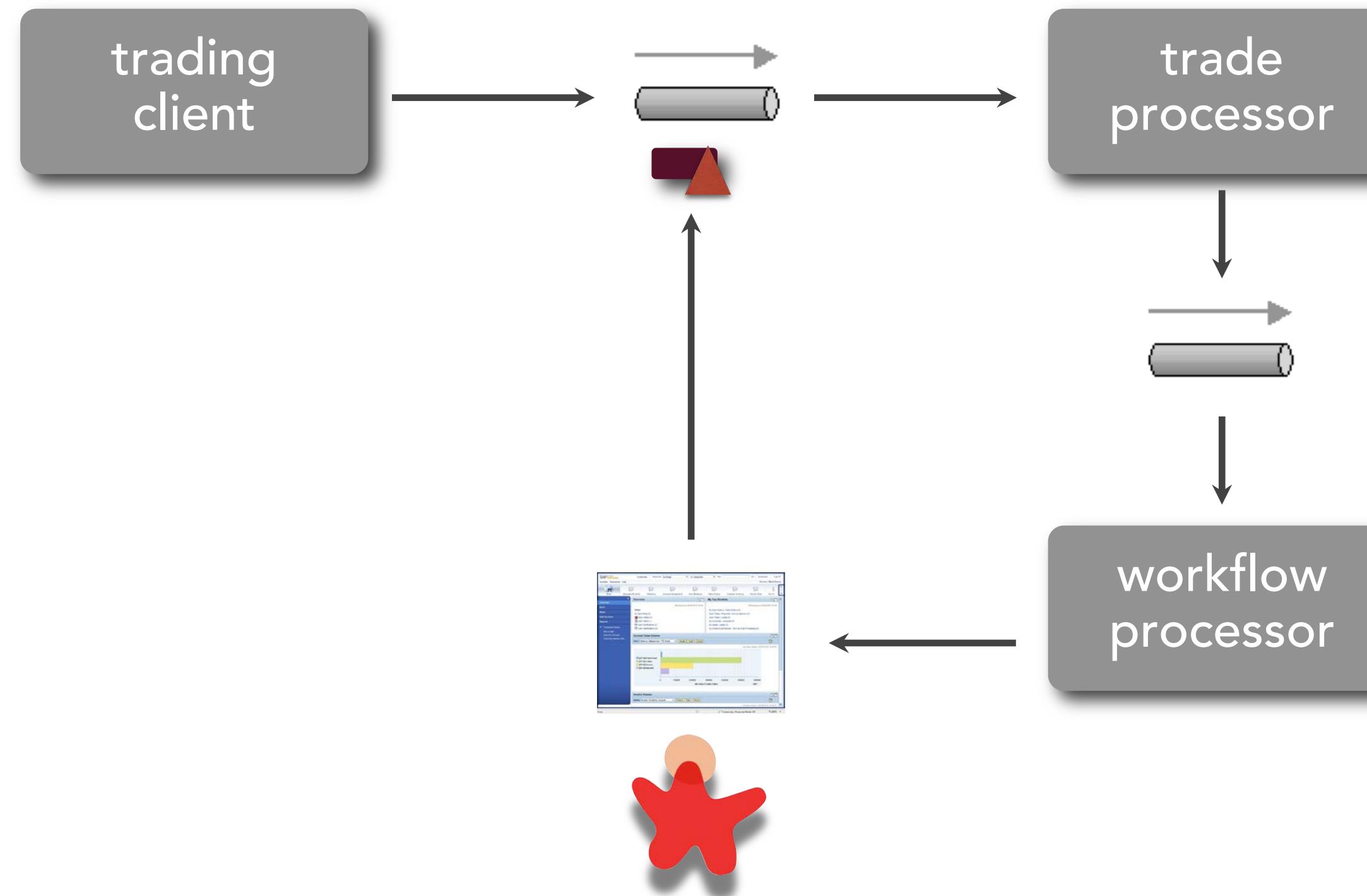
# workflow event pattern

while asynchronously processing trades an error occurs with one of the trade orders



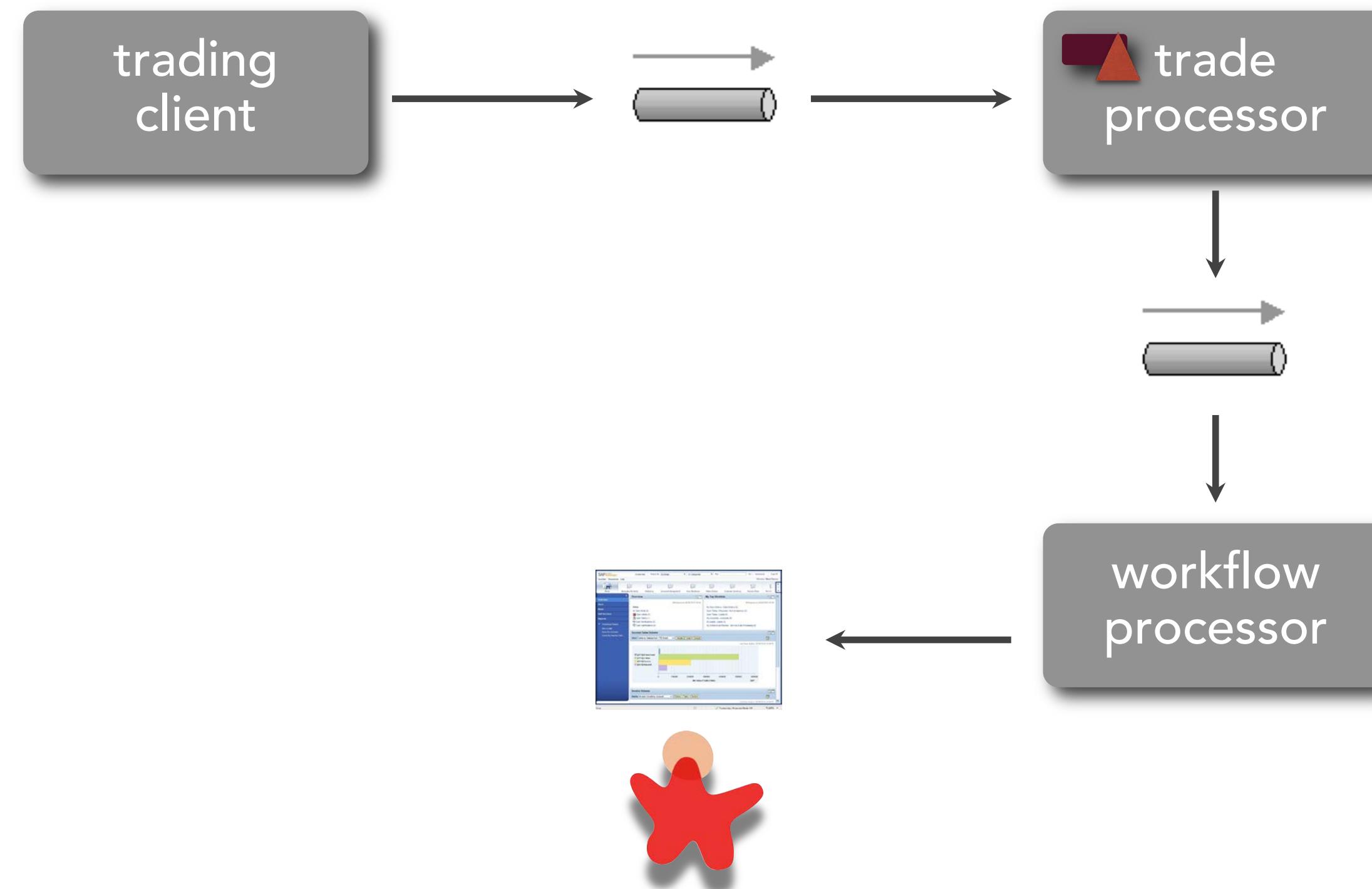
# workflow event pattern

while asynchronously processing trades an error occurs with one of the trade orders



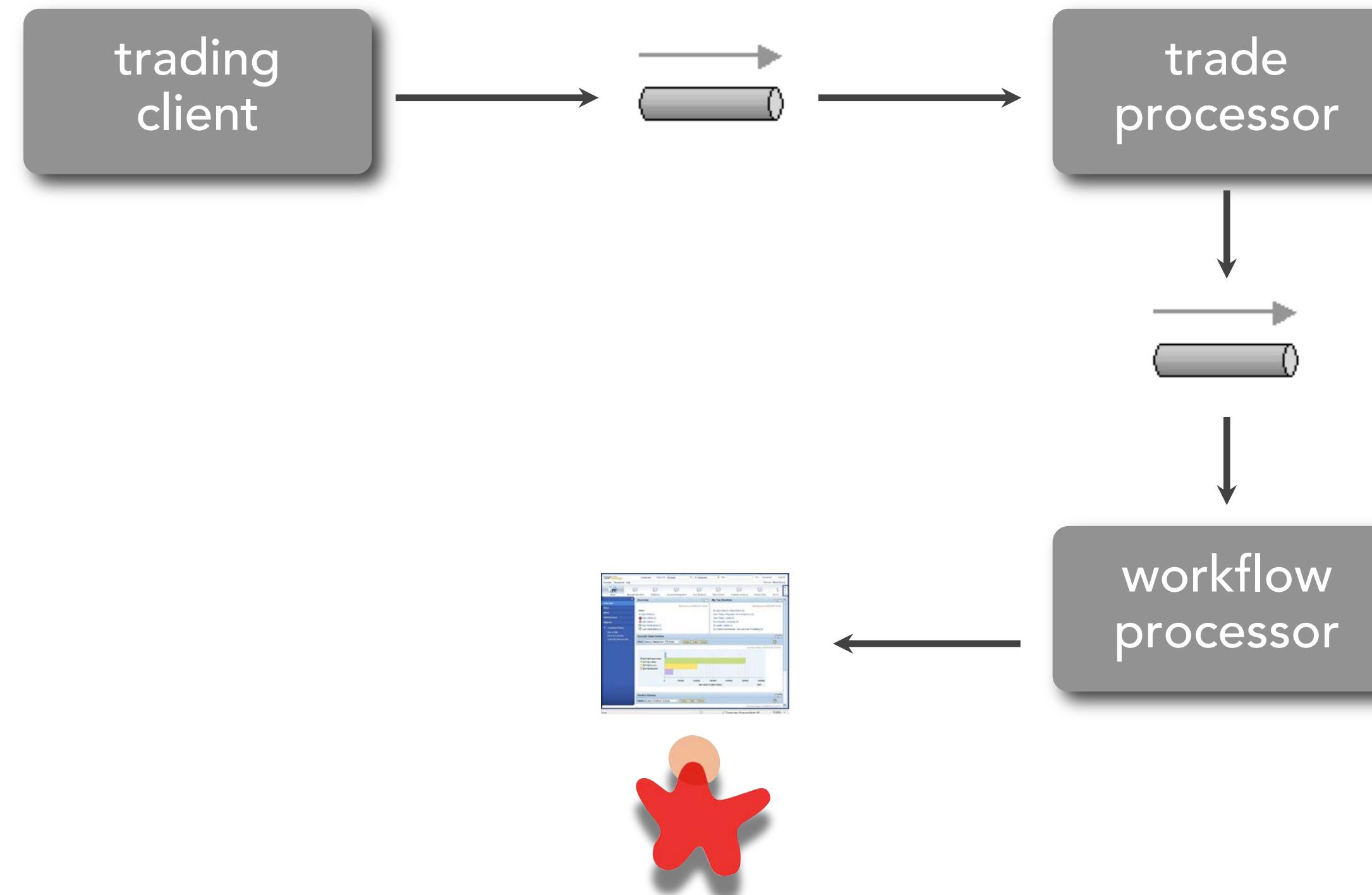
# workflow event pattern

while asynchronously processing trades an error occurs with one of the trade orders



# workflow event pattern

while asynchronously processing trades an error occurs with one of the trade orders



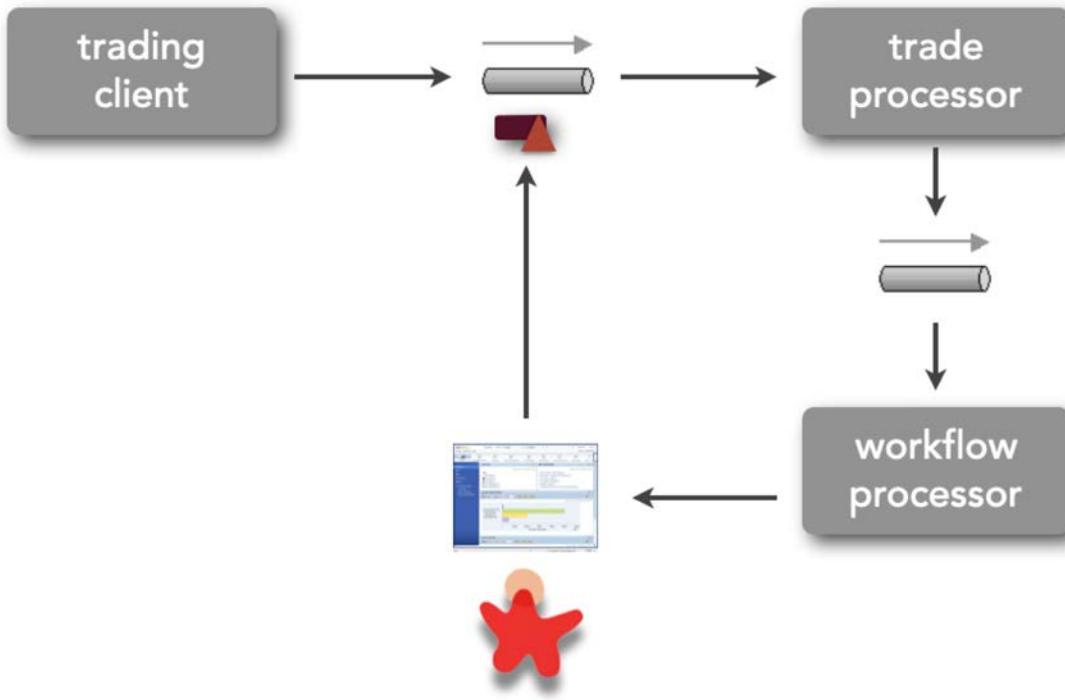
# workflow event pattern



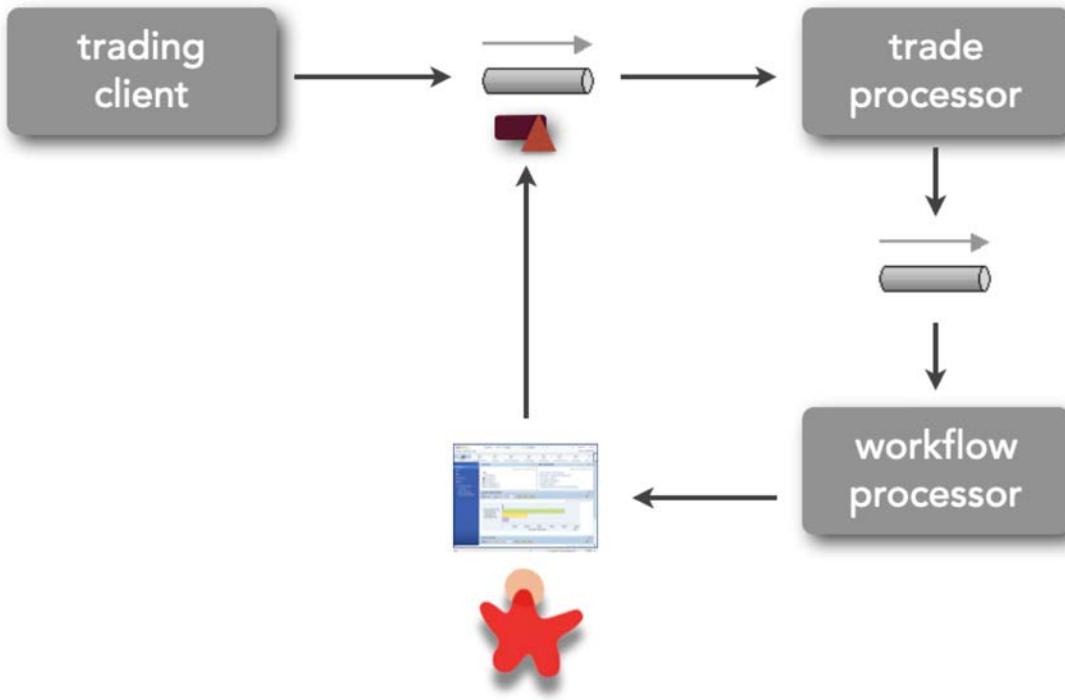
let's see the result...

