

Open Networking Korea (ONK) 2020

“Soft Networking for Untact age”

- 소속 : Carnegie Mellon University (CMU)
- 성명 : 김대혁
- 직위 : 박사과정 연구원
- 이메일 : daehyeok@cs.cmu.edu
- 발표 제목 : Enabling State-Intensive Network Functions on Programmable Switches



Network functions in the network

Network functions (NFs) are an essential component

- E.g., Load balancer, Firewall, Network address translator (NAT), ...

NF performance and scalability are key challenges



Approaches to deploying network functions



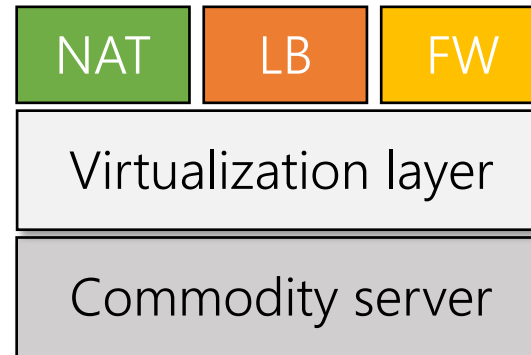
Standalone hardware

Fixed-function

Performance: $O(10 \text{ Gbps})$

Memory: $O(10\text{GB})$ DRAM

Price: $> \$40\text{K}$



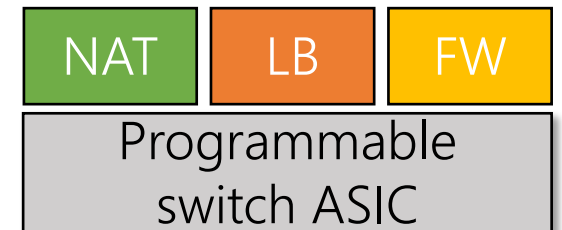
Server-based Software (NFV)

Programmable

Performance: $O(10 \text{ Gbps})$

Memory: $O(10\text{GB})$ DRAM

Price: $\$3\text{K}$



Switch-based NF

Programmable

Performance: $O(1 \text{ Tbps})$

Memory: $O(10\text{MB})$ SRAM

Price: $\$10\text{K}$



Problem: serving demanding workloads

None of the options can efficiently serve demanding workloads!

- Millions of concurrent flows ($O(100MB)$) + high traffic rate ($> 1 \text{ Tbps}$)



Standalone hardware

Fixed-function

Performance: $O(10 \text{ Gbps})$

Memory: $O(10GB)$ DRAM

Price: $> \$40K$

Server-based Software (NFV)

Programmable

Performance: $O(10 \text{ Gbps})$

Memory: $O(10GB)$ DRAM

Price: **\$3K**



Switch-based NF

Programmable

Performance: **$O(1 \text{ Tbps})$**

Memory: **$O(10MB)$ SRAM**

Price: $\$10K$



Root cause: limited on-chip SRAM

Limited on-chip SRAM space: $O(10\text{MB})$

Infeasible to maintain large flow state within on-chip SRAM

- E.g., LB state for 10M flows requires $\approx 100\text{MB}$

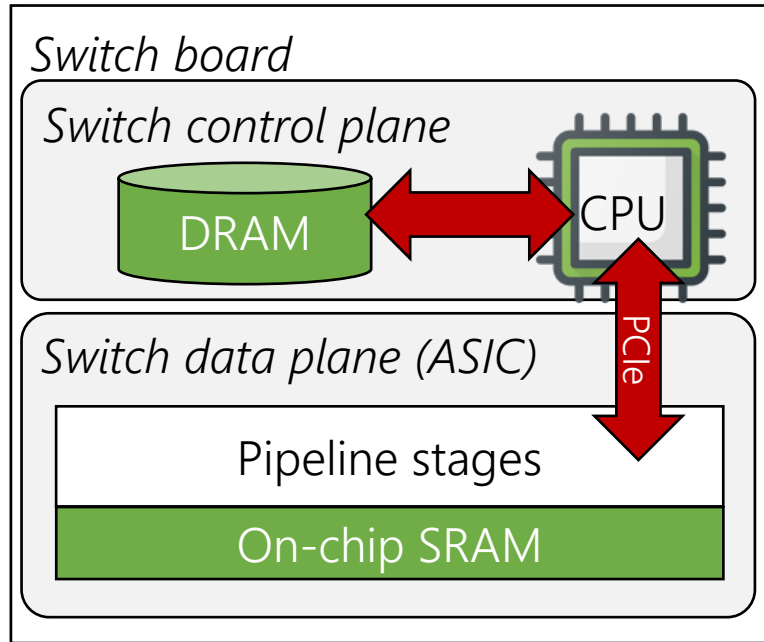
Adding more SRAM would be too expensive ☹️

Can we leverage larger and cheaper DRAM near switch ASICs?

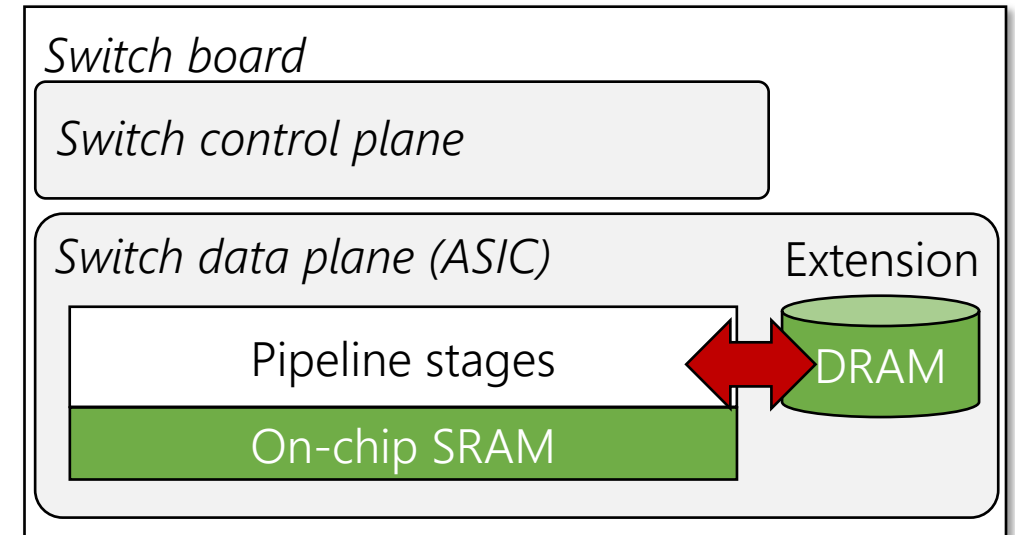


DRAM available on a switch board

Option #1:
DRAM on the switch control plane



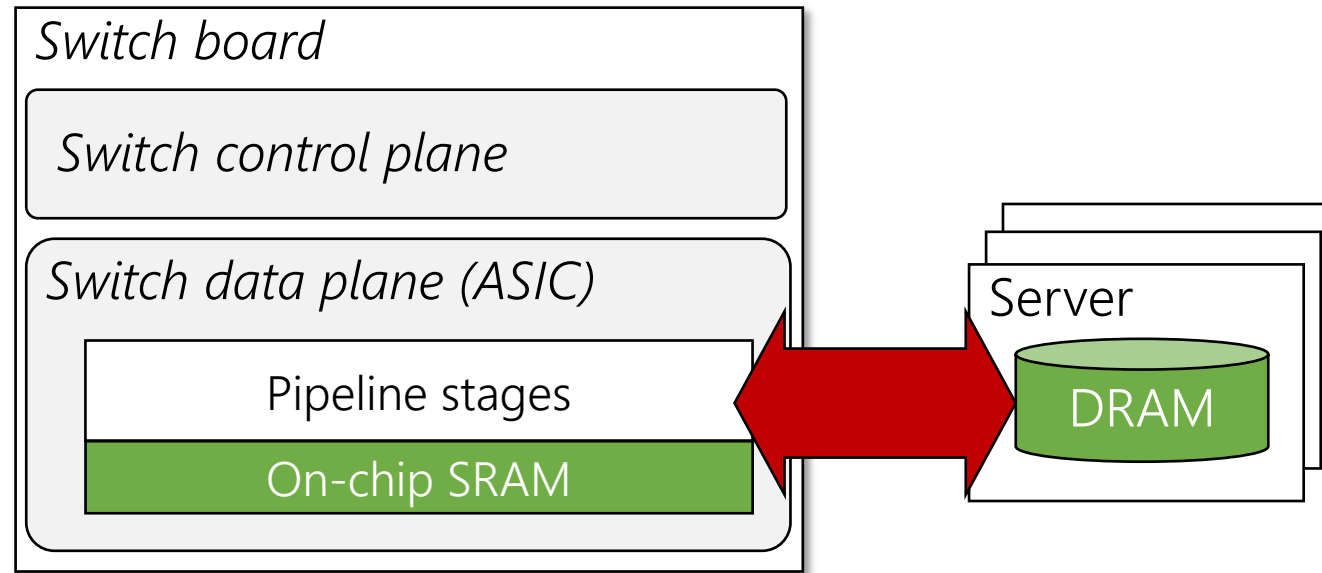
Option #2:
On-board, off-chip DRAM



- Limited scalability in terms of size and access bandwidth
- High cost

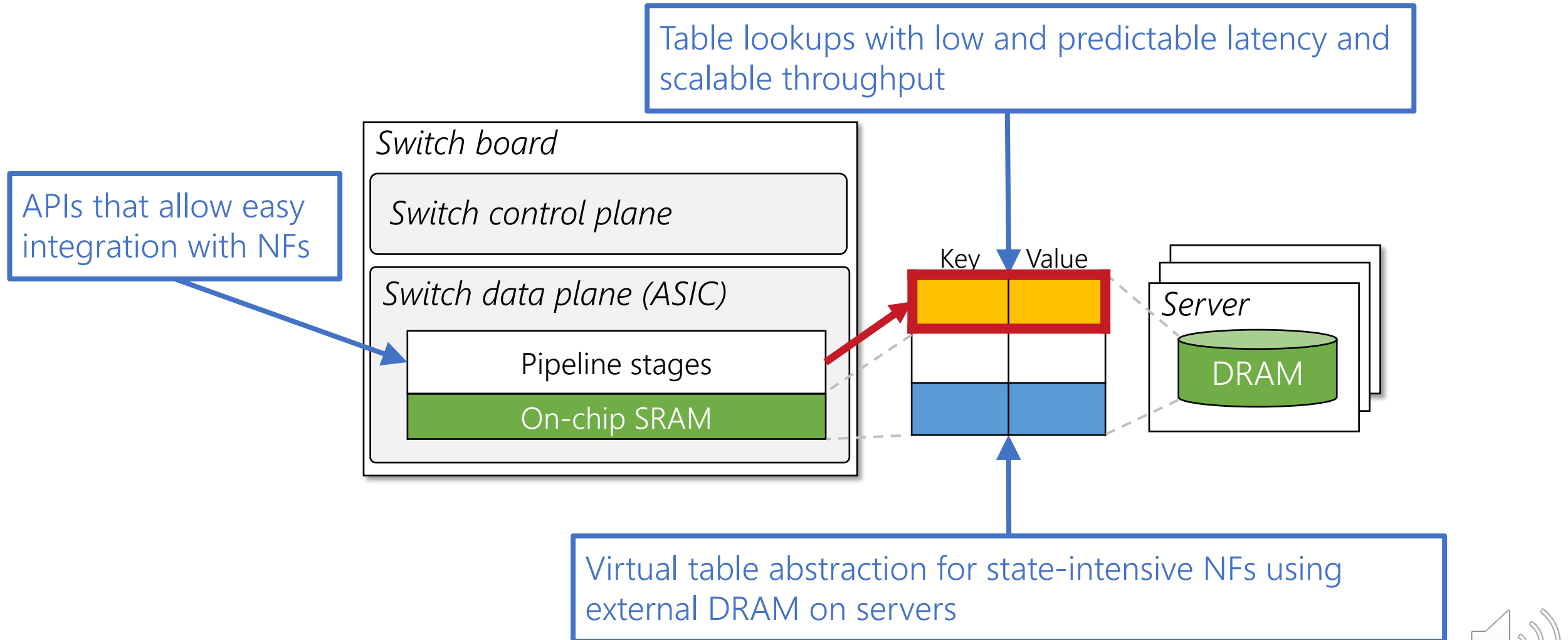


Opportunity: DRAM on commodity servers



- + Scalable memory size and bandwidth
- + Low cost

Table Extension Architecture (TEA)



