# Modern Blockchains for the Modern Security Engineer

ACM CCS Defi Workshop, Salt Lake City, USA
Oct 18, 2024

Prof George Danezis
University College London
Mysten Labs, Chief Scientist

# Brief Introduction

**George Danezis, Prof of Security and Privacy Engineering**

| | |
|---|---|
| 2000 | Cambridge |
| 2005 | KU Leuven |
| 2007 | MSR |
| 2013+ | UCL |
| 2018 | Chainspace (co-founder) |
| 2019 | FB: Libra, Diem, Novi |
| 2021+ | Mysten Labs (co-founder): Sui, Walrus |

Advisor to Vega Protocol, Nym Technologies, Celestia

# Outline: 4 Theses on Modern Blockchains

**Thesis 1** - Traditional blockchains set the vision but **lacked in realization** - yet even today set the research agenda.

**Thesis 2** - Modern blockchains, in contrast, **embody state of the art** systems, security and cryptography components.

**Thesis 3** - As systems modern blockchains implement **a traditional commercial security policy** framework - familiar to security engineers.

**Thesis 4** - Modern blockchains are the **best current platform** to build open distributed security systems. **Case Study**: Walrus.

# What is a blockchain?

**A secure decentralized transaction processing system & database**

Security: Consistency, Liveness, end-to-end verifiability and full auditability

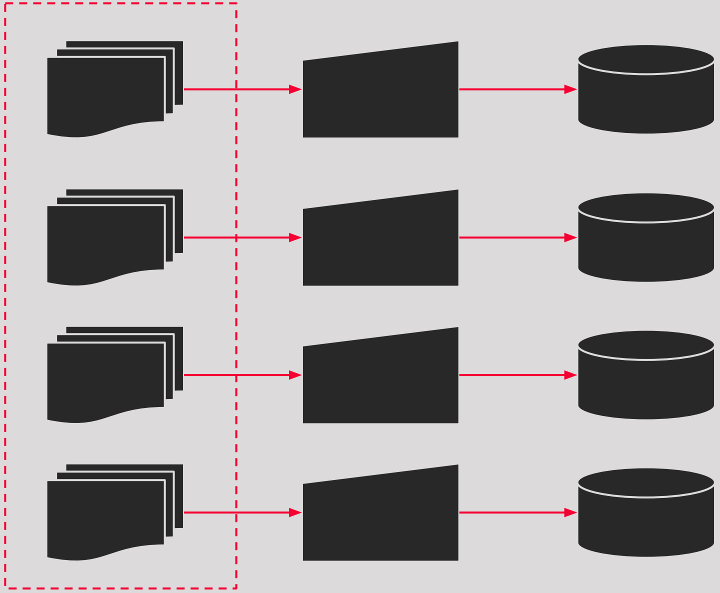
Based on the State Machine Replication (SMR) paradigm (Lamport, 1978)

## State Machine Replication: a reminder

Consistent Command Sequence (Consensus)

Deterministic Execution

Consistent State



# “Traditional” Blockchains, and their properties

Bitcoin (BTC $1.2T) & Ethereum (ETH $0.3T)

| | |
|---|---|
| **High Latency** | BTC: N x 10 min, ETH: N x 12 sec |
| **Low Throughput** | BTC: 9 tps, ETH: 50 tps |
| **High Fees** | BTC: $0.5 /tx, ETH: $1.5 /tx |
| **High energy usage** | BTC: 167 TWh/year, ETH: now PoS |
| **Probabilistic finality** | 1 block reorgs are routine, longer occur |
| **Restricted & unsafe exec** | BTC: very restricted bytecode,<br>ETH: untyped EVM multi-million $ hacks |

Maybe good enough for some use cases: store of value, NFT, Defi.

