Reduction of Operating System Jitter Caused by Page Reclaim

Yoshihiro Oyama[1,3] Shun Ishiguro[1] Jun Murakami[1]

Shin Sasaki[1] Ryo Matsumiya[1] Osamu Tatebe[2,3]

1. The University of Electro-Communications
2. University of Tsukuba
3. Japan Science and Technology Agency

Background

- OS jitter: interference into applications by OS
  - Services by OS kernel
    - E.g., interrupt handling and tasklets
  - Daemon processes developed to provide OS services
    - E.g., memory management daemons
- Jitter degrades application performance
  - It deprives applications from computing resources such as CPU and memory
- Minimizing the impact of jitter is critical in HPC

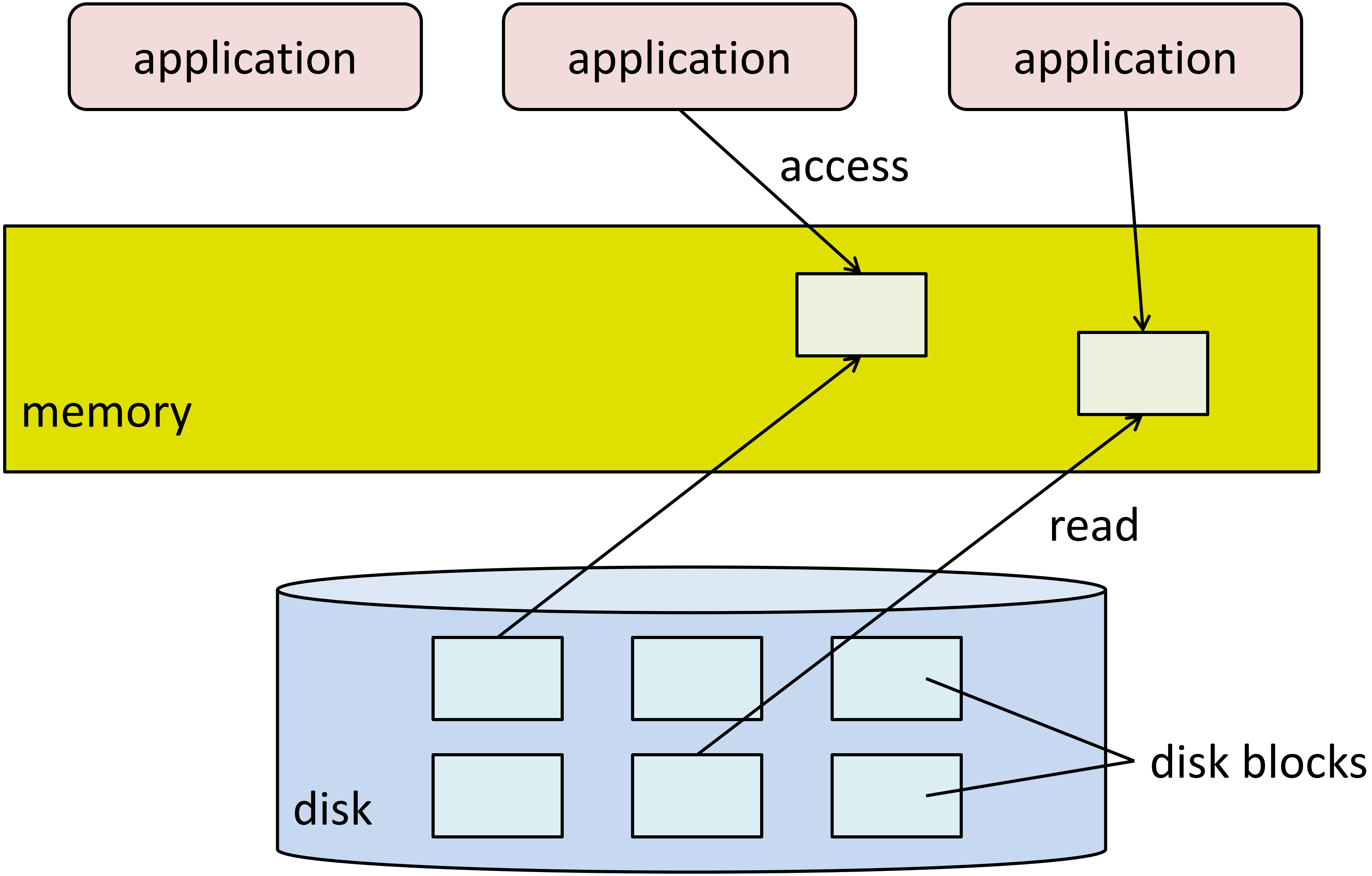
Jitter Focused in This Study

- We focus on jitter observed when application frequently executes disk I/O of large data
  - Footprint of file data exceeds the physical memory size
  - Kernel must discard page cache or swap out processes to obtain free memory
  - Overhead is imposed on memory allocation operations
- This jitter has not attracted much attention, but HPC people should be aware of its potential impact

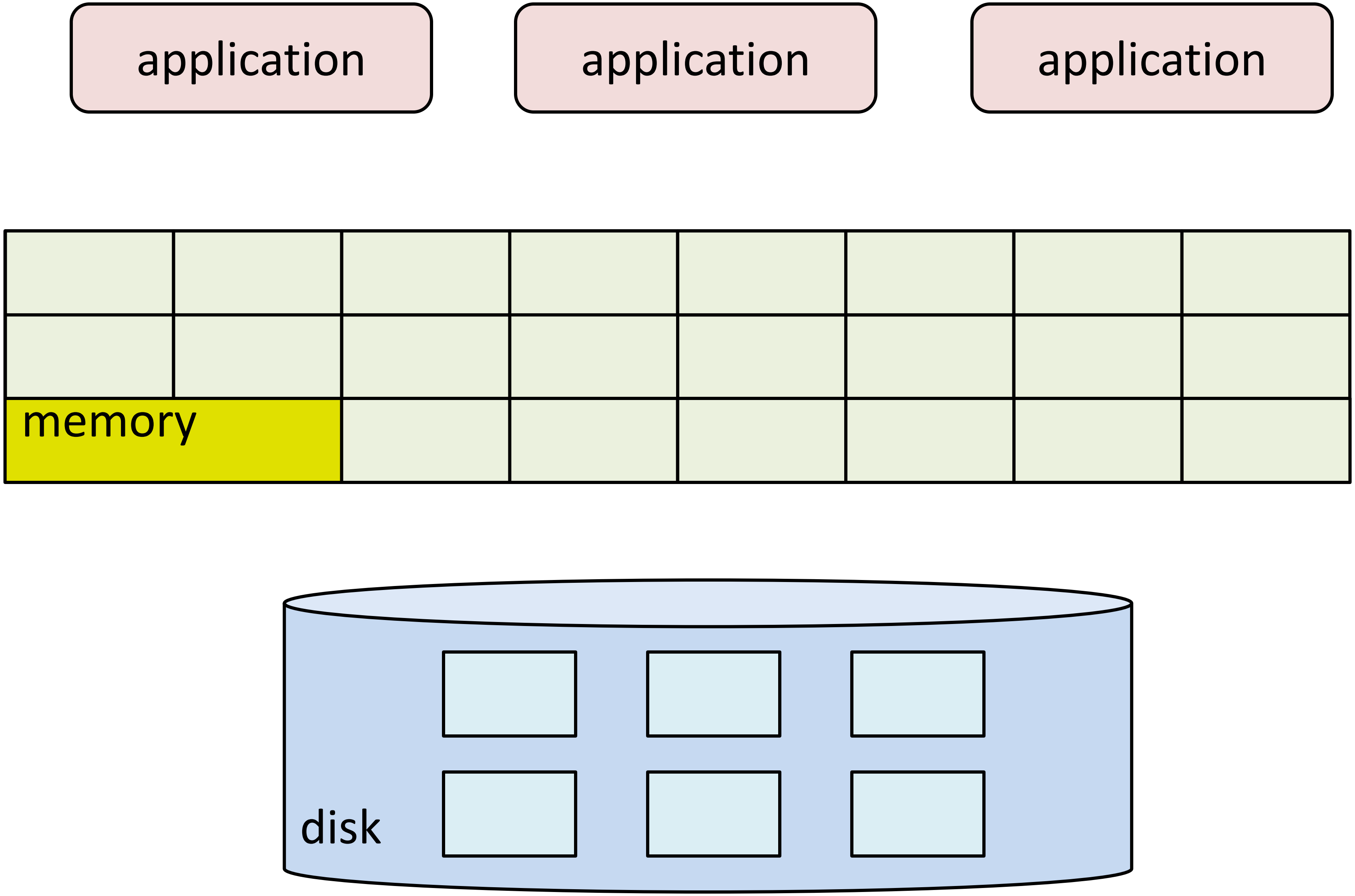
Overview of This Study

1. We clarify the impact of the jitter caused by page reclaim
   - Target OS is Linux
2. We propose a mechanism for minimizing the impact
   - It increases the amount of page cache released at one time
   - It reduces the number of page reclaim operations

