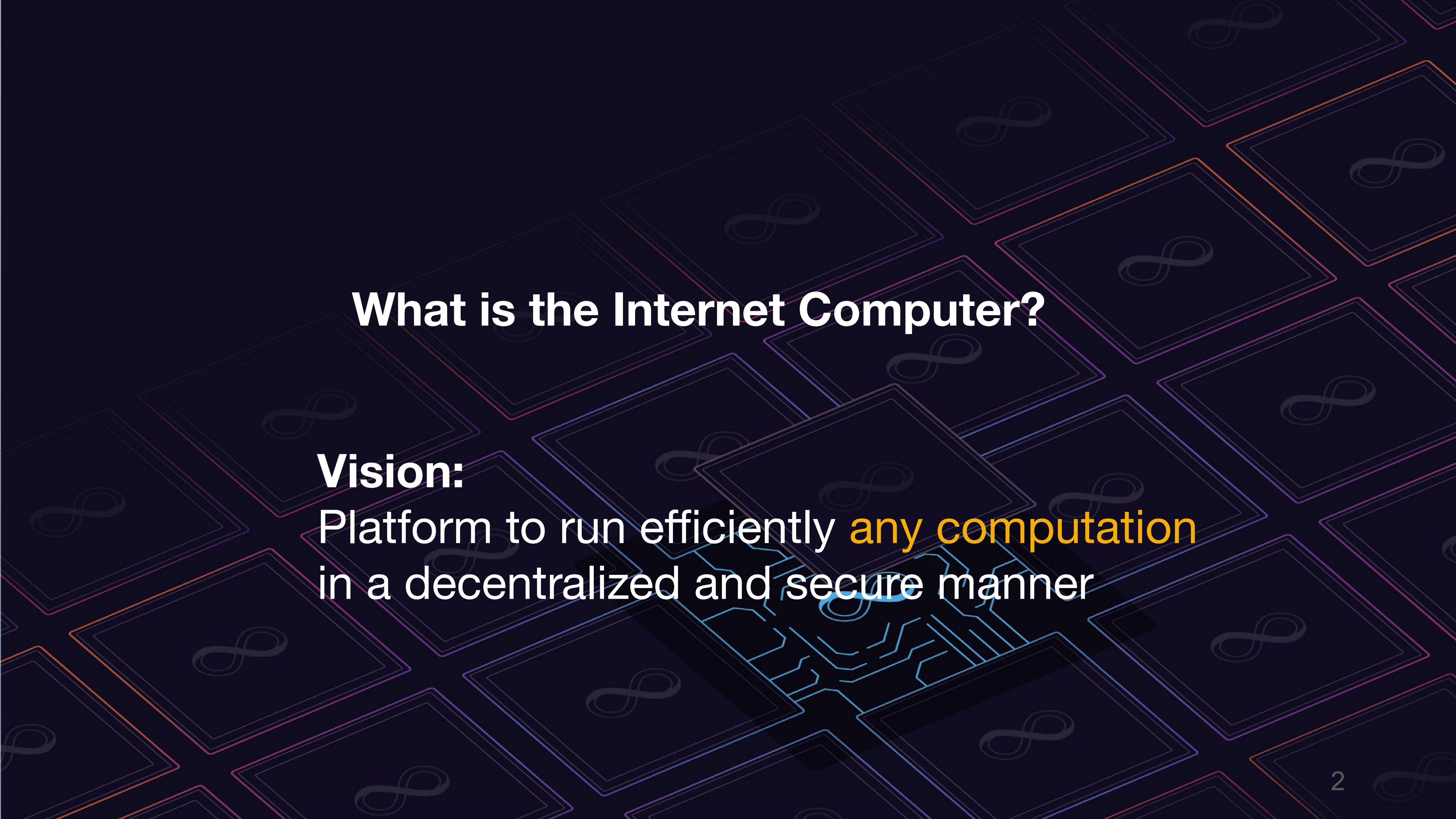
Decentralized and Stateful Serverless Computing on the Internet Computer Blockchain

Maksym Arutyunyan, Andriy Berestovskyy, Adam Bratschi-Kaye, Ulan Degenbaev, Manu Drijvers, Islam El-Ashi, Stefan Kaestle, Roman Kashitsyn, Maciej Kot, Yvonne-Anne Pignolet, Rostislav Rumenov, Dimitris Sarlis, Alin Sinpalean, Alexandru Uta, Bogdan Warinschi, Alexandra Zapuc

Presentation by: Alexandru Uta



What is the Internet Computer?

Vision:
Platform to run efficiently any computation
in a decentralized and secure manner

