Distributed Computing II

Jörn W. Janneck

Computer Science Dept.

Lund University

Introduction to Cloud Computing

homework solution

One prisoner is chosen to be the counter.
He keeps a count, starting at 0.

If the counter goes into the room:
- If the swith is up:
  - Move it down.
  - Increment the count.
  - When the count reaches 99 (100 - himself), tell the warden everyone has been in the room.
- If the switch is down:
  - Leave it.

If any other prisoner goes into the room:
- If the switch is down and he has not previously moved it up:
  - Move it up.
- Otherwise leave it.

(Emma Fitzgerald)

consensus & data structures

The Part-Time Parliament

LESLIE LAMPORT
Digital Equipment Corporation

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. The Paxon parliament's protocol provides a new way of implementing the state-machine approach to the design of distributed systems.

Categories and Subject Descriptors: C2.4 [**Computer-Communications Networks**]: Distributed Systems—*Network operating systems*; D4.5 [**Operating Systems**]: Reliability—*Fault-tolerance*; J.1 [**Administrative Data Processing**]: Government

General Terms: Design, Reliability

Additional Key Words and Phrases: State machines, three-phase commit, voting

This submission was recently discovered behind a filing cabinet in the *TOCS* editorial office. Despite its age, the editor-in-chief felt that it was worth publishing. Because the author is currently doing field work in the Greek isles and cannot be reached, I was asked to prepare it for publication.

The author appears to be an archeologist with only a passing interest in computer science. This is unfortunate; even though the obscure ancient Paxon civilization he describes is of little interest to most computer scientists, its legislative system is an excellent model for how to implement a distributed computer system in an asynchronous environment. Indeed, some of the refinements the Paxons made to their protocol appear to be unknown in the systems literature.

The author does give a brief discussion of the Paxon Parliament's relevance to distributed computing in Section 4. Computer scientists will probably want to read that section first. Even before that, they might want to read the explanation of the algorithm for computer scientists by Lampson [1996]. The algorithm is also described more formally by De Prisco et al. [1997]. I have added further comments on the relation between the ancient protocols and more recent work at the end of Section 4.

Keith Marzullo
University of California, San Diego

Authors' address: Systems Research Center, Digital Equipment Corporation, 130 Lytton Avenue, Palo Alto, CA 94301.

Chord: A Scalable Peer-to-peer Lookup Protocol for Internet Applications

Ion Stoica[†], Robert Morris[‡], David Liben-Nowell[‡], David R. Karger[‡], M. Frans Kaashoek[‡], Frank Dabek[‡], Hari Balakrishnan[‡]

Abstract—
A fundamental problem that confronts peer-to-peer applications is the efficient location of the node that stores a desired data item. This paper presents *Chord*, a distributed lookup protocol that addresses this problem. Chord provides support for just one operation: given a key, it maps the key onto a node. Data location can be easily implemented on top of Chord by associating a key with each data item, and storing the key/data pair at the node to which the key maps. Chord adapts efficiently as nodes join and leave the system, and can answer queries even if the system is continuously changing. Results from theoretical analysis and simulations show that Chord is scalable: communication cost and the state maintained by each node scale logarithmically with the number of Chord nodes.

I. Introduction

Peer-to-peer systems and applications are distributed systems without any centralized control or hierarchical organization, in which each node runs software with equivalent functionality. A review of the features of recent peer-to-peer applications yields a long list: redundant storage, permanence, selection of nearby servers, anonymity, search, authentication, and hierarchical naming. Despite this rich set of features, the core operation in most peer-to-peer systems is efficient location of data items. The contribution of this paper is a scalable protocol for lookup in a dynamic peer-to-peer system with frequent node arrivals and departures.

The *Chord protocol* supports just one operation: given a key, it maps the key onto a node. Depending on the application using Chord, that node might be responsible for storing a value associated with the key. Chord uses consistent hashing [12] to assign keys to Chord nodes. Consistent hashing tends to balance load, since each node receives roughly the same number of keys, and requires relatively little movement of keys when nodes join and leave the system.

Previous work on consistent hashing assumes that each node is aware of most of the other nodes in the system, an approach that does not scale well to large numbers of nodes. In contrast, each Chord node needs "routing" information about only a few other nodes. Because the routing table is distributed, a Chord node communicates with other nodes in order to perform a lookup. In the steady state, in an N-node system, each node maintains information about only $O(\log N)$ other nodes, and resolves all lookups via $O(\log N)$ messages to other nodes. Chord maintains its routing information as nodes join and leave the system.

A Chord node requires information about $O(\log N)$ other nodes for *efficient* routing, but performance degrades gracefully when that information is out of date. This is important in practice because nodes will join and leave arbitrarily, and consistency of even $O(\log N)$ state may be hard to maintain. Only one piece of information per node need be correct in order for Chord to guarantee correct (though possibly slow) routing of queries; Chord has a simple algorithm for maintaining this information in a dynamic environment.

