LinkedIn™

Java hates Linux. Deal with it.

Greg Banks

Staff Site Reliablity Engineer
LinkedIn
www.linkedin.com/in/gregnbanks

This is a story about Java and Linux

Java and Linux are a perfectly matched pair

Except when they're not

Then it's fireworks

@misslexirose

Java: a portable application platform

supposedly

Alternatively: Java is written for an imaginary OS

Which is sorta like the set-top box Java was designed for

Then shoe-horned into Linux, Solaris & Windows with hidden portability layers

Kevin Fream adroit.blog

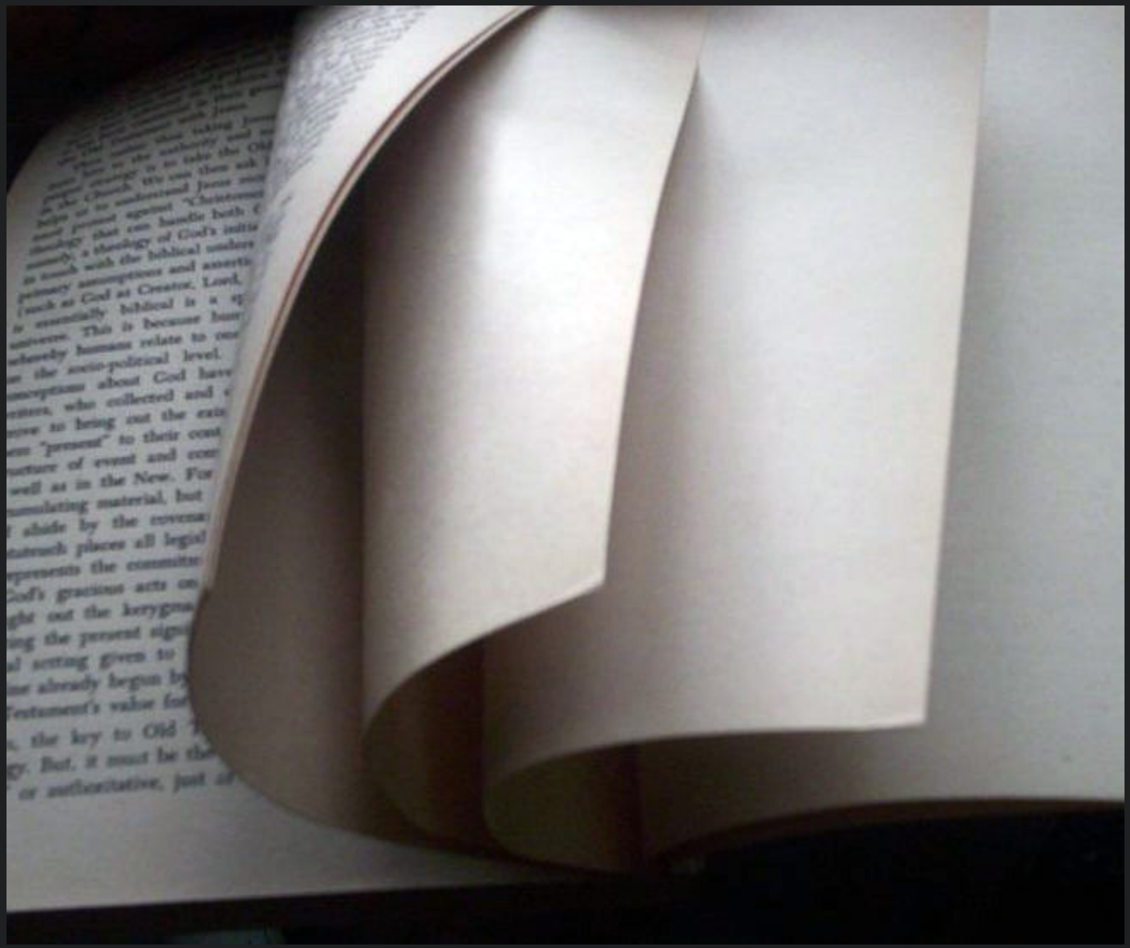
Garbage Collection vs Unix Virtual Memory

Garbage collection vs virtual memory

www.mckusick.com

Since 3BSD (1979) every Unix-like system's virtual memory subsystem has been designed around *locality of reference*

Locality of reference



A process’ memory space is managed in fixed size *pages*

A program uses a *working set* of pages frequently

Other pages are hardly ever used. They can be *swapped out* to disk, saving previous RAM

Java behaves just like that

…until GC happens

Javaworld.com

Every page in the Java heap - a multi-GB chunk of virtually contiguous address space - is read and written as fast as possible.

While the application is *stopped*.

This has to be milliseconds fast

Daytrip to Failtown

If enough of the Java heap is swapped out, GC can take *hundreds of seconds*

Your service's clients time out

Healthchecks fail

Latency spikes propagate

Your SLAs are shot

Real World Example

gc.log

2014-10-15T18:42:44.931+0000: 4651814.348: [GC 4651814.546: [ParNew

...

: 1152488K->274696K(1572864K), 106.7471350 secs] 15021460K->14143829K(32944128K), 106.9300350 secs] [Times: user=53.97 sys=518.37, real=107.11 secs]

Total time for which application threads were stopped: 107.5402200 seconds

Why? Swap is SLOW

Nationalgeographic.com

"If you're swapping out, you've already lost the battle"

Swapping in happens one page at a time

Swap has none of the tricks used to make filesystems fast (readahead, contiguous extents)

