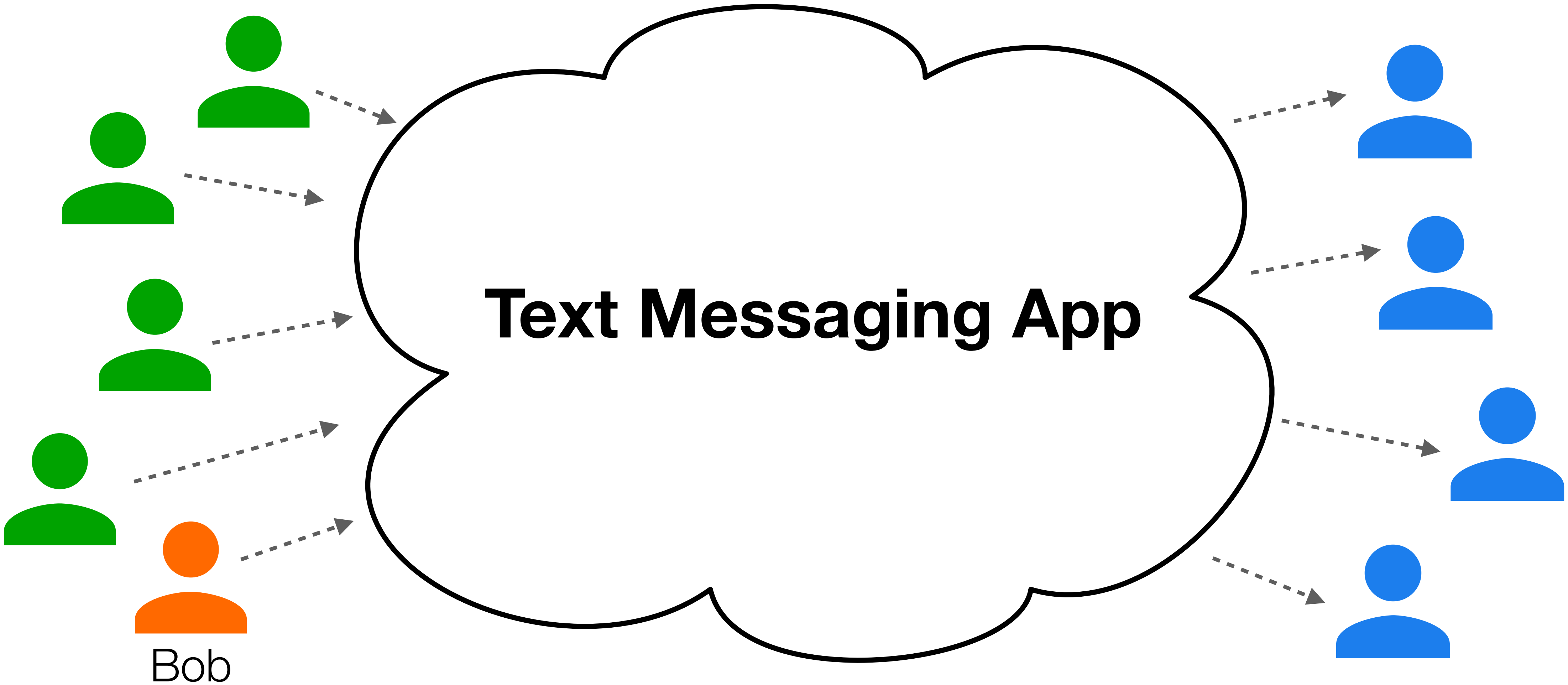
Karaoke

Distributed Private Messaging Immune to Passive Traffic Analysis

David Lazar, Yossi Gilad, Nickolai Zeldovich

Motivation: Report a crime without getting fired

Goal: Metadata-Private Text Messaging

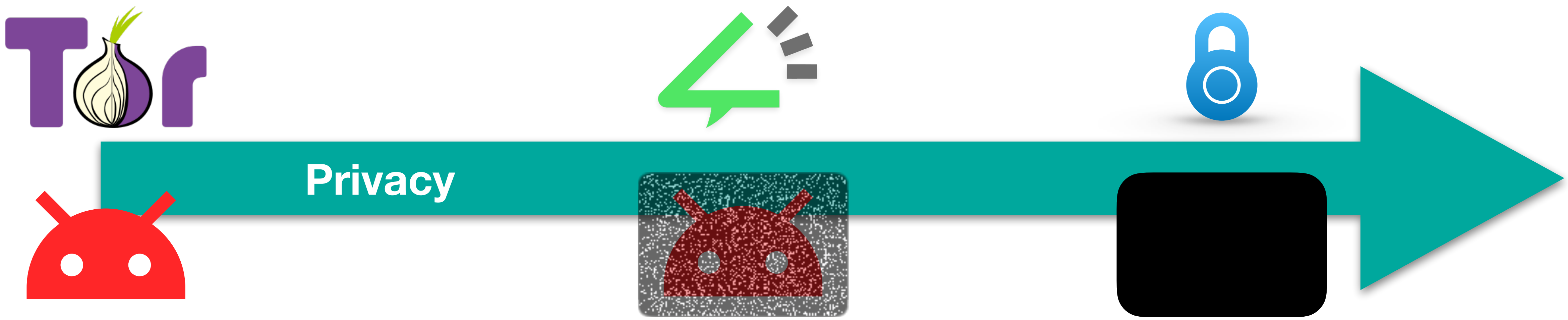


Threat Model: Global Adversary

Prior Approaches

Vuvuzela [SOSP 15]
Stadium [SOSP 17]

Pung [OSDI 16]
Dissent [OSDI 14]

