# Reasoning about Consensus Protocols

Ilya Sergey

ilyasergey.net

# Consensus

- *Common meaning*:
  a way for a **set of parties** to come to a **shared** agreement.
- *In computing*: ensuring that among the values proposed by a collection of processes, a ***single one*** is chosen.
  - **Uniformity**: Only a *single* value is chosen
  - **Non-triviality**: *Only* a value that has been proposed may be chosen
  - **Irrevocability**: Once *agreed* on a value, the processes do not change their decision.

# Why Consensus?

# Why Consensus at SIGPL School?

- Because distributed systems are *correctness-critical software*.
- PL area provides *verification methods* and *language abstractions*.
- Reasoning about correctness of distributed consensus and its applications is a *difficult problem*.

# Why Distributed Consensus is difficult?

- Arbitrary message delays (asynchronous network)
- Independent parties (nodes) can go offline (and also back online)
- Network partitions
- Message reorderings
- Malicious (Byzantine) parties

# Why Distributed Consensus is difficult?

- Arbitrary message delays (asynchronous network)
- Independent parties (nodes) can go offline (and also back online)
- Network partitions
- Message reorderings
- Malicious (Byzantine) parties

# Reaching a Consensus

(and constructing a protocol for this)


Jyoti
Parkview
La Yeon
Hollys
할리스
진양상가아파트
퇴계로
건강약국
SK Energy
SK주유소
7-Eleven
세븐일레븐
삼성중앙약국
명성빌딩
충무초등학교
디자인하우스 자료실
Toegye-ro 56ga-gil
이콘빌딩
대금빌딩
수용빌딩
지지빌딩
GS25
탑빌딩
장충동주민센터
윤창옥내과의원
문화빌딩
고려빌딩
제일병원 동별관
평화빌딩
충무태권도장
법적빌딩
신현빌딩
길성빌딩
신애빌딩
주암빌딩
신일산업빌딩
Buy the Way
바이더웨이
블루클럽
Jangchungdan-ro 6-gil
Jangchungdan-ro 4-gil
광성빌딩
토픽빌딩
필빌딩
동국약국
충무아파트
한국학생도서관
이성빌딩
성일빌딩
디오빌딩
장춘단교회
SK Energy
SK주유소
Paris Baguette
파리바게뜨
동영빌딩
CU
그랜드앰배서더호텔
정담빌딩
상지빌딩
파크원빌
해동빌딩
수월관음도
문중빌딩
대진빌딩
금강반야바라밀경
만해광장
동우빌딩
파라다이스빌딩
평안빌딩
원일빌딩
미도파빌딩
Seoae-ro 3-gil
3호선
허브레지던스
Dongho-ro
BH빌딩
서울불광사사
Caffebene
카페베네
Dongho-ro 20na-gil
아싸컴빌딩
리금빌딩
케냐명예총영사관
성실빌딩
Dongguk University
동대입구역
장충레지텔
동호로
장충교회
중국대사관 경제상무처
한아빌딩
진성빌딩
장충아파트
Dongho-ro 20da-gil
낙원빌딩
성운빌딩
Dongho-ro 25ga-gil
장충파출소
파리재활의학과의원
환경빌딩
설악약국
우진빌딩
Pildong-ro 1-gil
일일부독서구중생형극
동국대학교 대학원
삼진빌딩
아세아빌딩
동국대학교 서울캠퍼스
Dongho-ro 18-gil
평강빌딩
수표교
Papa John's
파파존스피자
Jangchungdan-ro
장충단공원
장충단비
동국대학교 문화예술대학원
인봉빌딩
상원빌딩
납석삼존불비상
동국대우체국
Dasan-ro 19
조암빌딩
원성빌딩
성도교회
보협인석탑
신라면세점
대현빌딩
필동빌딩
Paris Baguette
파리바게뜨
다담에뜰
남산빌딩
라비두스
호텔신라우편취급국우체국
야외조각공원
호텔신라 영빈관
동원빌딩
태호빌딩
장충수영장
Dongho-ro 17-gil
Oilbank
오일뱅크
미주아파트
신당동교회

# Reaching a Consensus on where to have a dinner

Jyoti

La Yeon

Parkview

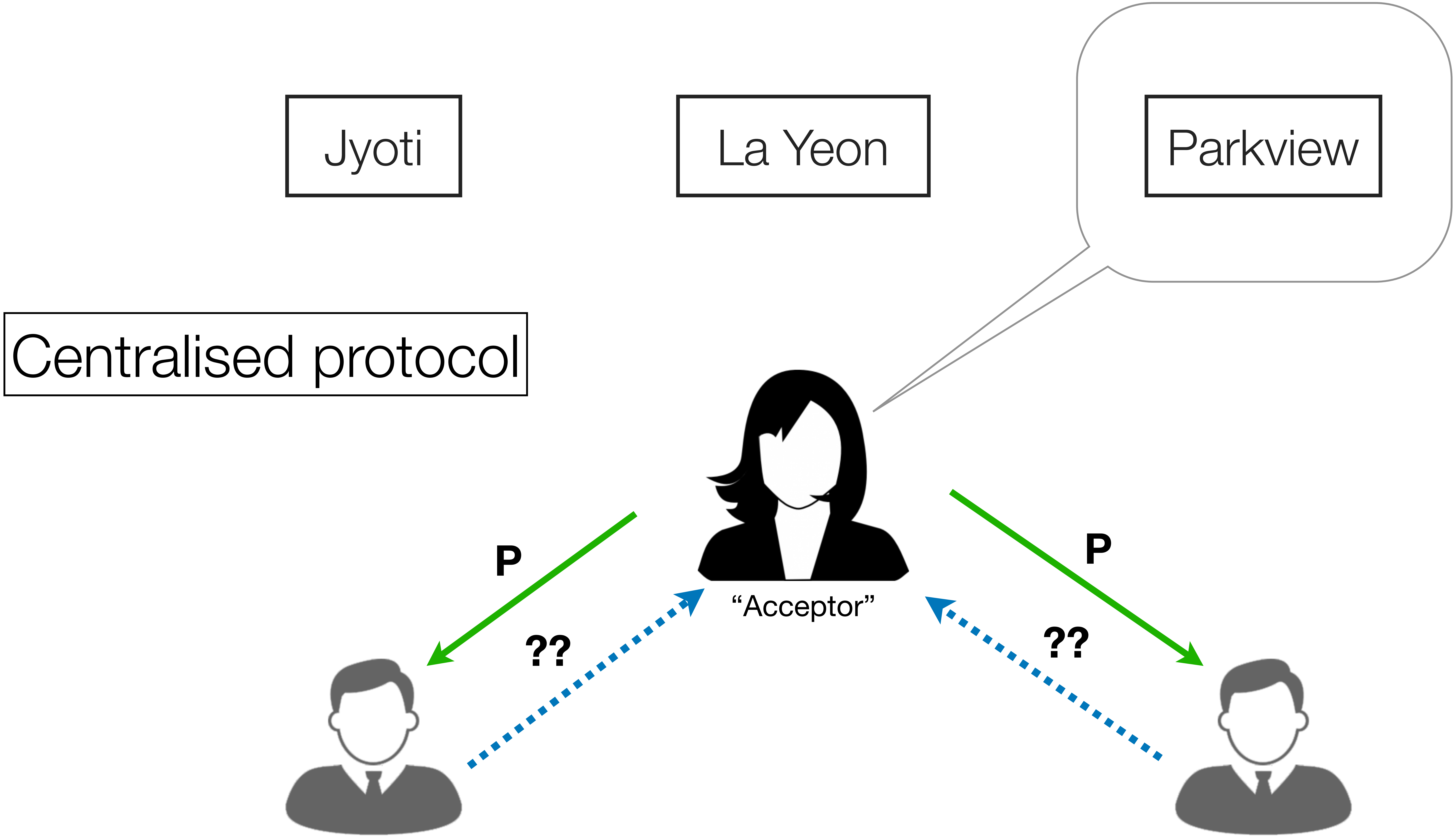
Jyoti

La Yeon

Parkview

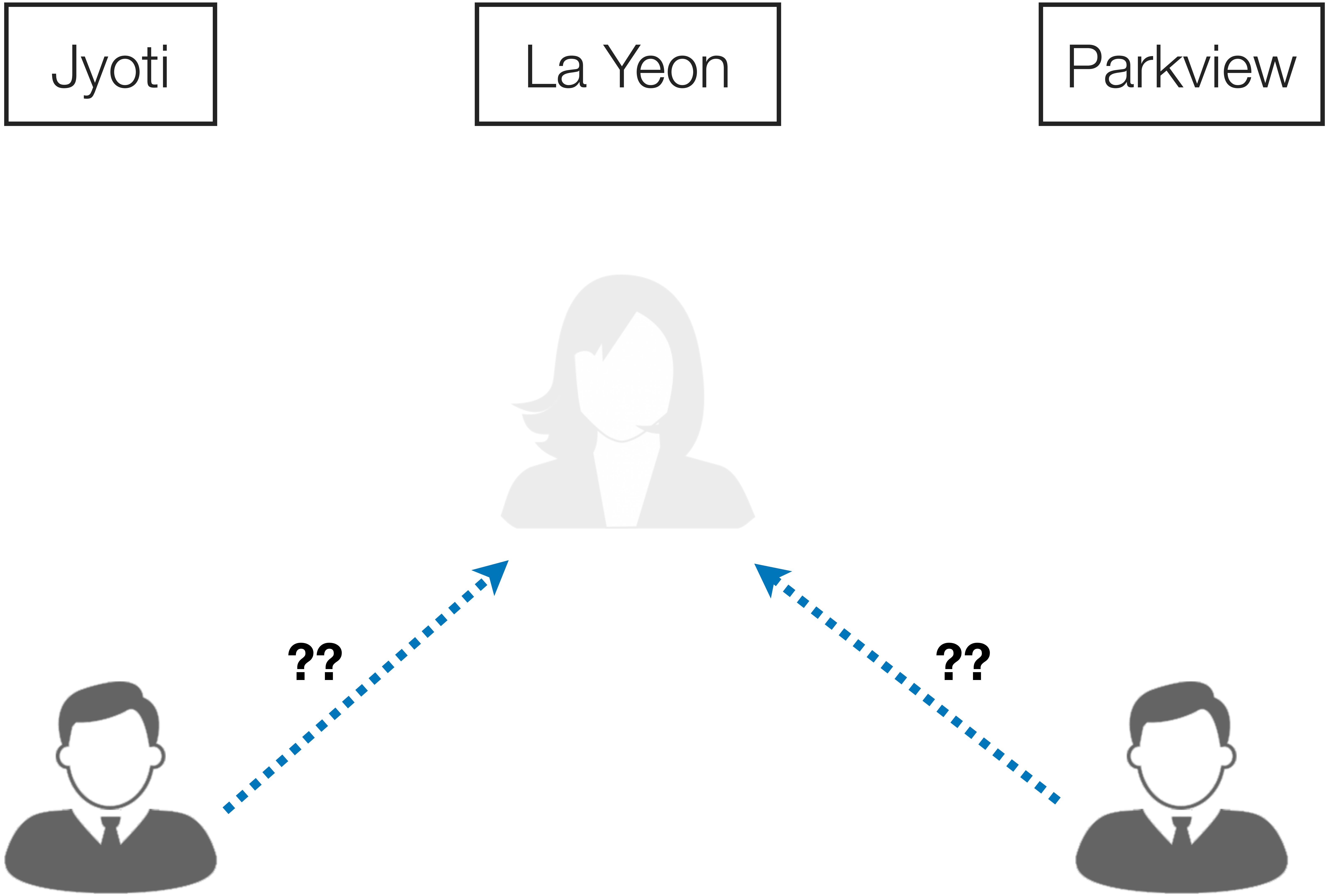
??

??

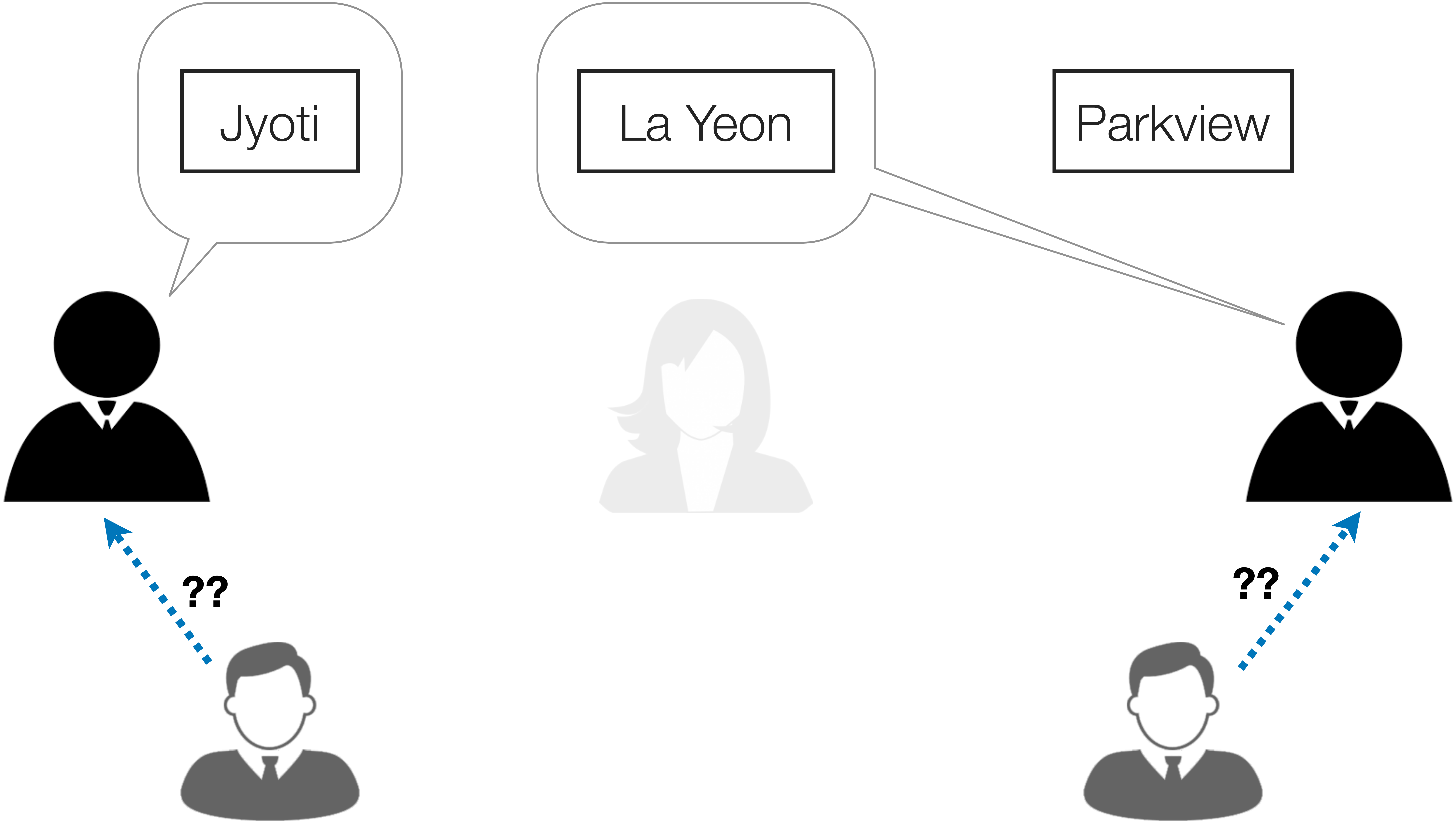

Jyoti
La Yeon
Parkview
Centralised protocol
P
P
“Acceptor”
??
??

# Problem 1

A single acceptor can *go offline* or *take forever* to answer.



Jyoti
La Yeon
Parkview
??
??


Jyoti
La Yeon
Parkview
??
??

