Byzantine-Tolerant Consensus in GPU-Inspired Shared Memory

Authors:
Chryssis Georgiou, University of Cyprus, Cyprus
Manaswini Piduguralla, IIT Hyderabad, India
Sathya Peri, IIT Hyderabad, India
29 August 2025

Table of Contents

1. Research Objective
2. Background
3. Proposed System Model
4. Byzantine Consensus
5. Proposed Solution
6. Conclusion

Table of Contents

1. Research Objective
2. Background
3. Proposed System Model
4. Byzantine Consensus
5. Proposed Solution
6. Conclusion

Research Objective

Agreement among multiple parties is a fundamental problem in distributed computing for a wide range of applications[1].

- GPUs have evolved from graphics to general-purpose parallel computation.
- Their SIMT[2] execution model and warp-based scheduling differ fundamentally from CPUs.
- We aim to explore fundamental problems of distributed computing in a General Purpose GPU (GPGPU) architecture.

[1] Ajay D. Kshemkalyani and Mukesh Singhal. *Distributed Computing: Principles, Algorithms, and Systems*. 1st. Cambridge University Press, 2011. ISBN: 9780521189842.

[2] Single Instruction Multiple Threads

Table of Contents

1. Research Objective
2. Background
3. Proposed System Model
4. Byzantine Consensus
5. Proposed Solution
6. Conclusion

The GPU Architecture

- Streaming Multiprocessors (SMs) host many simple cores.
- Threads are organized into **warps** (typically 32) executing in lock-step.
- Warp scheduler selects which warp executes in each cycle.

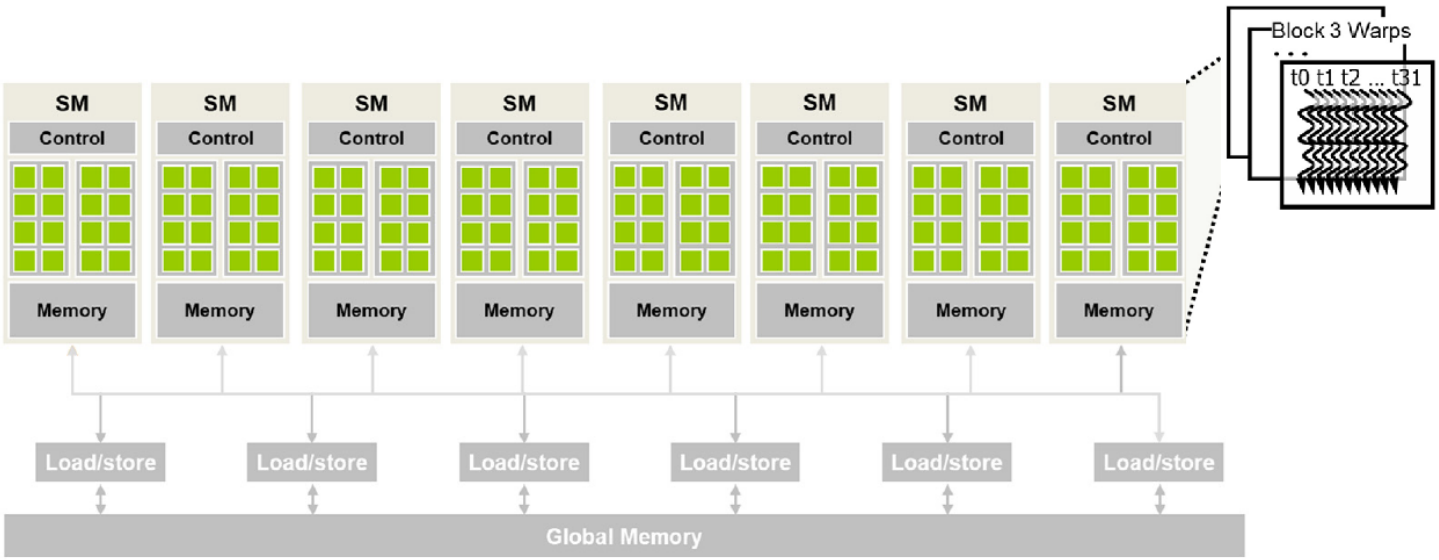


Figure: Illustration of CUDA-capable GPU Architecture[3]

[3] David B. Kirk and Wen-mei W. Hwu. "Chapter 4 - Memory and data locality". In: *Programming Massively Parallel Processors (Third Edition)*. 2017, pp. 71–101.

SIMT and Warp Scheduling

- **SIMT**: Single Instruction, Multiple Threads
- Within a warp: synchronous; across warps: asynchronous.
- Warps proceed as the scheduler issues a small, fixed number of instructions per cycle.

