



# Benchmarking FreeBSD

Ivan Voras <[ivoras@freebsd.org](mailto:ivoras@freebsd.org)>



# What and why?

- Everyone likes a nice benchmark graph :)
  - And it's nice to keep track of these things
- The previous major run comparing FreeBSD to Linux was done by Kris Kennaway in 2008.
- There are occasional benchmarks appearing on blogs and mailing lists which show “interesting” results...



# Target audience

- Mostly developers
  - The results are decent, but not rosy
  - Significant space for improvement
- Some system administrator material
  - What to do, what not to do
  - Tuning?
- “Avoid benchmarking?”
  - Do you need it?





# Purpose of benchmarking...

- To check your system for configuration bugs and obvious problems (hw/sw)
- To check your capacity for running an application (web server, db server, etc.)
- To compare your hardware to others'
- To compare your favorite application / operating system to others...



# Repeatability

- The best benchmarks are performed in a way anyone can repeat them
- Both result verification (error checking) and a way to compare their own setup to the one tested
- Repeatability is a good thing...





# Test hardware

- 2x IBM x3250 M3
- Xeon E3-1220 3.1 GHz, 4-core (no HTT)
- 32 GB RAM
- 4x SATA drive  
RAID0
- 1 Gbps  
NIC (2 ports)

Directly connected NICs





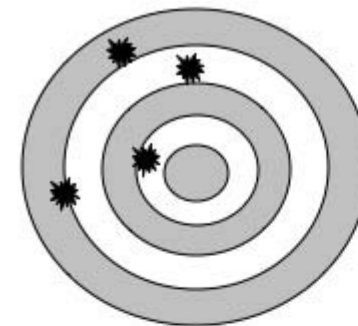
# Test software

- FreeBSD 9.1 and 10-CURRENT (HEAD)
- CentOS 6.3
- PostgreSQL 9.2.3
- blogbench 1.1
- bonnie++ 1.97
- filebench 1.4.8
- Mdcached 1.0.7

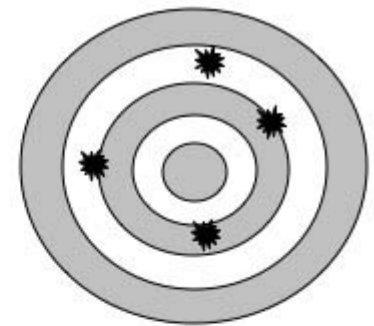


# Accuracy and precision

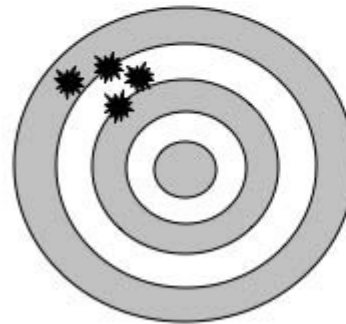
- **Accurate** = if it measures what we think it measures (or is it off the mark)
- **Precise** = is the measured value good approximation of the “real” one (or is it affected by noise)



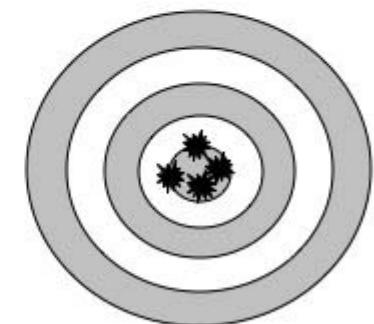
Not Accurate  
Not Precise



Accurate  
Not Precise



Not Accurate  
Precise



Accurate  
Precise





# Systematic and random errors

- **Systematic** = we're not measuring what we think we're measuring (the method of benchmarking is wrong, even though the results may be repeatable and look correct).
- **Random** = affected by “noise” outside our control, results in non-repeatable measurements.



# Expressing precision

- Find meaningful precision
  - (or: avoid false precision)
- ~~412.567 MB/s~~
- ~~412.5 MB/s~~
- ~~412 MB/s~~
- **410 MB/s**



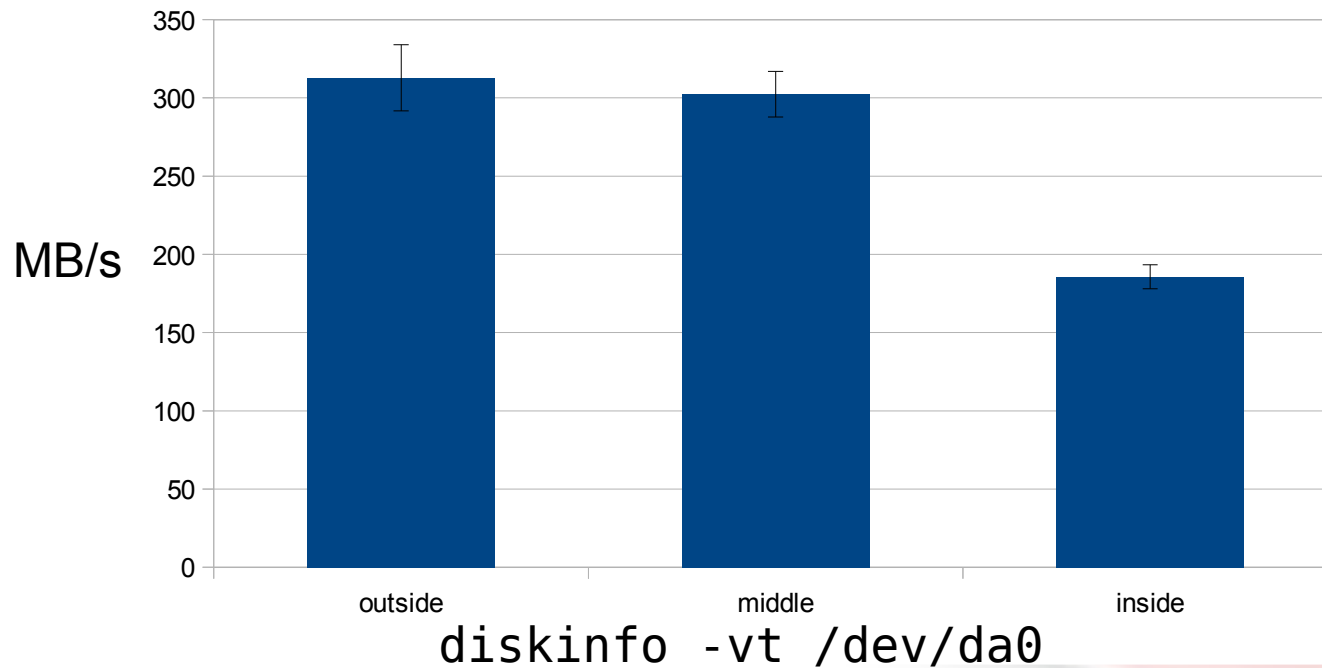
# Standard deviation

- The “error bars”
- $1234 \pm 5$  of something
- Expresses confidence in your measurement – how precise they are (clustered around the measured value)



# Example: hard drive performance

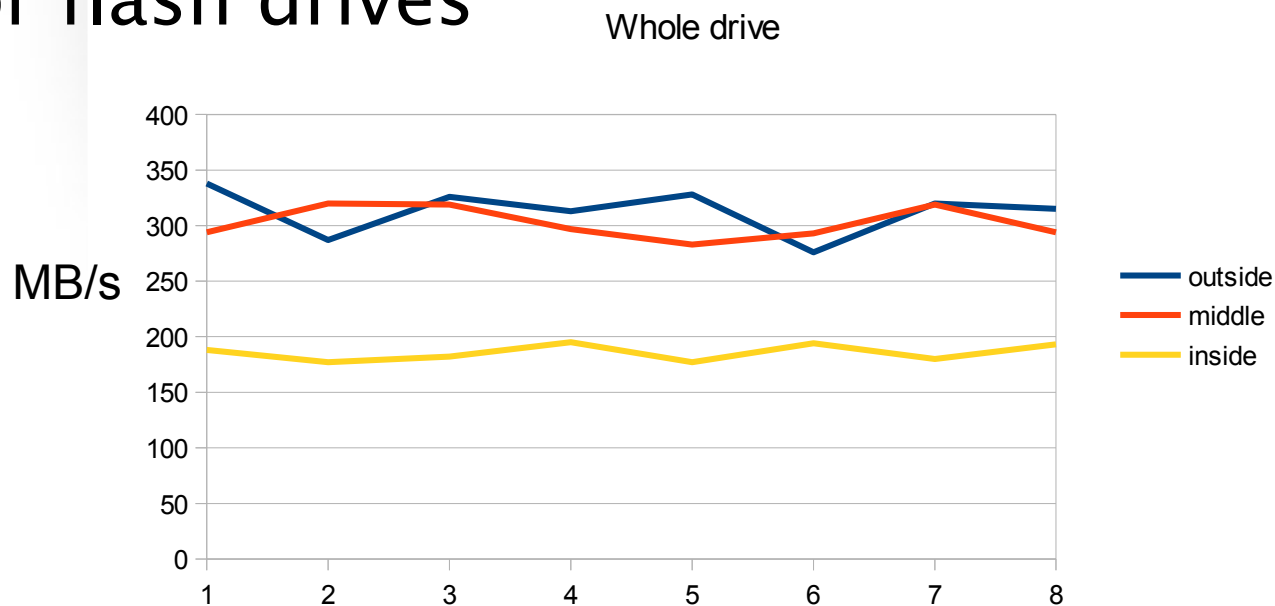
- (Systematic: you are not measuring what you think are measuring)
- File systems vs (spinning) drives