<https://github.com/wmr513/reactive/tree/master/workflow>

# workflow event pattern



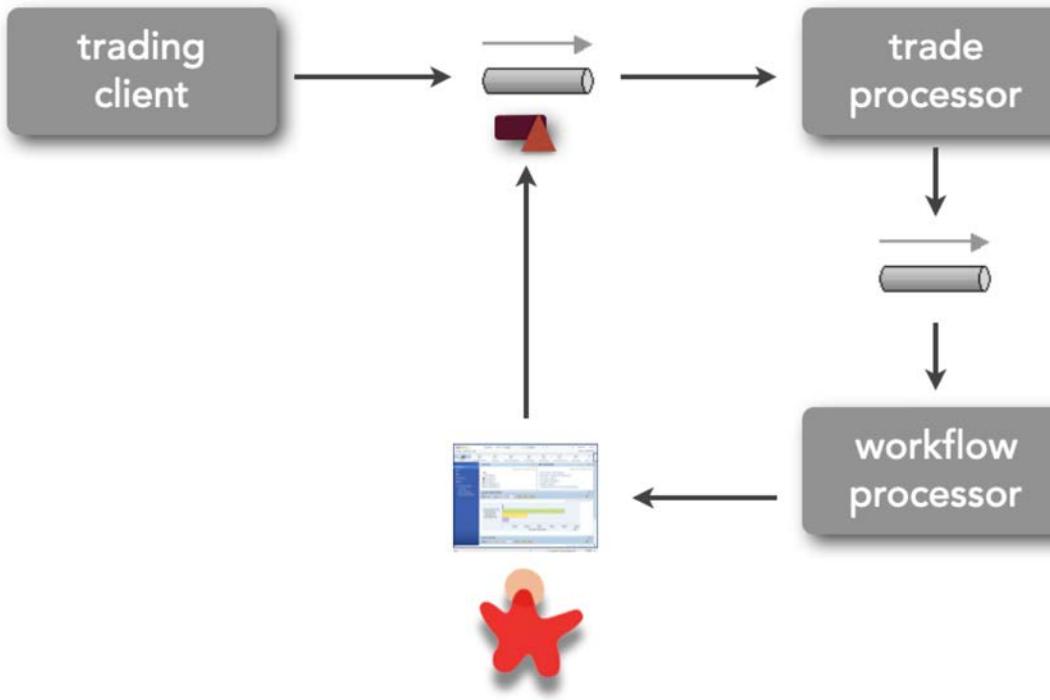
# workflow event pattern



programmatic error correction  
better processing throughput  
managed async error correction



# workflow event pattern



programmatic error correction  
better processing throughput  
managed async error correction

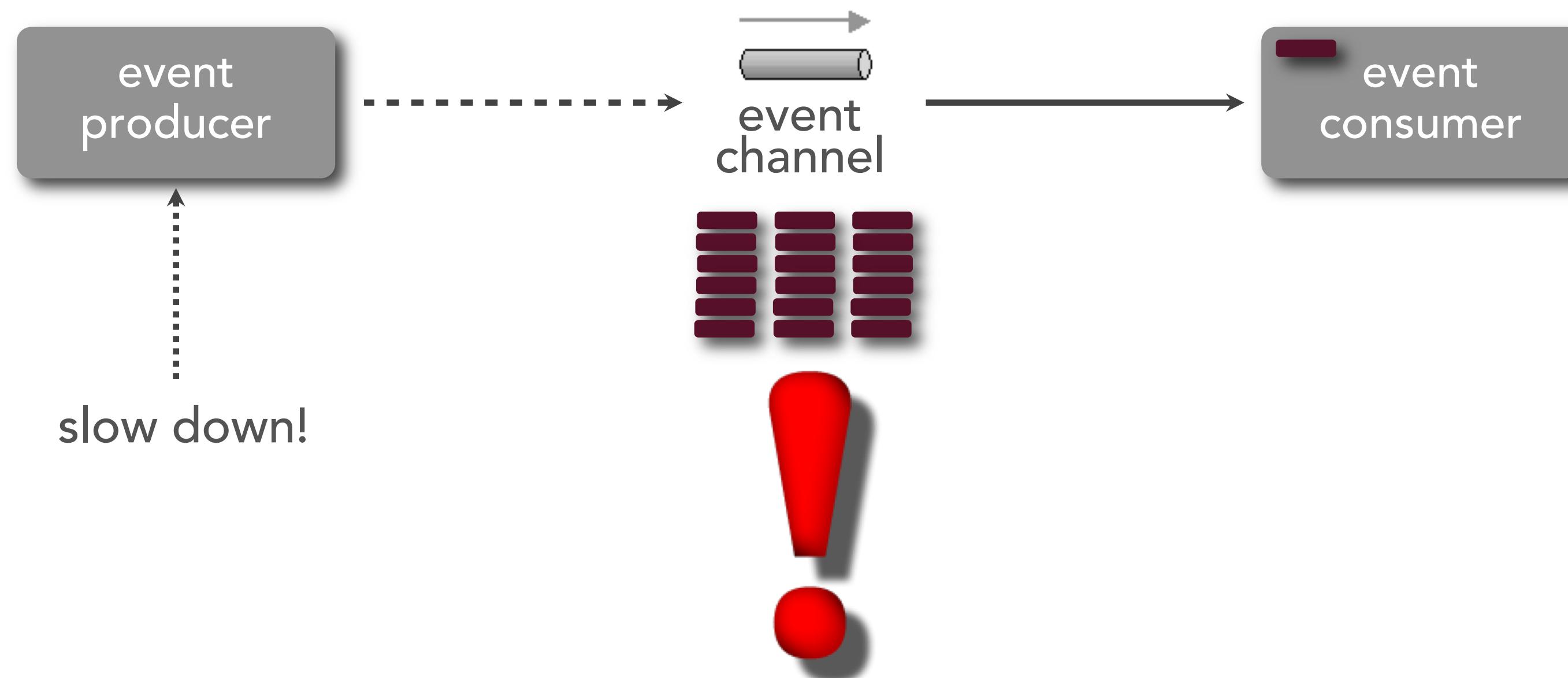


increased complexity  
message order issues  
increased cost

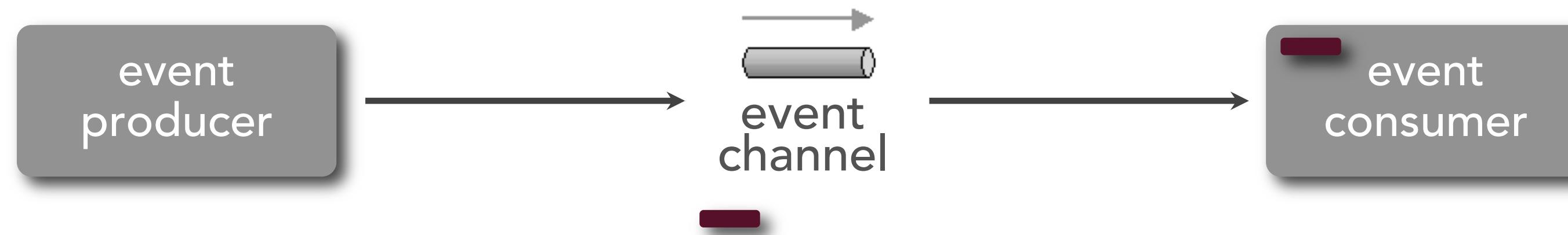
# Producer Control Flow Pattern

# producer control flow pattern

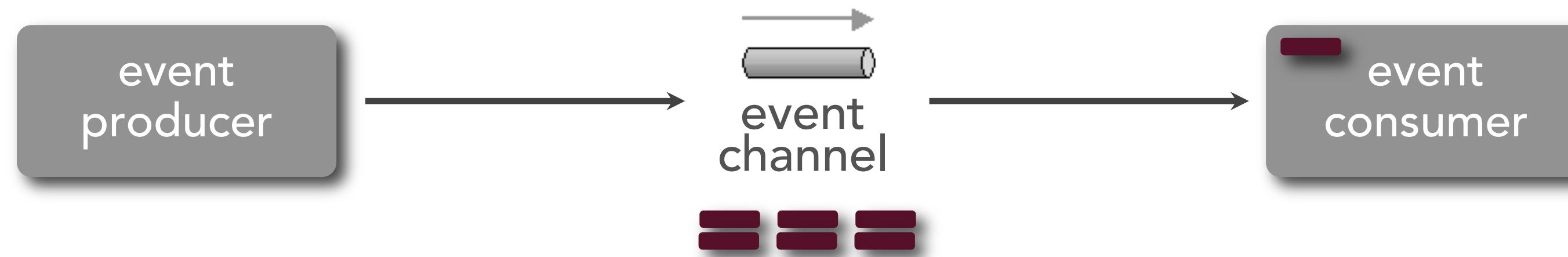
*“how can I slow down message producers when the messaging system becomes overwhelmed?”*



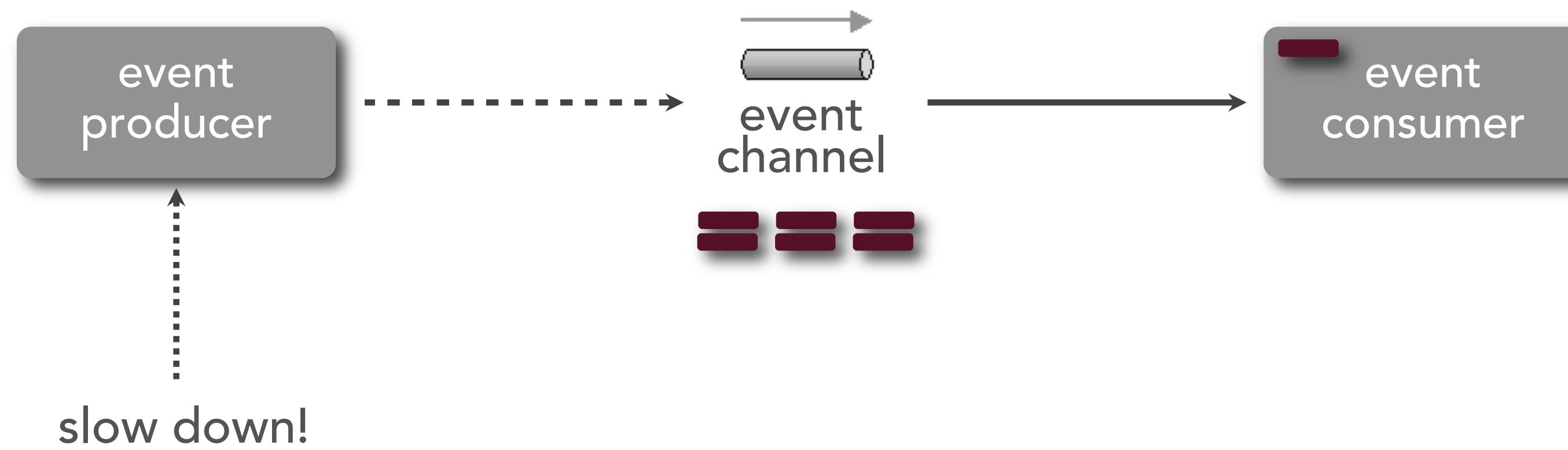
# producer control flow pattern



# producer control flow pattern



# producer control flow pattern



# producer control flow pattern

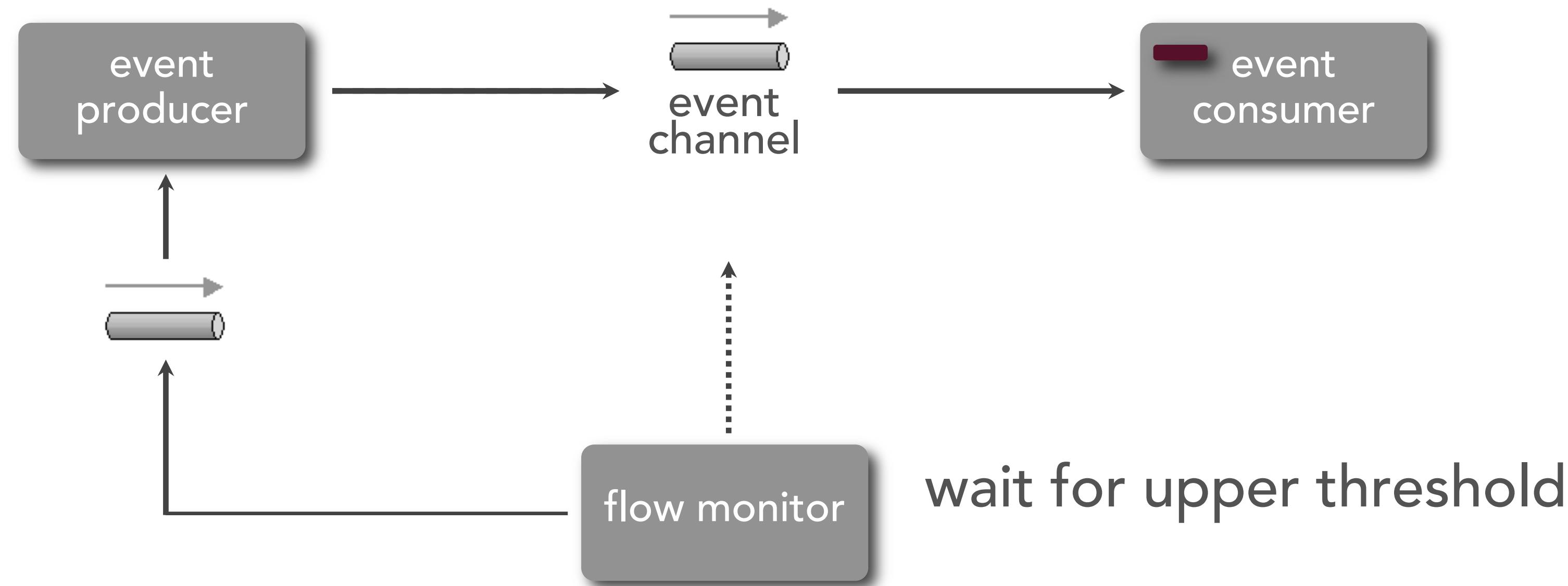


# producer control flow pattern

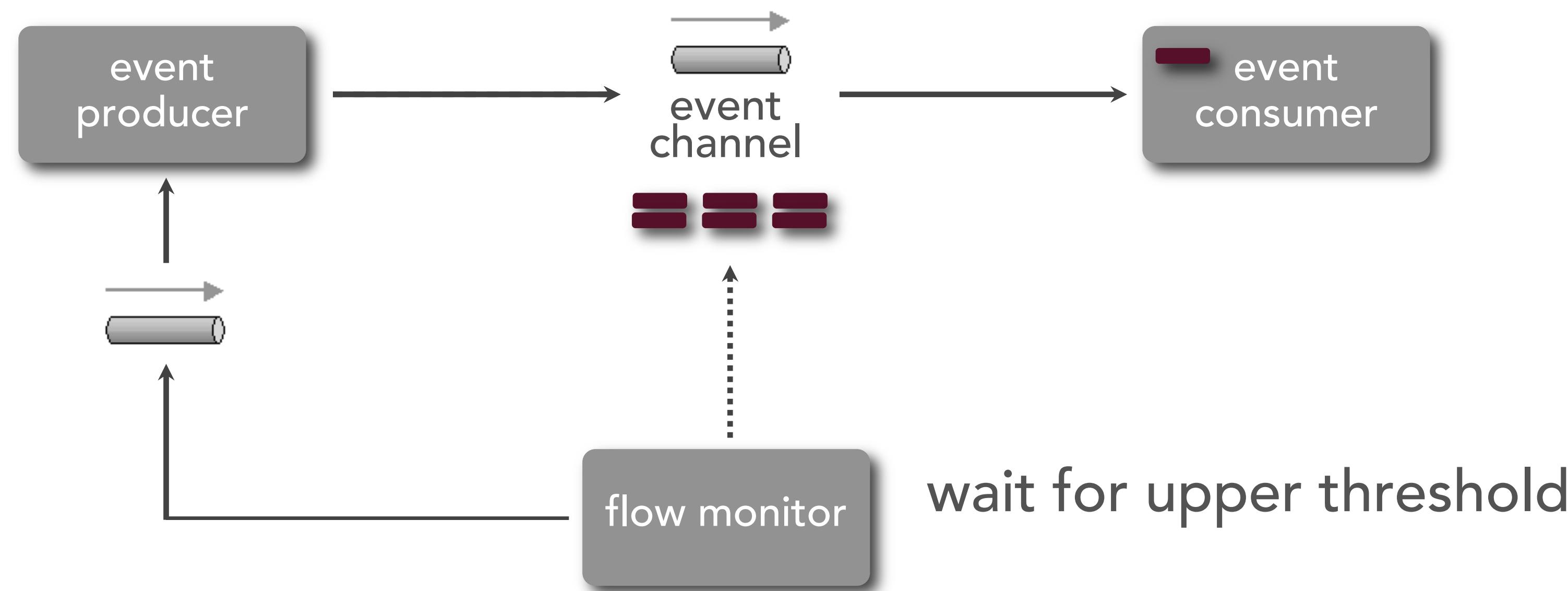


prevent send (broker) vs. slowdown send (pattern)

