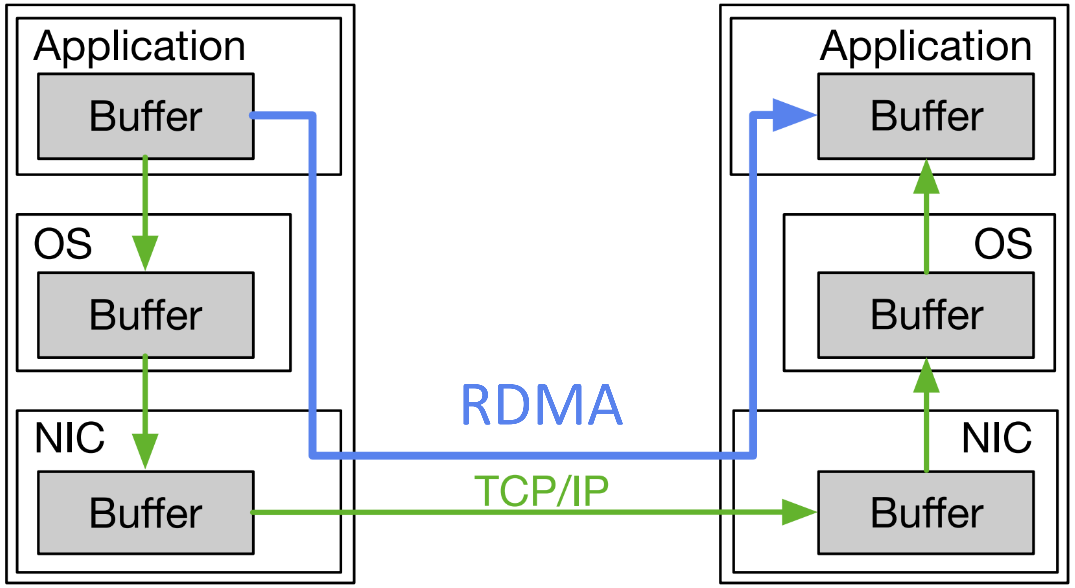
RDMA is Turing complete, we just did not know it yet!

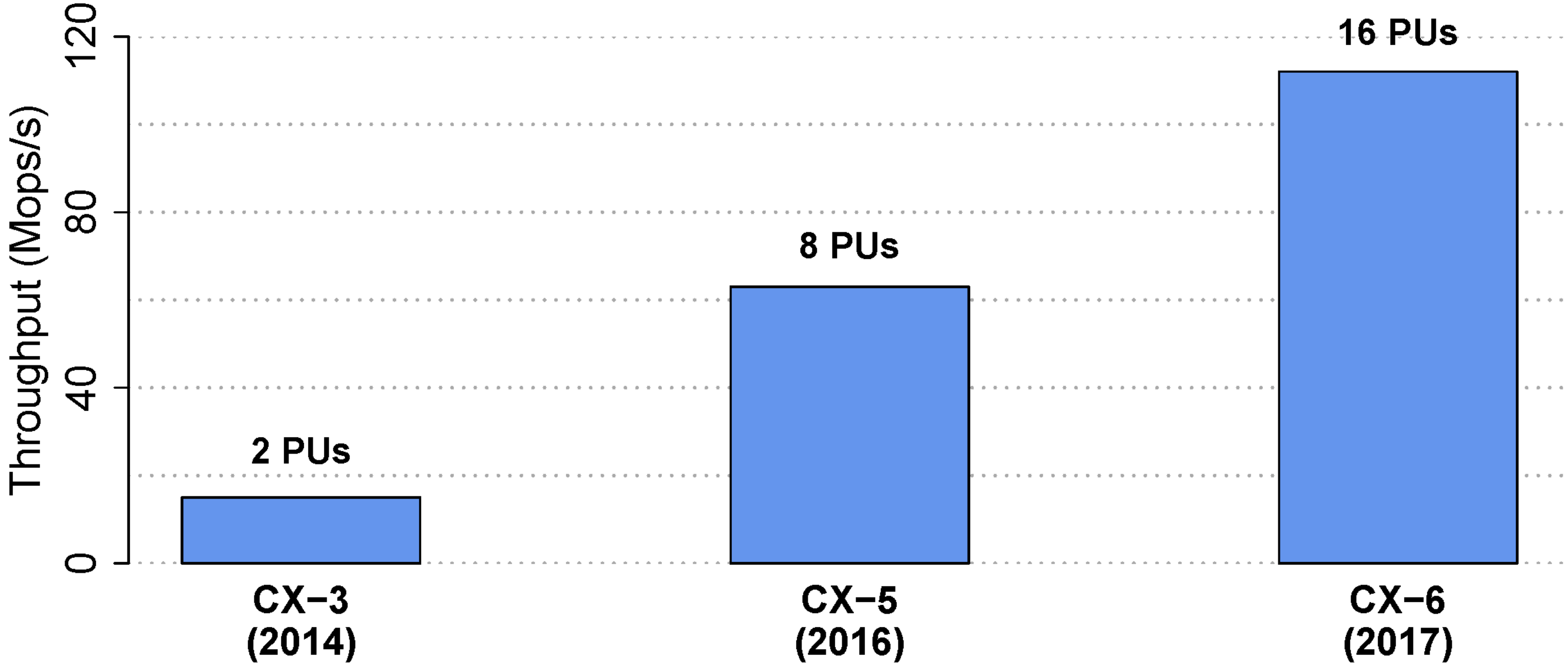
Waleed Reda, Marco Canini, Dejan Kostić, Simon Peter

Benefits of RDMA networking

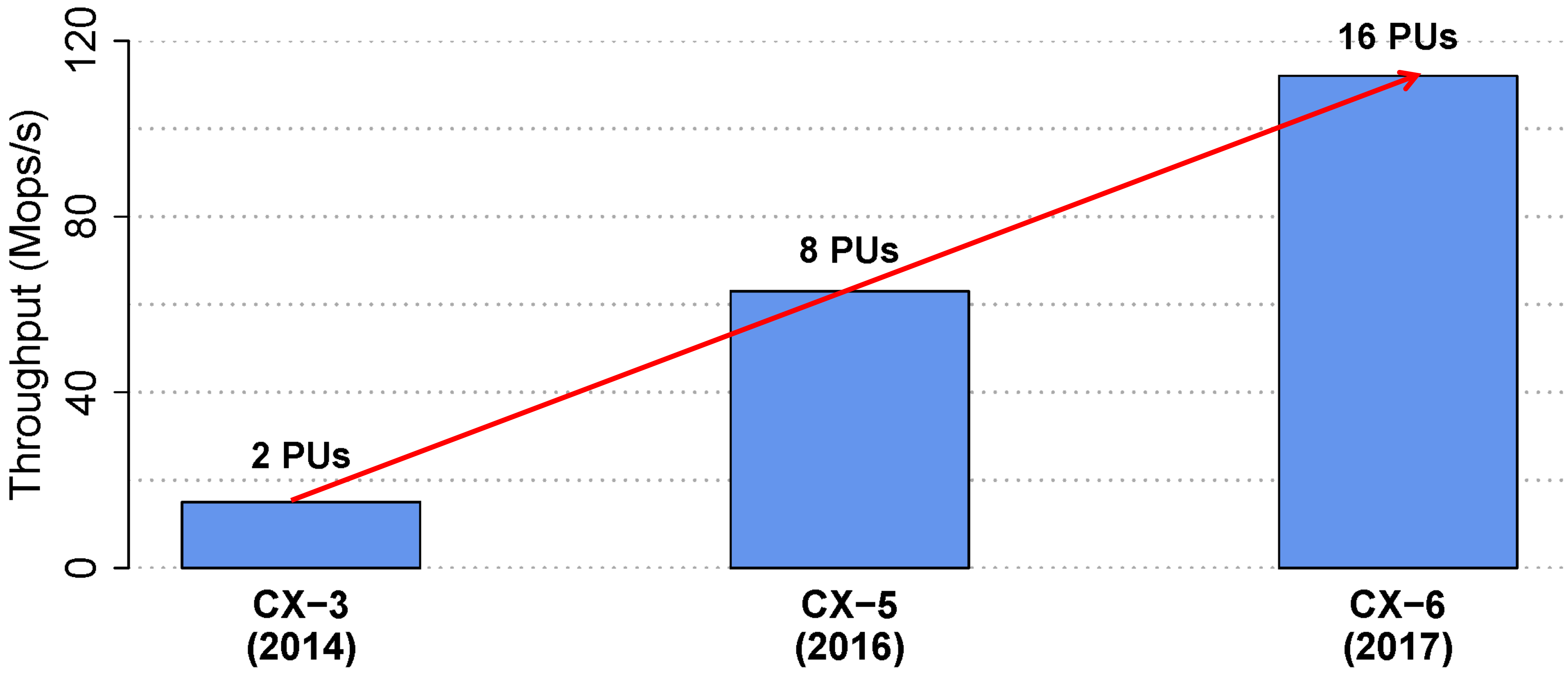
- Bypasses the kernel and allows zero-copy data transfers
- Offers one-sided operations
  - e.g. RDMA READ or RDMA WRITE
- Requires no CPU involvement
  - **But can only perform simple memory transfers!**



Massive growth in RDMA processing power



Massive growth in RDMA processing power

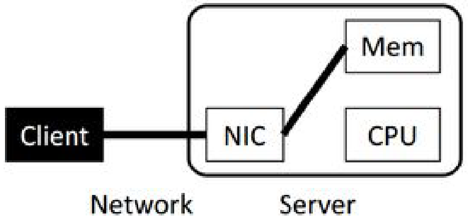


Almost 2x increase / year!

Existing designs for RDMA-based systems

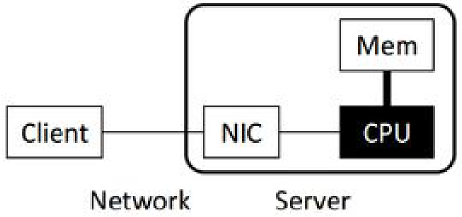
Commodity RNIC offloads

One-sided (e.g. FaRM-KV)



Limited by RDMA API.
Incurs extra roundtrips to serve requests

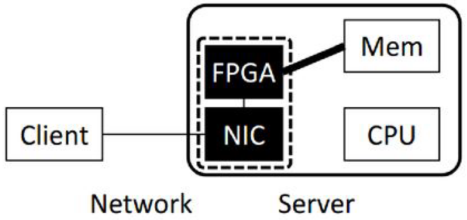
Two-sided (e.g. HERD)



Requires remote CPU involvement.

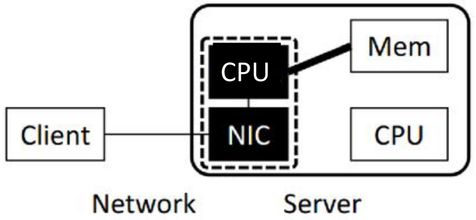
Smart NIC offloads

FPGA-offload (e.g. KV-DIRECT)



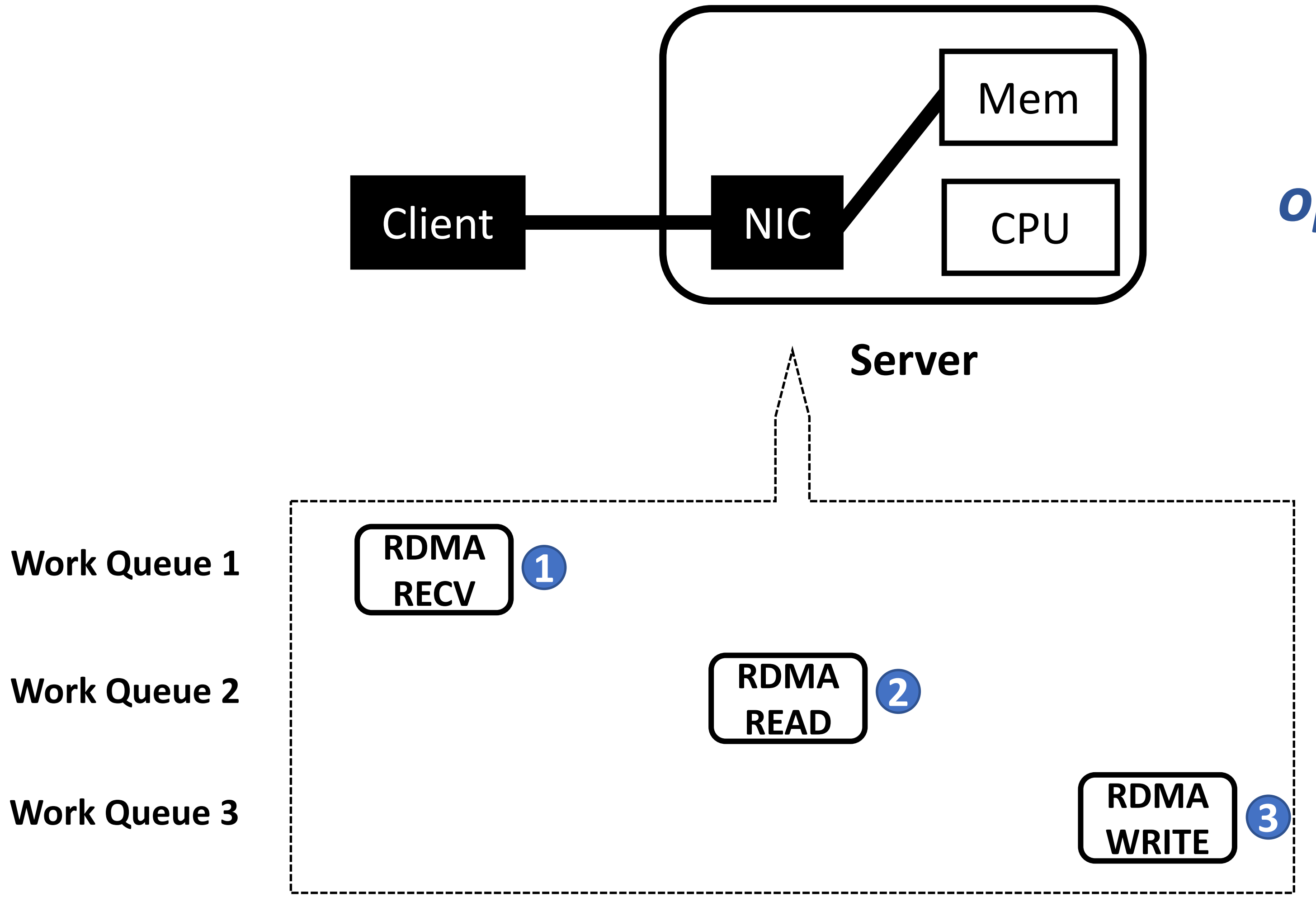
Expensive and difficult to program

NIC-CPU offload (e.g. LineFS)