# File systems are hard to measure

- Data location-dependant performance
- Data structure (re)use-dependant performance (newfs vs old system)
  - Double for flash drives
- Noise







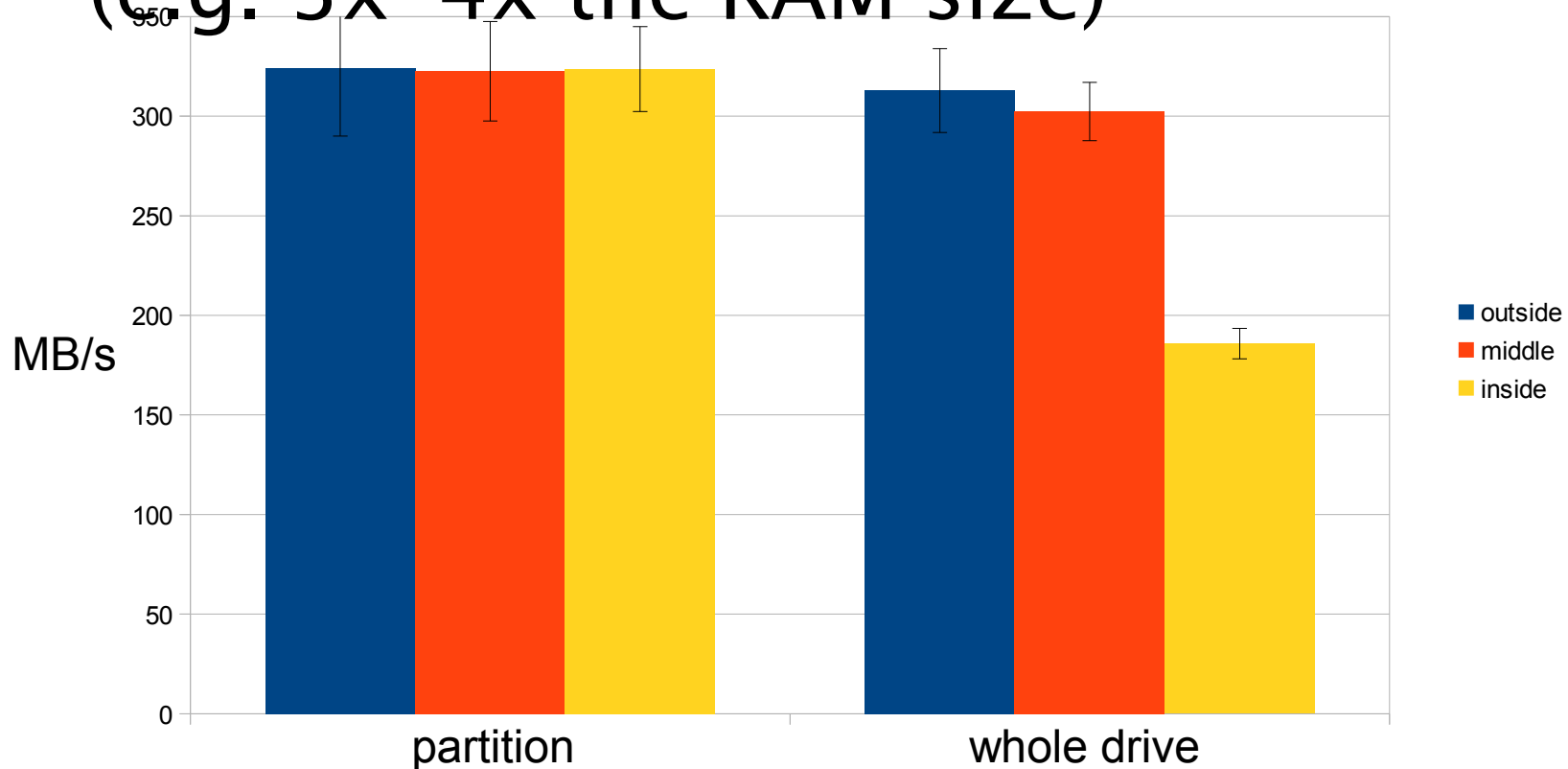
# Speaking of noise...





# Reducing the difference

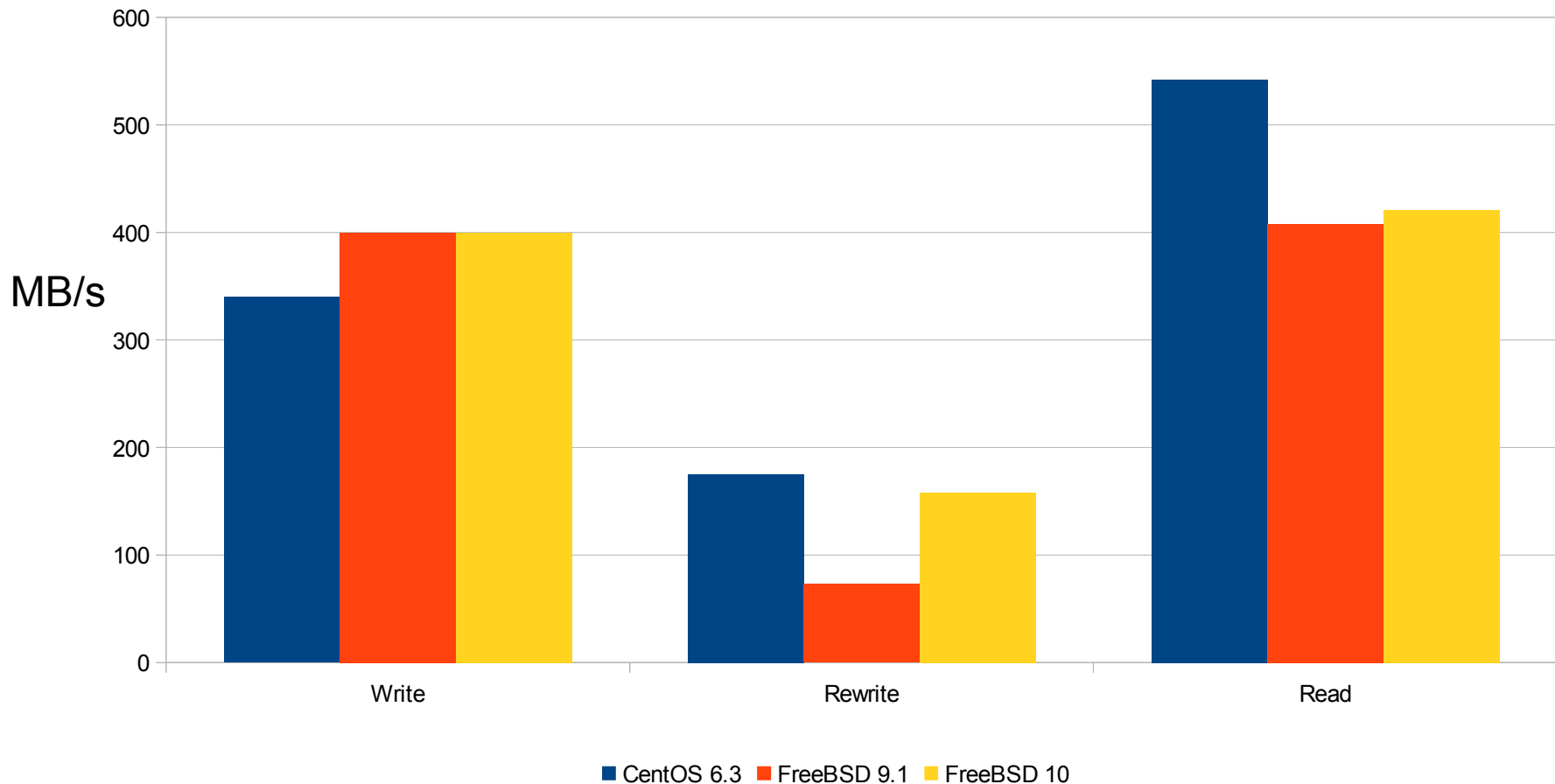
- Create a partition spanning the minimal required disk space  
(e.g. 3x–4x the RAM size)





