amsterCHEM
tailor-made engineering software solutions

Slide 1

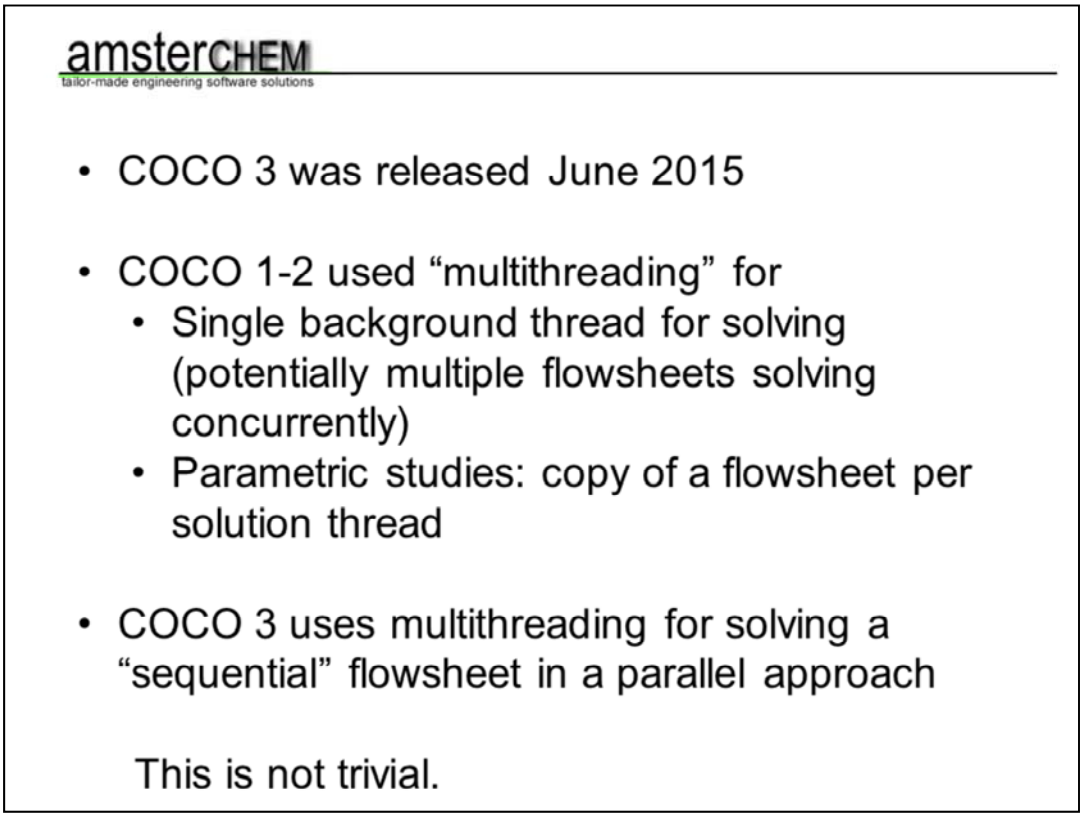
COCO 3.0

SOME EXPERIENCES WITH MULTI-THREADING

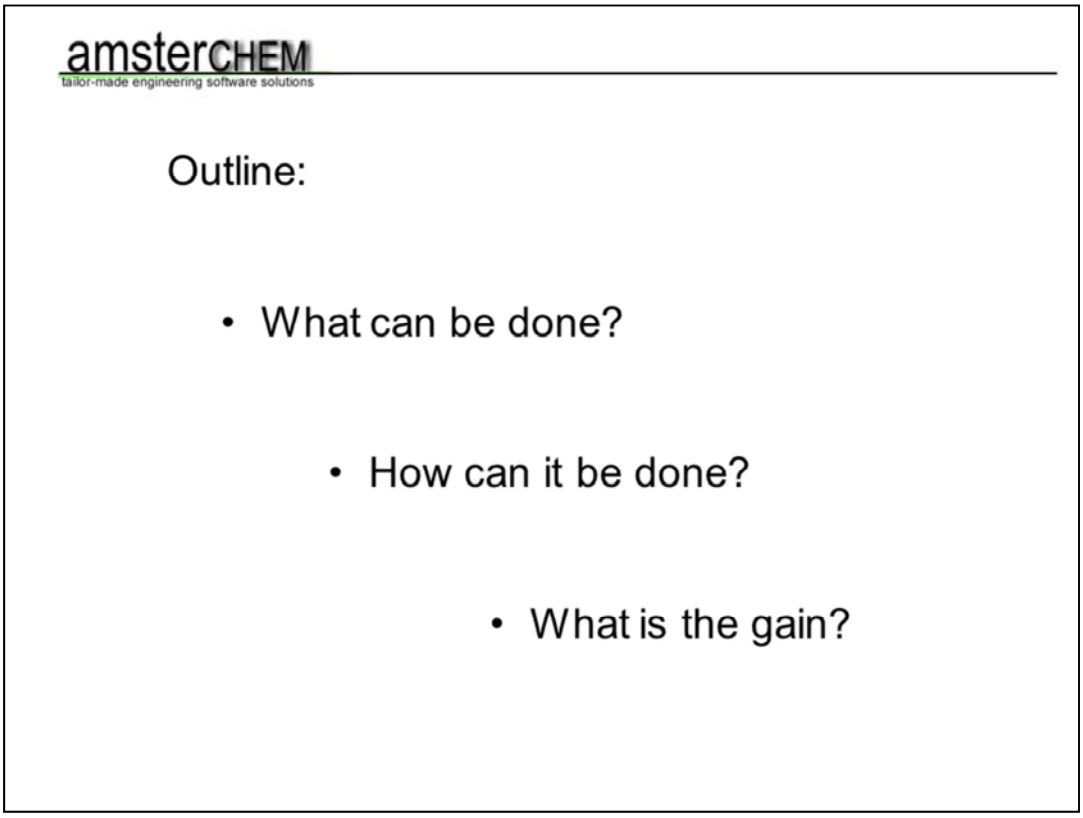
Jasper van Baten. AmsterCHEM
Richard Baur. COCOSimulator

CO-LaN Annual meeting Sept 2015, Amsterdam, Netherlands

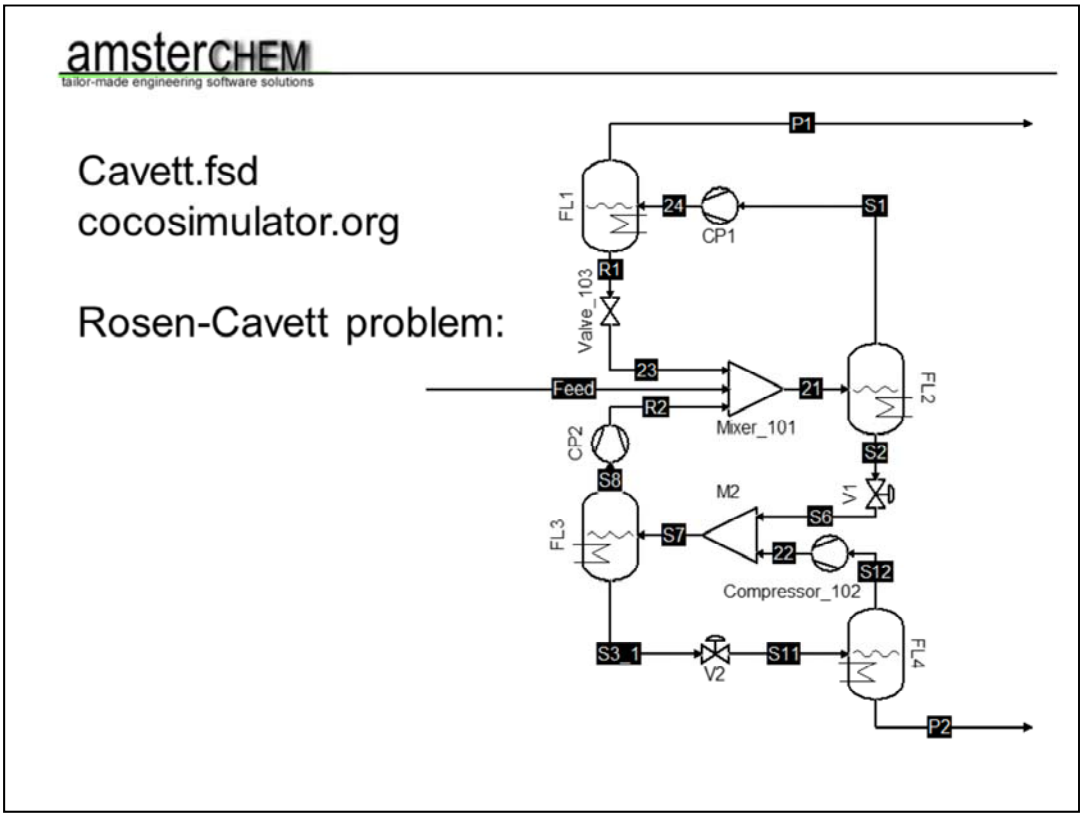
Hi. My name is Jasper. Together with Richard we thought of some ways that could make a parallel approach to sequential flowsheeting attractive. Although it only partially related to CAPE-OPEN, this is about COCO and about aspects of simulations that apply to multiple simulators, so I thought that the annual CO-LaN meeting was a good place to share some of my experiences with it.



COCO 3 was released in June this year. COCO 1 and COCO 2 already used multi-threading in some form, and I have reported on that in earlier presentations. However, the level of concurrent processing, that is processing at the same time using multiple threads, was limited to solving flowsheets in a single background thread and to solving individual cases of a parametric study each in their own thread. In COCO 3, we came up with a way to actually divide the calculations of solving the flowsheet over multiple threads. To an extent, that is. COCO is of course a sequential steady-state flowsheeting, and to make a process that is in nature sequential process in parallel is not trivial.



So we will look at what can be done and has been done. And how it can be done, and what the gain is. Let's set the scene.

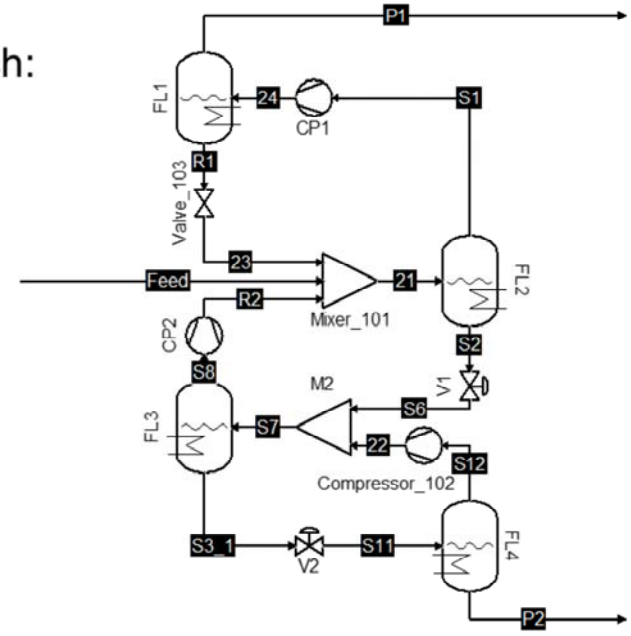


This is the Rosen-Cavett problem. Along with the HDA problem, this was the first demo flowsheet of COCO. I remember when this took minutes to solve. Currently it takes about a second, on my machine. Our starting point will be the classical approach.

amsterCHEM
tailor-made engineering software solutions

Classical approach:

1) Partition
2) Cut
3) Sequence
4) Iterate



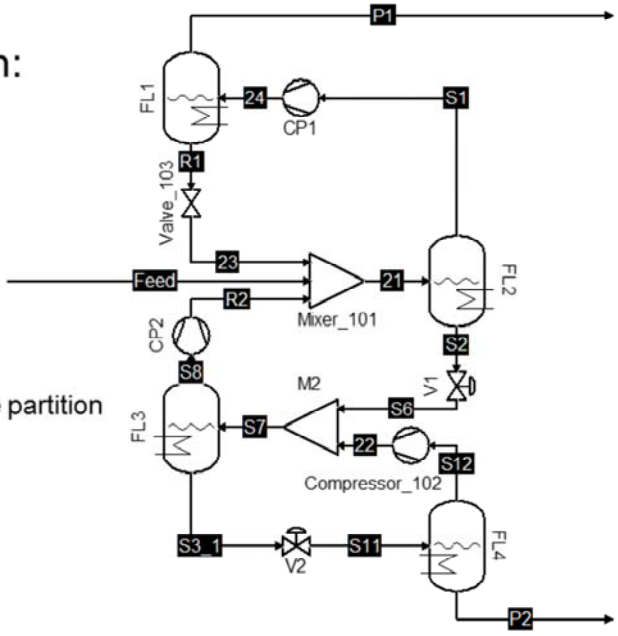
In the classical approach, we partition the flowsheet, then find tear streams, also known as cut streams, then we sequence the order of operations of unit operations, and then we iterate over guessing cut streams such that the cut stream guess results the same value of the cut stream after evaluation of all the unit operations, within tolerance. At that point, we have found the solution to the flowsheet problem.

