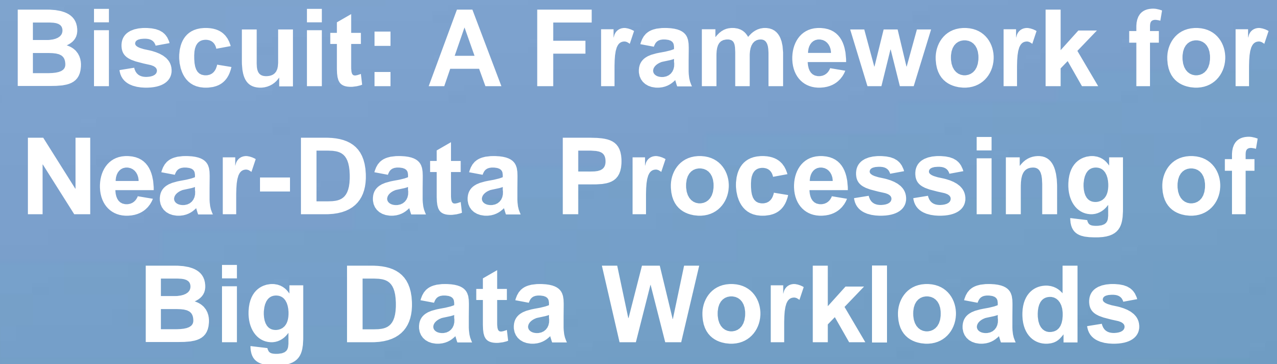


Oct 21, 2016

Duck-Ho Bae

Memory Business, Samsung Electronics

Outline

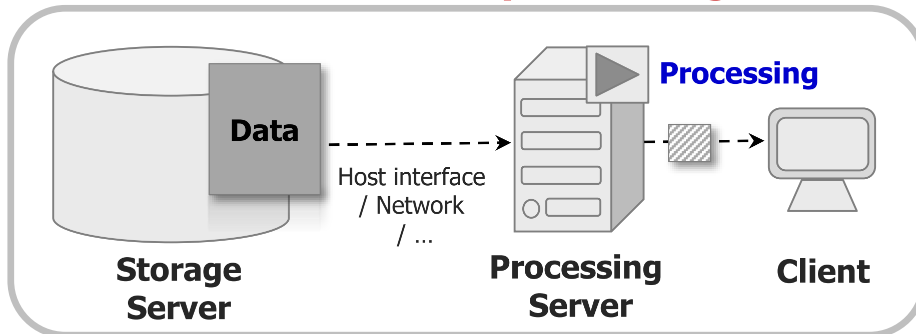
- Biscuit: A Framework for Near-Data Processing of Big Data Workloads, ISCA16
- YourSQL: A High-Performance Database System Leveraging In-Storage Computing, VLDB16

Near-Data Processing (NDP)

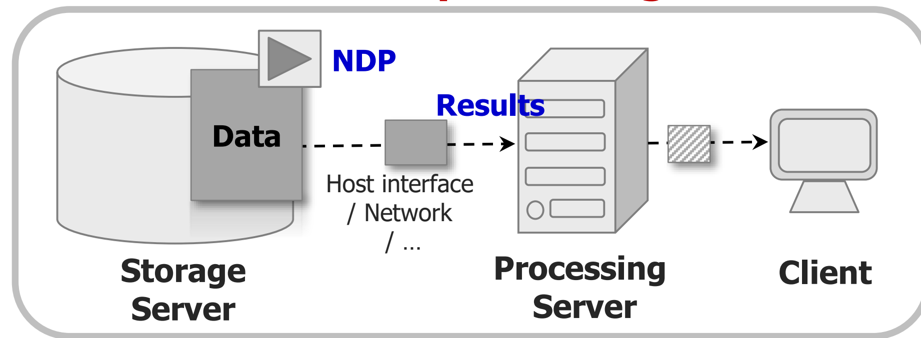
- "Moving Computation is Cheaper than Moving Data"

* HDFS Architecture Guide

Traditional data processing



Near-data processing

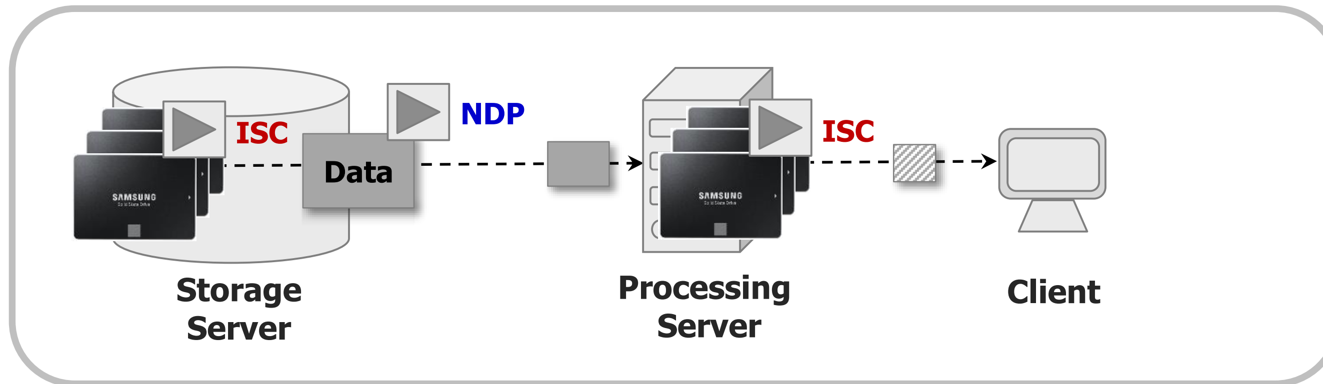


- Near-data processing moves computation to data
  - Computation is performed right at the data source
  - Efficient when the cost of moving data is very high

In-Storage Computing (ISC)

- The ultimate of near-data processing is "In-Storage Computing"

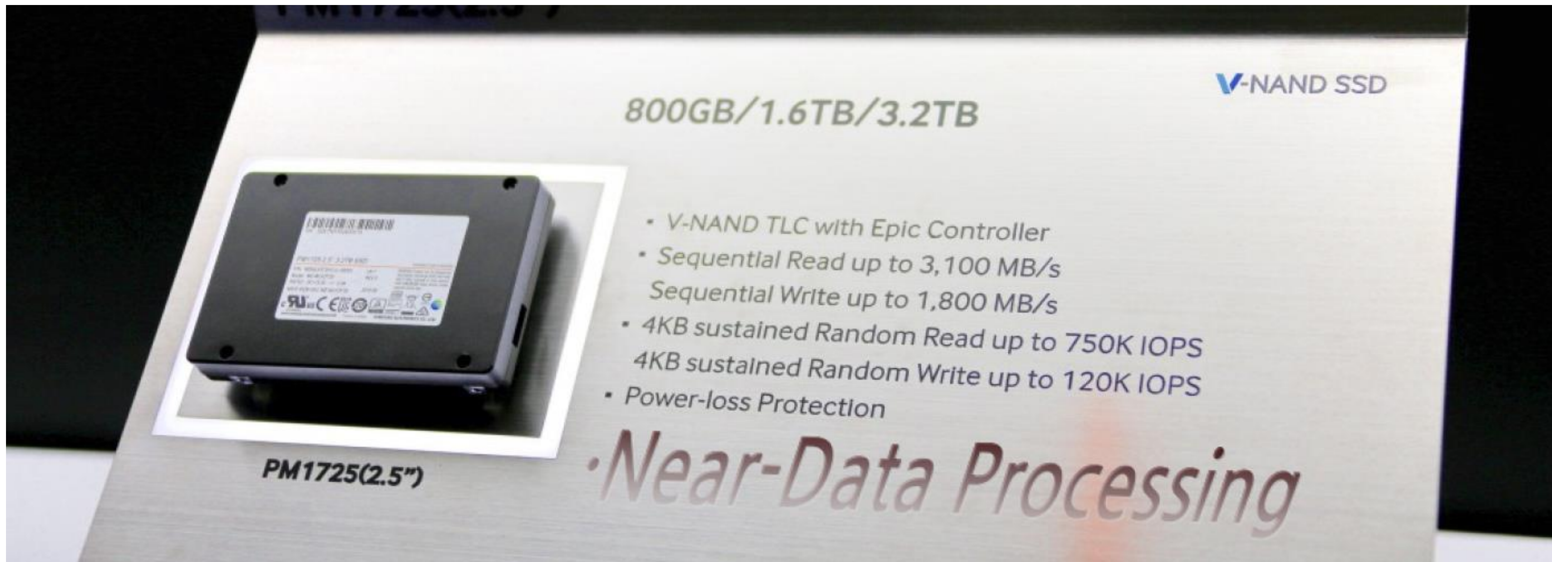
NDP with ISC



- Most prior work focuses on proving the concept of ISC
  - Little attention to designing and realizing a practical framework
  - Realistic large application studies were omitted

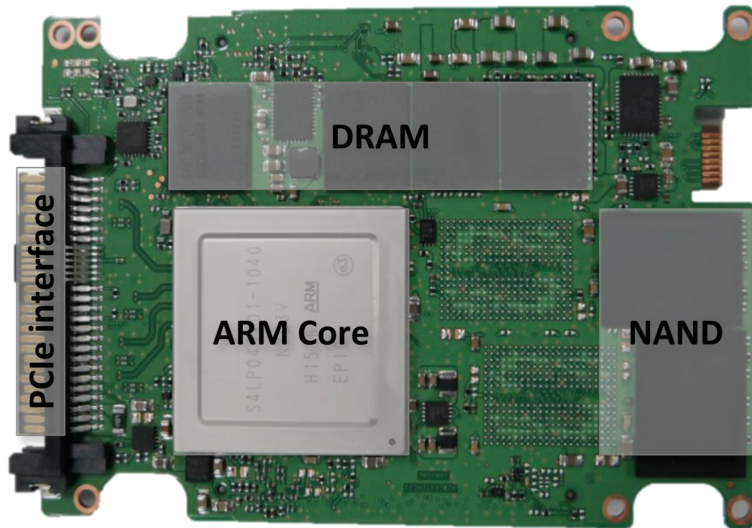
Samsung NVMe SSD (PM1725)

Biscuit



- A user-programmable NDP framework for SSDs and data-intensive applications
  - The first reported product-strength NDP system
  - Modern C++ support (including C++ standard library)
  - Dynamic loading of user programs
  - Multi-threading, multi-core support

