File IO and Network IO Challenges and Solutions

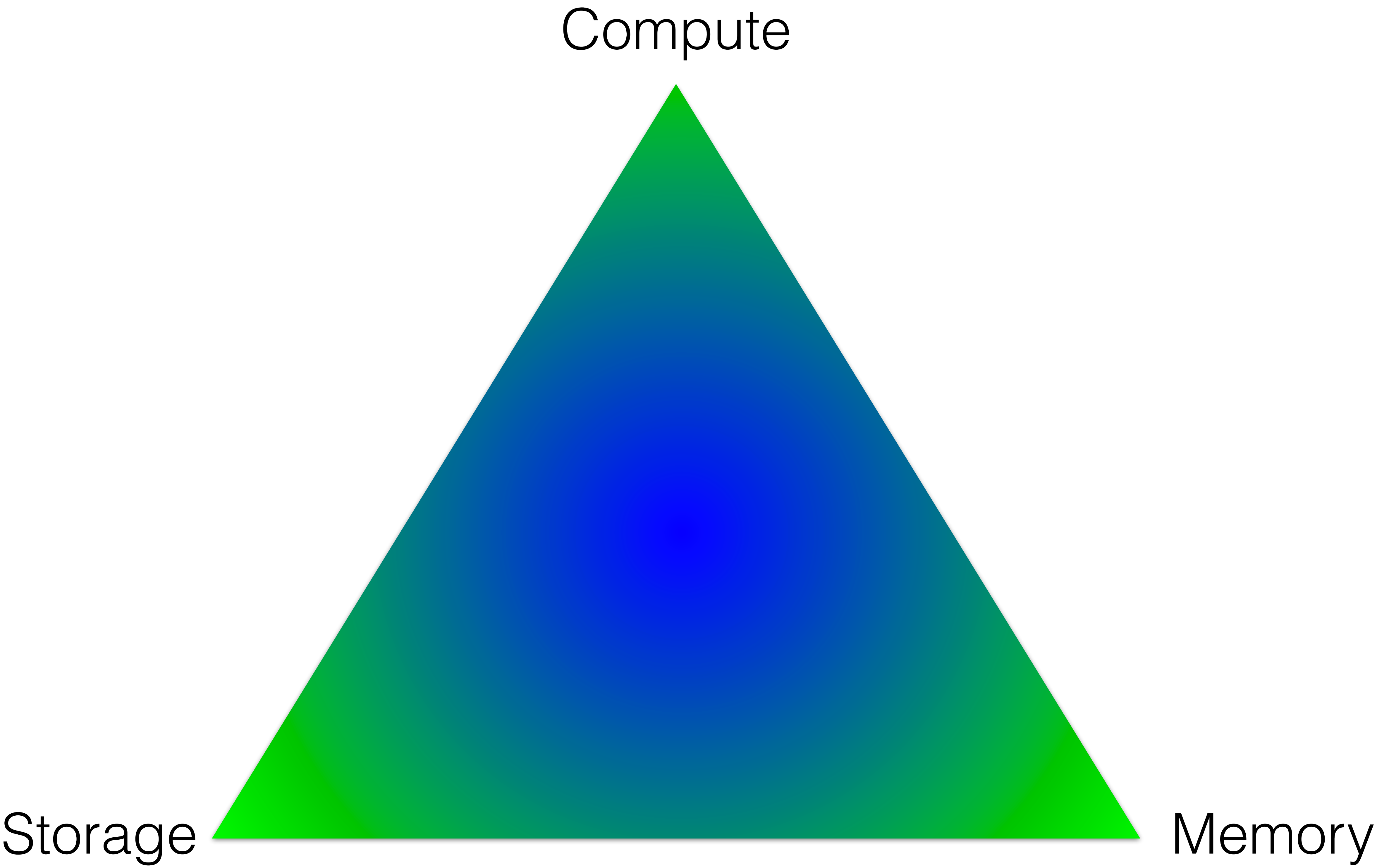
Warwick RSE

December 2024

Problems

- **All** computers can process data much faster than they can transfer it between memory and drive
- Intel Pentium Gold G6400 - Slowest CPU money can buy in 2024 (£80)
  - Transfer rate from memory to CPU 40GB/s
  - CPU can process data faster than memory can provide it
- KIOXIA KCMY1RUG30T7 - Fastest single drive money can buy 2024 (£6000)
  - Transfer rate from drive 30GB/s

Design Balance



Problems

- Difficult IO problems can cause
  - Increase in total time to solution for a problem
  - Limit on scaling of parallel code
    - Generally hard to get actual scaling of IO - a **good** solution writes data at the same rate as a single thread/rank
  - Degradation of performance for everyone else on a shared system

Terminology

Basic Terminology

- Memory - Working space that the computer uses for running programs
- Storage - Long term storage intended for keeping data that is not currently being worked on
- Filesystem - A system of data and metadata that allows storing of your information
- Mount - The act of making the data stored in a filesystem available to users (i.e. the filesystem was mounted)

Basic Terminology

- Hard Disk/HDD - An electromechanical storing information magnetically on the surfaces of several spinning platters. Normally connected using SATA in modern computers
- Solid state disk/SSD - A solid state disk that stores data in persistent solid state cells. Can be connected using SATA or NVMe (very occasionally PCIe) interface.