amsterCHEM
tailor-made engineering software solutions

Classical approach:

1) **Partition**
2) Cut
3) Sequence
4) Iterate

- This Flowsheet is a single partition
- Tarjan, 1972, adopted



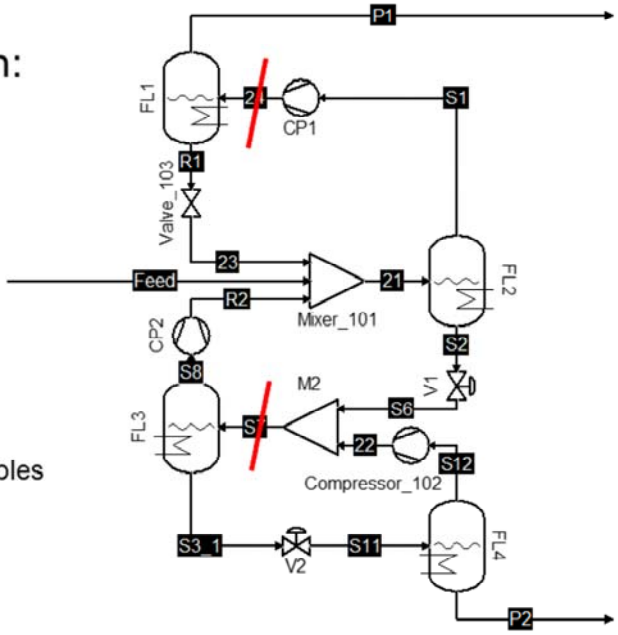
I can be quick about partitioning. Read Tarjan's article from 1972 about strongly connected components, and implement something along those lines. COCO 2 actually had a home brew partitioner, but Tarjan's solution I have to concede is superior, so I adopted it. This particular flowsheet is one single partition; all the units are strongly connected by recycles.

amsterCHEM
tailor-made engineering software solutions

Classical approach:

1) Partition
2) Cut
3) Sequence
4) Iterate

- Recycle detection
- Recycle breaking
- Minimum number of variables
- Combinatorial problem
- Not unique



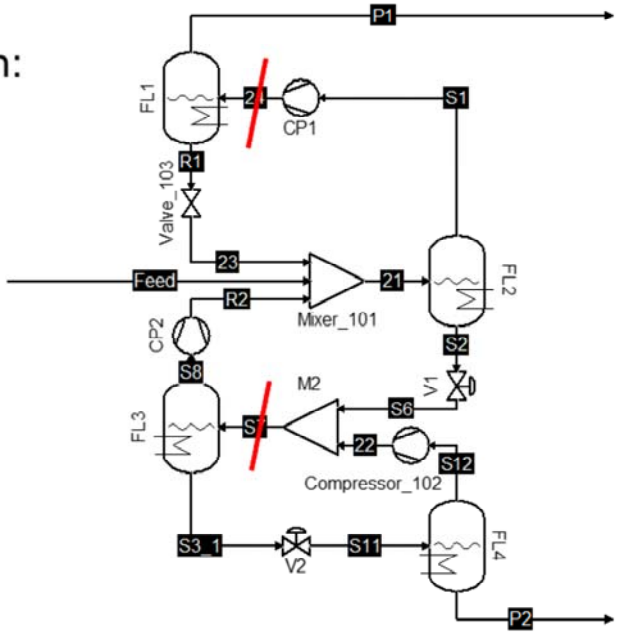
Next step us to find the cut streams. We must cut a sufficient amount of streams, such that all recycles are broken. But not too many. Particularly, each state variable on a cut stream gives us a variable in our overall problem, and we want to keep that to a minimum. So we are looking for a set of cut streams that results the minimum number of cut stream variables, while breaking all recycles. This is, in principle, a combinatorial problem, and that is exactly how it is approached in COFE. Note that the answer is not unique, so there is room for applying additional criteria to the cutting. We can for example cut trough streams 24 and S7, as shown on the slide.

amsterCHEM
tailor-made engineering software solutions

Classical approach:

1) Partition
2) Cut
3) **Sequence**
4) Iterate

- Evaluate cut streams
- Sequence units that have all feeds known



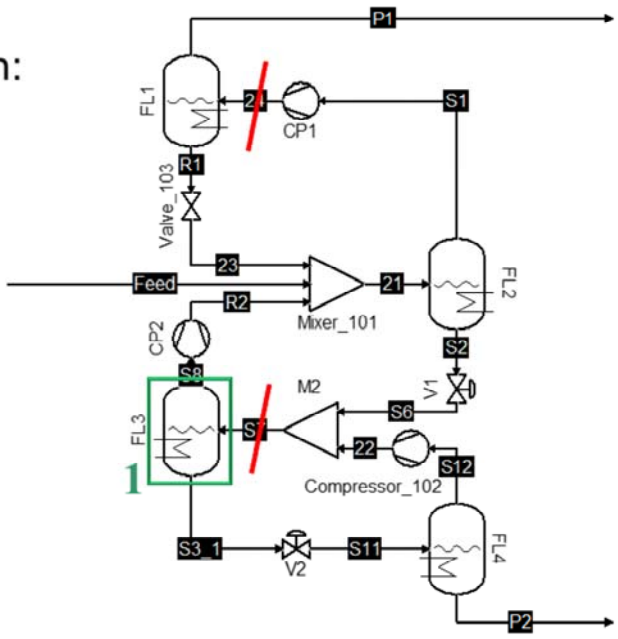
The sequencing is trivial. Given known guesses for cut streams, find unit operations that can be evaluated because all feeds are known. This in turn makes that their products are known, and we can sequence the next unit operations. For example….

amsterCHEM
tailor-made engineering software solutions

Classical approach:

1) Partition
2) Cut
3) **Sequence**
4) Iterate

- Evaluate cut streams
- Sequence units that have all feeds known

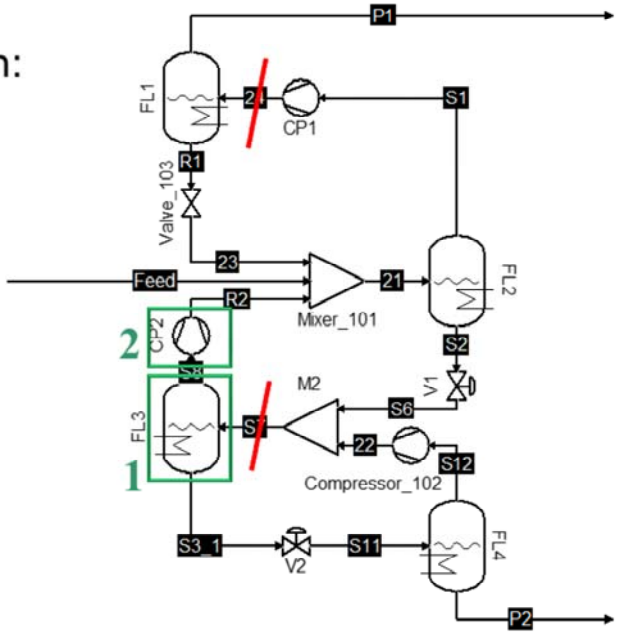


amsterCHEM
tailor-made engineering software solutions

Classical approach:

1) Partition
2) Cut
3) **Sequence**
4) Iterate

- Evaluate cut streams
- Sequence units that have all feeds known

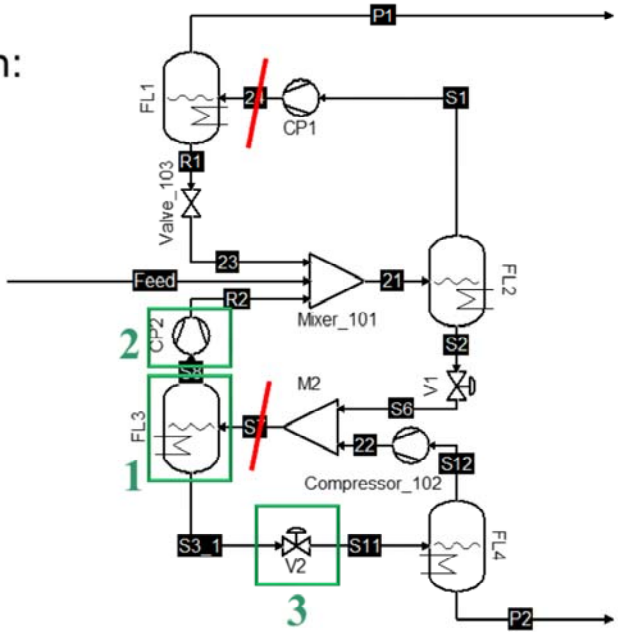


amsterCHEM
tailor-made engineering software solutions

Classical approach:

1) Partition
2) Cut
3) **Sequence**
4) Iterate

- Evaluate cut streams
- Sequence units that have all feeds known

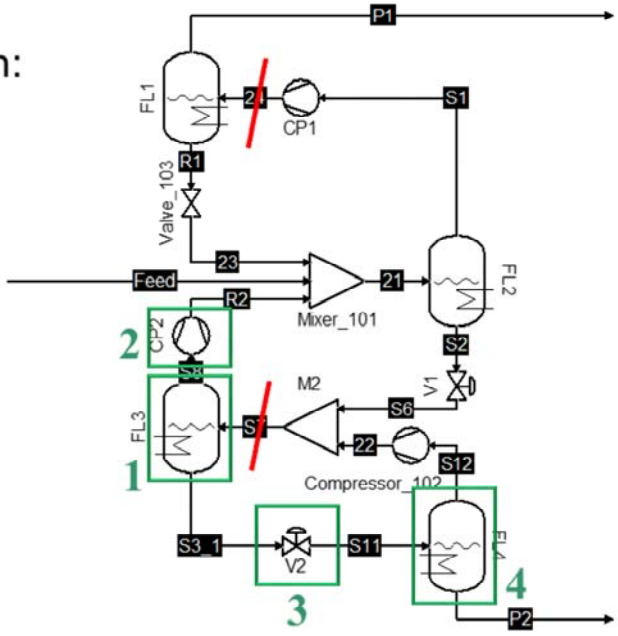


amsterCHEM
tailor-made engineering software solutions

Classical approach:

1) Partition
2) Cut
3) **Sequence**
4) Iterate

- Evaluate cut streams
- Sequence units that have all feeds known

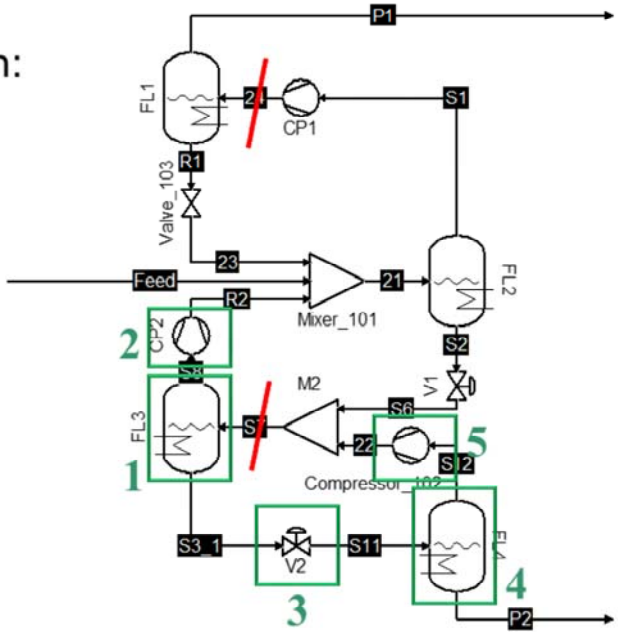


amsterCHEM
tailor-made engineering software solutions

Classical approach:

1) Partition
2) Cut
3) **Sequence**
4) Iterate

- Evaluate cut streams
- Sequence units that have all feeds known

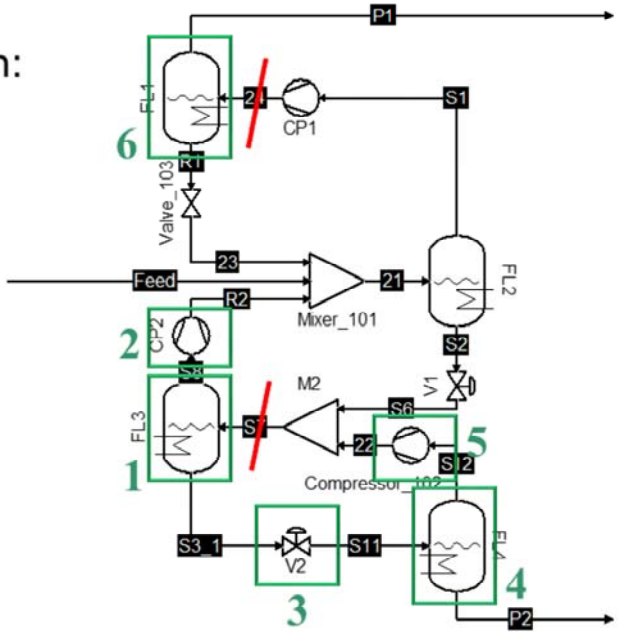


amsterCHEM
tailor-made engineering software solutions

Classical approach:

