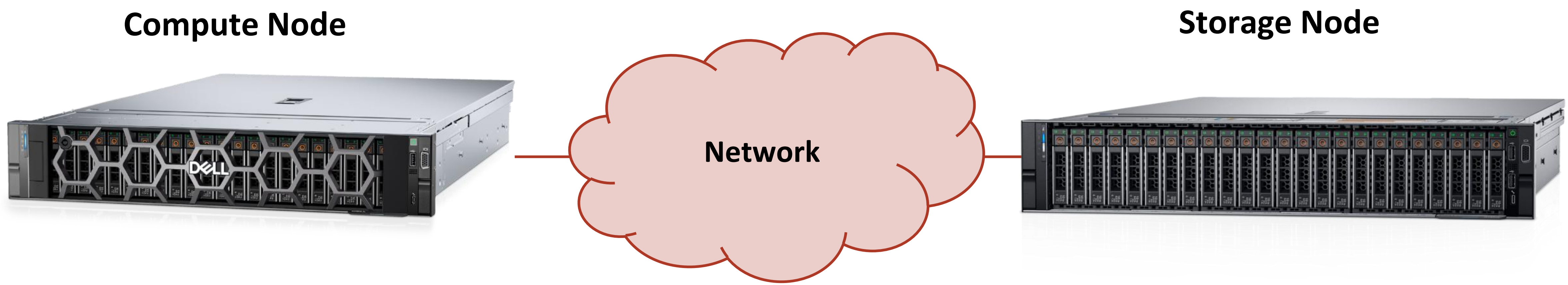
Understanding and Profiling NVMe-over-TCP Using `ntprof`

Yuyuan Kang, Ming Liu
University of Wisconsin - Madison

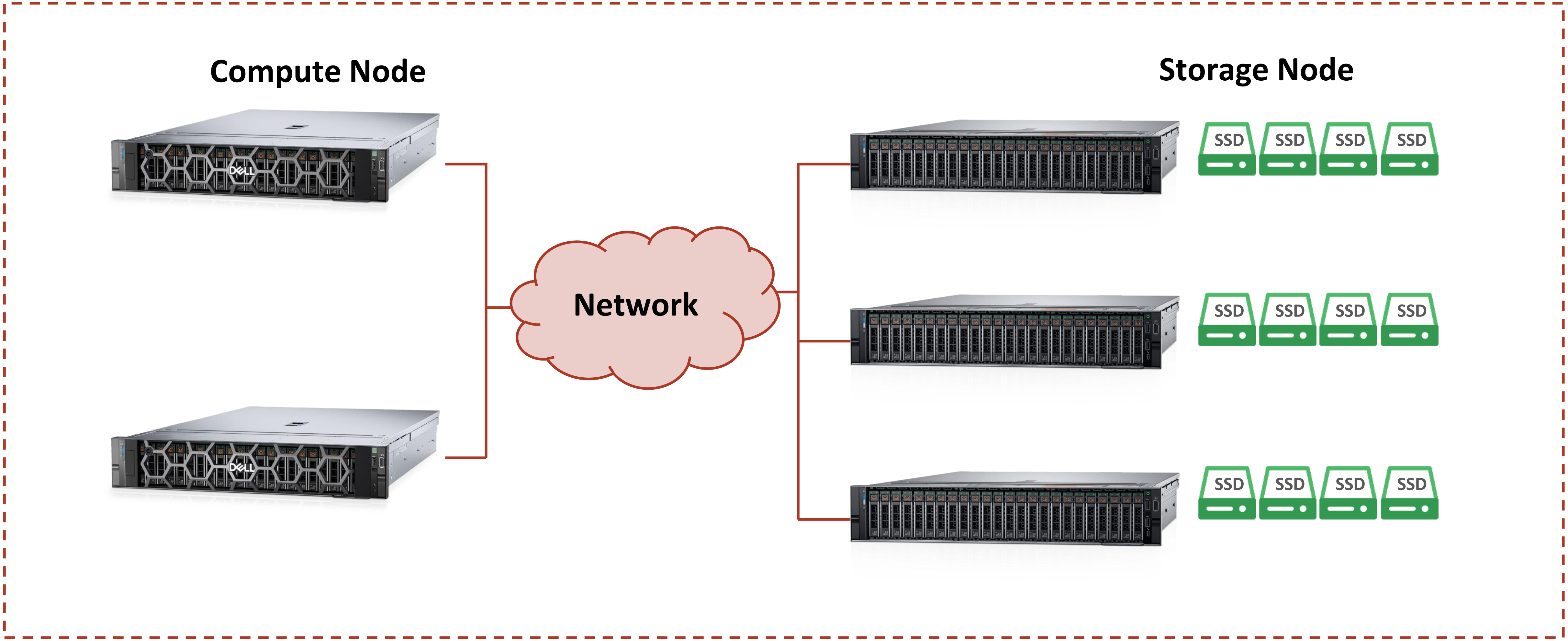
Disaggregated storage is widely deployed

- **Storage disaggregation** is a system infrastructure that separates compute from storage



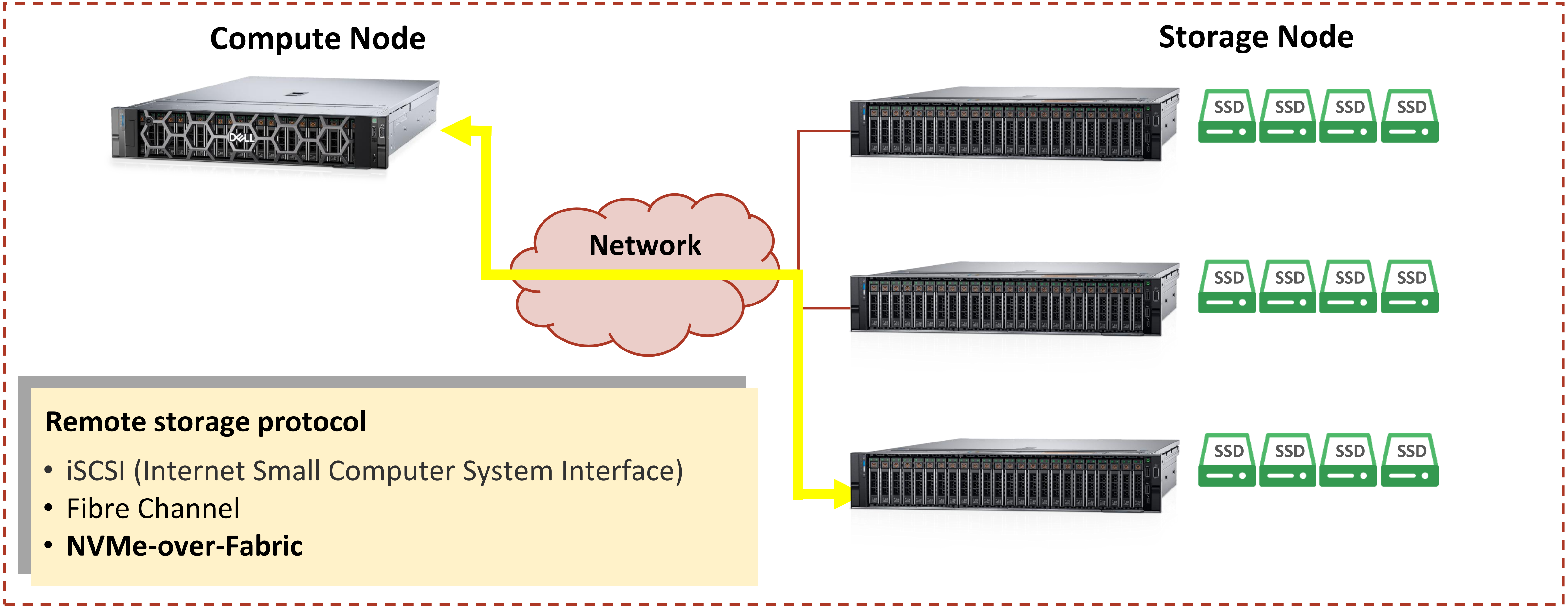
Disaggregated storage is widely deployed

- **Storage disaggregation** is a system infrastructure that separates compute from storage



NVMe/TCP enables fast storage disaggregation

- Remote protocol is essential to enable storage disaggregation

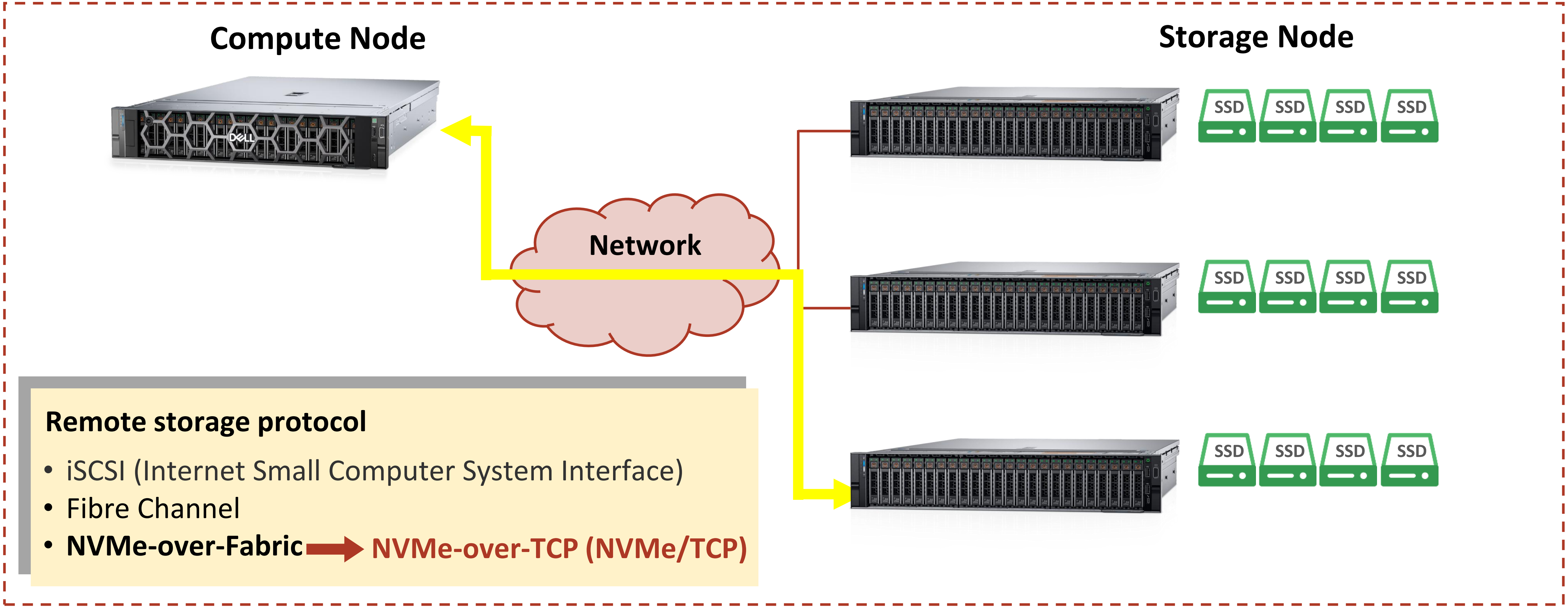


Remote storage protocol

- iSCSI (Internet Small Computer System Interface)
- Fibre Channel
- **NVMe-over-Fabric**

NVMe/TCP enables fast storage disaggregation

- Remote protocol is essential to enable storage disaggregation



Remote storage protocol

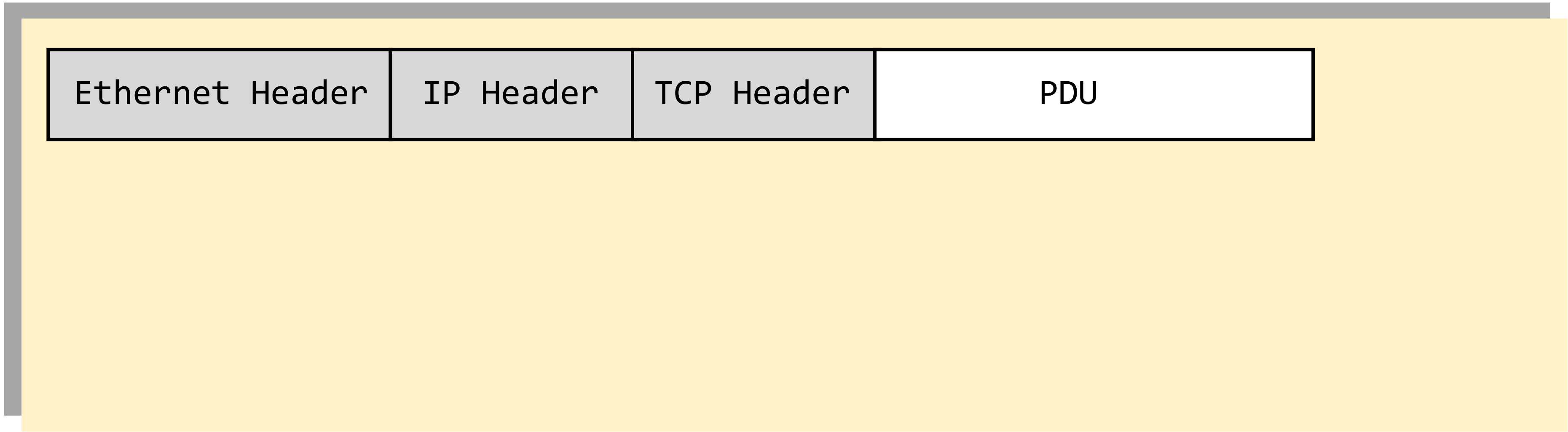
- iSCSI (Internet Small Computer System Interface)
- Fibre Channel
- **NVMe-over-Fabric** → **NVMe-over-TCP (NVMe/TCP)**

NVMe/TCP Primer

- **NVMe/TCP**: carries NVMe commands to remote NVMe subsystems via TCP/IP

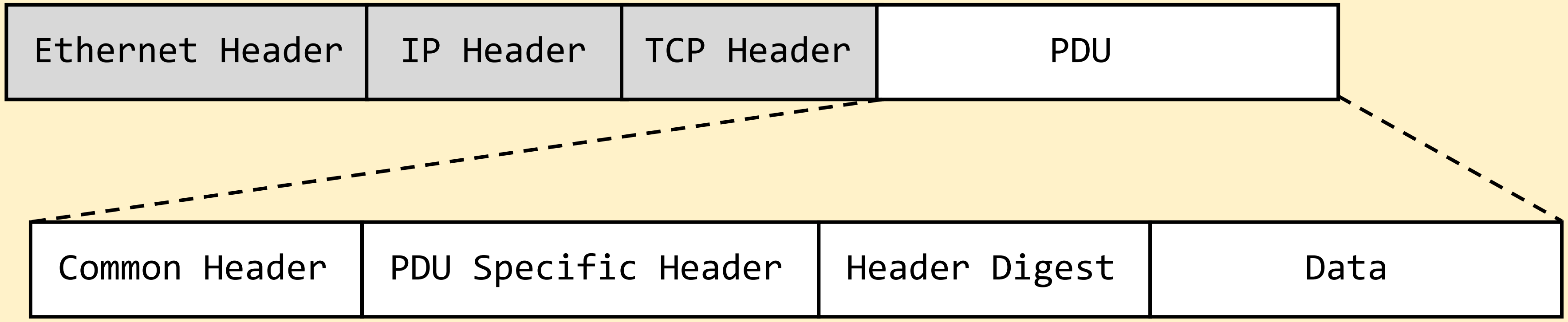
NVMe/TCP Primer

- **NVMe/TCP**: carries NVMe commands to remote NVMe subsystems via TCP/IP
- Transfer Unit: Protocol Data Unit (PDU)
  - I/O command
  - Data payload
  - Control status
- 5 Types of PDU
  - CapsuleCmd
  - CapsuleResp
  - C2HData
  - H2CData
  - R2T



