NAS, SANs and Parallel File Systems

Nick Maclaren

nmm1@cam.ac.uk

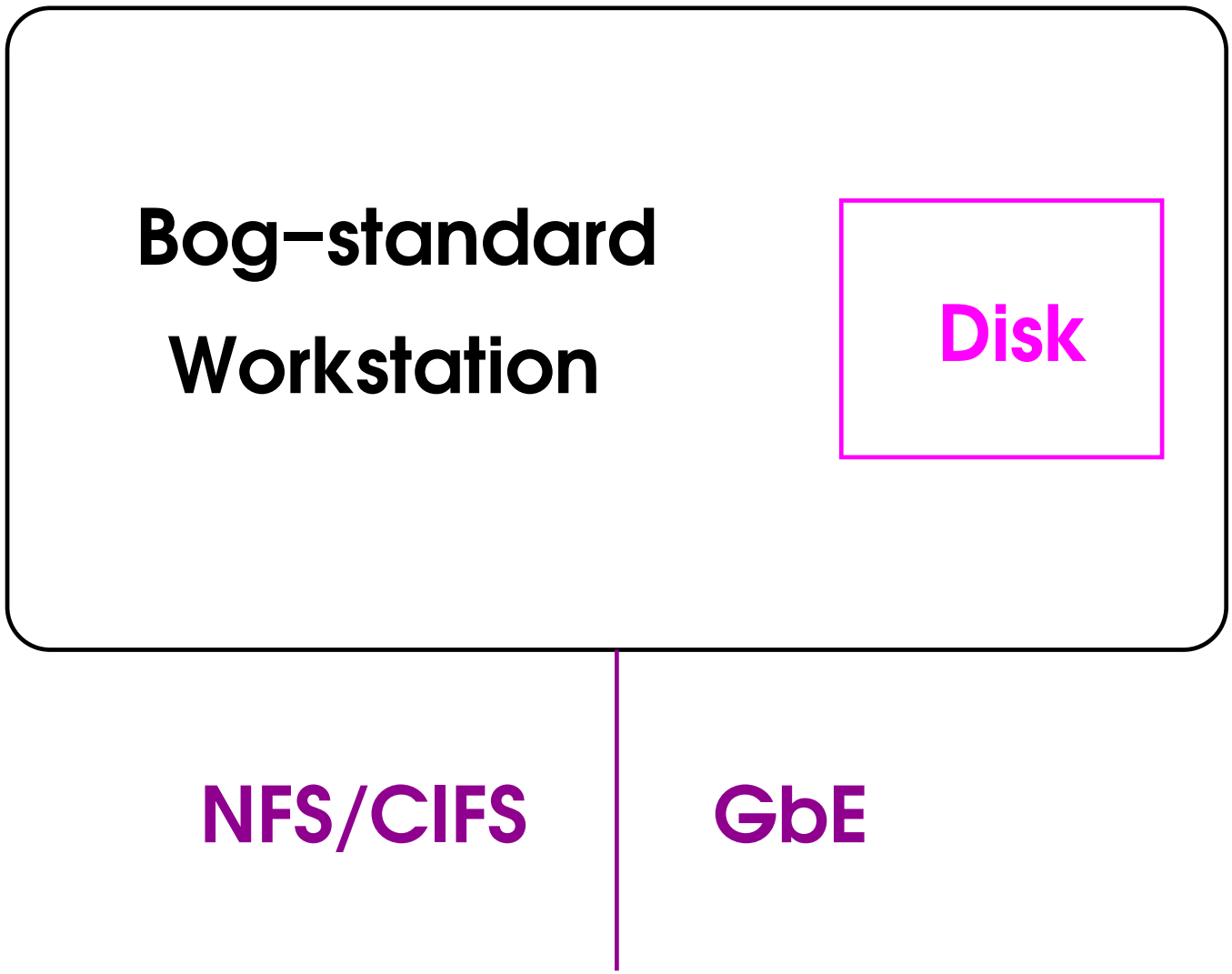
February 2008

Summary of Presentation

- Background and default technology
- What the terms mean (insofar as they do)
- How they are constructed and operate
- What they can do and what they will not do
- An overview of the more common software

This is an anti–salesman innoculation :–)

Absolutely Basic File Server



And Beyond That?

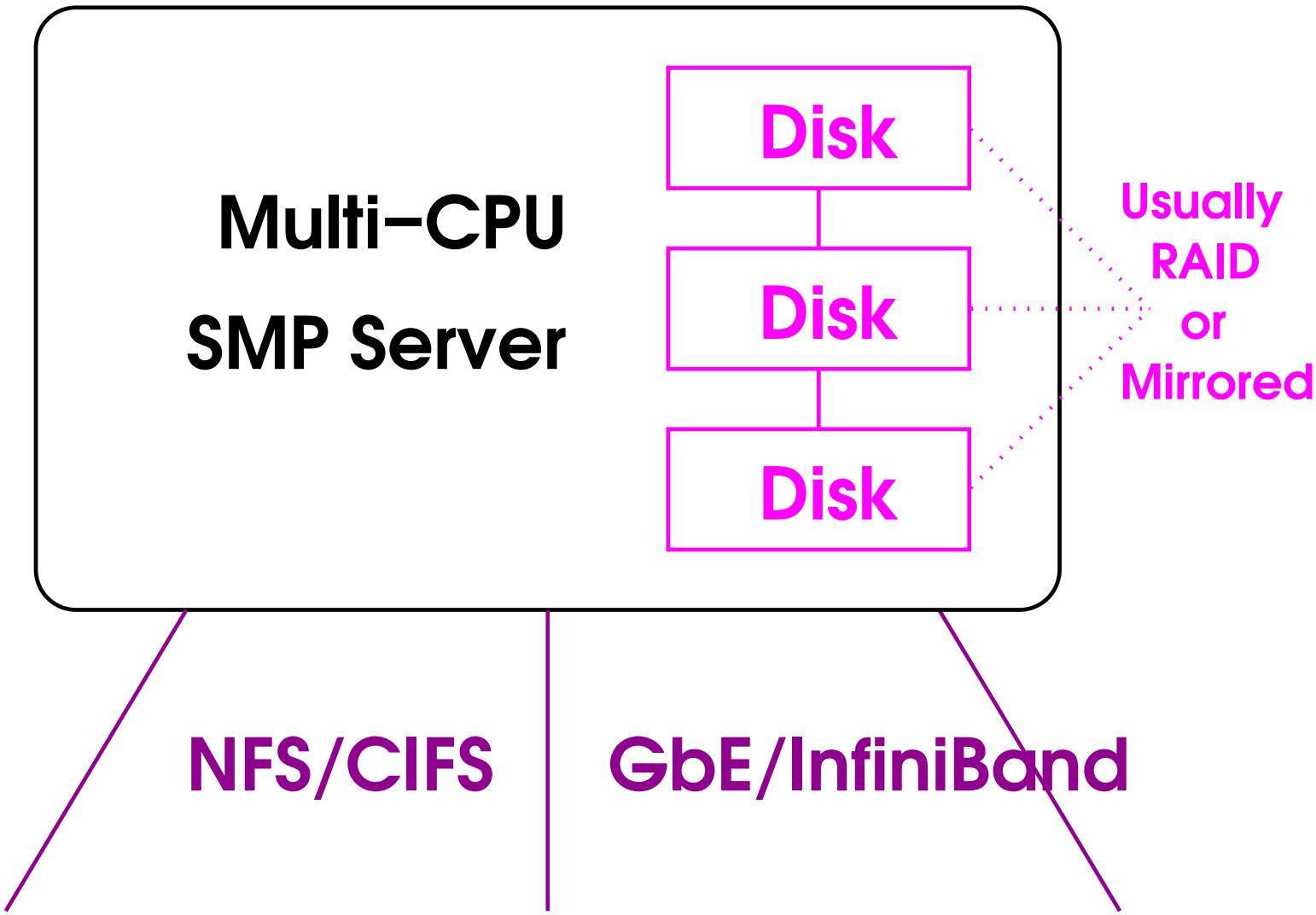
More than one disk (perhaps up to 6–10)
Limit is terabytes in 2008 (2–10 per box)

