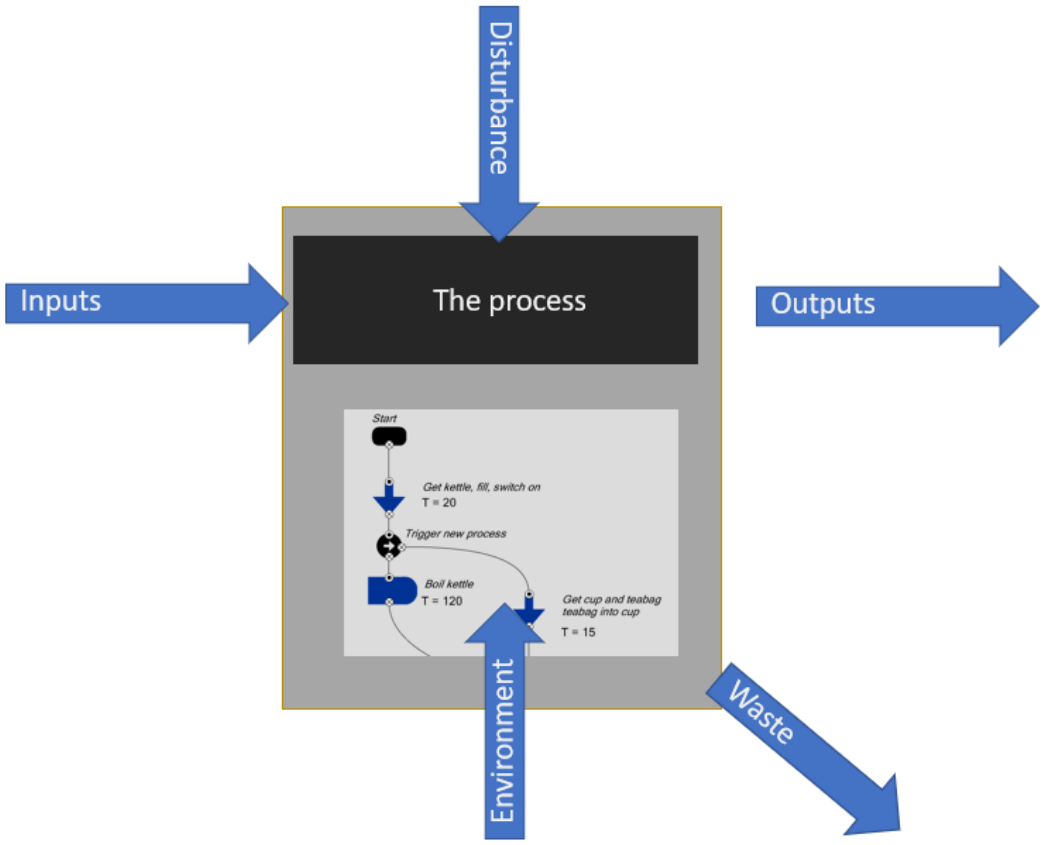
Chapter 8– Line balancing

We should, by now, understand the concepts of process mapping and production capacity. With these concepts clearly in place we can now start to look at the configuration of manufacturing systems.

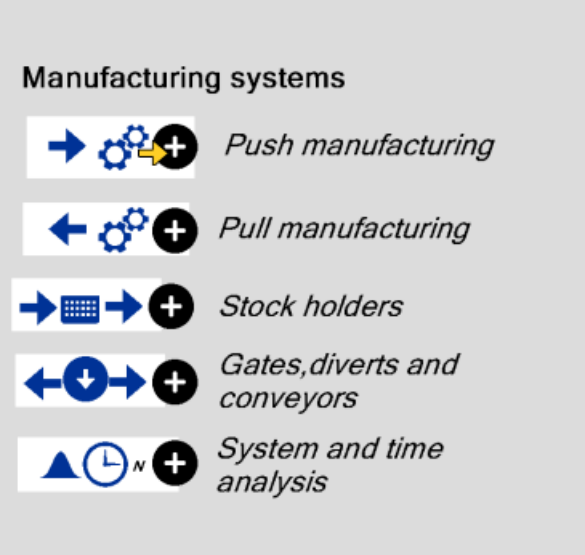
What is a manufacturing system? Well you can look at in a number of ways..

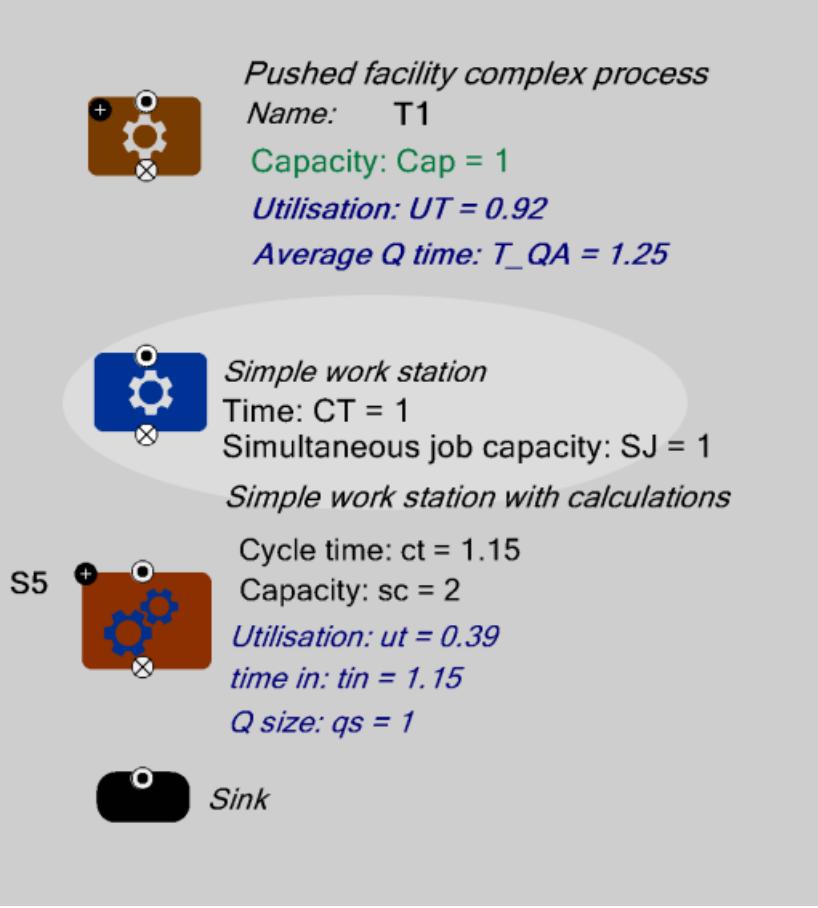
From one view it is a group of machines or other production facilities which combine to together to from an overall system with inputs and outputs as discussed in chapter 6.

But this system



One of the key ideas here is that a system contains systems – there is a hierarchy. So far we have built an thing that has a finite processing capacity, has inputs and outputs. And inherently runs a process. The most simple thing in Aliquis which represents this is the workstation block :





In this chapter we examine the interaction of these blocks with streams of work entering the system.

Creating the stream of work.

Actually this is a complex subject. In the real world a manufacturing system may have to deal with complicating factors such as erratic/random demand and variations on product configuration.

Demand variation can be due to time based factors such as time of day, day of week or seasonal. So, for example, if you are baking bread, you will have a higher demand early in the morning than just after lunch. You may see higher demand on Saturdays, and you might also see a lower sales volume in the summer than in the winter.

On top of this there is usually a random overlay. You may sell more bread on one Saturday in June than on the next.

Varying product configurations add an extra layer of complexity. Then the exact nature of what you are making is likely to change from one production event to the next. Ford motor company has long since stopped saying "You can have any colour as long as it's black". Our baker may make various types of bread in a batch-based system.

Push vs Pull

One approach is to manufacture only in response to real customer demands – in a pure pull manufacturing system, the random arrival of a customer spawns all the activities needed to satisfy the customers demand .

Another approach is to keep the machines busy with high utilisation – possibly making for stock rather than real customers. The real sales volumes may be forecast in advance and the production system is set up to produce to the forecast – not the actual demand on any one day. Variations between the forecast and actual either become excess stock or unsatisfied customers. But the machines are running efficiently!

Most real manufacturing systems are somewhere in between. Take for example a system which some people would characterise as a pull system. An ice cream cone van. The customer comes and the seller produces an ice-cream there and then for the customer. It sounds like pull – but if we dig a little the seller has a stock of cones and ice-cream which is manufactured in advance of the customer arriving.

The Kanban system , commonly used in industry, is actually a mixture too.

We will look more fully at pull vs push manufacturing and Kanban based production systems in chapter XXX

In this chapter, we will keep the demand very simple – constant rate manufacture in a push environment. For many mass production environments, this is the perceived reality that the manufacturing engineer has to work with. "Design me a production facility capable of producing 50000 cars a year!" The assumption is constant rate production – this is translated to 210 cars per shift if we assume a single shift system.

Before we start, let's define some key terms :

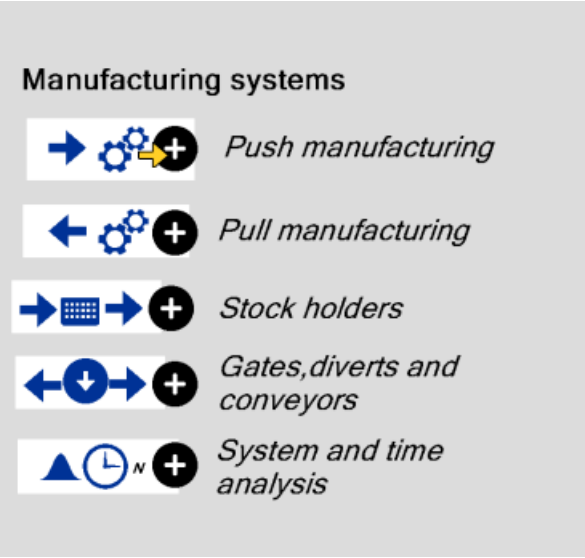
Takt time – this is the constant time between arrivals into the system.

Cycle time – this is the time each production unit takes to finish its work. It is related to the process which is carried out by the production unit. (you might think of the term production unit as a machine but, in reality, it could also be a person at an assembly bench etc)

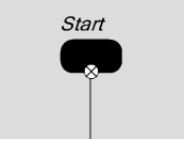
Let's make a start!

Setting up a constant rate demand :

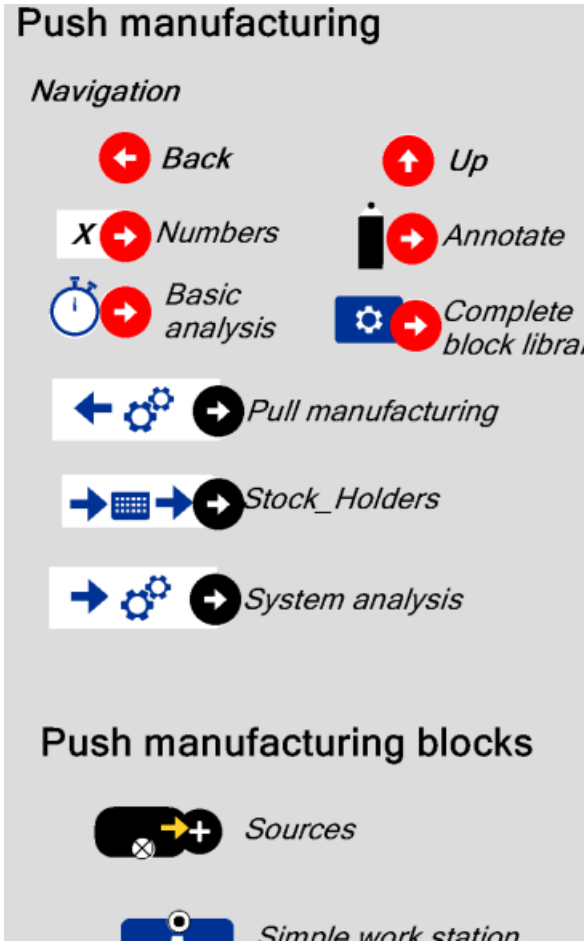
Go into the push manufacturing library :