Deal with it

The easy way

Disable swap entirely

Remove /etc/fstab entries

Check with swapon –s

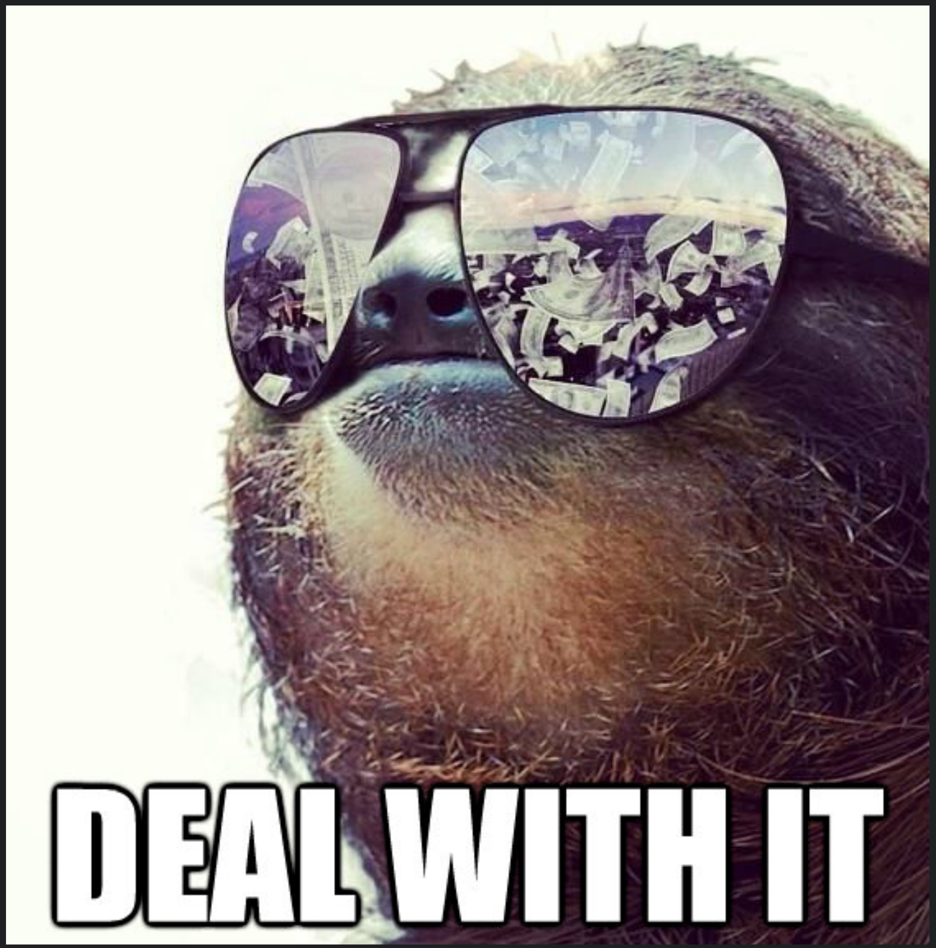
Affects all processes

OS-wide configuration change

We couldn’t do this for $reasons

Deal with it

The clever way



sysctl –w \

vm.swappiness=0

Well-known kernel tunable to limit swapping

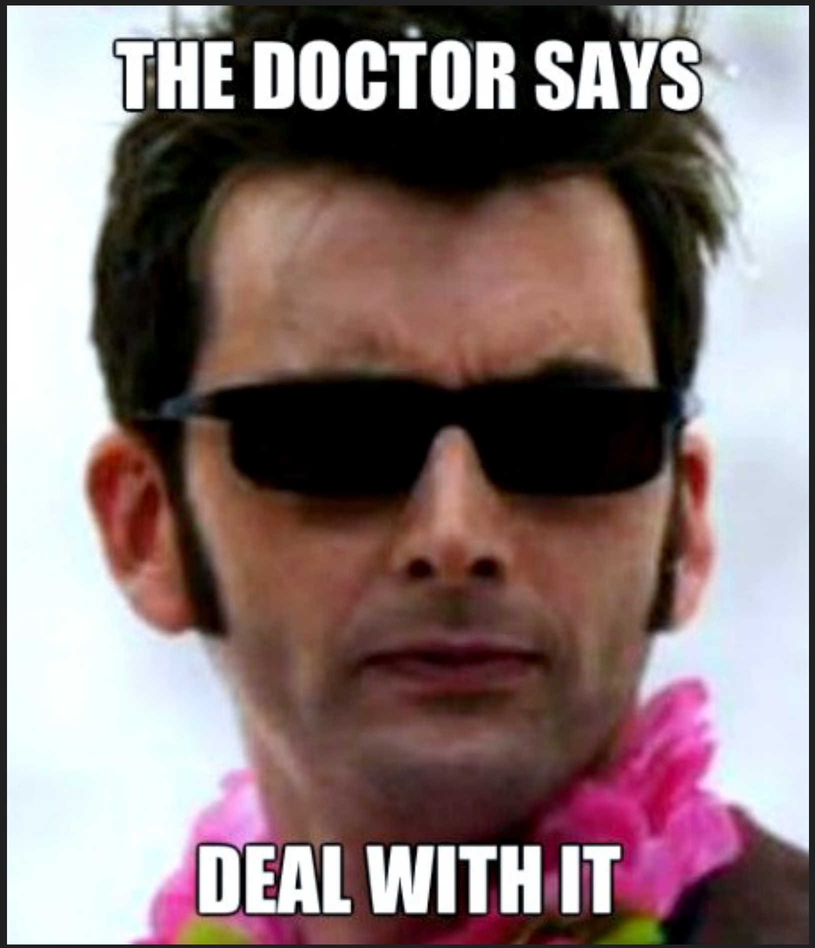
Doesn’t eliminate swapping

Behavior depends on kernel version

It just didn’t work

Deal with it

The horrible/cunning way



Lock the Java heap in RAM

A native system call using com.sun.jna

mlockall(MCL_CURRENT)

Call early in process lifetime

No more swapping

Resource Limits

Stephen Codrington CC BY 2.5

RLIMIT_MEMLOCK limits how much memory a process can lock

Kernel default is 64K, needs raising

```
echo \
"app - memlock unlimited" \
>>/etc/security/limits.conf
```

