In-Storage Compute: an Ultimate Solution for Accelerating I/O-intensive Applications

13 August 2015

YANG SEOK **KI**, Director, Memory Solutions Lab, Samsung Electronics

- **Disclaimer: The contents provided in this material are based on concepts and early research results, and are for technical discussions only. This material does not reflect any product-level plan of records.**

Contributors

Korea HST Team

Boncheol **GU**

Duckho **BAE**

Sungho **YOON**

US ADS Team

Yangwook **KANG**

Heekwon **PARK**

Special thanks to:
Yoonho **Chung**, Insoon **Jo**, Minwook **Jung**, Jungwook **Kang**, Moonsang **Kwon,** Truong **Nguyen**, Dongchul **Park,** Prem **Paulson**, Jonghyun **Yoon**

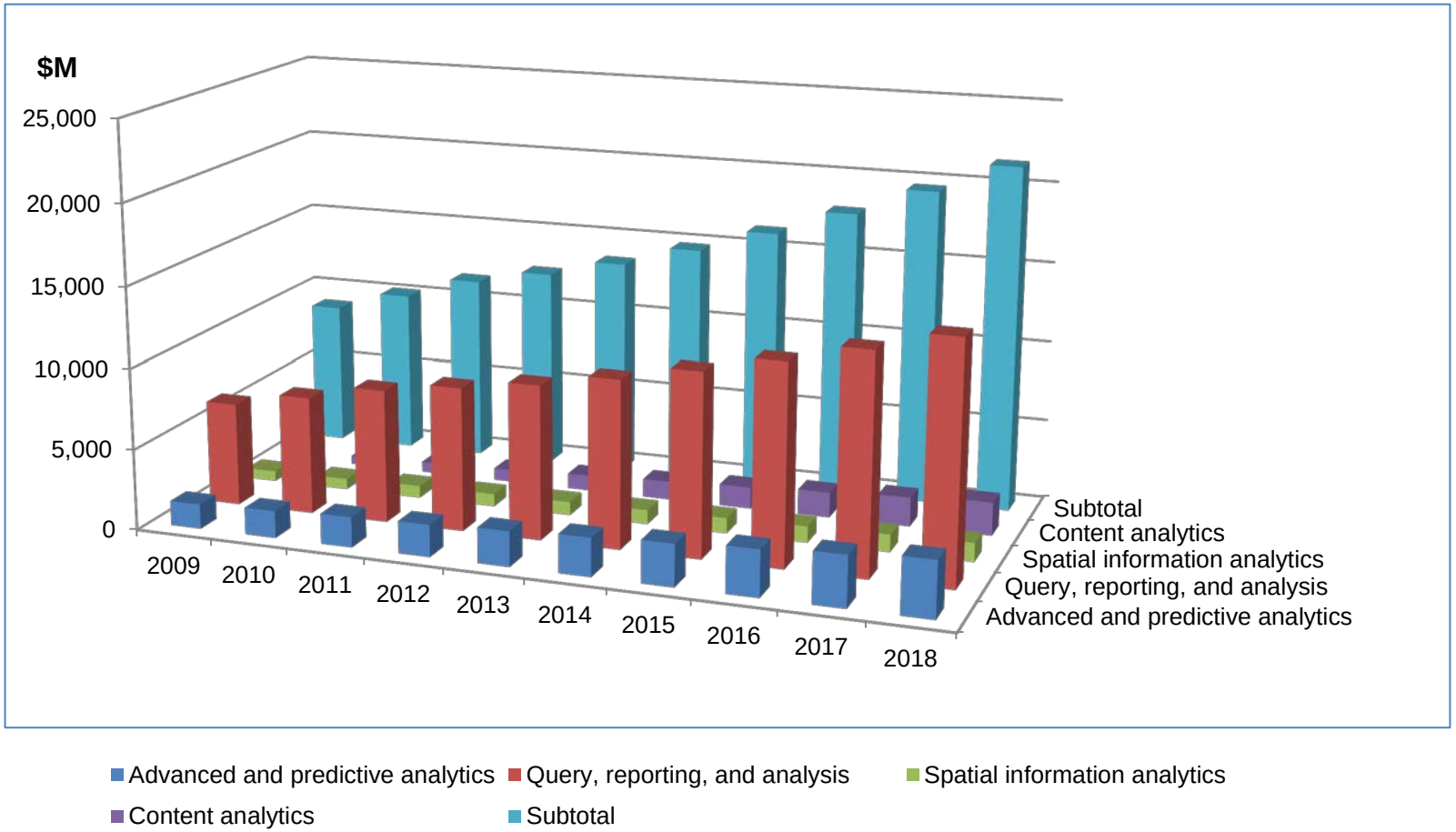
Outline

1. Background
2. In-Storage Compute Concept
3. ISC Prototype
4. Case Studies
5. Summary

Data Processing Market

- **Constant growth of business intelligence & analytics market**

Worldwide BI and Analytics Tools Revenue by Segment



Source: IDC, Worldwide Business Analytics Software 2014–2018 Forecast, Jul 2014 [1]

I/O Performance Issue

BI and analytics tools are I/O hungry!!

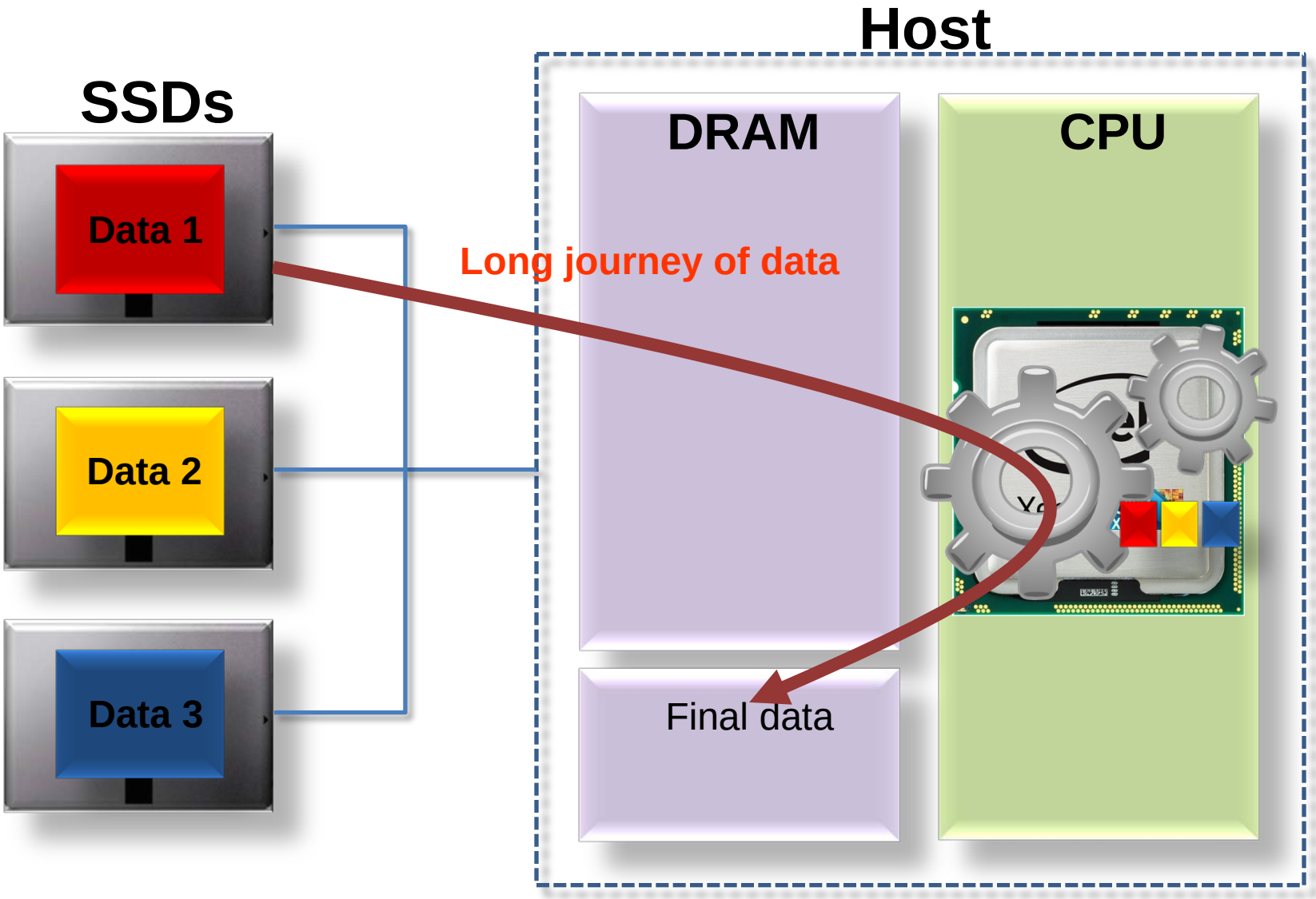
- They usually access terabytes (sometimes even petabytes) of data on slow storage device

OLAP Bottleneck!!

