# Making the "Box" Transparent:
## System Call Performance as a First-class Result

Yaoping Ruan, Vivek Pai
Princeton University

# Outline

- Motivation
- Design & implementation
- Case study
- More results

# Beginning of the Story

- Flash Web Server on the SPECWeb99 benchmark yield very low score – 200
  - 220: Standard Apache
  - 400: Customized Apache
  - 575: Best - Tux on comparable hardware
- However, Flash Web Server is known to be fast

Motivation

# Performance Debugging of OS-Intensive Applications

Web Servers (+ ***full*** SpecWeb99)

- High throughput
  - Large working sets
  - Dynamic content
  - QoS requirements
  - Workload scaling

CPU/network

disk activity

multiple programs

latency-sensitive

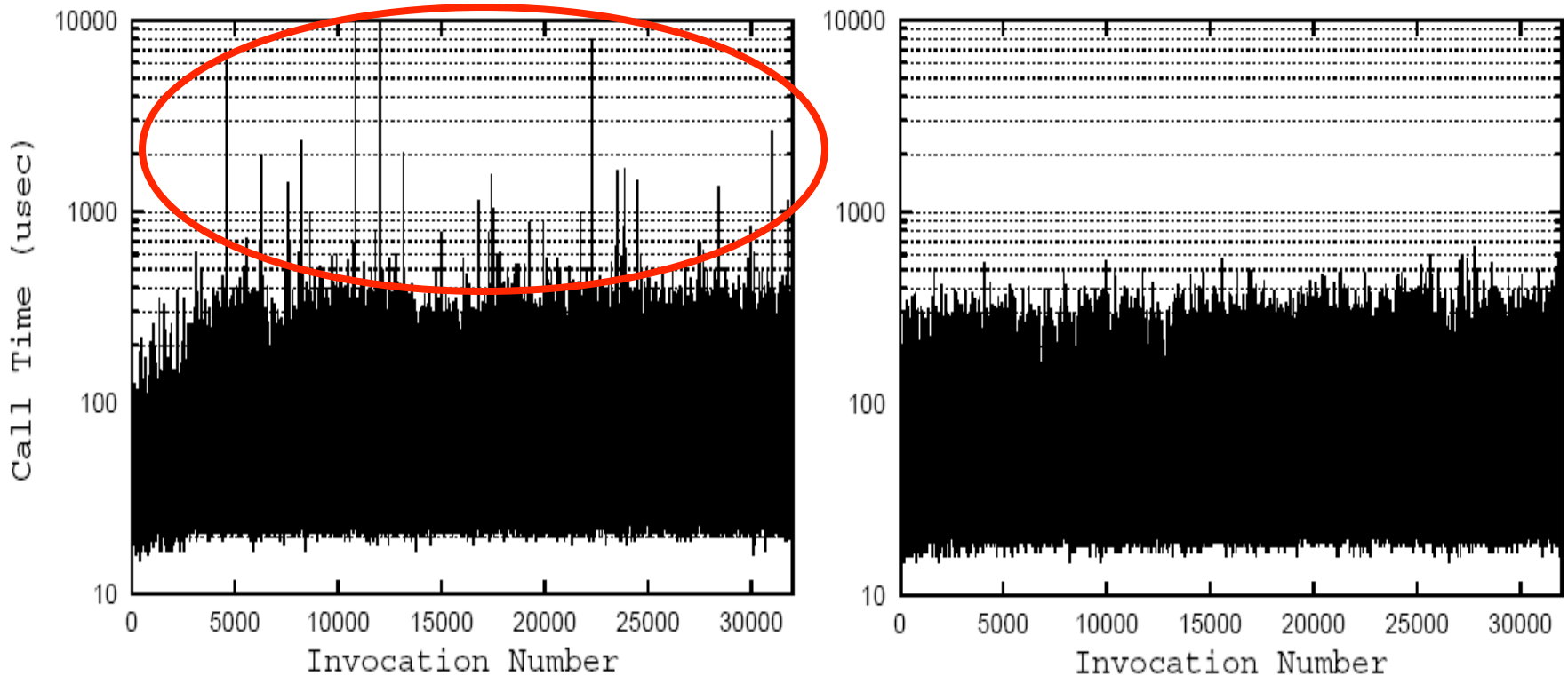
overhead-sensitive

How do we debug such a system?

# Current Tradeoffs

- Statistical sampling
  - DCPI, Vtune, Oprofile
  - ✓ Fast
  - ✗ Completeness
- Call graph profiling
  - gprof
  - ✓ Detailed
  - ✗ High overhead > 40%
- Measurement calls
  - getrusage()
  - gettimeofday( )
  - ✓ Online
  - ✗ Guesswork, inaccuracy

# Example of Current Tradeoffs




gettimeofday( )

In-kernel measurement

Wall-clock time of an non-blocking system call

# Design Goals

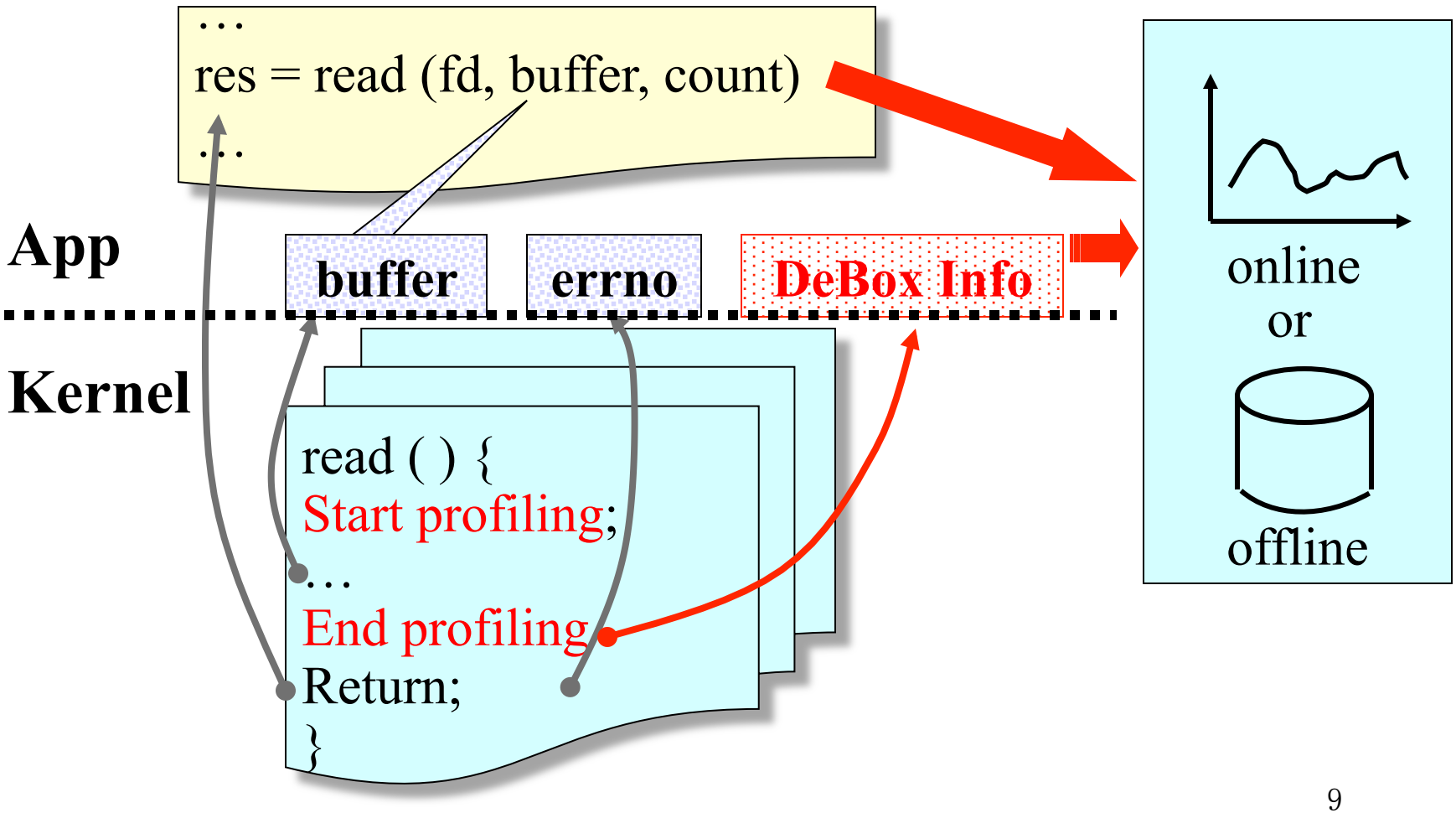
- Correlate kernel information with application-level information
- Low overhead on "useless" data
- High detail on useful data
- Allow application to ***control*** profiling and programmatically ***react*** to information