1) Partition
2) Cut
3) **Sequence**
4) Iterate

- Evaluate cut streams
- Sequence units that have all feeds known

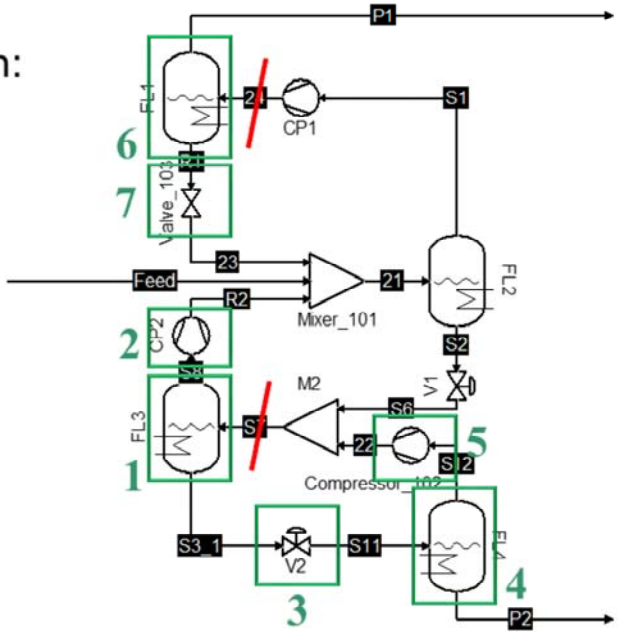


amsterCHEM
tailor-made engineering software solutions

Classical approach:

1) Partition
2) Cut
3) **Sequence**
4) Iterate

- Evaluate cut streams
- Sequence units that have all feeds known

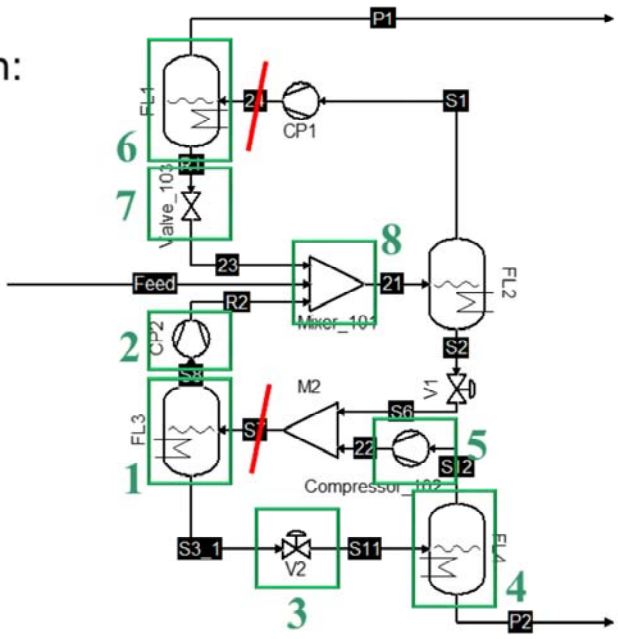


amsterCHEM
tailor-made engineering software solutions

Classical approach:

1) Partition
2) Cut
3) **Sequence**
4) Iterate

- Evaluate cut streams
- Sequence units that have all feeds known

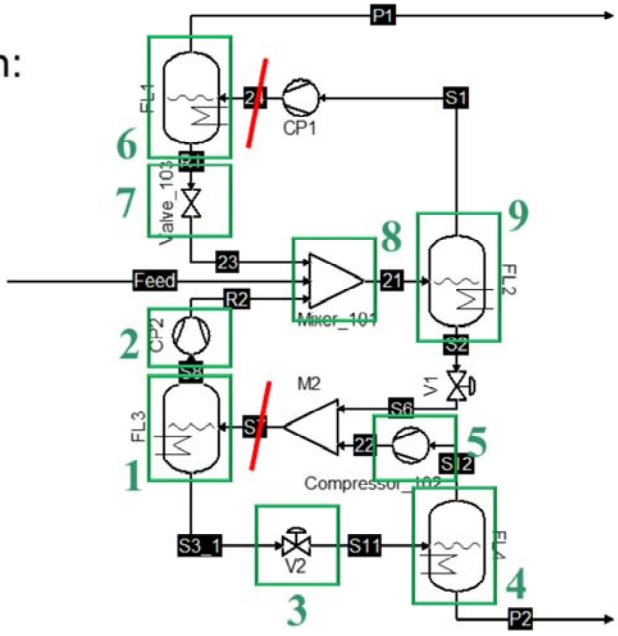


amsterCHEM
tailor-made engineering software solutions

Classical approach:

1) Partition
2) Cut
3) **Sequence**
4) Iterate

- Evaluate cut streams
- Sequence units that have all feeds known

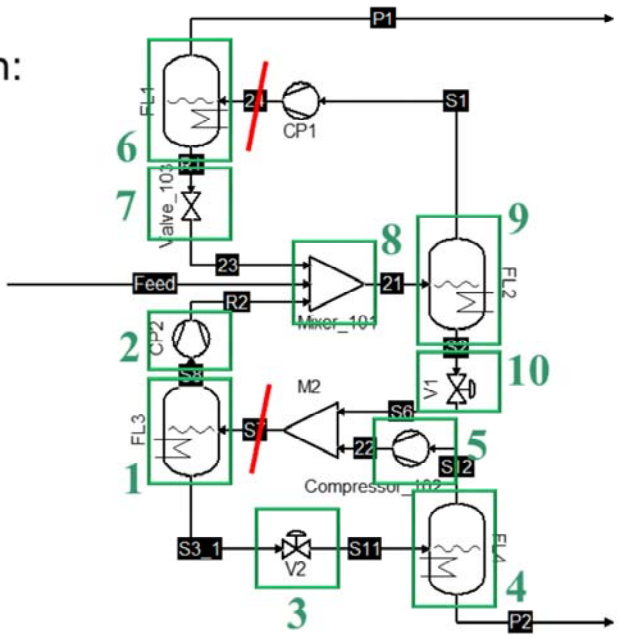


amsterCHEM
tailor-made engineering software solutions

Classical approach:

1) Partition
2) Cut
3) **Sequence**
4) Iterate

- Evaluate cut streams
- Sequence units that have all feeds known

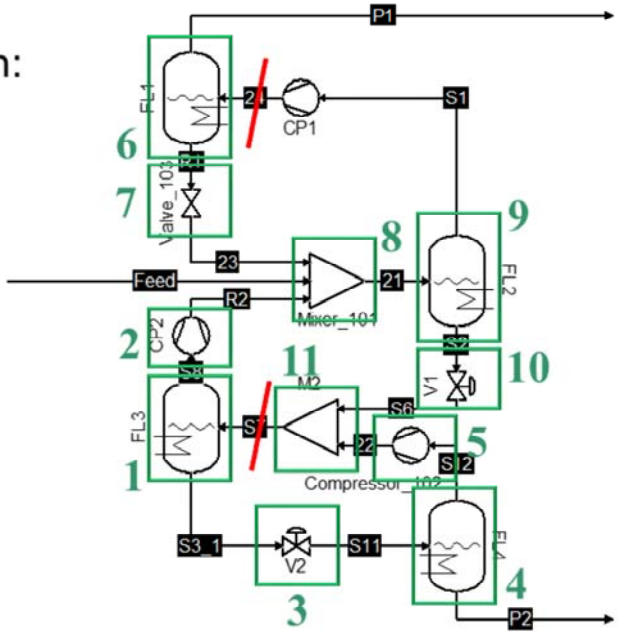


amsterCHEM
tailor-made engineering software solutions

Classical approach:

1) Partition
2) Cut
3) **Sequence**
4) Iterate

- Evaluate cut streams
- Sequence units that have all feeds known

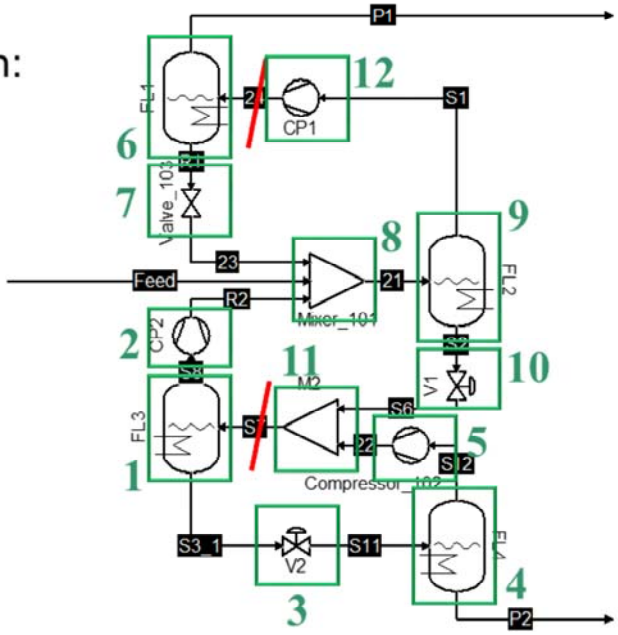


amsterCHEM
tailor-made engineering software solutions

Classical approach:

1) Partition
2) Cut
3) **Sequence**
4) Iterate

- Evaluate cut streams
- Sequence units that have all feeds known



We have sequenced all 12 unit operations, and it takes a sequence of 12 items. Obviously. If we are evaluating sequentially, we need 12 evaluation cycles to evaluate 12 unit operations.

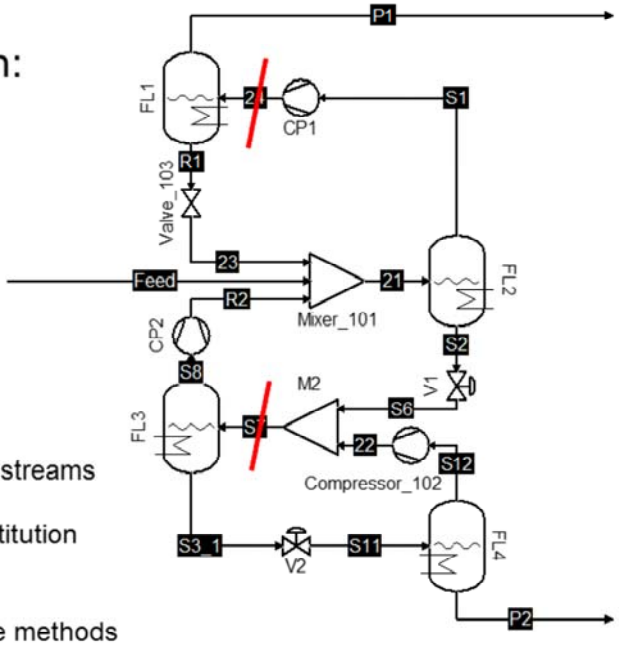
amsterCHEM
tailor-made engineering software solutions

Classical approach:

1) Partition
2) Cut
3) Sequence
4) **Iterate**

- Guess cut streams
- Evaluate all units
- Results new value for cut streams

- Amendable to direct substitution
  - Wegstein
  - GDEM
- Amendable to Newton-like methods
  - Derivatives determined by perturbation



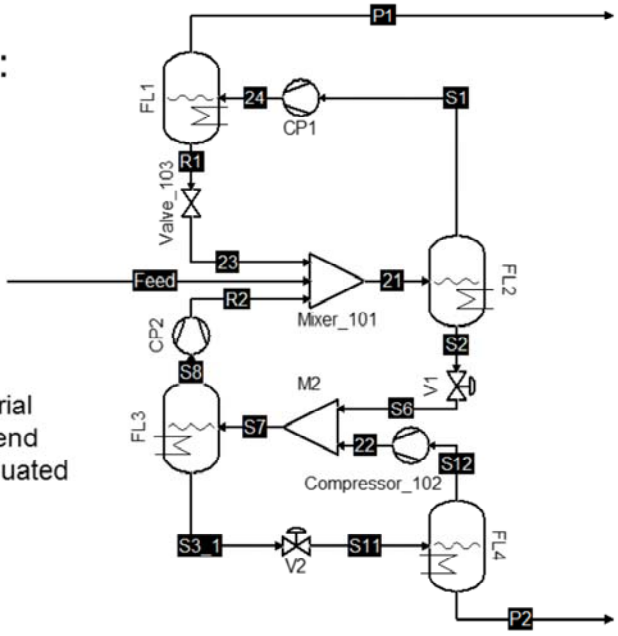
Then we need to iterate to solve the flowsheet. Starting from a guessed value of the cut streams, evaluate all units in the determined evaluation order. This results new values for the cut stream, and we are looking or cut stream guesses that results values for the cut stream equal to the guessed value. This problem is amendable to direct substitution. We can accelerate that with for example the Wegstein method, or the General Dominant Eigenvalue method. COFE implements Wegstein at this moment, and will soon provide a General Dominant Eigenvalue approach.
Sometimes we must fall back on a full Newton, because of its superior convergence behavior, or because the problem contains additional equations such as those imposed by controllers, that make that the problem is no longer amendable for direct substitution. This of course requires first order derivatives, that we do not have available analytically, so we estimate them using perturbations.
Most of you will be familiar with the classical approach outlined here. So let us see how we can modify it and benefit from multiple processors and concurrent evaluations.

