Fault Scalable Virtualized Infrastructure Management

Mukil Kesavan, Ada Gavrilovska, Karsten Schwan

VMware Inc., Georgia Tech

Typical Virtualized Datacenter Workloads

- Cluster Capacity Allocation
- Live Rolling Upgrades
- Backup/Restore for Disaster Recovery
- Continuous Integration Testing
- Generic: AWS CloudFormation, VMWare vRealize, OpenStack Heat

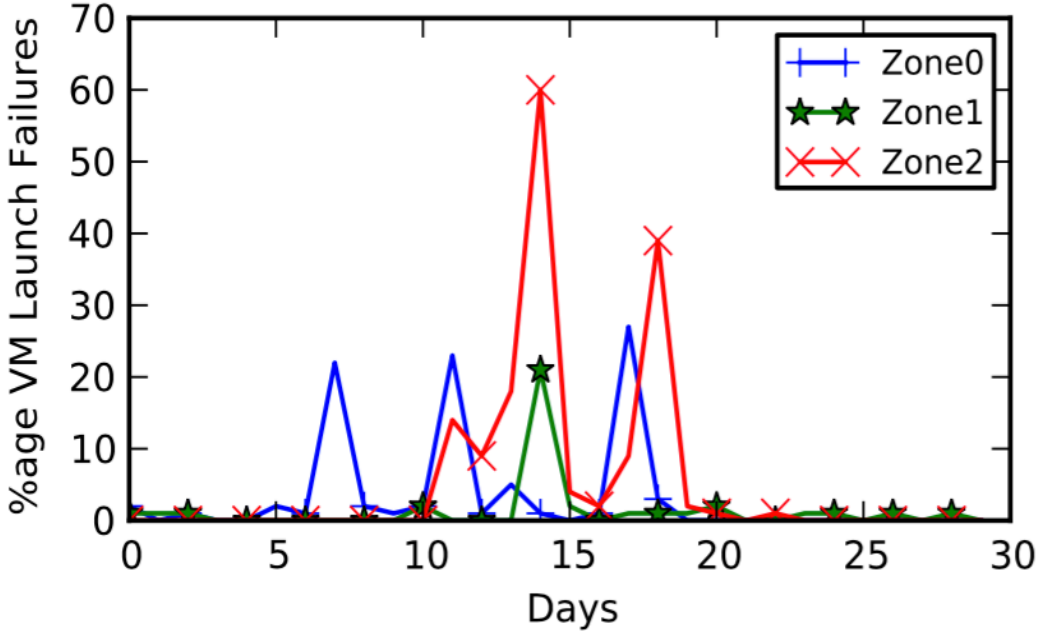
Management Workloads

=> Composed as workflows consisting of basic VM operations

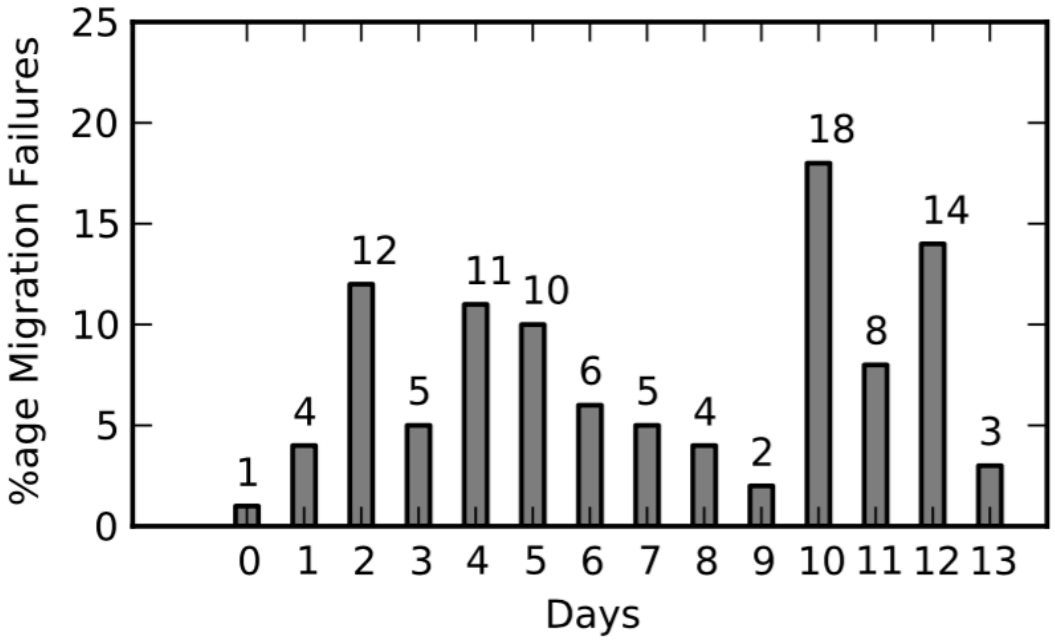
Basic Virtual Machine Operations

- Power-on Placement
- Live Migration
- State Snapshotting Ops
- Cloning
- Re-configuration (e.g. hot-add CPU, memory)

Basic VM Operation Failures



(a) VM instance launch in HP Public Cloud.



(b) VM migration failures at Georgia Tech.

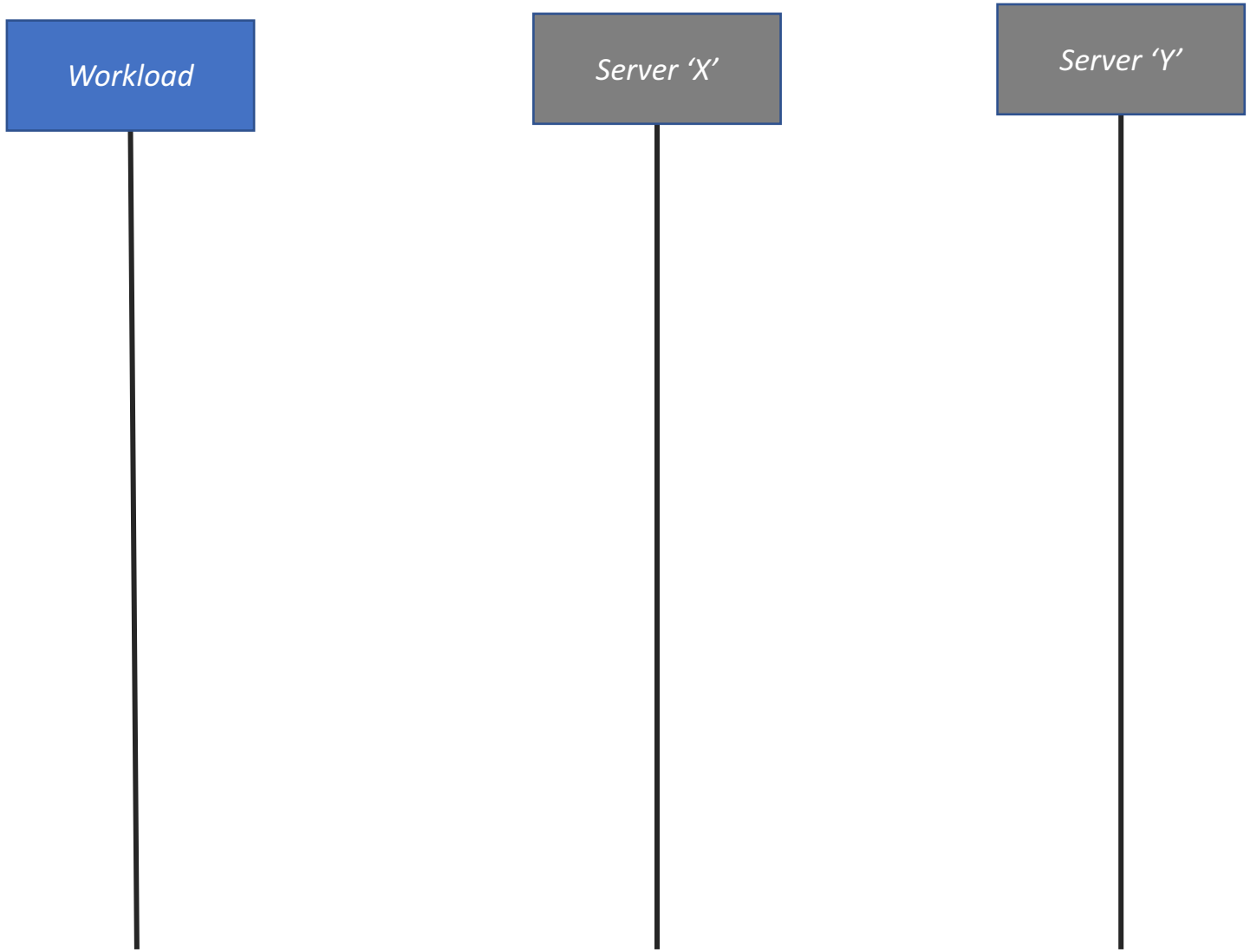
Non-trivial, highly variable basic VM op failures impair workloads!

<u>Problem</u>: How to Improve DC Workload Outcomes despite Failures?

State of the art solutions

- Root-cause Diagnosis
  - With wide-variety of causes => Extremely complicated to do
  - High fix and turnaround times
- Tolerate Infrastructure Failures at Application Layer
  - Usually just naïve backoff-retry based
  - Burdens developers
  - Lack of infra info to employ intelligent resiliency techniques

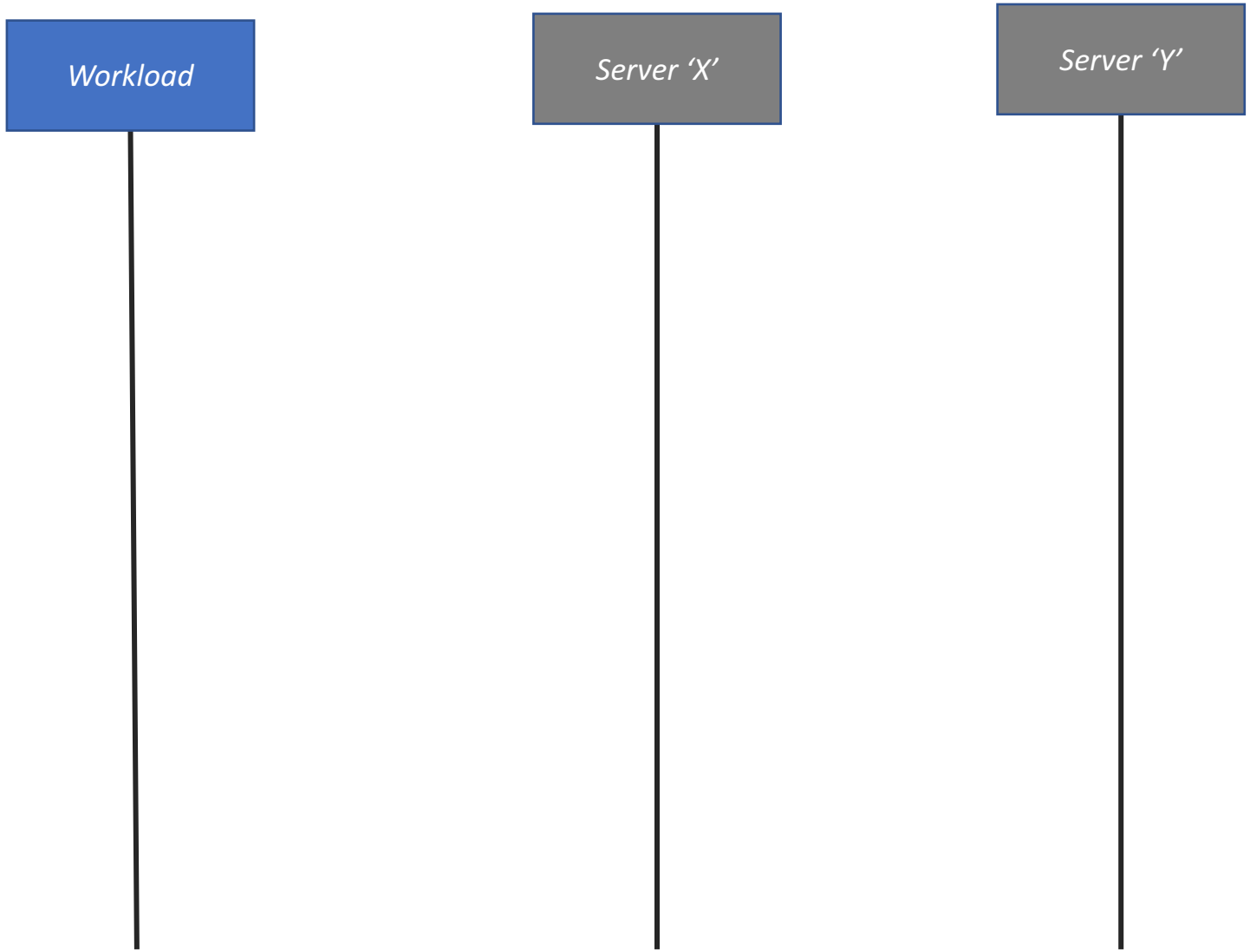
Naïve Retrying



Naïve Retrying

Invocation ‘**N**’:

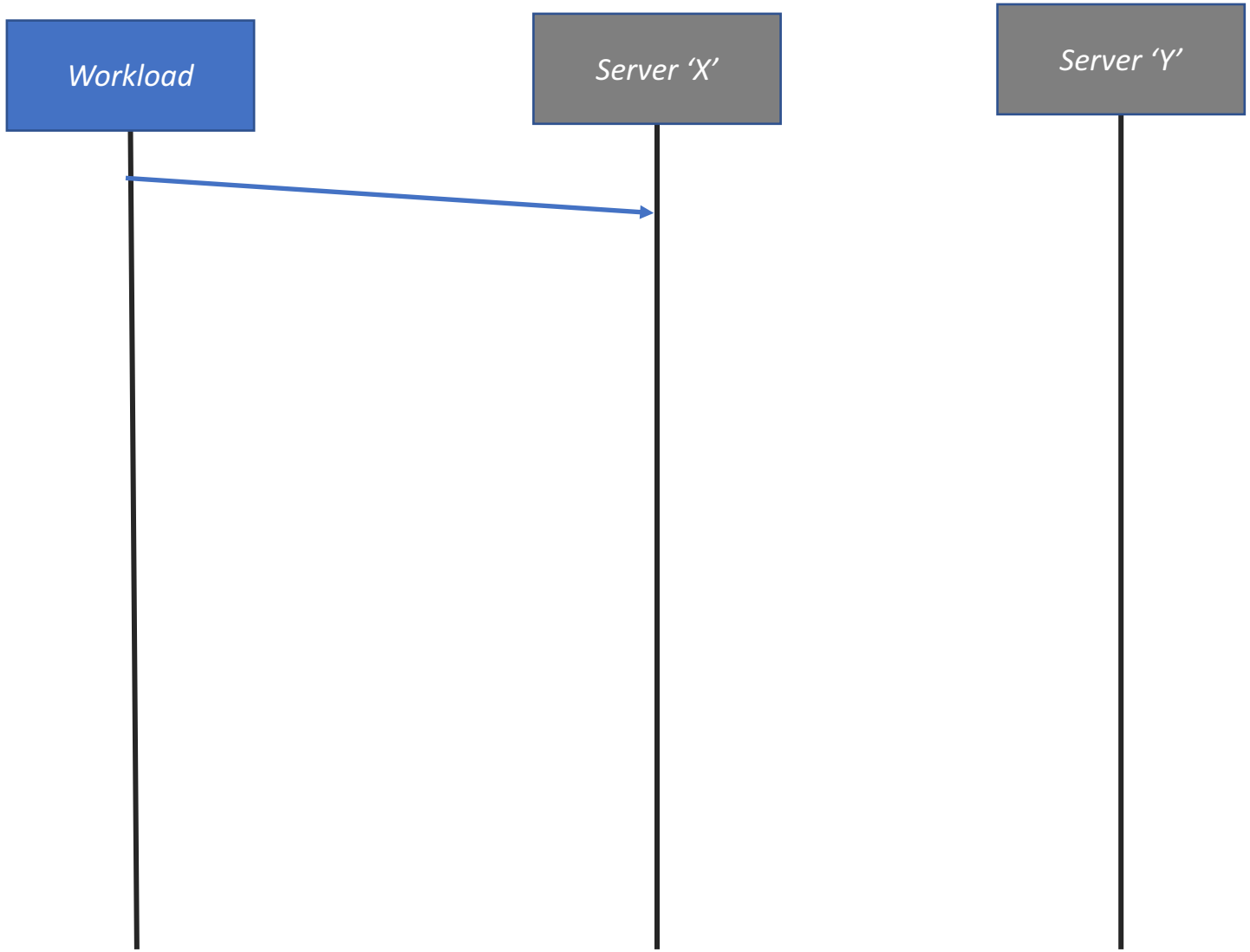
1. Run Algo
2. Do Op ‘p’ on VM ‘v’ on Server ‘**X**’



Naïve Retrying

Invocation ‘**N**’:

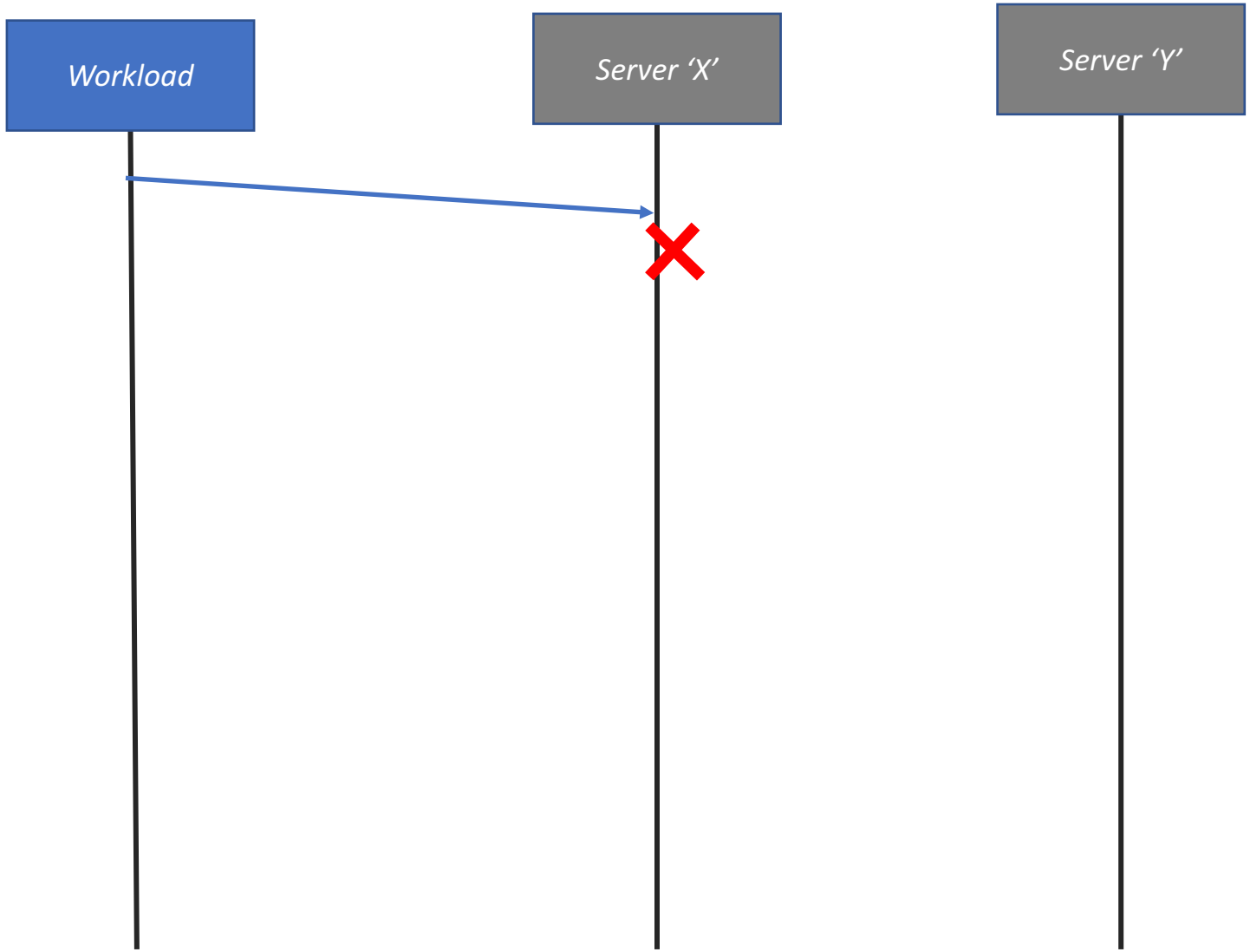
1. Run Algo
2. Do Op ‘p’ on VM ‘v’ on Server ‘**X**’



Naïve Retrying

Invocation ‘**N**’:

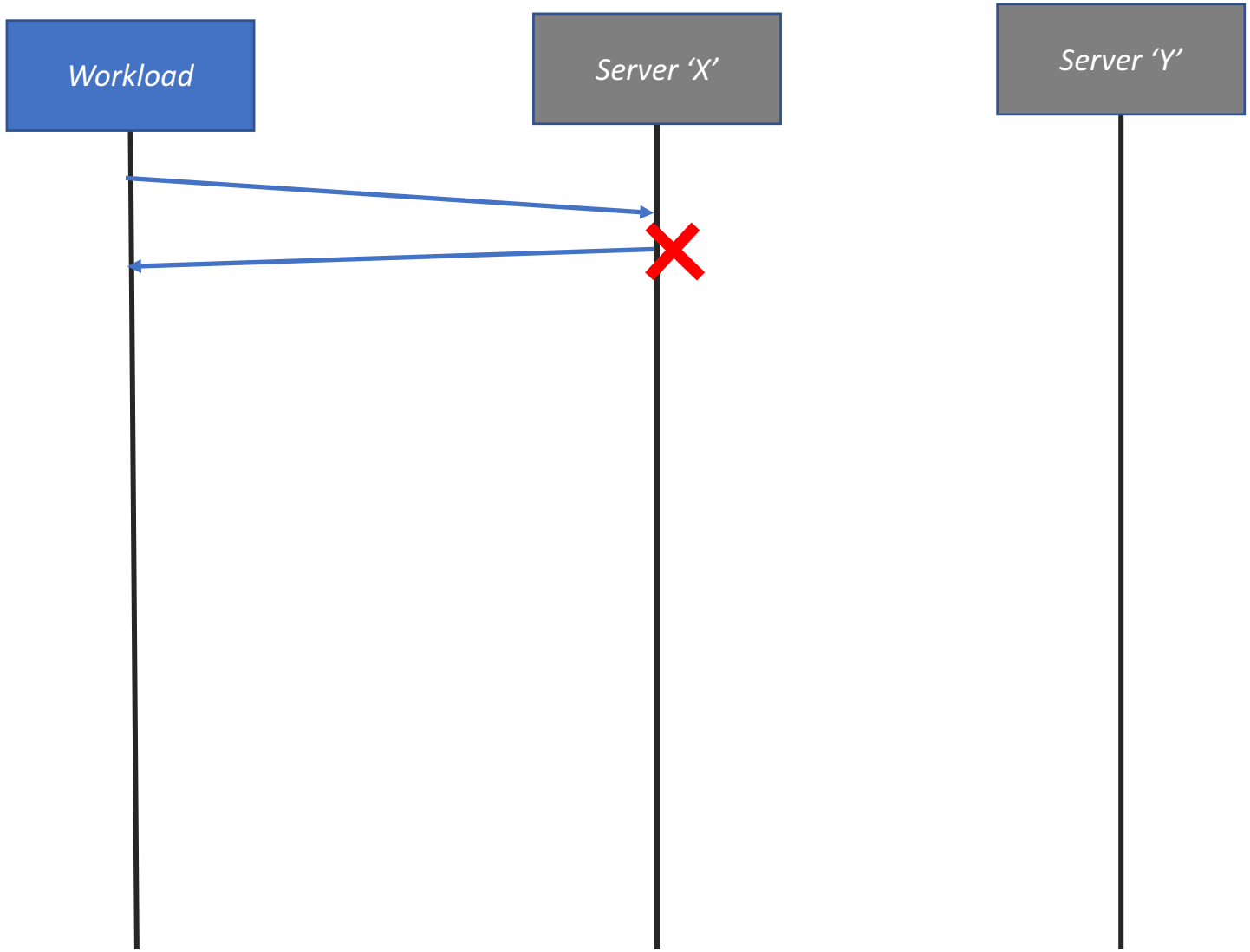
1. Run Algo
2. Do Op ‘p’ on VM ‘v’ on Server ‘**X**’



Naïve Retrying

Invocation ‘**N**’:

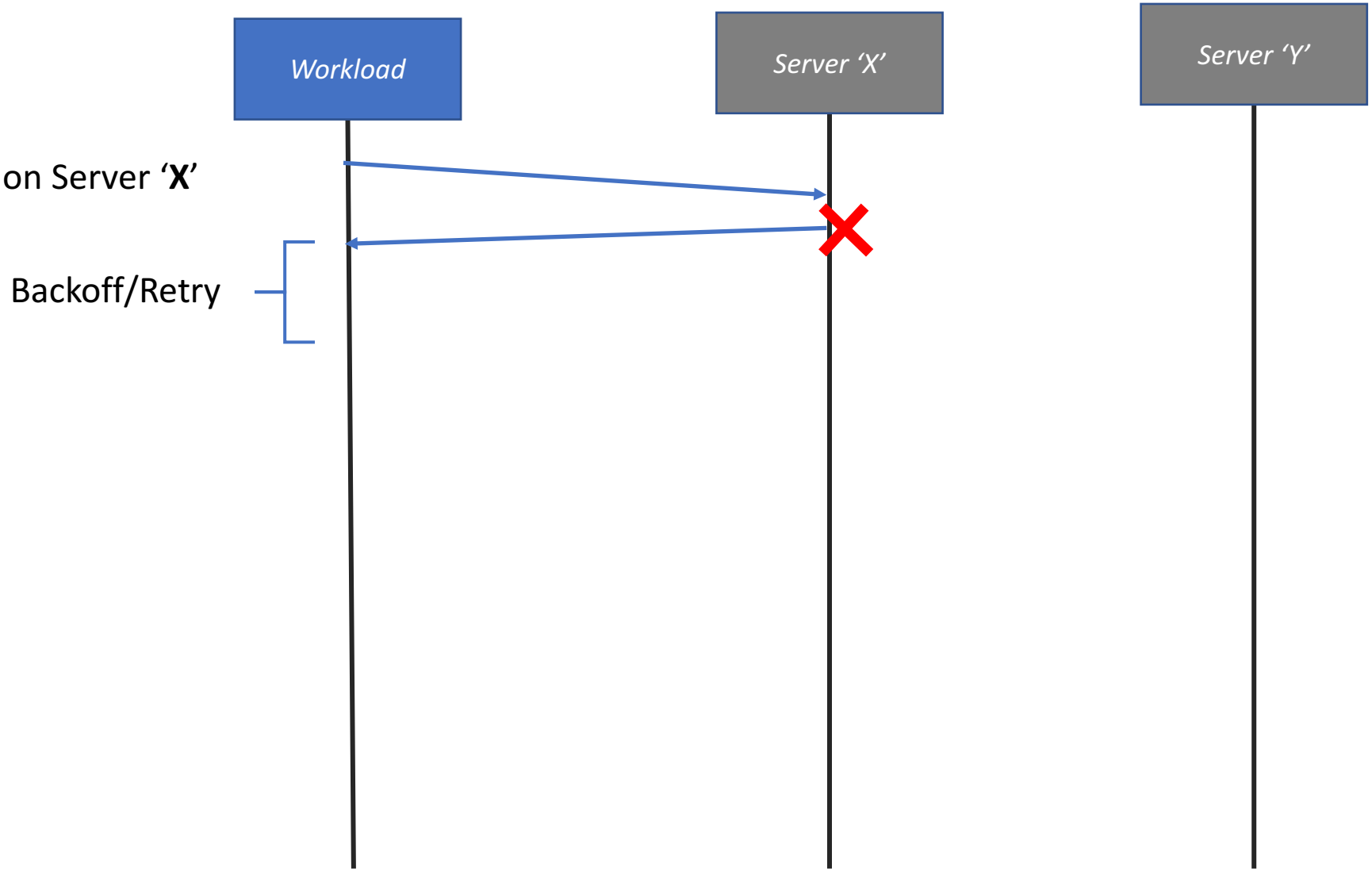
1. Run Algo
2. Do Op ‘p’ on VM ‘v’ on Server ‘**X**’



Naïve Retrying

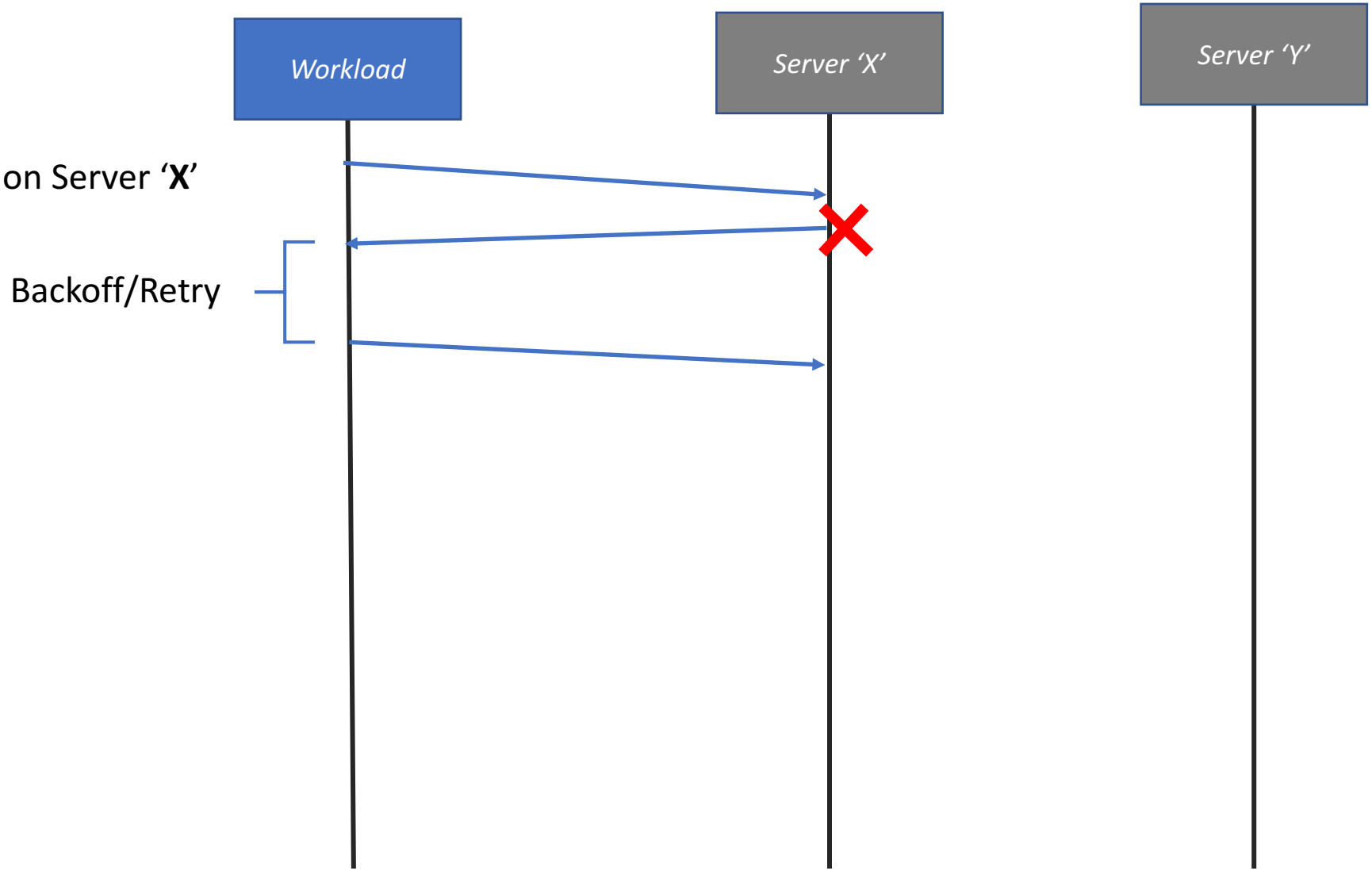
Invocation '**N**':

1. Run Algo
2. Do Op 'p' on VM 'v' on Server '**X**'



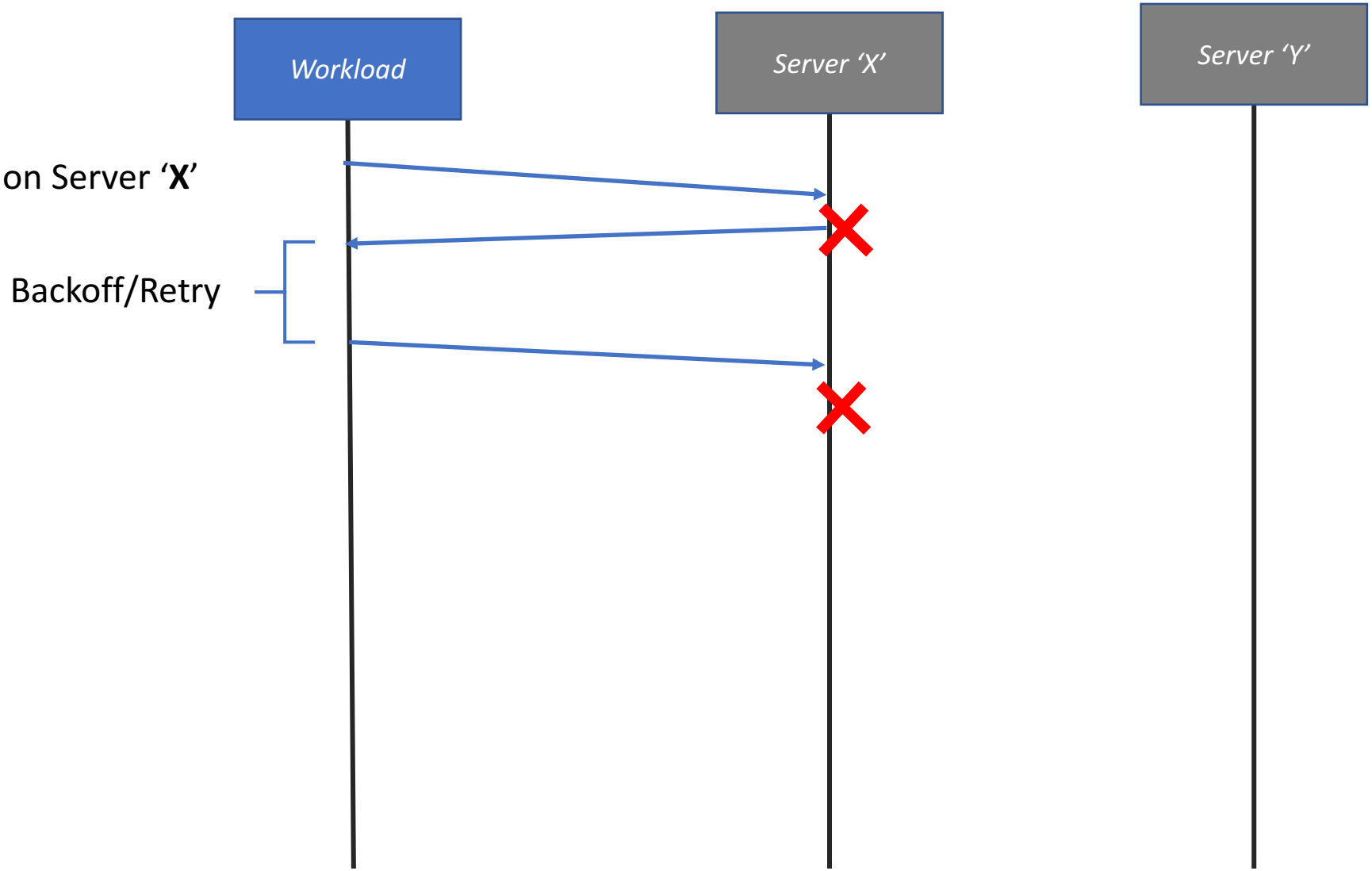
Naïve Retrying

Invocation '**N**':
1. Run Algo
2. Do Op 'p' on VM 'v' on Server '**X**'



Naïve Retrying

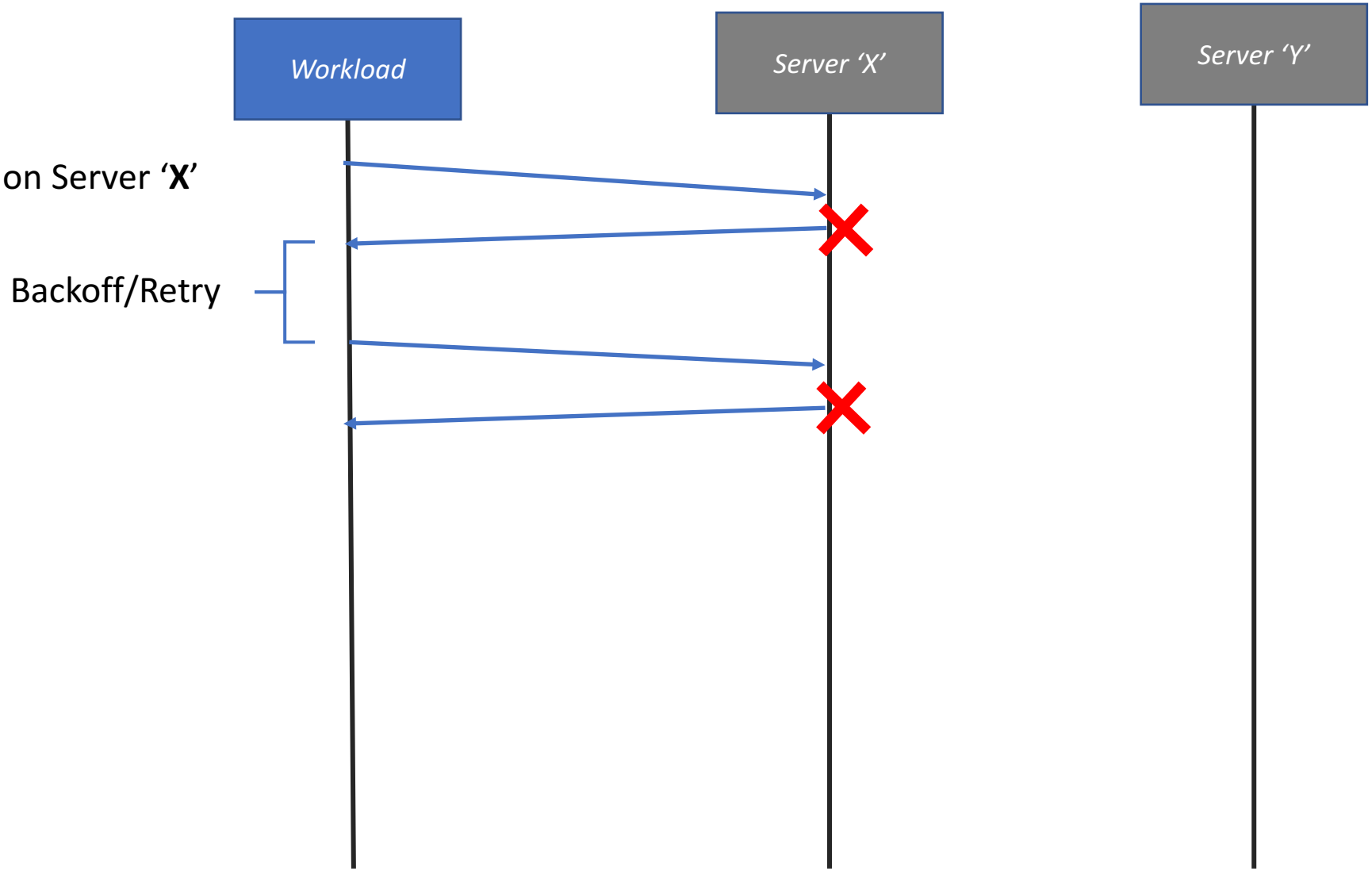
Invocation '**N**':
1. Run Algo
2. Do Op 'p' on VM 'v' on Server '**X**'



Naïve Retrying

Invocation '**N**':

