PMNet
In-Network Data Persistence

Korakit Seemakhupt, Sihang Liu, Yasas Senevirathne
Muhammad Shahbaz, Samira Khan

Artifact available at pmnet.persistentmemory.org

Summary

Motivation

- Datacenter applications usually store data in separate servers and manage through network
- Long latency of update requests slows down clients
- In-network compute reduces read requests' latency but not update requests

Key Insight

- Maintain persistent states in network devices by adding persistent memory
- Persist update requests in network to move server's latency off the critical path

PMNet

- Logs requests in network device's persistent memory
- Recovers server using logged requests in case of a failure
- Integrates in-network data persistence with data replication and caching

Evaluation

- End-to-end FPGA implementation of PMNet-enabled NIC and switch
- Improves throughput by 4.27x and tail latency by 3.23x over client-server baseline

Outline

Background and Motivation

In-network Data Persistence

PMNet Design

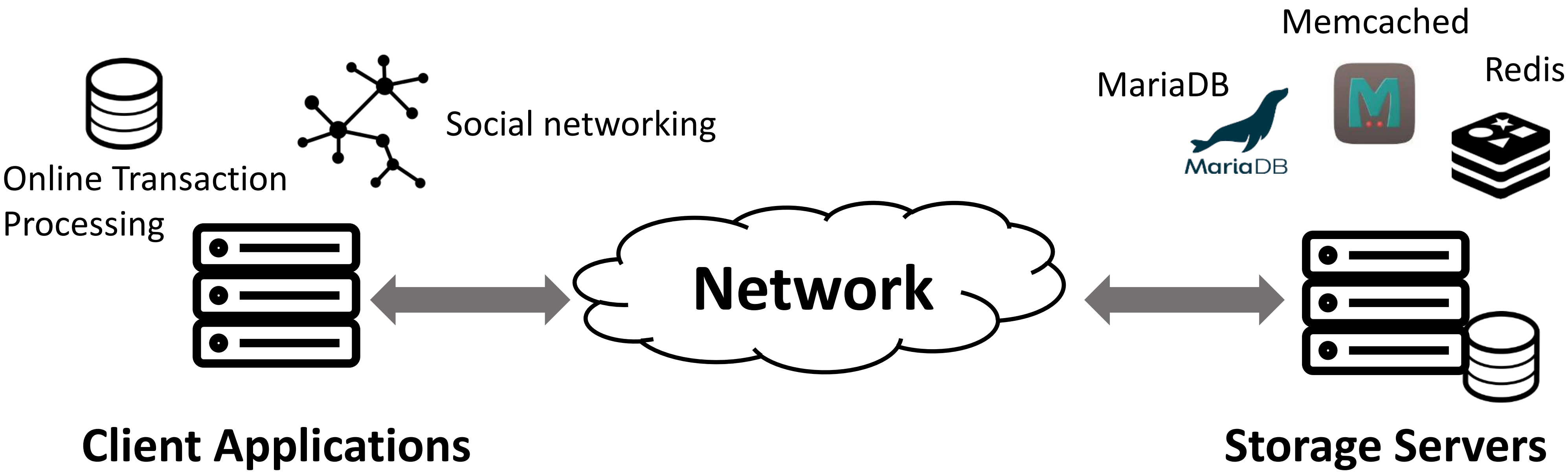
Caching and Replication

Evaluation

Conclusion

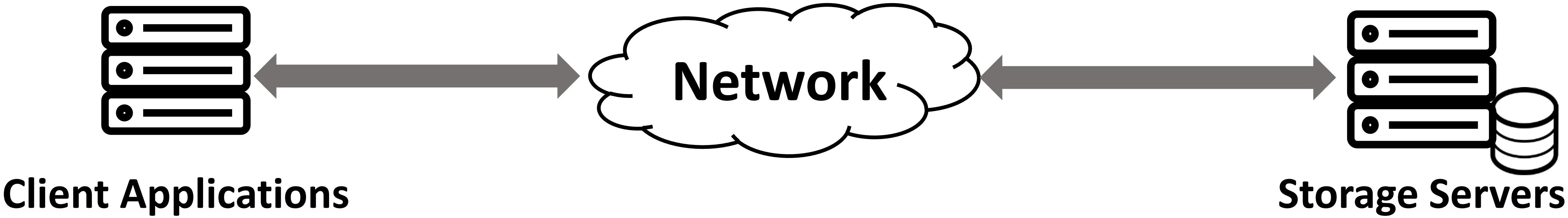
Storage Applications in Datacenter

Common datacenter applications store data in separate storage servers



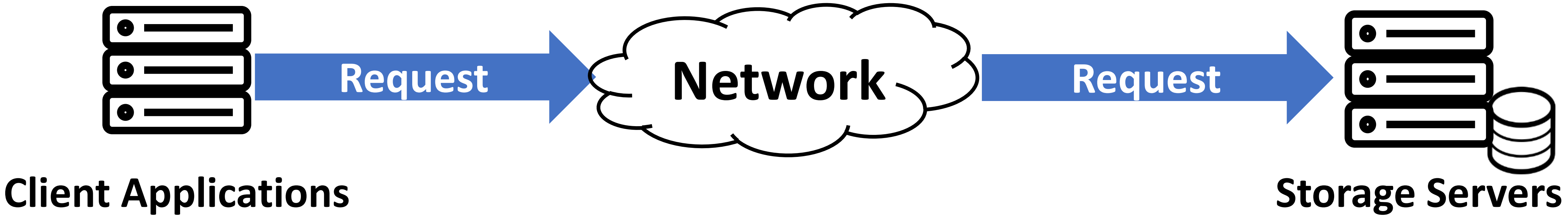
Latency of accessing data on the storage server is critical to the performance of these applications

Storage Applications in Datacenter



🏛 **A** transfers $10 to **B**

Storage Applications in Datacenter

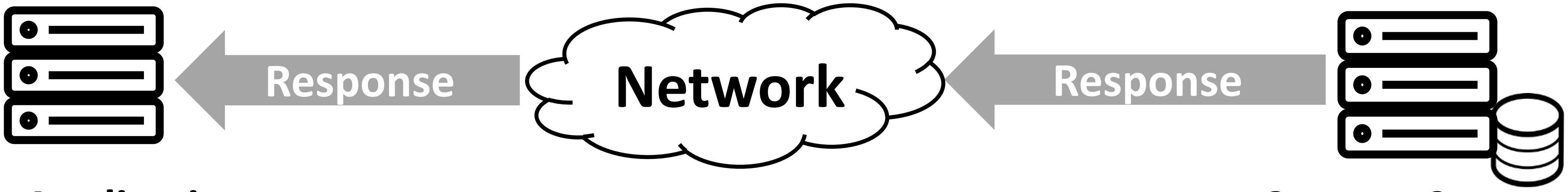


A transfers $10 to **B**

A’s balance - 10

B’s balance + 10

Storage Applications in Datacenter



Client Applications

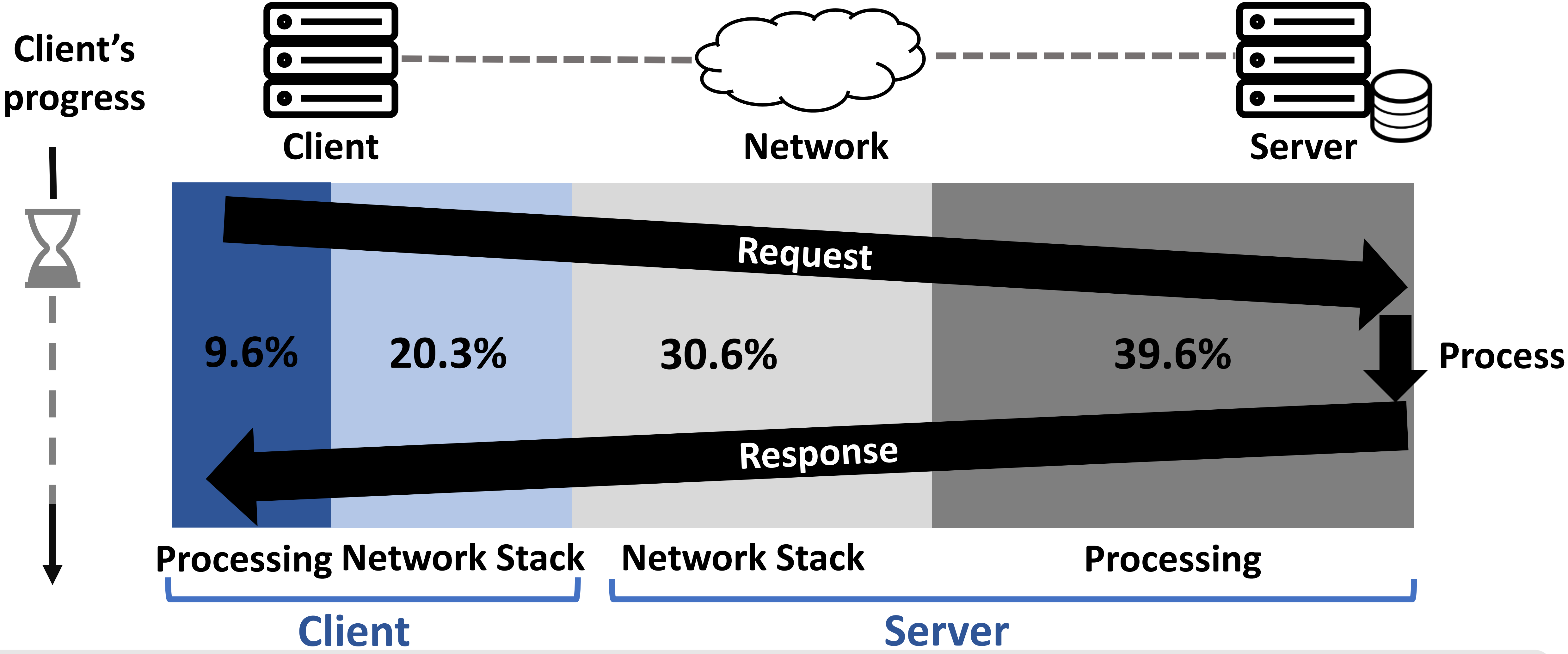
Storage Servers

🏛 **A** transfers \$10 to **B**
B receives \$10 from **A**

A’s balance - 10
B’s balance + 10

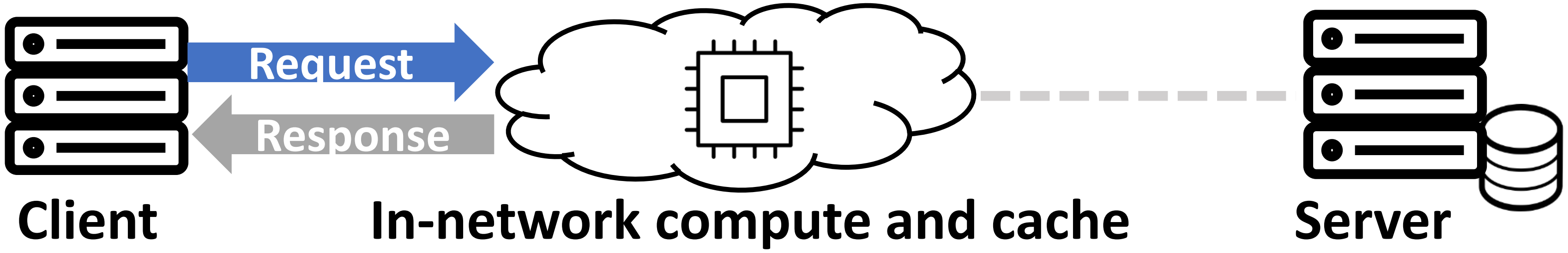
Update requests can be **latency-critical**

Latency of Accessing a Data Store



Observation: Majority of the latency is spent on the **server side**

Existing Approaches



In-network compute [Brainwave NPU ISCA’18, iSwitch ISCA’19, E3 ATC’19, iPipe SIGCOMM’19]

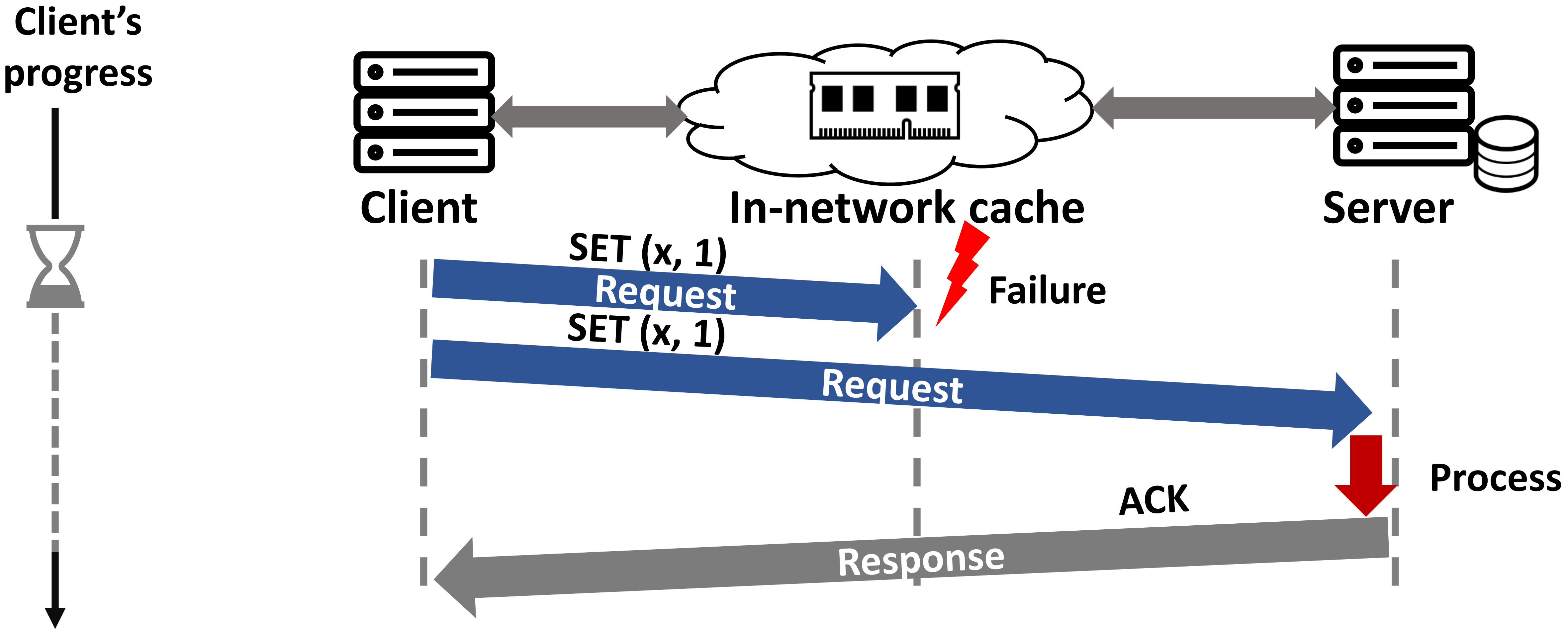
- Add compute logic to network devices, such as switches and NICs
- Reduce RTT of compute tasks

In-network data caching [NetCache SOSP’17, Incbricks SOSP’17, DistCache FAST’19]

- Add volatile cache in network devices
- Reduce RTT of GET requests

Some **read** requests can be served faster from the network device

In-network Caching: Update Requests



In-network caching has no **persistent storage** and cannot serve update requests