Mirroring (halves the space, but provides safety)
Software RAID–5 (needs 3+ disks etc.)

Multiple Ethernet ports (watch out here)
Or even an InfiniBand network!

More memory (ECC of course), SMP CPUs
UPS, remote power management etc.

Much Fancier File Server



Tens/Hundreds of Terabytes

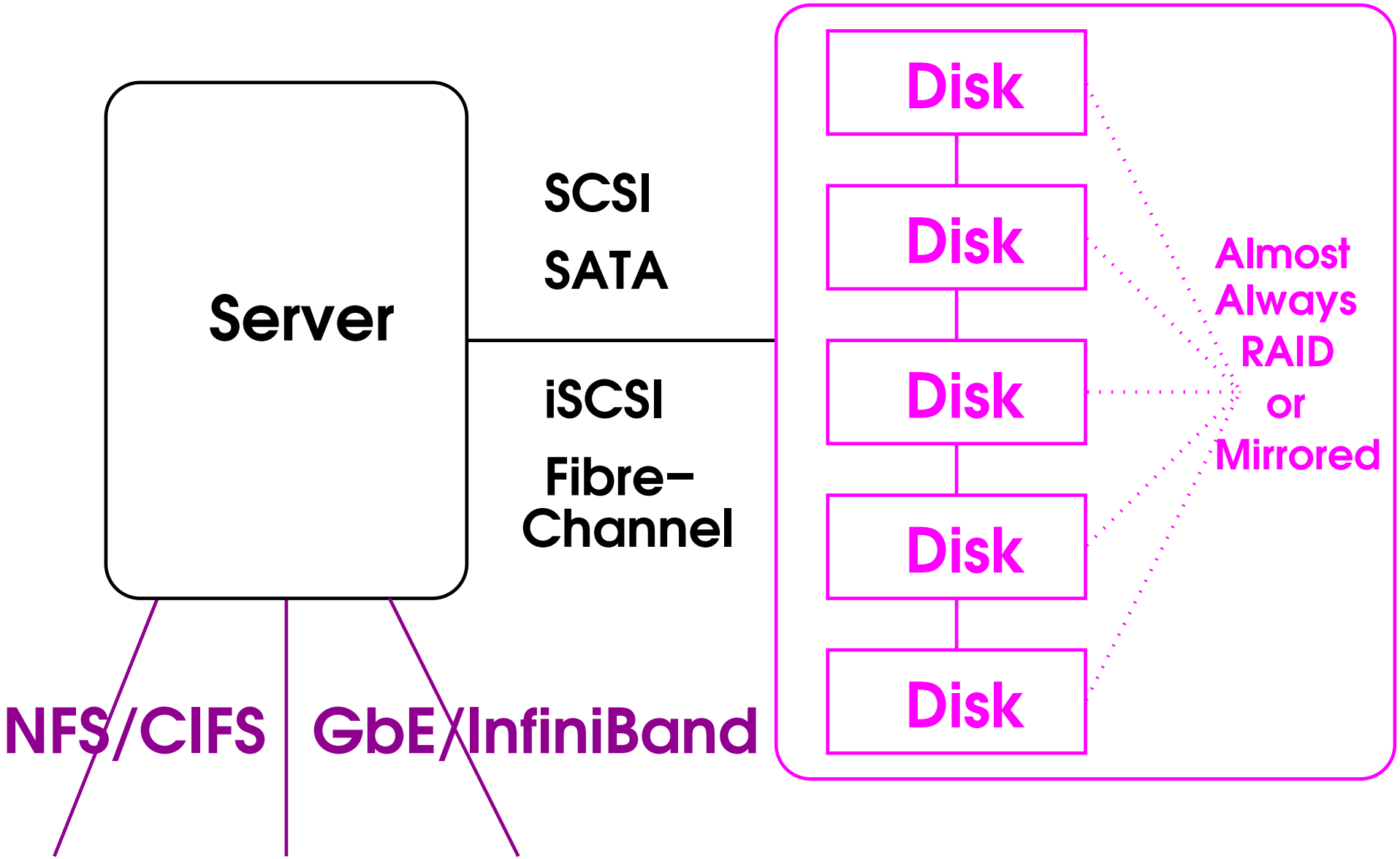
You need some sort of multi–disk shelf/box
All major server vendors sell them

Can choose to use hardware RAID–5 or not
Recovering tens of TB is incredibly tedious

Devices may be ‘local’ (SCSI, SATA etc.)
Or ‘networked’ (Ethernet, InfiniBand etc.)

