What properties should asymmetric quorum systems satisfy?

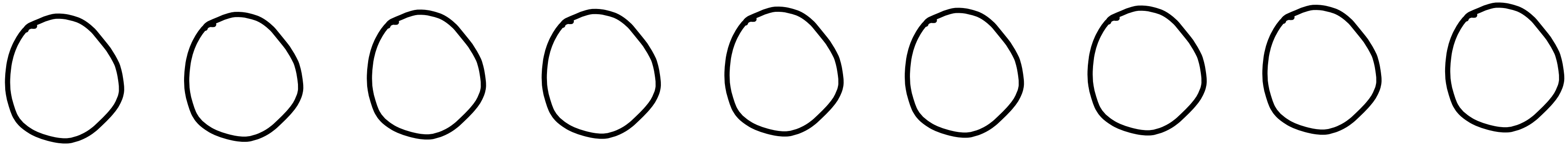
Juan Villacis
University of Bern

Failures

In distributed systems, failures are inevitable

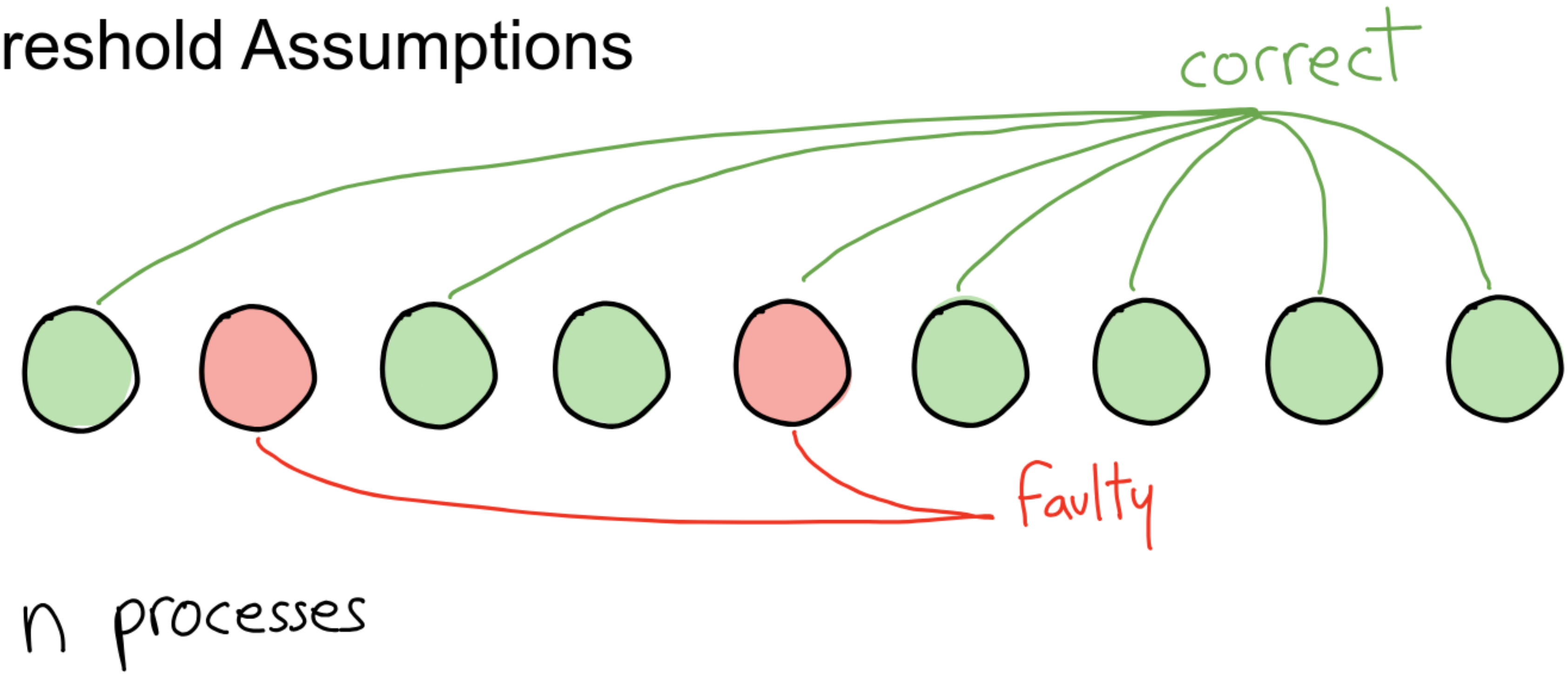
- Network errors
- Software bugs
- Malicious participants
- Machine crashes

Threshold Assumptions

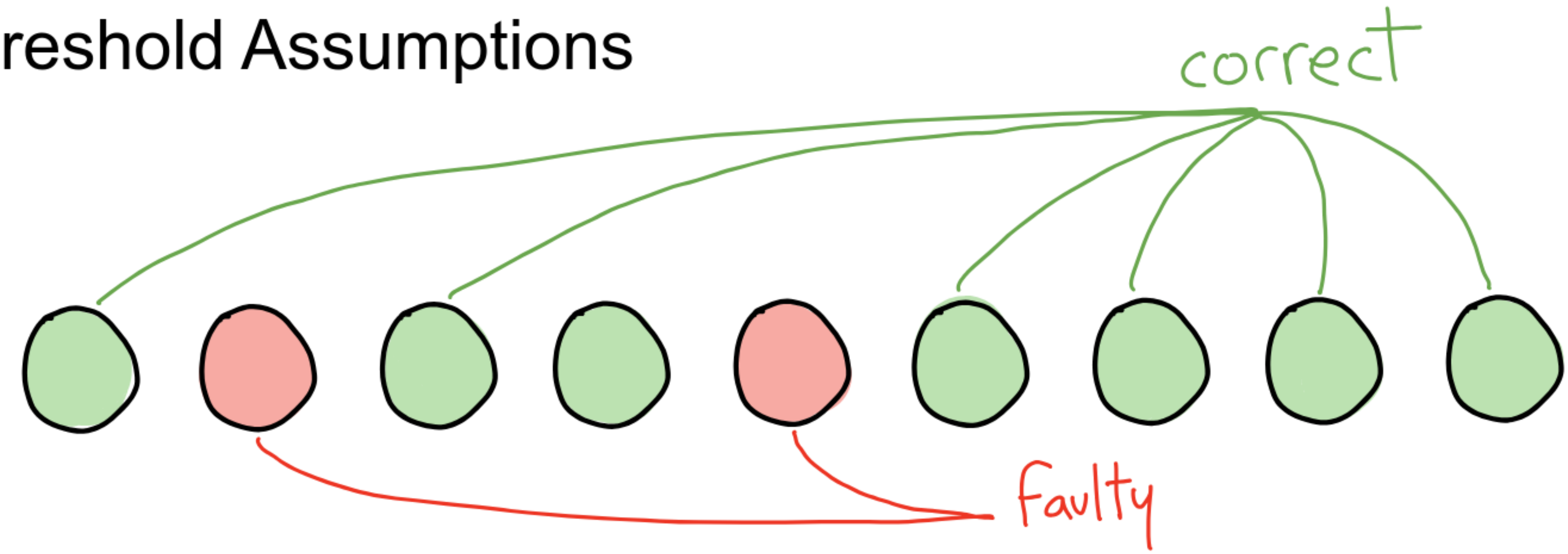


n processes

Threshold Assumptions



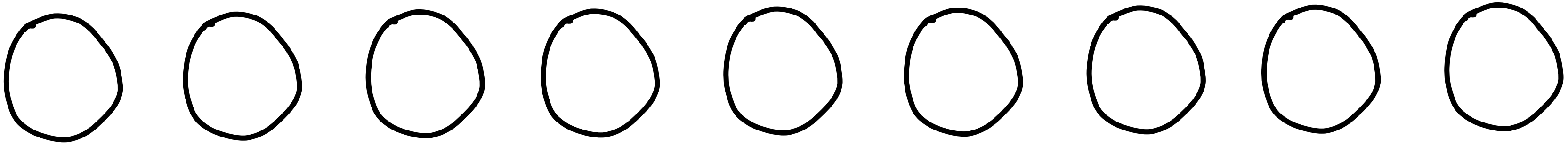
Threshold Assumptions



n processes

assumption: at most f processes are faulty

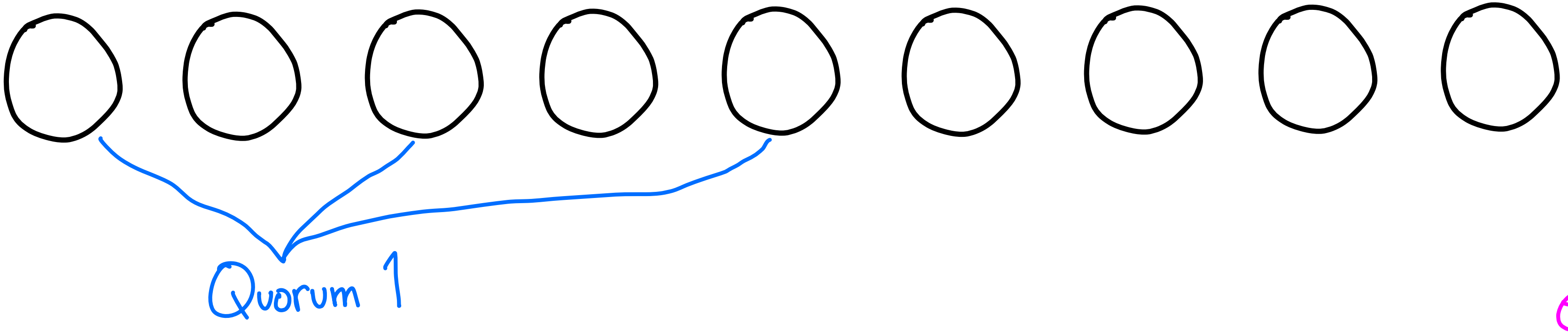
Symmetric Generalized Assumptions



shared by everyone

Define sets of processes that should not fail together (a.k.a quorums)

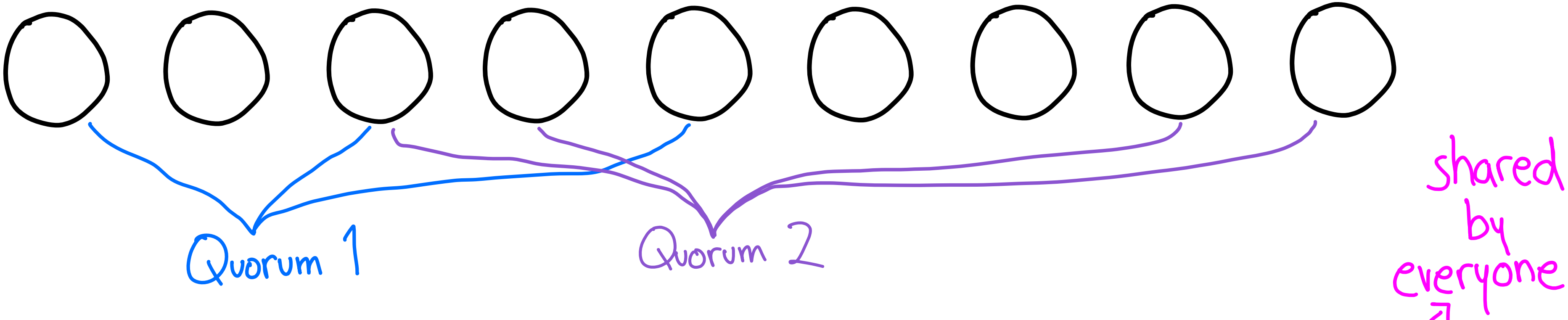
Symmetric Generalized Assumptions



shared by everyone

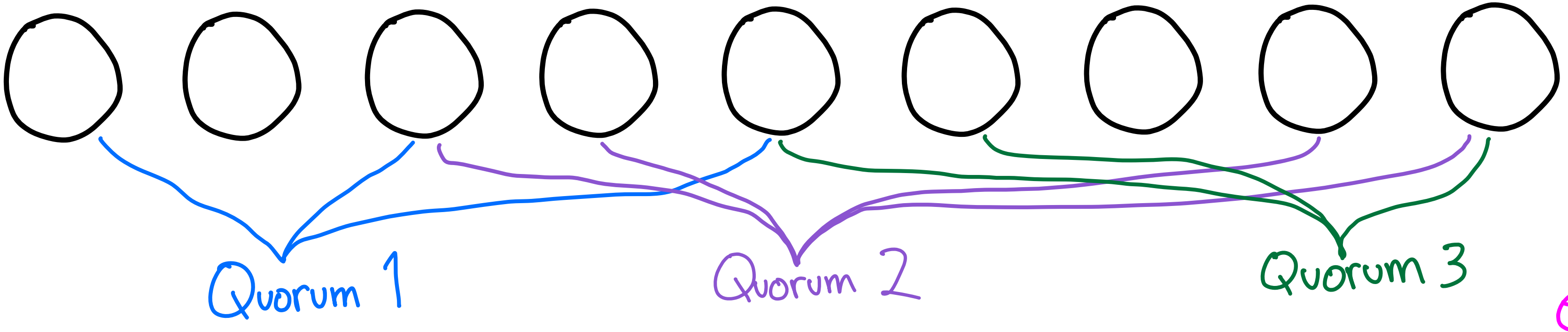
Define sets of processes that should not fail together (a.k.a quorums)

