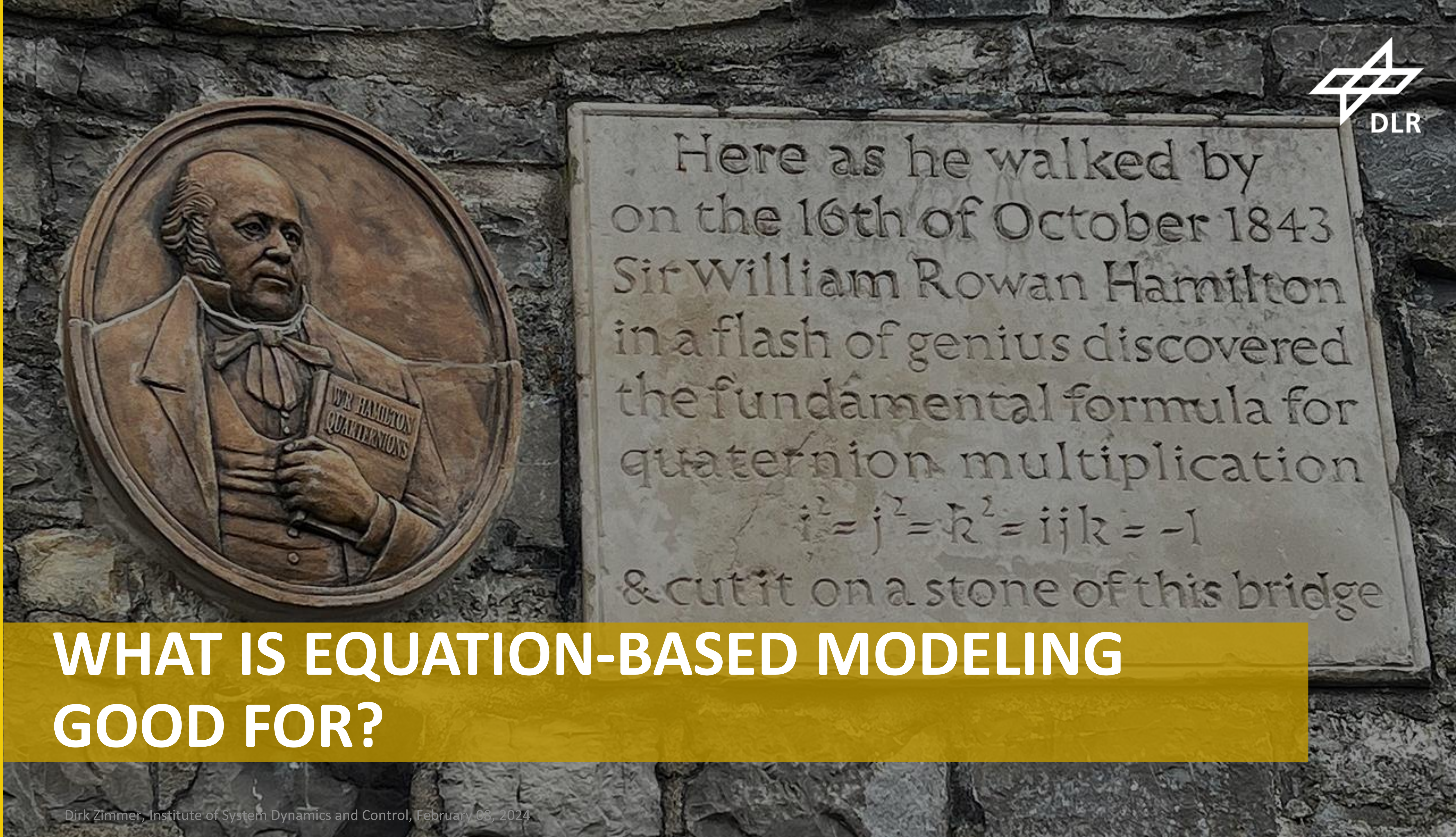
MODELICA LITE
TOWARDS A STRICT, ROBUST AND SCALABLE SUBSET OF MODELICA

Dirk Zimmer,
Institute of System Dynamics and Control

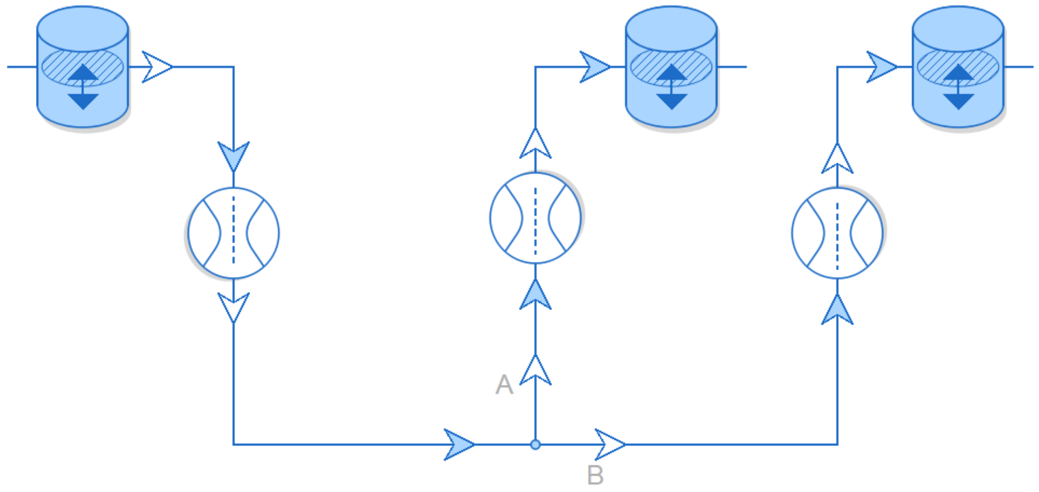
03.02.2025 OpenModelica Workshop



WHAT IS EQUATION-BASED MODELING GOOD FOR?

Object-Oriented Modeling in Classic Programming Languages

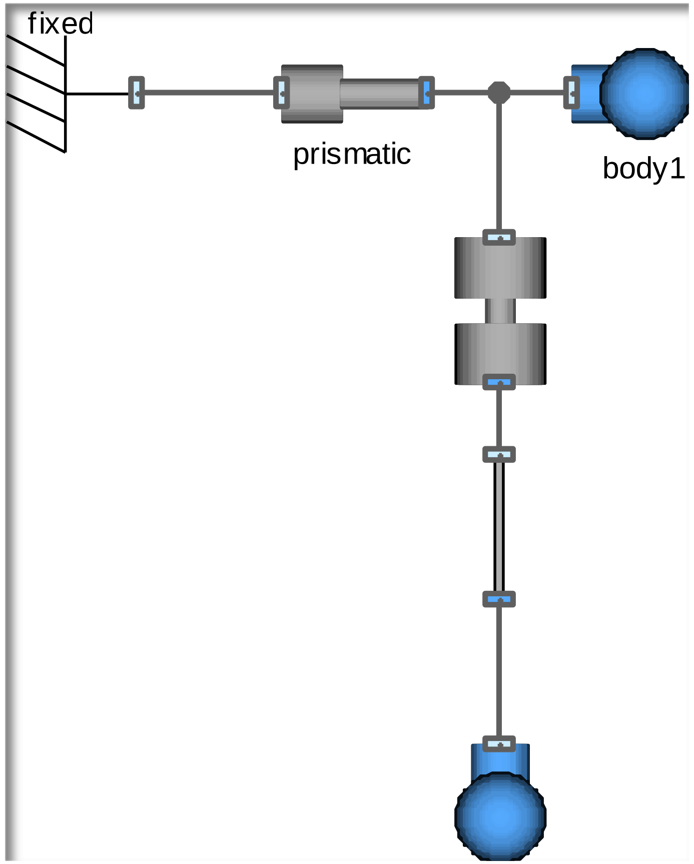
```
class ComVessels : public Component {
public:
  OutTank t1{};
  InTank t2{};
  InTank t3{};
  Splitter s{};
  PressureDrop p1{};
  PressureDrop p2{};
  PressureDrop p3{};

  Connections con {
    Connection{&t1.outlet, &p1.inlet},
    Connection{&p1.outlet, &s.inlet},
    Connection{&s.outlet1, &p2.inlet},
    Connection{&p2.outlet, &t2.inlet},
    Connection{&s.outlet2, &p3.inlet},
    Connection{&p3.inlet, &t3.inlet},
  };
...
```



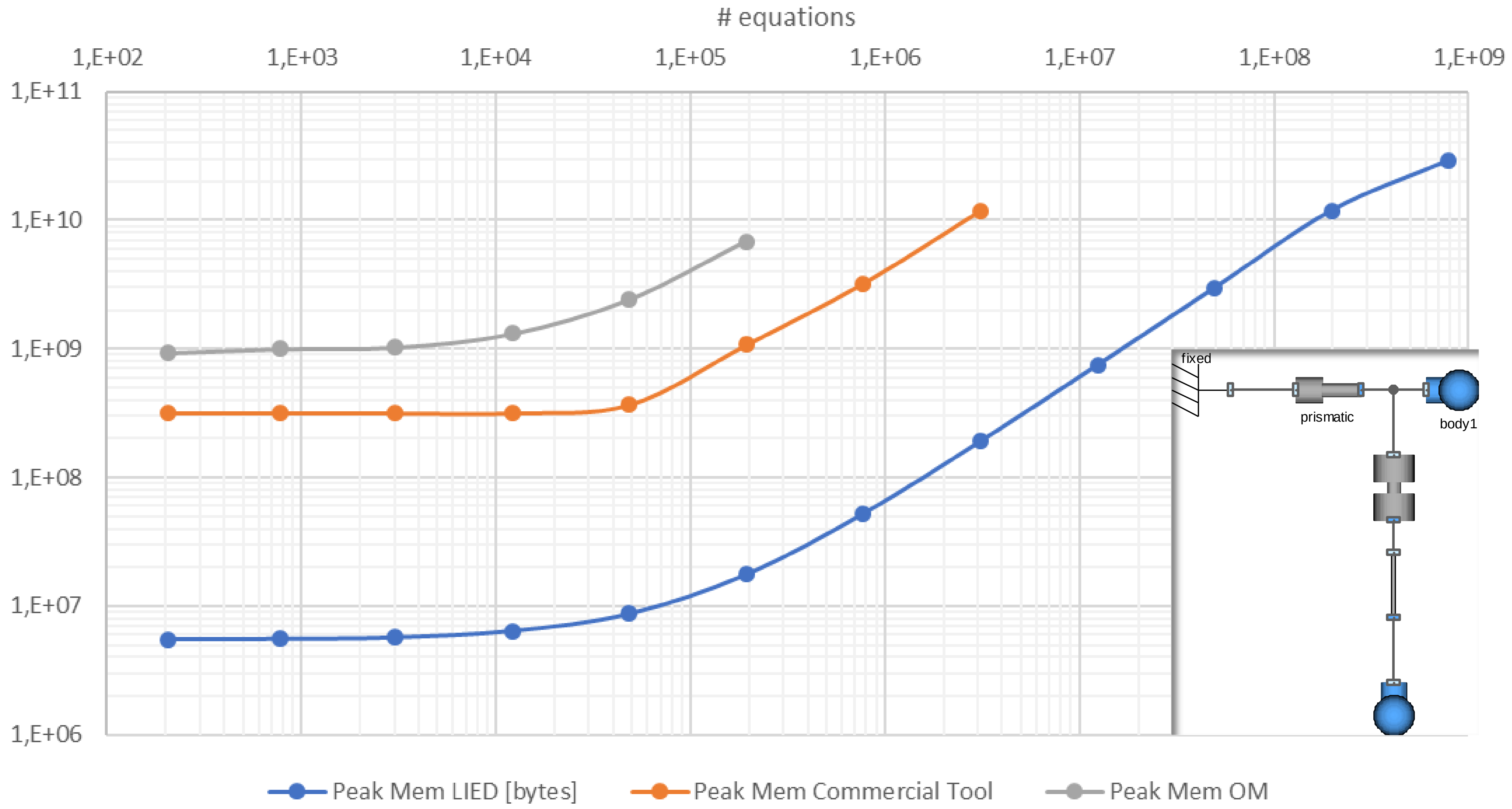

```
…
virtual void metainfo(Meta& meta) override{
    meta.regComp(&t1, "t1: first vessel");
    meta.regComp(&t2, "t2: second vessel");
    meta.regComp(&s,  "s: flow split");
    meta.regComp(&t3, "t3: third vessel");
    meta.regComp(&p1, "p1: first valve");
    meta.regComp(&p2, "p2: 2nd valve");
    meta.regComp(&p3, "p3: third valve");
   };
};
```

