ON-DEMAND-FORK: A Microsecond Fork for Memory-Intensive and Latency-Sensitive Application

Kaiyang Zhao, Sishuai Gong, Pedro Fonseca,

In Proceedings of the Sixteenth European Conference on Computer Systems (EuroSys '21)

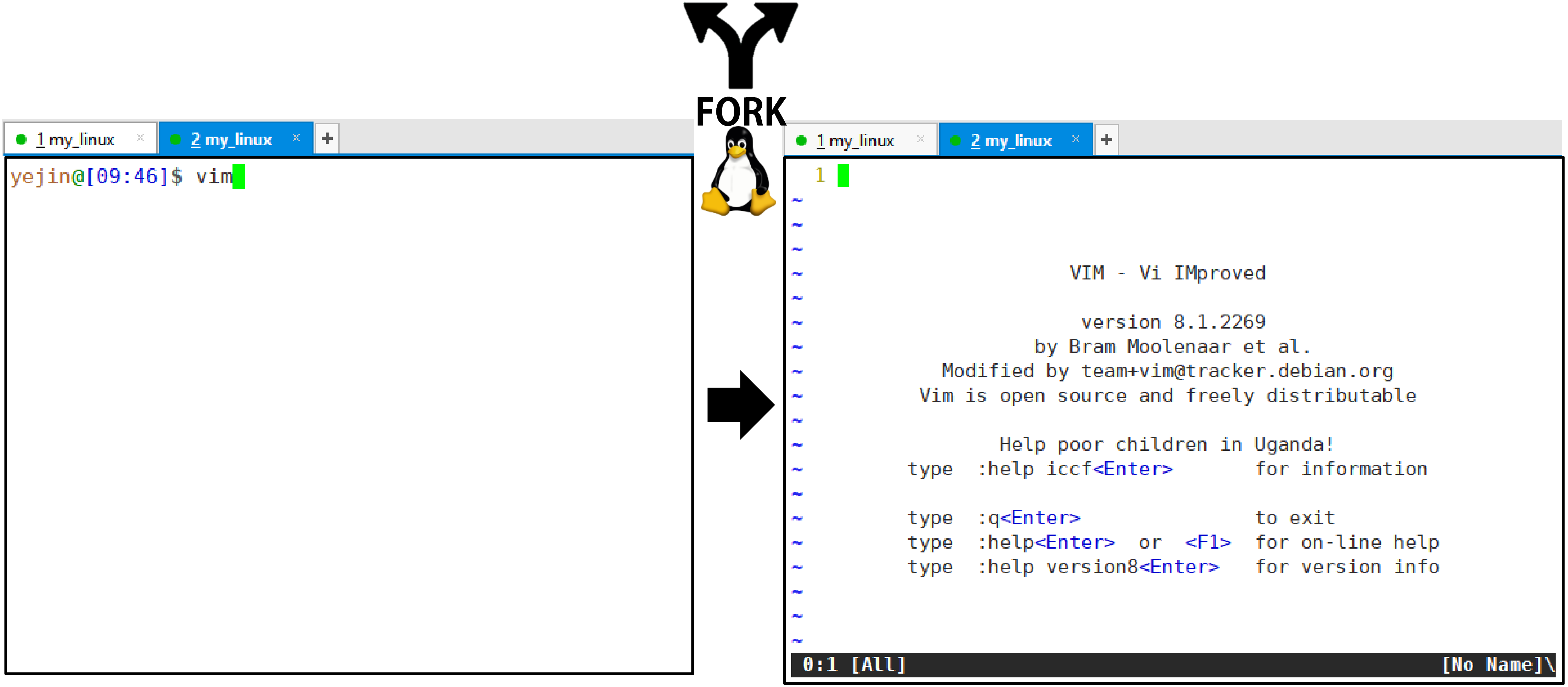
2022. 01. 18

Presentation by Han, Yejin

hyj0225@dankook.ac.kr

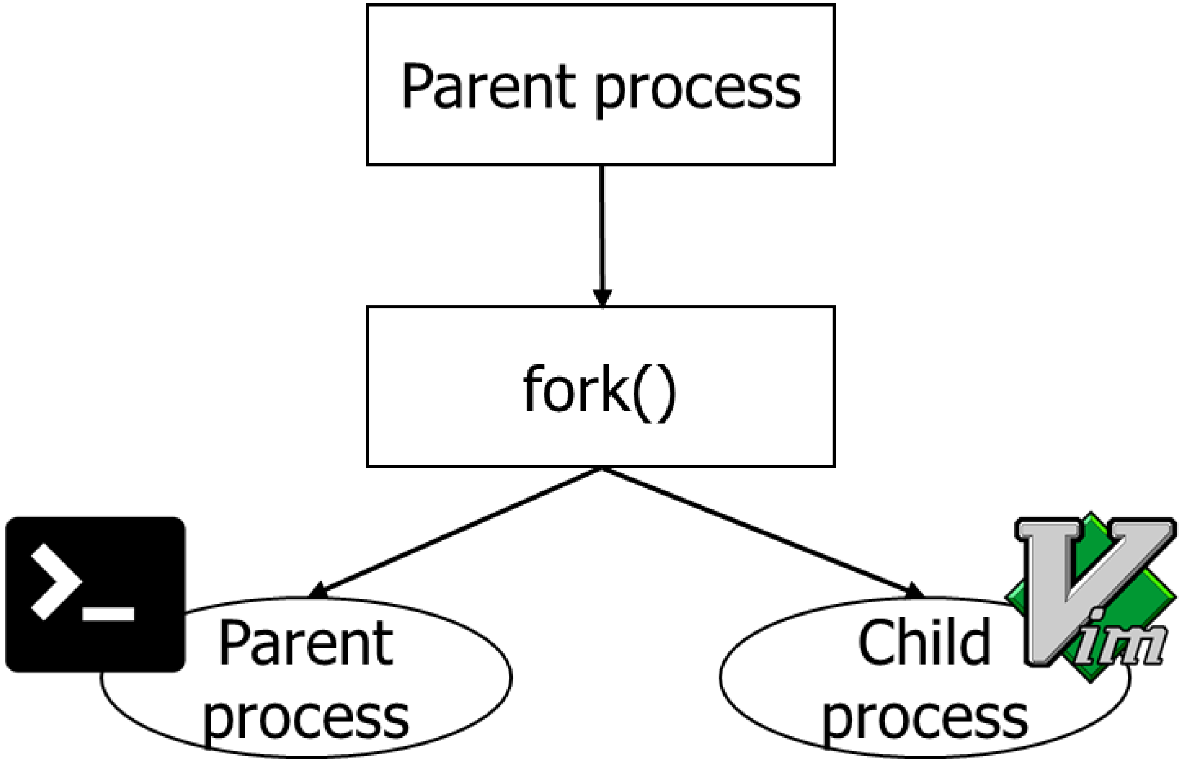
Contents

1. Introduction



What is fork system call?

- Creates a child process by **duplicating** the calling process
- Memory image is the same as the existing process



Where is fork system call used?