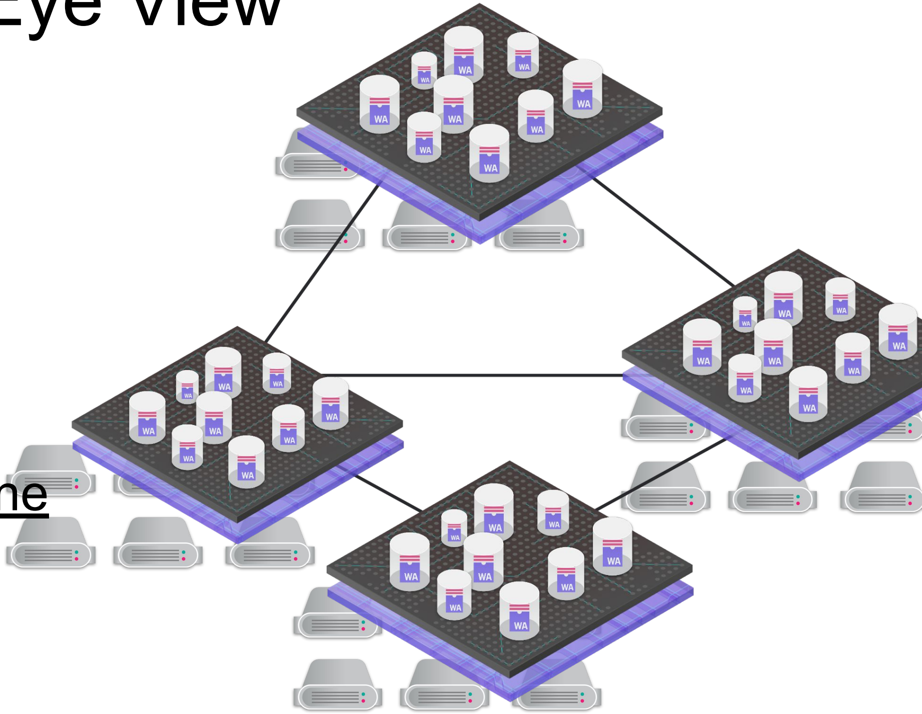
Internet Computer – Bird’s Eye View

- Collection of replicated state machines

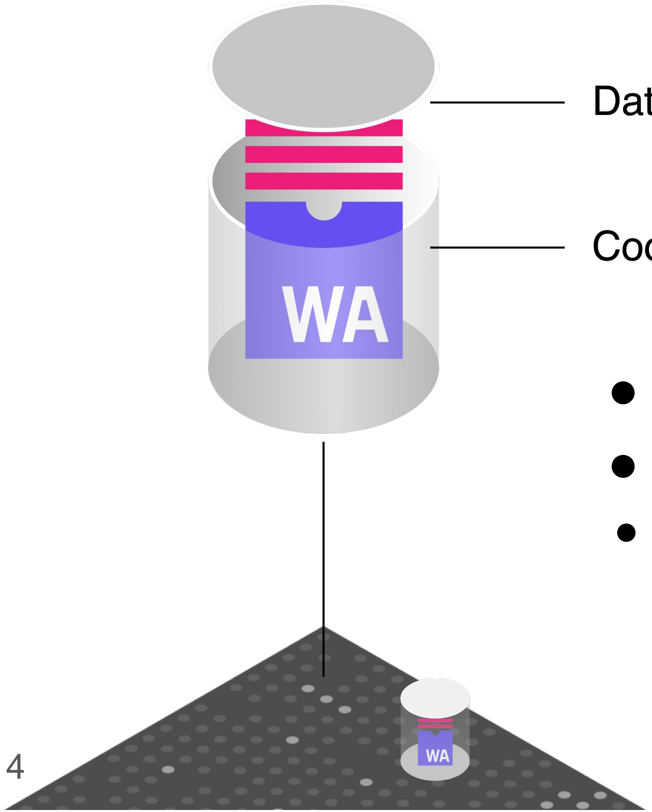
- Nodes in independent data-centers
- Nodes are partitioned into subnets

- Each subnet is a replicated state machine

- Each subnet runs canisters (smart contracts)



Canister Smart Contracts

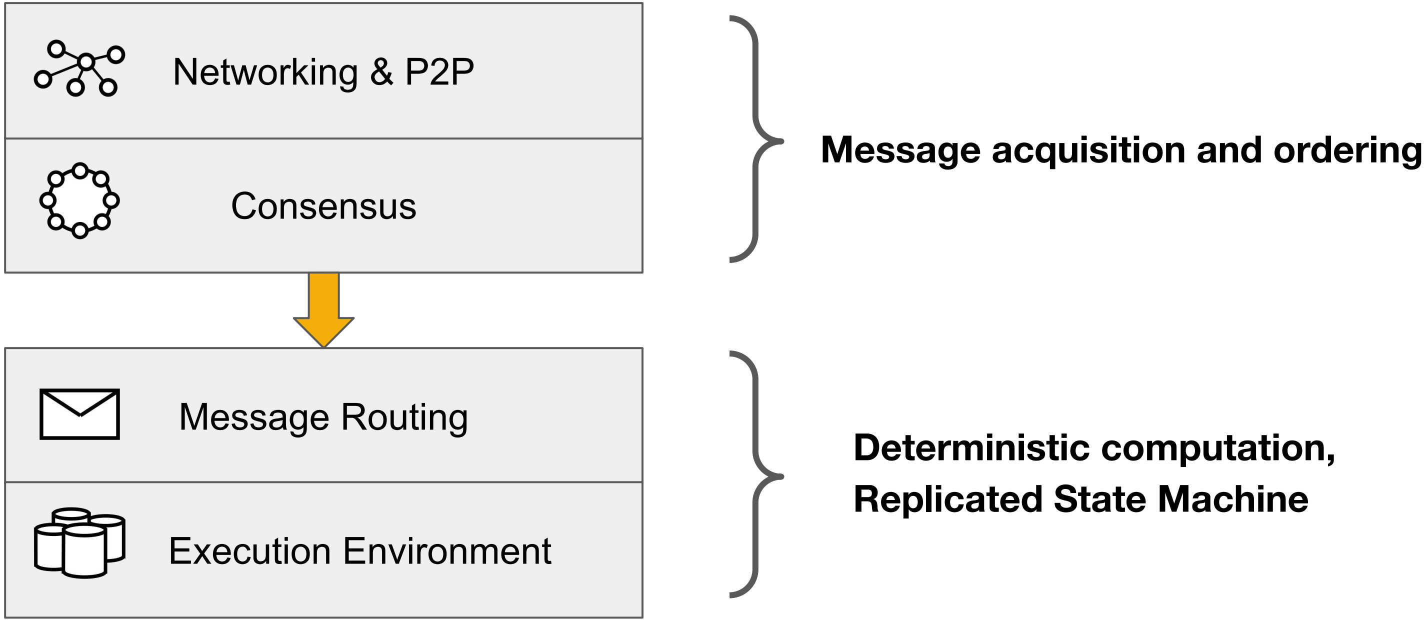


Data: Memory pages

Code: WebAssembly bytecode

- Can be programmed in Rust, Motoko, Python, Javascript
- 64 GB memory/storage
- Calls possible to:
    - canisters on the same/different subnets
    - exterior (http outcalls)

IC Layers



Networking & P2P

Consensus

Message Routing

Execution Environment

Message acquisition and ordering

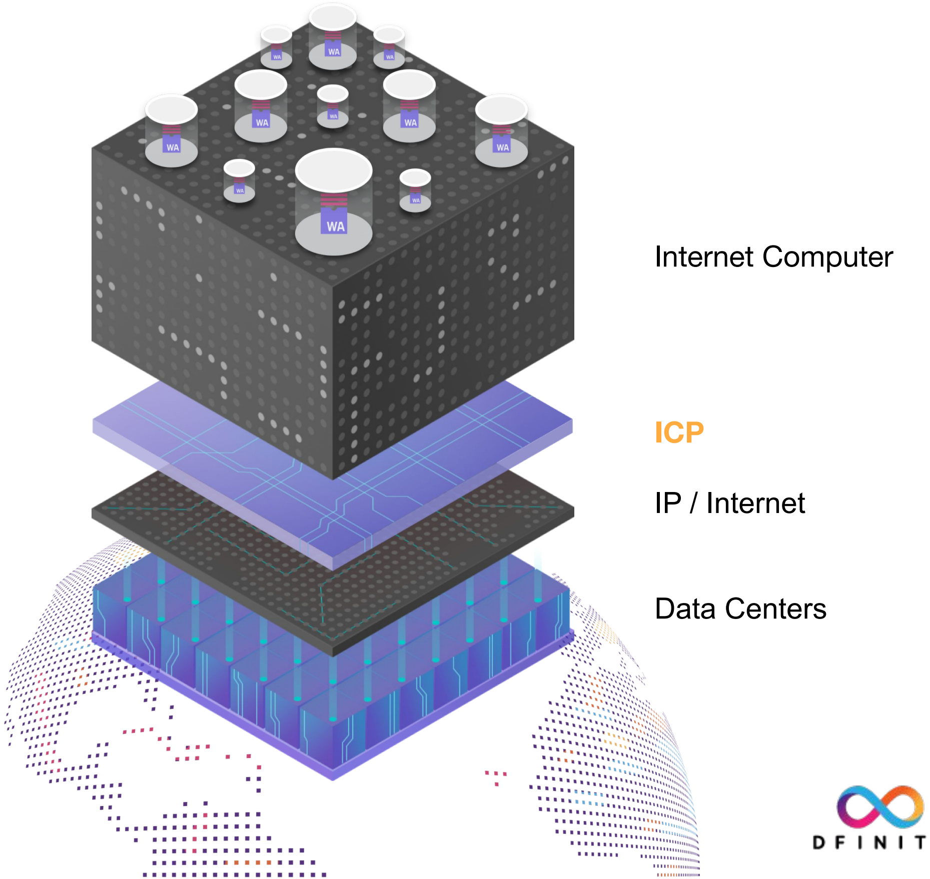
Deterministic computation,
Replicated State Machine