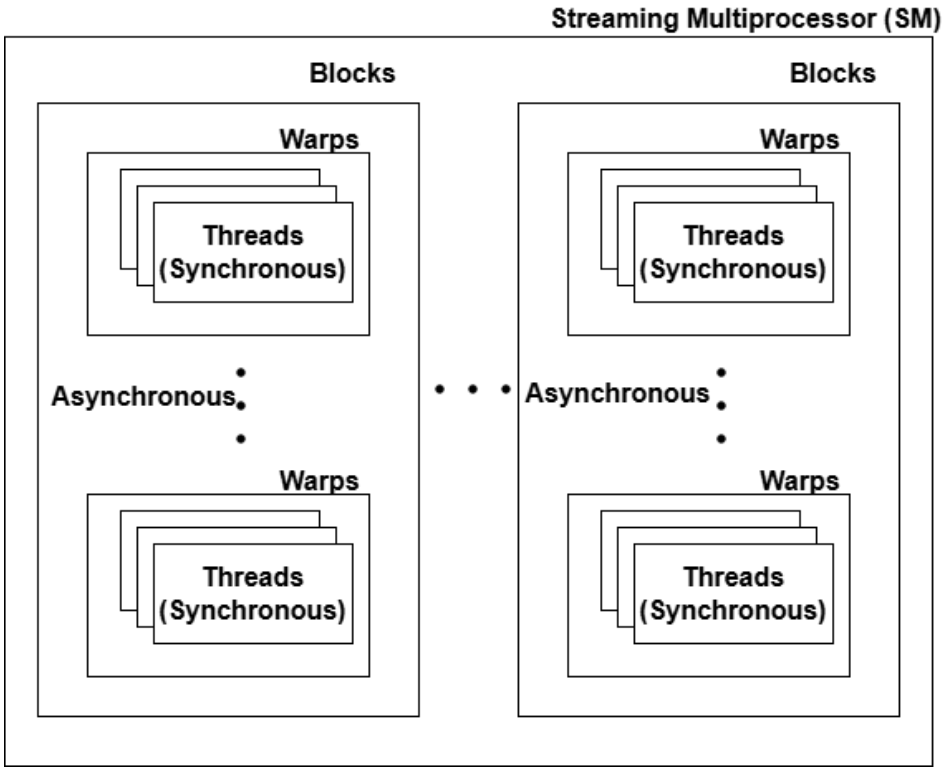


Figure: Illustration of GPU Hierarchy

Why Byzantine consensus in GPUs

- Multi-tenant GPU setups (e.g., MPS) and fairness in HPC clusters, fault-tolerant resource usage[4] motivated us to investigate consensus.
- GPUs show higher susceptibility to hardware errors than CPUs (e.g., transient bit flips)[5].
- Software malfunctions can lead to silent or inconsistent failures in GPU workloads.
- Byzantine model is well-suited for unpredictable failures, as it can provide correctness guarantees even in the presence of arbitrary faults.

[4] Alex Weaver et al. "Granularity- and Interference-Aware GPU Sharing with MPS". In: *SC24-W: Workshops of SC24*. 2024, pp. 1630–1637.

[5] Nevin Cini and Gulay Yalcin. "A Methodology for Comparing the Reliability of GPU-Based and CPU-Based HPCs". In: *ACM Comput. Surv.* 53.1 (Feb. 2020). ISSN: 0360-0300.

Table of Contents

1. Research Objective
2. Background
3. Proposed System Model
4. Byzantine Consensus
5. Proposed Solution
6. Conclusion

The GPU-Inspired Shared Memory Model

- System consists of a static set of n processes: $P_1, P_2, \ldots, P_n$.
- These processes correspond to threads of **a block** in a Streaming Multiprocessor (SM).
- Processes are grouped into warps of **size p**, total $\lceil n/p \rceil$ warps.
- Focus of this work: single block of a single SM (circled).

