Chapter 2 Back-of-The-Envelope Estimation

What is about?

- In SD interview, you are asked to **estimate system capacity or performance** requirement using a back-of-the-envelope estimation
- Jeff Dean: “back-of-the-envelope calculations are **estimates** you create **using a combination of thought experiments and common performance numbers** to get a good feel for which designs will meet your requirements”
- A few concepts: power of 2, latency numbers, availability numbers

Power of 2 for Storage Estimation

Power	Approximate value	Full name	Short name
10	1 Thousand	1 Kilobyte	1 KB
20	1 Million	1 Megabyte	1 MB
30	1 Billion	1 Gigabyte	1 GB
40	1 Trillion	1 Terabyte	1 TB
50	1 Quadrillion	1 Petabyte	1 PB

- Byte(B) vs bit(b). One Byte is 8 bits. An ASCII char is one Byte
- Remember the power for each. 2^10 = 1024 ≈ 1,000. Additional 3-zeros in decimal for every additional power of 10.

Latency numbers

Operation name	Time
L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	100 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	10,000 ns = 10 µs
Send 2K bytes over 1 Gbps network	20,000 ns = 20 µs
Read 1 MB sequentially from memory	250,000 ns = 250 µs
Round trip within the same datacenter	500,000 ns = 500 µs
Disk seek	10,000,000 ns = 10 ms
Read 1 MB sequentially from the network	10,000,000 ns = 10 ms
Read 1 MB sequentially from disk	30,000,000 ns = 30 ms
Send packet CA (California) ->Netherlands->CA	150,000,000 ns = 150 ms

Table 2-2

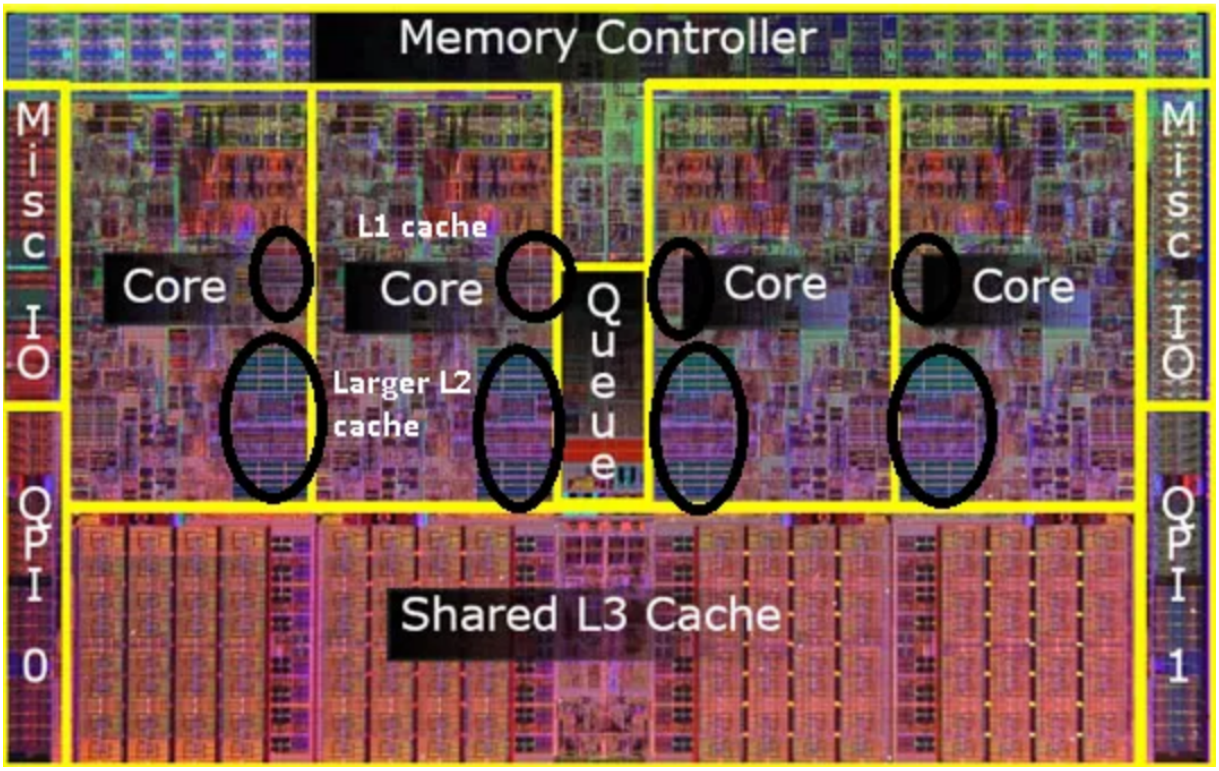
- The above numbers are typical computer operations in 2010. Some numbers are outdated as computers become faster and more powerful. (But they should still give relative fastness and slowness feelings)
- Some knowledge about computer architecture, network, OS are needed to deeply understand latency
- \L1 & L2 & L3 CPU on-die Cache: https://en.wikipedia.org/wiki/Cache_(computing)