# Problem 2

Multiple *acceptors* might disagree on the outcomes:

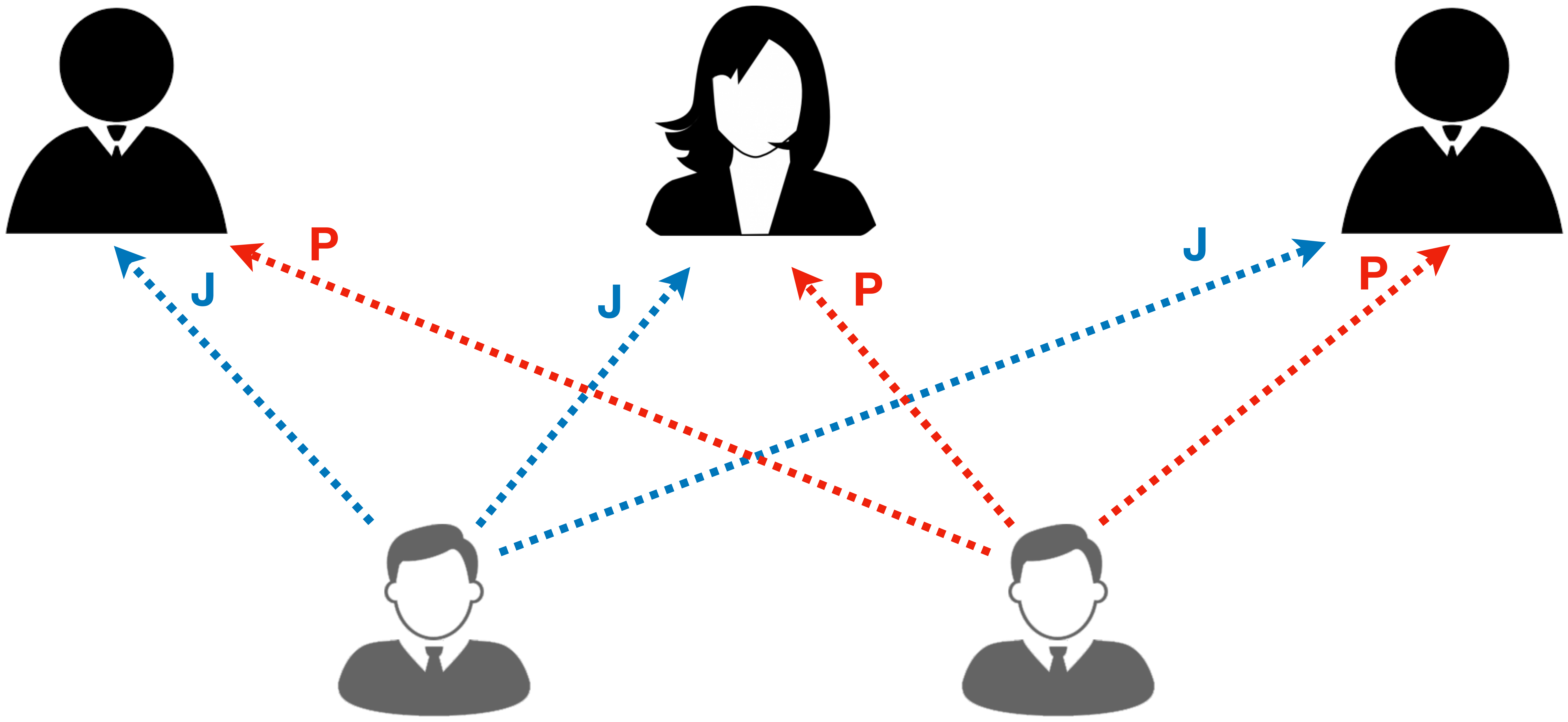
now they need to *reach a consensus* themselves.

# Separation of Concerns

- **Proposers**: suggest a value (a restaurant to go);
- **Acceptors**: support some proposal;
- The **proposer** with a *majority of acceptors* supporting its proposal wins.

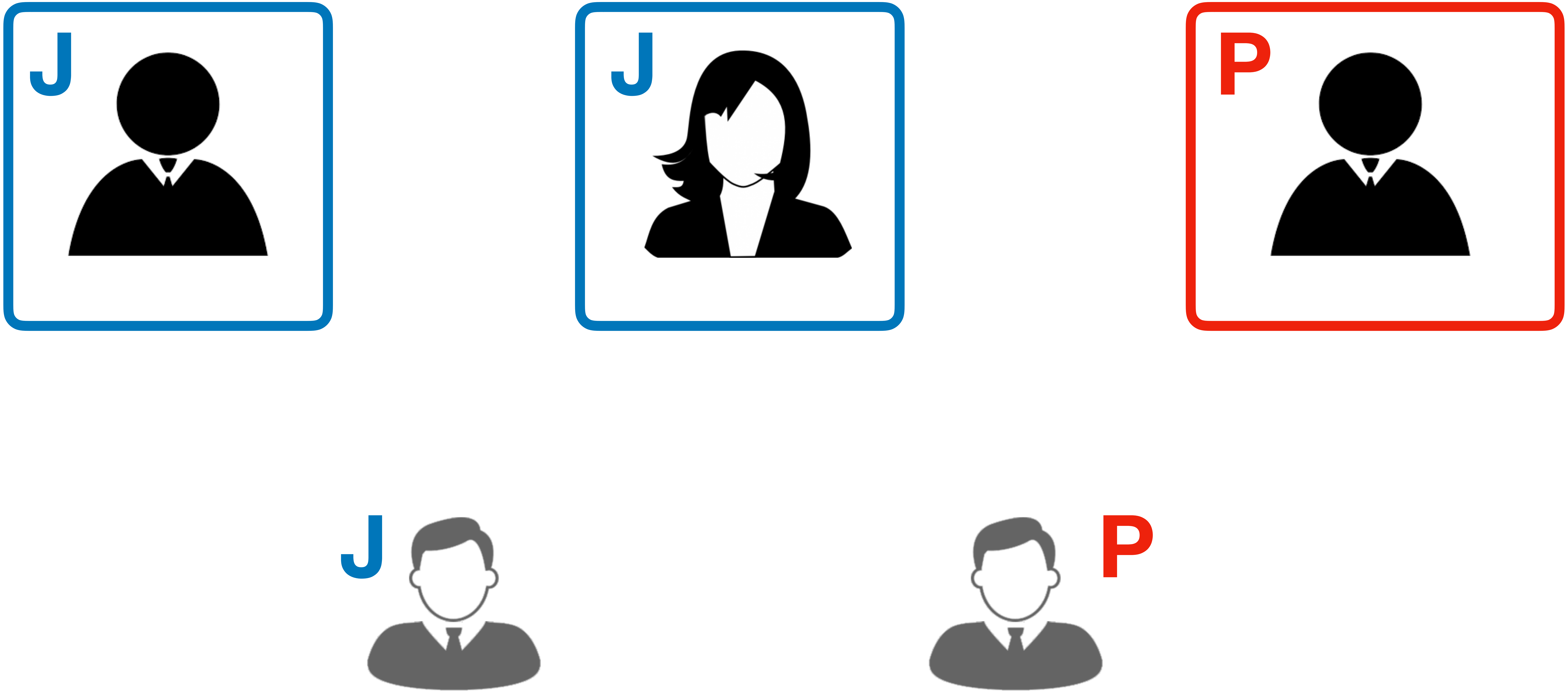
  Others learn the outcome by querying all the acceptors.

Acceptors




Proposers

Acceptors




Proposers

# Key Idea 1

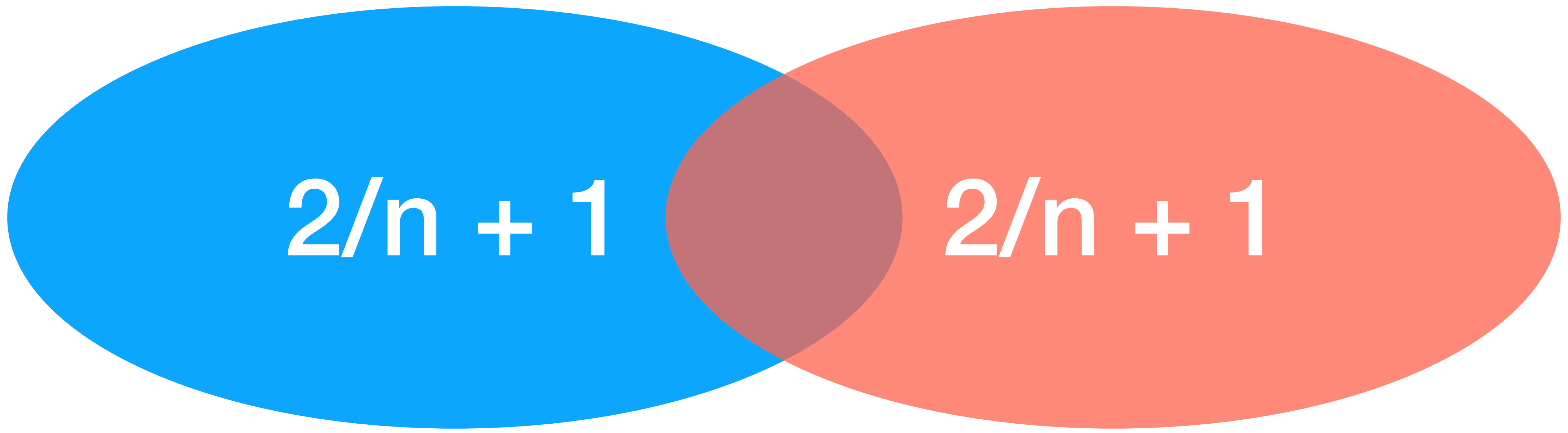
Rely on **majority quorums** for agreement

to prevent the “split brain” problem.

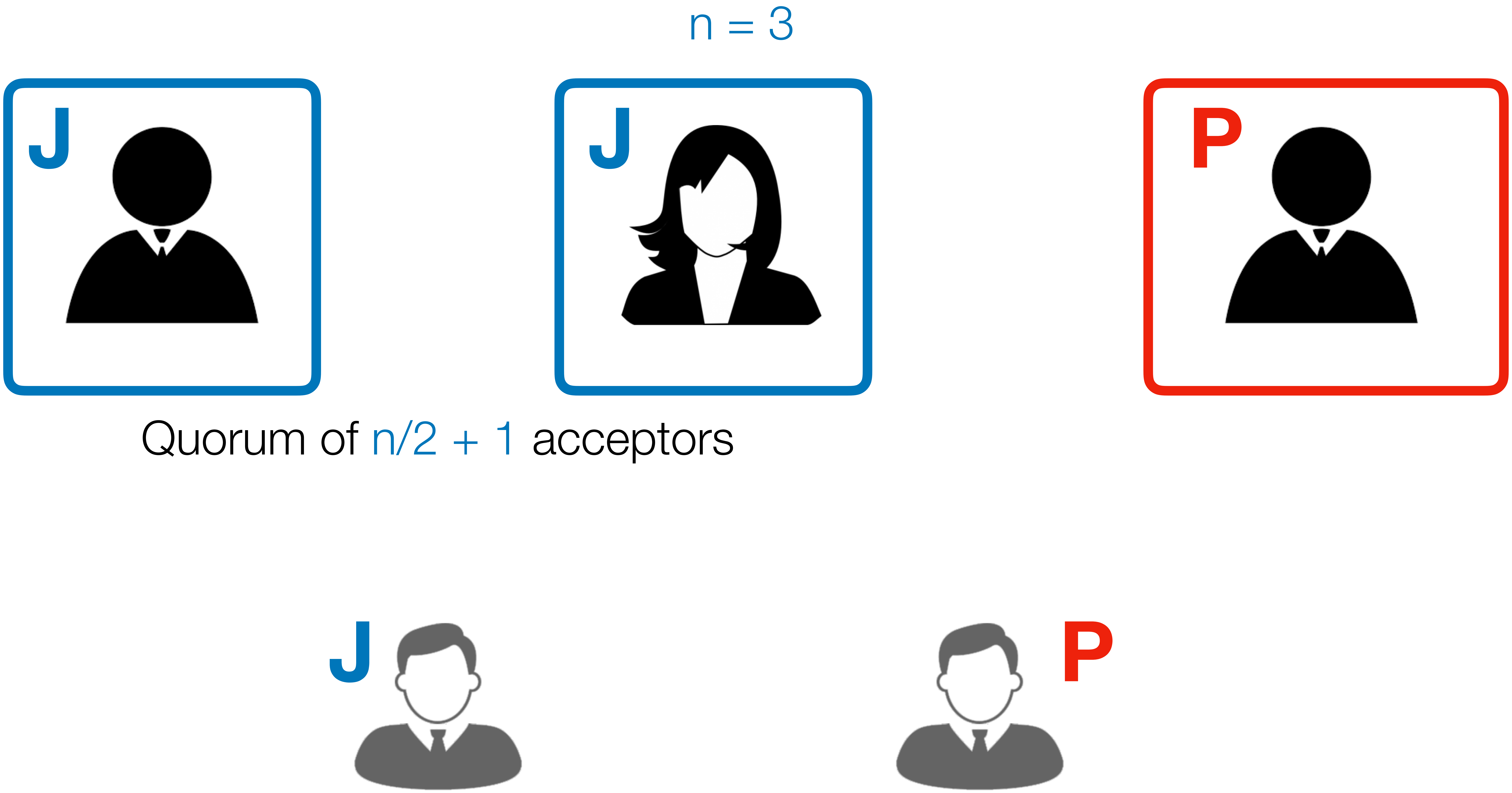
- *Common meaning*: Quorum is the minimum number of members to conduct the business on behalf of the entire group they represent;
- *In computing*: quorum is *a necessary number of processes* to agree on the decision in the presence of potentially faulty ones.

# Key Properties of Quorums

- *Property 1:* any two quorums must have non-empty intersection




- *Property 2:* no need for the *global* agreement: can tolerate some faults



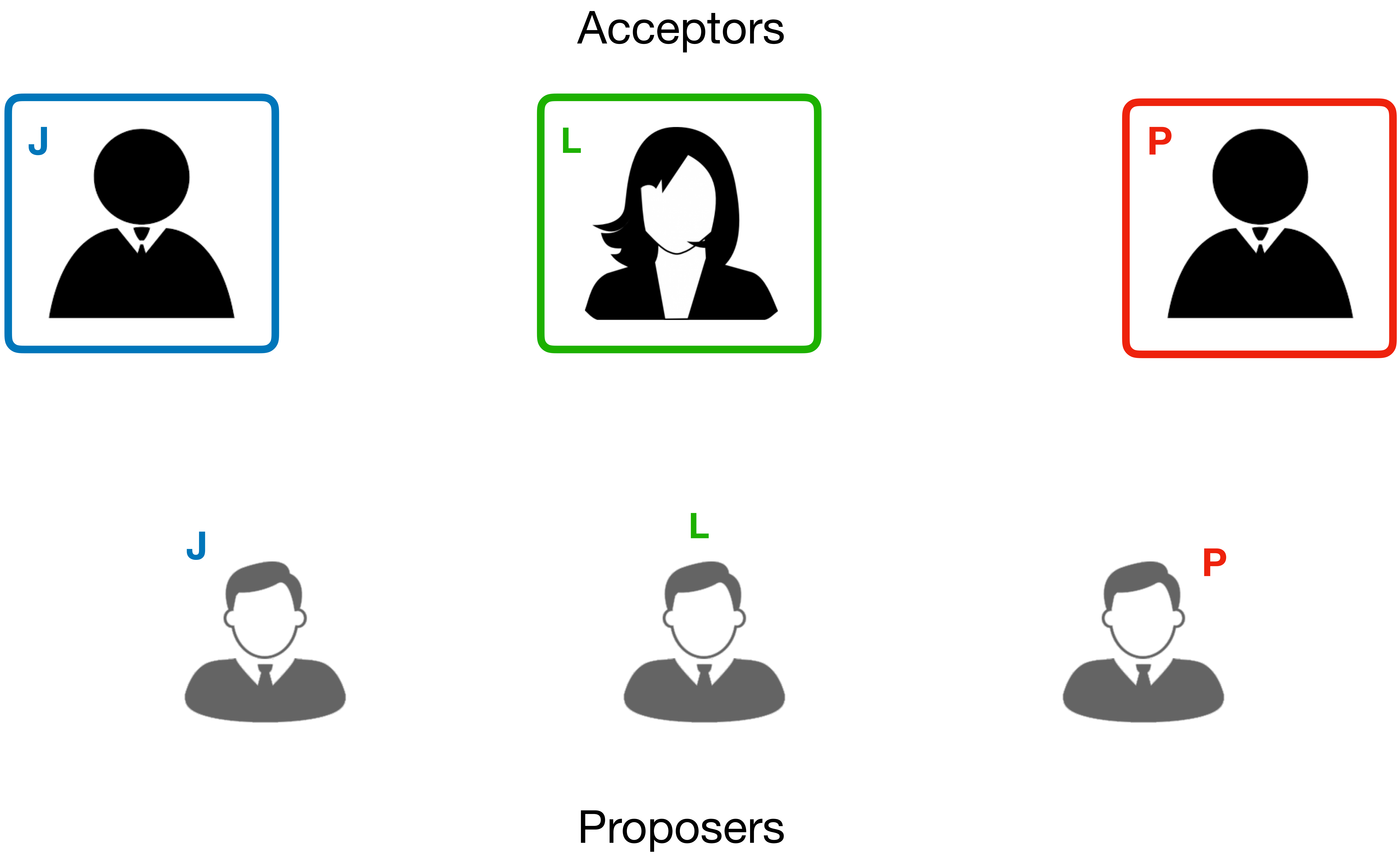
n = 3
J
J
P
Quorum of n/2 + 1 acceptors
J
P

# Problem

A quorum is difficult to obtain in a *single* interaction.

