RedPlane: Enabling Fault-Tolerant Stateful In-Switch Applications

Daehyeok Kim[§‡]

Jacob Nelson[‡], Dan Ports[‡], Vyas Sekar[§], Srinivasan Seshan[§]

[§]Carnegie Mellon University [‡]Microsoft

Programmable networks are stateful

Classical switches

Match	Action
IP addr/prefix	Forward (port)

"Stateless" packet processing

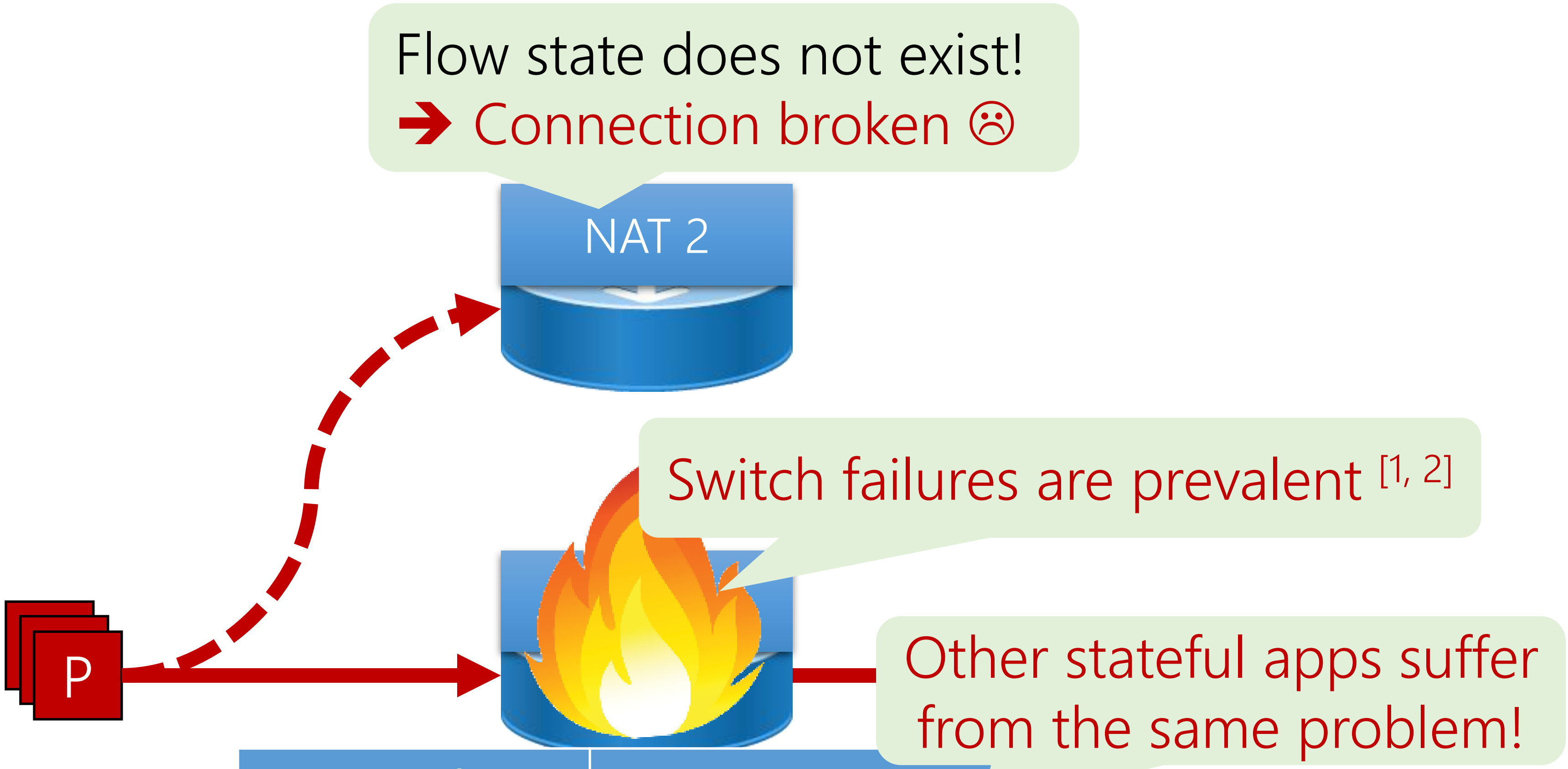
Programmable "data plane" switches

Stateful in-switch applications:
network functions, monitoring, accelerating distributed systems

Match	Stateful action
(IP addr, port)	NAT (IP addr., port)

Programmable switching ASICs
➔ "Stateful" packet processing

Problem: Switch failure

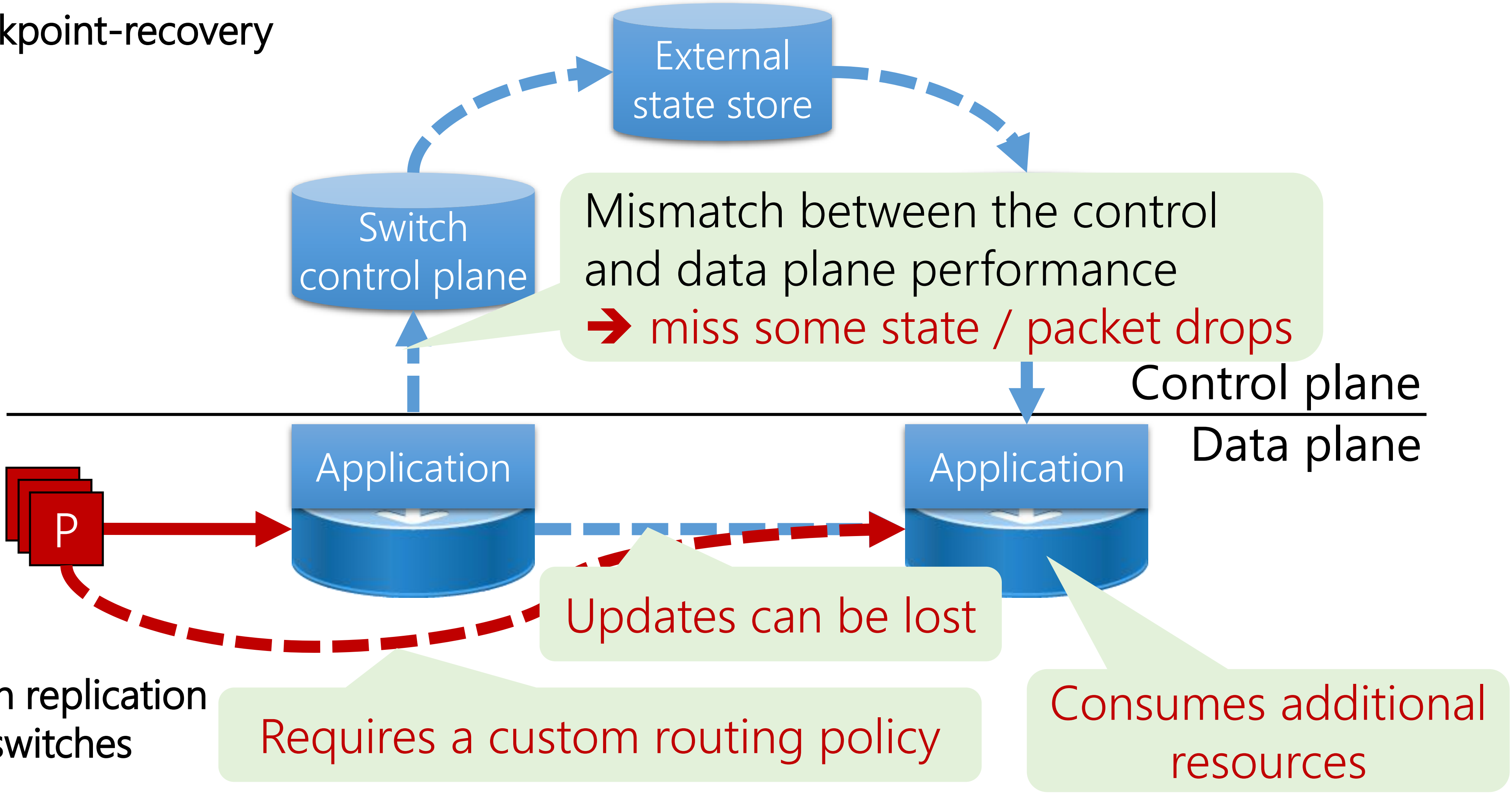


Match (Flow ID)	NAT action (IP, Port)
(10.0.0.1, 4321)	(192.168.10.1, 1234)

[1] Liu et al., Crystalnet: Faithfully emulating large production networks. In ACM SOSP 2017.
[2] Meza et al., A large scale study of data center network reliability. In ACM IMC 2017.