Basic Terminology

- SATA - Serial AT Attachment (AT doesn't have a formal meaning as an acronym, but goes it back to the IBM PC-AT in 1984). A way of connecting drives to a computer using a small 7 pin connector

- SAS - Serial Attached SCSI (Small Computer System Interface) - Like SATA but for enterprise applications

- PATA - Parallel AT Attachment an older (almost obsolete!) standard for connecting drives to computers using 40 wires

- PCIe - Peripheral component interface express - An interface bus for connecting the computer to external devices

- NVMe - Non-volatile Memory Express - An interface for communicating with storage devices over the PCIe bus

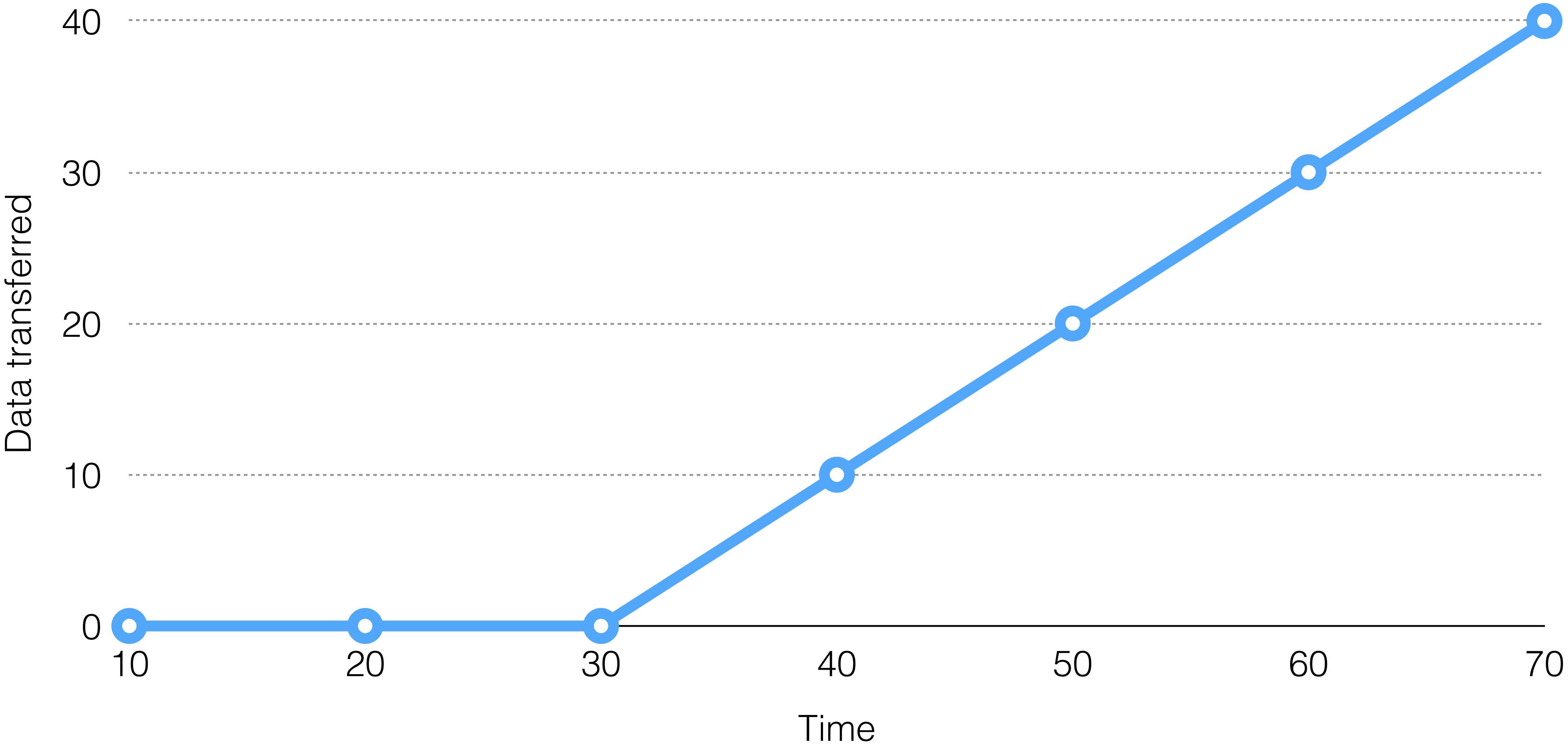
Basic Terminology

- Sequential access - Requesting data that is immediately after the data that you have just requested
- Random access - Requesting data that is not connected to data that is previously read
- Sequential access speeds are almost always faster than random access speeds

