

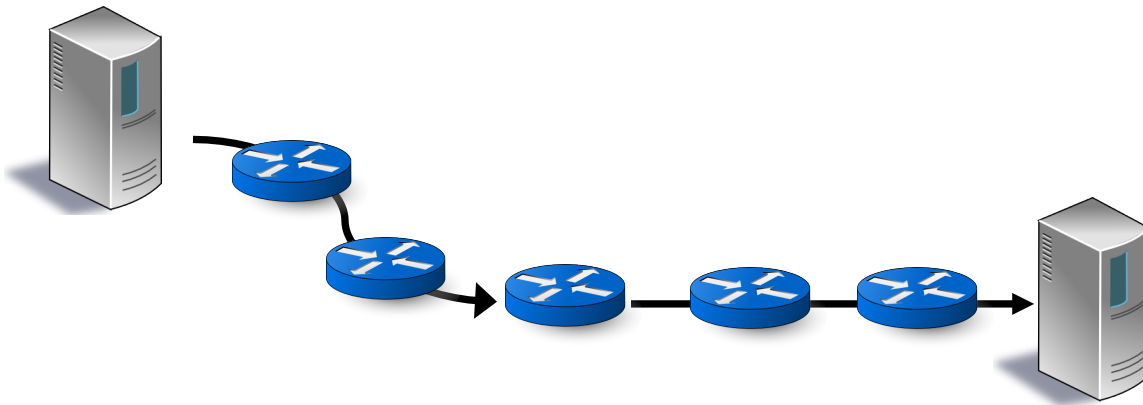
# Language-Directed Hardware Design for Network Performance Monitoring

**Srinivas Narayana**, Anirudh Sivaraman, Vikram Nathan,  
Prateesh Goyal, Venkat Arun, Mohammad Alizadeh, Vimal  
Jeyakumar, and Changhoon Kim

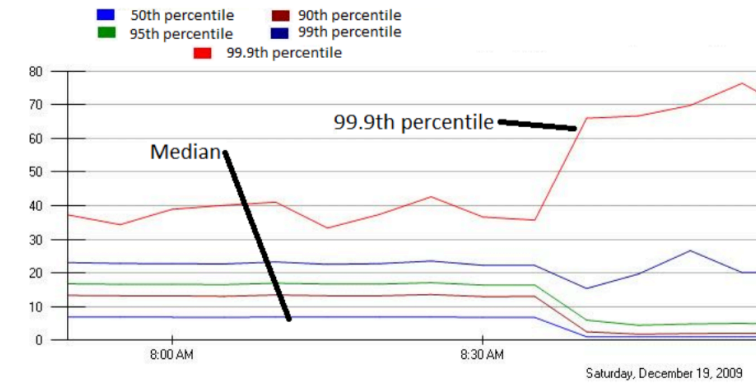


A network performance monitoring example:  
High tail latencies

# High tail latencies



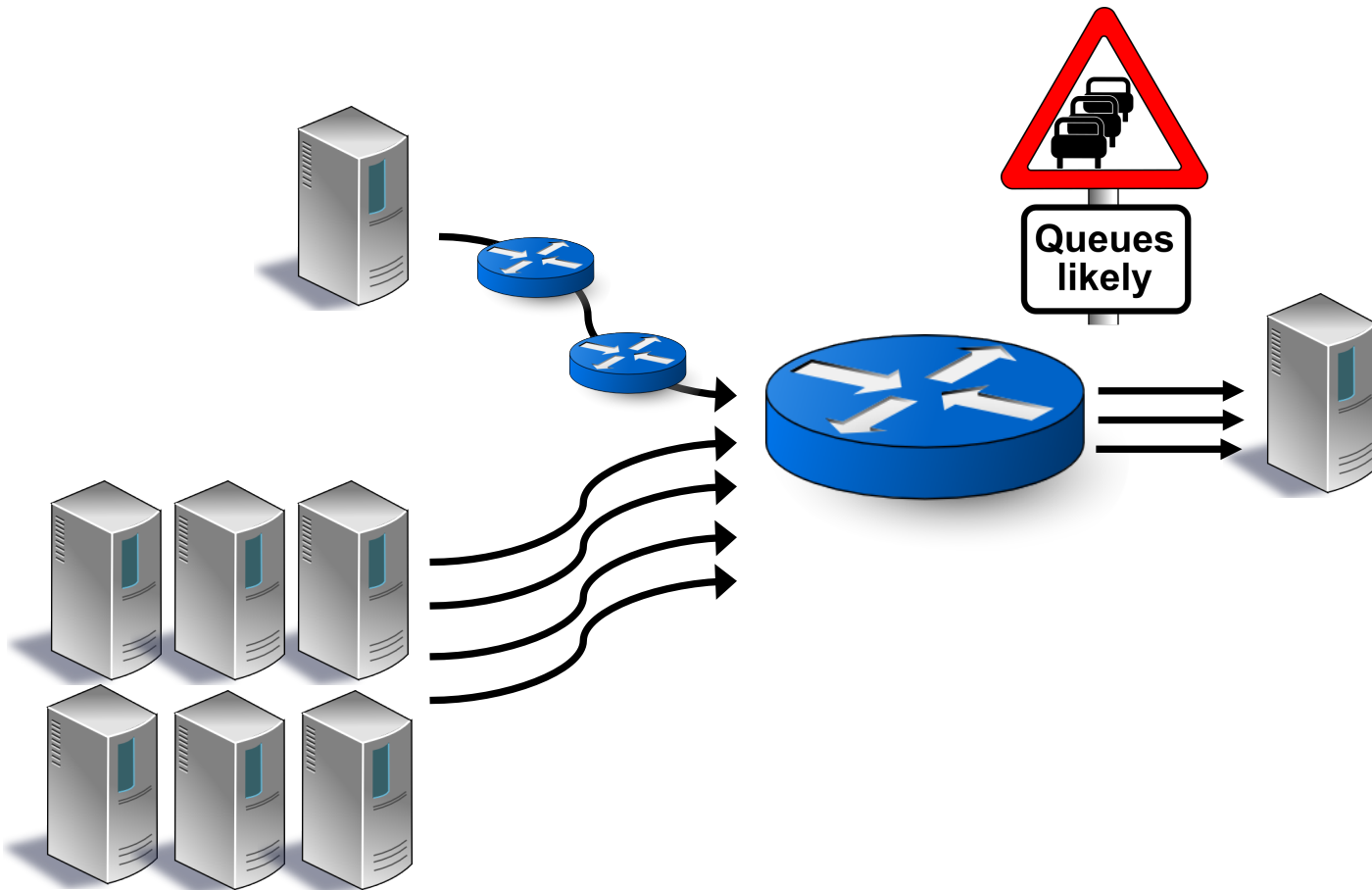
[DCTCP, Sigcomm'10]



