I believe the crux of systems research is identifying abstractions for systems that are intuitive for application developers and allow underlying implementations to meet application performance requirements. Developers prefer strong abstractions that make it easier to program correct applications [1, 8], such as those that mask concurrency or failures. However, strong abstractions impose high costs. Consequently, developers of applications with high performance requirements often either resort to weaker abstractions that increase code complexity or limit the scope of their application to reduce performance demands (e.g., by partitioning the userbase).

I aim to address this crux across the "distributed stack" from globally distributed to intra-datacenter to cross-cloud systems. As online services become more ubiquitous, they must reliably and securely process user data using distributed systems. At the same time, these services are supporting increasingly larger userbases with expectations of interactivity when sharing data. Existing abstractions for distributed systems require implementations that fundamentally cannot satisfy these evolving robustness and performance demands [4, 10, 11]. Furthermore, new hardware and software building blocks change these fundamental costs and create opportunities for co-designing systems with their underlying layers to meet performance demands. **My research focuses on new system designs that provide alternatives to prevailing abstractions with similar useability and reduced implementation costs.**

My dissertation research is an instantiation of this approach across three main contexts: (i) providing strong ordering guarantees for geo-replicated systems [5, 7, 14], (ii) providing robustness against malicious actors [9, 23], and (iii) co-designing systems with underlying hardware [6]. In addition to designing new systems that alleviate the tension between strong guarantees and high performance costs, I strive to produce open source, extensible prototypes with tooling for reproducing experimental evaluations. I also strive to contribute formally provable results about the nature of the design spaces for these systems. Through these complementary artifacts, my goal beyond improving today's systems is to contribute ideas that others can easily adopt in their own research.

1 Strongly Consistent Globally Distributed Storage

Strong consistency is an abstraction that guarantees operations appear to take effect in a total order consistent with a sequential system. While strong consistency simplifies distributed application development, it fundamentally costs more than weaker notions of consistency [4]. These costs are especially high in systems that are geo-replicated for robustness to geographically localized failures because of high latency between nodes. In my research, my collaborators and I argue that instead of foregoing strong consistency, these costs can be tamed to meet the performance requirements of a wider range of applications by rethinking the abstractions through which applications interact with these systems.

Serializable transactions under high contention. [7] Serializability is a form of strong consistency for transactional storage that provides the illusion of a centralized system executing transactions one at a time [20]. The simplicity of the serializability abstraction is a boon for developers, but a core performance issue is that under high contention, many transactions conflict and cannot execute concurrently. Existing solutions deal with concurrent conflicts *reactively* through aborting and retrying transactions with backoff. Instead of striving to more intelligently react, we argue that the *proactive* avoidance of concurrent conflicts enables new performance opportunities. Concurrent conflicts are the overlap of real-time windows that materialize when applications access data. Through this lens, we identify *transaction re-execution* as a means to rearrange

when applications access data, proactively realigning overlapping windows. Furthermore, we formalize this intuition by defining *serialization windows* and prove that no two windows on the same data can overlap in a serializable system. Morty is an interactive replicated transactional storage system that employs these ideas. Transaction re-execution for interactive transactions is possible because Morty uses the *continuation passing style* (CPS) for specifying transactions instead of the traditional imperative style. The CPS API imposes minimal additional burden on networked applications, as they typically are already written to avoid blocking on network requests. We implement and evaluate a prototype of Morty and find that it achieves $1.7x$-$96x$ the throughput of state-of-the-art systems [8, 25] with similar or better latency. We intend to open source the implementation and experiment scripts before this work appears at EuroSys'23.

Taming tail latency for linearizable storage. [5] Linearizability is another form of strong consistency that guarantees operations take effect in an order consistent with real-time [16]. Traditionally, linearizable storage systems are built using state machine replication [22] with consensus (SMR). While SMR allows for the implementation of any arbitrary deterministic state machine, it incurs high tail latency in practice. We argue that this performance woe is a fundamental consequence of the impossibility of consensus in an asynchronous environment [11]. Why does a linearizable storage system need the full power (and cost) of SMR? Instead, we propose the simpler abstraction of single-object reads, writes, and read-modify-writes (rmws) to rein in tail latency. Reads and writes comprise an overwhelming proportion of storage workloads and can be implemented with linearizability and guaranteed liveness [2]. Gryff is a linearizable key-value store that we designed which provides simple reads and writes with provable termination and rmws for strong synchronization (e.g., conditional updates). Gryff uses a novel ordering mechanism, *consensus-after-register timestamps (carstamps)*, to solve the challenge of ordering terminating reads and writes with (potentially) non-terminating rmws on the same data. Our implementation of Gryff reduces read tail latency to about 40% of MultiPaxos, an industry-standard approach for SMR, while maintaining similar median write/rmw latency. Gryff also reduces read tail latency to about 56% of EPaxos, a state-of-the-art geo-replicated SMR protocol, while trading off $2x$ higher median write latency. Our implementation and experiment scripts are available as an open source package [12].

