NFS Tuning for Oracle: Introducing DTrace

Kyle Hailey

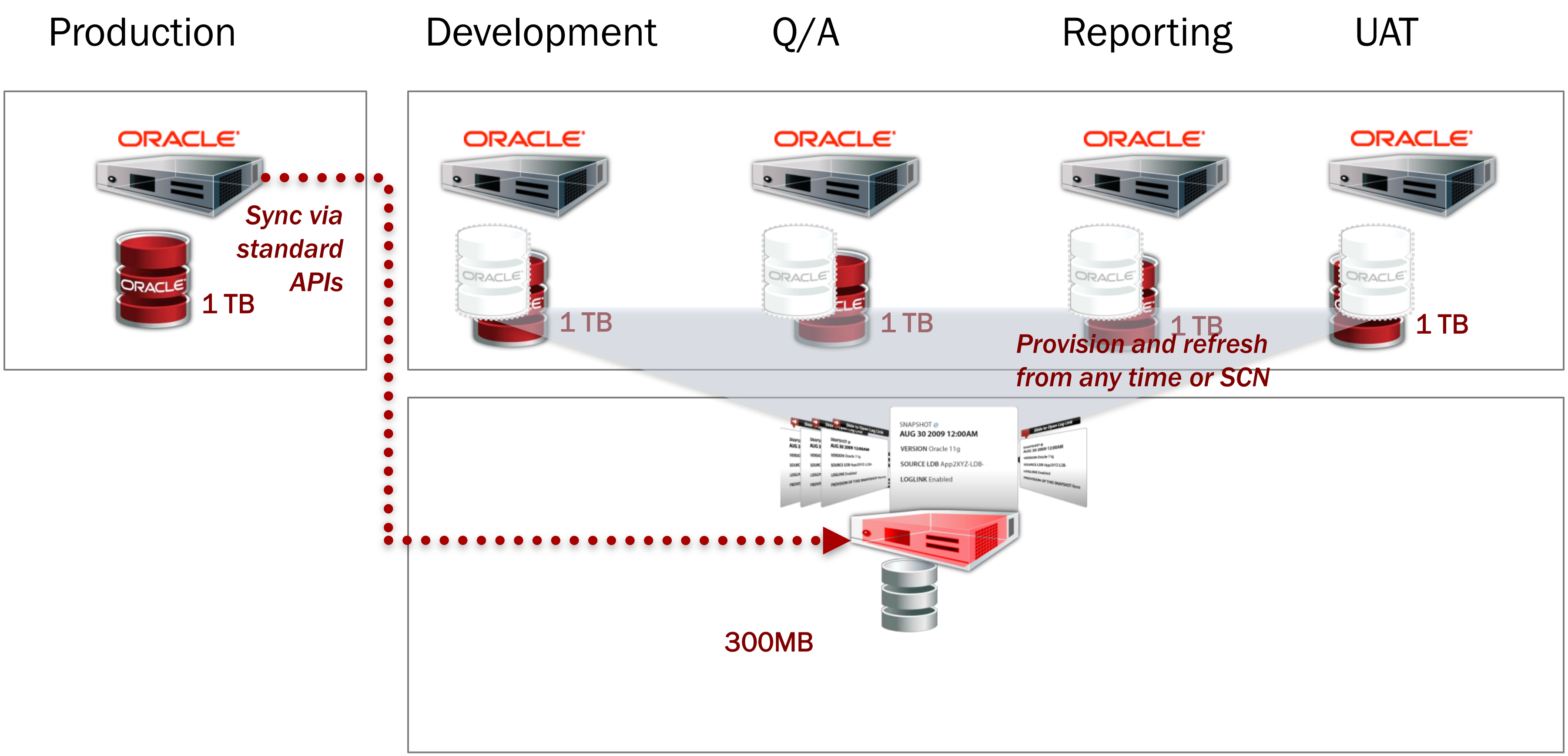
http://dboptimizer.com

June 2011

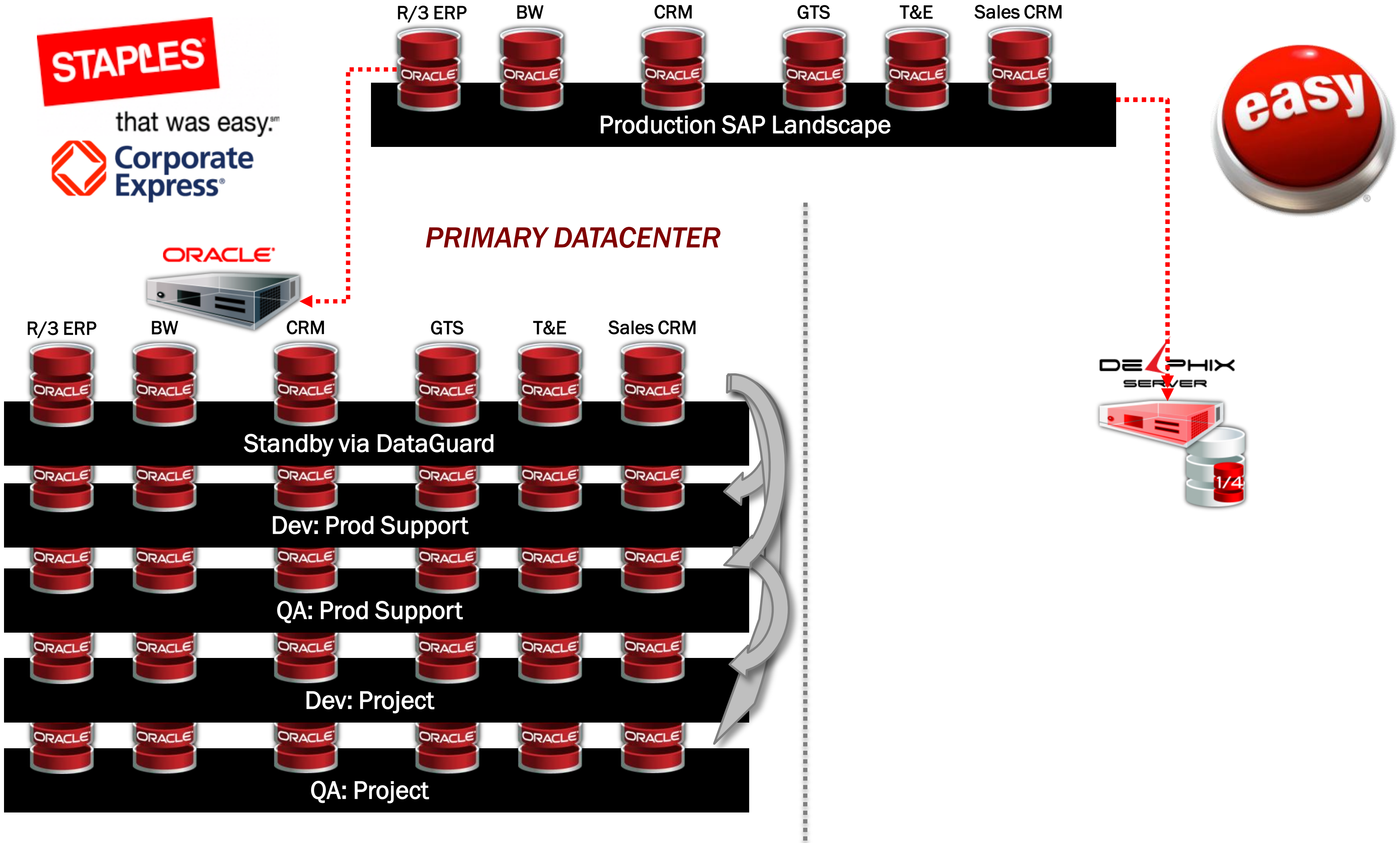
Intro

- Who am I
  - why am I interested in NFS tuning for Oracle?
- DAS vs NAS vs SAN
  - Throughput
  - Latency
- NFS configuration issues for non-RAC, non-dNFS
  - Network topology
  - TCP configuration
  - NFS Mount Options