Page Cache


application
application
application
access
memory
read
disk
disk blocks

Memory Pressure by Page Cache



application

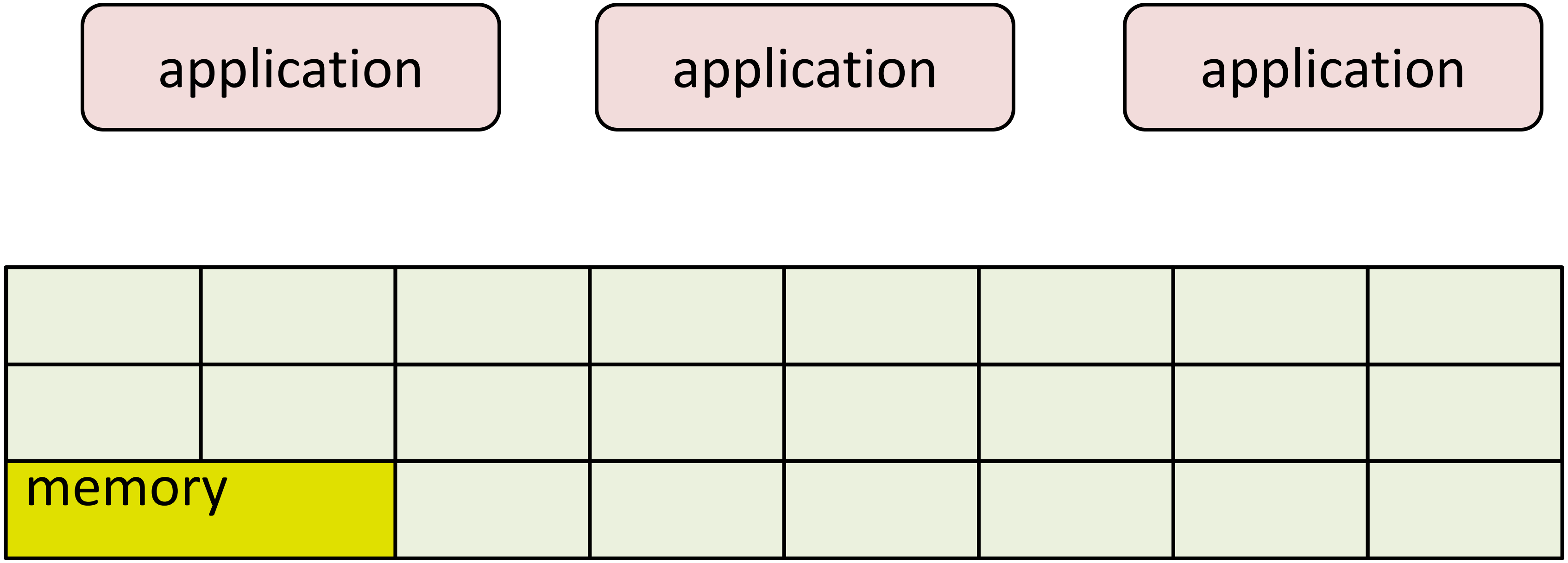
application

application

memory

disk

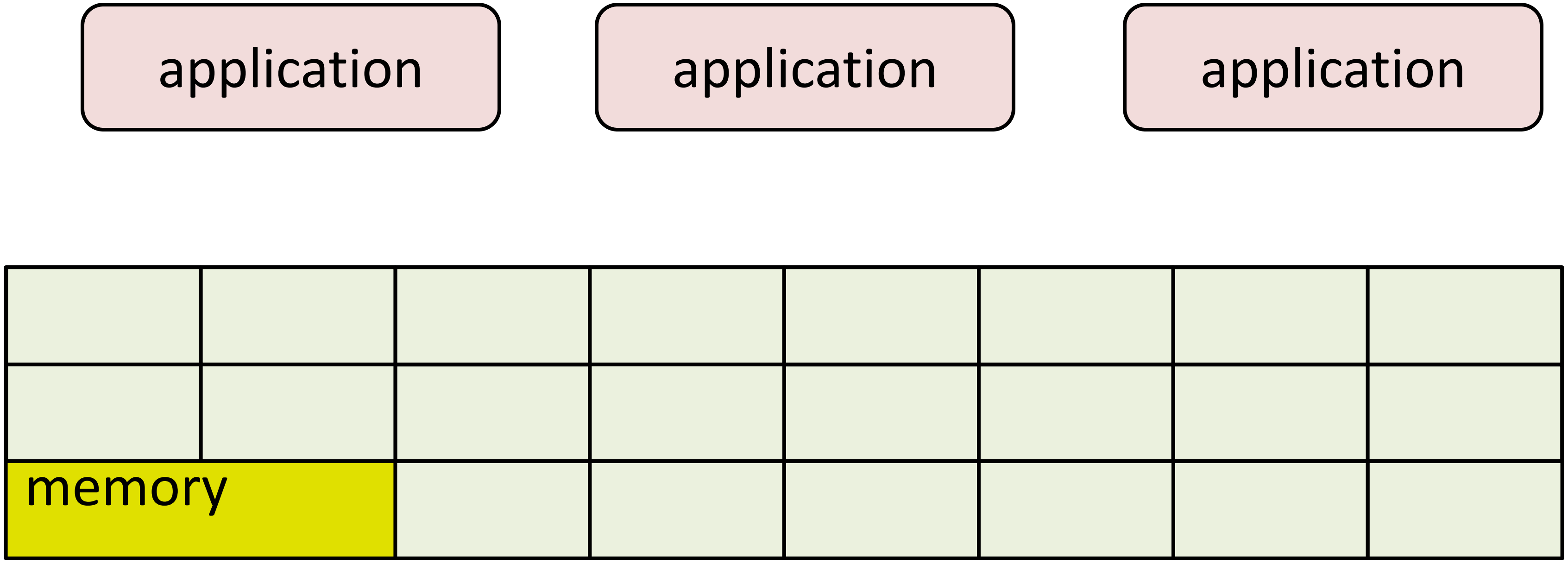
Memory Pressure by Page Cache



Page reclaim frequently occurs when:

- Pages are consumed fast
- Only a small number of pages are released at one time

Memory Pressure by Page Cache



application

application

application

memory

Page reclaim frequently occurs when:

- Pages are consumed fast
- Only a small number of pages are released at one time