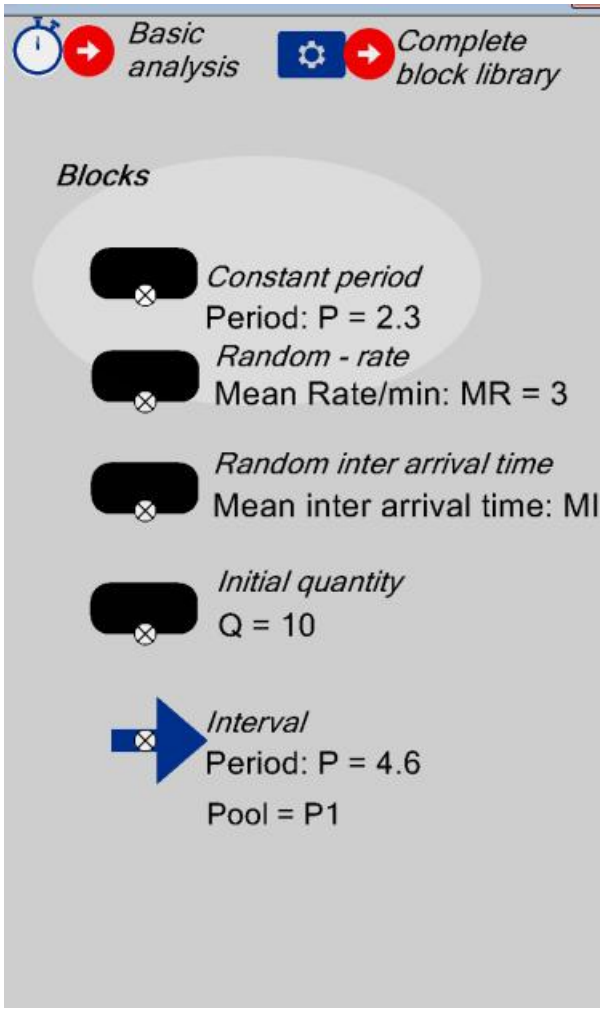


In your model, replace this :



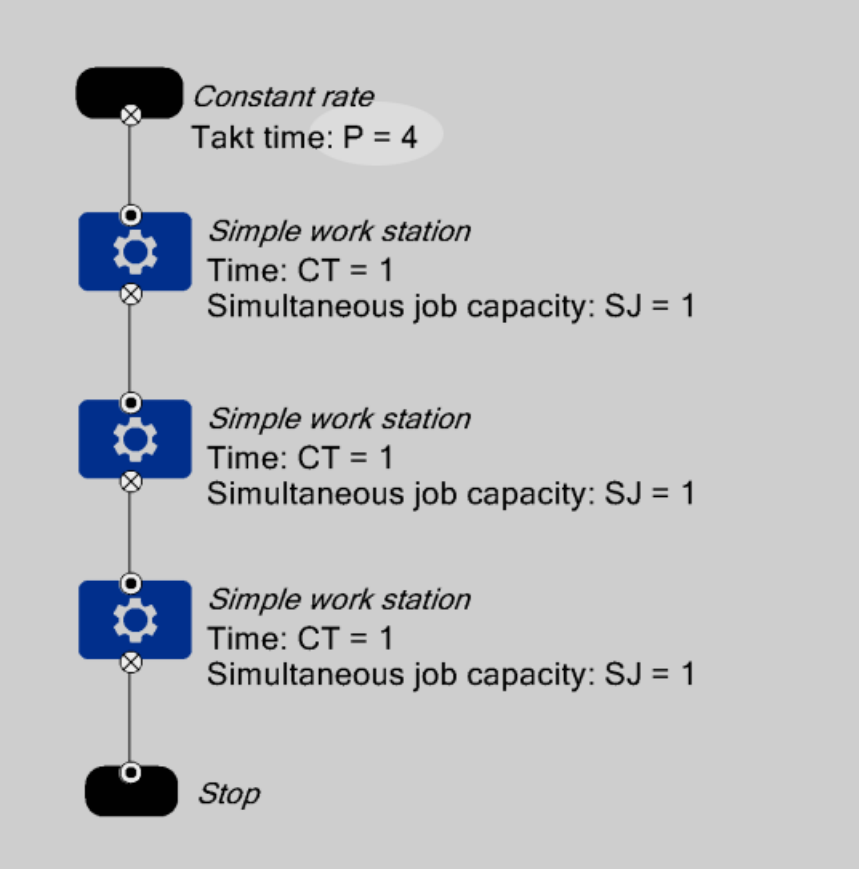
With this :


Push manufacturing
Navigation
Back
Up
Numbers
Annotate
Basic analysis
Complete block libra
Pull manufacturing
Stock_Holders
System analysis
Push manufacturing blocks
Sources
Simple work station

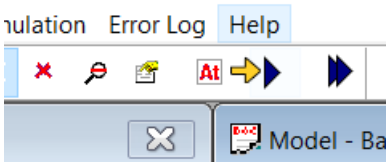


Now drop in 3 simple workstations in series :

Set the takt time to be 4 seconds :

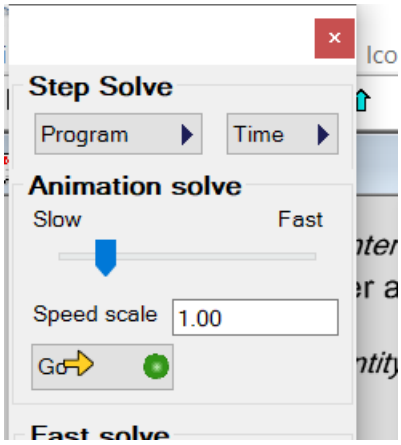


Lets step through in time steps :



Later we will calculate the utilisation but for now, we can visualise it.

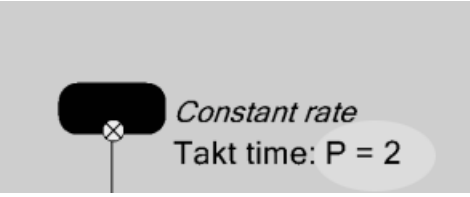
Animate the solve at an animation scale of 1 to 1:



Focus your eye on just one block. You will see how little it is used!

Clearly we can increase the production rate of this system :

Set the takt time to 2 :

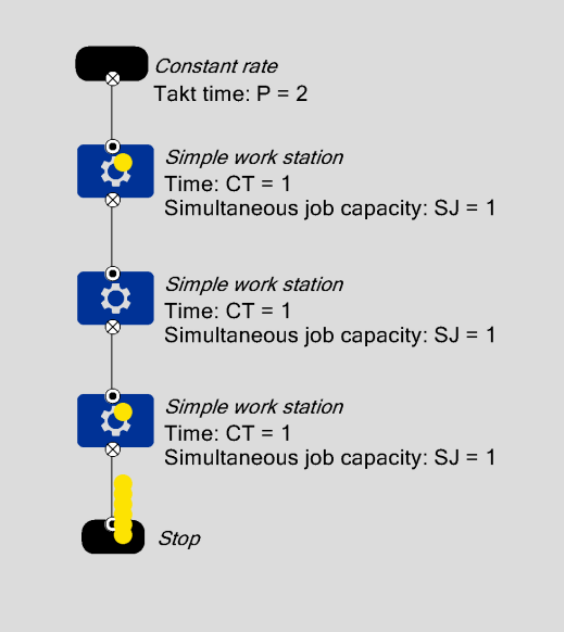


Animate again :

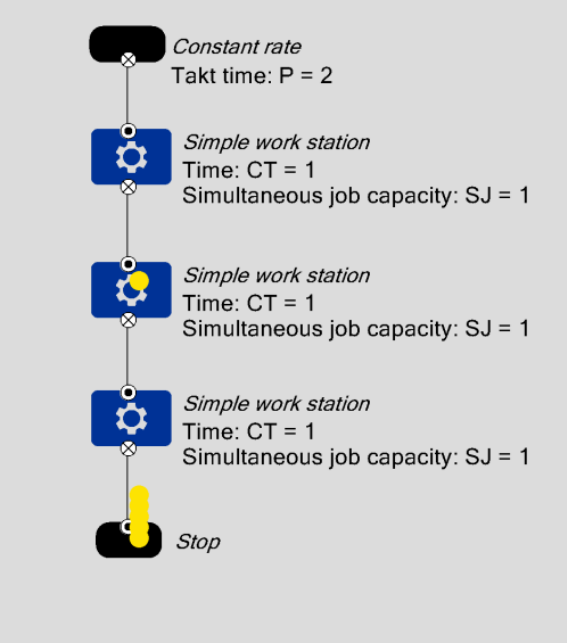
IF you focus on any one machine – you will see its utilisation is 50 percent.

If you focus on the system as a whole

Sometimes two tasks are being done simultaneously

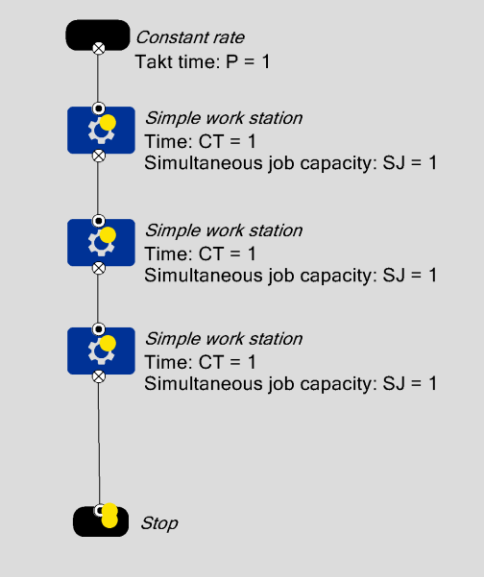


– and sometimes one :



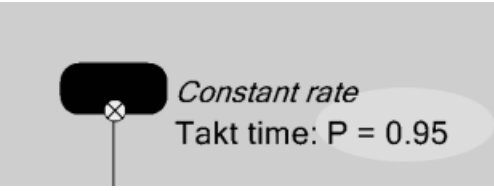
But the system is coping fine.

Now set the takt time to 1 and repeat the animation :

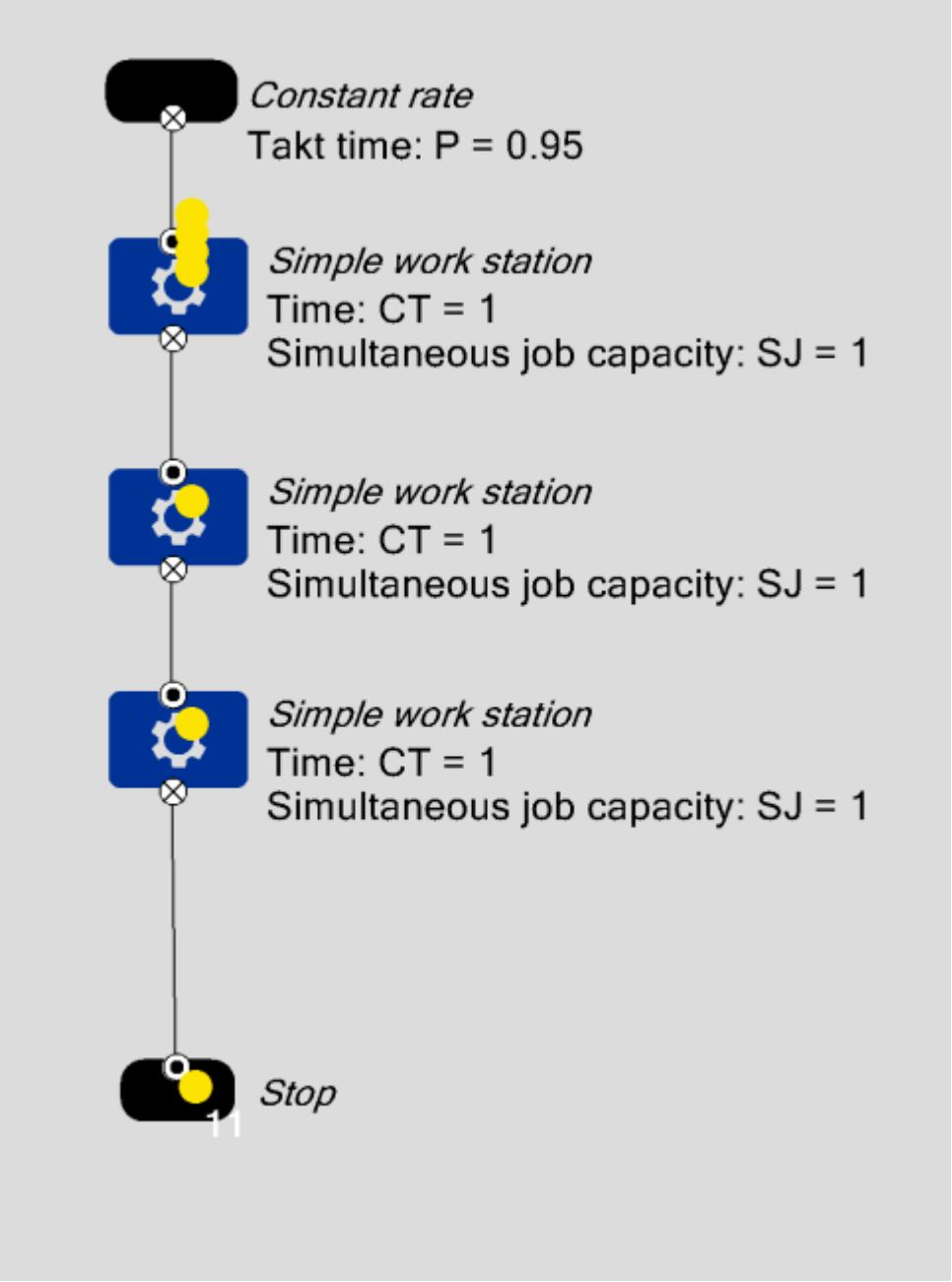


You will see that the machines are fully utilised. An ideal situation in many ways.