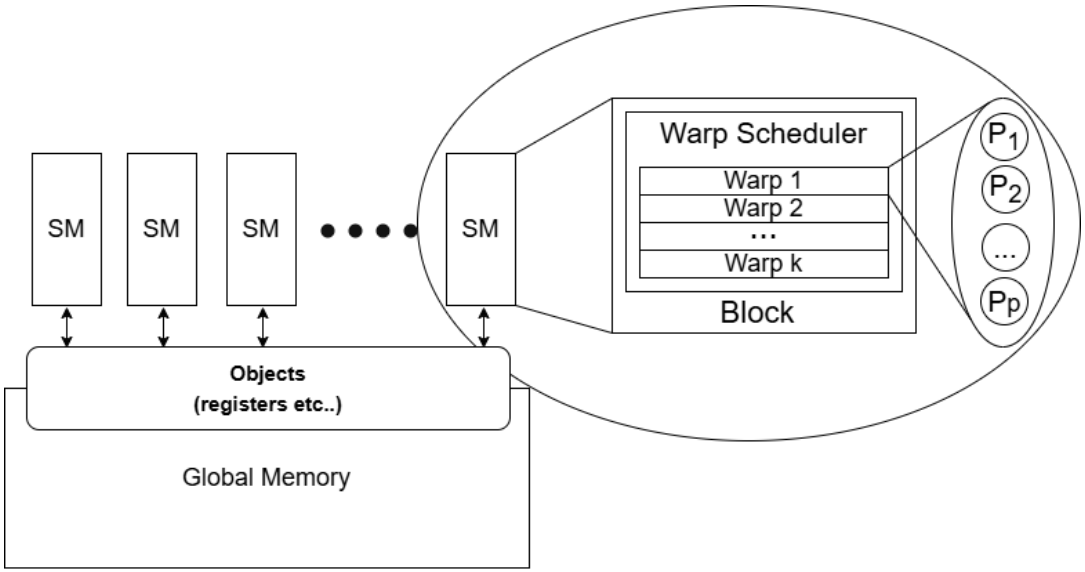


Figure: Illustration of the system model

Warp Scheduler

- Hardware scheduler maps warps to cores[6].
- Processes can only be activated by the scheduler.
- Byzantine threads cannot bypass or impersonate the scheduler.
- We assume **fair scheduling**[7,8]:
  - Each process eventually gets a chance to execute.
  - Scheduler itself is not Byzantine.

[6] Ogier Maitre. "Understanding NVIDIA GPGPU Hardware". In: *Massively Parallel Evolutionary Computation on GPGPUs*. 2013, pp. 15–34.

[7] Chee Siang Wong et al. "Towards Achieving Fairness in the Linux scheduler". In: *ACM SIGOPS Operating Systems Review* 42.5 (2008), pp. 34–43.

[8] Chandandeep Singh Pabla. "Completely fair scheduler". In: *Linux Journal* 2009.184 (2009), p. 4.

Phase-Based Computation

- Computation proceeds in synchronized **phases**.
- One warp (p processes) is scheduled per phase.
- Within a phase:
  - Processes in the warp execute synchronously in lock-step.
- Across different warps: execution is asynchronous.

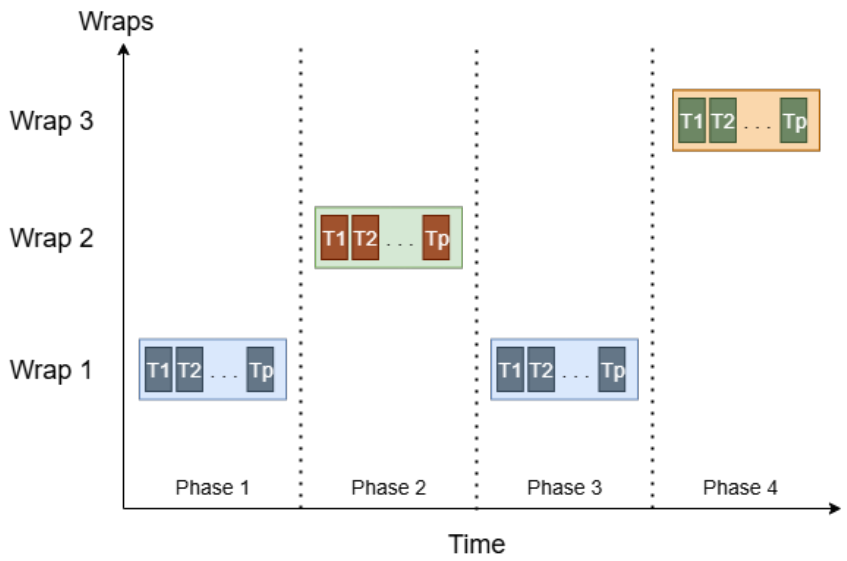


Figure: Illustration of Phase Based Execution.

Failure Model

- We consider **Byzantine failures**[9].
- Up to $f < n$ processes may be corrupted.
- Faulty processes:
  - May deviate arbitrarily from the protocol.
  - Cannot impersonate another process or exceed phase latency.
- Correct processes follow the protocol and take infinitely many steps.
- A protocol tolerating up to f Byzantine processes is called *f-resilient*.

