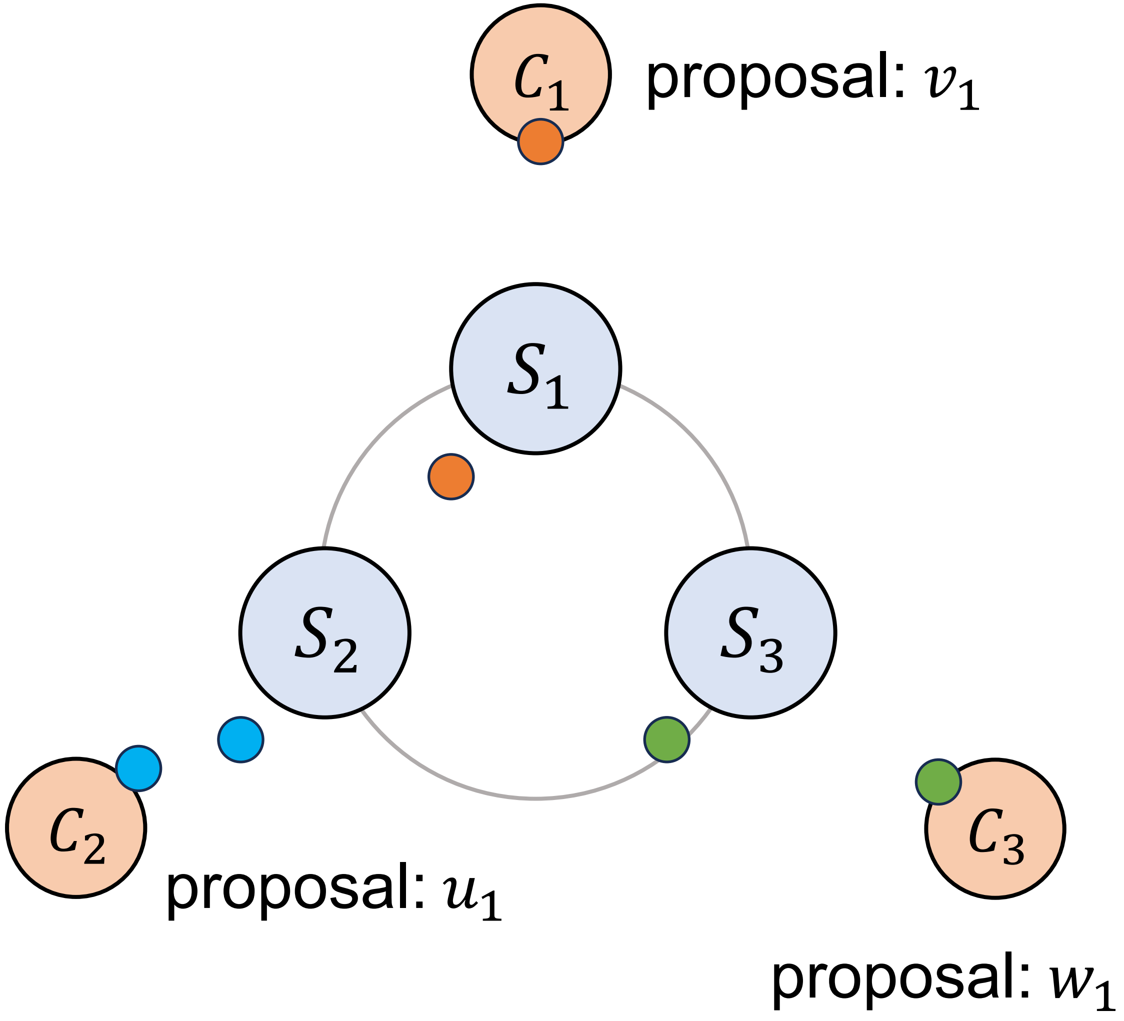
Cabinet: Dynamically Weighted Consensus Made Fast

Gengrui (Edward) Zhang, Ph.D.
Department of Electrical & Computer Engineering
Concordia University

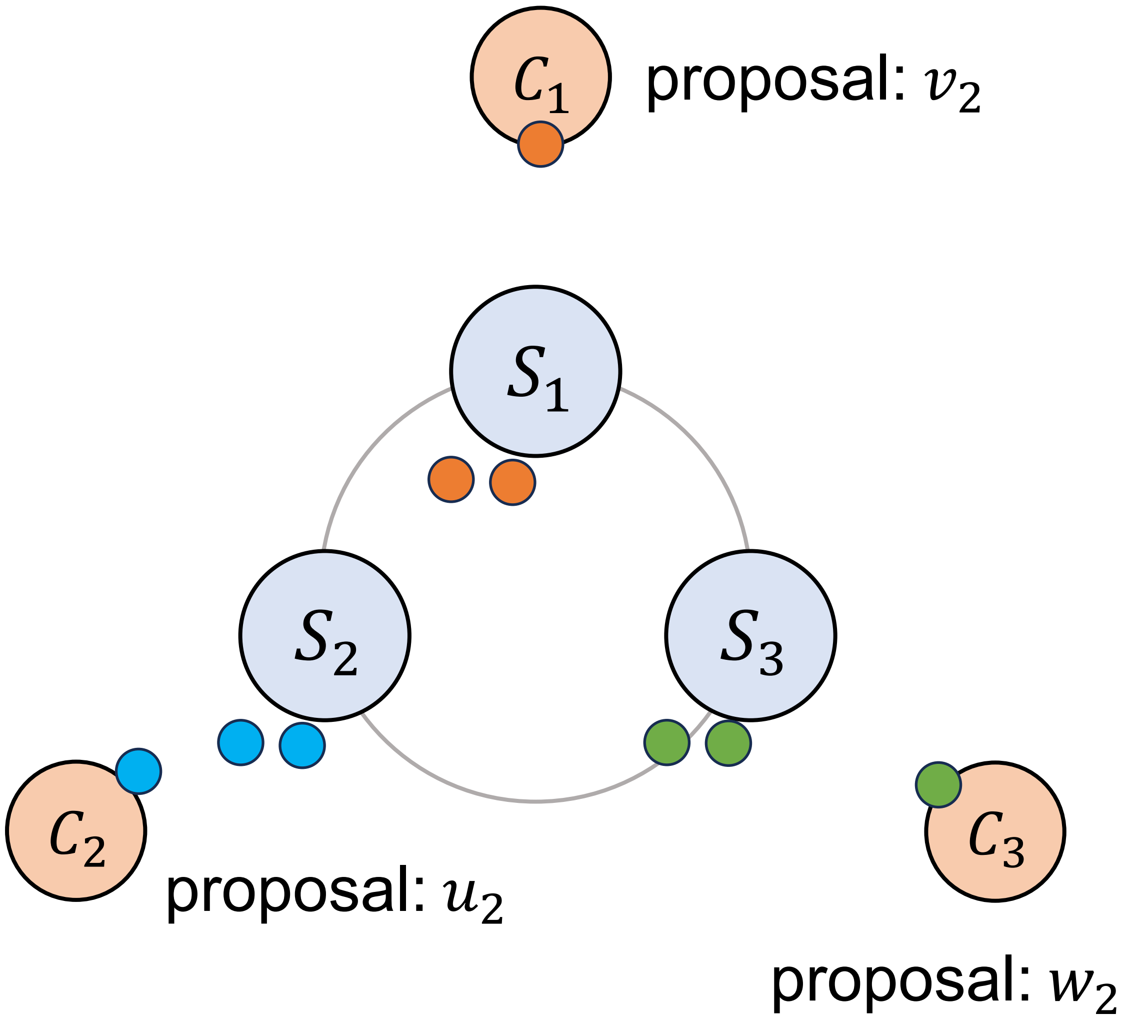
What is a consensus algorithm?

- Consensus algorithms reach agreement on committing values among all servers even if some servers experience failures



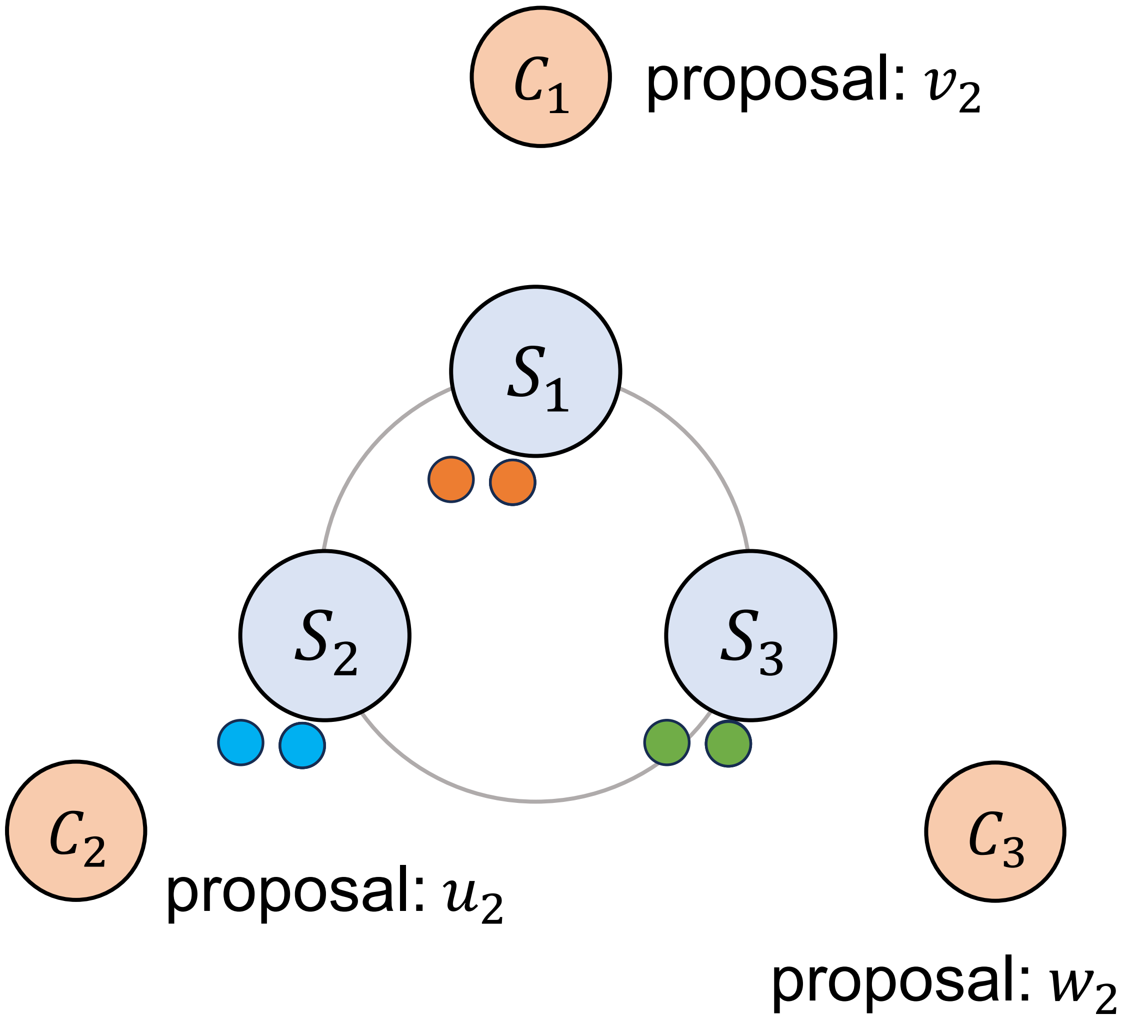
What is a consensus algorithm?

