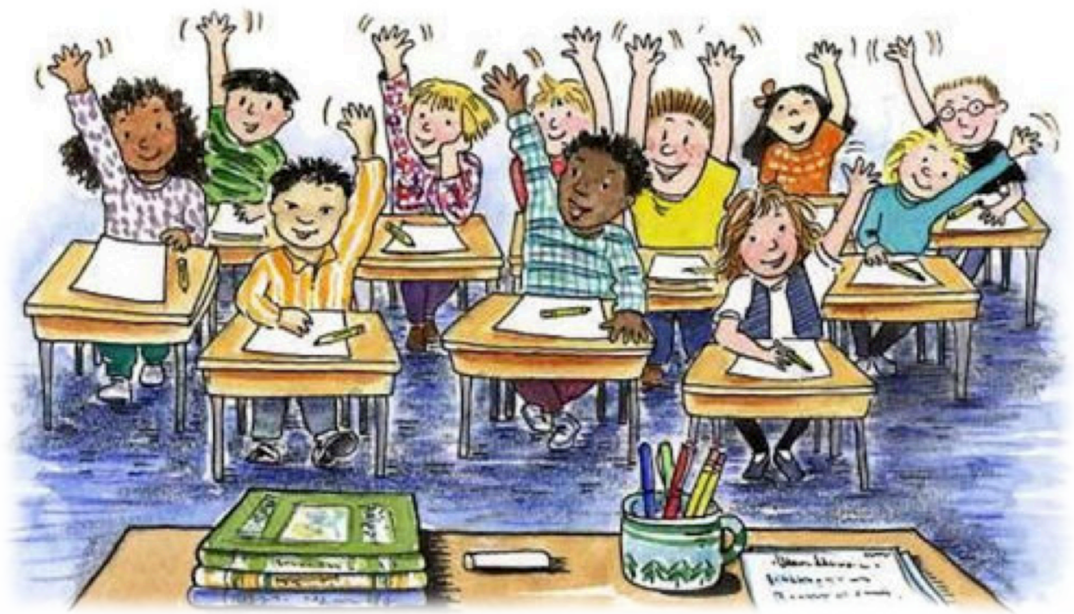


Teaching Modeling and Variability in Software Design

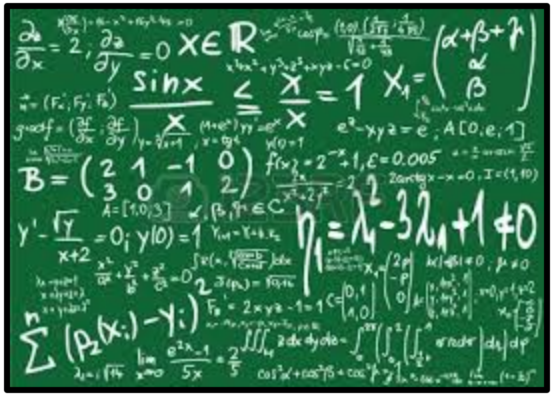
and its Importance to Science

Don Batory
Department of Computer Science
University of Texas at Austin
Austin, Texas 78712

Overture

- My work is in **automated software design (ASD)**
  - my interests: **software product lines (SPLs)**, **model driven engineering (MDE)**, refactoring
  - background in systems, ***not*** formal methods
  - flavor of my work is to use mathematics to explain what I have observed and done in practice

from practice → to theory



Heard Lectures by

Jay Misra

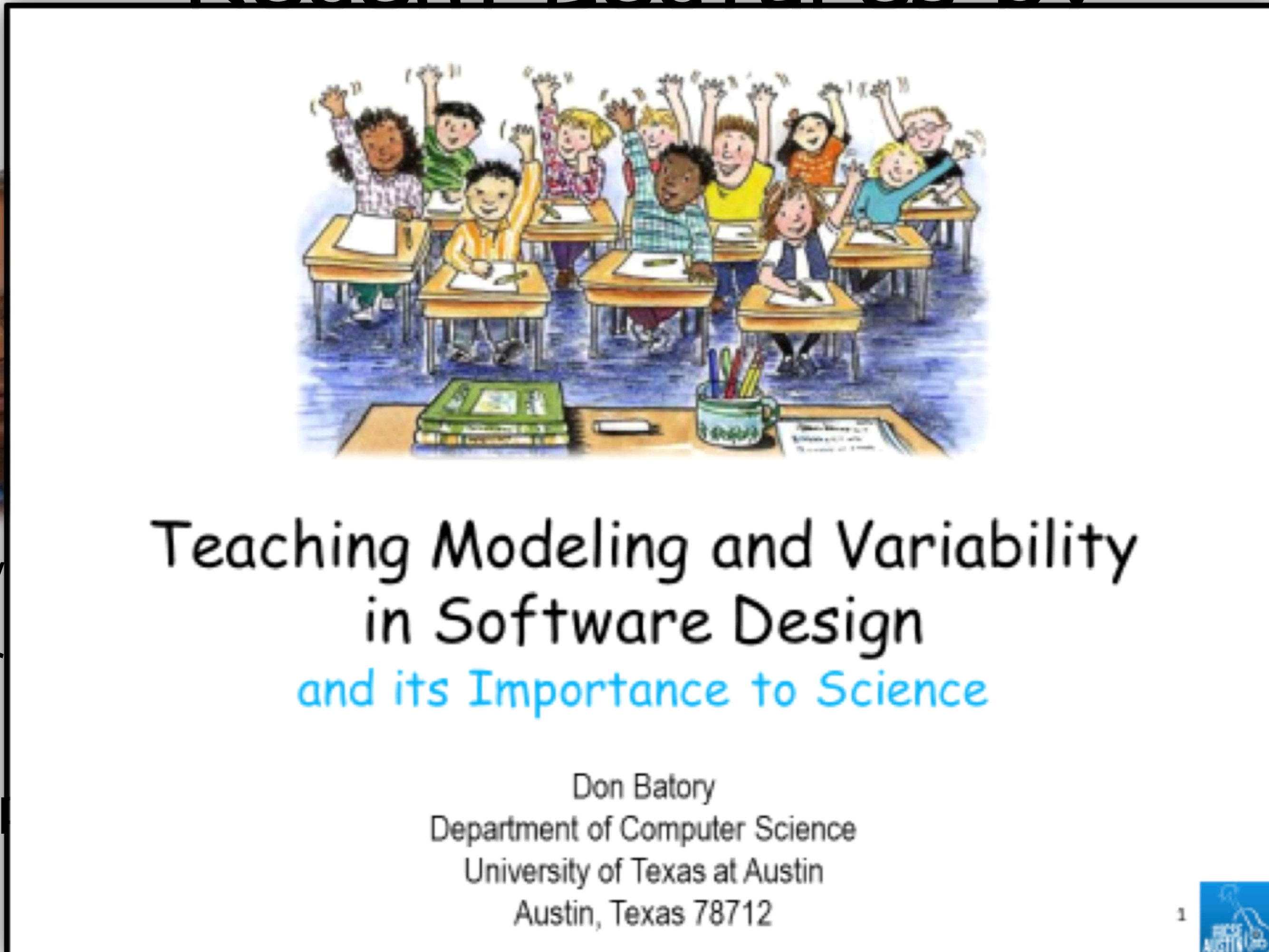
Tony Hoare

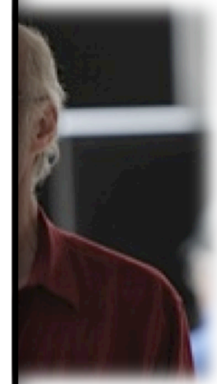
Ric Hehner

- Their work is to find algebraic laws of programming (esp. concurrency)
- My approach to software automation has this flavor (physics), but is not as formal
    - geared more toward practicing engineers
    - does not preclude others with stronger formal backgrounds to follow up

Recent Lectures by

Jay





nner

- Their

- My a ... formal

w up

Introduction

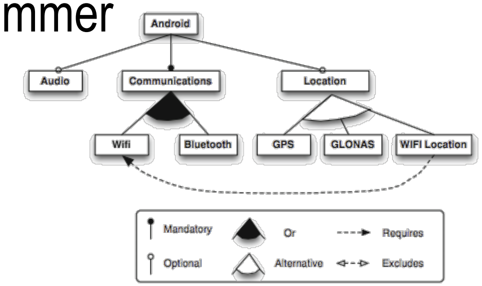
- I teach undergraduate and graduate software design courses

- Goal is not JUST to present software design as:
  - a collection of best practice techniques and tools to create understandable and cost-effective designs, but also
  - a foundation to explain classical and revolutionary concepts in last 25 years

Model Driven Engineering
Refactorings
Design Patterns
Parallel Architectures
Product Lines

Motivation #1 For This Talk

- I attended Software Product Line Conference in Nashville last summer

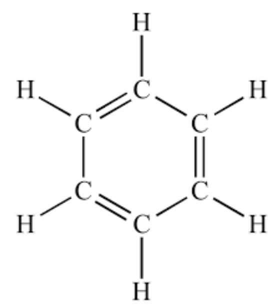


- Listened to a keynote of a founder of that Conference Series
- History and motivation behind the creation of SPLC
  - Software Reuse Conference lacked a focus on practical, cost-effective technology

  - and "we didn't have to worry about math"

Not Surprising

- Historically w.r.t. science, I think Software Design is ~1880s



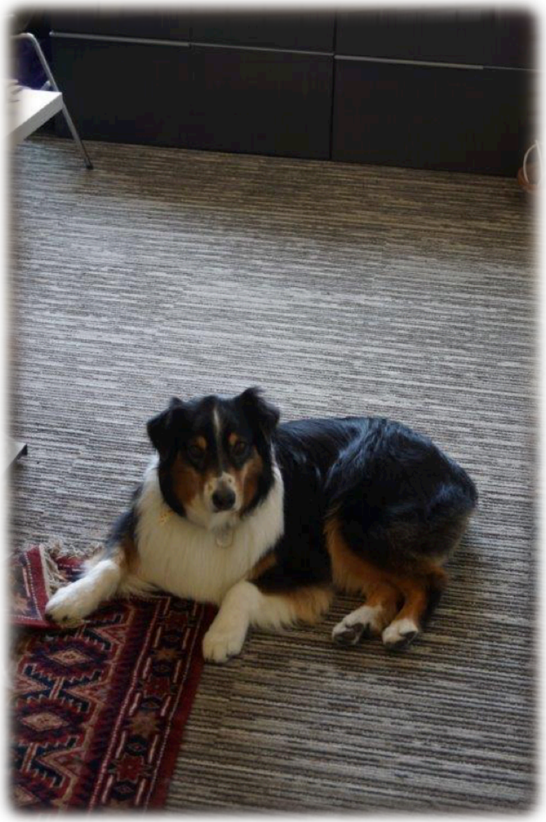
- Practice as an art dominated in chemistry
- Against the tide of the history of science

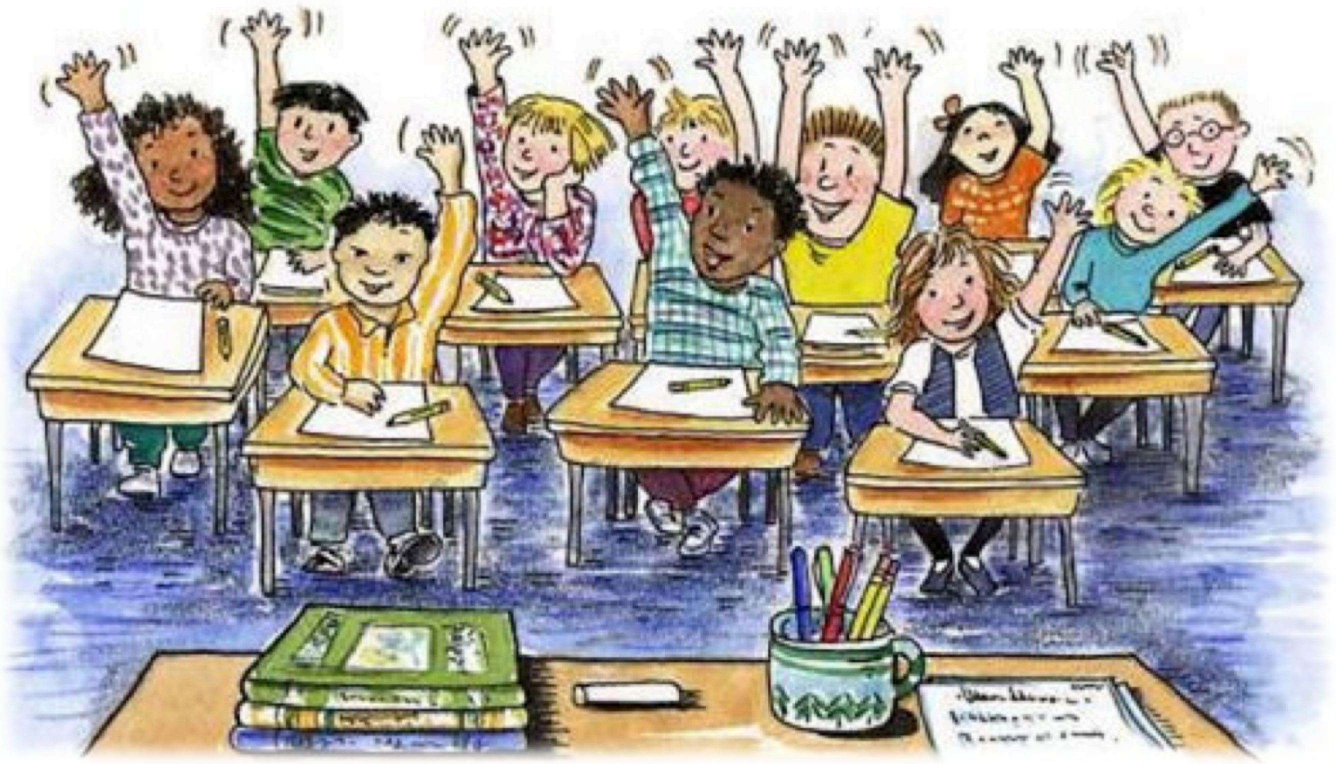
In practice, there is no difference between theory and practice.

