Algorithms and Distributed Systems 2019/2020 (Lab Six)

MIEI - Integrated Master in Computer Science and Informatics

Specialization block

João Leitão (jc.leitao@fct.unl.pt)

Class structure:

- Scribe Dissemination Protocol

(might be of use for the the Project)

- Support to the phase 2 of the project

SCRIBE

SCRIBE: The Design of a Large-Scale Event Notification Infrastructure

Antony Rowstron[1], Anne-Marie Kermarrec[1],
Miguel Castro[1], and Peter Druschel[2]

[1] Microsoft Research
7 J J Thomson Avenue, Cambridge, CB3 0FB, UK
{antr,anne-mk,mcastro}@microsoft.com
[2] Rice University MS-132, 6100 Main Street
Houston, TX 77005-1892, USA
druschel@cs.rice.edu

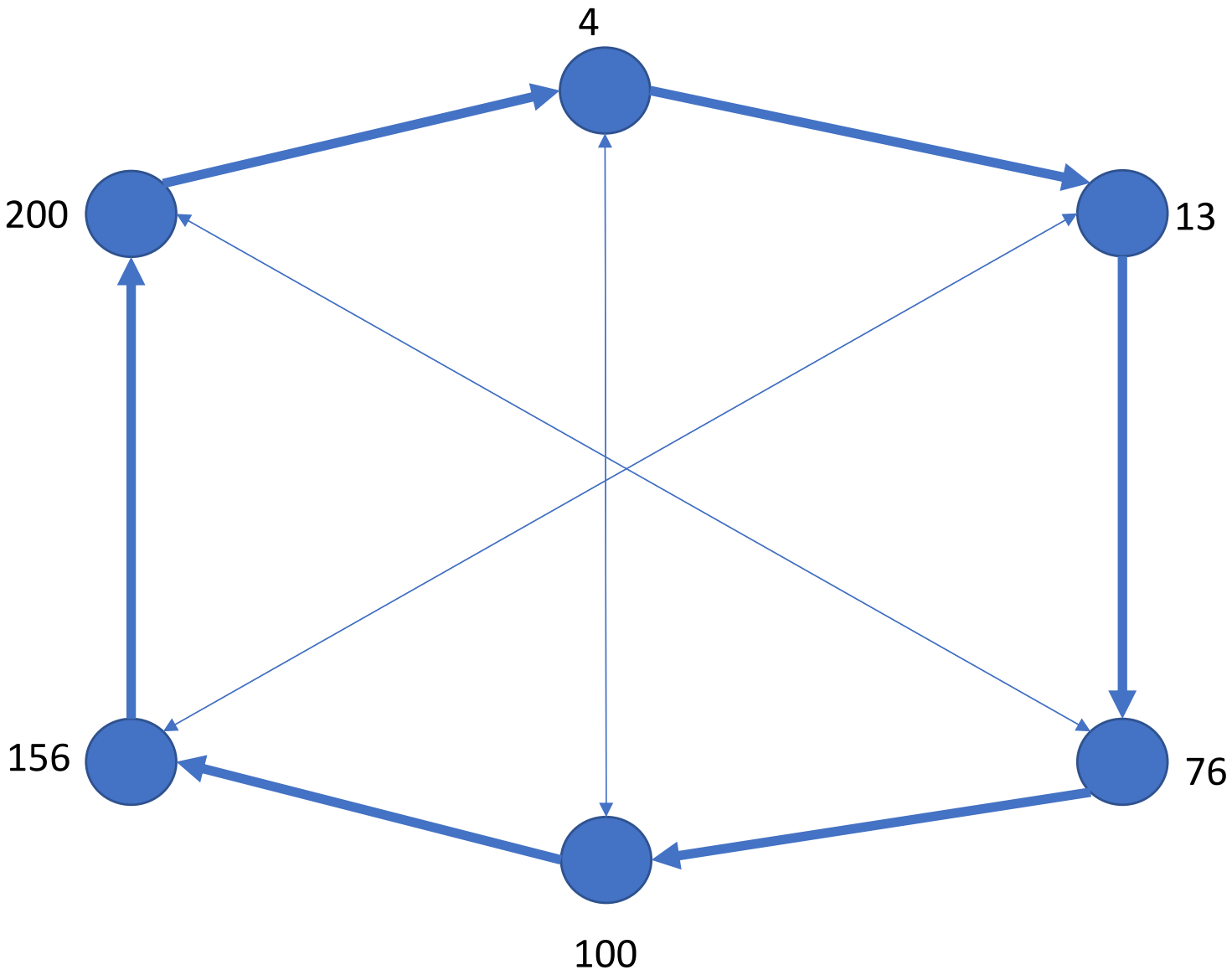
Abstract. This paper presents Scribe, a large-scale event notification infrastructure for topic-based publish-subscribe applications. Scribe supports large numbers of topics, with a potentially large number of subscribers per topic. Scribe is built on top of Pastry, a generic peer-to-peer object location and routing substrate overlayed on the Internet, and leverages Pastry's reliability, self-organization and locality properties. Pastry is used to create a topic (group) and to build an efficient multicast tree for the dissemination of events to the topic's subscribers (members). Scribe provides weak reliability guarantees, but we outline how an application can extend Scribe to provide stronger ones.

SCRIBE: The intuition

- Before understanding how Scribe operates, we must start by considering the easiest way to build a publish-subscribe system on top of a DHT.

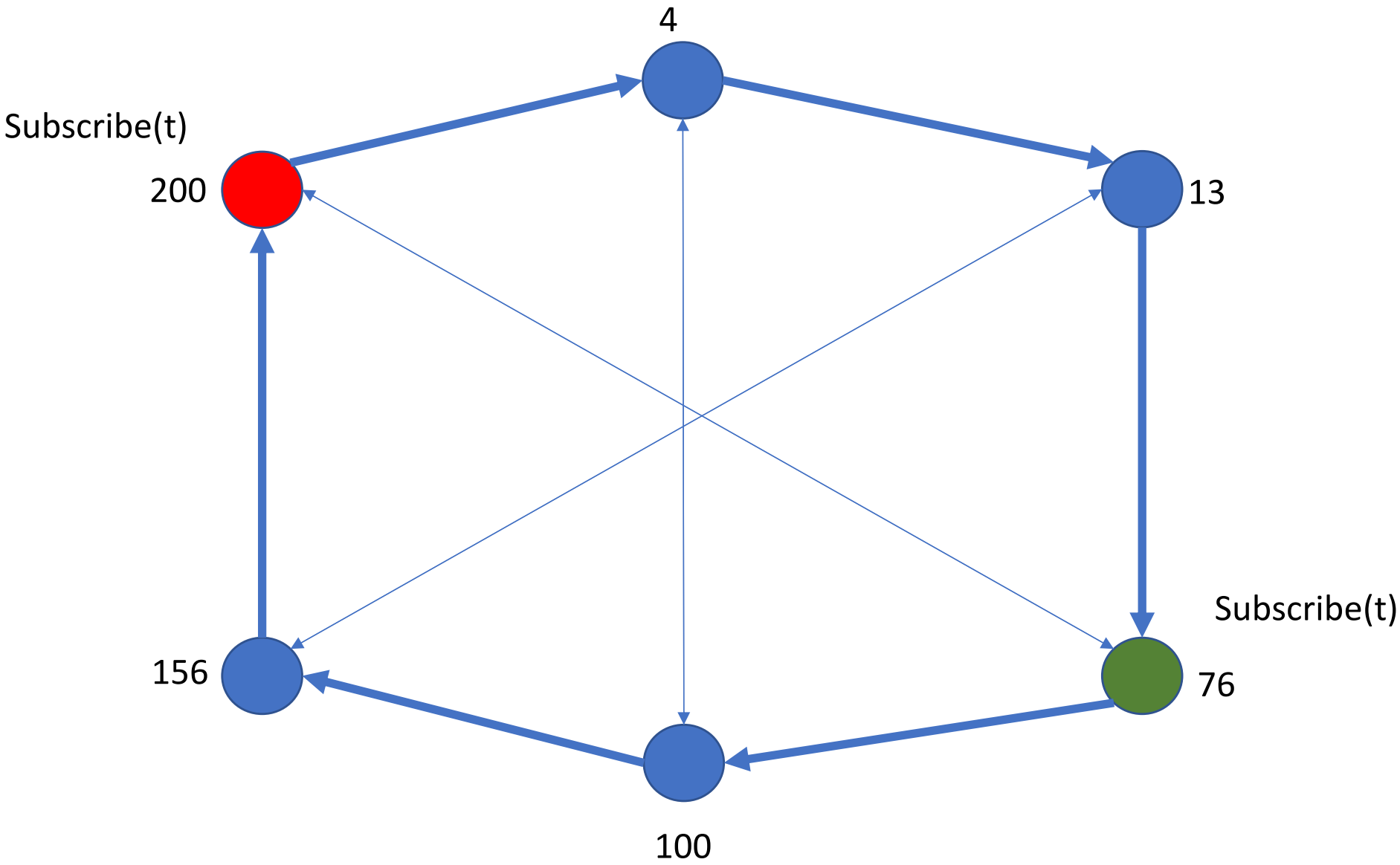
- How would you do it?