Outline

Background and Motivation

In-network Data Persistence

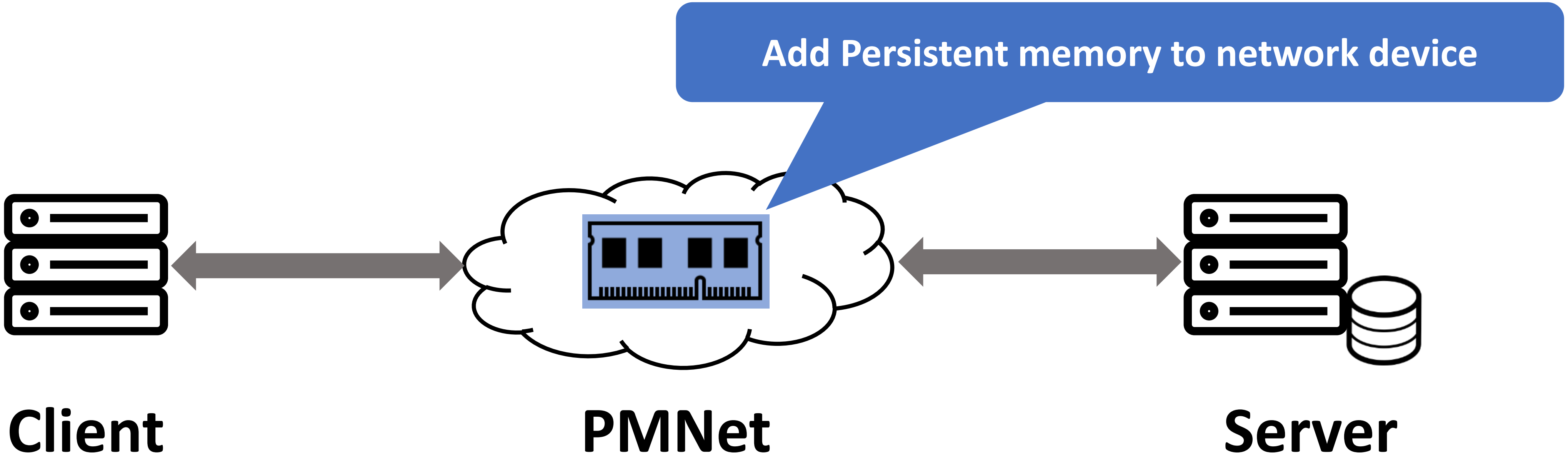
PMNet Design

Caching and Replication

Evaluation

Conclusion

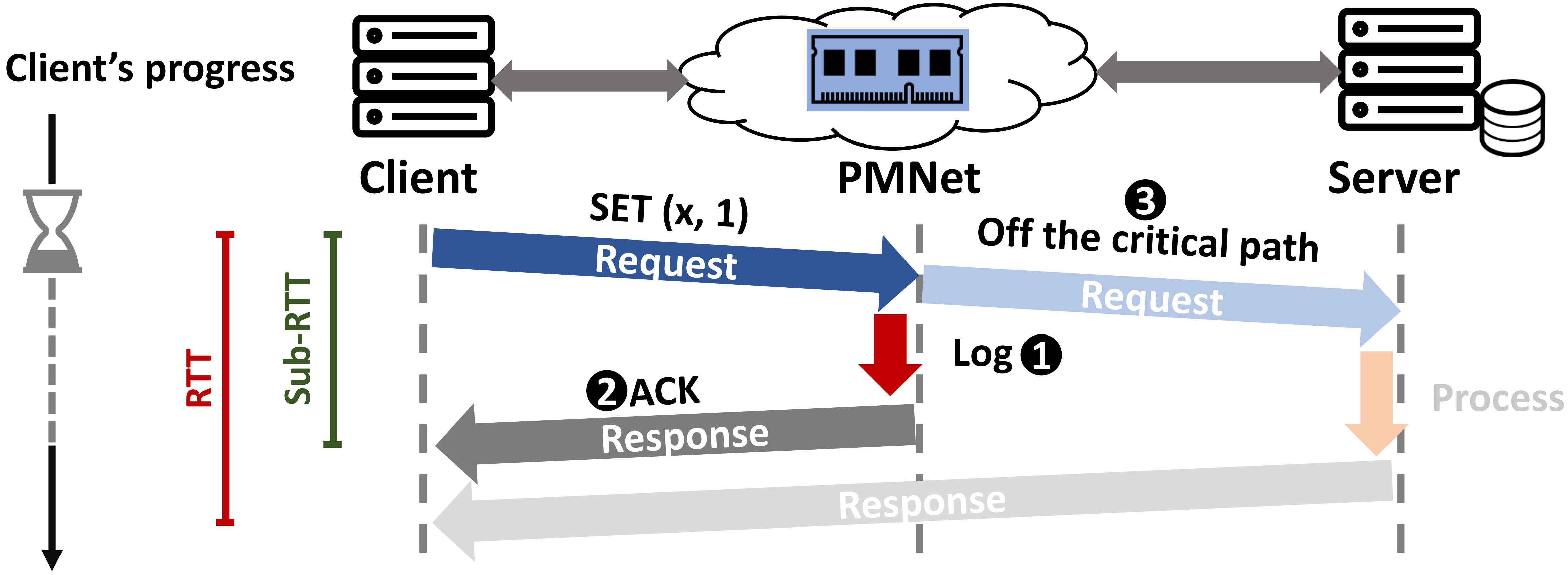
Our Proposal: **In-network Data Persistence**



Add persistent memory to **log** update requests

Key Idea: **Persistent Logging**

❶ Log request
❷ Send ACK to unblock the client as soon as the update request persists
❸ Forward the request to the server

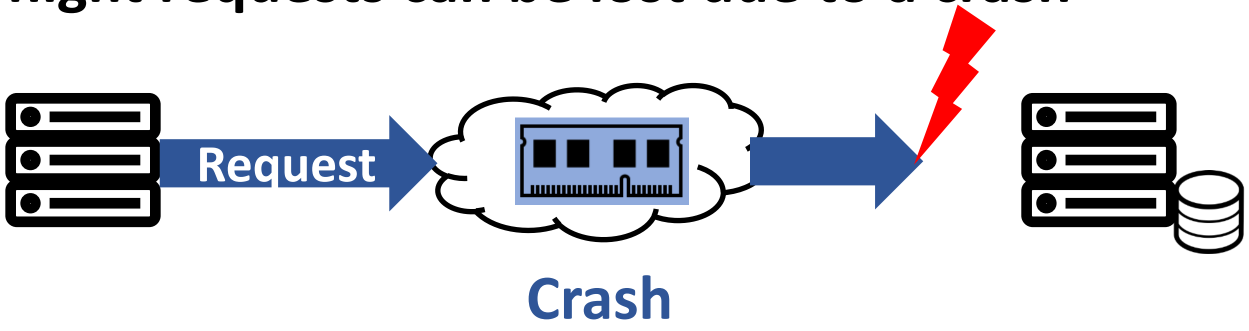


PMNet enables **sub-RTT** data persistence in the network

Persistent logging Challenges

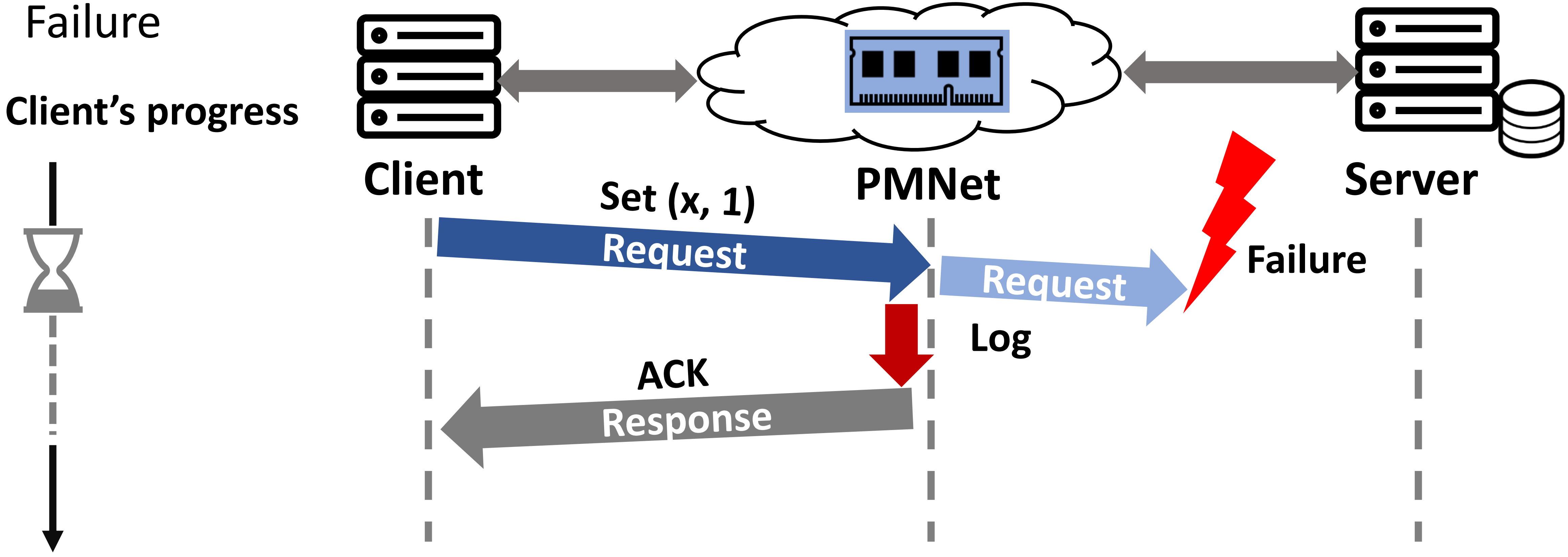
How to recover lost packets?

- **In-flight requests can be lost due to a crash**



Challenge: **System Recovery**

Failure

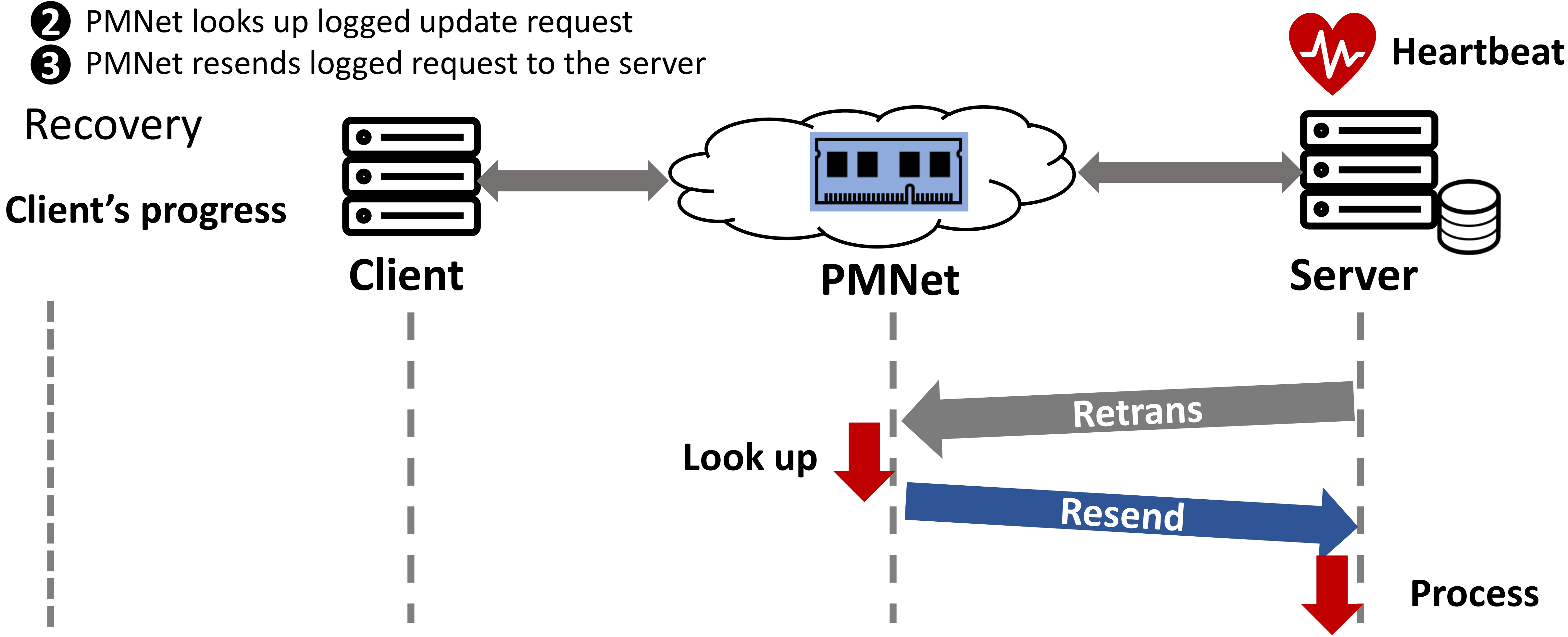


The client receives ACK and **cannot resend** the request

Solution: **Recover from Persistent Logs**

❶ Server sends Retrans
❷ PMNet looks up logged update request
❸ PMNet resends logged request to the server

Recovery

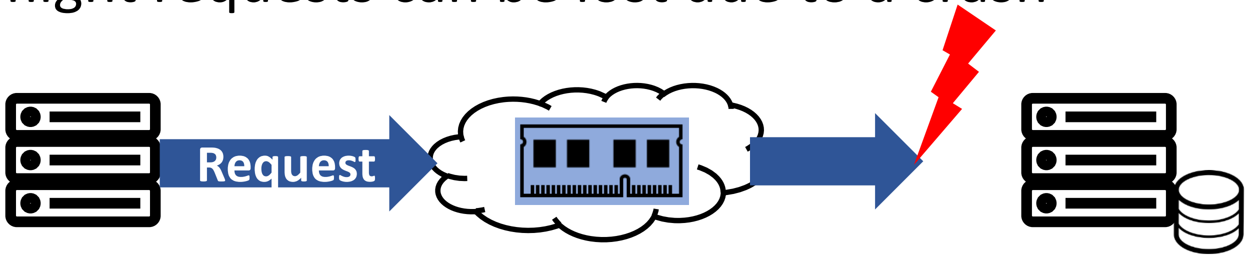


PMNet recovers lost requests from **persistent logs**

Persistent logging Challenges

How to recover lost packets?

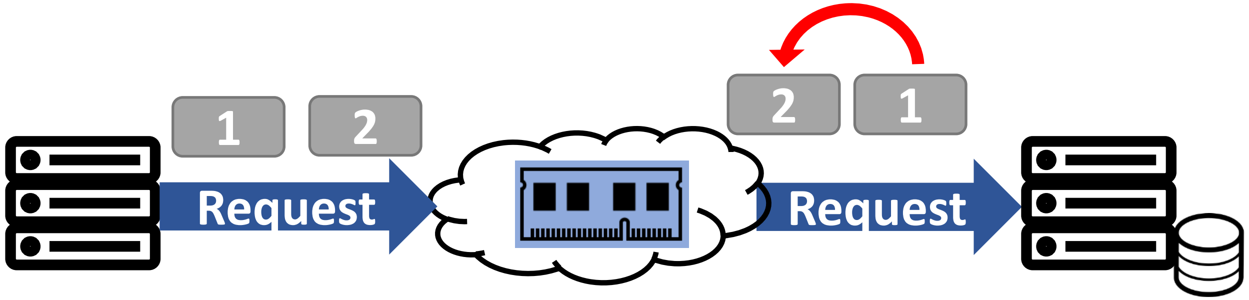
- In-flight requests can be lost due to a crash



Crash

How to ensure correct ordering?