In theory, there is.

The Difference is Obvious

- View Software Design from reductionist and mathematical perspective
- Only if you're in the right place
- At the right time
- Looking in the right direction
- You'll discover something beautiful





A Starting Point to Teach Modern Software Design

Basic Mathematics

- Addition and subtraction of **numbers**

identity	$A+0=A$
commutativity	$A+B=B+A$
associativity	$A+(B+C)=(A+B)+C$

- Some operations (multiplication) distribute over addition

$$R\cdot(G+H)=R\cdot G+R\cdot H$$

- Functions $F:\mathbb{R}\rightarrow\mathbb{R}$

not in this talk…

Dreaded Homework Assignments

- Are these expressions equal?

$$x\uparrow 2 + 5\cdot x + 6 = (x+3)\cdot(x+2)$$

- Obviously no! They are different

- We were taught to apply a series of identities replace equals with equals to prove their semantic equality or not

$$(x+3)\cdot(x+2)$$

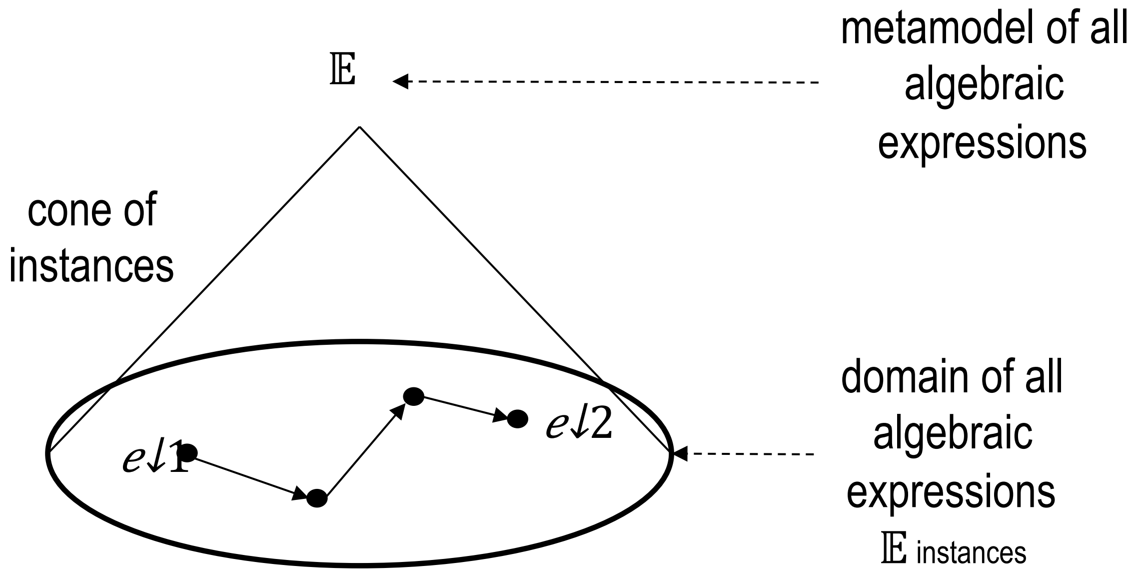
$$= (x+3)\cdot x + (x+3)\cdot 2$$

$$= x\uparrow 2 + 3\cdot x + 2\cdot x + 6$$

$$= x\uparrow 2 + 5\cdot x + 6$$

MDE Cosmic View

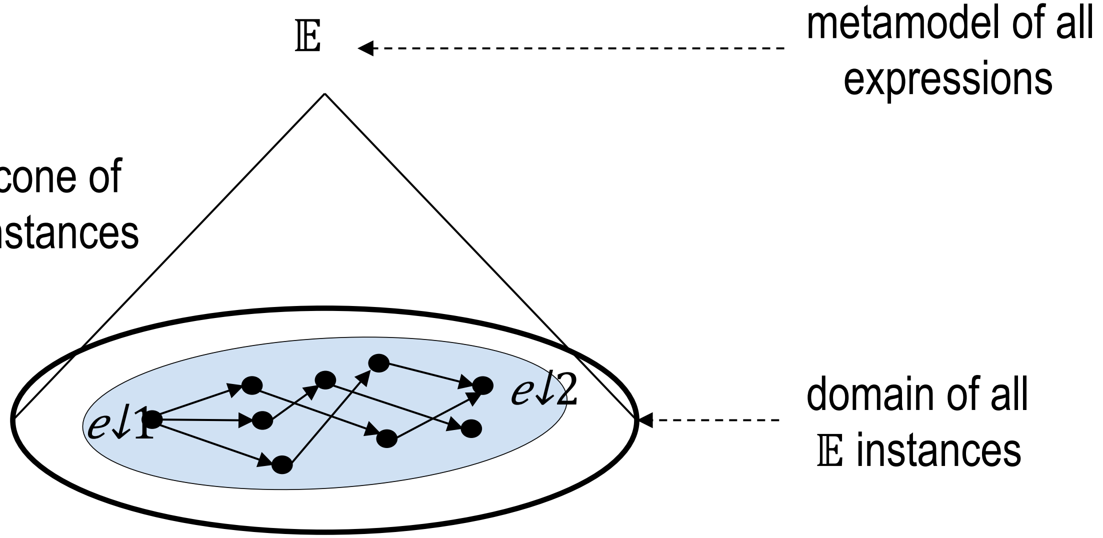
- We have a domain of algebraic expressions defined by metamodel $\mathbb{E}$



- We have 2 elements: expressions e_1 and e_2
- Is there a path proof between e_1 and e_2 using arrows identities that transforms e_1 into e_2 ?

MDE Cosmic View

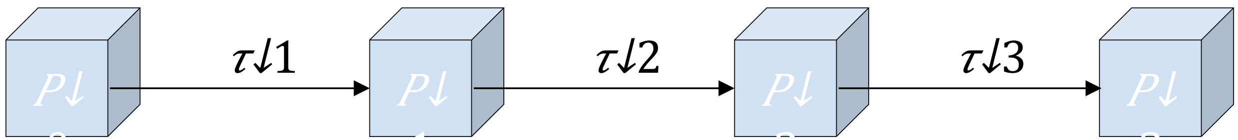
- Of course, there can be many paths derivations
- Can derive a large number of semantically equivalent expressions



- Yields a subdomain of "equivalent expressions"
- **A common source of variability in mathematics** that we don't think about
- You might say some expressions are better than others. More later…

Incremental Software Design

- Is a classical way to control software complexity



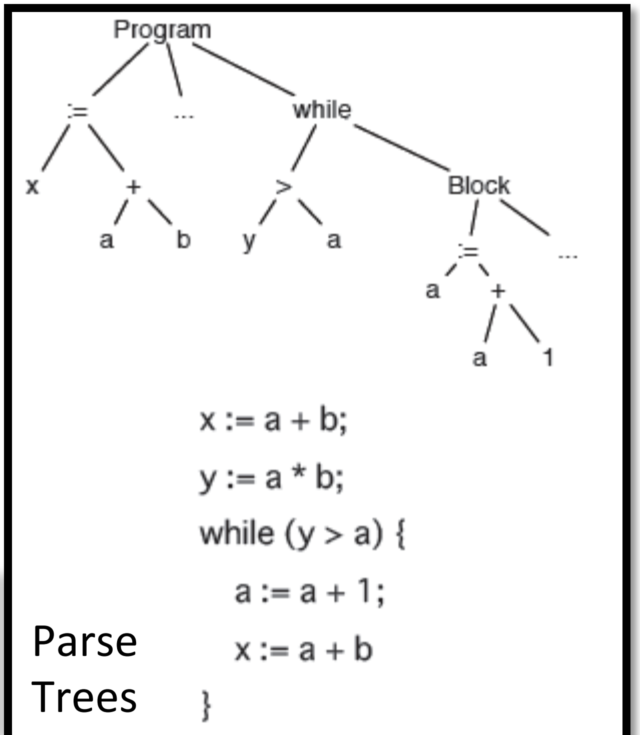
- Hallmark of Agile approaches, like XP

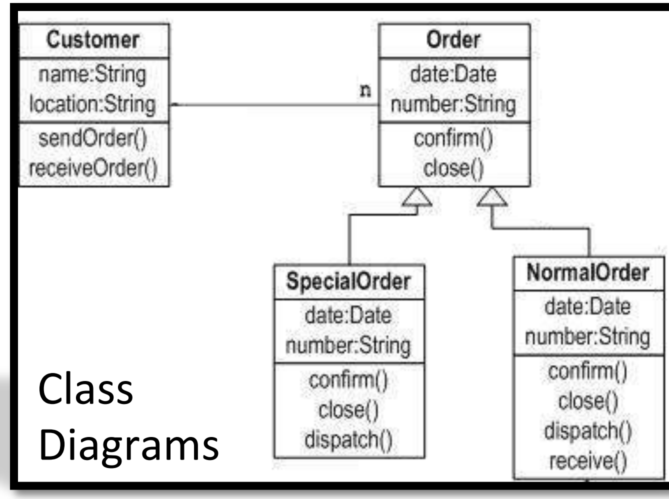
- Goal of **Automated Software Design** is to automate all or some the common arrows/transformations of software design

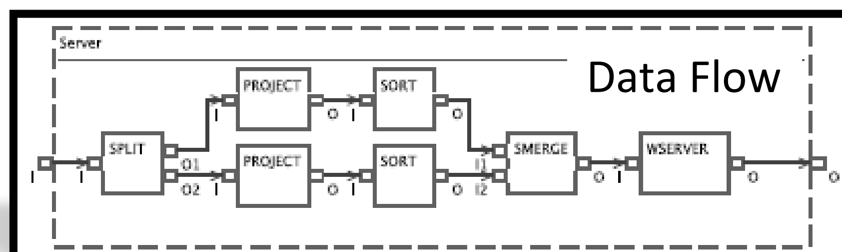
  - “all” is more likely in domain-specific applications
  - “some” expected in generic applications, like refactorings

Automated Software Design

- Programs are graphs







Foundations of Today's MDE

Automated Software Design

- Deals with the addition and subtraction of **graphs**, not numbers, w. similar properties

identity	$A+0=A$
commutativity	$A+B=B+A$
associativity	$A+(B+C)=(A+B)+C$

- Some operations distribute over addition: make graph Red

$$R\cdot(G+H)=R\cdot G+R\cdot H$$

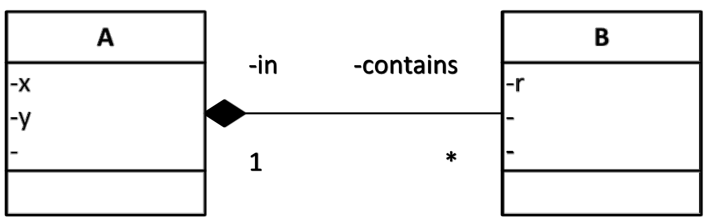
- Functions $F:\mathbb{G}\rightarrow\mathbb{G}$

not in this talk!

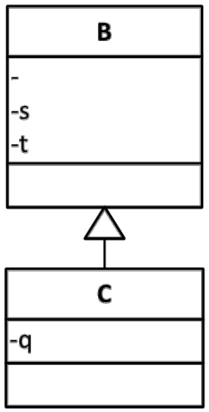
Make This Concrete!

- Start with UML class diagrams $\delta + \gamma = \phi$

$\delta =$



$\gamma =$



+ is a union-like op that has an identity 0, is commutative and associative

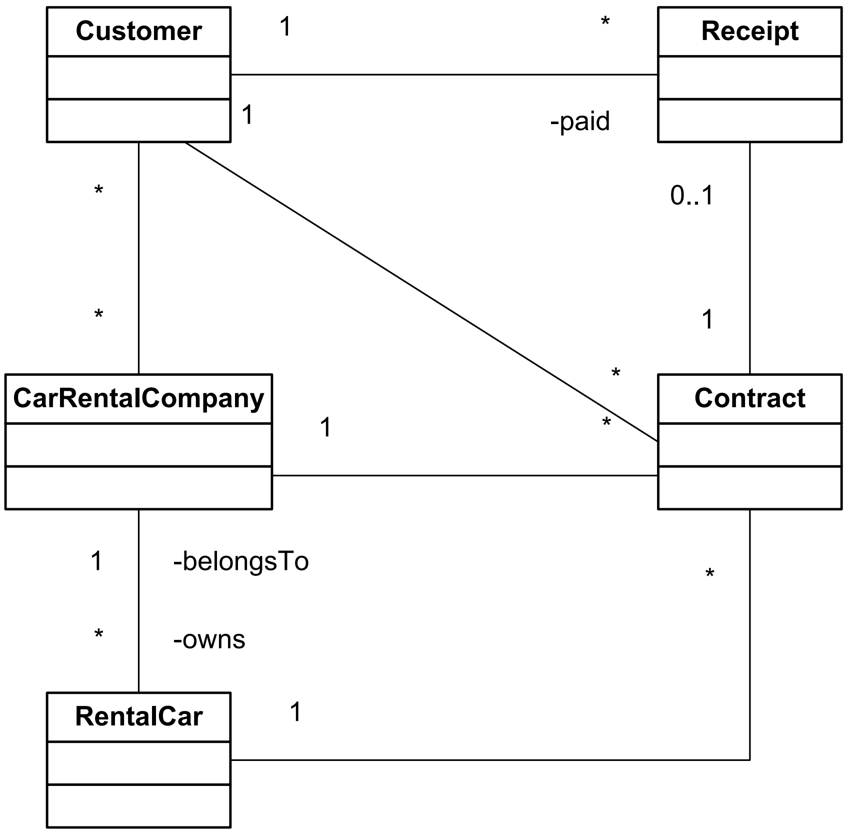
$\phi =$

– is a set subtraction operation

CUTE
BUT SO WHAT?