Strawman solutions

S1: Checkpoint-recovery



S2: Chain replication among switches

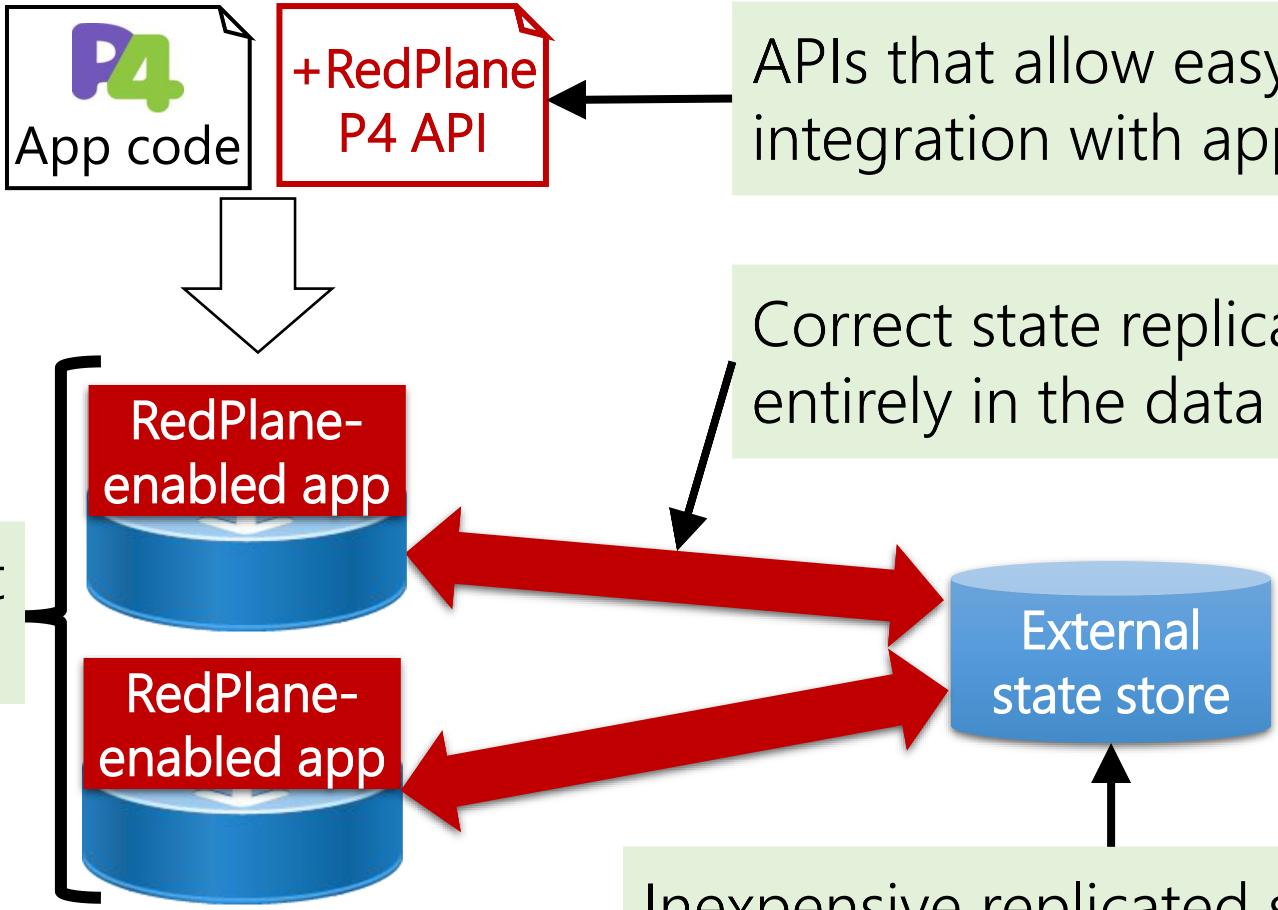
Our work: RedPlane

APIs that allow easy integration with apps

Correct state replication entirely in the data plane

One big fault-tolerant switch abstraction!

Inexpensive replicated state store on commodity servers



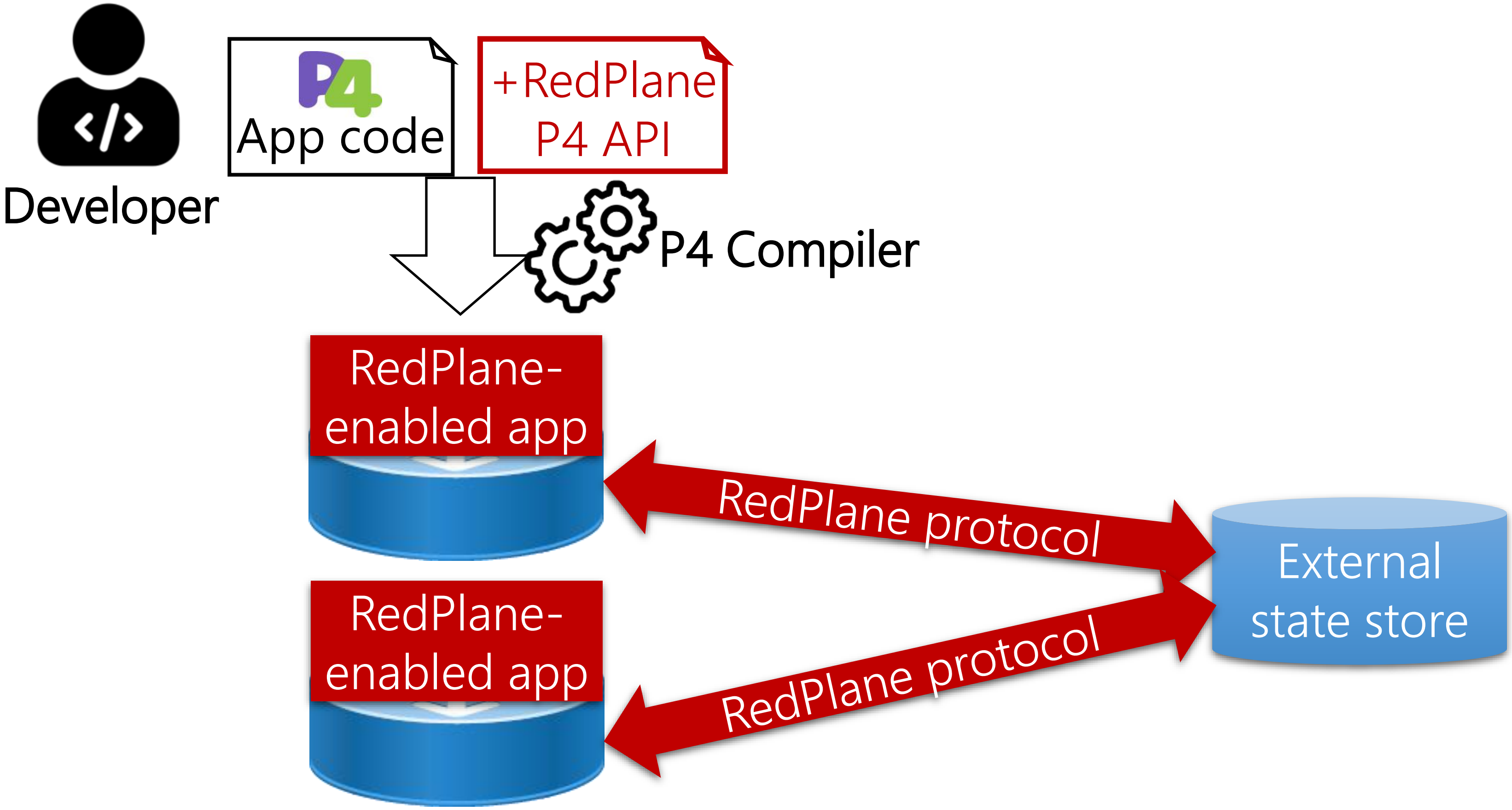
Outline

RedPlane motivation

RedPlane design

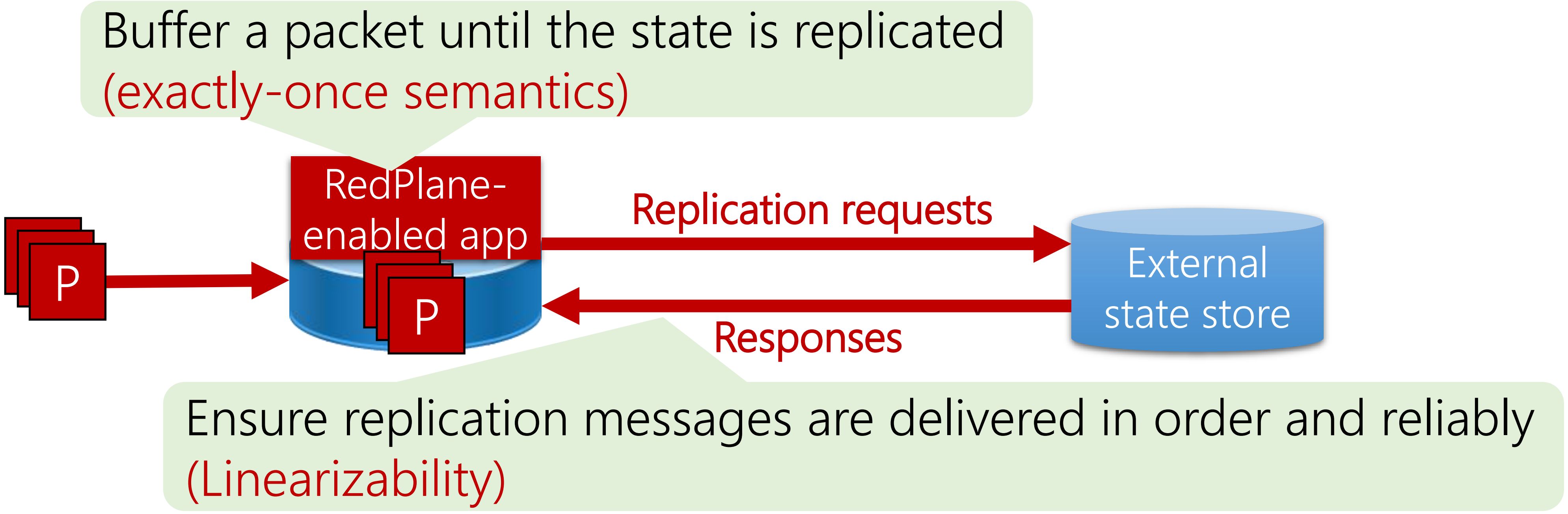
Results

RedPlane design overview



Challenge 1: Correct replication in the data plane

Strawman: strict correctness used in server-based replicated systems



Challenge 1: Correct replication in the data plane

Strawman: strict correctness used in server-based replicated systems

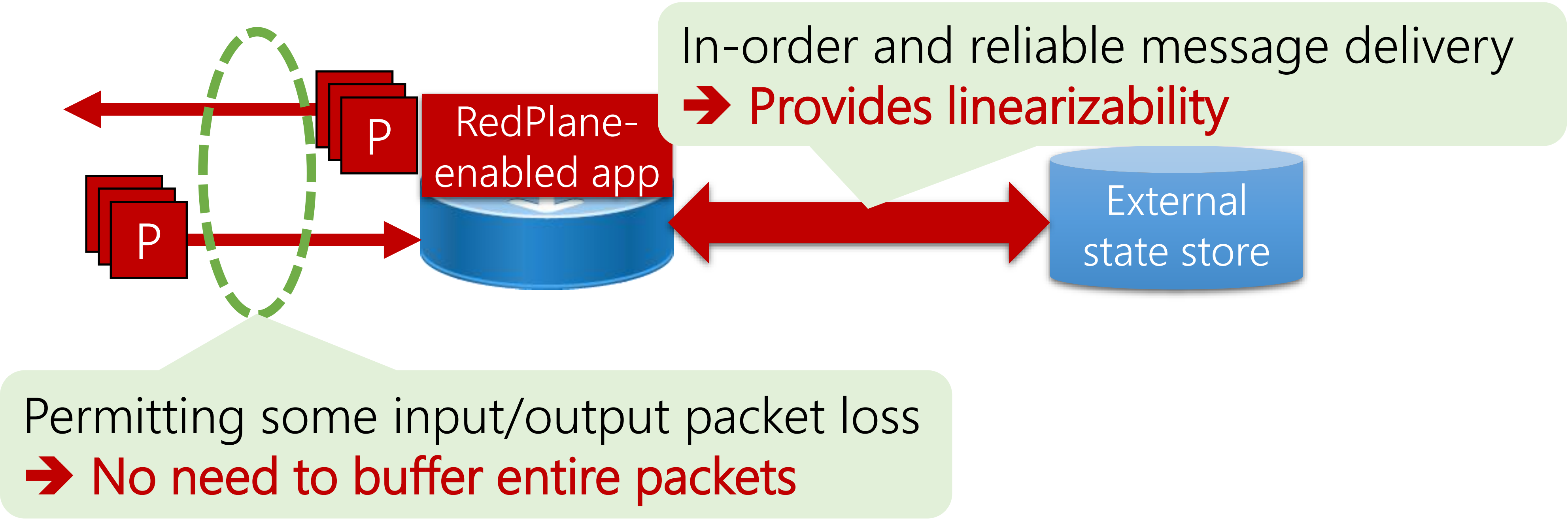
Expensive to buffer entire packets ☹

Expensive to realize reliable transport in the switch data plane ☹

Linearizable mode: Relaxed correctness

Insight: End-to-end network apps already tolerate lossy networks!