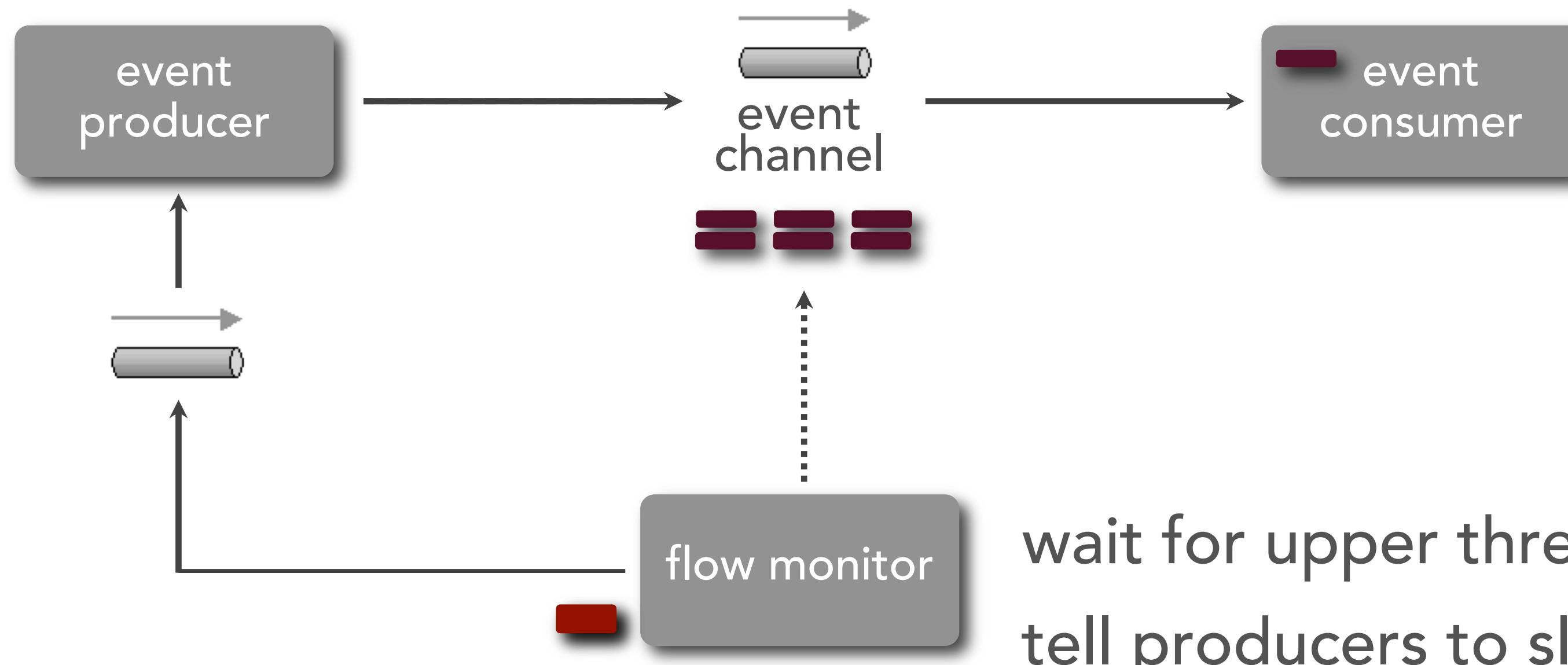
# producer control flow pattern



# producer control flow pattern

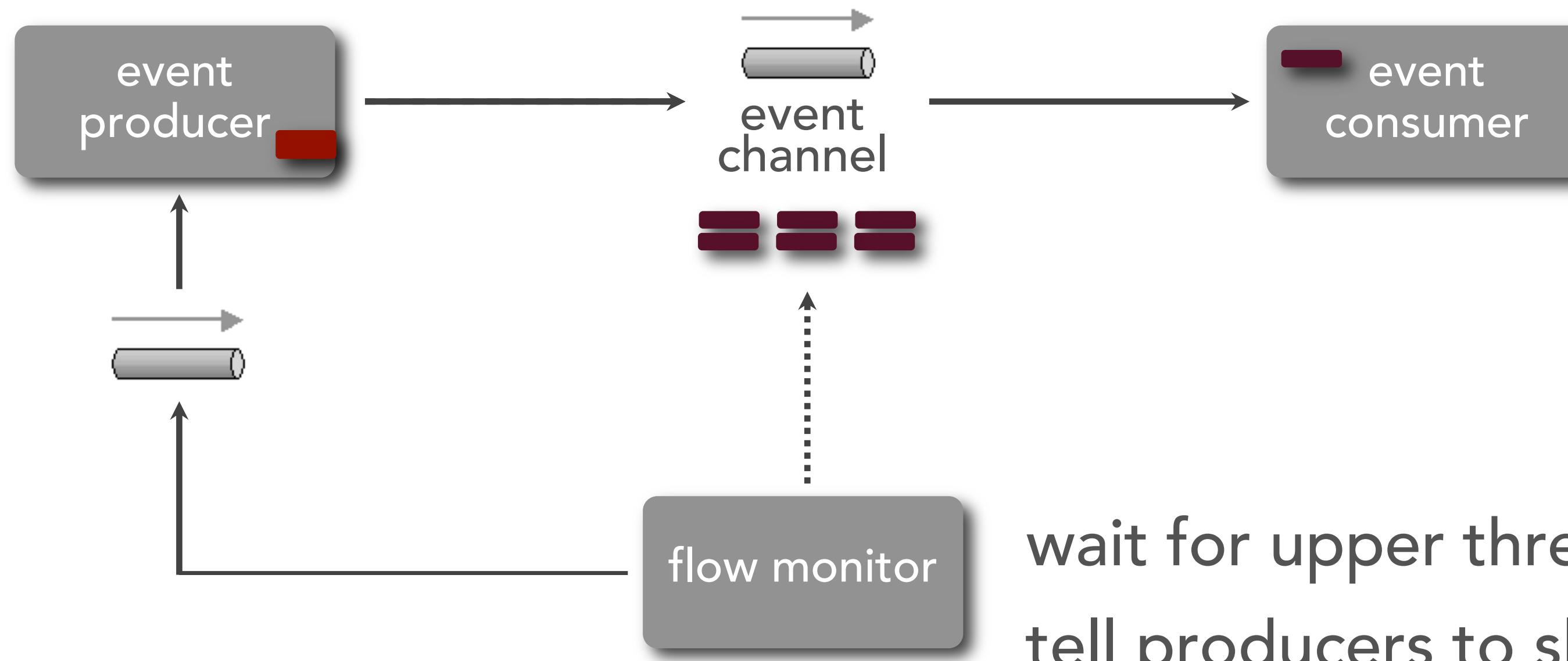


# producer control flow pattern



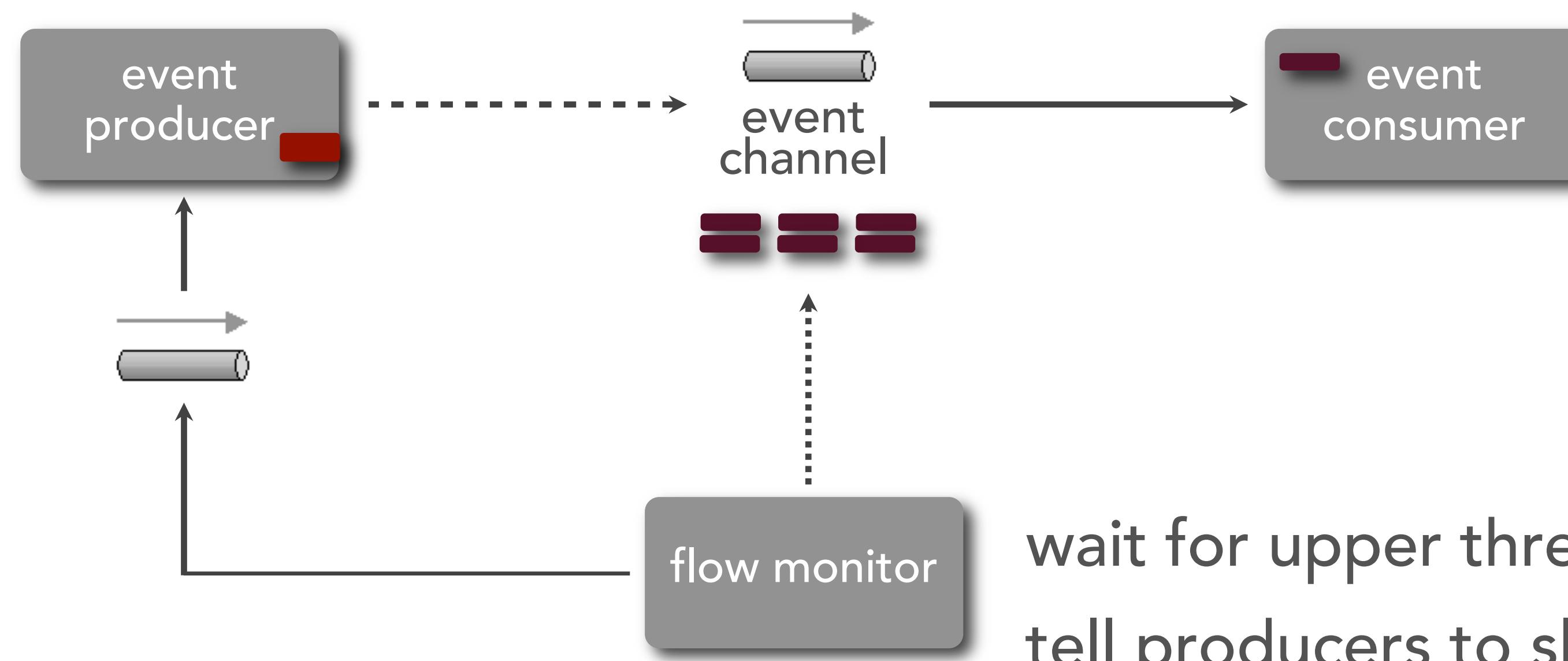
wait for upper threshold  
tell producers to slow down

# producer control flow pattern

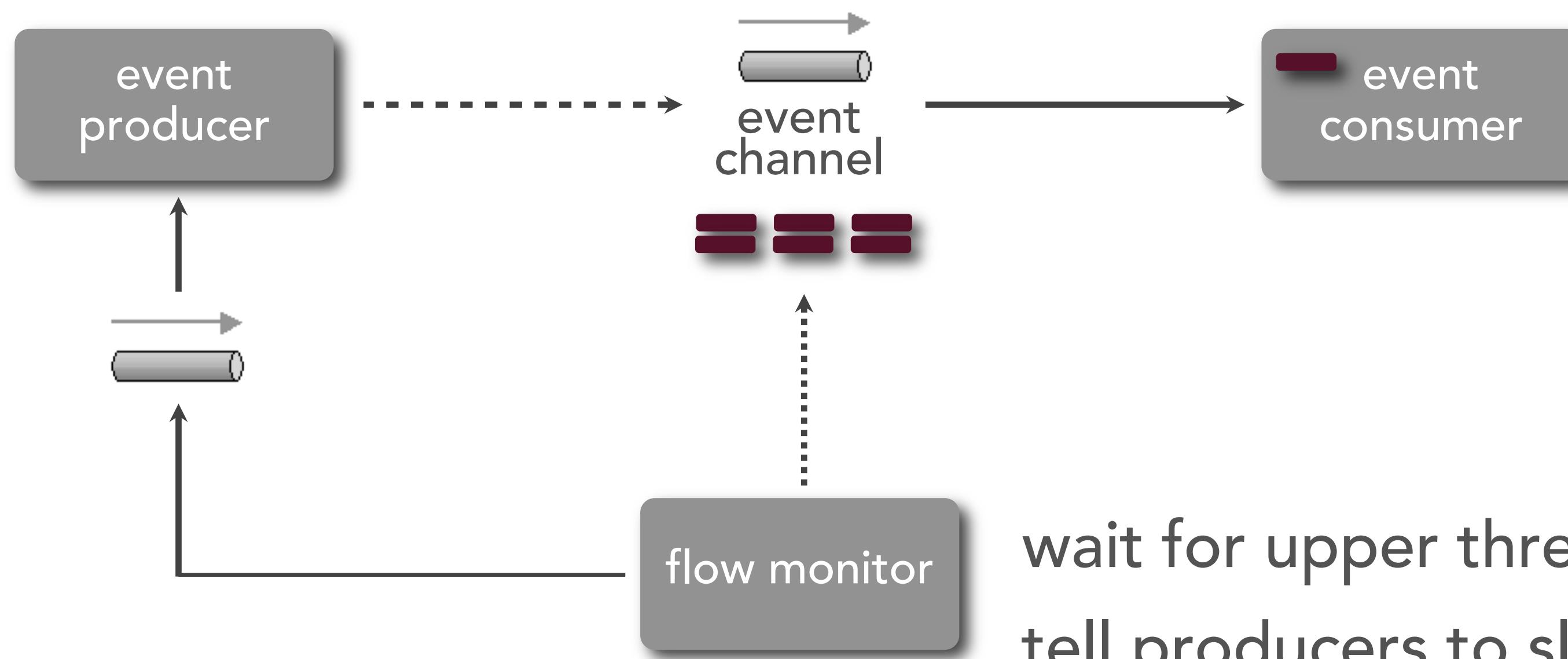


wait for upper threshold  
tell producers to slow down

# producer control flow pattern

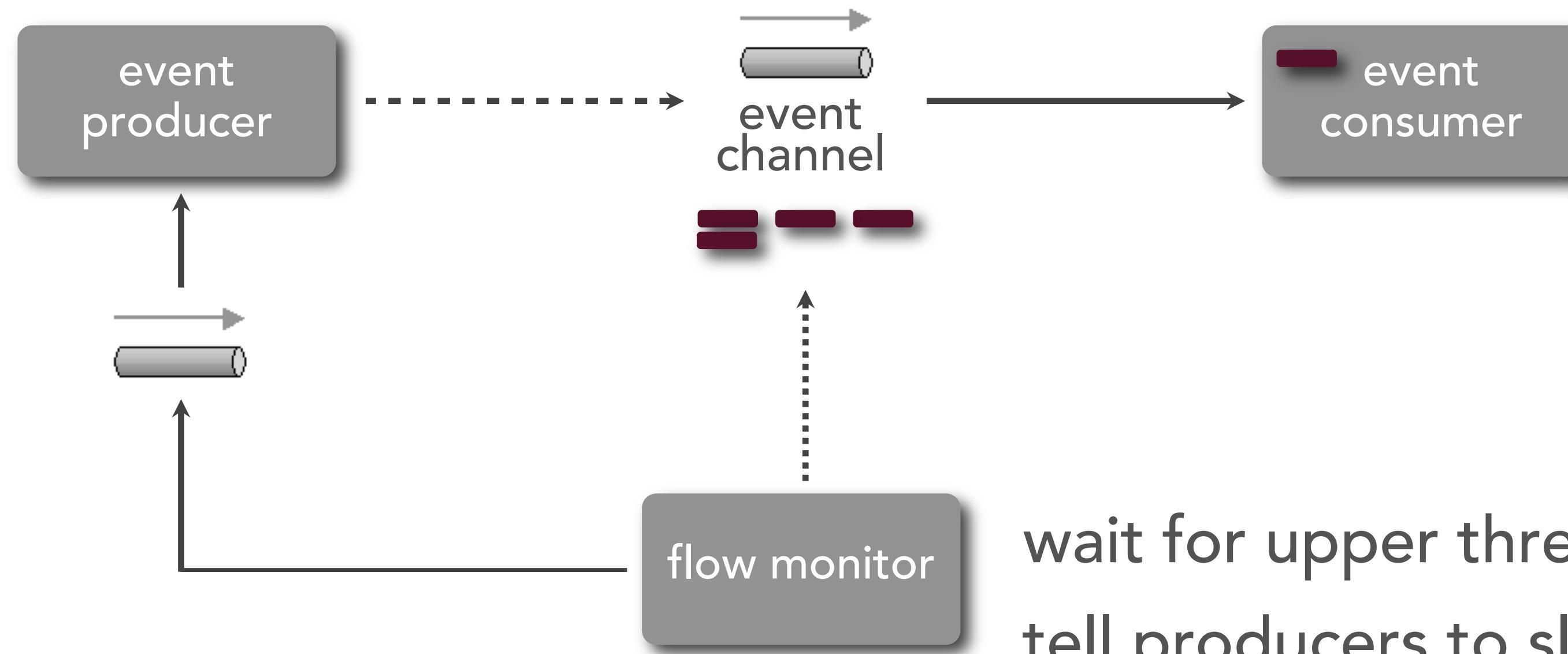


# producer control flow pattern



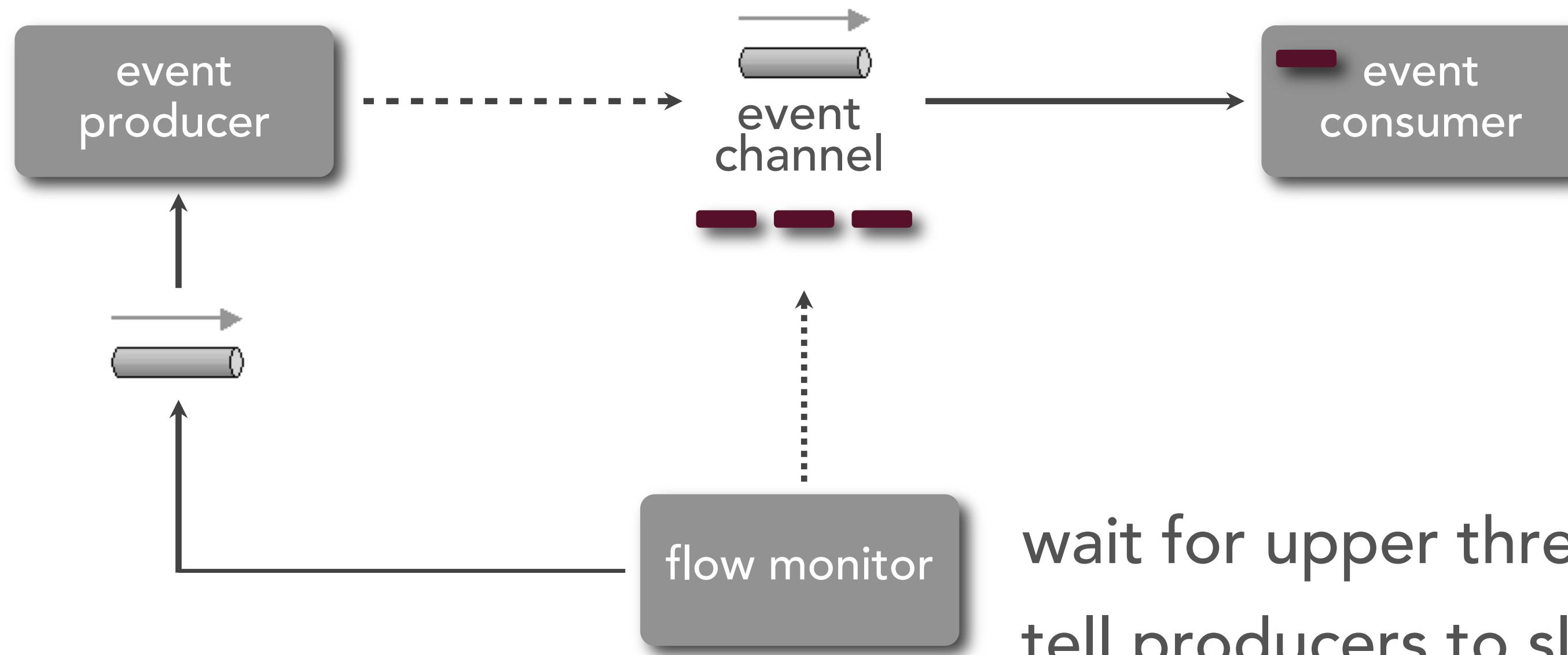
wait for upper threshold  
tell producers to slow down  
wait for lower threshold