Cache一词来源于1967年的一篇电子工程期刊论文。其作者将法语词“cache”赋予“safekeeping storage”的涵义，用于电脑工程领域。PC-AT/XT和80286时代没有Cache，CPU和内存都很慢，CPU直接访问内存。80386的芯片组增加了对可选的Cache的支持，高级主板带有64KB，甚至高端的128KB Write-Through Cache。80486 CPU里面加入了8KB的L1 Unified Cache，当时也叫做内部Cache，不分代码和数据，都存在一起；芯片组中的Cache，变成了L2，也被叫做外部Cache，从128KB到256KB不等；增加了Write-back的Cache属性。Pentium CPU的L1 Cache分为Code和data，各自8KB；L2还被放在主板上。Pentium Pro的L2被放入到CPU的Package上。Pentium 3开始，L2 Cache被放入了CPU的Die中。从Intel Core CPU开始，L2 Cache为多核共享。

当CPU处理数据时，它会先到Cache中去寻找，如果数据因之前的操作已经读取而被暂存其中，就不需要再从随机存储器（Main memory）中读取数据——由于CPU的运行速度一般比主内存的读取速度快，主存储器周期（访问主存储器所需要的时间）为数个时钟周期。因此若要访问主内存的话，就必须等待数个CPU周期从而造成浪费。

提供“缓存”的目的是为了让数据访问的速度适应CPU的处理速度，其基于的原理是内存中“程序执行与数据访问的局域性行为”，即一定程序执行时间和空间内，被访问的代码集中于一部分。为了充分发挥缓存的作用，不仅依靠“暂存刚刚访问过的数据”，还要使用硬件实现的指令预测与数据预取技术——尽可能把将要使用的数据预先从内存中取到缓存里。

CPU的缓存曾经是用在超级计算机上的一种高级技术，不过现今电脑上使用的的AMD或Intel微处理器都在芯片内部集成了大小不等的数据缓存和指令缓存，通称为L1缓存（L1 Cache即Level 1 On-die Cache，第一级片上高速缓冲存储器）；而比L1更大容量的L2缓存曾经被放在CPU外部（主板或者CPU接口卡上），但是现在已经成为CPU内部的标准组件；更昂贵的CPU会配备比L2缓存还要大的L3缓存（level 3 On-die Cache第三级高速缓冲存储器）。

- CPU running speed >> Memory access speed
- CPU on-die cache: CPU don't need to frequently access memory, which result in CPU idle
- Size: L1 < L2 < L3
- 

- Branch predictor https://en.wikipedia.org/wiki/Branch_predictor
  - 在计算机体系结构中，**分支预测器**（英语：Branch predictor）是一种数字电路，在分支指令执行结束之前猜测哪一路分支将会被执行，以提高处理器的指令流水线的性能。使用分支预测器的目的，在于改善指令流水线的流程，就像一家公司的员工提前预测公司所需要的东西，即交付不同单位进行准备工作，而那各个部门之间的等待交办的时间大大地缩短，整个公司的效率就会提高了。现代使用指令流水线处理器的性能能够提高，分支预测器对于现今的指令流水线微处理器获得高性能是非常关键的技术。

