# Correctness and Performance for Stateful Chained Network Functions

Junaid Khalid[W,G] and Aditya Akella[W]

*This work does not have any affiliation with Google

# Network Function Virtualization (NFV)

Hardware NF → software NF over commodity server

# Network Function Virtualization (NFV)

Hardware NF → software NF over commodity server

*Intrusion detection system (IDS)*

# Network Function Virtualization (NFV)

Hardware NF → software NF over commodity server

*Intrusion detection system (IDS)*

*Caching proxy*

# Network Function Virtualization (NFV)

Hardware NF → software NF over commodity server

*Intrusion detection system (IDS)*

*Caching proxy*

*Firewall*

# Network Function Virtualization (NFV)

Hardware NF → software NF over commodity server

*Intrusion detection system (IDS)*

*Caching proxy*

*Firewall*

*WAN optimizer*

# Network Function Virtualization (NFV)

Hardware NF → software NF over commodity server

- Enables ***resource consolidation***
- ***Dynamic allocation*** of packet processing
- Adding ***new functionality***

*Intrusion detection system (IDS)*

*Caching proxy*

*Firewall*

*WAN optimizer*

# Network Function Virtualization (NFV)

Hardware NF → software NF over commodity server

- Enables ***resource consolidation***
- ***Dynamic allocation*** of packet processing
- Adding ***new functionality***
- Simplifies ***service chaining***

*Intrusion detection system (IDS)*

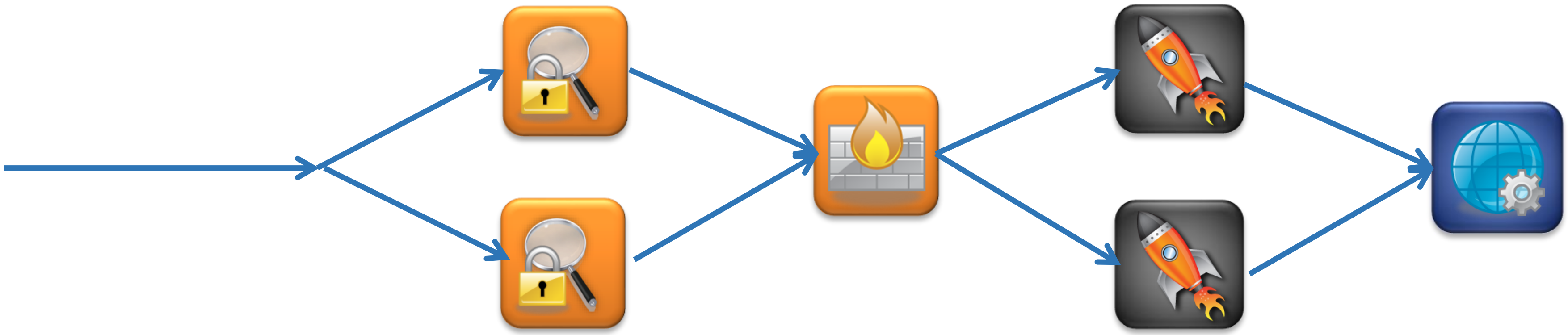
*Caching proxy*

*Firewall*

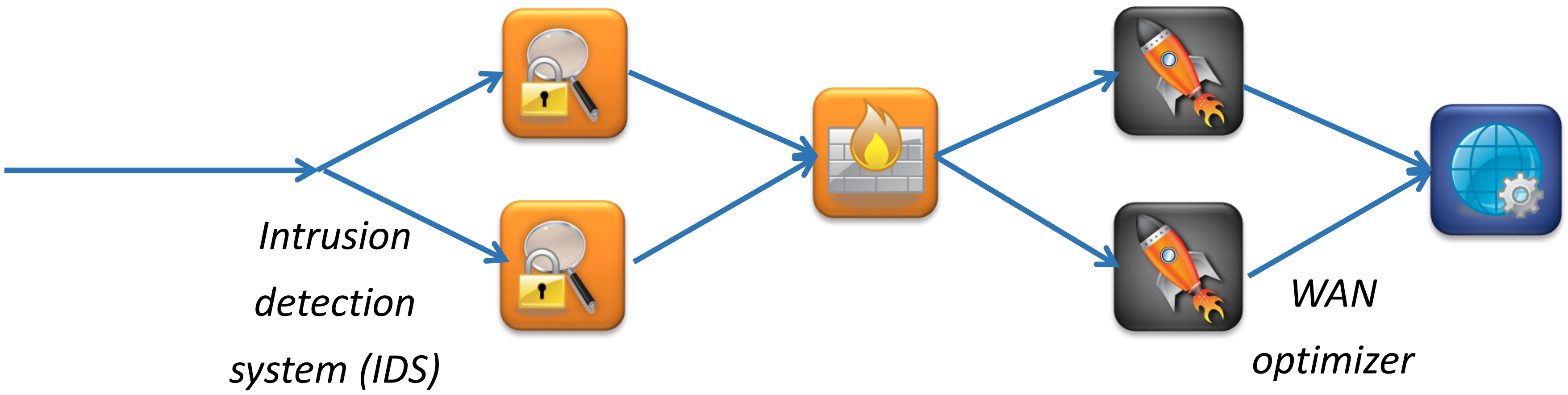
*WAN optimizer*

# Service Chaining

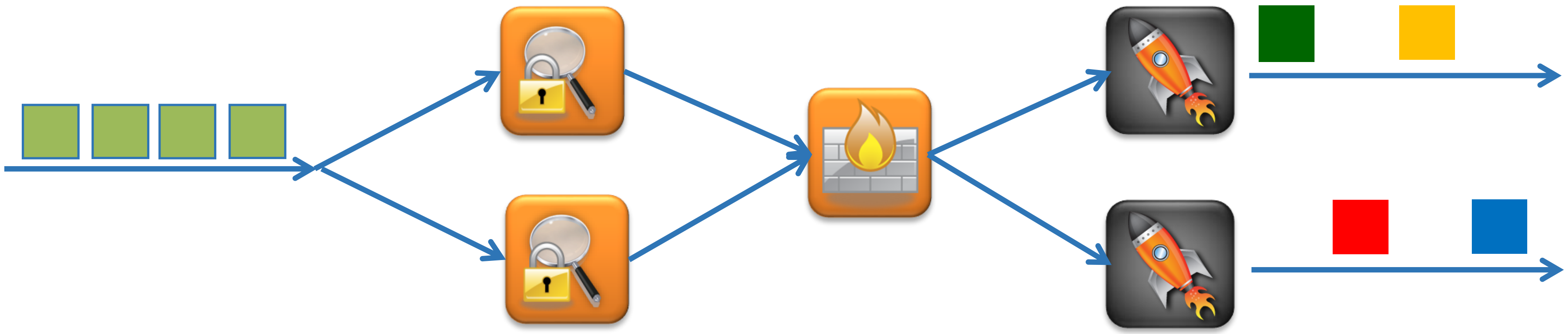
# Service Chaining



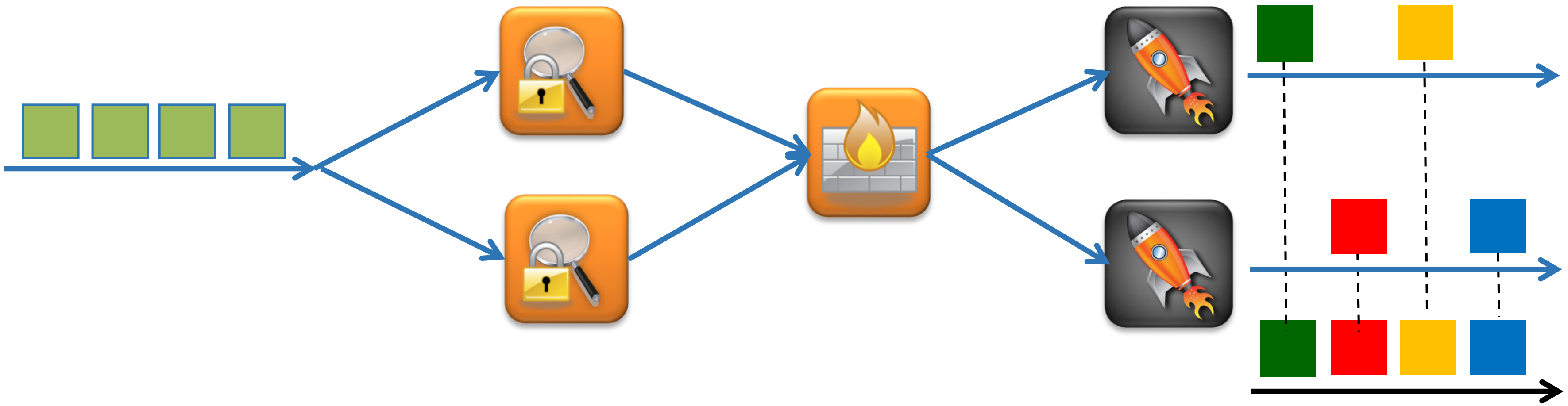
# Service Chaining



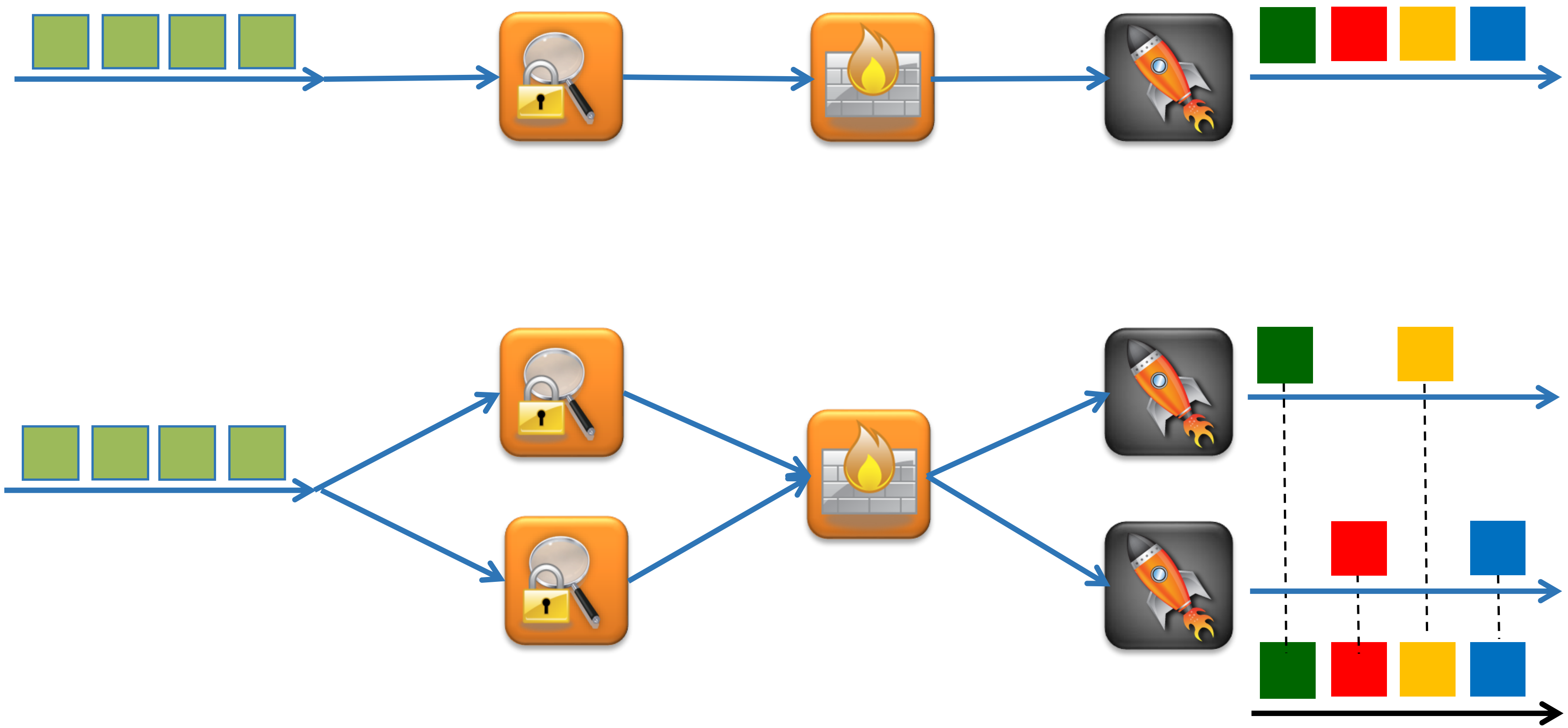
# Service Chaining



# Service Chaining

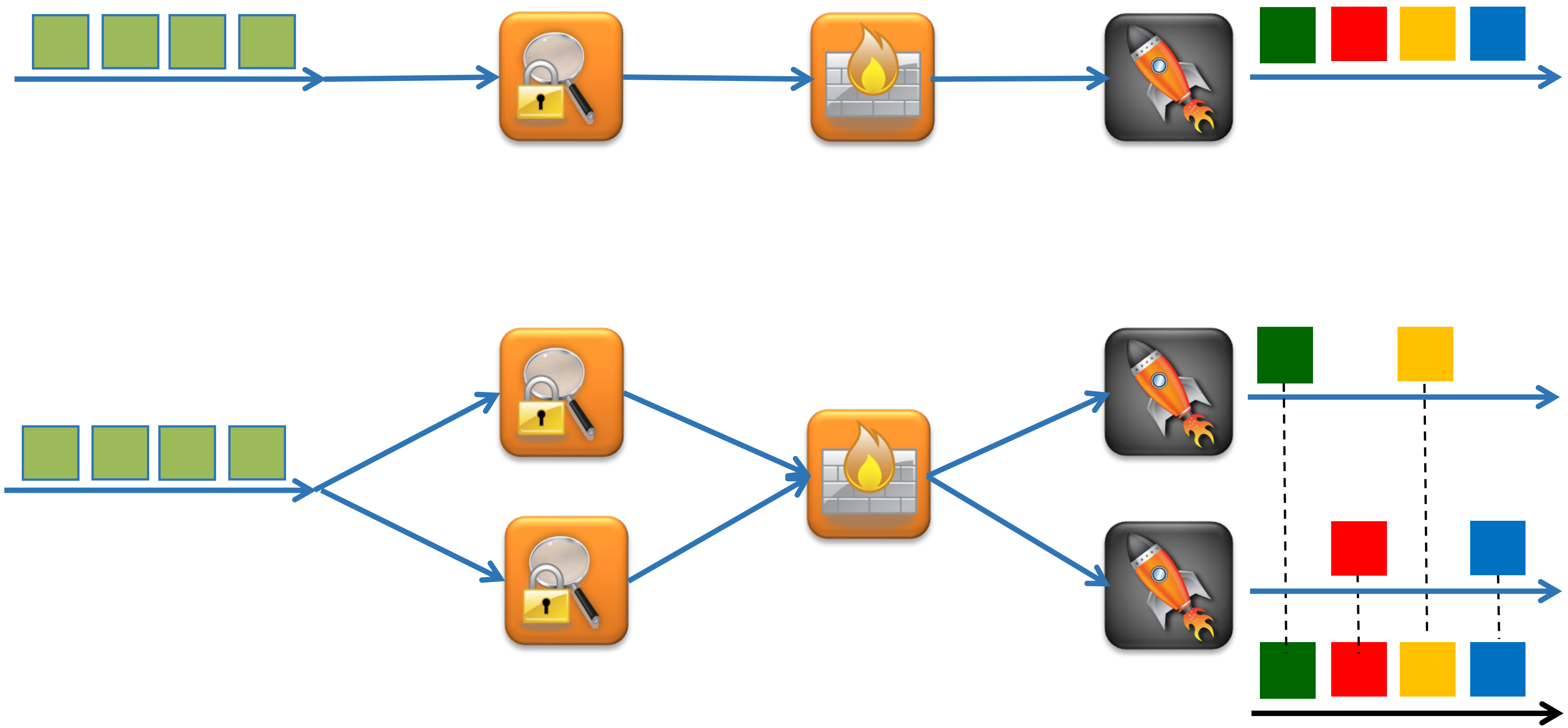


# Service Chaining



# Service Chaining

**Chain output equivalence (COE):** for any input the aggregate output of a dynamic set of instances should be equivalent to the output produced by a single instance



# Service Chaining

**Chain output equivalence (COE):** for any input the aggregate output of a dynamic set of instances should be equivalent to the output produced by a single instance

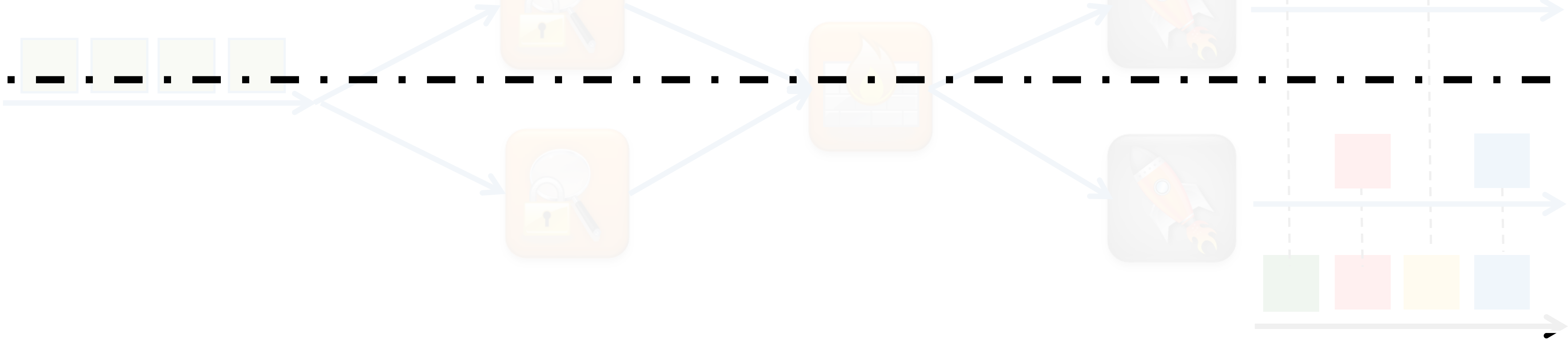
Our goal is to provide COE in service chaining ***without compromising*** ***performance*** or ***correctness***

# Service Chaining

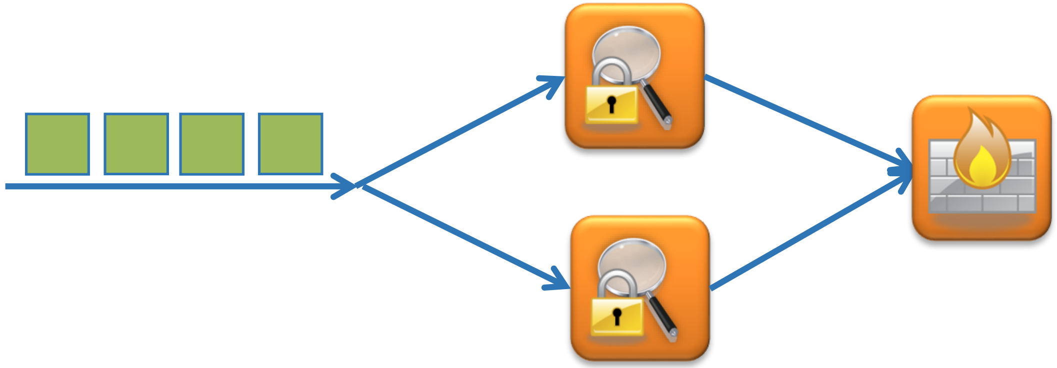
**Chain output equivalence (COE):** for any input the aggregate output of a dynamic set of instances should be equivalent to the output produced by a single instance

Our goal is to provide COE in service chaining ***without compromising*** ***performance*** or ***correctness***

Ensuring COE is challenging: ***NF chain attributes & Dynamic Actions***



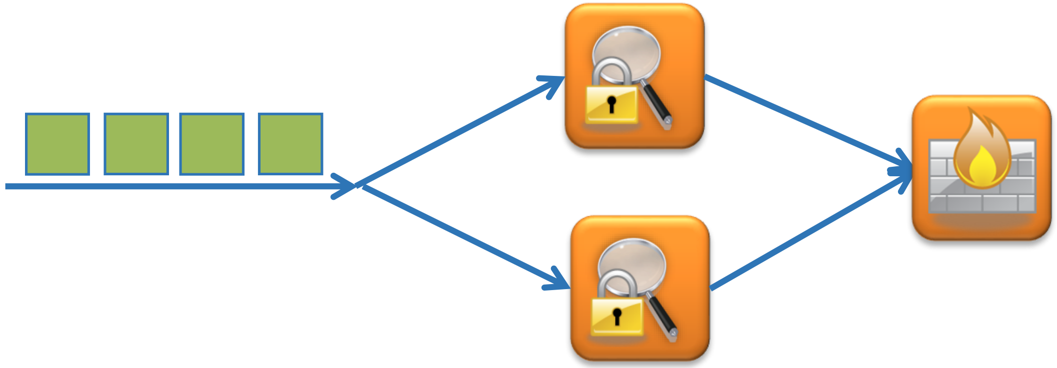
# NF Chain Attributes



# NF Chain Attributes

1. NF statefulness

- Perform sophisticated *stateful* actions on packets/flows

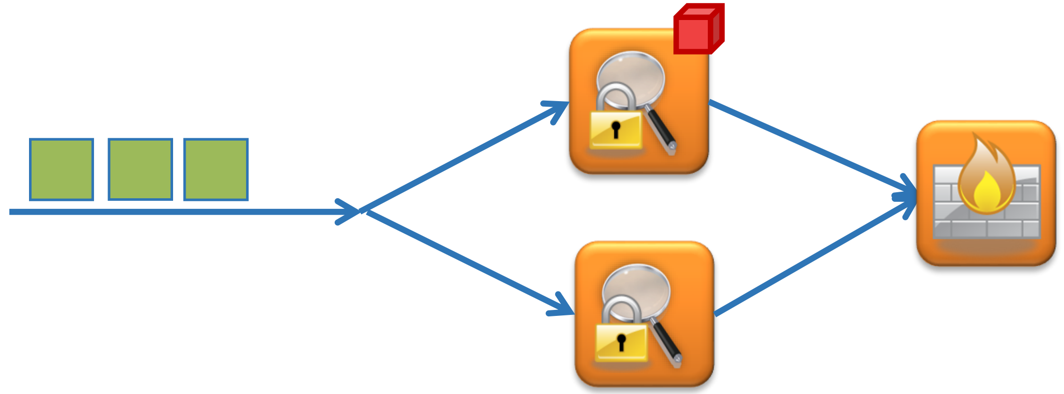


# NF Chain Attributes

1. NF statefulness

- Perform sophisticated *stateful* actions on packets/flows

IDS maintains ***cross-flows state*** (e.g., per host active conn. count) and ***per-flow state*** (e.g., TCP conn. state)

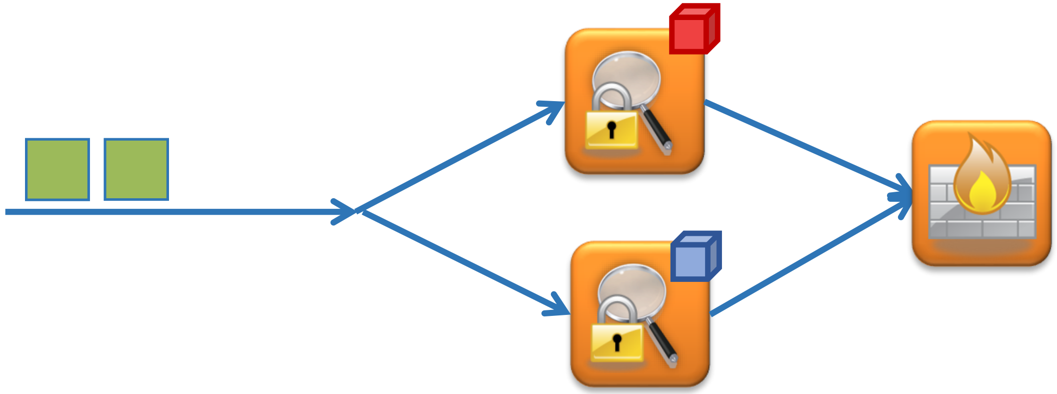


# NF Chain Attributes

1. NF statefulness

- Perform sophisticated *stateful* actions on packets/flows

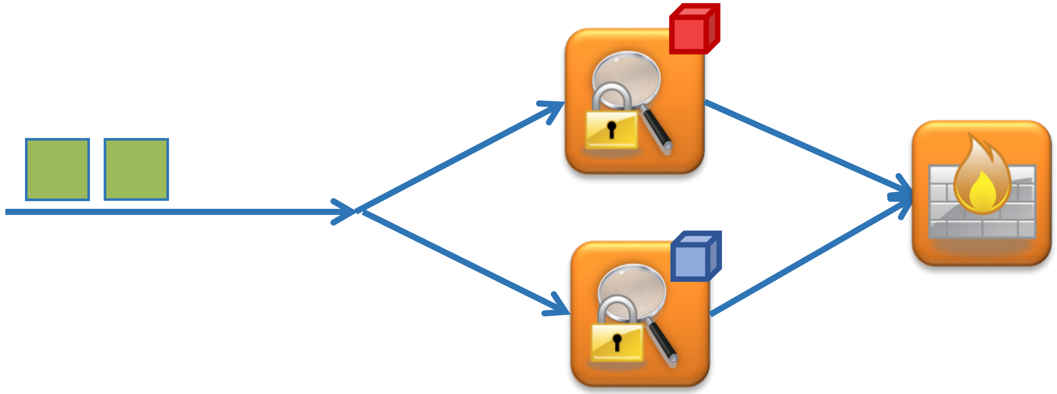
IDS maintains ***cross-flows state*** (e.g., per host active conn. count) and ***per-flow state*** (e.g., TCP conn. state)



# NF Chain Attributes

1. NF statefulness

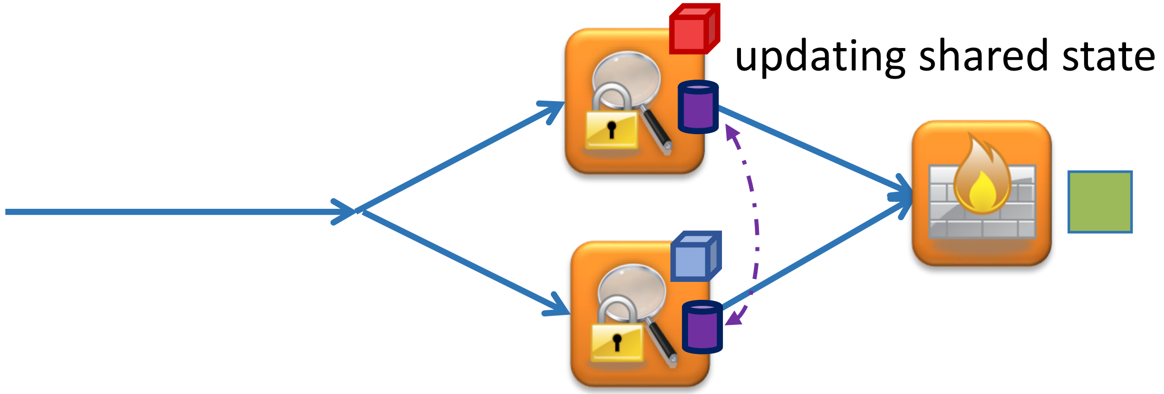
- Perform sophisticated *stateful* actions on packets/flows