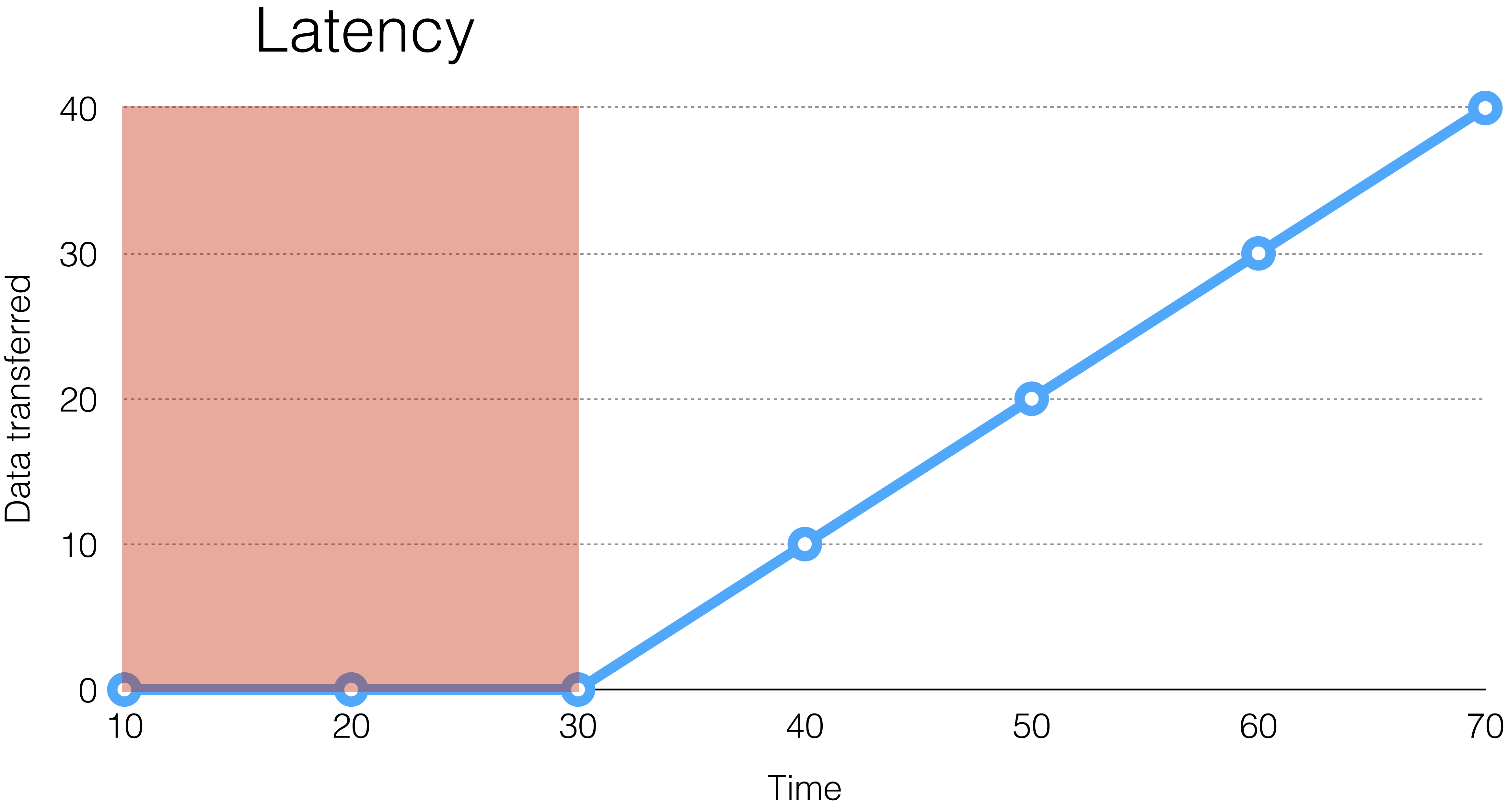
Terminology

- Latency - the time between a program asking for data and it being made available, usually measured in seconds (or milliseconds etc.)
  - One measure of network latency is the “ping time”
  - High latency is bad
- Bandwidth - the rate at which data is transferred once it is flowing, usually measured in GB/s (or MB/s etc.)
  - So you also have a higher bandwidth asking for data from memory than from disk
  - High bandwidth is good

