The Internet Computer for Systems Researchers

Stefan Kaestle
Ulan Degenbaev, Adam Bratschi-Kaye, Andriy Berestovskyy

June 2022

We are hiring: dfinity.org/careers

Agenda

1) What is the IC?
2) Interesting Systems problems
3) Numbers
4) Q&A

DFINITY

What is the Internet Computer?

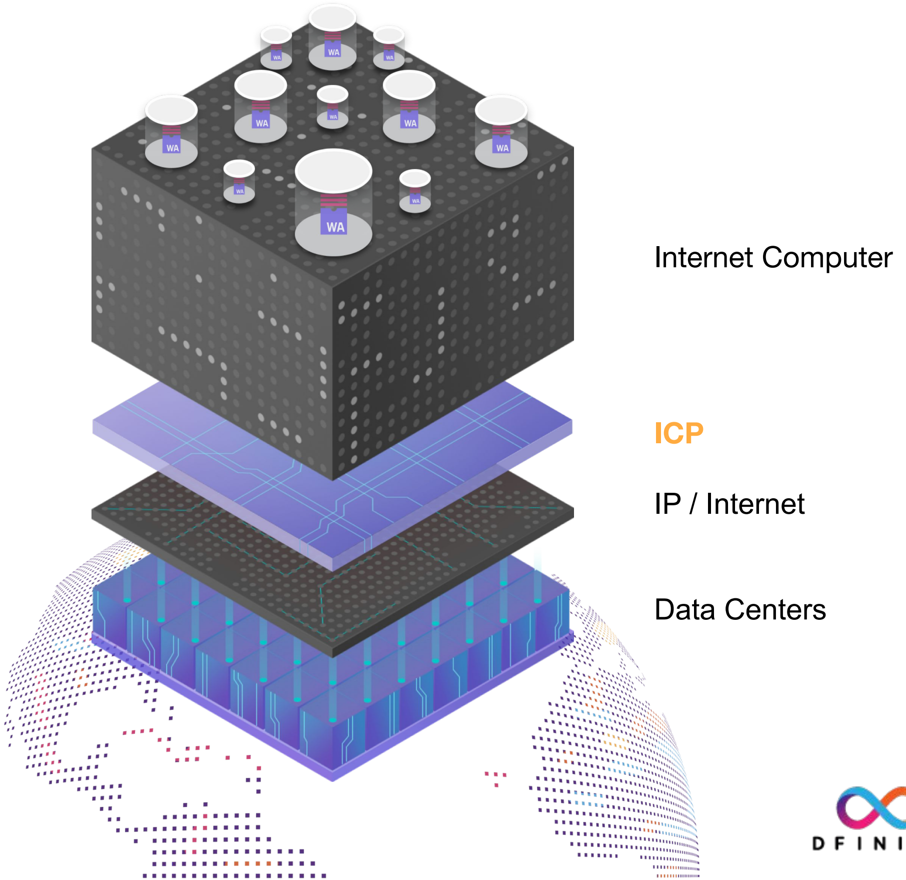
What is the Internet Computer?

Vision:

Platform to run any computation
in a decentralized and secure manner

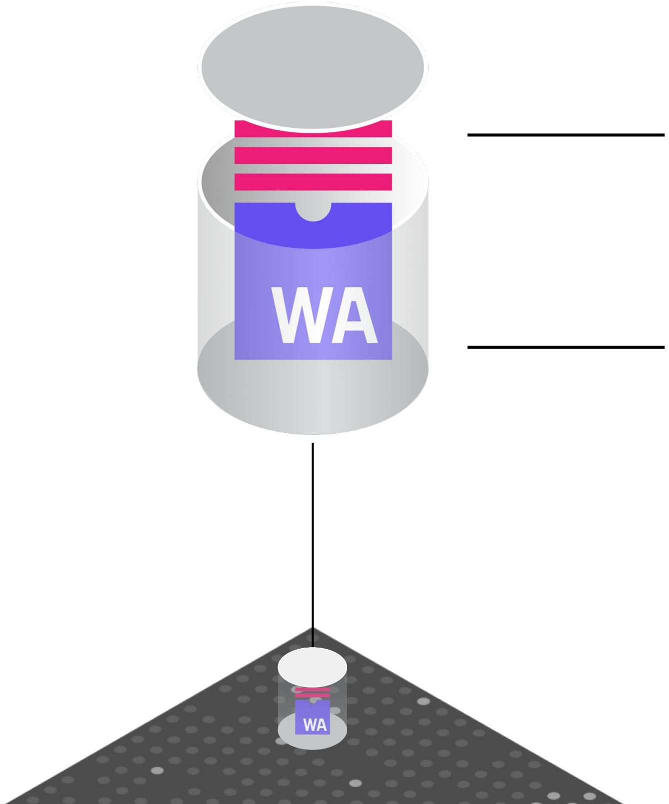
What’s different about the Internet Computer

- **Byzantine** fault tolerance
  - Up to f out of 3f + 1 malicious nodes
  - Individual **nodes cannot be trusted**
- Geo replicated
- Decentralized
  - DFINITY cannot access most nodes
- Self governing
  - No single person in control of the IC
  - Votes to apply changes



