Multi-Threshold Byzantine Fault Tolerance

Atsuki Momose *[†], Ling Ren[‡]

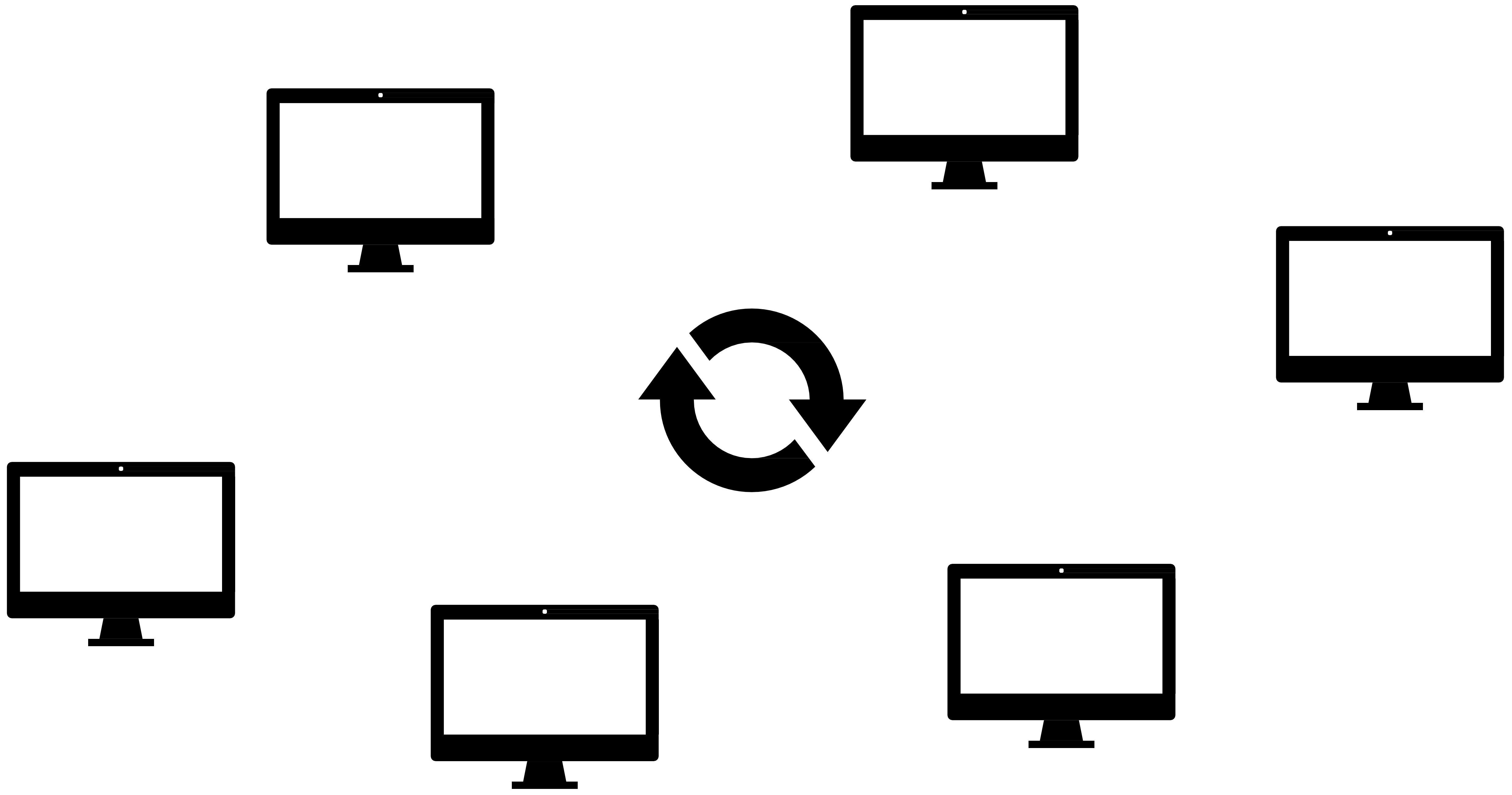
Intelligent Systems Laboratory, SECOM CO., LTD.