Object-Oriented Modeling in Classic Programming Languages

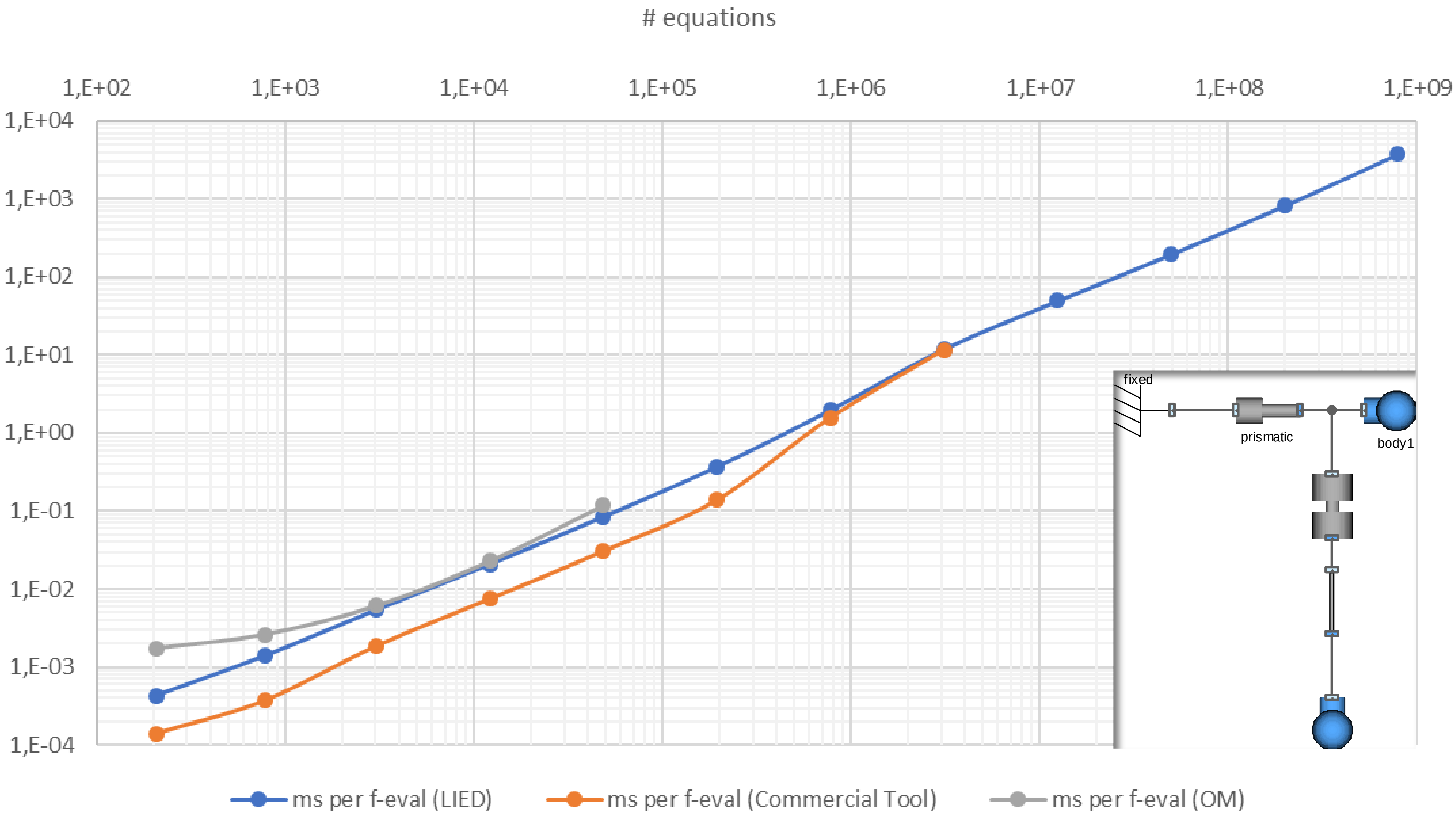
```
class CraneCrab : public Component  {
public:
    Fixed fixed{};
    PrismaticJoint prismatic1{Eigen::Vector2d{1.0,0}};
    Body body1{1.0,0.1};
    RevoluteJoint revolute2{};
    FixedTranslation rod2{Eigen::Vector2d{0.5,1.5}};
    Body body2{0.5,0.05};
    Connections con {
        Connection{&fixed.flangeOn,&prismatic1.flangeTo},
        Connection{&prismatic1.flangeOn,&body1.flangeTo},
        Connection{&prismatic1.flangeOn,&revolute2.flangeTo},
        Connection{&revolute2.flangeOn,&rod2.flangeTo},
        Connection{&rod2.flangeOn,&body2.flangeTo}
    };

    […]

};
```



Superior Scaling with Memory

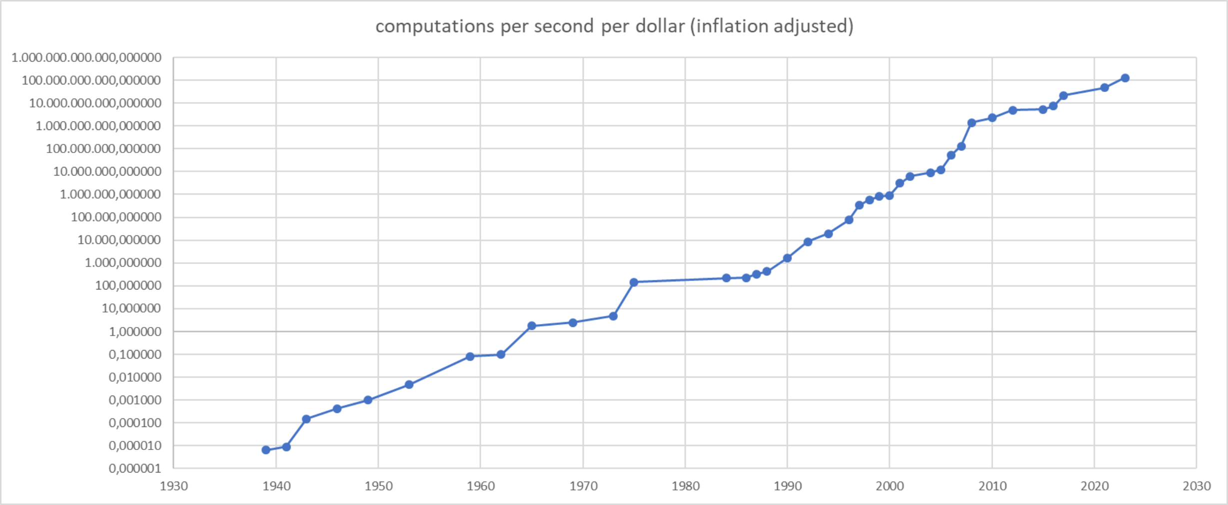


Acceptable Computational Efficiency



So why go for an equation-based language?

- Is it really worth all the effort of specification, compiler building, etc. Is the effort not better invested in implementing software libraries directly?
- There are two strong point in favor of equation-based languages:
  - Social
  - Technical



Social: Equation are useful to people

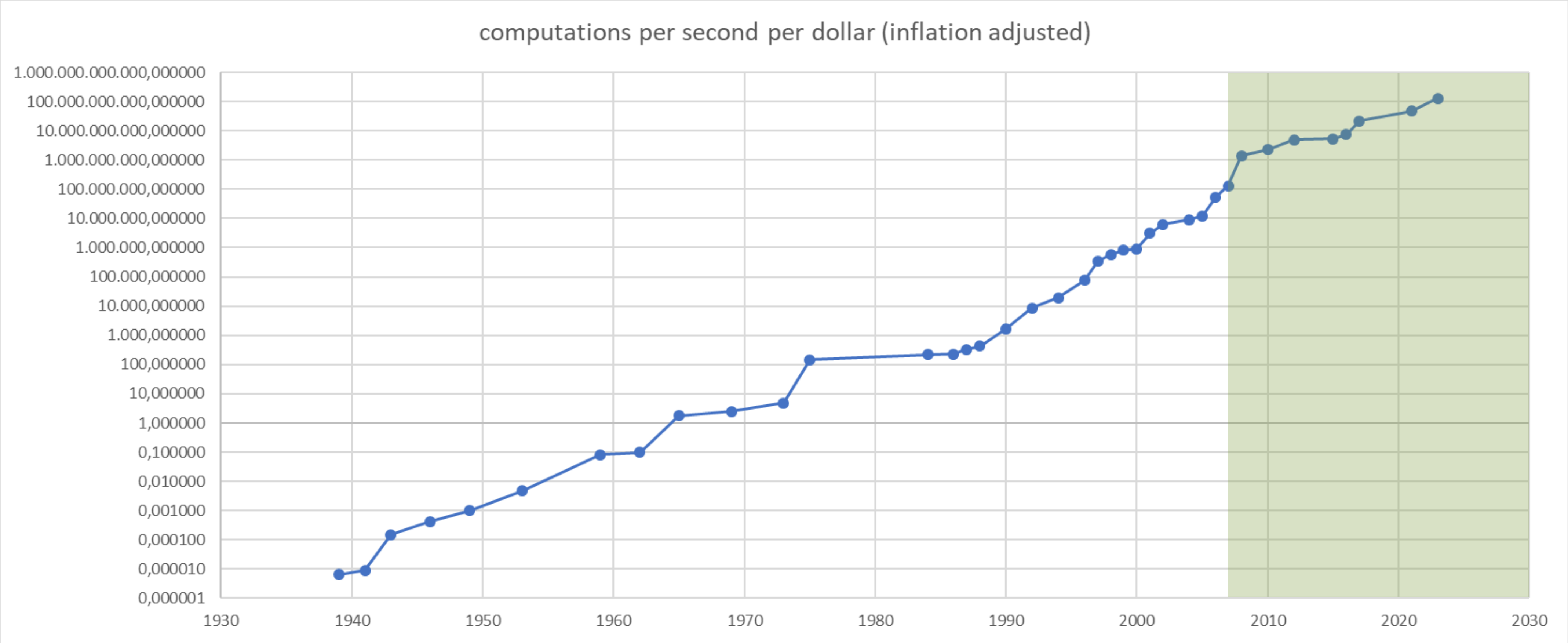
- Modelica is appealing to mathematical experts in engineering domains.
  - There is a demand of roughly 10 000s people world-wide.
  - This is a niche but not an unimportant one.
  - We succeed in this niche.

- Equations and corresponding models are dominant in technical and engineering education.
  - This a potential “market” of 10 000 000s people
  - This is almost mainstream
  - We mostly fail serving this market.

10000s

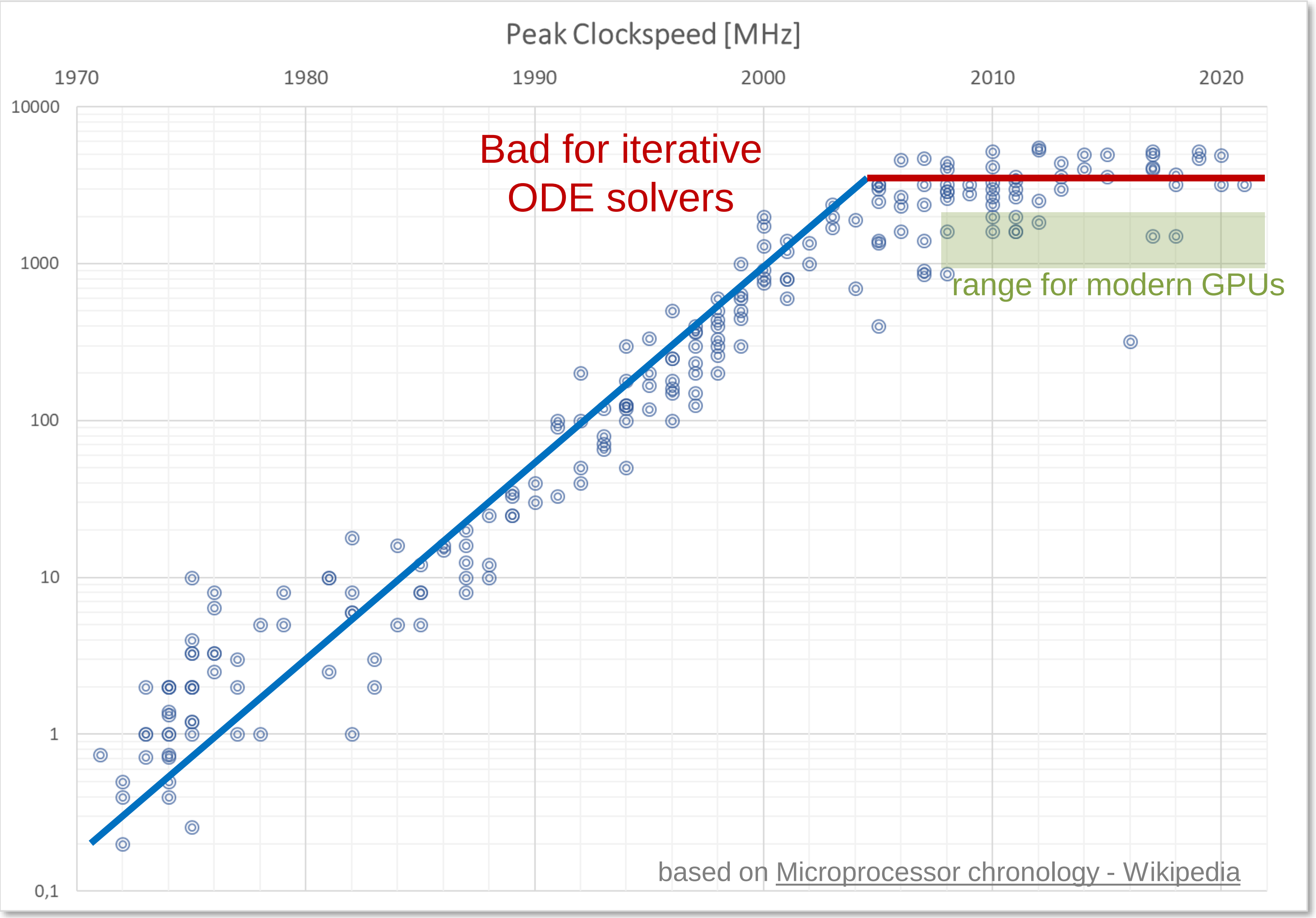
10000000s

Technical: Equations are oblivious to compute architecture



- Data and metric from Kurzweil 2023: “The Singularity is nearer”
- Relentless progress in the most relevant metric. Prime example of true human ingenuity.
- From 2008 on, GPUs dominate this chart