Outline

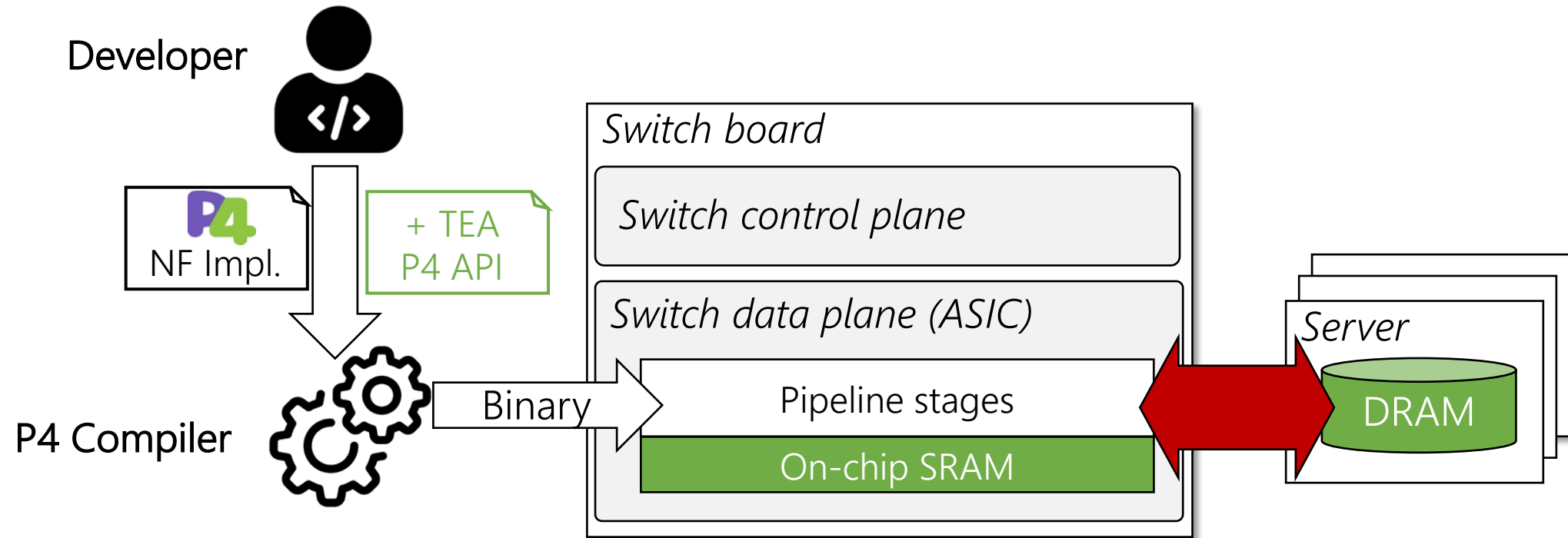
Motivation

TEA design

Results

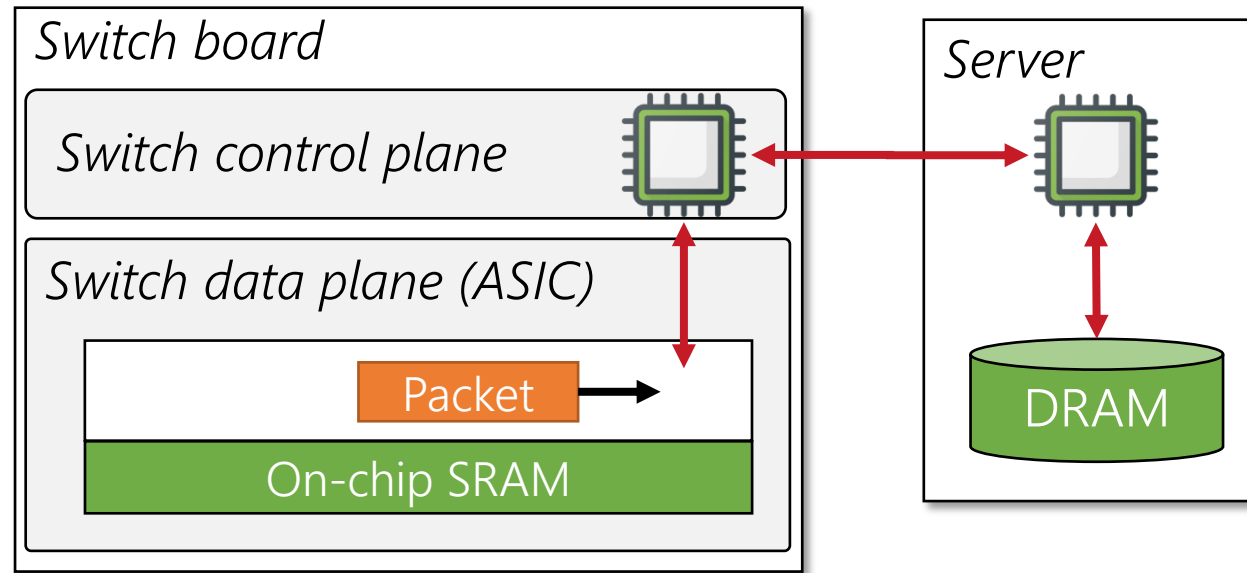


TEA design overview



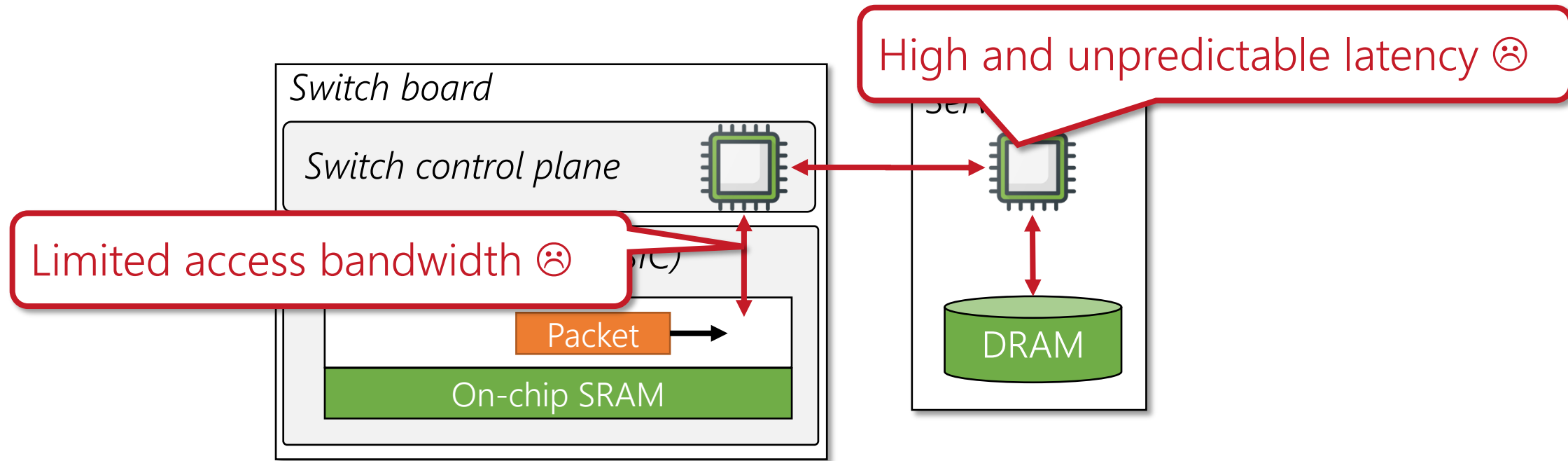
Challenge 1: Enabling external DRAM access from switch ASIC

Strawman: accessing external DRAM via the control plane



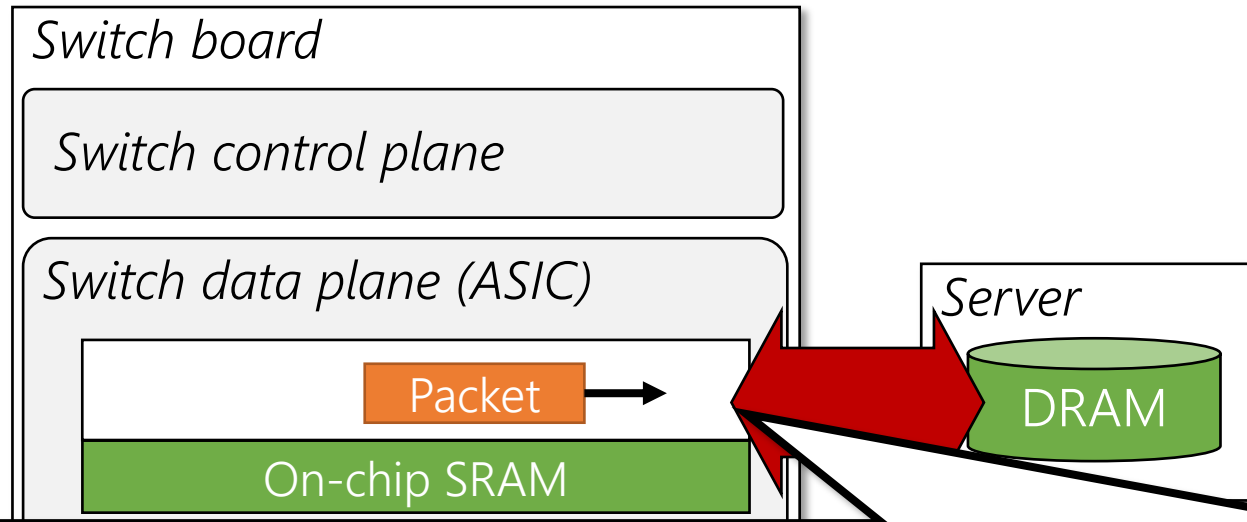
Challenge 1: Enabling external DRAM access from switch ASIC

Strawman: accessing external DRAM via the control plane



Challenge 1:

Enabling external DRAM access from switch ASIC

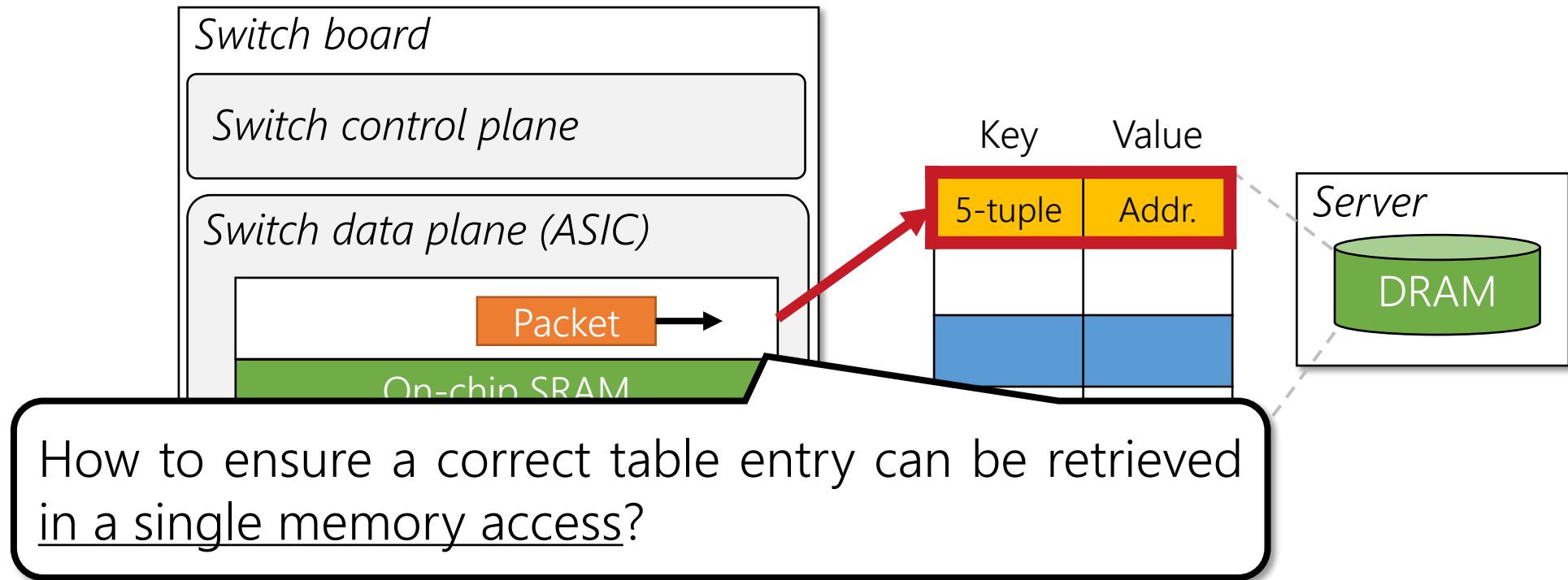


How to enable switch ASIC to directly access external DRAM without CPUs and hardware modifications?