amsterCHEM
tailor-made engineering software solutions

What can be done:

1) **Partition**
2) Cut
3) Sequence
4) Iterate

- Partioning itself is fast, serial
- Partitions that do not depend on each other can be evaluated simultaneously



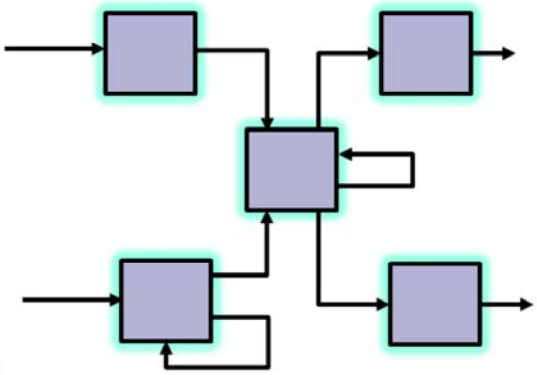
Partitioning itself is fast. We do it on a single core. So no change there. Note that the result of partitioning however does imply a change. Partitions that do not depend on each other can be evaluated concurrently. Let us have a closer look.

amsterCHEM
tailor-made engineering software solutions

What can be done:

1) **Partition**
2) Cut
3) Sequence
4) Iterate

- Partioning itself is fast, serial
- Partitions that do not depend on each other can be evaluated simultaneously



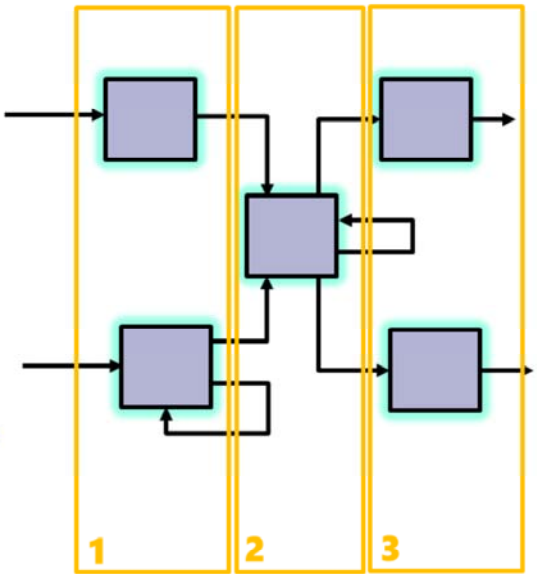
Here we have 5 partitions. Note that there are no recycles between partitions, because partitions are not strongly coupled components. However, recycles may appear within each partition.

amsterCHEM
tailor-made engineering software solutions

What can be done:

1) **Partition**
2) Cut
3) Sequence
4) Iterate

- Partioning itself is fast, serial
- Partitions that do not depend on each other can be evaluated simultaneously



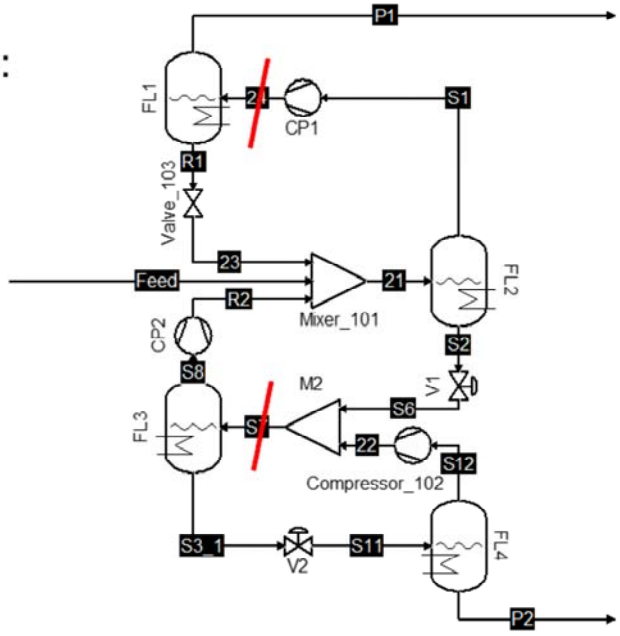
The first two partitions do not depend on each other and can be evaluated concurrently. The middle one can only be evaluated once the first two are done. Once the middle one is done, we can evaluate the remaining two concurrently again.

amsterCHEM
tailor-made engineering software solutions

What can be done:

1) Partition
2) Cut
3) **Sequence**
4) Iterate

- Items that do not depend on each other can be evaluated concurrently



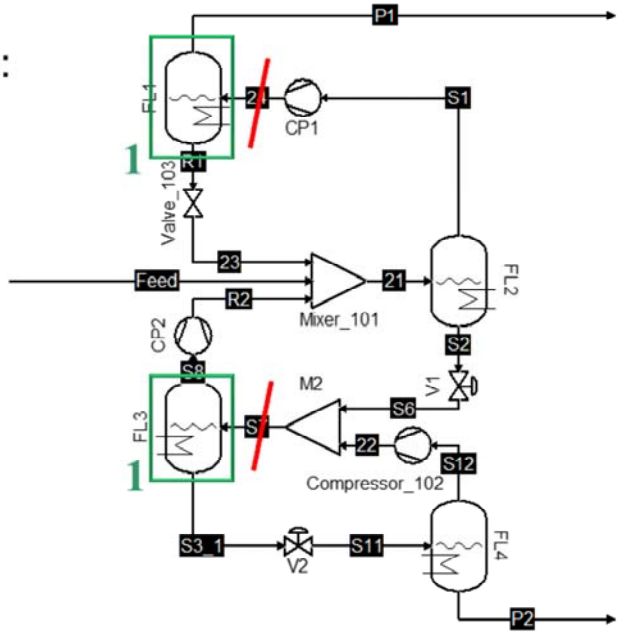
Let us first look at sequencing before we look at cutting. Starting with the same cut streams as before, we now see we have two unit operations with known feeds. Hence, we can sequence two unit operations concurrently.

amsterCHEM
tailor-made engineering software solutions

What can be done:

1) Partition
2) Cut
3) **Sequence**
4) Iterate

- Items that do not depend on each other can be evaluated concurrently



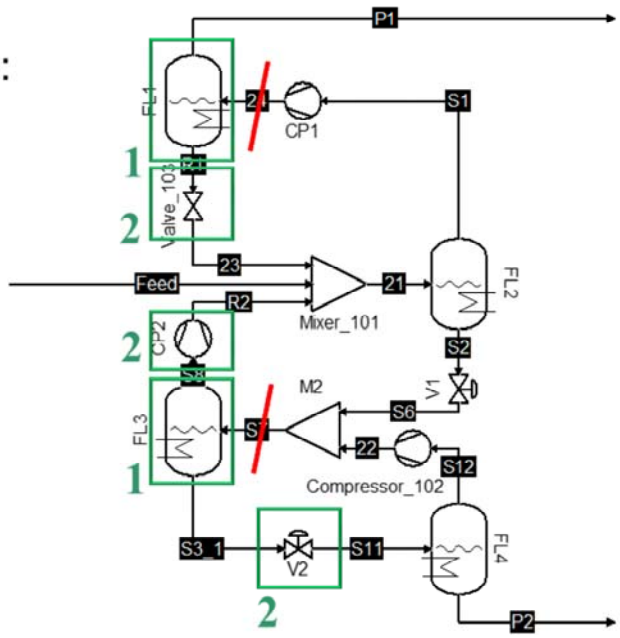
Next, we have 3 unit operations with known feeds.

amsterCHEM
tailor-made engineering software solutions

What can be done:

1) Partition
2) Cut
3) **Sequence**
4) Iterate

- Items that do not depend on each other can be evaluated concurrently



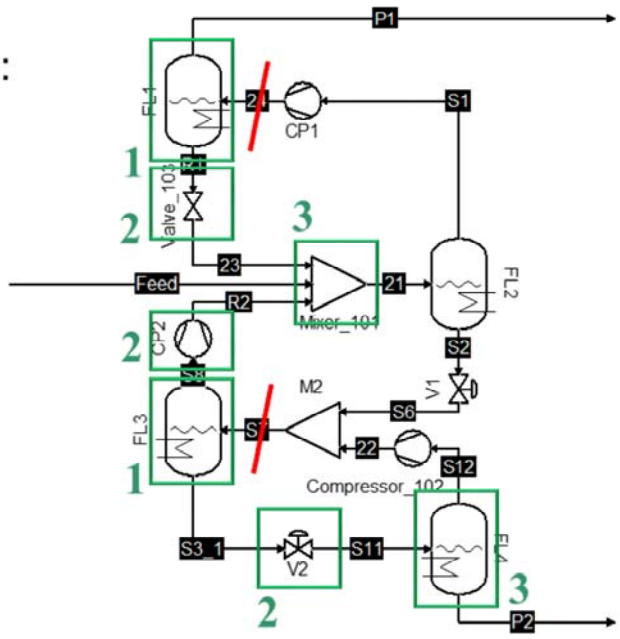
Then 2 more.

amsterCHEM
tailor-made engineering software solutions

What can be done:

1) Partition
2) Cut
3) **Sequence**
4) Iterate

- Items that do not depend on each other can be evaluated concurrently



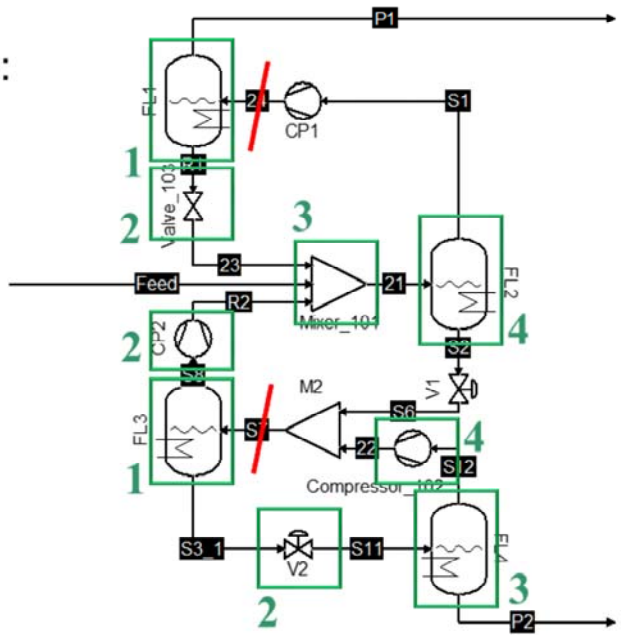
And another 2.

amsterCHEM
tailor-made engineering software solutions

What can be done:

1) Partition
2) Cut
3) **Sequence**
4) Iterate

- Items that do not depend on each other can be evaluated concurrently



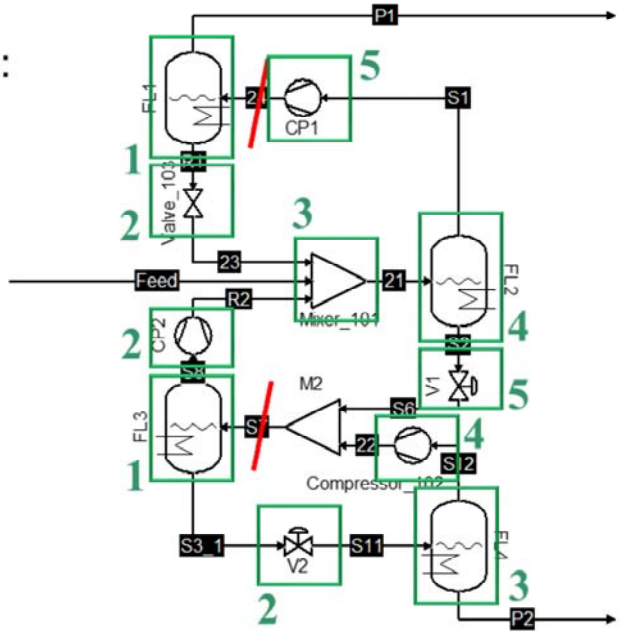
And two more.

amsterCHEM
tailor-made engineering software solutions

What can be done:

1) Partition
2) Cut
3) **Sequence**
4) Iterate

- Items that do not depend on each other can be evaluated concurrently



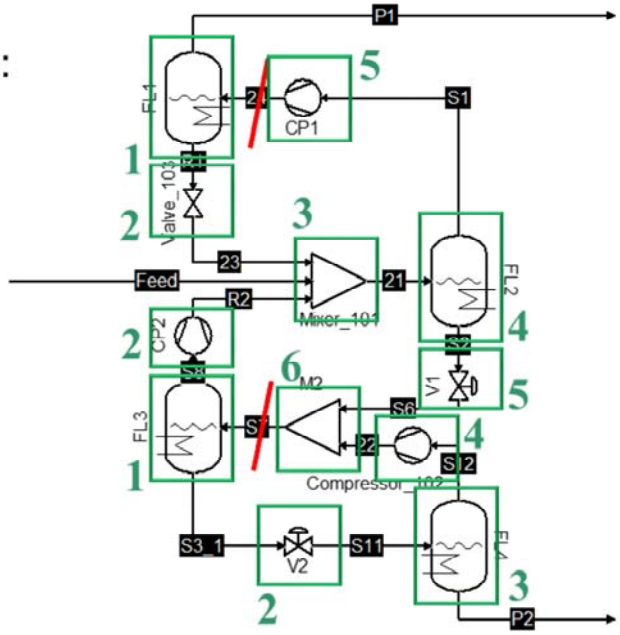
And two more.