DFINITY

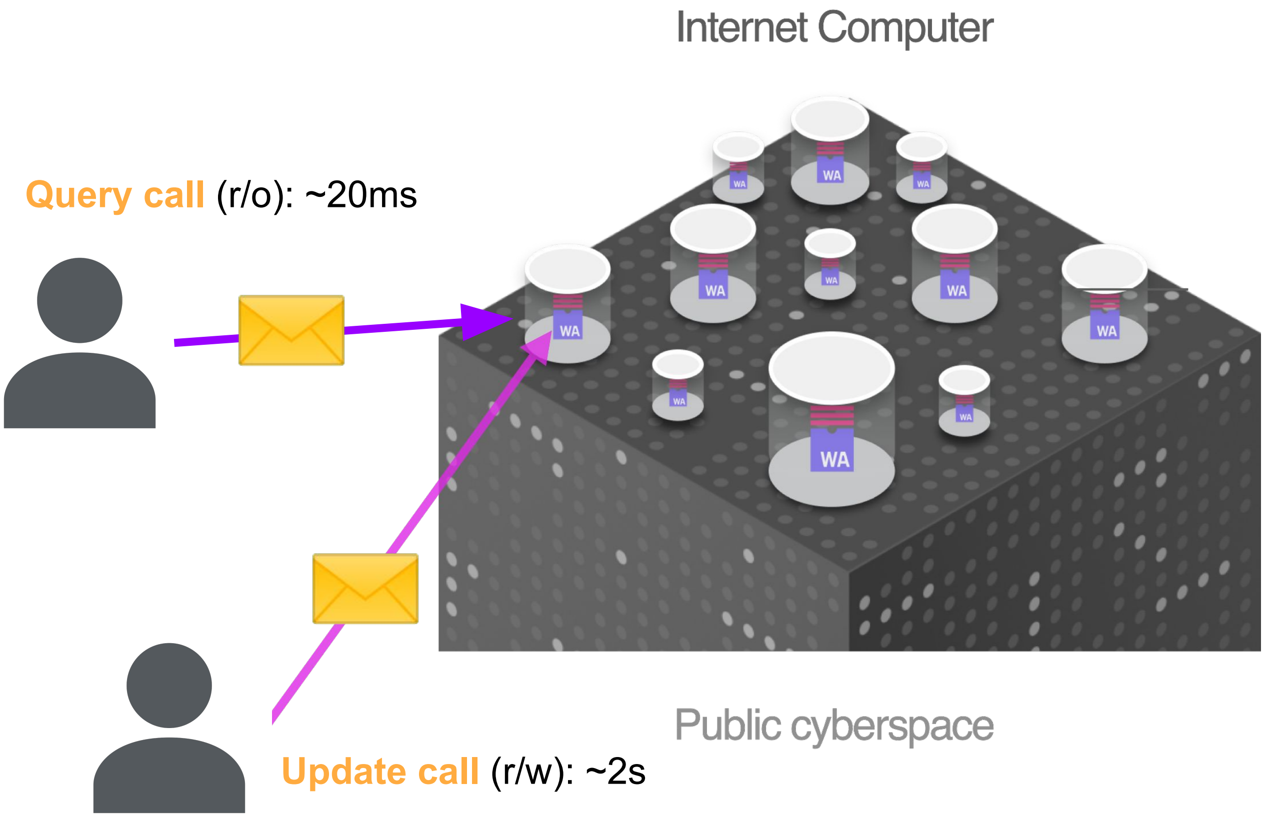
Canister Smart Contracts



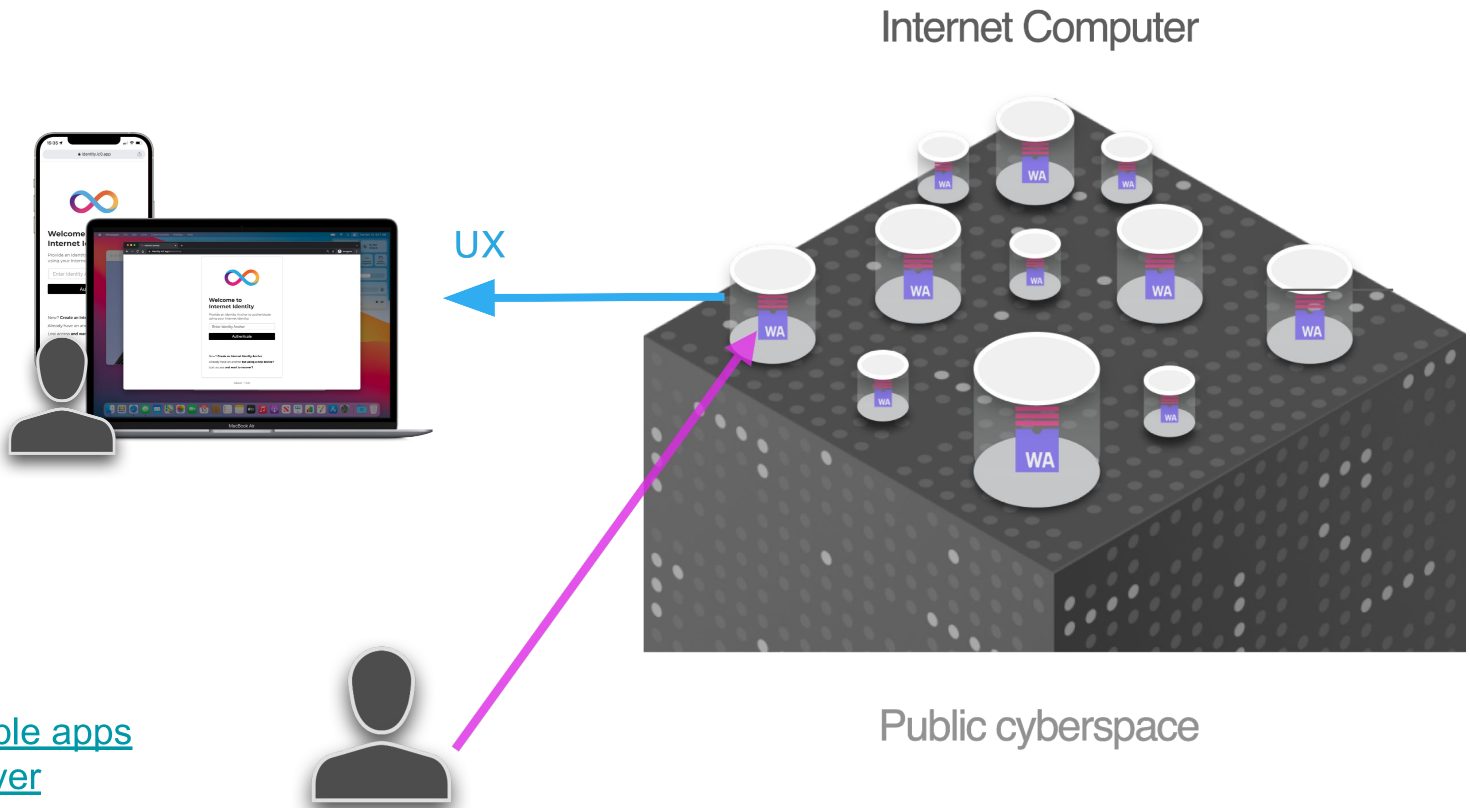
Data: Memory pages

Code: WebAssembly bytecode

Users interact directly with Canisters: raw calls



Developers and users interact directly with Canisters



[Example apps](#)
[Discover](#)

State Machine Replication (SMR)

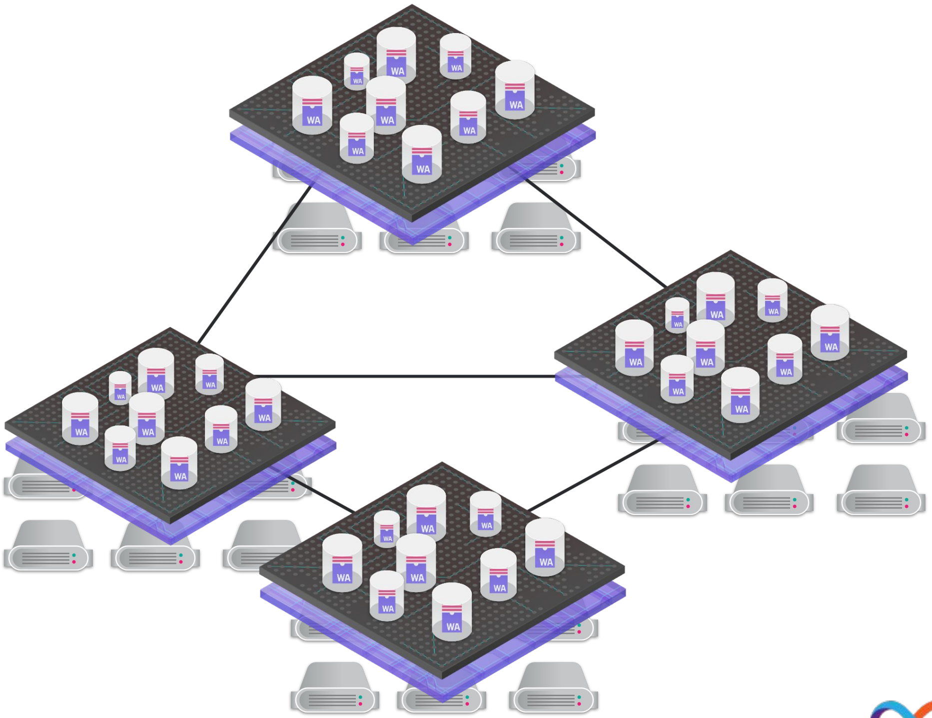
Nodes must have same state

1. State on all nodes is identical
2. Deterministic state transitions
3. Ordered input

→ State still the same after executing inputs

IC state:

- Canister code, data and queues
- System state



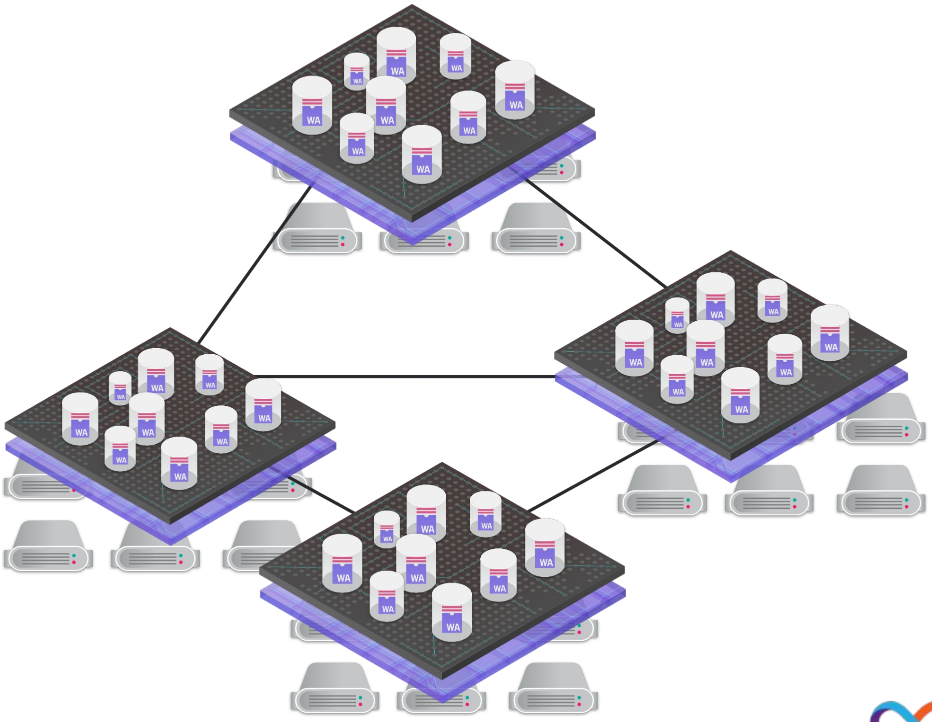
Scalability: Nodes and Subnets

Nodes are partitioned into subnets

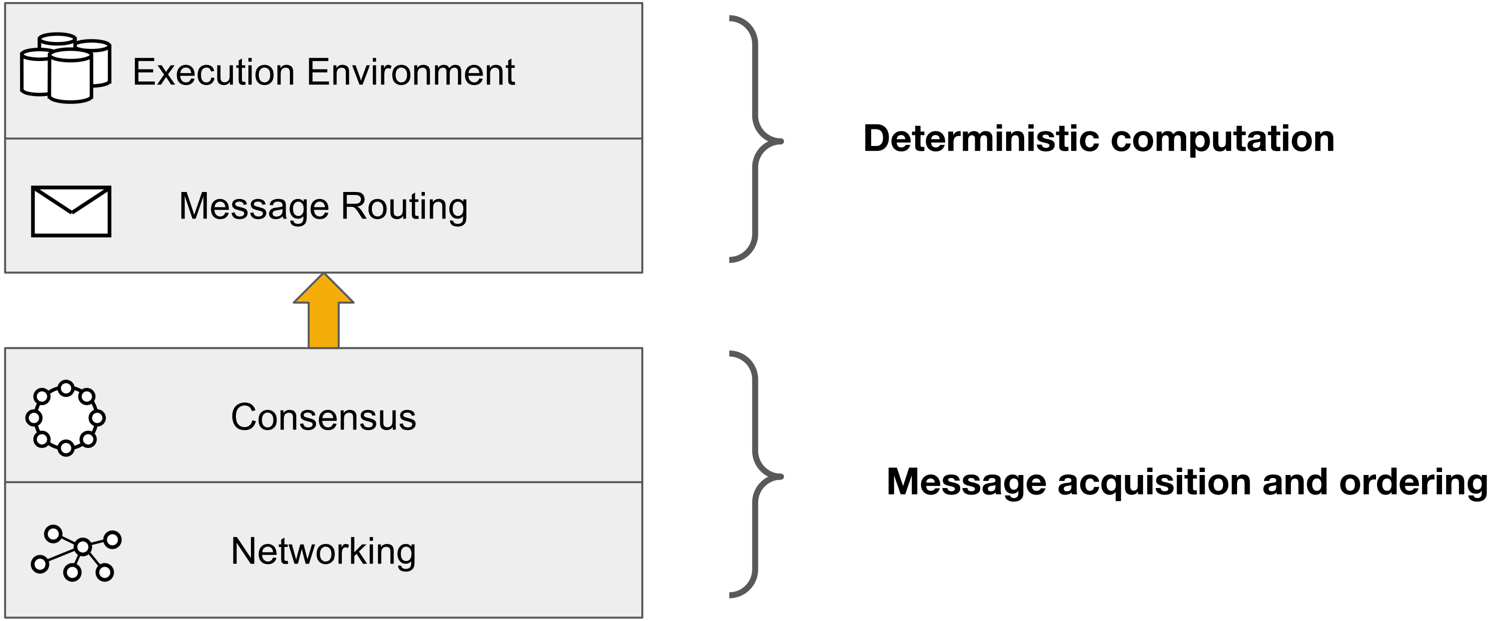
Each subnet runs instance of SMR

Each subnet hosts a subnet of canisters

Communication across subnets possible



ICP Layers


Execution Environment
Message Routing
Consensus
Networking
Deterministic computation
Message acquisition and ordering

Execution Environment

Hello World example app

```
#[query]
fn greet(name: String) -> String {
    format!("Hello {}", name)
}
```

- Canister code: wasm
  - Official support: Rust and Motoko
- Install: get canister ID
- Call via canister ID
  - Raw calls
  - HTTP calls

App with state: Orthogonal persistence

- Illusion: programs run forever
- Program state (incl. heap) is persisted/restored automatically

App with state: Orthogonal persistence

```
thread_local! {
    static STATE: RefCell<State> = RefCell::new(State::default());
}

#[derive(Default)]
pub struct State {
    owner: Option<Principal>,
    ledger: Option<Principal>,
    exchange: Exchange,
}
```

Sampel Defi

Note:
Programming is significantly simpler in Motoko

App with state: Orthogonal persistence

```
#[query(name = "getBalance")]
#[candid_method(query, rename = "getBalance")]
pub fn get_balance(token_canister_id: Principal) -> Nat {
    STATE.with(|s| s.borrow().exchange.get_balance(token_canister_id))
}
```

[Sampel Defi]()

App with state: Orthogonal persistence

```
#[update]
#[candid_method(update)]
pub async fn deposit(token_canister_id: Principal) -> DepositReceipt {
    let caller = caller();
    // Stuff happens here ..
    STATE.with(|s| {
        s.borrow_mut()
            .exchange
            .balances
            .add_balance(&caller, &token_canister_id, amount.to_owned())
    });
    DepositReceipt::Ok(amount)
}
```

Sampel Defi

Orthogonal persistence: Track changes + accounting

Challenge: Need to track changes to memory

Current solution (simplified): Map memory pages on demand

Example: Canister call

1. Initially: no page is mapped
2. Read access: page fault → map r/o, *increase read counter*
3. Write access: page fault → (re-)map r/w, *increase write counter + remember page*
    a. Query call: throw away dirty pages
    b. Update call: store changes in heap delta

Note: 95% of message executions change at most seven memory pages.

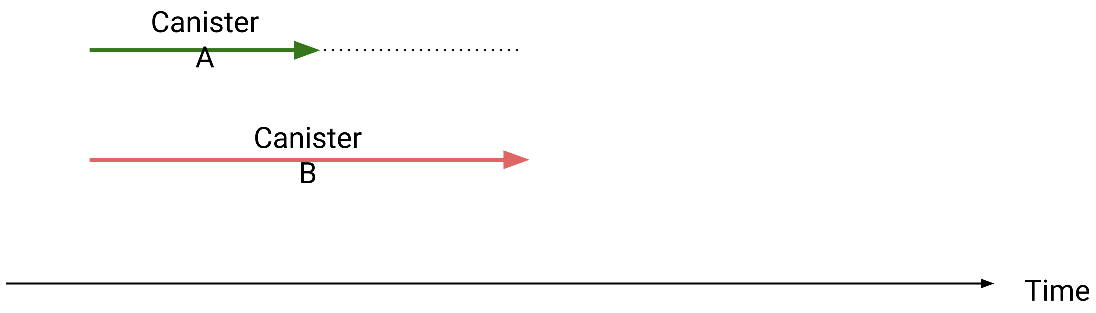
Orthogonal persistence: Performance

Naive solution quite slow.

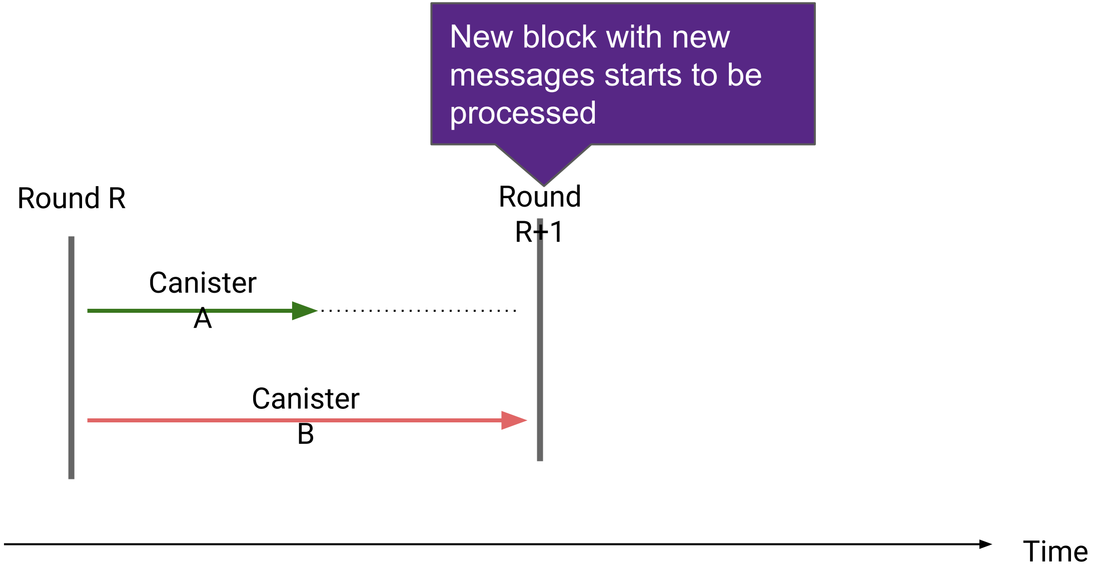
- Can **speculatively map** multiple consecutive pages: → trade accuracy for speed
  - diff on speculatively mapped r/w pages
- Map **r/w for query calls** (we throw changes away anyway)

Future: might explore modifying the wasm runtime to compile in profiling instructions