# producer control flow pattern



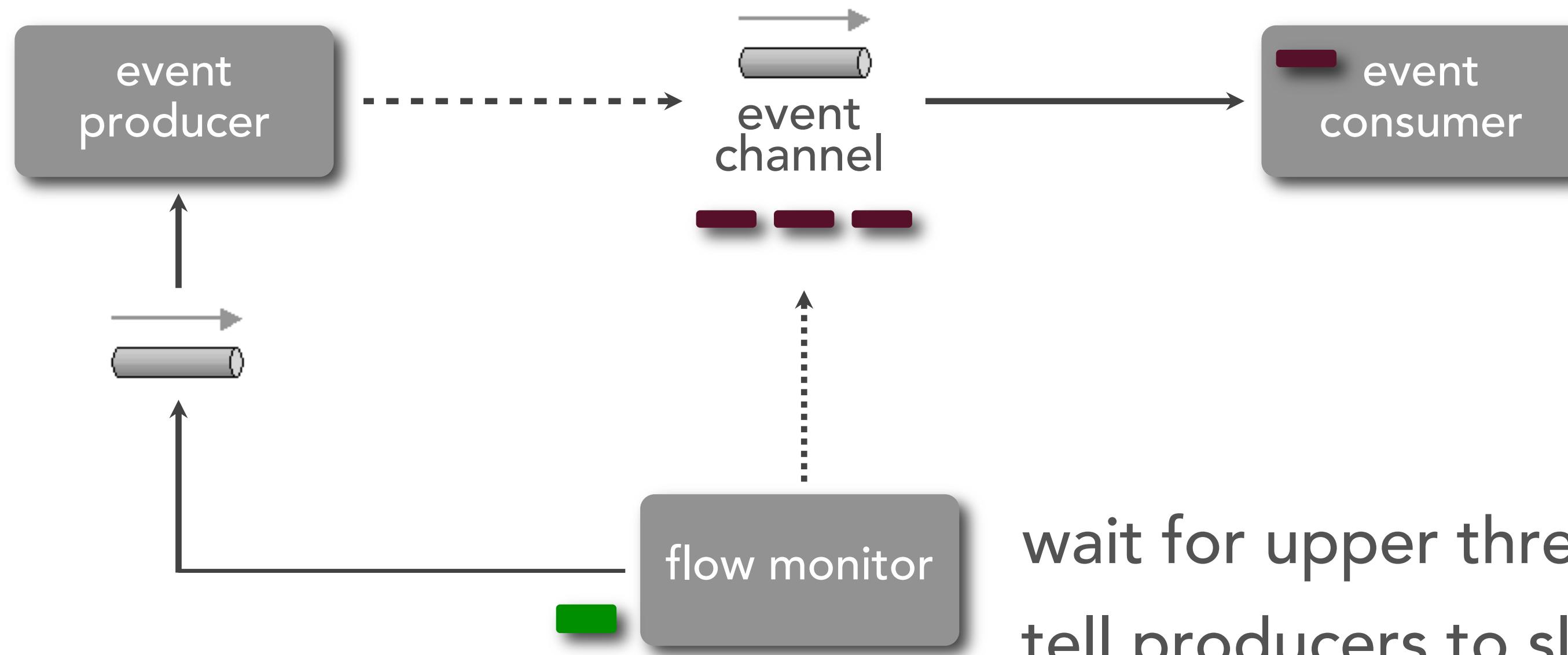
wait for upper threshold  
tell producers to slow down  
wait for lower threshold

# producer control flow pattern



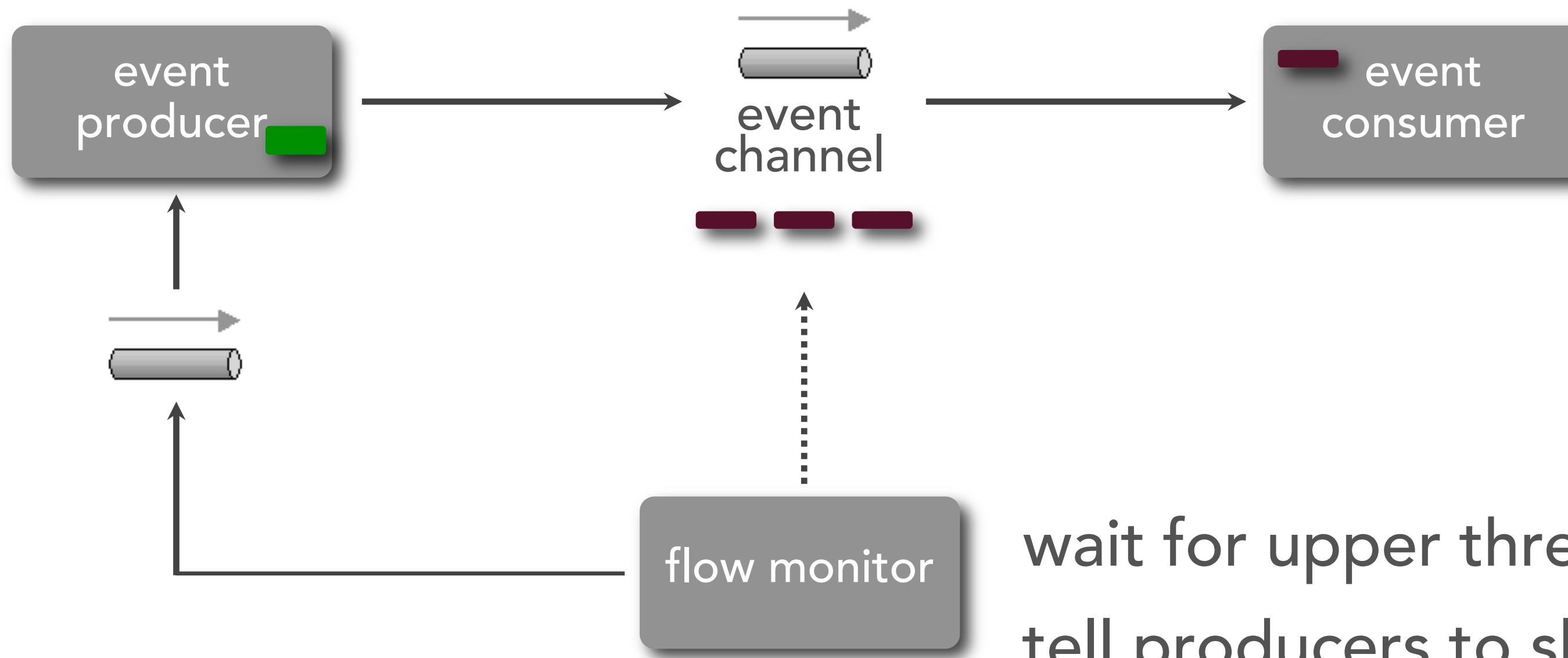
wait for upper threshold  
tell producers to slow down  
wait for lower threshold

# producer control flow pattern



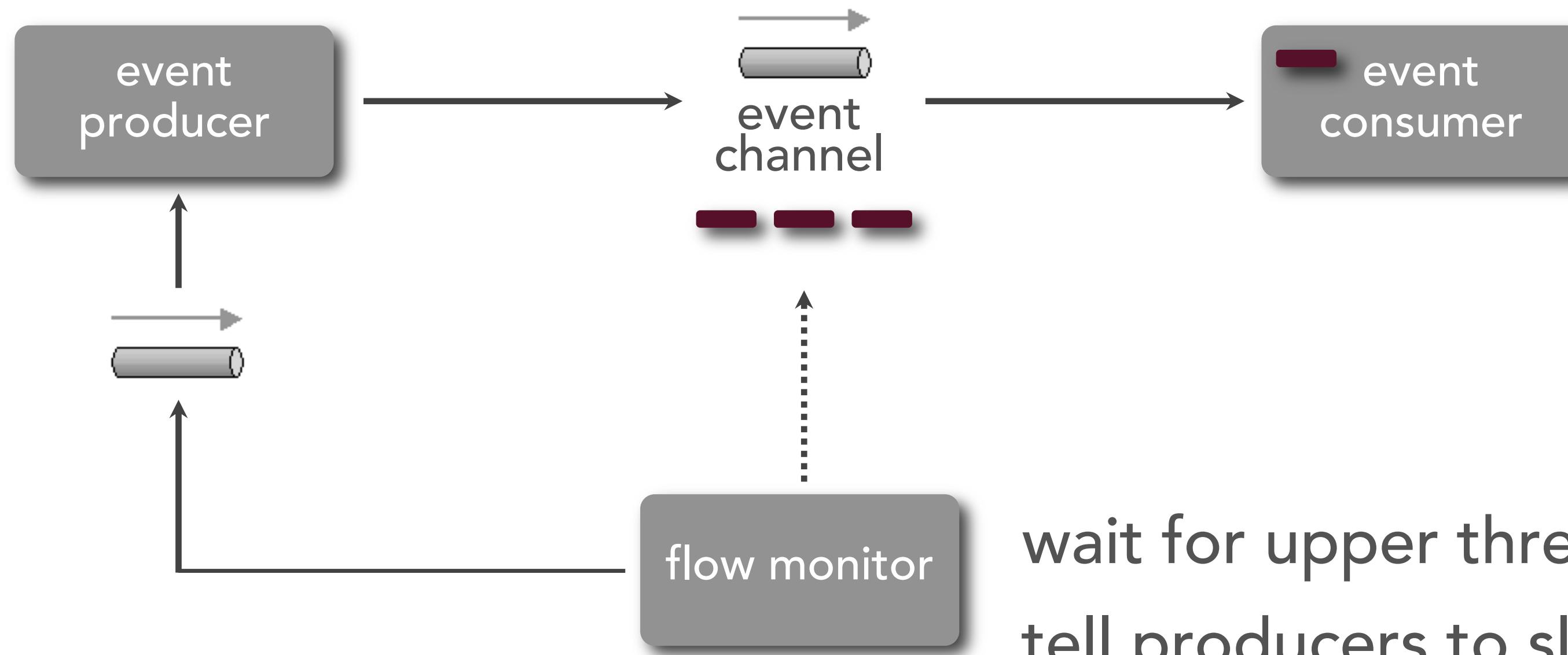
wait for upper threshold  
tell producers to slow down  
wait for lower threshold  
tell producers to resume

# producer control flow pattern



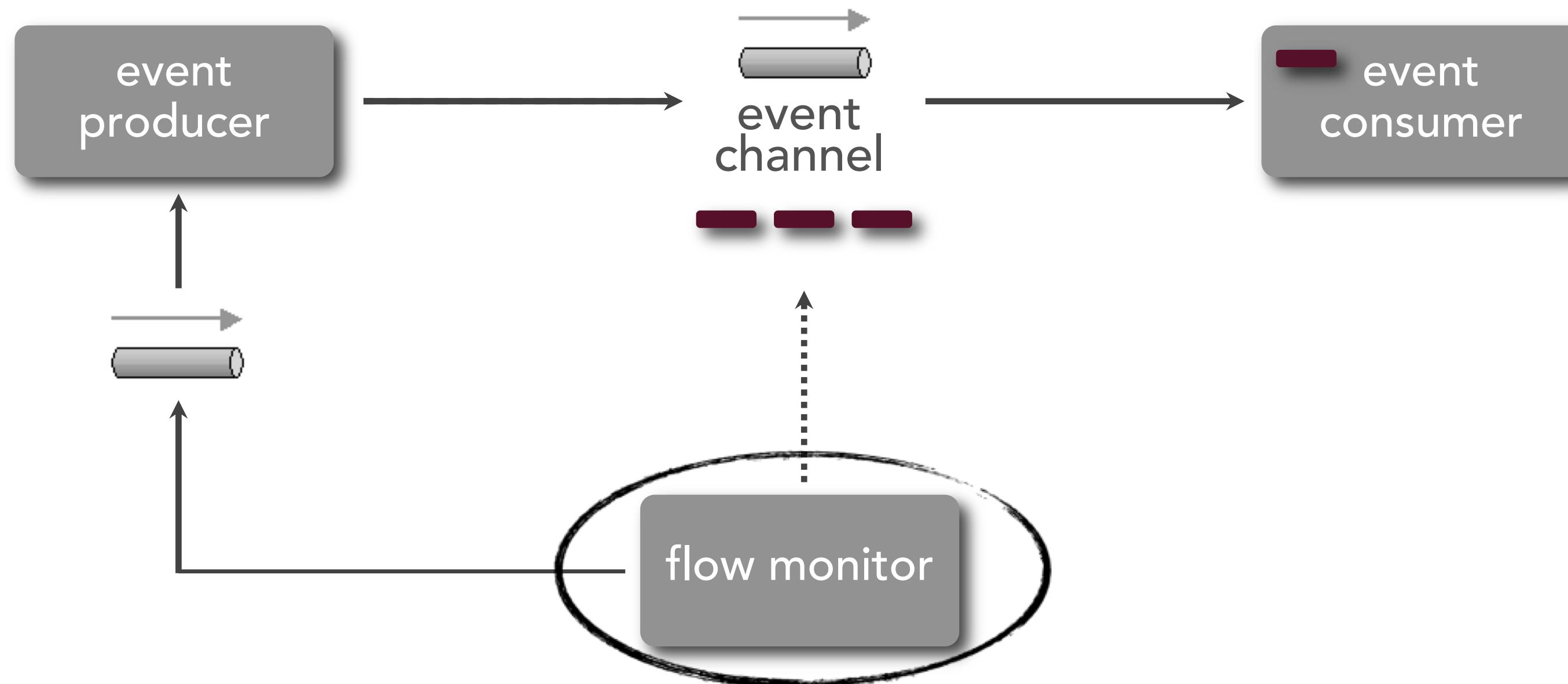
wait for upper threshold  
tell producers to slow down  
wait for lower threshold  
tell producers to resume

# producer control flow pattern



wait for upper threshold  
tell producers to slow down  
wait for lower threshold  
tell producers to resume

# producer control flow pattern



# producer control flow pattern

## flow monitor

```
public void execute() throws Exception {  
    //connect to message broker
```

# producer control flow pattern

## flow monitor

```
public void execute() throws Exception {  
    //connect to message broker  
    long threshold = 10;  
    boolean controlFlow = false;
```

# producer control flow pattern

## flow monitor

```
public void execute() throws Exception {  
    //connect to message broker  
    long threshold = 10;  
    boolean controlFlow = false;  
    while (true) {  
        long queueDepth = getMessageCount("trade.eq.q");
```

# producer control flow pattern

## flow monitor

```
public void execute() throws Exception {  
    //connect to message broker  
    long threshold = 10;  
    boolean controlFlow = false;  
    while (true) {  
        long queueDepth = getMessageCount("trade.eq.q");  
        if (queueDepth > threshold && !controlFlow) {  
            controlFlow = enableControlFlow(channel);  
        }  
    }  
}
```

# producer control flow pattern

## flow monitor

```
public void execute() throws Exception {  
    //connect to message broker  
    long threshold = 10;  
    boolean controlFlow = false;  
    while (true) {  
        long queueDepth = getMessageCount("trade.eq.q");  
        if (queueDepth > threshold && !controlFlow) {  
            controlFlow = enableControlFlow(channel);  
        } else if (queueDepth <= (threshold/2) && controlFlow) {  
            controlFlow = disableControlFlow(channel);  
        }  
        Thread.sleep(3000);  
    }  
}
```

# producer control flow pattern

## flow monitor

```
private boolean enableControlFlow(Channel channel) {  
    Message msg = createMessage(delay, 3000);  
    //send message to producer flow queue;  
    return true;  
}
```

