Distributed Systems

50.520 Systems Security

Paweł Szałachowski

“A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.”

–Leslie Lamport

Intro

- Systems get large and complex
- Distributed system (selected topics)
  - Authentication and access control (50.574 and here)
  - Concurrency (50.574)
  - **Failures and recovery** (partially 50.574)
  - **Naming**

Fault Tolerance and Failure Recovery

- Availability has been neglected in research
  - in contrast to confidentiality, authenticity, and integrity
  - >30% of IT budget goes for availability
  - Fault tolerance and failure recovery are critical for continuous operation

- Fault
  - Fault may cause an error (incorrect state), that may lead to a failure (deviation from system's specified behavior)
- Fault Tolerance
  - Based on logs and locking (very complicated to implement)
- Resilient system has a number of components
  - Fault detection, error recovery, failure recovery
  - Mean-time-before-failure (MTBF), mean-time-to-repair (MTTR)

Failure Models

- Failure models are subset of adversary/threat models
- Byzantine Failure
  - *n* generals defending Byzantium, *t* of whom are malicious
    - Generals can pass authentic and confidential messages
- What is max *t* that can be tolerated?

  n >= *3t* + *1* (Lamport, Shostak, and Pease)

Interaction with Fault Tolerance

- Constraining the failure rate
  - Redundancy
  - Fail-stop processors
    - Example: IBM System/88
      - 2x bus, 2x disk, 2x CPU
      - Stop if an error is detected
- Implications of redundancy and fail-stop behavior?
  - Confidentiality, DoS, …

PBFT

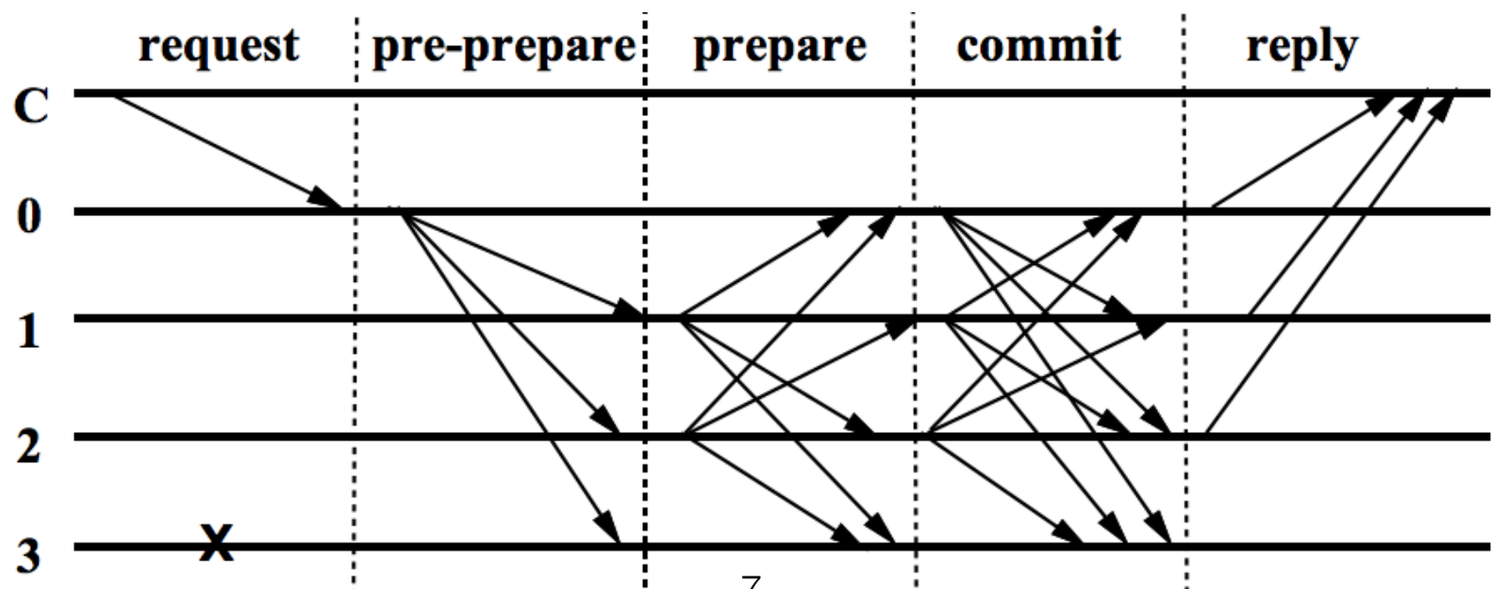
- Castro and Liskov “Practical Byzantine Fault Tolerance”, 1999