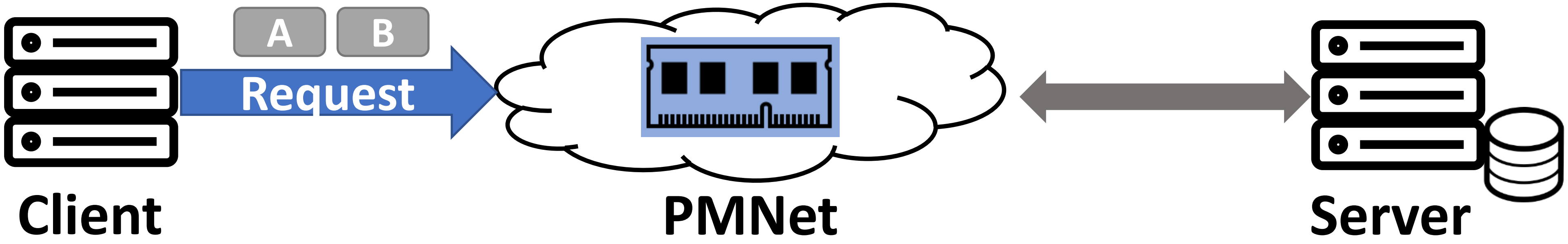
- **Requests from the same client can be reordered in the network**



Single-Client Reordering

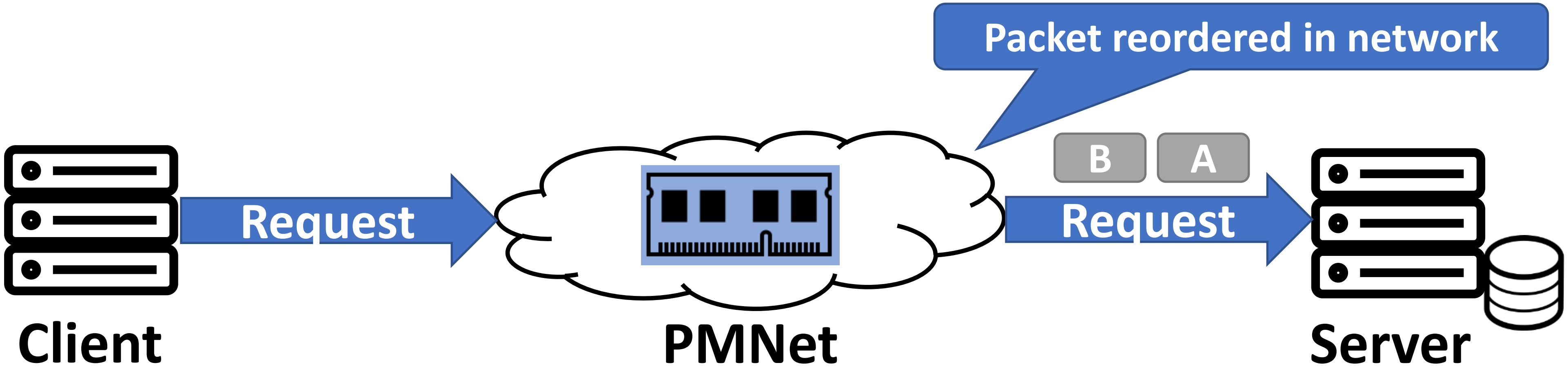
Challenge: **Request reordering**

Single client ordering



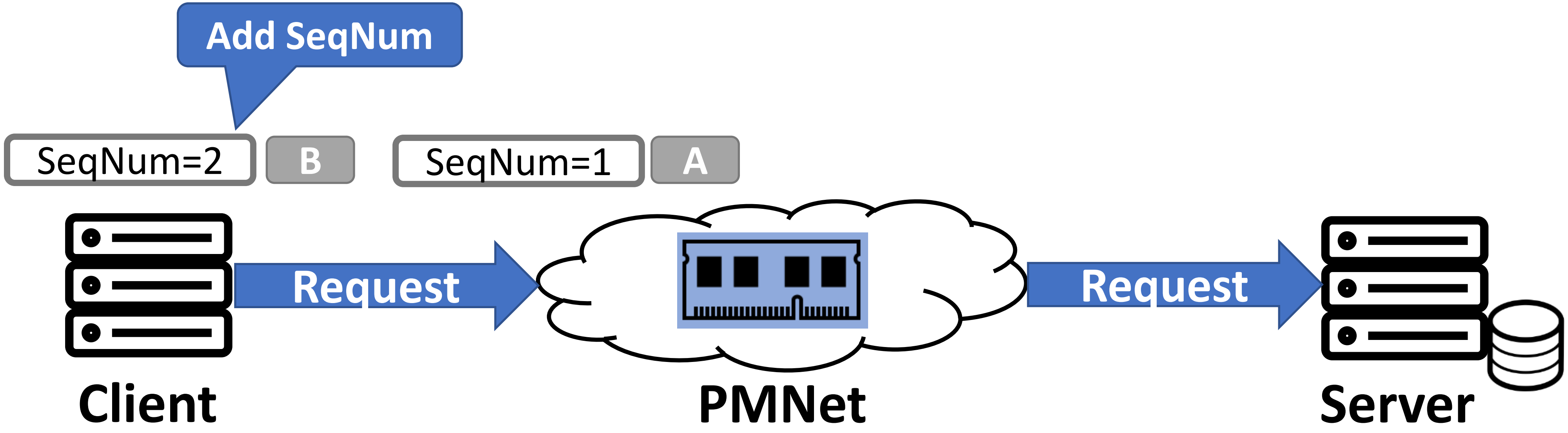
Challenge: **Request reordering**

Single client ordering



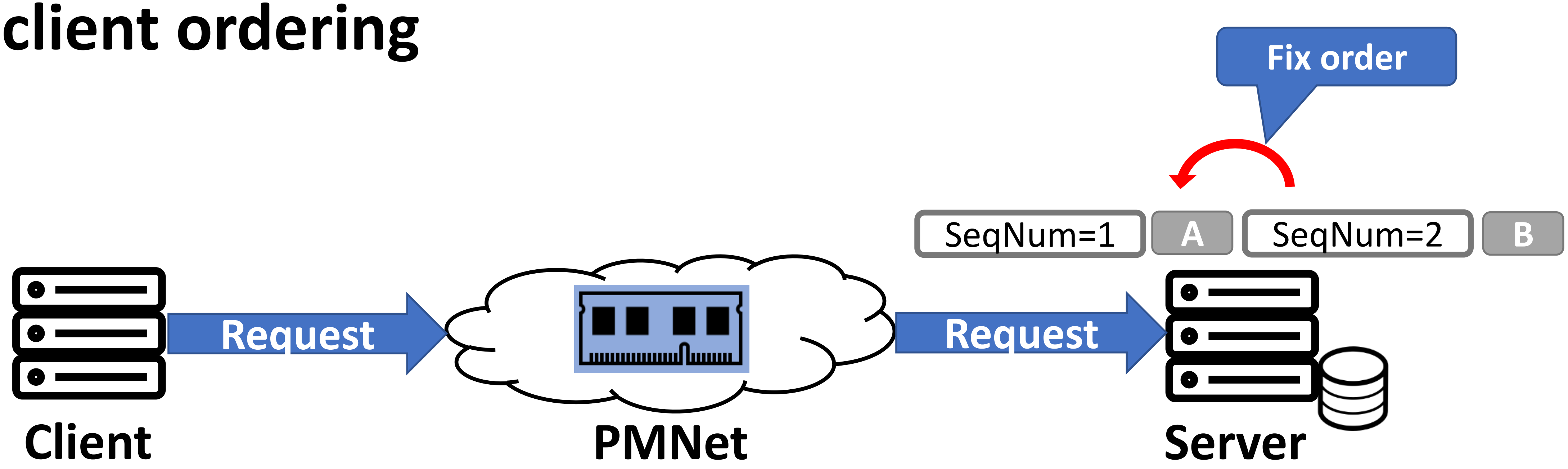
Solution: Single-client Ordering

Single client ordering



Solution: Single-client Ordering

Single client ordering

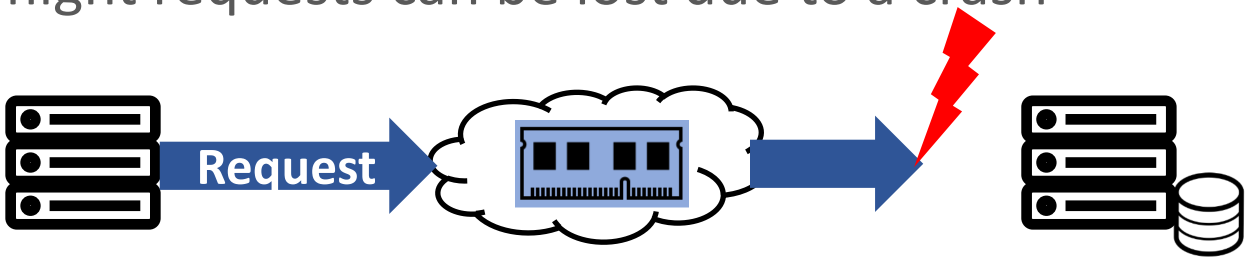


The server corrects order based on **Sequence Number** in each packet

Persistent logging Challenges

How to recover lost packets?

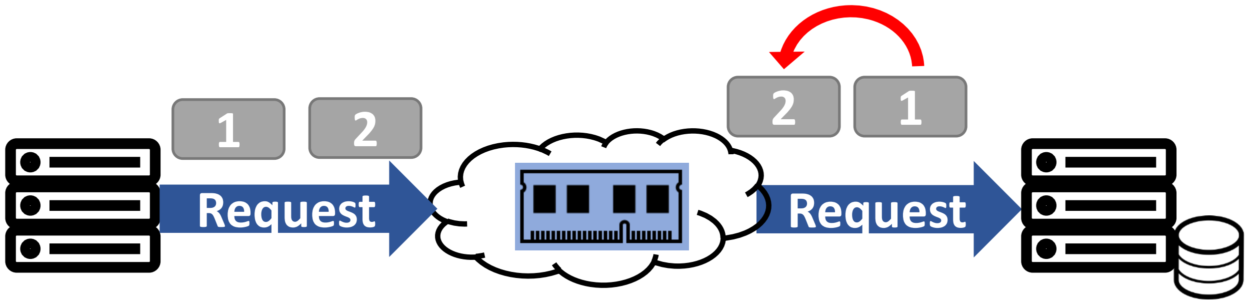
- In-flight requests can be lost due to a crash



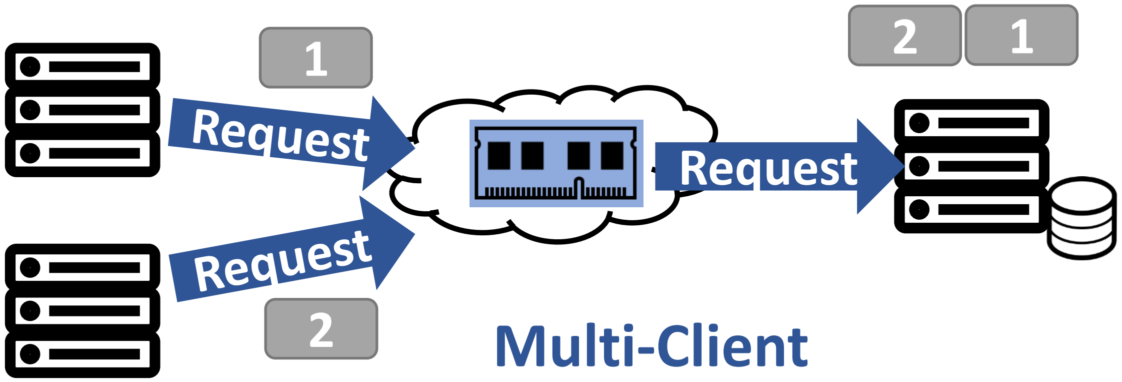
Crash

How to ensure correct ordering?

- Requests from the same client can be reordered in the network
- **Requests from multiple clients can become out of order**



Single-Client Reordering

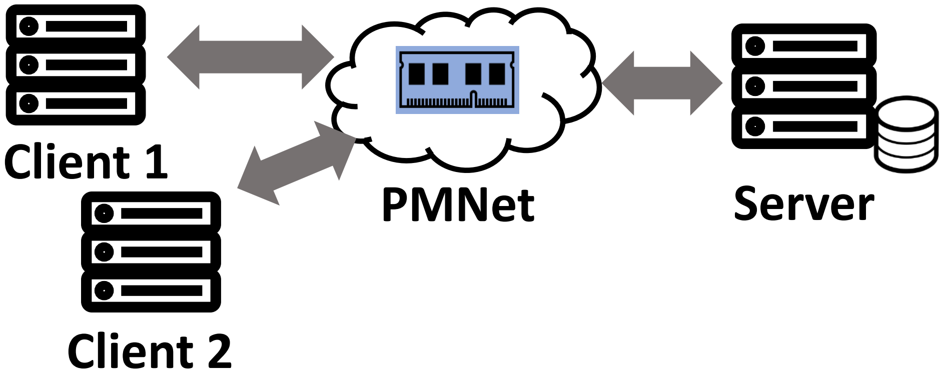


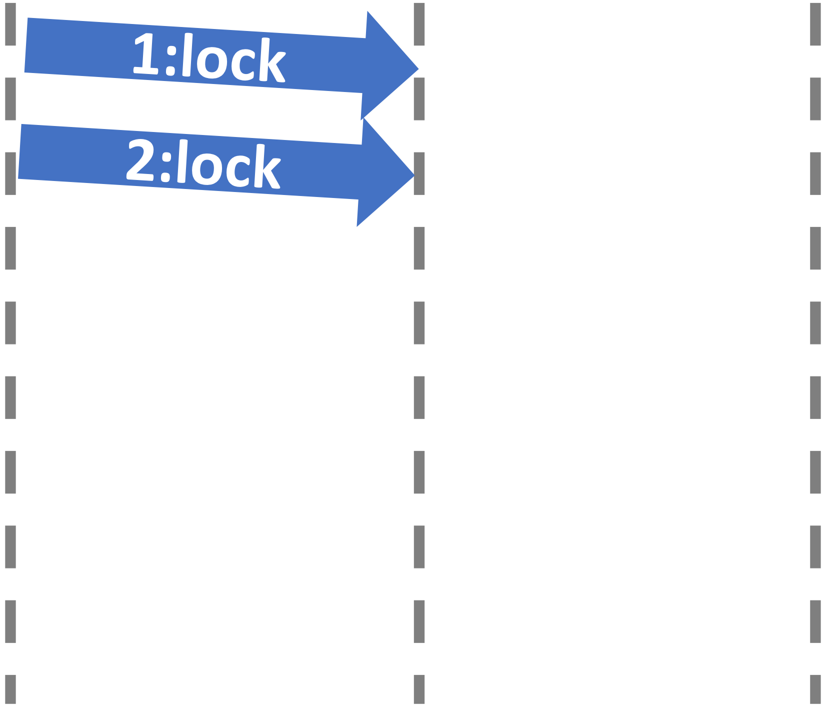
Multi-Client Reordering

Challenge: **Request reordering**

Multi client ordering

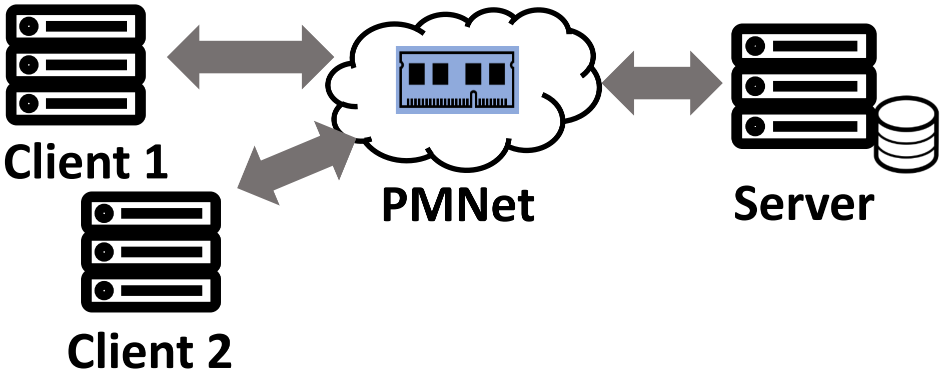
Time	Client 1	Client 2
1	**lock()**	**lock()**
2	tasks()	tasks()
3	unlock()	unlock()
4		



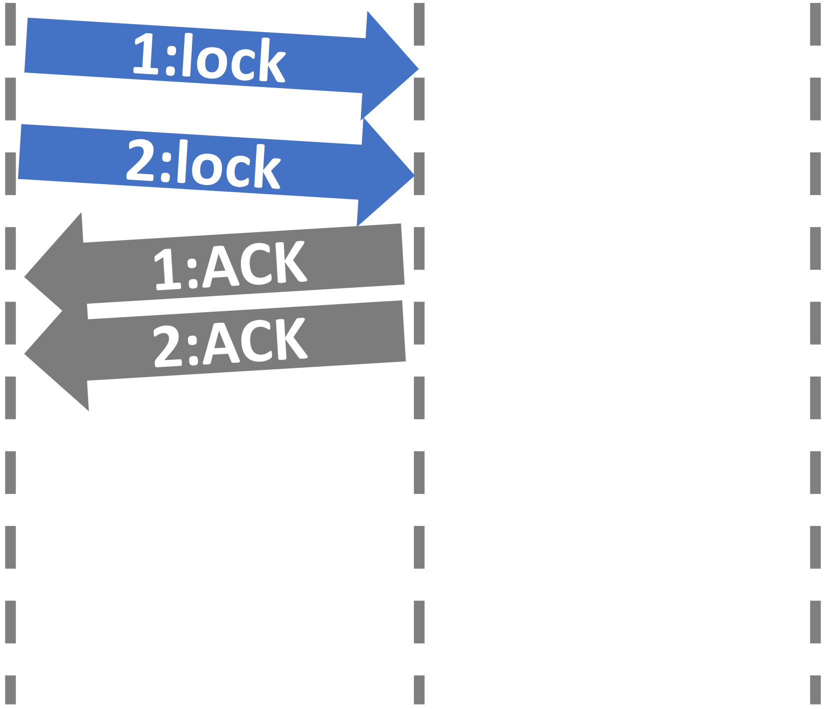


Challenge: **Request reordering**

Multi client ordering



```
Time  Client 1        Client 2
 1    lock()          lock()
 2    tasks()         tasks()
 3    unlock()        unlock()
 4
```