Generally, manage them much like disks
Attached as devices or point–to–point network

Using External Disks



Aside: a Word to the Wise

RAID–5 and UPS help only sometimes, not with:

- File system corruption caused by software
- Finger trouble by administrators and users
- Fire, flood, theft, malicious damage etc.

No substitute for backups to somewhere else
Even if just to a separate set of disks

You regularly check recovery, of course?

NAS – Network Attached Storage

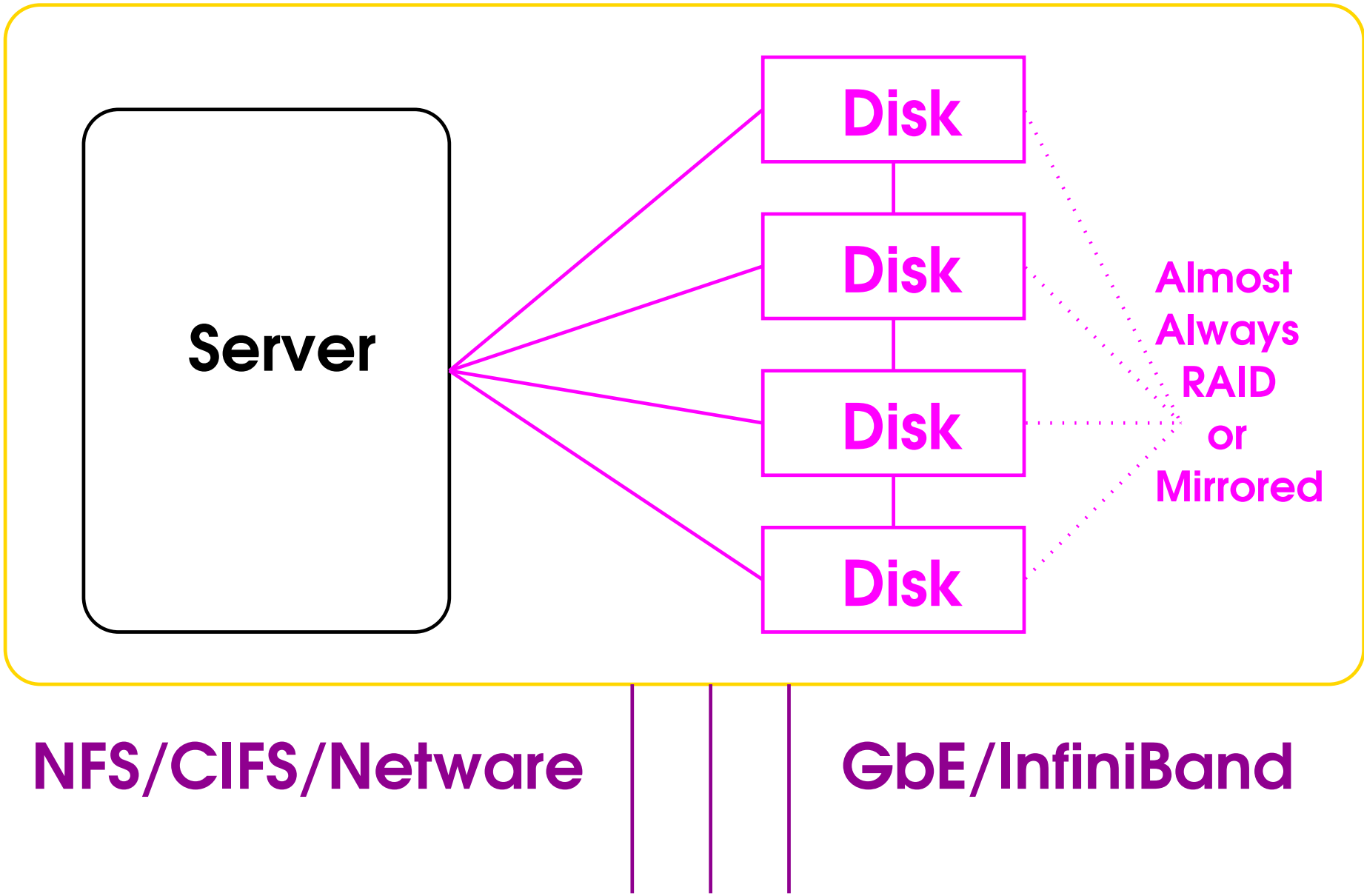
It's just a complete file server in a box

Can be easier to administer but less flexible
You usually pay extra for the pre–configuration
Remote management facilities of some sort

Occasionally the term is used differently
External disks connected over a network

Those are described under Simple SAN

Network Attached Storage



SAN – Storage Area Network

What on earth does that gibberish mean?