    1. A client sends a request to invoke a service operation to the primary

    2. The primary multicasts the request to the backups

    3. Replicas execute the request and send a reply to the client

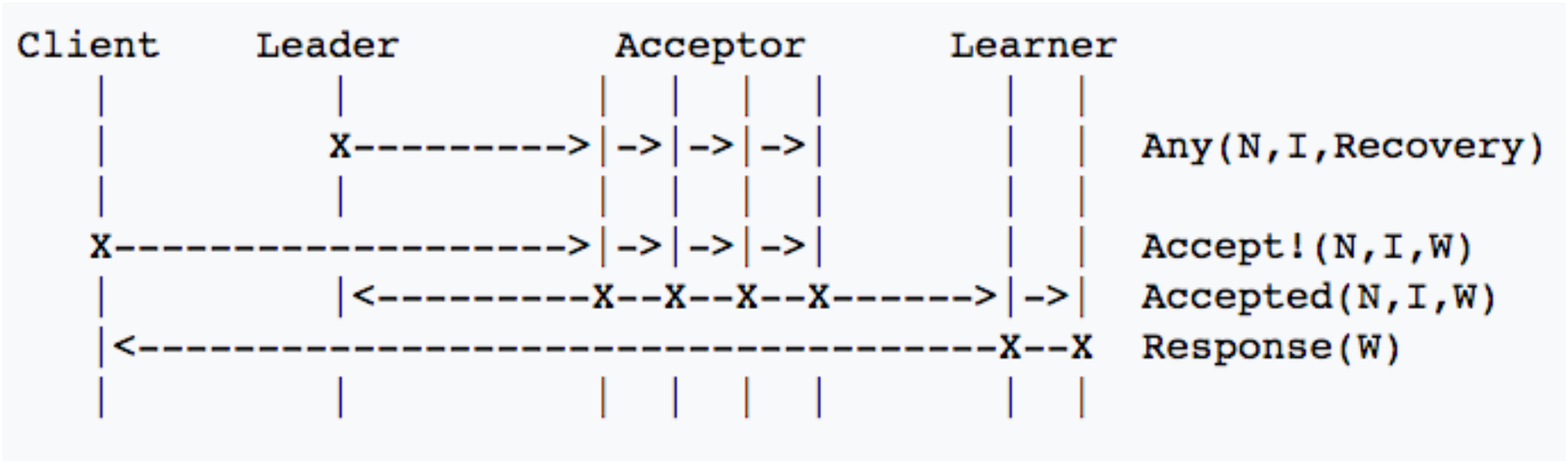
    4. The client waits for *f+1* replies from different replicas with the same result; this is the result of the operation.



