Sui Lutris: A Blockchain Combining Broadcast and Consensus

Alberto Sonnino
Byzantine Fault Tolerance
Byzantine Fault Tolerance

> 2/3
Typical Architecture

Mempool / Initial Checks
- P2P flood & Selection on fee
- Overlay flooding slow and with significant redundancy

Ordering
- Sequence all transactions in blocks
- Seconds latency, traditionally low throughput

(Sequential) Execution
- Execute each transaction (global lock)
- Single core does all computations. (eg EVM ~300 tps)

DB Update & High-Integrity DS
- Update DB, indexes, crypto (Merkle trees)
- Added latency of store, blocks, and crypto computations
**Typical Architecture**

- **Mempool / Initial Checks**
  - Overlay flooding slow and with significant redundancy
  - Seconds latency, traditionally low throughput

- **Ordering**
  - Sequence all transactions in blocks

- **Execution**
  - Execute each transaction (global lock)
  - Single core does all computations. (eg EVM ~300 tps)

- **Update DB, High-Integrity DS**
  - DB Update & High-Integrity DS
  - Added latency of store, blocks, and crypto computations
New Architecture
Secure Combination

FastPay + Narwhal = Bullshark
The Sui Lutris System
Architecture

Transaction → Consistent Broadcast → Consensus → Checkpoints, Merkle Trees → Agreed sequence for audit/sync

Contains shared-objects?

Execute → Parallel Execution → Certificate without consensus

Execute → Certificate with consensus
New Data Model
Consensus is not required

- Coins, balances, and transfers
- NFTs creation and transfers
- Game logic allowing users to combine assets
- Inventory management for games / metaverse
- Auditable 3rd party services not trusted for safety
- ...
New Data Model
Consensus is required*

- Increment a publicly-accessible counter
- Collaborative in-game assets
- Auctions
- Market places
- ...

*Note: Additional context or details may be required to fully understand the consensus process.
Consensus only when you need to
New Architecture

**Owned Objects**
- Objects that can be mutated by a single entity
- e.g., My bank account
- Do not need consensus

**Shared Objects**
- Objects that can be mutated by multiple entities
- e.g., A global counter
- Need consensus
Objects:

- Unique ID
- Version number
- Ownership Information
- Type (shared, owned)
Sui Transaction

Objects:
- Unique ID
- Version number
- Ownership Information
- Type (shared, owned)

Transaction’s content:
- Package, function
- Object Inputs
- Arguments
- Gas Information
- Signature

Coin::Send
- Alice’s account
- Bob’s account, Balance=5
- 0.001, max=0.005
Example Transaction

T1

**Inputs:** O1 (v10), O2 (v27), O3 (v1001)

**Output:** Mutate O1, Transfer O2, Delete O3, Create O4
**Send T1:** Disseminate the transaction

**Echo T1:** Nodes check and sign T1

**Cert T1:** User gather >2/3 signatures into a certificate and disseminate it

**Effect T1:** User gather >2/3 effect signatures for finality
**Send T1:** Disseminate the transaction

**Echo T1:** Nodes check and sign T1

**Cert T1:** User gather >2/3 signatures into a certificate and disseminate it

**Effect T1:** User gather >2/3 effect signatures for finality
Step 1: Owned object locks & version exist at validator

L1 = (O1, 10)
Owner=X : None

L2 = (O2, 27)
Owner=X : None

L3 = (O3, 1001)
Owner=X : None

We call these “locks”, and are initialised to None.
Step 2: Validator V checks / signs transactions

Transaction: T1
Inputs: (O1, 10), (O2, 27), (O3, 1001)
Move call details
Signature of X

Checks T1 (Validity)
• Well-formed (syntactic)
• Valid Signature from X
• Valid execution function
• Version owned by X

Checks T1 (Broadcast)
• Objects exist and lock is None
• Set lock to T1
Consensus-less Path