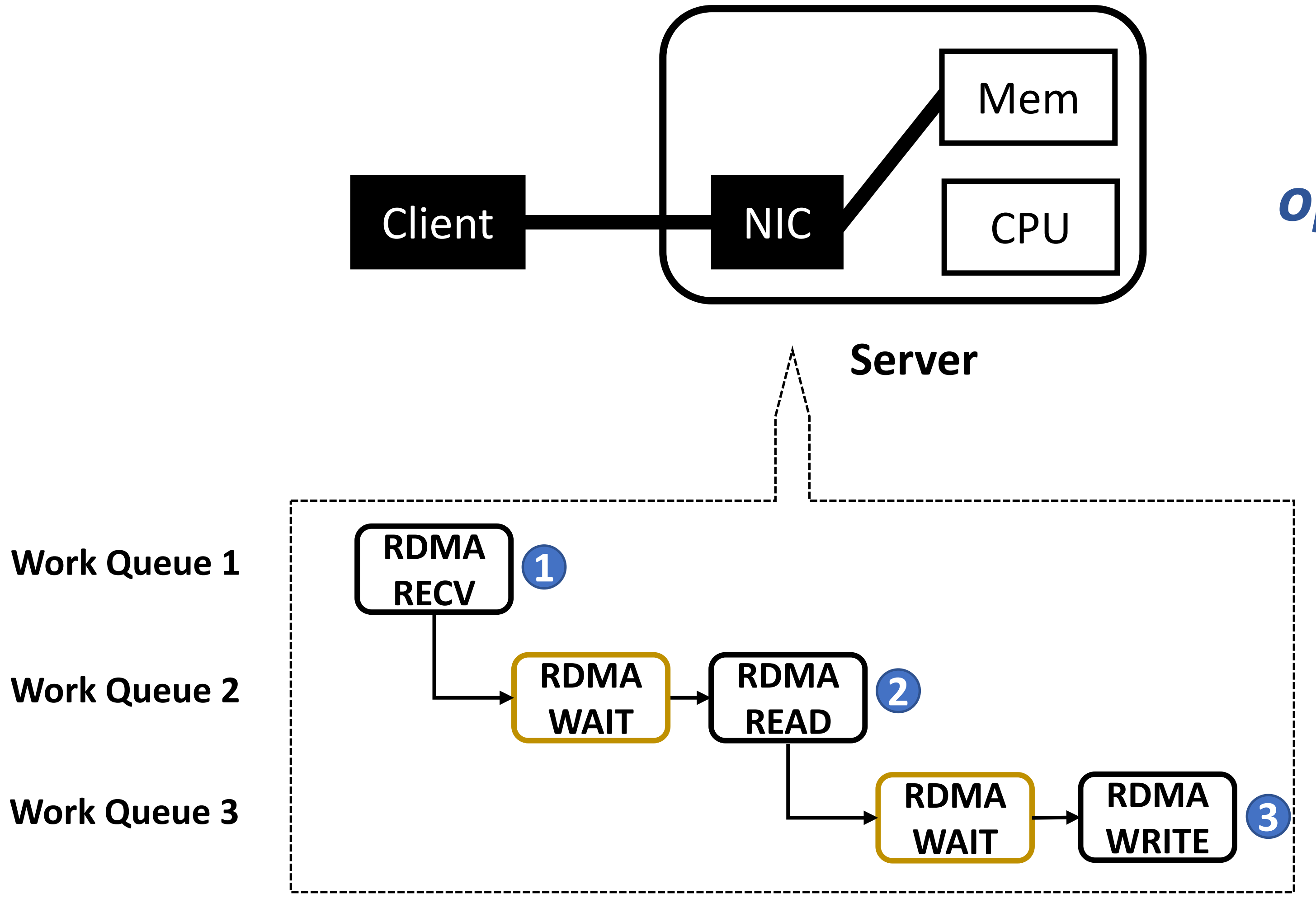
Expensive and uses slow “wimpy” cores.

Alternative Design: Exploit RNIC Processing Power



Insight #1: Perform complex operations using RDMA chains

Alternative Design: Exploit RNIC Processing Power

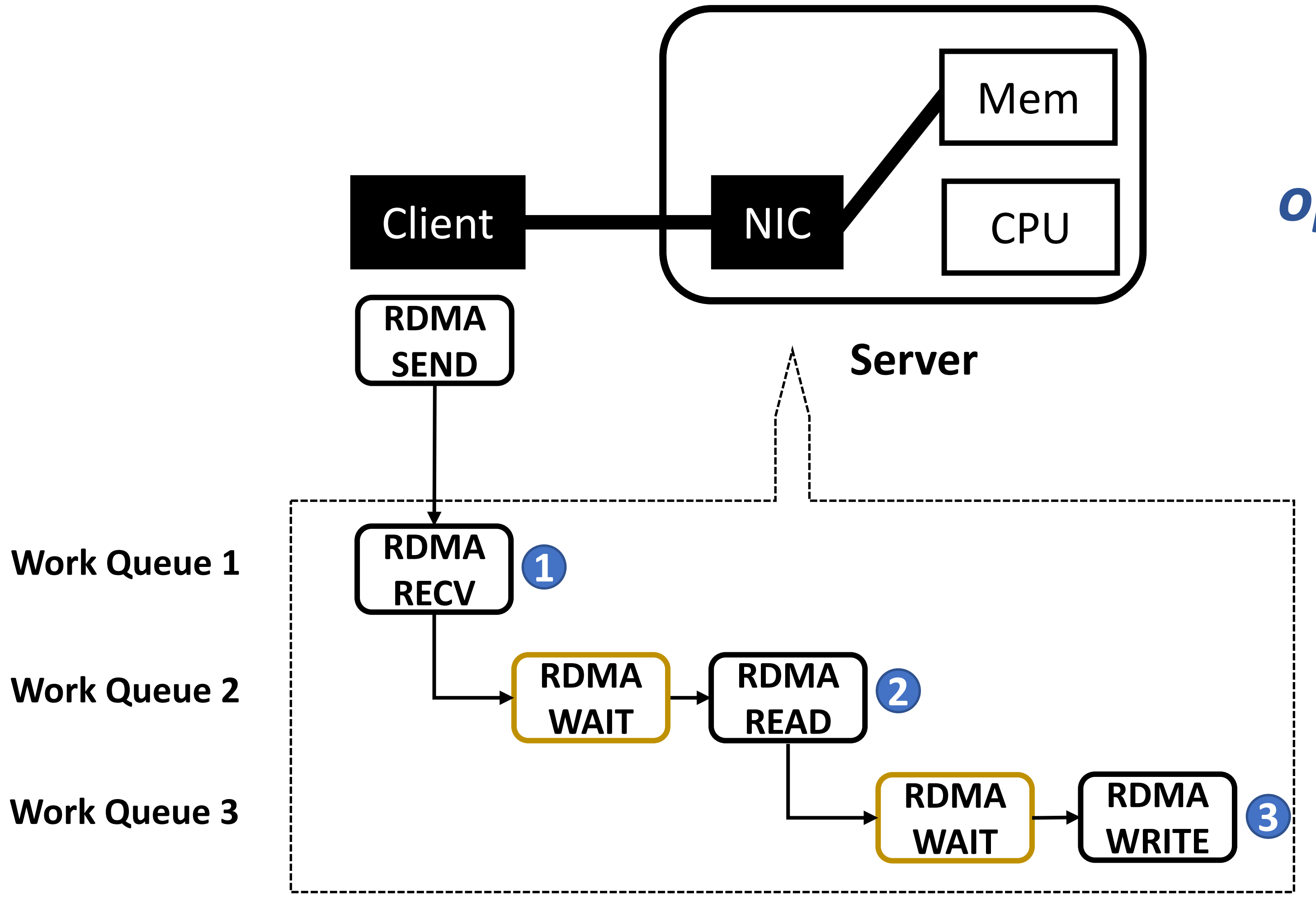


Insight #1: Perform complex operations using RDMA chains

Use **RDMA WAIT** feature

✓ Adds execution dependency between operations

Alternative Design: Exploit RNIC Processing Power

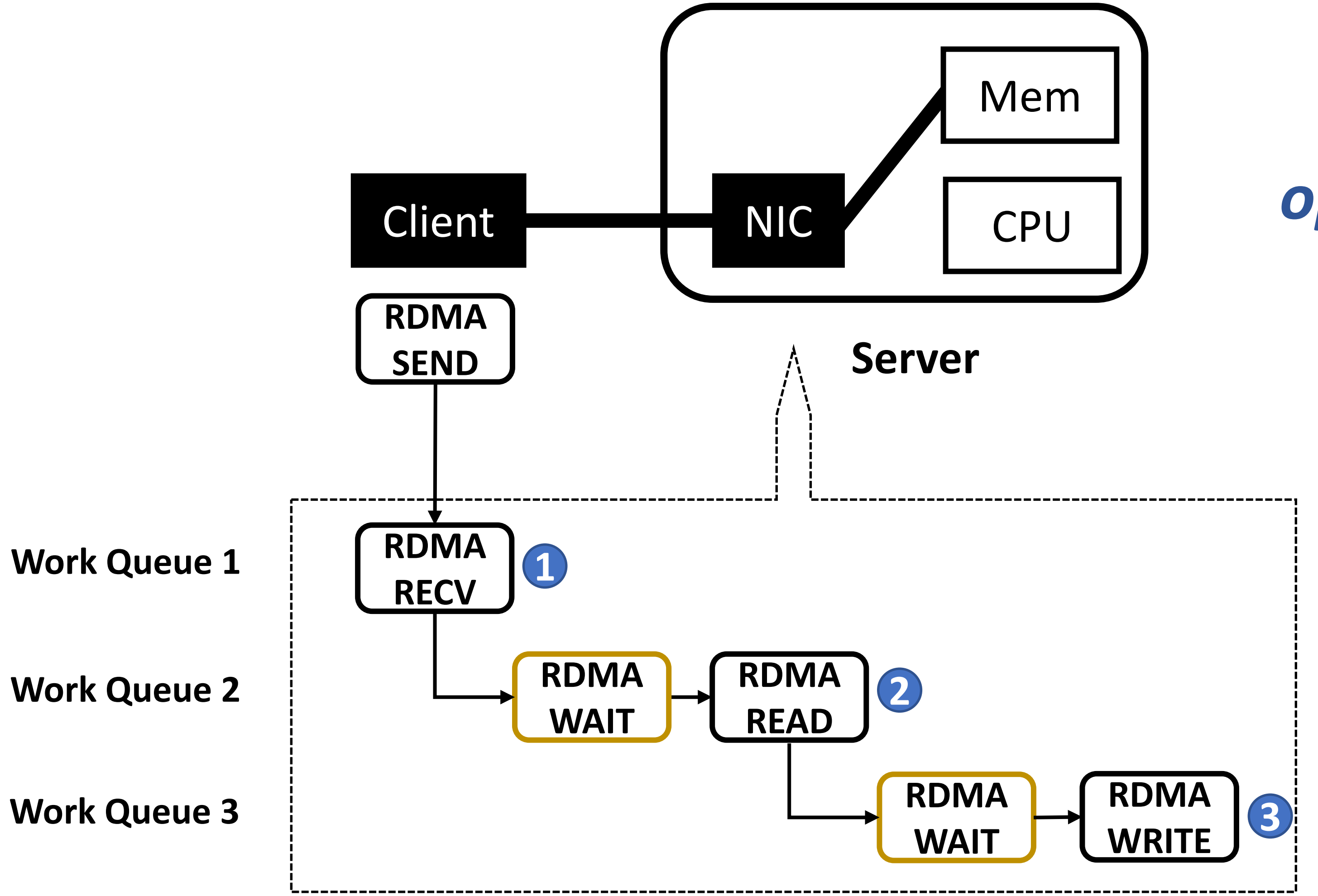


Insight #1: Perform complex operations using RDMA chains

Use **RDMA WAIT** feature

- ✓ Adds execution dependency between operations
- ✓ Allows clients to trigger server RDMA code

Alternative Design: Exploit RNIC Processing Power



Rich API for offloads + Uses commodity RNICs

Insight #1: Perform complex operations using RDMA chains

Use **RDMA WAIT** feature

- ✓ Adds execution dependency between operations
- ✓ Allows clients to trigger server RDMA code

But is this Turing complete?

- So far, we only managed to construct an imperative language for RDMA NICs

- To be Turing complete, two requirements must be met:
  - **R1** *The ability to read/write to an arbitrary amount of memory*
  - **R2** *Conditional branching (e.g. support for if/else statements)*

But is this Turing complete?

- So far, we only managed to construct an imperative language for RDMA NICs

- To be Turing complete, two requirements must be met:
  - **R1** *The ability to read/write to an arbitrary amount of memory*
  - **R2** *Conditional branching (e.g. support for if/else statements)*

But is this Turing complete?

- So far, we only managed to construct an imperative language for RDMA NICs

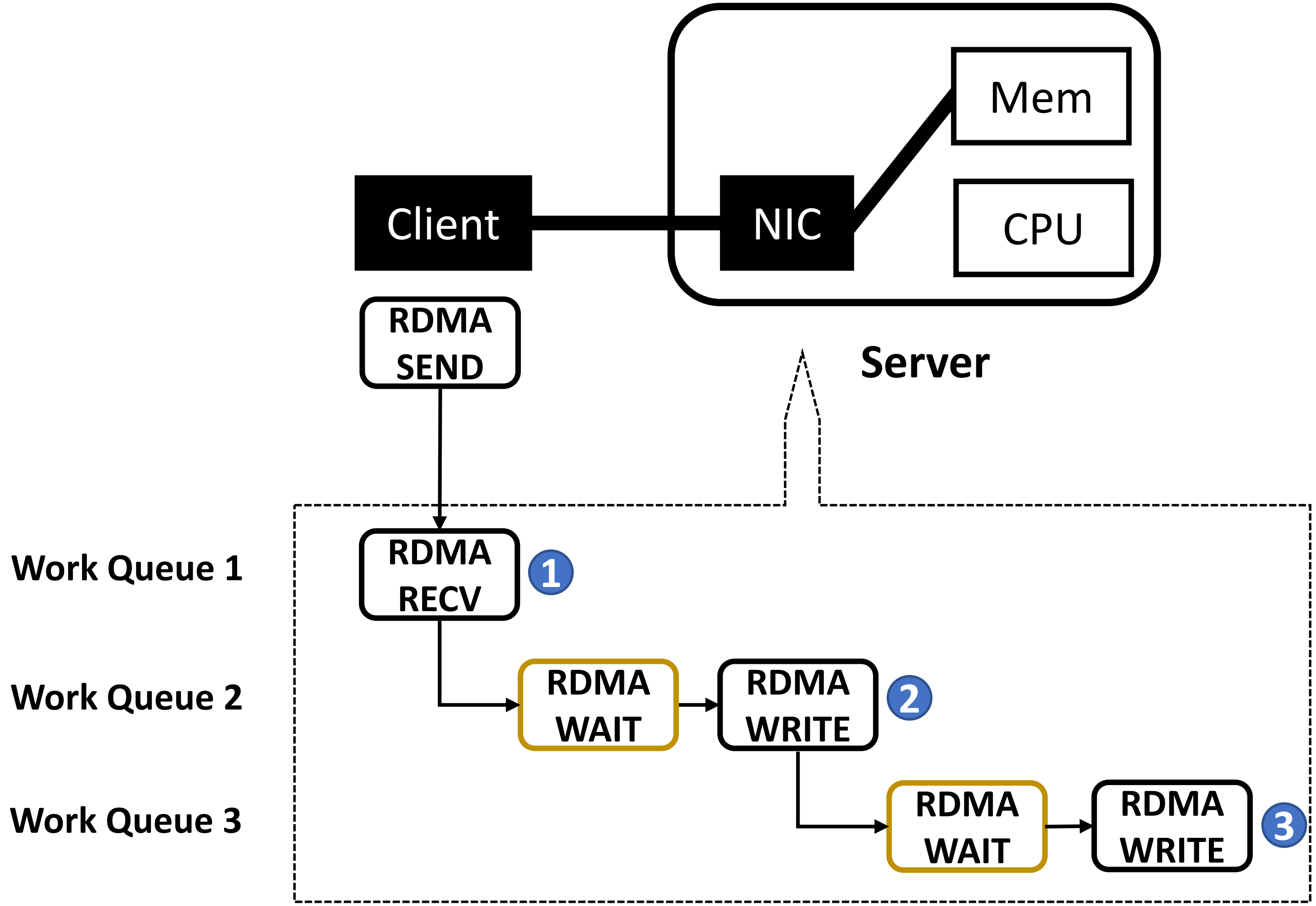
- To be Turing complete, two requirements must be met:
  - **R1** *The ability to read/write to an arbitrary amount of memory*
  - **R2** *Conditional branching (e.g. support for if/else statements)*

But is this Turing complete?