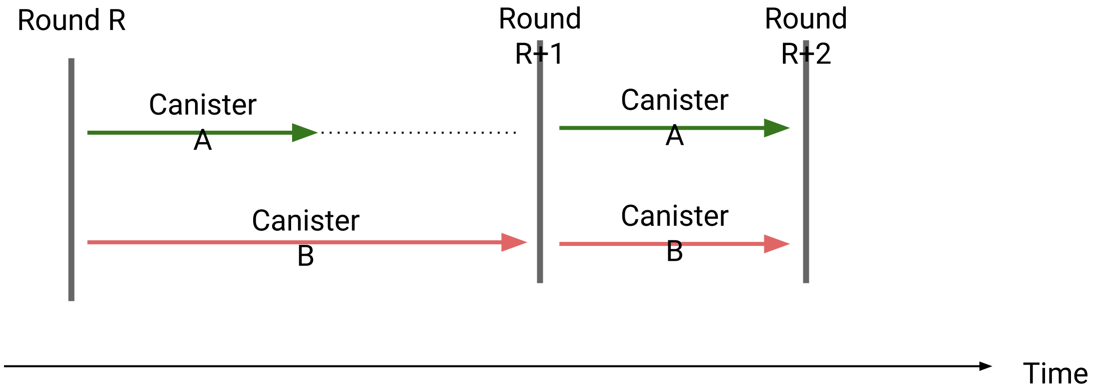
Multiple concurrent canister executions



Multiple concurrent canister executions



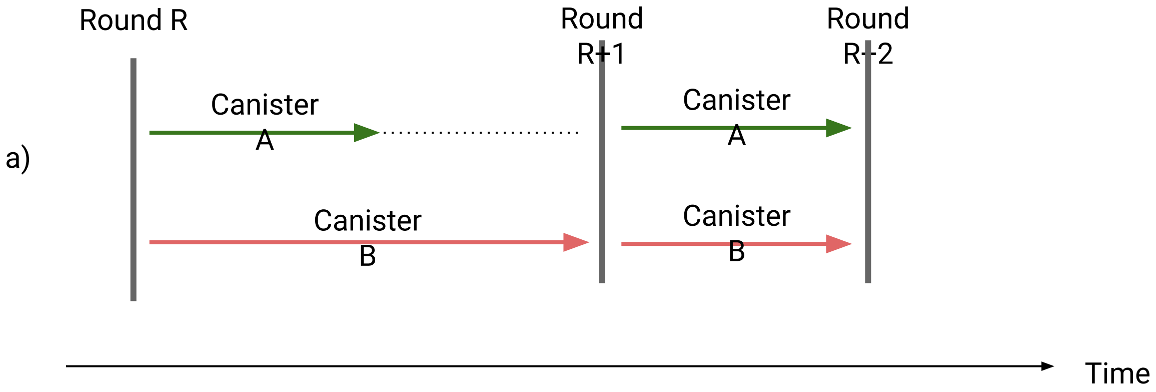
Multiple concurrent canister executions



Canister A might suffer from Canister B

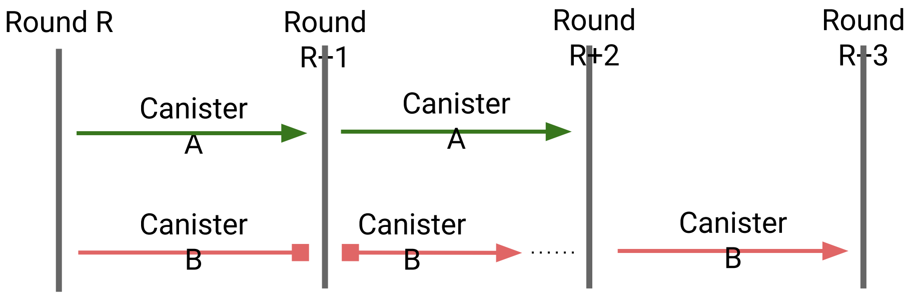
Multiple concurrent canister executions

- Want block ~1s → Execution has to process messages in ~1s



- Limit number of instructions per message
  a. But: Some messages take longer
     - E.g. canister upgrade, with expensive pre- and post-hooks
     - Garbage collection

Scheduling: time slicing



- Has to be **deterministic**
    a. Load balancing etc. gets harder

- Reservations (compute allocations)
- Good resource usage
    a. Fill with best-effort, fairness
- Intermediate state must not be observable
    a. Atomicity: Roll-back on error + Isolation

Time slicing and checkpointing

- Checkpoint to disk every 500 rounds (~500s, ~8min)
- Contains all state required to resume computation

- Partially executed messages at checkpoint?
    a. Nodes have be able to resume from checkpoints
    b. What to do with incomplete message executions?

Scheduling & time slicing

- Quite challenging
- Still ongoing discussion

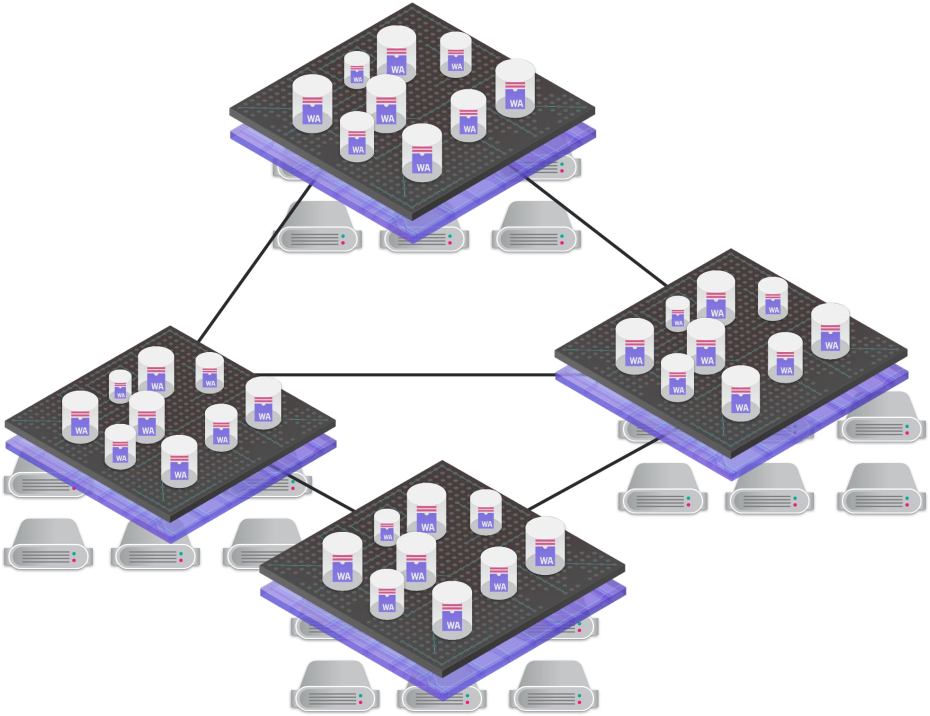
- Come talk to us if you are interested in working on things like this

Some numbers

The IC in Current Numbers

Network Layer:

- 477 nodes
  - From 54 node providers
- 33 subnets



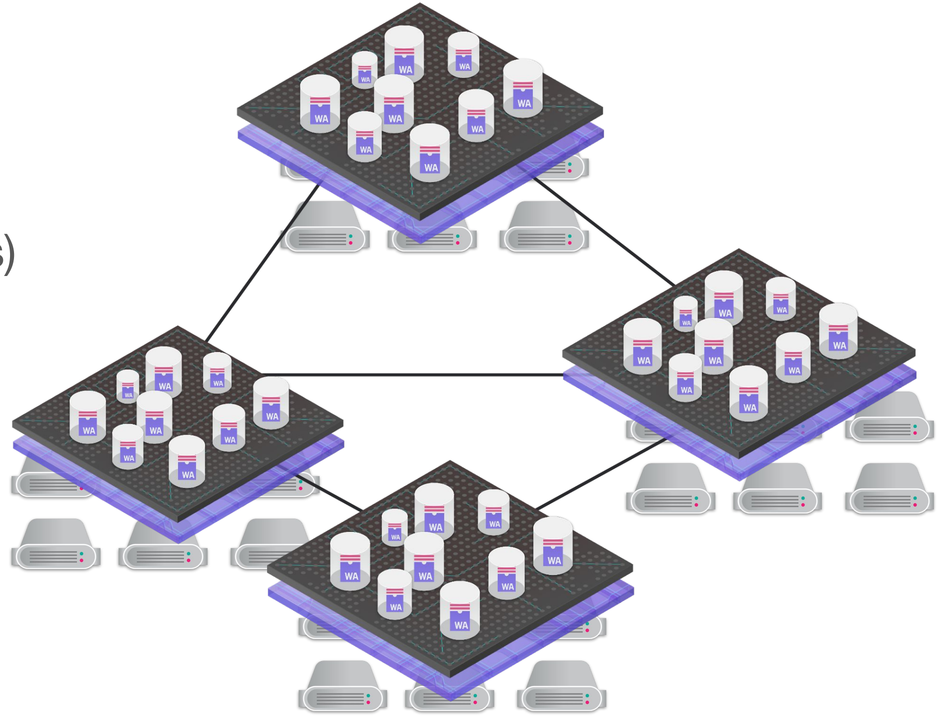
https://dashboard.internetcomputer.org/

DFINITY

The IC in Current Numbers

Application Layer:

- 75K+ canisters
- > 2 Mio registered identities (~users)
- ~1.1TB total state (and counting…)