条件分支指令通常具有两路后续执行分支。即不采取（not taken）跳转，顺序执行后面紧挨JMP的指令；以及采取（taken）跳转到另一块程序内存去执行那里的指令。

是否条件跳转，只有在该分支指令在指令流水线中通过了执行阶段（execution stage）才能确定下来。

如果没有分支预测器，处理器将会等待分支指令通过了指令流水线的执行阶段，才把下一条指令送入流水线的第一个阶段—取指令阶段（fetch stage）。这种技术叫做流水线停顿（pipeline stalled）或者流水线冒泡（pipeline bubbling）或者分支延迟间隙。这是早期的RISC体系结构处理器采用的应对分支指令的流水线执行的办法。

- Branch mispredict:

  分支预测器猜测条件表达式两路分支中哪一路最可能发生，然后推测执行这一路的指令，来避免流水线停顿造成的时间浪费。如果后来发现分支预测错误，那么流水线中推测执行的那些中间结果全部放弃，重新获取正确的分支路线上的指令开始执行，这招致了程序执行的延迟。

  在分支预测失败时浪费的时间是从取指令到执行完指令（但还没有写回结果）的流水线的级数。现代微处理器趋向采用非常长的流水线，因此分支预测失败可能会损失10-20个时钟周期。越长的流水线就需要越好的分支预测。

- Mutex lock/unlock (互斥锁): https://en.wikipedia.org/wiki/Mutual_exclusion

  互斥锁（英语：Mutual exclusion，缩写 Mutex）是一种用于多线程编程中，防止两条线程同时对同一公共资源（比如全局变量）进行读写的机制。该目的通过将代码切片成一个一个的临界区域（critical section）达成。临界区域指的是一块对公共资源进行访问的代码，并非一种机制或是算法。一个程序、进程、线程可以拥有多个临界区域，但是并不一定会应用互斥锁。

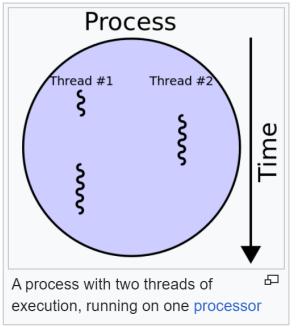
  需要此机制的资源的例子有：旗标、队列、计数器、中断处理程序等用于在多条并行运行的代码间传递数据、同步状态等的资源。维护这些资源的同步、一致和完整是很困难的，因为一条线程可能在任何一个时刻被暂停（休眠）或者恢复（唤醒）。

  例如：一段代码（甲）正在分步修改一块数据。这时，另一条线程（乙）由于一些原因被唤醒。如果乙此时去读取甲正在修改的数据，而甲碰巧还没有完成整个修改过程，这个时候这块数据的状态就处在极大的不确定状态中，读取到的数据当然也是有问题的。更严重的情况是乙也往这块地方写数据，这样的一来，后果将变得不可收拾。因此，多个线程间共享的数据必须被保护。达到这个目的的方法，就是确保同一时间只有一个临界区域处于运行状态，而其他的临界区域，无论是读是写，都必须被挂起并且不能获得运行机会。

  In computer science, a **thread** of execution is the smallest sequence of programmed instructions that can be managed independently by a scheduler, which is typically a part of the operating system.[1] In many cases, a thread is a component of a process.

  The multiple threads of a given process may be executed concurrently (via multithreading capabilities), sharing resources such as memory, while different processes do not share these resources. In particular, the threads of a process share its executable code and the values of its dynamically allocated variables and non-thread-local global variables at any given time.