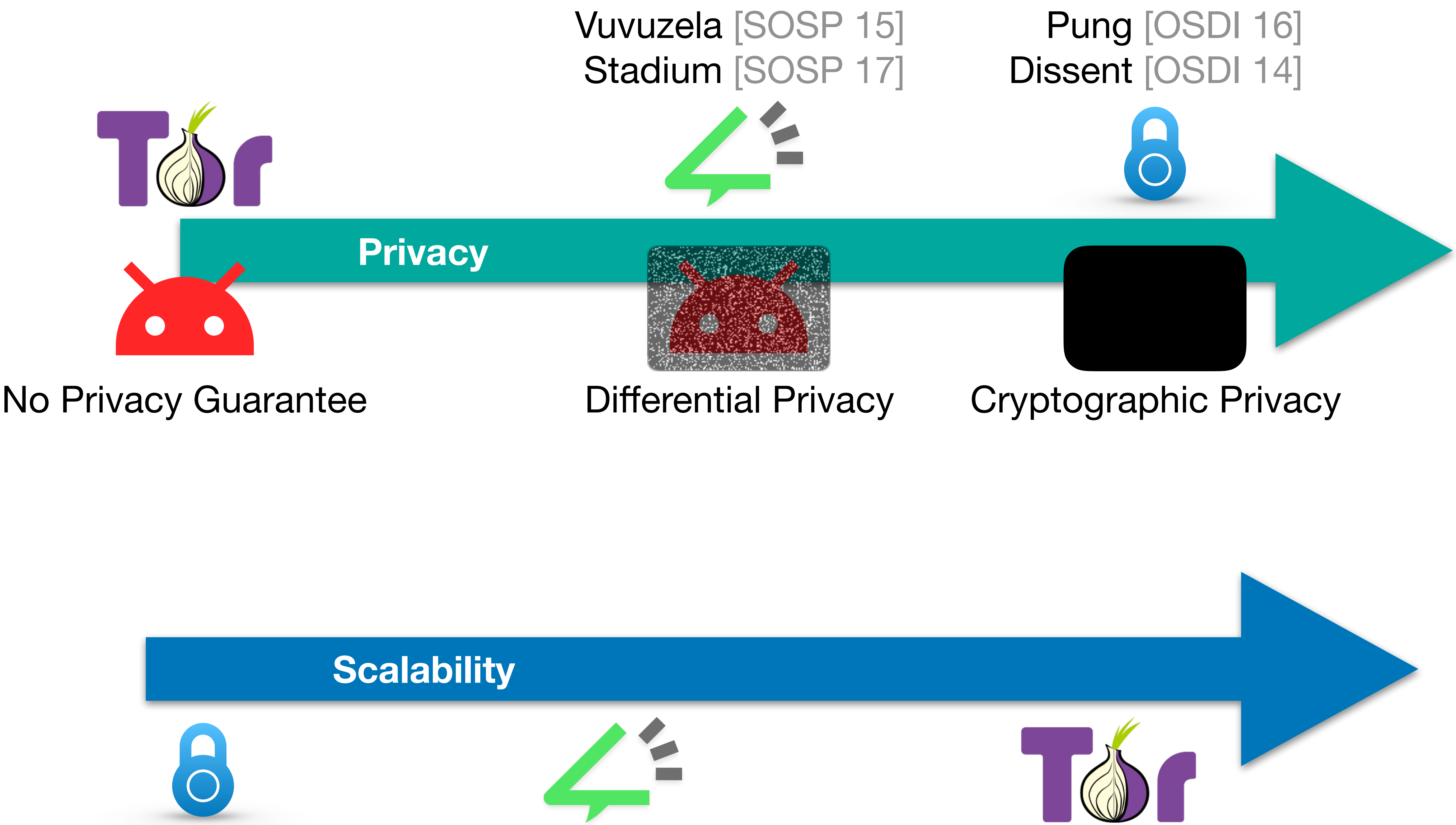


No Privacy Guarantee

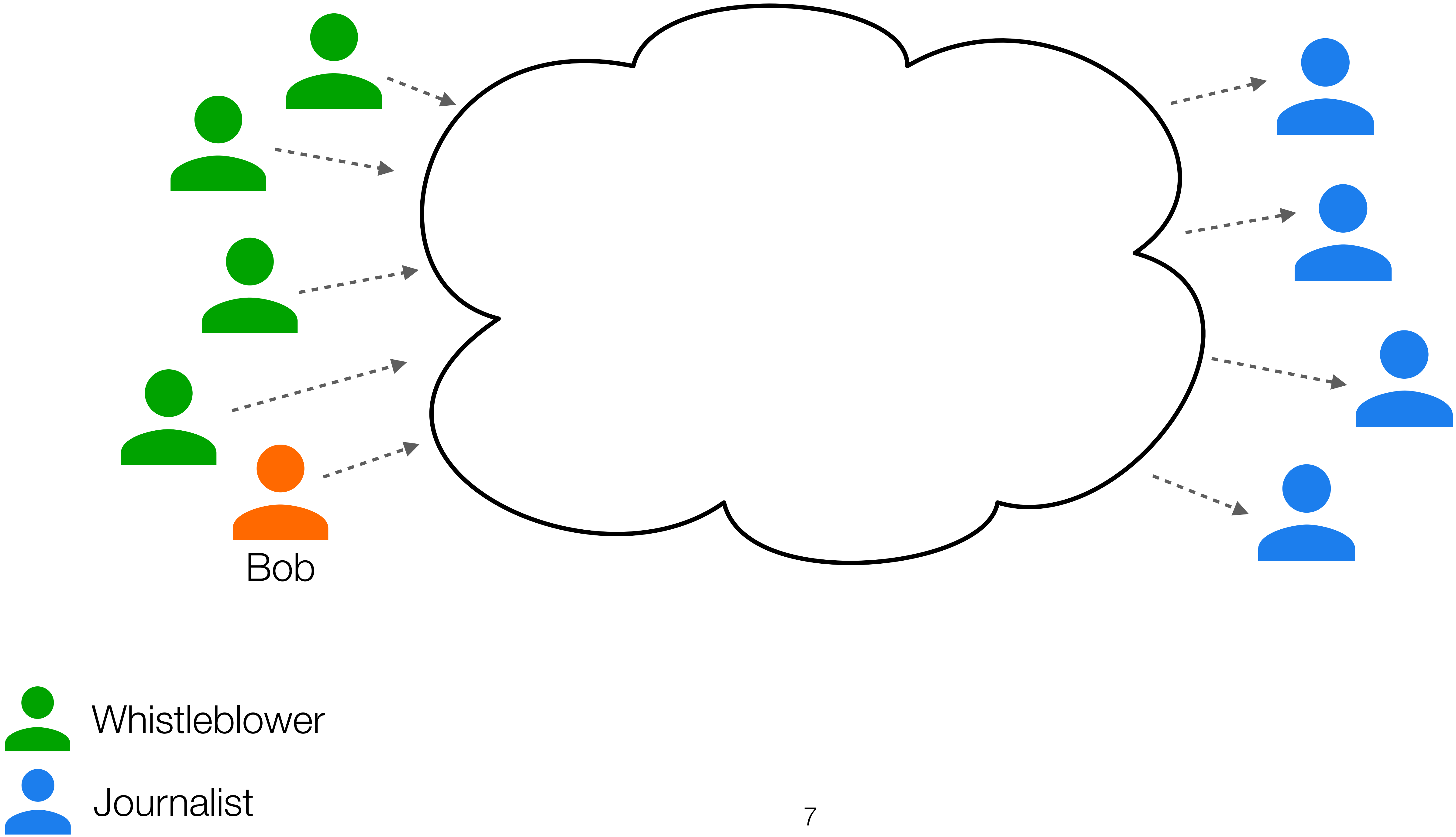
Differential Privacy

Cryptographic Privacy

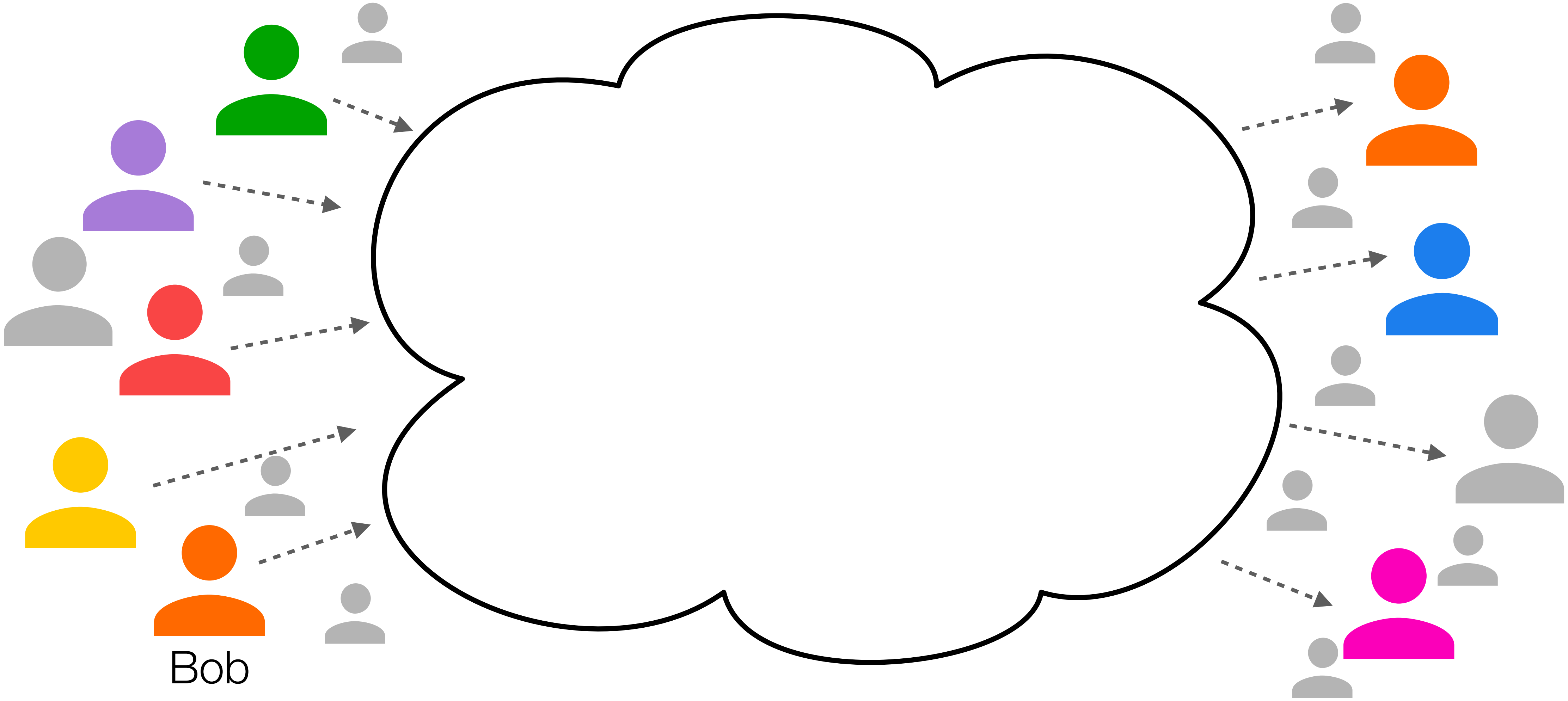
Prior Approaches



Scalability is critical for security



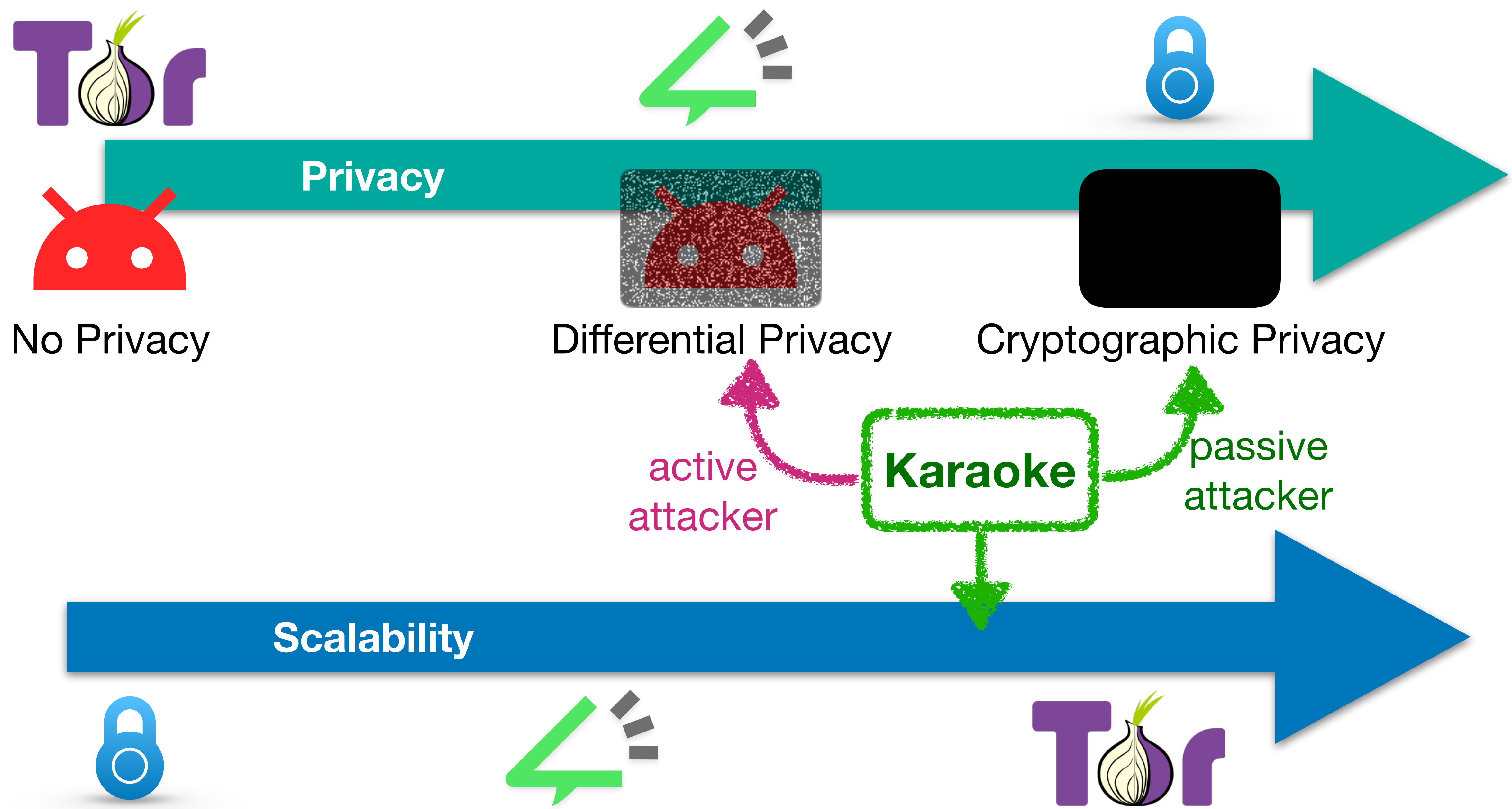
App must scale to everyone, so it isn’t suspicious when Bob joins



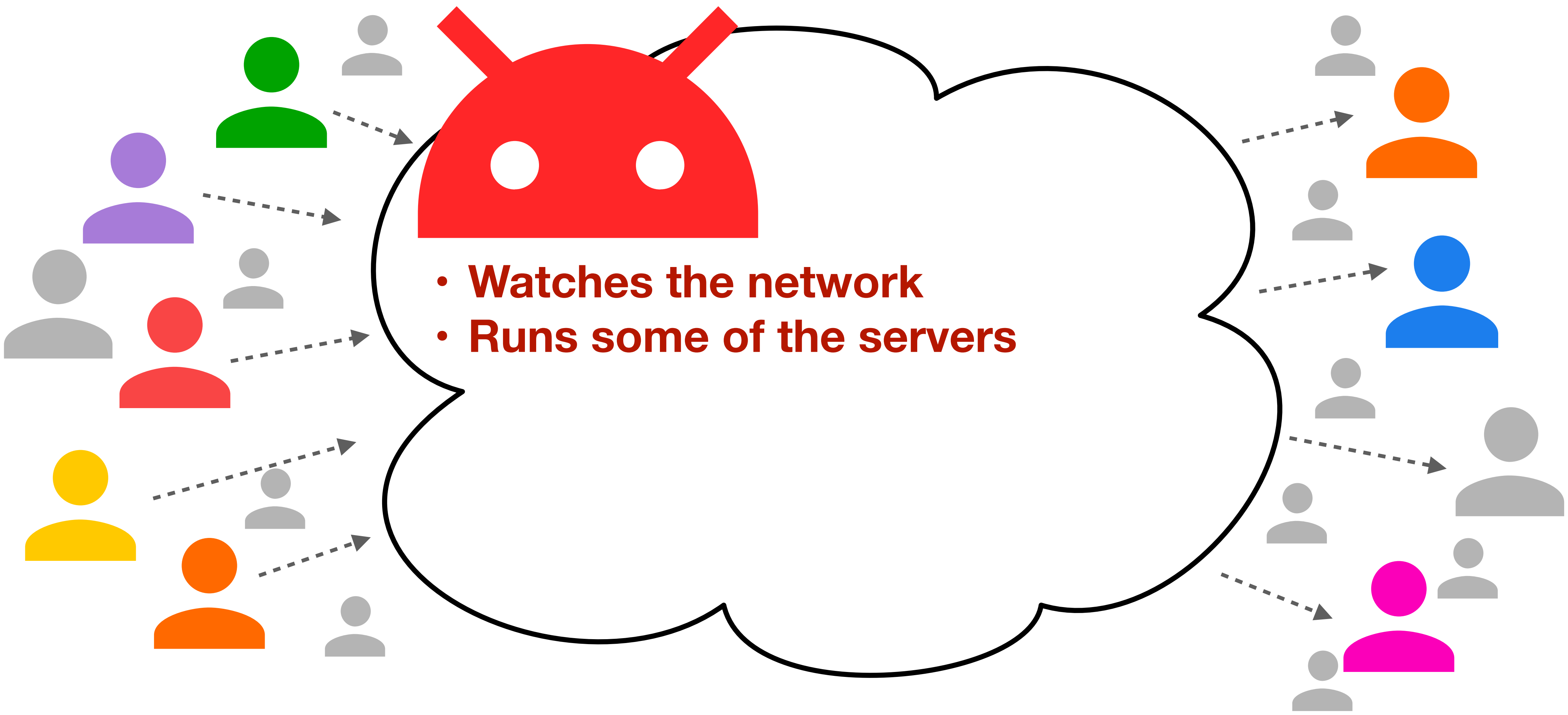
Contributions

- **Karaoke:** a distributed metadata-private messaging system that scales to more users
- Cryptographic privacy against **passive** attackers.
- Differential privacy against **active** attackers.
- 8s end-to-end latency with 4M users.
    - 5x to 11x faster than prior work.

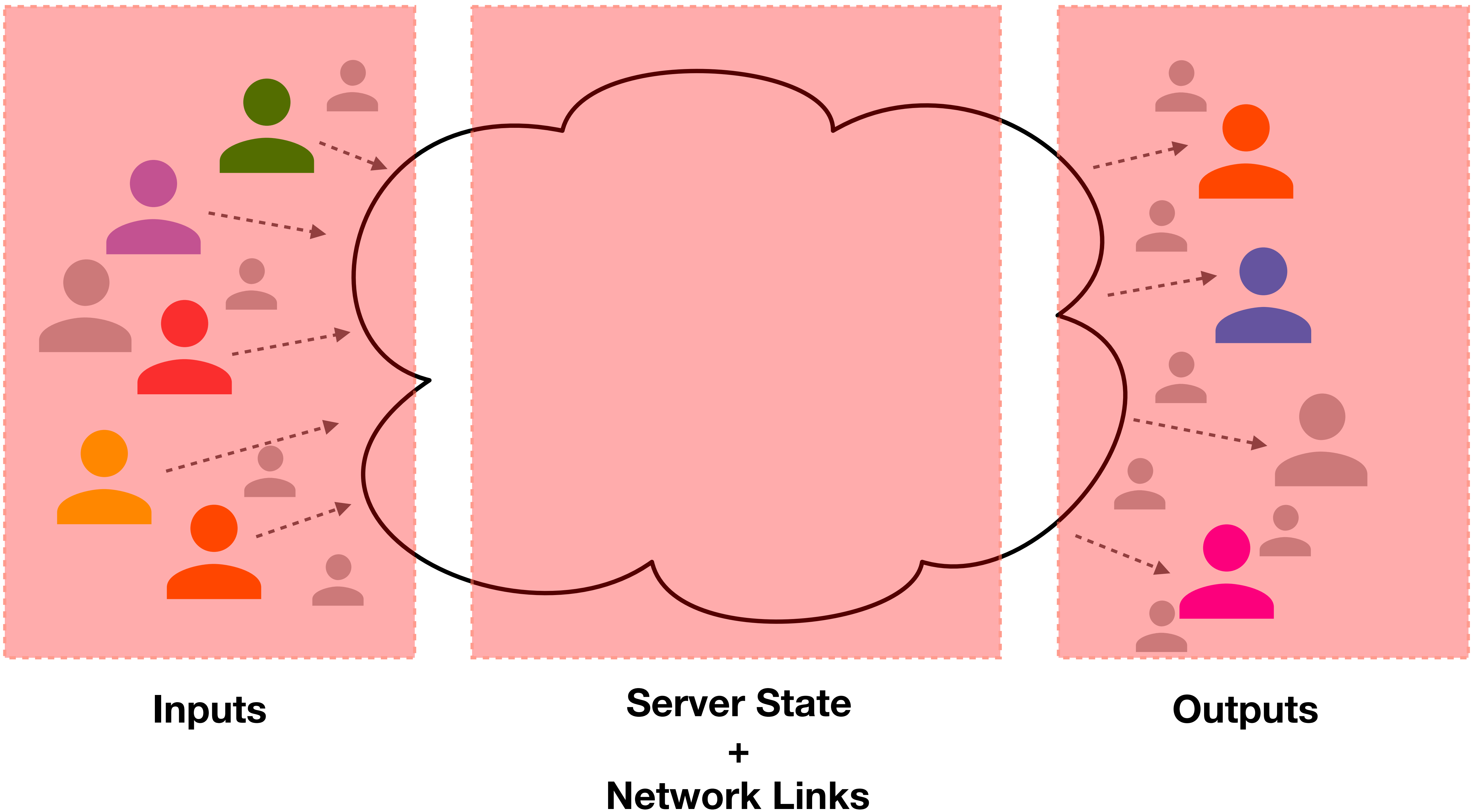
Insight: treat passive and active attackers separately