$\Rightarrow$ As far as I can tell, almost nothing

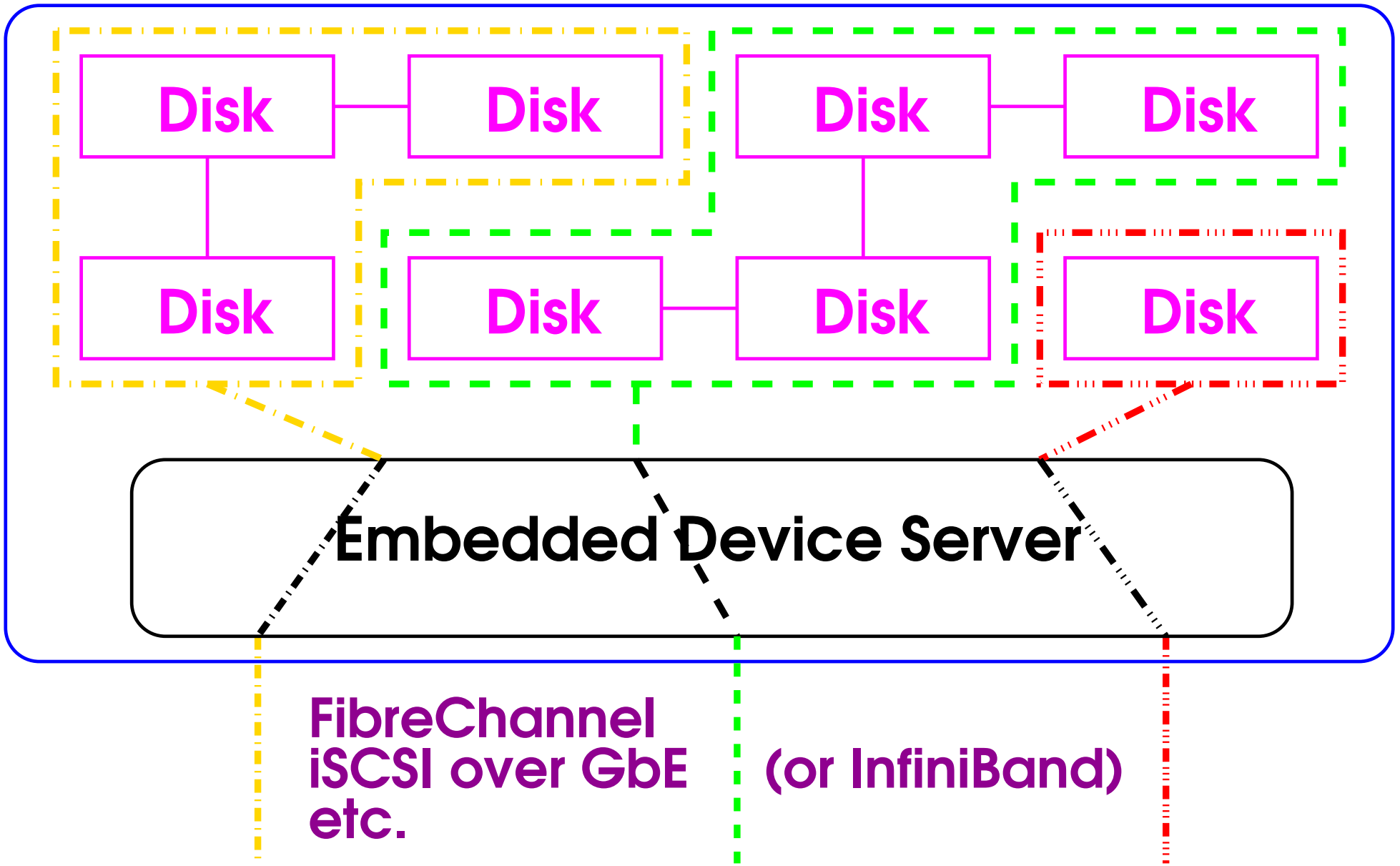
- Simplest use is consolidated external disks

Connected to servers by a network (LAN)
Can usually boot from and swap/dump to them

- Sometimes used for parallel file server/system

Will come back to this later

Simple SAN


Disk
Disk
Disk
Disk
Disk
Disk
Disk
Disk
Embedded Device Server
FibreChannel
iSCSI over GbE
etc.
(or InfiniBand)

Simple SANs (1)

Purpose is ease of administration
Centralise disk management in one place
A single rack contains many servers' disks

Can save money in UPS, space in racks
Can sometimes save effort in taking backups

Commercial sites seem to find them useful
Few academic ones do, as far as I can tell
The PWF does it with NetWare, though

Staff effort is cheap or free in academia :–(

Simple SANs (2)

Interface is generally as virtual disks (LUNs)
Connected to server as devices

Via a device–level interface (e.g. FibreChannel)
iSCSI is SCSI over TCP/IP
Can use Ethernet or InfiniBand

Disk space divided into partitions
A partition mounted by just one server
Avoids interlocking/consistency problems

Sharing Filesystems

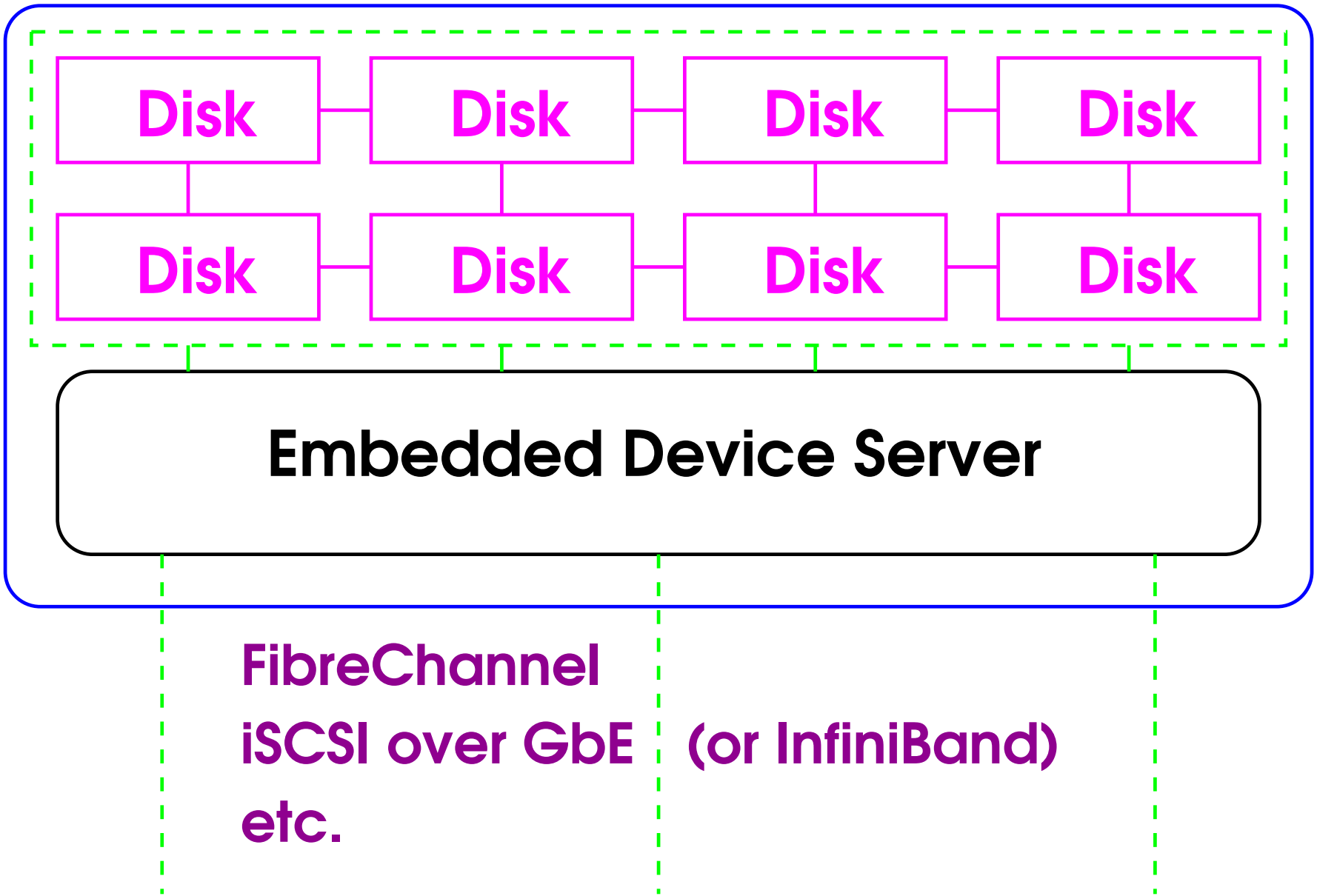
So far, so good – but also So What?