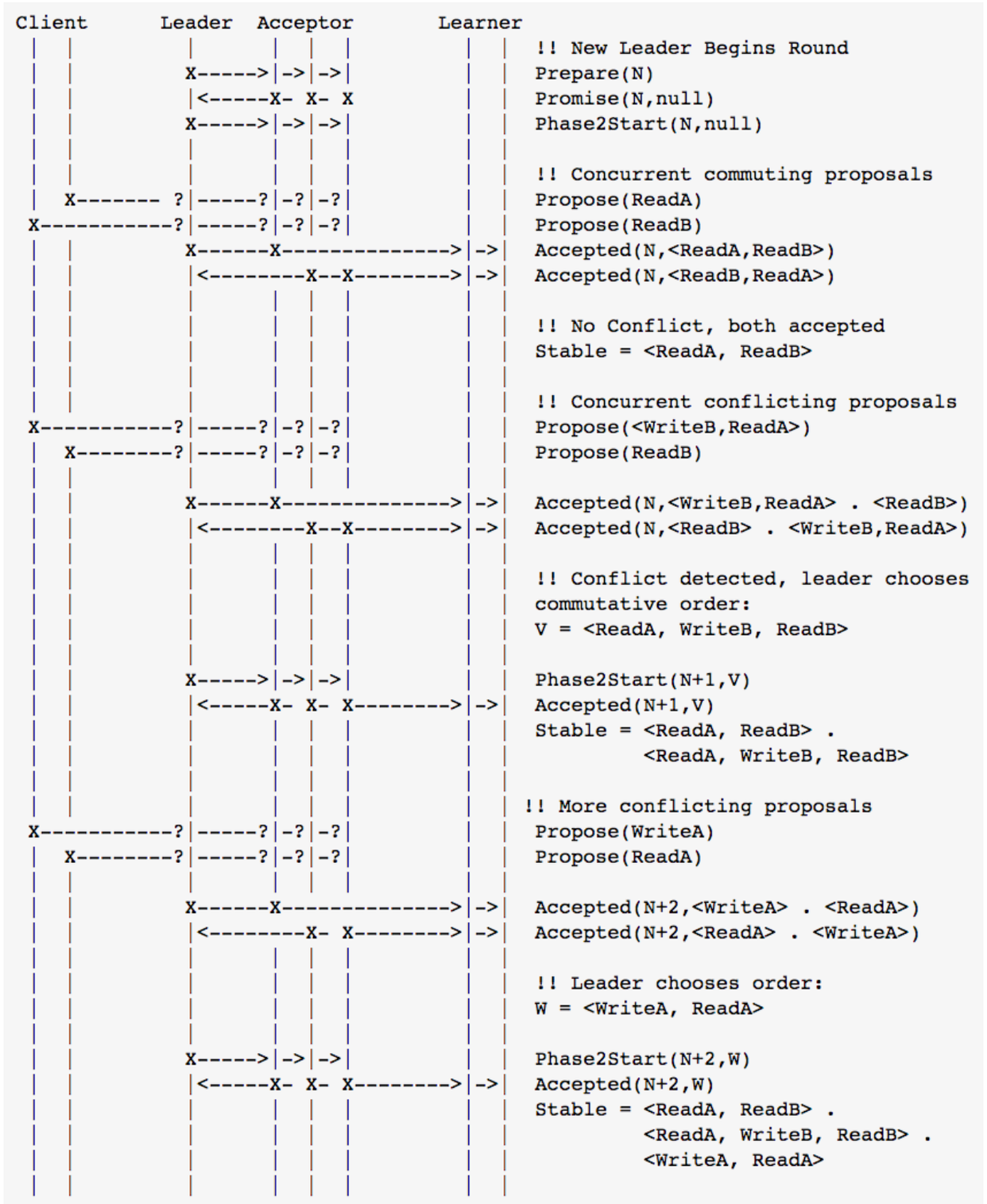
Other protocols

- Paxos
  - Synchronization problem
  - Used by Google (Chubby, Spanner, Megastore, …)
    - OpenReplica, IBM SAN Volume Controller, …
- Raft
  - designed as an alternative to Paxos
  - more understandable

Paxos


Client
Leader
Acceptor
Learner
Any(N,I,Recovery)
Accept!(N,I,W)
Accepted(N,I,W)
Response(W)

Paxos


Client Leader Acceptor Learner
!! New Leader Begins Round
Prepare(N)
Promise(N,null)
Phase2Start(N,null)
!! Concurrent commuting proposals
Propose(ReadA)
Propose(ReadB)
Accepted(N,<ReadA,ReadB>)
Accepted(N,<ReadB,ReadA>)
!! No Conflict, both accepted
Stable = <ReadA, ReadB>
!! Concurrent conflicting proposals
Propose(<WriteB,ReadA>)
Propose(ReadB)
Accepted(N,<WriteB,ReadA> . <ReadB>)
Accepted(N,<ReadB> . <WriteB,ReadA>)
!! Conflict detected, leader chooses
commutative order:
V = <ReadA, WriteB, ReadB>
Phase2Start(N+1,V)
Accepted(N+1,V)
Stable = <ReadA, ReadB> .
<ReadA, WriteB, ReadB>
!! More conflicting proposals
Propose(WriteA)
Propose(ReadA)
Accepted(N+2,<WriteA> . <ReadA>)
Accepted(N+2,<ReadA> . <WriteA>)
!! Leader chooses order:
W = <WriteA, ReadA>
Phase2Start(N+2,W)
Accepted(N+2,W)
Stable = <ReadA, ReadB> .
<ReadA, WriteB, ReadB> .
<WriteA, ReadA>

How to use?

- Implement a distributed key:value storage (filesystem)
  - Universal (easy to implement other primitives on top)
    - Distributed locks
    - Leader election (often protocols provide it by default)
    - Membership enumeration
    - …
- Easy to combine with other services, load balancers, etc…

CAP theorem

- "*it is impossible for a distributed data store to simultaneously provide more than two out of the following three guarantees*" - Eric Brewer

    - Consistency: Every read receives the most recent write or an error

    - Availability: Every request receives a (non-error) response – without guarantee that it contains the most recent write

    - Partition tolerance: The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes

- Examples?