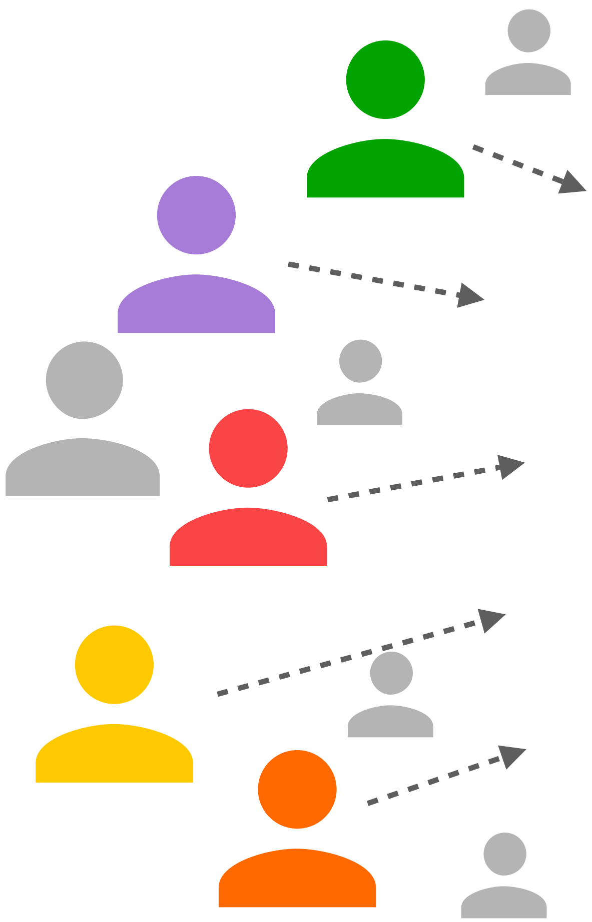
Global Passive Adversary



Observations by Adversary

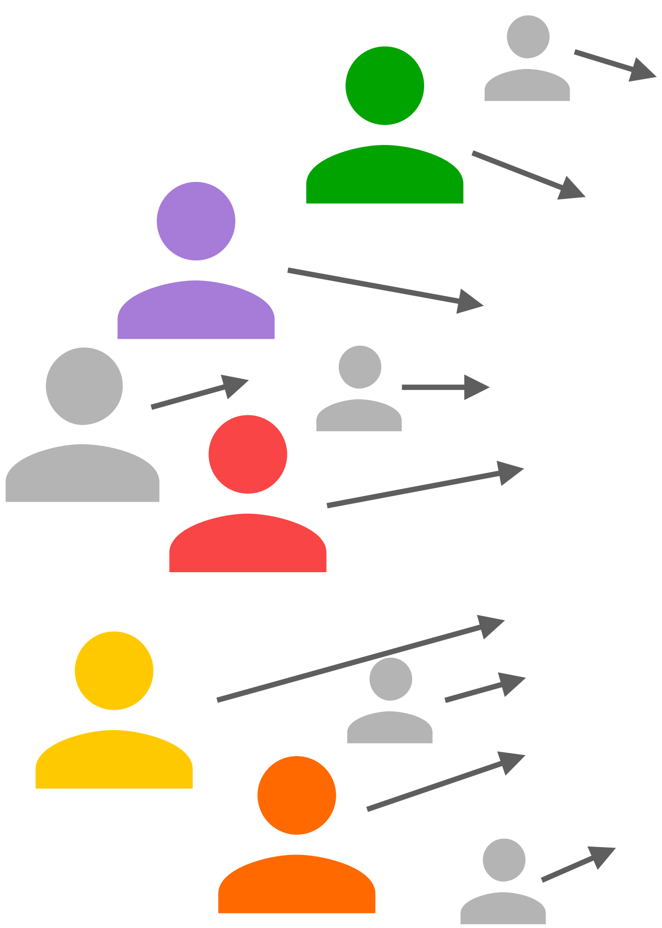


Observations by Adversary

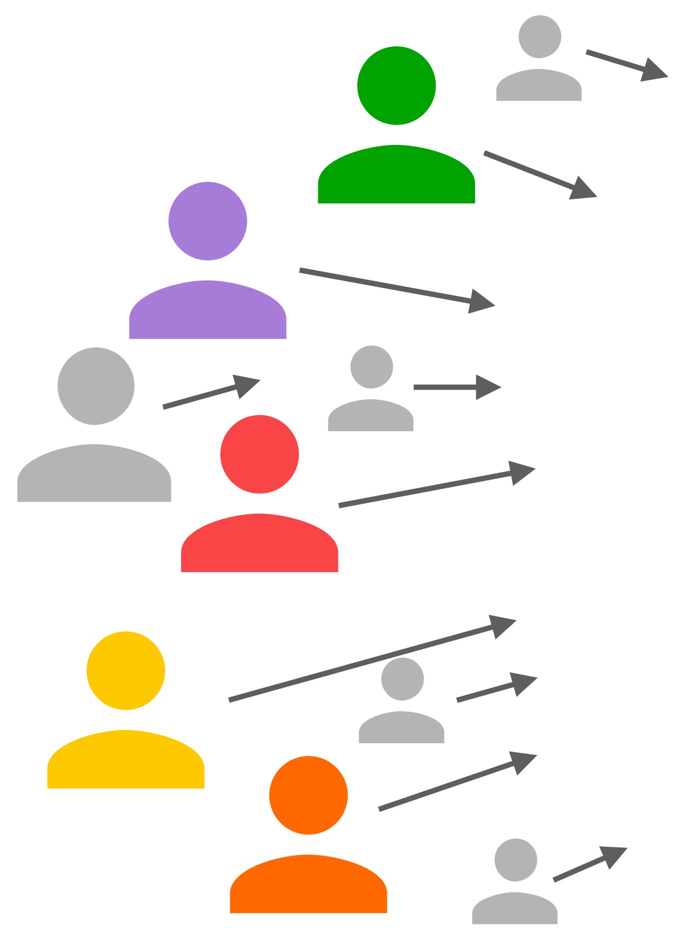


Inputs

Hiding inputs: constant cover traffic in rounds

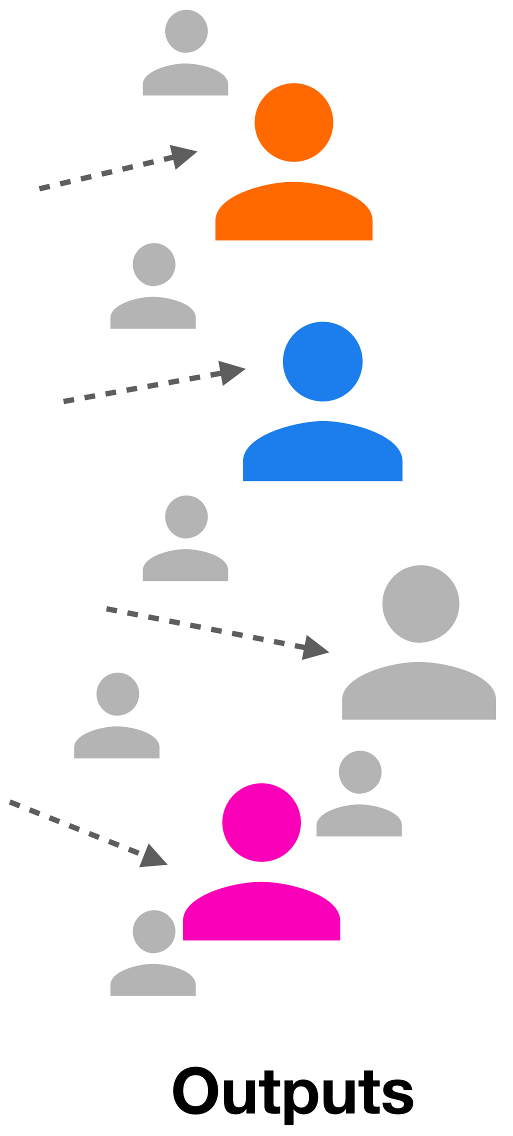


Round 1



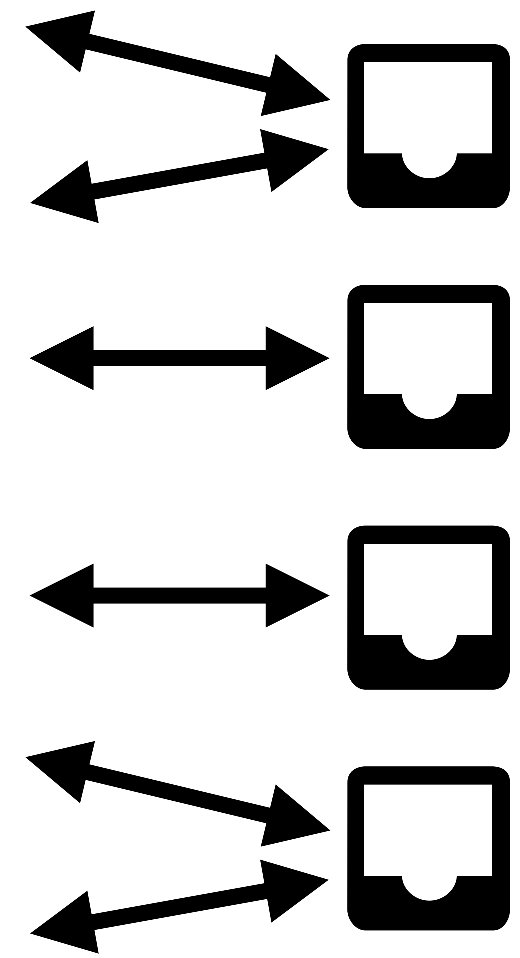
Round 2

Hiding outputs



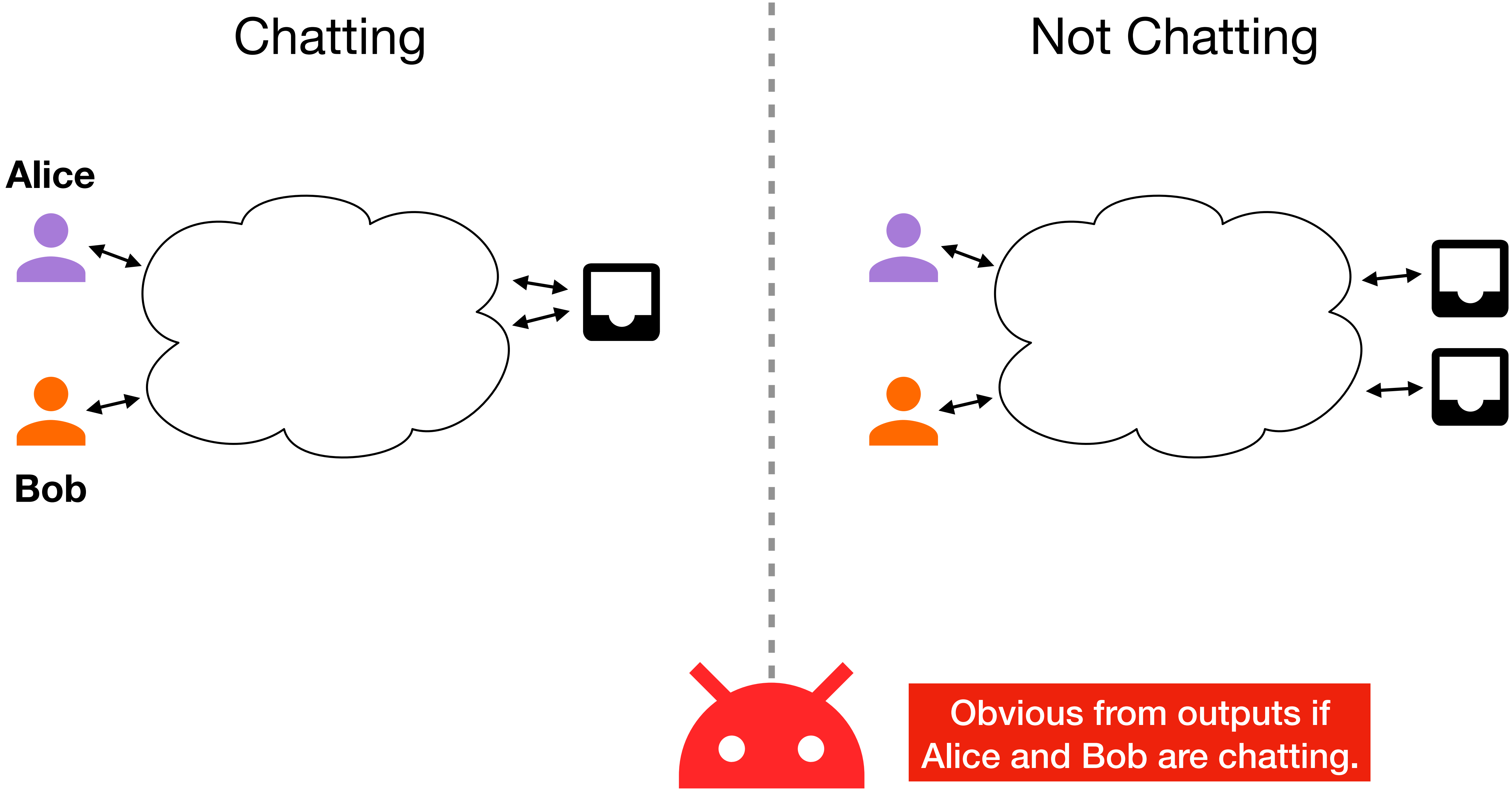
Hiding outputs with dead drops [Vuvuzela]

- **Dead drop:** designated location to exchange messages.
  - Named by pseudorandom ID, so reveals nothing about the users.
- When two users access the same dead drop, their messages are exchanged.
- Idle users result in dead drop with one access.

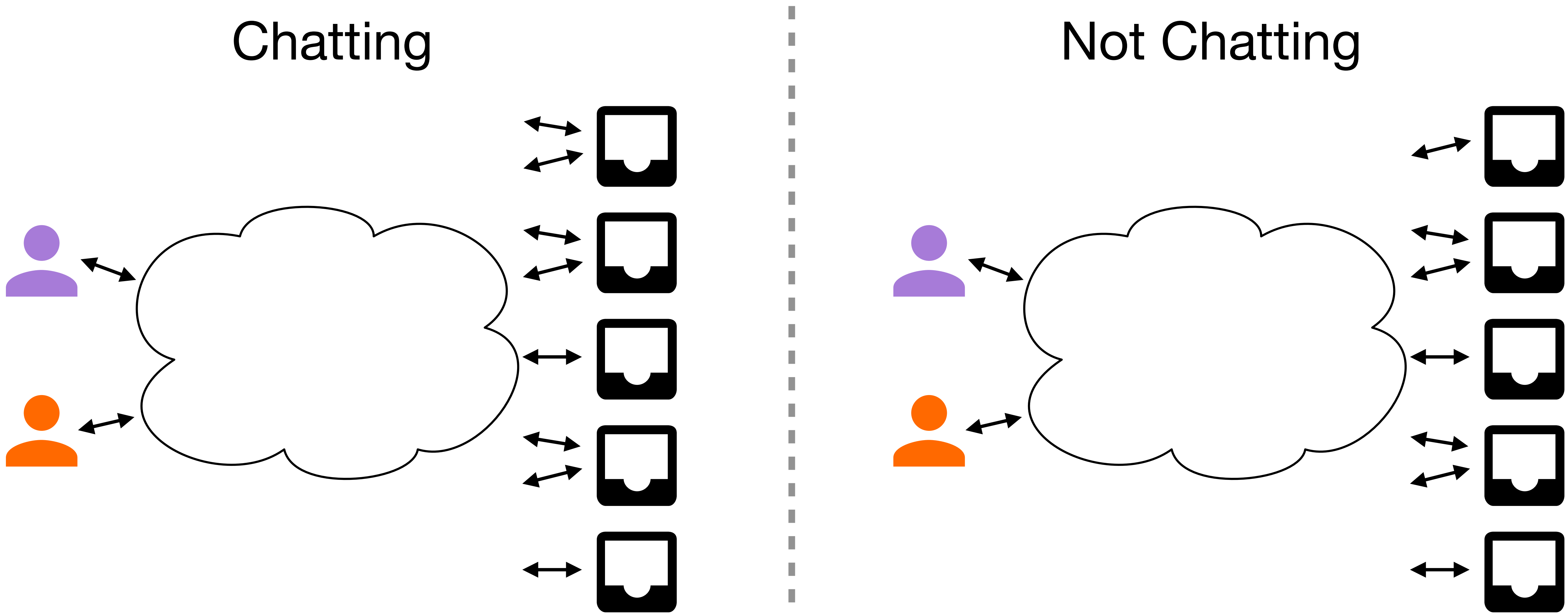


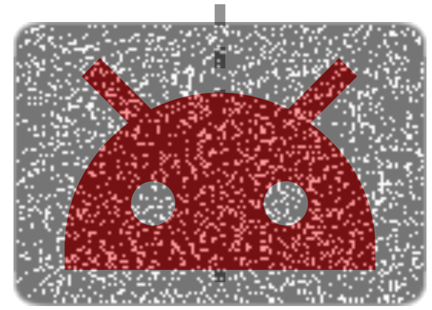
Dead drops

Dead drops alone are insufficient



Vuvuzela generates dummy accesses (noise)

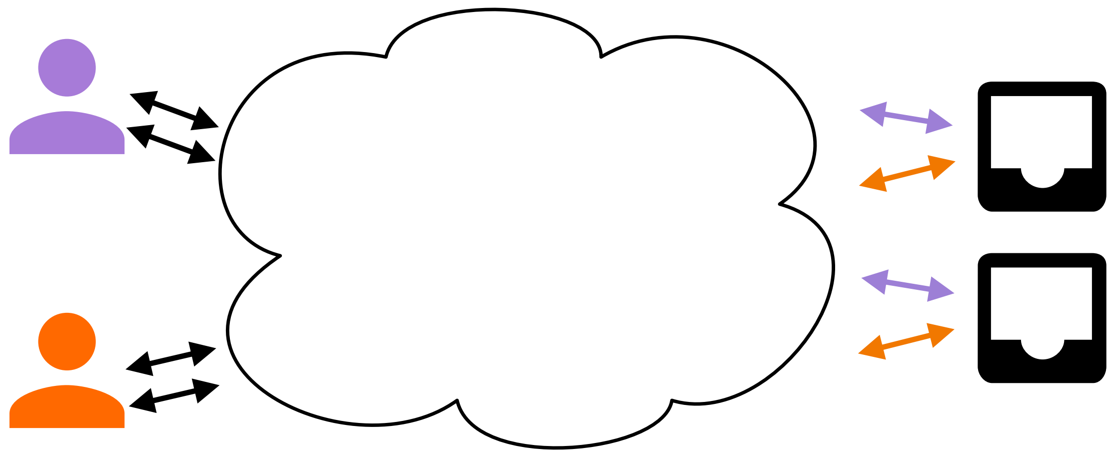




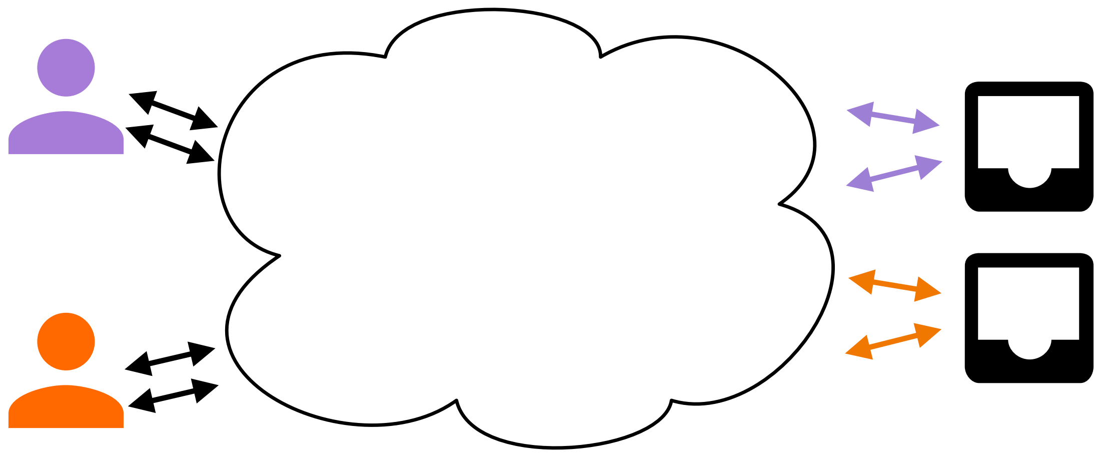
Differential privacy: no single round reveals much, but many rounds of observation might reveal a pattern.