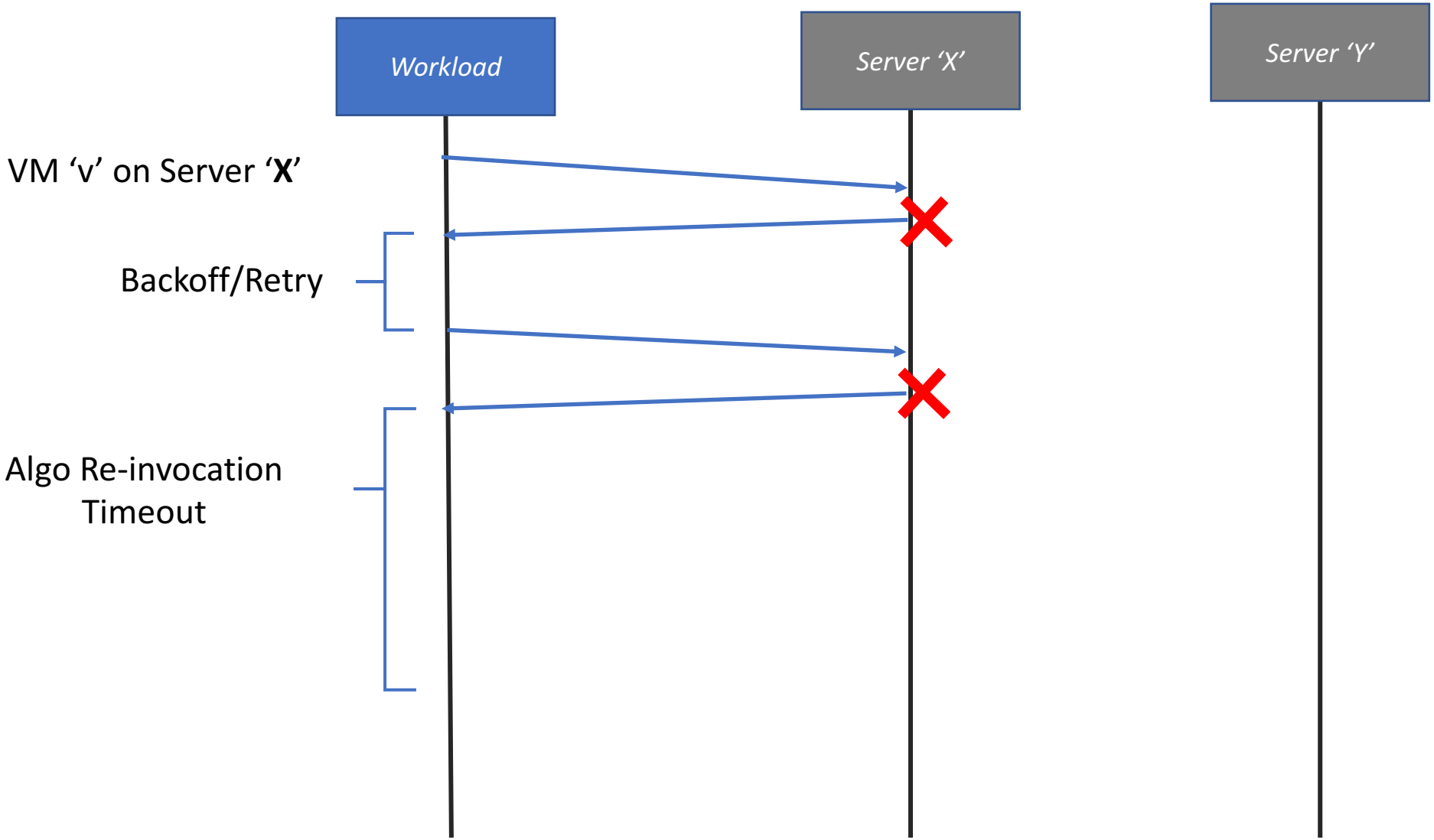
1. Run Algo
2. Do Op 'p' on VM 'v' on Server '**X**'



Naïve Retrying

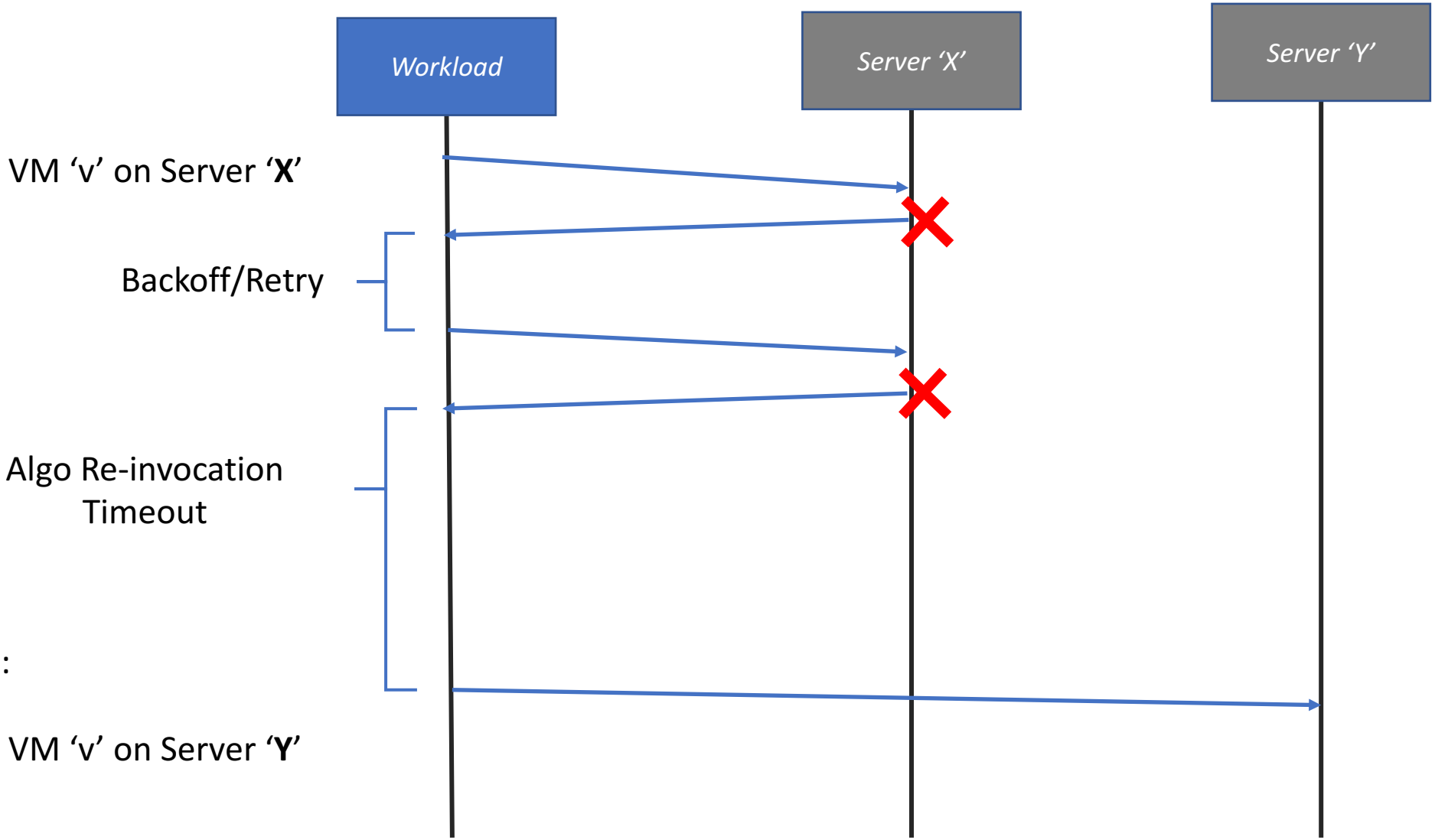
Invocation '**N**':

1. Run Algo
2. Do Op 'p' on VM 'v' on Server '**X**'



Naïve Retrying

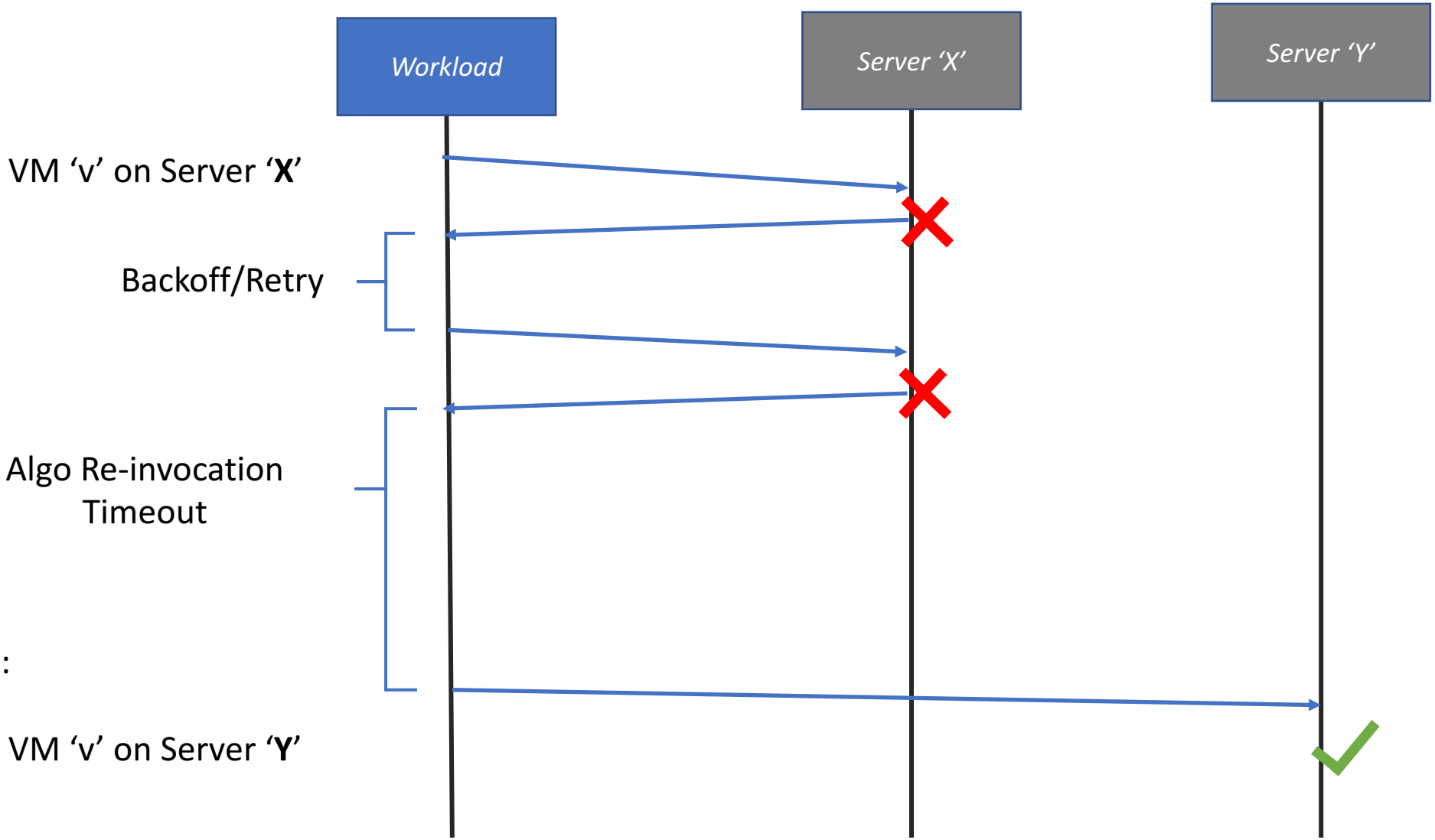
Invocation '**N**':
1. Run Algo
2. Do Op 'p' on VM 'v' on Server '**X**'



Invocation '**N+1**':
1. Run Algo
2. Do Op 'p' on VM 'v' on Server '**Y**'

Naïve Retrying

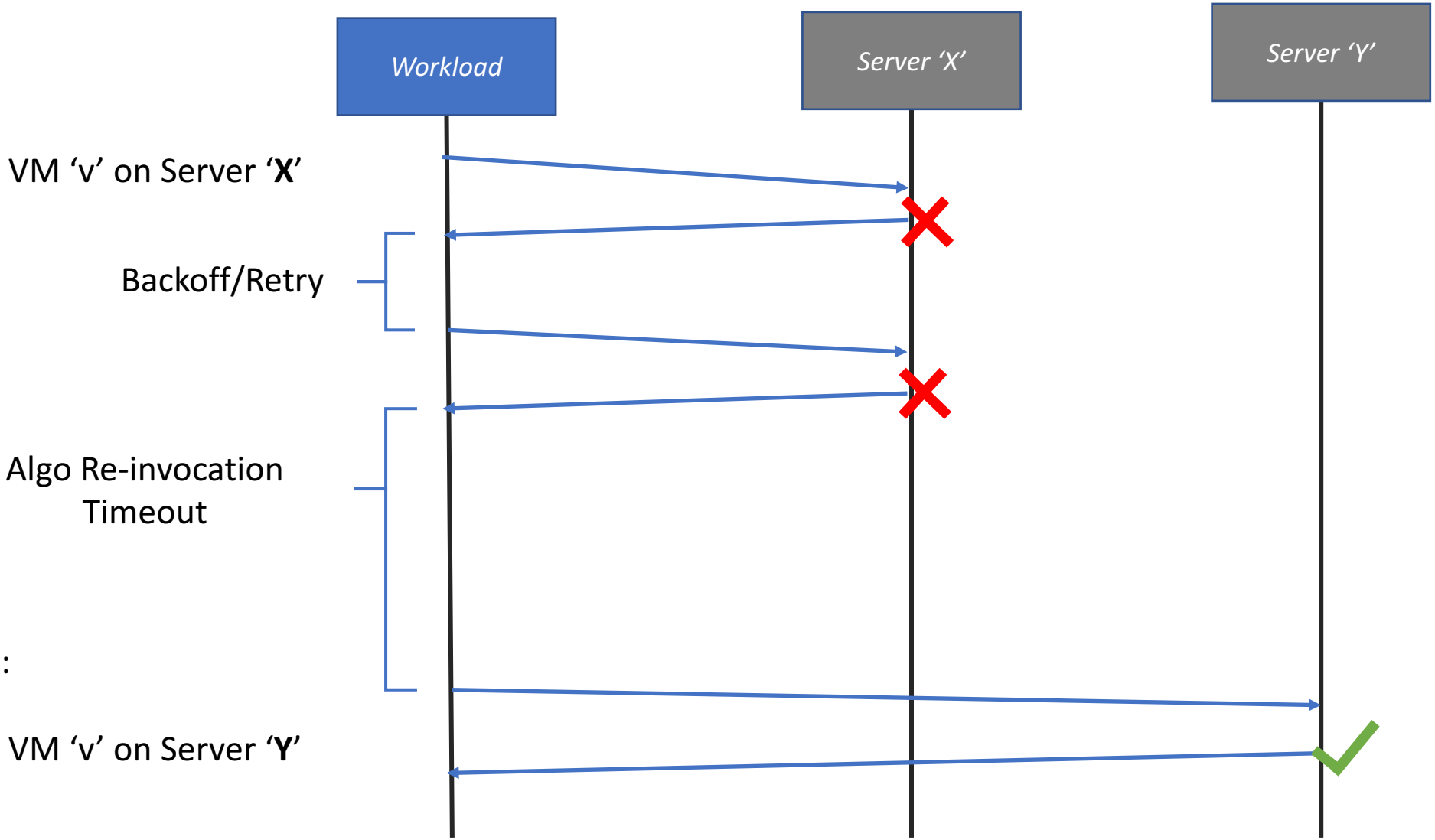
Invocation '**N**':
1. Run Algo
2. Do Op 'p' on VM 'v' on Server '**X**'



Invocation '**N+1**':
1. Run Algo
2. Do Op 'p' on VM 'v' on Server '**Y**'

Naïve Retrying

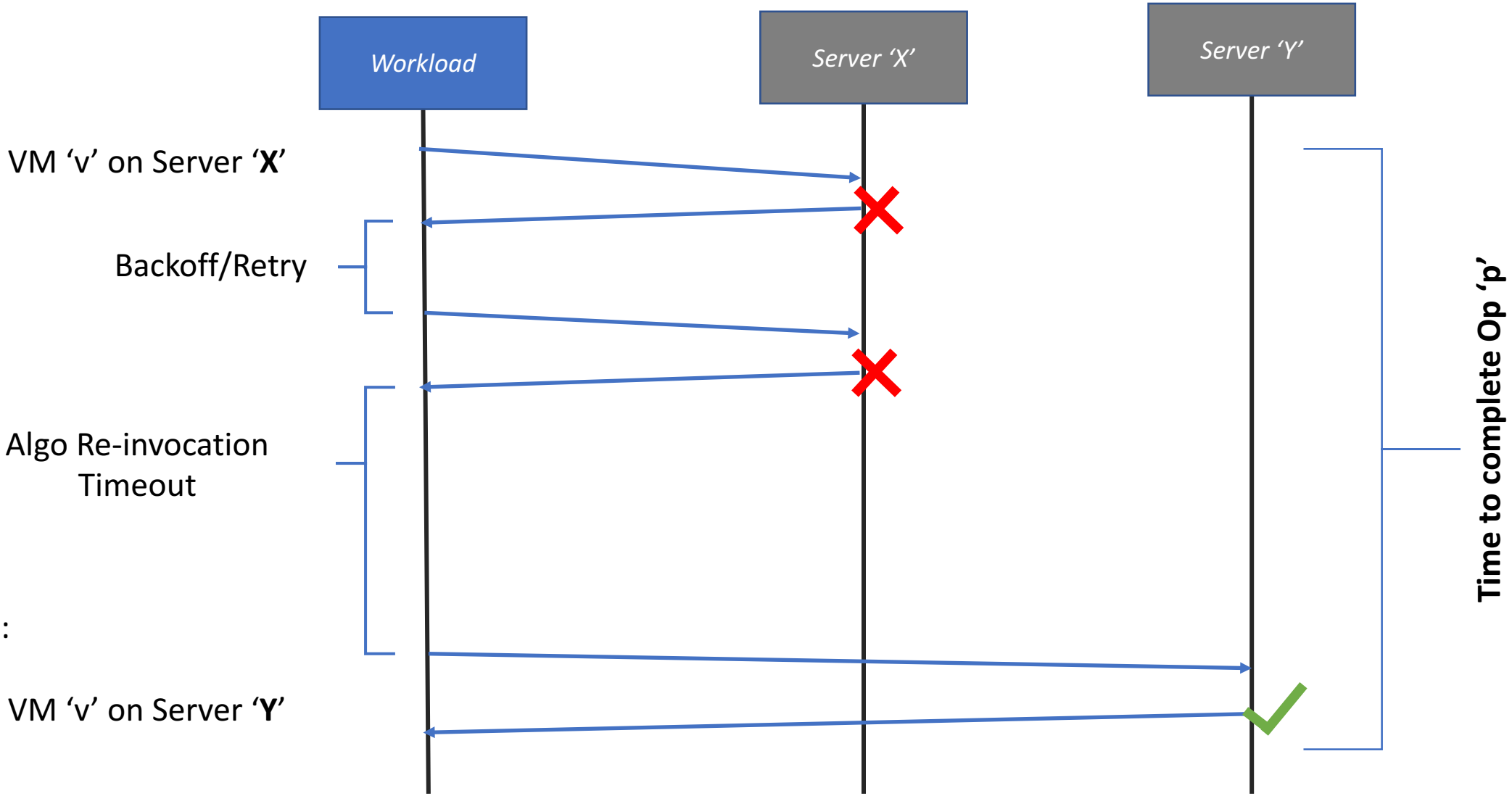
Invocation ‘**N**’:
1. Run Algo
2. Do Op ‘p’ on VM ‘v’ on Server ‘**X**’



Invocation ‘**N+1**’:
1. Run Algo
2. Do Op ‘p’ on VM ‘v’ on Server ‘**Y**’

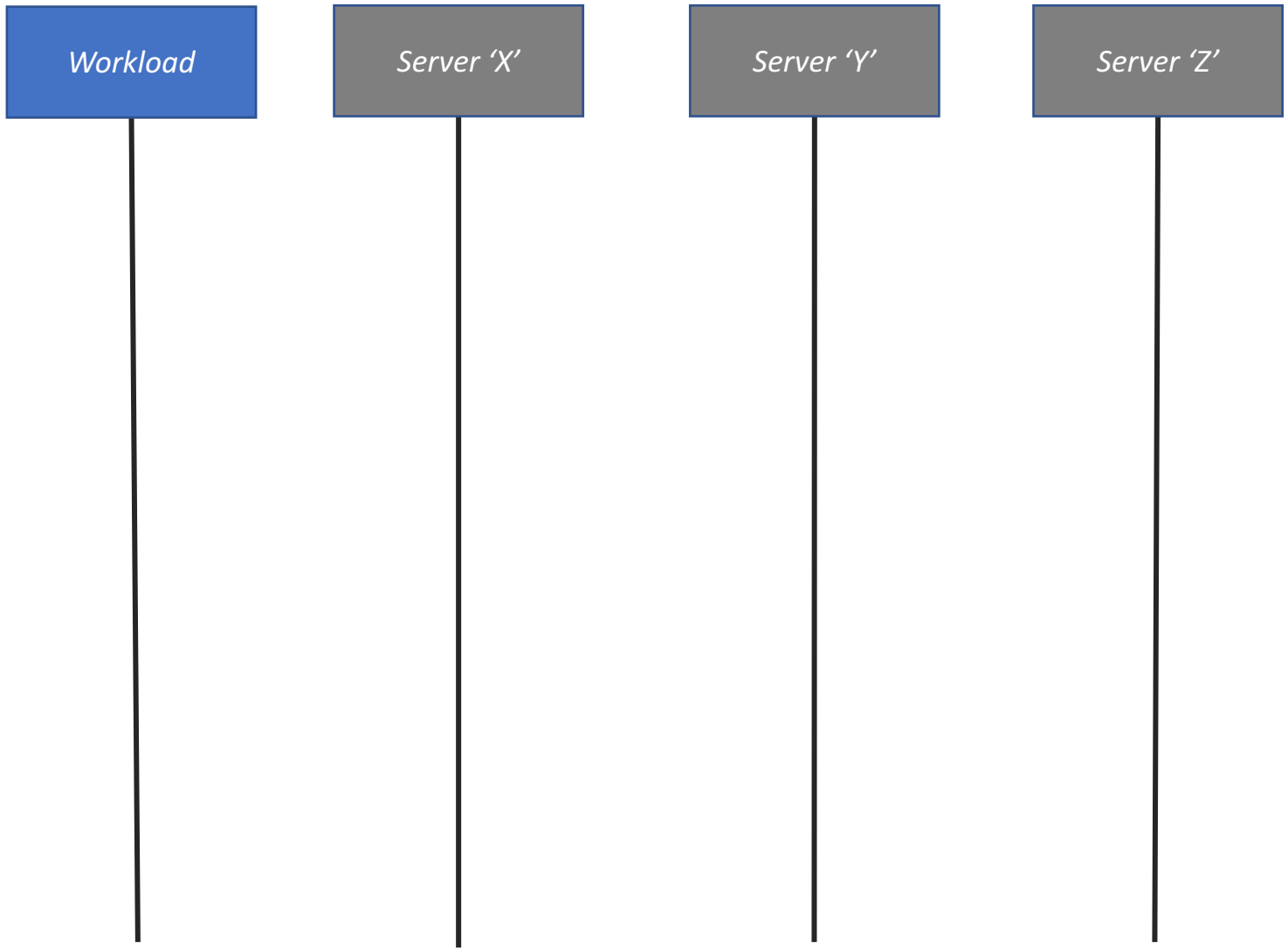
Naïve Retrying

Invocation '**N**':
1. Run Algo
2. Do Op 'p' on VM 'v' on Server '**X**'