Need to share filesystems between servers
Current solutions all have major disadvantages
NFS, AFS/DFS, CIFS/SMB, Netware, ...

Use a SAN to share at the device level?

Here be dragons!

Shared SAN


Disk
Disk
Disk
Disk
Disk
Disk
Disk
Disk
Embedded Device Server
FibreChannel
iSCSI over GbE
etc.
(or InfiniBand)

Why is That a Problem?

A version of the distributed database problem
Has been intractable for 40+ years

Parallel accesses introduce race conditions
Caching then introduces inconsistency

If done wrong, can even corrupt the filesystem
Same problem occurs with crashes and hangs
Including when a user flips the power switch

All problems are probabilistic (not repeatable)

Overview of Next Slides

- Start by describing original Unix I/O model

Was used by Microsoft until recently

- Then describe hacks used on SMP systems

- Then why simply sharing devices doesn't work

Will given only a few examples of failures

- Then onto how SAN filesystems actually work

And what their constraints and uses are

Original Unix I/O Model

In this context, Microsoft systems are Unix–like
I don't program them, so I shall describe POSIX

- All system calls are atomic
- The kernel is a single thread
- All I/O uses a single file cache

$\Rightarrow$ applications see a consistent filesystem

fsck restores filesystem integrity after a crash
This does not restore application–level consistency

SMP Issues

Simplest to use a single CPU for all I/O
That is too slow, so need to use several CPUs

SMP kernels include a variety of ad hoc locks
Modern filesystems are often journalled

Reasonable protection against filesystem corruption
Still application–level inconsistencies and races

Rare on small SMP systems – say, ≤ 8
Typically seen only by hard–core HPC people

Distributed systems

Problems may be thousands of times more likely

- Global locks are completely unacceptable
- Each system maintains its own cache
- Latencies are 10–1,000 times larger

Just about works for co–operating applications
Ones designed to avoid inconsistencies and races

Even then, filesystem corruption does occur
Quite often when one system crashes or hangs

Thousands of Times?

Race conditions need 2+ ‘simultaneous’ events
Probability proportional to square of frequency

One event lasting 0.003 seconds every 5 minutes
2 clients involved in causing events
$\Rightarrow$ a couple of race conditions a year

One event lasting 0.03 seconds every 30 seconds
20 clients involved in causing events
$\Rightarrow$ a race condition every few minutes

The above is a 100,000–fold increase

Appending to a File

Standard sequence of operations:

- Remove block from free block list
- Write data to new block on disk
- Update inode with block ptr and timestamp

Chaos if two servers do first step in parallel
Block allocated twice – and one might be a directory

That leads to serious filesystem corruption

Updating a File

Standard sequence of operations:

- Read previous block contents from disk
- Write new data to block on disk
- Update inode with address and timestamp

No possibility of filesystem corruption
Except when the block is in a directory

Race conditions in a dozen ways
High chance of application–level inconsistency

Deleting a File

Standard sequence of operations:

- Remove entry from parent directory
- Check use count in inode; if zero:
  - Restore blocks to free block list
  - Release inode for reuse

Chaos if the file is in use at the time
The inode and free blocks may be reallocated

That leads to serious filesystem corruption

Directories

No locking of directory access and update
No way of synchronising all directories in path

Chaos if a directory changes while being read
POSIX implies sanity, but is just plain wrong

- Most directories fit in a single block
- Most utilities read directories in a burst

$\Rightarrow$ Problems rarely occur on SMP systems

Pathname resolution

Implemented in kernel and usually quasi–atomic

On SANs, even pathname resolution is risky
Each directory level needs a separate read

What happens if a multi–block directory is changed?
Especially in B–tree directory implementations

That is Bad News – potentially even for security

Horrible Example

Master node:

mv /dump /dump.old ; mkdir /dump

Spawn dump request to all clients

On each client node:

cpio –o /local > /dump/cpio.$$

Return success indicator

On master, wait for all clients to finish

Only if all of the client dumps succeeded:

rm –r /dump.old

Now what if the directory update was delayed?

