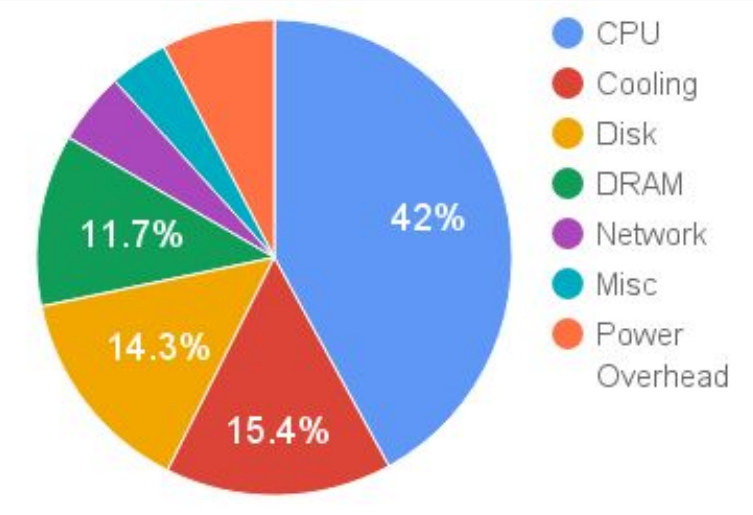
A Multilevel NOSQL cache architecture combining in-NIC cache and in-kernel cache

Yuta Tokusashi @Keio University, Japan

Green Computing in Data Centers

- Data centers accommodate **1k ~ 100k** servers
  - CPU and cooling consume more than half of the entire power consumption [Barroso]
- In-memory data store
  - Widely used in Social Networking Service, such as Facebook, Twitter, Wikipedia, etc.
  - Parameter server for machine learning



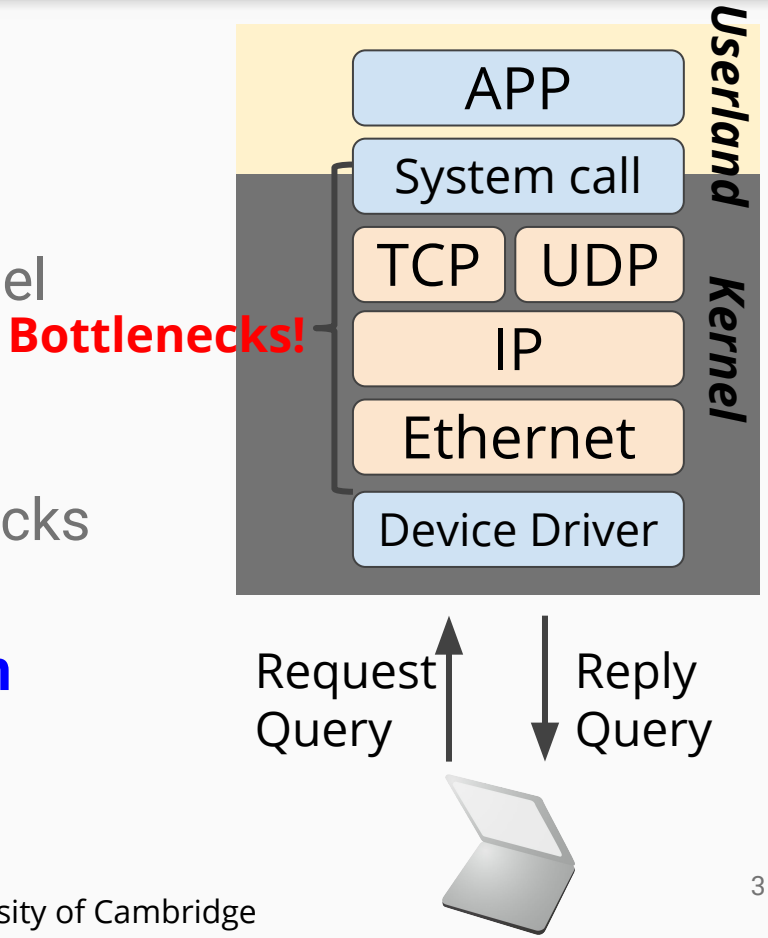
Power consumption in data center

Energy efficiency in data store should be improved !!

[Barroso] L. A. Barroso, et al, "The Datacenter as a Computer", 2nd ed. Morgan & Claypool Publishers, Jul. 2013

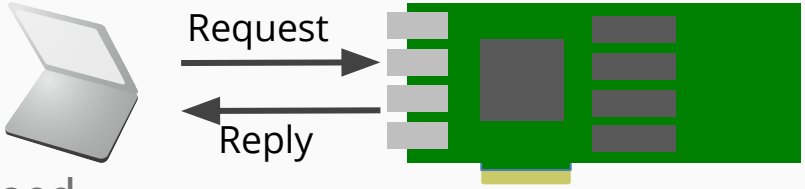
What is Bottleneck on KVS ?

- Three bottlenecks
  - Network protocol stack on kernel
  - Memory copy
  - System calls
- Two approaches for these bottlenecks
  - **In-NIC Processing Approach**
  - **In-Kernel Processing Approach**



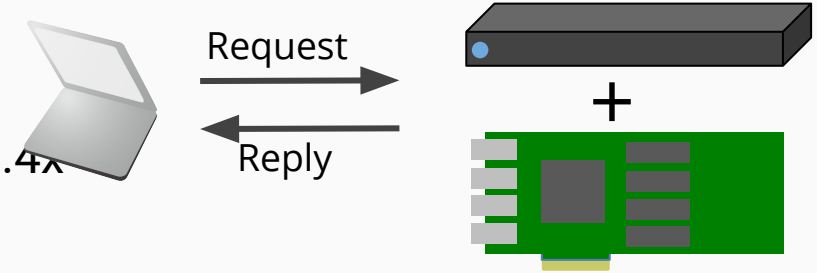
In-NIC Processing Approach

- Offloading data store on Network Interface Card
  - KVS processing core on FPGA or ASIC
  - Some sophisticated functions are processed on CPU



Standalone System [Blott]

- High energy efficiency (Perf./Watt)
  - Processing KVS in 10G line rate [Blott]
  - Performance per Watt is improved by 36.4x compared to Xeon server