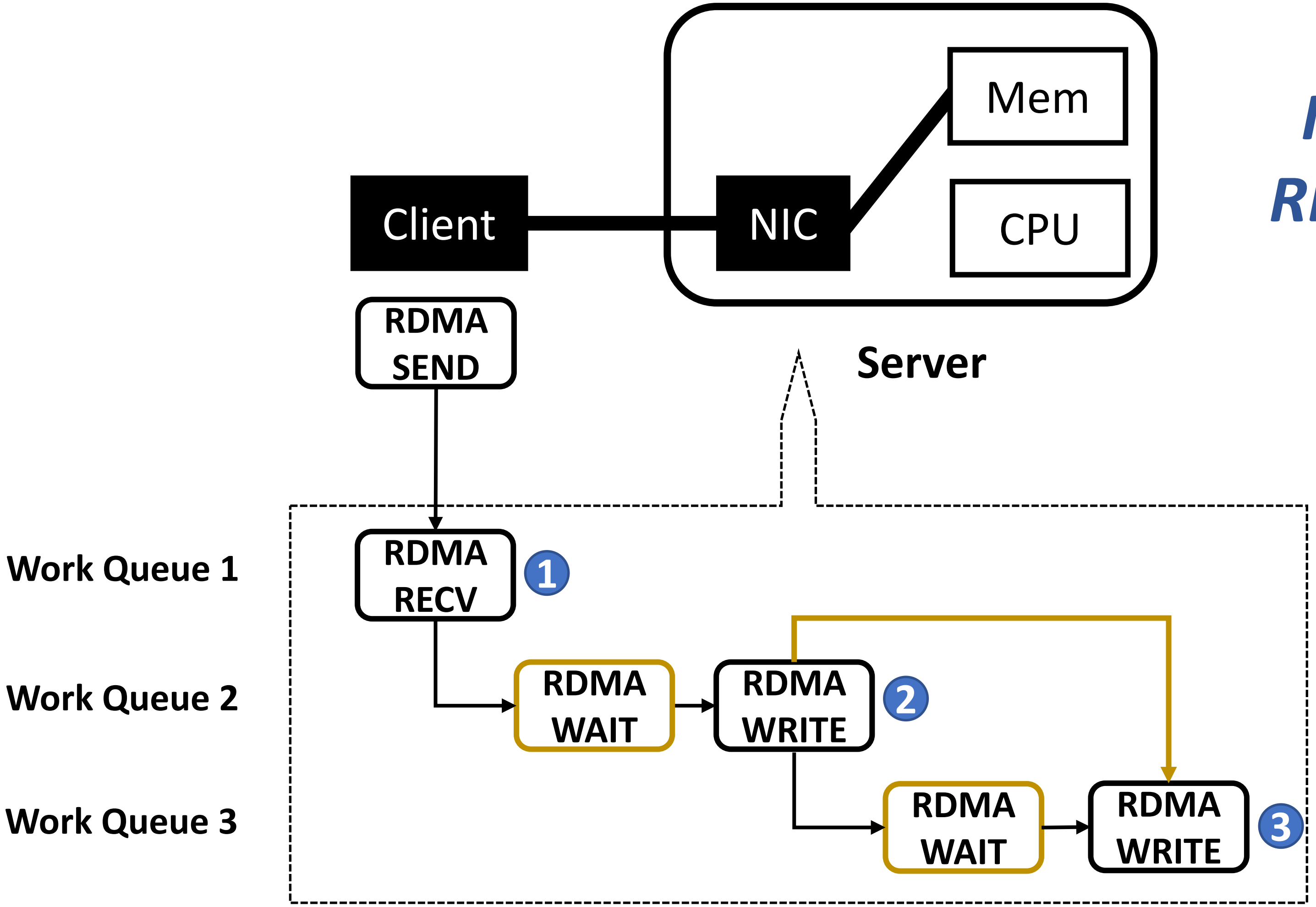
- So far, we only managed to construct an imperative language for RDMA NICs

- To be Turing complete, two requirements must be met:
  - **R1** *The ability to read/write*
  - **R2** *Conditional branching (e.g. support for if/else statements)*
  - **R3** *Support for loops or recursion*

Conditional Branching – is it possible?

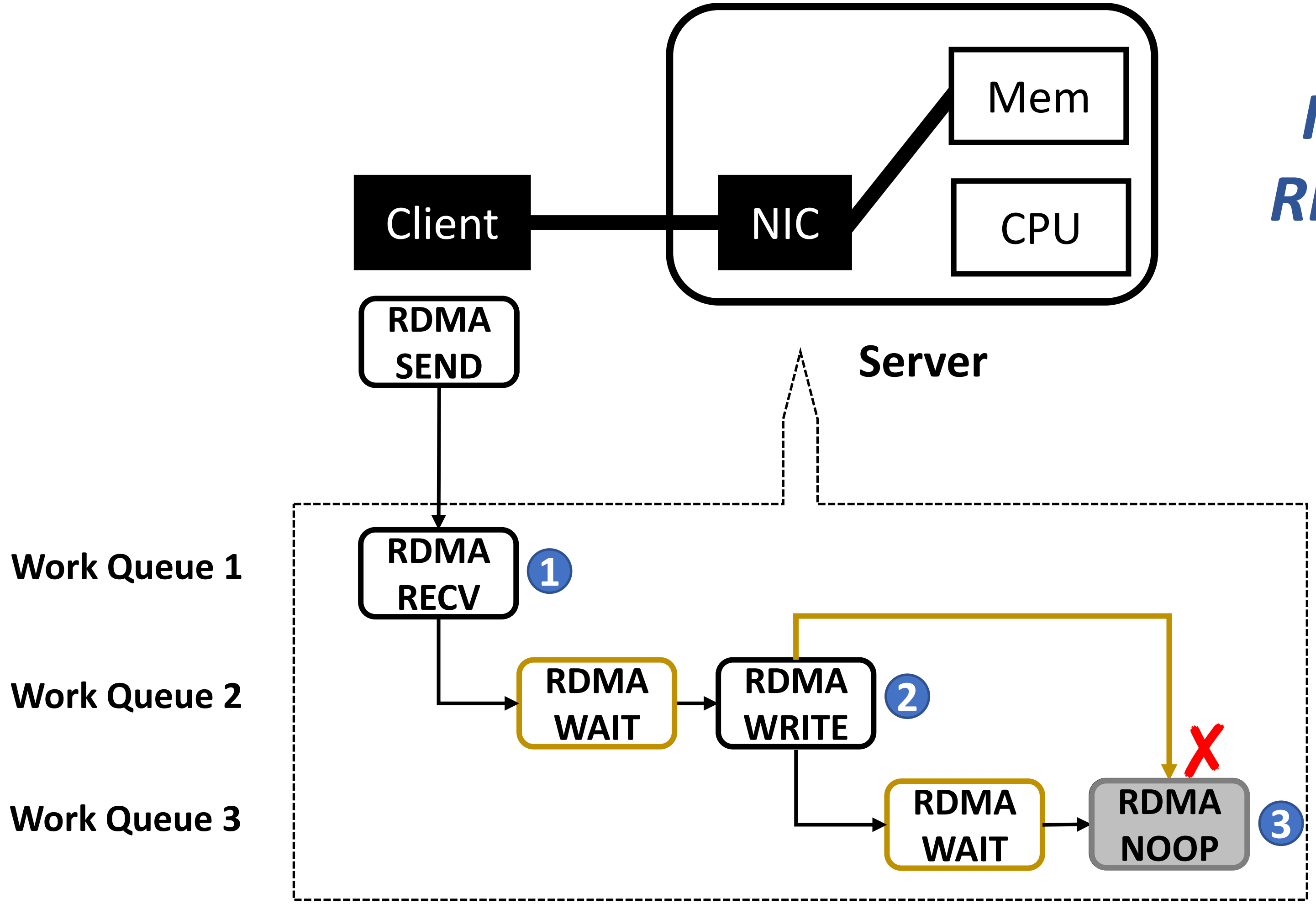


Conditional Branching – is it possible?



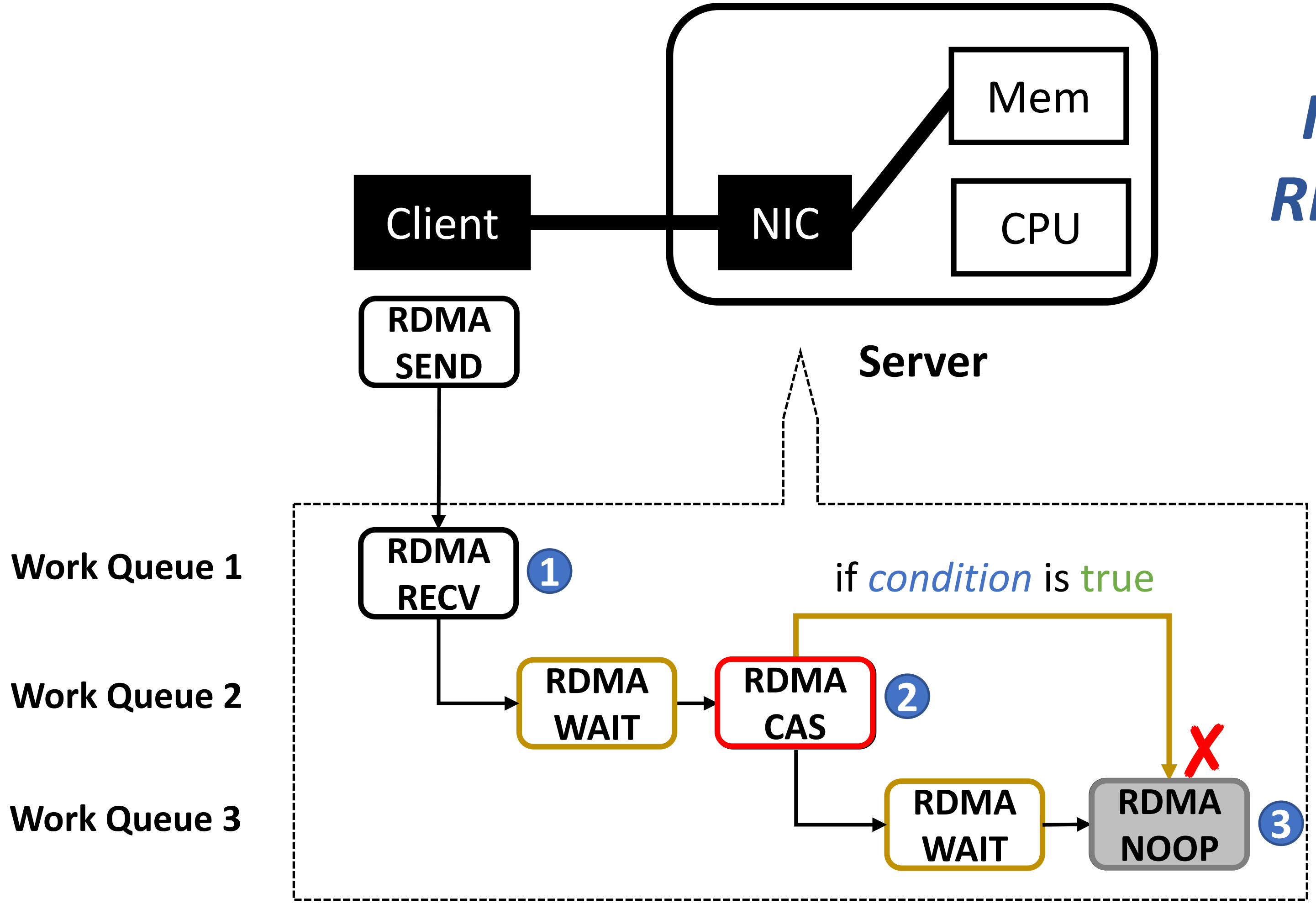
Insight #2: Use-self-modifying RDMA code to control execution

Conditional Branching – is it possible?



Insight #2: Use-self-modifying RDMA code to control execution

Conditional Branching – is it possible?



Insight #2: Use-self-modifying RDMA code to control execution

RDMA Compare-and-Swap (CAS)

to check conditions

- Typically used for simple transactions
- Supported by commodity RDMA NICs

Branching with Self-Modifying Code

Simple Example

Input x, y
If (x == y)
    return **foo**;
else
    return **bar**;

RDMA code (server-side):

Work Queues (WQs)

Branching with Self-Modifying Code

Simple Example

Input x, y
If ($x == y$)
    return **foo**;
else
    return **bar**;

RDMA code (server-side):

	opcode	*id*	
R1	RECV		
R2	CAS	old: NOOP	new: WRITE
R3	NOOP	data: **foo**	
R4	WRITE	data: **bar**	

Work Queues (WQs)

Branching with Self-Modifying Code

Simple Example

$\boxed{\text{Assume } x == y \text{ is true}}$

Input x, y
If ($x == y$)
        return **foo**;
else
        return **bar**;

RDMA code (server-side):

	opcode	*id*	
R1	RECV		
R2	CAS	old: NOOP	new: WRITE
R3	NOOP	data: **foo**	
R4	WRITE	data: **bar**	

Work Queues (WQs)

Branching with Self-Modifying Code

Simple Example

Assume $x == y$ is true

Input x, y
If ($x == y$)
    return **foo**;
else
    return **bar**;

RDMA code (server-side):

opcode id

R1 RECV

R2 CAS old: NOOP new: WRITE

R3 NOOP data: **foo**

R4 WRITE data: **bar**

Work Queues (WQs)

Client

SEND data: x, y

Branching with Self-Modifying Code

Simple Example

Assume $x == y$ is true

Input x, y
If ($x == y$)
    return **foo**;
else
    return **bar**;

RDMA code (server-side):

opcode *id*

R1	RECV	
R2	CAS	old: NOOP $\boxed{y}$ new: WRITE
R3	NOOP $\boxed{x}$	data: **foo**
R4	WRITE	data: **bar**

Work Queues (WQs)

Client

SEND data: x, y

Branching with Self-Modifying Code

Simple Example

Assume $x == y$ is true

Input x, y
If ($x == y$)
        return **foo**;
else
        return **bar**;

RDMA code (server-side):

opcode *id*

R1 RECV

R2 CAS old: NOOP y new: WRITE

R3 NOOP x data: **foo**

R4 WRITE data: **bar**

Work Queues (WQs)

Client

SEND data: x, y

Branching with Self-Modifying Code

Simple Example

Assume $x == y$ is true

Input x, y
If ($x == y$)
  return **foo**;
else
  return **bar**;

RDMA code (server-side):

opcode *id*

R1	RECV		
R2	CAS	old: NOOP y	new: WRITE
R3	WRITE	data: **foo**	
R4	WRITE	data: **bar**	

Work Queues (WQs)

Client

SEND data: x, y

Branching with Self-Modifying Code

Simple Example

Assume $x == y$ is true

Input x, y
If ($x == y$)
    return **foo**;
else
    return **bar**;

RDMA code (server-side):

opcode *id*

R1	RECV	
R2	CAS	old: NOOP $\boxed{y}$ new: WRITE
R3	$\boxed{\text{WRITE}}$	data: **foo**
R4	WRITE	$\boxed{\text{data: } \textbf{foo}}$

Work Queues (WQs)

Client

SEND data: x, y

Branching with Self-Modifying Code

Simple Example

Assume $x == y$ is true

Input x, y

If ($x == y$)

return **foo**;

else

return **bar**;

RDMA code (server-side):

opcode *id*

R1 RECV

R2 CAS old: NOOP $\boxed{y}$ new: WRITE

R3 WRITE data: **foo**

R4 WRITE data: **foo** → **foo**

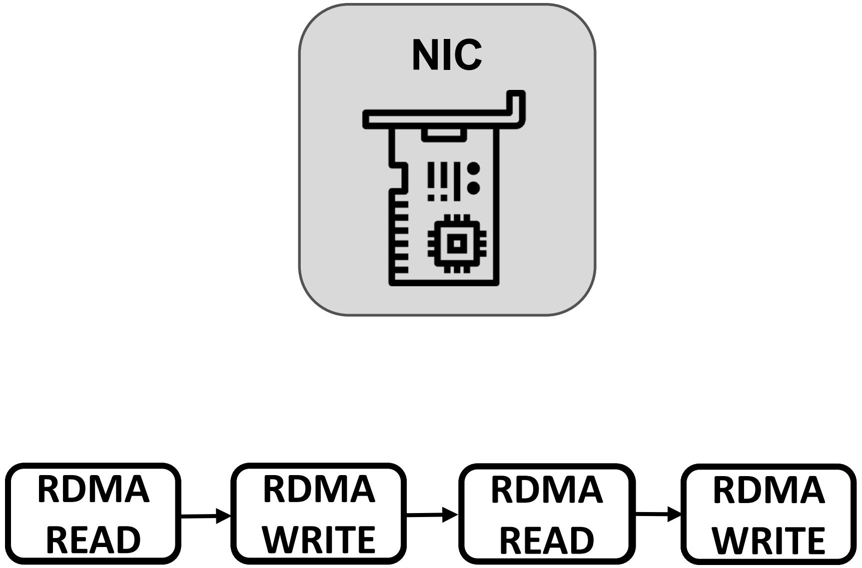
Client

SEND data: x, y

Work Queues (WQs)

What about loops?