amsterCHEM
tailor-made engineering software solutions

What can be done:

1) Partition
2) Cut
3) **Sequence**
4) Iterate

- Items that do not depend on each other can be evaluated concurrently



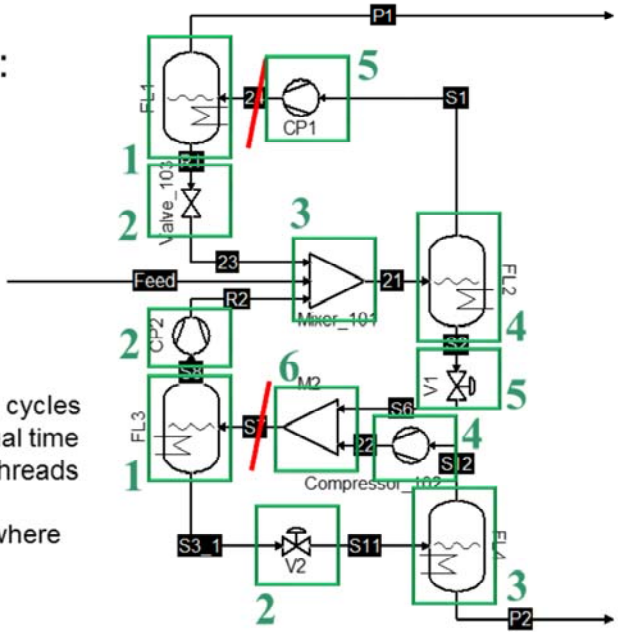
And finally the mixer.

amsterCHEM
tailor-made engineering software solutions

What can be done:

1) Partition
2) **Cut**
3) Sequence
4) Iterate

- 6 instead of 12 evaluation cycles
  - if each unit takes equal time
  - if we have sufficient threads
- The answer depends on where we cut!



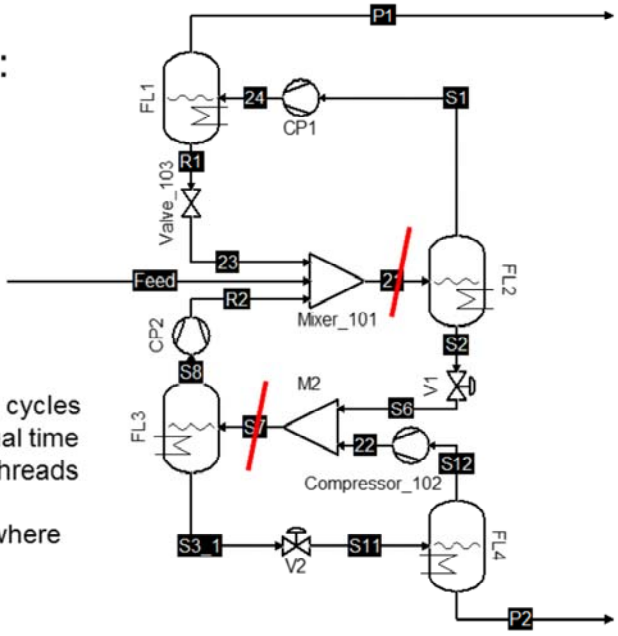
We see, that in case all unit operations would take an equal amount of time to evaluate, and we have a sufficient amount of threads available, we can do a single evaluation in 6 instead of 12 cycles. However, this answer depends on where we cut, so let's look at cutting again.

amsterCHEM
tailor-made engineering software solutions

What can be done:

1) Partition
2) **Cut**
3) Sequence
4) Iterate

- 6 instead of 12 evaluation cycles
  - if each unit takes equal time
  - if we have sufficient threads
- The answer depends on where we cut!



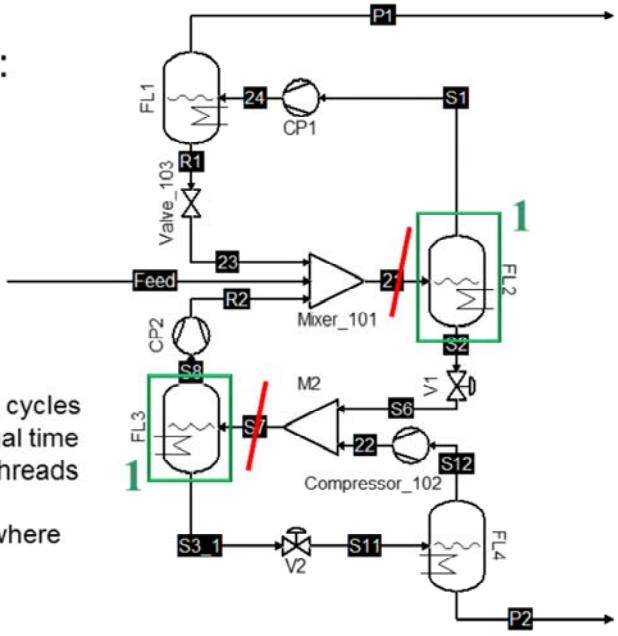
Here is another possible choice of cut streams that breaks all recycles with and equal amount of cut variables. If we sequence from here….

amsterCHEM
tailor-made engineering software solutions

What can be done:

1) Partition
2) **Cut**
3) Sequence
4) Iterate

- 6 instead of 12 evaluation cycles
  - if each unit takes equal time
  - if we have sufficient threads
- The answer depends on where we cut!



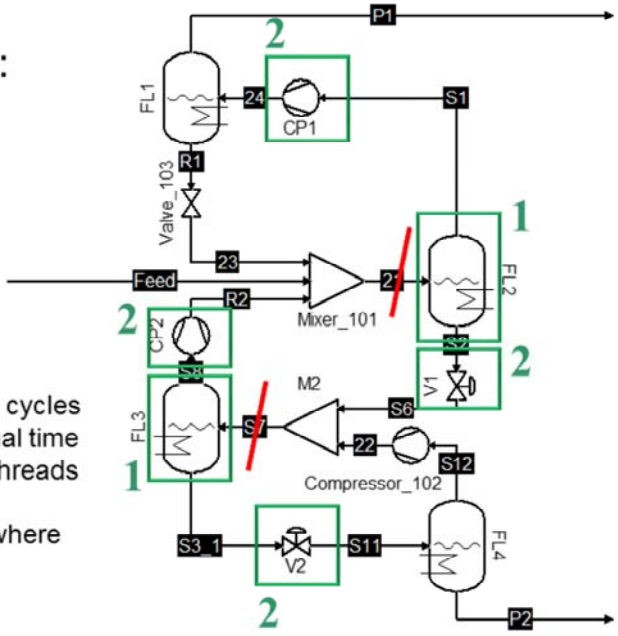
We can start with two unit operations,

amsterCHEM
tailor-made engineering software solutions

What can be done:

1) Partition
2) **Cut**
3) Sequence
4) Iterate

- 6 instead of 12 evaluation cycles
  - if each unit takes equal time
  - if we have sufficient threads
- The answer depends on where we cut!



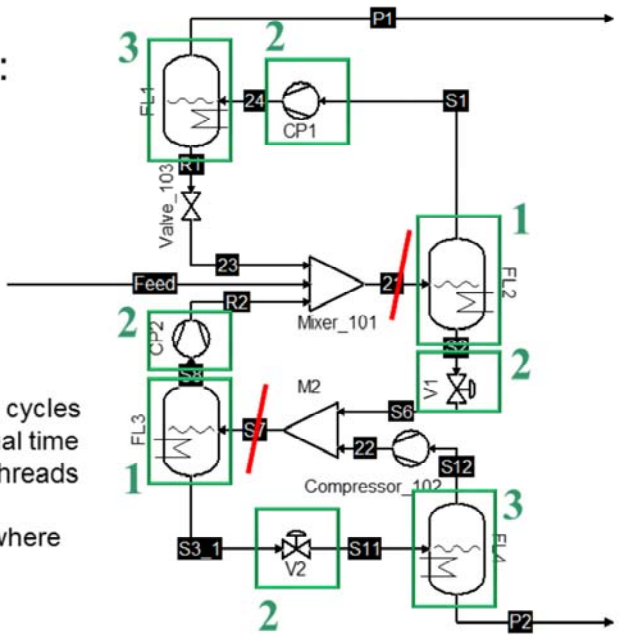
Then four.

amsterCHEM
tailor-made engineering software solutions

What can be done:

1) Partition
2) **Cut**
3) Sequence
4) Iterate

- 6 instead of 12 evaluation cycles
  - if each unit takes equal time
  - if we have sufficient threads
- The answer depends on where we cut!



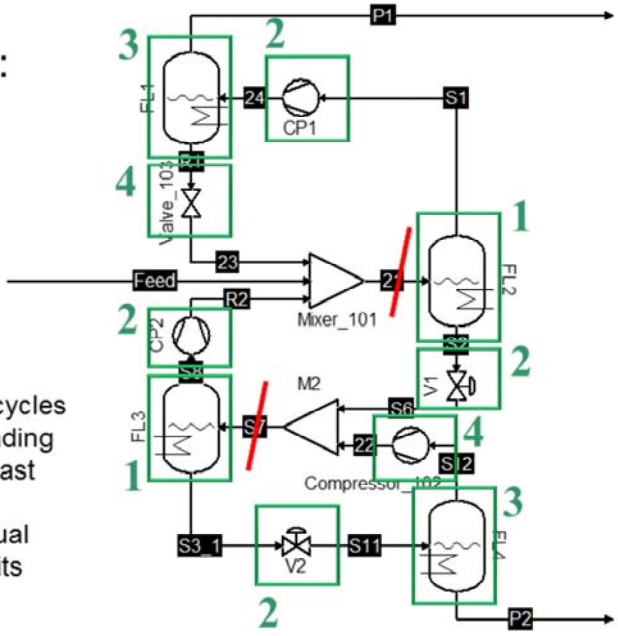
Another two.

amsterCHEM
tailor-made engineering software solutions

What can be done:

1) Partition
2) **Cut**
3) Sequence
4) Iterate

- 5 instead of 6 evaluation cycles
- optimize cut for multithreading
  - always cut through least number of variables
  - take into account actual evaluation time of units



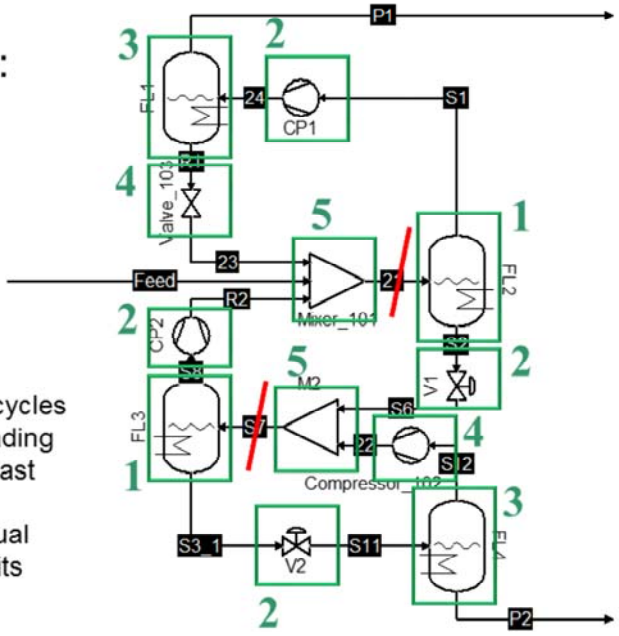
Two more

amsterCHEM
tailor-made engineering software solutions

What can be done:

1) Partition
2) **Cut**
3) Sequence
4) Iterate

- 5 instead of 6 evaluation cycles
- optimize cut for multithreading
  - always cut through least number of variables
  - take into account actual evaluation time of units



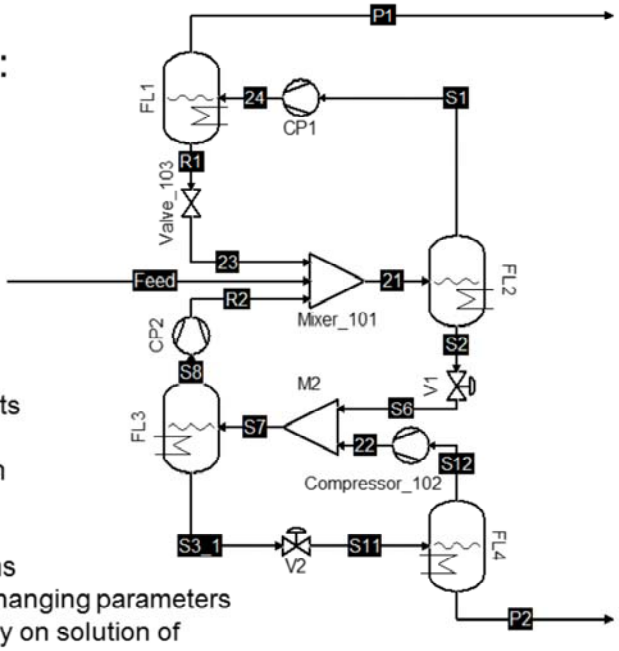
And the final two. Now we have 5 cycles instead of 6 cycles. This implies we need to optimize our cut stream determination for a good load division of calculations. While maintaining the requirement to cut through the least amount of variables and break all recycles. We should of course also take into account the actual time that is used for the evaluation of a unit operation. COFE takes a wall time average over the last 100 or so evaluations of each unit as an estimate.