Calls exposed by canisters

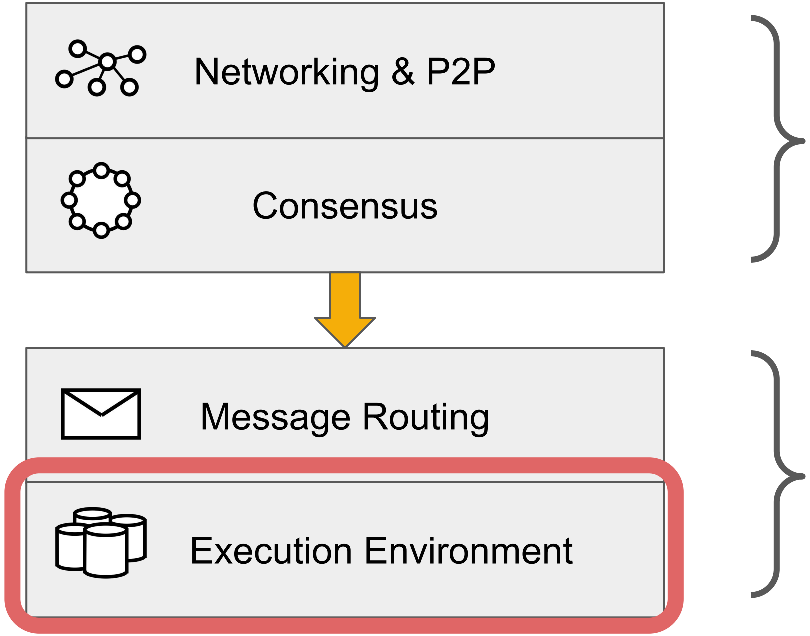
Update	**Query**
Slow (goes through consensus)	Fast (no consensus)
Persists state changes	State changes discarded
Replicated	Non-replicated

What’s different about the Internet Computer?

- **Byzantine** fault tolerance
    - Up to f malicious out of 3f + 1 nodes
    - Individual **nodes cannot be trusted**
- **Geo-replicated**
    - 549 nodes, DCs in 18 countries
- **Decentralized** (88 node providers)
    - DFINITY (or any other entity) can only access their own nodes
- **Self-governing**
    - No single entity in control of the IC
    - Votes to apply changes



Execution – Focus of This Presentation



Networking & P2P

Consensus

Message Routing

Execution Environment

Message acquisition and ordering

Deterministic computation,
Replicated State Machine

Systems Challenges

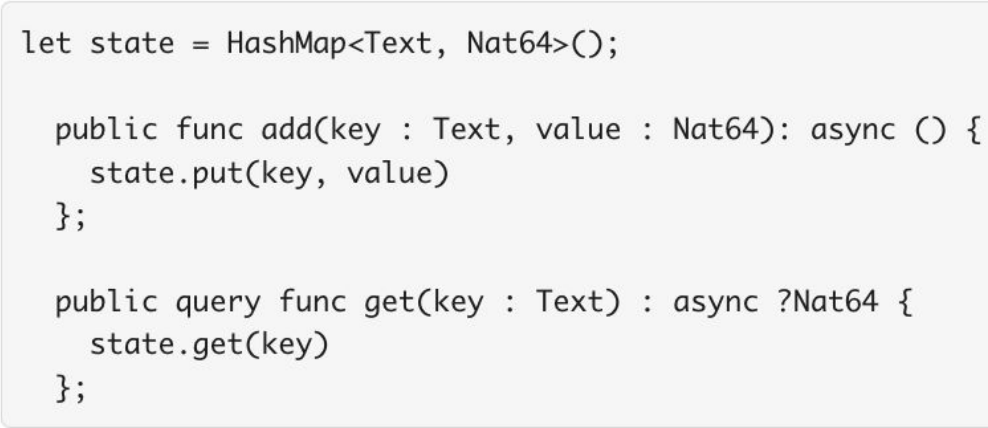
1. Statefulness
2. Deterministic Scheduling
3. Scalability*
4. Security*

*not in this talk, details in the article

Challenge 1 – Statefulness

- statefulness through *orthogonal persistence*
- Canisters are “forever-running” processes
- State is kept after replicated message execution
- No (or little) programmer work to achieve this

Motoko Key-Value store Canister

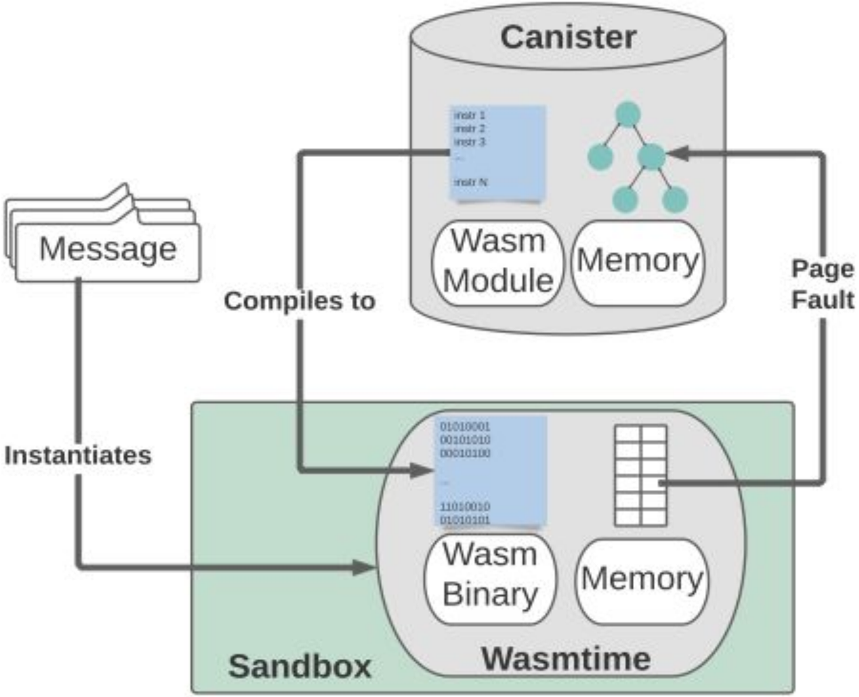


```
let state = HashMap<Text, Nat64>();

  public func add(key : Text, value : Nat64): async () {
    state.put(key, value)
  };

  public query func get(key : Text) : async ?Nat64 {
    state.get(key)
  };
```