Both clients receive ACKs from PMNet

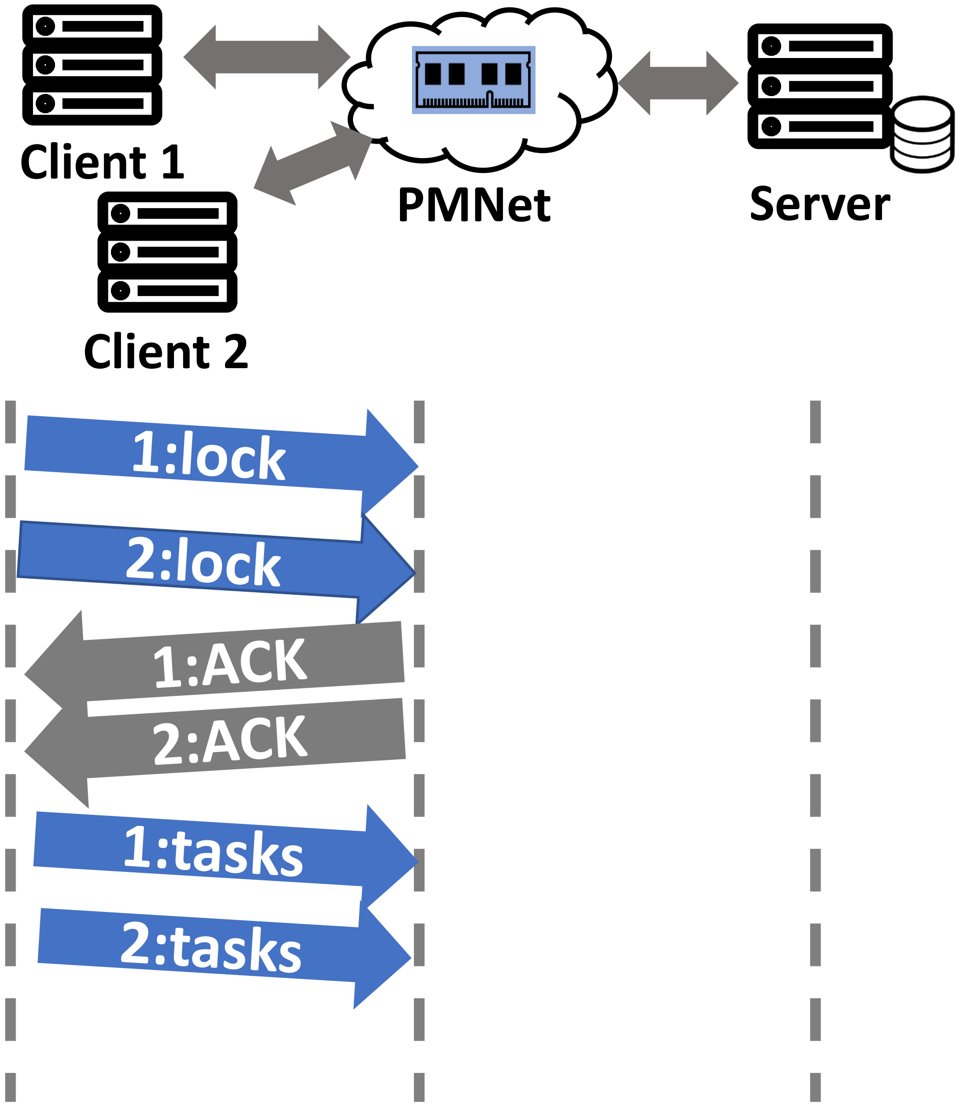
Challenge: **Request reordering**

Multi client ordering

Time	Client 1	Client 2
1	lock()	lock()
2	**tasks()**	**tasks()**
3	unlock()	unlock()
4		

✖

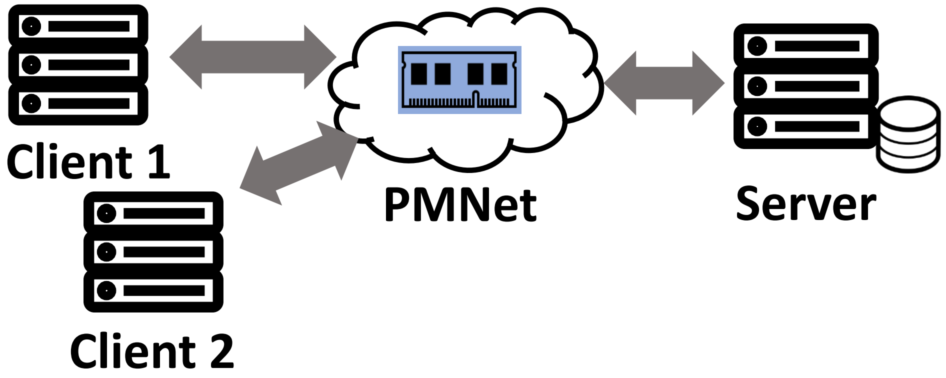
Both clients enter the critical section



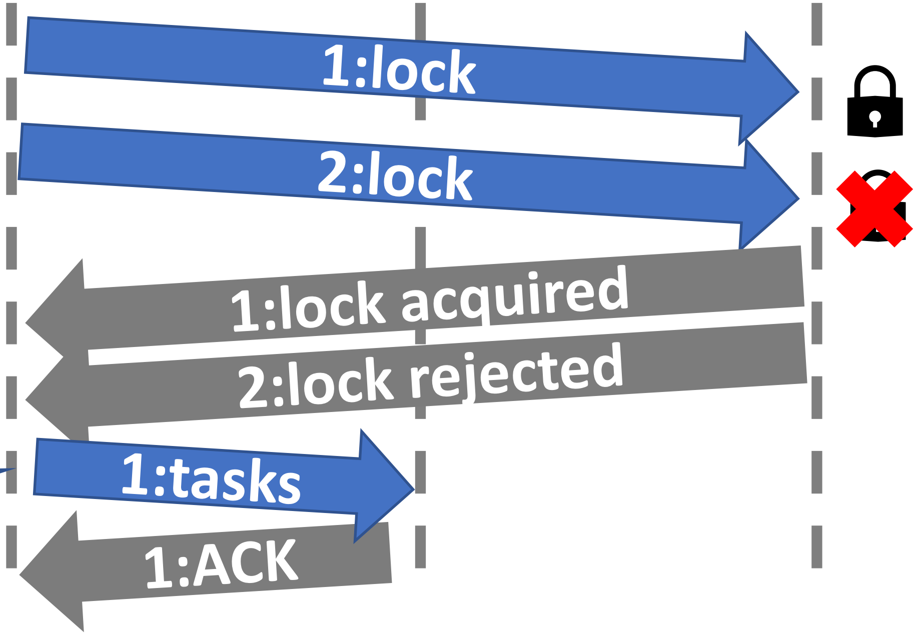
Solution: Multi-client Ordering

PMNet bypasses synchronization primitives to ensure single client enters the critical section

Updates in the critical section are still logged in PMNet



Time	Client 1	Client 2
1	lock()	**lock()**
2	**tasks()**	tasks()
3	unlock()	unlock()
4		



Only client 1 can enter the critical section

Bypass requests are infrequent: 13.7% in TPC-C

Outline

Background and Motivation

In-network Data Persistence

PMNet Design

Caching and Replication

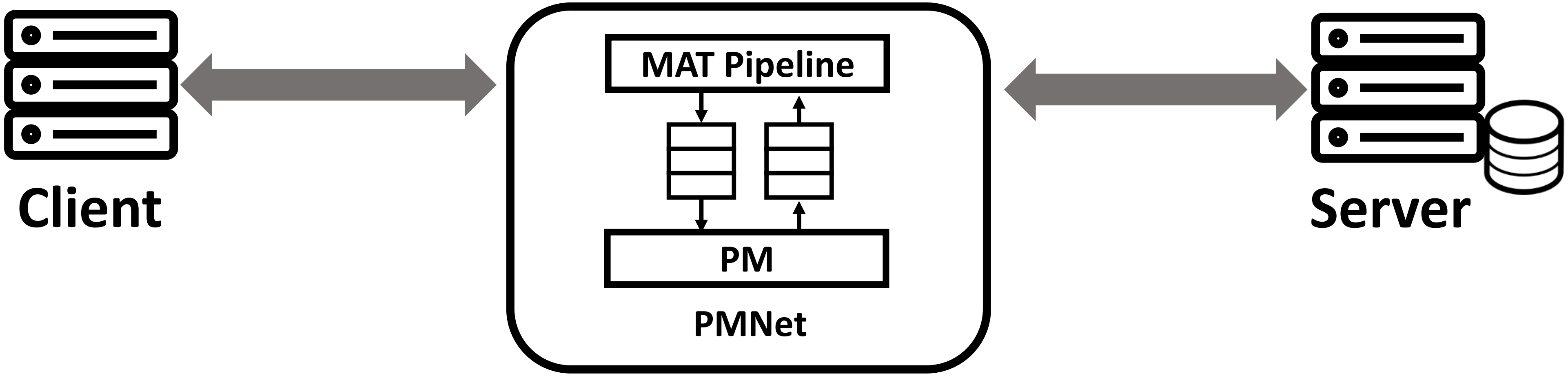
Evaluation

Conclusion

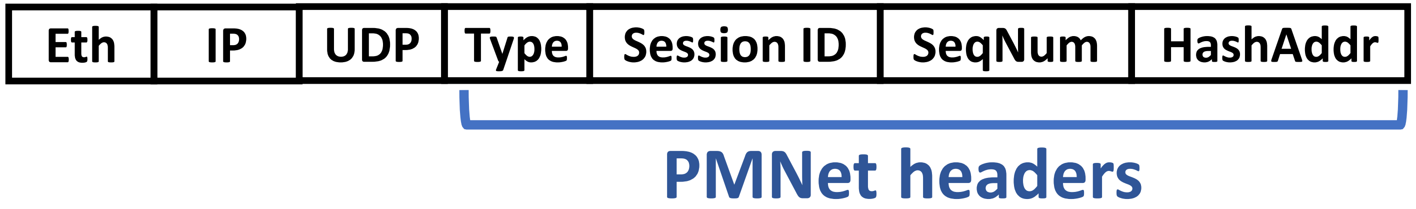
PMNet overview

PMNet pipeline: How does PMNet's hardware enable persistent logging?

PMNet protocol: How are PMNet packets defined?

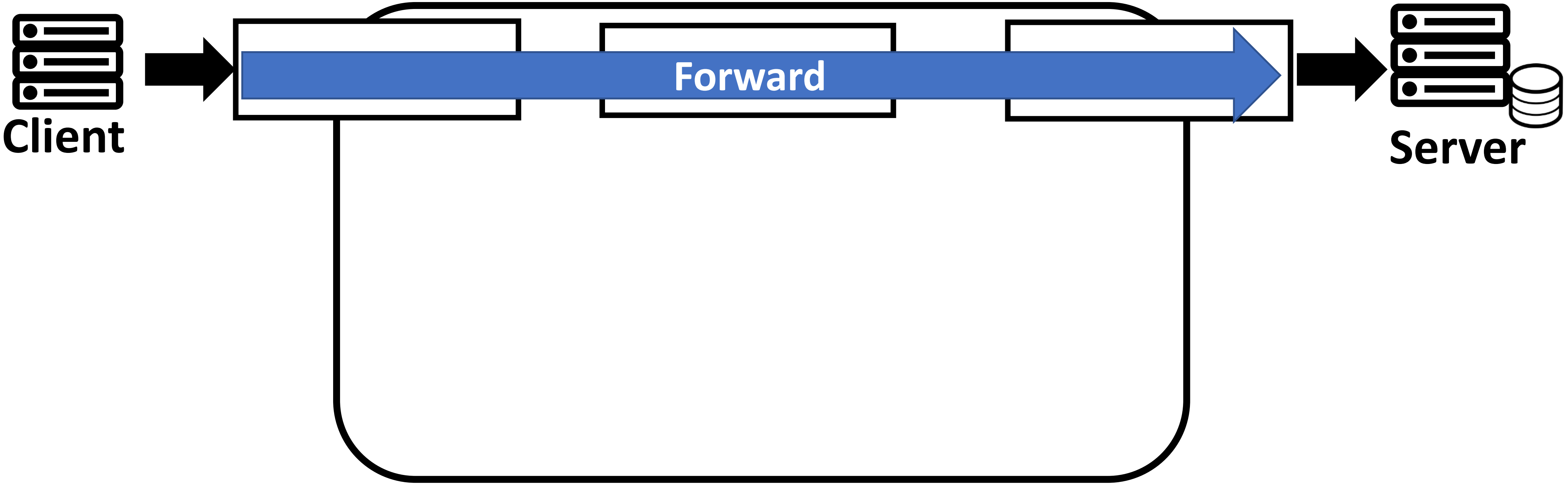


PMNet packet



Baseline NIC/Switch Architecture

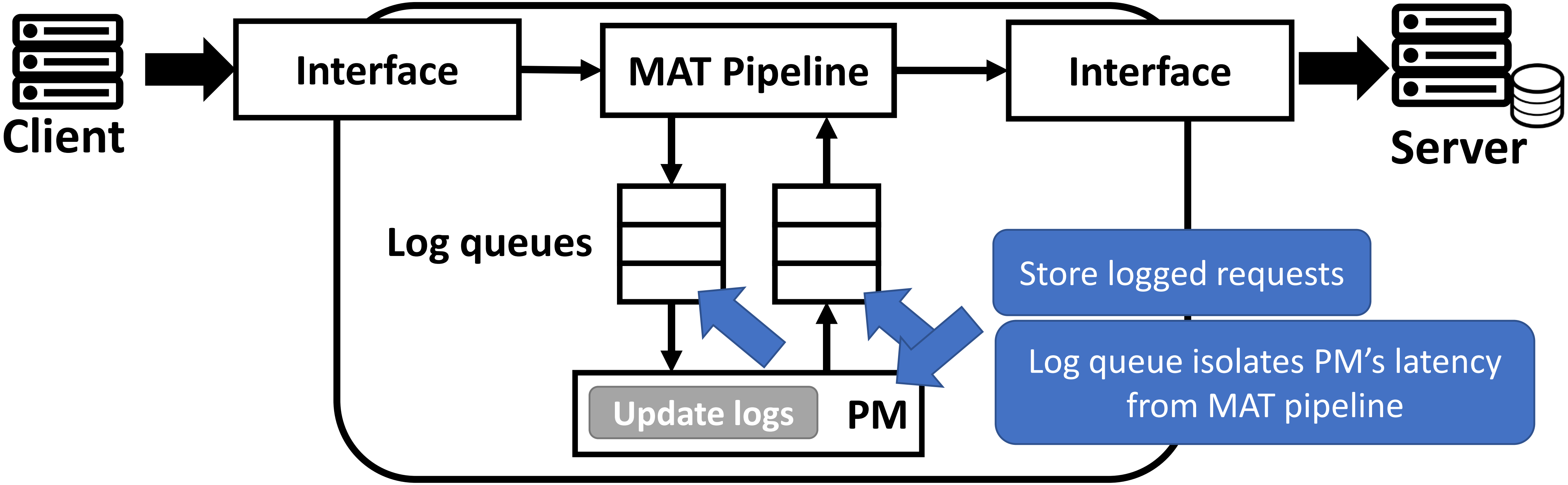
Baseline NIC and switch forward packet with rules in Match-action table (MAT) pipeline



Baseline NIC and switch **forwards** packet to the destination

PMNet NIC/Switch Architecture

PMNet NIC and switch's MAT pipeline process PMNet packet in addition to other packets