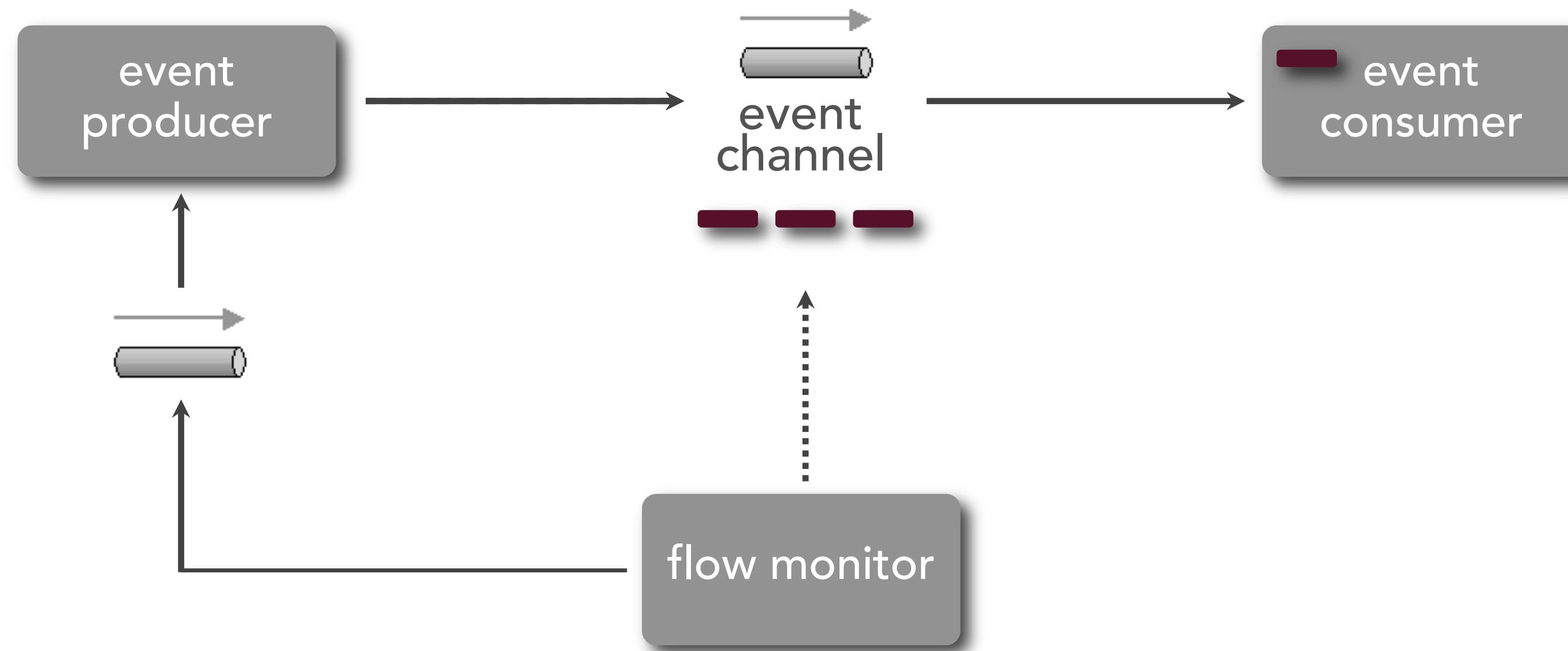
# producer control flow pattern

## flow monitor

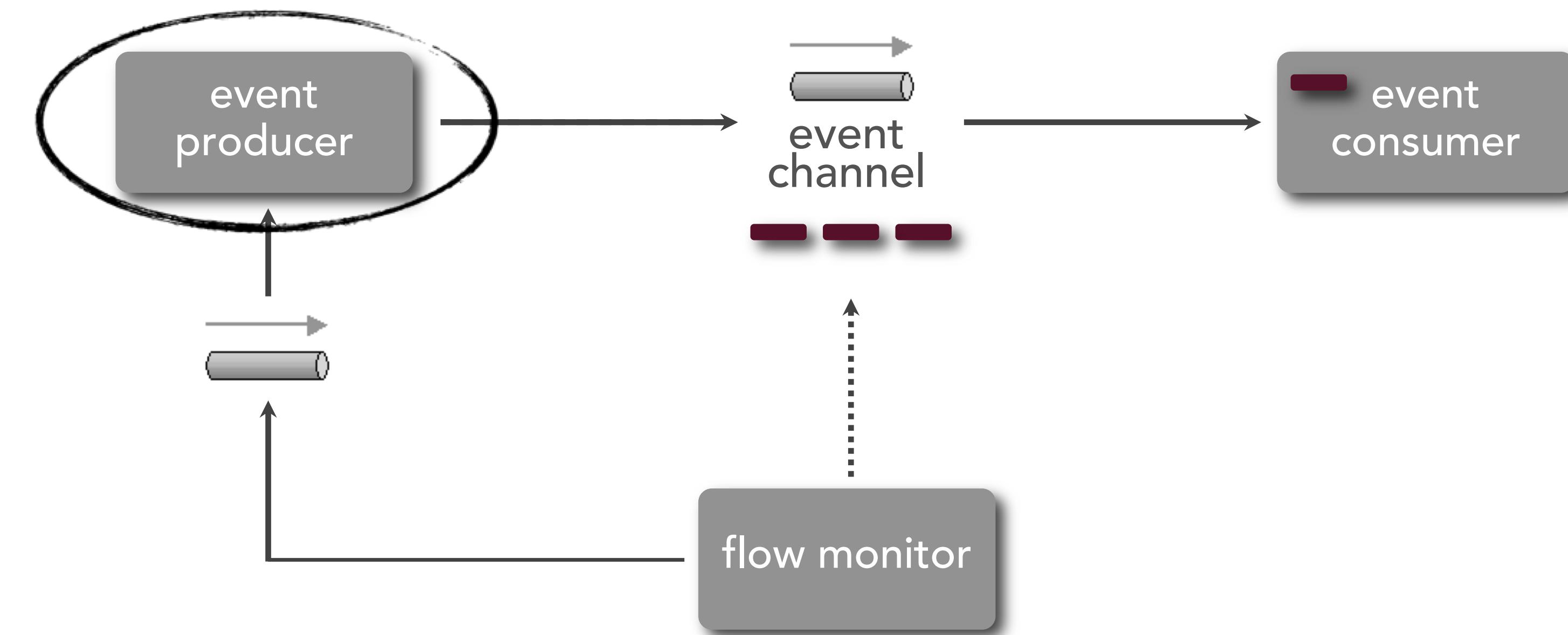
```
private boolean enableControlFlow(Channel channel) {  
    Message msg = createMessage(delay, 3000);  
    //send message to producer flow queue;  
    return true;  
}
```

```
private boolean disableControlFlow(Channel channel) {  
    Message msg = createMessage(delay, 0);  
    //send message to producer flow queue;  
    return false;  
}
```

# producer control flow pattern



# producer control flow pattern



# producer control flow pattern

## event producer

```
public void startListener() {  
    new Thread() { public void run() {  
        //connect to message broker and create consumer  
    } } .start();  
}
```

# producer control flow pattern

## event producer

```
public void startListener() {  
    new Thread() { public void run() {  
        //connect to message broker and create consumer  
        while (true) {  
            msg = getNextMessageFromQueue();  
        }  
    }  
}
```

# producer control flow pattern

## event producer

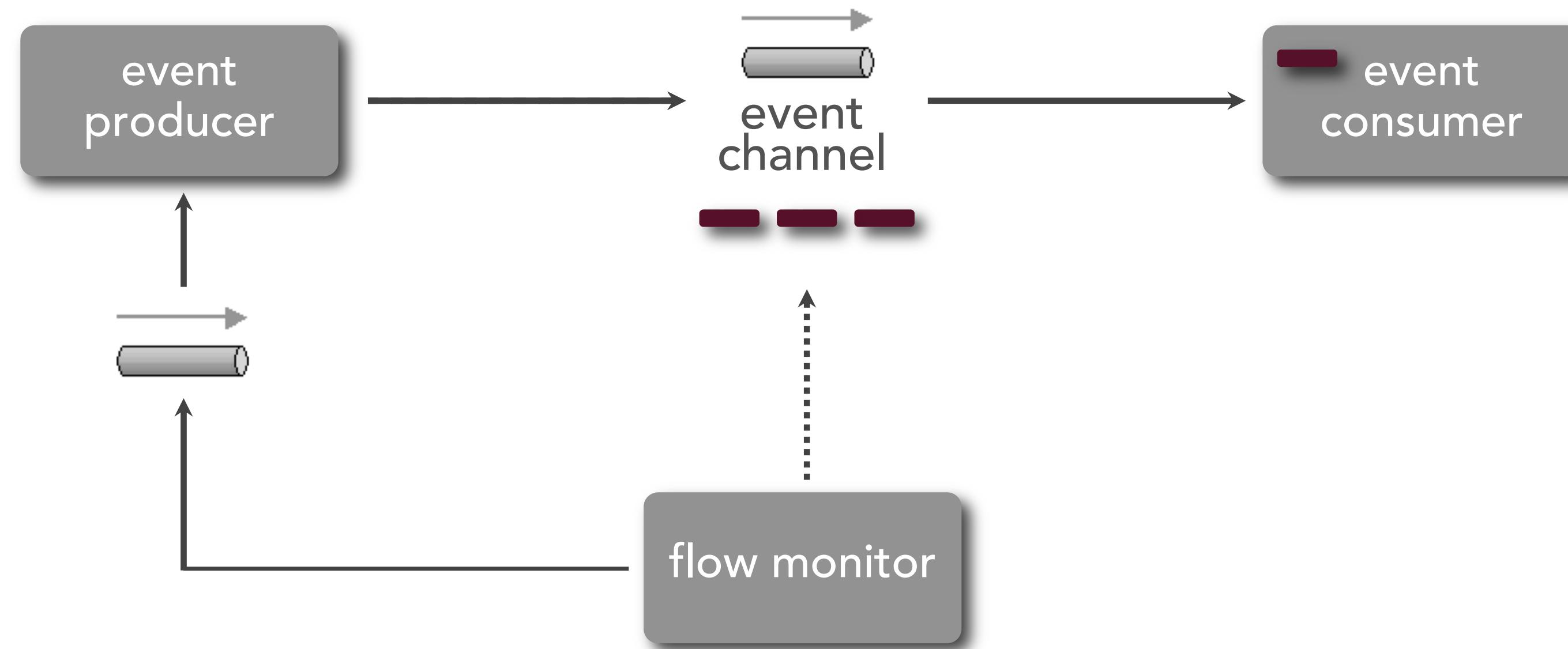
```
public void startListener() {  
    new Thread() { public void run() {  
        //connect to message broker and create consumer  
        while (true) {  
            msg = getNextMessageFromQueue();  
            long delayValue = msg.getDelayValue();  
            synchronized(delay) { delay = delayValue; }  
        }  
    }}.start();  
}
```

# producer control flow pattern

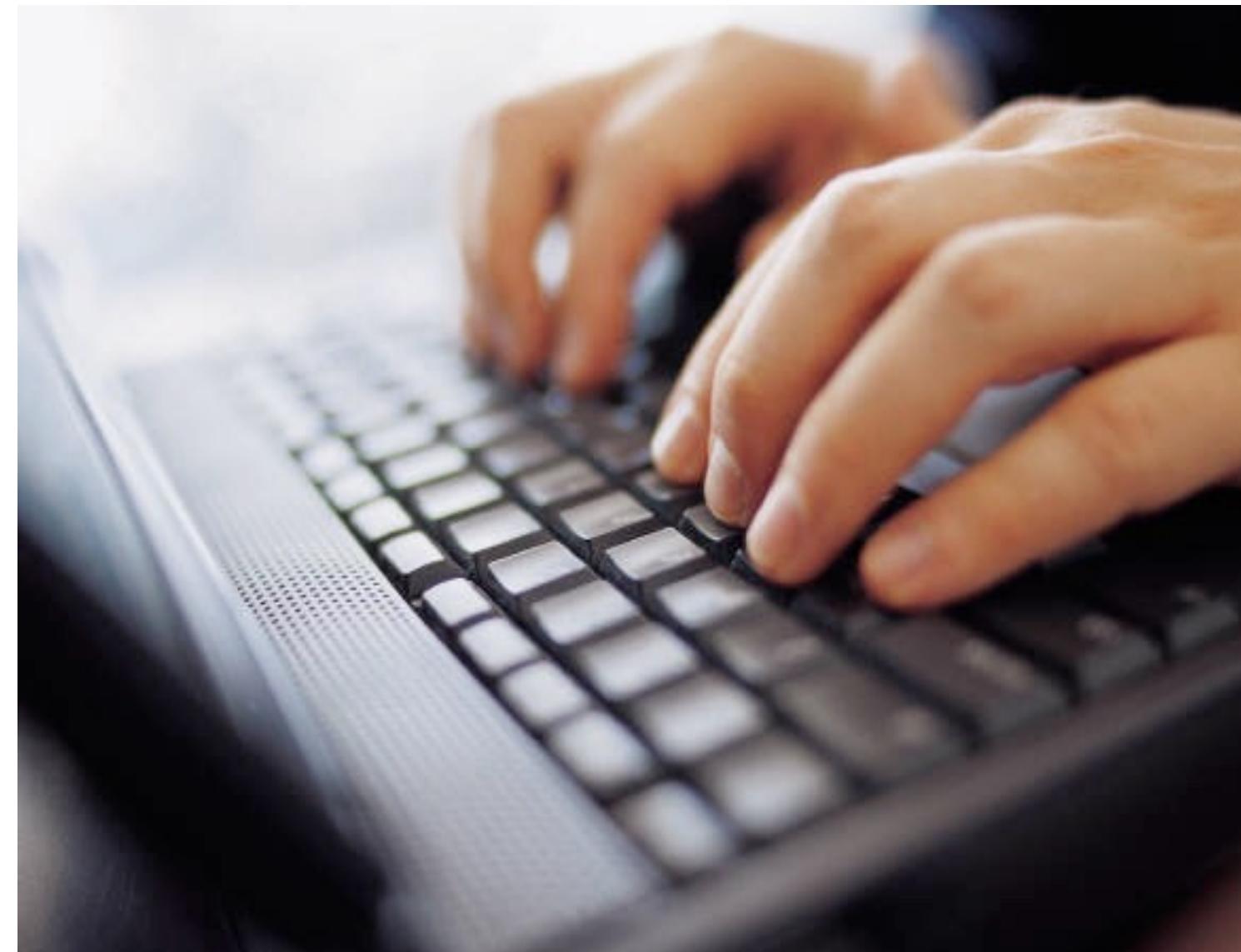
## event producer

```
public void startListener() {  
    new Thread() { public void run() {  
        //connect to message broker and create consumer  
        while (true) {  
            msg = getNextMessageFromQueue();  
            long delayValue = msg.getDelayValue();  
            synchronized(delay) { delay = delayValue; }  
        }  
    }}.start();  
}  
  
private void placeTrade() {  
    Thread.sleep(delay);  
    //send trade to processing queue...  
}
```

# producer control flow pattern



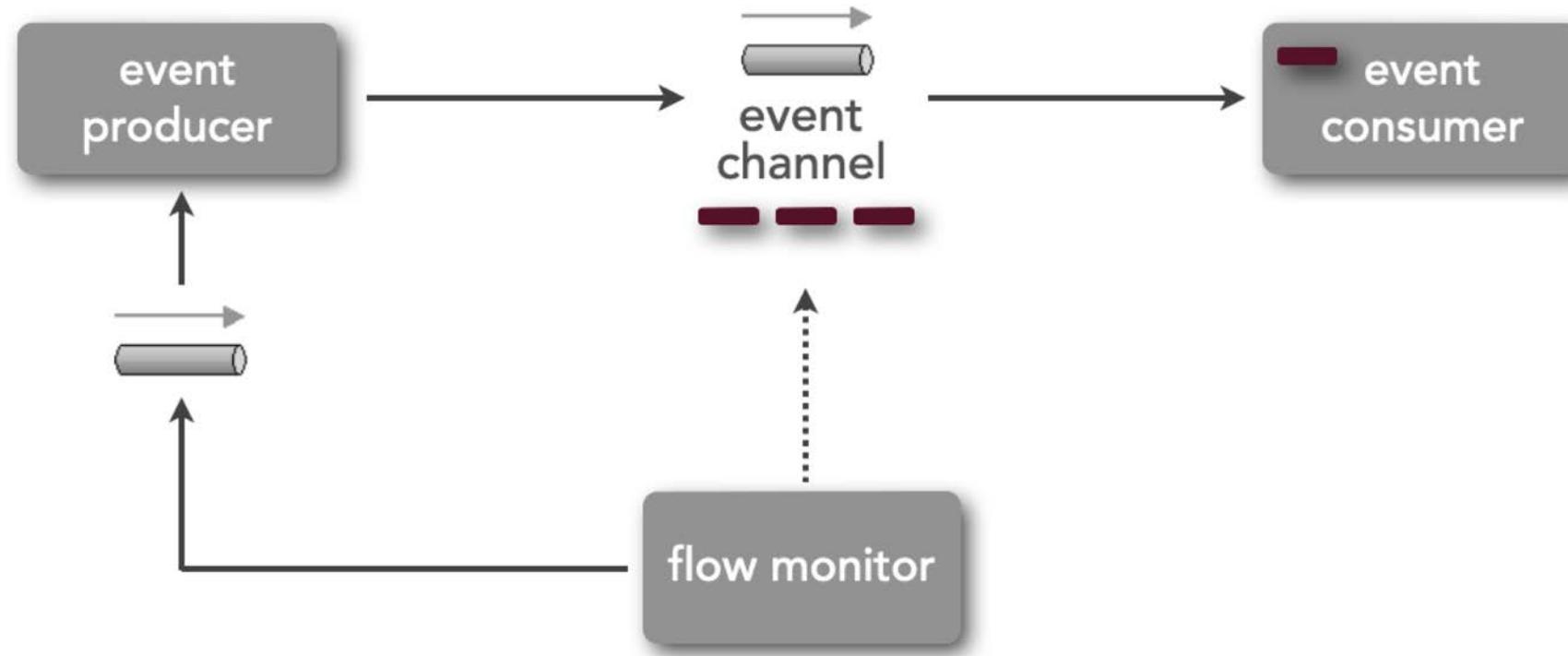
# producer control flow pattern



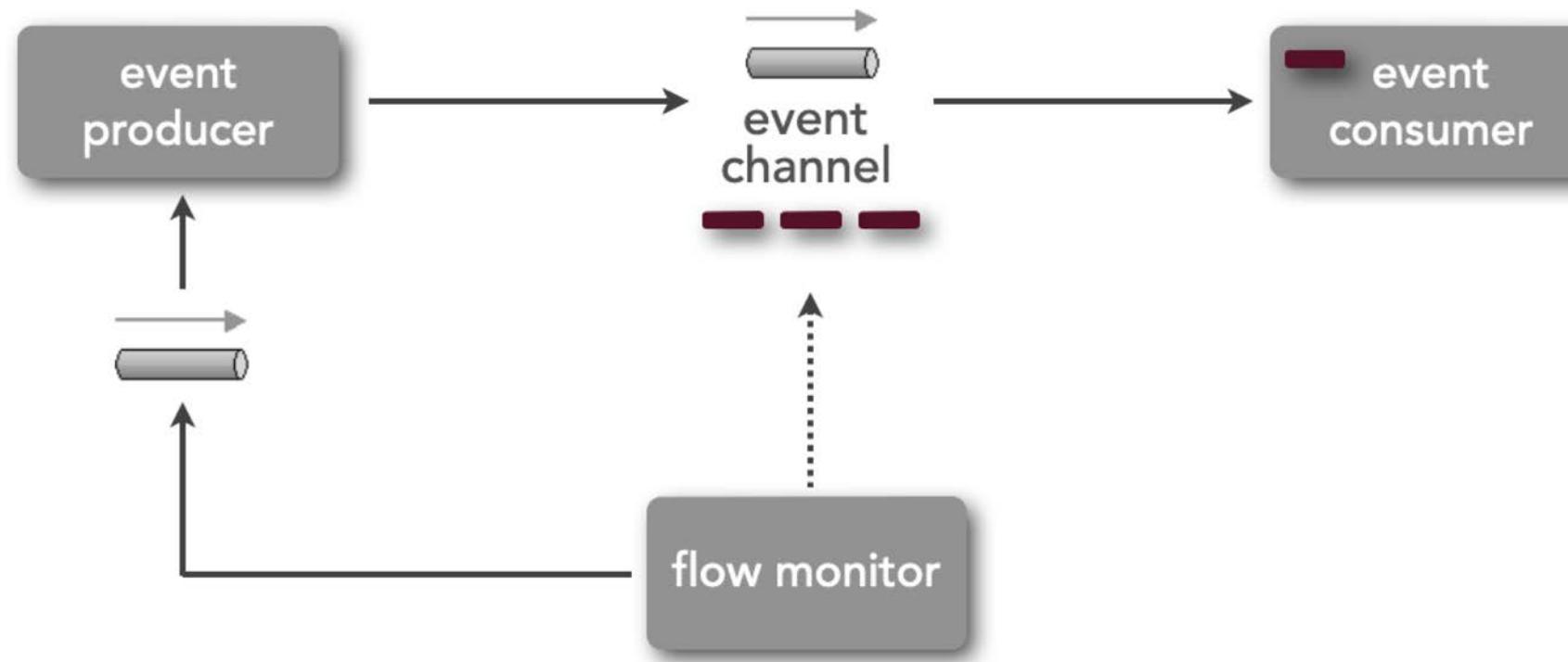
let's see the result...