As the result, such a system will often *get stuck*.


Acceptors
J
L
P
J
L
P
J
L
P
Proposers


Acceptors
J
L
P
J
L
P
Proposers

# Key Ideas 2 and 3

- Proceed in rounds:
  - A proposer first “secures” itself a quorum, willing to support its proposal (i.e., becomes a “leader”);
  - *Only if a quorum is secured*, it goes on to “propose” a value.

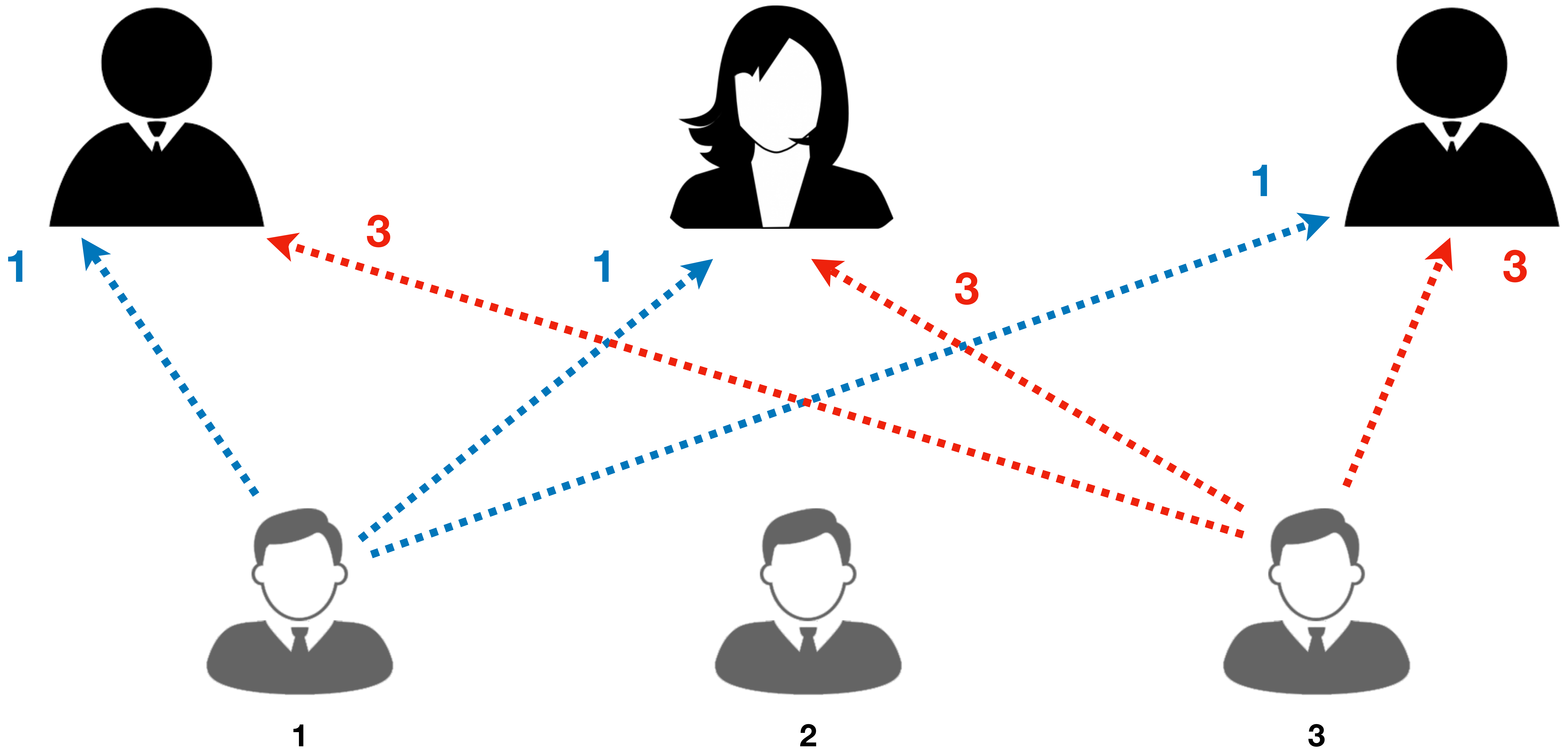
- Introduce fixed globally known *priorities* between proposers to “break ties” when securing quorums.
  - Acceptors only “choose to support” proposers with higher priorities than they have already seen.

# Some Terminology

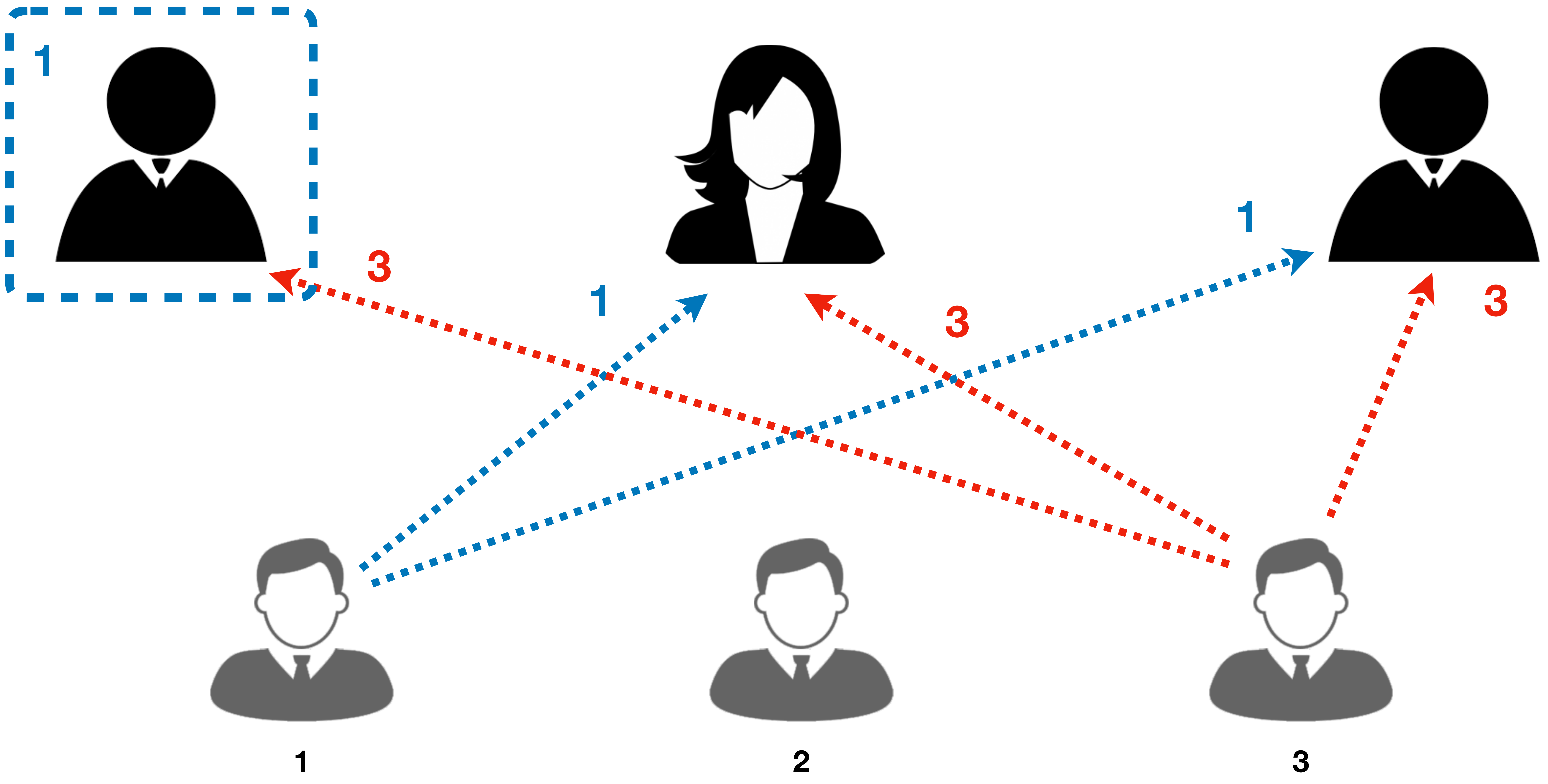
- Rounds — **Phases**
  - Phase 1 — “prepare”, securing quorums to propose
  - Phase 2 — “accept”, sending values to accept

