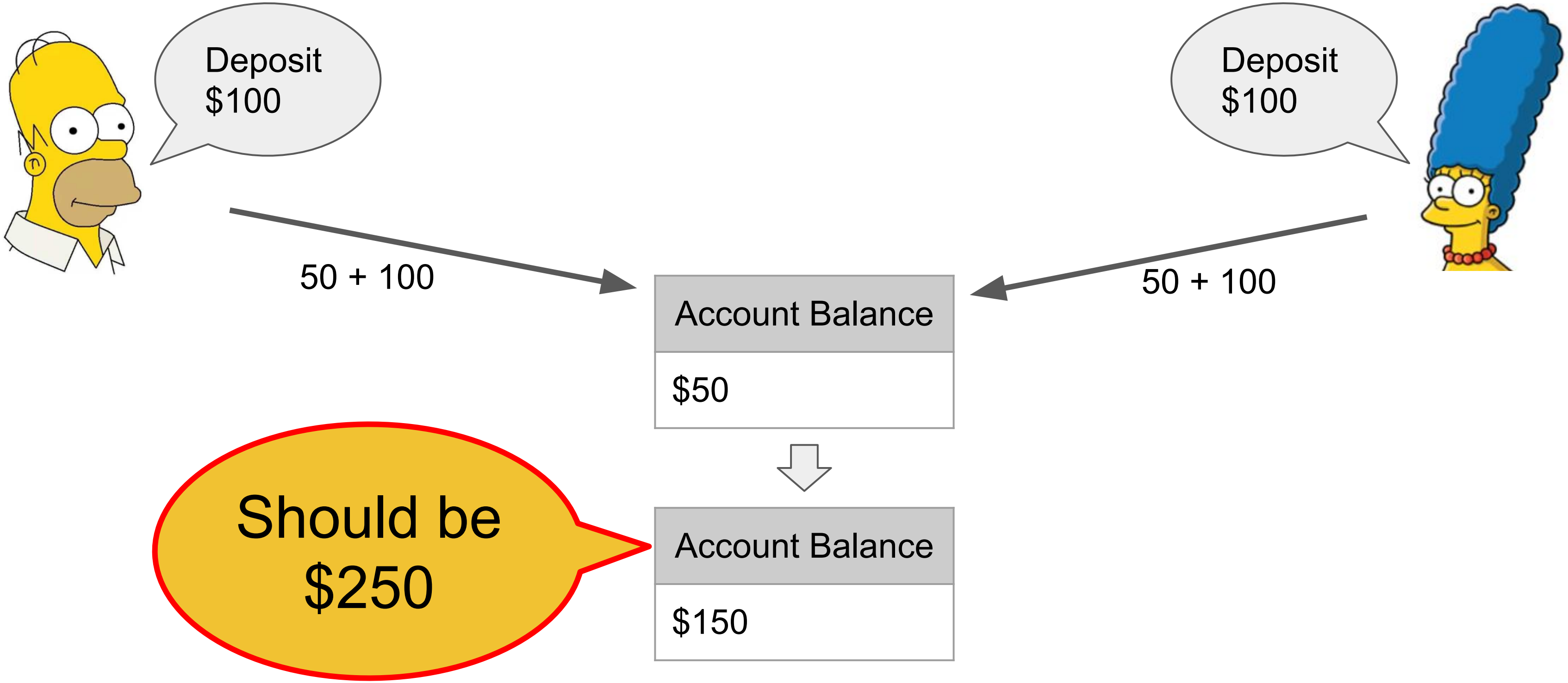
Infinite Resources for Optimistic Concurrency Control with NOCC

