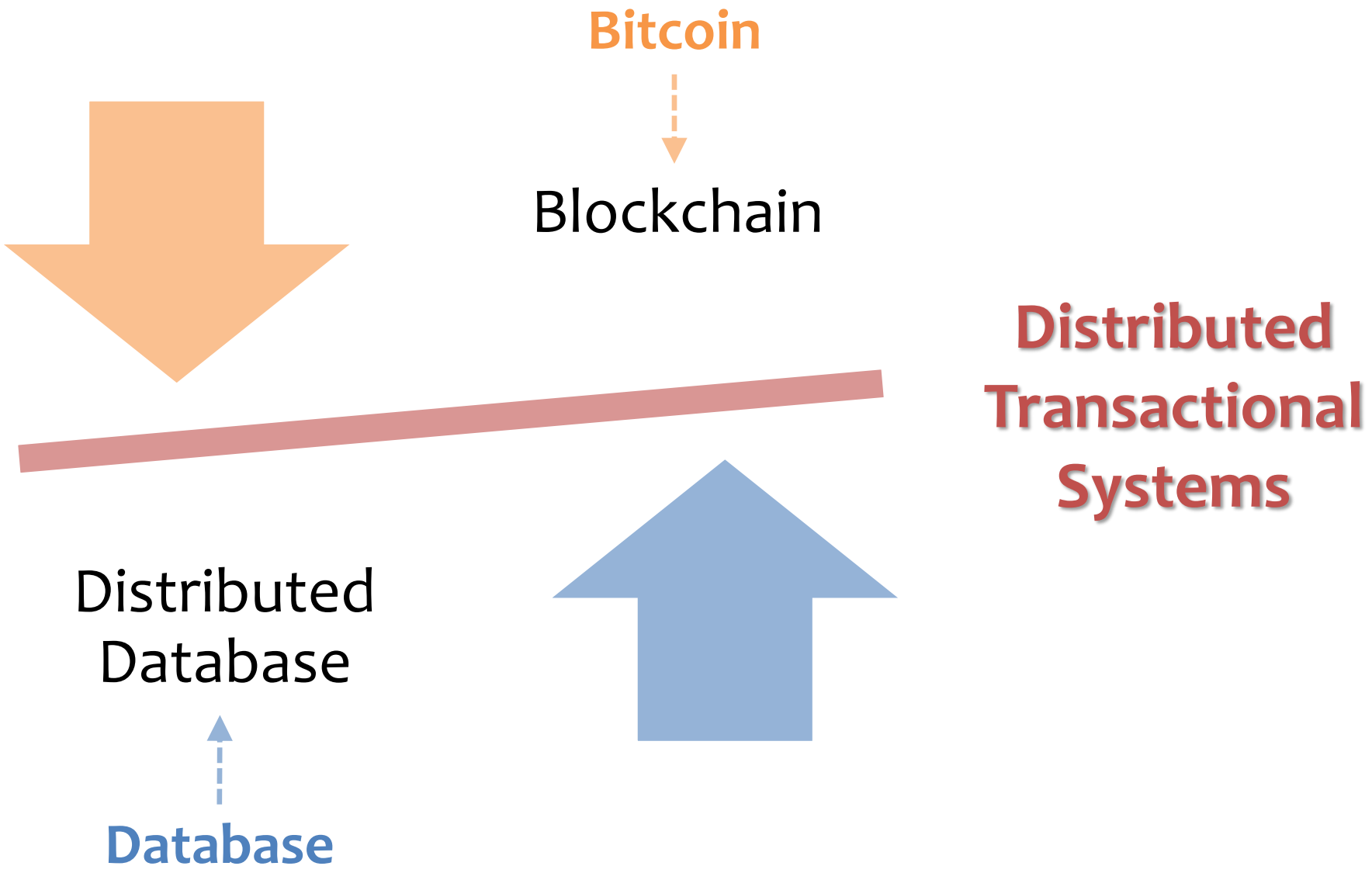
Fine-Grained, Secure and Efficient Data Provenance on Blockchain Systems

Pingcheng RUAN, Gang CHEN, Tien Tuan Anh DINH,
Qian LIN, Beng Chin OOI, Meihui ZHANG

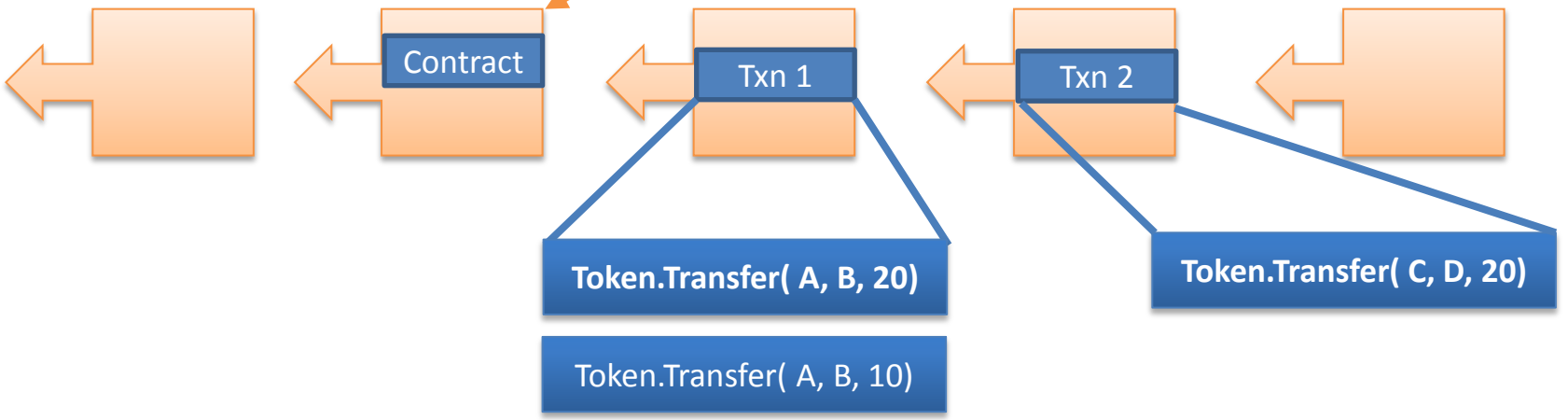
Blockchain Is a Class of Database



Blockchain Basics

- **P2P network**
  - Asynchronous transaction
- **Byzantine environment**
  - Mutual distrusting setup
- **Distributed ledger**
  - Smart contract
- **Inherent provenance-preserving**
  - ONLY for offline analytical query

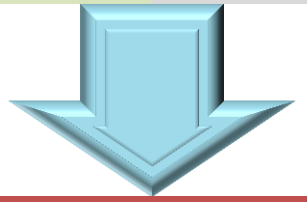
```
contract Token {
    method Transfer(sender, recipient, amount) {
        bal1 = gState[sender];
        bal2 = gState[recipient];
        if (amount < bal1) {
            gState[sender] = bal1 - amount;
            gState[recipient] = bal2 + amount;
} } }
```