NVMe/TCP Primer

- **NVMe/TCP**: carries NVMe commands to remote NVMe subsystems via TCP/IP
- Transfer Unit: Protocol Data Unit (PDU)
  - I/O command
  - Data payload
  - Control status
- 5 Types of PDU
  - CapsuleCmd
  - CapsuleResp
  - C2HData
  - H2CData
  - R2T



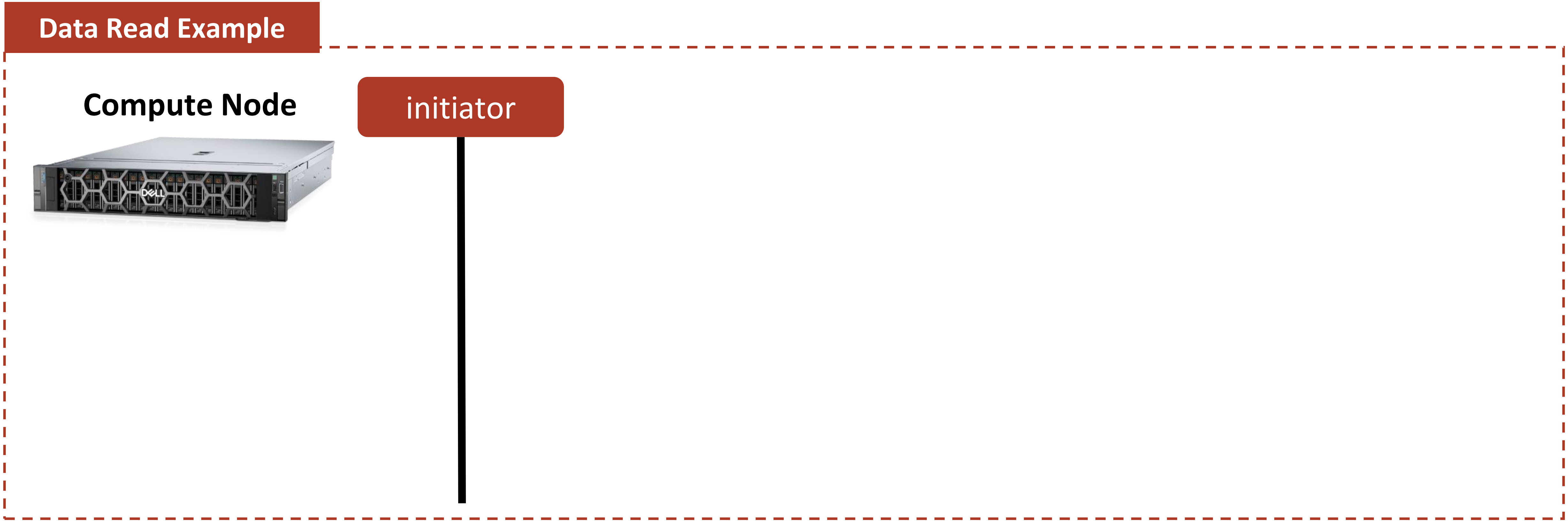
Ethernet Header	IP Header	TCP Header	PDU

Common Header	PDU Specific Header	Header Digest	Data

An example: C2H Data PDU

Read/Write I/O over **NVMe/TCP**

- **Initiator:** send NVMe commands to remote storage
- **Target:** receive NVMe commands, and reading/writing to the NVMe drive
- **NVMe/TCP Session:** a bidirectional communication channel established between two sides



Data Read Example

Read/Write I/O over **NVMe/TCP**

- **Initiator:** send NVMe commands to remote storage
- **Target:** receive NVMe commands, and reading/writing to the NVMe drive
- **NVMe/TCP Session:** a bidirectional communication channel established between two sides

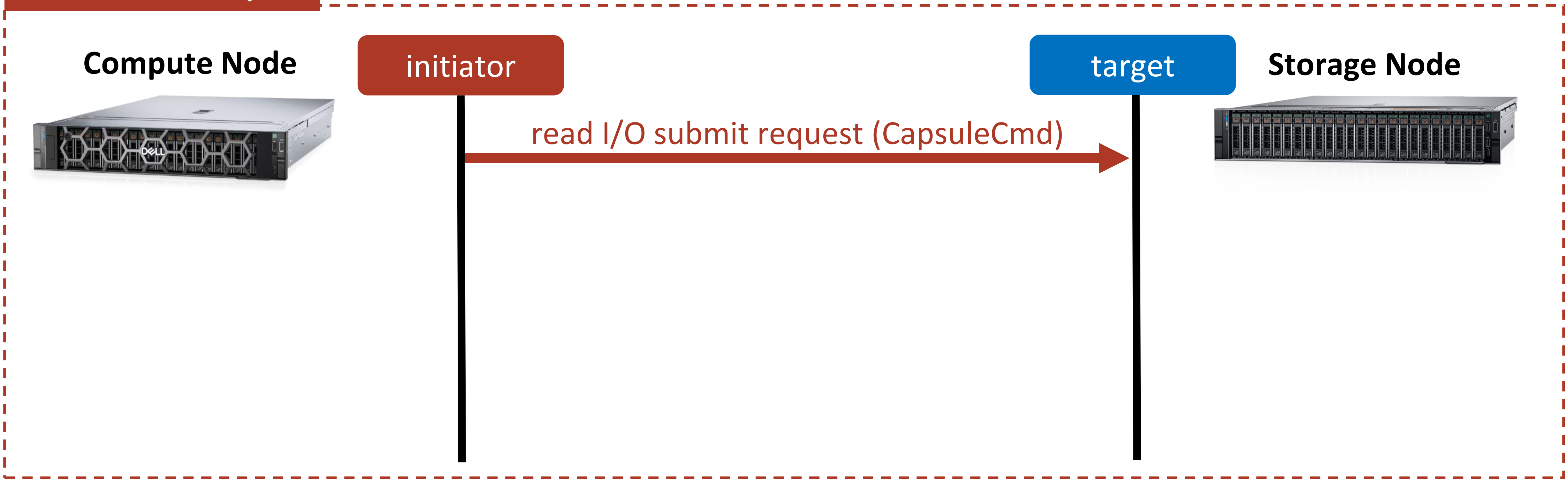
Data Read Example

Compute Node
initiator
target
Storage Node

Read/Write I/O over **NVMe/TCP**

- **Initiator:** send NVMe commands to remote storage
- **Target:** receive NVMe commands, and reading/writing to the NVMe drive
- **NVMe/TCP Session:** a bidirectional communication channel established between two sides

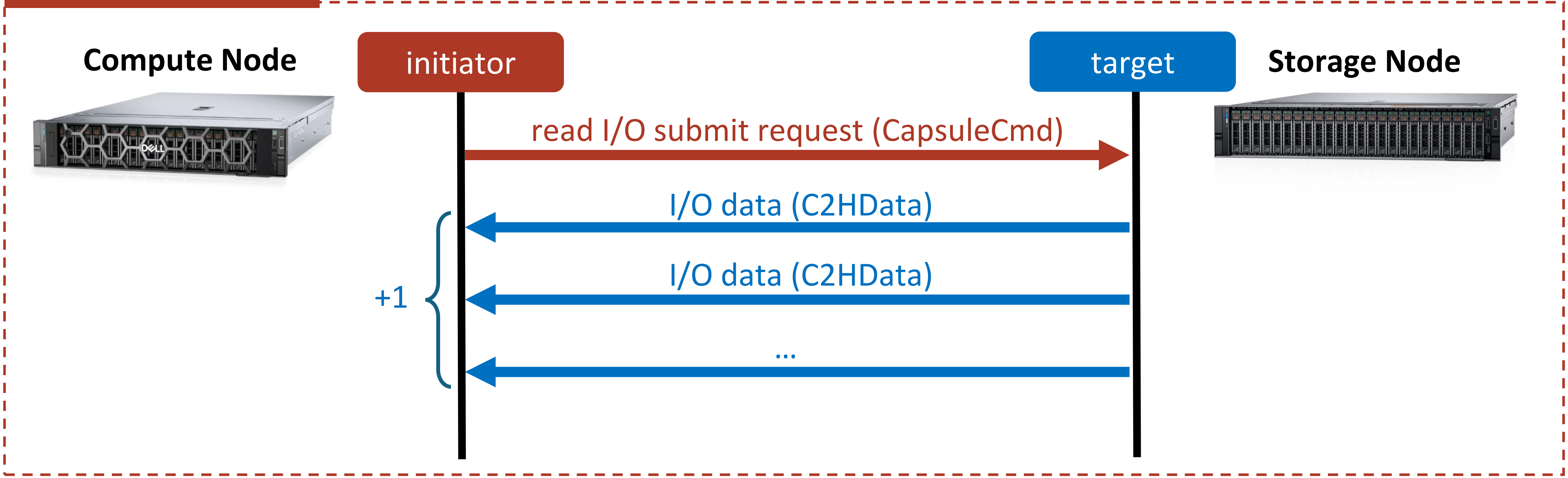
Data Read Example



Read/Write I/O over **NVMe/TCP**

- **Initiator:** send NVMe commands to remote storage
- **Target:** receive NVMe commands, and reading/writing to the NVMe drive
- **NVMe/TCP Session:** a bidirectional communication channel established between two sides

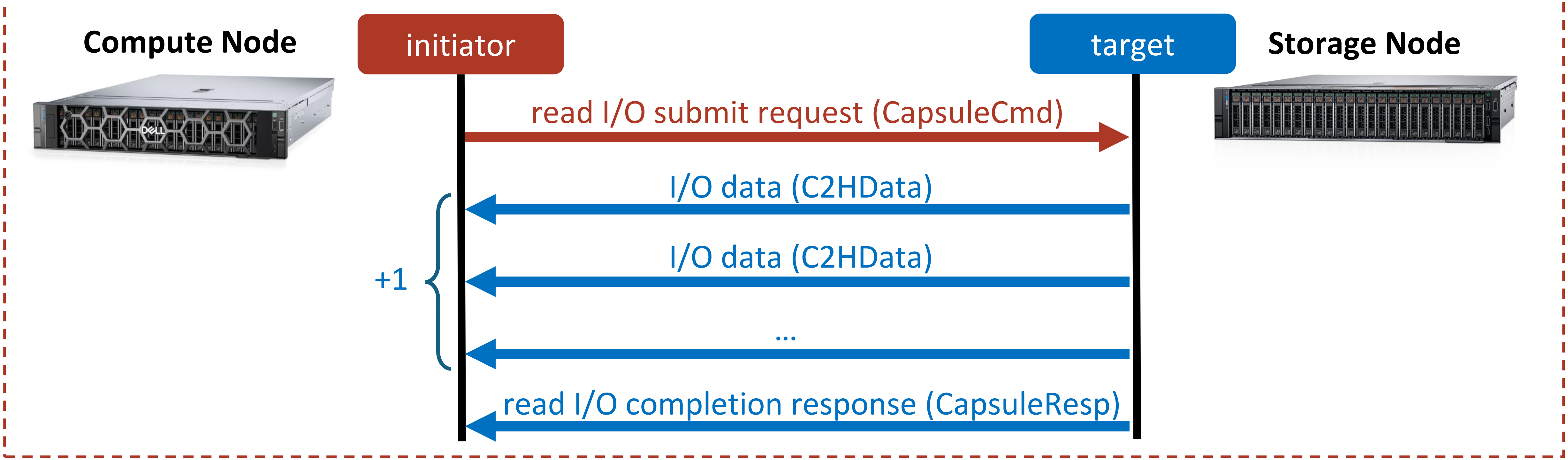
Data Read Example



Read/Write I/O over **NVMe/TCP**

- **Initiator:** send NVMe commands to remote storage
- **Target:** receive NVMe commands, and reading/writing to the NVMe drive
- **NVMe/TCP Session:** a bidirectional communication channel established between two sides

Data Read Example


Compute Node
initiator
target
Storage Node
read I/O submit request (CapsuleCmd)
I/O data (C2HData)
I/O data (C2HData)
+1
…
read I/O completion response (CapsuleResp)

Goal: Understanding and profiling NVMe/TCP

Outline

1 Challenges

2 Existing Solutions

3 ntprof Design and Implementation

4 Evaluation

5 Summary

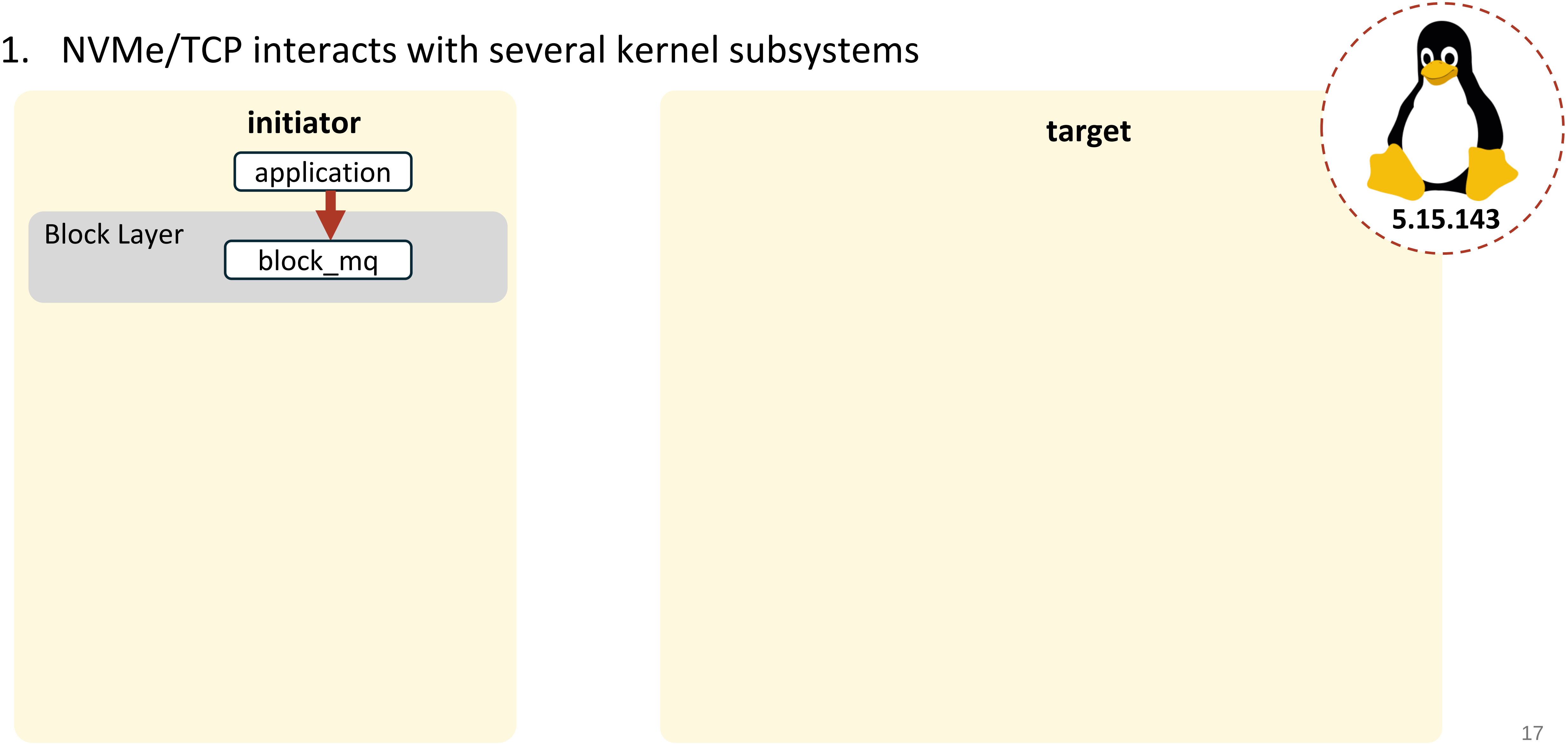
Challenge #1: Complicated Execution Path