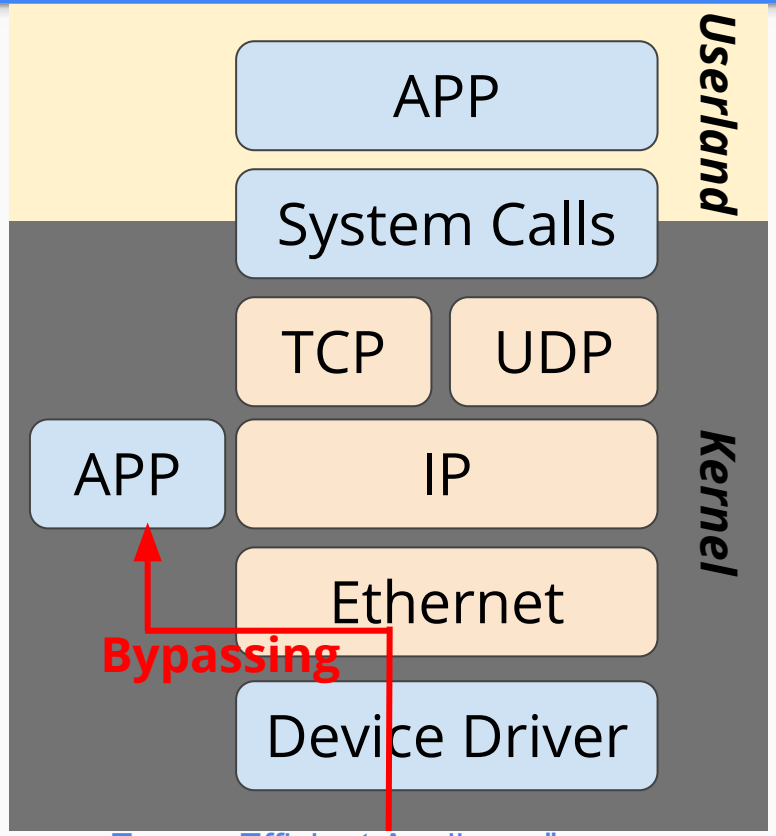
Host CPU + FPGA(or ASIC) System [Lim]

- Small cache capacity
  - On-board DRAM capacity is limited
  - NetFPGA-10G : 288MB, NetFPGA-SUME: 8GB

[Blott] M. Blott, et al, “Achieving 10Gbps Line-rate Key-value Stores with FPGAs”, HotCloud’13
[Lim] K. Lim, et al, “Thin Servers with Smart Pipes: Designing SoC Accelerators for Memcached”, ISCA’13

In-Kernel Processing Approach

- Reducing overhead related to networking and system calls
  - In-kernel processing for KVS
  - netfilter
- Huge host memory is used for cache
  - Latest mother board can equip a few TB memory
- Moderate perf./ Watt improvement
  - 3.0 M operations per second [Xu]
  - CPU consumes much power



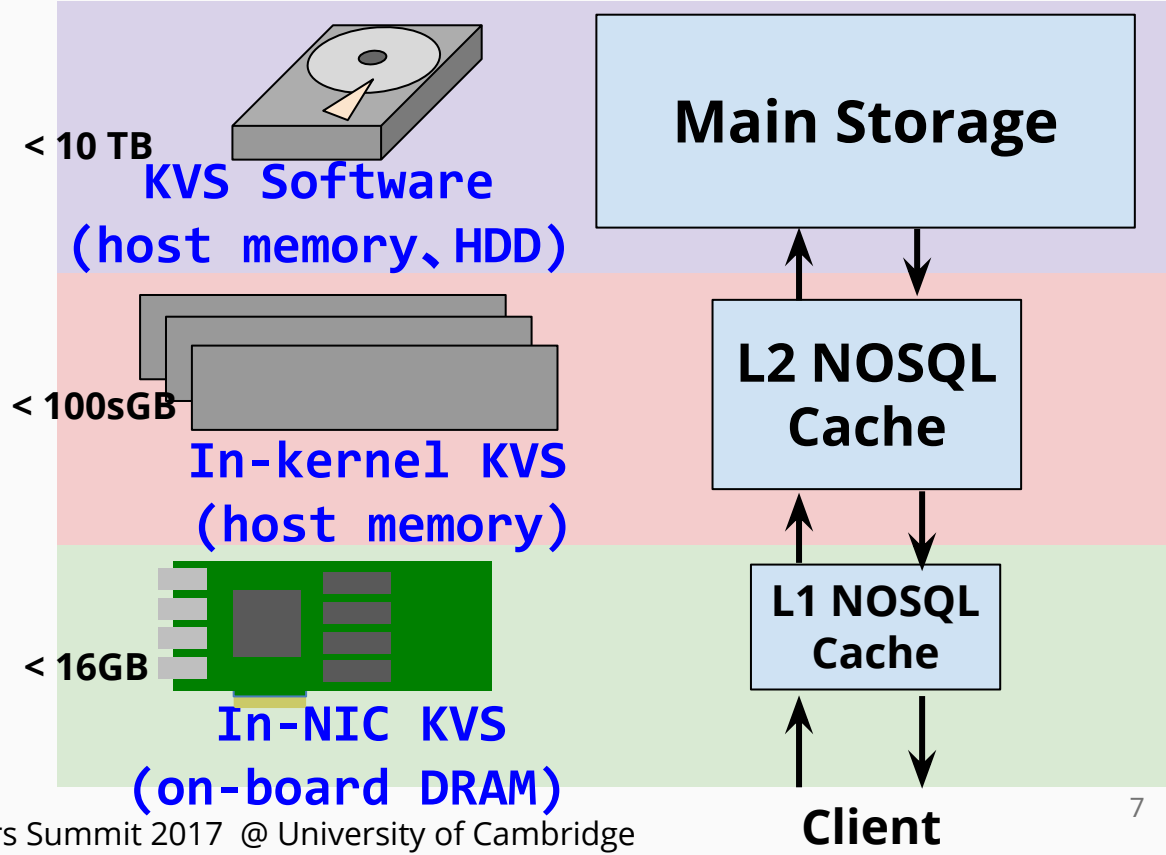
[Xu] Y. Xu, et al, “Building a High-Performance Key-Value Cache as an Energy Efficient Appliance”, Performance Evaluation 2014.