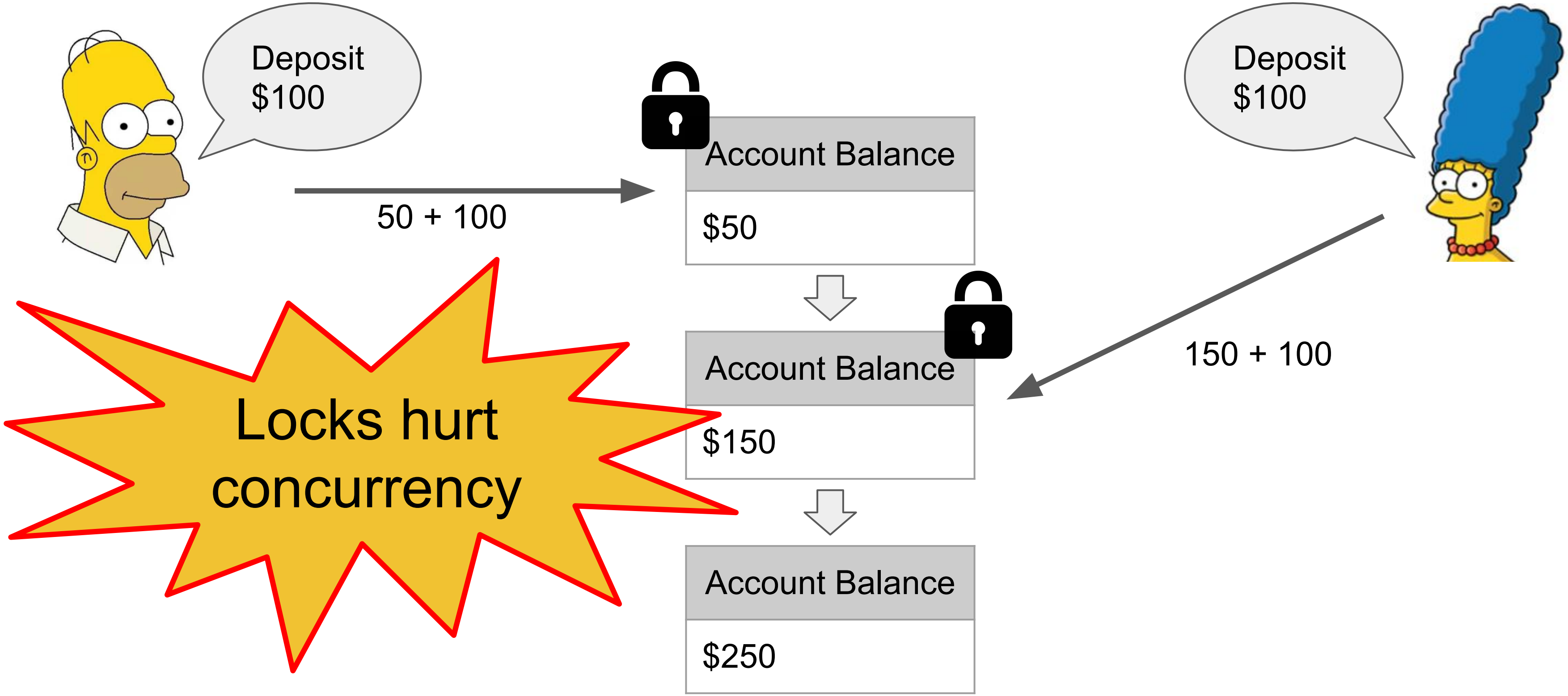
Theo Jepsen, Leandro Pacheco de Sousa, Masoud Moshref, Fernando Pedone, Robert Soulé

Università della Svizzera italiana (USI) and Barefoot Networks

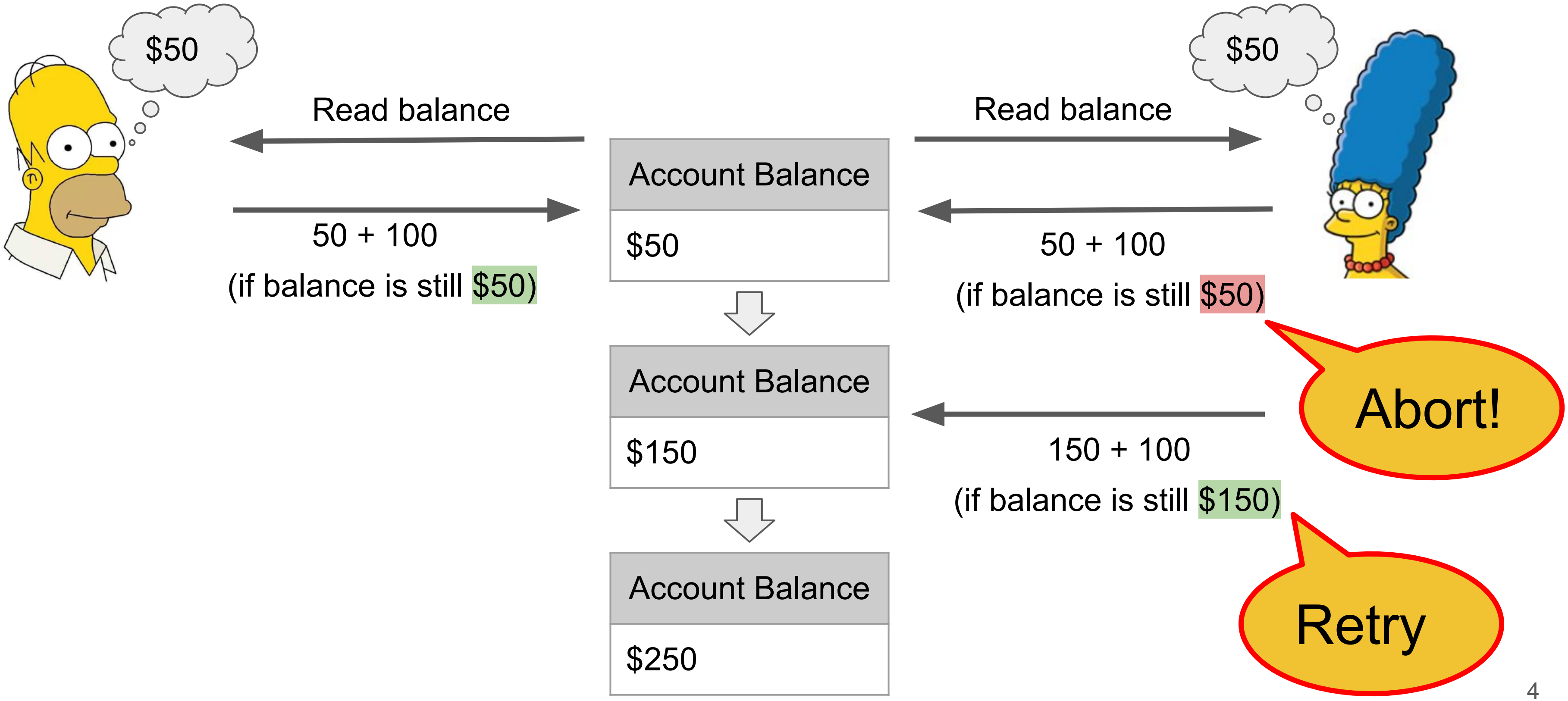
Why Do We Need Concurrency Control?