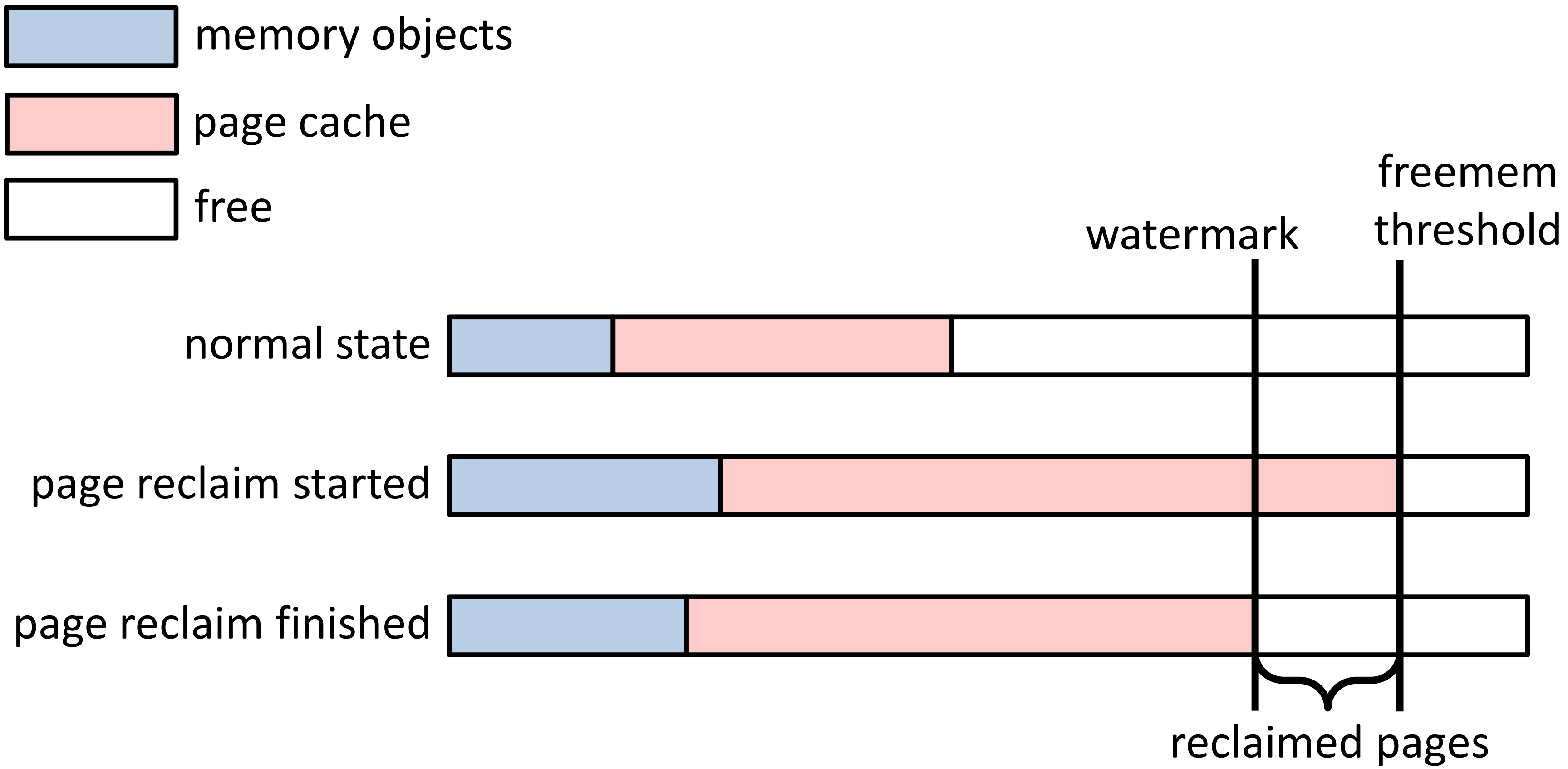
Page Reclaim in Linux

- Memory pages are running short
  -> it immediately reclaims memory (direct reclaim)
  -> it awakens a kernel thread kswapd
- kswapd reclaims pages by flushing page cache or swapping process memory
- Two values inside kswapd are particularly important
  - Freemem threshold
    - Kswapd is awakened if the amount of free memory falls below this threshold
  - Watermark
    - Kswapd continues to reclaim pages until the amount of free pages exceeds this value

Modifiable indirectly through `/proc/sys/vm/min_free_kbytes`

Unfortunately, it is the only parameter
effective in minimizing page reclaim jitter

Page Reclaim by kswapd



memory objects

page cache

free

watermark

freemem threshold

normal state

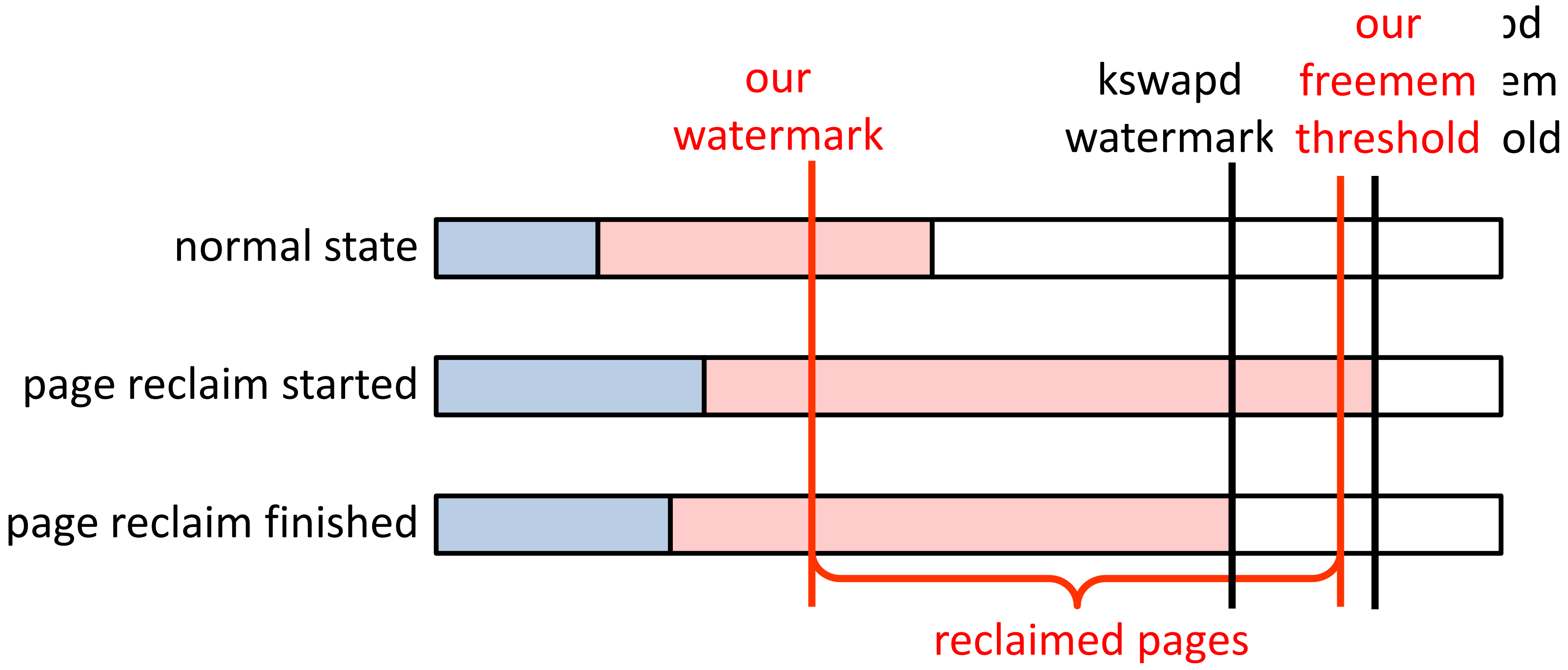
page reclaim started

page reclaim finished

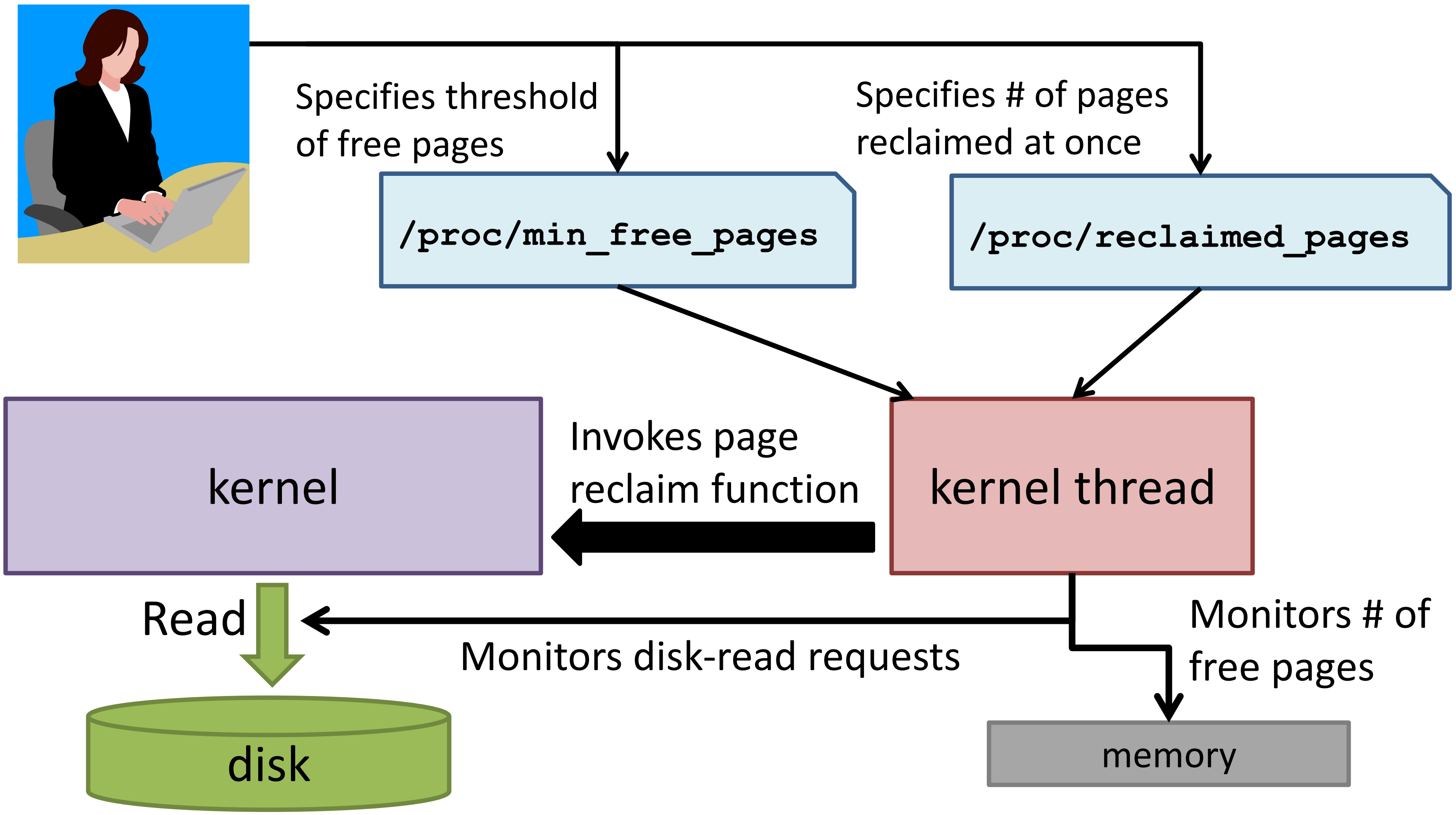
reclaimed pages

Proposed Mechanism (1)