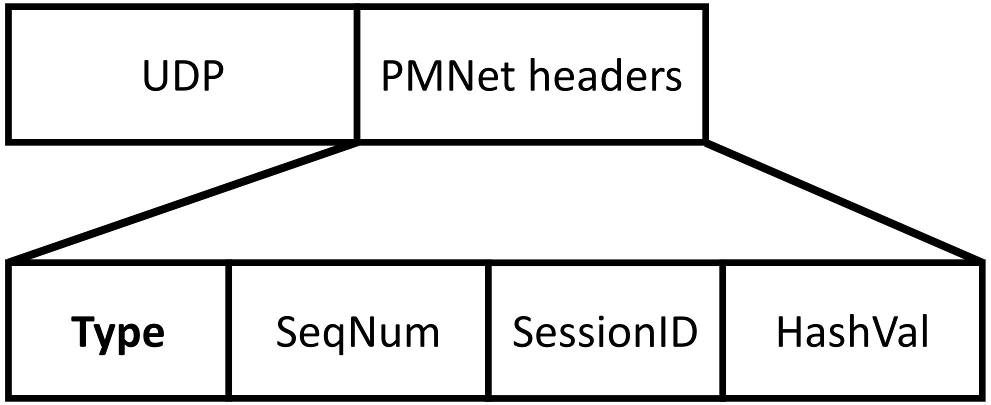
Store logged requests

Log queue isolates PM's latency from MAT pipeline

PMNet MAT Pipeline controls access to the **persistent memory**

PMNet Design: Protocol

Defines four packet types on top of UDP



UDP | PMNet headers

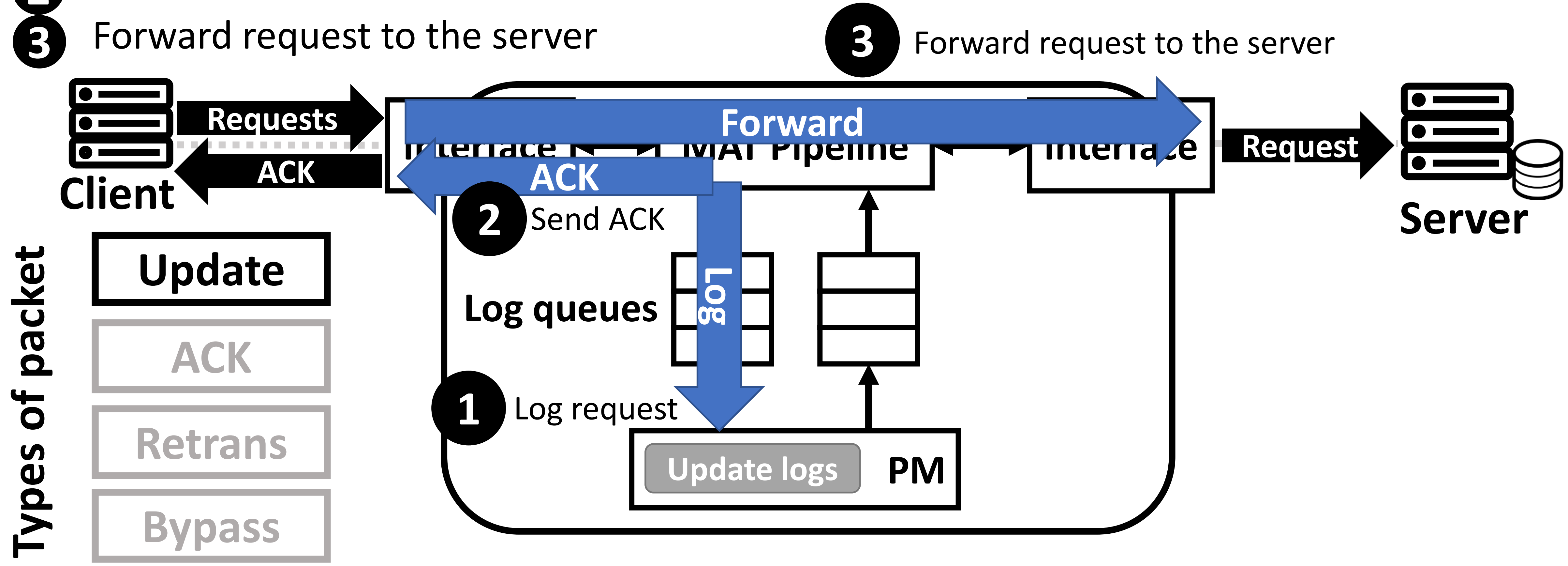
Type | SeqNum | SessionID | HashVal

Type	Action
Update	**Log+forward** request, send **ACK**
ACK	**Remove** log and **unblock** client
Retrans	**Resend** logged entry
Bypass	**Forward** request

PMNet performs different operation based on **packet type**

PMNet packet processing: Update requests

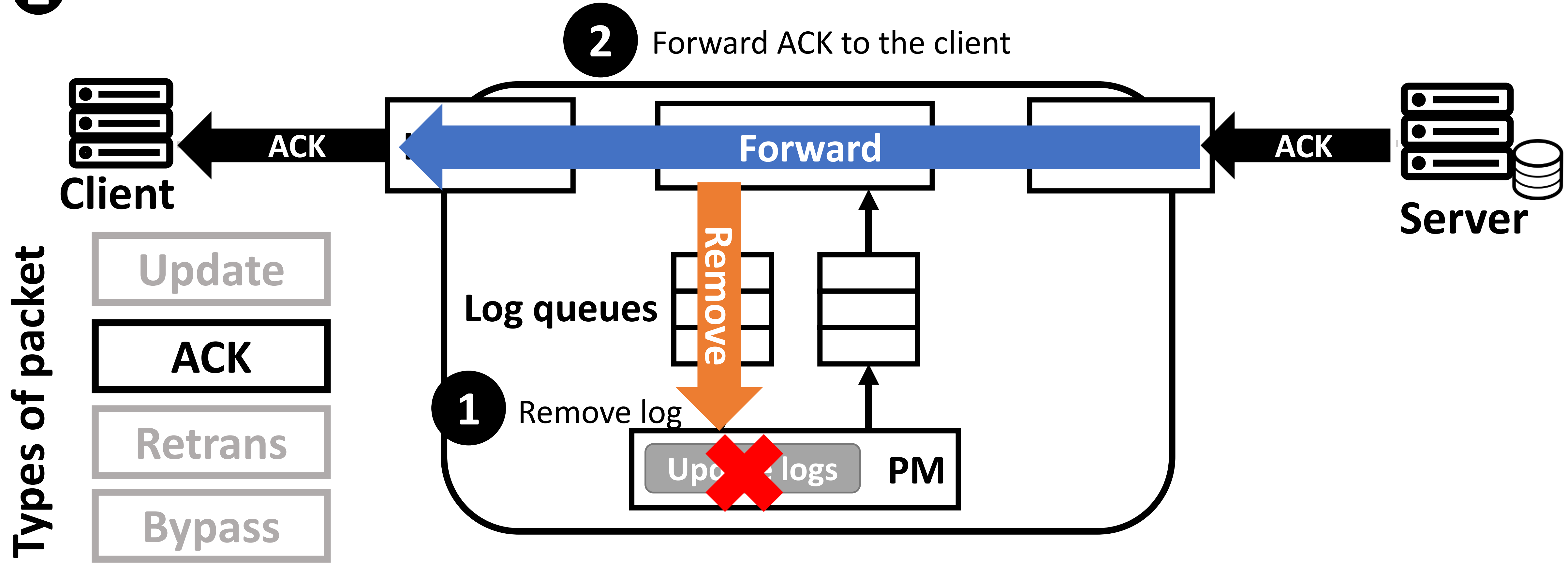
1. Log update request
2. Send ACK to client
3. Forward request to the server



PMNet **logs and forwards** update requests and sends **ACK** to unblock client

PMNet packet processing: ACK

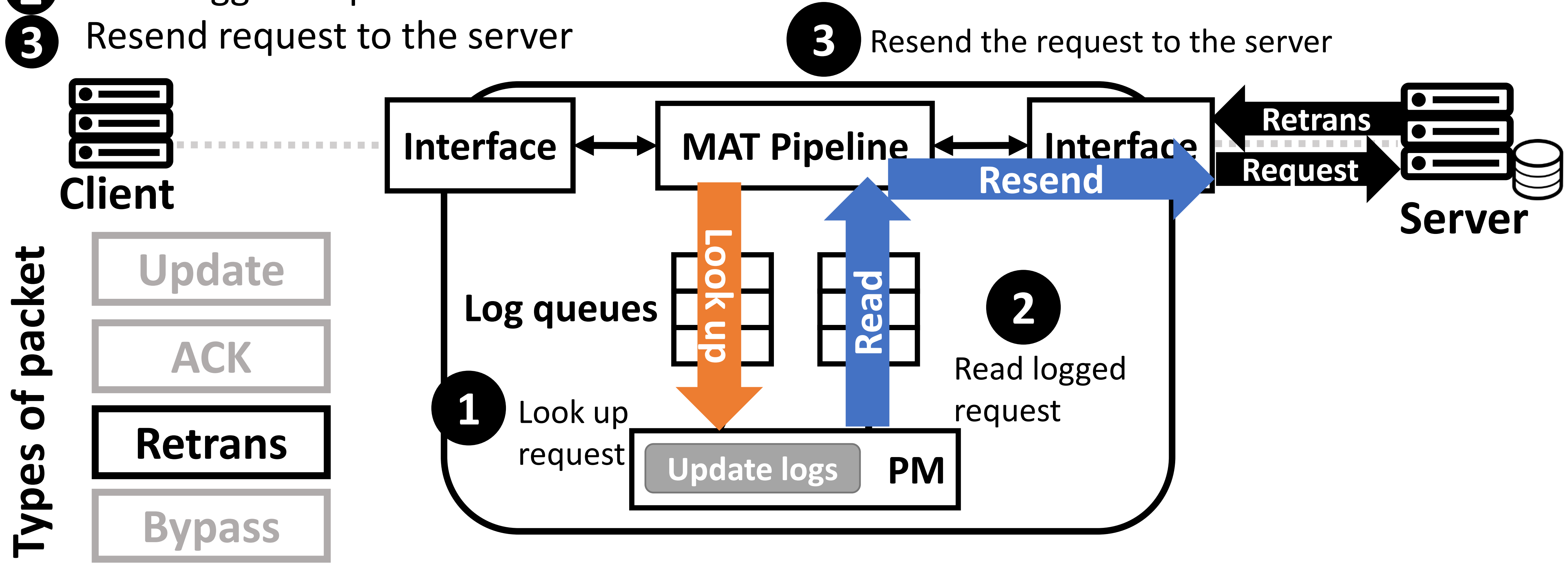
1. Remove logged request
2. Forward ACK to the client



PMNet ACK **removes** the logged entry

PMNet packet processing: Retrans

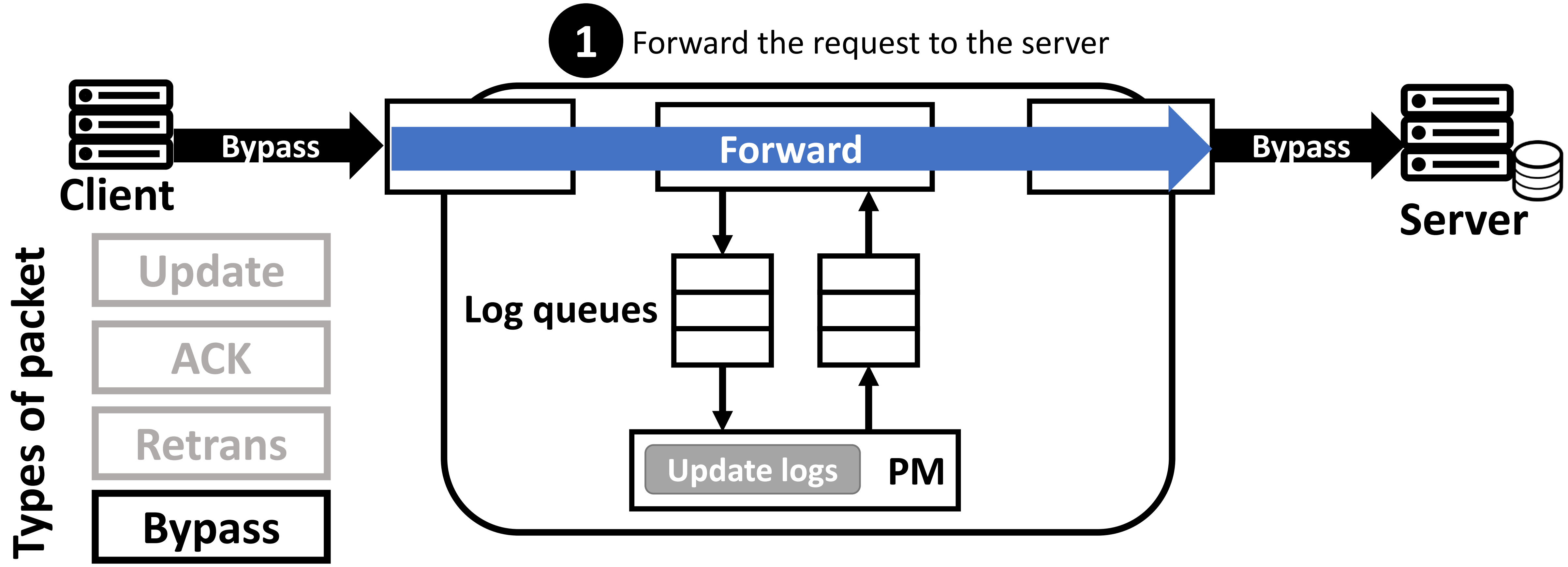
1. Look up logged request
2. Read logged request
3. Resend request to the server



PMNet Retrans **resends** the logged entry

PMNet packet processing: Bypass

❶ Forward the request

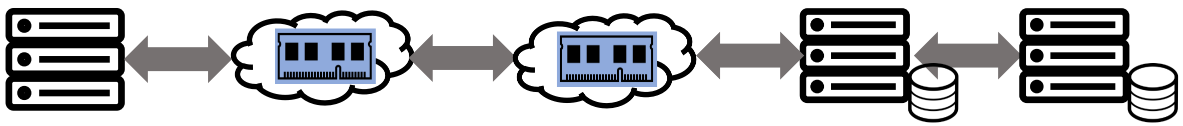


PMNet **forwards** Bypass packets

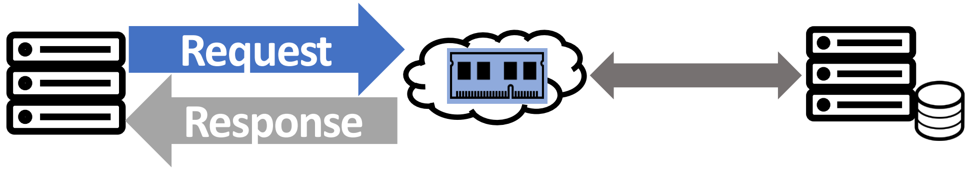
PMNet Design

PMNet **logs update requests** to move the server off the critical path

Applications of PMNet:



Replication



Caching

Outline

Background and Motivation

In-network Data Persistence

PMNet Design

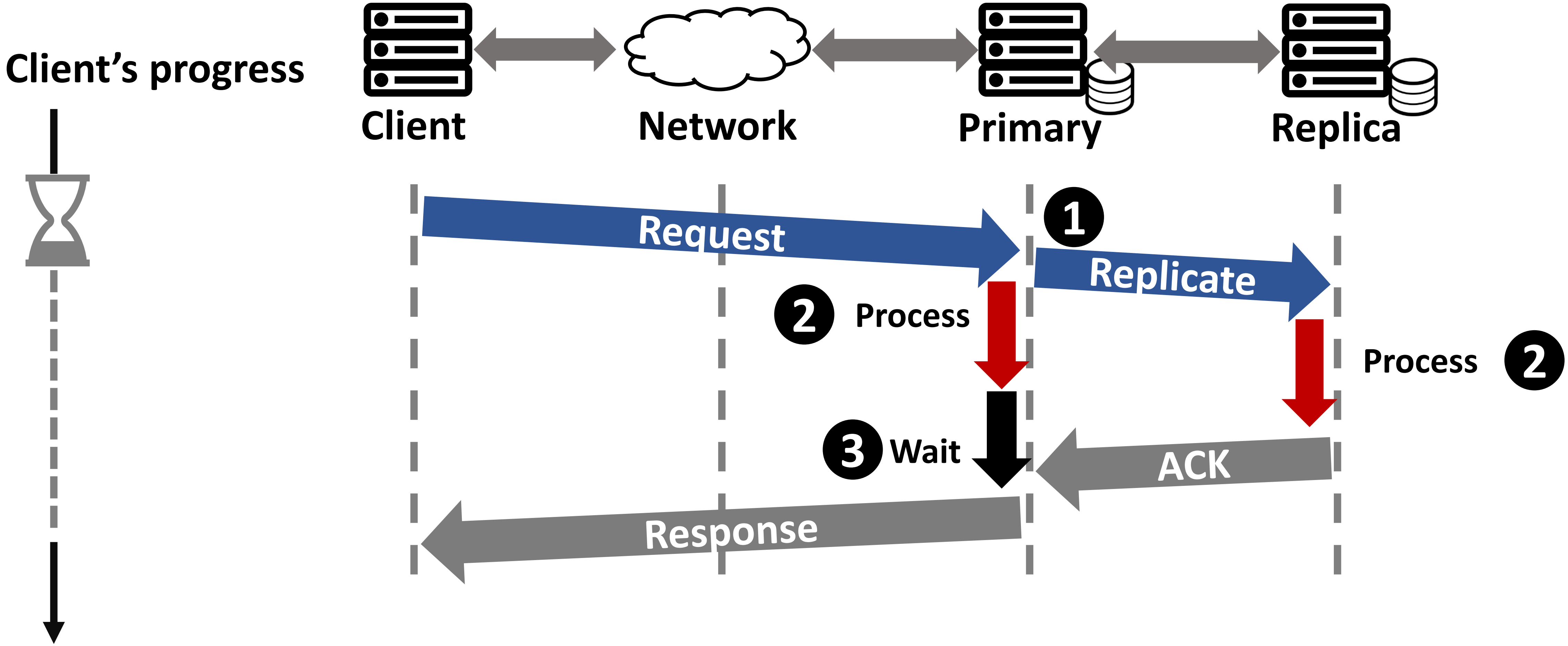
Caching and Replication

Evaluation

Conclusion

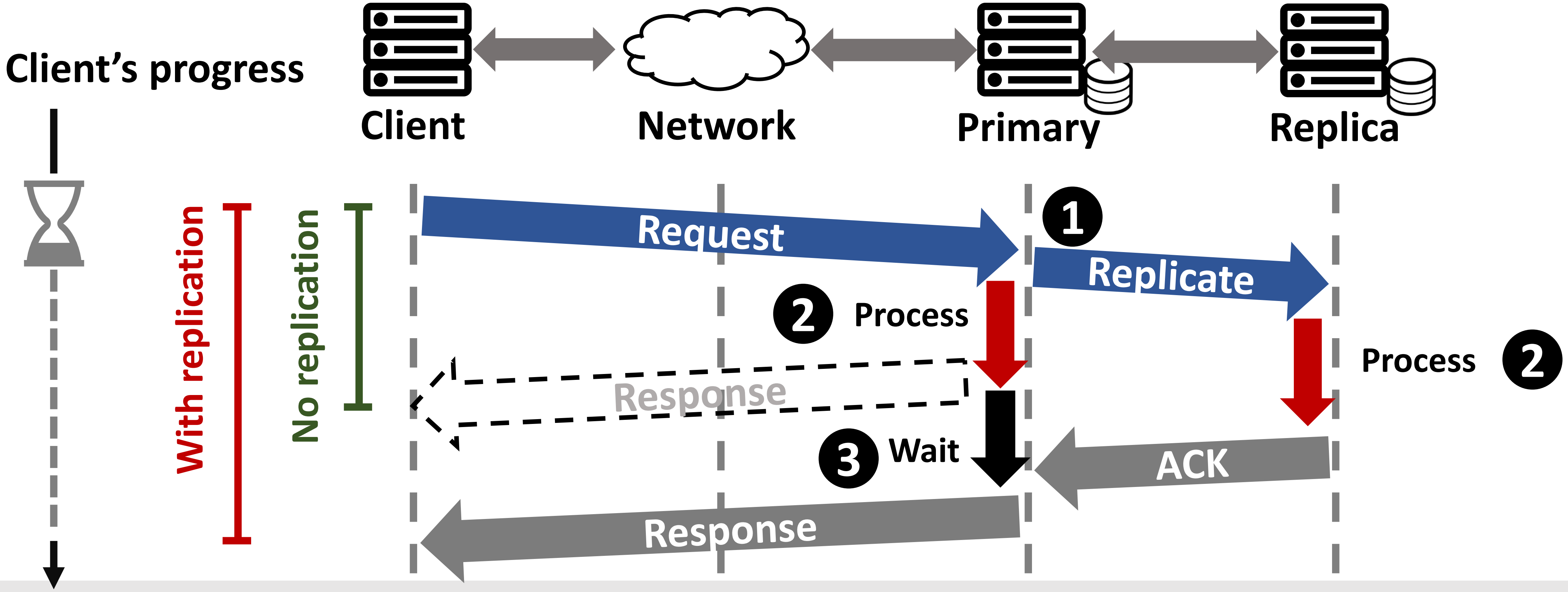
PMNet Replication: Baseline Replication

❶ Replicate request to all servers
❷ Process the request
❸ Wait until all servers respond



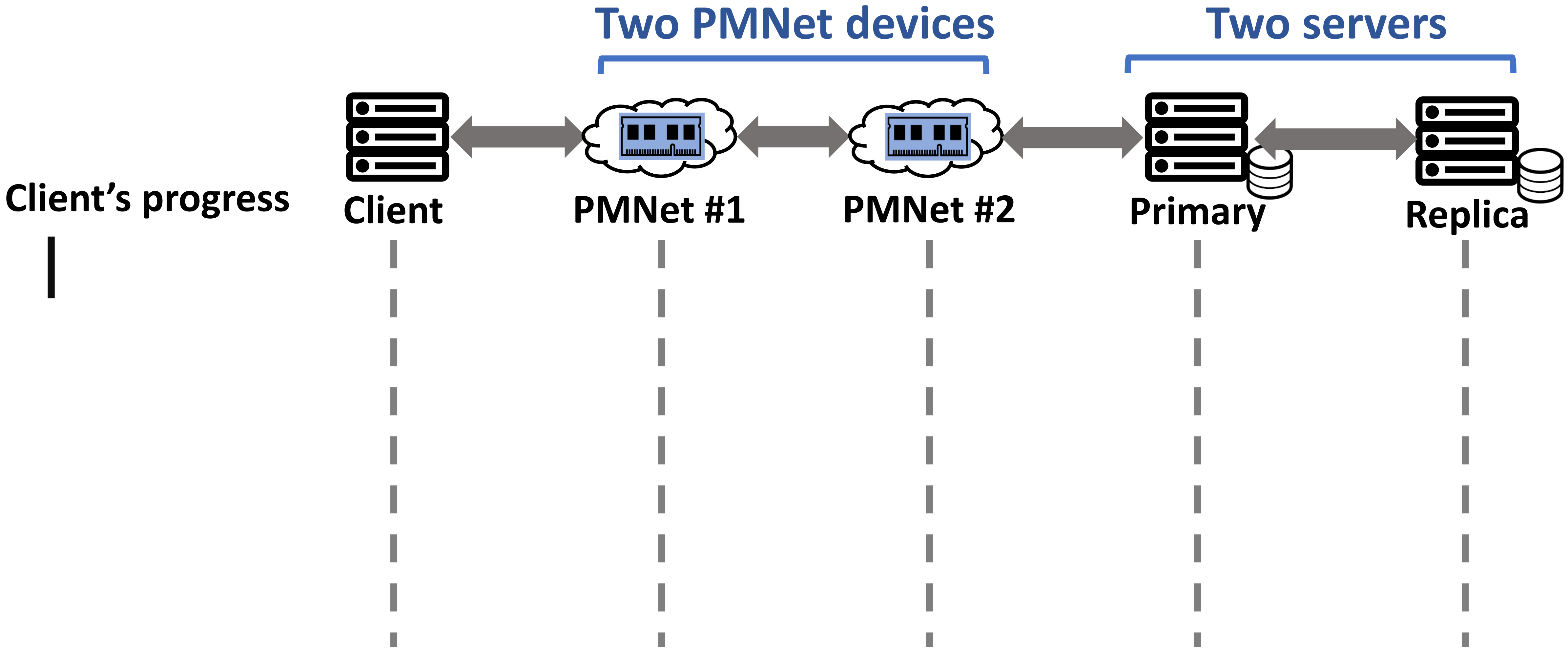
PMNet Replication: Baseline Replication

❶ Replicate request to all servers
❷ Process the request
❸ Wait until all servers respond



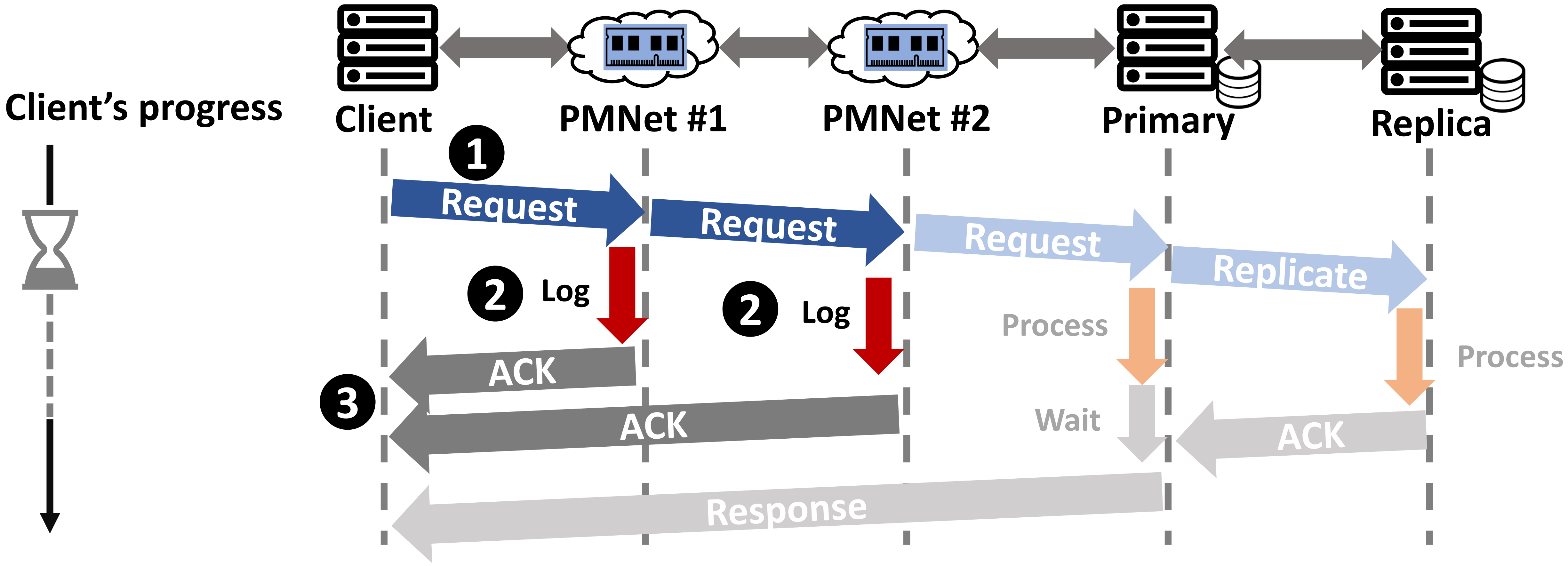
Replication increases **blocking latency**

PMNet Replication: Replication with PMNet



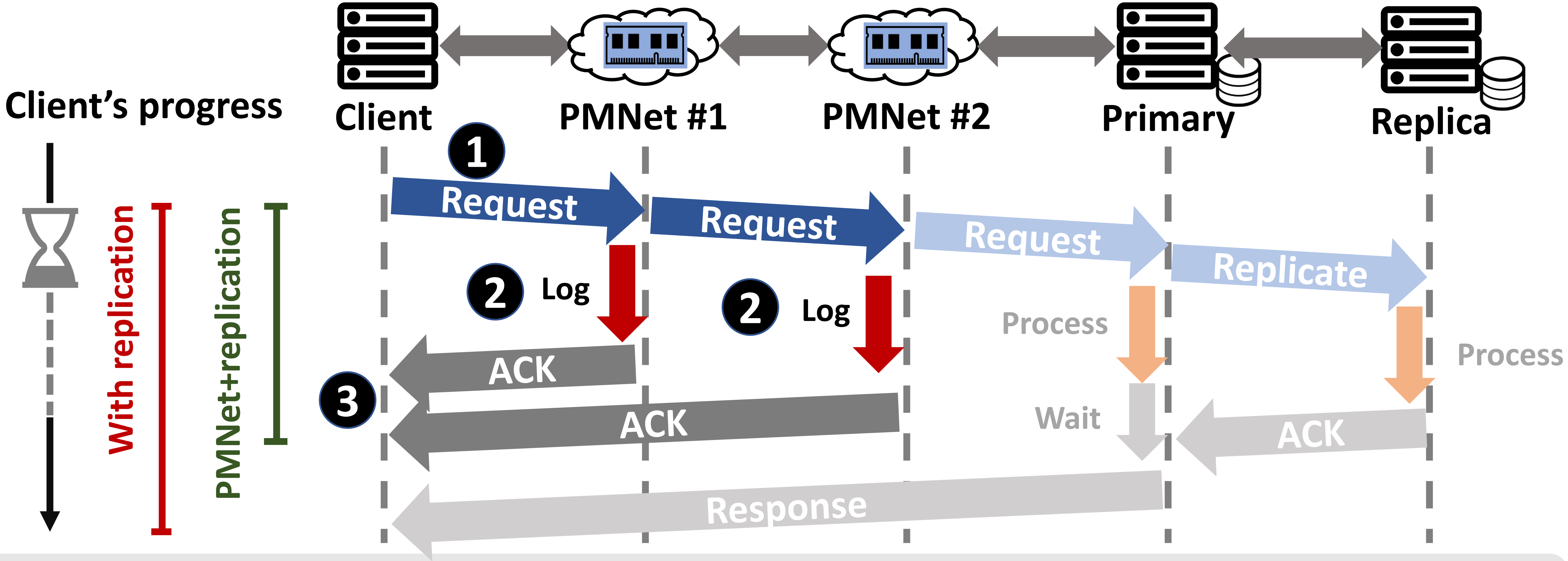
PMNet Replication: Replication with PMNet

❶ The client sends the request and waits for 2 ACKs
❷ PMNet #1 and #2 log the request and send ACK to the client
❸ Client waits until it receives both ACKs



PMNet Replication: Replication with PMNet

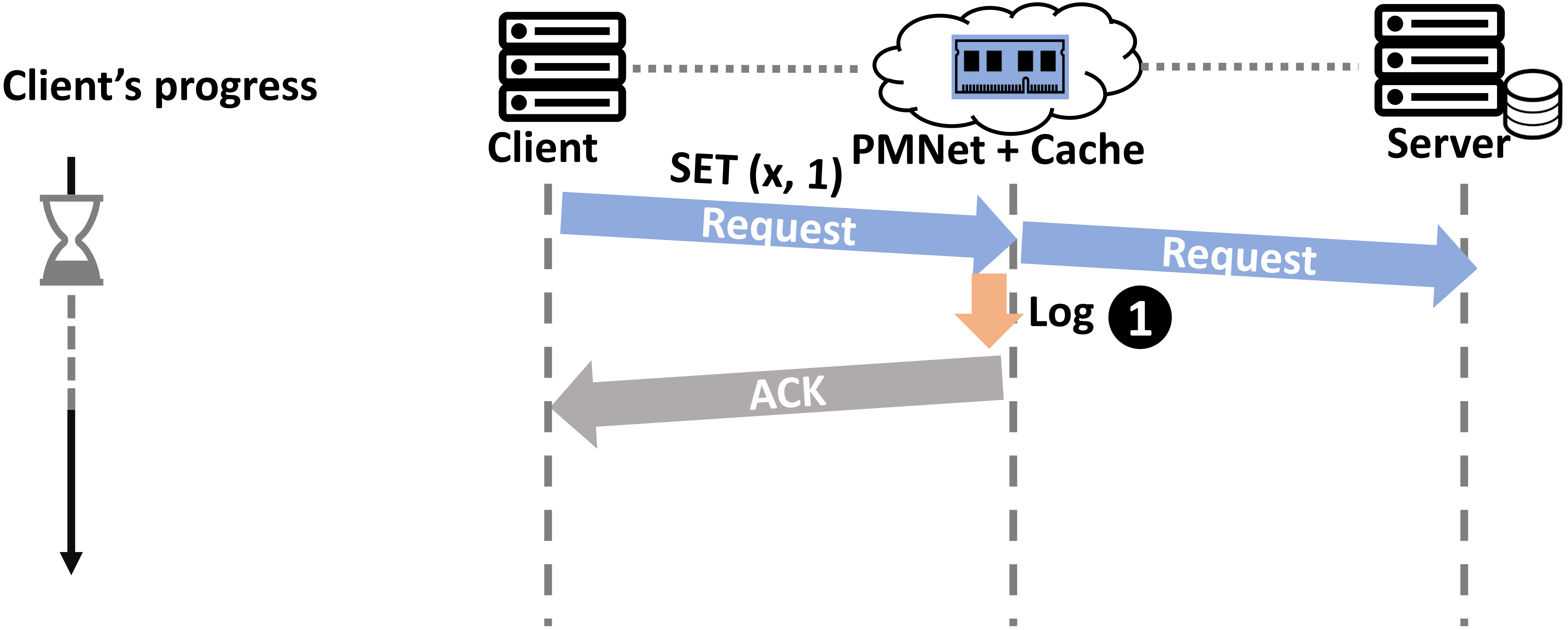
1. The client sends the request and waits for 2 ACKs
2. PMNet #1 and #2 log the request and send ACK to the client
3. Client waits until it receives both ACKs