- Consensus algorithms reach agreement on committing values among all servers even if some servers experience failures



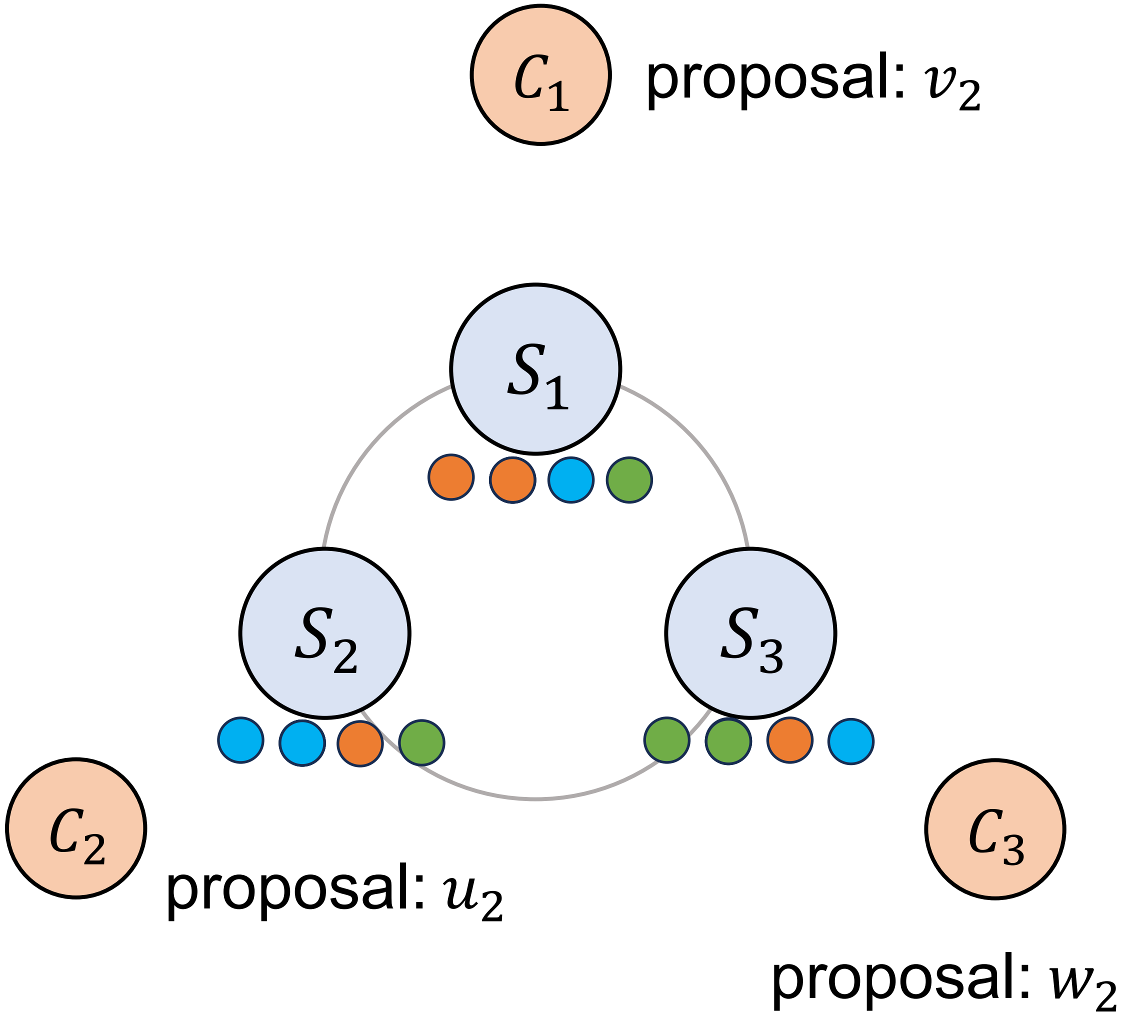
What is a consensus algorithm?

- Consensus algorithms reach agreement on committing values among all servers even if some servers experience failures



What is a consensus algorithm?

- Consensus algorithms reach agreement on committing values among all servers even if some servers experience failures

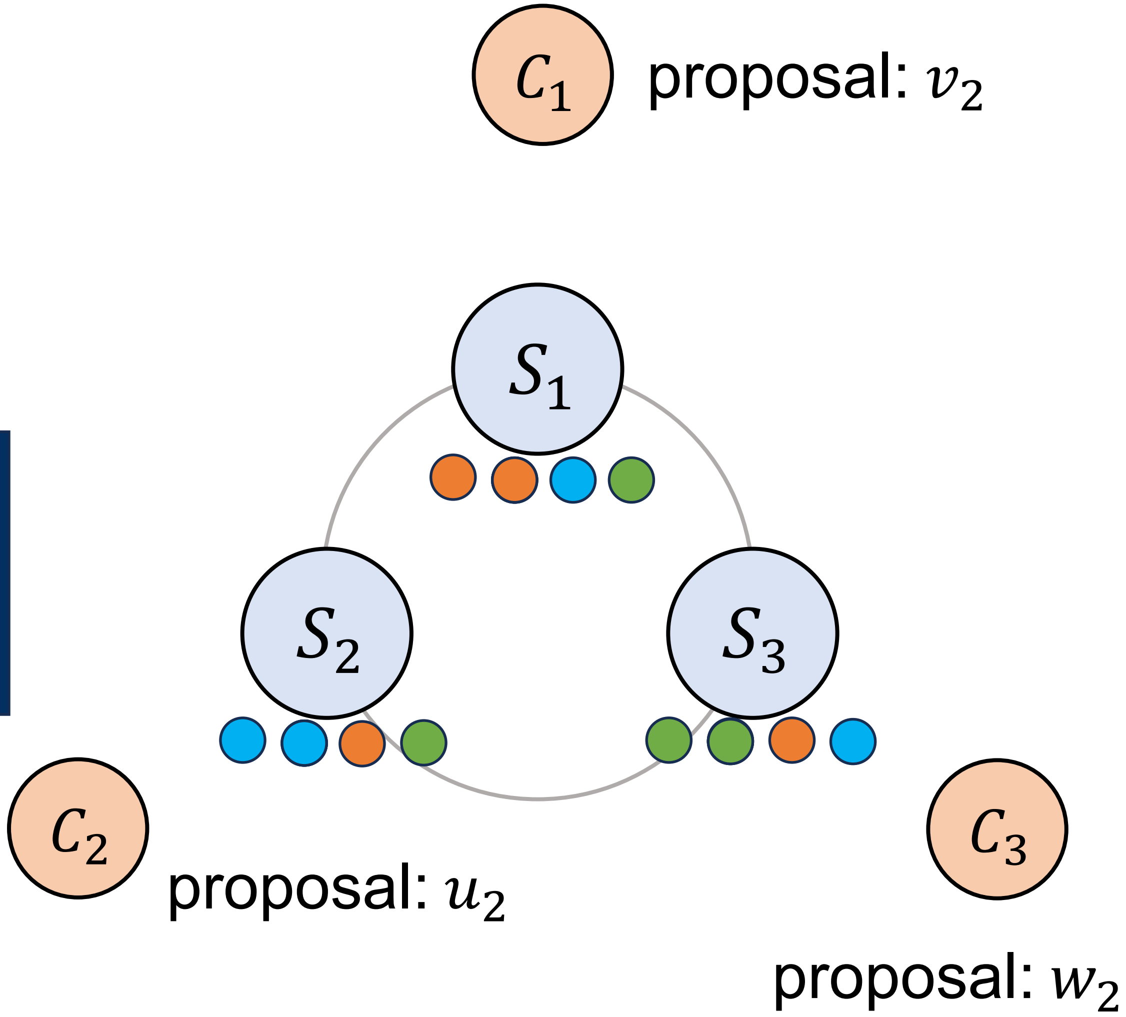


What is a consensus algorithm?

- Consensus algorithms reach agreement on committing values among all servers even if some servers experience failures

Committing the same value among all non-faulty servers at the same order

- Consensus enforces linearizability on servers, which is crucial for strong consistency applications
  - E.g., banking systems, blockchains

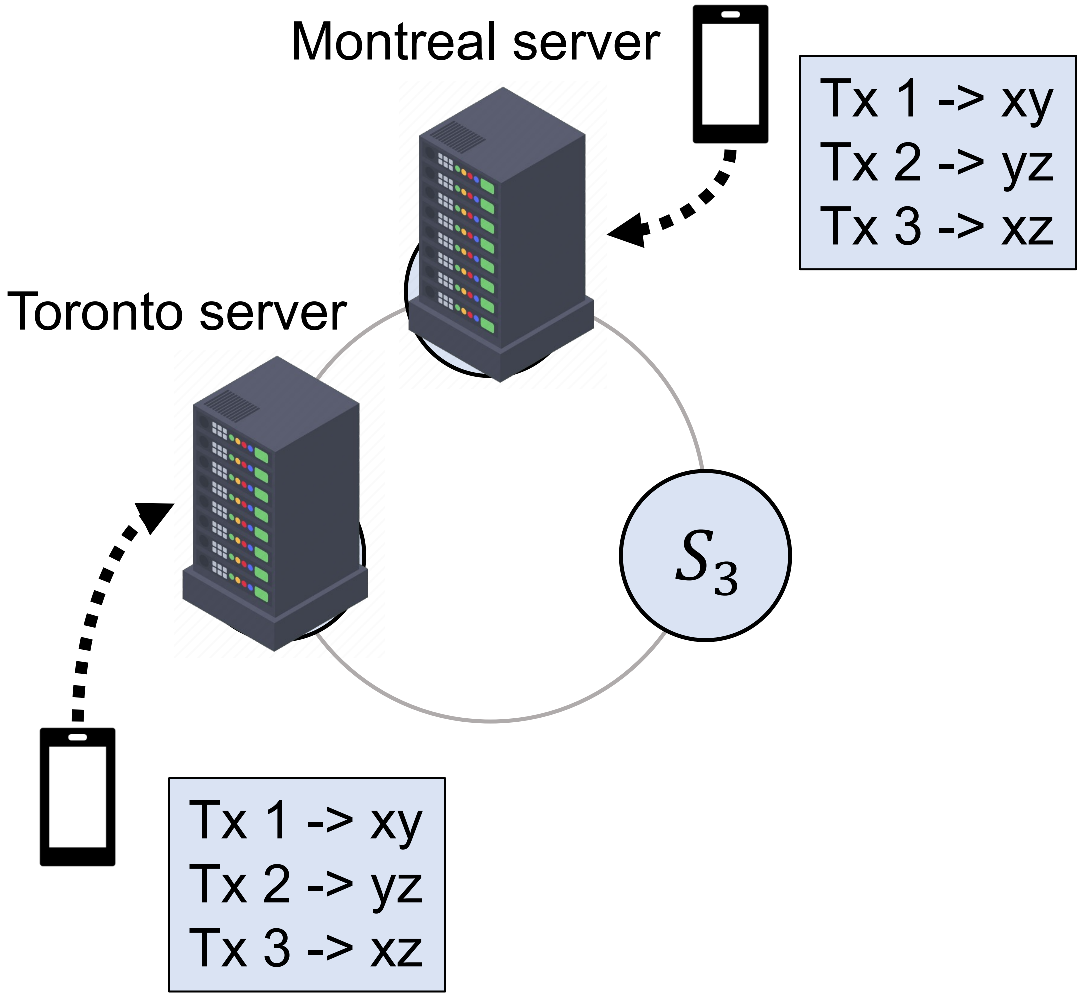


What is a consensus algorithm?

- Consensus algorithms reach agreement on committing values among all servers even if some servers experience failures

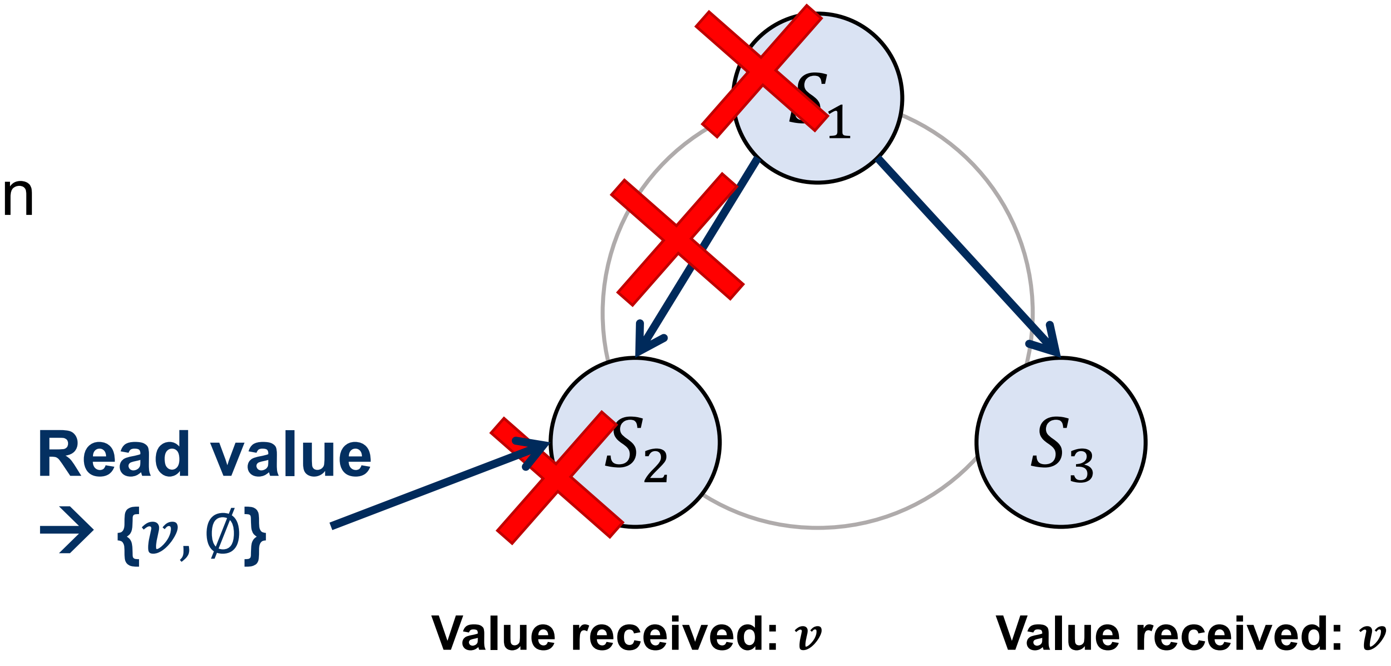
Committing the same value among all non-faulty servers at the same order