But what happens if we reduce the takt time any further? Lets try 0.95 :

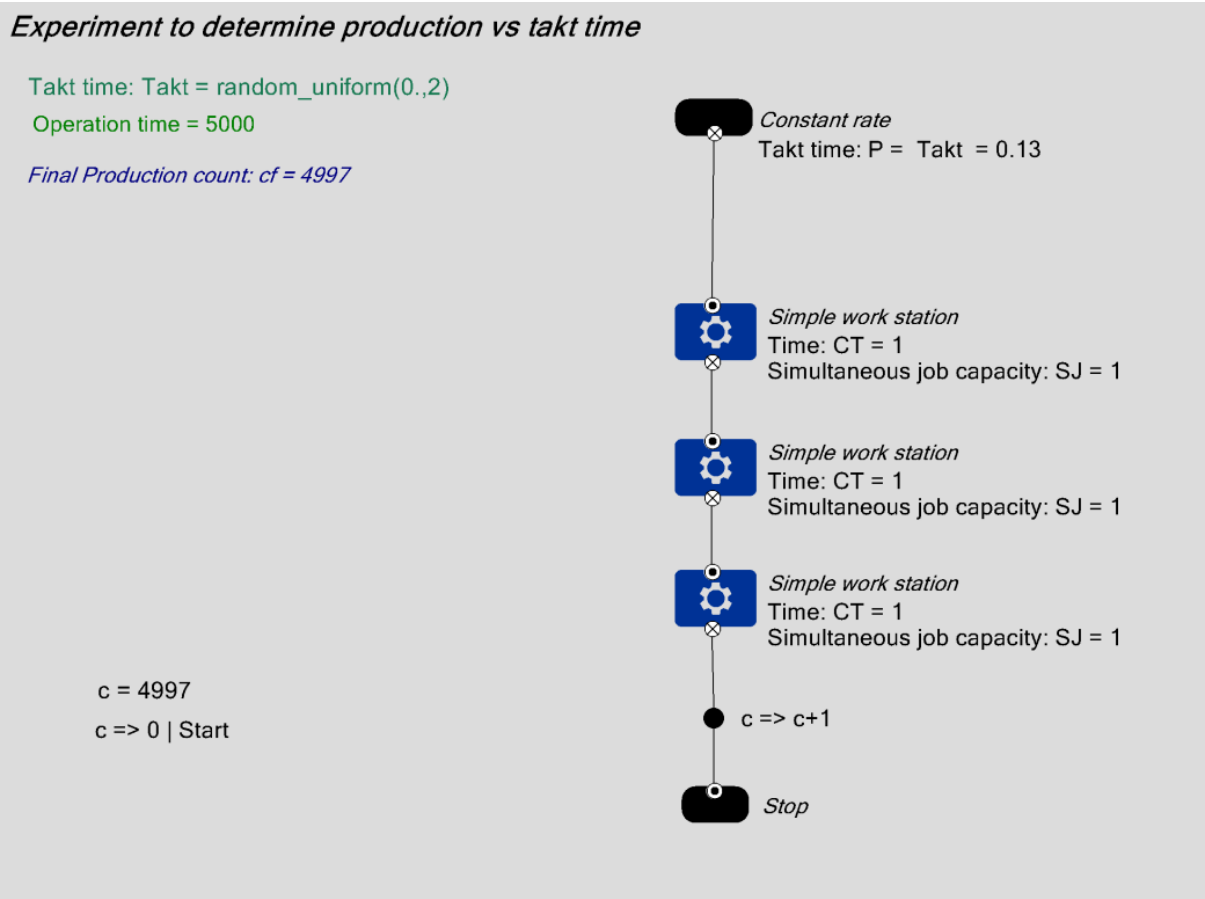


As the animation continues you will see a queue building up on the first workstation :



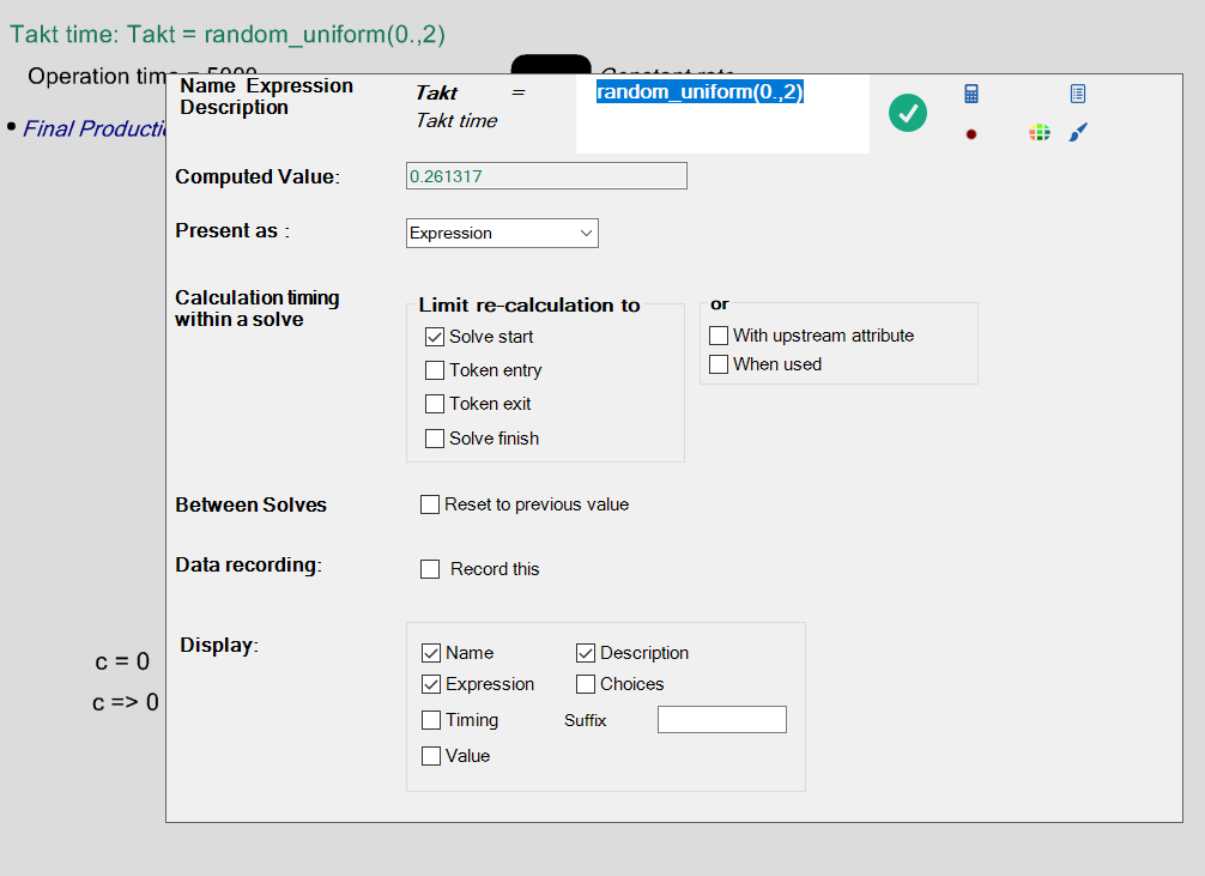
This is called a *bottleneck* it is the point in the production line which limits the production rate.

Open up Lesson8Ex1.



This experiment varies the takt time between 0.5 and 2 at the start of every solve:

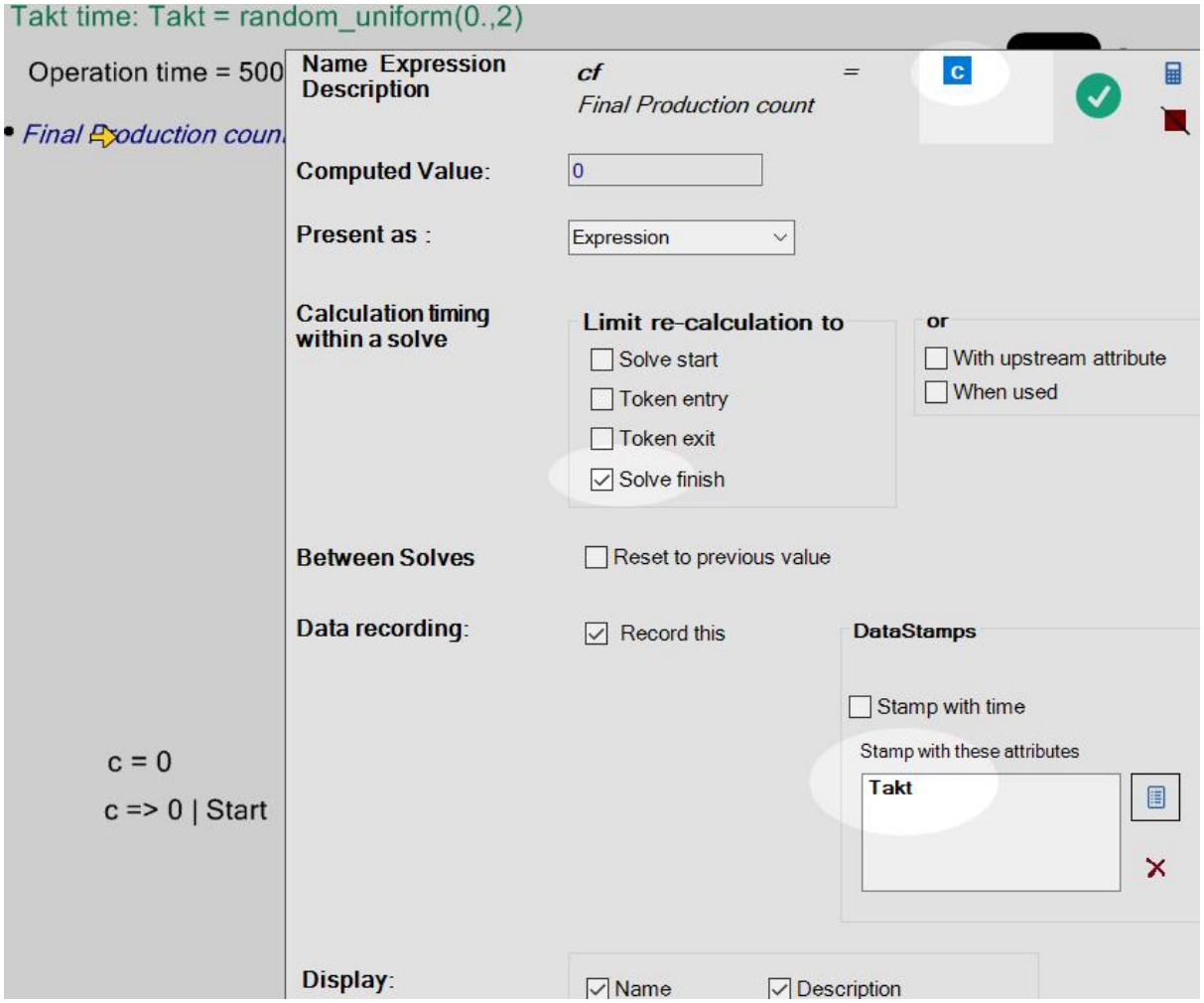
Let's take a look at the takt time parameter:



You can see here that it is set by the formula : *random_uniform(0,2)*

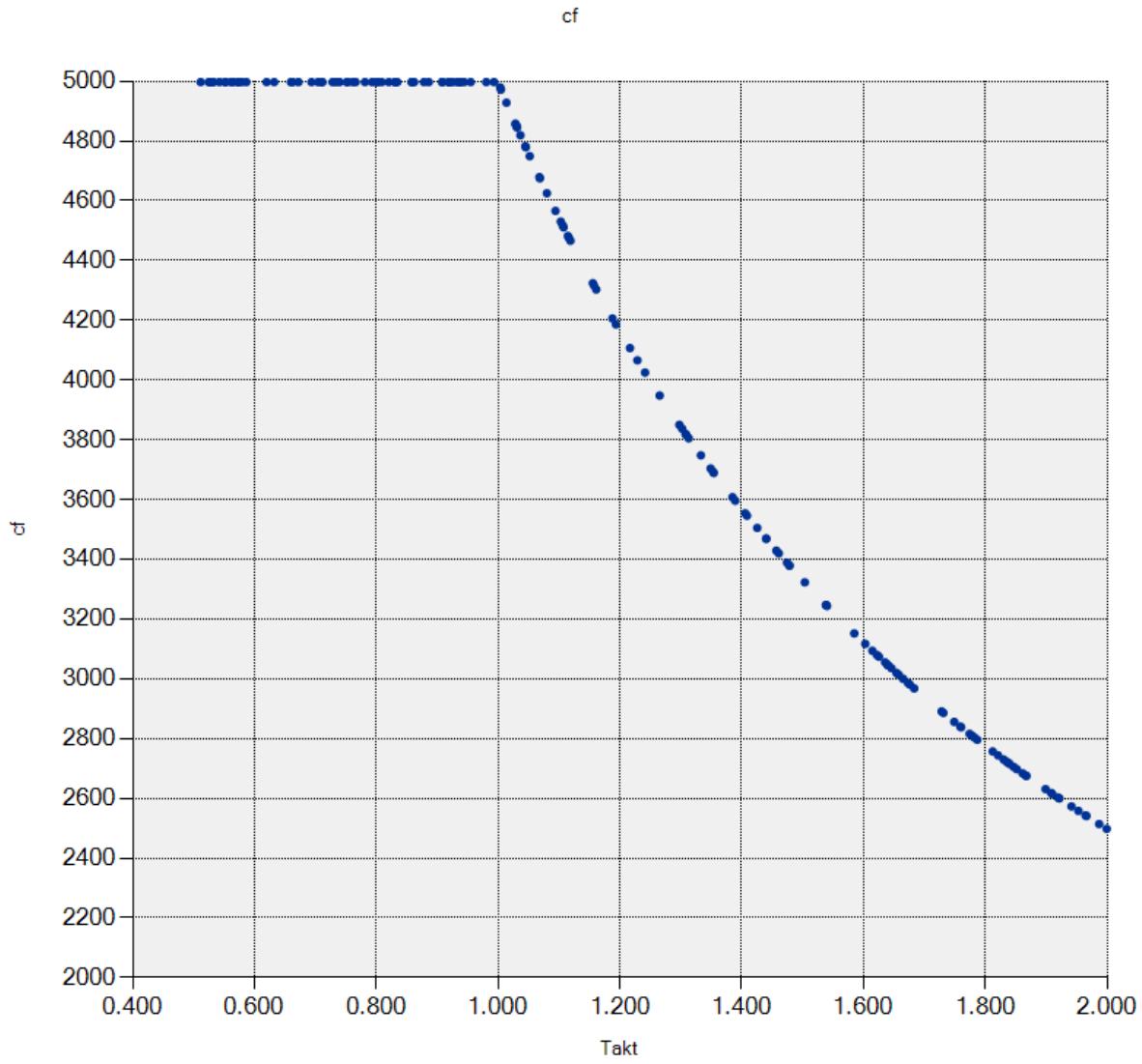
This means, that everytime the value is set, it will produce a random number between 0 and 2. The next question: When is it set? "Solve start" is ticked. This parameter is onloy set evey time a solve is started. In other words, it is constand during each solve.

Let's also have a close look at the user result : *Final production count*



First it is set to the value of c (the running count). Second, it is only set when the solve finishes and thirdly, when it is saved, the value of the takt time used for that solve is appended.

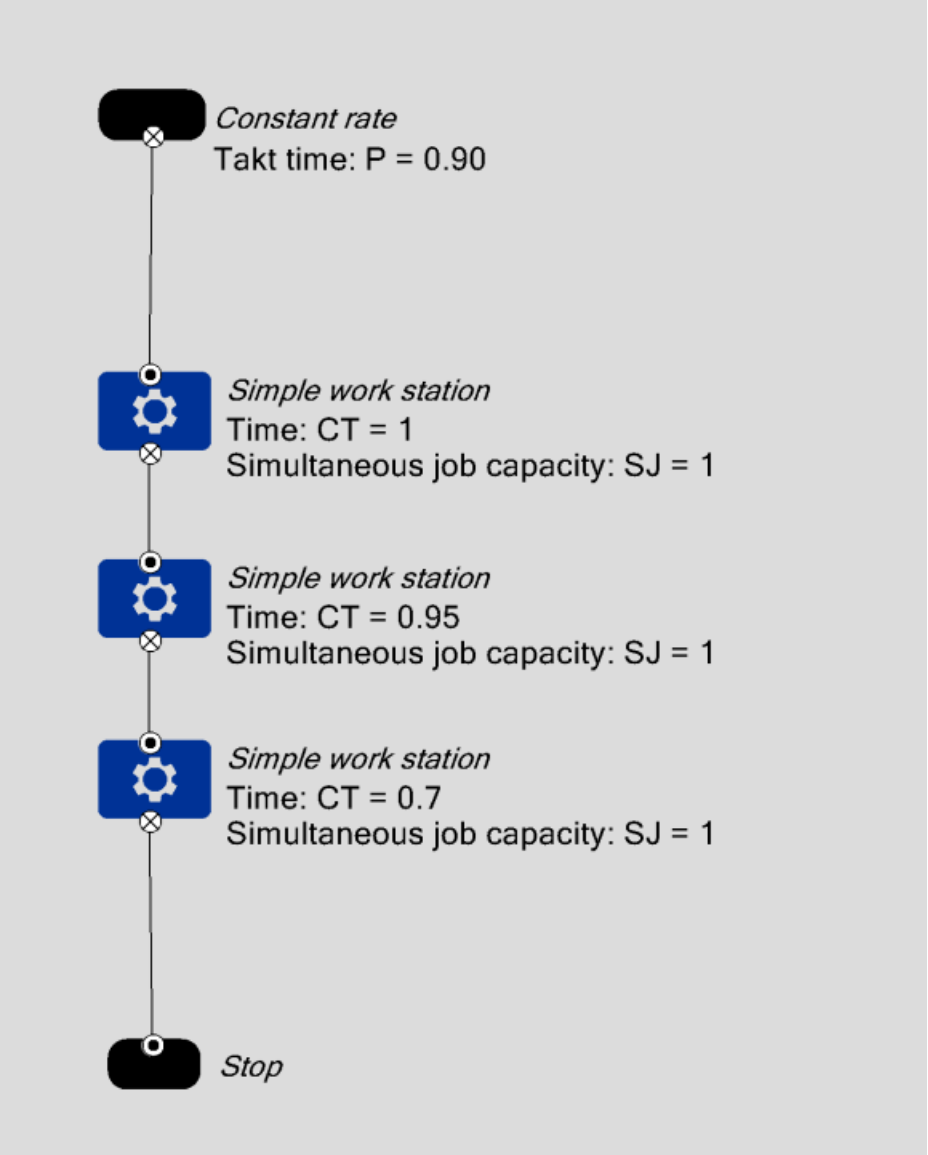
If you solve it 150 times and plot a scatter graph for the final production count against takt time you get this :



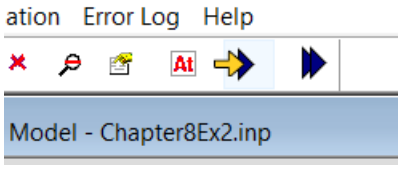
On the right you can see that it is possible to increase production by decreasing takt time. However, once you get below 1 second, you are not going to increase production whatever you do – but you will increase queues!

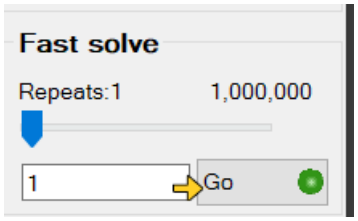
Fixing bottlenecks

Go to *Lesson8Ex2*

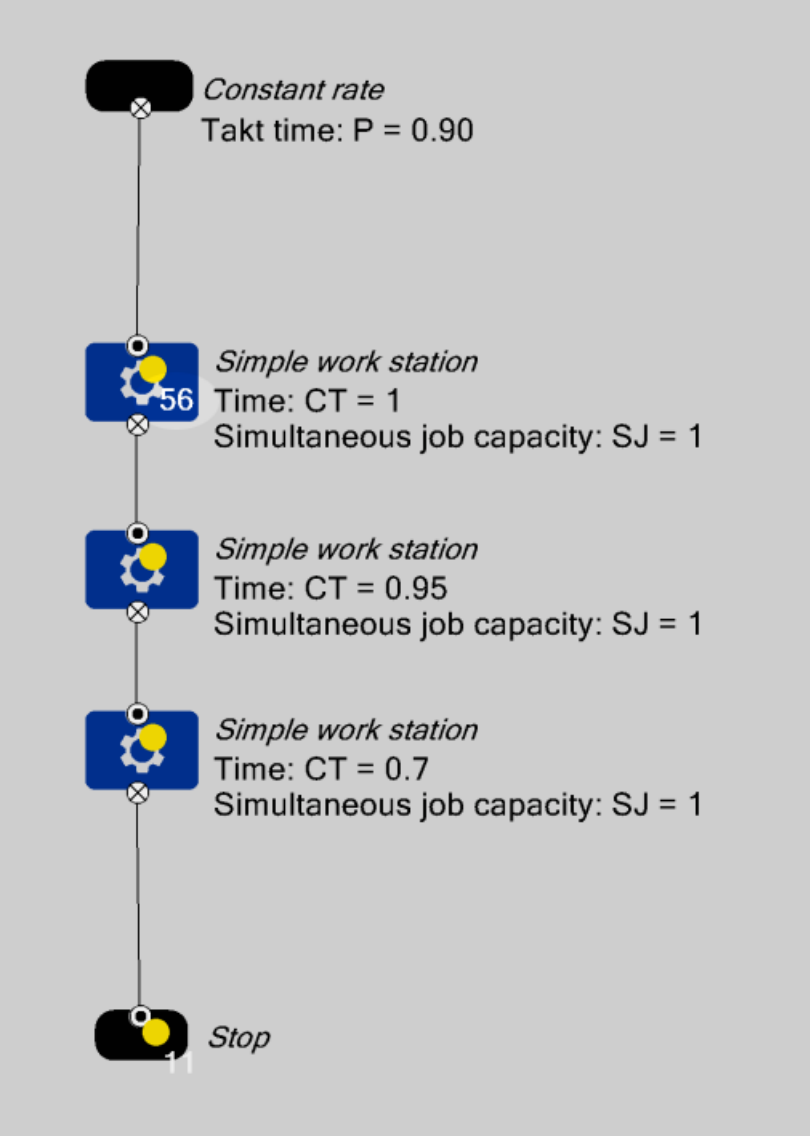


Do a quick solve:





You will see that there is a bottleneck at the first machine. We have 56 tokens waiting in the queue :



:

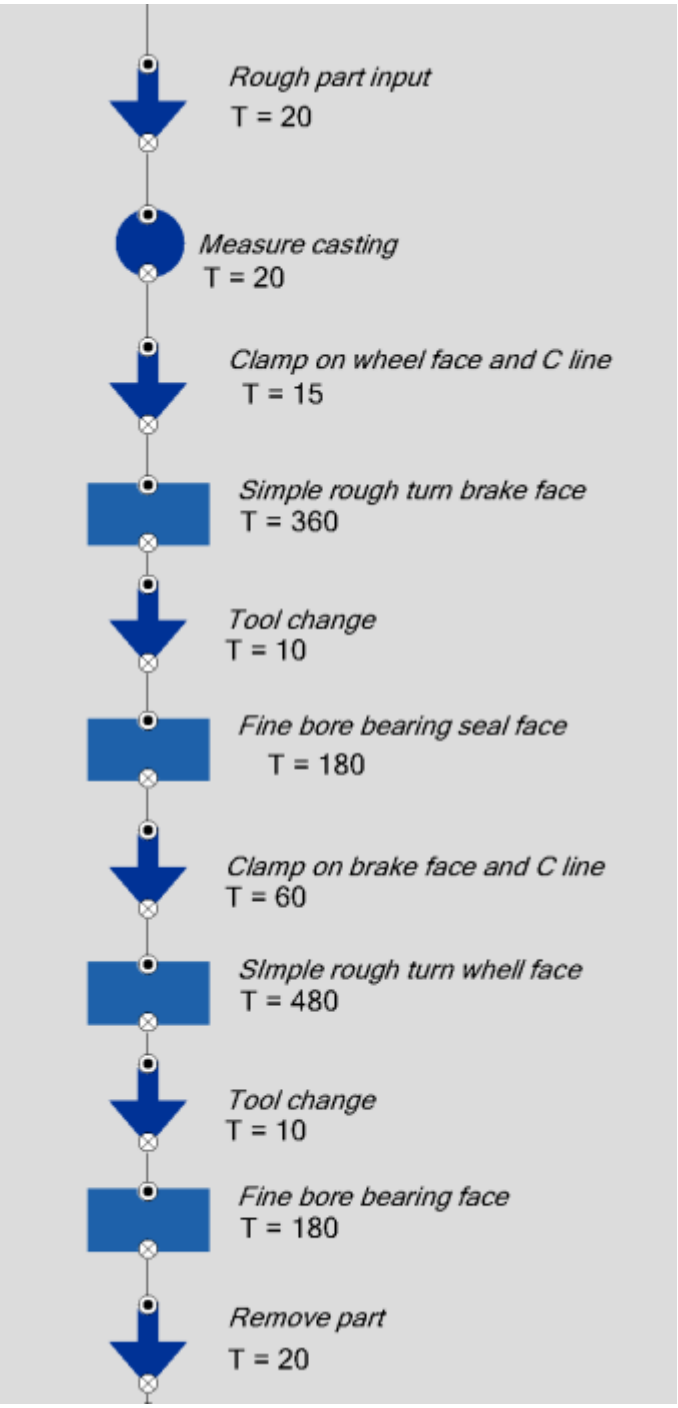
So how do you fix a bottleneck?