The contributions of this paper are the Chord algorithm, the proof of its correctness, and simulation results demonstrating the strength of the algorithm. We also report some initial results on how the Chord routing protocol can be extended to take into account the physical network topology. Readers interested in an application of Chord and how Chord behaves on a small Internet testbed are referred to Dabek *et al.* [9]. The results reported by Dabek *et al.* are consistent with the simulation results presented in this paper.

The rest of this paper is structured as follows. Section II compares Chord to related work. Section III presents the system model that motivates the Chord protocol. Section IV presents the Chord protocol and proves several of its properties. Section V presents simulations supporting our claims about Chord's performance. Finally, we summarize our contributions in Section VII.

II. Related Work

Three features that distinguish Chord from many other peer-to-peer lookup protocols are its simplicity, provable correctness, and provable performance.

To clarify comparisons with related work, we will assume in this section a Chord-based application that maps keys onto values. A value can be an address, a document, or an arbitrary data item. A Chord-based application would store and find each value at the node to which the value's key maps.

DNS provides a lookup service, with host names as keys and IP addresses (and other host information) as values. Chord could provide the same service by hashing each host name to a key [7]. Chord-based DNS would require no special servers, while ordinary DNS relies on a set of special root servers. DNS requires manual management of the routing information (NS records) that allows clients to navigate the name server hierarchy; Chord automatically maintains the correctness of the analogous routing information. DNS only works well when host names are structured to reflect administrative boundaries; Chord imposes no naming structure. DNS is specialized to the task of finding named hosts or services, while Chord can also be used to find

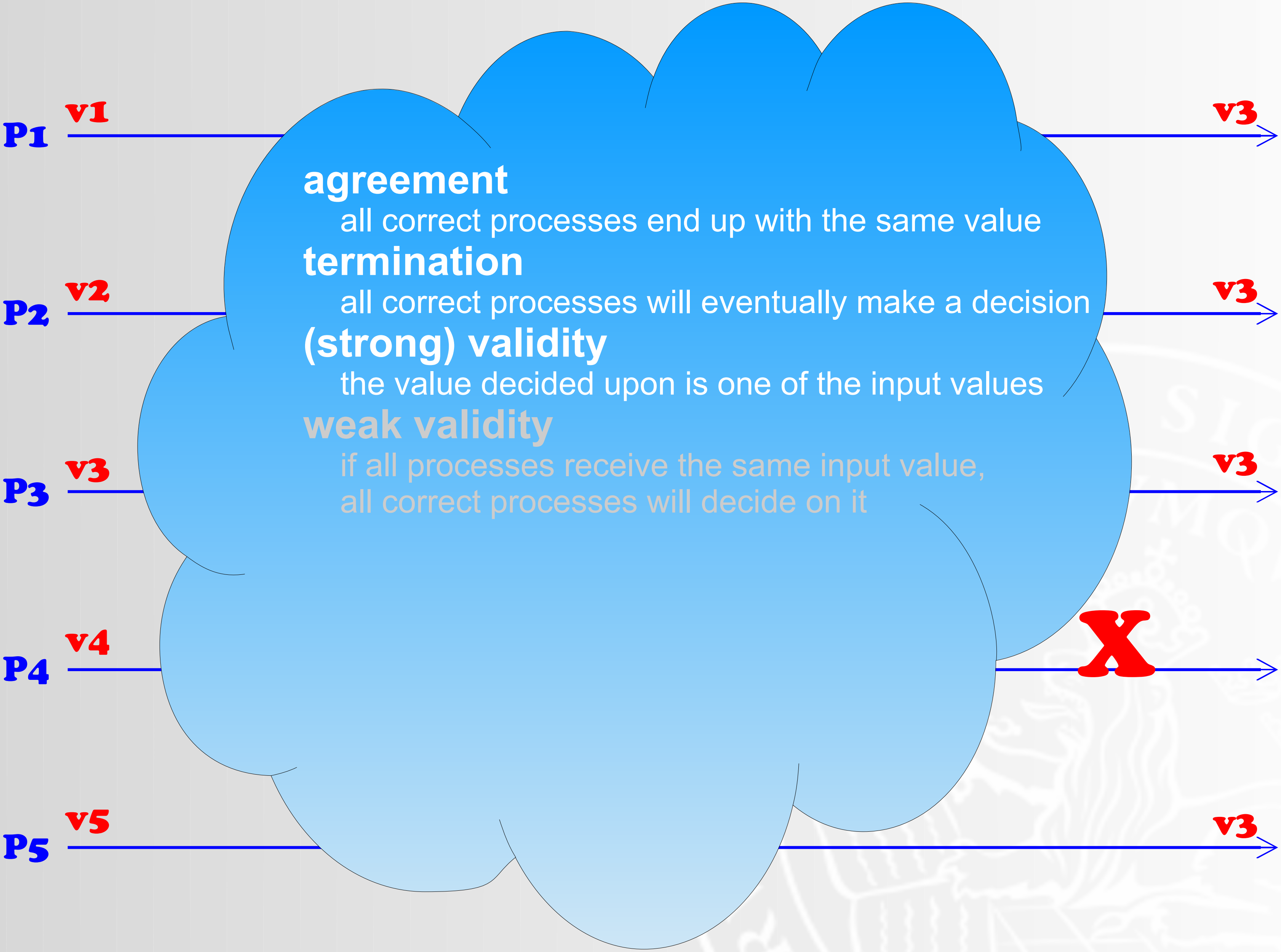
[†]University of California, Berkeley, istoica@cs.berkeley.edu
[‡]MIT Laboratory for Computer Science, {rtm, dln, karger, kaashoek, fdabek, hari}@lcs.mit.edu
Authors in reverse alphabetical order.

