Understand how the system works

What is the goal of the system? What problem does it solve? If the system improves or builds upon a previous system, what was the problem with that previous system?

> *This information can usually be found at the beginning of a paper or book section. In the case of a paper, the introduction will almost always explain the goal of the system. A related work section may discuss previous systems on which the designers are building theirs.*

What is a typical use-case for this system? In what environment does it run, or in what environment is it meant to run?

> *Typical environments in 6.033 are the Internet, a datacenter (a large network owned by a single company, for instance Google), a single user's machine, etc. The environment in which a system runs is often clear once you understand the problem it solves.*

What are the modules of the system? What modules communicate with one another? How does information flow through the system?

> *In many papers this information is described within the first few sections. There is often an architecture diagram that explains how information flows through the system. Once the basic modules of a system have been explained, subsequent sections of a paper might explain the details of each module.*

Understand how it was designed

What were the designers' priorities/principles (ex: fast performance, security, simplicity, consistency, etc.)?

> *Often you will find this information in the introduction of a paper. It can be helpful to contrast this system with the ones that came before it. Was the previous system too complex, too slow, etc?*

Does the system scale? If so, how, and in what sense (number of users, amount of data, etc.)?

> *Many of the systems that we study in 6.033 are large systems that are designed to function well under a large number of users, a high workload, or over a large number of machines. We say that these system scale well.*
>
> *Information about whether a system scales can often be found in the part of a paper that describes a system's environment; you may see an explicit count of how many users or machines the system can handle, or a statement explaining that the system runs across the entire Internet.*
>
> *If you are reading a paper about a system that has been designed and implemented, but not yet deployed, pay close attention to the Evaluation section of the paper. If the system claims to scale to thousands of machines, but is only evaluated on a few hundred, its ability*

to scale may be questionable.

Finally, how much a system needs to scale is directly tied to its environment. For example, if a system is designed to be used in a datacenter network with thousands of machines, the system does not need to be able to scale past that.

What types of failures could occur? How are they handled, if at all?

A common failure is for one of the system's machines to stop working. Think about what happens in this case: if there is data on the machine, can the system recover it? Handling failures becomes more important as a system becomes large; with small systems, failures might be rare, and simply re-starting the system, or re-running a query, may be an appropriate fix.

Information about how a system handles failures often comes after an explanation of its basic use. It is usually very explicit, though you may want to ask yourself if the paper has truly described and handled all possible failures.

Is security or trustworthiness a concern? If so, how is it solved?

For instance, is it easy for users of a system to verify that the information they retrieve is correct?

Not all systems consider security to be a major concern. If a single entity is running a system, and also controls the machines on which the system runs (e.g., Google running MapReduce in its own datacenter), the designers may not prioritize security.

(Note: We do not cover security in-depth until the end of 6.033. You should think broadly about security concerns before then, but we do not expect you to discuss threat models or attacks in great detail until then.)

Critique the system

Think about our design principles from class (simplicity, modularity, etc.). If any of them don't hold for this system, is that a problem? Alternatively, were any of the design principles used to a fault?

Consider the design principle of simplicity. One can ask if a system is too simple, i.e., whether a small amount of additional complexity may have added important functionality that was left out in the name of simplicity?

Think about scale, failures, and security/trustworthiness. If any of them aren't addressed, why not?

Example: if a system does not scale, perhaps it doesn't need to. If a system isn't designed for security, perhaps there is a separate system in place to handle security.

At the same time, if a system is designed to run across the Internet, but does not scale past a few hundred machines, that will almost certainly be a problem.

Does the system have a trade-off between generality and performance? If so, have the designers made an appropriate trade-off?

> *Some systems aim to work in every environment, and end up providing good performance in none. Other systems provide good performance under a very specific set of assumptions, but users may find that those assumptions are unrealistic.*

Within the system's environment, under which use cases does it perform poorly? Are those use cases common?

> *For instance, consider the performance of a filesystem under writes to a file. The system might work very well when a user is appending data to a file, reasonably well when user seeks to a certain point in a file and inserts a large amount of data there, and poorly when a user's writes are completely random. In such a system, you should ask yourself of the last use case (random writes) is common.*
>
> *Some papers are explicit about the conditions under which they perform poorly. Others report the assumptions that they make on a workload (in which case you can ask yourself if those assumptions are reasonable).*

Is the system easy to change, or iterate upon?

> *Well-designed systems with modular designs are often easy to change; one can update a module without needing to re-design the entire system.*