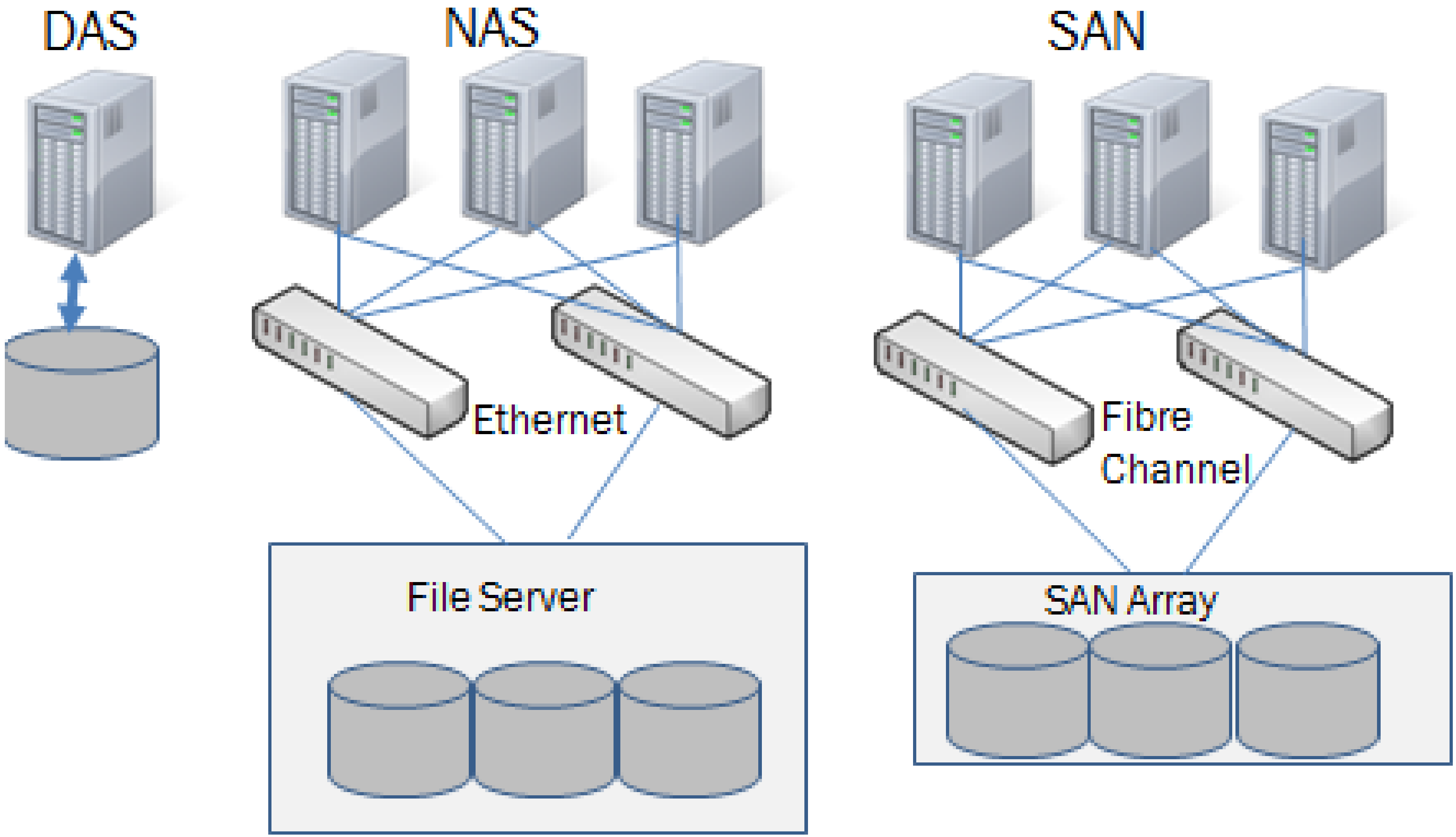
Fast, Non-disruptive Deployment



Combine Prod Support and DR

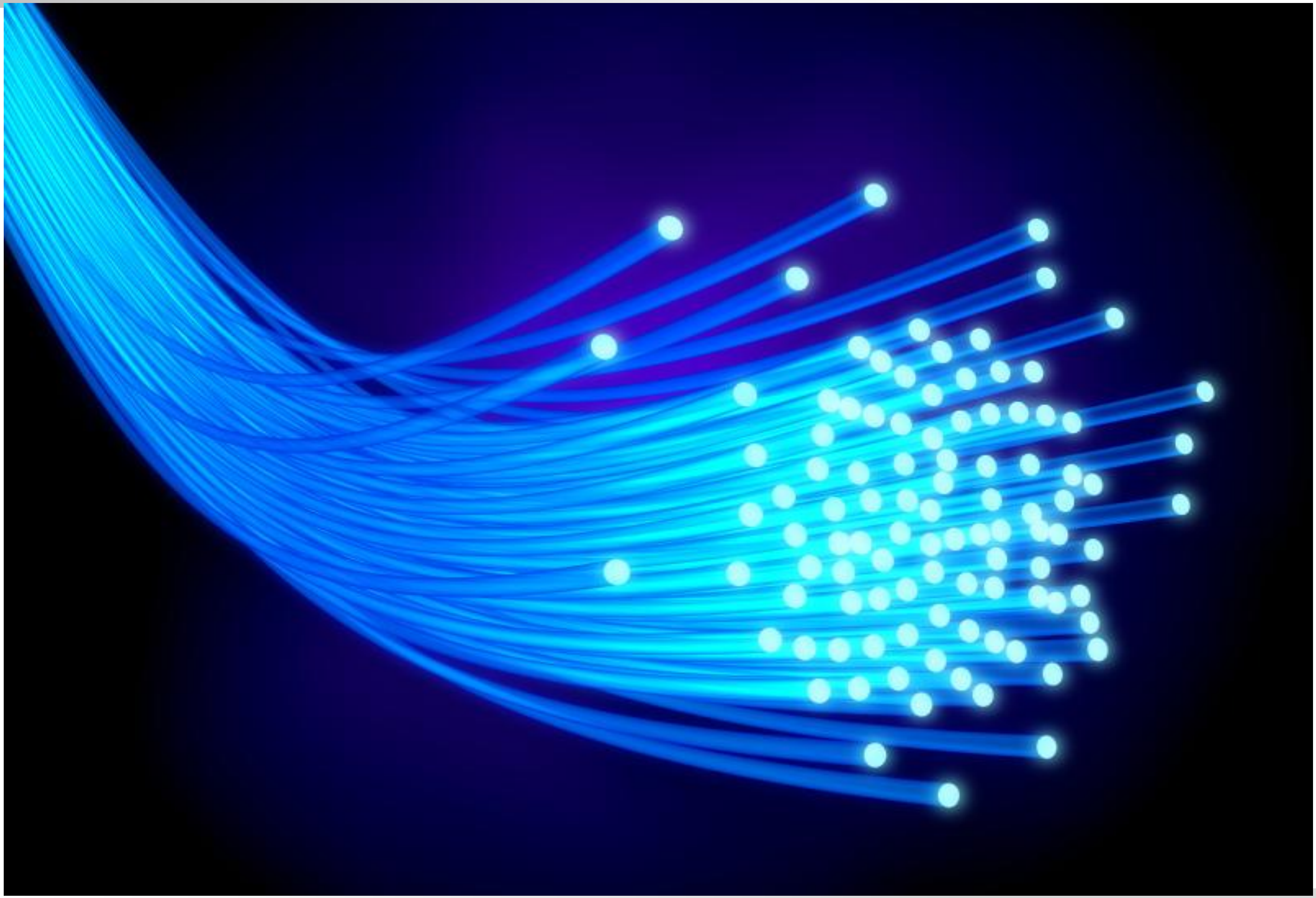


Which to use?



DAS is out of the picture

Fibre Channel



Manly men only use Fibre Channel

NFS - available everywhere

NFS is attractive but is it fast enough?

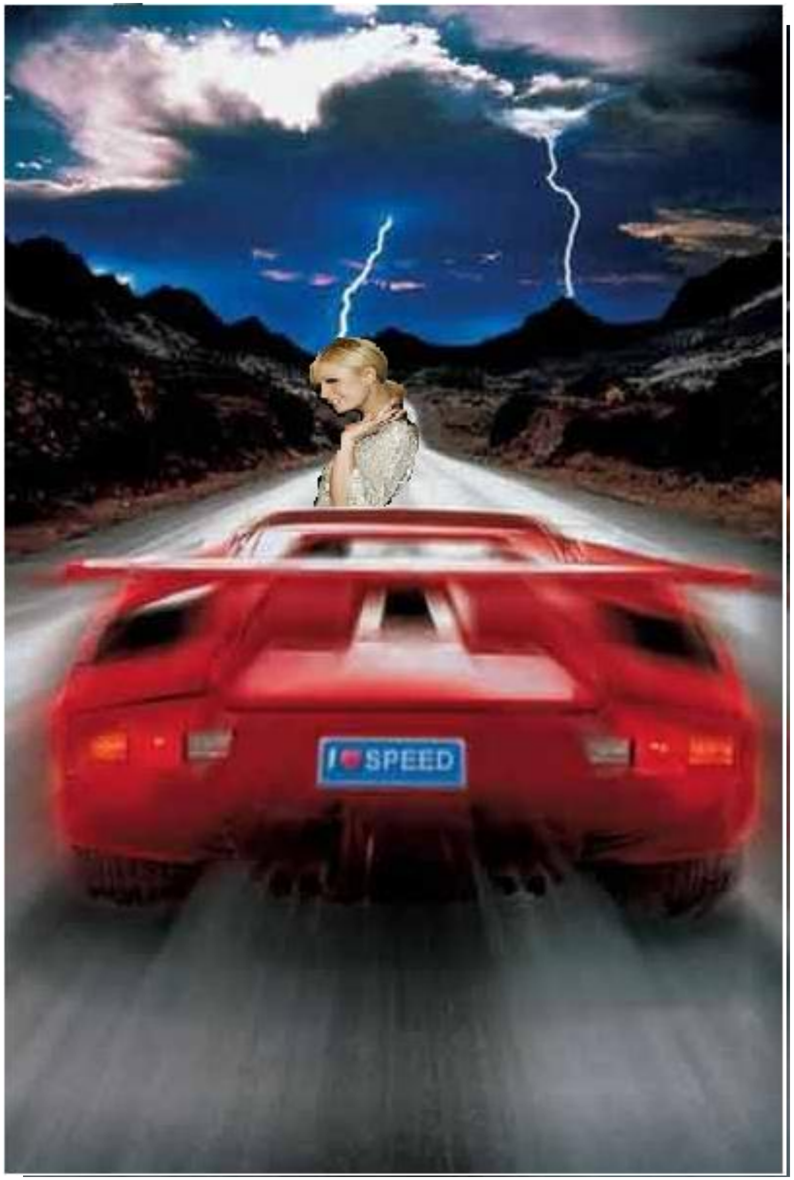
DAS vs NAS vs SAN

	attach	Agile	expensive	maintenance	speed
DAS	SCSI	no	no	difficult	fast
NAS	NFS - Ethernet	yes	no	easy	**??**
SAN	Fibre Channel	yes	yes	difficult	fast