SSD Hardware



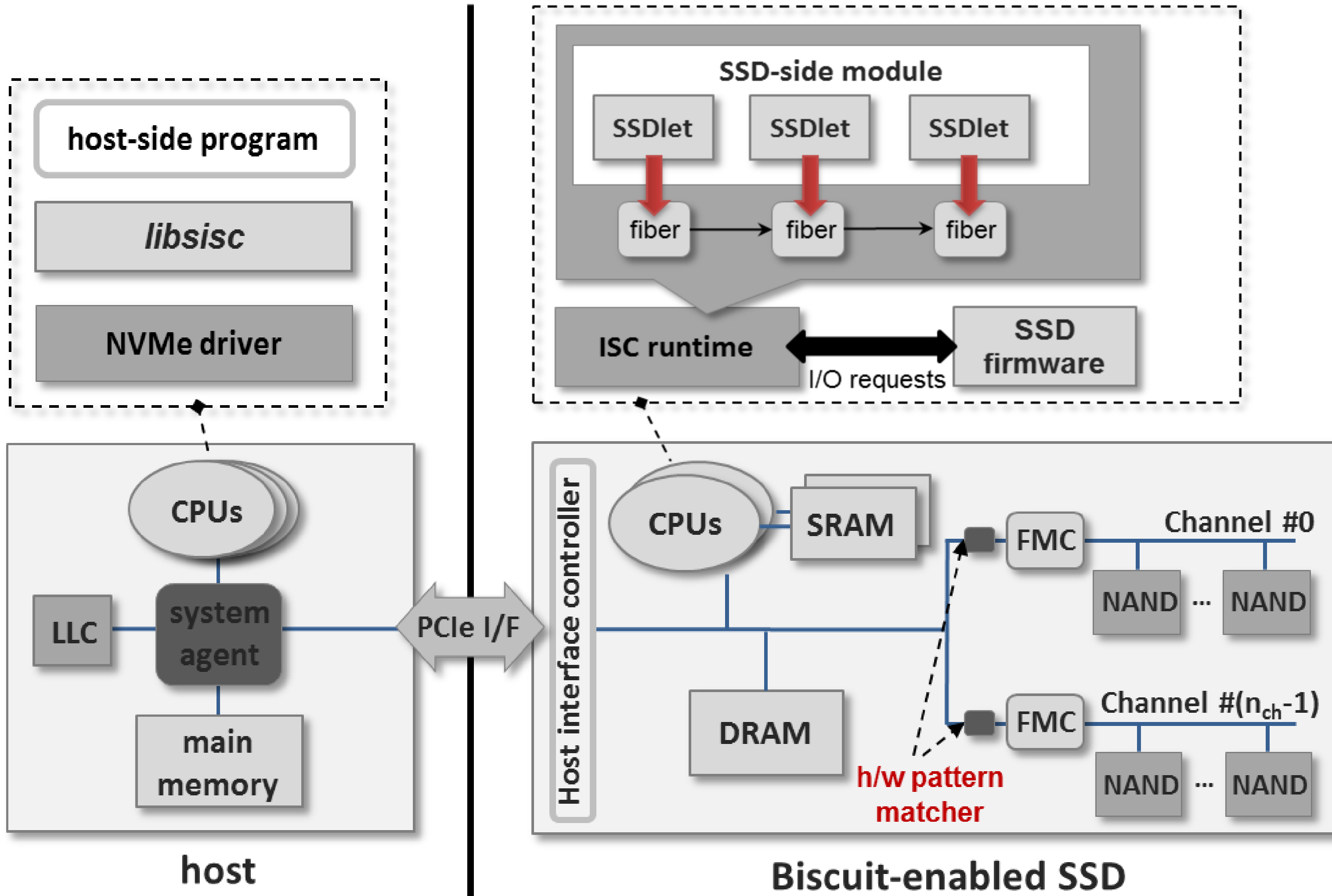
Item	Description
Host interface	PCIe Gen.3 x4 (3.2GB/s)
Protocol	NVMe 1.1
Device density	1 TB
SSD architecture	Multiple channels/ways/cores
Storage medium	Multi-bit NAND flash memory
Compute resource for Biscuit	Two ARM Cortex R7 cores @ 750MHz with L1 cache
On-chip SRAM	< 1 MiB
DRAM	≥ 1 GiB
Hardware IP	Key-based pattern matcher per channel

- Limitations
  - Low compute power, no cache coherence, a small amount of fast memory, no MMU, and restrictive synchronization primitives

Biscuit Runtime

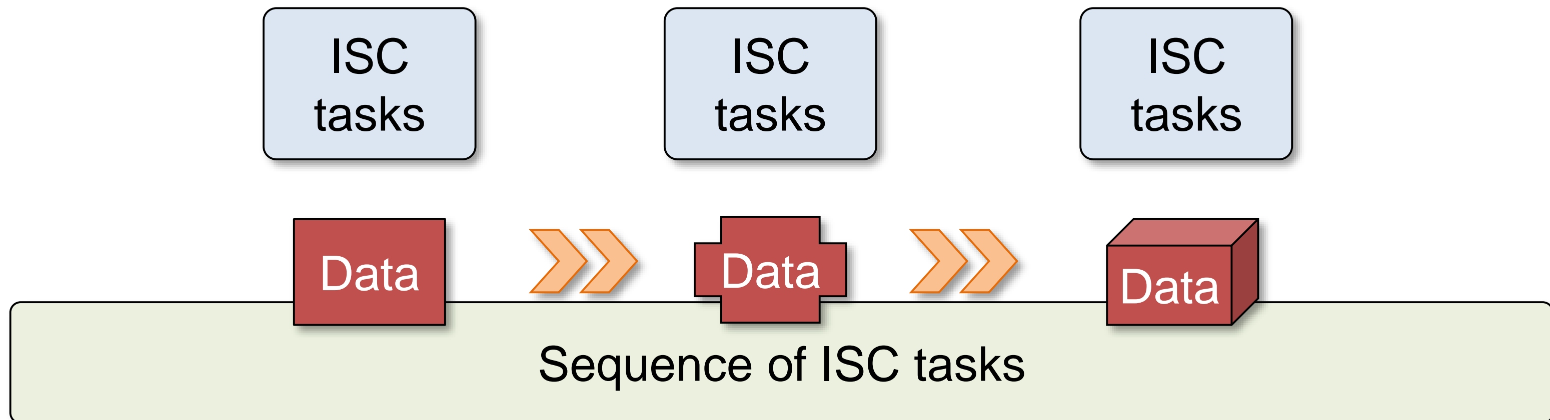
- Cooperative multi-threading
  - A limited form of multi-threading (fiber as a scheduling unit)
  - Less context switching overhead
  - Safe resource sharing without locking
- Shared nothing architecture
  - All data transmission among threads through I/O ports
  - Enforced by the programming model and APIs
  - C++11 move semantics supported
- Dynamic loader for user programs
  - User program as position-independent code (PIC)
  - Symbol relocation to locate each program in a separate address space

Biscuit System Architecture

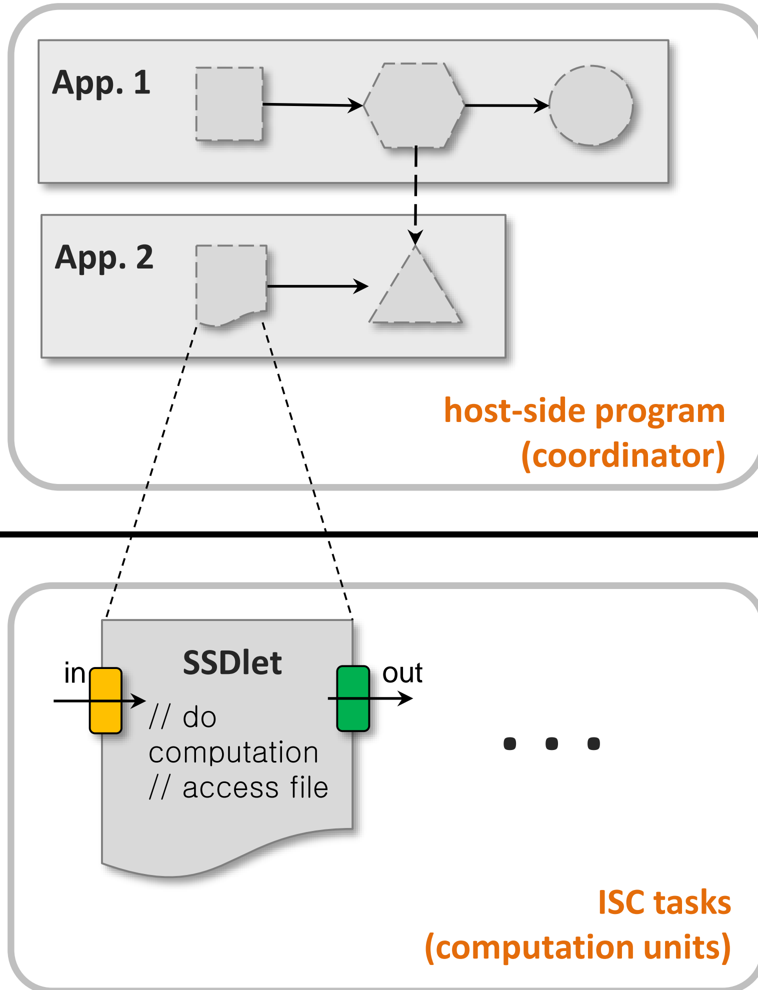


Biscuit Programming Model

- Biscuit follows a data-flow model
  - The data movement through ISC tasks determines their order of execution
  - On receiving all required inputs, an ISC task produces output and passes it to the next ISC tasks in the data-flow path



Biscuit Programming Model



- An ISC task is a unit of task that would run on an ISC-enabled SSD
- A host-side program creates and manages ISC tasks
- Both run concurrently in the ISC-enabled SSD and the host, respectively

Development Process

1 Write codes

Host-side task

```
int main(int argc, char *argv[])
{
    SSD ssd("/dev/nvme0n1p1");
    module_id_t mid = ssd.loadModule(File(ssd, "./libkvstore.so");
    Application app(ssd);

    SSDLet kvstore(app, mid, "KVStore");
    auto out_command = app.connectTo<String>(kvstore.in(0));
    auto out_key = app.connectTo<String>(kvstore.in(1));
    auto out_value= app.connectTo<String>(kvstore.in(2));
    auto in_result = app.connectTo<String>(kvstore.out(0));
    app.start();

    string command, key, value;
    while (std::cin >> command) {
        if (command == "get") {
            out_command.put(command);
            std::cin >> key;
            out_key.put(key);
            in_result.get(value);
            std::cout << value << std::endl;
        else if (command == "put") {
            out_command.put(command);
            std::cin >> key >> value;
            out_key.put(key);
            out_value.put(value);
        }
        else break;
    }
    return 0;
}
```

SSD-side task

```
class KVStore
    public SSDLet<IN_TYPE<SR(string), SR(string), SR(string)>,
                  OUT_TYPE<SR(string)>>
{
    public:
        map<string, string> table;

        void run()
        {
            auto in_command = getInputPort<0>();
            auto in_key = getInputPort<1>();
            auto in_value = getInputPort<2>();
            auto out_value = getOutputPort<0>();

            string command, key, value;
            while (true) {
                if (!in_command.get(command))
                    break;

                if (command == "get") {
                    if (!in_key.get(key))
                        break;
                    out_value.put(table[key]);
                }
                if (command == "put") {
                    if (!in_key.get(key) || !in_value.get(value))
                        break;
                    table[key] = value;
                }
            }
        }
};
```

2 X86 Compile

Host-side program

3 ARM Cross compile

SSD-side module

4 Copy the module into Biscuit SSD

5 Run host-side program

Host Computer

ISC

Experimental Setup

- H/W setup

System	Dell PowerEdge R720 server
CPU	2 Intel Xeon(R) CPU E5-2640 (12 threads per socket) @2.50GHz
Memory	64 GiB DRAM
OS	64-bit Ubuntu 15.04

- Basic performance results
  - Communication latency, data read latency, data read bandwidth
- Application level results
  - String search, pointer chasing, DB scan/filtering, TPC-H
- Notations
  - Conv: system configuration with a default conventional SSD
  - Biscuit: system configuration with the Biscuit framework on the SSD