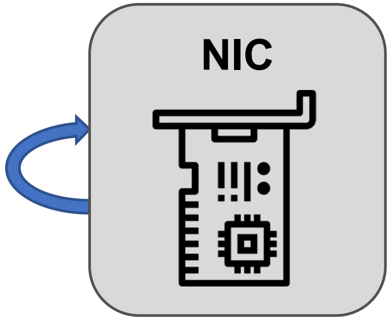
- **Observation:** RDMA operations are not deleted after execution

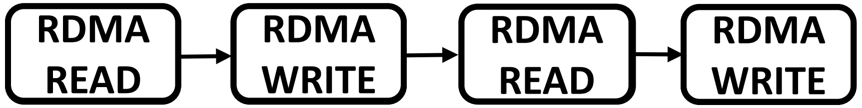


What about loops?

- **Observation:** RDMA operations are not deleted after execution

***Insight #3*: Recycle previously posted RDMA operations**

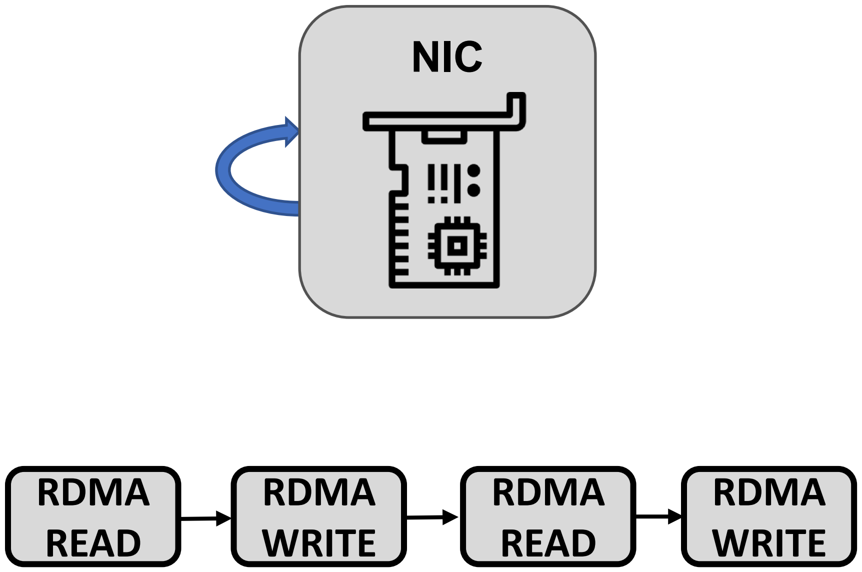




What about loops?

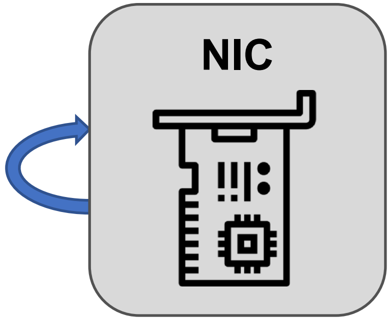
- **Observation:** RDMA operations are not deleted after execution

***Insight #3*: Recycle previously posted RDMA operations**



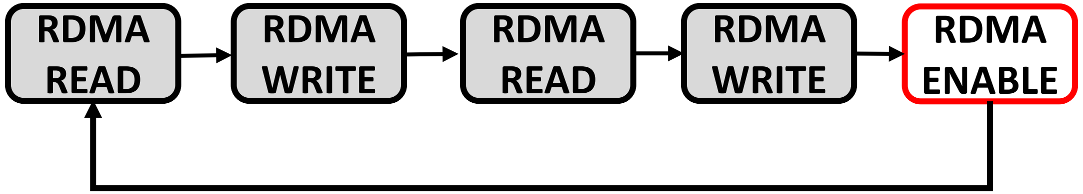
What about loops?

- **Observation:** RDMA operations are not deleted after execution



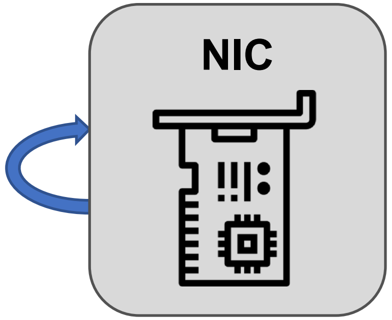
***Insight #3*: Recycle previously posted RDMA operations**

Use **RDMA ENABLE** at the end to re-trigger chain



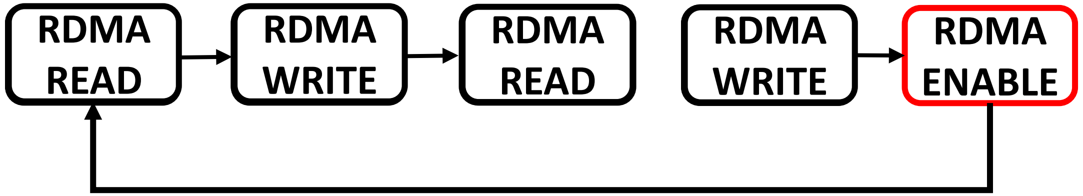
What about loops?

