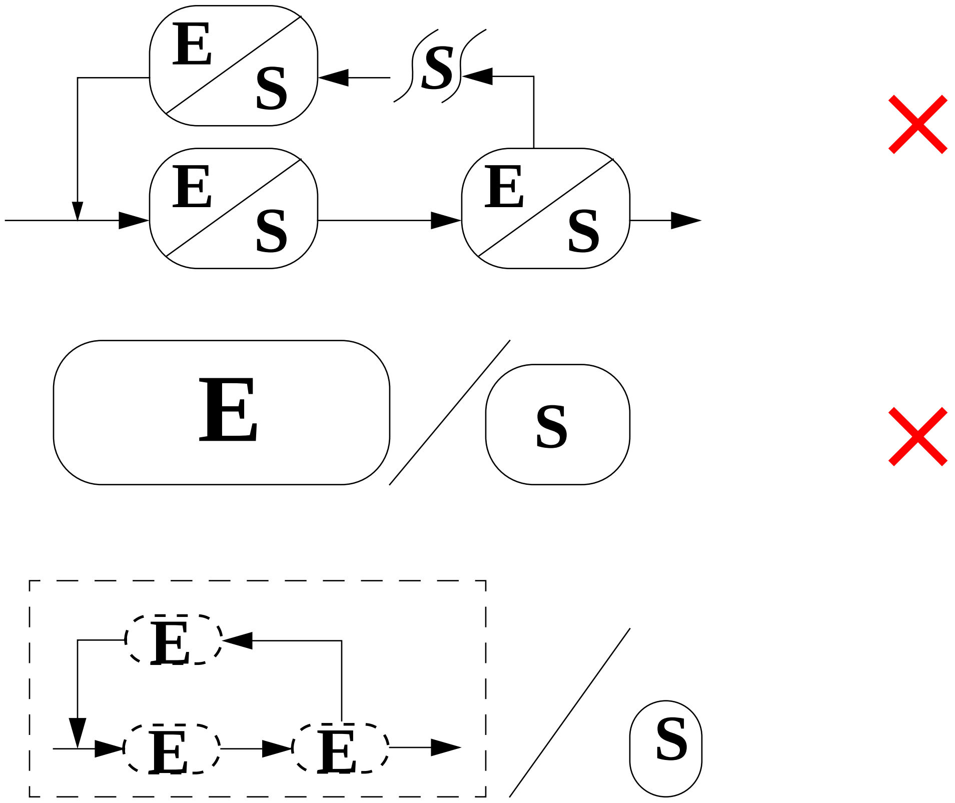
# The Importance of Directed Acyclic Graphs in Glass-box, Equation-based Modeling

November 19, 1997

(10X reduction in compiling times: ~~100~~ seconds ~~30~~ 10?)