SCRIBE: The intuition

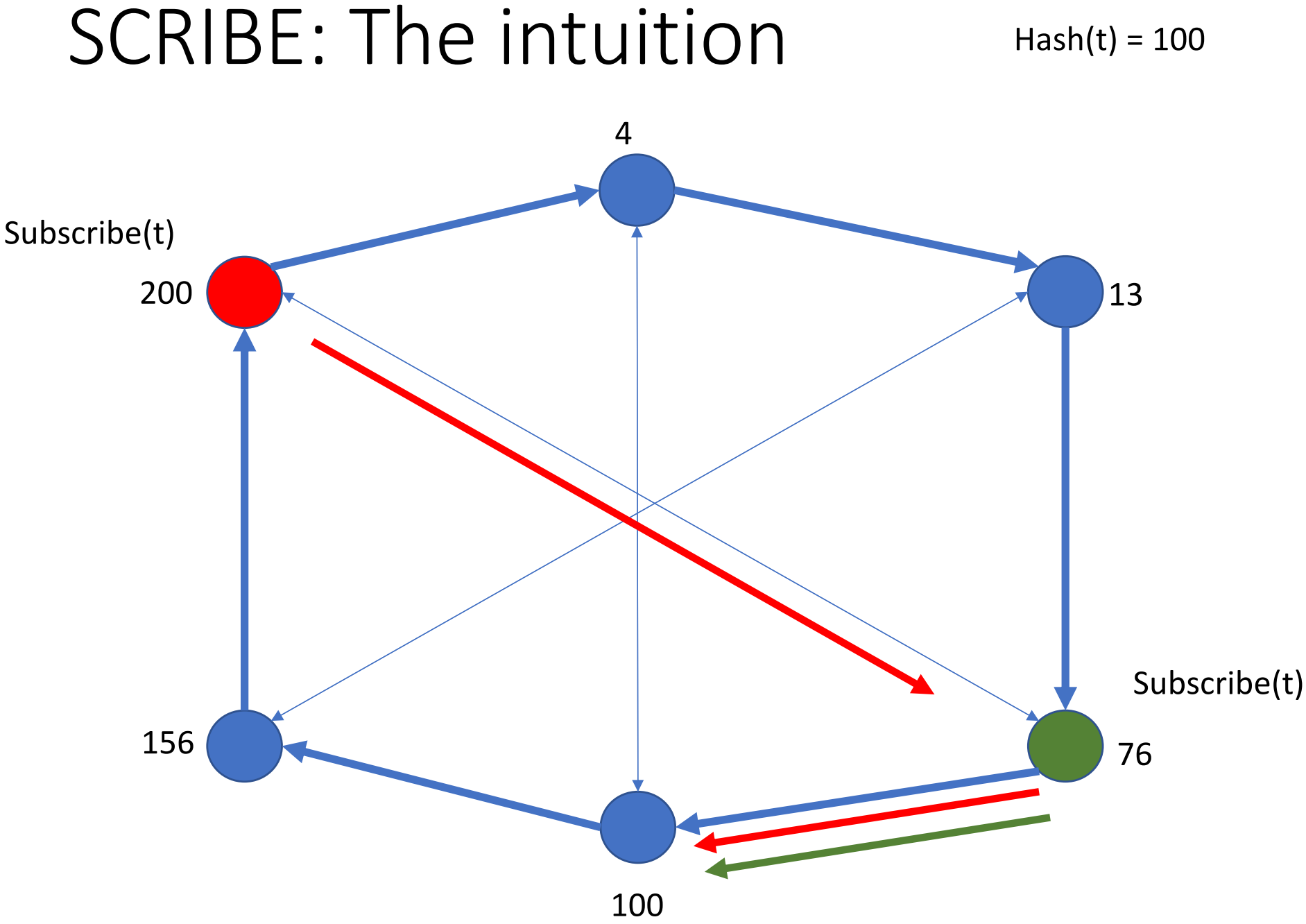


SCRIBE: The intuition

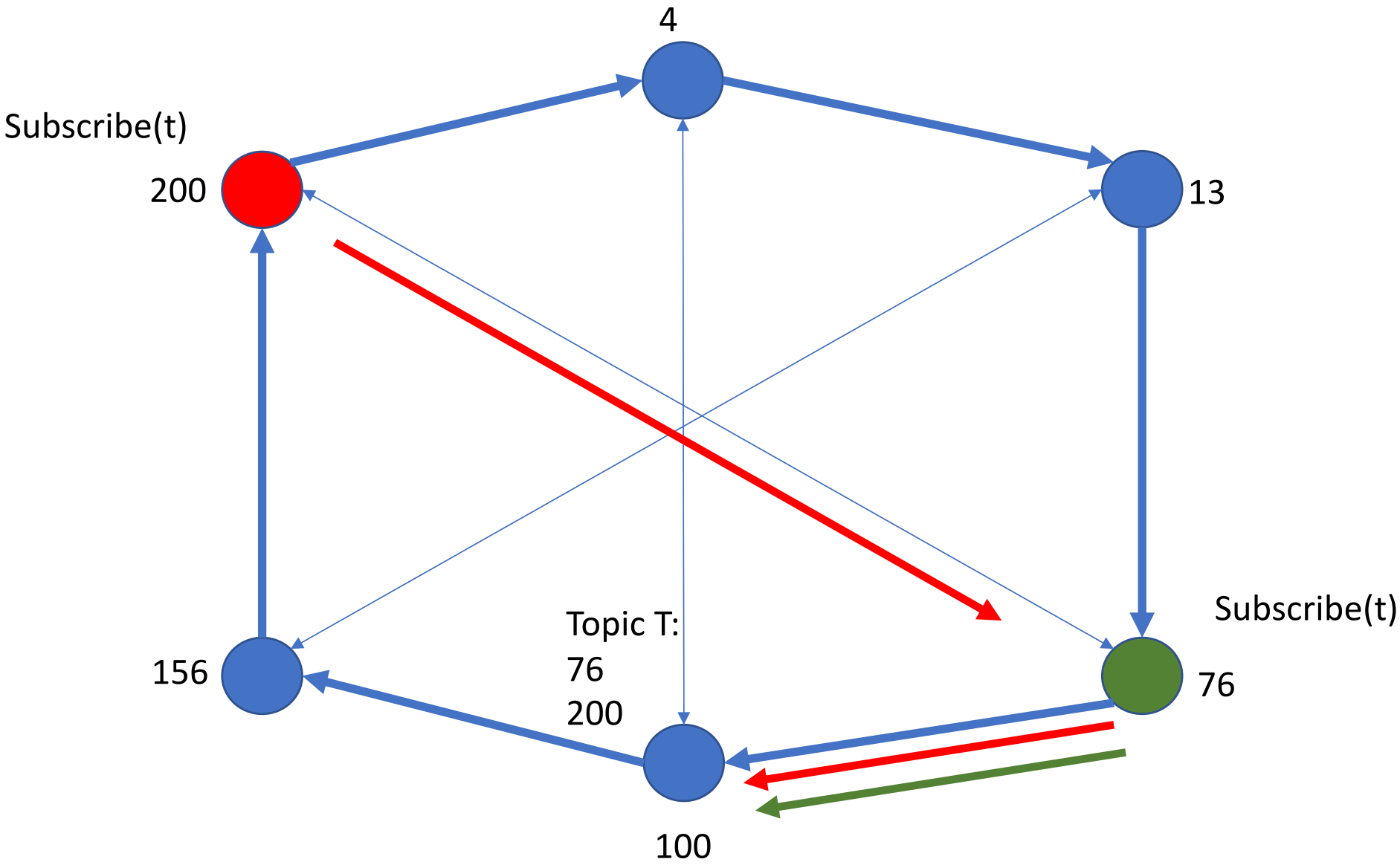
Hash(t) = 100



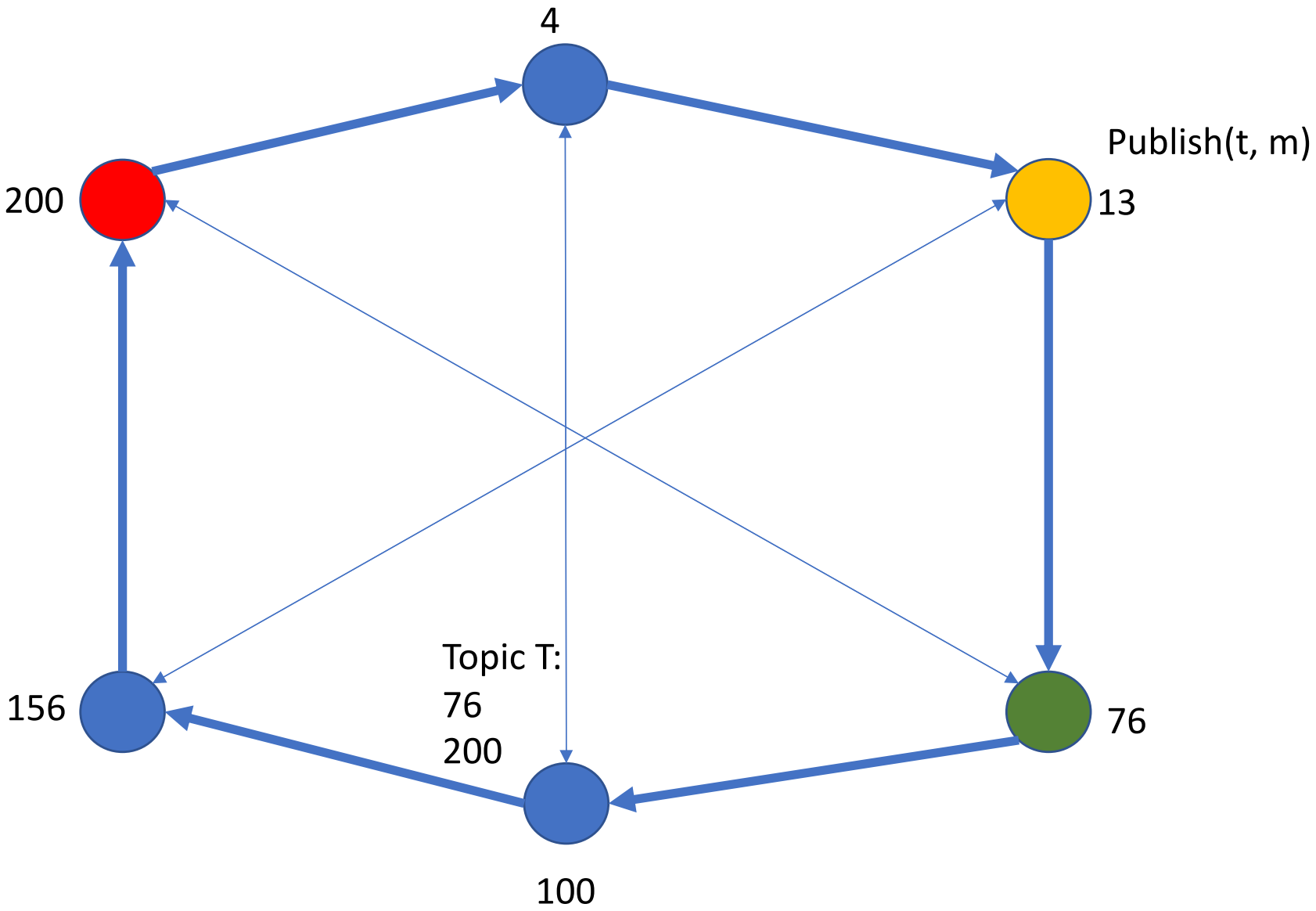
SCRIBE: The intuition



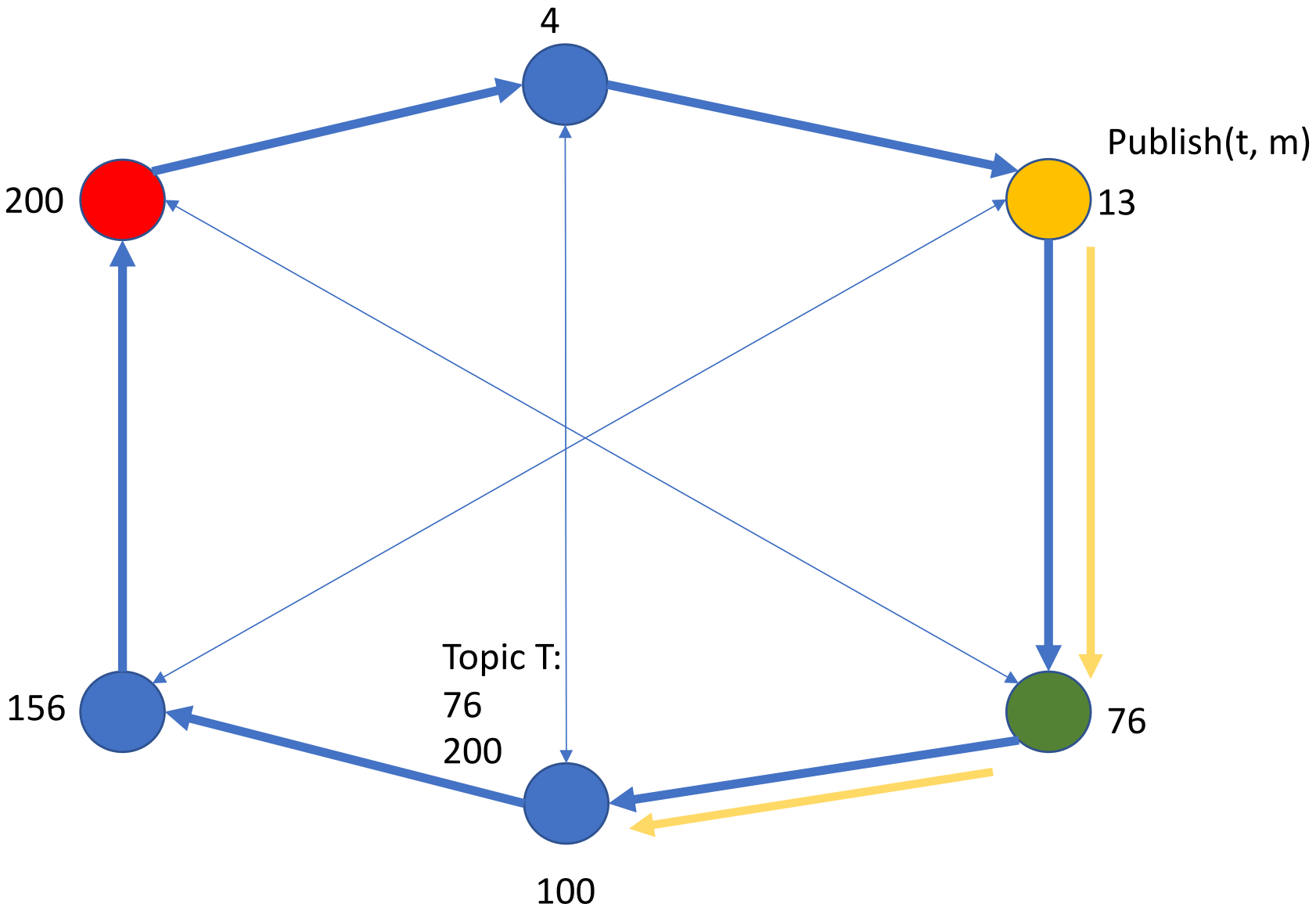