Symmetric Generalized Assumptions



Define sets of processes that should not fail together (a.k.a quorums)

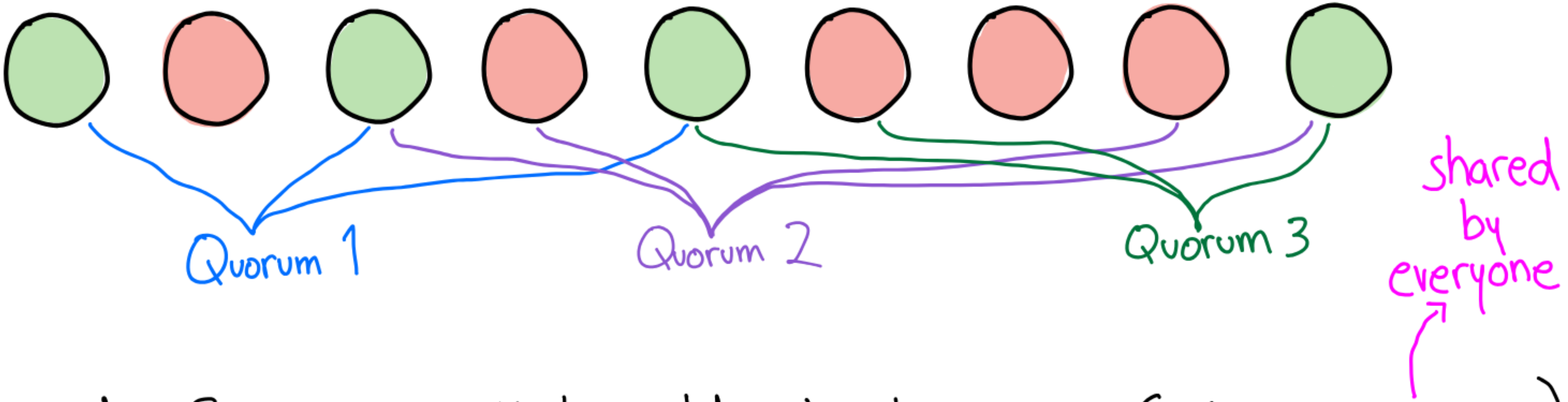
Symmetric Generalized Assumptions



shared by everyone

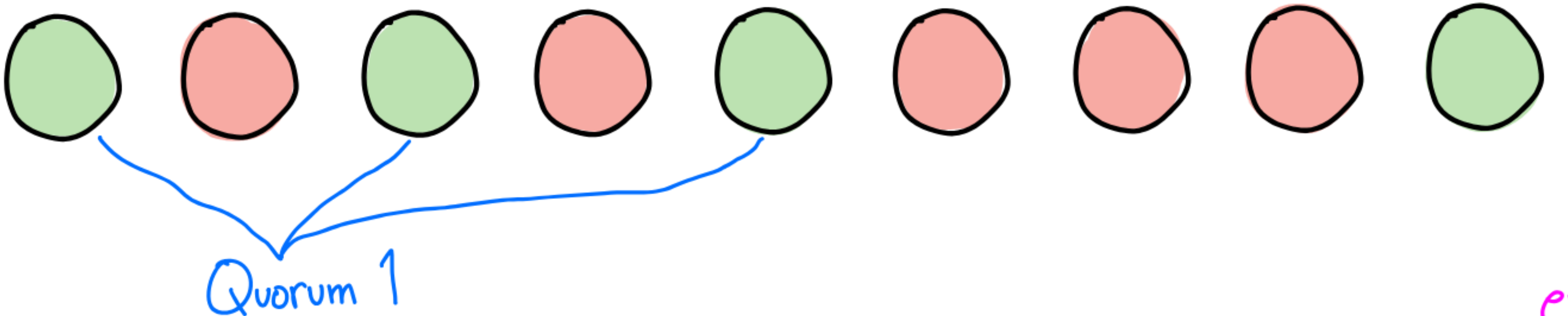
Define sets of processes that should not fail together (a.k.a quorums)

Symmetric Generalized Assumptions



Define sets of processes that should not fail together (a.k.a quorums)

Symmetric Generalized Assumptions



shared by everyone

Define sets of processes that should not fail together (a.k.a quorums)

assumption: at least one quorum has no faulty processes

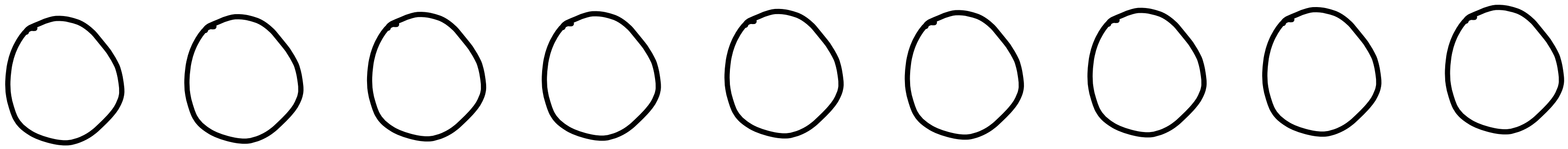
Trust is Subjective

Threshold and symmetric failure assumptions = shared by everyone

In reality not everybody trusts the same entities

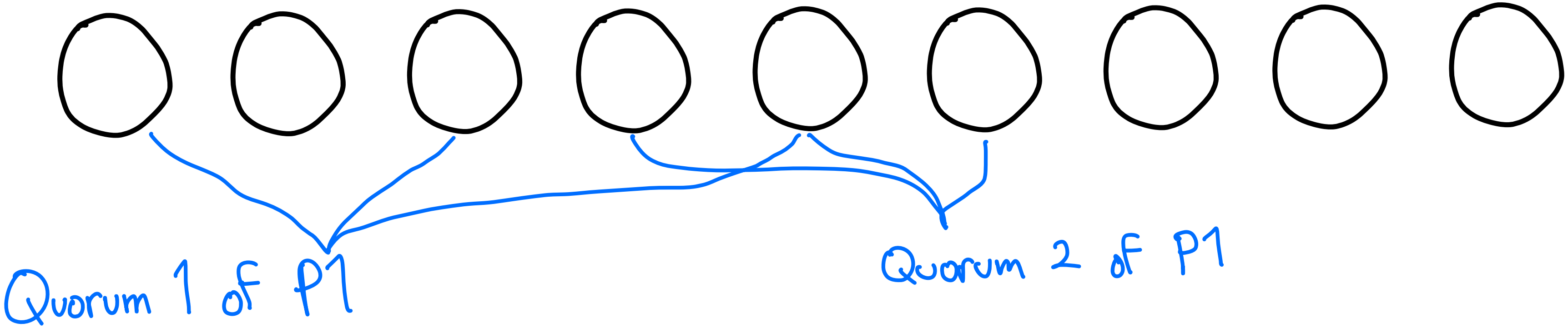
Each process should be able to determine who it trusts independently

Asymmetric Generalized Assumptions



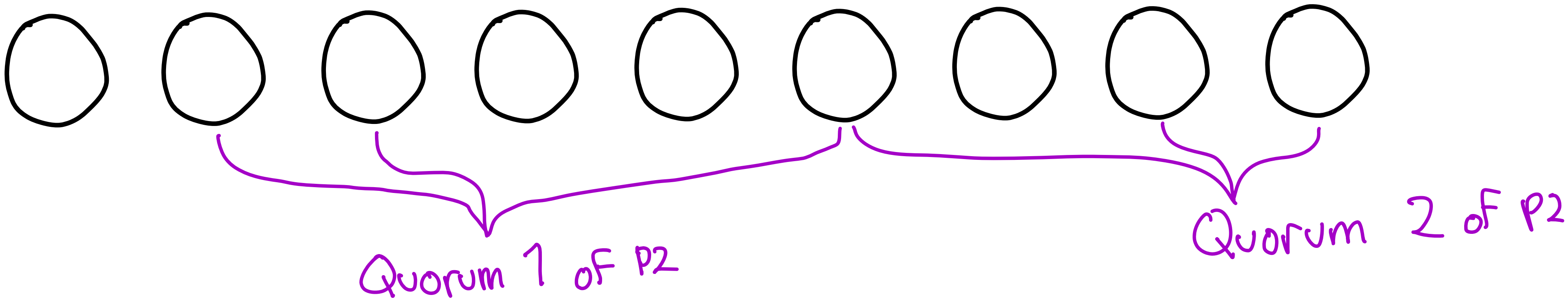
Each process defines its own quorums

Asymmetric Generalized Assumptions



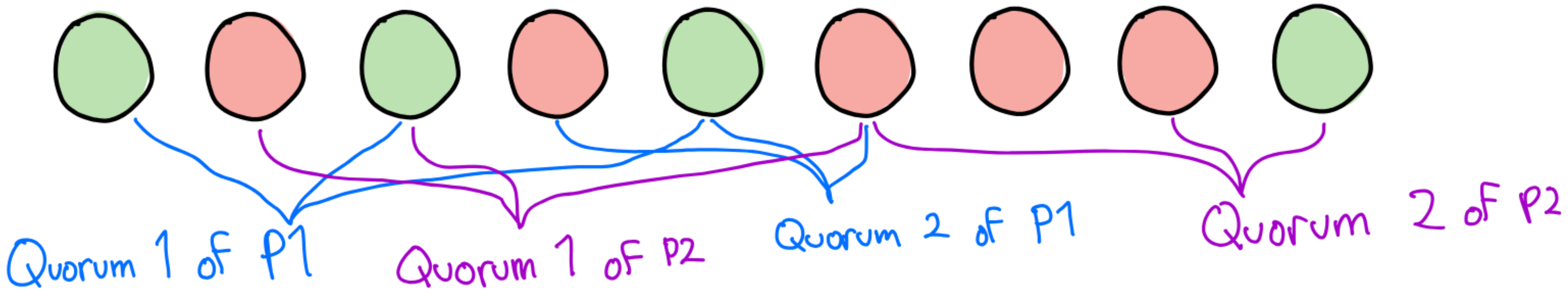
Each process defines its own quorums

Asymmetric Generalized Assumptions



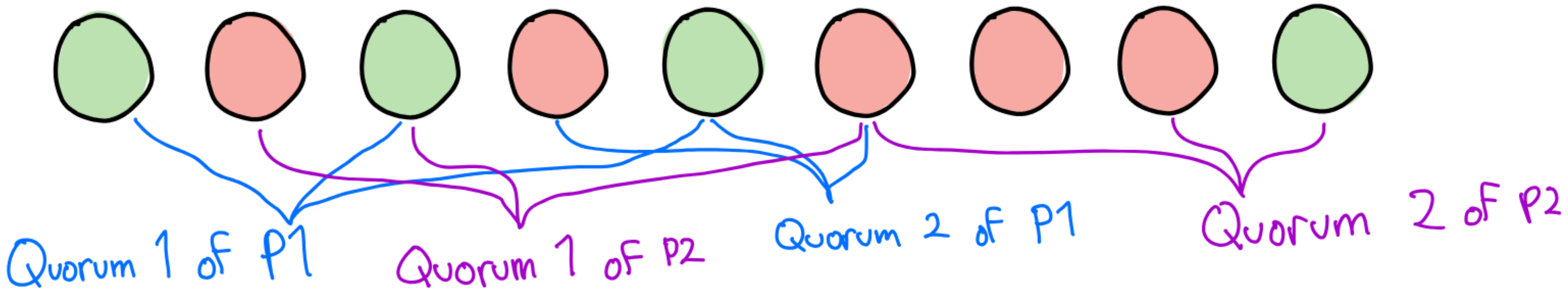
Each process defines its own quorums

Asymmetric Generalized Assumptions



Each process defines its own quorums

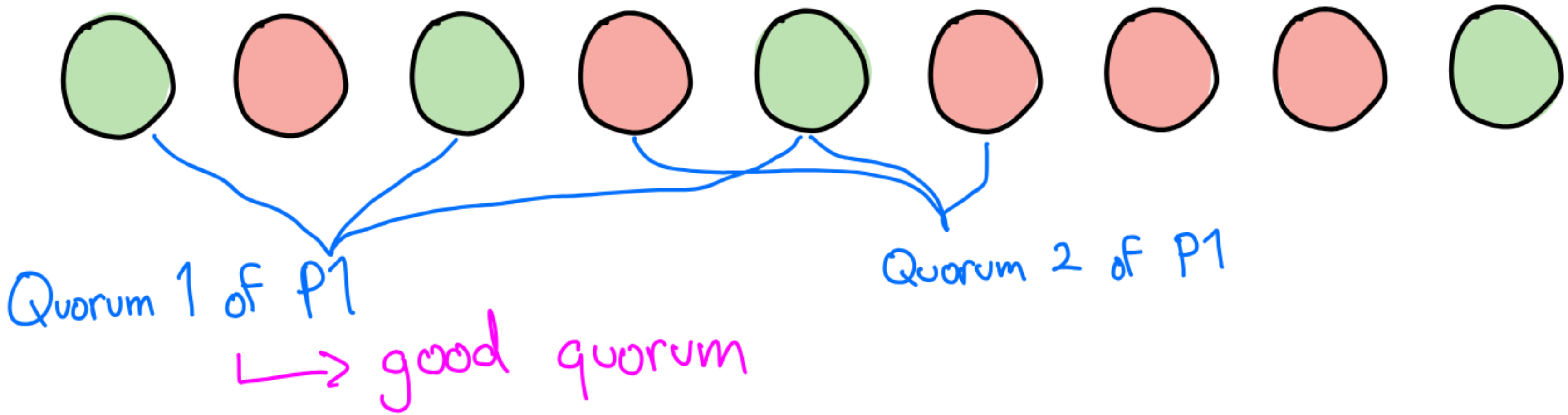
Asymmetric Generalized Assumptions



Each process defines its own quorums

assumption: if I defined my quorums correctly, at least one of my quorums has no faulty processes