Basic Performance Results – Data Read Latency

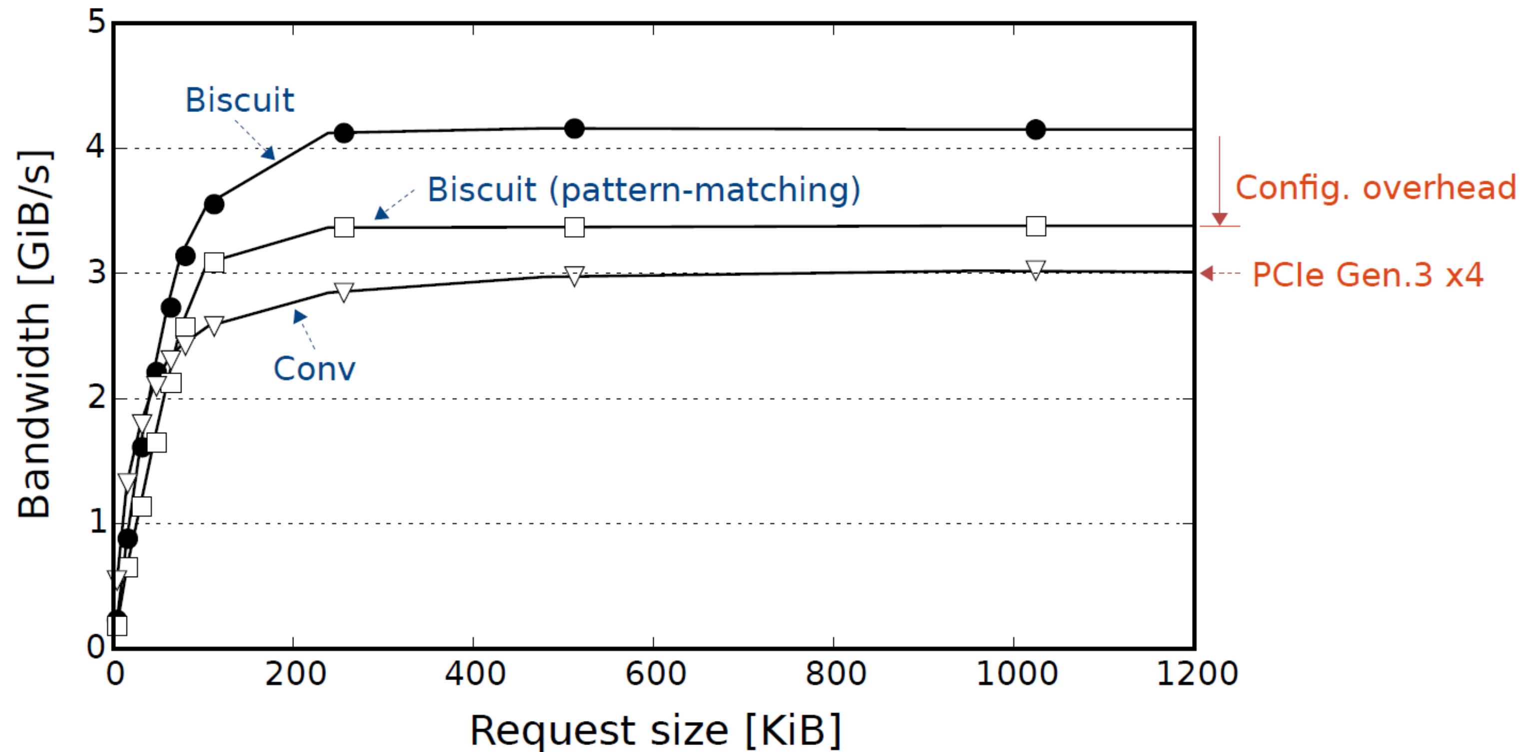
- Conv: Linux pread I/O primitive
- Biscuit: internal data read API

	Conv	Biscuit
Read Latency (us) – 4KiB	90.0	75.9

- Biscuit shows 18% shorter latency
- Biscuit has the shorter round-trip "path" — No data transmission from the device to the host over a host interface

Basic Performance Results – Data Read Bandwidth

- **Conv**: transfer data to the host-side program
- **Biscuit**: transfer data to the SSD-side module (i.e., internal read)



- Biscuit exploits the underutilized internal bandwidth

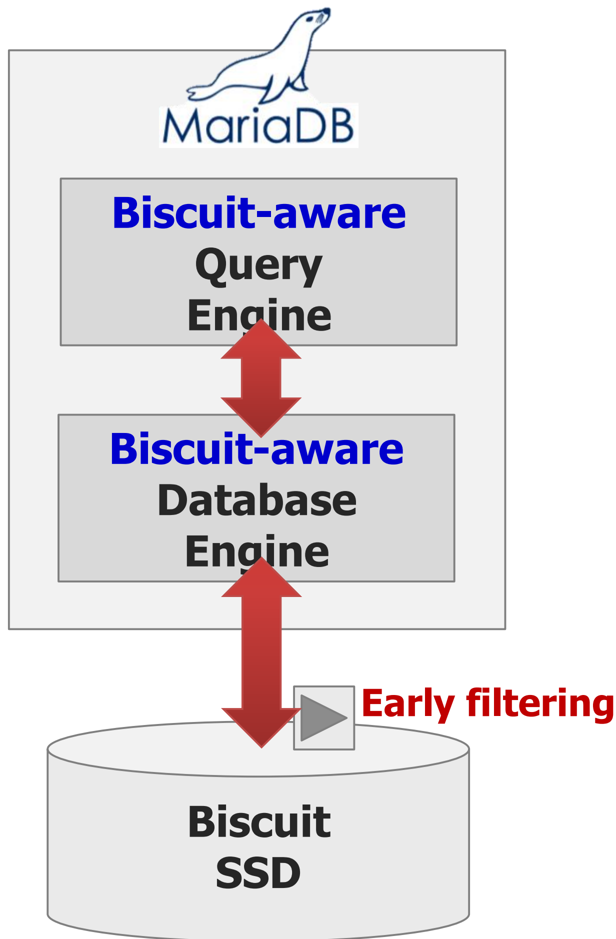
Application Level Results – Pointer Chasing

- Conv: round-trip operation between host and SSD
- Biscuit: perform data-dependent logic entirely within SSD

	Conv	Biscuit
Execution time (s) – 20GiB Twitter data – 100 starting nodes	138.6	124.4

- Biscuit achieves 11% performance gain
- This gain is comparable to the improvement in read latency with Biscuit

Application Level Results – DB Scan and Filtering

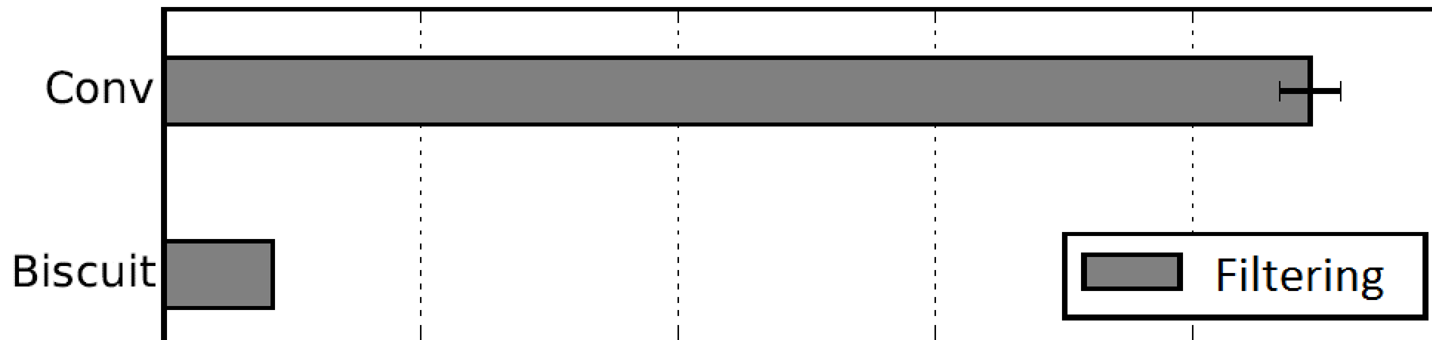


- Data analytics with a real DB engine
  - MariaDB 5.5.42 (XtraDB)
  - We modified the query engine to

    1. identify a candidate table amenable for offloading
    2. estimate its selectivity using a sampling method
    3. determine whether the table is indeed a good target (based on a selectivity threshold)
    4. and finally offload the identified filter to the SSD

Application Level Results – DB Scan and Filtering

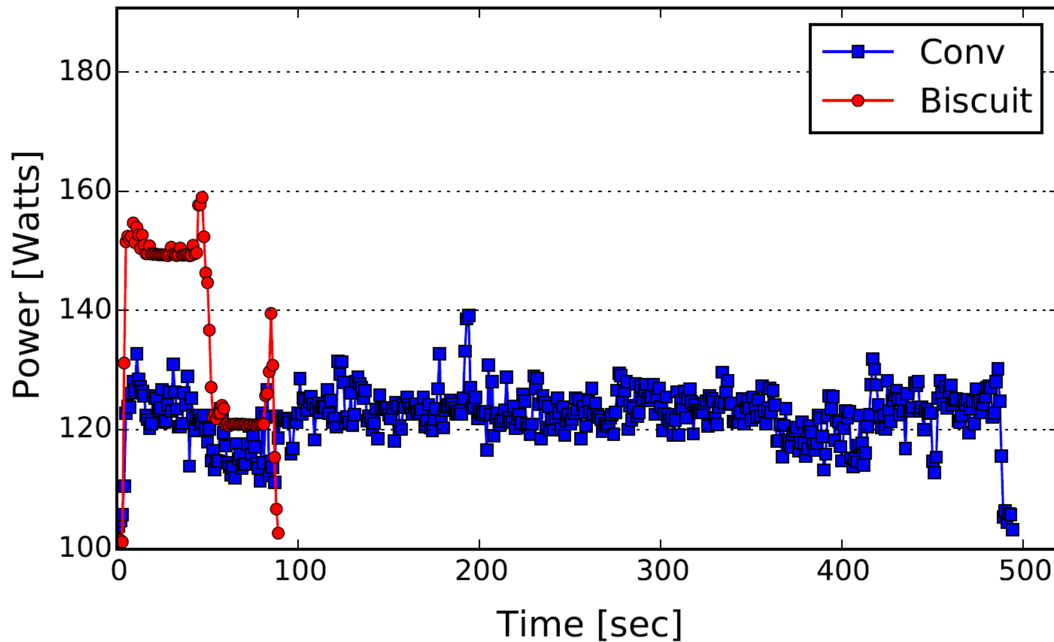
Filtering Query
SELECT l_orderkey, l_shipdate, l_linenumber
FROM lineitem
WHERE l_shipdate = '1995-1-17'



- Biscuit achieves speed-ups of about 11x
- Execution times on Biscuit were very consistent

Application Level Results – Power Consumption

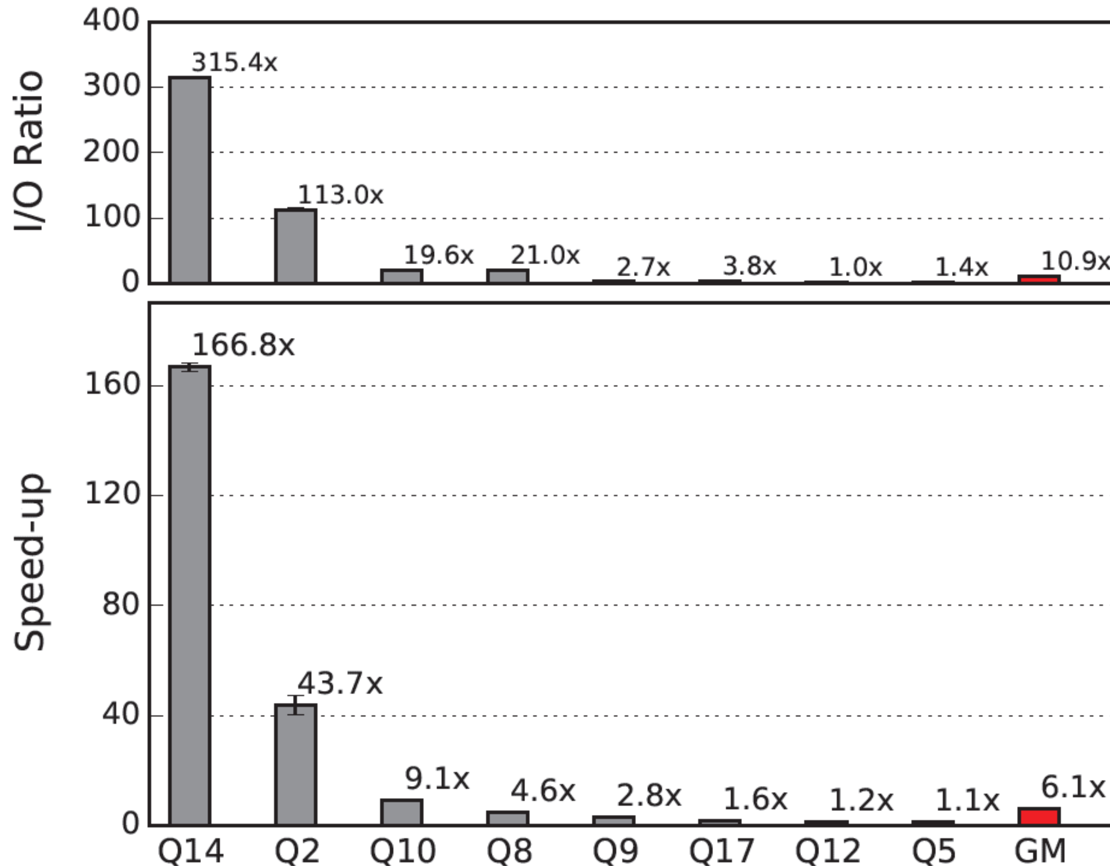
- **Filtering Query**



	Conv	Biscuit
Total Energy (kJ)	60.5	12.2

- Biscuit consumes more power during query processing
- Biscuit achieves significantly lower energy consumption thanks to its reduced execution time