Nagoya University[†]

University of Illinois at Urbana-Champaign[‡]

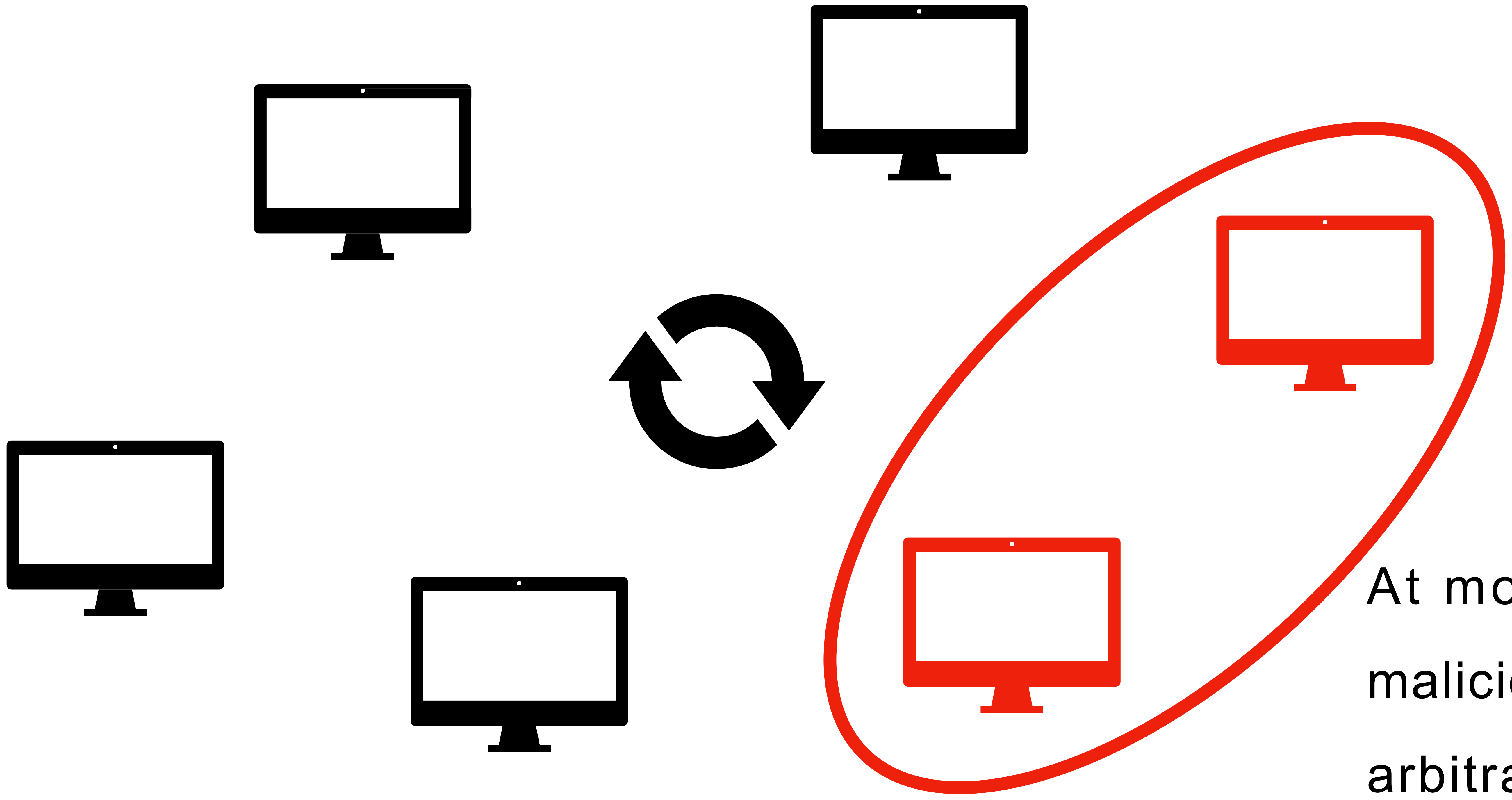
Byzantine fault tolerance (BFT)

Class of distributed algorithm that tolerates arbitrarily deviating faults.



Byzantine fault tolerance (BFT)

Class of distributed algorithm that tolerates arbitrarily deviating faults.



At most f nodes are malicious and behave arbitrarily

$\rightarrow$ Byzantine fault

Classic BFT design

Classic BFT design first selects its timing assumptions from below.

Model	Fault-tolerace	Protocol
Synchrony. Every message is delivered within Δ	$f < n/2$ or $f < n$	Sync HotStuff, Dolev-Strong
Asynchrony. No bound on message delay	$f < n/3$	HoneyBadgerBFT, BEAT, Dumbo
Partial-synchrony. Synchronous after GST		PBFT, HotStuff

tolerate more faults

tolerate asynchrony

Classic BFT design

Classic BFT design first selects its timing assumptions from below.

Model	Fault-tolerace	Protocol
Synchrony. Every message is delivered within Δ	$f < n/2$ or $f < n$	Sync HotStuff, Dolev-Strong
Asynchrony. No bound on message delay	$f < n/3$	HoneyBadgerBFT, BEAT, Dumbo
Partial-synchrony. Synchronous after GST		PBFT, HotStuff

tolerate more faults

tolerate asynchrony

Classic BFT design

Classic BFT design first selects its timing assumptions from below.