Extracting global causality from linearizability. [14] Even with more specialized abstractions for linearizable storage, linearizability imposes read latency costs [10] that are too high for some applications. This is because linearizability must order all reads in an order consistent with real-time, even reads that are not potentially causally related. My colleagues and I argue that this requirement is too strong for many applications. Instead, we propose a new consistency model, *regular sequential consistency (RSC)*, that ensures operations are totally ordered consistently with potential causality, but that ensures only conflicting operations are ordered consistently with real-time. RSC is weaker than linearizability—and thus, circumvents the lower bound on linearizable read latency [10]—but is stronger than sequential consistency because it provides real-time ordering guarantees for conflicting writes and reads. The real-time ordering of conflicting operations enables the efficient composition of RSC systems by introducing *real-time barriers*. Moreover, we prove that RSC is invariant equivalent to linearizability, so applications programmed for linearizable systems run correctly on RSC systems without modification. We implement and evaluate an RSC variant of Gryff that reduces read tail latency to 40% of linearizable Gryff. Again, our prototype of Gryff-RSC is open source [13].

2 Malicious Robustness in the Cloud

Certain applications require robustness beyond the benign failures around which my primary dissertation work focuses. In financial and healthcare applications, malicious actors may seek to attack services and end users. Safety guarantees against such adversaries are critical, but unsurprisingly impose additional performance costs. My colleagues and I advocate that safety is achievable with good performance by choosing appropriate abstractions for these systems.

Inverting the blockchain-database abstraction. [23] Permissioned blockchains are a rapidly growing solution to the problem of data sharing among mutually distrustful parties. They provide applications the abstraction of a totally ordered log that tolerates Byzantine failures [17] using SMR. Though powerful, Byzantine fault tolerant (BFT) SMR struggles to scale to the performance requirements of many applications because the totally ordered log abstraction is limited. My colleagues and I argue that the abstraction of a partially ordered serializable database is a better fit for these applications because they anyway use the totally ordered log to implement a database. By targeting the abstraction of a database, a system is not required to totally order non-conflicting transactions - flexibility that can be leveraged for improved performance. Basil is a BFT transactional and replicated key-value store that embraces this idea. By using database techniques for managing conflicts, Basil achieves $3.5x$-$5x$ higher throughput than baseline systems like PBFT and HotStuff that implement a database on top of the totally ordered log abstraction. Our prototype and experiment scripts are available as an open source package [3].

Leveraging transactions to hide access patterns. [9] Beyond Byzantine influence, malicious actors may attempt to access end users' private information by analyzing interactions between clients and system services. Such an attack is possible in the growing cloud ecosystem, where applications offload storage to third parties. While techniques like oblivious random access memory (ORAM) provide security against these attacks, existing ORAMs do not guarantee durability and afford only limited concurrency. To bring the security of ORAM to cloud applications with high robustness and performance requirements, my colleagues and I propose to strengthen the ORAM abstraction from simple reads and writes to serializable transactions. Surprisingly, providing the stronger transactional abstraction allows amortizing the cost of expensive ORAM and failure recovery operations across many transactional requests. Obladi is a transactional key-value store that leverages this observation. Obladi achieves within $5x$–$12x$ the throughput of non-secure MySQL on standard OLTP benchmarks like TPC-C with about $70x$ the latency.

3 Co-designing Systems with Underlying Layers

Much of my work focuses on the boundary between end user applications and the services that support them. I find that research opportunities also exist at the boundary between services and their underlying building blocks (e.g., the OS and network) as the abstractions and performance tradeoffs of underlying building blocks evolve with new hardware.

