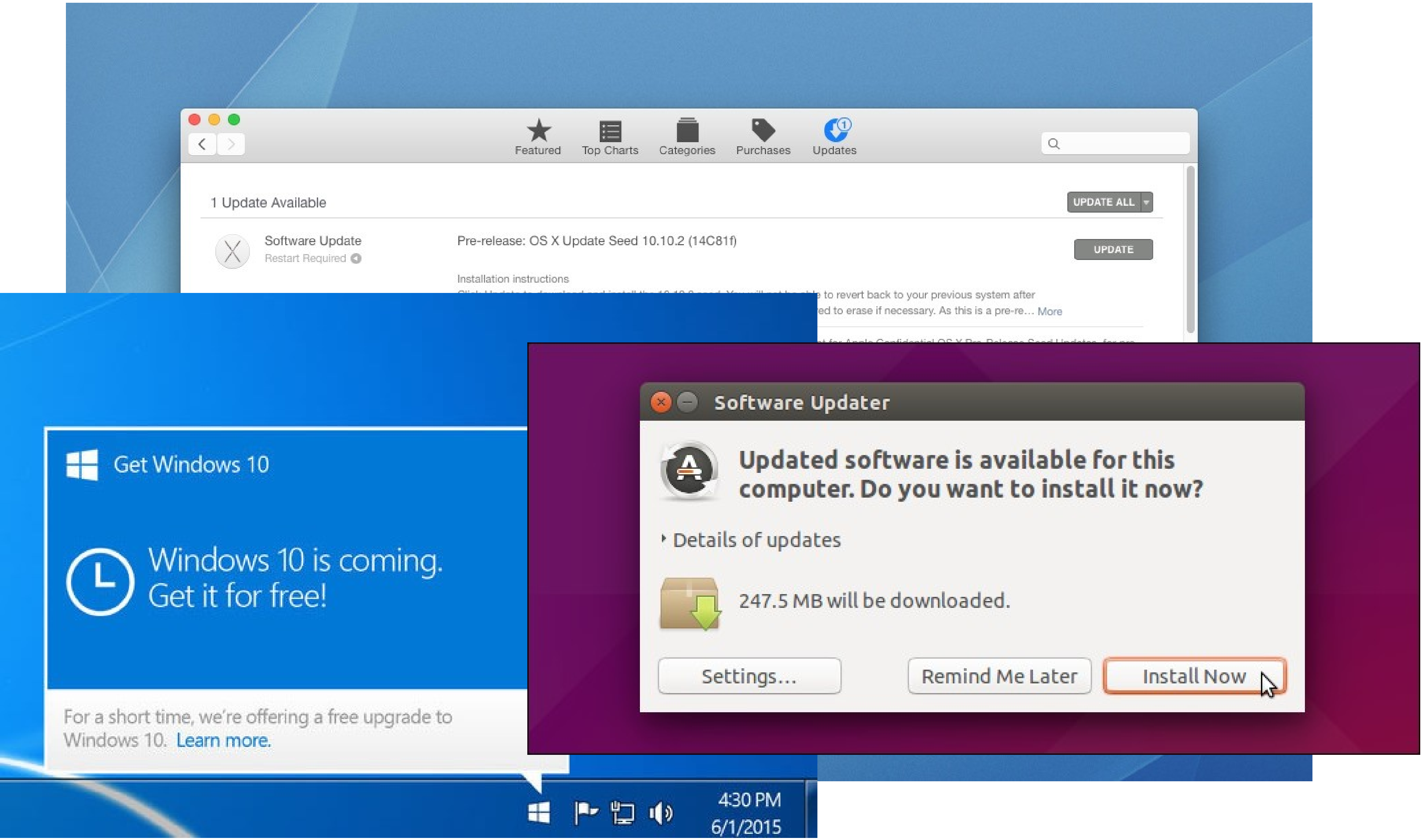
Instant OS Updates via Userspace Checkpoint-and-Restart

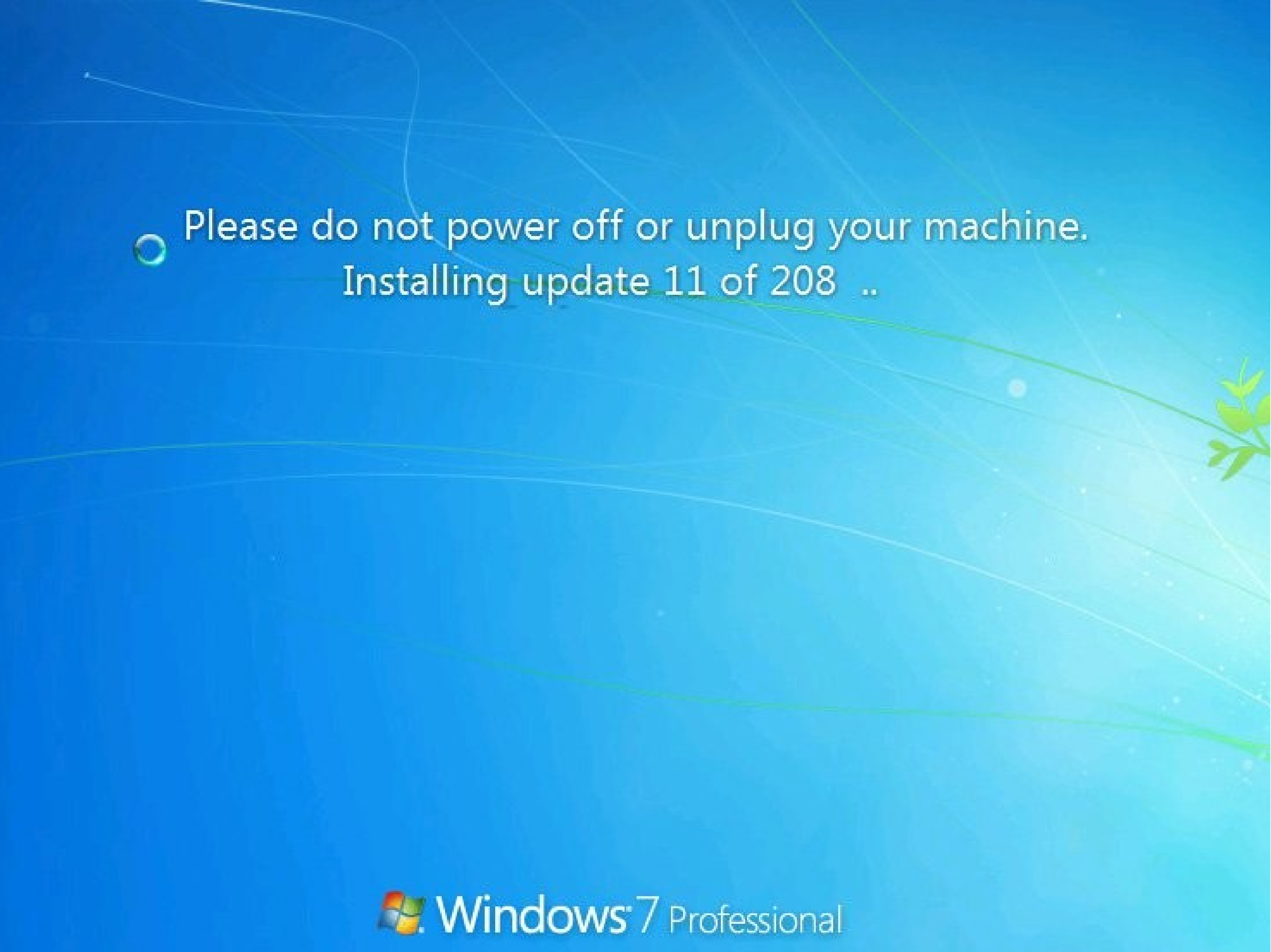
Sanidhya Kashyap, Changwoo Min, Byoungyoung Lee,
Taesoo Kim, Pavel Emelyanov

OS updates are prevalent



And OS updates are unavoidable

- Prevent known, state-of-the-art attacks
  - Security patches
- Adopt new features
  - New I/O scheduler features
- Improve performance
  - Performance patches


Please do not power off or unplug your machine.
Installing update 11 of 208 ..
Windows 7 Professional

Unfortunately, system updates come at a cost

- Unavoidable downtime
- Potential risk of system failure

Unfortunately, system updates come at a cost

- Unavoidable downtime
- Potential risk of system failure

THE FINANCIAL AND OTHER COSTS OF DATA CENTER DOWNTIME

Posted on March 30, 2014 by Mary Hiers

...

Amazon had 49 minutes of downtime in Ja
an estimated $4 million in lost sales, or $8
minute of the outage. When Google went
2013, it cost an estimated $545,000 in
minute. Obviously, downtime costs big co
losses than small companies, but regardl
an expense nobody wants to face. Here are some other important facts and figures.

$109k per minute
Hidden costs (losing customers)

Example: memcached

- Facebook's memcached servers incur a downtime of 2-3 hours per machine
  - Warming cache (e.g., 120 GB) over the network

Example: memcached

- Facebook's memcached servers incur a downtime of 2-3 hours per machine
  - Warming cache (e.g., 120 GB) over the network

Our approach updates OS in 3 secs for 32GB of data from v3.18 to v3.19 for Ubuntu / Fedora releases

Existing practices for OS updates

- Dynamic Kernel Patching (e.g., kpatch, ksplice)
  - **Problem:** only support minor patches
- Rolling Update (e.g., Google, Facebook, etc)
  - **Problem:** inevitable downtime and requires careful planning

Existing practices for OS updates

Losing application state is inevitable

→ Restoring memcached takes 2-3 hours

- Rolling Update (e.g., Google, Facebook, etc)
  - **Problem:** inevitable downtime and requires careful planning

Existing practices for OS updates

Losing application state is inevitable

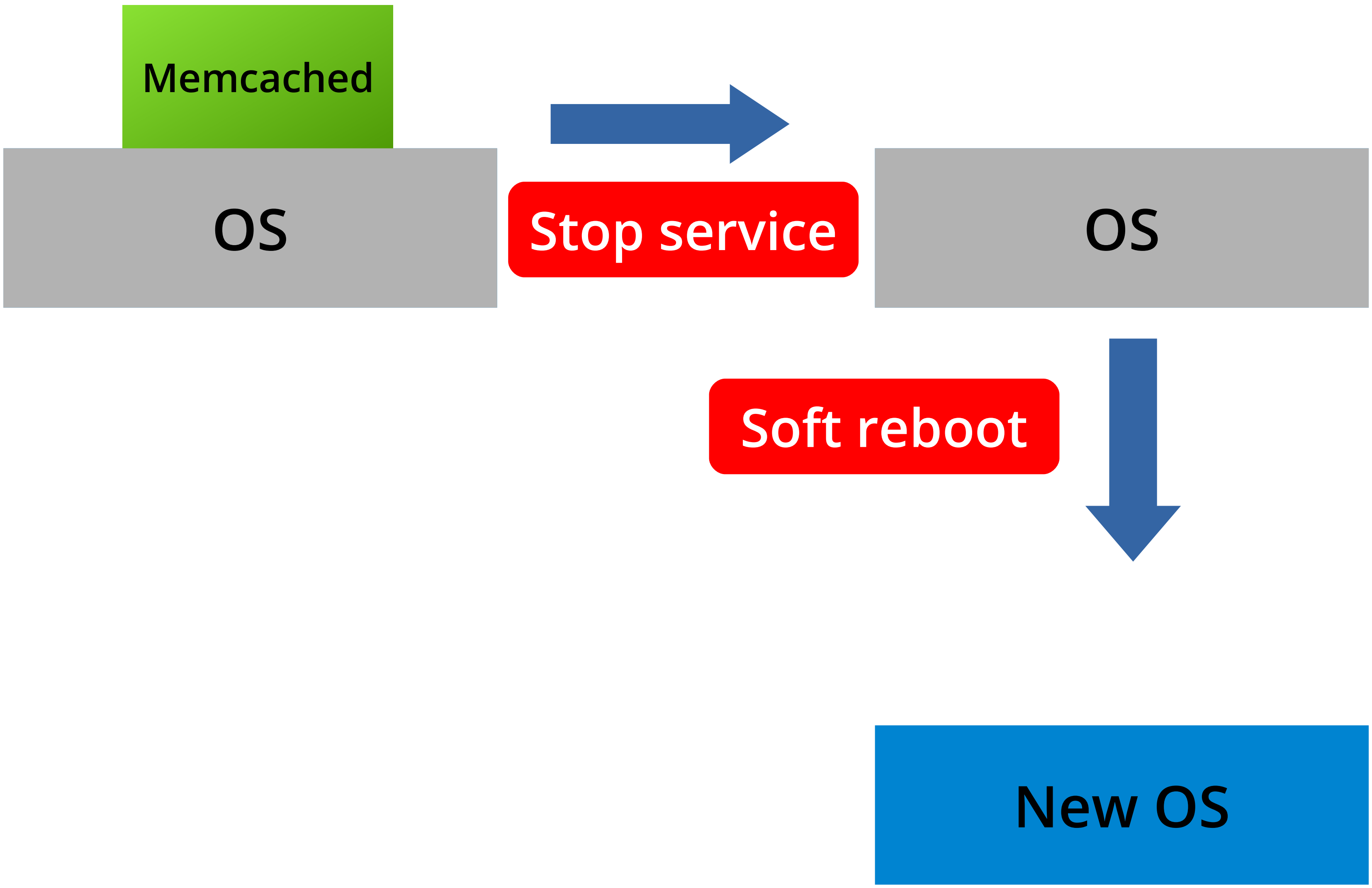
→ Restoring memcached takes 2-3 hours

Goals of this work:

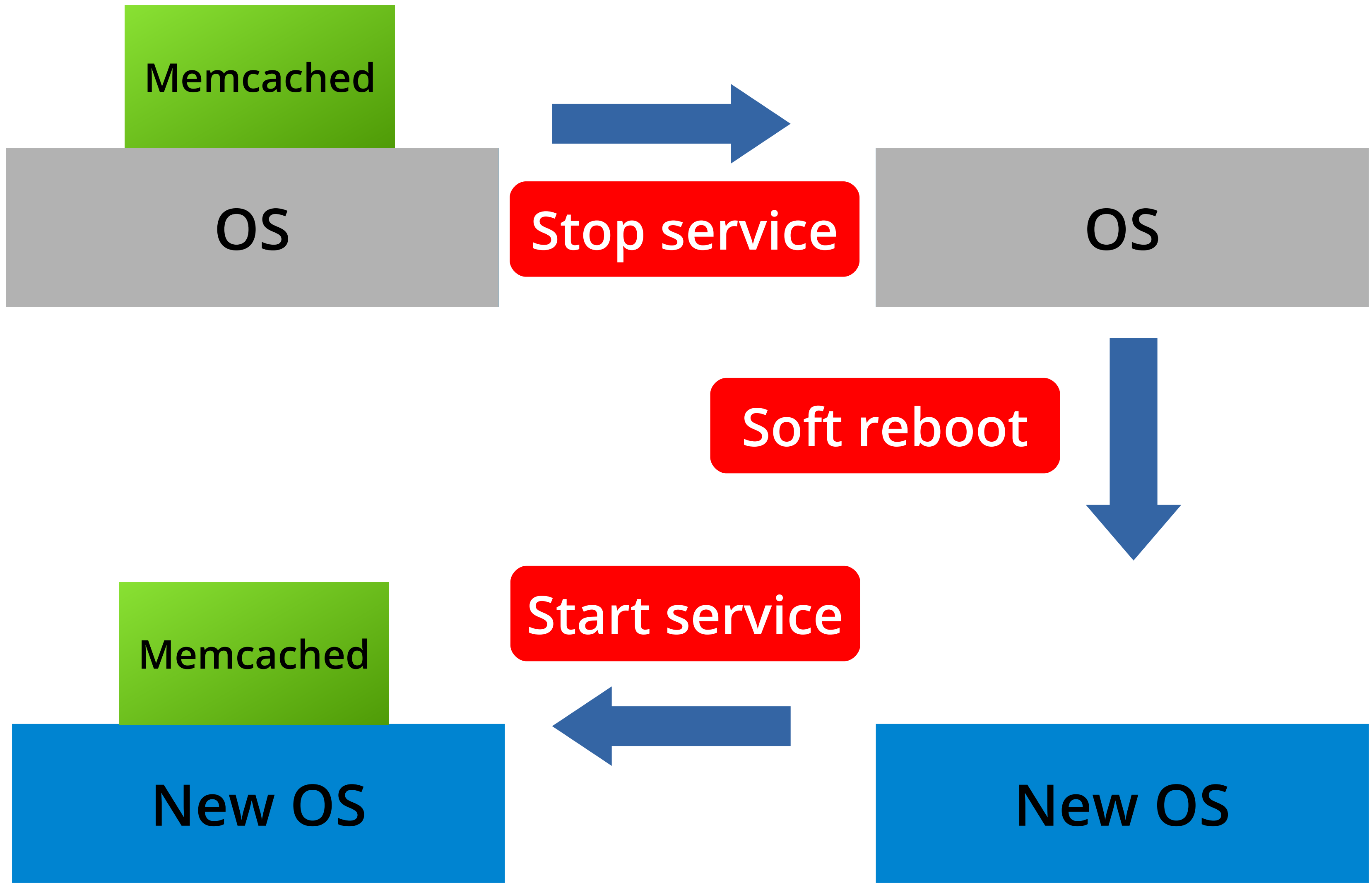
- **Support all types of patches**
- **Least downtime to update new OS**
- **No kernel source modification**