# NF Chain Attributes

1. NF statefulness
2. Consistent state updates

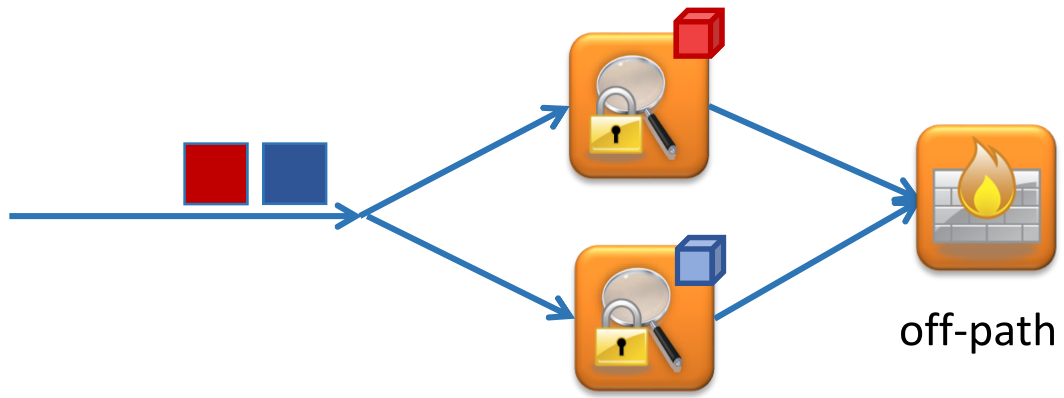
- Perform sophisticated *stateful* actions on packets/flows
- Action taken by an NF instance depends on the state updates from other NF instances



# NF Chain Attributes

1. NF statefulness
2. Consistent state updates

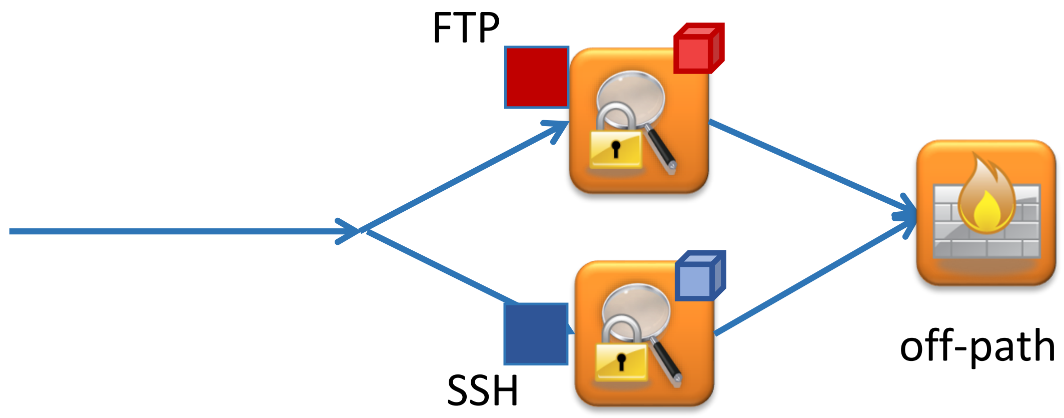
- Perform sophisticated *stateful* actions on packets/flows
- Action taken by an NF instance depends on the state updates from other NF instances



# NF Chain Attributes

1. NF statefulness
2. Consistent state updates
3. Dependency between different NF instances

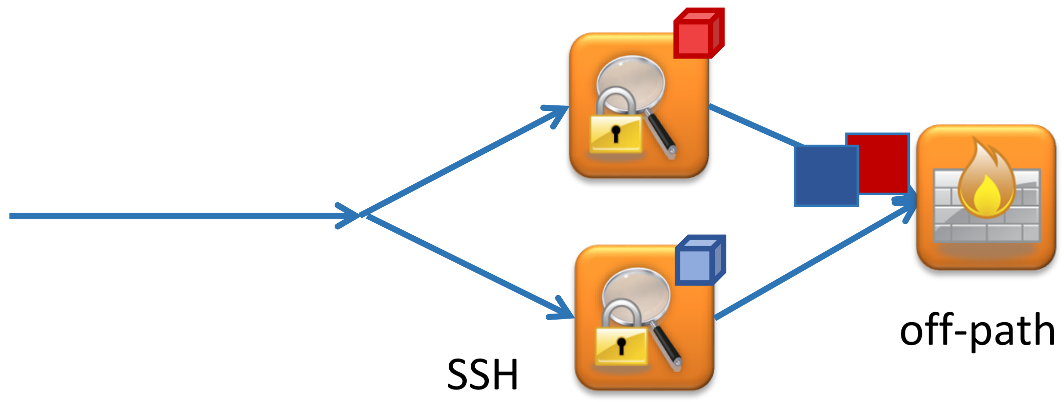
- Perform sophisticated *stateful* actions on packets/flows
- Action taken by an NF instance depends on the state updates from other NF instances
- Action at the downstream NF may depend on the upstream NFs



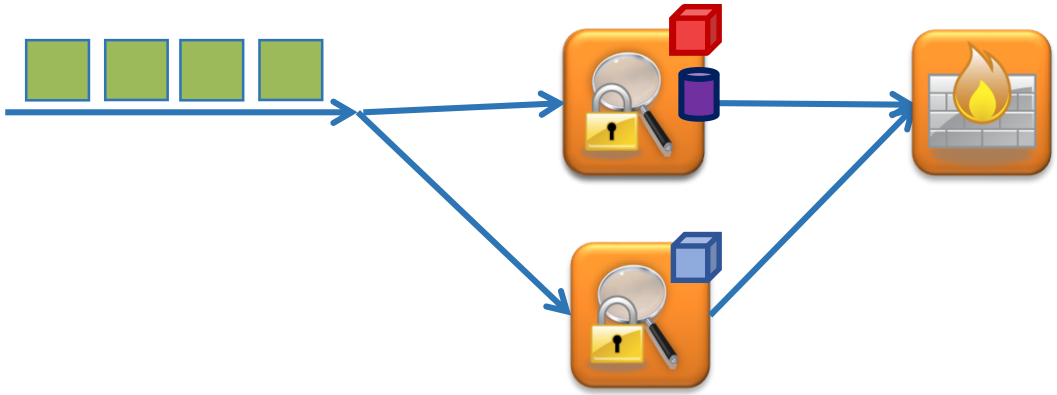
# NF Chain Attributes

1. NF statefulness
2. Consistent state updates
3. Dependency between different NF instances

- Perform sophisticated *stateful* actions on packets/flows
- Action taken by an NF instance depends on the state updates from other NF instances
- Action at the downstream NF may depend on the upstream NFs



# Dynamic Actions

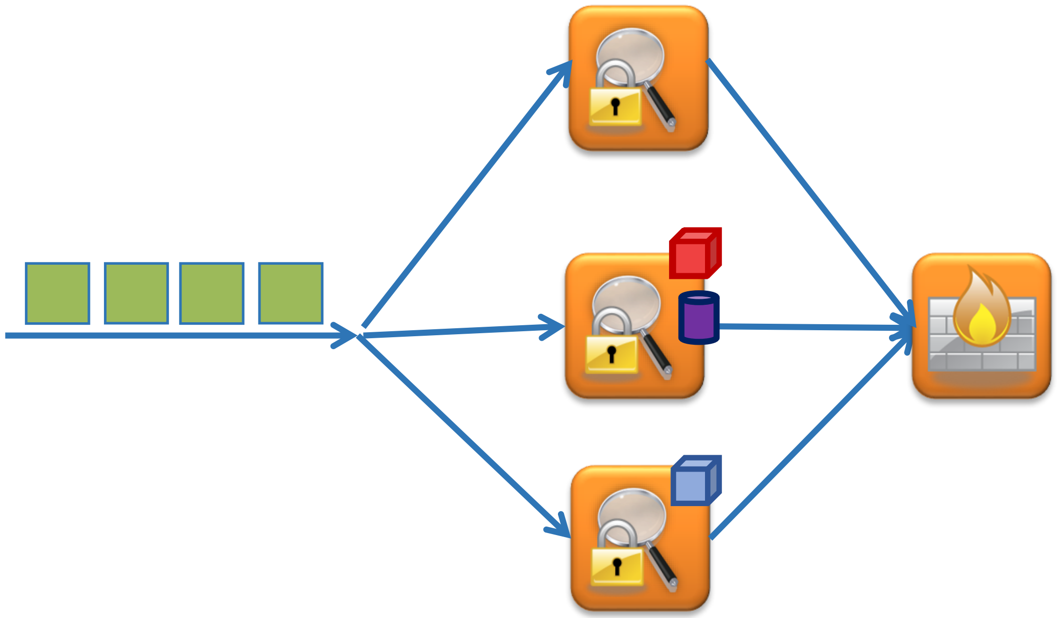


**Key requirements**

# Dynamic Actions

## Load balancing/elastic scaling

- Flows are moved from one instance to another to balance load or handle traffic spikes

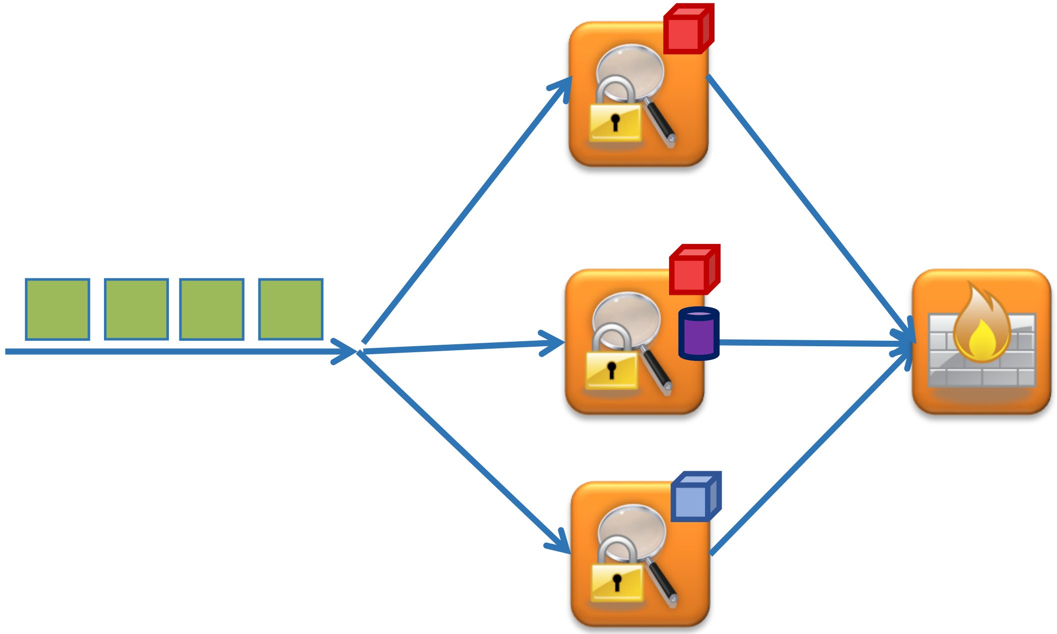


**Key requirements**

# Dynamic Actions

## Load balancing/elastic scaling

- Flows are moved from one instance to another to balance load or handle traffic spikes



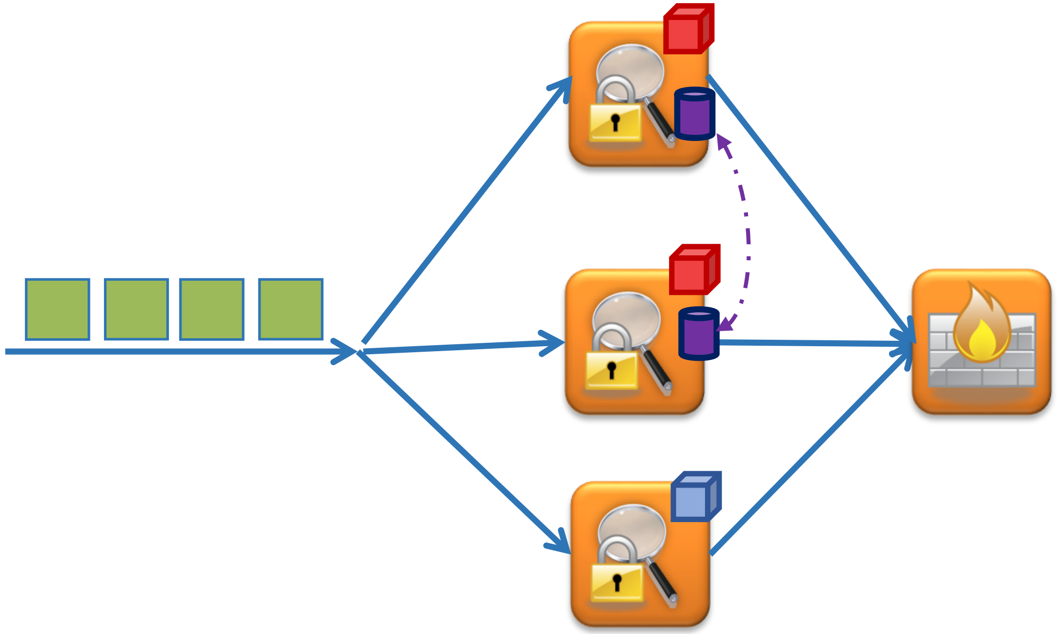
**Key requirements**

- Safe cross-instance state transfer

# Dynamic Actions

## Load balancing/elastic scaling

- Flows are moved from one instance to another to balance load or handle traffic spikes



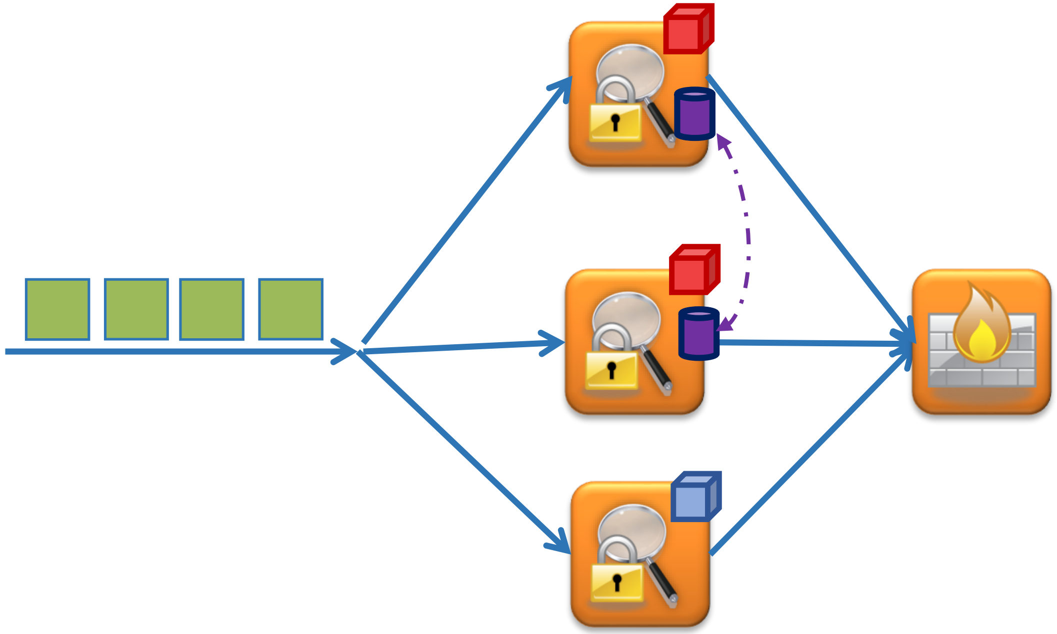
### Key requirements

- Safe cross-instance state transfer

# Dynamic Actions

## Load balancing/elastic scaling

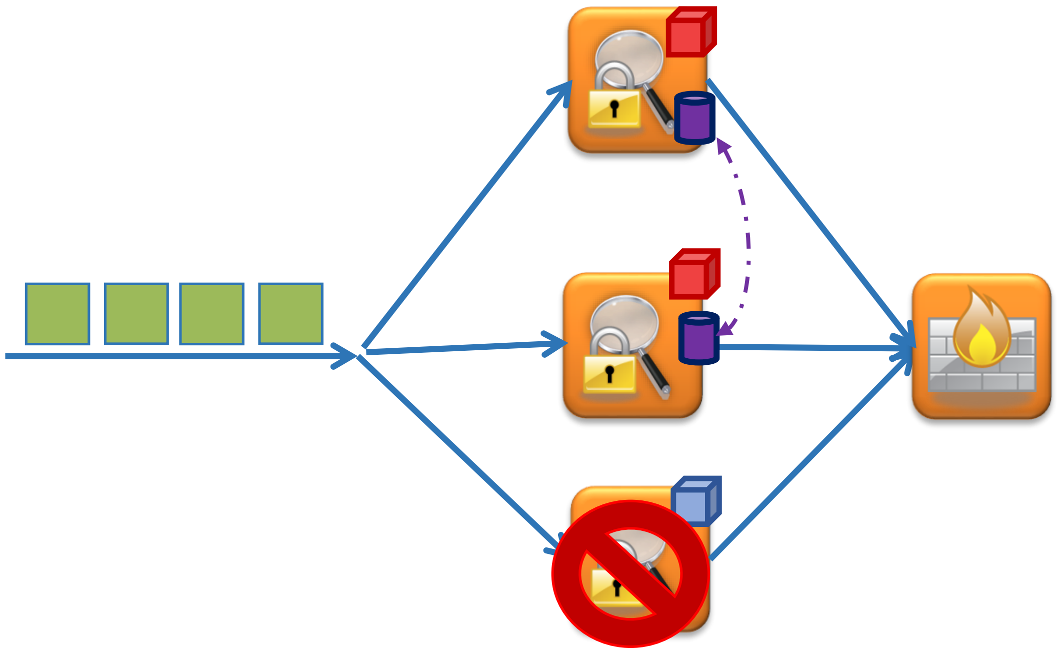
- Flows are moved from one instance to another to balance load or handle traffic spikes



## Key requirements

- Safe cross-instance state transfer
- Consistent shared state

# Dynamic Actions



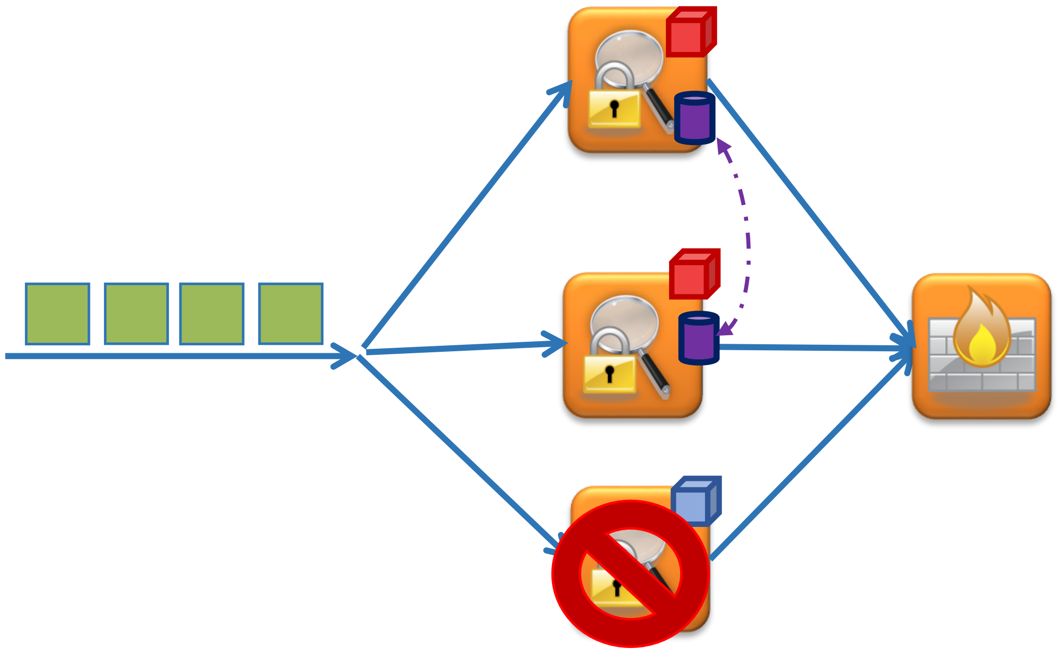
**Key requirements**

- Safe cross-instance state transfer
- Consistent shared state

# Dynamic Actions

## Failure recovery

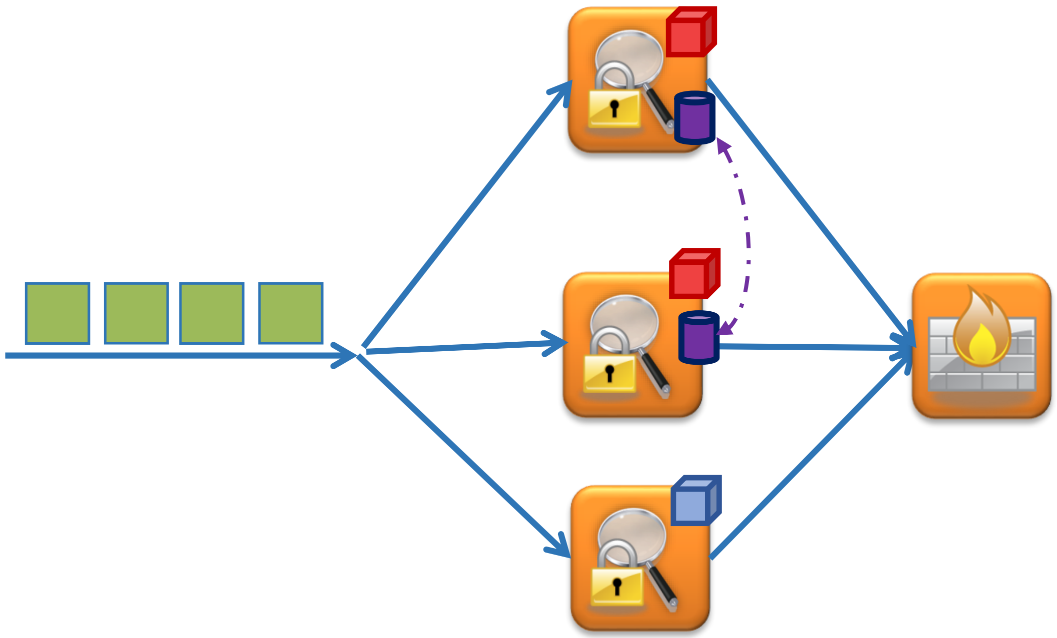
- When NF fails, all its state disappears. For fault tolerance, that state needs to be recovered



**Key requirements**

- Safe cross-instance state transfer
- Consistent shared state
- State availability

# Dynamic Actions



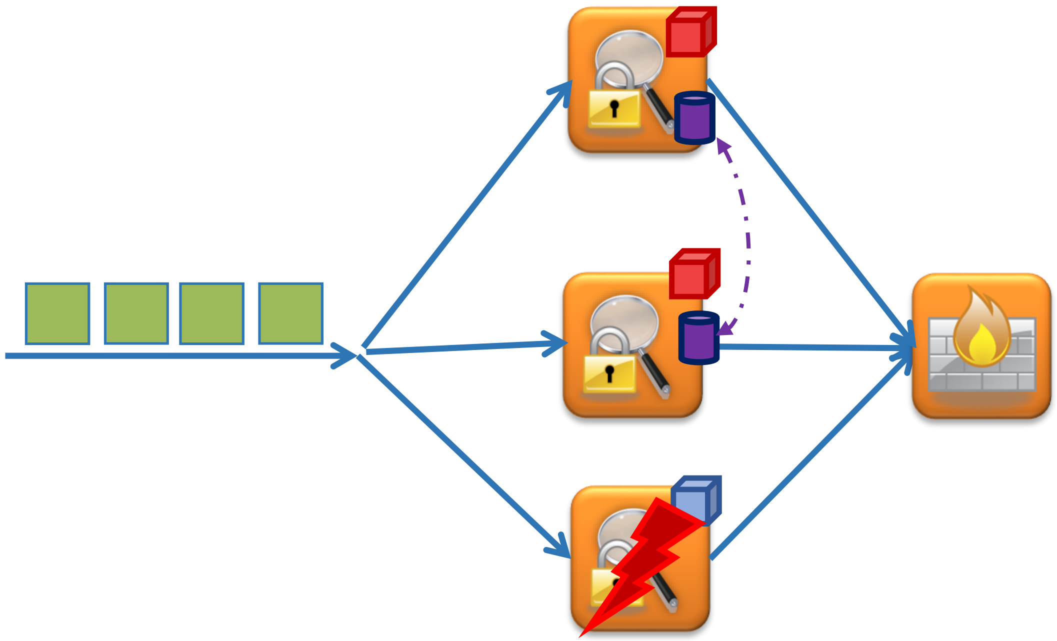
**Key requirements**

- Safe cross-instance state transfer
- Consistent shared state
- State availability

# Dynamic Actions

## Instance slowdown

- Clones may be launched to handle a straggler NF (a slow NF)



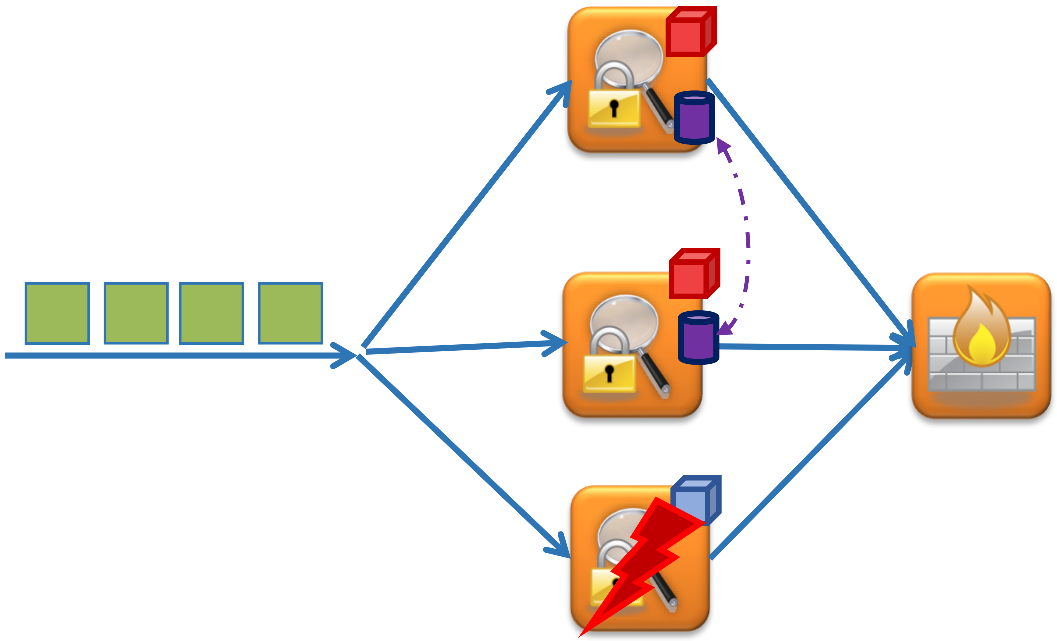
## Key requirements

- Safe cross-instance state transfer
- Consistent shared state
- State availability
- Duplicate suppression

# Dynamic Actions

## Instance slowdown

- Clones may be launched to handle a straggler NF (a slow NF)
- Downstream NFs rely on the order at upstream NFs



## Key requirements

- Safe cross-instance state transfer
- Consistent shared state
- State availability
- Duplicate suppression
- Chain-wide ordering

# Key Requirements for COE

# Key Requirements for COE

## NF chain attributes

- NF statefulness
- Consistent state updates
- Dependency between different NF instances

# Key Requirements for COE

## NF chain attributes

- NF statefulness
- Consistent state updates
- Dependency between different NF instances

X

## Dynamic actions

- Elastic scaling
- Failure recovery
- Instance slowdown

# Key Requirements for COE

## NF chain attributes

- NF statefulness
- Consistent state updates
- Dependency between different NF instances

**X**

## Dynamic actions

- Elastic scaling
- Failure recovery
- Instance slowdown

**=**

## Key requirements

- Safe cross-instance state transfer
- Consistent shared state
- State availability
- Duplicate suppression
- Chain-wide ordering

# Existing Solutions

| Framework | State availability | State transfer | Consistent shared state | Duplicate suppression | Chain-wide ordering |
|---|---|---|---|---|---|
| Split/Merge[NSDI'13] | | ✓ | | | |
| OpenNF[SIGCOMM'14] | | ✓ | ✓ | | |
| FTMB [SIGCOMM' 15] | ✓ | | | | |
| S6 [NSDI'18] | | | ✓ | | |
| Pico Rep.[SOCC'13] | ✓ | | | | |
| StatelessNF[NSDI'17] | ✓ | ✓ | | | |