1. NVMe/TCP interacts with several kernel subsystems

5.15.143

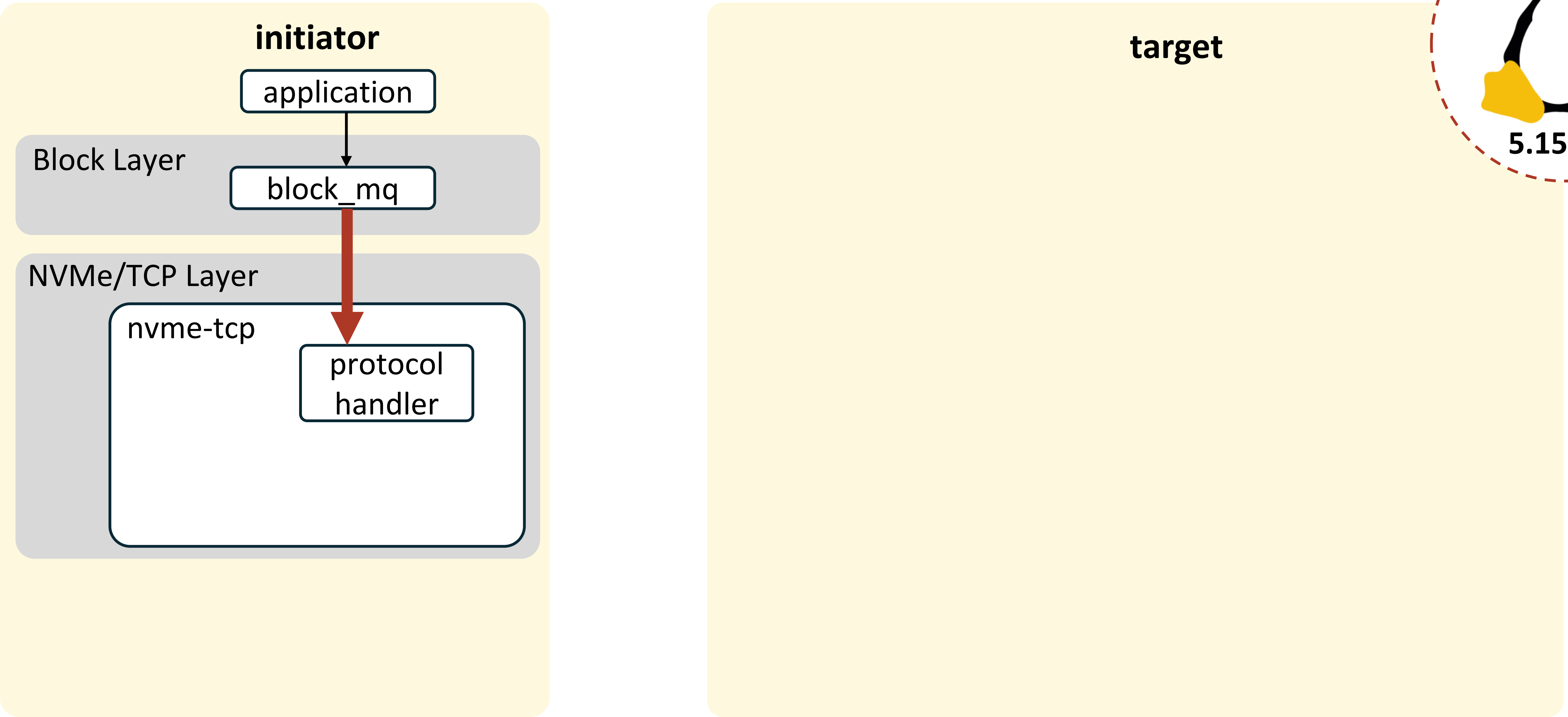
Challenge #1: Complicated Execution Path

1. NVMe/TCP interacts with several kernel subsystems



Challenge #1: Complicated Execution Path

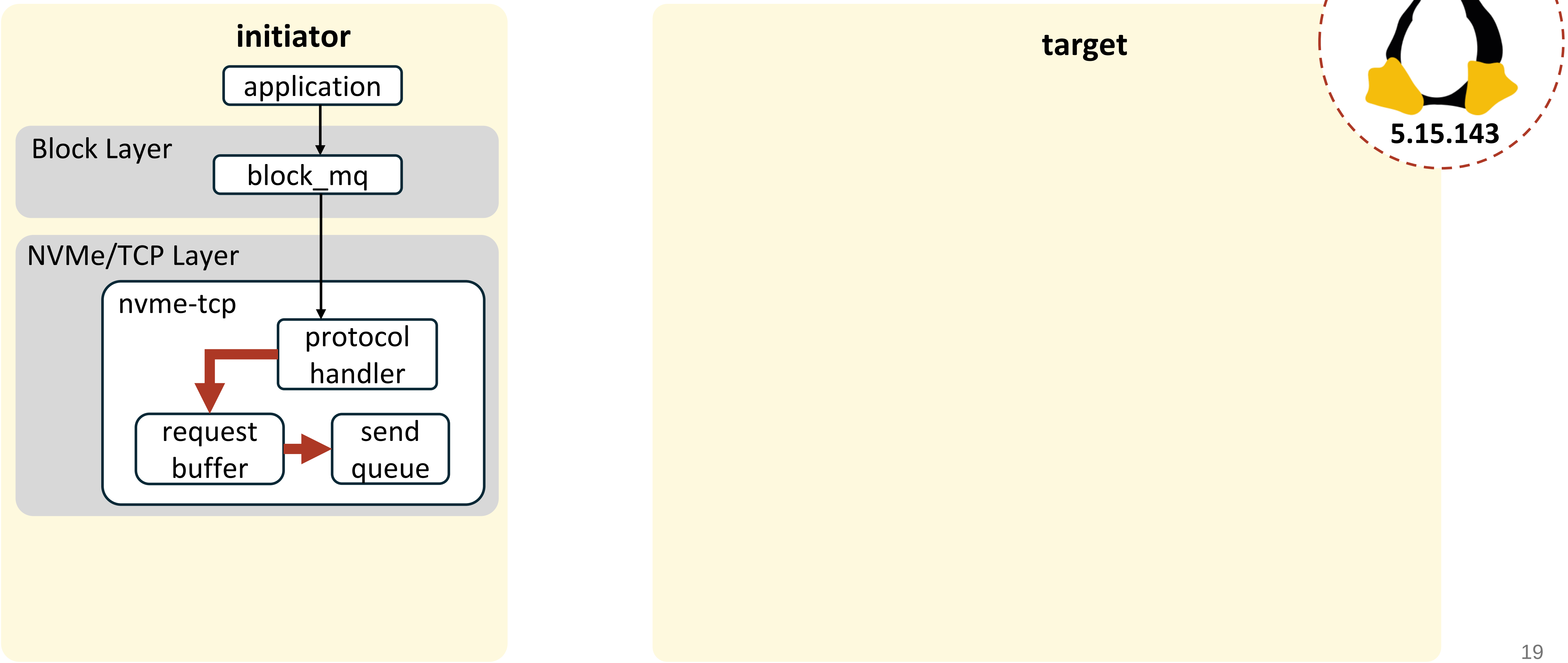
1. NVMe/TCP interacts with several kernel subsystems



5.15.143

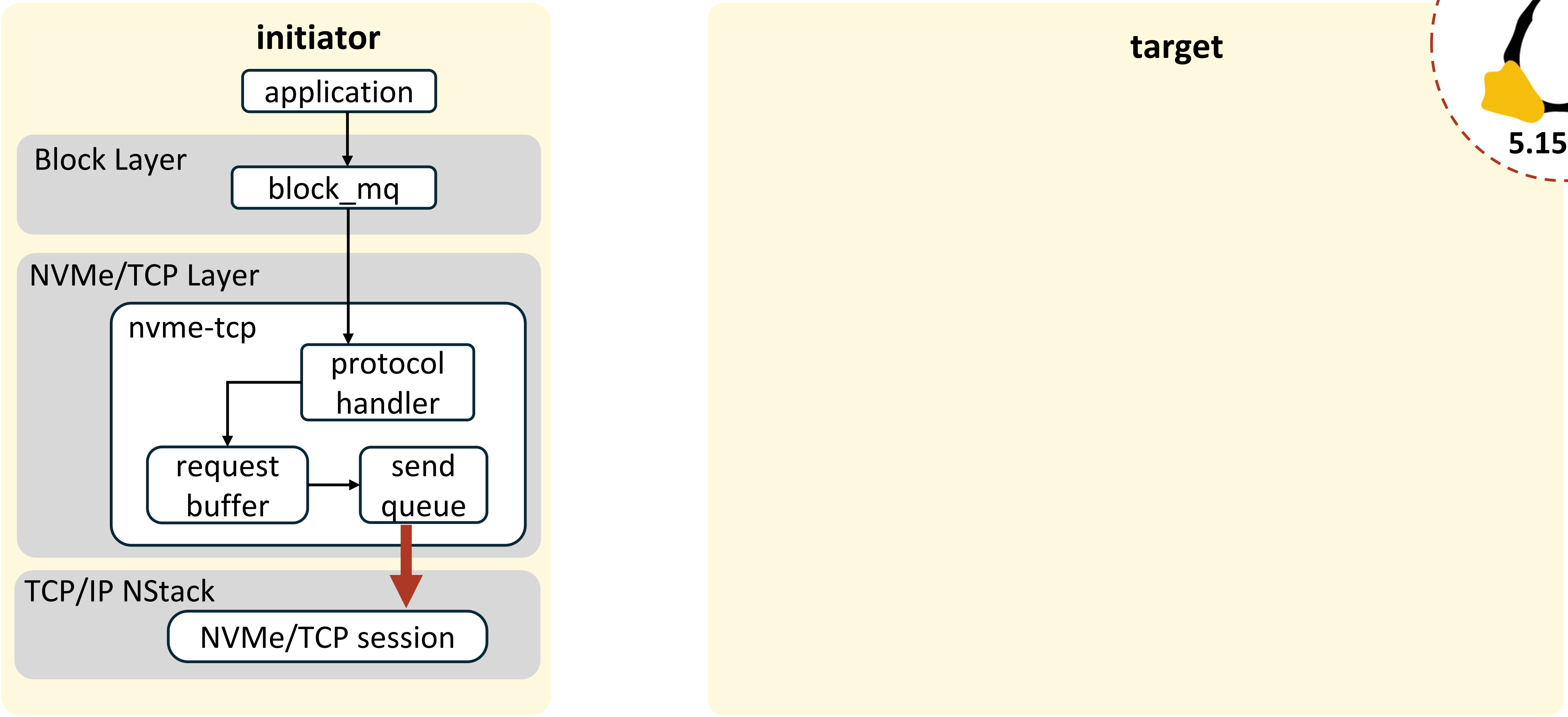
Challenge #1: Complicated Execution Path

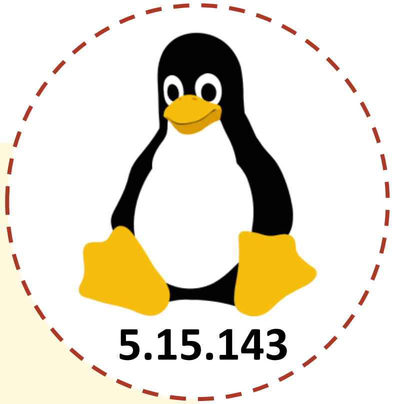
1. NVMe/TCP interacts with several kernel subsystems



Challenge #1: Complicated Execution Path

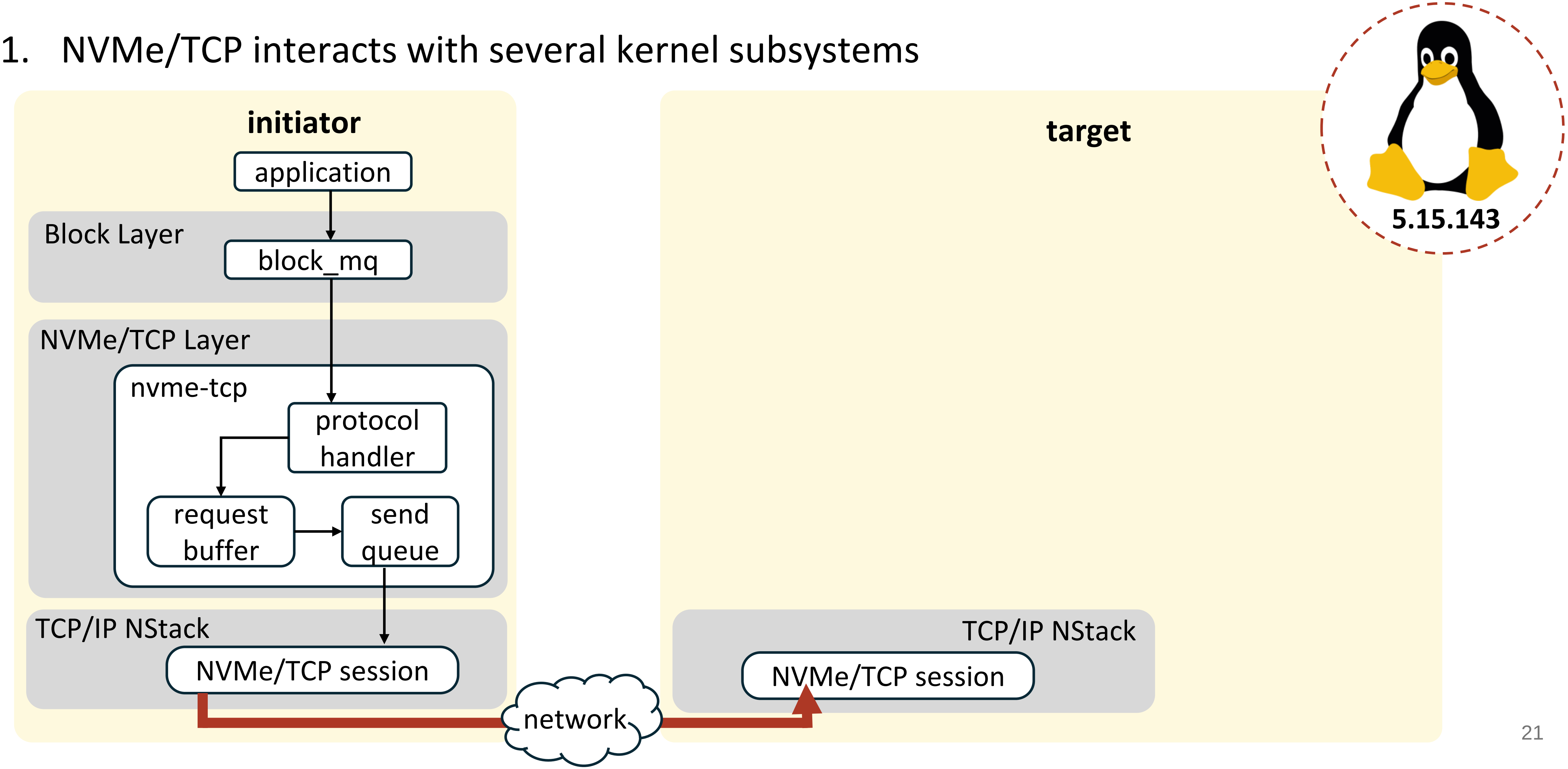
1. NVMe/TCP interacts with several kernel subsystems