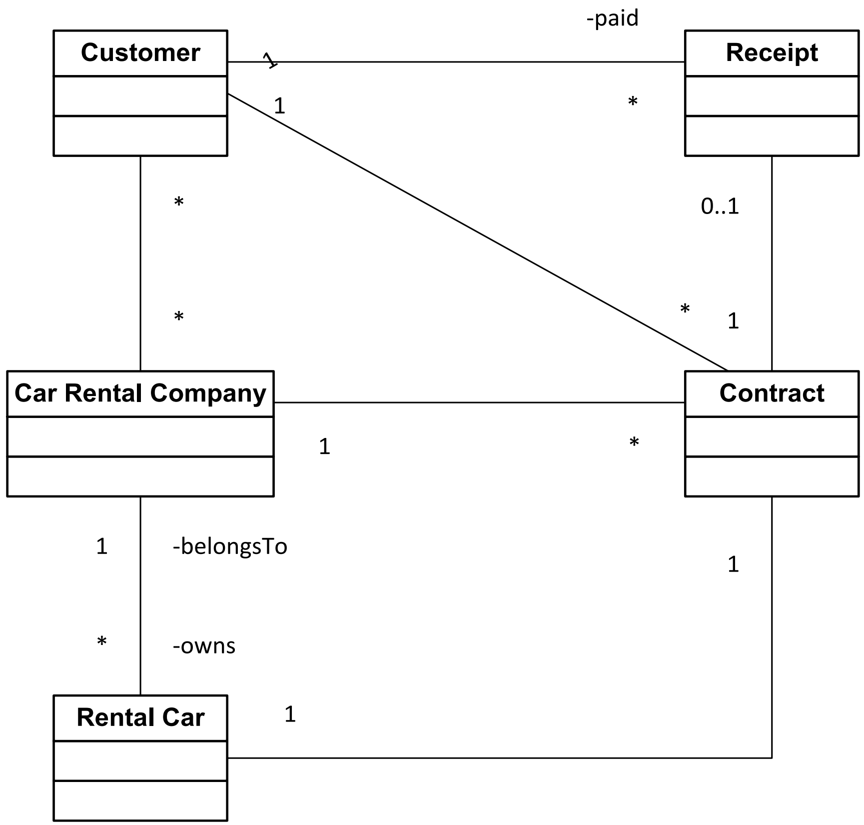
How Would You Explain

- A typical spaghetti class diagram to someone?



Using Graph Addition

- Build graph incrementally by adding subgraphs to a simple base graph

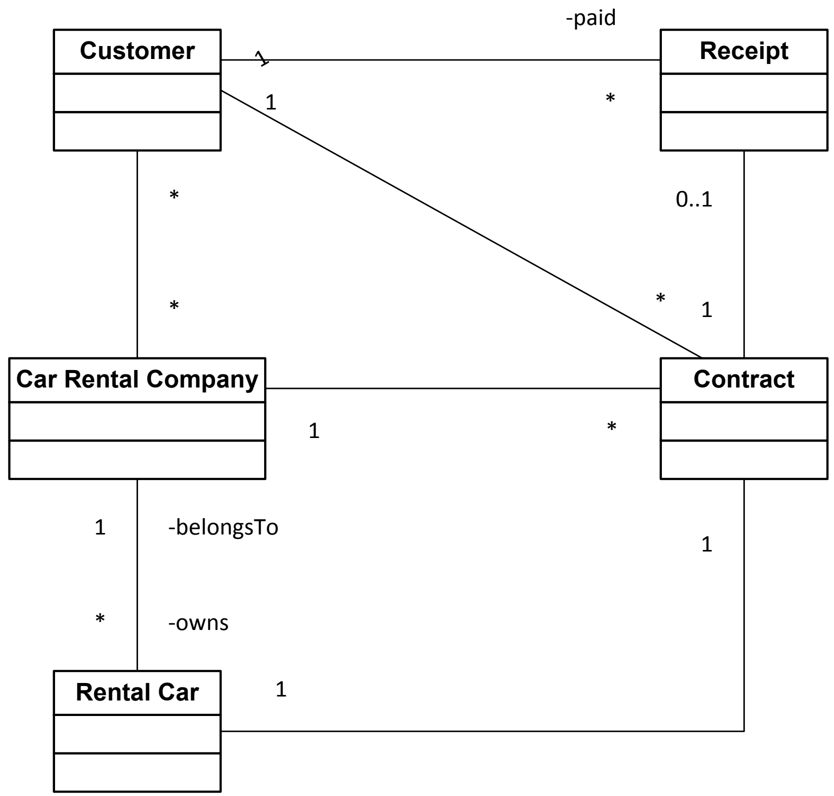


Explain Complexity in a Simple Way

- **Each step is understandable, implementable, and testable!**

No Unique Way to Construct Diagram

- Consequence of commutativity and associativity of graph addition



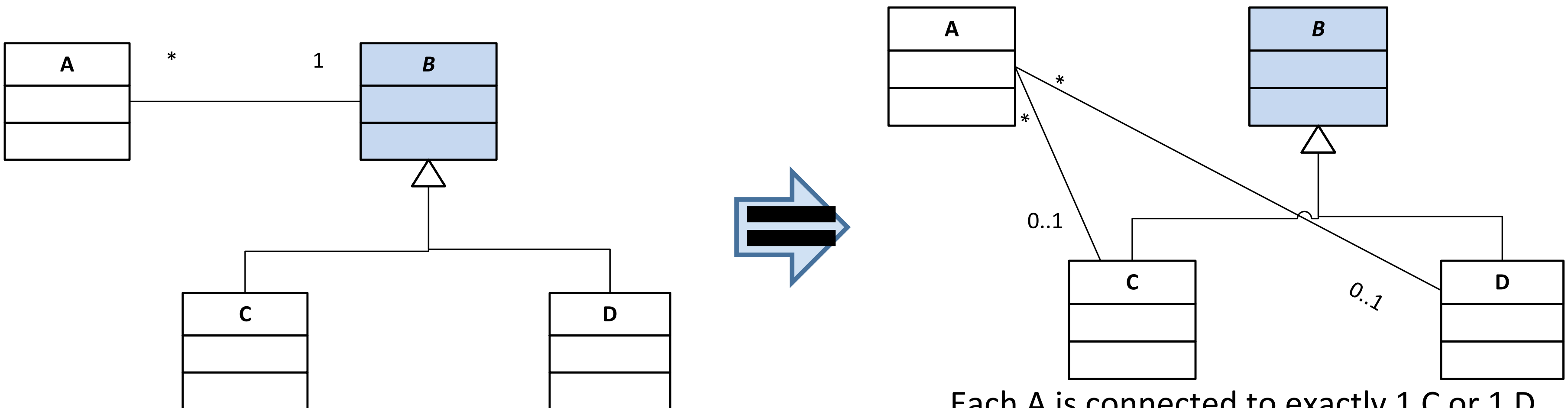
- **Math confirms the intuitive: one can create a design in any number of equivalent ways – more later…**

CUTE
BUT WHO CARES?

Graph Identities

often presented as Refactorings

- Push down association

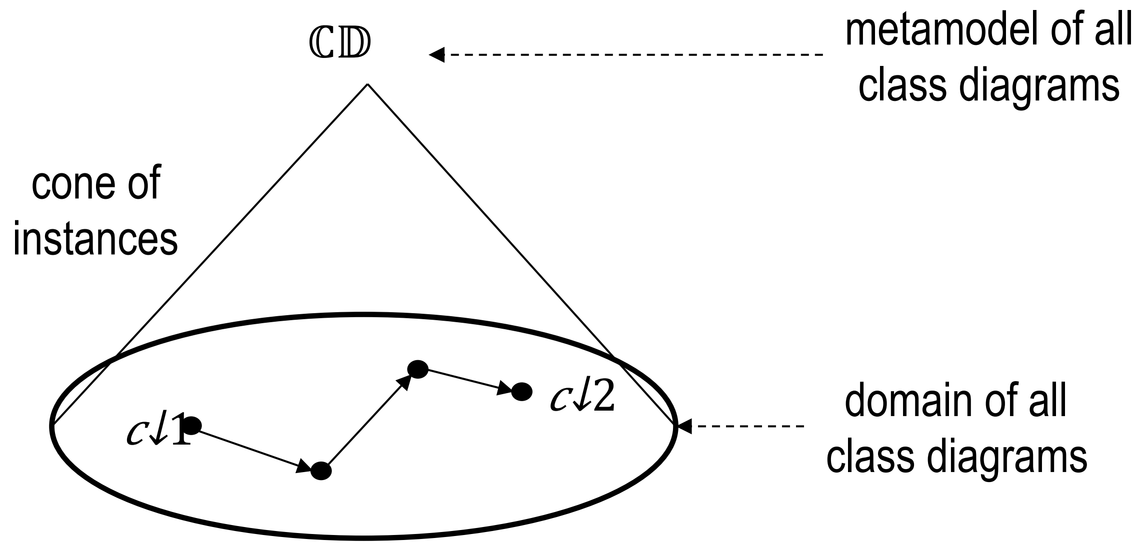


Each A is connected to exactly 1 C or 1 D,
Never both

- Refactorings are reversible (equalities)
- Lots of these identities – see more later…

MDE Universe

- Metamodel $\mathbb{CD}$ whose domain is class diagrams



- Given two class diagrams, c_1 and c_2 , does $c_1 = c_2$?
- Can we apply a set of graph identities to prove their equivalence?

- Let’s a look at an example

Motivation #2 for this Talk

- "Abstraction Challenges" Panel at MODELS 2013
  - I posed a question: if I give a modeling assignment** to my class…

** create a class diagram to express …

- Response caught me off-guard
  - if panelists said anything it was "but there is only one right answer"

Really??

Guess: Common Interpretation

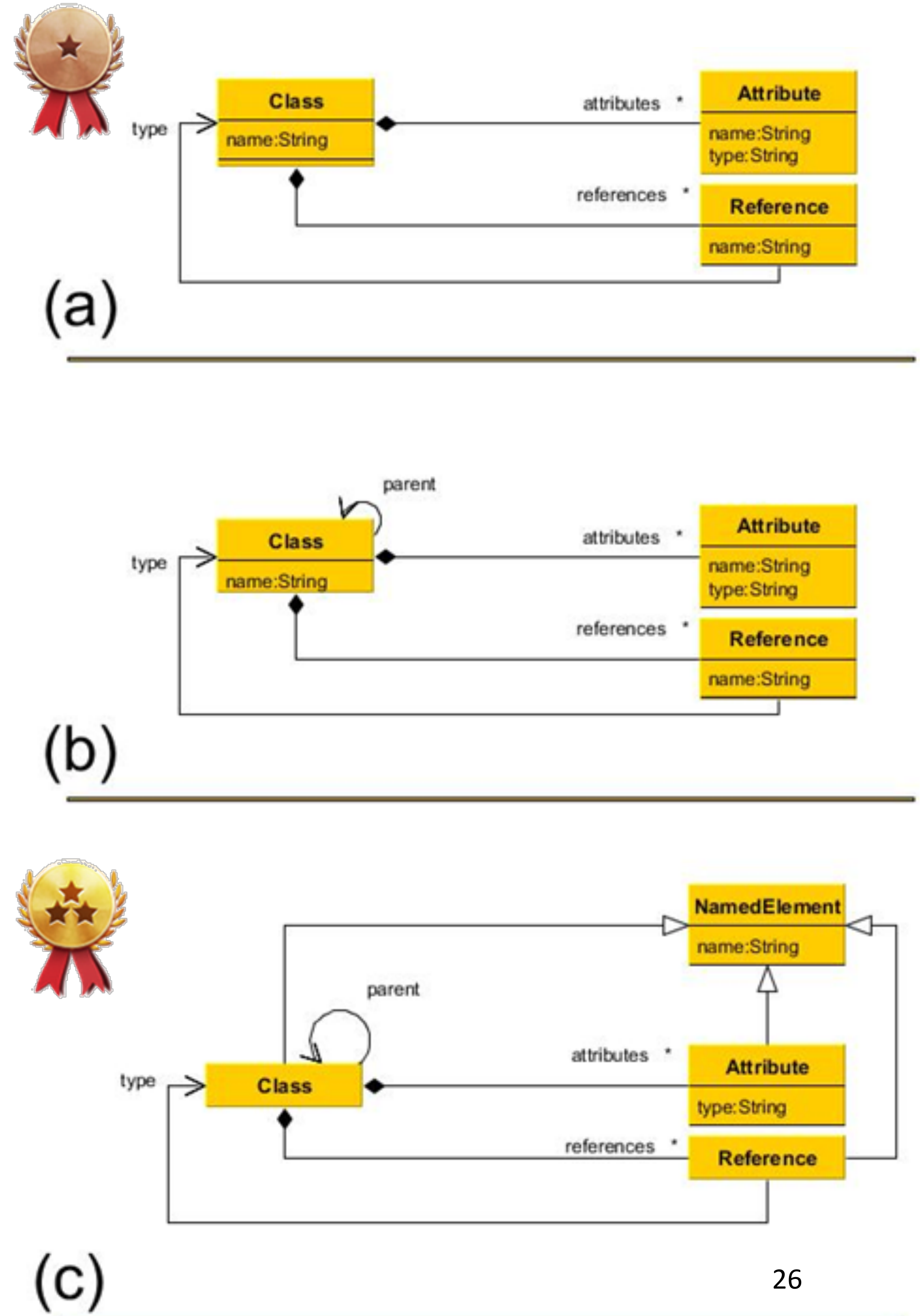
thank you Davide!