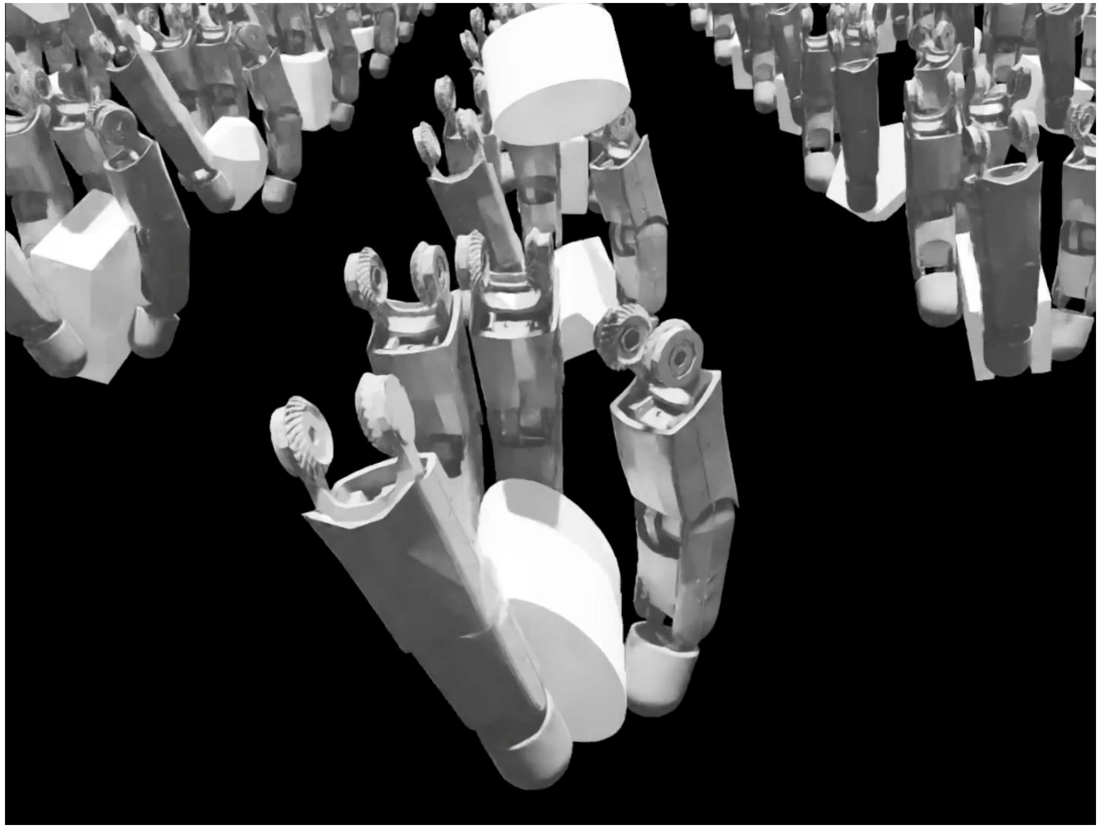
Technical: Equations are oblivious to compute architecture

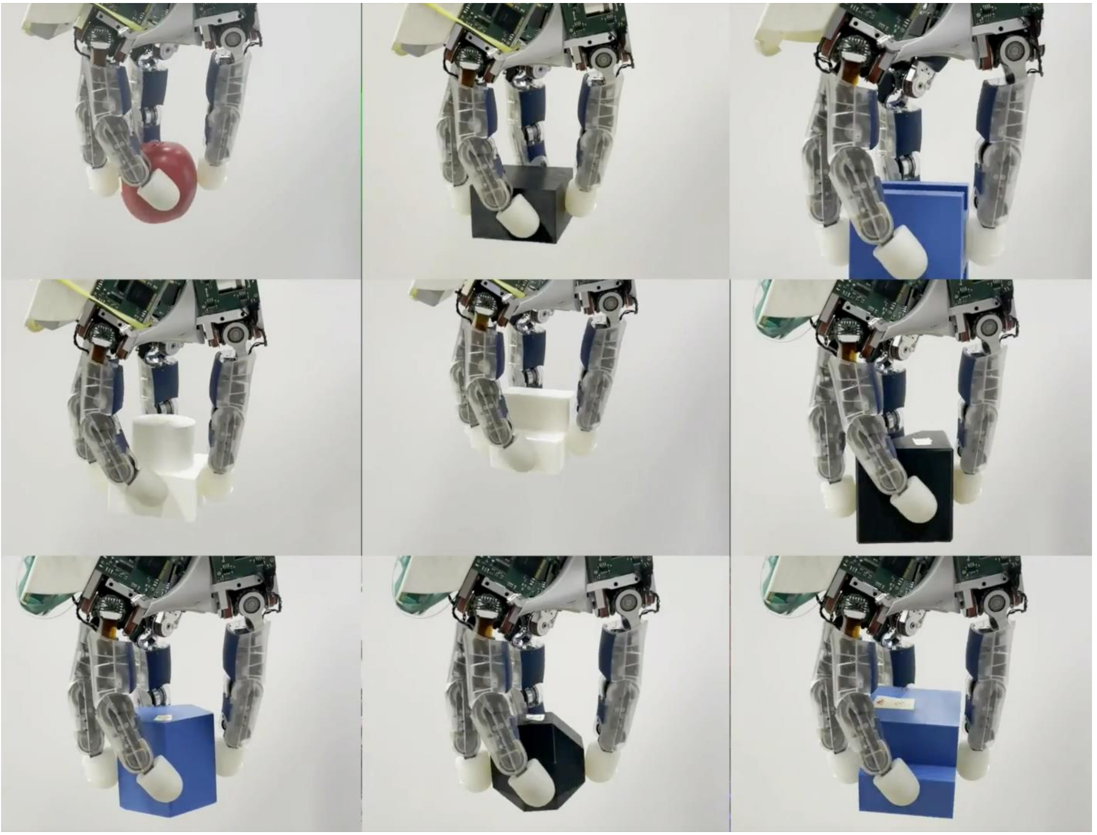


- Physics operates at high frequencies.
- Having 3-8 orders of magnitude between dynamics of interest and internal dynamics is normal.
- CPU frequencies are however not increasing since 20 years.
- Availability of compute changes from
  - Low latency architectures (CPU)
  - High throughput architectures (GPU)
- We must adapt, whether we like it or not!

Technical: Equation are oblivious to compute architecture

- >95% of all simulations in the near future will be performed on high-throughput architectures. Main purpose is AI training and optimization.
- 5h Training for the example below. One GPU. $1.20 electricity bill.





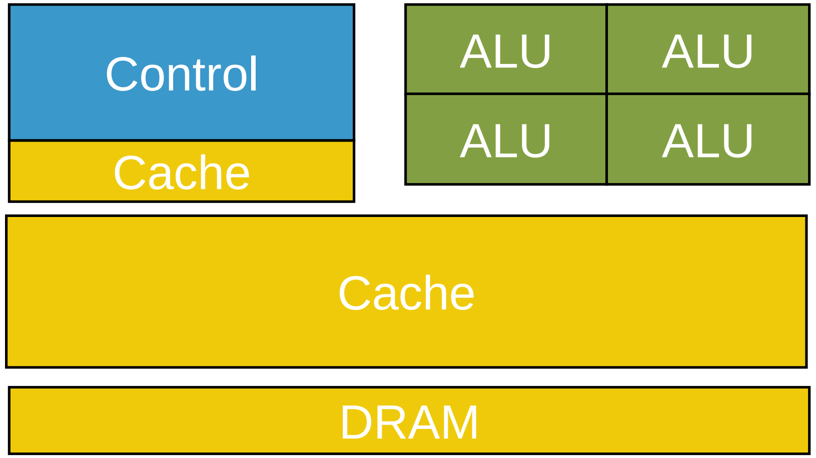
Röstel, Lennart, et al. "Estimator-coupled reinforcement learning for robust purely tactile in-hand manipulation." *2023 IEEE-RAS 22nd International Conference on Humanoid Robots (Humanoids)*. IEEE, 2023.

[3]

Technical: Equation are oblivious to compute architecture

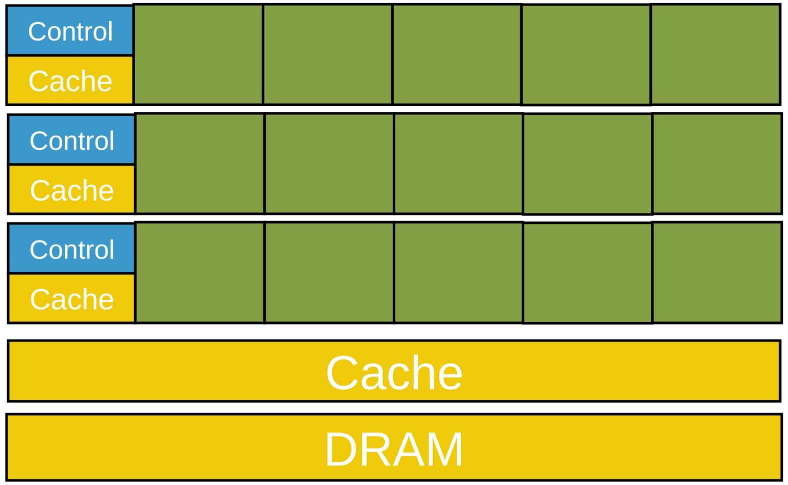
- Code generation for low-latency:
  - This works nice in practice.
  - Dealing with high frequency is often performed using implicit solvers for stiff systems.
  - Symbolic transformation helps as well (Flattening is overrated)

<5%



- Code generation for high-throughput.
  - I have seen no real good example. We fail albeit this should be our strength.
  - Multi-derivative methods are not exploited
  - Modular code for kernels are not supported

>95%



Summary: we fail to live up to our potential

from *Gulliver's Travels: Coloured Picture Book for the Nursery*, Thomas Nelson and Sons, London, Edinburgh, New York, 1883
wikimedia commons

- We have a user group of 10s of thousands but it should be 10s of millions
- A million times more simulation would soon be running Modelica models if we support GPU simulation for AI training.
- Why do we fail to exploit our potential?
  - We have simply become way to complex…
  - The learning hurdle is very steep.
  - So many specifics stipulate one (very complicated) way of code generation instead of upholding obliviousness of the compute architecture.
  - Hence we are tied down like Gulliver by a thousand strings.

Summary: we fail to live up to our potential

- I estimate that writing a Go compiler is 10x less complex than writing a Modelica compiler + IDE
- I estimate that there are 1000 times more Go users than Modelica users
- Our complexity per user ratio is 10000x times worse than Go.

➔ Our modus operandi is barely sustainable

➔ simple and strict languages have an appeal and a viable market (see SQL_lite)

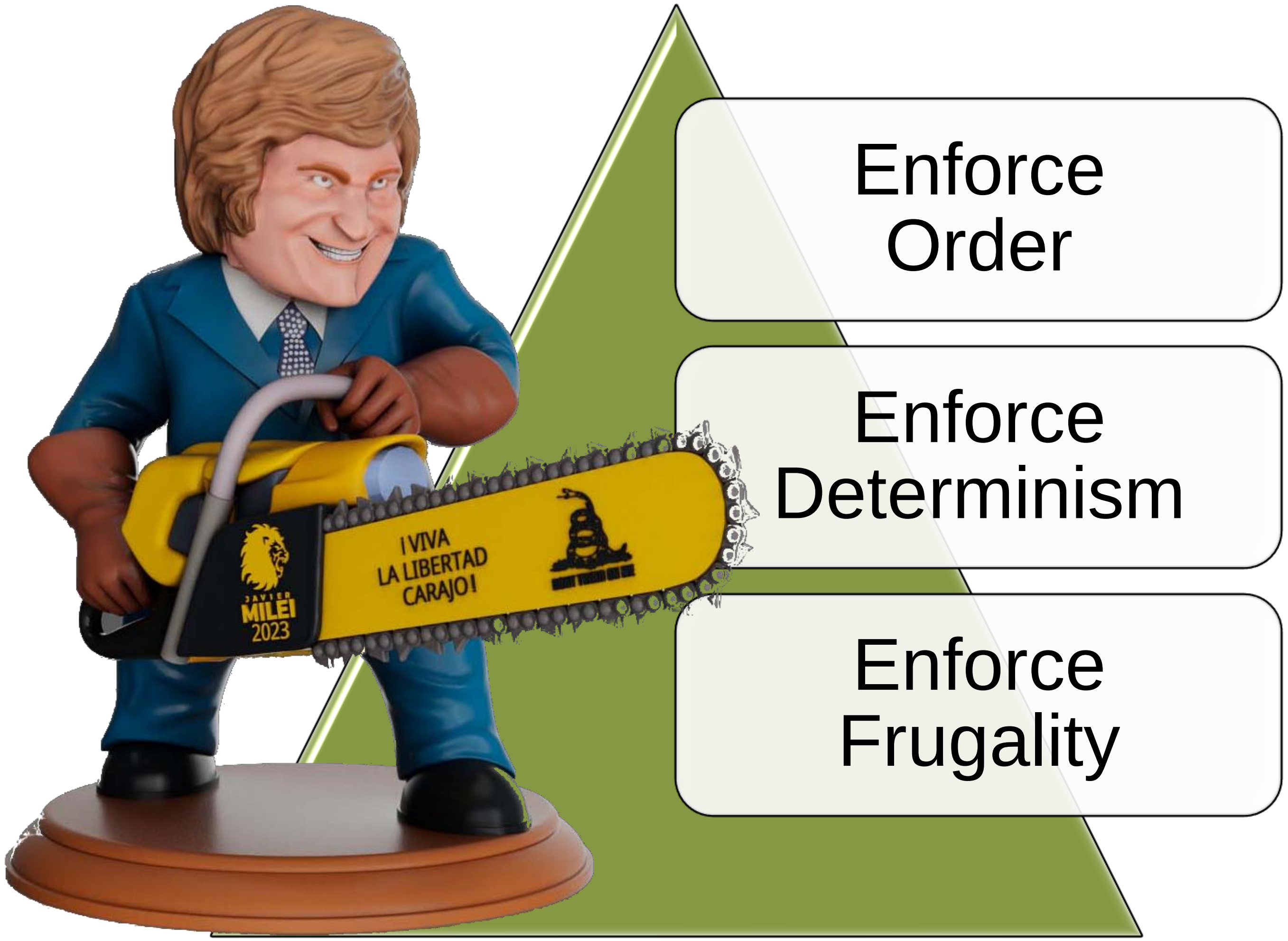
from *Gulliver's Travels: Coloured Picture Book for the Nursery*, Thomas Nelson and Sons, London, Edinburgh, New York, 1883
wikimedia commons