# Q: How can we create 1000s of equations describing process flowsheet physics?

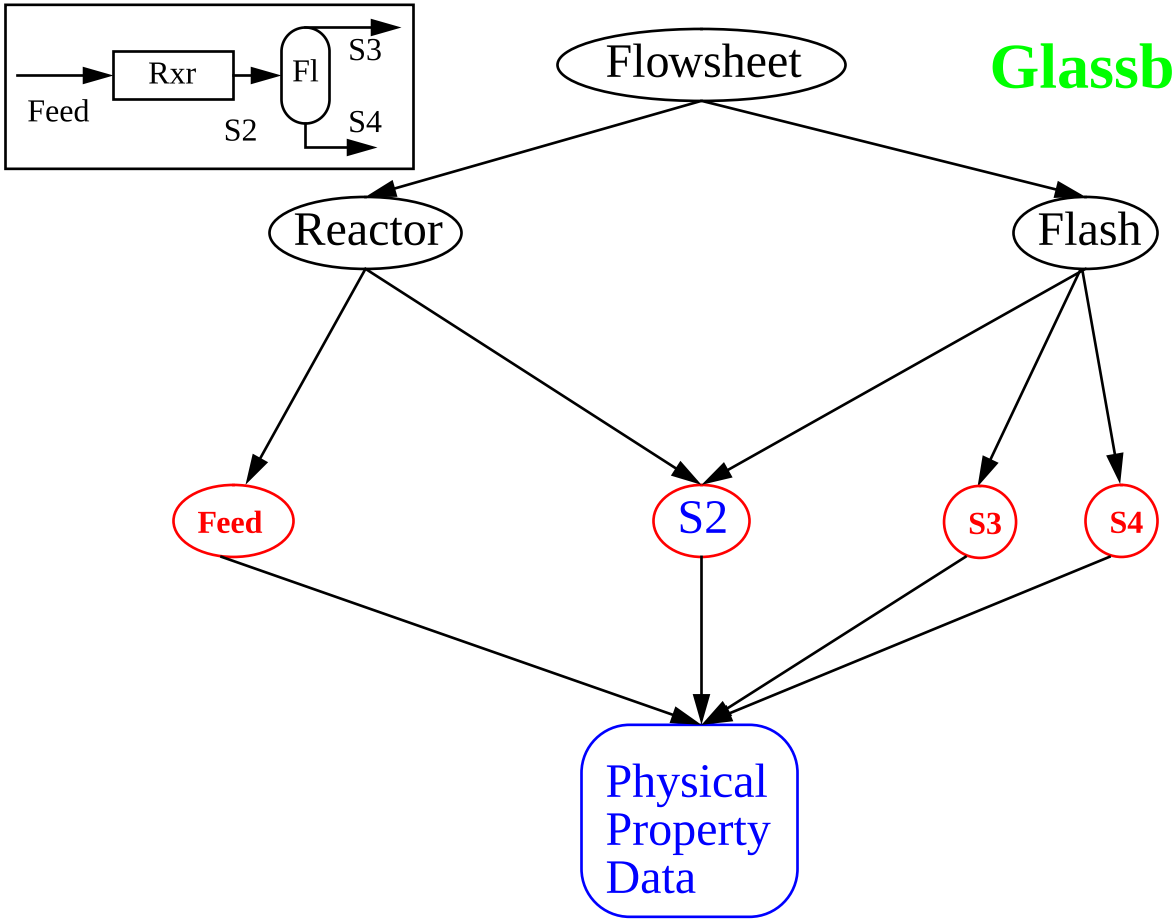


→ How do we get back tailored module efficiency?