- Example from Davide Di Ruscio clarified a common interpretation
  - What is a class diagram of a class diagram?

a) allows classes to have attributes and directed associations

b) answer (a) + class hierarchies

c) answer (b) + pull-up common attributes name



Teaching Databases Mid-1980s

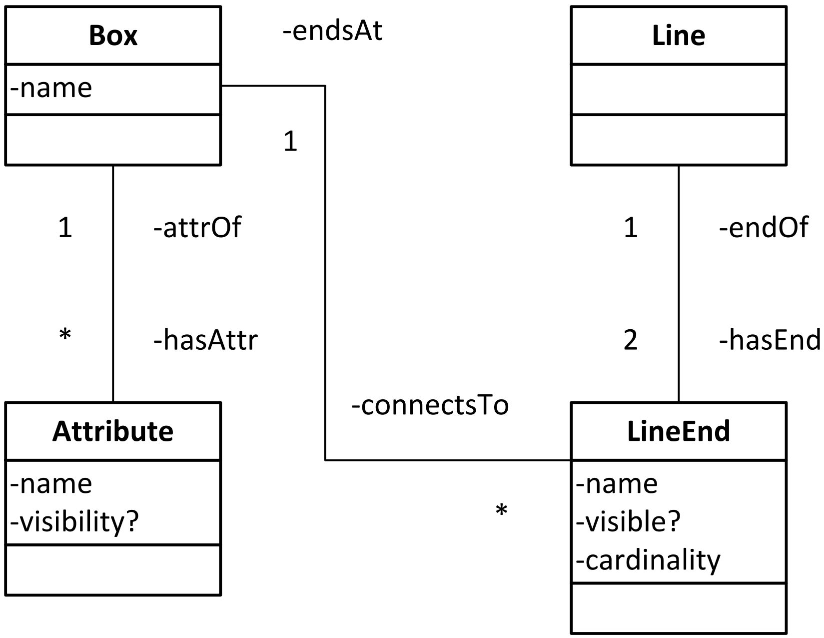
- I noticed the same problems
  - unless you are very specific about what you want students to model, you will get a zoo answers
  - I graded the same way as Davide…

- Decades later, in teaching Software Design, I took a broader perspective, I realized I needed to tighten the problem spec to reduce the zoo of answers

Where I Start Now

- What is a class diagram of a class diagram?

- Here is my in-class answer:

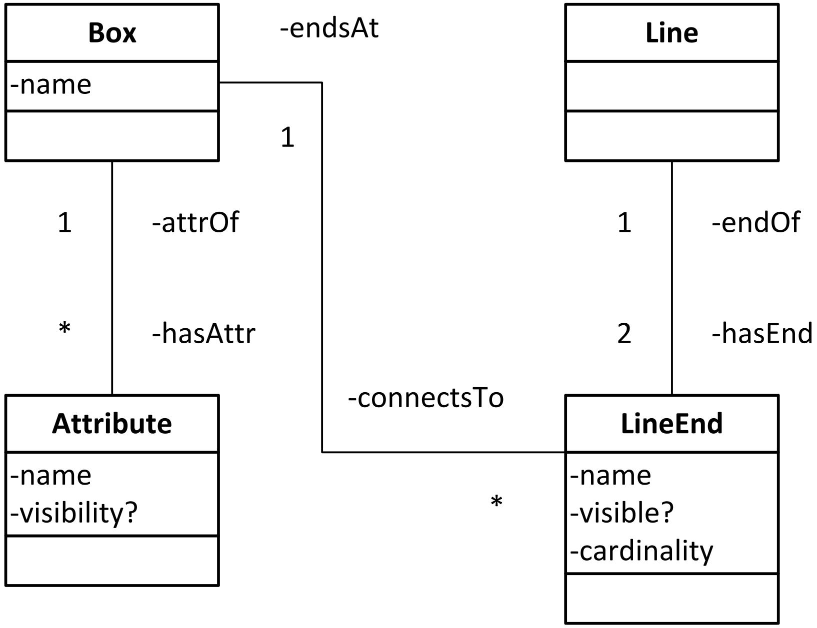


- For MDE purists,
  this CD is an instance of itself

The Assignment

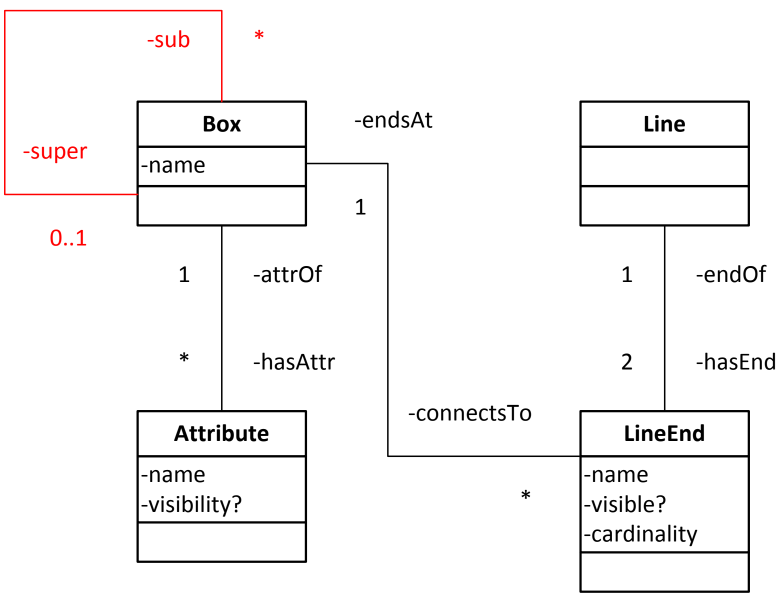
- Given this metamodel, what minimal change do you need to make to generalize it to express inheritance relationships among classes?

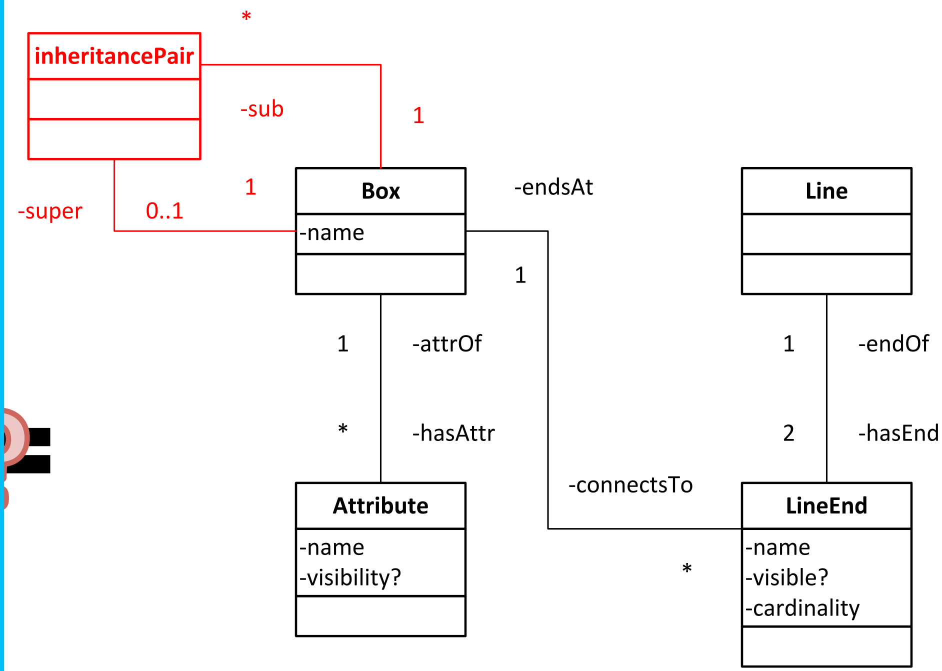
- And I get all sorts of answers…



Answers #1 and #2

- The left answer is my answer + constraint that there are no inheritance cycles not shown



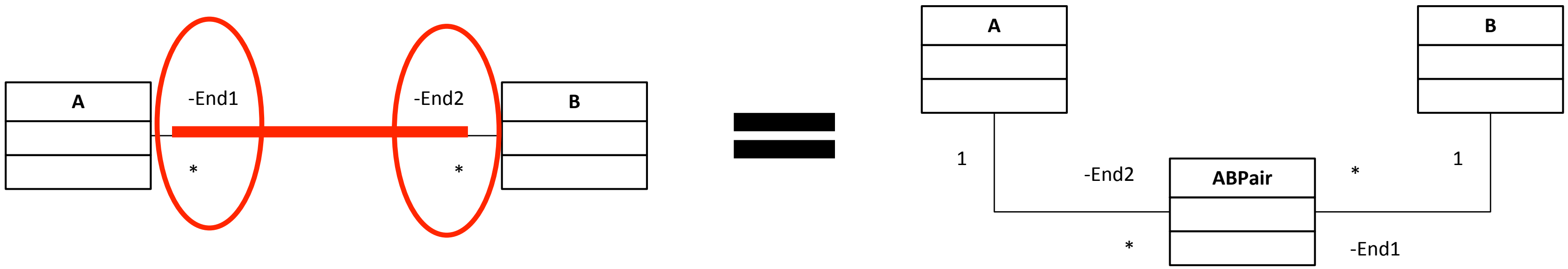


let's hope we agree on cardinality labeling, as it is non-standard.
I use Booch et al. UML convention

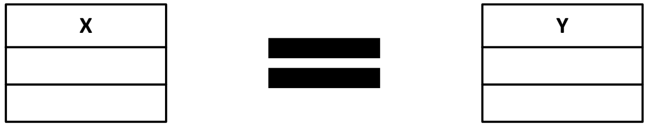
Graph Identities Known Beforehand

a.k.a. Refactorings

- A standard rewrite of database design circa mid-1980s called "normalize" association



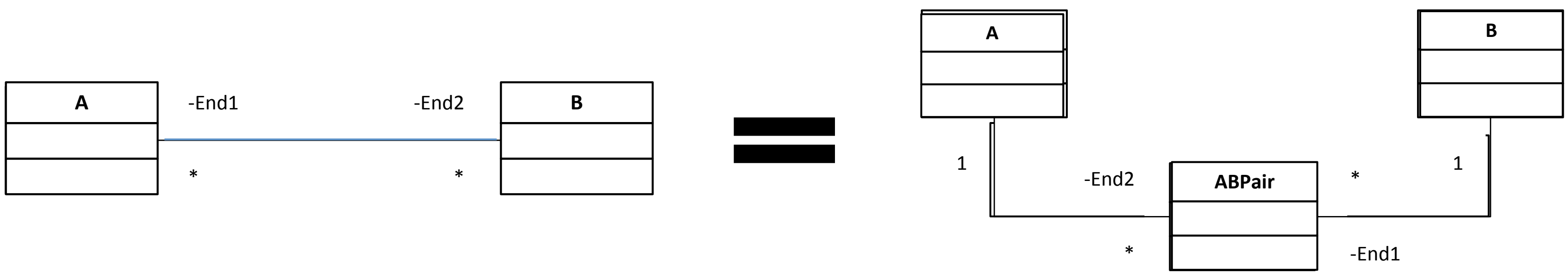
- And, of course, the rename refactoring which asserts "X" equals "Y"



