assignments in each process to use the updated values of both x and y. Each process can use
the updated value of its own variable, but it can't see the updated value of the other process's
variable. The solution you find in many textbooks is to use shared variables, so every
process can see the current values of all the variables, and also, according to operating
system textbooks, you need synchronization [24] here. You have to make sure that the
second assignment in each process waits until the first assignment of the other process is
finished. But the truth is, you don't need shared memory, and you don't need
synchronization. You just need to write what you meant. [25] Here's what you should have
written. First there's a parallel composition of 2 assignments. And then, sequentially after
that, there's another parallel composition. The need for shared memory and synchronization
within processes was just a symptom of writing the wrong program.

[26] I have been saying that concurrent composition is just conjunction, but that's
because I wasn't talking about time. If you include time, [27] here's concurrent composition.
It's still [28] P [29] and [30] Q, but [31] P might say one thing about the final time, and Q
might say something contradictory about the final time. And really, the final time is the
maximum of what P says it is and what Q says it is. So that's what this definition says.

[32] There are some laws about concurrent composition that might come in handy.
[33] The first one is the substitution law. When you have a concurrent composition of
assignments, followed by any specification, the result is the same as the specification but
make substitutions for all the assignments concurrently. The law shown here is for 2
assignments, but it could be any number. The substitution law we have been using is just the
special case when it happens to be one assignment. The [34] next law says concurrent
composition is symmetric, and the [35] next one says it's associative, so if we have 3 or
more processes, we don't need to bother with the parentheses. And the [36] first distributive
law also works for both sequential and concurrent composition. It says if you have P in
parallel with either Q or R, then either you have P in parallel with Q or you have P in
parallel with R. The [37] other 2 distributive laws don't work for sequential composition,
just for concurrent composition.

[38] Concurrent composition requires us to partition the variables, so each process
can work on its own variables independently. And that's fine when the variables are all small
ones, like integer variables and binary variables. But a list variable can be a lot of memory,
and sometimes we need to partition within the list to allow concurrent processes to work on
different parts of the list. [39] Here's the definition of array element assignment, saying that
one item is changing value, and all other items are staying unchanged, and all other
variables are also unchanged. But now, if this assignment occurs in a process in a concurrent
composition, then it should not say that everything else is unchanged. It should just say [40]
that all other items in this part of the partition are unchanged, and all other variables in this
part of the partition are unchanged. As a good example [41], I'll show you how to find the
maximum value in a nonempty list in log time. I [42] define specification findmax of two
variables i and j to say that the maximum value in the nonempty segment of the list from i to
j is put into L i. So findmax says find the maximum item in a segment, and put it at the left
end of the segment. We want the maximum of the whole list, so that's [43] findmax from 0
to the length of the list, and we'll look for the answer at L 0. And [44] here's the refinement.
[45] If j minus i equals 1, then it's a segment of size 1, so the maximum is the one and only
item, and it's already at the left end of the segment, so there's nothing to do. [46] Else. Else
means it's not 1 item, and we know it's nonempty, so it's 2 or more items, and we can split it
into 2 nonempty halves, and find the maximum of each half in parallel, and put those 2
maximums at the left end of each half. Now we want the maximum of the whole segment,
[47] so we take the maximum of the 2 items, and put it in the left end. And we're done. For
recursive time, we need t gets t plus 1 at the start of the else-part. [48] Recursive time is the
ceiling of the log of the segment length.