**Figure 8: Response time percentiles for a production application having the incast traffic pattern. Forwarded requests**

Delay in responding to application requests.

# High tail latencies



Where are queues building up?

How did queues build up?

- UDP on-off traffic?
- Fan-in of short flows?

Which traffic caused queue buildup?

- On-off → Throttle UDP
- Fan-in → Workload placement

# Classic measurement approaches

- ✘ • Sampling (netflow, sflow) Misses queue buildup events
- ✘ • Endpoint solutions (e.g., pingmesh) End-to-end; no direct visibility
- ✘ • Counting (flow counters, sketches) No visibility into queues
- ✘ • Packet capture (e.g., endace) Not everywhere & always

What monitoring support  
should we add to switches?

Goal: general and efficient

# Existing proposals

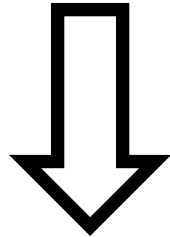
- In-band Network Telemetry (INT)
  - Queue sizes and timestamps on packets
- Flow telemetry (Tetration)
  - Packet latency, packet size variations, etc.
- Very specific (fixed) metrics, exposed at fixed granularity
  - e.g., What about latency variation (jitter)?

# Language-directed hardware design



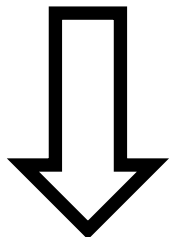
# Language-directed hardware design

Performance monitoring use cases



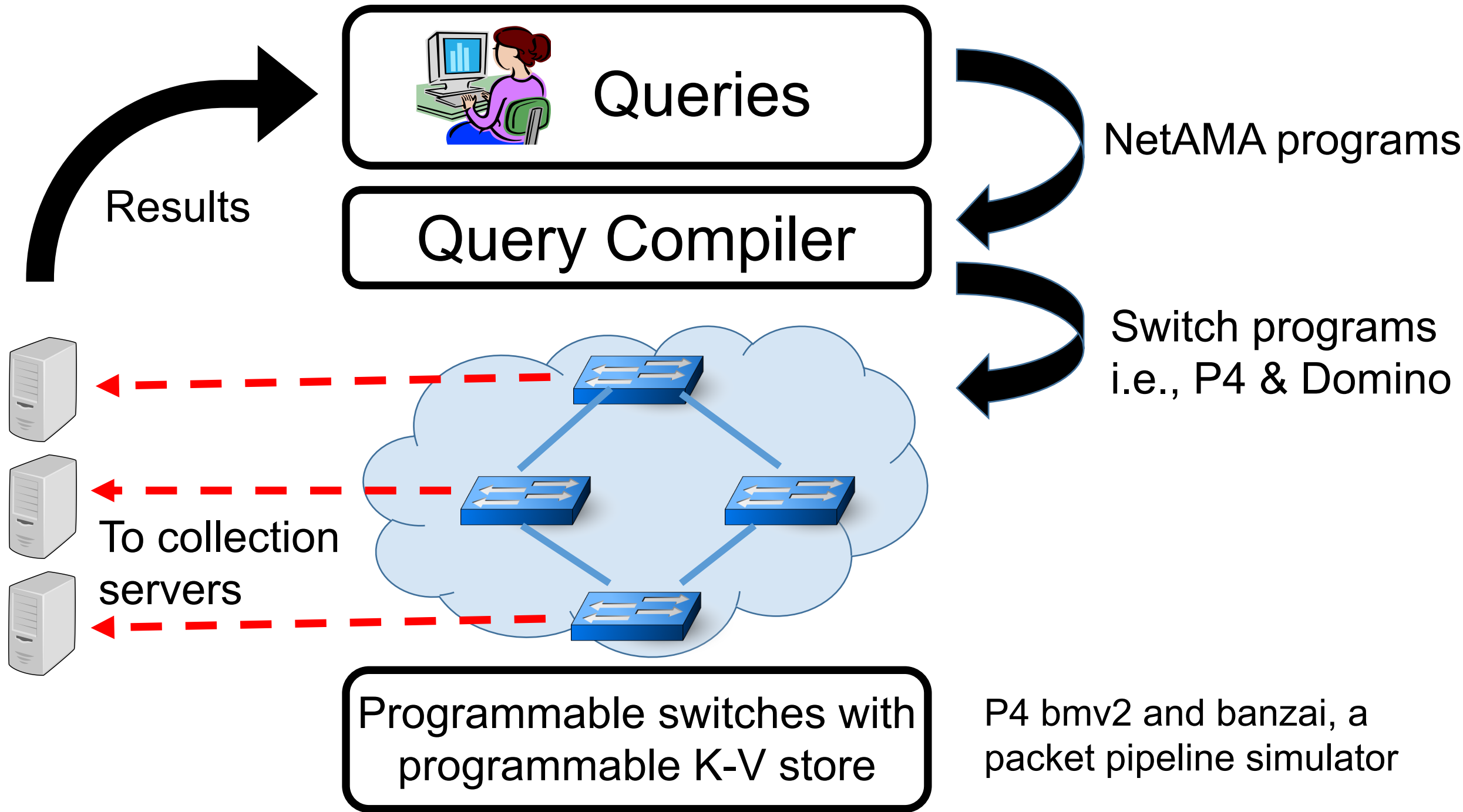
Expressive performance query language

NetAMA



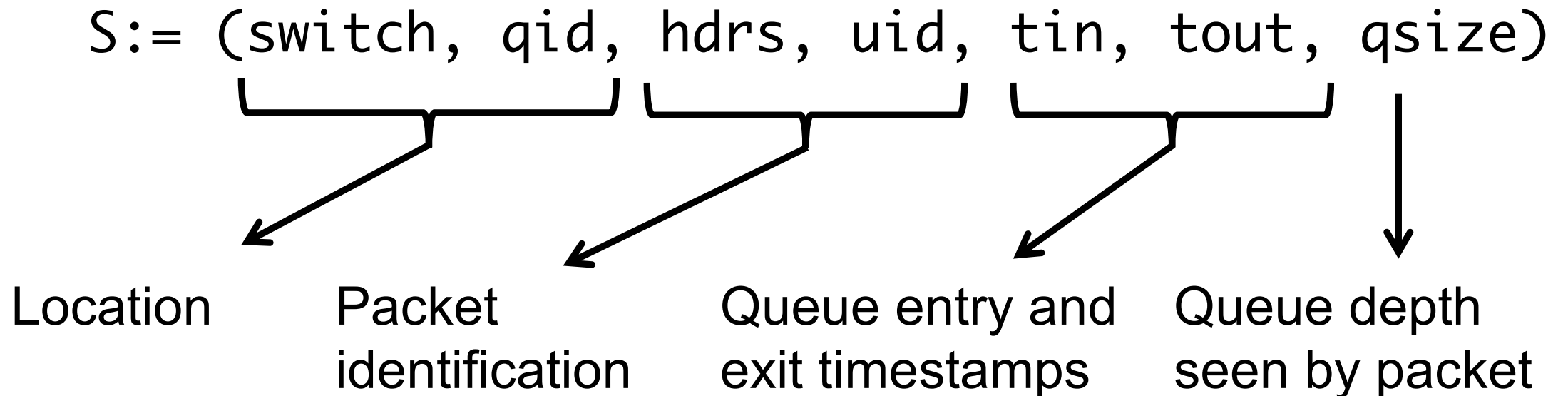