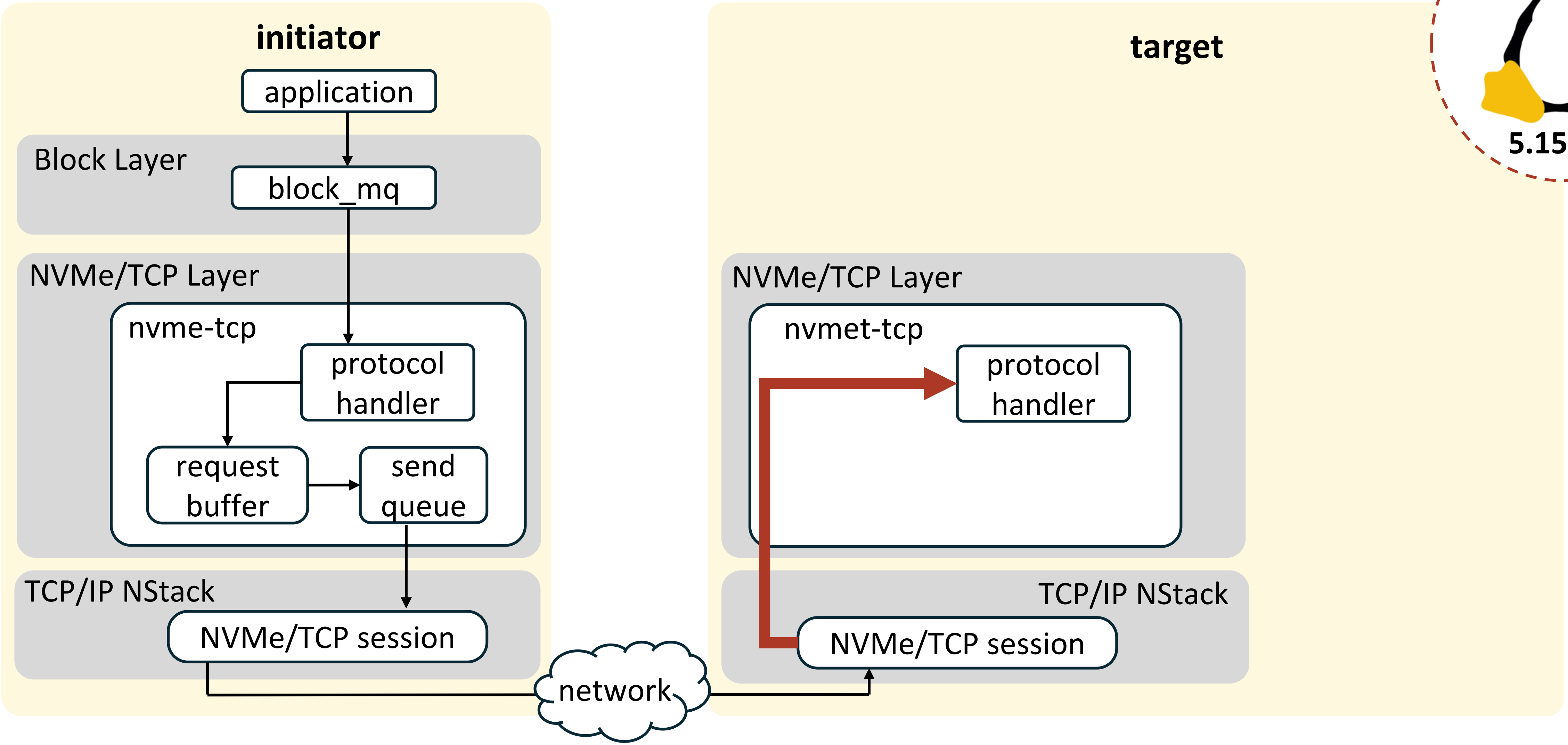
Challenge #1: Complicated Execution Path

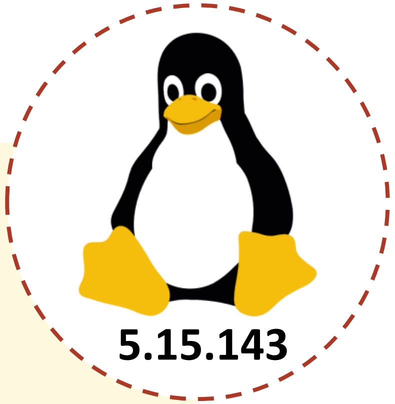
1. NVMe/TCP interacts with several kernel subsystems



Challenge #1: Complicated Execution Path

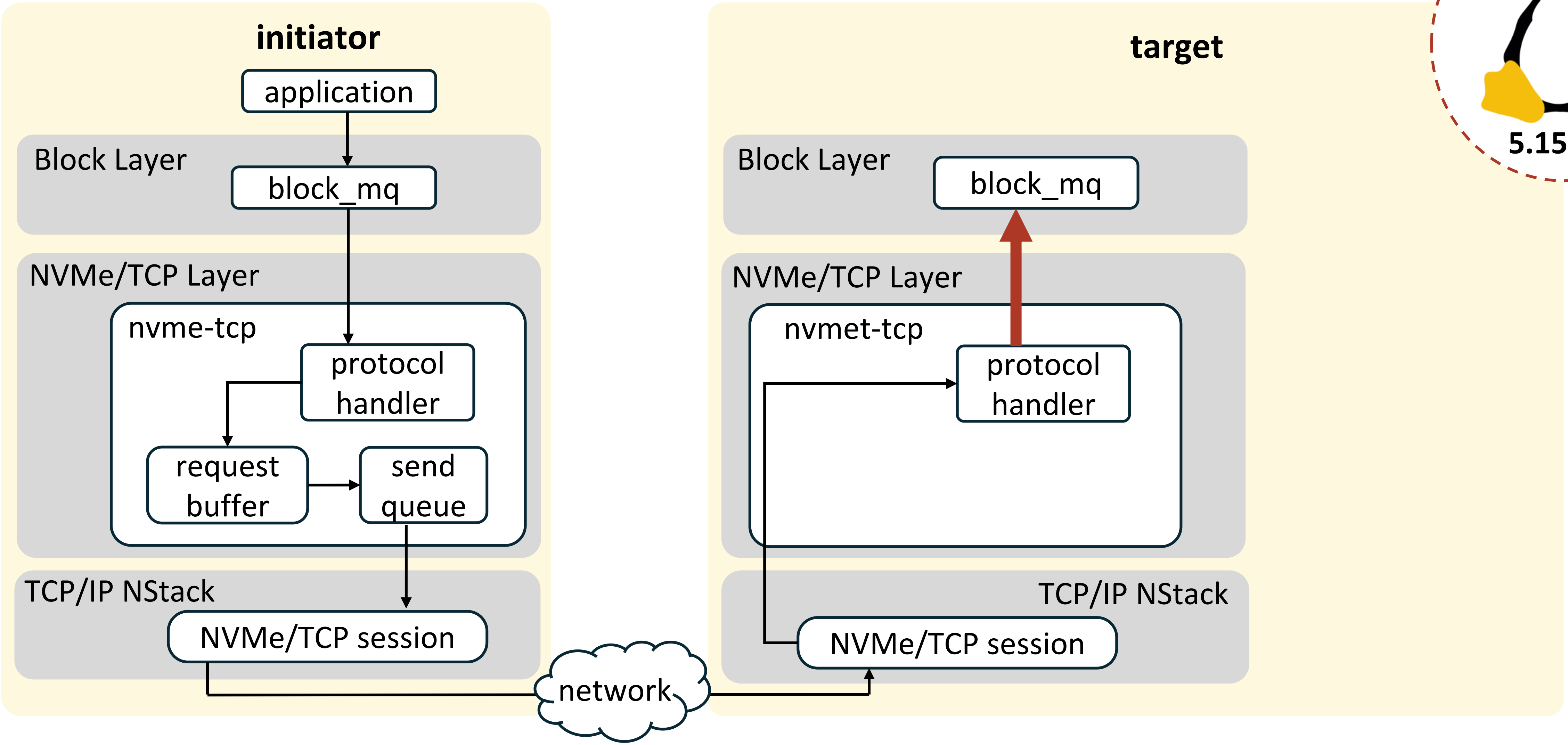
1. NVMe/TCP interacts with several kernel subsystems

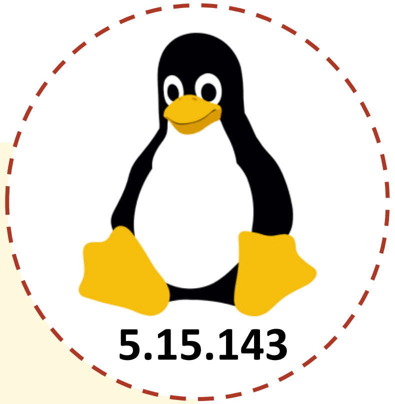




Challenge #1: Complicated Execution Path

1. NVMe/TCP interacts with several kernel subsystems

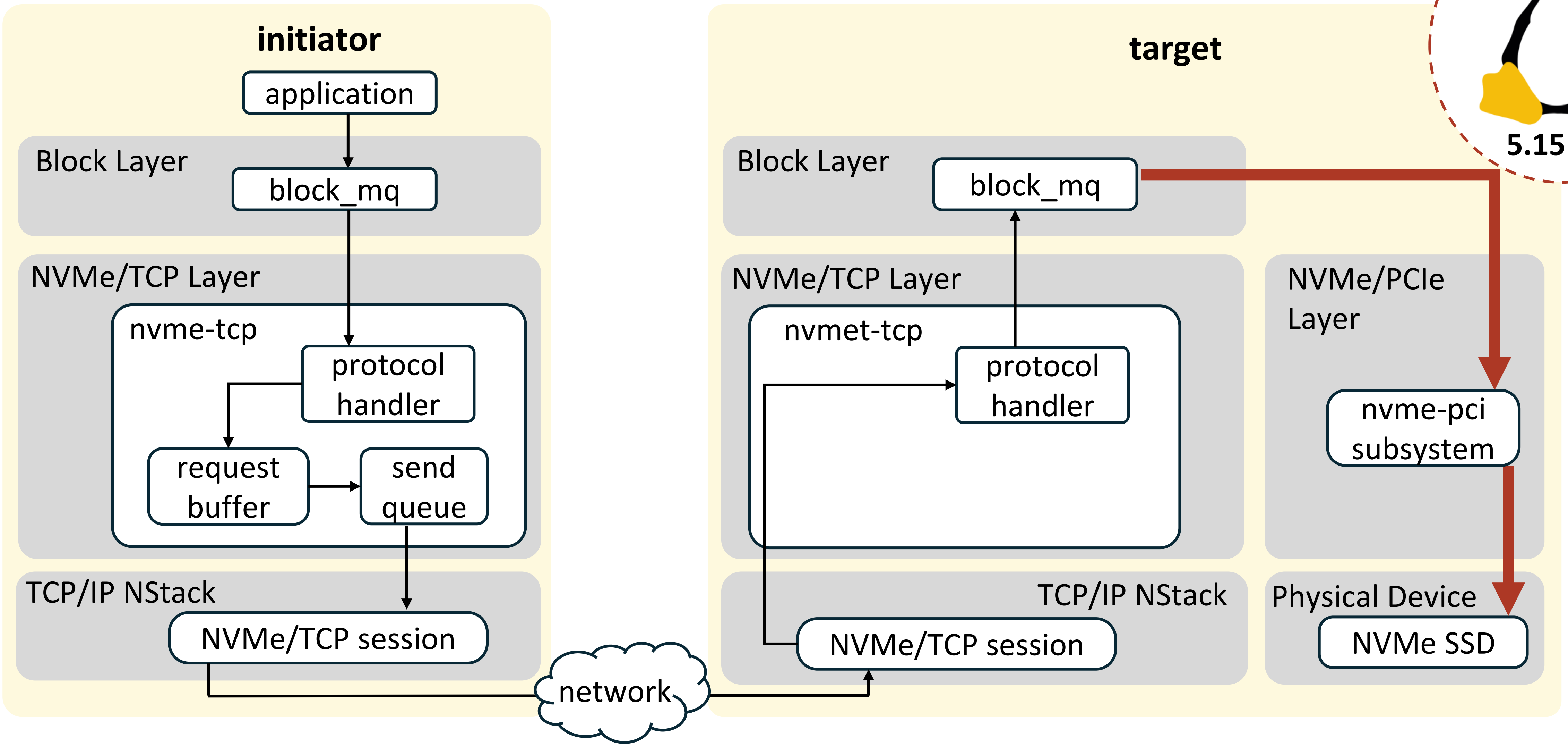



5.15.143

Challenge #1: Complicated Execution Path

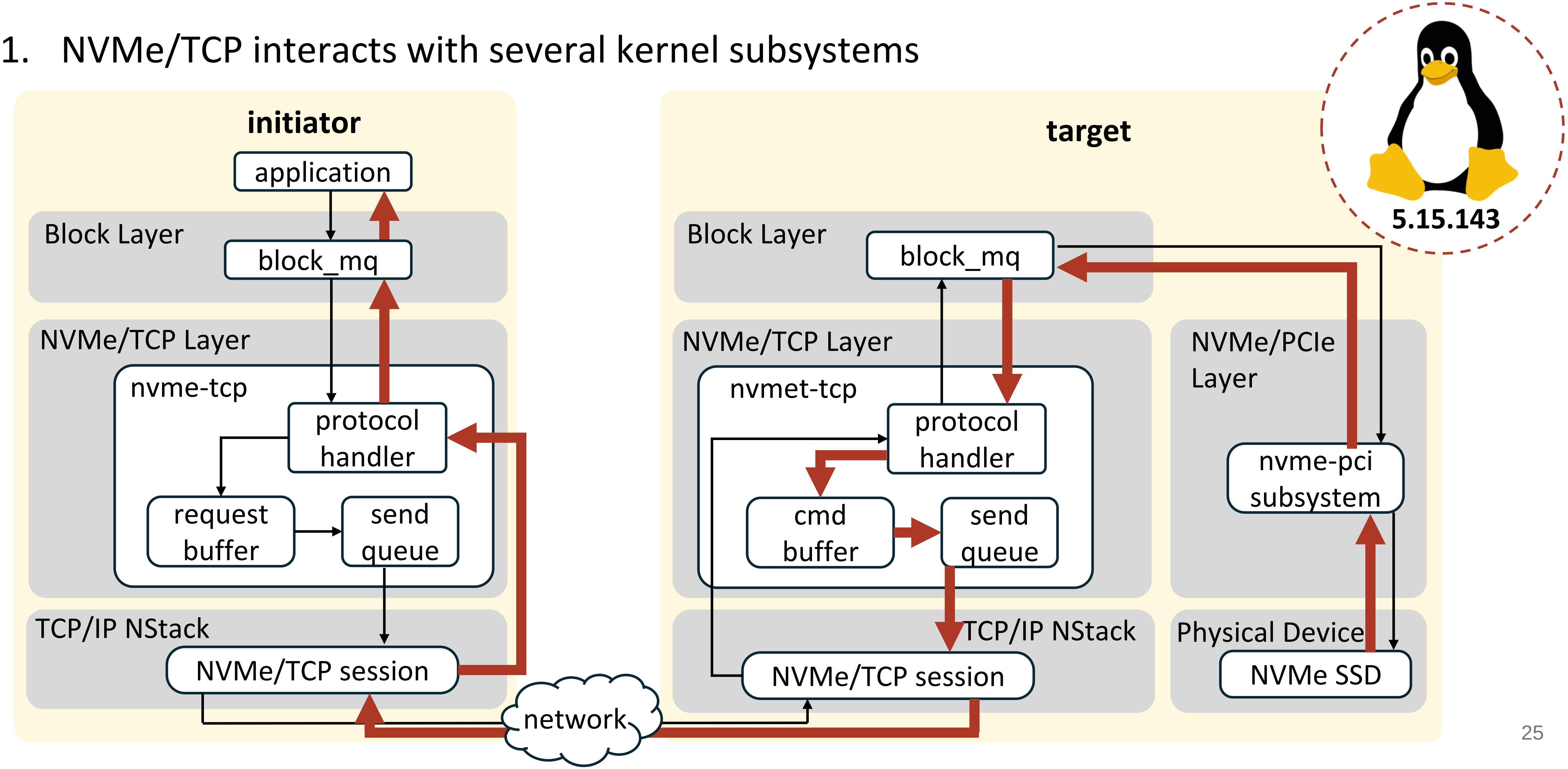
1. NVMe/TCP interacts with several kernel subsystems