<https://github.com/wmr513/reactive/tree/master/producercontrolflow>

# producer control flow pattern



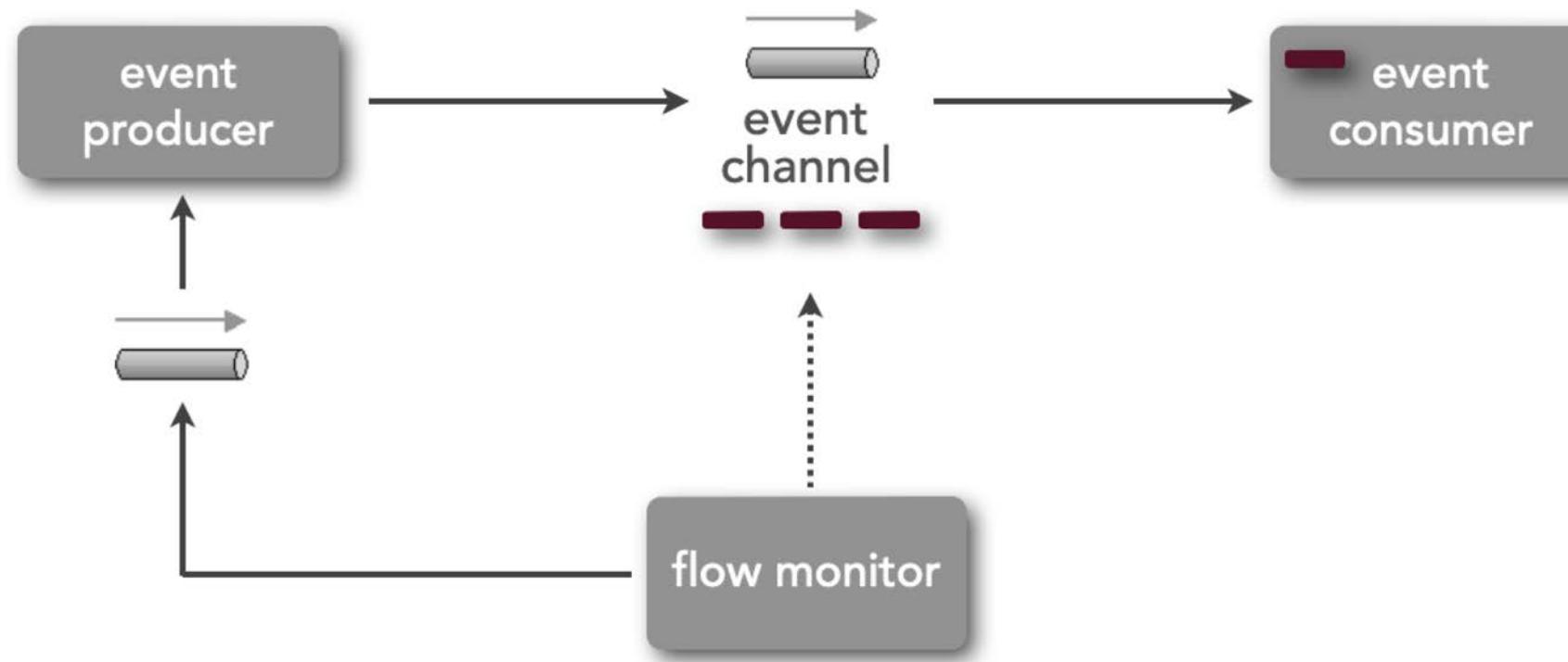
# producer control flow pattern



programmatic system repair



# producer control flow pattern



programmatic system repair

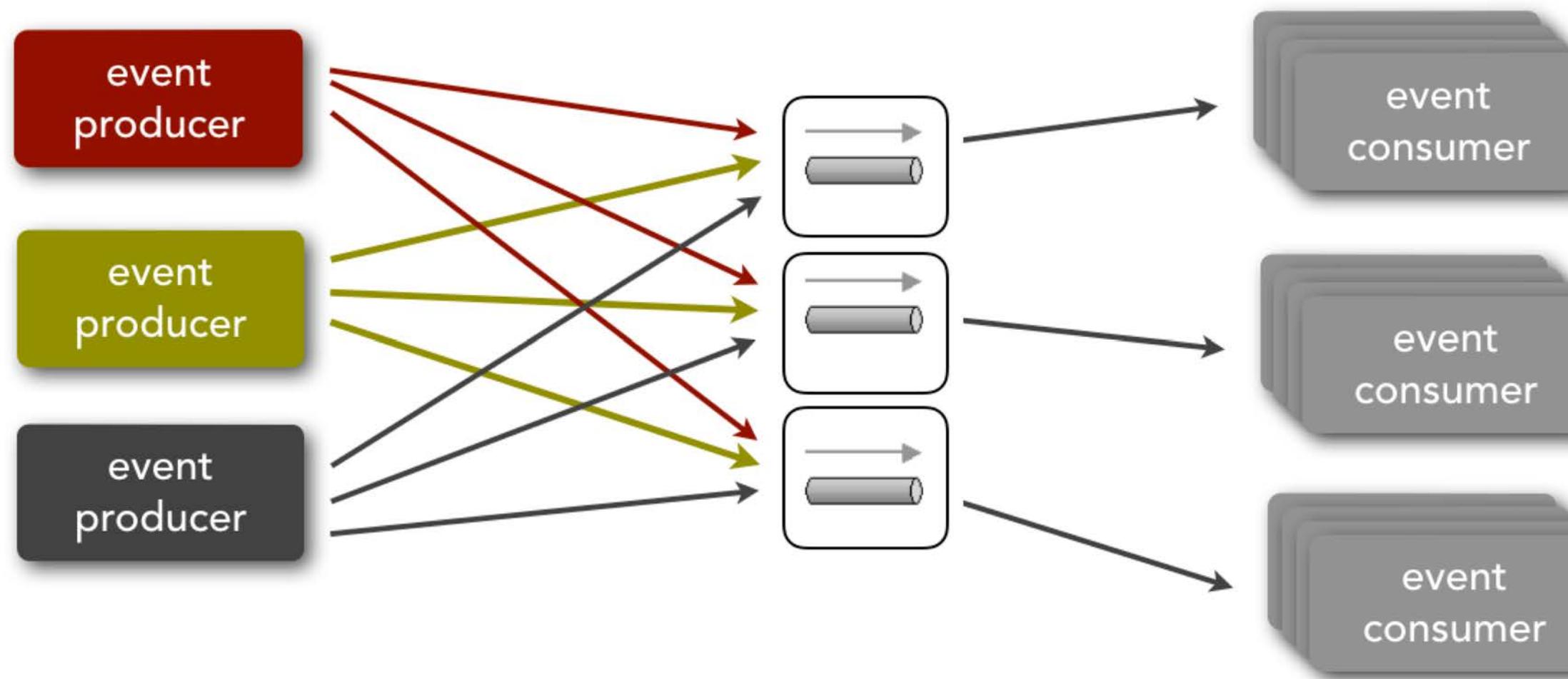


increased complexity  
impacts responsiveness

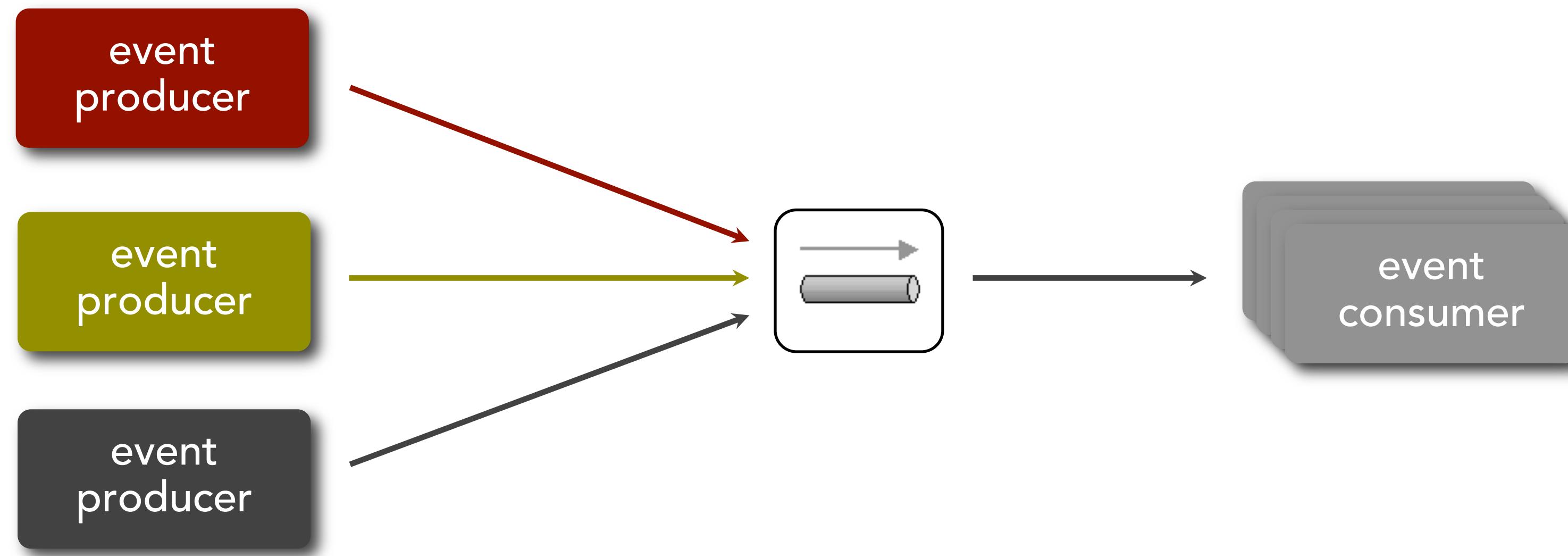
# Multi-Broker Pattern

# multi-broker pattern

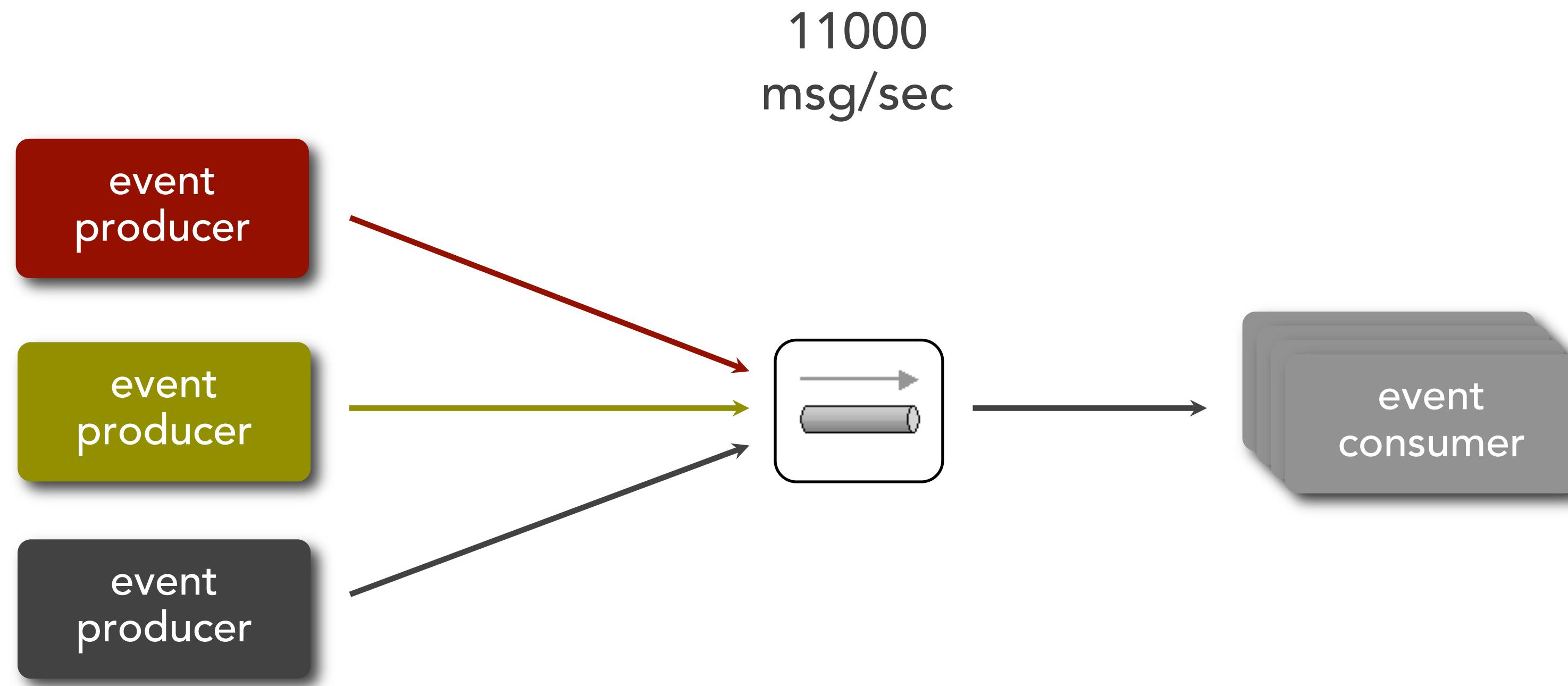
*“how can I increase the throughput and capacity of events through the system?”*