Well there are three basic approaches – all of which cost money in the real world.

1) Reduce the takt time.
2) Improve the speed of the machine – reducing its cycle time
3) Add additional production capacity and sharing the load.
4) Reduce stoppages (dealt with in chapter 10)

Assuming the takt time is actually based on customer based production requirements this is generally not such a good idea. You are going to reduce production per shift and lose the value of that production. However, if your demand turns out to be lower than expected, this might be a way to go.

Improving the speed of a machine – reducing the cycle time – is frequently possible. We can take here's an example of a lathe process :



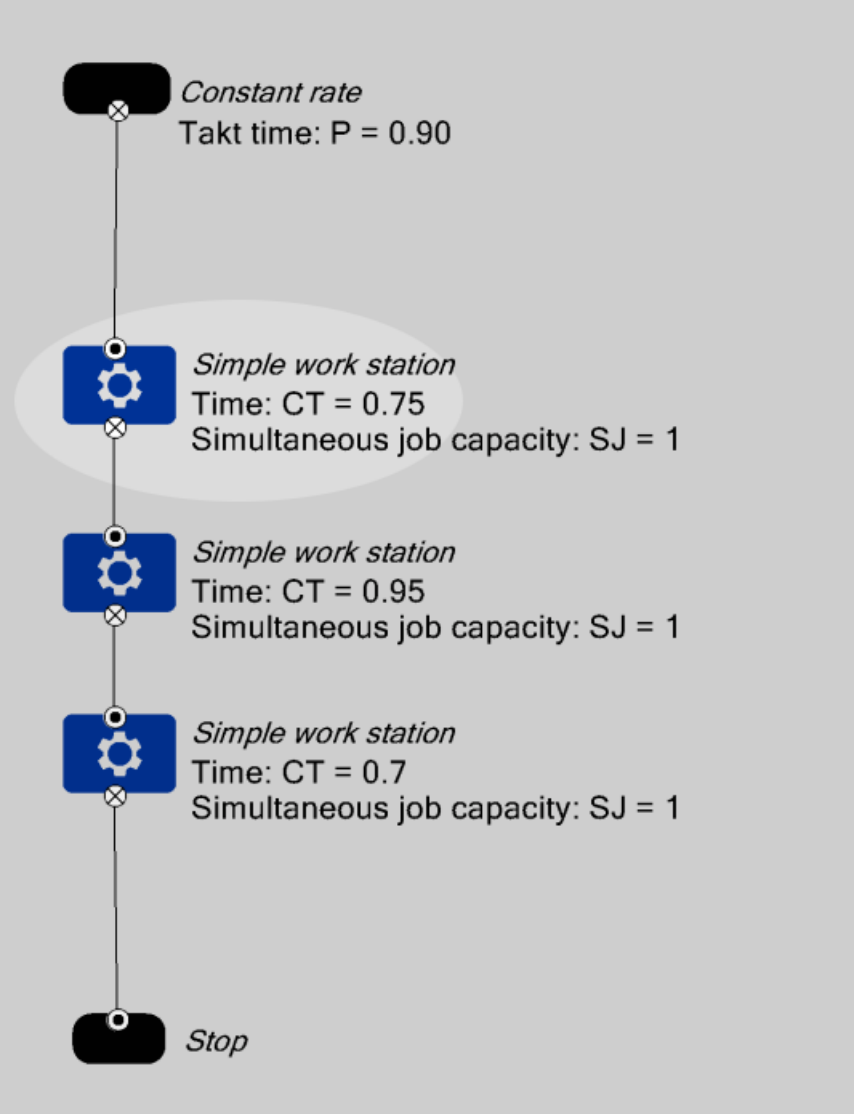
In this case we could ask ourselves :

- Can we automate tool changing and make it faster?
- Can we invest in higher quality tooling to make the cutting faster?
- Can we automate clamping and locating eg with a robot?

If any of these are possible, we can reduce the cycle time of the machine to clear the bottle neck.

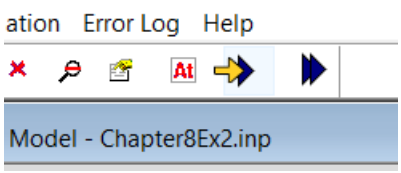
Finally we could add production - in this case add another machine and arrange to share it.

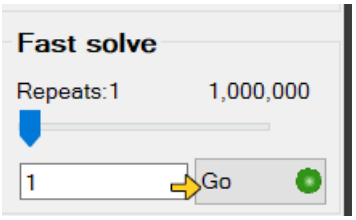
Lets start by reducing the cycle time of the first machine. Let's assume we can get the cycle time down to 0.75 as follows :



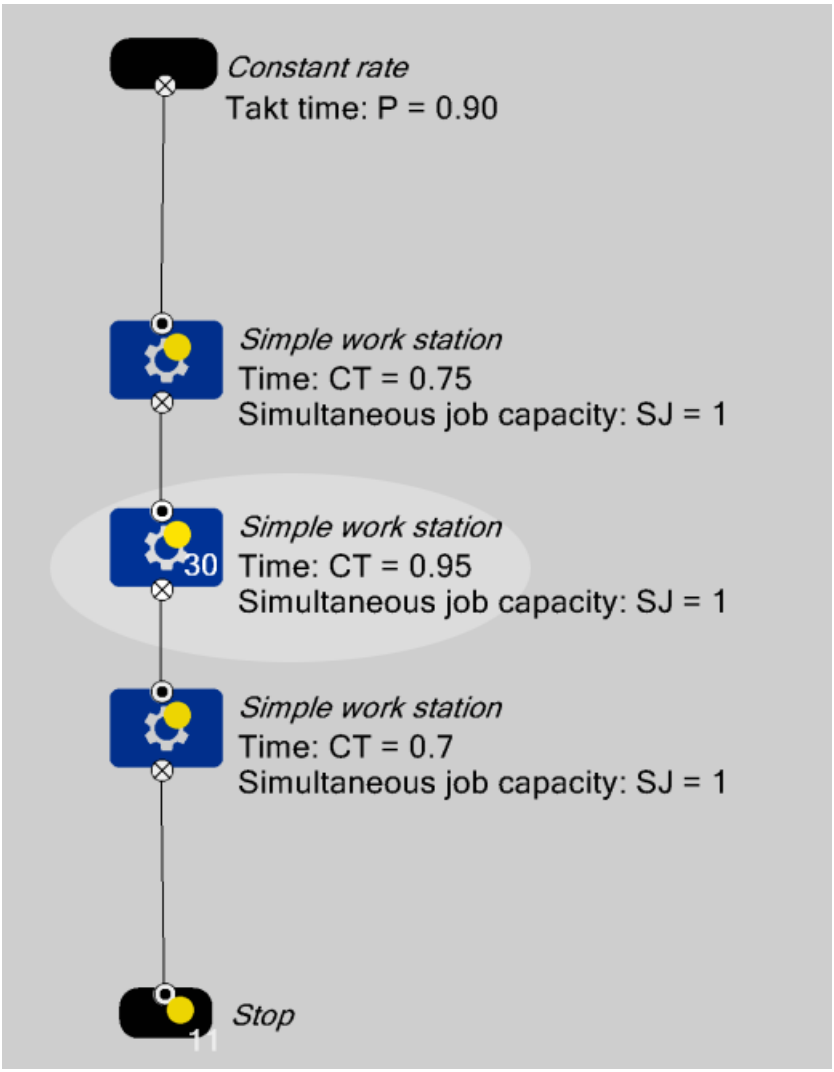
This is very easy to do in Aliquis but probably not so easy in real life!. Imagine you decide that you are going to improve the machine. Your company spends 3 * your annual salary on the improvements and you hit launch day!

Do another quick solve on the model :





Result :



Unfortunately, all you have succeeded in doing is shift the bottle neck to the next machine. In the previous solve, this potential bottleneck did not show itself because it was “shielded” by the actual bottleneck in the first machine.

The important lesson to learn from this is that if you are hunting bottlenecks – you need to find them all to achieve on-takt production levels.

In a simple serial production where:

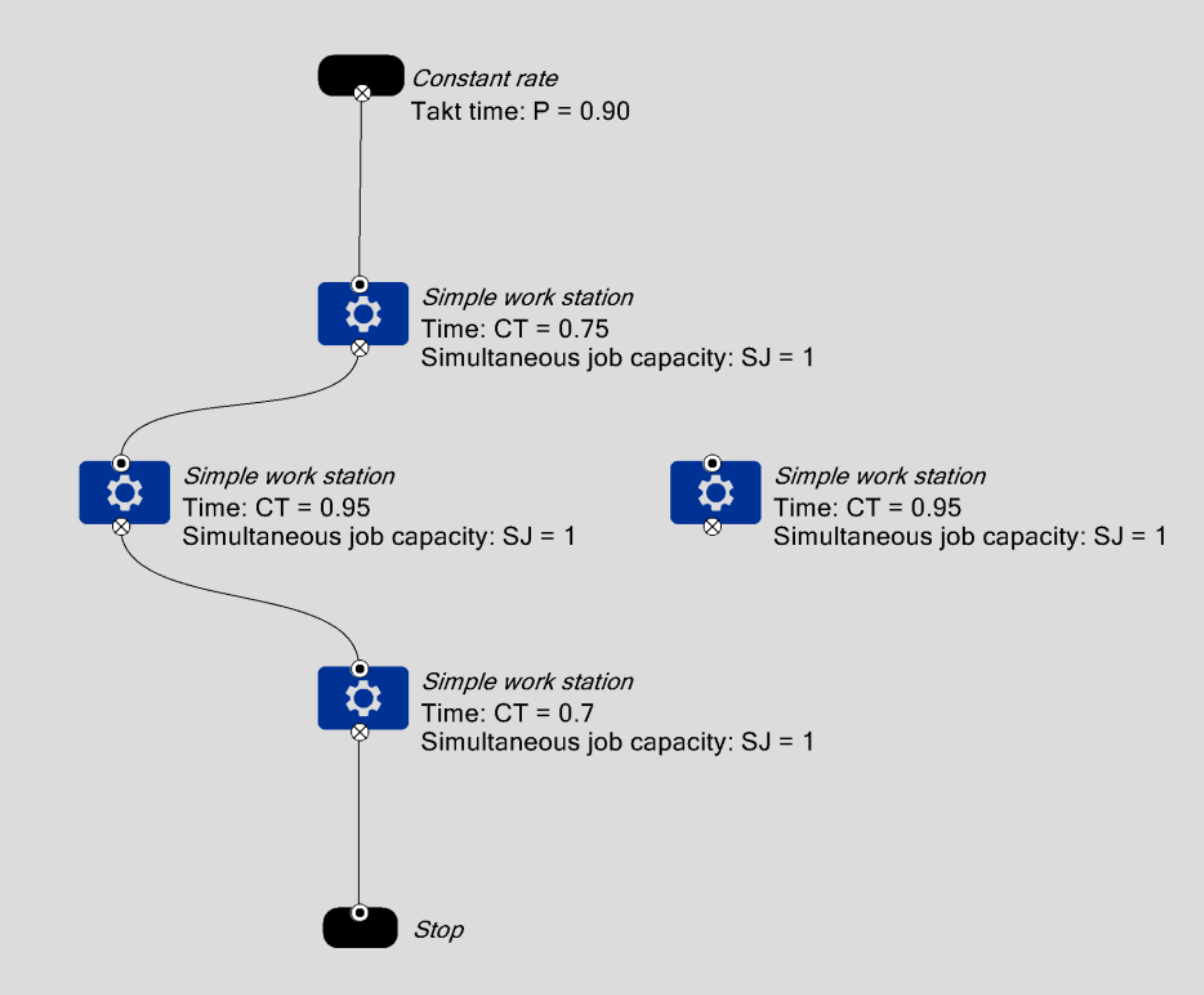
- There is no scrap
- the simultaneous production capacity is one

- the machines are arranged serially

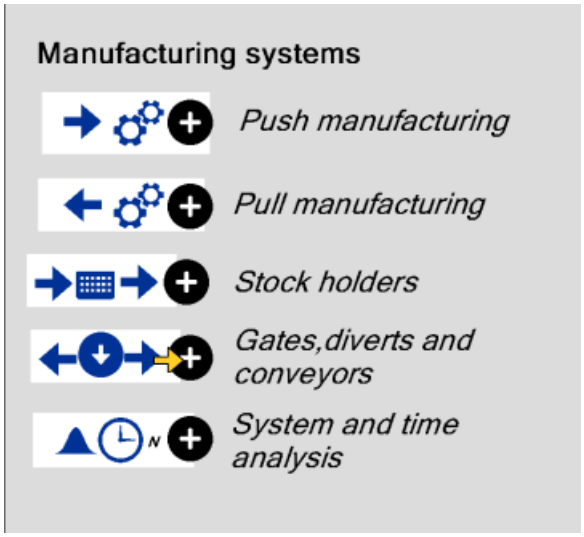
you will not achieve on takt production unless all the machines have a cycle time equal or lower than the takt time.

