Tardigrade:
Leveraging Lightweight Virtual Machines to Easily and Efficiently Construct Fault-Tolerant Services

Jacob R. Lorch Andrew Baumann

Microsoft Research

Lisa Glendenning

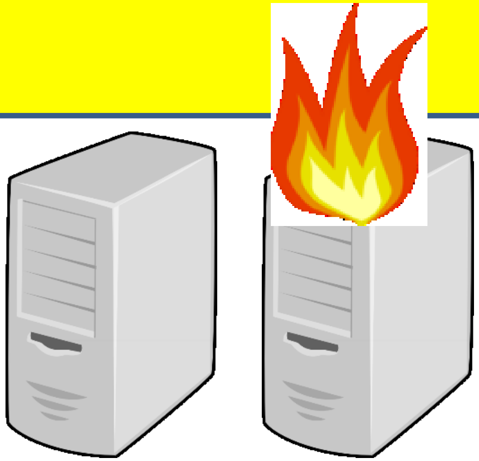
Dutch T. Meyer Andrew Warfield

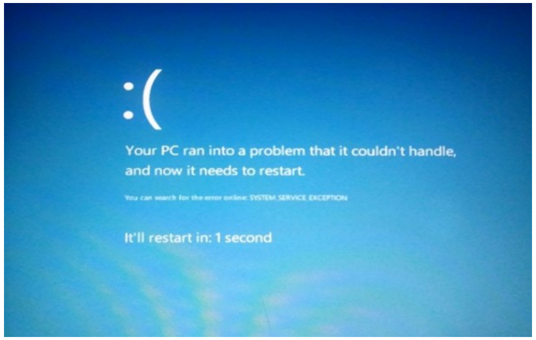
UNIVERSITY · OF · WASHINGTON · 1861 ·

UBC

Our goal:

*Turn **existing** binaries into fault-tolerant services.*





:(

Your PC ran into a problem that it couldn't handle, and now it needs to restart.

You can search for the error online: SYSTEM SERVICE EXCEPTION

It'll restart in: 1 second

Example: FDS Metadata Service

FDS Metadata server

FDS Cluster

[Nightingale et al., OSDI 2012]

Example: FDS Metadata Service

FDS Metadata serve

not safe because two active metadata servers will cause cluster corruption. Our current solution is simple: only one metadata server is allowed to execute at a time. If it fails, an operator must ensure the old one is decommissioned and start a new one. We are experimenting with usin o safely make this process automatic and reduce the impact to availability. Once a

Paxos leader election

FDS Cluster

[Nightingale et al., OSDI 2012]

Techniques for making code fault-tolerant

hav

Use state machine replication library

Better:

Transparently make the binary fault-tolerant

Potential for oversight

- Non-determinism
- Failing to persist state
- Exposing non-persisted data
- Bugs in crash recovery

Outline

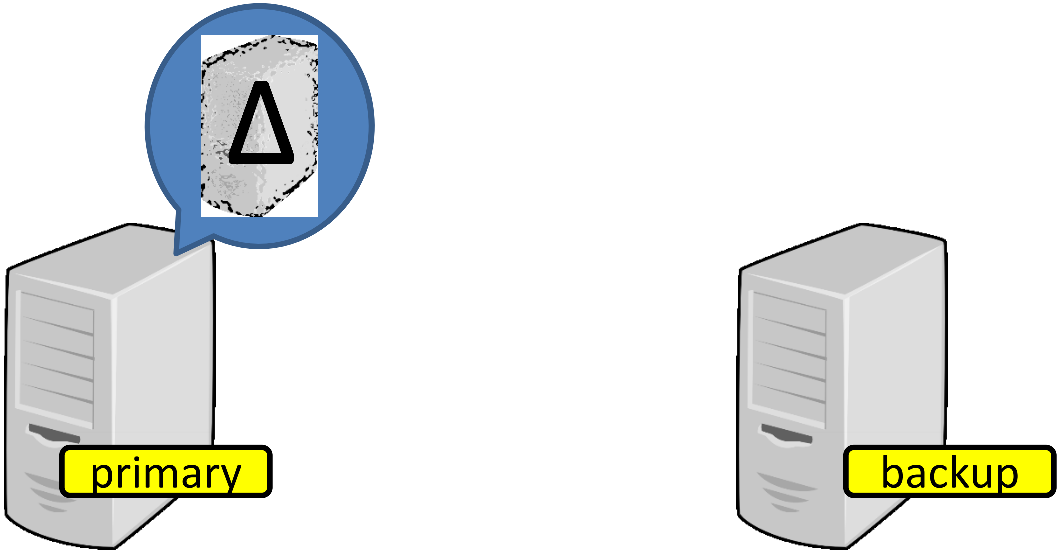
- Motivation
- Background: Asynchronous VM replication
- Our solution: Lightweight VM replication
- Challenges and solutions
- Evaluation

Outline

- Motivation
- Background: Asynchronous VM replication
- Our solution: Lightweight VM replication
- Challenges and solutions
- Evaluation

Asynchronous virtual machine replication - Remus

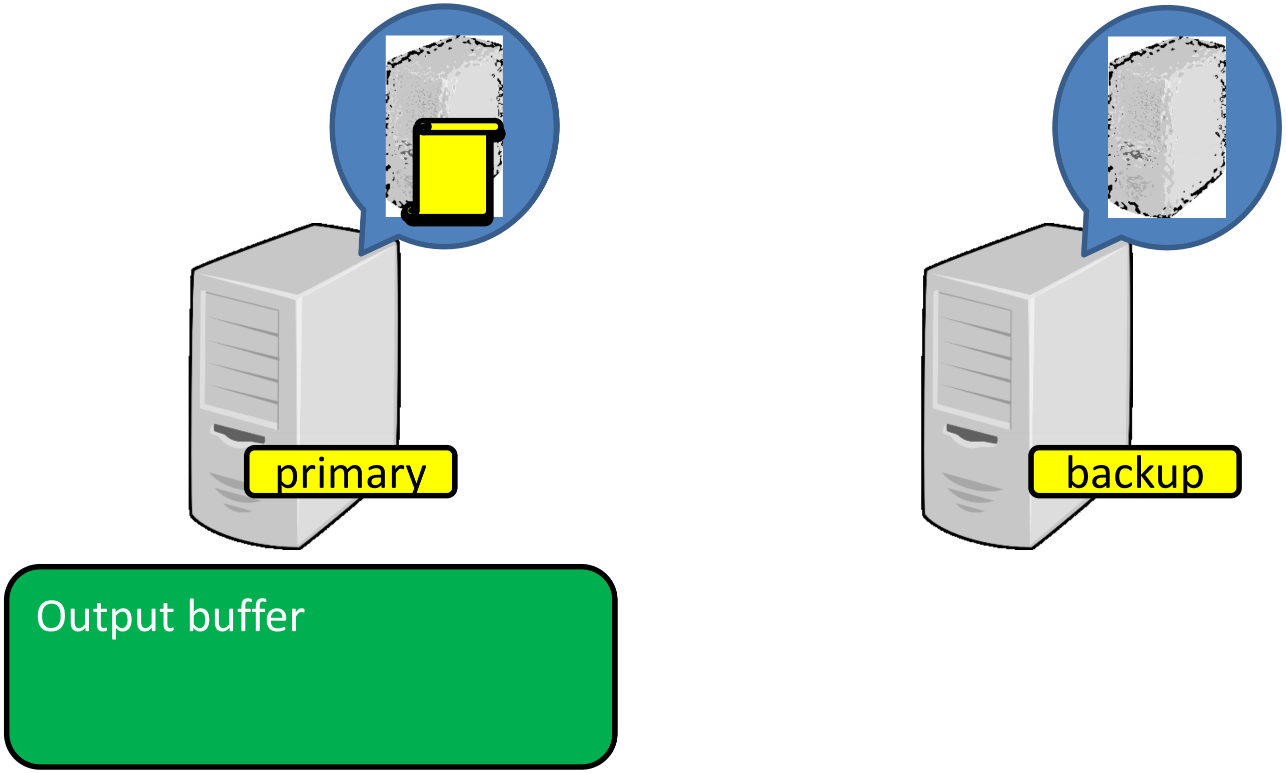
[Cully et al., NSDI 2008]



Primary can crash at any time; backup is always a bit behind.

Asynchronous virtual machine replication - Remus

[Cully et al., NSDI 2008]