# bonnie++

- File system bandwidth & ~~file op~~ latency





# File systems are complex beasts

- Think about it – it's a database...
- Data layout issues (esp. on rotating rust)
- Concurrent access issues
- “miscellaneous” features such as attributes, security, reliability, TRIM
- NFS is also a horror but for different reasons



# Blogbench

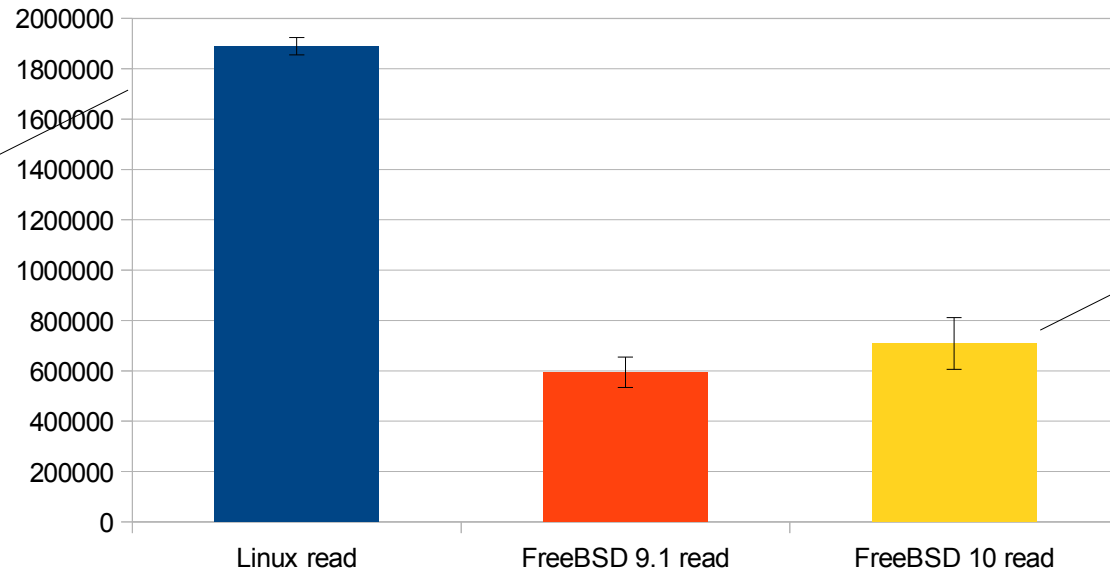
- Stresses file system parallelism and random disk IO
  - Creates a tree of small-ish files (2 KiB – 64 KiB)
  - Reads and writes them
  - Atomic renames for (some) writes
  - Multithreaded



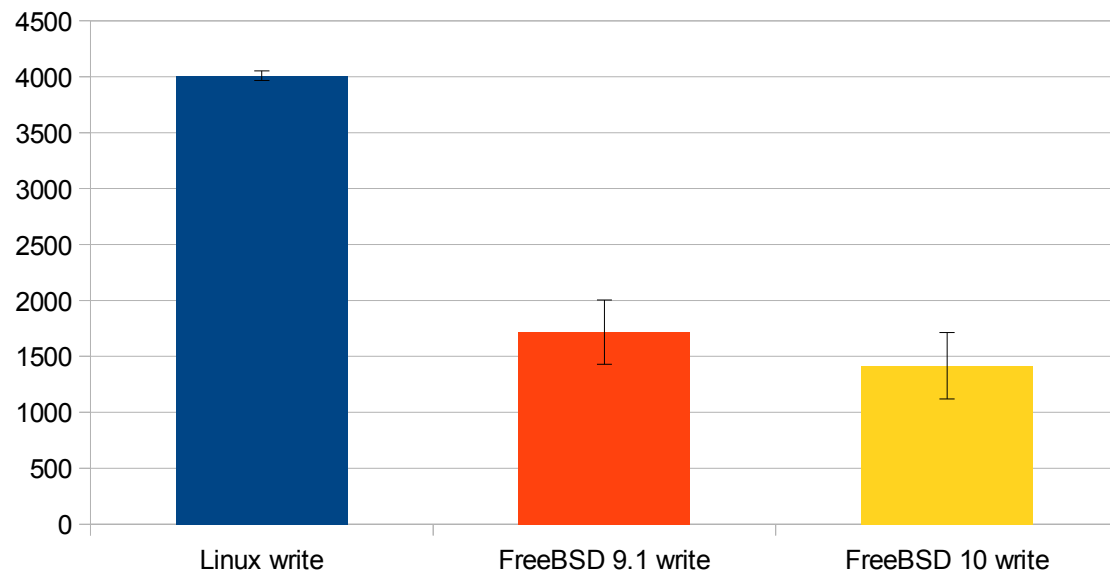


# Blogbench results

**1,900,000**



**700,000**





# Blogbench FreeBSD 9.1 vs 10

```
x read-91
+ read-10
+-----+
|xx x          * x x*x + *   x  +*  + +          +      + +|
| |           AM|           |   M  A           |           |
+-----+
      N           Min           Max           Median           Avg           Stddev
x   11           509450          695303          600394          593580.45        60234.934
+   11           577355          893889          695303          708560.45        102923.93
Difference at 95.0% confidence
      114980 +/- 75005.3
19.3706% +/- 12.6361%
(Student's t, pooled s = 84325.5)
```

```
x write-91
+ write-10
+-----+
| |           +          * x+      x x          x  x  *          x          x+  x|
| |           M  A|           |M           |           |           |
+-----+
      N           Min           Max           Median           Avg           Stddev
x   11           1299          2173          1734          1716.6364        287.0642
+   11           1111          2104          1299          1416             296.7814
Difference at 95.0% confidence
      -300.636 +/- 259.694
-17.5131% +/- 15.128%
(Student's t, pooled s = 291.963)
```



# Why is Blogbench slow on FreeBSD?

- A multithreaded mix of file operations
- On FreeBSD (UFS):
  - open(O\_WRITE)
  - write()
  - rename()
  - etc ... block each other (exclusive lock)
  - + writers block readers
- So called “write bias” of FreeBSD