K-V stores

Redis

(a few **TBs**)

Serverless

Firecracker

(a few **GBs**)

Fuzzers and testing

ClusterFuzz

(Hundreds of **MBs**)

Latency problem of fork system call

- Fork gets slower as memory gets larger
- Latency of each fork call deteriorate under concurrency

```c
for (int i = 1; i <= 120; i++) {
  size_t size = i * (1024 * 1024 * 512);
  void *buffer = mmap(NULL, size, ...);
  clock_gettime(..., &ts1);
  pid=fork();
  switch (pid) {
    case 0:     /* child */
      return 0;
    default: /* parent */
      clock_gettime(..., &ts2);
      print_cputime(ts1, ts2, size);
      waitpid(-1, NULL, 0);
  }
  munmap(buffer, size);
}
```

Figure 1. Fork benchmark program.

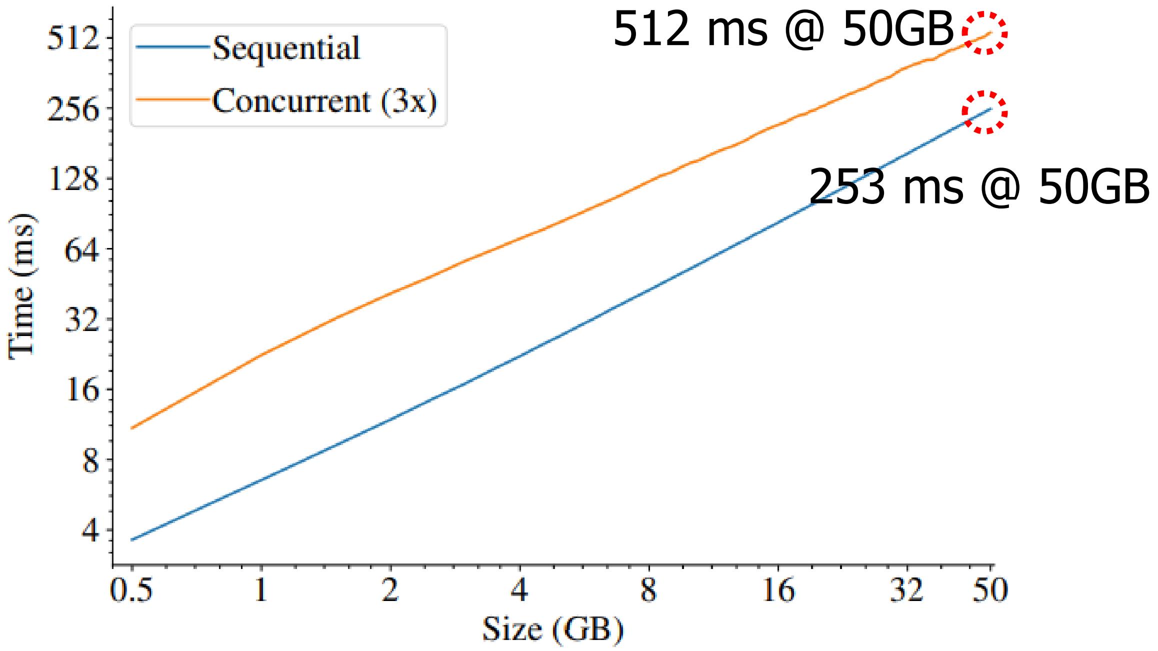
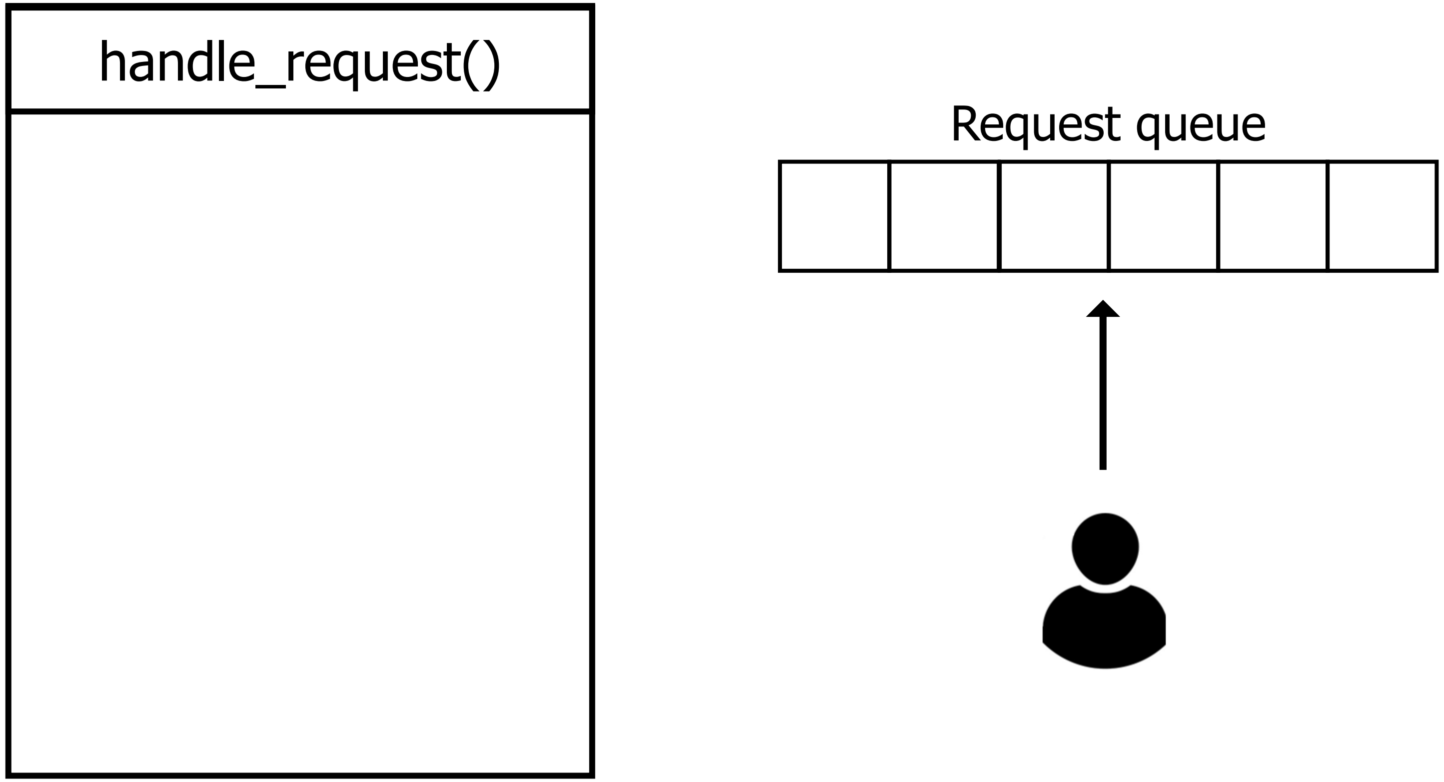


Figure 2. Fork execution time with different memory allocation sizes. Results include measurements with sequential executions and concurrent executions (with 3 concurrent instances of the benchmark). Tests ran on a 16-core machine and were repeated 5 times. Each run allocated a memory buffer that varied from 512 MB to 50 GB in 512 MB increments.