Problems of typical OS update

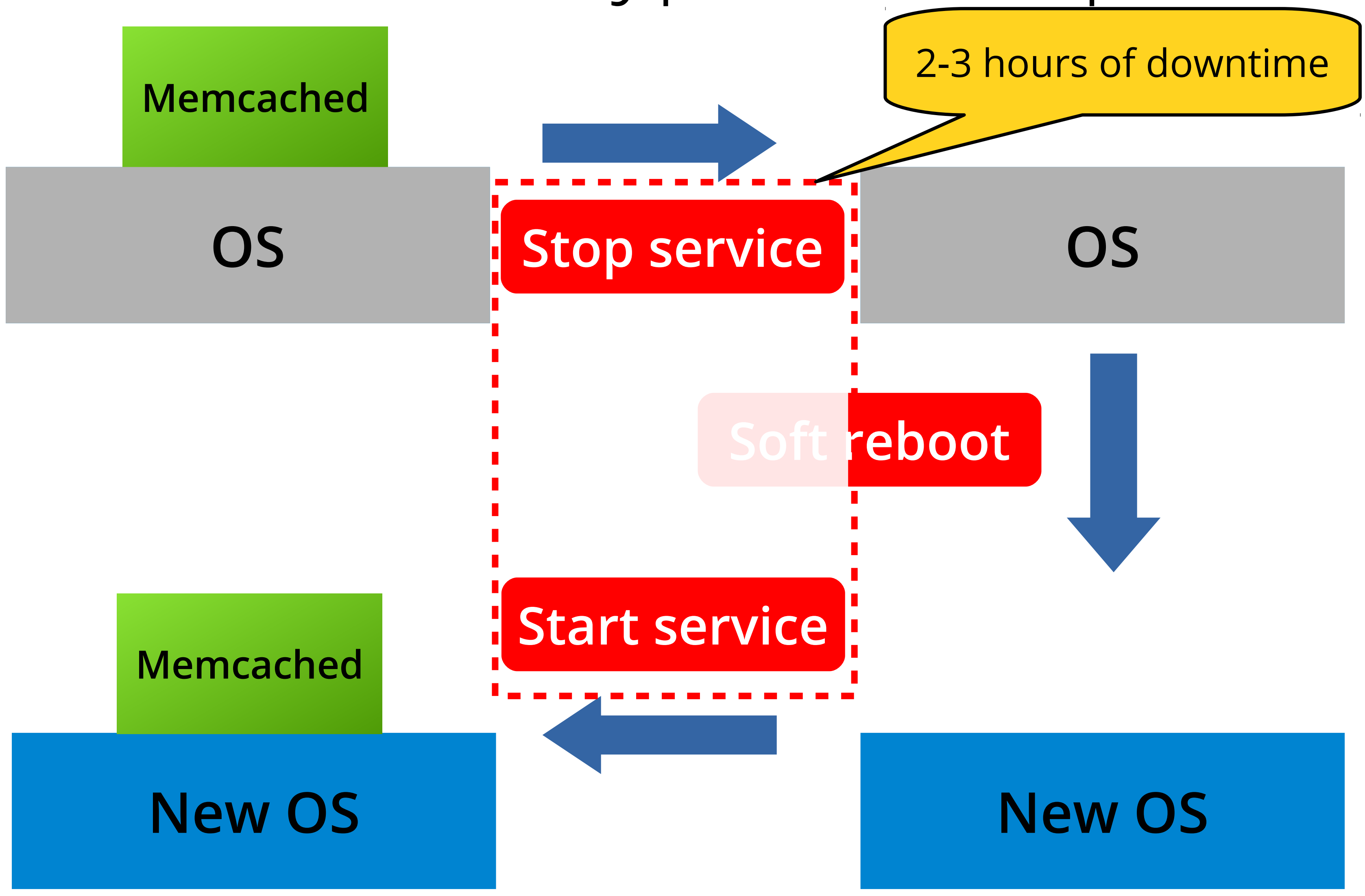
Problems of typical OS update



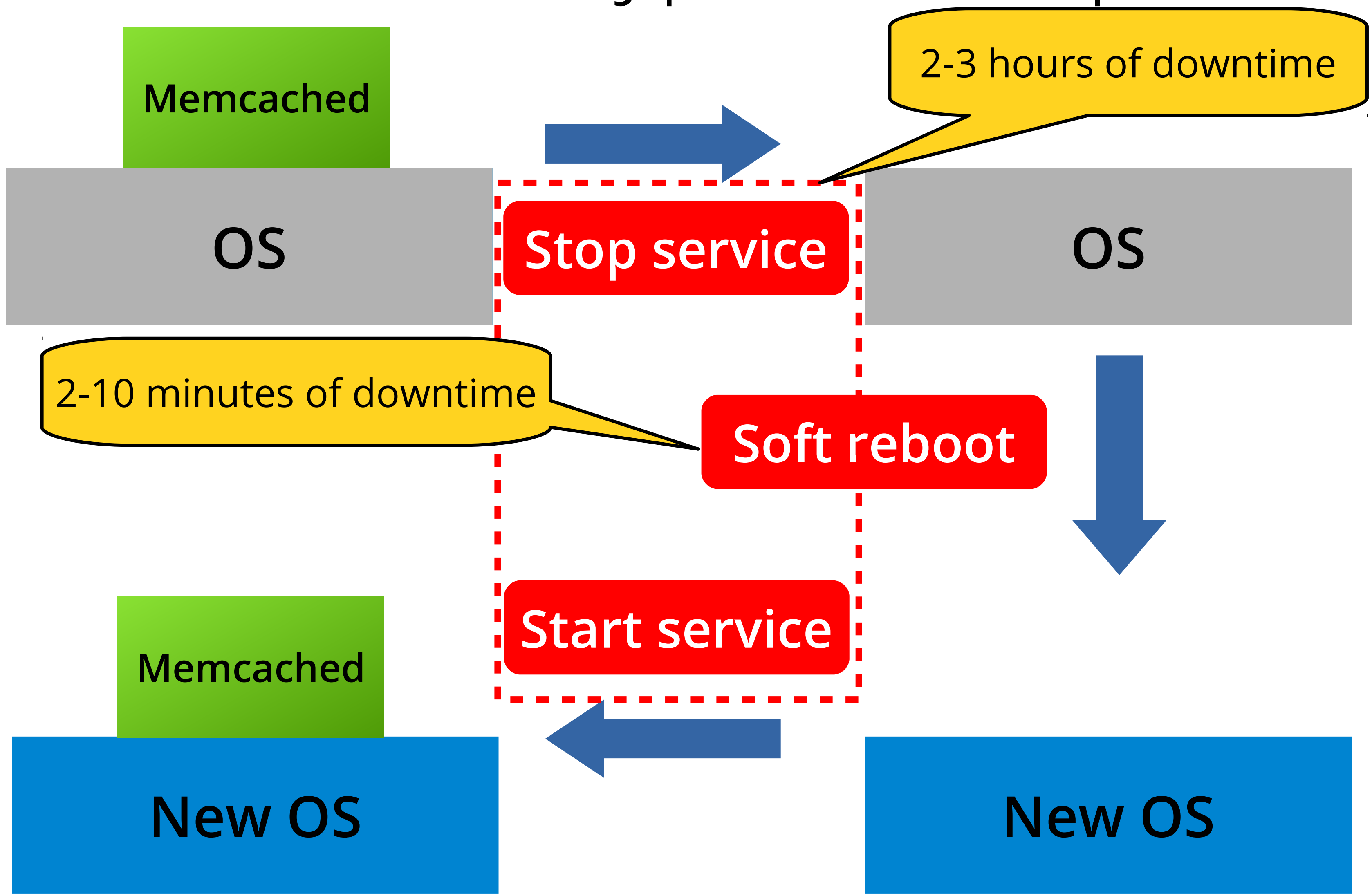
Problems of typical OS update



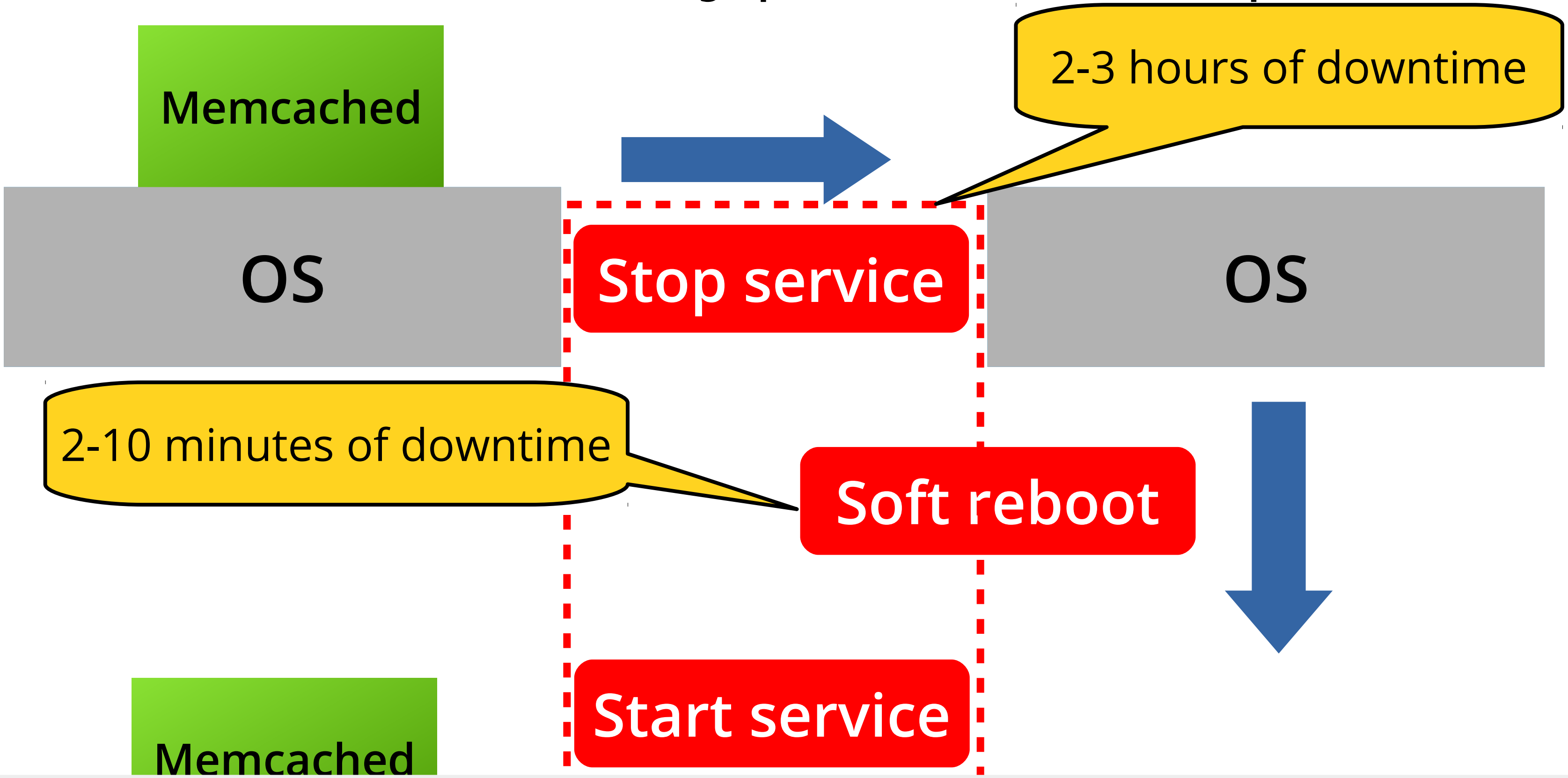
Problems of typical OS update



Problems of typical OS update

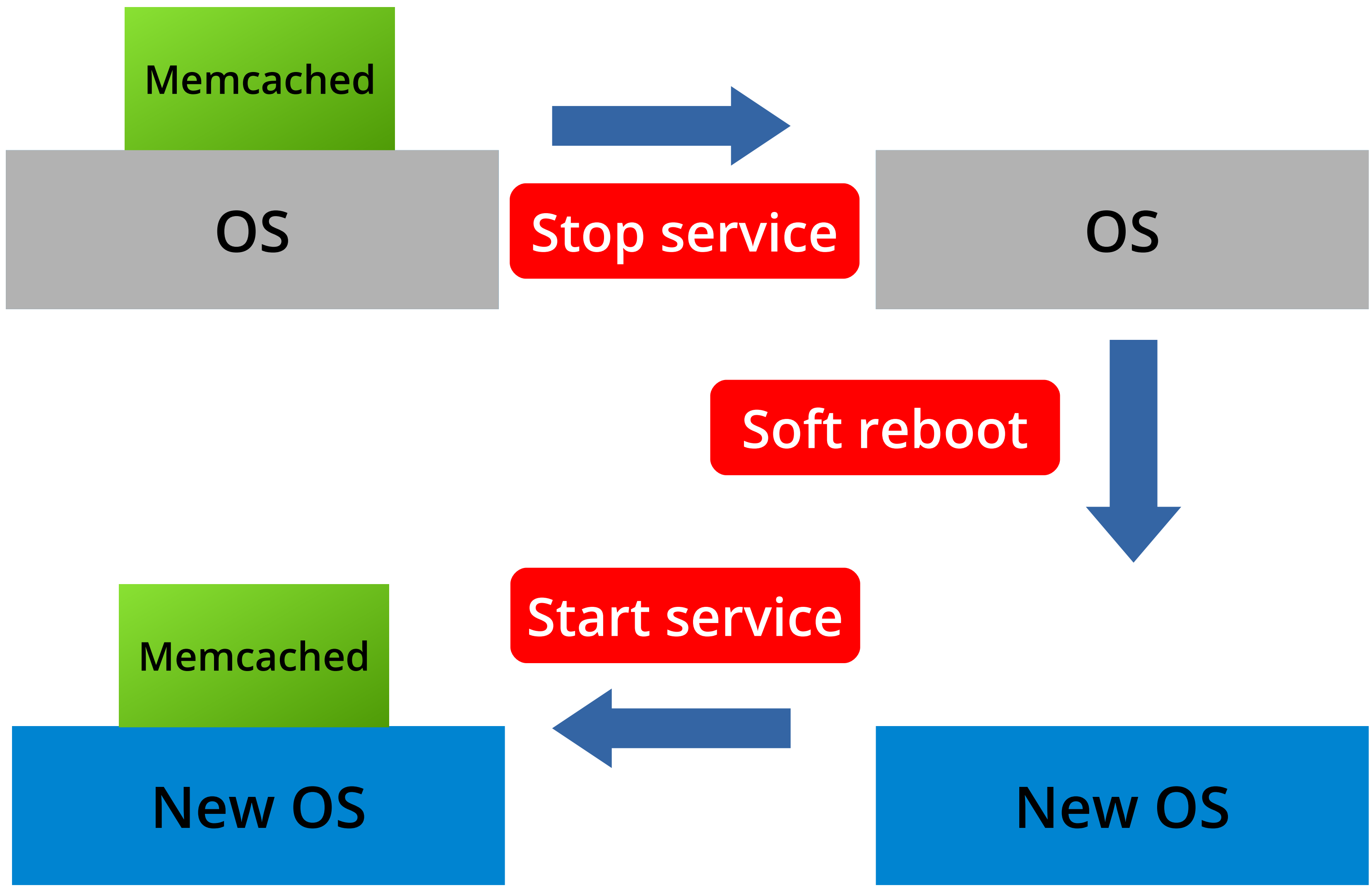


Problems of typical OS update

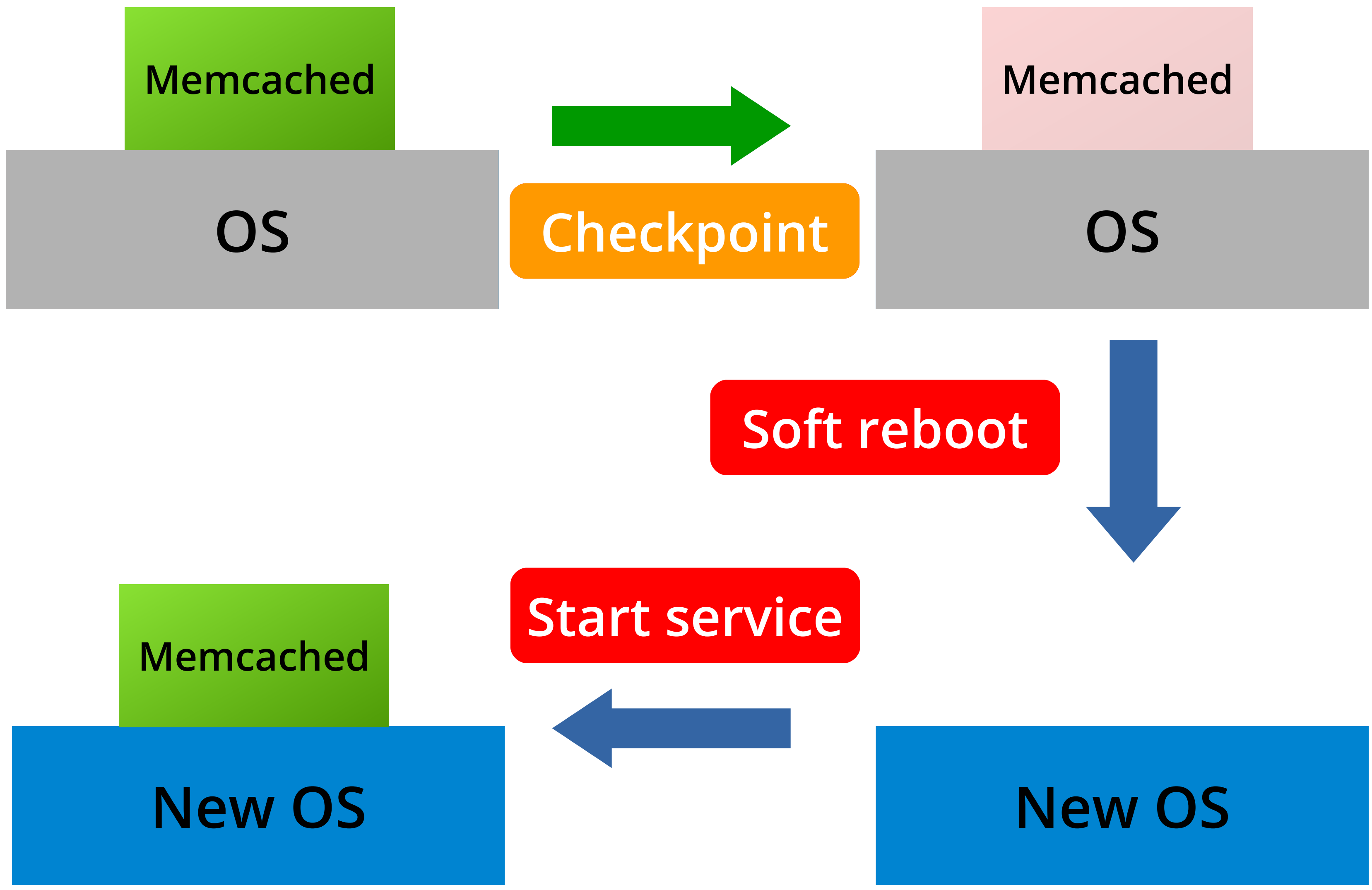


Is it possible to keep the application state?

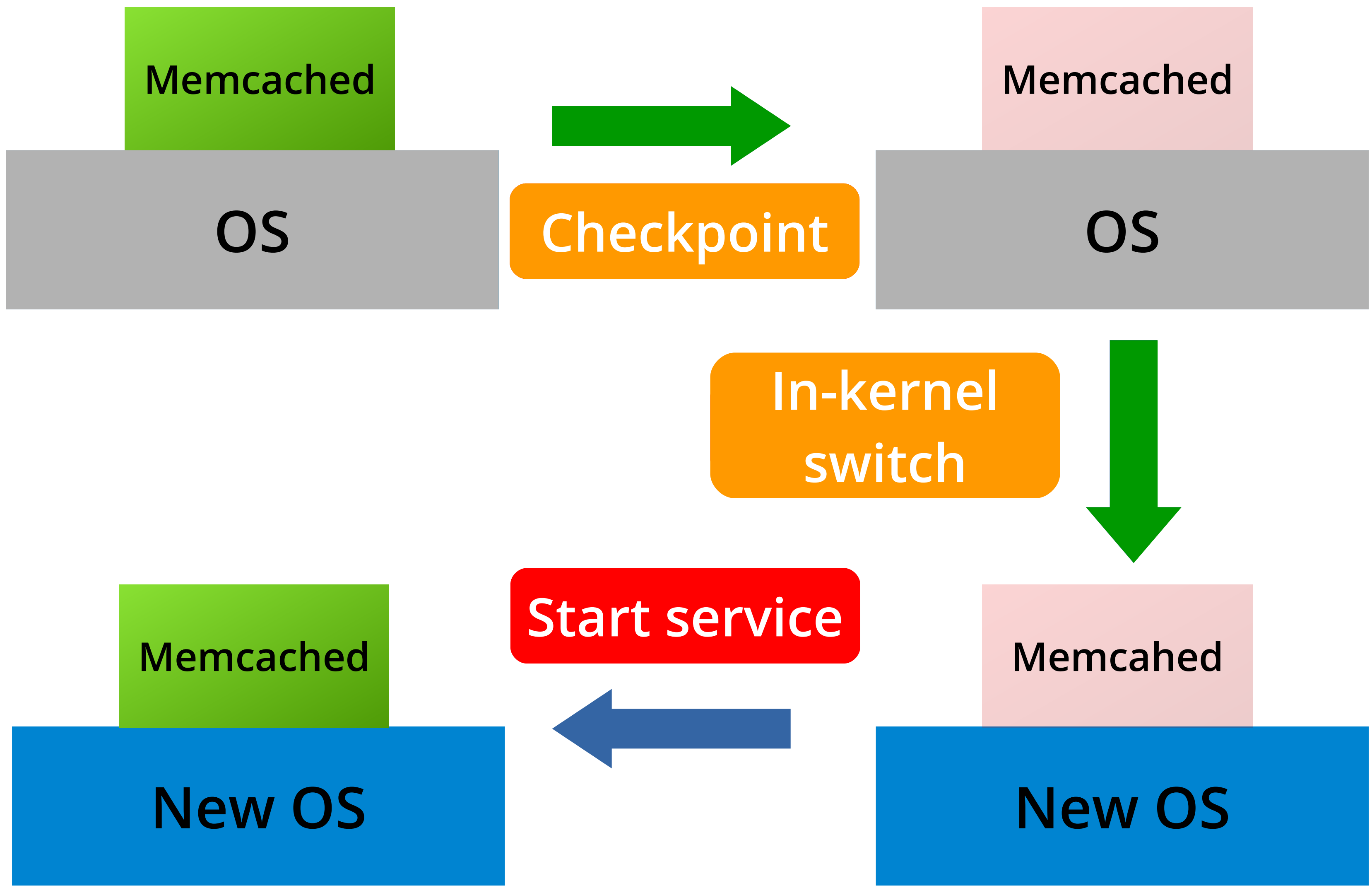
KUP: Kernel update with application checkpoint-and-restore (C/R)



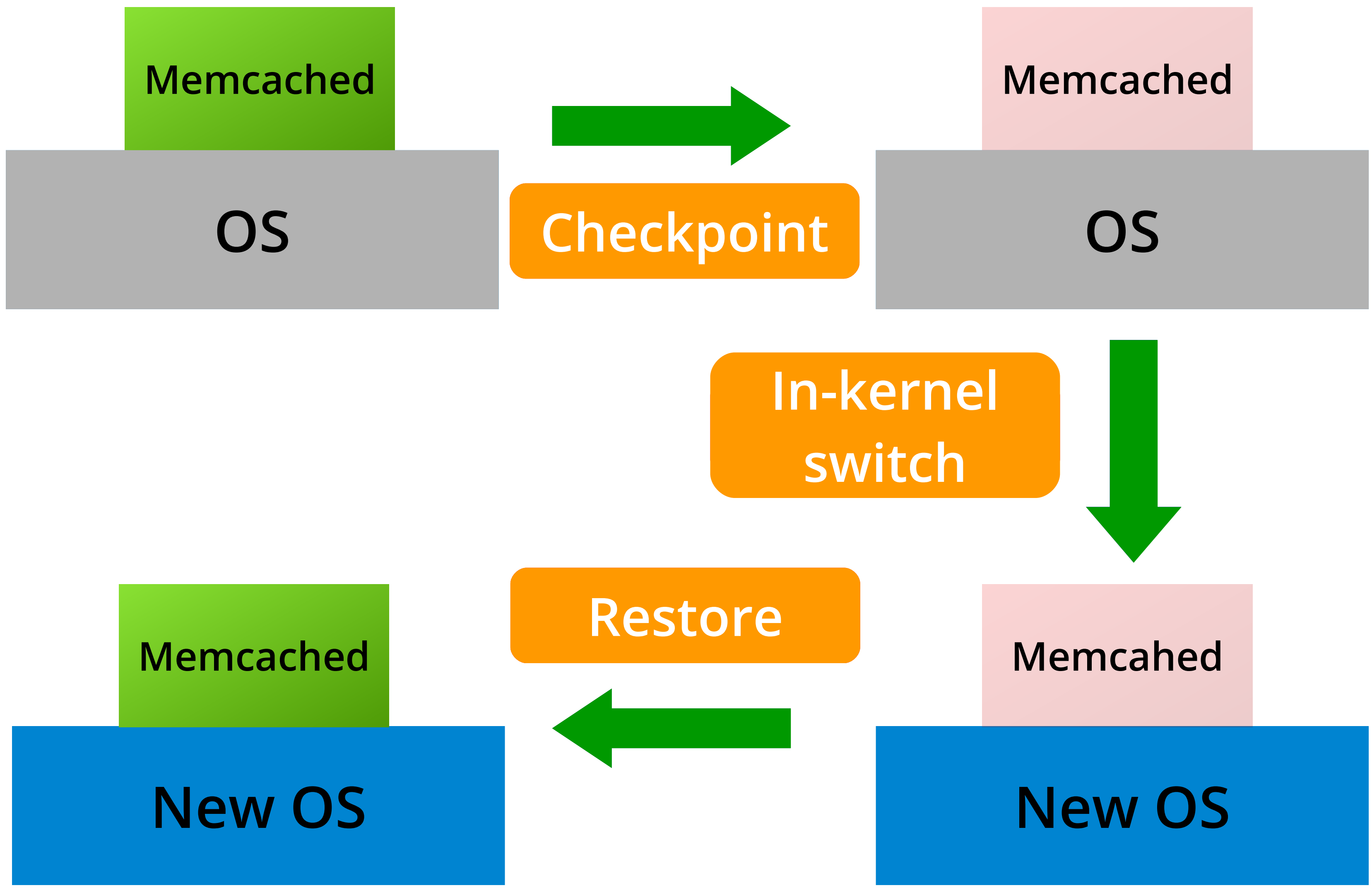
KUP: Kernel update with application checkpoint-and-restore (C/R)



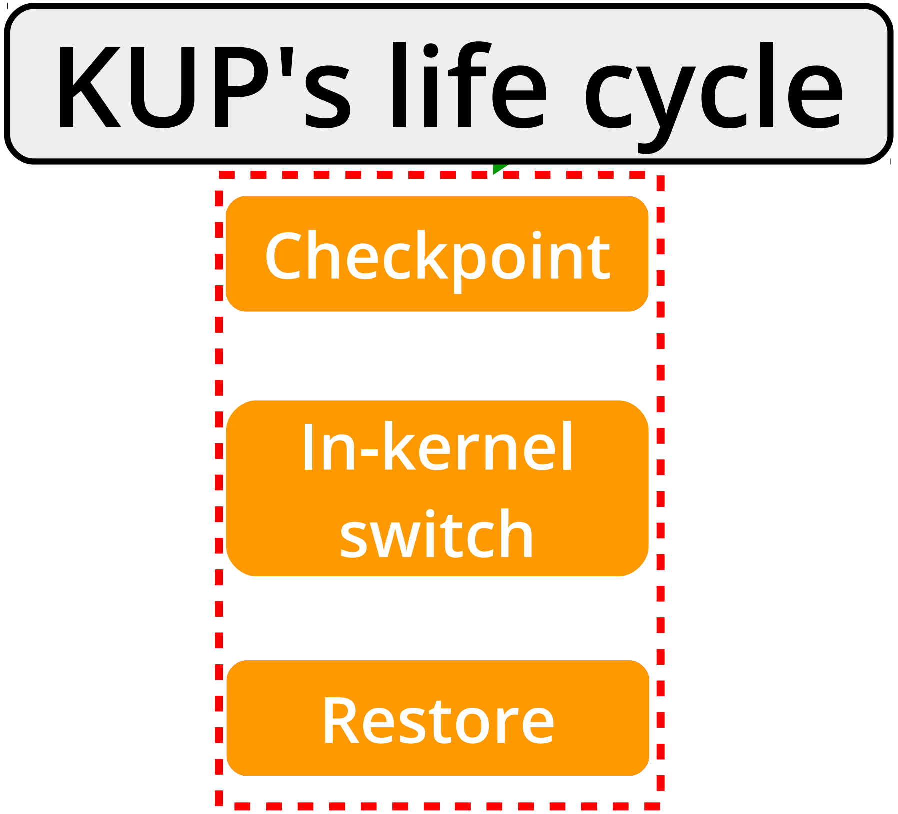
KUP: Kernel update with application checkpoint-and-restore (C/R)



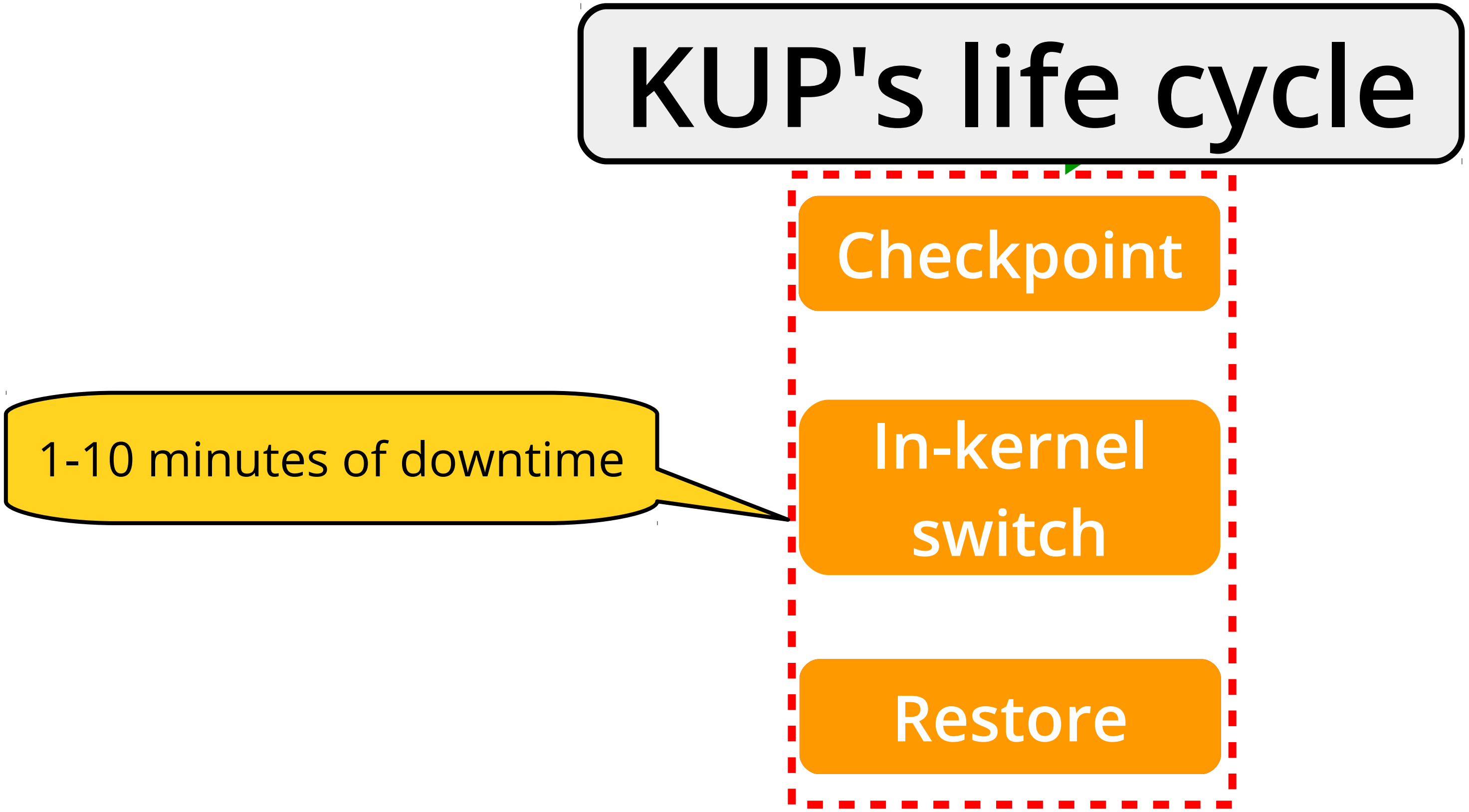
KUP: Kernel update with application checkpoint-and-restore (C/R)