- Consensus enforces linearizability on servers, which is crucial for strong consistency applications
  - E.g., banking systems, blockchains



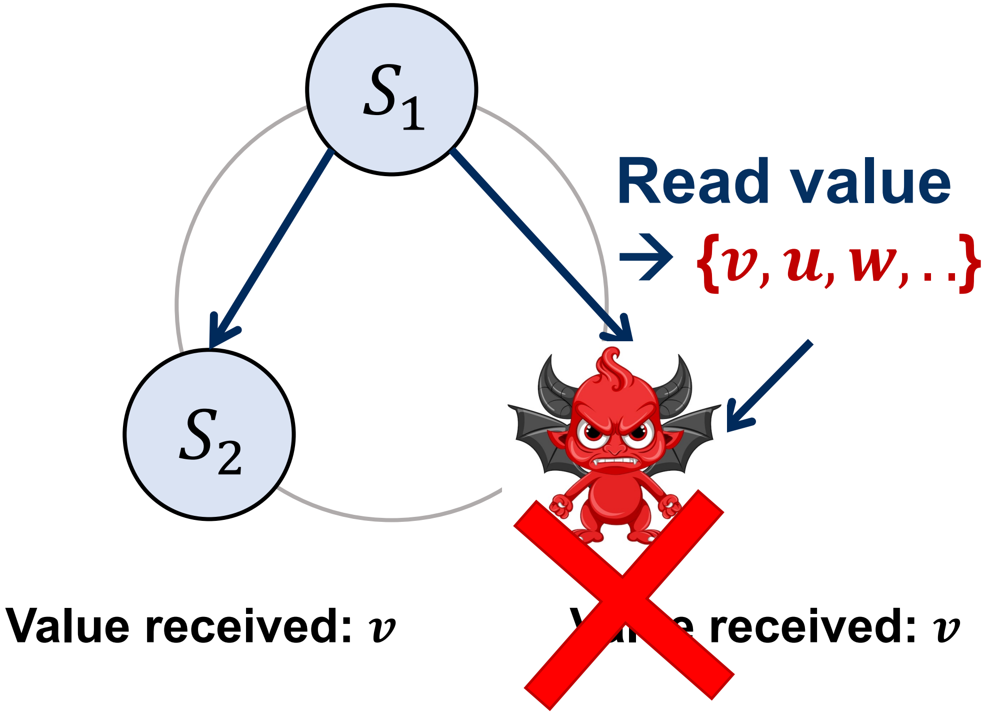
The worst case of benign failures is that the value is lost

- **Benign failures**
  - Crash
  - Send/Receive omission
  - Link failures
  - Timing failures



The worst case of Byzantine failures is any arbitrary behavior doing the most harm

- **Benign failures**
  - Crash
  - Send/Receive omission
  - Link failures
  - Timing failures
- **Byzantine failures**
  - Arbitrary behavior
  - Malicious attacks/value changes/protocol violation



A note on fault tolerance

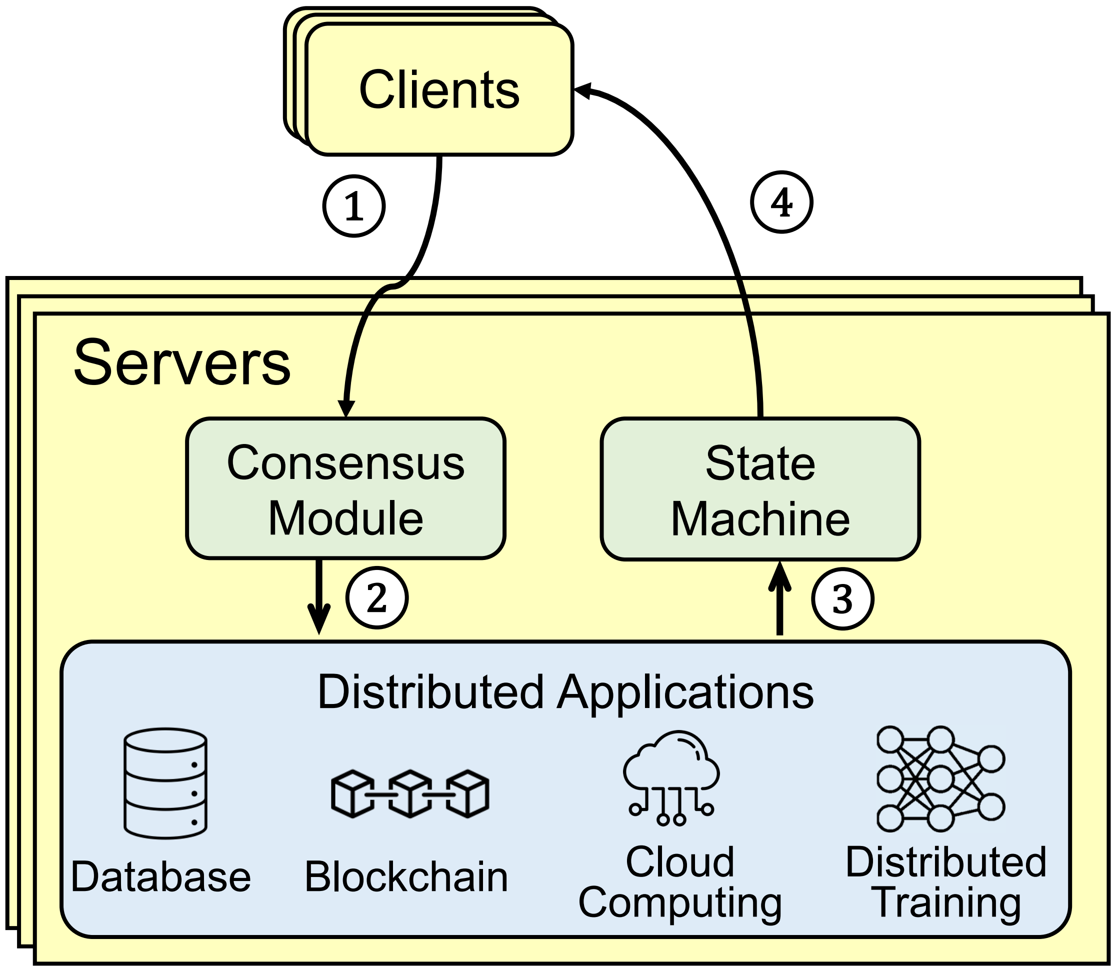
- Failures are mainly categorized into two groups:
  - **Benign failures**
    - Crash
    - Send/Receive omission
    - Link failures
    - Timing failures
  - **Byzantine failures**
    - Arbitrary behavior
    - Malicious attacks/value changes/protocol violation

- **CFT Exemplary algorithms**
  - Paxos, ViewStamped Replication, Raft [ATC'13]
- **Applications (everything distributed):**
  - File systems: HDFS and GFS
  - Databases: Google Spanner and etcd
  - Coordination: Chubby and Zookeeper

- **BFT Exemplary algorithms**
  - PBFT [OSDI'99], HotStuff [PODC'21], Pompe [OSDI'22]
- **Applications (safety critical):**
  - Unreliable hardware: Airplanes
  - Blockchains: Facebook Diem, Microsoft CCF

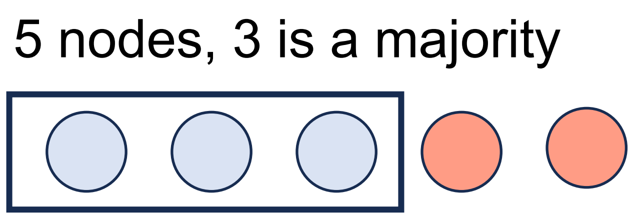
Ubiquitous consensus applications

- Consensus algorithms stand at the core of distributed applications
  - Transactions
  - Data consistency
  - Global state management
  - Coordination



Consensus: **How it started** and how's it going

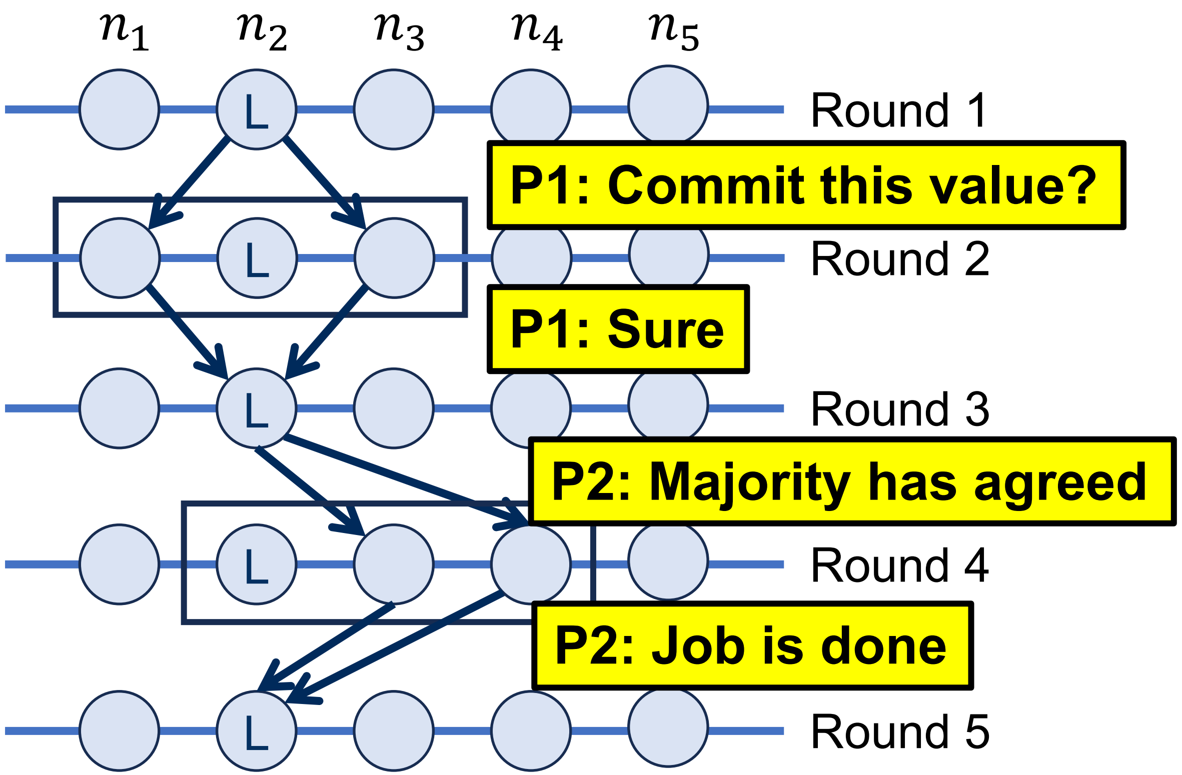
- How to reach consensus?
  - Majority quorums!
  - Every decision must be endorsed by a majority quorum
  - Tolerate $f = \frac{n-1}{2}$ failures



When $n = 5$,
- Quorum size $=\ 3$
- Tolerate 2 failures

Consensus: How it started and **how's it going**

- Modern consensus algorithms
  - Paxos [Lamport, TOCS'98]
    - Cheap Paxos [DSN'04], Mencius [OSDI'08], EPaxos [SOSP'13]
    - Google's **Spanner**, Apple's **FoundationDB**
  - Raft [Ongaro, ATC'13]
    - Strong leadership
    - **etcd** (Kubernetes) **CockroachDB** **MongoDB** (Raft variant)



Importance of consensus algorithms

Performance of consensus algorithms is curial to overall system performance;
e.g., distributed databases, blockchains, cloud computing, and training synchronization

However, state-of-the-art consensus algorithms are not efficient enough for modern computing applications

Features of modern computing (1)

1. System scales continue to grow
    - Netflix's Cassandra deployment (200+ nodes)
    - Google's Spanner scales to around 200 nodes

The automated tooling that Netflix has developed lets us quickly deploy large scale Cassandra clusters, in this case a few clicks on a web page and about an hour to go from nothing to a very large Cassandra cluster consisting of 288 medium sized instances, with 96 instances in each of three EC2 availability zones in the US-East region. Using an additional 60 instances as clients running the stress program we ran a workload of 1.1 million client writes per second. Data was automatically replicated across all three zones making a total of 3.3 million writes per second across the cluster.
able to complete within two hours with a total cost of a
and these EC2 instances were only in existence for the
There was no setup time, no discussions with IT operat
space and no more cost once the test was over.