Design

# DeBox: Splitting Measurement Policy & Mechanism

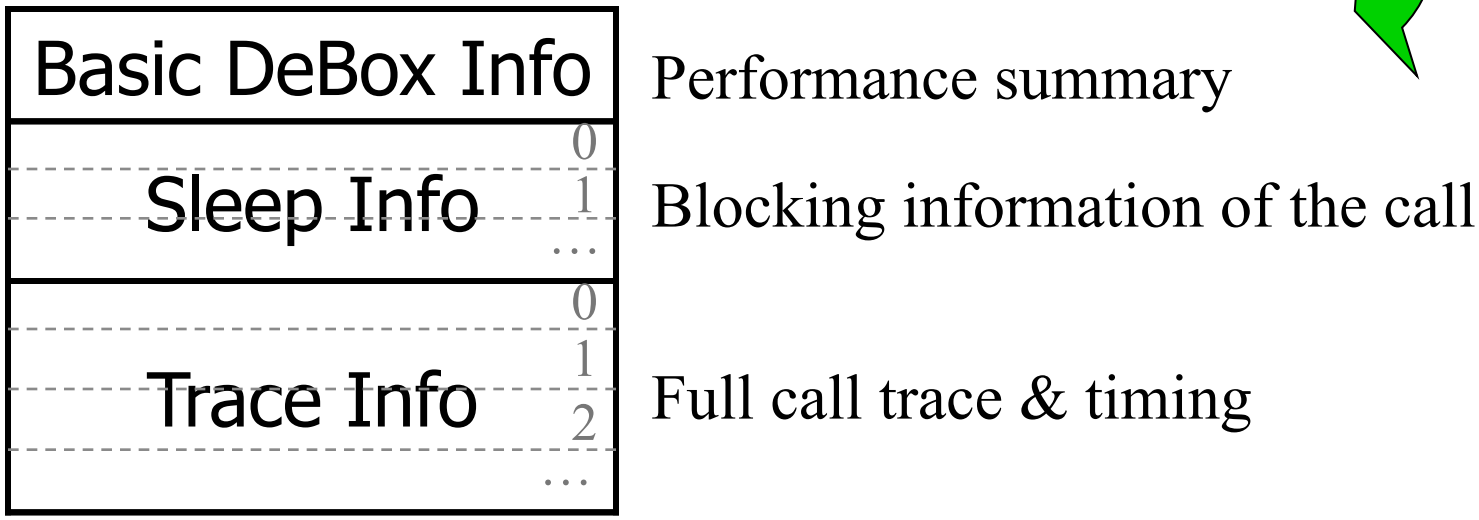
- Add profiling ***primitives*** into kernel
- Return feedback with each syscall
  - *First-class result*, like *errno*
- Allow programs to interactively profile
  - Profile by call site and invocation
  - Pick which processes to profile
  - Selectively store/discard/utilize data