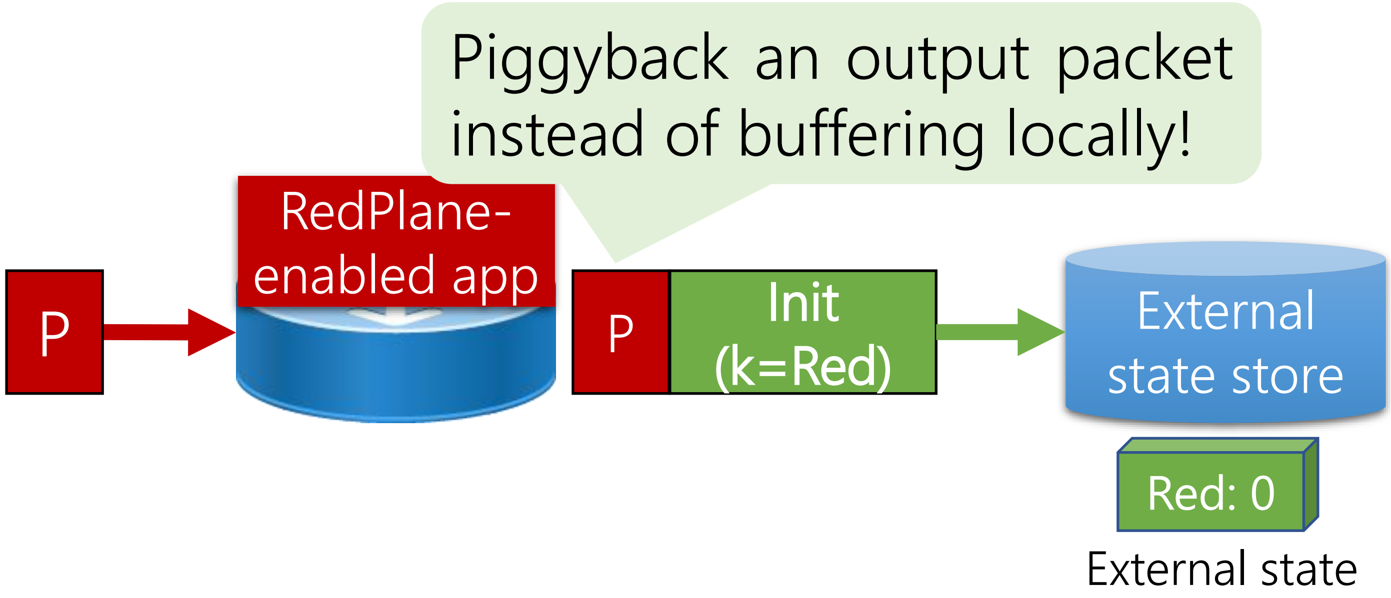
Our approach: Linearizability-based relaxed correctness