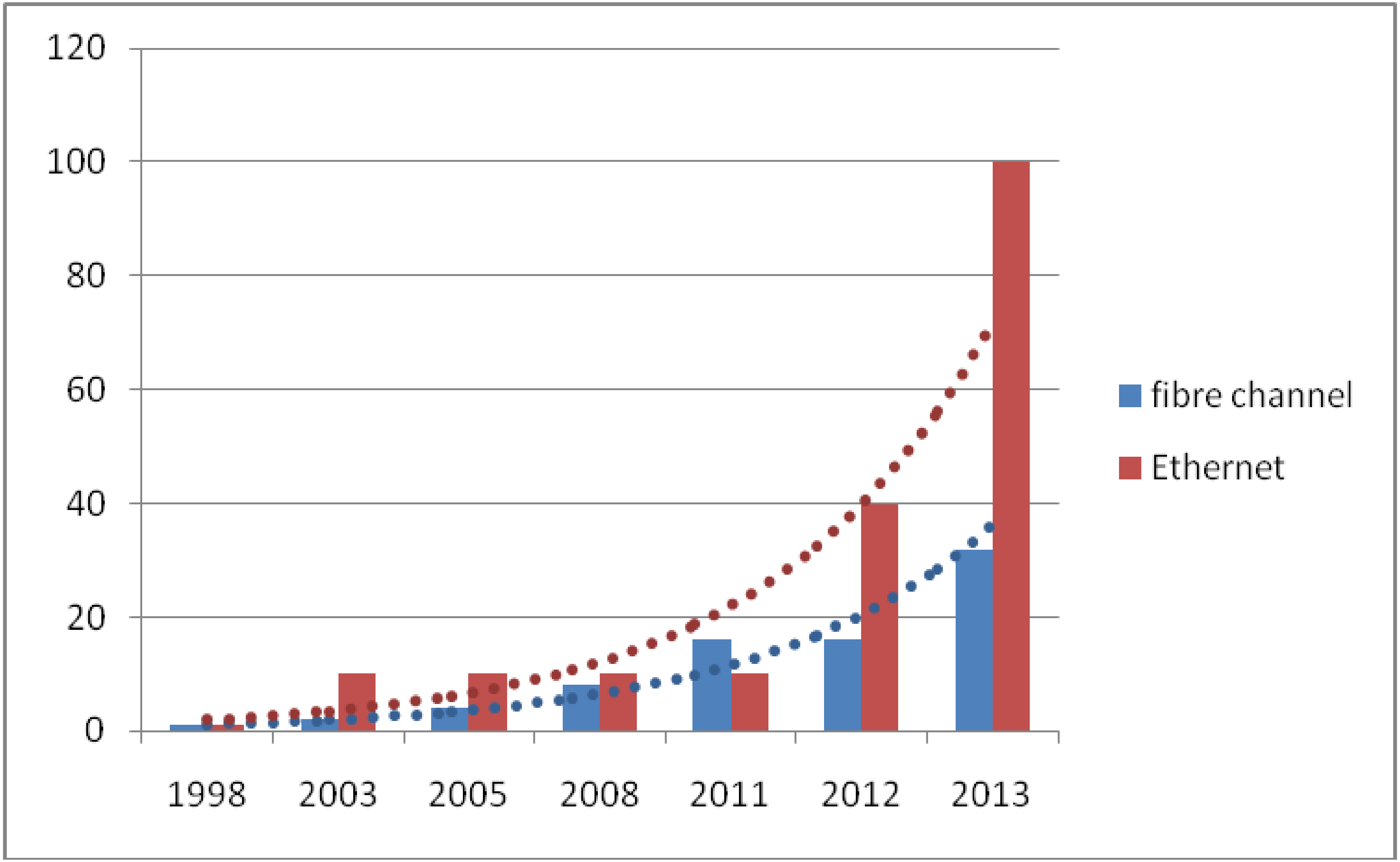
speed

Ethernet
- 100Mb 1994
- 1GbE - 1998
- 10GbE – 2003
- 40GbE – est. 2012
- 100GE –est. 2013

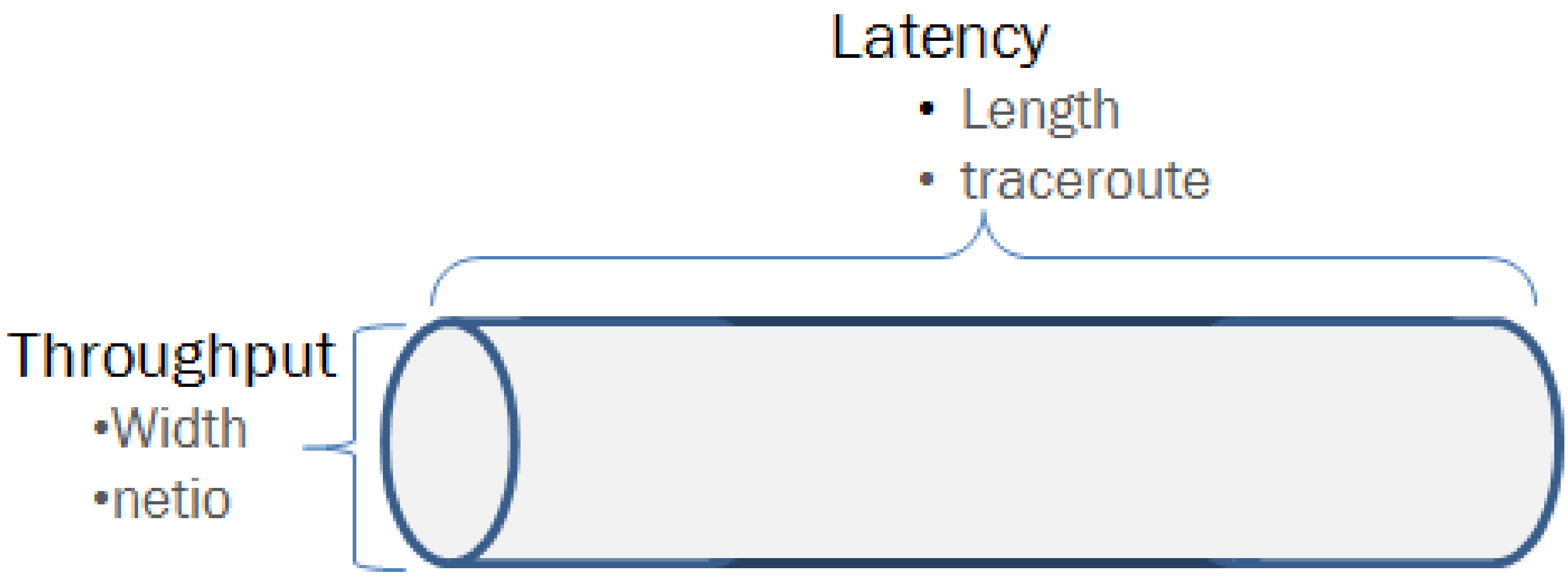
Fibre Channel
- 1G 1998
- 2G 2003
- 4G – 2005
- 8G – 2008
- 16G – 2011



Ethernet vs Fibre Channel



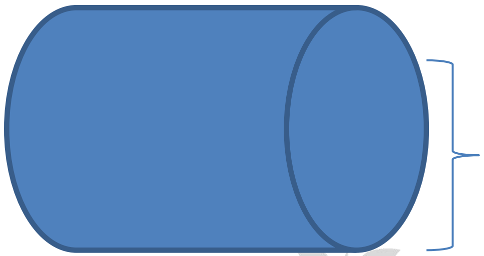
Throughput vs Latency



Throughput : netio

- 100MbE ~= 10MB/sec
- **1GbE ~= 100MB/sec** (125MB/sec max)
  - 30-60MB/sec typical, single threaded, mtu 1500
- 10GbE ~= 1GB/sec

b = bits, B = bytes (ie 8 bits)



Server machine

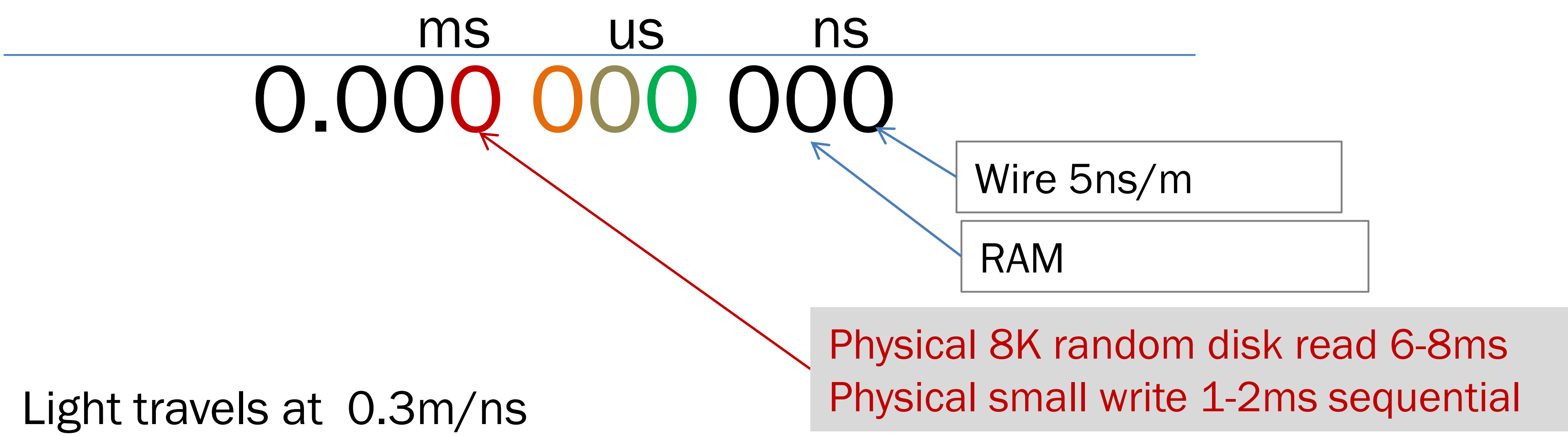
Test with

```
netio -s -b 32k -t -p 1234
```

Target

```
netio -b 32k -t -p 1234 delphix_machine
Receiving from client, packet size 32k ...  104.37 MByte/s
Sending to client, packet size 32k ...  109.27 MByte/s
Done.
```

Wire Speed – where is the hold up?