Trade-off between HW/SW approaches

	Cache Capacity	Energy Efficiency (Perf./Watt)
In-NIC Cache [Blott]	✖ Limited by on-board DRAM capacity **< 16 GB**	✔ High Perf/Watt
In-Kernel Cache [Xu]	✔ Main memory is used as storage **< Several 100s GB**	✖ CPU consumes high power, so drastic Perf./Watt improvement is difficult

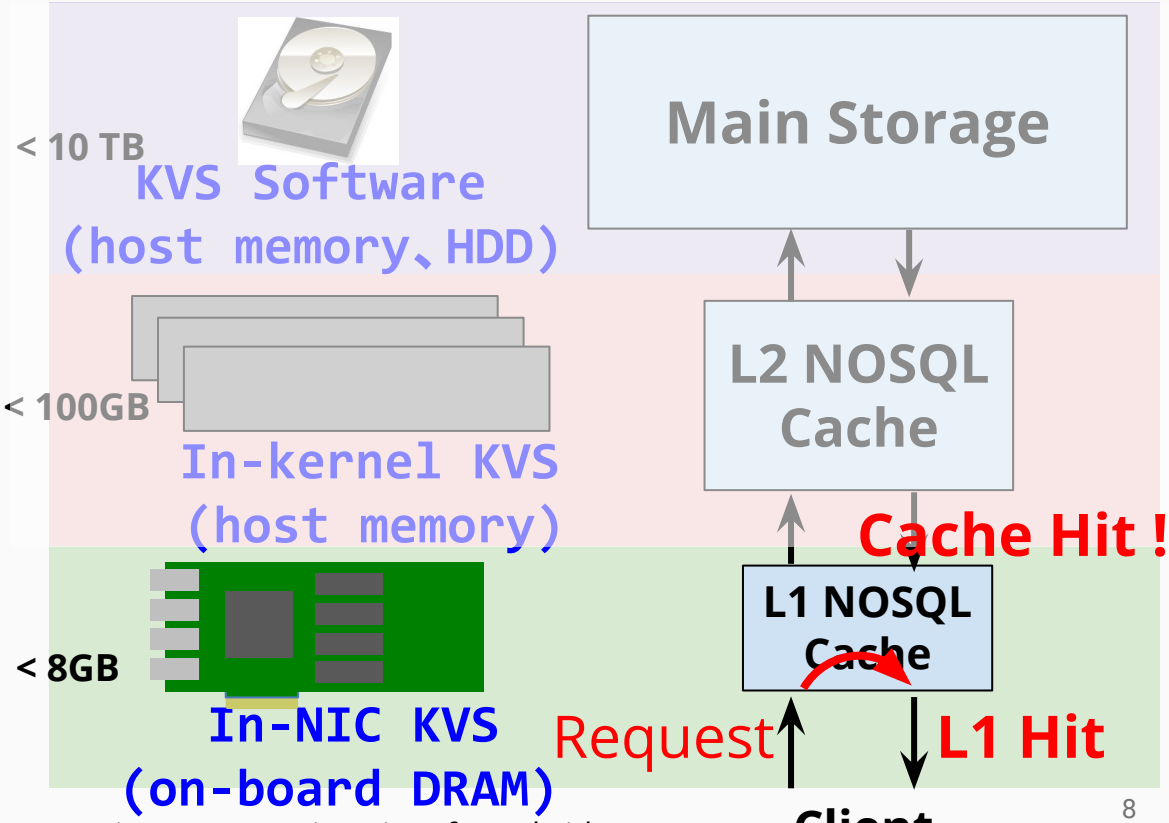
A Multilevel NOSQL Cache Design

- **A Multilevel NOSQL cache design**
  - Level 1 NOSQL cache
    - In-NIC processing approach
  - Level 2 NOSQL cache
    - In-kernel processing approach
  - User Space
    - General Keyvalue Store application