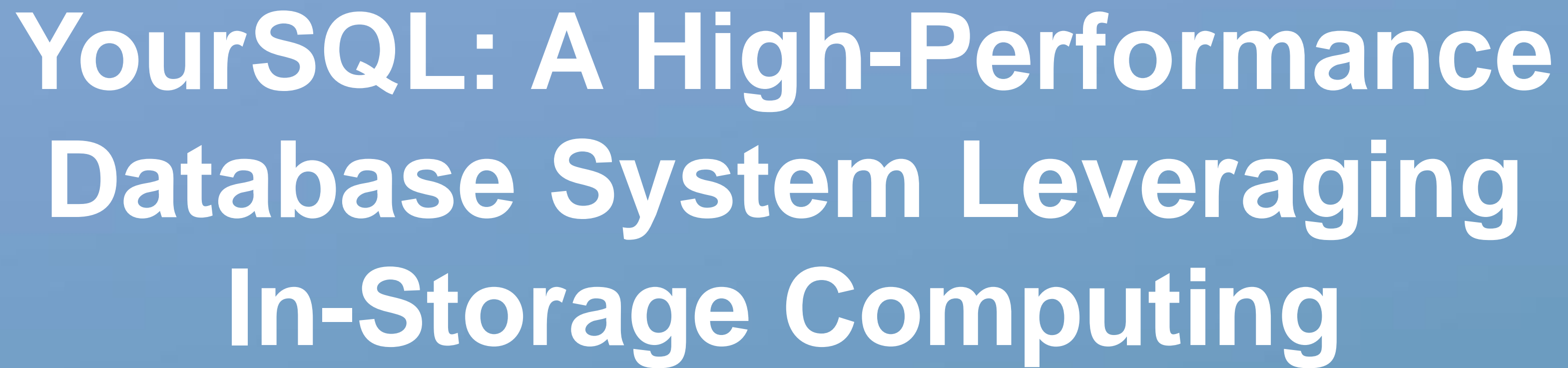
Application Level Results – TPC-H Results



- Running all queries, Conv takes nearly two days, while Biscuit takes about 13 hours (3.6x speed-up)
  - Top 5 queries take 70+% of total execution time

Conclusions

- We presented the design and implementation of Biscuit, an NDP framework built for high-speed SSDs.
- With Biscuit, we pursued achieving high programmability on distributed resources including processing units of SSDs as well as host CPUs.
- Biscuit is the first reported product-strength NDP system implementation.
- We successfully ported Biscuit on small and large data-intensive applications including MariaDB.
- Biscuit accomplished the performance improvement of up to 166x for TPC-H queries (average 6.1x improvement).


YourSQL: A High-Performance
Database System Leveraging
In-Storage Computing

YourSQL – ISC-enabled Database System

- Realizes very early-filtering of data by offloading data scanning of a query to ISC-enabled SSDs

- Why early-filtering?
  - Early-filtering is data-intensive, non-complex query operations
  - I/O reduction from the optimized join order and irrelevant data elimination is dramatic!

Join order	Table name	Access method	# of read requests
1	Region	All	16
2	Nation	Ref	13
3	Supplier	Ref	36,867
4	Partsupp	Ref	2,842,639
5	Part	Eq_ref	651,525
Total			**3,531,060**

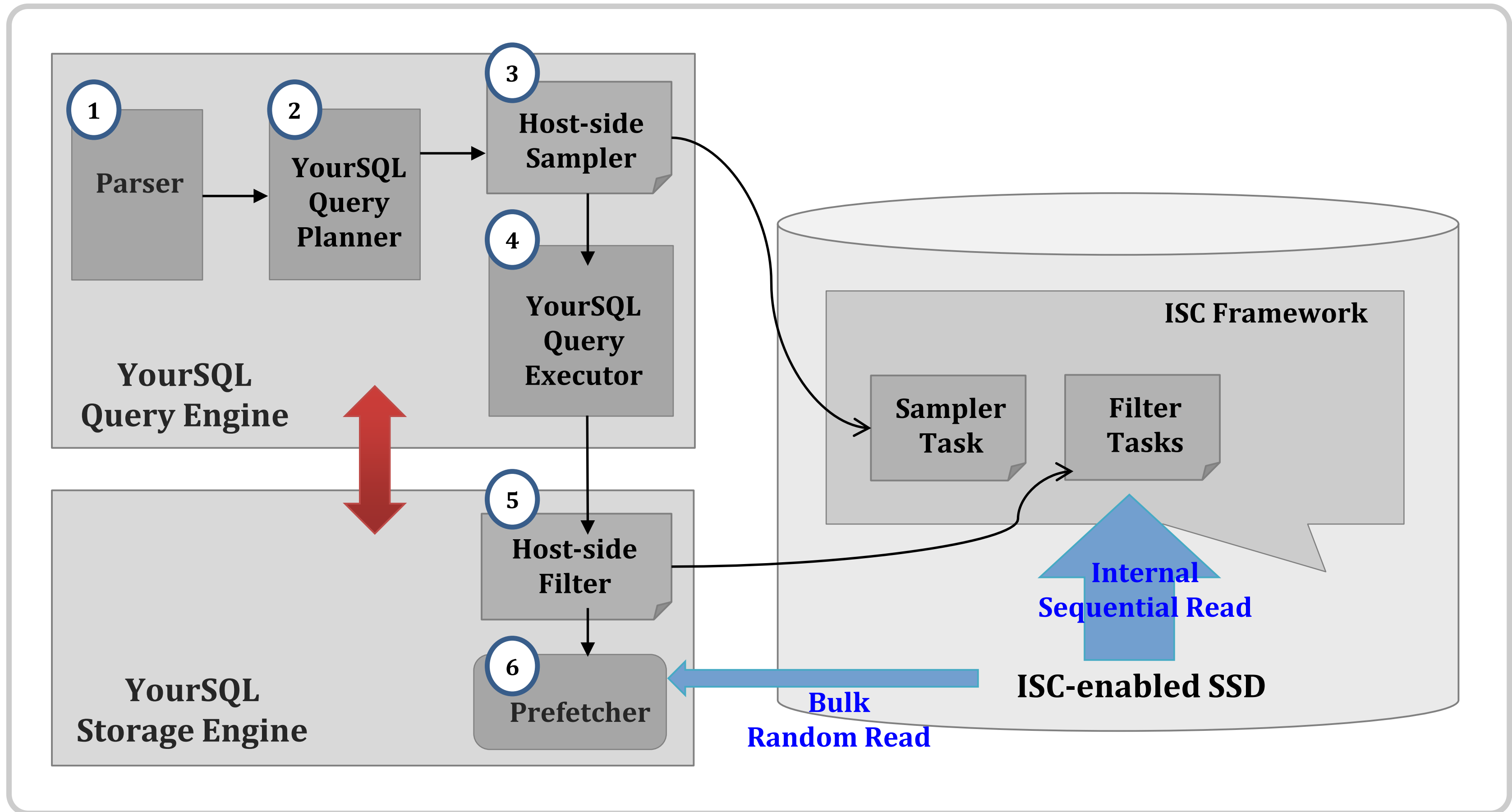
(a) MySQL w/o ICP

Join order	Table name	Access method	# of read requests
1	Part	Ref	245
2	Partsupp	Ref	98,520
3	Supplier	Eq_ref	45,679
4	Nation	Eq_ref	5
5	Region	All	4
Total			**144,453**

(b) MySQL w/ ICP

* TPC-H Q.2 on TPC-H dataset with a scale factor of 100

YourSQL Architecture

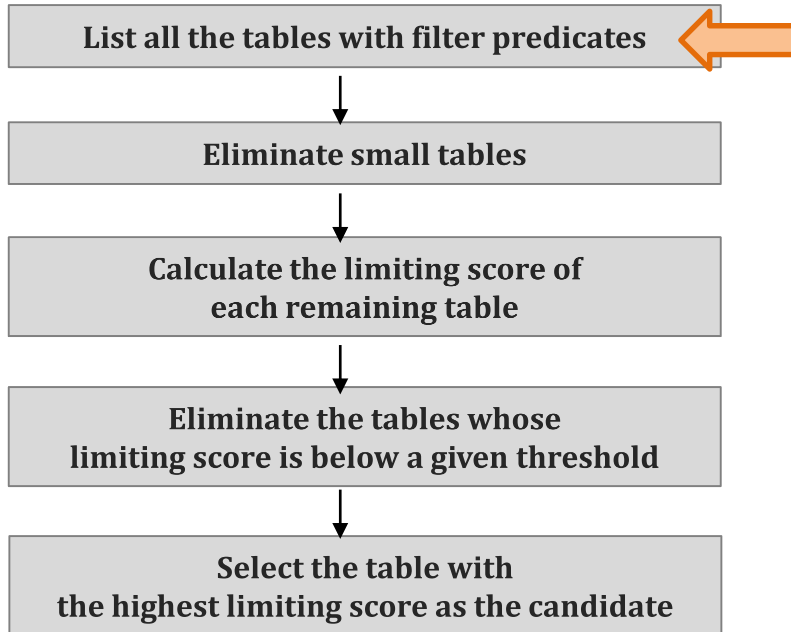


YourSQL Query Engine – Join Order Optimization

- Early-filtering target table is placed first in the join order
  - YourSQL assigns a limiting score for each filter predicate, which represents how restrictive its filter predicates are
  - The table with the highest limiting score is determined as the early filtering target