Model	Fault-tolerace	Protocol
Synchrony. Every message is delivered within Δ	$f < n/2$ or $f < n$	Sync HotStuff, Dolev-Strong
Asynchrony. No bound on message delay	$f < n/3$	HoneyBadgerBFT, BEAT, Dumbo
Partial-synchrony. Synchronous after GST		PBFT, HotStuff

tolerate more faults

tolerate asynchrony

How synchrony is useful?

- If the network synchrony helps tolerate more faults, what if asynchronous or partial synchronous protocols run in a synchronous network?
- Can we tolerate $\lfloor n/3 \rfloor - 1$ faults under asynchrony and $\geq n/3$ under synchrony?

Dual threshold BFT (Blum et al.—TCC'19, Crypto'20, Asiacrypt'21)

- A protocol simultaneously tolerates f_s faults under synchrony and f_a faults under asynchrony.
- Classic asynchronous protocols → $f_s = f_a = f = \lfloor n/3 \rfloor - 1$
- Dual threshold BFT is possible $\Leftrightarrow 2f_a + f_s < n$
    - $0 < f_a < n/3$ (i.e., tolerate asynchrony) and $f_s \geq n/3$ is possible (Good news)
    - If $f_a = \lfloor n/3 \rfloor - 1$, then $f_s = \lfloor n/3 \rfloor - 1$ (Bad news)

Multi-threshold BFT (this work)

- A protocol simultaneously tolerates (β_s, γ_s) faults under synchrony and (β_a, γ_a) faults under asynchrony (or partial-synchrony).
  $\rightarrow$ Achieve safety with β_s (or β_a) faults, and liveness with $_s$ (or $_a$) faults.
  - Safety - "nothing bad happens" (e.g., nodes do not decide differently)
  - Liveness - "something happens" (e.g., everyone decides eventually)
- Blum et al's bound $2f_a + f_s < n$ can be generalized to $2\beta_a + \gamma_s < n$
  - The trade-off is in $\beta_a \leftrightarrow \gamma_s$ but not in $\beta_a \leftrightarrow \beta_s$
  - $\beta_s \geq n/3$ and $\beta_a = \gamma_a = \gamma_s = \lfloor n/3 \rfloor - 1$ is possible (Main result)
    $\rightarrow$ control the network, or corrupt more to attack.

RBC, SMR

Reliable broadcast (RBC).

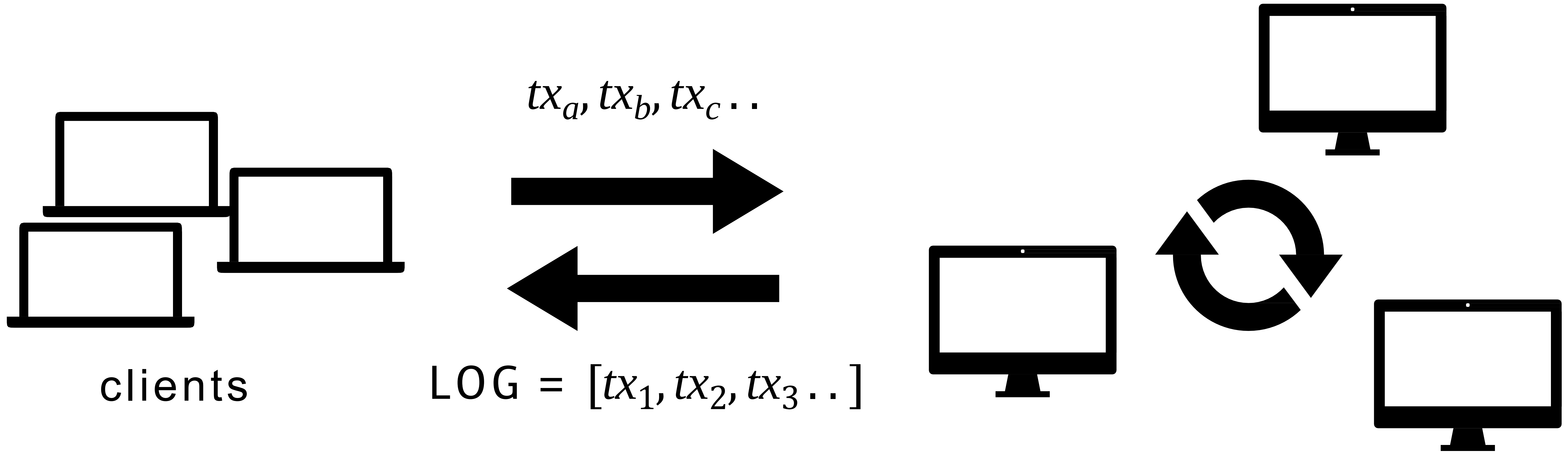
- A designated sender node broadcasts a value.
- A building block of many distributed cryptographic protocols, e.g., SMR, DKG.