You could also give the process the CAP_IPC_LOCK capability

Checking it worked

For a process with ~32G heap (-Xmx32684m)

egrep '^Vm(Lck|RSS|Size|Swap)' /proc/$pid/status

VmSize: 48767632 kB

VmLck: 39652024 kB ← needs to be > heap size

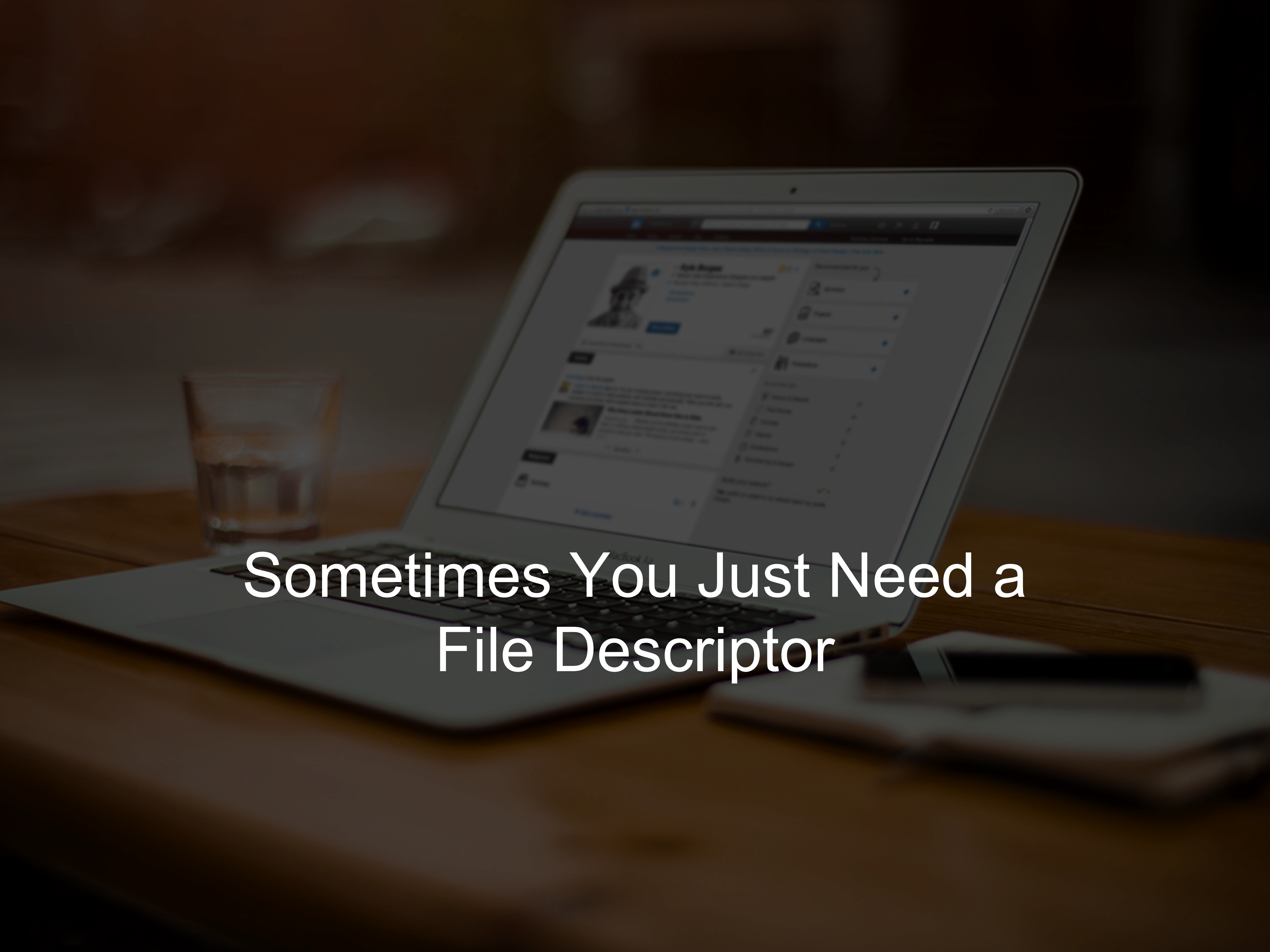
VmRSS: 35127672 kB

VmSwap: 26444 kB ← needs to be small

The whole heap is locked and resident

Some non-heap allocations came along for the ride

Others are eligible for swapping



Sometimes You Just Need a File Descriptor

Java Hides the OS

So you can write portable code

Also … so you can *only* write portable code

But sometime you need access to the underlying OS objects

Coachingbysubjectexperts.com

File Descriptors

Stationeryinfo.com

File descriptors are the small integers used to name open files and sockets to Unix system calls.

int read(**int fd**, void*buf, int len)

Java hides these from you because they're different on Windows.

The Goal

Gateway to the Rabbit Hole

Jenningswire.com

Network server performance analysis

Need the length of a socket's in-kernel input queue

Not a complicated example, but outside Java's API fence

Doing it in C

```
FILE *stream = …
int length;
int r = ioctl(fileno(stream), FIONREAD, &length);
/* error handling */
```

Doing it in Java

…because the build system cannot cope with mixed C & Java

First we have to convince Java to let us call ioctl()

```
import com.sun.jna.Native;
private static native int ioctl(int fd, int cmd, IntByReference valuep) throws LastErrorException;
Native.register(…)

private static final int FIONREAD = 0x541B;
```

Doing it in Java

part 2

Now we just need the Java equivalent of C's fileno()

Given an InputStream object return the Unix file descriptor

So simple…so impossible

FileDescriptor is Wrapped Tight

http://www.iconsdb.com/soylent-red-icons/private-4-icon.html

Class FileInputStream has

public FileDescriptor getFD();

And FileDescriptor has

private int fd;

But **NO WAY TO GET IT**

How to Unwrap a FileDescriptor ?

literateforlife.org

Well known trick using Reflection

…performance concerns

Tricks using Unsafe…are unsafe

sun.misc.SharedSecrets

This is the 2nd of the big ol' rugs under which Sun swept all the dust bunnies

SharedSecrets

```
public static JavaIOFileDescriptorAccess
getJavaIOFileDescriptorAccess();

public interface JavaIOFileDescriptorAccess  {
    public int get(FileDescriptor fd);
    public long getHandle(FileDescriptor obj);
}
```

Deal With It: Simple, Fast, Obscure

```
import sun.misc.SharedSecrets;

public static int getFileDescriptor(FileDescriptor fd) … {
 return SharedSecrets.getJavaIOFileDescriptorAccess().get(fd);
}

public static int getFileDescriptor(InputStream is) … {
 return getFileDescriptor(((FileInputStream)is).getFD());
}
```

NFS, Big Directories, Oh My

The Problem

Offtackleempire.com

We take database backup snapshots and copy them to a directory on an NFS filer.