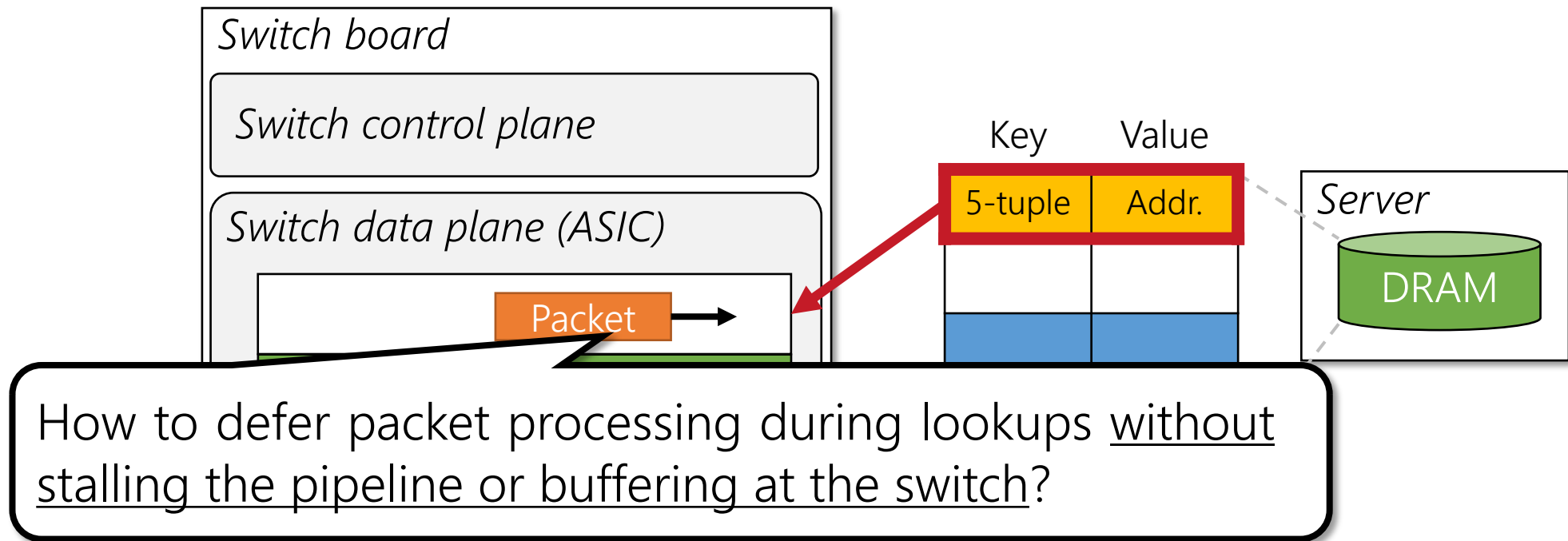


Challenge 2:

Enabling single round-trip table lookup

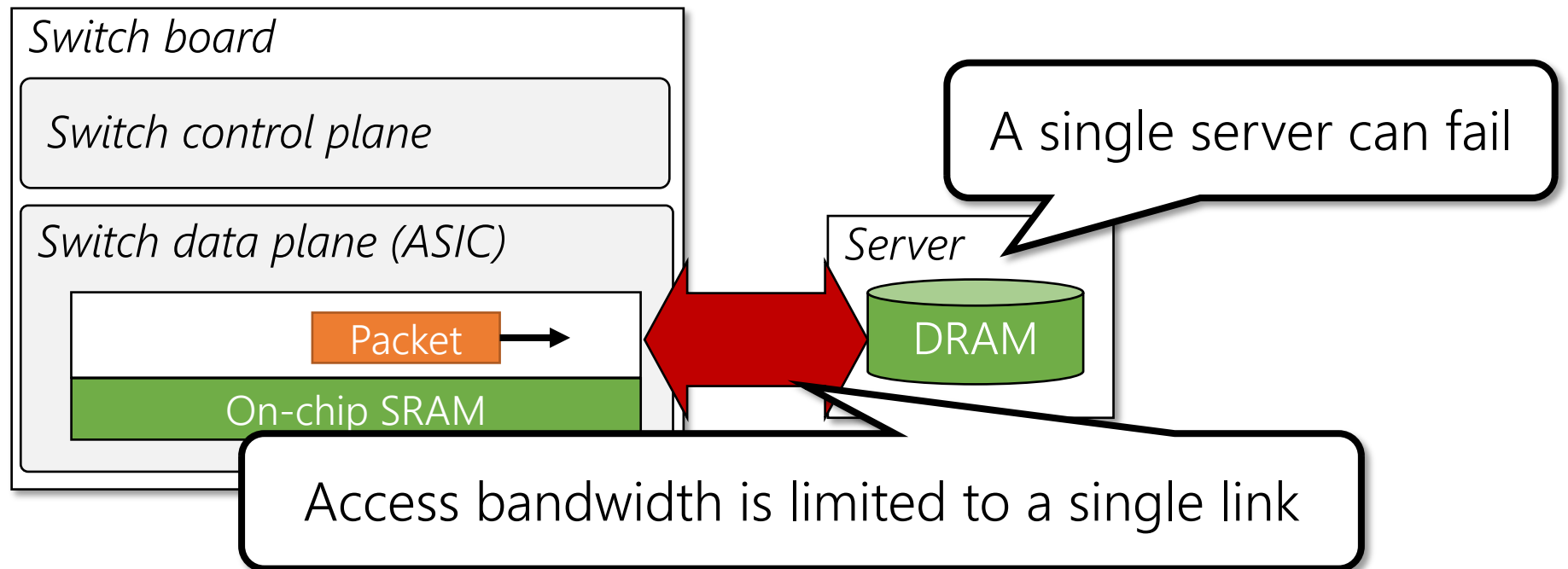


Challenge 3: Deferred packet processing

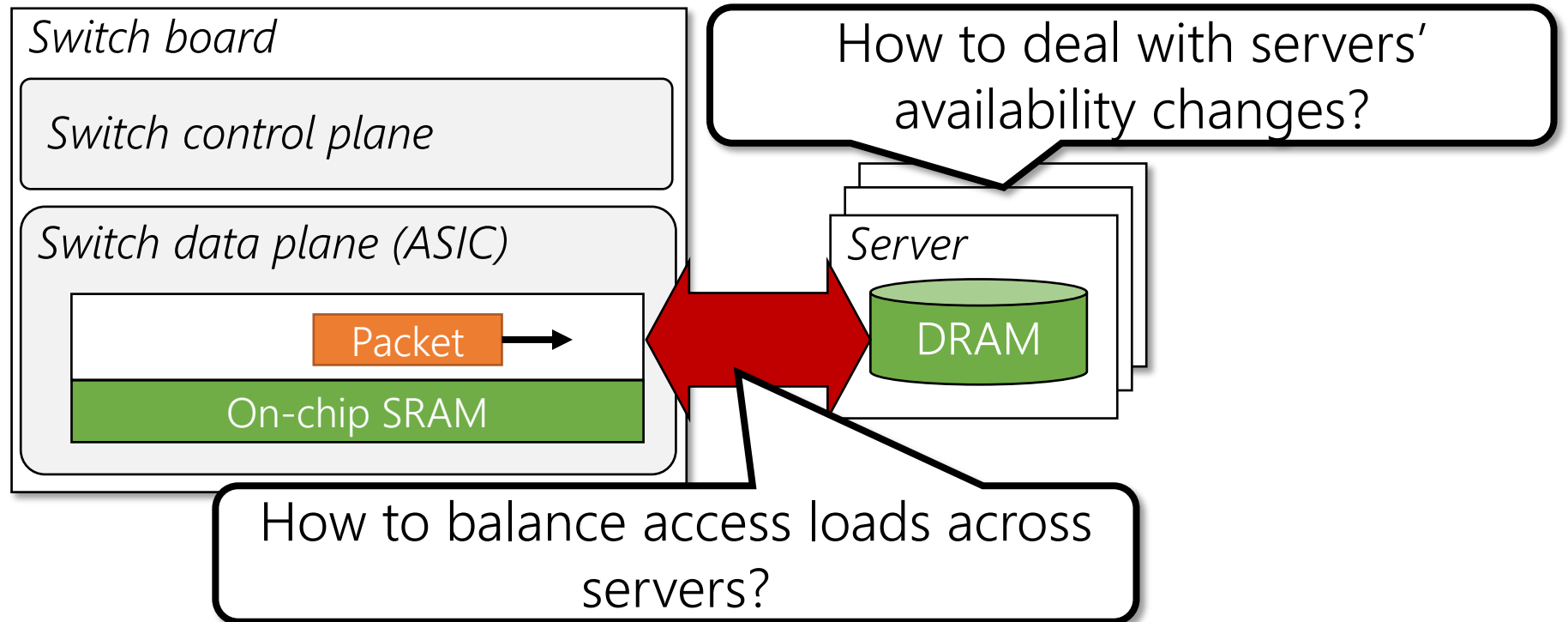


Challenge 4: Scaling TEA with multiple servers

With a single server, scalability and availability can be limited

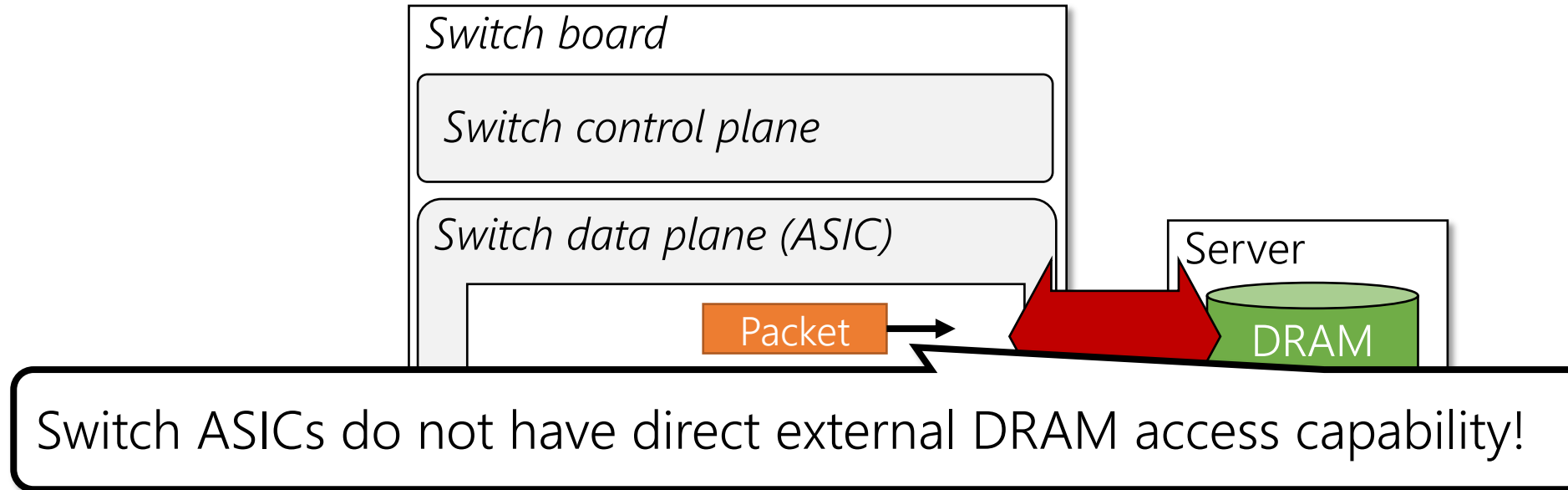


Challenge 4: Scaling TEA with multiple servers



Challenge 1:

How to access external DRAM in the data plane?

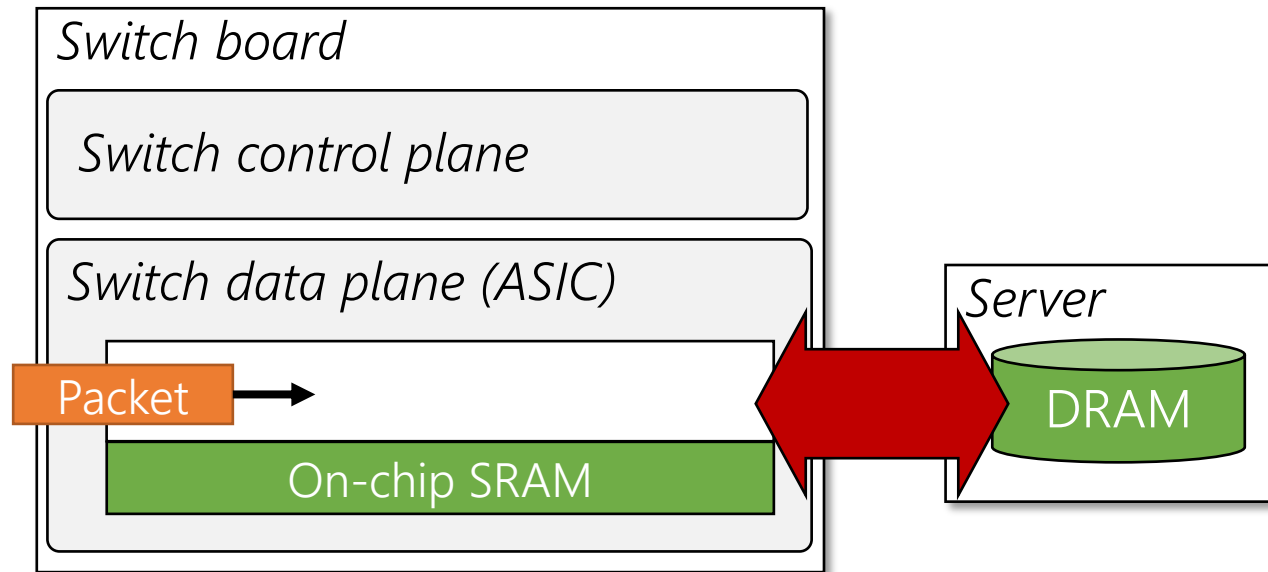


Is it possible to enable ASICs to access external DRAM without hardware modifications and CPU involvement?