Light travels at 0.3m/ns
If wire speed is 0.2m/ns

Data Center 10m = 50ns

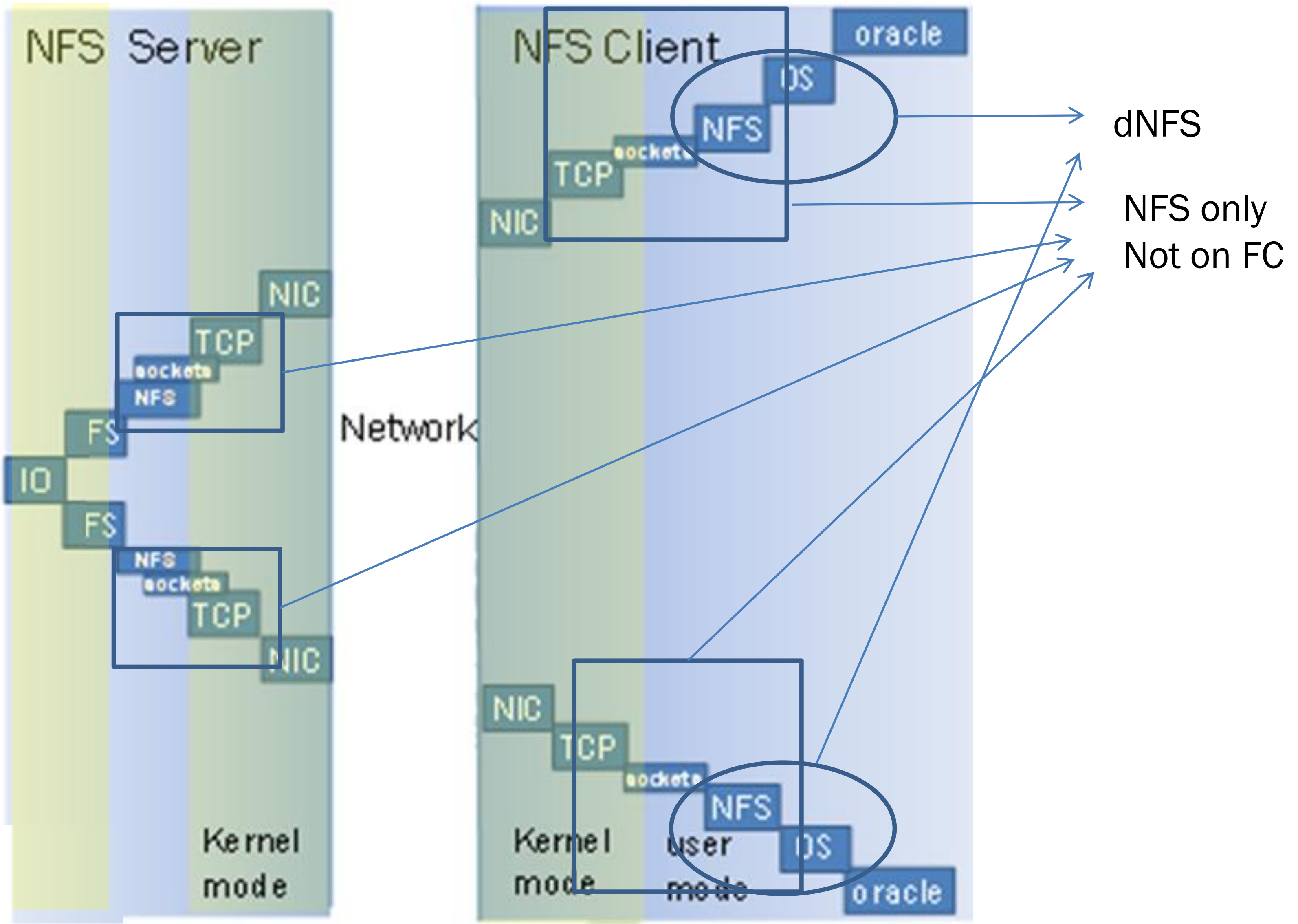
LA to London is 30ms
LA to SF is 3ms
(5us/km)

Why would FC be faster?

8K block transfer times

- 8GB FC = 10us
- 10G Ethernet= 8us

More stack more latency



Oracle and SUN benchmark

200us overhead of NFS over DAS

8K blocks 1GbE with Jumbo Frames , Solaris 9, Oracle 9.2

	UFS	NFS
I/O Response Time	7.19 ms	7.39 ms

Database Performance with NAS: Optimizing Oracle on NFS
Revised May 2009 | TR-3322
http://media.netapp.com/documents/tr-3322.pdf

80us is from wire transfer which goes down to 8us on 10GbE (like talking faster)

Latency has gotten even better.

8K block NFS latency overhead

- 1GbE -> 80us
- 10GbE -> 8us
- 200us on 1GbE = 128us on 10GbE
- If spindle I/O is 7ms
- Then NFS is over head is

(0.128ms/7ms) * 100 = 1.8% latency increase over DAS

Worth choosing FC?

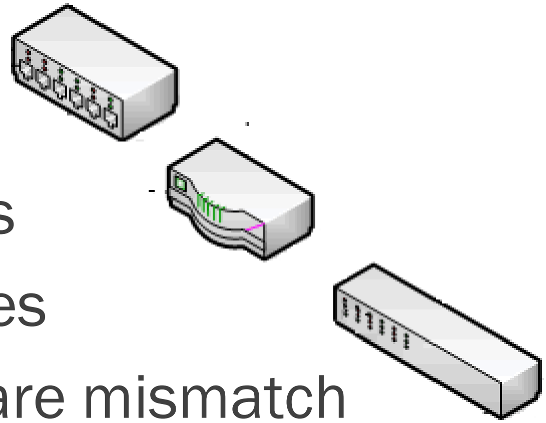
FC also has a latency overhead so the difference between FC and NFS is even smaller than 1.8%

NFS why the bad reputation?

- Given 1.8% overhead why the reputation?
- Historically slower
- Setup can make a big difference
  1. Network topology and load
  2. NFS mount options
  3. TCP configuration
- Compounding issues
  - Oracle configuration
  - I/O subsystem response

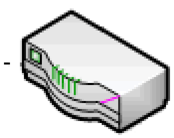
Network Topology

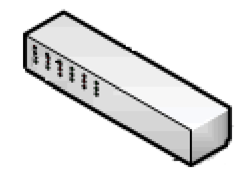
- Hubs
- Routers
- Switches
- Hardware mismatch
- Network Load

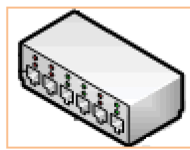


HUBs

Layer	Name		
7	Application		
6	Presentation		
5	Session		
4	Transport		
3	Network	Routers	IP addr
2	Datalink	Switches	mac addr
1	Physical	Hubs	Wire



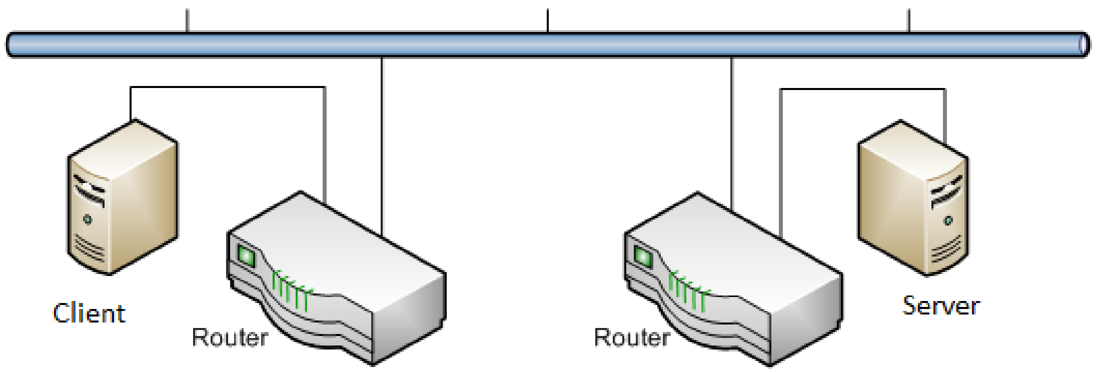




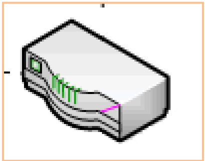
- Broadcast, repeaters
- Risk collisions
- Bandwidth contention

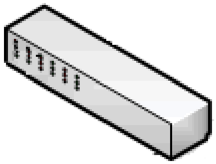
Routers

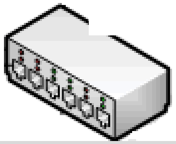
- Routers can add 300-500us latency
- If NFS latency is 350us (typical non-tuned) system
- Then each router multiplies latency 2x, 3x, 4x etc



Layer	Name		
3	Network	Routers	IP addr
2	Datalink	Switches	mac addr
1	Physical	Hubs	Wire







Routers: traceroute

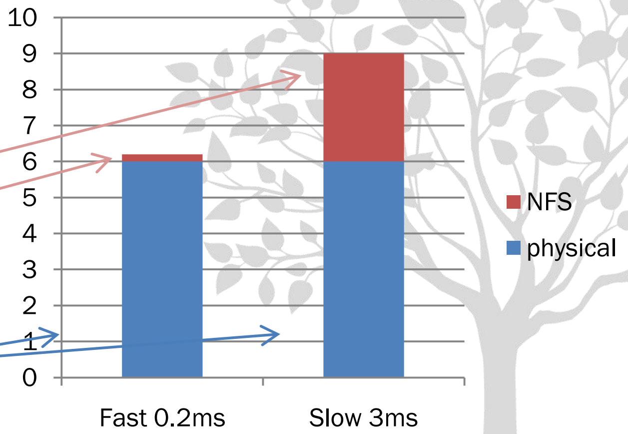
```
$ traceroute 101.88.123.195
 1   101.88.229.181 (101.88.229.181)   0.761 ms   0.579 ms   0.493 ms
 2   101.88.255.169 (101.88.255.169)   0.310 ms   0.286 ms   0.279 ms
 3   101.88.218.166 (101.88.218.166)   0.347 ms   0.300 ms   0.986 ms
 4   101.88.123.195 (101.88.123.195)   1.704 ms   1.972 ms   1.263 ms
sums (not shown )                        3.122      3.137        3.021
```

```
$ traceroute 172.16.100.144
 1   172.16.100.144 (172.16.100.144)   0.226 ms   0.171 ms   0.123 ms
```

3.0 ms NFS on slow network

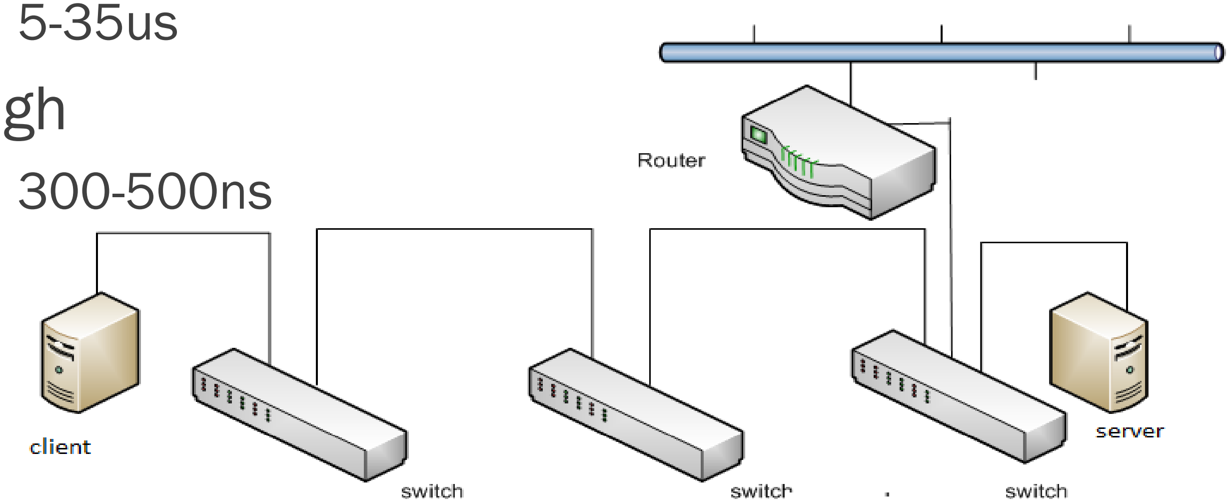
0.2 ms NFS good network

6.0 ms Typical physical read

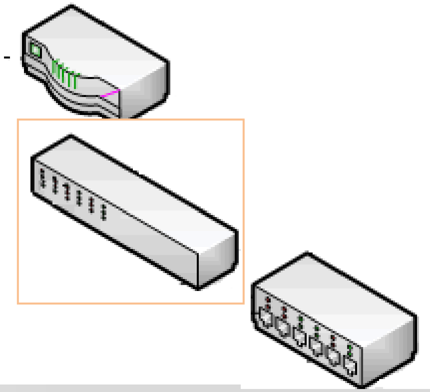


Multiple Switches

- Two types of Switches
  - Store and Forward
    - 1GbE 50-70us
    - 10GbE 5-35us
  - Cut through
    - 10GbE 300-500ns