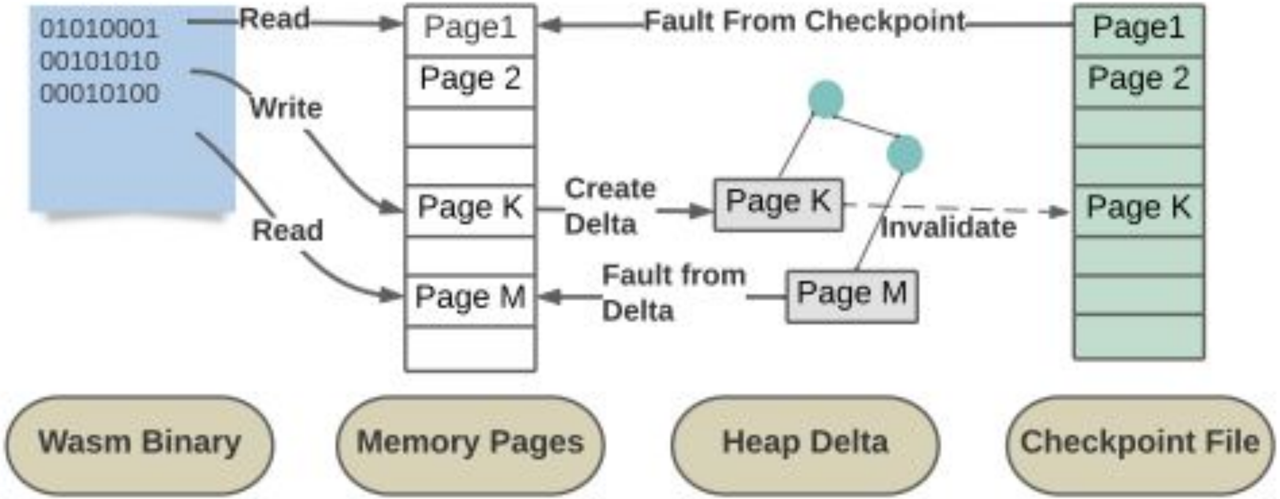
Motoko
Playground

Execution of Canister Messages

- Execution of message instantiates Wasmtime VM running in sandbox
- Execution happens in rounds
- Each canister can execute 1 or more messages per round
- Every N = 500 rounds, canister state is checkpointed



Memory Architecture



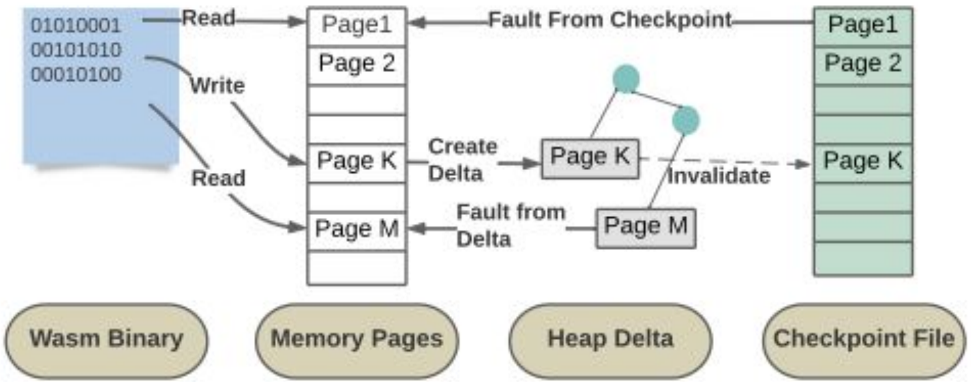
Statefulness: Tracking Changes

For Performance: Map memory pages on demand

- Page protection & signal handler to catch accesses

Canister call

1. Initially: no page is mapped
2. Read access: page fault → map r/o
3. Write access: page fault → create delta, (re-)map r/w,



Memory Optimizations

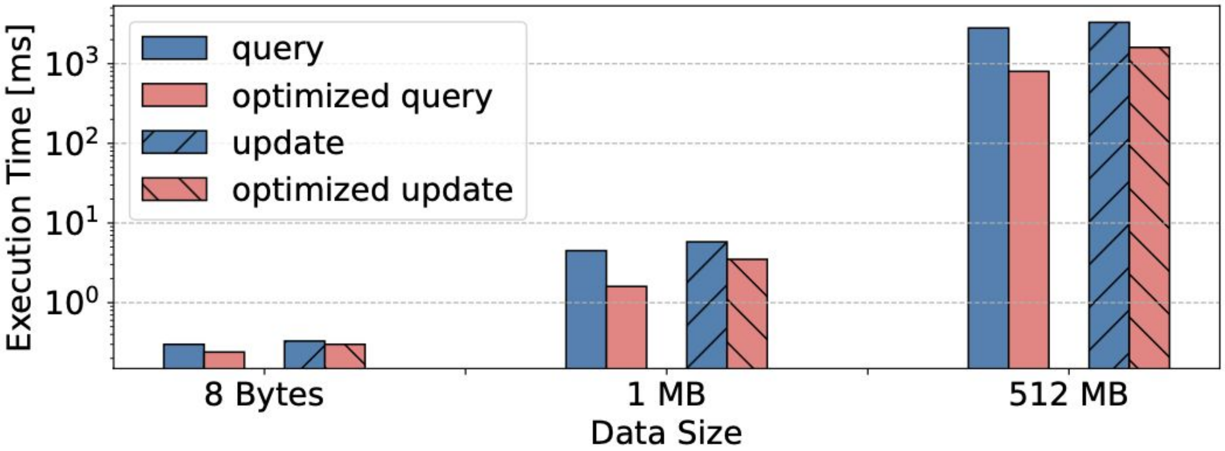


Figure 4: The performance improvement given by memory faulting optimizations (lower is better). Note the logarithmic vertical axis. Speedups range from 1.25X to 3.5X.

Challenge 2 – Deterministic Scheduling

- **(Sub-)Challenge 1: Determinism**

  Replicated state machine → all nodes in the state machine execute the same computation, in the same order

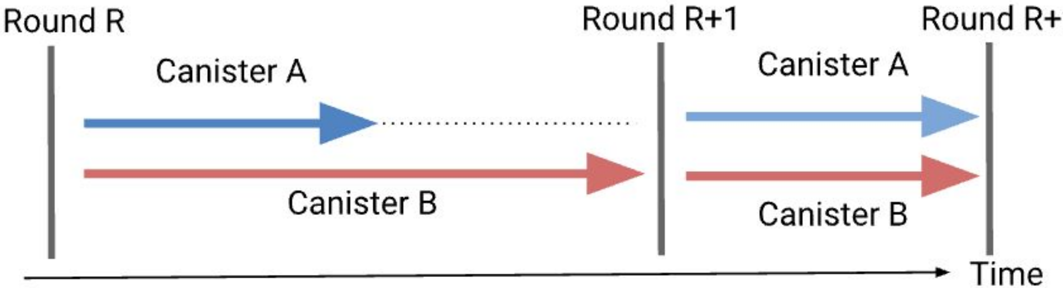
- **(Sub-)Challenge 2: Charging**
  Because of replicated computation can't use "time", but instruction counts