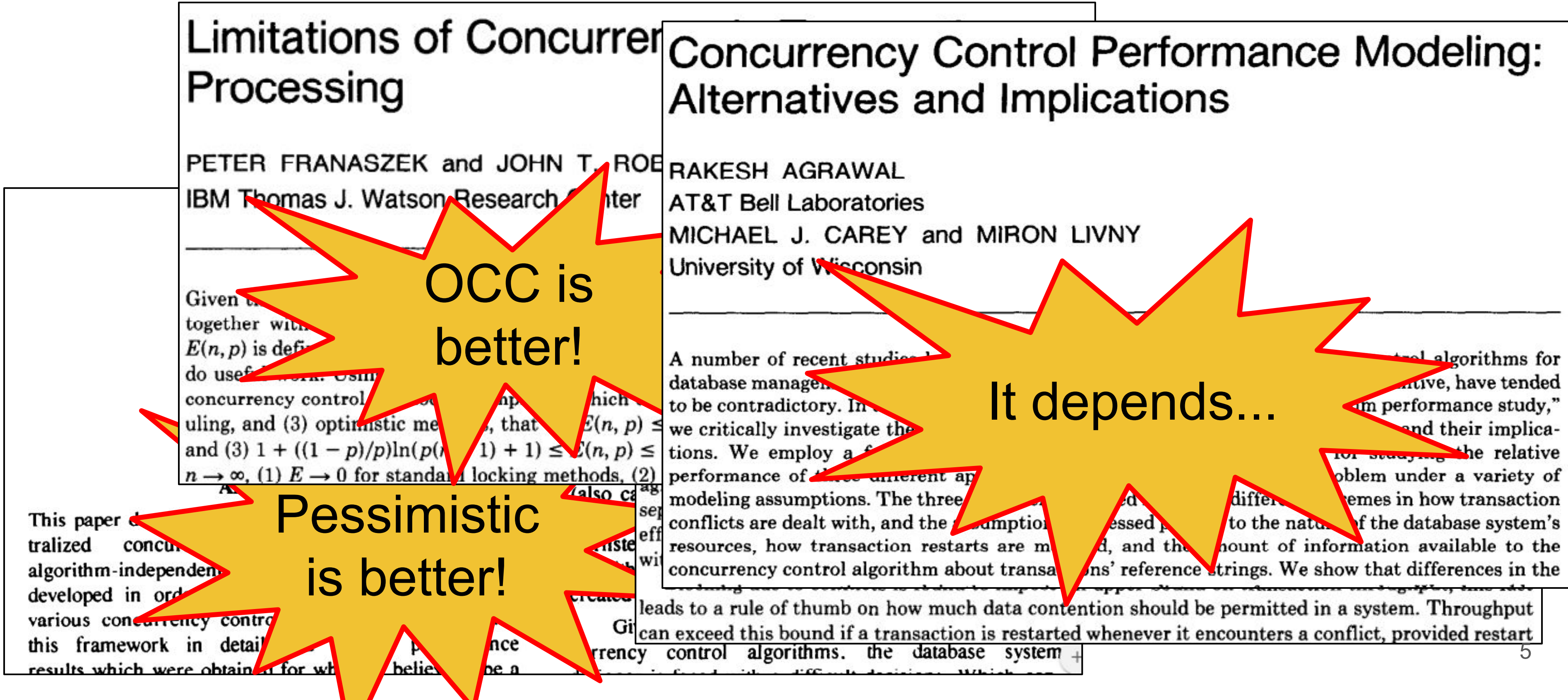
Pessimistic Concurrency Control



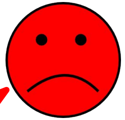
Optimistic Concurrency Control



Pessimistic vs Optimistic Concurrency Control

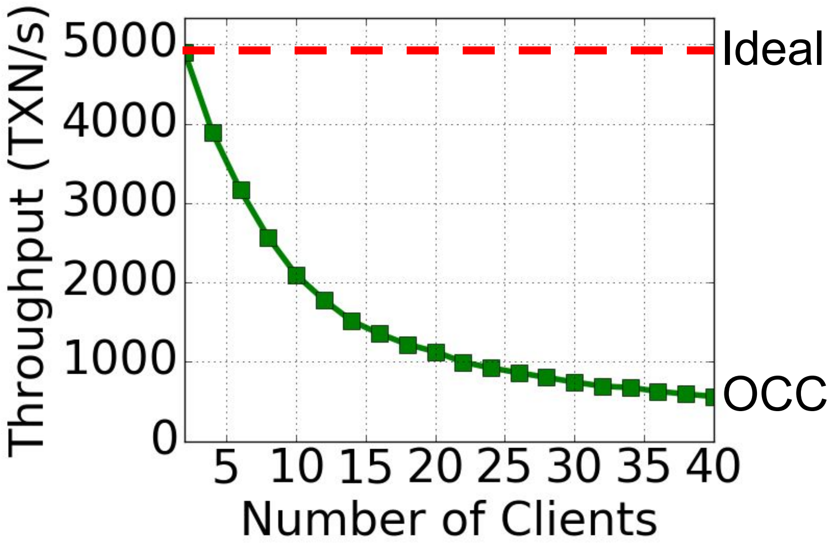


Pessimistic vs Optimistic Concurrency Control

	Pessimistic	Optimistic
Low contention		