SCRIBE: The intuition



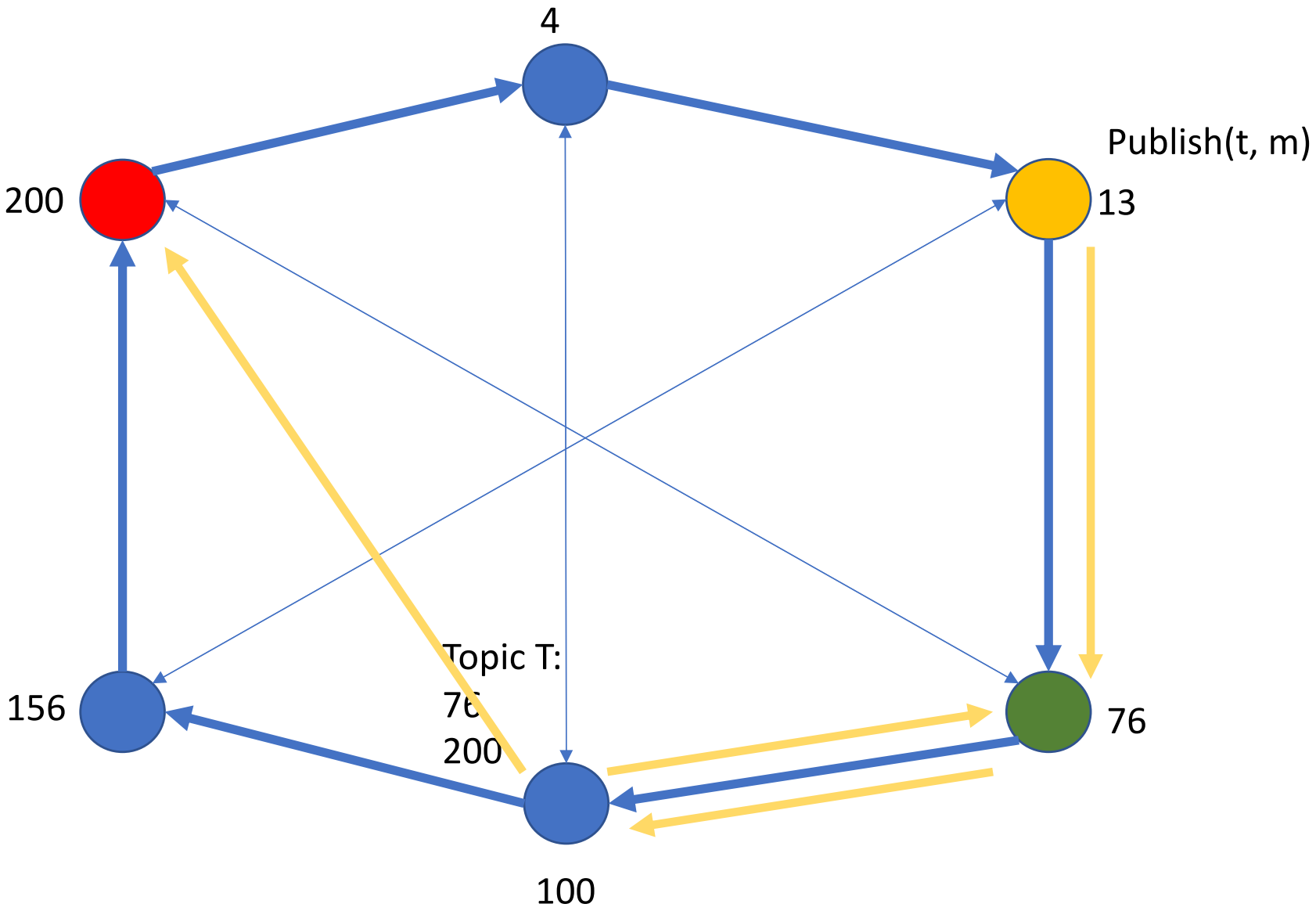
SCRIBE: The intuition



SCRIBE: The intuition



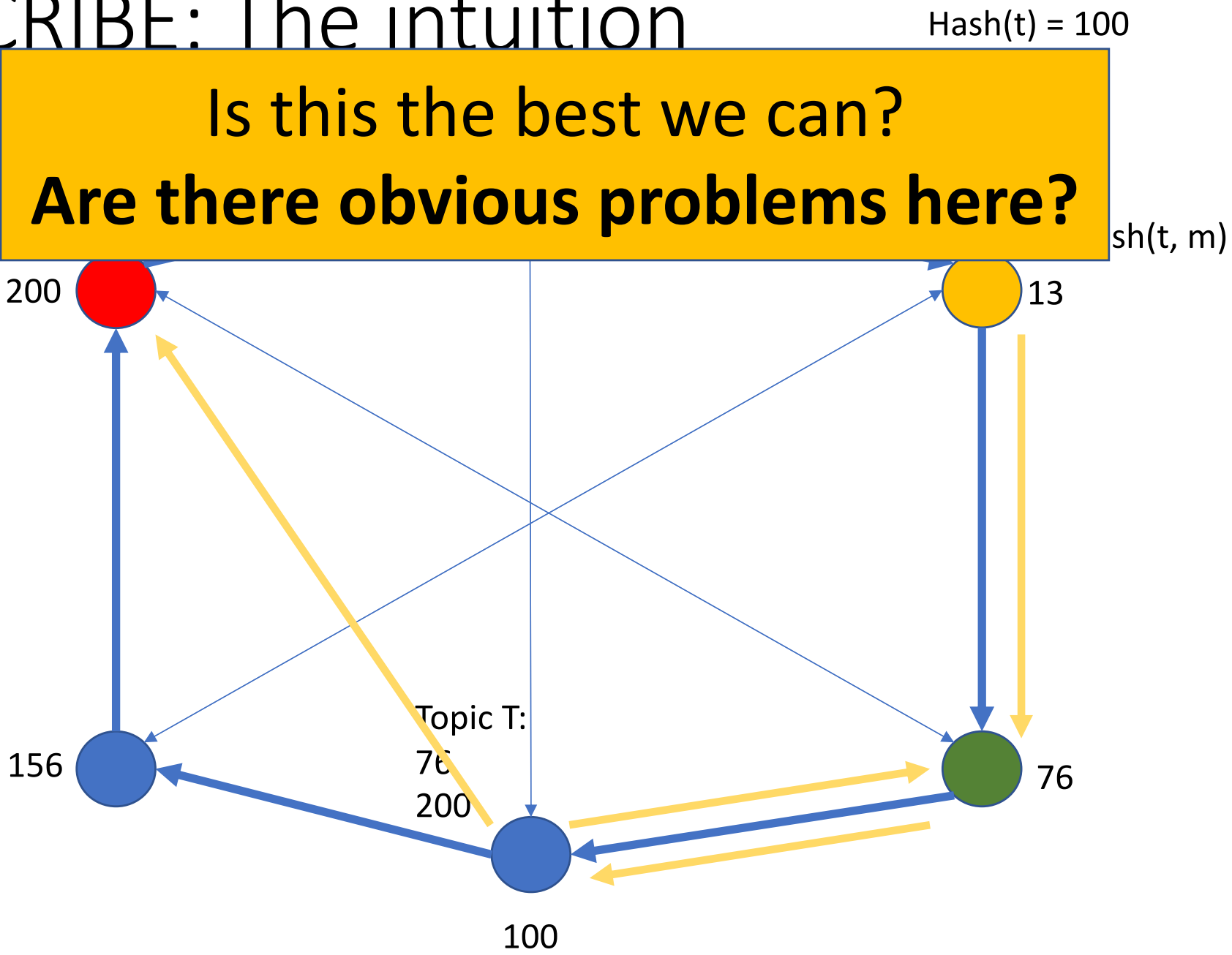
SCRIBE: The intuition



SCRIBE: The intuition

Is this the best we can?