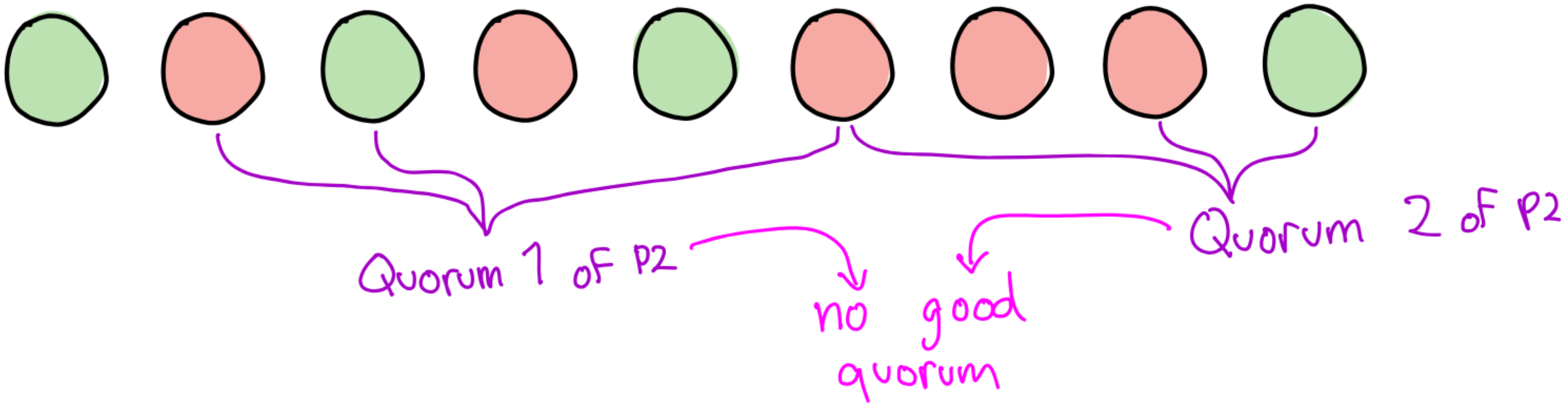
Asymmetric Generalized Assumptions



Each process defines its own quorums

assumption: if I defined my quorums correctly, at least one of my quorums has no faulty processes

Asymmetric Generalized Assumptions



Each process defines its own quorums

assumption: if I defined my quorums correctly, at least one of my quorums has no faulty processes

Secure Protocols with Asymmetric Trust

Ivan Damgård[1], Yvo Desmedt[2], Matthias Fitzi[3], and Jesper Buus Nielsen[1]

[1] Aarhus University, Dept. of Computer Science, 8200 Aarhus N, Denmark
{ivan,buus}(at)daimi.au.dk

[2] University College London, Dept. of Computer Science, London WC1E 6BT, United Kingdom
y.desmedt(at)cs.ucl.ac.uk

[3] ETH Zürich, Dept. of Computer Science, 8092 Zürich, Switzerland
fitzi(at)inf.ethz.ch

Asymmetric Distributed Trust[1]

Orestis Alpos[2,4]
University of Bern & Common Prefix
oralpos@gmail.com

Christian Cachin[2]
University of Bern
christian.cachin@unibe.ch

Björn Tackmann[3]
DFINITY
bjoern@dfinity.org

Luca Zanolini[4]
Ethereum Foundation
luca.zanolini@ethereum.org

3 May 2024

Stellar Consensus by Instantiation

Giuliano Losa[*]
Galois, Inc.
giuliano@galois.com

Eli Gafni[†]
UCLA
eli@ucla.edu

David Mazières[‡]
Stanford
http://www.scs.stanford.edu/~dm/addr/

August 12, 2019

Heterogeneous Paxos: Technical Report

Isaac Sheff
Max Planck Institute for Software Systems, Campus E1 5, Room 531, 66121 Saarbrücken, Germany
https://IsaacSheff.com
isheff@mpi-sws.org

Xinwen Wang
Cornell University, Gates Hall, 107 Hoy Road, Ithaca, New York, 14853, USA
https://www.cs.cornell.edu/~xinwen/
xinwen@cs.cornell.edu

Robbert van Renesse
Cornell University, 433 Gates Hall, 107 Hoy Road, Ithaca, New York, 14853, USA
https://www.cs.cornell.edu/home/rvr/
rvr@cs.cornell.edu

Andrew C. Myers
Cornell University, 428 Gates Hall, 107 Hoy Road, Ithaca, New York, 14853, USA
https://www.cs.cornell.edu/andru/
andru@cs.cornell.edu

Quorum Subsumption for Heterogeneous Quorum Systems

Xiao Li
University of California, Riverside, CA, USA

Eric Chan
University of California, Riverside, CA, USA

Mohsen Lesani
University of California, Riverside, CA, USA

Asymmetric Trust in the Wild

XRP LEDGER

Stellar

Model

Set of processes P

Each process knows the quorums of all members of P

Asynchronous communication

Byzantine faults

Asymmetric Trust

Processes are not correct/faulty anymore

Based on how accurate their trust assumptions, they can be classified as

- Faulty: if P_i is Faulty
- Naive: if P_i is Correct and all quorums have Faulty processes
- Wise: if P_i is Correct and at least one quorum has no Faulty processes

Symmetric vs Asymmetric Trust

Symmetric Reliable Broadcast

- **Consistency:** if a correct process delivers m and another correct process delivers m' then m=m'

- **Validity:** if broadcaster is correct then all correct processes will deliver its value

Asymmetric Reliable Broadcast

- **Consistency:** if a wise process delivers m and another wise process delivers m' then m=m'

- **Validity:** if broadcaster is correct then all processes that have sufficiently good quorums will deliver its value

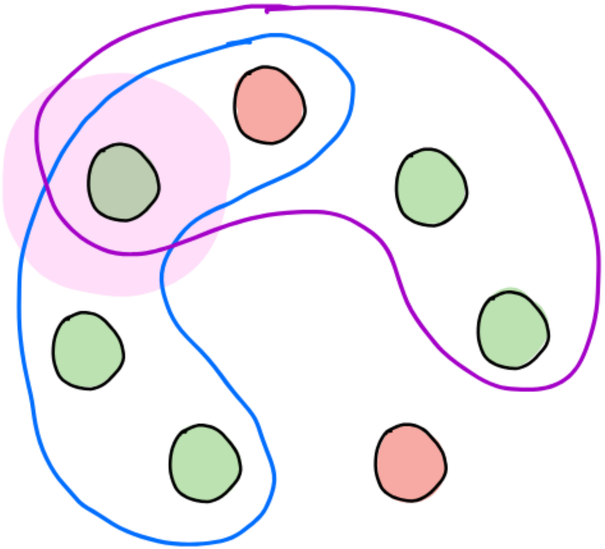
to be specified later

What properties should asymmetric quorum systems satisfy?

Juan Villacis
University of Bern

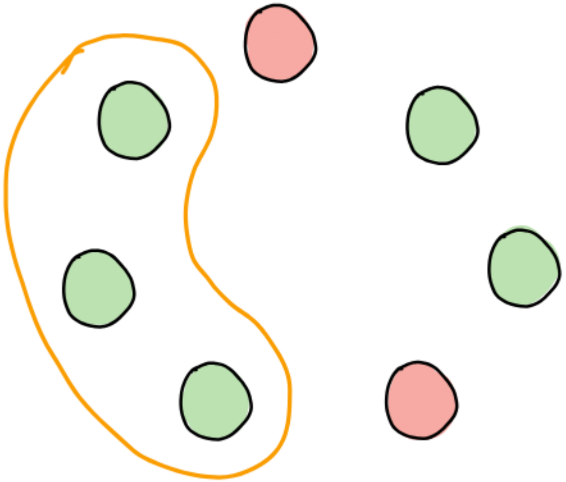
Properties: Symmetric Quorums

Consistency



Quorums intersect in a correct process

Availability



At least one quorum has no faulty processes

Properties: Asymmetric Quorums

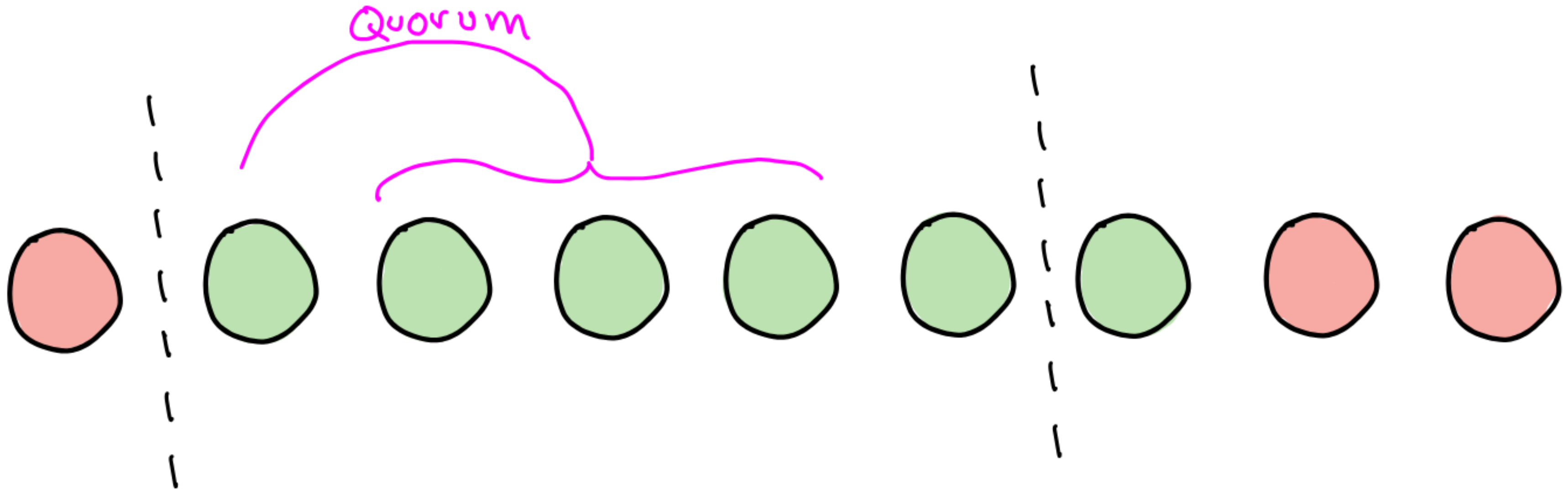
Consistency and availability

But we need more

Asymmetric quorum consistency and availability are not enough to solve reliable broadcast and consensus [LCL23]

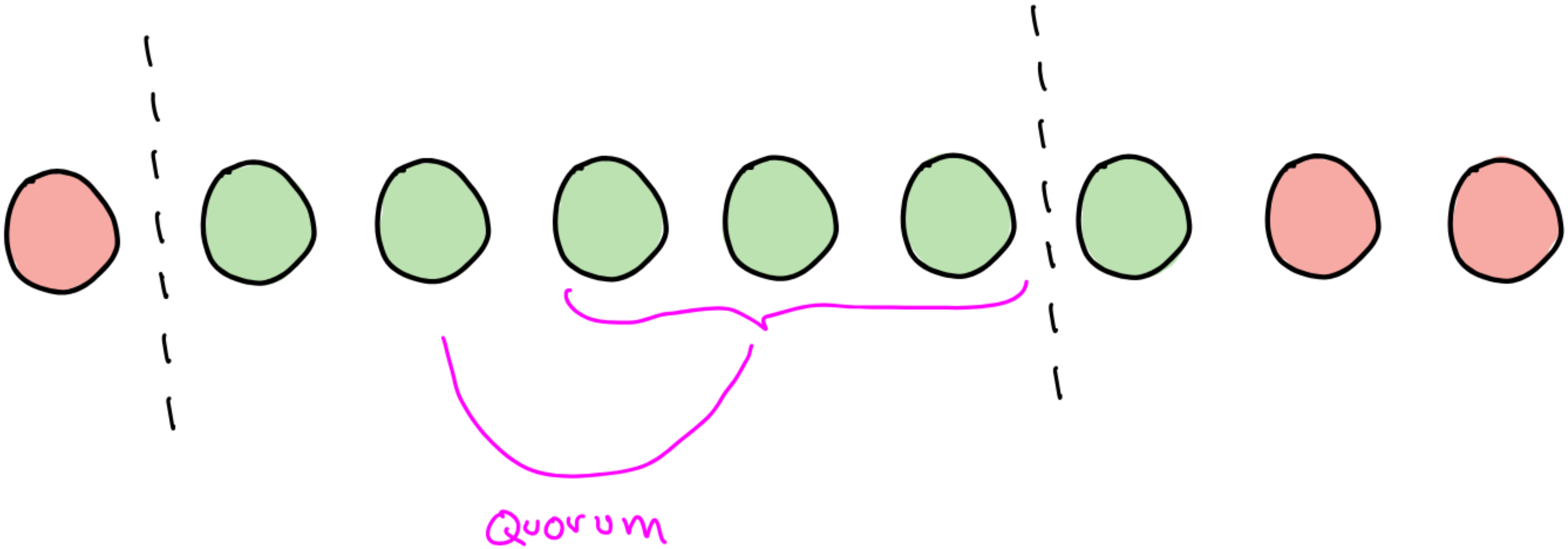
Guilds [ACTZ24]