[9] Leslie Lamport, Robert Shostak, and Marshall Pease. "The Byzantine Generals Problem". In: *ACM Trans. Program. Lang. Syst.* 4.3 (July 1982), pp. 382–401. ISSN: 0164-0925.

Table of Contents

1. Research Objective
2. Background
3. Proposed System Model
4. Byzantine Consensus
5. Proposed Solution
6. Conclusion

Introduction to Byzantine Consensus

- Agreement among multiple parties is a fundamental problem in distributed computing[10].
- Recently, there has been increased emphasis on **Byzantine fault-tolerant shared objects.**[11]
- Consensus: n processes propose values and must agree on a single value.
- Different variants exist depending on the validity property.

[10] Ajay D. Kshemkalyani and Mukesh Singhal. *Distributed Computing: Principles, Algorithms, and Systems*. 1st. Cambridge University Press, 2011. ISBN: 9780521189842.

[11] Idit Keidar Shir Cohen. "Tame the Wild with Byzantine Linearizability: Reliable Broadcast, Snapshots, and Asset Transfer". In: *DISC 2021*. 2021, 18:1–18:18.

Types of Byzantine Consensus Considered

- In this work, we consider:
  1. Strong Byzantine Consensus (SBC)[12]
  2. Common-Value Byzantine Consensus (CVBC)[13]
  3. Weak Byzantine Consensus (WBC)[11]

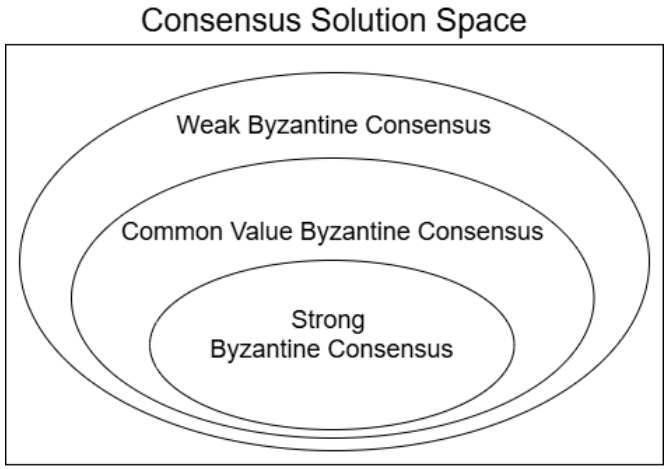


Figure: Illustration of Consensus Solution Space.

[12] Dahlia Malkhi et al. "Objects shared by Byzantine processes". In: *Distrib. Comput.* 16.1 (Feb. 2003), pp. 37–48. ISSN: 0178-2770.

[13] Paul Attie. "Wait-free Byzantine consensus". In: *Information Processing Letters* 83.4 (2002), pp. 221–227. ISSN: 0020-0190.

Strong Byzantine Consensus (SBC)

Definition (Strong Byzantine Consensus)

Given n processes, up to $f < n$ faulty, and V is the set of all proposed values, $V_c \subseteq V$ the set of values proposed by correct processes, the following must hold:

- **Termination:** Every correct process must eventually decide.
- **Agreement:** All correct processes decide the same value v_f.
- **Strong Validity:** The decision must be a value proposed by at least one correct process (*i.e.*, $v_f \in V_c$).

Common-Value Byzantine Consensus (CVBC): Validity: If $V_c = \{v\}$, then $v_f = v$.

Weak Byzantine Consensus (WBC): Validity: $v_f \in V$.

Table of Contents

1. Research Objective
2. Background
3. Proposed System Model
4. Byzantine Consensus
5. Proposed Solution
6. Conclusion

Proposed Solution

- We propose a novel shared object, called **StickyCAS** (Sticky Compare&Swap).
- Inspired by sticky bits[14], but generalized for *multi-valued consensus*.
- Acts as a non-corruptible shared memory primitive ensuring safety from Byzantine processes.

[14] Dahlia Malkhi et al. "Objects shared by Byzantine processes". In: *Distrib. Comput.* 16.1 (Feb. 2003), pp. 37–48. ISSN: 0178-2770.

StickyCAS Object: Specification

Maintains: A totally ordered list of up to n values.

Operations:

1. `StickyCAS-append`(*val*): Append *val* if possible.
   - Returns: *success*, *failed*, or *limit reached*.
   - Completes within one phase.
2. `StickyCAS-read`(*len*): Return the first *len* values in the list.
   - Can span across multiple phases.