5.15.143



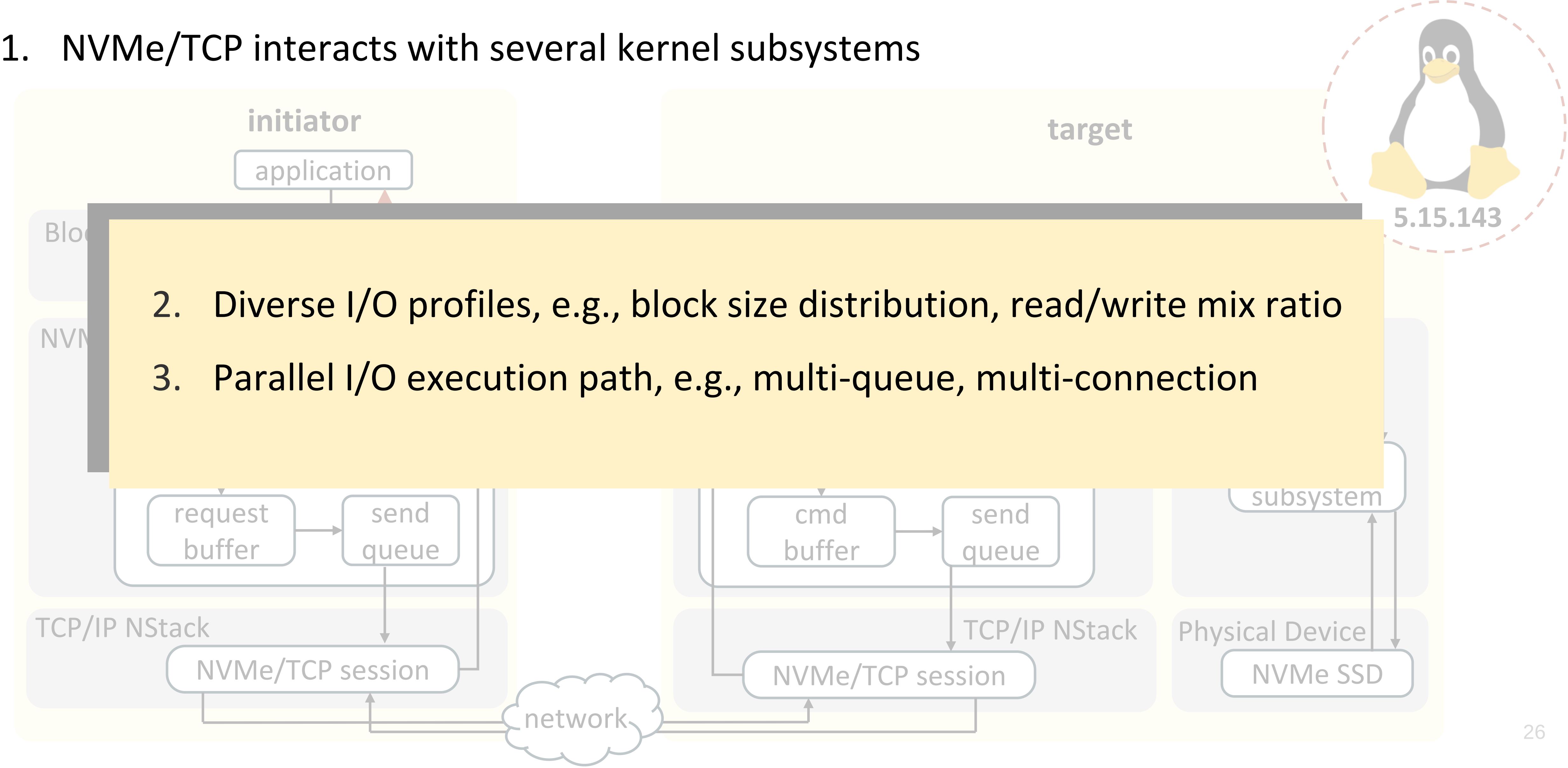
Challenge #1: Complicated Execution Path

1. NVMe/TCP interacts with several kernel subsystems



Other Challenges

1. NVMe/TCP interacts with several kernel subsystems



2. Diverse I/O profiles, e.g., block size distribution, read/write mix ratio
3. Parallel I/O execution path, e.g., multi-queue, multi-connection

Outline

1 Challenges

2 Existing Solutions

3 ntprof Design and Implementation

4 Evaluation

5 Summary

Existing Solutions

- Solution: manually profile from different layers and synthesize the results
- As an example,
  - Application-provided microbenchmarks , e.g., RocksDB's `db_bench`
  - System/language tools, e.g., `gprof`, `JProfiler`, `cProfiler`
  - Low-level infrastructure utilities, e.g., `Perf` , `iperf3`, `qperf`

Existing Solutions

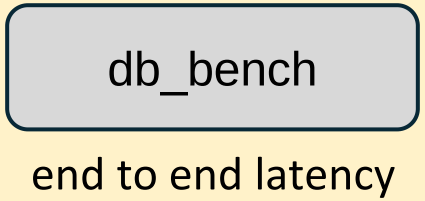
- Solution: manually profile from different layers and synthesize the results
- As an example,
  - Application-provided microbenchmarks , e.g., RocksDB’s db_bench
  - System/language tools, e.g., gprof, JProfiler, cProfiler
  - Low-level infrastructure utilities, e.g., Perf , iperf3, qperf

Why is RocksDB running slow?

Existing Solutions

- Solution: manually profile from different layers and synthesize the results
- As an example,
  - Application-provided microbenchmarks , e.g., RocksDB’s `db_bench`
  - System/language tools, e.g., `gprof`, `JProfiler`, `cProfiler`
  - Low-level infrastructure utilities, e.g., `Perf` , `iperf3`, `qperf`

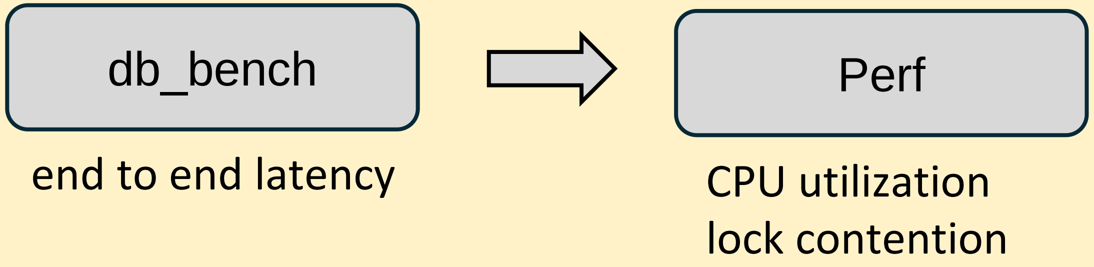
Why is RocksDB running slow?



Existing Solutions

- Solution: manually profile from different layers and synthesize the results
- As an example,
  - Application-provided microbenchmarks , e.g., RocksDB’s `db_bench`
  - System/language tools, e.g., `gprof`, `JProfiler`, `cProfiler`
  - Low-level infrastructure utilities, e.g., `Perf` , `iperf3`, `qperf`

Why is RocksDB running slow?



db_bench

end to end latency

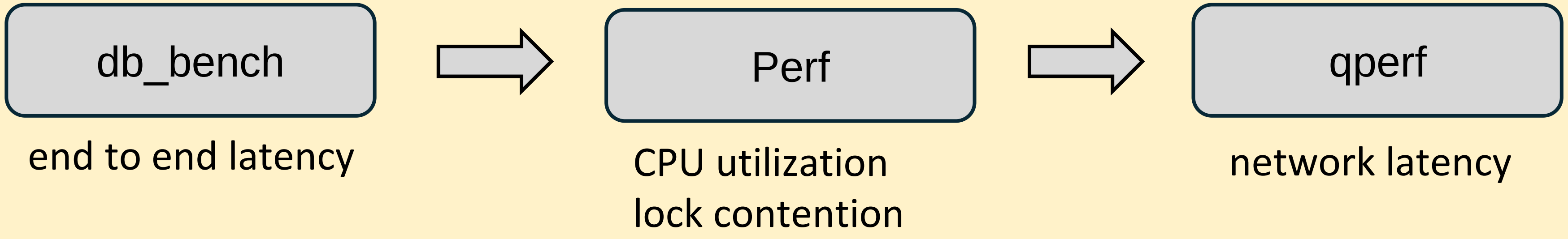
Perf

CPU utilization
lock contention

Existing Solutions

- Solution: manually profile from different layers and synthesize the results
- As an example,
  - Application-provided microbenchmarks , e.g., RocksDB's `db_bench`
  - System/language tools, e.g., `gprof`, `JProfiler`, `cProfiler`
  - Low-level infrastructure utilities, e.g., `Perf` , `iperf3`, `qperf`

Why is RocksDB running slow?



db_bench → Perf → qperf

db_bench: end to end latency

Perf: CPU utilization
lock contention

qperf: network latency

Existing Solutions

- Solution: manually profile from different layers and synthesize the results
- As an example,
  - Application-provided microbenchmarks , e.g., RocksDB’s `db_bench`
  - System/language tools, e.g., `gprof`, `JProfiler`, `cProfiler`
  - Low-level infrastructure utilities, e.g., `Perf` , `iperf3`, `qperf`

Limitations: tremendous manual efforts and inadequacy

Outline

1 Challenges

2 Existing Solutions

3 ntprof Design and Implementation

4 Evaluation

5 Summary

ntprof: an NVMe/TCP Profiler

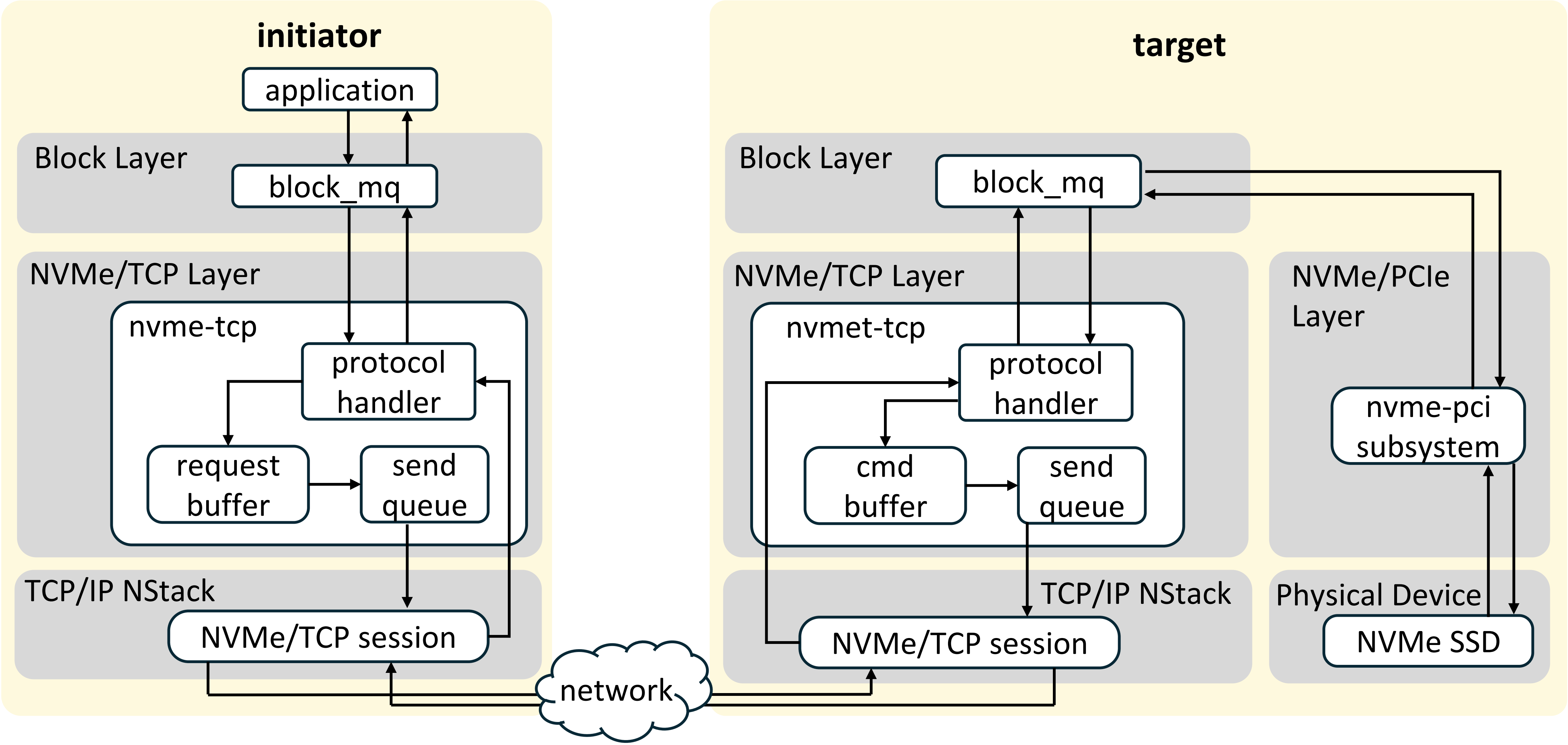
- `ntprof` is a profiling utility that dissects NVMe/TCP execution characteristics
  - Break down software processing overhead over I/O path
  - Analyze how NVMe/TCP interacts with underlying storage subsystems
  - Locate application bottlenecks when running atop NVMe/TCP disaggregated storage

- Design goals
  - Informative, profiling rich, lightweight

Key idea: model the NVMe/TCP as a network and apply network monitoring techniques