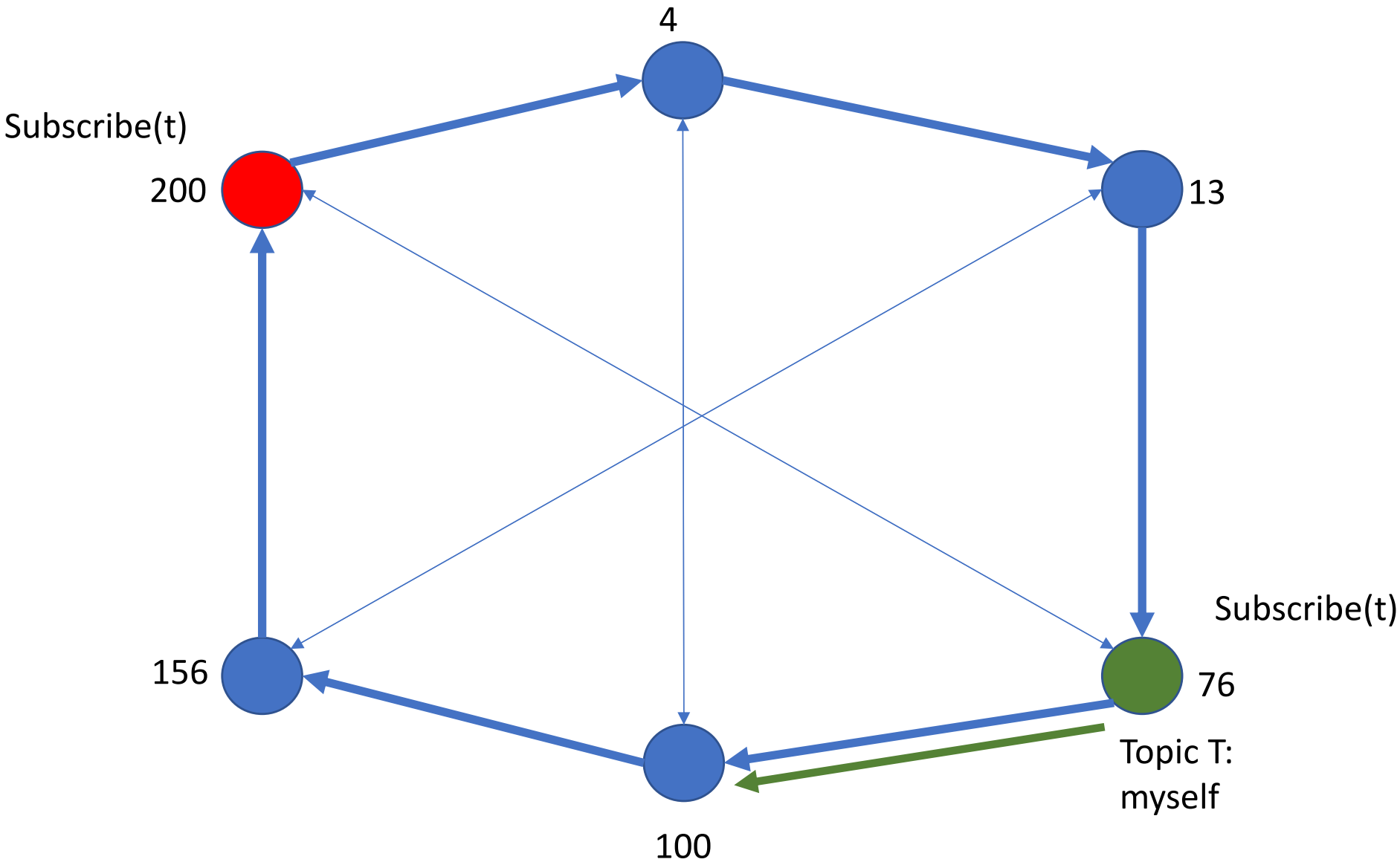
Are there obvious problems here?



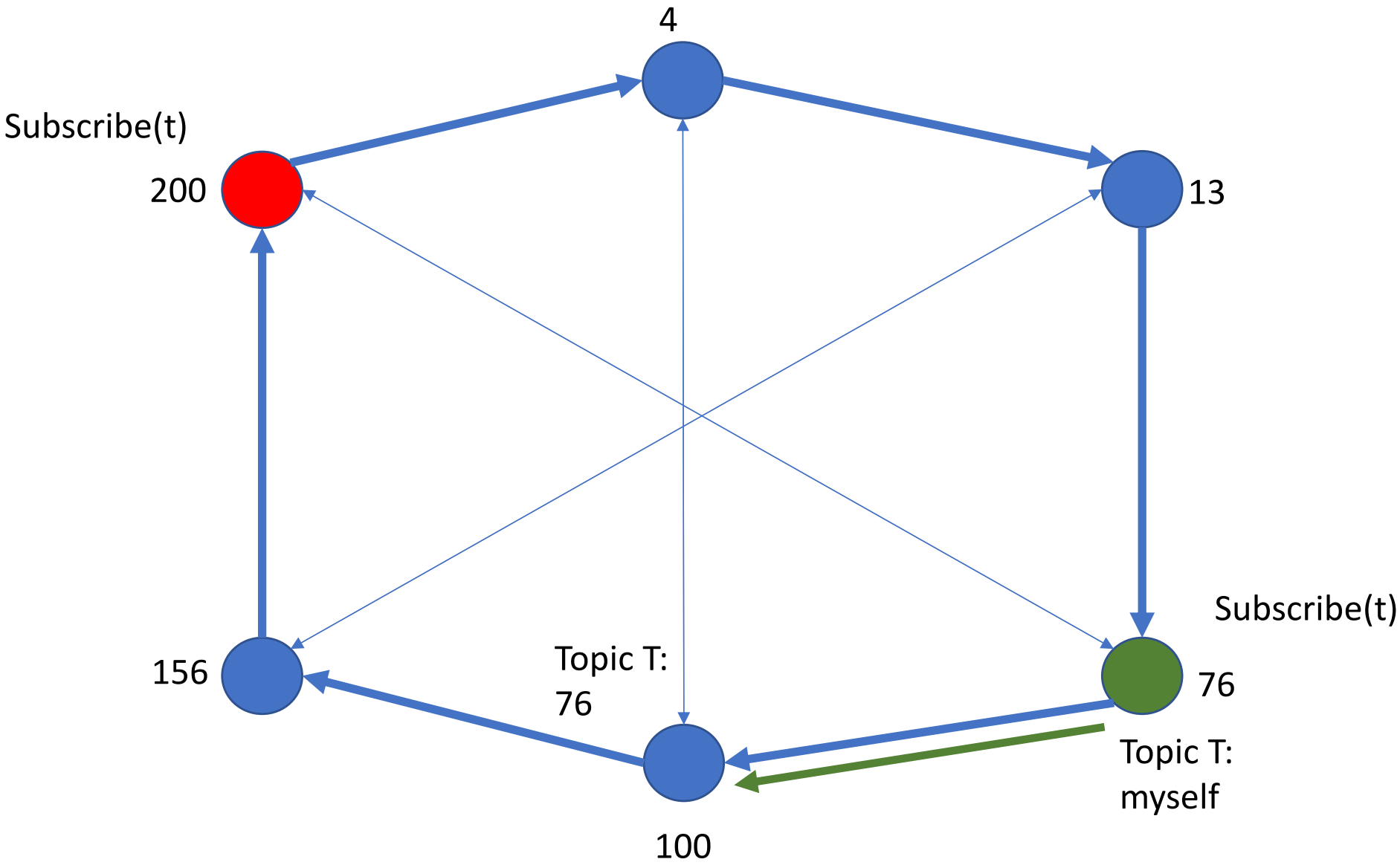
SCRIBE: The intuition

- As subscribe messages are routed through the overlay nodes can memorize the path for the subscriber on each topic
- As soon as the subscribe gets to a node that already had information for the topic being subscribed the message can stop
- Effectively this builds a tree for each topic t, that can then be used to disseminate messages published in t