Line-rate switch hardware primitives

Programmable  
key-value store



# NetAMA: Performance query language

- Packet performance *stream*
  - Headers and performance data for *each* packet at *each* queue



# Example performance queries (1/3)

- Collect all packets with a 1ms queueing delay

$$R1 = \text{filter}(S, t_{\text{out}} - t_{\text{in}} > 1\text{ms})$$

- Operators map streams to streams
  - Natural model to compose queries on results of other queries

# Example performance queries (2/3)

- Track a smoothed average queueing delay by connection

```
ewma_q = groupby(S, 5tuple, ewma)
```

```
def ewma(avg, tin, tout):  
    avg = beta*avg + alpha*(tout-tin)
```

User-defined fold functions

# Example performance queries (3/3)

- Only collect packets with EWMA latency higher than 1ms

```
R4 = filter(ewma_q, avg > 1ms)
```

# Other performance queries (see paper)

- Transport protocol diagnoses
  - Fan-in problems (incast and outcast)
  - Incidents of reordering and retransmissions
  - Interference from bursty traffic
- Flow-level metrics
  - Packet drop rates
  - Queue latency EWMA per connection
  - Incidence and lengths of flowlets
- Network-wide questions
  - Route flapping
  - High end to end latencies
  - Locations of persistently long queues
- ...

What hardware primitives implement the performance query language?