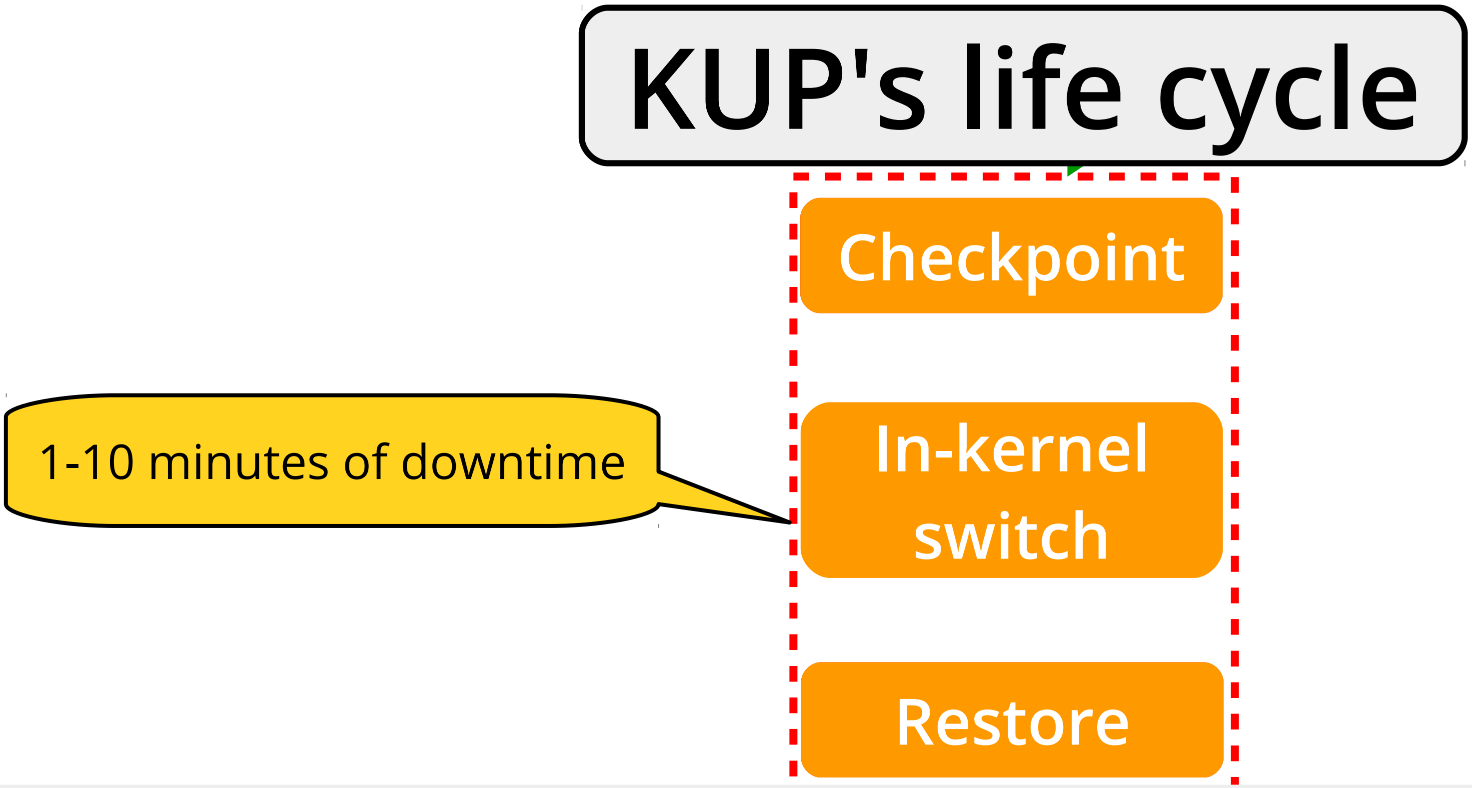
KUP: Kernel update with application checkpoint-and-restore (C/R)



KUP: Kernel update with application checkpoint-and-restore (C/R)

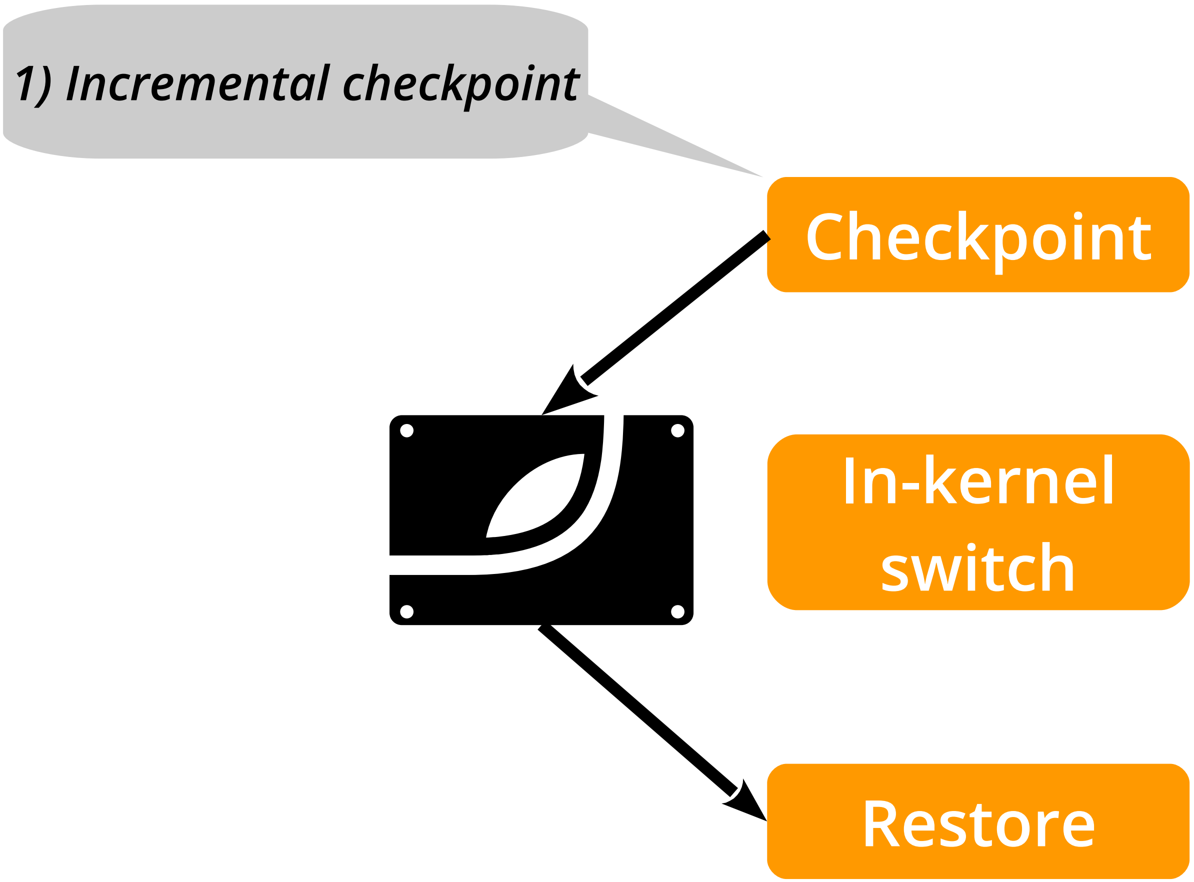


KUP: Kernel update with application checkpoint-and-restore (C/R)

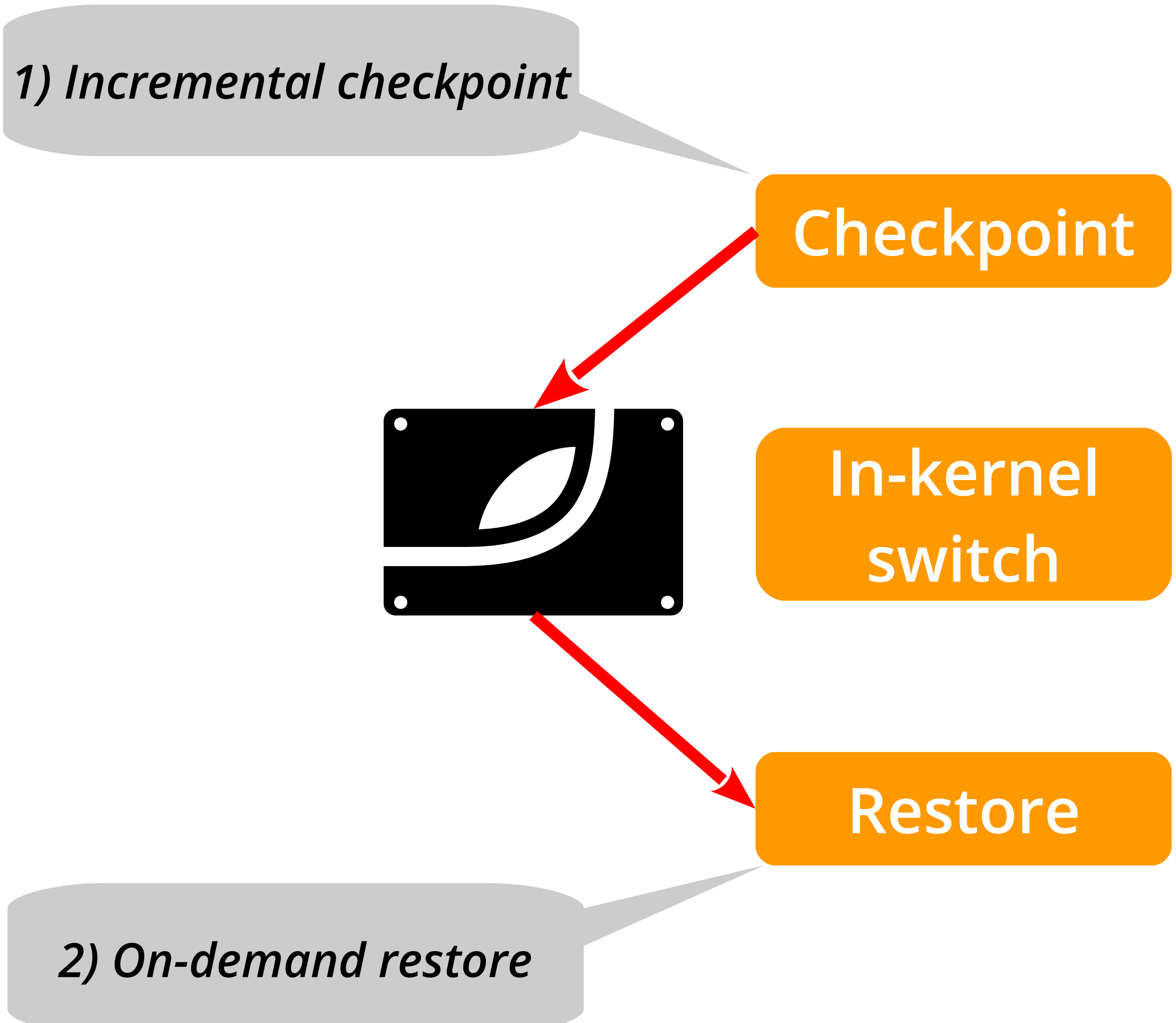


Challenge: how to further decrease the potential downtime?

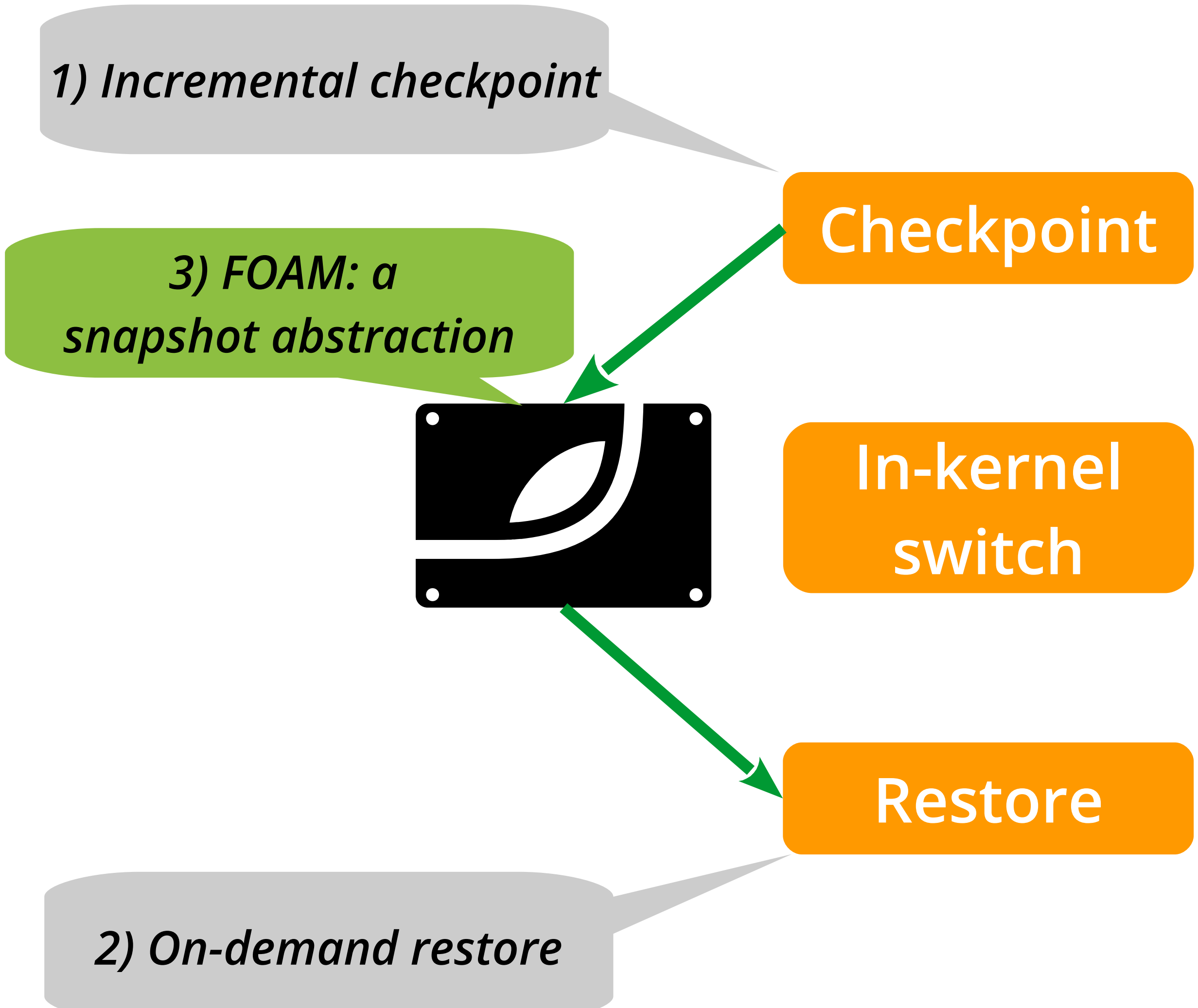
Techniques to decrease the downtime



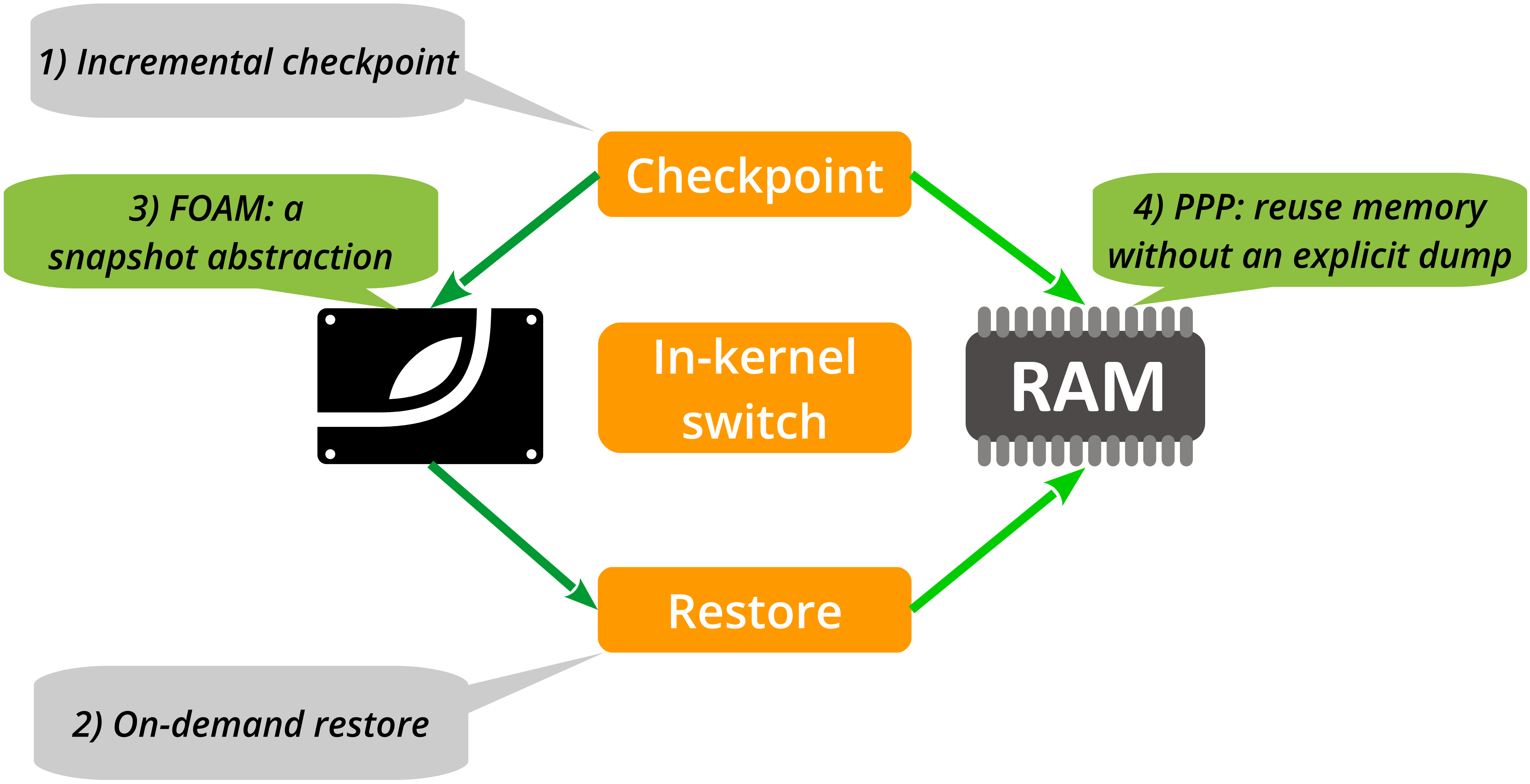
Techniques to decrease the downtime



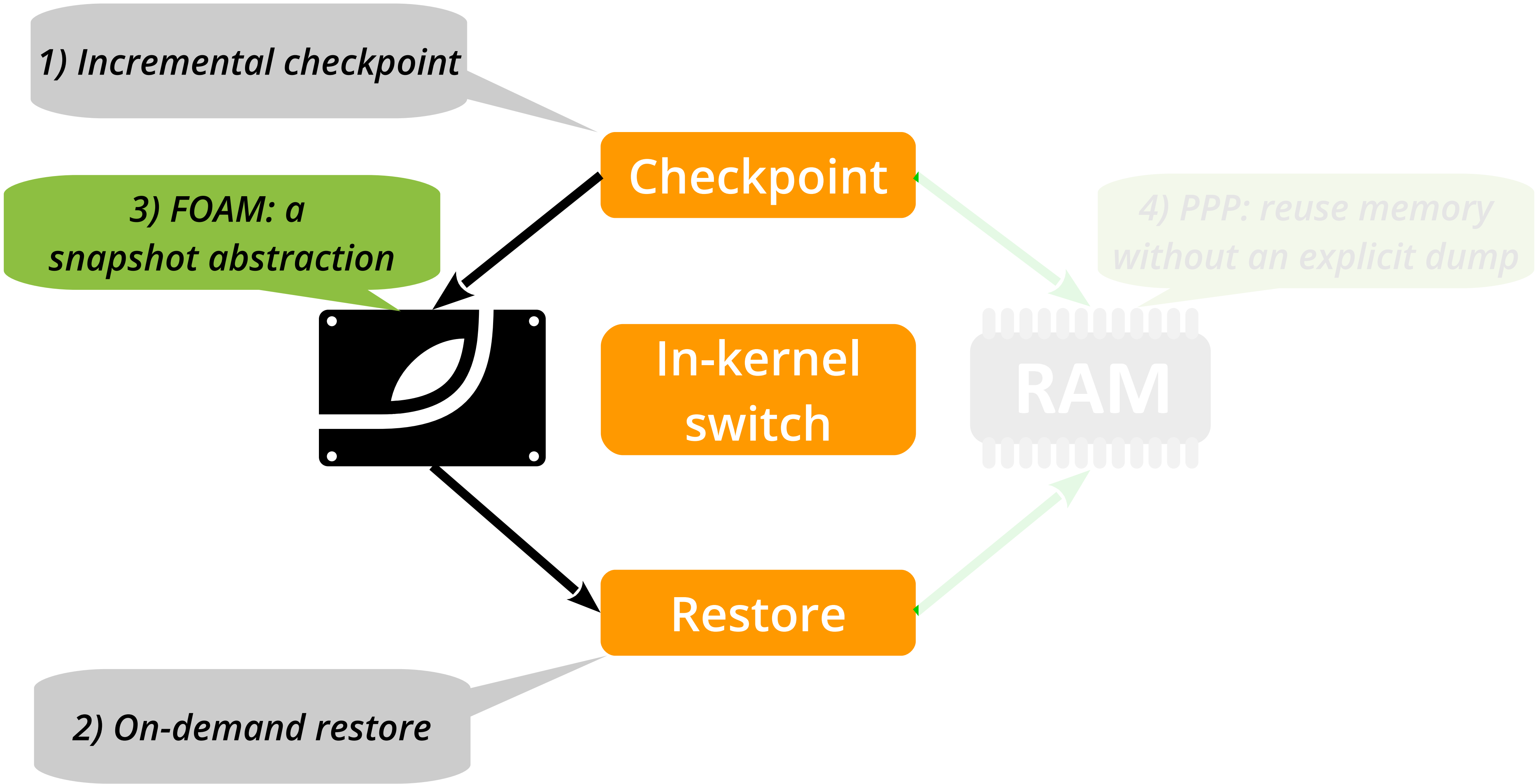
Techniques to decrease the downtime



Techniques to decrease the downtime

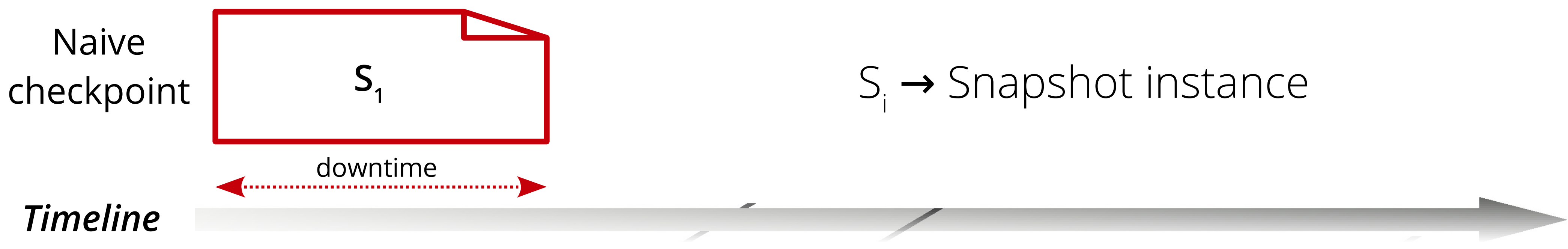


Techniques to decrease the downtime



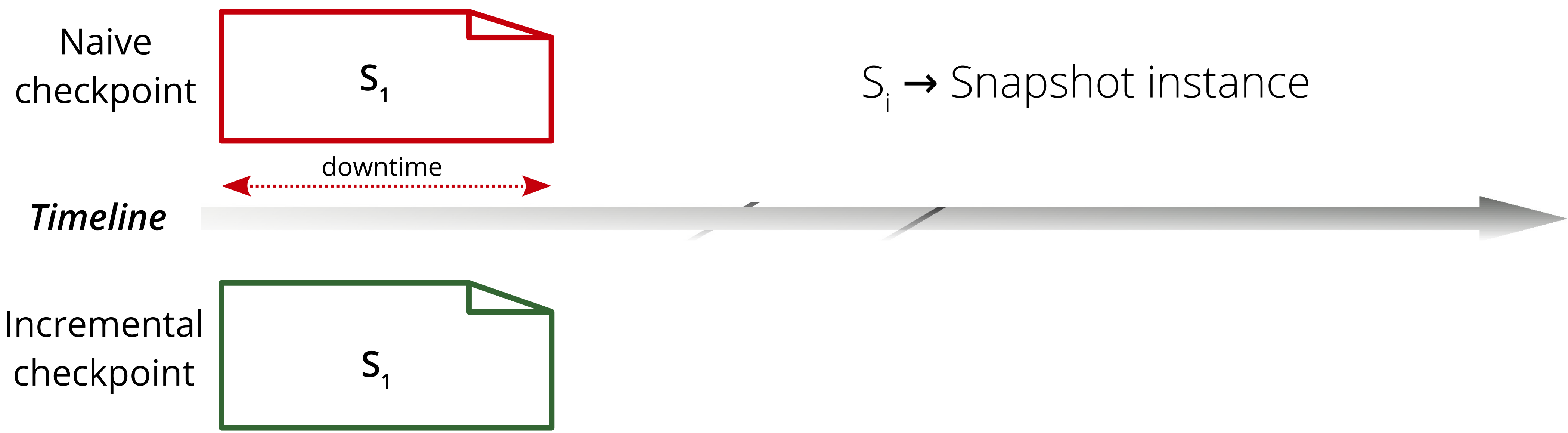
Incremental checkpoint

- Reduces downtime (up to 83.5%)
- **Problem**: Multiple snapshots increase the restore time