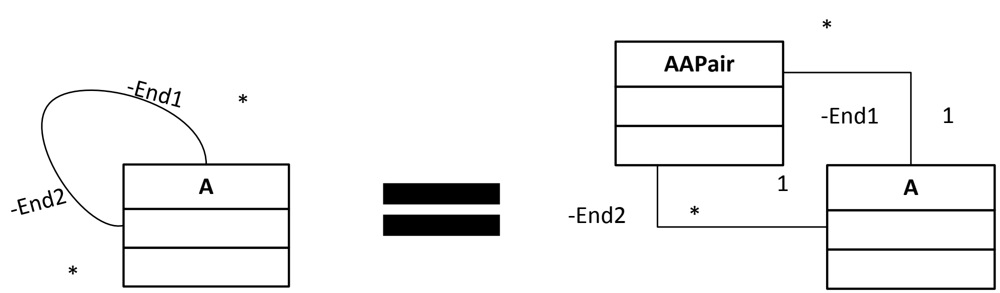
Graph Identities Known Beforehand

a.k.a. Refactorings

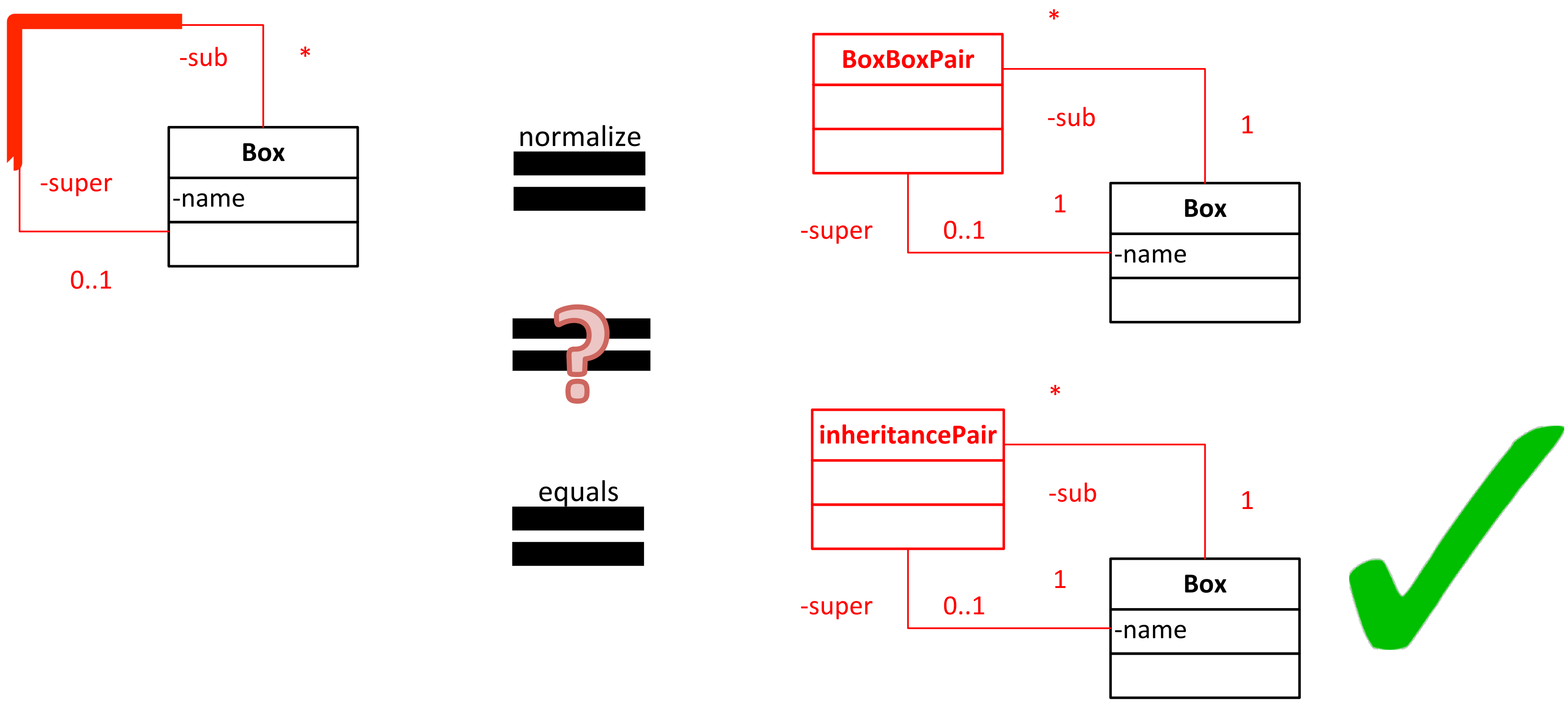
- A standard rewrite of database design circa mid-1980s called "normalize" association



- Special case where A = B

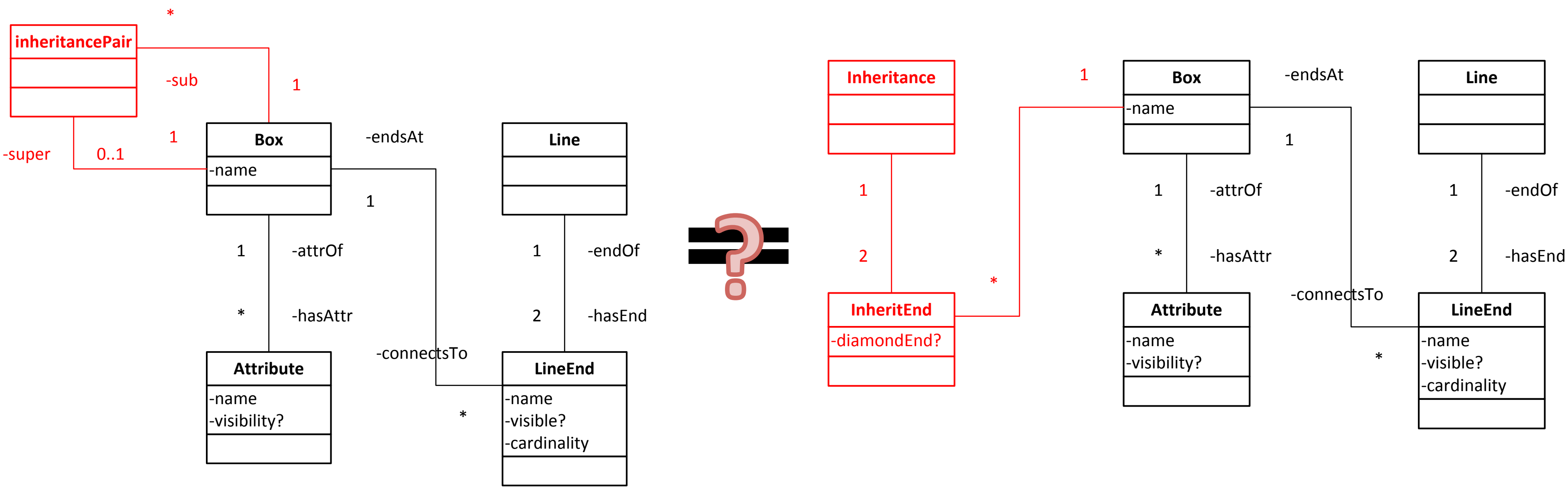


So Let's Derive their Equivalence



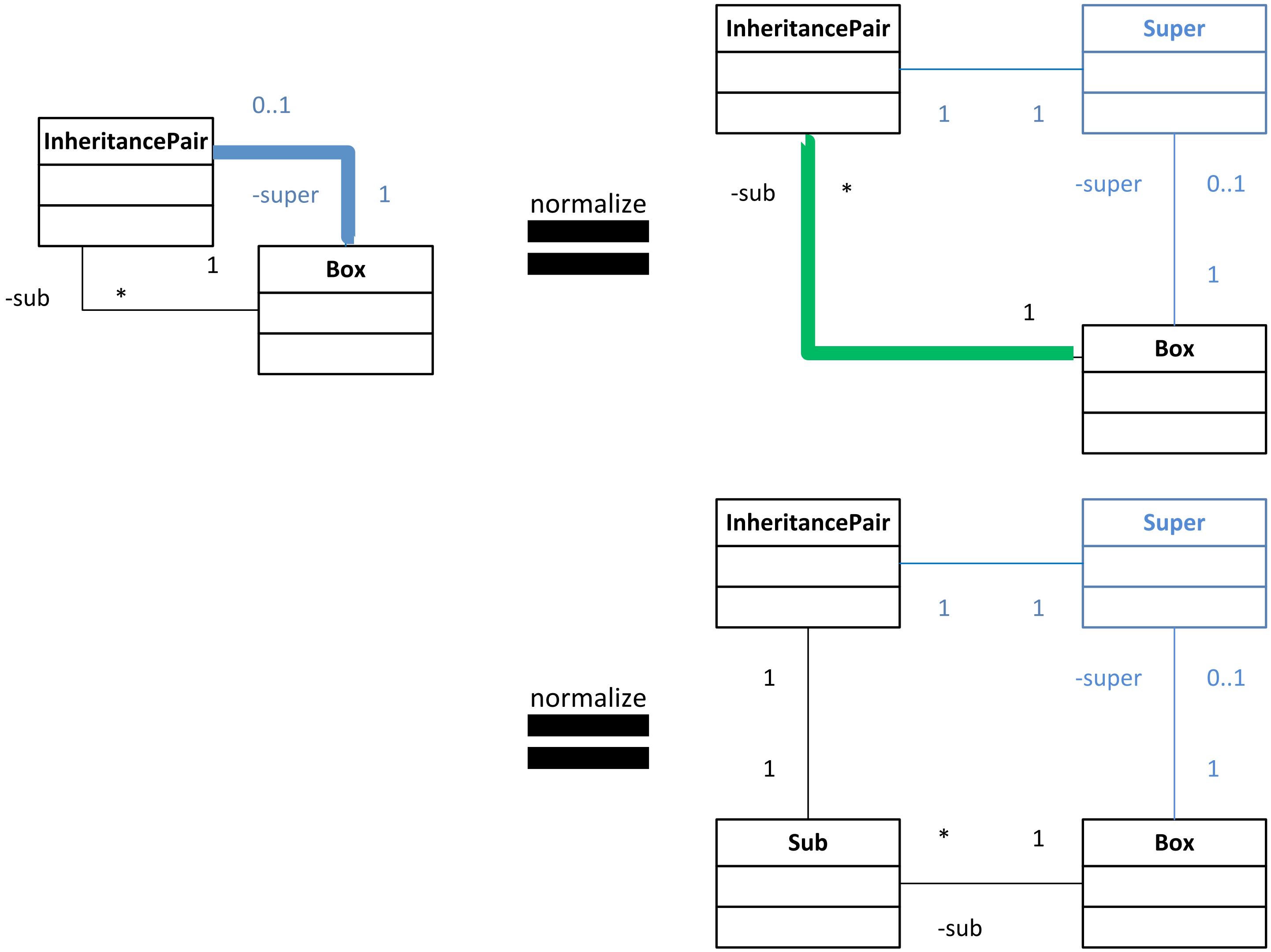
- Of course, the right diagram is more verbose than the left, but they are equivalent
- They don't get equivalent grades because the right CD is not minimal

Here's Another Student Answer

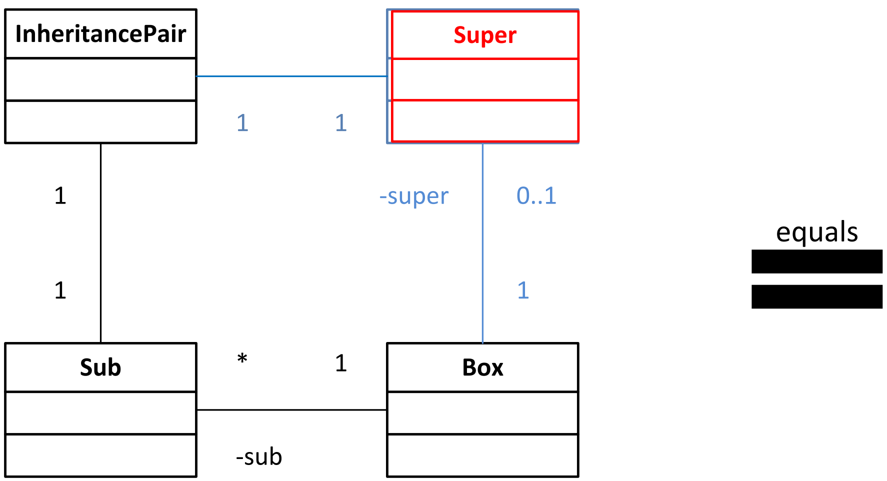


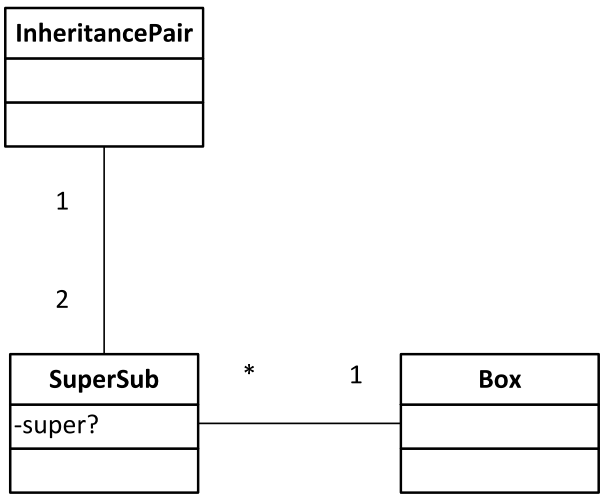
- Constraint on both diagrams: no box can have multiple super classes, no inheritance cycles, …

Again Use Normalization



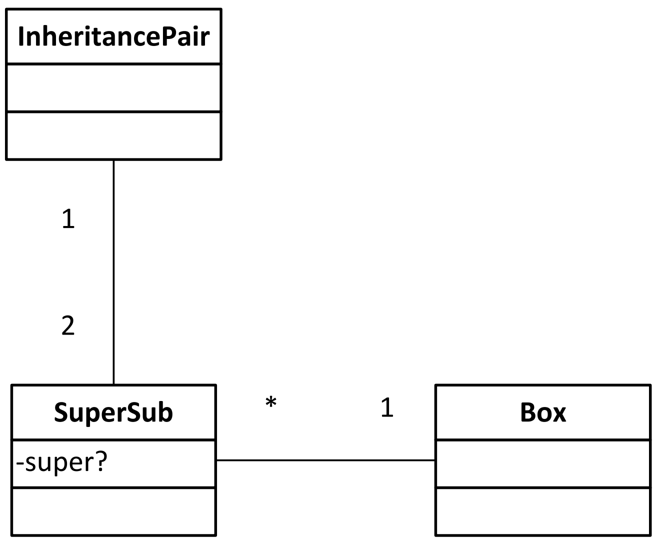
Next…





Each InheritancePair has precisely 1 "super" End and 1 "sub" end

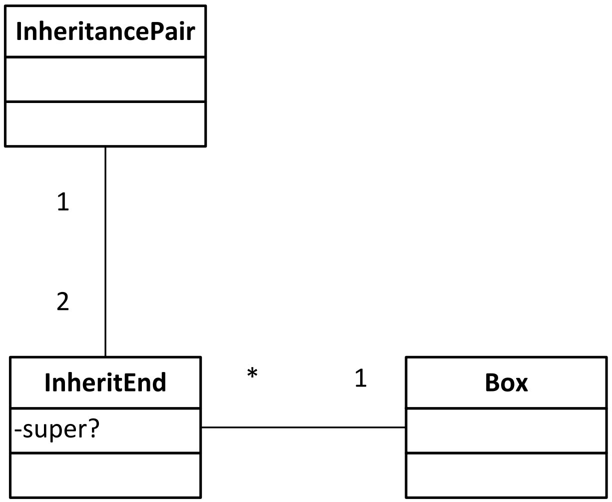
Last Step



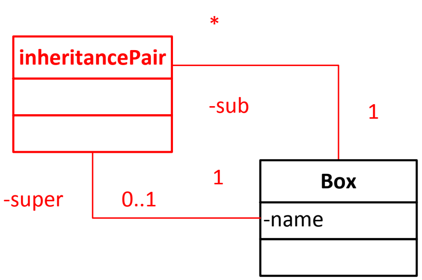
Each InheritancePair has precisely 1 “super” End and 1 “sub” end