# Lots of blockchain system research improves upon Traditional Blockchains

### Increase capacity

Ethereum L2s
Zk rollups
Optimistic rollups
Lightning Network
Plasma
Sharding

### Lower latency

Off-chain
BFT sidechains
Lightening
State channels

### Lower power

Eth DPoS migration

### Stronger Finality

BFT based L2s / side-chains
Finality gadgets

### Execution & Safety

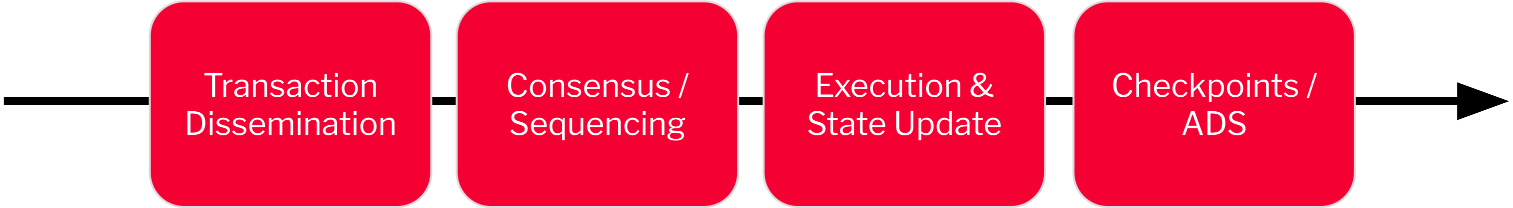
Solidity Audits & ML
EVM verification
Parallel execution
Custody & Multisig
Light clients

**Improvements that increase complexity ⇒ reduce assurance or performance**

# Modern Blockchains

Mysten Labs

# The Common Architecture: DPoS & BFT & VM & Merkle Trees




**Transaction Dissemination**

Validation, p2p dissemination, mempool.

Or direct communication

**Consensus / Sequencing**

Low-latency, high-throughput byzantine fault tolerant consensus.

Delegated Proof of Stake determined committee

**Execution & State Update**

Modern VM (Wasm, Move, eBPF)

Parallel execution of transactions

**Checkpoints / ADS**

State commitment

Database optimized for reads / RPCs

Command-Query Responsibility Segregation

Samples: Sui, Libra / Diem / Aptos, Solana, Cosmos ecosystem

# Move Programming Model ➡ Object Model

```
/// A basic Hello World example for Sui Move, part of the Sui Move intro course:
/// https://github.com/sui-foundation/sui-move-intro-course
///
module hello_world::hello_world {

    use std::string;
    use sui::object::{Self, UID};
    use sui::transfer;
    use sui::tx_context::{Self, TxContext};

    /// An object that contains an arbitrary string
    public struct HelloWorldObject has key, store {
        id: UID,
        /// A string contained in the object
        text: string::String
    }

    #[lint_allow(self_transfer)]
    public fun mint(ctx: &mut TxContext) {
        let object = HelloWorldObject {
            id: object::new(ctx),
            text: string::utf8(b"Hello World!")
        };
        transfer::public_transfer(object, tx_context::sender(ctx));
    }

}
```

**Modules** are the unit of Isolation and encapsulation. Outside code cannot construct / destruct structures, or directly access attributes.

**Structures** with key ability define top level objects with **unique IDs**. Note the **Linear type system**.

**Public functions** may be called from outside the module.

A **Programmable Transaction Block** is a sequence of calls to public functions executed atomically.

The context provides access to the **authenticated signer** of the transaction.

Public Transfer sends the top level objects to a new **owner**.

# A 1:1 Atomic Swap: Shared Objects, Assertions, Generics

```
module basic_swap::basic_swap;
use sui::coin::Coin;

public struct Swap<A, B> has key {
    id: UID,
    creator: address,
    creator_coin: Coin<A>,
}

public fun init_swap<A:store, B:store>(creator_coin: Coin<A>, ctx: &mut TxContext){
    let id = object::new(ctx);
    let creator = ctx.sender();
    let swap = Swap<A,B>{ id, creator, creator_coin, };
    transfer::share_object(swap);
}

public fun cancel<A,B>(swap: Swap<A,B>, ctx: &mut TxContext): Coin<A> {
    let Swap { id, creator, creator_coin } = swap;
    assert(ctx.sender() == creator, 0x0); // Authorization check
    object::delete(id);
    creator_coin
}

public fun swap<A,B>(swap: Swap<A,B>, coin_b: Coin<B>, ctx: &mut TxContext) : Coin<A> {
    let Swap { id, creator, creator_coin } = swap;
    assert(coin_b.value() == creator_coin.value(), 0x0); // Amount check
    transfer::public_transfer(coin_b, creator);
    object::delete(id);
    creator_coin
}
```

Define a **generic** top level struct that holds a coin from a creator.

Initialize the swap object, and set the **concrete types A and B** at runtime.