Set of correct processes that contains a quorum for each member



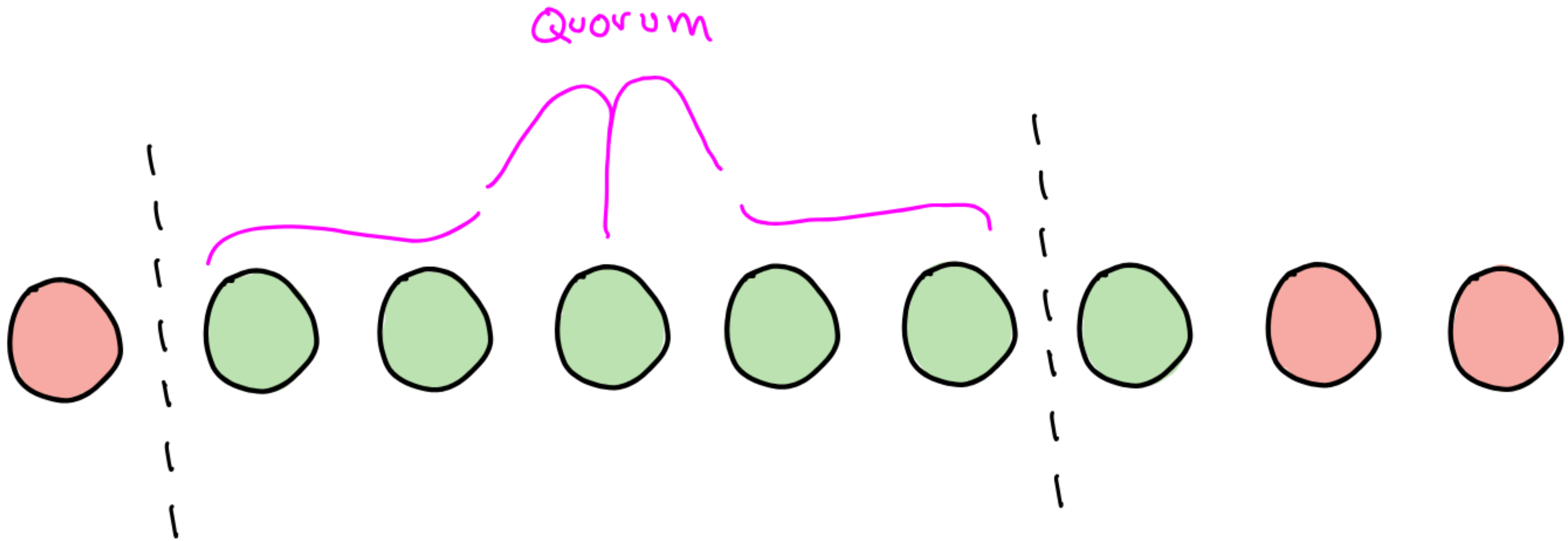
Guilds [ACTZ24]

Set of correct processes that contains a quorum for each member



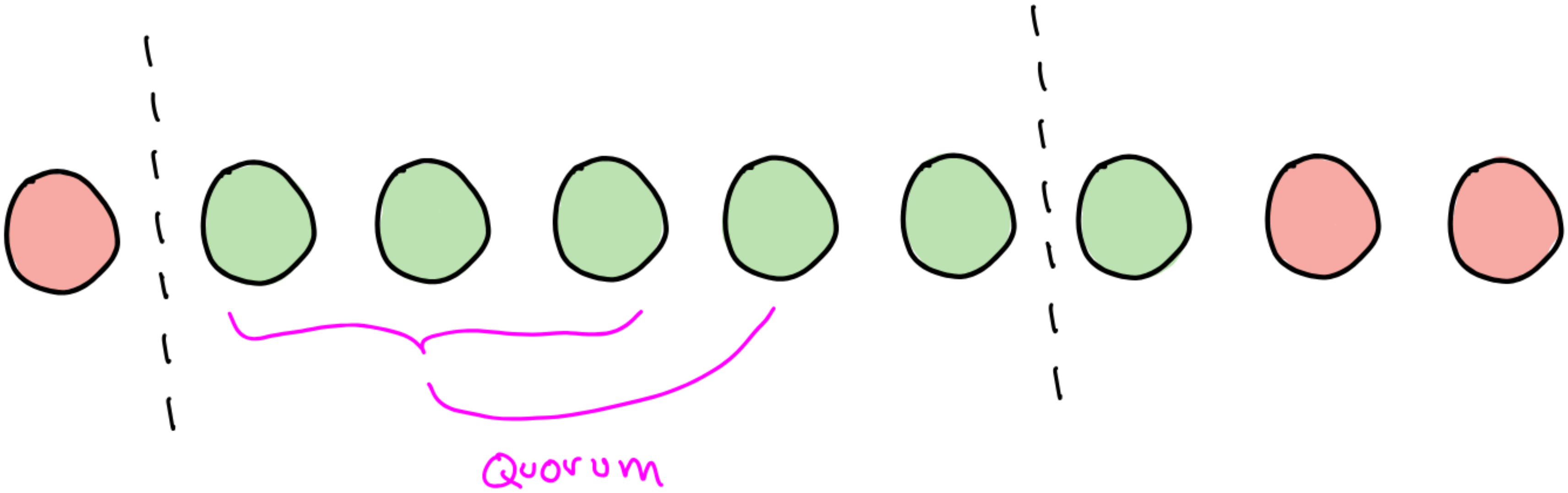
Guilds [ACTZ24]

Set of correct processes that contains a quorum for each member



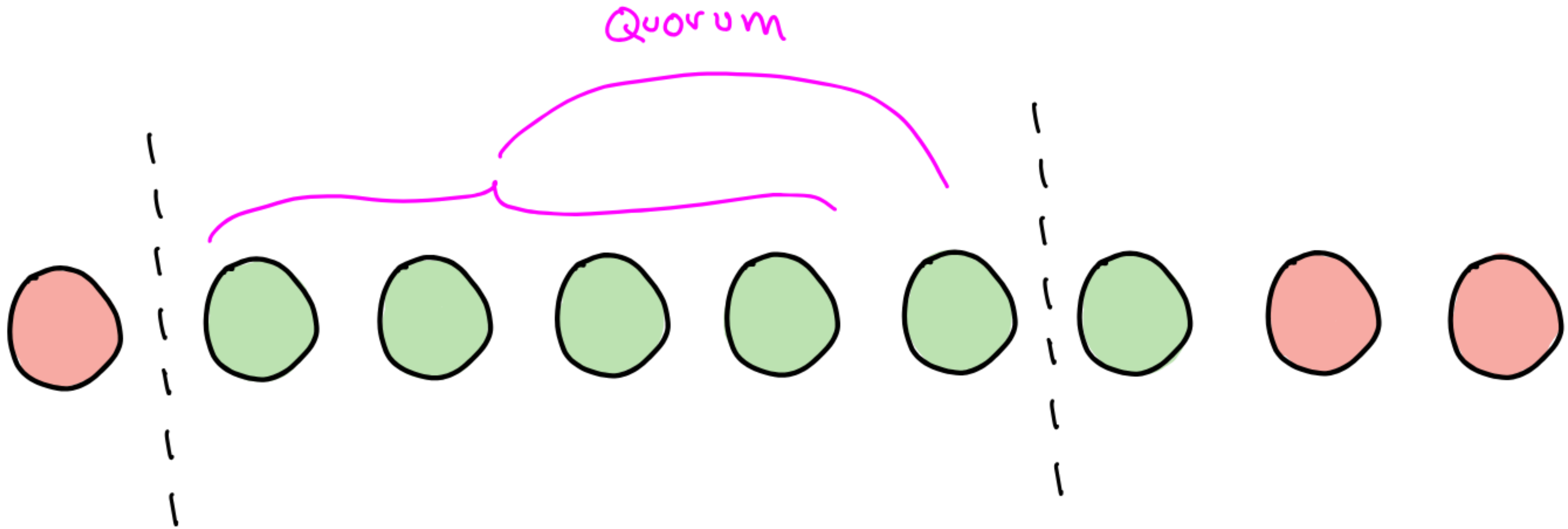
Guilds [ACTZ24]

Set of correct processes that contains a quorum for each member



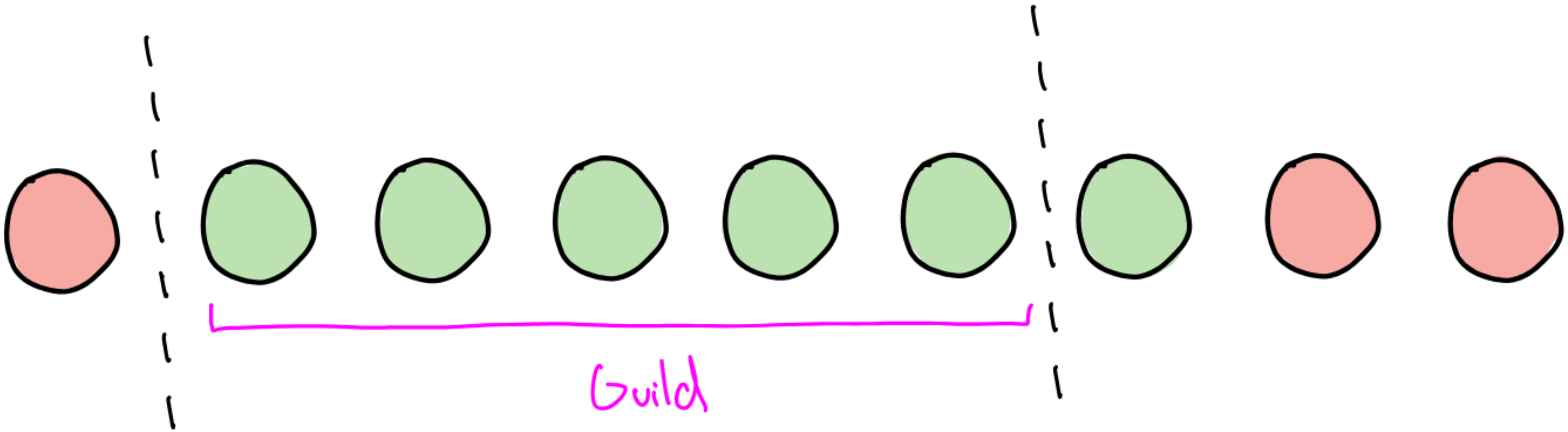
Guilds [ACTZ24]

Set of correct processes that contains a quorum for each member



Guilds [ACTZ24]

Set of correct processes that contains a quorum for each member



Guilds [ACTZ24]

The reliable broadcast and consensus algorithms of [ACTZ24] only work in executions that contain a guild

Guilds in Reliable Broadcast

Validity: if a correct process broadcasts a message then all processes in a guild will deliver the message

Totality: if a wise process delivers some message then all processes in a guild will deliver a message

Guilds in Other Models

[LGM19] use consensus clusters

[LCL23] use strong availability

Guilds are too Strong

Guilds are too Strong

If there is at least one guild we can build a symmetric quorum system from the asymmetric quorum system [ACV25]

No process will be worse off by using this symmetric version as its quorum system

So processes can derive the symmetric quorum system and use normal symmetric algorithms

Guild $\Rightarrow$ Symmetric Trust

Building a Symmetric Quorum System

Idea: treat each possible guild as a quorum

What is a possible guild?

A set of processes that is a guild in some execution

Building a Symmetric Quorum System

Idea: treat each possible guild as a quorum

What is a possible guild?

A set of processes that is a guild
in some execution

New symmetric quorum system = set of all possible guilds

$$Q = \{ guild_1, guild_2, \ldots, guild_m \}$$

Building a Symmetric Quorum System

😔: there is an exponential number of possible guilds, the new asymmetric quorum system is huge

Building a Symmetric Quorum System

😔: there is an exponential number of possible guilds, the new asymmetric quorum system is huge

😀: we don't need to 'build' the new quorum system, we just need to recognize its 'quorums' (i.e. guilds)

upon receiving messages from all process in some quorum do:

...