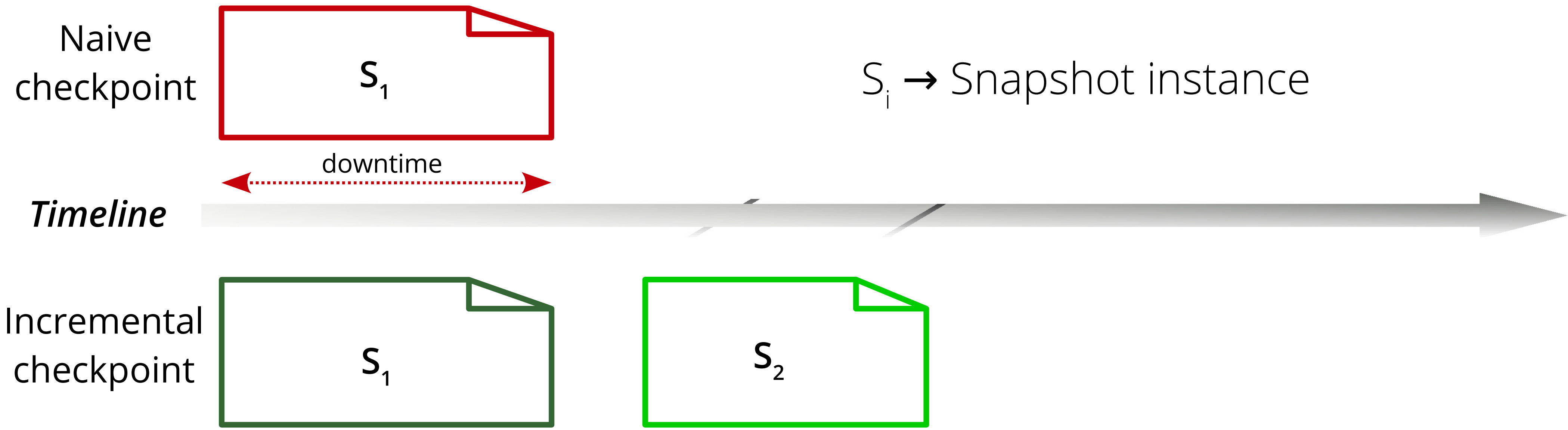
Incremental checkpoint

- Reduces downtime (up to 83.5%)
- **Problem**: Multiple snapshots increase the restore time



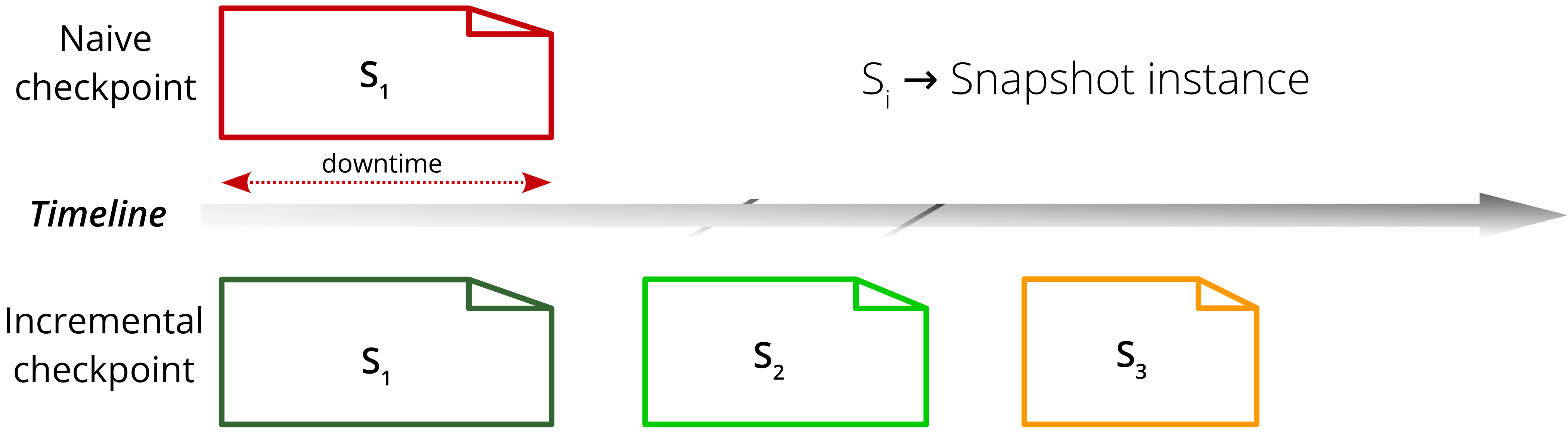
Incremental checkpoint

- Reduces downtime (up to 83.5%)
- **Problem**: Multiple snapshots increase the restore time



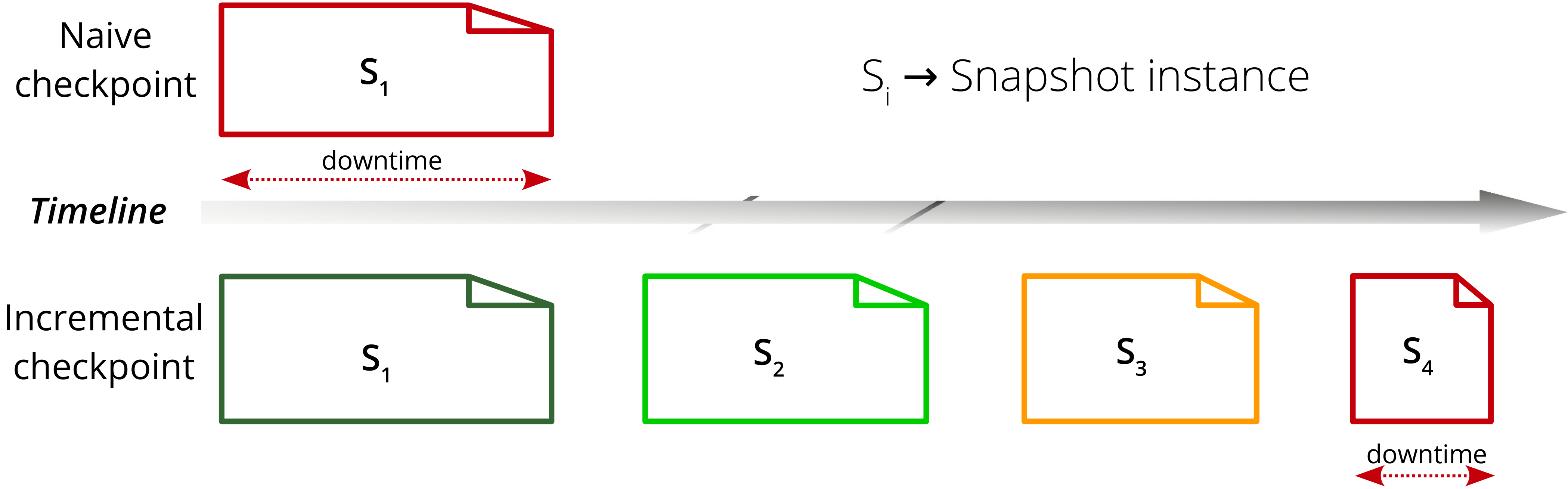
Incremental checkpoint

- Reduces downtime (up to 83.5%)
- **Problem**: Multiple snapshots increase the restore time



Incremental checkpoint

- Reduces downtime (up to 83.5%)
- **Problem**: Multiple snapshots increase the restore time

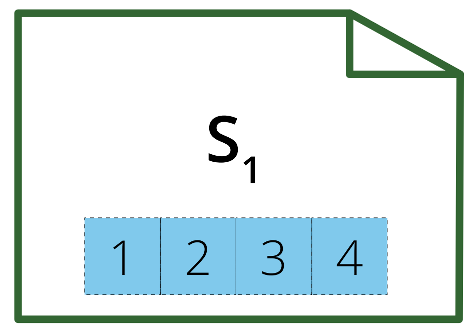


On-demand restore

- Rebind the memory once the application accesses it
  - Only map the memory region with snapshot and restart the application
- Decreases the downtime (up to 99.6%)
- **Problem**: Incompatible with incremental checkpoint

Problem: both techniques together result in inefficient application C/R

- During restore, need to map each pages individually
  - Individual lookups to find the relevant pages
  - Individual page mapping to enable on-demand restore

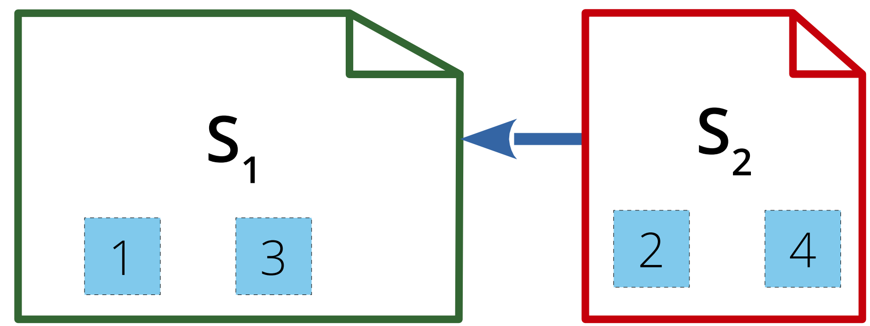


- An application has 4 pages as its working set size
- Incremental checkpoint has 2 iterations
  - 1st iteration → all 4 pages (1, 2, 3, 4) are dumped
  - 2nd iteration → 2 pages (2, 4) are dirtied

- Increases the restoration downtime (42.5%)

Problem: both techniques together result in inefficient application C/R

- During restore, need to map each pages individually
  - Individual lookups to find the relevant pages
  - Individual page mapping to enable on-demand restore



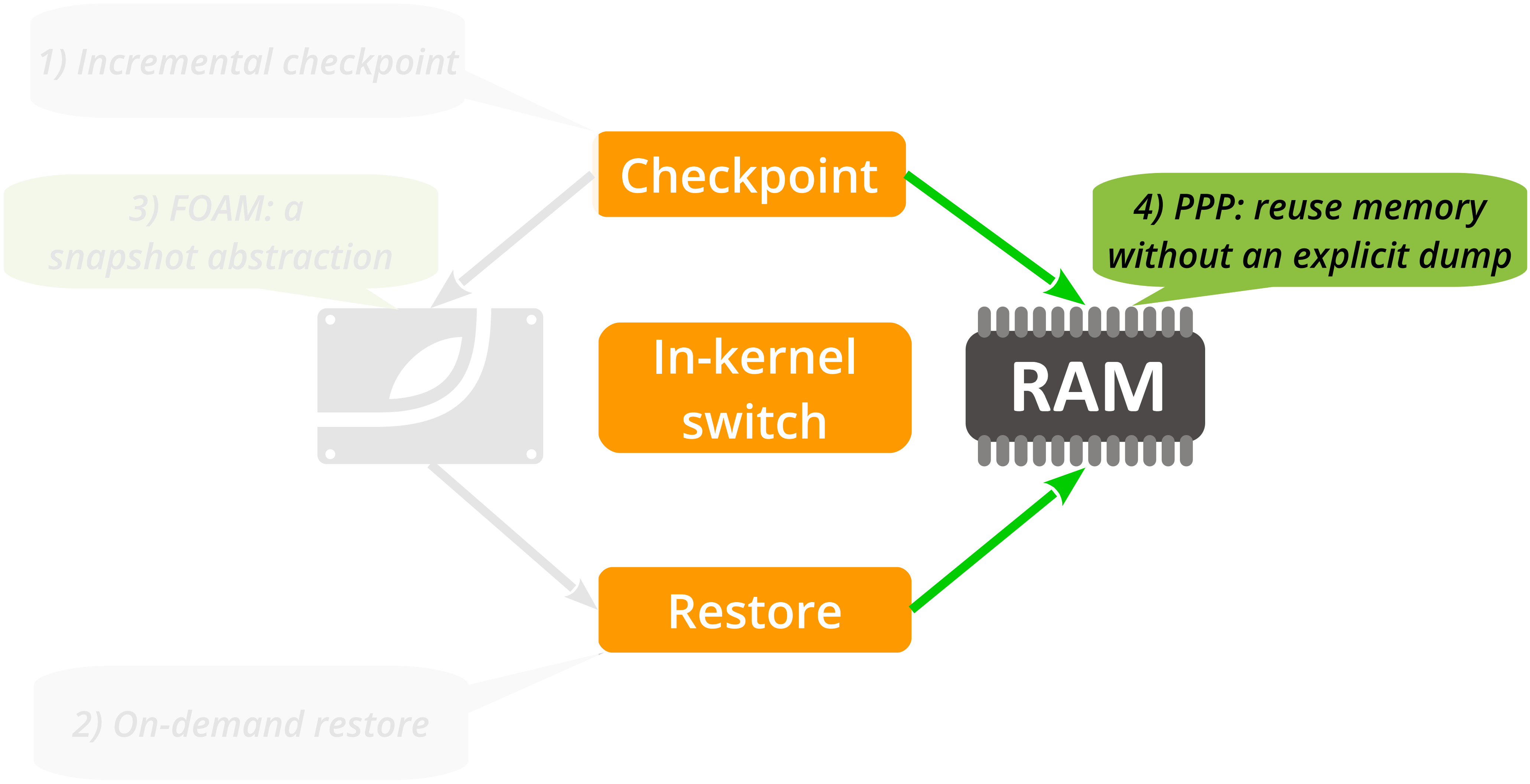
- An application has 4 pages as its working set size
- Incremental checkpoint has 2 iterations
  - 1st iteration → all 4 pages (1, 2, 3, 4) are dumped
  - 2nd iteration → 2 pages (2, 4) are dirtied

- Increases the restoration downtime (42.5%)

New abstraction: file-offset based address mapping (FOAM)

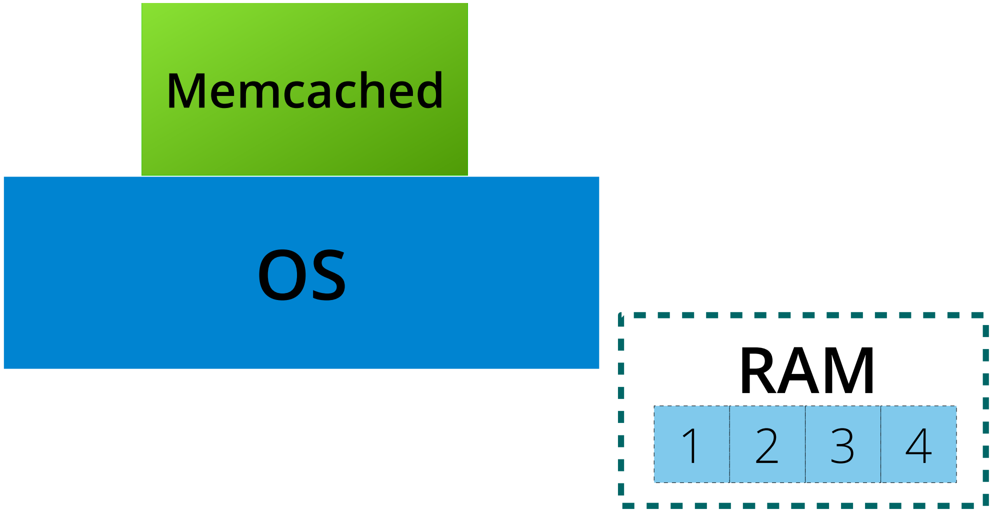
- Flat address space representation for the snapshot
  - One-to-one mapping between the address space and the snapshot
  - No explicit lookups for the pages across the snapshots
  - A few map operations to map the entire snapshot with address space
- Use sparse file representation
  - Rely on the concept of holes supported by modern file systems
- Simplifies incremental checkpoint and on-demand restore

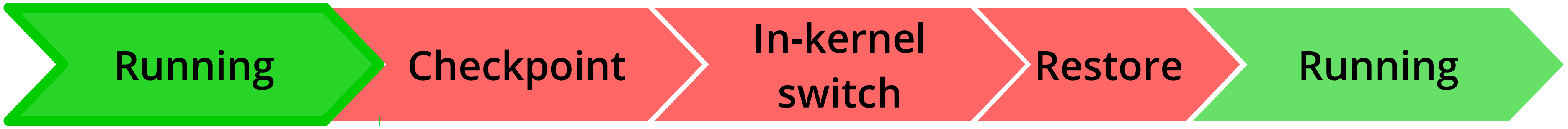
Techniques to decrease the downtime



Redundant data copy

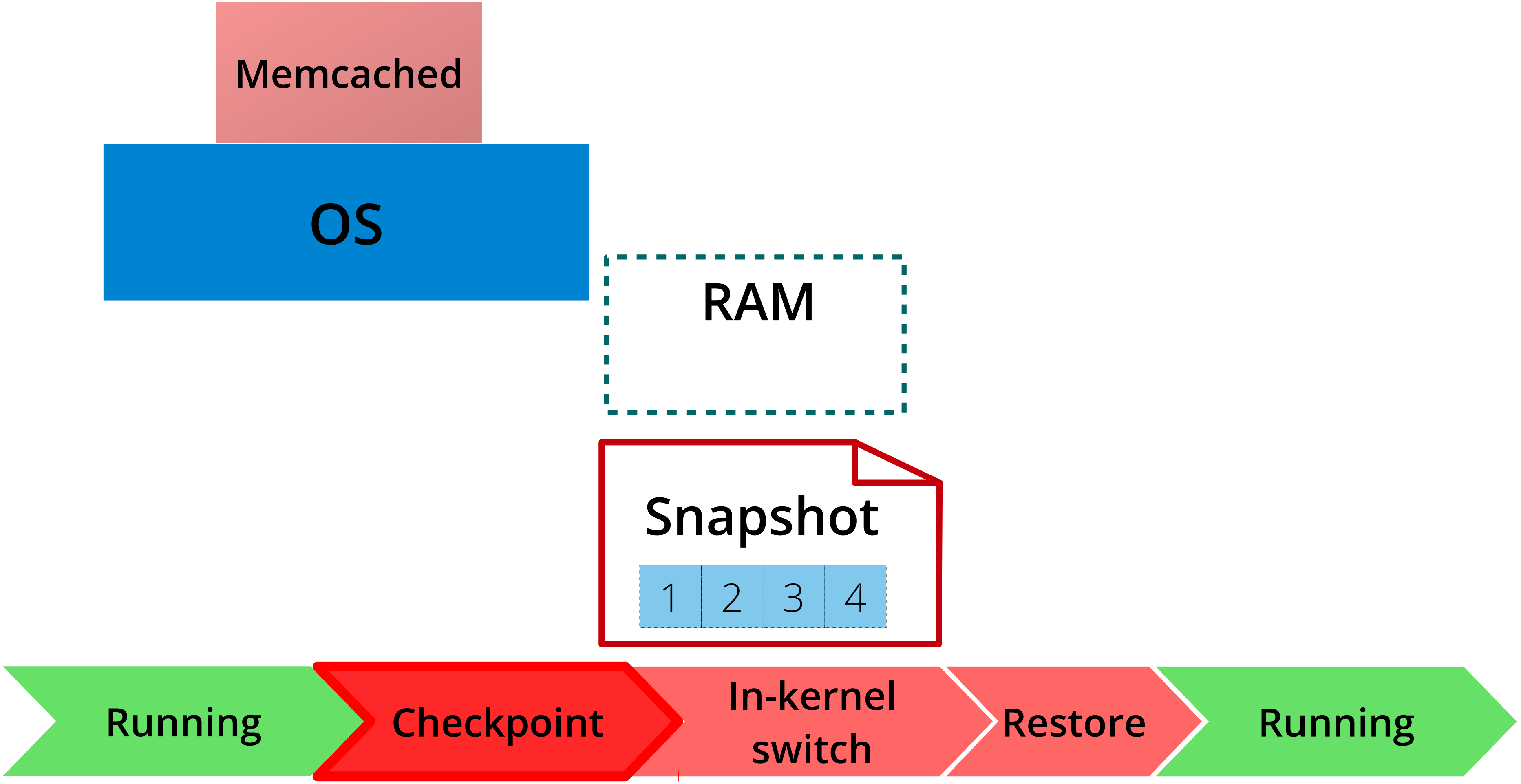
- Application C/R copies data back and forth
- Not a good fit for applications with huge memory





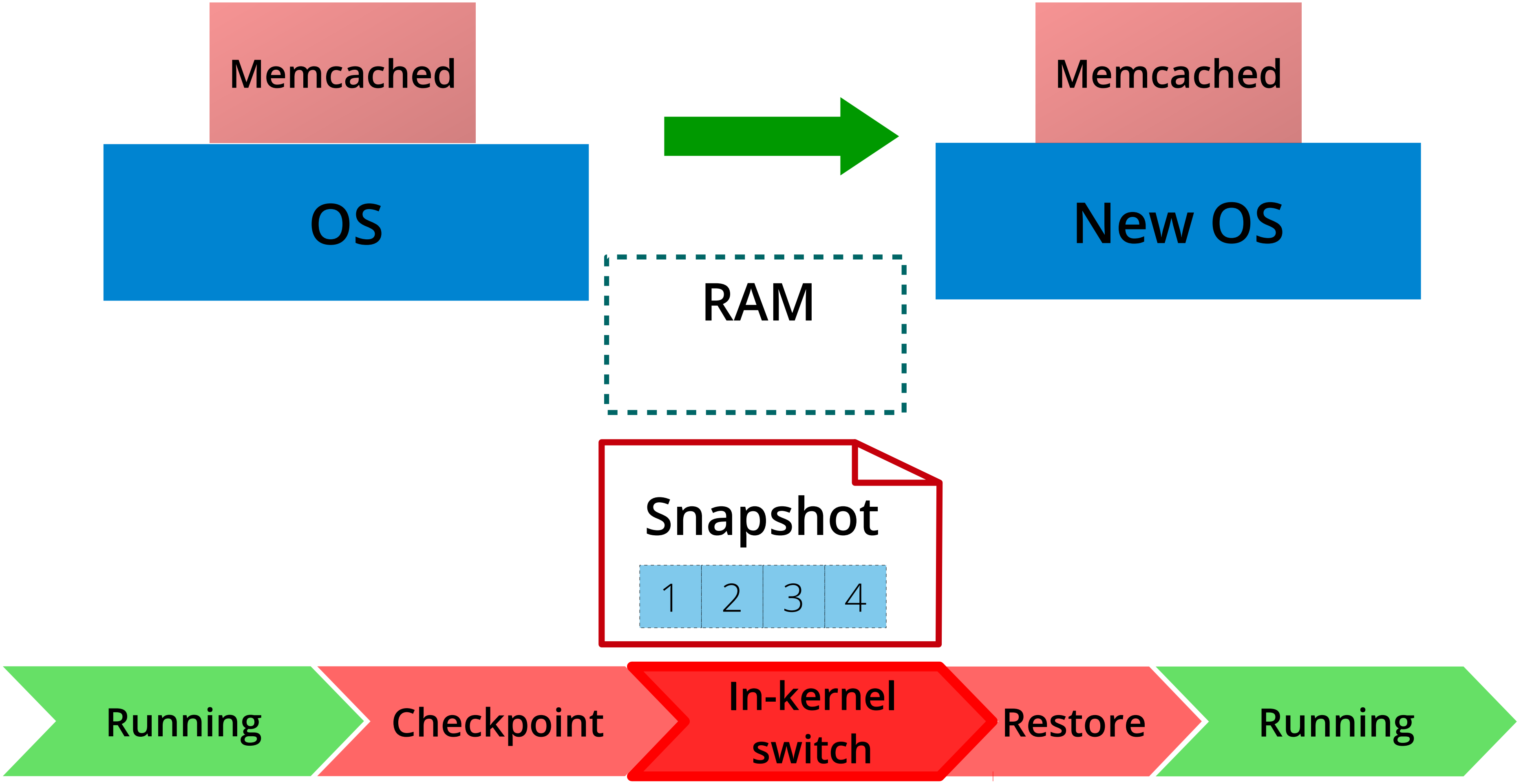
Redundant data copy

- Application C/R copies data back and forth
- Not a good fit for applications with huge memory



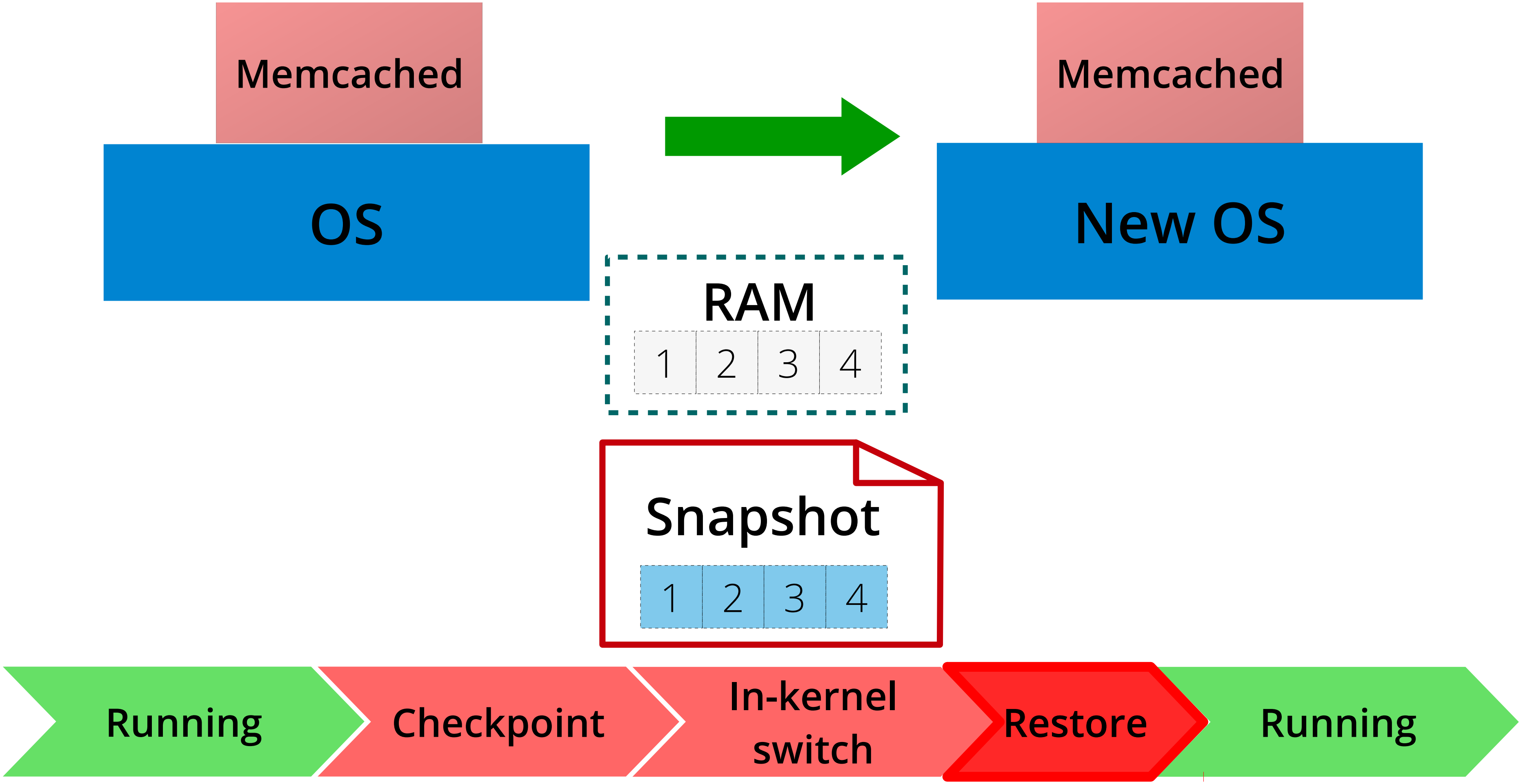
Redundant data copy

- Application C/R copies data back and forth
- Not a good fit for applications with huge memory