Basic RedPlane protocol: Realizing the linearizable mode in the data plane

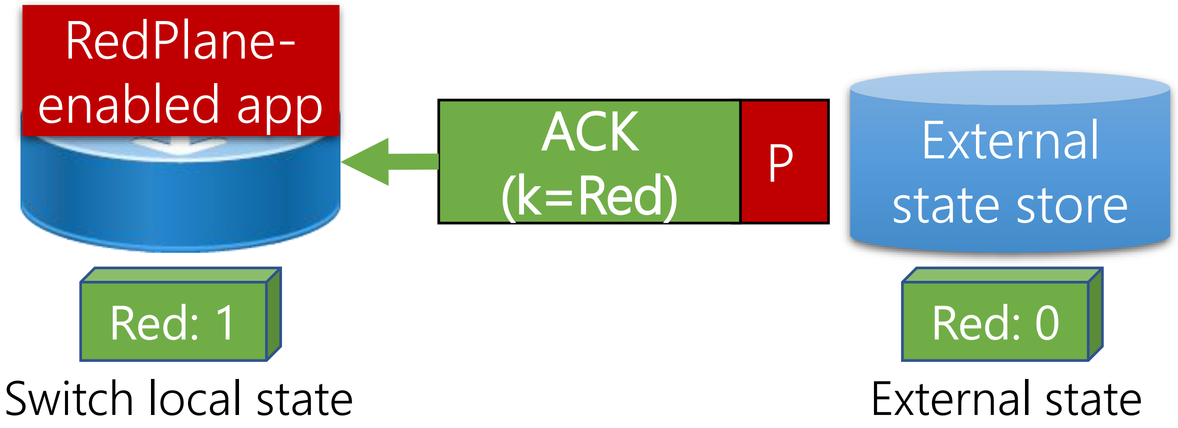
Example: per-flow packet counter

1. Sends a state initialization request



Basic RedPlane protocol: Realizing the linearizable mode in the data plane

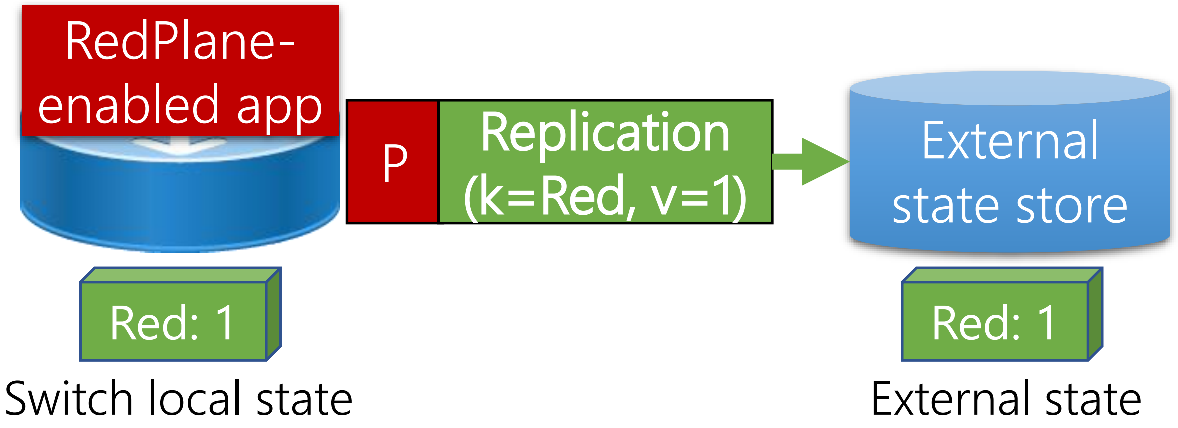
Example: per-flow packet counter



1. Sends a state initialization request
2. Receives an ACK & initializes the local state

Basic RedPlane protocol: Realizing the linearizable mode in the data plane

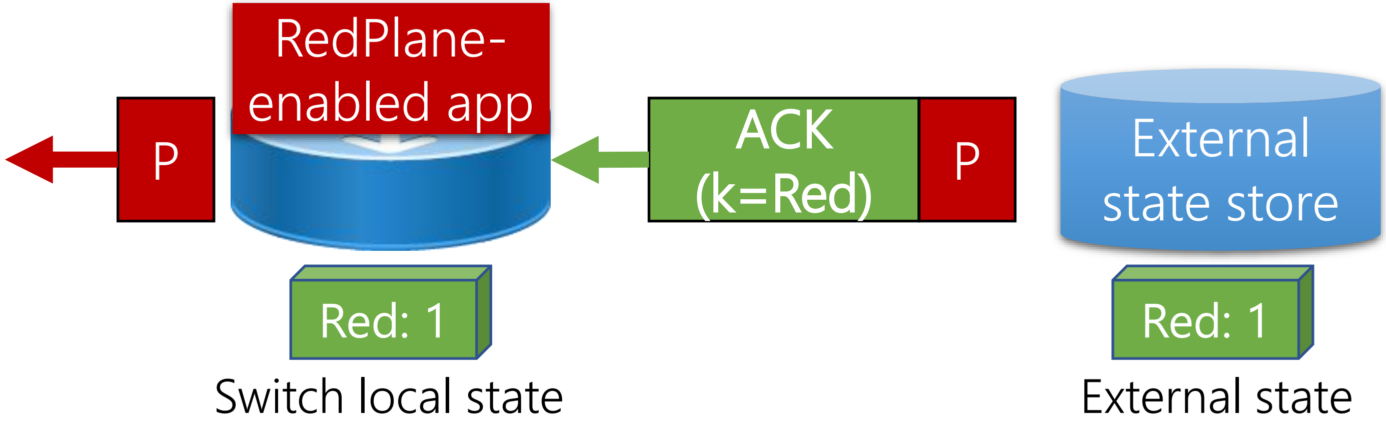
Example: per-flow packet counter



1. Sends a state initialization request
2. Receives an ACK & initializes the local state
3. Replicates the updated state

Basic RedPlane protocol: Realizing the linearizable mode in the data plane

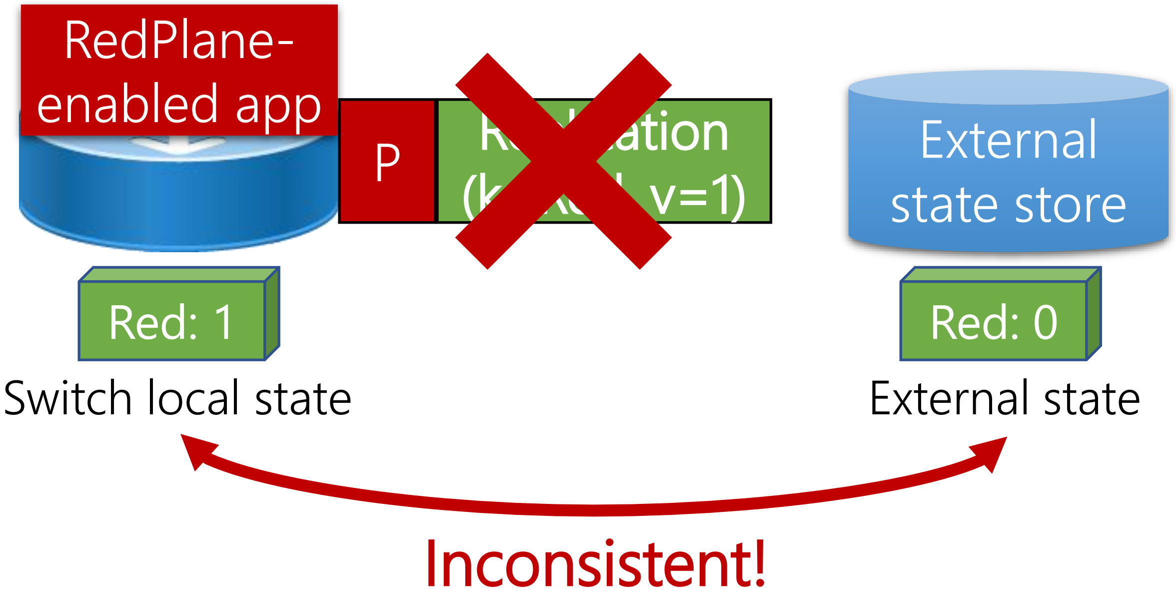
Example: per-flow packet counter



1. Sends a state initialization request
2. Receives an ACK & initializes the local state
3. Replicates the updated state
4. Receives an ACK & releases the output packet

Inconsistency due to unreliable channel

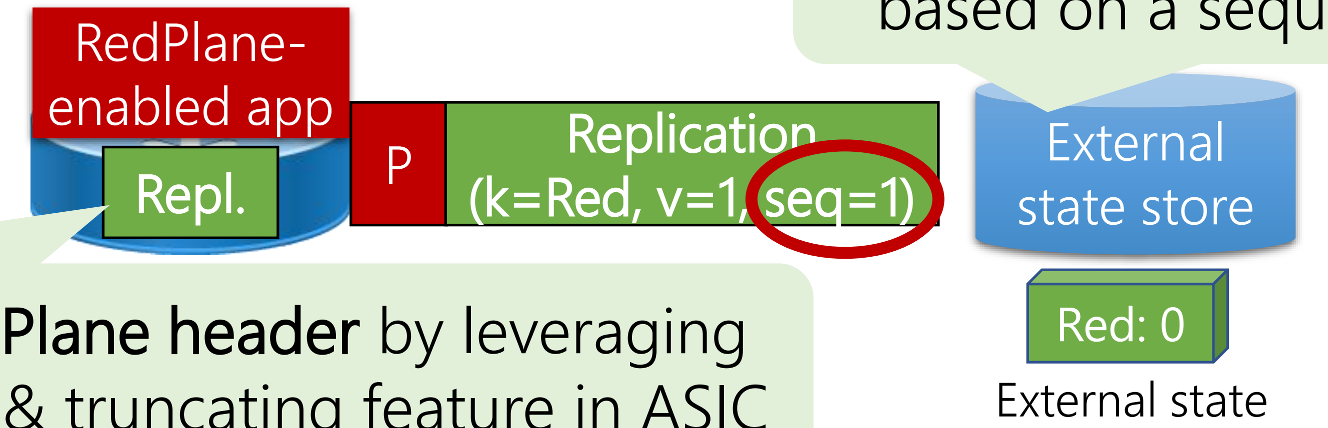
Problem: state in the switch and state store can be inconsistent due to out-of-order requests or request packet loss



Sequencing and lightweight retransmission

Our approach: A simple UDP-based transport with sequencing and lightweight retransmission

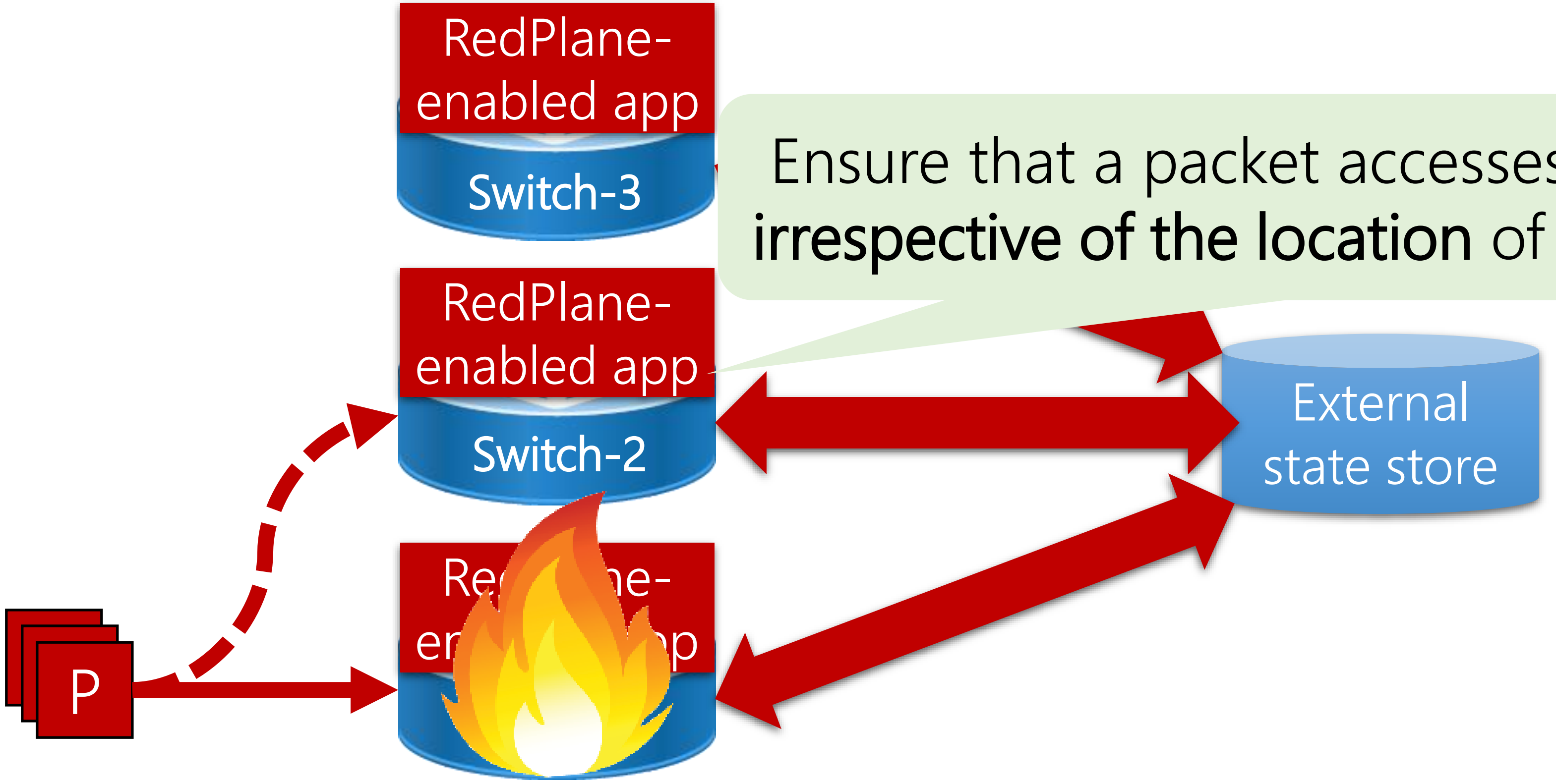
Commits the **latest requests only** based on a sequence number



Buffers **only RedPlane header** by leveraging packet mirroring & truncating feature in ASIC

Challenge 2: Transparent to routing policies

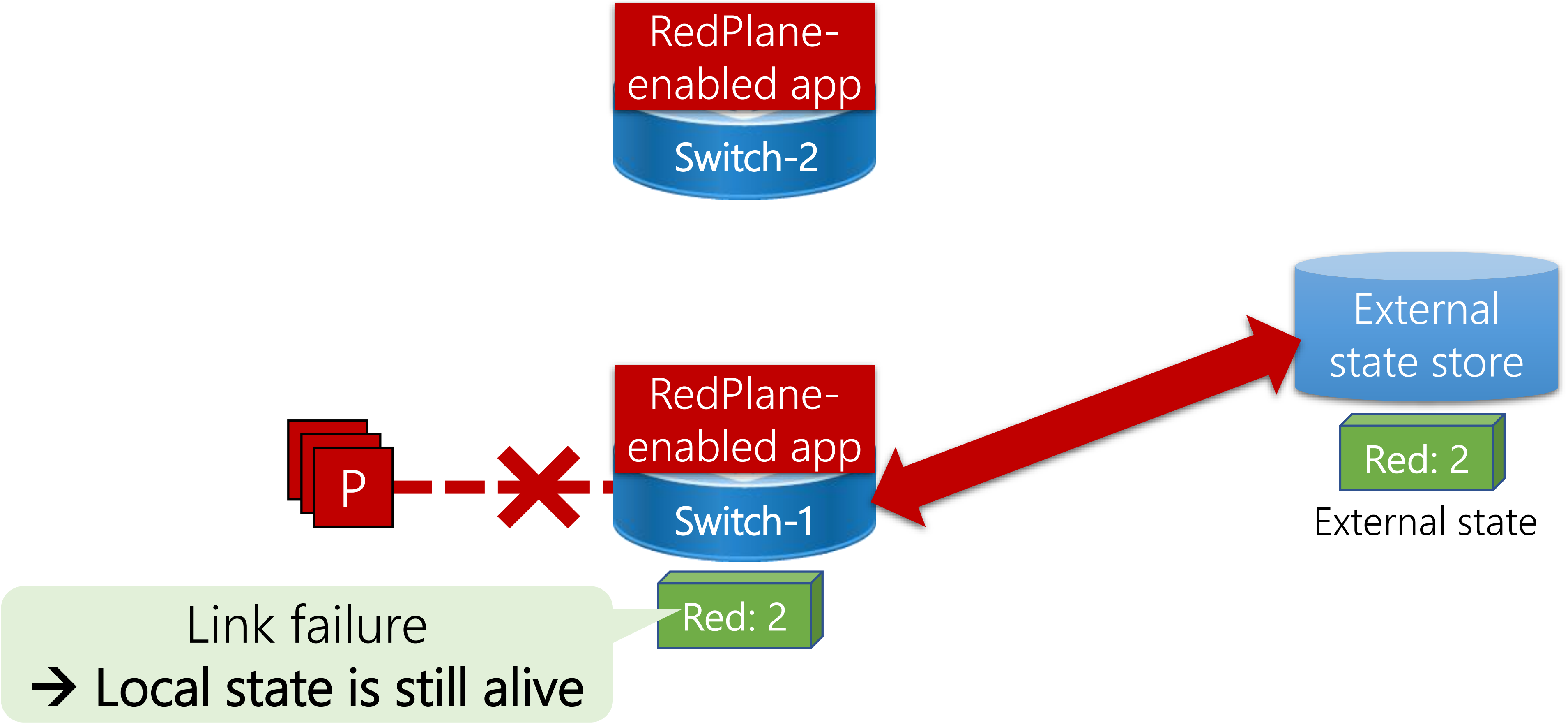
A switch failure or recovery can cause routing traffic to another switch