High contention		

Aborts reduce throughput

OCC: Aborts are Expensive



more clients→contention→more aborts→lower tput

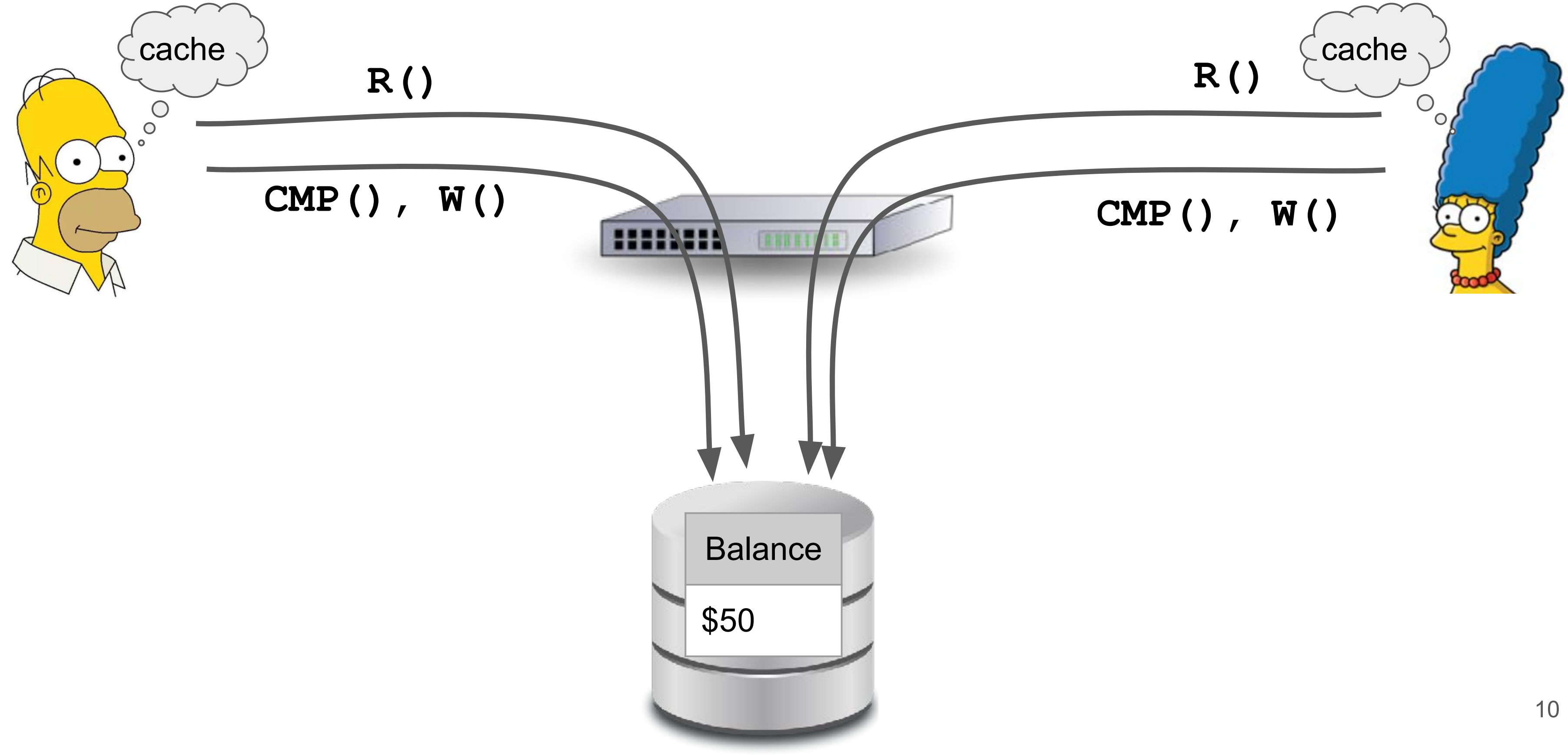
OCC With Infinite Resources

- What if we had infinite CPUs to abort transactions?
- Hardware can process aborts virtually instantly
- This hardware is already in the network