**Send T1:** Disseminate the transaction

**Echo T1:** Nodes check and sign T1

**Cert T1:** User gather >2/3 signatures into a certificate and disseminate it

**Effect T1:** User gather >2/3 effect signatures for finality
Step 3: Validator V process certificate

- **O1**
  - \( L_1 = (O_1, 10) \)
  - Owner=\( X \) : None

- **O2**
  - \( L_2 = (O_2, 27) \)
  - Owner=\( X \) : None

- **O3**
  - \( L_3 = (O_3, 1001) \)
  - Owner=\( X \) : None

**Transaction: T1**
- Inputs: \((O_1, 10), (O_2, 27), (O_3, 1001)\)
- Move call details
- Signature of \( X \)
- Signature (V1, ... V4)

**Checks T1 (Validity)**
- Again!

**Checks T1 (Broadcast)**
- Objects exist (with any lock)
- Certificate signed by quorum
Step 4: Validator V executes / signs effect

Transaction: T1
Inputs: (O1, 10), (O2, 27), (O3, 1001)
Move call details
Signature of X
Signature (V1, ... V4)

Execute T1
- O1 mutated
- O2 transferred
- O3 deleted
- O4 created
**Consensus-less Path**

**Send T1:** Disseminate the transaction

**Echo T1:** Nodes check and sign T1

**Cert T1:** User gather >2/3 signatures into a certificate and disseminate it

**Effect T1:** User gather >2/3 effect signatures for finality
Integration with Consensus

**Send T1:** Disseminate the transaction

**Echo T1:** Nodes check and sign T1

**Cert T1:** User gather >2/3 signatures into a certificate and disseminate it

**Effect T1:** User gather >2/3 effect signatures for finality
Integration with Consensus

Example Transaction

T2

**Inputs:** O1 (v10), S2

**Output:** Mutate O1, Mutate S2, Create O4
Integration with Consensus

Send T1: Disseminate the transaction

Echo T1: Nodes check and sign T1

Cert T1: User gather >2/3 signatures into a certificate and disseminate it

Effect T1: User gather >2/3 effect signatures for finality
Integration with Consensus

Step 1: Shared object locks exist at validator

L1 = (O1, 10)
Owner=X : None

L2 = (S2, *)

Do not check the version for shared objects
Integration with Consensus

Step 2: Validator V checks / signs transactions

L1 = (O1, 10)
Owner=X : None T2

L2 = (S2, *)
Owner=X

Transaction: T2
Inputs: (O1, 10), (S2, *)
Move call details
Signature of X

Checks T1 (Validity)
• Well-formed (syntactic)
• Valid Signature from X
• Valid execution function
• Version owned by X

Checks T1 (Broadcast)
• Objects exist and lock is None
• Set lock to T1
Integration with Consensus

- **Send T1:** Disseminate the transaction
- **Echo T1:** Nodes check and sign T1
- **Cert T1:** User gather >2/3 signatures into a certificate and disseminate it
- **Effect T1:** User gather >2/3 effect signatures for finality
Step 3: After consensus, assign shared objects locks

### Integration with Consensus

**L1 = (O1, 10)**
**Owner=X : None**
**T2**

**Transaction: T2**
**Inputs: (O1, 10), (S2, *)**
**Move call details**
**Signature of X**

**Assign Shared Locks**
- Every node sees the same sequence out of consensus
- So they can all assign the same shared object locks
Integration with Consensus

Step 3: Validator V process certificate

L1 = (O1, 10)
Owner=X : None T2

L2 = (S2, 4)

Transaction: T2
Inputs: (O1, 10), (S2, *)
Move call details
Signature of X

Checks T2 (Validity)
• Again!

Checks T2 (Broadcast)
• Objects exist (with any lock)
• Certificate signed by quorum
Integration with Consensus

Step 4: Validator V Applies / Signs Effect

<table>
<thead>
<tr>
<th>O1</th>
<th>L1 = (O1, 11)</th>
<th>Transaction: T2</th>
<th>Execute T1</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Owner=X : None</td>
<td>Inputs: (O1, 10), (S2, *)</td>
<td>• O1 mutated</td>
</tr>
<tr>
<td>S2</td>
<td>L2 = (S2, 4)</td>
<td>Move call details</td>
<td>• S2 mutated</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Signature of X</td>
<td>• O4 created</td>
</tr>
<tr>
<td>O4</td>
<td>L3 = (O4, 1)</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>Owner=X : None</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Send T1: Disseminate the transaction
Echo T1: Nodes check and sign T1
Cert T1: User gather >2/3 signatures into a certificate and disseminate it
Effect T1: User gather >2/3 effect signatures for finality
Transaction Execution

- First, execute all precedent transactions
- Once there is a certificate, any validator can download Tx and execute
Transaction Execution

Owned-objects

Core 1

Ox, 10

Tx

Ox, 11

Core 2

Oy, 65

Ty

Oy, 66

Always executed in parallel
(once they inputs ID/version are known)

Shared-objects

Core 3

Sv, ?