Strengthening the RDMA interface. [6] RDMA NICs are increasingly available and provide the opportunity to accelerate services by bypassing remote CPUs when sharing data. Yet a common theme is that adapting services to run on RDMA requires complex—and costly—contortions. Its simple READ/WRITE/CAS interface necessitates additional network round trips or involving receiver CPUs to implement distributed services. We propose that a small set of generic extensions to the RDMA interface is needed in order to fully utilize the capabilities of this new hardware. We propose the addition of *indirection*, *allocation*, *operation chaining*, and an *enhanced*-CAS, which enable efficient remote access patterns like data structure navigation, out-of-place

updates, and concurrency control. My colleagues and I argue that PRISM is feasible to implement in hardware because it reuses existing microarchitectural designs and we implement both a Snap-inspired [18] software prototype and a smart-NIC prototype running on a Mellanox BlueField [19]. We demonstrate that these primitives enable the implementation of more efficient RDMA services, such as in-memory storage, replicated storage, and transactional storage. We implement prototypes of these RDMA storage services and we find, in each case study, that PRISM improves either latency or throughput relative to baselines without sacrificing either.

4 Future Work

I plan to continue my research in the field of distributed systems, focusing on the challenges of providing strong consistency to applications with growing performance requirements.

Abstracting cloud services. Cloud computing is trending toward a provider agnostic ecosystem as users seek robustness to major single-provider outages and seek to leverage cross-provider competition to reduce costs. Providers are also working to enable cross-cloud portability to grow their market share. As a first step toward this trend, we must understand the fundamental costs associated with using cloud services as underlying building blocks. For example, clouds offer both strongly and weakly consistent storage systems with varying performance and financial costs. How can we design a storage layer above these services that provides strong consistency while meeting application performance requirements and minimizing financial costs? In such a system, how can the storage layer intelligently replicate data not only across services, but across providers to achieve robustness to major outages and further minimize costs? Sufficiently addressing these goals may benefit from co-design between this provider-agnostic storage layer and the underlying cloud service abstractions.

Modernizing serializable system designs. The conception that serializability is too expensive for replicated and geo-distributed deployments contributed to the rise of NoSQL systems in the 00s. However, orders of magnitude increases in network speeds on commodity hardware has made it possible to build in-memory serializable systems that can meet high performance requirements. As Morty [7] highlights, bottlenecks in the serializability abstraction are now primary performance limiters for traditional concurrency controls. Beyond the serialization windows identified in the Morty project, what other fundamental bottlenecks exist and how do we re-architect systems to minimize their impact? Similar to Morty, the Meerkat [24] project identifies the Zero Coordination Principle (ZCP) as a design goal for achieving optimal performance on low contention workloads. To what extent can serializable systems optimize for both high and low contention workloads? For example, is it fundamentally possible for a system to optimize the alignment of serialization windows and enforce the ZCP?

Relationship between liveness and performance. Consensus-based systems formally lack liveness and this informally manifests as increased tail latency. I would like to investigate to what extent this is a fundamental connection. Through a combination of formal methods, statistical modeling, and simulation, such a study could further motivate building systems like Gryff [5] that provide provable liveness guarantees. I also see opportunities for applying the Gryff approach to other types of systems. For example, message-broking systems [21] provide a queue abstraction to applications, which is fundamentally weaker [15] than the consensus-based SMR protocols that underpin popular implementations. More generally, is it possible to develop a black-box approach that takes in a sequential specification and synthesizes an implementation that judiciously uses consensus only for the subset of the specification's operations that need it?

References

[1] Phillipe Ajoux, Nathan Bronson, Sanjeev Kumar, Wyatt Lloyd, and Kaushik Veeraraghavan. Challenges to Adopting Stronger Consistency at Scale. In *ACM SIGOPS Workshop on Hot Topics in Operating Systems (HotOS)*, 2015.

[2] Hagit Attiya, Amotz Bar-Noy, and Danny Dolev. Sharing Memory Robustly in Message-Passing Systems. *Journal of the ACM (JACM)*, 42(1):124–142, 1995.

[3] `https://www.github.com/fsuri/Basil_SOSP21_artifact/`, 2021.

[4] Eric A Brewer. Towards Robust Distributed Systems. In *ACM Symposium on Principles of Distributed Computing (PODC)*, 2000.