amsterCHEM
tailor-made engineering software solutions

What can be done:

1) Partition
2) Cut
3) Sequence
4) Iterate

- Evaluate independent parts concurrently
- Newton: perturbations can be done concurrently
- Parametric study:
  - Check which partitions actually depend on changing parameters
  - Partitions depend only on solution of "parent" partition



For iterations, in general, we can concurrently evaluate independent parts. We have optimized the cut stream selection for single function evaluations, but we can also independently queue all perturbations in case of a Newton solver. In a parametric study things get more interesting: COFE keeps track on queueing each case's partitions after its dependencies within the same flowsheet, but also on the same partition in the flowsheet case that holds the initial guesses for the next step. It is not required to wait for the entire parent flowsheet to finished. Just the partitions that initialize the current partition.

amsterCHEM
tailor-made engineering software solutions

What can be done:

- Determine what can be independent
- Evaluate everything as chains of dependency
  - E.g. Newton reformulated as dependent steps
  - No fixed sequence!
- Task queuing:
  - Start *n* dependent threads
  - Job stack with tasks have no prerequisites
  - When tasks finishes, add dependents that have no prerequisites to the task stack

So to summarize, we can start evaluating unit operations for which all prerequisites are known. Independent items can be calculated concurrently. COFE's solvers have been rewritten from scratch: all is expressed as chains of dependent calculations. Newton for example is formulated as a line search that contains dependent function evaluations. The line search in turn depends on a direction calculation, which depends on a Jacobian, which depends on all the individual perturbations. These are all small tasks, and there is a stack of tasks that have no prerequisites. These tasks are divided over n threads. Once a task is finished, all tasks that depend on it, are checked for further dependencies. If there aren't any, the task is added to the execution stack.

amsterCHEM
tailor-made engineering software solutions

How can it be done:

- We need a copy of each PMC in each thread
  - Unit operations: lazy
- We may re-use MOs in each thread
  - Connect each MO to UO in each thread
  - Only if UO that uses MO or predicts MO is not evaluated concurrently
- COFE uses concurrent unit evaluations (Jacobian, Parametric study)
  - Copy of MO in each thread
  - MO State info copied between MOs in threads

About the how I will be short, as I have presented this before: in each thread you will need a copy of each PMC. Unit operations are created the first time they are needed in a thread, to prevent creating multiple copies in case we only need one, that is, for a unit operation that does not appear in a recycle.

For material objects we could get away with reusing a single copy in multiple thread, and connecting this material object to the same port of each copy of the unit operation in each thread, but only in case the material is not concurrently accessed, that is, if there is no other thread that operates on the feeds or products of a unit operation during its evaluation. In COFE, concurrent stream access happens: particularly during Jacobian evaluations and parametric studies. So COFE creates a copy of each material object in each thread. Clearly the state, including the entire phase equilibrium, must be copied between the threads. This state actually is associated with each task that is queued.

amsterCHEM
tailor-made engineering software solutions

How can it be done:

Jacobian construction:

- Cut stream perturbation
  - Remove unit operations that are not affected by perturbation
- Individual unit perturbations followed by chain rule

Which is better?

Interesting to look at, as it is time consuming and a good candidate for performance optimization, is the Jacobian for the Newton method. We can perturb cut stream variables, or we can perturb individual unit operations and construct the Jacobian. Which one is better depends on the problem at hand. Let's have a look.

amsterCHEM
tailor-made engineering software solutions

Cut stream perturbation:

$$f(x) = x^* - x$$

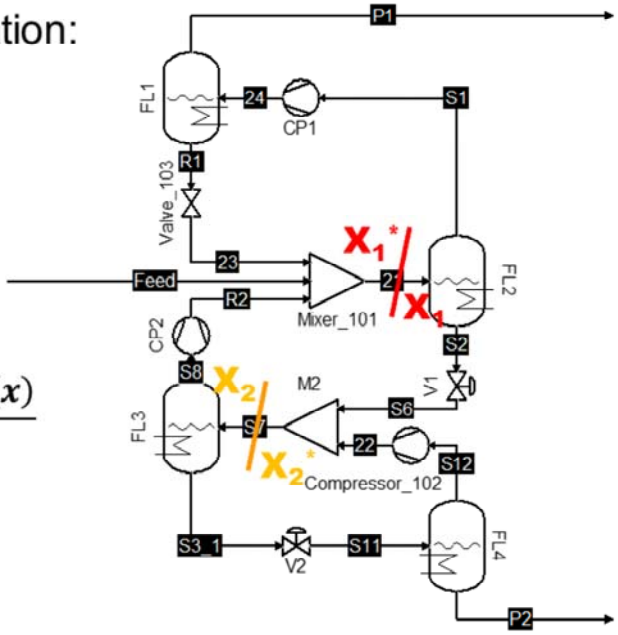
$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

$$x^* = \begin{bmatrix} x_1^* \\ x_2^* \end{bmatrix}$$

$$\frac{\partial f(x)}{\partial x_j} \approx \frac{f(x + \Delta x) - f(x)}{\Delta x_j}$$

$$\Delta x_j = \epsilon$$

$$\Delta x_{k,k \neq j} = 0$$

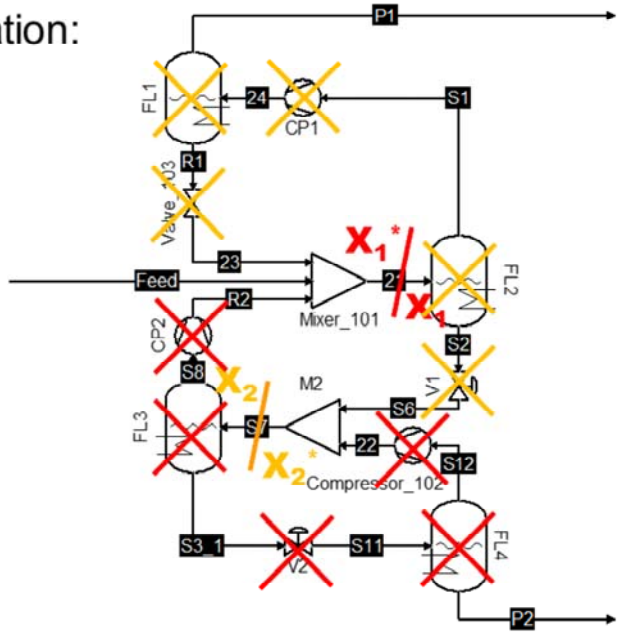


Cut stream perturbation is simple: we apply a small offset to one of the cut stream variables, re-evaluate all the units and check the effect on the outcome of all cut stream variables. COFE 2 did it like that, COFE 3 realizes that you do not need to re-evaluate all unit operations.

amsterCHEM
tailor-made engineering software solutions

Cut stream perturbation:

- Skip units not affected by Δx_j

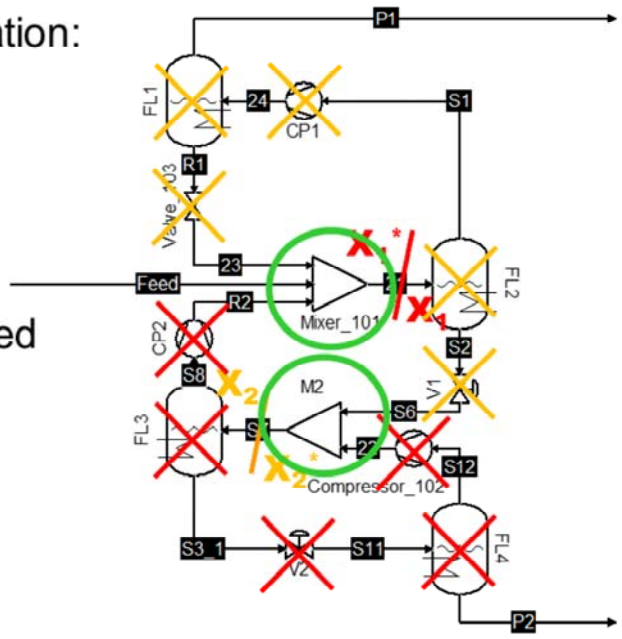


The units with a red cross though it are not affected by the perturbations on stream X1 and need not be reevaluated during such perturbations. Similarly, the units with an orange cross are not affected by perturbations of stream X2.

amsterCHEM
tailor-made engineering software solutions

Cut stream perturbation:

- Skip units not affected by Δx_j
- Remaining units are perturbed multiple times



Two units remain, which are perturbed twice.

amsterCHEM
tailor-made engineering software solutions

Reconstructed Jacobian:

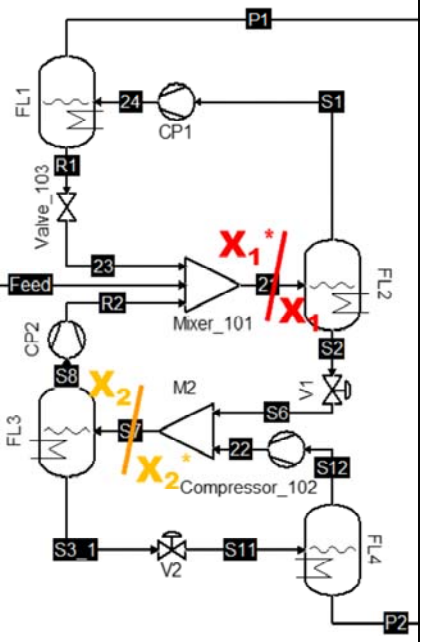
$$\frac{\partial x_1^*}{\partial x_1} = \left(\frac{\partial x_{S1}}{\partial x_1}\right)\left(\frac{\partial x_{24}}{\partial x_{S1}}\right)\left(\frac{\partial x_{R1}}{\partial x_{24}}\right)\left(\frac{\partial x_{23}}{\partial x_{R1}}\right)\left(\frac{\partial x_1^*}{\partial x_{23}}\right)$$

$$\frac{\partial x_2^*}{\partial x_1} = \left(\frac{\partial x_{S2}}{\partial x_1}\right)\left(\frac{\partial x_{S6}}{\partial x_{S2}}\right)\left(\frac{\partial x_{R1}}{\partial x_{S6}}\right)\left(\frac{\partial x_2^*}{\partial x_{S6}}\right)$$

$$\frac{\partial x_1^*}{\partial x_2} = \left(\frac{\partial x_{S8}}{\partial x_2}\right)\left(\frac{\partial x_{R2}}{\partial x_{S8}}\right)\left(\frac{\partial x_{21}}{\partial x_{R2}}\right)\left(\frac{\partial x_1^*}{\partial x_{21}}\right)$$

$$\frac{\partial x_2^*}{\partial x_2} = \left(\frac{\partial x_{S3\_1}}{\partial x_2}\right)\left(\frac{\partial x_{S11}}{\partial x_{S3\_1}}\right)\left(\frac{\partial x_{12}}{\partial x_{S11}}\right)\left(\frac{\partial x_{22}}{\partial x_{S12}}\right)\left(\frac{\partial x_2^*}{\partial x_{22}}\right)$$

For this flowsheet this evaluates in exactly the same number of unit operation evaluations, but..



Another approach would be to perturb the state variables on each feed to each unit operations. The derivatives follow from chain rule, as shown on this slide. For the Cavett flowsheet this results exactly the same number of unit evaluation evaluations, but…

amsterCHEM
tailor-made engineering software solutions

Reconstructed Jacobian:

- Special case for streams at phase boundary
  - Should be perturbed along the phase boundary
  - Saves a perturbation degree of freedom
- Often Jacobian reconstruction leads to fewer unit operation evaluations
- Jacobian reconstruction leads to more and smaller tasks: better load distribution
- (an optimization could be made for variables that remain constant, but this has a distribution penalty)

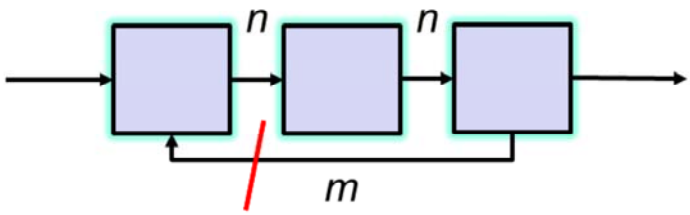
In general this is not the case. In case of multiple recycles, often Jacobian reconstruction leads to fewer unit operation evaluations. Also, we should not perturb away from a phase boundary, as this would result incorrect derivatives. So if we detect streams that are on the phase boundary, that is, streams that have an incipient phase, we can constrain perturbation to the phase boundary, which saves a perturbation.