# DeBox Architecture

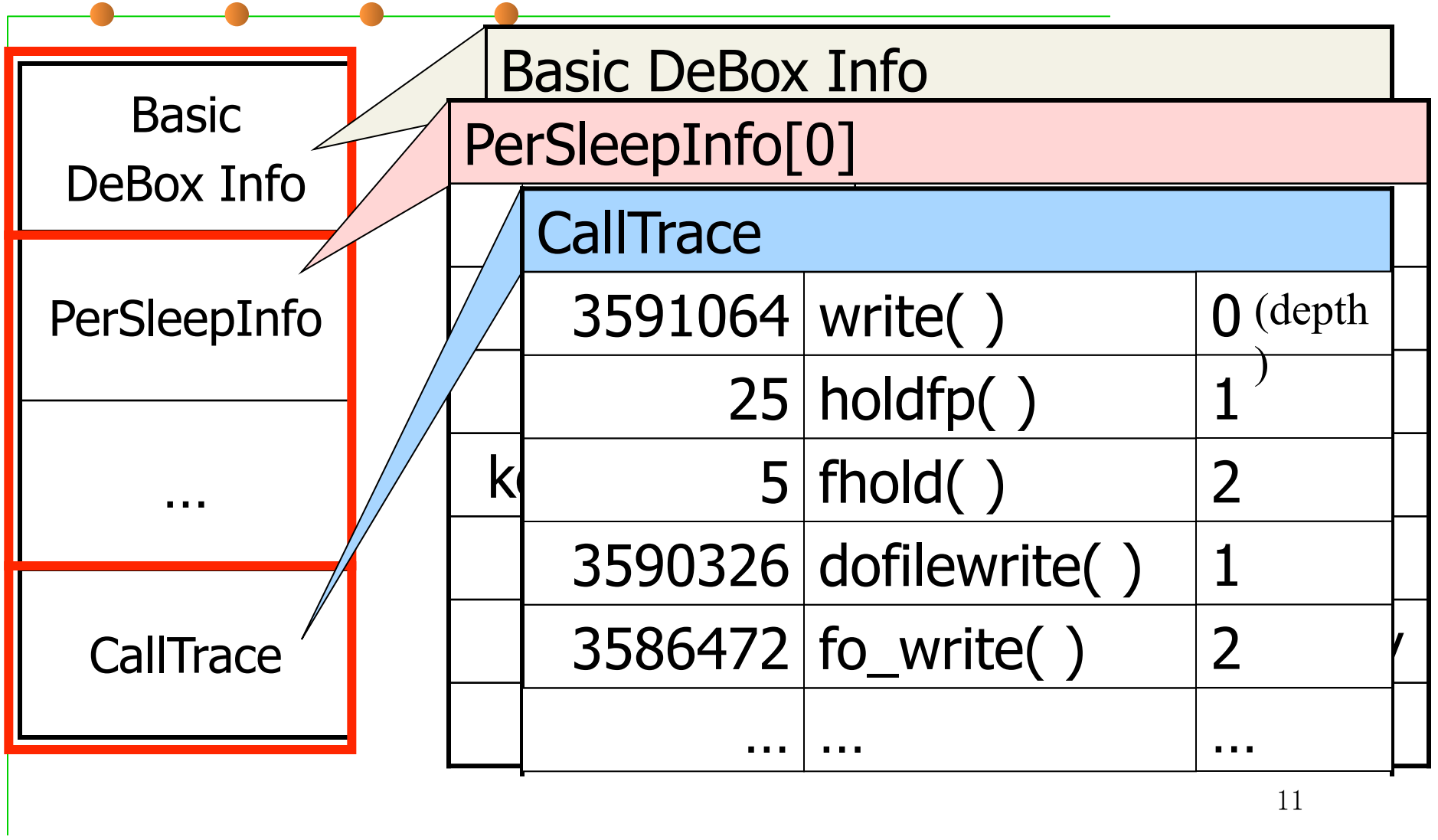


# DeBox Data Structure

**DeBoxControl ( *DeBoxInfo* *resultBuf,**
***int* maxSleeps,**
***int* maxTrace )**



# Sample Output: Copying a 10MB Mapped File



# In-kernel Implementation

- 600 lines of code in FreeBSD 4.6.2
- Monitor trap handler
  - System call entry, exit, page fault, etc.
- Instrument scheduler
  - Time, location, and reason for blocking
  - Identify dynamic resource contention
- Full call path tracing + timings
  - More detail than gprof's call arc counts

# General DeBox Overheads

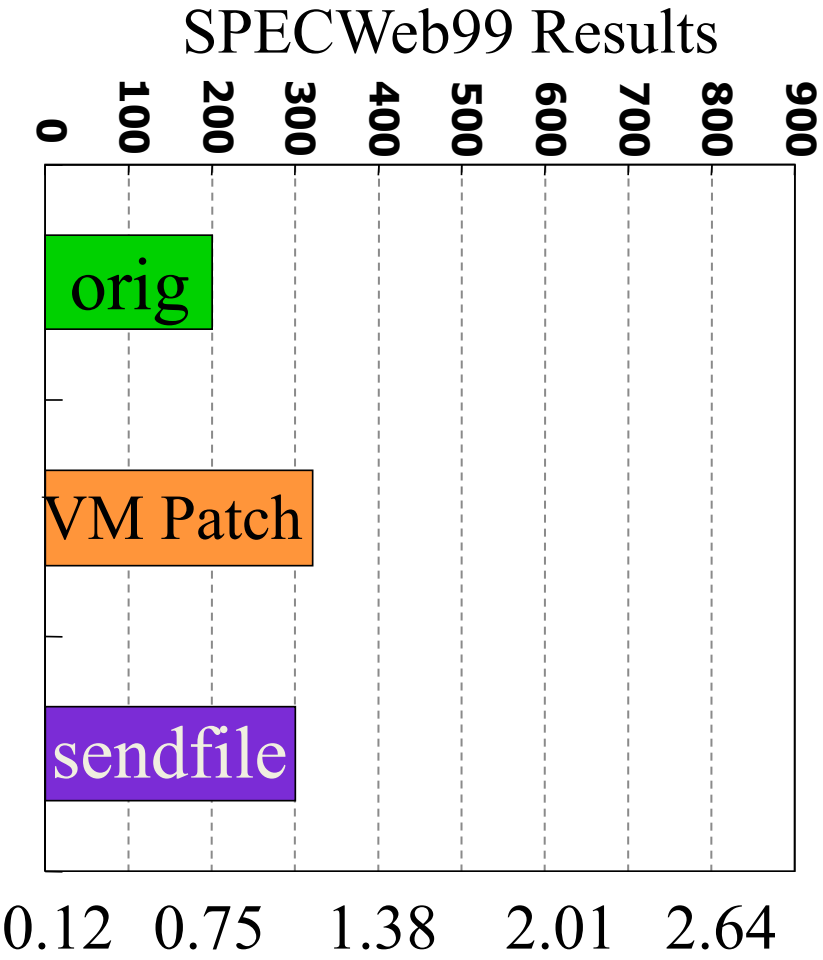
| call name or read size | base time | DeBox without call trace | | DeBox call trace | |
|---|---|---|---|---|---|
| | | off | on | off | on |
| getpid | 0.46 | +0.00 | +0.50 | +0.03 | +1.45 |
| gettimeofday | 5.07 | +0.00 | +0.43 | +0.03 | +1.52 |
| pread 128B | 3.27 | +0.02 | +0.56 | +0.21 | +2.03 |
| 1024 bytes | 6.74 | +0.00 | +0.68 | +0.27 | +2.02 |
| 4096 bytes | 18.43 | +0.03 | +0.74 | +0.29 | +2.16 |

| | tar-gz a directory with | | make kernel |
|---|---|---|---|
| | 1MB file | 10MB file | |
| base time | 275.61 ms | 3078.50 ms | 236.96 s |
| basic on | +0.97 ms | +22.73 ms | +1.74 s |
| full support | +1.03 ms | +44.58 ms | +7.49 s |

Case Study

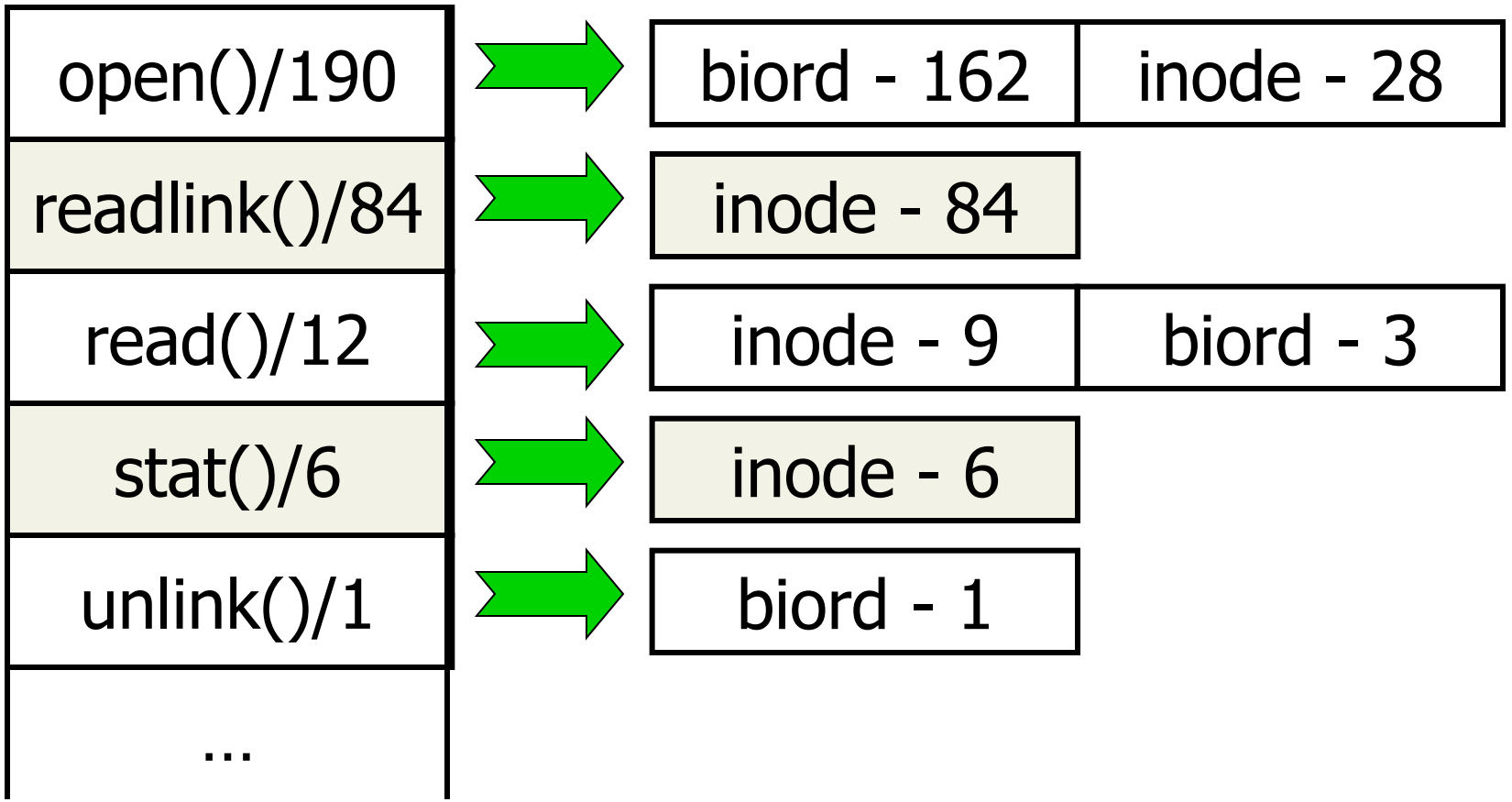
# Case Study: Using DeBox on the Flash Web server