- **Observation:** RDMA operations are not deleted after execution

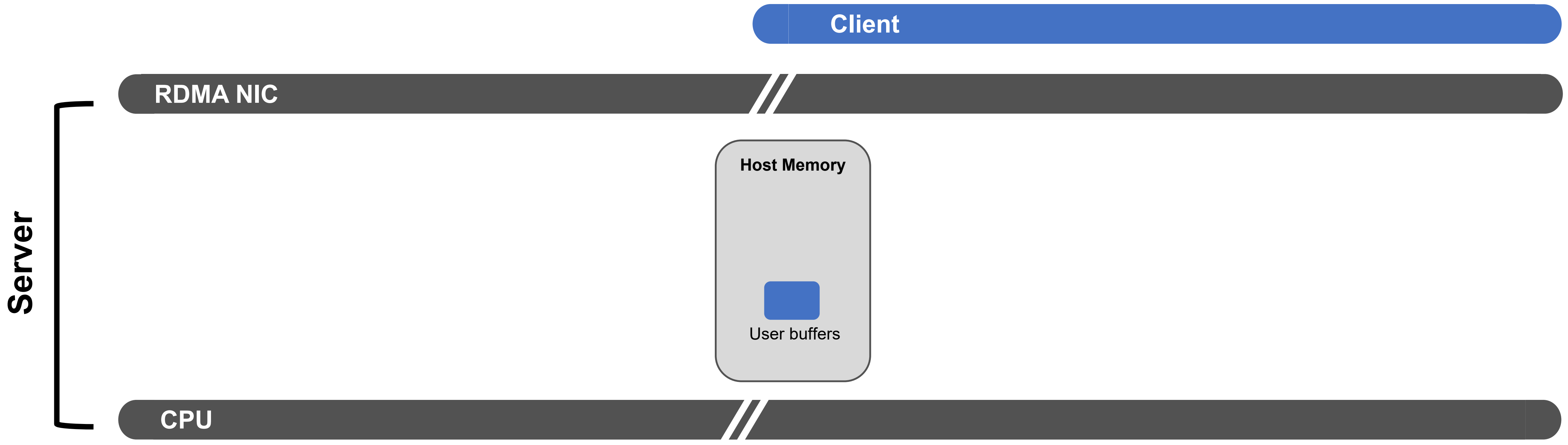


***Insight #3*: Recycle previously posted RDMA operations**

Use **RDMA ENABLE** at the end to re-trigger chain

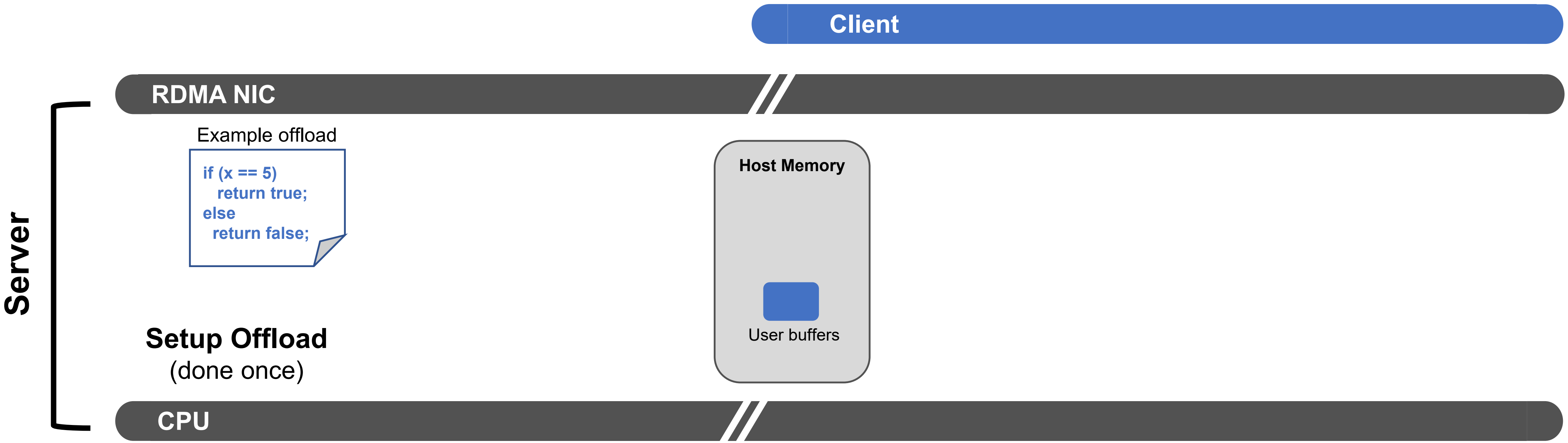


RedN Framework - Overview



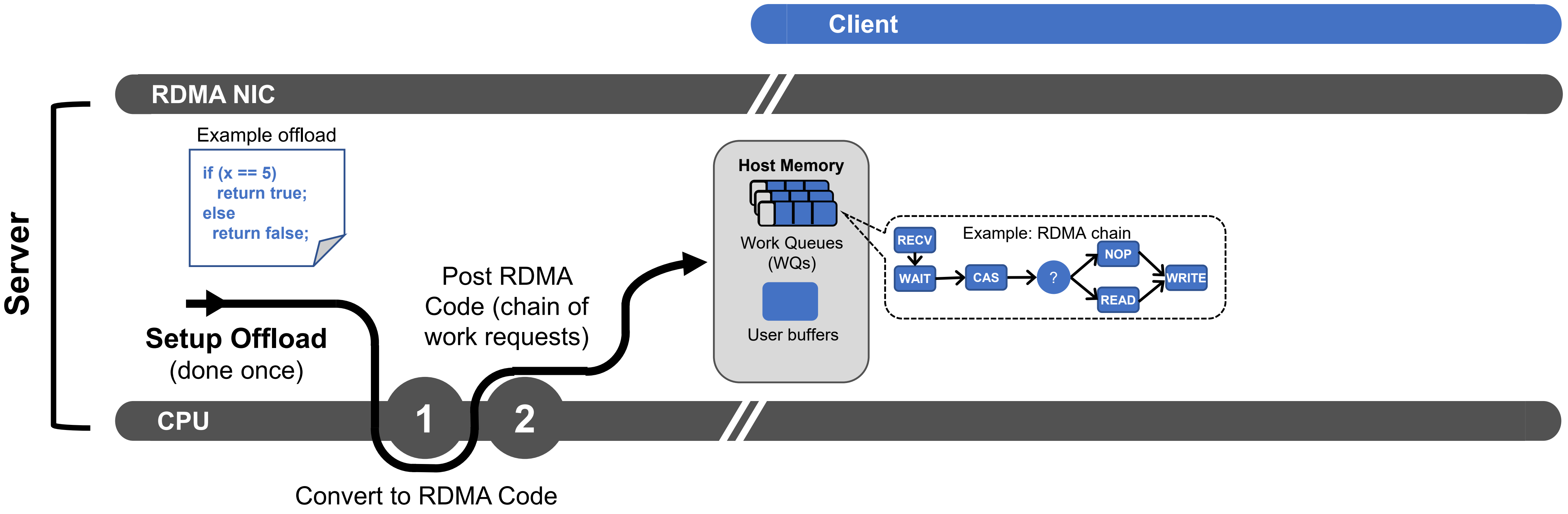
Convert to RDMA Code

RedN Framework - Overview

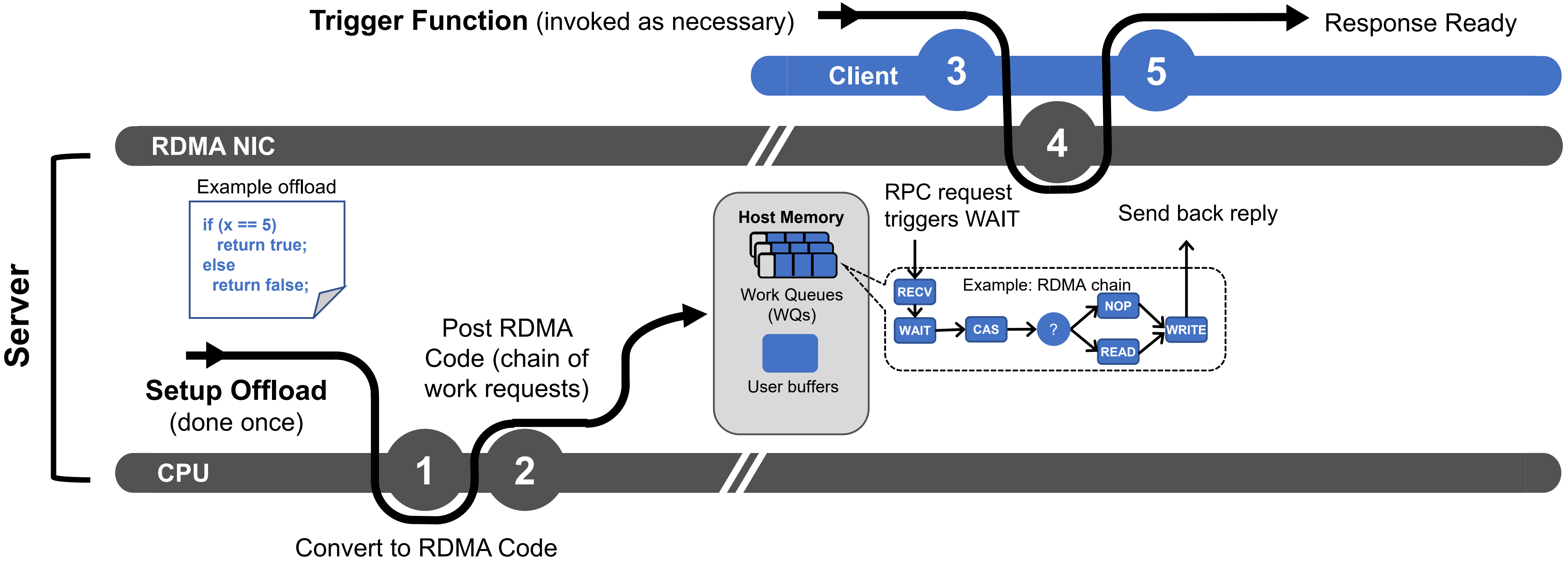


Convert to RDMA Code

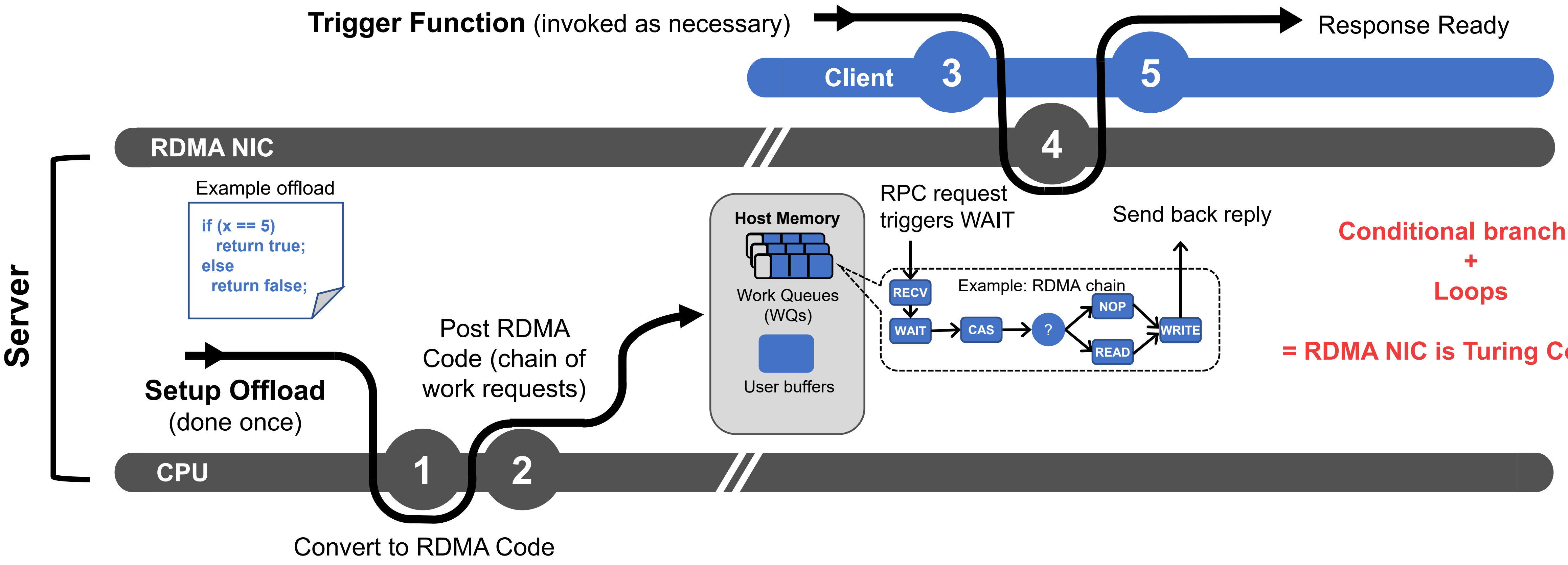
RedN Framework - Overview



RedN Framework - Overview



RedN Framework - Overview



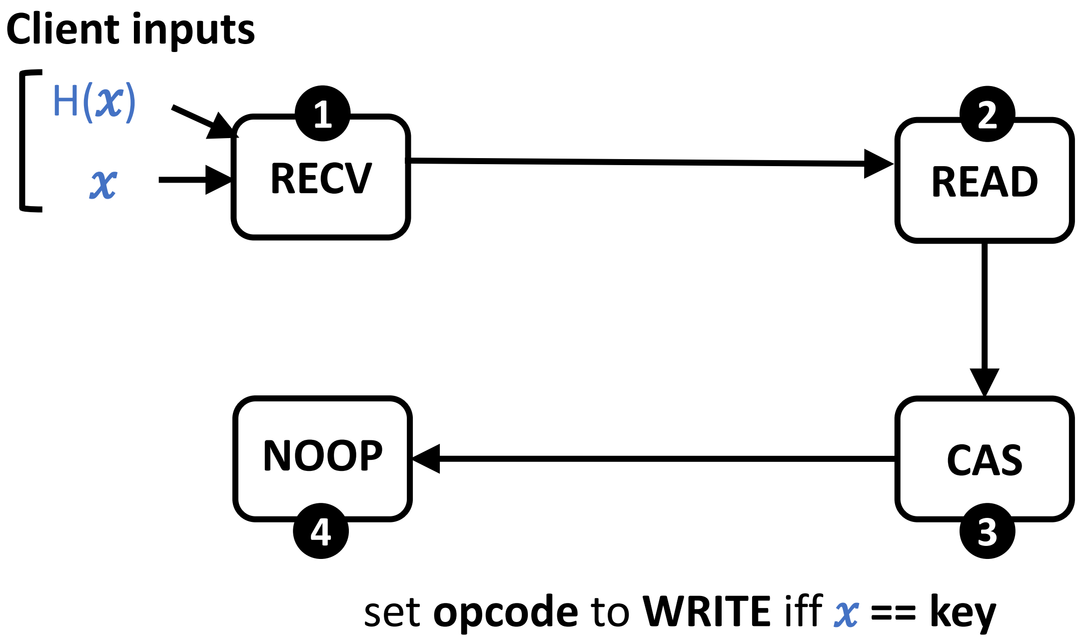
Conditional branching
+
Loops

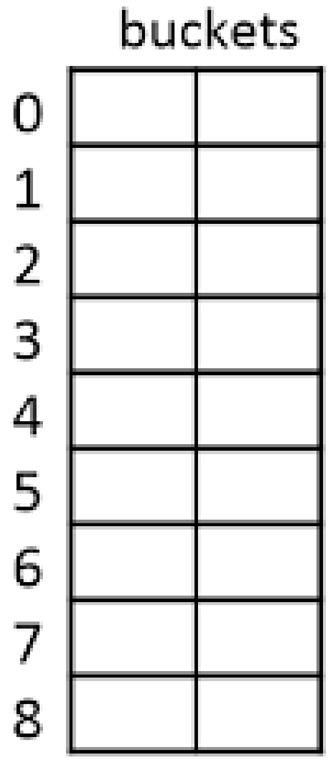
= RDMA NIC is Turing Complete

Evaluation

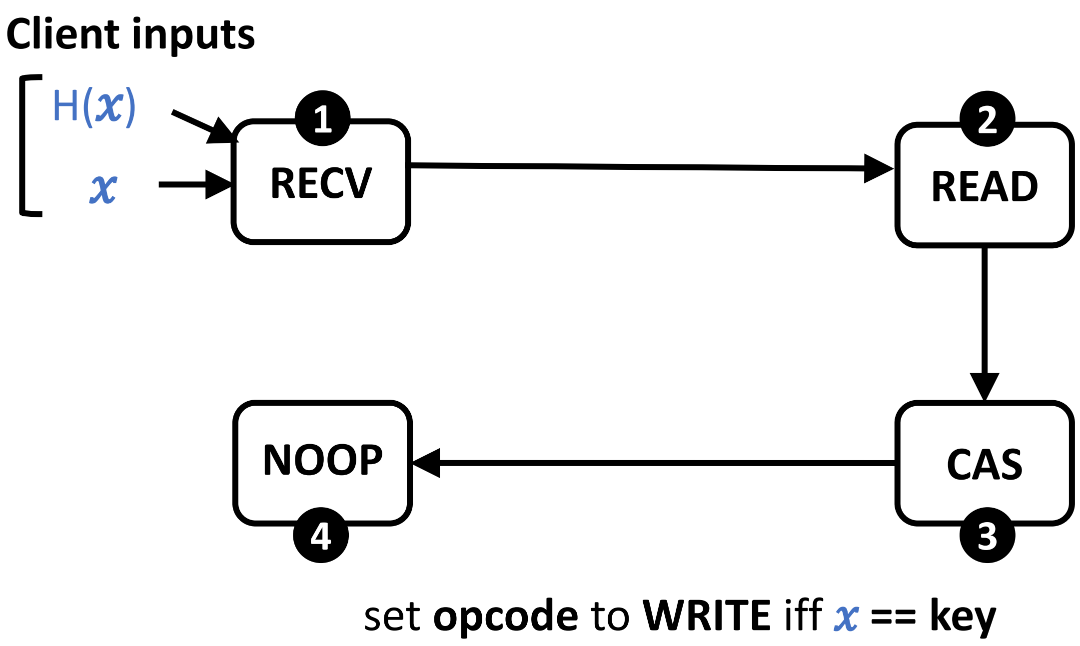
- Our experimental testbed consists of 3× dual-socket Haswell servers:
  - 3.2 GHz, with a total of 16 cores
  - 128 GB of DRAM
  - 100 Gbps dual-port Nvidia **ConnectX-5 Infiniband RNICs**.
  - Nodes are connected via back-to-back Infiniband links
- We evaluate RedN using microbenchmarks and real applications (e.g. Memcached)