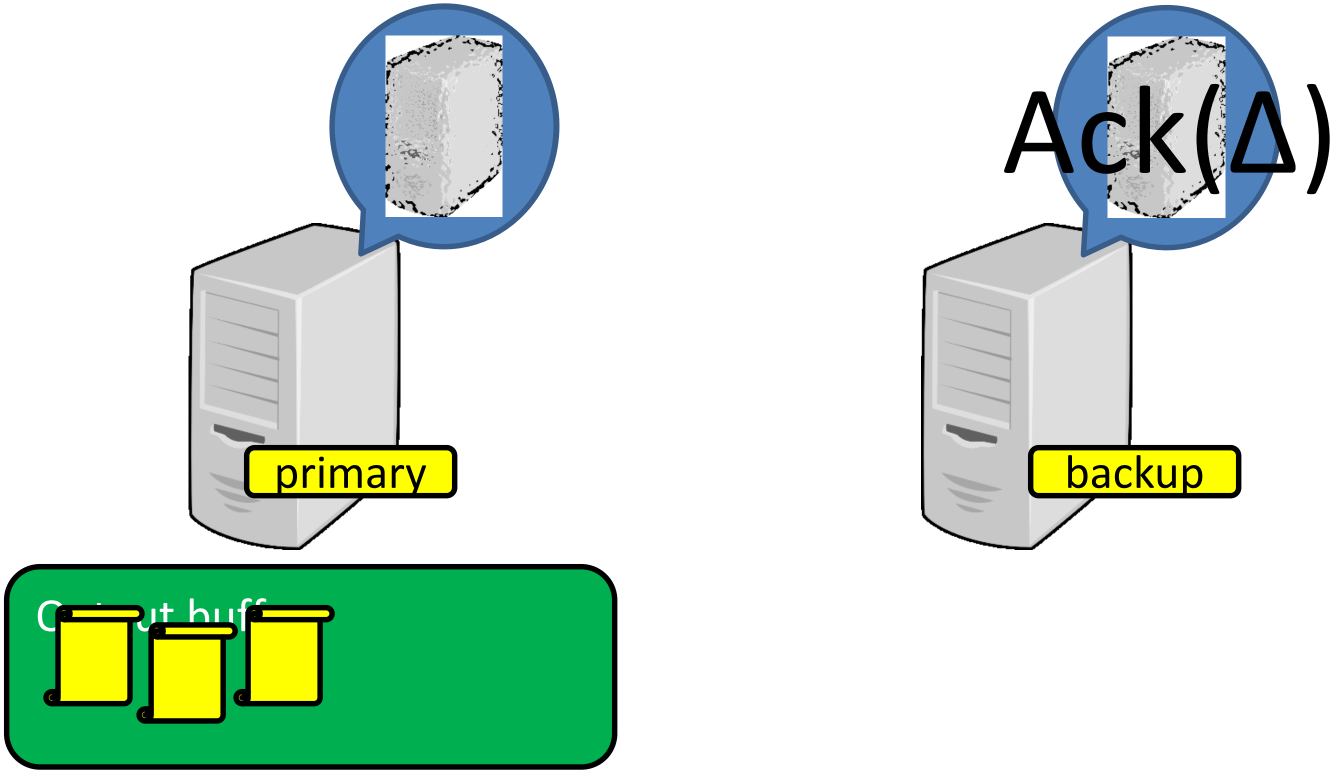
primary

backup

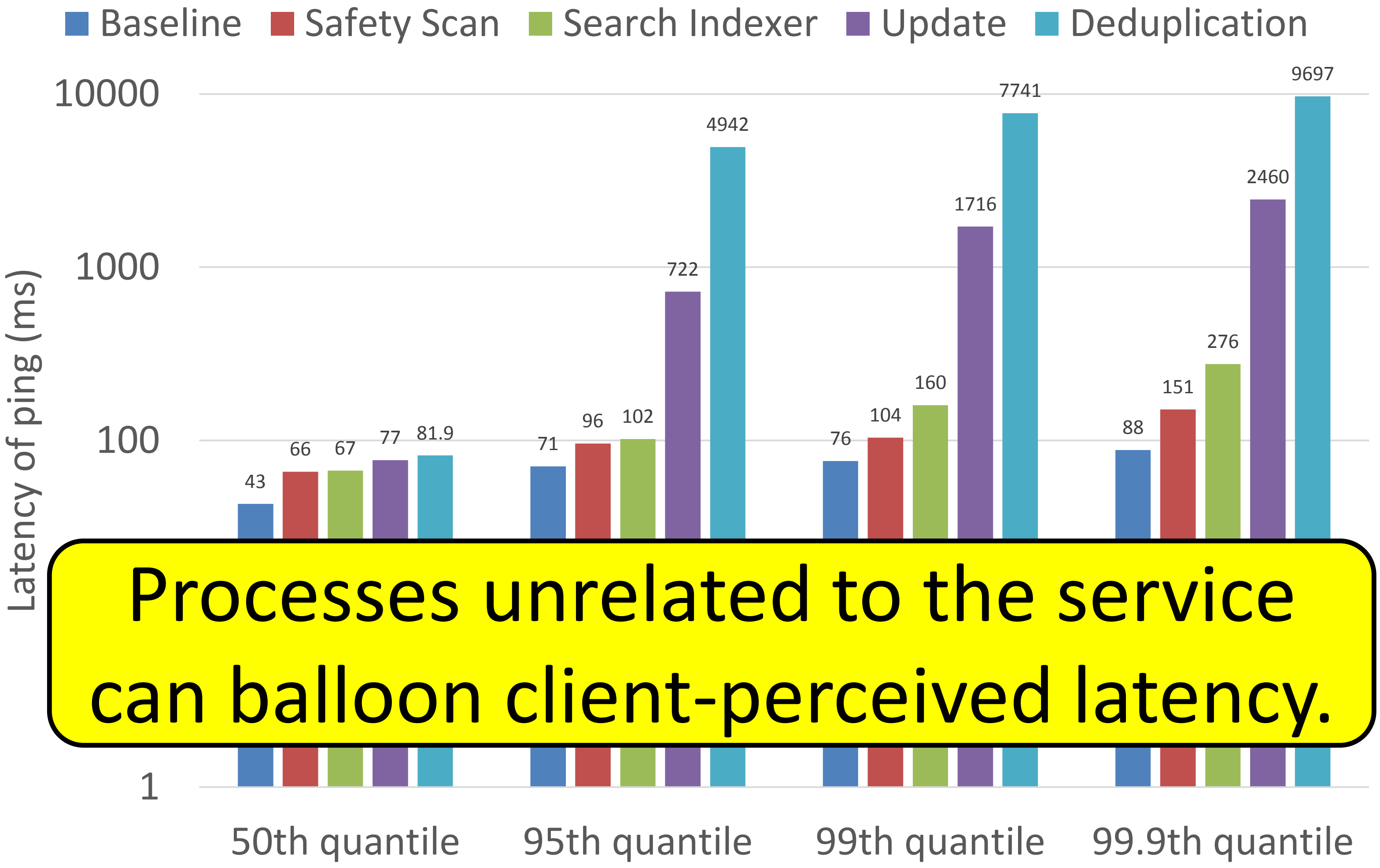
Output buffer

Asynchronous virtual machine replication - Remus

[Cully et al., NSDI 2008]



High VM activity can delay packets

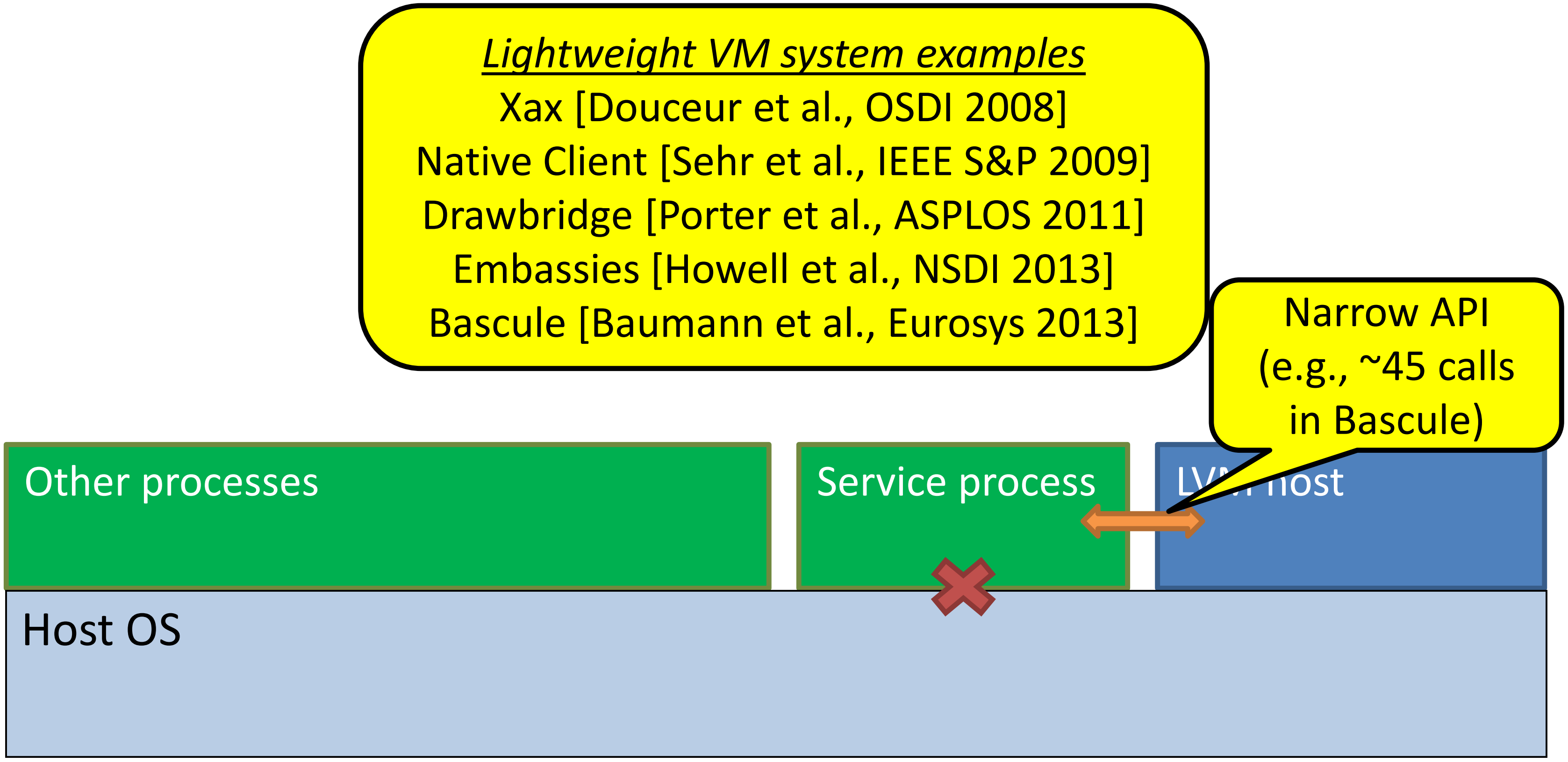


Processes unrelated to the service can balloon client-perceived latency.

Outline

- Motivation
- Background: Asynchronous VM replication
- Our solution: Lightweight VM replication
- Challenges and solutions
- Evaluation

Our solution: Use *lightweight* VMs instead



Lightweight VM system examples

Xax [Douceur et al., OSDI 2008]

Native Client [Sehr et al., IEEE S&P 2009]

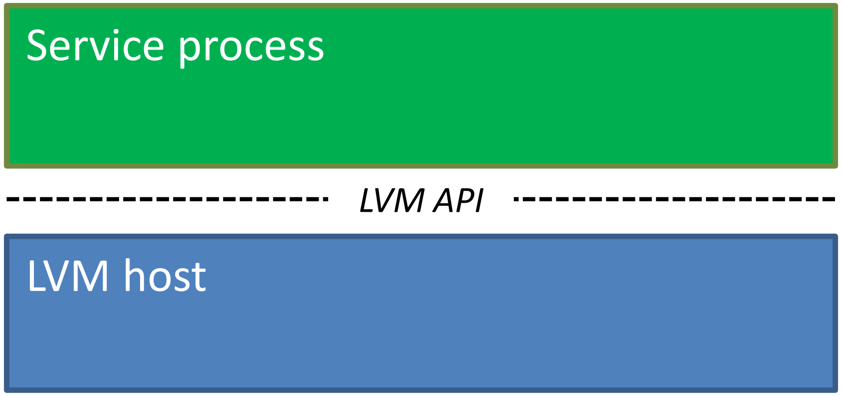
Drawbridge [Porter et al., ASPLOS 2011]

Embassies [Howell et al., NSDI 2013]

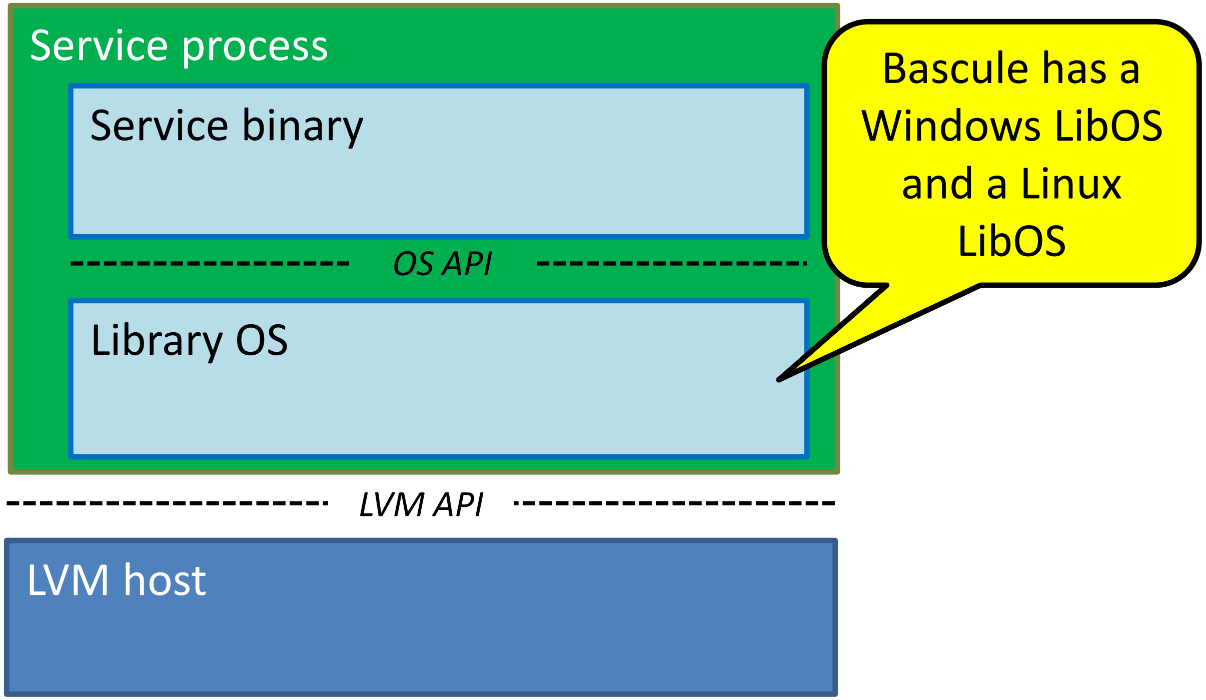
Bascule [Baumann et al., Eurosys 2013]

Narrow API (e.g., ~45 calls in Bascule)