Event	Waits	Time(s)	Avg. wait (ms)	% DB time	Wait class
Direct path read	4,604,339	567.141	123	63.67	User I/O
Direct path read temp	1,955,162	147,298	75	16.54	User I/O
DB CPU		38,874		4.36	
DB file sequential read	117,944	16,399	139	1.84	User I/O
Direct path write temp	597,138	13,507	23	1.52	User I/O

Source: HUAWEI, Accelerate Oracle Performance, Sep 2012 [2]

CPU-centric Computing Model (Von Neumann)



Moving Data is Expensive!!

Moving Computation is Cheaper than Moving Data

Source: HDFS Architecture Guide [3]

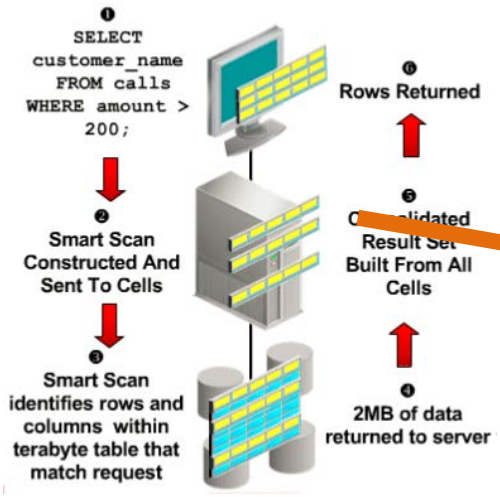
Reducing data movement can help improve both energy and performance

Source: USENIX HotPower, 2012 [4]

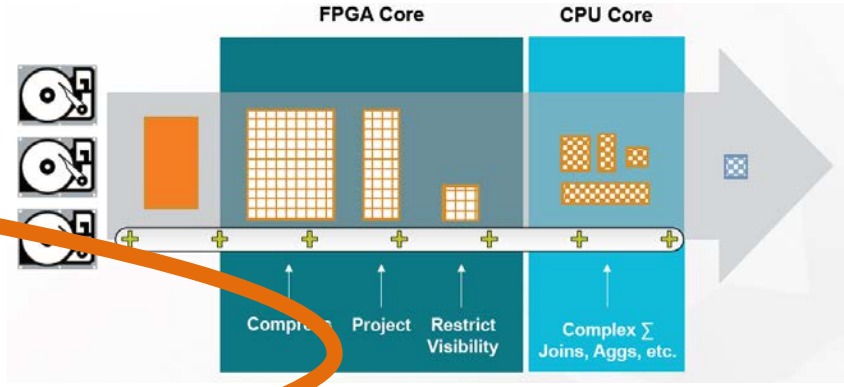
The energy consumed by data movement is starting to exceed the energy consumed by computation

Source: High Performance parallel IO, 2014 [5]

Near Data Processing Technology

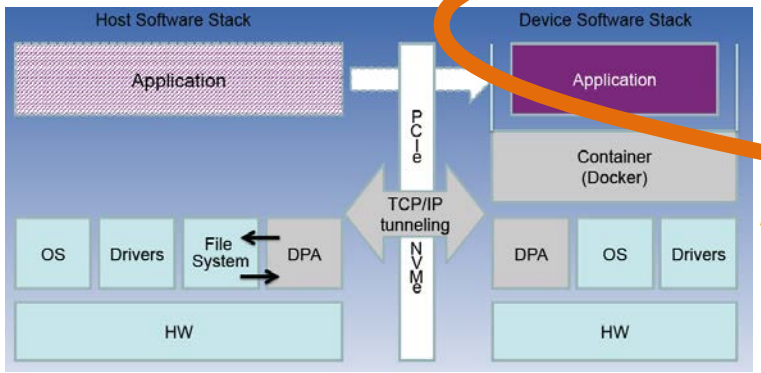


Exadata [Oracle]

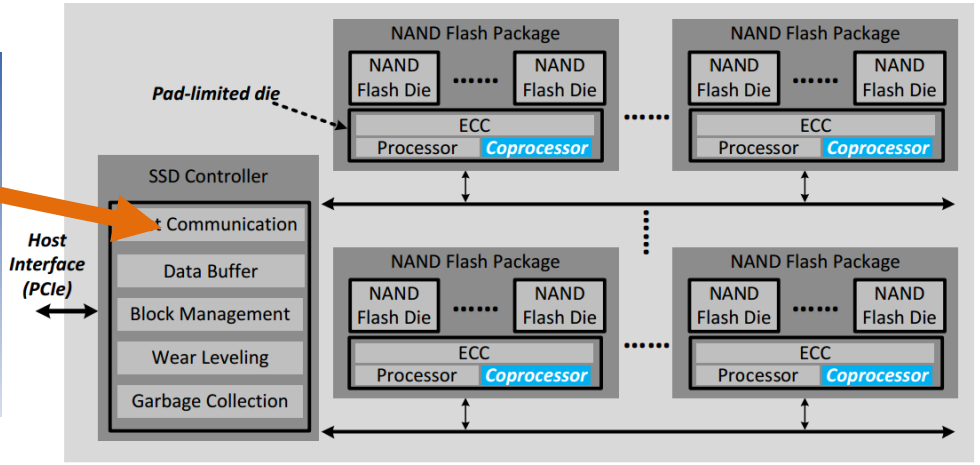


Netezza S-blade [IBM]

Closer to source



Intelligent SSD [NxGnData]

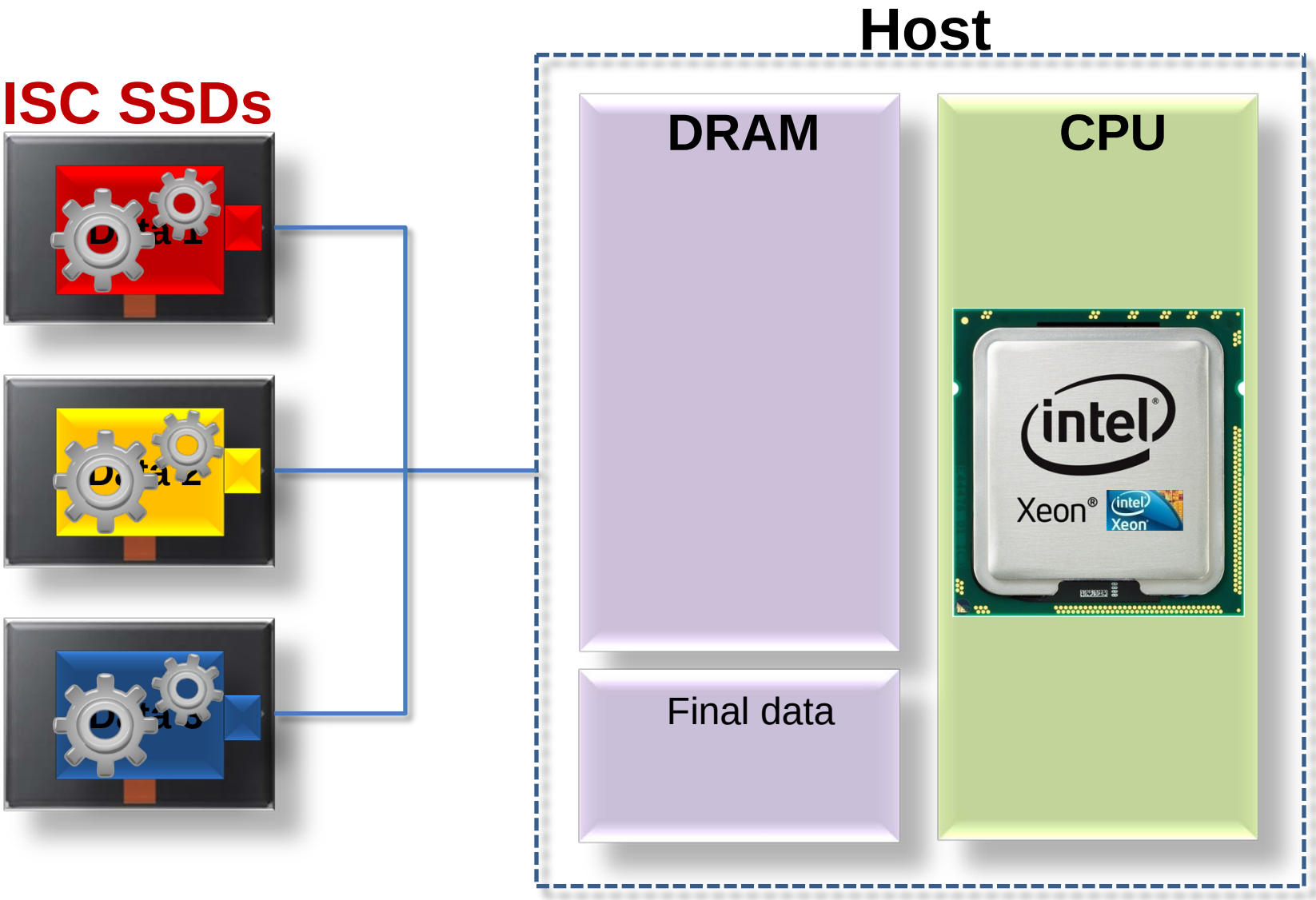


SPU (Storage Processing Unit) [Seagate]