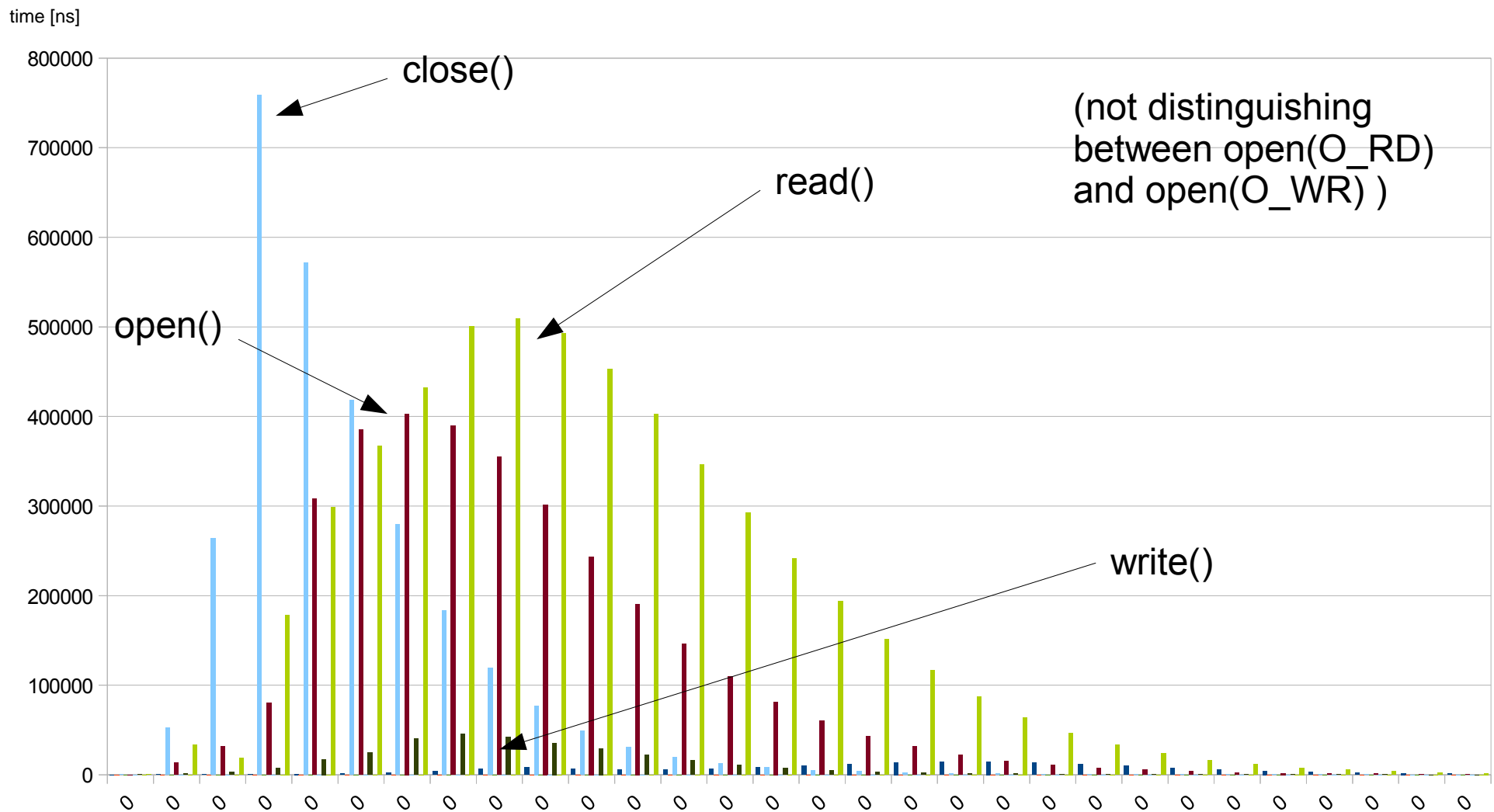


# Why is Blogbench slow on FreeBSD?

```
884 100130 blogbench 1.472372 CALL open(0x7ffffebf5770,0x601<O_WRONLY|O_CREAT|O_TRUNC>,0x180<S_IRUSR|S_IWUSR>)
884 100190 blogbench 1.472381 CALL close(0x2a)
884 100162 blogbench 1.472382 CALL read(0x31,0x602f60,0x10000)
884 100141 blogbench 1.472386 CALL read(0x32,0x602f60,0x10000)
884 100190 blogbench 1.472387 CALL open(0x7ffff73b9ba0,0<O_RDONLY>,<unused>0)
884 100162 blogbench 1.472395 CALL read(0x31,0x602f60,0x10000)
884 100130 blogbench 1.472398 CALL write(0x2a,0x612f60,0x1df2)
884 100141 blogbench 1.472402 CALL read(0x32,0x602f60,0x10000)
884 100162 blogbench 1.472407 CALL close(0x31)
884 100130 blogbench 1.472412 CALL close(0x2a)
884 100190 blogbench 1.472403 CALL open(0x7ffff73b9ba0,0<O_RDONLY>,<unused>0)
884 100162 blogbench 1.472418 CALL open(0x7ffffabd5ba0,0<O_RDONLY>,<unused>0)
884 100130 blogbench 1.472421 CALL rename(0x7ffffebf5770,0x7ffffebf5ba0)
884 100141 blogbench 1.472415 CALL close(0x32)
884 100190 blogbench 1.472423 CALL read(0x2a,0x602f60,0x10000)
884 100141 blogbench 1.472430 CALL open(0x7ffffd5eaba0,0<O_RDONLY>,<unused>0)
884 100162 blogbench 1.472433 CALL read(0x31,0x602f60,0x10000)
884 100190 blogbench 1.472438 CALL read(0x2a,0x602f60,0x10000)
884 100130 blogbench 1.472442 CALL open(0x7ffffebf5770,0x601<O_WRONLY|O_CREAT|O_TRUNC>,0x180<S_IRUSR|S_IWUSR>)
884 100190 blogbench 1.472443 CALL close(0x2a)
884 100141 blogbench 1.472451 CALL open(0x7ffffd5eaba0,0<O_RDONLY>,<unused>0)
884 100162 blogbench 1.472451 CALL read(0x31,0x602f60,0x10000)
884 100190 blogbench 1.472450 CALL open(0x7ffff73b9ba0,0<O_RDONLY>,<unused>0)
884 100141 blogbench 1.472460 CALL read(0x2a,0x602f60,0x10000)
884 100130 blogbench 1.472464 CALL write(0x32,0x612f60,0x13f2)
884 100190 blogbench 1.472465 CALL open(0x7ffff73b9ba0,0<O_RDONLY>,<unused>0)
884 100162 blogbench 1.472461 CALL close(0x31)
884 100190 blogbench 1.472475 CALL read(0x31,0x602f60,0x10000)
884 100162 blogbench 1.472477 CALL open(0x7ffffabd5ba0,0<O_RDONLY>,<unused>0)
884 100141 blogbench 1.472480 CALL read(0x2a,0x602f60,0x10000)
884 100130 blogbench 1.472478 CALL close(0x32)
```



# Why is Blogbench slow on FreeBSD?







# PostgreSQL pgbench

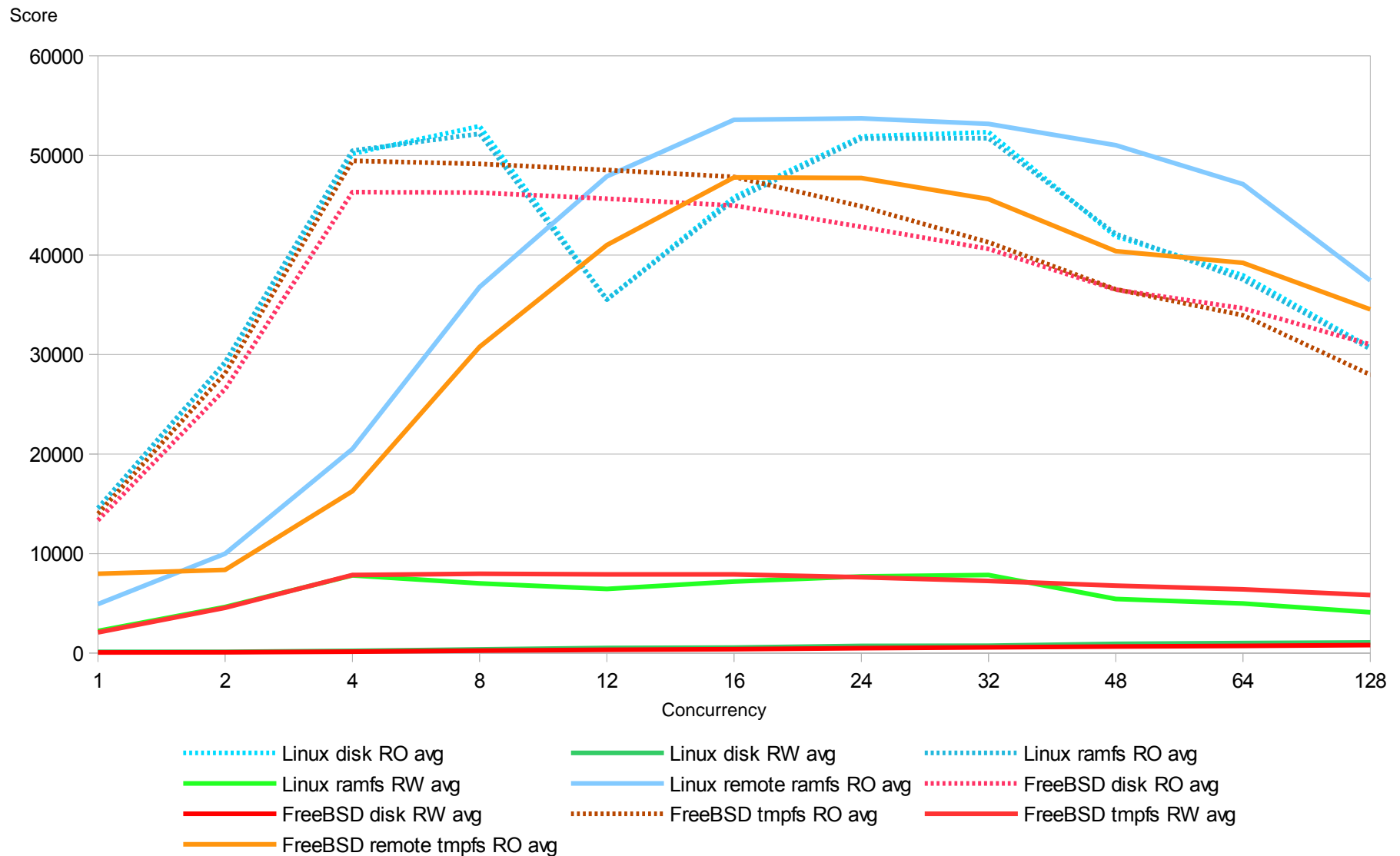
- Initialized with `-s 1000`
- 100,000,000 records, 16 GB size
  - (fits in RAM, for RO)
- PostgreSQL configuration:
  - 8 GB shared buffers
  - 32 MB work memory
  - autovacuum off
  - 30 checkpoint segments