Spanner: Google's Globally Distributed Database

Table IV. Two-Phase Commit Scalability. Mean and Standard Deviations over 10 Runs

participants	latency (ms)	
	mean	99th percentile
1	14.6 ± 0.2	26.550 ± 6.2
2	20.7 ± 0.4	31.958 ± 4.1
5	23.9 ± 2.2	46.428 ± 8.0
10	22.8 ± 0.4	45.931 ± 4.2
25	26.0 ± 0.6	52.509 ± 4.3
50	33.8 ± 0.6	62.420 ± 7.1
100	55.9 ± 1.2	88.859 ± 5.9
200	122.5 ± 6.7	206.443 ± 15.8

Features of modern computing (2)

2. Systems are becoming increasingly heterogeneous
   - Strong nodes
   - Weak nodes

Strong nodes often compute, store, and respond faster then weak nodes

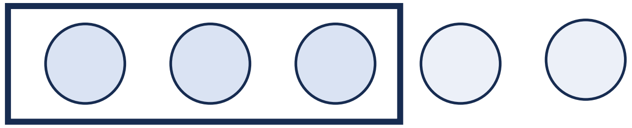
Strong nodes

Weak nodes

Under large-scale, especially heterogeneous clusters

Majority quorums become inefficient because of the quorum size required by each round

When $n = 5$, quorum size = 3, tolerating 2 failures



Strong nodes are compelled to wait for weak nodes, resulting in low throughput and high latency!

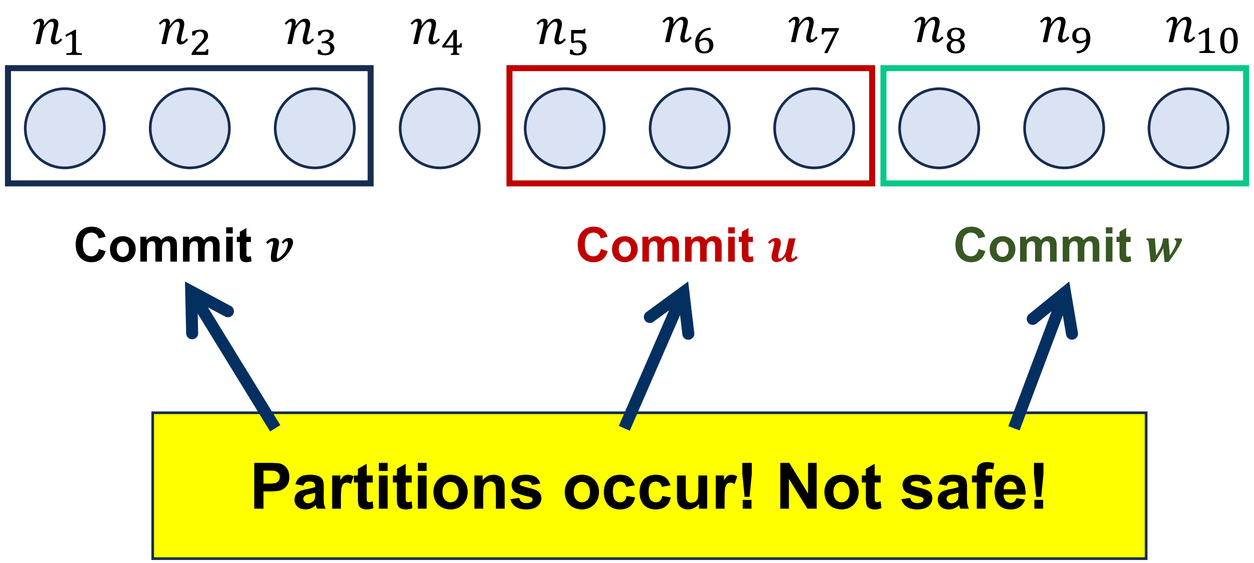
When $n = 100$, quorum size = 51, tolerating 49 failures

Do we need to tolerate 49 simultaneous failures by waiting for 51 replies each time?

Can we reduce quorum size? (First attempt)

- Attempt 1: Just collect fewer replies in a round

Let's set quorum size to 3



Safety
No two correct nodes decide differently

Can we reduce quorum size? (Second attempt)

- Attempt 2: Adding weights to nodes, but is any weight scheme okay?

Say consensus is reached when total weights (TW) is greater than 20 (consensus threshold)?

	n_1	n_2	n_3	n_4	n_5	n_6	n_7	n_8	n_9	n_{10}
Weights:	1	2	3	4	5	6	7	8	9	10

Safety
No two correct nodes decide differently

Group 1: n_1, n_2, n_9, n_{10}

$\sum_{i=1,2,9,10} w_i = 22 > 20$, **decide on v**

Group 2: n_6, n_7, n_8

$\sum_{i=6,7,8} w_i = 21 > 20$, **decide on w**

Partitions still occur ☹!
Not safe!

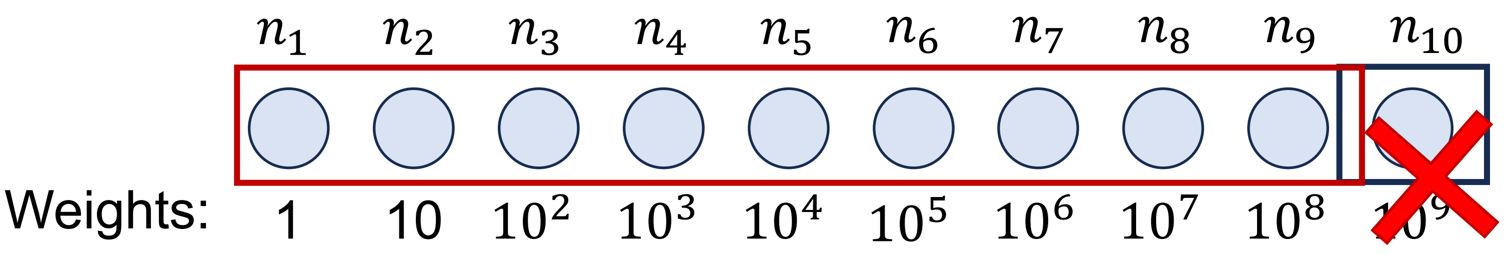
Motivation

M1: How to design a weight scheme (WS) and set a consensus threshold that can guarantee safety?
I.e., relation of WS and consensus threshold (CT)

Can we reduce quorum size? (Third attempt)

- Attempt 3: Let's have a larger value for consensus threshold

Say consensus is reached when total weights (TW) is greater than $\frac{10^9}{2}$ (consensus threshold)?



$\sum_{i=1\ to\ 9} w_i < \frac{10^9}{2}$, **cannot make a decision!**

n_{10} is the only node that can make a decision

One failure can stop the whole system ☹!
Not live!

Safety
No two correct nodes decide differently

Liveness
Nodes eventually decide

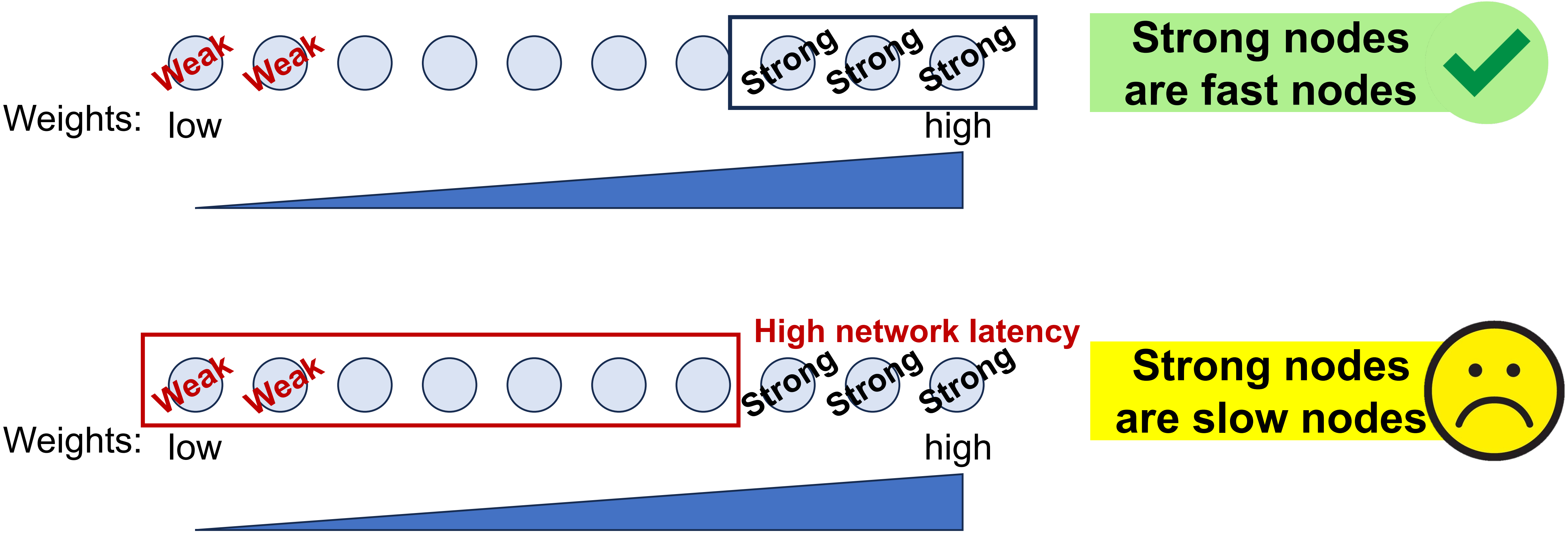
Motivation

M1: How to design a weight scheme (WS) and set a consensus threshold that can guarantee safety?
I.e., relation of WS and consensus threshold (CT)

M2: How to design a weight scheme (WS) and set a consensus threshold that can guarantee liveness?
I.e., relation of WS and fault tolerance

Operating conditions may change

- When high weight nodes become slow, weighted consensus may become worse than majority quorum consensus



Motivation

M1: How to design a weight scheme (WS) and set a consensus threshold that can guarantee safety?
I.e., relation of WS and consensus threshold (CT)

M2: How to design a weight scheme (WS) and set a consensus threshold that can guarantee liveness?
I.e., relation of WS and fault tolerance

M3: How to make a weight scheme (WS) adaptive to ongoing conditions such that system stays at optimal performance?
I.e., dynamic reassignment of weights

Motivation

The Cabinet

Established in Article II, Section 2 of the Constitution, the Cabinet's role is to advise the President on any subject he may require relating to the duties of each member's respective office.

Cabinet:

Dynamically Weighted Consensus Made Fast

scheme (WS) adaptive to ongoing

n stays at optimal performance?