StickyCAS Object: Properties

- Processes grouped in **warps**.
- In one warp phase:
  - At most one `StickyCAS-append` succeeds.
  - Other processes receive *failed*.
- `StickyCAS-read` is read-only and concurrent.
- Once a value is appended, it can never be deleted or overwritten.
- Crash tolerant implementation of StickyCAS is available in technical report[15].

[15] Chryssis Georgiou, Manaswini Piduguralla, and Sathya Peri. *Byzantine-Tolerant Consensus in GPU-Inspired Shared Memory*. 2025. arXiv: 2503.12788. URL: https://arxiv.org/abs/2503.12788.

Consensus Algorithm

Algorithm 1: Consensus Algorithm

```
1 DECIDE :
  /* On receiving first access to the StickyCAS by the
     Scheduler                                          */
2    StickyCAS-append(proposedValue) ;
  /* On receiving further accesses to the StickyCAS by the
     Scheduler                                          */
3    ProposedList ← StickyCAS-read(⌈n/p⌉);
4 return mode(ProposedList);
```

The *mode* of a set S is defined as the element $x \in S$ that occurs most frequently. In case of a tie, the smallest such element is chosen.

Correctness Assumptions

To prove consensus, we assume:

1. **Round-Robin Warp Scheduler** – Each warp scheduled fairly and cyclically[16].
2. **One** `StickyCAS-append` **per phase** – Fits within constant instructions.
3. **Controlled Global Memory Access** – Only via shared objects[17], i.e., in our solution StickyCAS.

[16] Hyeran Jeon. "GPU Architecture". In: *Handbook of Computer Architecture*. Ed. by Anupam Chattopadhyay. Springer Nature Singapore, 2025, pp. 531–559. ISBN: 978-981-97-9314-3.

[17] Idit Keidar Shir Cohen. "Tame the Wild with Byzantine Linearizability: Reliable Broadcast, Snapshots, and Asset Transfer". In: *DISC 2021*. 2021, 18:1–18:18.

Table of Contents

1. Research Objective
2. Background
3. Proposed System Model
4. Byzantine Consensus
5. Proposed Solution
6. Conclusion

Strong Byzantine Consensus (SBC)

Theorem (Strong Byzantine Consensus)

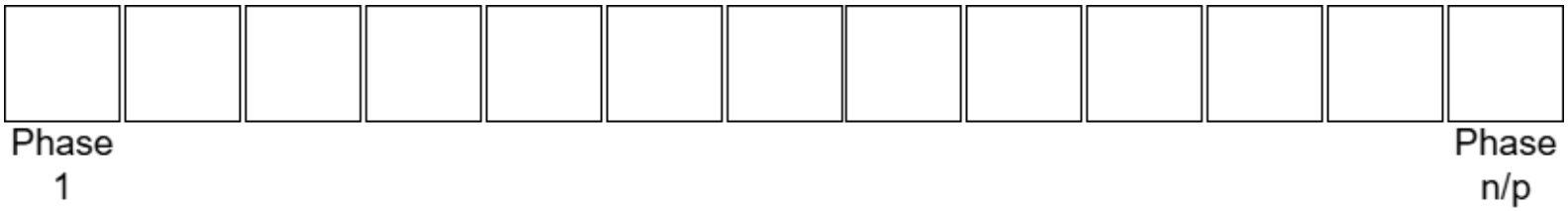
Proposed solution solves the Strong Byzantine Consensus problem within $\Theta(n^2/p^2 r)$ *phases, for* $f < \frac{n}{(|V_c|+1)p}$.

n	The total number of processes in the system.
V	The set of proposed values from all processes including the faulty ones.
V_c	The set of proposed values by correct processes.
V_f	Final consensus value decided by the correct processes
f	The maximum number of Byzantine processes that the system can tolerate.
p	The number of processes assigned per warp or phase.
r	The number of memory elements that can be read in a given phase.

Proof Intuition

Consider the case of $|V_c| = 2$:

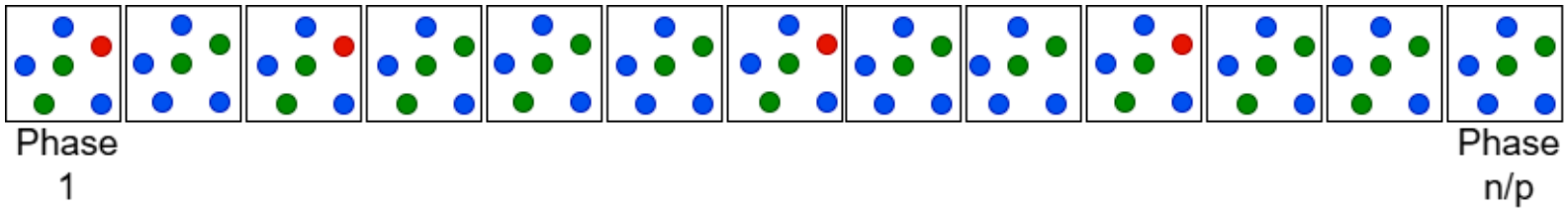
- We have n/p phases:



Proof Intuition

Consider the case of $|V_c| = 2$:

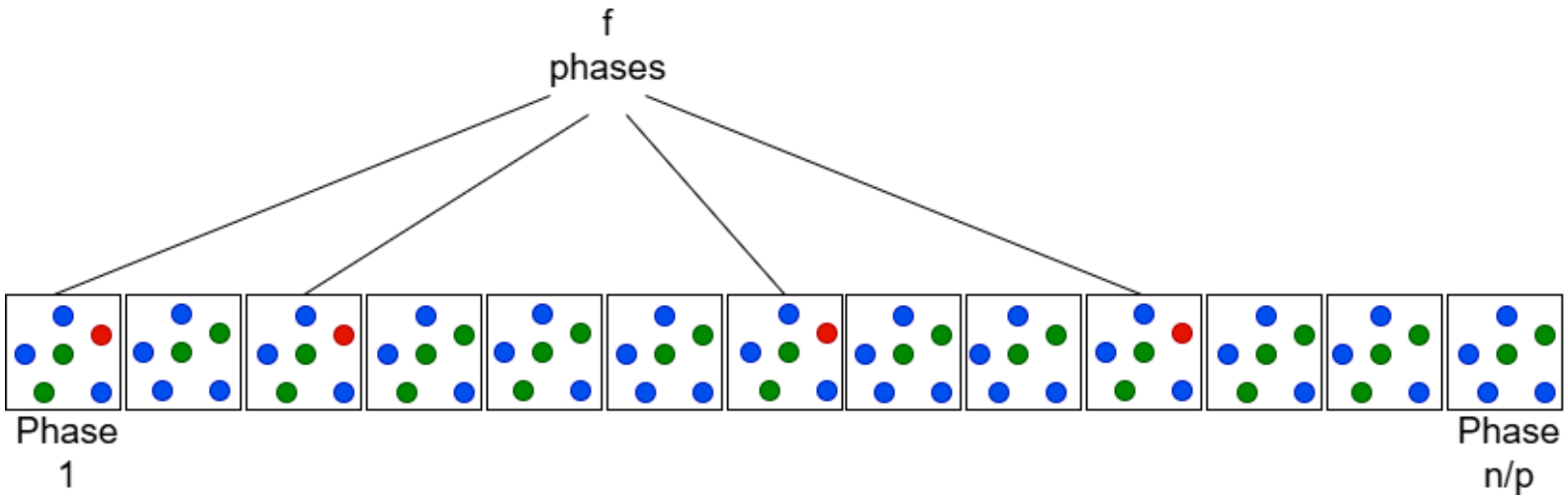
- We have n/p phases
- Red: Byzantine processes
- Blue: correct processes proposing value 1
- Green: correct processes proposing value 2



Proof Intuition

Consider the case of $|V_c| = 2$:

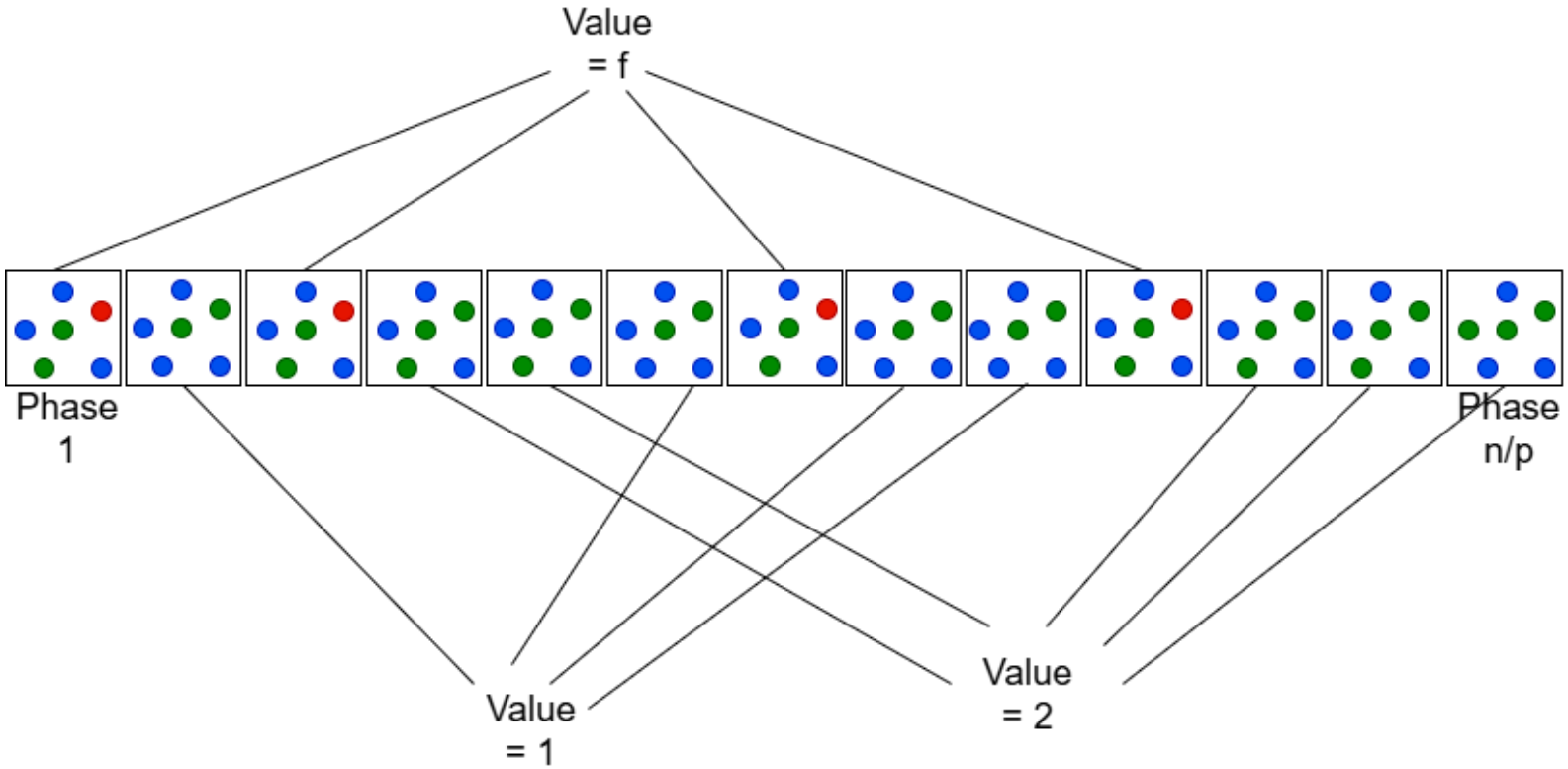
- At most f phases can have Byzantine processes
- $(n/p) - f$ phases will have all correct processes



Proof Intuition

Consider the case of $|V_c| = 2$:

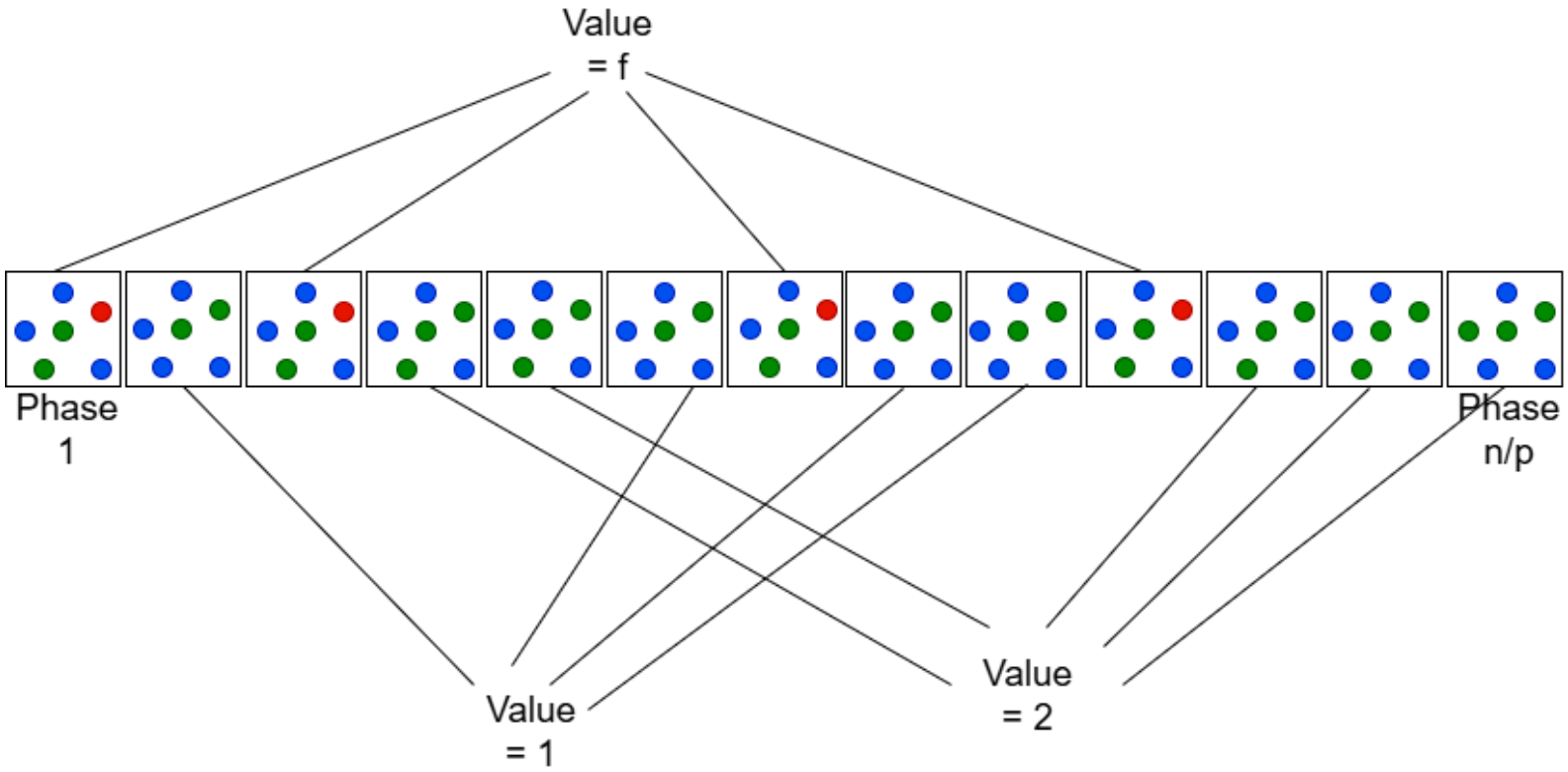
- Pessimistic scenario: f phases have Byzantine proposed values
- At least $((n/p) - f)/2$ phases will have winning value 1, while the remaining phases will have the winning value 2 (or vice versa).



Proof Intuition

Consider the case of $|V_c| = 2$:

- For consensus value to be 1 or 2, we need *mode* to be either 1 or 2 i.e., $((n/p) - f)/2 > f$
- Solving for f results in $f < n/(3p)$
- Extrapolating for all $|V_c|$, we derive $f < \frac{n}{(|V_c|+1)p}$



Conclusion

Summary:

- We introduced a **GPU-inspired Shared Memory Model** capturing unique architectural features.
- Proposed a novel object, **StickyCAS**, enabling Byzantine-tolerant consensus.
- Proved correctness and fault-resilience of our solution.

Future Directions:

- Explore inter-block computations with multiple schedulers.
- Investigate other fundamental problems under GPU-inspired models.
- Study optimal resilience bounds achievable in this setting.

Thank you

Why We Need StickyCAS

- In Byzantine settings, simple registers are unsafe:
  - A correct value can be overwritten by a faulty process.
  - Leads to inconsistent states and broken agreement.
- Sticky bits resist corruption but support only binary values.
- StickyCAS extends the idea to multi-valued consensus.

Why Assumptions are Reasonable

- RR scheduling: baseline GPU behavior[18].
- `StickyCAS-append` has constant cost (fits in one phase).
- `StickyCAS-read` may span multiple phases, consistent with GPU memory latency.
- Restricted access via objects is common for fault-tolerant designs.

[18] Hyeran Jeon. "GPU Architecture". In: *Handbook of Computer Architecture*. Ed. by Anupam Chattopadhyay. Springer Nature Singapore, 2025, pp. 531–559. ISBN: 978-981-97-9314-3.

Weak Byzantine Consensus (WBC)

Definition (Weak Byzantine Consensus)

Given n processes, up to $f < n$ faulty, and correct values V_c, the following must hold:

- **Termination:** Every correct process must eventually decide.
- **Agreement:** All correct processes decide the same value.

- Validity condition is dropped.
- Even if all correct processes propose the same value, the decision could be different.

Common Value Byzantine Consensus (CVBC)

Theorem (Common-value Byzantine Consensus)

Proposed solution solves solves the Common-value Byzantine Consensus problem within $\Theta(n^2/p^2 r)$ *phases, for* $f < \frac{n}{2p}$.