Karaoke dead drops are always doubles

Chatting

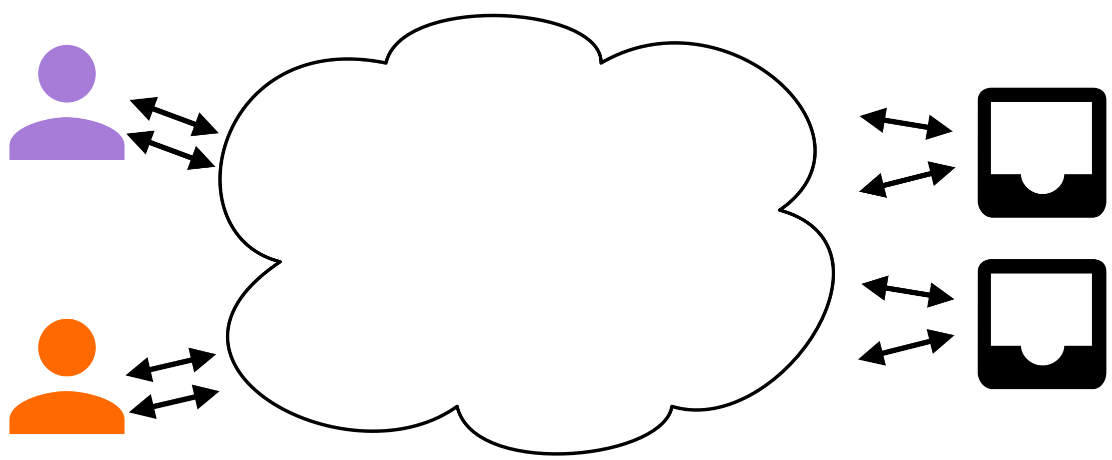


Not Chatting

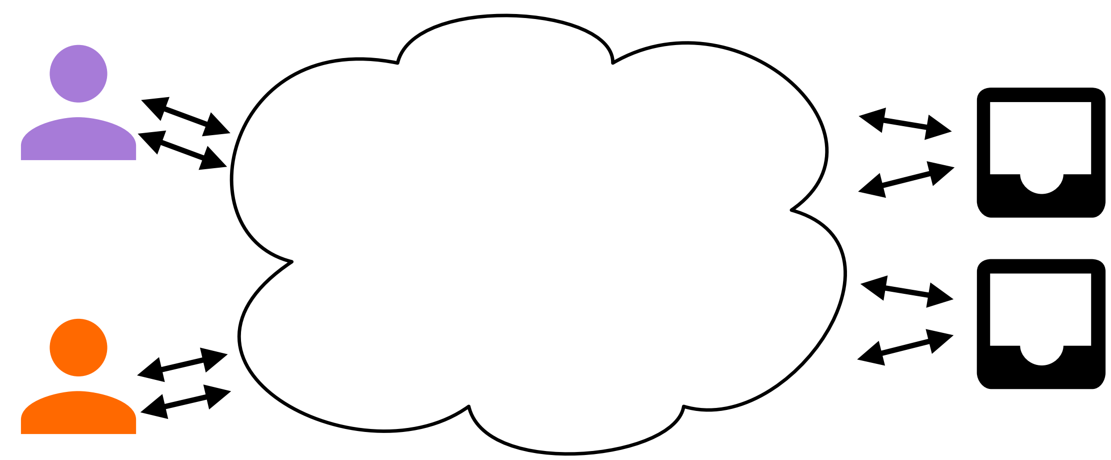


Message doubling provides cryptographic privacy

Chatting

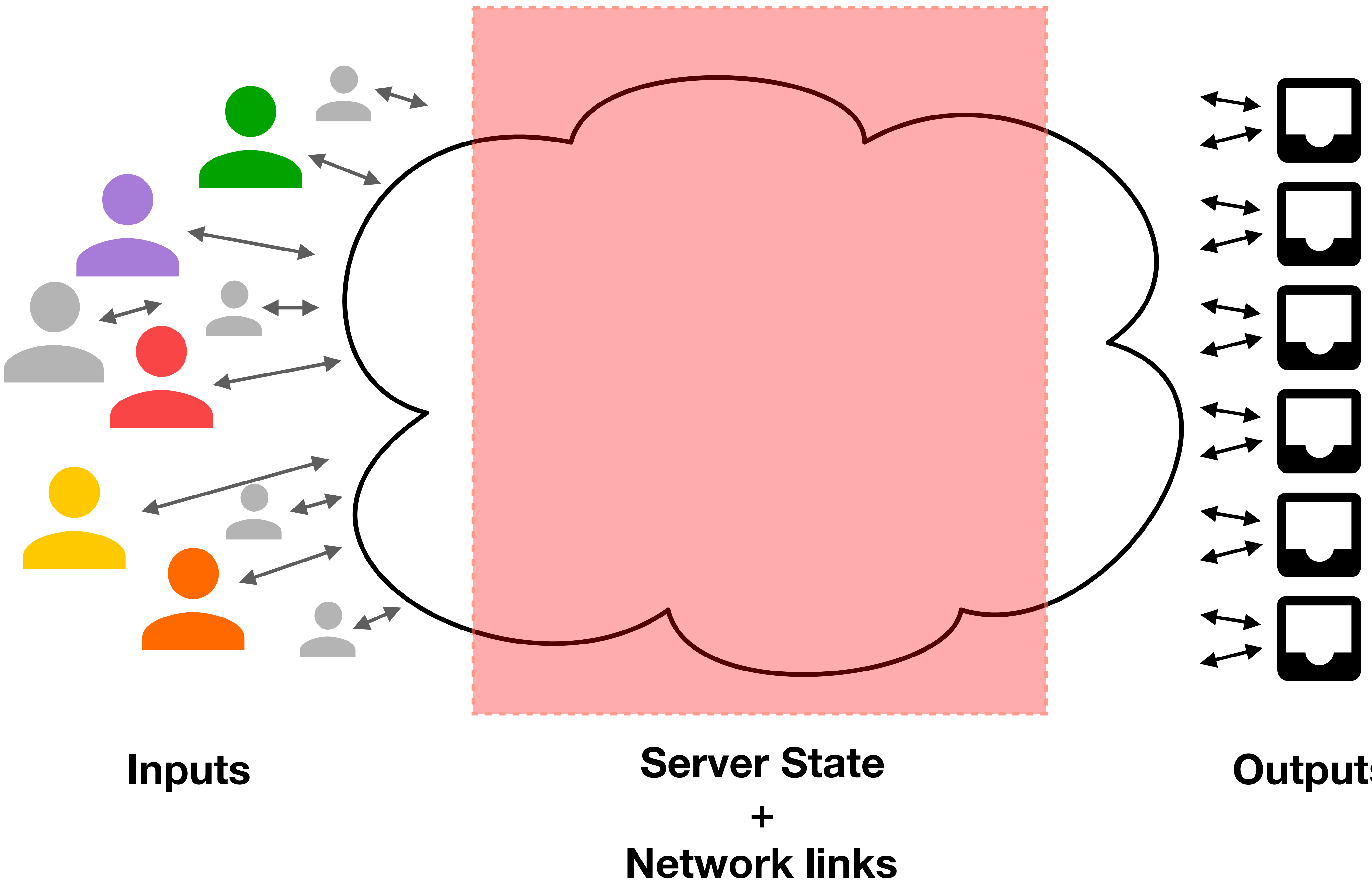


Not Chatting

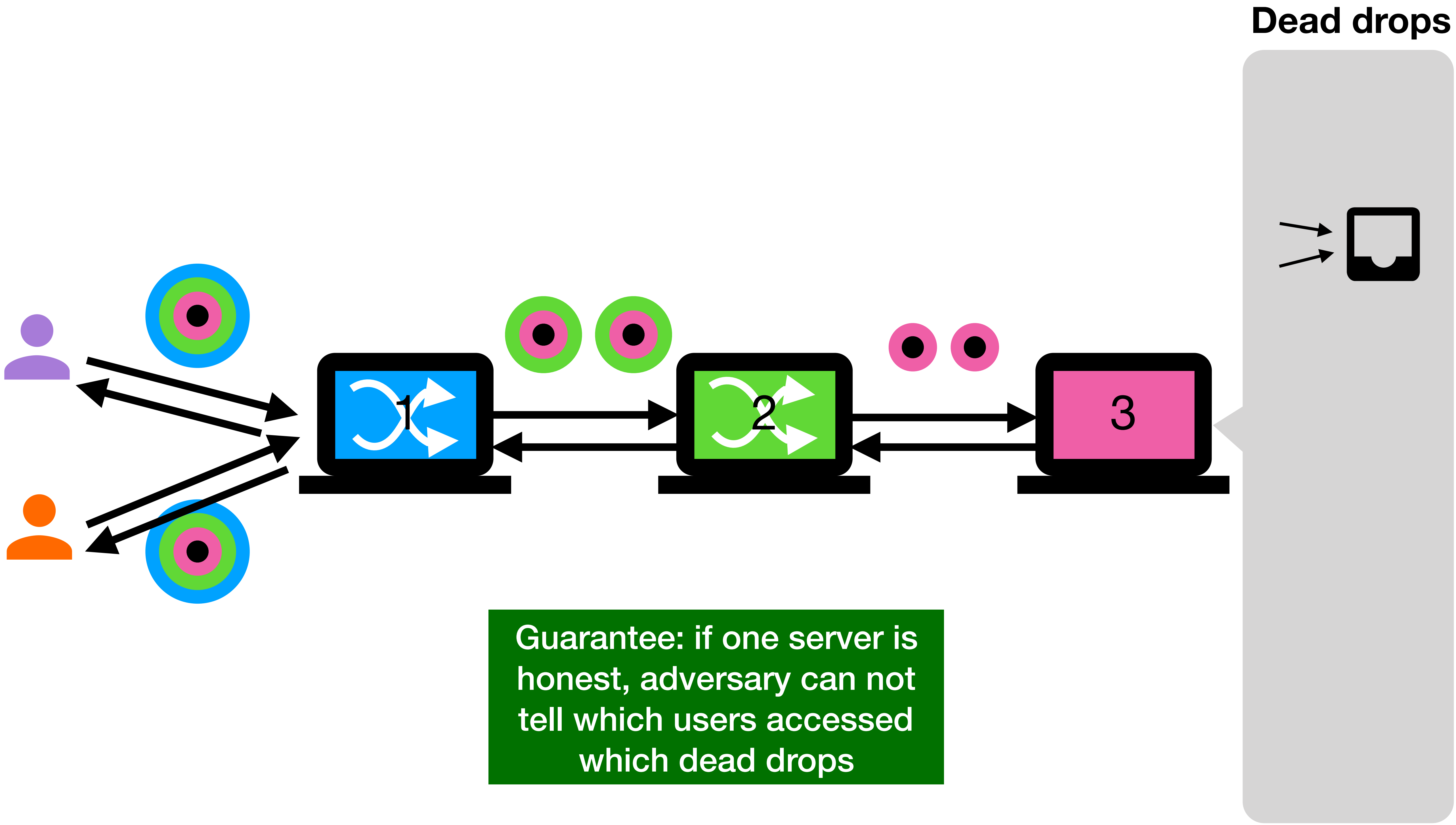


Cryptographic privacy: adversary can't distinguish whether Alice and Bob are chatting

Observations by Adversary

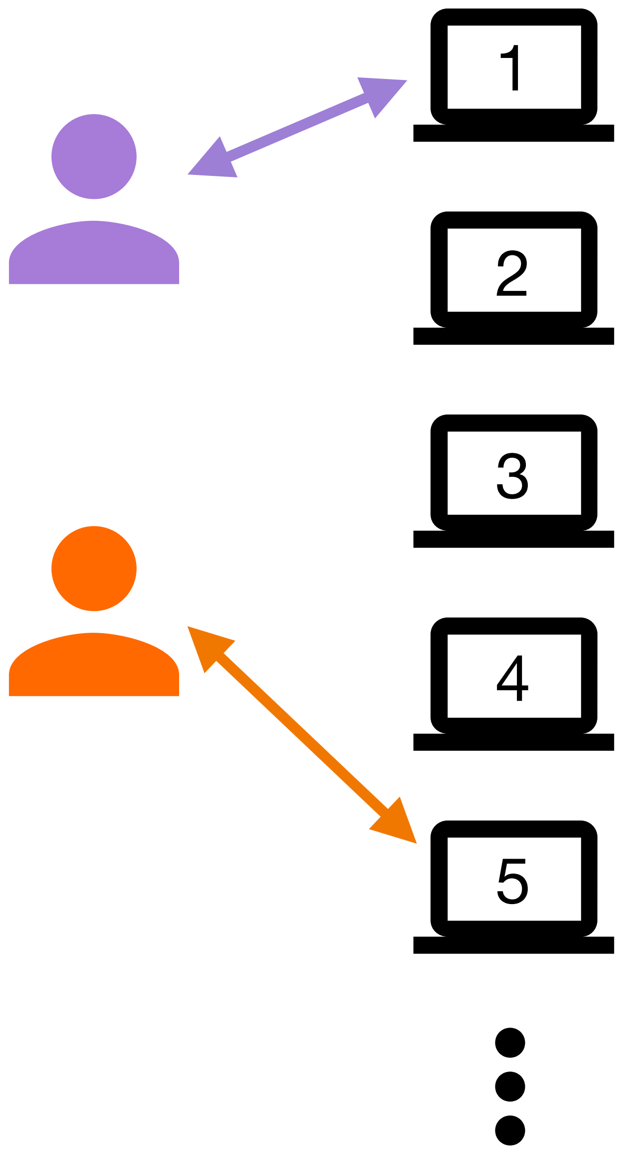


Mixnet Review

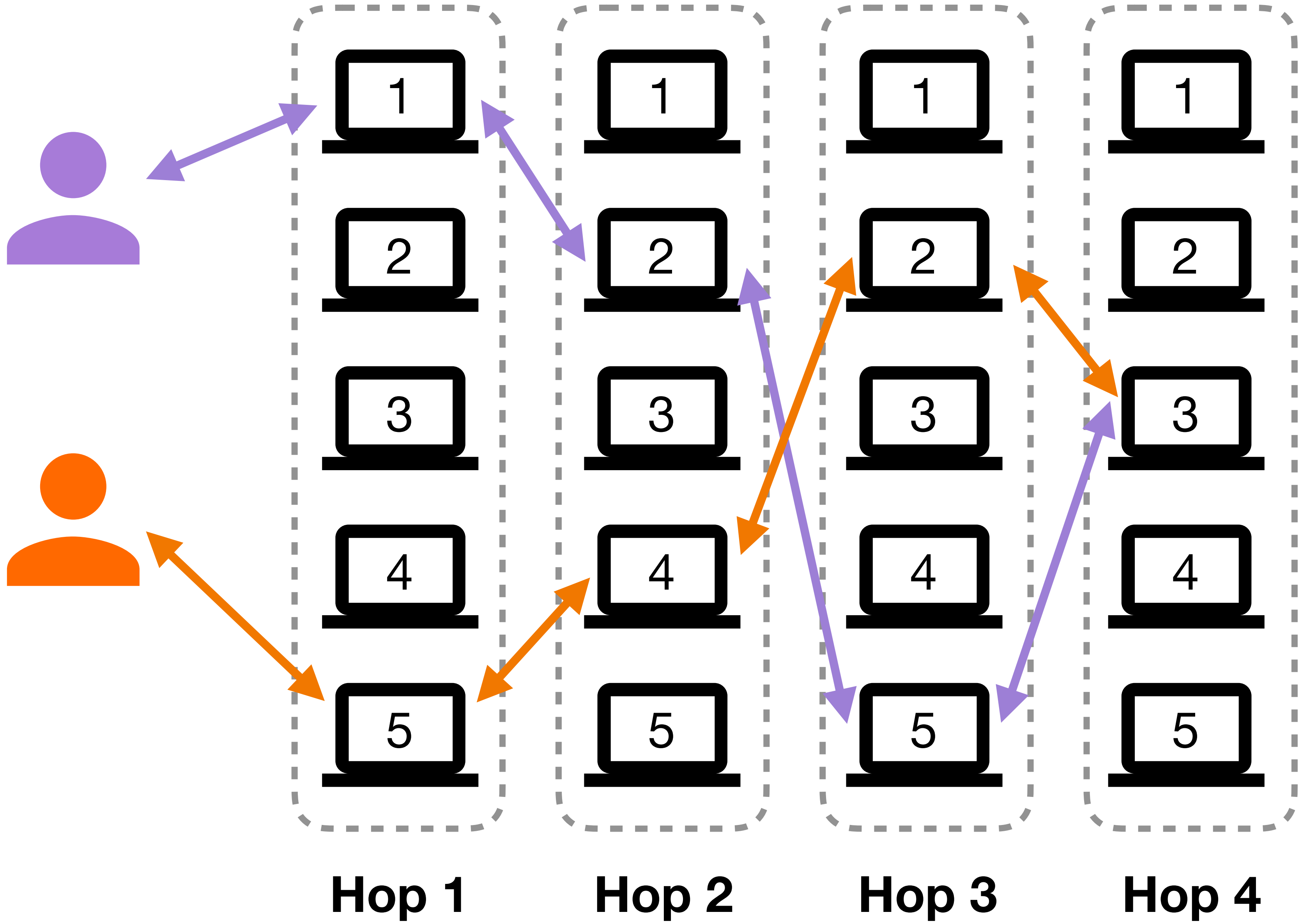


Guarantee: if one server is honest, adversary can not tell which users accessed which dead drops

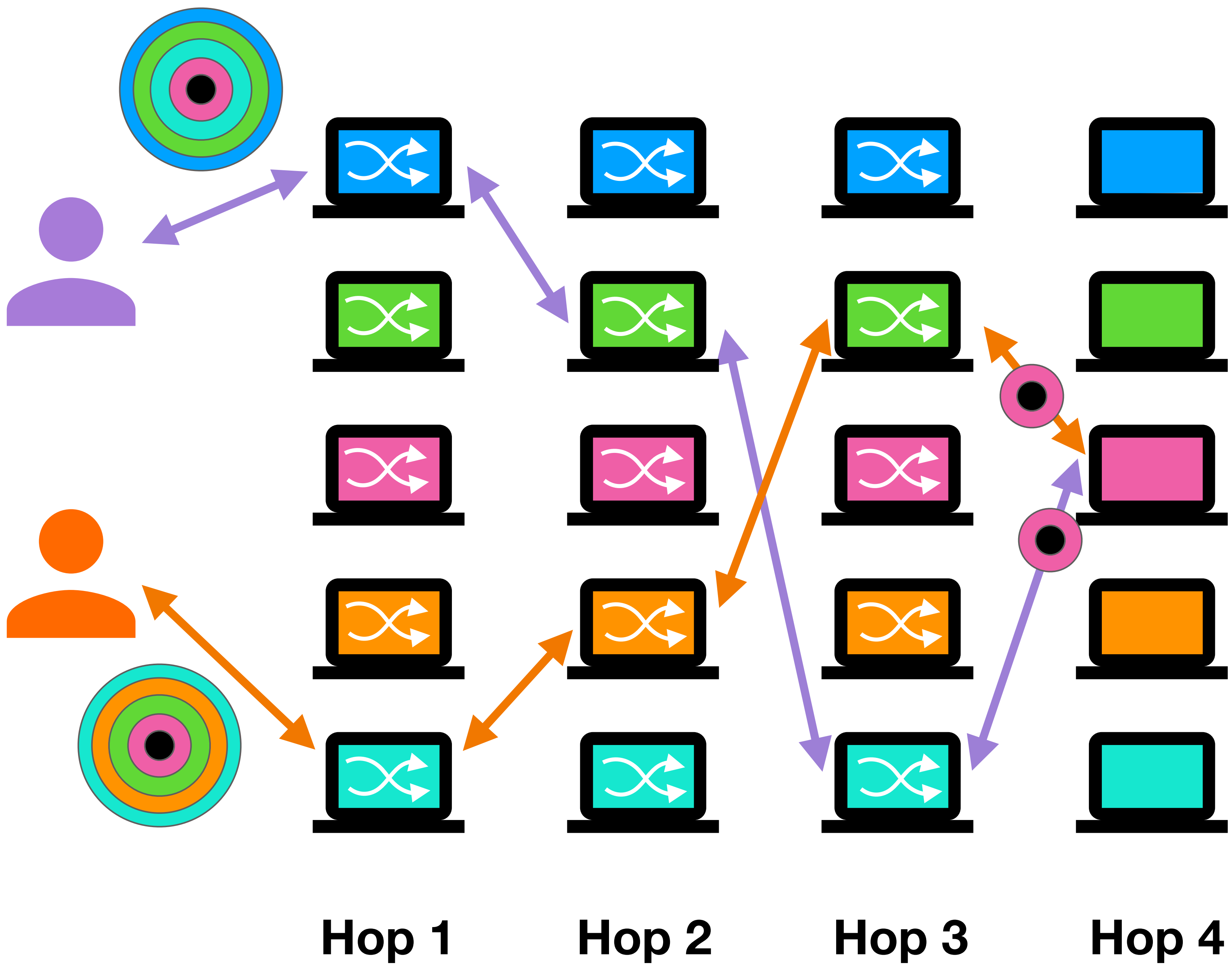
Distributed Mixnet: each server processes subset of messages