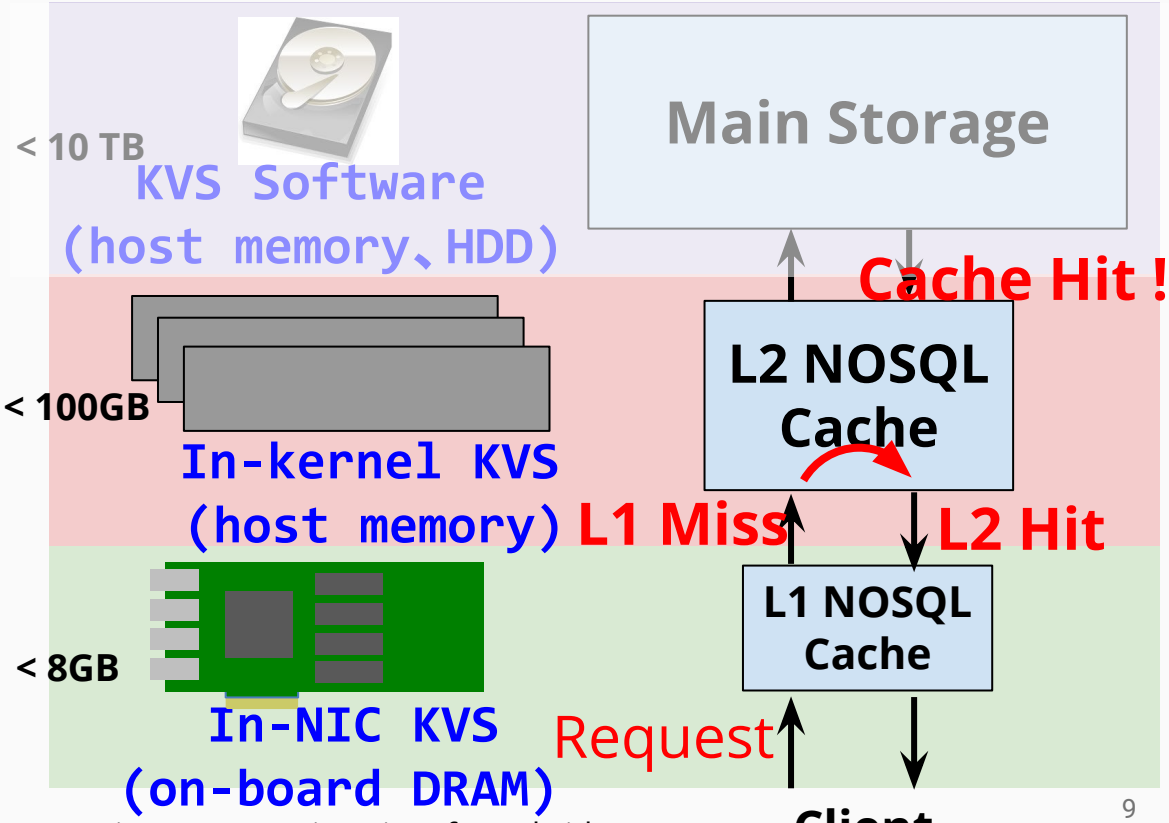
A Multilevel NOSQL Cache Design

When query hits in L1 NOSQL cache



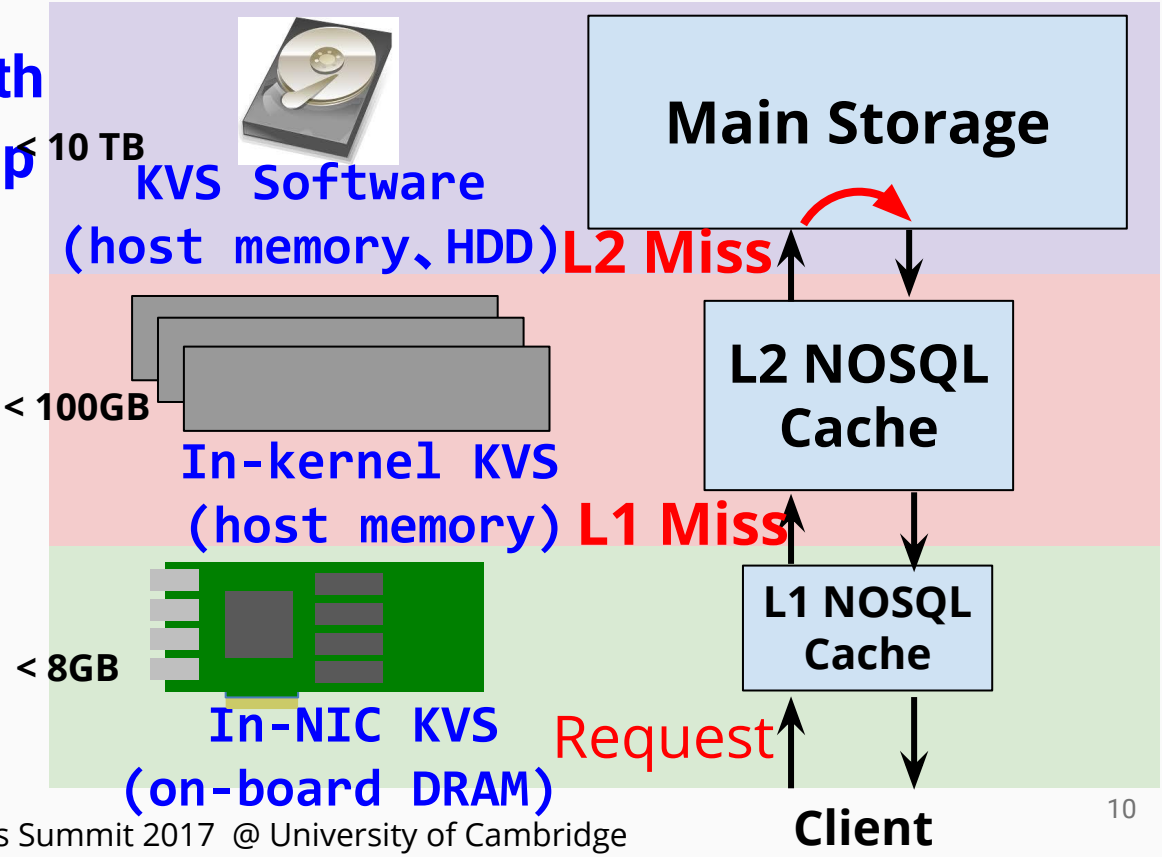
A Multilevel NOSQL Cache Design

When query hits in L2 NOSQL cache



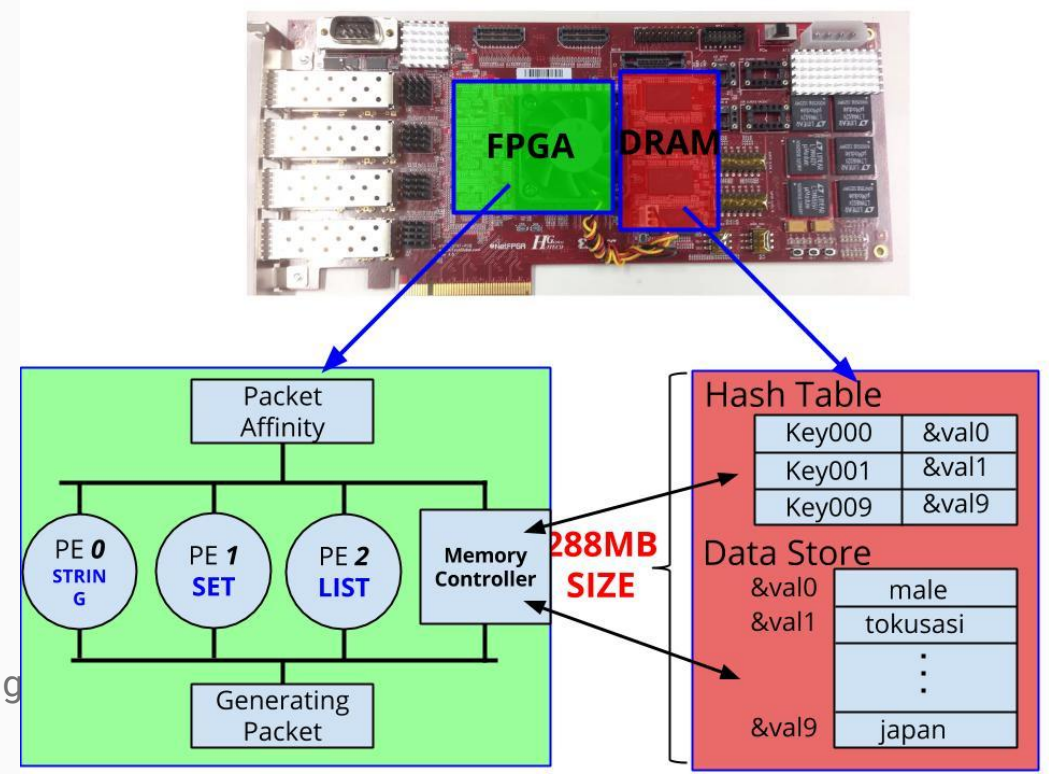
A Multilevel NOSQL Cache Design

When query misses in both caches, the query looks up main storage



Design and Implementation of L1 NOSQL cache

- Heterogeneous Multi-PE design
  - PEs per data structures
  - STRING, LIST, SET and HASH are supported
  - PEs and DRAM are connected via crossbar switch
- Target board: NetFPGA-10G
  - Virtex-5 XC4VTX240T
  - RLDRAM-II 288MB (NetFPGA-SUME : 8GB)
- PE performance
  - 1 PE has 1.4Mops
  - 10GbE line rate is achieved by integrating multiple PEs
  - Requires 7~9 PEs for 10GbE line rate



[Tokusashi] Yuta Tokusashi, Hiroki Matsutani, "NOSQL Hardware Appliance with Multiple Data Structures", Hot Chips'16, Aug 2016.

Result of implementation in L1, L2 NOSQL cache

In-NIC cache (L1 NOSQL Cache)

- Synthesis (Virtex-5 XC5VT240T)
  - Xilinx ISE13.4
  - Slice utilization 52% (NIC 48%)
  - Maximum frequency 160 MHz