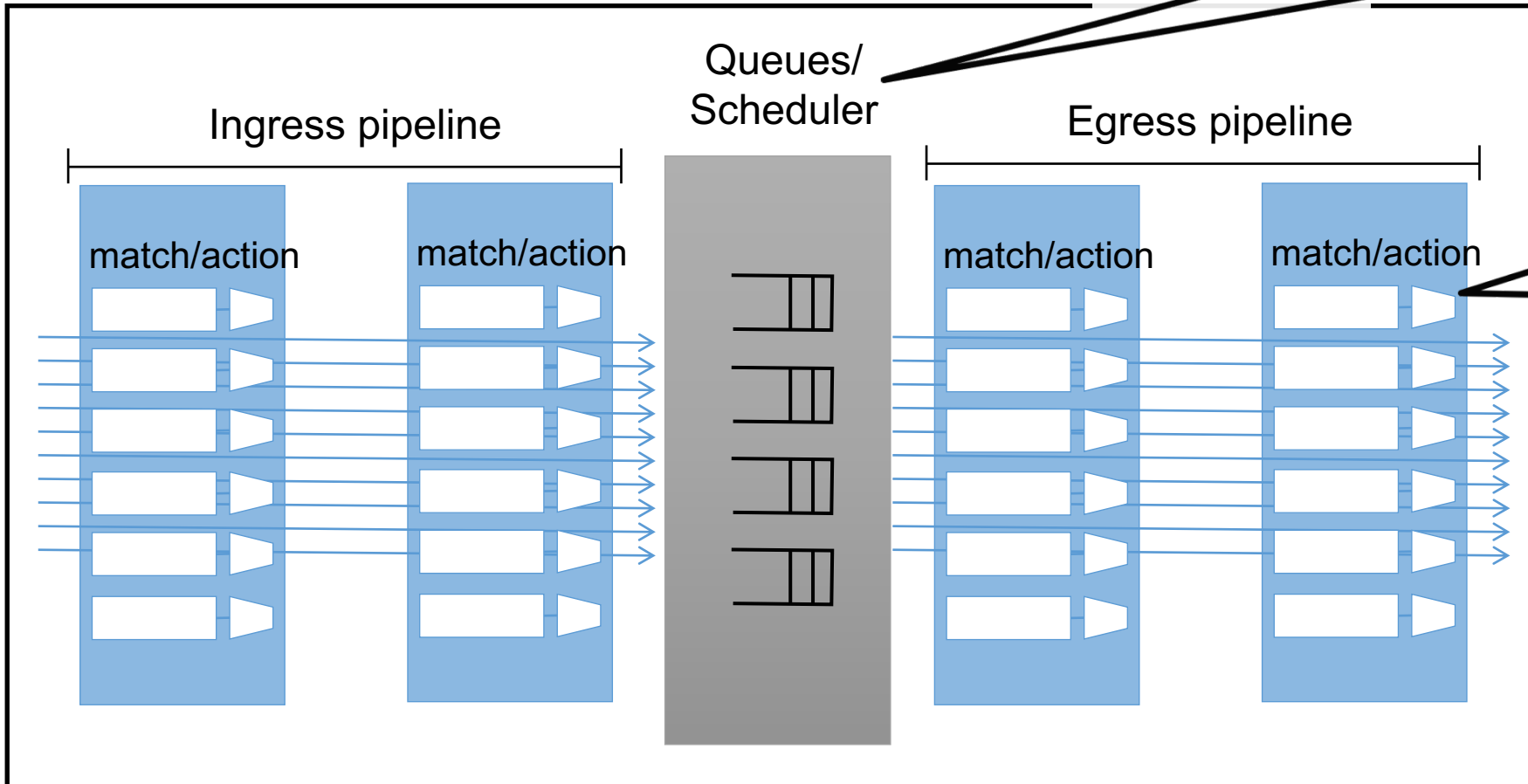


# Existing primitives are useful!

Queue length &  
pkt timestamps:  
INT metadata

filter: match-  
action rules

What remains:  
groupby!



# groupby: Challenges

```
ewma_query = groupby(S, 5-tuple, ewma)
def ewma(avg, tin, tout):
    avg = (1-alpha)*avg + alpha*(tout-tin)
```

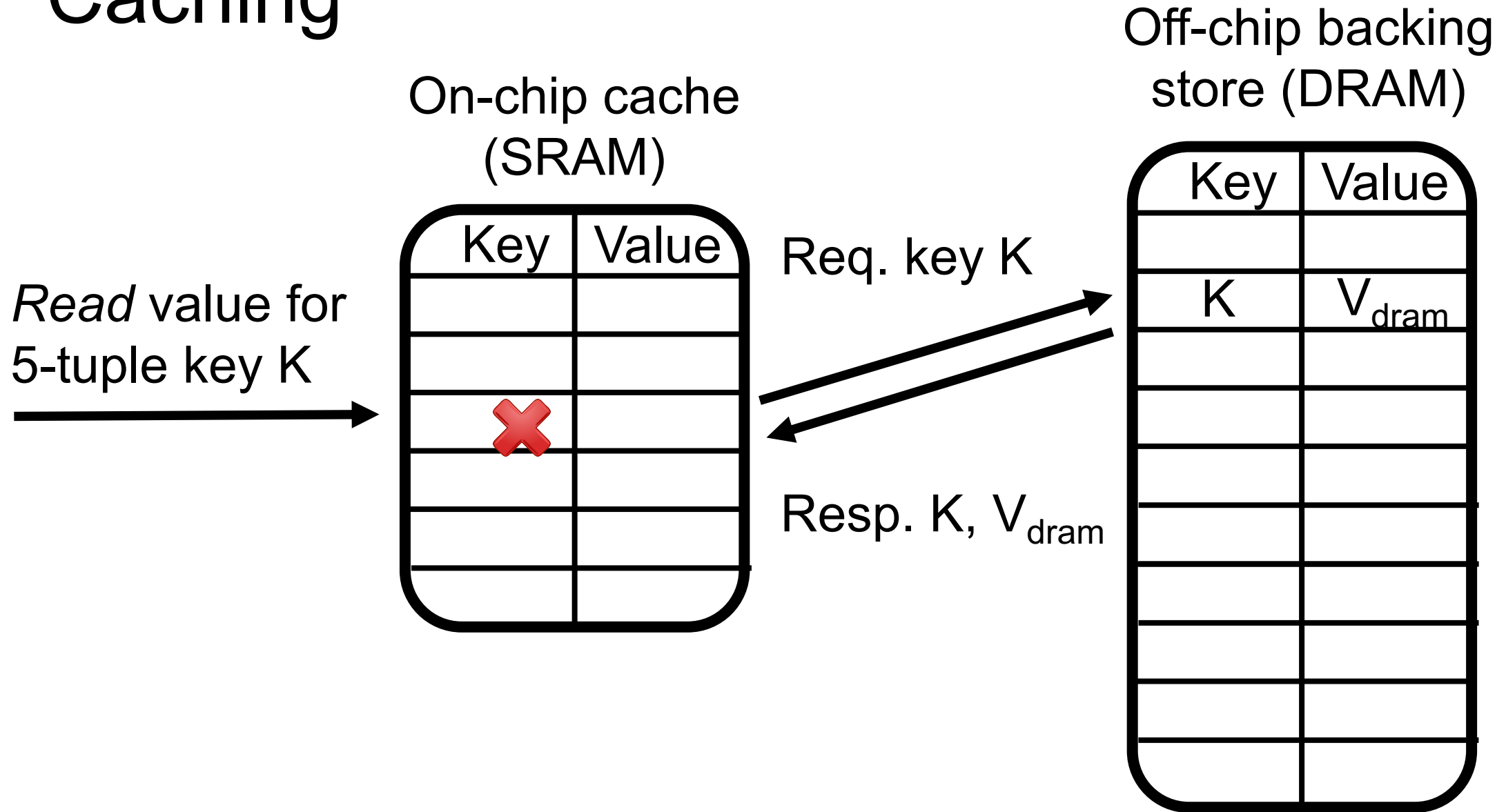
- Compute & update a value in memory for each packet (@1Ghz)
- Scale to millions of aggregation keys (e.g., 5-tuples)
- Memory must be fast *and* large: neither SRAM nor DRAM works!