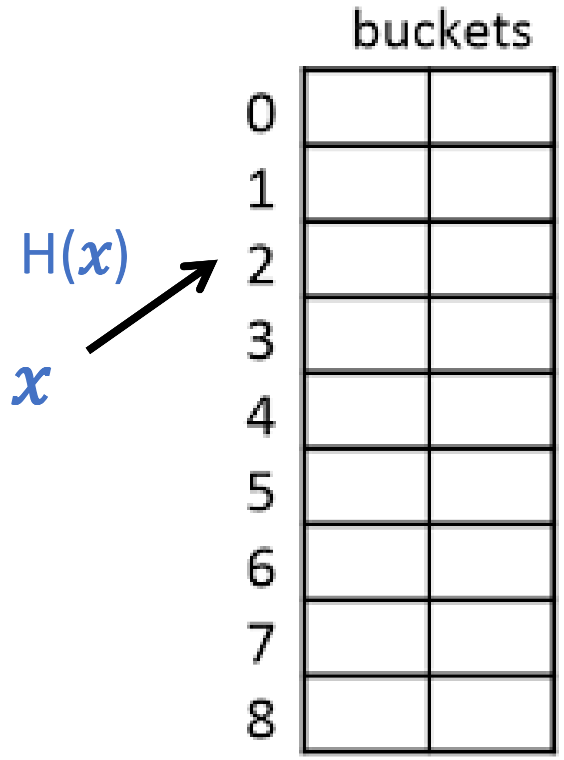
Use case: Memcached Lookups



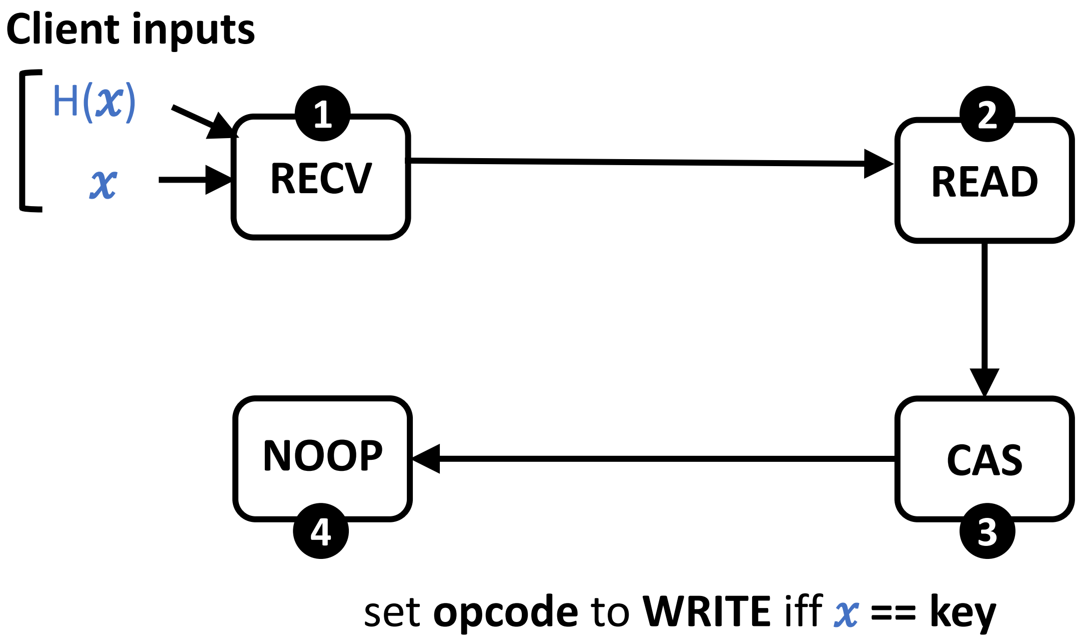


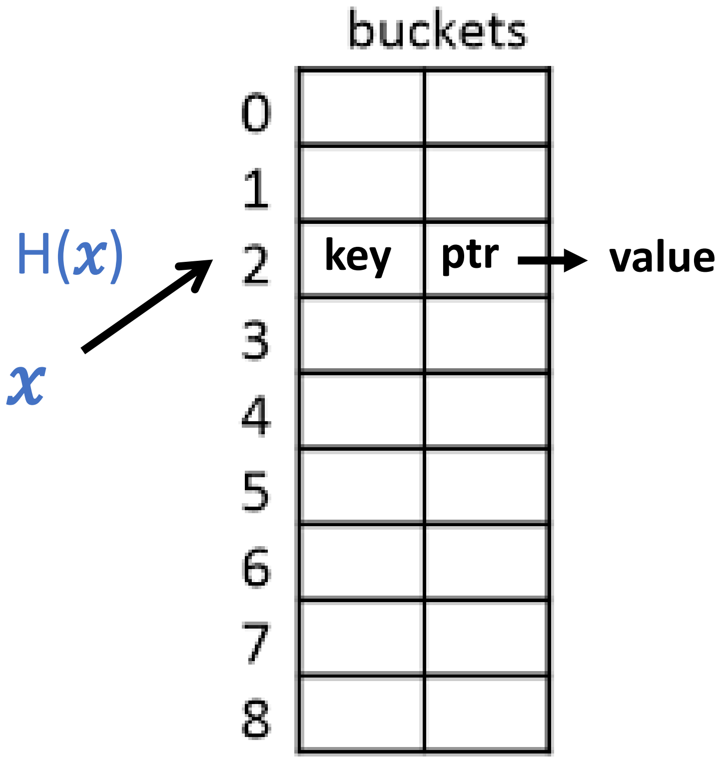
Use case: Memcached Lookups



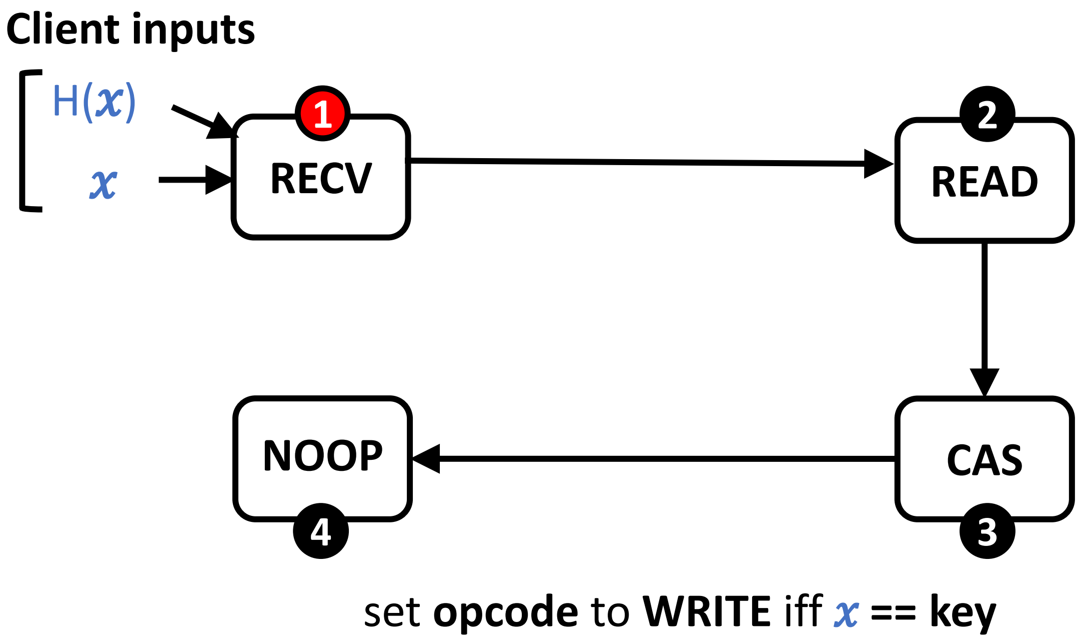


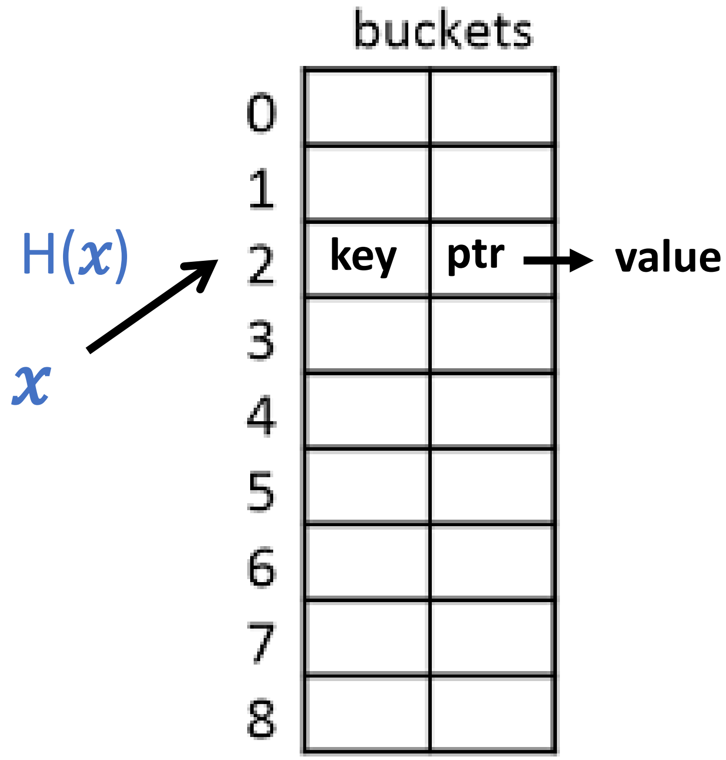
Use case: Memcached Lookups



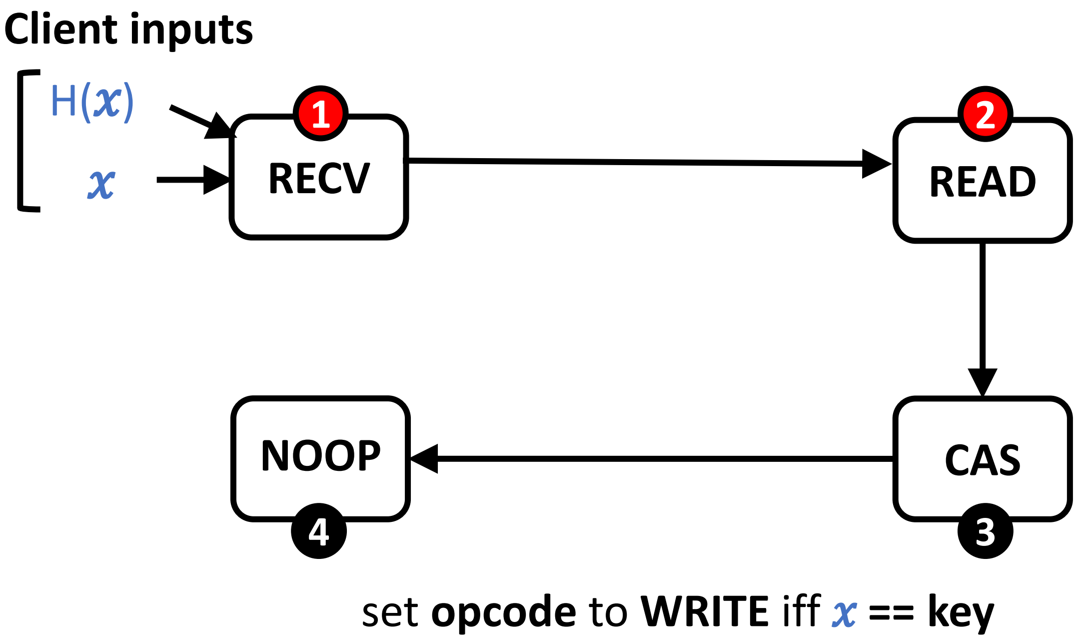


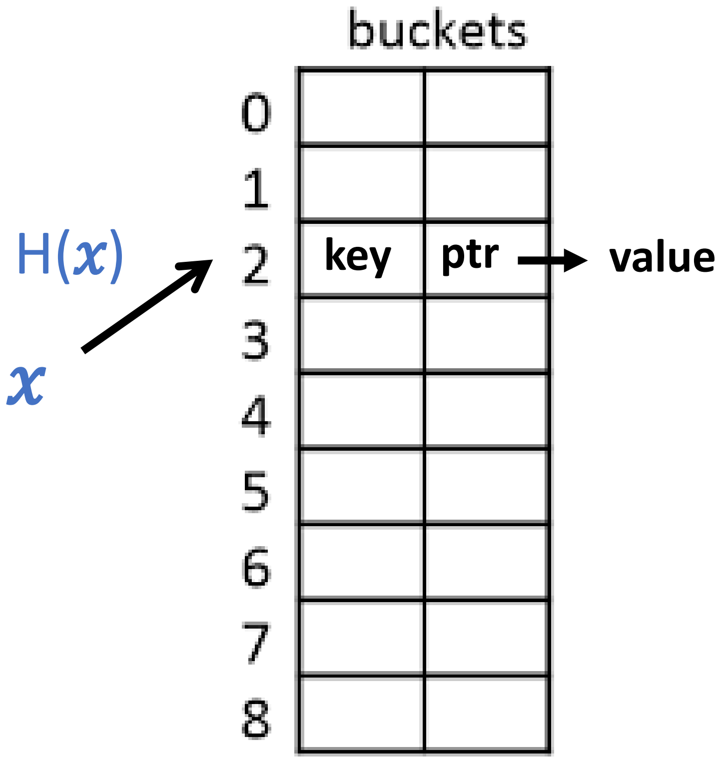
Use case: Memcached Lookups



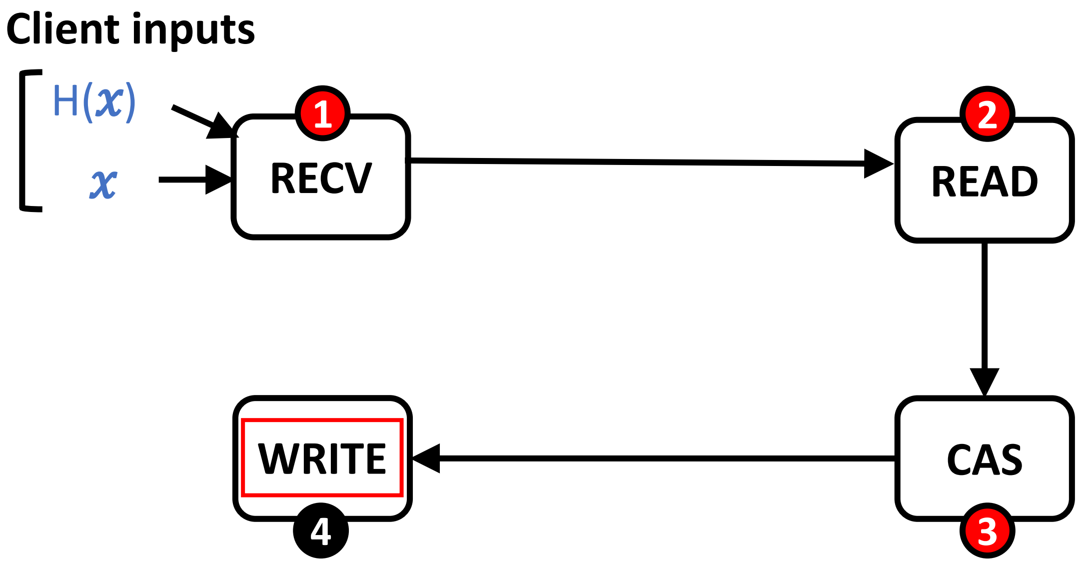


Use case: Memcached Lookups

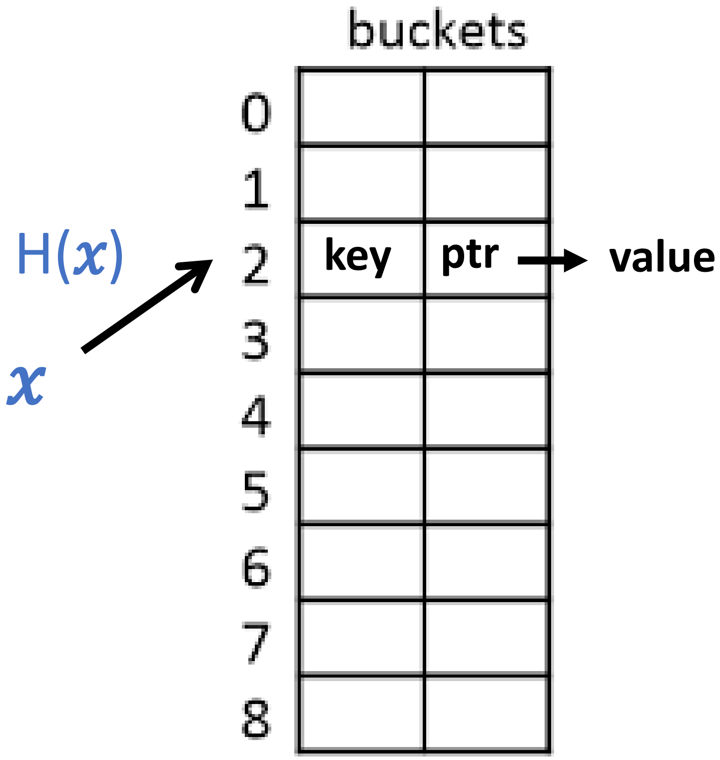