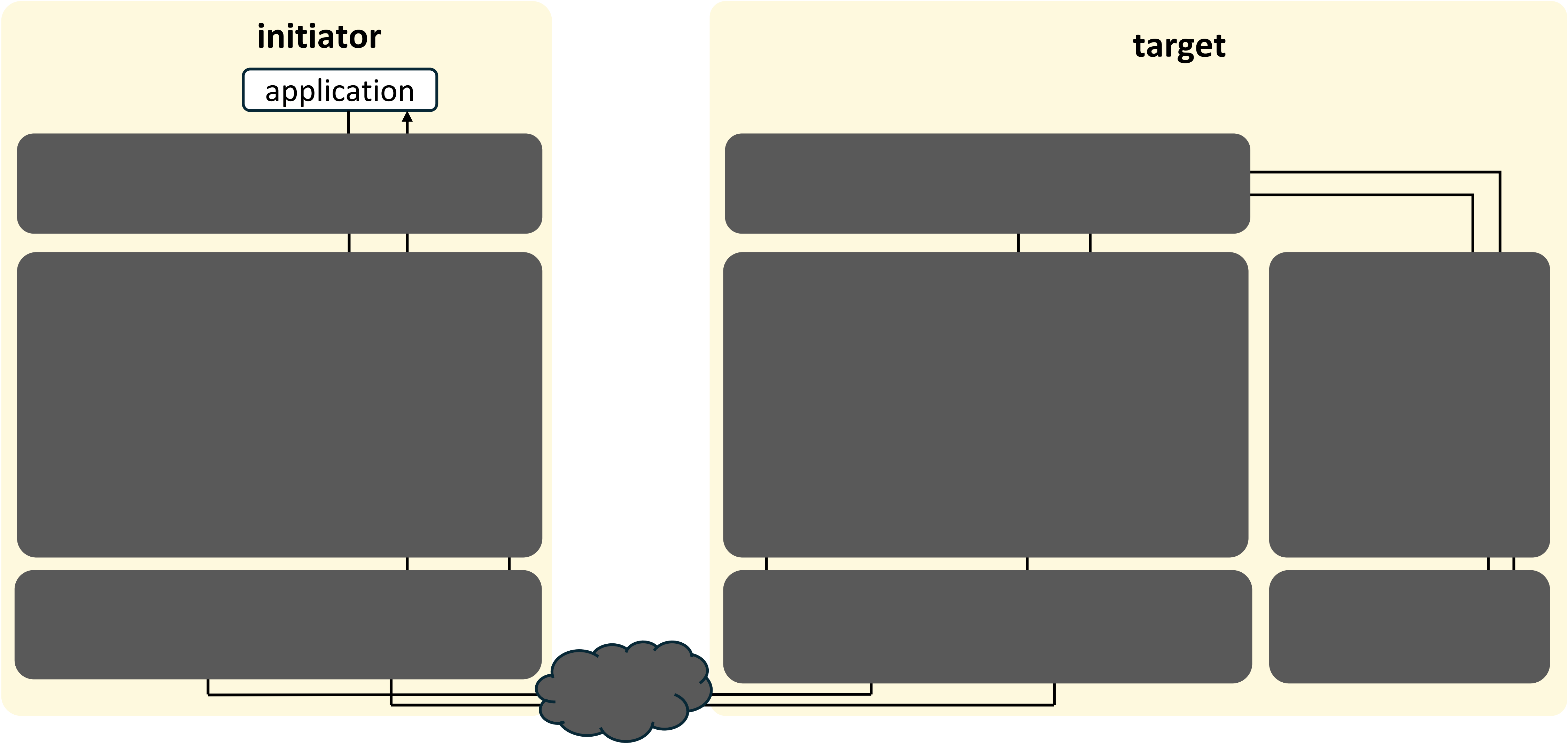
Model NVMe/TCP as a Network

- View a stage as a software switch



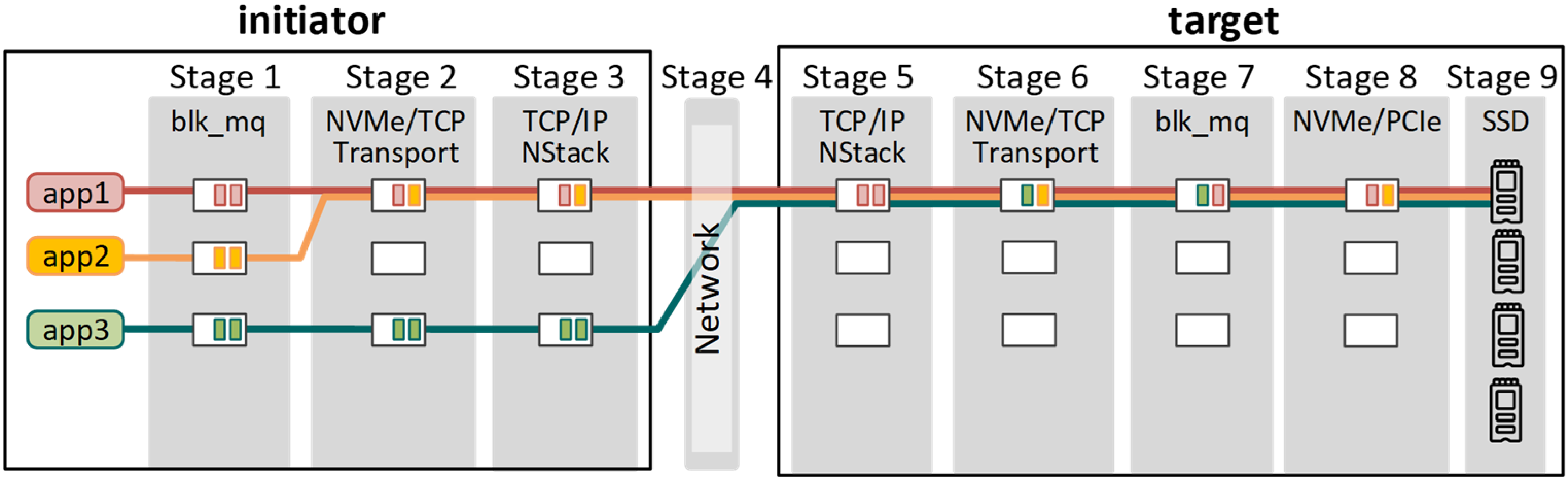
Model NVMe/TCP as a Network

- View a stage as a software switch



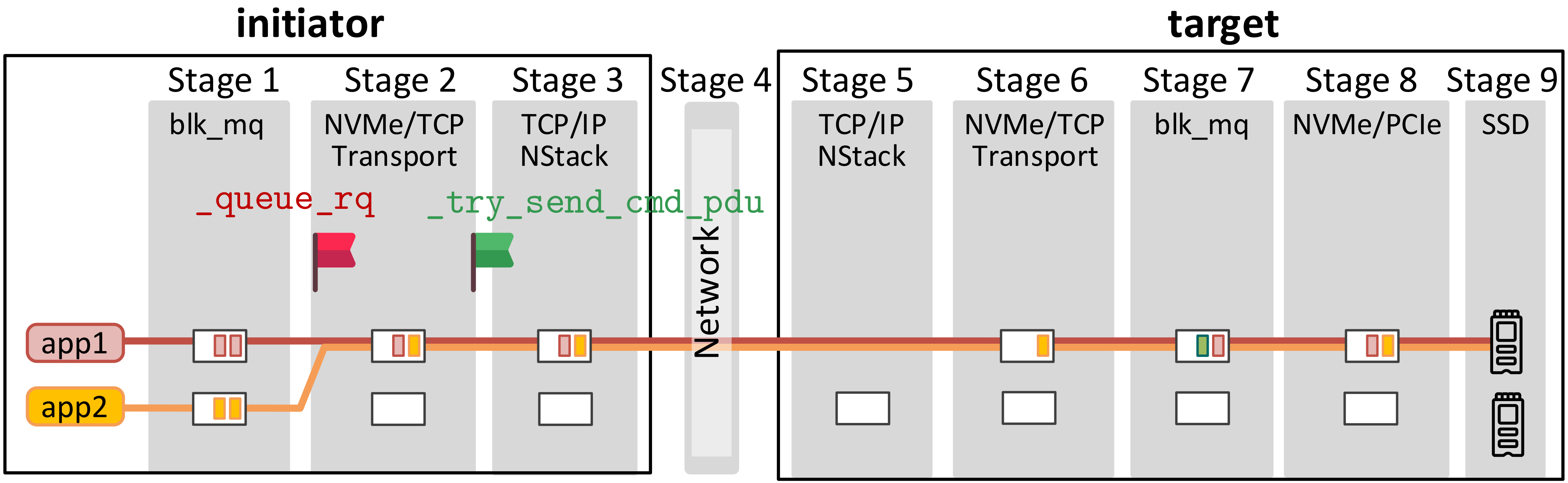
Modeling Stages

- View a stage as a software switch



Adding Tracepoints

- Collect and query statistics in each software switch
- **Tracepoint:** an instrumentational point exposing a hook to a customized function
- Examples:
  - _queue_rq: when a block I/O enters the NVMe/TCP layer
  - _try_send_cmd_pdu: when a PDU is copied to the TCP socket buffer

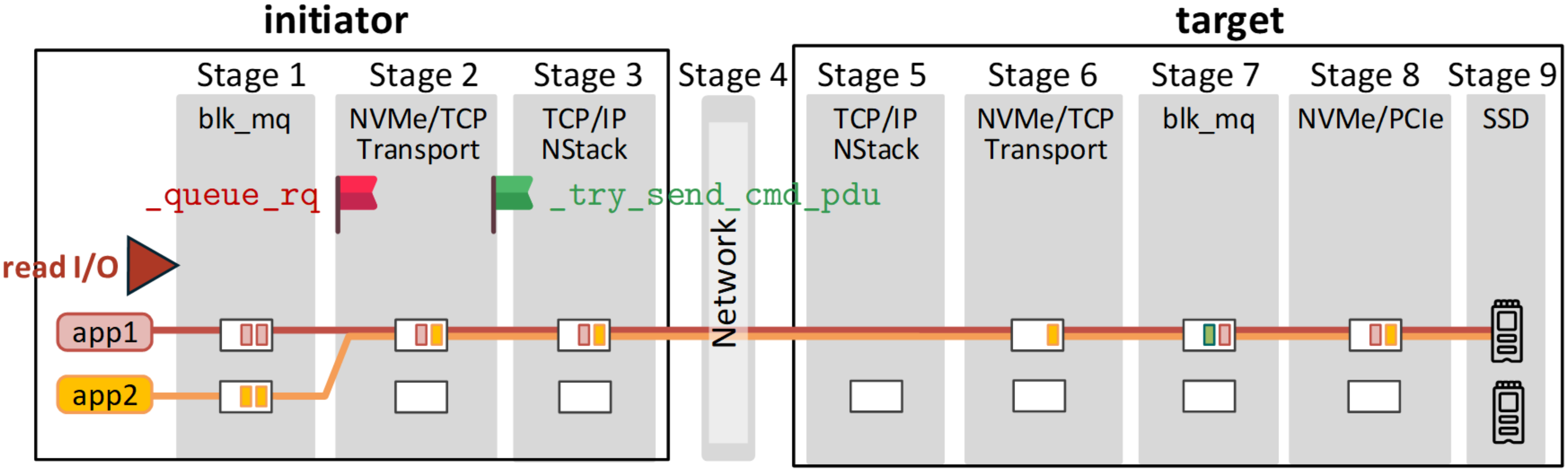


Collecting Tracepoints

- TPP[SIGCOMM’14]: a proactive network monitoring system
- Issue special I/O requests to collect runtime statistics

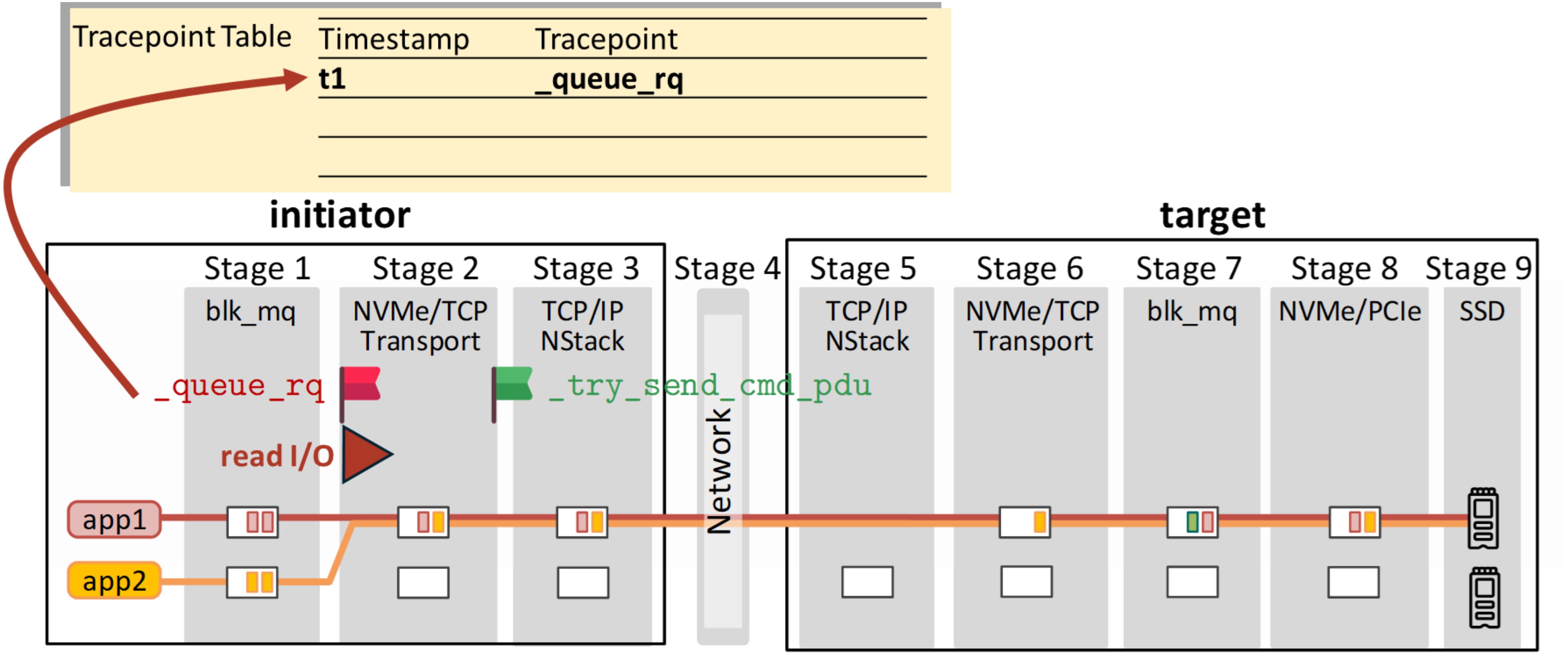
Tracepoint Table

Timestamp	Tracepoint



Collecting Tracepoints

- TPP[SIGCOMM’14]: a proactive network monitoring system
- Issue special I/O requests to collect runtime statistics