Enabling RDMA in the switch data plane

Key idea: Crafting RDMA packets using ASIC's programmability

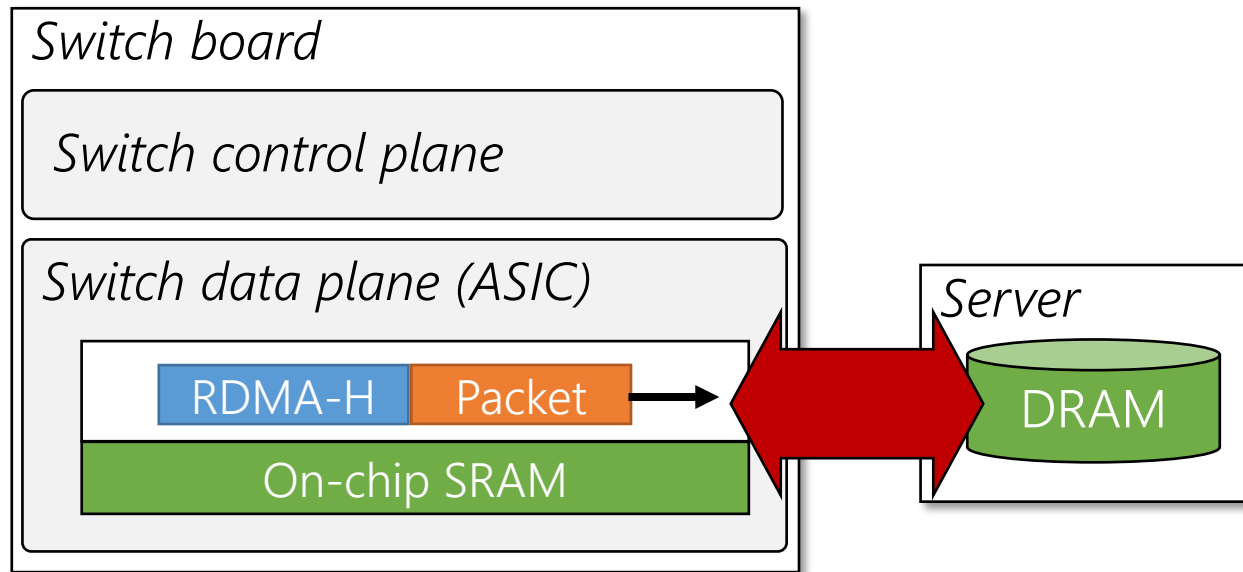


1. A packet comes into the pipeline



Enabling RDMA in the switch data plane

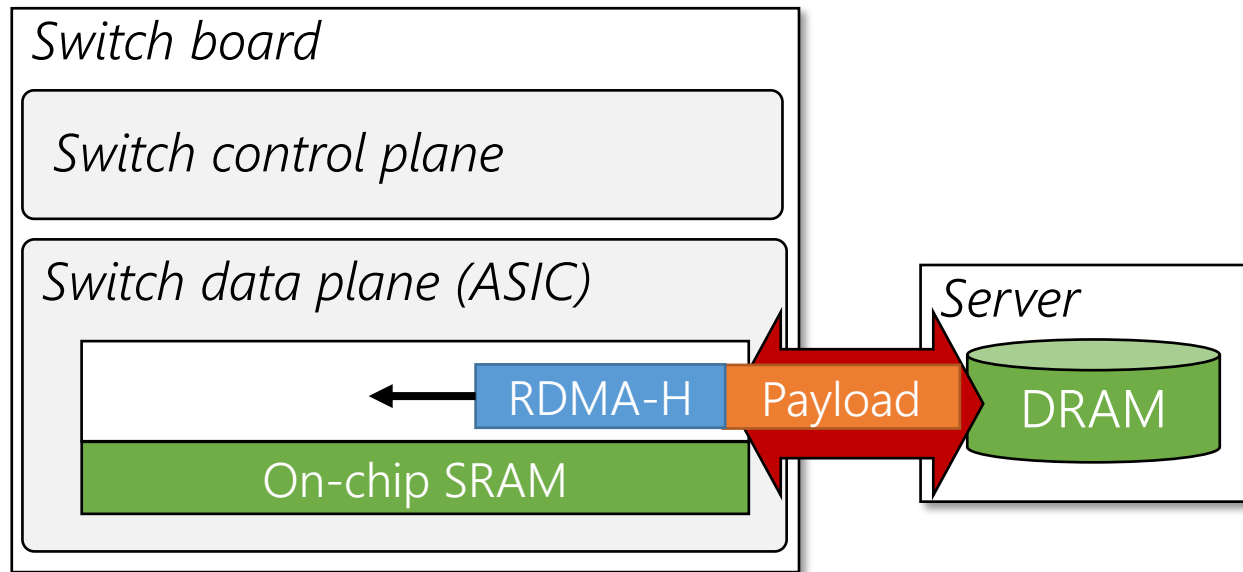
Key idea: Crafting RDMA packets using ASIC's programmability



1. A packet comes into the pipeline
2. The ASIC adds RDMA headers to craft an RDMA request

Enabling RDMA in the switch data plane

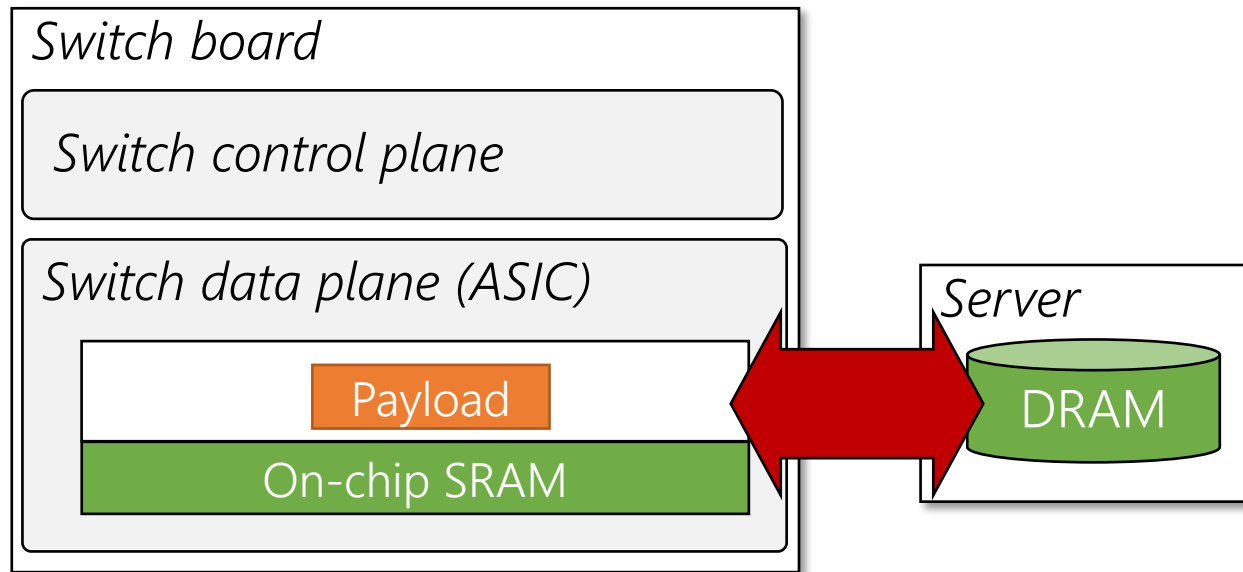
Key idea: Crafting RDMA packets using ASIC's programmability



1. A packet comes into the pipeline
2. The ASIC adds RDMA headers to craft an RDMA request
3. The server NIC replies as it would for any standard RDMA request

Enabling RDMA in the switch data plane

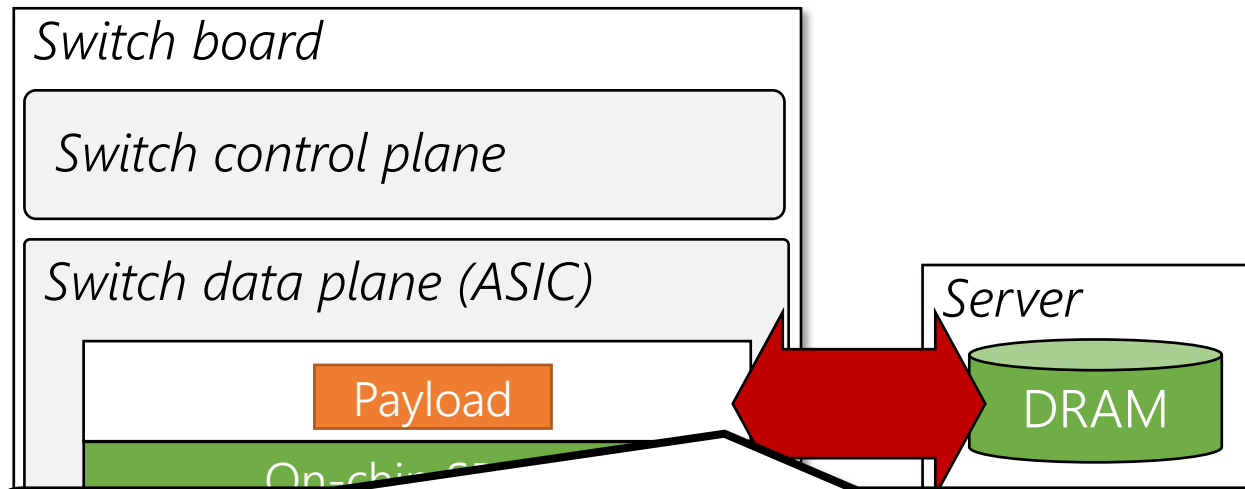
Key idea: Crafting RDMA packets using ASIC's programmability



1. A packet comes into the pipeline
2. The ASIC adds RDMA headers to craft an RDMA request
3. The server NIC replies as it would for any standard RDMA request
4. The ASIC parses the response

Enabling RDMA in the switch data plane

Key idea: Crafting RDMA packets using ASIC's programmability



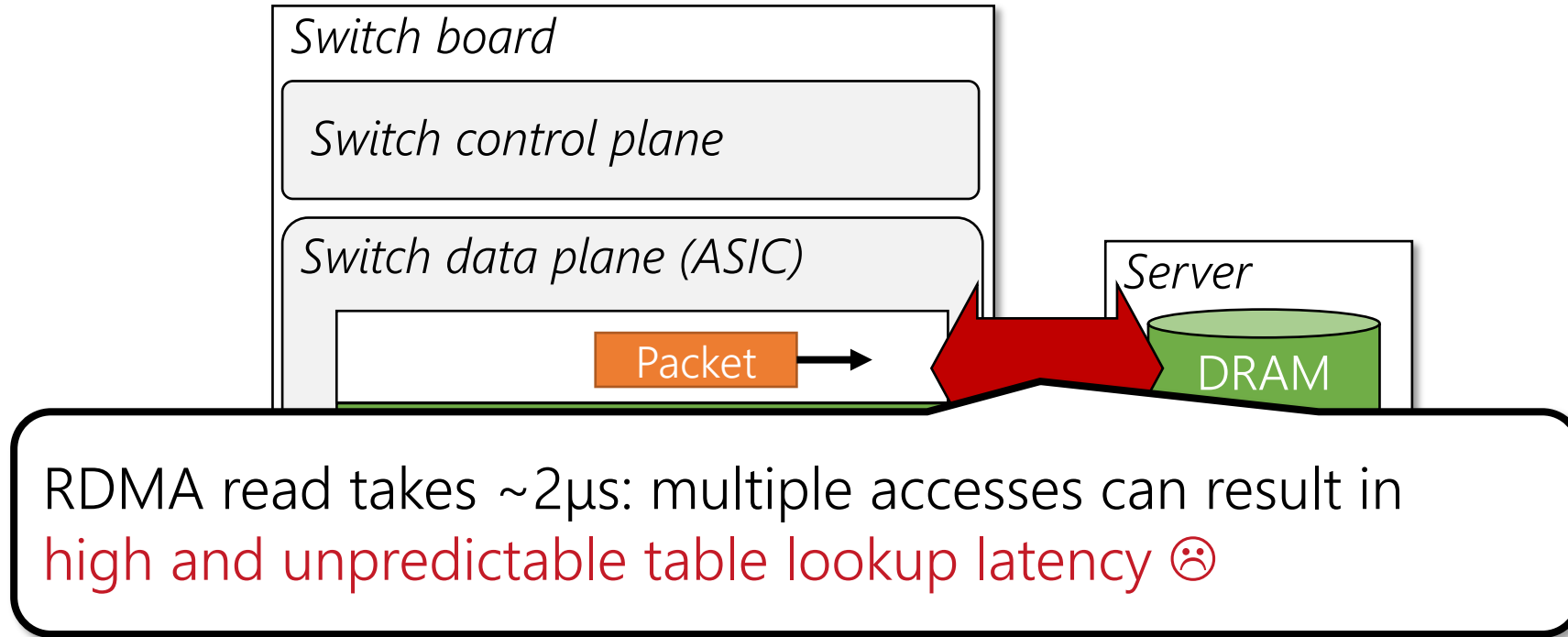
1. A packet comes into the pipeline
2. The ASIC adds RDMA headers to craft an RDMA request
3. The server NIC replies as it would for any standard RDMA request

Simple switch-side flow control prevents buffer overflows at the NIC
→ Simplified transport is enough!