Resilience

- "*ability to provide and maintain an acceptable level of service in the face of faults*"

- Building "virtual servers" on top of a number of cheap machines

  - Reduce risk by introducing many vendors

  - Easier to detect that one component is attacked/ malicious

- Distributed storage and redundancy

Redundancy

- Level of redundancy
  - Hardware (e.g., System/88)
    - disks, CPUs, mem, …
  - Process group redundancy (system level)
    - Multiple copies of a system (different locations)
- Backup (system/data level)
  - Copy of a system taken and archived at regular intervals
  - Journals: keeps track of changes not yet committed to the file system
- Fallback: limited functionality in application layer (less capable than backup)

RAID

- Redundant Arrays of Inexpensive Disks (RAID)
- Different levels
    - RAID 0: consists of striping, without mirroring or parity
    - RAID 1: consists of data mirroring, without parity or striping
    - RAID 2: consists of bit-level striping with dedicated Hamming-code parity
    - RAID 3: consists of byte-level striping with dedicated parity
    - …
- Hybrid
    - RAID 01: creates two stripes and mirrors them
    - RAID 10: creates a striped set from a series of mirrored drives

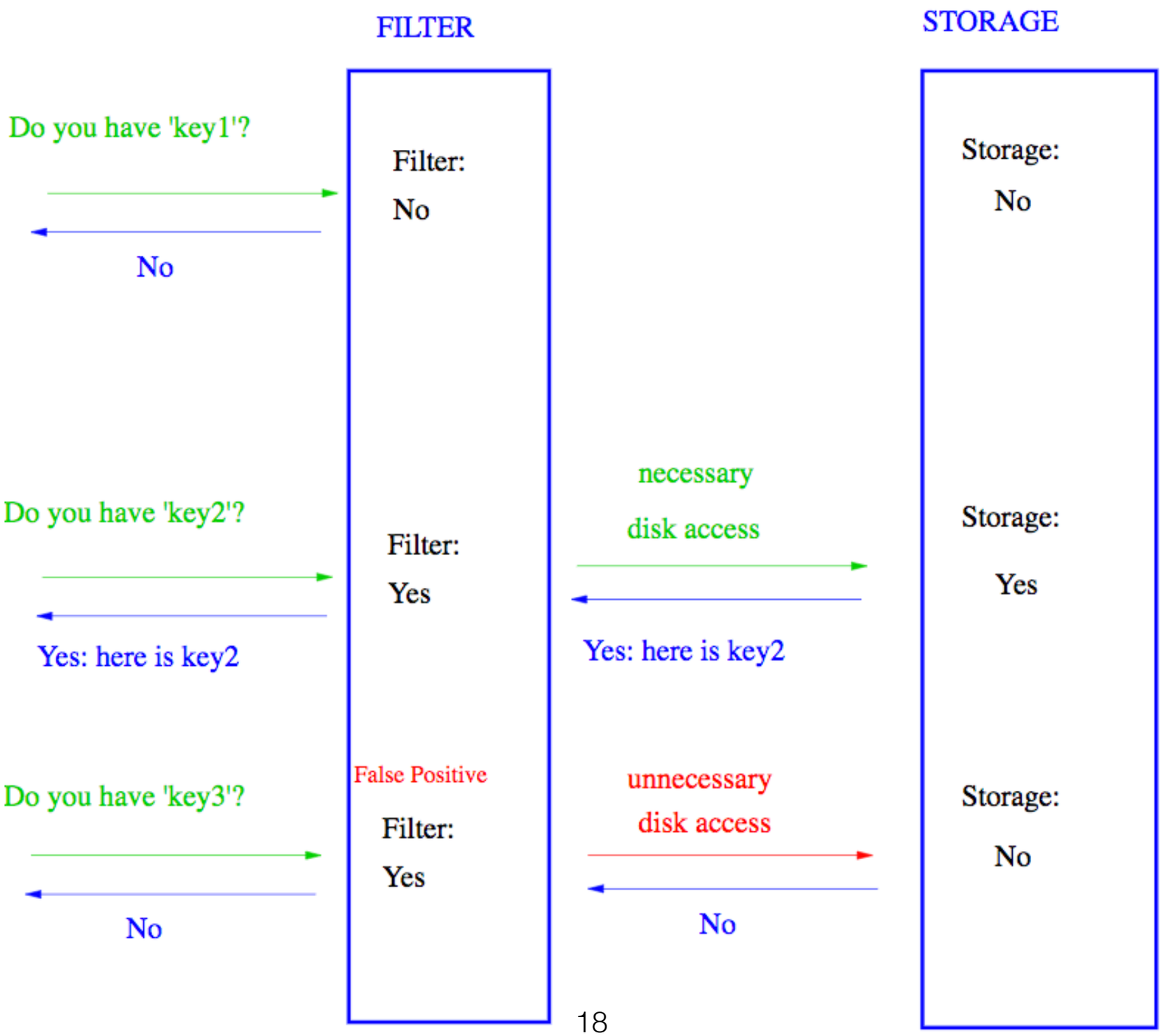
Service Availability

- Standard architecture of large databases
  - Cache + DB
    - Entries are taken from DB
      - Popular entries are cached (for efficiency reasons)
- Problem
  - Adversary can query database for non-existing entries
    - Search executes in the pessimistic time (often storage lookups)
- How to mitigate it?

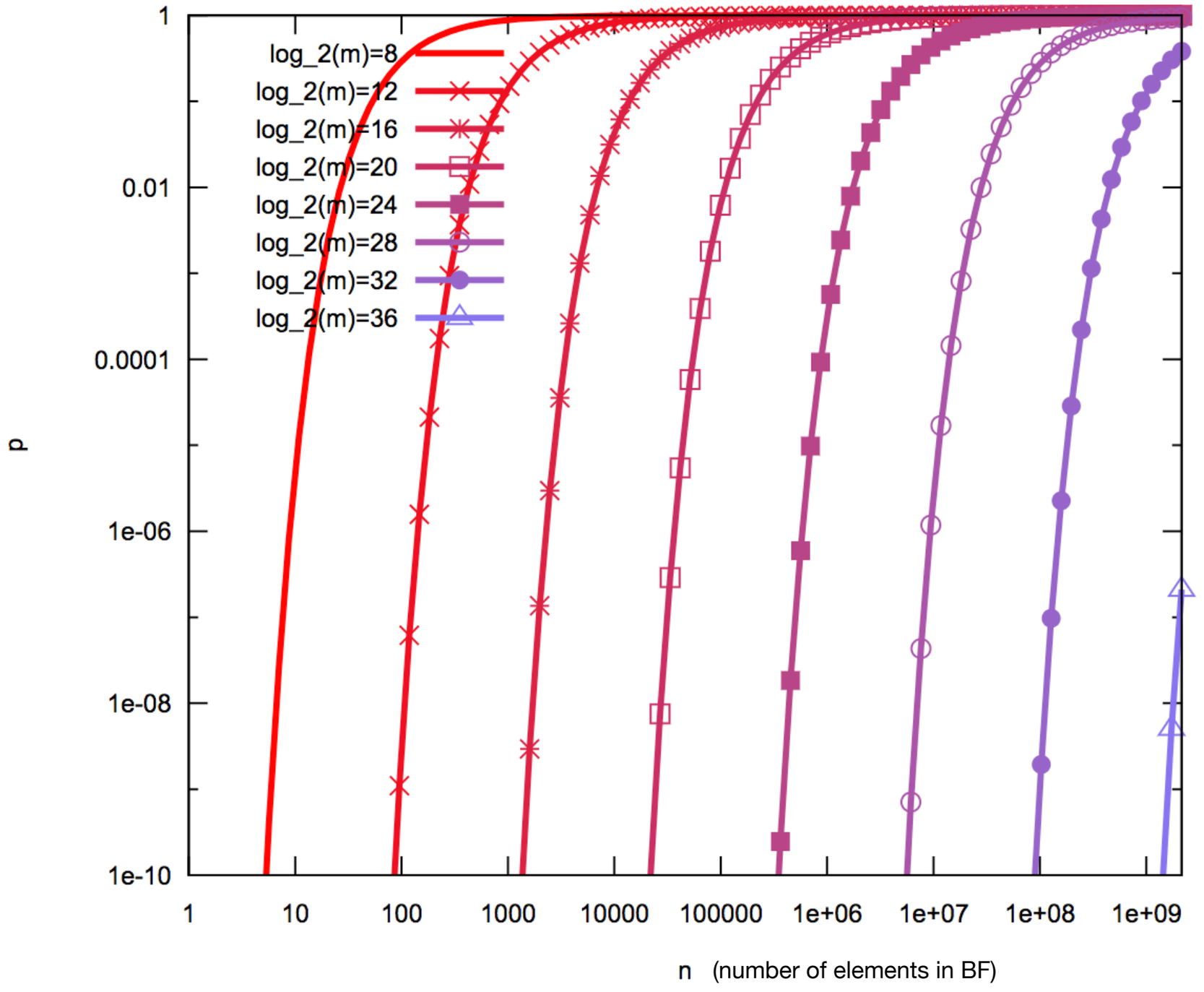
Bloom Filters (BFs)