- Throughput
  - SET 0.7 Mops (11% of 10GbE line rate)
  - GET 1.4 Mops (14% of 10GbE line rate)
- Power : approx. 30W (FPGA only)

In-Kernel cache (L2 NOSQL cache)

- Using Netfilter framework for KVS processing in linux kernel
- Throughput : around 3.0 Mops
- Power : around 50W

Memcached

- Throughput : 0.67 Mops (4 core)
- Power : around 68W

CPU	Intel Core i5-4590
Memory	4GB
OS	CentOS 6.7 (Linux Kernel 2.6)
NIC	NetFPGA-10G

NOSQL Server Machine Specification

Design Options for NOSQL Cache

- Inclusion / Non-inclusion
- Write-back / Write-through
- Cache Associativity
- Slab Configuration

Five memcached traces

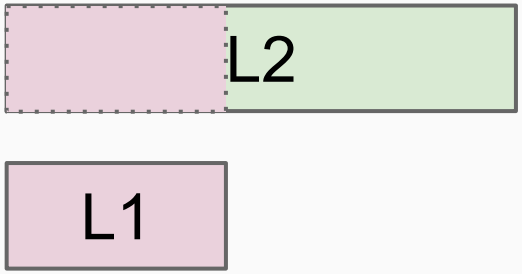
Trace Type	Disctiption
USR	Account Information
SYS	Server Information
APP	Application Meta Data
VAR	Browsing Data
ETC	General Porpose

- Simulation with memcached workload [Atikoglu]
  - Memcached workload emulation environment is built based on workload analysis [Atikoglue]
  - These design options are evaluated in terms of cache miss ratio

[Atikoglu] Berk Atikoglu, et al, Workload Analysis of a Large-Scale Key-Value Store, SIGMETRICS'12