Motivation

Expose provenance information to smart contracts both

Efficiently **Securely**



Enabler for provenance-dependent smart contracts

Enrich the transaction semantics

Provenance-dependent Contracts

- **Previous transfer precondition:**
  - Enough balance from the sender
  - CURRENT STATE ONLY
- **New transfer precondition:**
  - Historical balance > threshold
  - Recipient not transacted with certain blacklisted addresses recently
  - HISTORICAL STATE & PROVENANCE INFO

```
contract Token {
    method Transfer(sender, recipient, amount) {
        bal1 = gState[sender];
        bal2 = gState[recipient];
        if (amount < bal1) {
            gState[sender] = bal1 - amount;
            gState[recipient] = bal2 + amount;
} } }
```

Workarounds

Workaround 1:

- Dump every thing into current state
- **Effort-needed, expensive, error-prune**

Workaround 2:

- Offline analytics + Online transactions
- **Break of serializability**
- **Transaction-ordering attacks**

Workaround 3:

- Minimum system instrumentation
- NOT protocol level (e.g., Hyperledger Fabric v1.0+)
- **Data tampering**

Account1_v1: 10
Account1_v2: 20
Account1_v3: 15
Account2_v2: 12

Holistic Approach:

- **Protocol-level enhancement → Secure**
- **Performance-aware → Efficient**

Challenges

NO standardized operations

- With clearly-defined transformation semantics
- E.g
  - Map and reduce in Hadoop
  - Select, join and aggregation in SQL