- Introduces new kernel module and kernel thread
  - Starts page reclaim before kswapd
  - Reclaims larger # of pages at once



System Structure



Specifies threshold of free pages

Specifies # of pages reclaimed at once

/proc/min_free_pages

/proc/reclaimed_pages

kernel

Invokes page reclaim function

kernel thread

Read

Monitors disk-read requests

Monitors # of free pages

disk

memory

Proposed Mechanism (2)

- It starts when both conditions are satisfied:
  - Cond. 1: # of free pages < our freemem threshold
  - Cond. 2: our mechanism determines that memory shortage is caused by frequent I/O
- Otherwise, our kernel thread does not start
  - And eventually kswapd will be awakened
    - We expect kswapd will do a good job in minimizing page-outs of memory objects

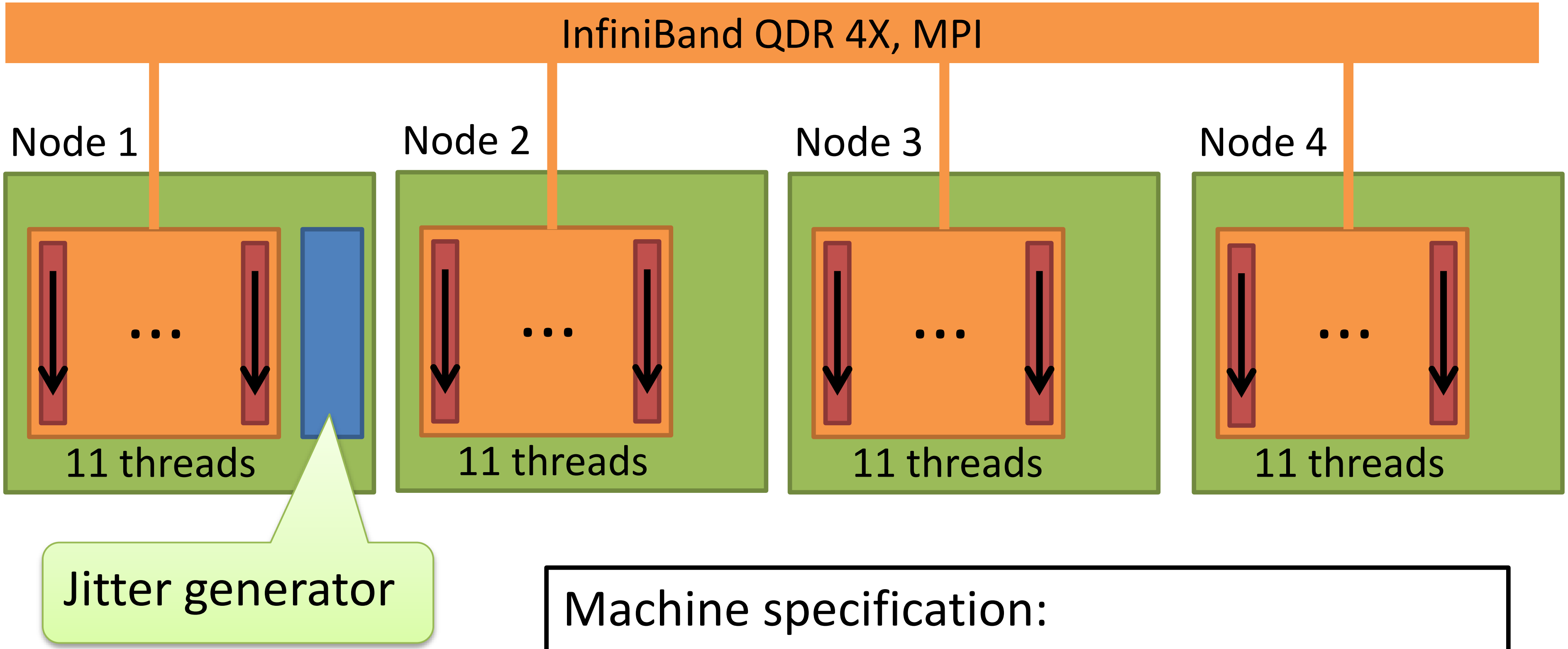
Discussion

- Q: Why introducing a kernel thread, instead of customizing kswapd?
  - Tuning kswapd parameters
  - Modifying kswapd code
- A: Kswapd provides only a few parameters
  - For example, kswapd users cannot directly specify the amount of reclaimed memory
  - But, we would like to investigate a vast space of parameters and algorithms
  - This inconvenience is also pointed out by another Linux engineer: https://lwn.net/Articles/422291/

Experiments

- We measured the impact of jitter on the performance of a scientific application
- Application: WRF (weather forecasting software)
  - Simulated the weather around Japan in one hour (6 s x 600 steps)
- Jitter generator
  - Program that repeatedly reads a 100-GB file sequentially
    - Although it represents an extreme case, we believe that a similar case can possibly occur in some configurations and job sets

Condition

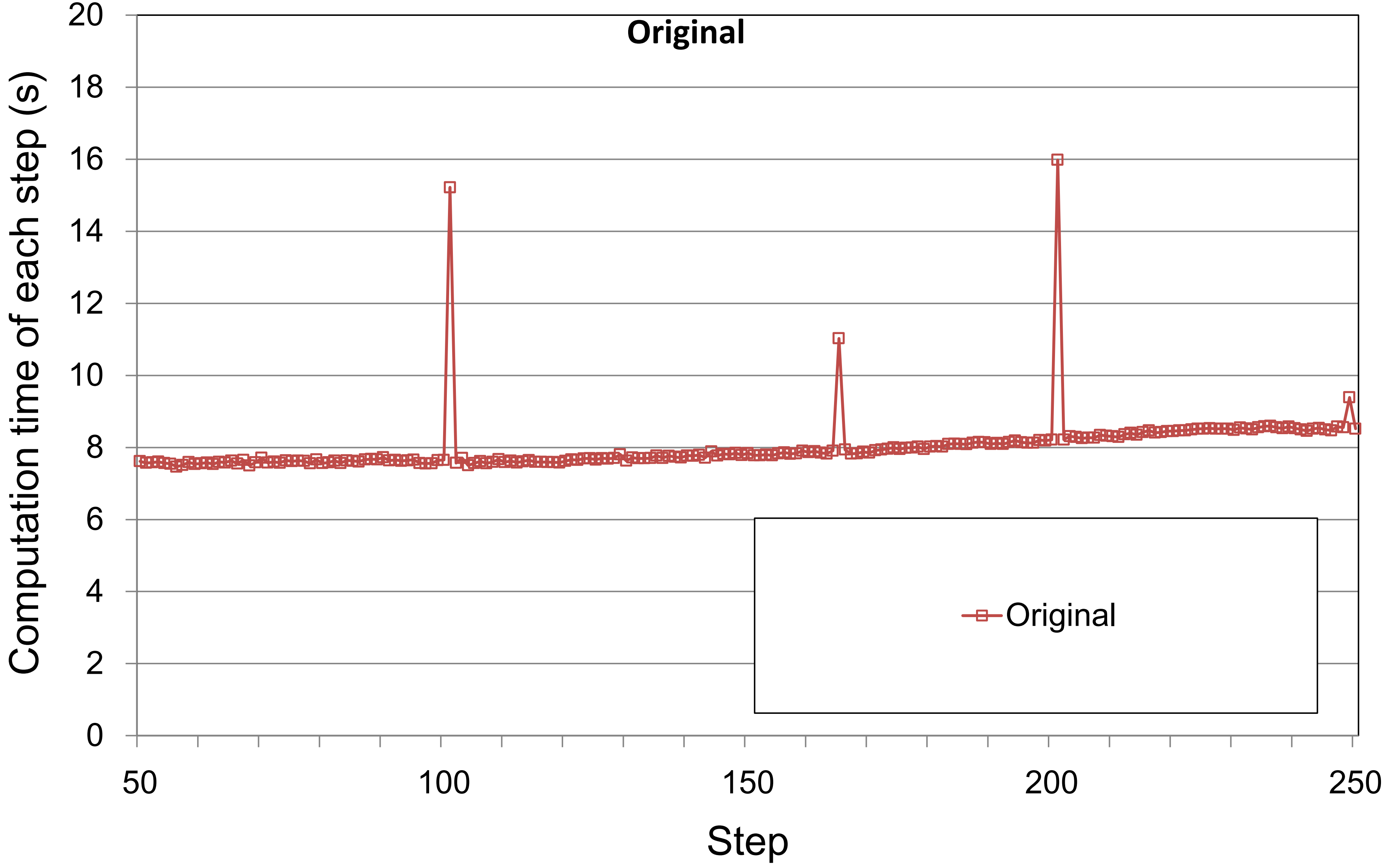


Machine specification:
CPU: Intel Xeon E5645 2.4 GHz (6 cores) x 2
Memory: 48 GB
HDD: SAS 15,000 rpm