This research was sponsored by the Defense Advanced Research Projects Agency (DARPA) and the Space and Naval Warfare Systems Center, San Diego, under contract N66001-00-1-8933.

Victor: Paxos

Manfred: DHT

consensus



agreement

all correct processes end up with the same value

termination

all correct processes will eventually make a decision

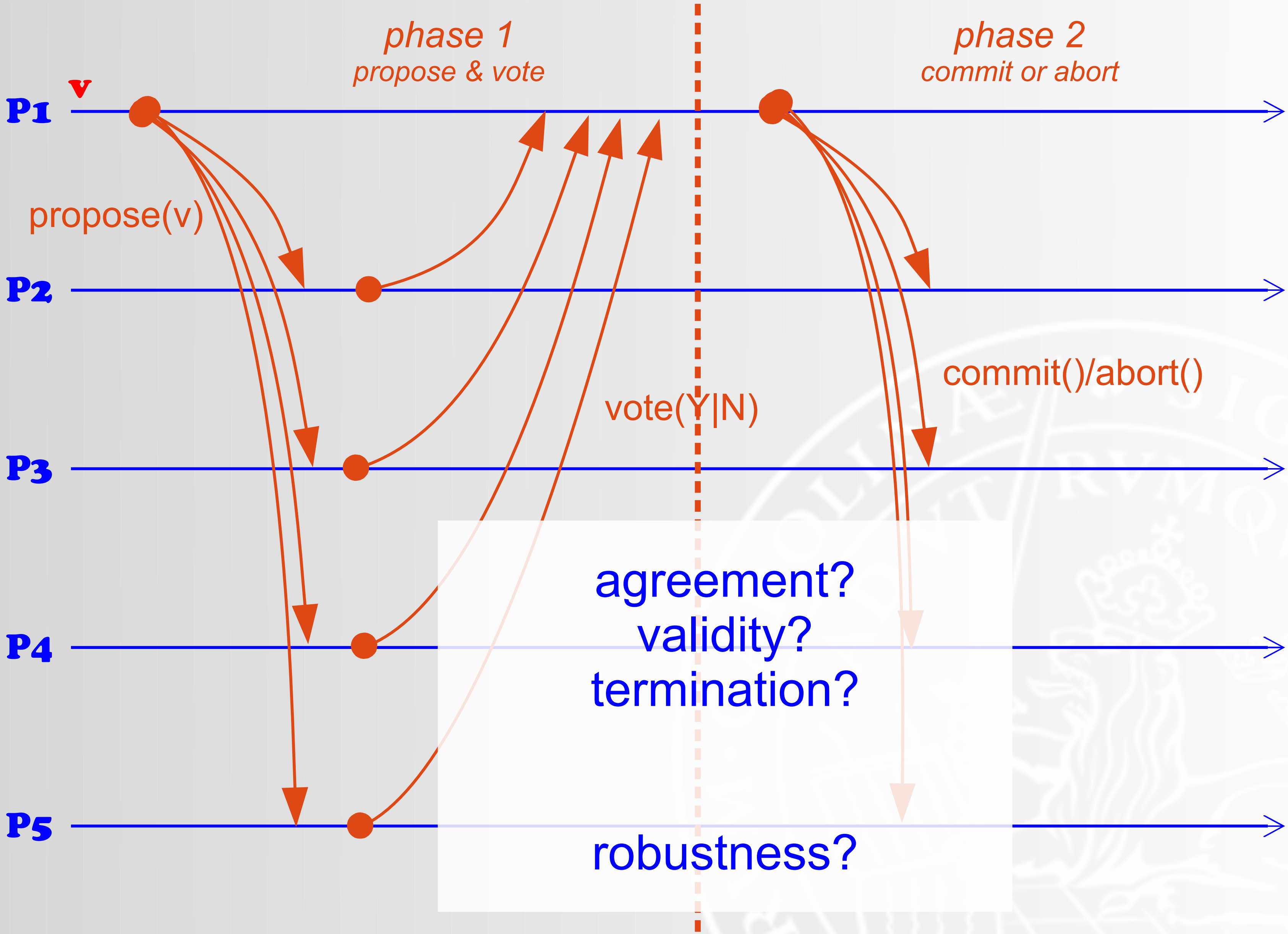
(strong) validity

the value decided upon is one of the input values

weak validity

if all processes receive the same input value,
all correct processes will decide on it