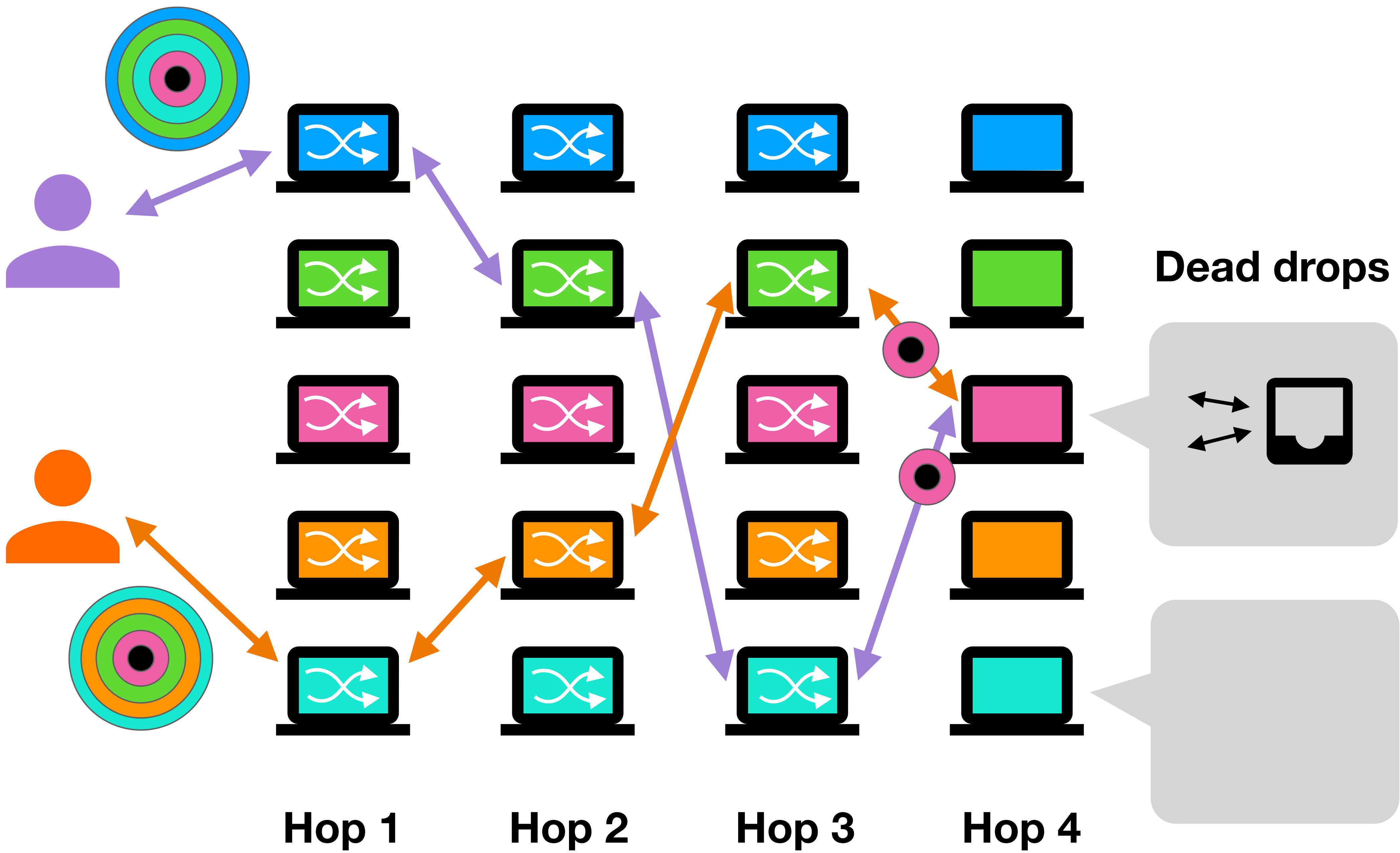
Users pick random paths through the network



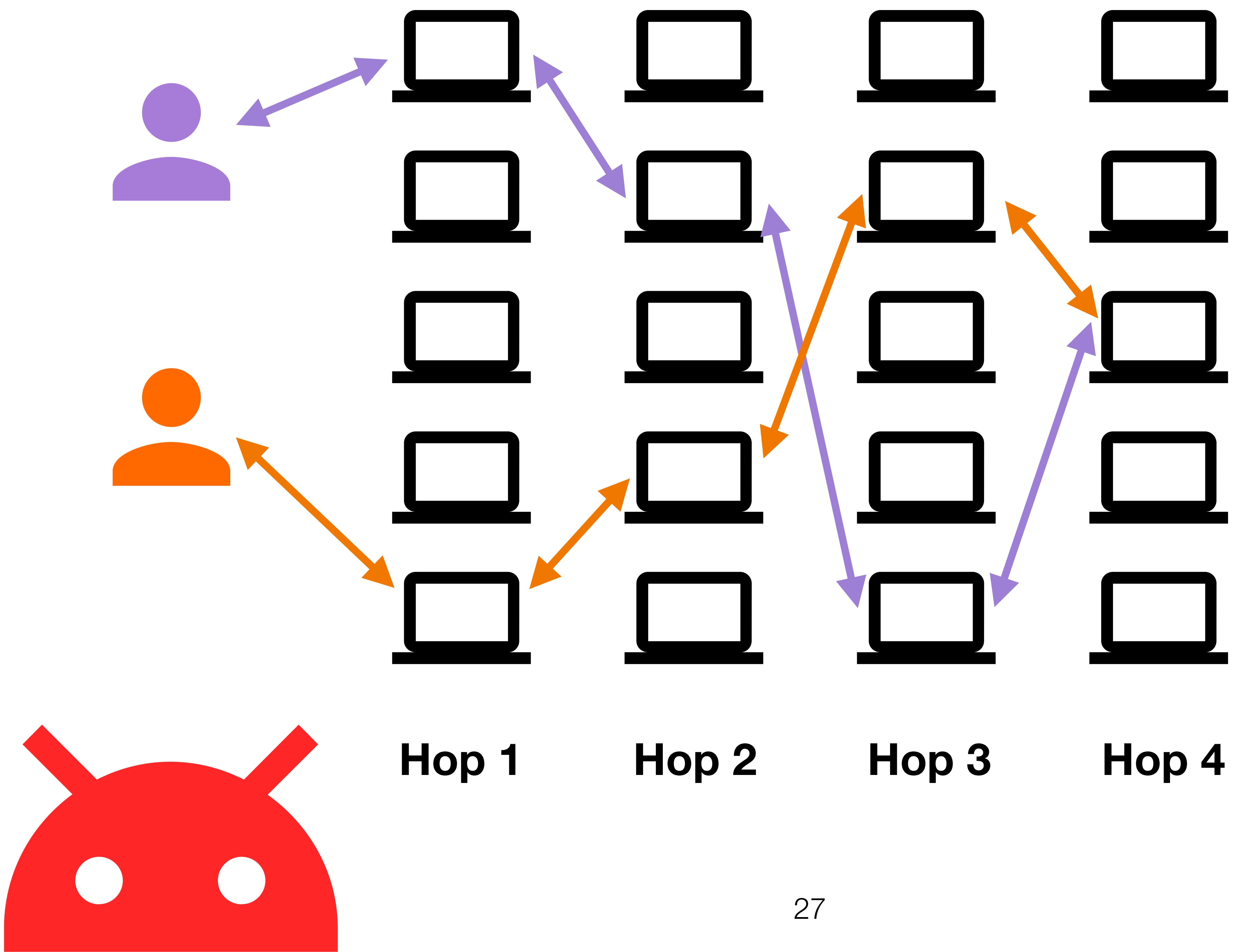
Servers decrypt and shuffle incoming messages at each hop



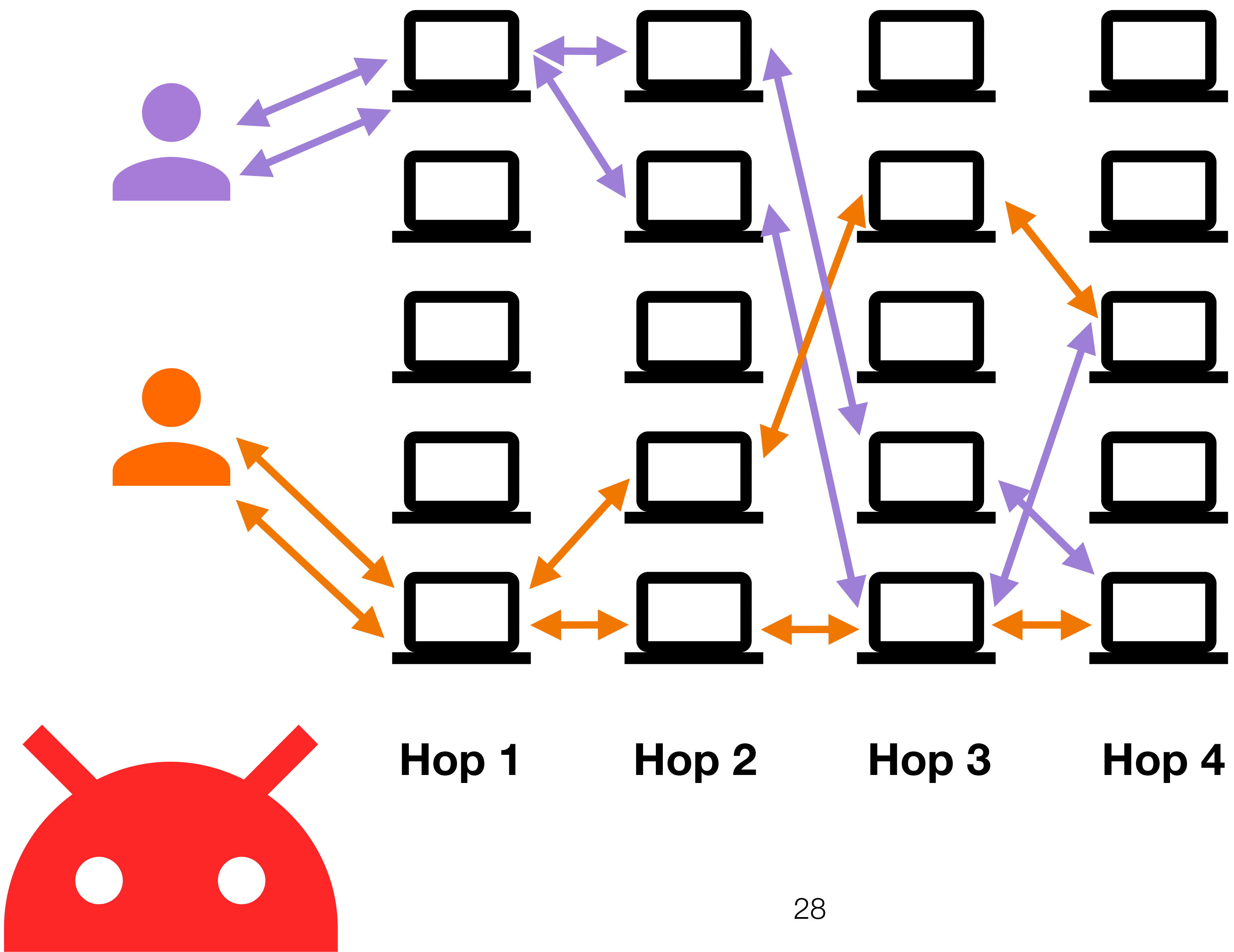
Last hop does the dead drop exchanges



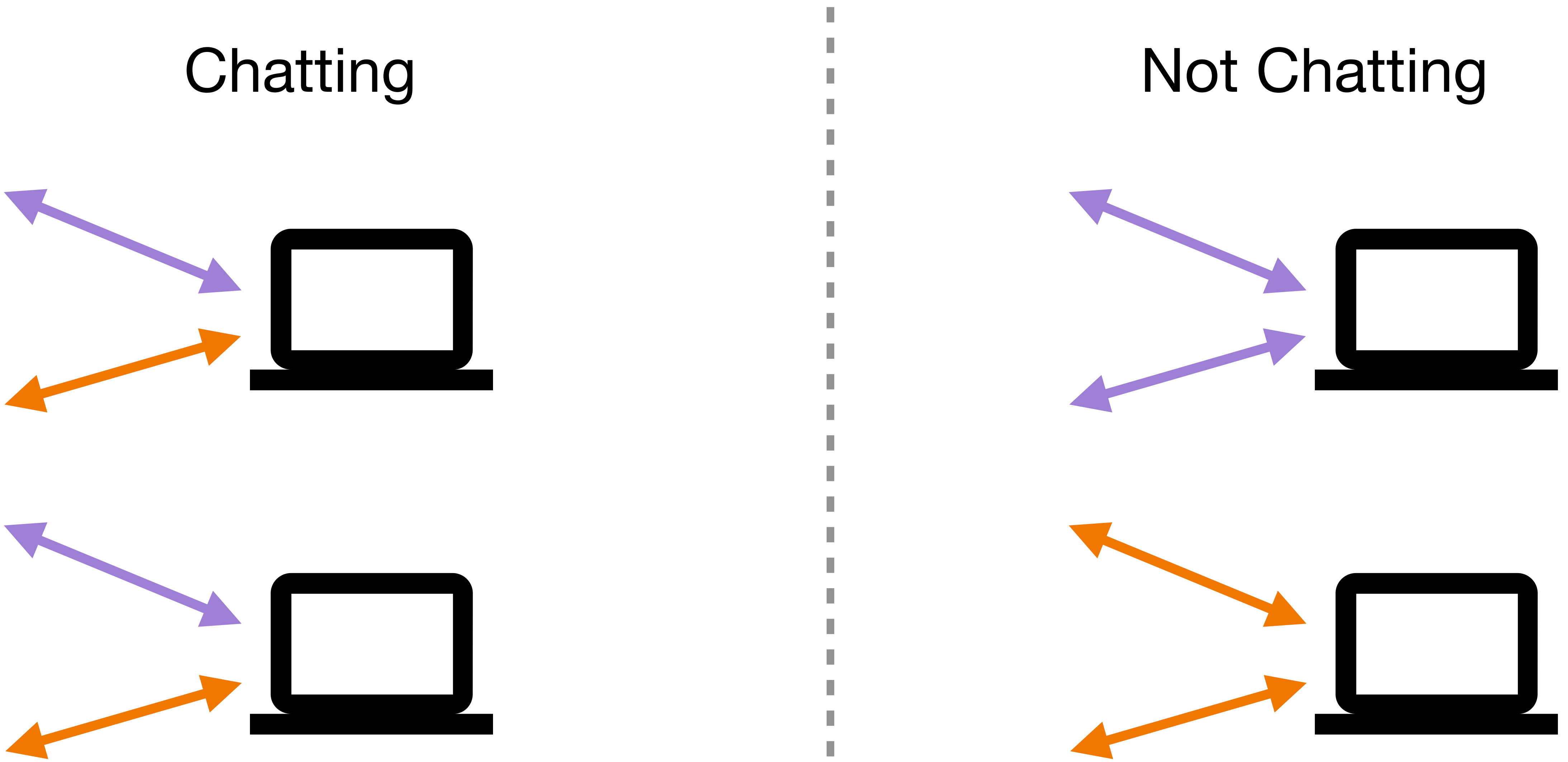
Challenge: network links between hops show Alice is talking to Bob!



Karaoke's message doubling gives us some hope!