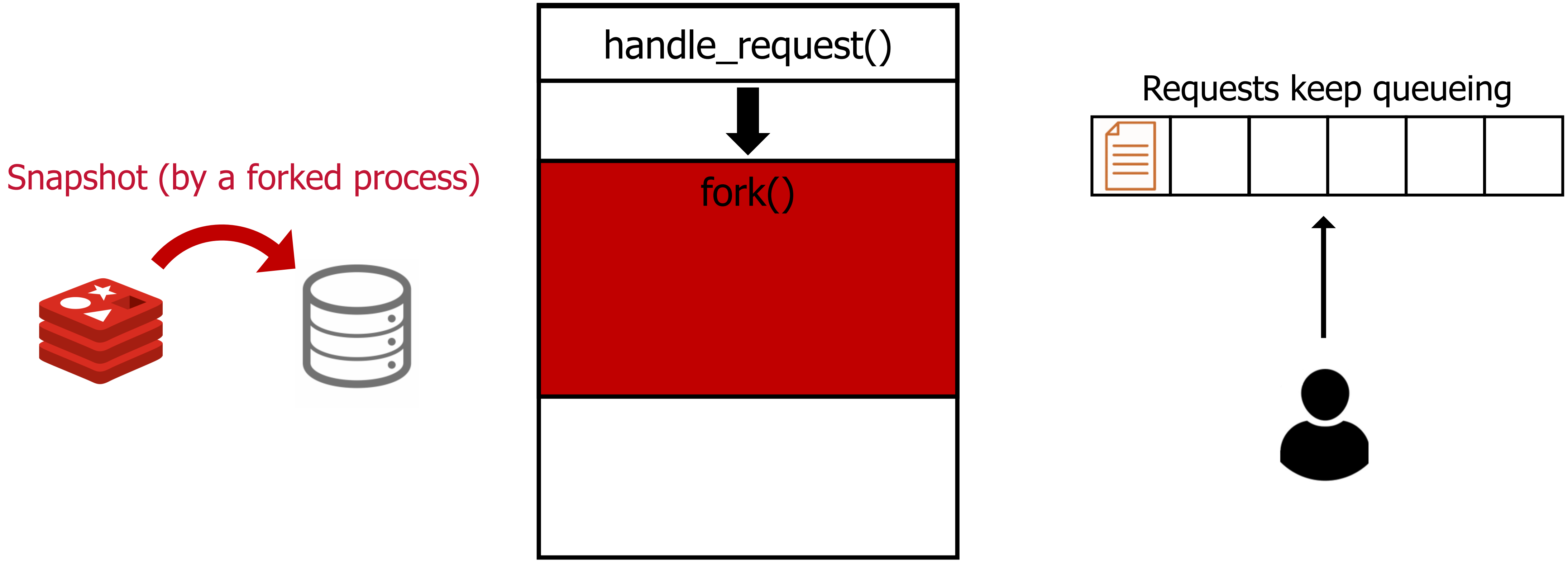
What is the impact of slow fork system call?

- Redis is an in-memory K-V store and uses fork to perform snapshots
- During fork syscall, the parent process is unable to serve any requests



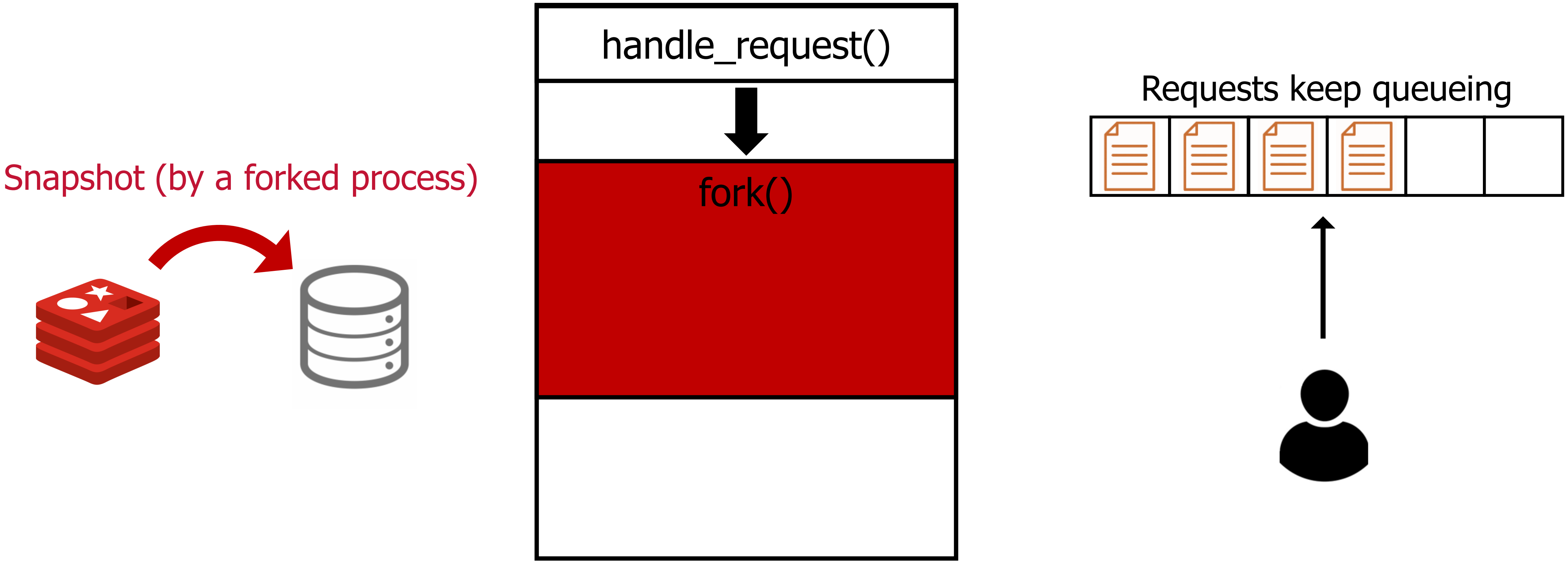
What is the impact of slow fork system call?

- Redis is an in-memory K-V store and uses fork to perform snapshots
- During fork syscall, the parent process is unable to serve any requests



What is the impact of slow fork system call?

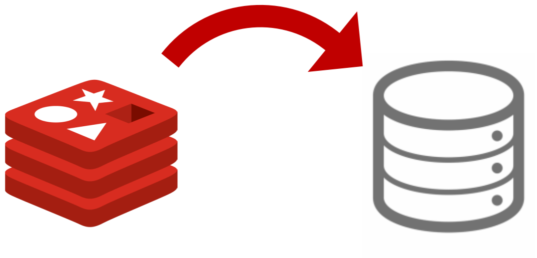
- Redis is an in-memory K-V store and uses fork to perform snapshots
- During fork syscall, the parent process is unable to serve any requests

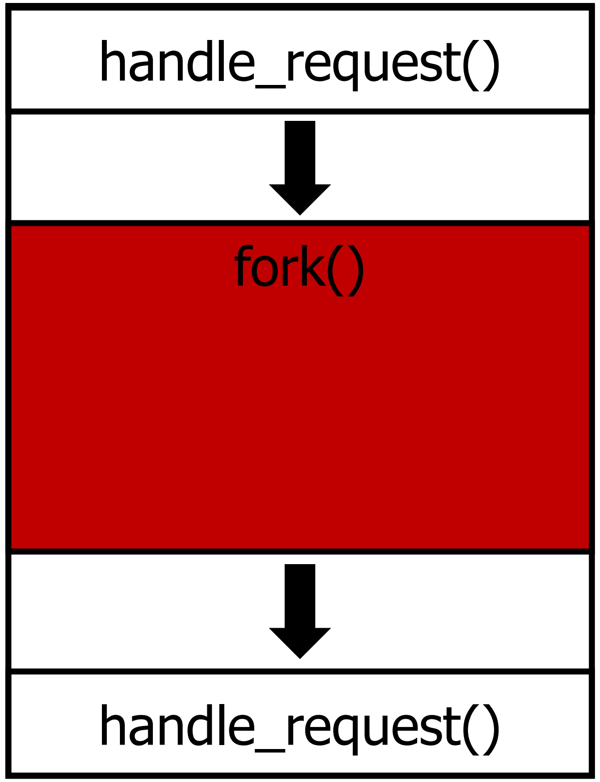


What is the impact of slow fork system call?

