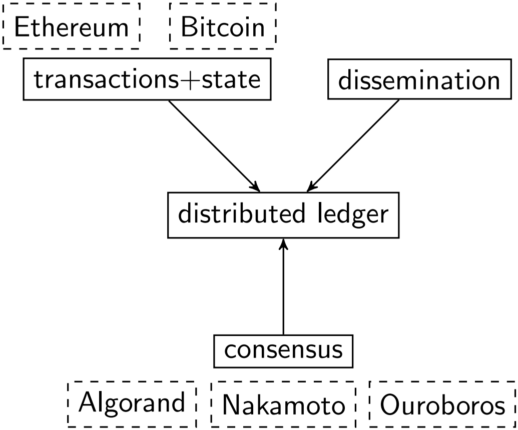
Kaizen: Building a Performant Blockchain System Verified for Consensus and Integrity

Faria Kalim[†], Karl Palmskog[*], Jayasi Mehar[‡], Adithya Murali[†],
P. Madhusudan[†] and Indranil Gupta[†]

[†]University of Illinois at Urbana-Champaign
[*]KTH; work done while at UT Austin and UIUC
[‡]Facebook; work done while at UIUC

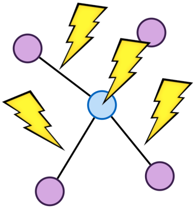
Blockchains and Cryptocurrencies



Consensus Protocol Challenges

Distributed protocols need to handle:

- communication delays (asynchrony)
- node crashes, corruption
- message drops, duplication, forging



Protocol implementation challenges:

- conformance to protocol specification
- node-local performance
- absence of bugs compromising safety

Consensus System Formal Verification

Project	**Paper**	**Protocol**	**Tool**	**LOC**
Disel	POPL '18	2-phase commit	Coq	5k+
Verdi Raft	CPP '16	Raft	Coq	50k+
Velisarios	ESOP '18	PBFT	Coq	50k+
<u>Ironfleet</u>	SOSP '15	Paxos	<u>Dafny</u>	20k+
<u>Toychain</u>	CPP '18	proof-of-X	<u>Coq</u>	10k+

Interactive vs. Mostly-Automated Verification

Coq proof assistant

- \- much training required
- \+ explicit proofs
- \+ many libraries
- \- purely functional (extraction to OCaml/Haskell)

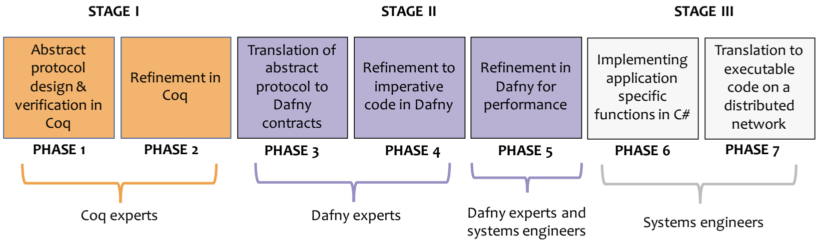
Dafny verification environment

- \+ less training required
- \- implicit proofs
- \- few libraries
- \+ functional & imperative (C# code generation)

Our Contributions

- novel combination of **Coq & Dafny** to build performant and verified blockchain system, Kaizen
- methodology based on continuous refinement
  - adapted & instantiated Coq model
  - translated Coq code (not proofs) to Dafny imperative code
  - refined C# code and linked to network shim
- performed evaluation measuring minting and consensus time

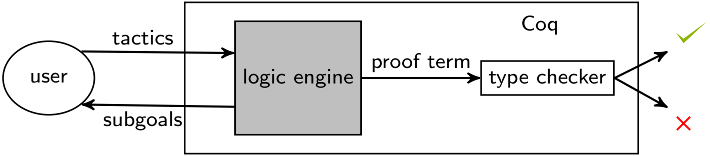
Methodology Overview*



*system is fully verified until Stage III

Stage I: Modeling and Verification Using Coq

1. encode system in higher-order functional language (Gallina)
2. prove specification **interactively** using powerful **tactics**
3. check soundness of every low-level step