State machine replication (SMR).

- The most practical formulation of consensus problem.
- The underlying problem of blockchain.
- Provide clients with an abstraction of a single non-faulty server.

State machine replication (SMR)

Nodes agree on a growing log of requests from clients.



- Safety. Honest nodes do not output different requests at the same log position.
- Liveness. Every request is eventually included in a log.

Clients can verify the correctness of a log—public verifiability

β : safety

: liveness

Tight fault tolerance

Problem	Tight fault tolerance
RBC	$\beta_a = n - 2\gamma_s - 1$ $\beta_s = n - 1$ $\gamma_a = \min\{\beta_a, \gamma_s\}$
SMR	$\beta_a = n - 2\gamma_s - 1$ $\beta_s = n - \gamma_s - 1$ $\gamma_a = \min\{\beta_a, \gamma_s\}$

$2\gamma_s + \beta_a < n$

The generalized Blum et al’s bound

β : safety

: liveness

Tight fault tolerance

Problem	Tight fault tolerance
RBC	$\beta_a = n - 2\gamma_s - 1$ $\beta_s = n - 1$ $\gamma_a = \min\{\beta_a, \gamma_s\}$
SMR	$\beta_a = n - 2\gamma_s - 1$ $\beta_s = n - \gamma_s - 1$ $\gamma_a = \min\{\beta_a, \gamma_s\}$

tolerate arbitrary high fault
for synchronous safety

Tight fault tolerance

β : safety

: liveness

Problem	Tight fault tolerance
RBC	$\beta_a = n - 2\gamma_s - 1$ $\beta_s = n - 1$ $\gamma_a = \min\{\beta_a, \gamma_s\}$
SMR	$\beta_a = n - 2\gamma_s - 1$ $\beta_s = n - \gamma_s - 1$ $\gamma_a = \min\{\beta_a, \gamma_s\}$

$\gamma_s = \beta_s = f \Rightarrow f < n/2$

(Schneider's bound)

$\beta_s + \gamma_s < n$

another bound due to public verifiability

$\beta_s < 2n/3$ while

$\beta_a = \gamma_a = \gamma_s < n/3$ is possible

A generic upgrading framework

Existing asynchronous or partially synchronous protocol can be upgraded to achieve the optimal synchronous safety tolerance.

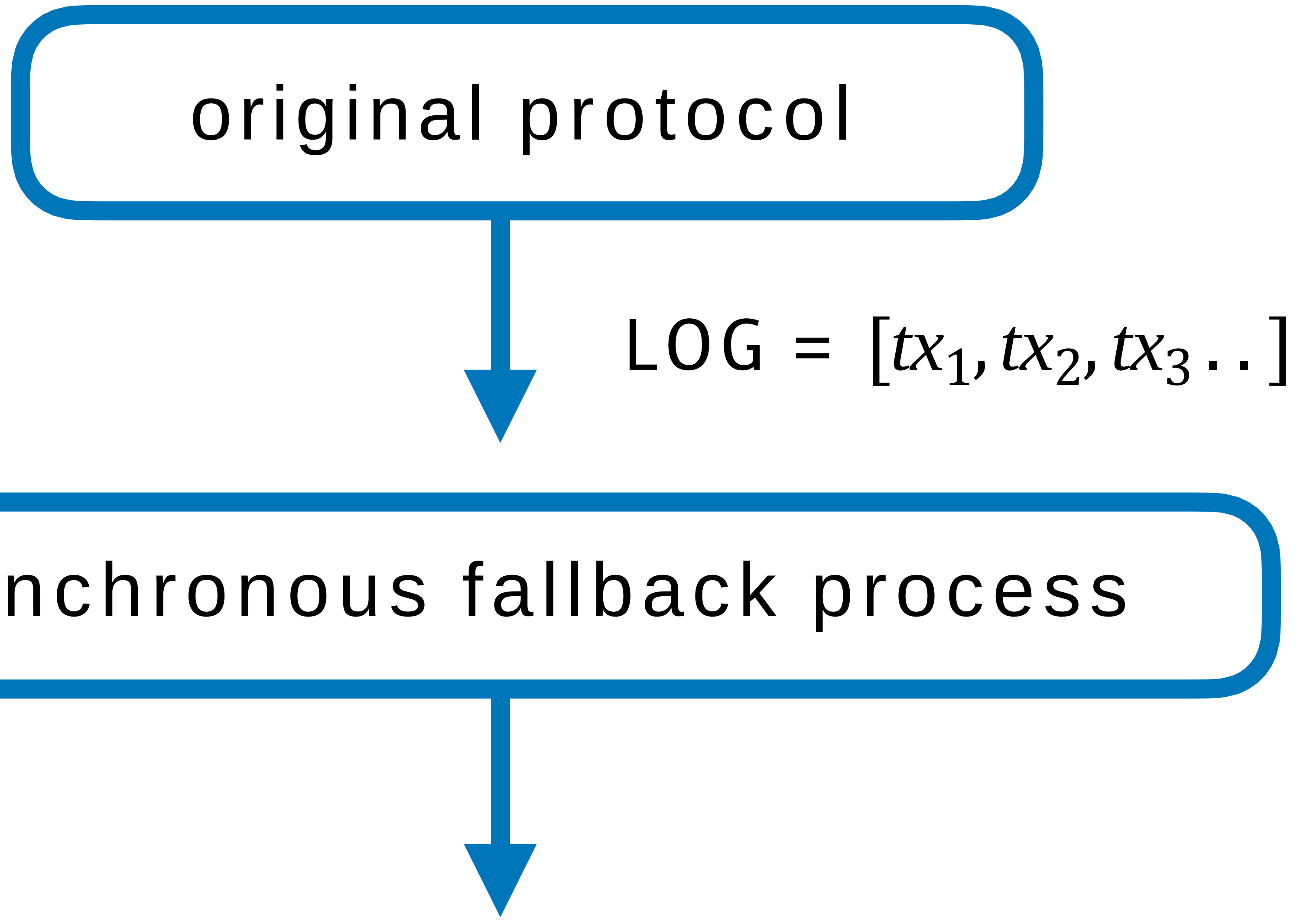
Any asynchronous or partially synchronous BFT SMR protocol with $\beta_s = \gamma_s = \beta_a = \gamma_a < n/3$