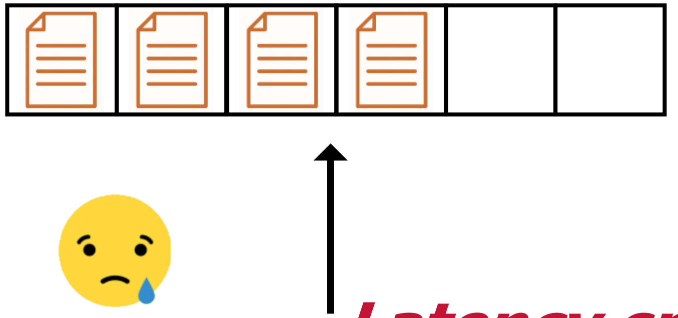
- Redis is an in-memory K-V store and uses fork to perform snapshots
- During fork syscall, the parent process is unable to serve any requests

Snapshot (by a forked process)





Requests keep queueing

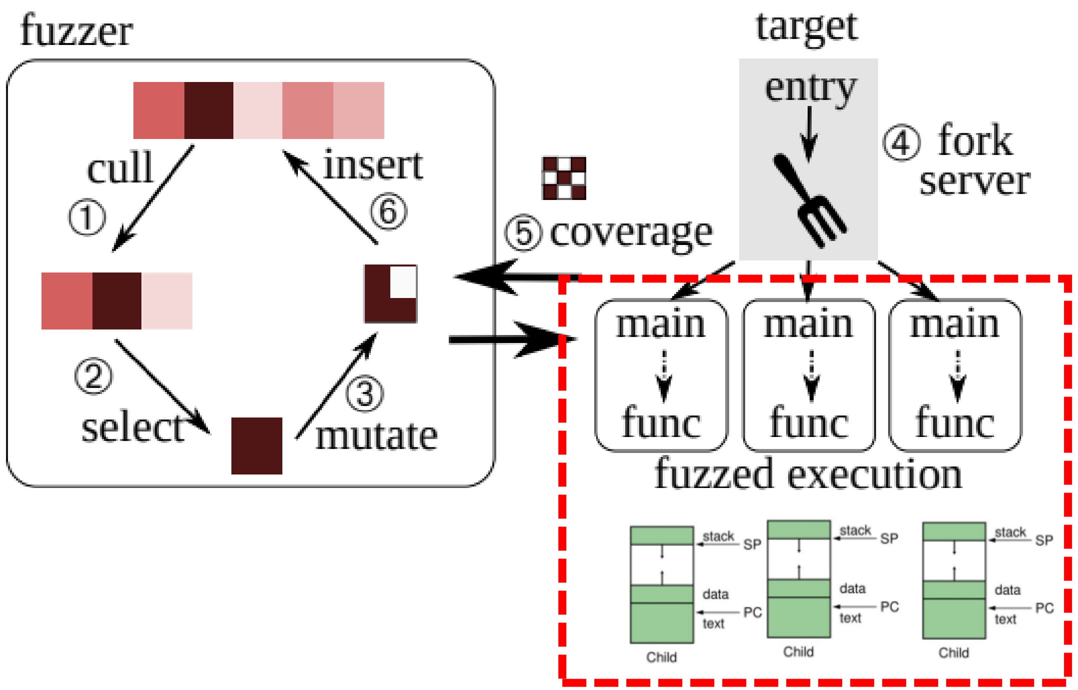


Percentile	ms
	Fork
≥50%	4.319
≥90%	5.247
≥95%	5.343
≥99%	5.695
≥99.9%	6.335
≥99.99%	16.255

Latency spikes

Efficiency problem of fork system call

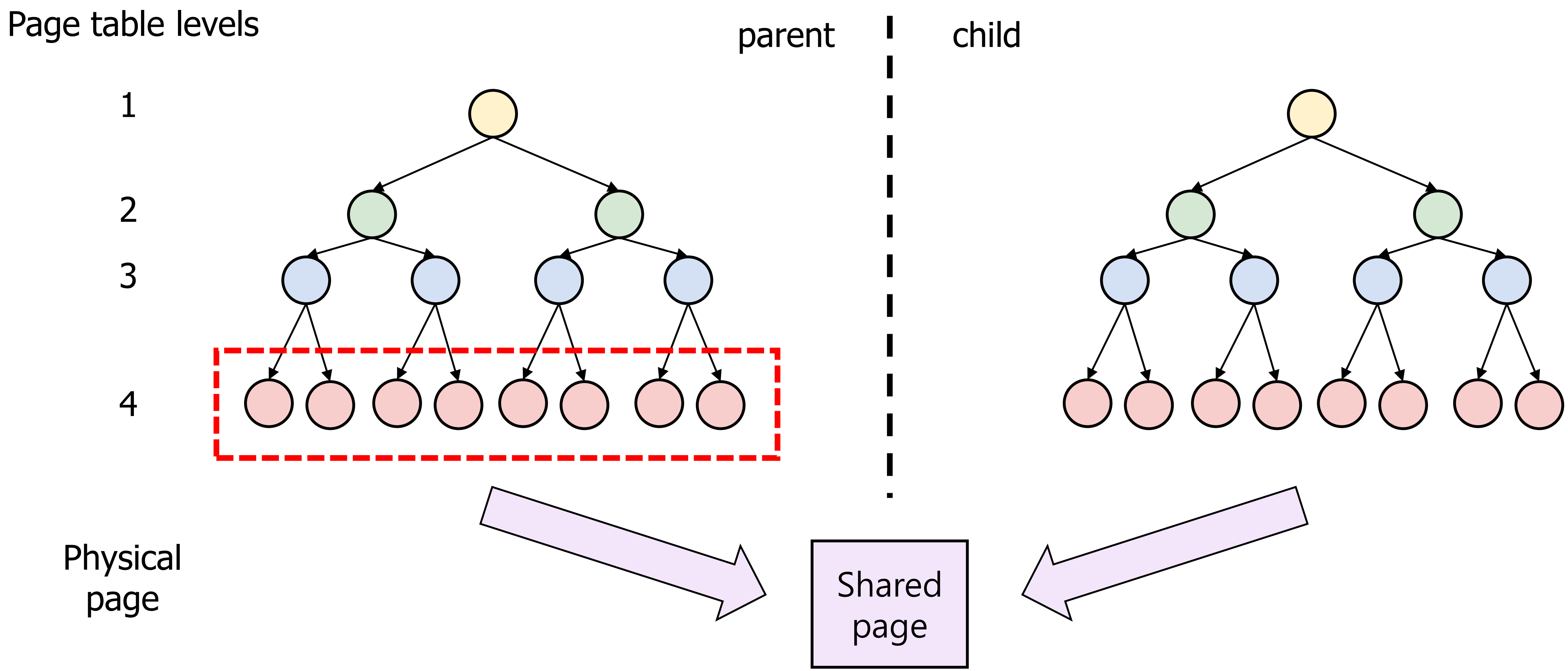
- Fork sets up the entire address space of the child process
- But some applications only access a small portion of the memory in the child process
- Ex, when an application is being fuzzed