Toychain Examples

```
Record Block := mkB { prevBlockHash : Hash;
  txs : seq Transaction; proof : VProof }.

Record State := mkS { id: Address; peers: seq Address;
 forest: map Hash Block; txpool: seq Transaction }.

Definition valid_chain_block (bc:seq Block) (b:Block):=
 VAF (proof b) bc (txs b) && all [pred t | txValid t bc] (txs b).
```

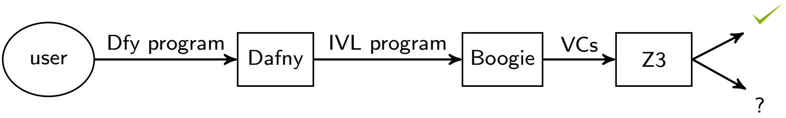
Toychain Results and Extensions

- Toychain proves that in quiescent state, all nodes know the same (canonical) chain
- we added support for coinbase transactions
- we added checking of proof-of-work validity of chains
- we changed Toychain nodes to avoid unncessary messages

All changes are proof-preserving and now merged into Toychain.

Stage II: Refinement and Verification Using Dafny

1. encode programs and their contracts in imperative language
2. try to prove automatically that contracts are fulfilled
3. add more annotations if necessary



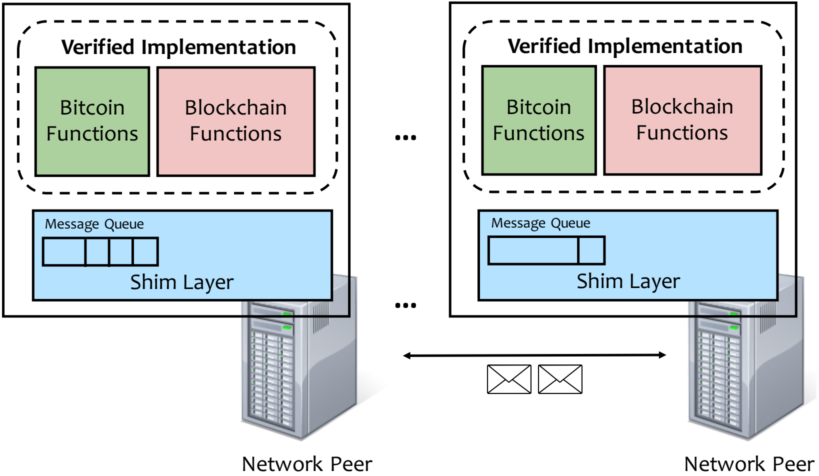
Dafny Examples

```
datatype Block = Block(prevBlockHash: Hash,
 txs: seq<Transaction>, proof: VProof)

datatype State = Node(id: Address, peers: seq<Address>,
 forest: map<Hash,Block>, txpool: seq<Transaction>)

class StateImpl {
  var id : Address;
  var peers : ...; var forest : ...; var txpool : ...;
  ghost var st: State;

  predicate Valid() { ...}

  method ProcMsgImpl(from: Address, msg: Message, ts: Timestamp)
   returns (pt: seq<Packet>)
  requires Valid();
  ensures Valid();
  ensures st =procMsg(old(st), from, msg, ts).0;
  ensures pt =procMsg(old(st), from, msg, ts).1;
  { ...}
}
```

Stage III: Refinements in C#

- block and proof-of-work generation
- define and inject miner rewards
- store pre-computed chains
- add network shim based on UDP