Layer	Name		
3	Network	Routers	IP addr
2	Datalink	Switches	mac addr
1	Physical	Hubs	Wire



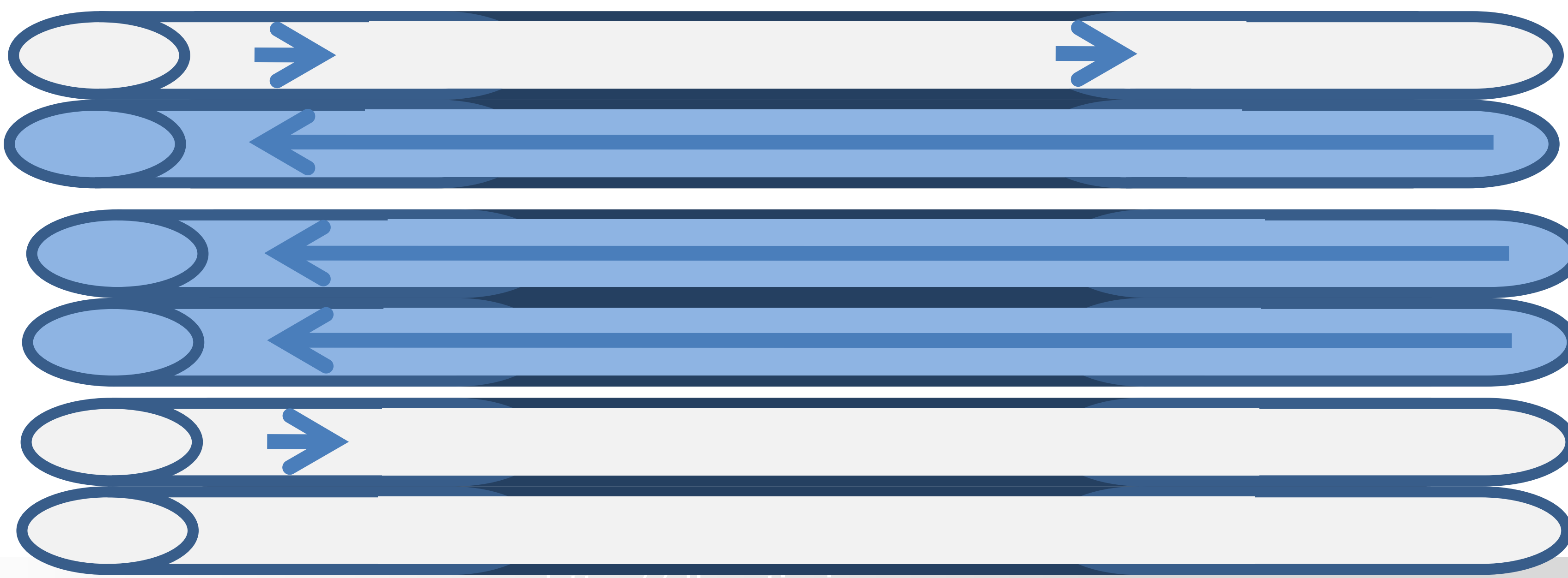
Hardware mismatch

- Speeds and duplex are often negotiated

Example Linux:

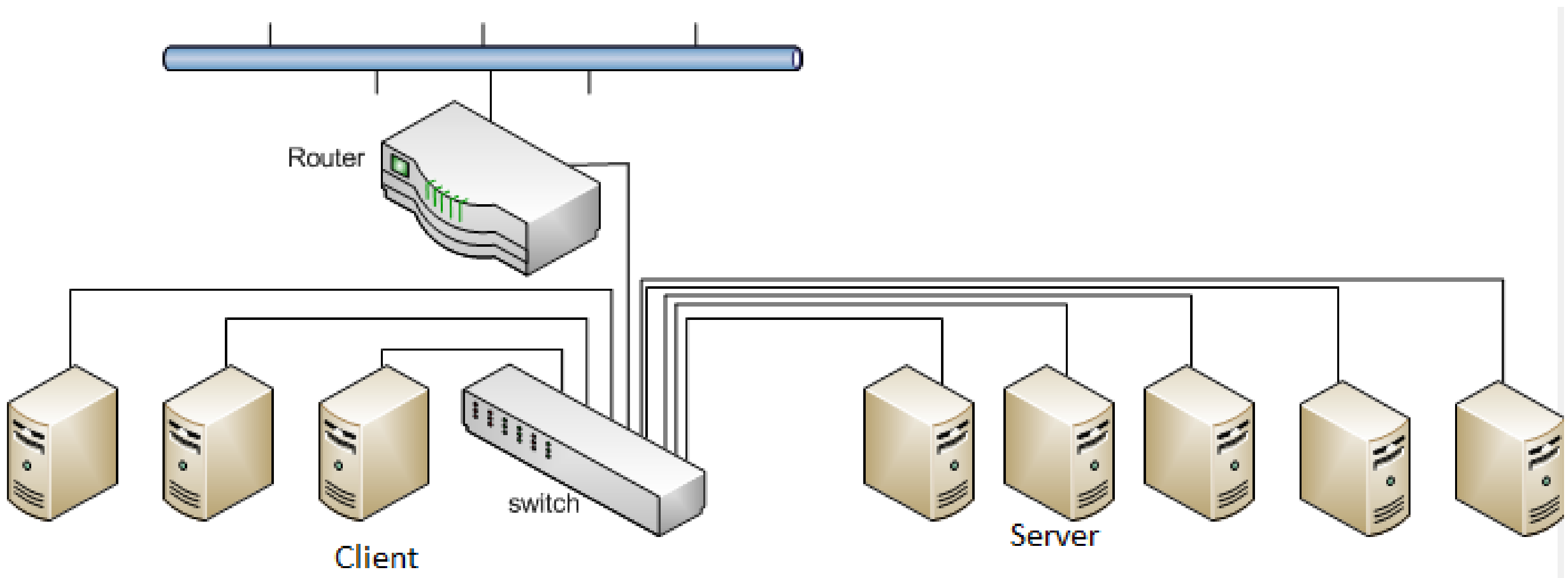
```
$ ethtool eth0
Settings for eth0:
        Advertised auto-negotiation: Yes
        Speed: 1000Mb/s
        Duplex: Full
```

- Check that values are as expected



Busy Network

- Traffic can congest network
  - Caused drop packets
  - Out of order packets
  - Collisions on hubs, probably not with switches



Busy Network Monitoring

- Visibility difficult from any one machine
  - Client
  - Server
  - Switch(es)

```
$ nfsstat -cr
Client rpc:
Connection oriented:
badcalls    badxids     timeouts    newcreds    badverfs    timers
89101       6           0           5           0           0           0
```

```
$ netstat -s -P tcp 1
TCP     tcpRtoAlgorithm     =     4     tcpRtoMin           =   400
        tcpRetransSegs      =  5986     tcpRetransBytes     =8268005
        tcpOutAck           =49277329   tcpOutAckDelayed    =473798
        tcpInDupAck         =357980     tcpInAckUnsent      =     0
        tcpInUnorderSegs    =10048089   tcpInUnorderBytes   =16611525
        tcpInDupSegs        = 62673     tcpInDupBytes       =87945913
        tcpInPartDupSegs    =    15     tcpInPartDupBytes   =   724
        tcpRttUpdate        =4857114    tcpTimRetrans       =  1191
        tcpTimRetransDrop   =     6     tcpTimKeepalive     =   248
```

Busy Network Testing

Netio is available here:http://www.ars.de/ars/ars.nsf/docs/netio

On Server box

```
netio -s -b 32k -t -p 1234
```

On Target box:

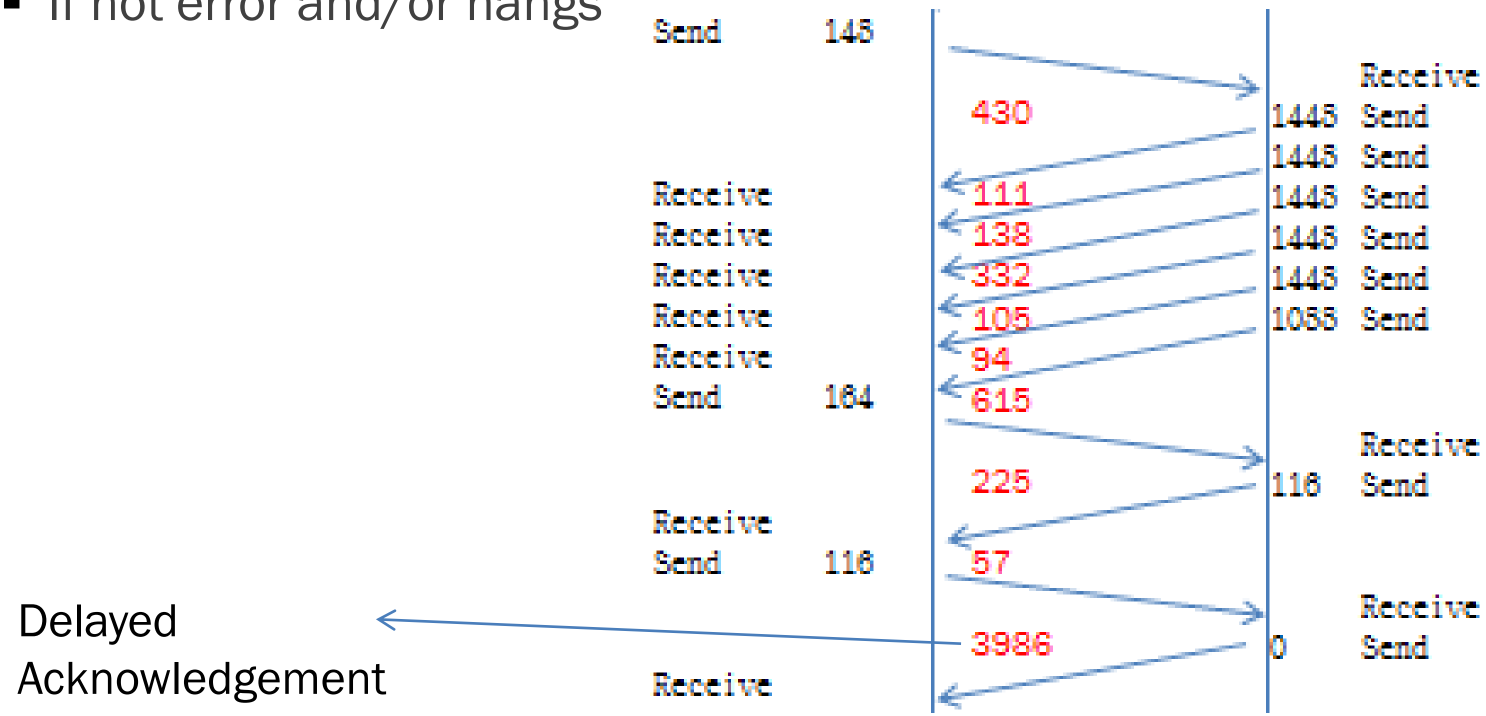
```
netio -b 32k -t -p 1234 delphix_machine
NETIO - Network Throughput Benchmark, Version 1.31
(C) 1997-2010 Kai Uwe Rommel
TCP server listening.
TCP connection established ...
Receiving from client, packet size 32k ...   104.37  MByte/s
Sending to client, packet size 32k ...   109.27  MByte/s
Done.
```

TCP Configuration

- MTU
- Socket buffer sizes
- TCP window size
- TCP congestion winow sizes

MTU 9000 : Jumbo Frames

- MTU – maximum Transfer Unit
  - Typically 1500
  - Can be set 9000
  - All components have to support
    - If not error and/or hangs



Jumbo Frames : MTU 9000

8K block transfer

Change MTU
ifconfig eth1 mtu 9000 up

Default MTU 1500

delta	send		recd
		<--	164
152	132	-->	
40	1448	-->	
67	1448	-->	
66	1448	-->	
53	1448	-->	
87	1448	-->	
95	952	-->	
= 560			

Now with MTU 900

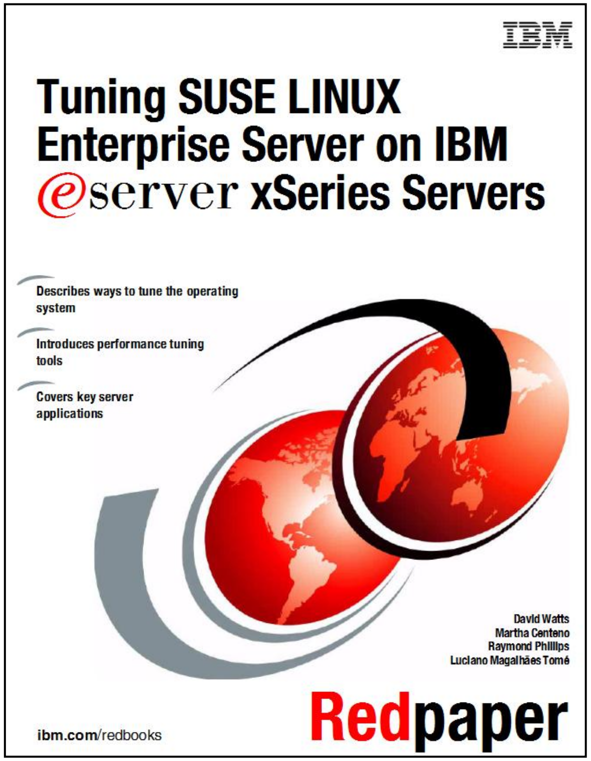
delta	send		recd
		<--	164
273	8324	-->	

Warning: MTU 9000 can hang if any of the hardware in the connection is configured only for MTU 1500

TCP Sockets

- Memory allocated to TCP send and receive buffers.
- If maximum is reached packets are dropped.

Excellent book



➔

- LINUX
  - Set the max OS send buffer size (wmem) and receive buffer size (rmem) to 8 MB for queues on all protocols:
    - sysctl -w net.core.wmem_max=8388608
    - sysctl -w net.core.rmem_max=8388608
  - These specify the amount of memory that is allocated for each TCP socket when it is
  - created. In addition, you should also use the following commands for send and receive buffers. They specify three values: minimum size, initial size, and maximum size:
    - sysctl -w net.ipv4.tcp_rmem="4096 87380 8388608"
    - sysctl -w net.ipv4.tcp_wmem="4096 87380 8388608"

TCP window sizes

- maximum amount of data to send or receive
- Subset of the TCP socket sizes

TCP window size

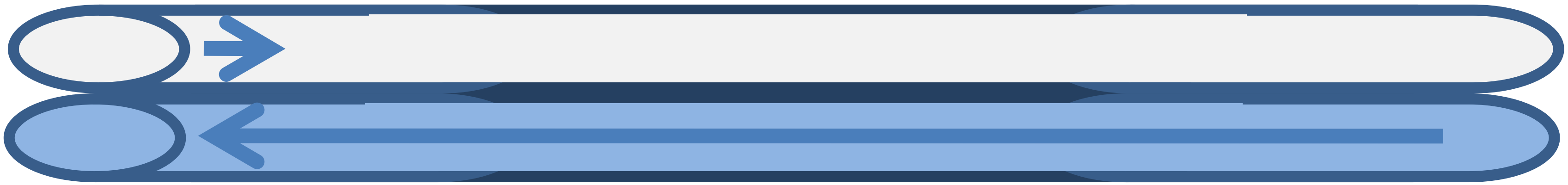
= latency * throughput

for example with 1ms latency over a 1Gb network

TCP window size = 1Gb/sec * 0.001s = 100Mb/sec * 1Byte/8bits= 125KB

Optimal TCP window size is generally cited as being twice this value

Optimal TCP window size =2 * latency * throughput = RTT * throughput



Congestion window

unack bytes sent	unack byte received	delta us	bytes sent	bytes received	send window	receive window	cong window
139760	0	31	1448 \		195200	131768	144800
139760	0	33	1448 \		195200	131768	144800
144104	0	29	1448 \		195200	131768	146248
145552	0	31		/ 0	195200	131768	144800
145552	0	41	1448 \		195200	131768	147696
147000	0	30		/ 0	195200	131768	144800
147000	0	22	1448 \		195200	131768	76744
147000	0	28		/ 0	195200	131768	76744
147000	0	18	1448 \		195200	131768	76744

Unacknowledged bytes
Hits the congestion window size

congestion window size is
drastically lowered

NFS mount options

- Forcedirectio
- Rsize / wsize
- Actimeo=0, noac

Sun Solaris	rw,bg,hard,**rsize=32768,wsize=32768**,vers=3,[**forcedirectio** or llock],nointr,proto=tcp,suid
AIX	rw,bg,hard,**rsize=32768,wsize=32768**,vers=3,**cio**,intr,timeo=600,proto=tcp
HPUX	rw,bg,hard,**rsize=32768,wsize=32768**,vers=3,nointr,timeo=600,proto=tcp, suid, **forcedirectio**
Linux	rw,bg,hard,**rsize=32768,wsize=32768**,vers=3,nointr,timeo=600,tcp,**actimeo=0**

Forcedirectio

- Causes UNIX file cache to be bypassed
- Data is read directly into UNIX
- Controlled by init.ora parameter
  - Filesystemio_options=SETALL or directio
  - Except HPUX where mount option is the only way
  - Solaris doesn’t require the mount option

Sun Solaris	**Forcedirectio – sets directio but not required** **Fielsystemio_options will set directio without mount option**
AIX	
HPUX	**Forcedirectio – only way to set directio** **Filesystemio_options has no affect**
Linux	

Direct I/O

query doing

77951 physical reads for the second execution

(ie when data should already be cached)