The ultimate of close-to-data compute for high performance & low power is

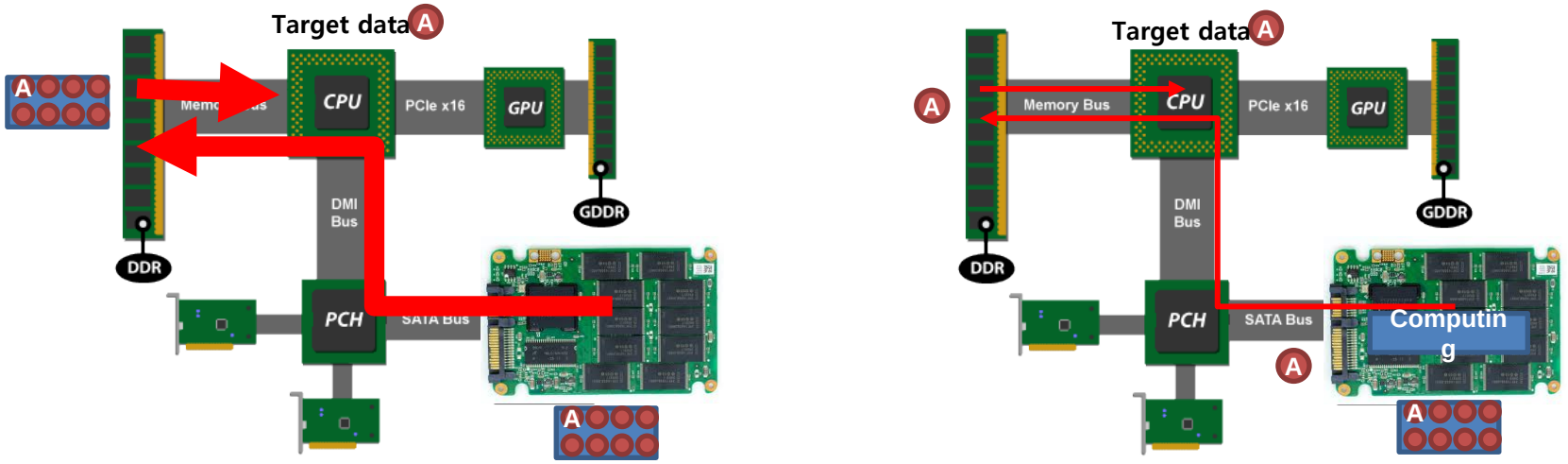
"In-Storage Compute (ISC)"

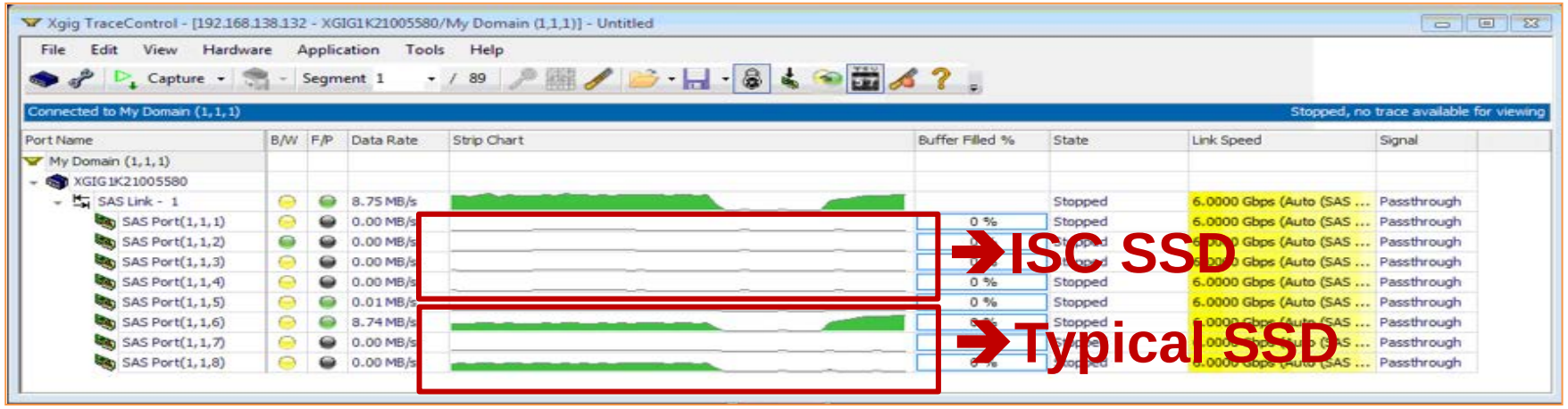
What is ISC (In-Storage Compute)?



Why? IO Traffic

- **ISC is an ultimate approach to IO reduction/avoidance**

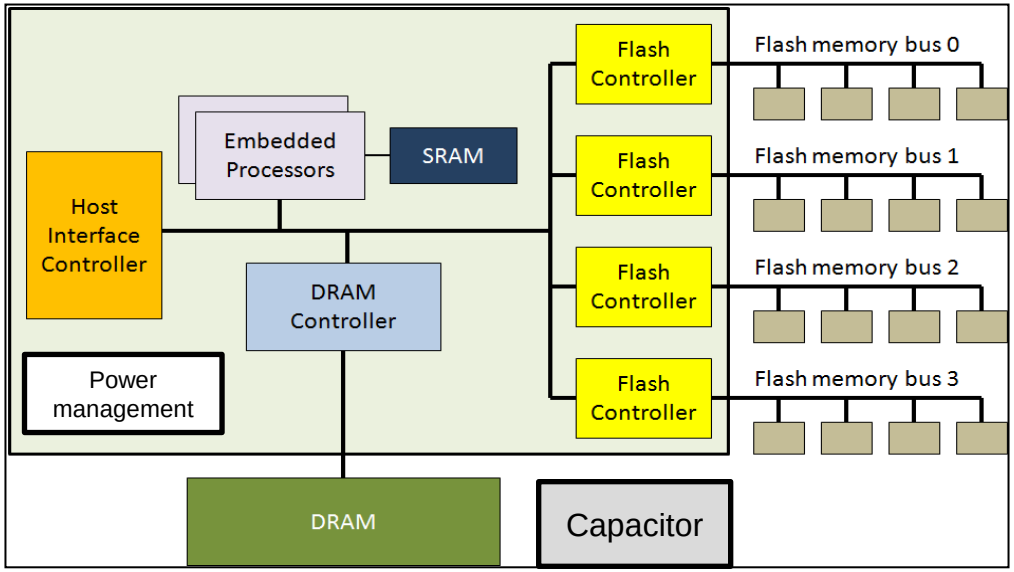




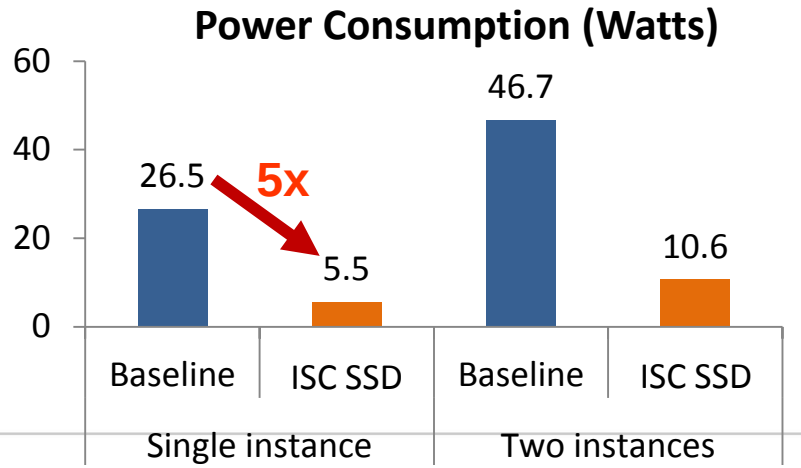
Samsung SAS-based ISC with a Hadoop application