- **(Sub-)Challenge 3: Fairness, DoS protection**

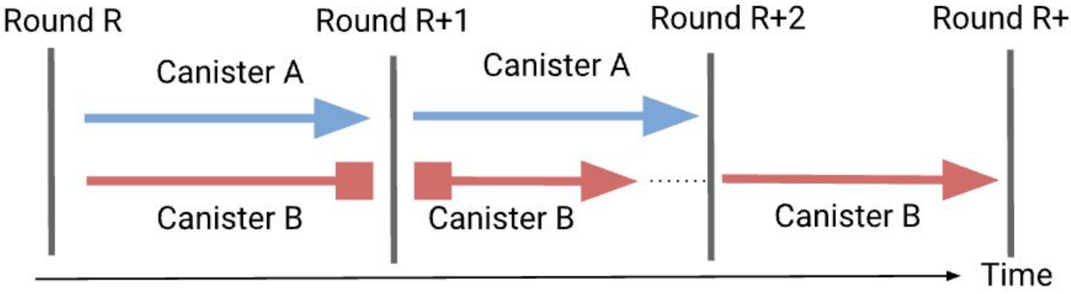
- **(Sub-)Challenge 4: Scale,** need to support 100K+ canisters

Challenge 2 – Deterministic “Time” Slicing

w/o slicing



w/ slicing



* does not work on time units, but rather on numbers of instructions

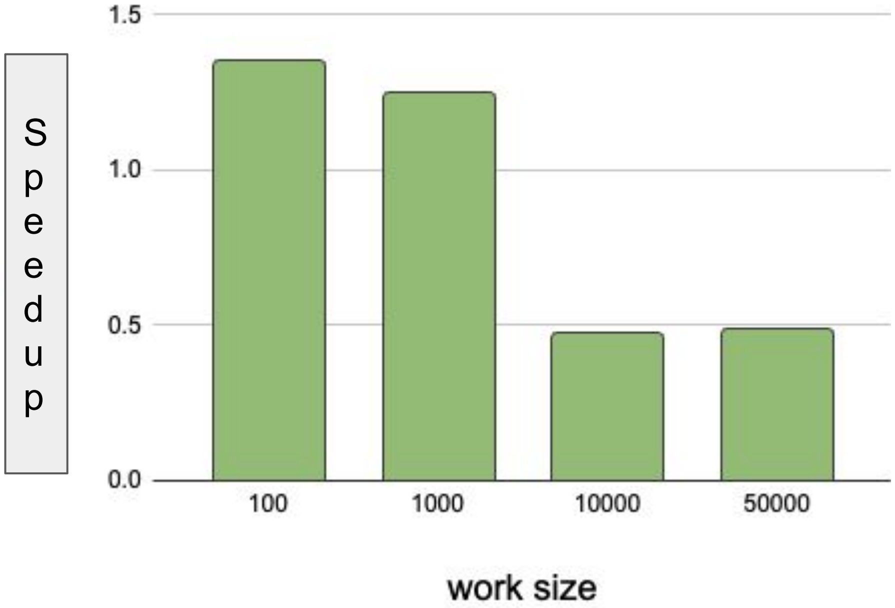
Is the IC serverless?

Serverless	**Internet Computer**
Devs do not admin machines/nodes	✓ (node providers, IC protocol)
Devs break code into small functions	✓ (canisters, update/query calls)
Functions usually short-running	✓ (ideally calls < 1s)
Fine-grained billing	✓ (at the level of instructions)

✗ Single-cloud provider	✓ Decentralized
✗ Need to use external service	✓ Stateful

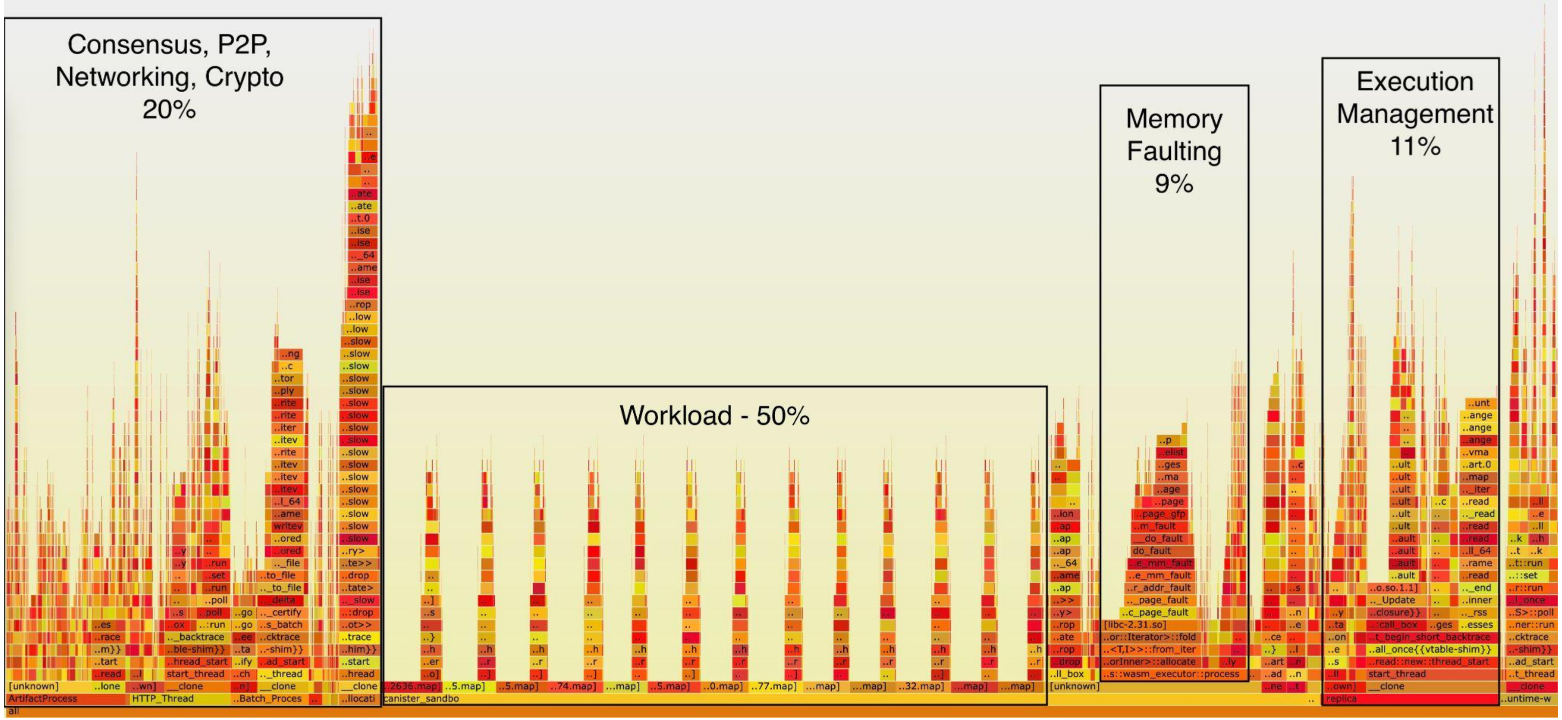
IC vs. Serverless Performance Comparison

- Compute-intensive workload
- Just execution time, no networking or other overheads
- Comparison with one of top-3 serverless platforms