Ensure that a packet accesses the correct state **irrespective of the location** of the current switch

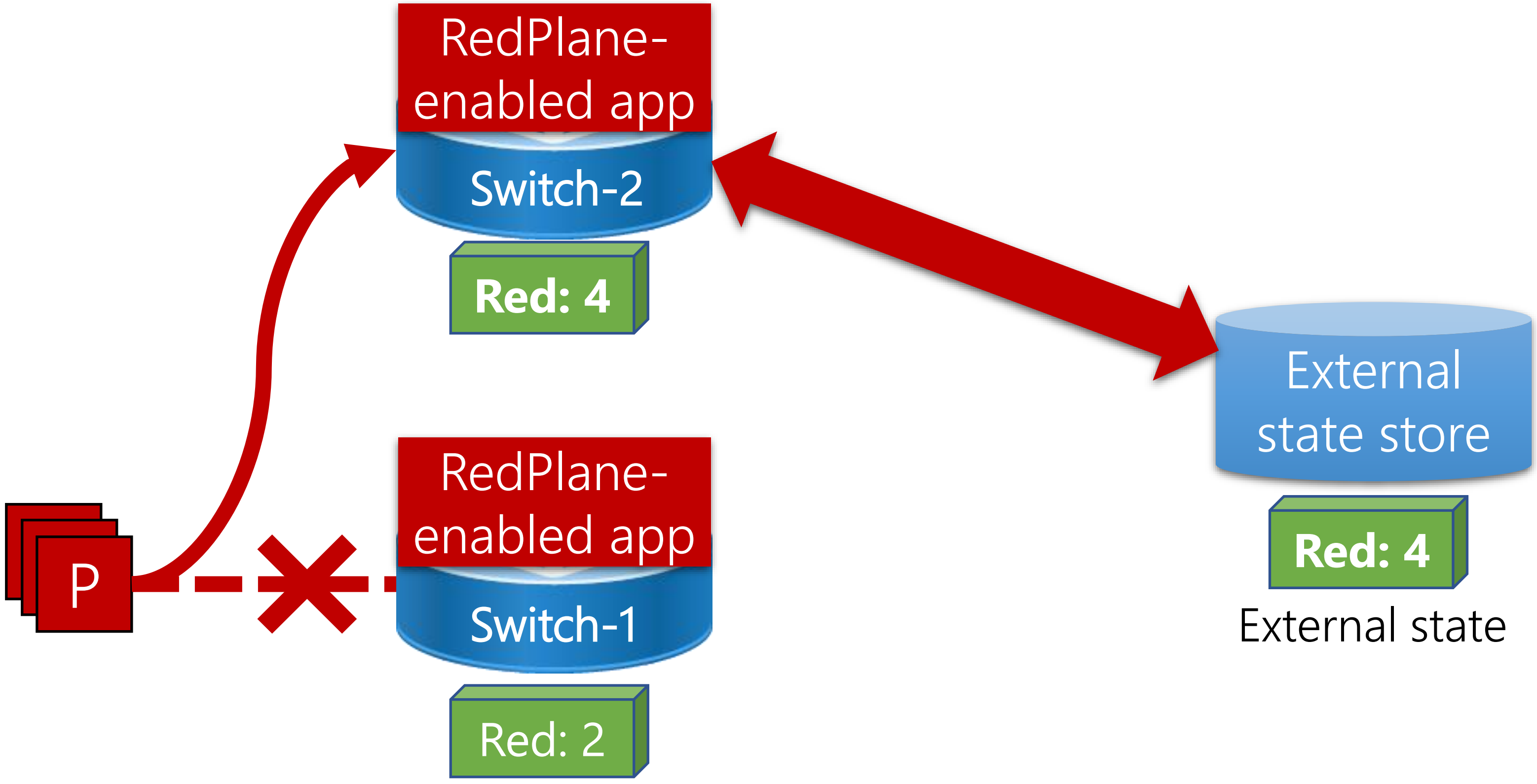
Accessing stale state

Problem: A packet may access state state during failover or recovery



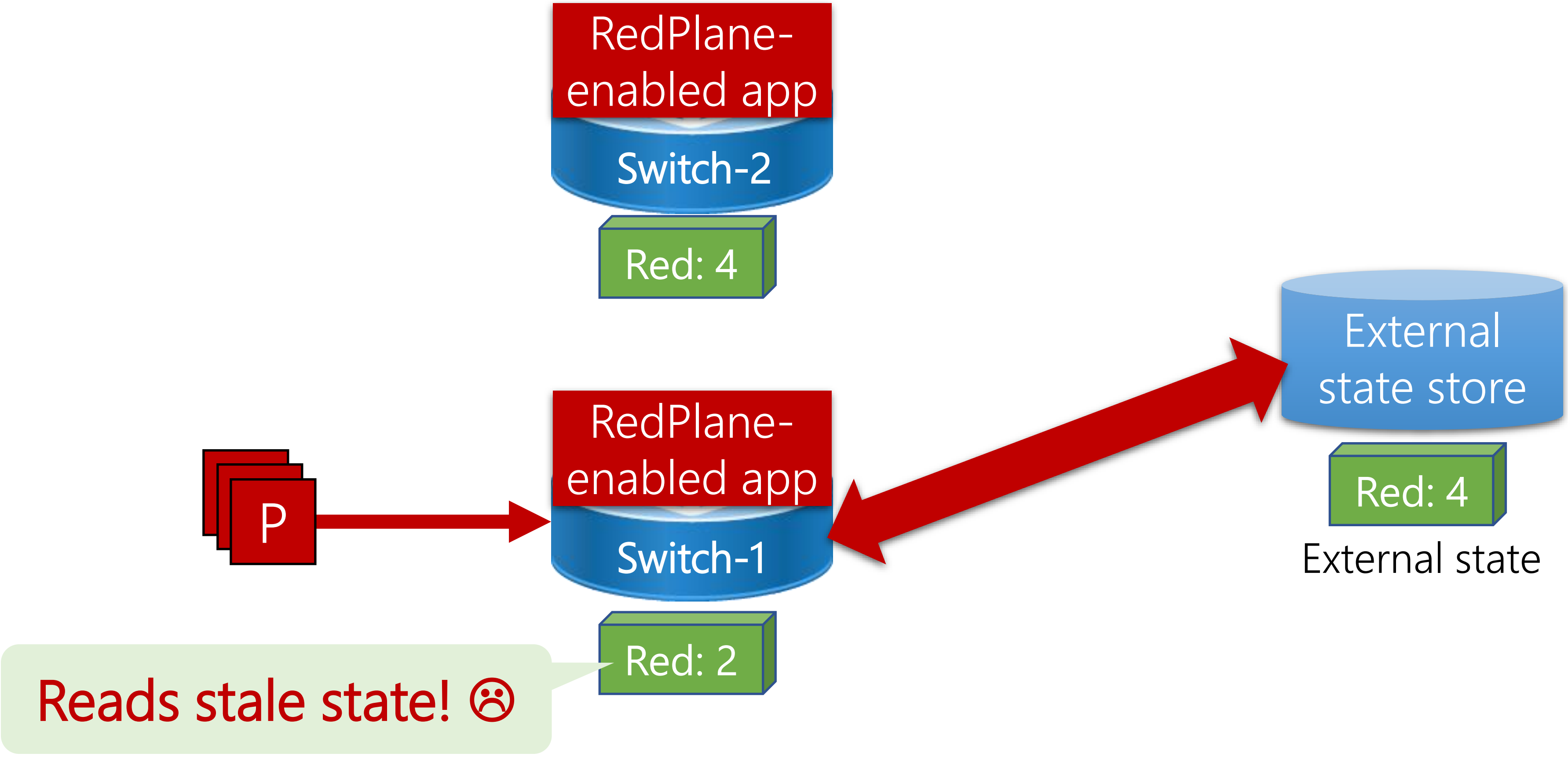
Accessing stale state

Problem: A packet may access state state during failover or recovery



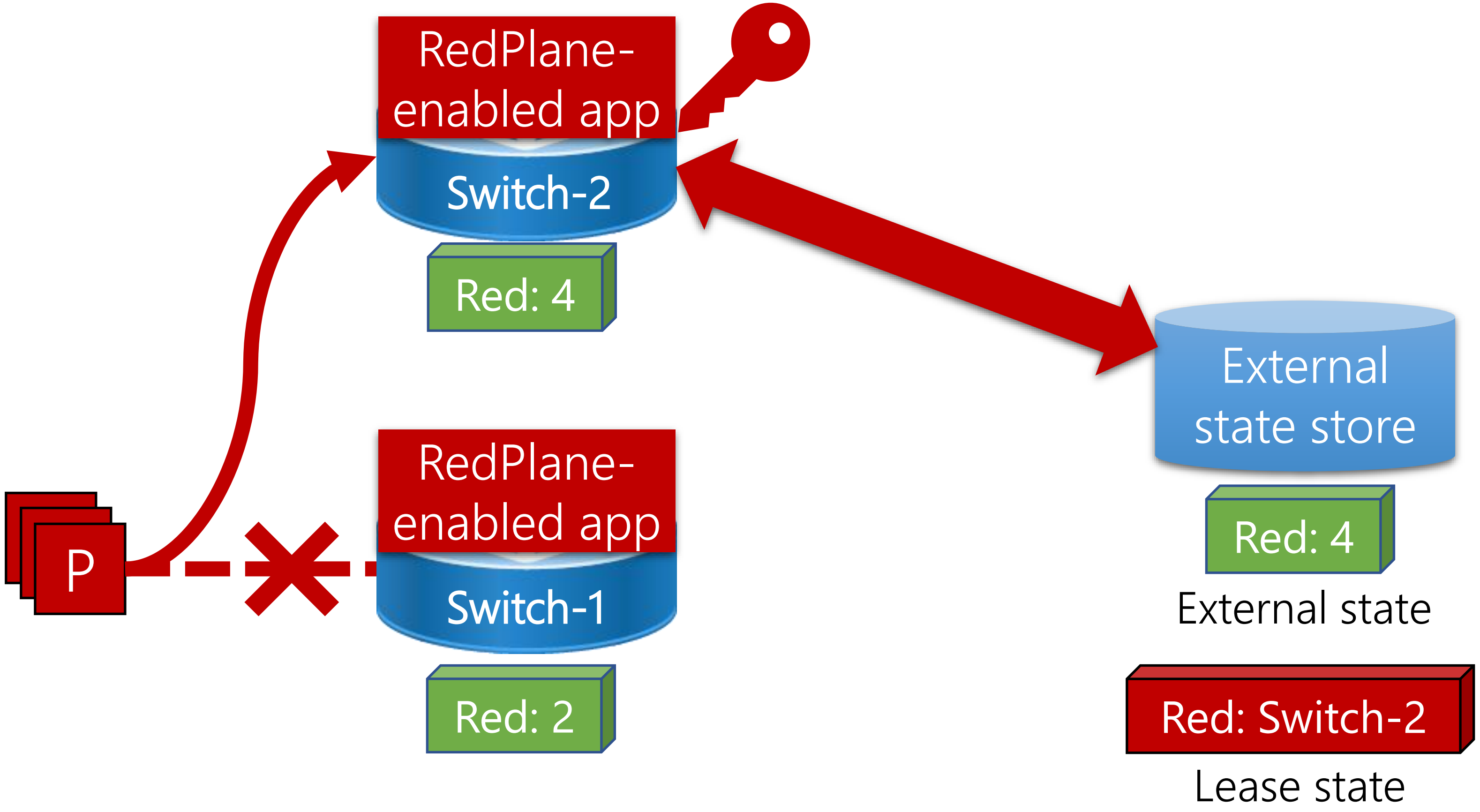
Accessing stale state