Modelica .Lite

CUTTING DOWN COMPLEXITY

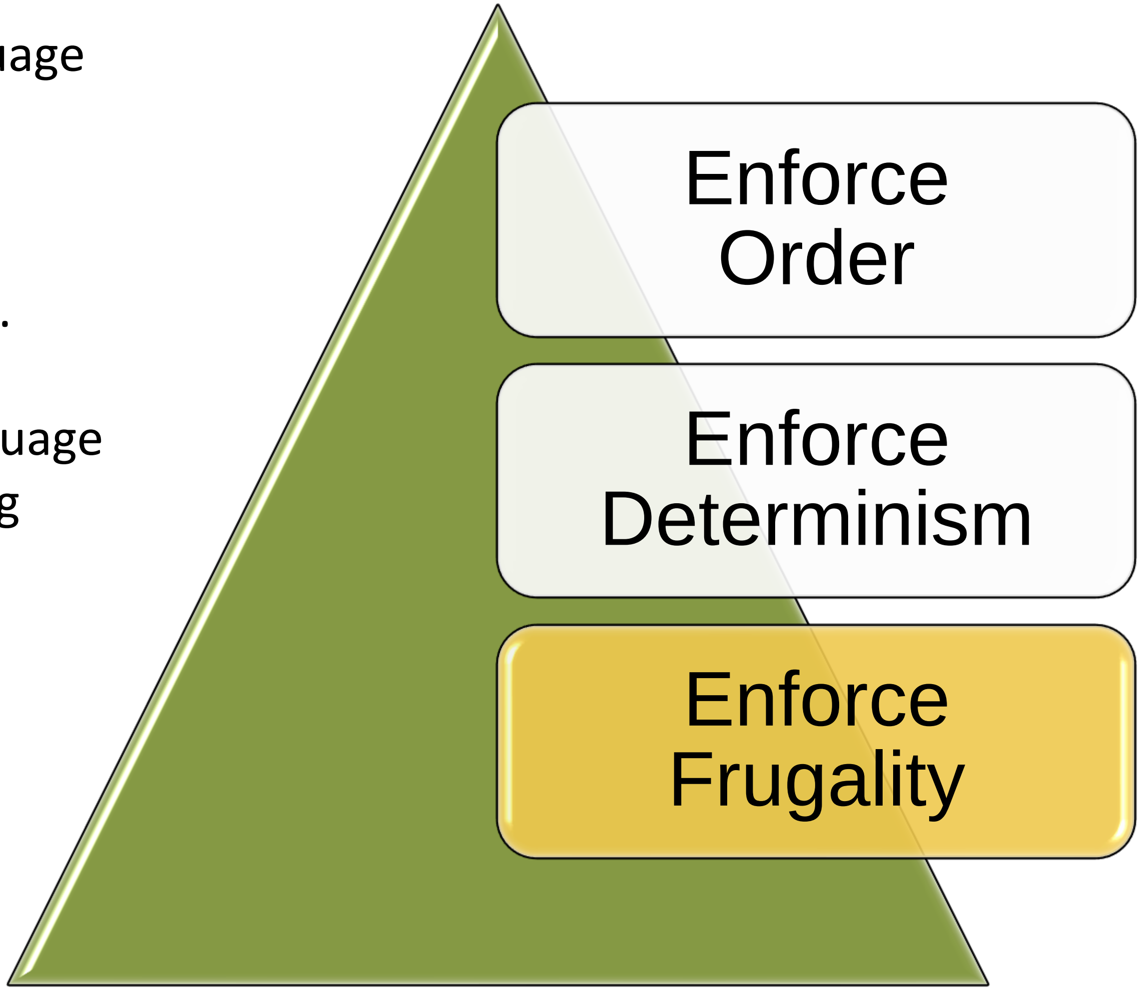
ModelicaLite: Easy to learn, Easy to process

>20x complexity reduction



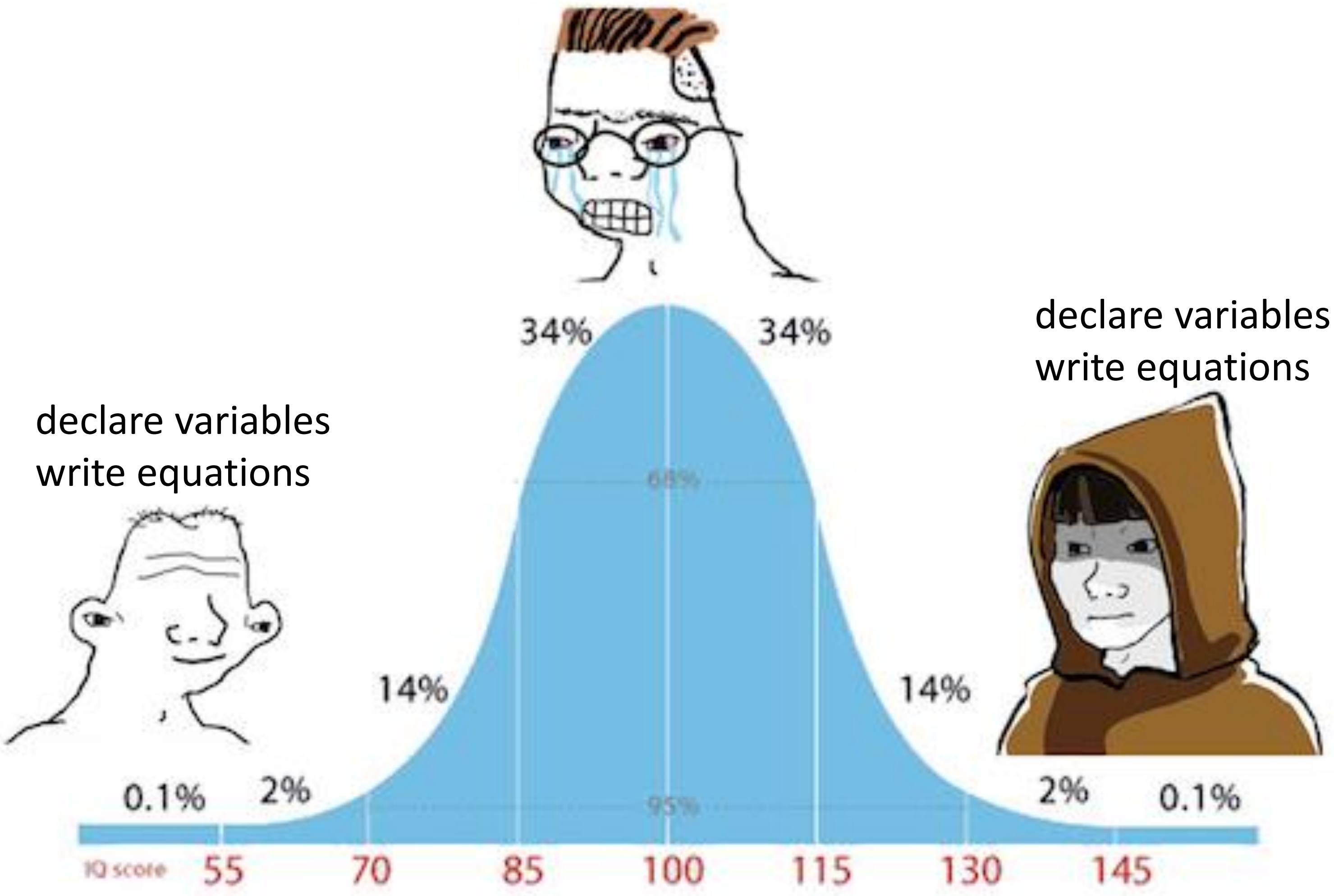
ModelicaLite: Enforce Frugality

- Modelica v1.4 is a very well designed language
- We can renew from within!
- I see no need to start over, like with Modia.
- ModelicaLite will be technically a new language but so that it is 100% compatible to existing compilers.
- ModelicaLite libraries will all run in OpenModelica.
- We can thus also reuse existing IDEs.
- It will require new libraries though.



ModelicaLite: Enforce Frugality

connection.branch; instream(h_outflow); homothopy(); fixed=false;
expandable connectors; operator overload; constrainedBy, inline



declare variables
write equations

declare variables
write equations

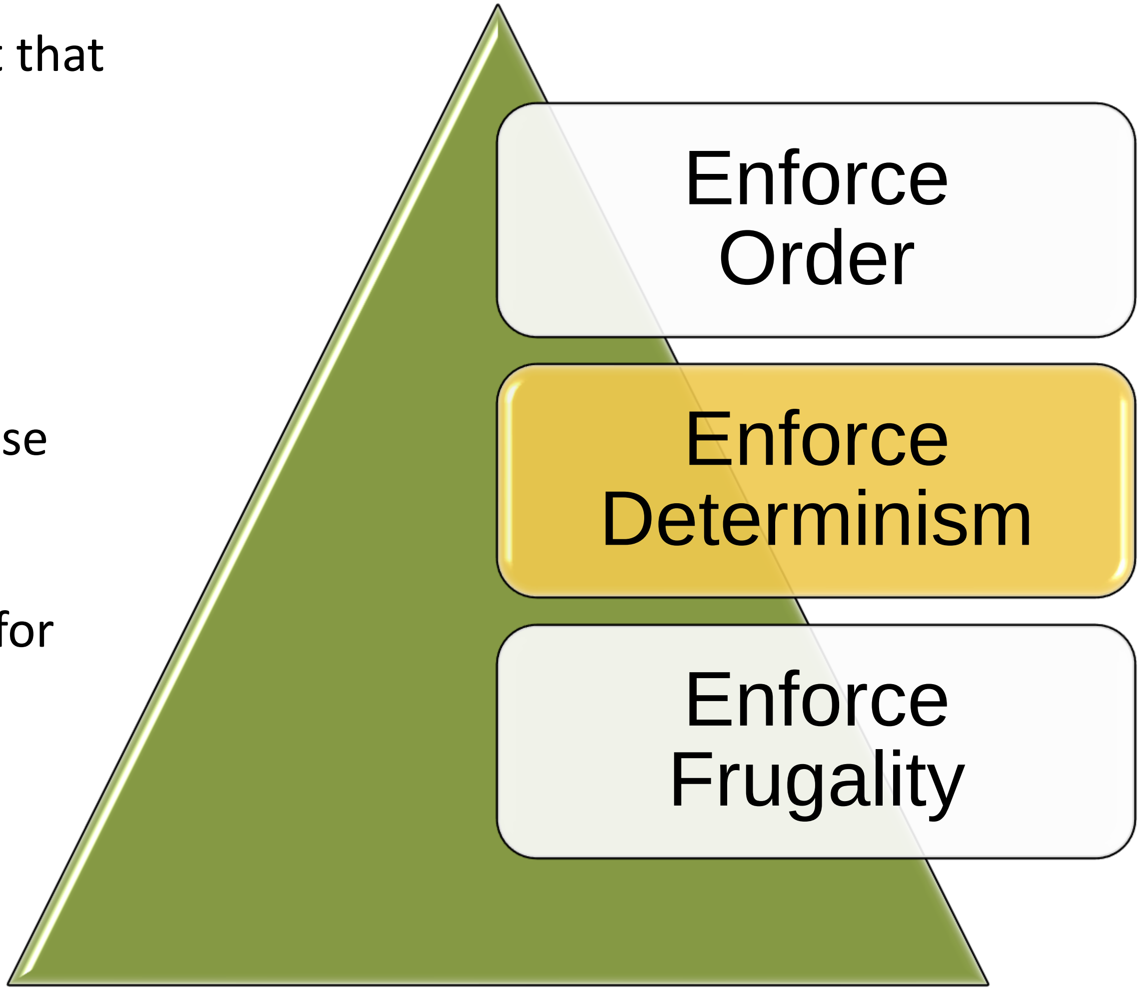
ModelicaLite: Enforce Determinism

- A lot of complexity originates from the fact that arbitrary hybrid non-linear DAEs are an extremely complex problem class.

$$0 = F(x, \dot{x}, h, h', u, t)$$

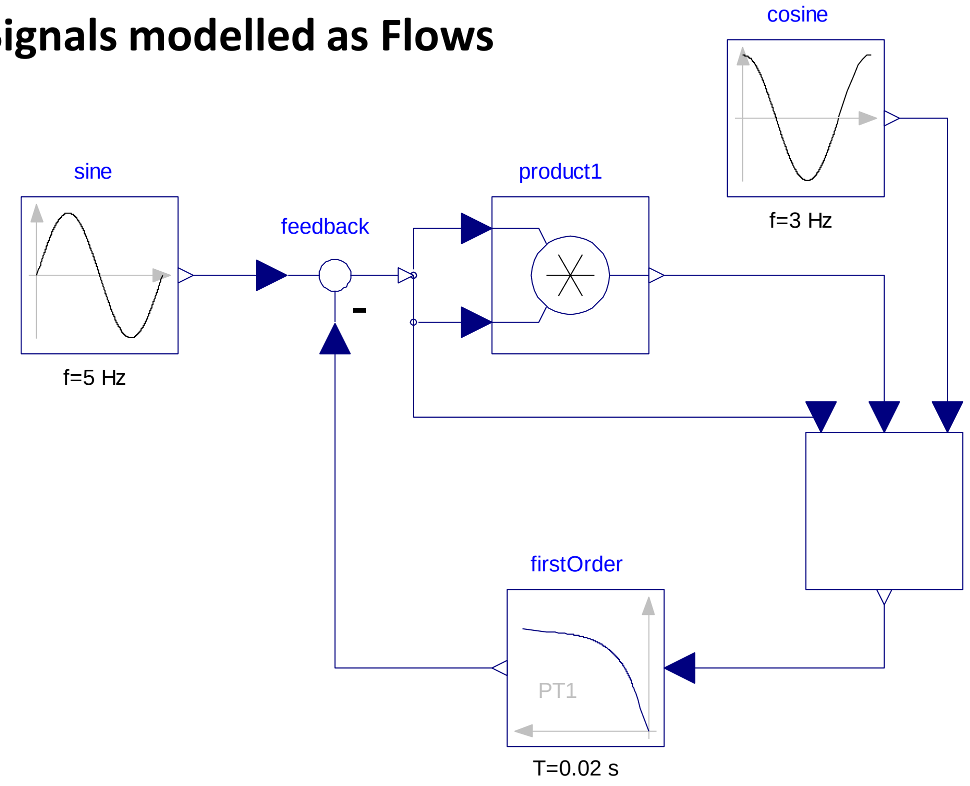
- Even in Modelica, it is unclear which of these problems are actually solvable.