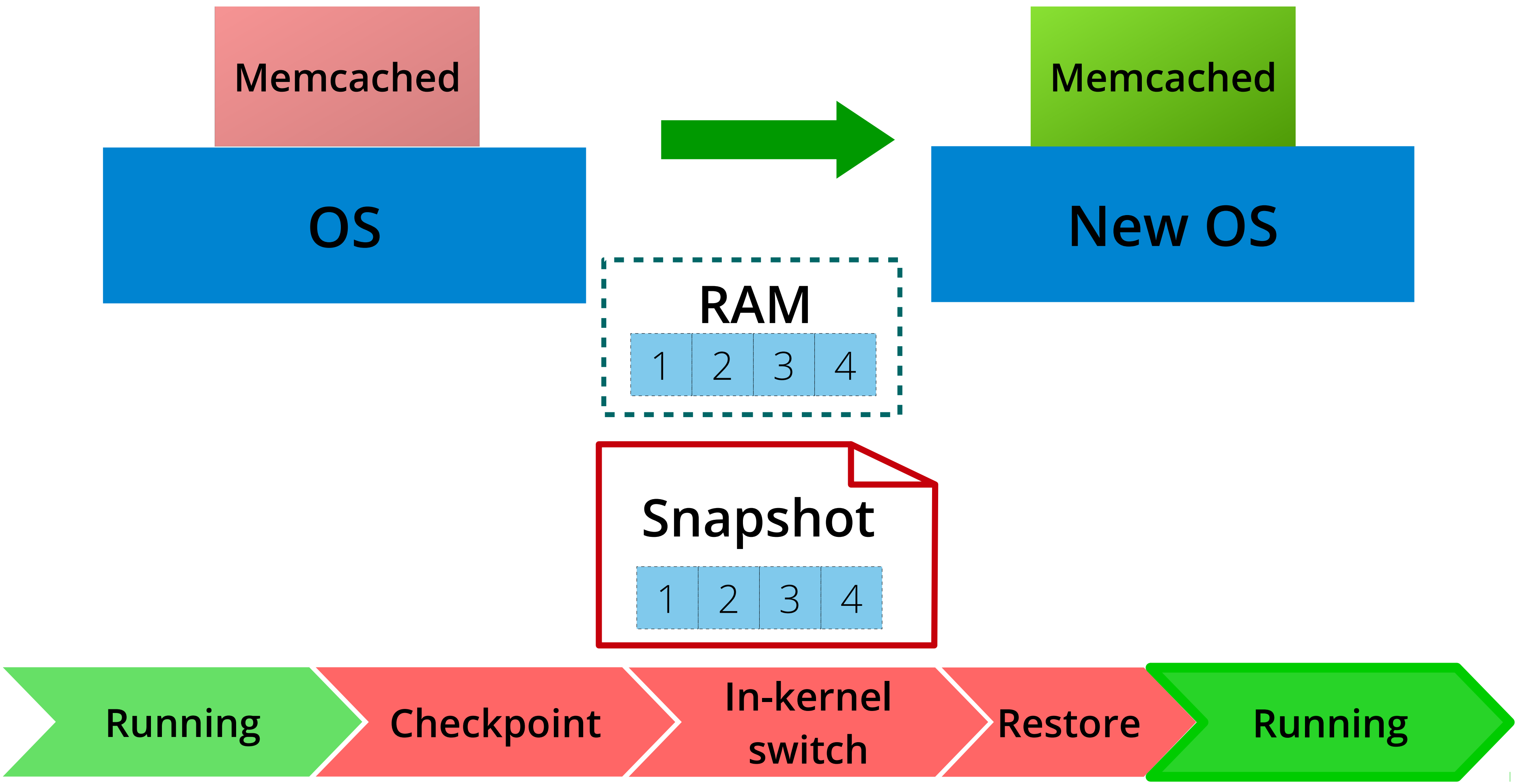
Redundant data copy

- Application C/R copies data back and forth
- Not a good fit for applications with huge memory



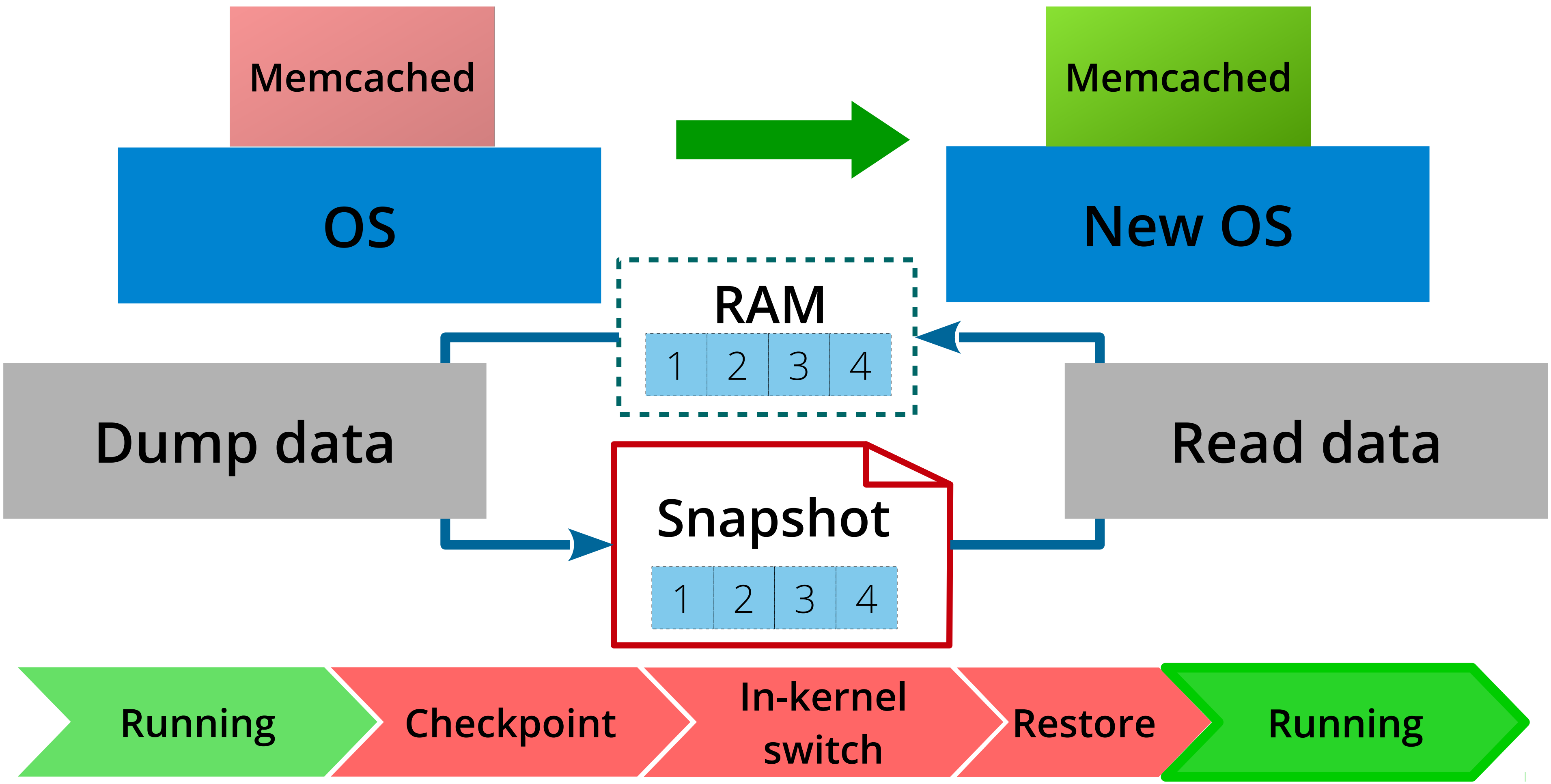
Redundant data copy

- Application C/R copies data back and forth
- Not a good fit for applications with huge memory



Redundant data copy

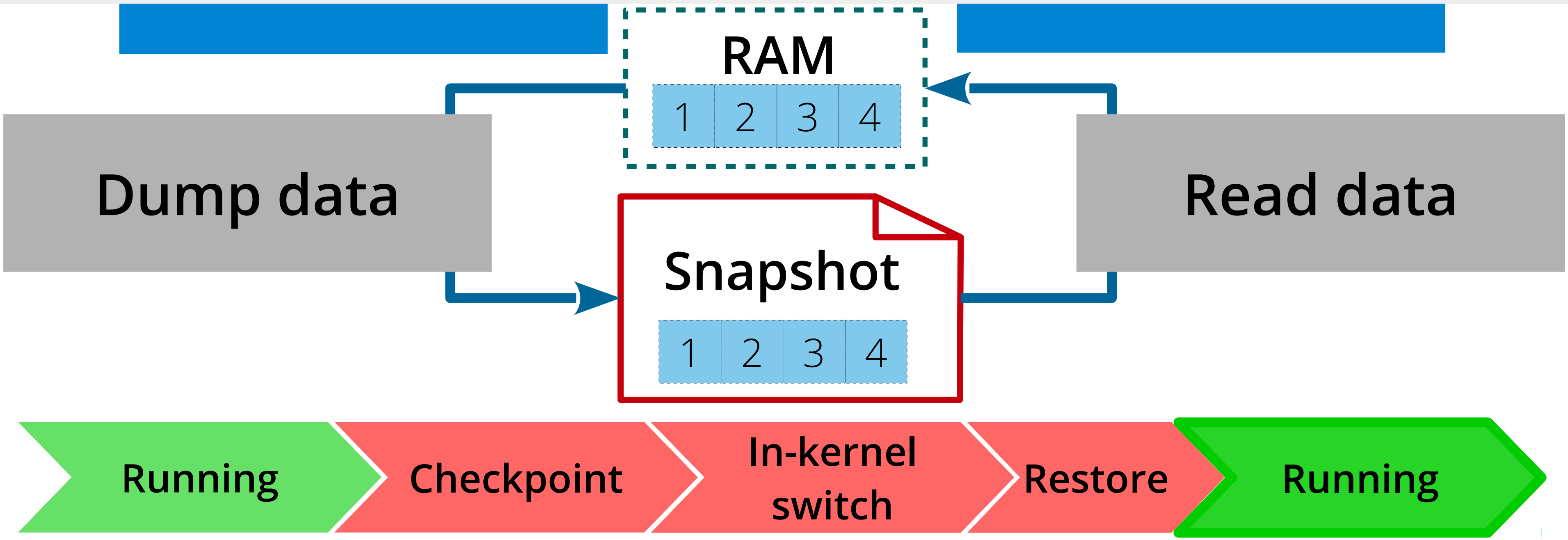
- Application C/R copies data back and forth
- Not a good fit for applications with huge memory



Redundant data copy

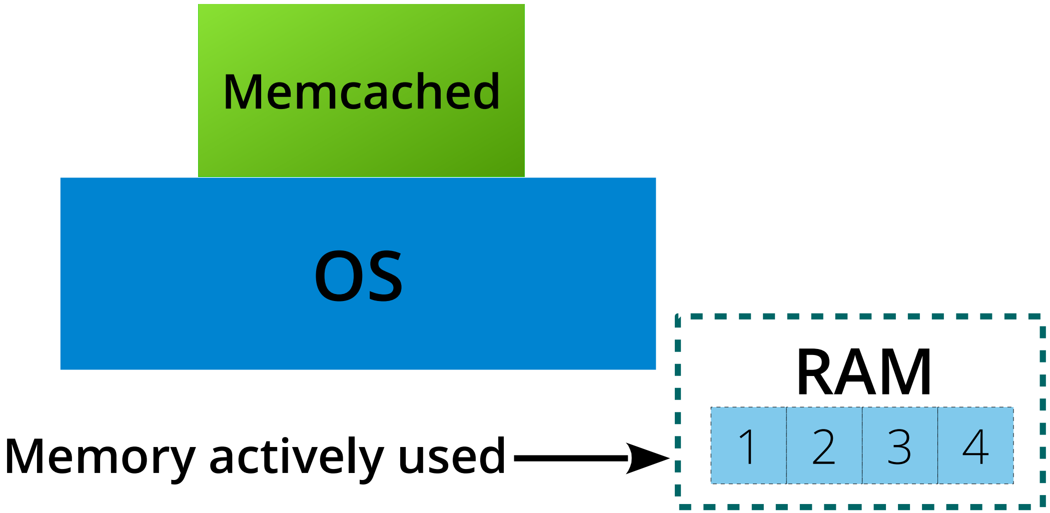
- Application C/R copies data back and forth

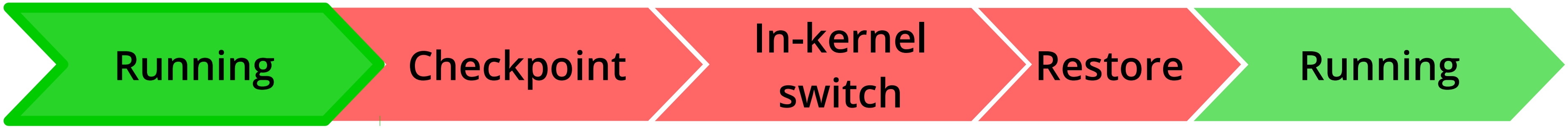
Is it possible to avoid memory copy?



Avoid redundant data copy across reboot

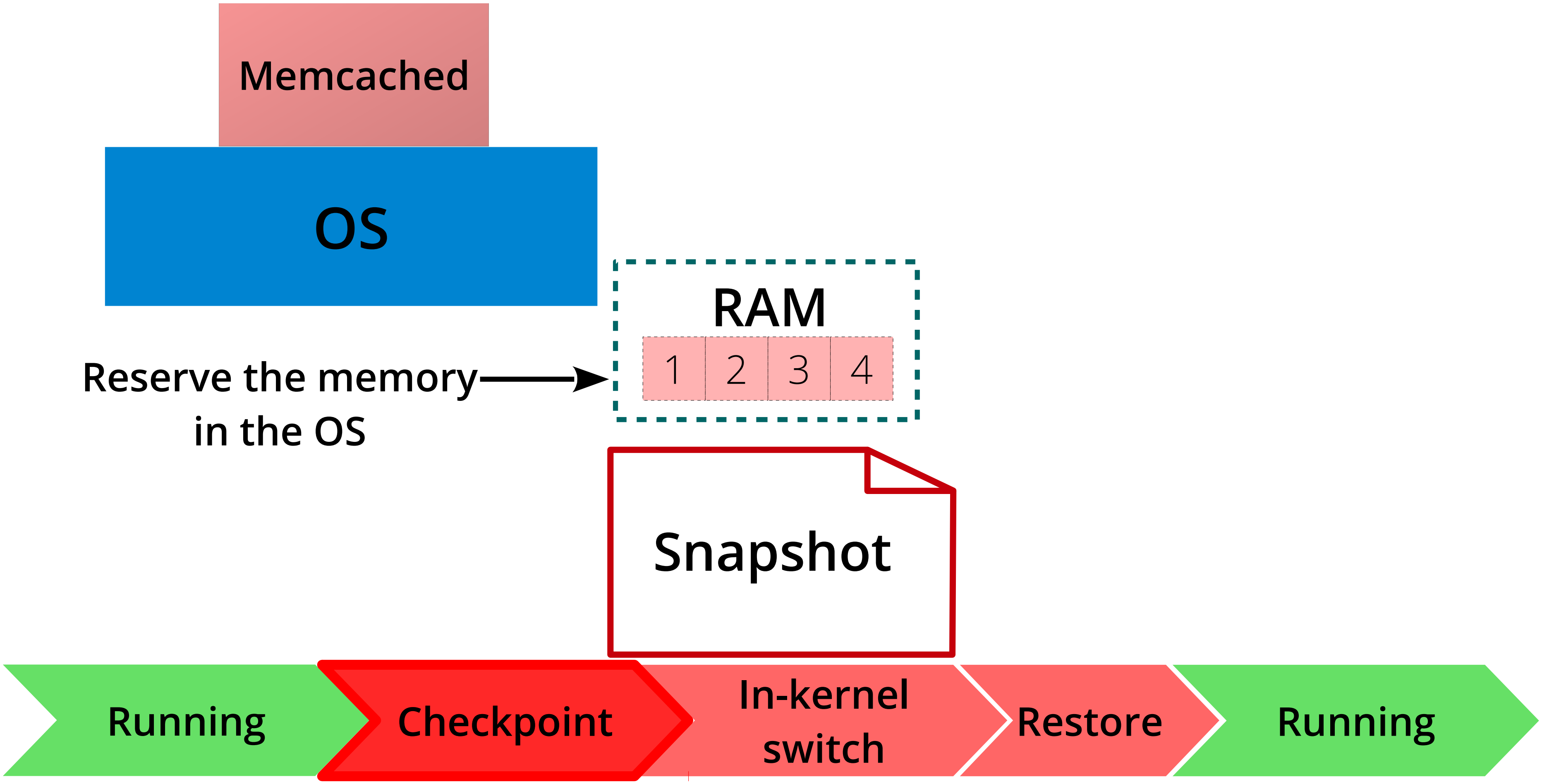
- Reserve the application's memory across reboot
- Inherently rebind the memory without any copy





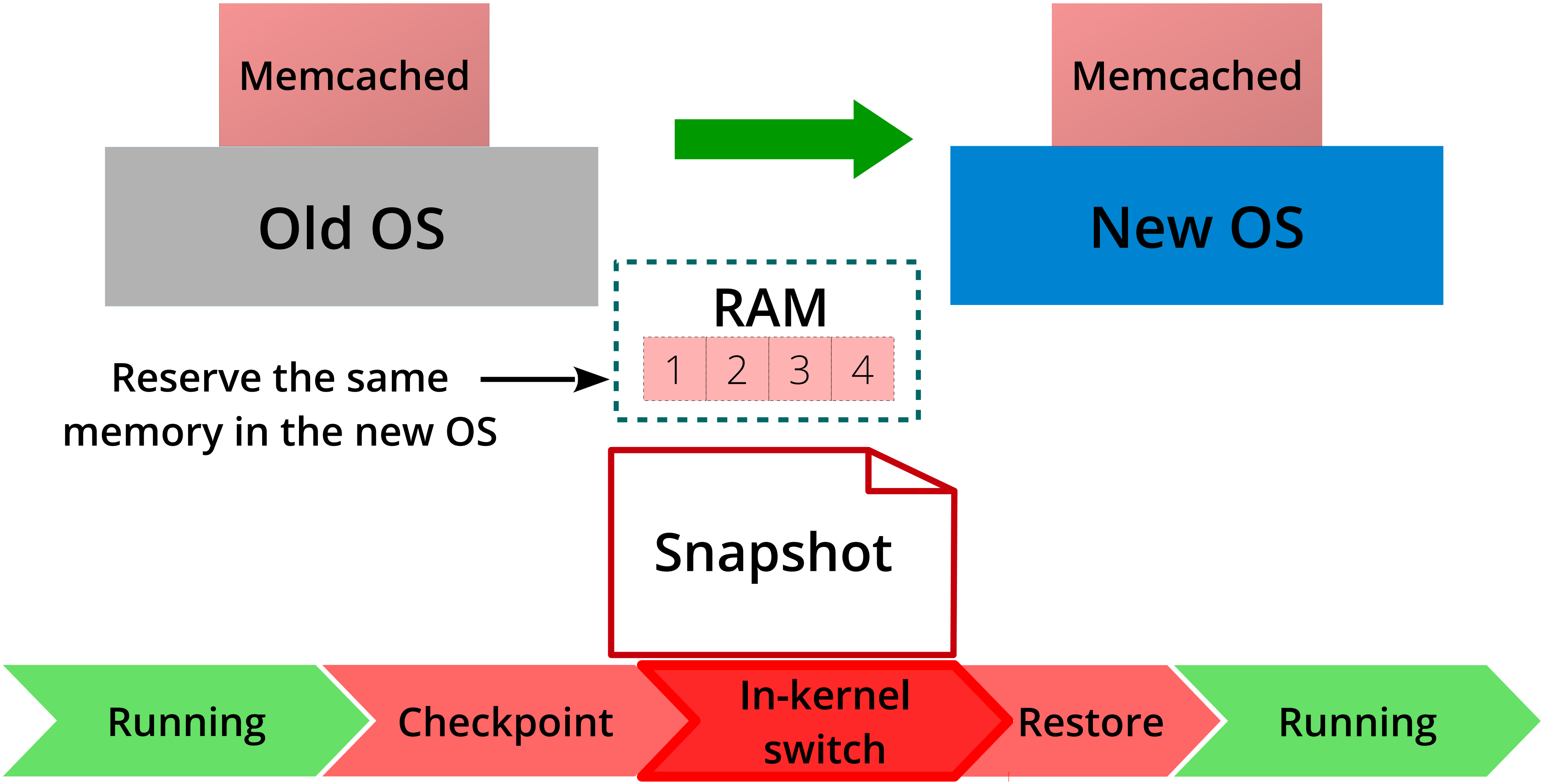
Avoid redundant data copy across reboot

- Reserve the application's memory across reboot
- Inherently rebind the memory without any copy



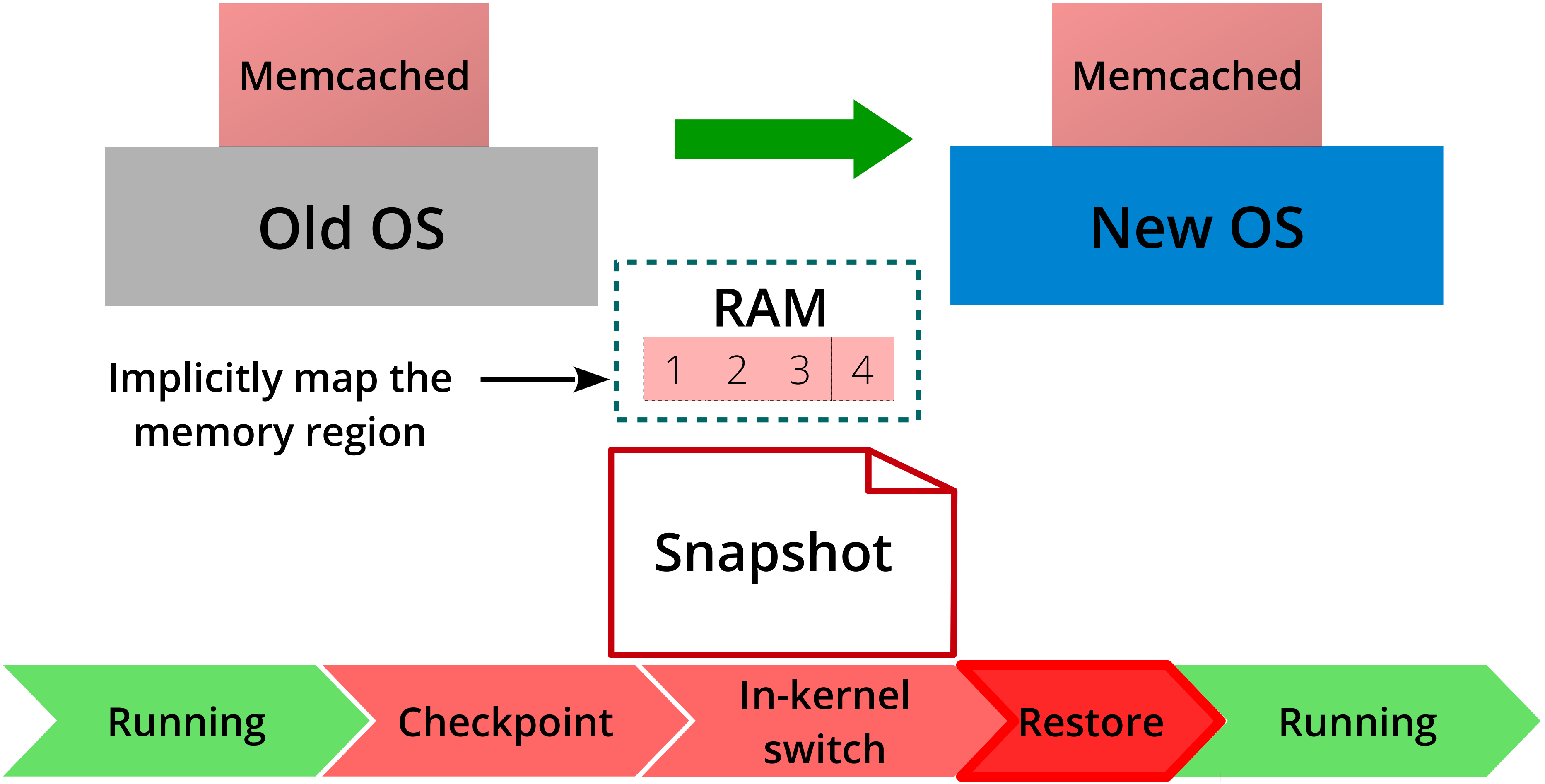
Avoid redundant data copy across reboot

- Reserve the application's memory across reboot
- Inherently rebind the memory without any copy