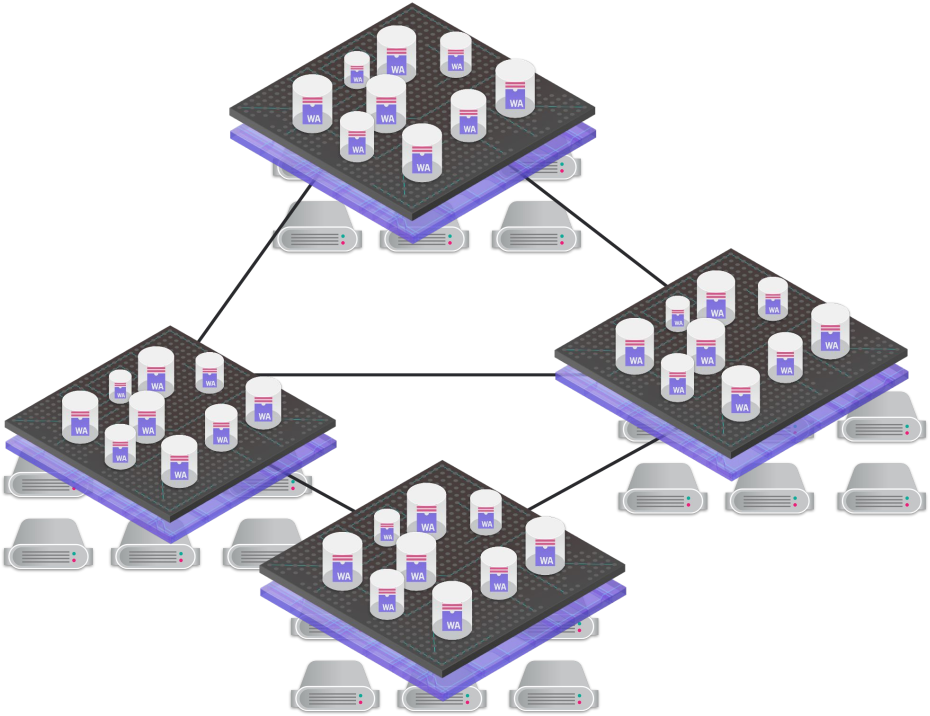
https://dashboard.internetcomputer.org/

DFINITY

The IC in Current Numbers

Consensus

- 850M+ blocks created
- ~34 blocks per second
- ~3500 messages per second



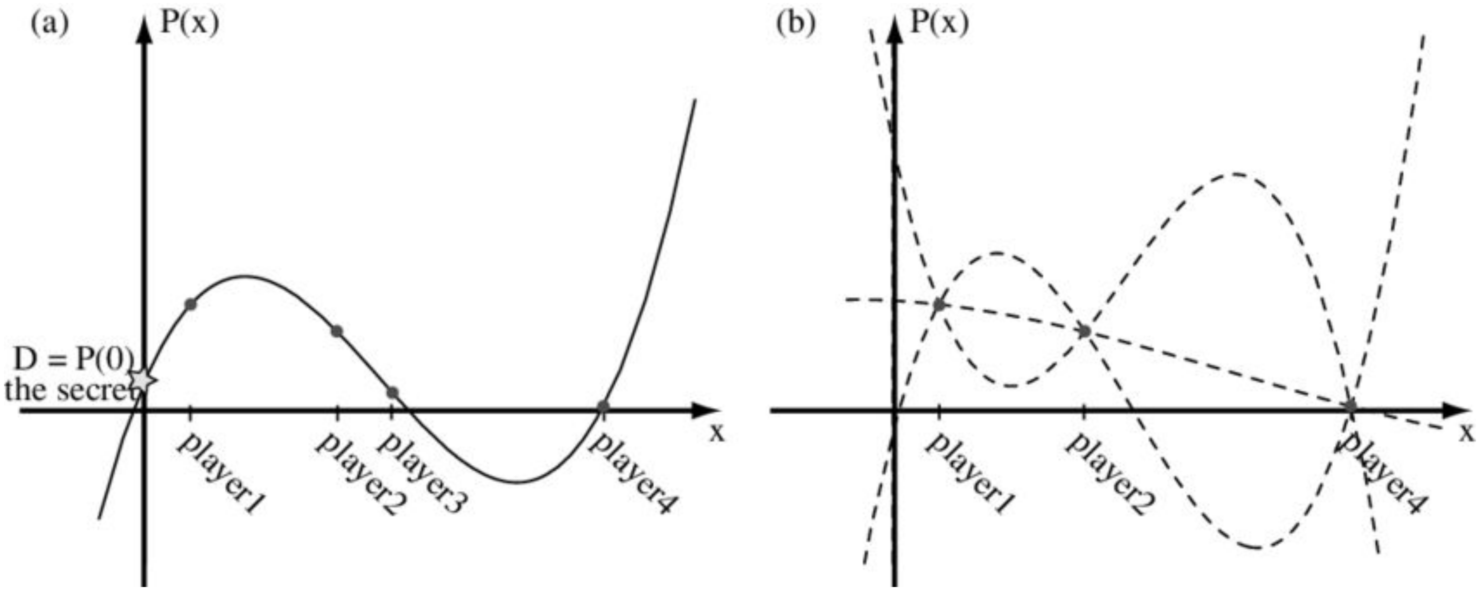
https://dashboard.internetcomputer.org/

DFINITY

Energy use of the IC

- Blockchains have a bad reputation
  - Mostly due to proof of work
- We don’t do that
- We have random beacon and threshold cryptography
  - Single public key that can be used to verify responses from IC
  - Can throw away old state (don’t need to maintain forever)

Threshold Cryptography in a nutshell



Shamir's polynomial of degree 4

Energy use of the IC

- Peak power consumption of node machines: 700W
- Power usage effectiveness (PUE): 2.33 (extremely conservative)
  - A PUE of 1: all power is spent on compute
  - A PUE of 2: as much power for cooling etc as for compute
  - 2.33 is quite conservative (e.g. Google closer to 1.1)
- With PUE: 1631W per IC node

- Number of machines: 518 + 11 boundary nodes (as of weekend)
- Total max power consumption of all nodes: ~863kW

- ~3300 transactions / s → 261.45 Ws per transaction = 261.45 Joule
- Conservatice: hardware currently is underutilized

DFINITY

Energy use of the IC

- ~3300 transactions / s → 261.45 Ws per transaction = 261.45 Joule
- Conservatice: hardware currently is underutilized

[Solana Energy Usage Report]

Activity	Energy Used, in Joules (J)
A single Google Search [1]	1,080 J
A single Solana transaction	*1,837 J*
Keeping an LED light bulb on for one hour [2]	36,000 J
Using a fully-charged iPhone 13 on battery [3]	44,676 J
Working for an hour with a computer and monitor [4]	46,800 J
One eth2 transaction [5]	126,000 J
Watching an hour of television on a 40 inch+ LCD TV [4]	540,000 J
Playing one hour of a PlayStation 5 game [6]	708,840 J
Running a refrigerator for one hour [4]	810,000 J
One hour of central air conditioning [4]	12,600,000 J
Using one gallon of gasoline [7]	121,320,000 J
One Ethereum transaction [8]	692,820,000 J
One Bitcoin transaction [9]	6,995,592,000 J

DFINITY

Questions? Reach out to:

stefan.kaestle@dfinity.org
ulan.degenbaev@dfinity.org
adam.bratschikaye@dfinity.org

We are hiring: dfinity.org/careers

BACKUP

Scalability: Nodes and Subnets

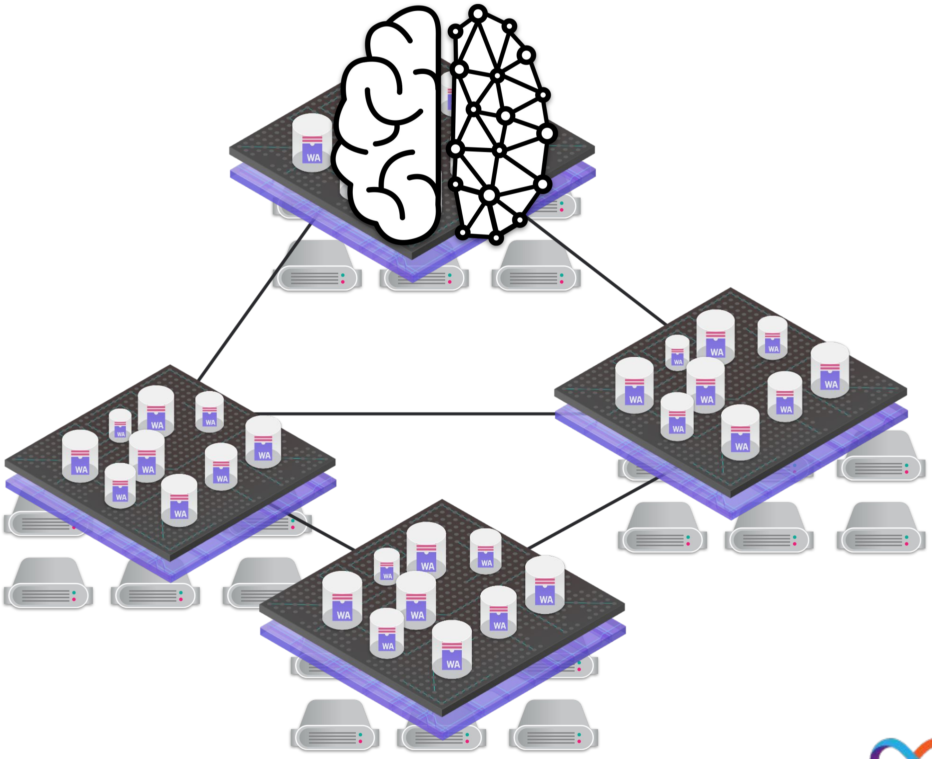
Nodes are partitioned into subnets

Canister smart contracts are assigned to different subnets

One subnet is special: it host the **Network Nervous System (NNS)** canisters which govern the IC

ICP token holders vote on

- Creation of new subnets
- Upgrades to new protocol version
- Replacement of nodes
- …