The Usual Resolution

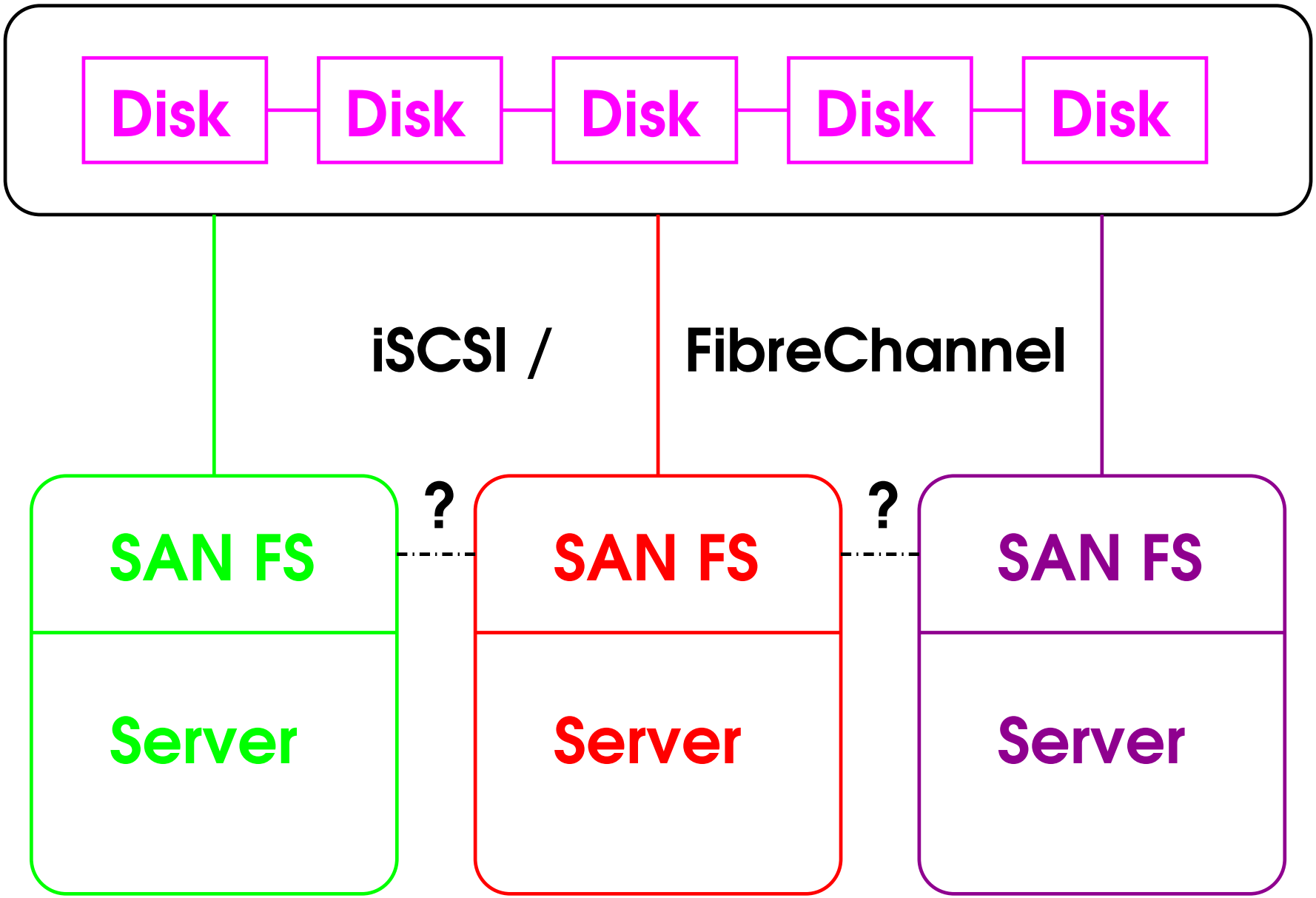
Use a SAN filesystem (i.e. a distributed one)
Every system must run the client code

Contains enough handshaking to avoid corruption
This may or may not involve global locking

It may not prevent application–level inconsistency
Different SAN filesystems have different rules

Not all POSIX–conforming programs will work
Most mailers and job schedulers don't

SAN Filesystem



Common Restrictions

- Parallel access to same object must be read–only

And that includes access to directory trees
Major restriction on find, make, tar etc.

- Timestamps may not be consistent with data

Don't trust fsync on SAN filesystems

- There may be a delay as updates become visible

Don't synchronise I/O by using message–passing
Or vice versa ...

Horrible Warning!

Remember the Horrible Example?

It can still happen – and I have seen it do so
You often need explicit synchronisation

E.g. pass the inode number of /dump
Clients then loop until 'ls –i' matches
Or whatever …

And leave 'make –j' to masochists …

POSIX Conformance

Some have levels of POSIX conformance
For example, Lustre / HP SFS does

The more conformance, the less scalability
Bad news for hard–core HPC people

Remember that POSIX doesn't specify much
Most applications also rely on de facto behaviour
[E.g. parallel reading and modifying a directory]

And you won't get de facto conformance

Administration Constraints

Identical SAN FS versions and configurations
Potential chaos if you get them out of step

Whole SAN may fail if one system crashes or hangs
$\Rightarrow$ One person had better manage all systems

Avoiding corruption leads to space leaks
Fairly frequent recompaction may be needed

OK for dedicated clusters in machine rooms
Not good news for workstations on users' desks

Can We Simplify?

The general case is always complicated
Are there simpler cases that work better?

The answer is, of course, "yes, but ..."
Very few are of much use in academia
Use them in a complicated way once, and ...
[Probabilistically, of course]

Single–writer systems are the most useful
[Including most forms of "replication"]
Hot failover is commonly touted by salesmen

Single-Writer Systems (1)

Only one system with update privileges
All others mount the filesystem read–only

Avoids the worst of the problems
No filespace corruption or data inconsistency
[At least in theory ...]

The read–only systems may see inconsistencies
Solution is to remount the shared filesystem

Single-Writer Systems (2)

Very limited experience around the University
Suitable for high–profile Web servers etc.

The Sanger did (does?) this with GPFS
All really big servers (like Google) do it

You are advised to proceed with caution!