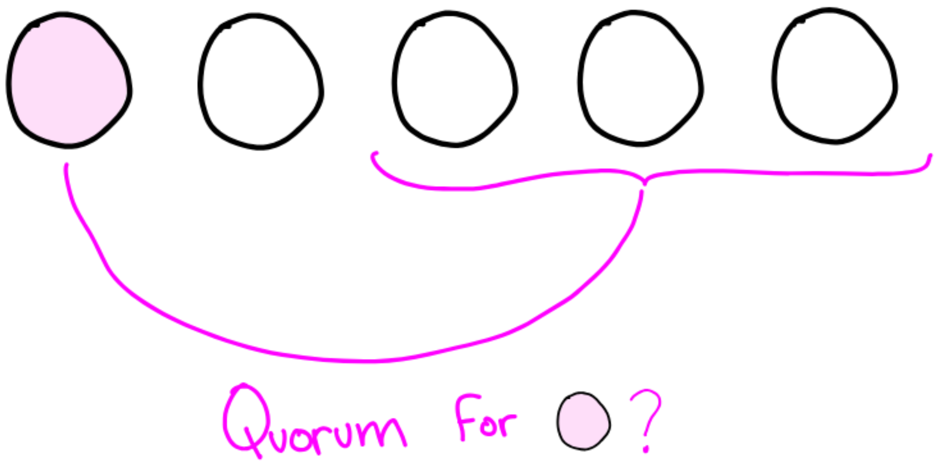
we need to recognize if we have received messages from a quorum

Recognizing Guilds Efficiently

Given a set of processes S

For each member: check if at least one of its quorums is a subset of S

If yes, then S is a guild in the execution where all members of S are not faulty



Asymmetric Trust in the Crash-fault Model

[SC25a] show that in the asynchronous crash-fault model, asymmetric trust reduces to symmetric trust

Derive a valid symmetric quorum system from the asymmetric one, keeping the same guarantees for all processes

It makes no sense to use asymmetric trust in the crash-fault scenario

Current Situation

All known algorithms for reliable broadcast and consensus only work if the execution has a guild

Guild ⇒ symmetric trust

Status: it makes no sense to use asymmetric trust in the Byzantine-fault scenario if you require guilds

Goal: is it possible to solve asymmetric reliable broadcast and consensus without guilds

How to Proceed?

Guilds: ❌ are too strong!

Consistency and availability: ❌ are too weak!

We need properties that are neither too weak nor too strong

Weaker Assumptions for Asymmetric Trust

Ignacio Amores-Sesar[1], Christian Cachin[2], and Juan Villacis[2]