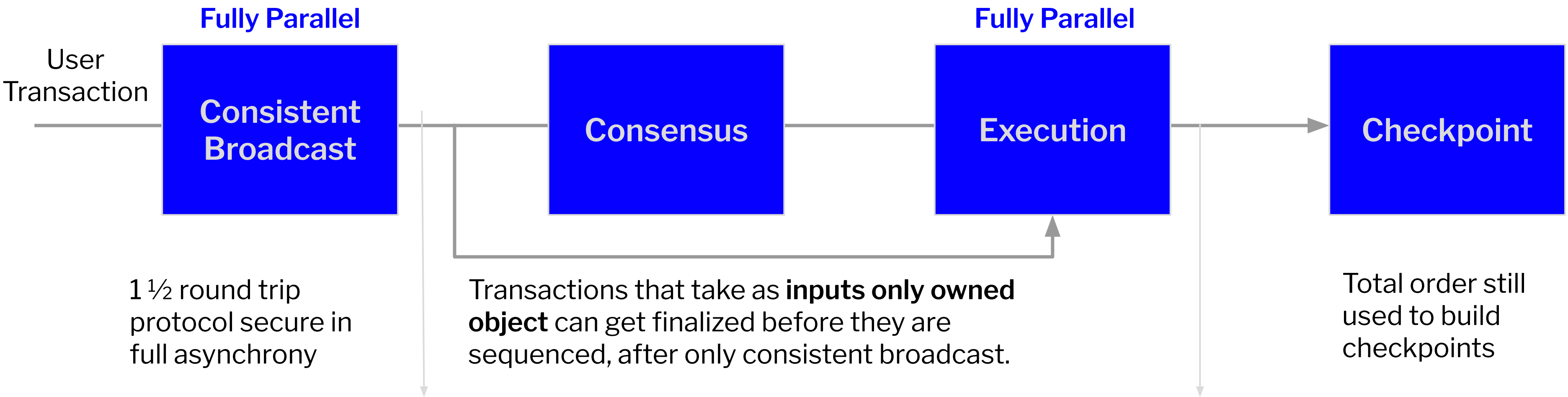
Make it **shared** - now anyone can use it in transactions.

**Assertion**, creator only may cancel swap and get back the coin.

Anyone may do the swap by providing the creator an **owned** coin of the second type with the correct value.

**Atomic** transaction execution ensures atomic swap.

# Fast Path with Sui Lutris: Finality Before Consensus for Owned objects




1 ½ round trip protocol secure in full asynchrony

Transactions that take as **inputs only owned object** can get finalized before they are sequenced, after only consistent broadcast.

Total order still used to build checkpoints

**Transaction finality:** the transaction will proceed to execute no matter what.

**Execution finality:** the given effects will persist no matter what.

**Sui Lutris: A Blockchain Combining Broadcast and Consensus.** Blackshear, Sam ; Chursin, Andrey ; Danezis, George ; Kichidis, Anastasios ; Kokoris-Kogias, Lefteris ; Li, Xun ; Logan, Mark ; Menon, Ashok ; Nowacki, Todd ; Sonnino, Alberto ; Williams, Brandon ; Zhang, Lu. **ACM CCS 2024**

# Fast Consensus with the Mysticeti DAG

All Validators make **blocks** in **rounds**

Contain **transactions** and **backlinks** to ⅔ previous blocks

A block at r may have a **skip or cert pattern** at r+2

Define **decision blocks**

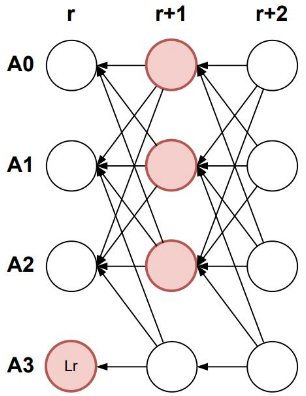
If ⅔ **r+2 blocks have a pattern** for block at r, decide!

Otherwise **continue, and decide later**.

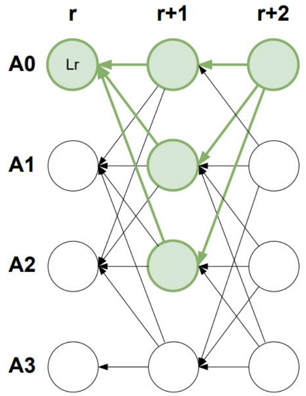
**Symmetric network utilization.**

**One network primitive to optimize: broadcast sync.**

Assumption: ⅔ stake correct & partial synchrony.