- Please tell me of your experiences

Hot Failover

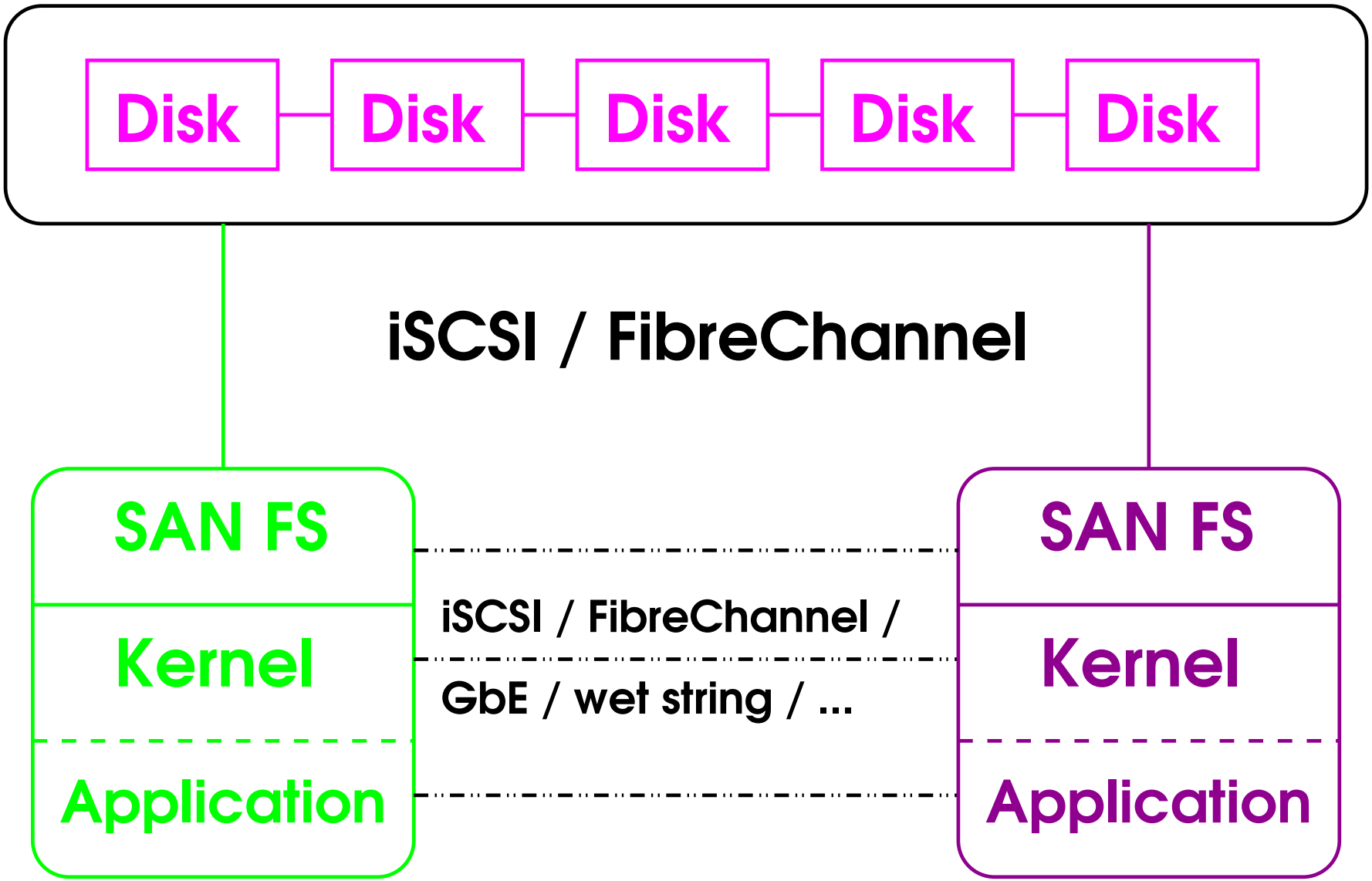
A SAN is an important component of this
But it is not a solution on its own

Need the whole system to be designed for failover
Reason is that critical state is kept at all levels

- In the actual disk blocks (obviously)
- In the filesystem cache (obviously)
- In the kernel (e.g. state in file descriptors)
- In the application (more state, locks etc.)

Most experiences with this are not positive :–(

Hot Failover



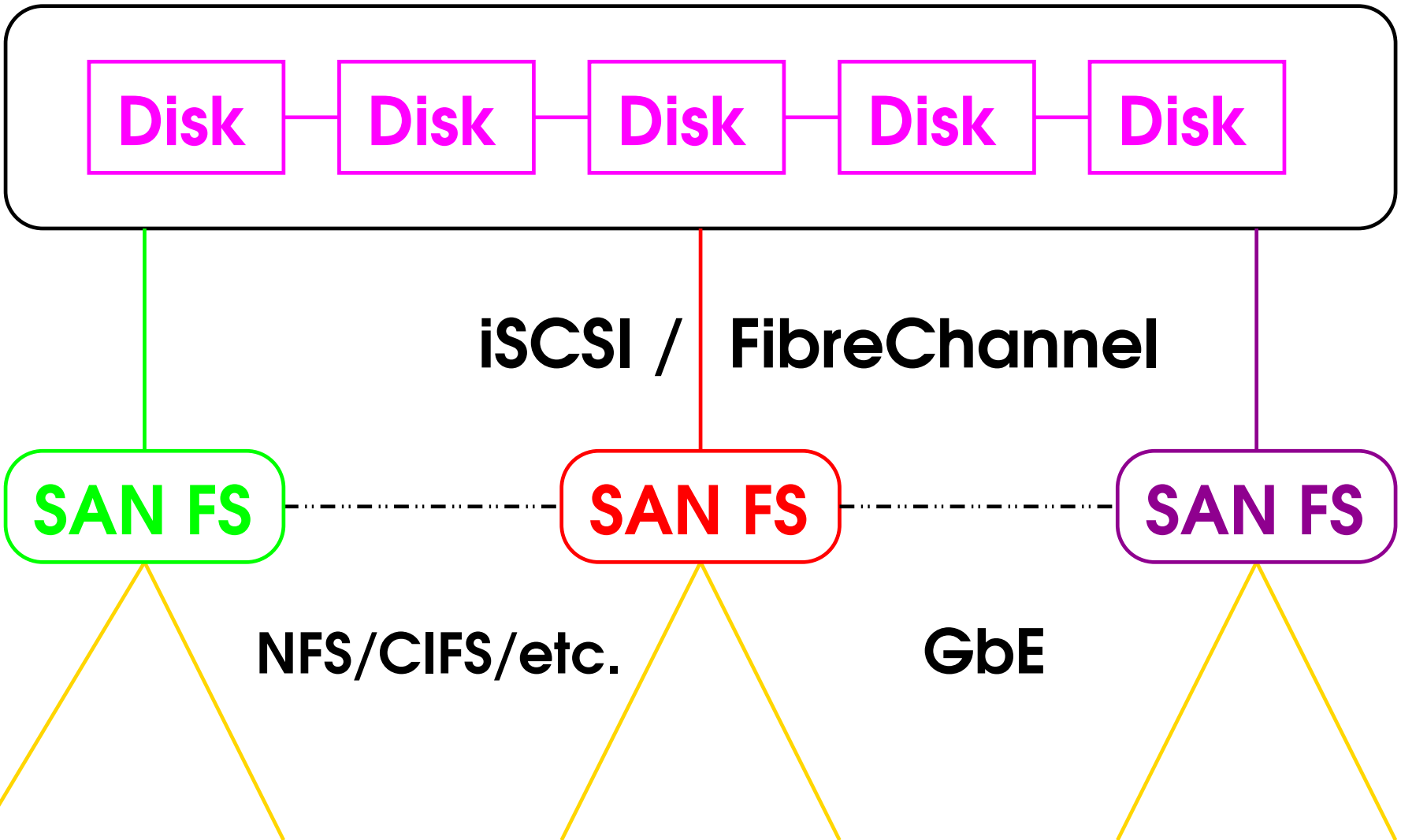
Parallel File Servers

For when a single file server is inadequate
Needed for many clients and heavy I/O only
Provides scalability, sometimes hot failover