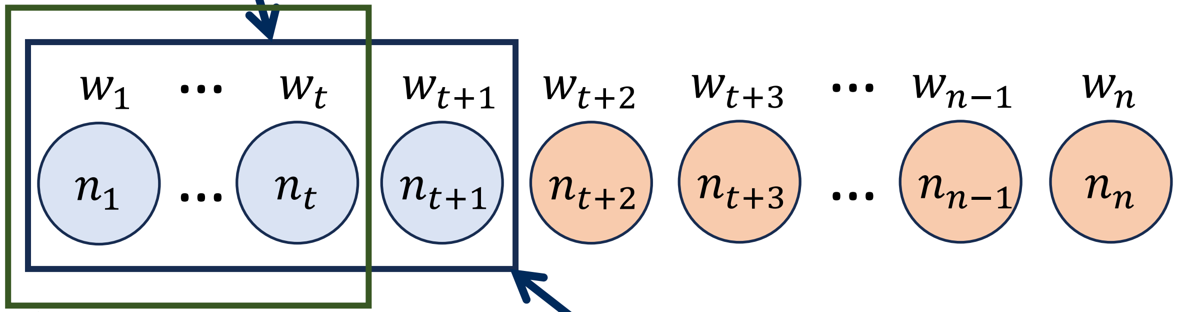
ssignment of weights

One-size-fits-all weight scheme

- A configurable failure threshold, t $(1 \le t \le \left\lfloor \frac{n-1}{2} \right\rfloor)$
  - Tolerate at least t failures with a quorum size of $t + 1$

Cabinet members: the $t + 1$ nodes with the top $t + 1$ highest weights

$$CT = \sum_{i=0}^{n} \frac{w_i}{2}$$



Invariant 2: $\sum_{i=1 \to t} w_i < CT$

Invariant 1: $\sum_{i=1 \to t+1} w_i > CT$

Safety
No two correct nodes decide differently

Liveness
Nodes eventually decide

Fast Agreement
System wide agreement can be made as soon as $t + 1$ nodes reach an agreement (e.g., $t = 5, n = 100$)

Quick proof of safety of majority of weights

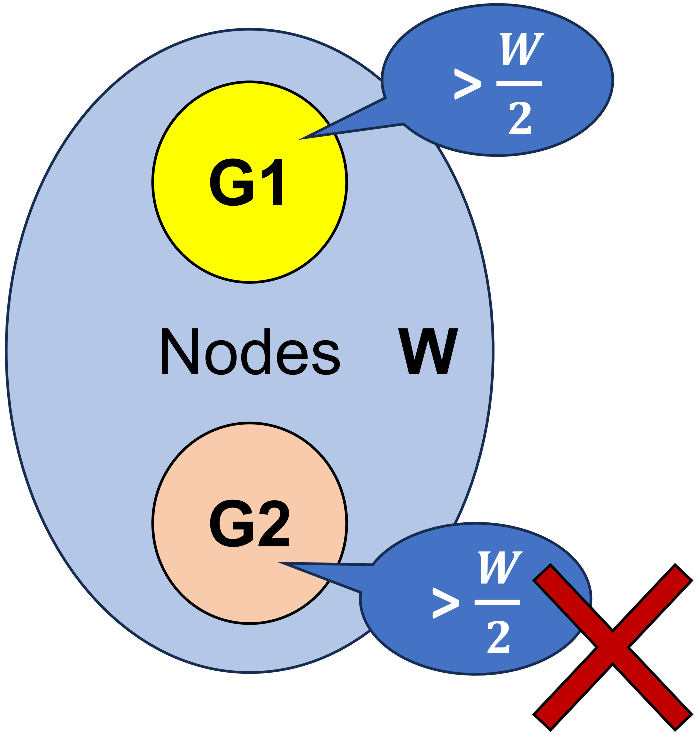
- Define consensus threshold (CT), which is half of the total weights

$$CT = \frac{1}{2}\sum_{i=1}^{n} w_i$$

- Say G_i is a group of nodes

Lemma

If the weight of G_i is greater than CT, no other group has a total weight greater than CT



Cabinet's implementation of weight scheme

- Cabinet uses geometric sequences to construct weight schemes

Weights:	w_1	$>$	w_2	$>$ ... w_{n-2}	$>$	w_{n-1}	$>$	w_n	
Geometric sequence:	$a_1 r^{n-1}$	$>$	$a_1 r^{n-2}$	... $a_1 r^2$	$>$	$a_1 r$	$>$	a_1	

$$CT = \sum_{i=0}^{n} \frac{w_i}{2}$$

Simple and effective!

Have your own weight scheme with n and t
(n: # of nodes; t: # of failures)

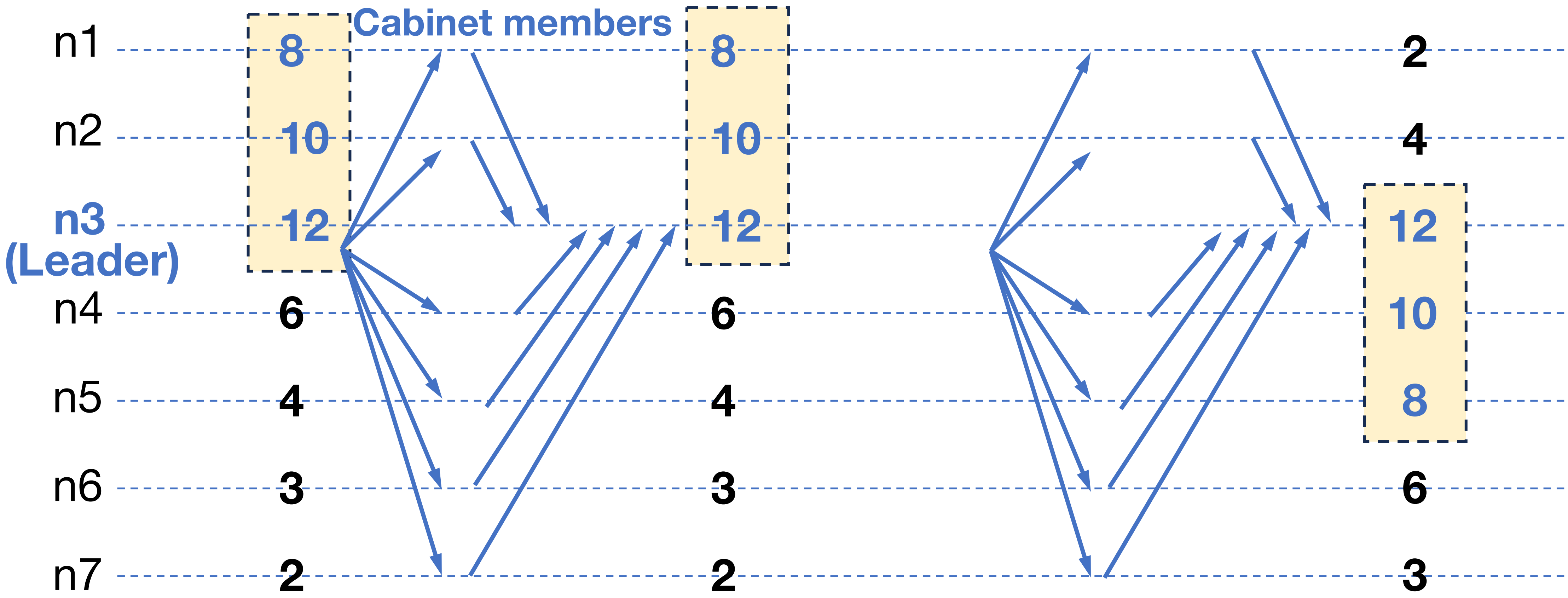
Invariant 2: $\sum_{i=1 \to t} w_i < CT$

Invariant 1: $\sum_{i=1 \to t+1} w_i > CT$

$$\sum_{i=n-t}^{n-1} a_1 r^i < CT = \frac{1}{2} \sum_{i=0}^{n-1} a_1 r^i < \sum_{i=n-t-1}^{n-1} a_1 r^i$$

$$\boxed{r^{n-t-1} < \frac{1}{2}(r^n + 1) < r^{n-t}}$$

Dynamic weight assignment (t=2)



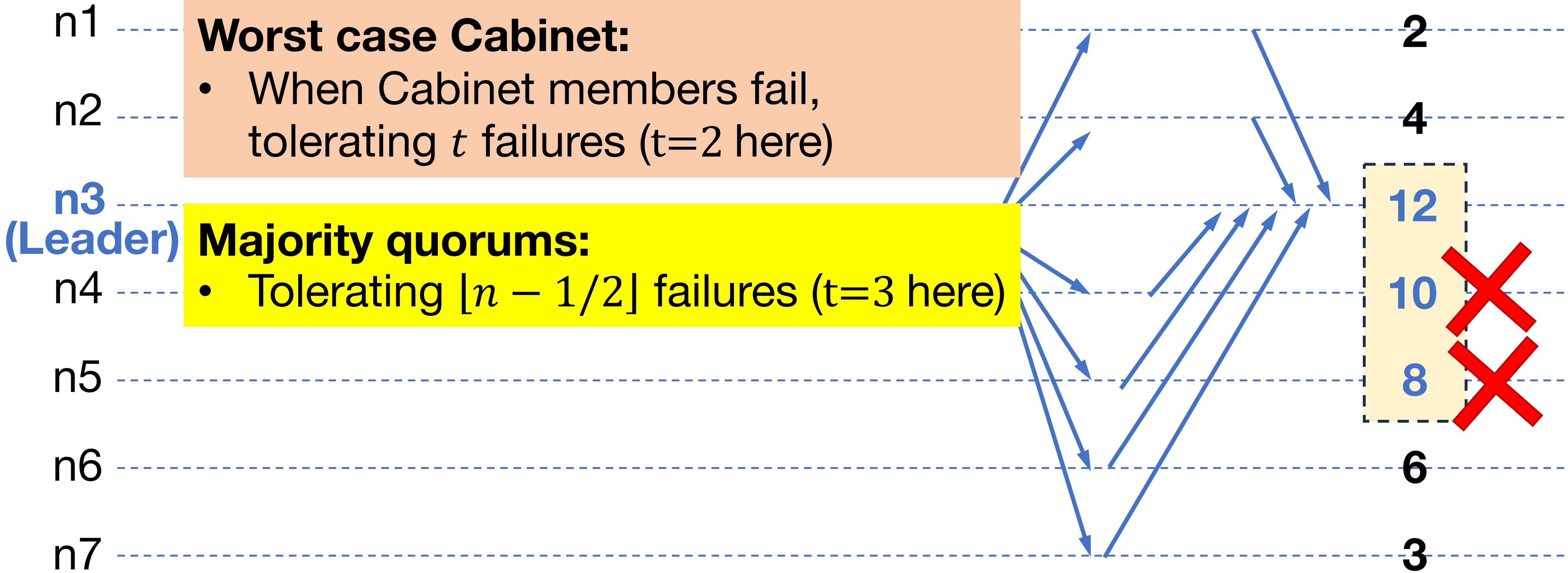
{n2, n1, n4, n5, n6, n7}

{n4, n5, n6, n2, n7, n1}

$$CT = \sum \frac{W}{2} = 22.5$$

The first $t+1$ replying nodes in round r become the cabinet members in round $r+1$

Feature 1: Tolerating more than t failures



Worst case Cabinet:

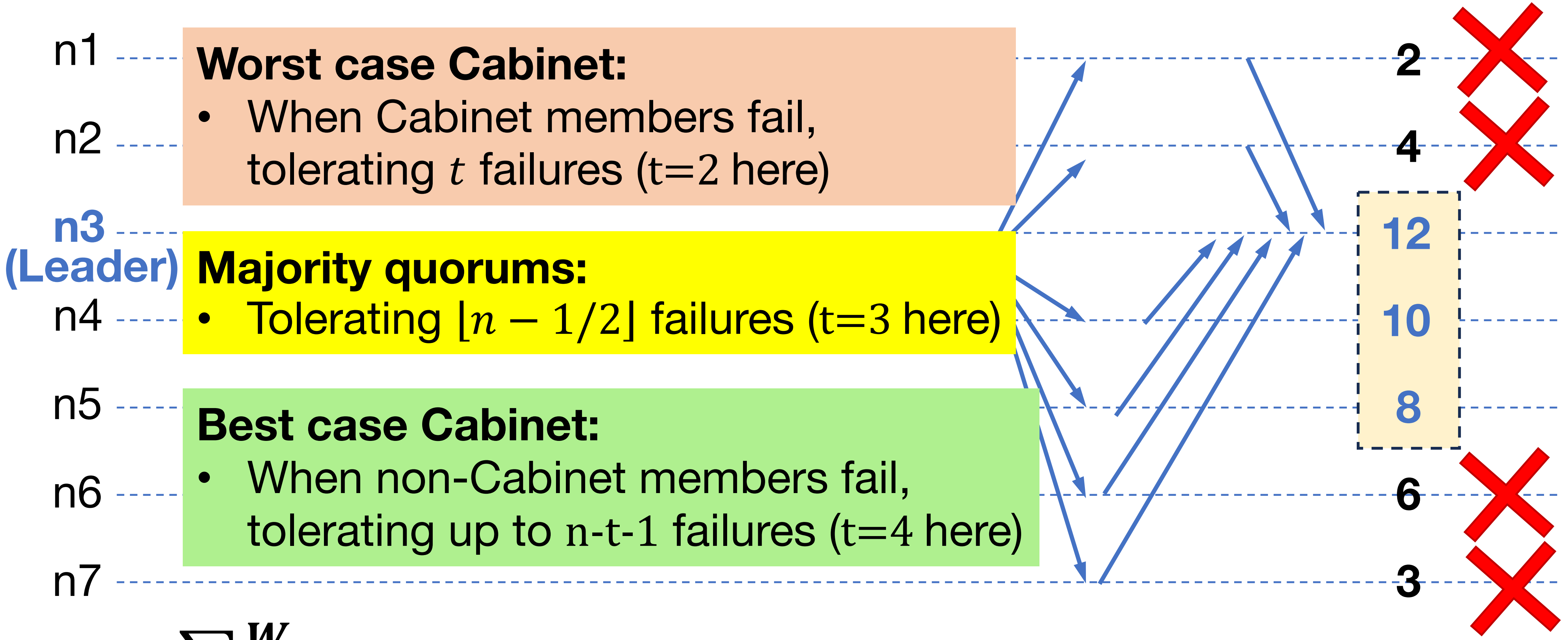
- When Cabinet members fail, tolerating t failures (t=2 here)