[1] Aarhus University, Denmark
[2] University of Bern, Switzerland

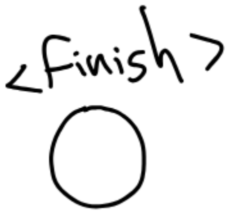
Messages Needed in an Algorithm

send <start>

upon quorum of <start>
    send <echo>

upon quorum of <echo>
    send <finish>

Messages Needed in an Algorithm



send <start>

upon quorum of <start>
    send <echo>

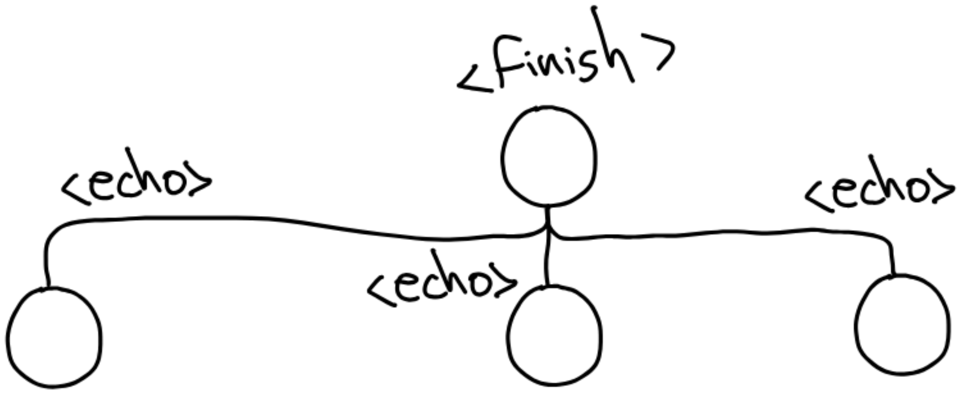
upon quorum of <echo>
    send <finish>

Messages Needed in an Algorithm

send <start>

upon quorum of <start>
    send <echo>

upon quorum of <echo>
    send <finish>

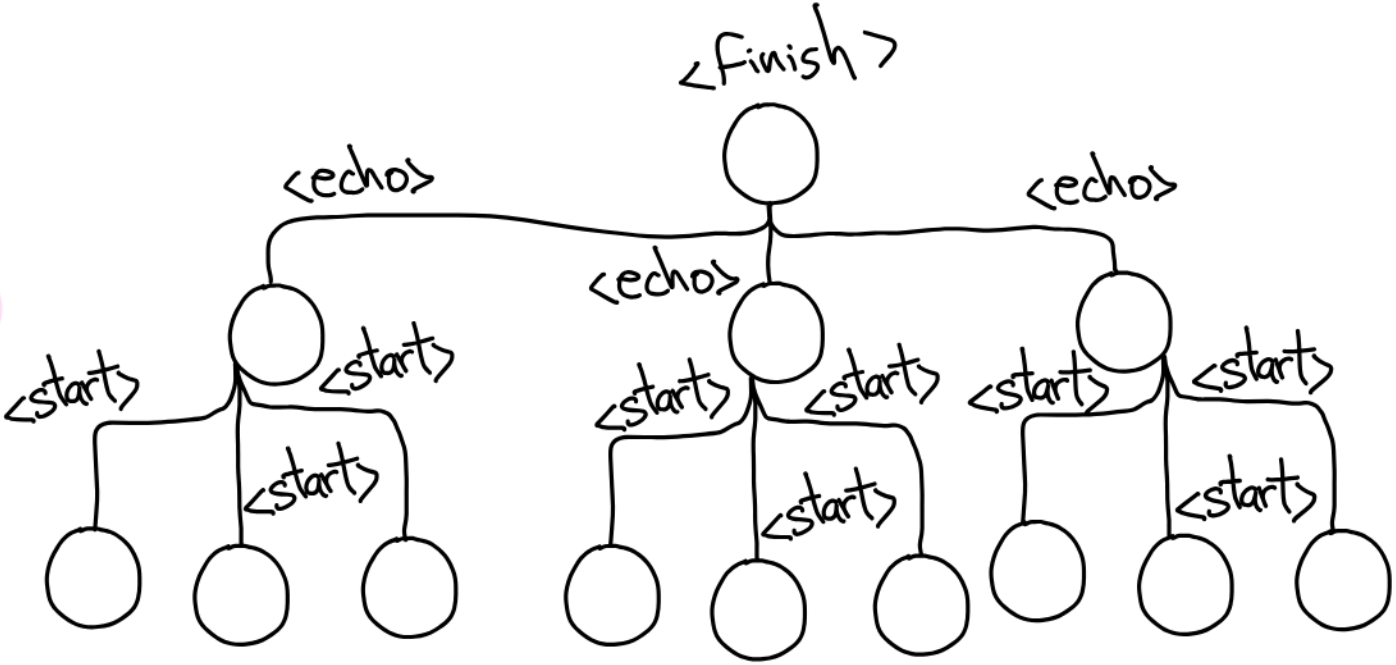


Messages Needed in an Algorithm

send <start>

upon quorum of <start>
    send <echo>

upon quorum of <echo>
    send <finish>

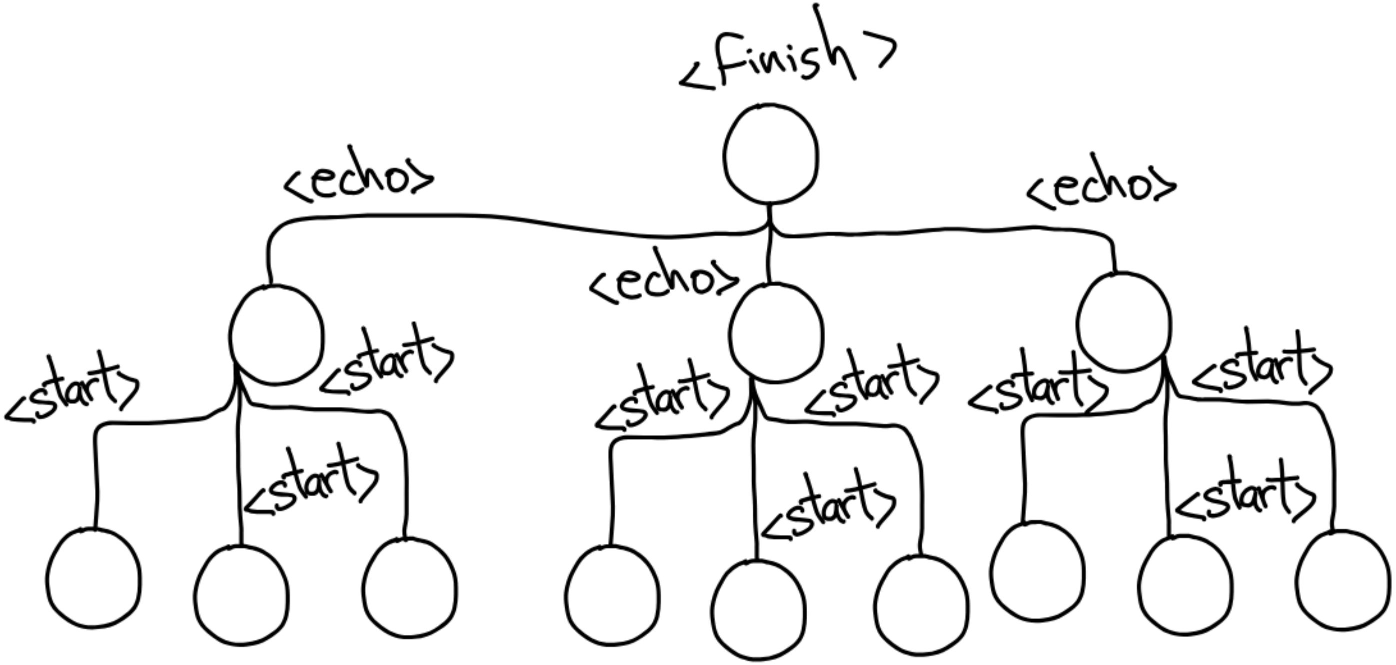


Messages Needed in an Algorithm

send <start>

upon quorum of <start>
 send <echo>

upon quorum of <echo>
 send <finish>

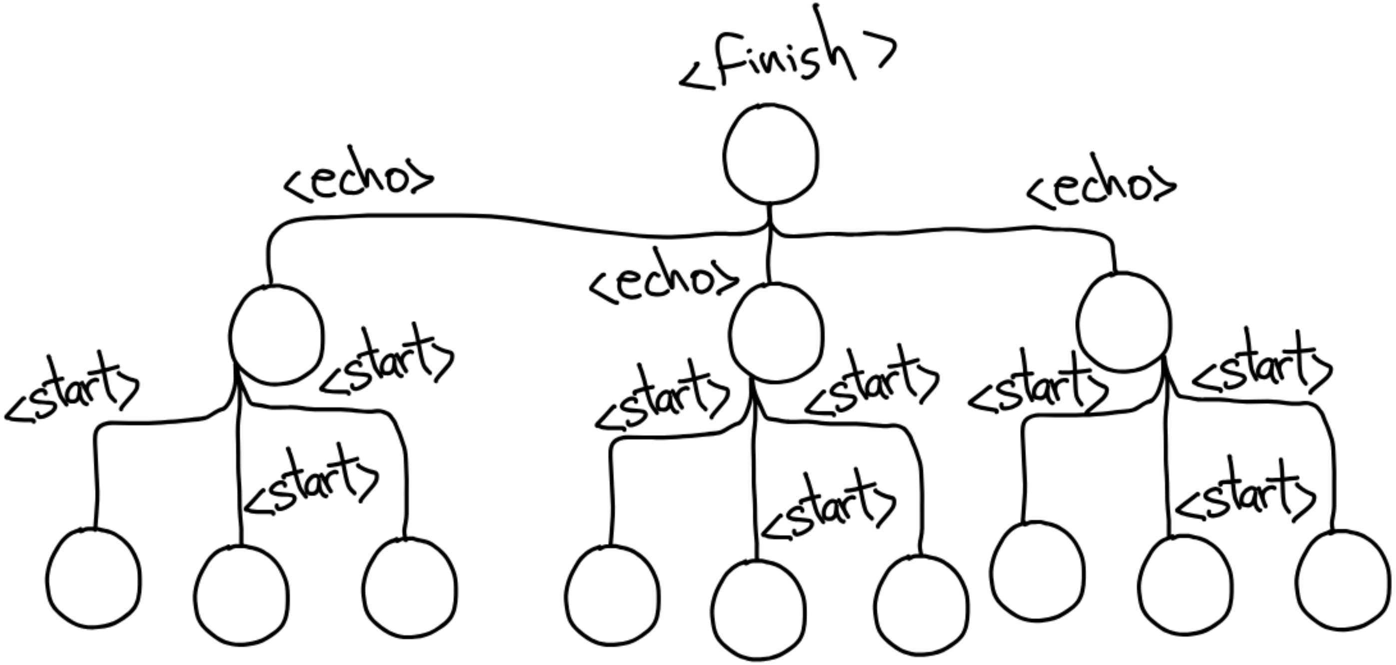


Messages Needed in an Algorithm

send <start>

upon quorum of <start>
send <echo>

upon quorum of <echo>
send <finish>



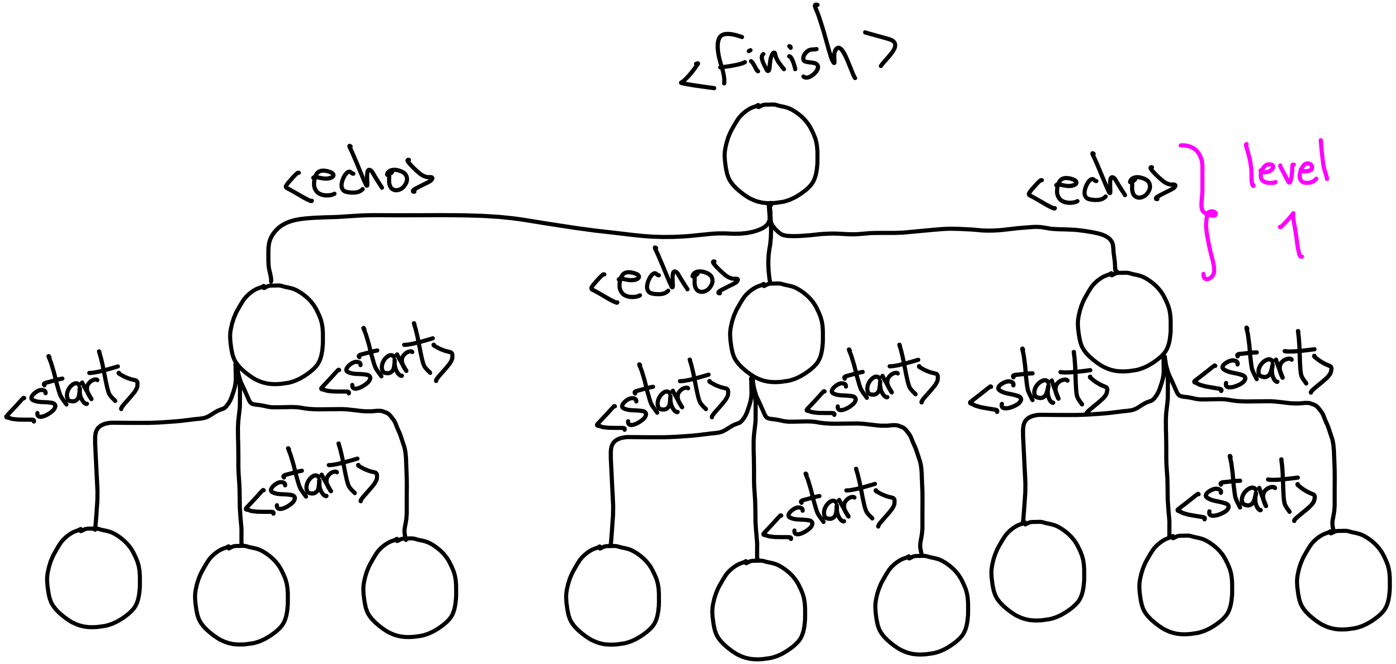
Requirement: the quorums of a quorum should send <start>

Messages Needed in an Algorithm

send <start>

upon quorum of <start>
send <echo>

upon quorum of <echo>
send <finish>

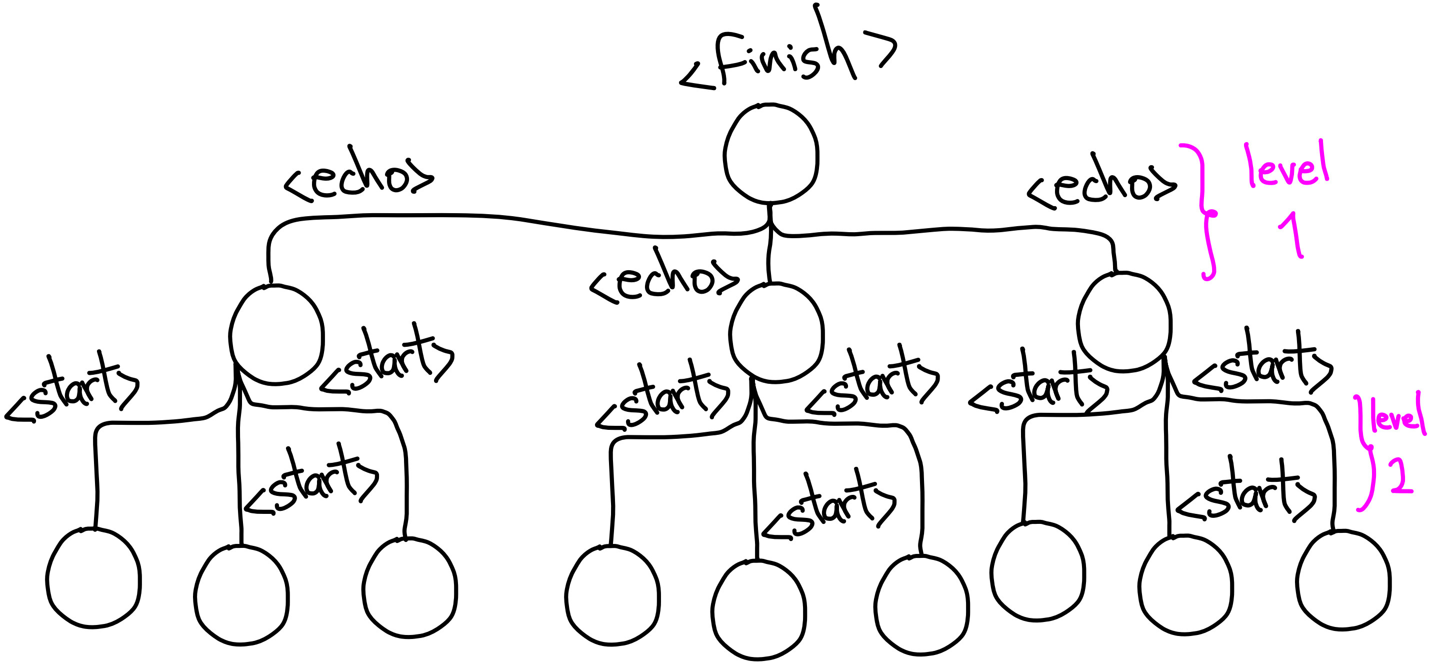


Messages Needed in an Algorithm

send <start>

upon quorum of <start>
    send <echo>

upon quorum of <echo>
    send <finish>



Depth of a Process

- A process has depth 0 if it is correct
- Furthermore, a process has depth d if it is correct and it has a quorum such that all its members have depth $\geq d-1$

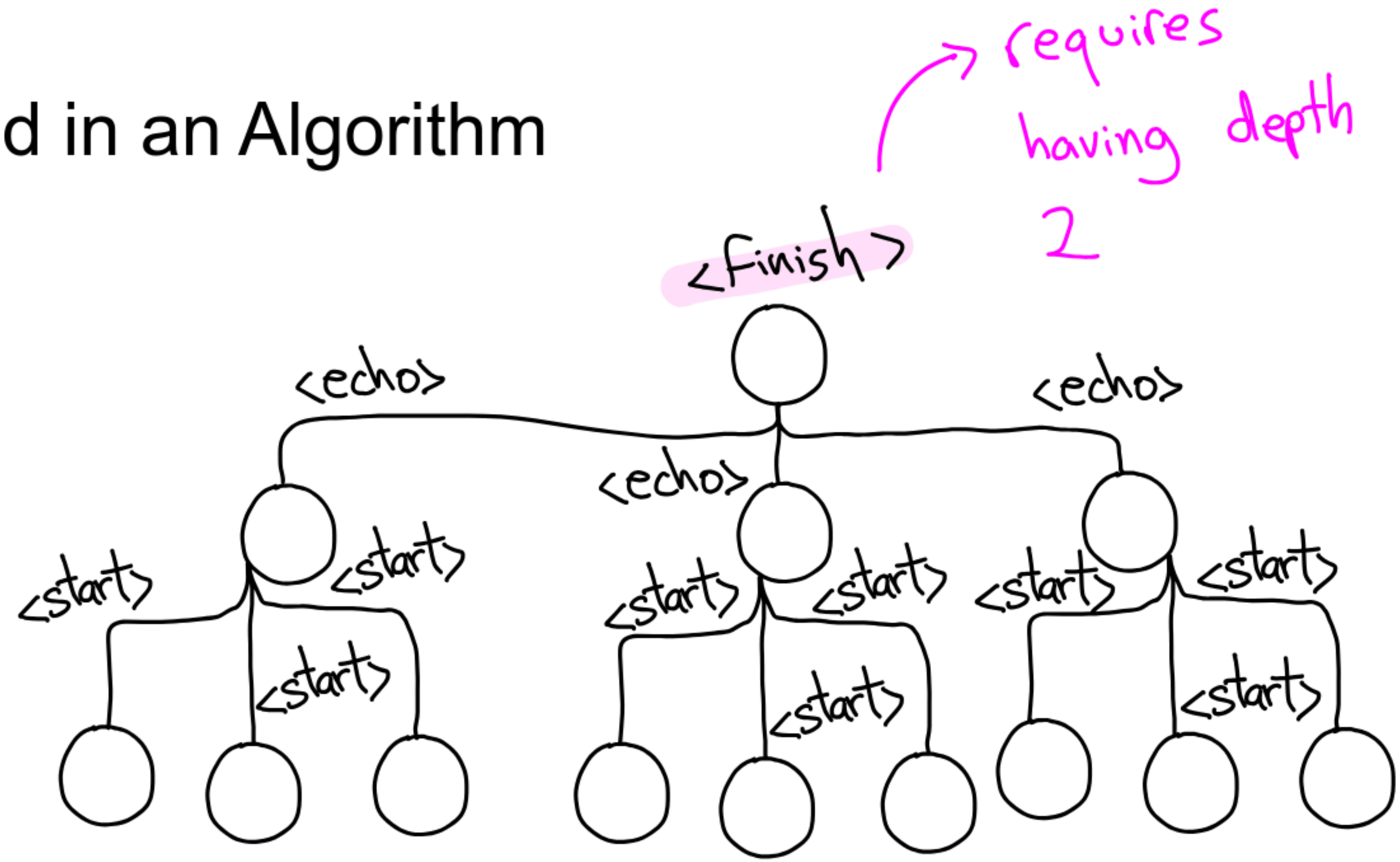
We focus only on the maximal depth of a process