Even if the amount of unit operation evaluations is equal, in the reconstruction approach there are more, but smaller tasks, that lead to a better load division.

You could also detect by perturbation that a given unit operates at constant pressure and does not change the product pressure, for example. This would eliminate the need to perturb the downstream unit for feed pressure. However, this approach has a penalty of load division, as perturbations need to wait for each other, and COFE does not do so.

amsterCHEM
tailor-made engineering software solutions

Cut stream perturbation may be faster:



Faster to perturb cut steam if $m < n$
Not uncommon:
$n > 10$ (material stream)
$m = 1$ (information stream

Reconstruction is not always faster. Take this example, where we have a unit operations connected with streams with n variables and single recycle with m variables. If m is smaller than n, cutting the recycle and perturbing it will lead to fewer unit operation perturbations. This is not an uncommon situation, if the streams connecting the unit operations are material streams and the recycle is an information stream with for example a controller set point.

Which method is better depends on the partition at hand and can be determined as part of the partition analysis.

amsterCHEM
tailor-made engineering software solutions

What is the gain?

- Parallelization went hand-in-hand with solver improvements
- All demo flowsheets solve 6x – 20x faster now
- Considerable part to be attributes to fewer iterations
- Considerable part to be attributes to fewer evaluations due to
  - Elimination of unit operation evaluations during cut-stream perturbation for Jacobian
  - Elimination of unit operation evaluations due to reconstruction of Jacobian by chain rule
- Remaining metric:
  - CPU usage
  - Wall time for 1 thread vs wall time for *n* threads

Talking about the gain is not so simple. Comparing with previous versions of COCO is not realistic. We have re-written all solvers to fit into the new scheme, and doing so revealed some solver bugs and room for solver improvement. If you would compare, you will find that all demo flowsheets solve 6x to 20x faster now, but a substantial part of that can be contributed to fewer iterations, and to fewer unit operation evaluations when calculating the Jacobian.

The remaining metrics are: how does solving with 1 thread compare to solving with n threads, and what is the CPU coverage.

amsterCHEM
tailor-made engineering software solutions

What is the gain?

- CPU usage near 100% during Jacobian perturbations
  - Wall time near 1/*n* for *n* threads.
- CPU usage near 100% during parametric studies
  - Wall time near 1/*n* for *n* threads.
- CPU usage during 'sequential' function evaluations depends highly on flowsheet structure
  - Gain is higher with more recycles
- Gain here is not great in case of 'bottle neck' unit operation(s)

Scaling and CPU coverage are near perfect during Jacobian perturbations, and during parametric studies.

During 'sequential' function evaluation, the gain depends highly on the structure of the flowsheet. If there are more independent partitions, and more recycles within a partition, the gain is higher.

The gain for sequential evaluations is not great if one path within a recycle uses considerably more CPU time than all other paths. This presents a bottle neck.

amsterCHEM
tailor-made engineering software solutions

What is the gain?

Cavett	1 node	8 nodes
Average (ms)	1780	1366
StdDev (ms)	622 (!)	33

$1/n \rightarrow 77\%$

Newton only:

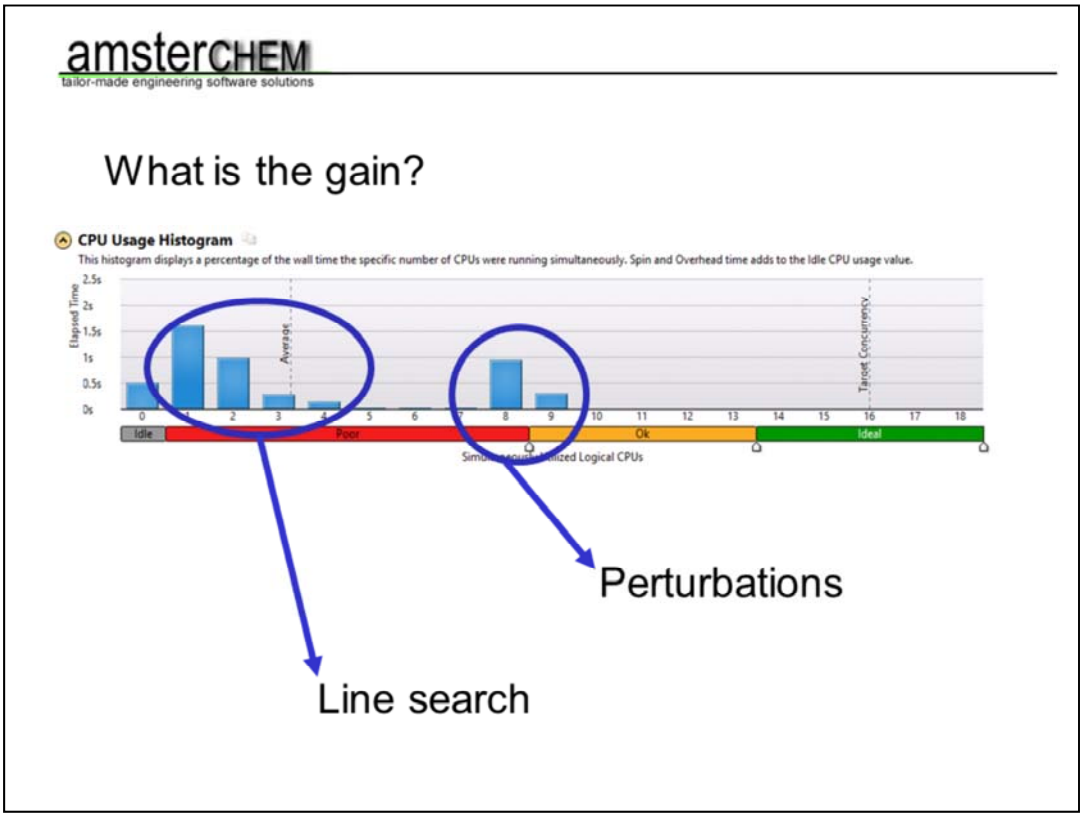
Cavett - Newton	1 node	8 nodes
Average (ms)	16854	5220
StdDev (ms)	2091(!)	83

$1/n \rightarrow 31\%$

(Target for 8 nodes is $1/n \rightarrow 12.5\%$)

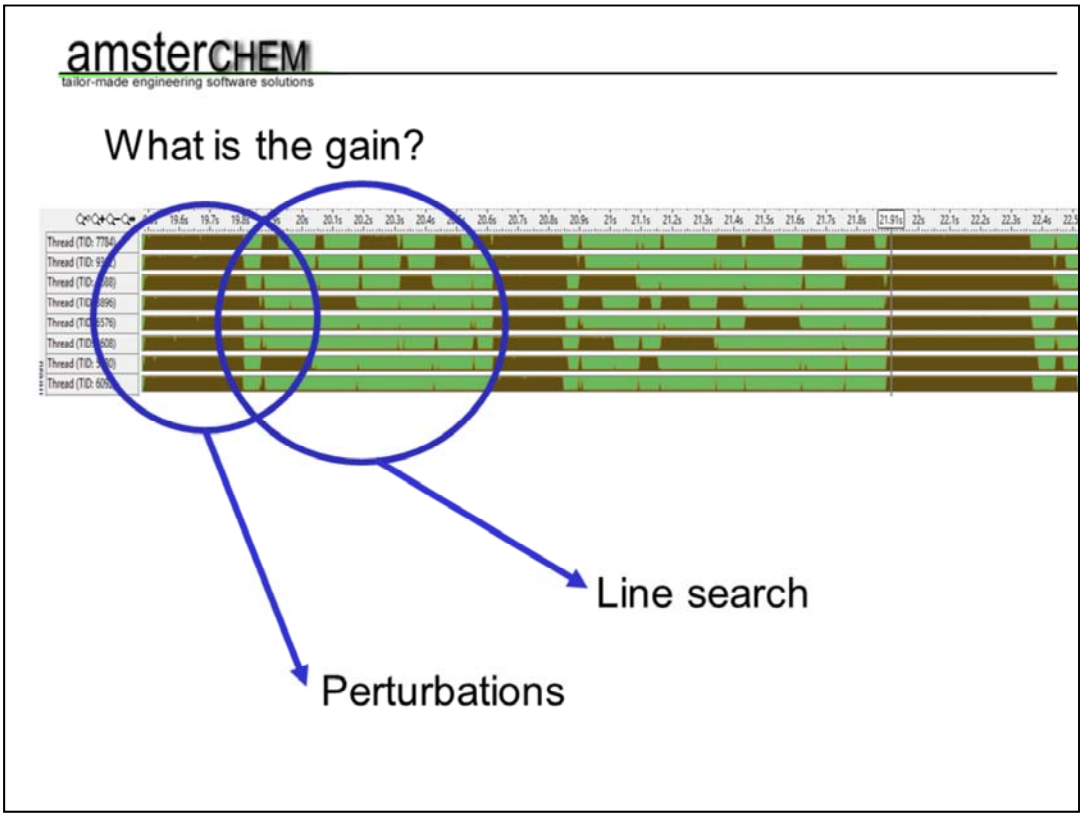
Here we see some results from the Cavett problem. Note that this is not the best problem to test with: we compare a serial solution with a concurrent solution, where the solution time is quick and the penalty for the concurrent solution is the creation of additional objects, the transfer of stream data between threads and the synchronization. We see that using 8 nodes, Cavett solves in 77% of the time compared to the serial case. Note that the standard deviation over 10 measurements is much higher in the serial case, because any delay in a thread diffuses in many threads in the concurrent case. You could argue, considering the standard deviation of the serial case, that there is no difference. The reason in this case is that the flowsheet is so simple it solves with direct substitutions only and there is a bottle neck coming from the topmost two units in the flowsheet.

If we switch off the direct substitutions we notice the 8-node version uses only a third of the time of the serial solution. Clearly more gain there. Why not one 8th? Let's take a peek at the CPU coverage.

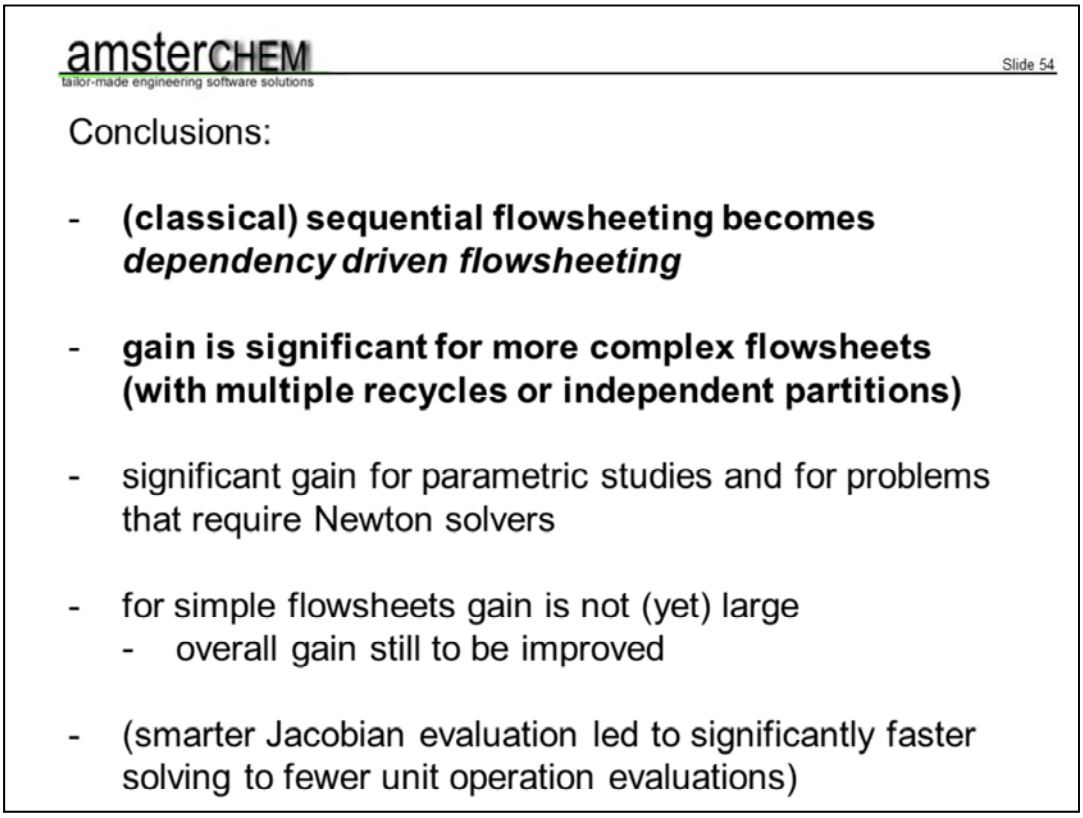


We see that the CPU usage is a 100% of 8 cores a substantial part of the solution process. This represents the Jacobian perturbations. The other part scales as poor as the direct substitution case.

(don’t mind the target concurrency here, this was done on machine with 16 logical but 8 physical nodes).



Here is another way to look at the same data. This is a representation of the CPU usage of the 8 threads during some Newton iterates. All threads are used while perturbing. In between the perturbations is the line search, where there is some but not much concurrency. The bottle neck units are the ones that take longer time to evaluate in this figure.