Caching:  
the illusion of fast and large memory

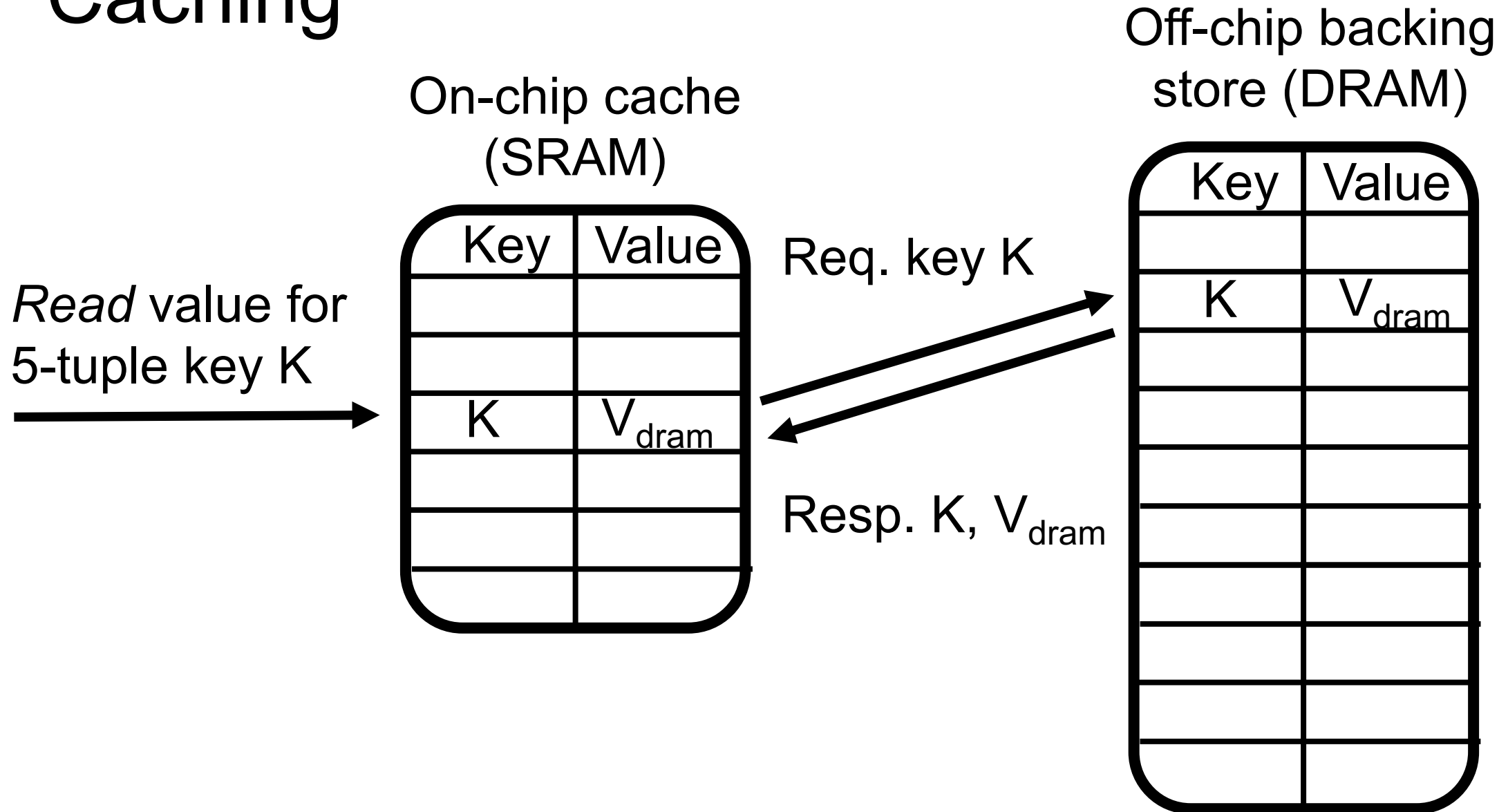




# Caching



# Caching

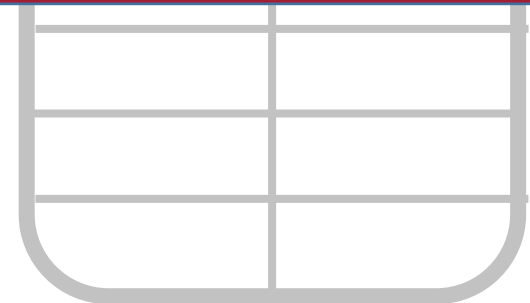


# Caching

Off-chip backing

Modify and write must wait for DRAM.