Network OCC (NOCC)

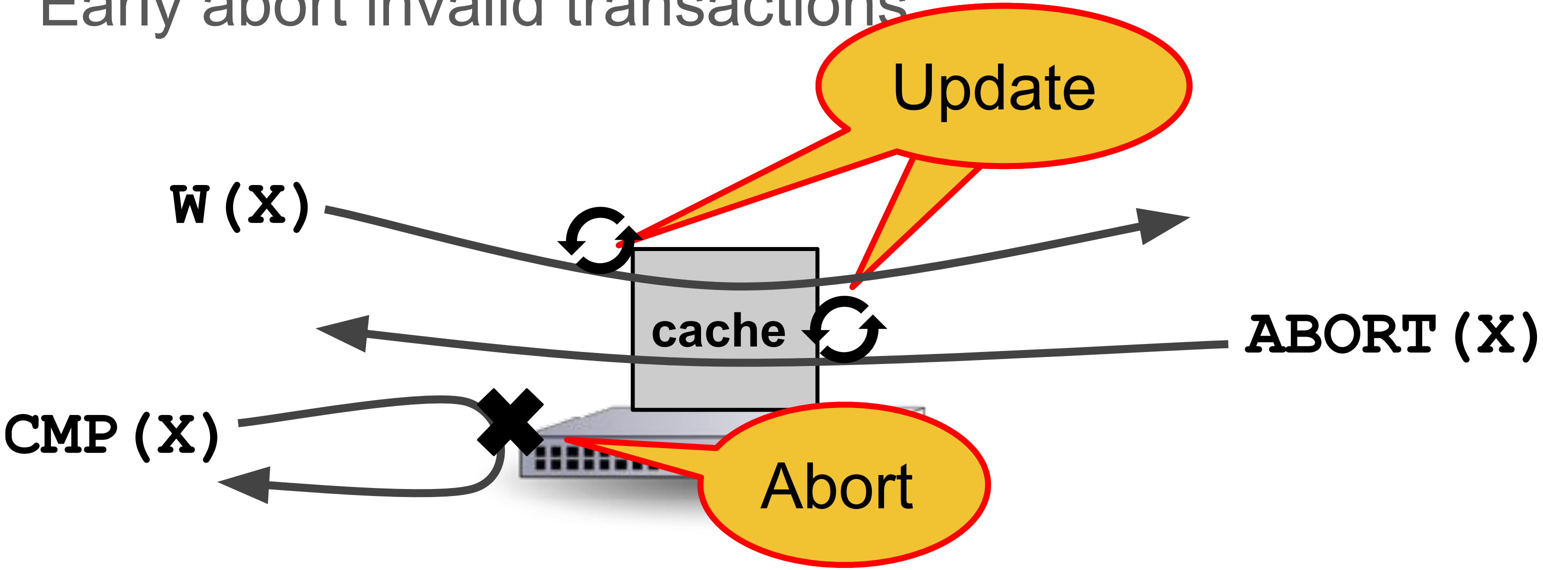
- Offload transaction verification to the switch
- High parallelism for high-contention workloads
- Reduces server load for workloads (like TPC-C)