Why? Coprocessor

- **SSD is a complete computer with high performance low power processor**



VS

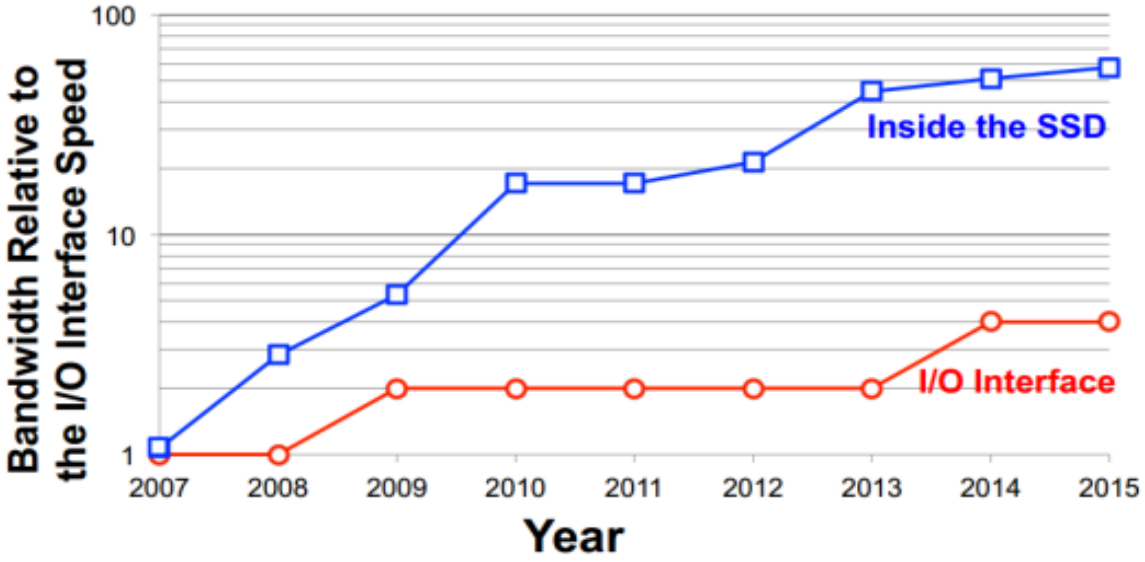


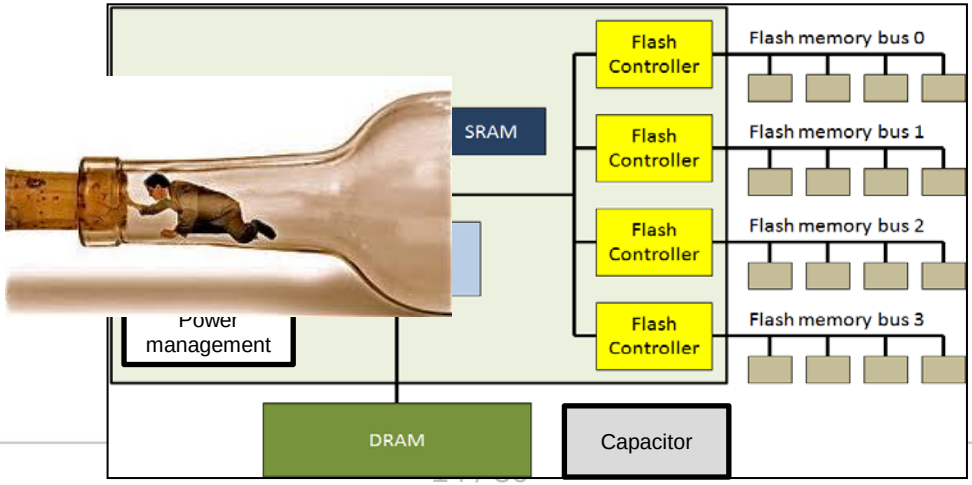
Samsung SAS-based ISC with a Hadoop application