DFINITY

Research on the IC

Open Research Problems

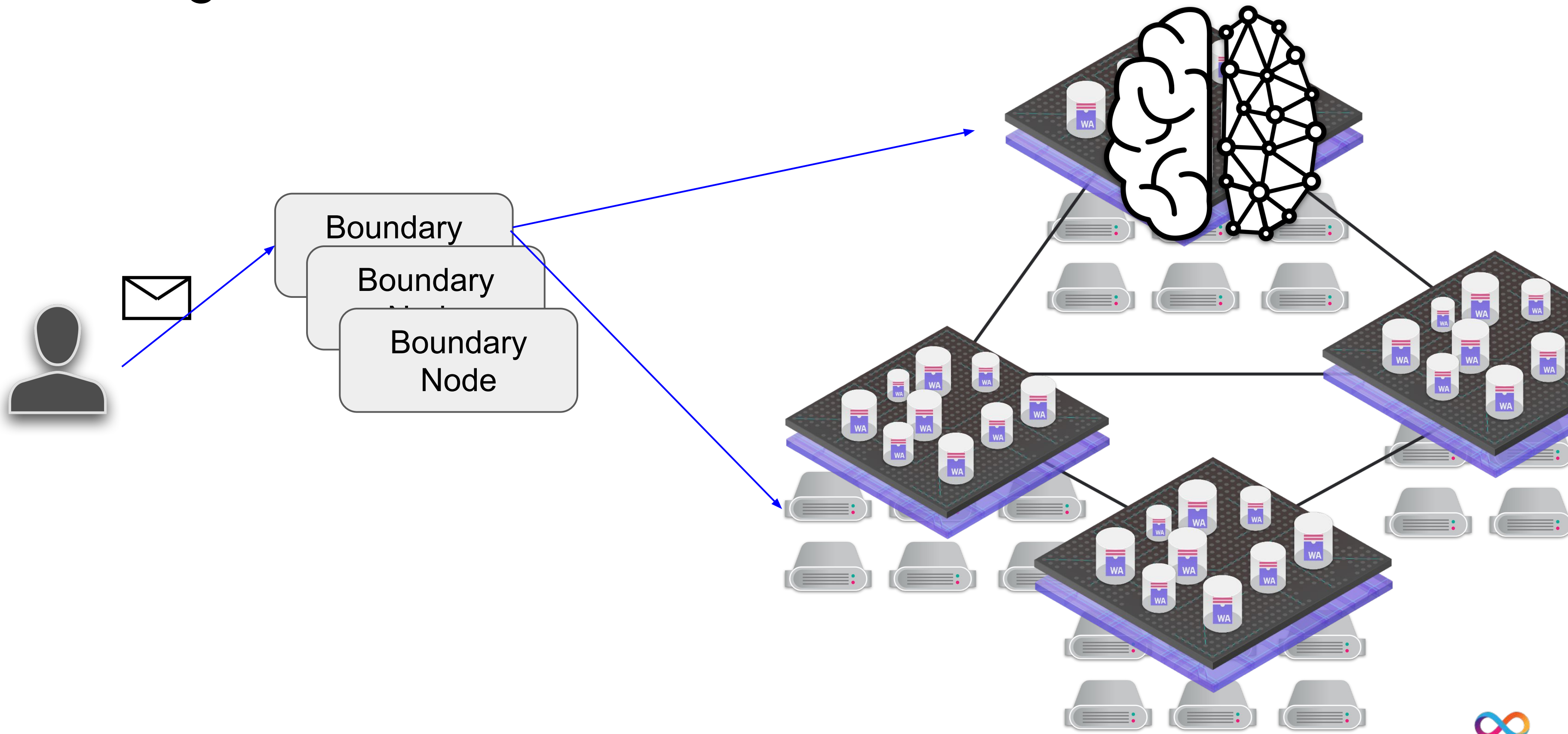
- Intra-subnet communications scalability (growing size of subnets)
- Inter-subnet communications scalability (growing number of subnets)
- Ongoing firewall rule management
- Resilience against malicious activity
- Monitoring of node and network behavior
- Dynamic load balancing
- Caching
- Canister addressing

IC Networking

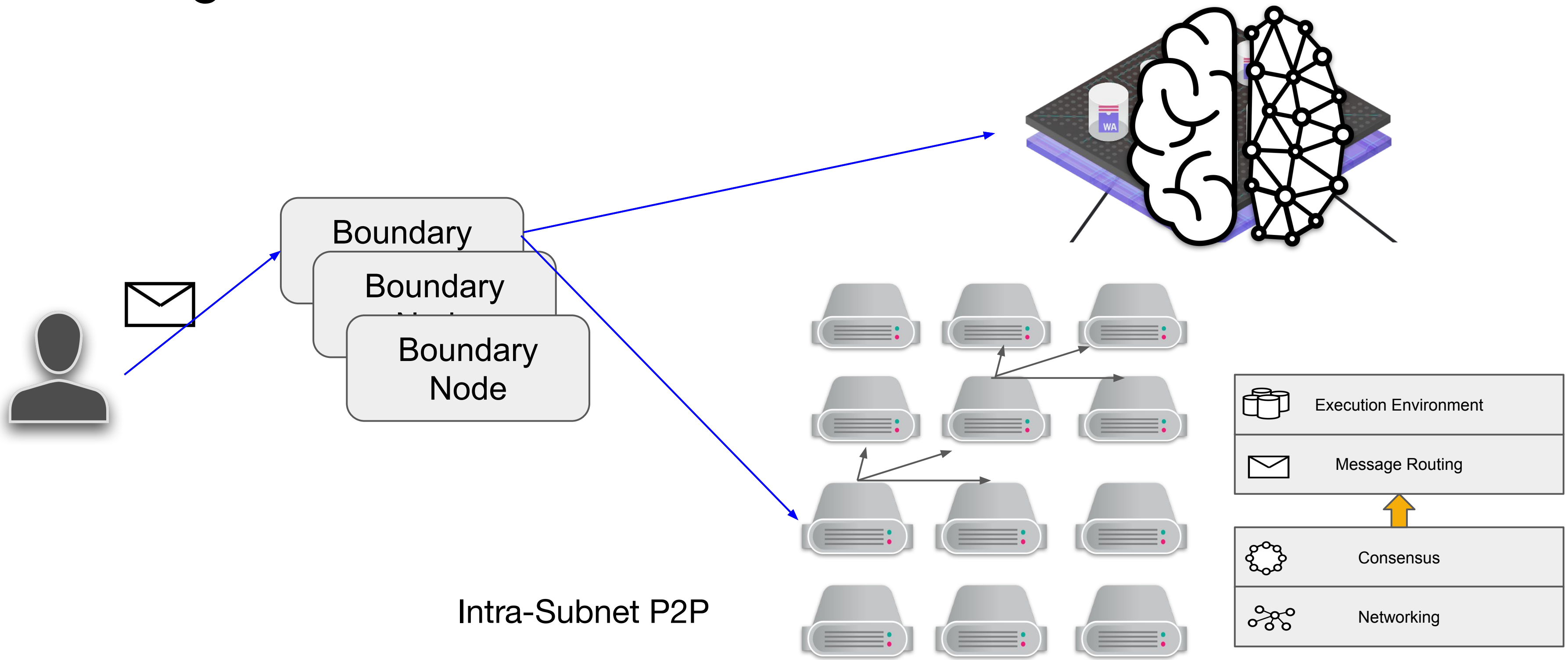
Consensus

- Byzantine fault tolerance
- Random beacon

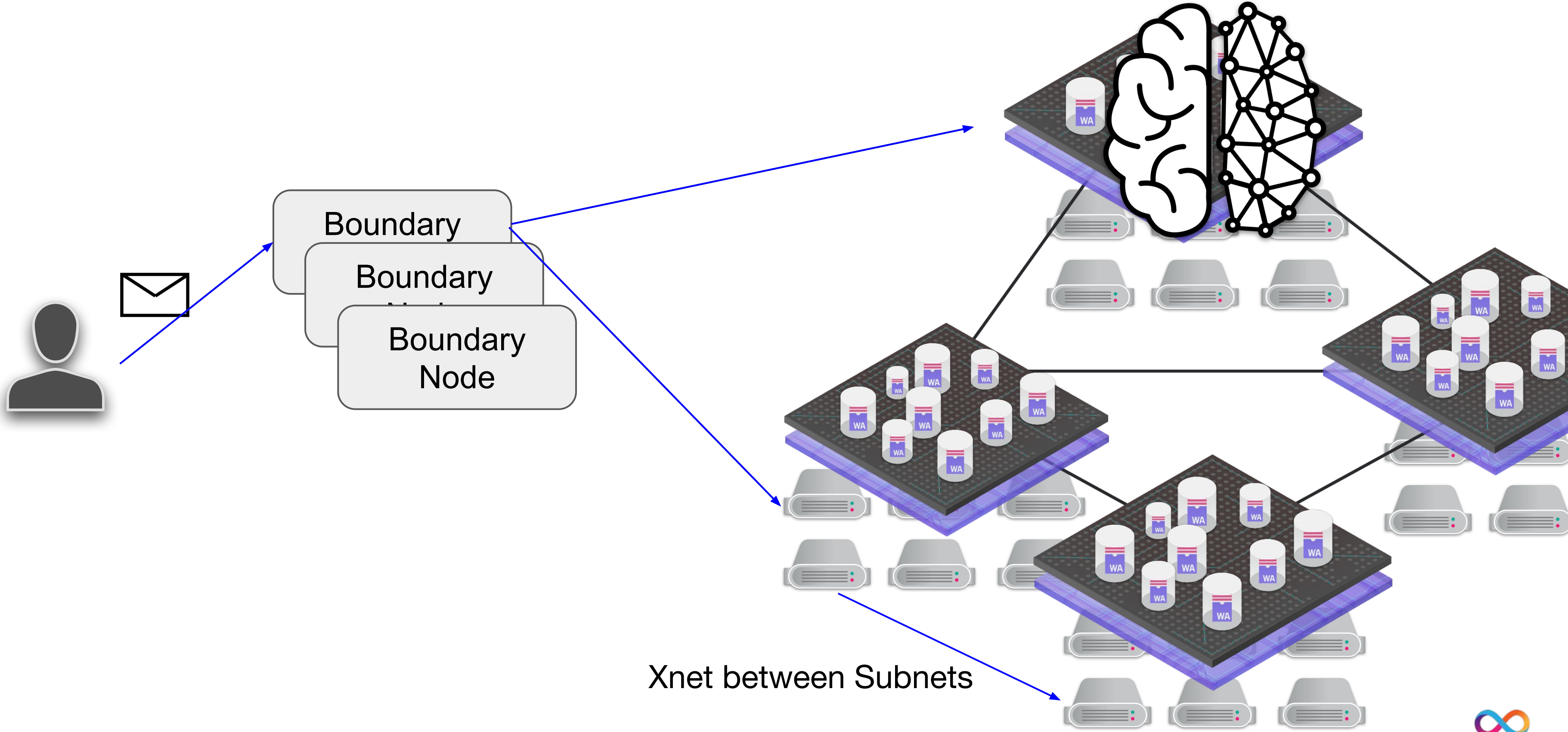
Following a canister call



Following a canister call



Following a canister call



Requirements 1/2

- **Bounded-time/eventual delivery despite Byzantine faults**

Up to a certain maximum volume of valid artifacts that are not dropped by any honest node reaches all honest nodes in bounded time/eventually despite attacks (under certain network assumptions).