PMNet moves replication **off the critical path** with the **same level of protection**

In-network Caching: Update Requests

❶ PMNet logs update requests

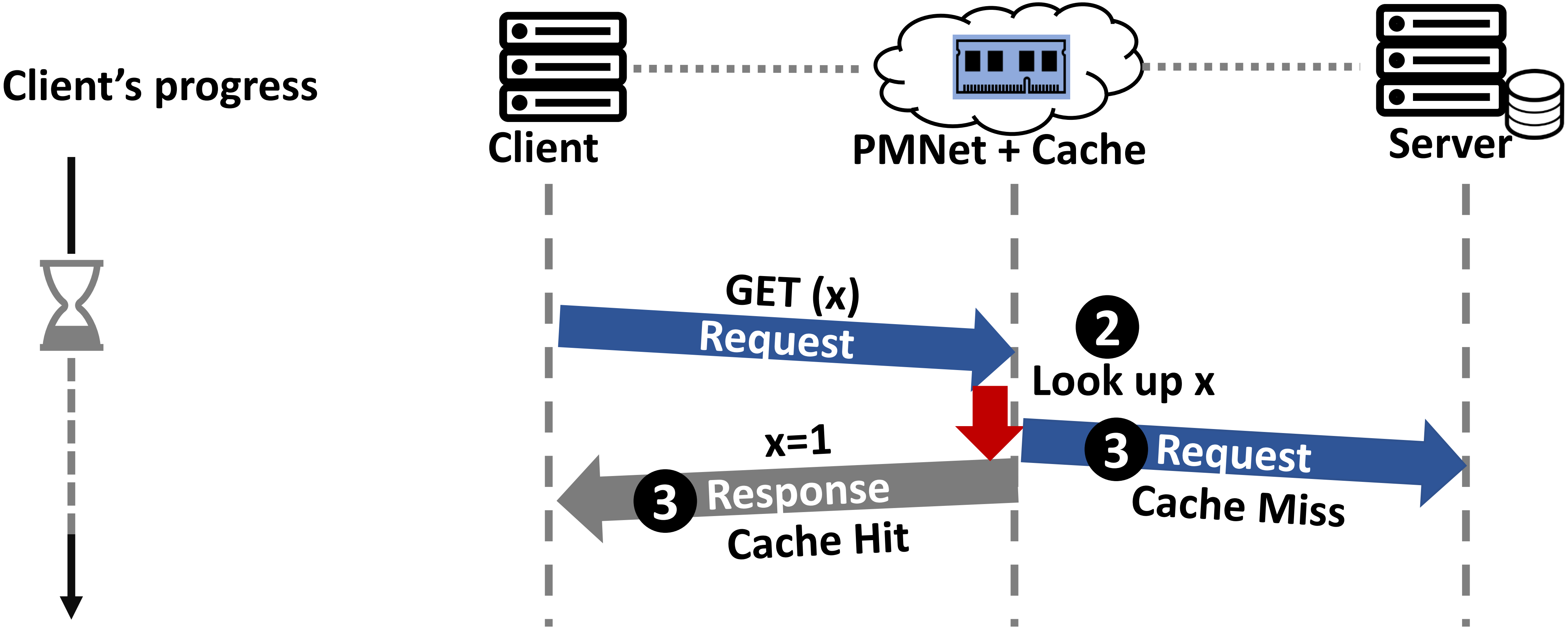


In-network Caching: Read Requests

❶ PMNet logs update requests

❷ PMNet receives read request and looks up an associated logged request in the PM

❸ PMNet responds the read request (Hit) or forward the request (Miss)

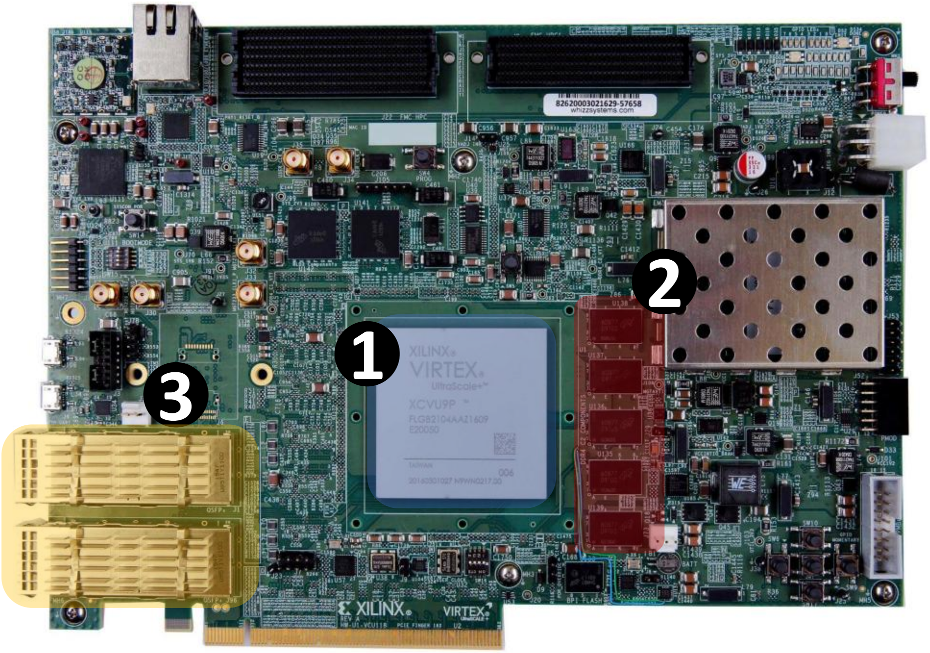


PMNet can use logged entry to respond **read** requests.

Methodology

Hardware

PMNet	Xilinx VCU118 Evaluation platform
Server	Intel Cascade Lake, 20 Cores, 192GB DRAM, 256GB DCPMM
Client	Intel Haswell, 6 Cores, 64GB DRAM



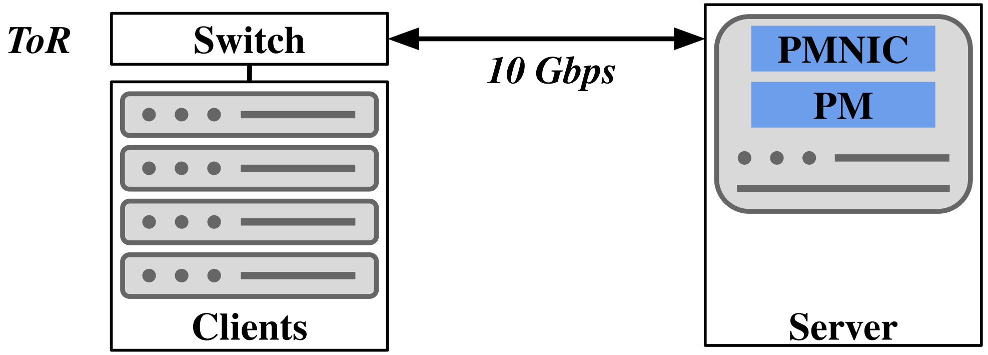
PMNet evaluation platform

❶ MAT pipeline
❷ Emulated persistent memory
❸ Network interfaces

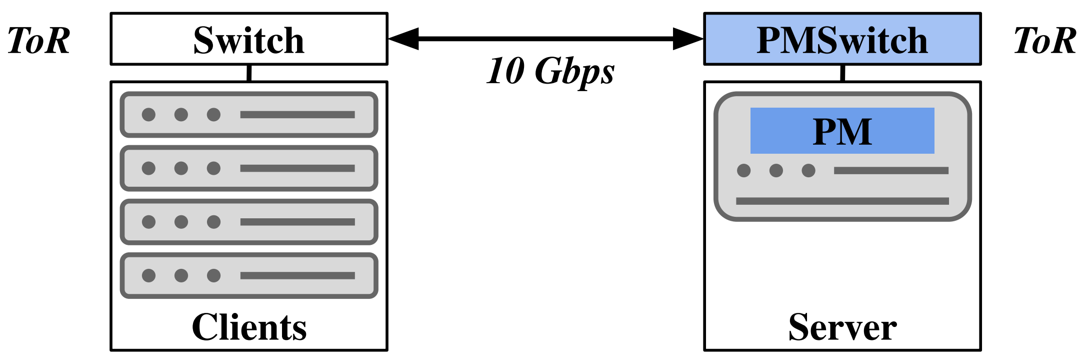
Methodology

Design points

- **PMNet-Switch**: PMNet as a bump-in-the-wire in the TOR switch of server rack
- **PMNet-NIC**: PMNet as a bump-in-the-wire in the server's NIC