Directory has 1000s of files

A Java process lists this directory and reports file names and sizes

SLOOOOWLY

How NFSv3 Works

when we run "ls"

Gxdmtl.com

A process on the NFS client wants to read the contents of a directory

getdents() system call

NFS client sends READDIR rpc to the server, caches result

The READDIR call

greatly simplified

Theodysseyonline.com

Maps to getdents()

Arguments

nfs_fh3, count

Results

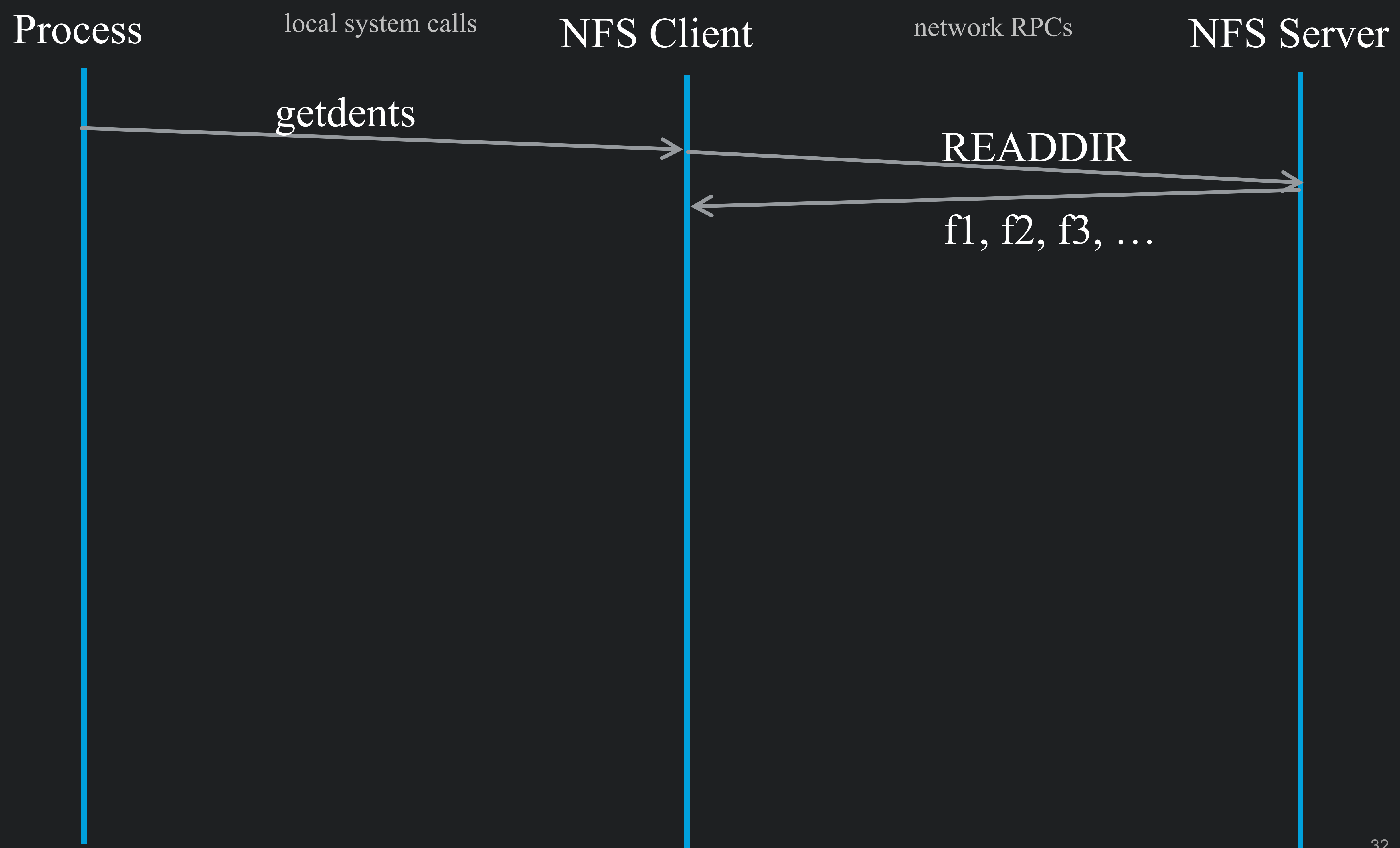
list of {

fileid3 fileid; // inode #

filename3 name; // string

}

Message Flow for "ls /d"