System Model

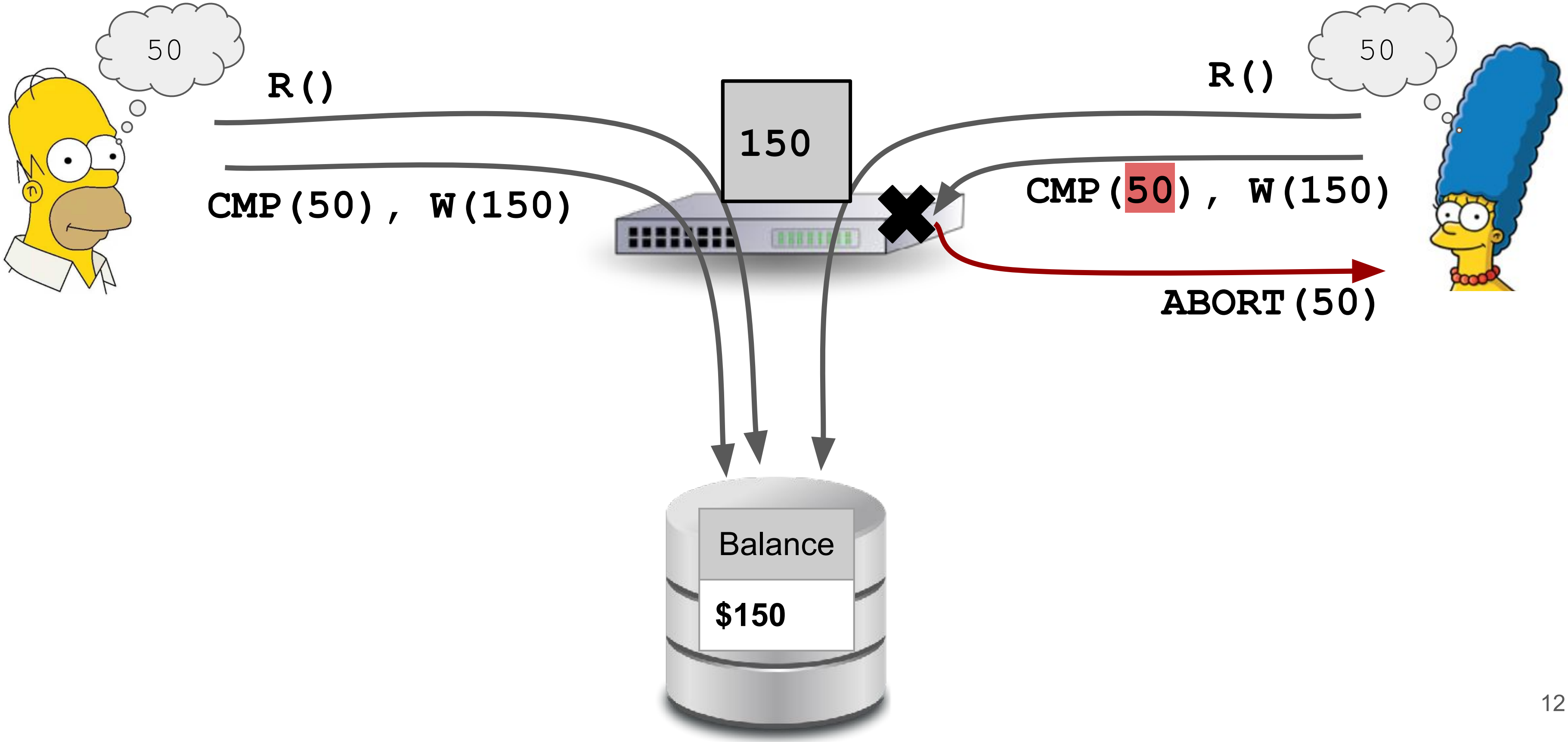


The NOCC Approach

- Update cache with write values
- Update cache with ABORT values
- Early abort invalid transactions



NOCC Example



NOCC Correctness

- Strong consistency:
  - Reads are not handled by switch – no stale reads
- Liveness:
  - Transactions eventually commit

Implementation

Switch Implementation: Key Challenges

- Storing cached values on the switch
- Processing packet headers containing transactions

Processing Transactions

- Each transaction contains one or more operations:
  - `read()`, `cmp()`, `write()`
- The P4 program iterates over the operations:
  - If invalid `cmp()`, abort transaction
  - If `write()`, update cache
- P4 doesn't have iteration primitives
  - So we recirculate the packet

Switch Cache

- We use SRAM registers
- Values (128 bits) are too large for a single register
  - So we shard the value across multiple registers

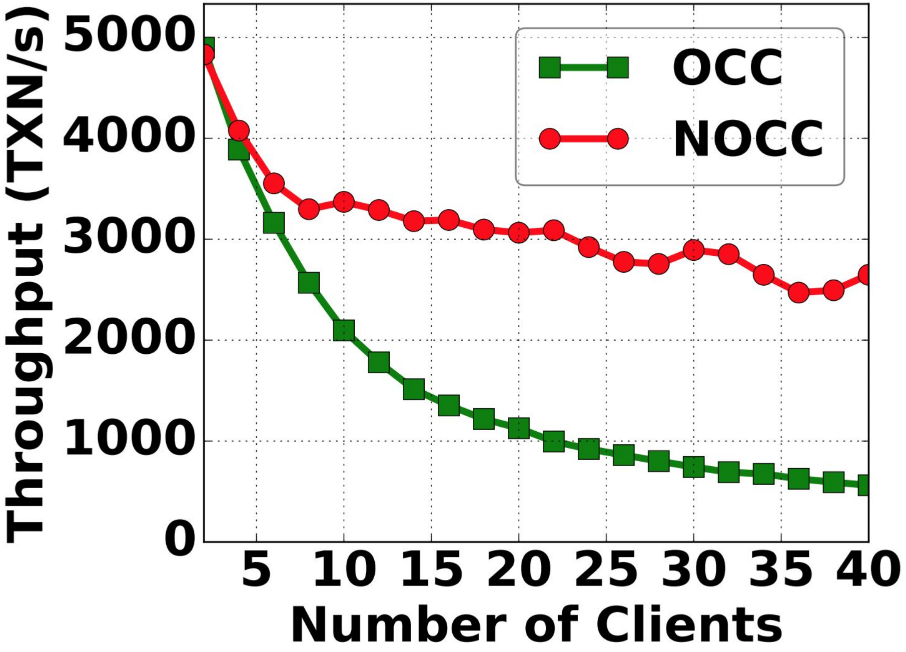
Reg1	Reg2	Reg3	Reg4
val[0...31]	val[32...63]	val[64...95]	val[95...128]