# Existing Solutions

| Framework | State availability | State transfer | Consistent shared state | Duplicate suppression | Chain-wide ordering |
|---|---|---|---|---|---|
| Split/Merge[NSDI'13] | | ✓ | | | |
| OpenNF[SIGCOMM'14] | | ✓ | ✓ | | |
| FTMB [SIGCOMM' 15] | ✓ | | | | |
| S6 [NSDI'18] | | | ✓ | | |
| Pico Rep.[SOCC'13] | ✓ | | | | |
| StatelessNF[NSDI'17] | ✓ | ✓ | | | |

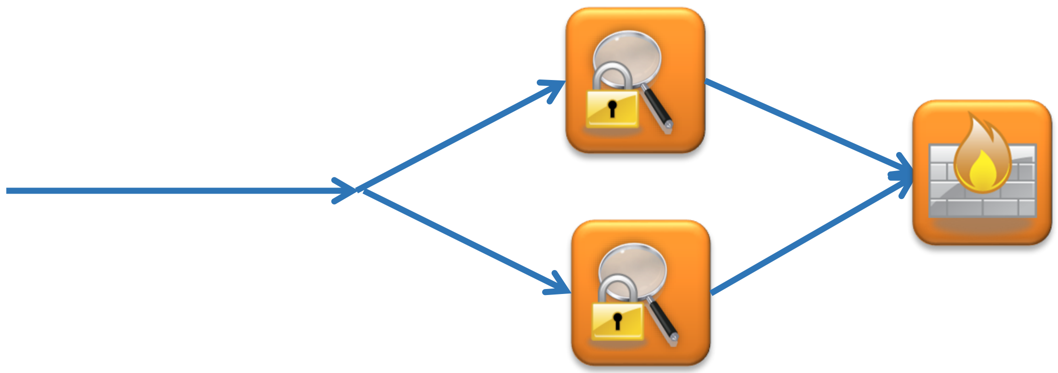
Incomplete support → restricted functionality

# CHC

CHC is a generic NFV framework to support all of these requirements without trading off ***correctness*** for ***performance*** or ***functionality***

# CHC

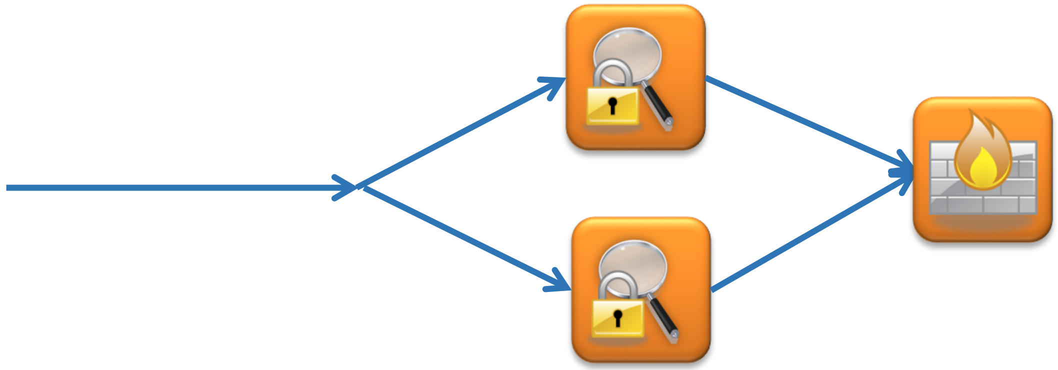
CHC is a generic NFV framework to support all of these requirements without trading off ***correctness*** for ***performance*** or ***functionality***



# CHC

CHC is a generic NFV framework to support all of these requirements without trading off ***correctness*** for ***performance*** or ***functionality***

CHC consist of three main  building blocks

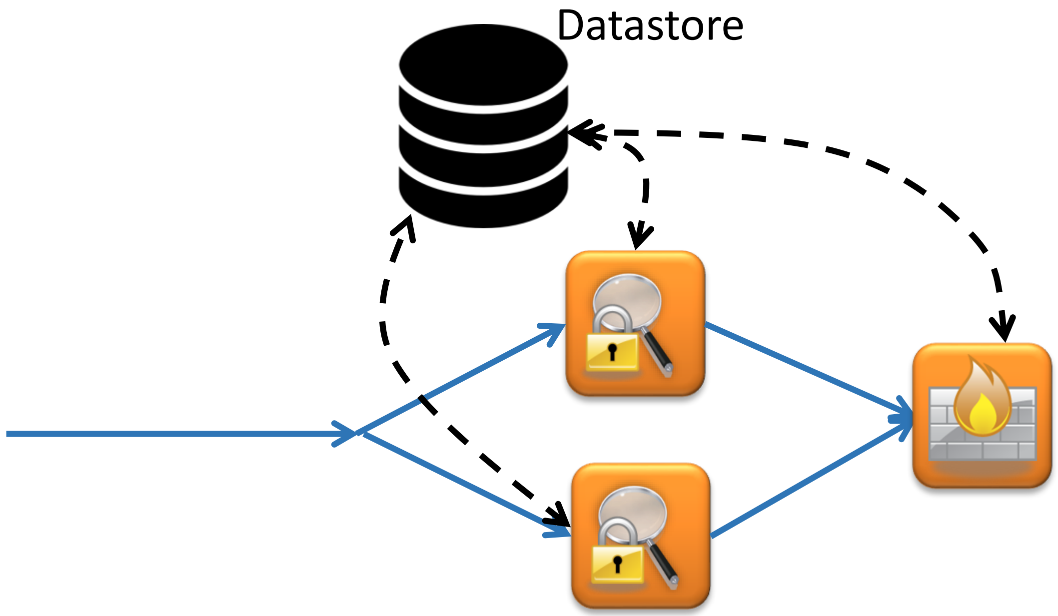


# CHC

CHC is a generic NFV framework to support all of these requirements without trading off ***correctness*** for ***performance*** or ***functionality***

CHC consist of three main  building blocks

1. State store external to NF

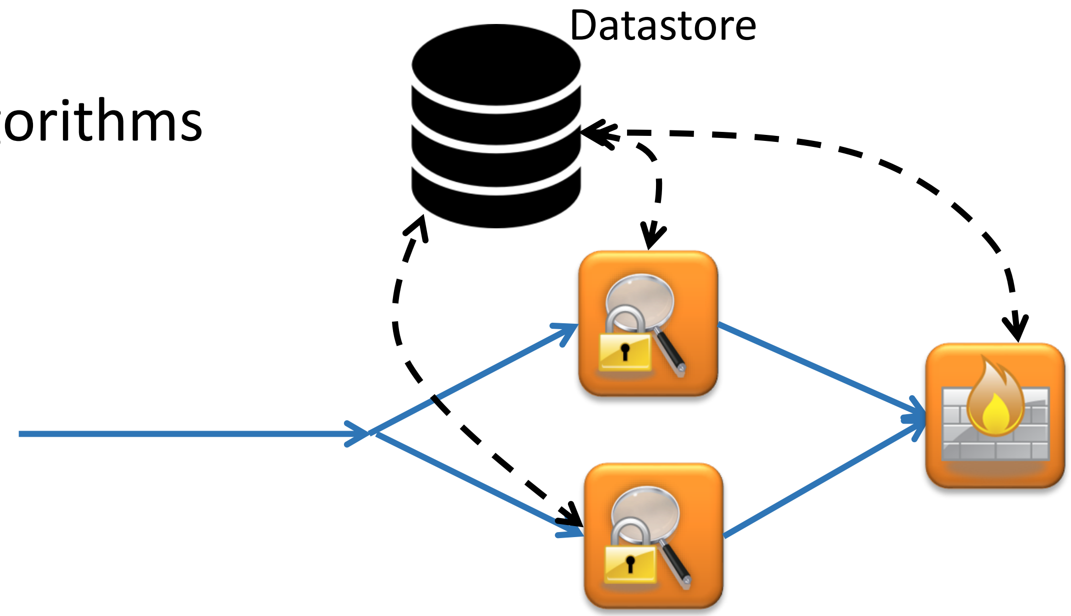


# CHC

CHC is a generic NFV framework to support all of these requirements without trading off ***correctness*** for ***performance*** or ***functionality***

CHC consist of three main  building blocks

1. State store external to NF
2. NF state-aware state management algorithms

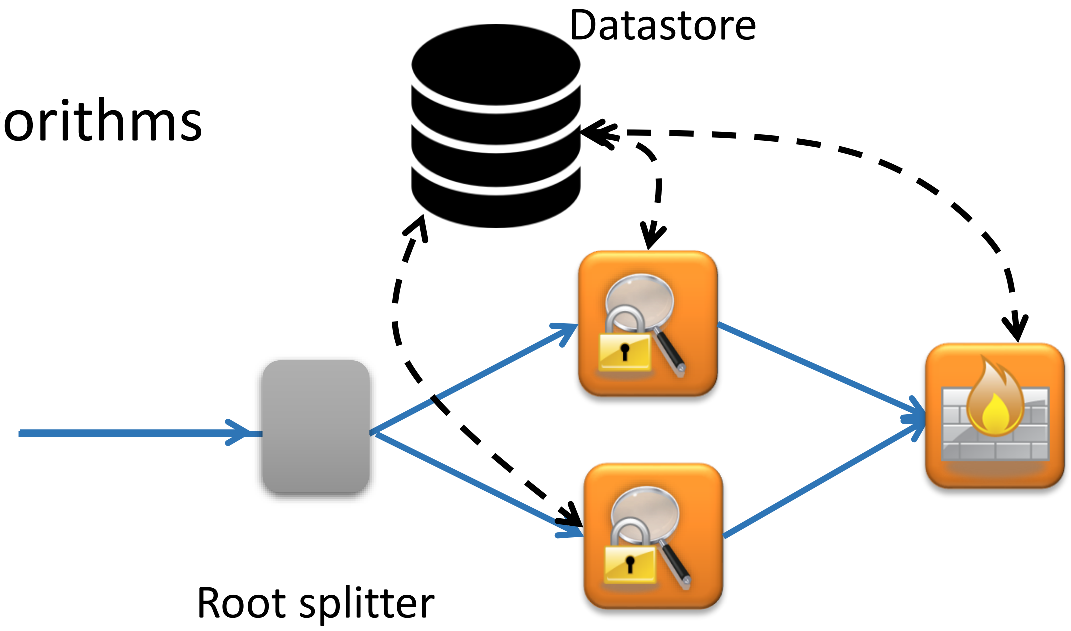


# CHC

CHC is a generic NFV framework to support all of these requirements without trading off ***correctness*** for ***performance*** or ***functionality***

CHC consist of three main  building blocks

1. State store external to NF
2. NF state-aware state management algorithms
3. Metadata – logical clock and logs



# CHC

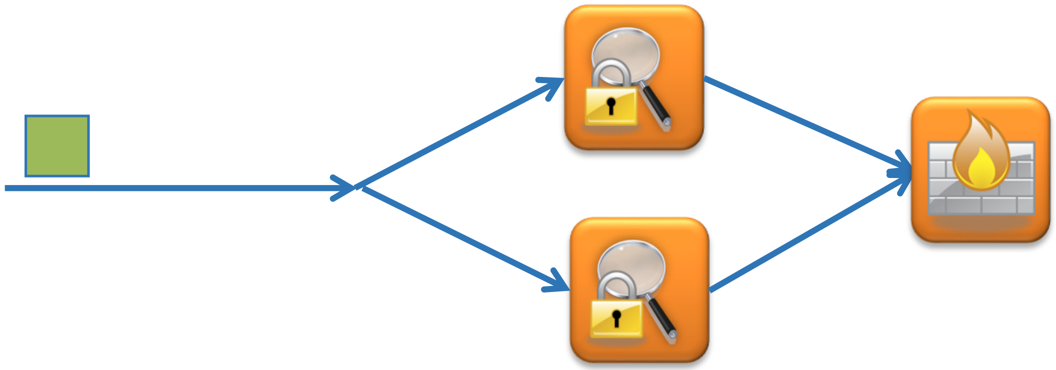
CHC is a generic NFV framework to support all of these requirements without trading off ***correctness*** for ***performance*** or ***functionality***

CHC consist of three main  building blocks

1. State store external to NF
2. NF state-aware state management algorithms
3. Metadata – logical clock and logs

# CHC – State Externalization

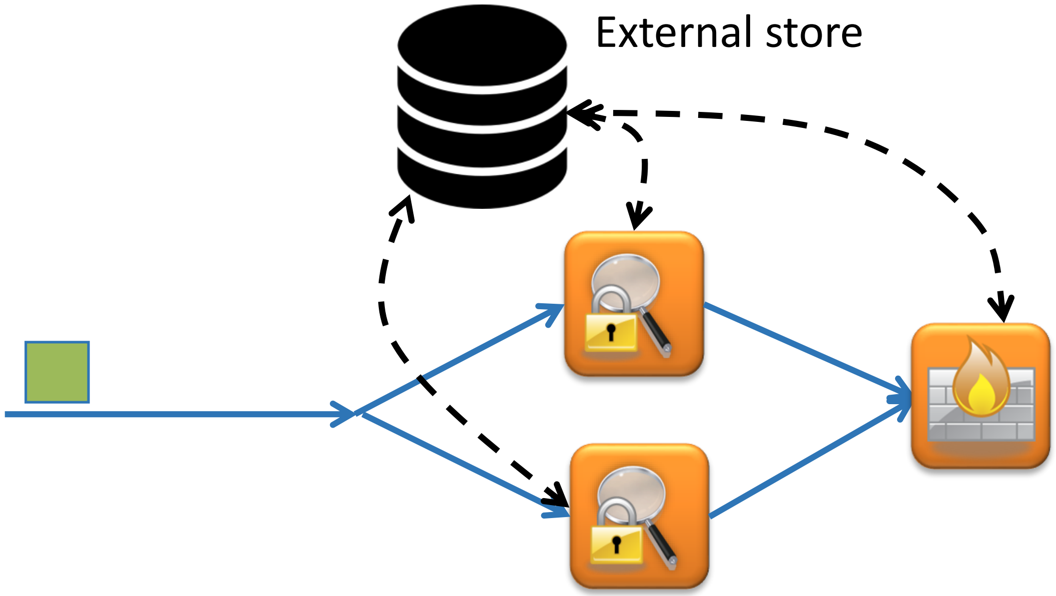
NF state is stored in an in-memory external state store (similar to statelessNF)



# CHC – State Externalization

NF state is stored in an in-memory external state store (similar to statelessNF)

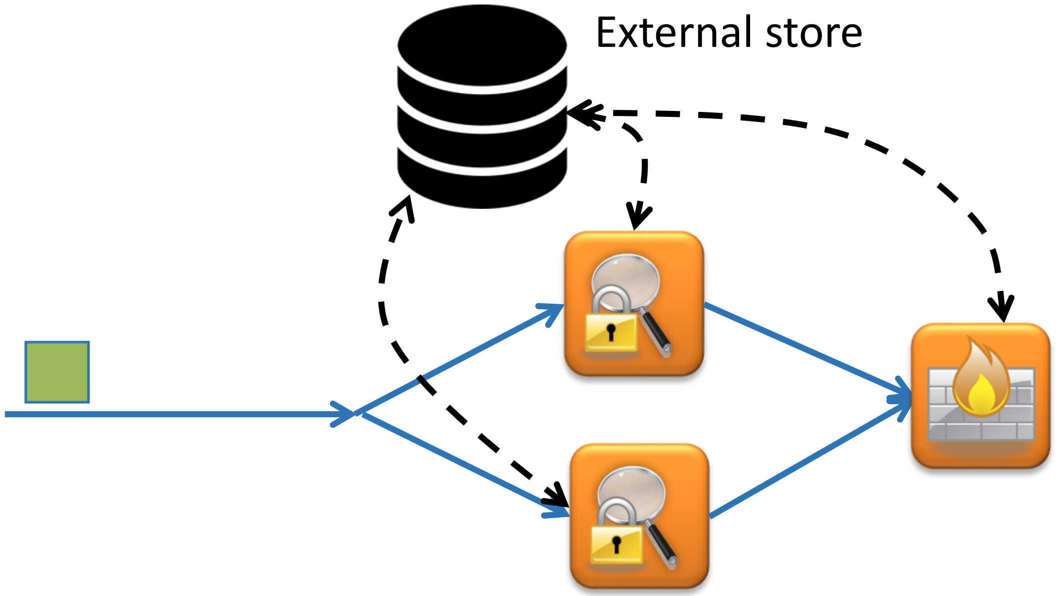
- This ensures ***state availability*** and simplifies reasoning about ***state ownership*** and ***concurrency control*** across instances



# CHC – State Externalization

NF state is stored in an in-memory external state store (similar to statelessNF)

- This ensures ***state availability*** and simplifies reasoning about ***state ownership*** and ***concurrency control*** across instances




Naively ***externalizing*** the state can ***degrade*** NF performance

# CHC

CHC is a generic NFV framework to support all of these requirements without trading off ***correctness*** for ***performance*** or ***functionality***

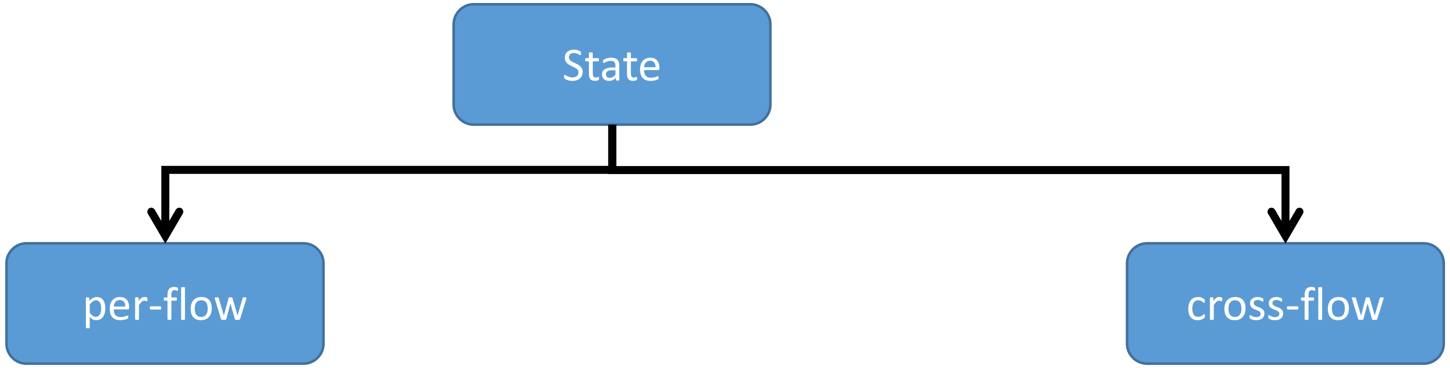
CHC consist of three main  building blocks