Setting up the whole address space is wasteful

(Source: WINNIE: Fuzzing Windows Applications with Harness Synthesis and Fast Cloning)

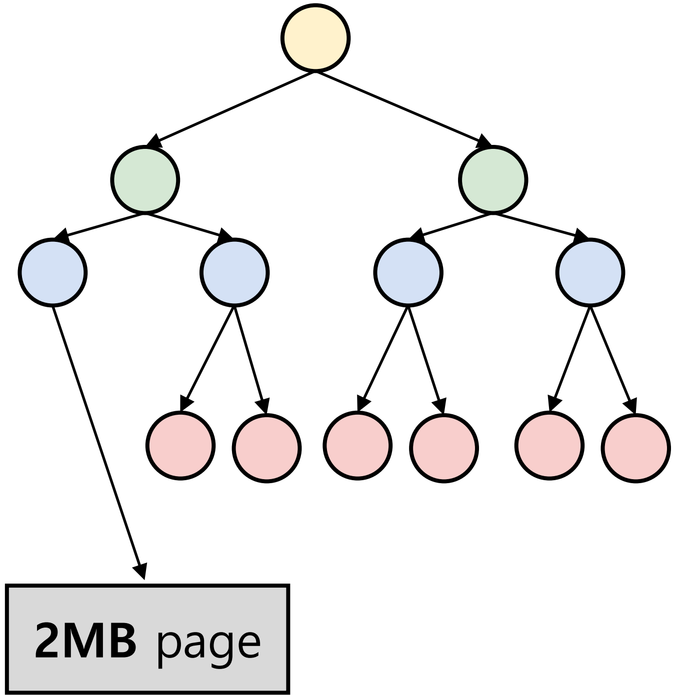
Why is fork slow and inefficient?



Copying is very expensive for large applications

What about huge pages?

- Fewer pages: fewer page tables to copy
- Lower the latency of fork, but suffer from other factors



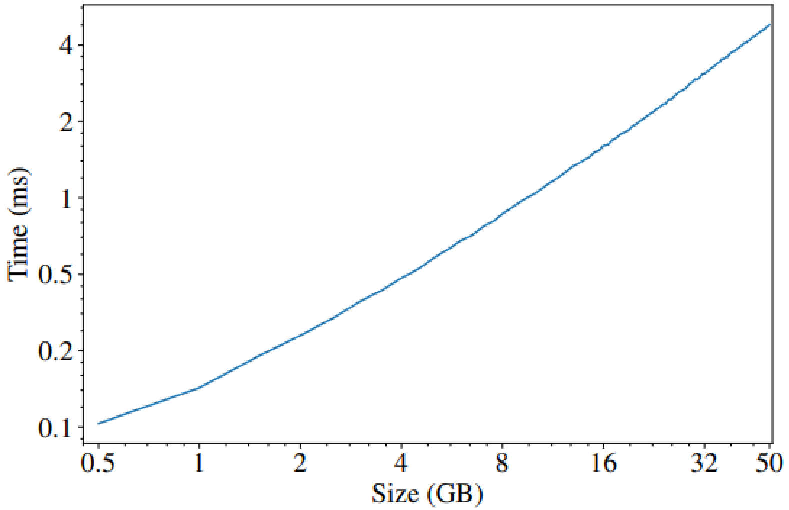


Figure 4. Time to fork vs. size of memory allocated with huge pages. Size is in 512 MB increments. Only 1 instance of the benchmark program ran. Tests ran on a 16-core machine and were repeated 5 times.

- Increased internal fragmentation
- Expensive page faults
- System-wide latency spike

On-demand-fork overview

- Shares last level page tables across processes during fork
- No issues of huge pages

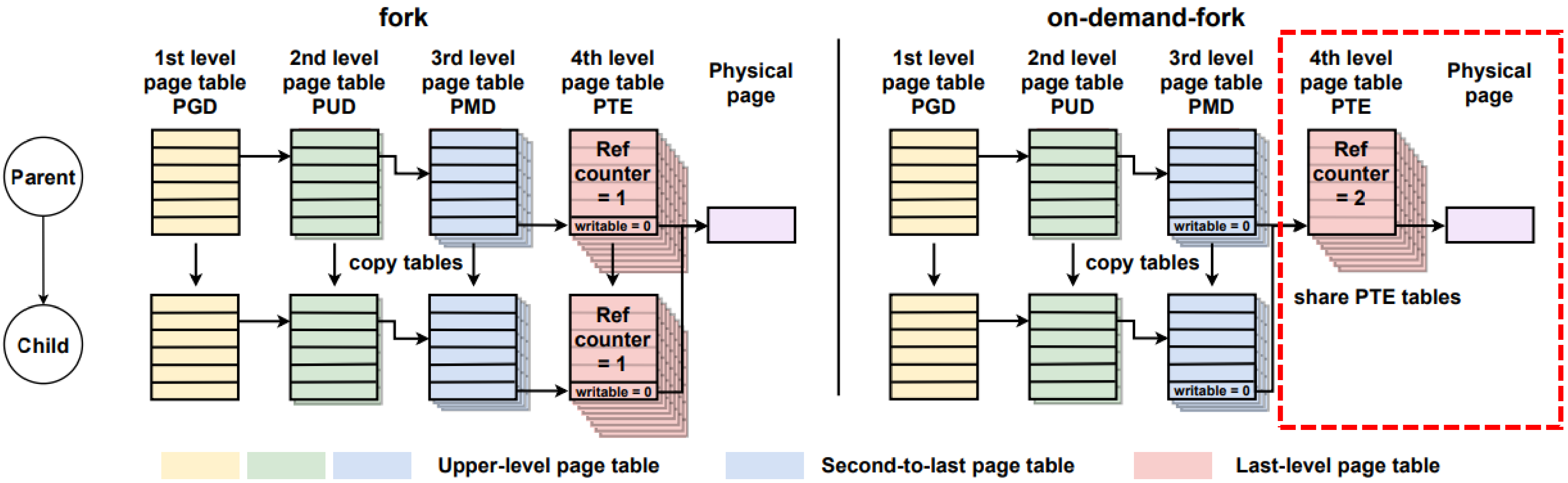
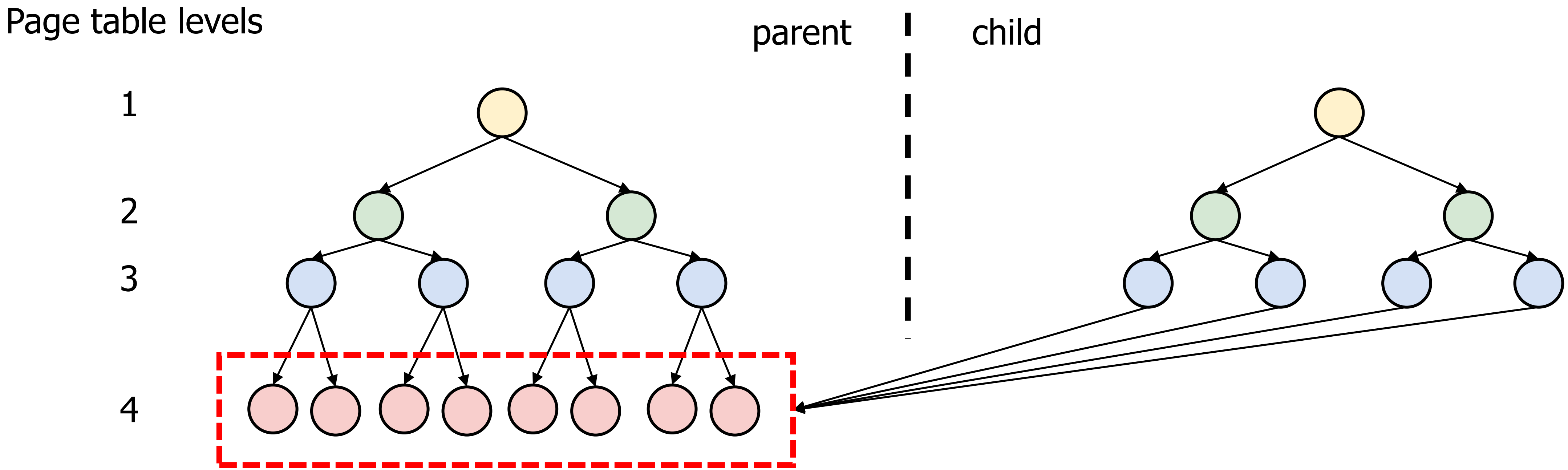