Evaluation on Hardware

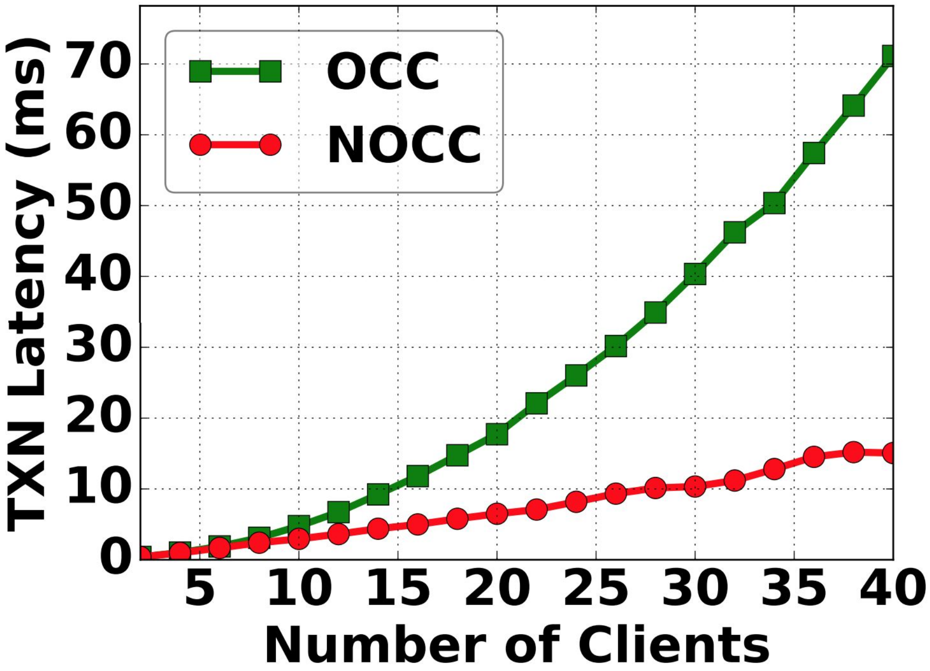
Experimental Setup

- Clients and store run on seperate servers
- Connected via a Barefoot Tofino switch running NOCC
- Evaluated with microbenchmarks and TPC-C