1. State store external to NF
2. NF state-aware state management algorithms
3. Metadata – logical clock and logs

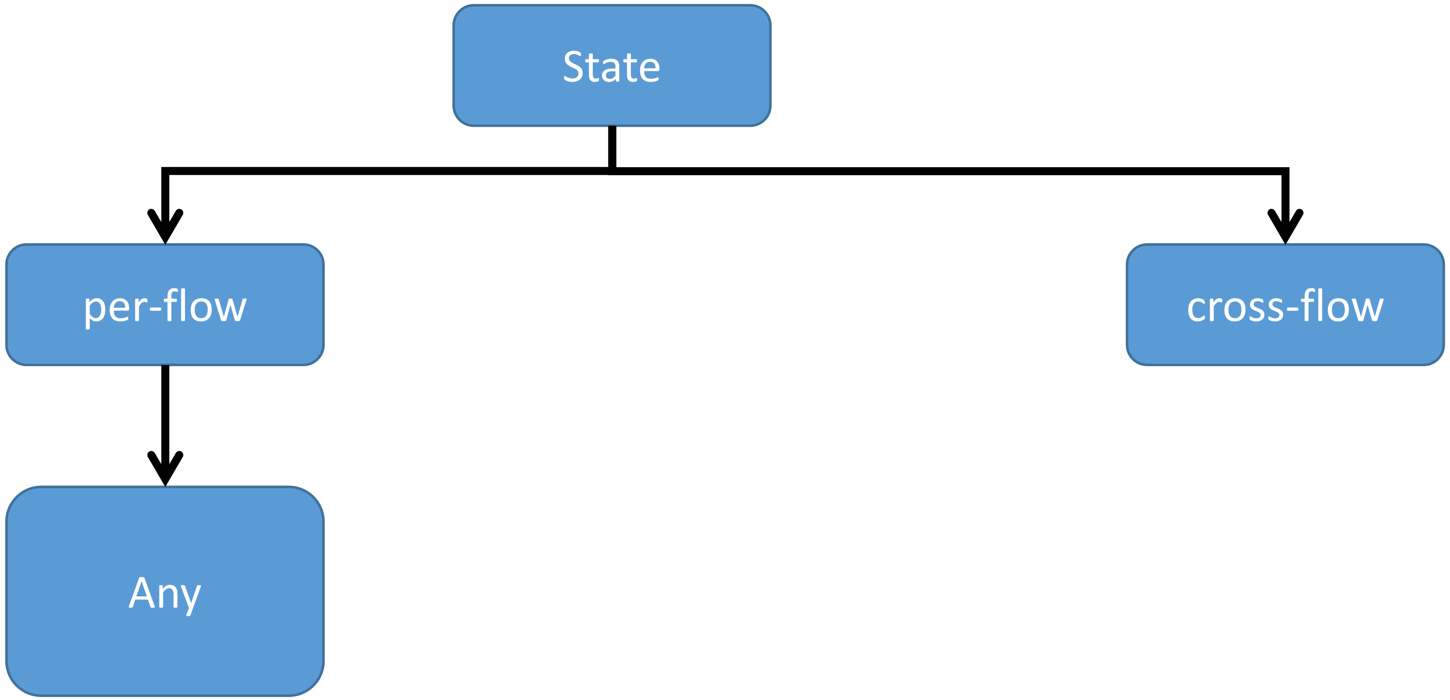
# State Management Strategies

State

# State Management Strategies

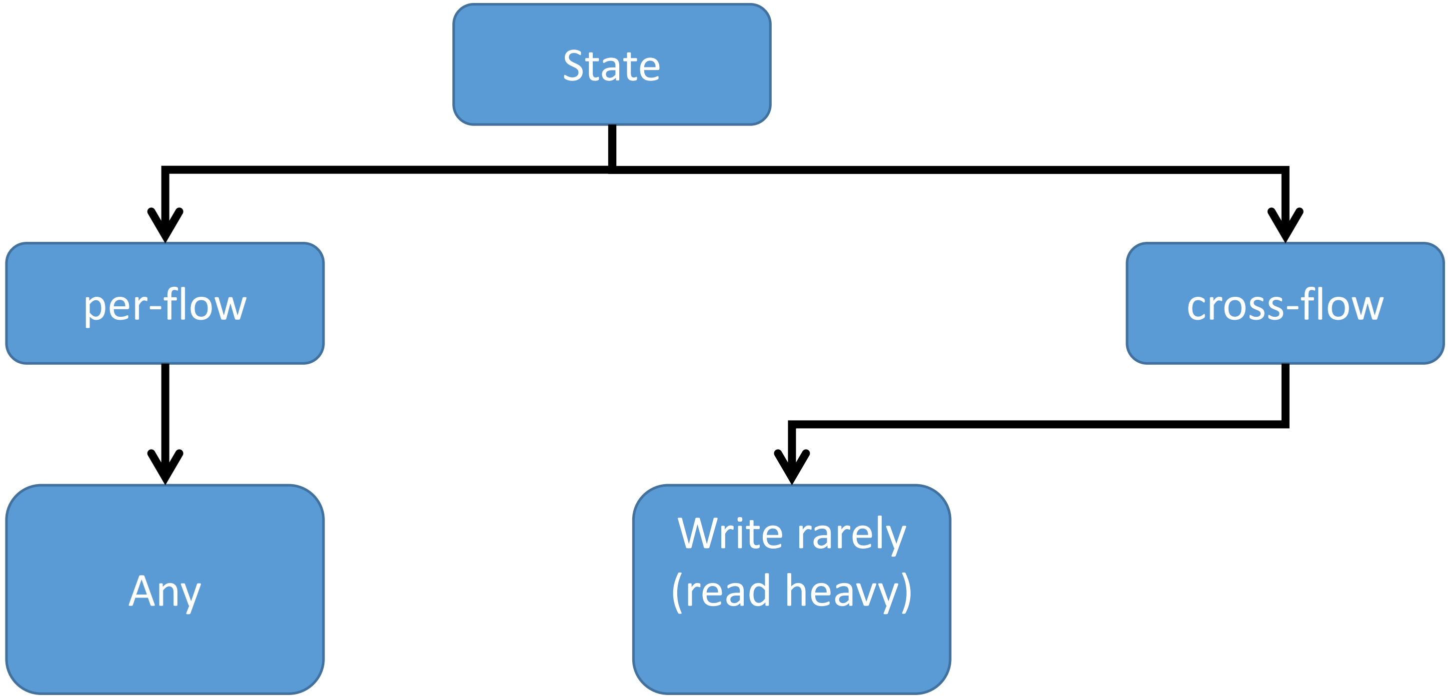


# State Management Strategies




Instance-local caching w/
periodic nonblocking flush

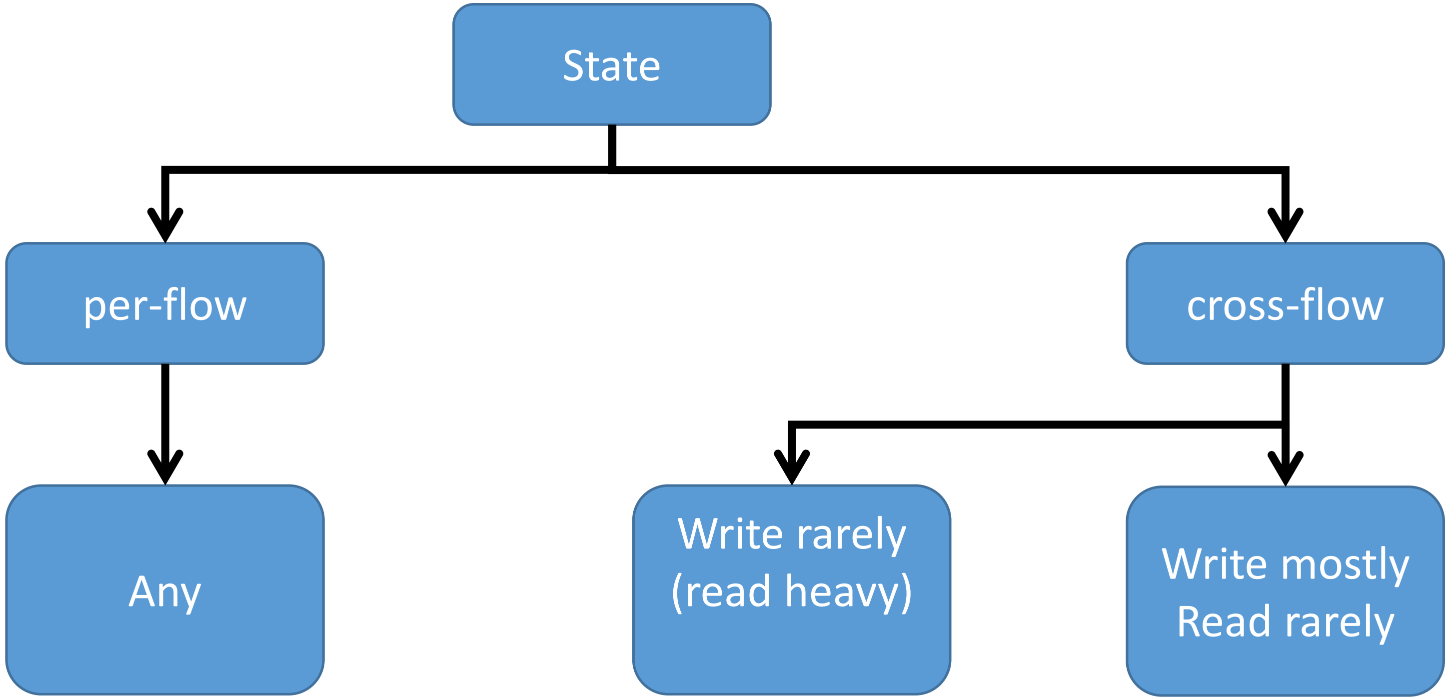
# State Management Strategies




Instance-local caching w/ periodic nonblocking flush

Instance-local caching w/ callbacks

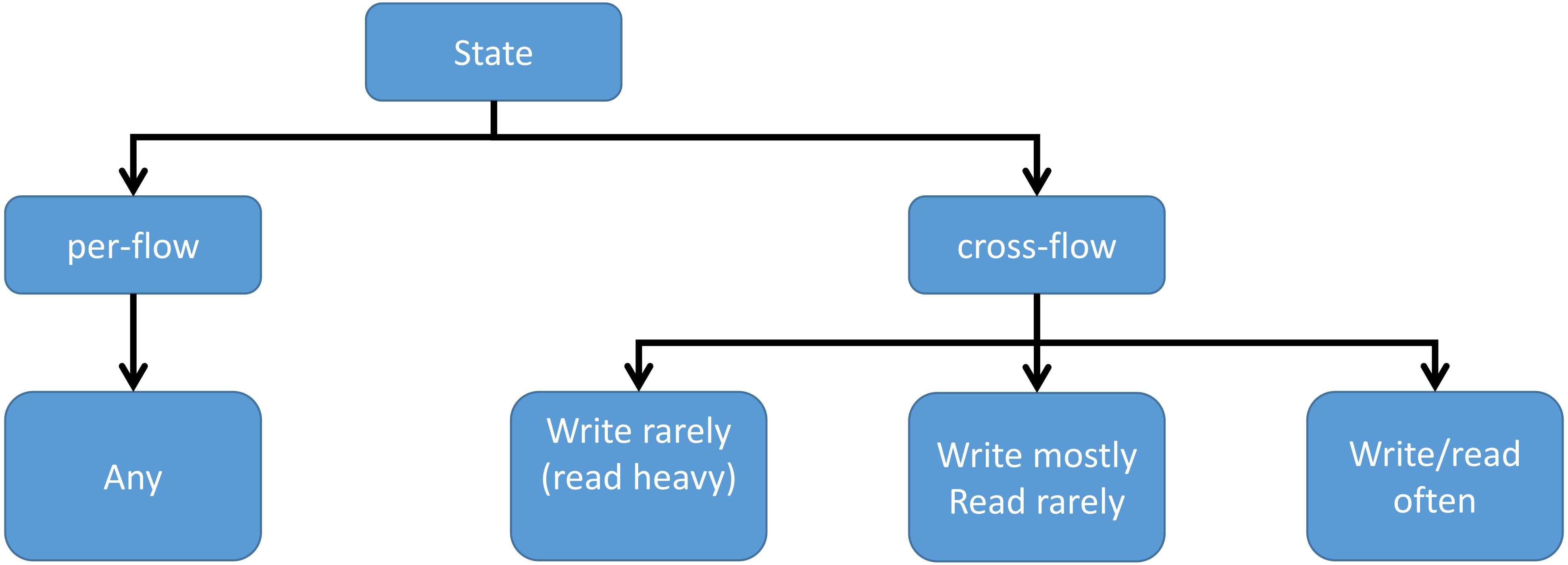
# State Management Strategies




Instance-local caching w/
periodic nonblocking flush

Instance-local caching
w/ callbacks

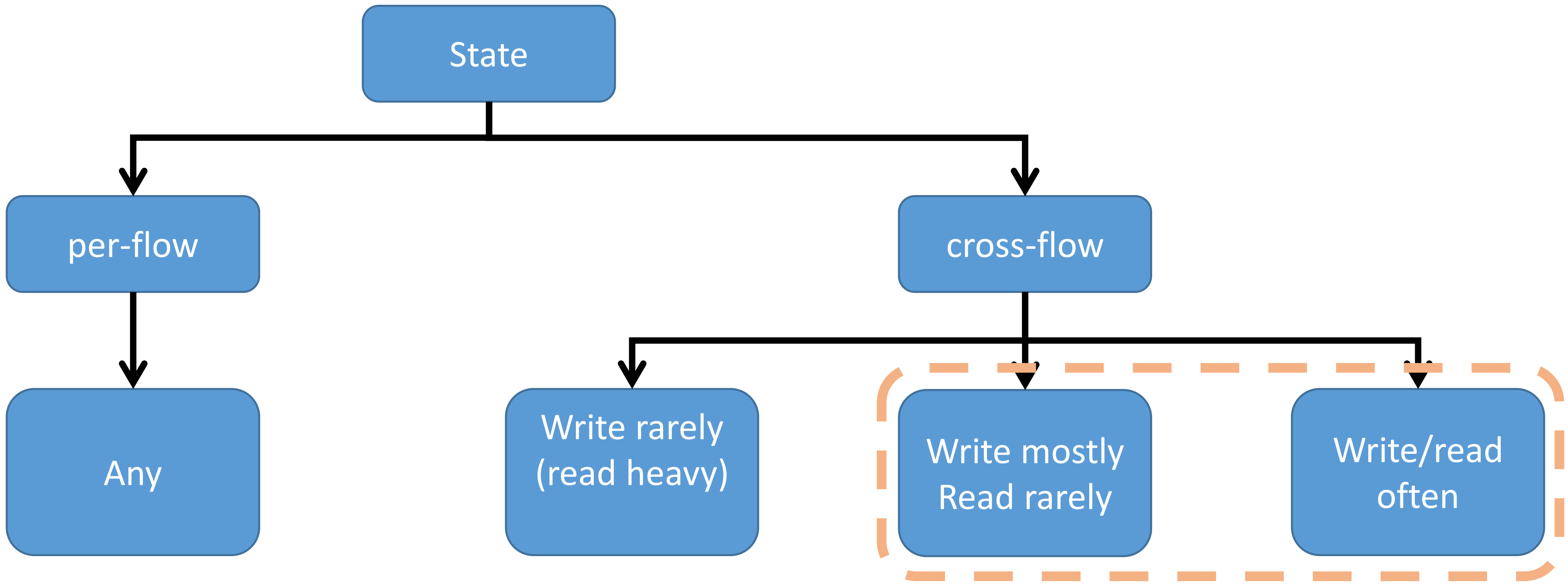
# State Management Strategies




Instance-local caching w/
periodic nonblocking flush

Instance-local caching
w/ callbacks

# State Management Strategies




Instance-local caching w/ periodic nonblocking flush

Instance-local caching w/ callbacks

Operation offloading

# State Maintenance - Offloading Operation

An NF instance can offload operations and instruct the datastore to perform them on its behalf

# State Maintenance - Offloading Operation

An NF instance can offload operations and instruct the datastore to perform them on its behalf

| Operation | Description |
|---|---|
| Increment/Decrement a value | Increment or decrement the value stored at key by the given value |
| Push/pop a value to/from list | Push or pop the value in/from the list stored at the given key |
| Compare and update | Update the value, if the condition is true |

# State Maintenance - Offloading Operation

An NF instance can offload operations and instruct the datastore to perform them on its behalf

| Operation | Description |
|---|---|
| Increment/Decrement a value | Increment or decrement the value stored at key by the given value |
| Push/pop a value to/from list | Push or pop the value in/from the list stored at the given key |
| Compare and update | Update the value, if the condition is true |

The datastore serializes operations issued by different instances for the same shared state object and applies them in the background

# State Maintenance - Offloading Operation

An NF instance can offload operations and instruct the datastore to perform them on its behalf

# State Maintenance - Offloading Operation

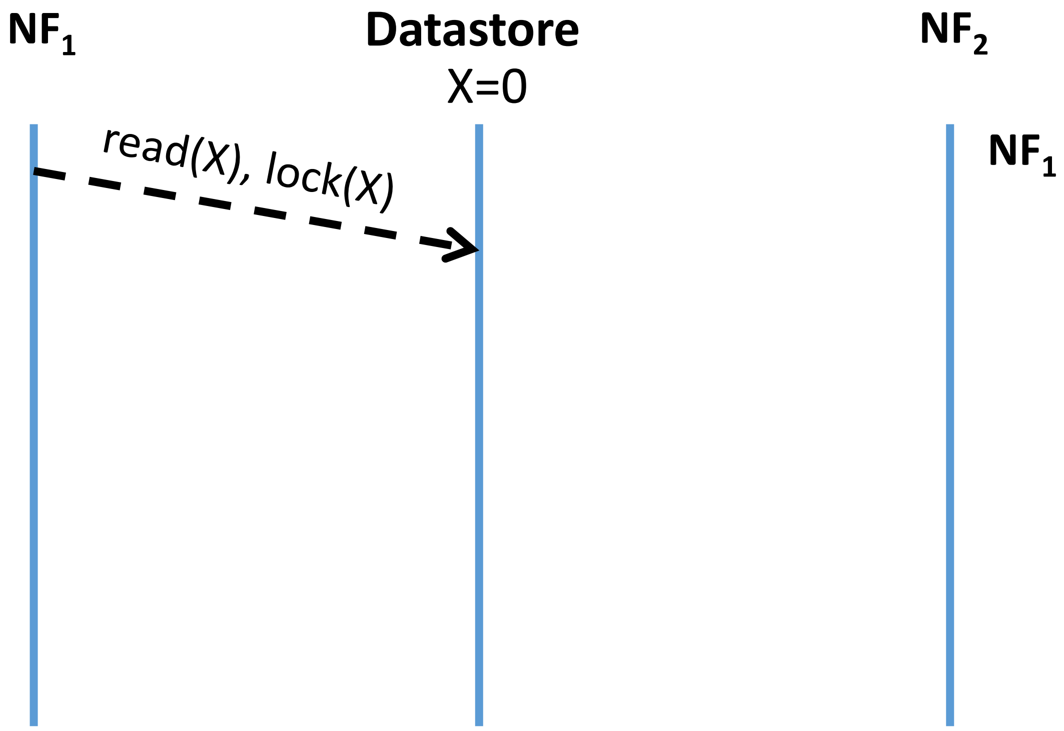
An NF instance can offload operations and instruct the datastore to perform them on its behalf

Without operation offload

# State Maintenance - Offloading Operation

An NF instance can offload operations and instruct the datastore to perform them on its behalf

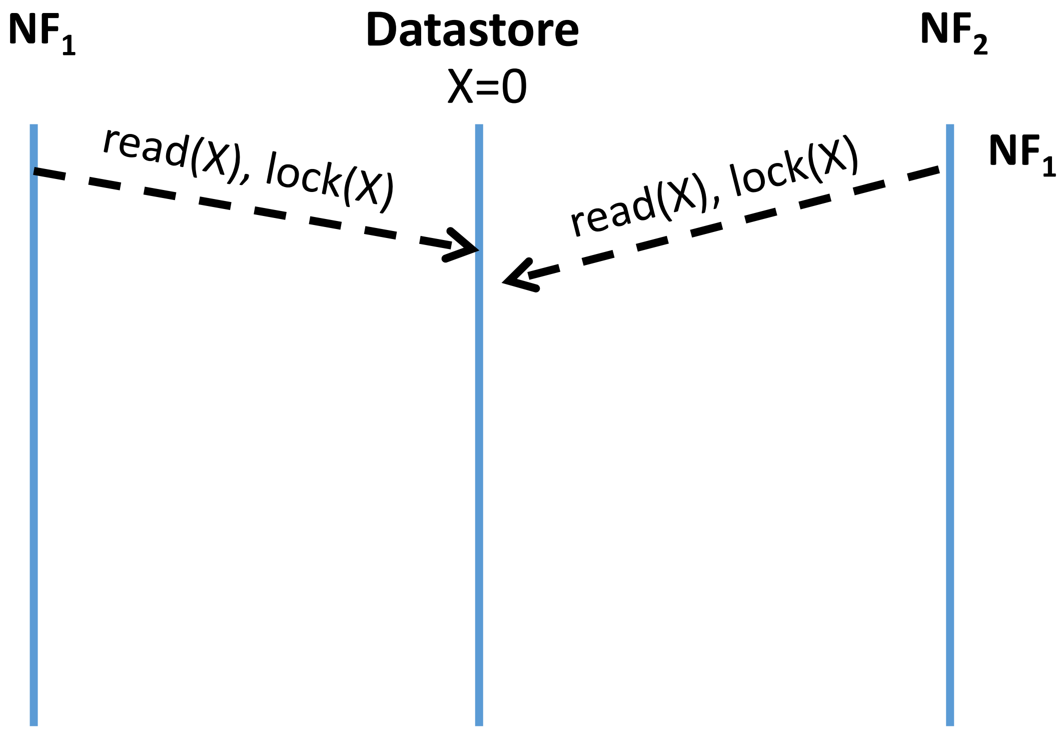
Without operation offload



# State Maintenance - Offloading Operation

An NF instance can offload operations and instruct the datastore to perform them on its behalf

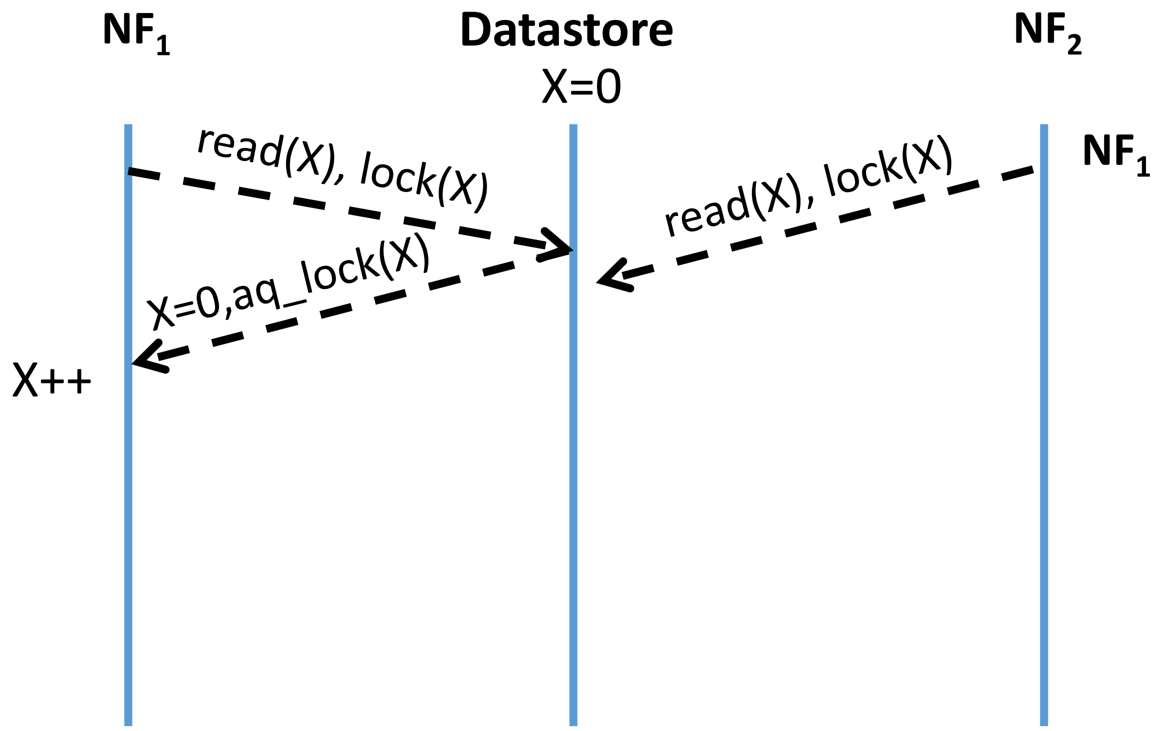
Without operation offload



# State Maintenance - Offloading Operation

An NF instance can offload operations and instruct the datastore to perform them on its behalf

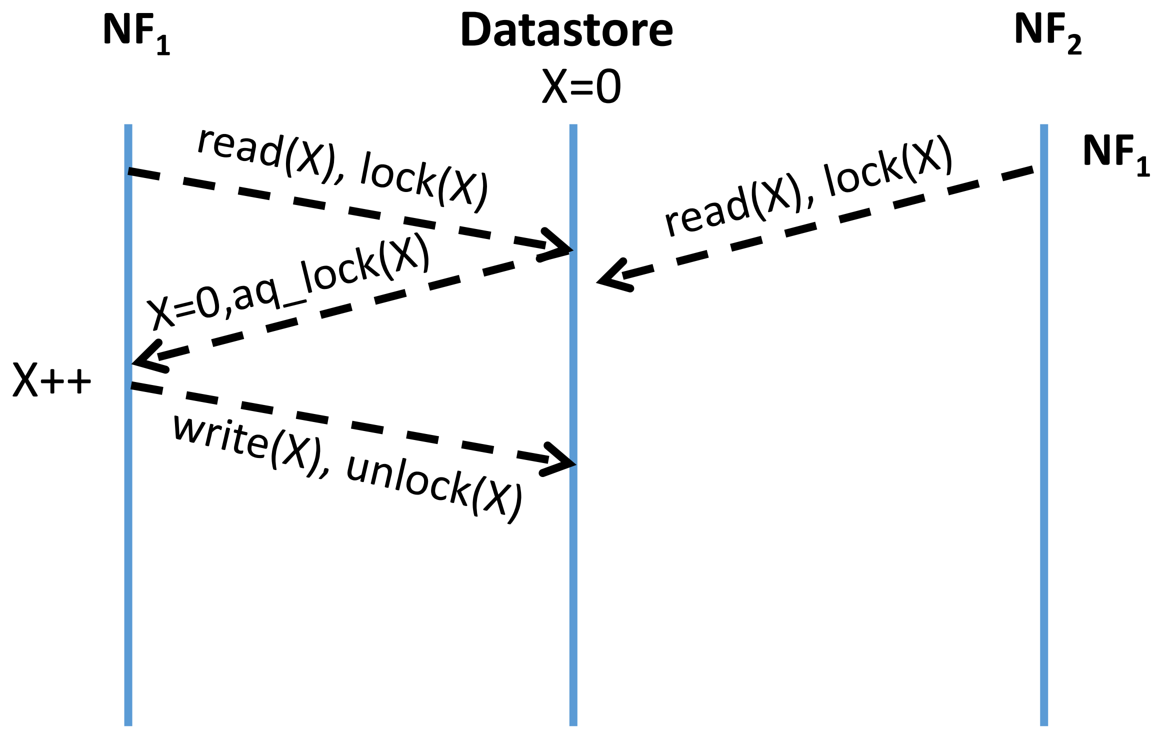
Without operation offload



# State Maintenance - Offloading Operation

An NF instance can offload operations and instruct the datastore to perform them on its behalf

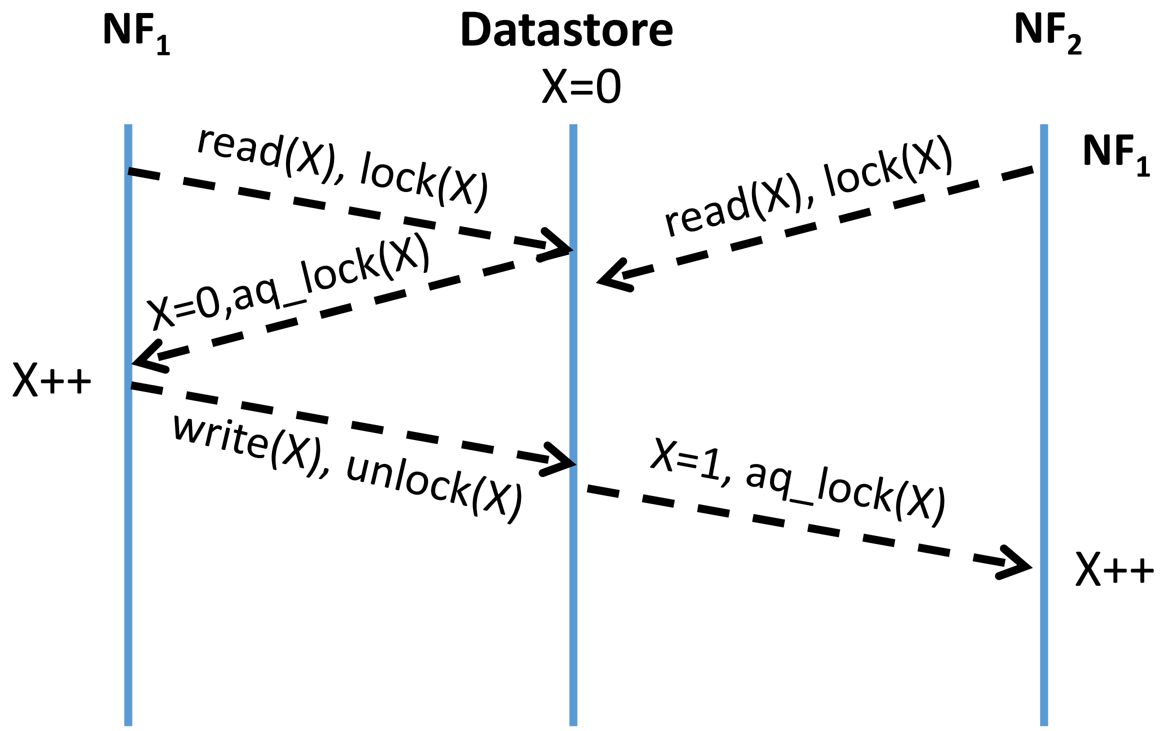
Without operation offload



# State Maintenance - Offloading Operation

An NF instance can offload operations and instruct the datastore to perform them on its behalf

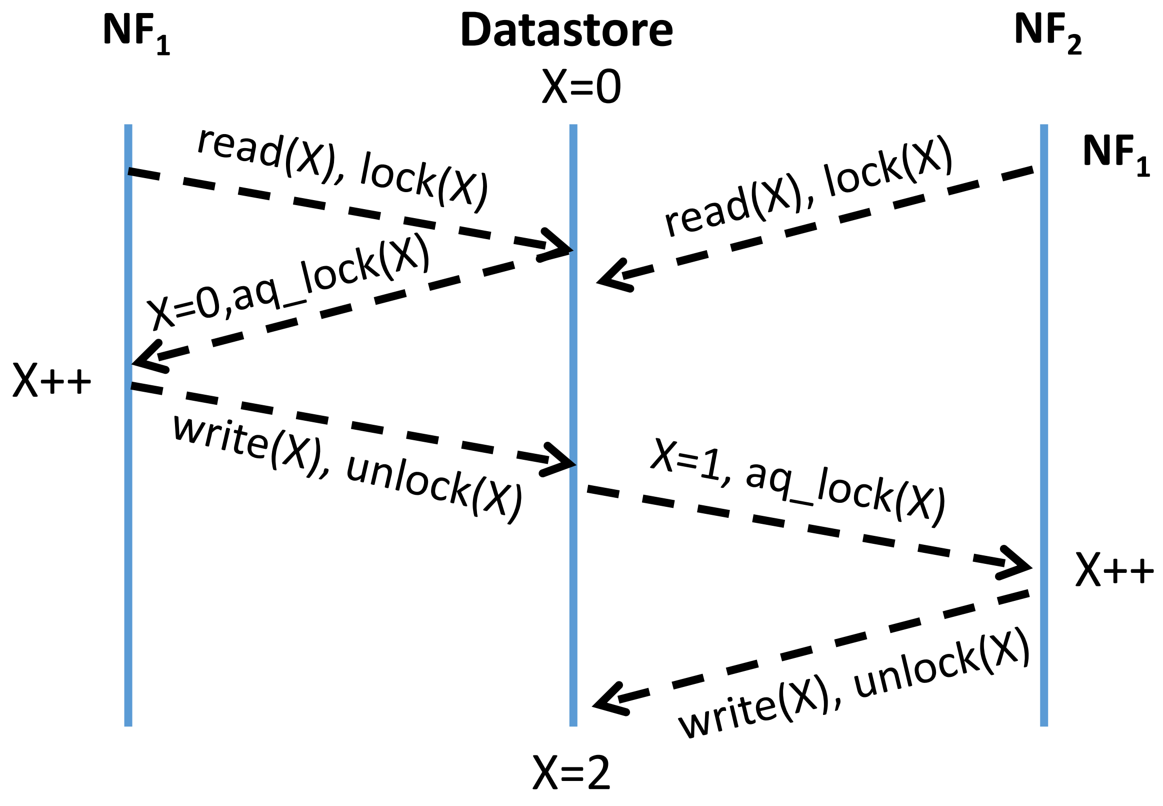
Without operation offload



# State Maintenance - Offloading Operation

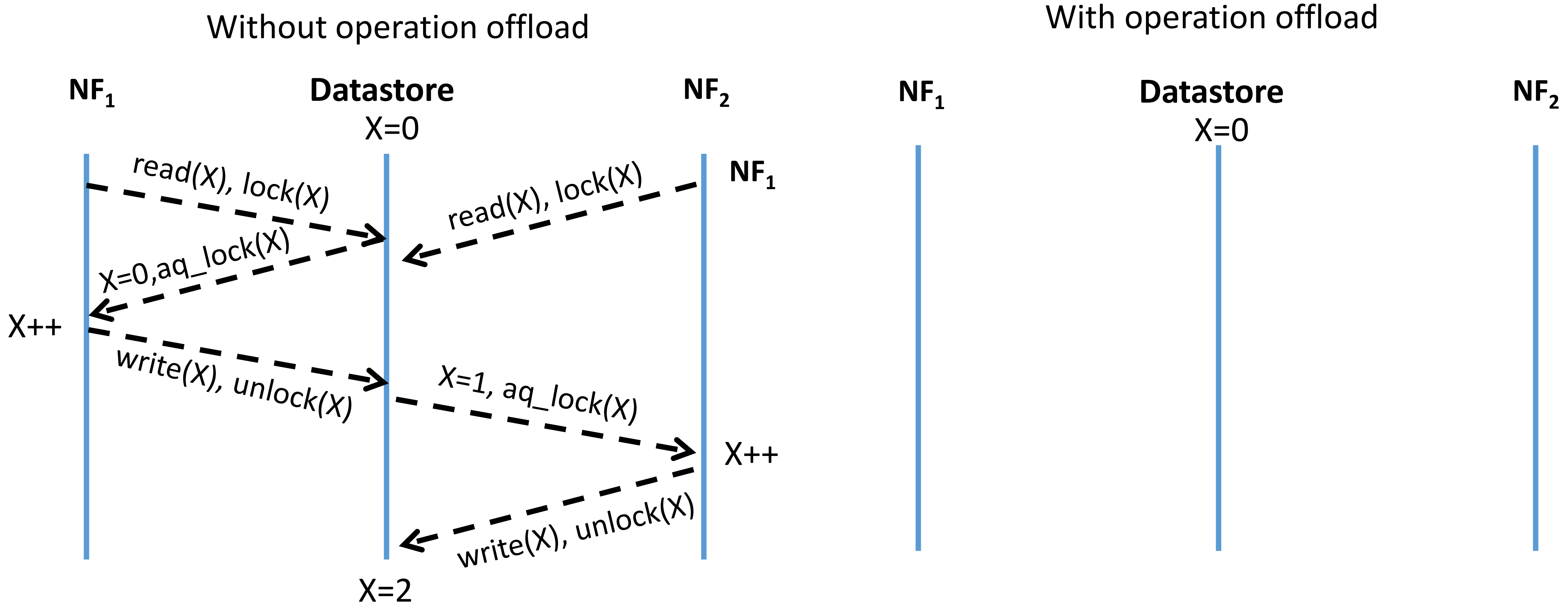
An NF instance can offload operations and instruct the datastore to perform them on its behalf