Messages Needed in an Algorithm

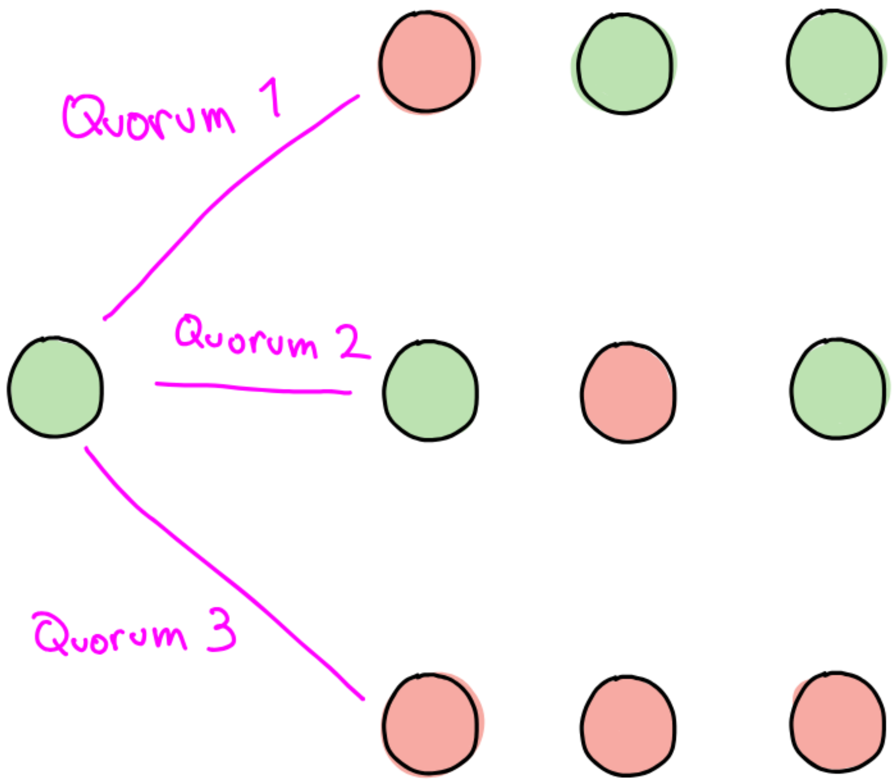
send <start>

upon quorum of <start>
    send <echo>

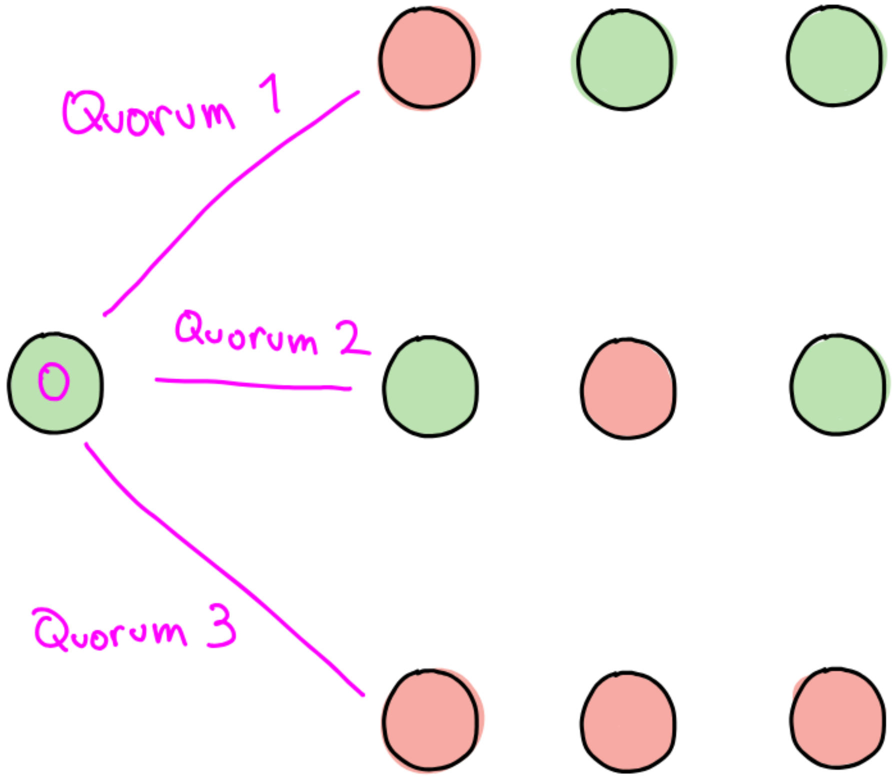
upon quorum of <echo>
    send <finish>



Depth of a Process



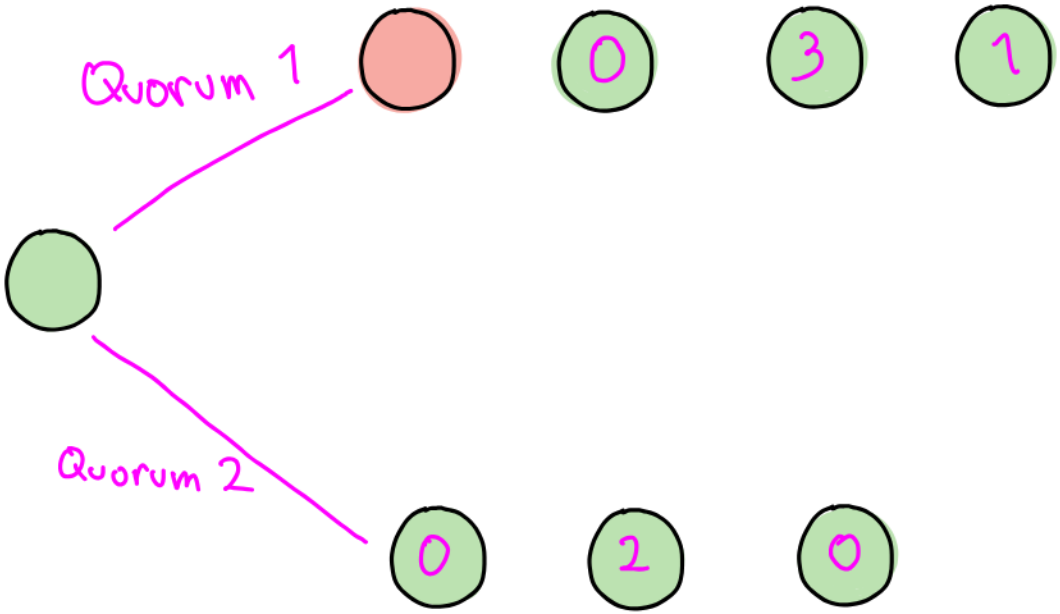
Depth of a Process



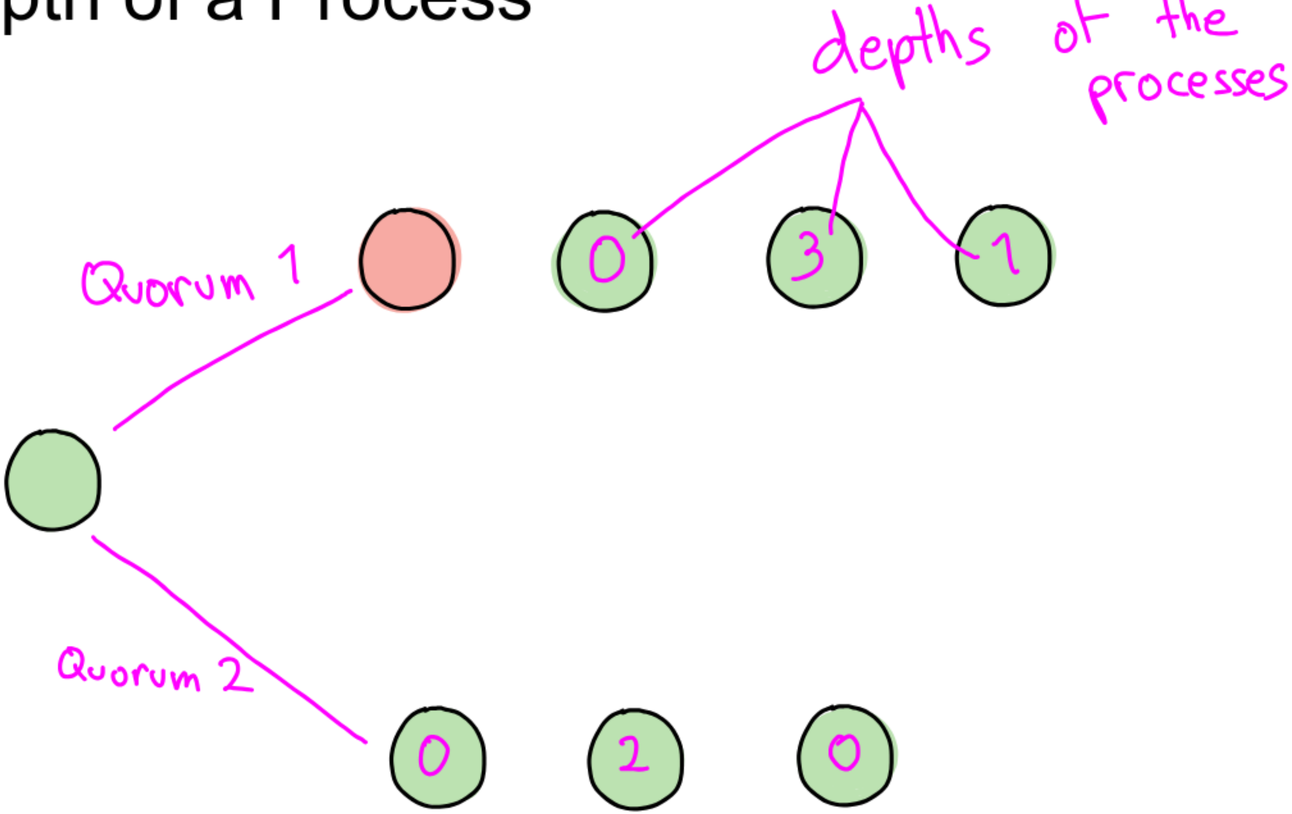
All quorums have at least one faulty process

Depth = 0 (naive)

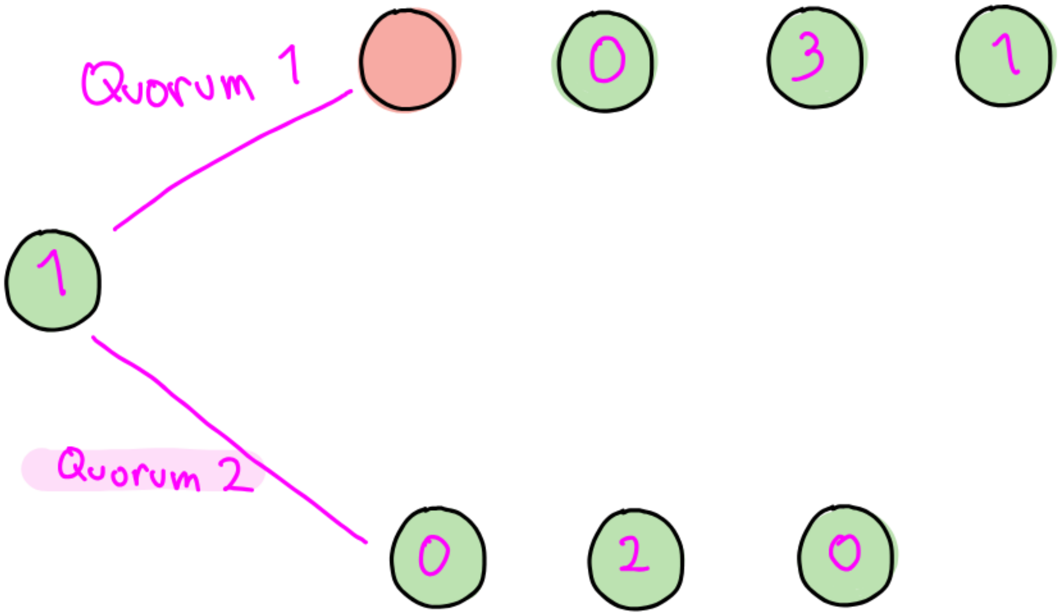
Depth of a Process



Depth of a Process



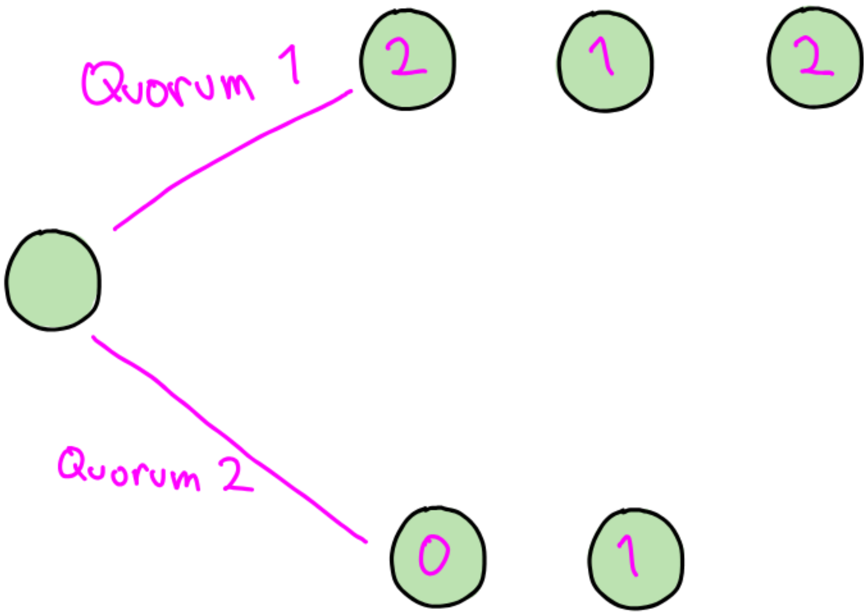
Depth of a Process



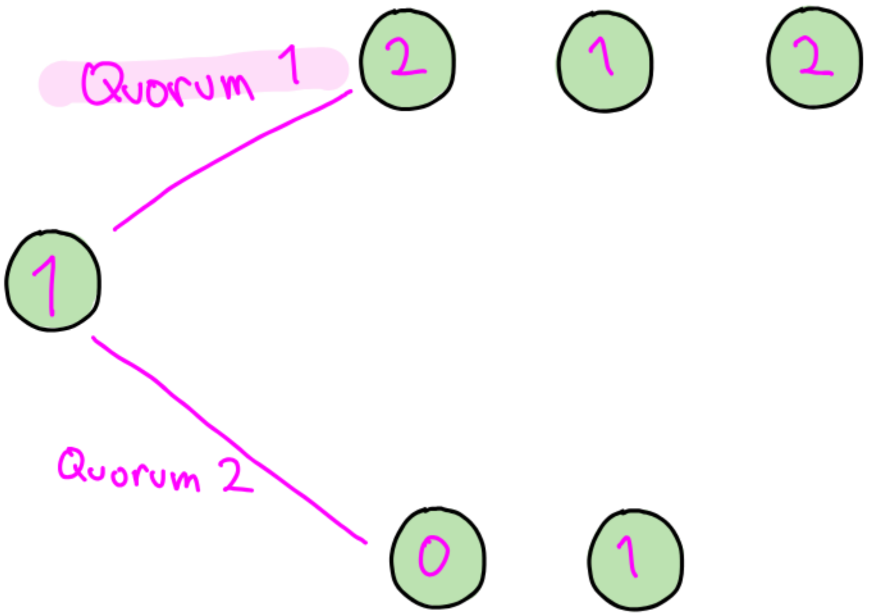
One quorum where all processes have depth ≥ 0

Depth = 1

Depth of a Process



Depth of a Process



One quorum where all processes have depth ≥ 1

Depth = 1

Depth in the Asymmetric Model

Depth generalizes the existing classification of processes

Depth

0

1

2

.

.

.

∞

Depth in the Asymmetric Model

Depth generalizes the existing classification of processes

Depth

0 $\longrightarrow$ naive

1

2

.

.

.

∞

Depth in the Asymmetric Model

Depth generalizes the existing classification of processes

Depth

0 → naive

1

2

.

.

.

∞ → guild member

Depth in the Asymmetric Model

Depth generalizes the existing classification of processes

Depth

0 → naive

1

2

.

.

.

∞ → guild member

wise

much more expressivity!