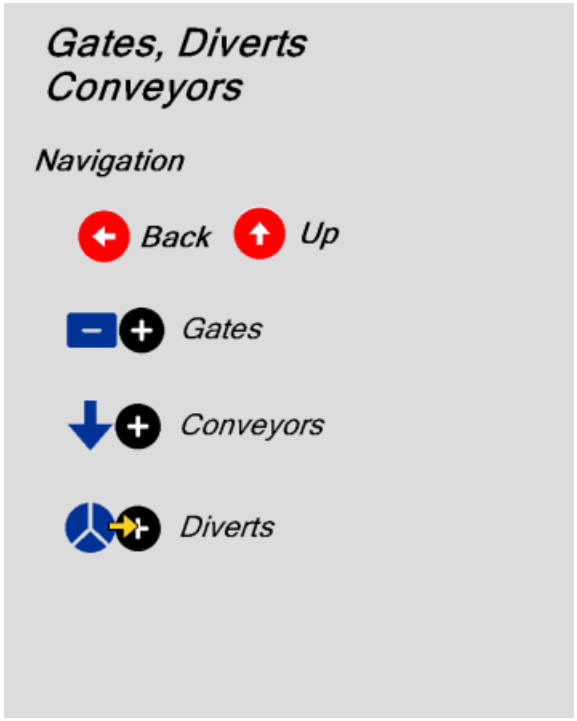
Lets continue with our quest to remove bottlenecks. This time we will add some capacity in parallel to the second machine:

Using the scratch area, copy the second machine and rearrange the model to get this :

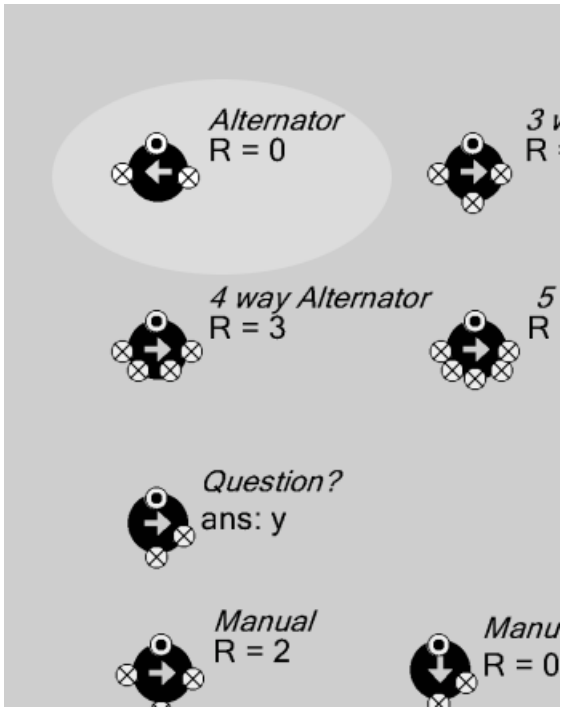


Go into “Gates, diverts and conveyors” the “diverts”

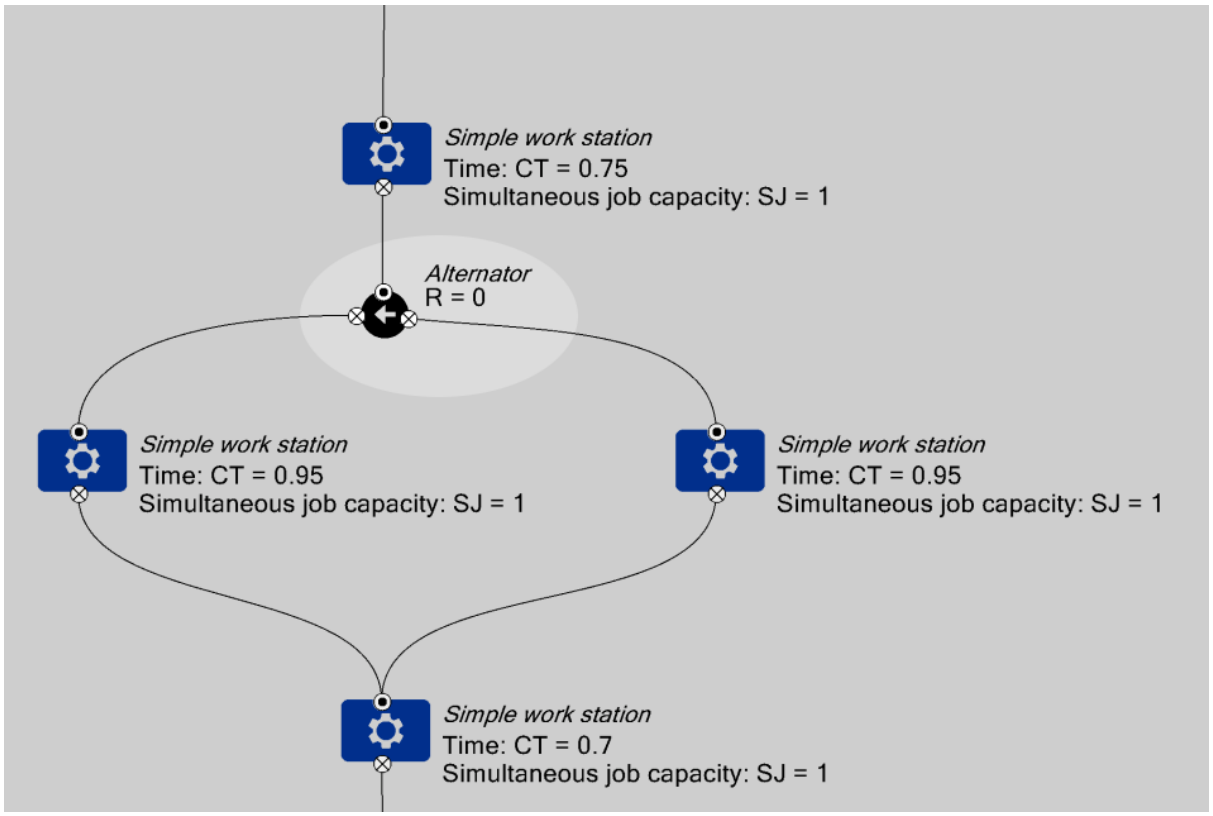




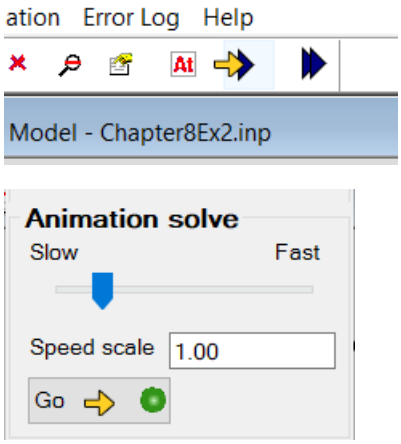
Lets start with a simple alternator :



Place it here in your model (and connect as shown):



Animate the solve:

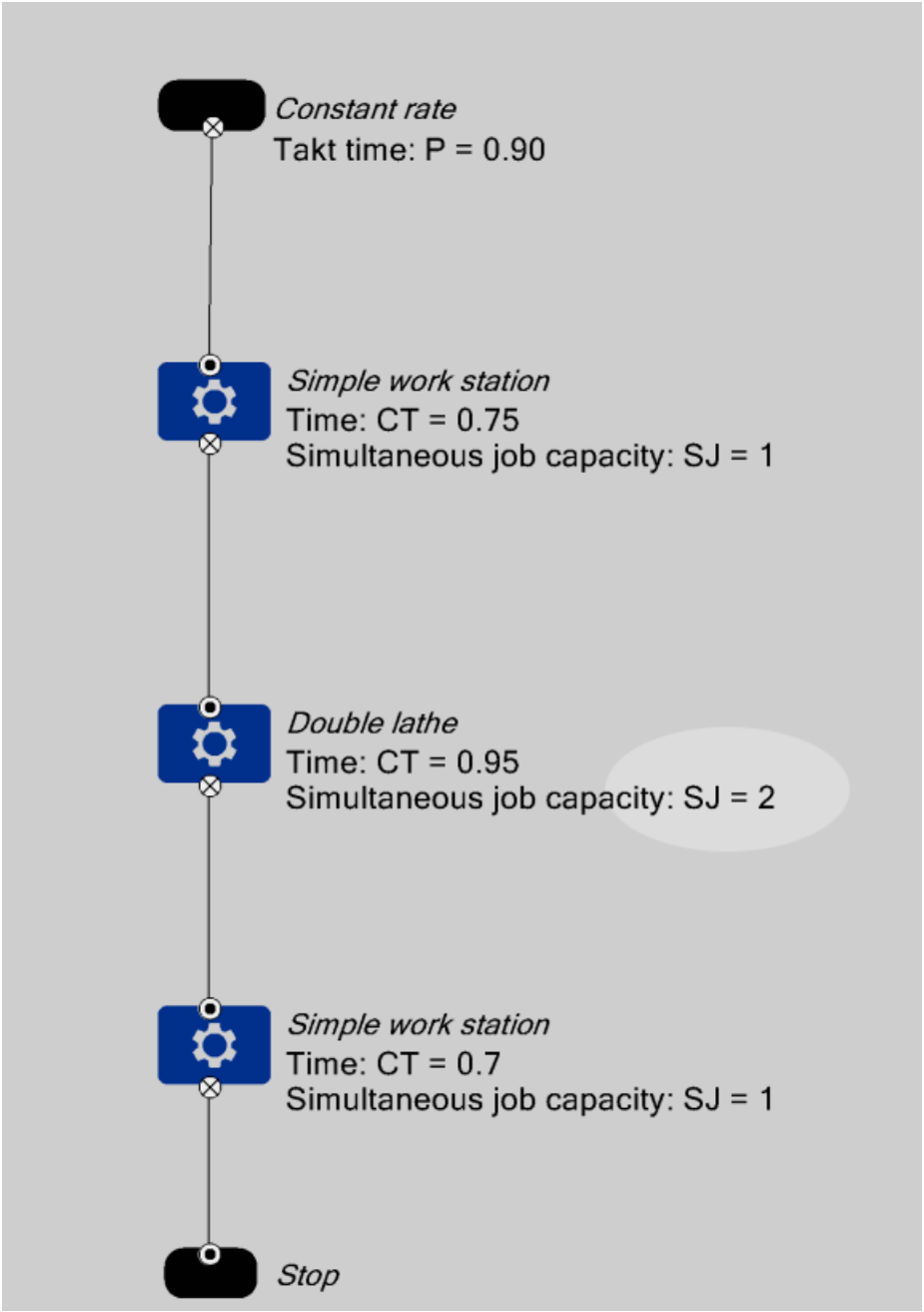


You will see that all the bottlenecks have now been eliminated.

If you need a reference model at this point, go to *Chapter8Ex2Done*.

Adding capacity the easy way :

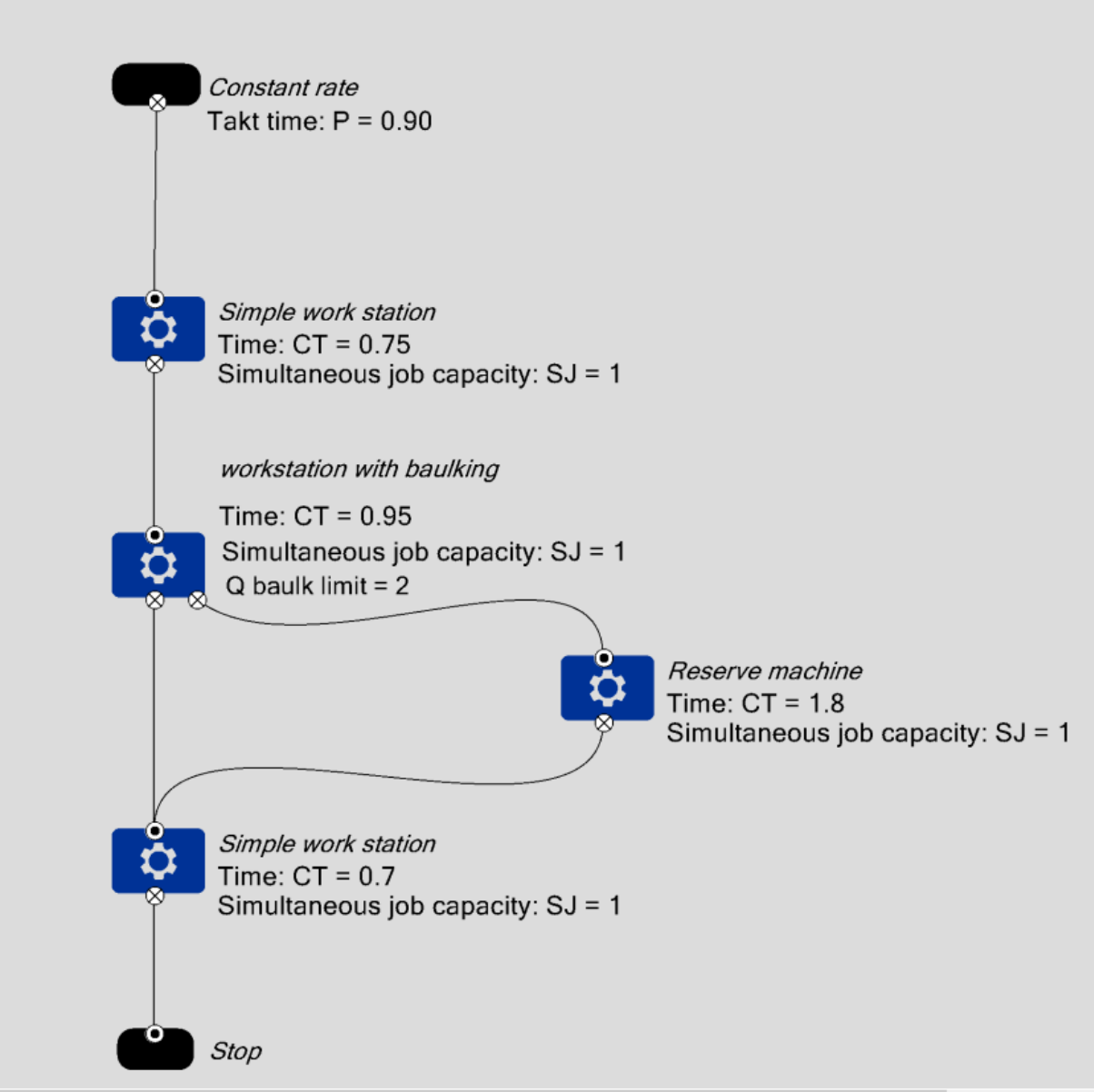
If you can imagine a lathe with two spindles or an oven which cooks 60 cakes at a time you have the capacity to produce more than one product at any one time. This is what the system attribute “Simultaneous job capacity” actually means. We will get the same results but not the same visualisation if we do this :