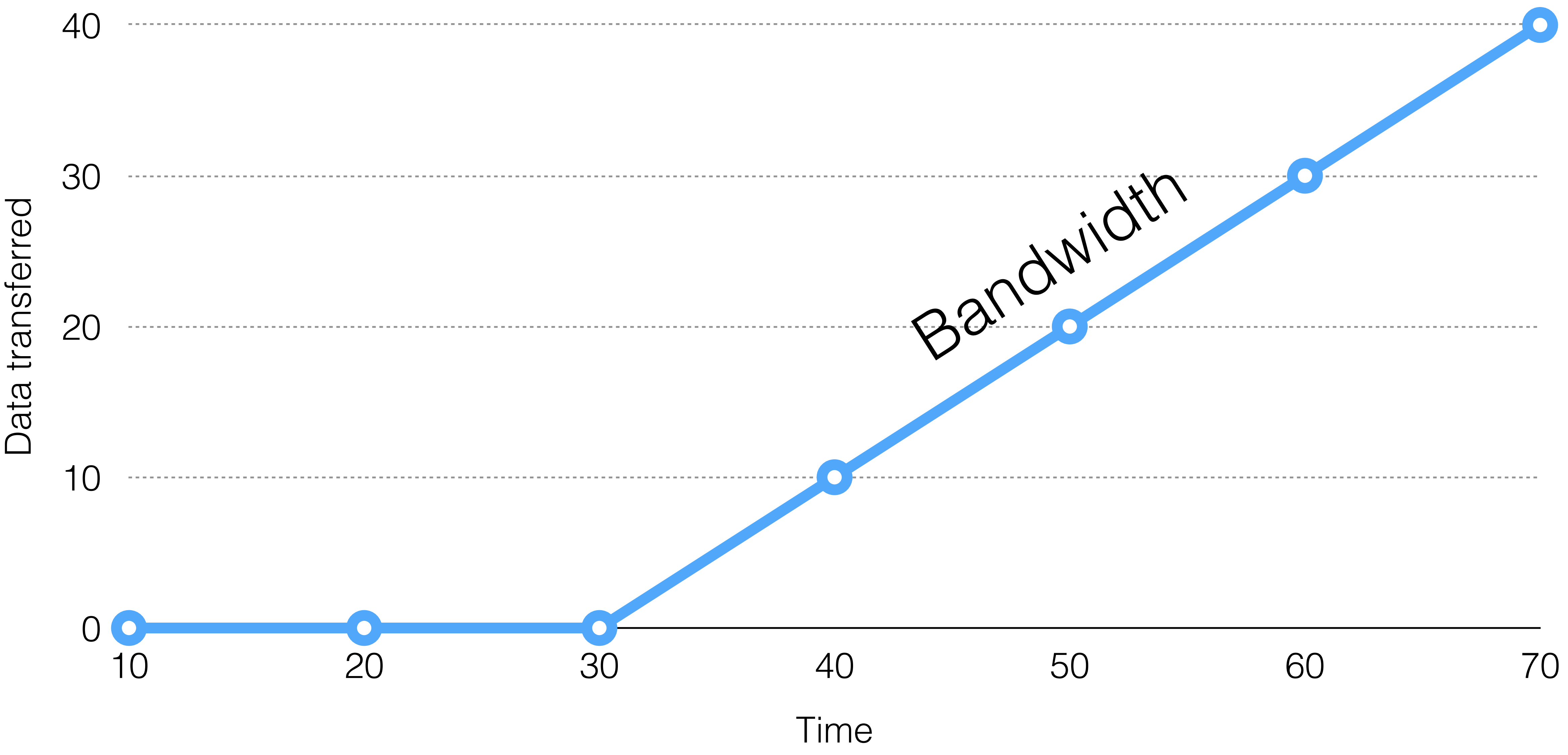
Latency



Latency



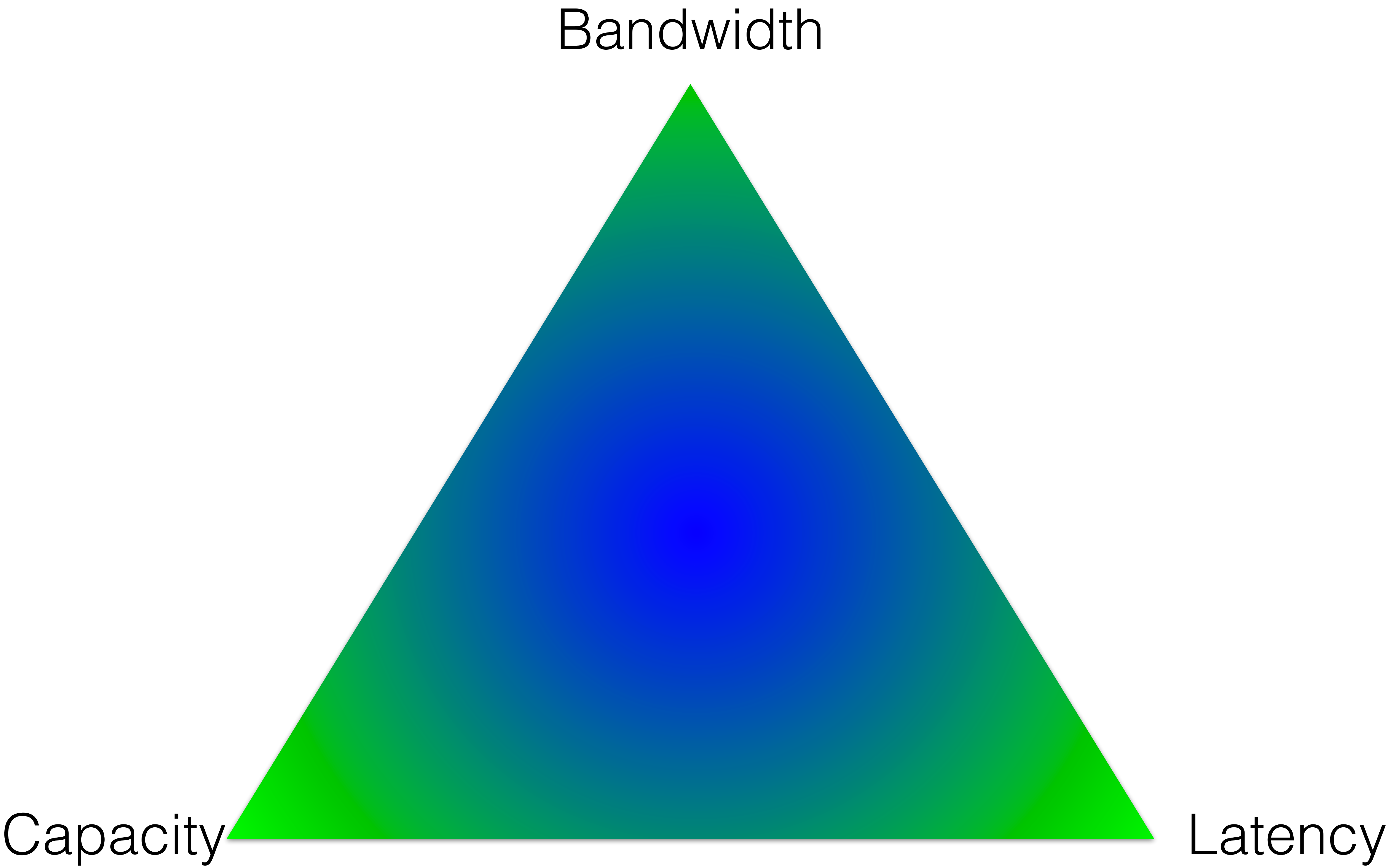
Bandwidth


Bandwidth
Data transferred
40
30
20
10
0
10
20
30
40
50
60
70
Time

Terminology

- Bandwidth matters for large files
- Latency matters for small files
- SSDs have higher bandwidth and lower latency than HDDs
    - Starting to get SSDs replacing HDDs in high capacity uses, but they are still **much** more expensive
    - 6TB Enterprise SSD - £1800
    - 6TB Enterprise HDD - £180
- A given system will generally be tuned for latency, bandwidth or capacity
    - Some things improve all of them but generally either at cost or by being less reliable

Design Balance



What happens?