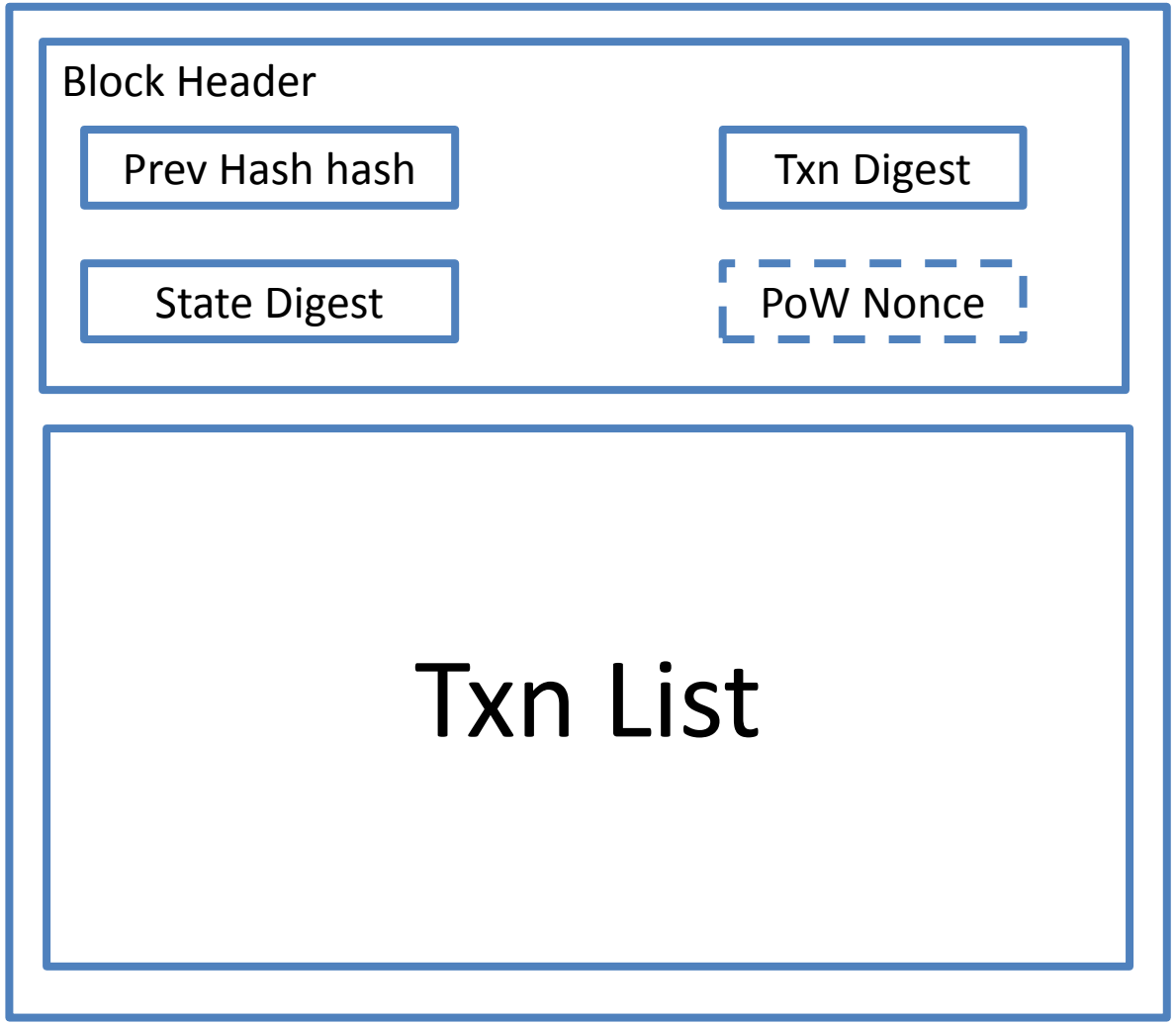
Byzantine environment

- Tamper evidence
- Integrity proof

Ever-growing ledger

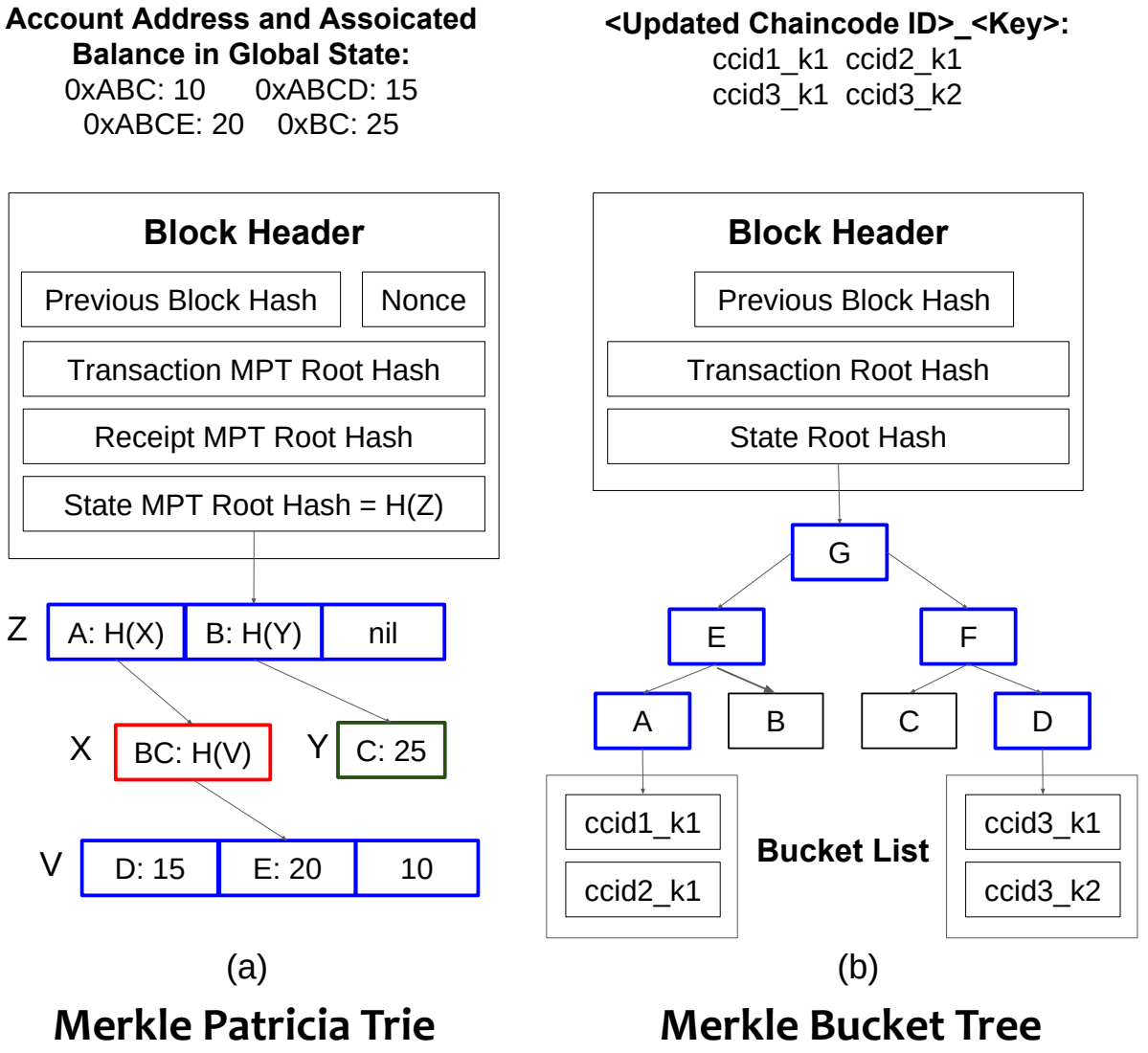
- Gas mechanism
- Verifier's dilemma

Block Structure

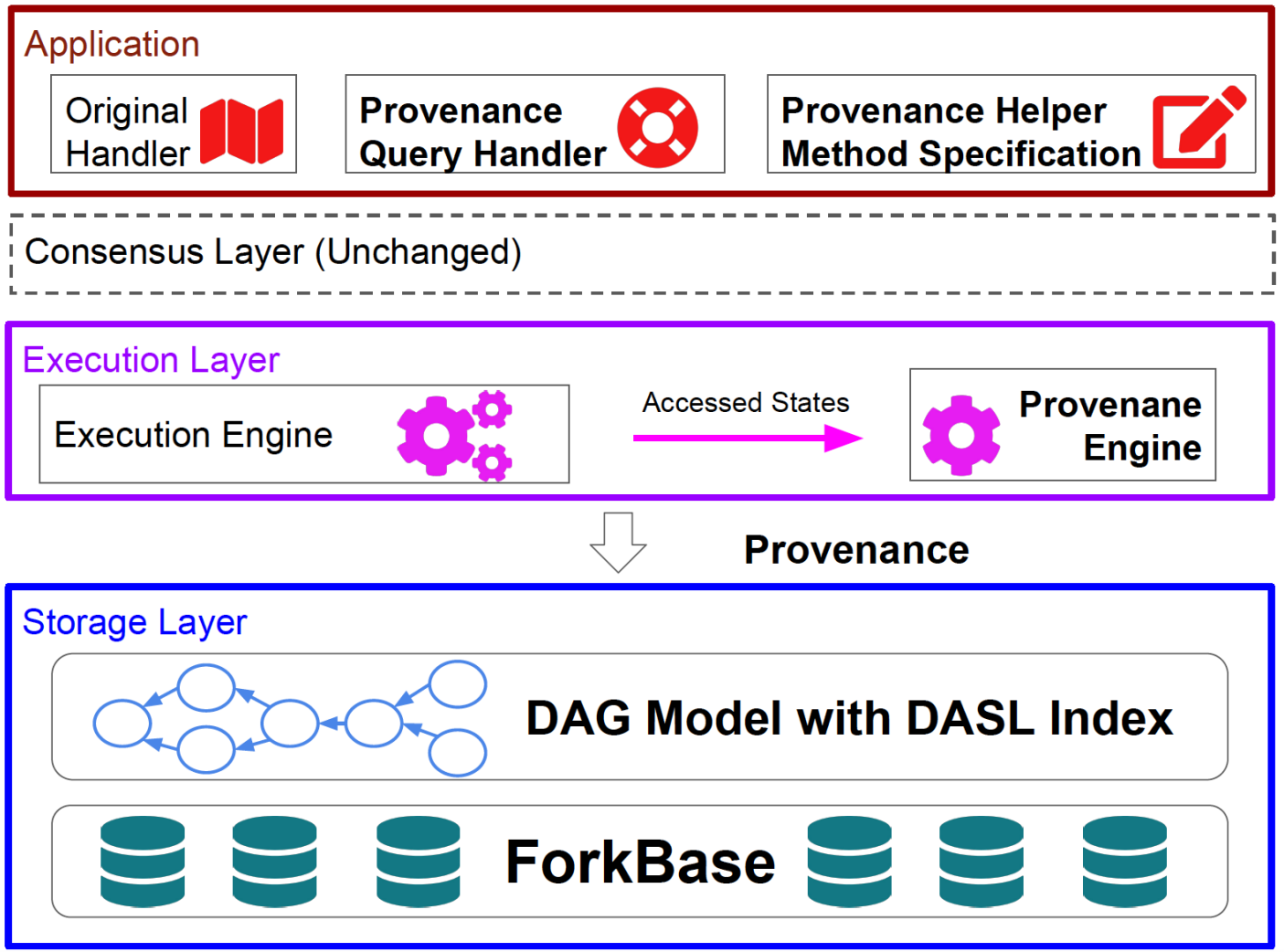


Enhancement Basis (Merkle Tree Variants)

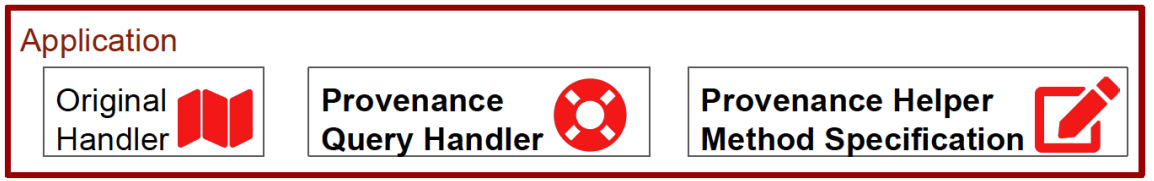
- **Limitation**
  - Latest State only
- **Tamper evidence**
  - Succinct digest (root hash)
  - Integrity proof (access path)