- For the remaining join order, it follows MySQL's decision

YourSQL Query Engine – Join Order Optimization

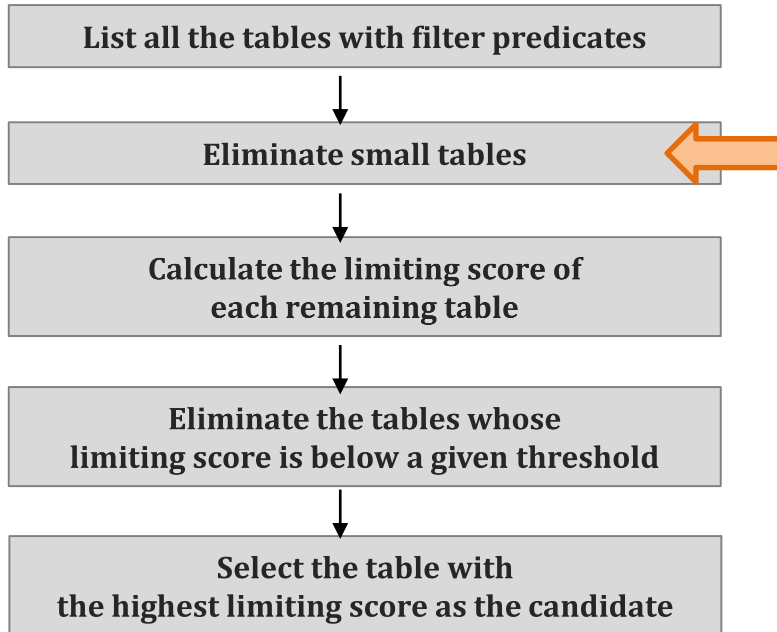


TPC-H Query 2

```
SELECT s_acctbal, s_name, n_name, p_partkey, p_mfgr,
s_address, s_phone, s_comment
FROM part, supplier, partsupp, nation, region
WHERE p_partkey = ps_partkey AND s_suppkey =
ps_suppkey AND p_size = 15 AND p_type LIKE
'%BRASS' AND s_nationkey = n_nationkey AND
n_regionkey = r_regionkey
AND r_name = 'EUROPE' AND ps_supplycost =
    (SELECT MIN(ps_supplycost)
    FROM partsupp, supplier, nation, region
    WHERE p_partkey = ps_partkey AND s_suppkey =
    ps_suppkey AND s_nationkey = n_nationkey AND
    n_regionkey = r_regionkey
    AND r_name = 'EUROPE')
ORDER BY s_acctbal DESC, n_name, s_name, p_partkey
LIMIT 100;
```

- Region table: Single predicate
- Part table: Two predicate

YourSQL Query Engine – Join Order Optimization

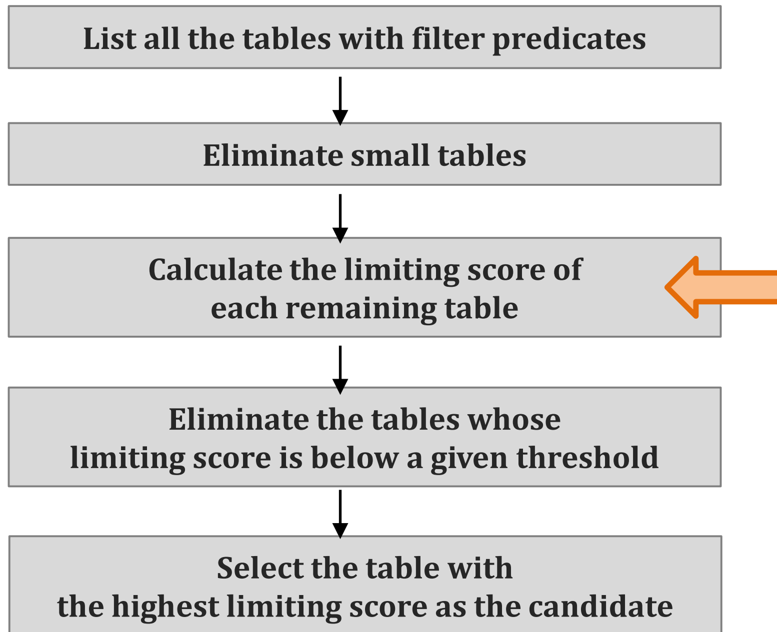


TPC-H Query 2

SELECT s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address, s_phone, s_comment
FROM part, supplier, partsupp, nation, region
WHERE p_partkey = ps_partkey AND s_suppkey = ps_suppkey AND **p_size = 15 AND p_type LIKE '%BRASS'** AND s_nationkey = n_nationkey AND n_regionkey = r_regionkey
AND **r_name = 'EUROPE'** AND ps_supplycost =
(**SELECT** MIN(ps_supplycost)
FROM partsupp, supplier, nation, region
WHERE p_partkey = ps_partkey AND s_suppkey = ps_suppkey AND s_nationkey = n_nationkey AND n_regionkey = r_regionkey
AND r_name = 'EUROPE')
ORDER BY s_acctbal DESC, n_name, s_name, p_partkey
LIMIT 100;

- ~~Region table: Single predicate~~ **only five rows**
- Part table: Two predicate

YourSQL Query Engine – Join Order Optimization



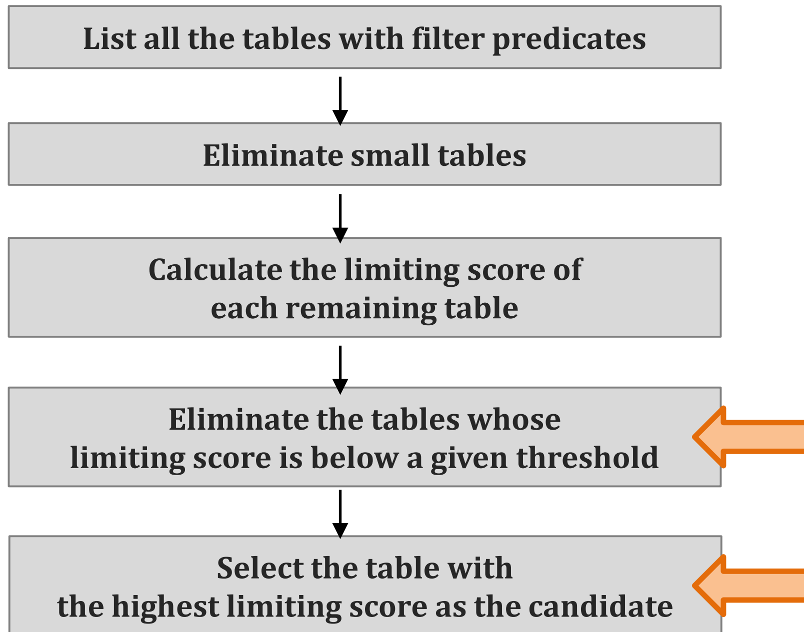
TPC-H Query 2

SELECT s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address, s_phone, s_comment
FROM part, supplier, partsupp, nation, region
WHERE p_partkey = ps_partkey AND s_suppkey = ps_suppkey AND **p_size = 15 AND p_type LIKE '%BRASS'** AND s_nationkey = n_nationkey AND n_regionkey = r_regionkey
AND **r_name = 'EUROPE'** AND ps_supplycost =
(**SELECT** MIN(ps_supplycost)
FROM partsupp, supplier, nation, region
WHERE p_partkey = ps_partkey AND s_suppkey = ps_suppkey AND s_nationkey = n_nationkey AND n_regionkey = r_regionkey
AND r_name = 'EUROPE')

- **Add a limiting score of each filter predicate**
- **A filter predicate gets a higher score as its type of operation is more restrictive (e.g., EQUAL)**

  - ~~Region table: Single predicate~~
  - Part table: Two predicate

YourSQL Query Engine – Join Order Optimization



TPC-H Query 2

SELECT s_acctbal, s_name, n_name, p_partkey, p_mfgr, s_address, s_phone, s_comment
FROM part, supplier, partsupp, nation, region
WHERE p_partkey = ps_partkey AND s_suppkey = ps_suppkey AND **p_size = 15 AND p_type LIKE '%BRASS‘** AND s_nationkey = n_nationkey AND n_regionkey = r_regionkey
AND **r_name = 'EUROPE‘** AND ps_supplycost =
(**SELECT** MIN(ps_supplycost)
FROM partsupp, supplier, nation, region
WHERE p_partkey = ps_partkey AND s_suppkey = ps_suppkey AND s_nationkey = n_nationkey AND n_regionkey = r_regionkey
AND r_name = 'EUROPE')
ORDER BY s_acctbal DESC, n_name, s_name, p_partkey
LIMIT 100;

- ~~Region table: Single predicate~~
- Part table: Two predicate

YourSQL Query Engine – Join Order Optimization

- Query Plan of YourSQL for TPC-H Query 2

Join order	Table name	Access method
1	Region	All
2	Nation	Ref
3	Supplier	Ref
4	Partsupp	Ref
5	Part	Eq_ref

(a) MySQL w/o ICP

Join order	Table name	Access method
1	Part	Ref
2	Partsupp	Ref
3	Supplier	Eq_ref
4	Nation	Eq_ref
5	Region	All

(b) MySQL w/ ICP

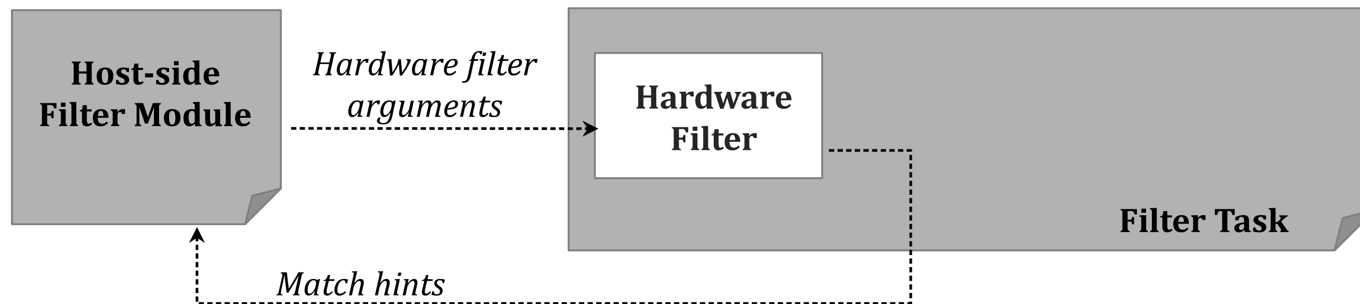
Join order	Table name	Access method	Key
1	Part	All	Null
2	Partsupp	Ref	PK
3	Supplier	Eq_ref	PK
4	Nation	Eq_ref	PK
5	Region	All	Null

(c) YourSQL

- Intermediate row sets can significantly be reduced at the earliest stage of join!
- MySQL w/ICP performs early filtering by secondary indexes on filter columns. In contrast, YourSQL performs early filtering with the ISC filters, which scan the early filtering target.

YourSQL Storage Engine – Filtering Condition Pushdown (FCP)

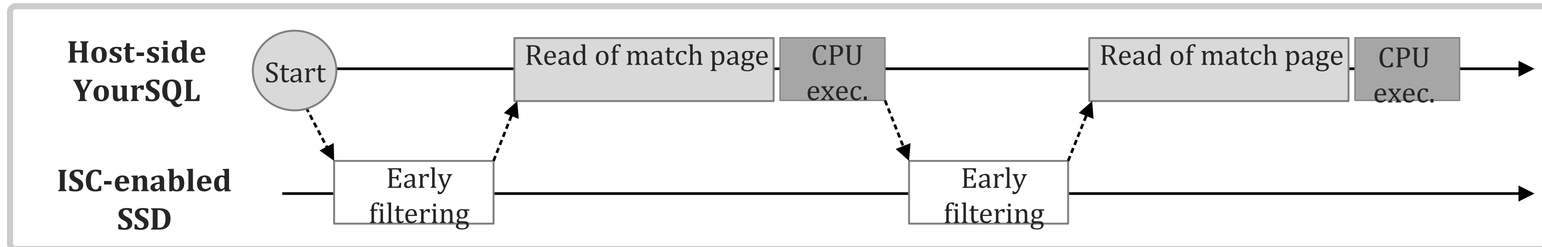
- An optimization for the case where YourSQL retrieves rows from a table using filter predicates
- YourSQL's filter leverages the H/W pattern matcher
  - Transforms filter predicates into binary patterns and feeds them to the ISC Filter task
  - E.g., in TPC-H Query2, p_type LIKE '%BRASS' is converted into binary keys, '42 52 41 53 53'



- *Match hints: a byte array whose element is set to 1 if the corresponding page satisfies filtering conditions.*