Tracepoint Table

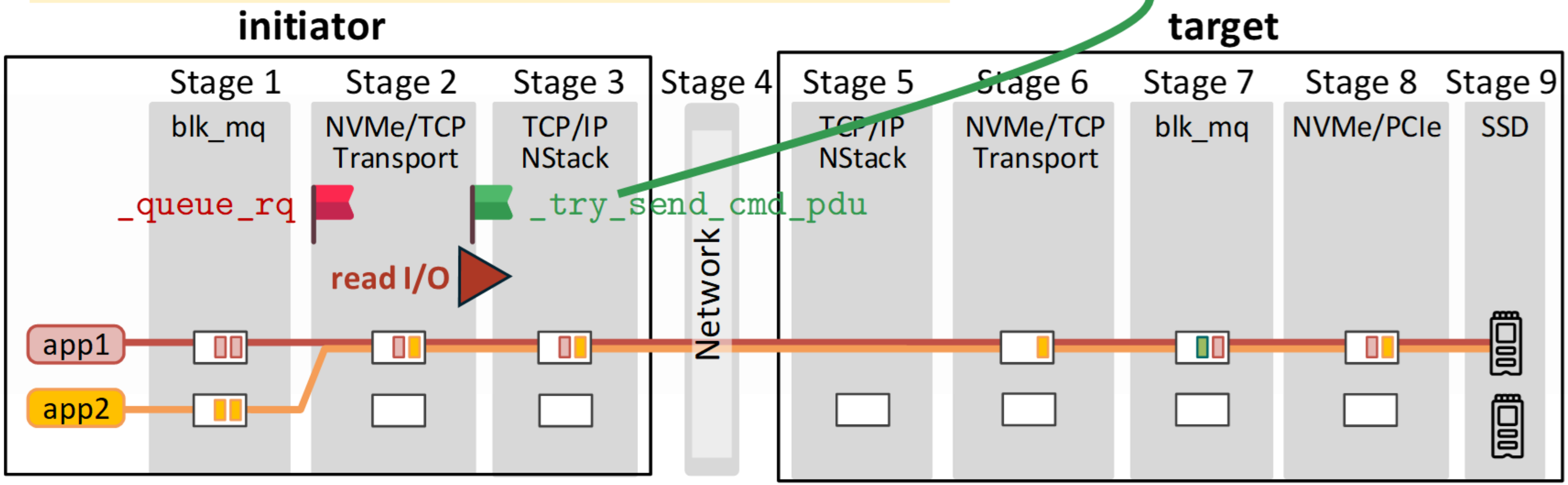
Timestamp	Tracepoint
t1	**_queue_rq**

Collecting Tracepoints

- TPP[SIGCOMM’14]: a proactive network monitoring system
- Issue special I/O requests to collect runtime statistics

Tracepoint Table

Timestamp	Tracepoint
t1	_queue_rq
t2	**_try_send_cmd_pdu**

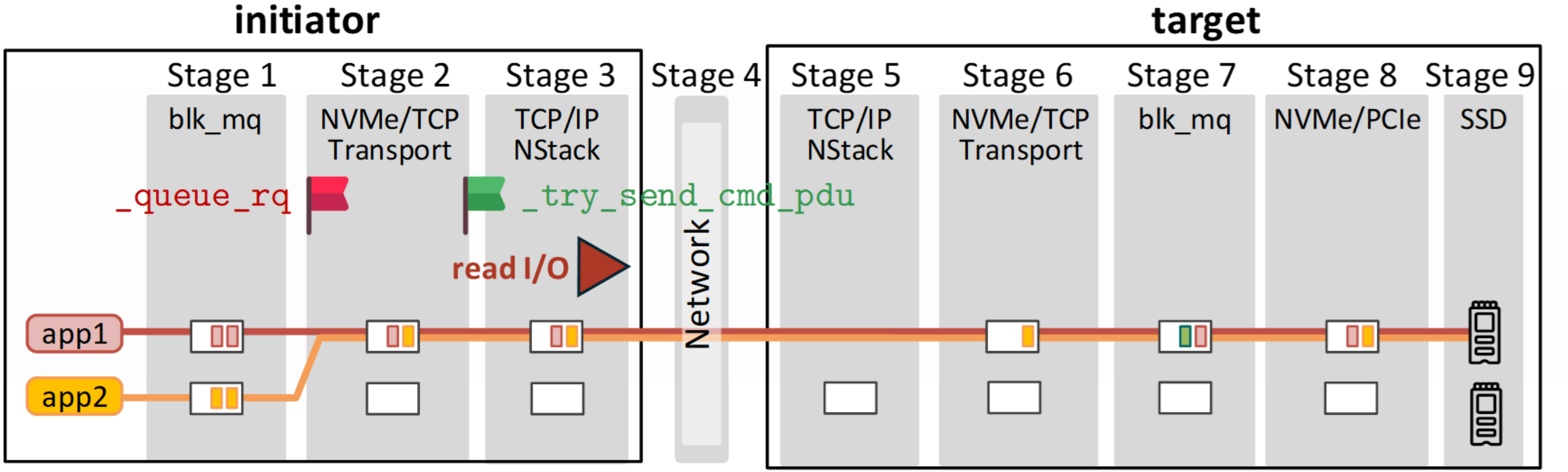


Collecting Tracepoints

- TPP[SIGCOMM’14]: a proactive network monitoring system
- Issue special I/O requests to collect runtime statistics

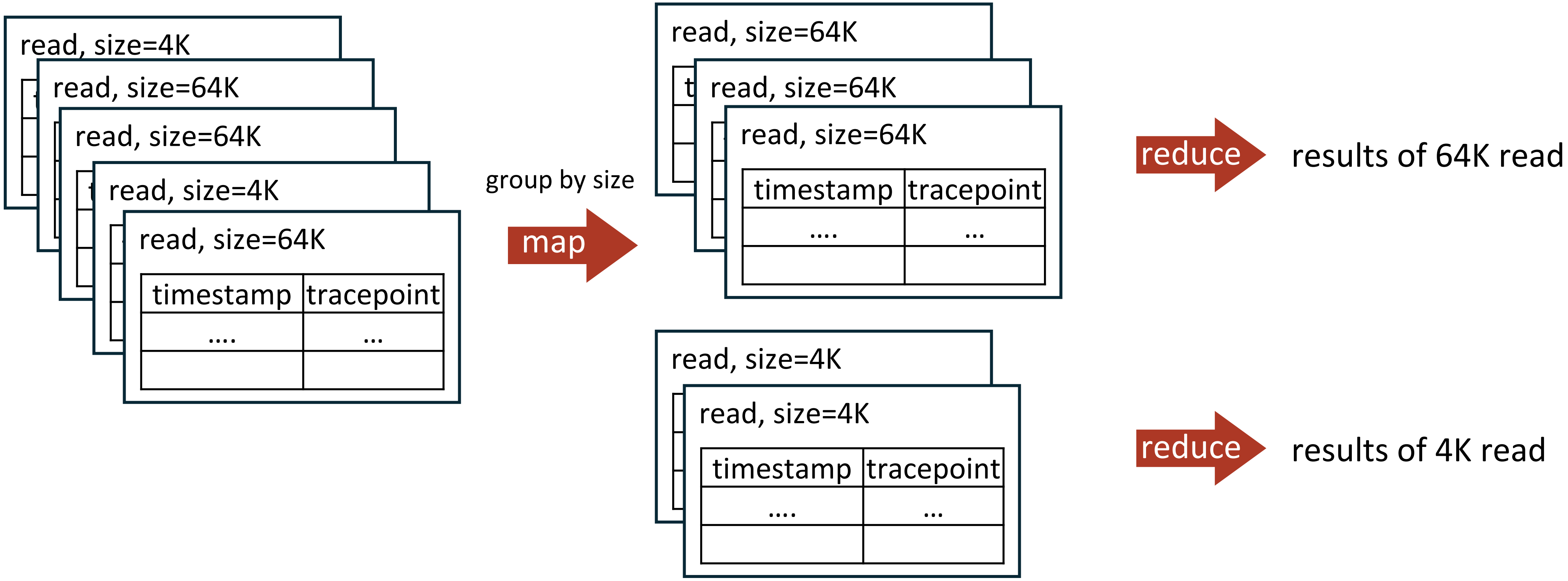
Tracepoint Table

Timestamp	Tracepoint
t1	_queue_rq
t2	_try_send_cmd_pdu
…	…



Profiling Results Analyzer

- MapReduce-like processing
  - grouper/aggregator functions



read, size=4K

read, size=64K

read, size=64K

read, size=4K

read, size=64K

timestamp	tracepoint
….	…

group by size

map

read, size=64K

read, size=64K

read, size=64K

timestamp	tracepoint
….	…

read, size=4K

read, size=4K

timestamp	tracepoint
….	…

reduce

results of 64K read

reduce

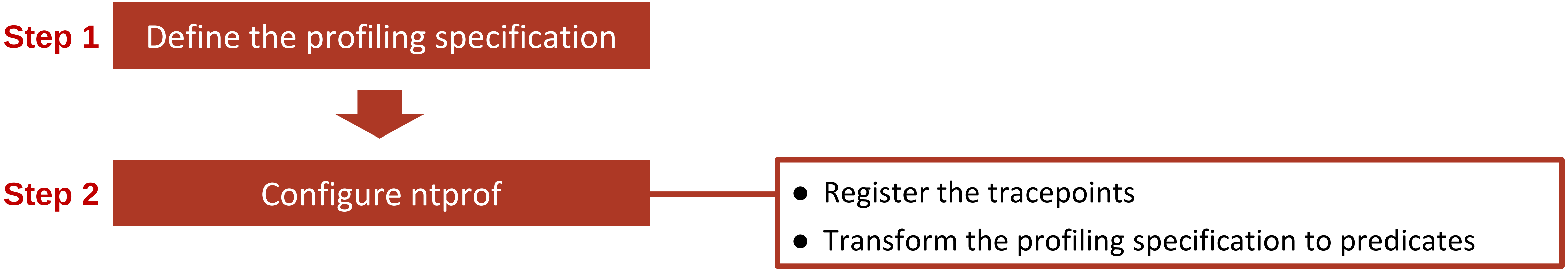
results of 4K read

ntprof Workflow

Step 1 Define the profiling specification

- Workload specification, e.g., type, size
- Profiler specification, e.g., sample frequency
- Execution specification, e.g., application setup
- Report specification, e.g., analyzing statistics

ntprof Workflow

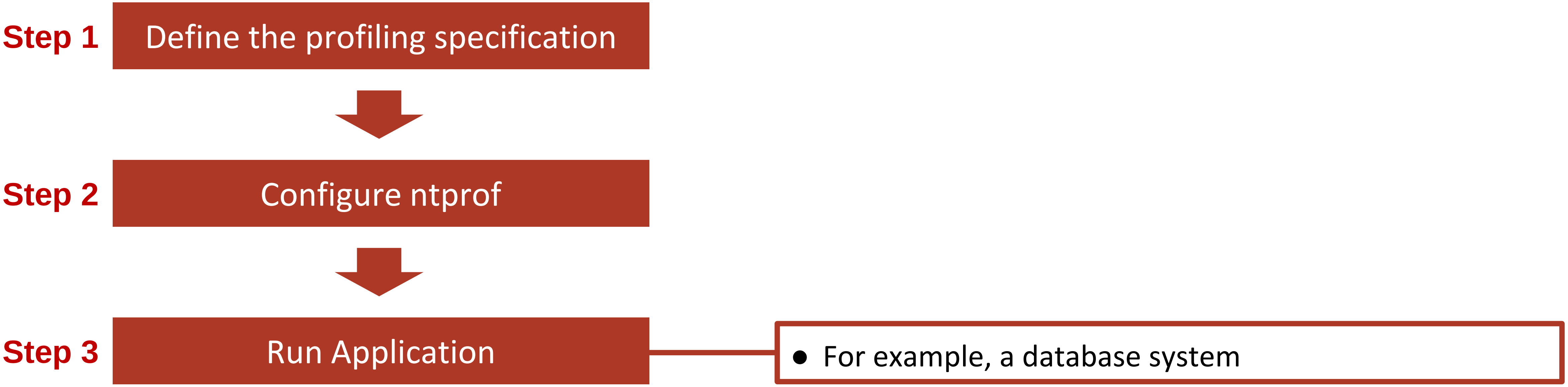


Step 1 Define the profiling specification

Step 2 Configure ntprof

- Register the tracepoints
- Transform the profiling specification to predicates

ntprof Workflow



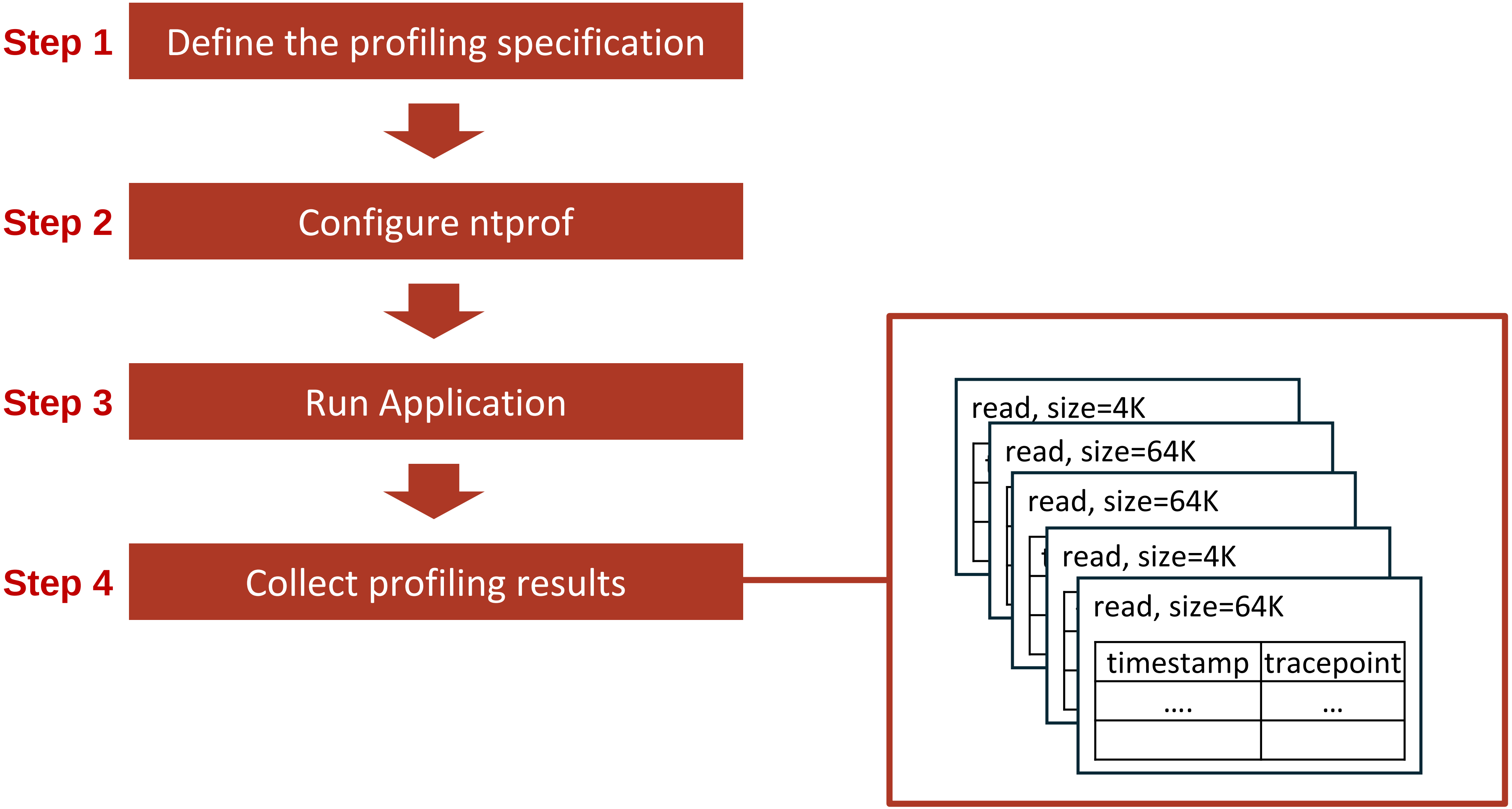
Step 1 Define the profiling specification