- This full generality is however not needed for >90% of the applications.



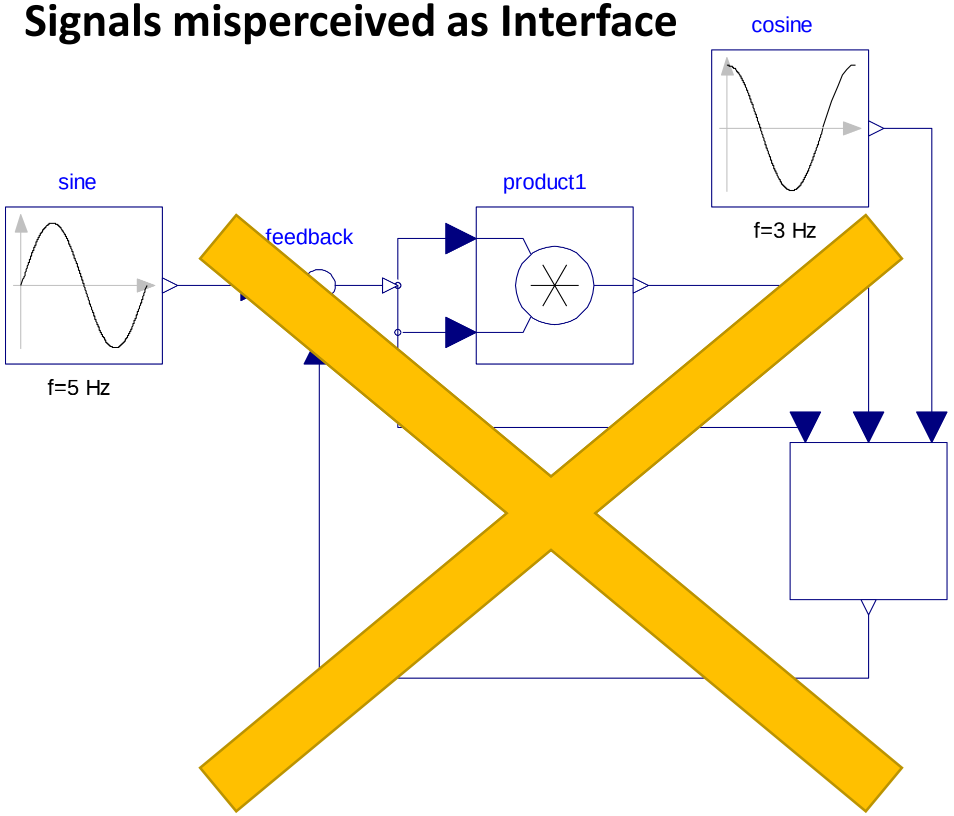
ModelicaLite: Enforce Determinism for Signal Flows

Signals modelled as Flows



- Local analysis suffices
- Deterministic (unique solution)
- Information complete

Signals misperceived as Interface

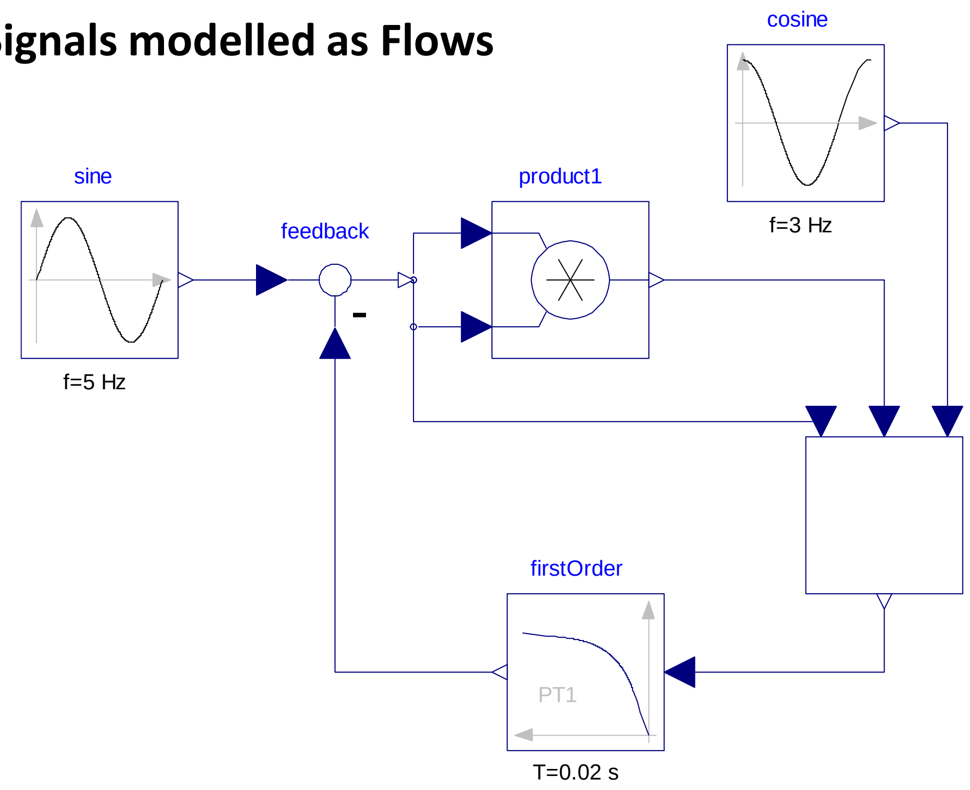


- Global analysis necessary
- Non-Deterministic
- Information incomplete
  (many constructs attempt (but fail) to fix this
  fixed=false, homothopy, etc.)

ModelicaLite: Enforce Determinism for Signal Flows

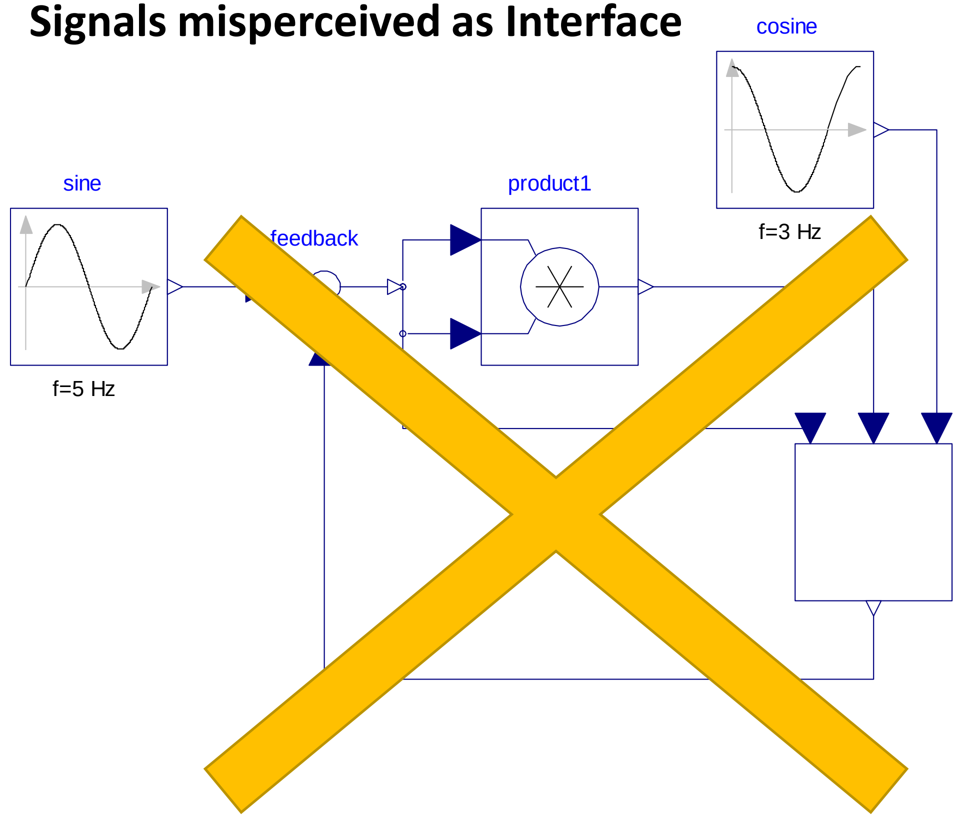
DLR

Signals modelled as Flows



Knowing

Signals misperceived as Interface



Guessing

ModelicaLite: Enforce Determinism for Physical Systems

- Signals are insufficient to properly model physical systems
- However, physical systems are only a tiny subset of non-linear DAEs.
- What do we actually need?

- It turns out that physical systems derived from the stationary action principle have very favorable characteristics
  - (semi-) local analysis sufficient
  - Deterministic (unique solution)
  - Information complete

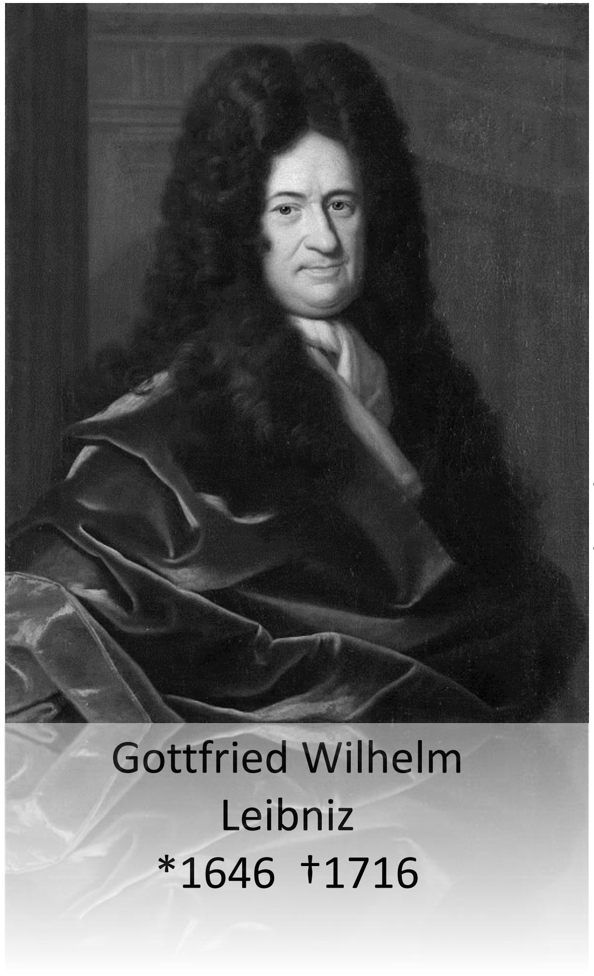
- How powerful and useful is this approach? Let us review physics.

What constitutes classic physics

- Already Leibniz in 1696 argued for an extremal principle:

$$\delta \hat{S} = \delta \int I ds = 0$$

- Physics is described by the transport of impulse I through space s
- The action $\hat{S}$ is the integral over all these transports.
- This does not happen in any arbitrary way but in a (locally) extremal way so that $\delta \hat{S} = 0$



Gottfried Wilhelm Leibniz
*1646 †1716

Source: Wikipedia Commons

A different view on energy

- We observe that the transport of Impulse takes time t
- It is thus attractive to reformulate the integral as an integral over time.

$$\int I ds = \int mv\, ds = \int m \frac{ds}{dt}\, ds = \int m \left(\frac{ds}{dt}\right)^2\, dt = \int mv^2\, dt = \int 2T\, dt$$

- The transformation to a time integral replaces the impulse I with the kinetic energy T as the quantity to be integrated.
- Energy has now been defined without defining force.