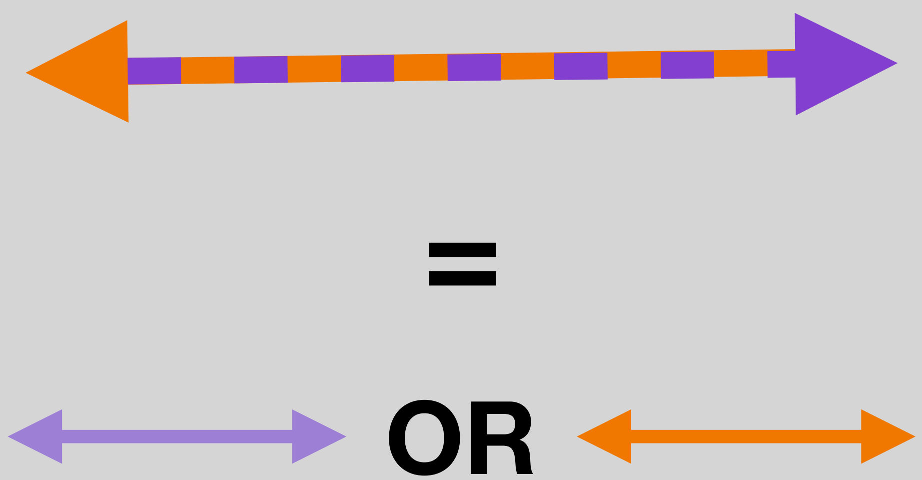


Possible cases for the last hop

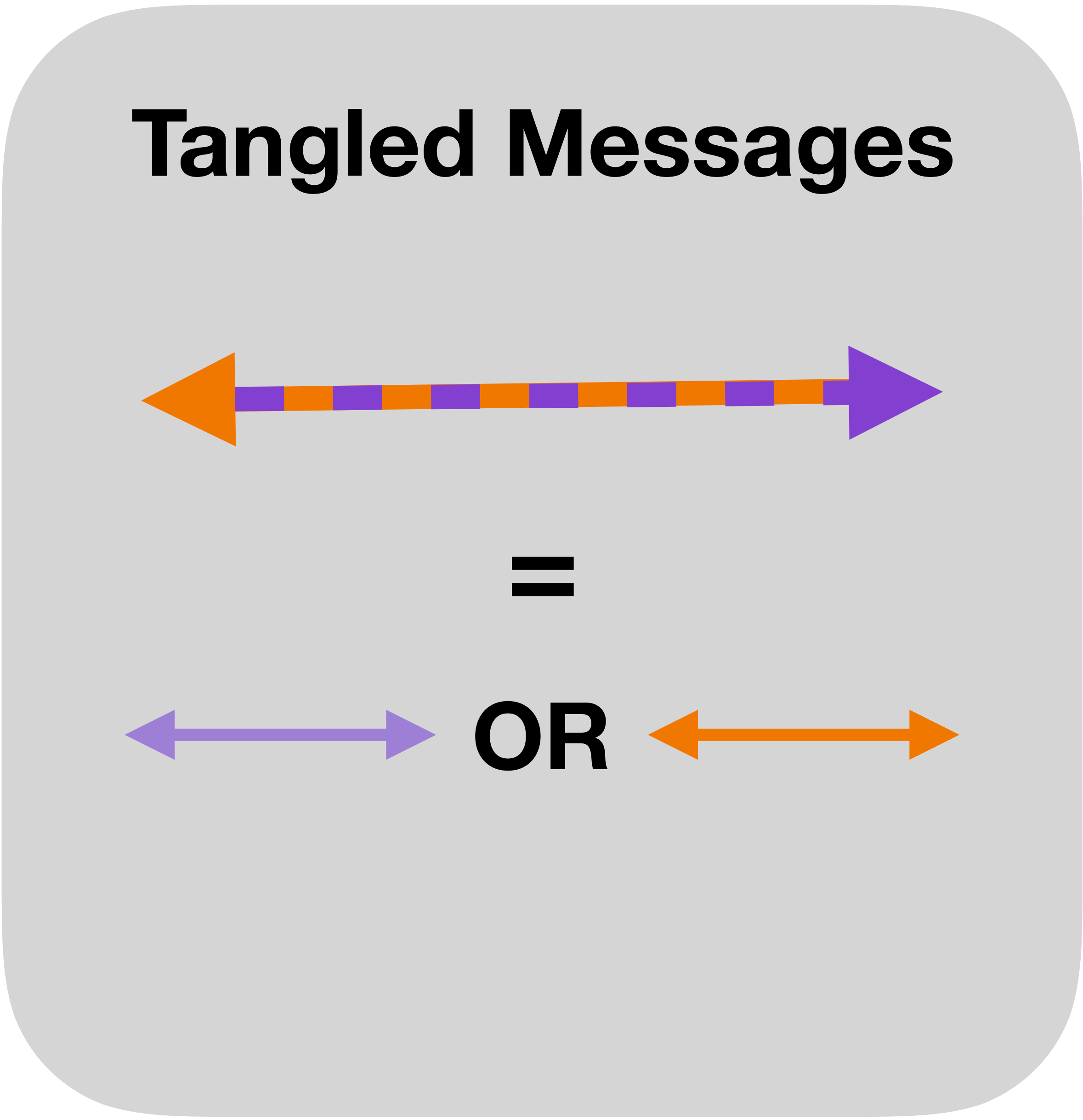


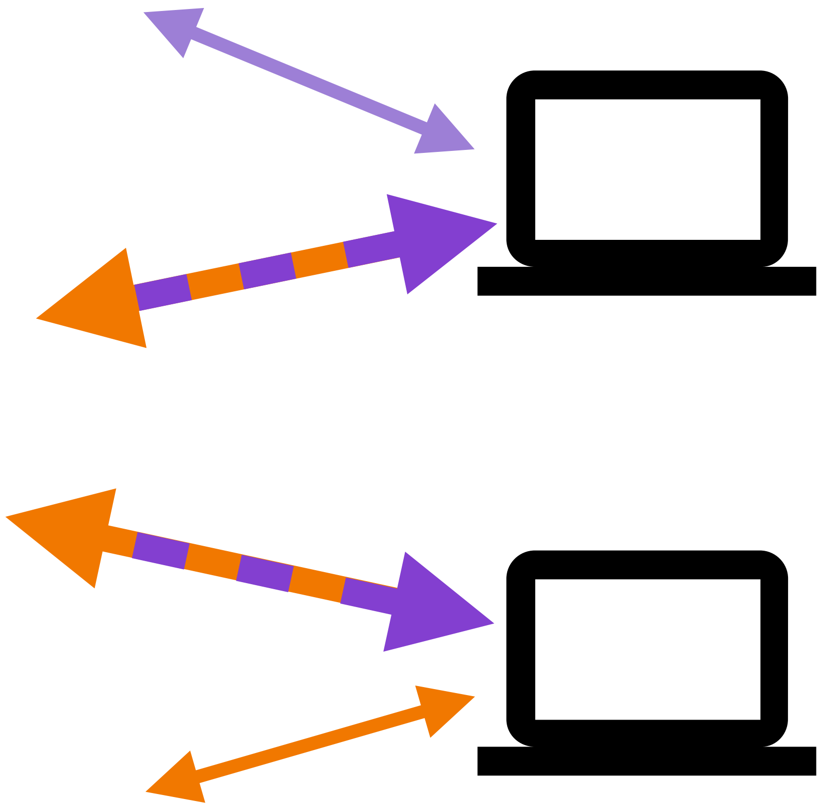
Goal: make these cases indistinguishable so the rest of the links don't matter.