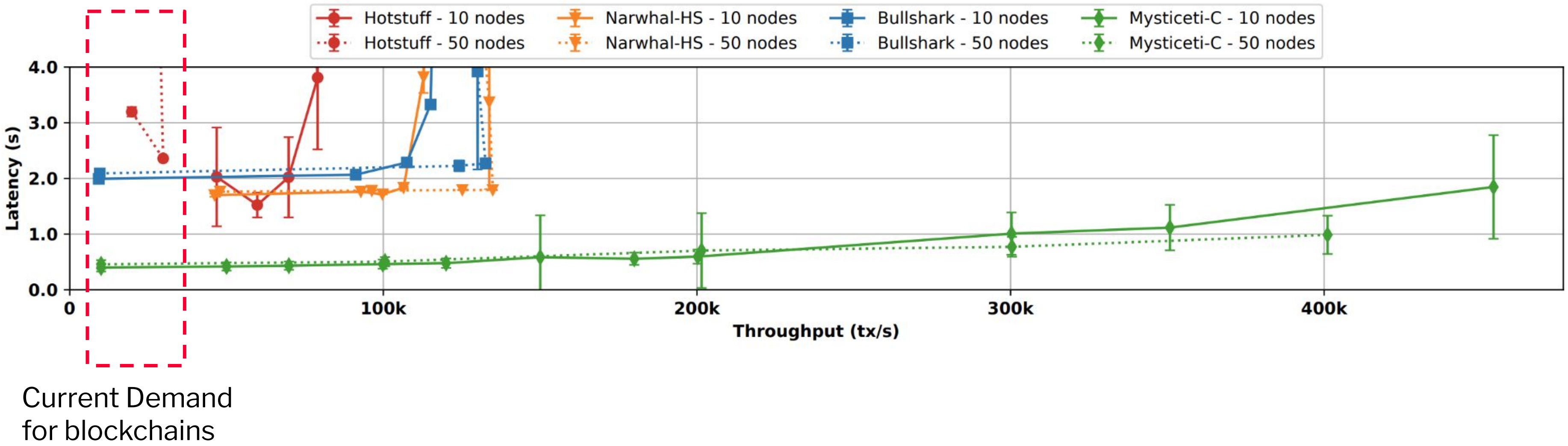

(a) Illustration of *skip* pattern, blocks $(A_0, r+1, \cdot), (A_1, r+1, \cdot), (A_2, r+1, \cdot)$ do not support $(A_3, r, L_r)$.




(b) Illustration of *certificate* pattern, block $(A_0, r+2, \cdot)$ is a certificate for $(A_0, r, L_r)$.

**Mysticeti: Reaching the Limits of Latency with Uncertified DAGs.** Babel, Kushal ; Chursin, Andrey ; Danezis, George ; Kichidis, Anastasios ; Kokoris-Kogias, Lefteris ; Koshy, Arun ; Sonnino, Alberto ; Tian, Mingwei. **NDSS 2025.**

# Laboratory Performance - Mysticeti alone

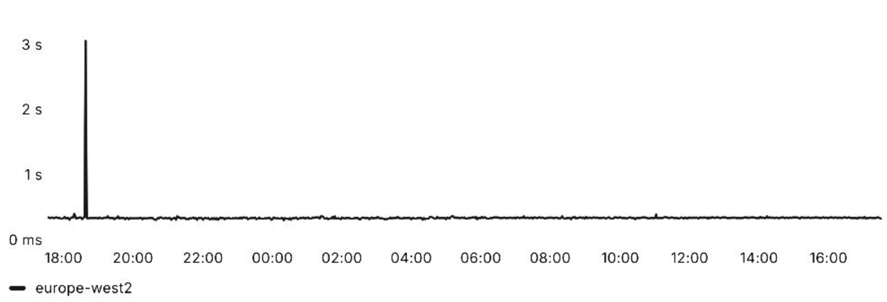



Key insight: **separating data dissemination from agreement** on metadata using a worker primary architecture leads to **practically limitless throughput** at the cost of 1 round trip of latency.

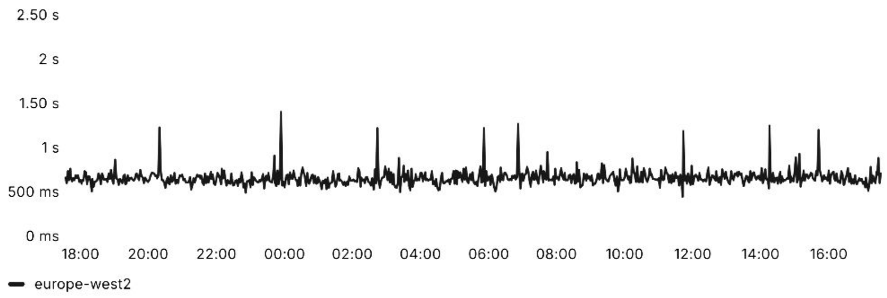
**Narwhal and Tusk: a DAG-based mempool and efficient BFT consensus.** George Danezis, Lefteris Kokoris-Kogias, Alberto Sonnino, Alexander Spiegelman. **EuroSys 2022: 34-50**

# Lutris & Mysticeti Latency in Production - 106 nodes, mainnet

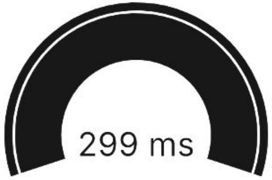
Latency (owned object)