Adding reserve capacity

In manufacturing industry, it is common that the added machine is of lower spec than the main machine. It is only brought on stream when the main machine is not coping with demand.

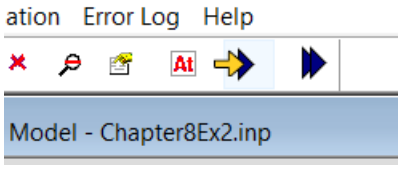
Take a look at *Chapter8Example1:*

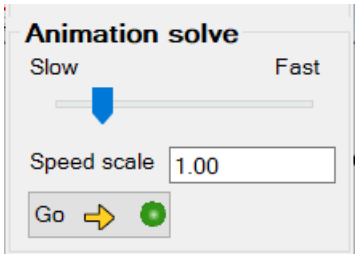


The second machine has been replaced with a baulking workstation – this can also be found in the push manufacturing sub-library

Notice the reserve machine has a quite high cycle time. But it still adds enough capacity to prevent a bottleneck.

Animate the system to understand how it works :





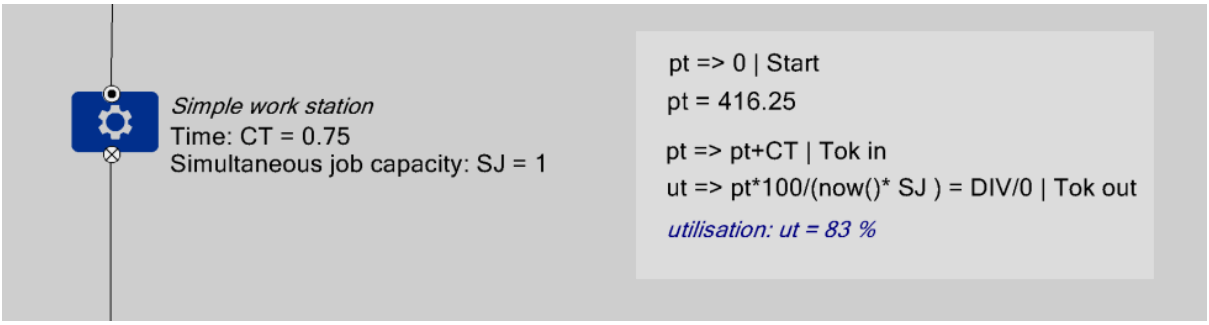
Calculating utilisation

Utilisation is :

$$\frac{Cumulative\ Actual\ production\ time}{Potential\ production\ time}$$

In a simple case like this, where the cycle time is a constant and we are not looking at times when the whole factory is shut down, the actual production time is the number of tokens that have gone through multiplied the cycle time. The potential production time is the time since the simulation started times the simultaneous capacity.

We can create a block which calculates its own utilisation. Check out *Chapter8Example2* :



This block includes two variables :

pt is production time. *ut* is utilisation.

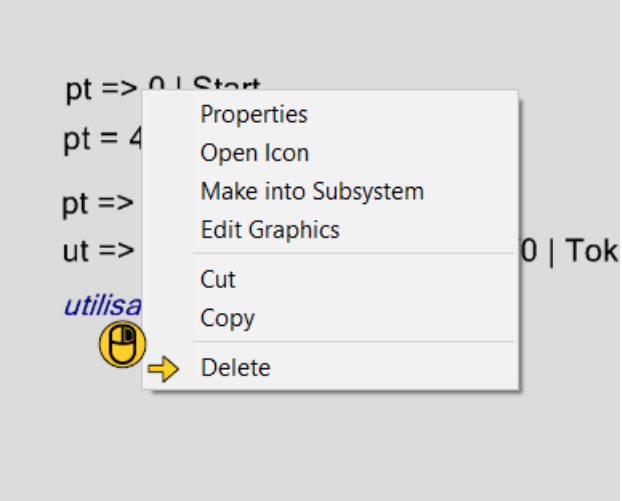
The first assignment *pt =>0 | start* initialises *pt.*

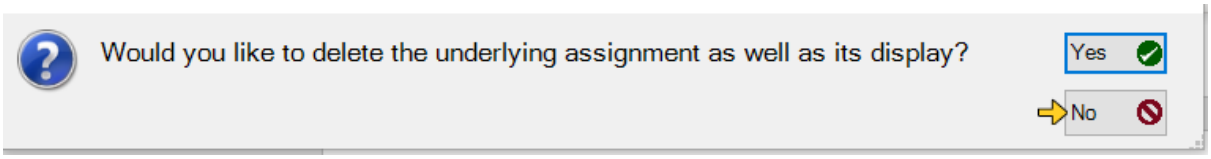
When a token arrives, *pt ->pt+CT* increases pt by the cycle time.

The final assignment *ut => pt*100/ (now() * SJ*) sets the utilisation

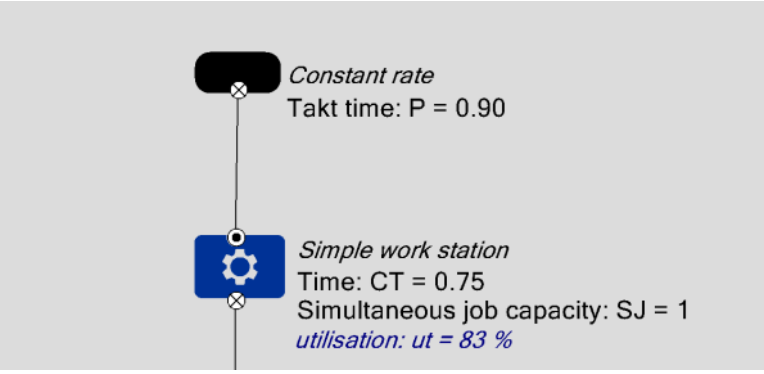
If you step solve this example you will see that initially the utilisation is calculated low. This is because the model does not immediately send tokens into the block. A warm-up period is required to allow the model to settle down. This is discussed further in chapter XXX.

Of course, once you have established that these calculations are correct, you can hide them. Delete the displays but keep the underlying variables and assignments :

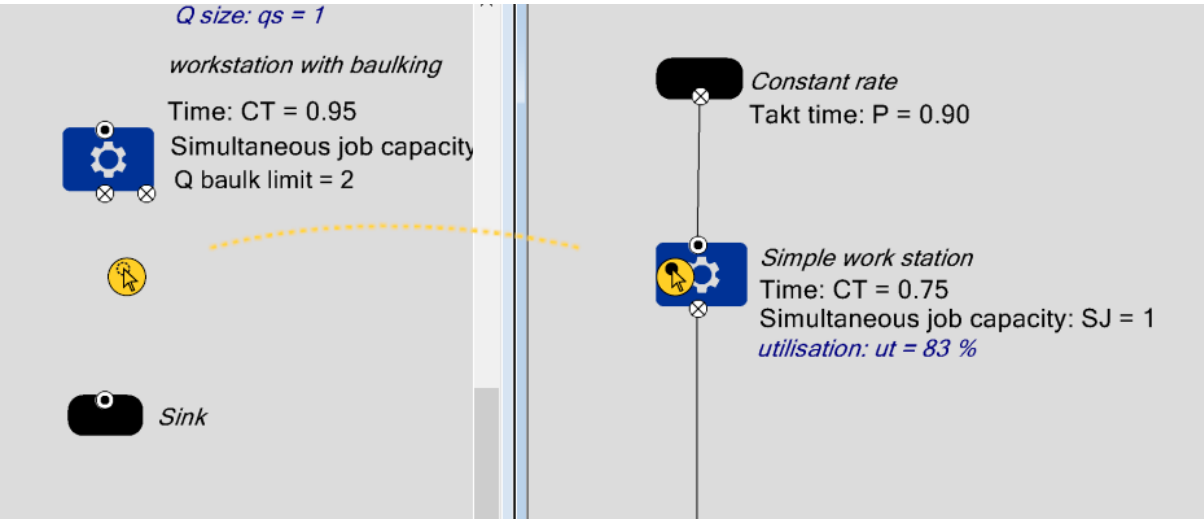




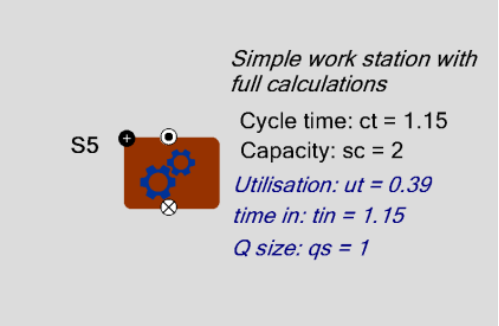
You should aim for something like this :



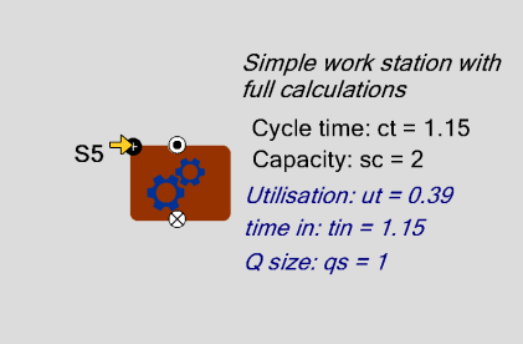
If you are intending to use constant cycle times, this is a perfectly good block. IT is problematic if there is a random element to the job time. By all means, add it to your library and save the library:

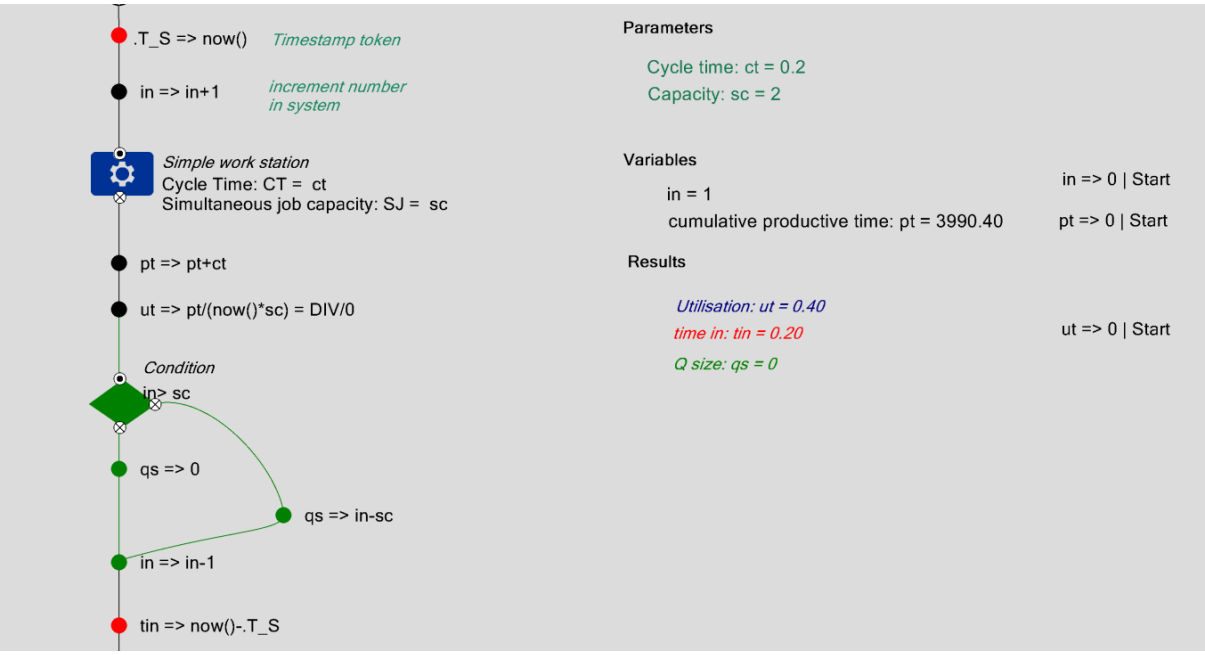


However, you may have already noticed this block in the push manufacturing library :



This is actually a subsystem. The user sets the cycle time and the simultaneous capacity. The subsystem calculates utilisation, the time a token spends in the block (including queueing) and the queue size. This works better with random demand. By all means, take a look inside :





Hopefully this is reasonably self explanatory. One of the main points here is that in Aliquis you can make your own complex blocks.