- Asynchronous protocol
  $\to$ HoneyBadgerBFT, Dumbo.
- Partially synchronous protocol
  $\to$ PBFT, HotStuff

A BFT SMR protocol with $\gamma_s = \beta_a = \gamma_a < n/3$ and $\beta_s < 2n/3$

A generic upgrading framework

A synchronous fallback process check if safety violation happens in the original protocol.



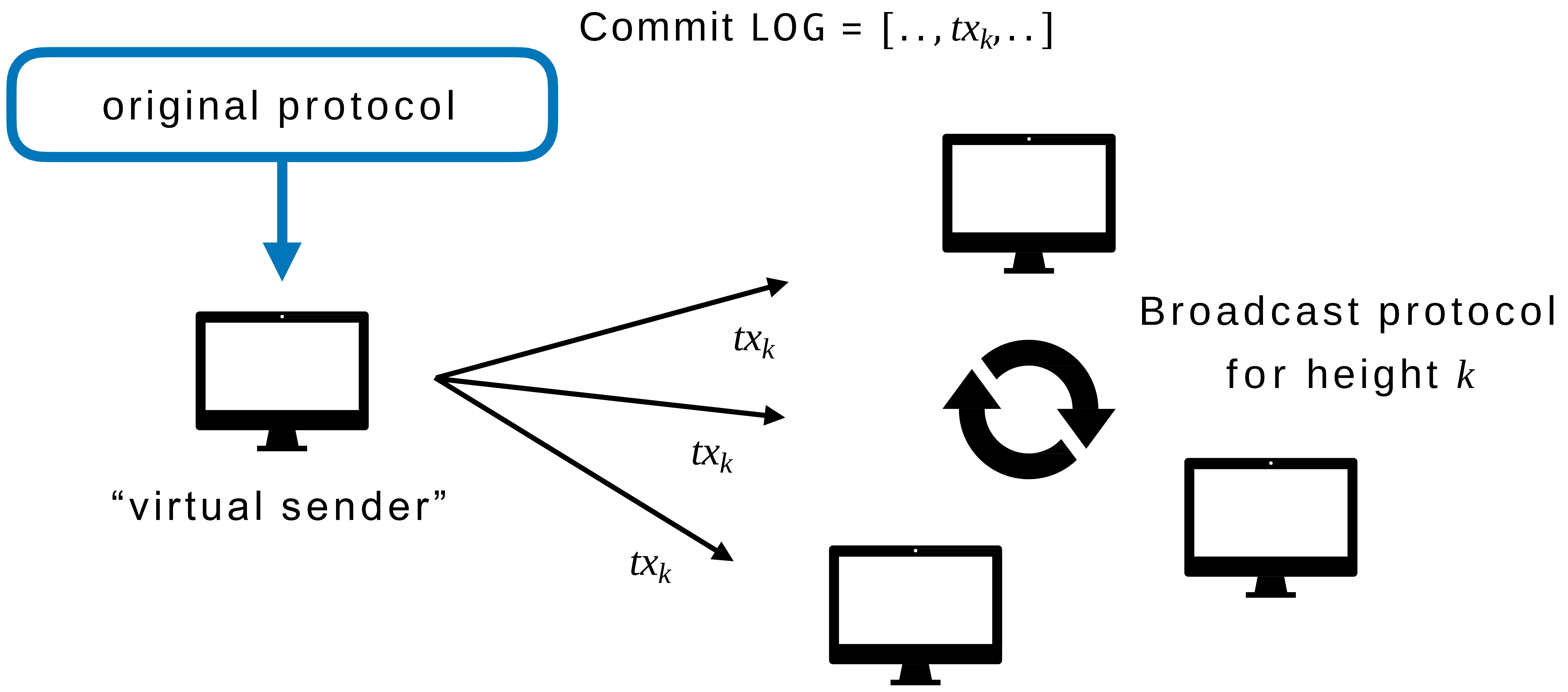
If detect safety violation, stop committing.
Otherwise, commit LOG