Latency (shared object)



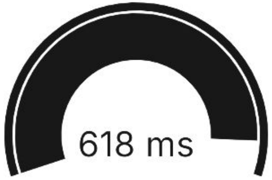

p25 Owned Object Latency




p50 Owned Object Latency


p90 Owned Object Latency


p25 Shared Object Latency




p50 Shared Object Latency


p90 Shared Object Latency

# Flexible Authentication & ZKLogin

## Table Stakes Authentication & Cryptography

**Basic Signature Schemes**

Ed25519, ECDSA Secp256k1 & Secp256r1

**Native Multi-signature**

Define up to 10 public keys, weight and threshold

Valid if the weight of all signatures exceeds threshold

Can mix & match schemes

**Move Crypto**: BLS12381, Groth16 Verifier, SHA256, SHA3-256, blake2b256, keccak256

## ZKLogin Authentication

Generate an Ephemeral key pair

Generate a JSON Web Token (JWT)

Request the user's unique salt or use PIN

Generate a zk proof

Sign transaction with ephemeral key, and authorize key with the zkproof

**Result: can authorize on-chain action using OAuth**

**zkLogin: Privacy-Preserving Blockchain Authentication with Existing Credentials.** Foteini Baldimtsi, Konstantinos Kryptos Chalkias, Yan Ji, Jonas Lindstrøm, Deepak Maram, Ben Riva, Arnab Roy, Mahdi Sedaghat, Joy Wang. **ACM CCS 2024**

# Demo: Receive Sui Tokens via QR Code + Gmail/Twitch login

getstashed.com/claim...

# Again…

getstashed.com/claim...

# Secure Time, Native Randomness, Fresh nonces

## Real time clock

Mysticeti blocks contain time

All blocks include ⅔ previous blocks, and their time

All committed blocks have ⅔ subsequent blocks (cert)

Bounds checks prevent Byzantine validators from going too slow or too fast

**Move**: Clock shared object is updated with the commit block time

## Native Secure Randomness

Each epoch validators run a DKG

For each round validators reveal shares of a BLS signature on the round number

This is guaranteed to be fresh and not guessable

**Move**: functions may read the round randomness from a randomness shared object

## Fresh nonces

All Sui object IDs are guaranteed to be fresh

This is done via cryptographic hashing and lamport timestamps

**Move**: may request fresh identifiers, and use them to identify Capabilities or other actions

# Secure Capability Authorization and Programming with Types

**Updating contracts** controlled by capability.

**Regulated coin operations** controlled by Capability.

Ownership checked at system level.

Capability pattern supported through the linear type system: by default cannot create, clone, copy, or drop objects.

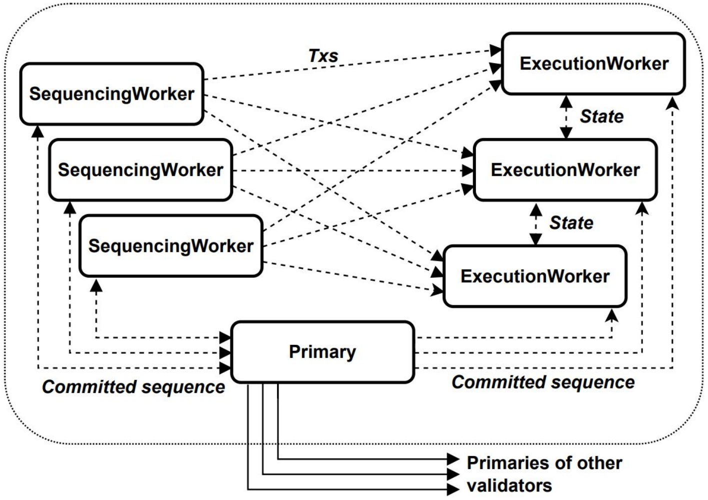
Only through well defined module functions.