Why? Bandwidth Gap

Superfluous internal bandwidth

- To hide processing overhead of host interface and FTL





Why? Resource Utilization

- **Storage resource is underutilized**

How? ISC Application Development Process

Host app

ssdlet

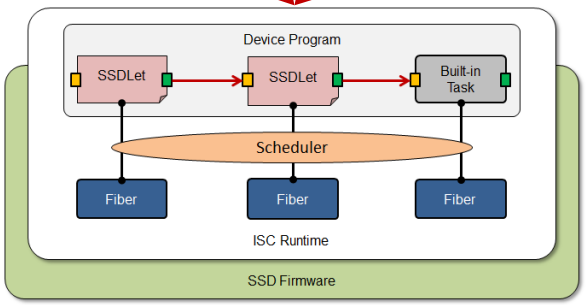
1 C/C++
- Support C++11/STL

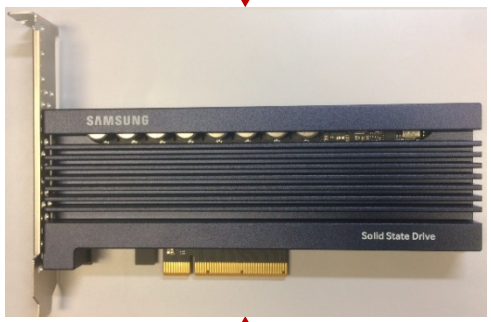
2 X86 Compile

4 ARM Cross compile

3 ISC SSD Emulation

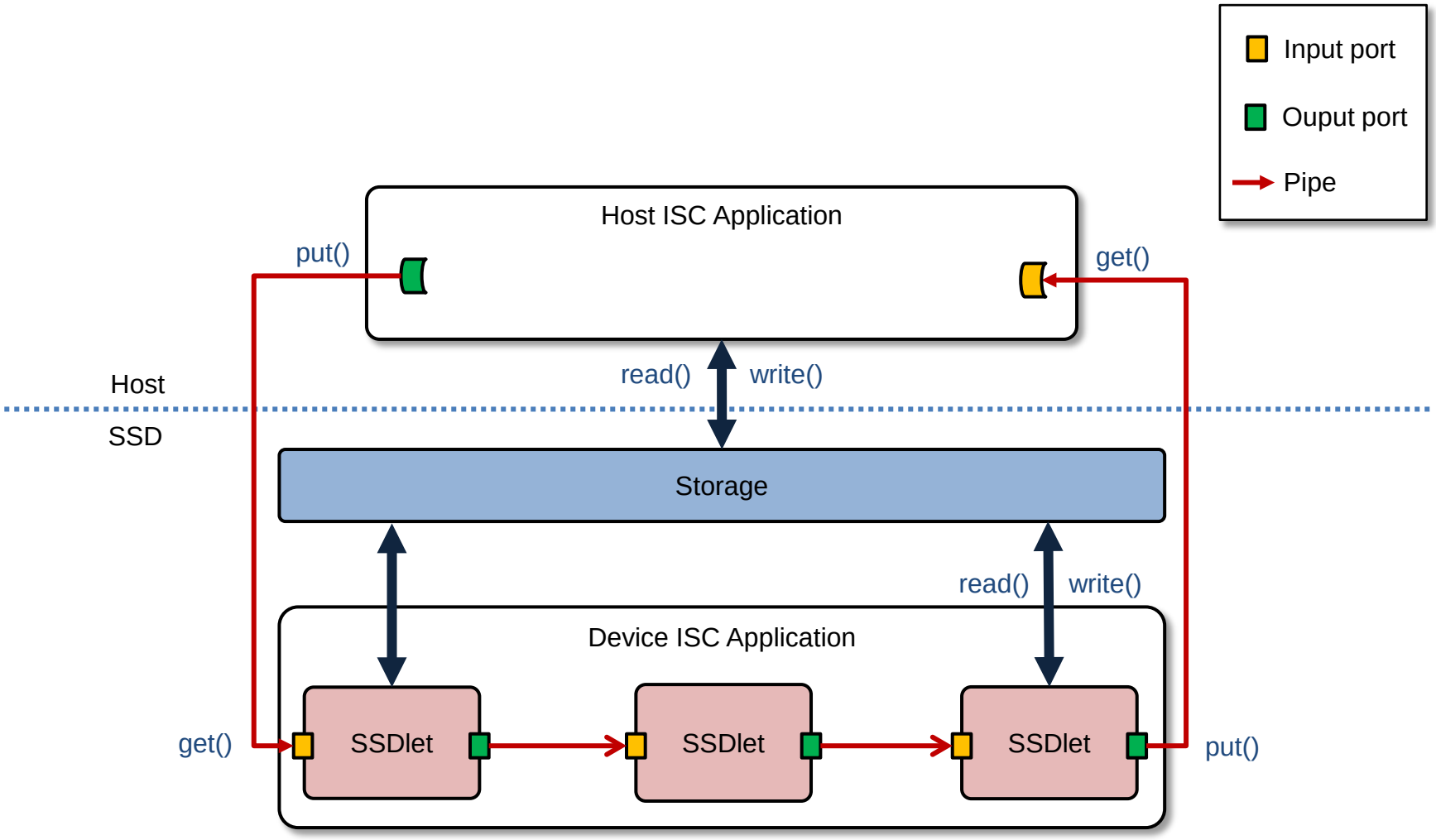
5 Download /isc/myprogram/ssdlet



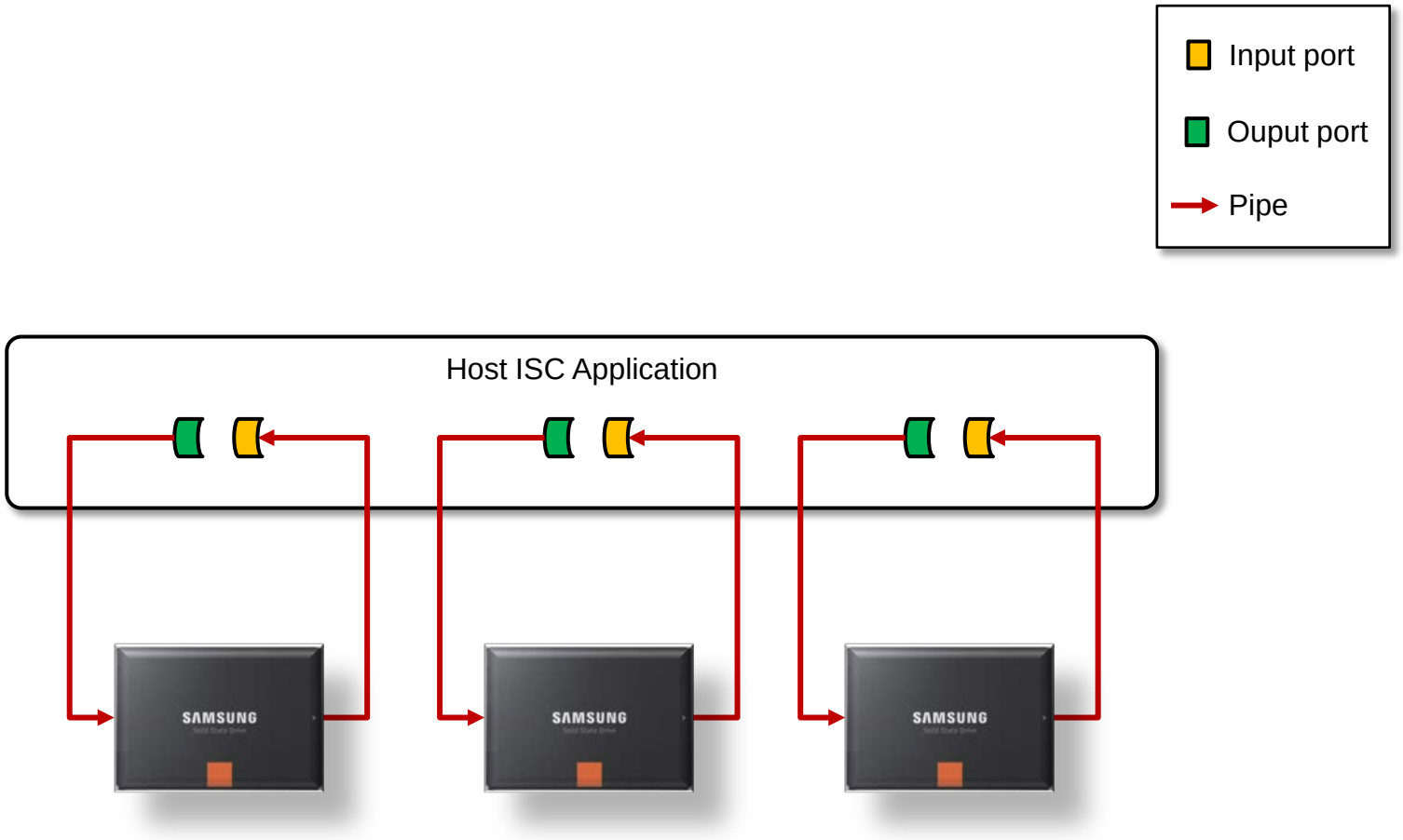


6 Run /isc/myprogram/host

ISC Dataflow Programming Model



ISC Multiple Device Model



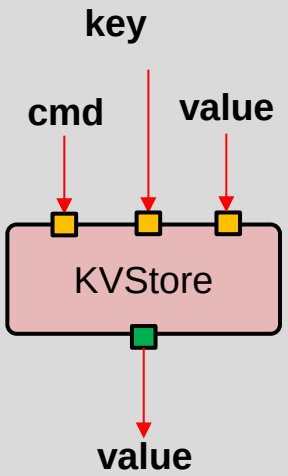
Simple Key Value Store in ISC Programming


```
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;
                }
            }
        }
};
```

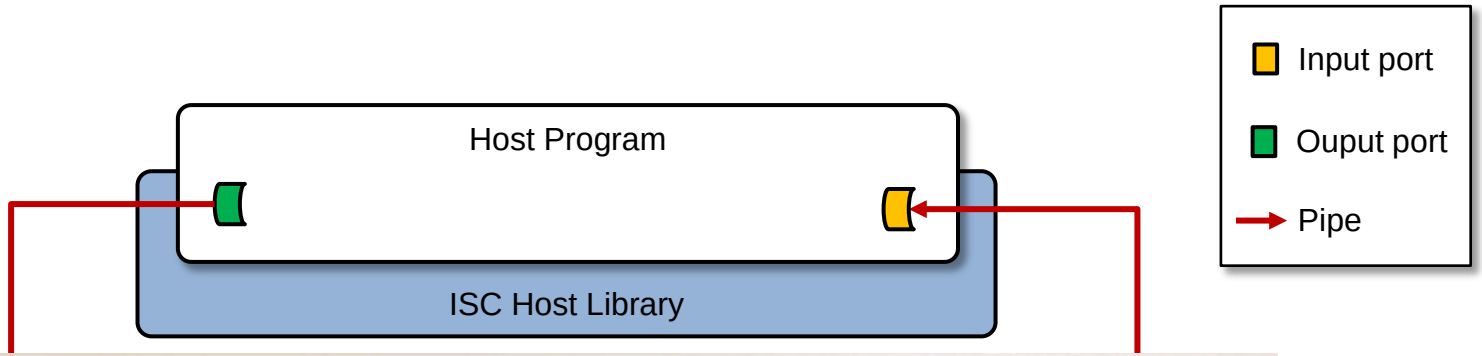
SSDlet

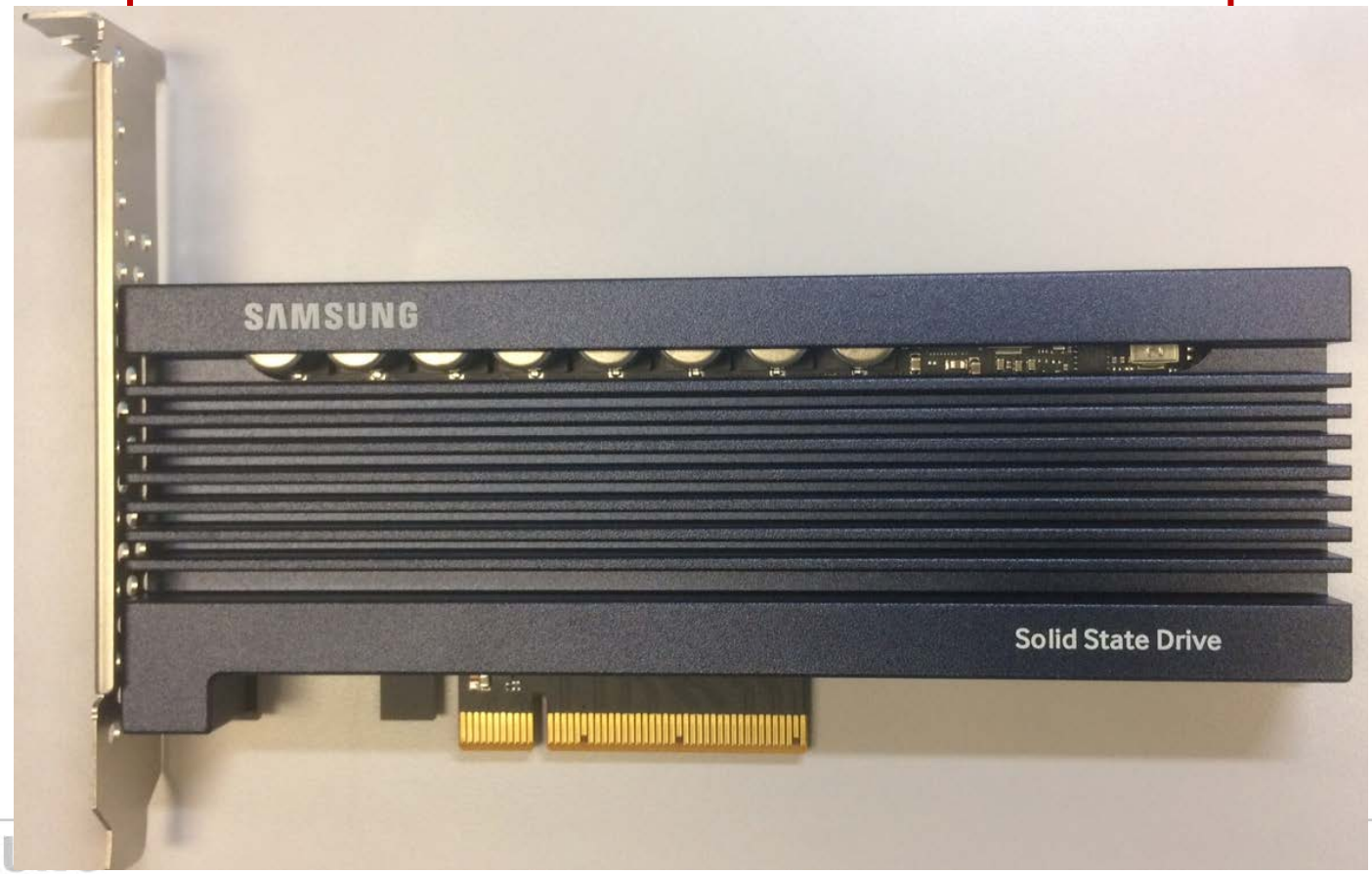
```
int main(int argc, char *argv[])
{
    SSD ssd("/dev/nvme0n1p1");
    module_id_t mid = ssd.loadModule(File(ssd, "./libkvstore.so");
    Application app(ssd);

    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;
}
```