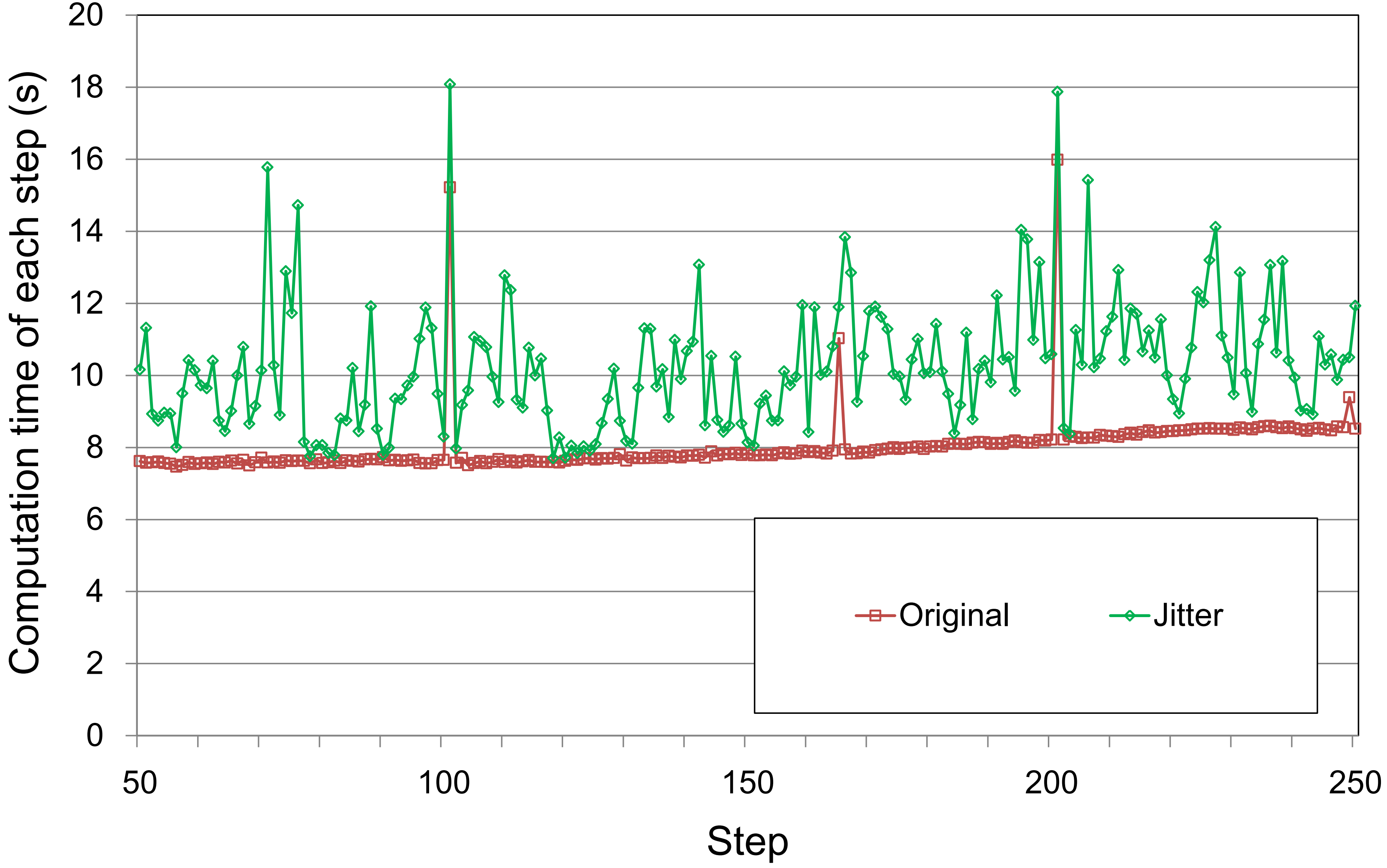
Experiment 1

- We compared WRF performance in 3 cases
  - Original
  - With jitter
  - With jitter and proposed mechanism (Jitter+Proposed)

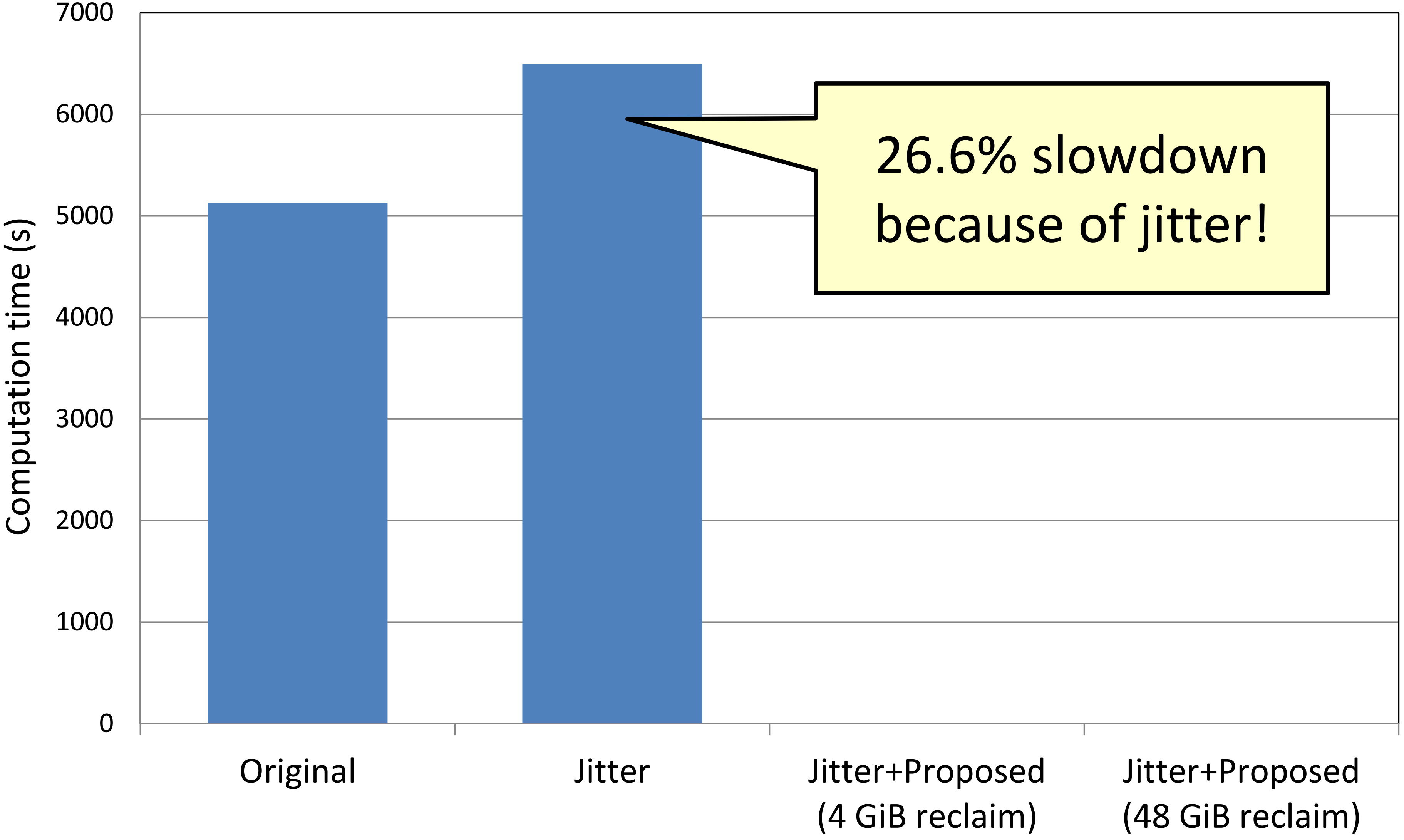
Result
(Not Using Proposed Mechanism)



Result (Not Using Proposed Mechanism)