Lightweight VMs can support unmodified binaries via a library OS

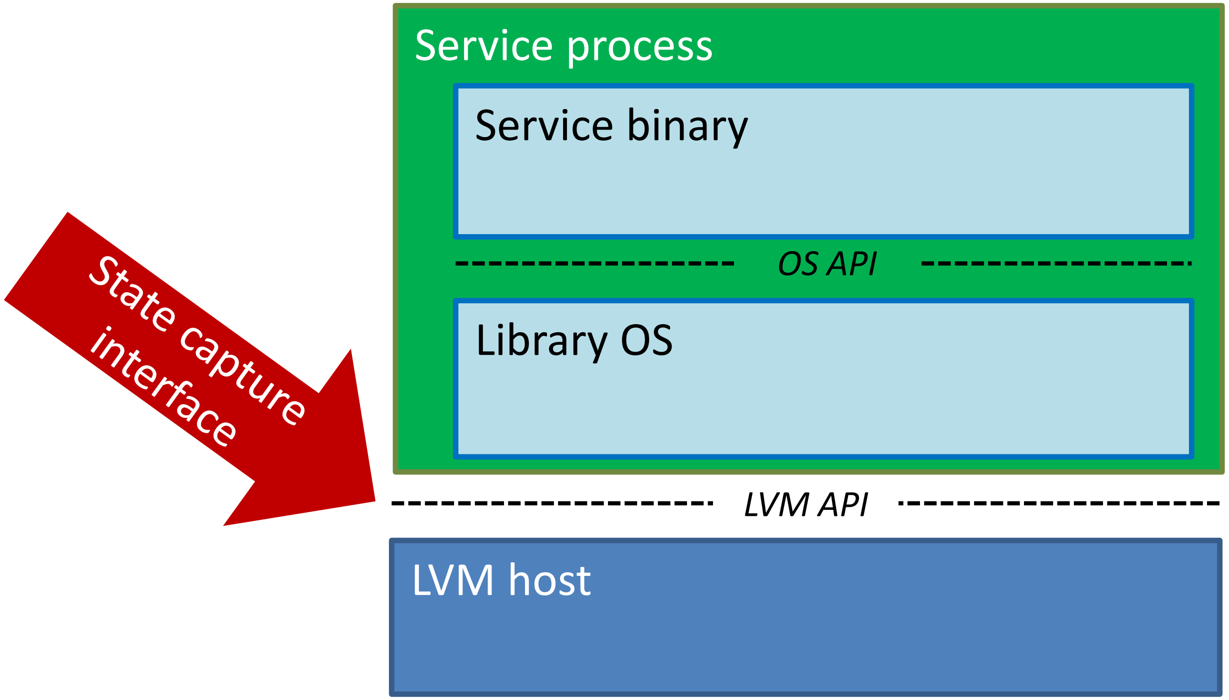


Lightweight VMs can support unmodified binaries via a library OS



Bascule has a Windows LibOS and a Linux LibOS

A lightweight VM is encapsulated by virtue of having a narrow interface



Service process

Service binary

OS API

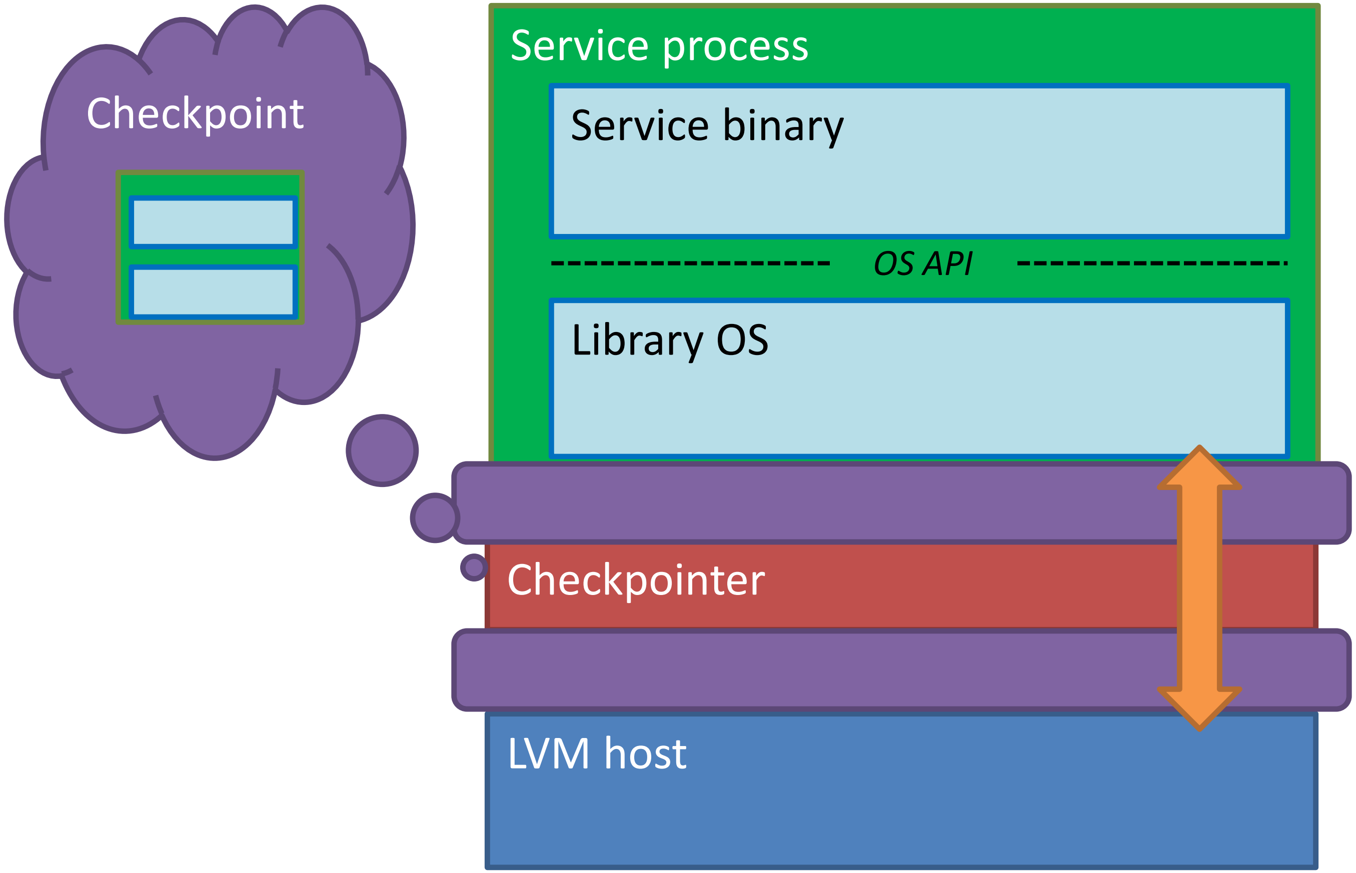
Library OS

LVM API

LVM host

State capture interface

Our approach: Checkpoint by interposing on existing LVM API

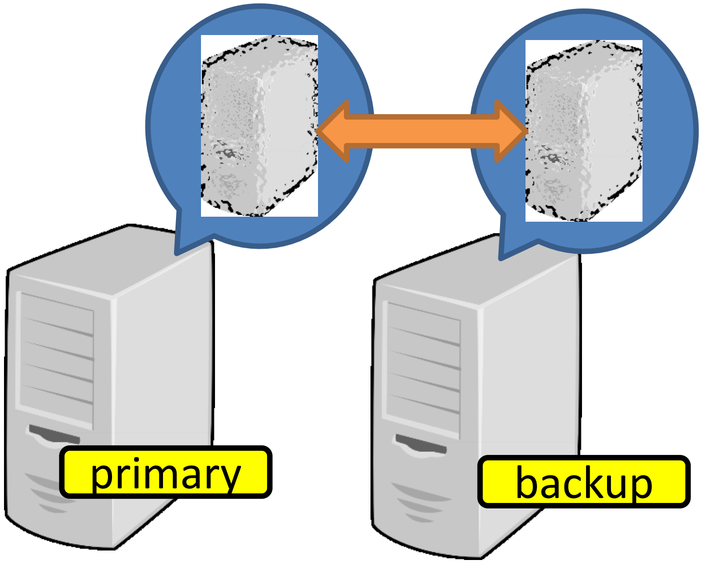


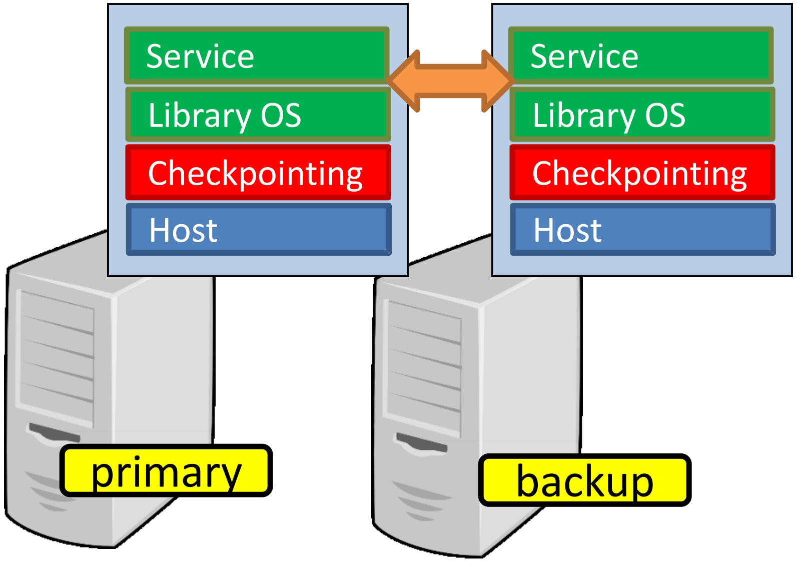
Interposition using existing API means LVM and LibOS don't have to change

[Cully et al., NSDI 2008]

Asynchronous
Virtual
Machine
Replication

Lightweight
Virtual
Machine
Replication



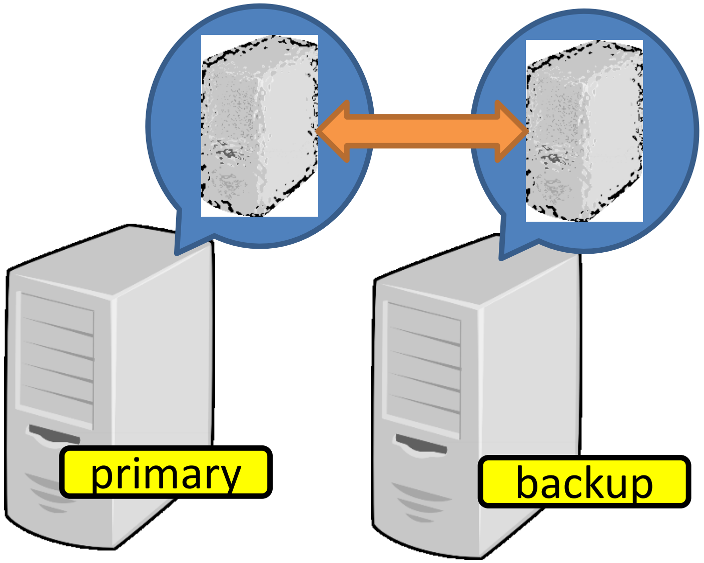


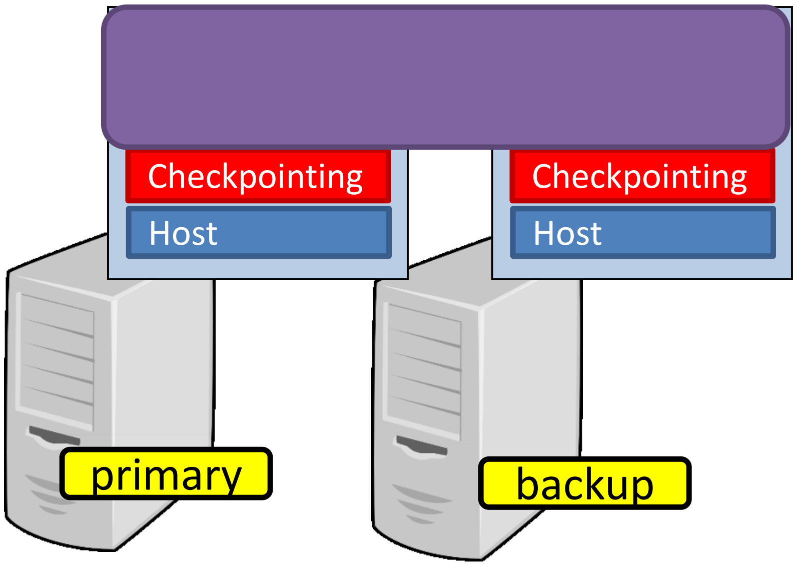
[Cully et al., NSDI 2008]