# PostgreSQL / pgbench caveats

- WAL logging means:
  - data is written to checkpoint segments, restarts are needed between WRITE benchmarks
  - data is transferred to “proper” storage almost unpredictably, long-ish runs are needed for repeatable benchmarks
- Autovacuum can run almost unpredictably

# PostgreSQL results

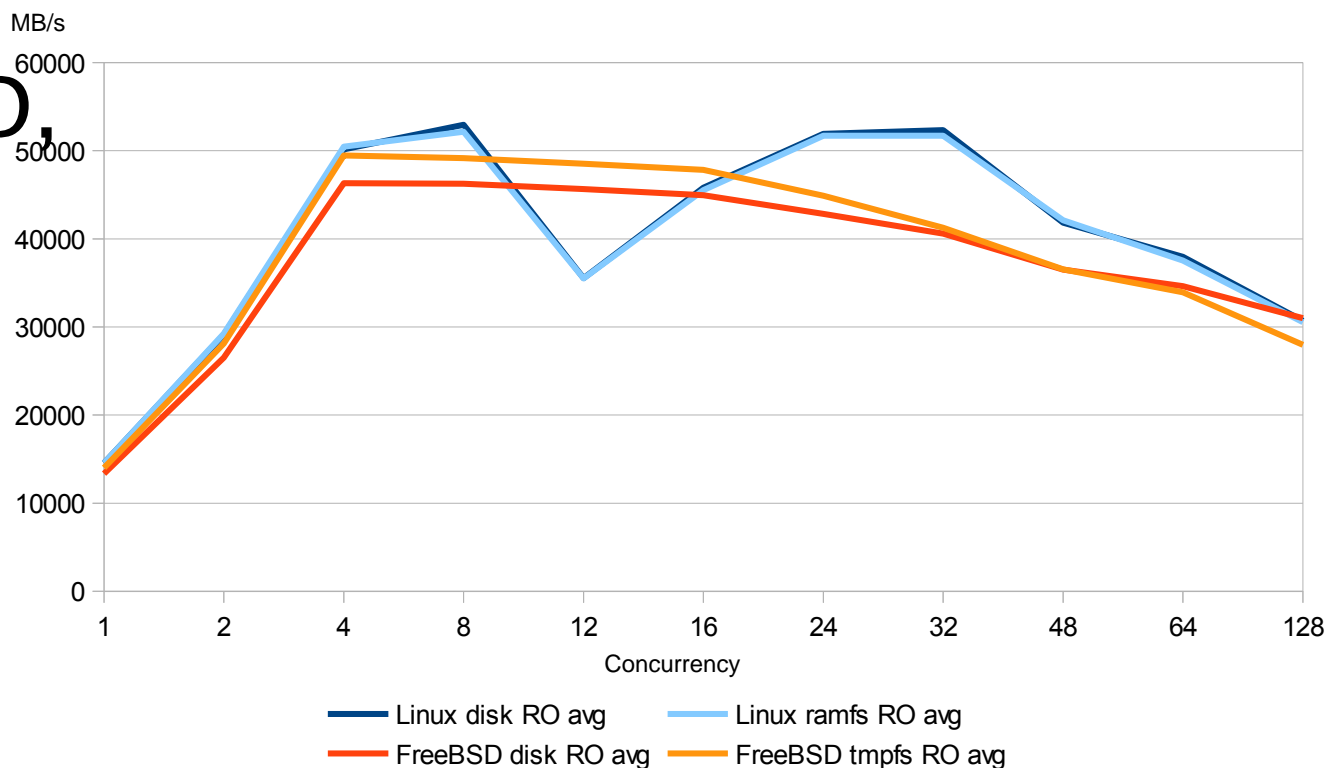




# PostgreSQL result notes

- On Linux, there's almost no difference between ext4 when data is fully cached (warmed) and ramfs

- On FreeBSD, tmpfs is faster

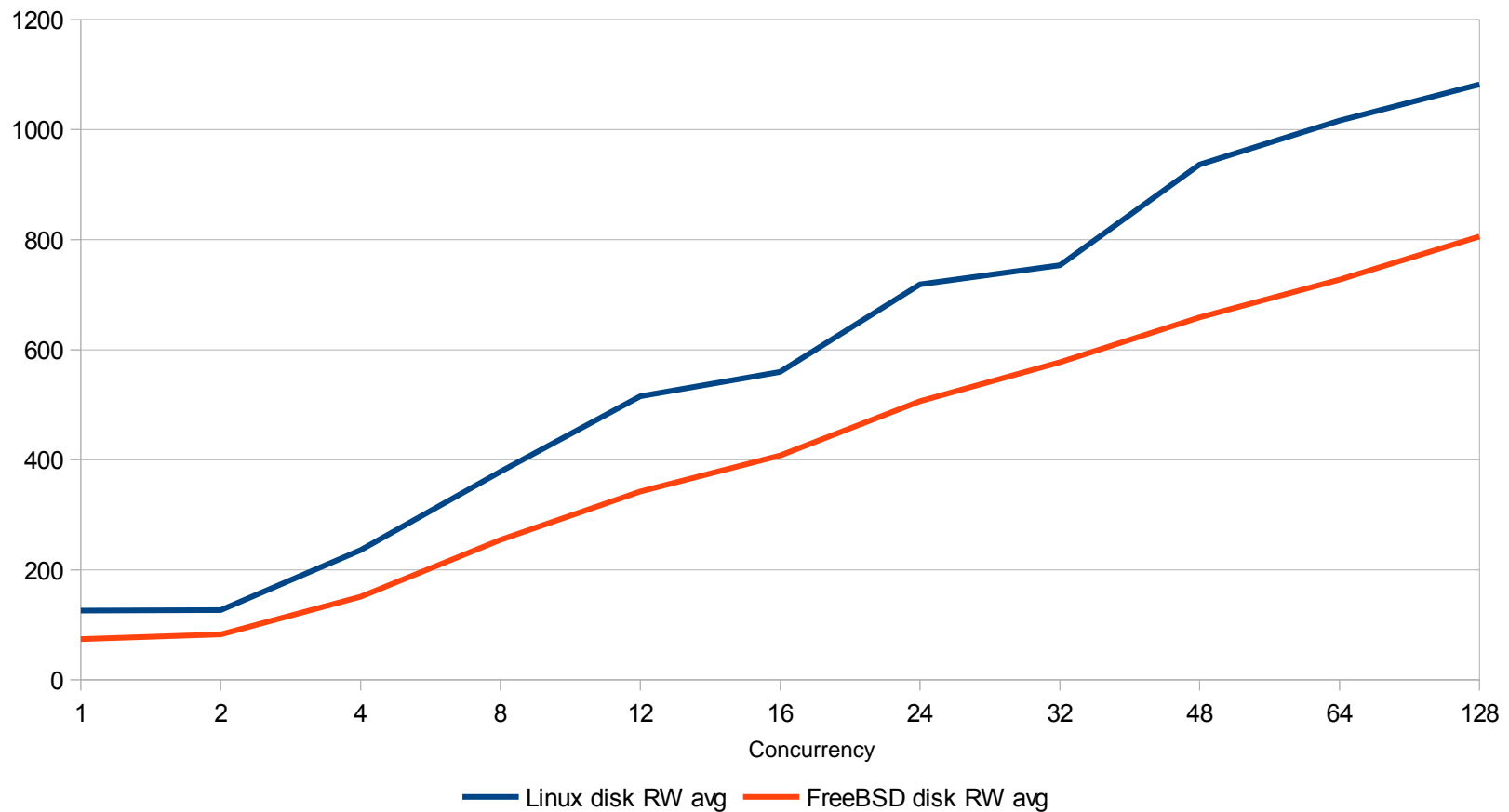




# PostgreSQL result notes

- Linux is consistently 5%–10% better

MB/s (cached)