The Lagrangian

- In reality, we observe motions that differ from a straight line.
- This is because kinetic energy T can be stored in potentials V
- Also total energy is conserved: $E_{tot} = T + V$ or $2T = T - V + E_{tot}$
- We can now define S:

$$\delta S = \frac{\partial}{\partial q(t)} \int \underbrace{T - V}_{L(q,\dot{q})} dt = 0$$

Lagrangian: $L(q, \dot{q})$

- Please note that the Lagrangian enables powerful idealizations:
  - We can choose our dimensions and coordinates
  - We can select the potentials of interest
  - We can define the aggregate quantities
  - However, we <u>must not neglect</u> the kinetic energy T

From Lagrangian to Hamiltonian

- To define and distribute the resulting differential equations we need the Hamiltonian form:

$$L(q,\dot{q}) \rightarrow H(q,p)$$

- … by introducing the generalized momentum:

$$p_i = \frac{\partial L}{\partial \dot{q}^i}$$

- The Hamiltonian is then expresses the total energy*:

$$H = T + V$$

* under certain conditions

William Rowan Hamilton
*1805 †1865

$$\frac{dq}{dt} = \frac{\partial H}{\partial p}$$

$$\frac{dp}{dt} = -\frac{\partial H}{\partial q}$$

Energy Flows to describe the Hamiltonian

- The Hamiltonian H leads to our pairs of effort and flow that describe the energy flows whose sum is the Hamiltonian

Domain	Translational Mechanics	Rotational Mechanics	Hydraulics	Electrics	Thermal	…
Potential	r	φ	P	V	T	
Flow	f	τ	$\dot{Q}$	i	Q	

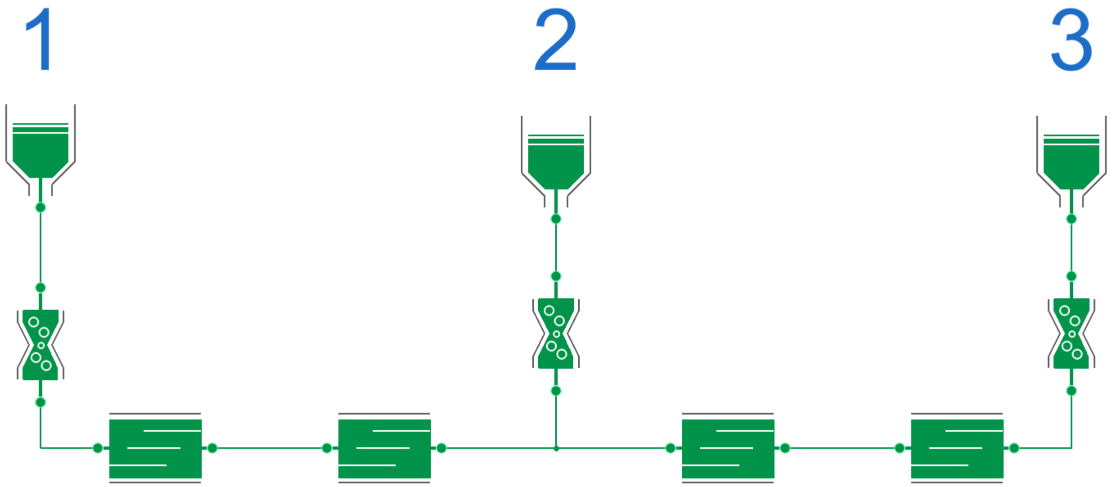
- **But beware!** Whereas each Hamiltonian can be expressed by a sum of energies, not all sums of energies represent a Hamiltonian. This misconception of energies as generic interface leads to non-physical systems and non-deterministic systems!

- Bond-graphs are the prime example of this fallacy.

ModelicaLite: Enforce Determinism for Energy Flows

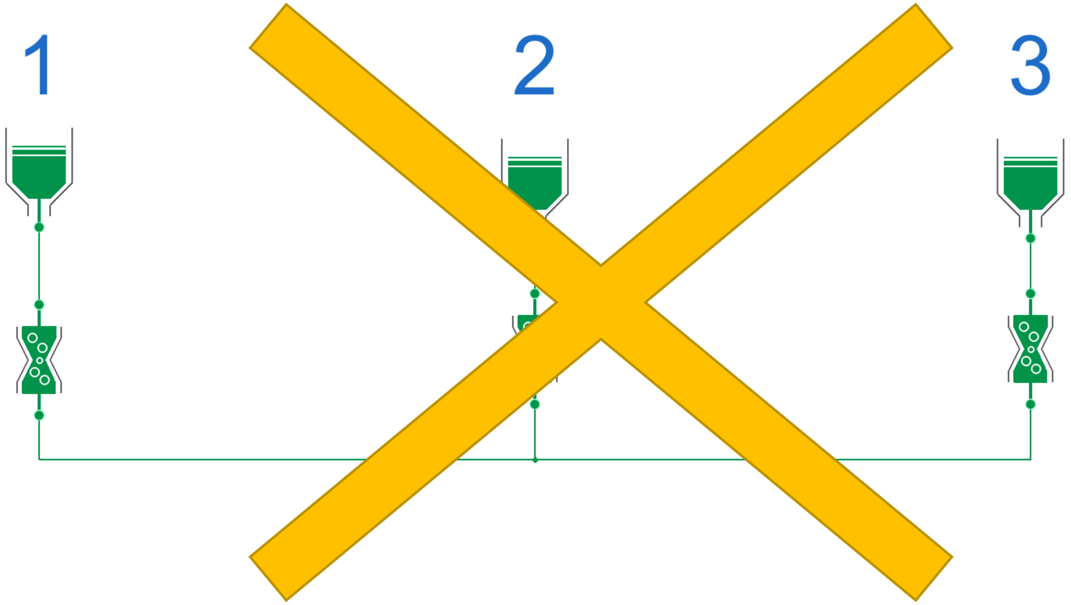
DLR

Energy flows modeling a physical System



- Local analysis suffices
- Deterministic (unique solution)
- Information complete

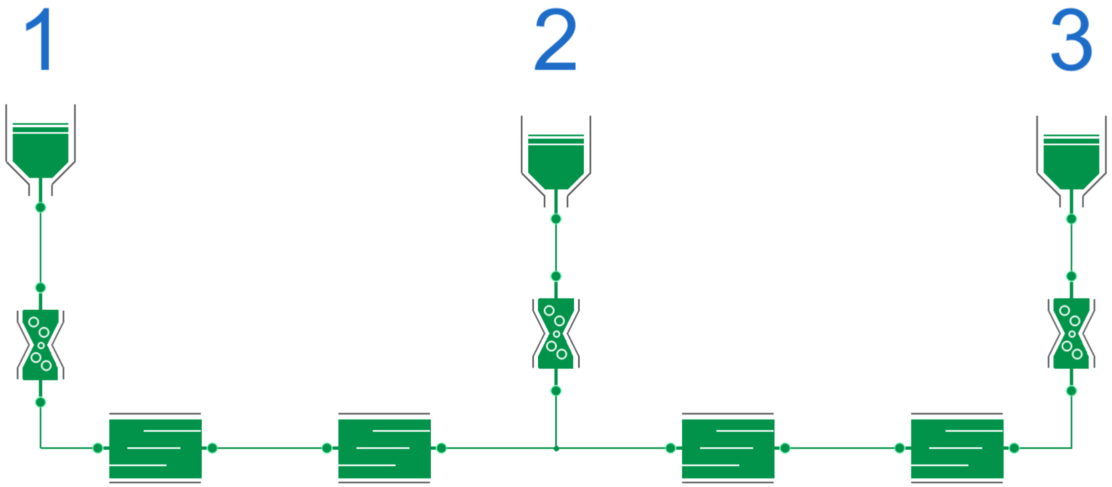
Energy flows misperceived as Interface



- Global analysis necessary
- Non-Deterministic
- Information incomplete

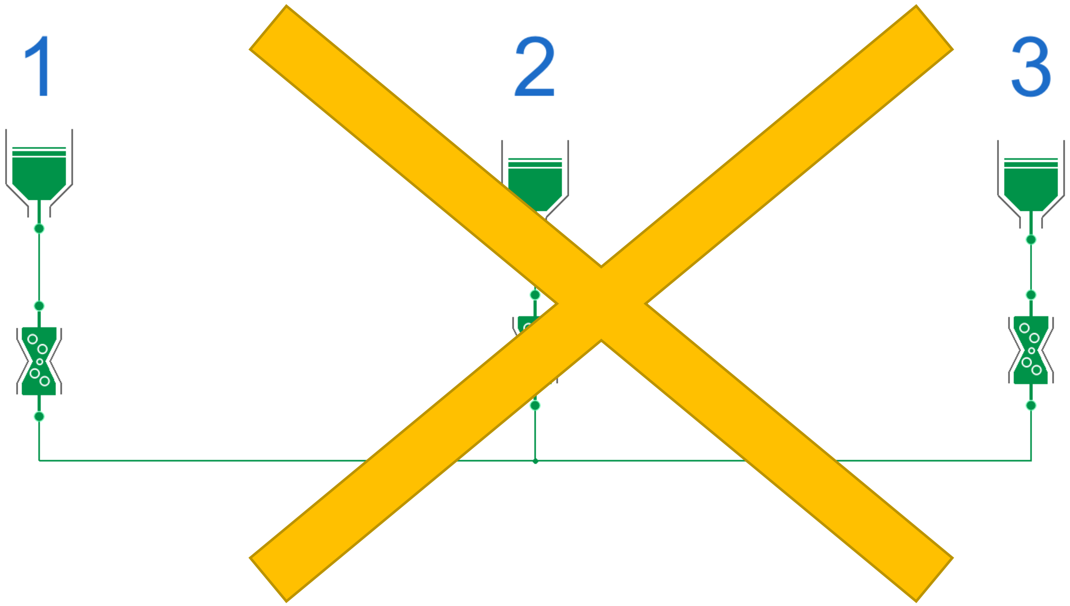
ModelicaLite: Enforce Determinism for Energy Flows

Energy flows modeling a physical system



Knowing

Energy flows misperceived as Interface



Guessing

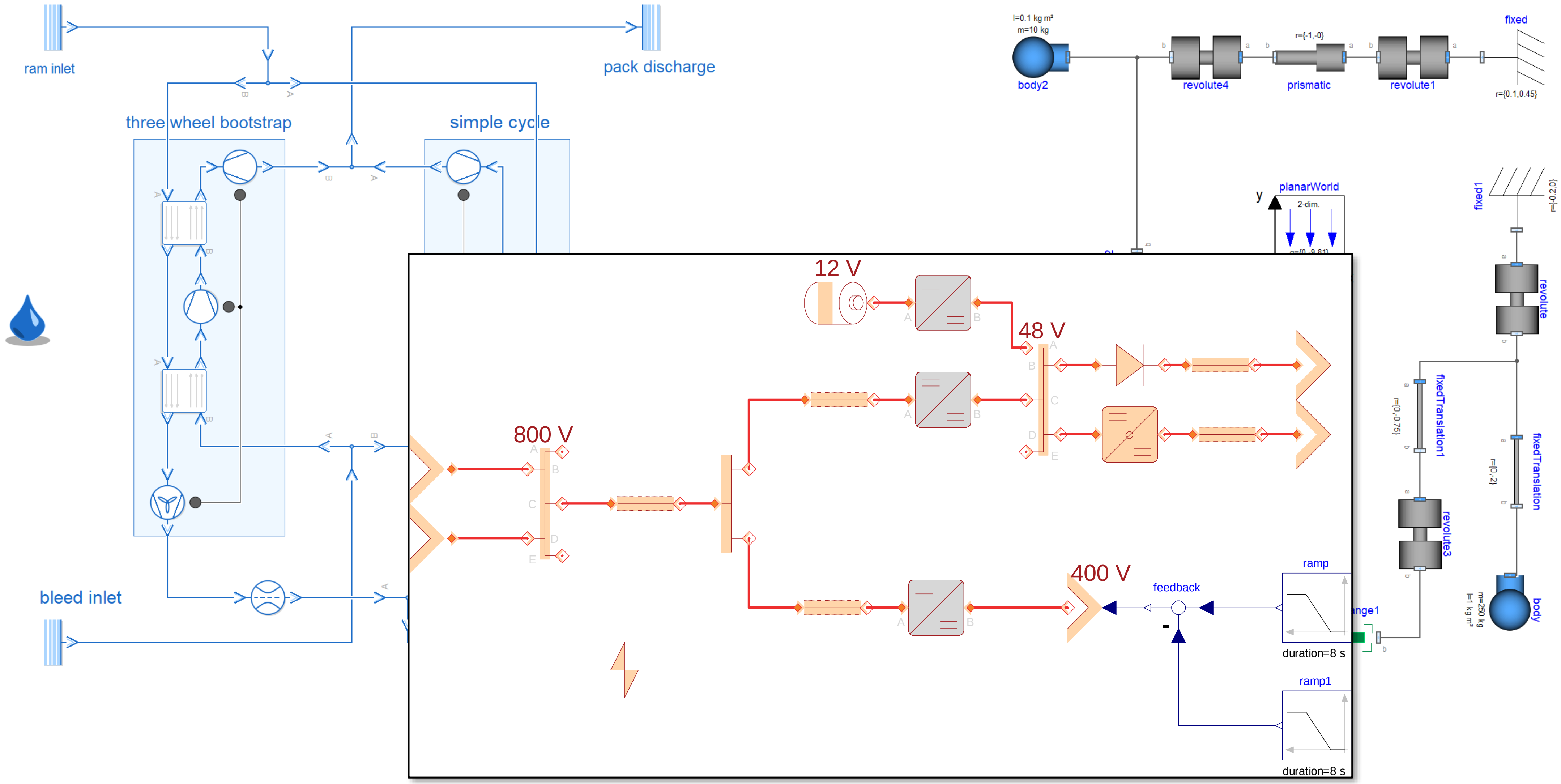
ModelicaLite: No Flow without a Signal Flow

- The best currently known cure for this problem is to bind the pair to a signal
  The pair then represents the part of the Lagrangian subjected to the Legendre transformation to get the Hamiltonian.