Figure 5. FORK and ON-DEMAND-FORK page table management comparison. Unlike FORK, ON-DEMAND-FORK shares the last level page tables across processes, which represent the vast majority of the page table structure.

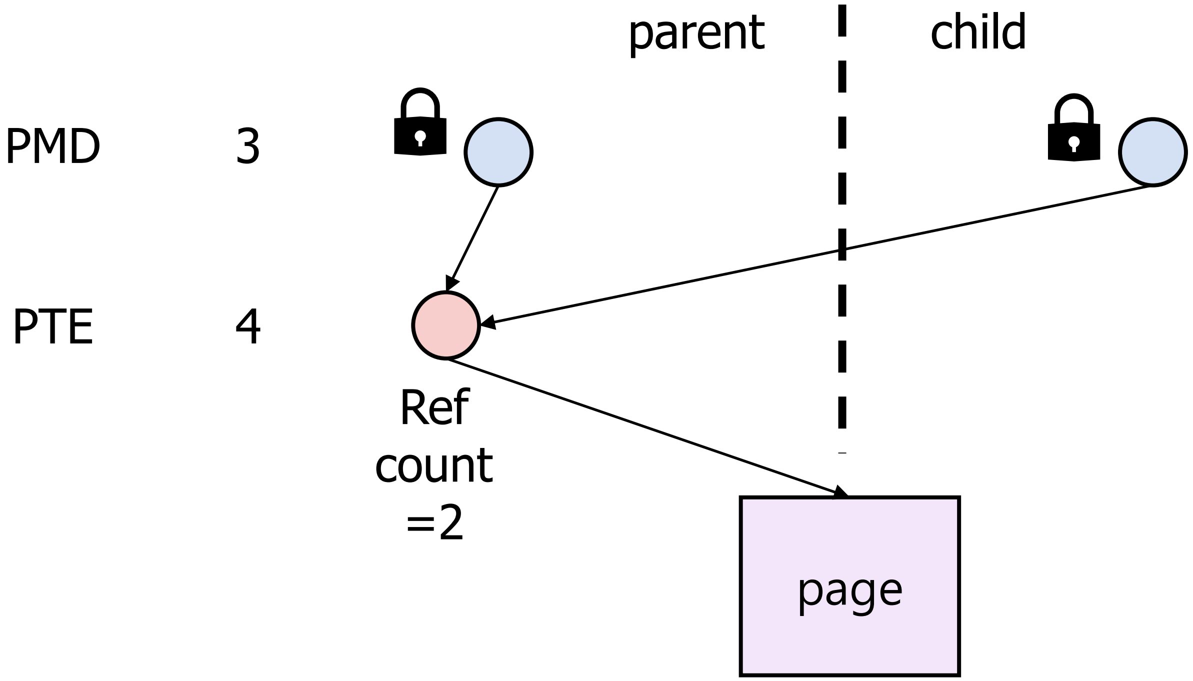
Fast read after fork

- No cost of copying page tables for read access
- If a process reads from memory mapped by a shared page table, virtual memory translation is done without triggering page fault



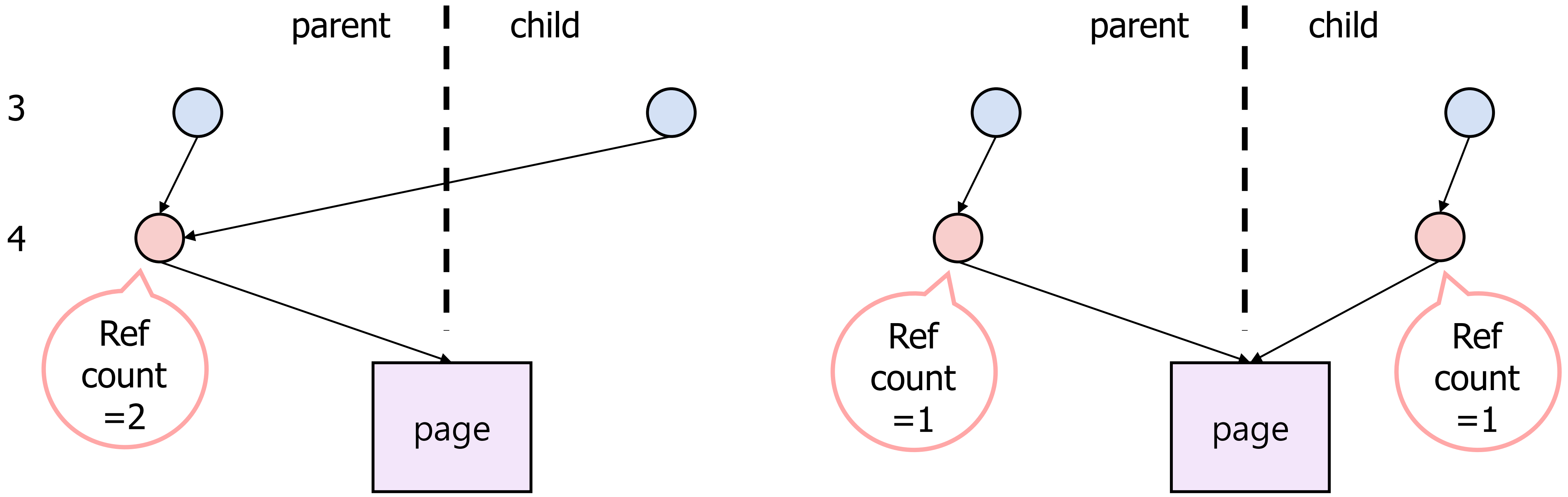
Preserving copy-on-write

- On-demand-fork disables the write permission in 3rd level tables
- Increments the reference counter of the parent's PTE tables and child's PMD entries point to them