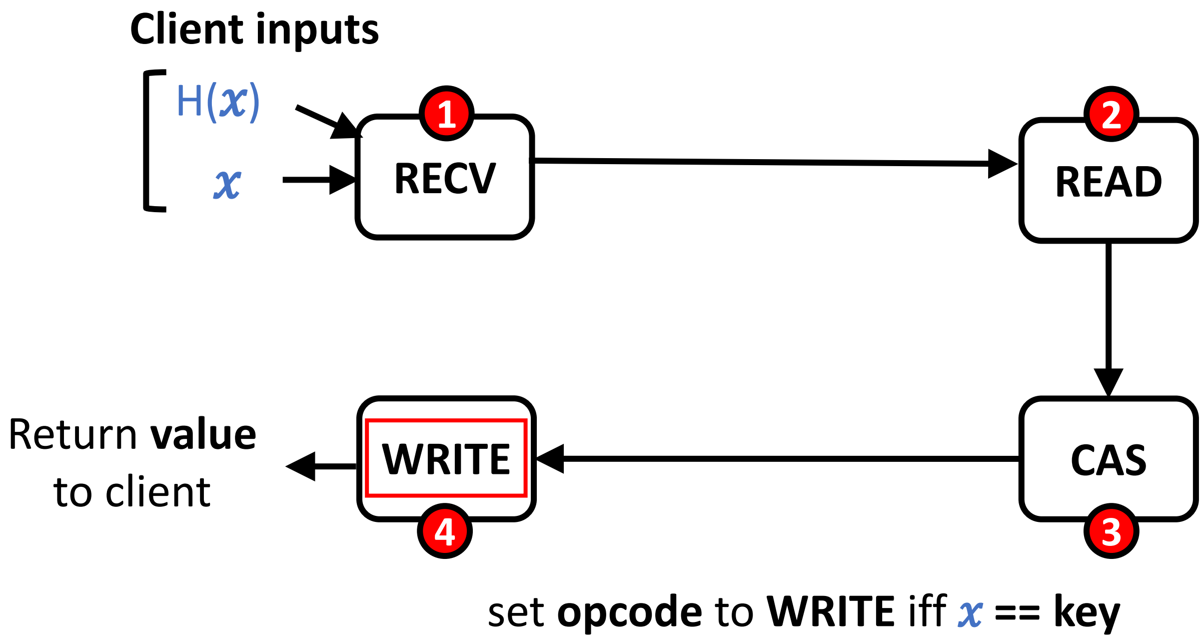
Use case: Memcached Lookups

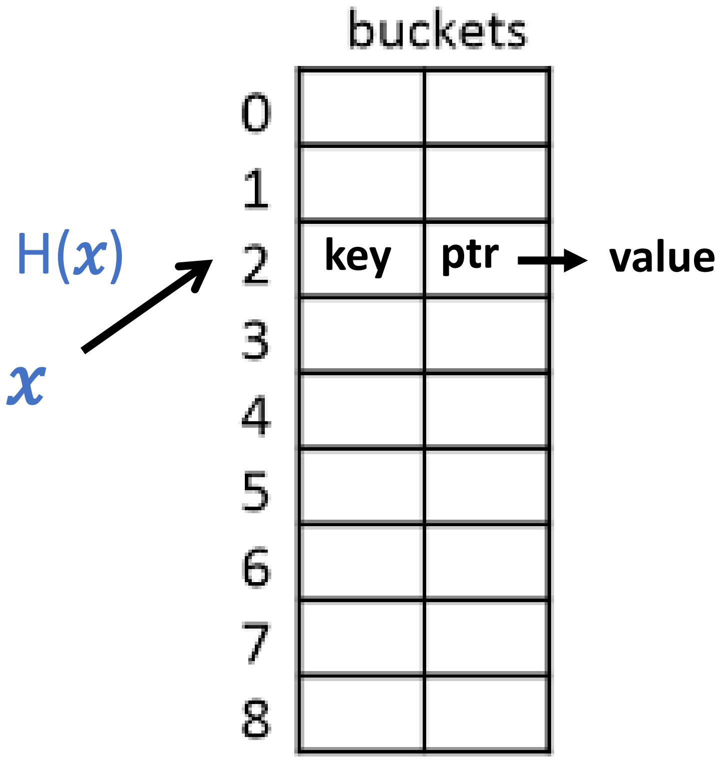


set **opcode** to **WRITE** iff x **== key**

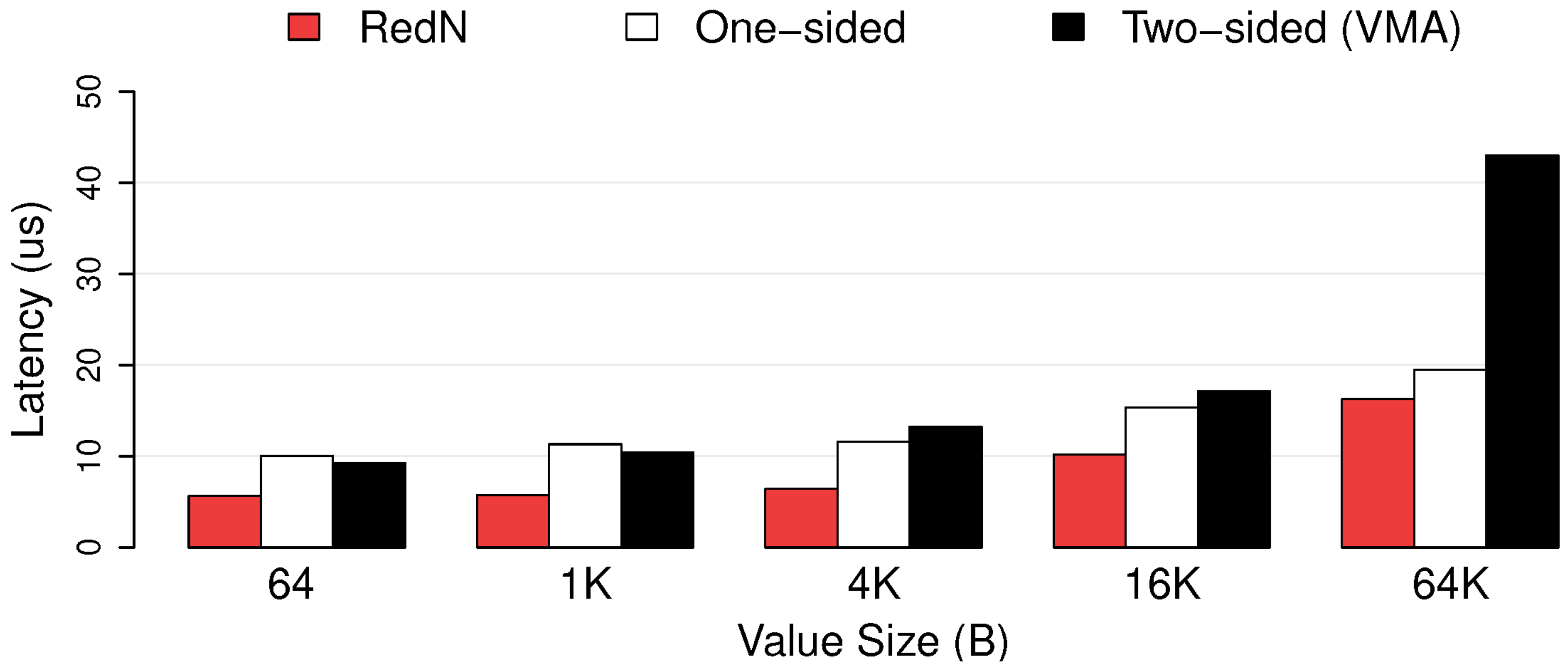


Use case: Memcached Lookups

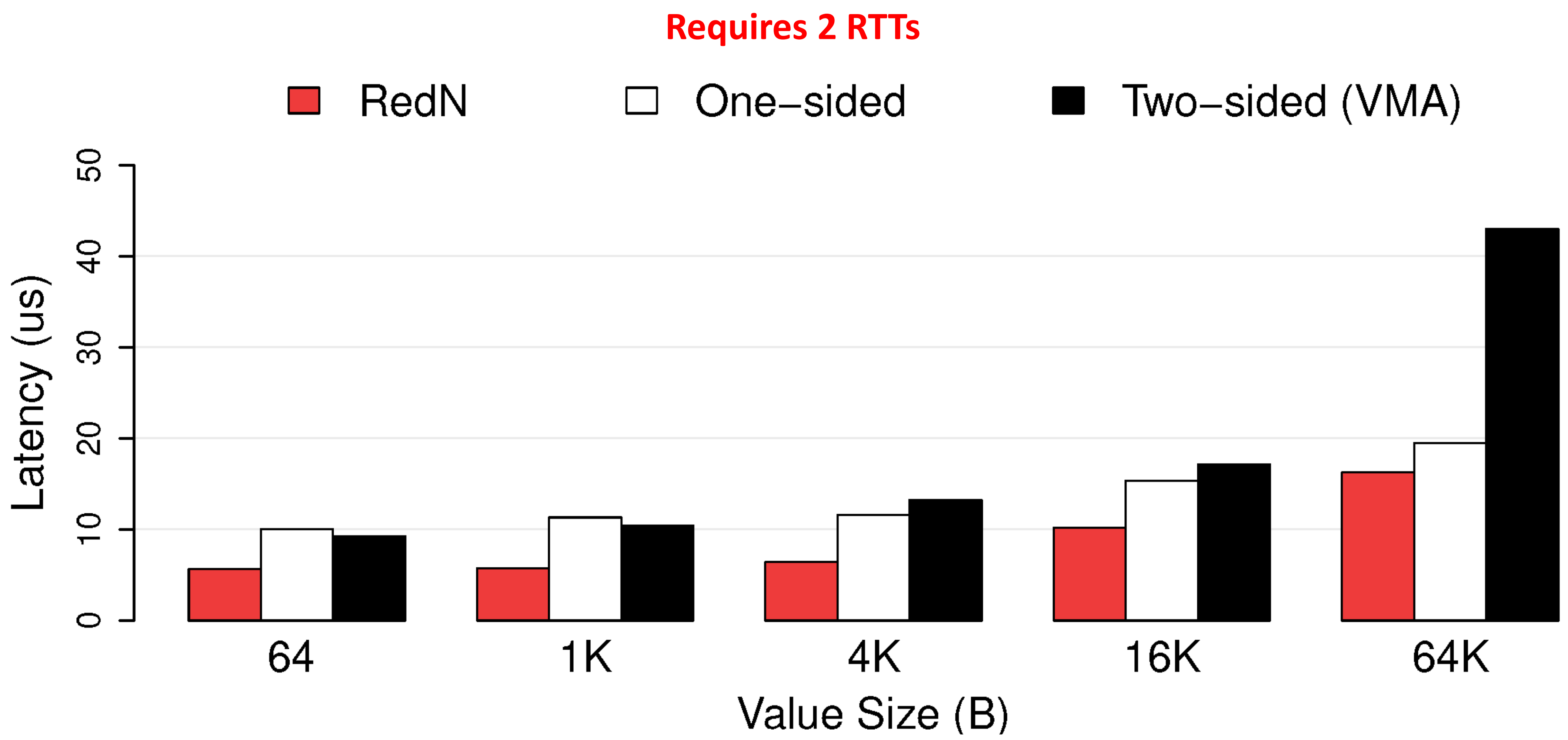




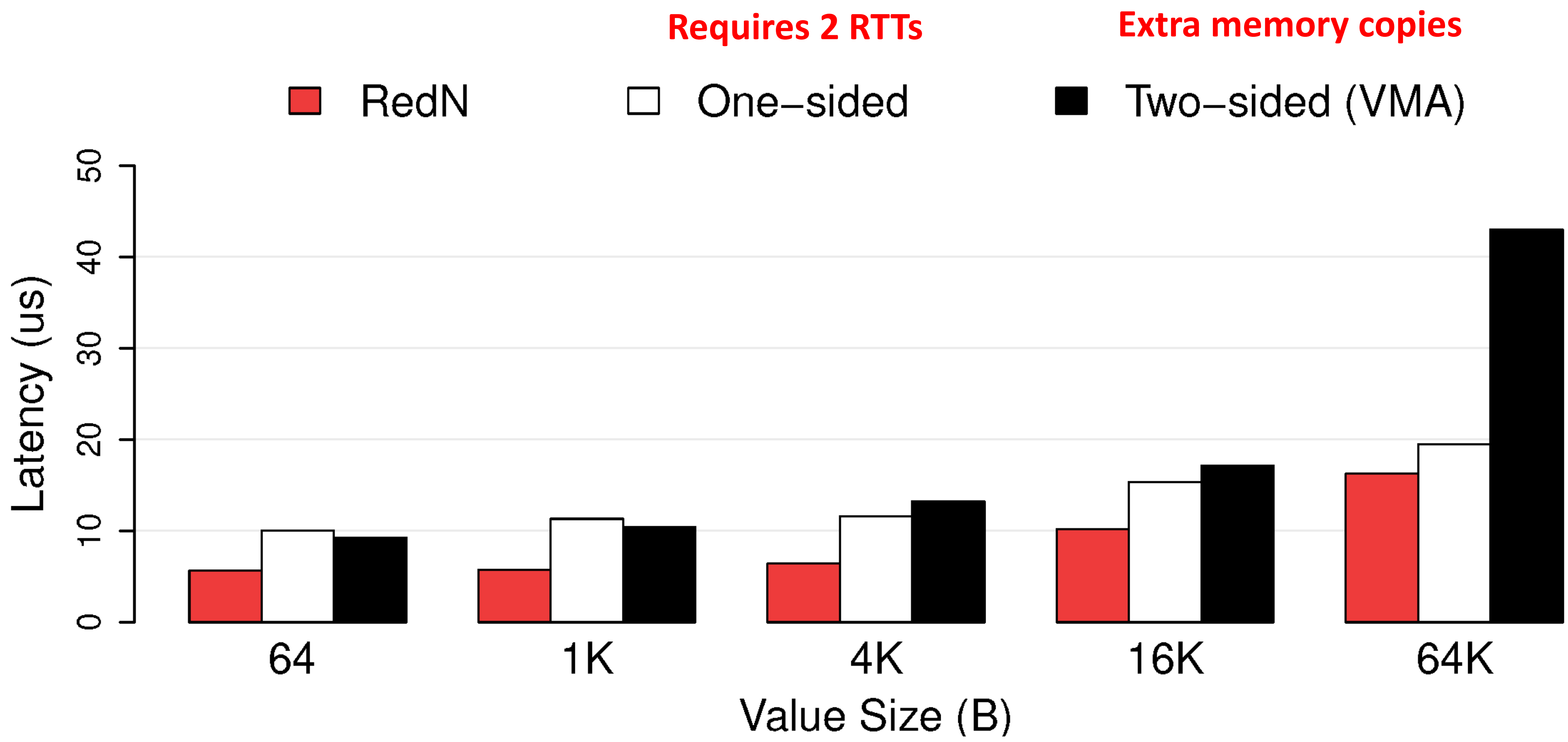
Results: Memcached *get* latency



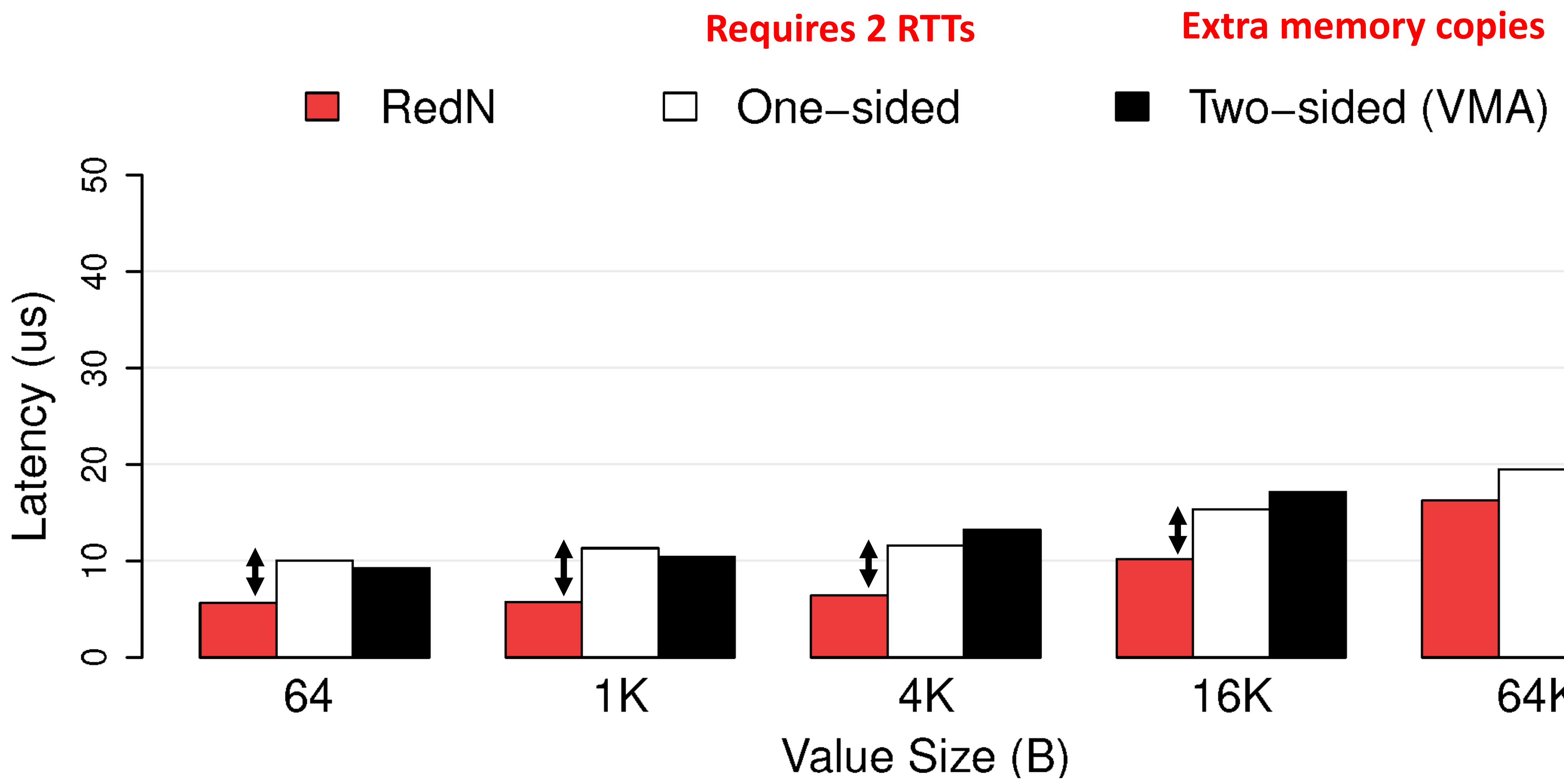
Results: Memcached *get* latency



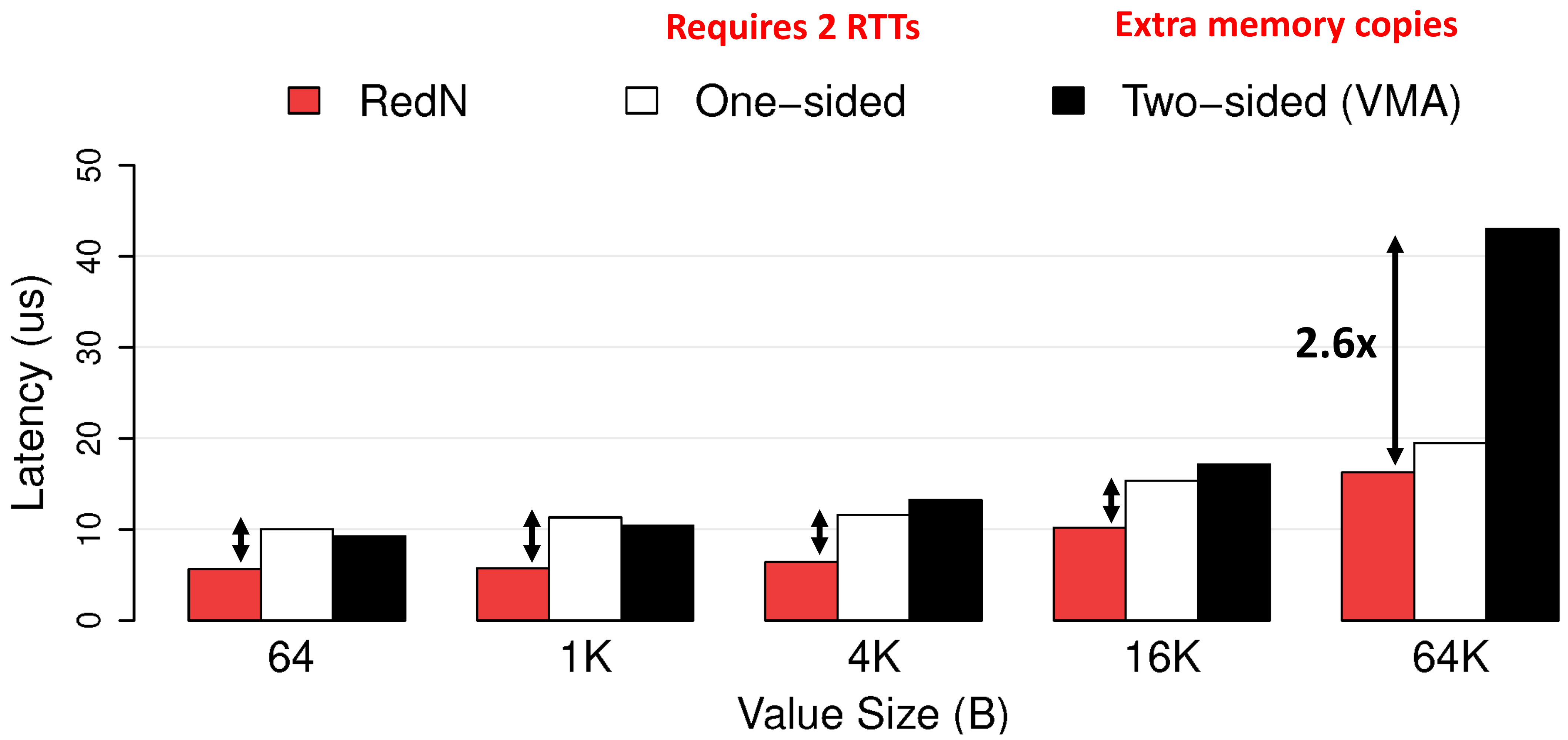
Results: Memcached *get* latency



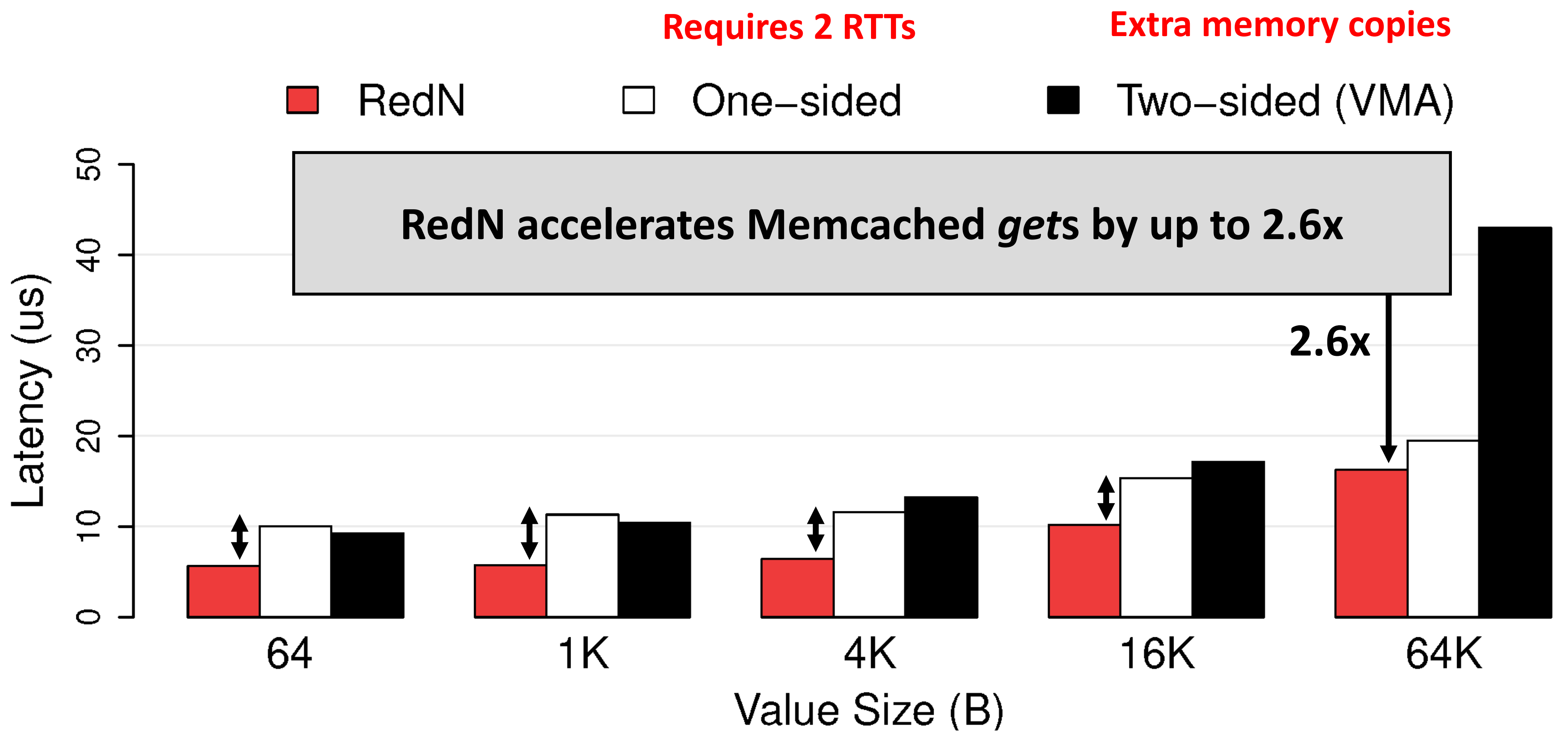