SCRIBE: The intuition

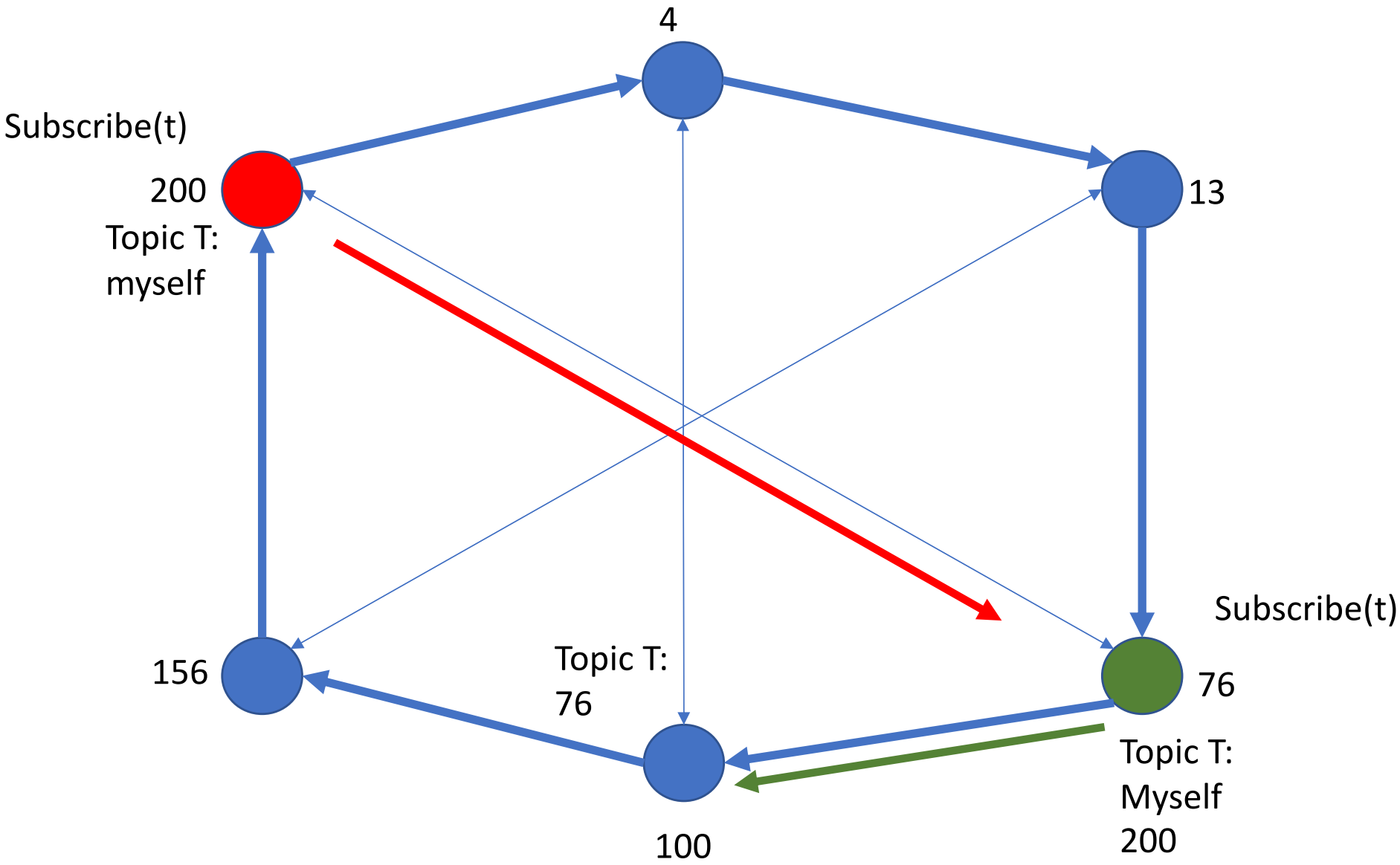


SCRIBE: The intuition

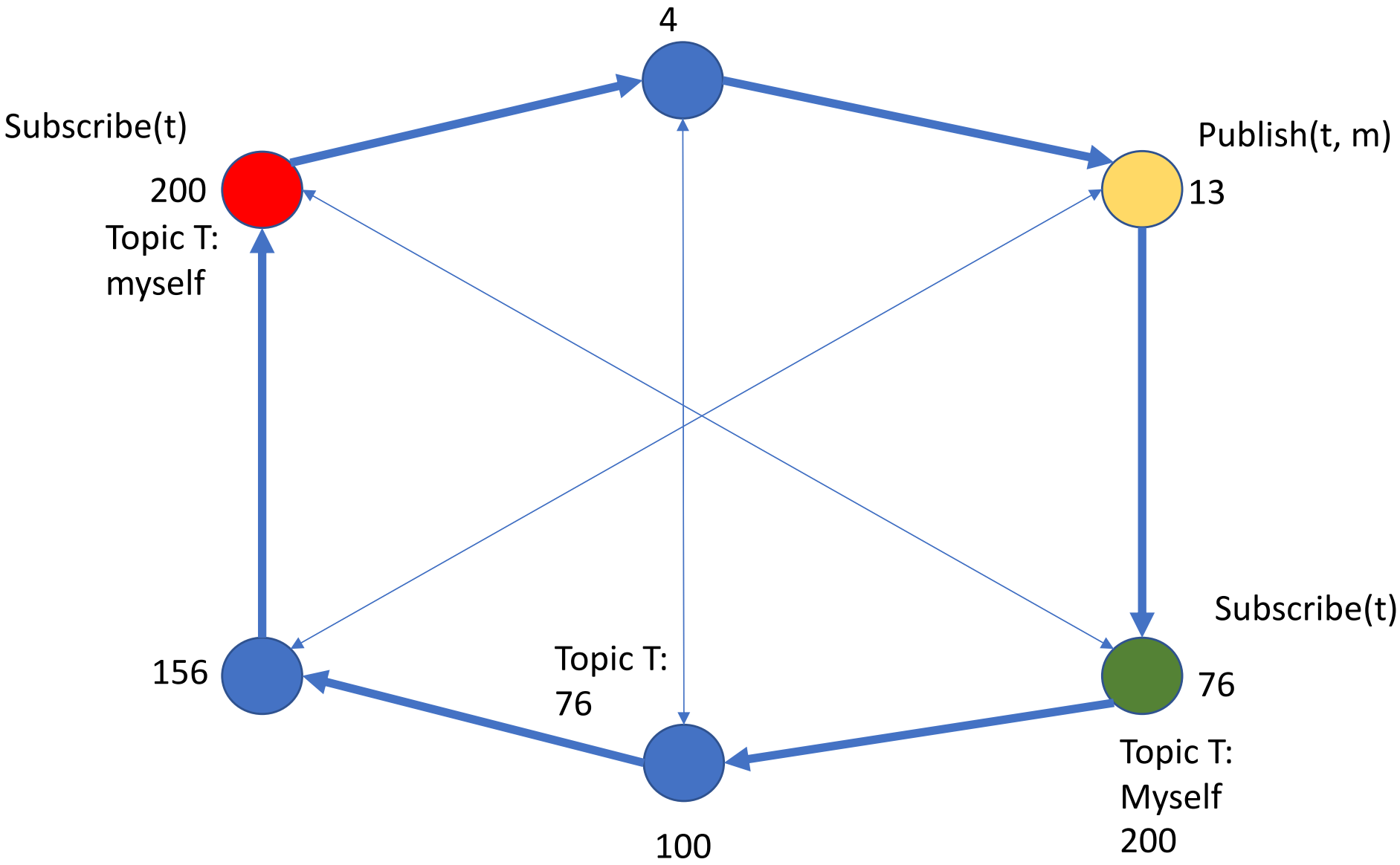


SCRIBE: The intuition

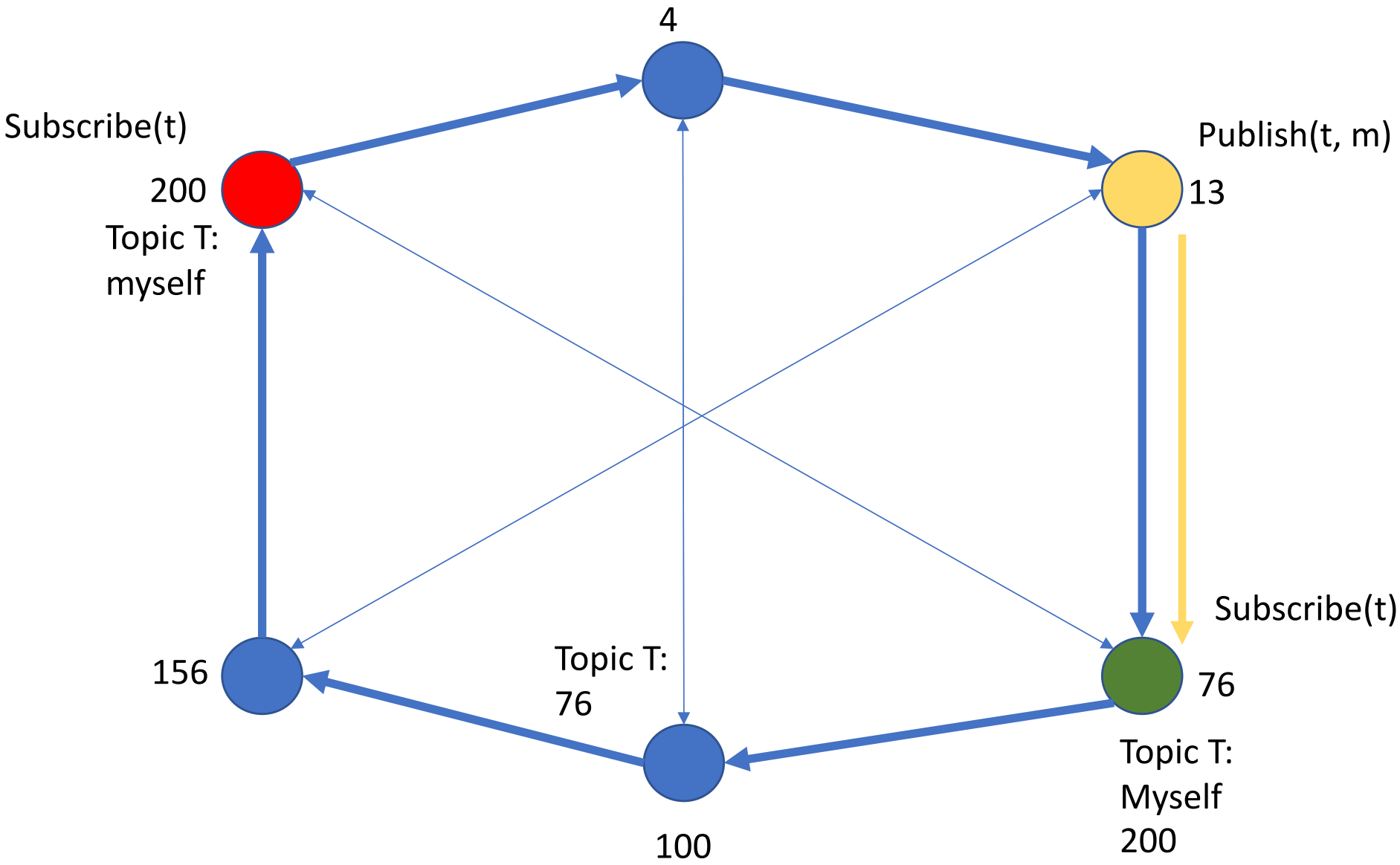
Hash(t) = 100



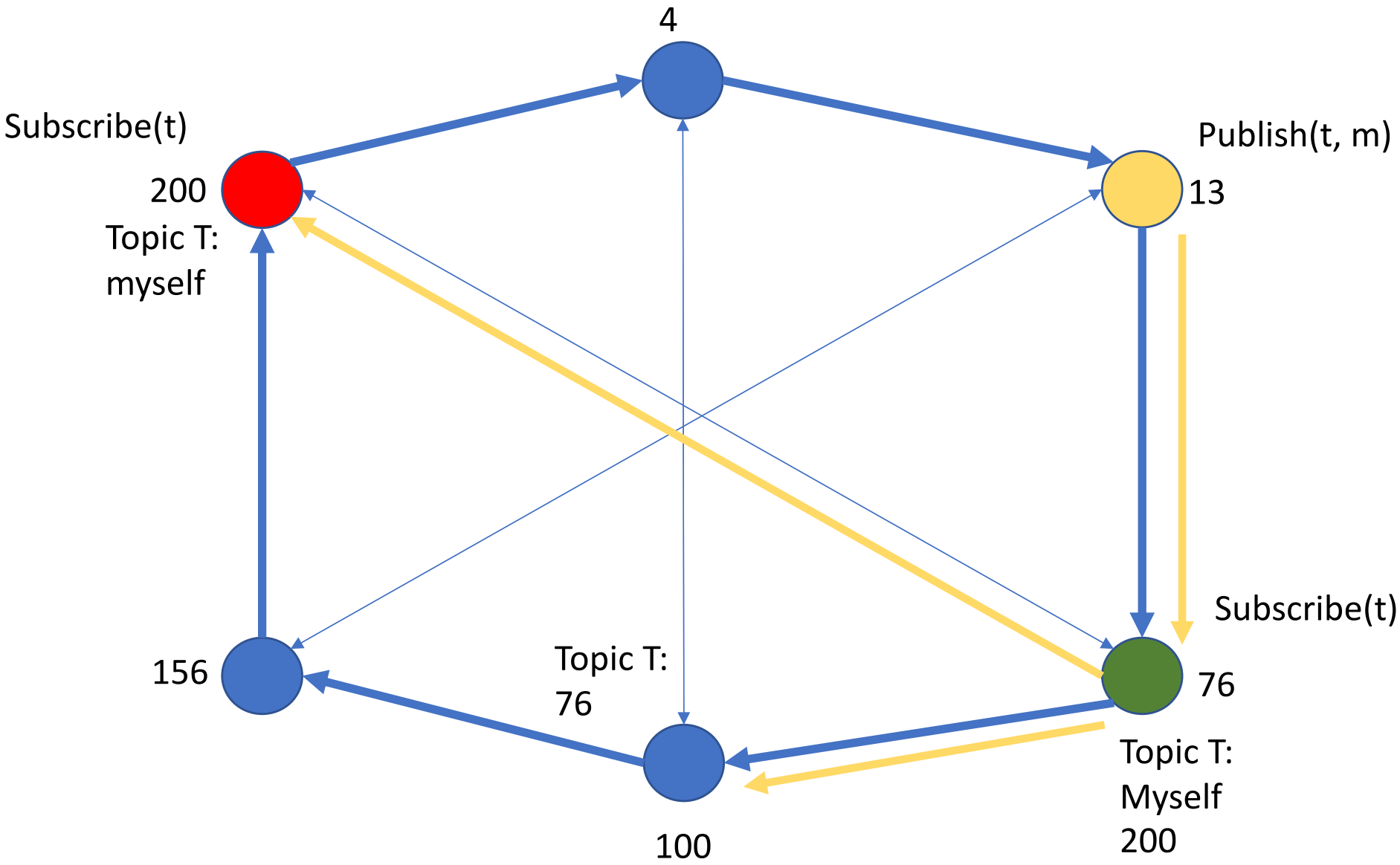
SCRIBE: The intuition



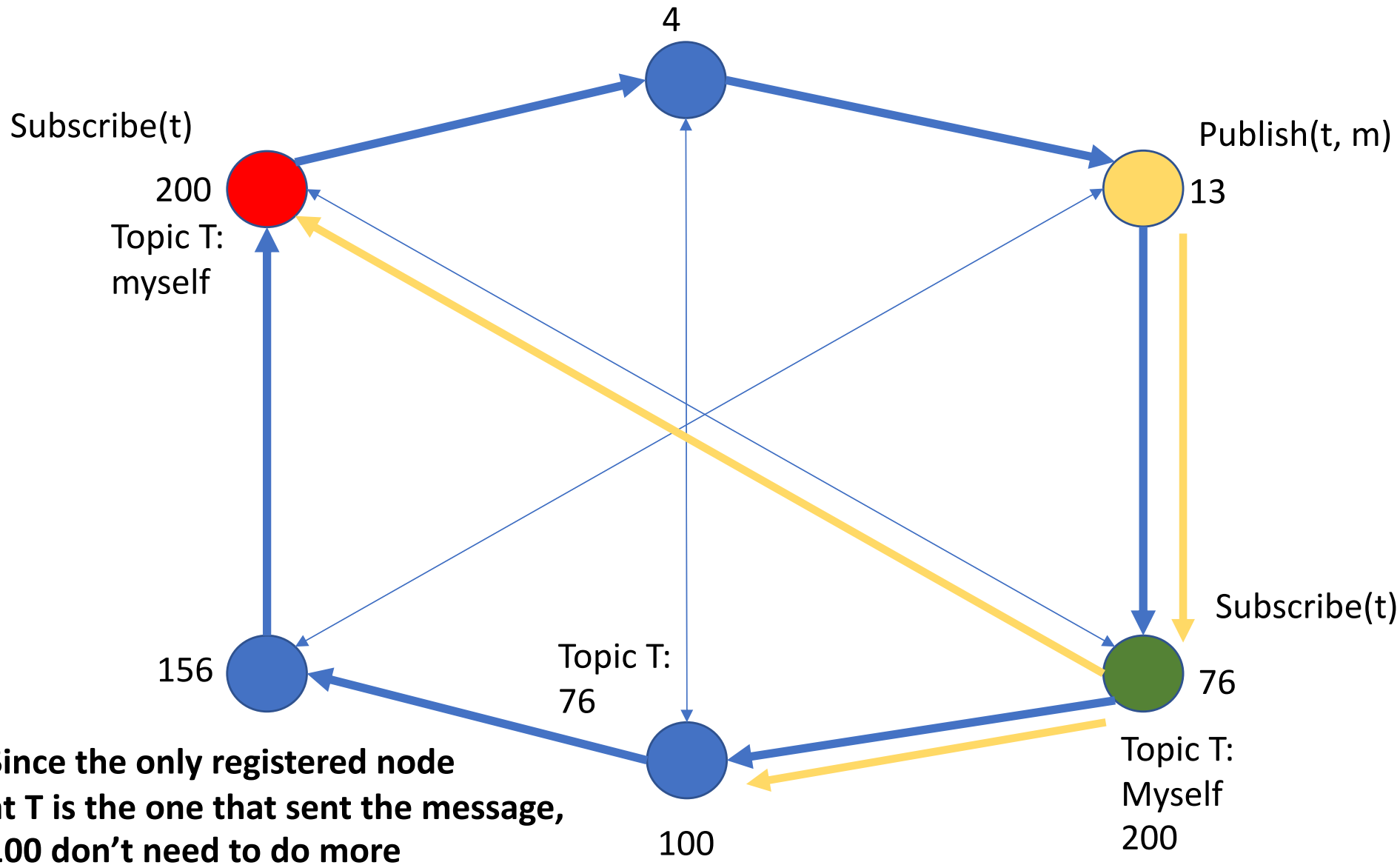
SCRIBE: The intuition



SCRIBE: The intuition



SCRIBE: The intuition



Since the only registered node at T is the one that sent the message, 100 don't need to do more

SCRIBE: The algorithm

- The algorithm assumes that the DHT (Chord) is modified to notify the publish-subscribe layer (Scribe) of messages being routed through it.
  - It is essential to allow nodes to store state about subscriptions
  - It also allows to remove state when all nodes below a given node for a topic t unsubscribe
  - You can also optimize the path that messages when being published.

SCRIBE: The algorithm

```
(1) forward(msg, key, nextId)
(2)     switch msg.type is
(3)         SUBSCRIBE :if !(msg.topic ∈ topics)