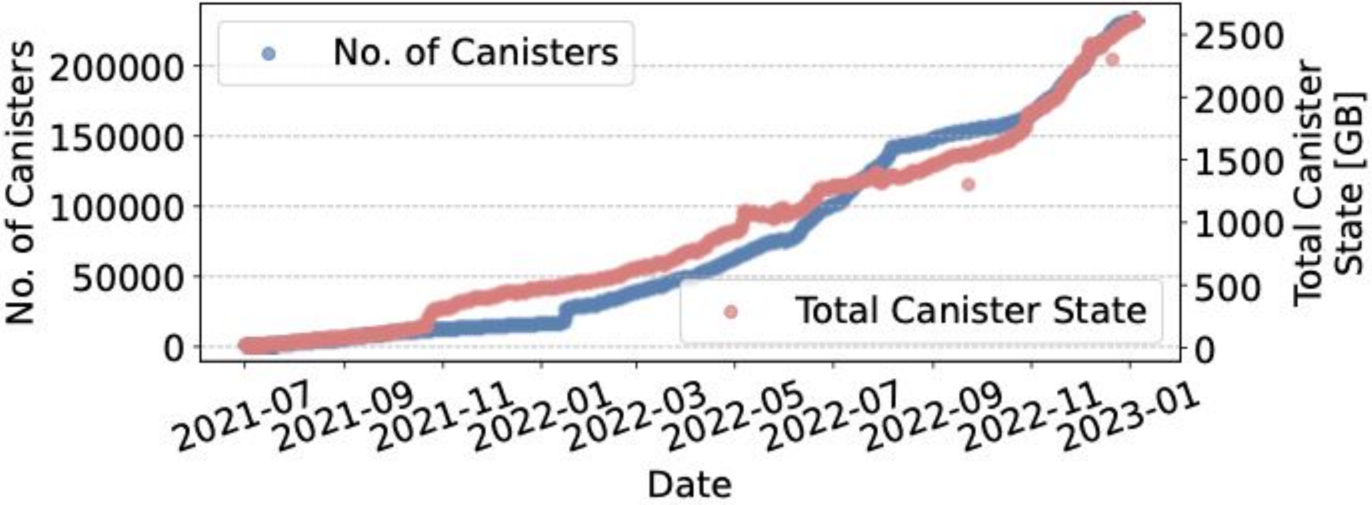
- Promising performance, overheads to improve

Performance Overhead – Memory Intensive Workload



The Internet Computer in Data

- Launch in May 2021
- Data as of Jan 2023 (more growth in the meantime)



Conclusion

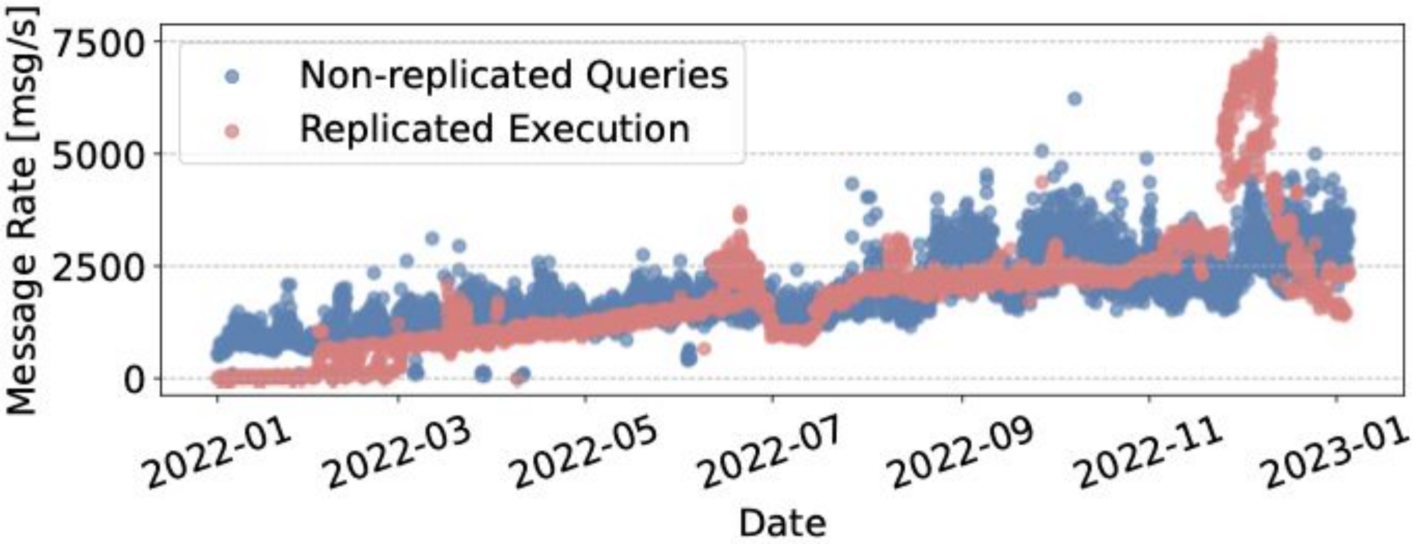
- The IC has been running since May 2021
- Steadily growing in terms of users and workload
- Performance is good, but still room for improvement
- Large team effort, many thanks to all collaborators!

- Lots of systems challenges
- Join us in solving them!
- Join the IC in building new (d)apps!

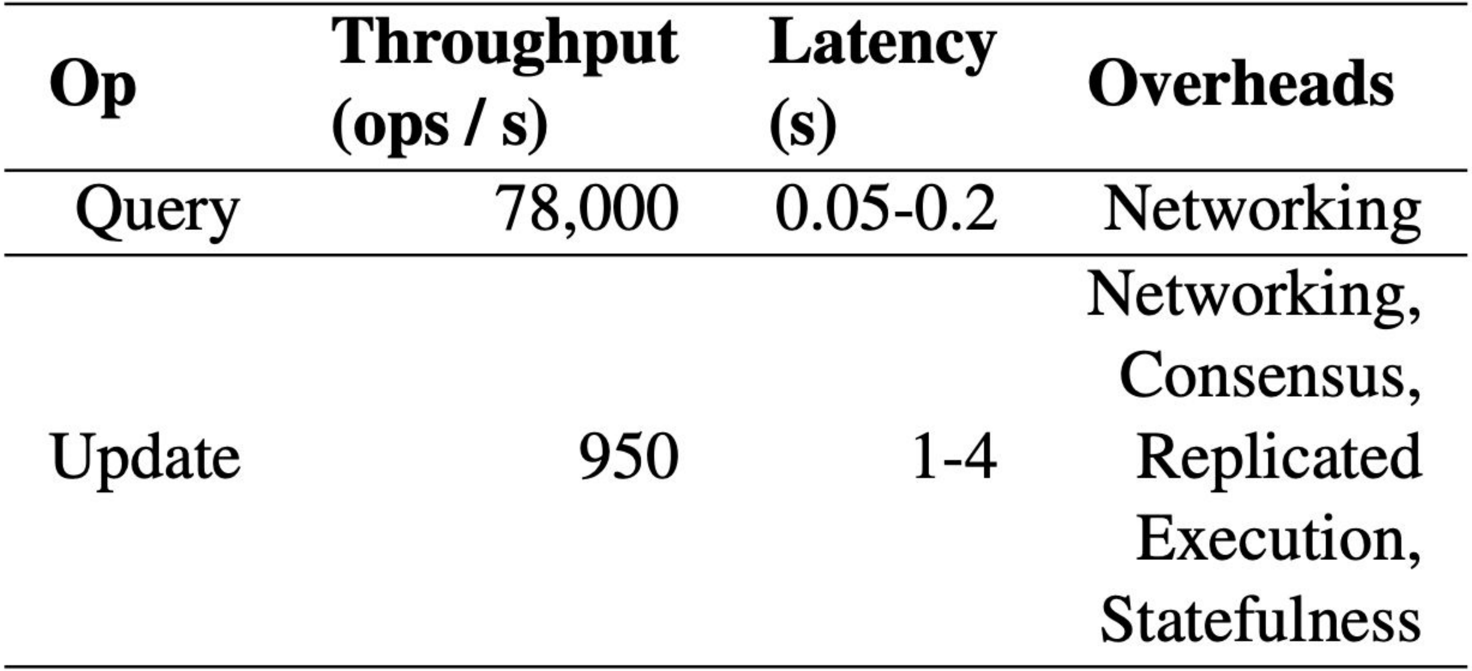
IC code: https://github.com/dfinity/ic
Dashboard: https://dashboard.internetcomputer.org/
Dataset API: https://ic-api.internetcomputer.org/api