Inclusive vs. Non-inclusive Policy

Inclusion



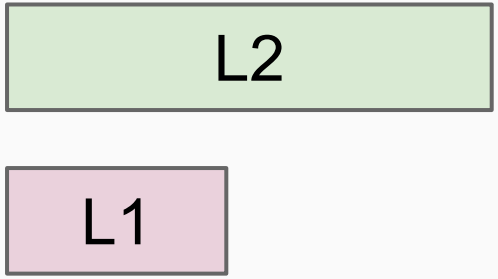
Advantages

- Engineering cost is small
- Consistency is kept

Disadvantages

- When “L1/L2” ratio is small, L2 NOSQL cache is not effective

Non- inclusion



Advantages

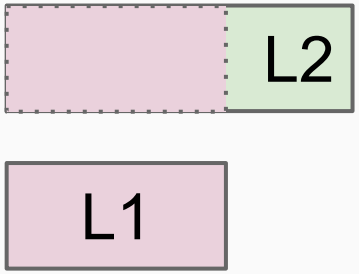
- Efficient use of capacity

Disadvantages

- Engineering cost is high
- Data consistency is not kept

Inclusive vs. Non-inclusive Policy

Inclusion



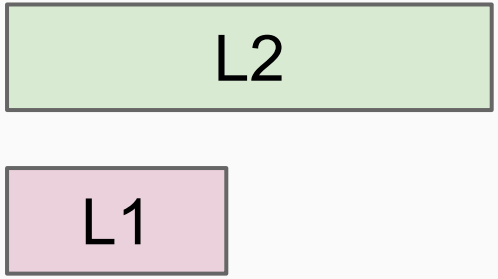
Advantages

- Engineering cost is small
- Consistency is kept

Disadvantages

- When "L1/L2" ratio is small, L2 NOSQL cache is not effective

Non- inclusion



Advantages

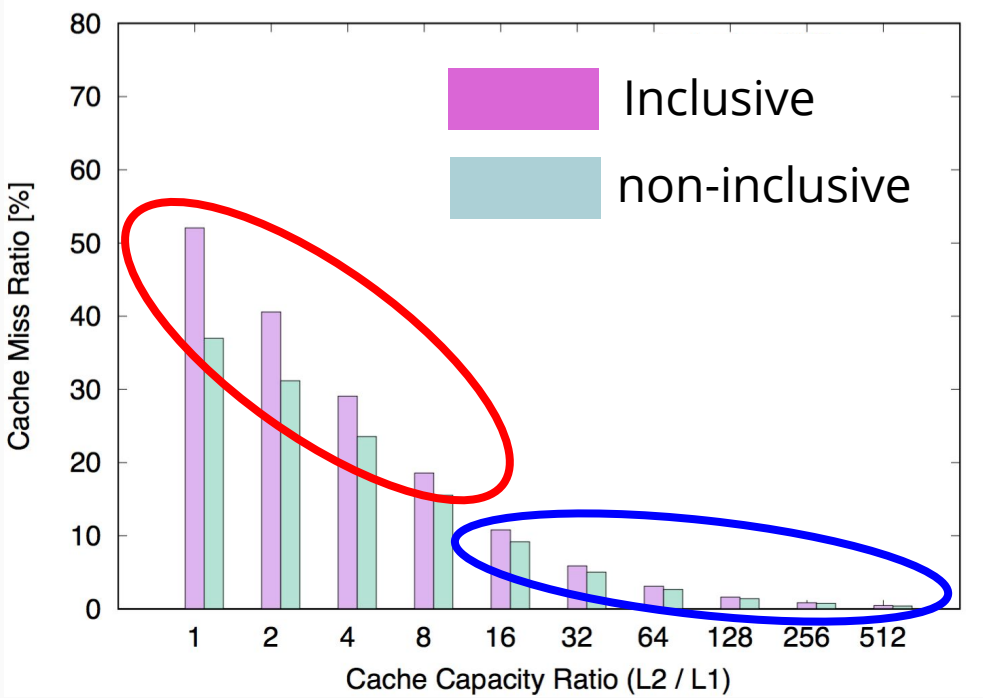
- Efficient use of capacity

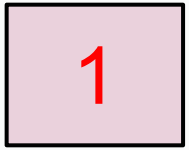
Disadvantages

- Engineering cost is high
- Data consistency is not kept

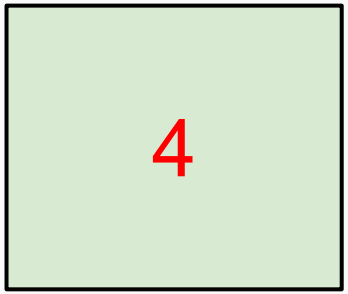
Inclusive and Non-inclusive Options

- When capacity ratio is low, **non-inclusive can reduce miss ratio**
- When capacity ratio is high, **Inclusive policy is proper due to simple cache hierarchy design**