What seems like happens

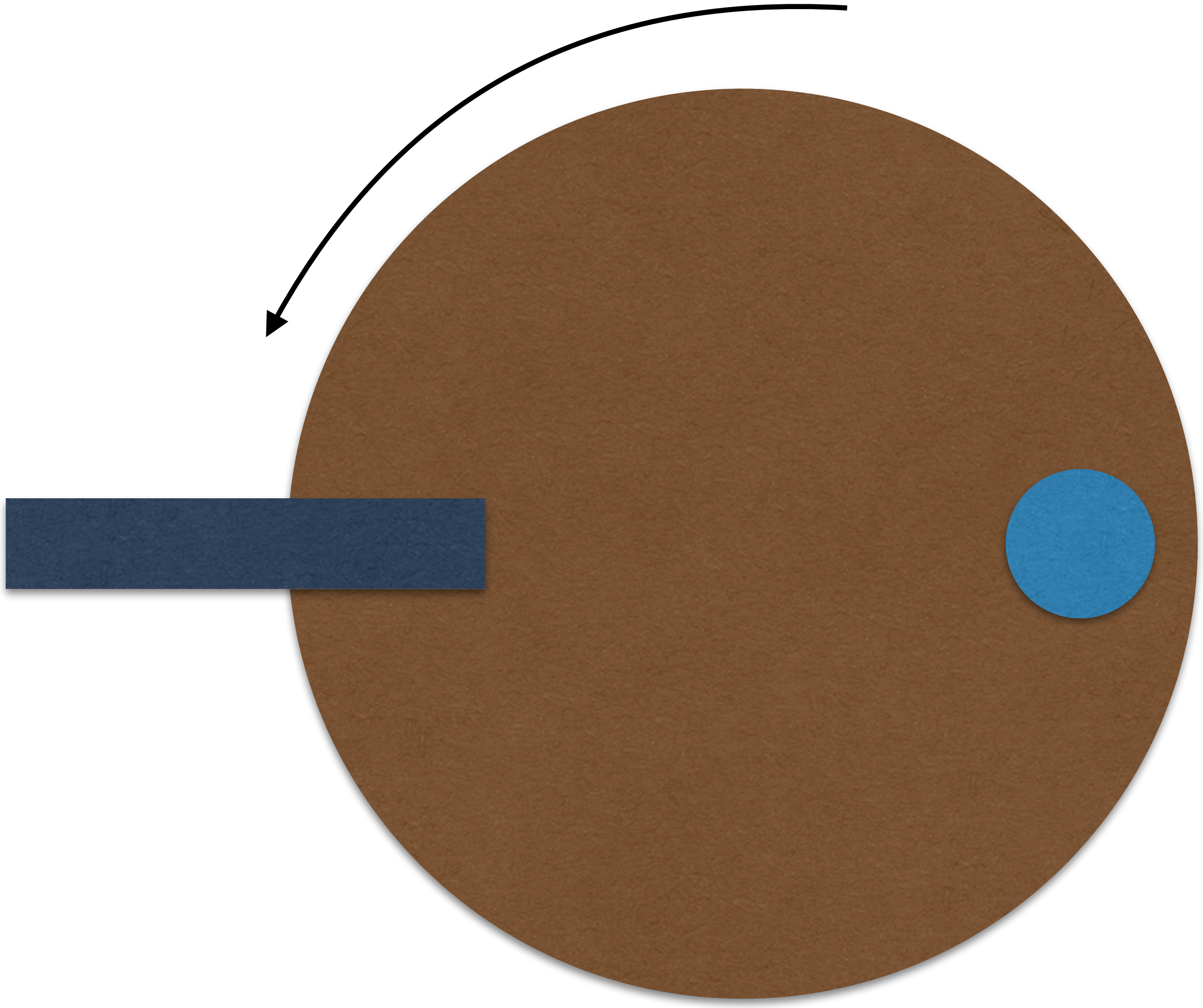
- You call a write function in your code
- The data gets written to disk

Closer to reality

- You call a write function in your code
  - Almost every language only specifies that this function will return when you can safely reuse the input variable
- Your language may require (C++,C) or permit (Fortran,Python) buffering in the language
- Once your language writes the data your OS may buffer output
- The filesystem may the buffer output
- There may be a RAM buffer in the hard disk unit
- Then finally data winds up on the disk

Why buffers?

- Can only write (or read) when the right bit of the disk is under the hard drive head (SSDs are different but we'll ignore them here)
- Long and unpredictable wait for every write
    - Latency!
- Performance of code becomes unpredictable



How do buffers help?

- Buffers combine output at various levels
- Multiple writes are replaced by a single write
- Rather than paying disk latency for every write you only pay it once when the buffer is actually written
- Your program can continue as soon as the output data is safely in the buffer
    - "Effective" bandwidth massively increased

What are the problems?

- Mostly buffers are unambiguously a good thing
- Even if you are both reading and writing data from a file it'll be sorted out - you will never see an "old" version of the file when reading
- The only problem is that if your program crashes before the (language/runtime) buffer is written out
  - You can manually flush the buffer to avoid this but this removes the performance benefit
  - Your code should only crash while you are debugging it so write your codes' debug mode to flush log files etc. don't use it in production mode

Any surprises?

- A couple
- Closing a file generally flushes output
- Output to screen generally flushes the output buffers when a newline character is received (UNIX-like systems at least)
- In C++ writing std::endl to a stream **always** causes the buffer to flush
  - "Inserts a newline character into the output sequence os and flushes it as if by calling os.put(os.widen('\n')) followed by os.flush()"
  - If you want a platform independent newline then use os.widen('\n')

Takeaway notes

- File buffers are ubiquitous and generally very good for performance
  - You can sometimes tune the sizes of some of them which for large data writes can be helpful
- It is tempting to turn them off because of the data loss problems, especially for debugging logs etc.
  - Make this an option if you do it! You don't want it on for normal production
  - Look into alternatives
    - Code that fails gracefully under all circumstances - buffers flush properly
    - Logging to external program that doesn't crash - database IO

Files and Filesystems

File systems

- All computers nowadays (and really since the 1960s) have **filesystems**
- That is a part of the computer operating system that manages files and directories on the disks
- It handles storing data to physical locations on the disk, mapping filenames to those physical locations and handles the hierarchical directory structure
- All of this information about how it does this is also stored on the disk (in normal systems, there are odd ones)
  - **metadata**

File systems

- On UNIX systems like Linux the metadata is mostly in the form of things called **inodes**
  - Index nodes
- Each inode refers to a single file or directory on disk
- When a file is created (and sometimes when it is modified) the **inode table** has to be modified to show where the file is being stored
- Often about 5% of a disks total capacity is used to store inode information - it's one reason why hard drives don't have their nominal capacity when you finally see them in the OS

Performance

- There is a cost for updating an inode
    - In general the inode table won't be near your data on the disk
- This cost is O(1) (i.e. independent of) filesize
    - It takes as long to do for a small file as it does for a large file
- Smaller files are slower to write than larger files (per byte written)
    - Also, just opening a file can take a long time

inode exhaustion

- The other problem with many small files is that for “normal” filesystems a fixed number of inodes are created when the file system is created and more can’t be created easily

- You can exhaust the inodes by writing many small files even if you still have space on the disk for more data