Domain	Translational Mechanics	Rotational Mechanics	Thermo Fluids	Electrics	?	…
Potential	v_{kin}	ω_{kin}	r	?	…	
Flow	f	τ	$\dot{m}$	?	…	
Signal	r	φ	$\mathbf{\Theta}$	?	…	

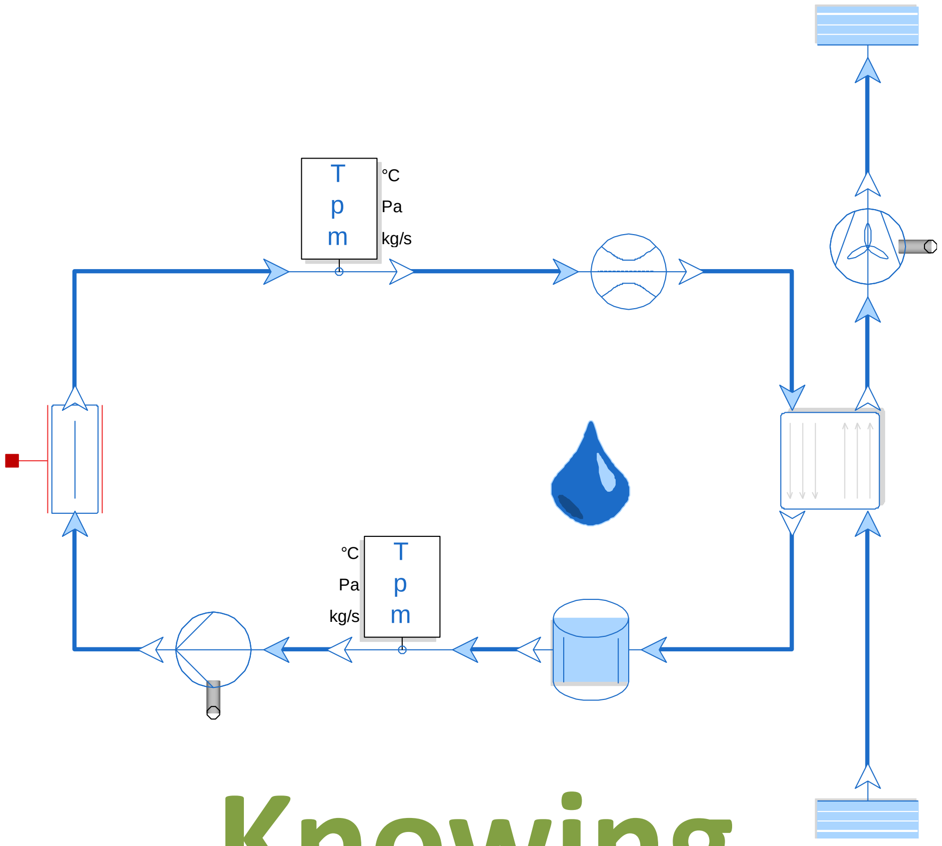
- There are already useful libraries according to this principle
  - ThermoFluid Stream
  - Dialectic Mechanics
  - Controlled Energy Flows

Motivation: From Necessary to Sufficient



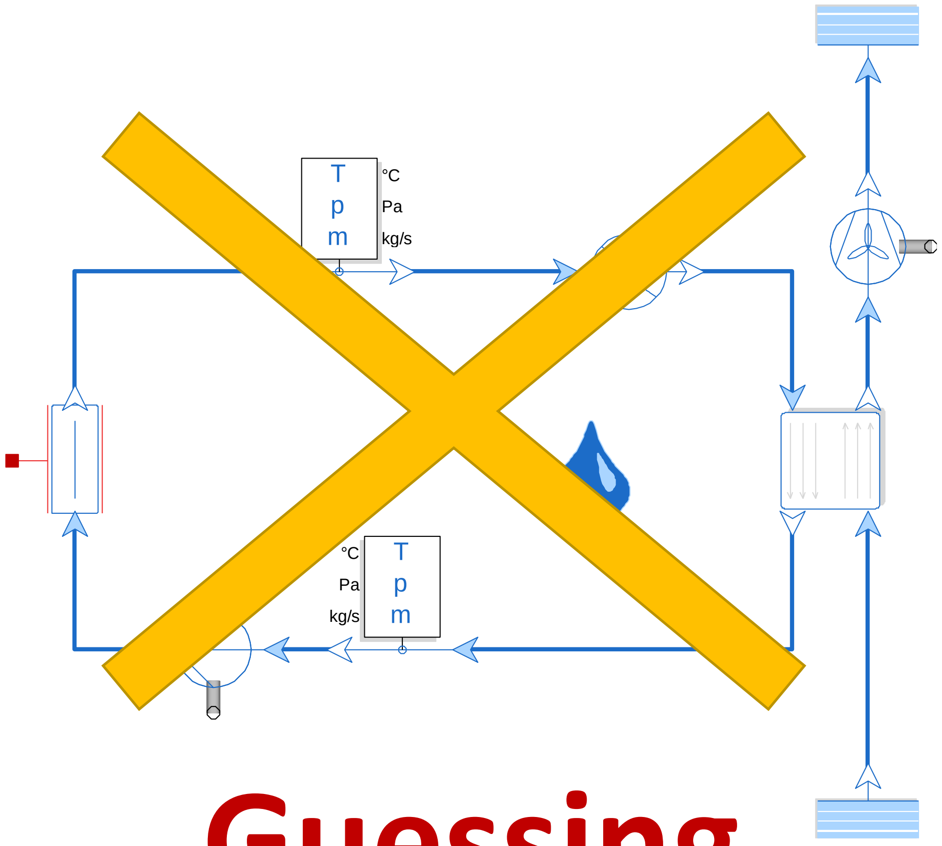
ModelicaLite: Enforce Determinism for Energy Flows

Energy flows modeling a physical system



Knowing

Energy flows misperceived as Interface



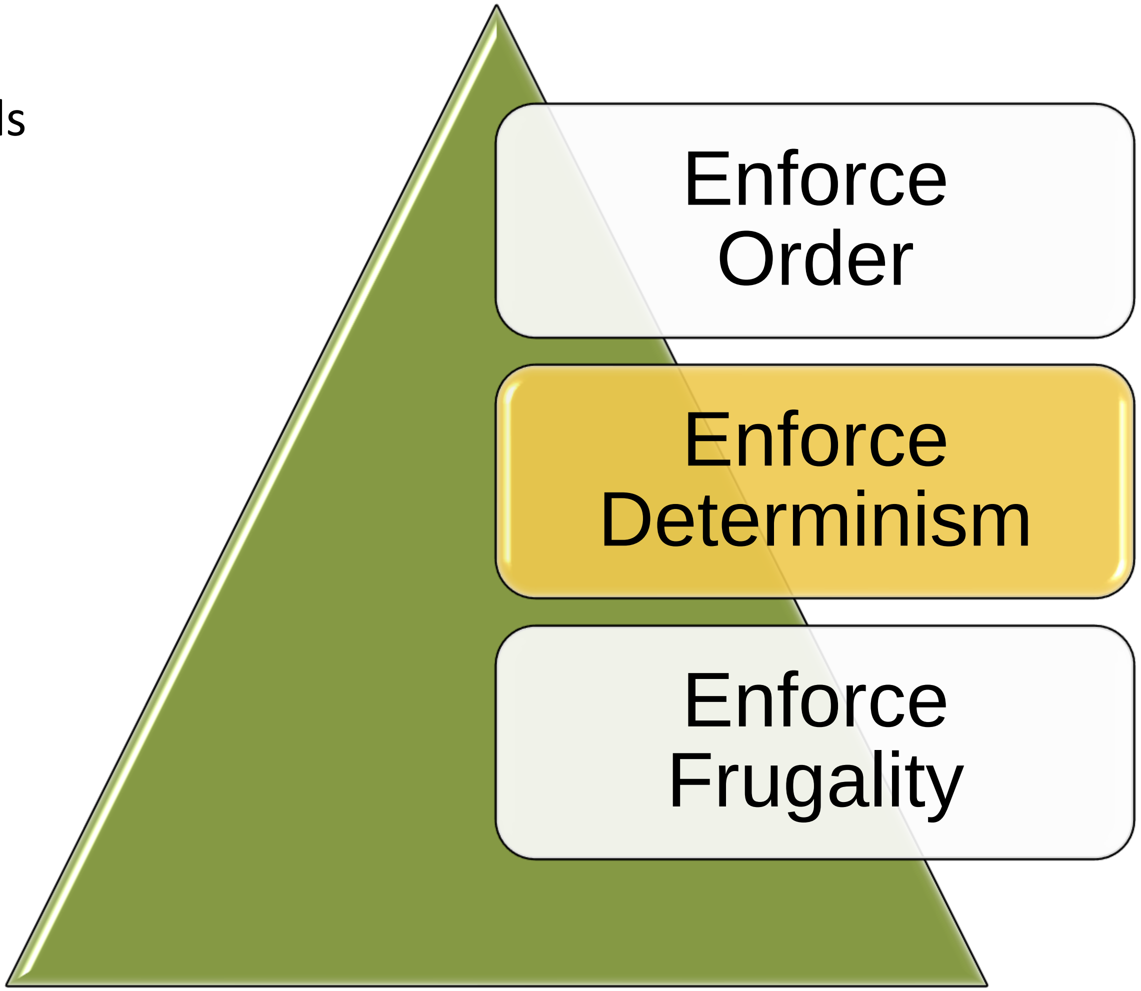
Guessing

ModelicaLite: Enforce Determinism

- Enforcing
  - Enables modular compilation of kernels
  - Scalable system composition
  - Robust modeling

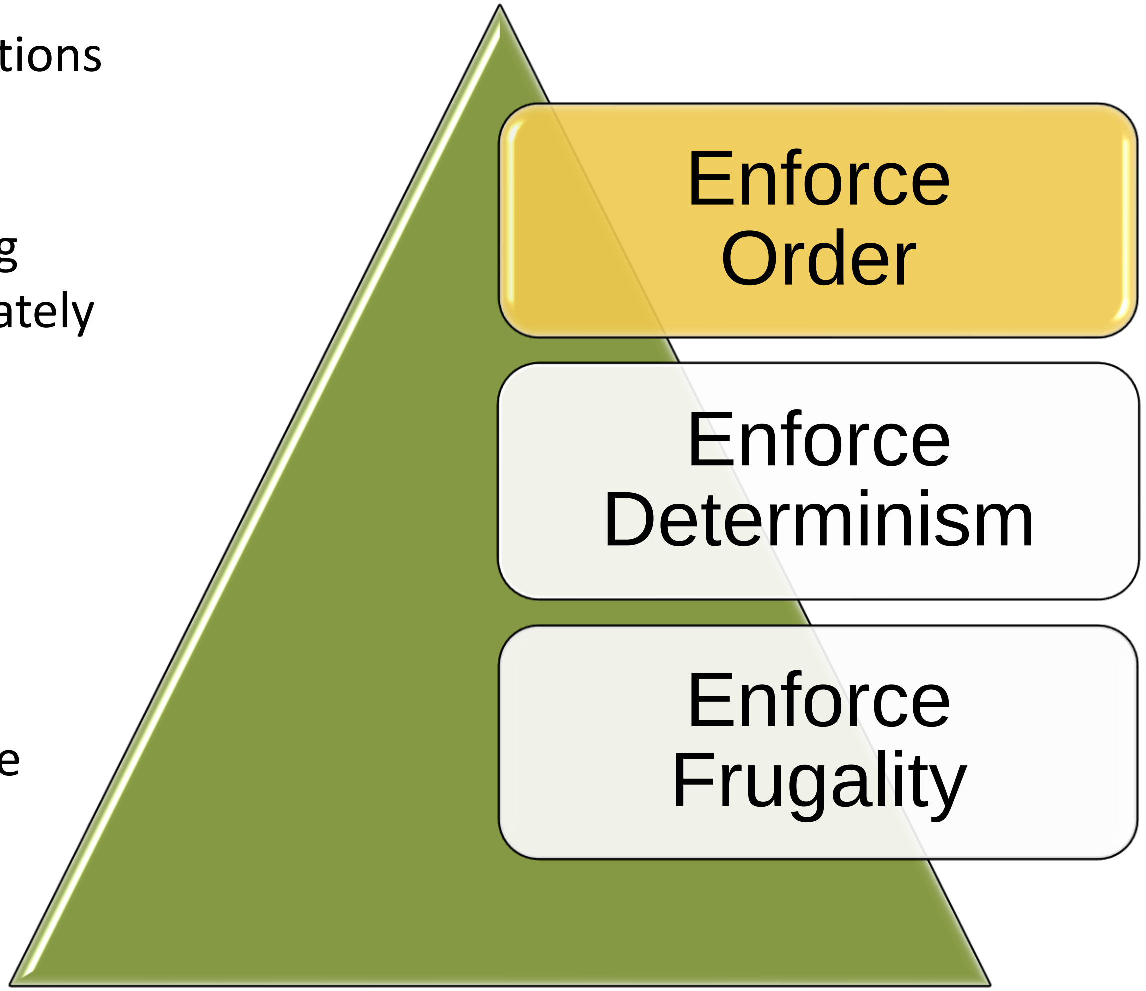
- No need for
  - Dulmage Mendelsohn Permutation
  - Higher Index Reduction
  - Dynamic State Selection
  - Tearing Heuristics
  - Non-linear root solvers

- Still need for:
  - Tarjan (modified)
  - Differential index reduction
  - Linear equation system solvers



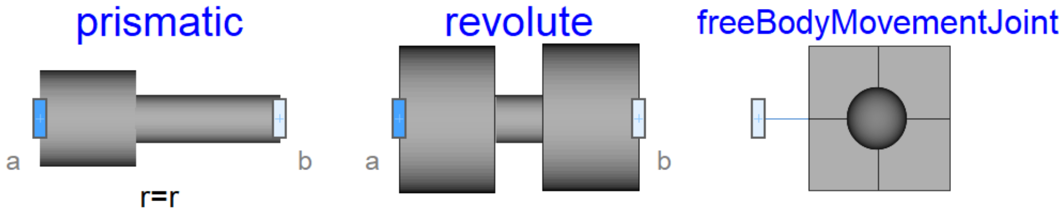
ModelicaLite: Enforce Order

- A Modelica compilers can re-order declarations and equations.
- However, this should not encourage writing messy models and packages, but unfortunately it does.
- Enforcing order:
  - Keeps package dependencies in check
  - Enables error messages on the line
- Each line in ModelicaLite shall be able to be understood without looking forward.