  The implementation of threads and processes differs between operating systems. In *Modern Operating Systems*, Tanenbaum shows that many distinct models of process organization are possible.[2][*page needed*]

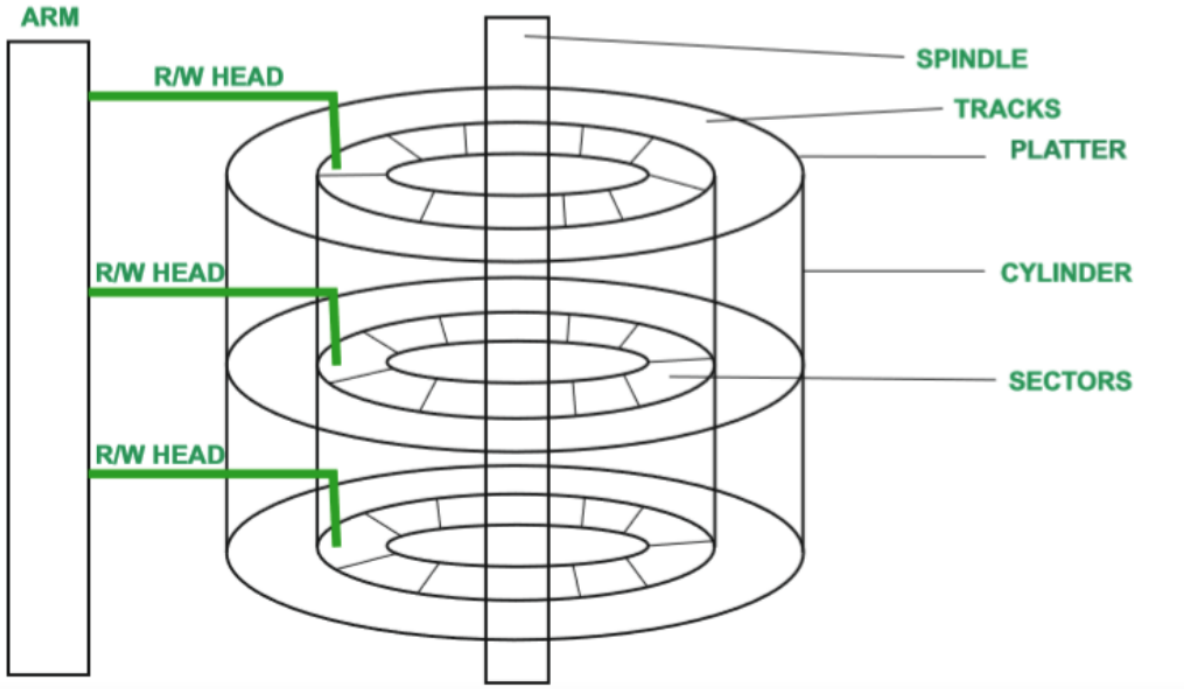
  A process with two threads of execution, running on one processor

- History [edit]

- Snappy (previously known as Zippy)
  - a fast data compression and decompression library written in C++ by Google based on ideas from LZ77 and open-sourced in 2011.
  - Compression speed is 250 MB/s and decompression speed is 500 MB/s using a single core of a circa 2011 "Westmere" 2.26 GHz Core i7 processor running in 64-bit mode.
  - The compression ratio is 20–100% lower than gzip

Send 2K bytes over 1 Gbps network	20,000 ns = 20 µs

  - How the numbers were calculated: 2k / (128MBps = 128*1024 KBps)* 10^6 = 15.25 micro sec
  - Note the difference: b vs B
  - Mbps or Gbps are commonly used by internet service providers
- Disk seeking



- Send packet CA -> Netherlands -> CA https://en.wikipedia.org/wiki/Network_packet

  In telecommunications and computer networking, a **network packet** is a formatted unit of data carried by a packet-switched network. A packet consists of control information and user data;[1] the latter is also known as the *payload*. Control information provides data for delivering the payload (e.g., source and destination network addresses, error detection codes, or sequencing information). Typically, control information is found in packet headers and trailers.

  In packet switching, the bandwidth of the transmission medium is shared between multiple communication sessions, in contrast to circuit switching, in which circuits are preallocated for the duration of one session and data is typically transmitted as a continuous bit stream.