Asynchr
Virtual
Machine
Replication

Our implementation of LVMR is called Tardigrade

Lightweight
Virtual
Machine
Replication

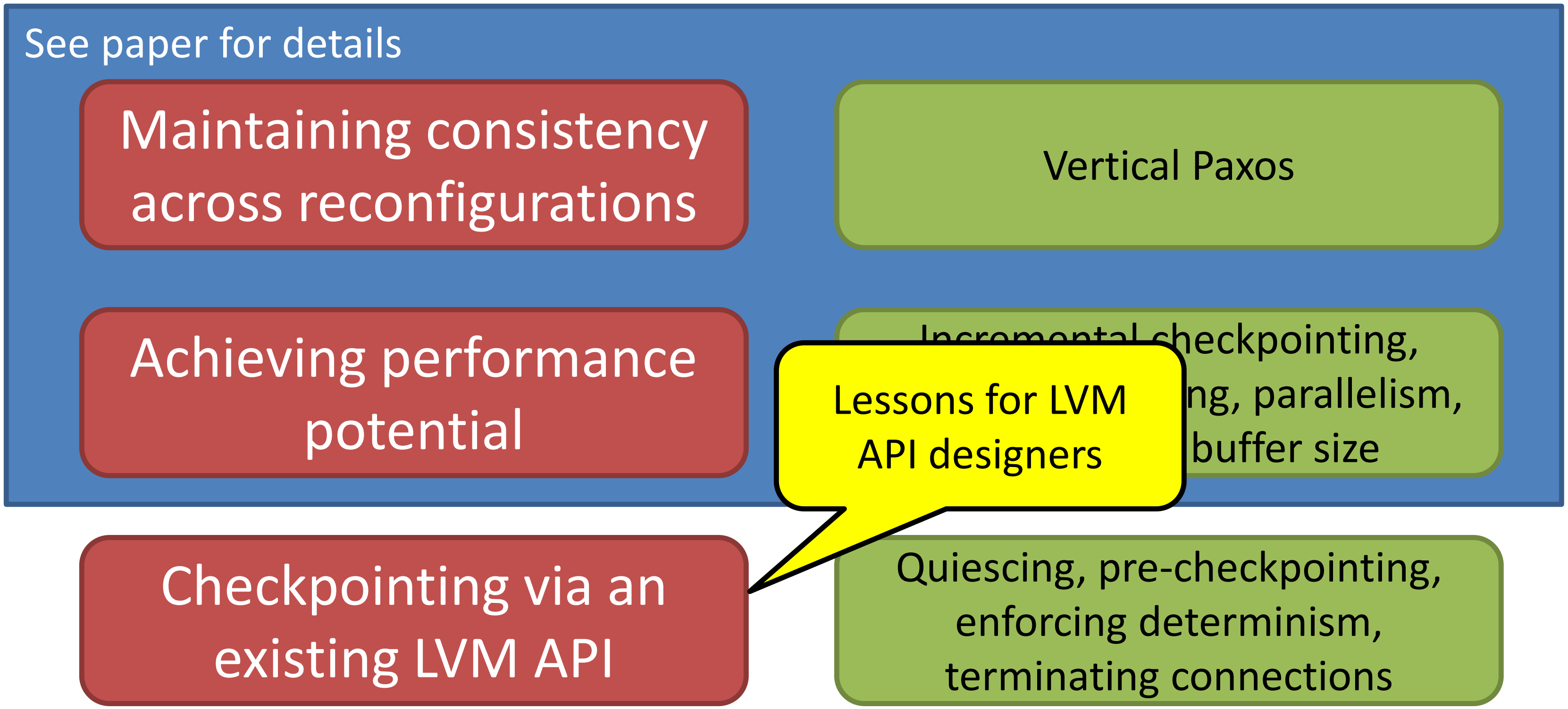




Outline

- Motivation
- Background: Asynchronous VM replication
- Our solution: Lightweight VM replication
- Challenges and solutions
- Evaluation

Practical LVMR poses challenges



Challenges

Solutions

See paper for details

Maintaining consistency across reconfigurations

Vertical Paxos

Achieving performance potential

Incremental checkpointing,
ng, parallelism,
buffer size

Lessons for LVM API designers

Checkpointing via an existing LVM API

Quiescing, pre-checkpointing, enforcing determinism, terminating connections

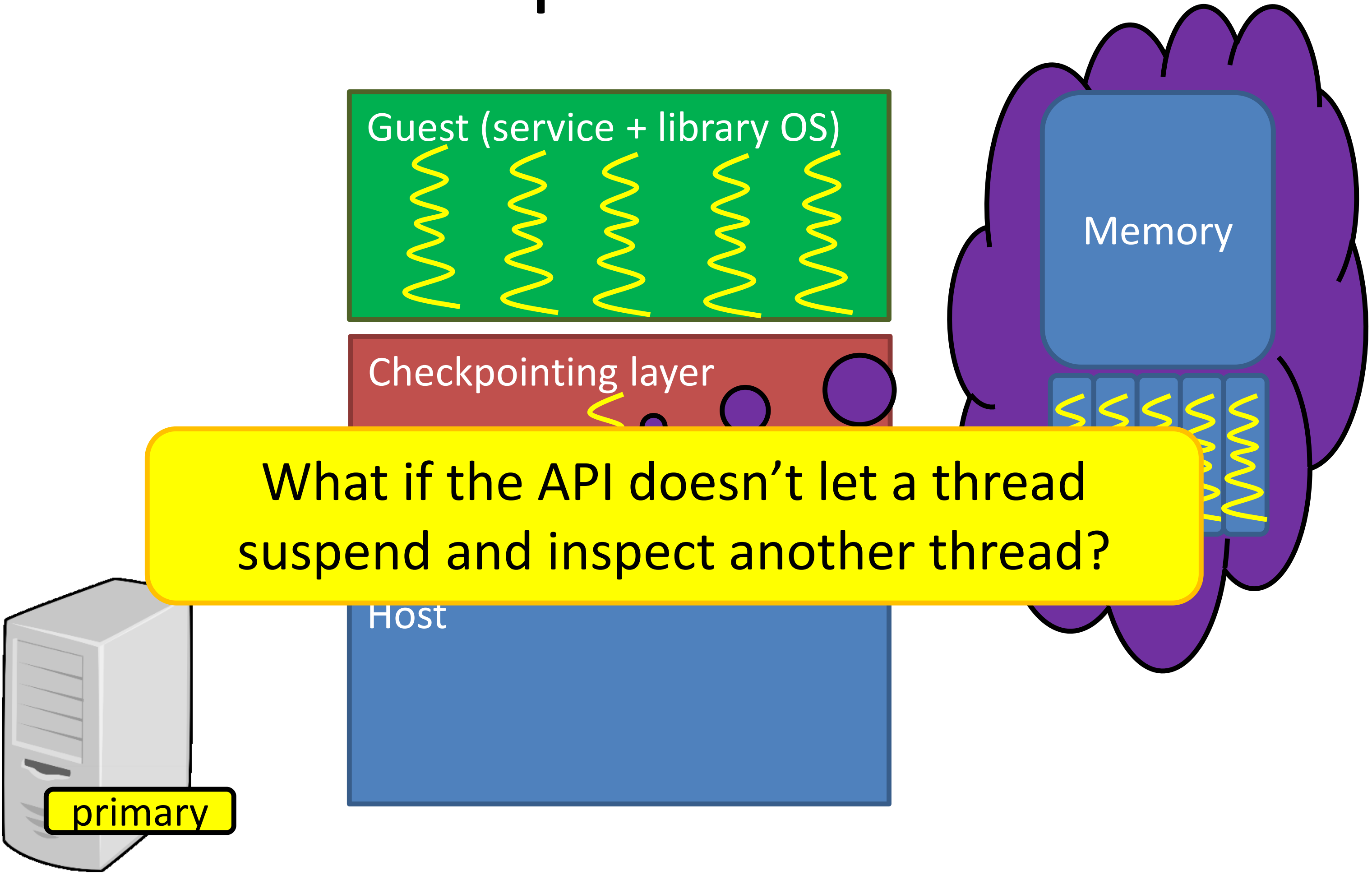
Checkpointing uses certain LVM API features

Feature	Purpose
Ability to track changed memory pages	Efficiently compute checkpoint deltas
Ability to suspend and inspect other threads	Capture consistent snapshot
Determinism when API calls are replayed	Prevent divergence on failover
Host state either replayable or regeneratable	Recreate host state on backup

Features may not always be in LVM APIs

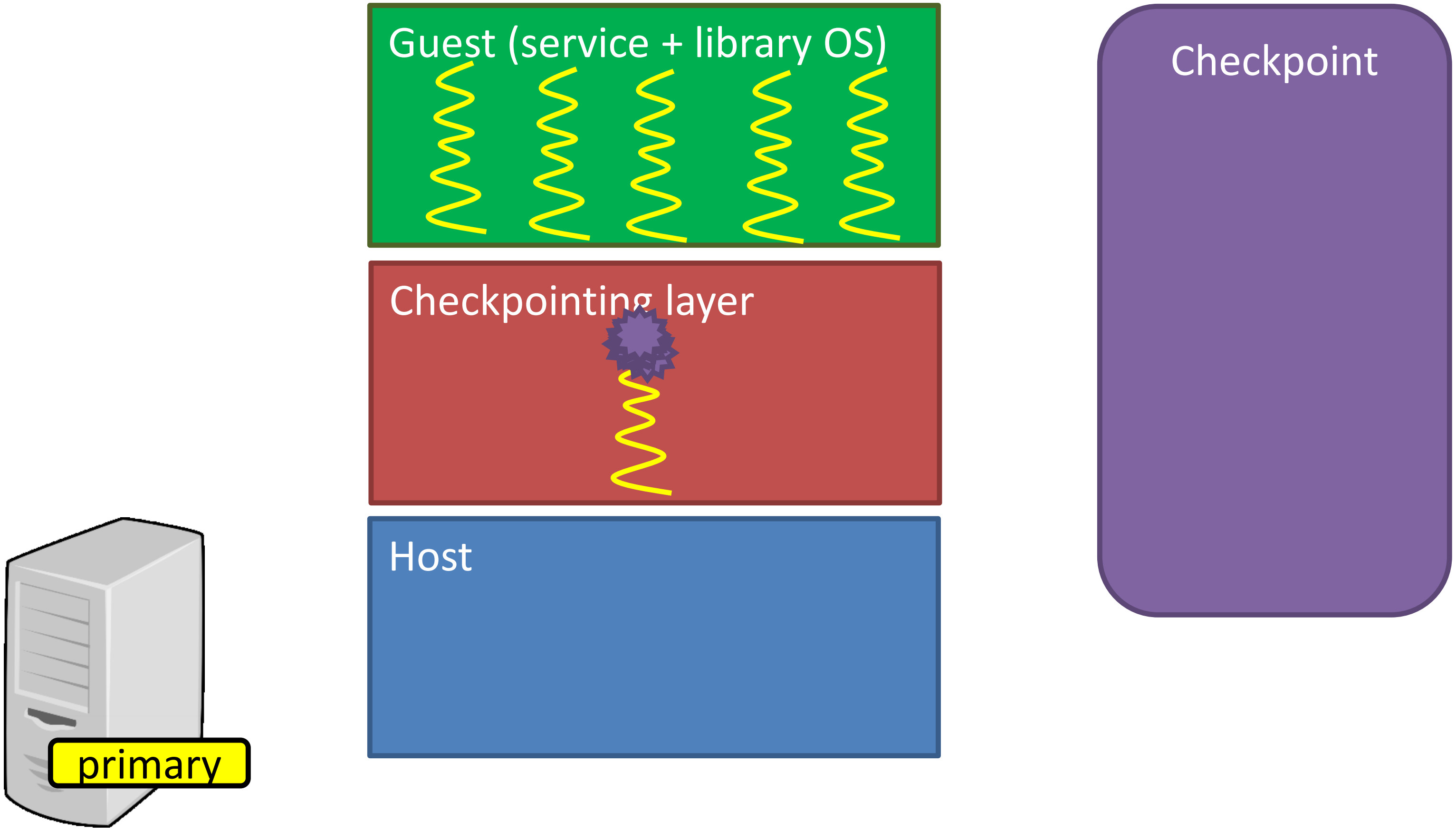
Feature	Workaround
Ability to track changed memory pages	
Missing ability to suspend and inspect other threads	Use exceptions, pre-checkpointing
Non-determinism when API calls are replayed	Hide non-determinism
Host state **not** replayable or regeneratable	Expose divergence as error condition

To capture a checkpoint, we must quiesce and capture all threads' state.