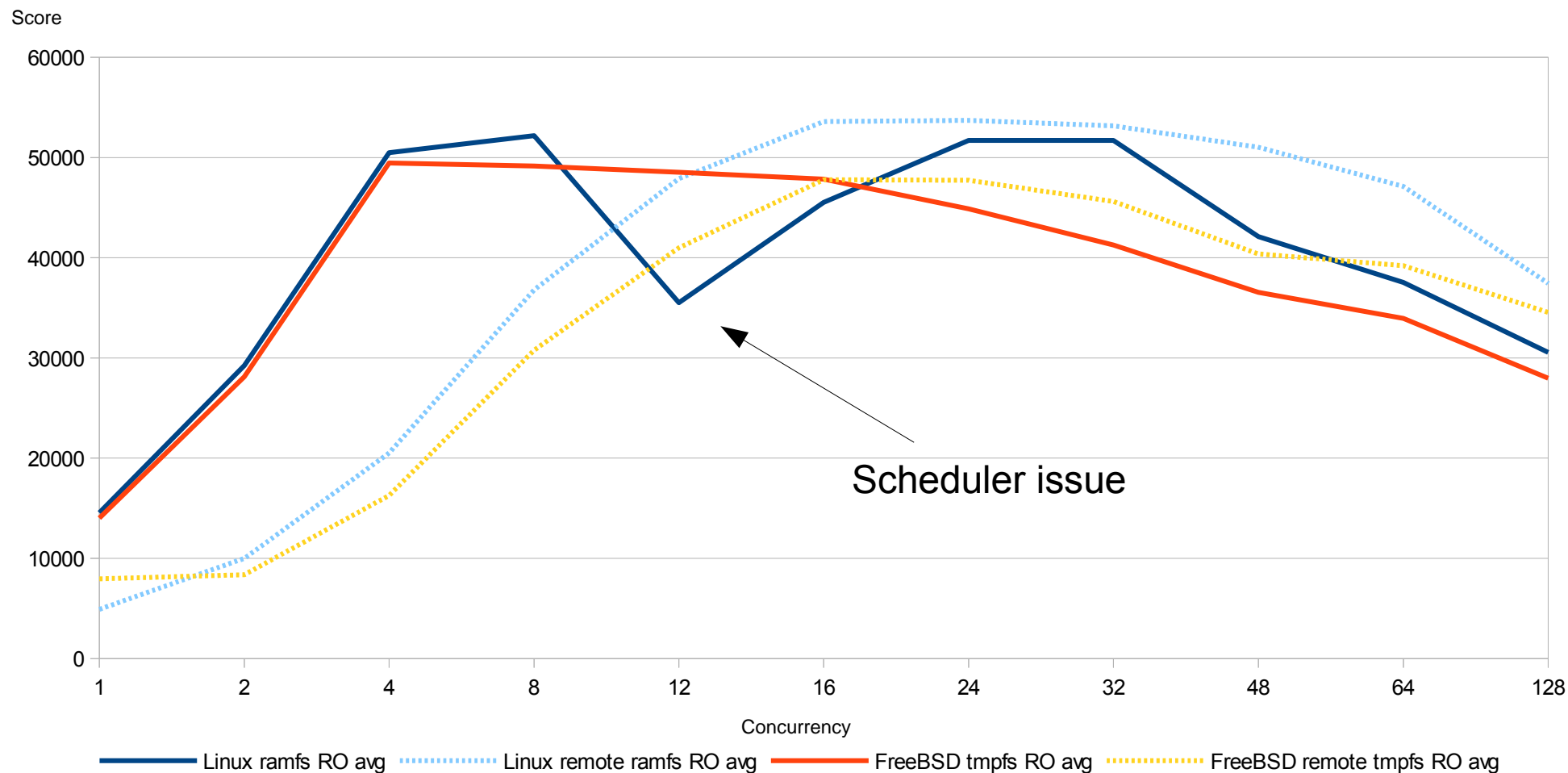






# PostgreSQL result notes

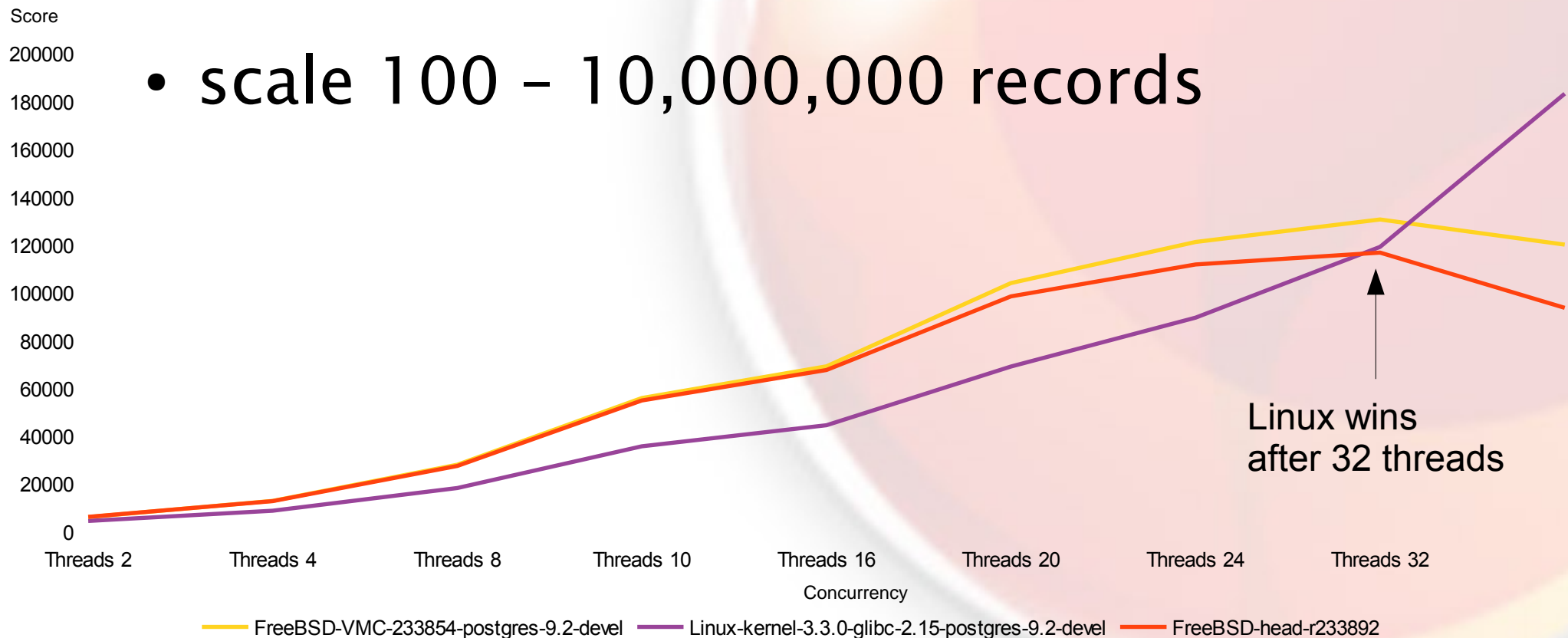
- Remote access doesn't help much





# However...

- Different benchmark, by Florian Smeets
- 40-core, 80-thread system, 256 GB RAM
- scale 100 – 10,000,000 records