LineageChain Overview



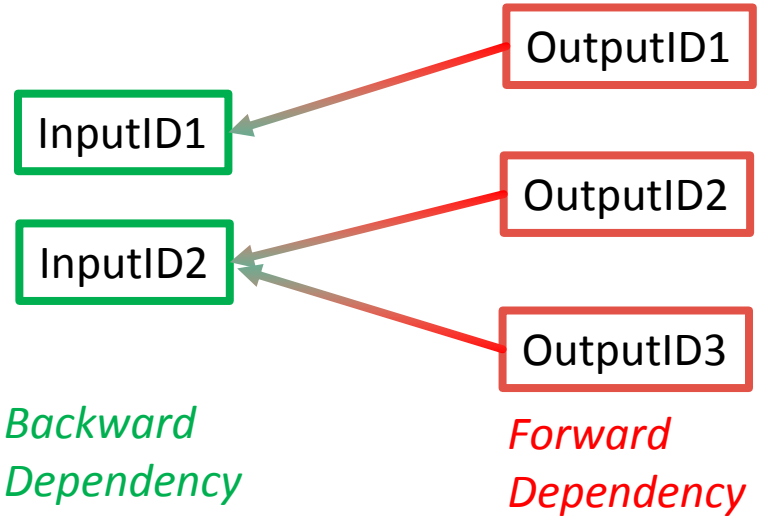
Application Layer



- **Provenance specification**
  - User-defined input-output dependency

```
method prov_helper(name: string,
                   reads: map(string, byte[]),
                   writes: map(string, byte[]))
            returns map(string, string[]);
```

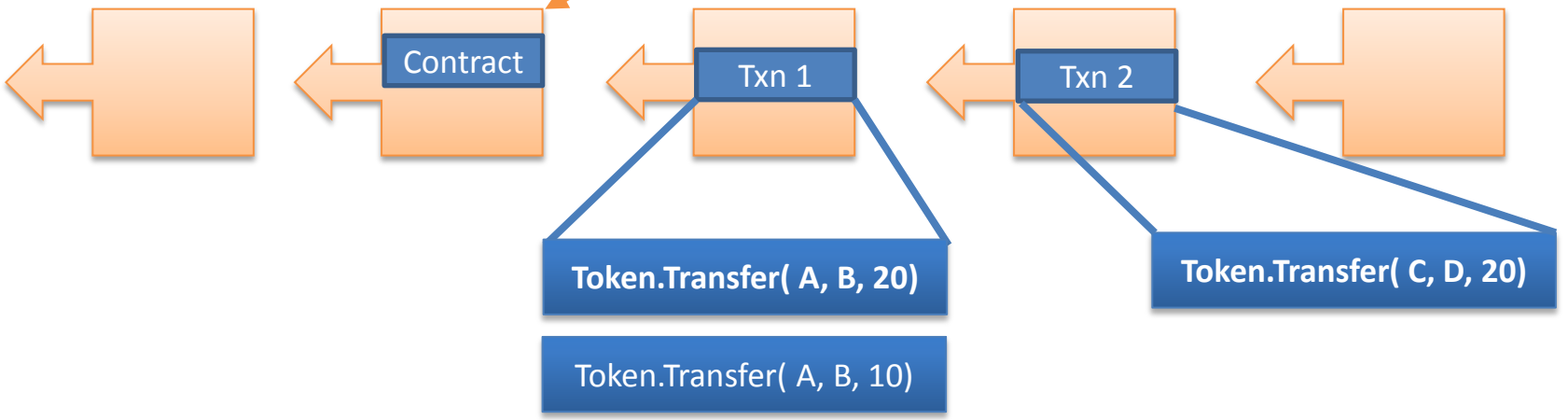
- **Provenance query handler**
  - `Hist(stateID, [blockNum])`
    → `(val, blkStart, txnID)`
  - `Backward(stateID, blkNum)`
    → `List<(depStateID, depBlkNum)>`
  - `Forward(stateID, blkNum)`
    → `List<(depStateID, depBlkNum)>`



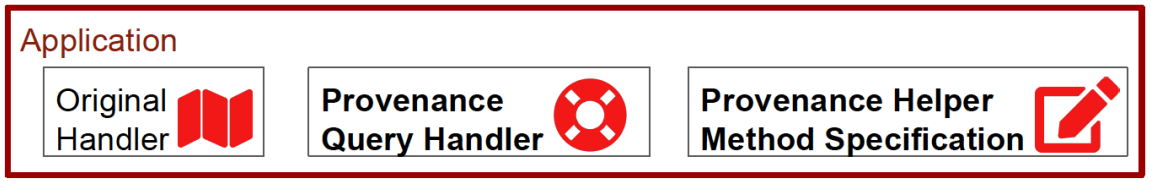
Blockchain Basics

- **P2P network**
  - Asynchronous transaction
- **Byzantine environment**
  - Mutual distrusting setup
- **Distributed ledger**
  - Smart contract
- **Inherent provenance-preserving**
  - ONLY for offline analytical query

```
contract Token {
    method Transfer(sender, recipient, amount) {
        bal1 = gState[sender];
        bal2 = gState[recipient];
        if (amount < bal1) {
            gState[sender] = bal1 - amount;
            gState[recipient] = bal2 + amount;
} } }
```



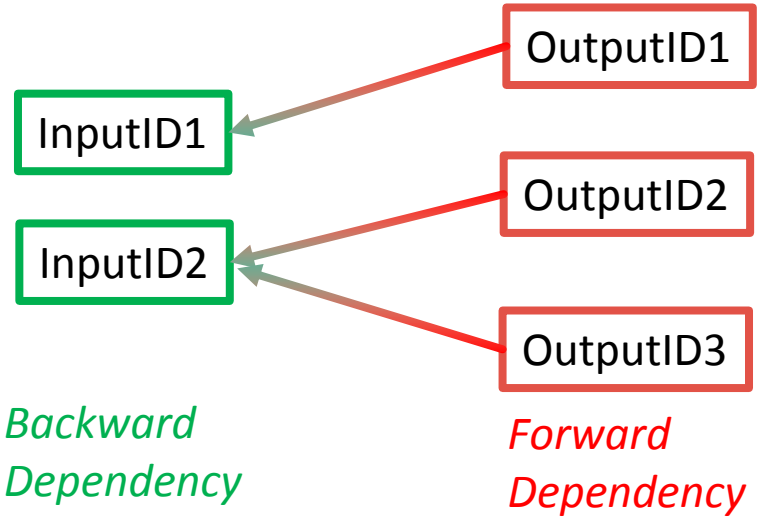
Application Layer



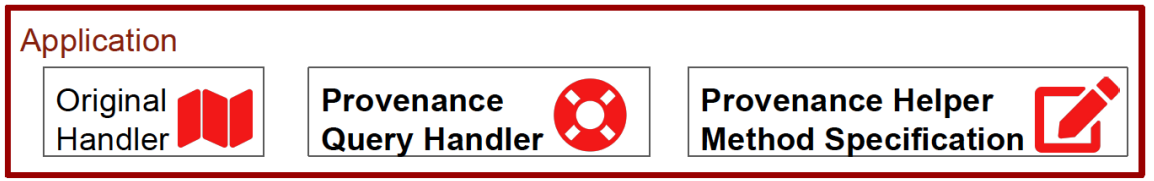
- **Provenance specification**
  - User-defined input-output dependency

```
method prov_helper(name: string,
                   reads: map(string, byte[]),
                   writes: map(string, byte[]))
            returns map(string, string[]);
```