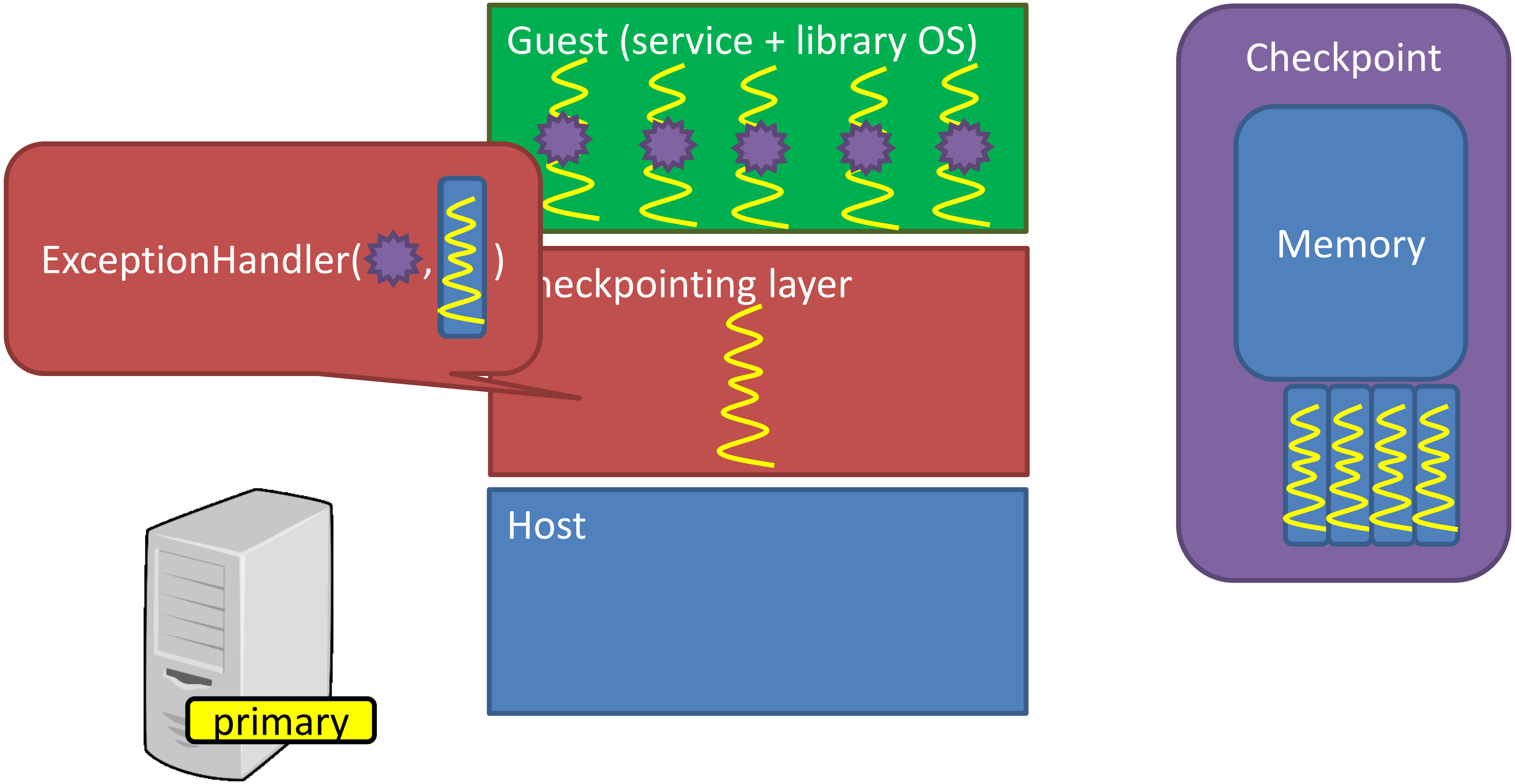


What if the API doesn't let a thread suspend and inspect another thread?

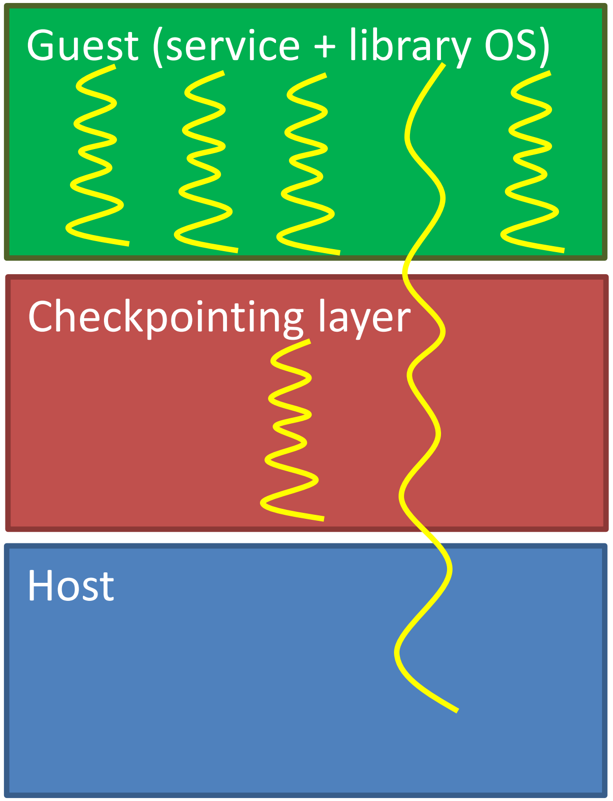
We can use exceptions to quiesce guest threads

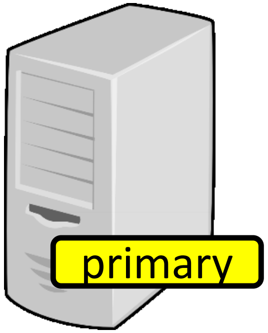


Exception handler quiesces and captures each guest thread's state

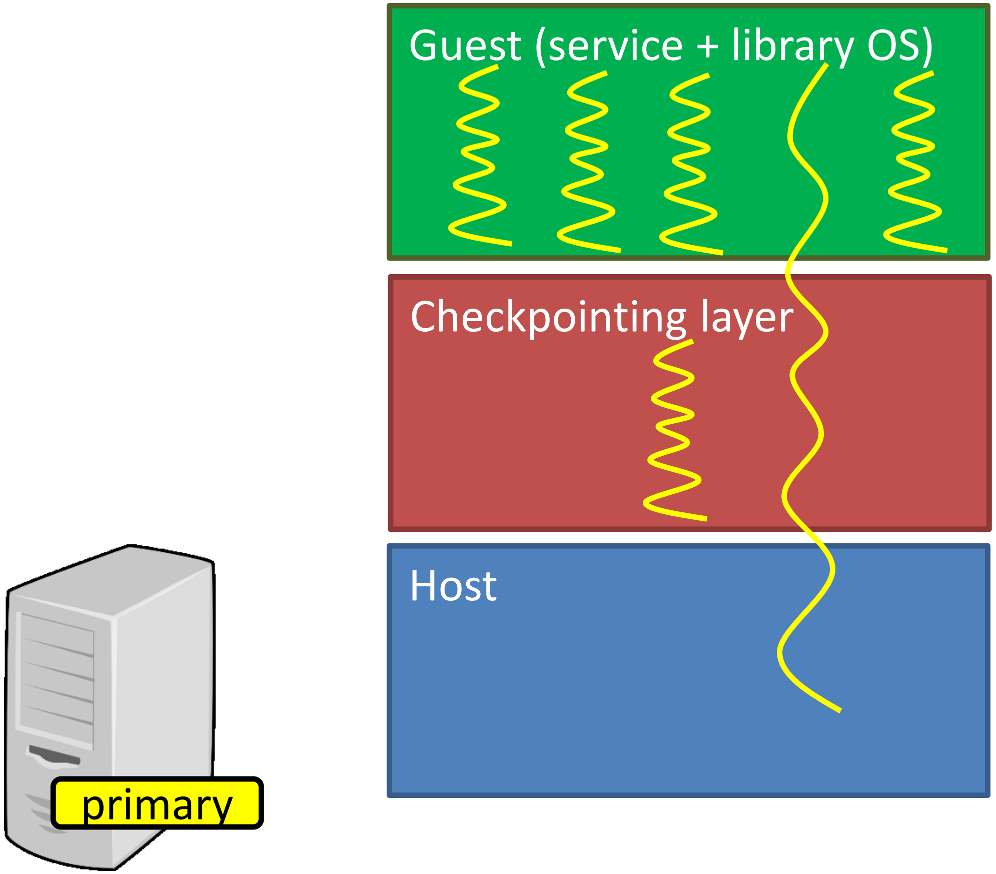


Synchronous system calls complicate quiescence



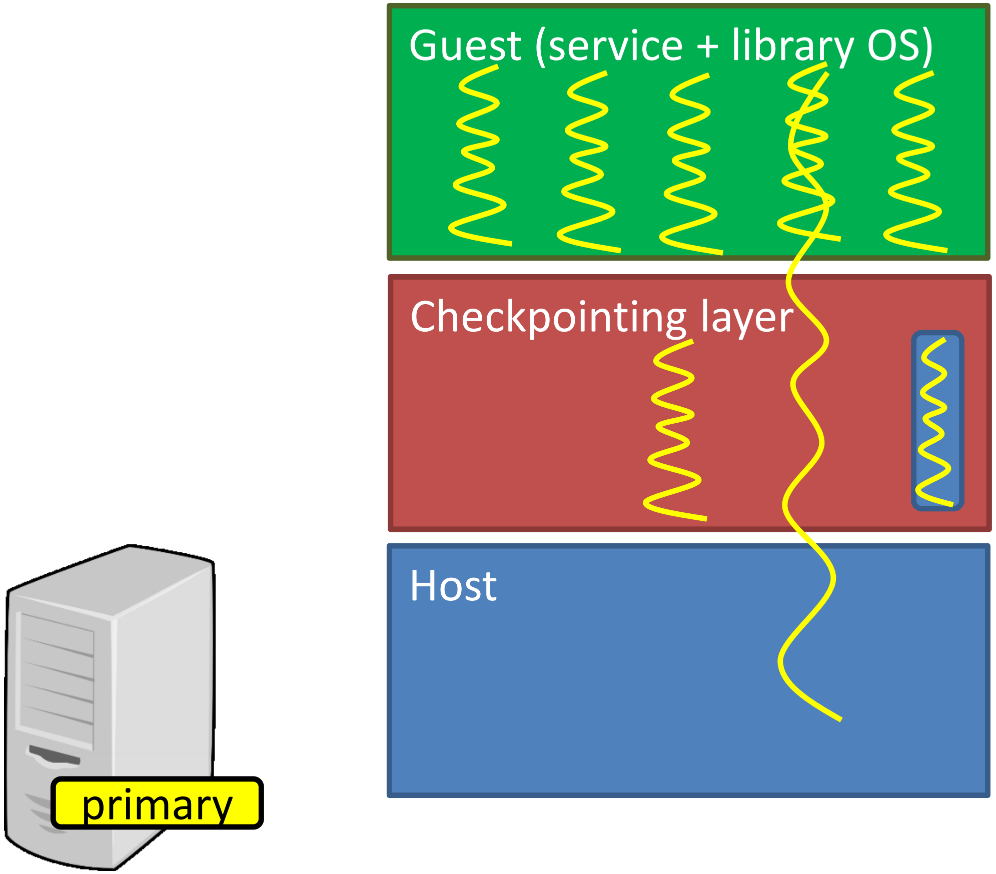


The wait system call is easy to deal with



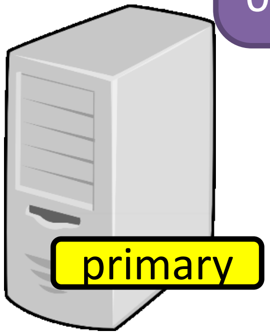
select() file descriptor list
0x1AC
0x3BB
0x907
time-to-checkpoint