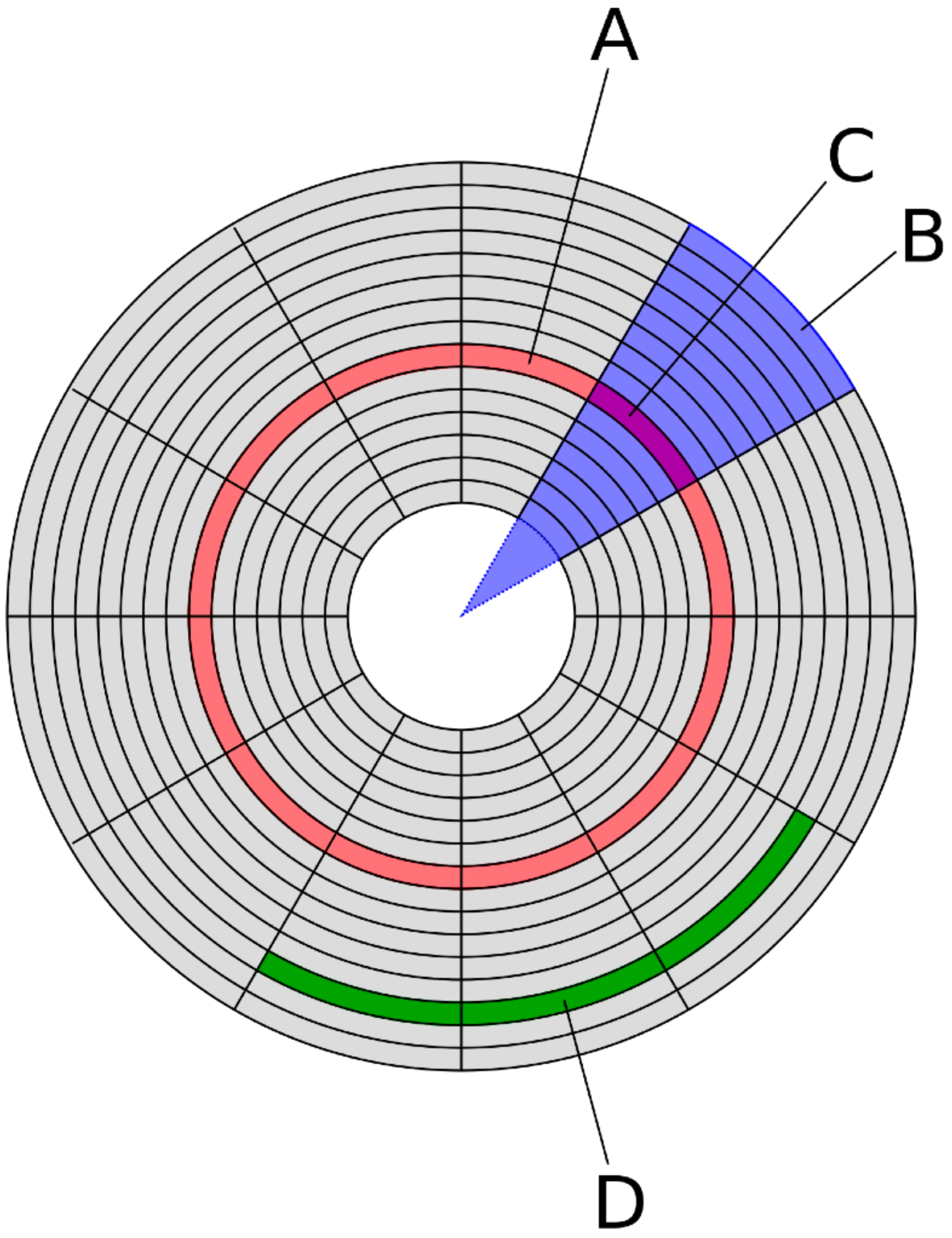
- Filesystems can be tuned for any given purpose but on shared systems it is always tuned for general use (mix of small and large files)

Takeaway notes

- Filesystems are always a tradeoff between use cases
- Very small files are often disproportionately slow to write
  - Overhead of creating file
  - Overhead of writing metadata
- Even if you can tolerate the performance drop you can run into scenarios like inode exhaustion
- **Don't write many small files!**