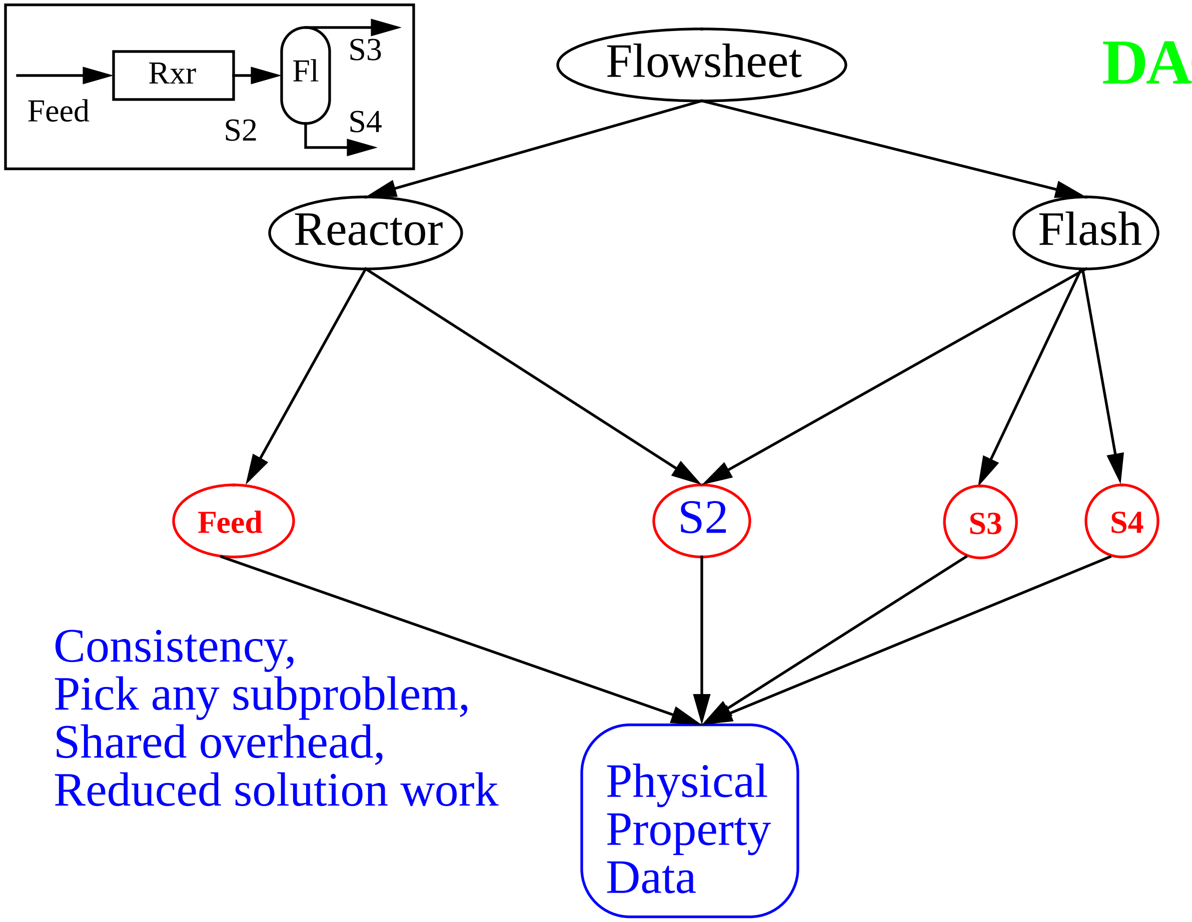
# Hypothesis: Passing objects makes reuse easier in glassbox equation-based modeling.

- Appears in limited tests to be good for users. (thesis)
- Definitely good for the computer. (today's talk)

