Storing data on blockchain:

In Ethereum, there is theoretically no limit for the block size. However, blockchain is inherently not meant for data storage because storing large documents will be computationally very expensive. However, there are some instances where people hacked into Bitcoin's Blockchain and stored some unexpected data. You will have to compress and store the doc/PDF/audio in Hexadecimal format.

That said, many blockchain-like solutions have been designed recently just to store data. Storj is one of the best examples. Storj is a protocol, cryptocurrency, and suite of decentralized applications that allows users to store data in a secure and decentralized manner. It uses a transaction ledger, public/private key encryption, and cryptographic hash functions for security. Storj nodes, sell resources to store and transfer information and earn Storjcoin X in exchange for their services. You can run the software and earn some extra cash for leasing your hard drive and bandwidth. Filecoin is another such solution which is yet to materialize. Enigma is yet another initiative.

More overview:

Each smart contract running in the Ethereum Virtual Machine (EVM) maintains state in its own permanent storage. This storage can be thought of as a very large array, initially full of zeros. Each value in the array is 32-bytes wide, and there are 2^{256} such values. A smart contract can read from or write to a value at any location. That's the extent of the storage interface.

I encourage you to stick with the "astronomically large array" mental model, but be aware that this is not how storage is implemented on the physical computers that make up the Ethereum network. Storage is extremely sparsely populated, and there's no need to store the zeros. A key/value store mapping 32-byte keys to 32-byte values will do the job nicely. An absent key is simply defined as mapping to the value zero.

Because zeros don't take up any space, storage can be reclaimed by setting a value to zero. This is incentivized in smart contracts with a *gas refund* when you change a value to zero.

Locating Fixed-Sized Values

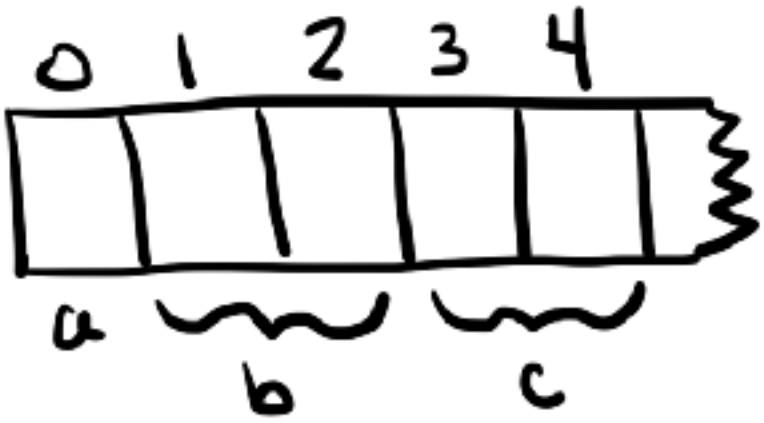
In this storage model, where do things actually go? For known variables with fixed sizes, it makes sense to just give them reserved locations in storage. The Solidity programming language does just that.

```
contract StorageTest {
    uint256 a;
    uint256[2] b;

    struct Entry {
        uint256 id;
        uint256 value;
    }
    Entry c;
}
```

In the above code:

- `a` is stored at slot 0. (Solidity's term for a location within storage is a "slot.")
- `b` is stored at slots 1, and 2 (one for each element of the array).
- `c` starts at slot 3 and consumes two slots, because the `Entry` struct stores two 32-byte values.



These slots are determined at compile time, strictly based on the order in which the variables appear in the contract code.

Using reserved slots works well for fixed-size state variables, but it doesn't work for dynamically-sized arrays and `mappings` because there's no way of knowing how many slots to reserve.

If you're thinking of computer RAM or hard drive as an analogy, you might expect that there's an "allocation" step to find free space to use and then a "release" step to put that space back into the pool of available storage.

This is unnecessary due to the astronomical scale of smart contract storage. There are 2^{256} locations to choose from in storage, which is approximately the number of atoms in the known, observable universe. You could choose storage locations at random without ever experiencing a collision. The locations you chose would be so far apart that you could store as much data as you wanted at each location without running into the next one.

Of course, choosing locations at random wouldn't be very helpful, because you would have no way to find the data again. Solidity instead uses a hash function to uniformly and repeatably compute locations for dynamically-sized values.

Blockchain data storage Tutorial:

Follow tutorial in class and use repo:
Links below to more

IFPS

IPFS is general purpose, and has little storage limitations. It can serve files that are large or small. It automatically breaks up larger files into smaller chunks, allowing IPFS nodes to download (or stream) files from not just one server like with HTTP, but hundreds of them simultaneously. The IPFS network becomes a finely-grained, distributed, easily federated Content Delivery Network (CDN). This is useful for pretty much everything involving data->

images, video streaming, distributed databases, entire operating systems and most importantly , static web sites.

because storing large documents becomes computationally very expensive. On the other hand, The IPFS protocol and its implementations are still in heavy development

Append data to trans:
Some blockchains offer the possibility to append data to a transaction within their protocols. In this case we can simply append our data to our transaction.

Summary
Each smart contract has storage in the form of an array of 2256 32-byte values, all initialized to zero.
Zeros are not explicitly stored, so setting a value to zero reclaims that storage.
Solidity locates fixed-size values at reserved locations called slots, starting at slot 0.
Solidity exploits the sparseness of storage and the uniform distribution of hash outputs to safely locate dynamically-sized values.
The following table shows how storage locations are computed for different types. The "slot" refers to the next available slot when the state variable is encountered at compile time, and a dot indicates binary concatenation:

Good Links:

Understanding blockchain storage

https://programtheblockchain.com/posts/2018/03/09/understanding-ethereum-smart-contract-storage/

Read storage into:
https://medium.com/aigang-network/how-to-read-ethereum-contract-storage-44252c8af925