Synchrony + $\geq n/3$ fault
$\rightarrow$ The fallback process can detect safety violation.

Asynchrony + $< n/3$ fault
$\rightarrow$ The original protocol is already safe.

Fallback process

The fallback process is similar to a synchronous broadcast protocol.

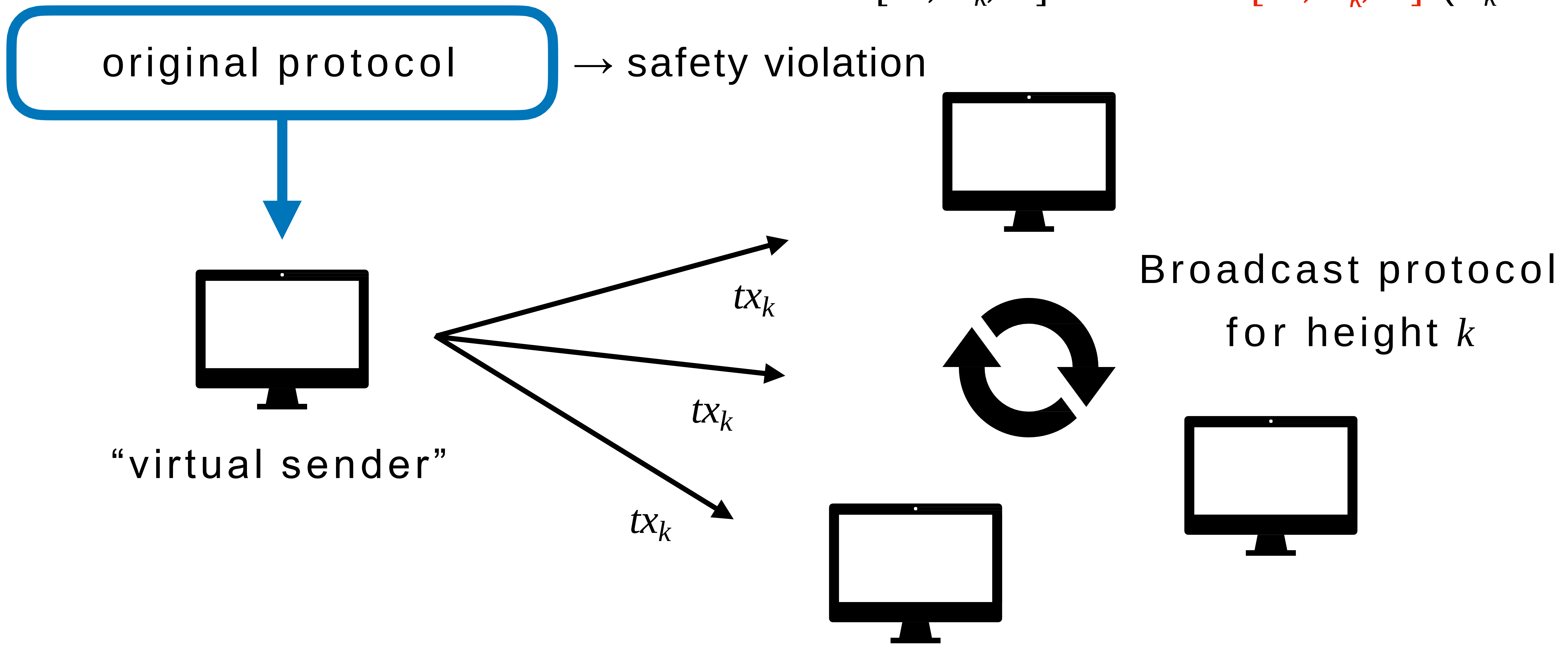


Fallback process

The fallback process is similar to a synchronous broadcast protocol.