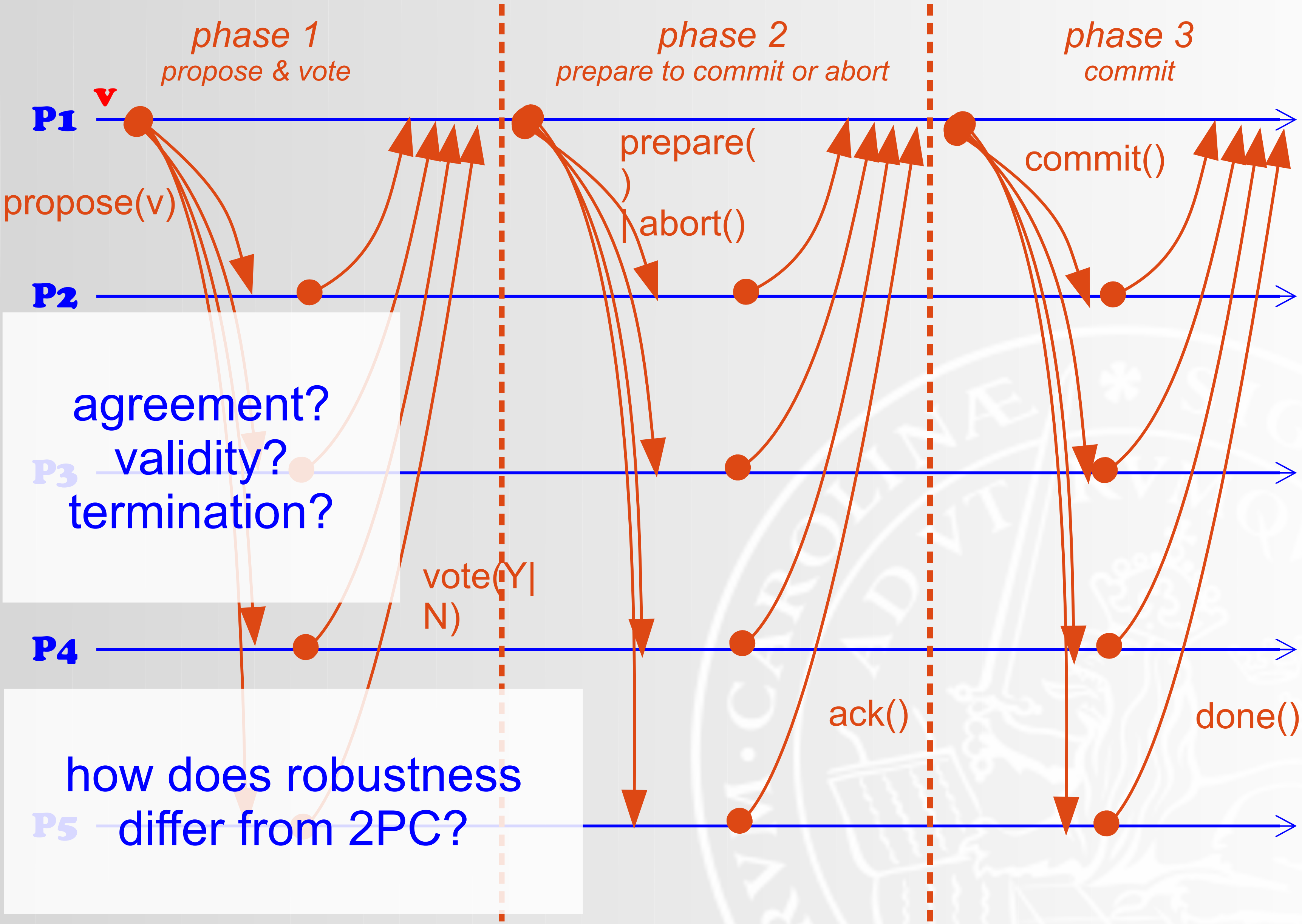
2PC (two-phase commit)



agreement?
validity?
termination?

robustness?

3PC (three-phase commit)



agreement?
validity?
termination?

how does robustness
differ from 2PC?