ponse

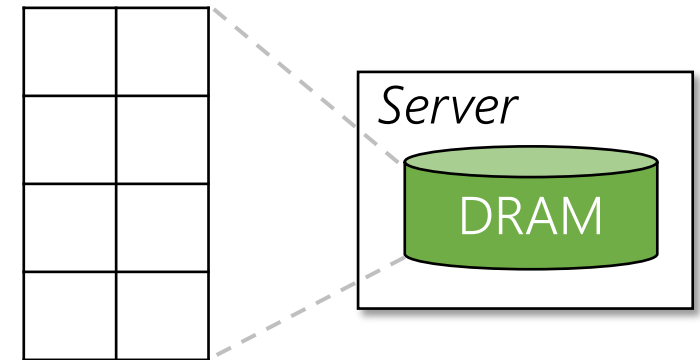
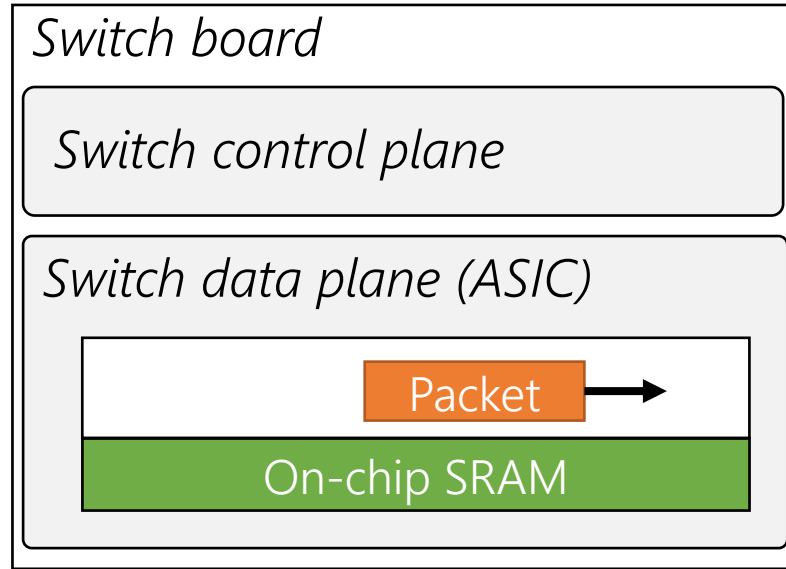


Challenge 2: Single round-trip table lookups



Can we enable external table lookups in a single round trip?
i.e., we need $O(1)$ lookup mechanism

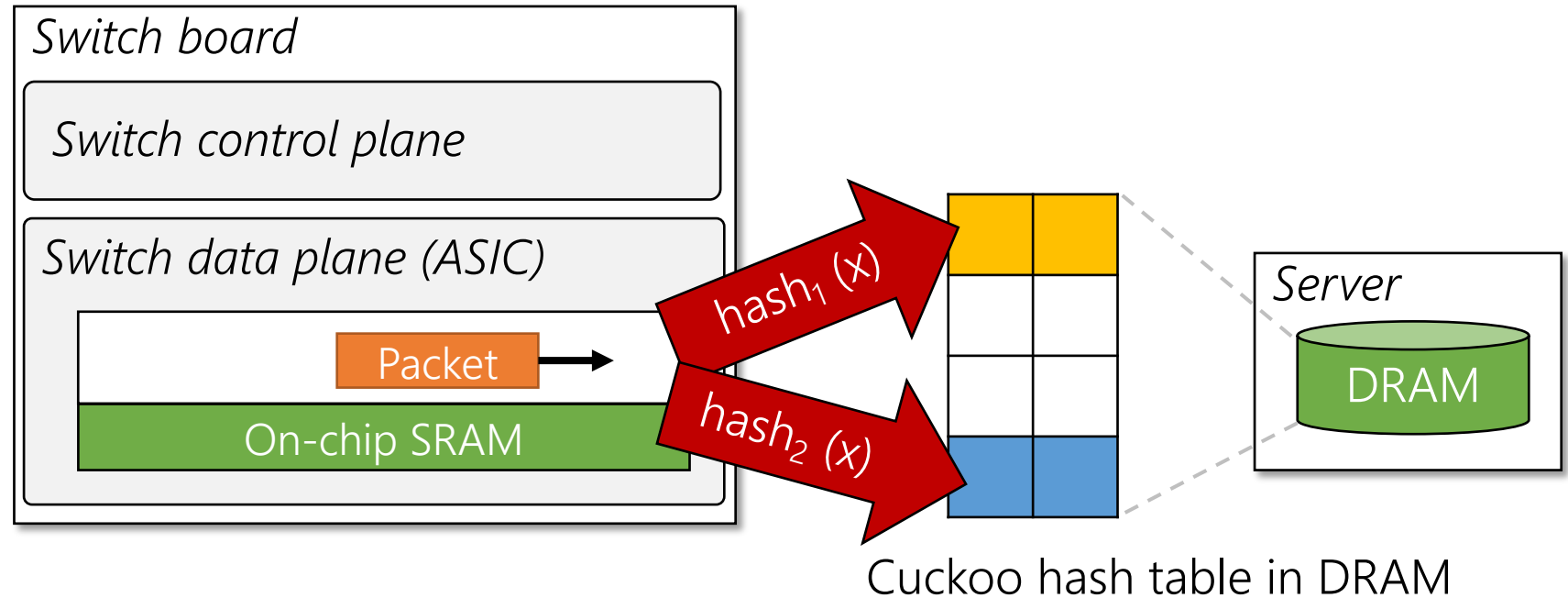
Cuckoo hashing as a potential approach



Cuckoo hash table in DRAM



Cuckoo hashing as a potential approach



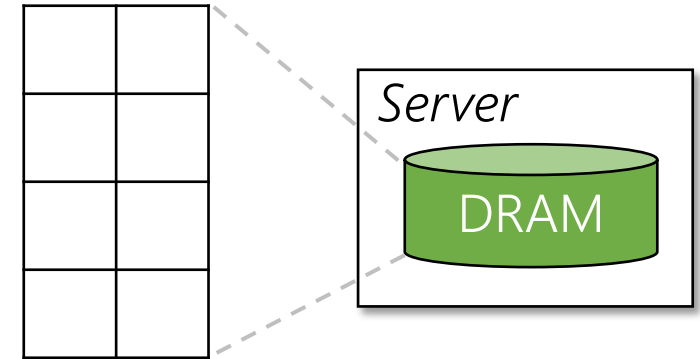
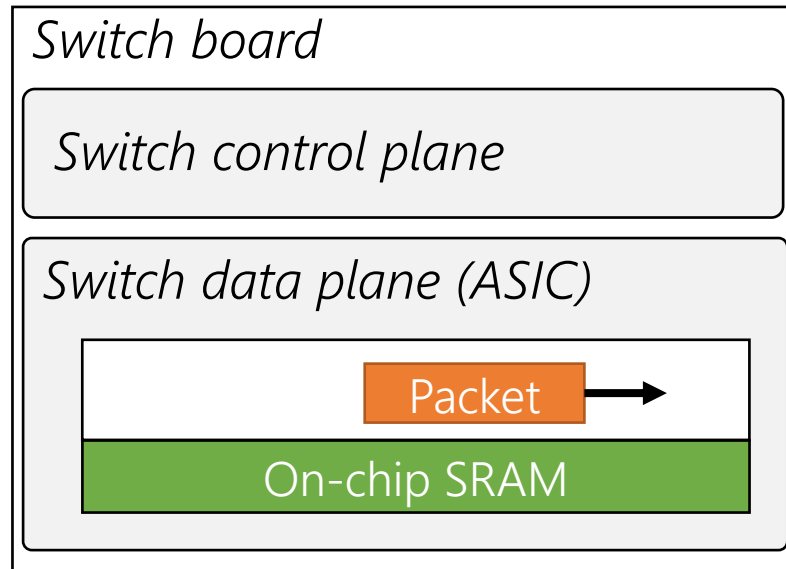
Can we enable table lookup with a single memory access?



TEA-table: lookup data structure

Designed for improving cache hit rate in software switch [Zhuo'19*]

Key idea: Repurposing bounded linear probing (BLP)



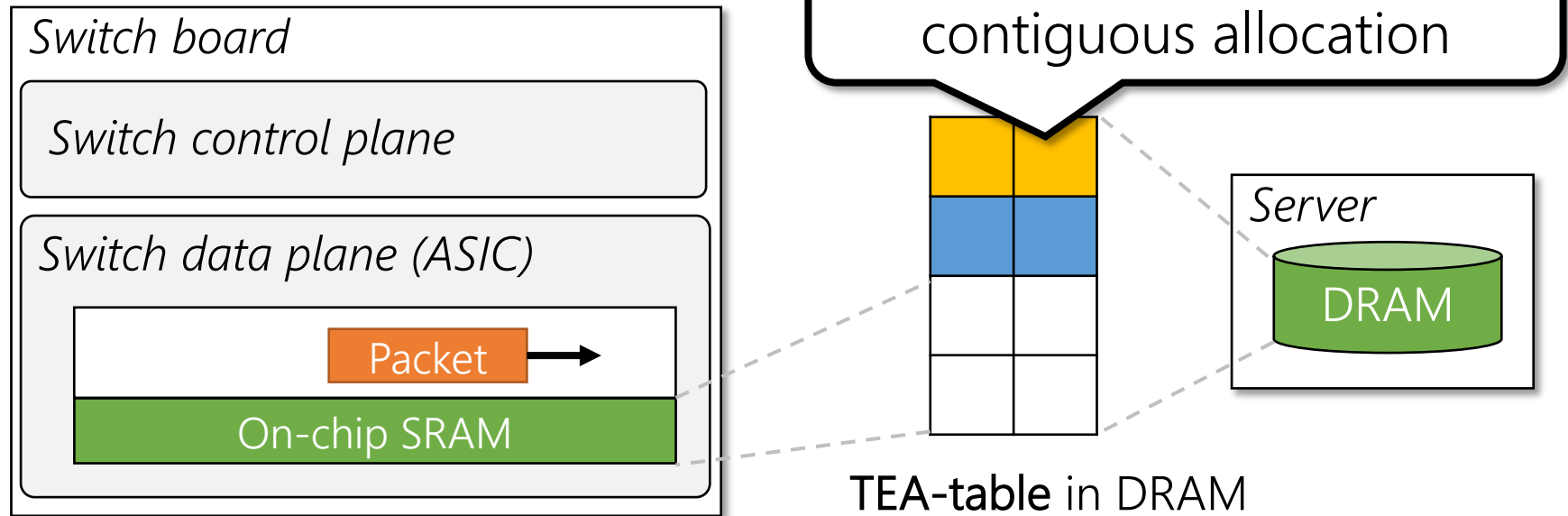
TEA-table in DRAM

*Dong Zhou. Data Structure Engineering for High Performance Software Packet Processing. Ph.D. Dissertation. Carnegie Mellon University, 2019.



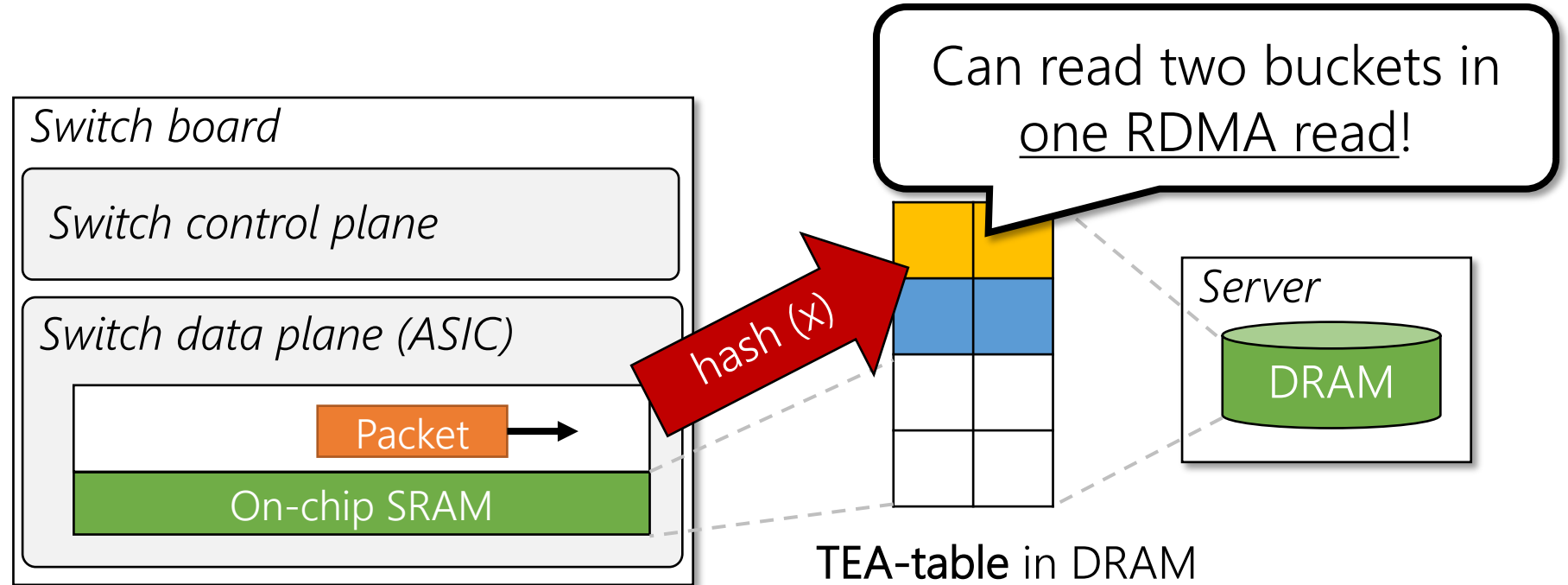
TEA-table: lookup data structure

Key idea: Repurposing bounded linear probing (BLP)