Tv

Sv, ?+1

Core 4

Sw, ?

Tw

Sw, ?+1

Often executed in parallel
(Sequentially for each shared object)
The Sui System
Shared objects

Assign locks:
A single task assigns versions, e.g., Ov=5 Ow=18

Execute:
Multiple tasks execute (if they can)
Transaction Execution

**Schedule**

Single task schedules transactions:

- \((Tx1, Sv) \rightarrow 5\)
- \((Tx1, Sw) \rightarrow 17\)
- ...
- \((Tx2, Sw) \rightarrow 6\)

**Execute**

Many tasks try to execute transactions:

- \((Tx1, Sv) == db[Sv]\)
- \(db[Sv] += 1\)

**Missing owned-objects dependency?**

- Tell the client
- Synchronise
- Retry
What we didn’t cover

- (Very) Detailed Algorithms
- Checkpointing
- Reconfiguration
- Proofs
- Production-readiness Insights
- ...

---

**Sesi Lutris: A Blockchain Combining Broadcast and Consensus**

<table>
<thead>
<tr>
<th>Name</th>
<th>Affiliation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sasi Bhaskar</td>
<td>Myntz Labs</td>
</tr>
<tr>
<td>Ammar Elkhaddar</td>
<td>Myntz Labs</td>
</tr>
<tr>
<td>Mark Logis</td>
<td>Myntz Labs</td>
</tr>
<tr>
<td>Kandis McCall</td>
<td>Myntz Labs</td>
</tr>
<tr>
<td>Lior Rozenfeld</td>
<td>University College London</td>
</tr>
<tr>
<td>...</td>
<td></td>
</tr>
</tbody>
</table>

**Abstract**

We describe a first production-grade smart contract platform that leverages consensus agreement to achieve sub-second ripple in deployment work, but in the public blockchain context. Our platform supports complex smart contracts, including a broad spectrum of governance-driven smart contracts, in a completely distributed manner. Our solution is based on a novel consensus protocol that achieves highly efficient and scalable performance, supporting state-of-the-art blockchain features.

**1 Introduction**

Traditional blockchains lack transaction speed, often resulting in delays and bottlenecks. “Broadcast-based” schemes, such as 0x, aim to solve this issue by introducing a decentralized network of nodes that validate transactions and ensure security. However, these schemes suffer from scalability issues, limiting their practicality in real-world applications. Sesi Lutris addresses this challenge by providing a highly scalable and secure platform for deploying and executing complex, stateful smart contracts. Our approach leverages a novel consensus mechanism that balances efficiency and security, enabling the deployment of sophisticated, real-world applications on a blockchain framework.

**Challenges**

- **Scalability:** Balancing the need for high throughput with the requirement for strong consensus agreement.
- **Security:** Ensuring the integrity and confidentiality of transactions while maintaining efficiency.
- **Interoperability:** Supporting the seamless integration of Sesi Lutris with existing blockchain ecosystems.

**Contributions**

- A scalable, secure, and efficient consensus protocol that enables the deployment of complex smart contracts.
- A robust framework for developing and deploying decentralized applications on a blockchain.
- A novel approach to consensus that supports the needs of the emerging blockchain ecosystem, bridging the gap between theoretical research and practical deployment.
Conclusion

The Sui Lutris System

- Separate owned and shared objects
- **Only use consensus when you need to**
- Execute in parallel whenever you can

- **Paper:** https://sonnino.com/papers/sui-lutris.pdf
- **Code:** https://github.com/mystenlabs/sui
alberto@mystenlabs.com