- **Provenance query handler**
  - `Hist(stateID, [blockNum])`
    → `(val, blkStart, txnID)`
  - `Backward(stateID, blkNum)`
    → `List<(depStateID, depBlkNum)>`
  - `Forward(stateID, blkNum)`
    → `List<(depStateID, depBlkNum)>`



Application Layer



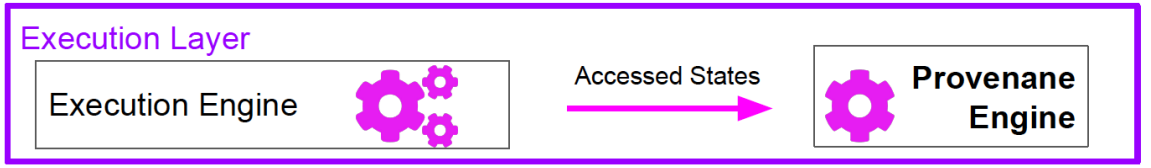

```
contract Token {
    method Transfer(...){...} // as above
    method prov_helper(name, reads, writes) {
      if name == "Transfer" {
        for (id,value) in writes {
            if (reads[id] < value) {
                recipient = id;
            } else {sender = id; }
        }
        // dependency list with a
        // single element.
        dep = [sender];
        return {recipient:dep};
      }
      ...
    }
}
```

```
method Refund(addr) {
  blk := last block in the ledger
  first_blk := first block in this month
  sum = count = 0;
  while (first_blk < blk) {
    val, startBlk, txnID = Hist(addr, blk);
    blk = startBlk - 1;
    sum += val;
    count += 1;
  }
  avg = sum / count;
  refund_amount := refund amount based on avg
  gState[addr] += refund_amount;
}
```

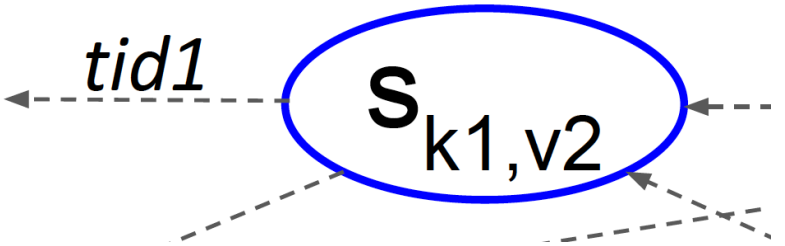
Recipient -> Sender

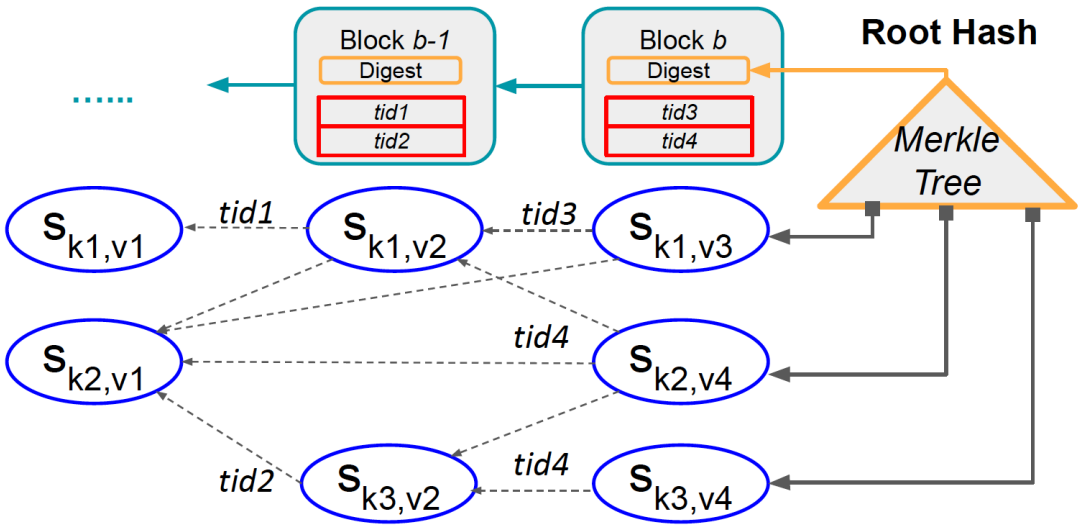
```
method Blacklist(addr) {
  blk := last block in the ledger
  blacklisted = false;
  iterate 5 times {
    val, startBlk, txnID = Hist(addr, blk);
    for (depAddr, depBlk)
          in (Backward(addr, startBlk)
           or Forward(addr, startBlk)) {
      if depAddr in gState["blacklist"] {
        gState["blacklist"].append(addr);
        return;
      }
    }
    blk = startBlk - 1;
  }
}
```

Execution Layer

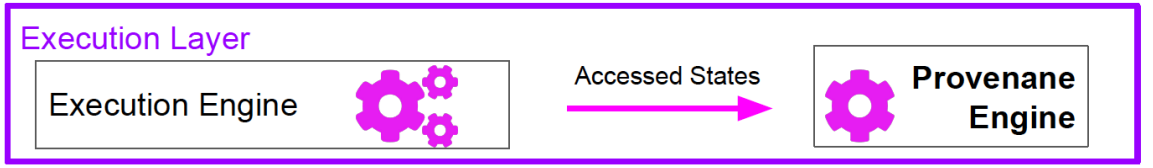


- **Receive**
  - Contract invocation context
  - Provenance specification
- **Compute**
  - Transaction results
  - Concrete dependency
- **Prepare Merkle DAG**
  - Introduce one layer of direction
  - Hash reference to encode provenance backward dependency