Problem: A packet may access state state during failover or recovery



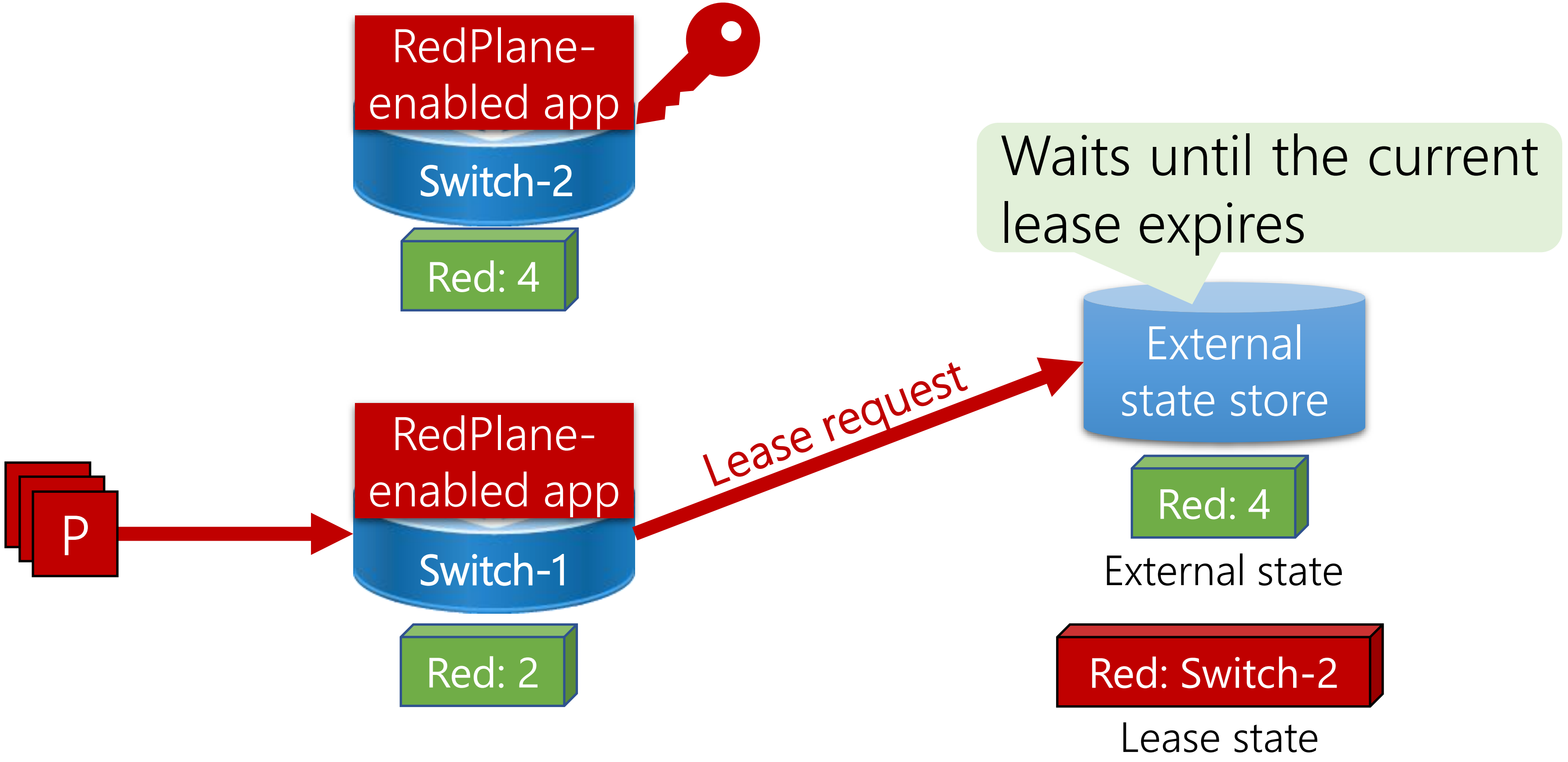
Lease-based state ownership management

Our approach: For a given flow, ensuring only one switch processes packets at a time using leases



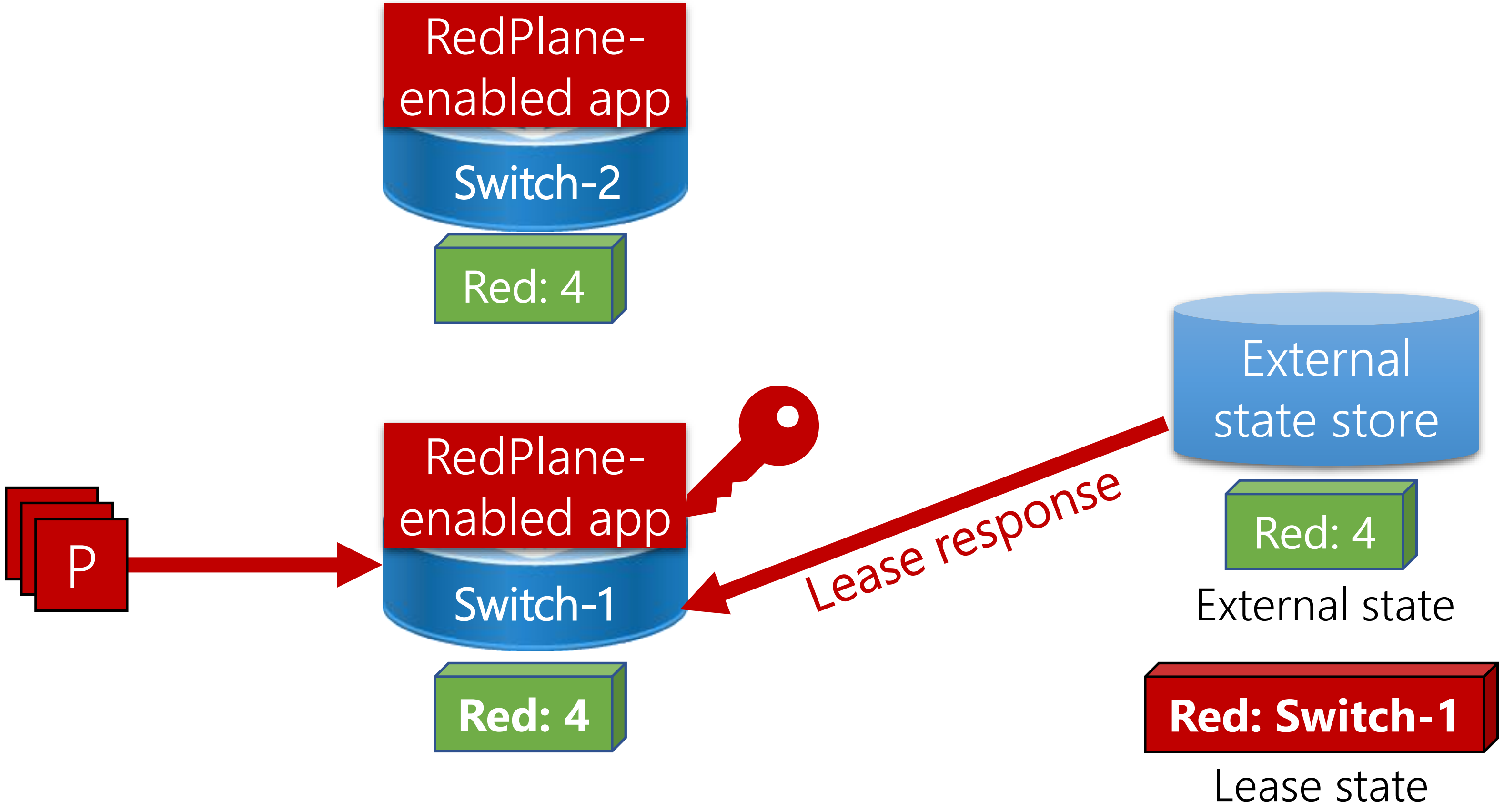
Lease-based state ownership management

Our approach: For a given flow, ensuring only one switch processes packets at a time using leases