Majority quorums:

- Tolerating $\lfloor n-1/2 \rfloor$ failures (t=3 here)

$$CT = \sum \frac{W}{2} = 22.5$$

Feature 1: Tolerating more than t failures



Worst case Cabinet:

- When Cabinet members fail, tolerating t failures (t=2 here)

Majority quorums:

- Tolerating $\lfloor n-1/2 \rfloor$ failures (t=3 here)

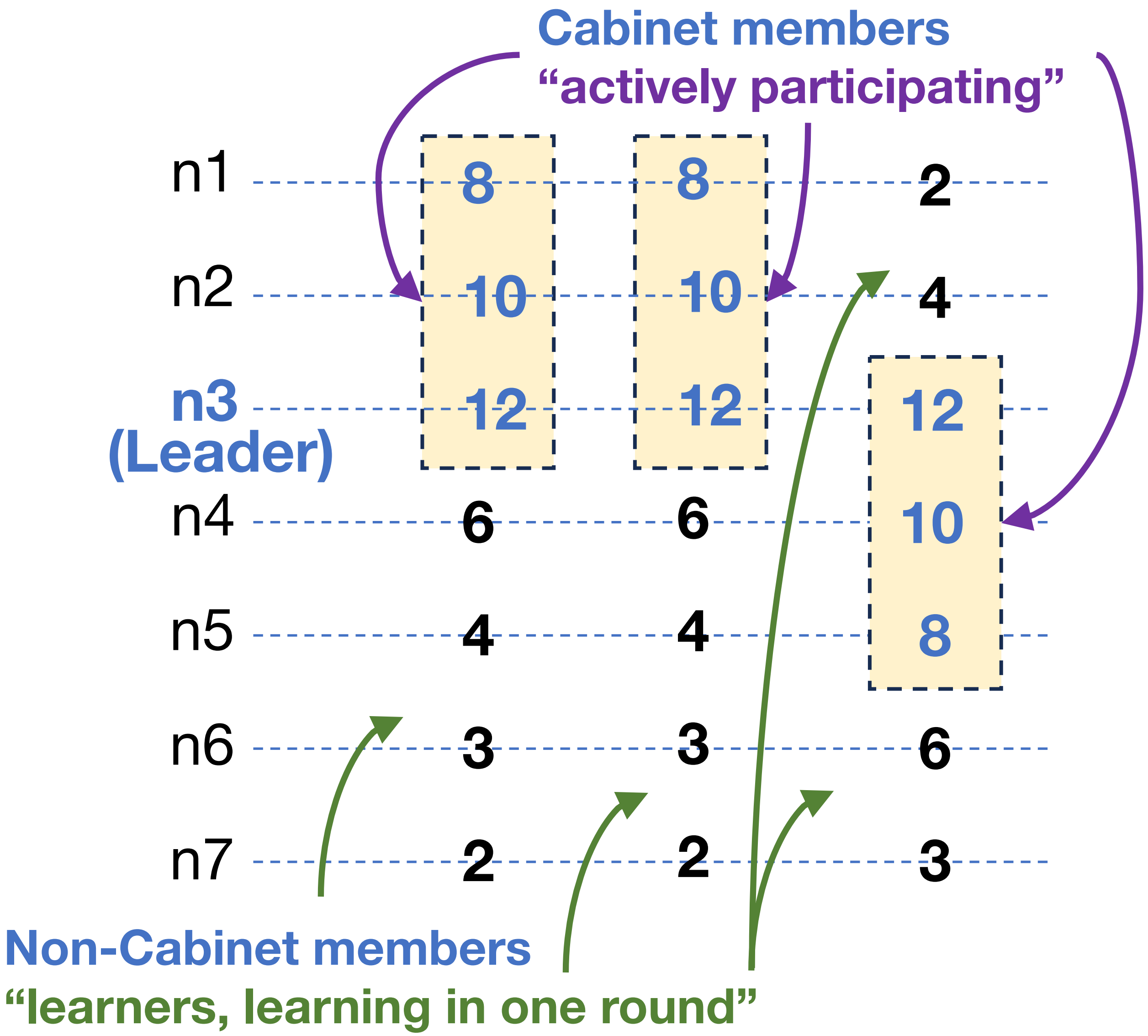
Best case Cabinet:

- When non-Cabinet members fail, tolerating up to n-t-1 failures (t=4 here)

$$CT = \sum \frac{W}{2} = 22.5$$

Feature 2: Avoid manual role selections

- Under large replication deployment, Cabinet does not need to manually partition nodes to “acceptors/followers” or “learners”
- Cabinet members are actively participating nodes – aka **acceptors/followers**
- Non-Cabinet members are **learners**, and still learn the results in one round



Example of Cabinet's weight schemes

# of failures tolerated

Cabinet members

t	r	w_1	w_2	w_3	w_4	w_5	w_6	w_7	w_8	w_9	w_{10}
1	1.40	**20.7**	**14.8**	10.5	7.5	5.4	3.8	2.7	2.0	1.4	1
2	1.38	**18.2**	**13.2**	**9.5**	6.9	5.0	3.6	2.6	1.9	1.4	1
3	1.19	**4.8**	**4.0**	**3.4**	**2.8**	2.4	2.0	1.7	1.4	1.2	1
4	1.08	**2.0**	**1.9**	**1.7**	**1.6**	**1.5**	1.4	1.3	1.2	1.1	1

Cabinet weight schemes with different customized failure thresholds ($t = 1, 2, 3, 4$) in a $n = 10$ system

Dynamic weight reassignment

- Under dynamic networks, cabinet members (who have higher weights) may not always be the best-performing nodes
  - If high-weight nodes become slow, consensus process becomes slow
  - If quorum size exceeds $|\mathrm{Q}| > \frac{n}{2}$, it may become slower than majority-quorum consensus

Dynamically reassign weights to nodes in **each consensus round** such that **faster-responding nodes are always assigned higher weights**

Cabinet inherits Raft's implementation and achieves dynamic weight reassignment by adding only **two parameters** in Raft's communication

Dynamic weight reassignment

- In Round r,
  - Leader issues RPCs to operate consensus tasks
    - Collects replies in a queue, namely, wQ; $wQ \leftarrow$ <n_i, w_i>
    - First reply, first enqueue
    - Consensus reached when the total weight in wQ exceeds CT

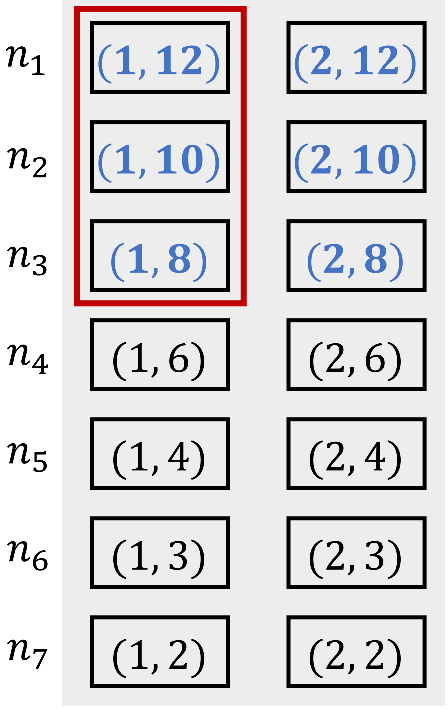
$$\sum_{for\ each\ i\ that\ n_i \in wQ} w_i > CT$$

  - **Start weight reassignment:** Leader assigns the highest to lowest weights to the first replying node to last replying node in the queue (wQ)
    - Faster nodes get higher weights

The first $t + 1$ replying nodes in round r becomes the cabinet members in round $r + 1$

Examples of dynamic weight reassignments

- $n = 7, t = 2, CT = 22.5$
- # of cabinet members = 3 (**in blue font**)
- n_1 is the leader
- (x, y) means in round x, n_i's weight is y
- Each example has two rounds
  - wQ collects replies in the first round
  - Replying nodes are boxed

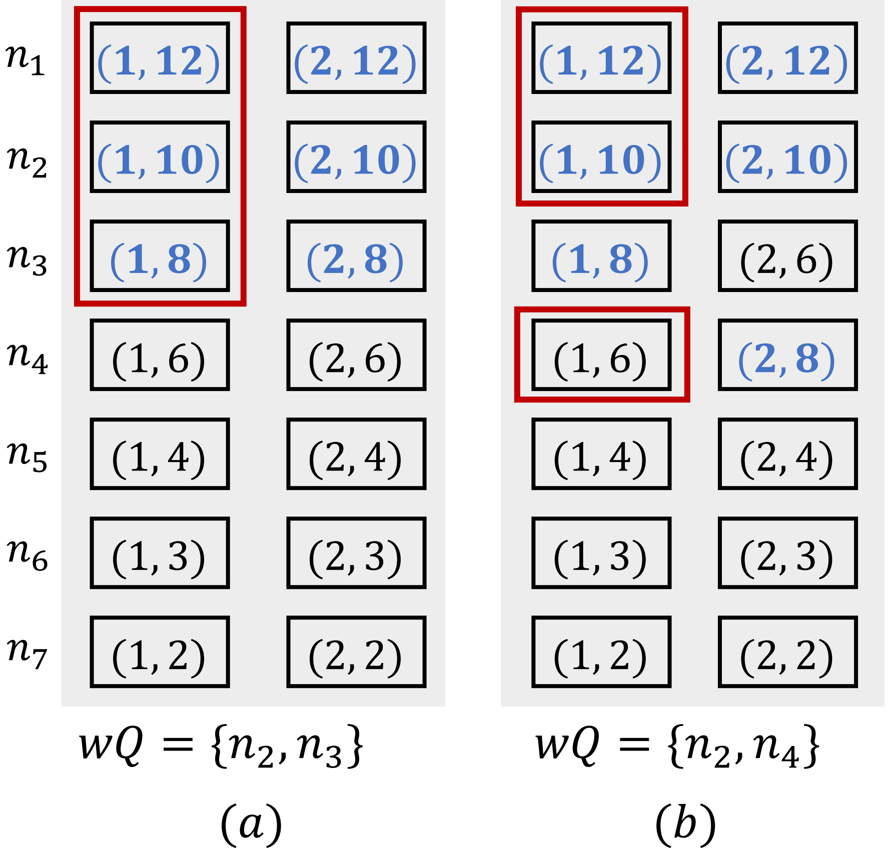


$wQ = \{n_2, n_3\}$

(a)