(4)                        topics = topics ∪ msg.topic
(5)                        msg.source = thisNodeId
(6)                        route(msg,msg.topic)
(7)                    topics[msg.topic].children ∪ msg.source
(8)                    nextId = null  // Stop routing the original message
```

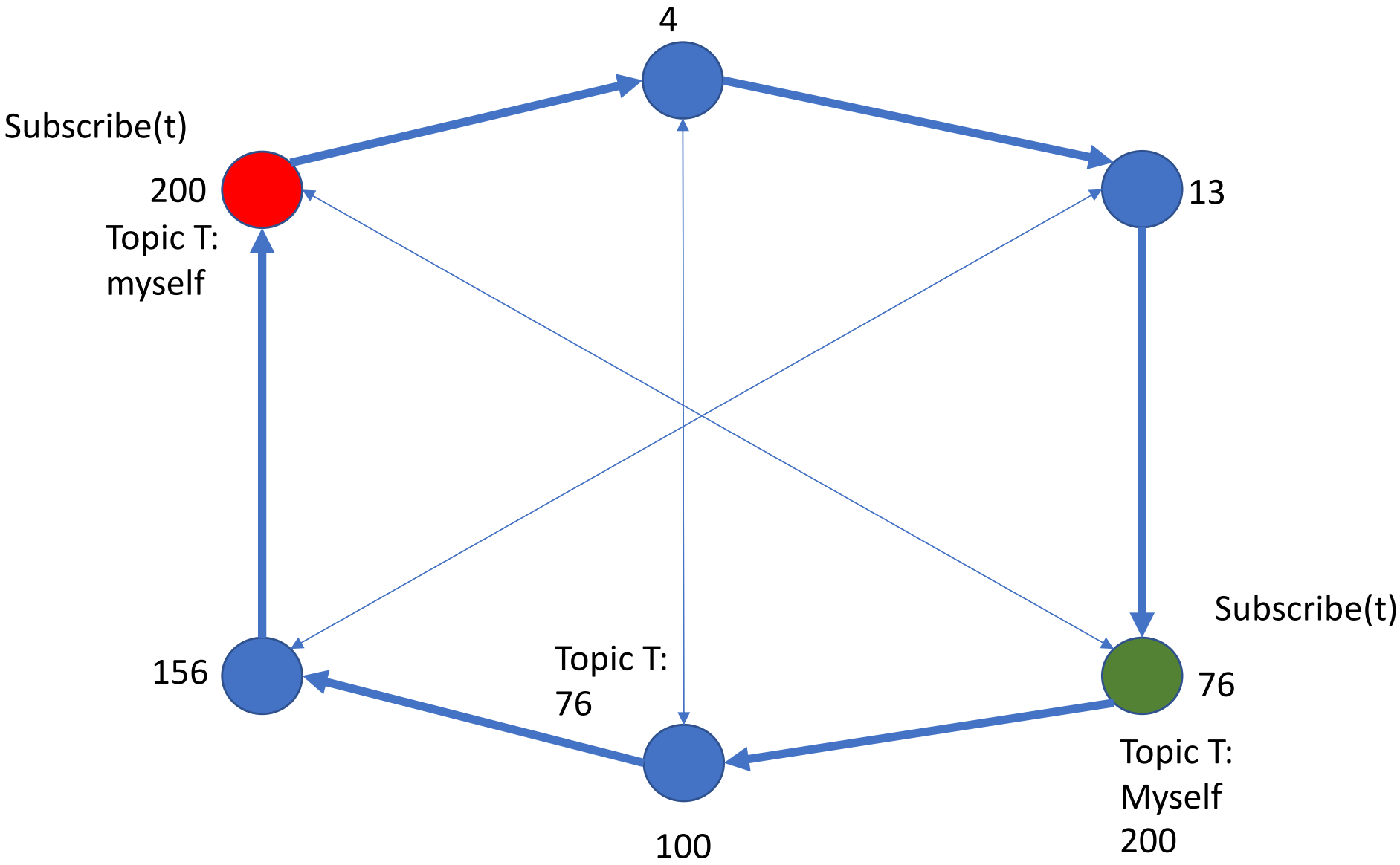
SCRIBE: The algorithm

```
(1) deliver(msg,key)
(2)     switch msg.type is
(3)         CREATE :      topics = topics ∪ msg.topic
(4)         SUBSCRIBE :   topics[msg.topic].children ∪ msg.source
(5)         PUBLISH :     ∀ node in topics[msg.topic].children
(6)                           send(msg,node)
(7)                       if subscribedTo(msg.topic)
(8)                           invokeEventHandler(msg.topic, msg)
(9)         UNSUBSCRIBE : topics[msg.topic].children =
                                  topics[msg.topic].children - msg.source
(10)                      if (|topics[msg.topic].children| = 0)
(11)                          msg.source = thisNodeId
(12)                          send(msg,topics[msg.topic].parent)
```