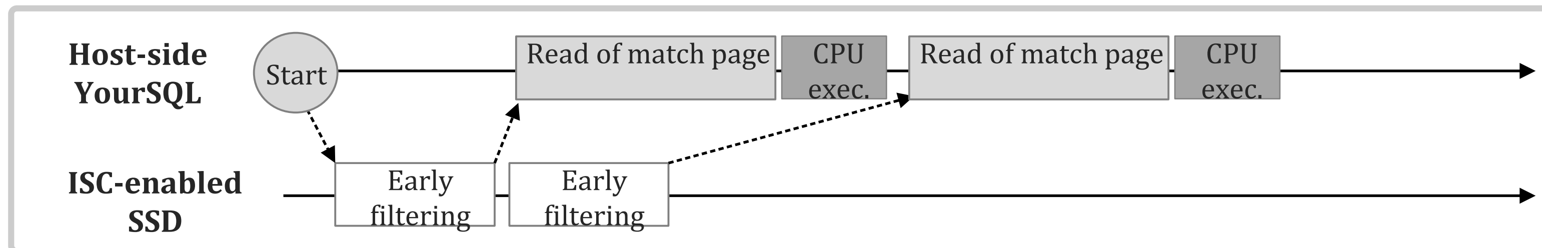
YourSQL Storage Engine – Table Access using Match Hints

- YourSQL checks match hints first, and fetches pages whose match hint is set to one with "normal host read"



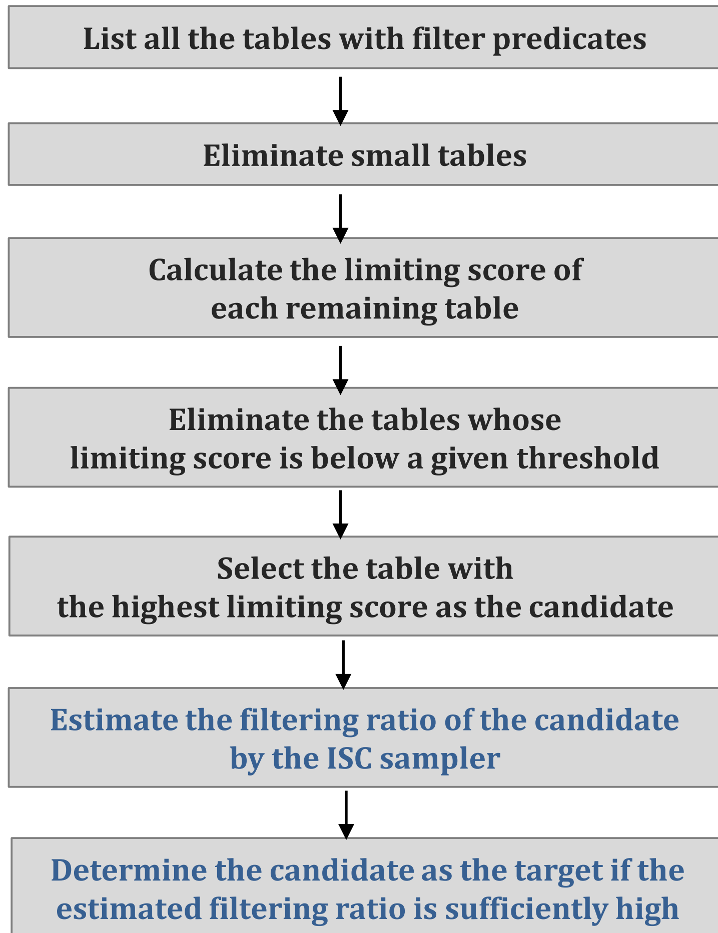
(a) Sequential processing

- Early filtering task and the remaining tasks (i.e., match page reads and row processing) run concurrently in the ISC-enabled SSD and the host



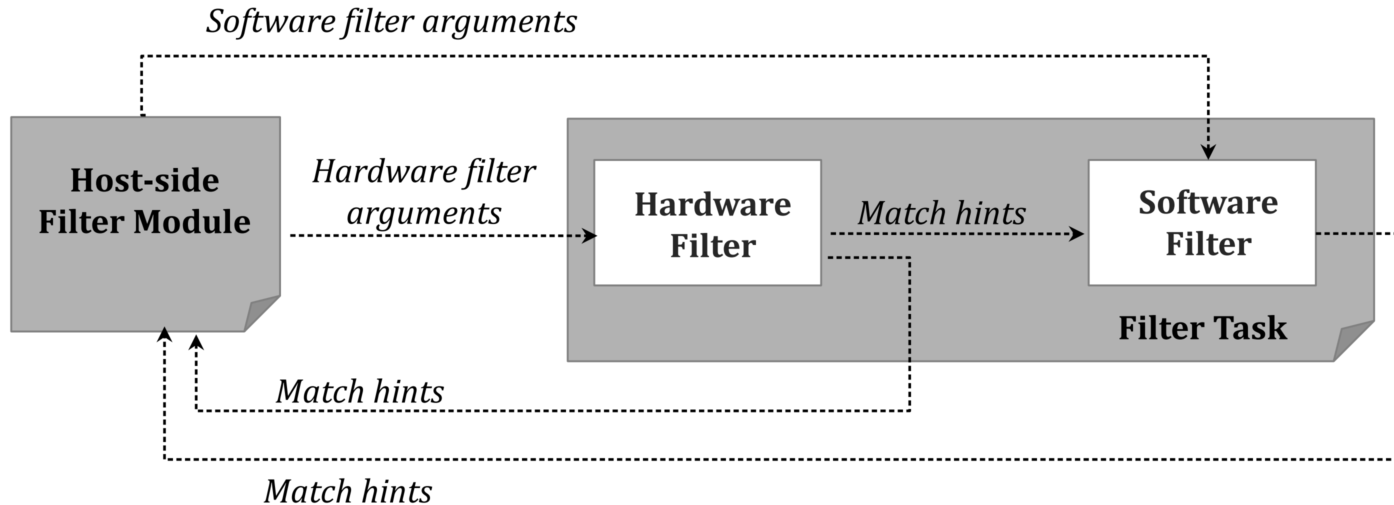
(b) Concurrent processing

Optimization – Sampling-driven FCP



- The limiting score is a simple heuristic, but not quantitatively correlated with filtering ratio
- An ISC task called "sampler" is used to provide a quantitative estimation of filtering ratio
  - Sampler is the same as the filter functionality-wise, but scans the sampling region only
- The estimated filtering ratio enables YourSQL to check further if early filtering for a candidate table would really be beneficial in terms of execution time

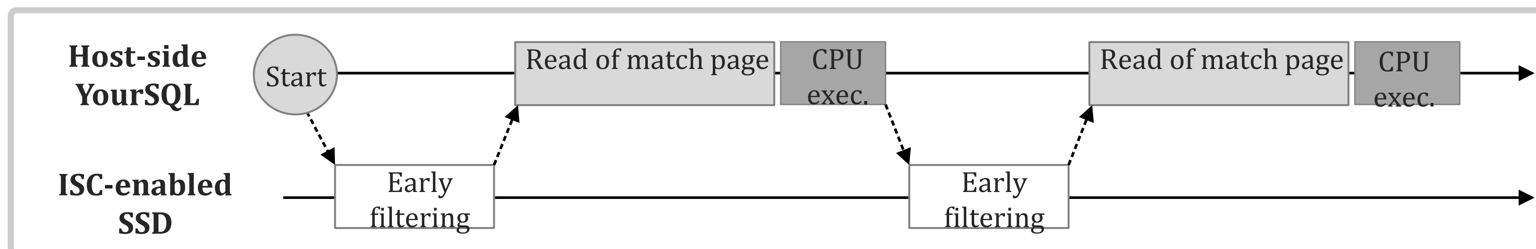
Optimization – Software Filtering



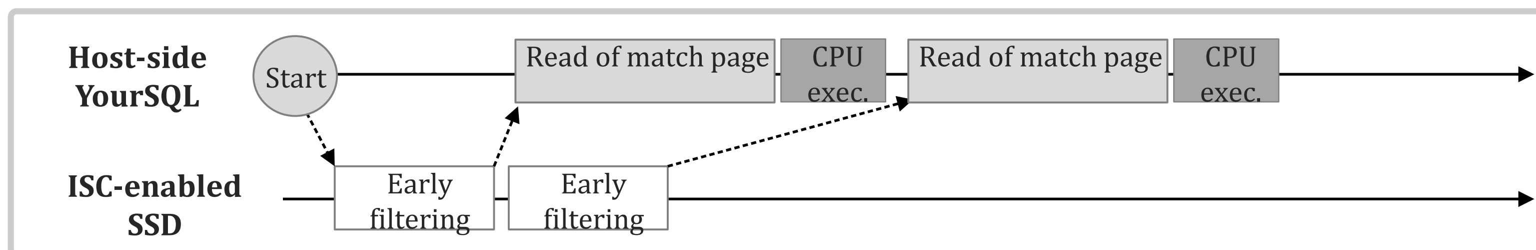
- Hardware matcher only performs byte-granular matching and the filtered data may still contain false positives depending on the filtering conditions
  - e.g., shipdate > `1995-09-01' and l shipdate <`1995-09-01' + INTERVAL 1 MONTH

    -> `8F 97 21' and `8F 97 41‘ -> `8F 97‘ (extract common two byte sequence)
  - ‘8F 97’ would match sequences from ‘8F 97 00’ through ‘8F 97 FF’

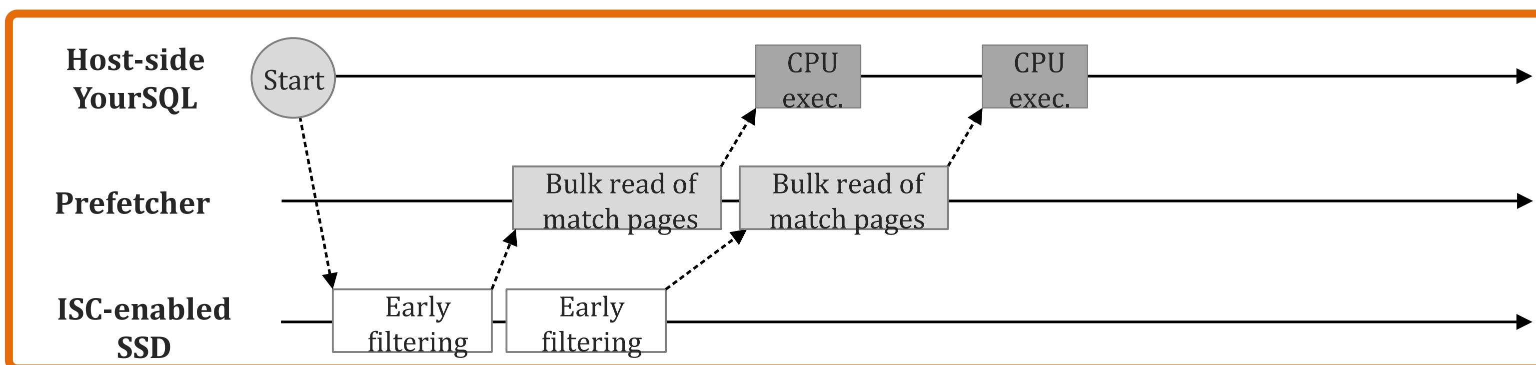
Optimization – Highly Accurate Bulk Prefetch