Without operation offload



# State Maintenance - Offloading Operation

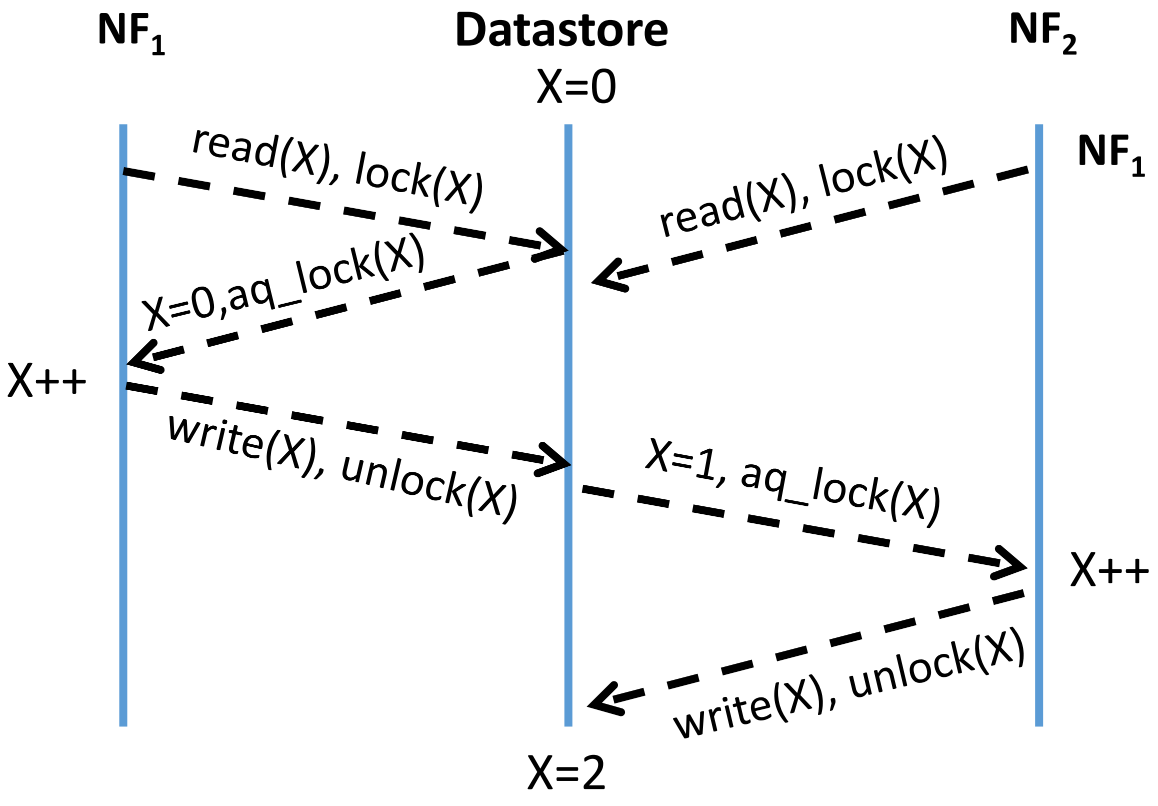
An NF instance can offload operations and instruct the datastore to perform them on its behalf



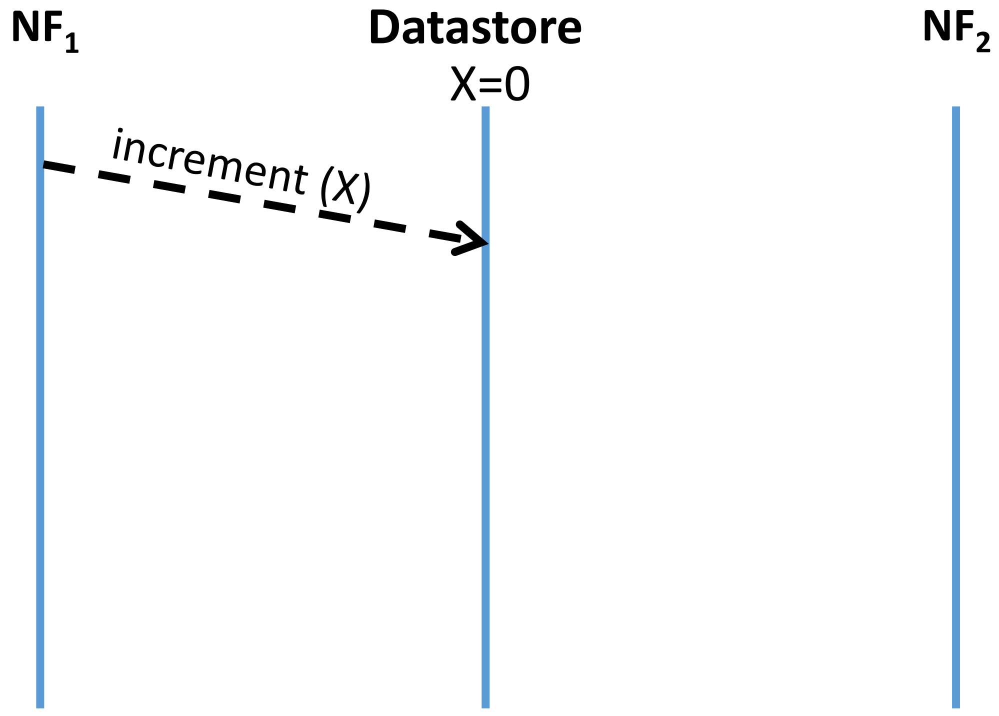
# State Maintenance - Offloading Operation

An NF instance can offload operations and instruct the datastore to perform them on its behalf

Without operation offload

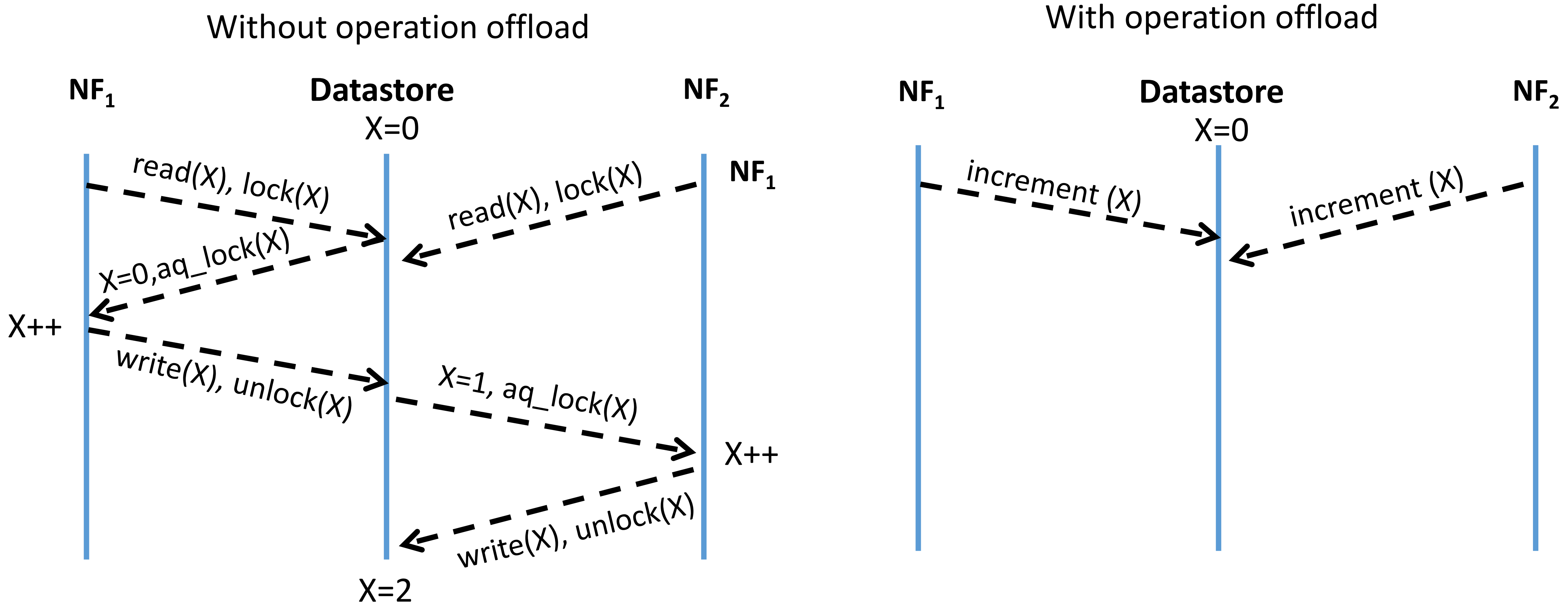



With operation offload



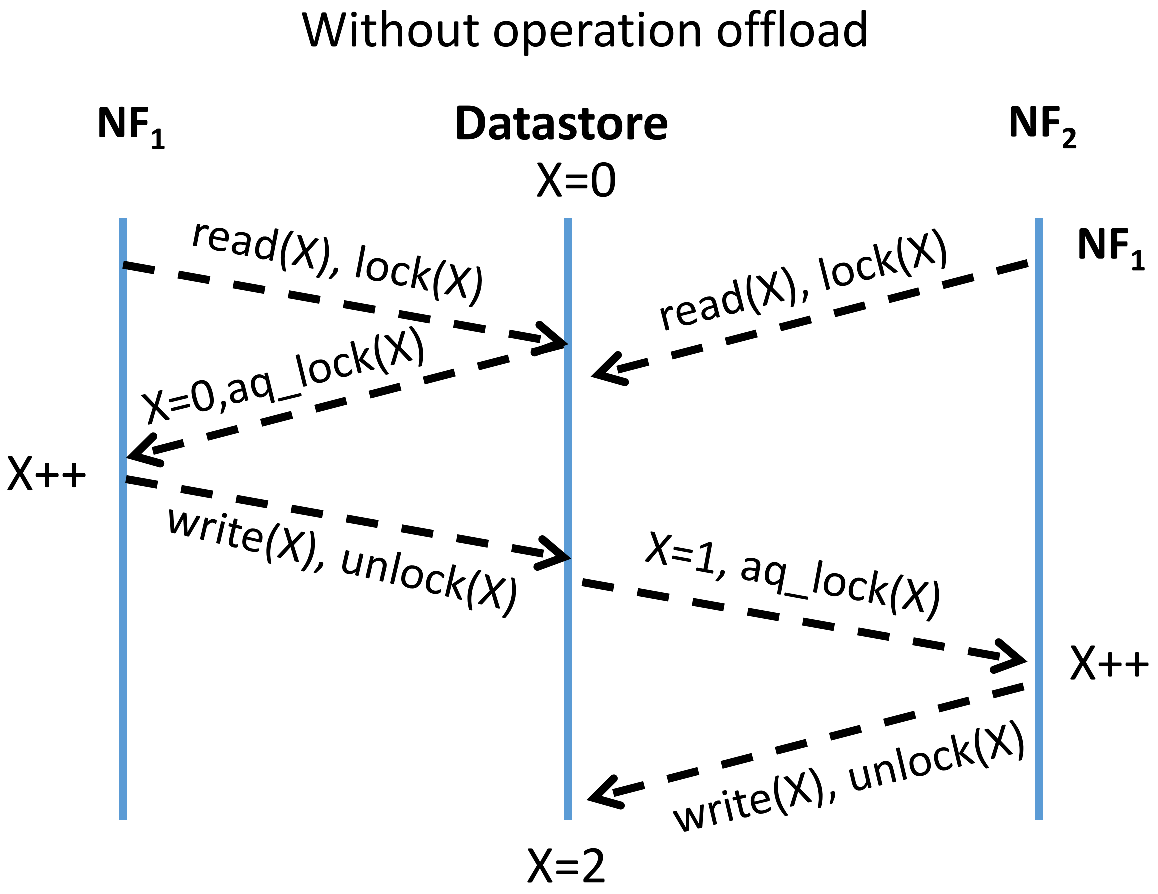
# State Maintenance - Offloading Operation

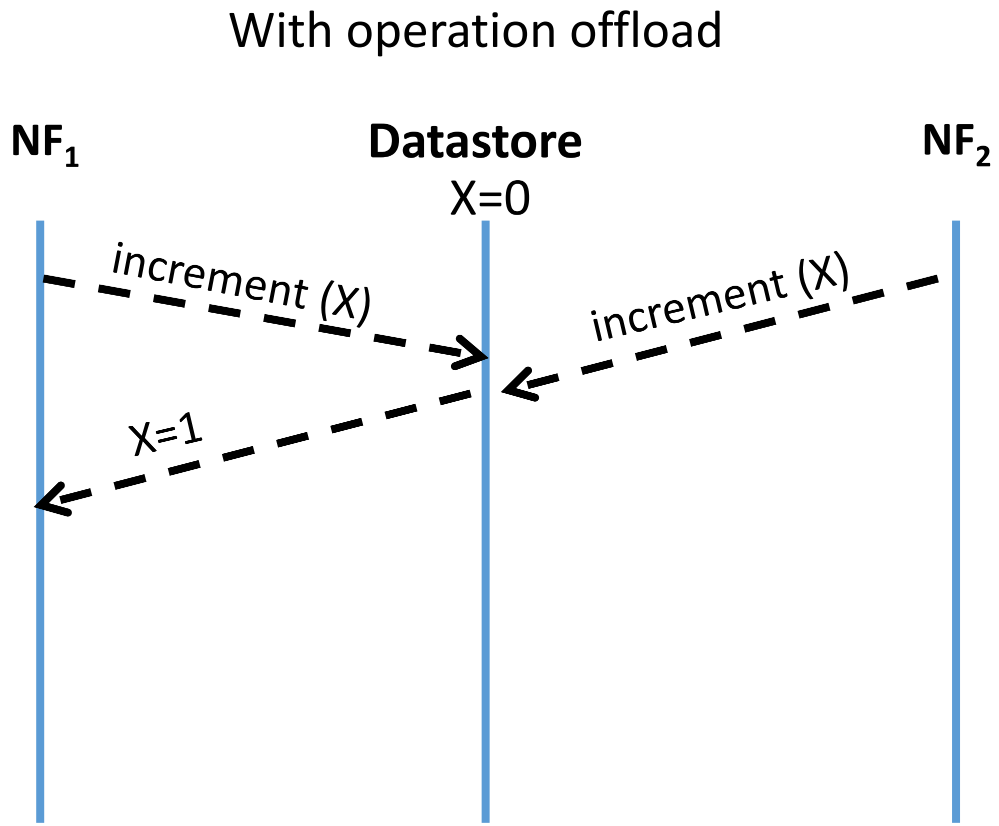
An NF instance can offload operations and instruct the datastore to perform them on its behalf



# State Maintenance - Offloading Operation

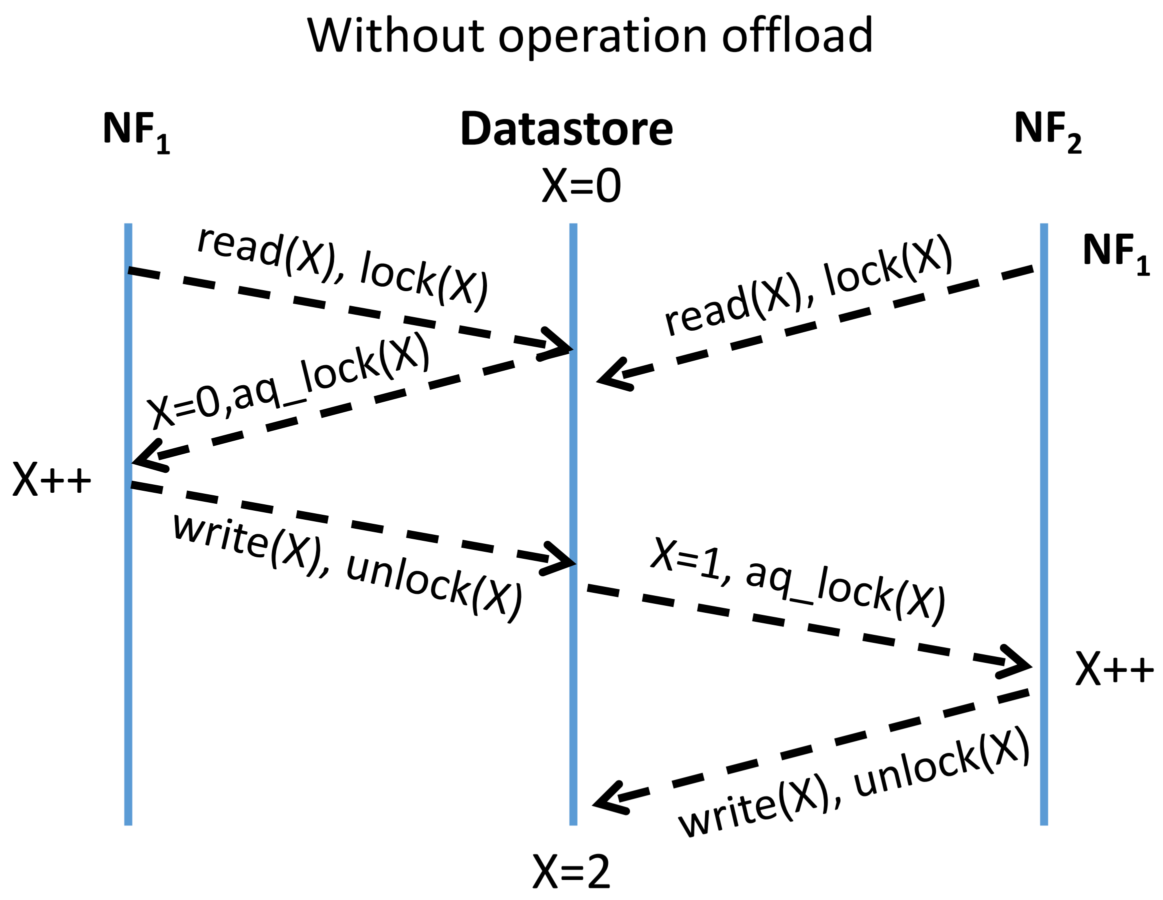
An NF instance can offload operations and instruct the datastore to perform them on its behalf

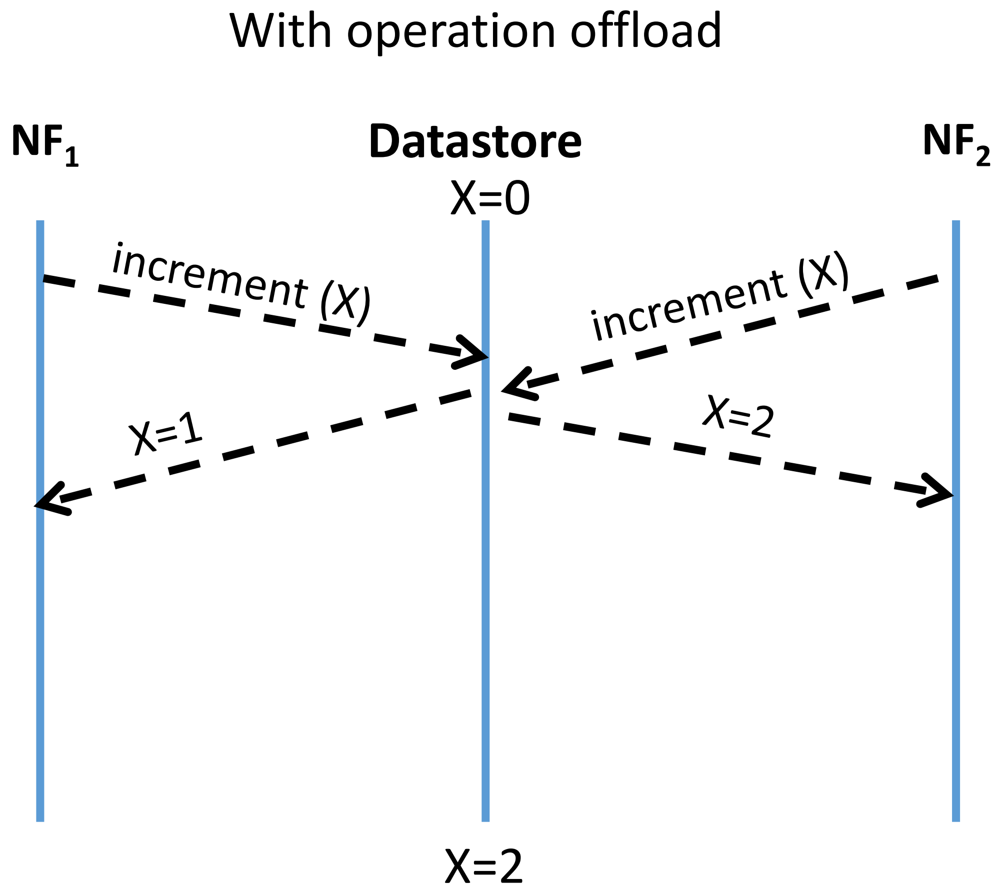


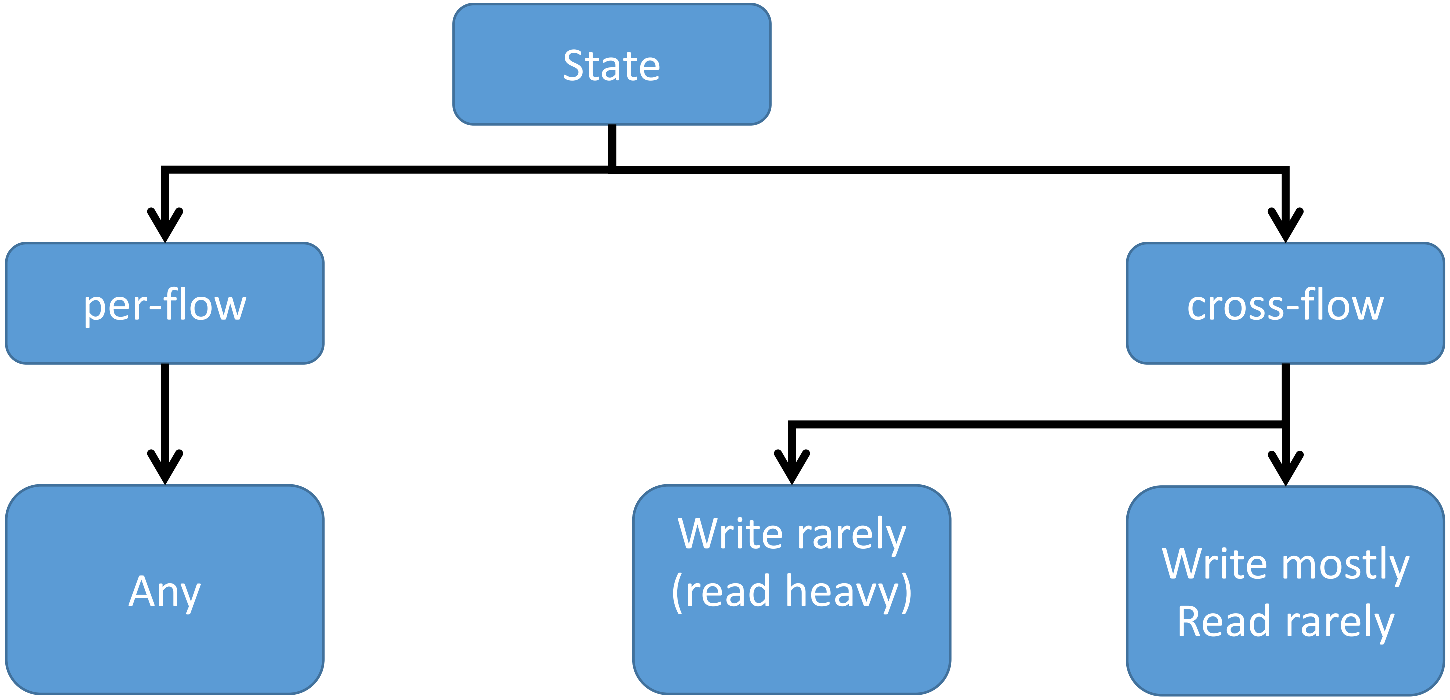
# State Maintenance - Offloading Operation

An NF instance can offload operations and instruct the datastore to perform them on its behalf



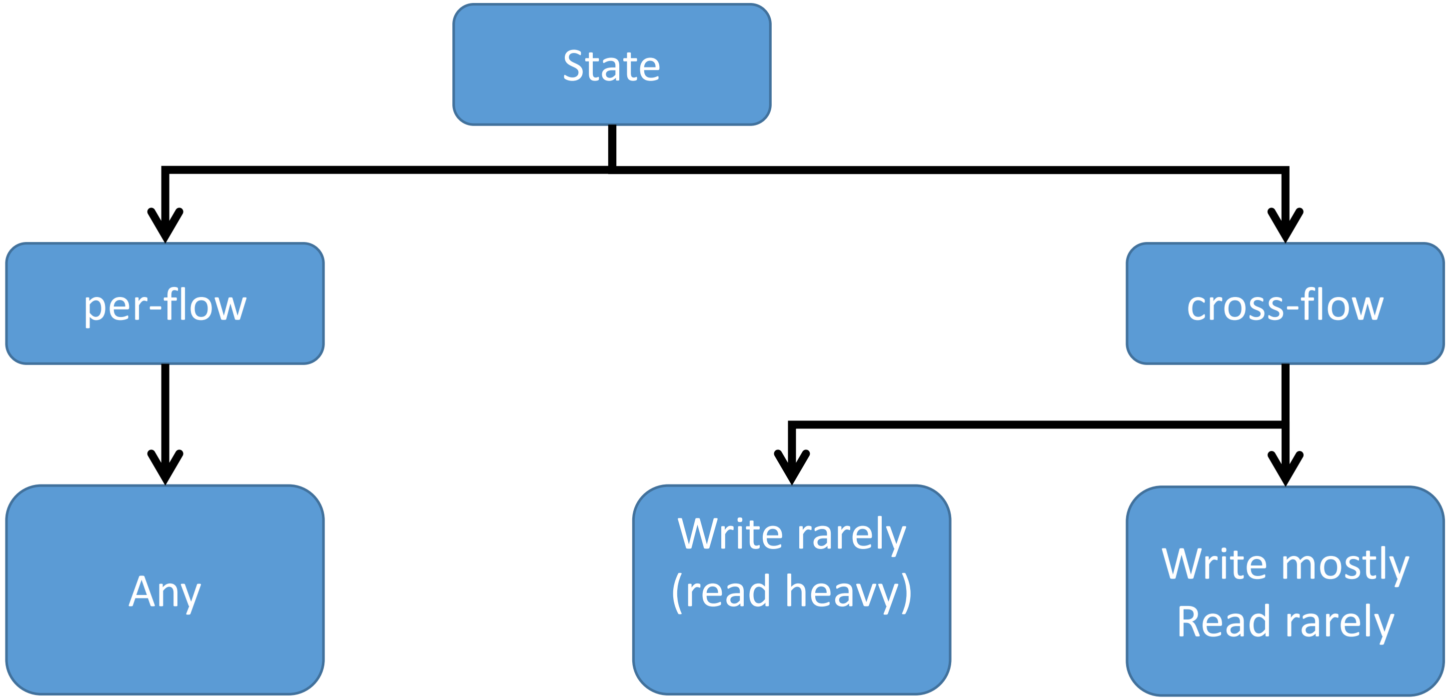
# State Management Strategies




Instance-local caching w/ periodic nonblocking flush

Instance-local caching w/ callbacks

# State Management Strategies

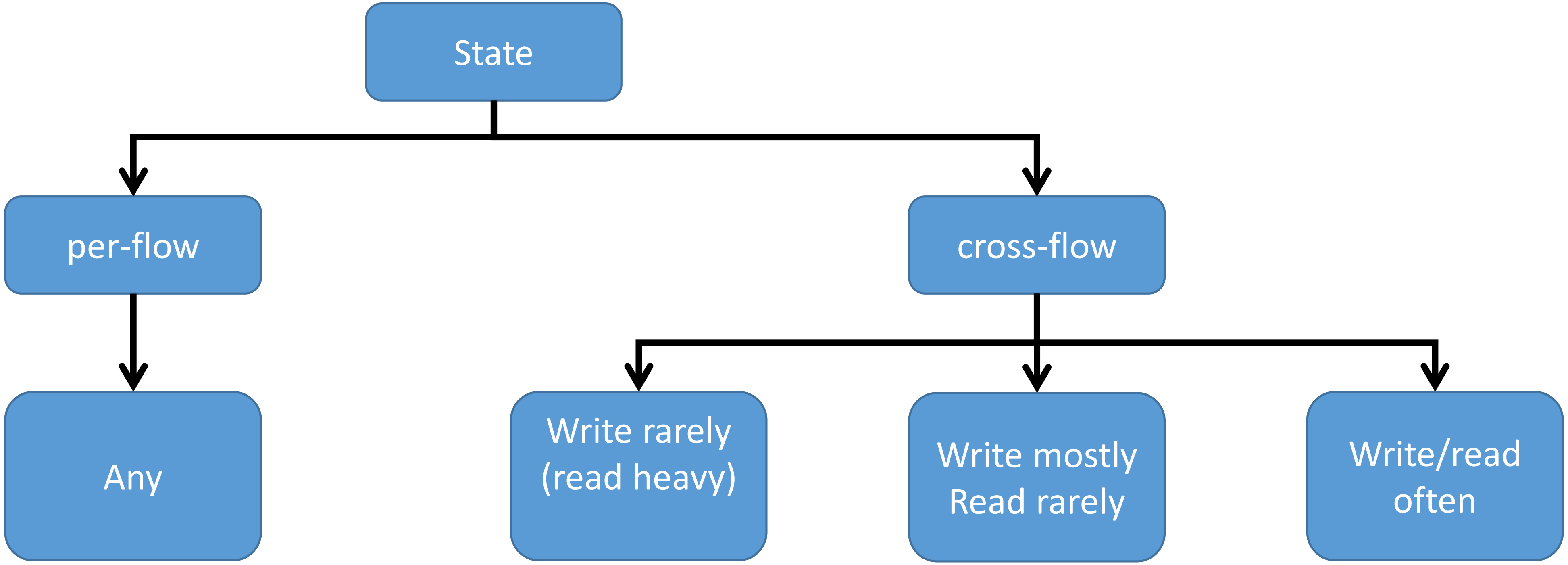



Instance-local caching w/ periodic nonblocking flush

Instance-local caching w/ callbacks

Non-blocking operation without caching

# State Management Strategies




Instance-local caching w/ periodic nonblocking flush

Instance-local caching w/ callbacks

Non-blocking operation without caching

Depends upon traffic split. Cache, if split allows; flush periodically

# CHC

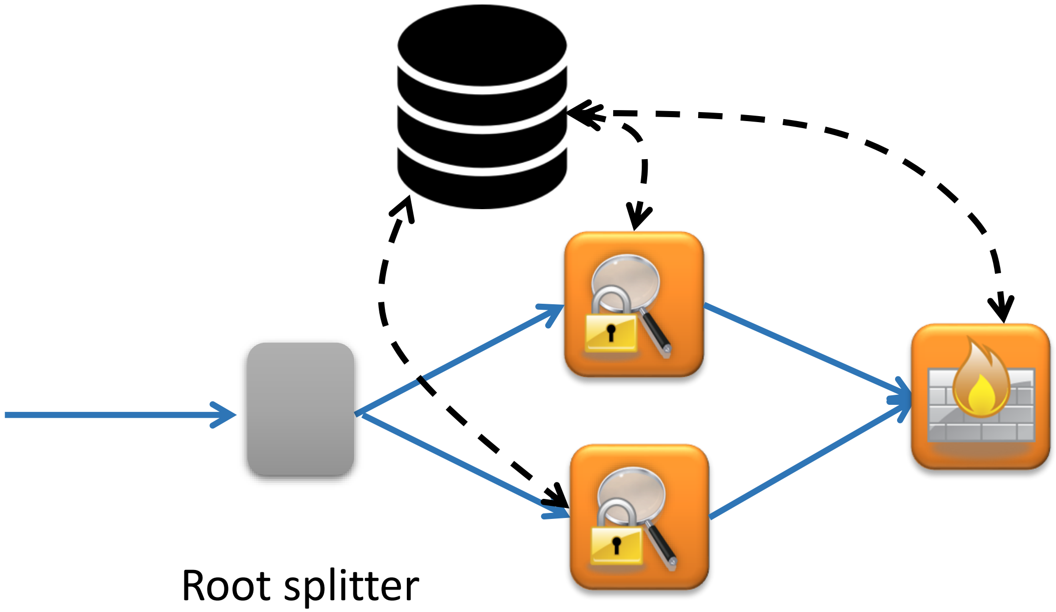
CHC is a generic NFV framework to support all of these requirements without trading off ***correctness*** for ***performance*** or ***functionality***