# Glassbox DAG

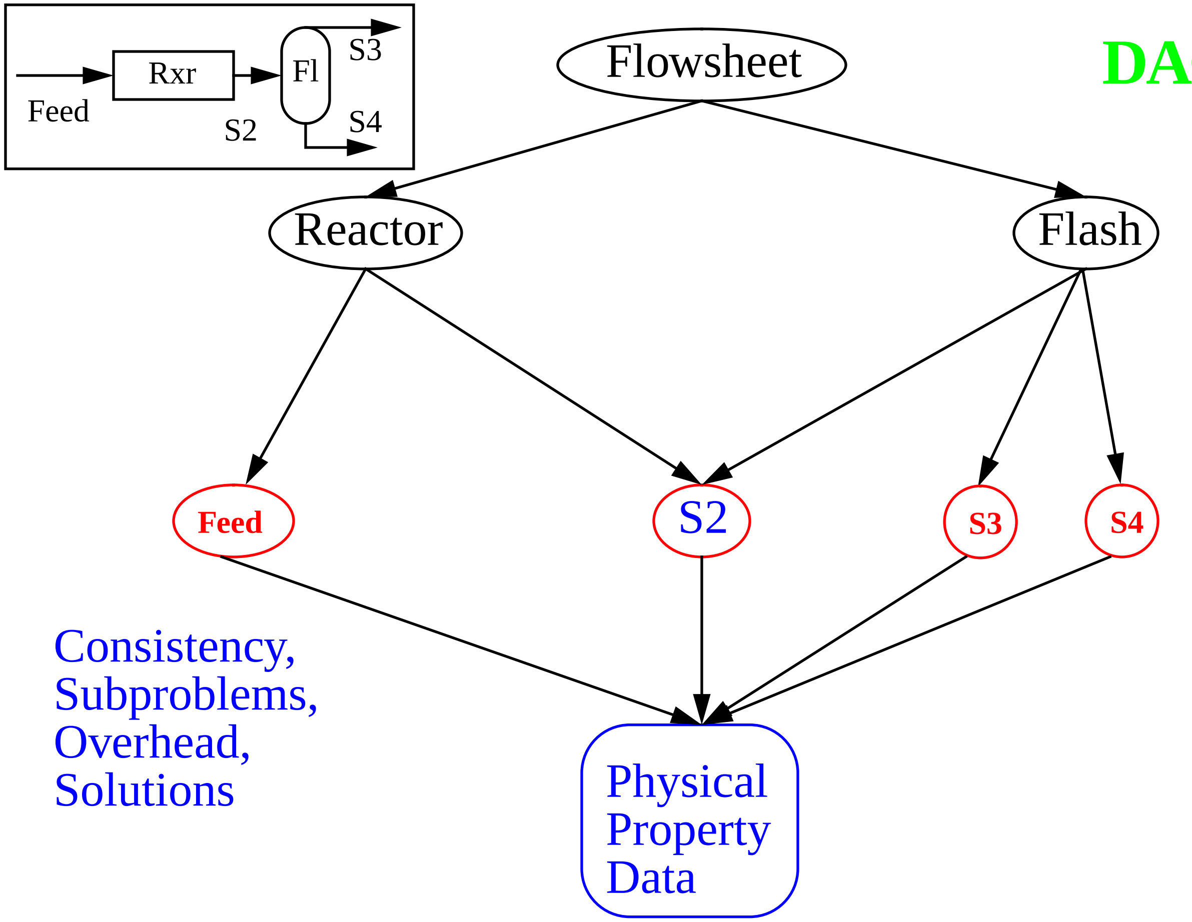


# DAG



Consistency,
Pick any subproblem,
Shared overhead,
Reduced solution work

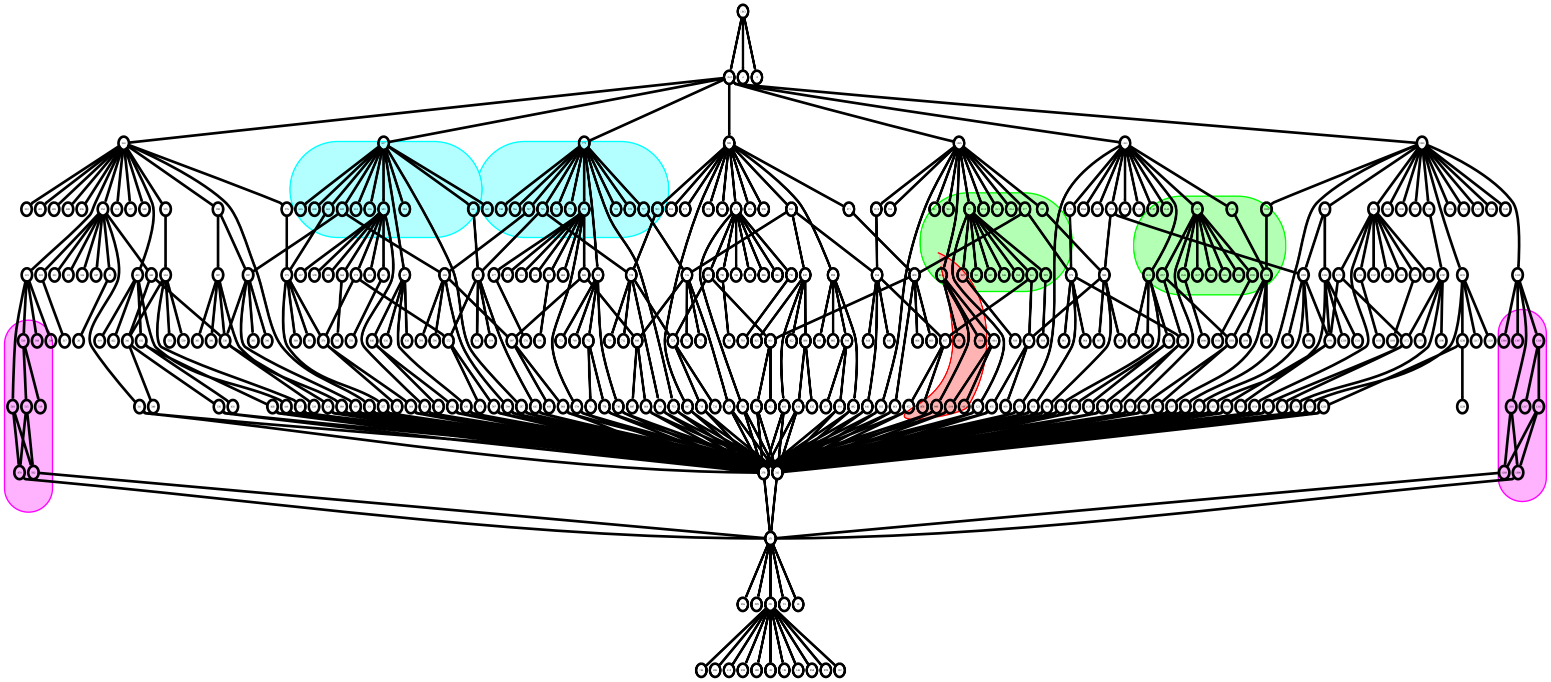
# DAG



Anonymous Stream Subclass

Consistency, Subproblems, Overhead, Solutions

# Very Simple Flowsheet



# Anonymous Subclass =

Formal Class + Constant Values + *Shared Structures Pattern*

13 classes, 26 *anonymous subclasses*, 300+ objects

## Goal:

- Compile each anonymous subclass *once*. (expensive)
- *Share* as needed for repeated structures. (very cheap)

# Intermediate Test (ASCEND IV)

2 Columns, 194 stages, 17,500 equations to solve

| Equations compiled | Compile, seconds | Memory, megabytes | Anonymous Class Detection, seconds |
|---|---|---|---|
| Unshared<br>17,500 | 114 | 26 | --- |
| Shared<br>118 | 35 | 11 | 2.5 |