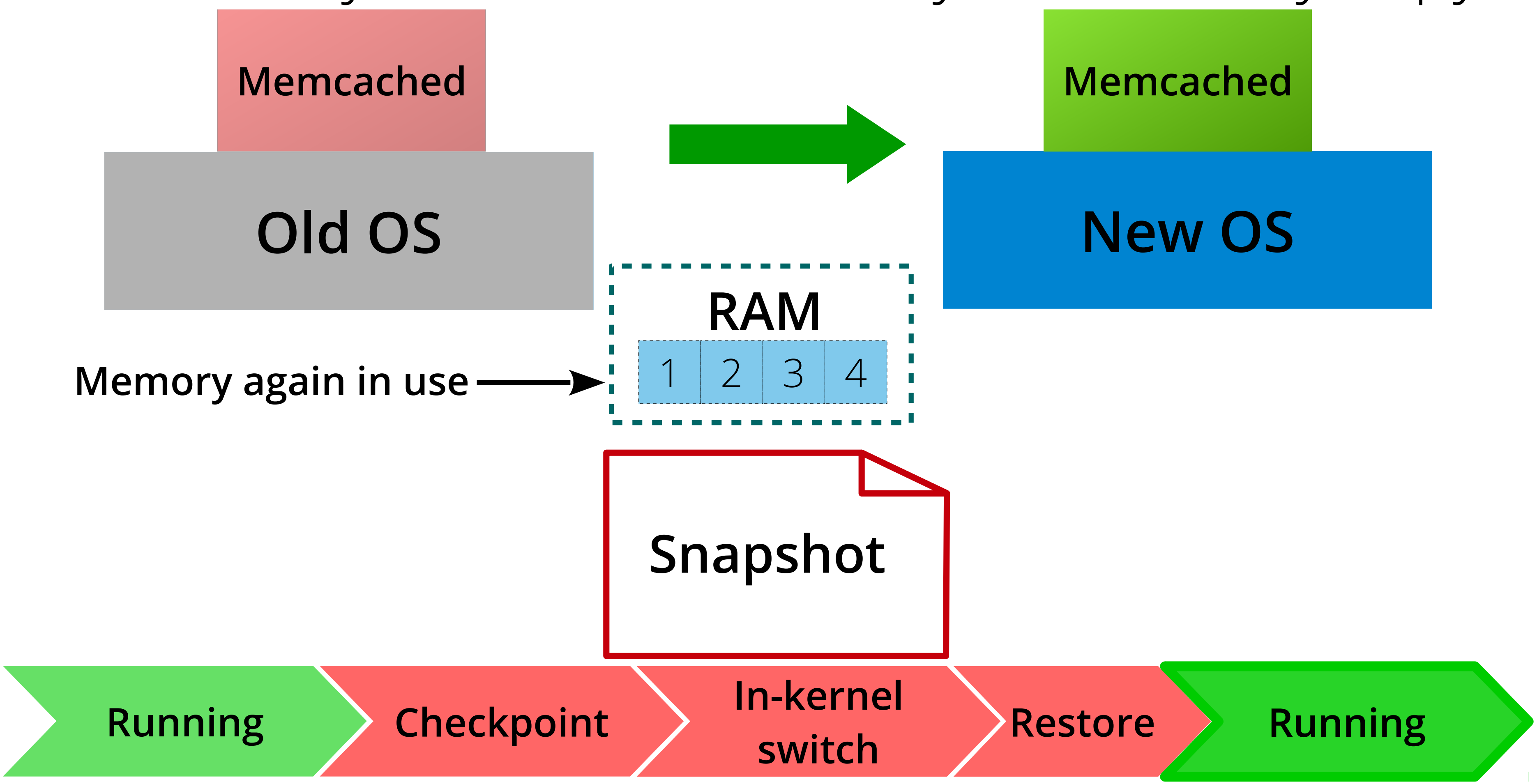
Avoid redundant data copy across reboot

- Reserve the application's memory across reboot
- Inherently rebind the memory without any copy



Avoid redundant data copy across reboot

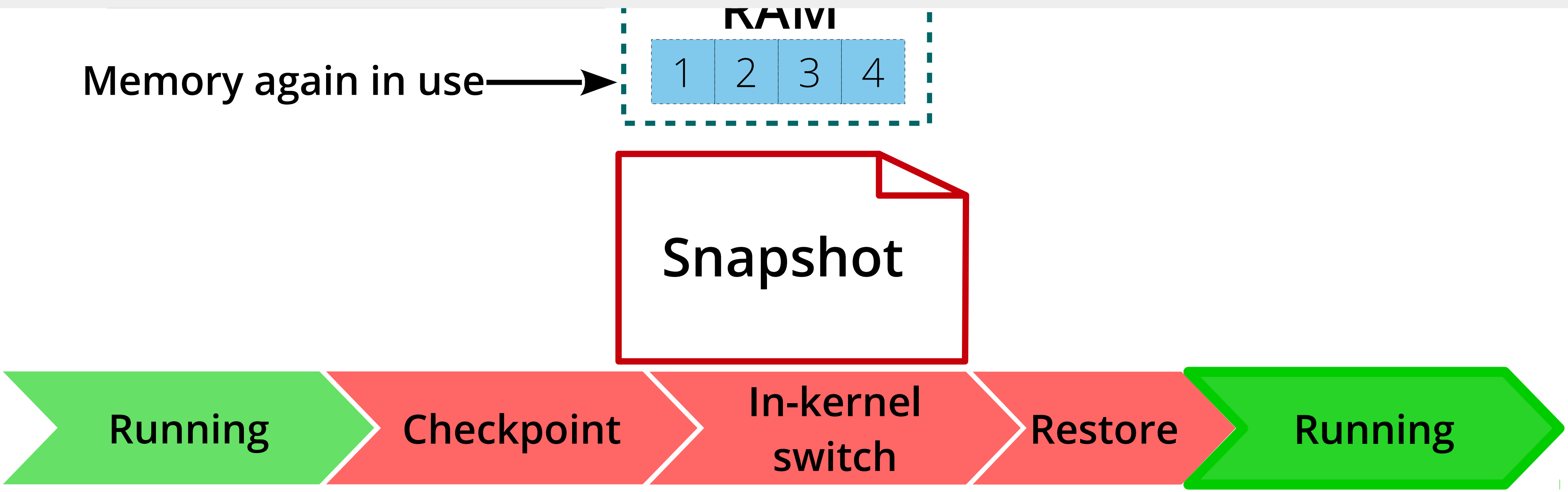
- Reserve the application's memory across reboot
- Inherently rebind the memory without any copy



Avoid redundant data copy across reboot

- Reserve the application's memory across reboot
- Inherently rebind the memory without any copy

Challenge: how to notify the newer OS without modifying its source?



Persist physical pages (PPP) without OS modification

- Reserve virtual-to-physical mapping information
  - Static instrumentation of the OS binary
  - Inject our own memory reservation function, then further boot the OS
- Handle page-faults for the restored application
  - Dynamic kernel instrumentation
  - Inject our own page fault handler function for memory binding

Persist physical pages (PPP) without OS modification

- Reserve virtual-to-physical mapping information

- **No explicit memory copy**
- **Does not require any kernel source modification**

  - Dynamic kernel instrumentation
  - Inject our own page fault handler function for memory binding

Implementation

- Application C/R → *criu*
  - Works at the namespace level
- In-kernel switch → *kexec* system call
  - A mini boot loader that bypasses BIOS while booting

Component	**Lines of code**
`criu` / on-demand restore	810 lines of C
`criu` / `FOAM`	950 lines of C
`criu` / `PPP`	600 lines of C
KUP `systemd, init`	1040 lines of Python/Bash
`criu` / others, `kexec()`, etc.	150 lines of C
Total	3,550 lines of code

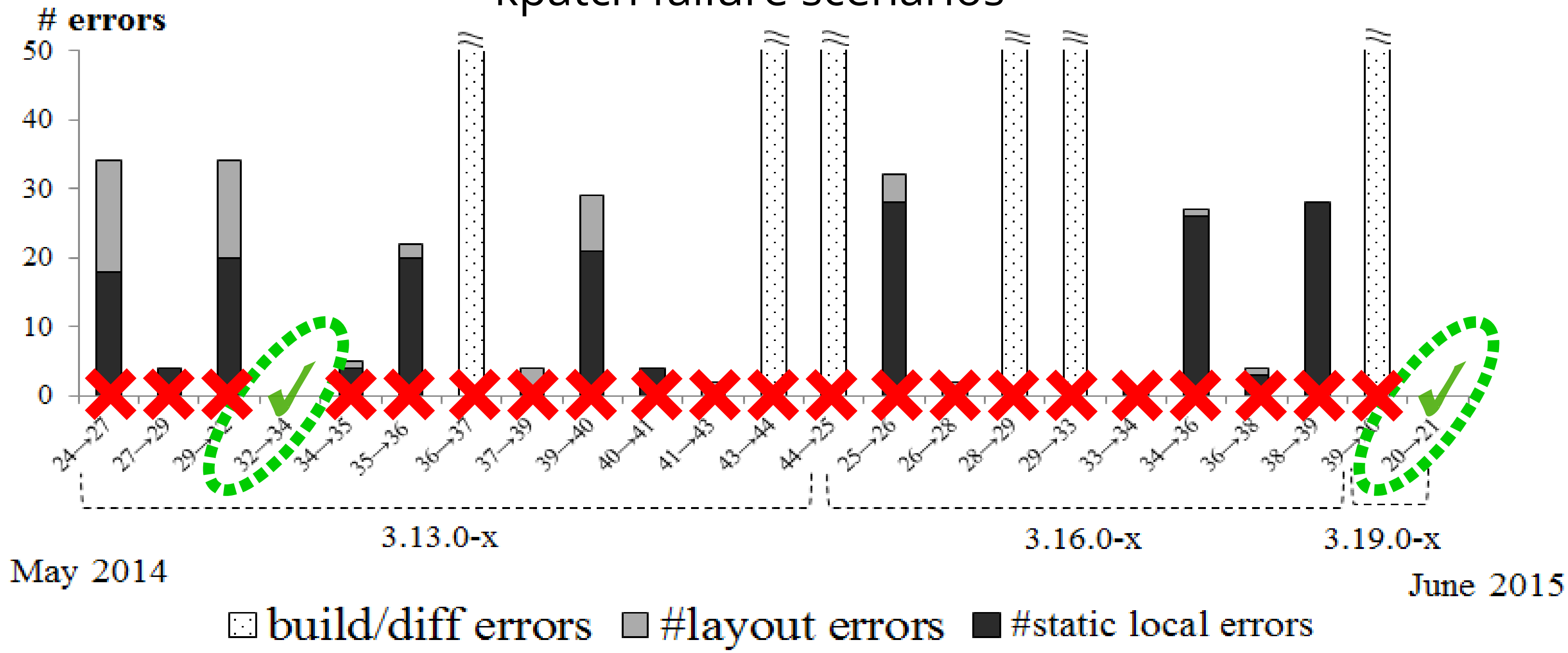
Evaluation

- How effective is KUP's approach compared to the in-kernel hot patching?
- What is the effective performance of each technique during the update?

KUP can support major and minor updates in Ubuntu

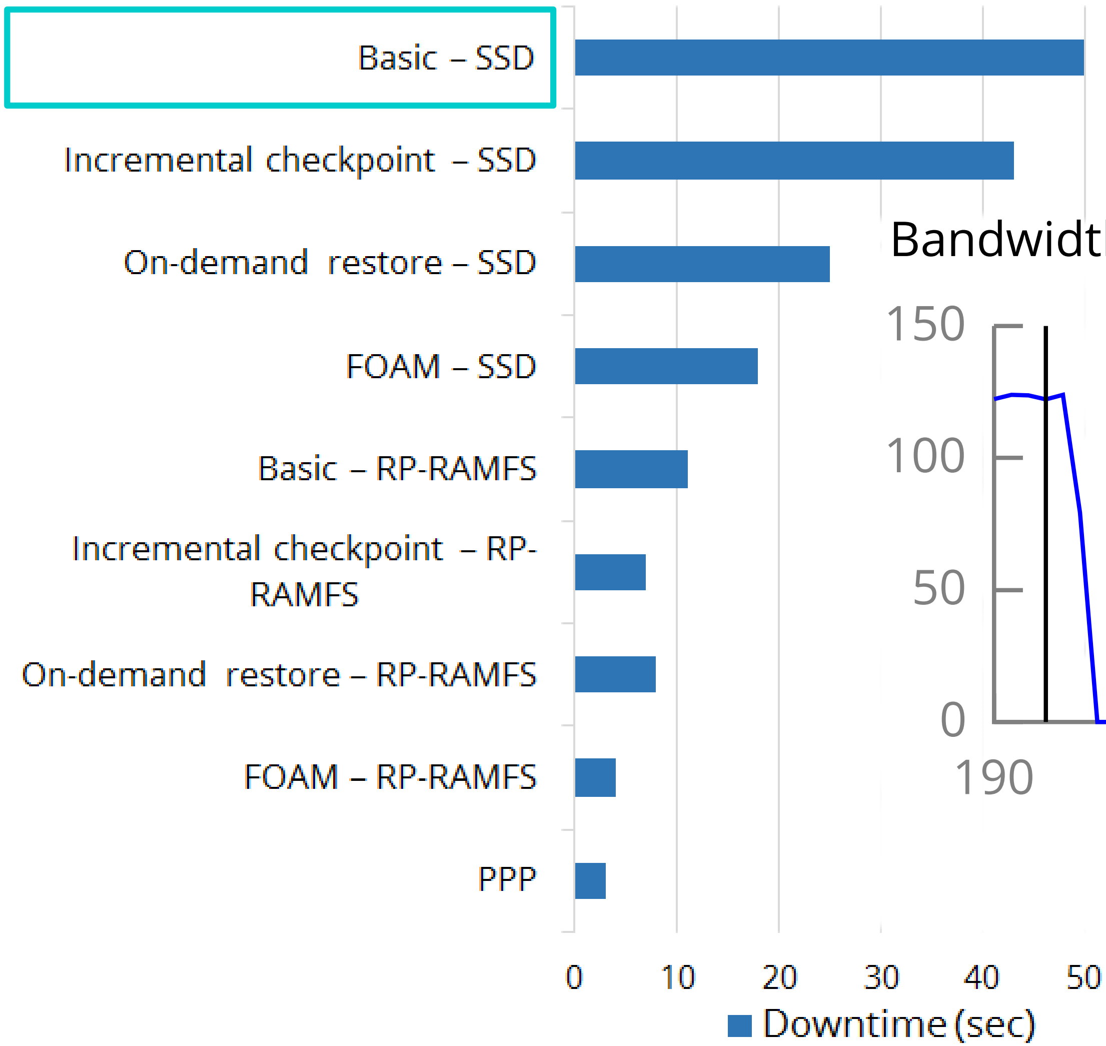
- KUP supports 23 minor/4 major updates (v3.17–v4.1)
- However, kpatch can only update 2 versions
  - e.g., layout change in data structure

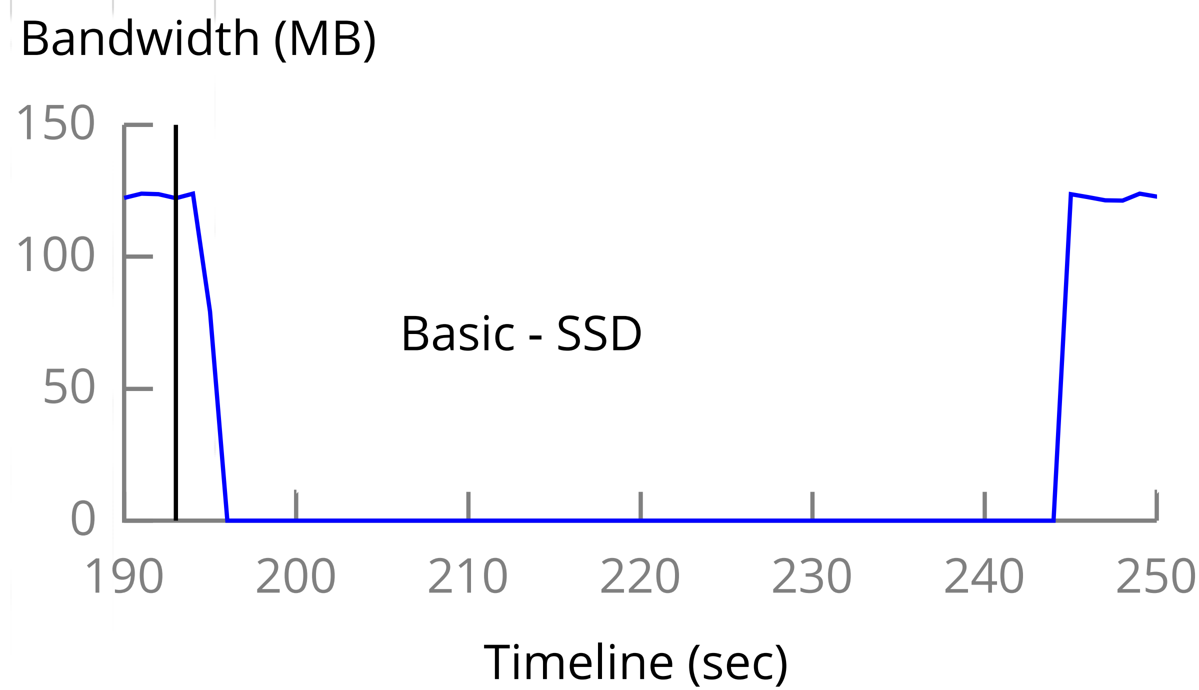
kpatch failure scenarios



Updating OS with memcached

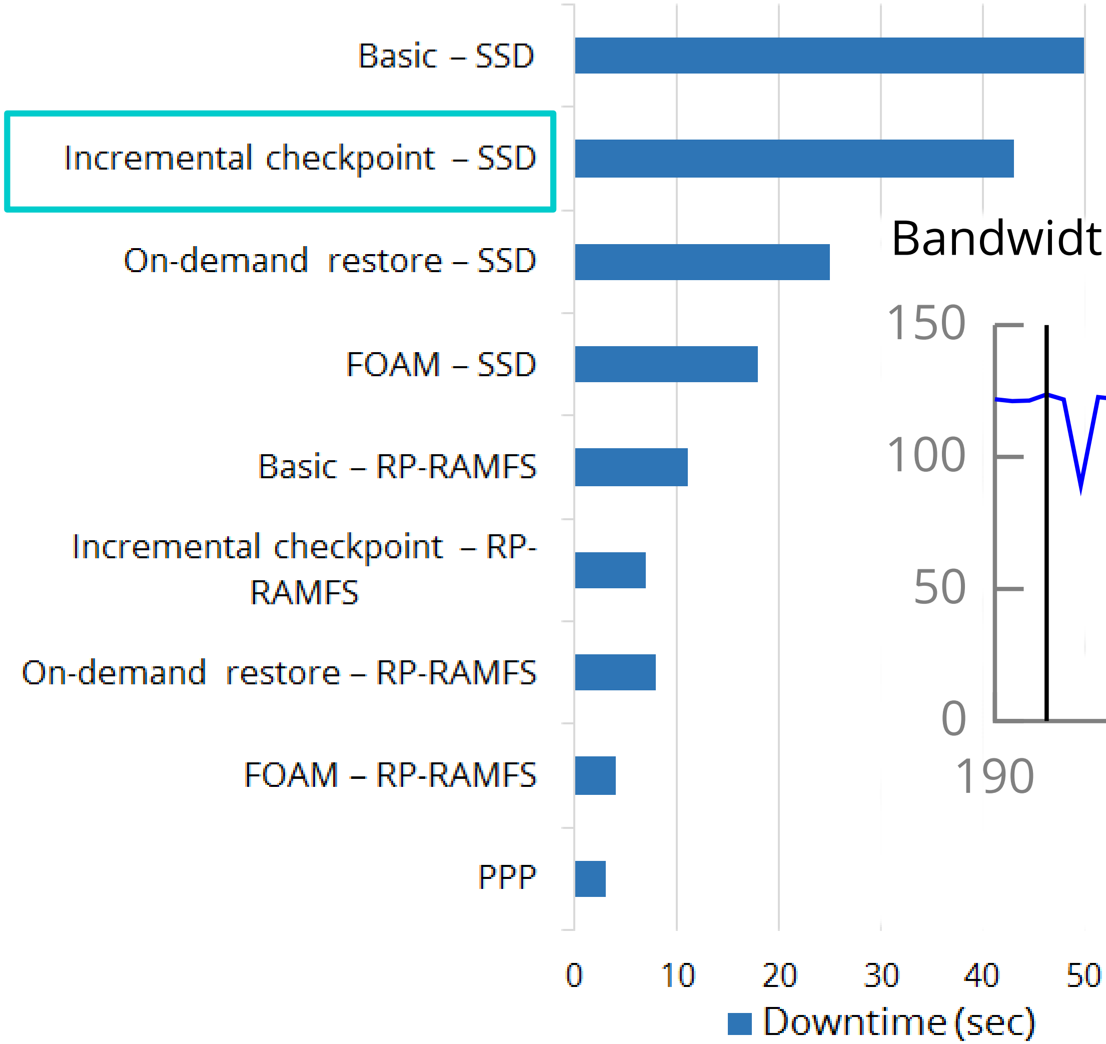
- PPP has the least degradation
- Storage also affects the performance

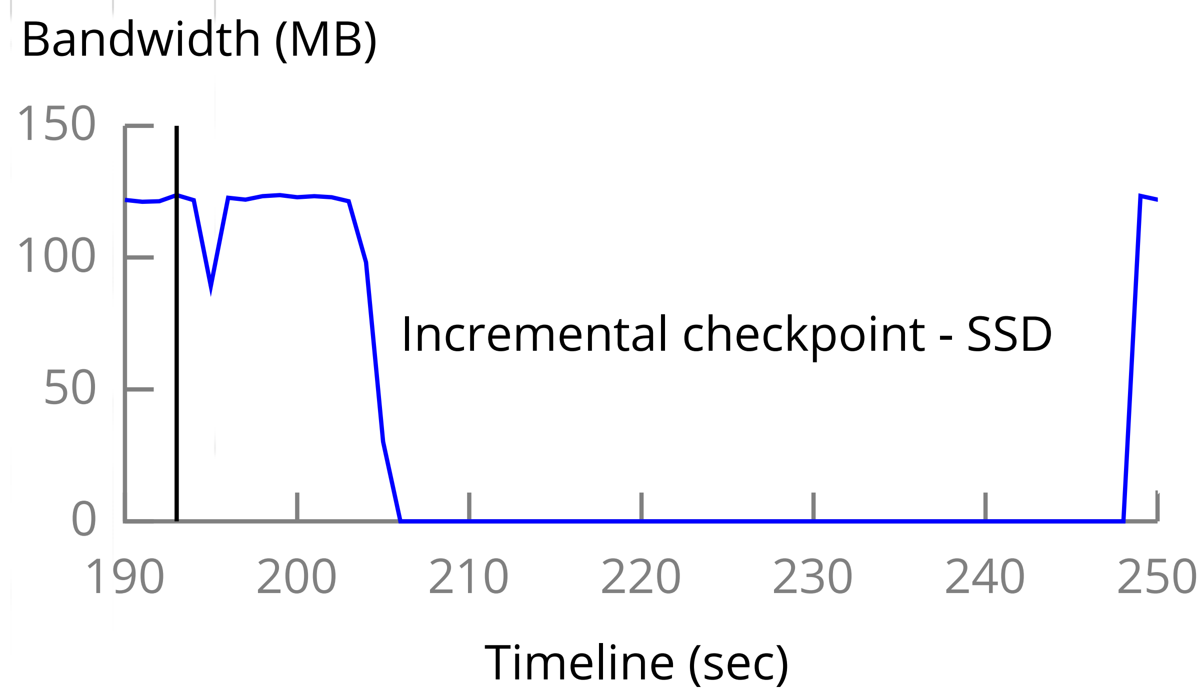




Updating OS with memcached

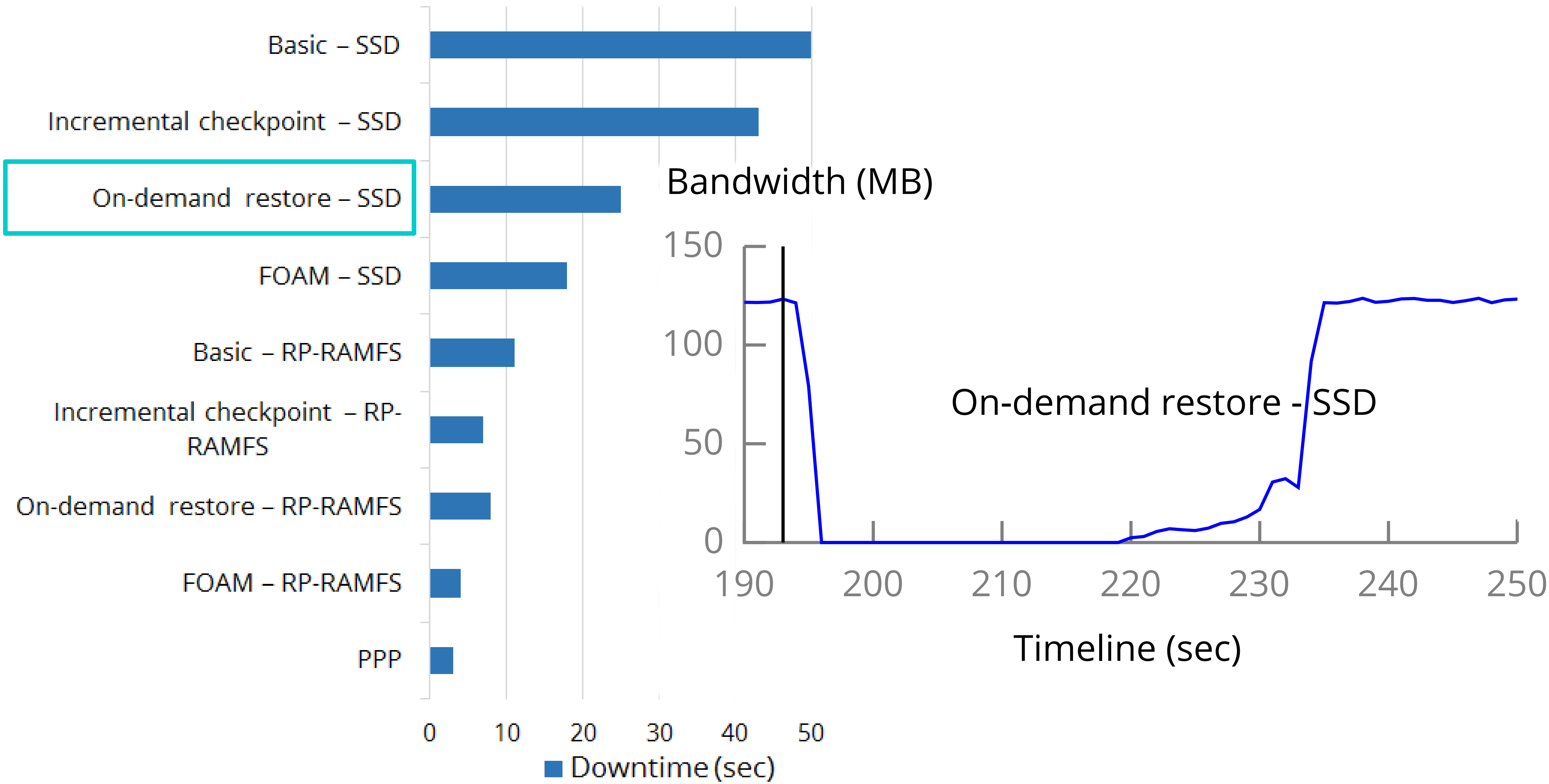
- PPP has the least degradation
- Storage also affects the performance