(a) Synchronous reads of single-page units



(b) Synchronous reads of multi-page units



(c) Asynchronous reads of multi-page units

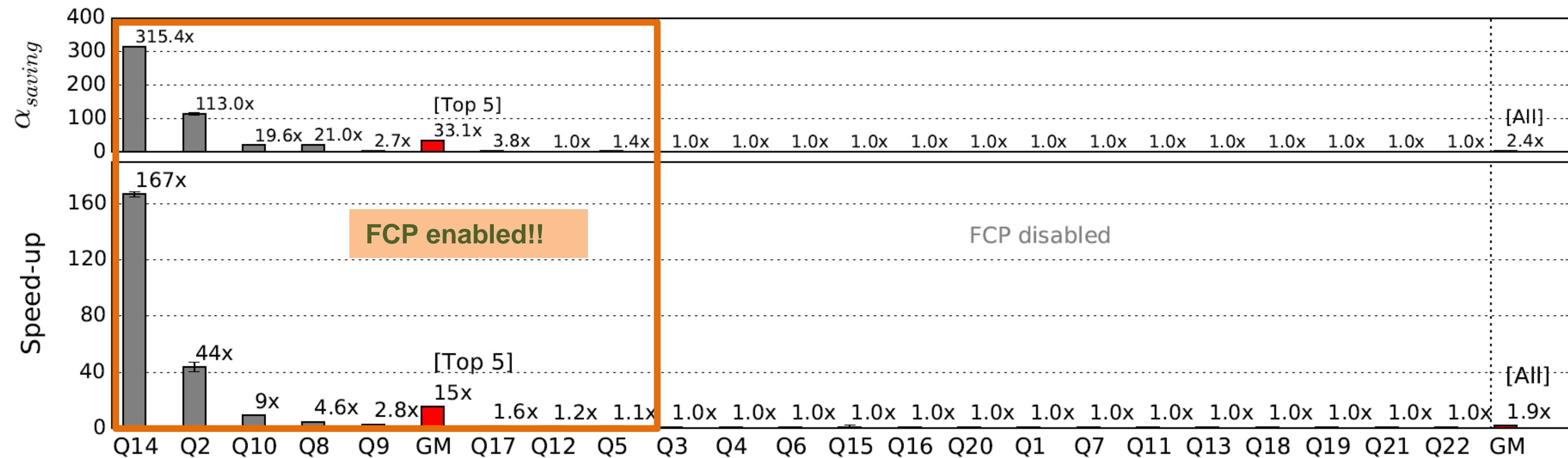
Experimental Setup

- H/W setup

System	Dell PowerEdge R720 server
CPU	2 Intel Xeon(R) CPU E5-2640 (12 threads per socket) @2.50GHz
Memory	16 GiB DRAM
SSD	Samsung PM1725 1TB (ISC-enabled)
OS	64-bit Ubuntu 15.04

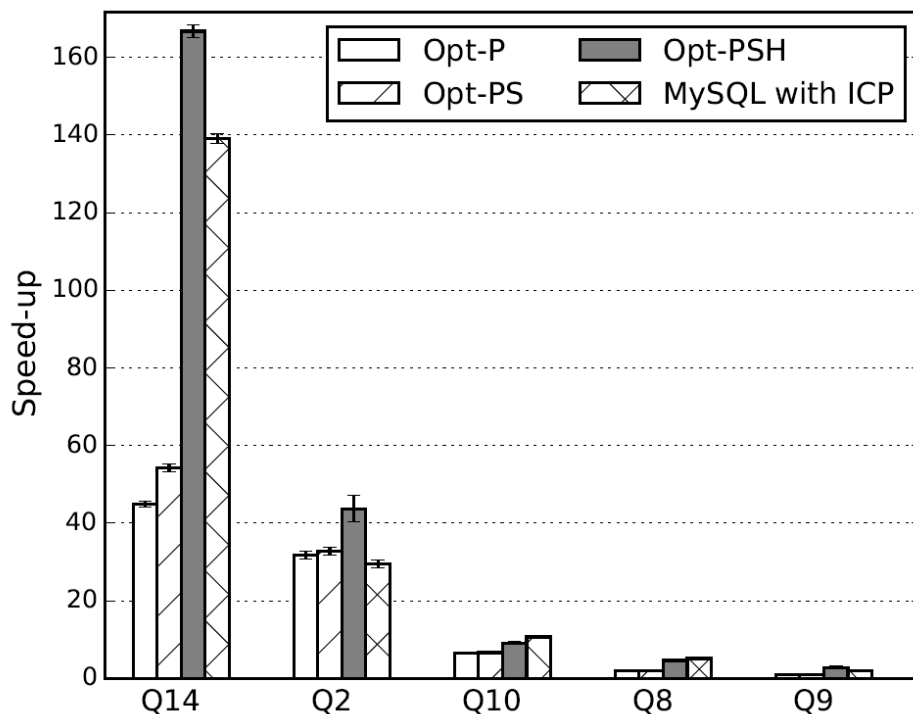
- Baseline system and workload
  - MariaDB 5.5.42 was integrated with Biscuit framework
  - TPC-H with a scale factor of 100 was chosen

Evaluation Results – TPC-H results



- Out of 22 queries, eight queries were FCP-enabled
  - The rest queries had no filter predicates or YourSQL did not expect speed-up for FCP
- The average speed-up of the top five queries reached 15x
- 3.6x reduction of the overall execution time was achieved

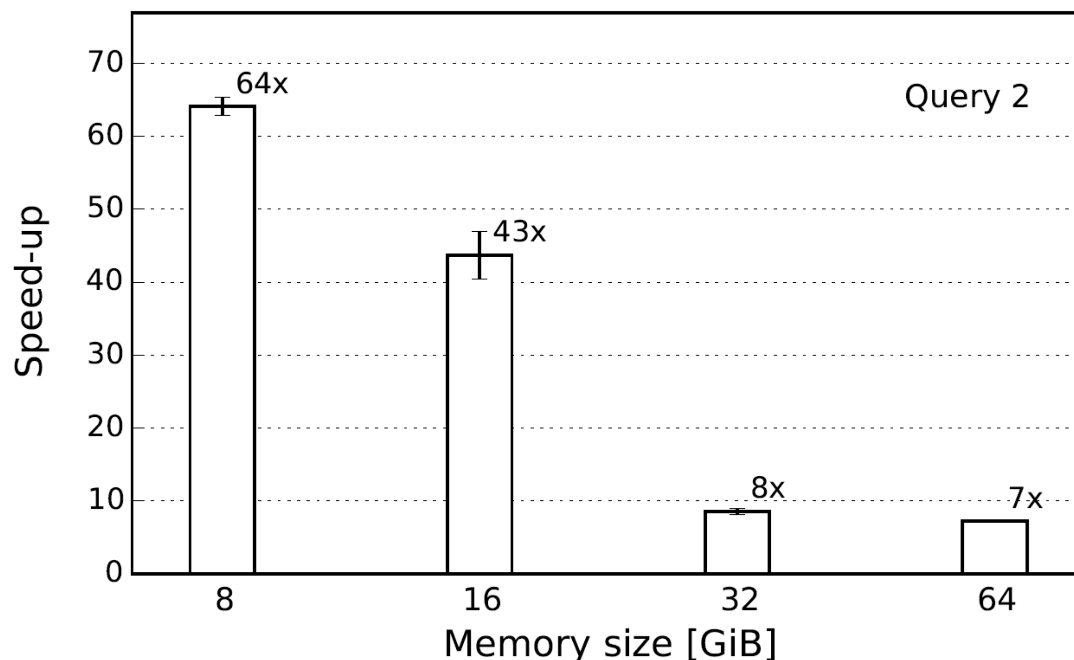
Evaluation Results – Optimization Techniques



Scheme	Configuration
Opt-P	Hardware filter
OPT-PS	Hardware filter + Software filter
OPT-PSH	Hardware filter + Software filter + HABP (Highly Accurate Bulk Prefetch)

- More optimizations yield higher speed-up, since each optimization scheme is orthogonal to one another
- The biggest improvement seen in Opt-PSH implies that the host-side read operation was the limiting factor in accelerating the overall performance

Evaluation Results – Memory Size



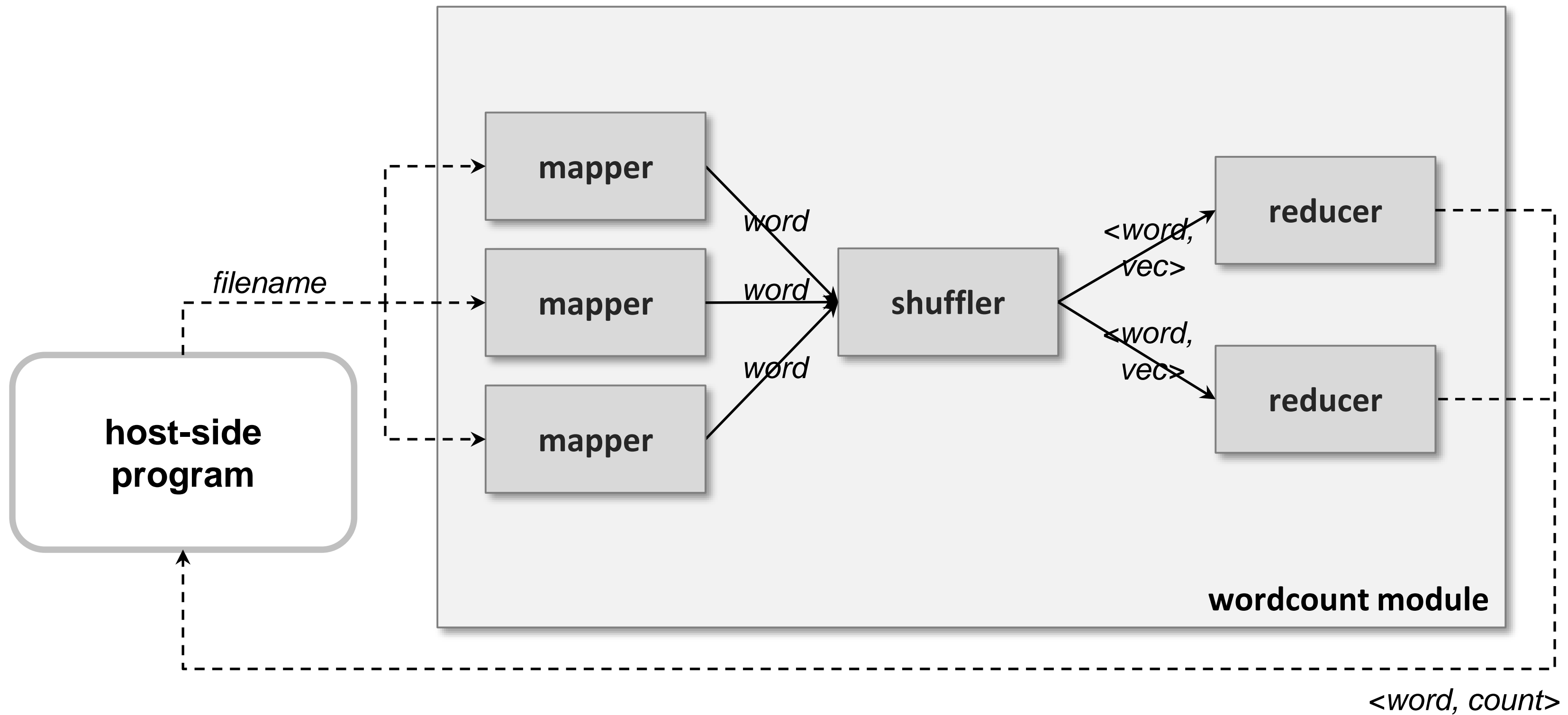
- As the memory size decreases, the resulting speed-up becomes higher
- When the memory usage becomes tighter, the relative cost of read I/O is increased and the impact of its reduction becomes more prominent

Conclusions

- We presented the design and implementation of YourSQL, an ISC-enabled database system.
- With YourSQL, we pursued accelerating data-intensive queries with the help of additional in-storage computing capabilities.
- We seamlessly integrated query offloading to SSDs into one of the most popular database systems, MySQL.
- YourSQL accomplished the 3.6x reduced execution time for TPC-H queries.