- Experimental setup
  - Mostly 933MHz PIII
  - 1GB memory
  - Netgear GA621 gigabit NIC
  - Ten PII 300MHz clients



Case Study

# Example 1: Finding Blocking System Calls

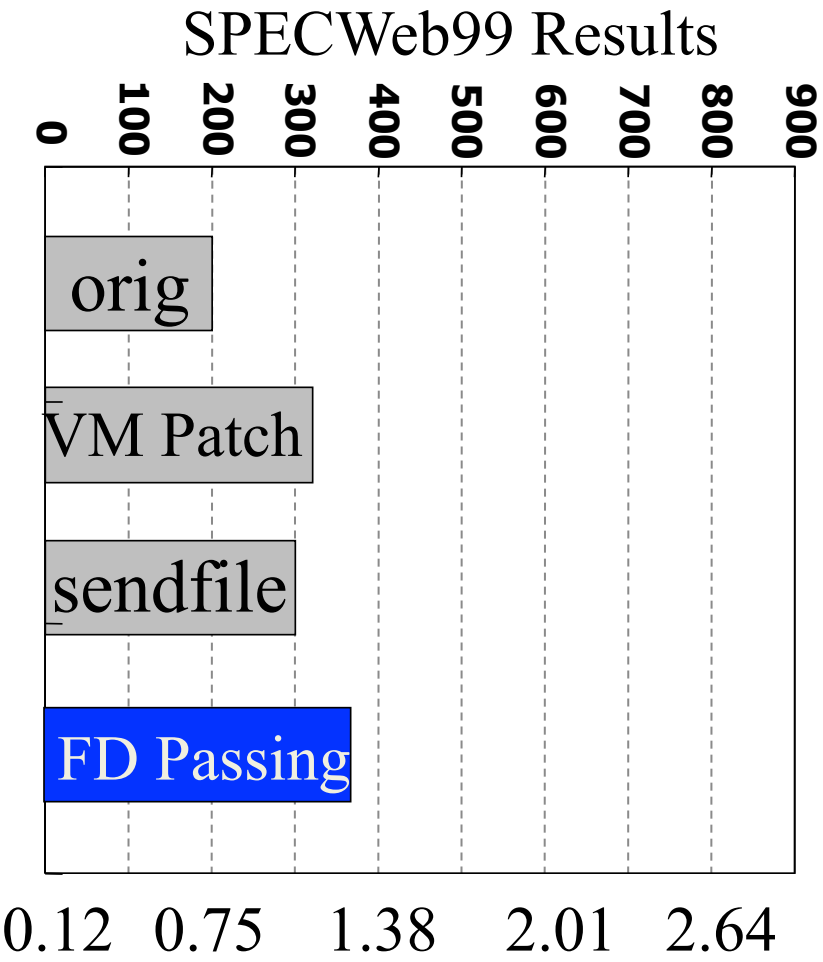



Blocking system calls, resource blocked and their counts

Case Study

# Example 1 Results & Solution

- Mostly metadata locking
- Direct all metadata calls to name convert helpers
- Pass open FDs using *sendmsg( )*



Case Study

# Example 2: Capturing Rare Anomaly Paths

```
FunctionX( ) {
   …
   open( );
   … }
FunctionY( ) {
   open( );
   …
   open( );  }
```

When blocking happens:

abort( ); (or fork + abort)