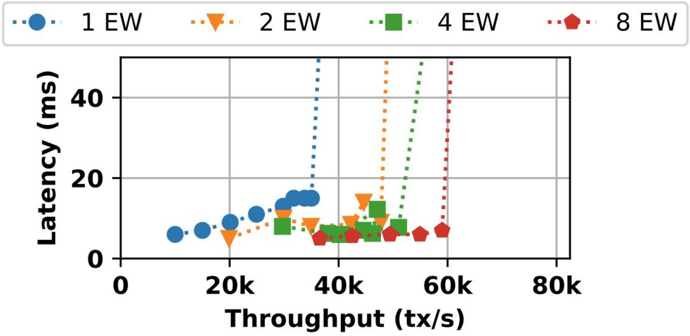
**Having an object of a type can denote authorization to act on it.**

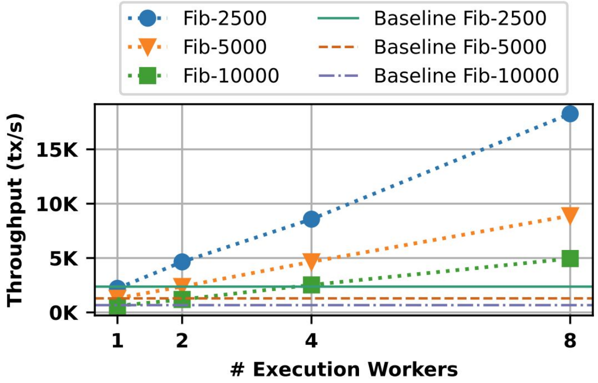
# Distributed Execution with Pilotfish

Today: Leverage **parallel execution** of transactions on independent objects

Tomorrow: **distributed execution**!




**Pilotfish: Distributed Transaction Execution for Lazy Blockchains.** Kniep, Quentin ; Kokoris-Kogias, Lefteris ; Sonnino, Alberto ; Zablotchi, Igor ; Zhang, Nuda. arXiv:2401.16292

# The big picture

**Modern Blockchains:**

Low-latency
High-throughput
Cheap fees
Built in flexible auth (AAA)
Great security services: random, time, nonce
Great built-in crypto
Safe, expressive languages

**Best choice if you need consensus**
**Best choice if you need consistent broadcast**
**Great if you need isolated VM**
**Great if you want parallel execution**
**Great if you want to re-use SSO**

# Great technical pieces.

# A modern blockchain as a whole system?

# The Clark-Wilson (CW) commercial security policy framework (1987)

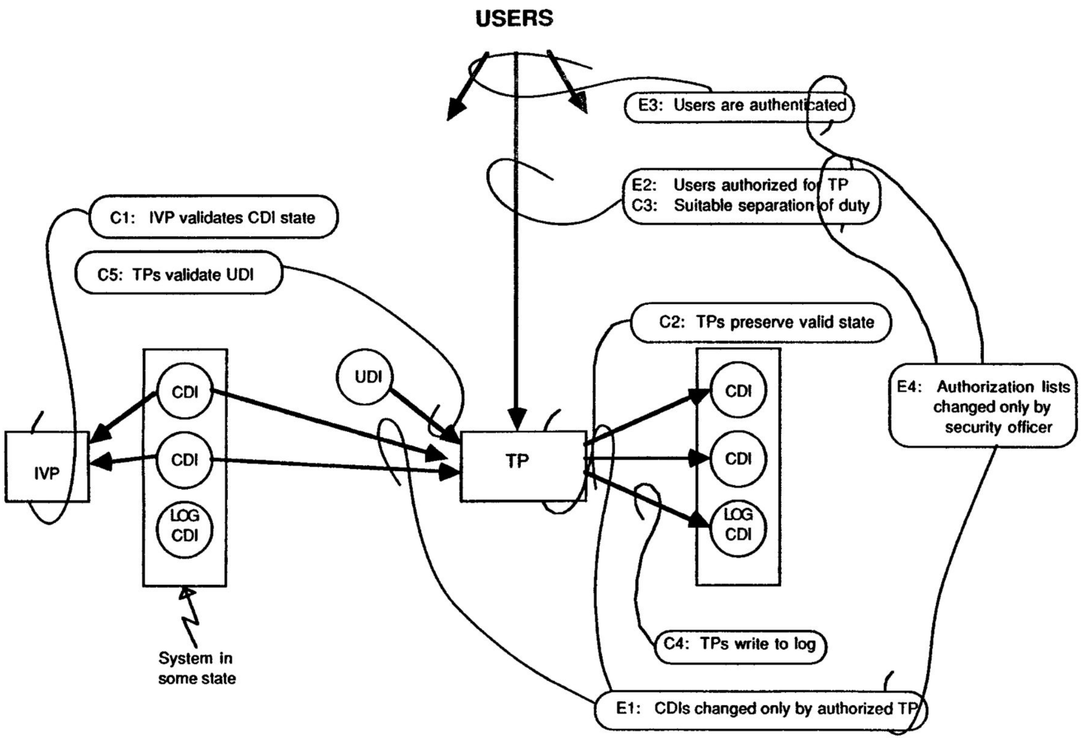
Commercial security needed a framework focused on authenticity, integrity and audit.

In contrast with Orange book MLS.