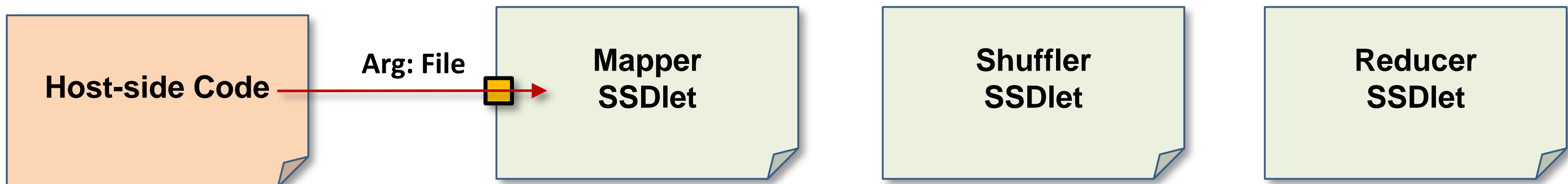
Appendix

Wordcount Example



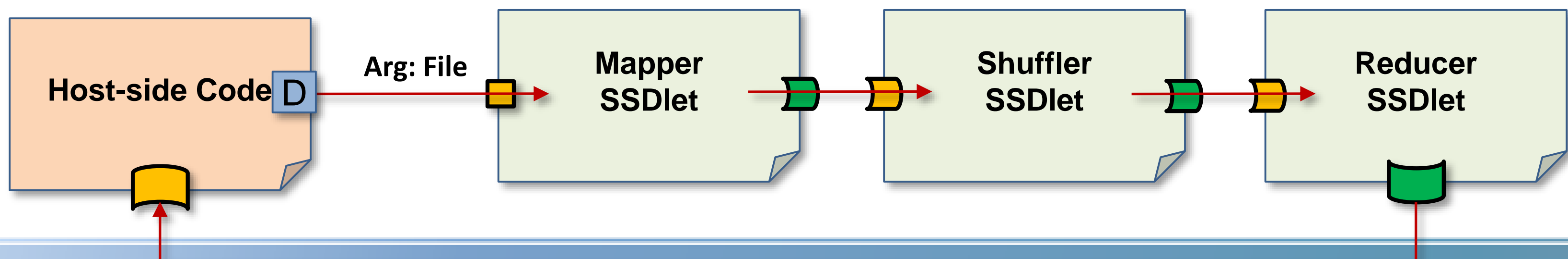
Wordcount Example – Host-side Code

```
int main(int argc, char *argv[]) {
    // create an instance of the SSD class that corresponds to an Smart SSD
    SSD ssd("/dev/nvme0n1p1");

    // load an SSDlet stored on the Smart SSD
    File file(ssd, "/var/isc/slets/libwordcount.so");
    module_id_t mid = ssd.loadModule(std::move(file));

    // create an instance of the Application class to manage SSDlets on the Smart SSD
    Application wordcount(ssd);

    // create instances of necessary SSDlet classes included in the loaded module
    auto args = std::make_tuple(File(ssd, argv[1]));
    SSDLet mapper(wordcount, mid, "idMapper", std::move(args));
    SSDLet shuffler(wordcount, mid, "idShuffler");
    SSDLet reducer(wordcount, mid, "idReducer");
```



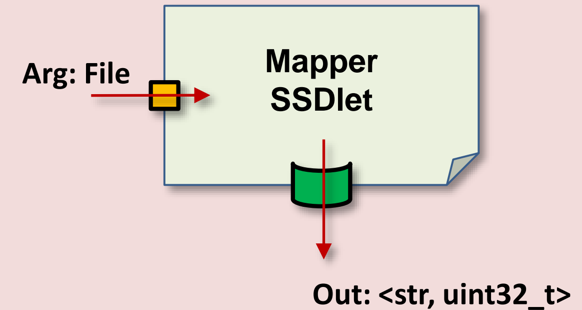
Wordcount Example – Host-side Code

```
    // make connections between SSDlets
    wordcount.connect(mapper.out(0), shuffler.in(0));
    wordcount.connect(shuffler.out(0), reducer.in(0));

    auto port = wordcount.connectTo<std::pair<std::string, uint32_t>>(reducer.out(0));

    // starting application would make all SSDlets begin execution
    wordcount.start();

    std::pair<std::string, uint32_t> value;
    // keep reading as long as output is available
    while (port.get(value))
            std::cout << value.first << "\t" << value.second << std::endl;

    ssd.unloadModule(mid);
    return 0;
}
```



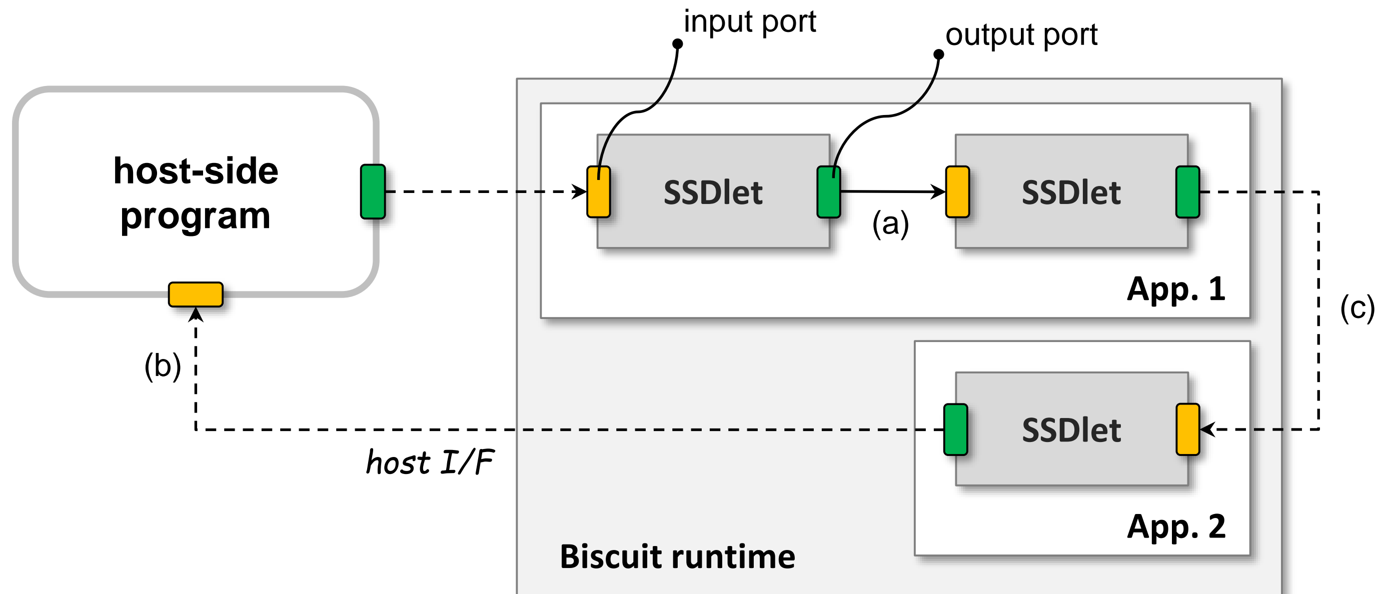
Wordcount Example – SSDlet: Mapper

```
class Mapper
    : public SSDLet<OUT_TYPE<std::pair<std::string, uint32_t>>,
                    ARG_TYPE<File>> {
public:
    // SSDlet start function
    void run() {
        // get filename as argument from host-side code
        auto& file = getArgument<0>();
        // get outputPort connected with Shuffler SSDlet
        auto output = getOutputPort<0>();

        // do Mapper tasks
        FileStream fs(std::move(file));
        while (true) {
            sstring line;
            if (!readline(fs, line))
                break;

            line.tokenize();
            sstring::const_iterator word;
            while ((word = line.next_token()) != line.cend()) {
                // send results to Shuffler SSDlet through pipe
                if (!output.put({std::string(word), 1}))
                    return;
}}}};
// register ‘Mapper’ SSDlet
RegisterSSDLet(idMapper, Mapper)
```

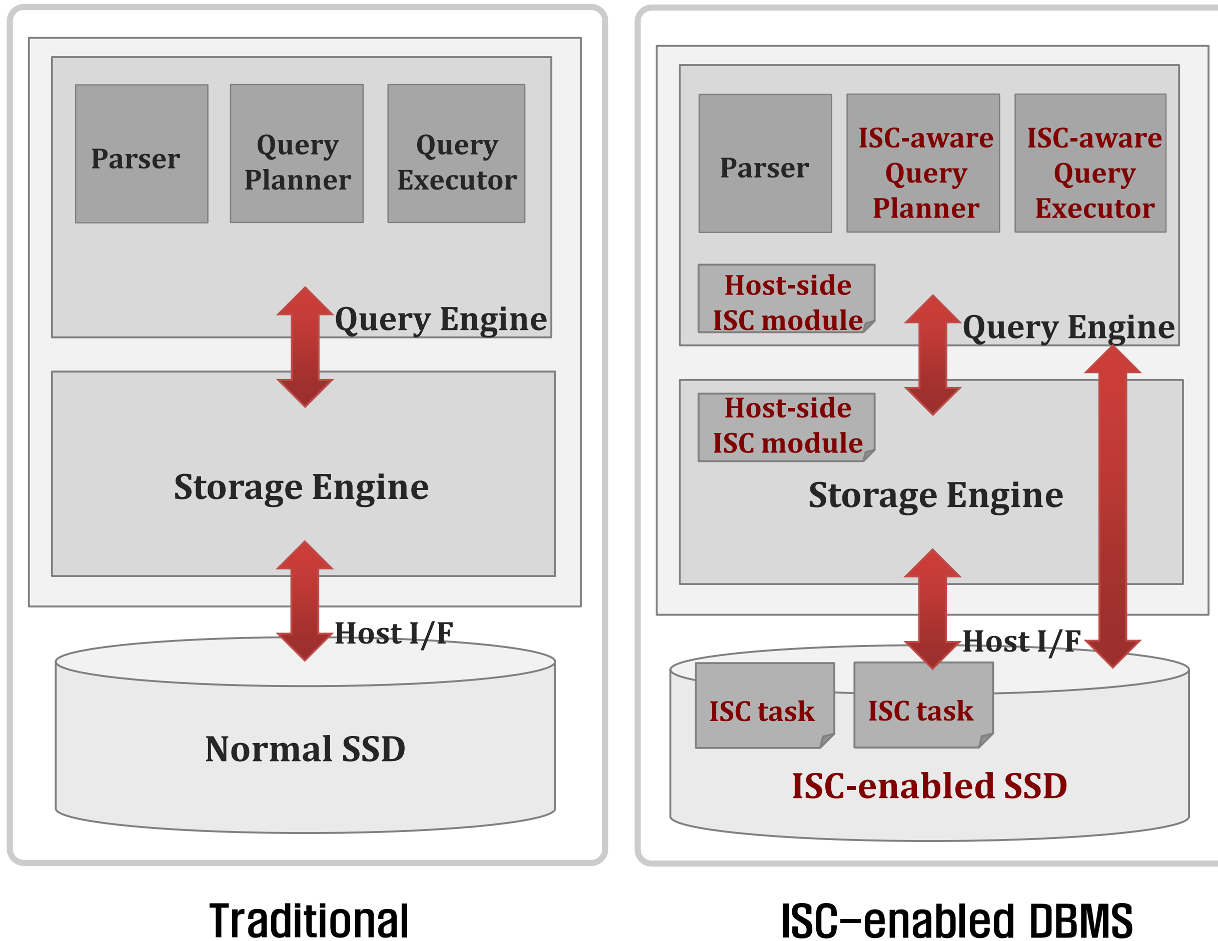


Biscuit I/O Ports

- Communication through ports
  - (a) Inter-SSDlet ports: among SSDlet instances belonging to a single Application instance
  - (b) Host-to-device ports: between an SSDlet instance and a host program
  - (c) Inter-application ports: between two SSDlets from different Application instances



ISC-enabled Database System



- Design Considerations
  - Partitioning host/ISC tasks
  - Defining interfaces between a host and ISC tasks
  - Optimizing query planner for ISC
  - Reorganizing datapath for ISC database system