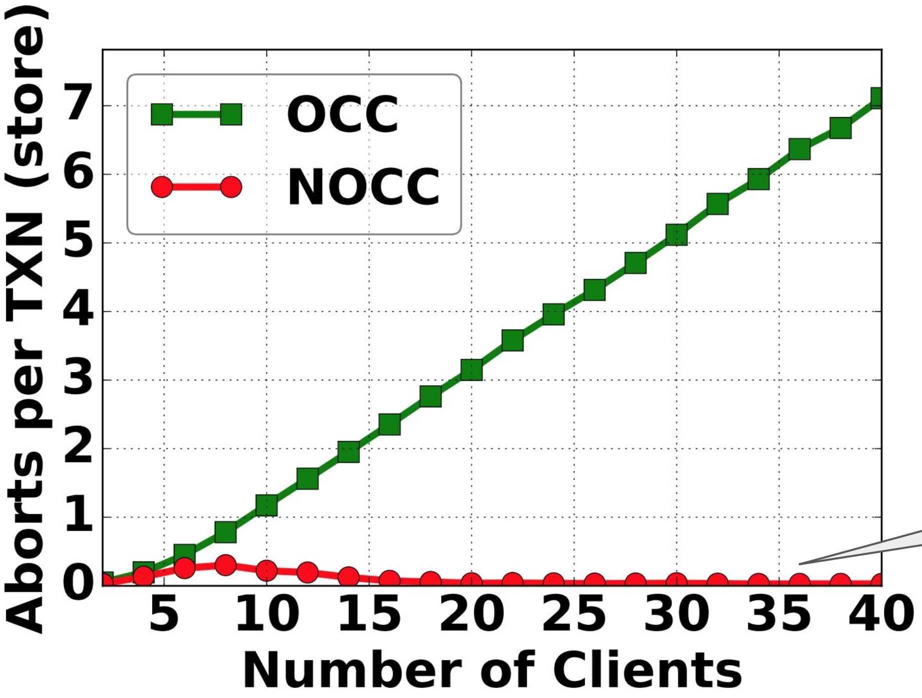
NOCC has Higher Throughput



NOCC Reduces End-to-End Latency

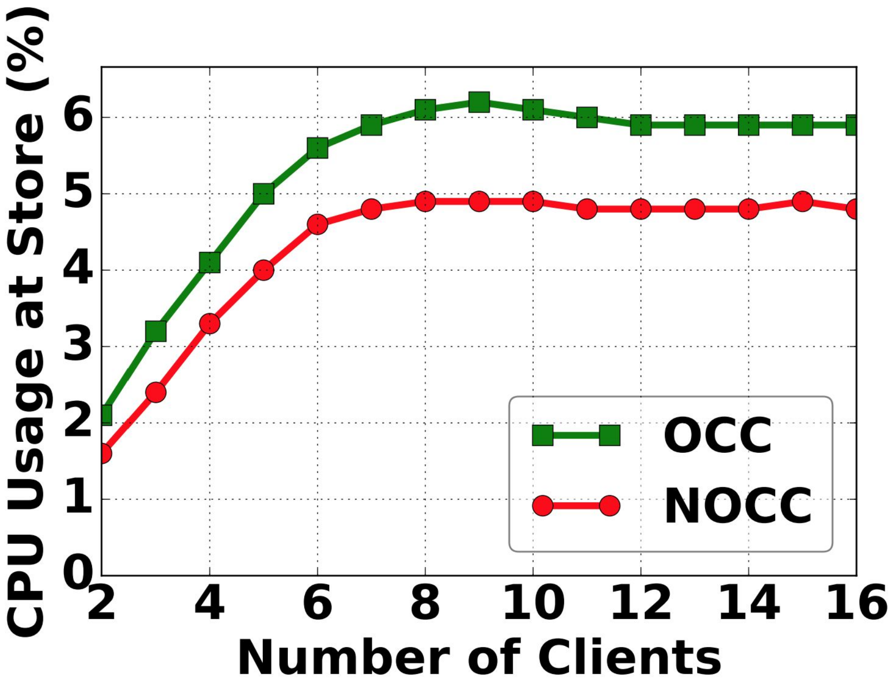


NOCC Reduces Aborts from the Store

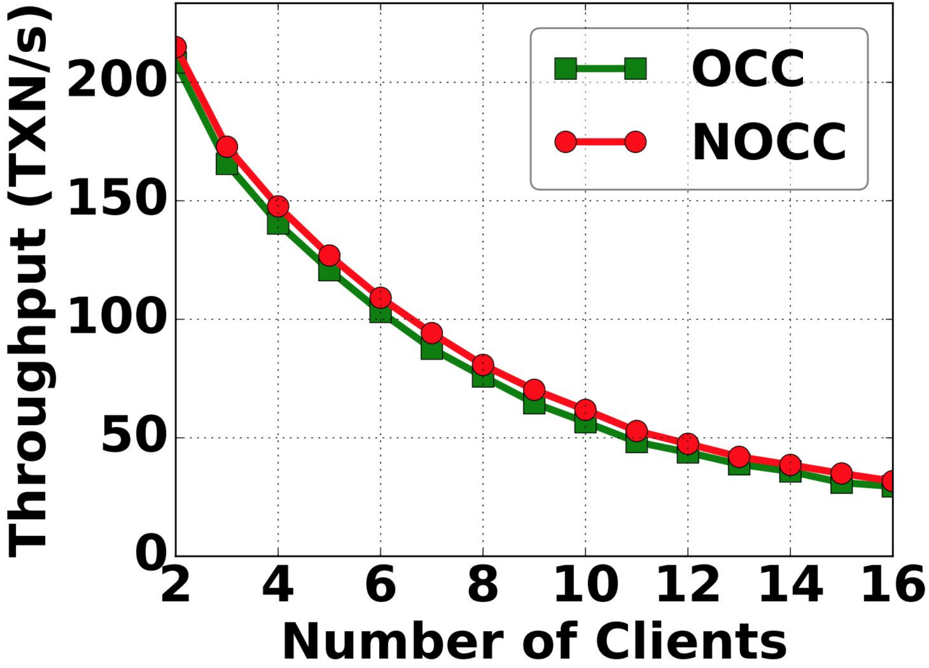


Commits all transactions

NOCC Reduces Server Load for TPC-C



Minimal Throughput Overhead for TPC-C



In Conclusion, NOCC...

- Offloads transaction verification logic to the network
- Provides high throughput under high contention
- Reduces CPU load on the server

https://github.com/usi-systems/nocc

Extra Slides

Packet Header Format

```
header_type nocc_hdr_t {
  fields {
    bit<1> msg_type; // REQ/RES
    bit<1> from_switch;
    bit<32> txn_id;
    bit<8> frag_seq;
    bit<8> frag_cnt;
    bit<8> status;
    bit<8> op_cnt;
  }
}
```

Number of following nocc_op headers

The nocc_hdr is followed by a nocc_op header for each operation

```
header_type nocc_op_t {
  fields {
    bit<8> op_type;
    bit<32> key;
    bit<1024> value;
  }
}
```