What is Cycvar?
Cycvar is fundamentally just a series of variable filters strapped onto box loaders tied together by some sort of cyclical logic. To elaborate, the initial concept, as put forth by Datnerd, is a series of variable filters loading boxes. You input a set of boxes, and each var filter sets an item type from a box, and then looks for that item type in each successive box. By this method, all of each item set by a variable filter in a given pass will be removed from all boxes and consolidated into fewer output boxes.

Cycvar is one way of doing variable sorting, which organizes a mixed input into single item type boxes. It has the benefit of ideal output, which is the unique quality of every item type being organized into the fewest number of boxes, or one partial per type. Any boxes other than that partial will be full. While Cycvar, in its most basic form, accomplishes this and was aimed to do so, it does so at the cost of speed.

Example to Illustrate Speed Problems
Say there are four slices, and 4 boxes in the set to be unloaded. The first slice is assigned to stone, which the first box has 4 stacks of.

It unloads the 4 stacks, and then the box is sent to the next slice, which is assigned to oak planks, which the first box has 5 stacks of.
While that is happening, the second box is checked for stone, doesn't have any, and so it is broken, and queued in the second slice, waiting for the 5 stacks of oak planks to be fully unloaded.

The third box has no stone as well, so it doesn't unload in the first slice. The final box has a stack of stone, which is unloaded before the 5 stacks of oak planks are, and thus, now all four boxes are in the second slice, waiting for the oak planks to finish unloading.

The first box finishes unloading oak planks, moves to slice 3, and unloads its three copper bulbs into the slice, and quickly moves to the fourth slice, where it sets gray candles as the filters, of which it has a stack. Meanwhile, the second box is unloading oak planks, of which it has 10 stacks. Meanwhile, the other boxes are stuck, waiting for the oak planks to unload. The first box finishes and enters the buffer.

The Issues
The unloading speed is a clear issue. The longer the boxes spend in slices, the more they prevent the following boxes from moving forward.
Item types, even ones with slices assigned, cannot be accessed while their boxes are stuck waiting in the queue behind slower unloading operations.
Slices are inactive at the beginning and end of every cycle, as the first box sets the filters, and the last checks against all of them.

Solutions
There are four solutions to these speed concerns, as listed below.

Hyperthreading
(*Initially suggested by Data)*
Hyperthreading is a process aimed at greater utilization of the total number of slices. It achieves this by running ‘cycles’ of the system together. The problem it aims to solve is that every single cycle, there is a ramp-up and a ramp-down point.

As the first box is entering each slice for the first time, setting filters, the system gains access to more slices. As the last box leaves each slice, the system loses active slices and potential speed.

In order to solve this, Data proposed and implemented hyperthreading. As slices are assigned on the ramp down, the system begins sending the next cycle through the system. This means that you start ramping up the next cycle while the previous cycle is ramping down, effectively eliminating this access problem at the beginning and end.

Carts
(*Initially suggested by Monica)*
By using carts, you can unload the items 8x faster, which helps in two ways. Firstly, it just unloads items 8x faster, which is obviously a speed up. Less obviously, however, is that it chews through blockages more quickly, allowing boxes access to slices faster, meaning there will be shorter stalls where only one slice is active.

There is a trade-off, though (other than complexity). The unloading speed bump is accompanied by a significant testing period. What do I mean by that? Basically, if a box does not have the item matching a filter, how long does it take to determine that the box doesn't belong? It's effectively the minimum amount of time a box can spend in a slice, which is important because it determines how long that slice will be inactive for per non-matching box process.

In a cart cycvar, that testing period is significantly longer, because it is determined by a full cart cycle, instead of a hopper cycle. This becomes a massive weakness when dealing with extremely high variance item sets, where the filter set by a slice is only held in the initial box. This is the only situation where, theoretically, a cycvar could be faster than a cart cycvar, specifically when the quantity of each item is also very low, and it is purely due to the extreme testing period.

Groupers
(Initially suggested by Monica)
The grouper optimization is extremely powerful.

A grouper is a variable filter method that places the box on that variable filter, and if an item is sucked out of the box, it redirects it to the 'matched output' and recombines it with the sucked

item. The filter can be reset at any time, and when it's reset, the buffered box is recombined with the filter item.

In practice, groupers route boxes to appropriate slices based on what items they contain. Boxes only visit slices where they have matching items, bypassing all others. This allows boxes to skip over unloading boxes and map directly to the first slice with a matching item. Further, all boxes with no matching items can be skipped entirely and passed to the buffer. This effectively eliminates every form of waste present in traditional cycvar.

That is not to say grouper-based logic is perfect. The primary concern with groupers is protecting them from empty boxes and lead unstackables, both of which break an empty grouper. This adds a high cost to moving boxes around to and from a lead item filter and causes significant speed losses during the reset process. Further, groupers buffer a box to maintain their filter, which ties up a box that could otherwise be processing and reduces throughput during certain phases.

Sacrificing Ideal Output

(*Source unclear, but possibly Monica)*

Ideal output is a very taxing condition. It effectively demands that you must have every box check against every filter every cycle, and that results in a lot of otherwise inefficient actions. One thing that some cycvar projects have done to optimize for speed is by abandoning the initial goal of ideal output and simply trying to significantly reduce the number of merges required within a certain input, rather than entirely eliminate them. One such system is hidden squids, which abandons the typical cyclical structure of the overarching logic, and opts to feed groups of boxes with the same item types to variable un/loading slices. This theoretically allows for significantly greater slice utilization, as when a slice is done, regardless of the rest of the boxes in other slices at the time, the system can reset that slice and begin processing new boxes.

Finished Cycvars

Balancing these objectives becomes a logic and trade-off problem. As a testament to this, every cycvar seen to date has had different logic. Different builders have prioritized different optimizations, resulting in distinct implementations:

Datnerd - Original Cycvar

Traditional cyclical logic, with ideal output and hopper speed filtering. Extremely slow, but much simpler than all the following iterations.

Datnerd - Hyperthreaded Cycvar

Utilizes hyperthreading, or the chaining together of cycles to increase the slice utilization of hopper speed slices. Still slow, but faster than its predecessor, at the cost of increased complexity.

Skyzy - Cart Cycvar

Uses traditional logic, but slices are cart-based, making the filtering significantly faster. Logic is simple, and while it uses carts, it is optimized for size and simplicity. Uses an adaptation of camphorwood's cycvar slice. Ideal output.

TisUnfortunate - Grouper-Based Cart Cycvar
Uses grouper per slice logic, while maintaining ideal output, along with cart-based filters, which can unload at high speeds. Logically complex to accommodate the optimizations, and is very large as a result.

Hidden Squid - Grouper-Based Cart Cycvar with Individual Slice Reset
Uses a grouper in the front end to feed slices of boxes, and unloads them quickly using carts. Does not maintain ideal output as a speed optimization. It is large and logically complex to accommodate optimizations aimed at speed.