- Fixed priorities — **Ballots**

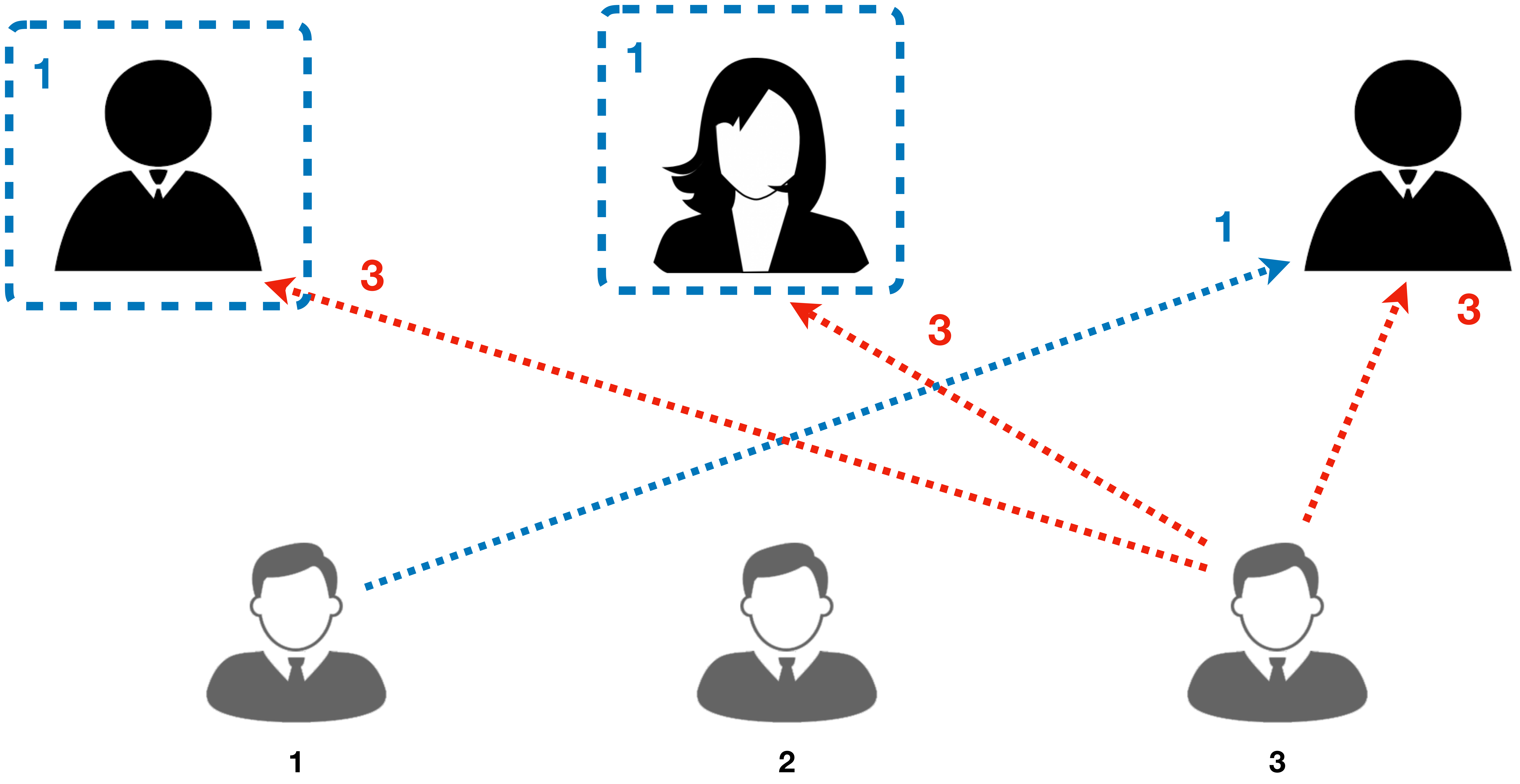
# Phase 1



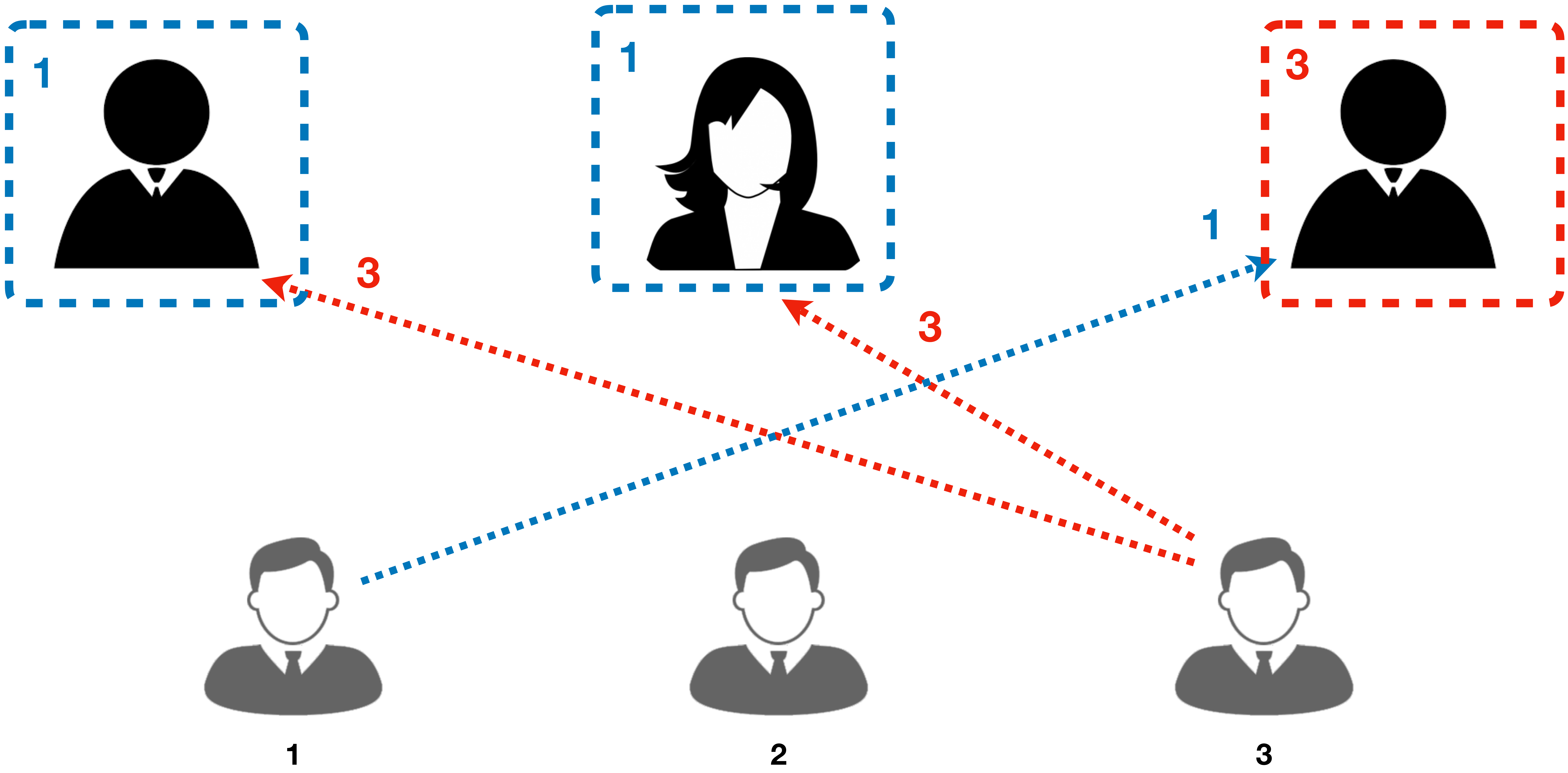
## Phase 1



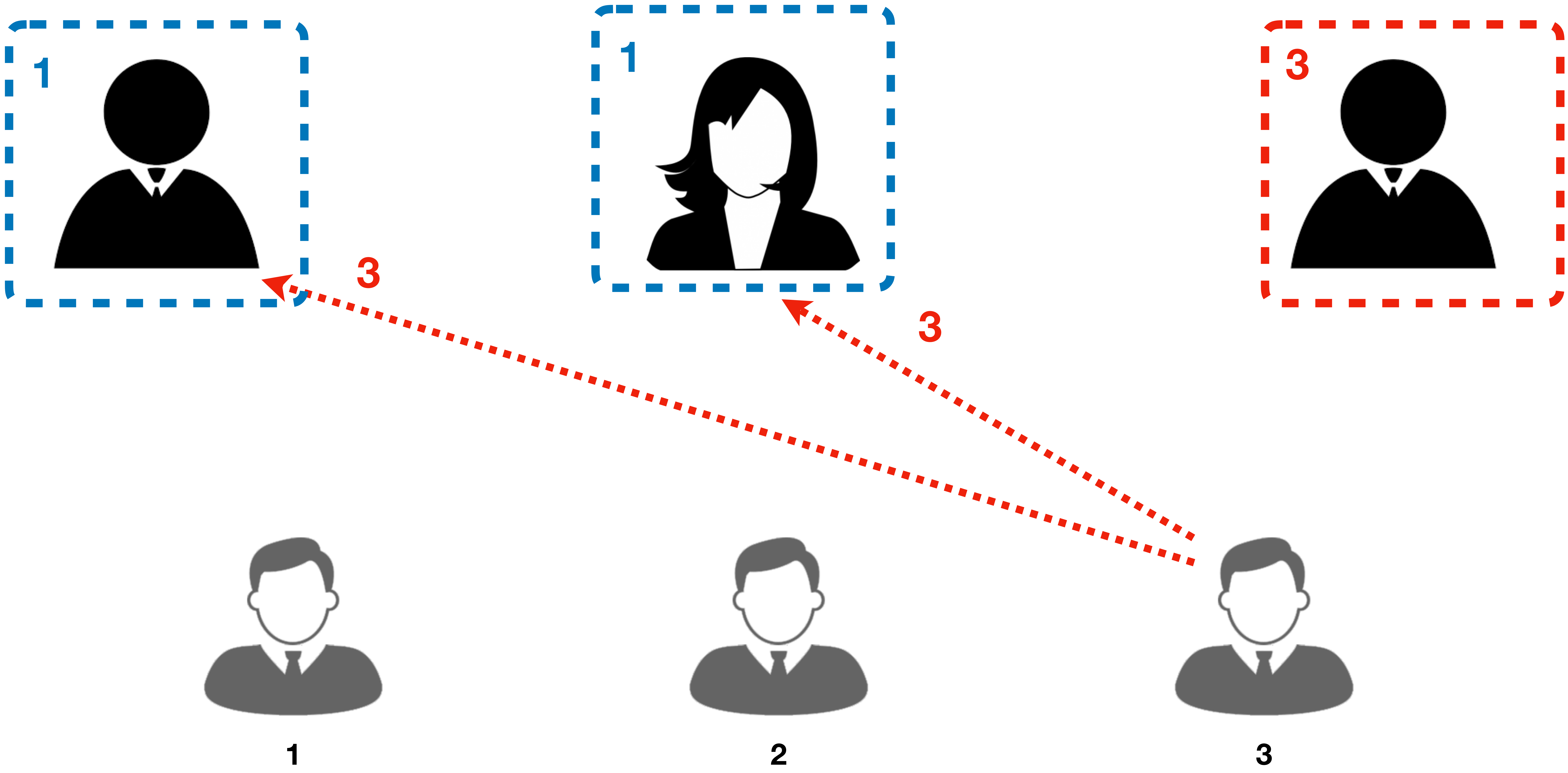
# Phase 1



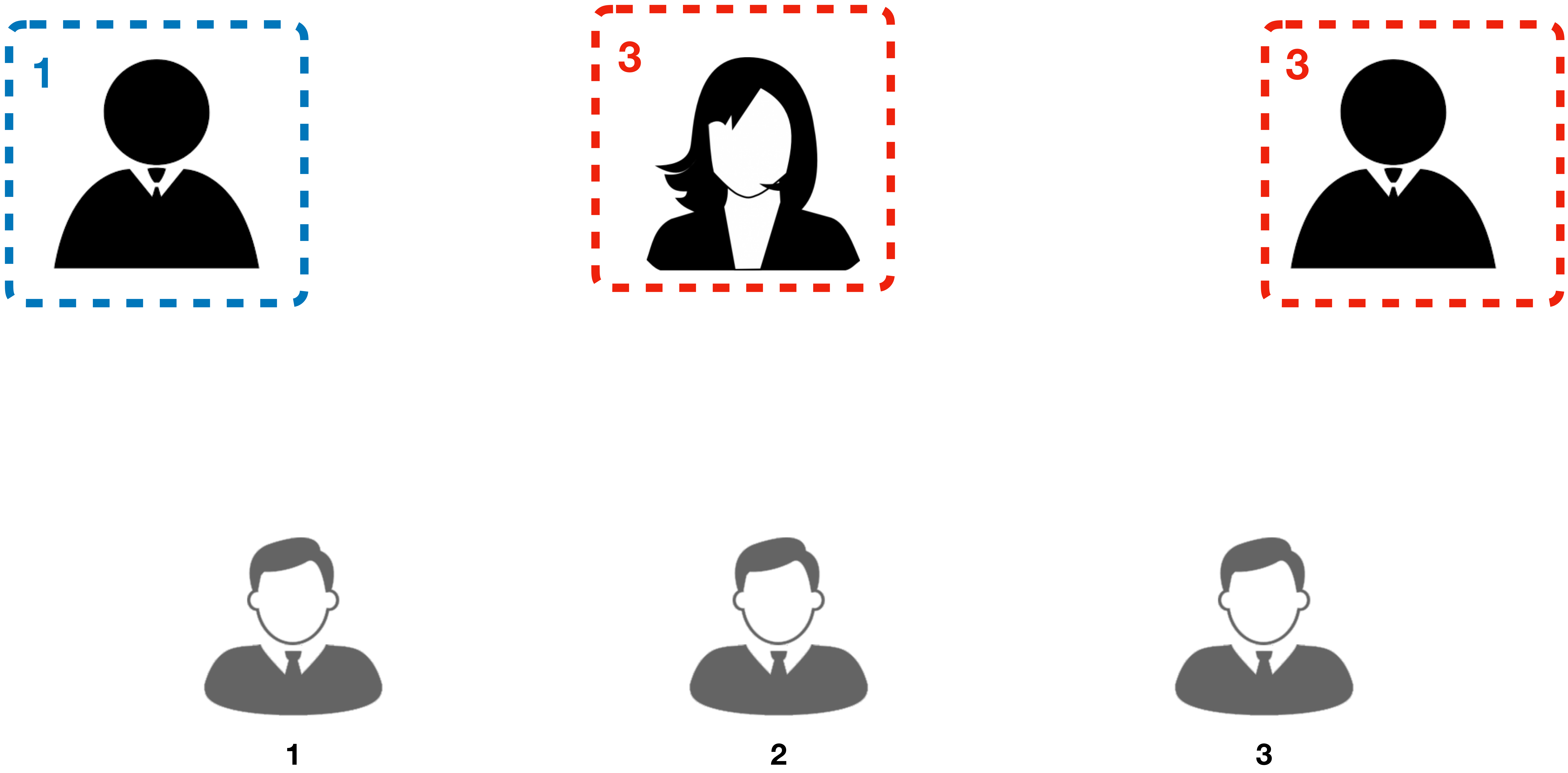
# Phase 1



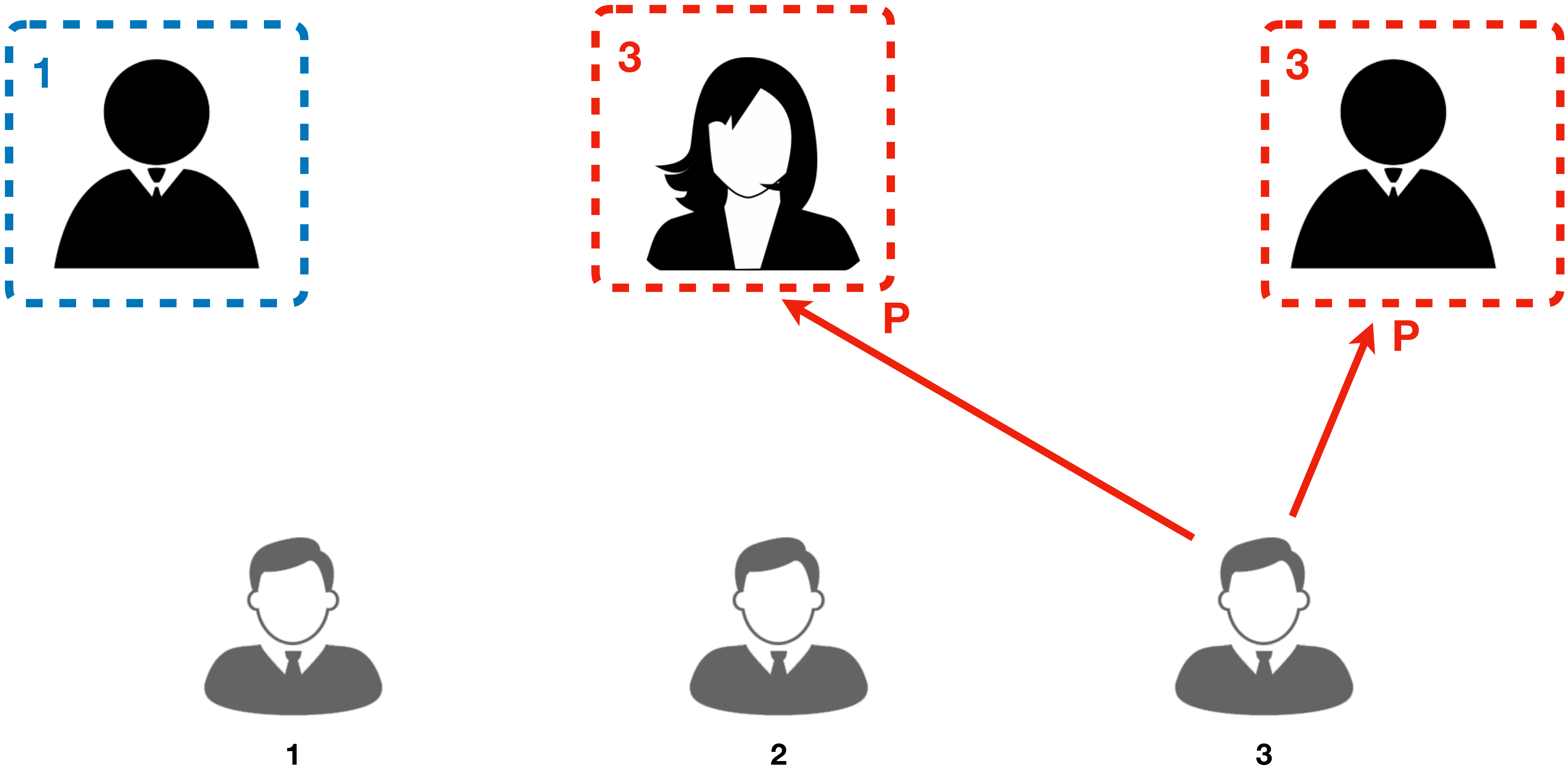
# Phase 1



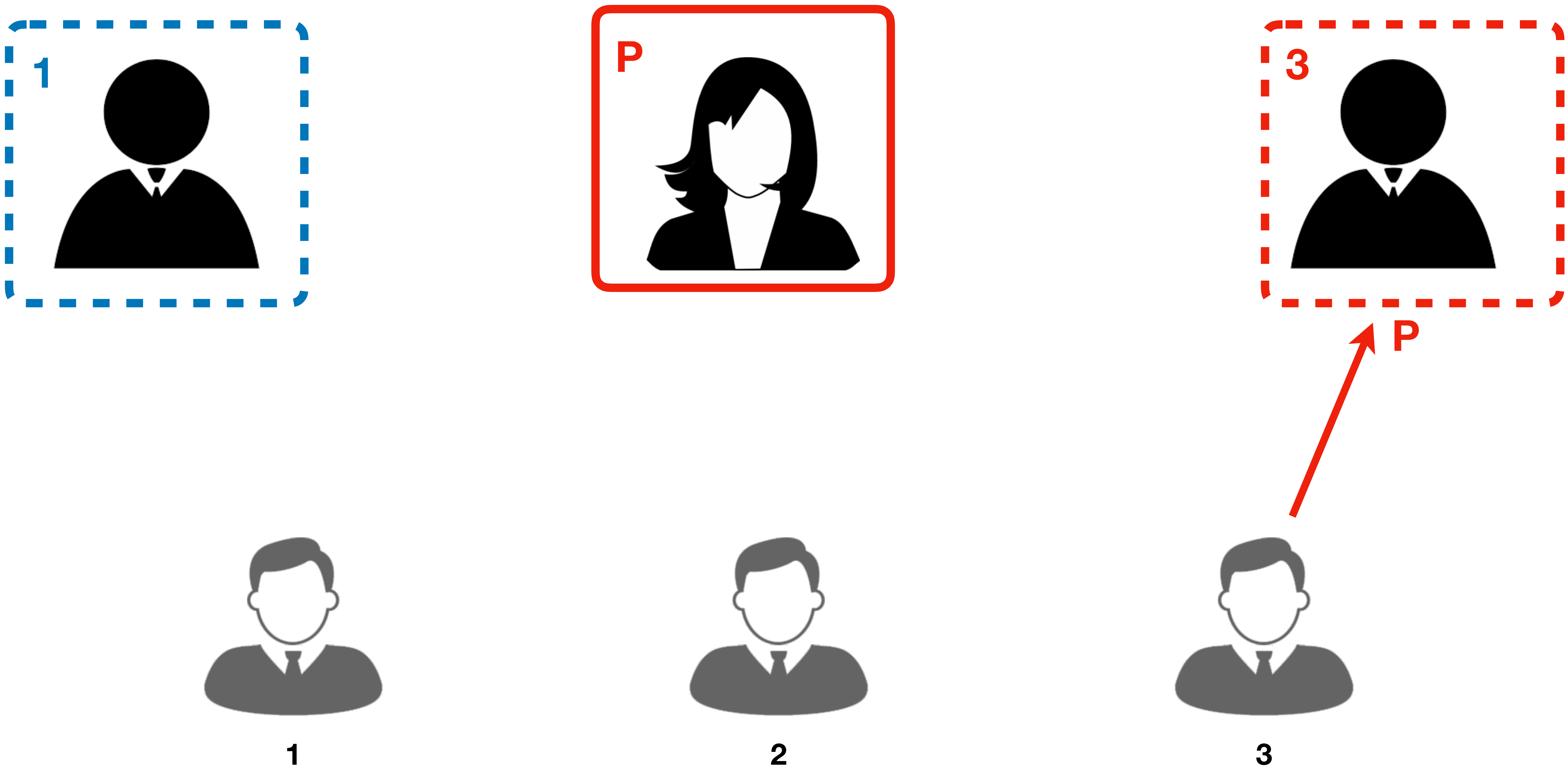