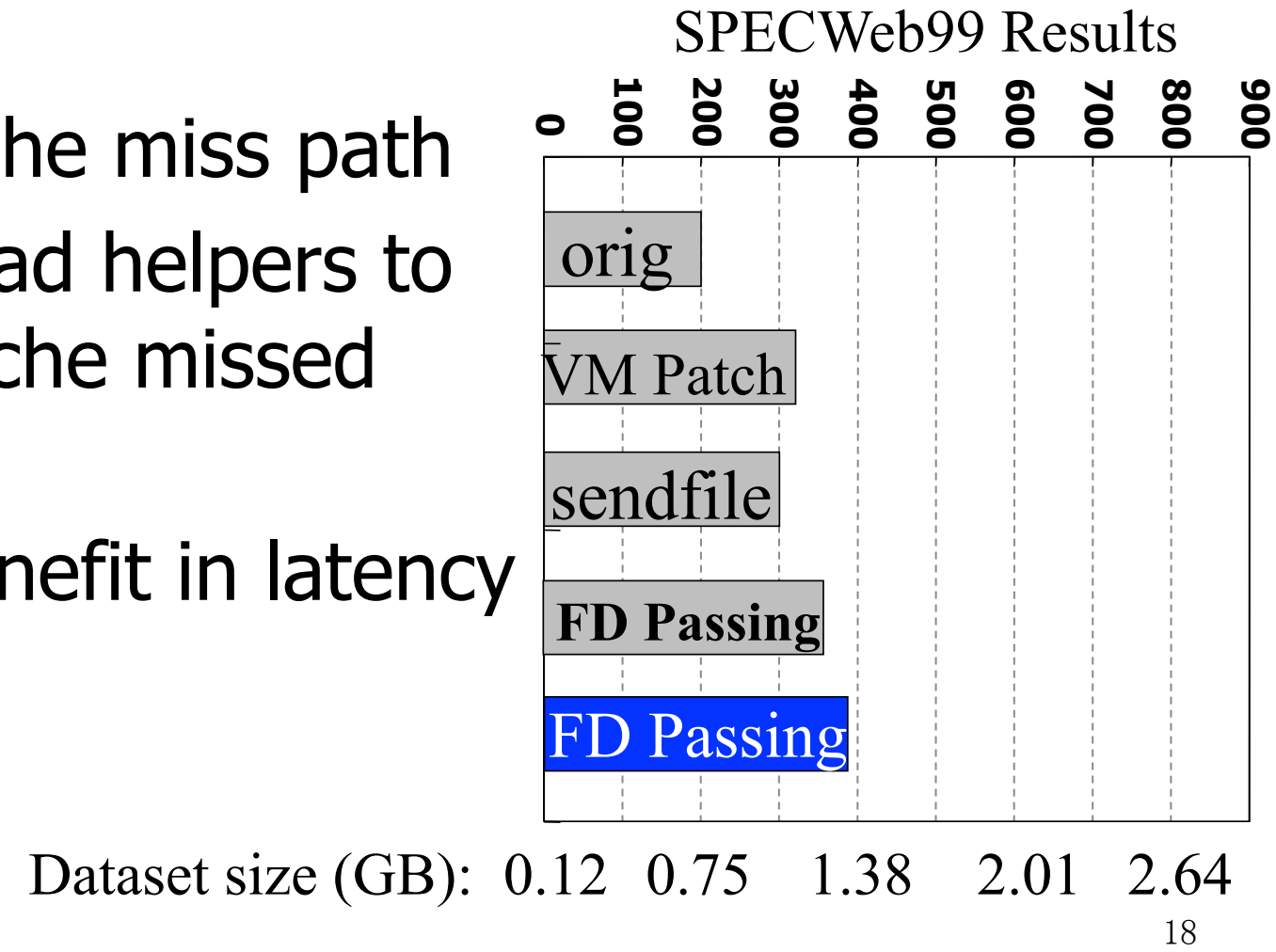
Record call path

1. Which call caused the problem
2. Why this path is executed (App + kernel call trace)

Case Study

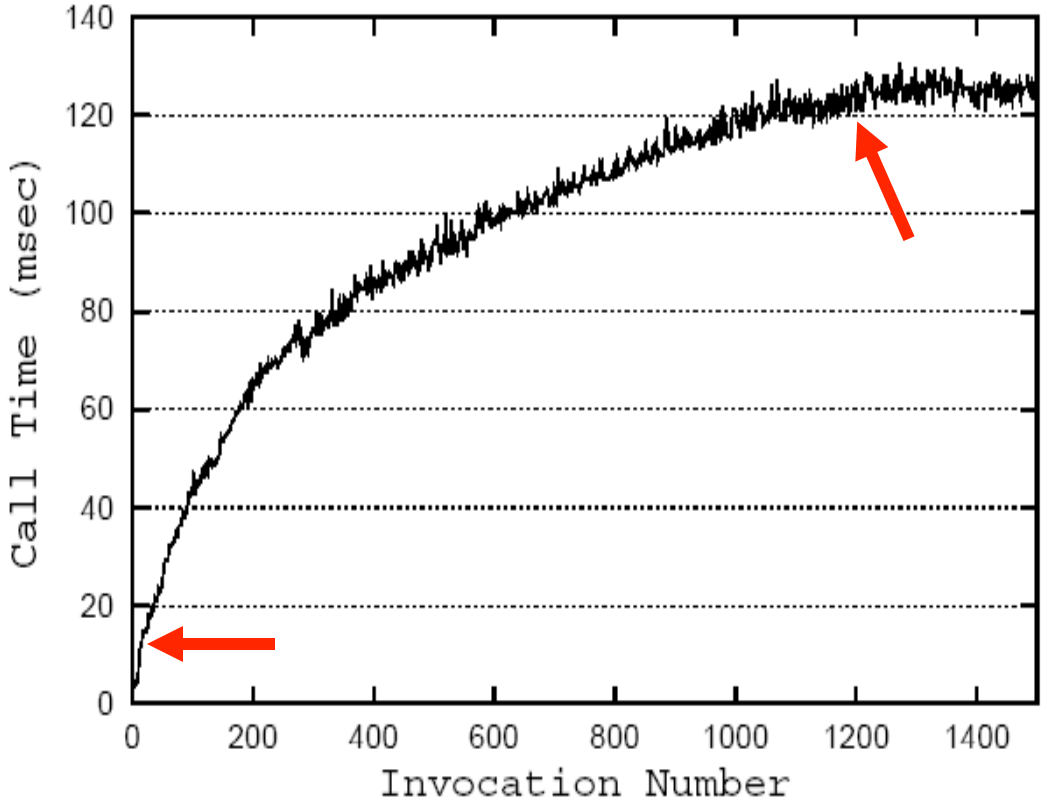
# Example 2 Results & Solution

- Cold cache miss path
- Allow read helpers to open cache missed files
- More benefit in latency



Case Study

# Example 3: Tracking Call History & Call Trace




Call trace indicates:

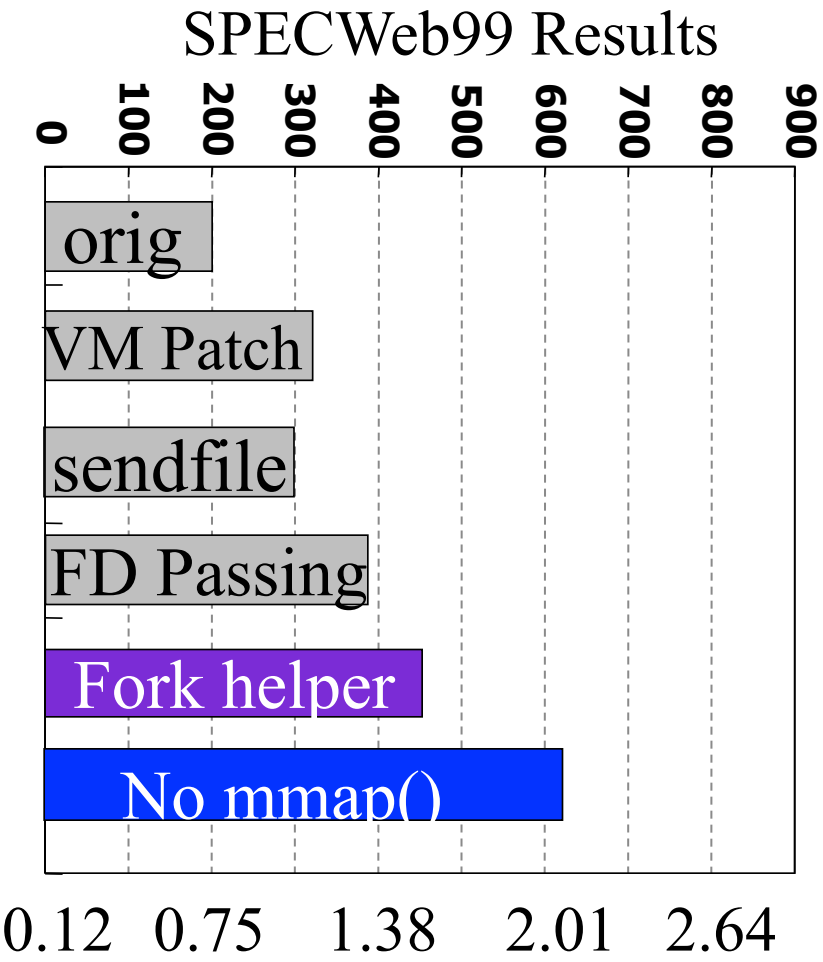
- File descriptors copy - fd_copy( )
- VM map entries copy - vm_copy( )