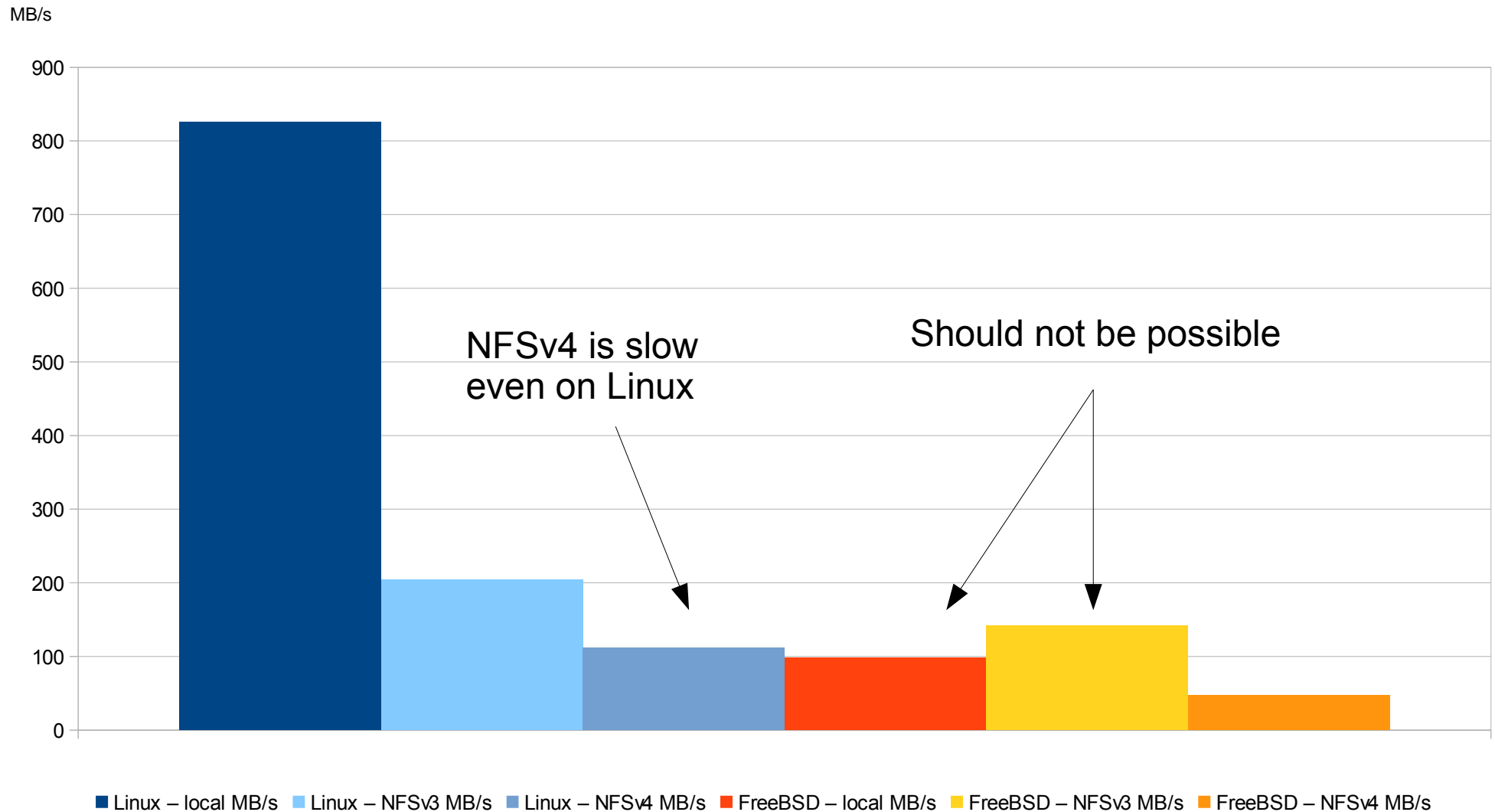


# Filebench

- Dubious correctness of the port...
- “fileserver” and “webproxy” profiles
  - **fileserver**: “Emulates simple file-server I/O activity. This workload performs a sequence of creates, deletes, appends, reads, writes and attribute operations on a directory tree. 50 threads are used by default. The workload generated is somewhat similar to SPECsfs.”
  - **webproxy**: “Emulates I/O activity of a simple web proxy server. A mix of create-write-close, open-read-close, and delete operations of multiple files in a directory tree and a file append to simulate proxy log. 100 threads are used by default.”
- Local drive, NFSv3 and NFSv4

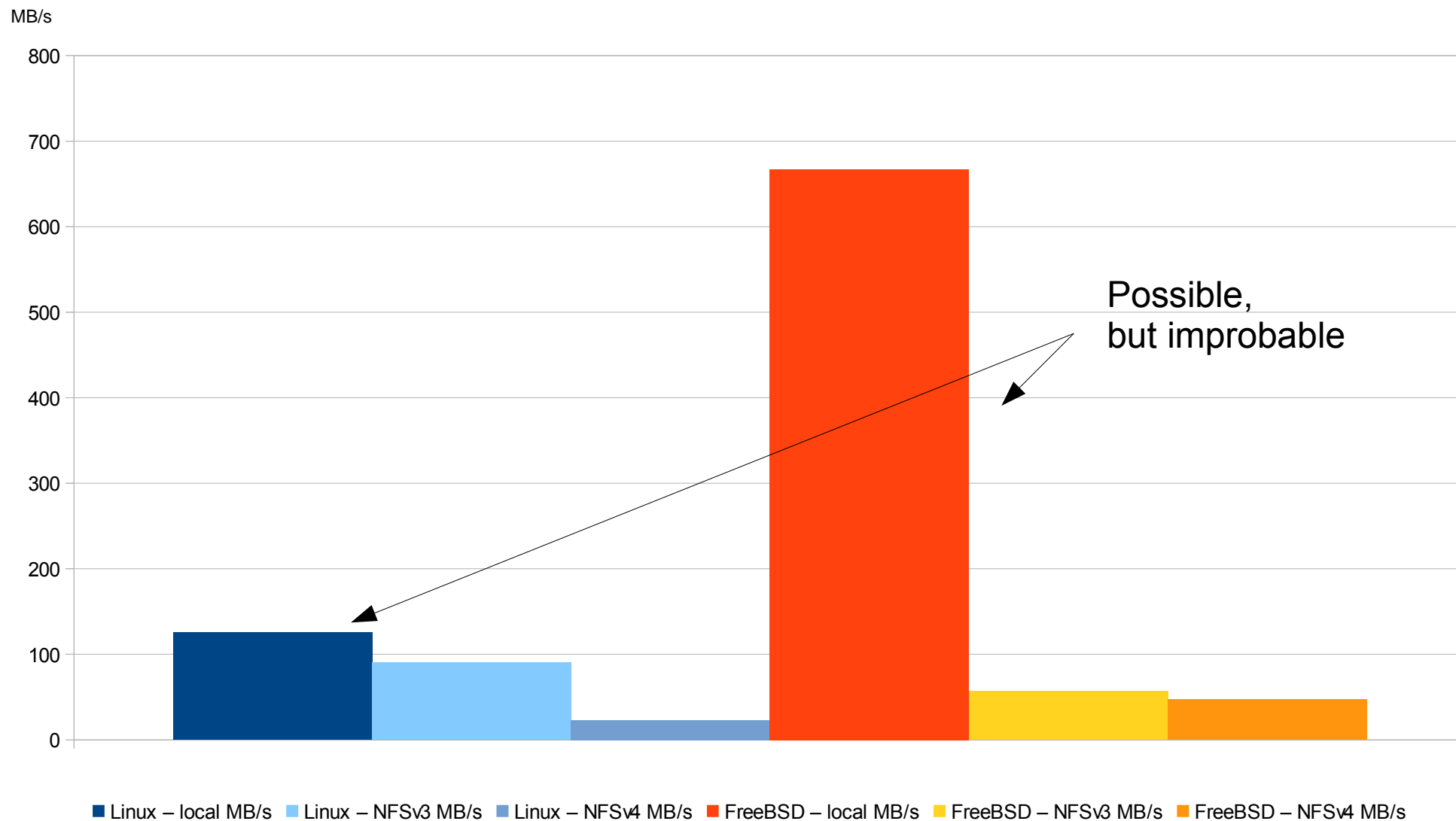


# fileserver profile





# webproxy profile





# Cross-benchmarking

- FreeBSD server, Linux client
- Linux does caching...