L1 NOSQL cache

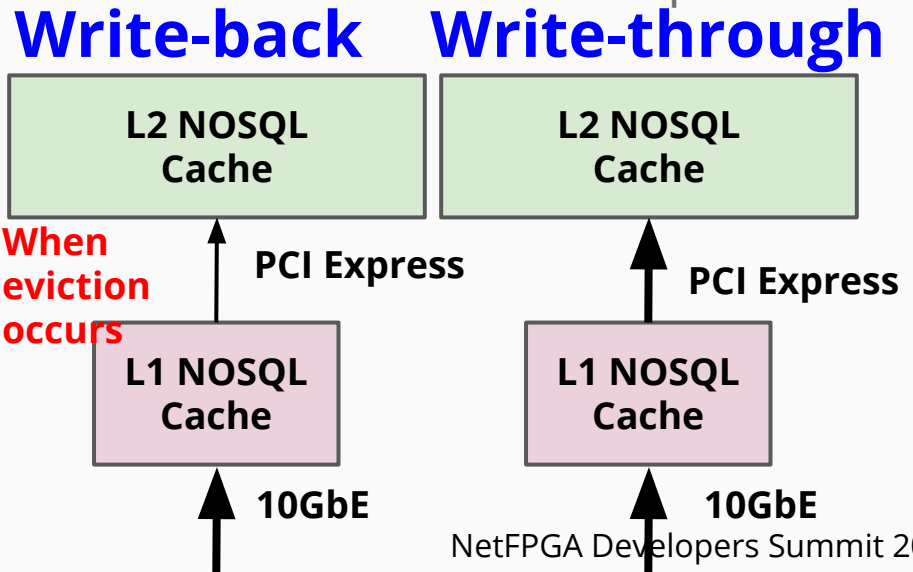


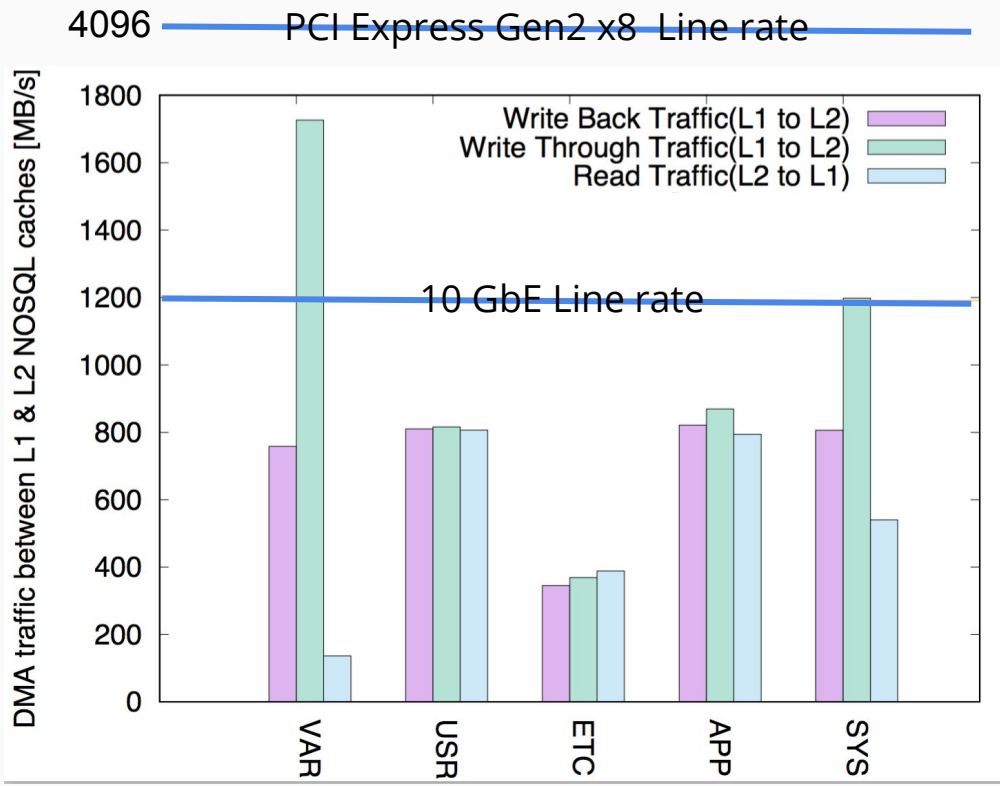
L2 NOSQL cache

L1 is Larger ⟷ L1 is small

Write-back vs. Write-through

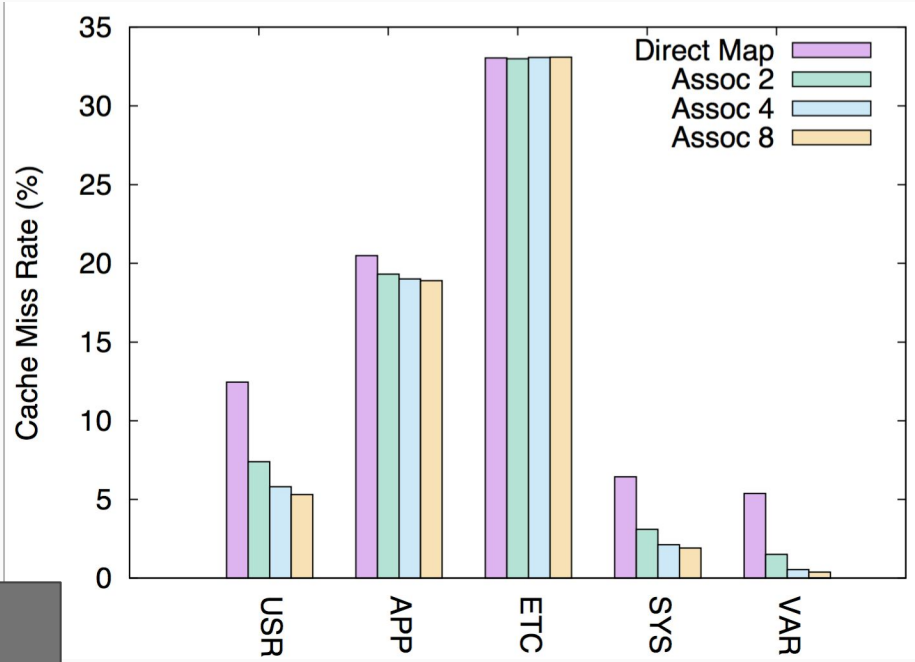
- 10GbE NIC throughput is bottleneck
- VAR and SYS throughputs are limited by network interface
- Write-back is a better option





Cache Associativity

- Hash conflict occurs
  - Conflict misses can be mitigated by increasing the cache associativity
- Increasing cache associativity to 1, 2, 4 and 8
  - USR, SYS and VAR can reduce cache miss ratio by 5%
  - In APP and ETC, it is not improved so much



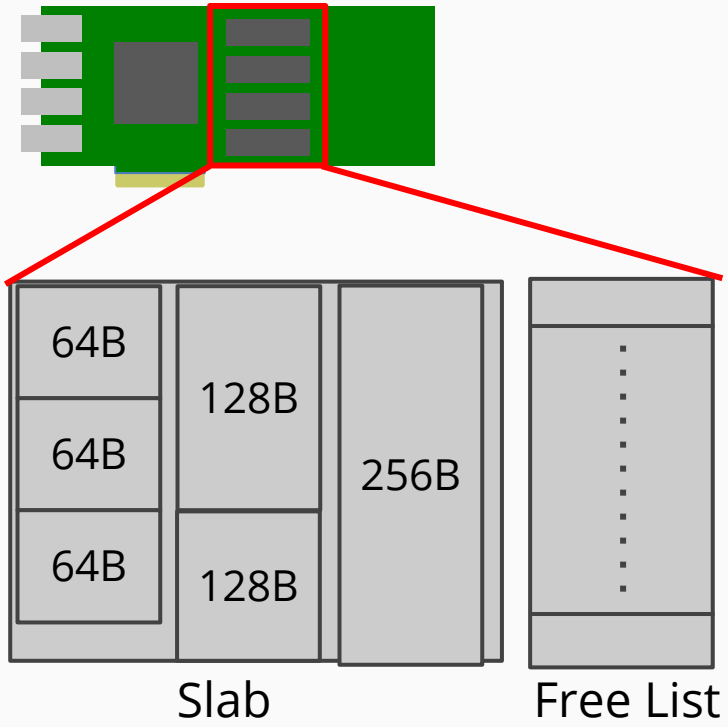
Associativity vs. cache miss ratio

Associativity : 4

Index				
0	Key16	Key32	Key0	Key8
1	key41	key9		
:				

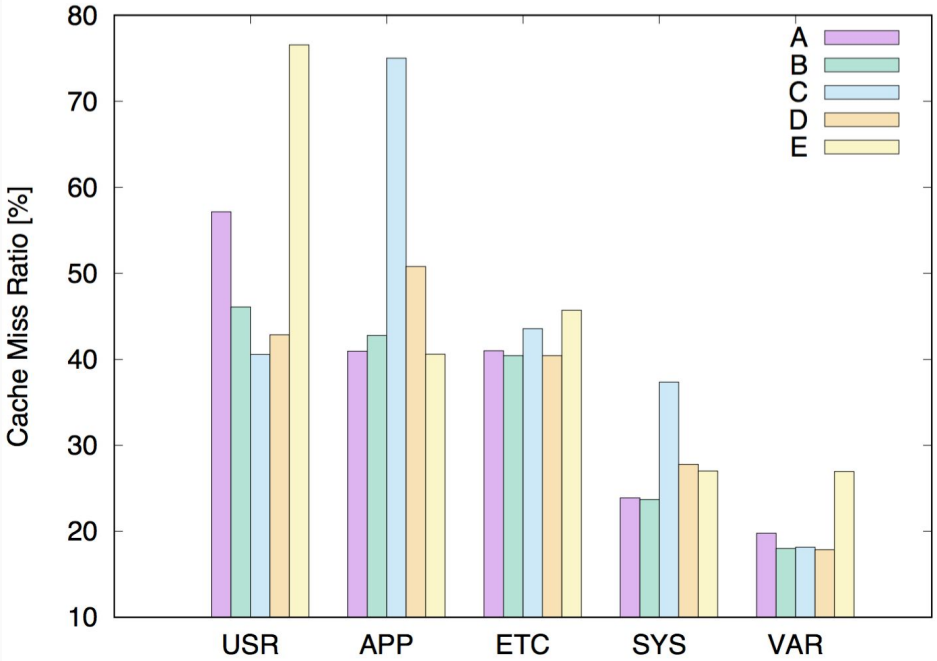
Slab Allocation in L1 NOSQL cache

- Slab Allocator
  - To support Variable value length
  - Chunks per size (e.g. 64B, 128B, etc.)
  - Using soft-CPU on FPGA
  - Free List manages unused chunks.
- Static allocation
  - Allocated chunk in boot time.
  - User can custom setting the number of chunk and chunk size.
  - Chunk setting is critical against workload