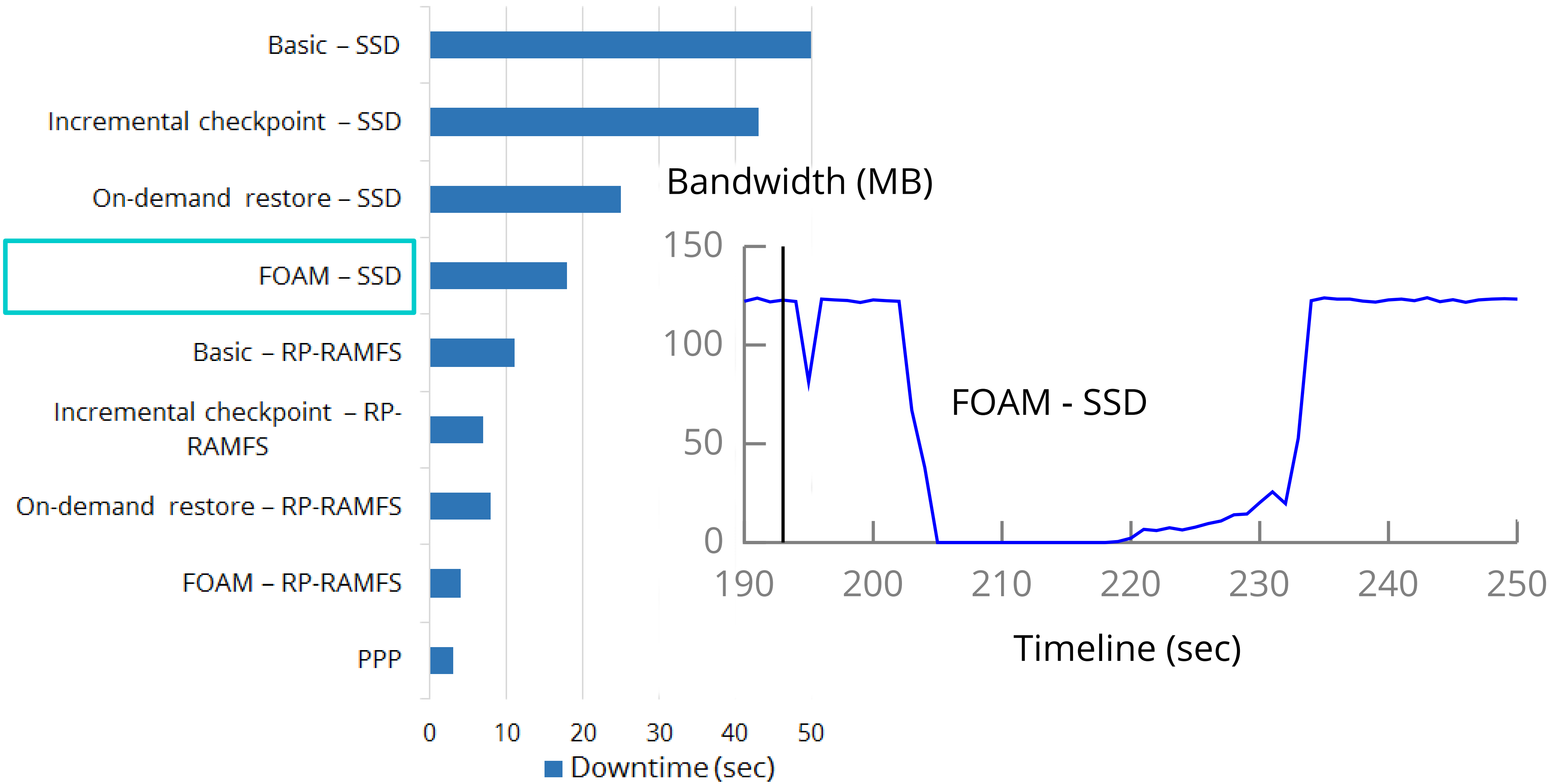
Updating OS with memcached

- PPP has the least degradation
- Storage also affects the performance



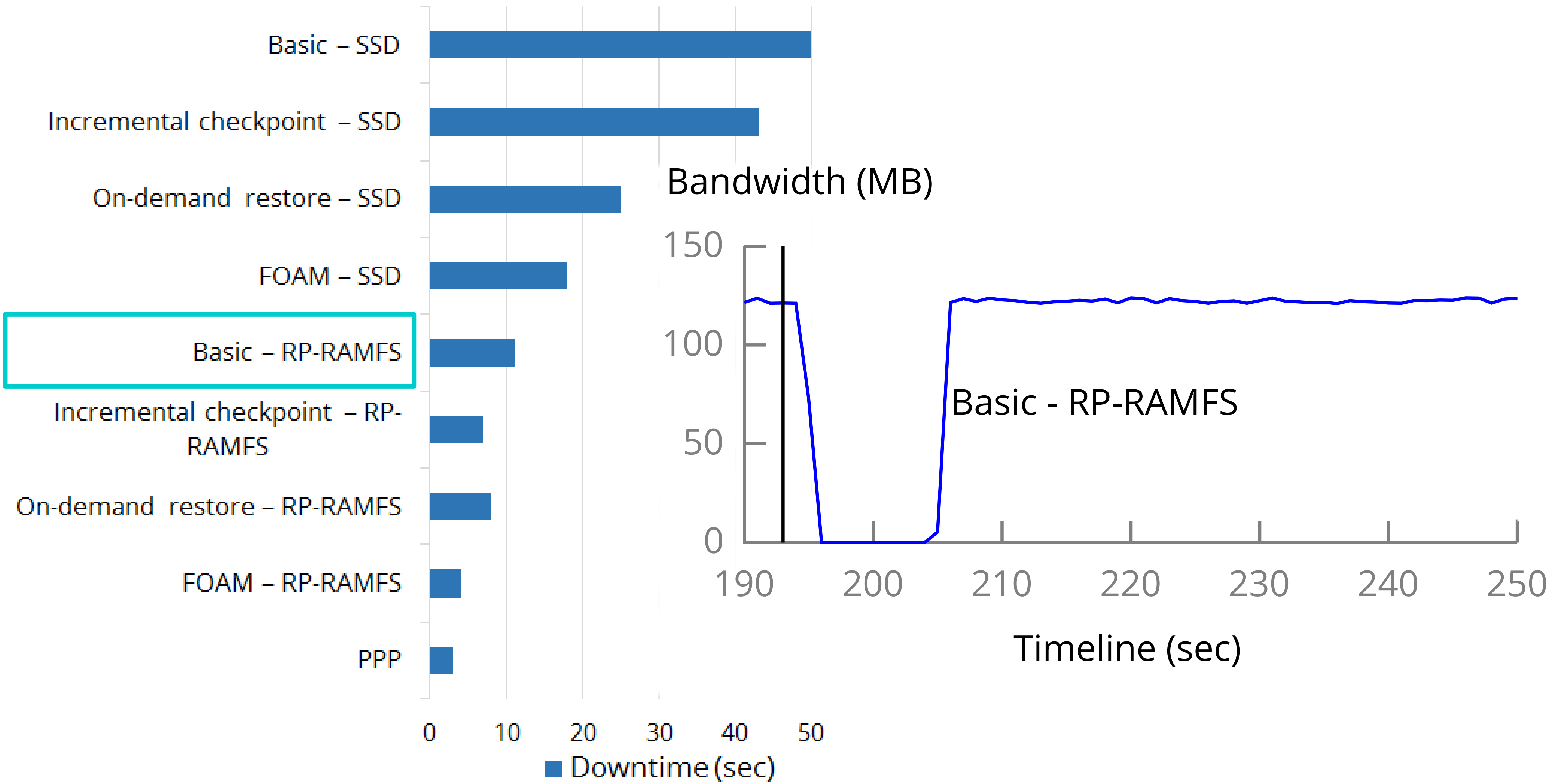
Updating OS with memcached

- PPP has the least degradation
- Storage also affects the performance



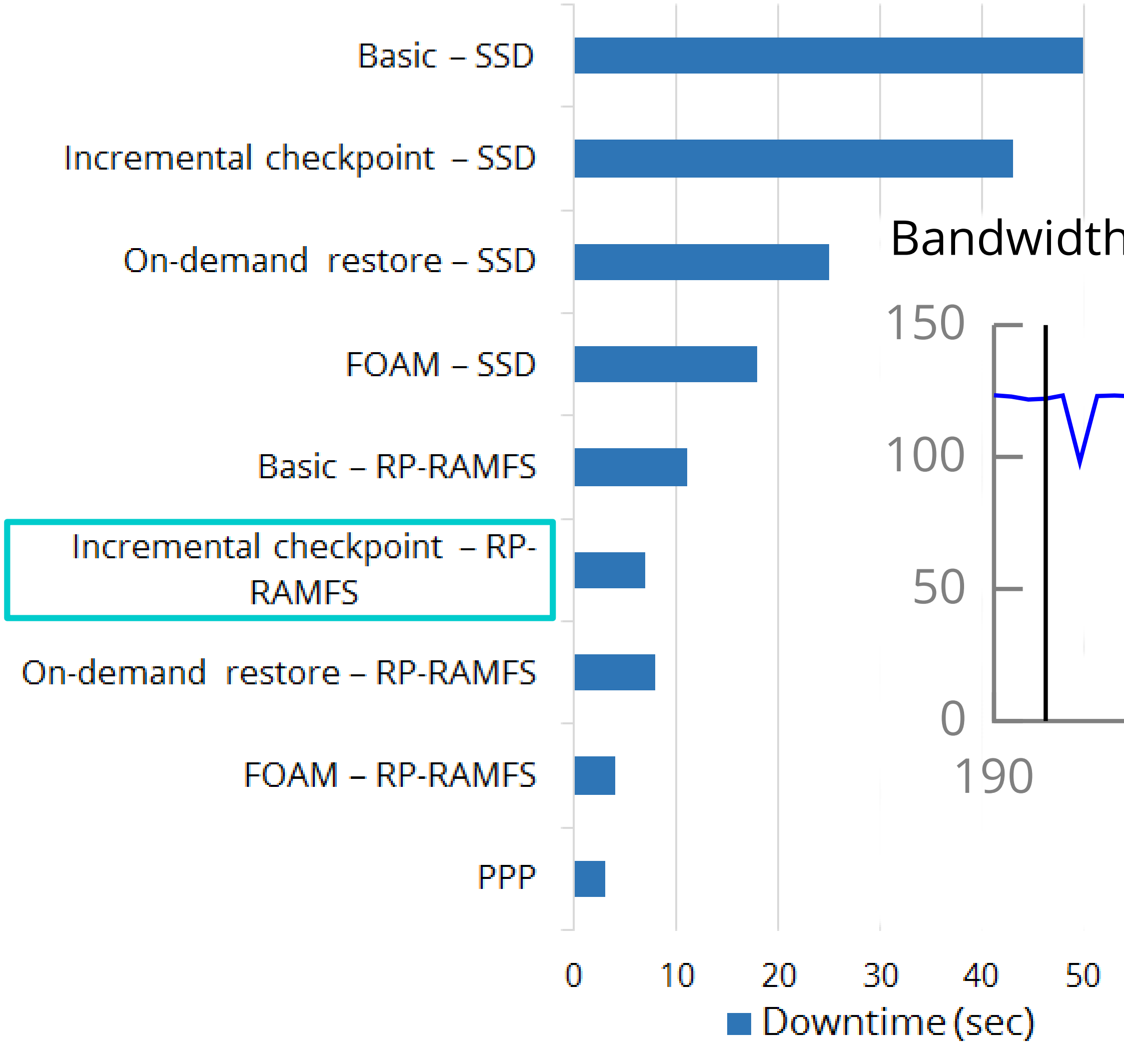
Updating OS with memcached

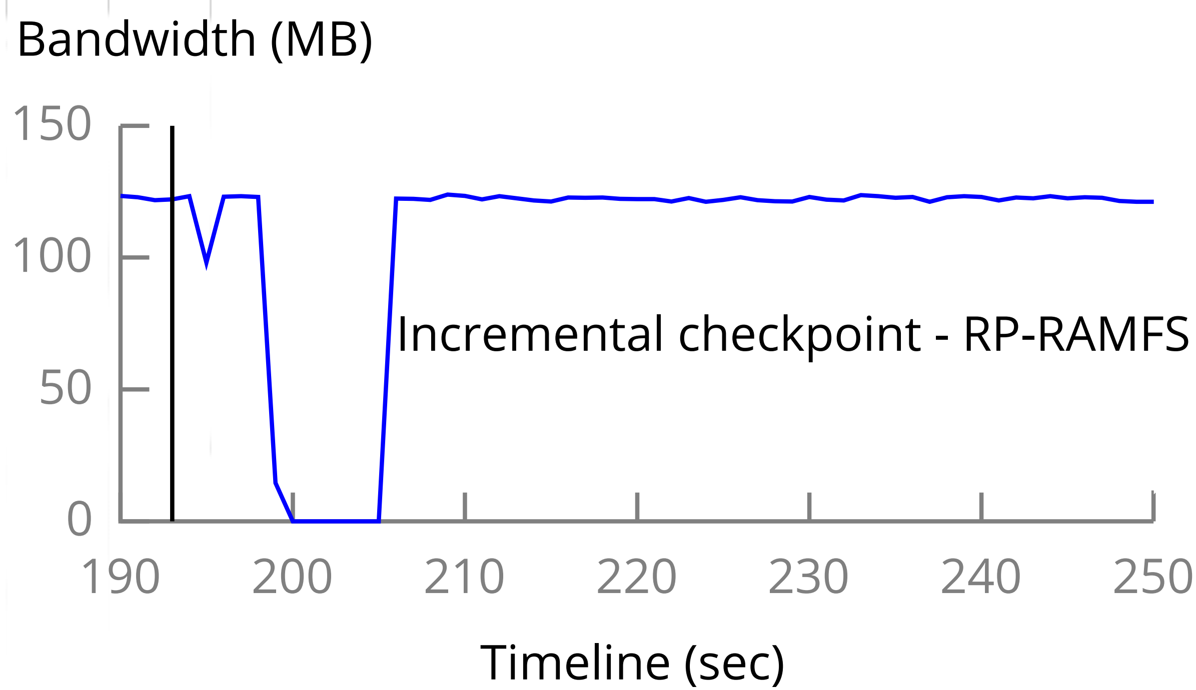
- PPP has the least degradation
- Storage also affects the performance



Updating OS with memcached

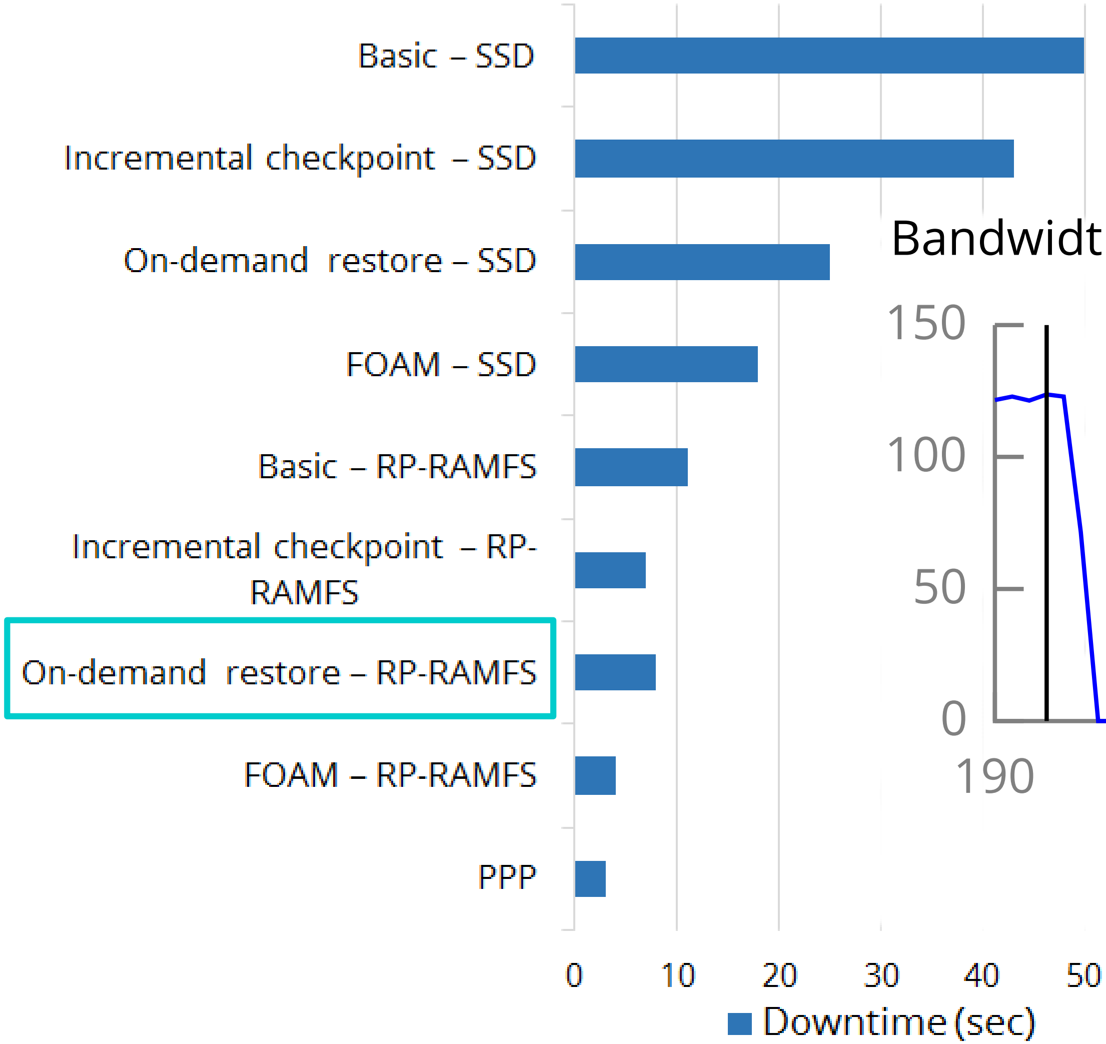
- PPP has the least degradation
- Storage also affects the performance

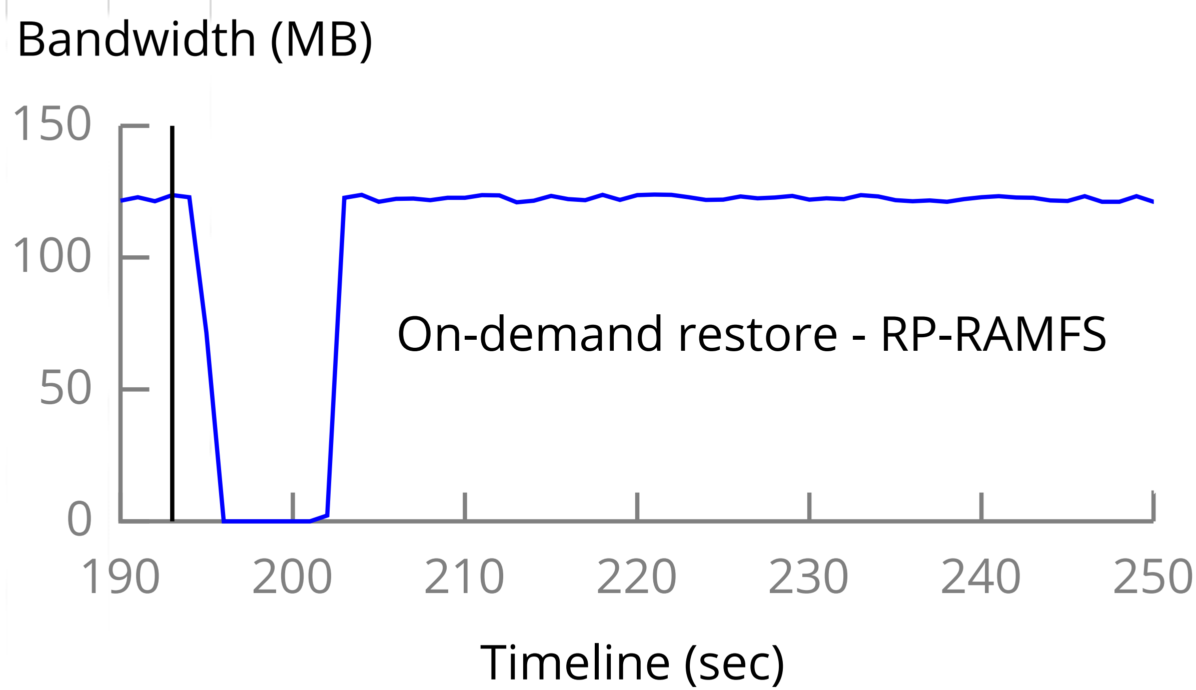




Updating OS with memcached

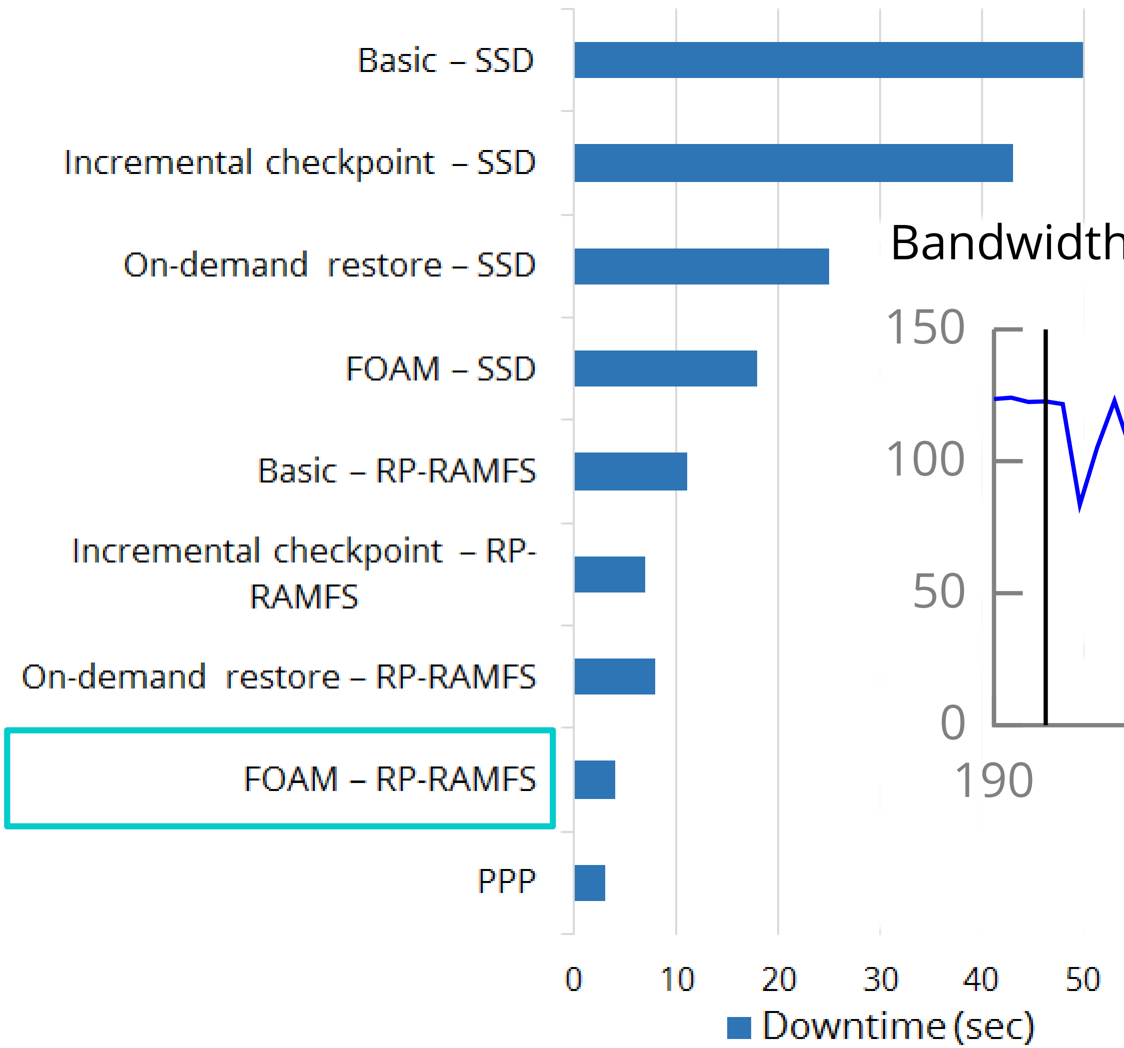
- PPP has the least degradation
- Storage also affects the performance