The Trouble With READDIR

The "ls -l" problem

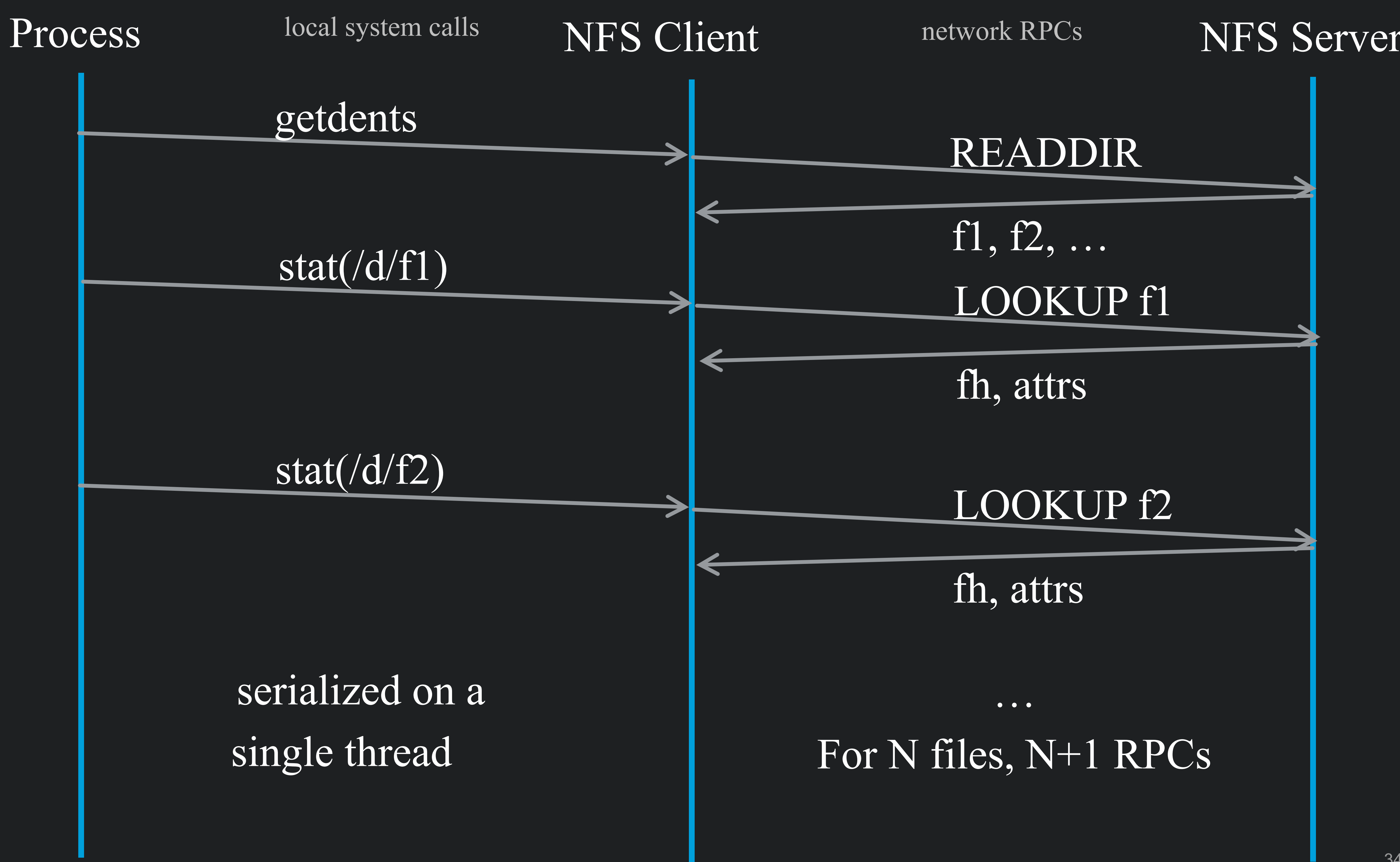
Reference.com

Only returns names, not file attributes or a file handle

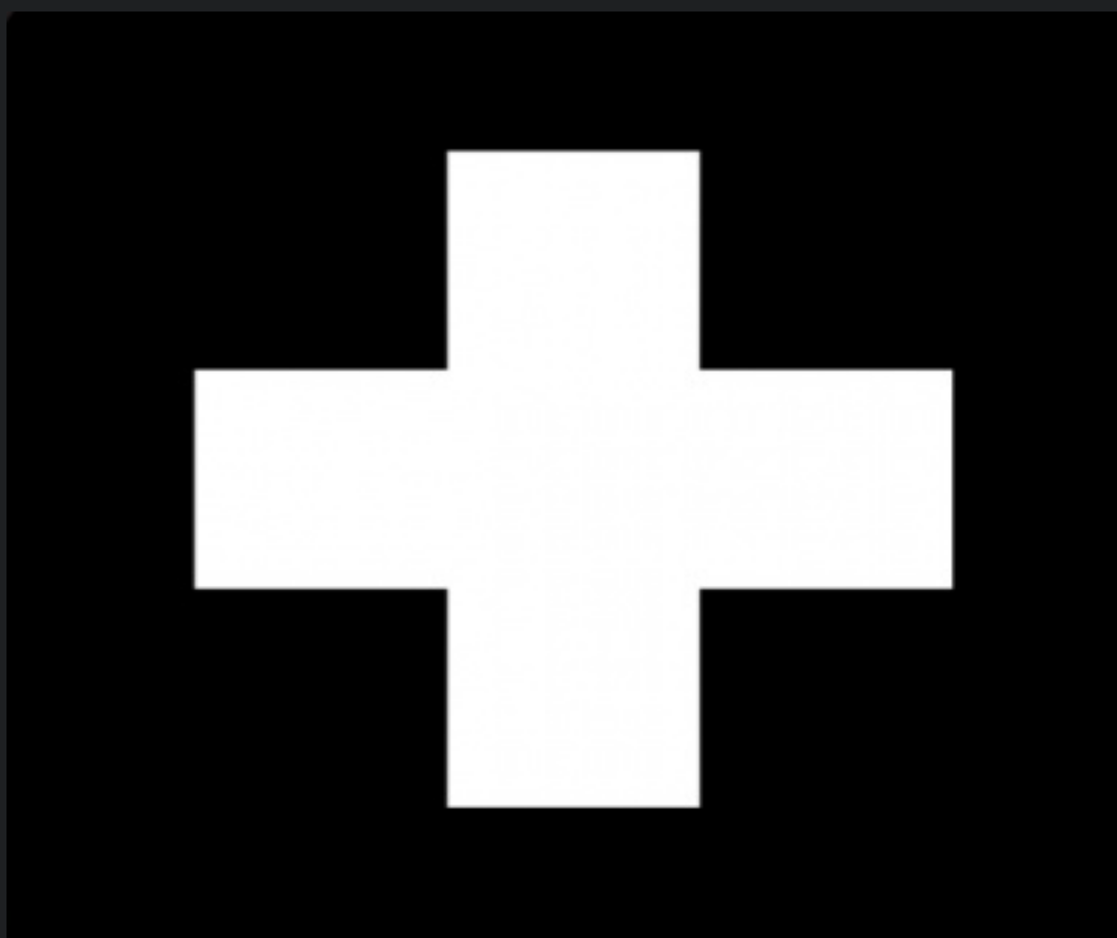
If the process does a stat() or open() system call on the files, the NFS client needs to do a LOOKUP rpc, maybe GETATTR

Those RPCs are *one per file*

Message Flow for "ls -l /d"