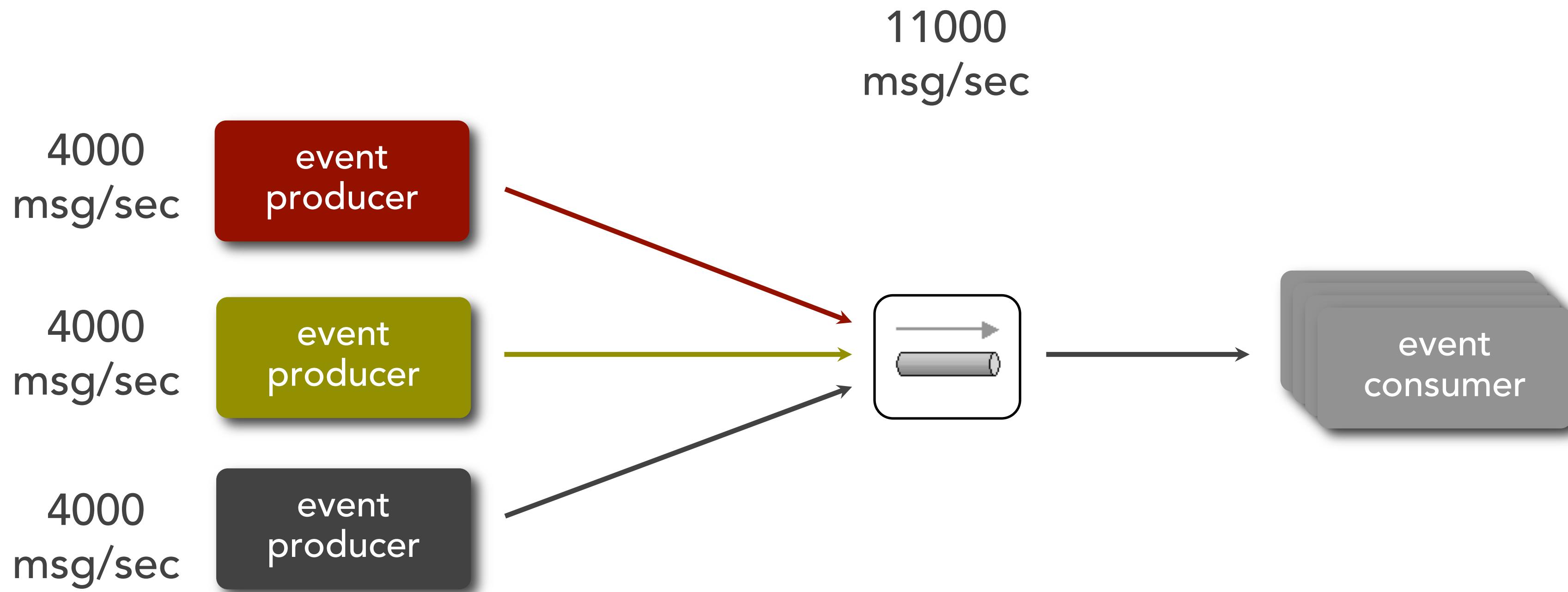
# multi-broker pattern



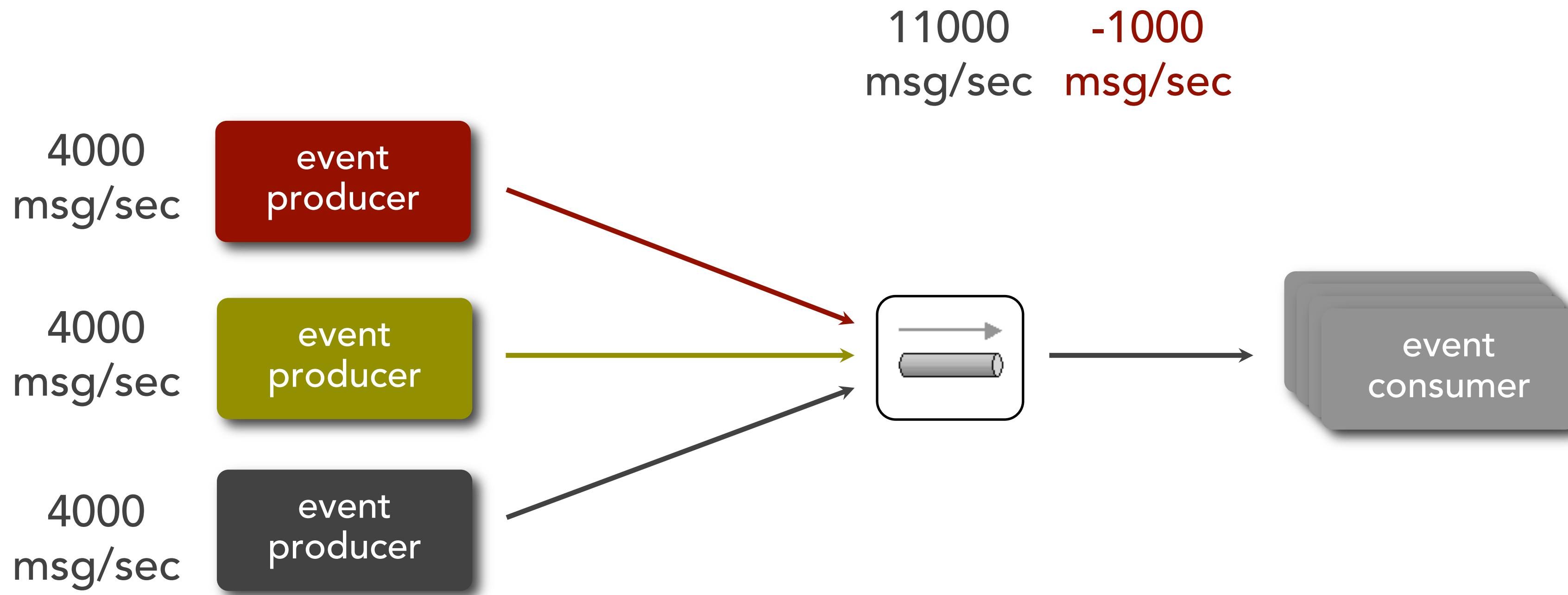
# multi-broker pattern



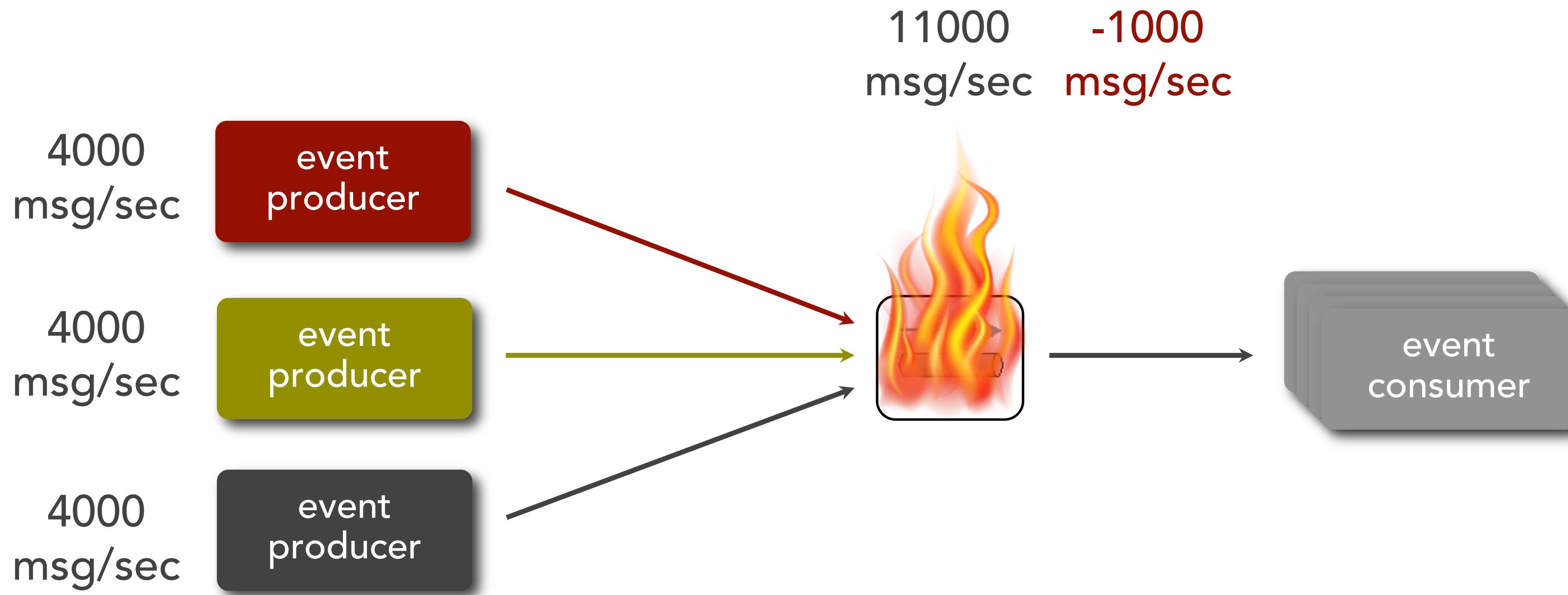
# multi-broker pattern



# multi-broker pattern



# multi-broker pattern



# multi-broker pattern

4000  
msg/sec

event  
producer

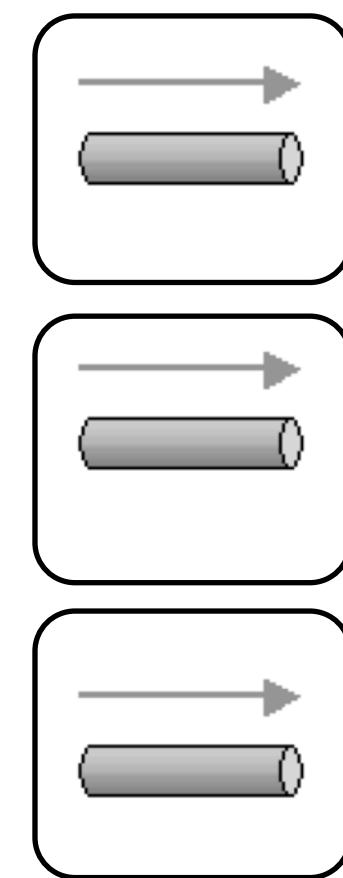
4000  
msg/sec

event  
producer

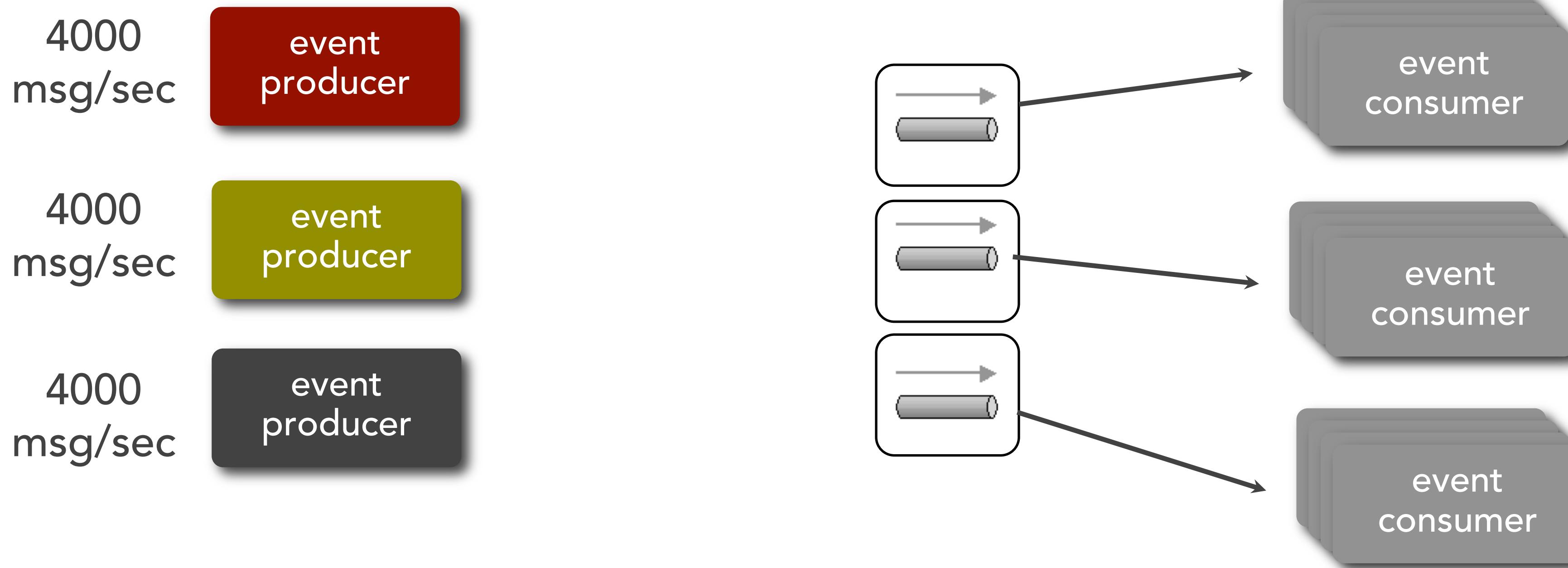
4000  
msg/sec

event  
producer

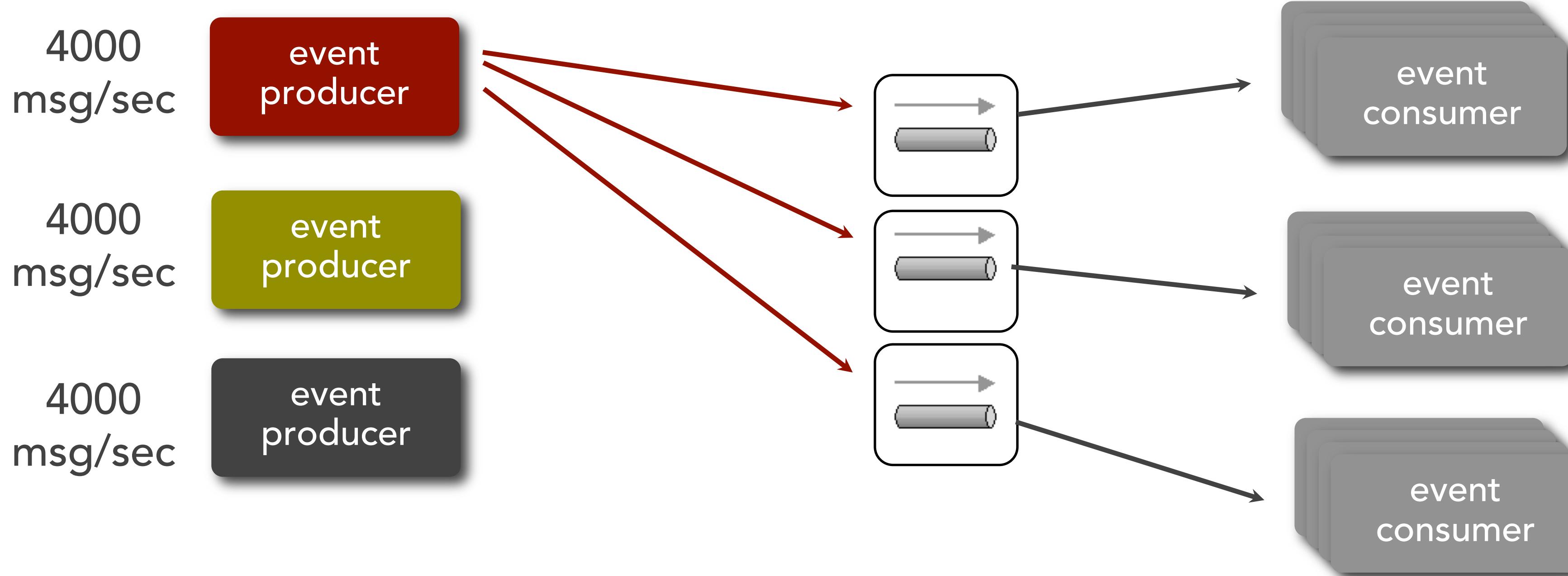
# multi-broker pattern



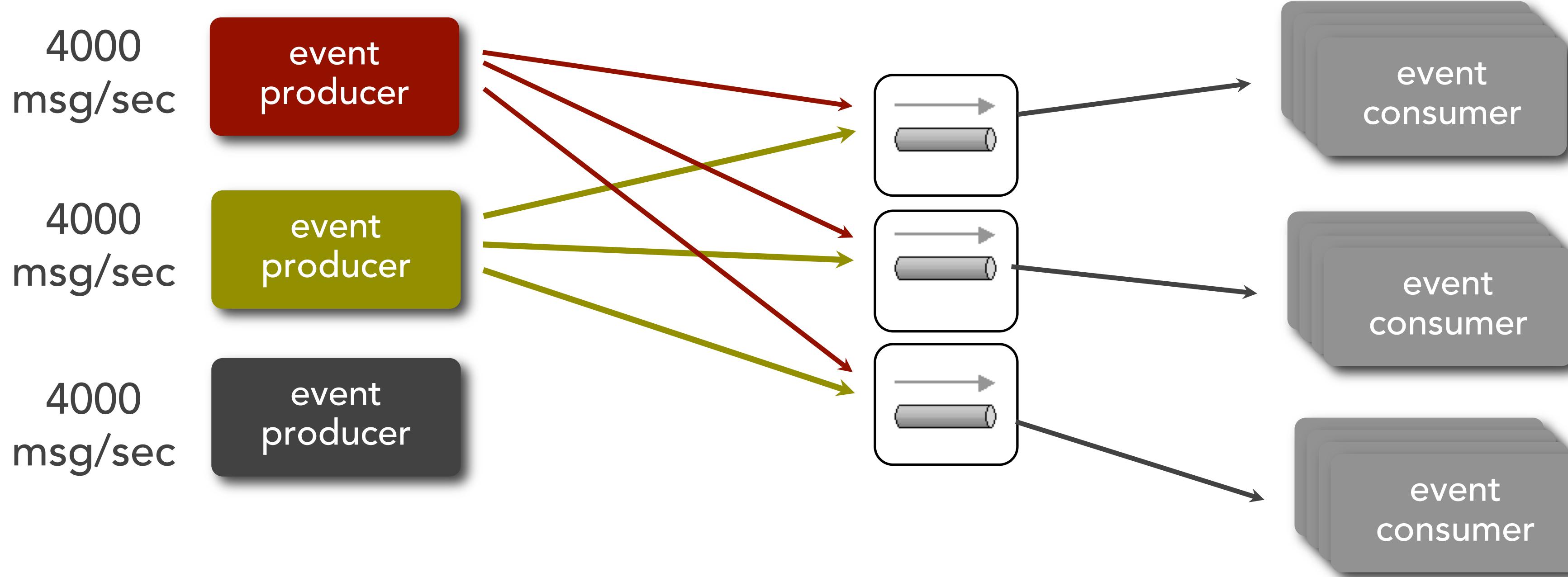
# multi-broker pattern



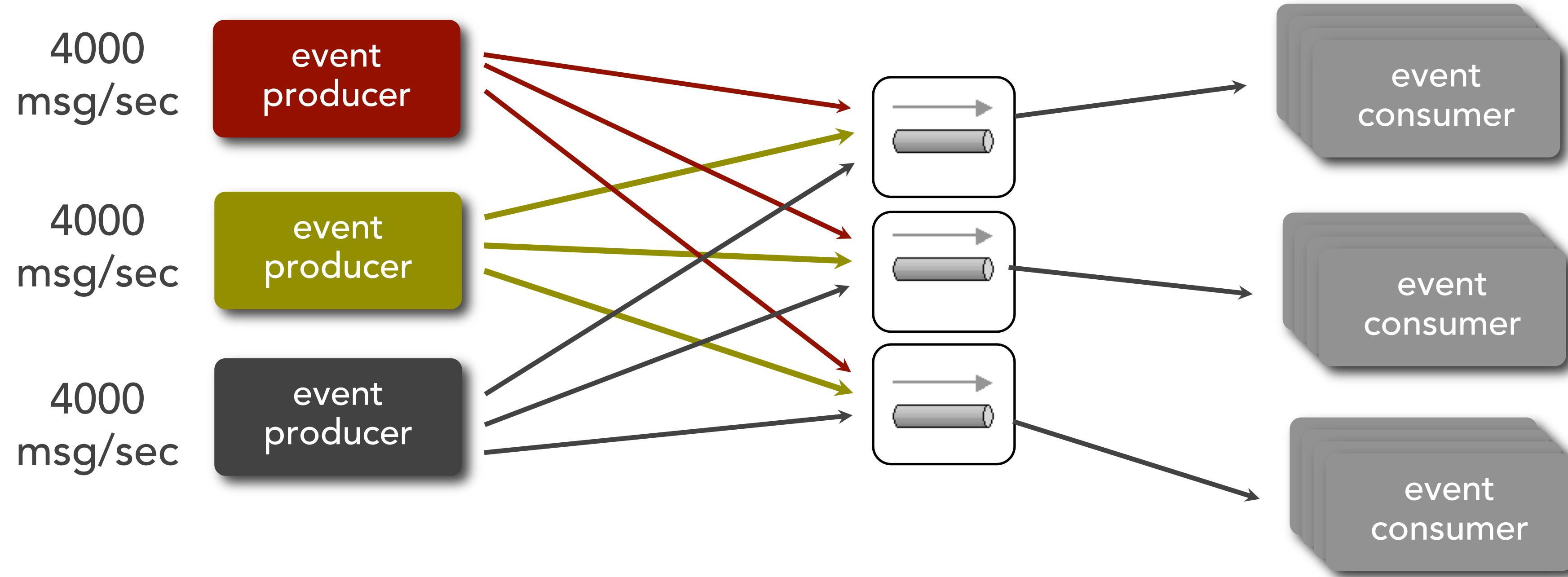
# multi-broker pattern



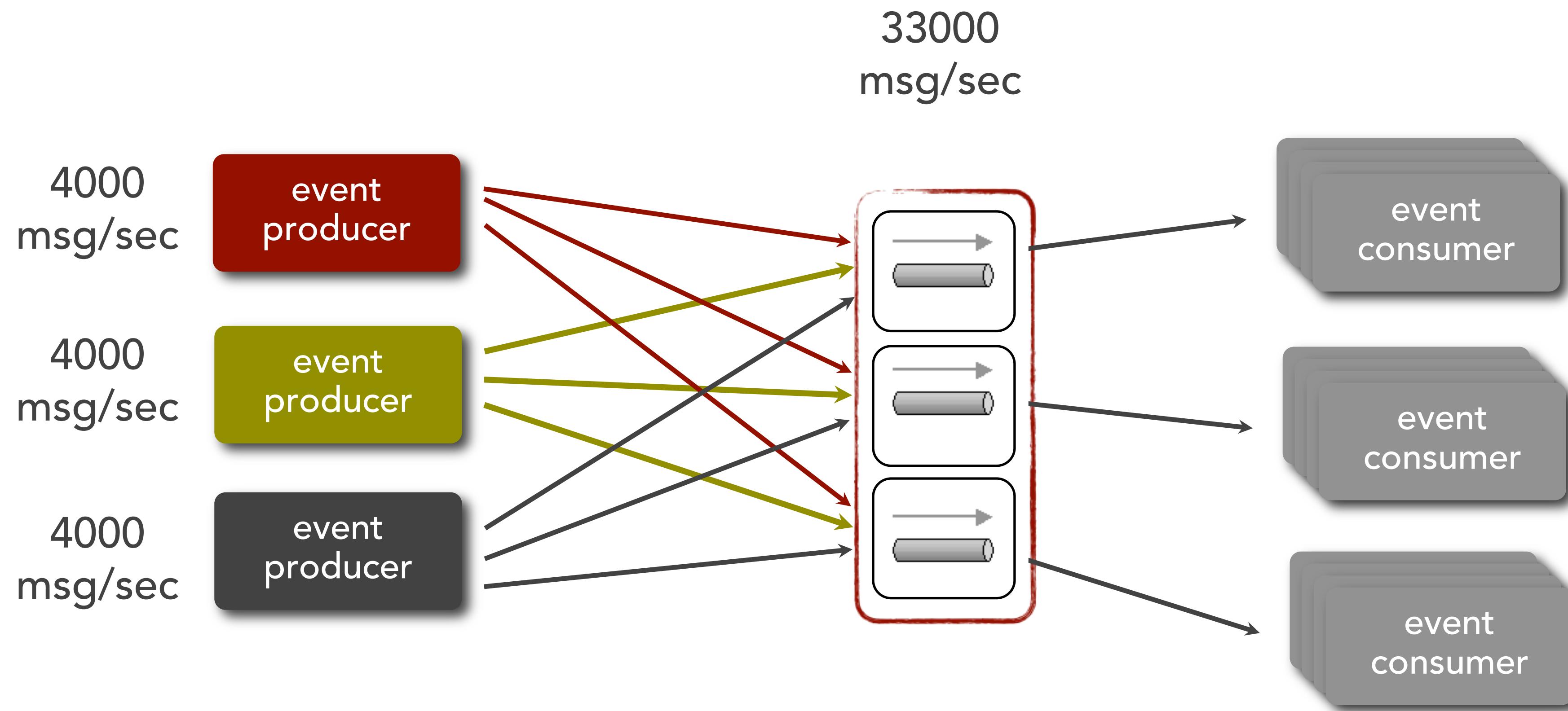