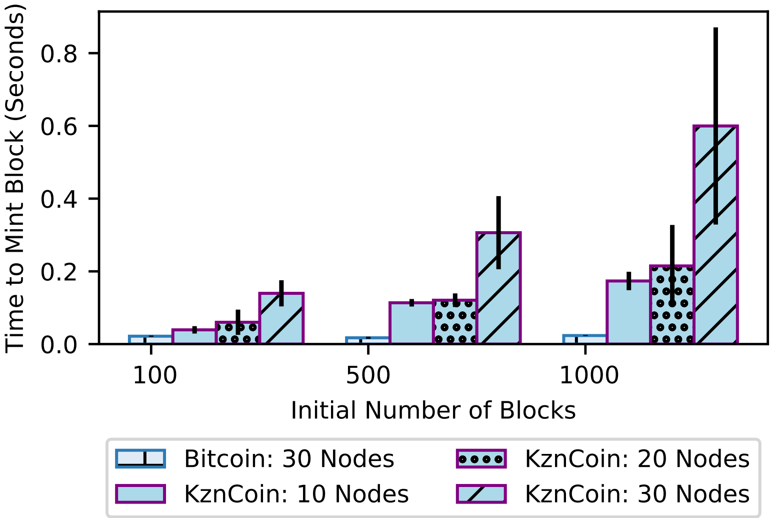
Implementation Architecture



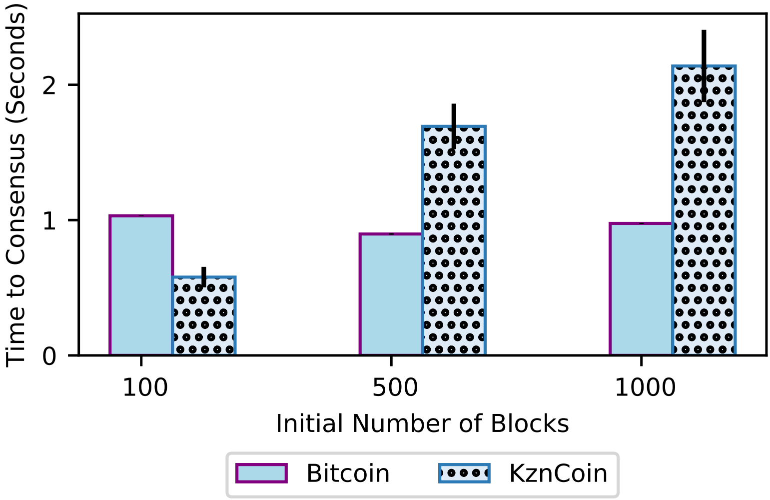
Evaluation Setup

- metrics: block minting time and consensus time
- use 30-node cluster of 2.4GHz processors w/ 64GB RAM
- baseline: performance of stock Bitcoin implementation
- workload: traces of arrival times of 50 transactions from realistic dataset

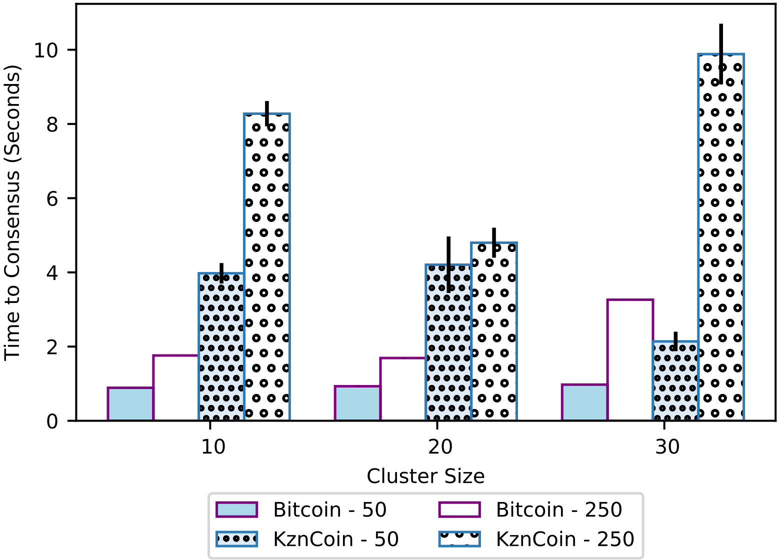
Evaluation: Block Minting Time



Evaluation: Consensus Time



Evaluation: Scalability



Components and Effort

Component	**Lines of Code**
Coq refinement	$\approx$ 1k
Dafny refinement	$\approx$ 5k
C# refinement	$\approx$ 1k
C# network shim	$\approx$ 4k

Development effort $\approx$ 6 person months across four people

Lessons Learned and Future Work

- holistic expertise necessary in Coq/Dafny/systems for Kaizen
- "easy" change can require large changes at earlier stages
- local node computation took most effort to optimize (rather than network messaging)
- future Coq proofs of Toychain Byzantine tolerance transferrable to Kaizen (see WIP by Gopinathan and Sergey, CoqPL '19)

Conclusion

- system development methodology combines interactive and mostly-automated verification, **Coq & Dafny**
- verified executable blockchain system **Kaizen**
- evaluation gives encouraging results on performance

More information:

- GitHub: `https://github.com/palmskog/kaizen`
- contact me: **Karl Palmskog** `palmskog@kth.se`