Host App



ISC Runtime Framework





Samsung ISC SSD Prototype

Commodity SSD

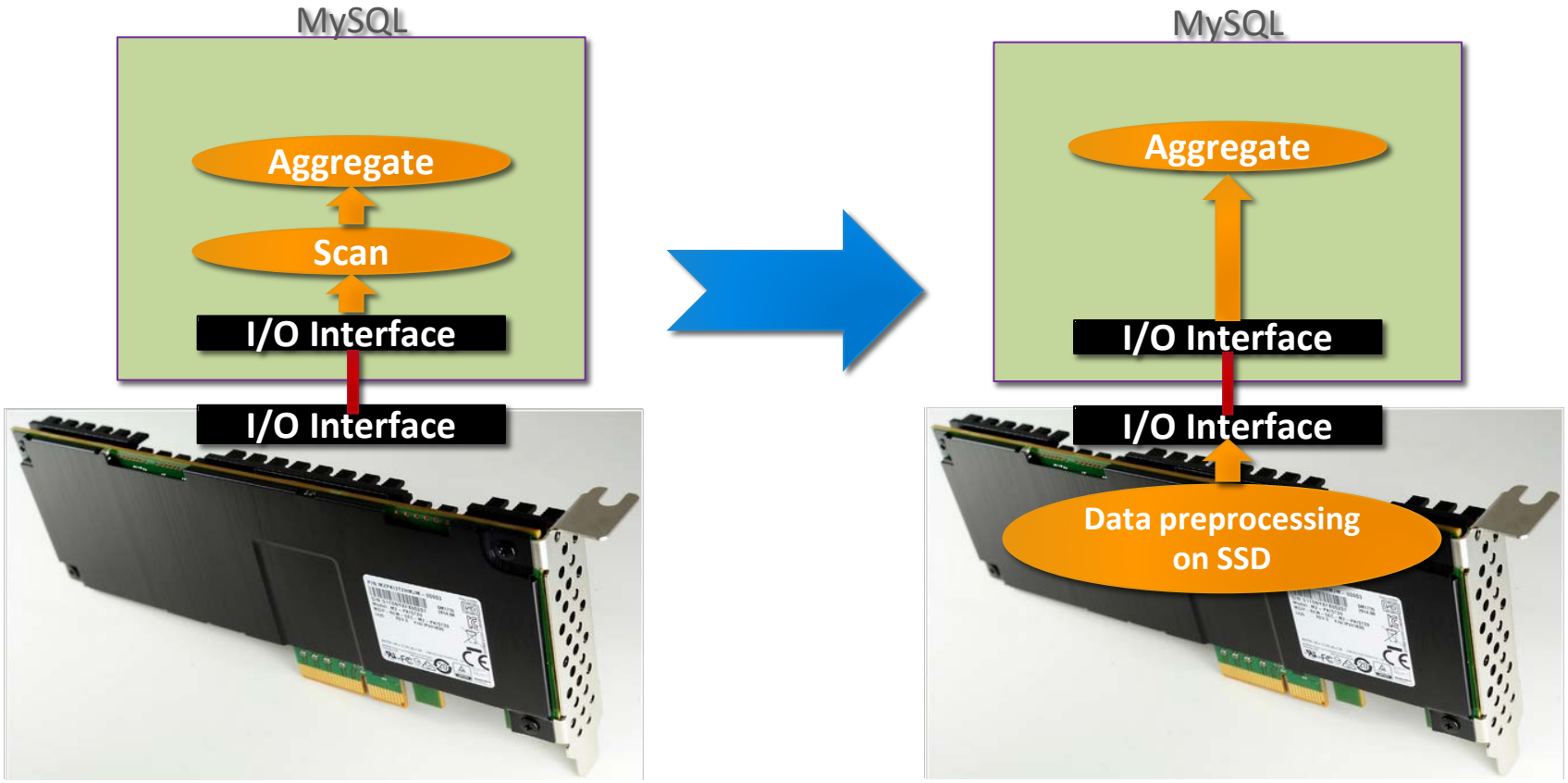
- Samsung PM1725 NVMe with the ISC feature
- PCIe 3.0 x4
- 800 GB

Software

- C++11
- C++ STL
- g++
- Software emulator

Case Study 1: Data Analytics with MySQL

- **MySQL determines data pages to fetch according to relevance hints from SSD**
  - MySQL gets relevance hints for pages in a given range all at once
  - Filter out access to pages with irrelevant data



Elapsed time of TPC-H query 2

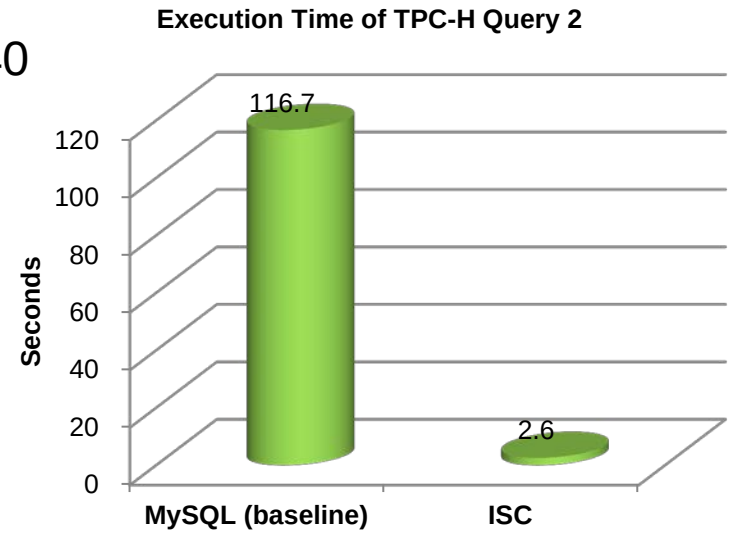
- An analytic query to find a minimum cost supplier

```
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 = 'EURO
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;
```

The most efficient plan is to put part table first in the join order and filter out its irrelevant pages!