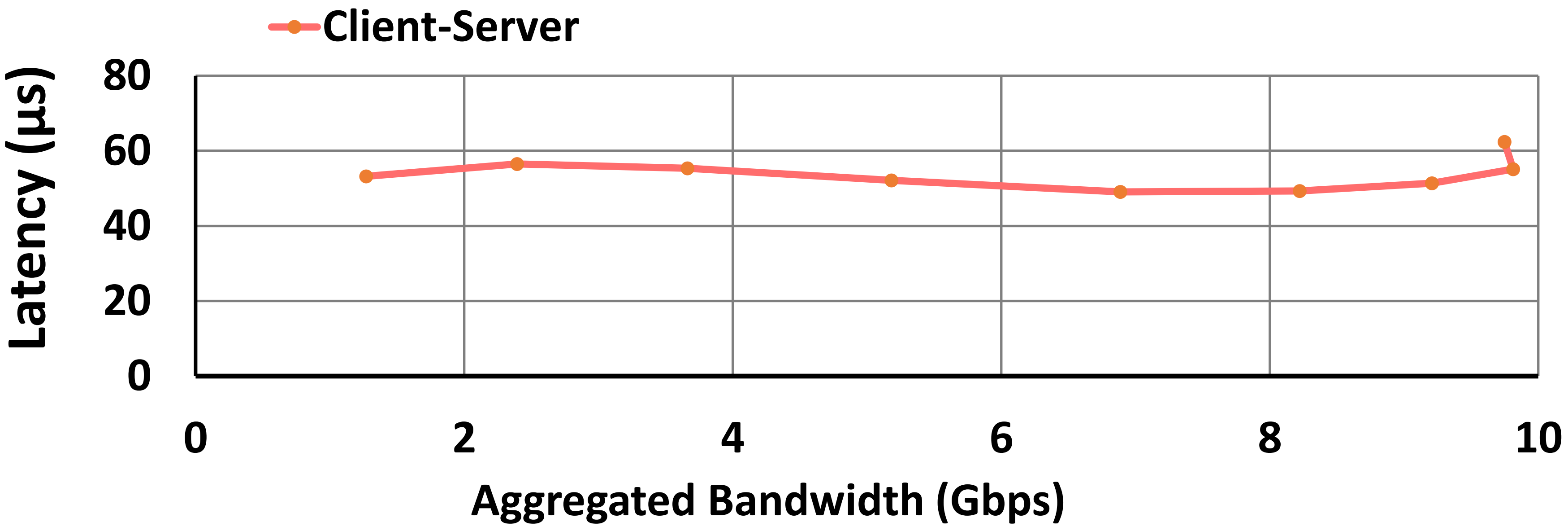
PMNet-NIC



PMNet-Switch

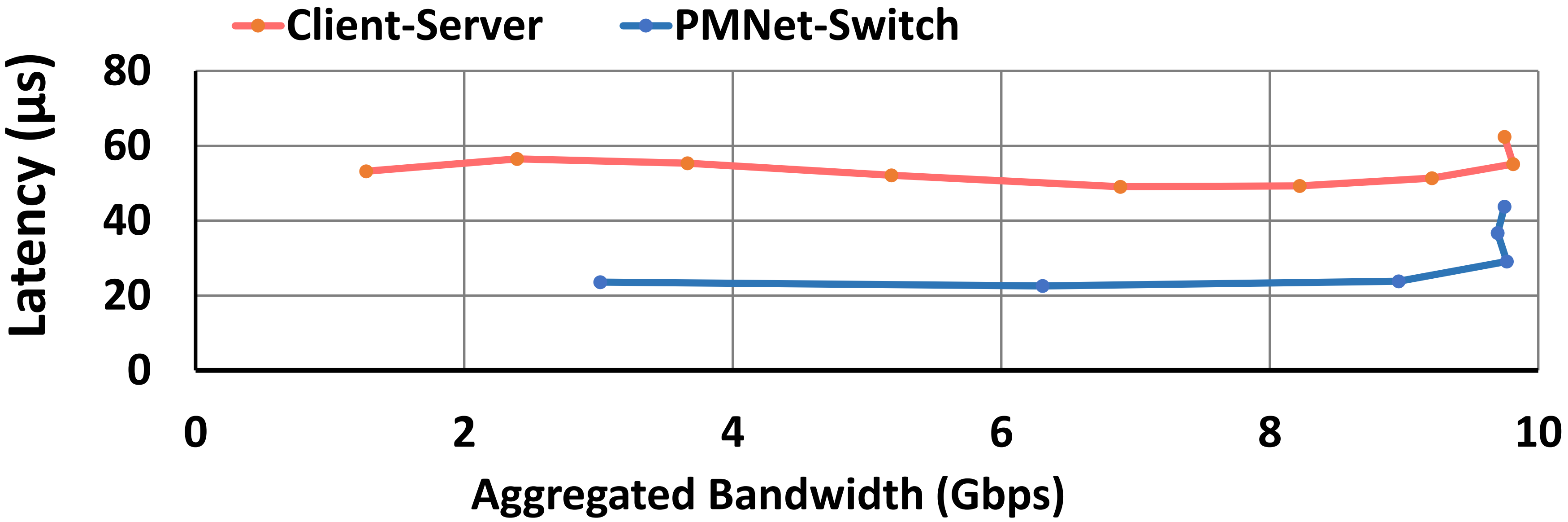
Results

Update request Bandwidth vs. Latency



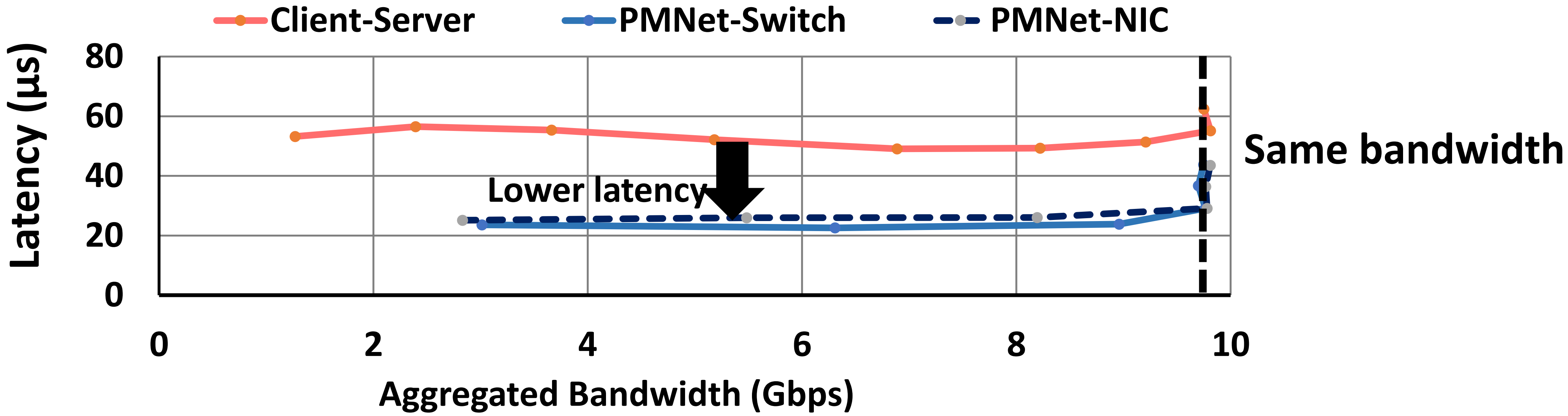
Results

Update request Bandwidth vs. Latency



Results

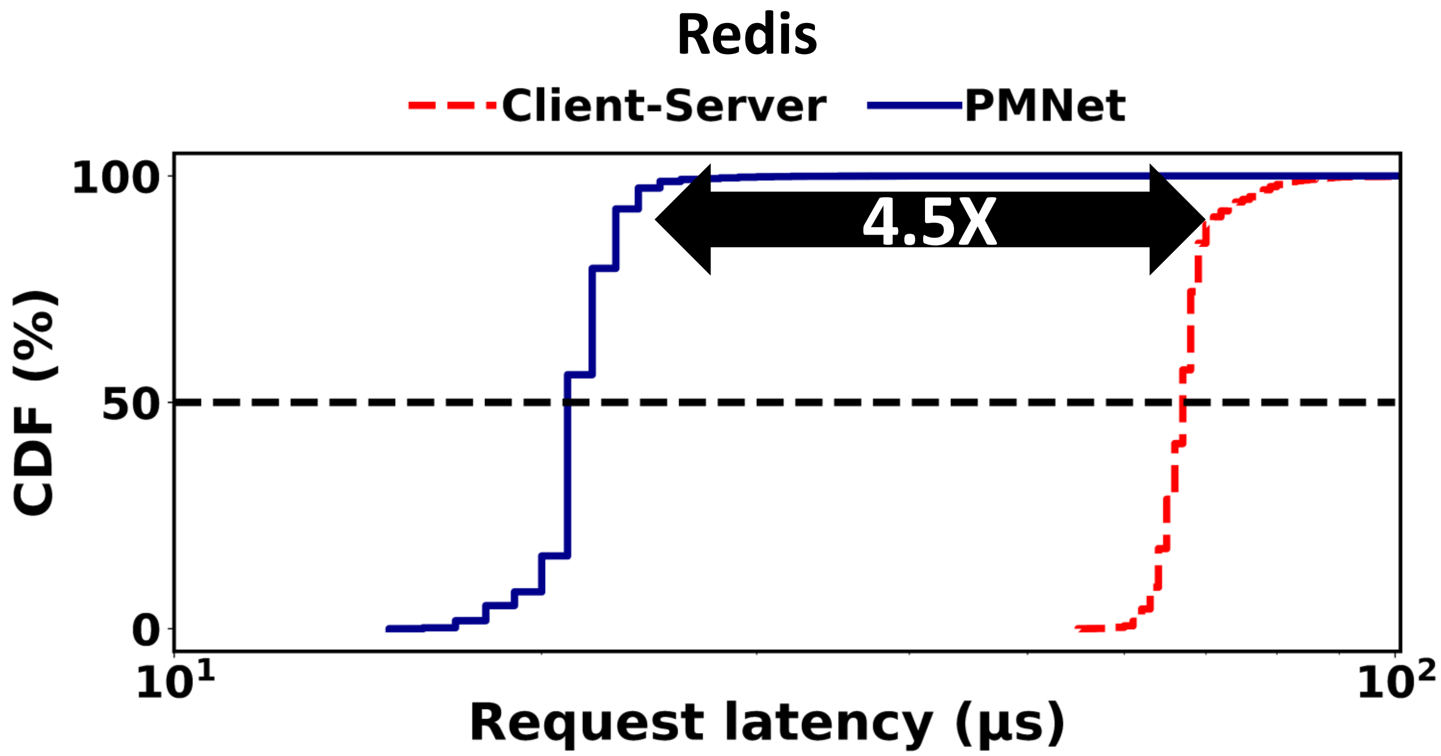
Update request Bandwidth vs. Latency



Both PMNet-Switch and PMNet-NIC provide **lower update latency** and **same bandwidth** as the baseline.

Results

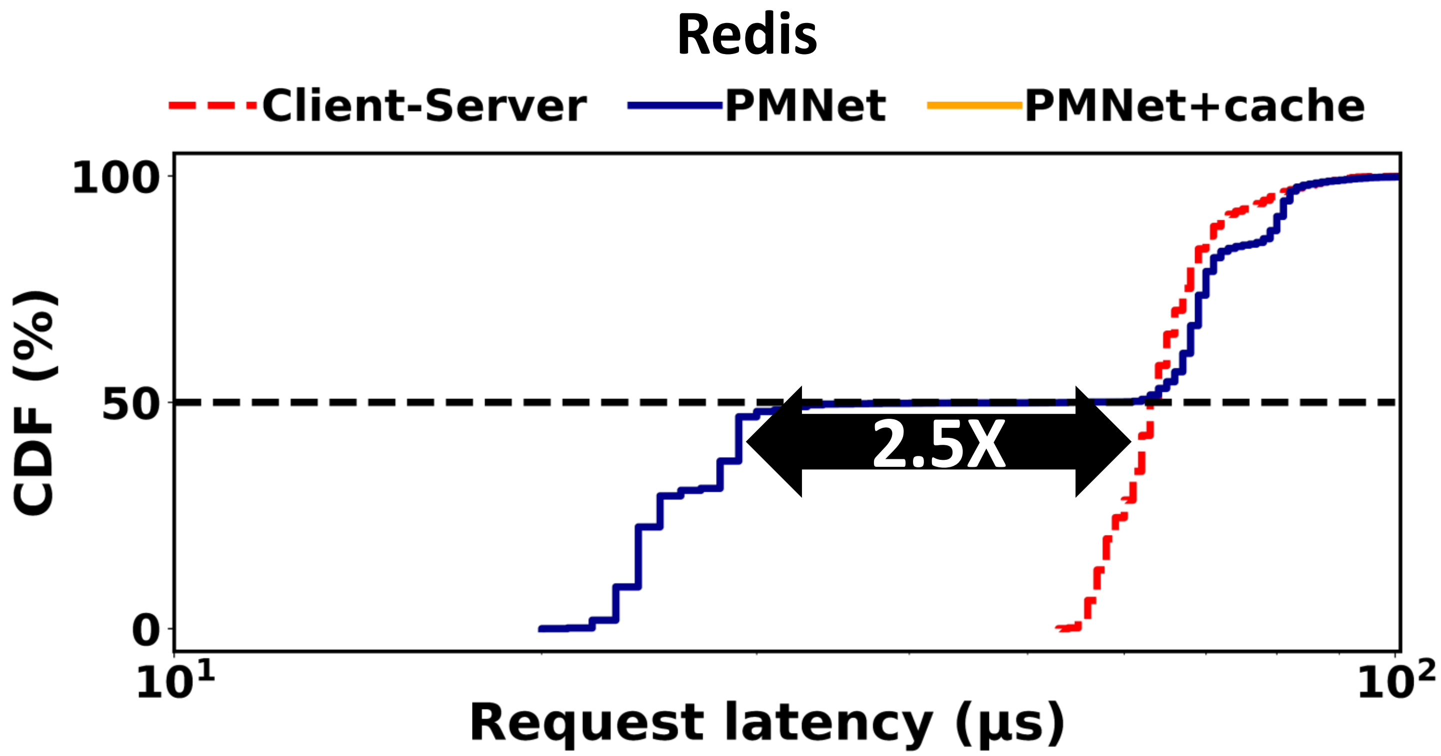
Tail latency: 100% Update requests



PMNet significantly improves 99% **tail-latency** of update requests

Results

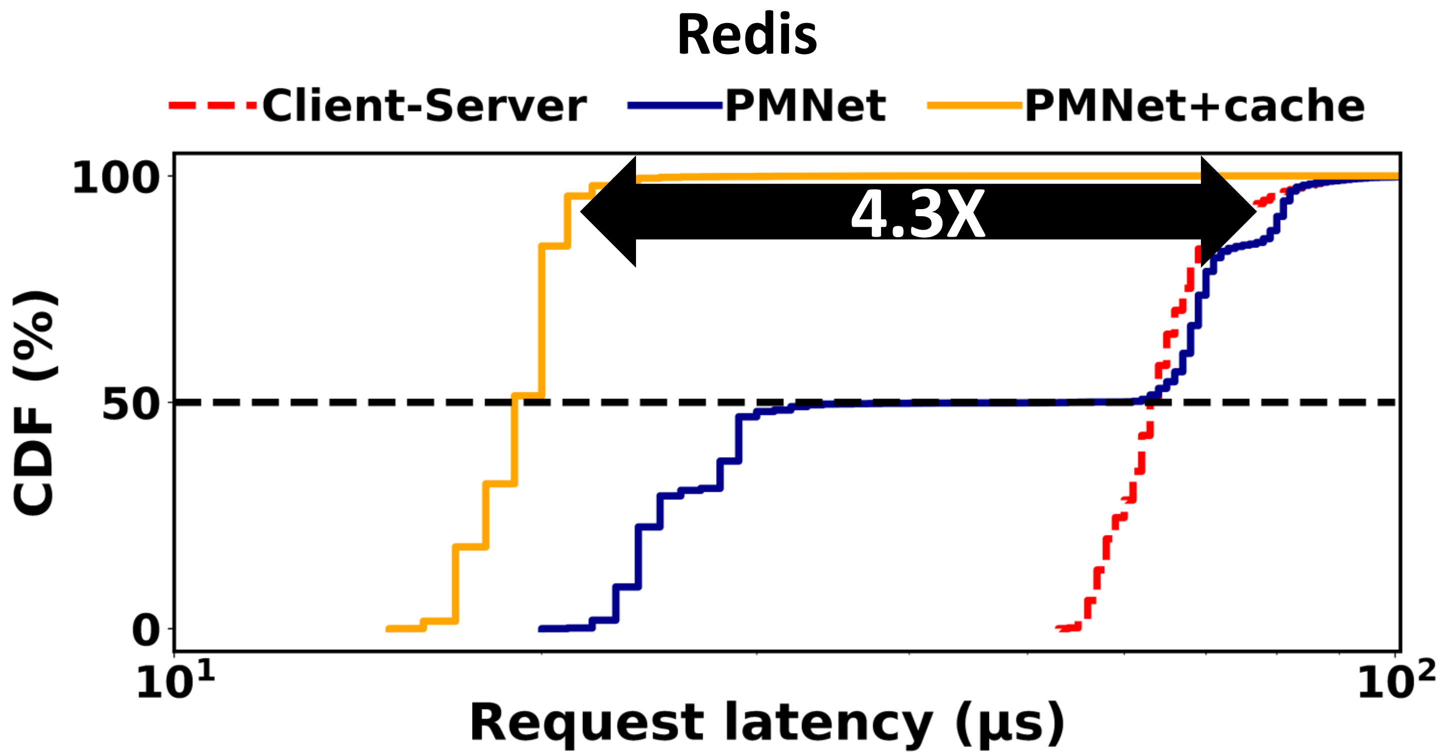
Tail latency: 50%-50% Update-read requests & Cache



Without read cache, PMNet only improves **update** requests' latency.

Results

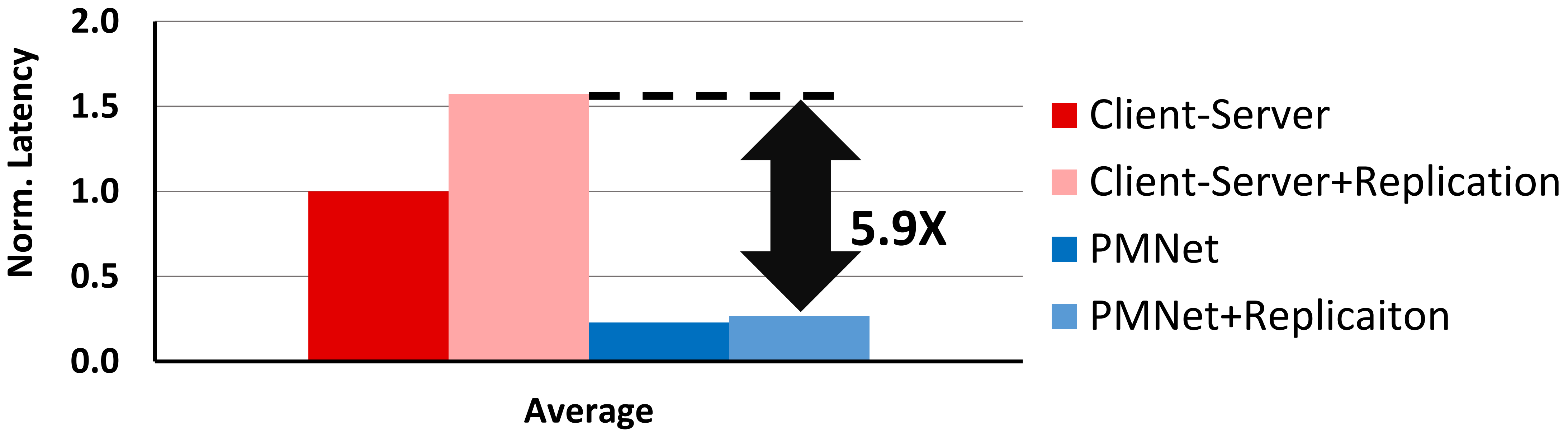
Tail latency: 50%-50% Update-read requests & Cache



With read cache, PMNet improves both **read** and **update** requests' latency.

Results

3-way Server Replication (R=3)



PMNet replication **reduces replication latency** while offering the same level of protection.

Outline

Background and Motivation

In-network Data Persistence

PMNet Design

Caching and Replication

Evaluation

Conclusion

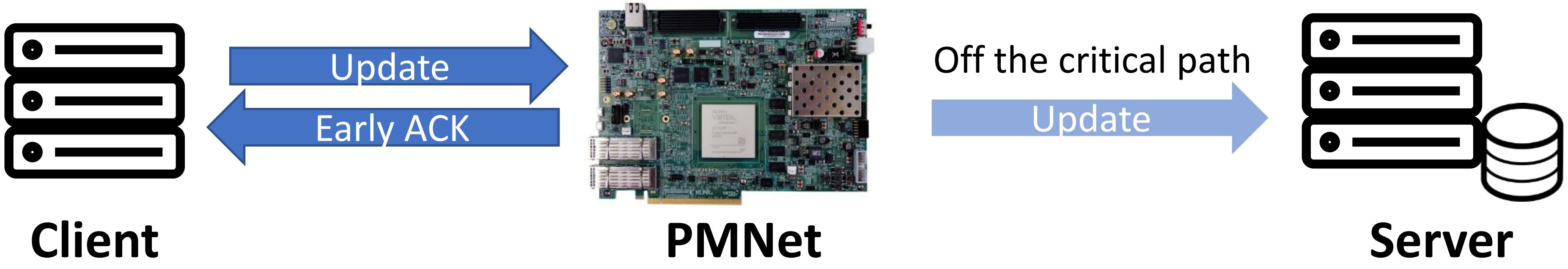
Summary

PMNet

- Logs requests in network device's persistent memory
- Recovers server using logged requests in case of a failure
- Integrates in-network data persistence with data **replication** and **caching**

Evaluation

- End-to-end FPGA implementation of PMNet-enabled NIC and switch
- Improves update throughput by 4.27x and tail latency by 3.23x over client-server baseline
- Improves **3-way replication (R=3)** latency by 5.9X on average
- Improves **50-50% read-write** latency by 3.36X with read caching on average



Artifact available at pmnet.persistentmemory.org