*Non-deterministic latencies stall packet pipeline.*

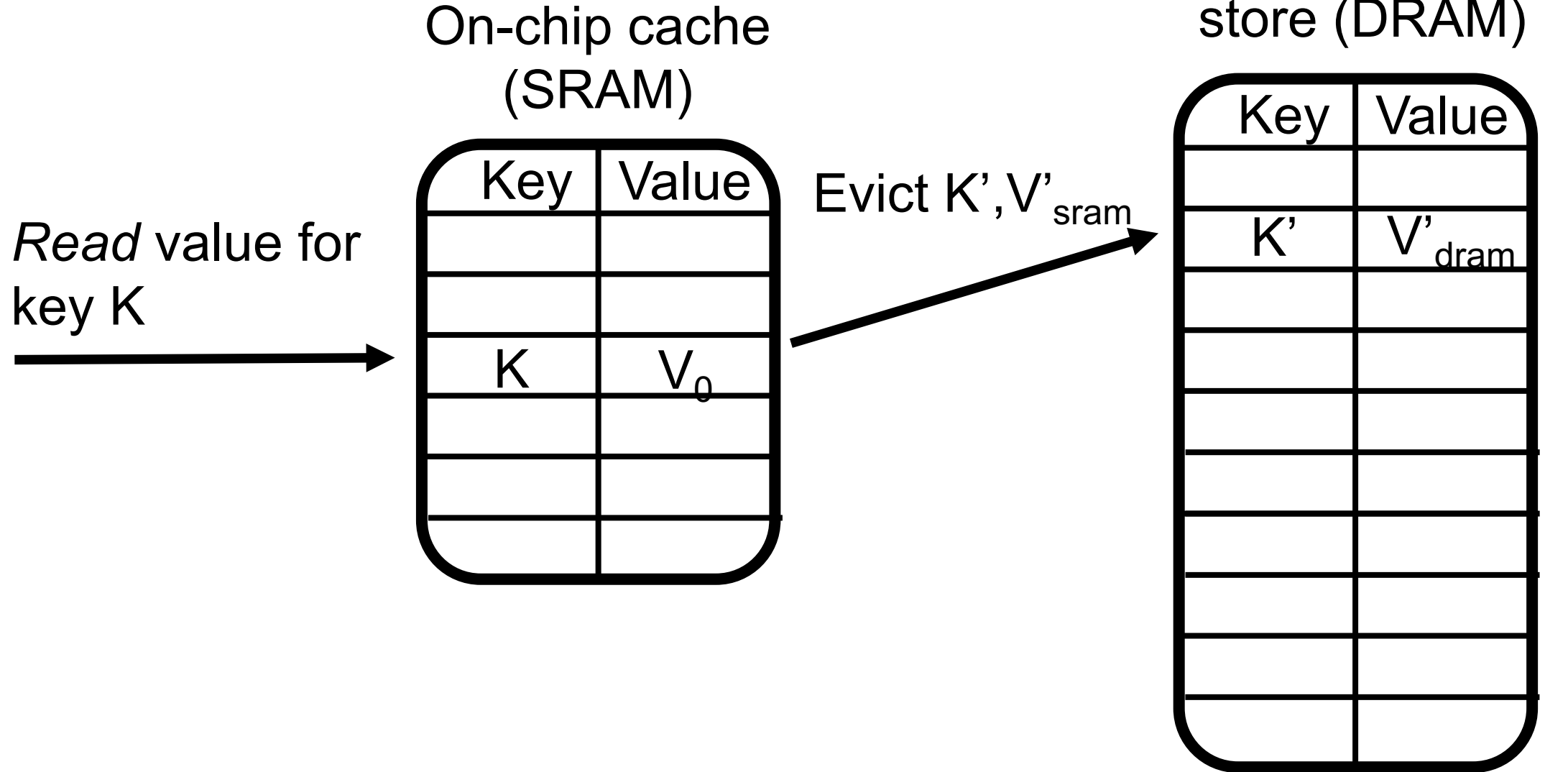




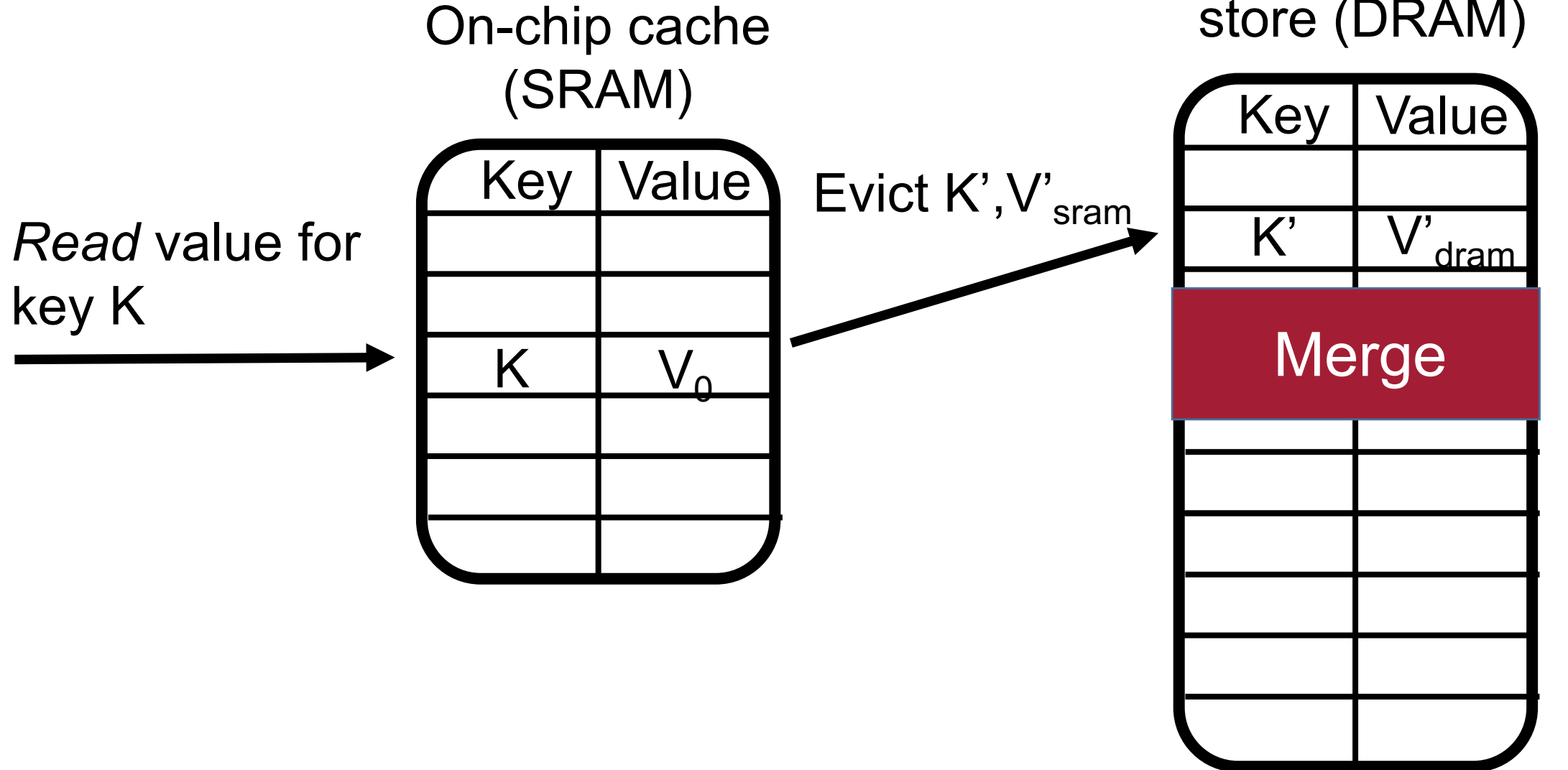
Instead, we treat cache misses as packets from new flows.