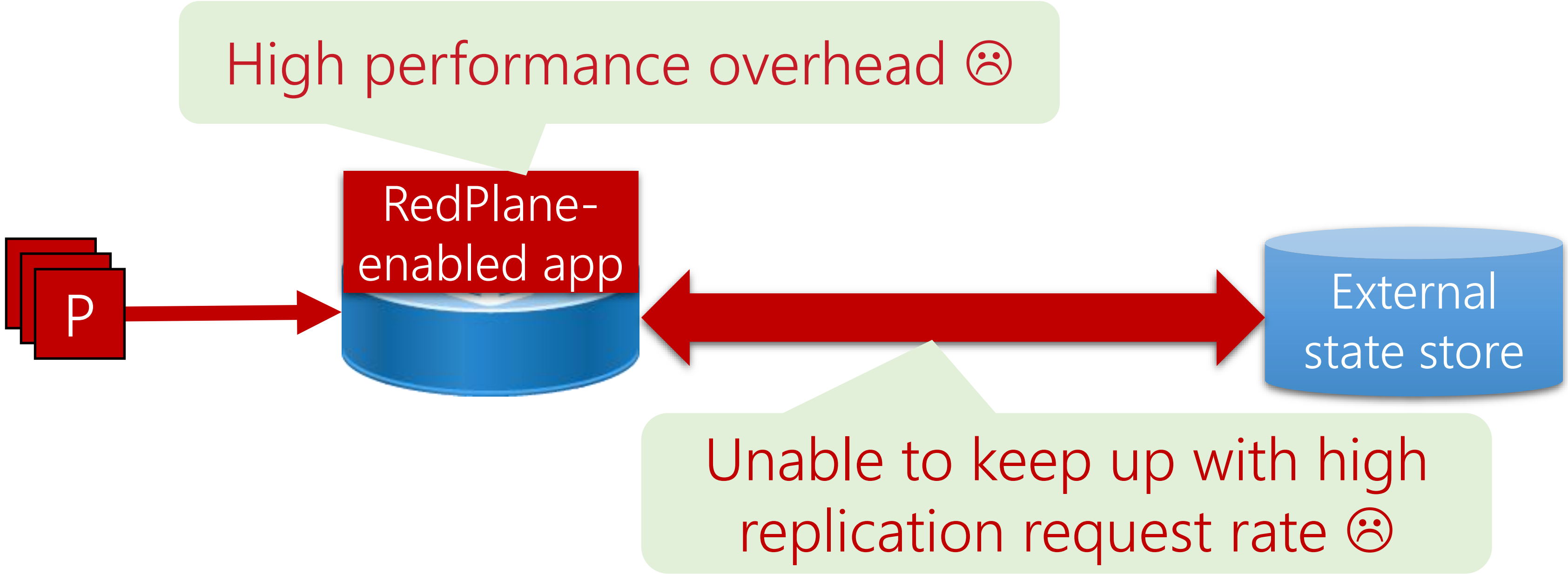
Lease-based state ownership management

Our approach: For a given flow, ensuring only one switch processes packets at a time using leases



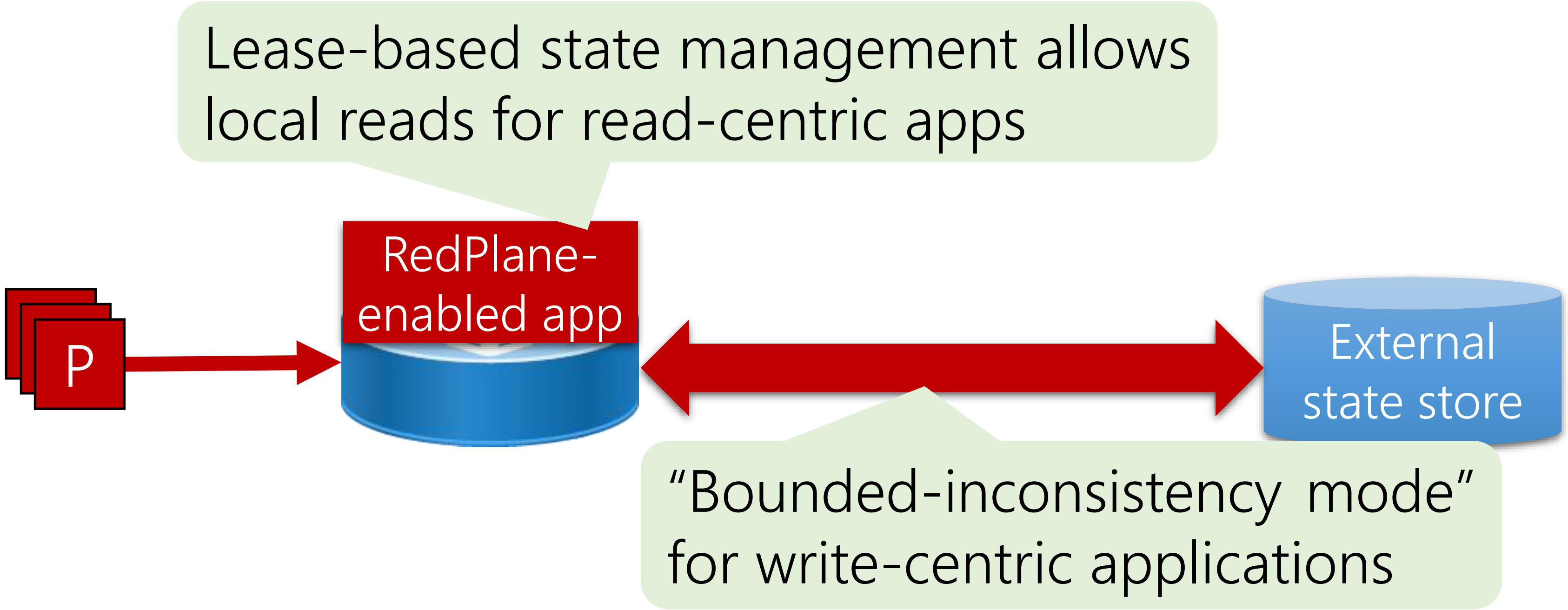
Challenge 3: Handling high traffic volume

Switch data plane operates at up to a few billion packets per second



Challenge 3: Handling high traffic volume

Switch data plane operates at up to a few billion packets per second

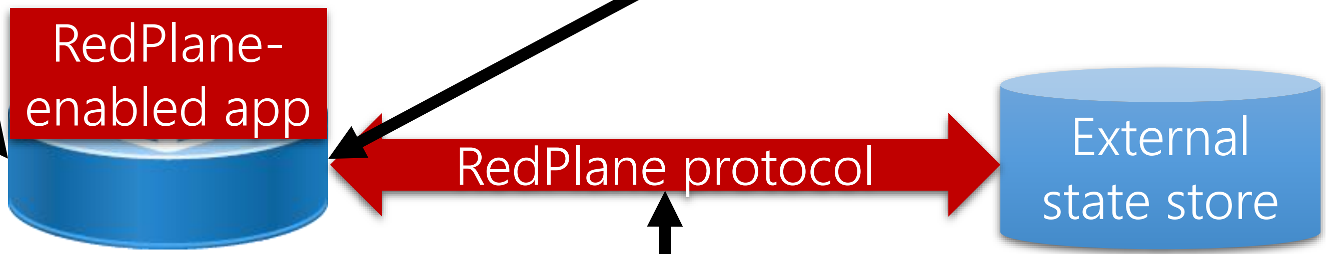


Putting it all together

RedPlane provides **a fault-tolerant state store abstraction** to applications

Sequencing and lightweight retransmission mechanism **(Correctness)**

Lease-based state management **(Routing agnostic, Performance)**



Linearizability-based correctness definition **(Correctness)**
Bounded-inconsistency for write-centric applications **(Correctness, Performance)**