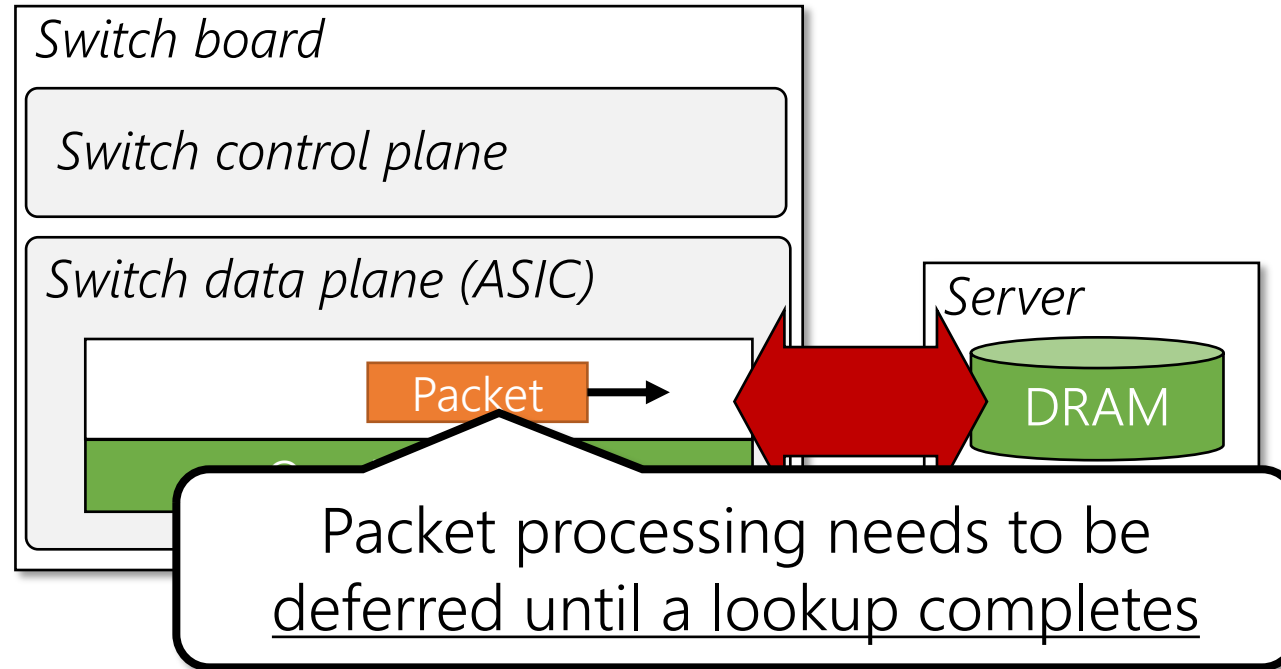


TEA-table: lookup data structure

Key idea: Repurposing bounded linear probing (BLP)



Challenge 3: Deferred packet processing

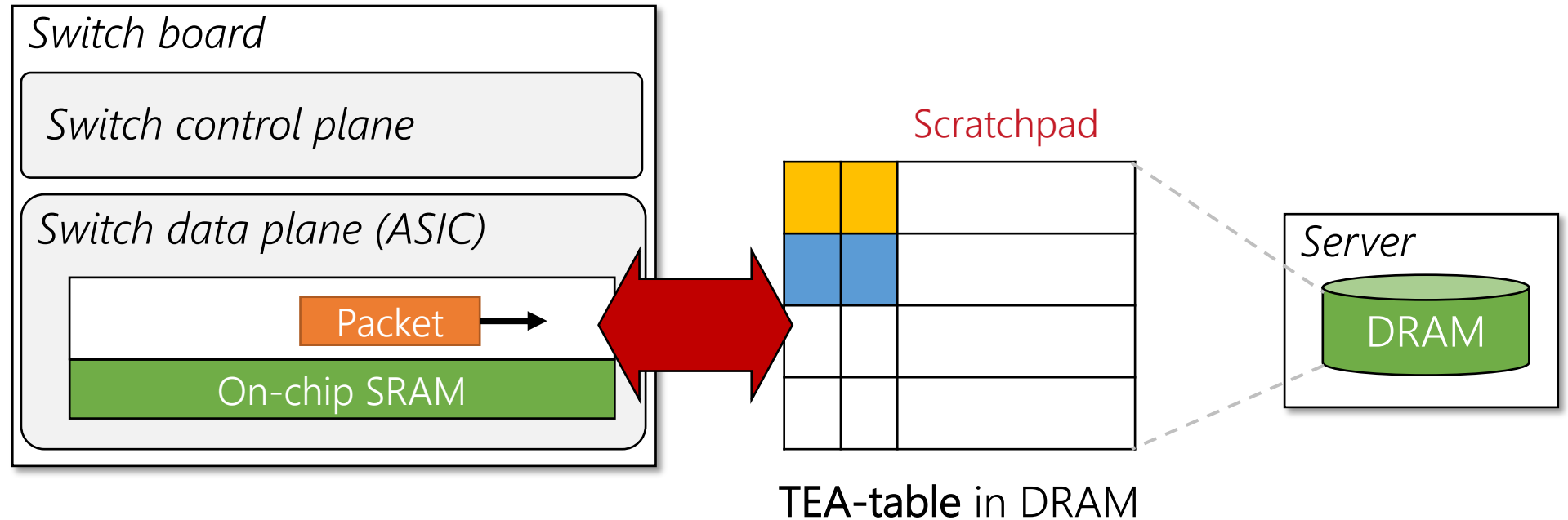


Can we defer only a select packet without stalling the pipeline?



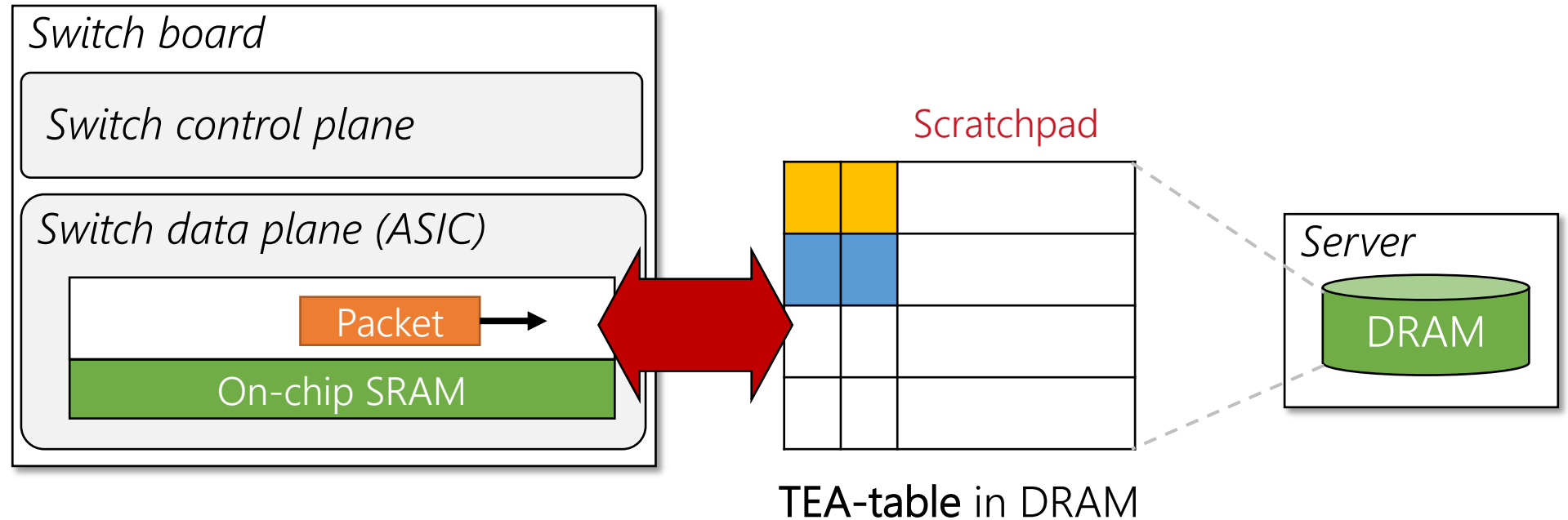
Offloading packet store to TEA-table for asynchronous packet processing

Idea #1: Employing scratchpad in TEA-table to buffer packets



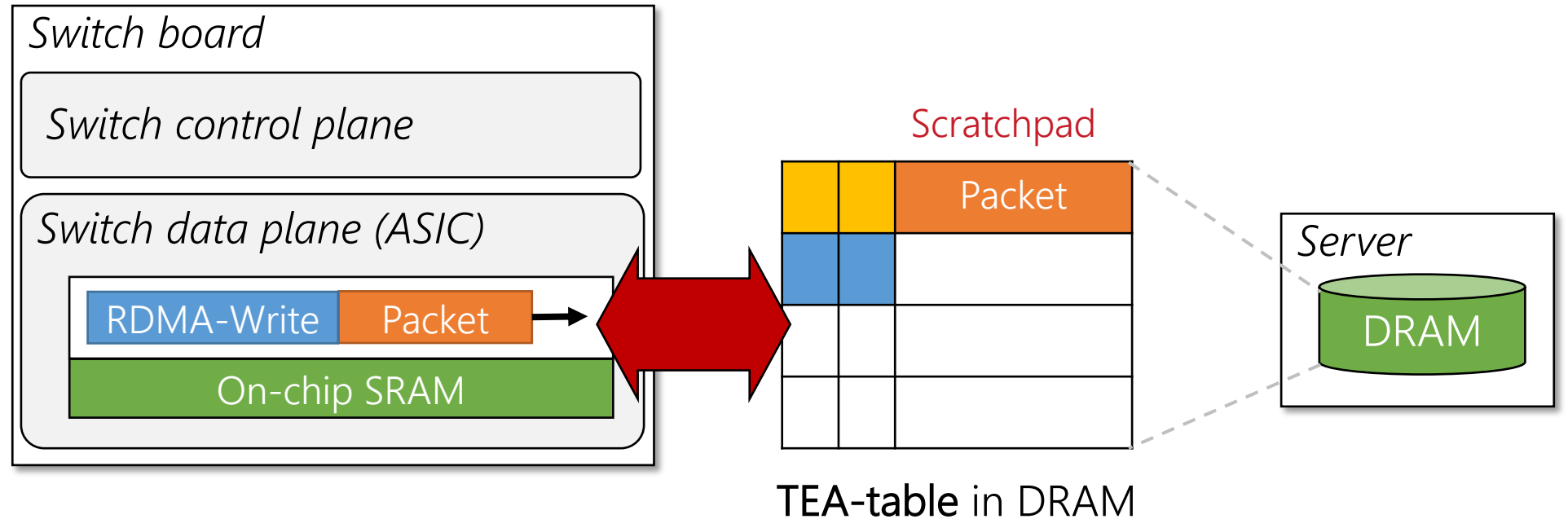
Offloading packet store to TEA-table for asynchronous packet processing

Idea #1: Employing scratchpad in TEA-table to buffer packets



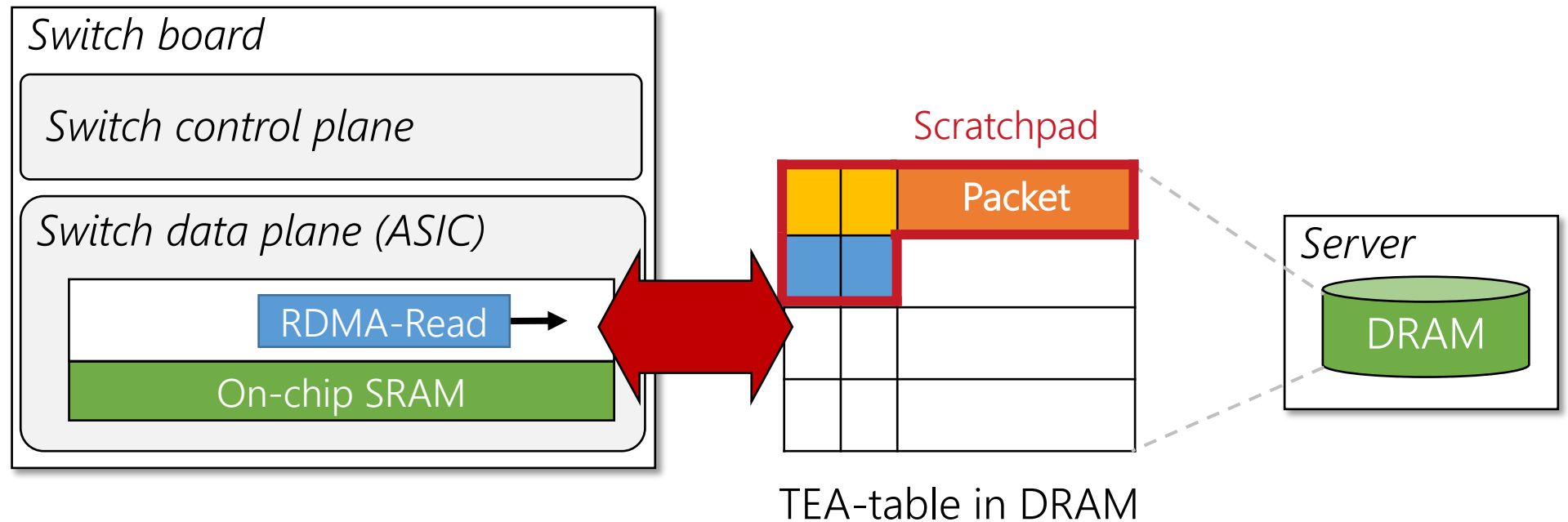
Offloading packet store to TEA-table for asynchronous packet processing