The READDIRPLUS call



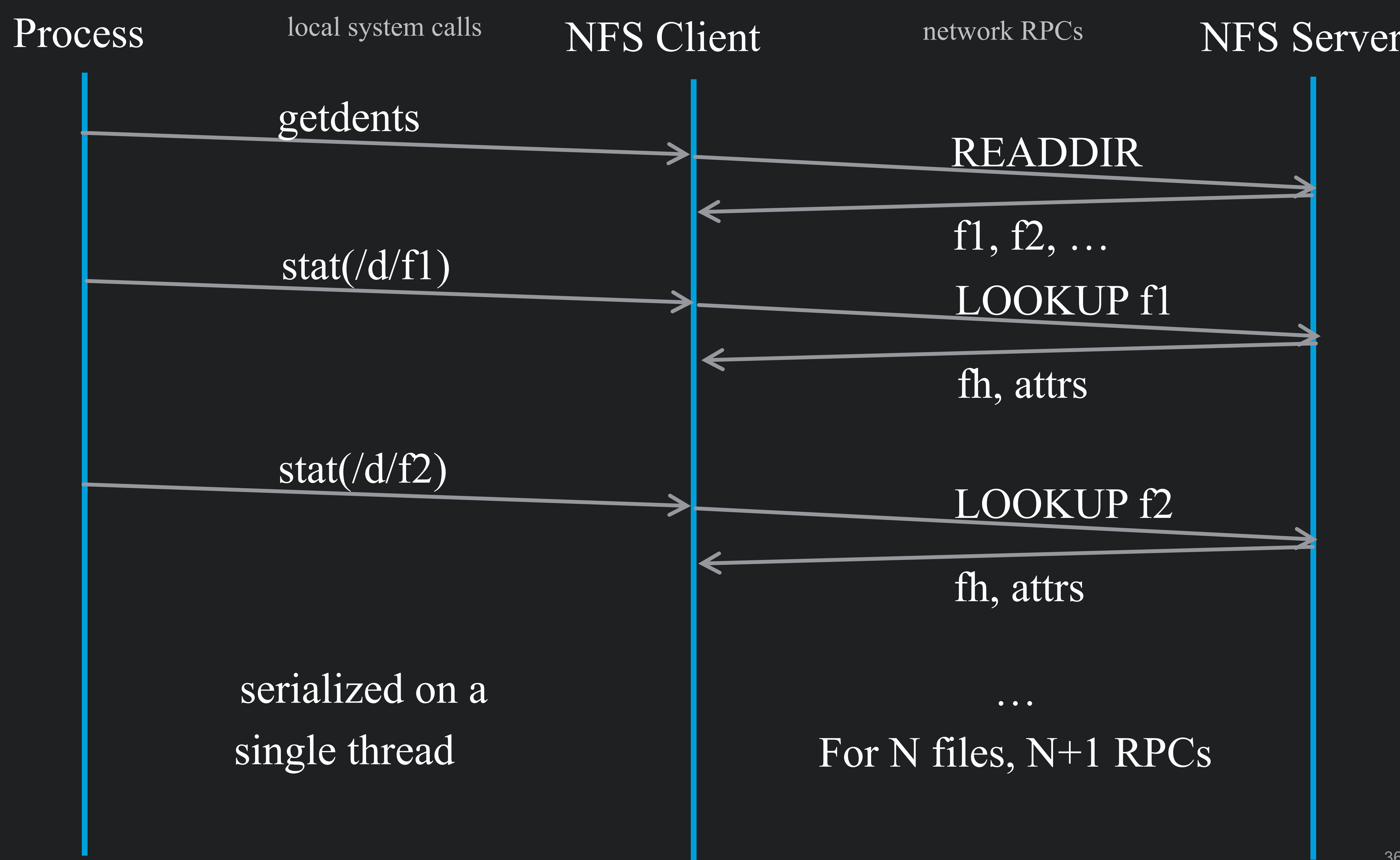
Clipartkid.com

READDIR but returns file handles and attributes for each file.

Frontloads the information needed for the process to stat() or open() a file.

Call Sequence for "ls -l /d"

with READDIRPLUS



So we just use READDIRPLUS right?

If only it were that easy

Iconfinder.com

The NFS protocol puts an upper limit on the encoded size of the results

A really big directory, needs more READDIRPLUS than READDIR

Tradeoff: READDIR is faster with large directories if you don't want to stat() the files.

Heuristics To The Rescue

111emergency.co.uk

On the first getdents() send a READDIRPLUS

If the process open()s or stat()s set a flag

On subsequent getdents() if the flag is set, send READDIRPLUS else READDIR

These Patterns Are Optimal

"ls" = processes which do

getdents → READDIRPLUS

getdents → READDIR

getdents → READDIR

"ls –l" = processes which do

getdents → READDIRPLUS

stat, stat, … (cached)

getdents → READDIRPLUS

stat, stat, … (cached)

getdents → READDIRPLUS

stat, stat, … (cached)

"ls -l /d" in Java

This is the solution you will find on StackOverflow


```
File dir = File("/d");
File[] list =
        dir.listFiles();
for (File f: list) {
  print(f.getName(),
        f.length());
}
```

Steven James Keathley

java.io.File

Stationeryinfo.com

Wraps a string filename

File.length() → stat()

File.listFiles() reads the whole directory using N x getdents(), returns File[]

The API forces this behavor

"ls -l": C vs Java

on large directories

C processes do

getdents → READDIRPLUS

stat, stat, … (cached)

getdents → READDIRPLUS

stat, stat, … (cached)

getdents → READDIRPLUS

stat, stat, … (cached)

Java processes do

getdents → READDIRPLUS

getdents → READDIR

getdents → READDIR

stat, stat, … (cached)

stat → LOOKUP

stat → LOOKUP

…

100s x slower

Deal With It: Rewrite Using java.nio.files

new in Java 8

```
Path dir = Paths.get("/d");
DirectoryStream<Path> stream =
      Files.newDirectoryStream(dir);
for (Path p: stream) {
  File = p.toFile();
  print(f.getName(), f.length());
}
```

The iterator object delays the getdents()until they're needed

Understatement

Theflyingtortoise.blogspot.com

"[newDirectoryStream] may be more responsive when working with remote directories"

-- Java 8 Documentation

GC: Log Hard
With a Vengeance

Java GC is Important

HearMeSayThis.org

GC is a limiting factor on the availability of Java services

In production, it's important to keep GC behaving well

"You can't manage what you can't measure"

→ GC logging in production.