# Phase 1



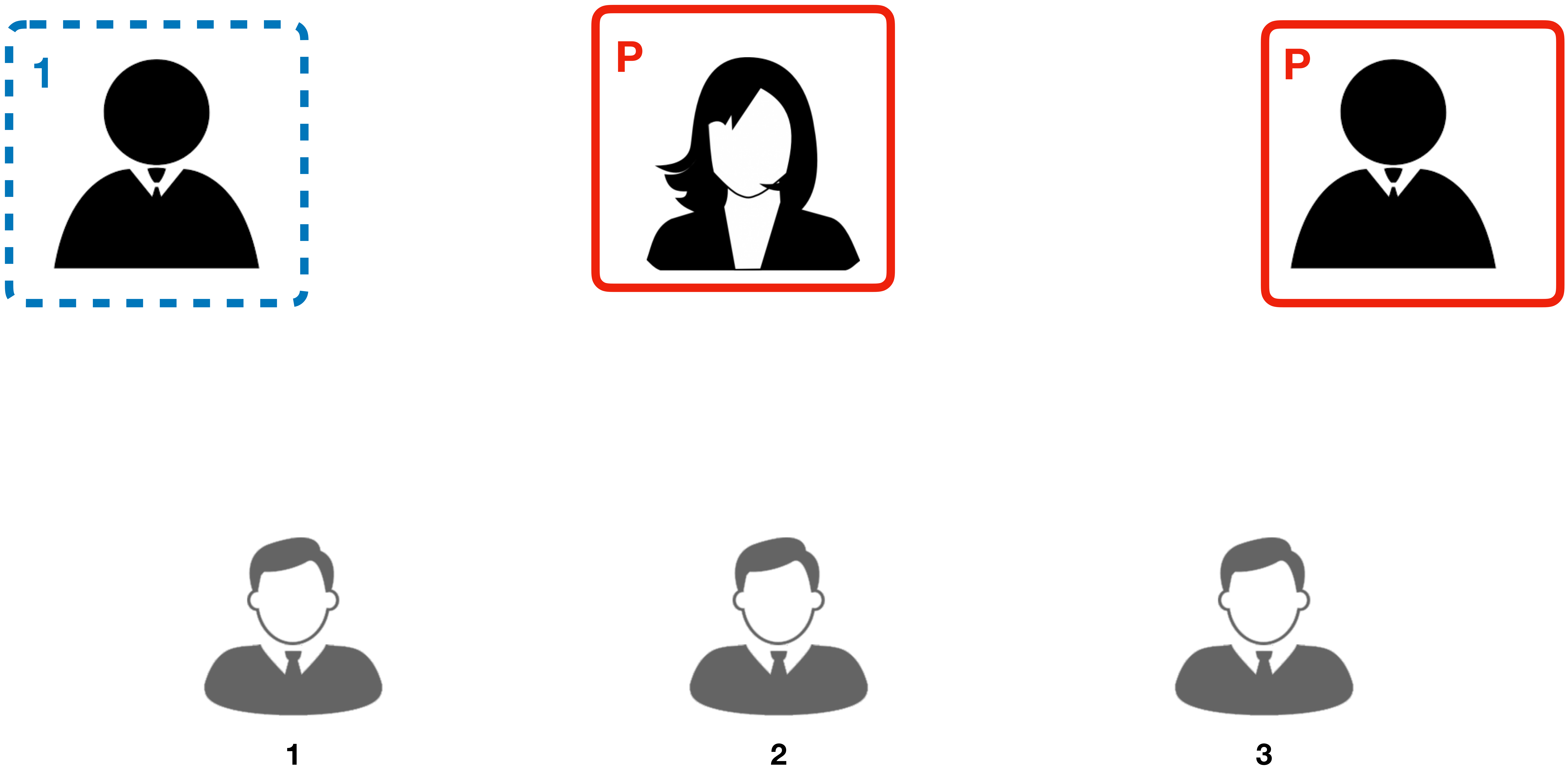
## Phase 2



# Phase 2

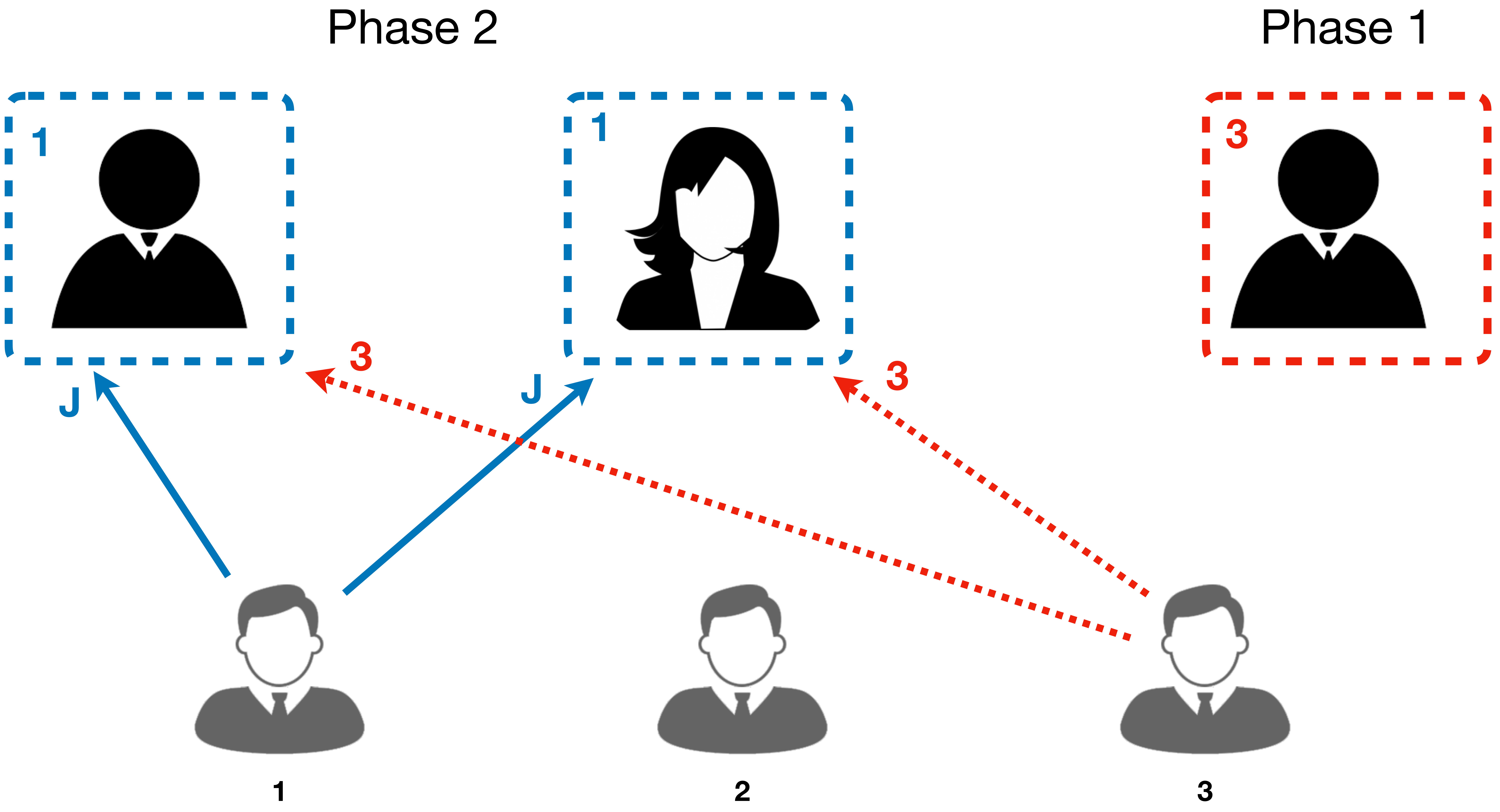


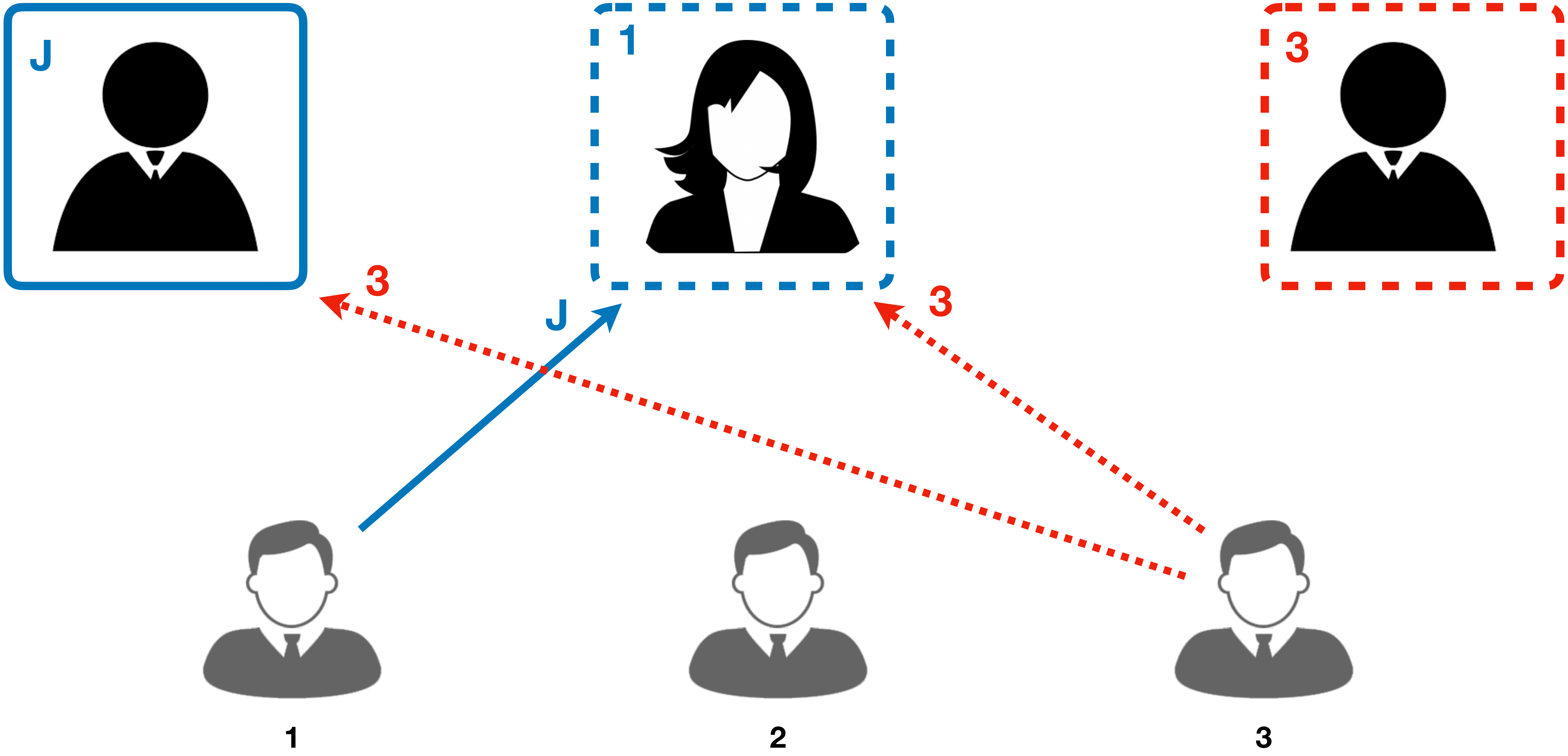
# Phase 2

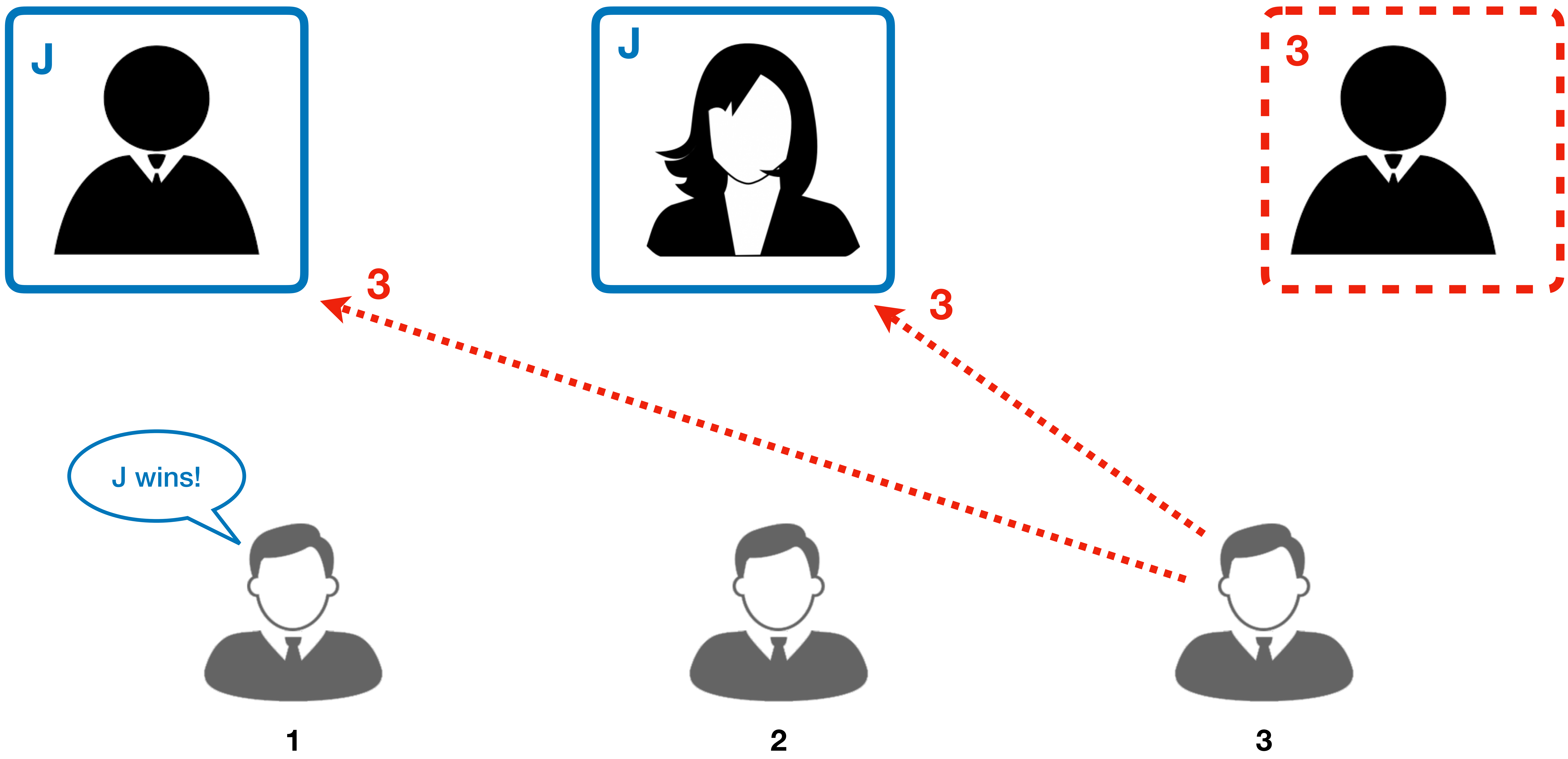


# Problem 3

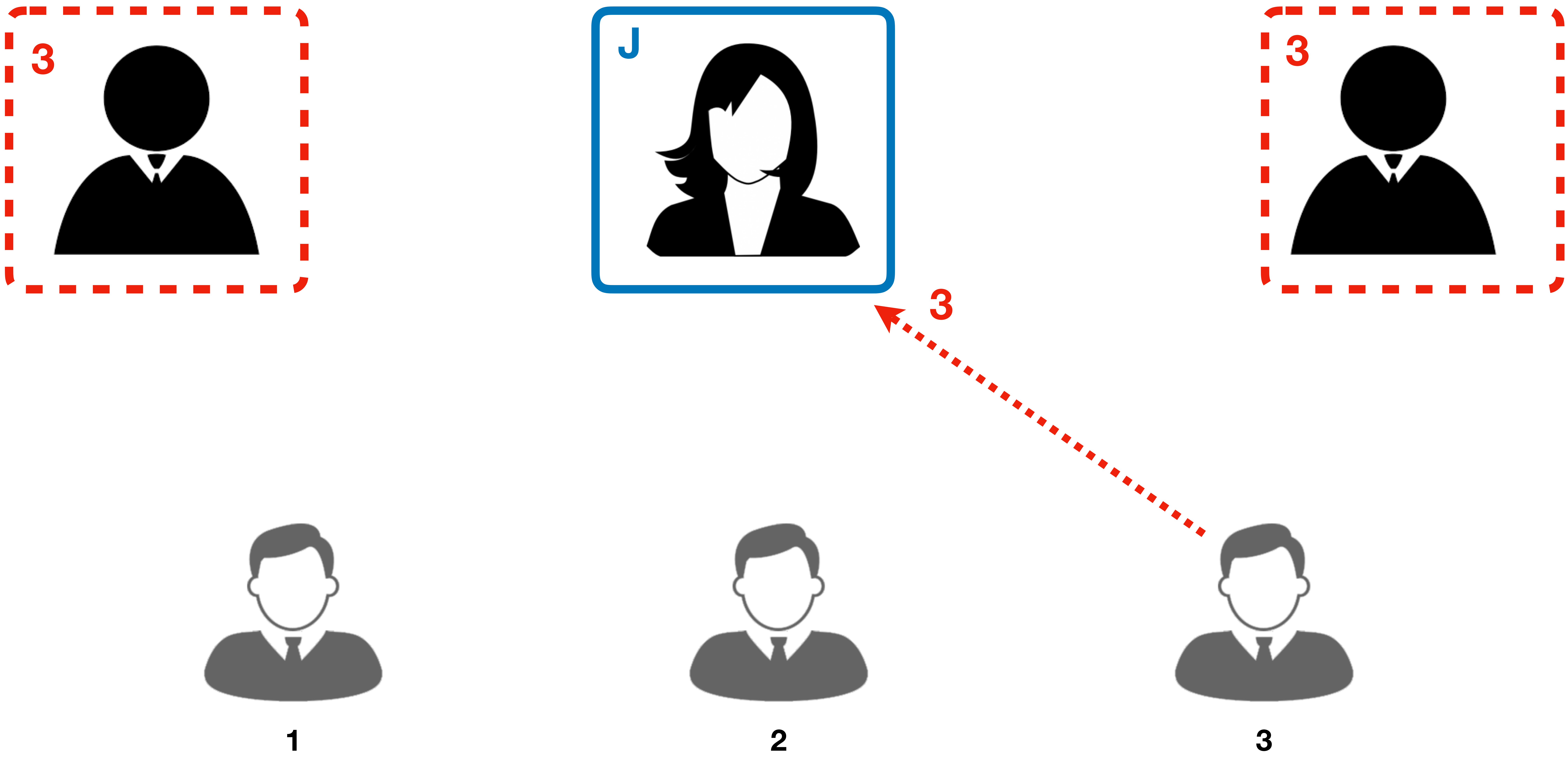
Because of asynchrony, low-priority Phase 2 can be *interrupted* by a high-priority Phase 1