- **Reserved resources for different components/peers**

Memory/bandwidth/CPU guarantees for different components and peers

- **Prioritization for different artifacts**

Not all artifacts are equal, different priorities depending on attributes (e.g., type, size, round,...). Priorities change over time.

Requirements 2/2

- **High efficiency**

High throughput is more important than low latency

Avoid duplicates: don't waste bandwidth downloading same artifact "too many times"

- **DOS/SPAM resilience**

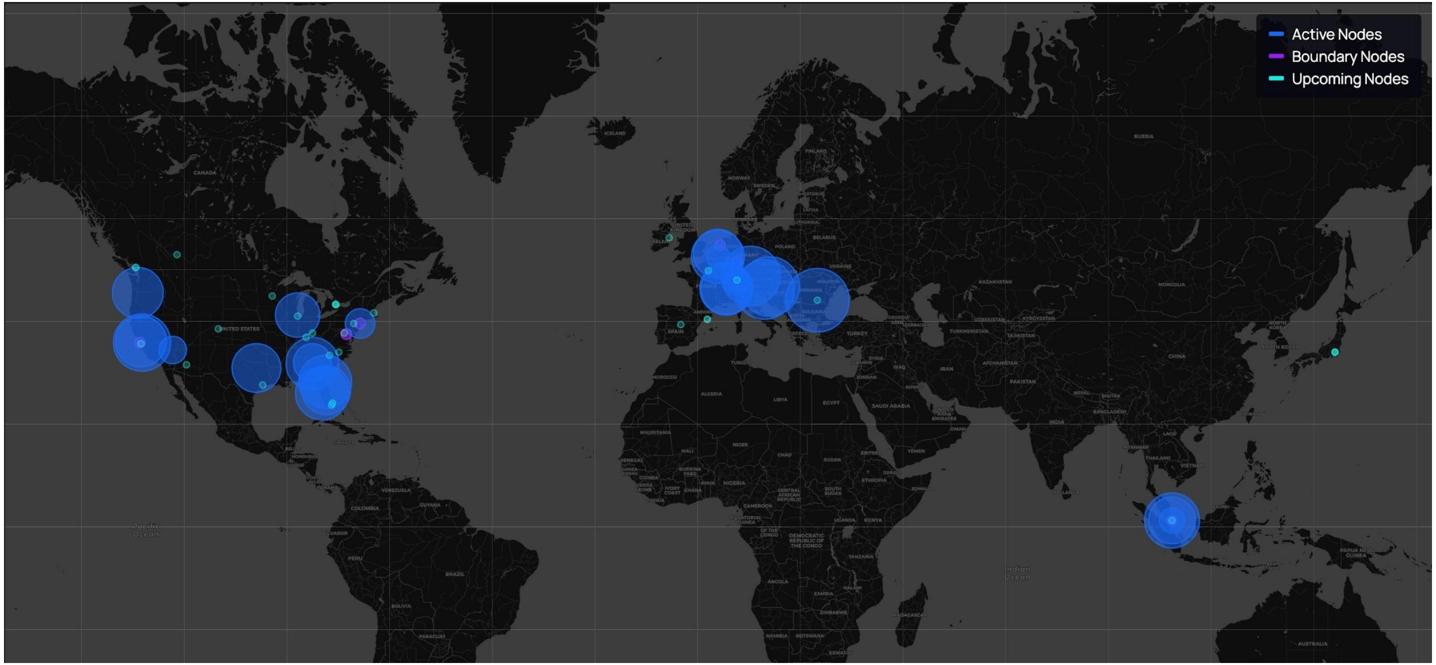
Bad participants cannot prevent progress.

- **Low accessibility requirements for users**

Support browser and IPv4 access

Networking of the IC

- **Geographically distributed**: datacenters all over the world



Networking of the IC

- **Geographically distributed**: datacenters all over the world
- **Decentralized**: a subnet is composed of nodes in different datacenters
  → Some nodes in the same subnet may be very far apart
  → Independent node providers with different skills and DC contracts
  → Communication over public internet
    - High latencies possible
    - Many transient network failures
- **Secure**: a subnet should make progress even if up to ⅓ of the nodes are malicious / faulty
  → We can't trust specific nodes (e.g., geographically close by)
  → Even nodes in the same subnet should not trust each other

Xnet Inter-Subnet Networking

- Canisters on one subnet can send messages to canisters on other subnets, called "cross-net communication" (or Xnet)
- Currently this is done quite naively, where any node on one subnet can fetch messages from any other node on the other subnet with a HTTPS request
- We can probably improve this on several aspects:
  - Scalability: decide which nodes connect to which
  - Performance: leverage the fact that some nodes in both subnets are close to each other (content is signed by the subnet, so we do not need to trust a specific node up to some extent)

Comparison with other Blockchain Systems

As of June:
> 20,000 TPS
> 2,000,000 QPS

coincodex

Layer-1 Performance Comparison

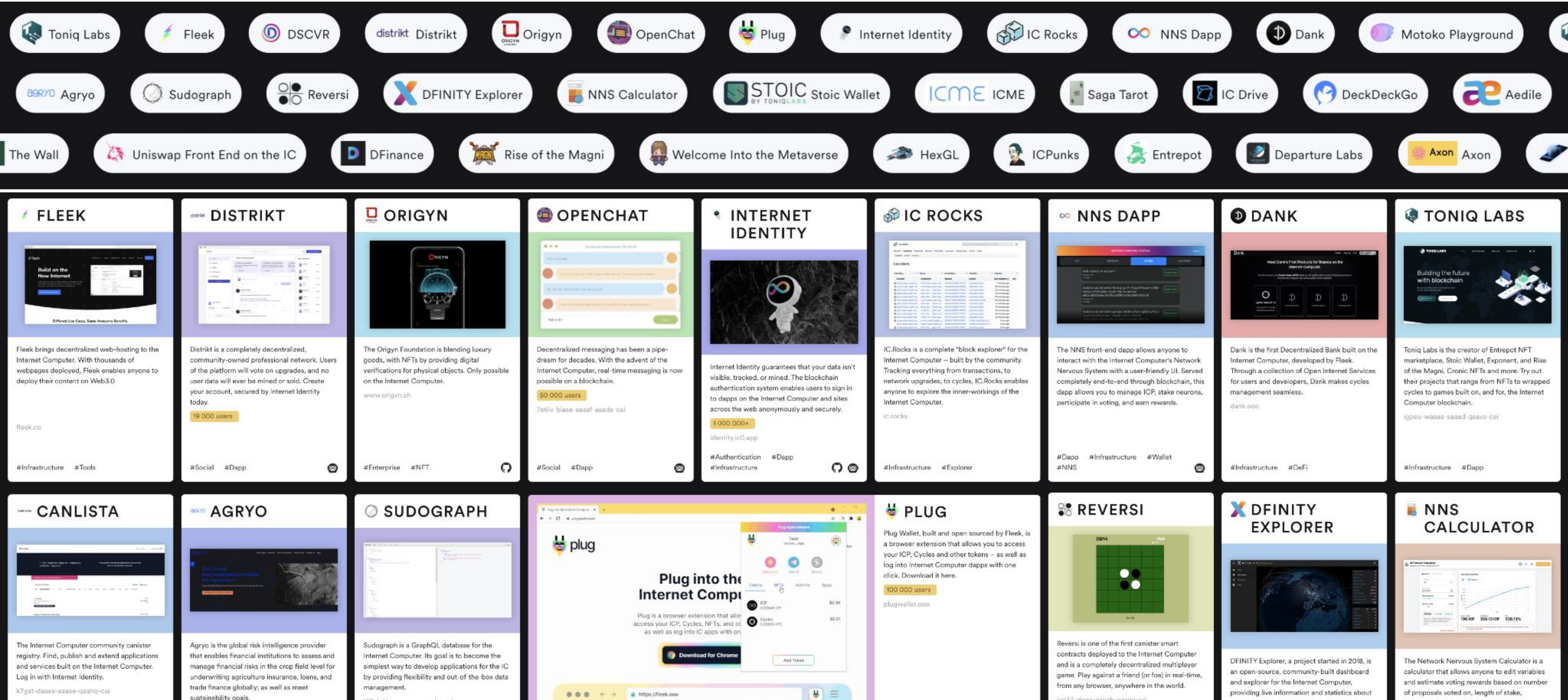
	Ethereum	Cardano	Solana	Avalanche	Algorand	Internet Computer
Transaction Speed	15-20 TPS	2 TPS	2,000-3,000 TPS	4,500 TPS	20 TPS	11,500 TPS 250,000 QPS
Transaction Finality	14 minutes	10-60 minutes	21-46 seconds	2-3 seconds	4-5 seconds	1 second
Scalability	Not very scalable	Not very scalable	Not very scalable	Not very scalable	More scalability	Indefinite scalability
Node Count	6,000 nodes	3,173 nodes	1,603 nodes	1,243 nodes	1,997 nodes	443 nodes
Storage Costs	$73,000,000 / GB	Inadequate data storage	$1,000,000 / GB	$988,000 / GB	IPFS off-chain storage	$5 / GB
Cloud Service Dependency	70% of nodes run on AWS	Unclear how many are cloud	Most nodes run on cloud	Unclear how many are cloud	Most nodes run on cloud	Independent data centers

