Abstract
Distributed systems are hard to build. Existing validation methods fall under proof-based verification of the high-level description, and black box testing of the implementation. We introduce the use of P, a DSL for writing specifications which compile into executable production code, to build a distributed system. We develop the Raft protocol in P and extend P to interface the generated implementation with a K/V application for proof of concept.

Background
P is a language for writing asynchronous, event-driven code. Systems are specified in P as a collection of interfacing state machines. The semantics are interpretable by the model checker, and alternatively can be generated into executable C or C# code.

Modeling Raft In P
Servers receive un-coordinated events sent from both the client and other servers. These can arrive at tricky times like server shutdown and leader election, or come from invalid sources. We maintain leader, candidate, and follower states, and instead capture this complexity through event handlers. Outlining novel changes below:

- Logical clock hybridization to fit P event-driven code to concept of heartbeats and leader election.
- Implemented Single-server cluster membership changes in lieu of original joint consensus approach to reduce complexity.
- Created cluster manager to abstract servers from clients and ensure servicing of requests on top of log matching guarantee.
- 100% native P code, no foreign, unverifiable functions.

Interfacing with External Environment
The application must generate new machines, queue events, and store pointers to those machines. See our C/Go runtime framework for multi-threaded communication with the Cluster Manager below.

Discussion
PRaft has promising performance against SOTA etcd, and guaranteed bisimulation of model and code. Lack of log compaction accounts for slowdown over time of PRaft, and the logical clock allowed for exact \( \Delta \) calculation.

Future Work
- RAFT Improvements
- Log compaction
- Multi-process P
- Overriding send, new
- Scheduler prioritization
- Execution Interface
- Export event structure
- Stack traces
- Run more evaluations
- Event overhead

Conclusion
We have shown that building a distributed consensus protocol using automatic generation from a formal model is practically achievable. We have also developed a framework to interface the resultant generated code with user applications.

References