Backup Slides

Message Rate

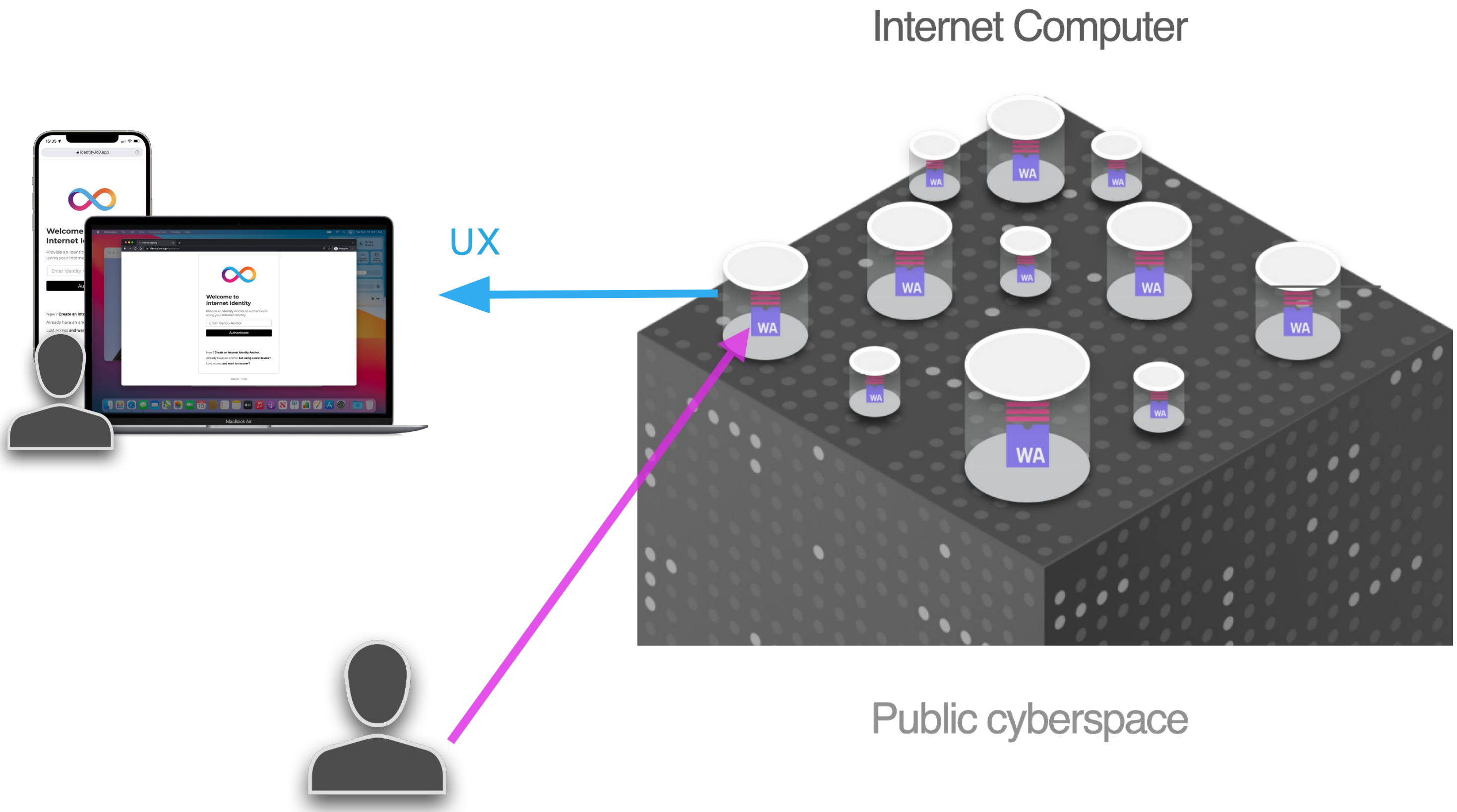


End-to-End Performance per Subnet



Op	Throughput (ops / s)	Latency (s)	Overheads
Query	78,000	0.05-0.2	Networking
Update	950	1-4	Networking, Consensus, Replicated Execution, Statefulness