- 60 secs => direct I/O
- 5 secs => no direct I/O
- 2 secs => SGA

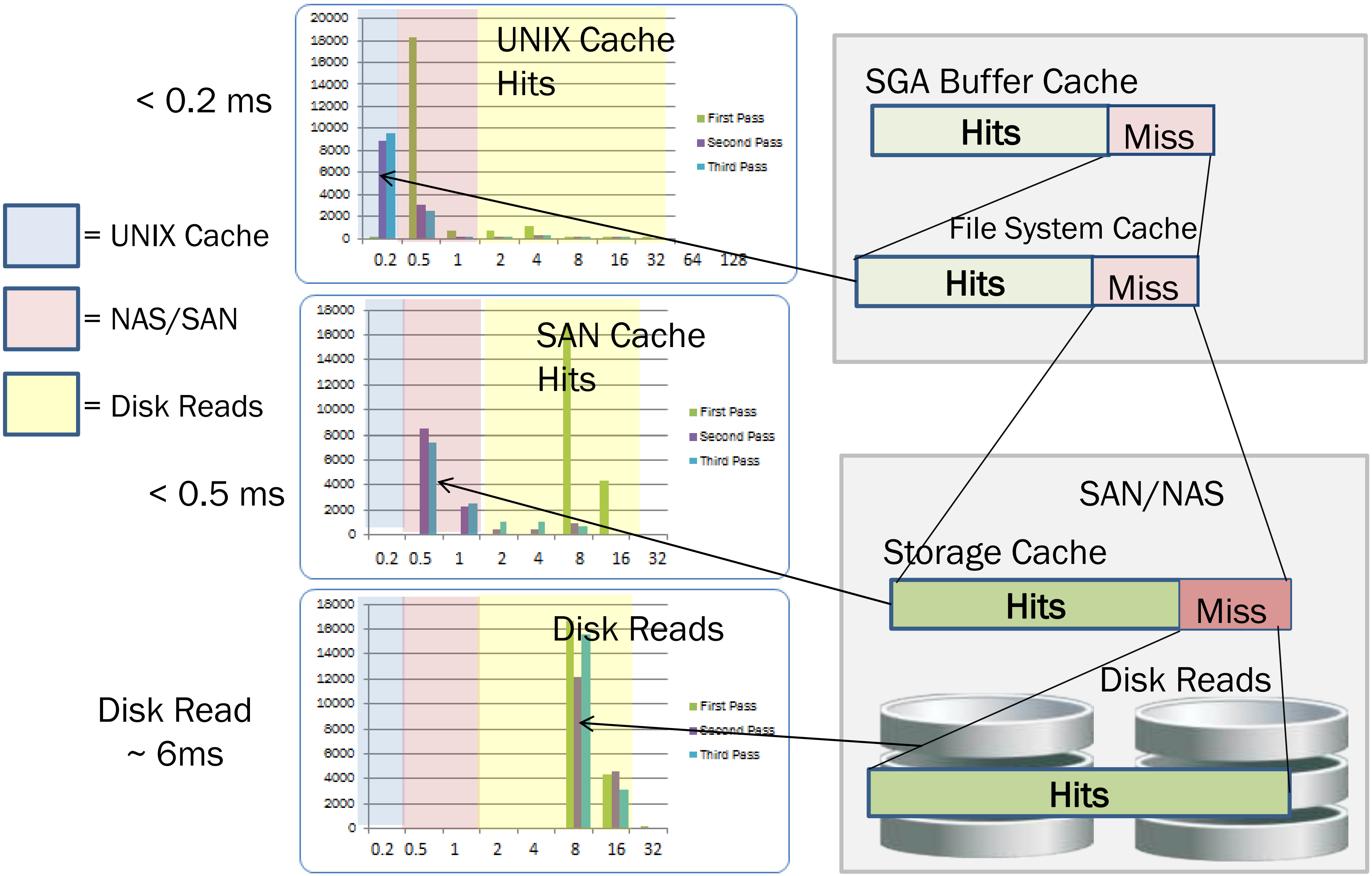
- Why use direct I/O?

Direct I/O

- Advantages
  - Faster reads from disk
  - Reduce CPU
  - Reduce memory contention
  - Faster access to data already in memory, in SGA
- Disadvantages
  - Less Flexible
  - More work
  - Risk of paging , memory pressure
  - Impossible to share memory between multiple databases

Cache	**OS**	**Rows/sec**	**Usr**	**sys**
FS	S9	287,114	71	28
DB	S9	**695,700**	94	5

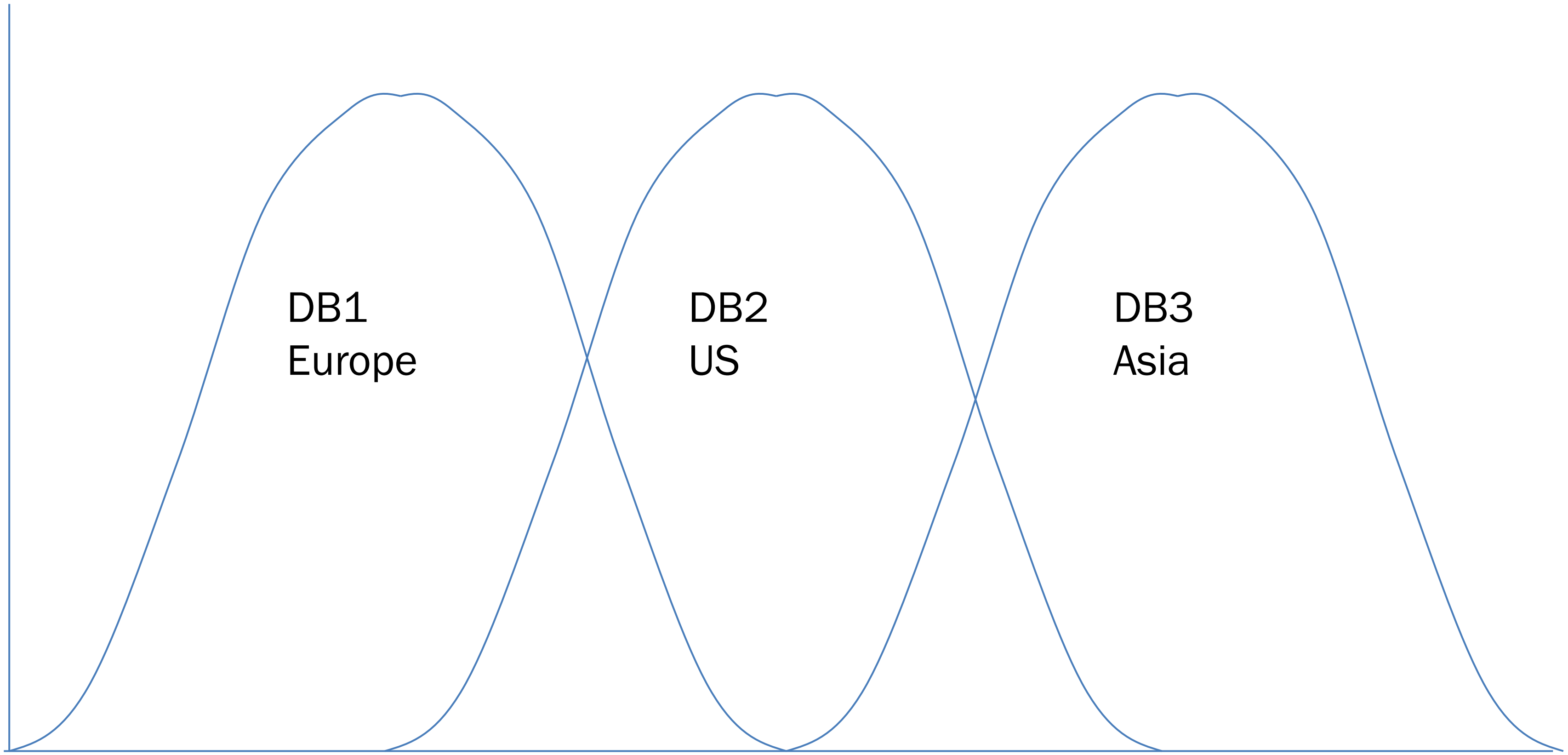
http://blogs.oracle.com/glennf/entry/where_do_you_cache_oracle



< 0.2 ms
= UNIX Cache
= NAS/SAN
= Disk Reads
< 0.5 ms
Disk Read
~ 6ms
UNIX Cache
Hits
First Pass
Second Pass
Third Pass
SAN Cache
Hits
Disk Reads
SGA Buffer Cache
Hits
Miss
File System Cache
Hits
Miss
SAN/NAS
Storage Cache
Hits
Miss
Disk Reads
Hits

Direct I/O Challenges

Database Cache usage over 24 hours



ACTIMEO=0 , NOAC

- Disable client side file attribute cache
- Increases NFS calls
- Significantly increases latency and reduces throughput
- Not required on single instance Oracle
- Metalink says it’s required on LINUX
- Another metalink it should be taken off

=> It should be take off

rsize/wsize

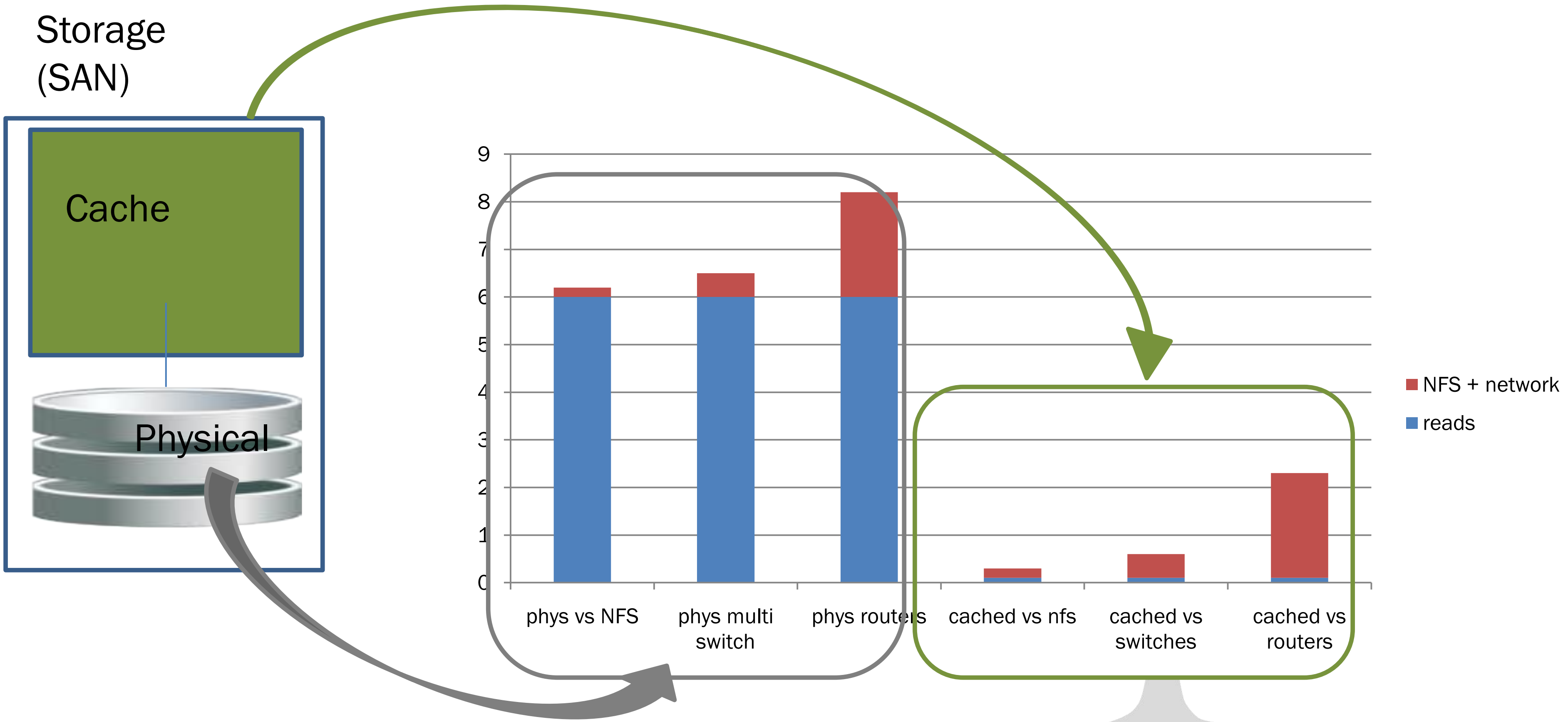
- NFS transfer buffer size
- Oracle says use 32K
- Platforms support higher values and can significantly impact throughput