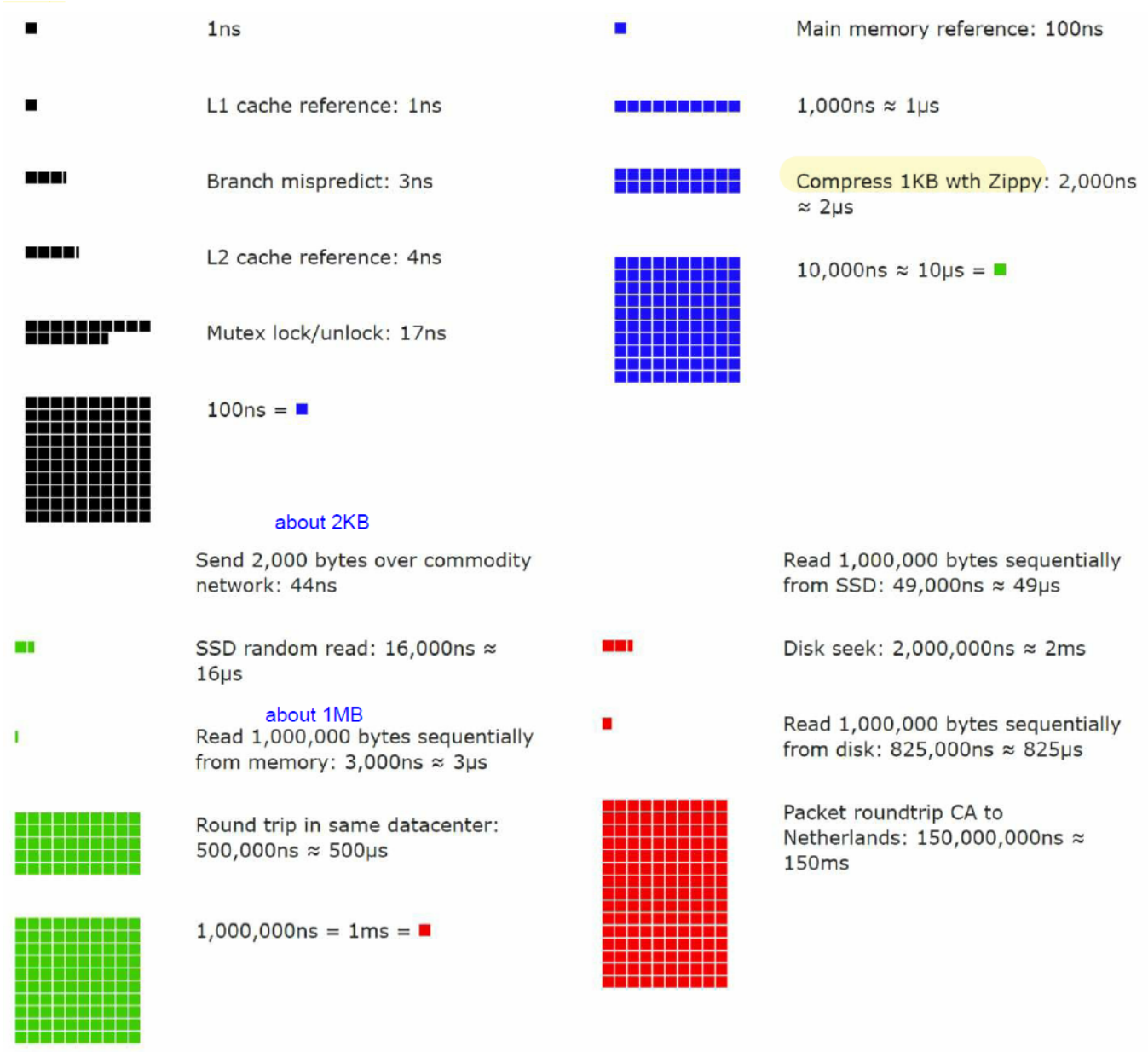
  数据包（英语：Data packet），又称分组，是在分组交换网络中传输的格式化数据单位。