Policy framework mapping controls to a transaction processing system.

**Key thesis: Modern Blockchains provide the most high-assurance, performant, ergonomic and featureful platform for implementing a CW policy.**

Figure 1: Summary of System Integrity Rules



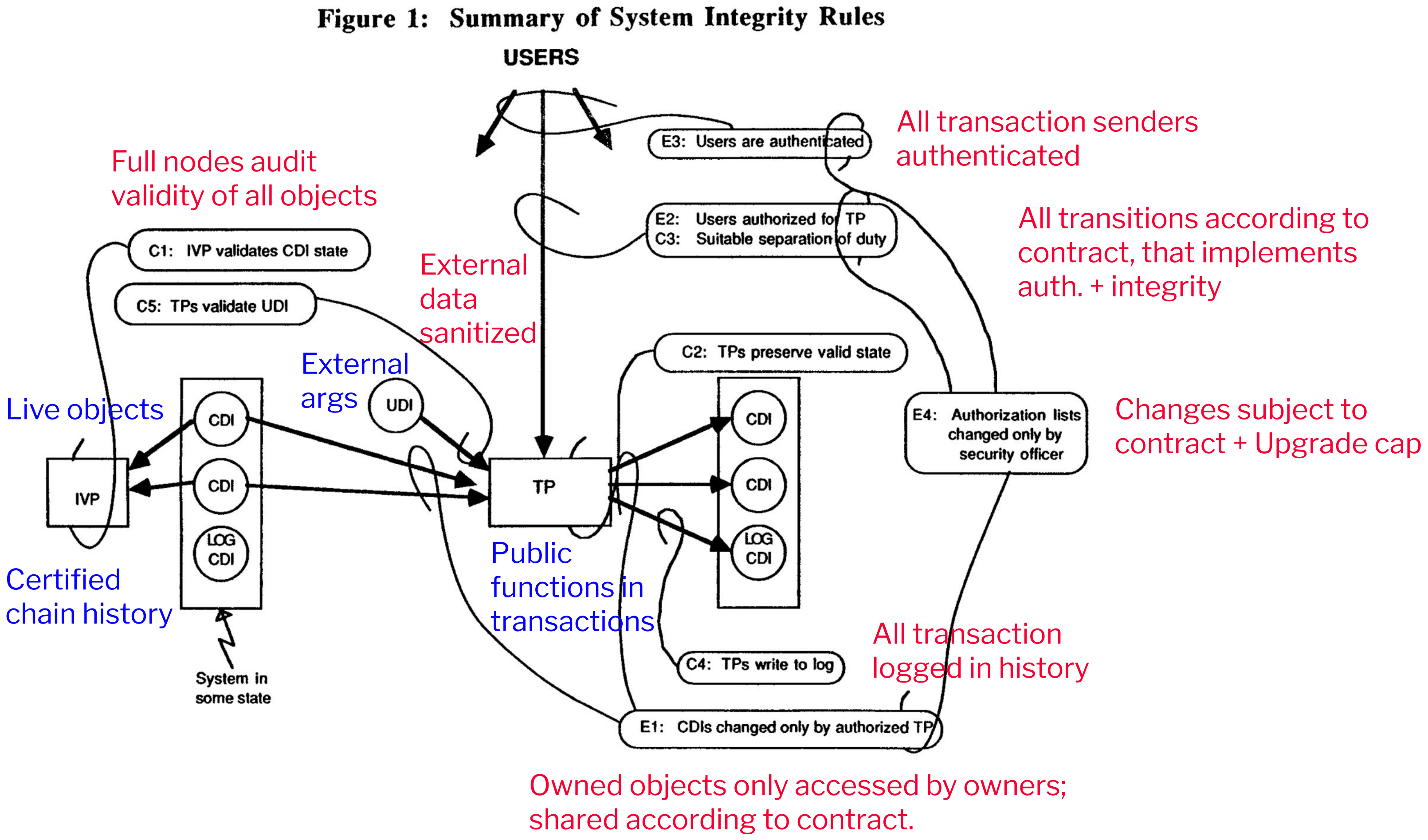


Figure 1: Summary of System Integrity Rules
USERS
E3: Users are authenticated
All transaction senders authenticated
Full nodes audit validity of all objects
E2: Users authorized for TP
C3: Suitable separation of duty
All transitions according to contract, that implements auth. + integrity
C1: IVP validates CDI state
C5: TPs validate UDI
External data sanitized
C2: TPs preserve valid state
External args
UDI
Live objects
CDI
CDI
LOG CDI
IVP
TP
E4: Authorization lists changed only by security officer
Changes subject to contract + Upgrade cap
Certified chain history
Public functions in transactions
System in some state
C4: TPs write to log
All transaction logged in history
E1: CDIs changed only by authorized TP
Owned objects only accessed by owners; shared according to contract.