Can build big apps fully on chain

https://coincodex.com/article/14198/layer-1-performance-comparing-6-leading-blockchains/

DFINITY

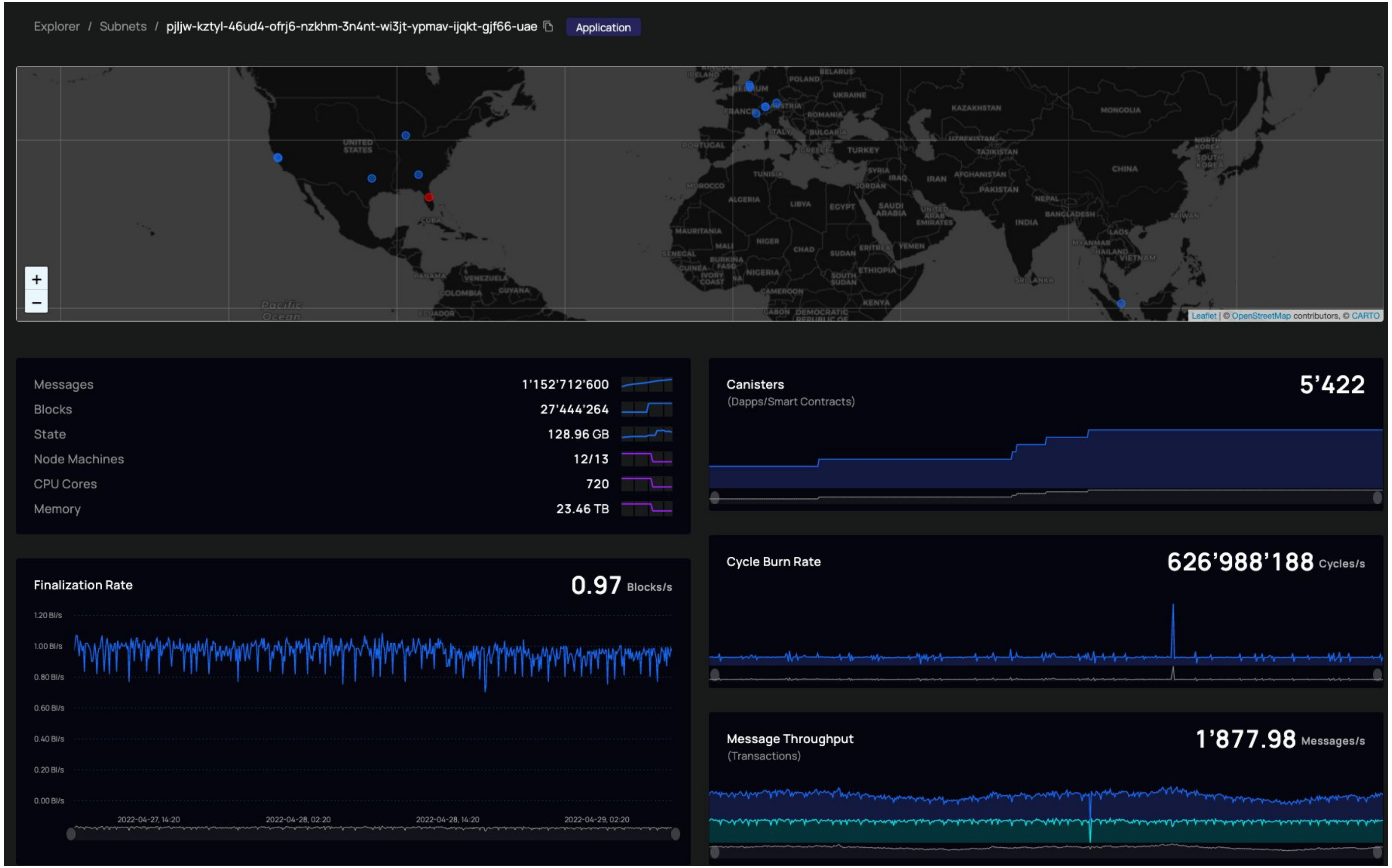
Fast Growing Ecosystem



Intra-Subnet P2P Networking

- Peer-to-peer network of nodes
  - Gossip protocol for artifact distribution
    - Advert - Request - Response
  - Eventual / bounded time delivery with priorities (~reliable broadcast optimized for Consensus)
- Untrusted communication
  - TLS / TCP to all nodes in the subnet, certificates in NNS
  - Authenticity and integrity of artifacts can be verified by higher layers
  - Nodes can still do evil

Example Subnet Dashboard



Testnets

DFINITY-internal infrastructure

- Deploy complete IC instances in our 5 data centers (2 more in May)
  - Chicago, San Francisco, Des Moines, Frankfurt, Zurich, ..
- Variable size and VM capabilities
- Can be used for experiments, metrics, correctness and performance tests

Logging

- Events can be logged in the code
- Log can be fetched from testnet machines
- Policy monitoring with MonPoly from Prof. Basin’s group

Case Study: “Idle” vs. Workload Traffic

31 nodes deployment

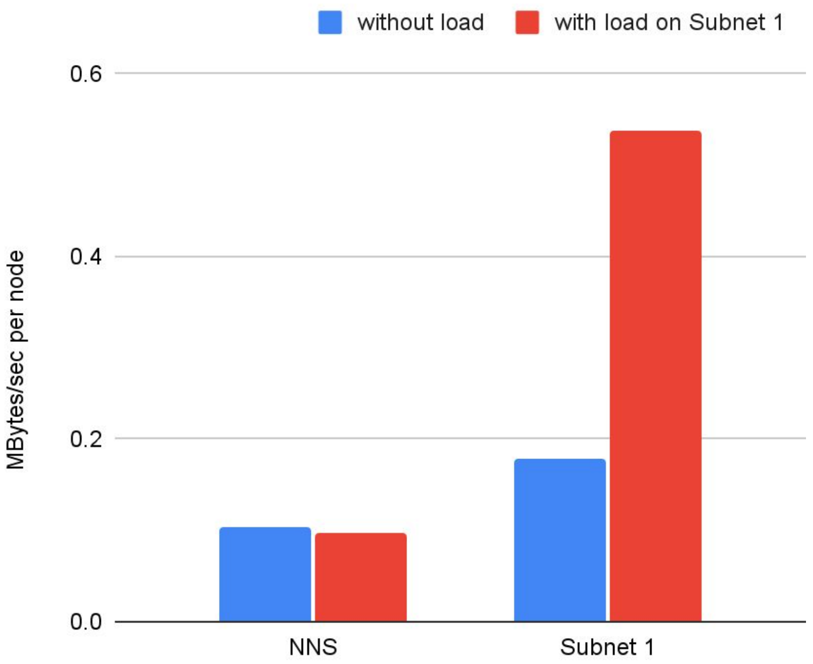
- 13 in NNS
- 18 in Subnet 1

Workload generation

- only in Subnet 1
- 100 requests per sec
- 1 kb each

Conclusion

- ICP produces 0.1-0.2MBytes/s for the protocol to make progress.



DFINITY

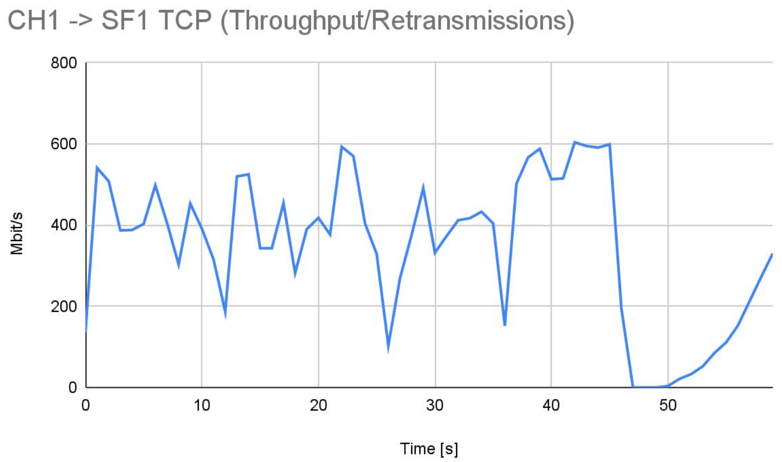
Case Study: “Intra DC” == Internet

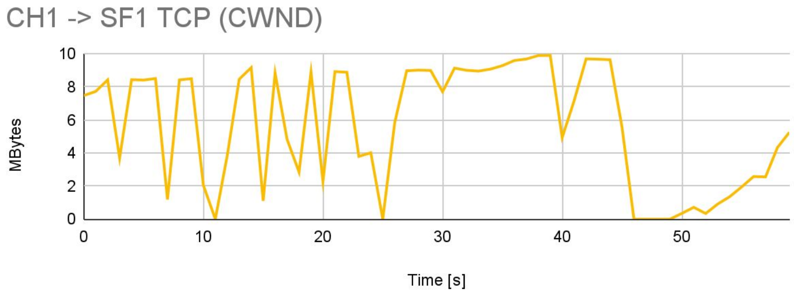
iperf between testnet hosts

- Chicago to San Francisco
- 60s in total

Conclusion

- Packet loss has a significant impact on the achieved throughput.





More information

- Infographic: here
- Technical Library: here (videos of talks) and here (blogposts)
- 200,000,000 CHF Developer Grant Program here
- DFINITY SDK: here

DFINITY