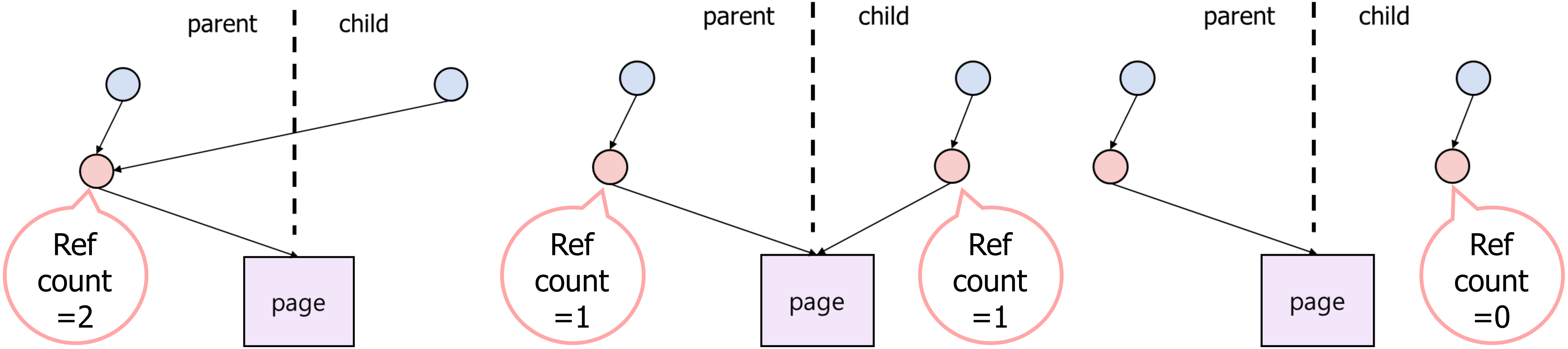
On-demand page table copying

- Page faults for write access only
- Increased cost for only the first write access



Keeping track of shared tables

- Challenge: Need to know when to free last-level page tables
- Solution: reference counts last-level page tables
- Last-level page tables are freed after count reaches zero



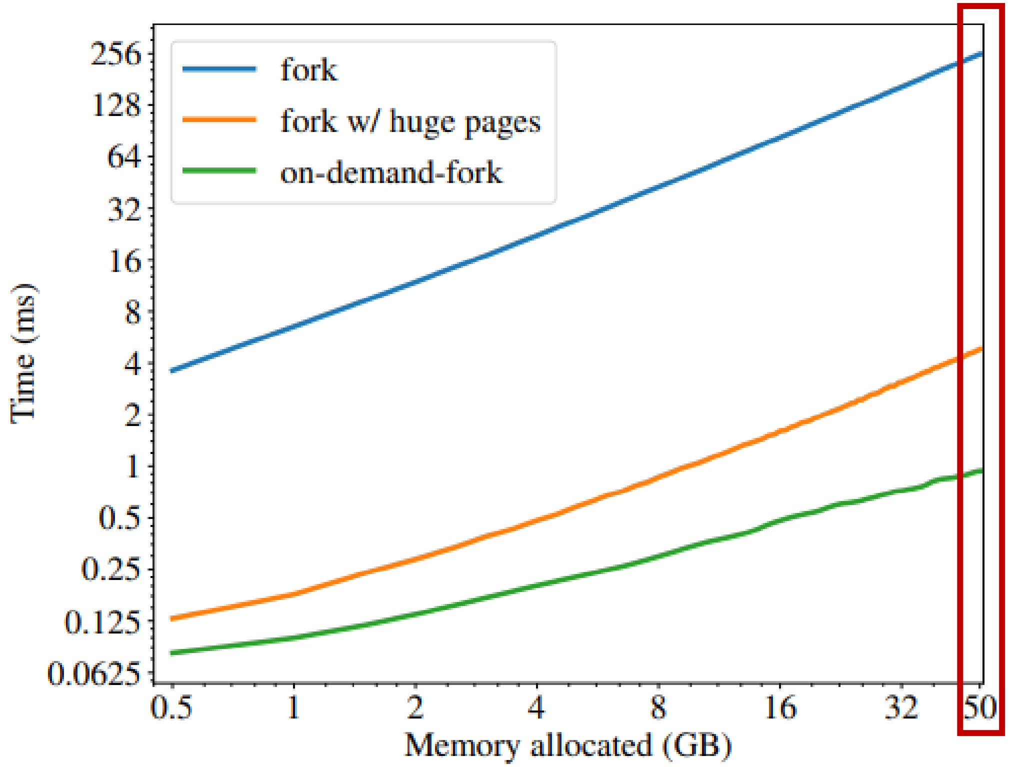
During system call **On-demand copying** **Unmapping**

Evaluation Environment

CPU	16-core AMD EPYC 7302P CPU
Memory	256 GB RAM
OS	64-bit Linux 5.6.19
Workload	Microbenchmarks/Real-world Applications

Microbenchmarks

- **System call latency**



270 times faster at 50 GB
Faster than huge pages

- **Page fault handling**

Type	Avg. time (ms)	
FORK	0.0023	
FORK w/ huge pages	0.1984	→ 86.3x
ON-DEMAND-FORK	0.0122	→ 5.3x

Table 1. Worst-case cost to handle a page fault using FORK, with regular and huge pages, and ON-DEMAND-FORK. The results are the average of 10 runs.