Tangled Messages

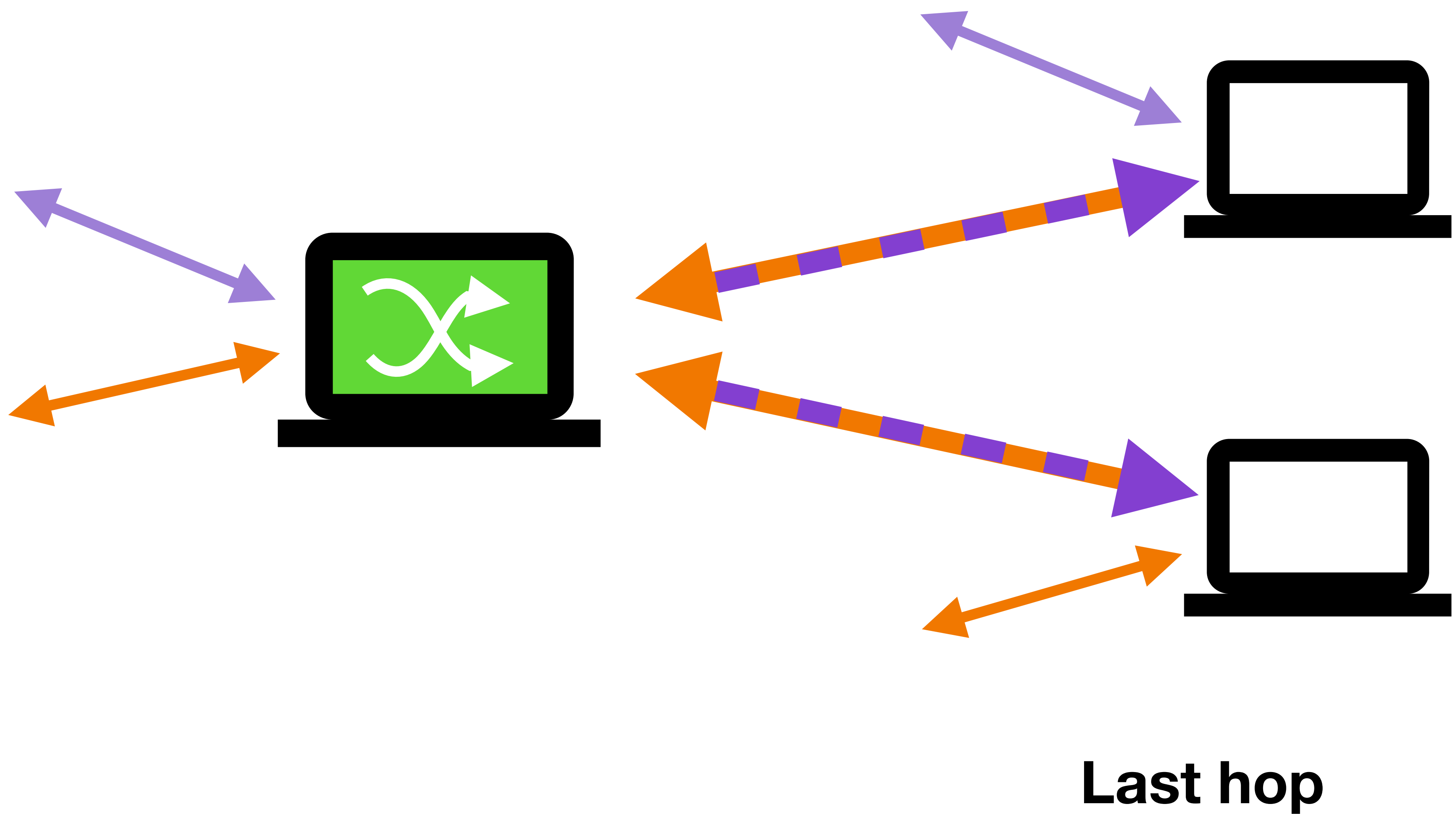


Tangling one of Alice's and one of Bob's messages achieves our goal

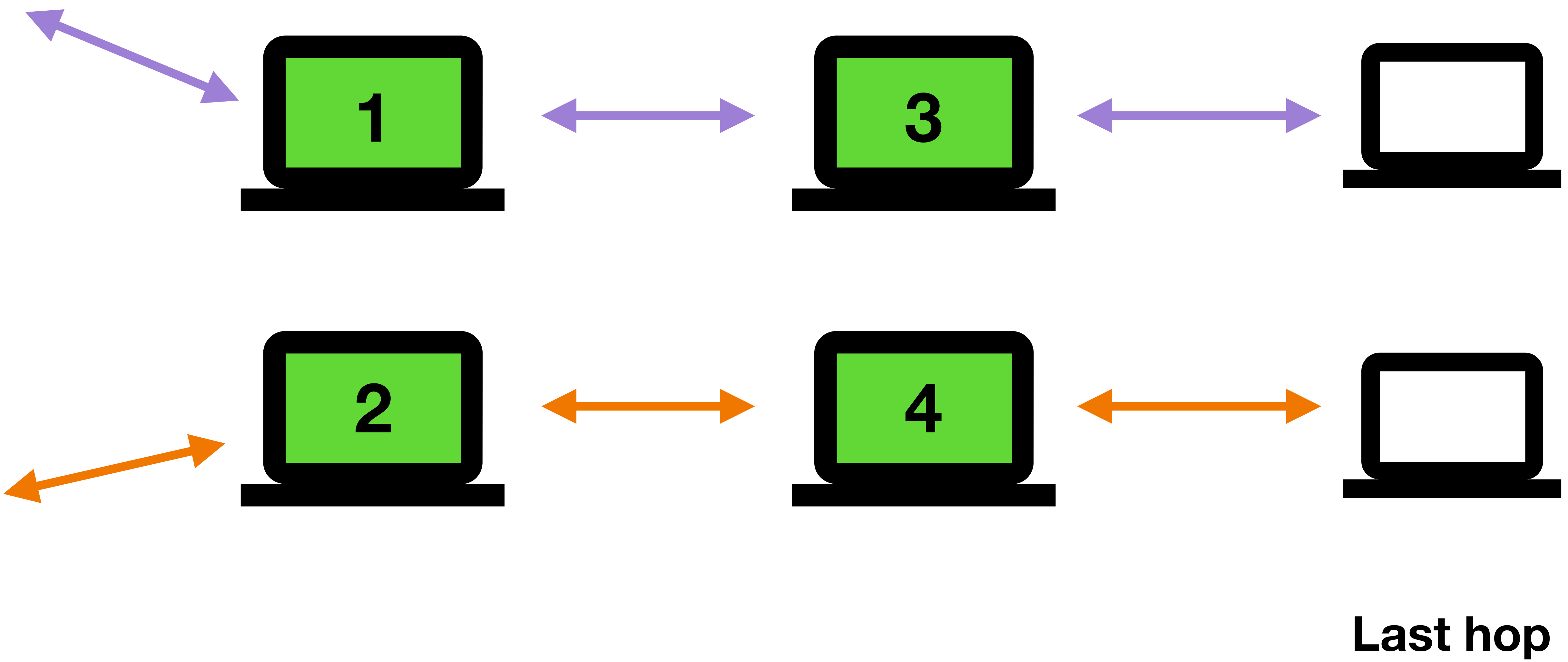




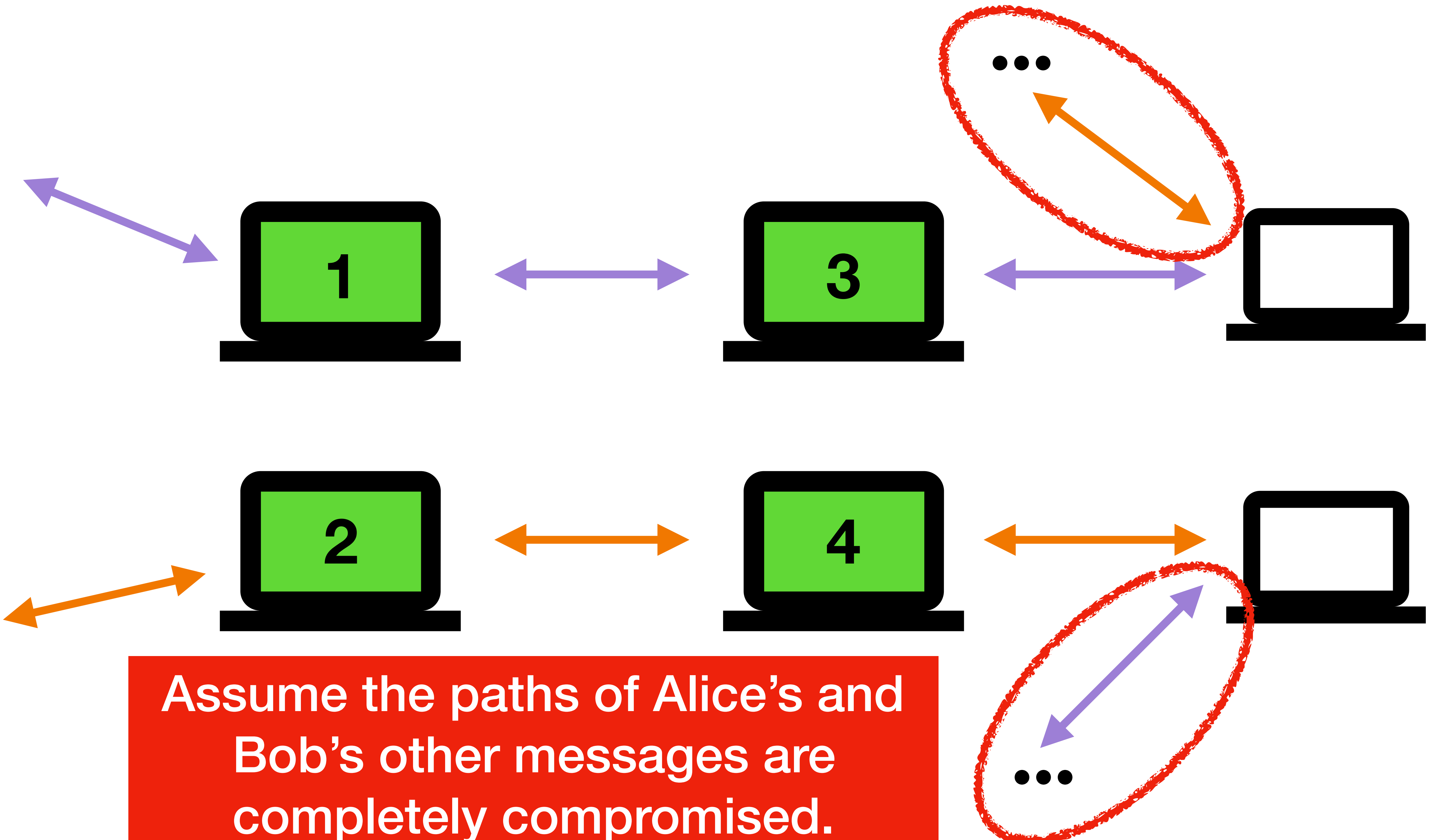
An honest server tangles messages



Problem: Alice and Bob's messages might not intersect at an honest server

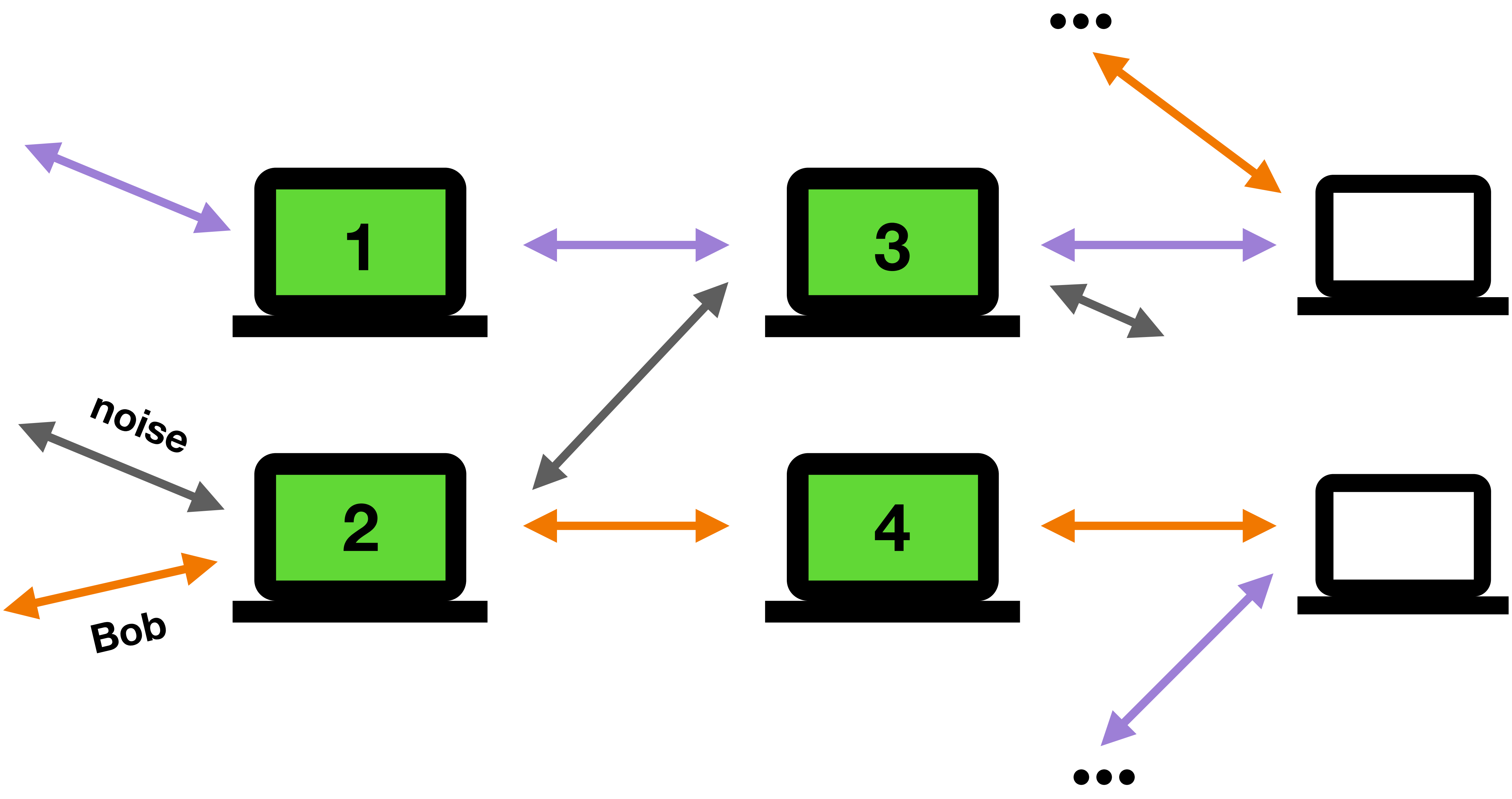


Problem: Alice and Bob's messages might not intersect at an honest server

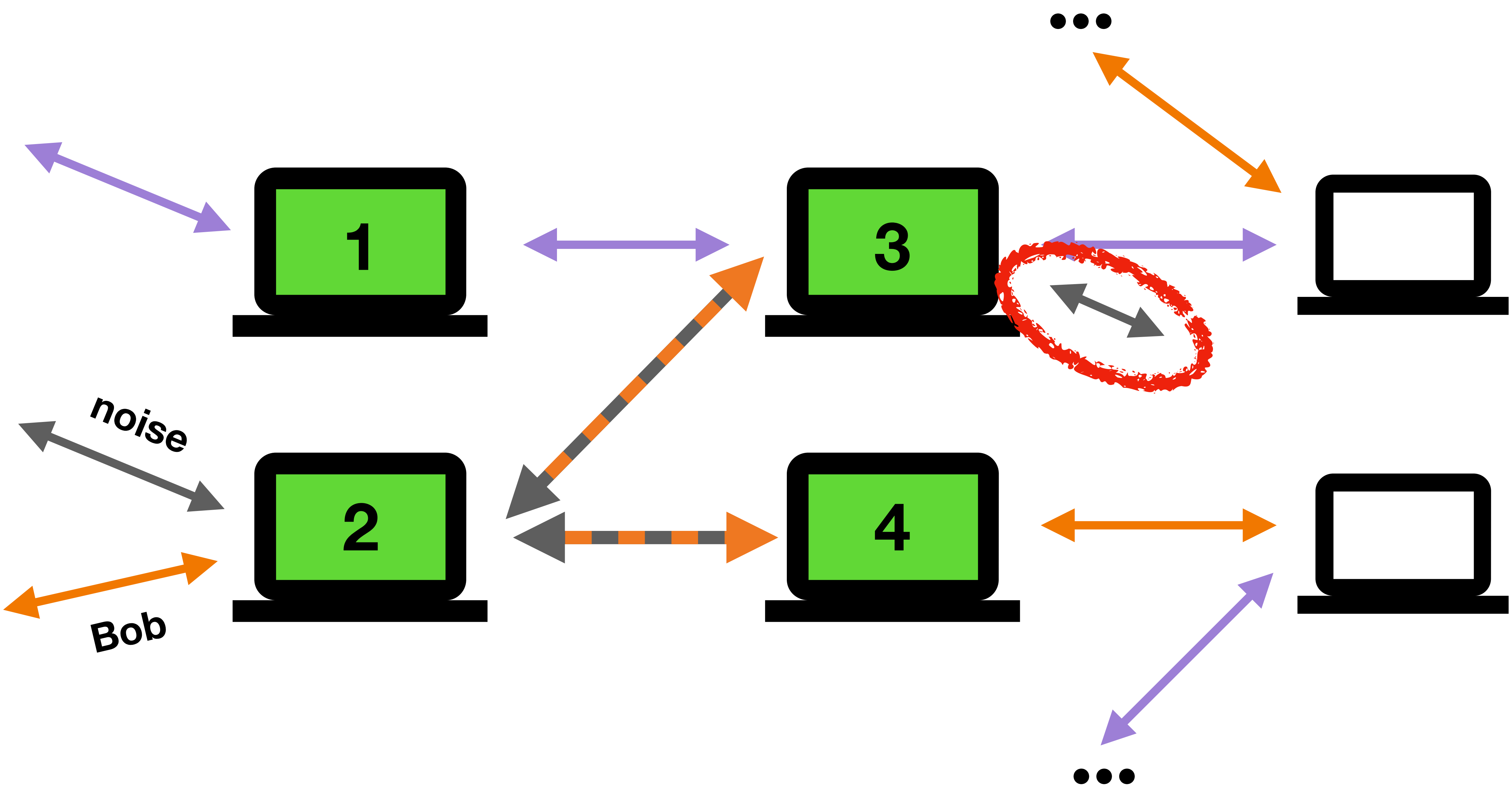


Assume the paths of Alice's and Bob's other messages are completely compromised.

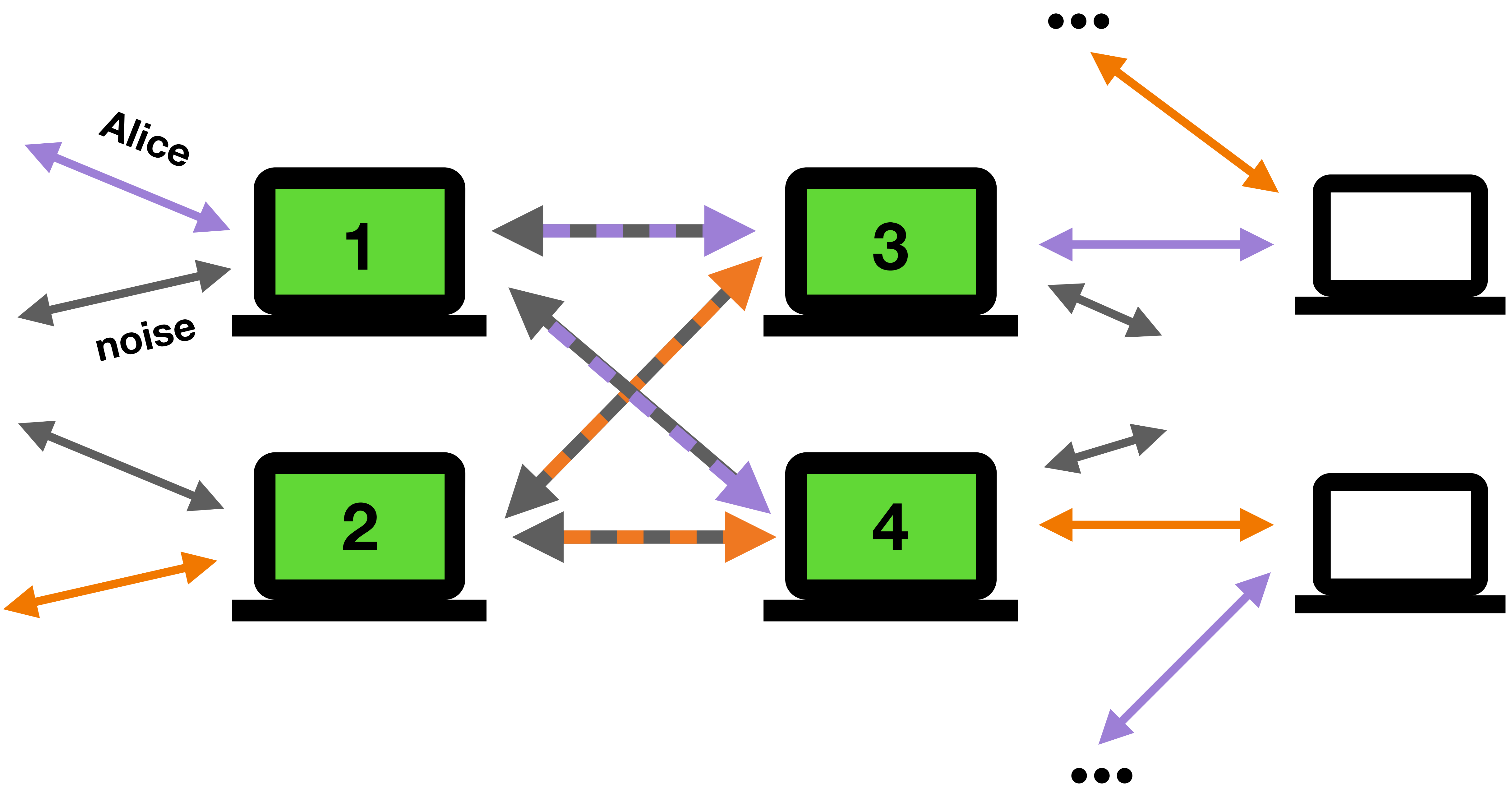
Karaoke servers generate dummy messages that can be used for tangling



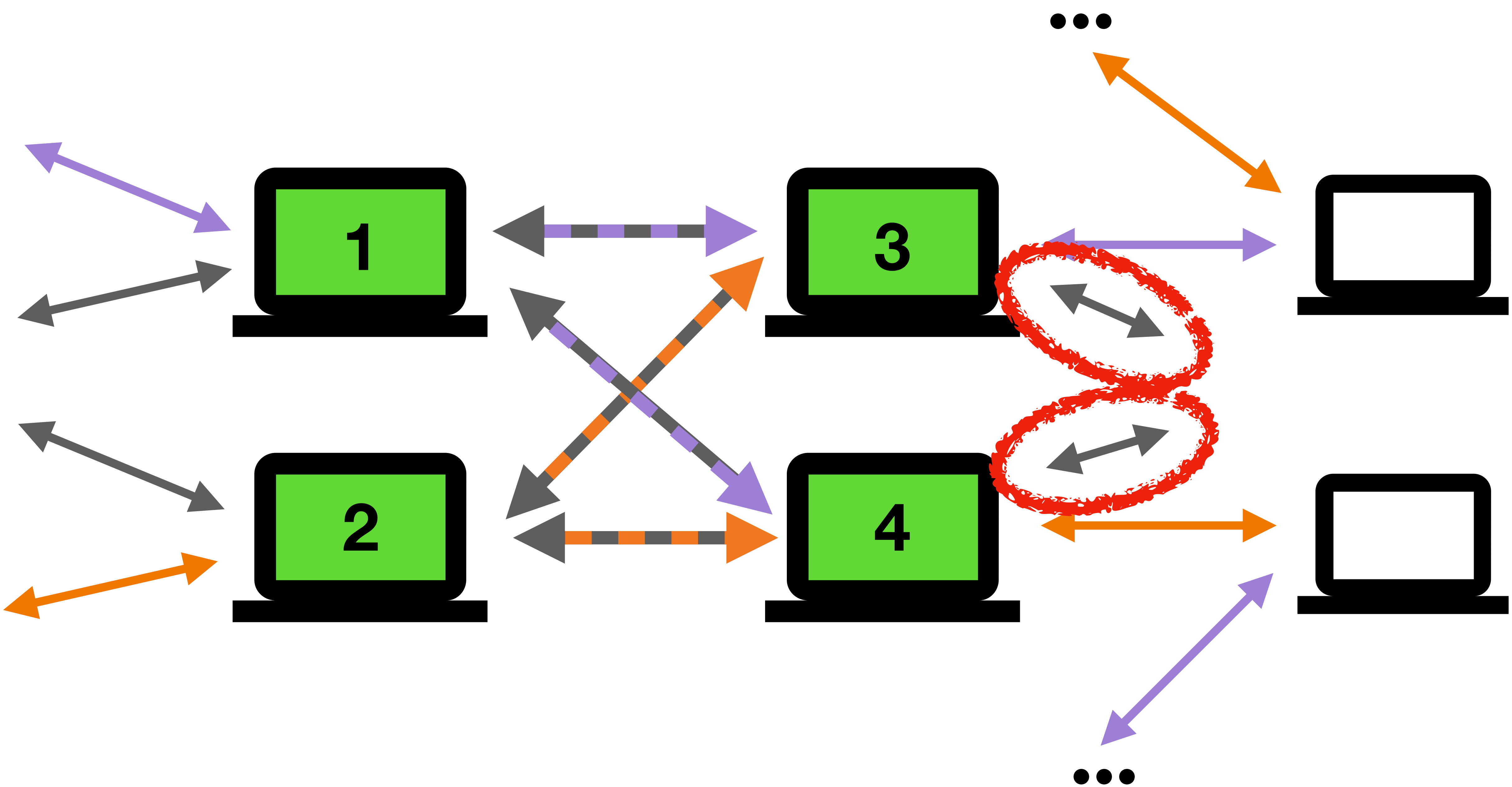
Bob's message is now tangled with noise



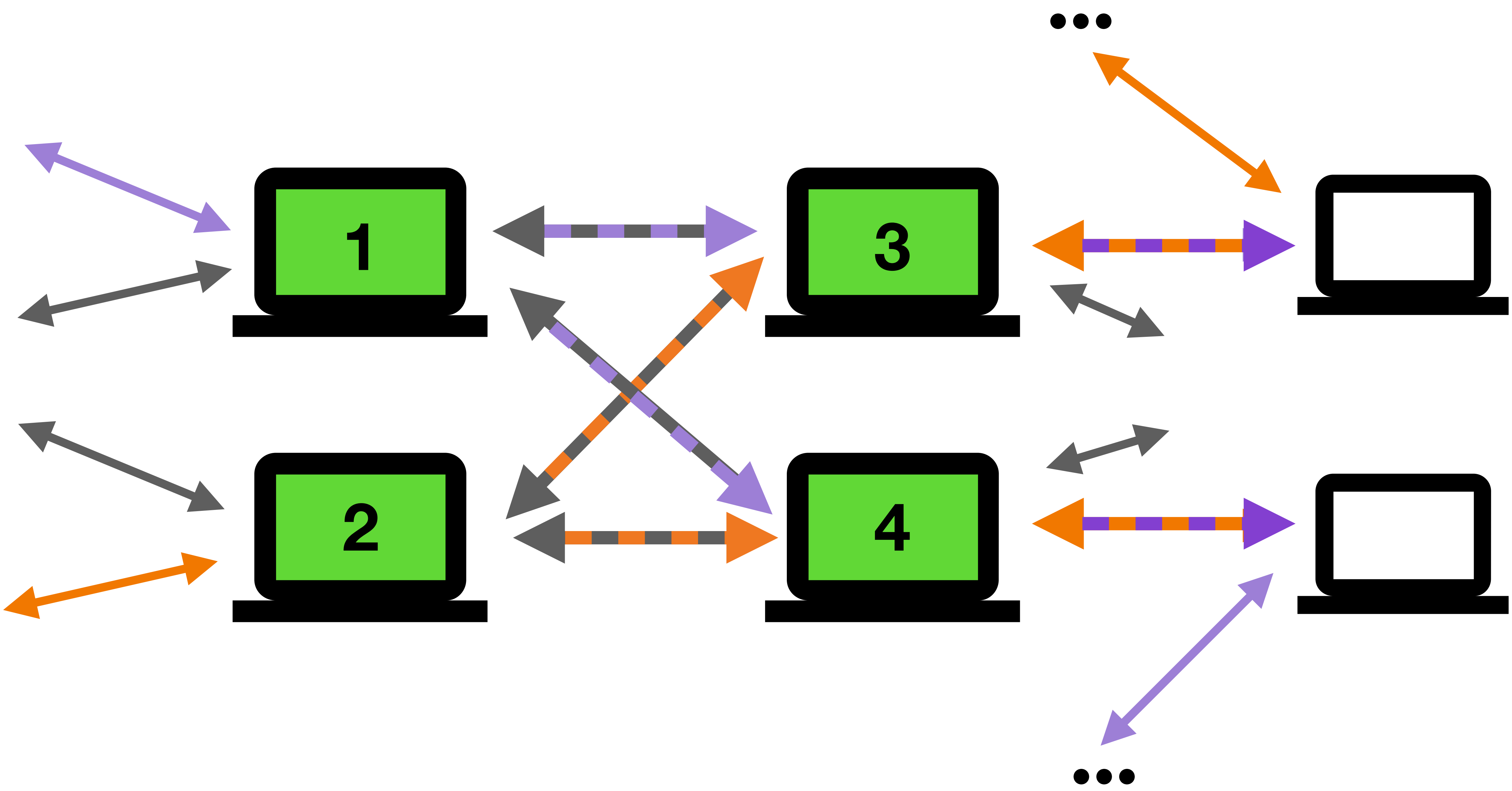
Similarly, Alice's message can tangle with noise



Is it possible that the noise messages swapped places?