General synchronous system calls require *pre-checkpointing*

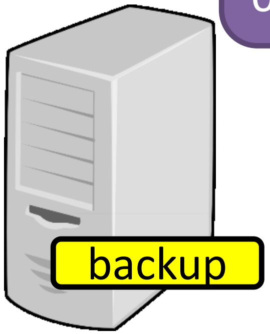


API non-determinism undermines replay

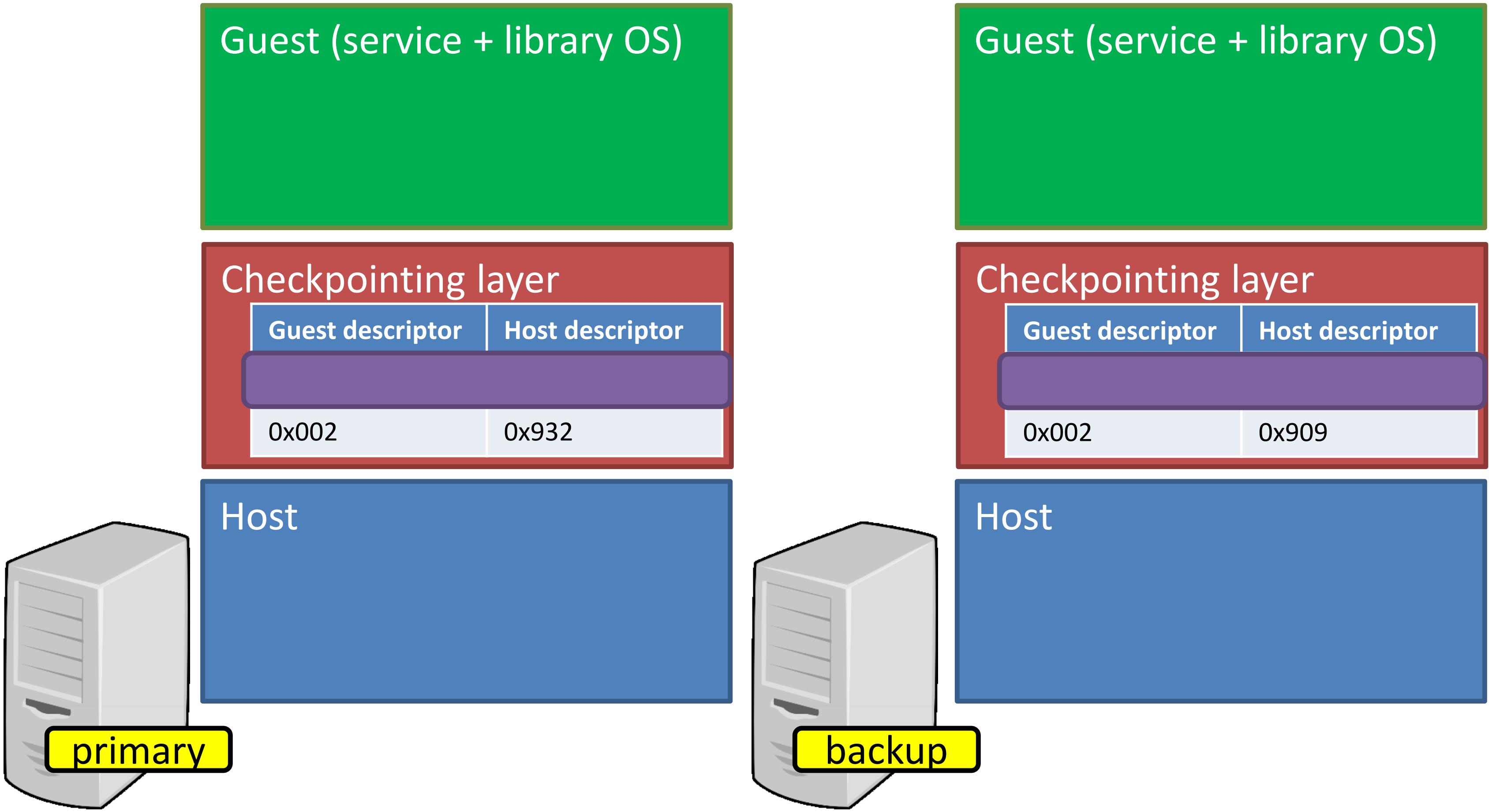
CreateSemaphore() returns descriptor 0xAAA



CreateSemaphore() returns descriptor 0xBBB



An indirection table can hide non-determinism



Guest (service + library OS)

Checkpointing layer

Guest descriptor	Host descriptor
0x002	0x932

Host

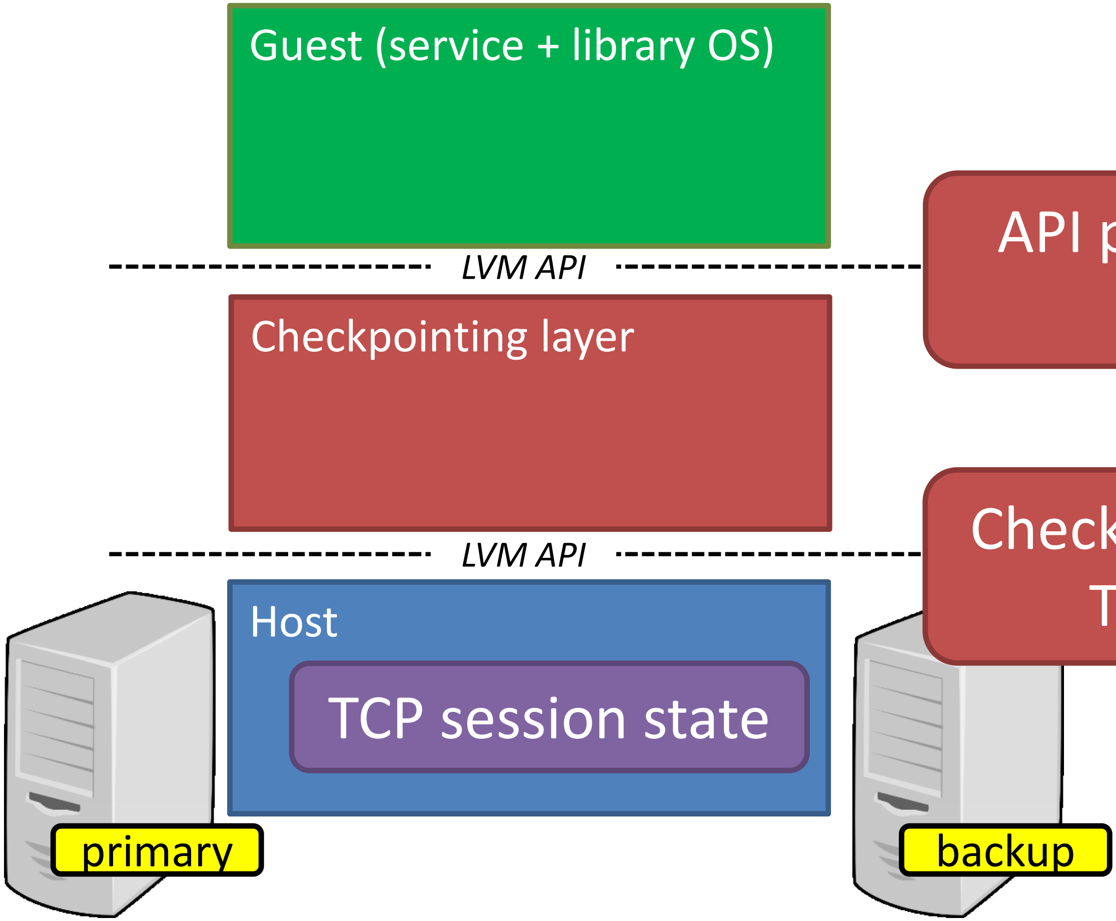
Guest (service + library OS)

Checkpointing layer

Guest descriptor	Host descriptor
0x002	0x909

Host

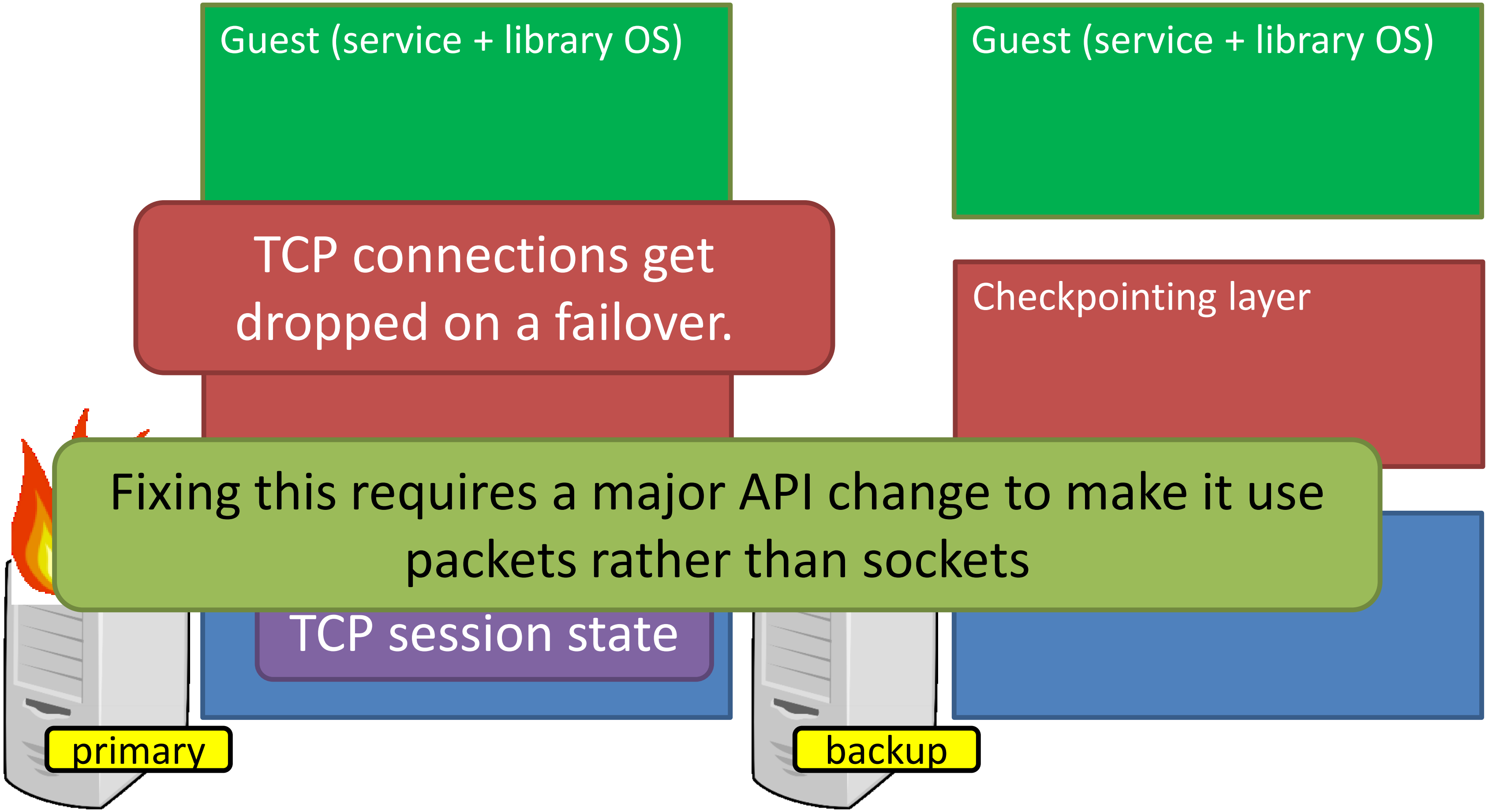
State external to guest needs to be replayable or regeneratable



API provides sockets, not packets

Checkpointer can’t capture TCP session state!