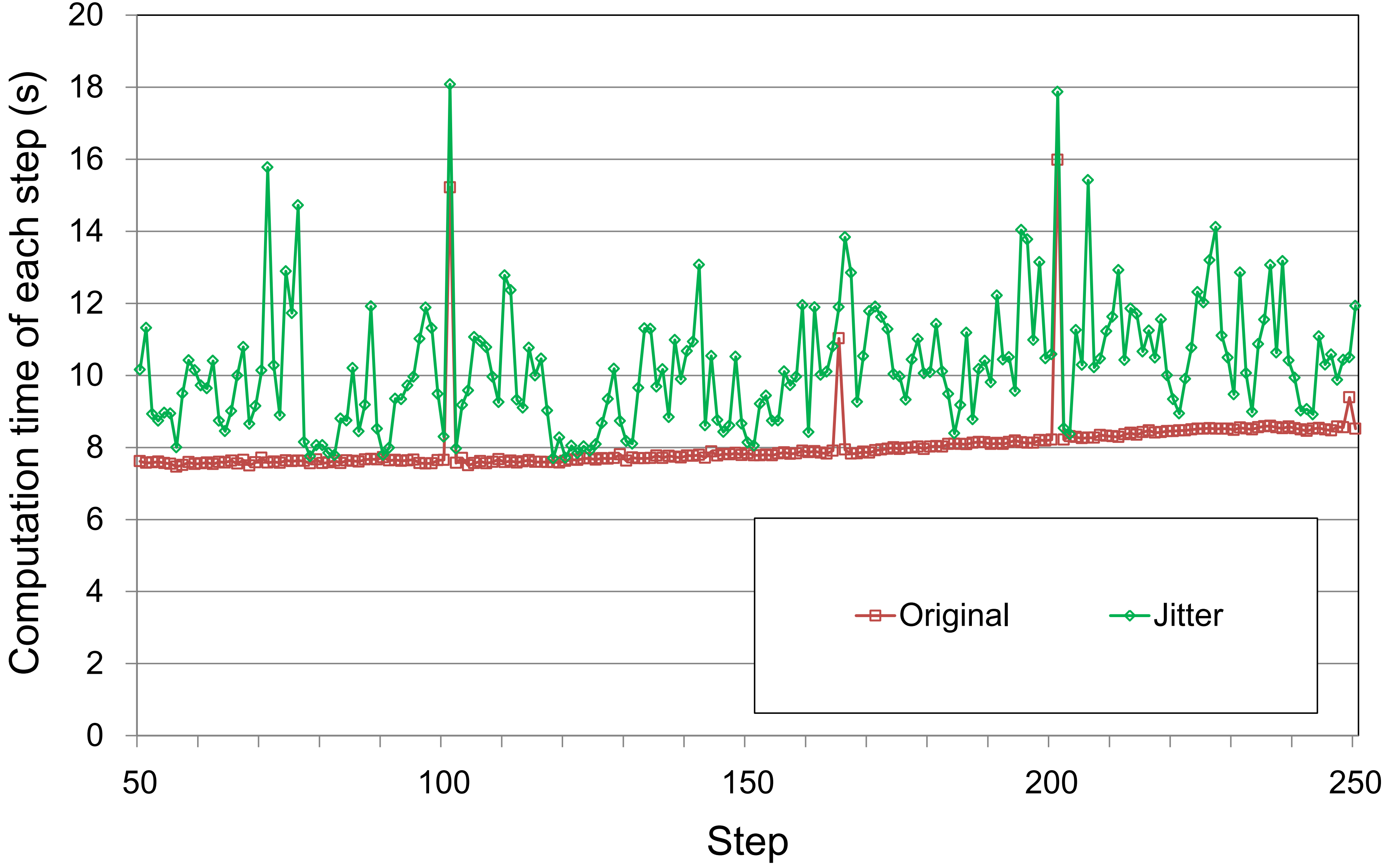


Accumulated Computation Time (Not Using Proposed Mechanism)

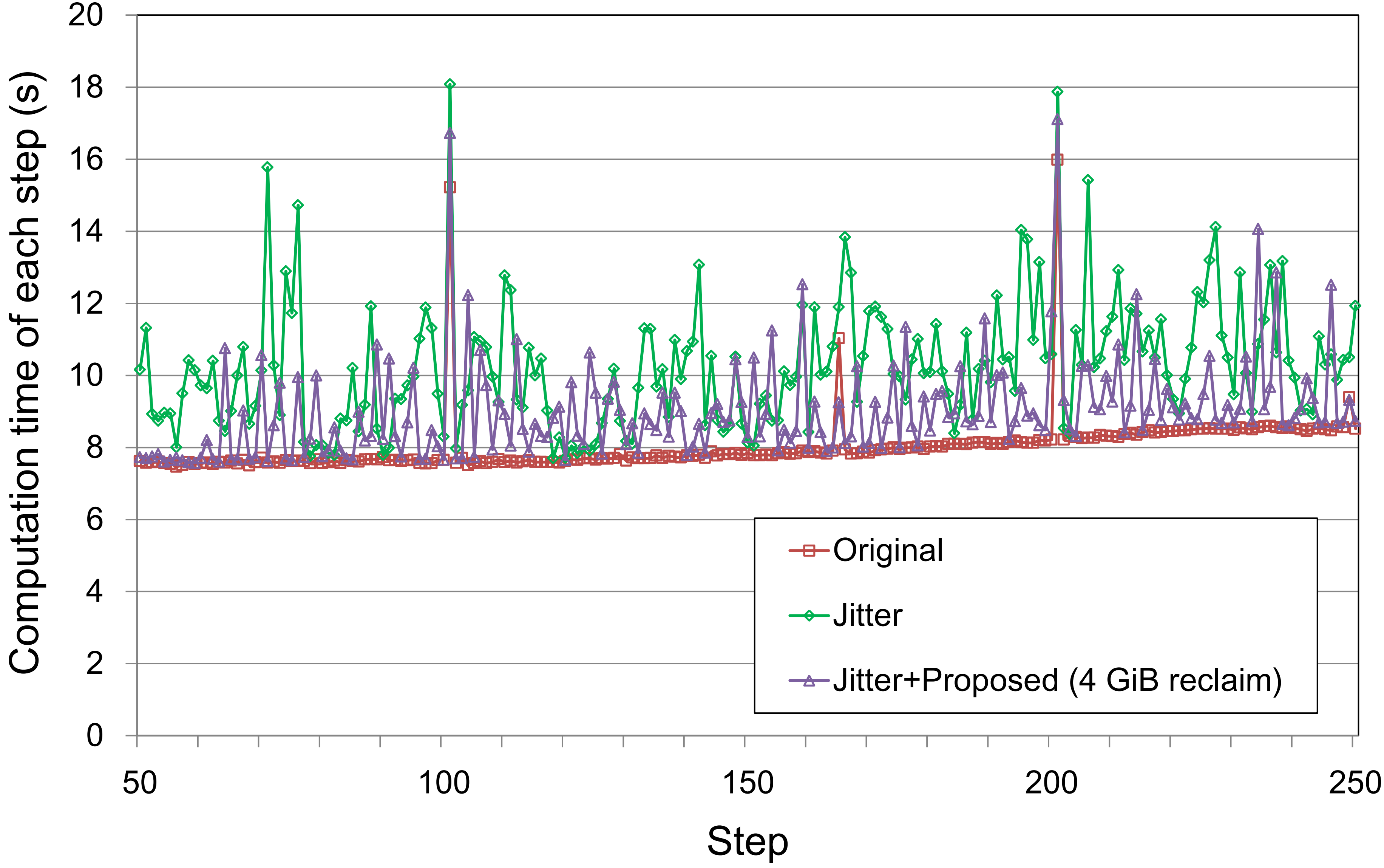


26.6% slowdown because of jitter!

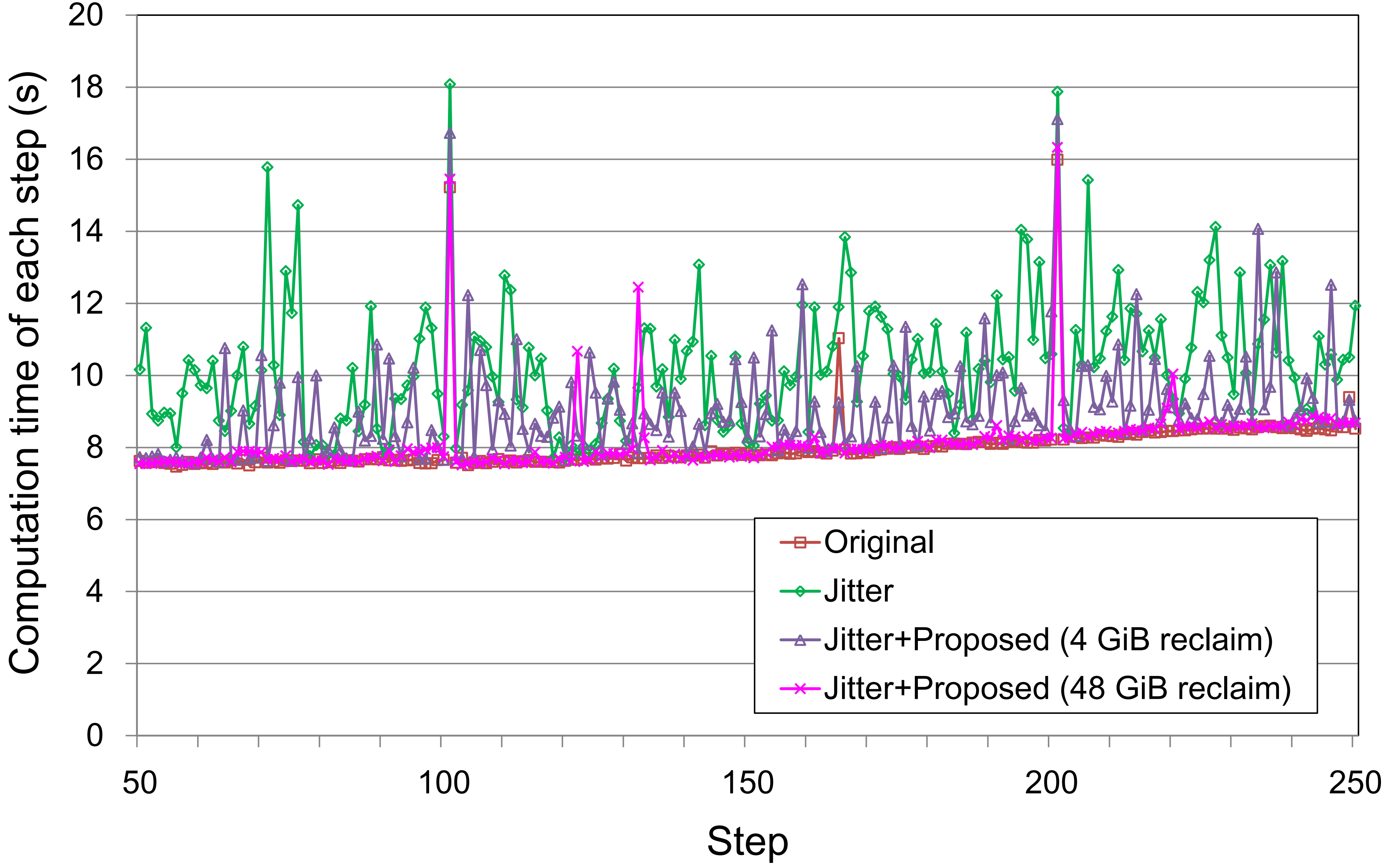
Result
(Using Proposed Mechanism)