Hardware is cheaper than a large SMP server
Perhaps not when including software and support

This is major and growing use in the outside world
Not just in research HPC but in commerce

SAN–Based Fileserving



True Parallel Filesystems

I mean ones designed to increase performance
Especially that of parallel applications

Directly connected to each node of a cluster
Extreme HPC – for I/O–bound applications
Most SANs don't support this type of use

Genuinely parallel (non–POSIX) filesystem
Need to design application for the filesystem
As an example of this, look at MPI–2 I/O

Common Parallel Filesystems

Lustre is the HPC market leader (so they say)

Experiences with IBM GPFS are very mixed
Good: 1 Linux + 1 AIX; bad: 1 Linux + 1 AIX

No personal experience with RedHat GFS,
Panasas, SGI VxFS, Lustre or others

TerraScale looks to be a good design

I was not impressed by Ibrix – to be polite

Lustre / HP SFS

I spent some time investigating this

HP SFS is a Lustre offshoot
Both fiendishly complicated in all respects

Sun have bought Lustre and CFS
So, bye, bye QFS – few people will cry

Must have full support contract or dedicated expert
Probably only from Sun or HP

RedHat GFS and GFS2

From http://www.redhat.com/gfs/:

> Fully POSIX–compliant, meaning applications don't have to be rewritten to use GFS

*********! – er, sorry, I meant twaddle!

It seems to provide block–level consistency

It surely must provide filesystem integrity

> But I can find no references as to how ...

It seems to be still a research project

pNFS (Parallel NFS)

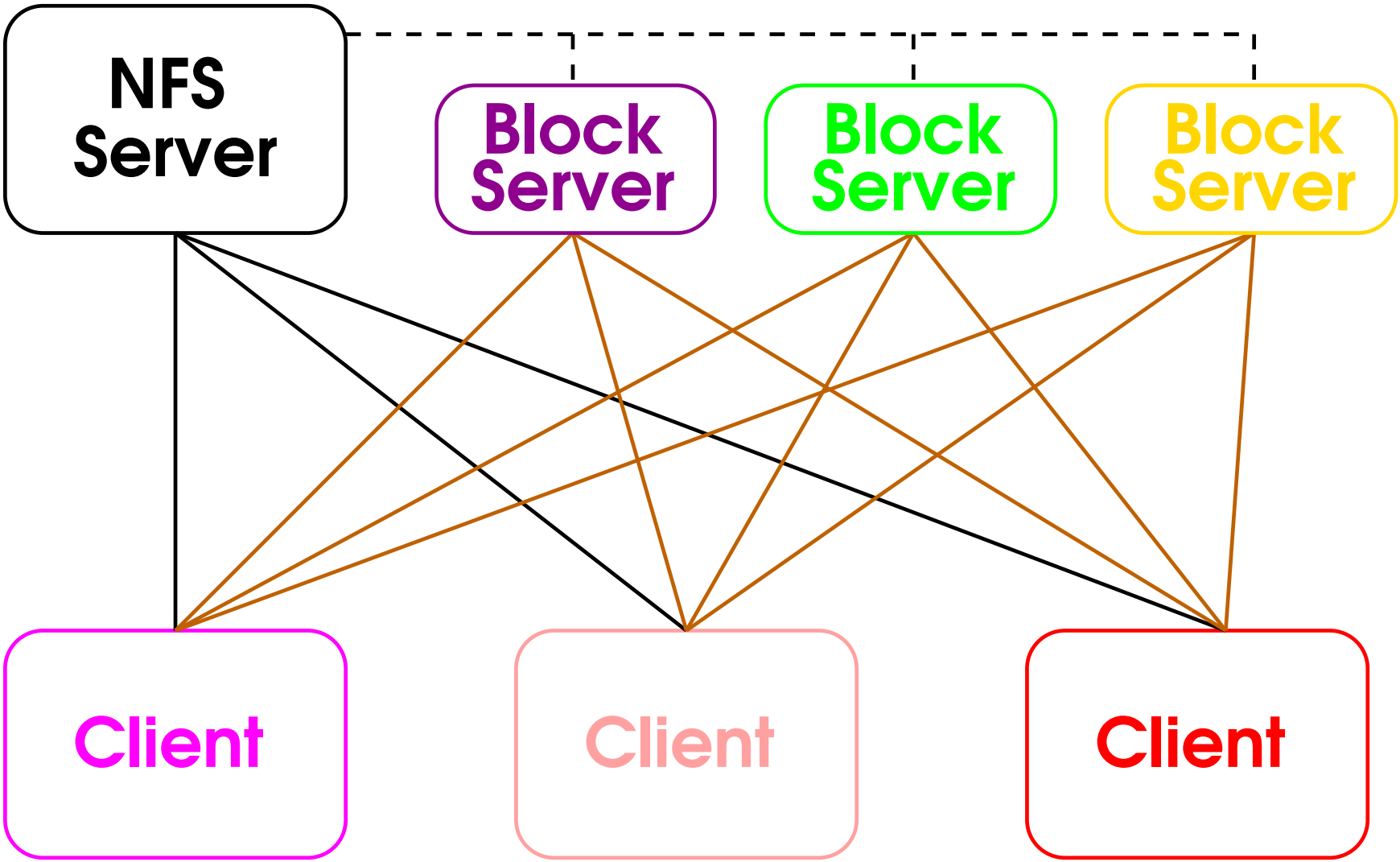
Extensions to NFSv4 by Internet RFCs
http://tools.ietf.org/wg/nfsv4/

"Object" extension more interesting to commerce
Semi–standard "Object Storage Devices" protocol

Clients have direct "block" access to storage
Massive scope for inconsistency and confusion
Possibly too clever by half and may flop, horribly

Partly driven by Panasas, to compete with Lustre
http://www.pnfs.com/

pNFS Block Access Design



Single-Writer Filesystems

No known users of Novell, Oracle CFS etc.
The PWF uses Novell in non–parallel mode

The Google file system seems to be private

There are doubtless many others I haven't found

Most people use a more general SAN filesystem
Which may even work, when in single–writer mode

Microsoft DFS and FRS

One (?) department uses Microsoft DFS, happily
My comments are based on Microsoft documents

Like a Simple SAN, but provides a single view
Generally, each directory lives on one server
Client access is via CIFS/SMB to servers only

Assumes and relies on a single–writer model
Uses replication for multiple servers
Not efficient for heavily updated directories