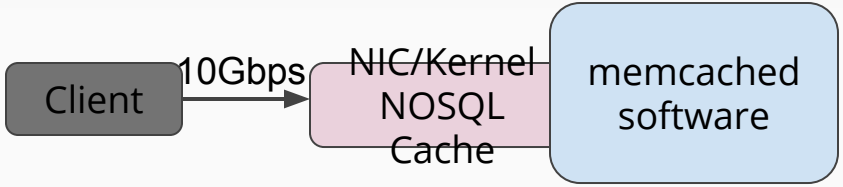
Slab Configurations

- L1 NOSQL cache has limited cache capacity (< 8GB)
  - Chunk configuration (sizes of chunks and their numbers) should be customized for workload
- Each workload has unique characteristic in value size
  - Configurations A-E are examined (see Table)
- **USR, APP, and SYS can reduce cache miss ratio by configurations A, E, and B**

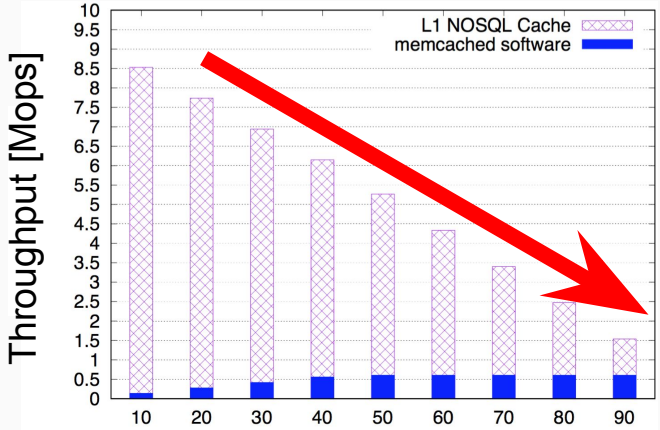


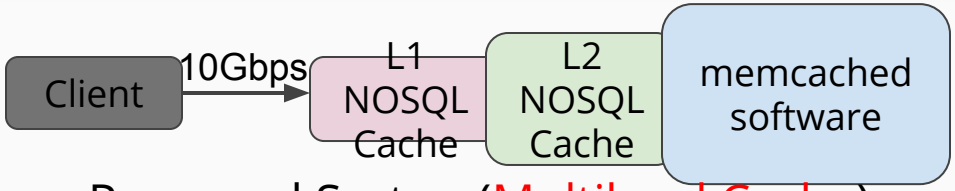
Type	64B	128B	256B	512B	1kB	2kB	Note
A	70k	70k	70k	70k	70k	70k	Uniform
B	120k	100k	80k	60k	40k	20k	More small sizes
C	160k	140k	120k	0	0	0	No large sizes
D	140k	120k	100k	30k	20k	10k	More small sizes
E	20k	40k	60k	80k	100k	120k	More large sizes

Cache Miss Ratio and Throughput Improvement

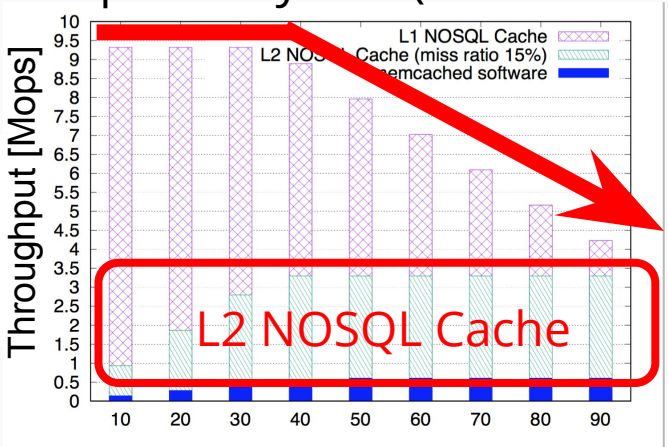


Conventional System(In-NIC Cache Only)





Proposed System(Multilevel Cache)



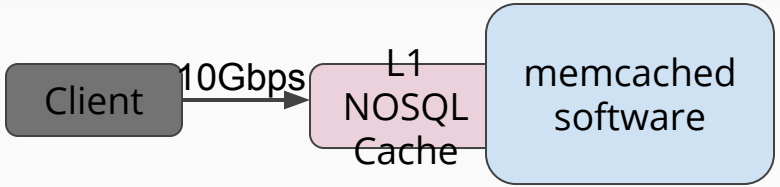
L1 NOSQL Cache

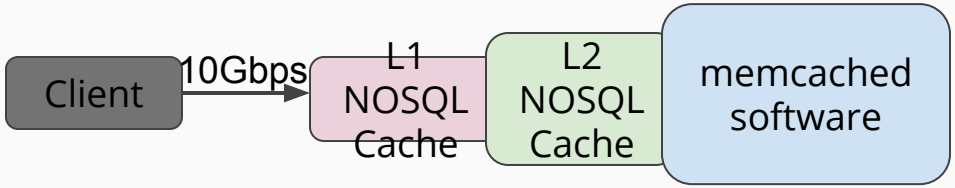
L2 NOSQL Cache

NOSQL APP

Multilevel NOSQL cache can improve throughput by reducing the cache miss

Conclusion





- Multilevel NOSQL Cache
  - L1 NOSQL cache : In-NIC cache
  - L2 NOSQL cache : In-kernel cache
  - Userland : NOSQL application
- Design Exploration on NOSQL cache hierarchy
  - write back vs. write-through
  - Inclusive vs. non-inclusive
  - Etc…
- Multilevel NOSQL cache (L1 + L2 NOSQL caches) improves entire cache miss ratio and throughput

➡ **This paper will be the first guideline for multilevel NOSQL cache design**

Thank you !