CHC consist of three main  building blocks

1. State store external to NF
2. NF state-aware state management algorithms
3. Metadata – logical clock and logs

# Metadata

CHC adds a “root splitter” at the entry of a chain that:

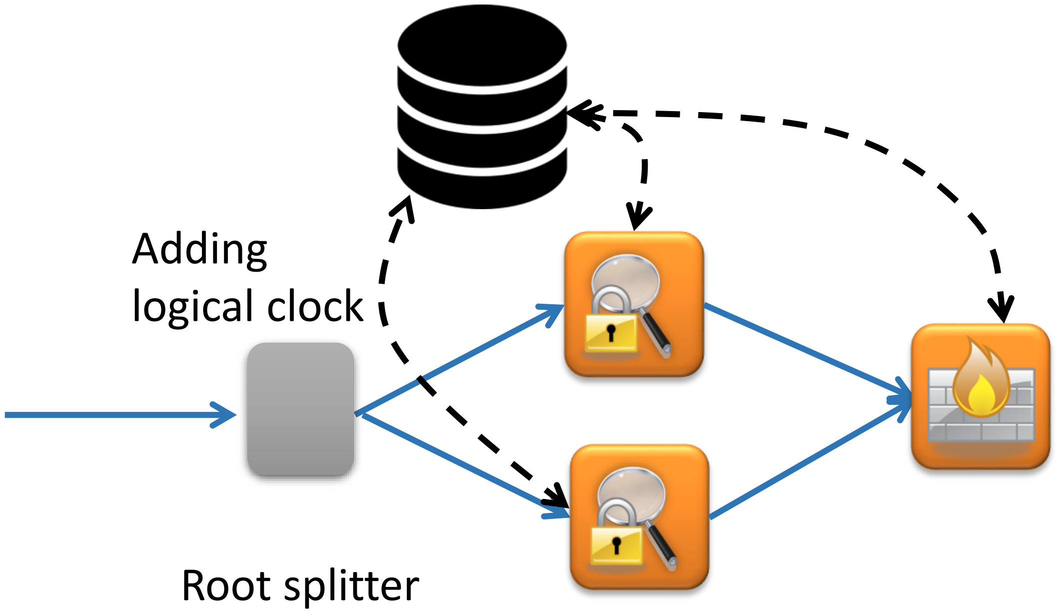


Root splitter

# Metadata

CHC adds a "root splitter" at the entry of a chain that:

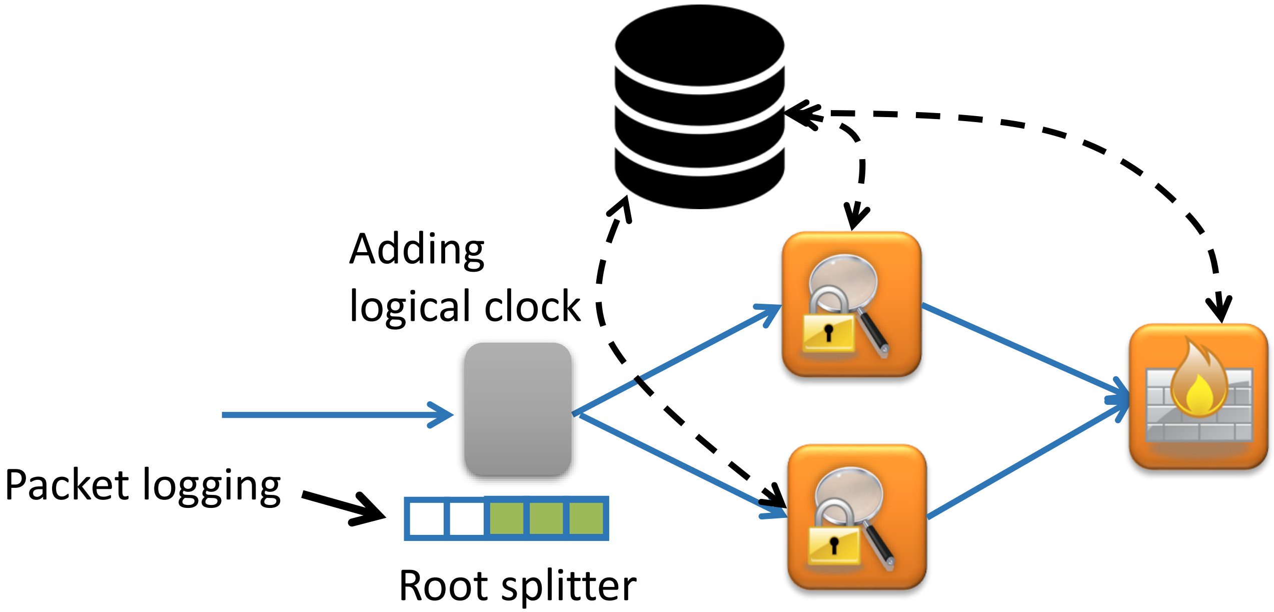
- Root splitter attaches a unique logical clock with each packet. Logical clock is used for ***duplication suppression, ordering,*** and ***traffic replay***



# Metadata

CHC adds a "root splitter" at the entry of a chain that:

- Root splitter attaches a unique logical clock with each packet. Logical clock is used for ***duplication suppression, ordering,*** and ***traffic replay***
- It also logs all the in-transit packets

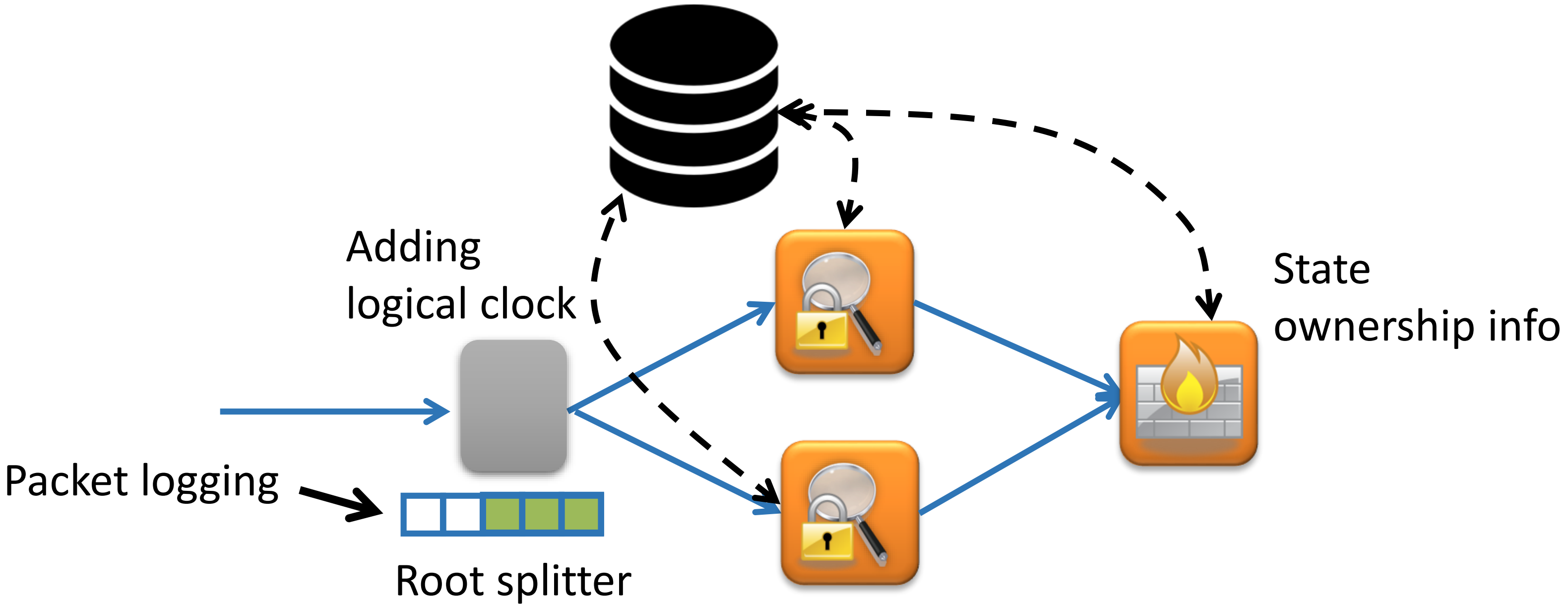


# Metadata

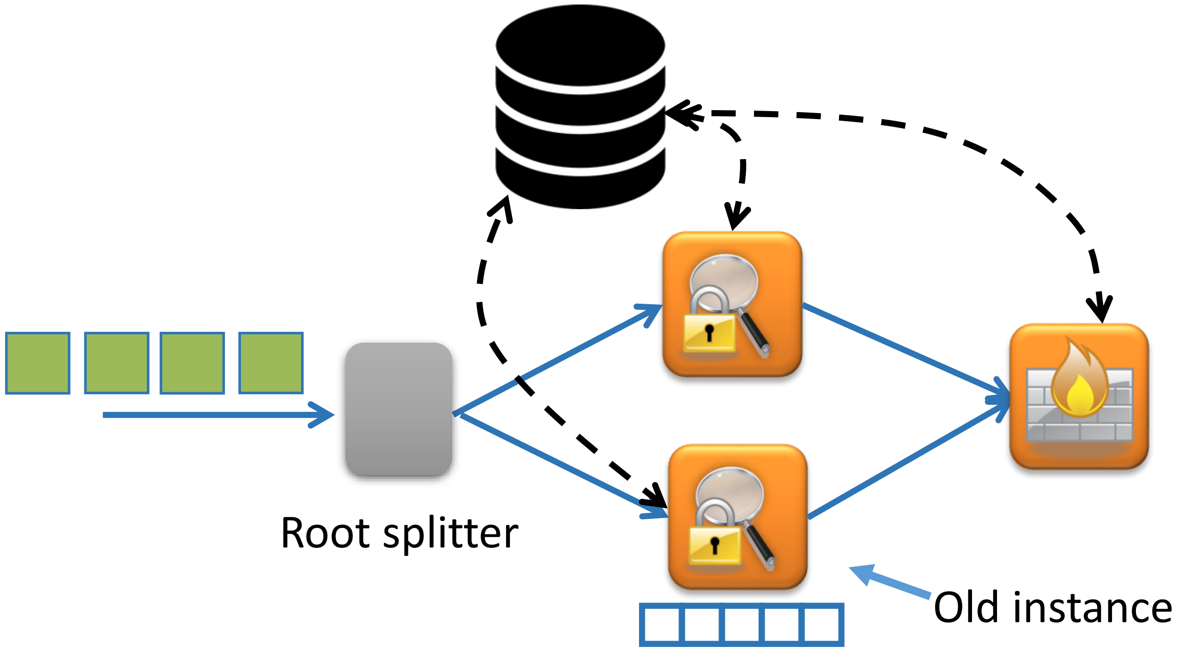
CHC adds a “root splitter” at the entry of a chain that:

- Root splitter attaches a unique logical clock with each packet. Logical clock is used for ***duplication suppression, ordering,*** and ***traffic replay***
- It also logs all the in-transit packets

CHC encodes state object’s ownership information and logical clock associated with state operations as metadata

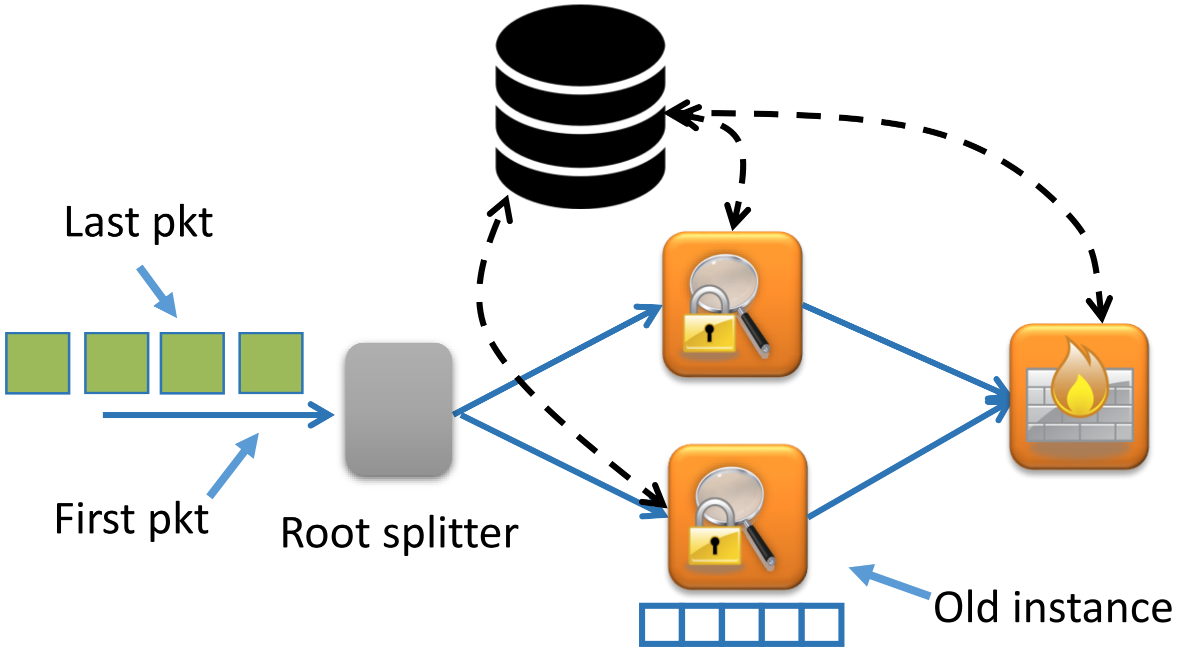


# CHC – Elastic Scaling



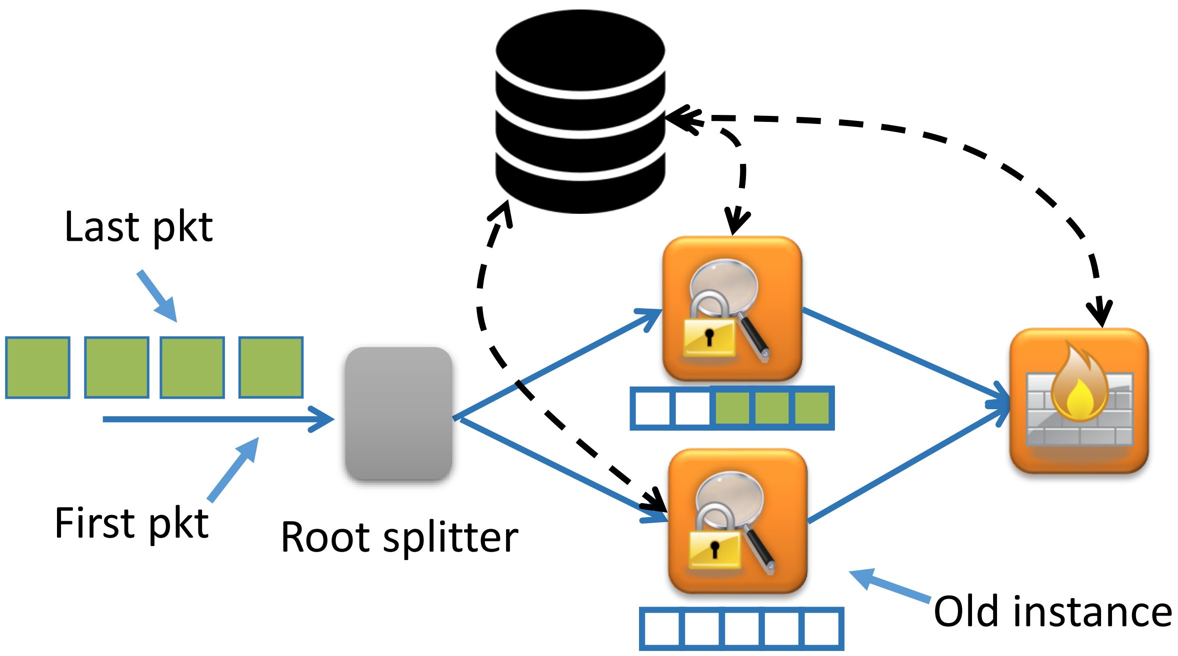
# CHC – Elastic Scaling

- CHC marks the last packet going to the old instance and first packet going to the new instance



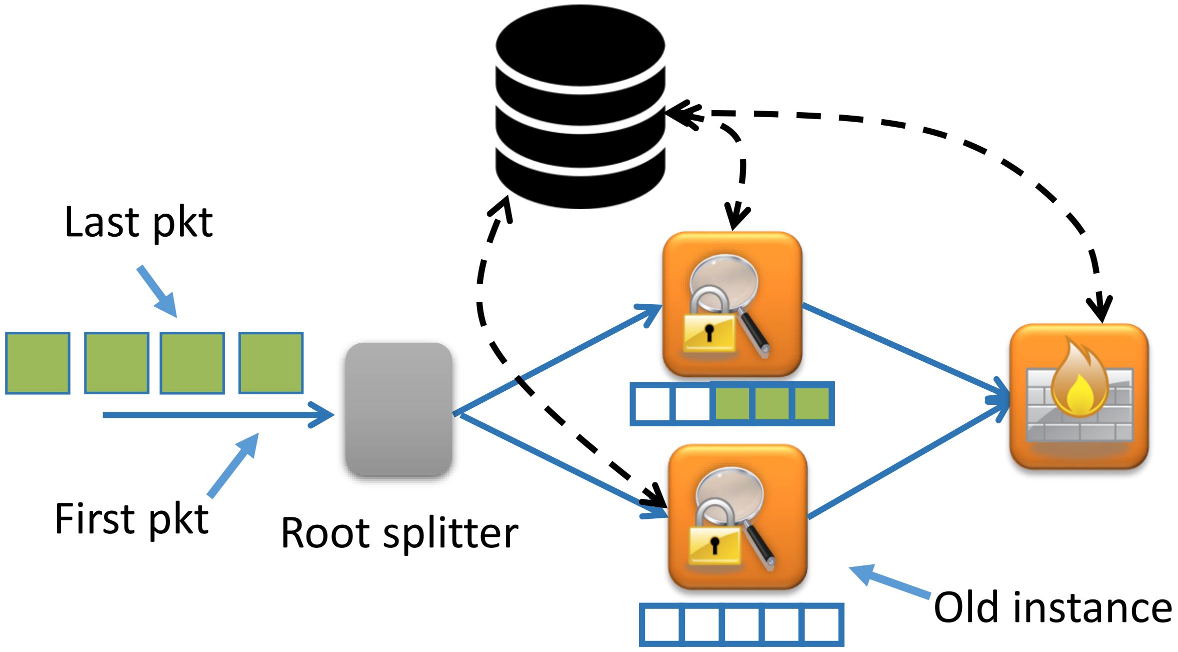
# CHC – Elastic Scaling

- CHC marks the last packet going to the old instance and first packet going to the new instance
- Ownership information encoded as metadata of state objects is used to ensure consistent handover of ***per-flow state***



# CHC – Elastic Scaling

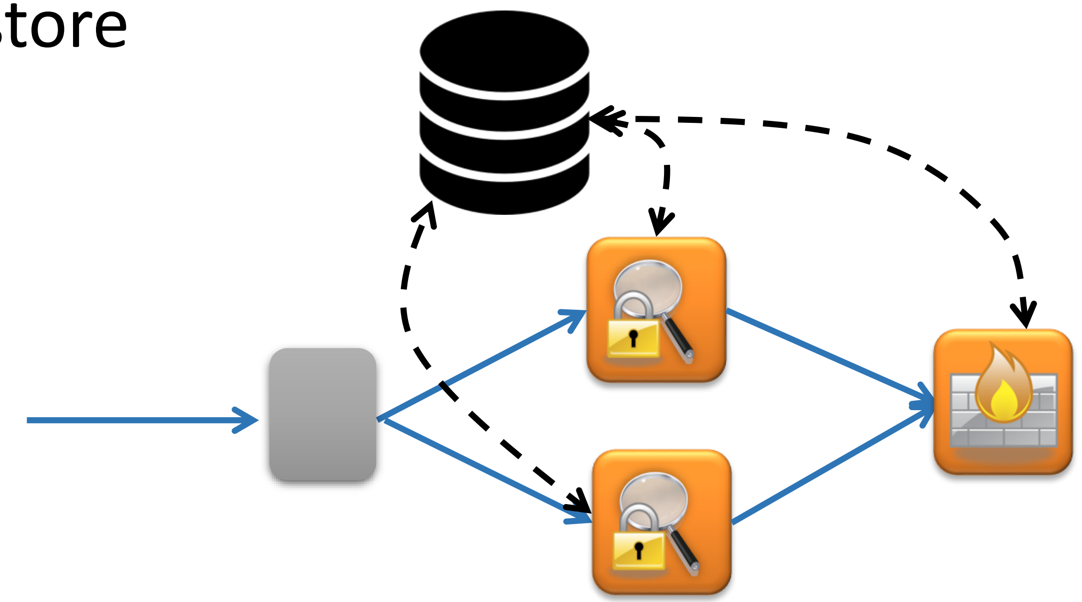
- CHC marks the last packet going to the old instance and first packet going to the new instance
- Ownership information encoded as metadata of state objects is used to ensure consistent handover of ***per-flow state***
- ***Cross-flow state*** does not require any special handling as operation offloading is used to update it



# CHC – Fault Tolerance

CHC provides fault tolerance for:

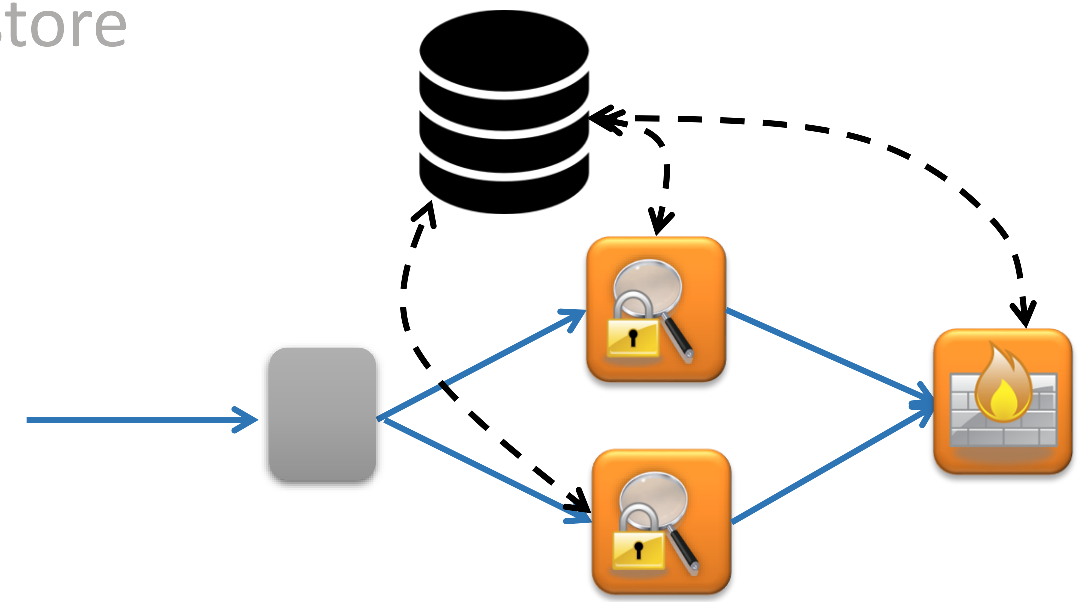
- NF instance
- Root splitter
- Datastore