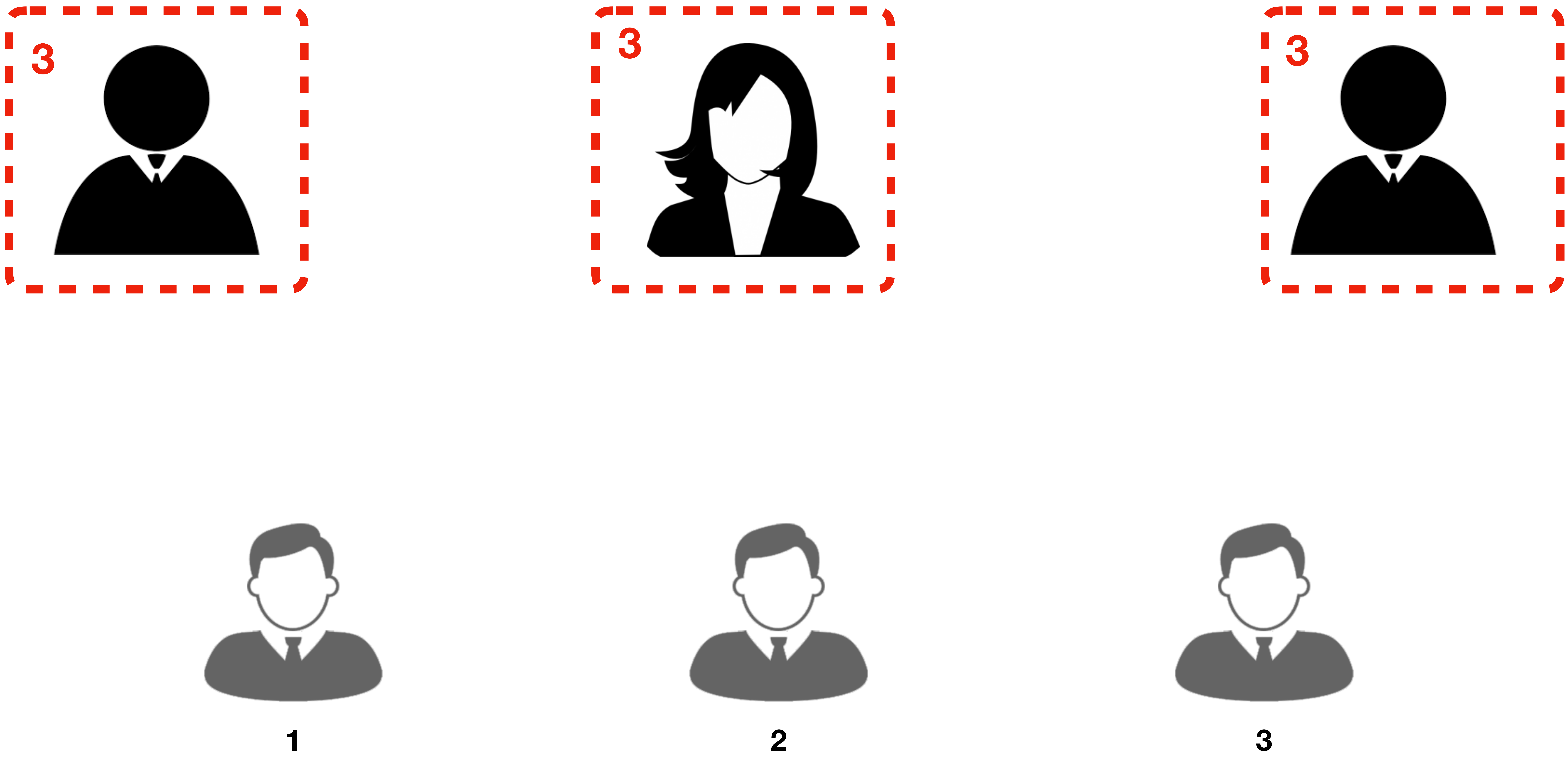

Phase 2
Phase 1
1
1
3
J
J
3
3
1
2
3

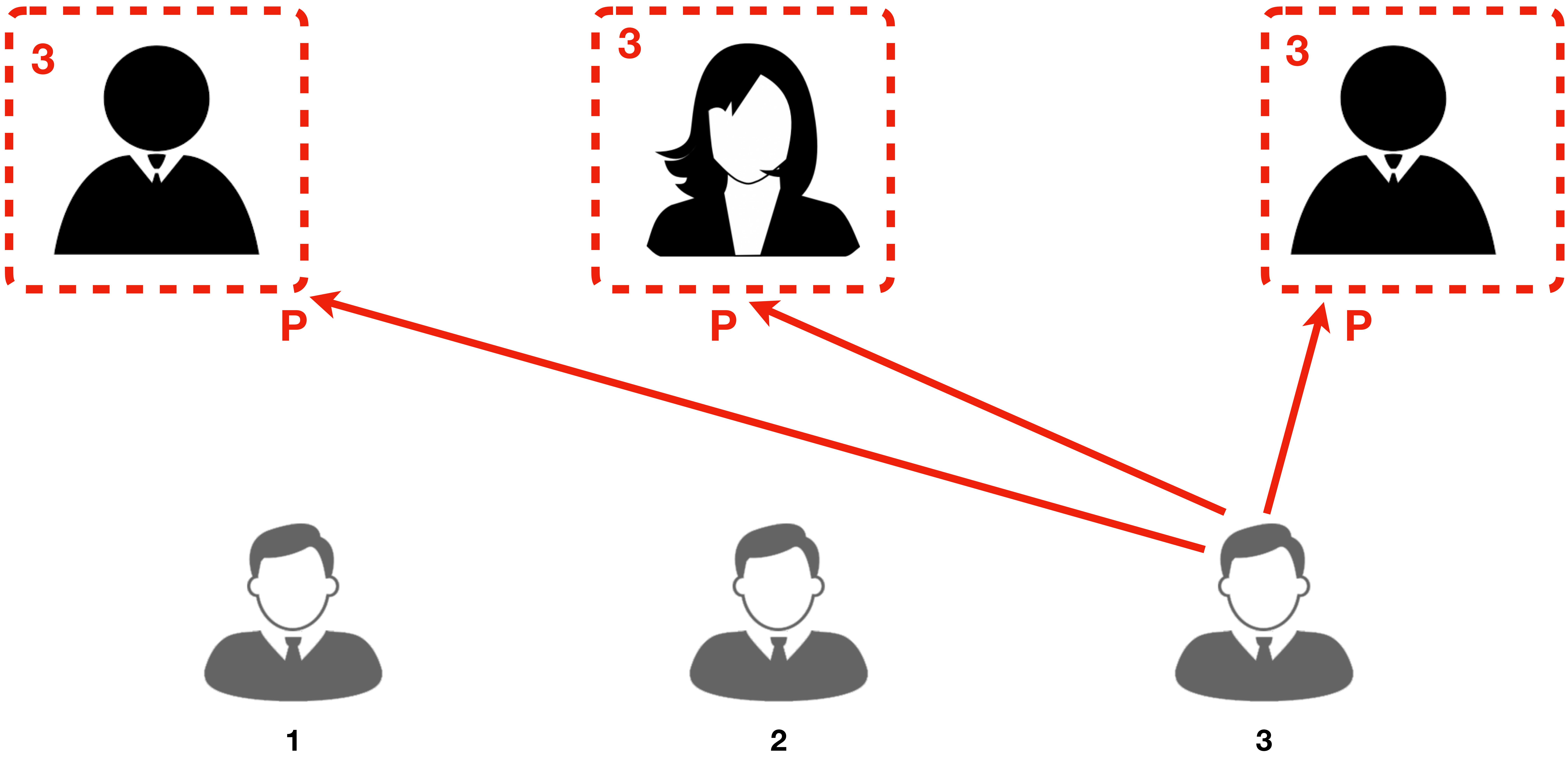

J
1
3
3
J
3
1
2
3



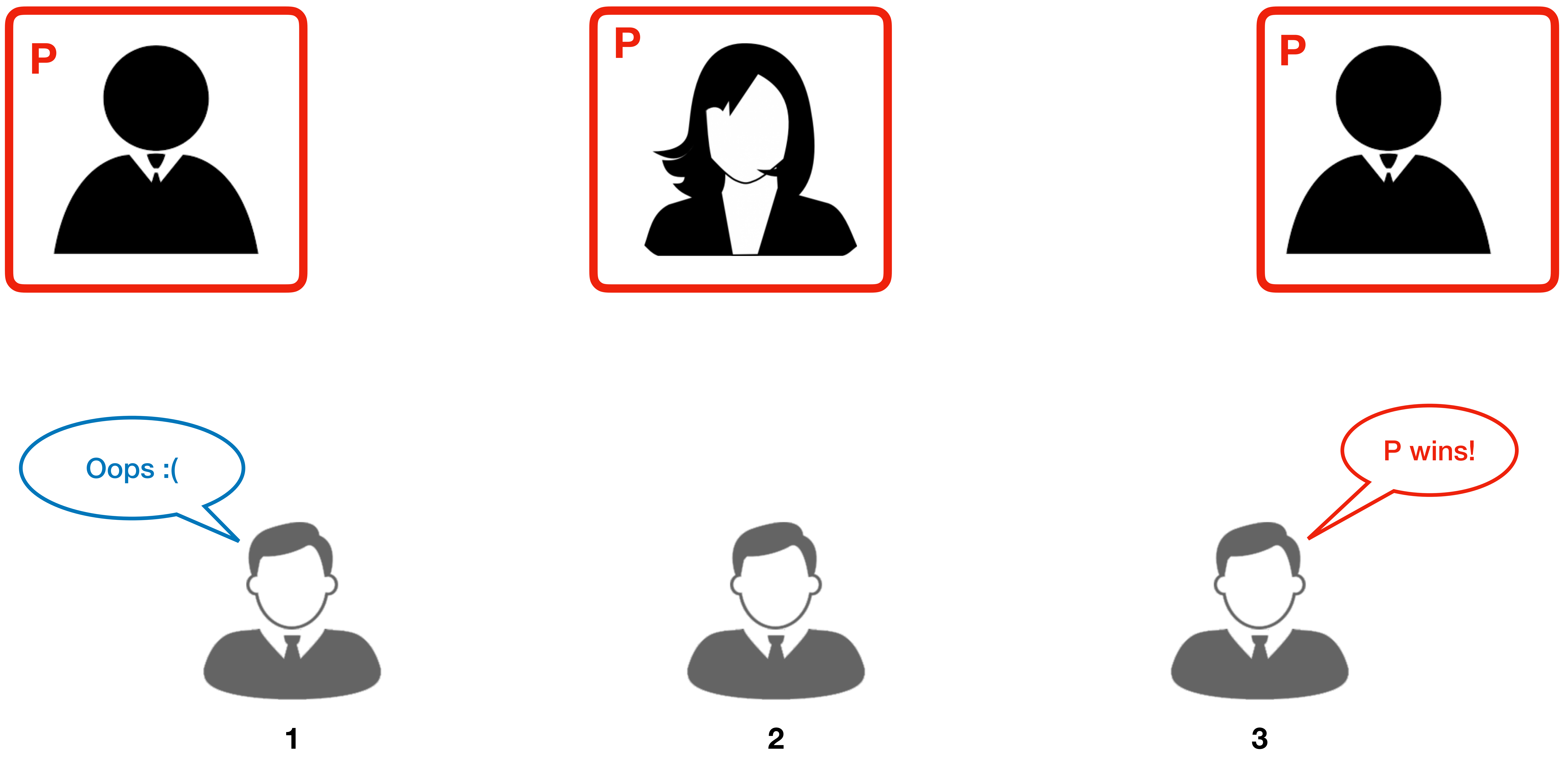
J
J
3
3
3
J wins!
1
2
3


3
J
3
3
1
2
3


3
3
3
1
2
3



3
3
3
P
P
P
1
2
3


P
P
P
Oops :(
P wins!
1
2
3

# Problem 3

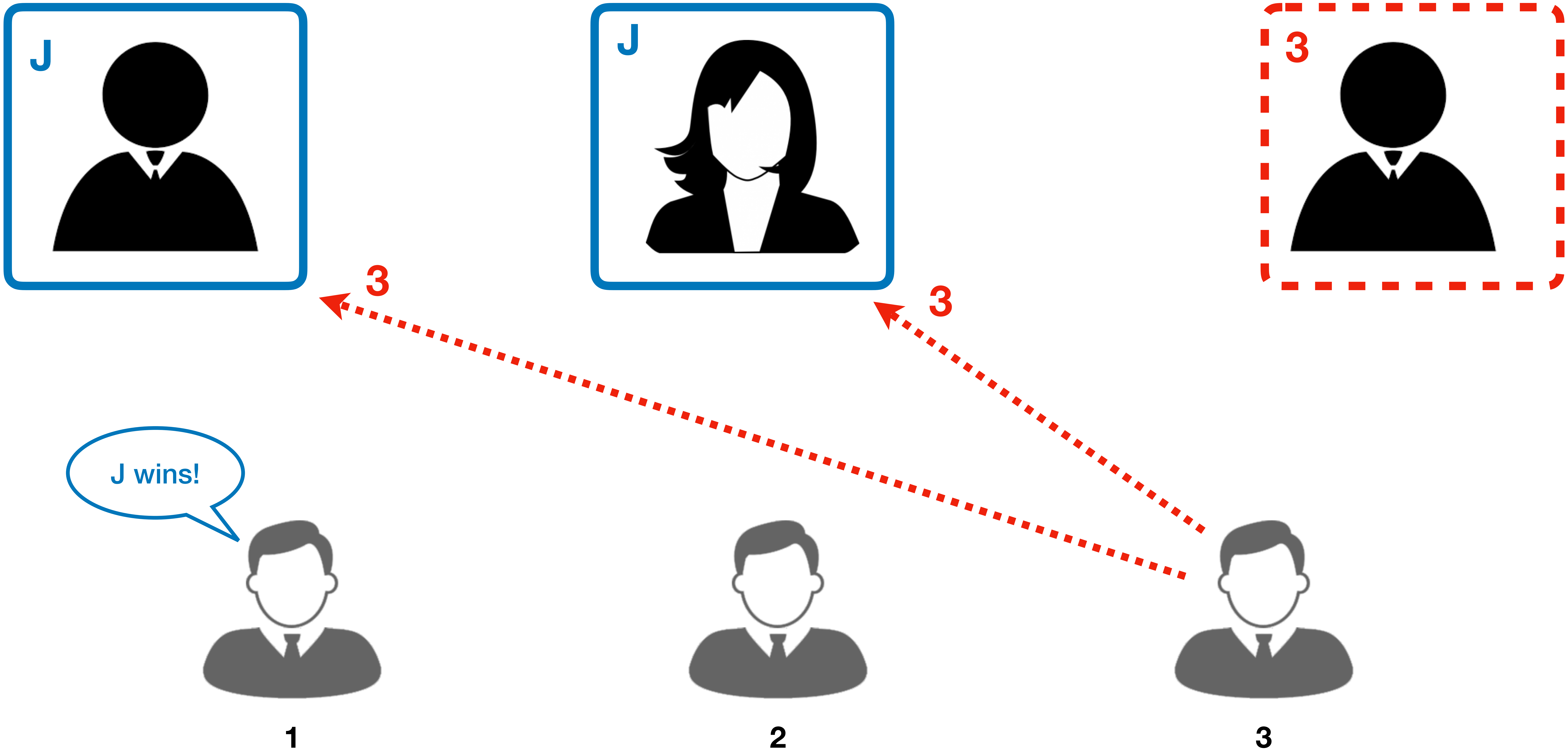
How to ensure irrevocability of consensus