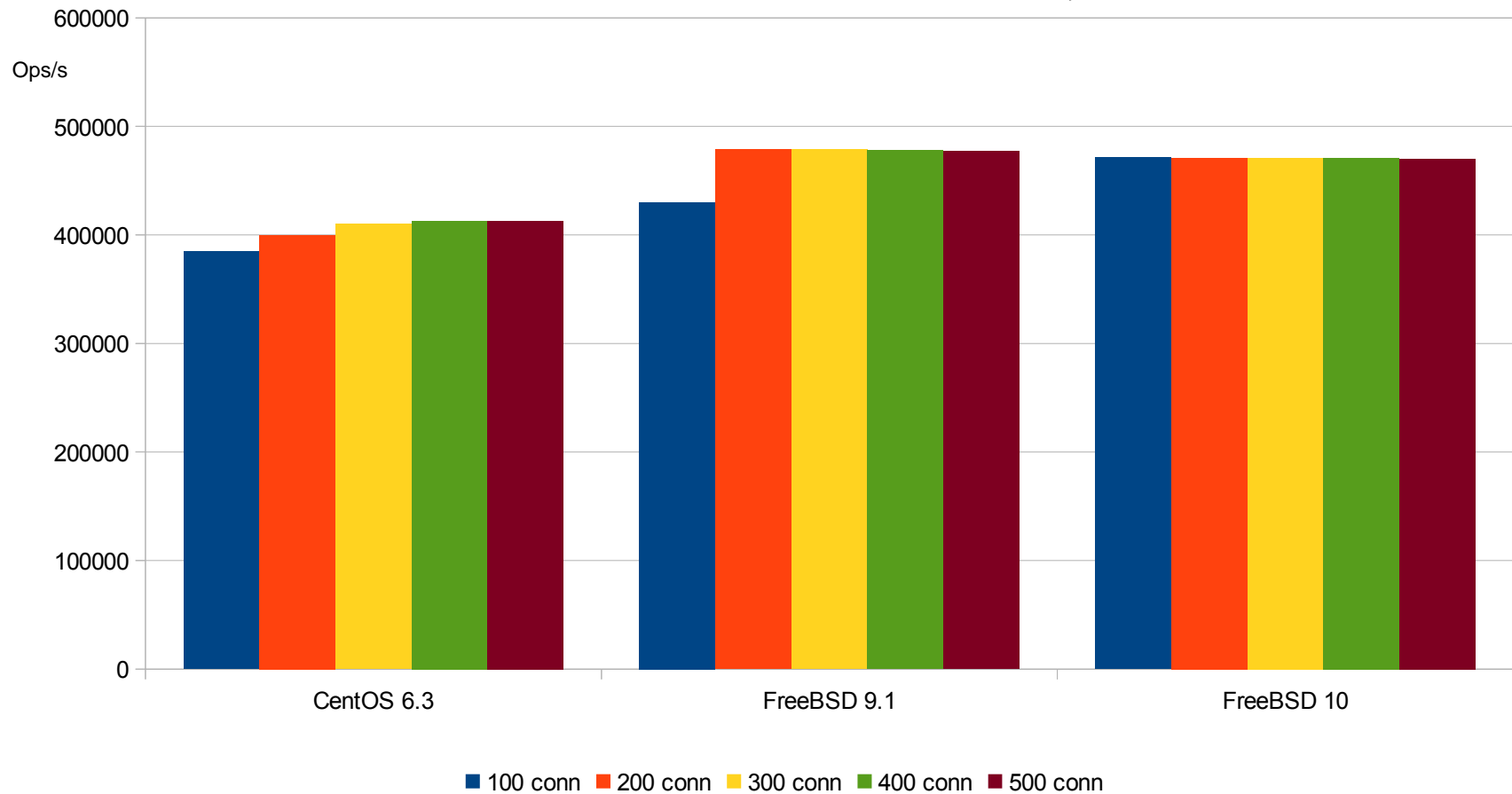


# Bullet Cache

- My own cache server, presented at BSDCan 2012 :)
- Lots of small (32 byte – 128 byte) TCP command / response transactions
- Multithreaded + non-blocking IO
- (nearly 2,000,000 TPS over Unix sockets on medium-end hardware)



- Linux support incomplete, lacks `epoll()`
- Within measurement error, 9.1 == 10





## Result notes

- TCP is fairly concurrent in FreeBSD, multiple TCP streams do not block each other
- $> 470,000$  PPS per direction
- Over Unix sockets (local): 1,020,000 TPS



## On tuning...

- The year is 2013 and FreeBSD actually auto-tunes reasonably well
- Example #1: `vfs.read_max` + `hi/lobufspace`
- Example #2: `hw.em.txd/rxd`
- Example #3: `kern.maxusers`
- Example #4: `vm.pmap.shpgperproc`



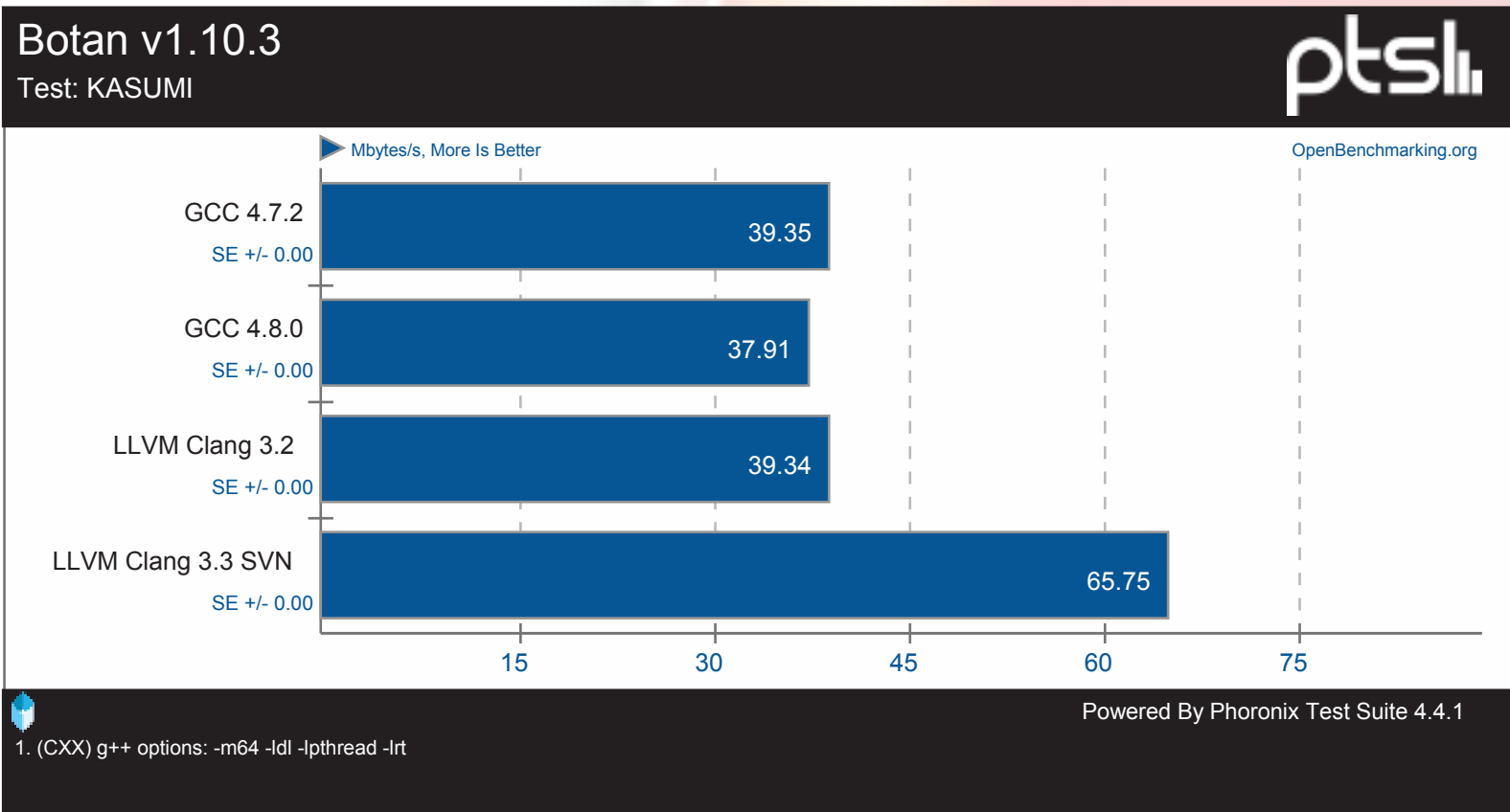
# Why no CPU benchmarks?

- You are not going to influence raw CPU performance from the OS (except in edge cases / misconfiguration)
- You can test the quality of the **libc** and **libm** implementations... but be sure that is what you want to
- You can also test the quality of compiler optimizations... if you want to.



# (LLVM vs GCC)

- (do not draw conclusions based on this)
- (Phoronix benchmark, Botan/KASUMI)







# Why NOT benchmark?

- When is benchmarking important?
  - Is performance more important than features?
  - Is it cheaper to buy another machine than to reconfigure / develop a better OS?
- The future is actually quite good
  - During 7.x timeframe: 8 CPU scalability
  - During 10.x timeframe: 32 CPU scalability



# When you do need to benchmark?

- As a sysadmin, you just want to know
- Planning / budgeting for a project
- When your boss tells you to :)
- Advocacy...



# Continuous benchmarking?

- Some talks were had...
- It would probably be easier to set up now after the developer cluster has been updated...
- Performance regression monitoring



**The End**

# **Benchmarking FreeBSD**

Ivan Voras <[ivoras@freebsd.org](mailto:ivoras@freebsd.org)>