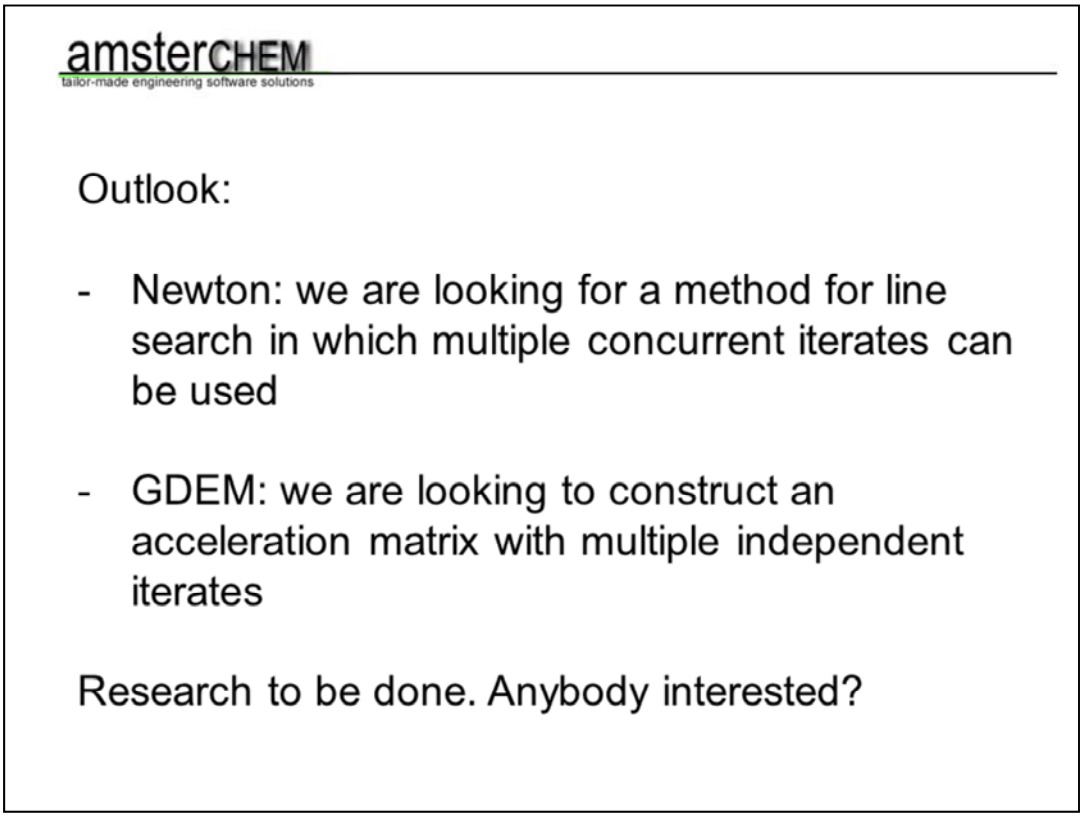
To conclude: classical sequential flowsheeting becomes dependency driven flowsheeting if you want to distribute the load over several nodes.

The gain is higher, and I cannot stress this enough, where you need it. With complex flowsheets.

The gain is significant for Newton iterations and parametric studies.

For simple flowsheets the gain may be not so great yet, but we are still working on improvements.

And a conclusion for COFE 3 vs COFE 2 is that smarter Jacobian evaluations lead to fewer unit operation evaluations, but this is independent of concurrency of course.

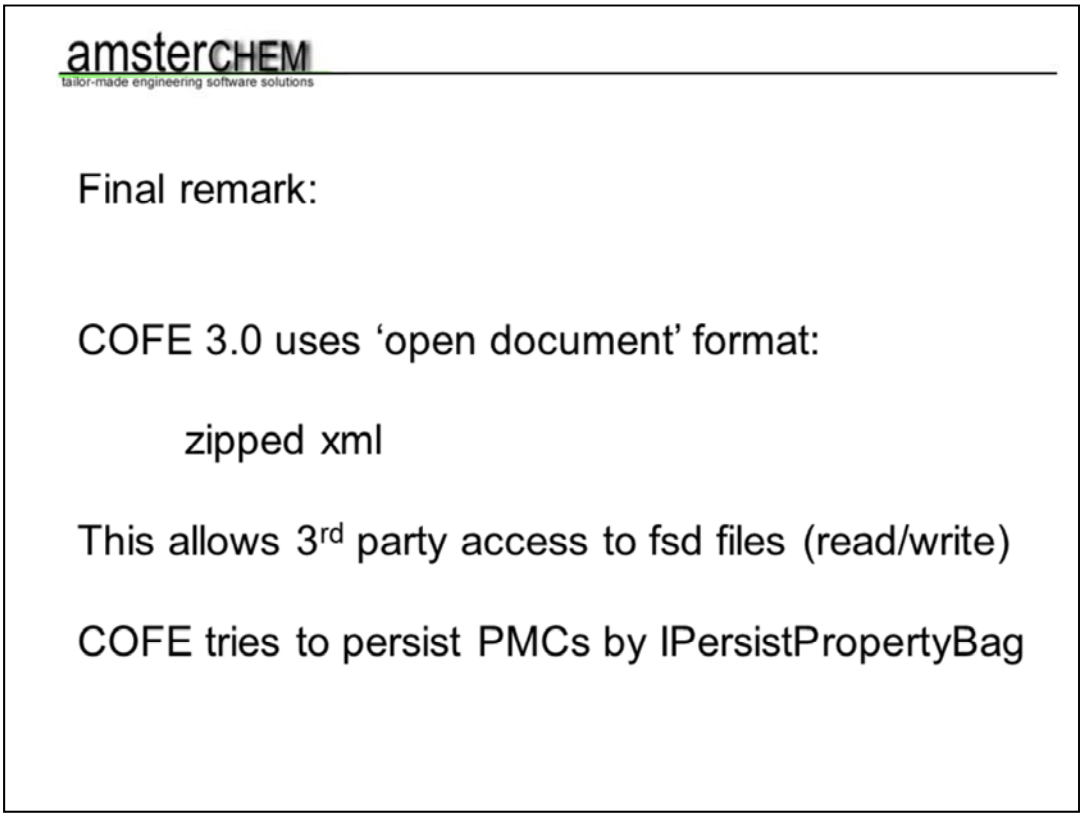


We do want to aim for improved performance also during function evaluations:

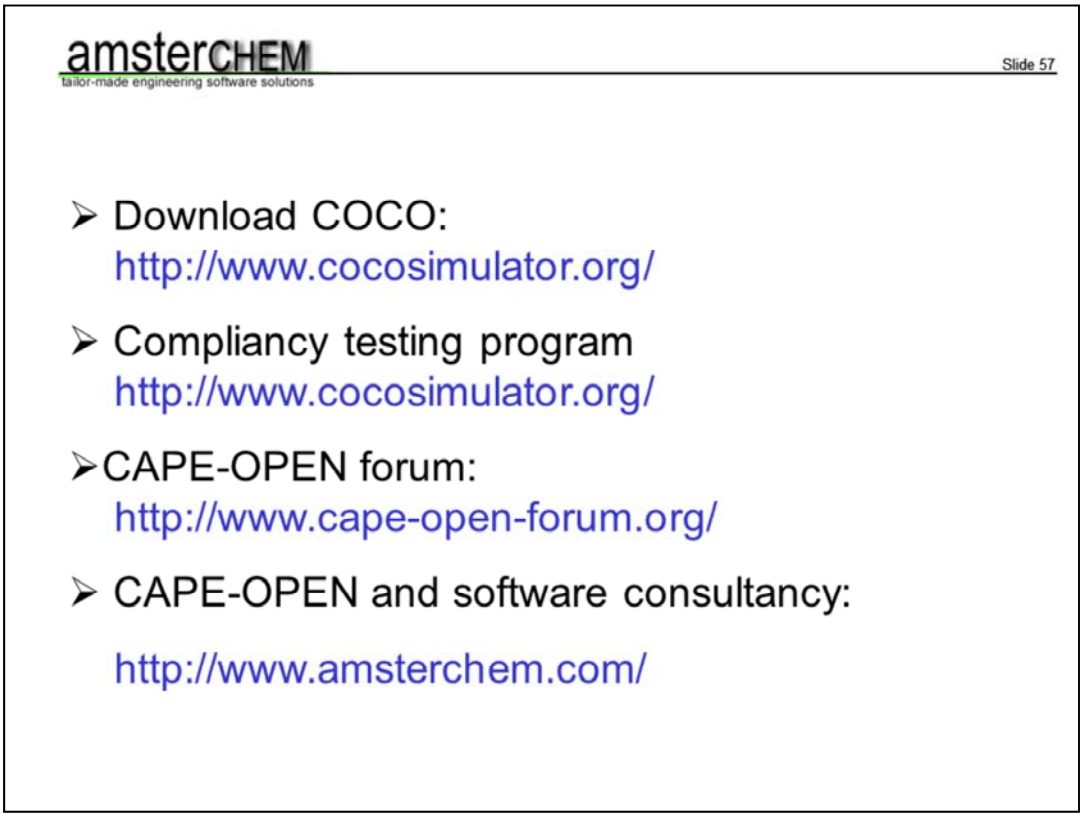
We are looking for a way to use multiple concurrent function evalutions in the Newton line search.

Also the General Dominant Eigenvalue Method may be promising in case we can use independent concurrent function evaluations to construct the predictor matrix.

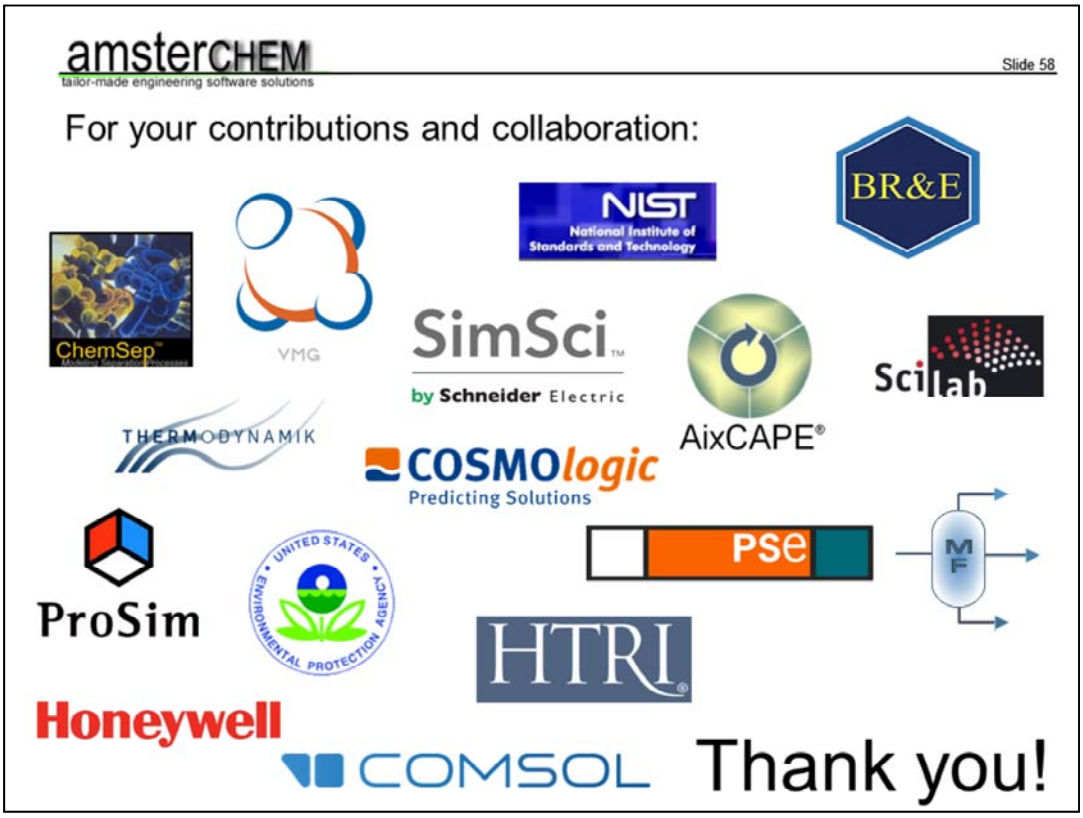
Research is to be done here, and if anybody is interested or has ideas, please contact us.



Finally, unrelated to the remainder of this presentation, COFE 3 uses a new file format: zipped XML. This allows third party applications to directly analyse and manipulate COFE flowsheet files. For PMC developers this implies that it may help to provide an interface to persistence that allows for storage of data in human readable chunks. The Property Bag interface is suitable for that, and I hope some PME vendors will provide support for that.



As always, COCO is free. Feel free to download. If you are a CAPE-OPEN developers we encourage you to take part in the compliancy testing programme at cocosimulator.org. And of course the two other development resources should be mentioned: the CAPE-OPEN forum, and myself at AmsterCHEM.



Thank you for your attention. I would also like to thank all software vendors that help develop COCO into what is it today, either by cooperations, or by interop testing licenses. Thank you indeed!