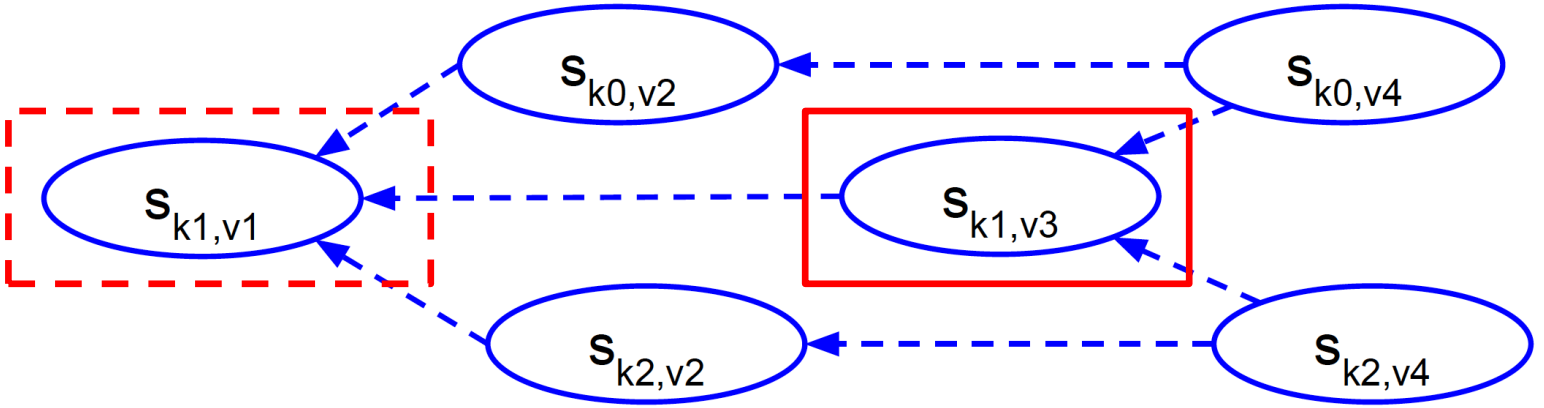




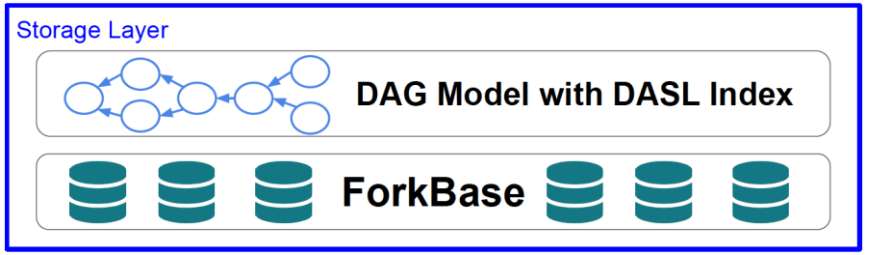
Execution Layer



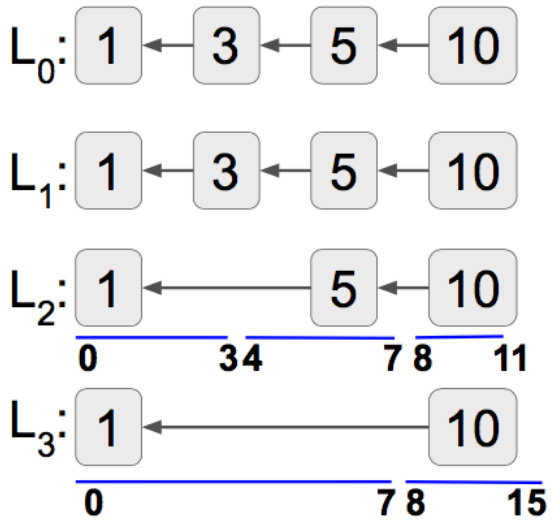
- **Forward tracking**
  - Problem: Undecided forward dependency during state update
- **Solution**
  - Lazily store forward dependency on the successor state entry



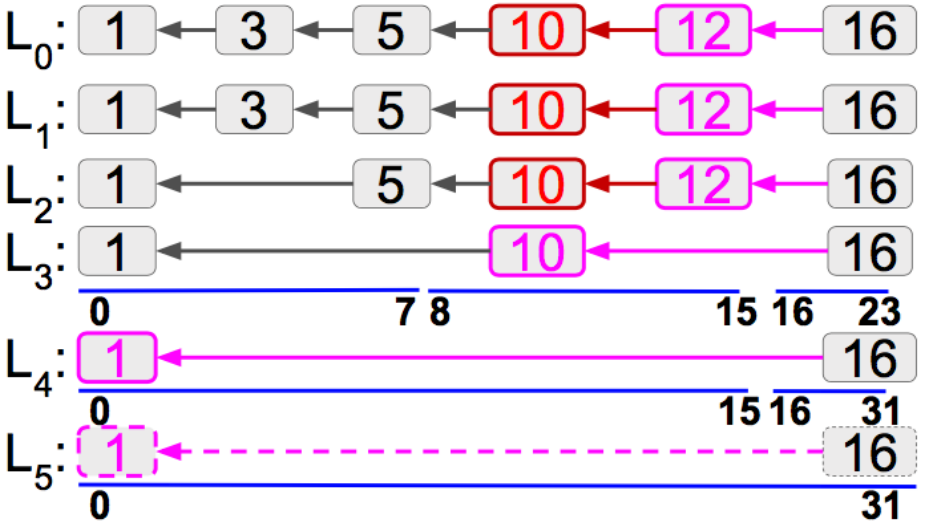
Storage Layer



- **Problem**
  - Efficient version-based (historical) query for a state ID
- **Solution:**
  - **D**eterministic **A**ppend-only **S**kip **L**ist
  - Hash-based reference



After appending versions 12 and 16



DEFINITION 6. *Let* $V_k = \langle v_0, v_1, ... \rangle$ *be the sequence of version numbers of states with identifier* k*, in which* $v_i < v_j$ *for all* $i < j$*. A DASL index for* k *consists of* N *linked lists* $L_0, L_1, .., L_{N-1}$*. Let* v_j^{i-1} *and* v_j^i *be the versions in the* $(j-1)^{th}$ *and* j^{th} *node of list* L_i*. Let* b *be the base number, a system-wide parameter. The content of* L_i *is constructed as follows:*

1) $v_0 \in L_i$

2) Given v_{j-1}^i*,* v_j^i *is the smallest version in* V_k *such that:*

$$\left\lfloor \frac{v_{j-1}^i}{b^i} \right\rfloor < \left\lfloor \frac{v_j^i}{b^i} \right\rfloor \tag{5}$$

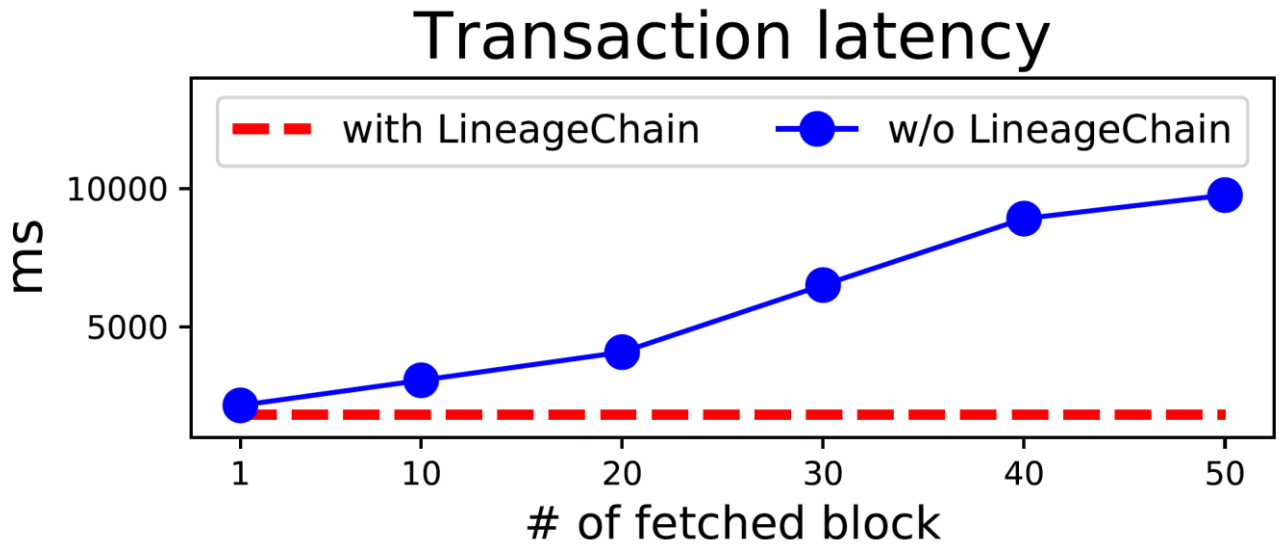
Evaluation

- **MICRO benchmarking** (vs. flat storage)
  - Preference to recent version query (with DASL)
  - More efficient BFS enabled by backtrack (with ForkBase)