- Space-efficient probabilistic data structure
- Set membership (check quickly whether an element is in a set, without storing the element)
- *m*-bits long bit array (initially, all bits set to 0)
- *k* different hash functions, each maps set's element to one of *m* array positions (usually $k << m$)
    - They do not have to be cryptographically-strong hash functions
- Adding element
    - Hash element with *k* hash functions and set 1 on the obtained positions
- Querying element
    - Hash element with *k* hash functions, if bits on all positions equal 1 return TRUE, o/w FALSE
    - False positives possible, no false negatives

BF-backed Database



False Positives Probability



k=(m/n)ln2 (optimal)

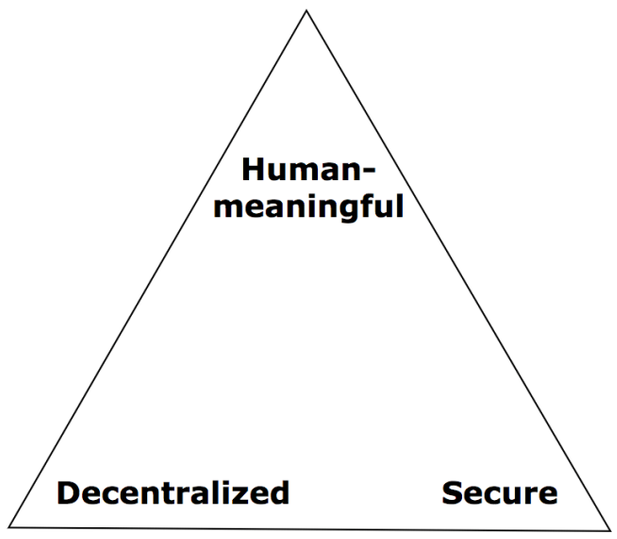
Naming

Distributed Naming

- Naming in distributed systems is hard
  - Especially for security
- How to name machines, organizations, persons, entities, … ?
  - Name collision may lead to authentication/authorization failures
- Names exist in contexts
- What namespaces do you know?

Zooko’s Triangle

- No single kind of name can achieve more than two:
  - Human-meaningful: Meaningful and memorable (low-entropy) names are provided to the users.
  - Secure: The amount of damage a malicious entity can inflict on the system should be as low as possible.
  - Decentralized: Names correctly resolve to their respective entities without the use of a central authority or service.
- Examples
  - DNSSec
  - .onion and Self-certifying File System (SFS)



- It is believed that open and distributed consensus (blockchains) relaxes it

Needham's Principles

1. The function of names is to facilitate sharing
2. The naming information may not all be in one place, and so resolving names brings all the general problems of a distributed system
3. It is bad to assume that only so many names will be needed
4. Global names buy you less than you think
5. Names imply commitments, so keep the scheme flexible enough to cope with organizational changes
6. Names may double as access tickets, or capabilities
7. Things are made much simpler if an incorrect name is obvious
8. Consistency is hard, and is often fudged. If directories are replicated, then you may find yourself unable to read, or to write, depending on whether too many or too few directories are available
9. Don't get too smart. Phone numbers are much more robust than computer addresses
10. Some names are bound early, others not; and in general it is a bad thing to bind early if you can avoid it

Issues

- Participants may be known by different names
  - Bank account, email, id, …
- Identity vs name ?
  - Identity is when different names correspond to the same principal
  - “The Elliott Bell who owns bank account number 12345678 is the Elliott James Bell with the US passport number 98765432 and date of birth 3/4/56”
- Semantic content of names
- Cultural assumptions (name changes, human naming, …)

Issues

- Uniqueness of names
    - Growth of travel in Middle Ages led to a need of surnames
    - When we need unique names?
        - emails and usernames have scalability problems (Internet)
        - Sequence numbers are often used
- Stability of names and addresses
    - IPv6 mapping, emails, domains
- Social context
    - Accounts, urls, …
- Restrictions
    - Private names, restricted names, …

Questions ?