Invocation '**N+1**':
1. Run Algo
2. Do Op 'p' on VM 'v' on Server '**Y**'

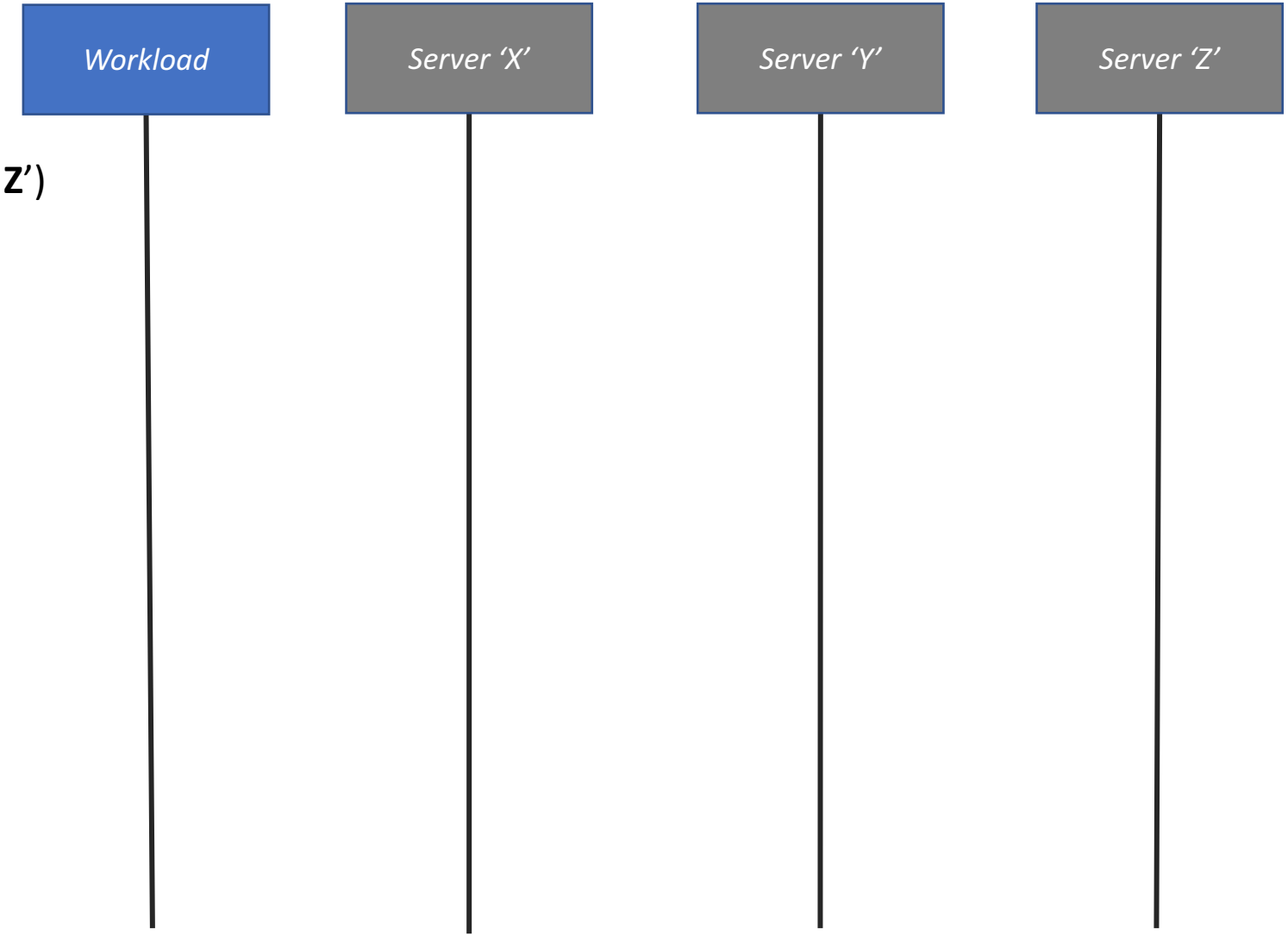
Speculative Operation Replication in Space



Speculative Operation Replication in Space

Invocation ‘**N**’:

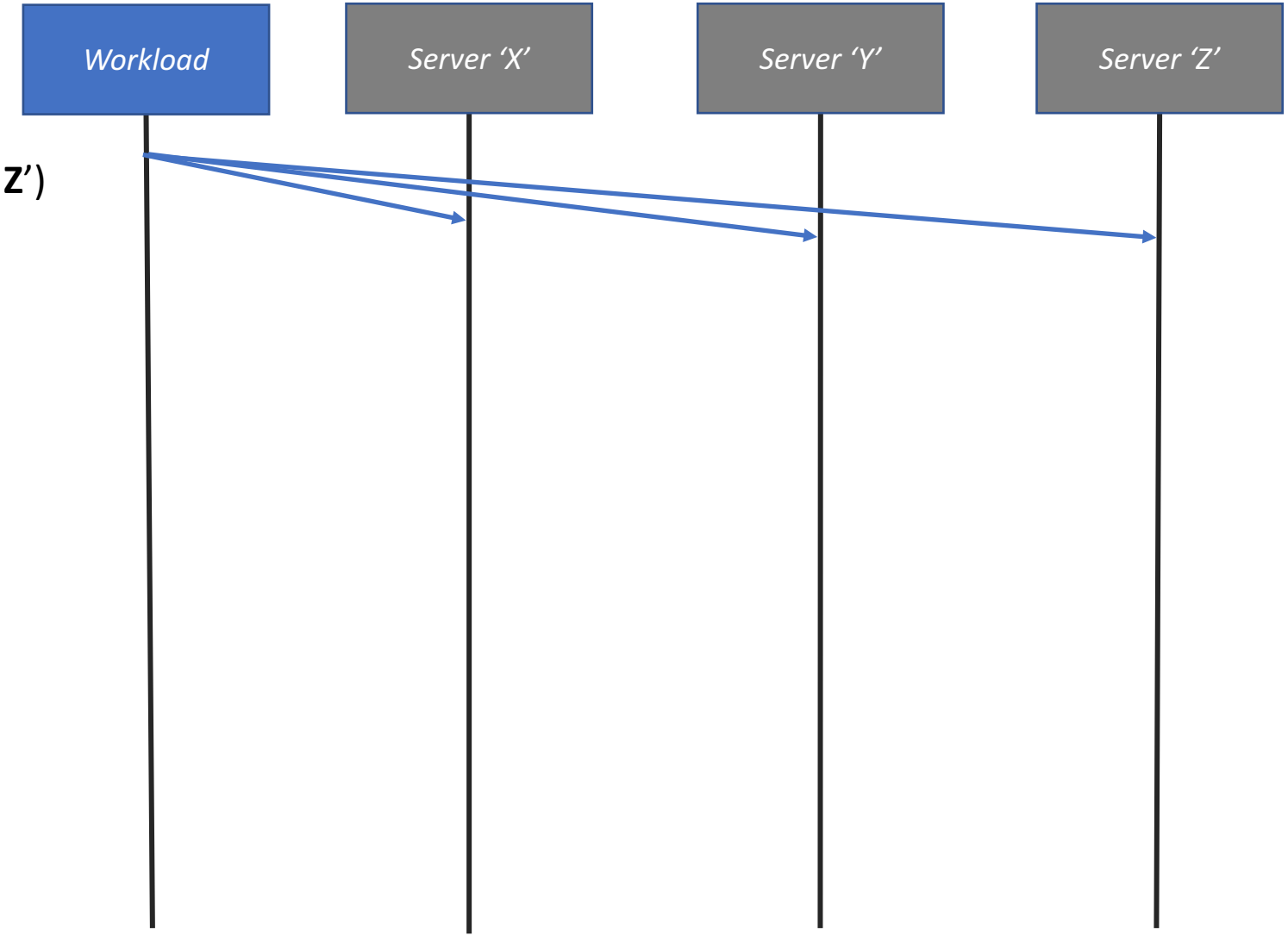
1. Run Algo
2. Do Op ‘p’ on VM ‘v’ on OneOf(‘**X, Y, Z**’)



Speculative Operation Replication in Space

Invocation ‘**N**’:

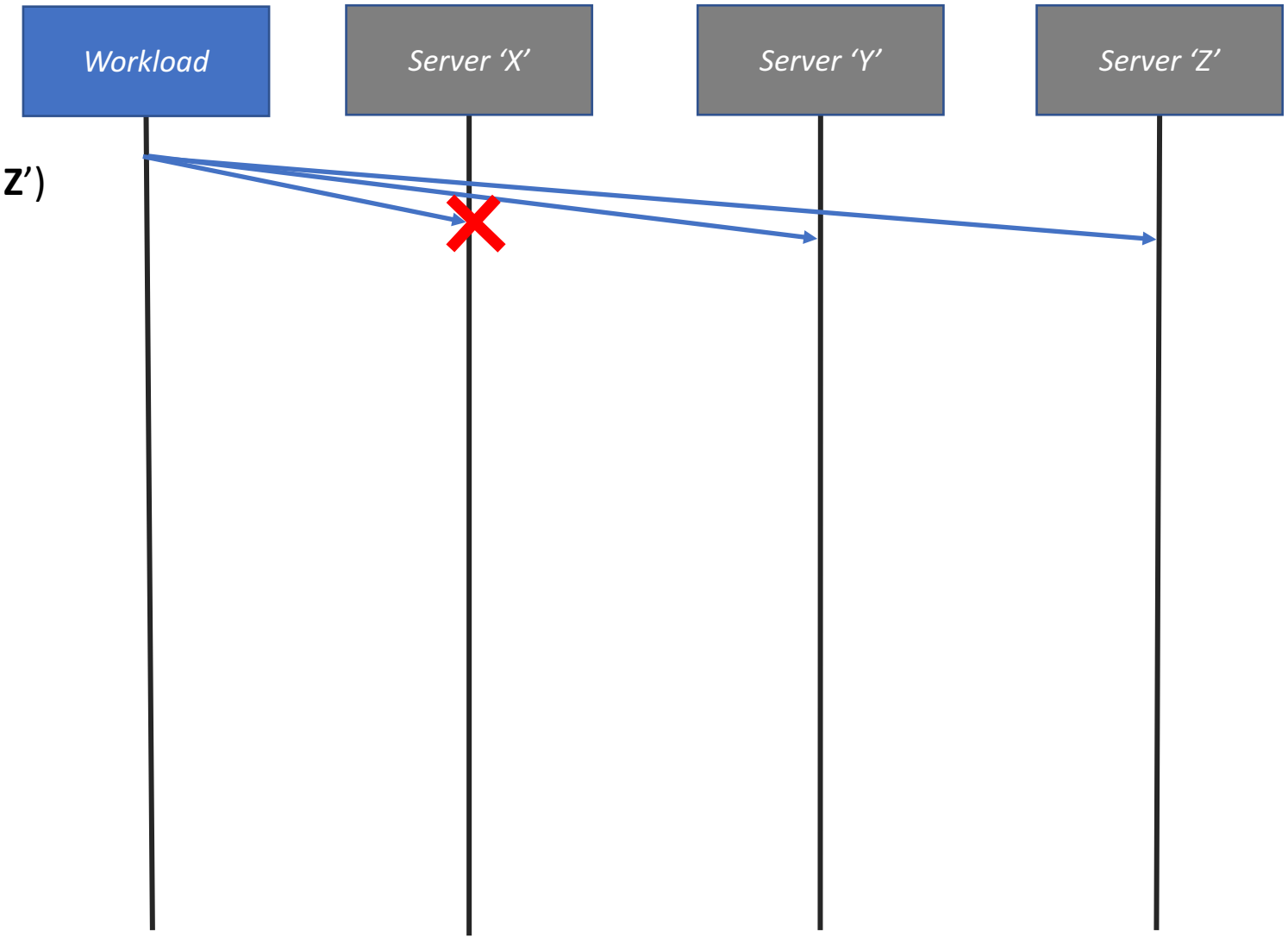
1. Run Algo
2. Do Op ‘p’ on VM ‘v’ on OneOf(‘**X, Y, Z**’)



Speculative Operation Replication in Space

Invocation ‘**N**’:

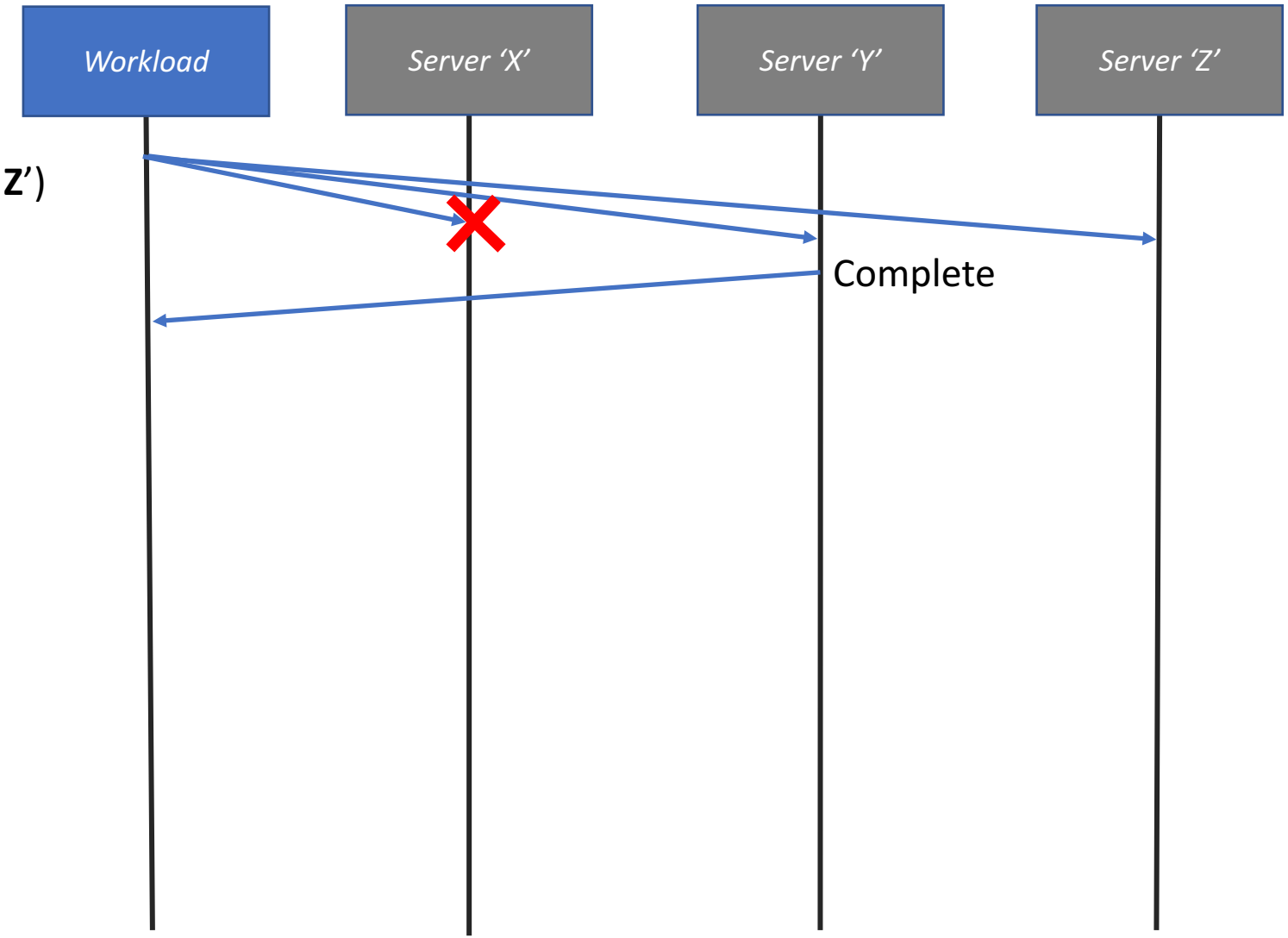
1. Run Algo
2. Do Op ‘p’ on VM ‘v’ on OneOf(‘**X, Y, Z**’)



Speculative Operation Replication in Space

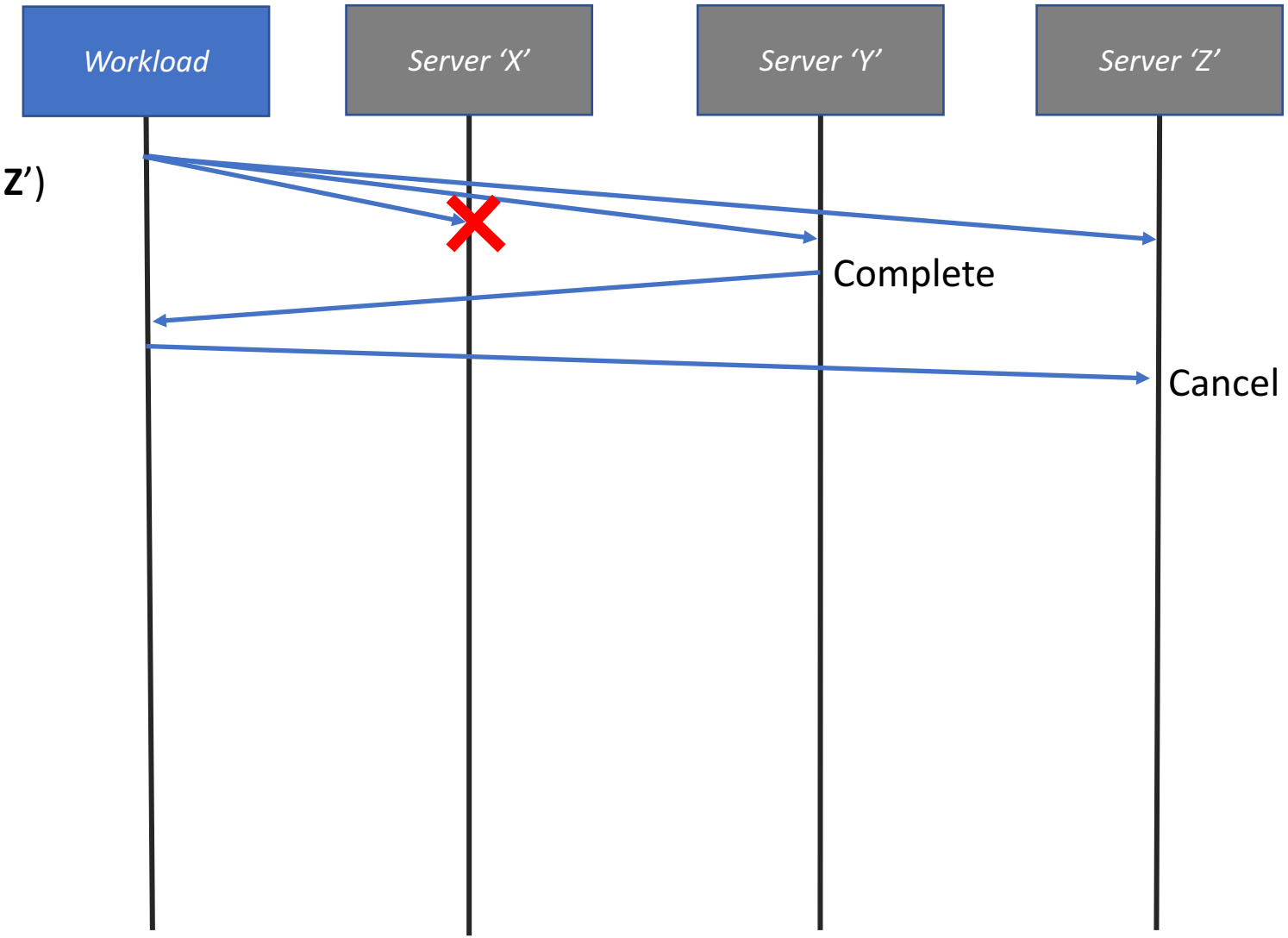
Invocation ‘**N**’:

1. Run Algo
2. Do Op ‘p’ on VM ‘v’ on OneOf(‘**X, Y, Z**’)



Speculative Operation Replication in Space

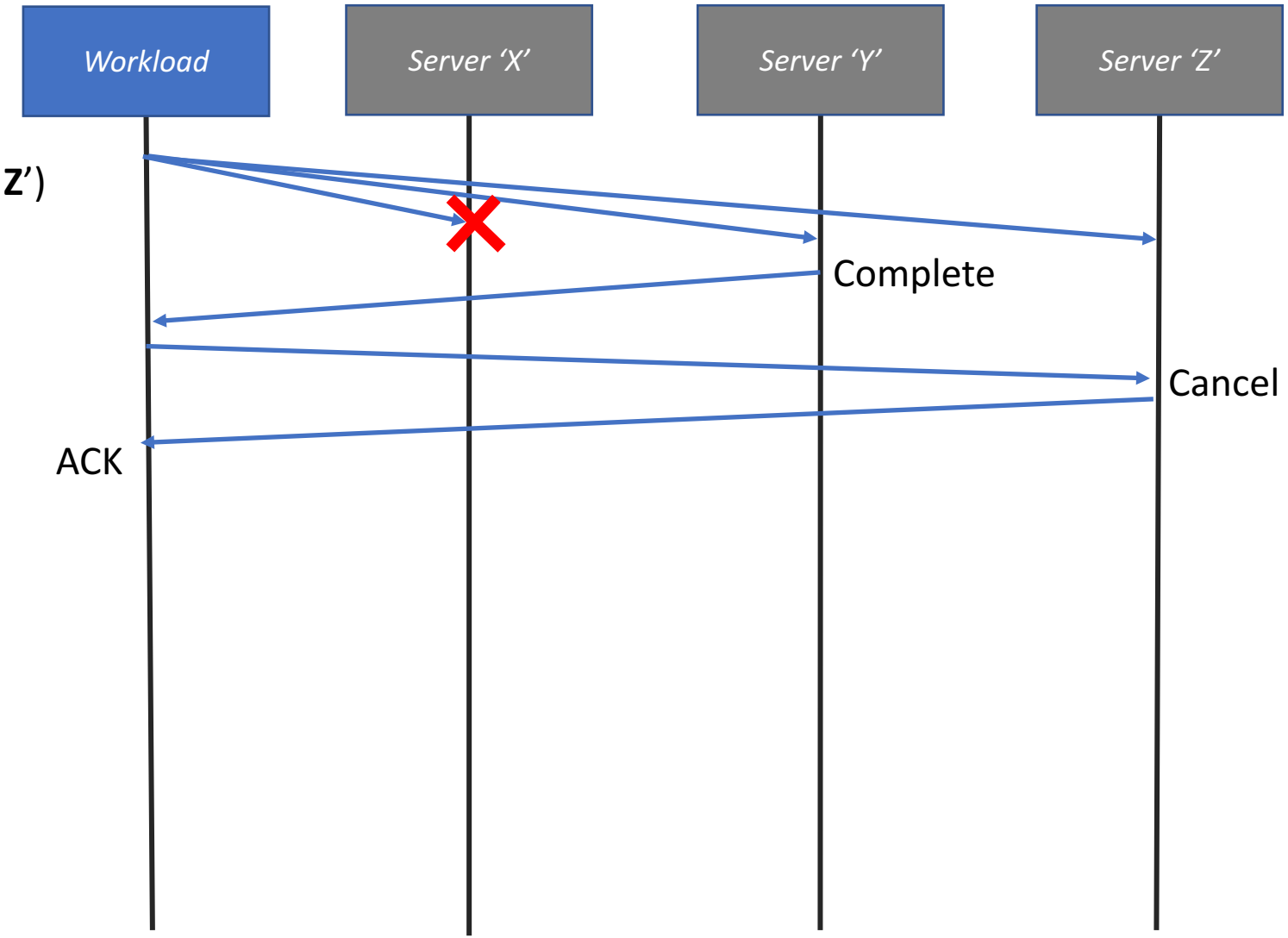
Invocation '**N**':
1. Run Algo
2. Do Op 'p' on VM 'v' on OneOf('**X, Y, Z**')



Speculative Operation Replication in Space

Invocation ‘**N**’:

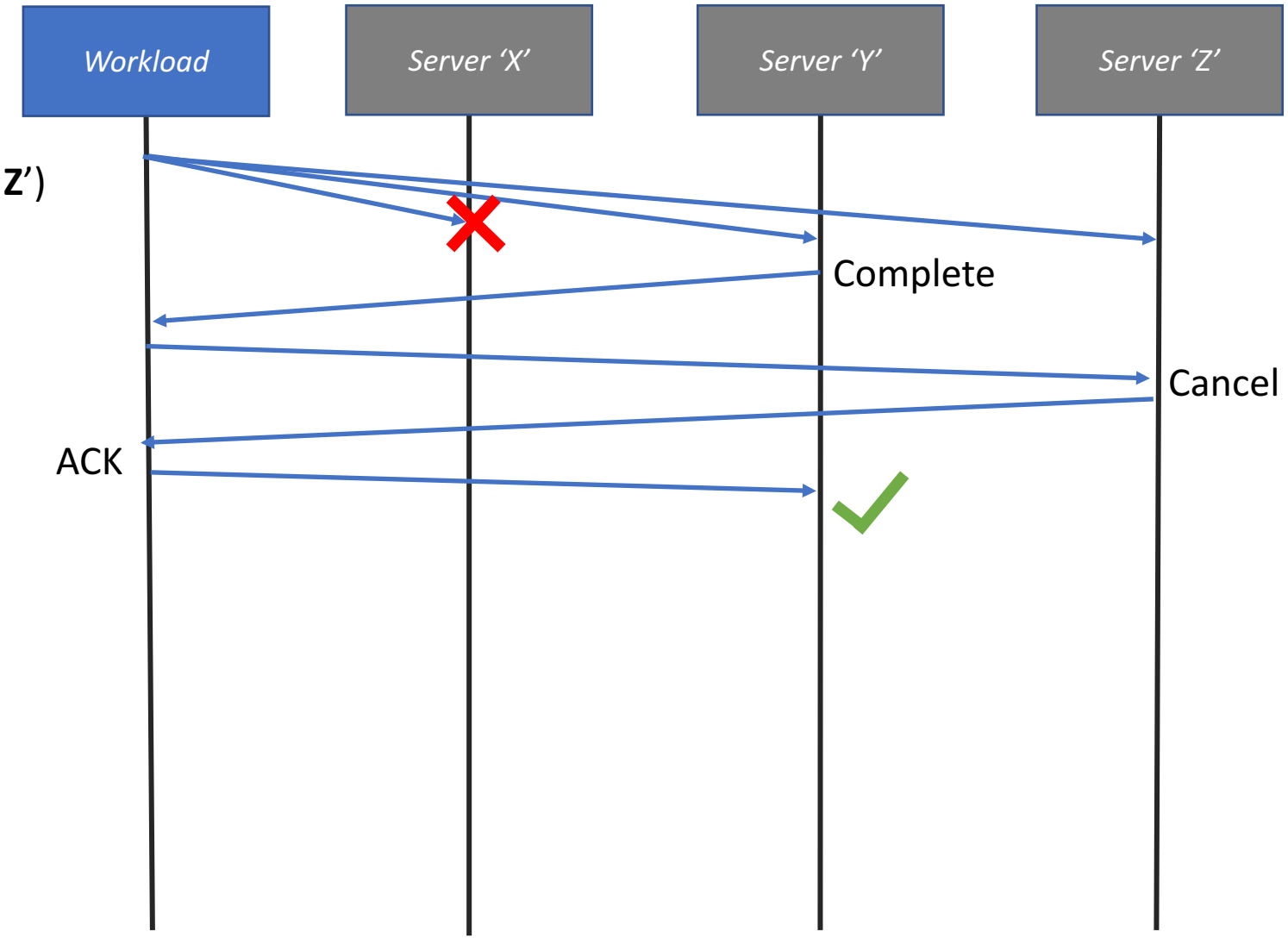
1. Run Algo
2. Do Op ‘p’ on VM ‘v’ on OneOf(‘**X, Y, Z**’)



Speculative Operation Replication in Space

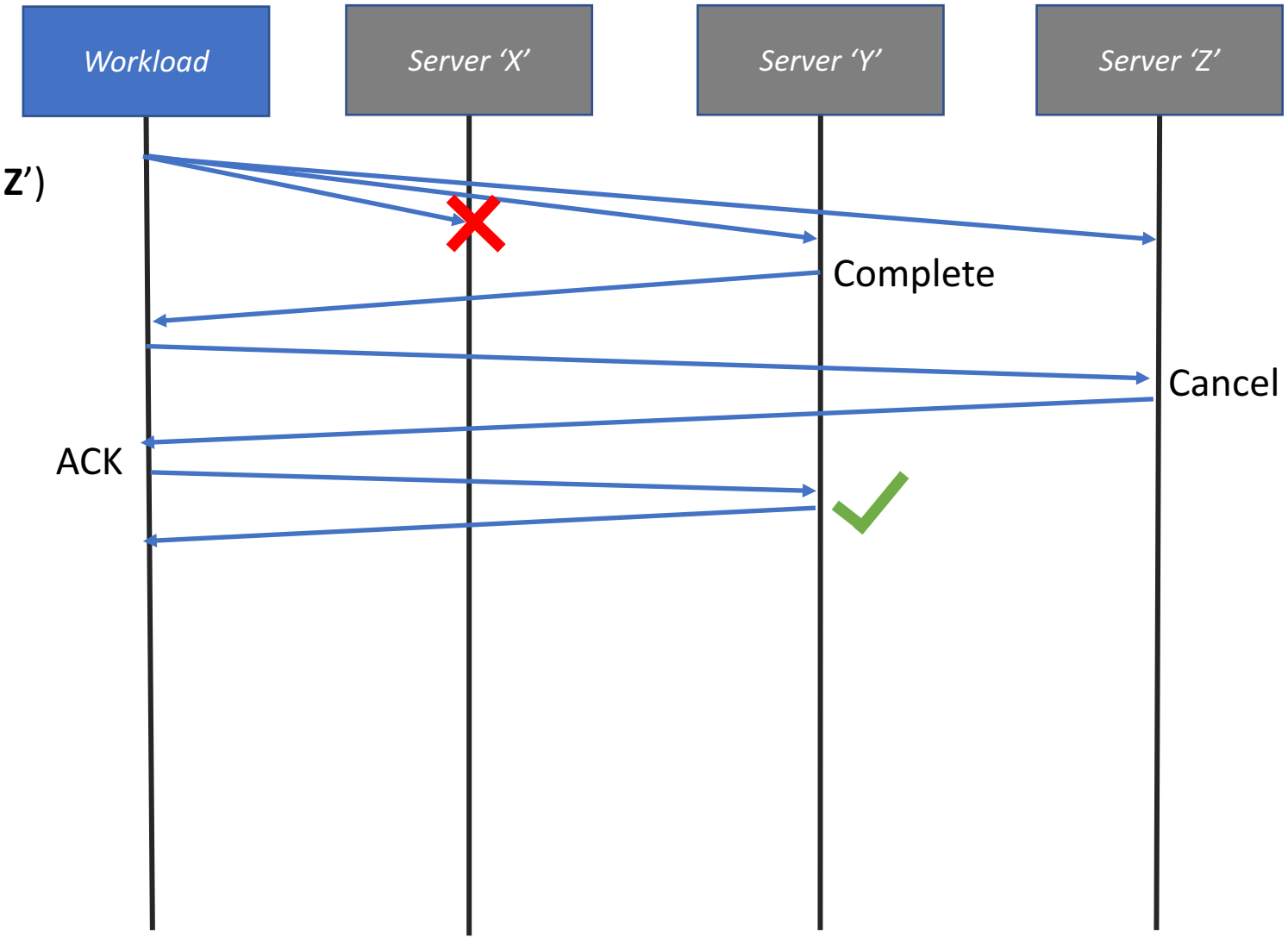
Invocation '**N**':

1. Run Algo
2. Do Op 'p' on VM 'v' on OneOf('**X, Y, Z**')



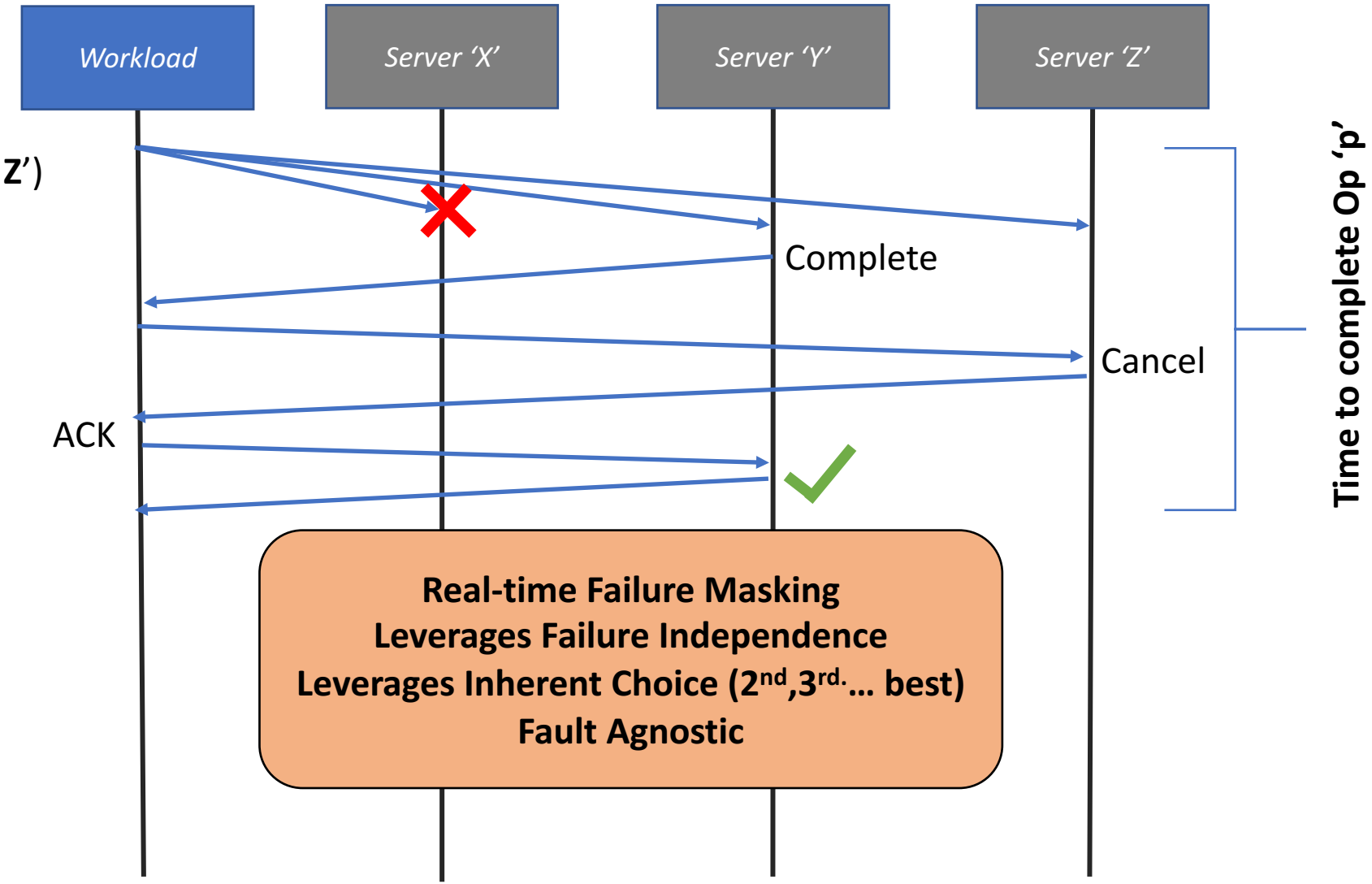
Speculative Operation Replication in Space

Invocation '**N**':
1. Run Algo
2. Do Op 'p' on VM 'v' on OneOf('**X, Y, Z**')



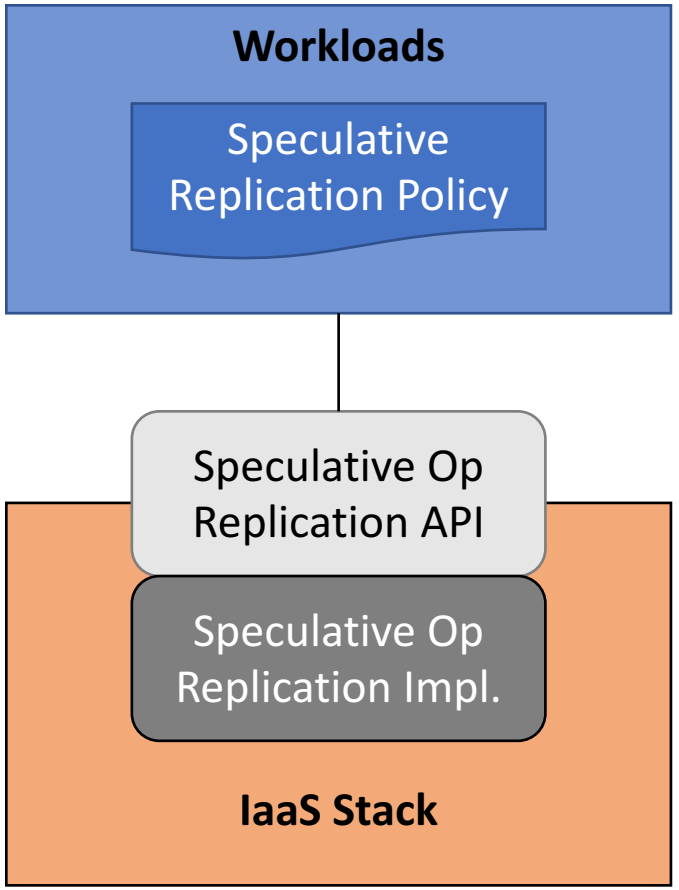
Speculative Operation Replication in Space

Invocation '**N**':
1. Run Algo
2. Do Op 'p' on VM 'v' on OneOf('**X, Y, Z**')

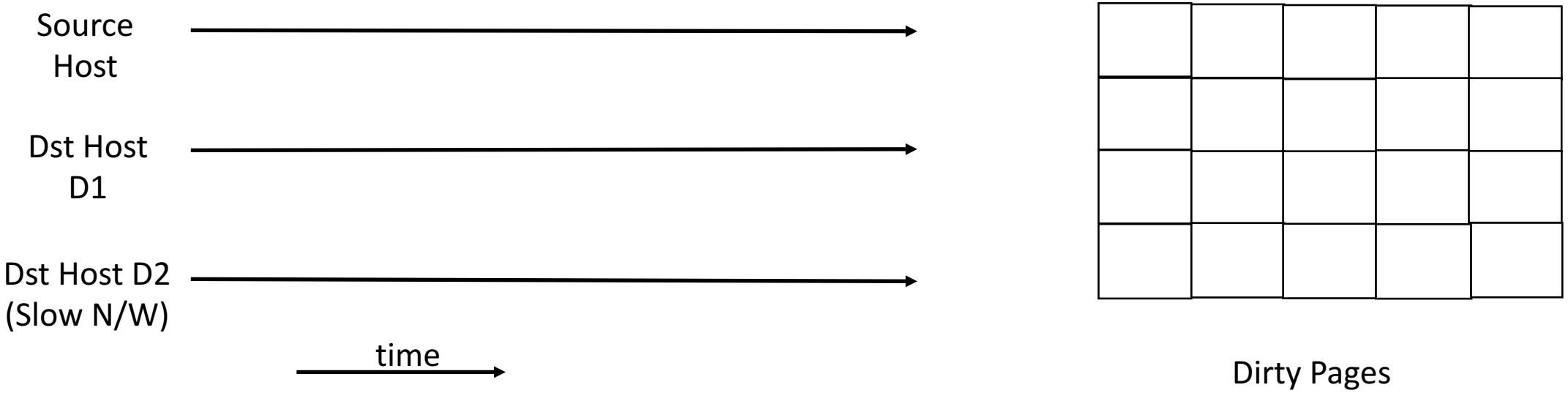


Solution Design

- **Speculatively Replicated Operations for IaaS Stacks**
  - Correctness: Only one replica wins
  - SpecOp(IN vm, IN targets[], IN cost)
- **Separate Policy from Implementation**
- Workloads control replication and its cost (policy):
  - Number of Operation Replicas
  - Cost of each Operation Replica
    - Timeout based Replica Abort
    - Resource consumption based Replica Abort
    - Progress Rate based Replica Abort