GC Logging Options We Use

Hank Rabe hankstruckpictures.com

-Xloggc:$filename

-XX:+PrintGC

-XX:+PrintGCDetails

-XX:+PrintGCDateStamps

-XX:+PrintGCApplicationStoppedTime

-XX:+PrintGCApplicationConcurrentTime

-XX:+PrintTenuringDistribution

More data is better, right

What gets logged?

Major GC event

```
2017-03-12T23:19:01.480+0000: 3382210.059: Total time for which application threads were stopped: 0.0004373 seconds
2017-03-12T23:22:01.186+0000: 3382389.765: Application time: 179.7055362 seconds
2017-03-12T23:22:01.186+0000: 3382389.766: [GC (Allocation Failure) 3382389.766: [ParNew
Desired survivor size 134217728 bytes, new threshold 15 (max 15)
- age   1:     998952 bytes,     998952 total
- age   2:      12312 bytes,    1011264 total
- age   3:      10672 bytes,    1021936 total
- age   4:      10480 bytes,    1032416 total
- age   5:     753312 bytes,    1785728 total
- age   6:      11208 bytes,    1796936 total
- age   7:     149688 bytes,    1946624 total
- age   8:       9904 bytes,    1956528 total
- age   9:      11000 bytes,    1967528 total
- age  10:      10136 bytes,    1977664 total
- age  11:      10184 bytes,    1987848 total
- age  12:      10304 bytes,    1998152 total
- age  13:      92360 bytes,    2090512 total
- age  14:     176648 bytes,    2267160 total
- age  15:      70328 bytes,    2337488 total
: 539503K->10753K(786432K), 0.0073635 secs] 2119127K->1590653K(3932160K), 0.0075104 secs] [Times: user=0.09 sys=0.00, real=0.01 secs]
```

Byte & Line Rate of Logging

Visitmelbourne.com

Major GC up to 1200 B

Minor GC ~200 B

	Light Load	Heavy Load
Bytes/day	3.4 MB/d	33.0 MB/d
Lines/day	39 KL/d	543 KL/d

The Problem

Eureferendum.com

The GC log is written from JDK's C code in Stop-The-World

No userspace buffering. Every line is a write() and an opportunity to block in the kernel

If the root disk is heavily loaded, the long tail latency can be 1-5 sec

Client timeouts typically 1-5 sec

Possible Solutions

Disable GC logging. Flying blind.

Log4j 2.x async logging. GC iog is written from C code

Log to a named FIFO. If the reader process dies the app is blocked.

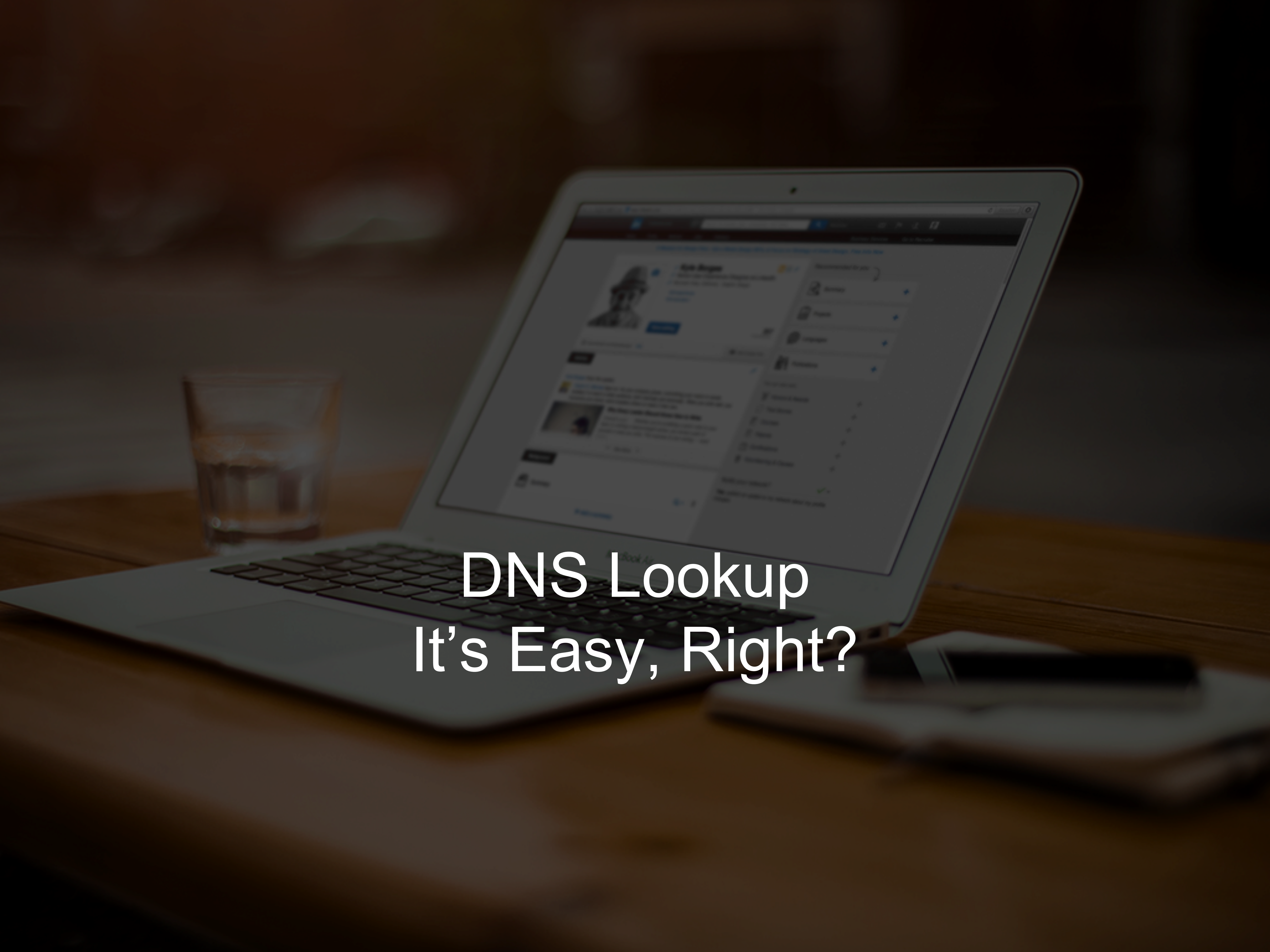
En.oxforddictionaries.com

Deal With It: Log To FUSE

Log to a file mounted on a FUSE fileystem.