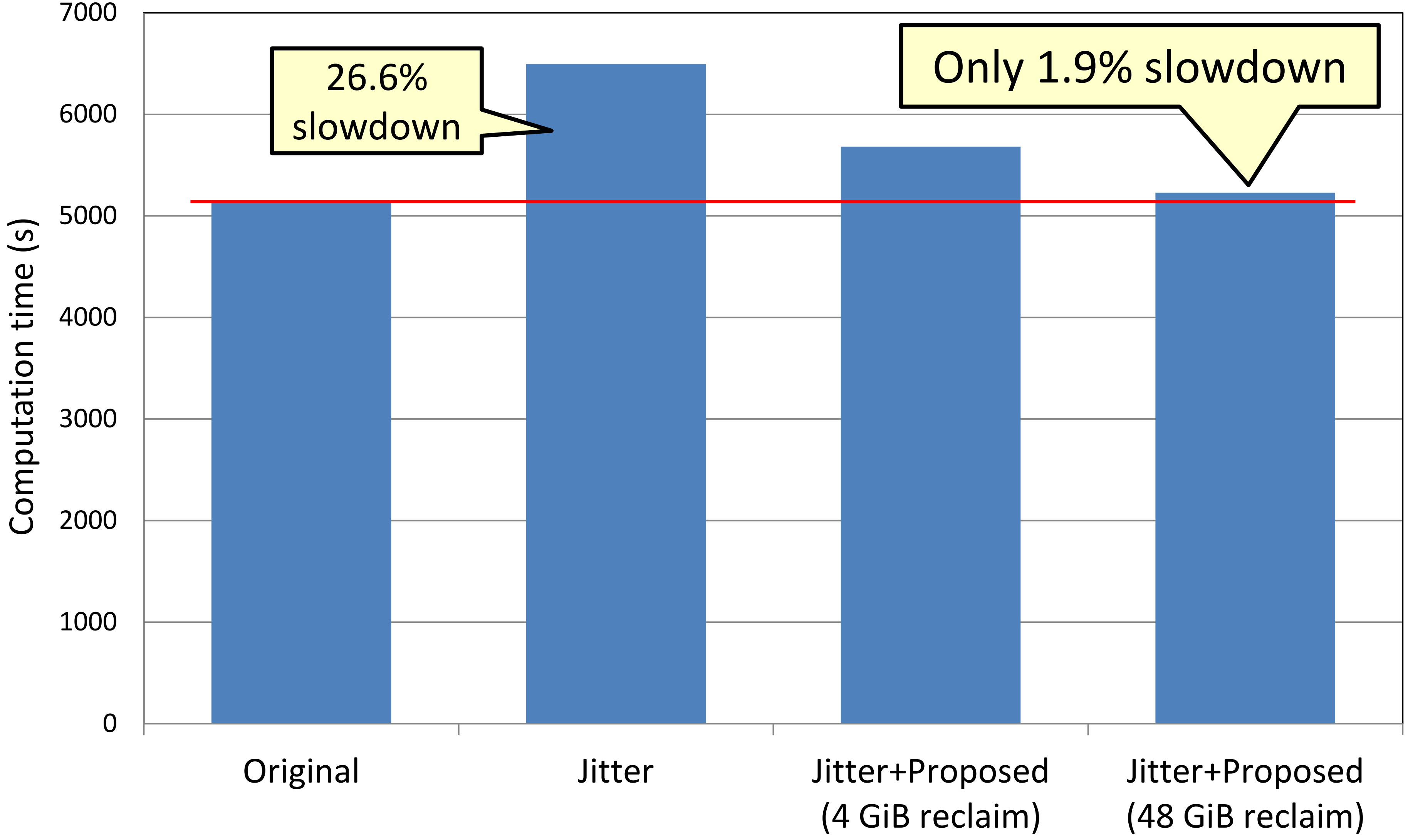
Result
(Using Proposed Mechanism)



Result
(Using Proposed Mechanism)



Accumulated Computation Time (Using Proposed Mechanism)