Blocks and Block Devices

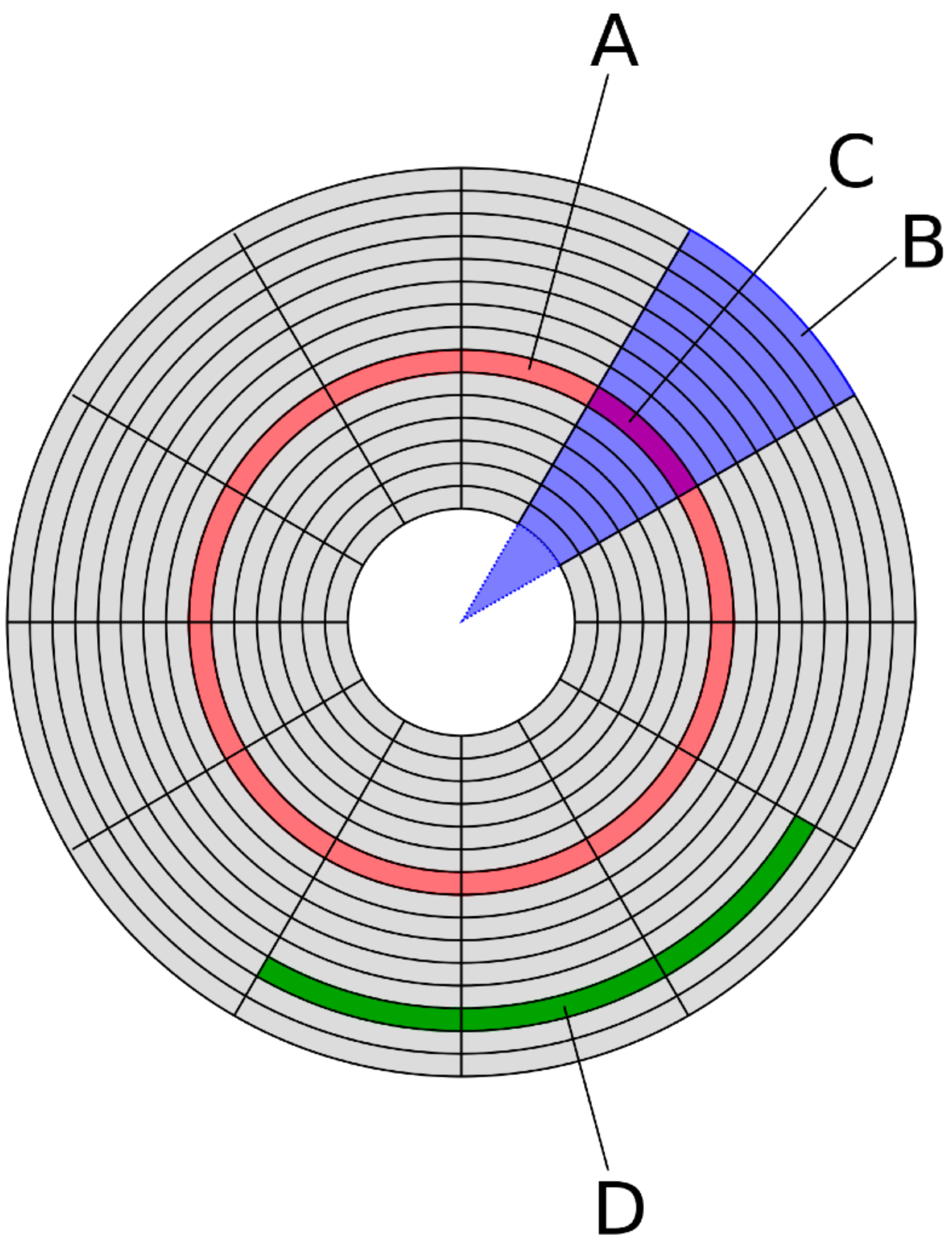
File system blocks



Heron2/MistWiz
Wikipedia, used under license

- Filesystems are abstractions over how disks actually lay out data
- For example the left shows how data may be stored on a magnetic or optical disk (actually only true for really old ones!)
    - A - Track (or cylinder for drives with multiple disks)
    - B - Geometrical Sector
    - C - Sector (usually about 4k of data)
- So a piece of data is written to a given track on a given sector (with a given **head number** if there are multiple disks)

File system blocks



Heron2/MistWiz
Wikipedia, used under license

- D shows a **block** (sometimes called a cluster or allocation unit, but we'll stick with block)
- A block is the smallest part of the disk that the filesystem writes to or reads from
- Some filesystems allow multiple files in a block (block fragments) but the most common ones don't
- Assume that files have a minimum size
  - Typically about 4kB but can be much bigger

Takeaway notes

- Filesystems don’t write files with bitwise granularity and drives don’t store data in an undifferentiated soup of bits
- Data is generally written and read in units of the filesystem block size
  - This can be much larger than the actual size of a small file
- Small files cannot take up less than a block of storage (on most filesystems)
  - Small files take up more space than their actual contents
  - Technically larger files are always rounded up to the next blocksize boundary, but 4k on a 10GB file doesn’t matter as much as 4k on a 20byte file

Network filesystems

Network filesystems

- UNIX OSes generally make filesystem boundaries seamless
  - In Windows/DOS by default you get a different drive letter for each filesystem
- On UNIX you can make a different filesystem appear anywhere - it just looks like a directory
- So on an SCRTP machine looking at the folder */usr* you are on the local hard drive, */home* you are on a network filesystem on the machine **hermatus** and */storage* you are on the machine **nef**

Network filesystems

- There are a lot of network filesystems but two that you will “normally” encounter
    - SMB - Server Message Block - originally IBM but championed by Microsoft
    - NFS - Network File System - Originally Sun Microsystems but now an IETF open standard
- We use NFS version 4 here at Warwick for all network filesystems on the “normal” machines

A bad joke

"Hi, I'd like to hear a TCP joke.”
"Hello, would you like to hear a TCP joke?"
"Yes, I'd like to hear a TCP joke."
"OK, I'll tell you a TCP joke."
"Ok, I will hear a TCP joke."
"Are you ready to hear a TCP joke?"
"Yes, I am ready to hear a TCP joke."
"Ok, I am about to send the TCP joke. It will last 10 seconds, it has two characters, it does not have a setting, it ends with a punchline."
"Ok, I am ready to get your TCP joke that will last 10 seconds, has two characters, does not have an explicit setting, and ends with a punchline."
"I'm sorry, your connection has timed out. Hello, would you like to hear a TCP joke?

Network problems

- That shows a chunk of the problem with network filesystems
- Networks and network protocols are generally quite chatty
- Latency goes up massively as soon as things are going to move over a network
- Once the server is more than a few meters away from the client even speed of light delays can come into play on latency

Network problems

- Network bandwidth will be between 1Gbps and 10Gbps generally
    - Not a huge limit for a single user but can sometimes can be for many users
- Bigger problem is that both the server and the client actually have to do work to communicate
    - Much of the work is in opening and closing files **not** in writing so the total effort on the server is mostly controlled by the number of clients
    - Most of the rest of the effort is O(1) in the size of the write (I want to write this much data to this file, here is the data) so many small writes are bad (Buffers!)

Takeaway notes

- Network file systems are much like local filesystems in general **BUT**
  - Latency is much, much higher
  - Bandwidth is lower, especially to and from the machine you are sat at
  - Better between datacentre machines and actually better than a normal desktop in the clusters
- Network filesystems have to be designed and tuned for their intended purpose
  - The home areas of SCRTP machines are designed for interactive sessions **NOT** heavy data output

Cluster filesystems

Sulis

- EPSRC Tier-2 High Performance computing system
- Intended for high throughput and ensemble computing more than conventional high performance computing
- 21,376 compute cores
- 90 NVIDIA A100 GPUs
- 54 NVIDIA L40 GPUs
- 2.6PB of HDD storage (IBM Storage Scale/GPFS)
- 800TB of SSD storage (IBM Storage Scale/GPFS)

Cluster Systems

- Because data has to be available on all compute nodes, clusters have to use network file systems