[5] Matthew Burke, Audrey Cheng, and Wyatt Lloyd. Gryff: Unifying Consensus and Shared Registers. In *USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2020.

[6] Matthew Burke, Sowmya Dharanipragada, Shannon Joyner, Adriana Szekeres, Jacob Nelson, Irene Zhang, and Dan R. K. Ports. PRISM: Rethinking the RDMA Interface for Distributed Systems. In *ACM Symposium on Operating System Principles (SOSP)*, 2021.

[7] Matthew Burke, Florian Suri-Payer, Jeffrey Helt, Lorenzo Alvisi, and Natacha Crooks. Morty: Scaling Concurrency Control with Re-Execution. In *ACM SIGOPS European Conference on Computer Systems (EuroSys)*, 2023.

[8] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, JJ Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. Spanner: Google's Globally-Distributed Database. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2012.

[9] Natacha Crooks, Matthew Burke, Ethan Cecchetti, Sitar Harel, Rachit Agarwal, and Lorenzo Alvisi. Obladi: Oblivious Serializable Transactions in the Cloud. In *USENIX Symposium on Operating Systems Design and Implementation (OSDI)*, 2018.

[10] Partha Dutta, Rachid Guerraoui, Ron R. Levy, and Arindam Chakraborty. How Fast can a Distributed Atomic Read be? In *ACM Symposium on Principles of Distributed Computing (PODC)*, 2004.

[11] Michael J. Fischer, Nancy A. Lynch, and Michael S. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM (JACM)*, 32(2):374–382, 1985.

[12] `https://www.github.com/matthelb/gryff/`, 2020.

[13] `https://www.github.com/princeton-sns/gryff-rs`, 2021.

[14] Jeffrey Helt, Matthew Burke, Amit Levy, and Wyatt Lloyd. Regular Sequential Serializability and Regular Sequential Consistency. In *ACM Symposium on Operating System Principles (SOSP)*, 2021.

[15] Maurice Herlihy. Wait-Free Synchronization. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 13(1):124–149, 1991.

[16] Maurice P. Herlihy and Jeannette M. Wing. Linearizability: A Correctness Condition for Concurrent Objects. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 12(3):463–492, 1990.

[17] Leslie Lamport, Robert Shostak, and Marshall Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[18] Michael Marty, Marc de Kruijf, Jacob Adriaens, Christopher Alfeld, Sean Bauer, Carlo Contavalli, Michael Dalton, Nandita Dukkipati, William C. Evans, Steve Gribble, Nicholas Kidd, Roman Kononov, Gautam Kumar, Carl Mauer, Emily Musick, Lena Olson, Erik Rubow, Michael Ryan, Kevin Springborn, Paul Turner, Valas Valancius, Xi Wang, and Amin Vahdat. Snap: A Microkernel Approach to Host Networking. In *ACM Symposium on Operating System Principles (SOSP)*, 2019.

[19] `https://www.nvidia.com/en-us/networking/products/data-processing-unit/`, 2022.

[20] Christos H Papadimitriou. The Serializability of Concurrent Database Updates. *Journal of the ACM (JACM)*, 1979.

[21] `https://www.rabbitmq.com/`, 2022.

[22] Fred B. Schneider. Implementing Fault-Tolerant Services Using the State Machine Approach: A Tutorial. *ACM Computing Surveys (CSUR)*, 22(4):299–319, 1990.

[23] Florian Suri-Payer, Matthew Burke, Zheng Wang, Yunhao Zhang, Lorenzo Alvisi, and Natacha Crooks. Basil: Breaking up BFT with ACID (transactions). In *ACM Symposium on Operating System Principles (SOSP)*, 2021.

[24] Adriana Szekeres, Michael Whittaker, Jialin Li, Naveen Kr Sharma, Arvind Krishnamurthy, Dan RK Ports, and Irene Zhang. Meerkat: Multicore-Scalable Replicated Transactions Following the Zero-Coordination Principle. In *ACM SIGOPS European Conference on Computer Systems (EuroSys)*, 2020.

[25] Irene Zhang, Naveen Kr Sharma, Adriana Szekeres, Arvind Krishnamurthy, and Dan RK Ports. Building Consistent Transactions with Inconsistent Replication. In *ACM Symposium on Operating System Principles (SOSP)*, 2015.