- **MACRO benchmarking** (applied to Hyperledger Fabric v0.6 and v1.3)
  - Negligible runtime overhead
    - Tiny proportion of latency
  - Negligible storage overhead
    - >70% of space for blocks
    - 25% for historical states
    - 2~4% for DASL indexes and hash pointers

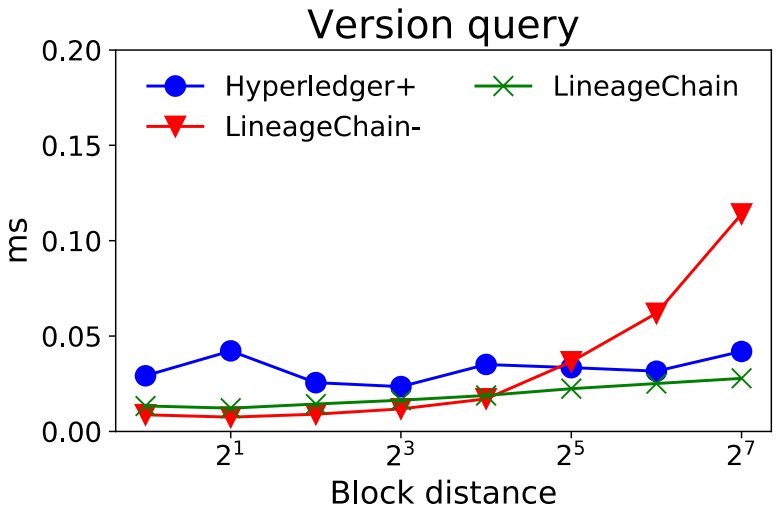
Performance of Provenance Query

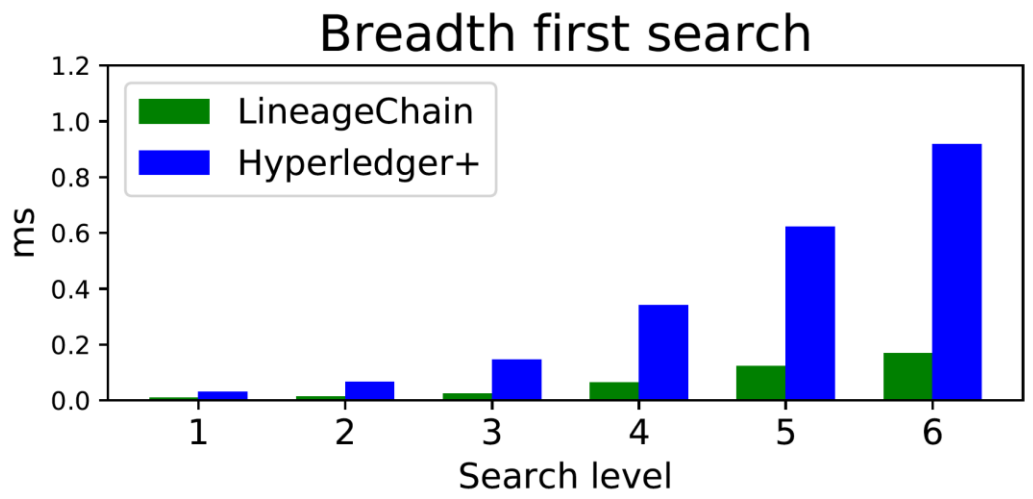
- **vs. Workaround 2**
  - Compute data provenance offline and conditionally trigger online transaction

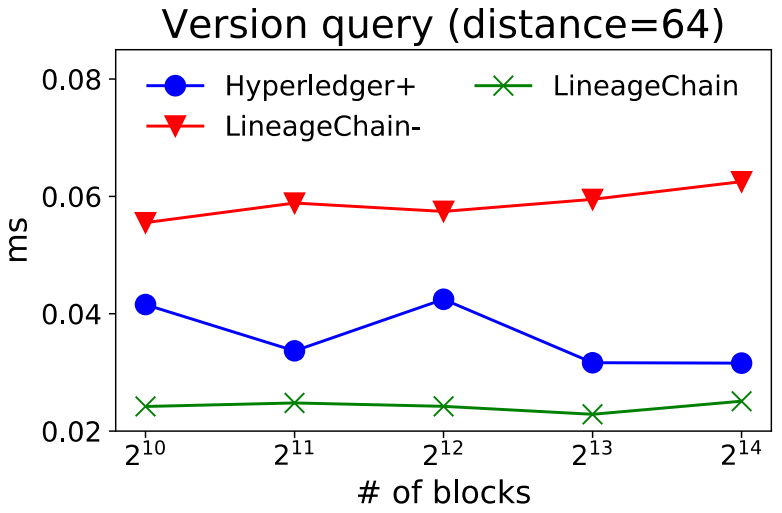


Micro Performance of Provenance Query

- **vs. Workaround 1**
  - Dump everything into the current state
- **vs. Workaround 3**
  - Use Hyperledger Fabric's built-in HistoryDB