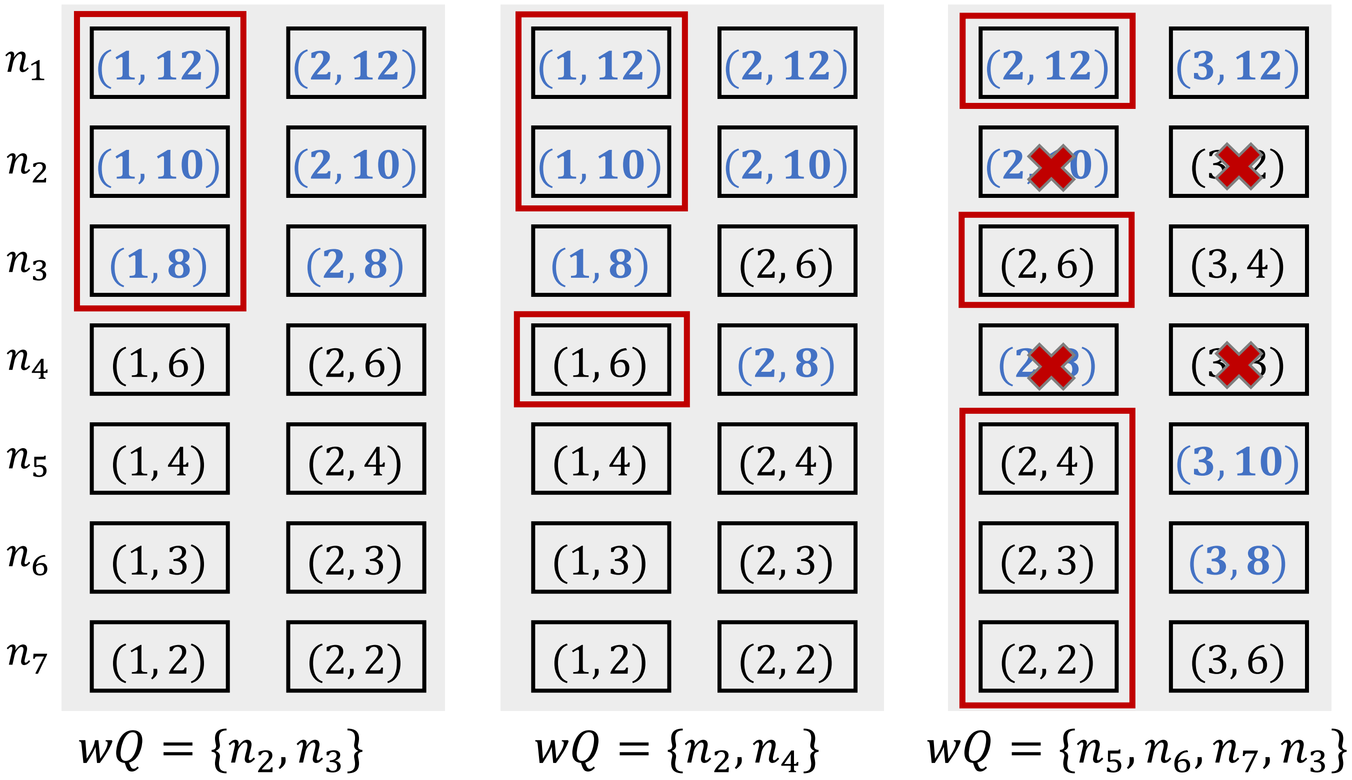
Examples of dynamic weight reassignments

- $n = 7, t = 2, CT = 22.5$
- # of cabinet members = 3 (**in blue font**)
- n_1 is the leader
- (x, y) means in round x, n_i's weight is y
- Each example has two rounds
  - wQ collects replies in the first round
  - Replying nodes are boxed



Examples of dynamic weight reassignments

- $n = 7, t = 2, CT = 22.5$
- # of cabinet members = 3 (**in blue font**)
- n_1 is the leader
- (x, y) means in round x, n_i's weight is y
- Each example has two rounds
  - wQ collects replies in the first round
  - Replying nodes are boxed



The first $t + 1$ replying nodes in round r becomes the cabinet members in round $r + 1$

Cabinet: Dynamically weighted consensus made faster

Cabinet enables a new tradeoff frontier between consensus performance and fault tolerance

Fault tolerance

Enable a tradeoff

Performant consensus

(e.g., tolerating 10 absolute failures in a 100-node cluster as opposed to 49 failures; in return we get 5× higher throughput and lower latency)

This tradeoff is yield to users/applications to achieve customizable consensus process and flexible fault tolerance

Evaluation: cluster setup

- Homogeneous and heterogeneous clusters of sizes of $n = 10, 20, 50, 100$
  - Heterogeneity of CPUs, RAM, and Disk

Cluster	Zone 1	Zone 2	Zone 3	Zone 4	Zone 5
Homogeneous	$n/5 \times$ **C3**	$n/5 \times$ **C3**	$n/5 \times$ **C3**	$n/5 \times$ **C3**	$n/5 \times$ **C3**
Heterogeneous	$n/5 \times$ **C1**	$n/5 \times$ **C2**	$n/5 \times$ **C3**	$n/5 \times$ **C4**	$n/5 \times$ **C5**

C1 1c-7.5gb-56
C2 2c-15gb-92
C3 4c-15gb-164
C4 8c-30gb-308
C5 16c-60gb-596

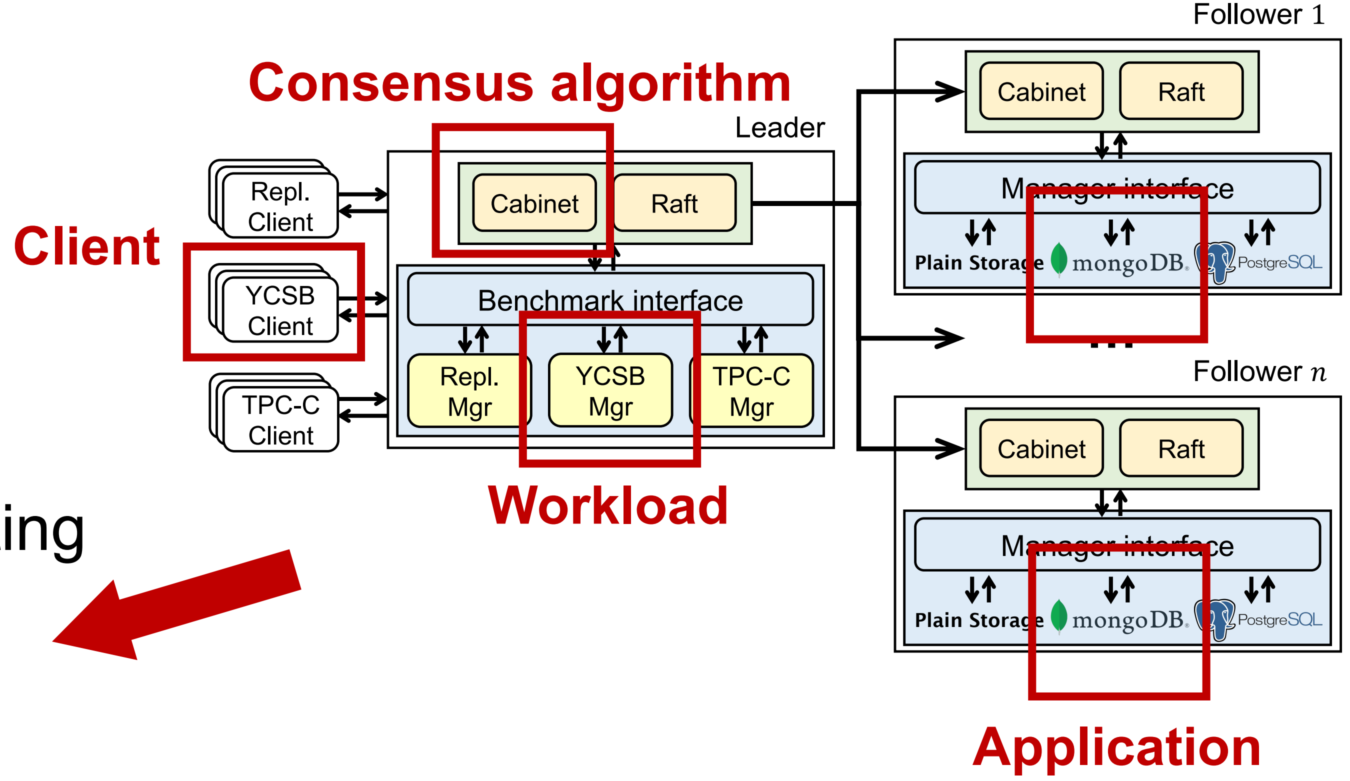
CPU: 2.40 GHz Intel Xeon processors (Skylake)
Operating system: Ubuntu 18.04.1 LTS
TCP/IP bandwidth: $\approx$400 Megabytes/s
Raw network latency: $<$1 ms

1 vCPU, 7.5GB RAM, and 56GB Disk

Homogeneous	$n = 50$	Heterogeneous
10 nodes × C3	Z1	10 nodes × C1 (weakest)
10 nodes × C3	Z2	10 nodes × C2
10 nodes × C3	Z3	10 nodes × C3
10 nodes × C3	Z4	10 nodes × C4
10 nodes × C3	Z5	10 nodes × C5 (strongest)

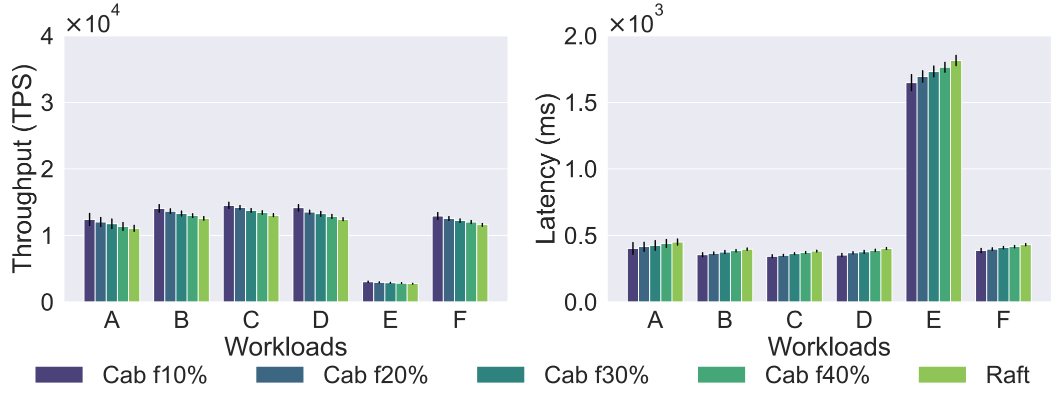
Cabinet benchmark framework

- It is tailored for leader-based consensus applications
  - Leader
  - Followers
  - Benchmarking clients
- It provides versatile and adaptable interfaces that seamlessly integrate existing benchmarks
  - MongoDB + YCSB
  - PostgreSQL + TPC-C



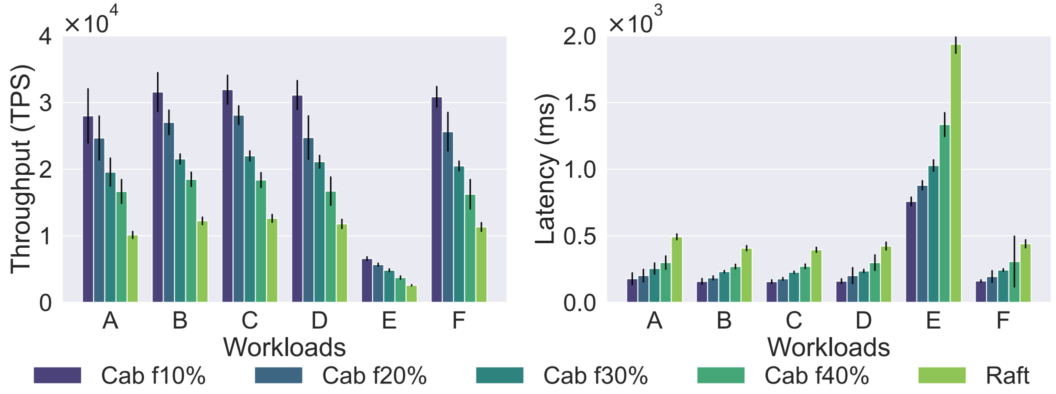
Performance under MongoDB+YCSB (50 nodes)