Idea #1: Employing scratchpad in TEA-table to buffer packets



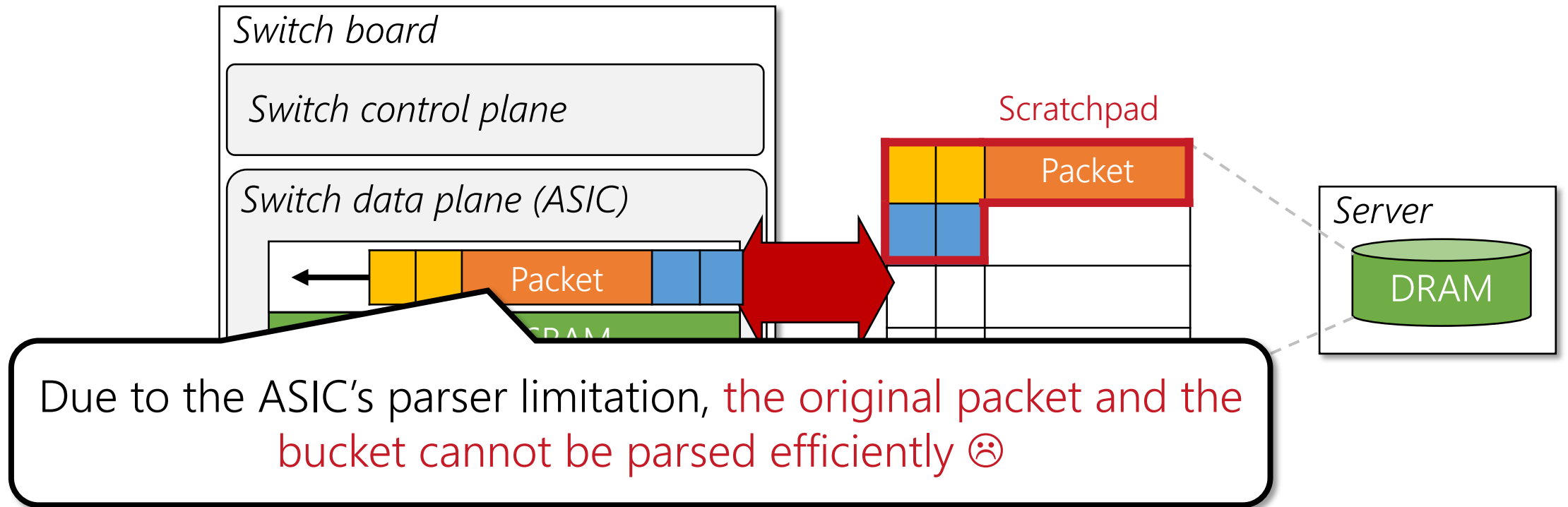
Offloading packet store to TEA-table for asynchronous packet processing

Idea #1: Employing scratchpad in TEA-table to buffer packets



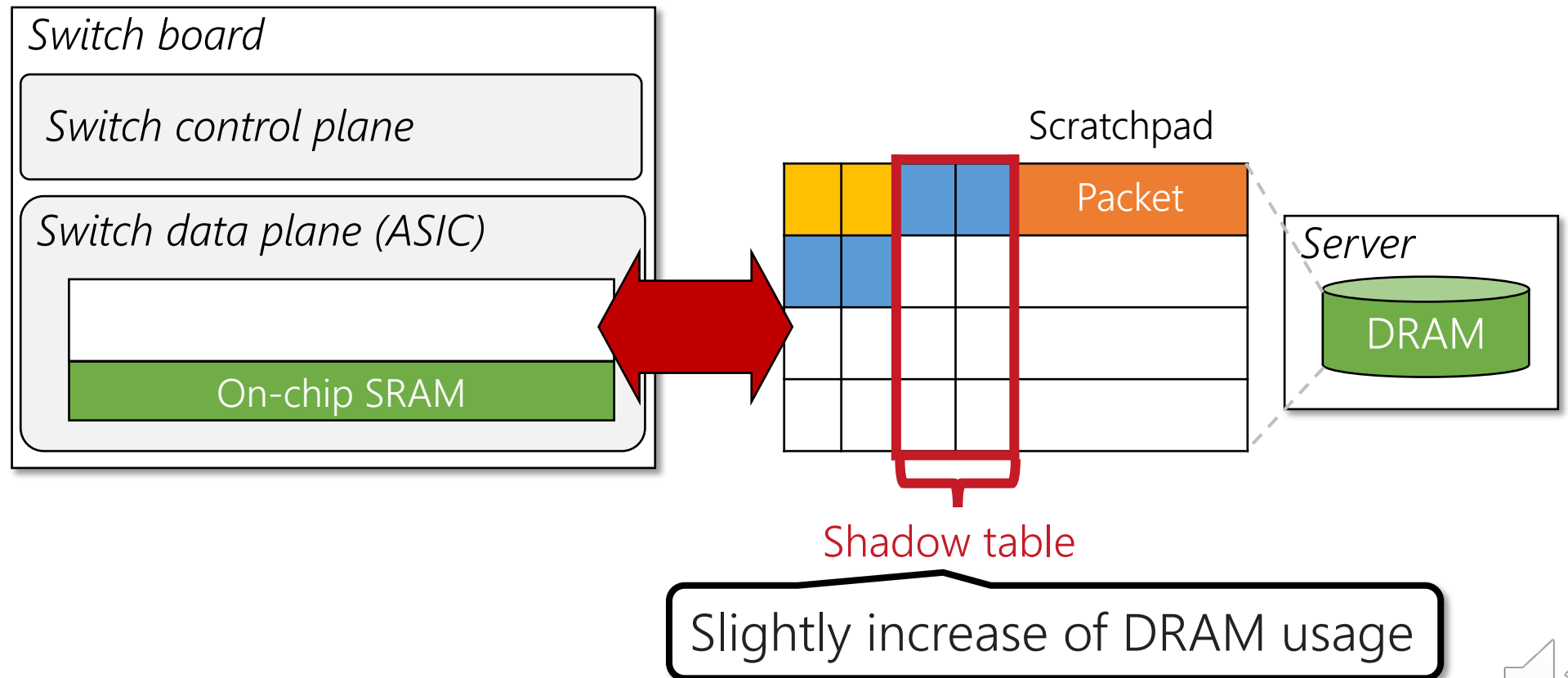
Offloading packet store to TEA-table for asynchronous packet processing

Idea #1: Employing scratchpad in TEA-table to buffer packets



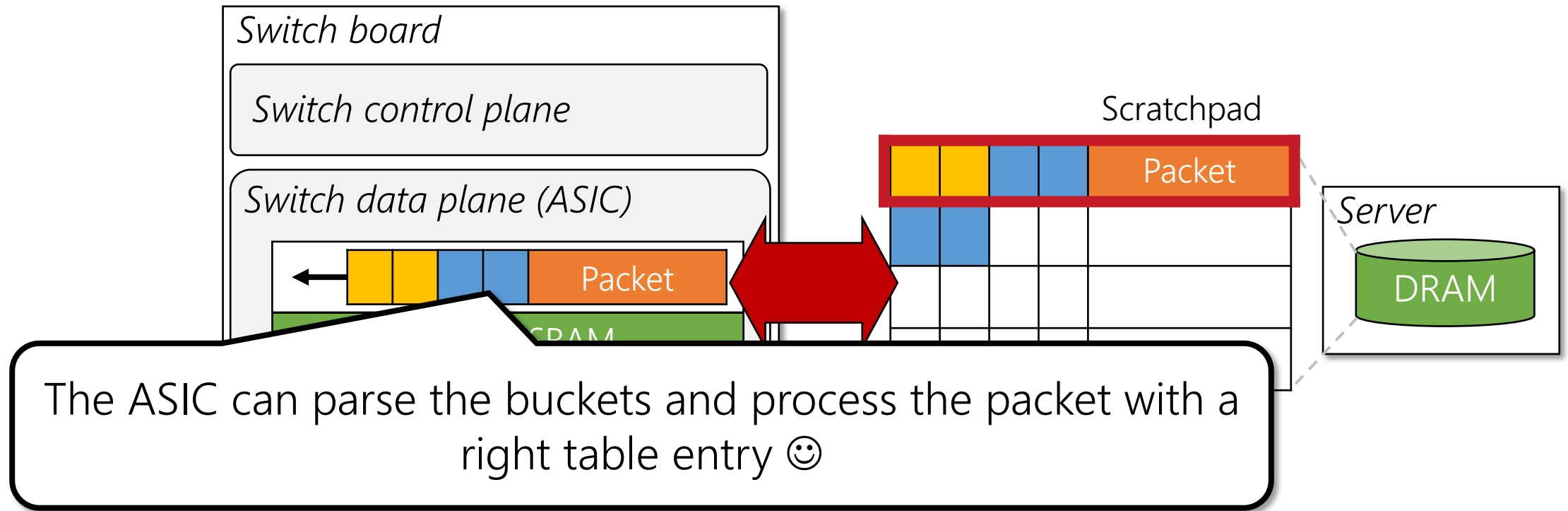
Offloading packet store to TEA-table for asynchronous packet processing

Idea #2: Employing shadow table in TEA-table

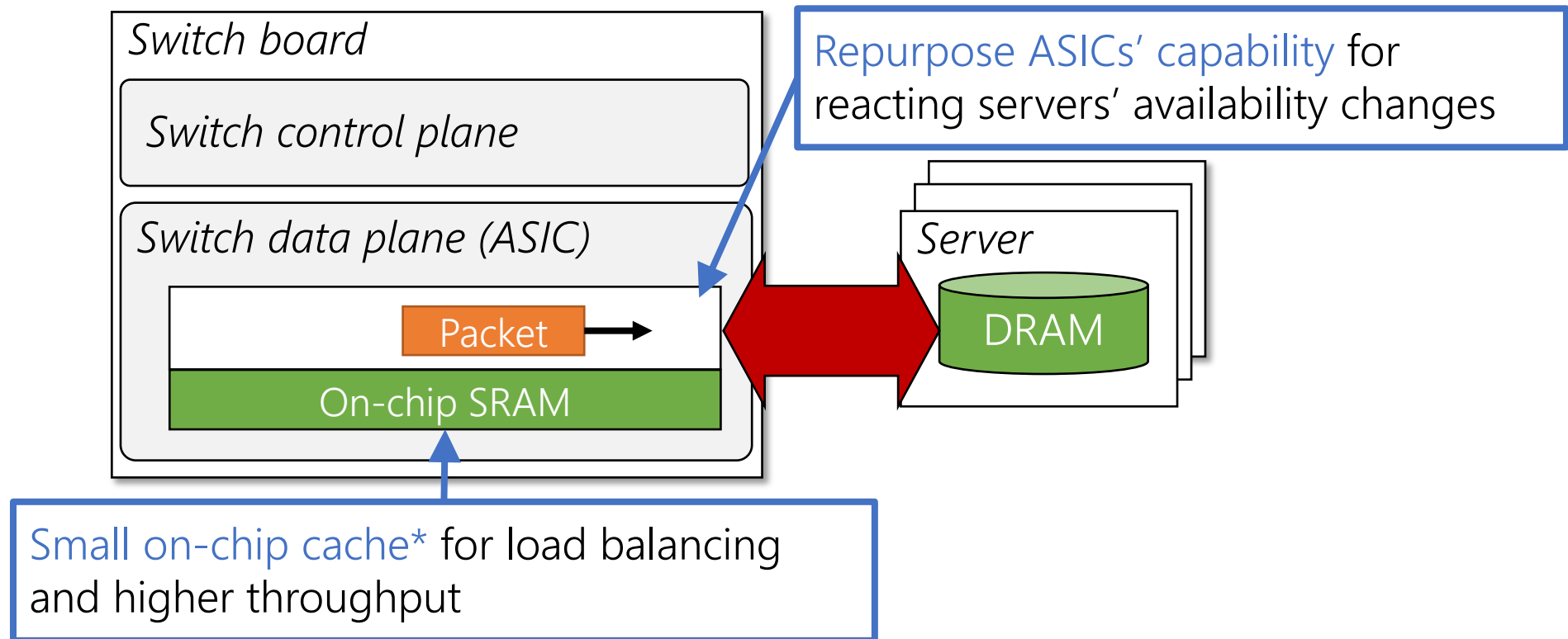


Offloading packet store to TEA-table for asynchronous packet processing

Idea #2: Employing shadow table in TEA-table



Challenge 4: Scaling TEA with multiple servers



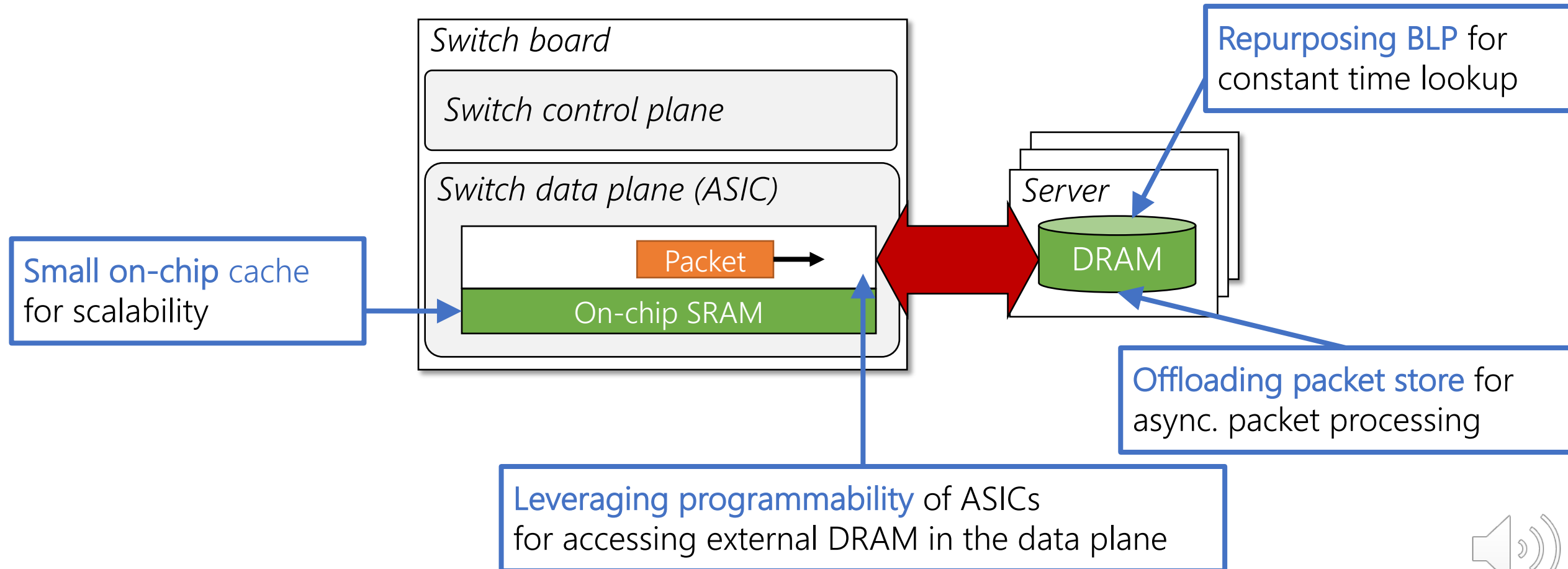
* $O(N \log N)$, where N is the number of servers.