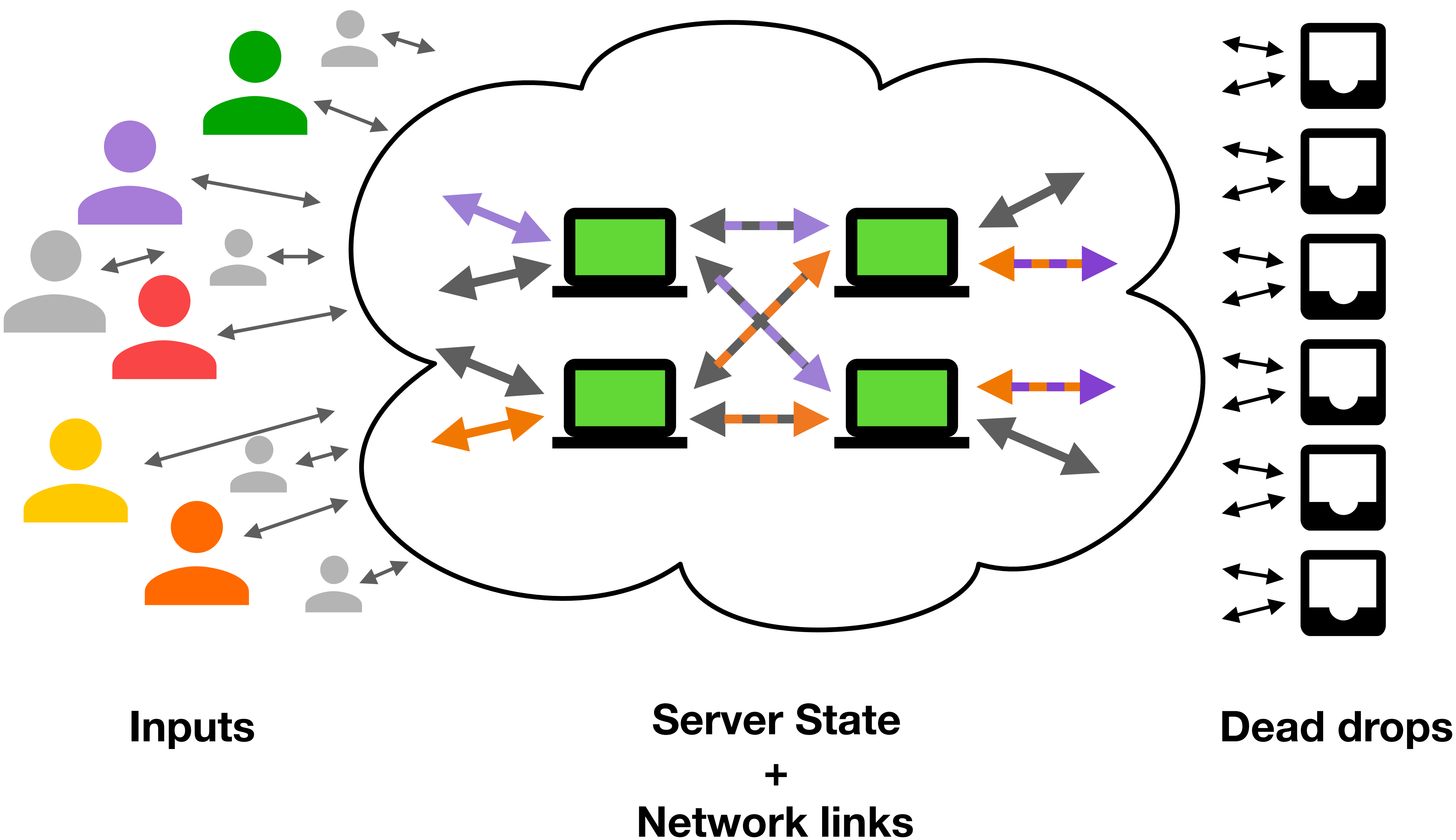
As a result, Alice's and Bob's messages could also have switched places



Tangling with high probability

- The “shape” we just saw is a bit complicated, but it enables Alice and Bob to get tangled with high probability
- Assuming 80% of the servers are honest
  - **14 hops** results in tangling with high probability
  - Servers need to add a small amount of noise messages per outgoing link

Karaoke Summary



Defending against a global active adversary

- Karaoke provides **differential privacy** against a global active adversary

- Karaoke adds additional noise messages to protect against message drops

- Due to message doubling, active attacks (message drops) are rare and detectable, so Karaoke needs far less noise compared to prior work.

- We use bloom filters to ensure malicious servers don't discard the noise. **(See paper)**

Implementation

- 4000 lines of **Go** code
- Major CPU cost is onion decryption
- Configured to resist 200 active attacks per user (see paper)

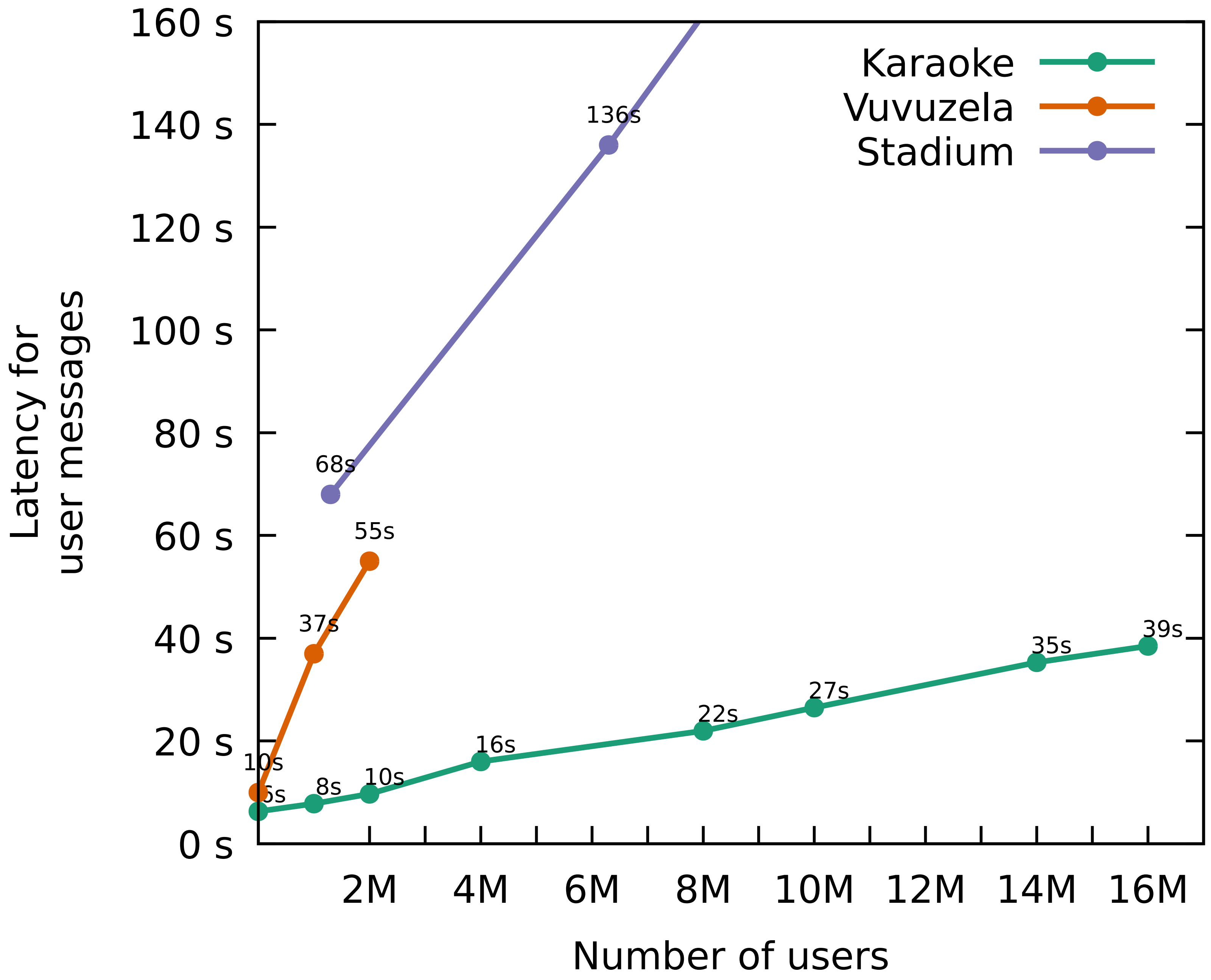
Evaluation

- Does Karaoke support a large number of users with good end-to-end latency?
- How does Karaoke's performance compare to prior work?
- Does it scale? (i.e., does Karaoke support more users by adding more servers?)

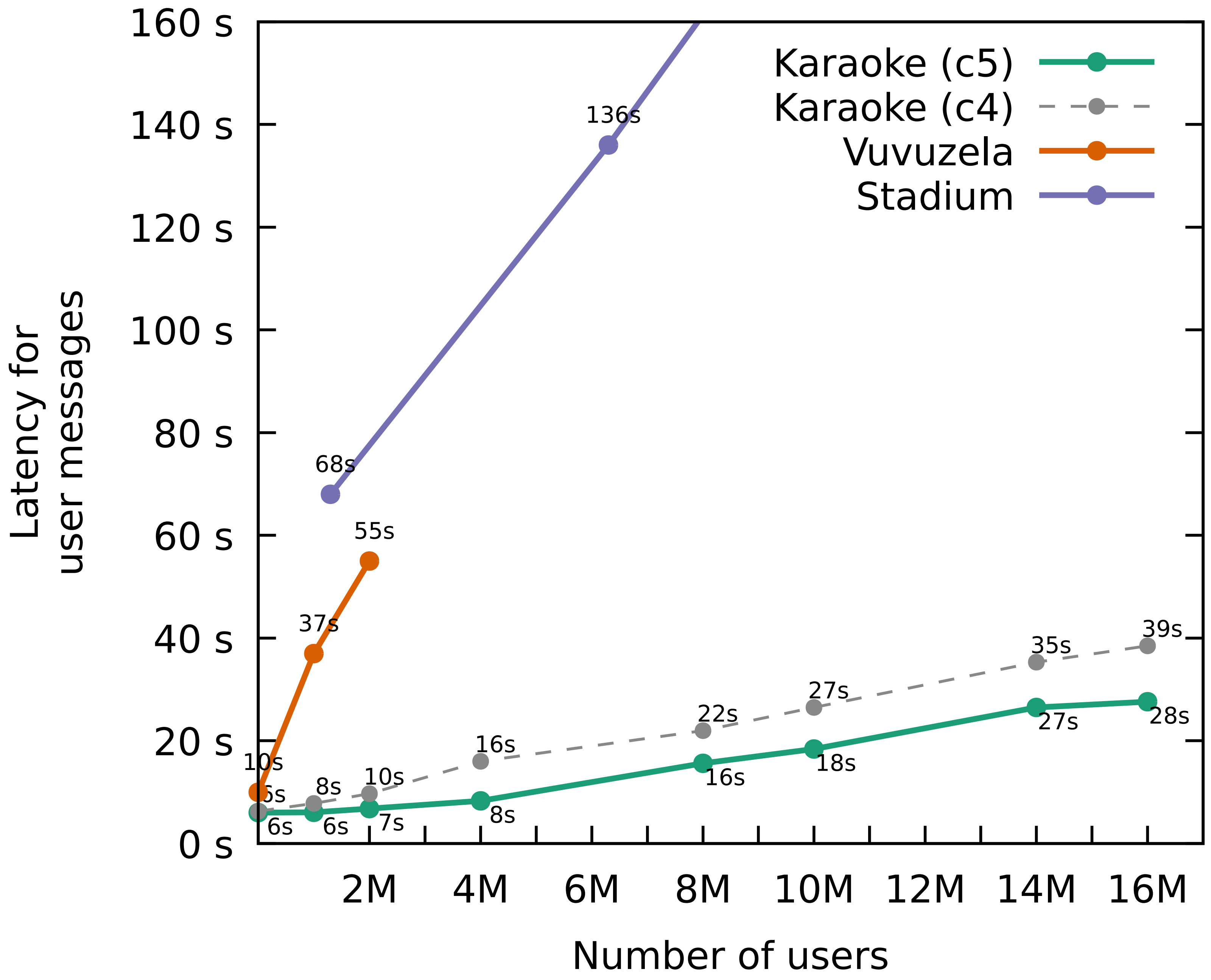
Experimental Setup

- 50 to 200 Amazon EC2 instances
  - **c4.8xlarge** (36 cores) instances for comparison to Vuvuzela and Stadium
  - **c5.8xlarge** instances for all other experiments
- 10 Gbps links
- 100 ms of simulated network latency between instances

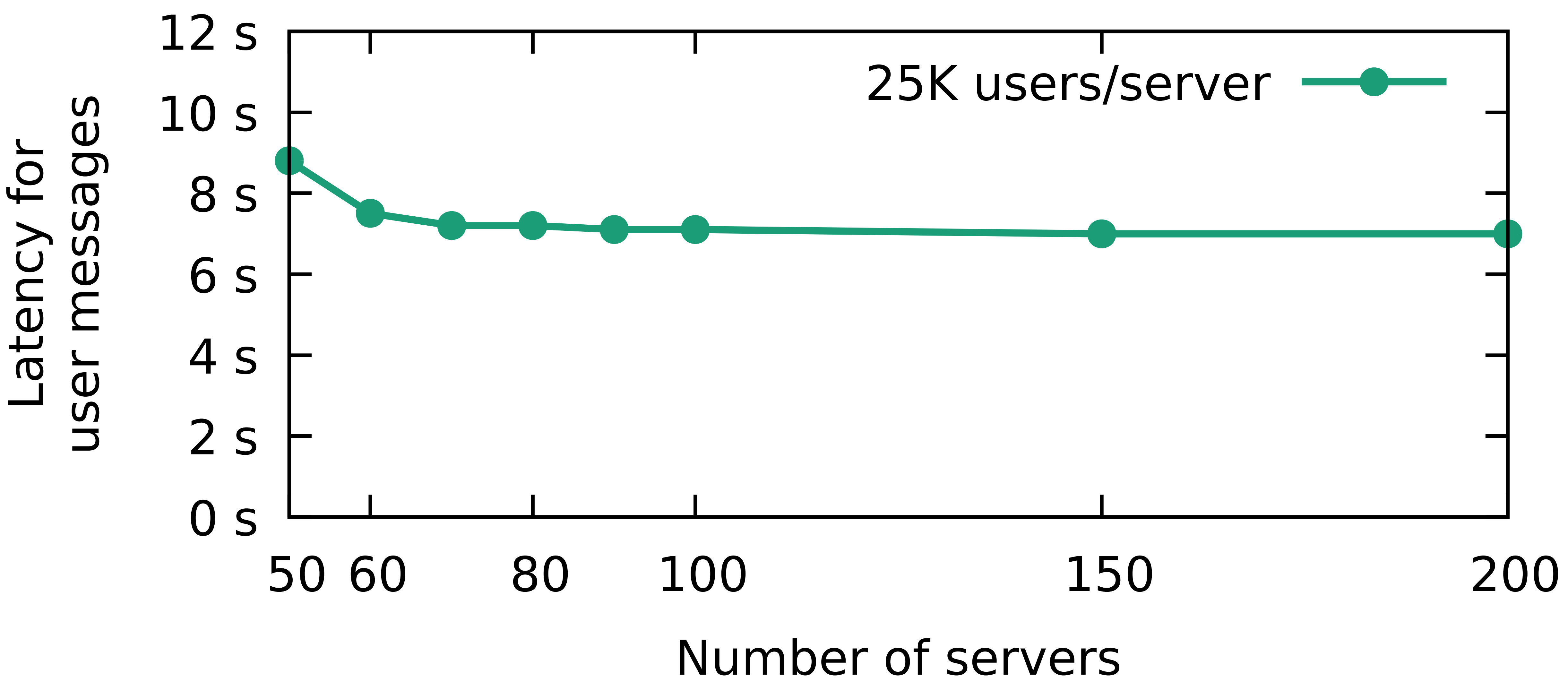
Karaoke achieves low latency for many users



Karaoke is CPU bound



Karaoke supports more users by adding servers



Conclusion

- **Karaoke:** distributed metadata-private messaging system that scales to more people
- Cryptographic privacy against **passive** attackers
  - **Technique:** message doubling + message tangling
- 8 seconds end-to-end latency for 4 million users
  - 5x-11x faster than Vuvuzela/Stadium

https://vuvuzela.io