System-specific modifications may be necessary



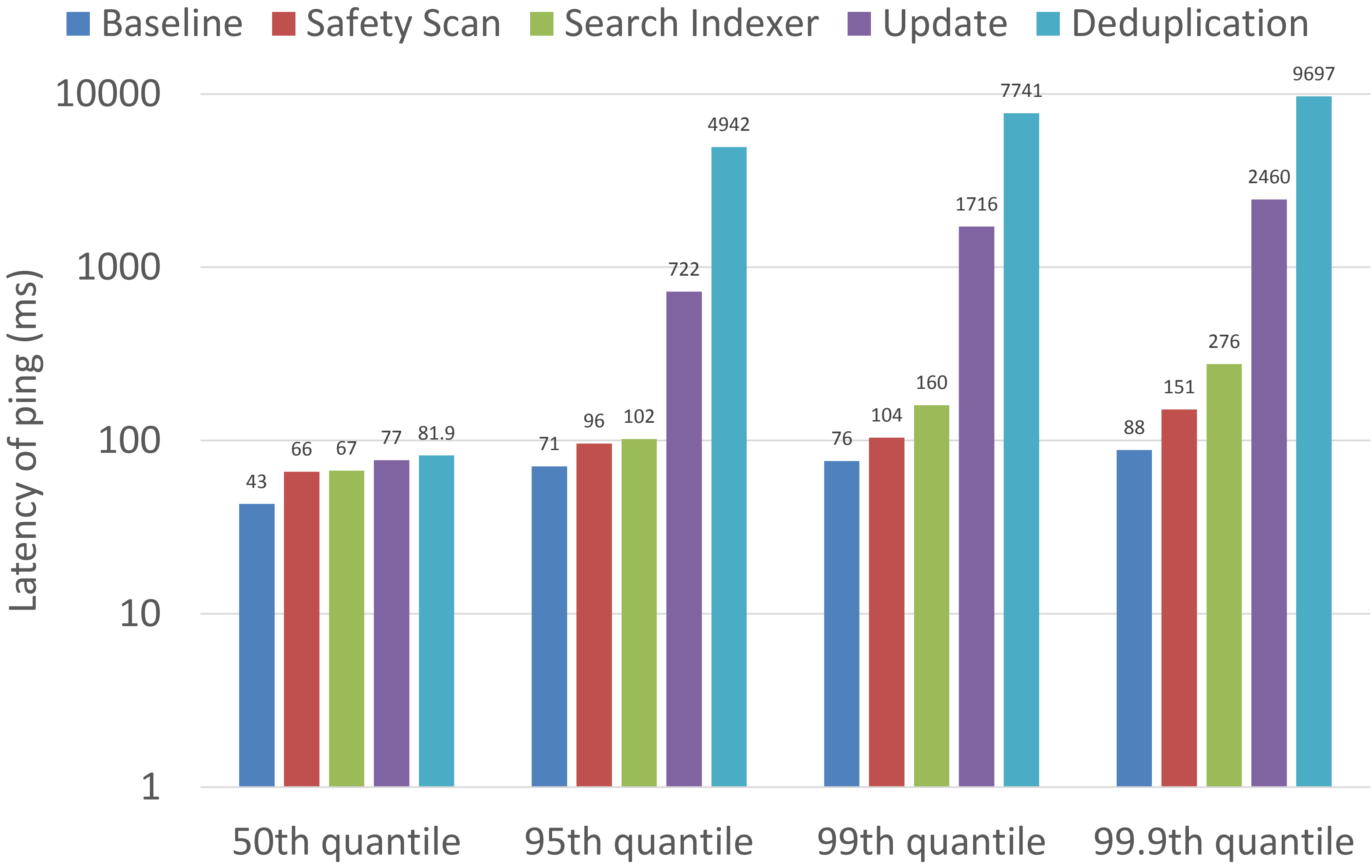
TCP connections get dropped on a failover.

Fixing this requires a major API change to make it use packets rather than sockets

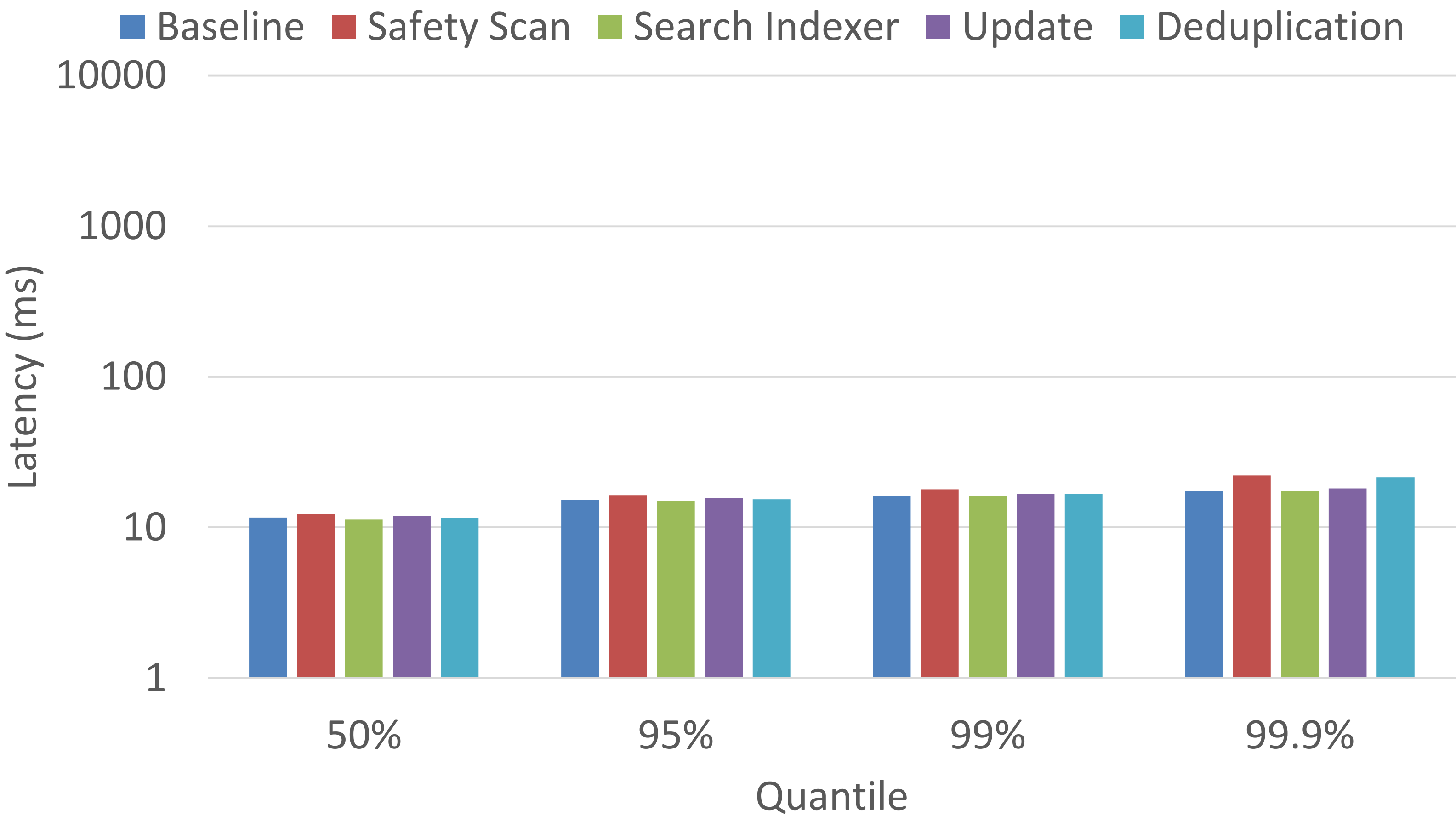
Outline

- Motivation
- Background: Asynchronous VM replication
- Our solution: Lightweight VM replication
- Challenges and solutions
- Evaluation