- ISC reduces the query time to less than 1/40

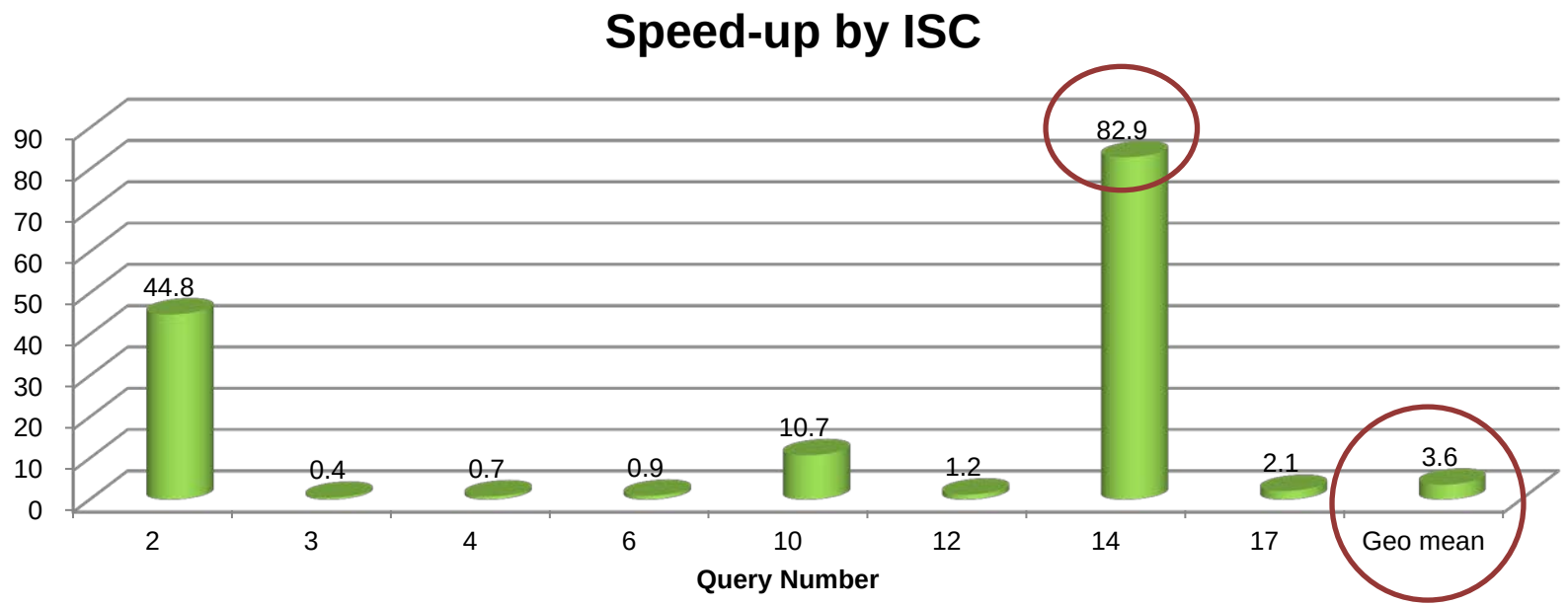
# of pages read w/ MySQL (baseline)	Table name	# of pages read w/ ISC
1,325,978	Total	**22,317**
325,386	Part	**7,525**
15,229	Supplier	**4,582**
985,354	Partsupp	**10,201**
5	Nation	5
4	Region	4



3.6X TPC-H Query Processing Speed Up

- **A representative TPC-H benchmark subset is expected to reveal over 3.6x performance gains**

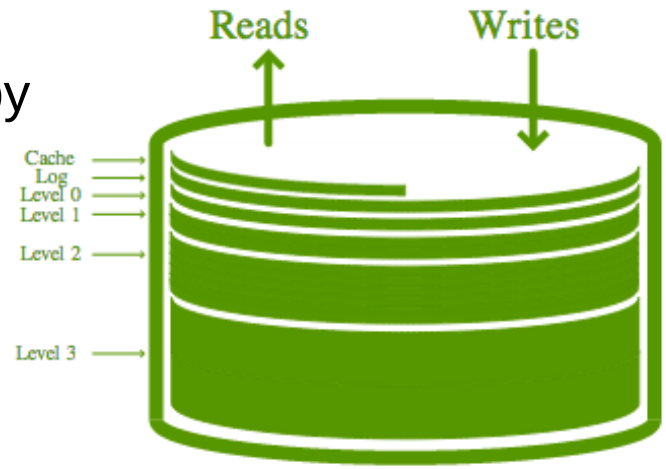
Host server	**Dell PowerEdge R720** **- Intel(R) Xeon(R) CPU E5-2640 0 @ 2.50GHz x2** **- 3G of DRAM** **- OS device: Samsung MZ-6ER100T SAS 100GB SSD** **- Data device: PM1725 480GB NVMe SSD (SR=3GB/s)**
OS	Ubuntu 15.04 (3.19.0 kernel)
Software	Mariadb-5.5.42 & TPC-H 2.17.0
TPC-H dataset	20G of dataset (with scale factor of 10)



Case Study 2: Storage Compaction

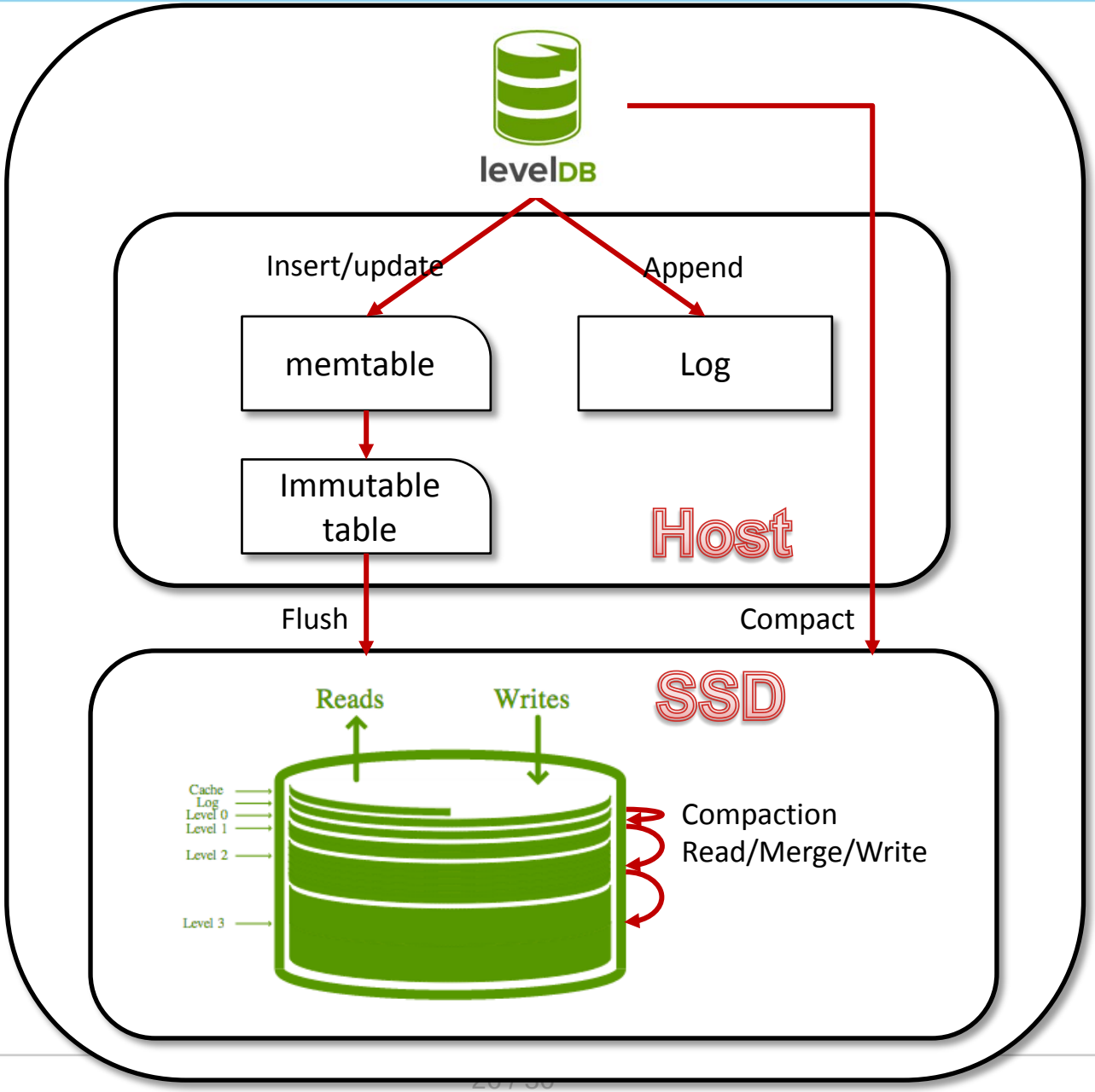
LevelDB

- One of popular embedded databases
- Open-source, embedded key/value store by Google
- Base database system for other open source projects
  - RocksDB (LevelDB+HBase), HyperLevelDB
  - Riak, Ceph storage backend