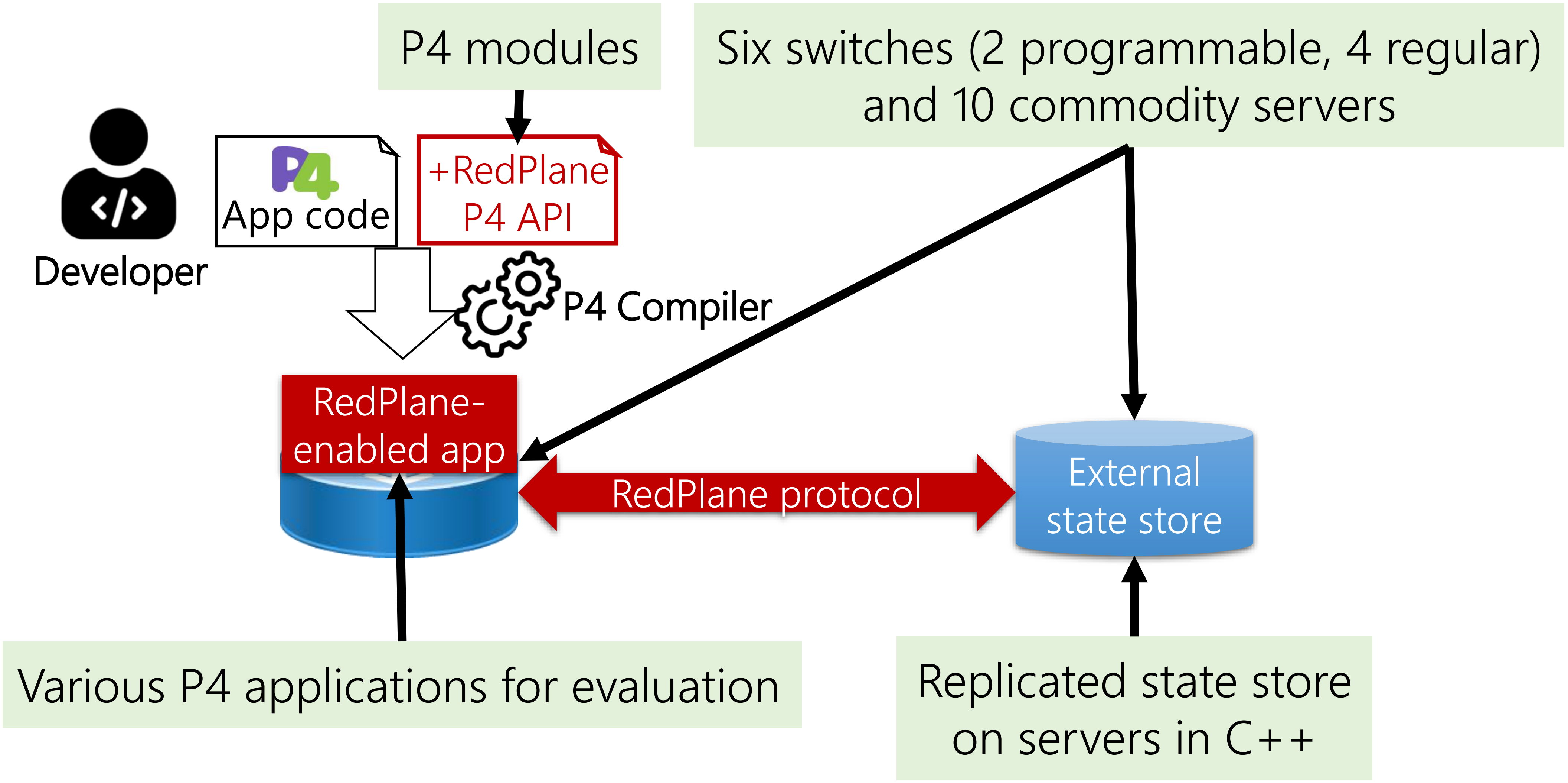
Outline

RedPlane motivation

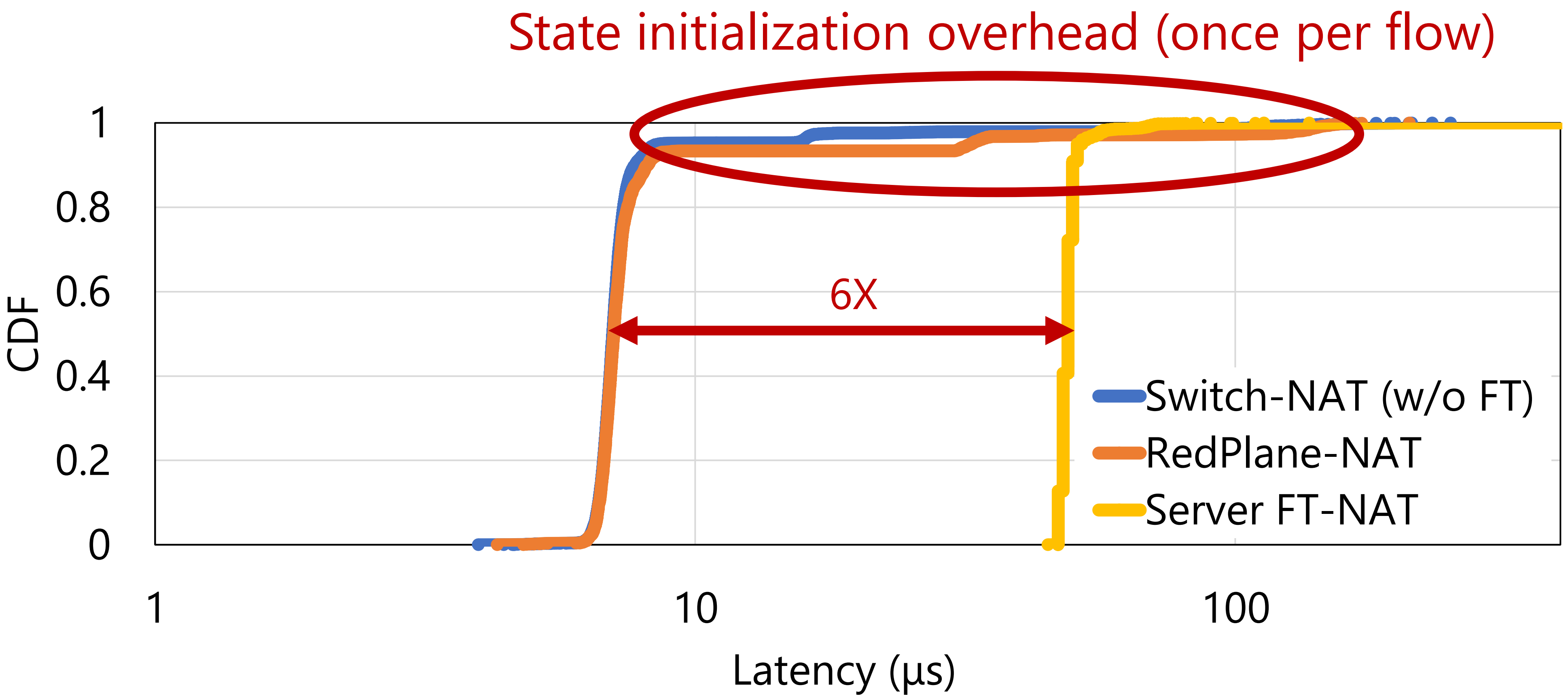
RedPlane design

Results

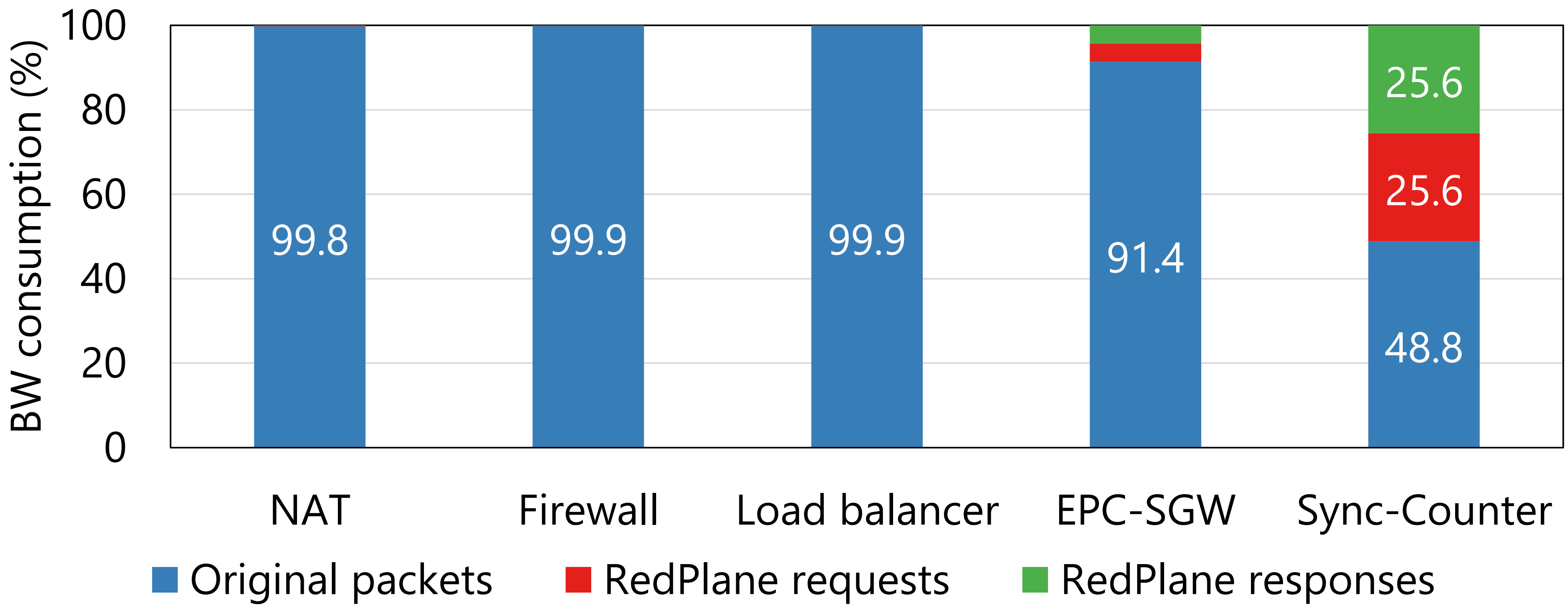
Implementation



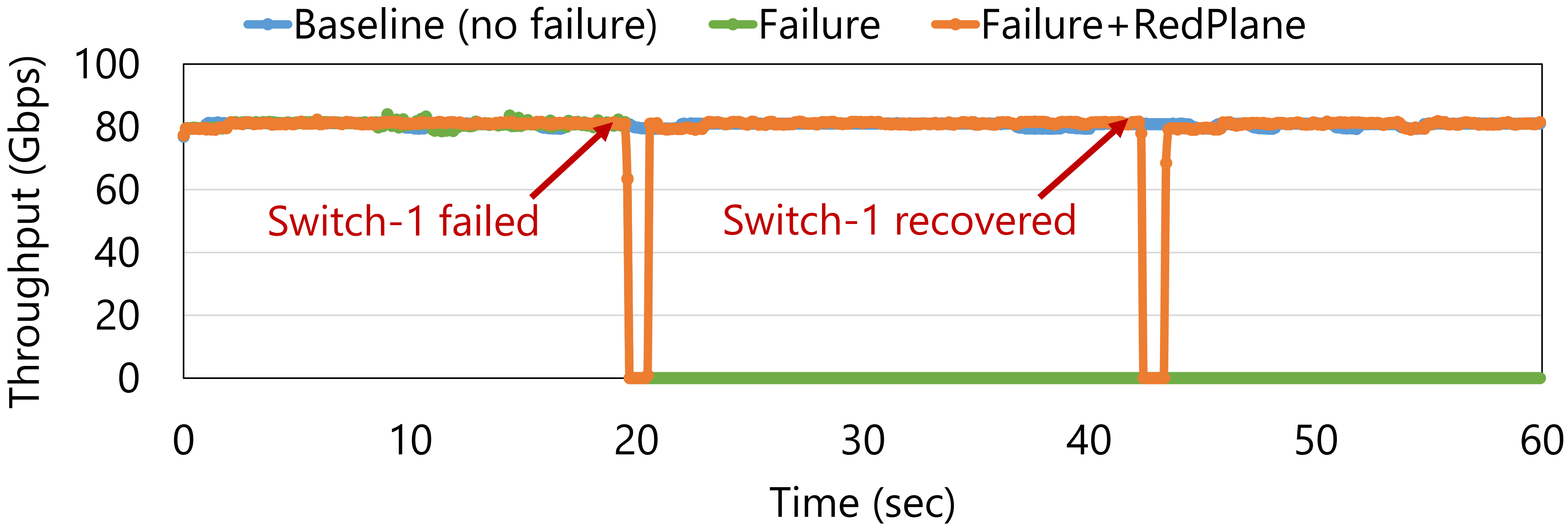
How does RedPlane affect application latency?



How much BW overhead does RedPlane add?



How fast the connectivity can be recovered?



Other results

Throughput of RedPlane-enabled applications

Low switch resource overhead of reliable replication protocol

Less than 13% of switch ASIC resource usage

Model checking for RedPlane protocol by using TLA+

Future directions

Better support for write-centric apps

Supporting non-partitionable states

Automatically enabling fault-tolerance with compiler/language support

Next generation switch architectures for fault-tolerance

Conclusions

Switch failures can affect the correctness of stateful in-switch apps

RedPlane provides a fault-tolerant state store abstraction

- Linearizability-based practical correctness definition for in-switch apps
- Bounded inconsistency mode for write-centric apps
- Sequencing and lightweight retransmission for reliable replication
- Lease-based state ownership management

Offers fault tolerance with minimal performance and resource overhead

- No per-packet latency overhead for read-centric apps
- End-to-end connectivity is recovered within a second

github.com/daehyeok-kim/redplane-public