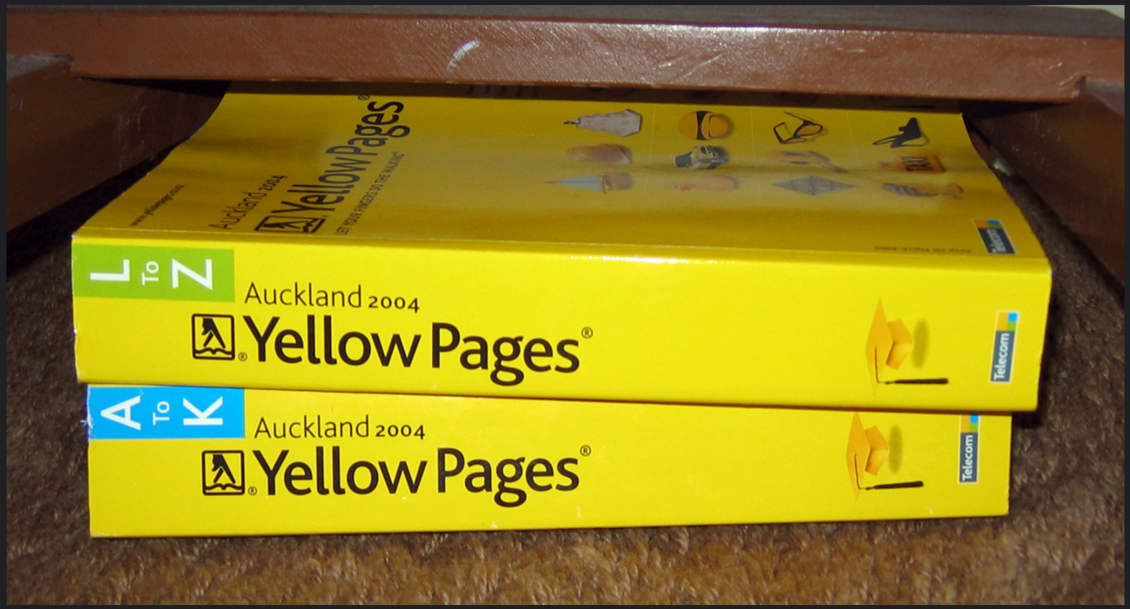
The filesystem's daemon accepts writes and queues them in userspace, writes to disk. Provides asynchrony.

If the daemon dies, app `write()`s fail immediately with `ENOTCONN`, app continues.


DNS Lookup
It’s Easy, Right?

Java Hostname Lookup

Java makes this easy




```
String host = "www.example.com";
int port = 80;
Socket sock = Socket();
Sock.connect(host, port); ←
```

Peeling The Onion

High Mowing Seed Company

The easy APIs are layers of wrappers around

```
class InetAddress {
public static InetAddress[] getAllByName(String host) …
```

By default calls libc's getaddrinfo()

Which calls Linux's NSCD

Linux NSCD

Name Service Cache Daemon

Theodysseyonline.com

Standard (in the glibc repo)

Runs locally on every box

Configurable. Supports multiple name service providers, like a DNS client.

Understands & obeys TTL

Java's In-Process Cache

InetAddress conveniently caches the results in RAM

For 30 sec. Ignores the TTL returned from DNS

Java could have gotten this right

int __gethostbyname3_r(const char *name, …, int32_t *ttlp, …);

Why Does Java Do This?

Hostname Resolver Latencies

Norfolkwildlifetrust.co.uk

Java cache 53 ± 2 µs

NSCD 72 ± 3 ms

DNS server 192 ± 8 ms

Measured in production using a specially written Java program.

The Problem: Long TTLs

Yathin S Krishnappa CC BY-SA 3.0

Most stable A/AAAA records are configured with a TTL of 1h or 1d

Talking to NSCD every 30 sec is wasteful

Worse Problem: Short TTLs

spirit-animals.com

One way of achieving load balancing is with Round-Robin DNS

Some implementations rely on an ultra short TTL << 30sec to achieve failover

Deal With It: Disable Java's Cache

easy but impactful

Add to
$JAVA_HOME/lib/security/java.security

networkaddress.cache.ttl=0

or (older)
sun.net.inetaddr.ttl=0

This is global to the host

Deal With It: Bypass Cache using Reflection

Reflection is almost never the answer

```
static InetAddress[] getAllByNameUncached(String hostname) {
  Field field = InetAddress.class.getDeclaredField("impl");
  field.setAccessible(true);
  Object impl = field.get(null);
  Method method = null;
  for (Method m : impl.getClass().getDeclaredMethods()) {
    if (m.getName().equals("lookupAllHostAddr")) {
      method = m;
      break;
    }
  }
  method.setAccessible(true);
  return (InetAddress[]) method.invoke(impl, hostname);
}
```

Deal With It: DNS With JNDI

least worst option

Use com.sun.jndi.dns to make your own DNS requests for A/AAAA records

Quick, easy

Does not affect other lookups in the same process

Bypasses nscd entirely; all lookups talk to DNS server.

Deal With It: DNS SRV Records with JNDI

very useful and very hard

Use com.sun.jndi.dns to make your own DNS requests for SRV records

Need to handle stale entries

Have to parse out SRV response

Does not affect other lookups in the same process

Bypasses nscd entirely; all lookups talk to DNS server.


…and the Lessons Learned

The Two Hardest Things in CS

1. Naming
2. Cache Invalidation
3. Off By One Errors

The Three Hardest Things in CS

1. Naming
2. Cache Invalidation
3. Off By One Errors
4. Making Java Behave Rationally

Java Is Not Magic

Anne Zweiner

Understand that Java doesn't always do the right thing with your OS.

Horrible Things Live In The Corners

Thekidsshouldseethis.com

Modern software is complicated and has corner cases. Bad bad horrible things live there.

Portable code is nice

Working code is better

Do what is needful

Do not be afraid to subvert Java's portability fascism

Know your OS

rickele @ flickr.com