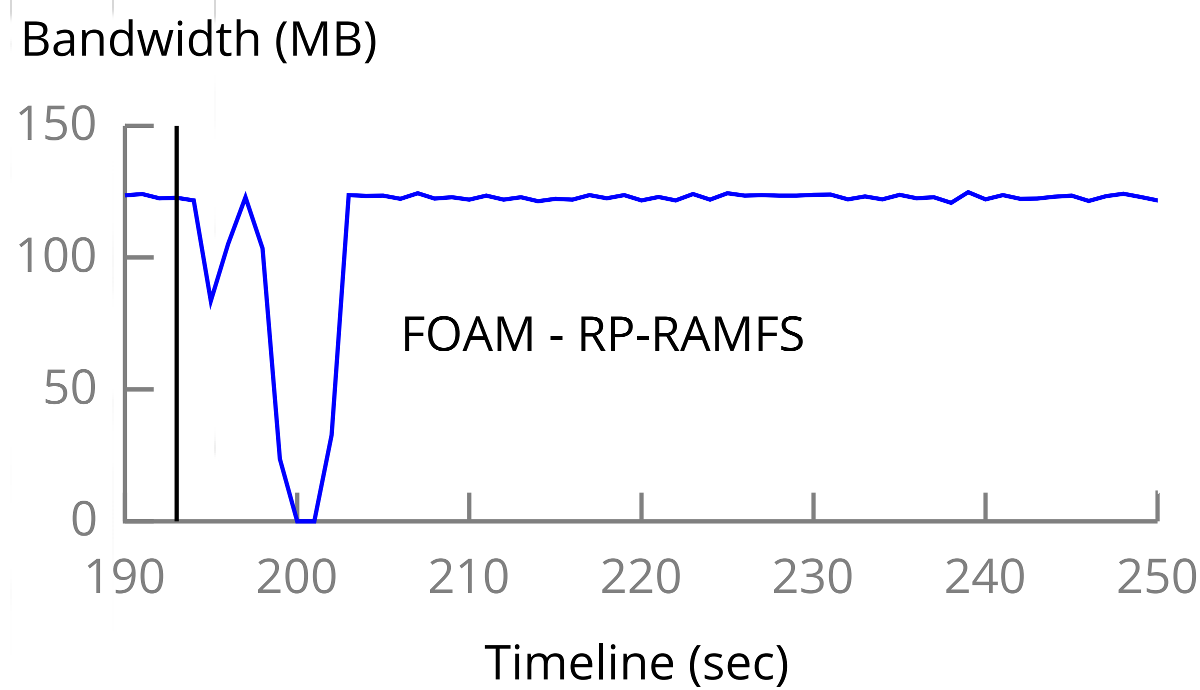




Updating OS with memcached

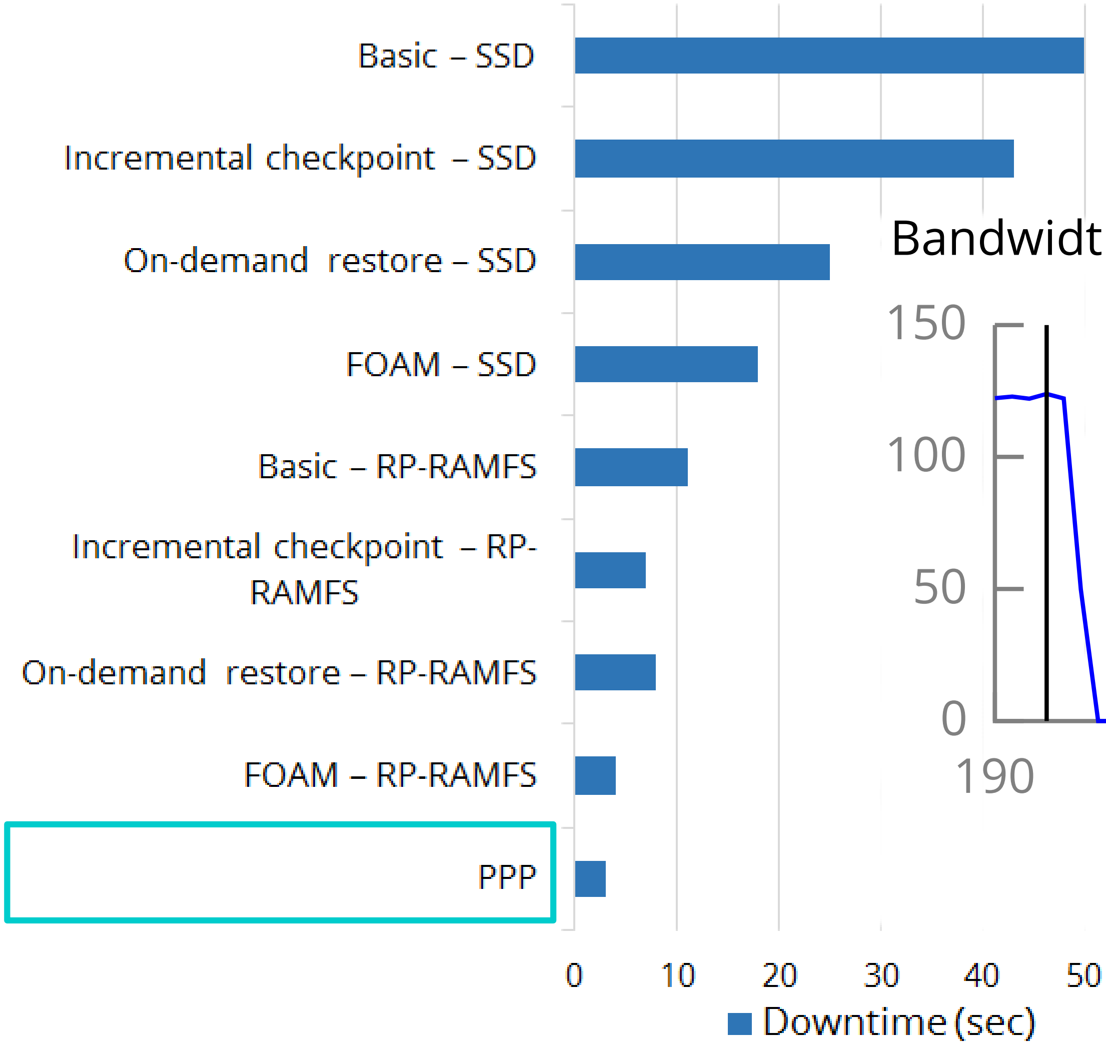
- PPP has the least degradation
- Storage also affects the performance

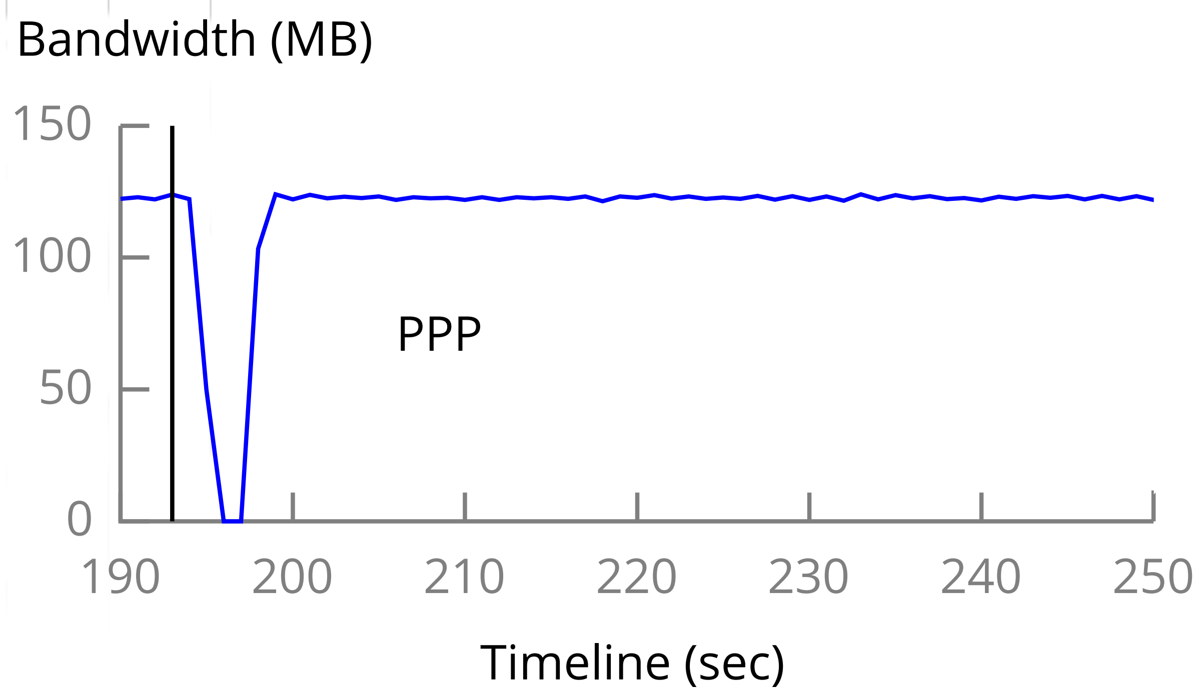




Updating OS with memcached

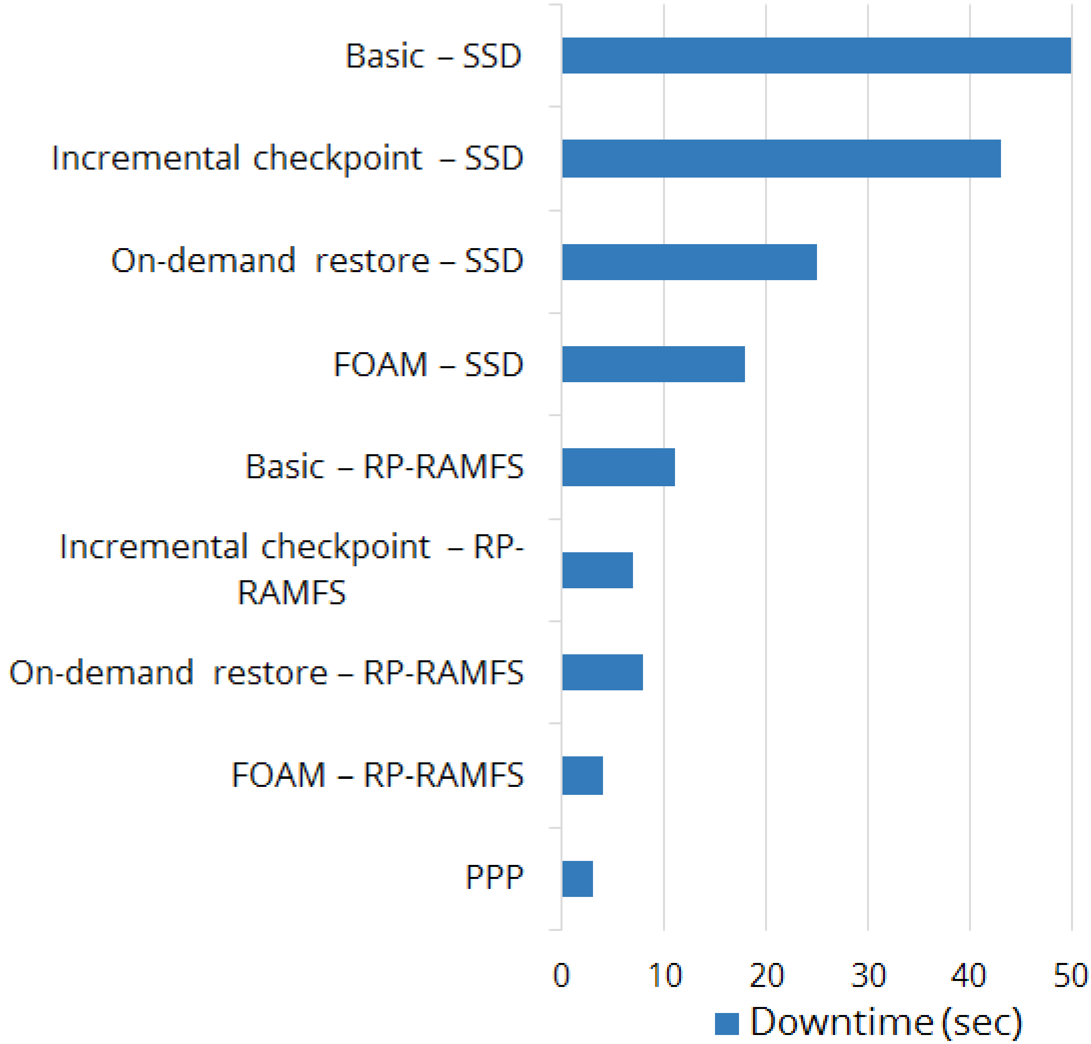
- PPP has the least degradation
- Storage also affects the performance

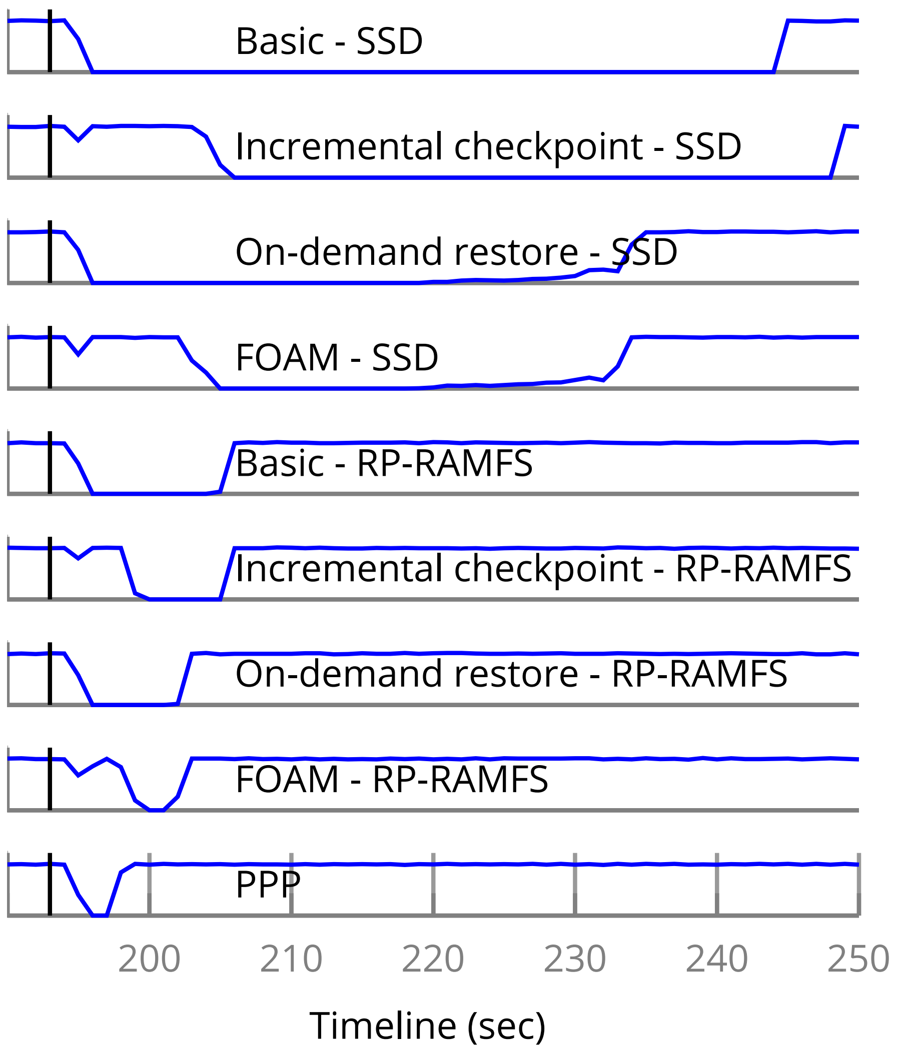




Updating OS with memcached

- PPP has the least degradation
- Storage also affects the performance





Limitations

- KUP does not support checkpoint and restore all socket implementations
  - TCP, UDP and netlink are supported
- Failure during restoration
  - System call is removal or interface modification

Demo

Summary

- KUP: a simple update mechanism with application checkpoint-and-restore (C/R)
- Employs various techniques:
  - New data abstraction for application C/R
  - Fast in-kernel switching technique
  - A simple mechanism to persist the memory

Summary

- KUP: a simple update mechanism with application checkpoint-and-restore (C/R)
- Employs various techniques:
  - New data abstraction for application C/R
  - Fast in-kernel switching technique
  - A simple mechanism to persist the memory

Thank you!

Backup Slides

Handling in-kernel states

- Handles namespace and cgroups
- ptrace() syscall to handle the blocking system calls , timers, registers etc.
- Parasite code to fetch / put the application's states
- /proc file system exposes the required information for application C/R
- A new mode (TCP_REPAIR) allows handling the TCP connections

What cannot be checkpointed

- X11 applications
- Tasks with debugger attached
- Tasks running in compat mode (32 bit)

Possible changes after application C/R

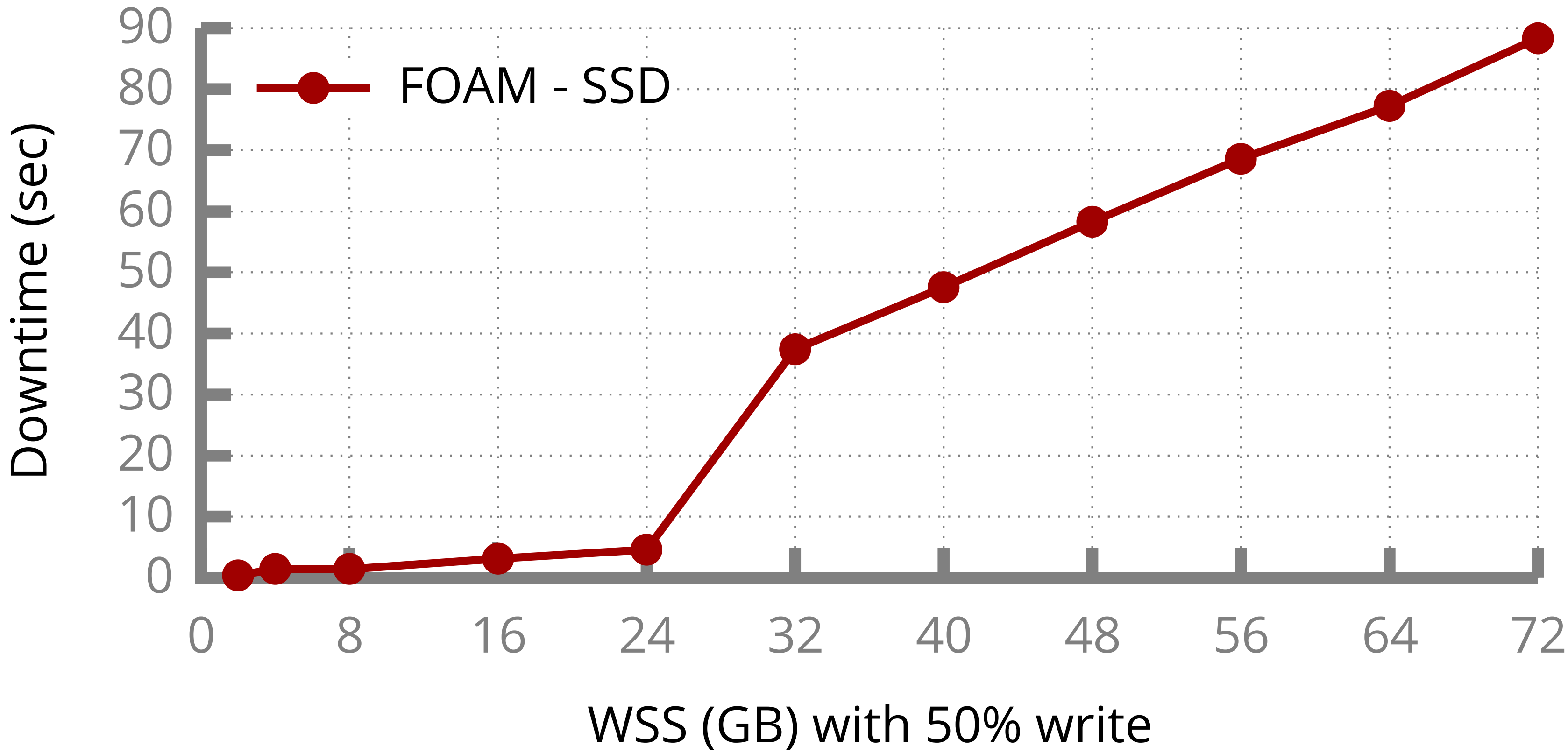
- Per-task statistics
- Namespace IDs
- Process start time
- Mount point IDs
- Socket IDs (st_ino)
- VDSO

Suitable applications

- Suitable for all kinds of applications
- PPP approach supports all types of applications
  - May fail to restore on the previous kernel
- FOAM is not a good candidate for write-intensive applications
  - More confidence in safely restoring the application on the previous kernel

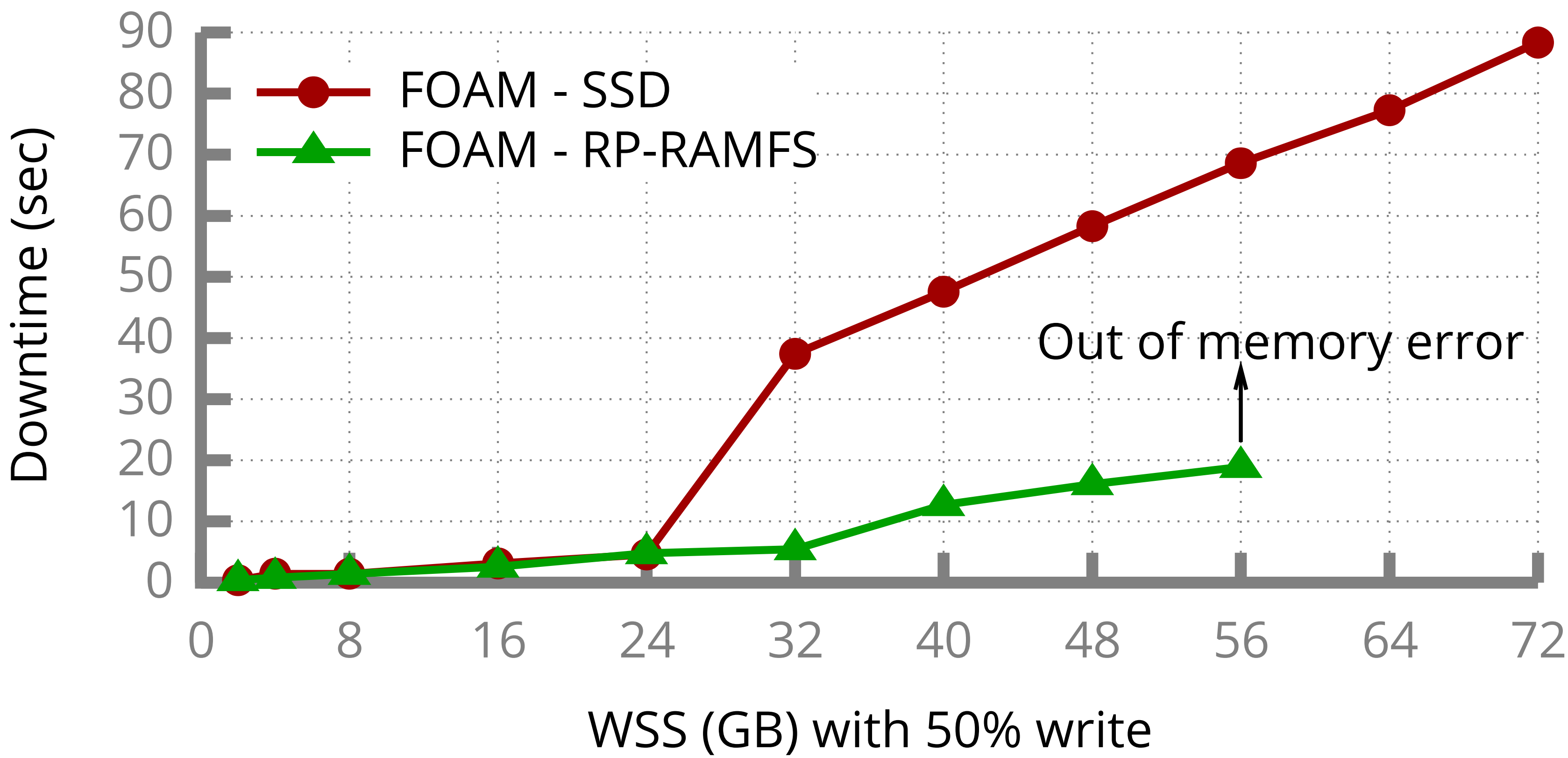
PPP works effectively

- FOAM on SSD → slow
- FOAM on RP-RAMFS → space inefficient



PPP works effectively

- FOAM on SSD → slow
- FOAM on RP-RAMFS → space inefficient



PPP works effectively

- FOAM on SSD → slow
- FOAM on RP-RAMFS → space inefficient