Solving Problems with Depth

We must adapt asymmetric problems to use depth

If we can solve problems for all processes with depth $< \infty$ then we don't need a guild

Reliable Broadcast with Depth (RB[d])

We adapt the properties of problems to use depth

Consistency: if a process with depth d delivers m and another process with depth d delivers m’, then m=m’

Totality: if a process with depth d delivers a message, then all processes with depth d will deliver a message

Reliable Broadcast with Depth (RB[d])

[ACTZ24] algorithm solves RB[∞] (i.e., for guild members)

It is not possible to solve RB[d] for d ≤ 1

We want to solve RB[d] for the minimum possible d

Reliable Broadcast with Guilds [ACTZ24] (simplified)

Bracha broadcast with asymmetric quorums

Kernel: set of processes that intersects all my quorums

…

upon quorum of <ready, m>
    deliver m

upon kernel of <ready, m>
    send <ready, m>

…

Reliable Broadcast with Guilds [ACTZ24] (simplified)

Bracha broadcast with asymmetric quorums

Kernel: set of processes that intersects all my quorums

…

upon quorum of <ready, m>
    deliver m

upon kernel of <ready, m>
    send <ready, m>
…

guorantees that if one guild member delivers m, all guild members will deliver m

Reliable Broadcast with Guilds [ACTZ24] (simplified)

Bracha broadcast with asymmetric quorums

Kernel: set of processes that intersects all my quorums

…

upon quorum of <ready, m>
    deliver m

upon kernel of <ready, m>
    send <ready, m>

…

Kernel contains at least one guild member, so m will always be a good value

Reliable Broadcast with Guilds [ACTZ24] (simplified)

Bracha broadcast with asymmetric quorums

Kernel: set of processes that intersects all my quorums

…

upon quorum of <ready, m>
    deliver m

what happens when there is no guild?

upon kernel of <ready, m>
    send <ready, m>

…

Reliable Broadcast with Guilds [ACTZ24] (simplified)

Bracha broadcast with asymmetric quorums

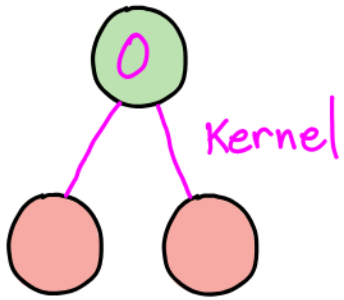
Kernel: set of processes that intersects all my quorums

…

upon quorum of <ready, m>
    deliver m

upon kernel of <ready, m>
    send <ready, m>
…

malicious processes
can trick a process
with depth 0



Reliable Broadcast with Guilds [ACTZ24] (simplified)

Bracha broadcast with asymmetric quorums

Kernel: set of processes that intersects all my quorums

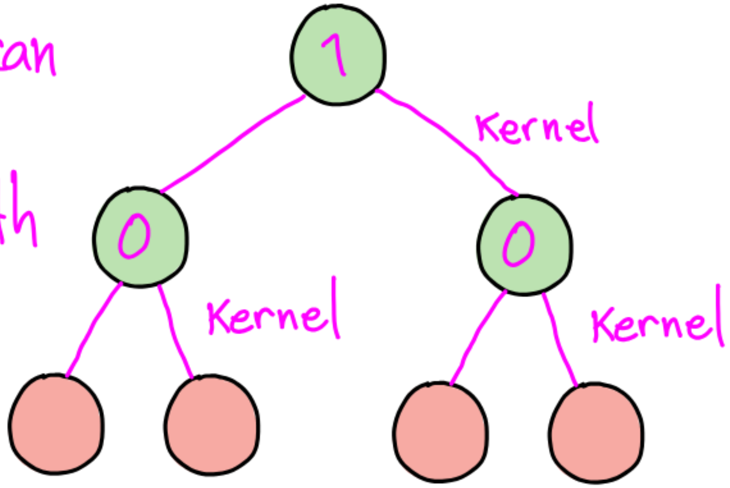
…

upon quorum of <ready, m>
    deliver m

upon kernel of <ready, m>
    send <ready, m>

…

and this can lead to tricking a depth 1 process



Reliable Broadcast with Guilds [ACTZ24] (simplified)

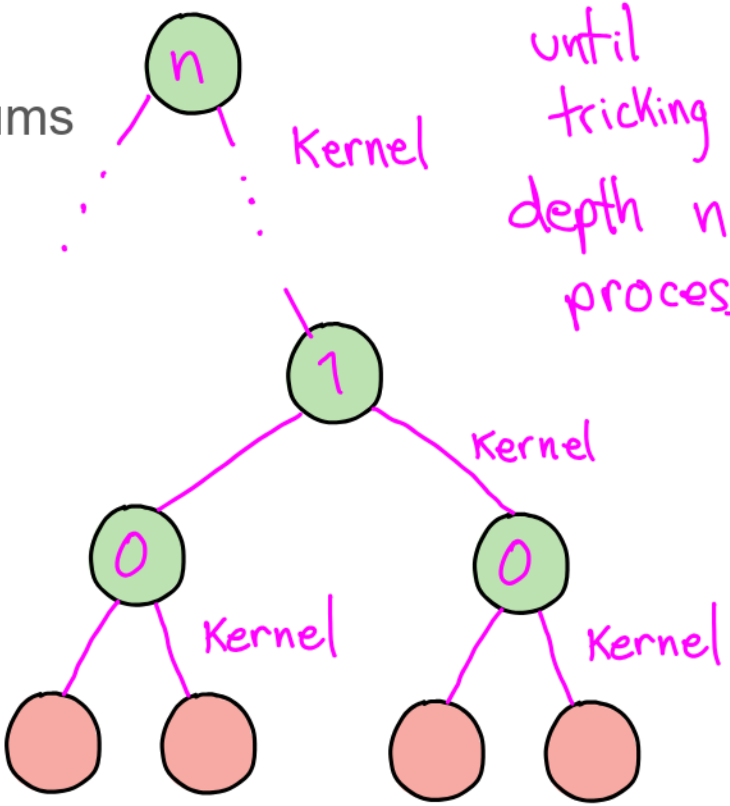
Bracha broadcast with asymmetric quorums

Kernel: set of processes that intersects all my quorums

…

upon quorum of <ready, m>
    deliver m

upon kernel of <ready, m>
    send <ready, m>
…



and continue until tricking a depth n process

An Algorithm for RB[3]

…

upon quorum of <ready, m>
    deliver m

upon kernel of <ready, m>
    send <ready-extra, m>

upon quorum of <ready-extra, m>
    send <ready, m>

…

An Algorithm for RB[3]

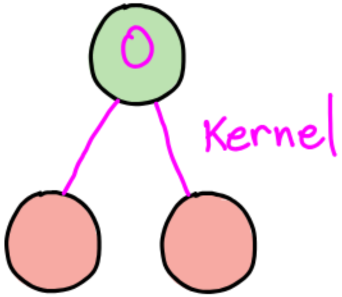
…

upon quorum of <ready, m>
    deliver m

upon kernel of <ready, m>
    send <ready-extra, m>

upon quorum of <ready-extra, m>
    send <ready, m>
…

malicious processes can still trick a depth 0 process



An Algorithm for RB[3]

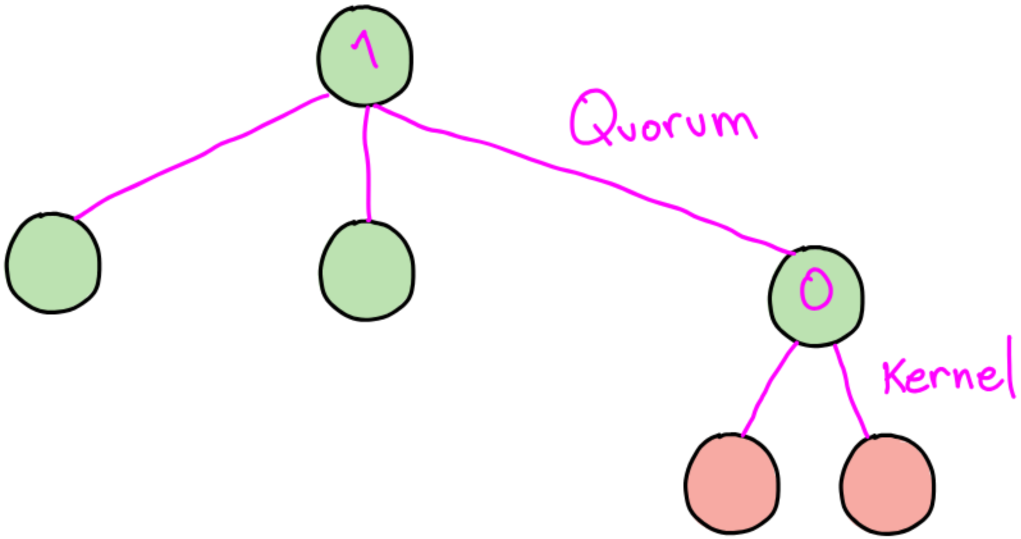
…

upon quorum of <ready, m>
    deliver m

upon kernel of <ready, m>
    send <ready-extra, m>

upon quorum of <ready-extra, m>
    send <ready, m>
…

now, to trick a depth 1 process, a quorum is needed



An Algorithm for RB[3]

…

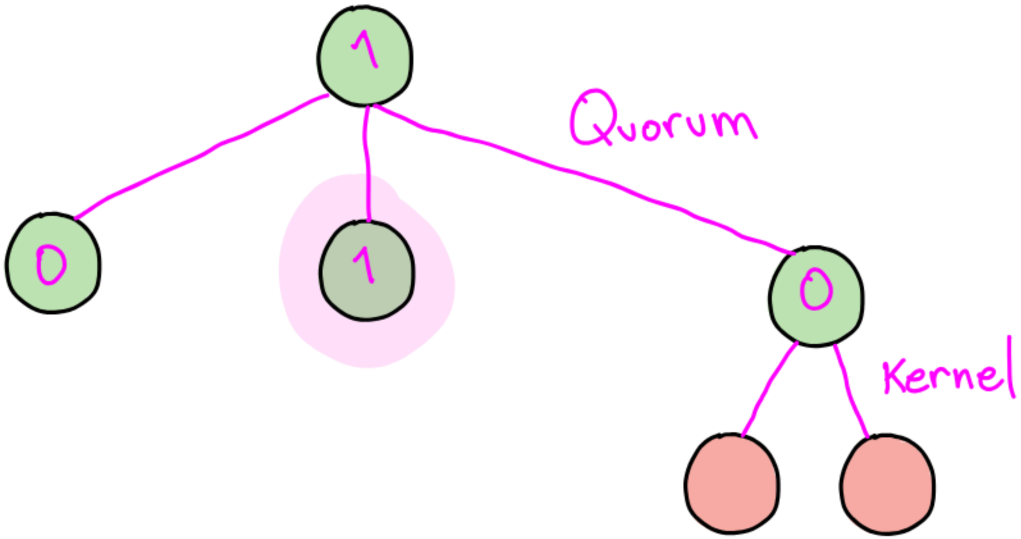
upon quorum of <ready, m>
    deliver m

upon kernel of <ready, m>
    send <ready-extra, m>

upon quorum of <ready-extra, m>
    send <ready, m>

…

By quorum intersection, can't be composed only by processes with depth 0



An Algorithm for RB[3]

We propose an algorithm that solves RB[3]

This reduces assumptions strength from depth ∞ to depth 3

Open question: is there an algorithm for RB[2]?

Consensus with Depth

Does consensus have bounded depth?

Main challenge: consensus can have executions with infinite length

Consensus with Depth

If consensus has infinite depth:

- Doesn't make sense to solve consensus in the asymmetric trust model (use equivalent symmetric algorithms instead)

If depth of consensus is bounded:

- An interesting algorithm
- Makes sense to use asymmetric trust for consensus

Conclusions

Guild assumptions are so strong that asymmetric trust becomes unnecessary

We propose using depth to characterize asymmetric problems

We solve reliable broadcast with weaker assumptions

$$RB[\infty] \rightarrow RB[3]$$

Future Work

Consensus with bounded depth

Determine the minimum depths that allow to solve reliable broadcast and eventually consensus

Do other asymmetric models suffer a similar *guild effect*?

References

[ABY22] Ittai Abraham, Naama Ben-David, and Sravya Yandamuri. Efficient and adaptively secure asynchronous binary agreement via binding crusader agreement. In PODC 2022.

[ACTZ24] Alpos, O., Cachin, C., Tackmann, B., & Zanolini, L. Asymmetric distributed trust. Distributed Computing, 37(3), 247-277.

[ACV25] Amores-Sesar, I., Cachin, C., & Villacis, J. Weaker Assumptions for Asymmetric Trust.

[LCL23] Li, X., Chan, E., & Lesani, M. Quorum subsumption for heterogeneous quorum systems. In DISC 2023.

[MR98] Malkhi, D., & Reiter, M. Byzantine quorum systems. Distributed computing, 11(4), 203-213.

[SC25a] Senn, M., & Cachin, C. Asymmetric Failure Assumptions for Reliable Distributed Systems. In PaPOC 2025.

[SC25b] Senn, M., & Cachin, C. Asymmetric Grid Quorum Systems for Heterogeneous Processes.

[Z23] Zanolini, L. Asymmetric Trust in Distributed Systems. PhD Thesis