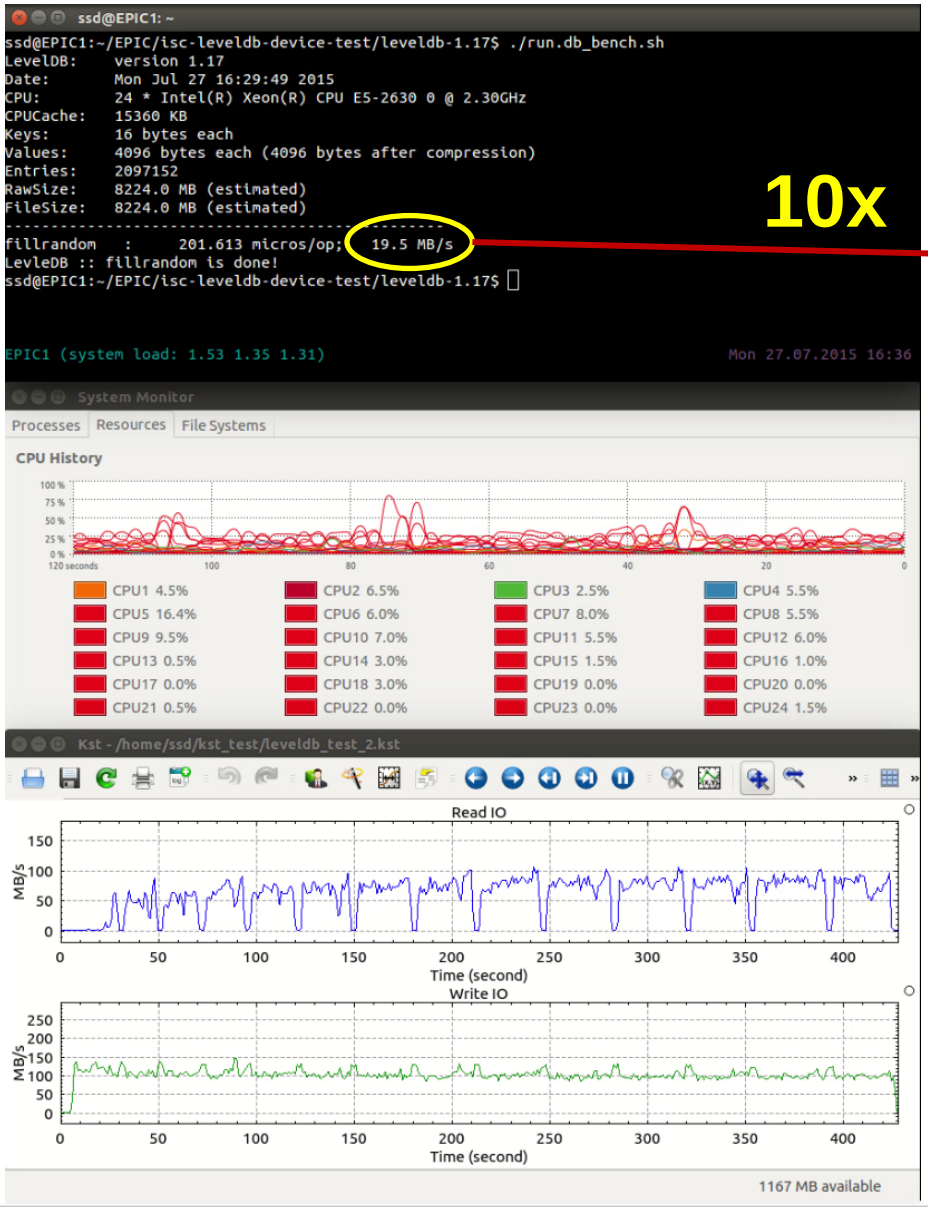


Log:	Max size of 4MB then flushed into a set of Level 0 SST files
Level 0:	Max of 4 SST files then one file compacted into Level 1
Level 1:	Max total size of 10MB then one file compacted into Level 2
Level 2:	Max total size of 10 x Level 1 then one file compacted into Level 3
Level 3+:	Max total size of 10 x previous level then one file compacted into next level

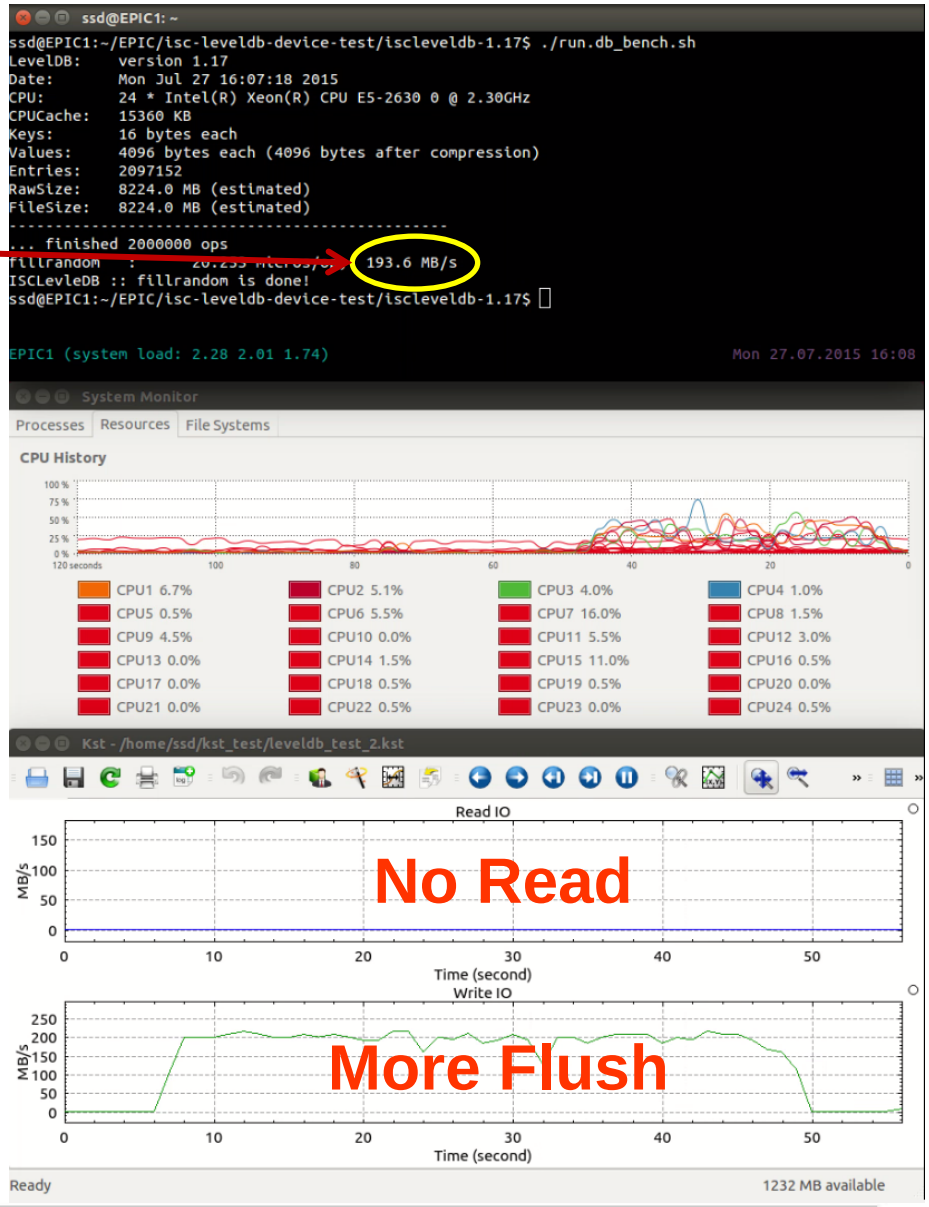
New LevelDB with Compaction Powered by ISC



Up to 10X Throughput Improvement



10x



Take-Away Messages

- **Computing paradigm shift from CPU-centric to data-centric for I/O intensive applications**
- **Samsung ISC realizes heterogeneous computing framework across general purpose CPU and SSD.**
- **IO intensive applications can benefit from low power high performance of embedded processors and high internal bandwidth of SSDs.**
- **Samsung ISC prototype**
  - ISC-aware MySQL achieves performance improvement up to 80x or 3.6x on average with TPC-H
  - ISC-aware LevelDB achieves up to 10x throughput improvement with dbbench (default benchmark)

Reference

[1] IDC, “Worldwide Business Analytics Software 2014–2018 Forecast and 2013 Vendor Shares,” Jul 2014.

[2] HUAWEI, “Accelerate Oracle Performance by Using ASM Preferred Read Failure Group with Dorado,” Sep 2012.

[3] HDFS Architecture Guide, https://hadoop.apache.org/docs/r1.2.1/hdfs_design.html

[4] Devesh Tiwari et al., “Reducing Data Movement Costs using Energy-Efficient, Active Computation on SSD,” HotPower’12, 2012.

[5] Prabhat and Quincey Koziol, “High Performance Parallel I/O,” CRC Press book, Oct 2014.

Meet our engineers at booth 307
4-7PM (Tue), 12-7PM (Wed), 10:30AM-2PM (Thu)

Thank You!

yangseok.ki@samsung.com