Detection cost something like: O(Nodes)

**Scalable!**

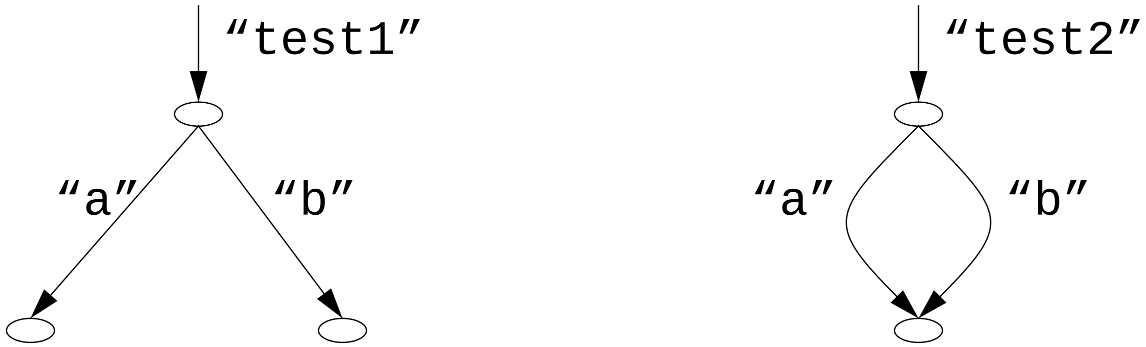
# Conclusions from DAG work

- Exploiting the underlying DAG is important and cheap.
- Concepts useful in any ***hierarchical*** modeling system, any discipline.
- Concept can aid computing on scalar, vector, and distributed hardware.
- Glassbox equation-based systems can attain the same efficiencies as blackbox modular systems.

# Object Passing Sample

```
Property_Data     P(“methanol”, “water”);
Liquid_Stream     S[1..4](P, “UNIFAC”);
Reactor           R1(IN: S[1], OUT: S[2]);
Flash             F1(IN: S[2], OUT: S[3], S[4]);
```

# Substructure is part of Anonymous Subclass



```
TestClass(real_pointer a, real_pointer b) {
    equation: (*a) = (*b);
}

Example() {
    real x,y;
    TestClass test1(&x, &y);
    TestClass test2(&x, &x);
}
```