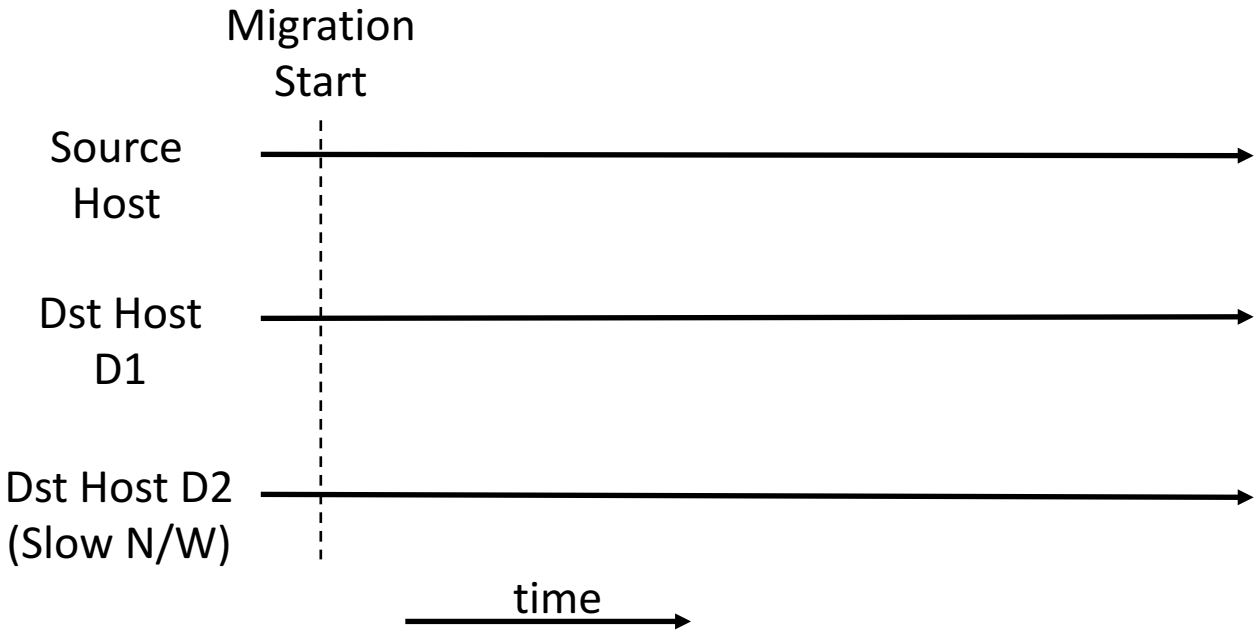


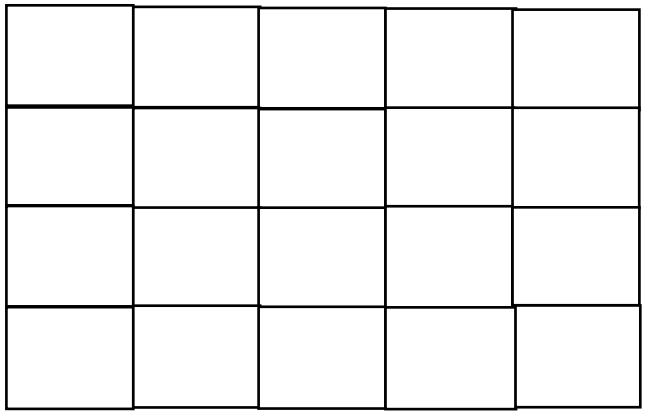
Example: Speculative VM Migration Repl. Impl.



- Pre-copy round durations vary => Dirty pages per target, per round vary
- One dirty bitmap per migration thread
- Reset shadow PTs at end of any pre-copy migration round

Example: Speculative VM Migration Repl. Impl.

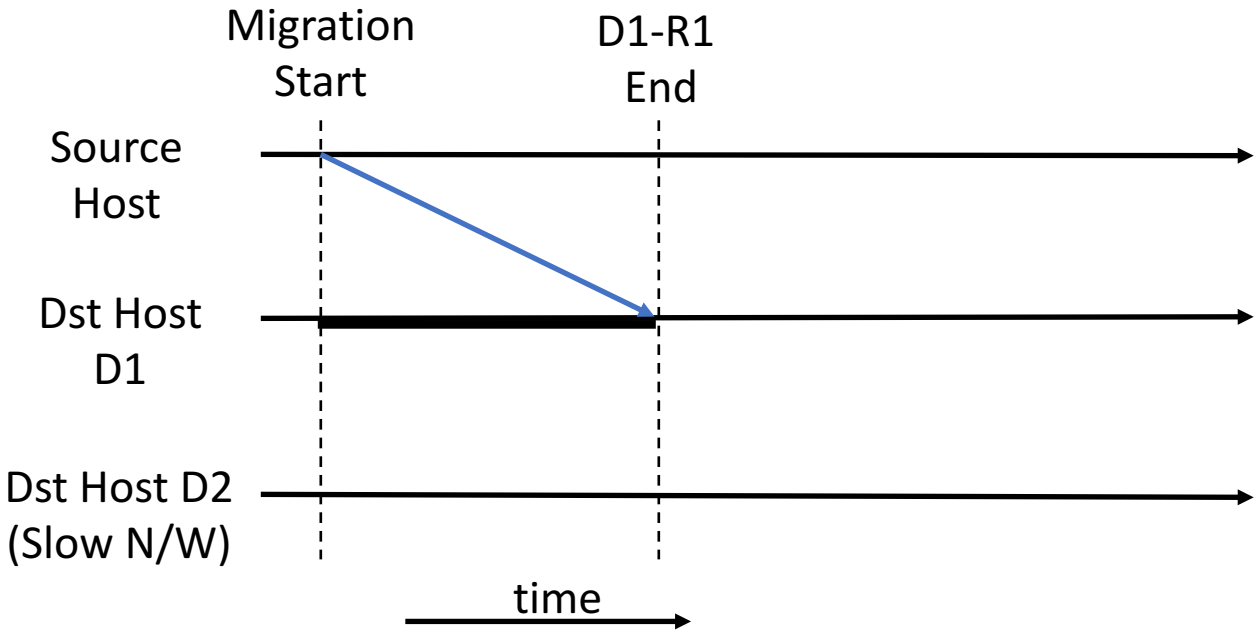


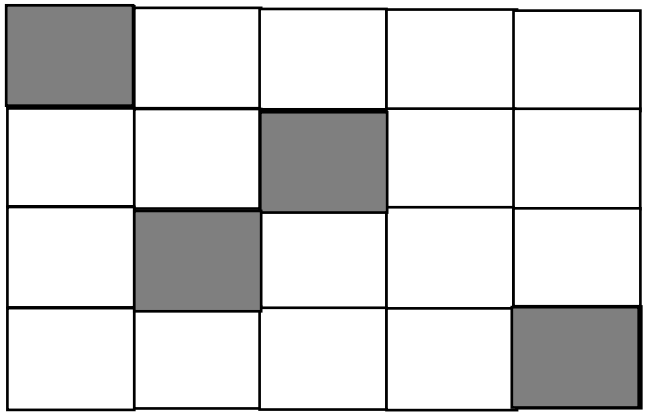


Dirty Pages

- Pre-copy round durations vary => Dirty pages per target, per round vary
- One dirty bitmap per migration thread
- Reset shadow PTs at end of any pre-copy migration round

Example: Speculative VM Migration Repl. Impl.

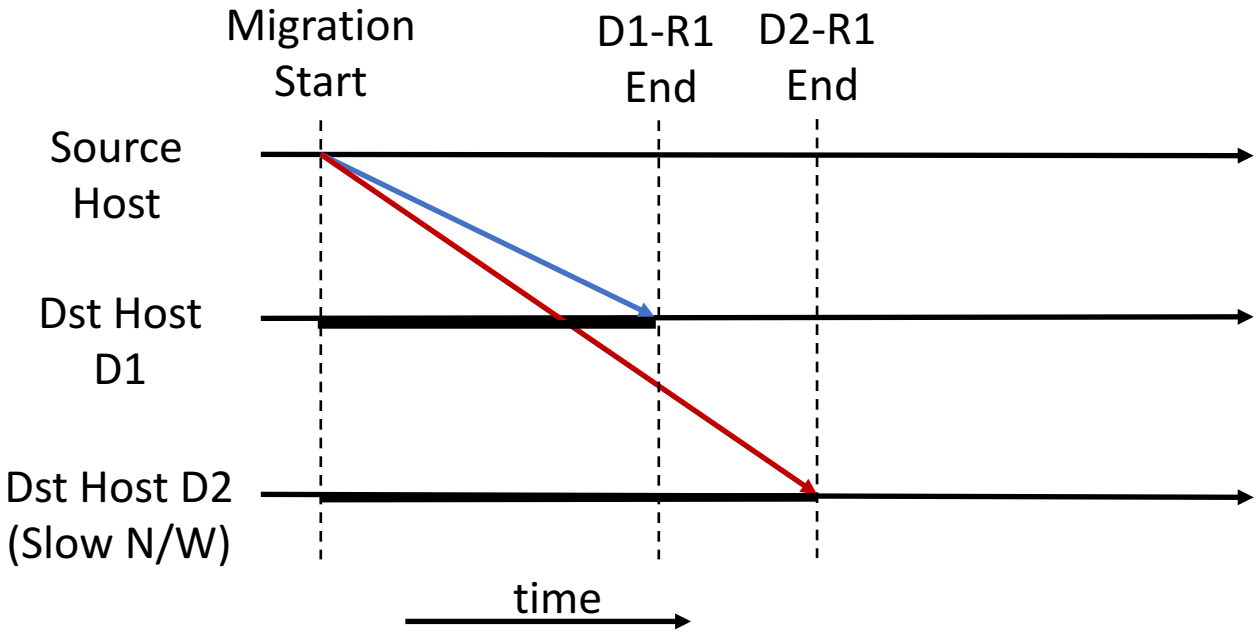


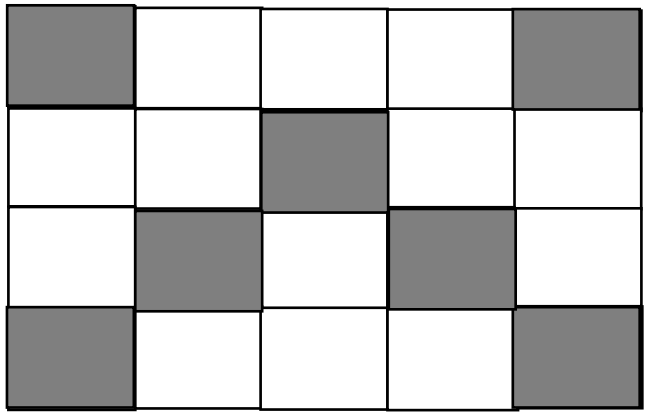


Dirty Pages

- Pre-copy round durations vary => Dirty pages per target, per round vary
- One dirty bitmap per migration thread
- Reset shadow PTs at end of any pre-copy migration round

Example: Speculative VM Migration Repl. Impl.

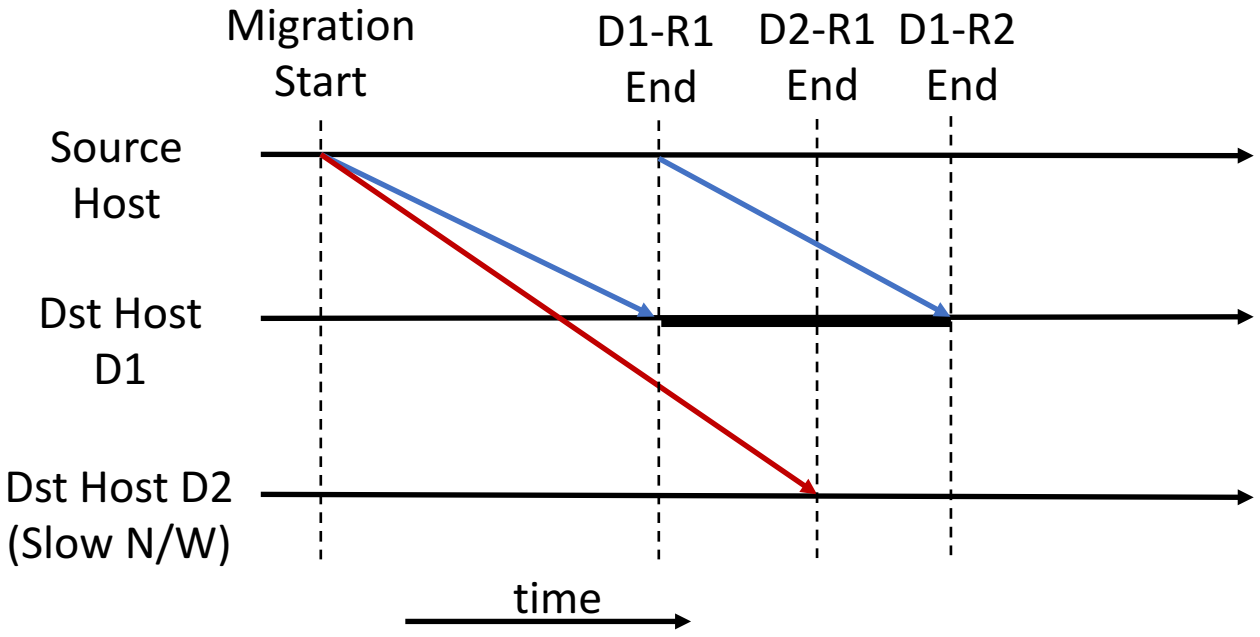


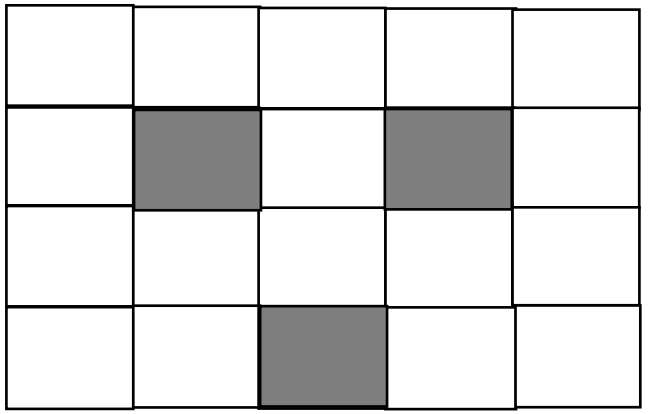


Dirty Pages

- Pre-copy round durations vary => Dirty pages per target, per round vary
- One dirty bitmap per migration thread
- Reset shadow PTs at end of any pre-copy migration round

Example: Speculative VM Migration Repl. Impl.

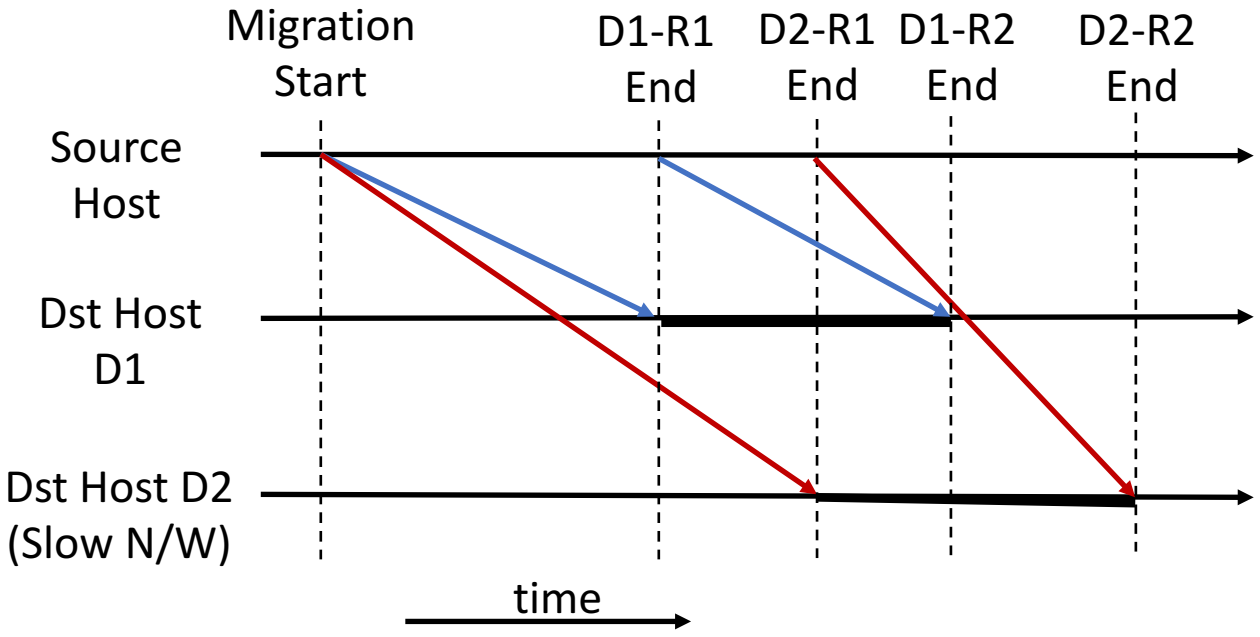


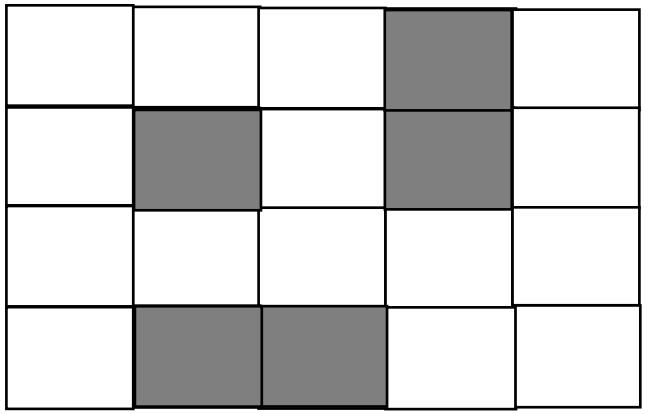


Dirty Pages

- Pre-copy round durations vary => Dirty pages per target, per round vary
- One dirty bitmap per migration thread
- Reset shadow PTs at end of any pre-copy migration round

Example: Speculative VM Migration Repl. Impl.