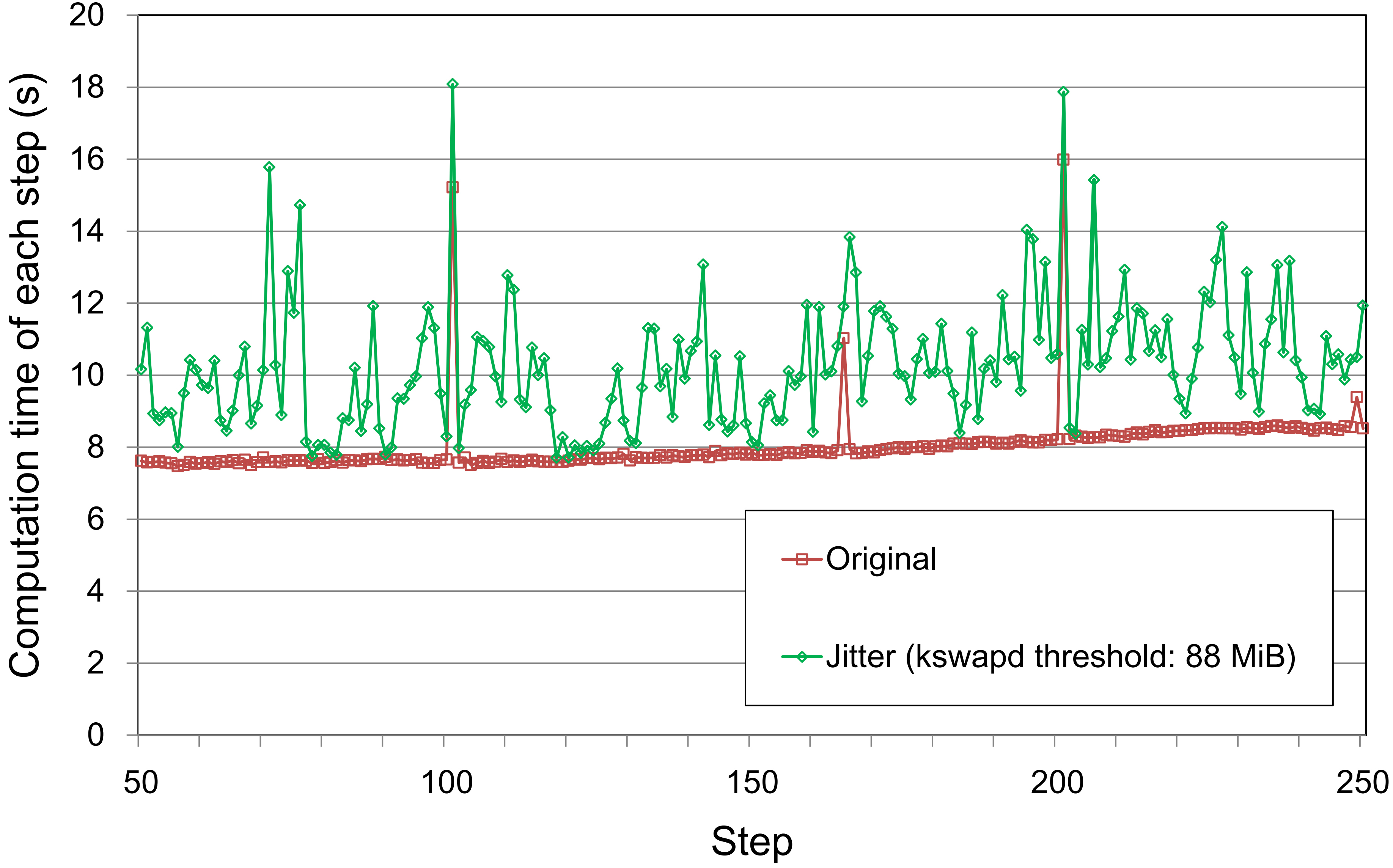
26.6% slowdown

Only 1.9% slowdown

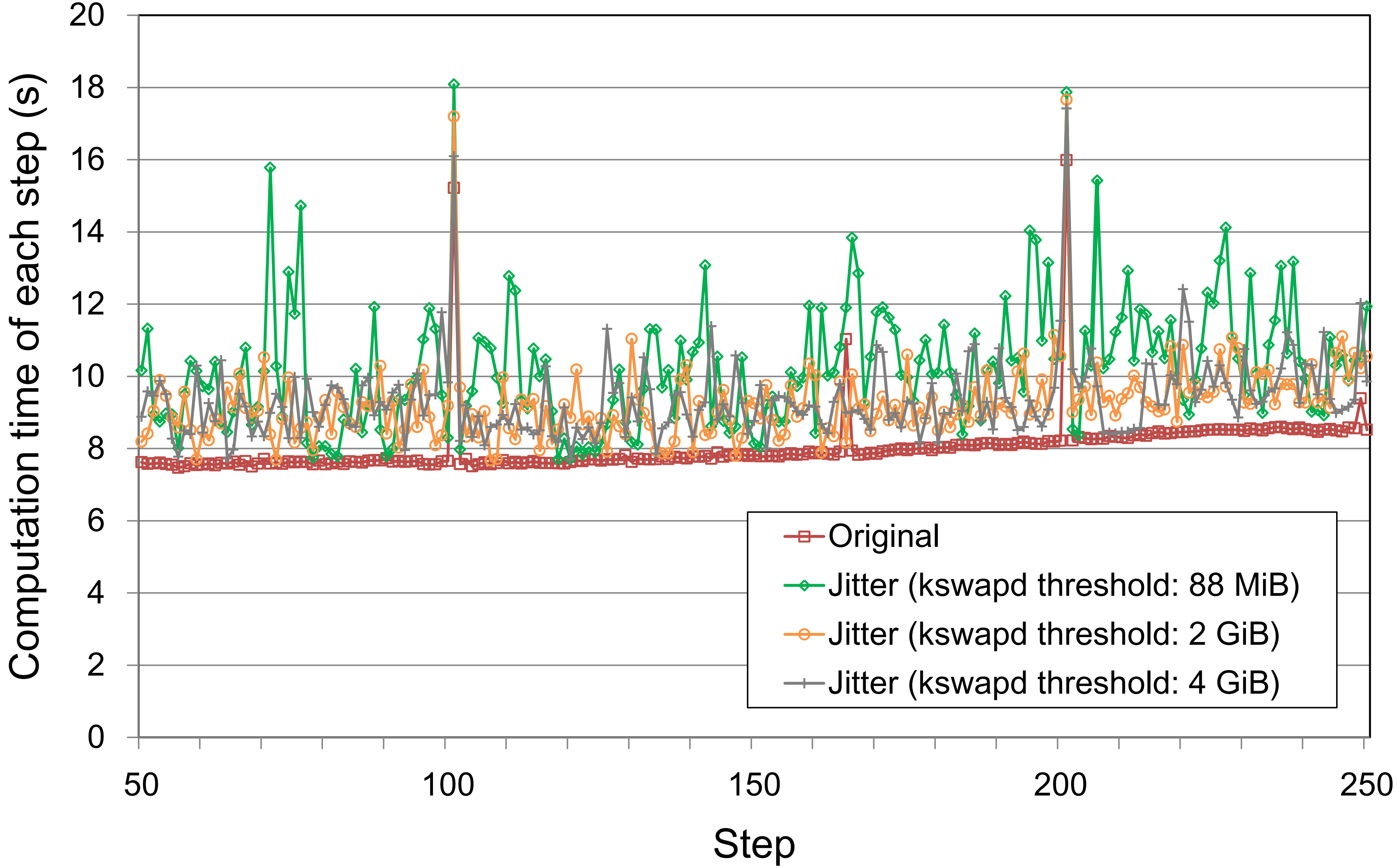
Experiment 2

- In addition, we must answer
  - “How good performance can we get by changing parameters of kswapd?”
  - “Is kswapd parameter tuning sufficient to obtain comparative performance?”
- We measured WRF performance in Jitter case with various kswapd parameters

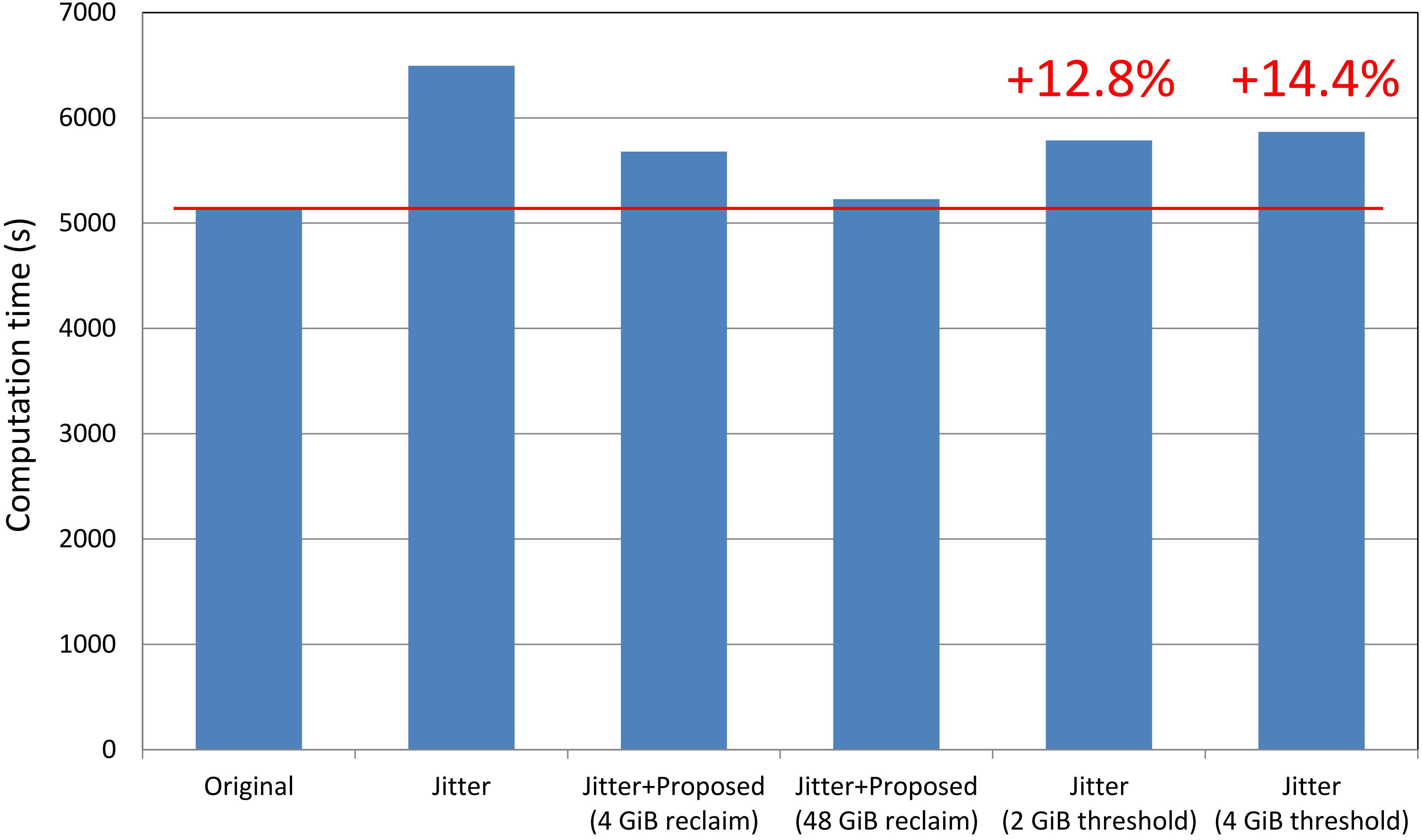
Effect of kswapd Parameter Changes



Effect of kswapd Parameter Changes



Effect of kswapd Parameter Changes



Related Work

- “Core separation” approaches
  - [De et al. IPDPS 2009], [Oral et al. 2010], [Rosenthal et al. 2013], [Seelam et al. IPDPS 2011]
  - Executes the kernel and daemons on dedicated CPU cores
  - Executes applications on remaining CPU cores
  - Prevents the kernel and daemons from depriving applications of CPU resources

  It is unclear how many CPU cores are sufficient for hosting kswapd threads and other system tasks

  - Their approach should be combined with another approach for reducing the impact of jitter

Summary and Future Work

- Summary
  - We proposed a mechanism for reducing the impact of jitter caused by page reclaim
  - Jitter caused by an I/O-intensive process increased the execution time of WRF by 26.6%
  - The mechanism lowered the increase to 1.9%
- Future Work
  - Understanding jitter caused by reading many small files or by writing to a file
  - Improving the proposed mechanism in order to monitor accesses to files on remote I/O nodes
  - Analyzing the experimental results in more detail