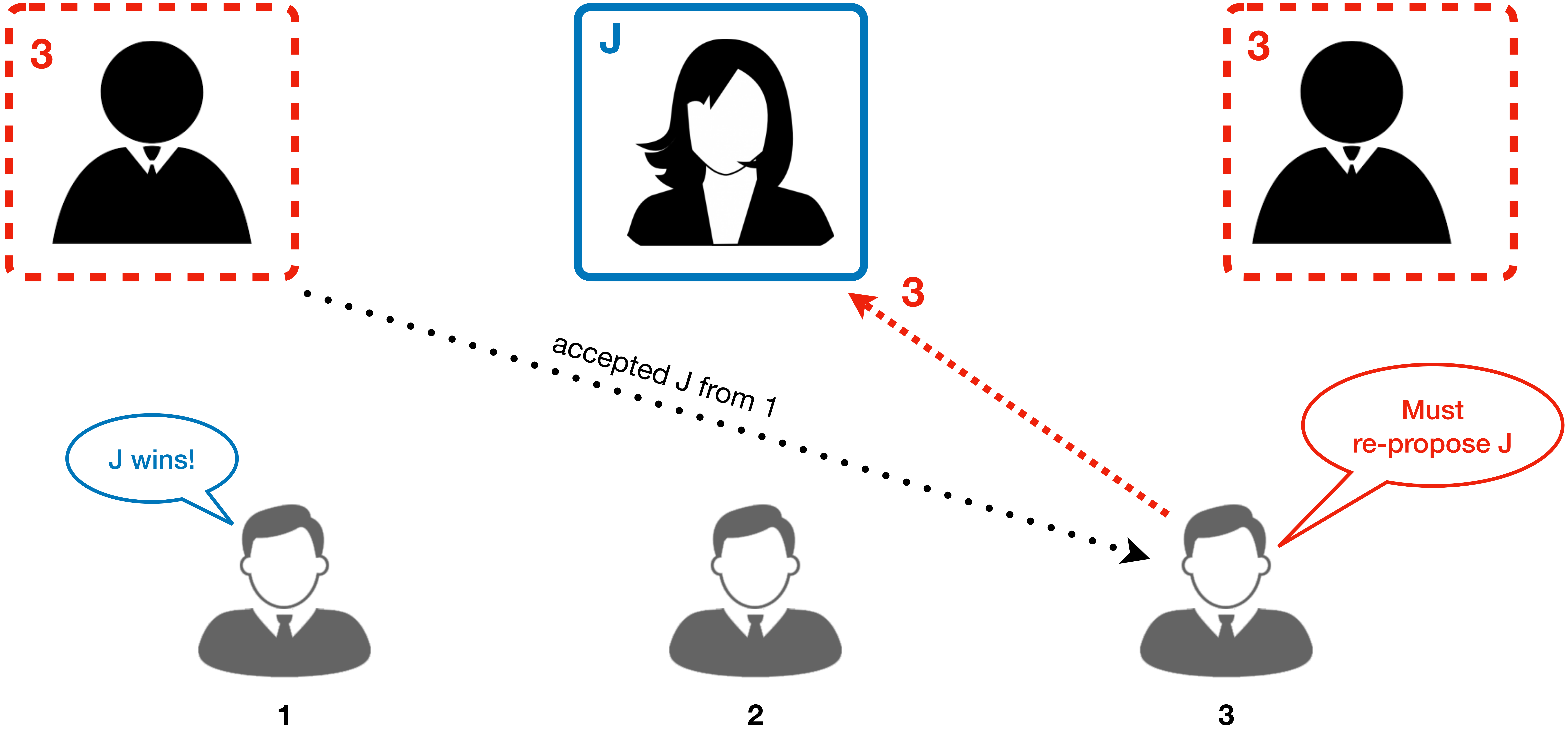
in the presence of *priorities* and *asynchrony*?

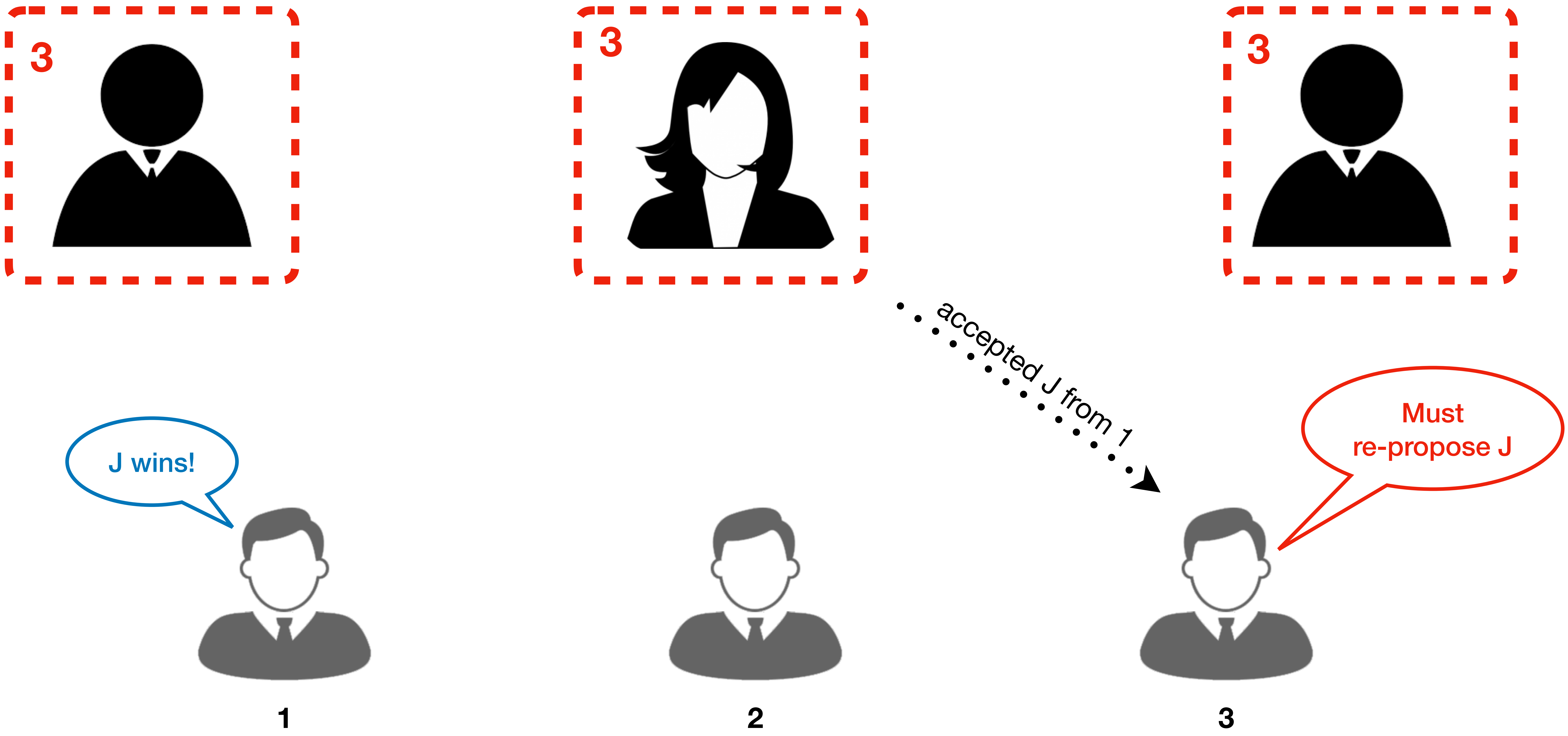
# Key Idea 4

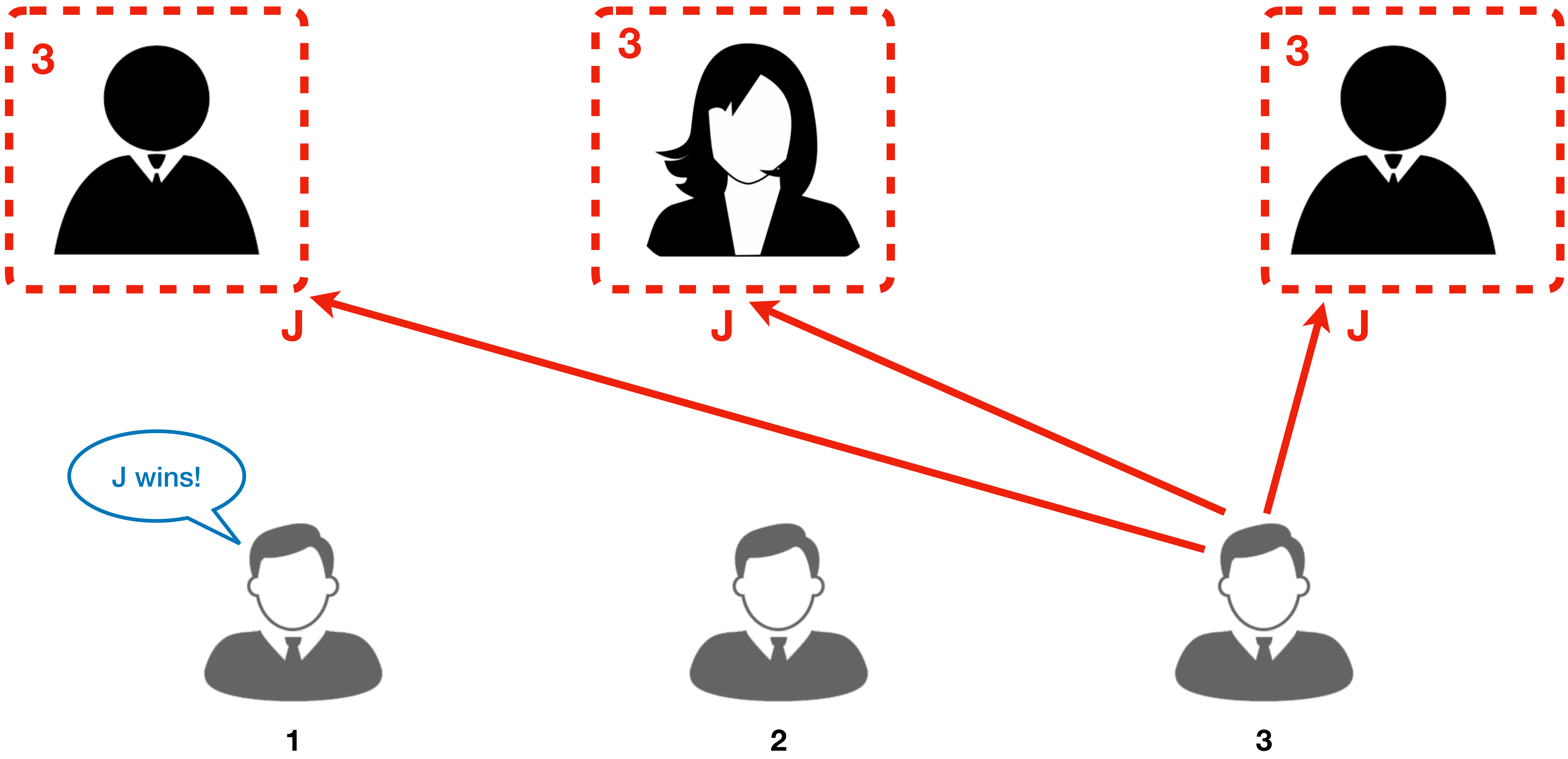
- Cooperation between Proposers and Acceptors:
  - Acceptors, when agreeing to support a proposer, *must* “tell” what was the *highest-ballot value* they have accepted;
  - *Higher-ballot* proposers *re-propose* already (partially) accepted values from the *lower-ballot* proposers, who secured the quorum before.

- This way, a proposer “knows" that, once it secured its quorum, either
  - its own proposal, or some higher-ballot one will be accepted
  - if its proposal got accepted, it will not be revoked
    (thanks to quorum intersection)

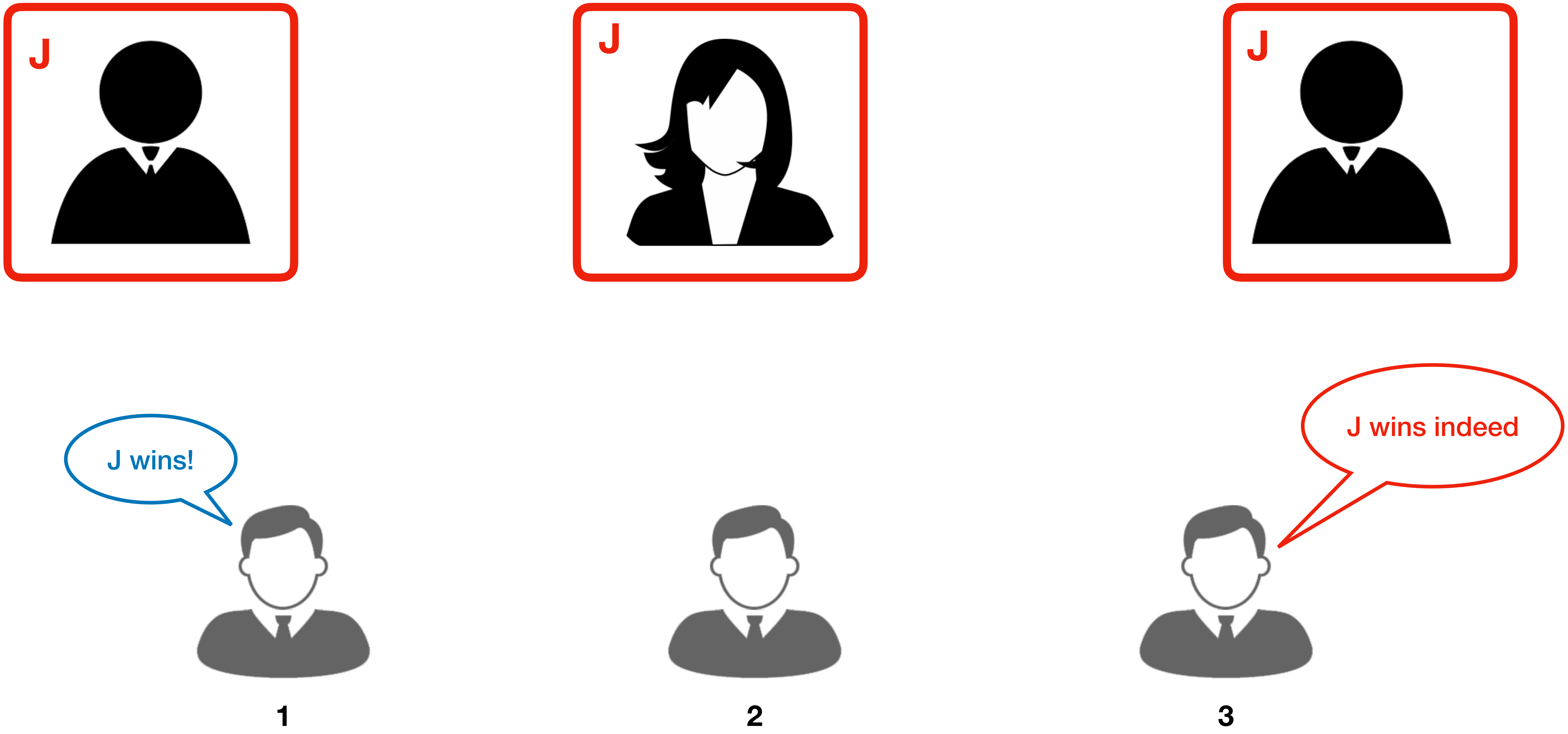

J
J
3
3
3
J wins!
1
2
3


3
J
3
3
accepted J from 1
J wins!
Must
re-propose J
1
2
3


3
3
3
accepted J from 1
J wins!
Must
re-propose J
1
2
3



3
3
3
J
J
J
J wins!
1
2
3


J
J
J
J wins!
J wins indeed
1
2
3

# Two-Phase Ballot-based Consensus

- Proposers suggest values, acceptors decide upon acceptance;
- Each proposal goes in two rounds:
  - Phase 1: securing a quorum of acceptors for a proposal
  - Phase 2: sending out the proposal
- Acceptors agree only to support ballots higher than what they've seen;
- They inform proposers of previously accepted values,
  which those then re-propose.