Step 2 Configure ntprof

Step 3 Run Application

- For example, a database system

ntprof Workflow



Step 1 Define the profiling specification

Step 2 Configure ntprof

Step 3 Run Application

Step 4 Collect profiling results

read, size=4K

read, size=64K

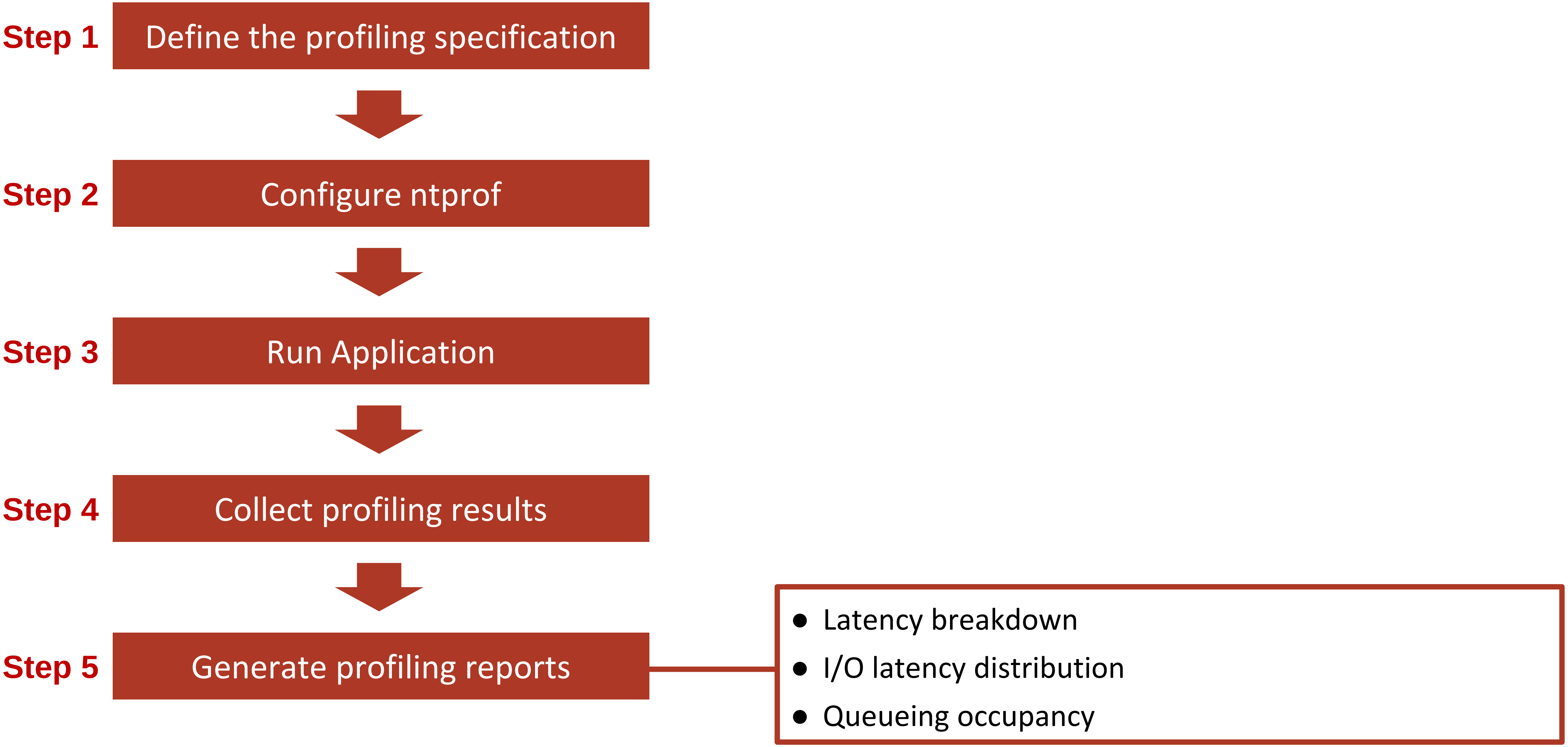
read, size=64K

read, size=4K

read, size=64K

timestamp	tracepoint
….	…

ntprof Workflow



Step 1 Define the profiling specification

Step 2 Configure ntprof

Step 3 Run Application

Step 4 Collect profiling results

Step 5 Generate profiling reports

- Latency breakdown
- I/O latency distribution
- Queueing occupancy

Outline

1 Challenges

2 Existing Solutions

3 ntprof Design and Implementation

4 Evaluation

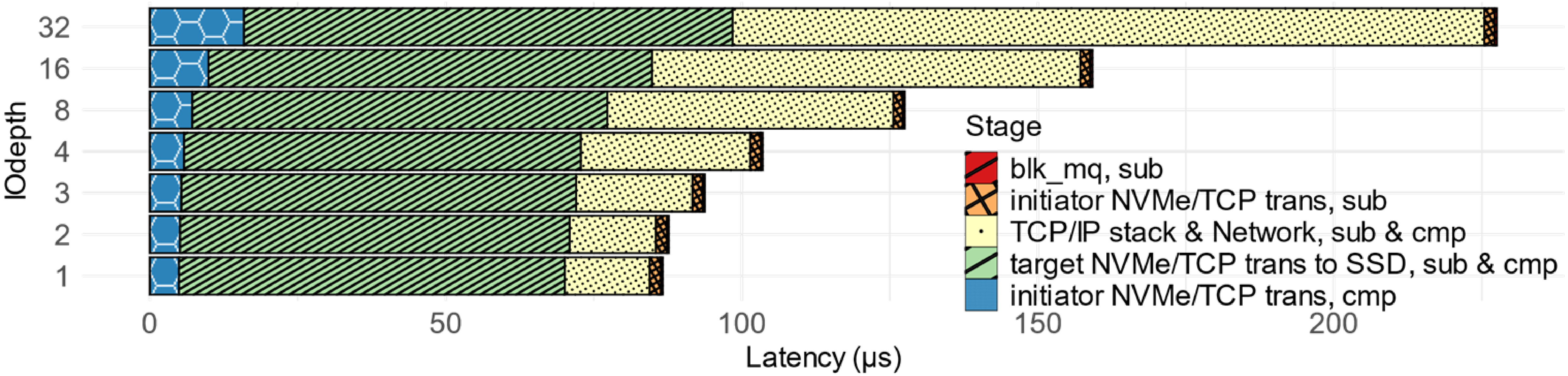
5 Summary

Experimental Methodology

- Hardware testbed – sm110p in CloudLab
  - 2x Intel Xeon Silver 4314CPU, 128GB DDR4
  - 100 GbE NVIDIA/Mellanox CX6 + 4 NVMe SSDs
- Software setup
  - Ubuntu 20.04 with kernel v5.15.143
  - Synthetic (fio) and real-world application (Apache IoTDB, F2FS)
- Implementation details
  - Kernel modification, e.g, adding tracepoints in nvme-tcp kernel module
  - A new kernel module, a user space utility (about 7K LOCs)

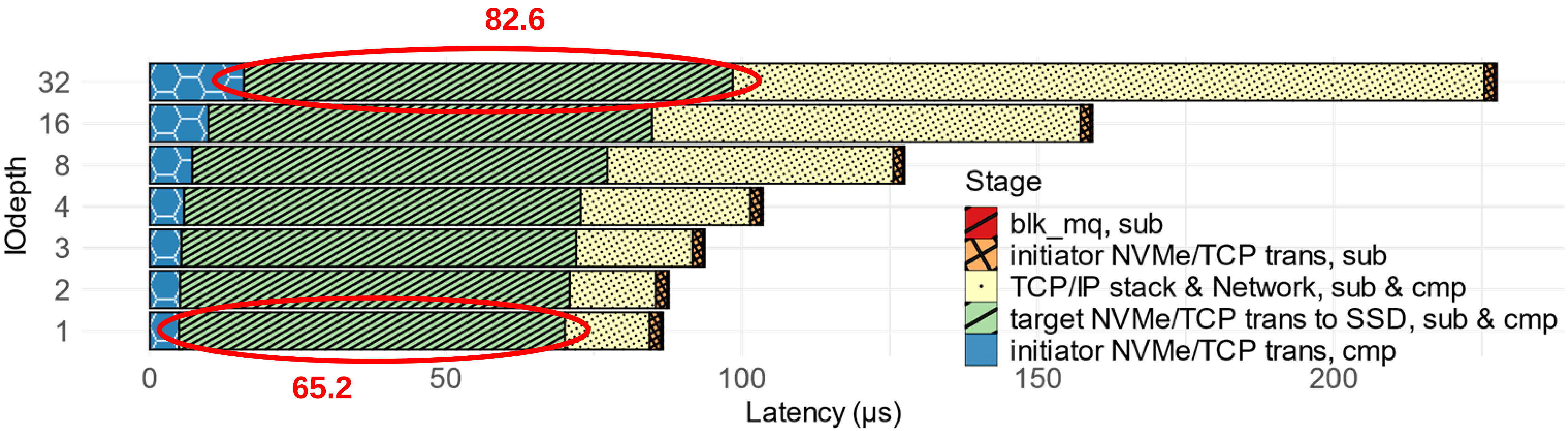
Use Case: Latency Breakdown

- Use ntprof to break down the latency of 4KB random read over NVMe/TCP



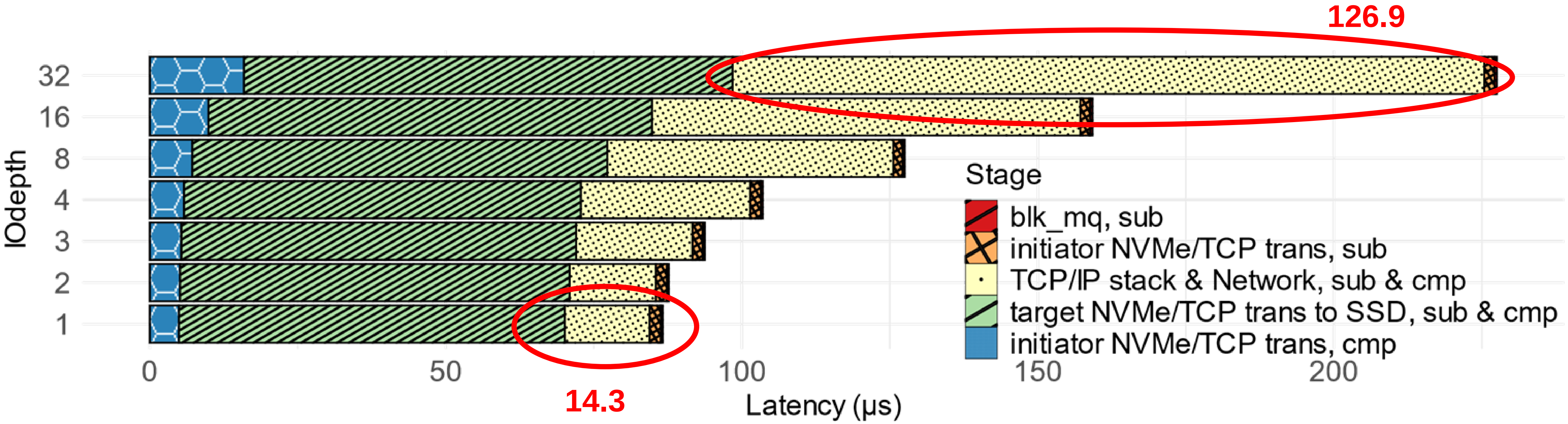
Use Case: Latency Breakdown

- Use ntprof to break down the latency of 4KB random read over NVMe/TCP



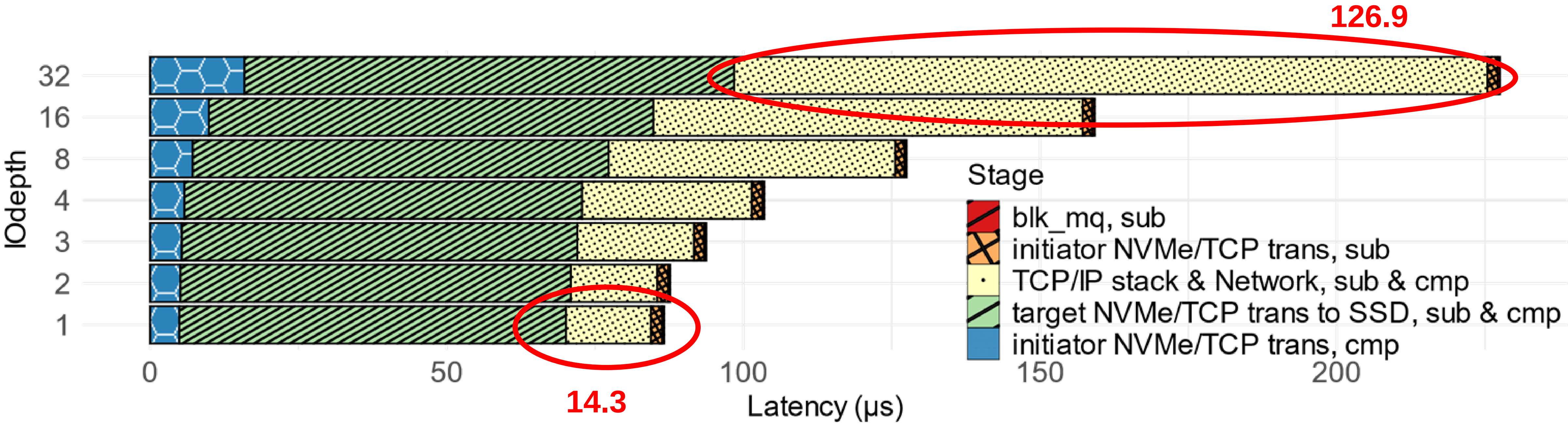
Use Case: Latency Breakdown

- Use ntprof to break down the latency of 4KB random read over NVMe/TCP



Use Case: Latency Breakdown

- Use ntprof to break down the latency of 4KB random read over NVMe/TCP



Bandwidth(MB/s)
achieved / max = 300 / 12,000

Other Use Cases

- Software bottleneck localization
- Hardware bottleneck localization
- Interference analysis of concurrent I/O streams
- Real world application diagnostics
- …

Outline

1 Challenges

2 Existing Solutions

3 ntprof Design and Implementation

4 Evaluation

5 Summary

Summary

- NVMe/TCP: emerging disaggregated storage protocol but lacking a profiling tool
- Existing solutions are tedious and inadequate
- ntprof: model the NVMe/TCP as network
  - View stages as software switches
  - Use tracepoints for statistics collection
  - Apply map-reduce processing for analyzing results
- ntprof enables different I/O profiling tasks
  - Latency breakdown
  - Software/Hardware bottleneck localization
  - Interference analysis
- GitHub: https://github.com/netlab-wisconsin/nvme-tcp
- Project website: https://ntprof.cs.wisc.edu/