Call time of fork( ) as a function of invocation

Case Study

# Example 3 Results & Solution

- Similar phenomenon on *mmap( )*
- Fork helper
- eliminate *mmap( )*



Case Study

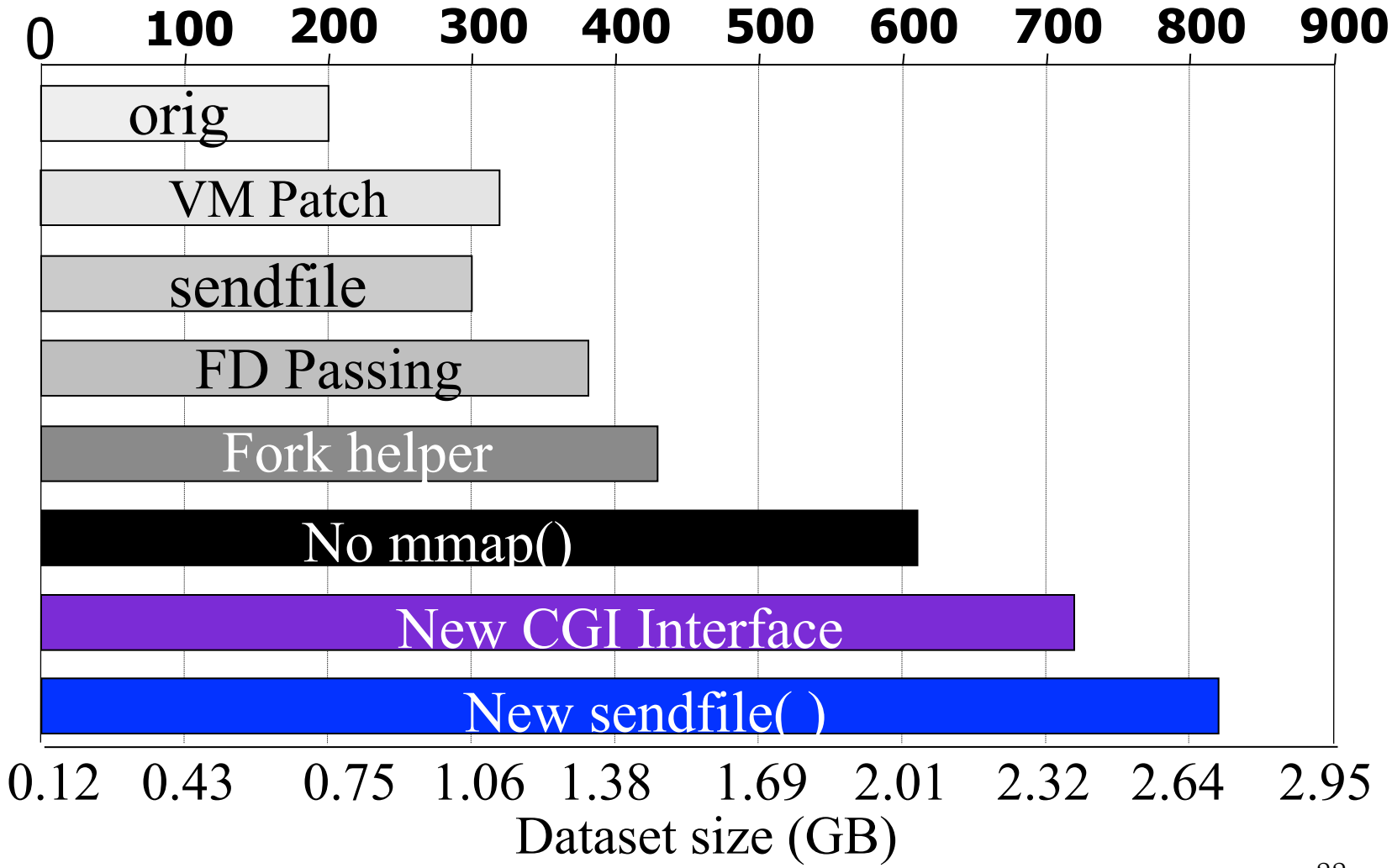
# Sendfile Modifications (Now Adopted by FreeBSD)

| time | label | kernel file | line |
|---|---|---|---|
| 6492 | sfbufa | kern/uipc_syscalls.c | 1459 |
| 702 | getblk | kern/kern_lock.c | 182 |
| 984544 | biord | kern/vfs_bio.c | 2724 |

- Cache pmap/sfbuf entries
- Return special error for cache misses
- Pack header + data into one packet