Homogeneous cluster



slightly higher throughput & lower latency

Heterogeneous cluster

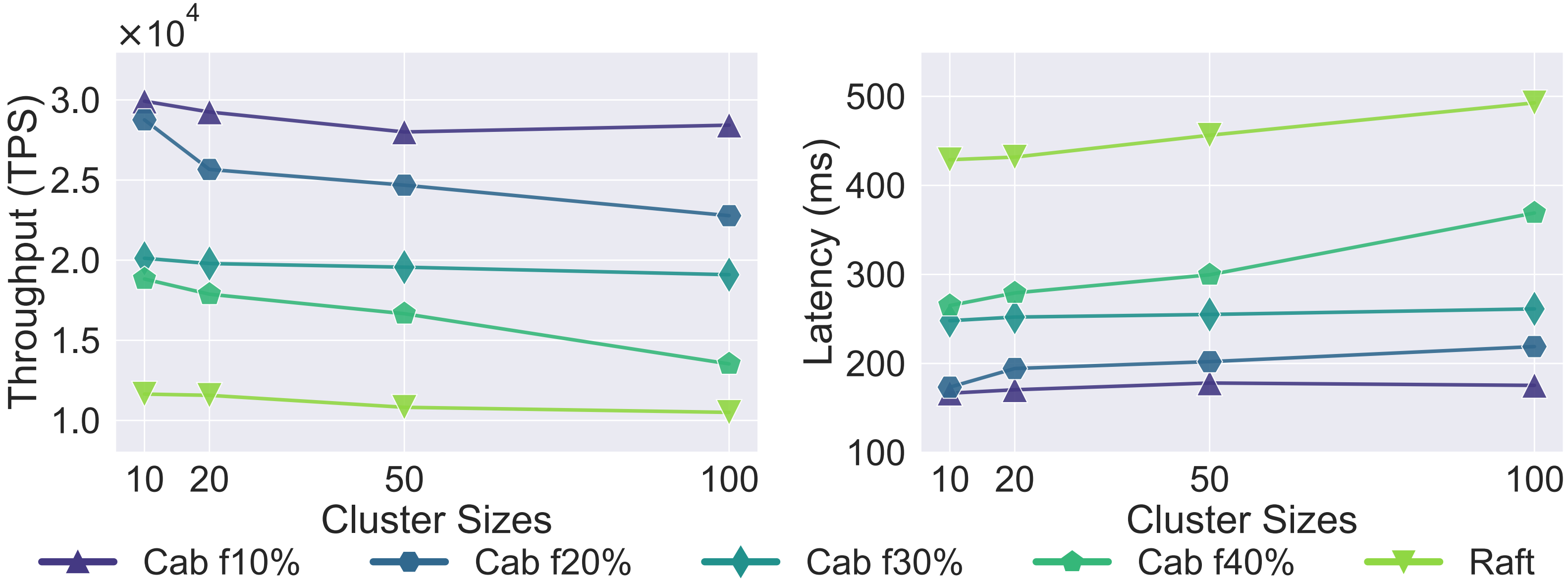


$3\times$ higher throughput

$2.8\times$ lower latency

Strong nodes become cabinet members, leading to high performance by emancipating themselves from waiting for weak nodes

Performance under scaling clusters



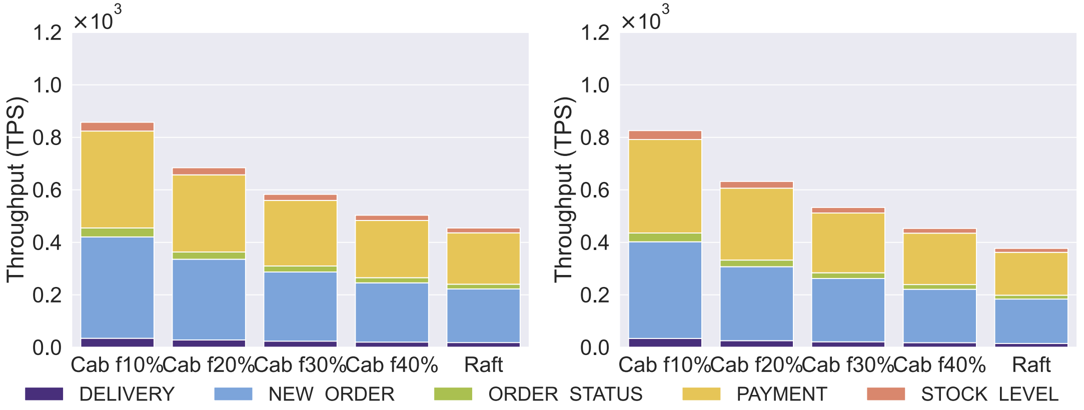
Heterogeneous clusters under
YCSB workload A

Cabinet's weighted consensus mechanism **consistently outperforms** Raft's traditional consensus at all scales

Performance under PostgreSQL+TPC-C

Homogeneous cluster

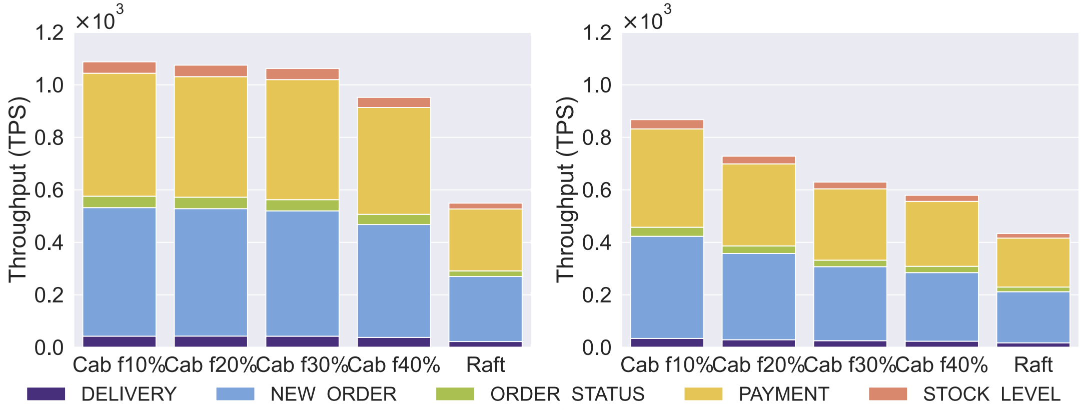
$n = 10$ $\qquad$ $n = 50$



2× higher throughput

Heterogeneous cluster

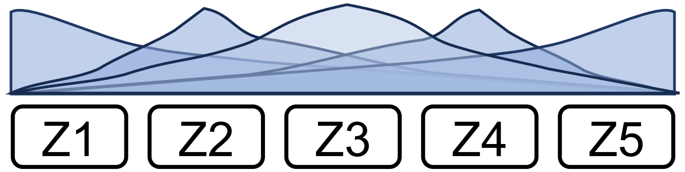
$n = 10$ $\qquad$ $n = 50$



2.2× higher throughput

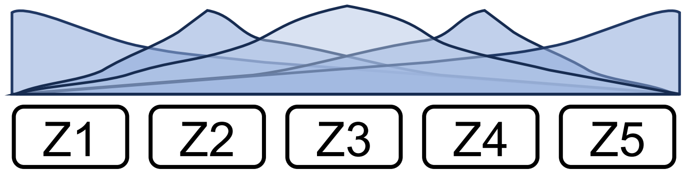
Compared to YCSB, performance gain decreases, due to decreased parallelism in workload

Performance under dynamic network delays



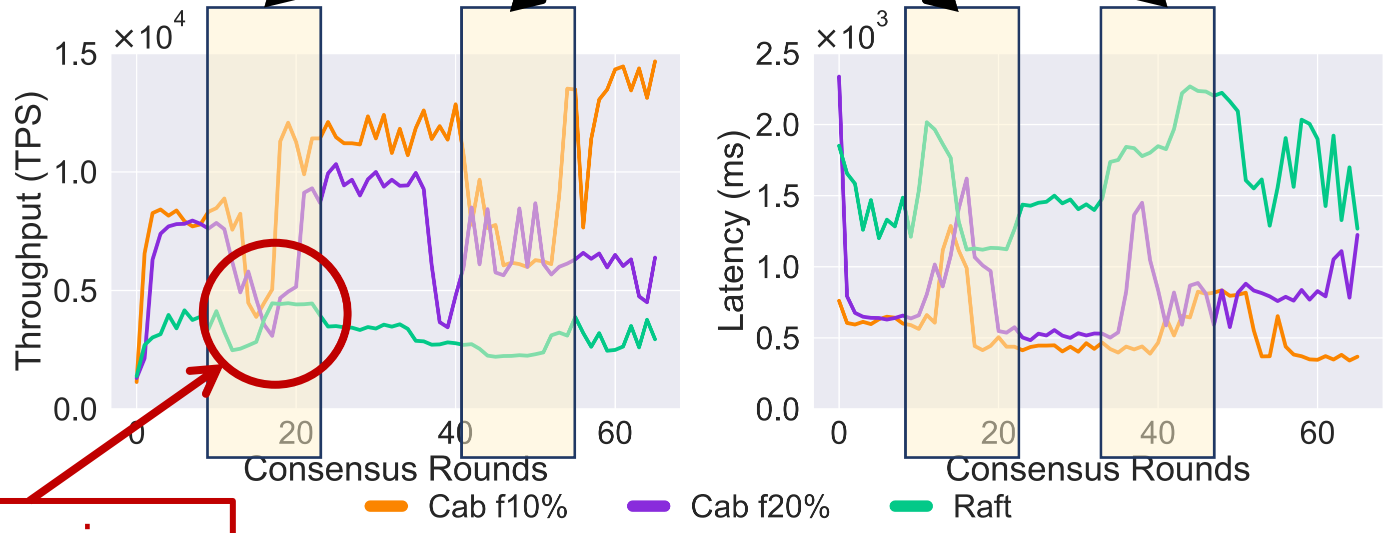
Network delays of $1000 \pm 200ms$ to $100 \pm 20ms$ are dynamically imposed across 5 zones

Performance under dynamic network delays



Network delays of $1000 \pm 200ms$ to $100 \pm 20ms$ are dynamically imposed across 5 zones

Weight reassignment promptly reassigns high weights to currently faster nodes.
Cabinet stays at optimal performance!

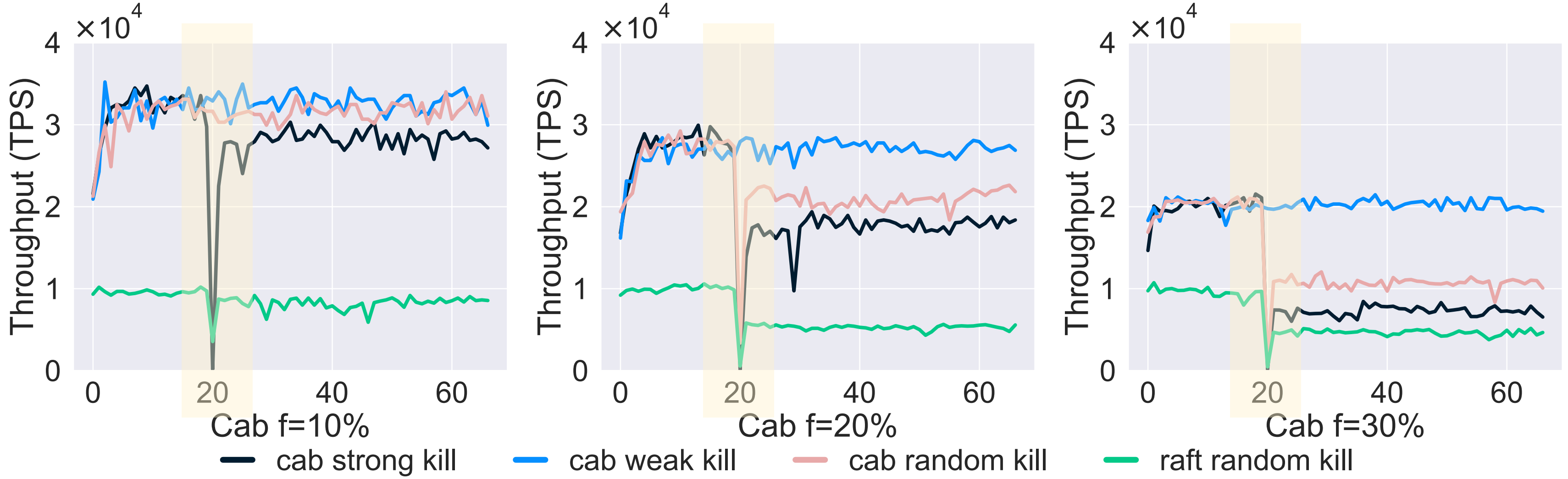


Strong nodes experience high network delays

$n = 50$ heterogeneous cluster under YCSB workload A

Performance under crash failures

- **Strong kills** crash x top highest-weight nodes
- **Weak kills** crash x bottom lowest-weight nodes
- **Random kills** randomly crash x nodes



Cabinet outperforms Raft under all failure strategies

Conclusions

Cabinet is the first dynamically weighted consensus algorithm, contributing to both theoretical foundations and systems implementation

Formulates weighted consensus
(e.g., invariants and proofs)

Rearranges of weights based on node responsiveness

Enables flexible fault tolerance
(e.g., customized failure thresholds)

Theoretical foundations

System implementation

Simple integration to Raft
(e.g., AppendEntries RPCs)

Versatile benchmark framework for leader-based consensus applications

High-performance implementation under scaling clusters, dynamic networks, and crash failures

Paxos
1990

Cabinet
2024

Raft
2014

Cabinet's dynamically weighted consensus makes it an ideal choice for large-scale, especially heterogeneous, consensus applications!

Thank you!