# multi-broker pattern



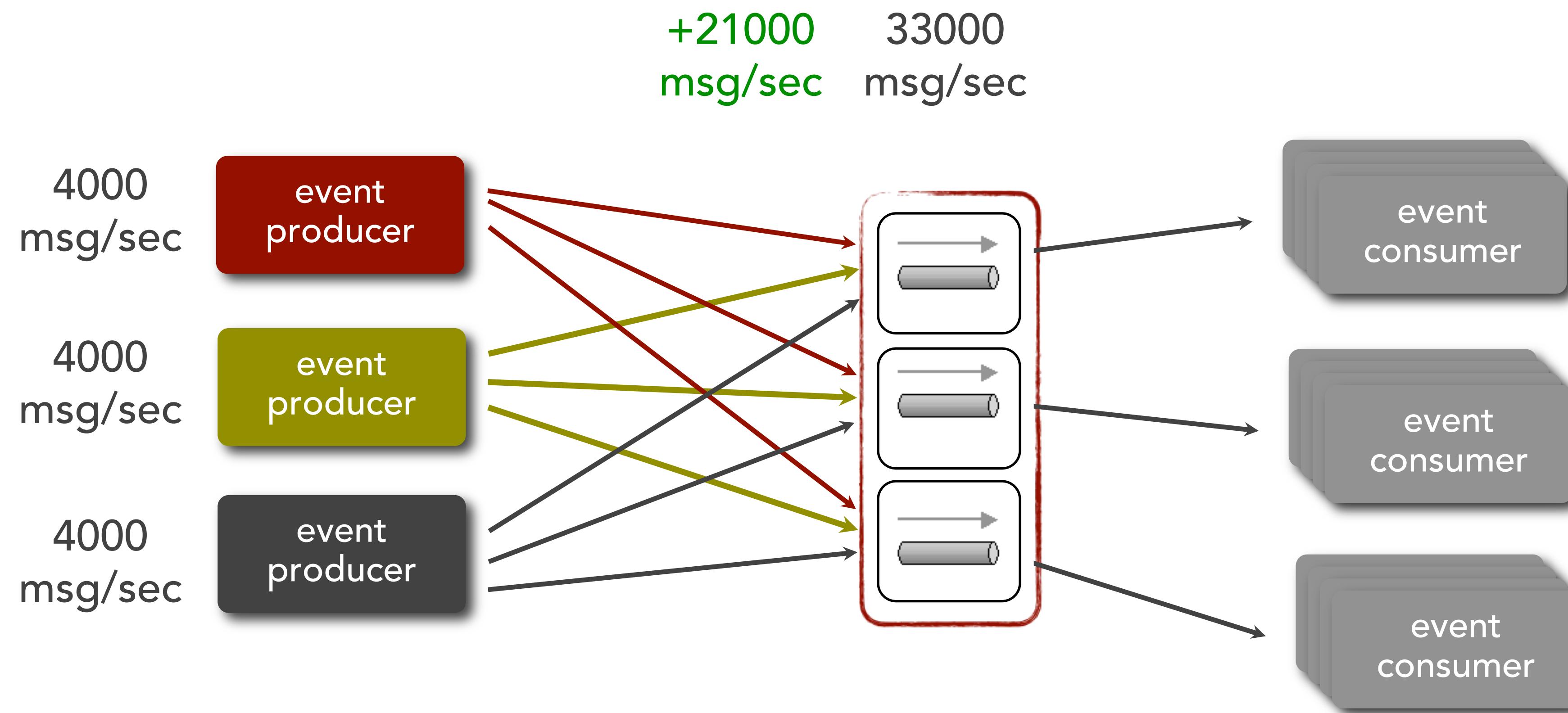
# multi-broker pattern



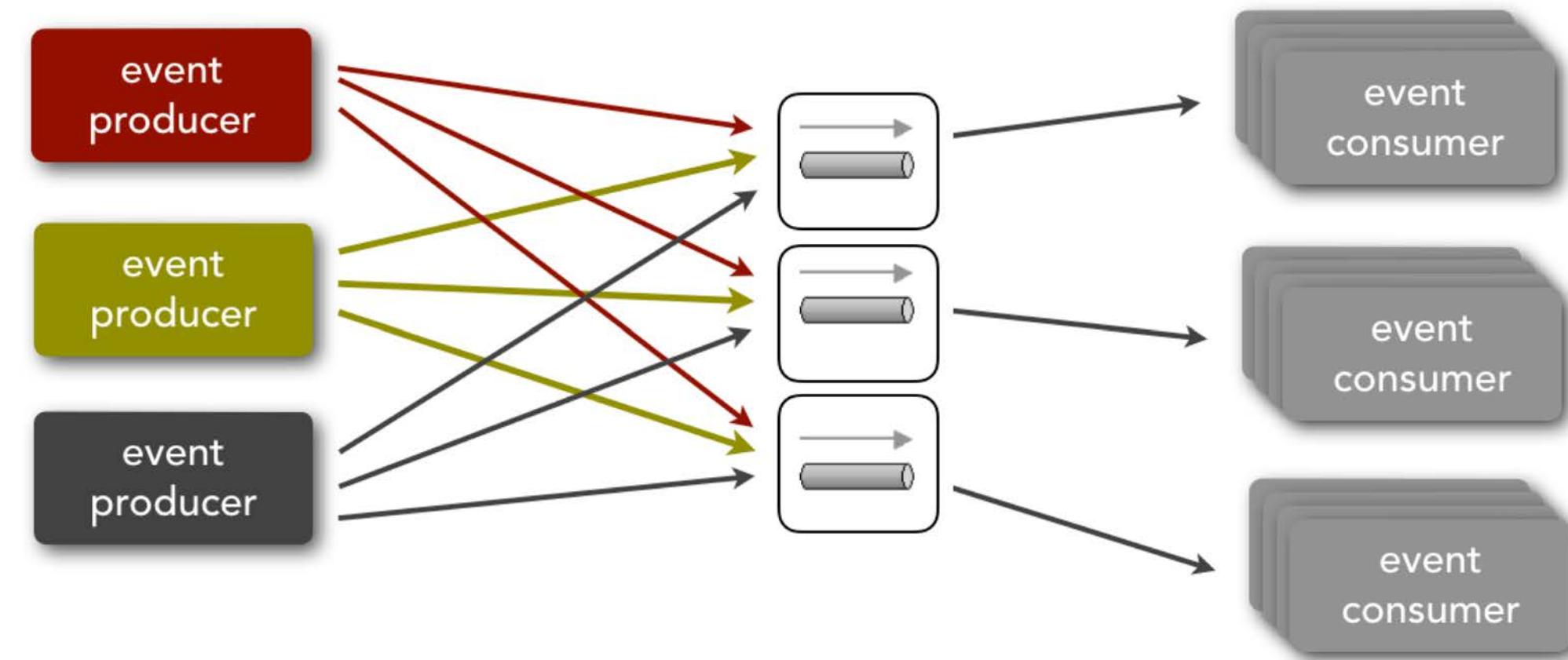
# multi-broker pattern



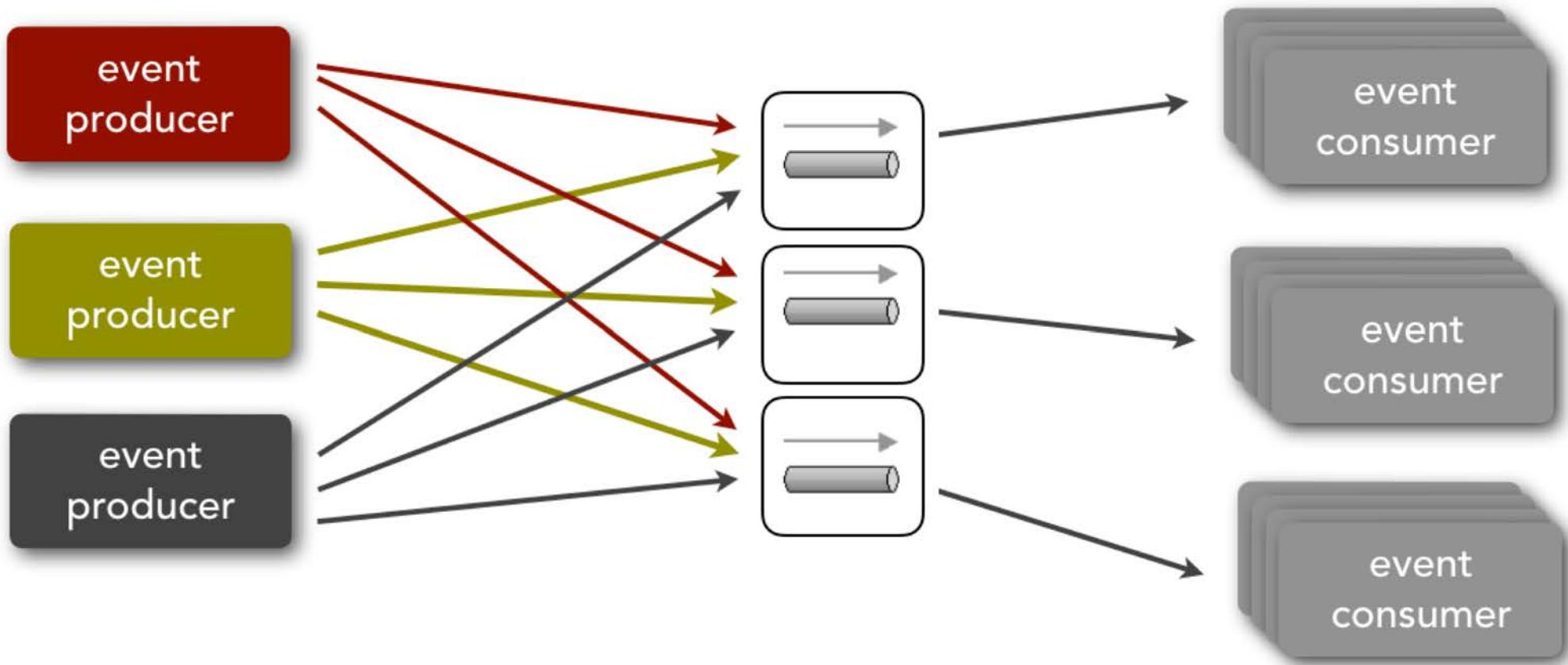
# multi-broker pattern



# multi-broker pattern



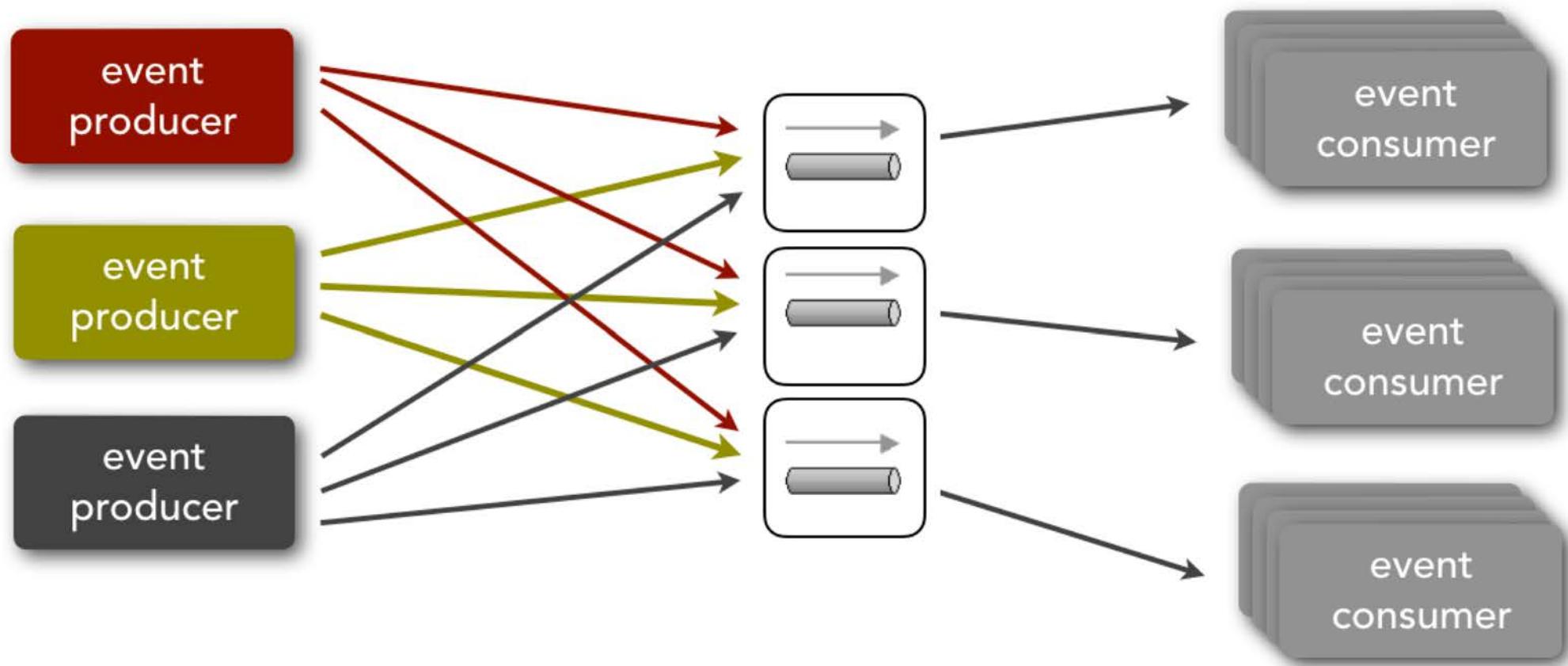
# multi-broker pattern



throughput  
performance  
scalability  
capacity



# multi-broker pattern



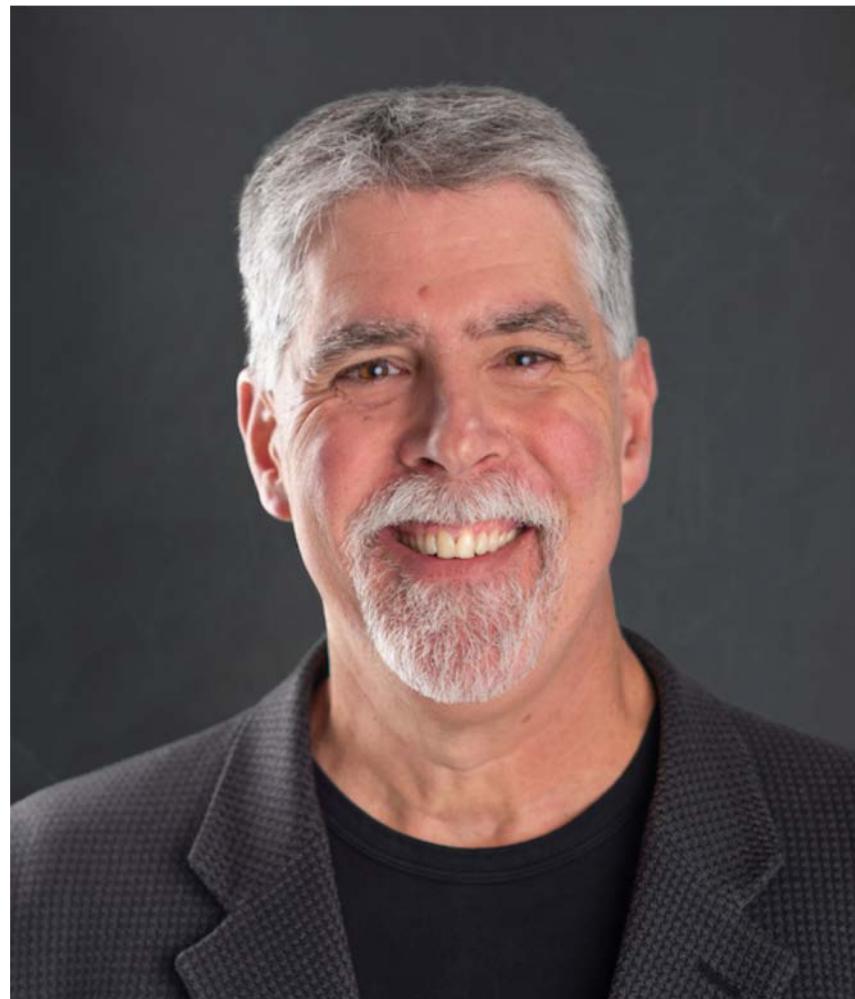
throughput  
performance  
scalability  
capacity



complexity  
cost



# Mastering Patterns in Event-Driven Architecture



**Mark Richards**

**Independent Consultant**

Hands-on Software Architect / Published Author

Founder, [DeveloperToArchitect.com](#)

<https://www.linkedin.com/in/markrichards3>

@markrichardssa