Case Study

# SPECWeb99 Scores

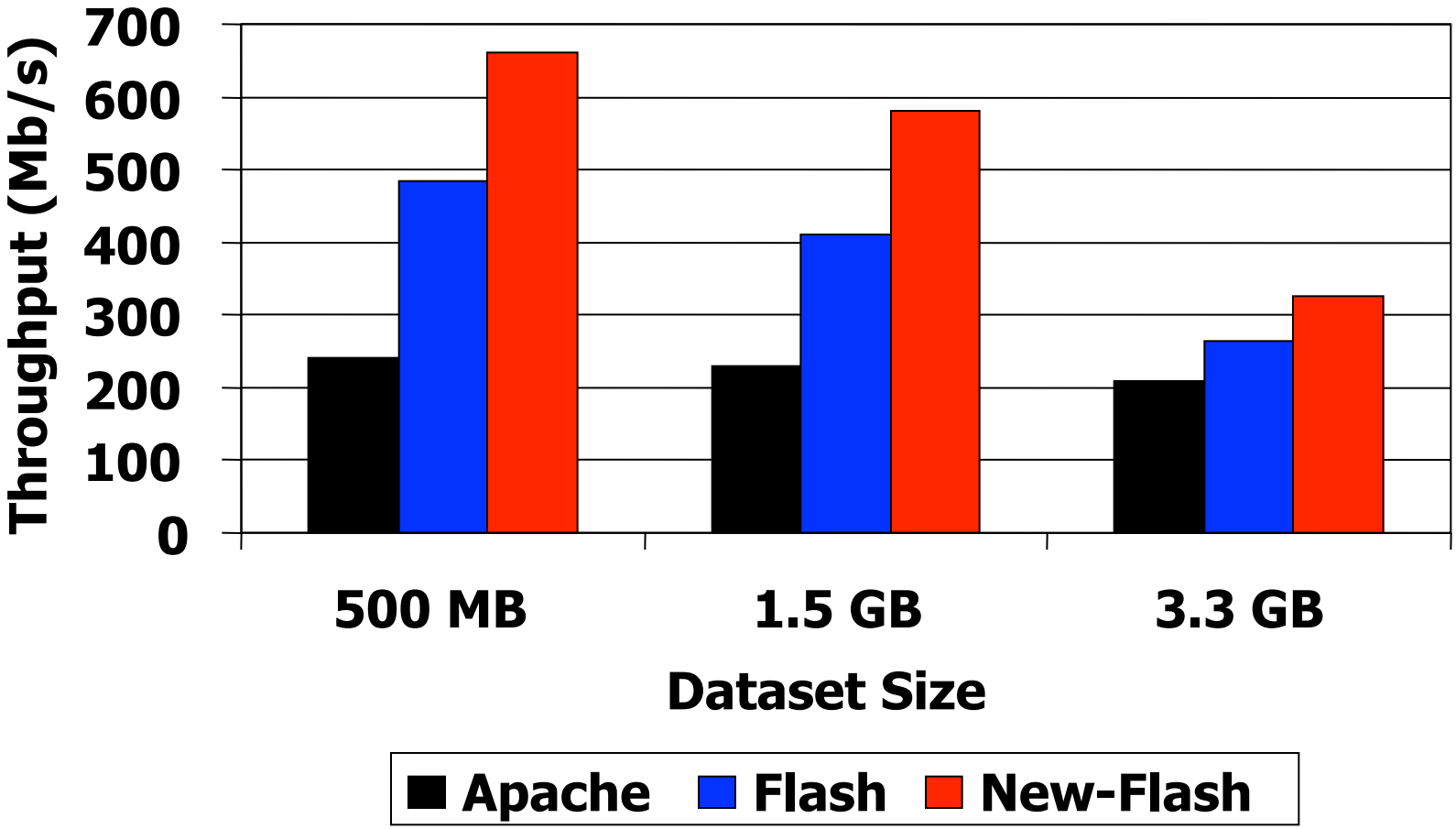


# New Flash Summary

- Application-level changes
  - FD passing helpers
  - Move fork into helper process
  - Eliminate mmap cache
  - New CGI interface
- Kernel sendfile changes
  - Reduce pmap/TLB operation
  - New flag to return if data missing
  - Send fewer network packets for small files