equals

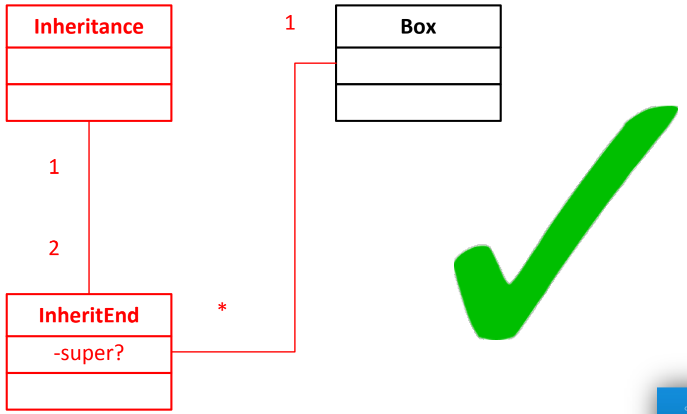
=



Each InheritancePair has precisely 1 “super” End and 1 “sub” end



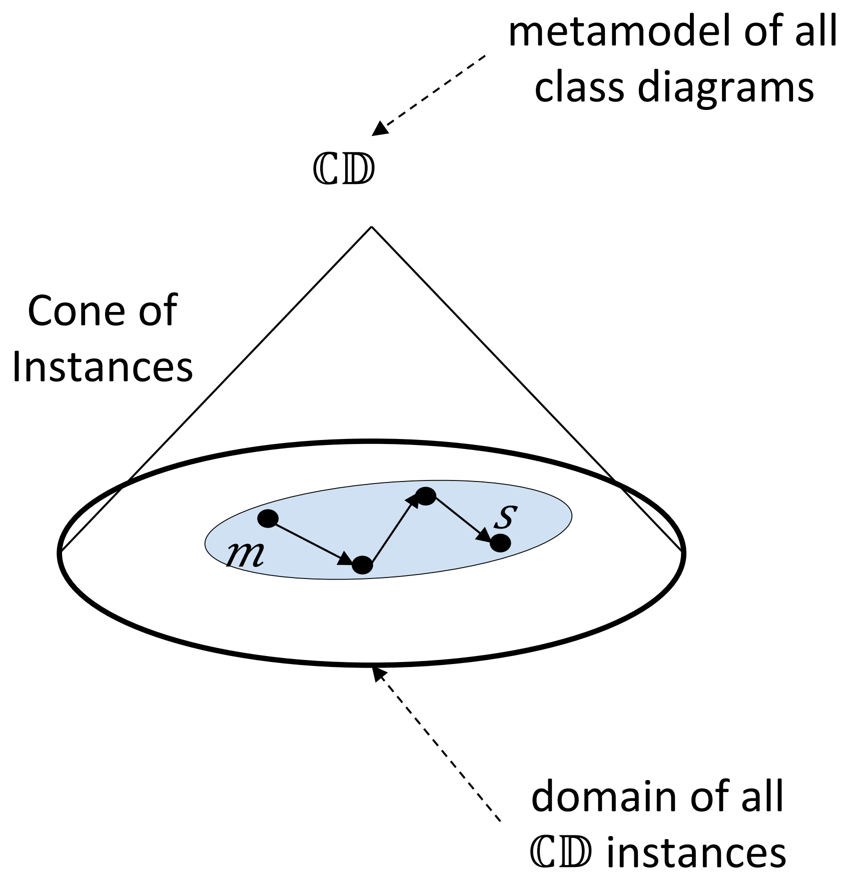
=



Ea Each Inheritance has precisely 1 “super” InheritEnd
And 1 “sub” InheritEnd

Big Picture

- Graph refactorings are Graph identities
- We should be teaching is the mathematics of software design – **this is the Science of Design**
- Variations in designs are explained by the application of graph identities
- Interesting assignments for students to think about what identities they need to grade each other's answers



CUTE
BUT ENGINEERS DON'T NEED THIS…

Really???

Word *of the* Day   Translate   Games   Blog

Dictionary.com

definitions   engineering

engineering

[en-*juh*-**neer**-ing]

Spell   Syllables

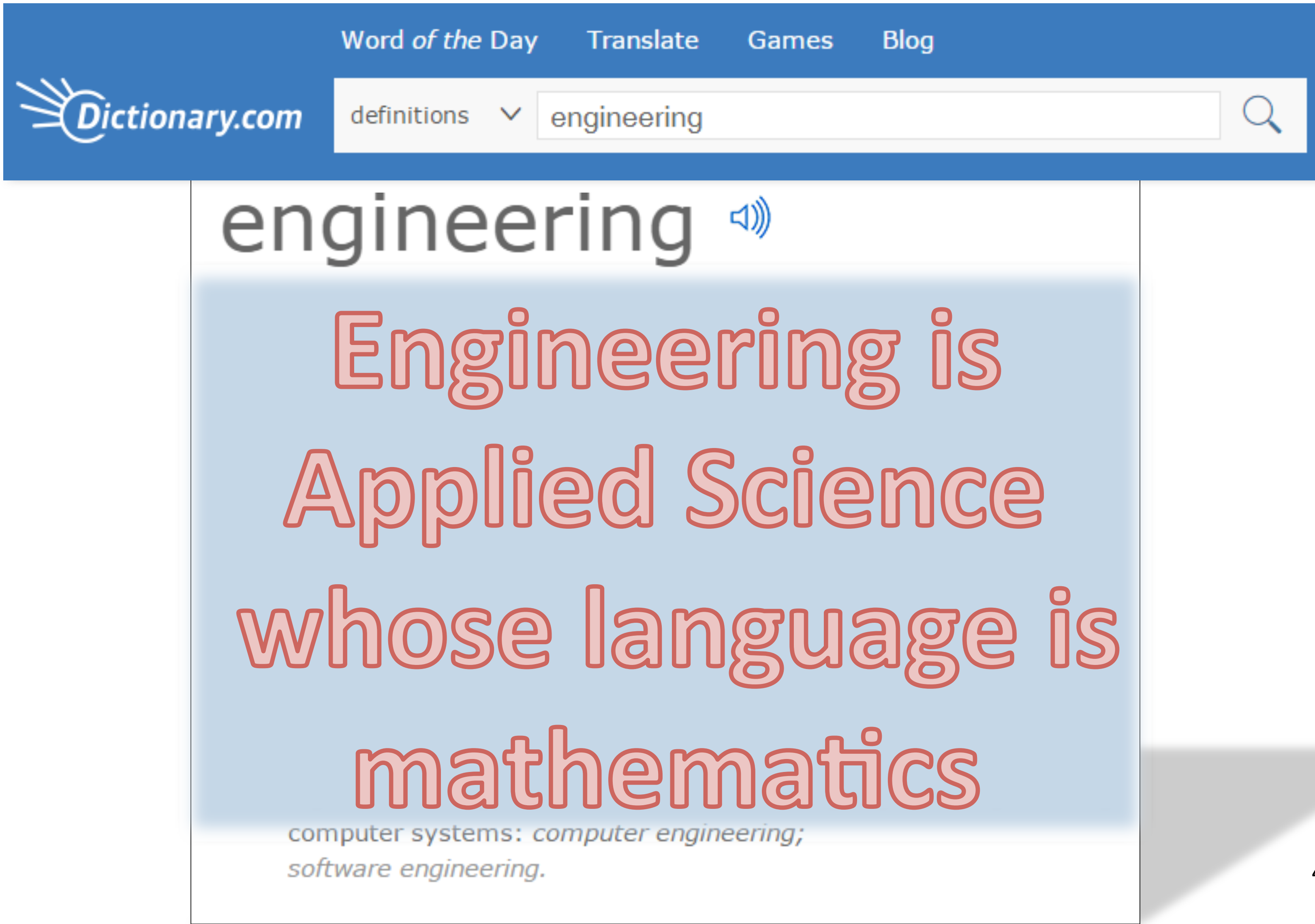
Examples   Word Origin

See more synonyms on Thesaurus.com

noun

1. the art or science of making practical application of the knowledge of pure sciences, as physics or chemistry, as in the construction of engines, bridges, buildings, mines, ships, and chemical plants.
2. the action, work, or profession of an engineer.
3. *Digital Technology.* the art or process of designing and programming computer systems: *computer engineering; software engineering.*

Really???



Engineering is Applied Science whose language is mathematics

Science is Everything in ASD

- It is the cooperation of theory and experiments
  - experiments give observational data
  - theory distills seemingly unrelated observations into a system of laws

- Go back in time to see the origins of **Automated Software Design (ASD)**

In ~1986, Keys to the Future of Software Development

- Paradigms of the future must embrace ***at least***:
  - **Compositional Programming**
    - develop software by composing "modules" *(not writing code)*
  - **Generative Programming**
    - want software development to be automated
  - **Domain-Specific Languages (DSLs)**
    - not C or C++, use domain-specific notations
  - **Automatic Programming**
    - declarative specs → efficient programs
  - **Verification**
    - want our programs to be correct
- Need simultaneous advance in all fronts to make a significant impact

Yeh, Right

- But … an example of this futuristic paradigm realized 7 years earlier (1979) around time when many AI researchers gave up on automatic programming

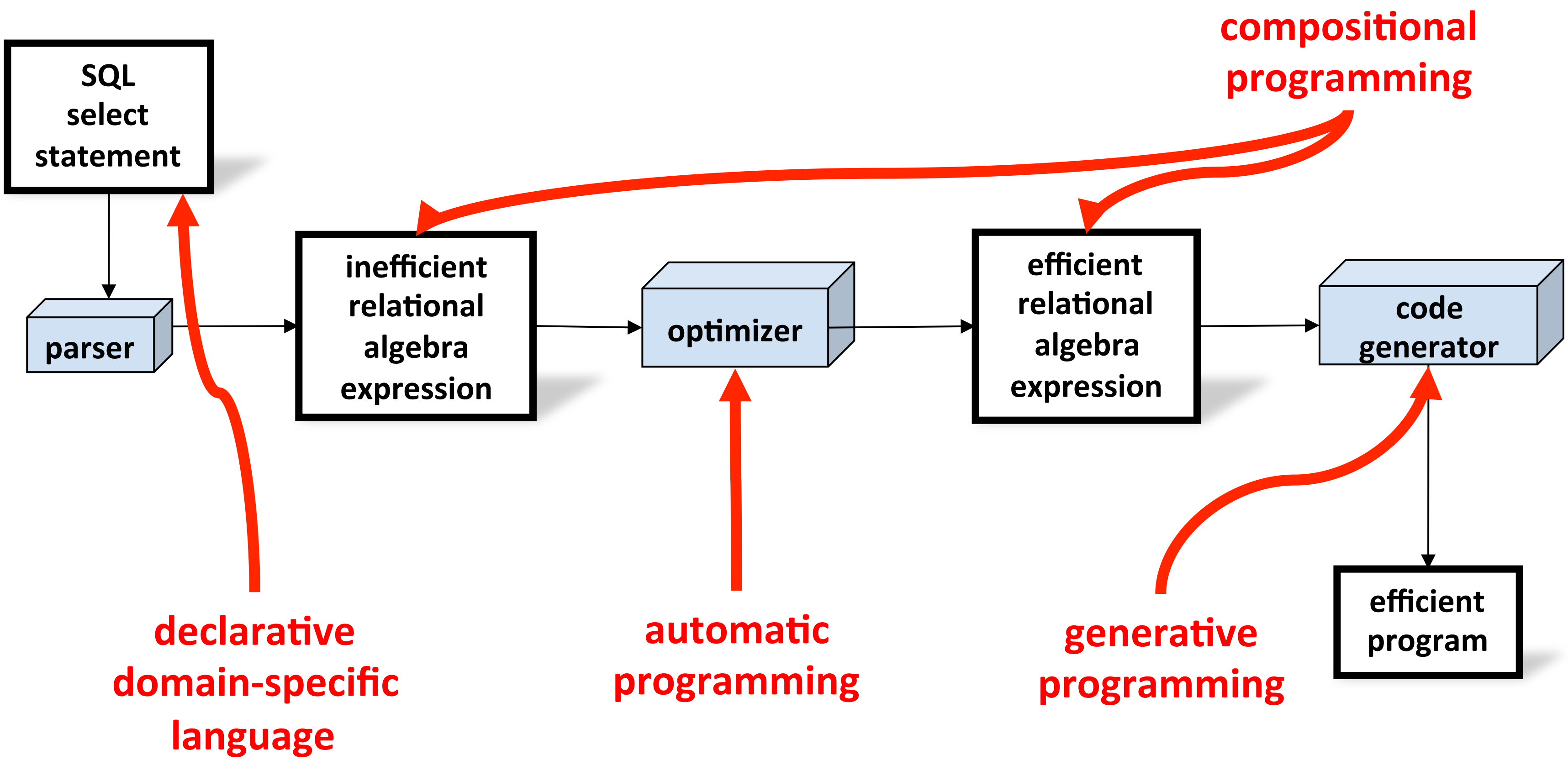
Relational Query Optimization

Selinger ACM SIGMOD 79

- <u>IMO – most significant result in ASD and automated construction. Period.</u>

- Rarely mentioned in typical texts and research papers in SE, software design, modularity, product lines, DSLs, MDE, software architectures…