# CHC – Fault Tolerance

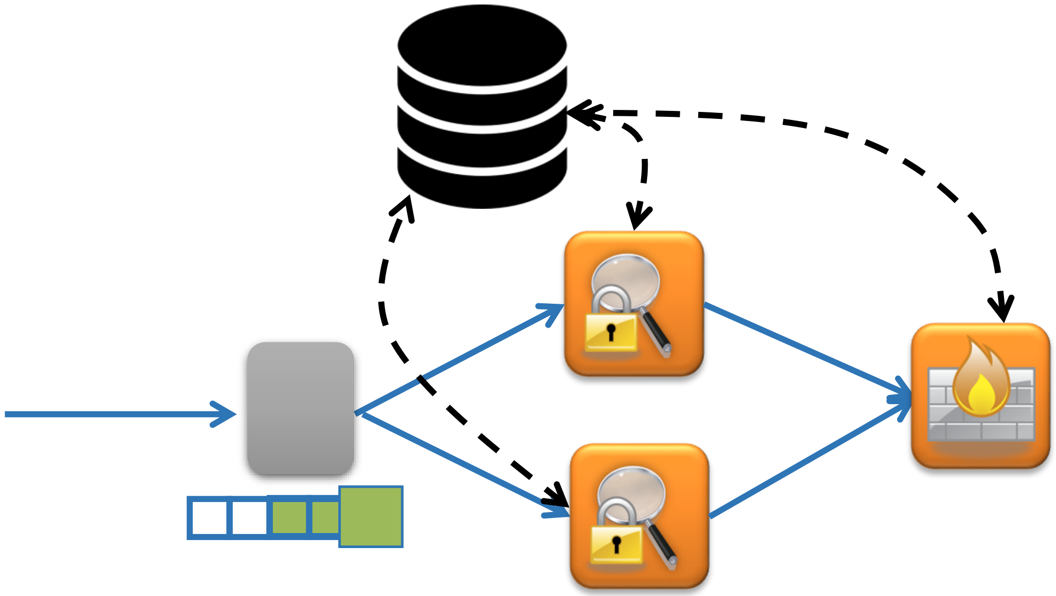
CHC provides fault tolerance for:

- NF instance
- Root splitter
- Datastore



# CHC – Fault Tolerance

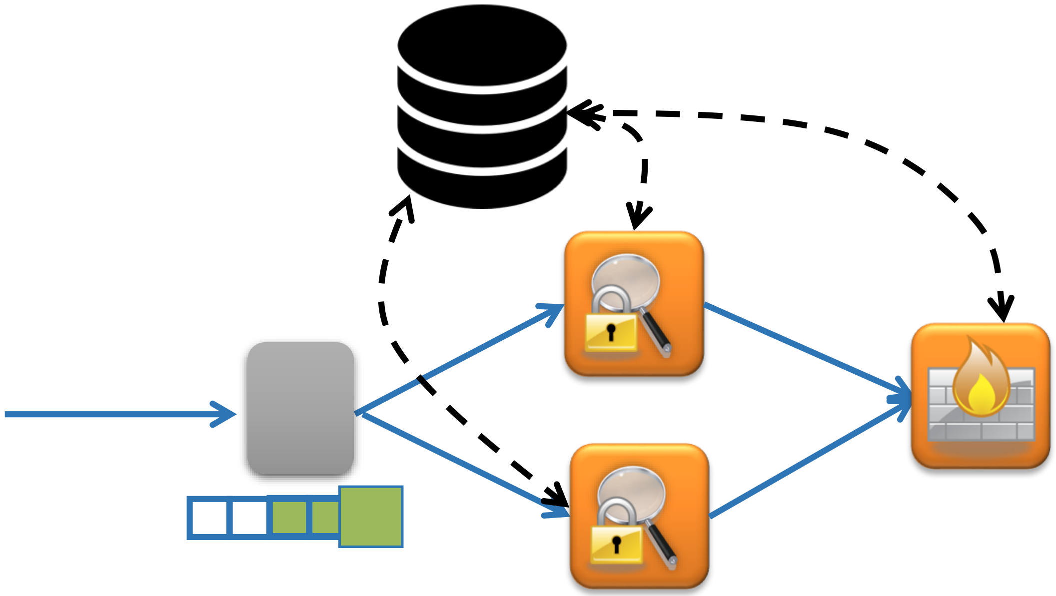
NF instance failure recovery:



# CHC – Fault Tolerance

NF instance failure recovery:

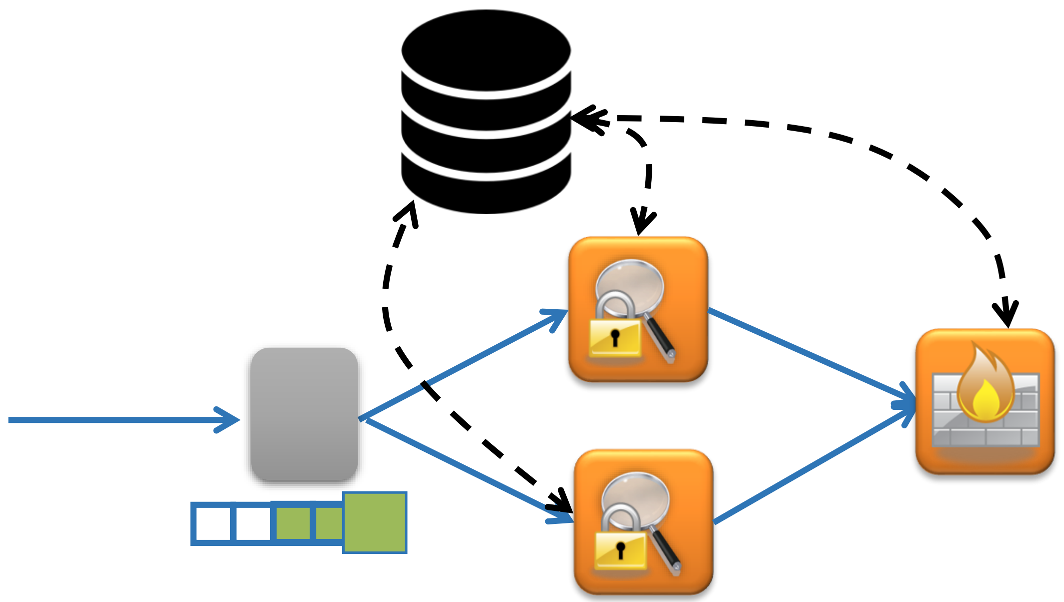
- Failover instance takes over



# CHC – Fault Tolerance

NF instance failure recovery:

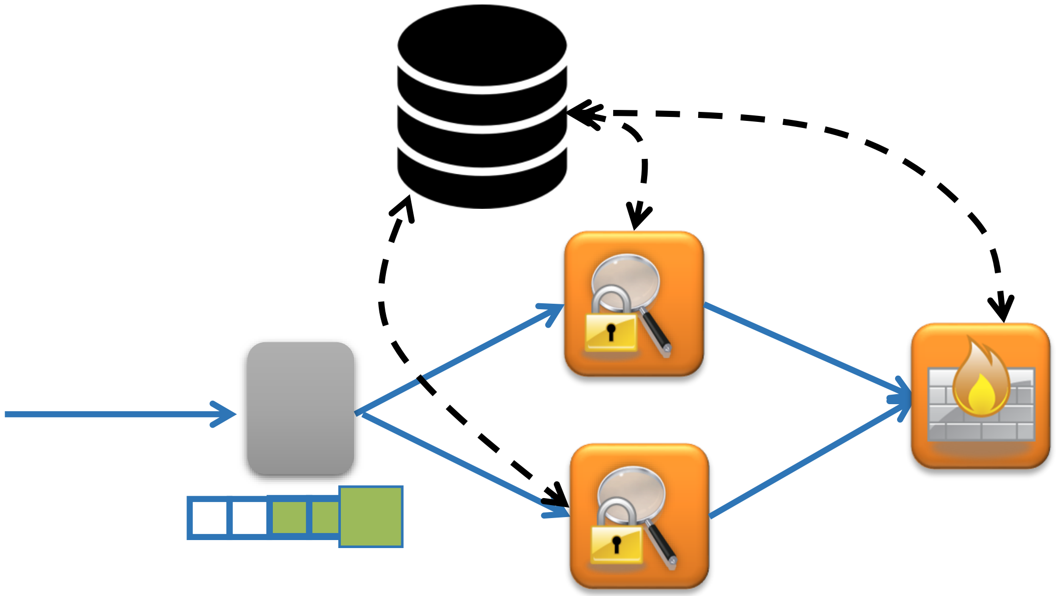
- Failover instance takes over
- Datastore associates the ***failover instance ID*** with the relevant state



# CHC – Fault Tolerance

NF instance failure recovery:

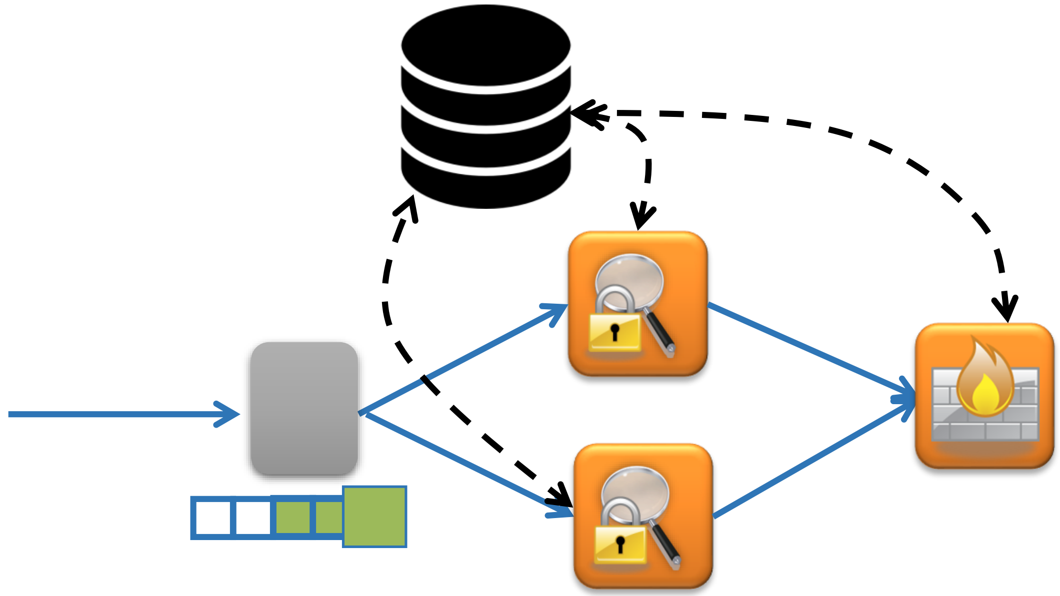
- Failover instance takes over
- Datastore associates the ***failover instance ID*** with the relevant state
- Root ***replays*** the packet



# CHC – Fault Tolerance

NF instance failure recovery:

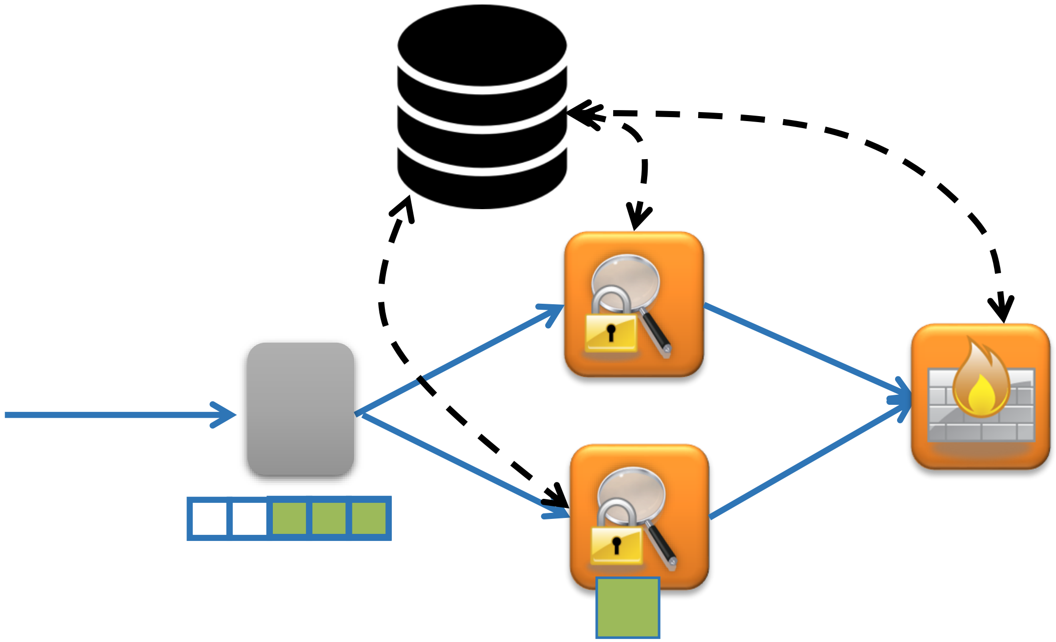
- Failover instance takes over
- Datastore associates the ***failover instance ID*** with the relevant state
- Root ***replays*** the packet



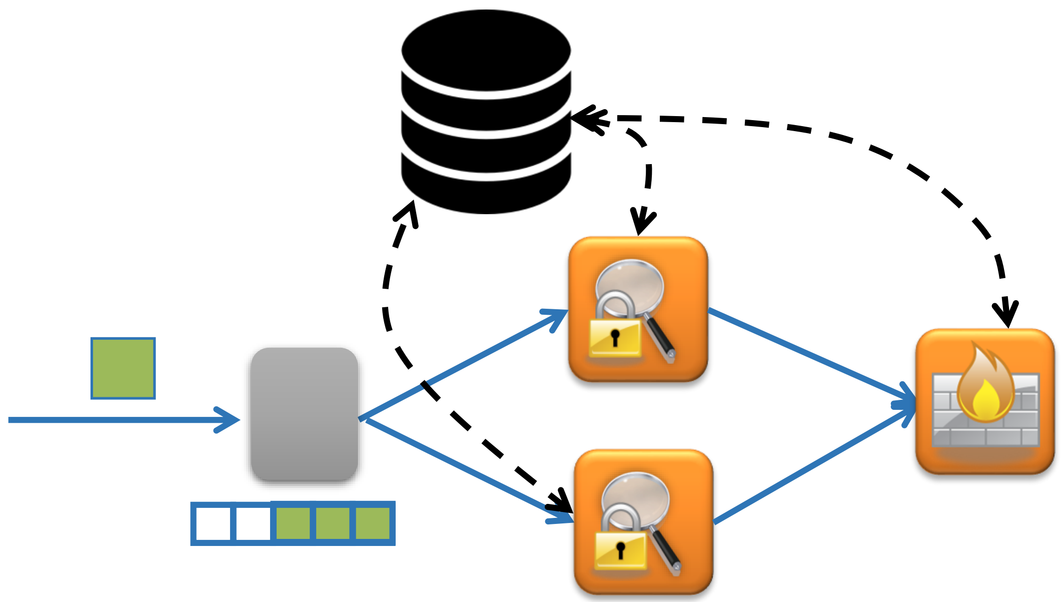
# CHC – Fault Tolerance

NF instance failure recovery:

- Failover instance takes over
- Datastore associates the ***failover instance ID*** with the relevant state
- Root ***replays*** the packet
- Metadata is used to ***suppress duplicate*** state-update and processing

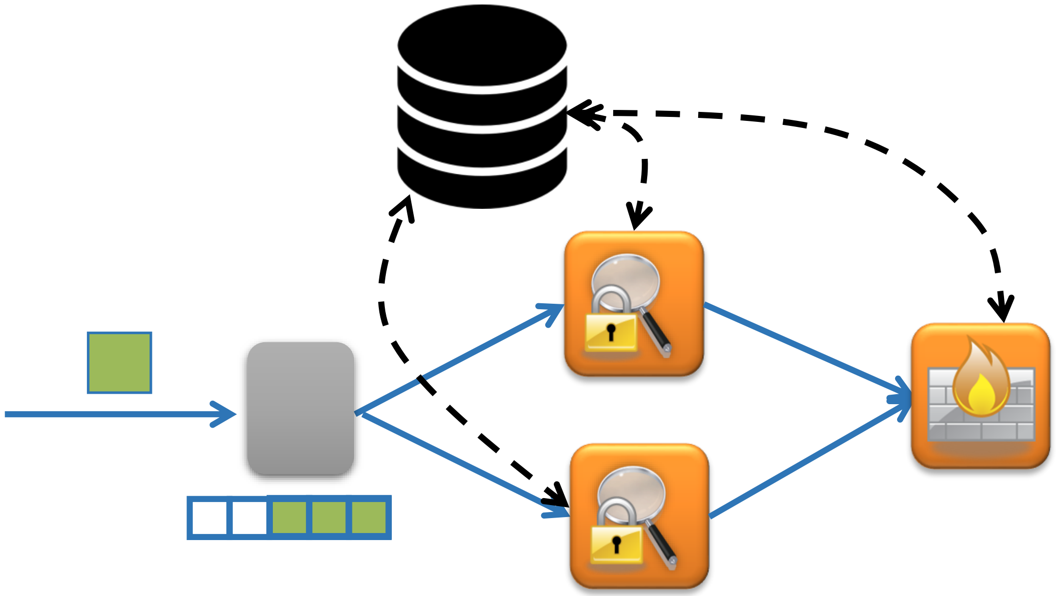


# CHC – Straggler Mitigation



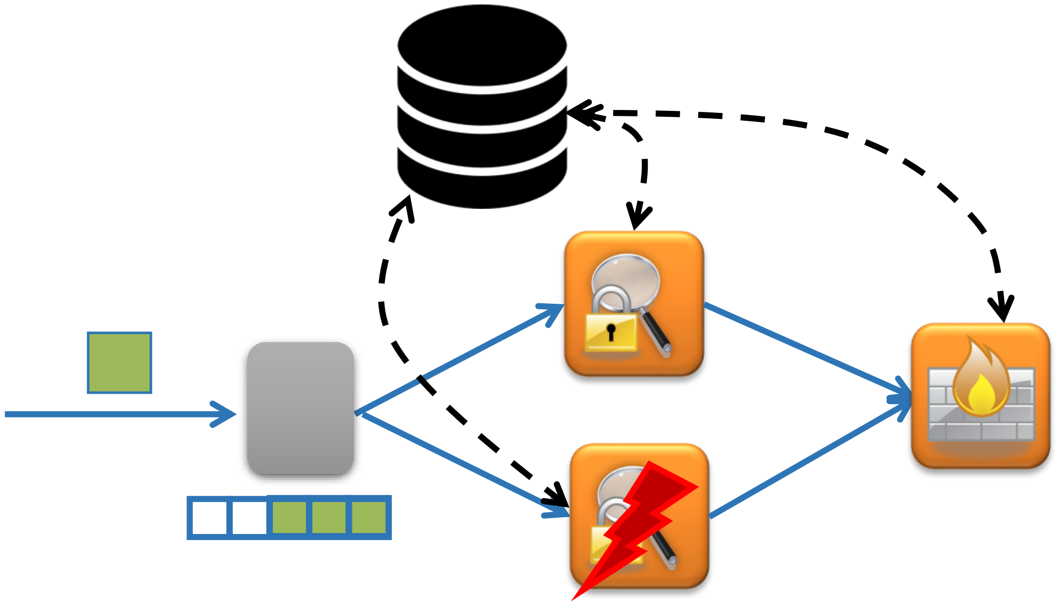
# CHC – Straggler Mitigation

- Metadata (logical clocks) is used to suppress duplicate state updates at the datastore and duplicate packets at downstream NFs



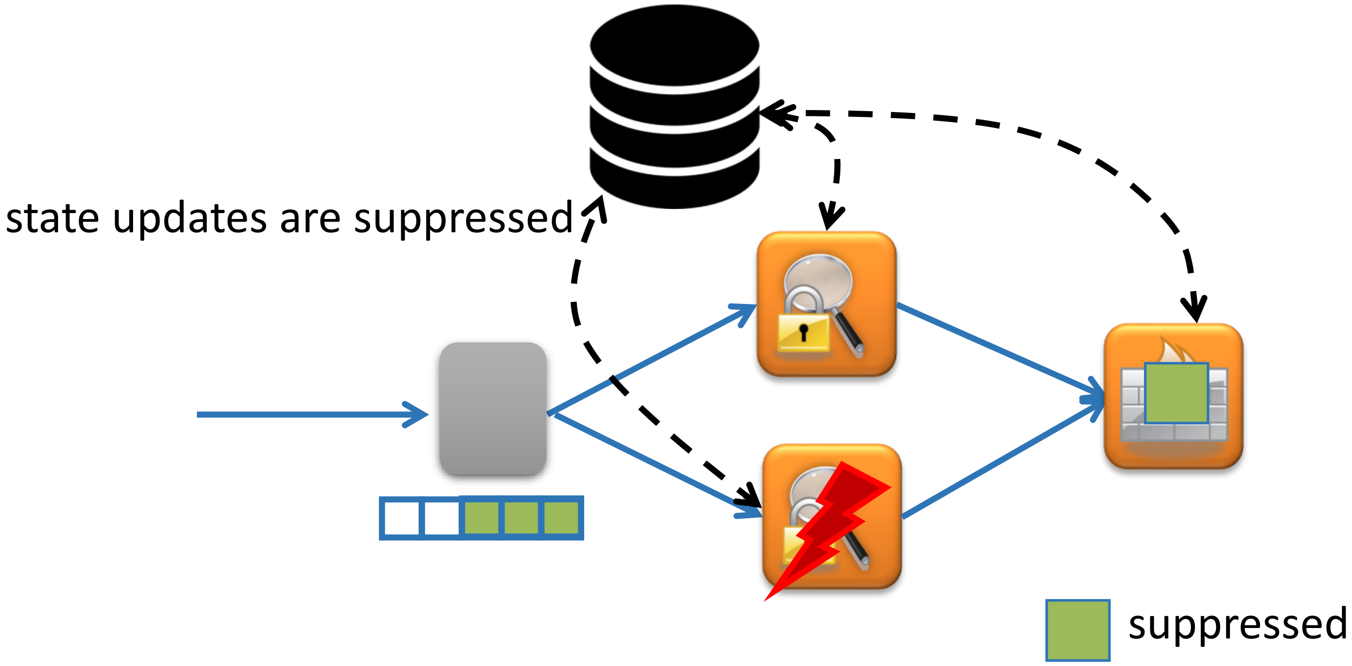
# CHC – Straggler Mitigation

- Metadata (logical clocks) is used to suppress duplicate state updates at the datastore and duplicate packets at downstream NFs



# CHC – Straggler Mitigation

- Metadata (logical clocks) is used to suppress duplicate state updates at the datastore and duplicate packets at downstream NFs



# Implementation of CHC

# Implementation of CHC

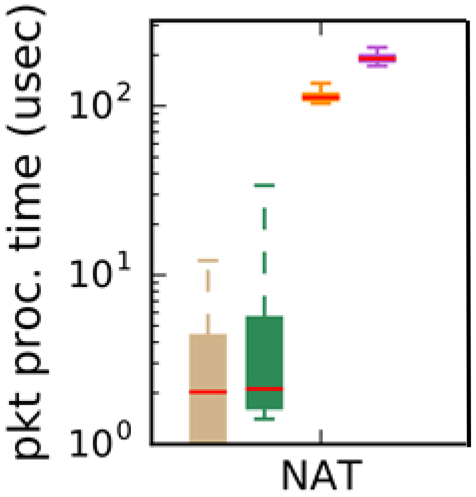
- Prototype is implemented in C++
- Leverages Mellanox messaging accelerator for low latency communication

# Implementation of CHC

- Prototype is implemented in C++
- Leverages Mellanox messaging accelerator for low latency communication

- We implemented four NFs on top of CHC
  - NAT
  - Trojan detector
  - Portscan detector
  - Load balancer

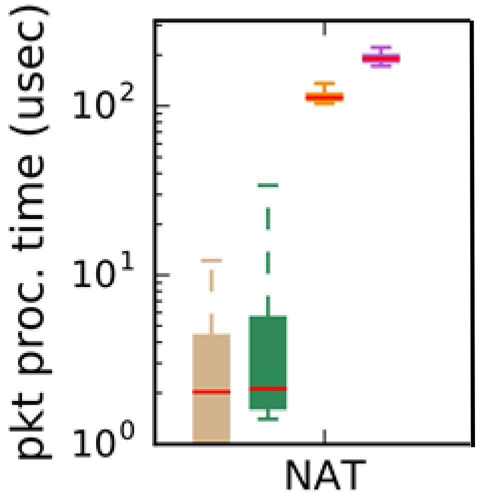
# Evaluation – Performance




| State variable | Scope |
|---|---|
| Port mapping | per-flow |
| Total TCP pkt count | cross flow |
| Total IP pkt count | cross low |

Traditional NF with infinite capacity

# Evaluation – Performance

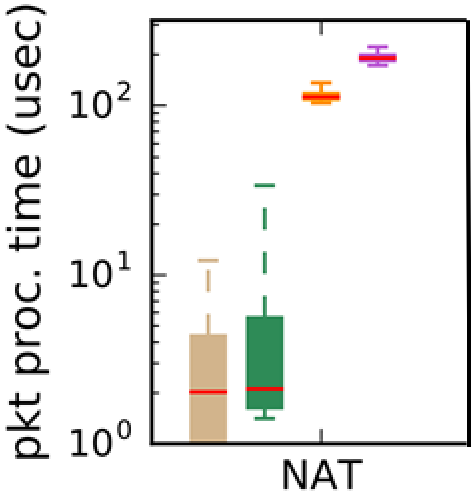



| State variable | Scope | State Externalization |
|---|---|---|
| Port mapping | per-flow | ✔ |
| Total TCP pkt count | cross flow | ✔ |
| Total IP pkt count | cross low | ✔ |

Traditional NF with infinite capacity

Externalized state operations

# Evaluation – Performance



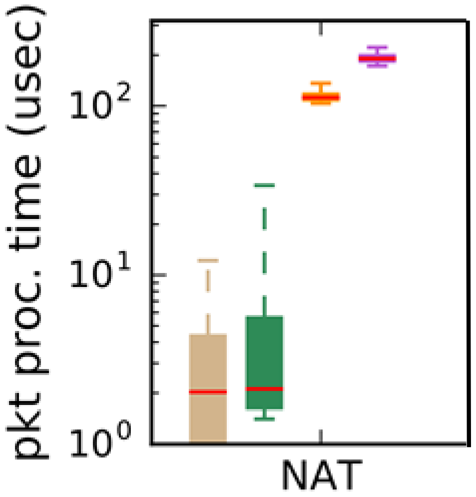

| State variable | Scope | State Externalization | Caching |
|---|---|---|---|
| Port mapping | per-flow | ✓ | ✓ |
| Total TCP pkt count | cross flow | ✓ | |
| Total IP pkt count | cross low | ✓ | |

Traditional NF with infinite capacity

Externalized state operations

State externalization with caching

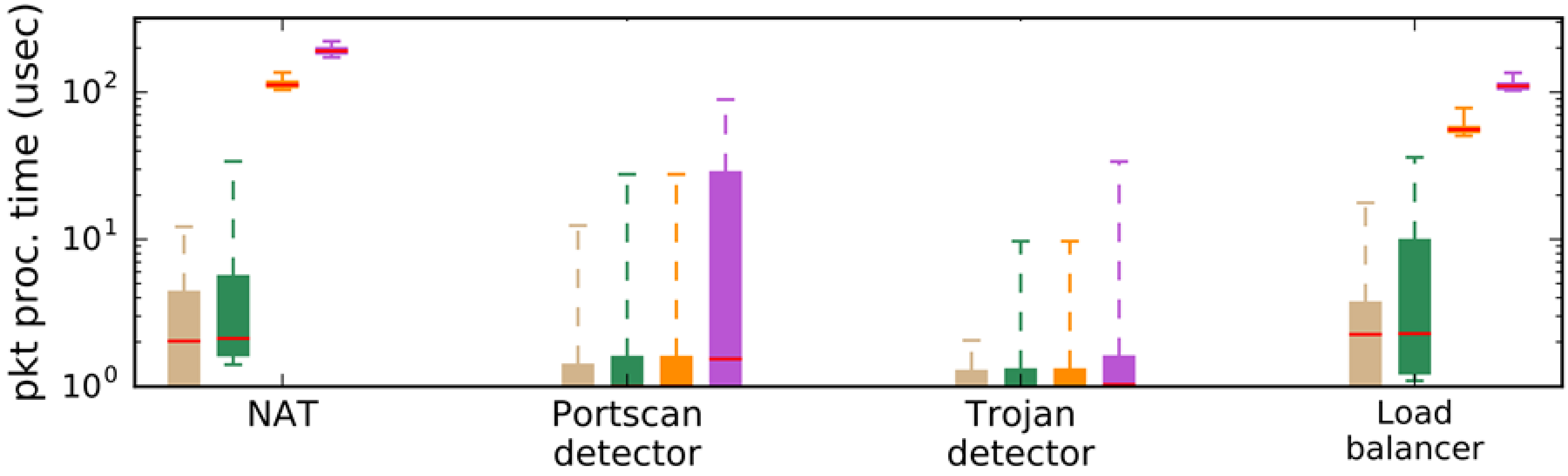
# Evaluation – Performance




| State variable | Scope | State Externalization | Caching | Asynch. + op offload |
|---|---|---|---|---|
| Port mapping | per-flow | ✔ | ✔ | |
| Total TCP pkt count | cross flow | ✔ | | ✔ |
| Total IP pkt count | cross low | ✔ | | ✔ |