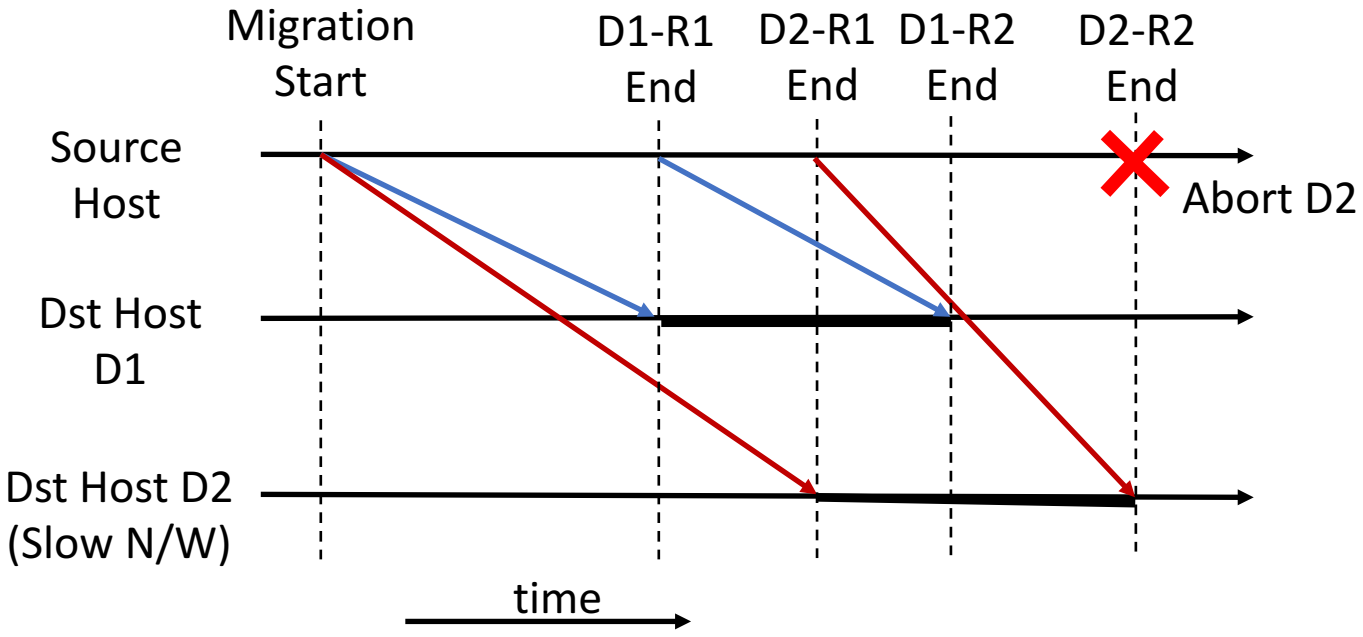


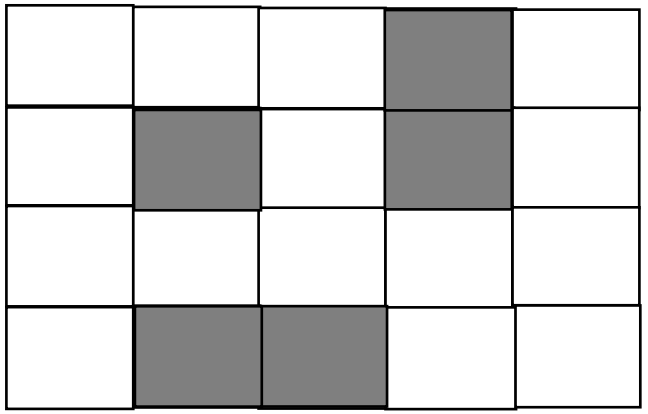


Dirty Pages

- Pre-copy round durations vary => Dirty pages per target, per round vary
- One dirty bitmap per migration thread
- Reset shadow PTs at end of any pre-copy migration round

Example: Speculative VM Migration Repl. Impl.

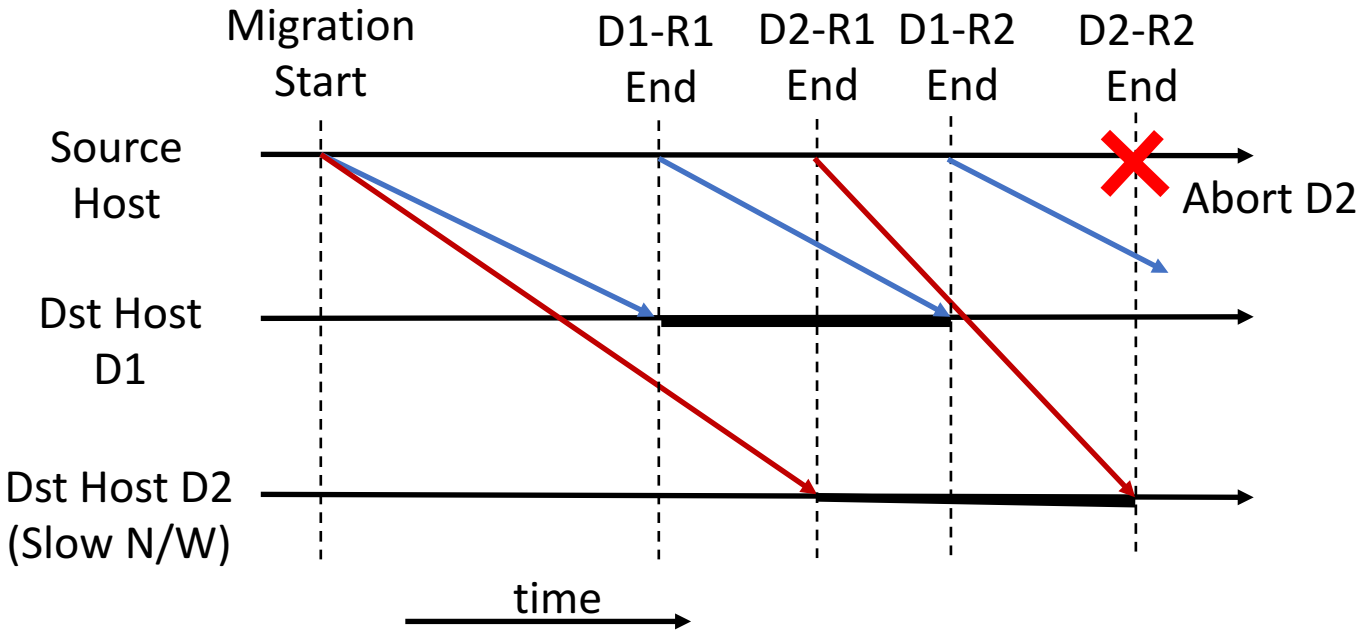


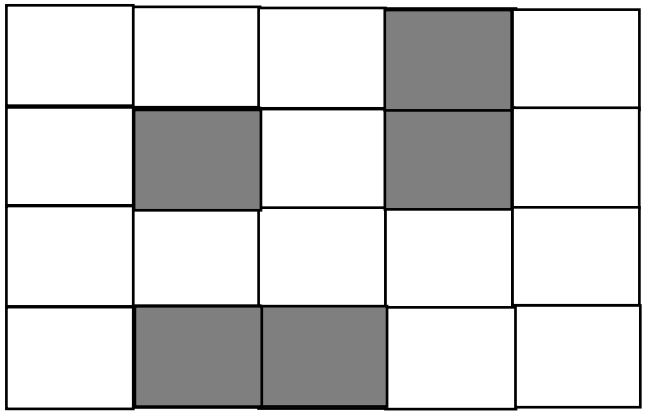


Dirty Pages

- Pre-copy round durations vary => Dirty pages per target, per round vary
- One dirty bitmap per migration thread
- Reset shadow PTs at end of any pre-copy migration round

Example: Speculative VM Migration Repl. Impl.





Dirty Pages

- Pre-copy round durations vary => Dirty pages per target, per round vary
- One dirty bitmap per migration thread
- Reset shadow PTs at end of any pre-copy migration round

Implementation and Evaluation

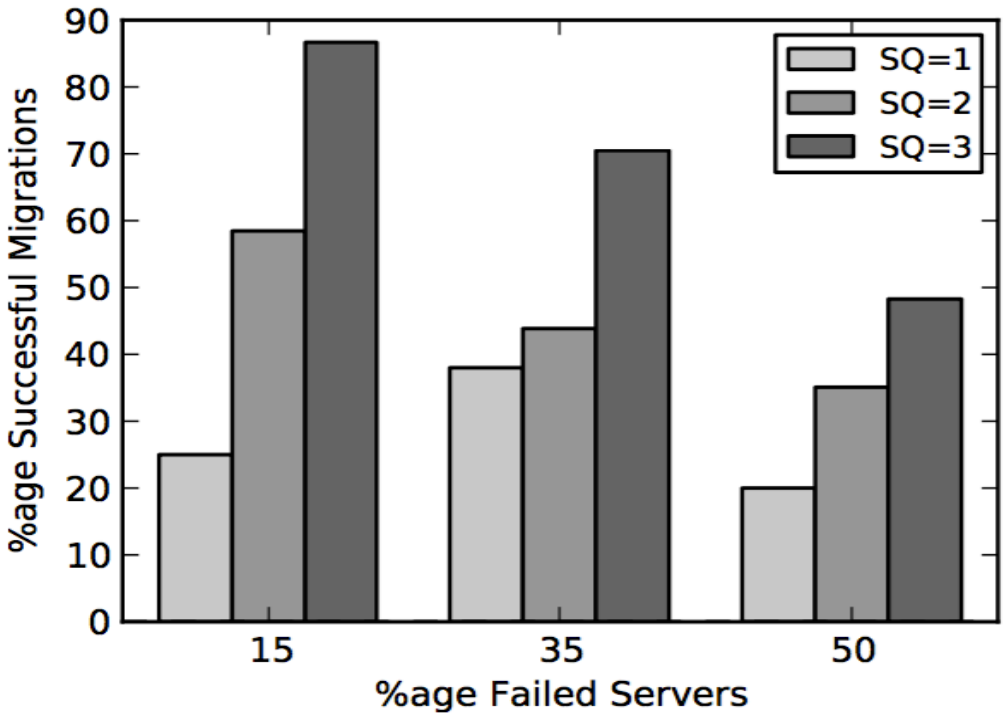
- Speculatively Replicated VM Migration and PowerOn Placement
- Implemented in Xen hypervisor and API Exposed via OpenStack Nova
- Parametric Failure Simulation
  - Maintains configurable %age of servers in “failed” state
  - Failure event durations vary uniformly randomly
- Results on 12-server Physical Testbed
- 1024-server Custom Datacenter Discrete Event Simulator Results

Resource Capacity Multiplexing Workload

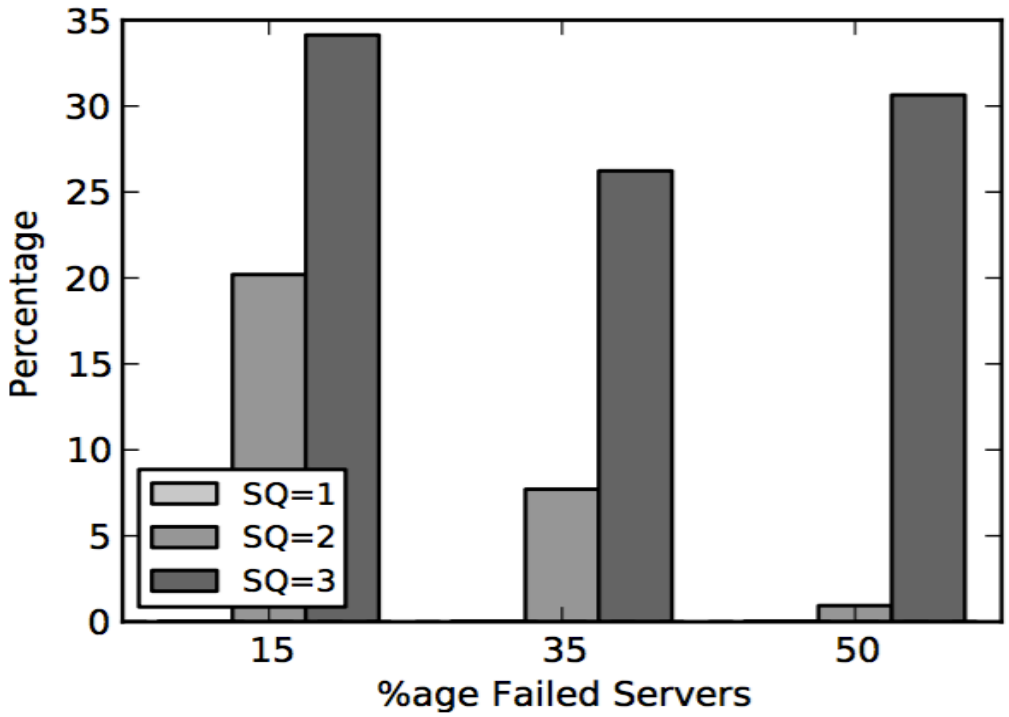
- Periodically Migrate VMs from overloaded to underloaded servers
- Traditional Algo: Picks one underloaded destination for each VM
- Modified: Pick more than one underloaded server for each VM
- (VM + Target Servers) => Composite Speculatively Rep. Migration Op

- <u>Goal</u>: Maximize load-shedding from overloaded hosts asap

Resource Capacity Multiplexing Results



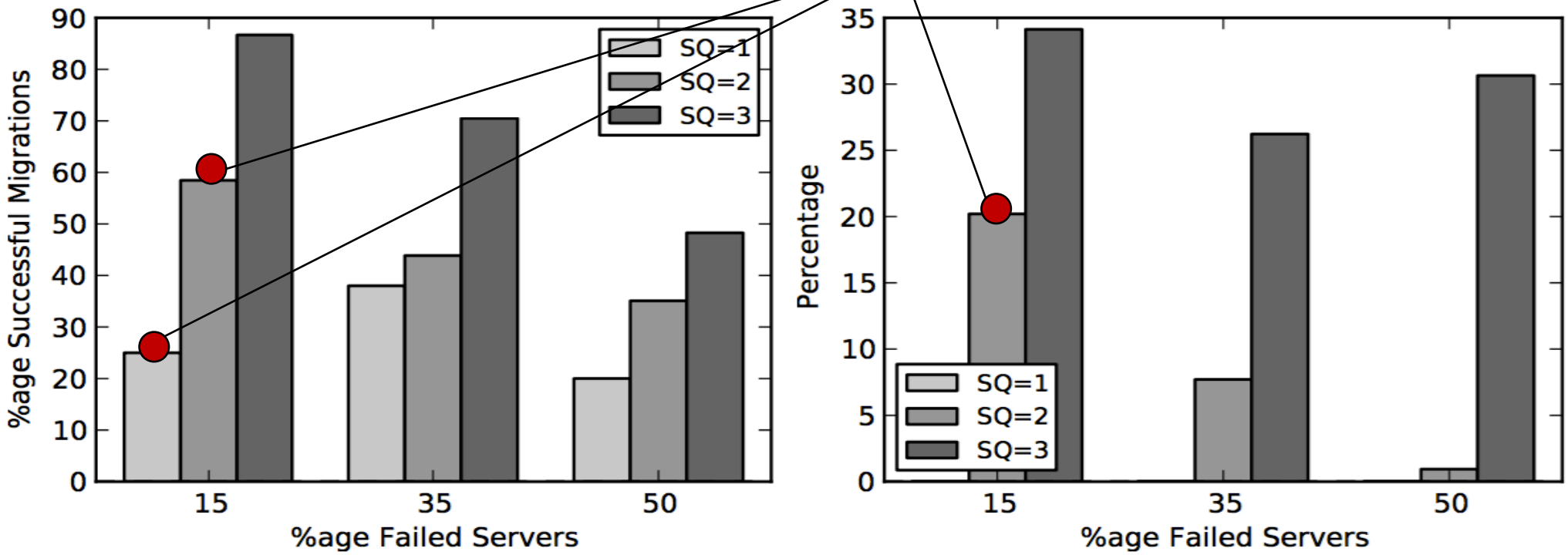
(a) Fraction of successful logical migrations.



(b) Normalized excess network usage per successful VM migration (replicated or otherwise).

Resource Capacity Multiplexing Results

> 2x gains, 20% overhead!



(a) Fraction of successful logical migrations.

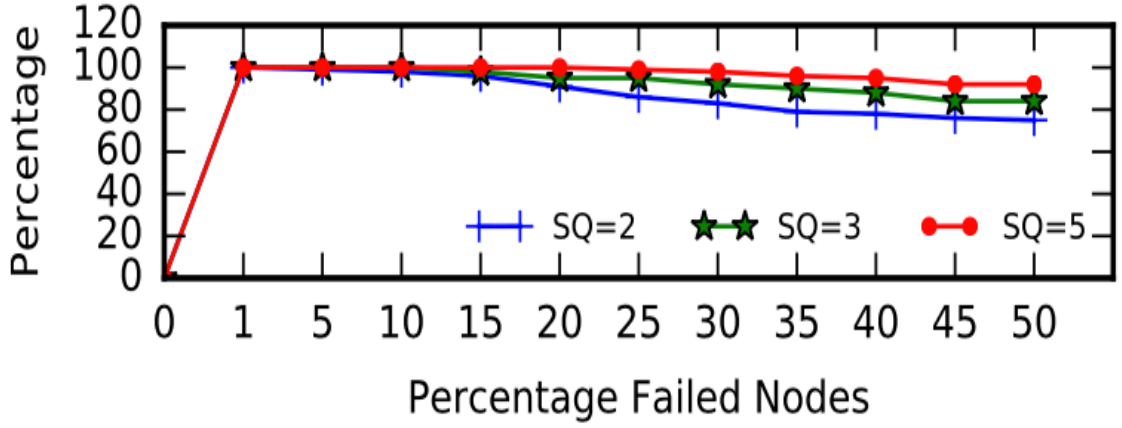
(b) Normalized excess network usage per successful VM migration (replicated or otherwise).

Continuous Integration Testing Workload

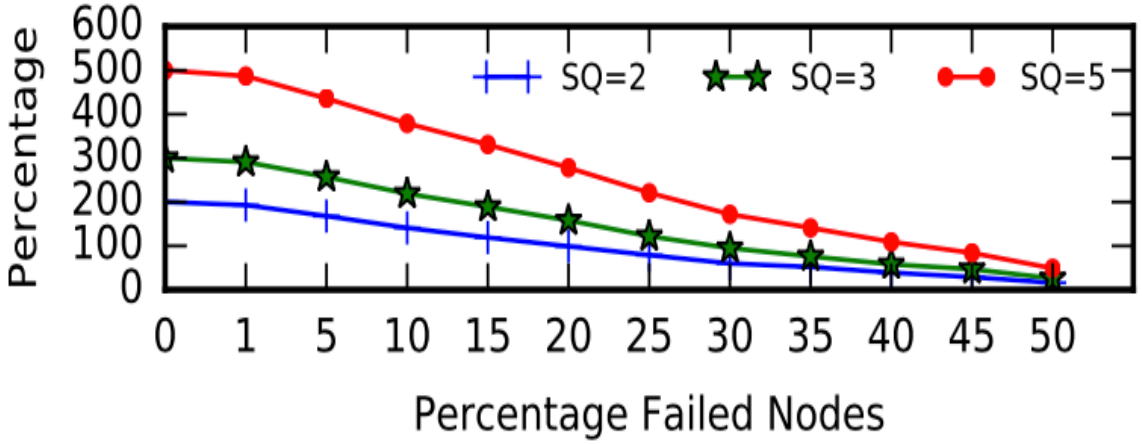
- PowerOn VMs in response to code checkin, build and run tests on it
- When a PowerOn fails, the VM goes back in backlog and tried again
- Modified: Pick multiple eligible servers per VM at random