Results: Memcached *get* latency



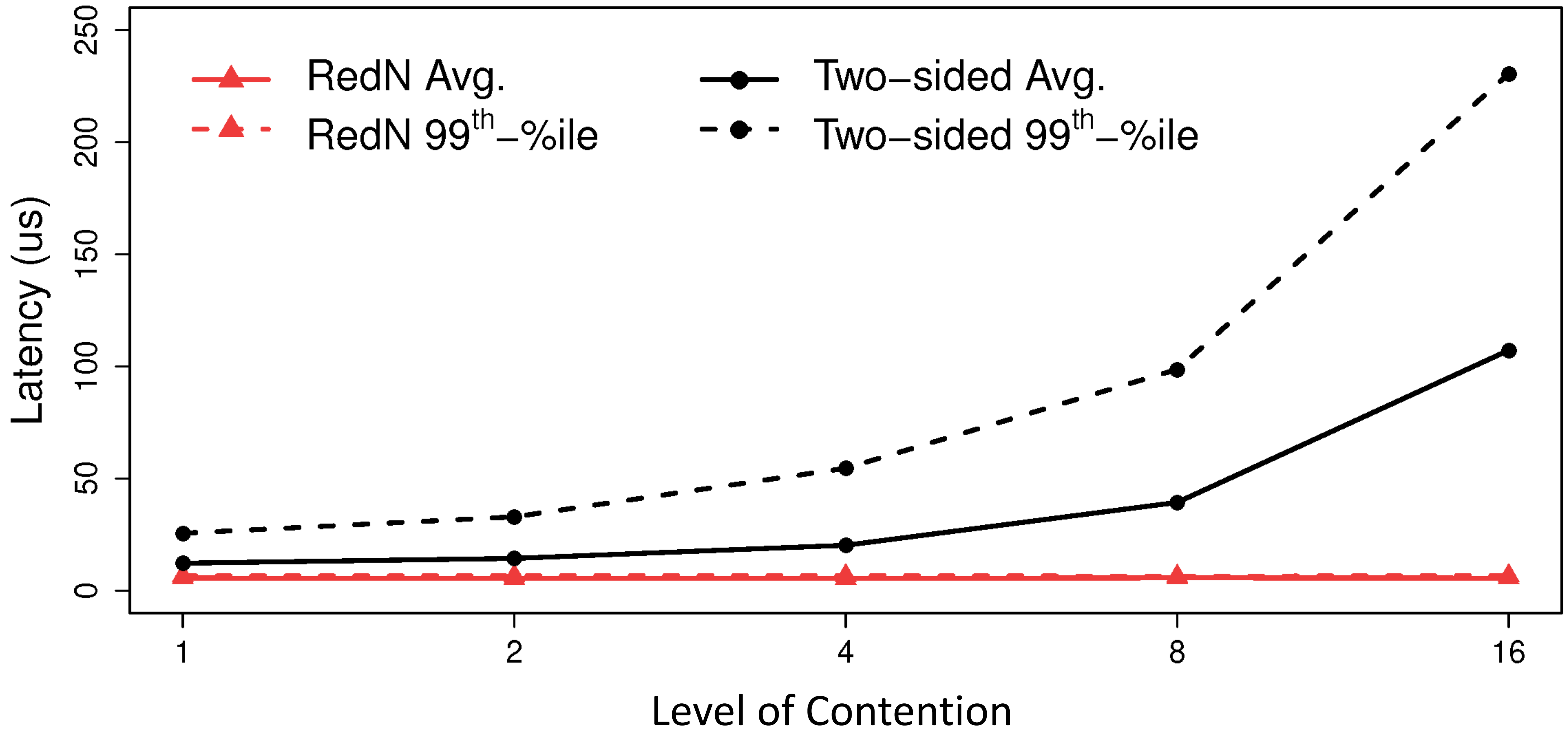
Results: Memcached *get* latency



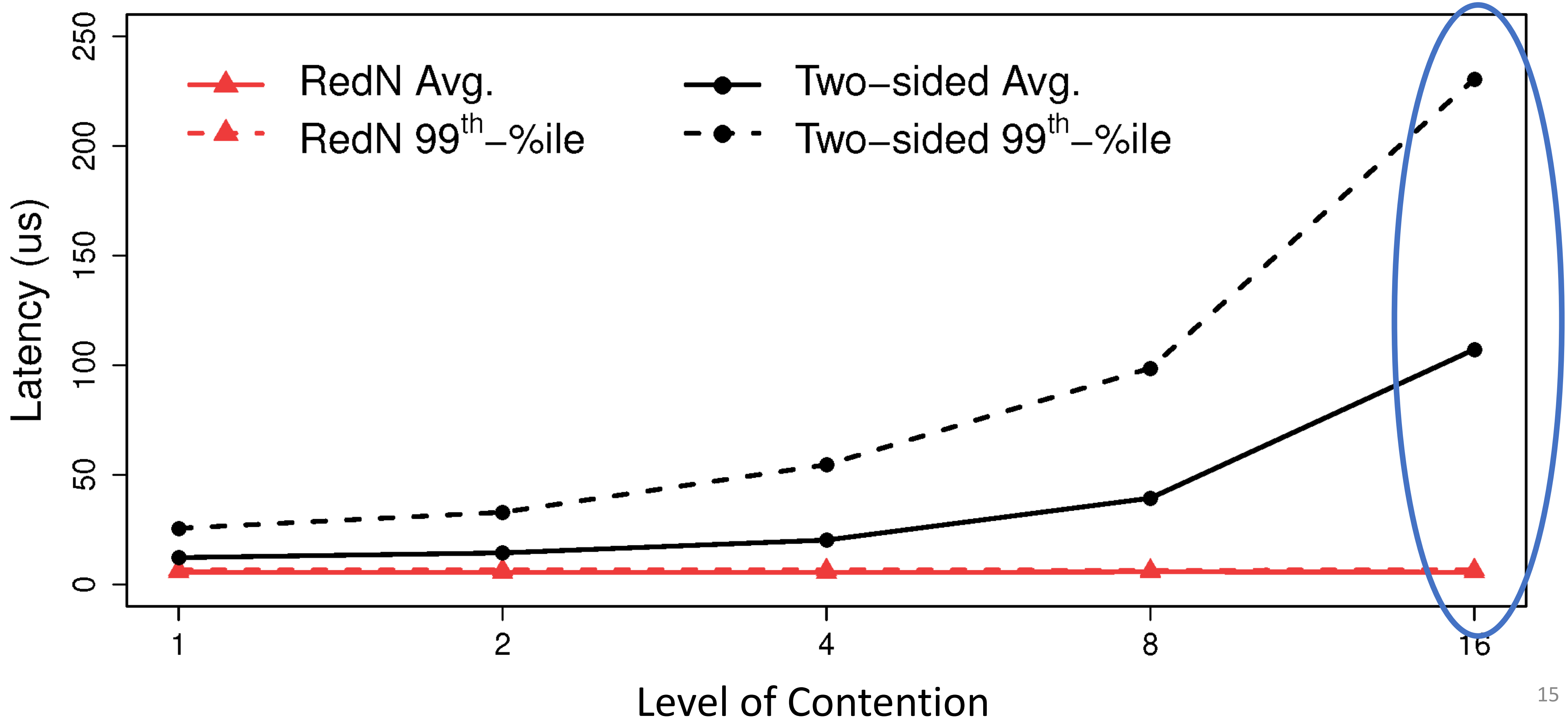
Results: Memcached *get* latency



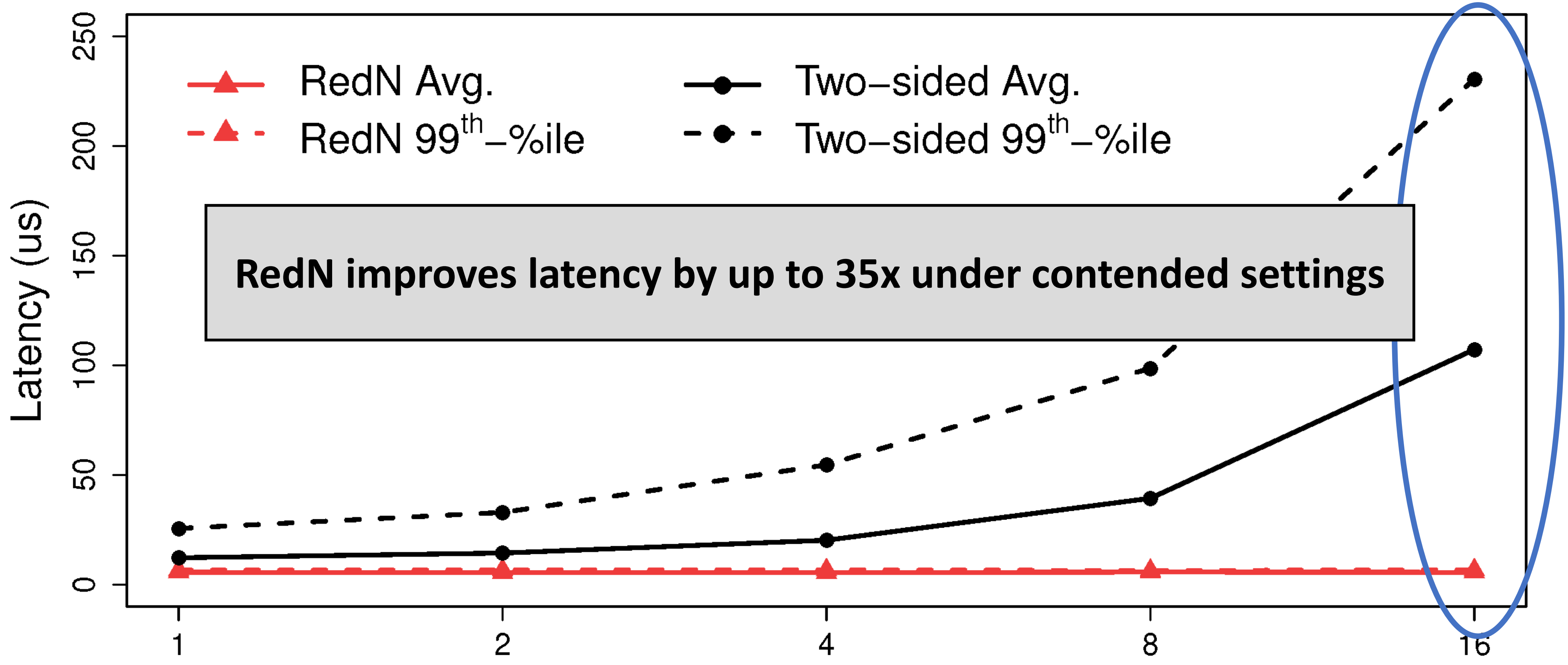
Results: Memcached *get* (contention)



Results: Memcached *get* (contention)



Results: Memcached *get* (contention)

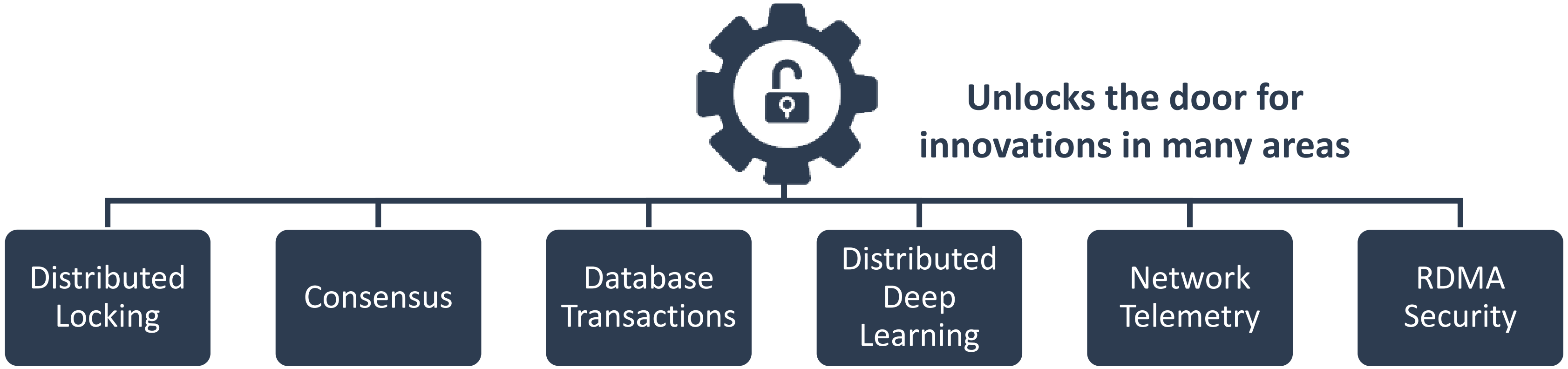


RedN improves latency by up to 35x under contended settings

Level of Contention

Conclusion

- **RedN** shows that RDMA is Turing complete



- Source code: redn.io