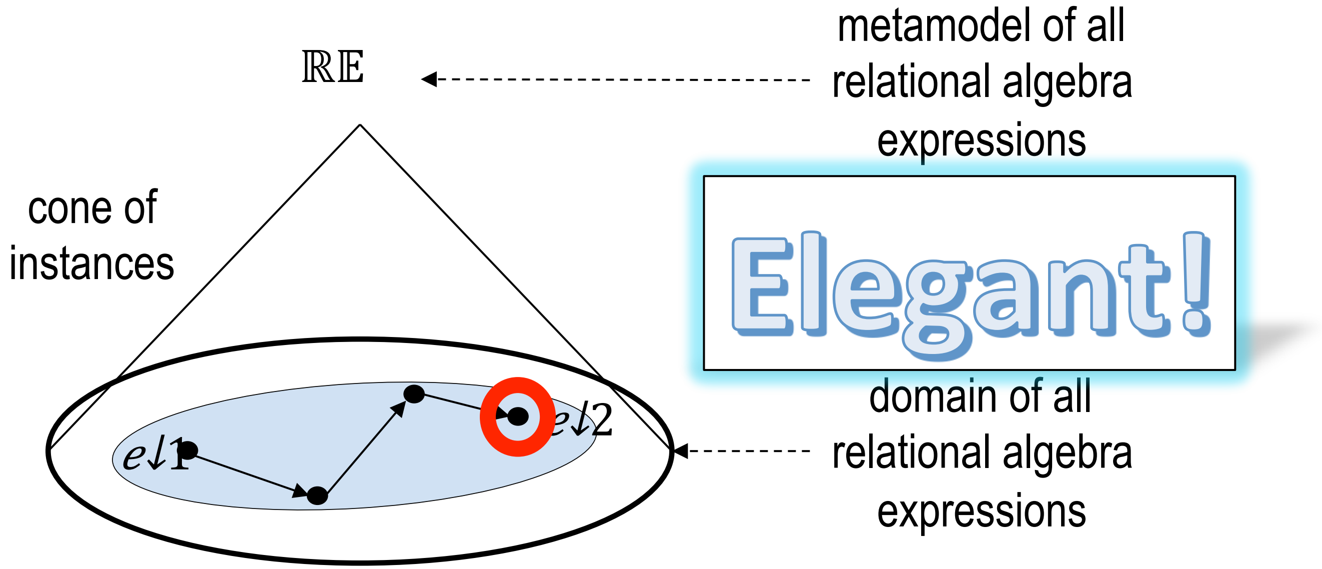
Relational Query Optimization (RQO)



correct-by-construction
(see this later)

What RQO Did

- Started with a simple relational algebra expression $e↓1$ derived from SQL SELECT
- Applying algebraic identities, created a subdomain of equivalent expressions, incl $e↓2$
- Ranked expressions by efficiency and chose the cheapest, ex: $e↓2$
- That's the implementation of the SQL SELECT to use



Keys to RQO Success

- Automated development of query evaluation programs
  - hard-to-write, hard-to-optimize, hard-to-maintain
  - revolutionized and simplified database usage
- Based on algebra of tables (not numbers)
  - different table expressions represented different programs
- Program designs / expressions can be optimized automatically
  - key is finding relational algebra identities
- Gave me a framework about how to think about ASD

NICE

SHOW ME SOMETHING USEFUL

Really??

- Revolutionizing database management was not useful?

- While you think about an answer,
  let me show others this example about dataflow applications…

Softw Syst Model
DOI 10.1007/s10270-014-0403-7

REGULAR PAPER

ReFlO: an interactive tool for pipe-and-filter domain specification and program generation

Rui C. Gonçalves · Don Batory · João L. Sobral

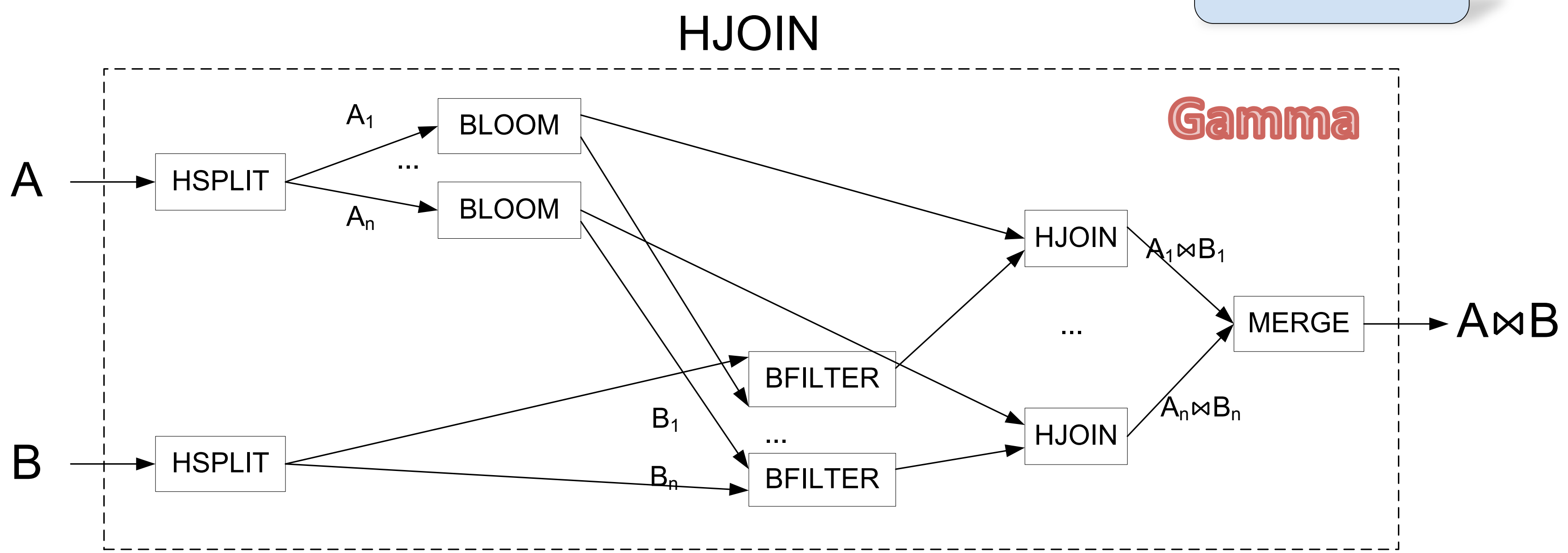
Received: 10 March 2013 / Revised: 17 January 2014 / Accepted: 3 February 2014

Abstract `ReFlO` is a framework and interactive tool to record and systematize domain knowledge used by experts to derive complex *pipe-and-filter (PnF)* applications. Domain knowledge is encoded as transformations that alter PnF graphs by *refinement* (adding more details), *flattening* (removing modular boundaries), and *optimization* (substituting inefficient PnF graphs with more efficient ones). All three software development and, like actual circuit design tools, can express hierarchical systems by levels of abstraction: a component at level `i` is defined in terms of a circuit of more primitive components at level `i + 1`, recursively. CBSE is an early example of *Model Driven Engineering (MDE)* where models (i.e. hierarchical circuit diagrams) are transformed into executables.

How Do You Explain…

- This spaghetti diagram: it is a dataflow graph of a fundamental parallel hash join algorithm, similar to what is used in database machines today

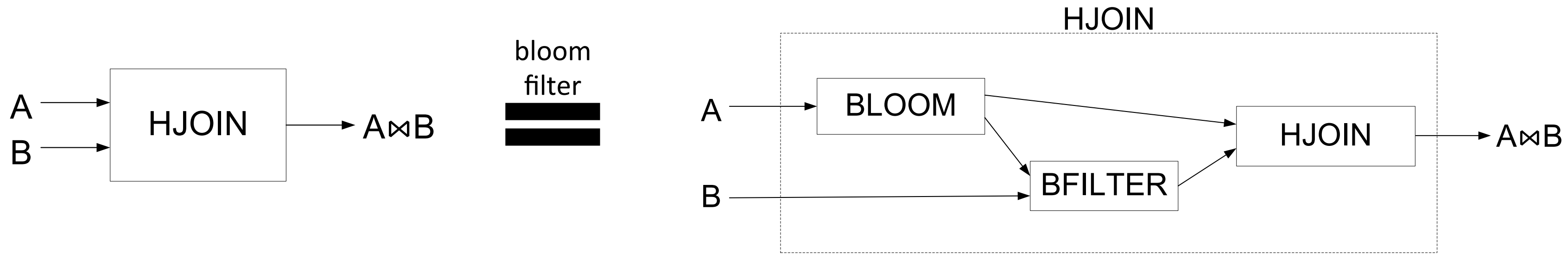
DeWitt, et al.
IEEE TKDE 1990



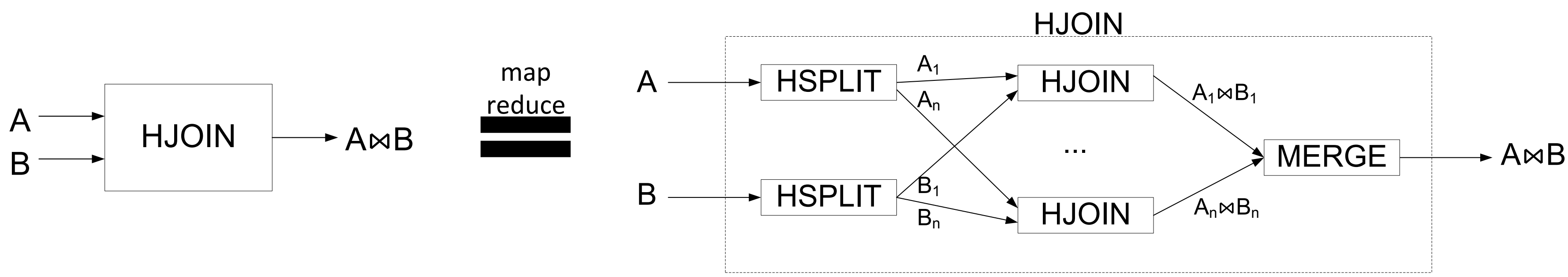
- To explain & derive it, you need data flow graph identities

Simple Way To Derive Gamma

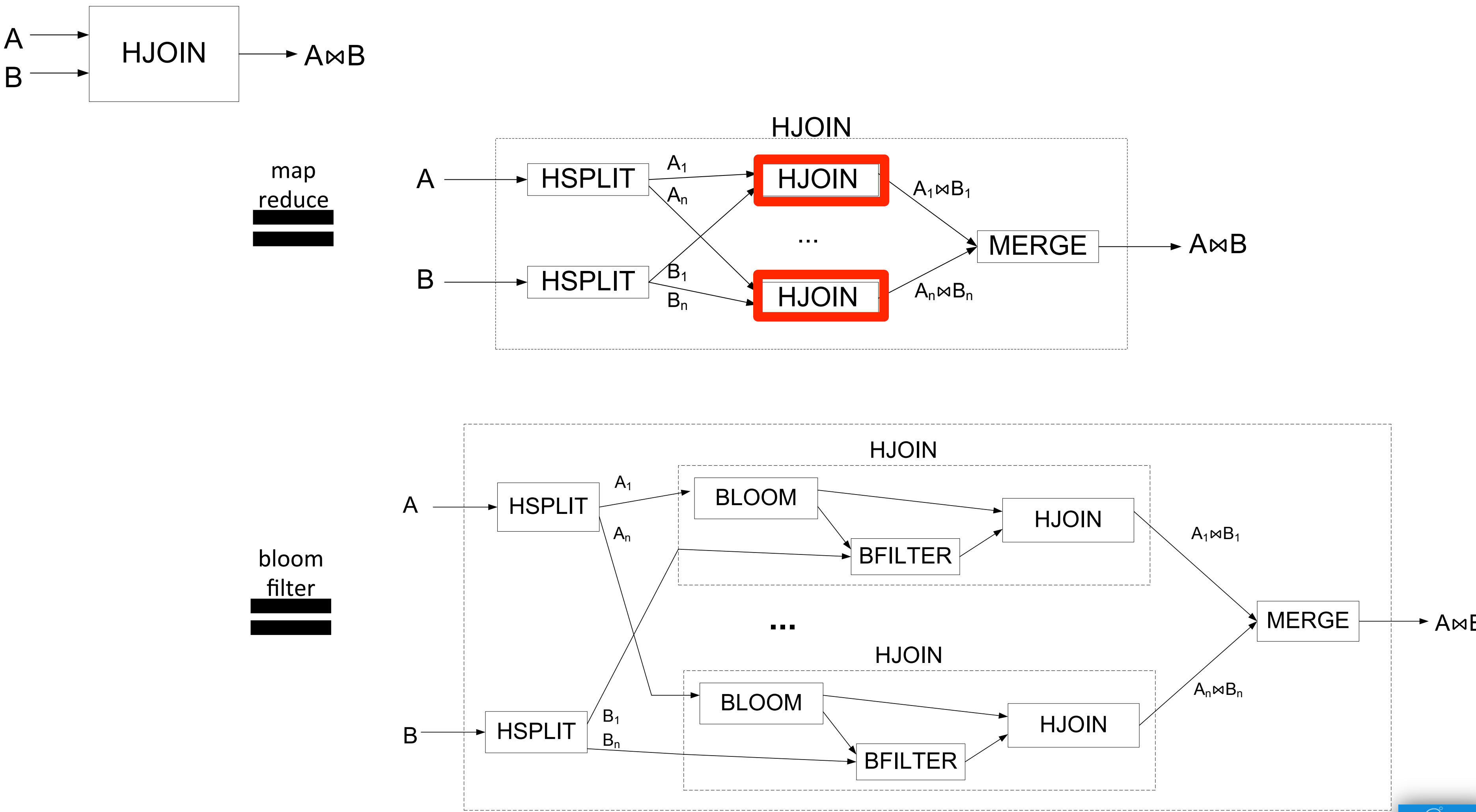
- Need 2 identities that are well-known to database researchers but few others
- Bloom filters remove tuples from stream B that provably cannot join with stream A