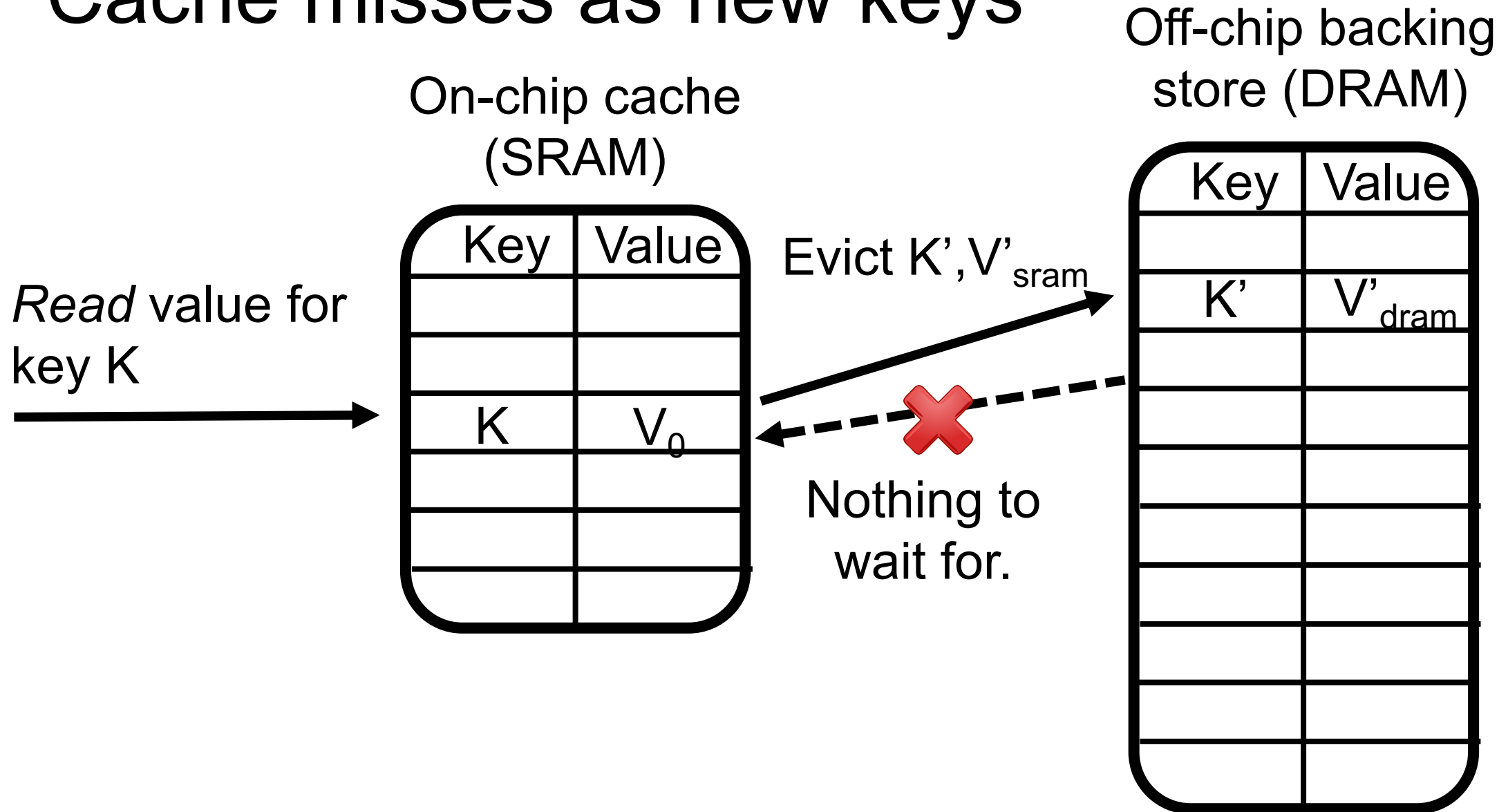
# Cache misses as new keys



# Cache misses as new keys



# Cache misses as new keys



# Cache misses as new keys

On-chip cache

Off-chip backing store (DRAM)

Packet processing doesn't wait for DRAM.

Retain 1GHz packet processing rate! 👍



# What should the merge do?

- Accumulate results of folds across evictions of a key
- Suppose: Fold function  $g$  over a packet sequence  $p_1, p_2, \dots$

$$\underbrace{g(s)}$$

Action of  $g$  over the packet sequence  $s$

# The Merge operation

$$\text{merge}( \underbrace{g(s1)}_{V_{\text{dram}}}, \underbrace{g(s2)}_{V_{\text{sram}}} ) = \underbrace{g( s1; s2 )}_{\text{Original fold over the entire packet sequence}}$$

- Example: if  $g$  is a packet counter, merge is just addition!



For general folds, it is impossible to merge accurately with small extra memory.