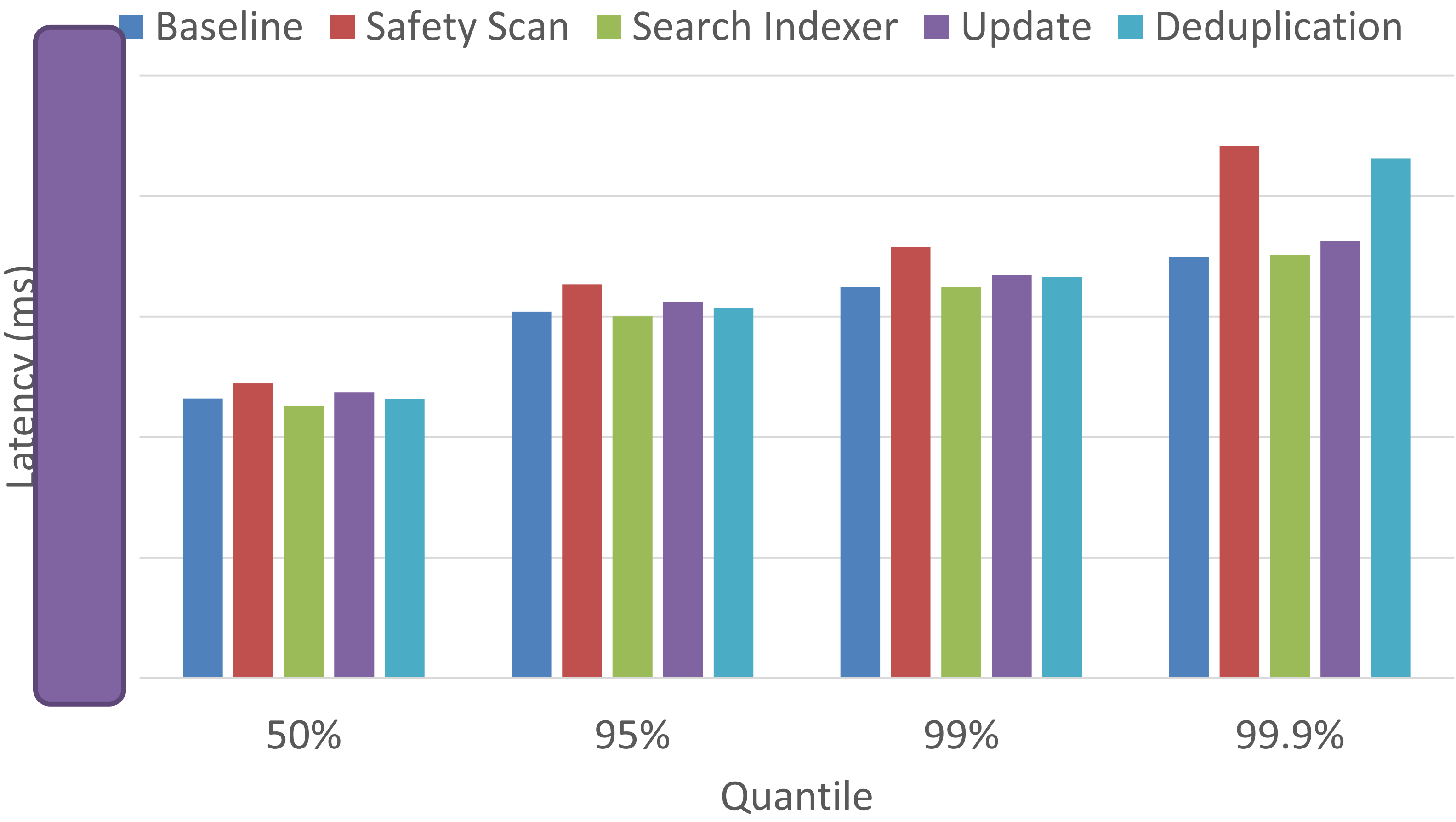
Effect of external processes - Remus



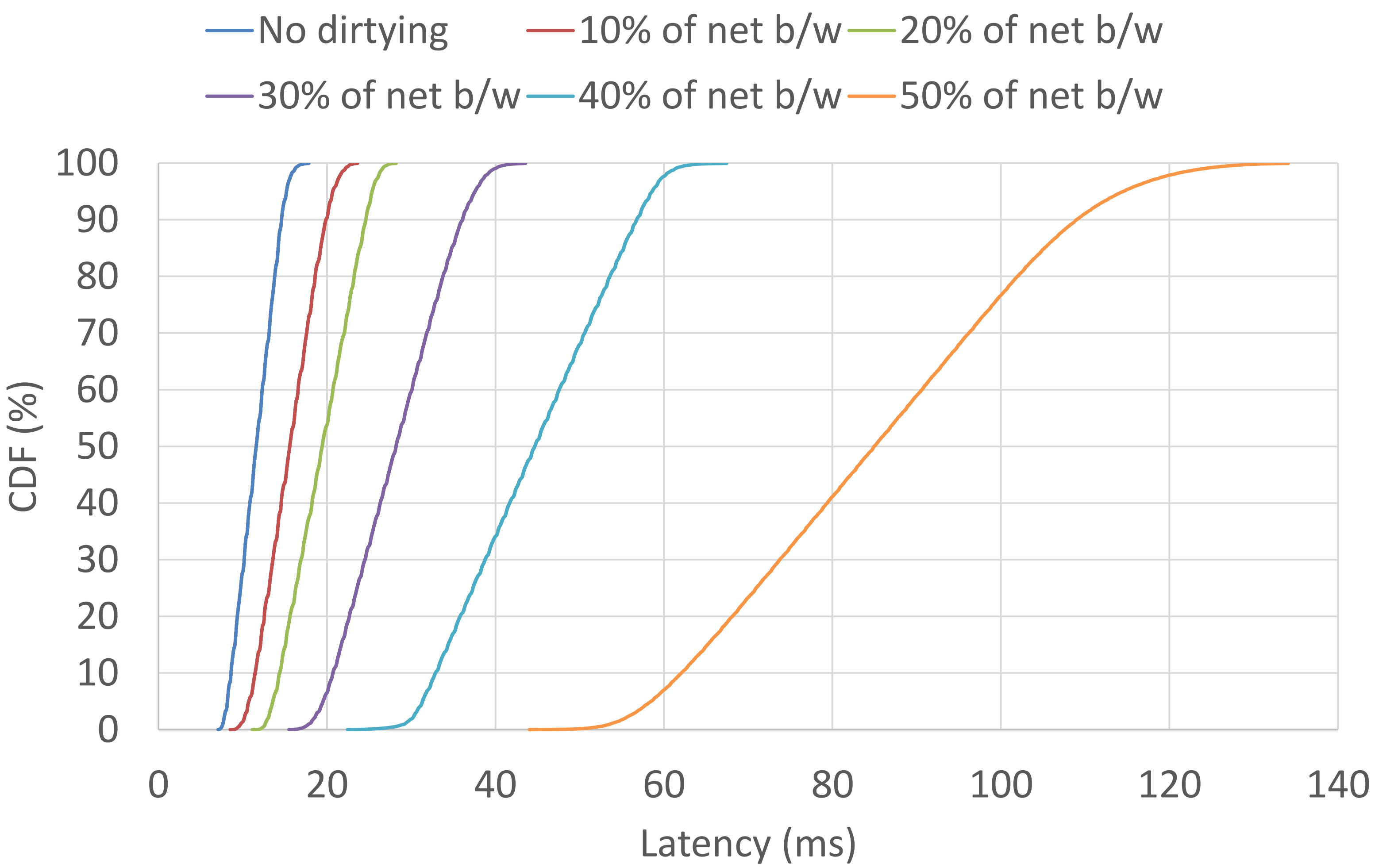
Effect of external processes - Tardigrade



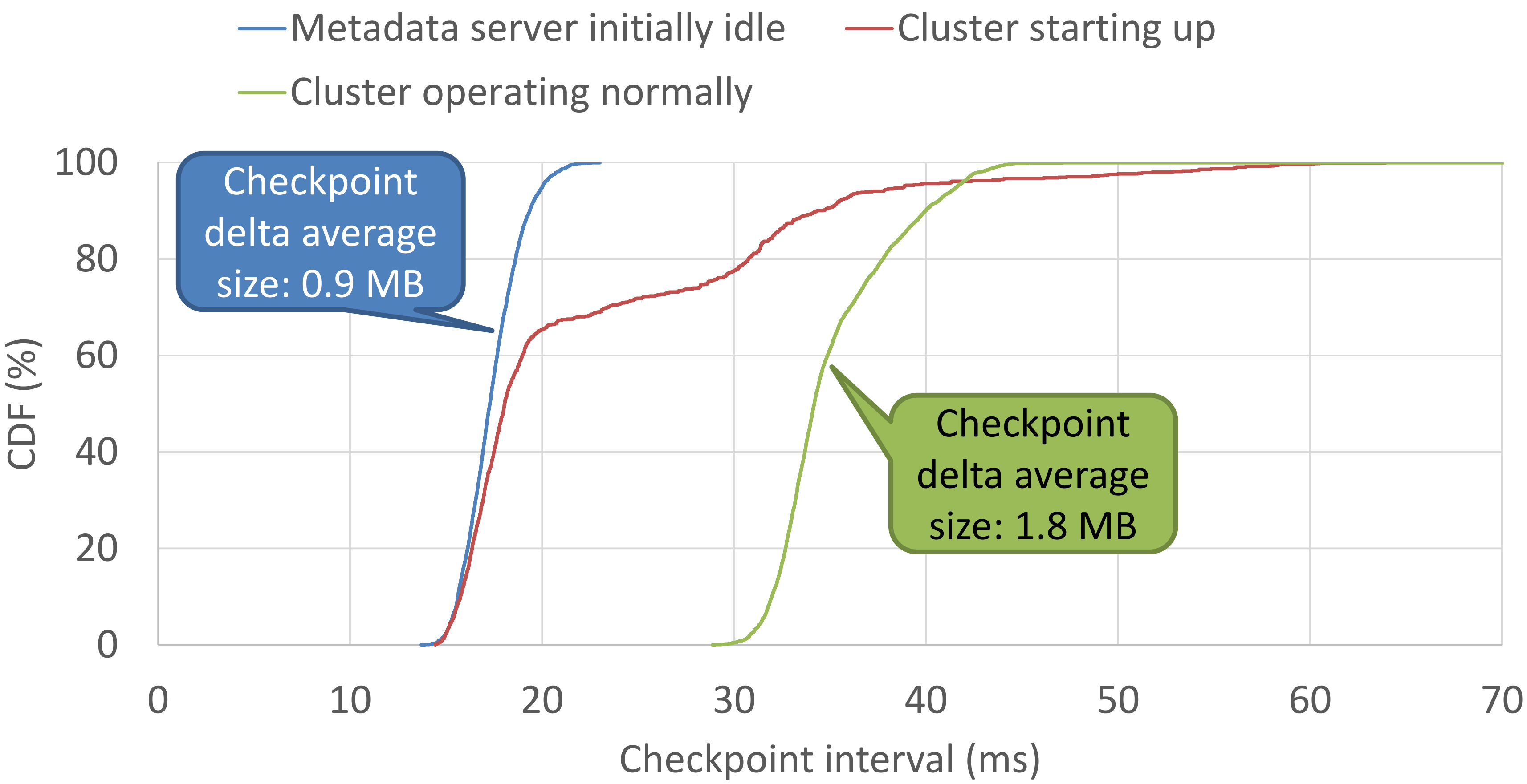
Effect of external processes - Tardigrade



Memory dirtying affects checkpoint latency



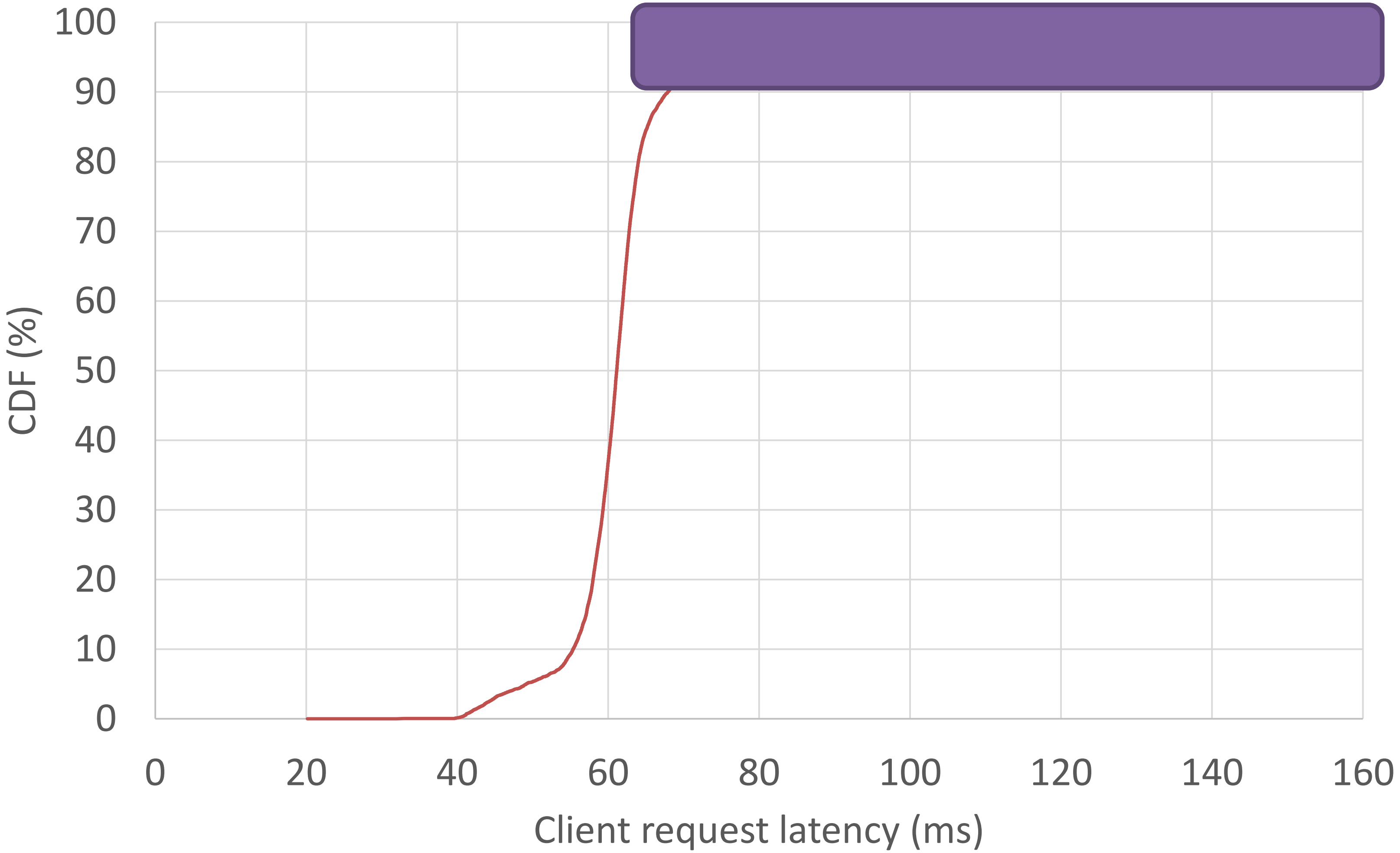
FDS metadata service



Checkpoint delta average size: 0.9 MB

Checkpoint delta average size: 1.8 MB

ZKLite, a simple non-fault-tolerant Java implementation of the Zookeeper API



Conclusions

No changes to binaries needed, making deployment simple

Re ly

gets:

s

es

s

Li

Lightweight VM replication is practical for making existing service binaries fault-tolerant

Reasonable performance if memory dirtying rate and load are low