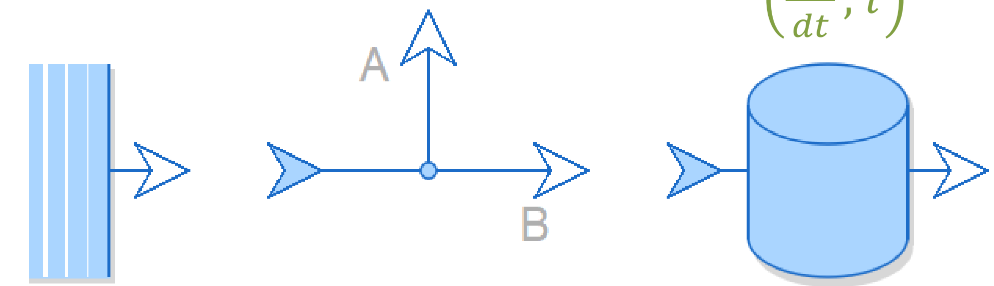
ModelicaLite: Enforce Order with assumed Causality

- Binding the pairs to a signal flow enables an a-priori casualization of the corresponding pair:
  - The potential (and its derivatives) shall have the same causality as the signal
  - The flow (and its derivatives) shall have the inverse causality as the signal
- Static pre-determination of states and tearing variables is also feasible
- This means that the component shall be formulated under the assumption of this causality.
- <u>This enables error messages on the line</u>
- Modelling beginners will thank for this guidance



s, v $\quad \varphi, \omega$ $\quad \{x, y, \varphi\}, \{v_x, v_y, \omega\}$

$\left(\frac{ds}{dt}, f\right)$ $\quad \left(\frac{d\varphi}{dt}, \tau\right)$ $\quad \left(\frac{dx}{dt}, f_x\right), \left(\frac{dy}{dt}, f_y\right), \left(\frac{d\varphi}{dt}, \tau\right)$



$\dot{m}$ $\quad \dot{m}$ $\quad \dot{m}, \boldsymbol{\Theta}$

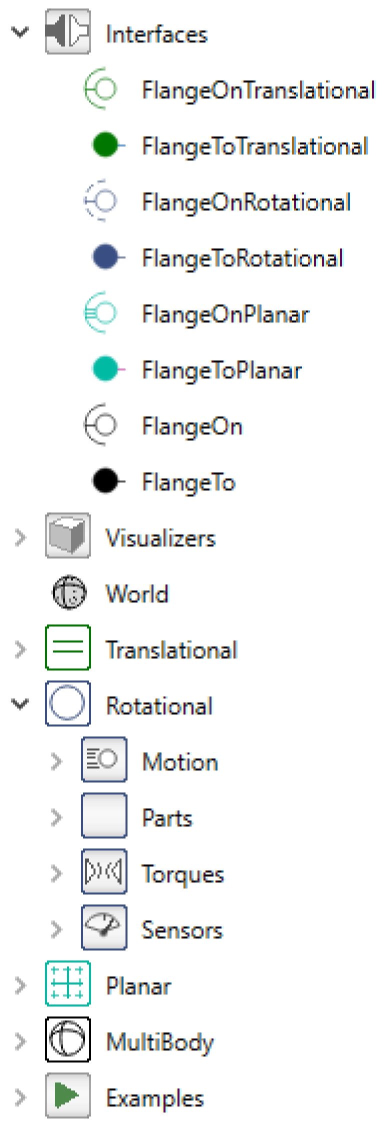
$\left(\frac{d\dot{m}}{dt}, \Delta r\right)$ $\quad \left(\frac{d\dot{m}}{dt}, \Delta r\right)$ $\quad \left(\frac{d\dot{m}}{dt}, \Delta r\right)$

ModelicaLite: Enforce Order among Packages

- There is a reason why a modern language like GO is so strict about packages.
  - If package B depends on A, then A must not depend on B directly or indirectly
  - Loadable packages must be encapsulated
  - Imports only directly in encapsulated packages
  - If package A imports C, but C is not used then compile is aborted.

- If discipline is not enforced, everything will eventually depend on everything.

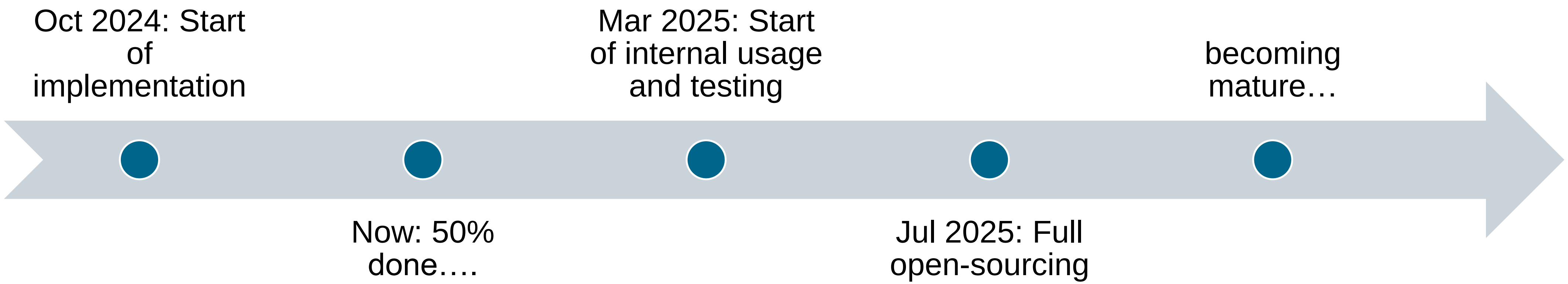
- Go learned its lesson from C++. We shall learn our lesson from the MSL.



IMPLEMENTATION

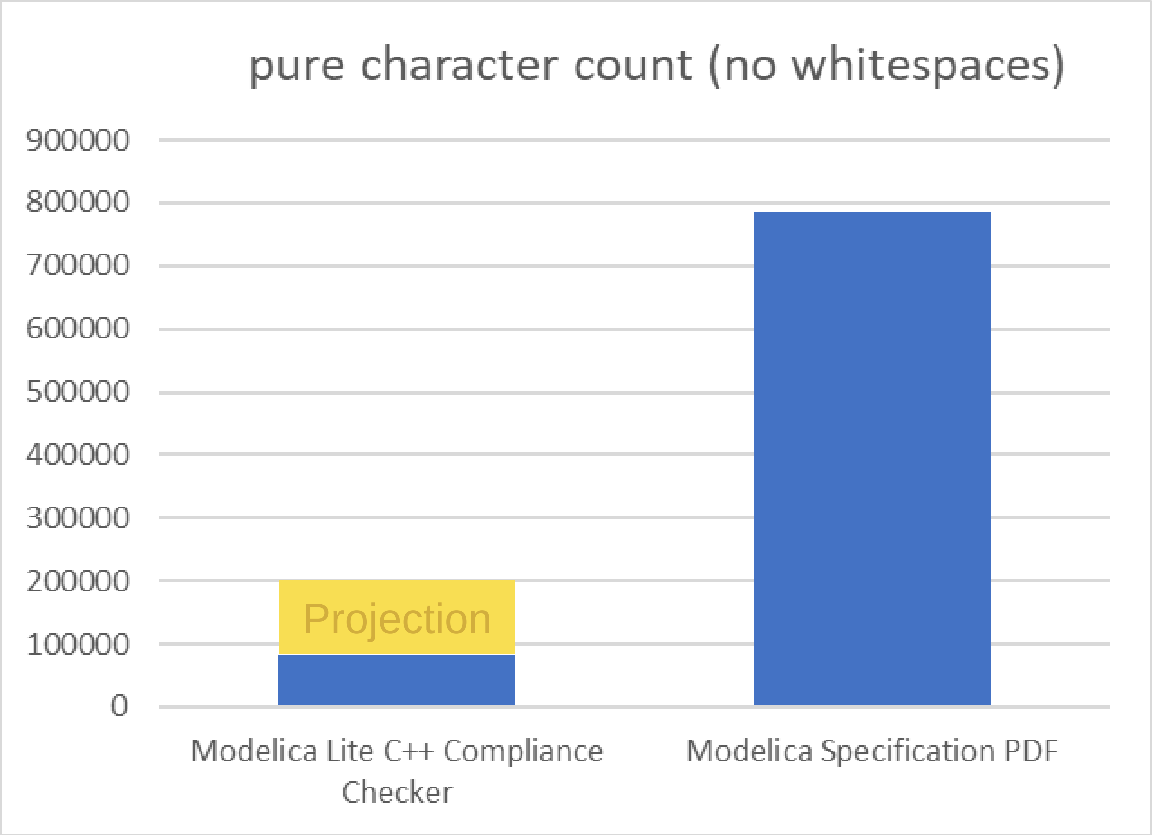
```cpp
        return false;
    }

bool Parser::expectModel(shared_ptr<HierachicalElem> Parent) {
    bool partial = expect(Tokens::_partial, false);
    if (expect(Tokens::_model, partial)) {
        if (expect(Tokens::_name, true)) {
            auto curDef = make_shared_tk<Model>(lastToken, lastToken.text, Parent, partial);
            bool valid = true;

            while (expectExtendsStmt(curDef));

            ScopeSpecifier scope = ScopeSpecifier::publicScope;
            while (expectScopeSpecifier(scope) || expectDeclaration(curDef, false, scope));

            if (expect(Tokens::_equ_section, false)) {
                while (expectEquation(curDef) || expectConnection(curDef)) {
                    expect(Tokens::_scolon, true);
                }
            };

            if (expect(Tokens::_annotation, false)) {
                valid &= expect(Tokens::_scolon, true);

            valid &= expect(Tokens::_end, true);
            if (expect(Tokens::_name, true)) {
                if (curDef->getName() != lastToken.text) {
                    errorstream << ErrorMsgToken(curToken) << "Identifiers do not match. Found : " << lastToken.text
```

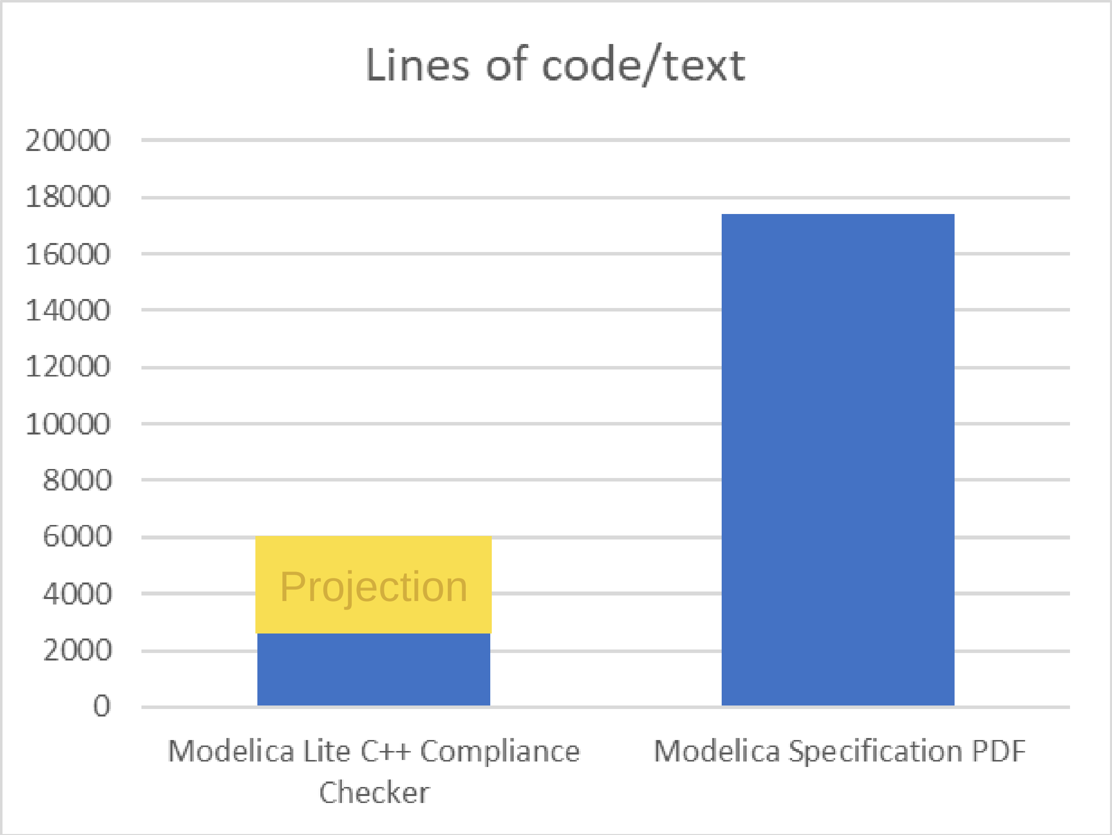
ModelicaLite: Compliance Checker



- Fully self-contained implementation in C++17
  (Handwritten Parser and Lexer. Just uses STL nothing else)
- Checks Compliance with Modelica Lite
- Shall enable independent implementation of ModelicaLite Libraries.
  (Running as console application in parallel to Modelica IDE)
  - 1D – 3D Mechanics
  - ThermoFluid Stream Lite
  - Controlled Energy Flows
  - …
- Should be 75% of the work for a compiler to another high-level language.

ModelicaLite: Measuring Complexity Reduction





- Goal is to avoid the writing of an explicit specification but make the code so good and transparent that it serves as specification.
- Reusing existing IDEs and tools is also of big help

ModelicaLite: Cooperative Mode for Tool Development

	Cooperate	Compete
Cooperate	fmi Functional Mock-up Interface Modelica Lite	non-equilibrium
Compete	non-equilibrium	Modelica

- Implementing compilers is mostly just cost without revenue stream.
- Hence one is motivated to reduce and share costs.
- Unfortunately this is not true for Modelica compilers
- FMI did a better job (could build on a matured association)
- We shall also do better with ModelicaLite