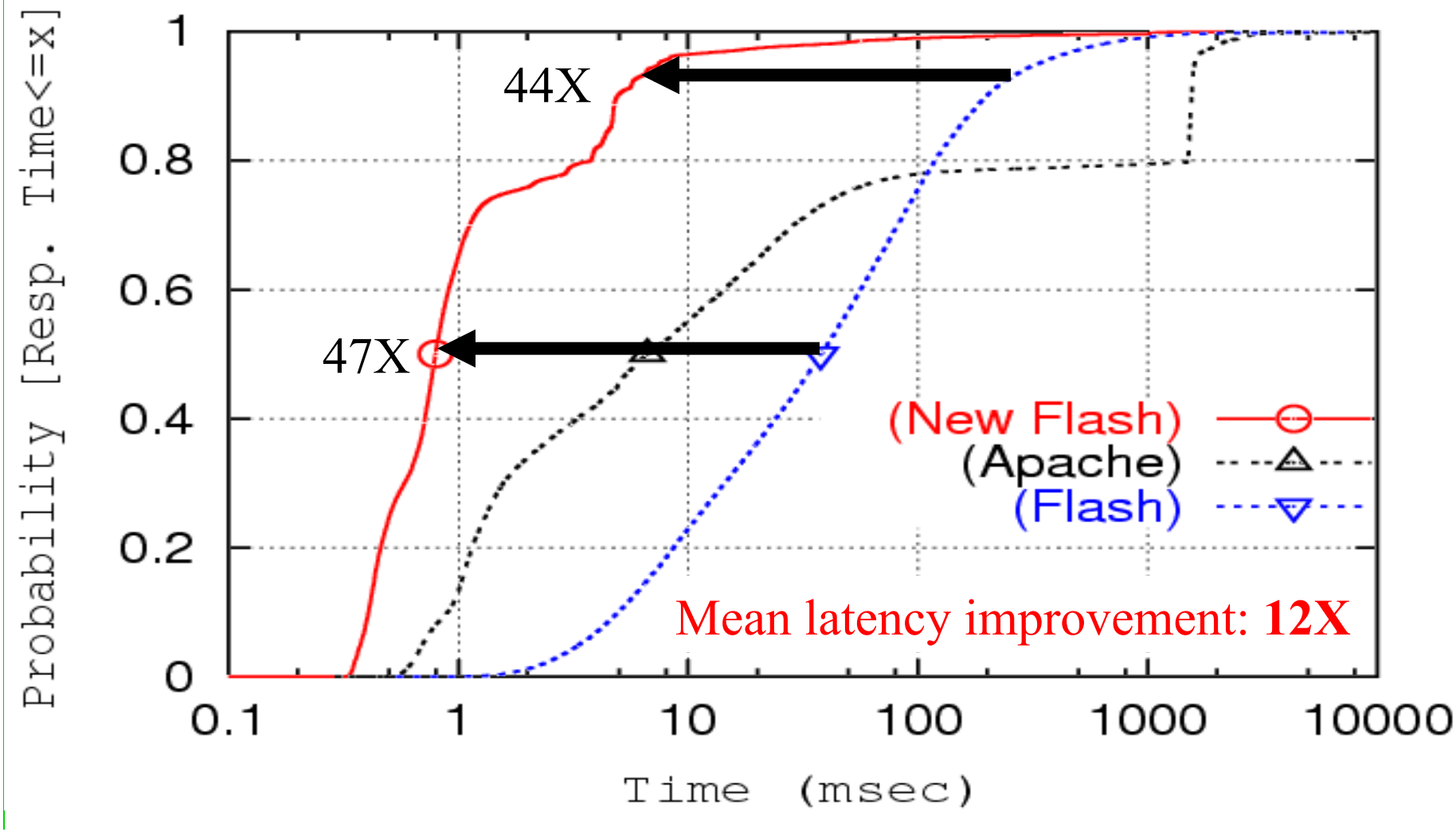
Other Results

# Throughput on SPECWeb Static Workload

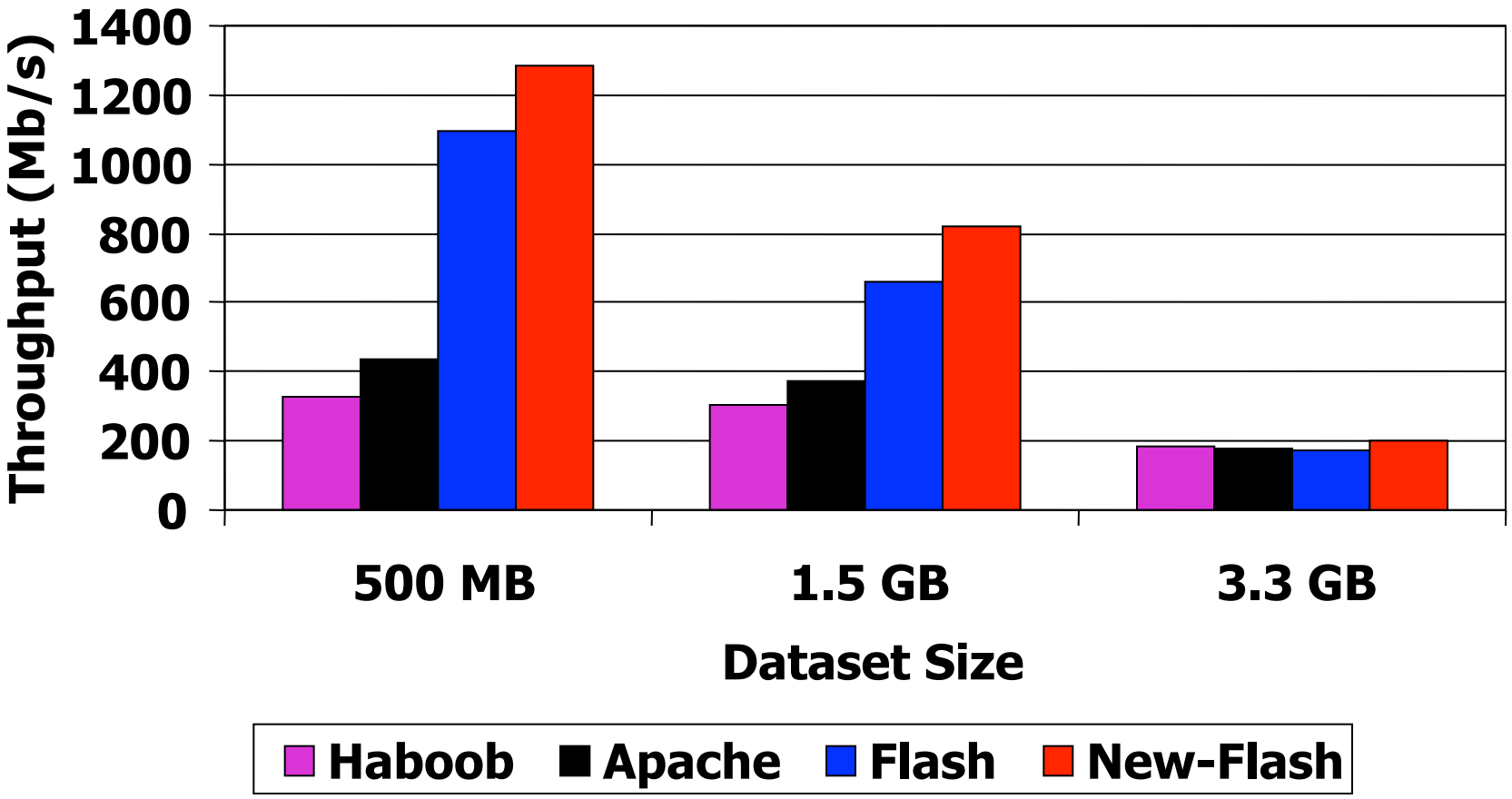


Other Results

# Latencies on 3.3GB Static Workload



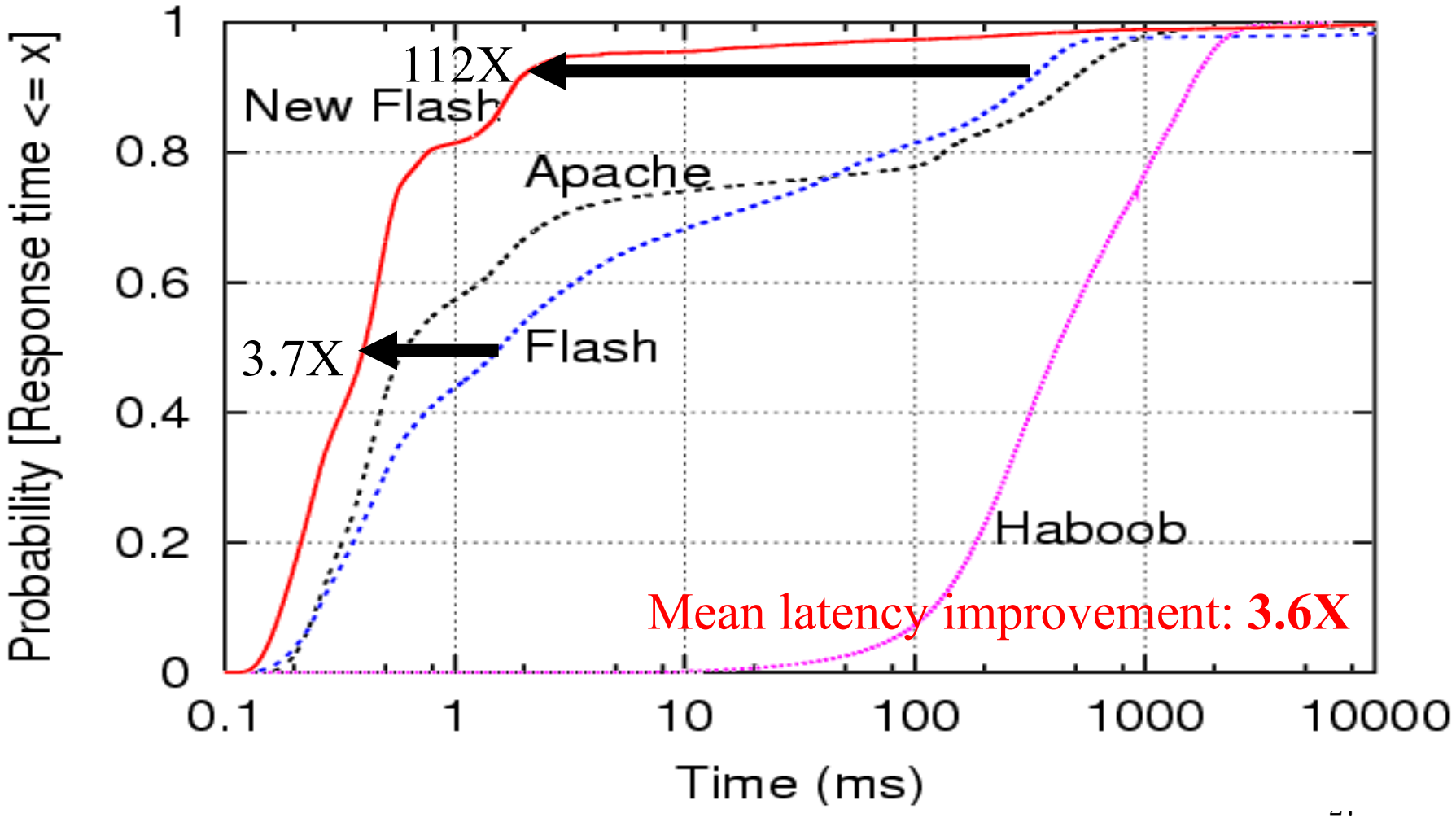
# Throughput Portability (on Linux)




(Server: 3.0GHz P4, 1GB memory)

Other Results

# Latency Portability (3.3GB Static Workload)



# Summary

- DeBox is effective on OS-intensive application and complex workloads
  - Low overhead on real workloads
  - Fine detail on real bottleneck
  - Flexibility for application programmers
- Case study
  - SPECWeb99 score quadrupled
    - Even with dataset 3x of physical memory
  - Up to 36% throughput gain on static workload
  - Up to 112x latency improvement
  - Results are portable

www.cs.princeton.edu/~yruan/debox

yruan@cs.princeton.edu
vivek@cs.princeton.edu

# Thank you

# SpecWeb99 Scores

- Standard Flash 200
- Standard Apache 220
- Apache + special module 420
- Highest 1GB/1GHz score 575
- Improved Flash 820
- Flash + dynamic request module 1050

# SpecWeb99 on Linux

- Standard Flash 600
- Improved Flash 1000
- Flash + dynamic request module 1350

- 3.0GHz P4 with 1GB memory

# New Flash Architecture