  一个数据包（packet）分成两个部分，包括控制信息，也就是表头资料（header），和资料本身，也就是负载（payload）。

  我们可以将一个数据包比作为一封信，表头资料相当于信封，而数据包的数据部分则相当于信的内容。和信不同的是，有时候一个大数据包可以分成多个小数据包。

  In telecommunications, **packet switching** is a method of grouping data into *packets* that are transmitted over a digital network. Packets are made of a header and a payload. Data in the header is used by networking hardware to direct the packet to its destination, where the payload is extracted and used by an operating system, application software, or higher layer protocols. Packet switching is the primary basis for data communications in computer networks worldwide.

Visualized Latency Numbers (2020)



- Conclusions:
  - Memory is fast but the disk is slow.
  - Avoid disk seeks if possible.
  - Simple compression algorithms are fast.
  - Compress data before sending it over the internet if possible
  - Data centers are usually in different regions, and it takes time to send data between them.
- 上面这个图里的数据，是不是不支持 "compress data before sending it over the internet if possible"? 相对于这个visual, 上面Table 2-2 表格里, 二者时间更接近。另外是不是考虑一次压缩以后多次send 的情况?
- SSD (Solid-state drive) https://en.wikipedia.org/wiki/Solid-state_drive

固态硬盘或**固态驱动器**（英语：Solid-state drive或Solid-state disk，简称**SSD**）是一种以集成电路制作的电脑存储设备，虽然价格及存储容量与机械硬盘有少许差距，但固态硬盘读取的速度可比机械式硬盘的快200倍。[1] （页面存档备份，存于互联网档案馆）

可以用非易失性存储器（主要以闪存中的 NAND Flash）作为永久性存储设备，也可以用易失性存储器（例如DRAM）作为临时性存储设备。

- 固态硬盘常采用SATA、PCI Express、mSATA、M.2、ZIF、IDE、U.2、CF、CFast等接口。

Availability Numbers

Availability %	Downtime per day	Downtime per year
99%	14.40 minutes	3.65 days
99.9%	1.44 minutes	8.77 hours
99.99%	8.64 seconds	52.60 minutes
99.999%	864.00 milliseconds	5.26 minutes
99.9999%	86.40 milliseconds	31.56 seconds

Table 2-3

- High availability is the ability of a system to be continuously operational for a desirably long period of time. High availability is measured as a percentage, with 100% means a service that has 0 downtime. Most services fall between 99% and 100%.
- A service level agreement (SLA) is a commonly used term for service providers.
  - this agreement formally defines the level of uptime your service will deliver
  - AWS, Azure: 99.9% +. The more the nines, the better.
  -

Tips during Back-of-The-Envelope Estimation Interview

Back-of-The-Envelope Estimation is all about the process. Solving the problem is more important than obtaining results. Interviewers may test your problem-solving skills

- Use rounding and approximation, when needed. Precision is not expected
- **Write down** your **assumption** to be references later
- **Write down unit** for numbers to remove ambiguity
- Commonly asked estimation: QPS, peak QPS, storage, cache, number of servers, etc. Use Twitter as an example

Example: Estimate Twitter QPS and storage requirements

Please note the following numbers are for this exercise only as they are not real numbers from Twitter.

Assumptions:

- 300 million monthly active users.
- 50% of users use Twitter daily.
- Users post 2 tweets per day on average.
- 10% of tweets contain media.
- Data is stored for 5 years.

Estimations:

Query per second (QPS) estimate:

- Daily active users (DAU) = 300 million * 50% = 150 million
- Tweets QPS = 150 million * 2 tweets / 24 hour / 3600 seconds = ~3500
- Peek QPS = 2 * QPS = ~7000

We will only estimate media storage here.

- Average tweet size:
  - tweet_id 64 bytes
  - text 140 bytes
  - media 1 MB
- Media storage: 150 million * 2 * 10% * 1 MB = 30 TB per day
- 5-year media storage: 30 TB * 365 * 5 = ~55 PB