Sun Solaris	**rsize=32768,wsize=32768 , max is 1M**
AIX	**rsize=32768,wsize=32768 , max is 64K**
HPUX	r**size=32768,wsize=32768 , max is 1M**
Linux	r**size=32768,wsize=32768 , max is 1M**

On full table scans using 1M has halved the response time over 32K
Db_file_multiblock_read_count has to large enough take advantage of the size

NFS Overhead Physical vs Cached IO

100us extra over 6ms spindle read is small
100us extra over 100us cache read is 2x as slow
SAN cache is expensive – use it for write cache
Target cache is cheaper – put more on if need be



Conclusions

- NFS performance can come close to FC
- Requires
  - Network topology be clean – no routers, fast switches
  - Mount options correct (and/or dNFS , version 11g)
    - Rsize/wsize at maximum
    - Avoid actimeo=0 and noac
  - TCP configuration – MTU 9000 (tricky)
- Drawbacks
  - NFS failover can take 10s of seconds
  - With Oracle 11g dNFS can be handled transparently

Conclusion: Give NFS some more love

NFS

gigabit switch can be anywhere from
10 to 50 times cheaper than an FC switch

Annan <3

dtrace

List the names of traceable probes:

dtrace –ln provider:module:function:name

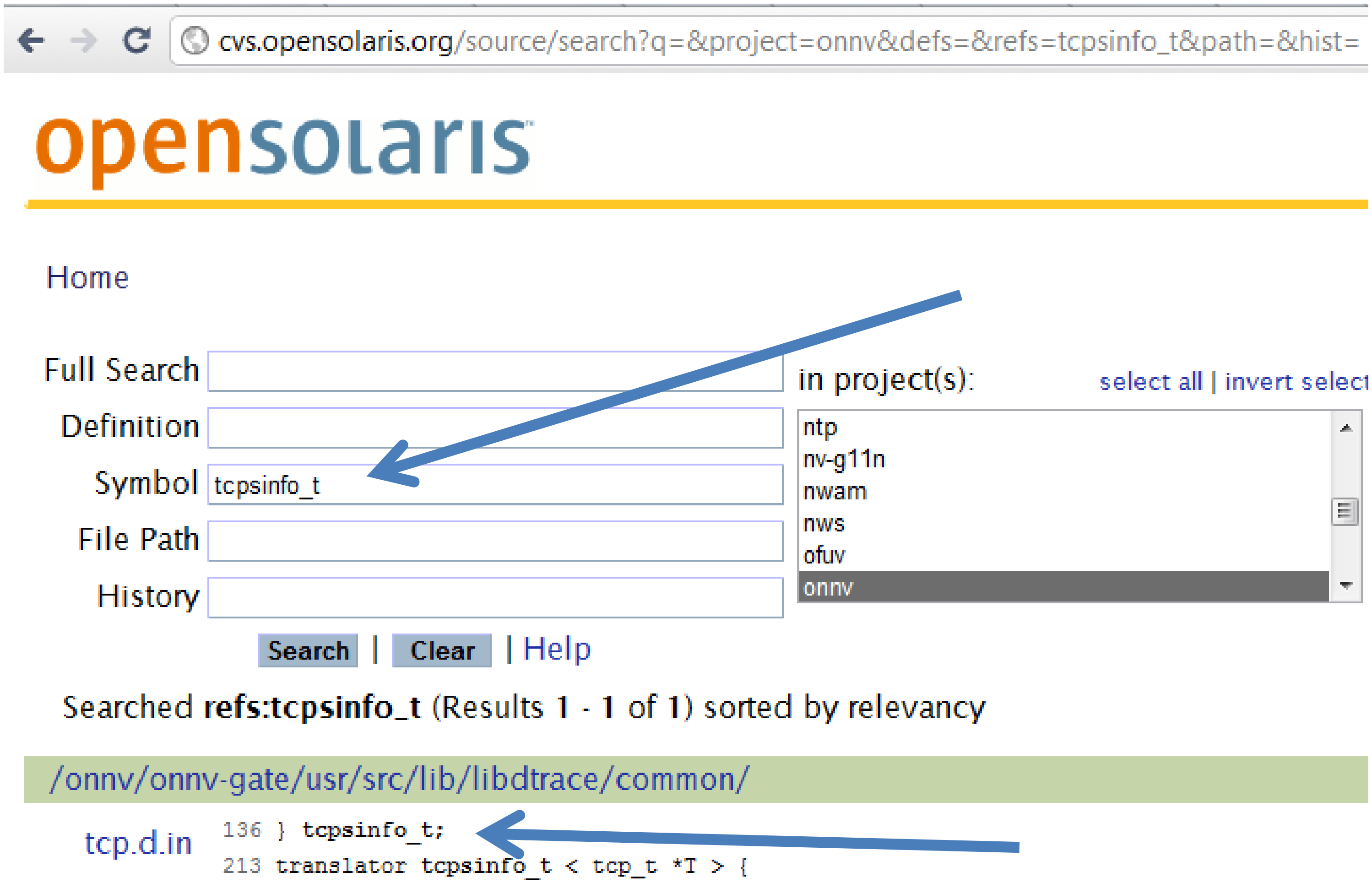
- -l = list instead of enable probes
- -n = Specify probe name to trace or list
- -v = Set verbose mode

Example
dtrace –ln tcp:::send

```
$ dtrace -lvn tcp:::receive
 5473  tcp  ip  tcp_output send

        Argument Types
                args[0]: pktinfo_t *
                args[1]: csinfo_t *
                args[2]: ipinfo_t *
                args[3]: tcpsinfo_t *
                args[4]: tcpinfo_t *
```

http://cvs.opensolaris.org/source/



cvs.opensolaris.org/source/xref/onnv/onnv-gate/usr/src/lib/libdtrace/common/tcp.d.in#136

opensolaris

xref: /onnv/onnv-gate/usr/src/lib/libdtrace/common/tcp.d.in

Home | History | Annotate | Line # | Download | Search only in common

```
typedef struct tcpsinfo {
        uintptr_t tcps_addr;
        int tcps_local;                 /* is delivered locally, boolean */
        int tcps_active;                /* active open (from here), boolean */
        uint16_t tcps_lport;            /* local port */
        uint16_t tcps_rport;            /* remote port */
        string tcps_laddr;              /* local address, as a string */
        string tcps_raddr;              /* remote address, as a string */
        int32_t tcps_state;             /* TCP state */
        uint32_t tcps_iss;              /* Initial sequence # sent */
        uint32_t tcps_suna;             /* sequence # sent but unacked */
        uint32_t tcps_snxt;             /* next sequence # to send */
        uint32_t tcps_rack;             /* sequence # we have acked */
        uint32_t tcps_rnxt;             /* next sequence # expected */
        uint32_t tcps_swnd;             /* send window size */
        int32_t tcps_snd_ws;            /* send window scaling */
        uint32_t tcps_rwnd;             /* receive window size */
        int32_t tcps_rcv_ws;            /* receive window scaling */
        uint32_t tcps_cwnd;             /* congestion window */
        uint32_t tcps_cwnd_ssthresh;    /* threshold for congestion avoidance */
        uint32_t tcps_sack_fack;        /* SACK sequence # we have acked */
```

Dtrace

```
tcp:::send, tcp:::receive
{   delta= timestamp-walltime;
    walltime=timestamp;
    printf("%6d %6d %6d %8d \ %8s  %8d %8d %8d %8d  %d  \n",
        args[3]->tcps_snxt - args[3]->tcps_suna ,
        args[3]->tcps_rnxt - args[3]->tcps_rack,
        delta/1000,
        args[2]->ip_plength - args[4]->tcp_offset,
        "",
        args[3]->tcps_swnd,
        args[3]->tcps_rwnd,
        args[3]->tcps_cwnd,
        args[3]->tcps_retransmit
      );
}
tcp:::receive
{     delta=timestamp-walltime;
      walltime=timestamp;
      printf("%6d %6d %6d %8s / %-8d  %8d %8d %8d %8d  %d  \n",
        args[3]->tcps_snxt - args[3]->tcps_suna ,
        args[3]->tcps_rnxt - args[3]->tcps_rack,
        delta/1000,
        "",
        args[2]->ip_plength - args[4]->tcp_offset,
        args[3]->tcps_swnd,
        args[3]->tcps_rwnd,
        args[3]->tcps_cwnd,
        args[3]->tcps_retransmit
      );
}
```

Dtrace

```
#!/usr/sbin/dtrace -s
#pragma D option quiet
#pragma D option defaultargs
inline string ADDR=$$1;
tcp:::send, tcp:::receive
/     ( args[2]->ip_daddr == ADDR || ADDR == NULL ) /
{
    nfs[args[1]->cs_cid]=1; /* this is an NFS thread */
    delta= timestamp-walltime;
    walltime=timestamp;
    printf("%6d %6d %6d %8d \ %8s  %8d %8d %8d %8d  %d  \n",
        args[3]->tcps_snxt - args[3]->tcps_suna ,
        args[3]->tcps_rnxt - args[3]->tcps_rack,
        delta/1000,
        args[2]->ip_plength - args[4]->tcp_offset,
        "",
        args[3]->tcps_swnd,
        args[3]->tcps_rwnd,
        args[3]->tcps_cwnd,
        args[3]->tcps_retransmit
      );
}
tcp:::receive
/ ( args[2]->ip_saddr == ADDR || ADDR == NULL ) && nfs[args[1]->cs_cid] /
{
      delta=timestamp-walltime;
      walltime=timestamp;
      printf("%6d %6d %6d %8s / %-8d  %8d %8d %8d %8d  %d  \n",
        args[3]->tcps_snxt - args[3]->tcps_suna ,
        args[3]->tcps_rnxt - args[3]->tcps_rack,
        delta/1000,
        "",
        args[2]->ip_plength - args[4]->tcp_offset,
```