Developers and users interact directly with Canisters

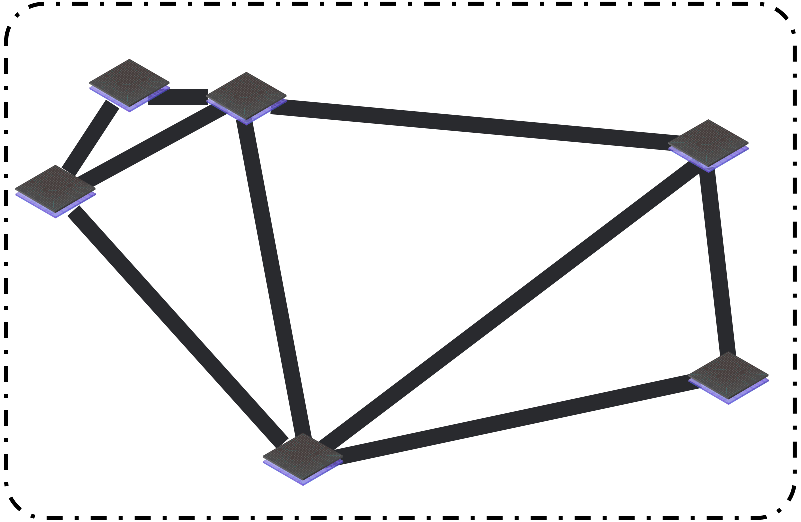


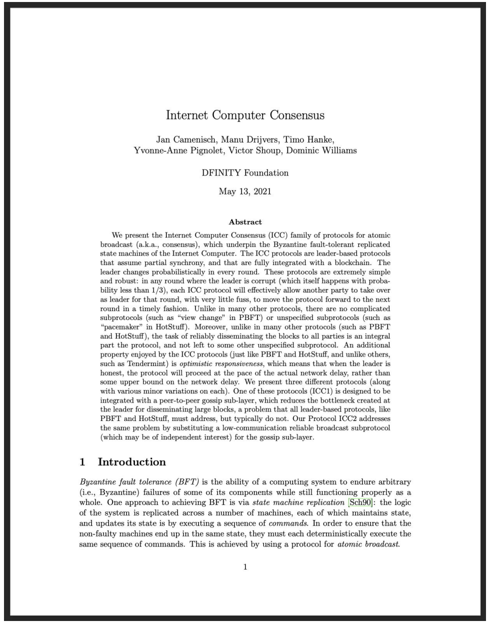
Internet Computer Consensus

Assumption: **n > 3f**

Guarantees **agreement** even under asynchrony

Guarantees **termination** under partial synchrony





Internet Computer Consensus

Jan Camenisch, Manu Drijvers, Timo Hanke,
Yvonne-Anne Pignolet, Victor Shoup, Dominic Williams

DFINITY Foundation

May 13, 2021

Abstract

We present the Internet Computer Consensus (ICC) family of protocols for atomic broadcast (a.k.a., consensus), which underpin the Byzantine fault-tolerant replicated state machines of the Internet Computer. The ICC protocols are leader-based protocols that assume partial synchrony, and that are fully integrated with a blockchain. The leader changes probabilistically in every round. These protocols are extremely simple and robust: in any round where the leader is corrupt (which itself happens with probability less than 1/3), each ICC protocol will effectively allow another party to take over as leader for that round, with very little fuss, to move the protocol forward to the next round in a timely fashion. Unlike in many other protocols, there are no complicated subprotocols (such as "view change" in PBFT) or unspecified subprotocols (such as "pacemaker" in HotStuff). Moreover, unlike in many other protocols (such as PBFT and HotStuff), the task of reliably disseminating the blocks to all parties is an integral part the protocol, and not left to some other unspecified subprotocol. An additional property enjoyed by the ICC protocols (just like PBFT and HotStuff, and unlike others, such as Tendermint) is *optimistic responsiveness*, which means that when the leader is honest, the protocol will proceed at the pace of the actual network delay, rather than some upper bound on the network delay. We present three different protocols (along with various minor variations on each). One of these protocols (ICC1) is designed to be integrated with a peer-to-peer gossip sub-layer, which reduces the bottleneck created at the leader for disseminating large blocks, a problem that all leader-based protocols, like PBFT and HotStuff, must address, but typically do not. Our Protocol ICC2 addresses the same problem by substituting a low-communication reliable broadcast subprotocol (which may be of independent interest) for the gossip sub-layer.

1 Introduction

Byzantine fault tolerance (BFT) is the ability of a computing system to endure arbitrary (i.e., Byzantine) failures of some of its components while still functioning properly as a whole. One approach to achieving BFT is via *state machine replication* [Sch90]: the logic of the system is replicated across a number of machines, each of which maintains state, and updates its state is by executing a sequence of *commands*. In order to ensure that the non-faulty machines end up in the same state, they must each deterministically execute the same sequence of commands. This is achieved by using a protocol for *atomic broadcast*.

1

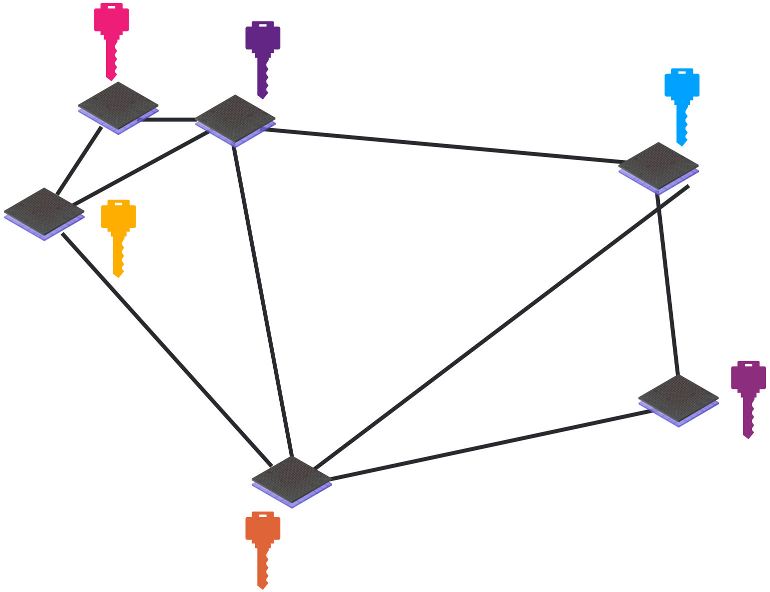
PODC'22

https://internetcomputer.org/how-it-works/consensus

Chain Key Cryptography

Single 48-byte public key

for a secret-shared private key



Non-interactive distributed key generation and key resharing

Jens Groth[1]

`jens@dfinity.org`
DFINITY Foundation

Draft
March 16, 2021

Abstract. We present a non-interactive publicly verifiable secret sharing scheme where a dealer can construct a Shamir secret sharing of a field element and confidentially yet verifiably distribute shares to multiple receivers. We also develop a non-interactive publicly verifiable resharing scheme where existing share holders of a Shamir secret sharing can create a new Shamir secret sharing of the same secret and distribute it to a set of receivers in a confidential, yet verifiable manner.

A public key may be associated with the secret being shared in the form of a group element raised to the secret field element. We use our verifiable secret sharing scheme to construct a non-interactive distributed key generation protocol that creates such a public key together with a secret sharing of the discrete logarithm. We also construct a non-interactive distributed resharing protocol that preserves the public key but creates a fresh secret sharing of the secret key and hands it to a set of receivers, which may or may not overlap with the original set of share holders.

Our protocols build on a new pairing-based CCA-secure public-key encryption scheme with forward secrecy. As a consequence our protocols can use static public keys for participants but still provide compromise protection. The scheme uses chunked encryption, which comes at a cost, but the cost is offset by a saving gained by our ciphertexts being comprised only of source group elements and no target group elements. A further efficiency saving is obtained in our protocols by extending our single-receiver encryption scheme to a multi-receiver encryption scheme, where the ciphertext is up to a factor 5 smaller than just having single-receiver ciphertexts.

The non-interactive key management protocols are deployed on the Internet Computer to facilitate the use of threshold BLS signatures. The protocols provide a simple interface to remotely create secret-shared keys to a set of receivers, to refresh the secret sharing whenever there is a change of key holders, and provide proactive security against mobile adversaries.

1 Introduction

The Internet Computer hosts clusters of nodes running subnets (shards) that host finite state machines known as canisters (advanced smart contracts). The

https://internetcomputer.org/how-it-works/chain-key-technology