- Goal: Minimize average delay in launching a testing VM

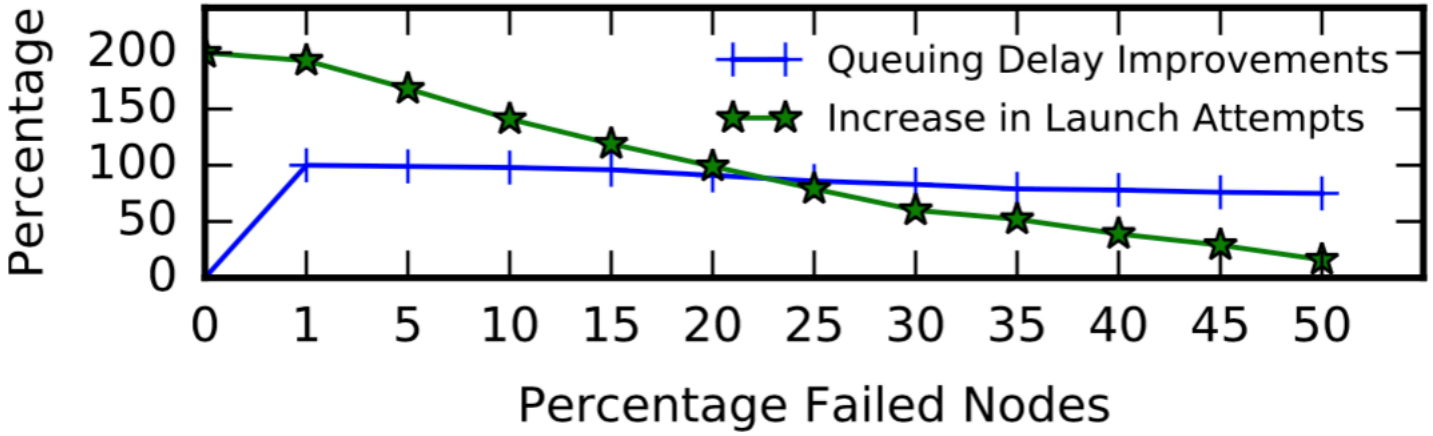
Continuous Integration Testing Results



(a) Avg. Launch Delay Improvement Per VM

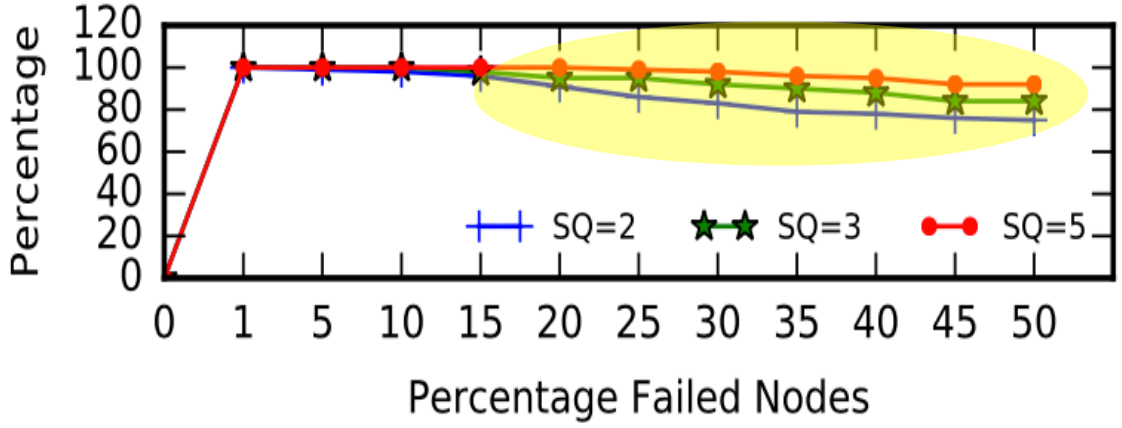


(b) Avg. Relative Launch Attempts Per VM

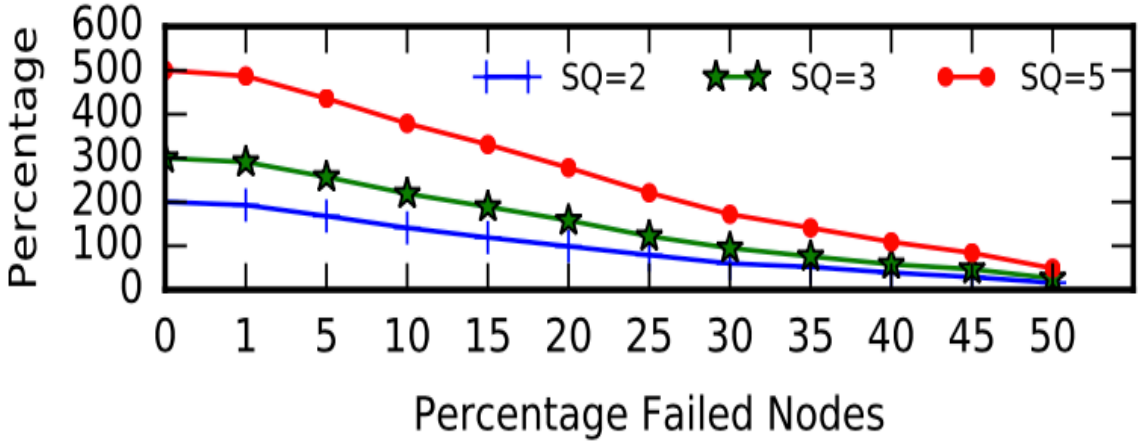


(c) Cost vs. Benefit Analysis for SQ=2

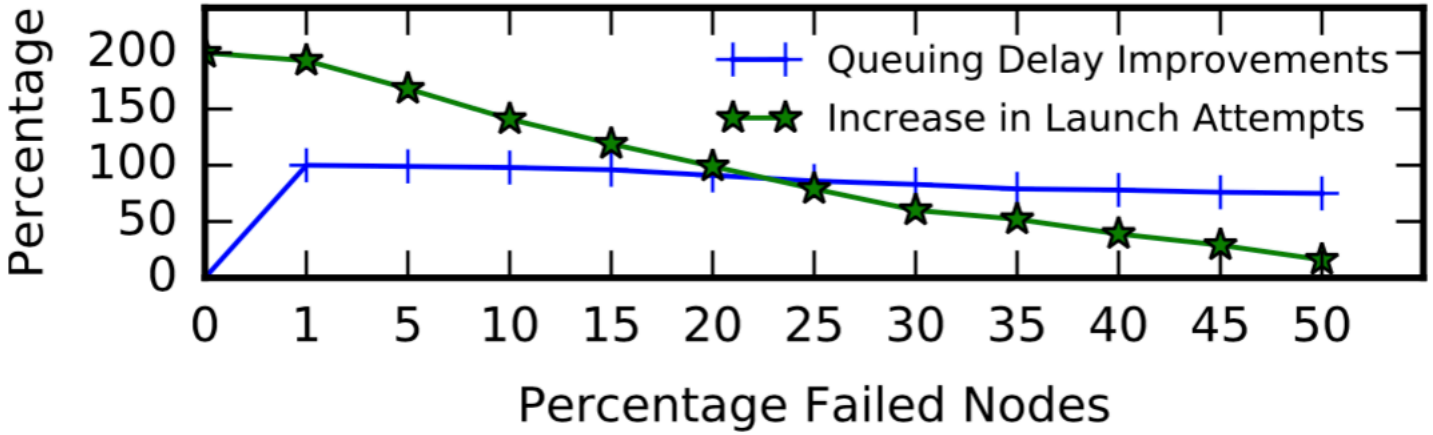
Continuous Integration Testing Results



(a) Avg. Launch Delay Improvement Per VM



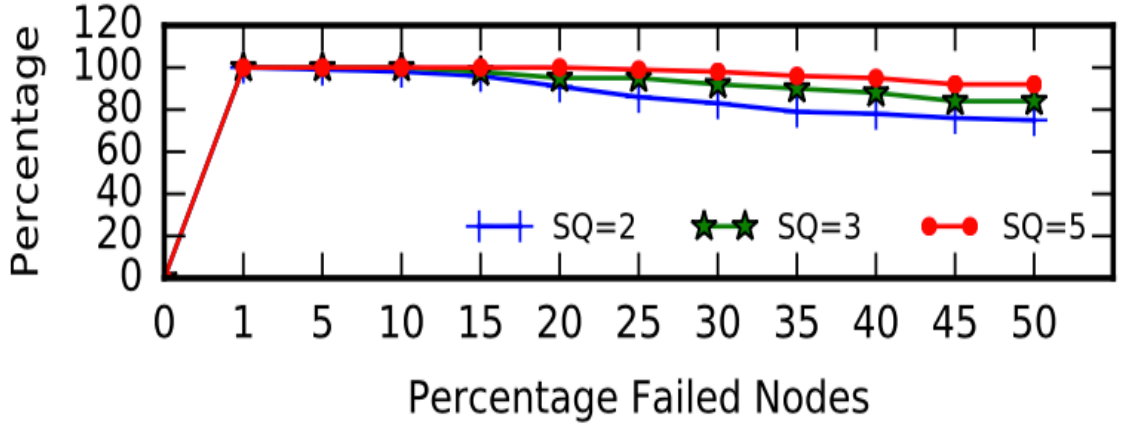
(b) Avg. Relative Launch Attempts Per VM



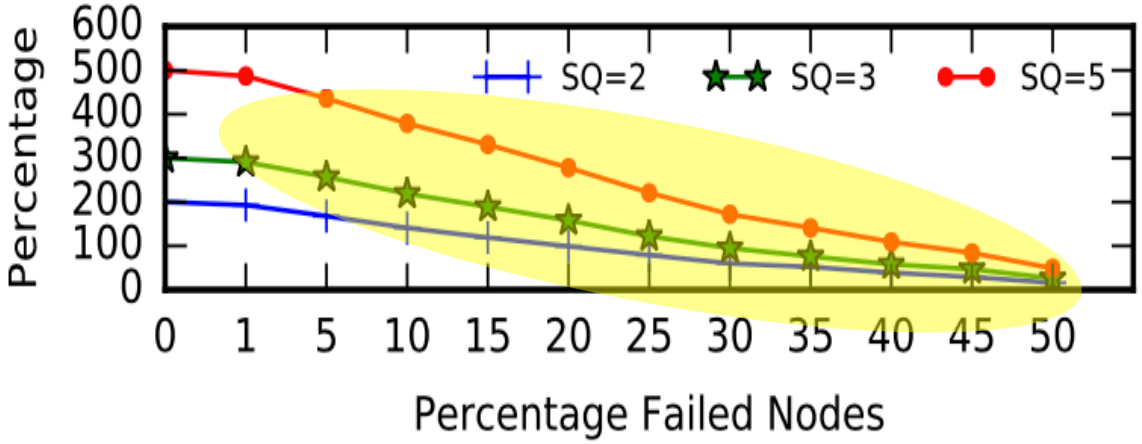
(c) Cost vs. Benefit Analysis for SQ=2

Minimal replication still Leads to large benefits!

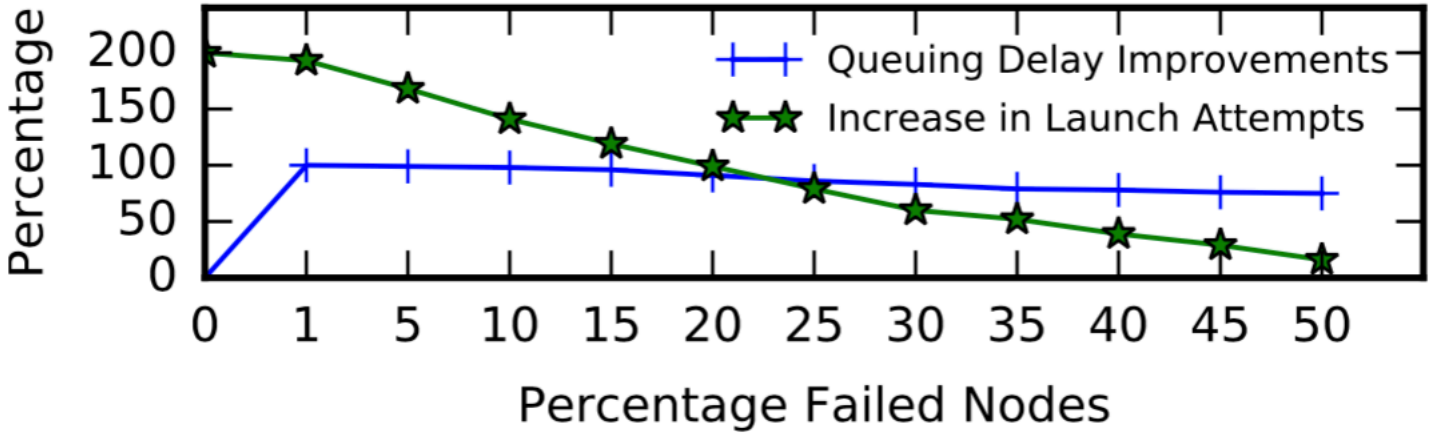
Continuous Integration Testing Results



(a) Avg. Launch Delay Improvement Per VM



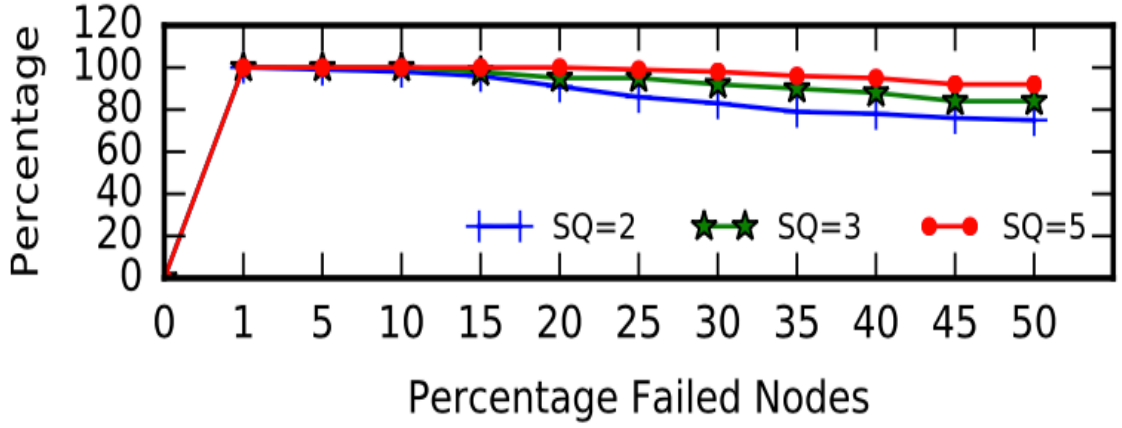
(b) Avg. Relative Launch Attempts Per VM



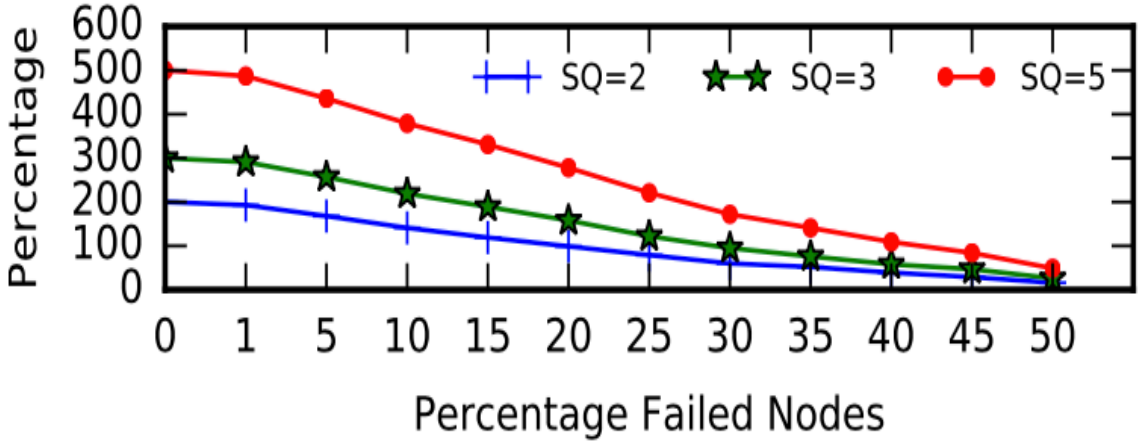
(c) Cost vs. Benefit Analysis for SQ=2

Replication cost steadily approaches retry cost as failures increase!

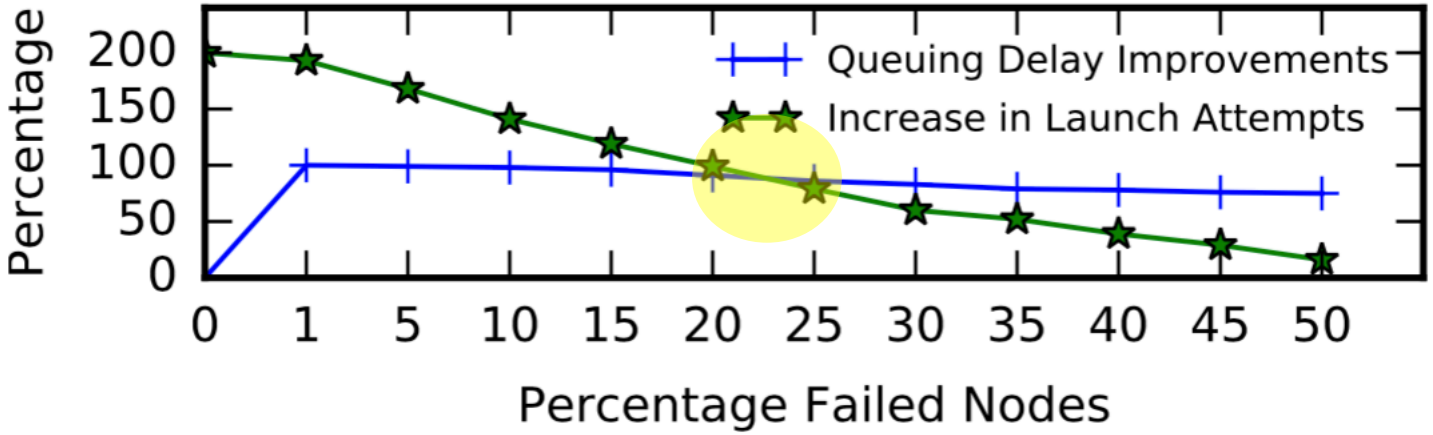
Continuous Integration Testing Results



(a) Avg. Launch Delay Improvement Per VM



(b) Avg. Relative Launch Attempts Per VM



(c) Cost vs. Benefit Analysis for SQ=2

Workloads can dynamically control replication based on observed failure-rates to lower cost & increase benefit

Conclusion

- We propose extensions to VM operations API exposed by IaaS Stacks
- IaaS stack implements speculatively replicated version of VM ops
- Workloads control cost vs. benefit tradeoff using specific knowledge
- System is fault-agnostic, provides fault-scalable performance
- Improves outcomes for typical datacenter workloads

Backup

Overall Flow

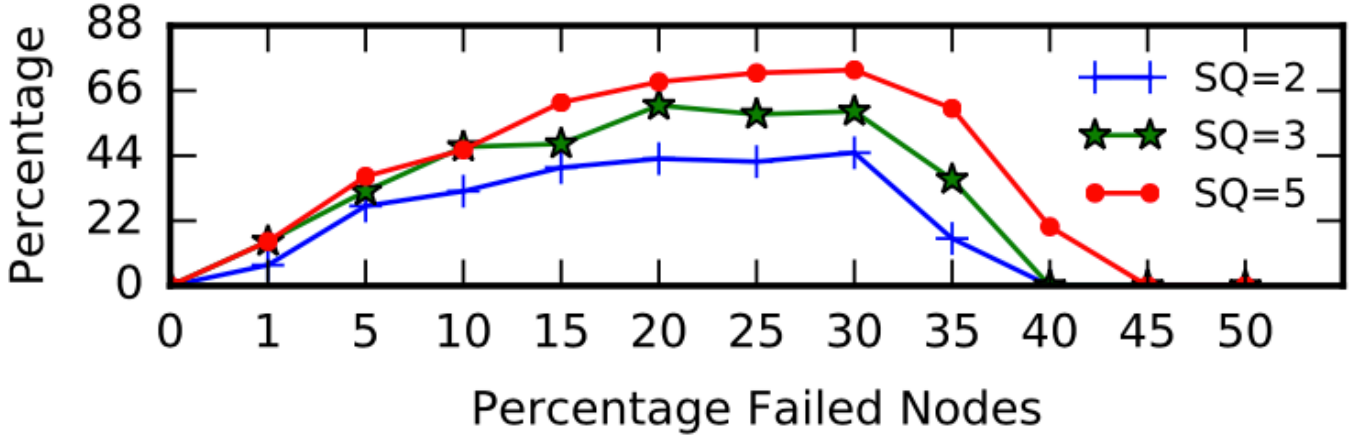
- Background on datacenter automation services
- Impairment Problem + Motivation
- Approach + Philosophy
- Implementation of Speculatively Replicated Migration
- Redesigned Auto Services: LB, RollUpgrade, CI/CD
- Results

Rolling Cluster Server Software Upgrades

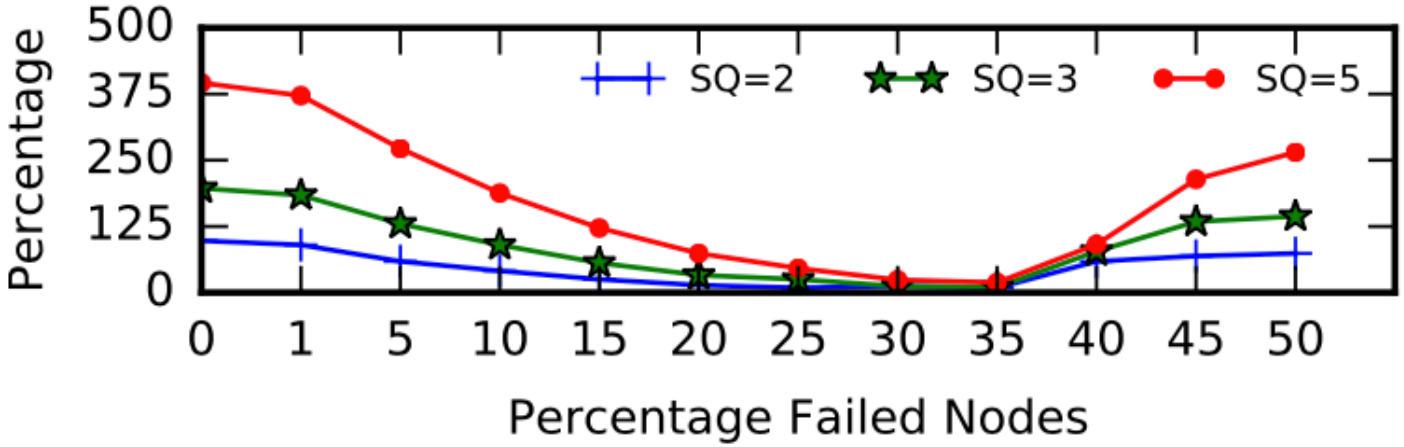
- Migrate away all VMs on server to other servers in cluster
  - Prefer set of already upgraded servers, then pending servers
- Upgrade evacuated server and remove VMs
- Repeat for all servers in cluster without taking down application
- Modified: Picks multiple available migration targets per VM
- (Dst servers + VM) => Composite Speculatively Rep. Migration Op

- <u>Goal</u>: Minimize total time to upgrade cluster

Rolling Software Upgrades Results



(a) Total Relative Upgrade Time Improvement



(c) Relative Additional Migration Attempts