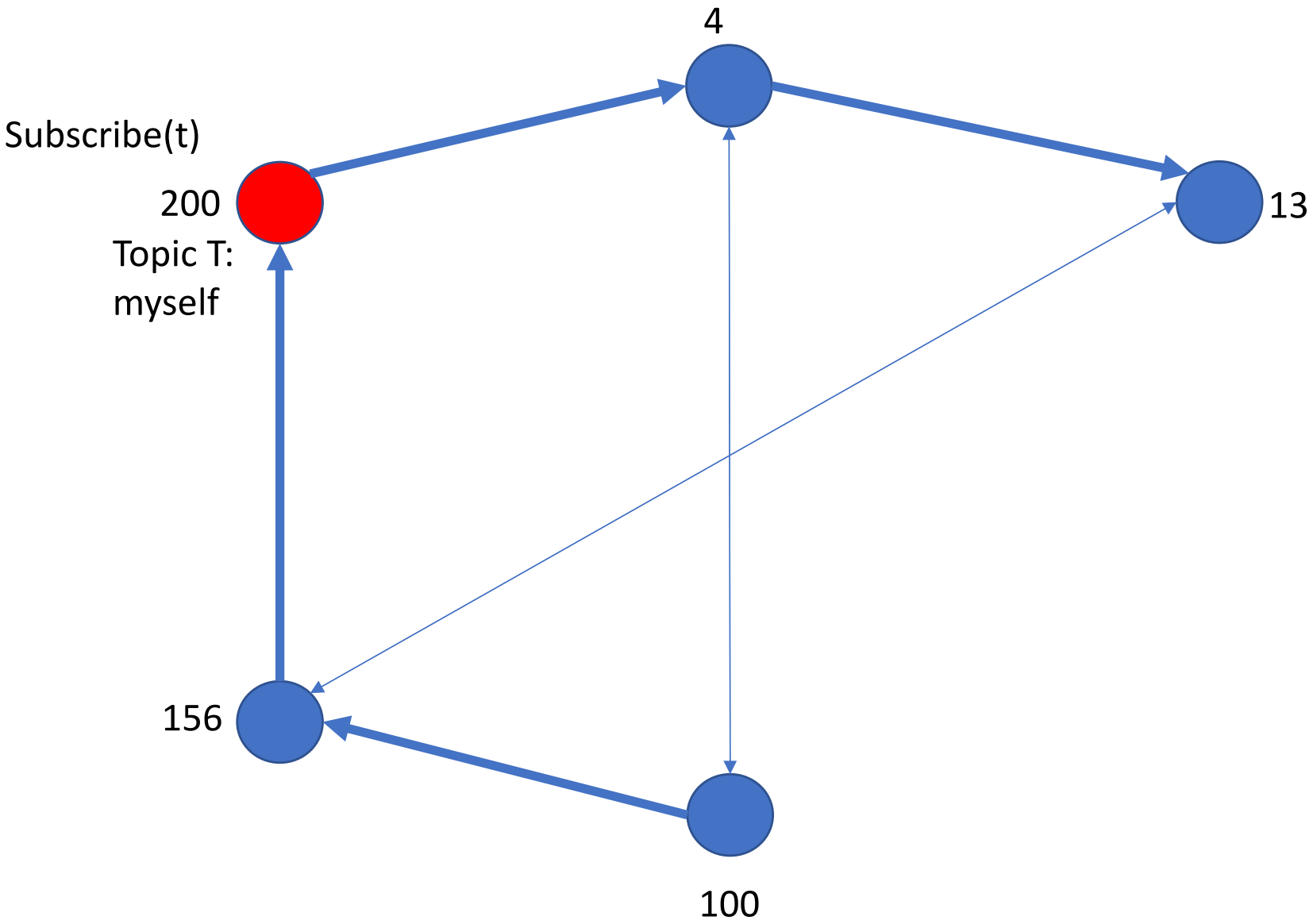
SCRIBE: Fault Tolerance

- What do you do if a node fails?