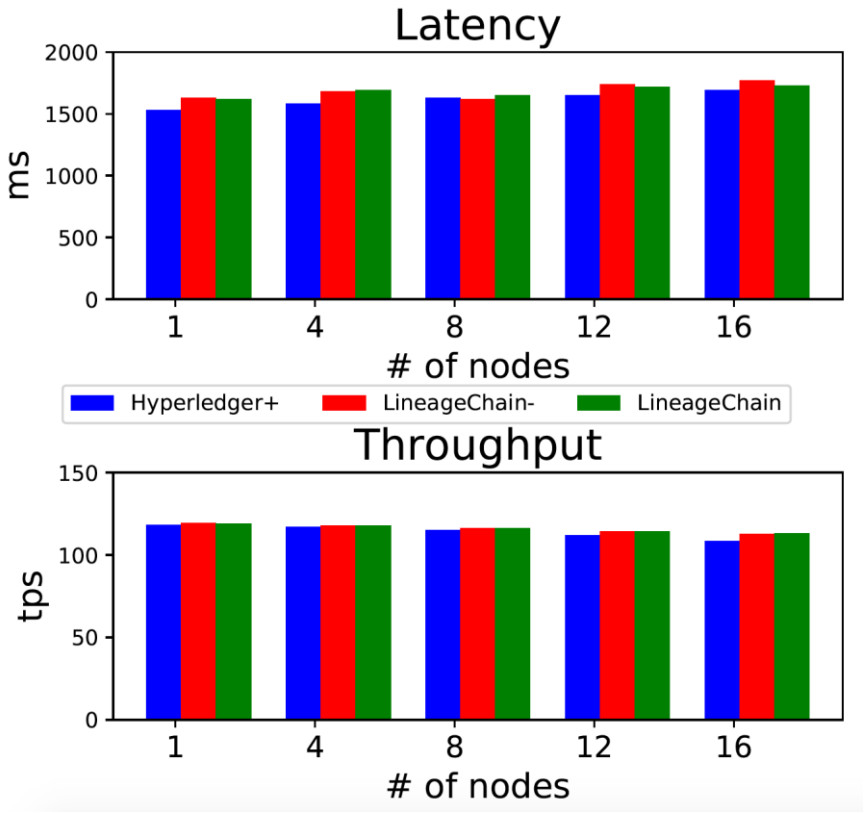




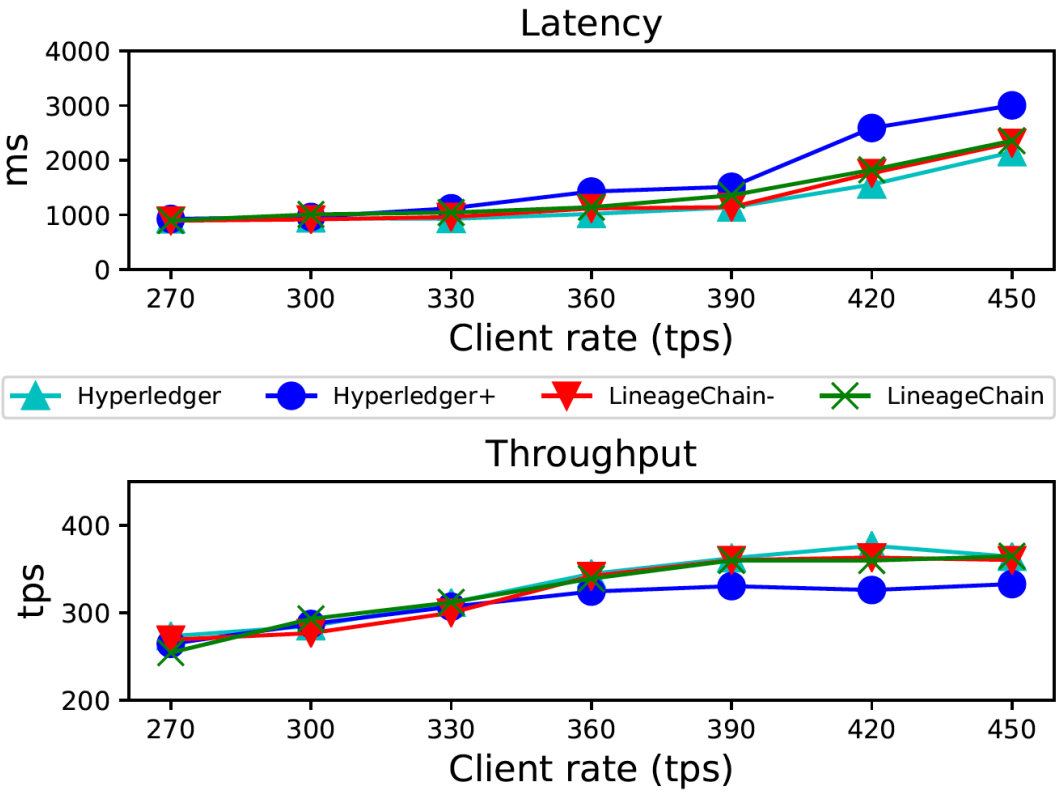


Runtime Overhead

- **Transaction processing**

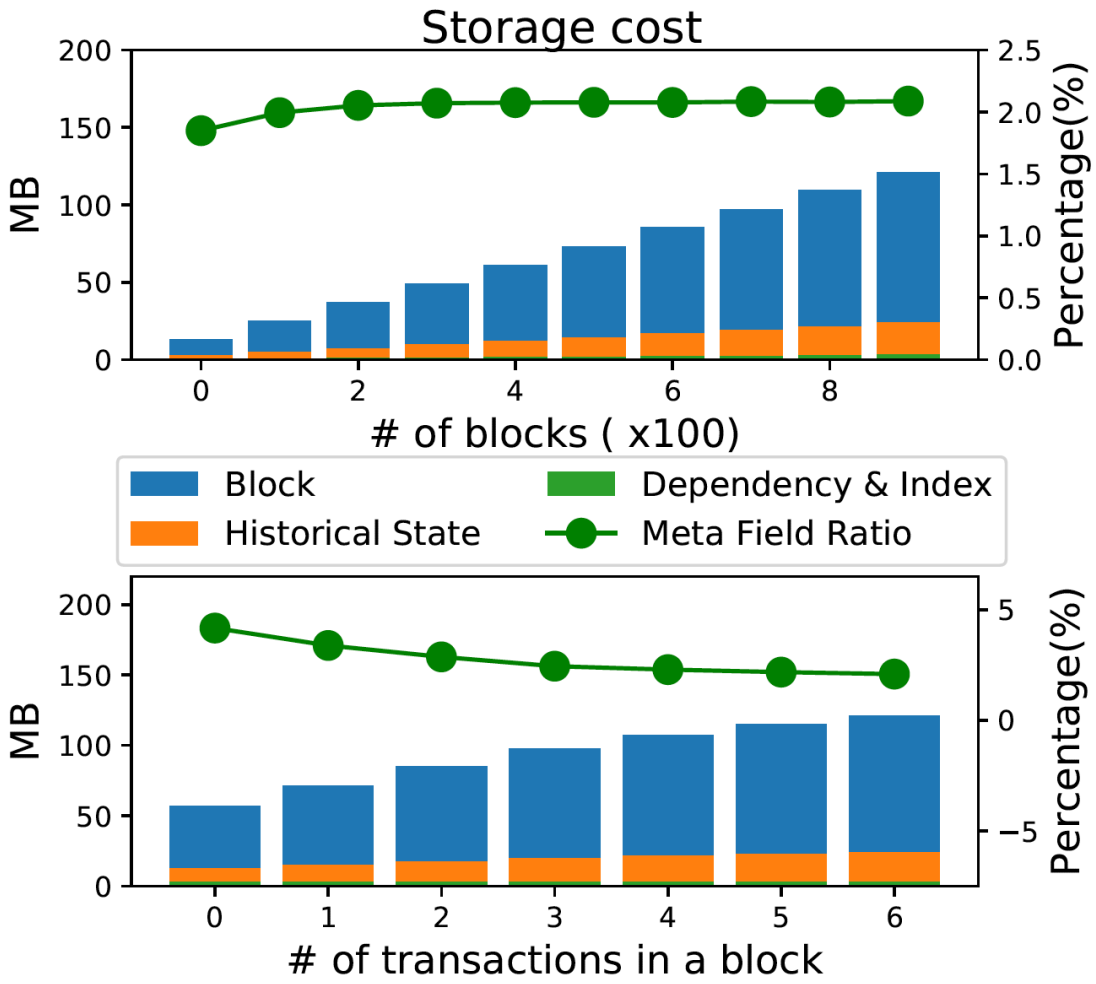


Hyperledger Fabric v0.6



Hyperledger Fabric v1.3

Storage Overhead



Conclusion

- **LineageChain**
  - Enabler for provenance-dependent blockchain applications
  - Protocol-level enhancement w.r.t. efficiency and security
  - Negligible performance and storage overhead
- **Key designs**
  - User-defined dependency specification
  - Merkle DAG with dependency tracking
  - DASL index to accelerate data provenance query
  - Adoption in Hyperledger Fabric (v0.6 & v1.3)

Thank You!