- There are several cluster filesystems, but two main ones

  - LUSTRE - Has been owned by just about everyone at some point

  - GPFS/Storage Scale - IBM Proprietary system

    - **Sulis uses this one**

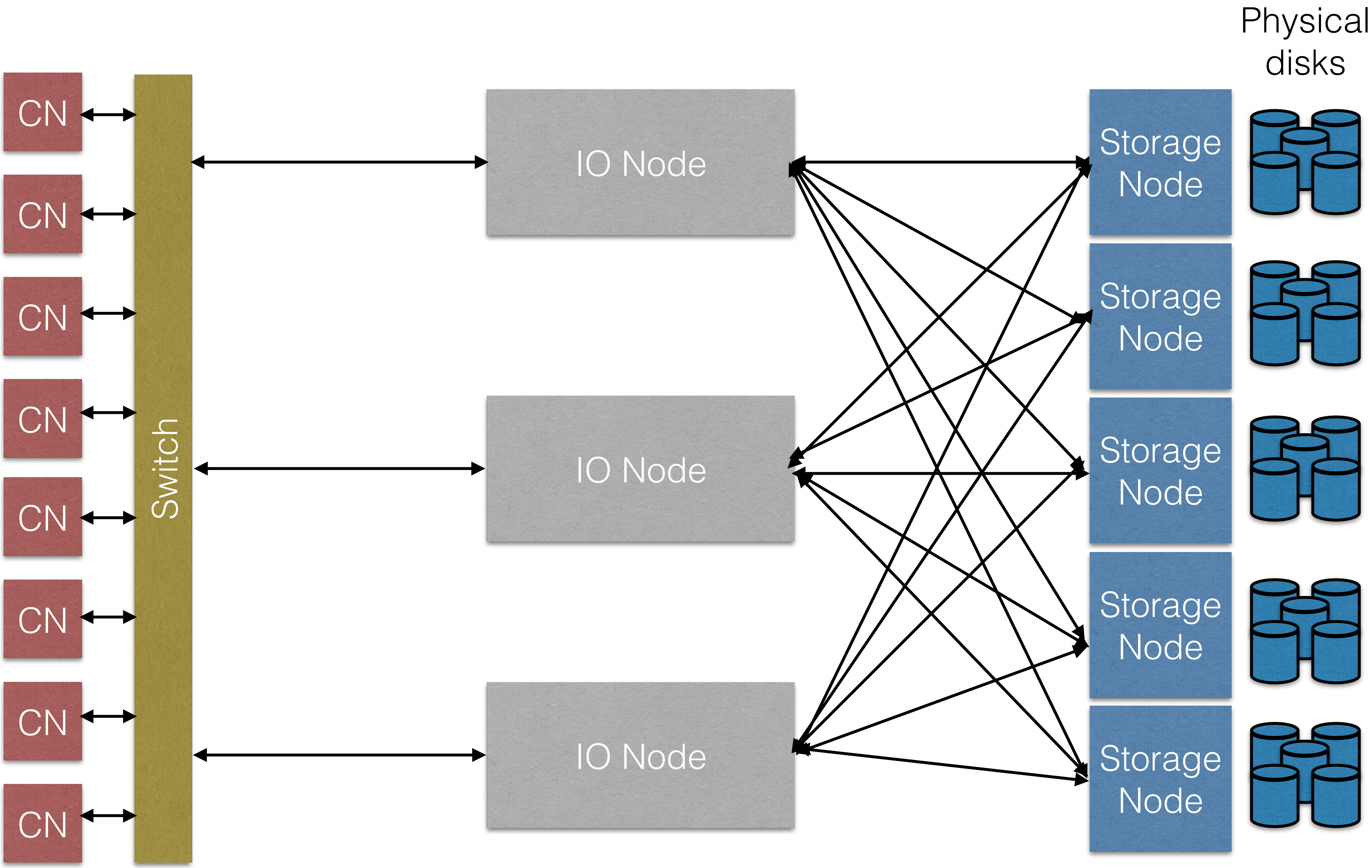
Cluster Networks

- Clusters mainly only use ethernet for systems management and control
- Usually use some variation on a system called Infiniband
  - Up to 1.2Tbit/s (100Gbit/s on Sulis) (c.f. 10Gbit/s for normal datacentre ethernet)
  - <0.6 microseconds latency (c.f. about 1-4 microsecond for 10Gbit/s ethernet, usually 1microsecond on a cluster)
- Much better bandwidth, slightly better latency
  - Advantage of big files vs. many files is even more pronounced!

How does it work?

- Basically there are two operations to writing a file
  - Updating the metadata about where the file is stored
  - Actually storing the data
- First part is mostly latency sensitive
  - Clusters speed this up by having multiple servers handling the metadata - connect to them to balance the load
- Second part will be bandwidth sensitive for large files
  - Clusters speed this up by splitting files across storage servers, so that you multiply the transfer rate by the number of storage servers

Simplified Diagram



Burst buffers

- You can attach faster solid state drives to almost any point in that structure to speed up transfer rates
    - Typically called burst buffers
- There are advantages to doing any (or all) of the options, but you pretty much just have to work within the system as it is set up
    - It is not really possible to have user configurable IO systems

Cluster Systems

- Clusters have even more unbalanced IO than normal computers
- For a more typical workstation than the earlier example, you might have 350GB/s memory bandwidth, 10GB/s drive bandwidth
- For a cluster like Sulis, the advanced IO can go up to 200GB/s, but the total memory bandwidth across all nodes is now 60TB/S
- Relatively speaking you have about 1/10 of the IO per memory transfer than you do on a “normal” workstation

Shared files

- One question that comes up regularly is "how can I share files with other researchers?"

- Basic answer is "Yes, but ask to do it"

    - Shared directories etc. can be created allowing any Sulis users to access the same files without the security risks of allowing other people access to your filespace

- Access to people who aren't Sulis users is not our remit

Shared files

- Similarly if there are common data sets that many people use, we might already have them installed
- If not we can create one for you
- Once again ask!

Conclusions

- Data IO is a major problem
- You don’t have enough bandwidth to deal with all the data that you can generate even in an ideal world
  - Data reduction before output
  - Latency hiding
  - Latency avoidance
- Rest of this workshop is saying **how**