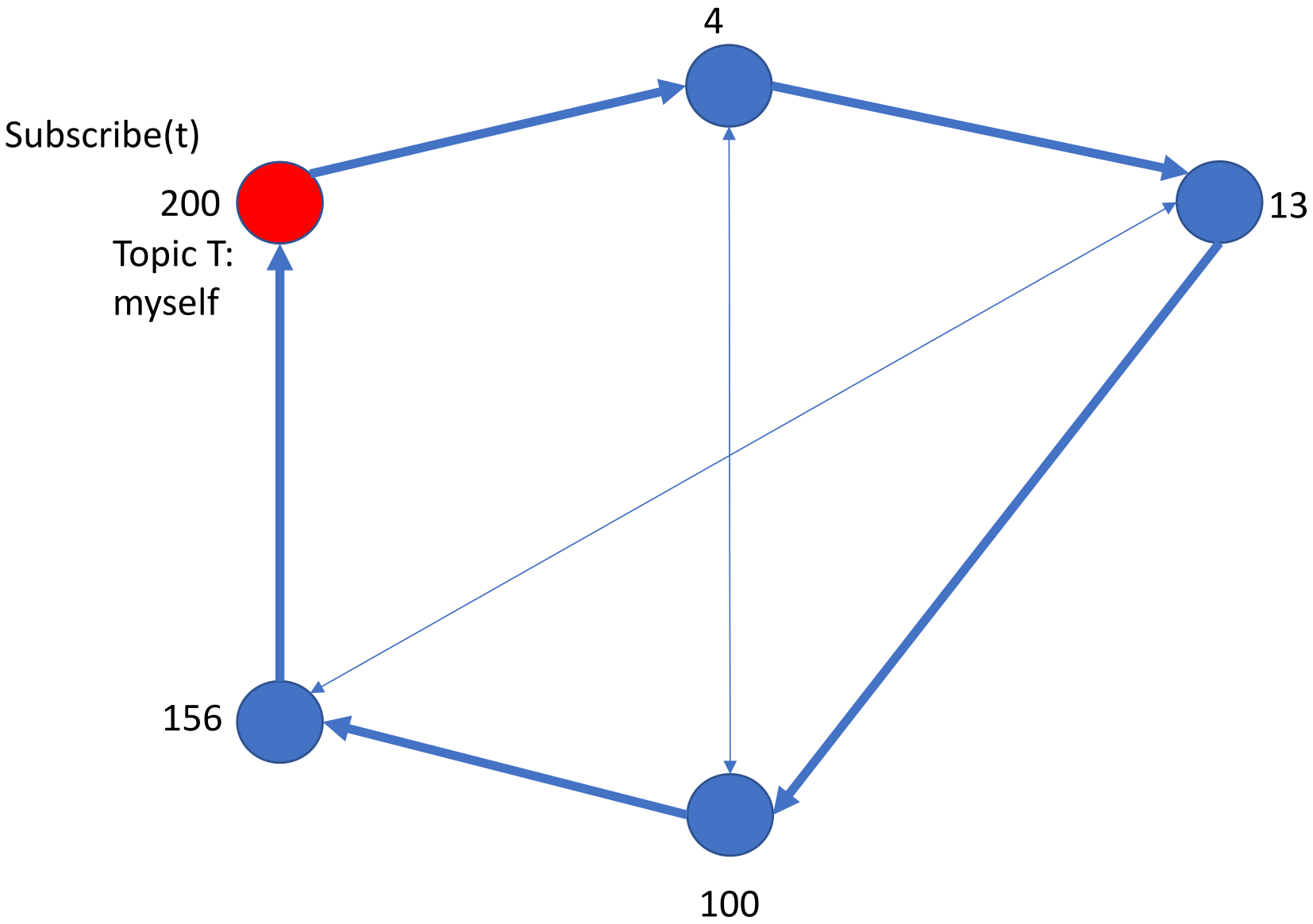
SCRIBE: Fault Tolerance



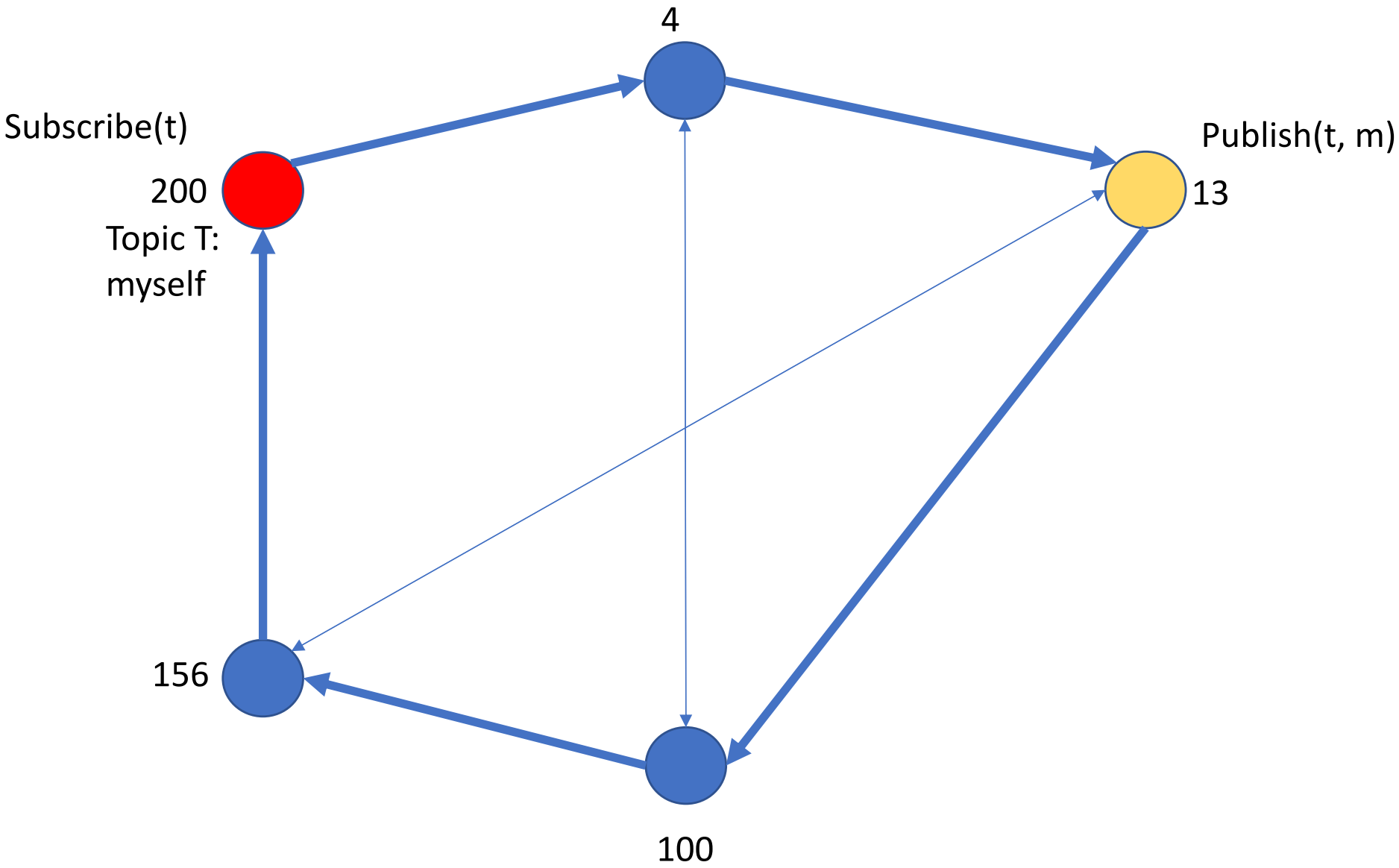
SCRIBE: Fault Tolerance



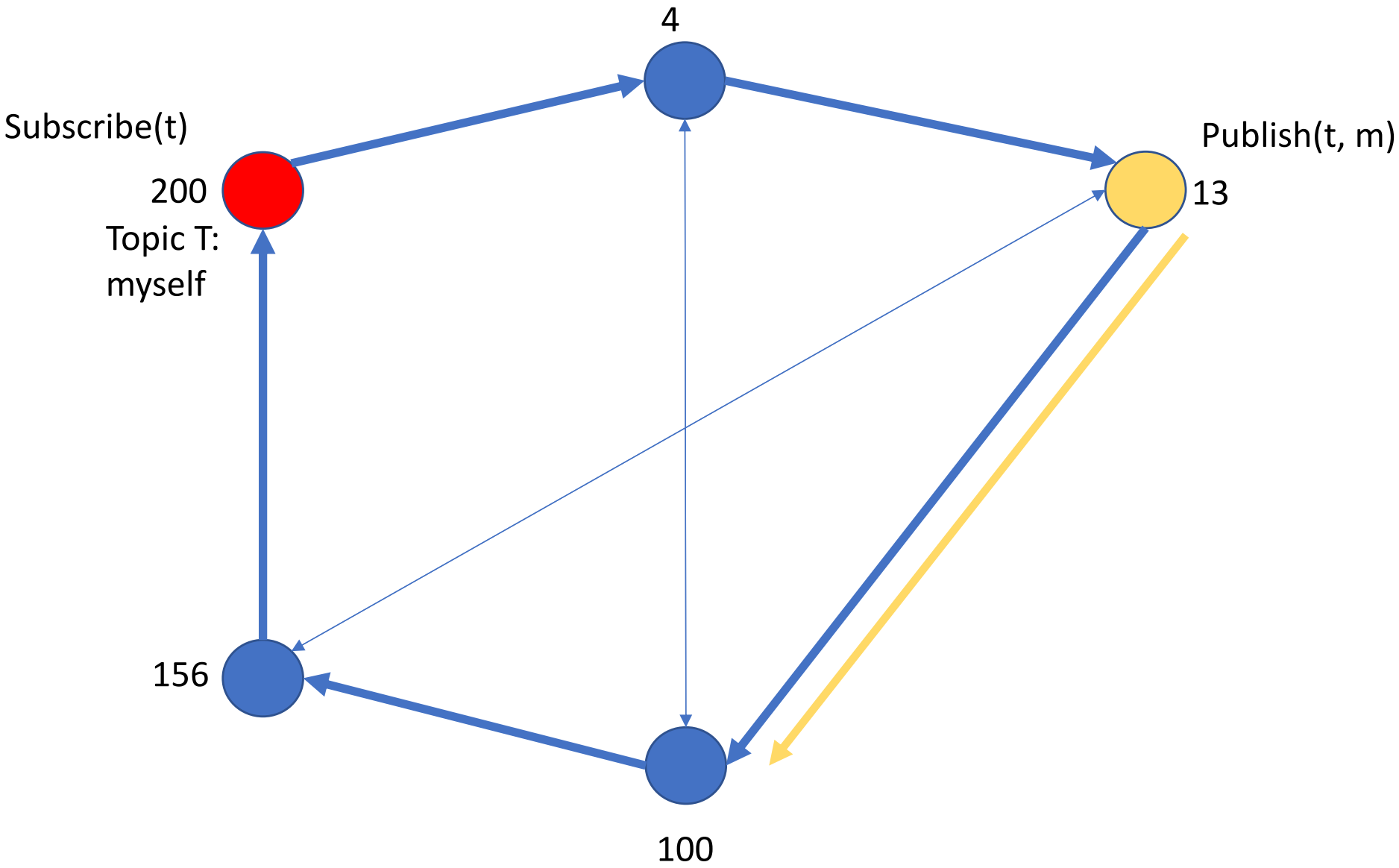
SCRIBE: Fault Tolerance



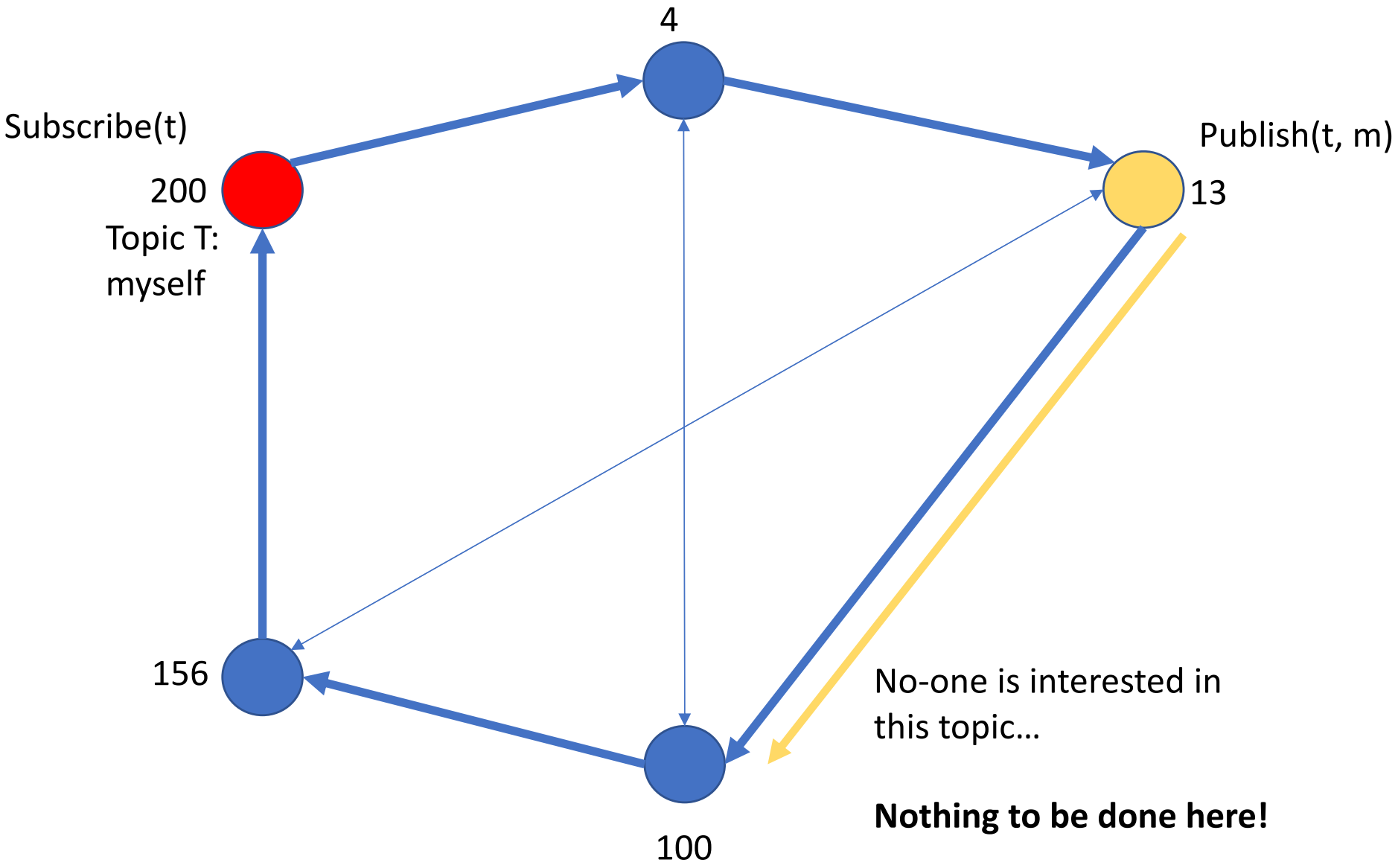
SCRIBE: Fault Tolerance



SCRIBE: Fault Tolerance



SCRIBE: Fault Tolerance



SCRIBE: Fault Tolerance

- The key trick is to use **soft state**
- Whenever a node subscribes to a topic, that subscription has a time validity.
- Before the expiration of that validity the node has to subscribe again.
- This will regenerate routing state for that topic in case of failures (still you can lose messages, but that would require having additional caching mechanisms).