Commit LOG = $[.., tx_k, ..]$ & LOG′ = $[.., tx'_k, ..]$ $(tx_k \neq tx'_k)$

→ safety violation

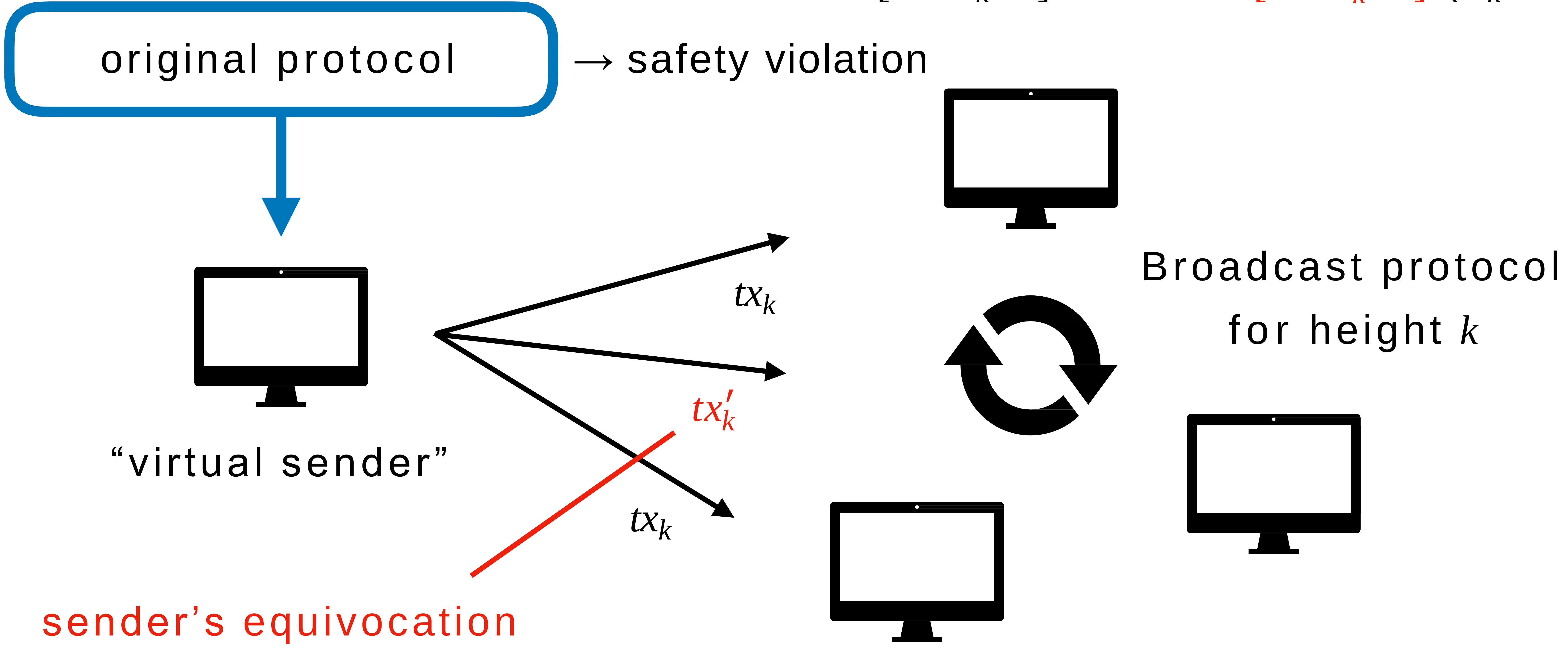


Fallback process

The fallback process is similar to a synchronous broadcast protocol.