# Level of Assurance Provided

**Quorum Unconditional Validity** - based on security of sender signatures.

**Safety under ⅔ correct quorum & asynchrony**, ie. Byzantine Fault Tolerance.

**Liveness under partial synchrony**.

Actuals:

- End-to-end audit trail based on public verifiability + cryptographic authentication.
- 100x+ geo-distributed fully replicated execution. Real-time.
- 500x+ real time validity verification and further replication. Real-time & audit.

**Compare with Traditional Trusted Computing Base (TCB) = a computer with an administrator & some backup computer.**

# Modern Blockchains from a Security Engineer's Perspective

**Define security policy through smart contract**

Objects / Structures: define Constrained Data Items (CDI).

Public functions: define security policy for the application.

Transformation Procedures (TP) sanitize UDI to CDI.

Transformation Procedures (TP) mutate CDI to CDI.

Define access control for shared state CDI + TP.

Define rules to change access control, subject to policy.

**Smart contracts are a security policy language and blockchains the systems that run and enforce it as a CW policy.**

**Get for free**

- Authentication + Authorization (owned objects)
- Secure + private SSO integration.
- Audit log CDI.
- All TP in tamper evident history + certified.
- All CDI transitions follow policy.
- High throughput, Low-latency, cheap
- Open system: economics, DoS protection.
- Security services: randomness, time, crypto functions.

**Secure composition via using common system.**

**Objects (CDI) and TPs from one realm can be securely composed by other modules to construct complex interoperable secure application.**

# Case Study: Walrus Decentralized Storage

Mysten Labs

# Decentralized Storage, in the past

## The Classical Era

Centralized Systems

Unstructured peer-to-peer systems

Distributed Hash Tables

Bittorrent

**No transactional semantics**

**No erasure coding - requires coordination**

## The Traditional Blockchain Era

IPFS

Filecoin

Arweave

**Build a traditional blockchain and storage**

**Full-replication**

# Walrus: Decentralized Storage in the Era of Modern Blockchains

## Protocol Outline

**Committee of Storage Nodes in epochs**

**Write**

1. Erasure code blob, derive Blob ID and size.
2. Buy storage and register blob ID on chain.
3. Upload shares on all storage nodes.
4. Get signatures if shares valid.
5. Make ⅔ Proof of Availability certificate.
6. Certify the Blob ID on blockchain.

**Point of Availability**

**Read:**

7. Read Blob ID from ⅓ of storage nodes.
8. Reconstruct Blob + Check Blob ID.

See https://docs.walrus.site/

## Usage of Modern Blockchain

Manage the **storage node committee** in epochs.
Delegated proof of stake mechanism.

Manage the **assignments of shares** to storage nodes.

Manage the **price** and amount of **free space**.

Get **payments** for buying empty storage.
Secondary storage market.

Register & Certify Blobs = **Prove Availability**.

**Extend & Delete Blobs** if authorized.

**Report Invalid** Blob encodings.

Manage **deny list** for compliance.

Coordinate **epoch change**, ready and done.

**A secure decentralized consistent core.**

# What open distributed infrastructures do you want to build?

## The Cookbook

Define **governance as smart contract** on Modern Blockchain: payments, control, resources, consistent core.

**Off-chain infrastructure** uses Modern Blockchain events to update local state machines.

Do meta-data management on-chain, crypto protocol off-chain.

…

$$$.

## Distributed things we do not have …

Secure cryptographic election as a Service.

Private Information Retrieval as a Service

Multi-Party Computation as a service

Prover farms as a Service

Public-Key Infrastructures & Certificate Transparency & Routing table maintenance.

ORAM Services.

Mysten Labs

# In conclusion

# In conclusion: the old ways, and the new ways

There was a time when Security engineers:

- Would build own cipher
- Would design own authentication protocol
- Would design own channel encryption
- Would implement own authorization framework
- Or write your own database?

**Bad idea: specialized tasks, that are best done by small expert teams** with high assurance and re-used by all.

**Modern Blockchains: Same dynamic for high-integrity public applications.**

- Security Engineers that wanted to build a secure public app involving transactions would start from scratch.
- There is however no way to compete with starting with a modern blockchain.

**Added benefit: composability** - no matter how good a standalone secure app is, it is hard to make it work with others.
**Added benefit: naturally open and networked** - perfect for coordinating other decentralized systems.

**Key challenge: integrate confidentiality policies, without sacrificing benefits.**