*Bin Fan et. al., Small Cache, Big Effect: Provable Load Balancing for Randomly Partitioned Cluster Services, ACM SOCC 2011.



Putting it all together

TEA provides a virtual table abstraction to state-intensive NFs



Outline

Motivation

TEA design

Results



Implementation and evaluation setup

Implemented **TEA API** as P4 modules (aka. control block in P4)

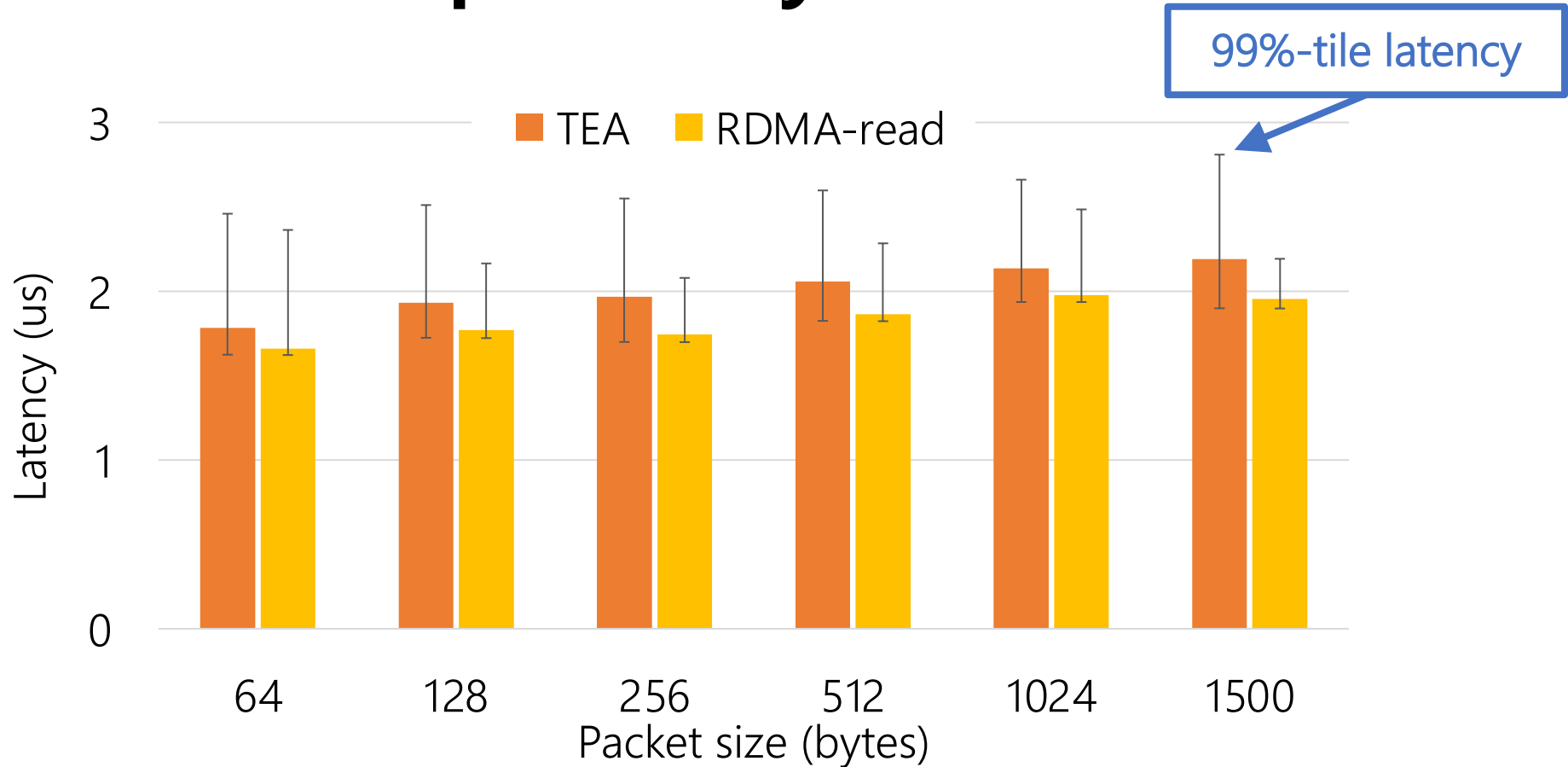
Implemented **canonical NFs** including NAT and stateful firewall

- Load 10 million table entries

Testbed setup: Tofino-based switch + 12 servers with RDMA NICs

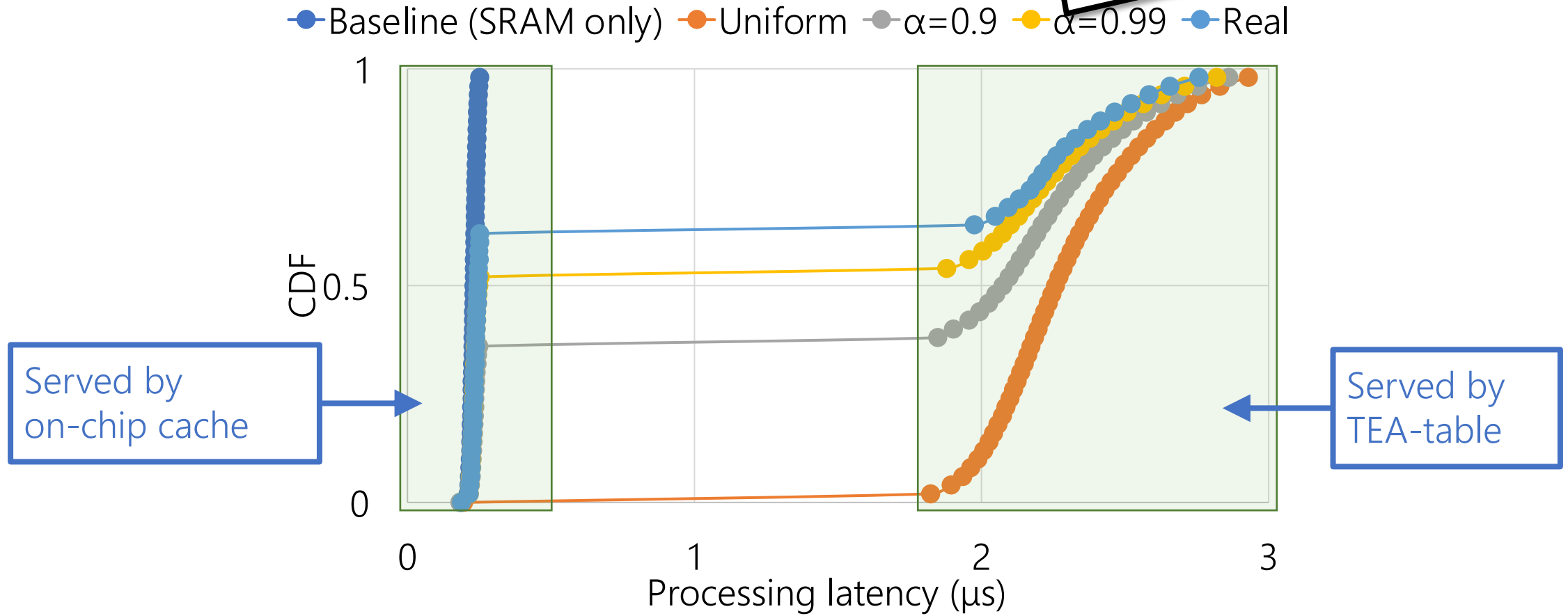


Does TEA access channel provide low and predictable lookup latency?

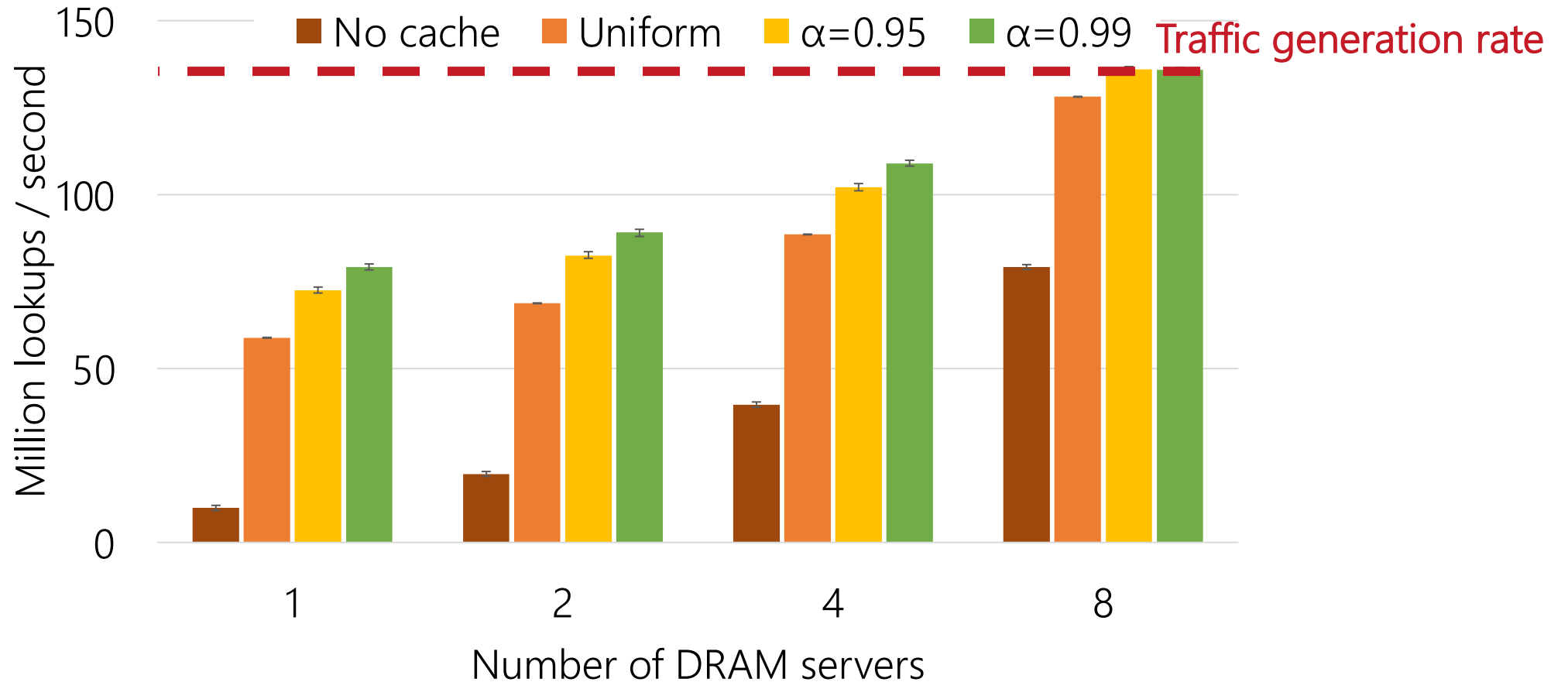


How does TEA affect NF processing latency?

Flow size skewness (α) affects on-chip cache hit rate



Does TEA provide scalable throughput?



Other results

Cost efficiency of TEA-enabled NFs compared to server-based counterparts.

Handling server failures within a second with a slight throughput degradation

Less than 9% of switch ASIC resource usage

See paper for more details! *

*D. Kim et al., "TEA: Enabling State-Intensive Network Functions on Programmable Switches.", ACM SIGCOMM



Conclusions

Limited on-chip memory restricts potentials of switch-based NFs

Table Extension Architecture for programmable switches:

- RDMA-based external DRAM access
- BLP-based efficient lookup table structure
- Asynchronous packet processing by offloading packet store
- Small on-chip cache for scalability

Provides low and predictable latency with scalable throughput

- Latency: 1.8 - 2.2 μ s
- Throughput: 138 million lookups/sec with 8 servers