Commit LOG = $[..,tx_k,..]$ & LOG$'$ = $[..,tx'_k,..]$ $(tx_k \neq tx'_k)$

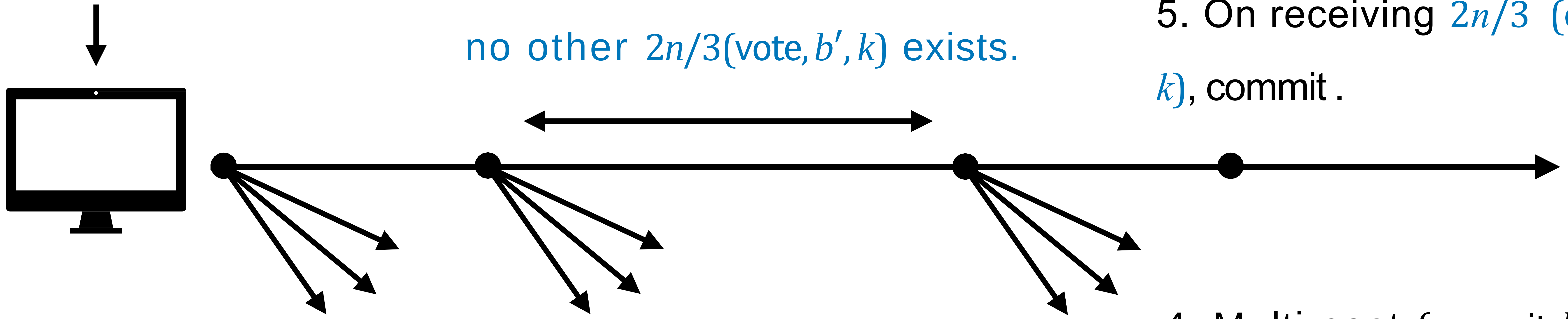
$\rightarrow$ safety violation



Fallback process—for height k

We adopt the Xiang et al.'s method [PODC'21] for detecting equivocation.

The original protocol commits at height k



1. Multi-cast (vote, b, k)

2. On receiving $2n/3$ (vote, b, k), forward them to all.

3. Wait for Δ, and check that no other $2n/3 (\text{vote}, b', k)$ exists.

4. Multi-cast (commit, b, k),

5. On receiving $2n/3$ (commit, b, k), commit .

PV proof of committing

PV (publicly verifiable) proof that original protocol commits

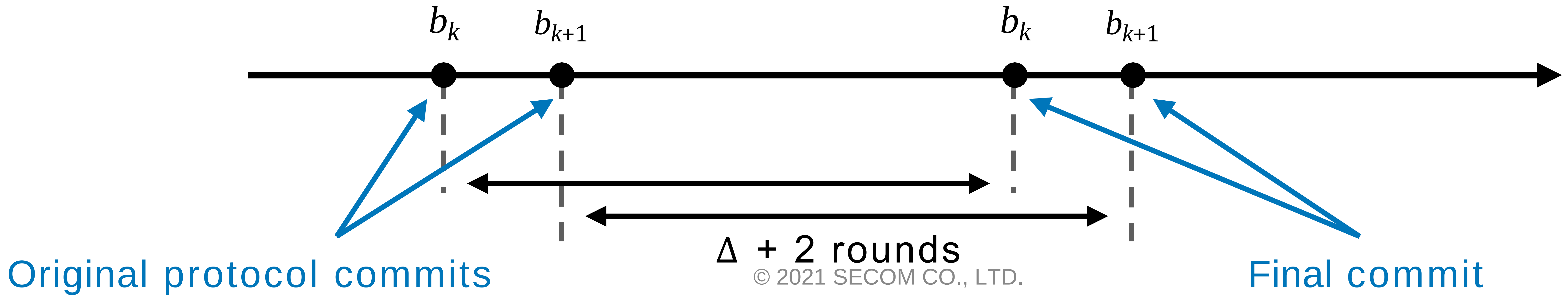
Overhead (in theory)

Latency.

- Latency of the original protocol + Δ + 2 rounds.
- Not responsive, i.e., depends on Δ, which is inherent if $\beta_s \geq n/3$ is desired.

Throughput.

- $O(n^2)$ communication overhead → original protocols usually cost $\Omega(n^2)$
- Δ-waiting step does not hurt the throughput.



β : safety

: liveness

Flexible threshold parameters.

We show a protocol (combining Sync HotStuff and PBFT) that allows any fault thresholds in the optimal trade-off in the partial synchrony model.

Problem	Tight fault tolerance
SMR	$\beta_a = n - 2\gamma_s - 1$ $\beta_s = n - \gamma_s - 1$ $\gamma_a = \min\{\beta_a, \gamma_s\}$

Safety favoring.

$\gamma_s = \gamma_a < n/4, \beta_a < n/2, \beta_s < 3n/4$

High availability under synchrony.

$\gamma_s < 9n/20, \beta_s < 11n/20, \beta_a = \beta_a < n/10$

Summary

- Classic BFT: one fault threshold, and one timing assumption.
  → trade-off in timing assumption and fault tolerance.

- Multi-threshold BFT: separate fault thresholds for
  1. different timing assumptions–synchrony and asynchrony
  2. security properties–safety and liveness.

- Higher synchronous safety tolerance $\beta_s < 2n/3$ is possible with $\beta_a = \gamma_a = \gamma_s = \lfloor n/3 \rfloor - 1$.