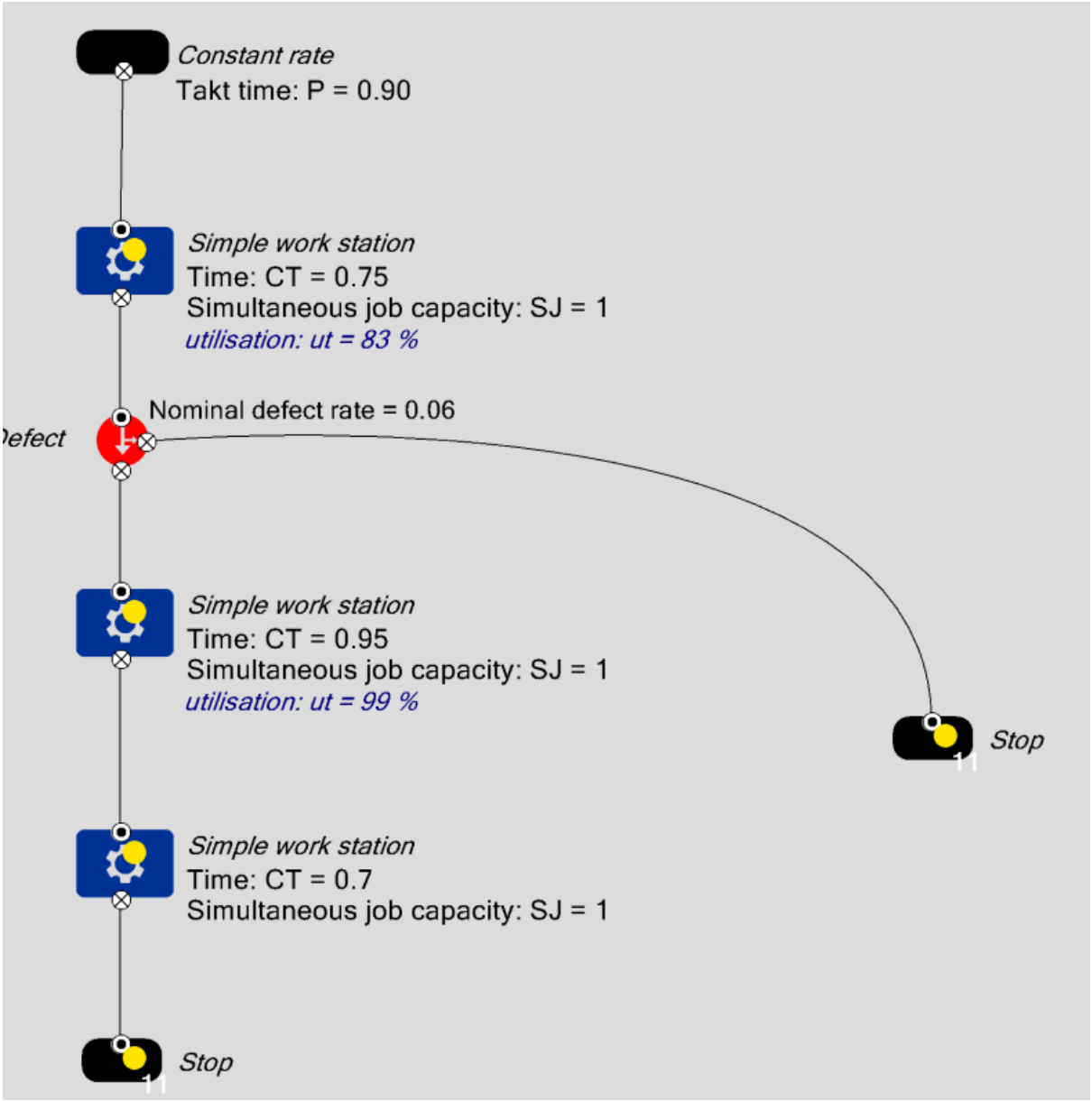
This will be considered more in chapter XXX.

Bottleneck Masking – Another example

We have already seen that an upstream bottleneck can mask one further downstream. Another common example of bottleneck masking is caused by scrappage.

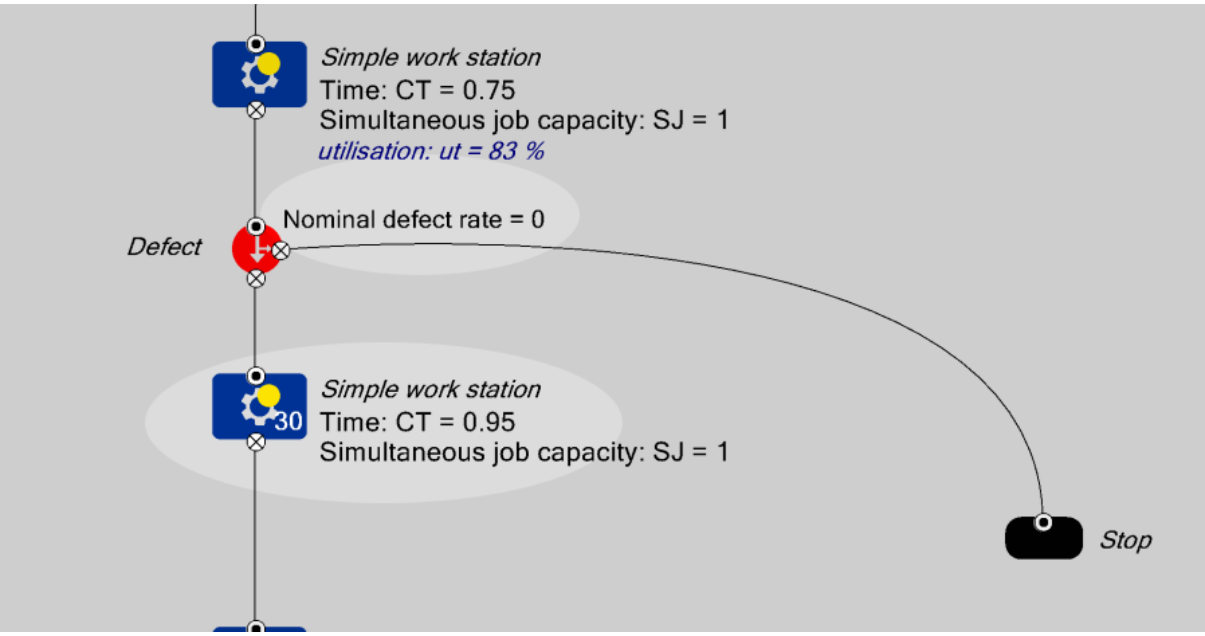
Check out *Chapter8Example3*:

Do a quick solve :



The second workstation has a cycle time greater than the takt time. Why does it not form a bottleneck?

Try reducing the scrap rate to zero and try again.

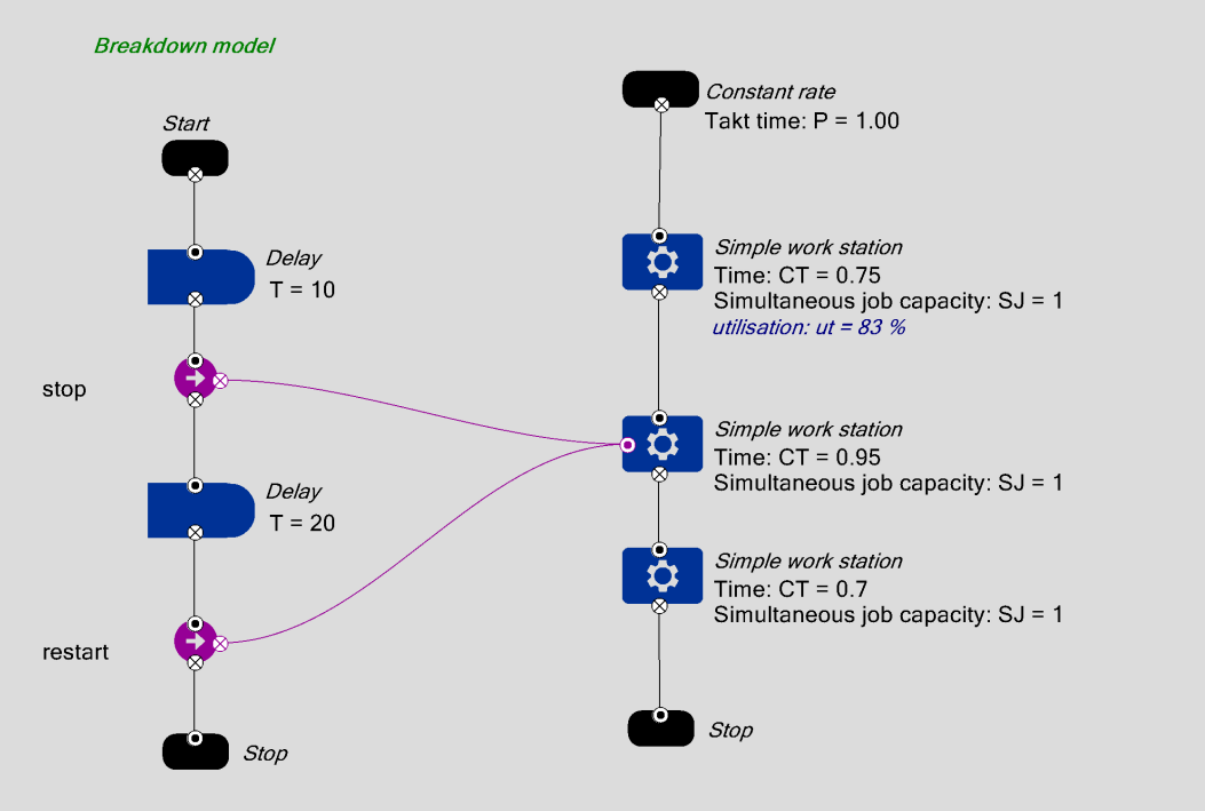


The scrap block reduces the average rate that tokens hit the second workstation. In the real world, you might find that you invest in systems with superior quality and succeed in reducing scrap rates for a particular process. This might mean that you expose a bottleneck further downstream.

Breakdown effects

Breakdown modelling is dealt with more fully in chapter 9. However it is worth noticing the effect that a break down might have on queues.

Open up *Chapter4Example4*



On the left we have a simple breakdown model for the second workstation. After 10 seconds a stop message will be sent to the workstation. The breakdown will last for 20 seconds and the workstation will be restarted.

Animate the solve. Notice how the queue builds up after the stop. Notice also how it *slowly* recovers after the restart.

If you set the second workstation cycle time to the takt time (1). You will notice the queue never reduces. If you reduce it to say 0.8, you will notice that the queue reduces faster. In a system where there are many breakdowns of variable duration a bottleneck can form because of them. Bottleneck masking on the downstream elements can also occur. Similarly to the other masking phenomena, it is possible to invest in the reliability of machines, only to find that it unmasks a downstream bottle neck.

Summary

This chapter brings in some of the key concepts of manufacturing systems engineering . You should now understand what takt time, cycle time and production capacity mean – and how they are interrelated. You should have an appreciation for utilisation and how queues (bottlenecks) can be formed. Finally you should realise the existence of masked bottlenecks. Manufacturing engineers often complain that they are chasing their tails – Investing money on fixing a problem, only to find a “new” problem occurs further down

stream. Actually these are not new problems – rather deficiencies in the system design which only come to light when other problems are fixed.

Exercise

Open Chapter8Exersise1. This is a simplified process for manufacturing a wheel hub. It is colour coded to show which kind of facility must be used for which process elements. Your company needs to make 50000 of these hubs a year in a single shift system. What is your recommended takt time. Neglecting breakdowns and scrap, devise a production system which maximises utilisation but ensures that no bottle necks occur

Questions

1. This chapter outlines one of the key reasons for modelling a production system before you spend real money on it. What is it?

Answers

1. Many problems of hidden bottlenecks can be resolved in advance with modelling.