Worst-case page fault handling time is reasonable

Real-world Applications

- **Fuzzing: AFL**

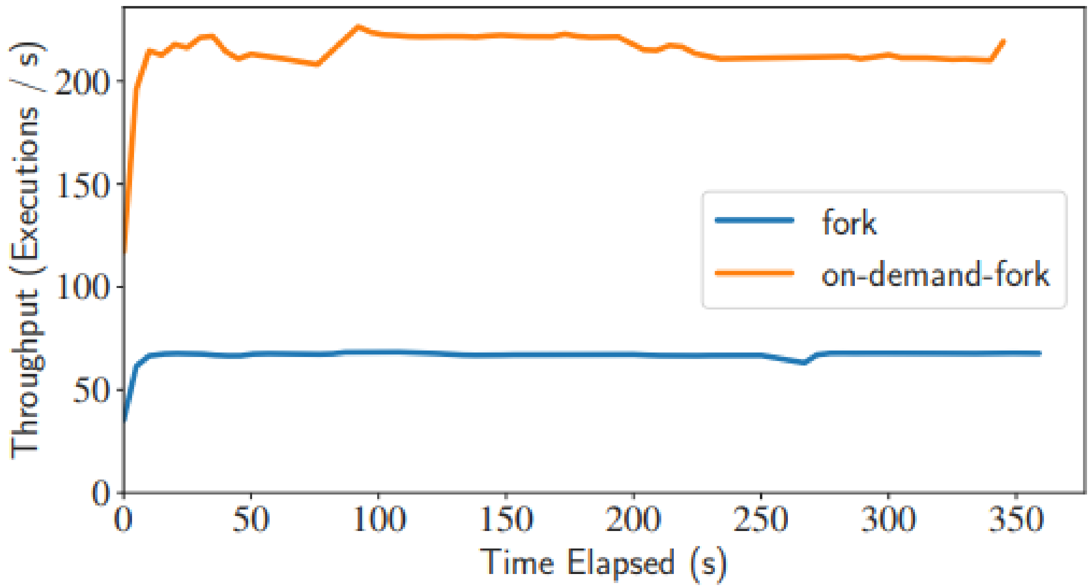
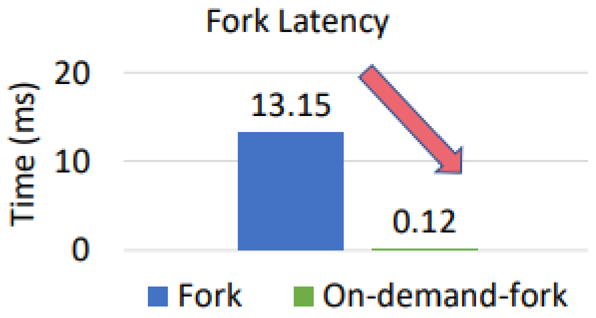


Figure 9. Execution throughput of AFL on SQLite over the duration of a test campaign. Results show the throughput with FORK and ON-DEMAND-FORK.

2.26x throughput increase

- **Unit Testing: SQLite**



99.1% shorter than FORK

Phase	FORK	ON-DEMAND-FORK
Forking	13.15 (98.6%)	0.12 (36.4%)
Testing	0.18 (1.4%)	0.21 (63.6%)
Total	13.33	0.33

Table 3. The time in milliseconds taken to run SQLite test cases in a child process, using FORK vs. ON-DEMAND-FORK. The results are the average of 10 runs.

Real-world Applications

- **Snapshot: Redis**

Percentile	Latency (ms)		Reduction
	Fork	On-demand-fork	
≥50%	4.319	3.871	10.37%
≥90%	5.247	4.159	20.74%
≥95%	5.343	4.255	20.36%
≥99%	5.695	4.575	19.67%
≥99.9%	6.335	4.799	24.25%
≥99.99%	16.255	5.535	65.95%

Table 4. Redis request-response percentile latency when configured to take snapshots with Fork and On-demand-fork. The benchmark ran for 135 seconds, averaging over 1.5 million requests per second. The latency values are the average of 5 repeated runs.

Type	Fork	On-demand-fork	Reduction
Mean (ms)	7.40	0.12	98.38%
Std. Dev. (ms)	0.42	0.007	98.33%

Table 5. The time Redis takes to fork when taking snapshots, using Fork vs. On-demand-fork. The results are the average of 5 issued snapshot commands.

- **TriforceAFL: VM Cloning**

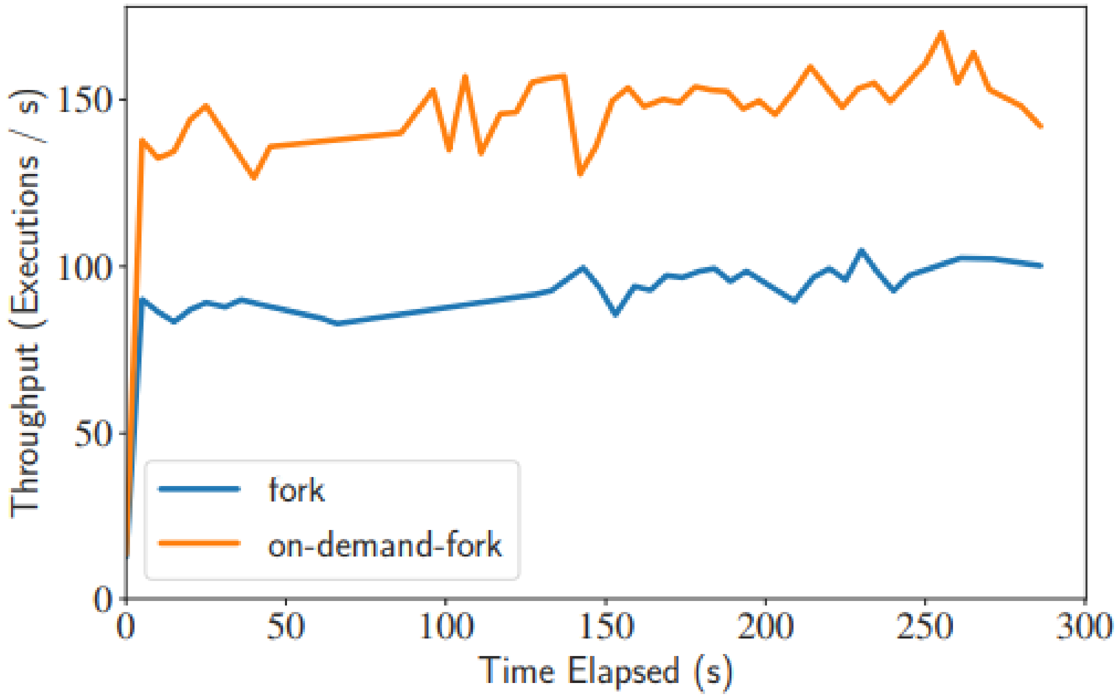


Figure 10. TriforceAFL execution throughput using Fork and On-demand-fork. The dips are due to inputs that cause long system calls.

59.3% higher throughput

Real-world Applications

- Apache HTTP Server

	Fork	ON-DEMAND-FORK	Difference
Mean (μs)	34.3	33.7	-1.75%
Max (μs)	285.2	304.0	+6.59%

Table 6. Response latency of Apache HTTP Server immediately after it is started. The latency values are the average of 5 experiment runs.

No significant benefits

Percentile	Latency (μs)		Difference
	Fork	ON-DEMAND-FORK	
≥50%	35.0	32.4	-7.4%
≥75%	36.5	36.4	-0.3%
≥90%	38.0	39.8	+4.7%
≥99%	51.8	53.6	+3.5%

Table 7. Response latency distribution of Apache HTTP Server immediately after it is started. The latency values are the average of 5 experiment runs.

Not all workloads benefit from ON-DEMAND-FORK

ON-DEMAND-FORK

is fast and efficient

- **270 times faster fork**
- **2.26 times fuzzing throughput**
- **65% lower Redis tail request latency**

Traditional fork

is slow

https://github.com/rssys/on-demand-fork

ON-DEMAND-FORK: A Microsecond Fork for Memory-Intensive and Latency-Sensitive Application

Kaiyang Zhao, Sishuai Gong, Pedro Fonseca,

In Proceedings of the Sixteenth European Conference on Computer Systems (EuroSys '21)

Thank You!

2022. 01. 18

Presentation by Han, Yejin

hyj0225@dankook.ac.kr