# The Algorithm in a Nutshell

Phase 1

**Proposer**

- Send my ballot **b** to all acceptors
- Wait for response of *at least* n/2 + 1 acceptors

**Acceptor**

- Upon receiving a **ballot b**
  - if it's the first one, remember it and send "ok" back.
  - if it's higher than **b'** we supported before, send back a previously accepted **(b', v')**, and remember **b** as what's currently supported.

Phase 2

**Proposer**

- When heard back from n/2 + 1 acceptors, send them back (**b**, **w**), where
  - **b** is my ballot
  - **w** is the value from the acceptors with the highest ballot, or *my own* value.

**Acceptor**

- Accept incoming value **w** if it comes with a ballot **b**, which we currently support; ignore otherwise.

# Learning an Accepted Value

- Send request to all acceptors;
- If at least $n/2 + 1$ acceptors respond back with the same value **v**, this is an accepted value.
- Correctness of this reasoning follows from *irrevocability*.

# Paxos

- A practical fault-tolerant distributed consensus algorithm;
- Invented in 1990, published in 1998;
- Nowadays used everywhere: Google (Bigtable, Chubby), IBM, Microsoft;
- You have just seen it explained.

# History of Paxos

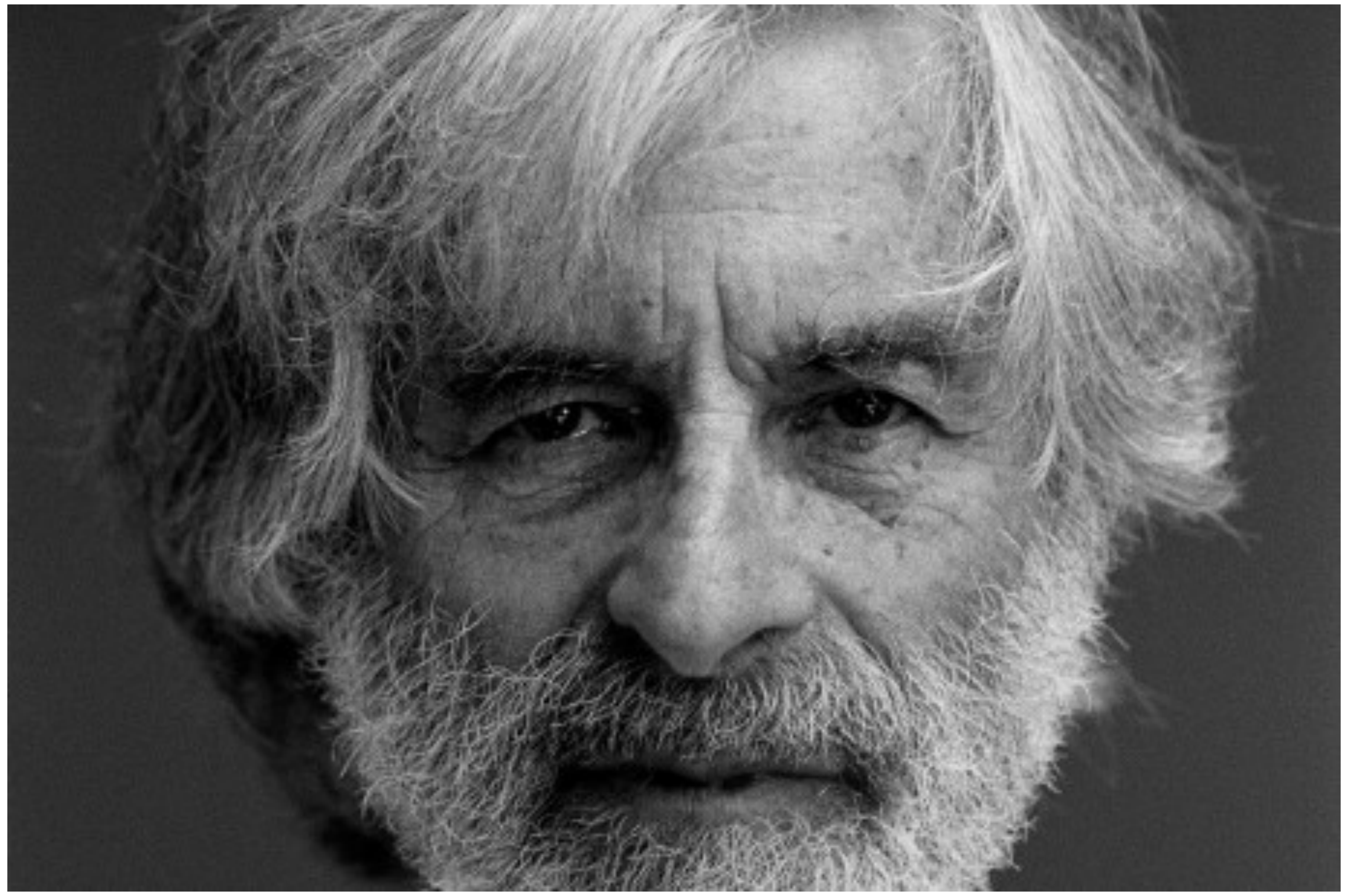
1990: Paxos first described

1998: Paxos paper published

2005: First practical deployments

2010: Widespread use!

2014: Lamport gets Turing Award



**Leslie Lamport**
(also known for LaTeX, Vector clocks, TLA)
Turing Award winner 2014

# History of Paxos

1990: Paxos first described

1998: Paxos paper published

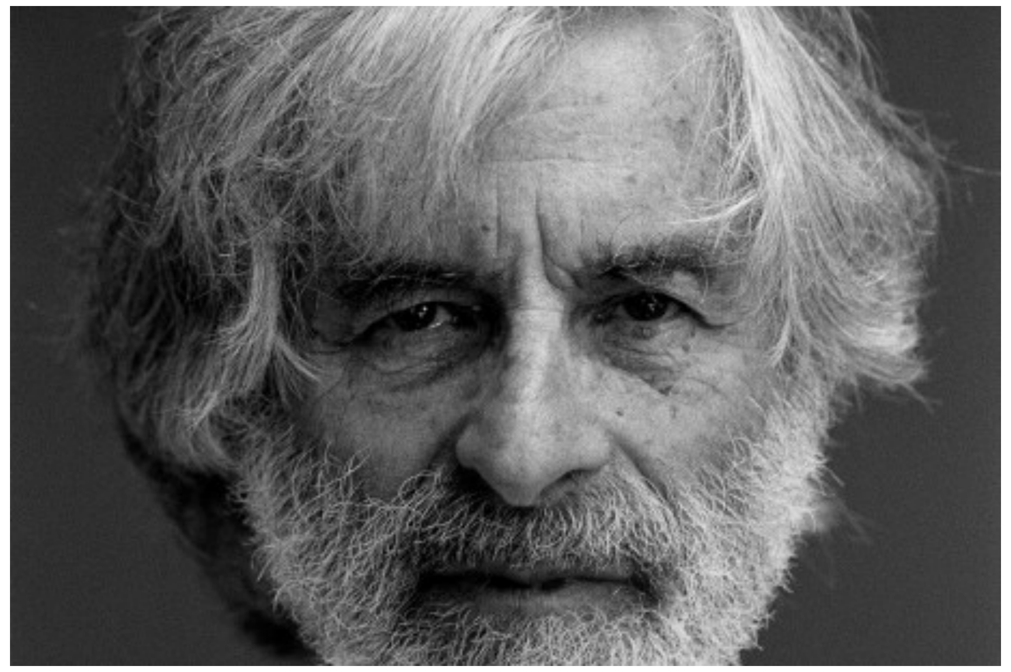
2005: First practical dep

2010: Widespread use!

2014: Lamport gets Turi

Recent archaeological discoveries on the island of Paxos reveal that the parliament functioned despite the peripatetic propensity of its part-time legislators.

The legislators maintained consistent copies of the parliamentary record, despite their frequent forays from the chamber and the forgetfulness of their messengers



Leslie Lamport
(also known for LaTeX, Vector clocks, TLA)
Turing Award winner 2014

# History of Paxos

1990: Paxos first described

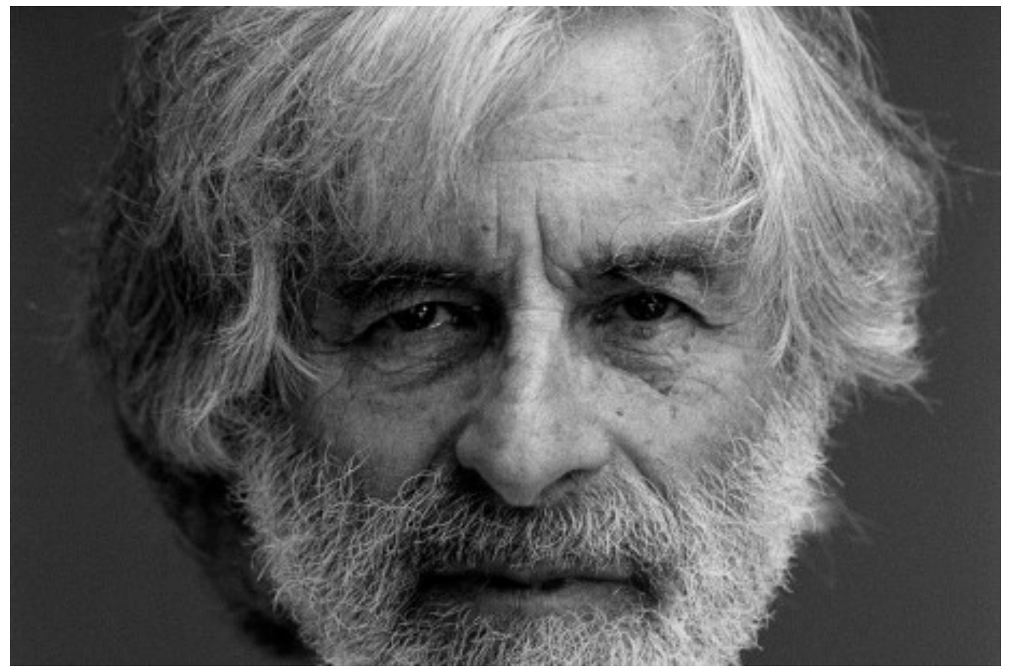
1998: Paxos paper published

2005: Fi

2010: W

2014: La

- The ABCDs of Paxos [2001]
- Paxos Made Simple [2001]
- Paxos Made Practical [2007]
- Paxos Made Live [2007]
- Paxos Made Moderately Complex [2011]
- Paxos Consensus, Deconstructed and Abstracted [2018]



Leslie Lamport
(also known for LaTeX, Vector clocks, TLA)
Turing Award winner 2014

# Multi-Paxos

- Presented in the original Lamport's 1998 paper.
- Uses the described idea for a *sequence* of "slots" (think *transactions*).
- Includes *reconfiguration* (changing set of acceptors on the fly).
- Naive implementation: run Simple Paxos for each slot.
    - Better approach — secure a quorum for several slots.

# Exploring the Paxos Zoo with Network Combinators

- A framework for combining different optimisations of Simple/Multi Paxos
- Written in Scala/Akka, available at https://github.com/certichain/network-transformations
- Accompanying paper:
  *Paxos Consensus, Deconstructed and Abstracted* by García-Pérez *et al*, 2018.

```
def setupAndRunPaxos[A](slotValueMap: Map[Int, List[A]], factory: PaxosFactory[A]) {
  val acceptorNum = 7
  val learnerNum = 3
  val proposerNum = 5

  val instance = factory.createPaxosInstance(system, proposerNum, acceptorNum, learnerNum)

  proposeValuesForSlots(slotValueMap, instance, factory)

  Thread.sleep(400) // Wait for some time
  learnAcceptedValues(slotValueMap, instance, factory)
}
```

# Alternative Consensus Protocols

- View-Stamped Replication
  by Brian M. Oki and Barbara Liskov, 1989
- Raft
  by Diego Ongaro and John K. Ousterhout, 2014

# Formal Verification of Consensus

- Initially only the *model* of the protocol was verified:
  - P. Kellomäki, 2004, Simple Paxos in **PVS**
  - M. Jaskelioff and S. Merz, 2005, Disk Paxos in **Isabelle/HOL**
  - O. Padon et al. 2017, Simple/Multi-Paxos in **Ivy**
- Verified runnable implementations came later:
  - V. Rahli et al., 2015, Multi-Paxos in **EventML**
  - C. Hawblitzel et al., 2015, Multi-Paxos in **Dafny**
  - J. Wilcox et al., 2015, Raft in **Coq**
  - C. Dragoi et al., 2016, (Synchronous) Simple Paxos in **PSync**
  - A. Pillai, 2018, Simple Paxos **Coq** (incomplete)

# To Take Away

- Fault-Tolerant Consensus Protocols are a *critical component* of modern *distributed systems* and *applications*
- Consensus properties are *uniformity*, *non-triviality*, and *irrevocability*
- The key ideas of Lamport's Paxos protocol are:
  - Majority *quorums* (avoiding split brain and enabling fault-tolerance);
  - *Two-phase* structure (secure-commit);
  - Dichotomy and cooperation between *proposers and acceptors*.

To be continued…

# Bibliography


- L. Lamport. *The part-time parliament*. ACM Trans. Comput. Syst., 16(2):133–169, 1998.
- L. Lamport. *Paxos made simple*. SIGACT News, 32, 2001.
- T.D. Chandra et al. *Paxos made live: an engineering perspective*. PODC 2007
- B. W. Lampson, *The ABCD's of Paxos*. PODC 2001
- P. Kellomäki. *An Annotated Specification of the Consensus Protocol of Paxos Using Superposition in PVS*. 2004
- C. Dragoi et al. *PSync: a partially synchronous language for fault-tolerant distributed algorithms*. In POPL, 2016.
- M. Jaskelioff and S. Merz. *Proving the correctness of disk Paxos*. Archive of Formal Proofs, 2005.
- C. Hawblitzel et al. *IronFleet: proving practical distributed systems correct*. In SOSP 2015.
- D. Ongaro and J. K. Ousterhout. *In search of an understandable consensus algorithm*. USENIX Annual Technical Conference, 2014
- B.M. Oki and B. Liskov, *Viewstamped Replication: A General Primary Copy*. PODC 1988
- O. Padon, et al. *Paxos made EPR: decidable reasoning about distributed protocols.* PACMPL, 1(OOPSLA):108:1–108:31, 2017.
- V. Rahli, et al. *Formal specification, verification, and implementation of fault-tolerant systems using EventML*. In AVOCS. EASST, 2015.
- A. Pillai, *Mechanised Verification of Paxos-like Consensus Protocols*, BSc Thesis, 2018
- R. van Renesse and D. Altinbuken. Paxos Made Moderately Complex. ACM Comput. Surv., 47(3):42:1–42:36, 2015.
- J.R. Wilcox et al., *Verdi: a framework for implementing and formally verifying distributed systems*, PLDI 2015
- Á. García-Pérez et al., *Paxos Consensus, Deconstructed and Abstracted*, ESOP 2018