- Traditional NF with infinite capacity
- Externalized state operations
- State externalization with caching
- State externalization with caching and asynchronous + offloaded updates

# Evaluation – Performance




Traditional NF with infinite capacity

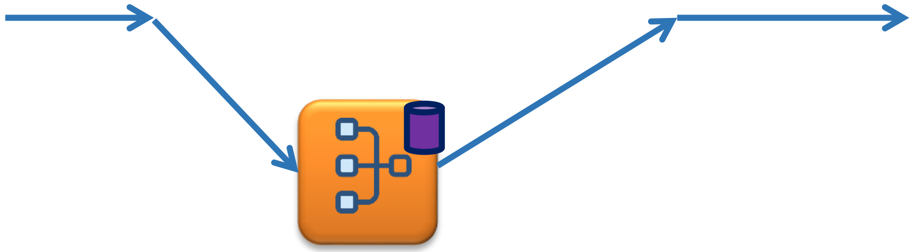
Externalized state operations

State externalization with caching

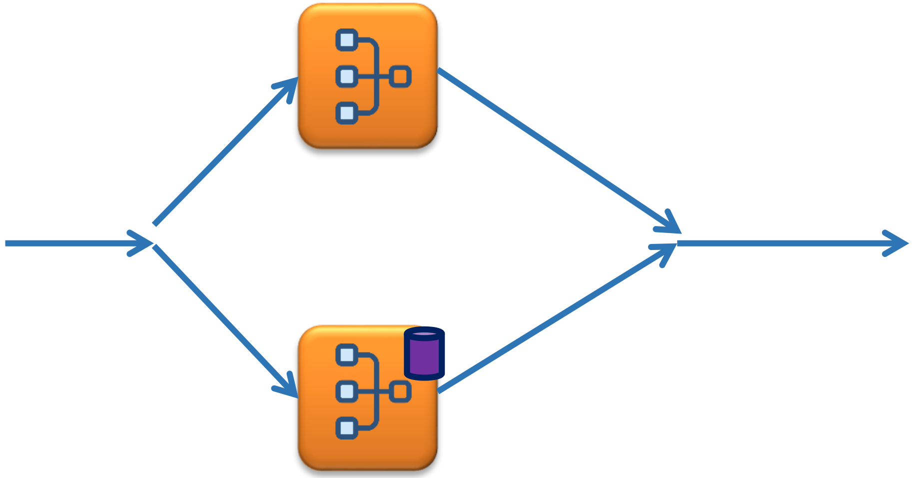
State externalization with caching and asynchronous + offloaded updates

Less than ***0.6μs*** increase in the median per-NF packet processing latency

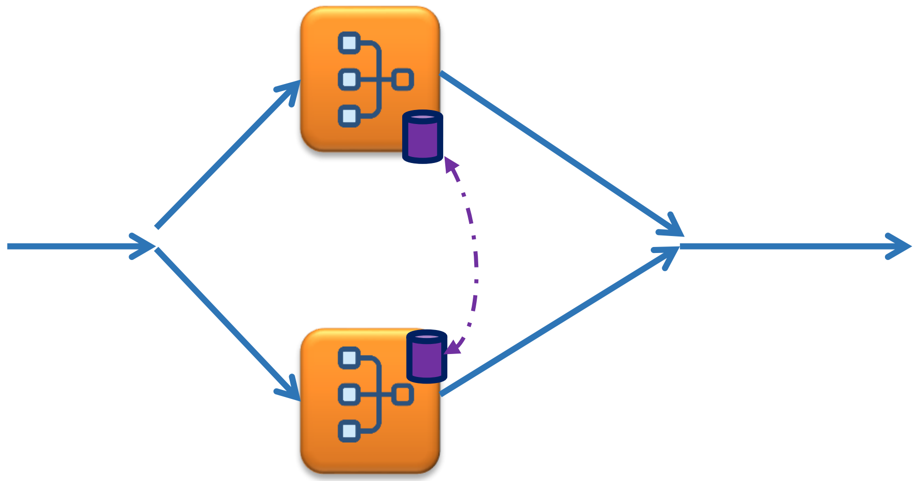
# Evaluation – Dynamic Actions



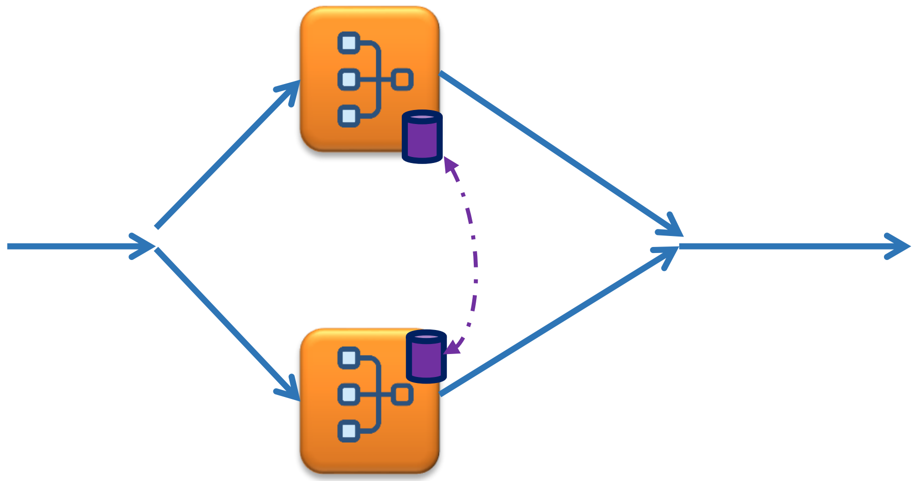
# Evaluation – Dynamic Actions



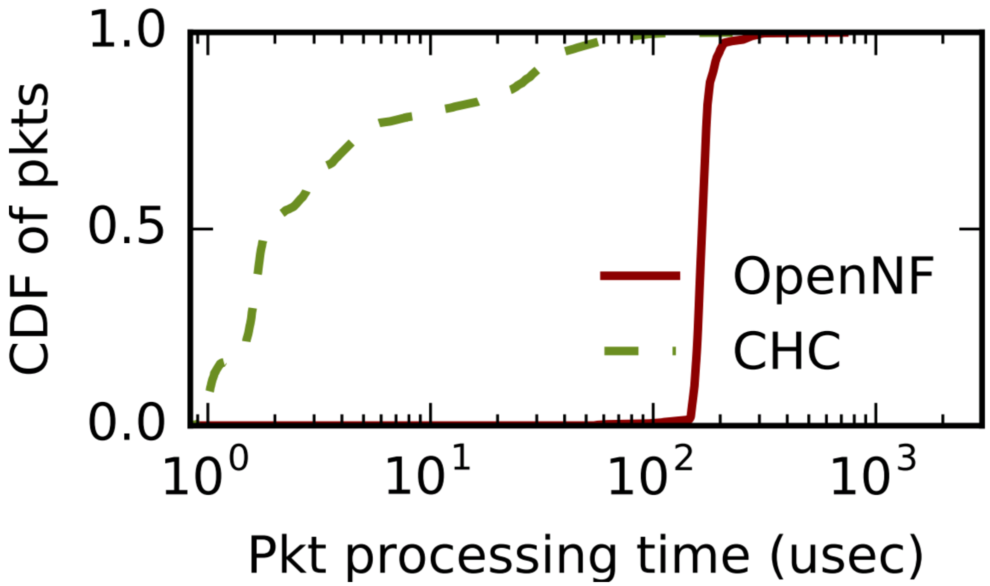
# Evaluation – Dynamic Actions



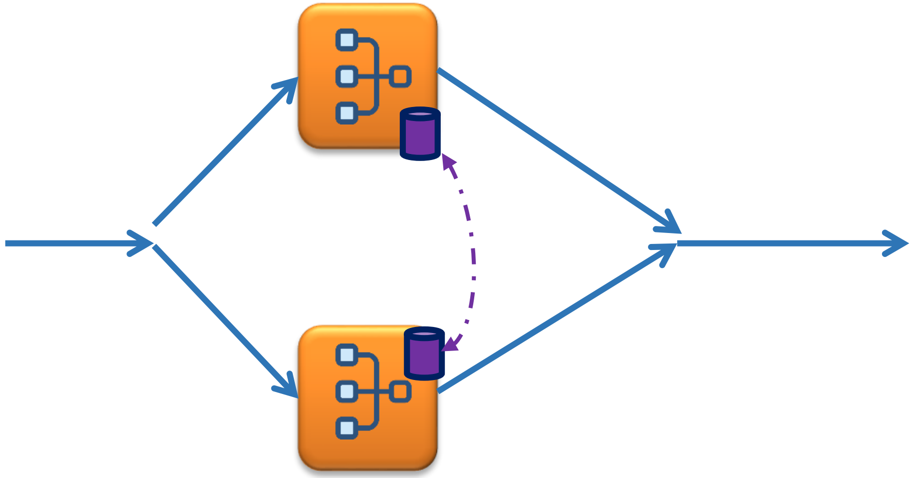
# Evaluation – Dynamic Actions



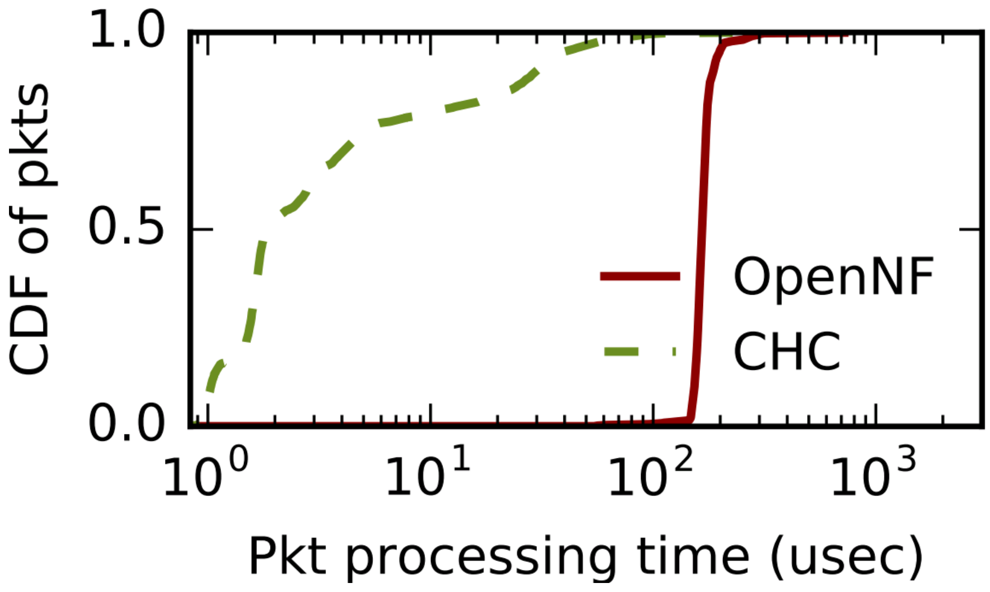
During cross instance state sharing



# Evaluation – Dynamic Actions



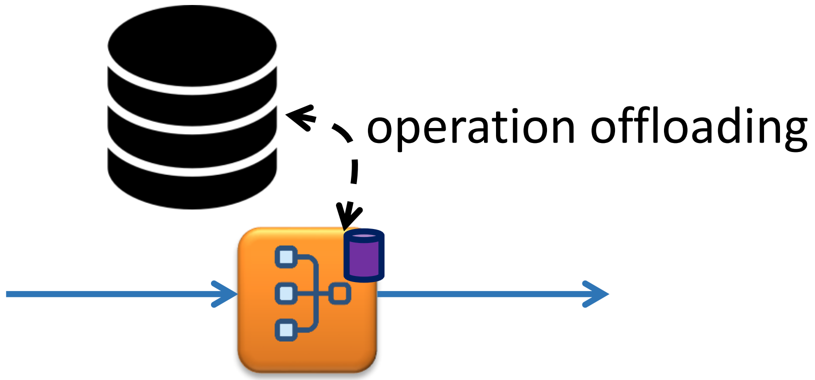
During cross instance state sharing




75th%-ile latency of CHC is ***20 times* lower than** OpenNF

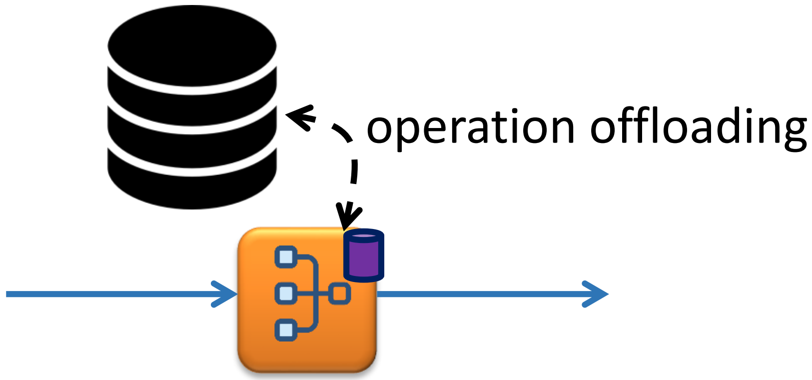
# Evaluation – Dynamic Actions

CHC

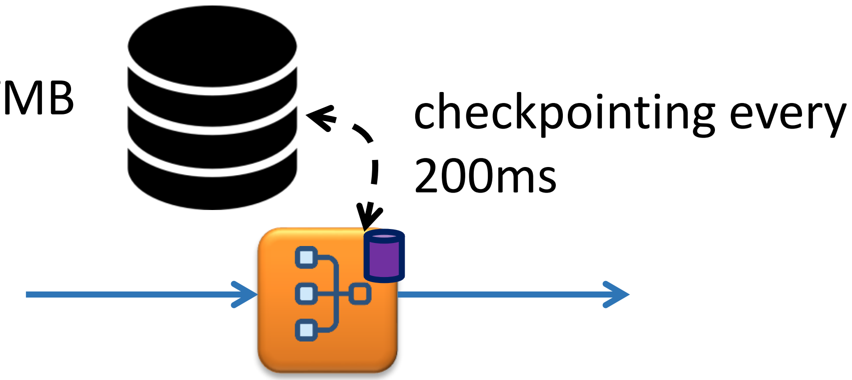


# Evaluation – Dynamic Actions

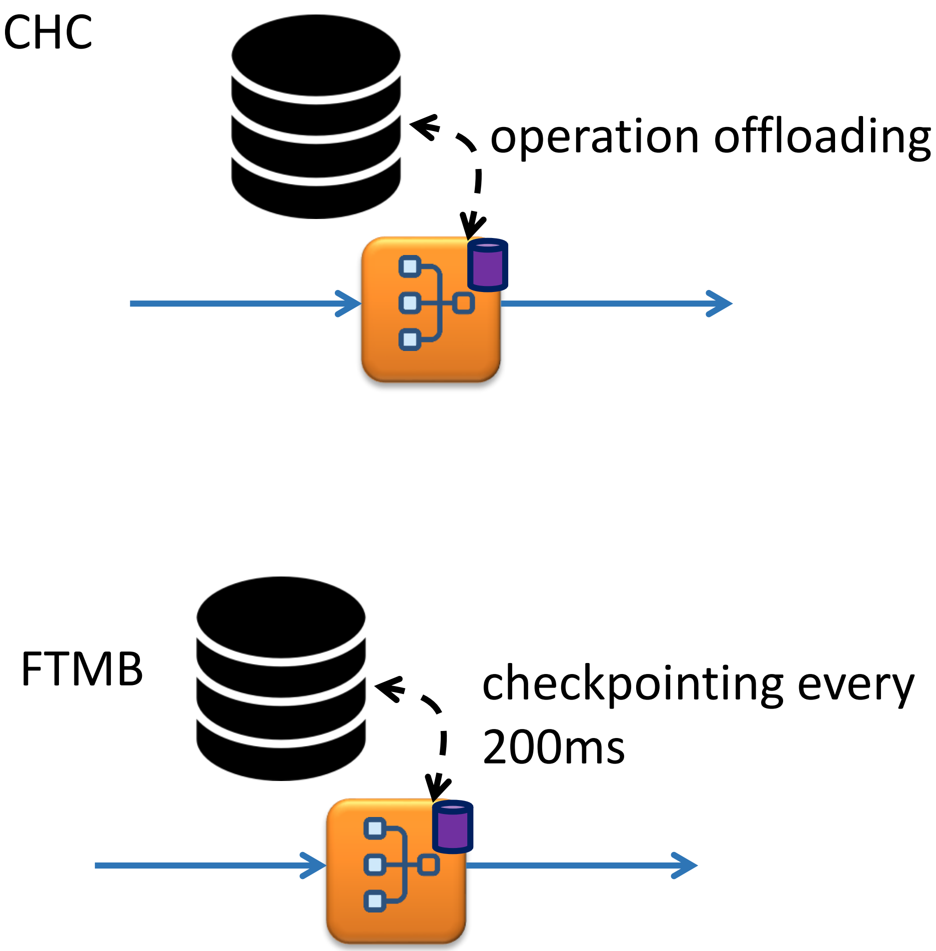
CHC



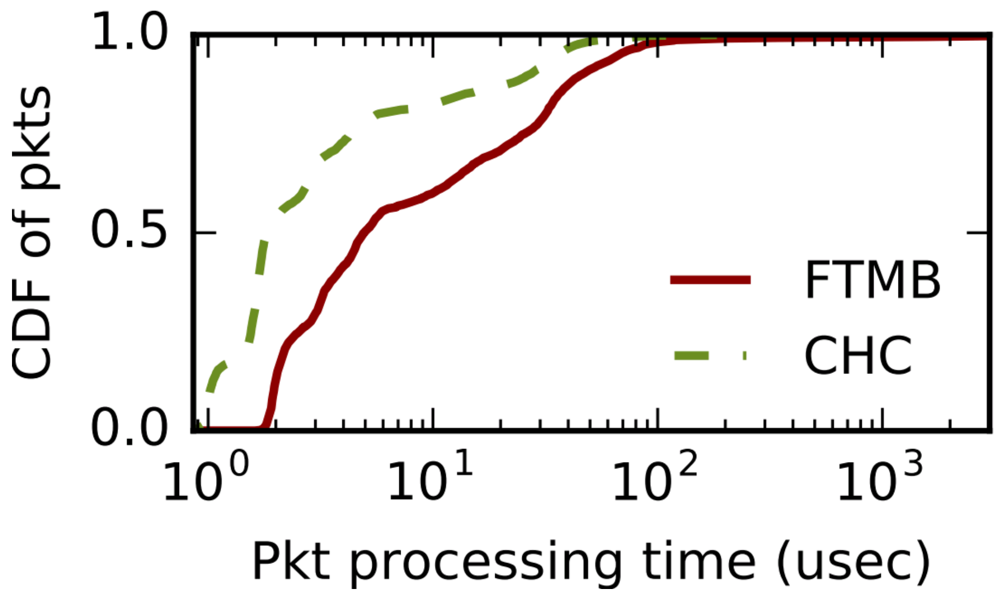

FTMB



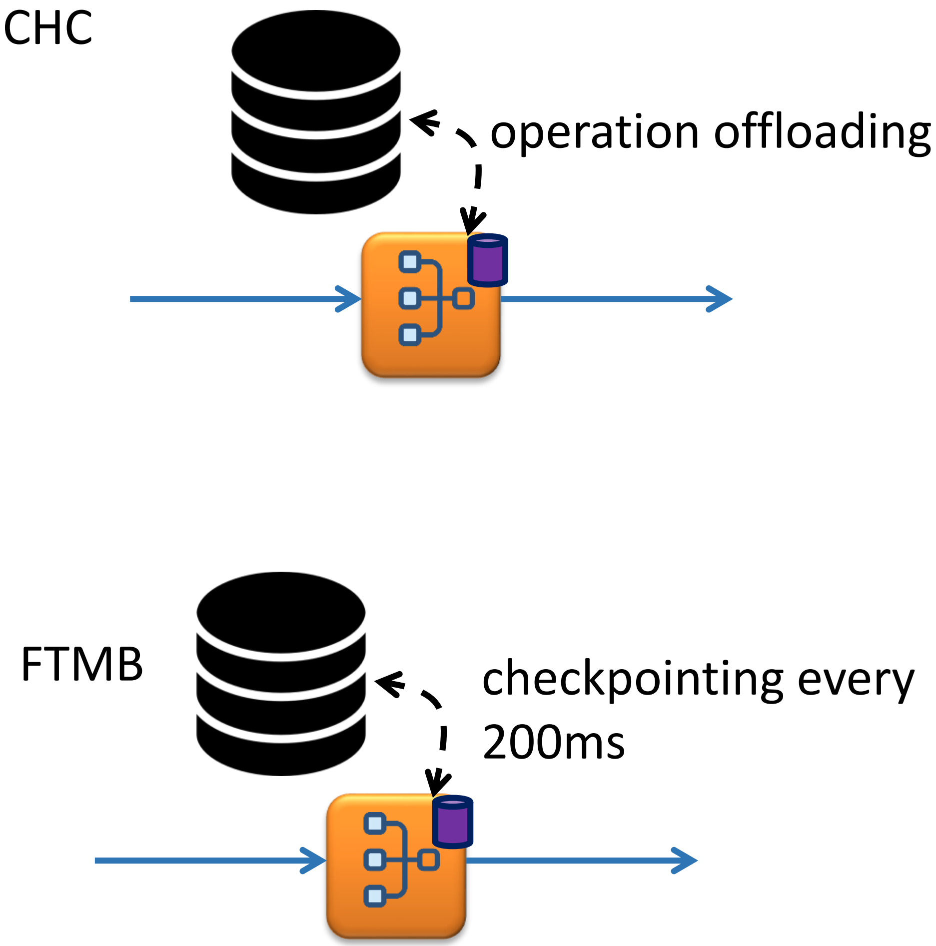
# Evaluation – Dynamic Actions



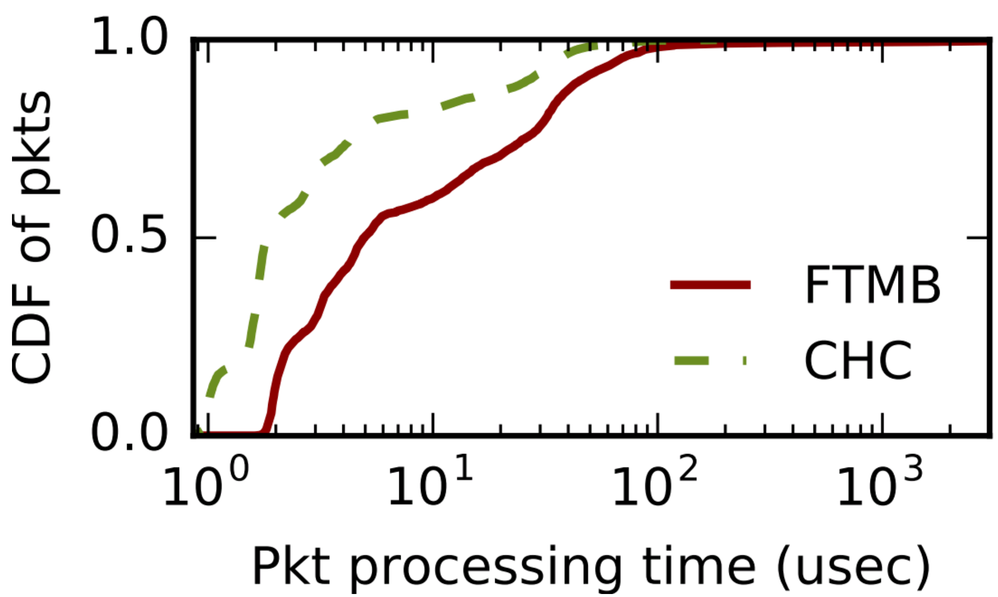

Ensuing Fault tolerance



# Evaluation – Dynamic Actions

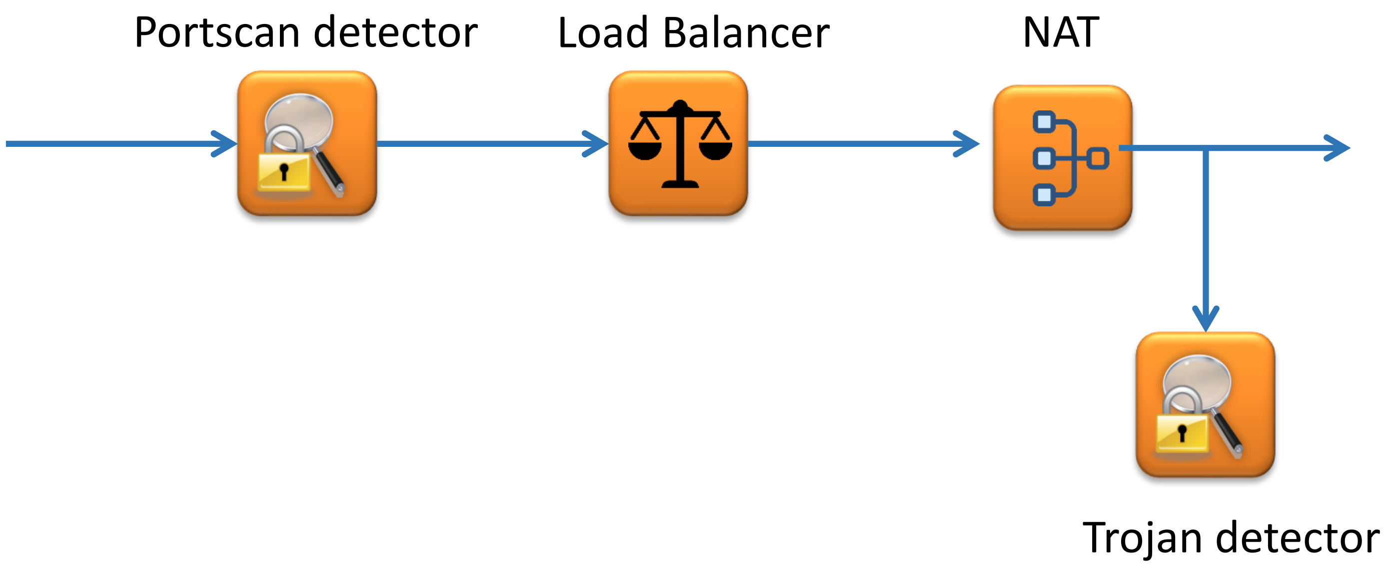



Ensuing Fault tolerance




75th%-ile latency of CHC is ***6 times* lower than** FTMB

# Evaluation




CHC operates at line rate with an end-to-end median per packet processing overhead of ***11.3us***

# Evaluation

- State management performance
- Metadata overhead
- Correctness requirements:
  - State availability
  - Cross instance state transfer
  - Cross instance state sharing
  - Chain wide ordering
  - Duplication suppression
  - Fault tolerance

# Summary

- CHC supports output equivalence and high performance state management for NFV chains
- It hides the complexity of handling states during dynamic actions (elastic scaling and failure recovery)
- It relies on managing state external to NFs, but couples it with several caching and state update algorithms to ensure low latency