(formal result in paper)

# Linear-in-state: Class of mergeable folds

- Updates mergeable with small memory:

$$S = A * S + B$$

State of the fold function

Functions of a bounded number of packets in the past

- Examples: Packet and byte counters, EWMA, any functions of bounded number of packets, ...

# Linear-in-state: Merge operation

- EWMA :  $S = A * S + B$  where A and B are *constants*

$$\text{merge}(V_{\text{dram}}, V_{\text{sram}}) = V_{\text{sram}} + (A^N) * (V_{\text{dram}} - V_0)$$

**System feasibility**

# Is the system design feasible?

- Cache design feasibility

- Hash-based lookup ✓
- Value update operations ✓
- Cache eviction logic ✓
- SRAM area

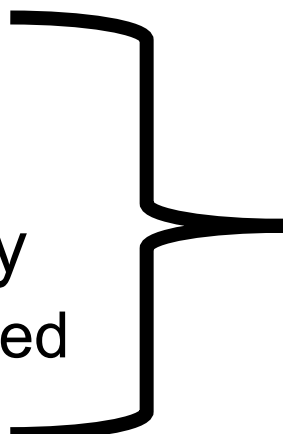
Match-action tables

Multiply-accumulate instruction (<10 stages)

LRU processor caches

- Backing store feasibility

- Tuples/second processed

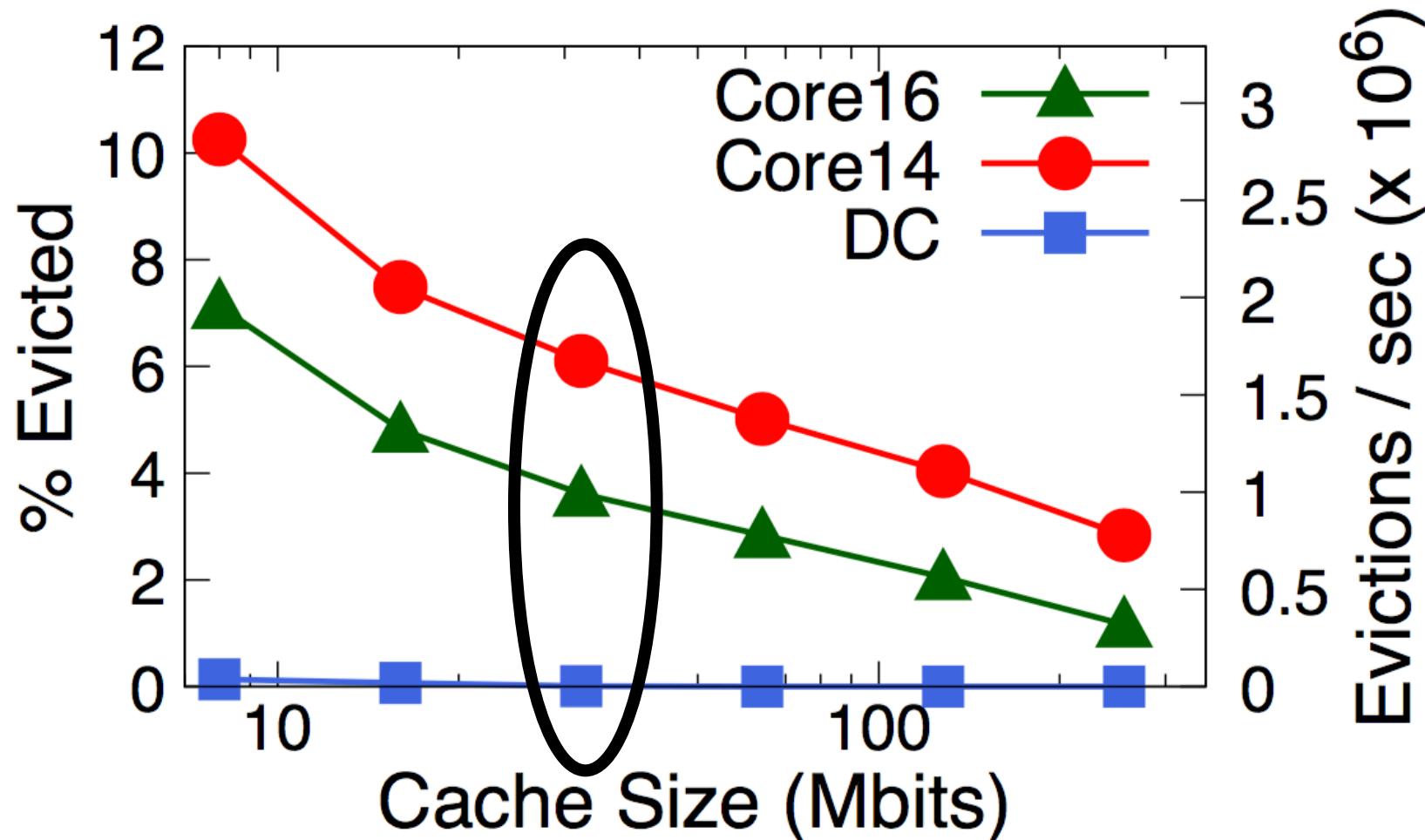


Tradeoff: Trace-driven evaluation

# Measuring Evictions vs. Cache size

- Core router and data center traces with ~100M packets each
  - Core routers from 2014, 2016 and university data center from 2010
- 64-port 10Gbit/s switch
- Query aggregates by 5-tuple with 24-bit state
- Evictions vs. 8-way LRU cache memory sizes

# Measuring Evictions vs. Cache size



1.67M tuples/s  
with a 32Mbit  
cache.

2.5% area  
overhead

Feasible to  
process evictions  
with state of the  
art KV stores

# Takeaways

- Design expressive language to identify flexible hardware primitives
- Linear in state: fully accurate per-flow aggregation at line rate

[alephtwo@csail.mit.edu](mailto:alephtwo@csail.mit.edu)