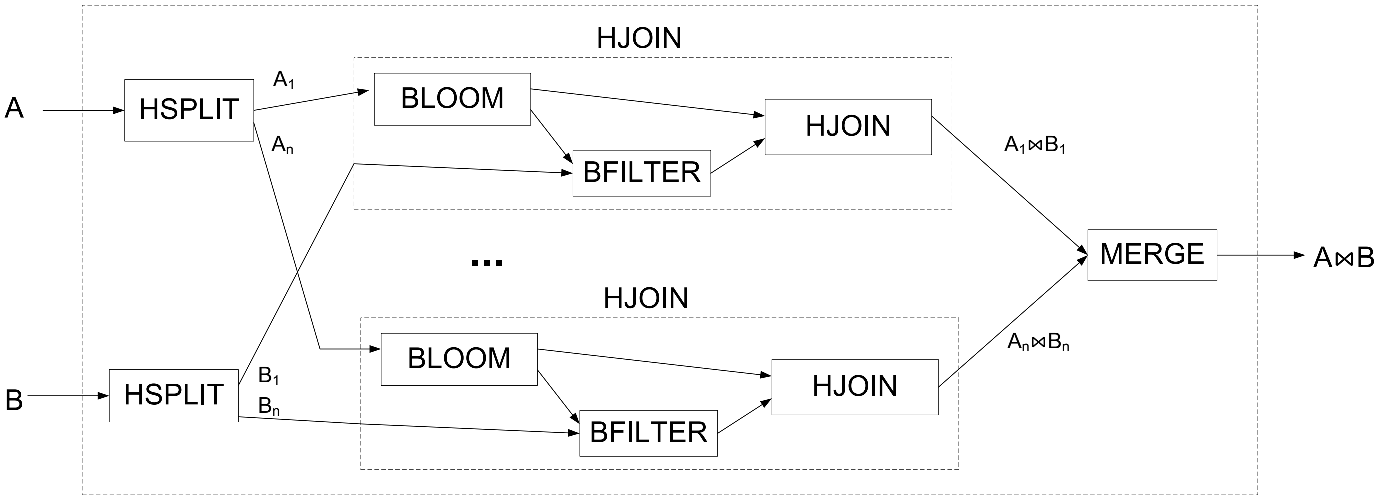
- Parallelize HJOIN operation via map-reduce:



Derivation of Gamma

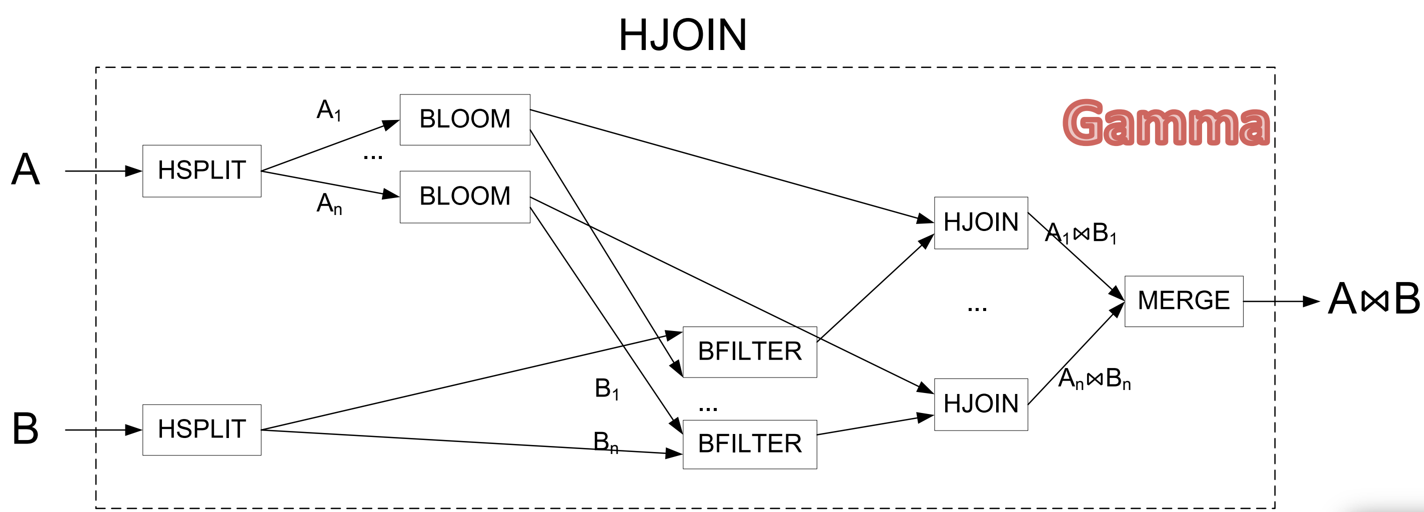


Derivation of Gamma



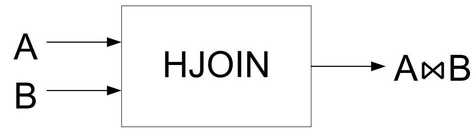
graph rearrangement

=

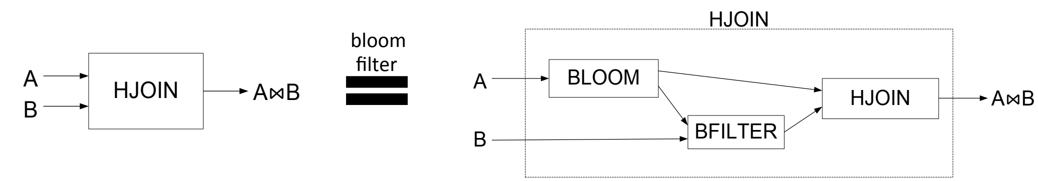


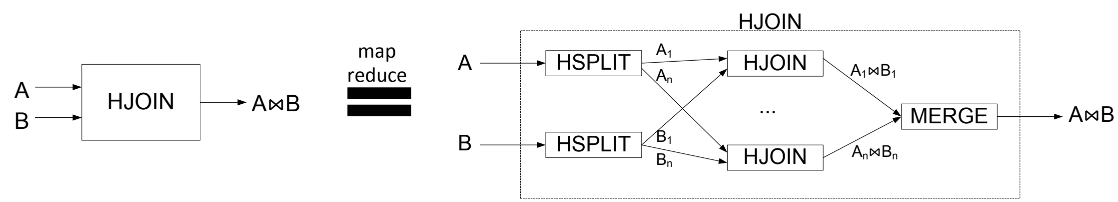
Design is Correct By Construction

- Initial graph is correct

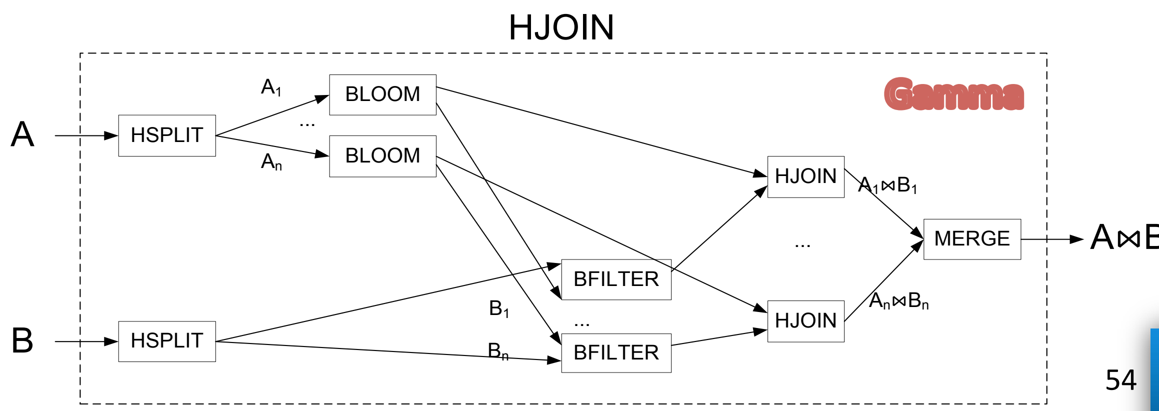


- Rewrites are correct



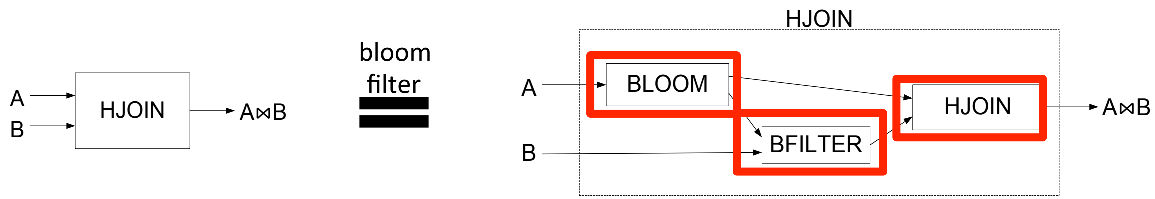


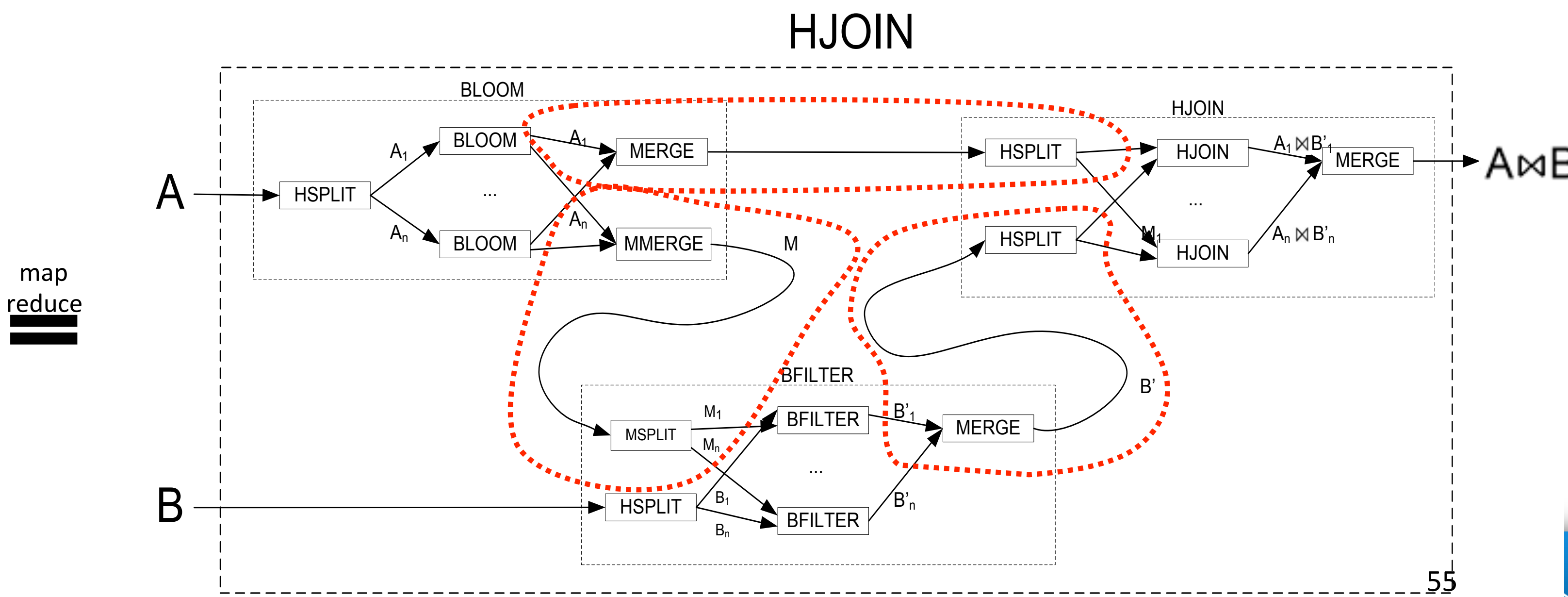
- End result is correct



Remember!

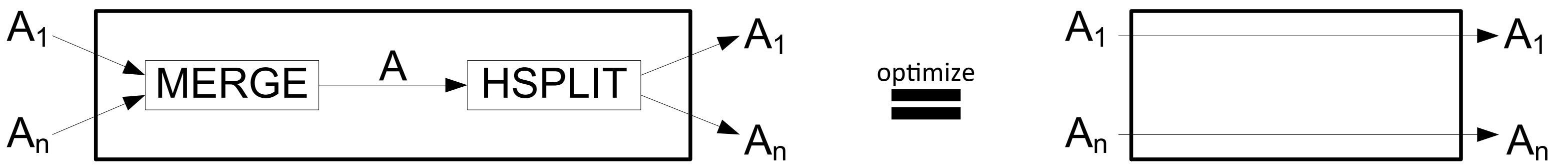
- There are many ways to derive the same graph
  - simple exploration of this space reveals other fundamental identities



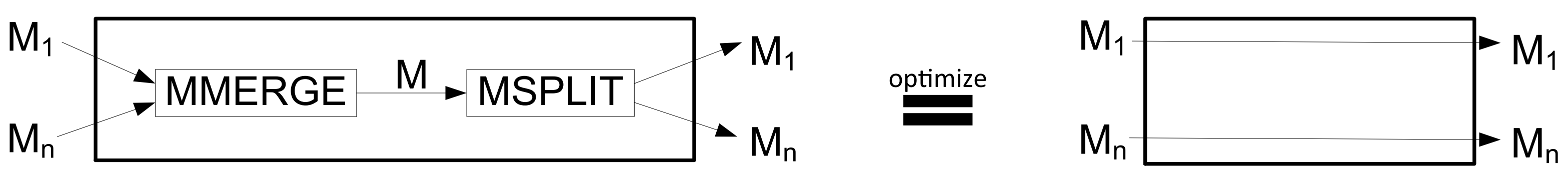


Identity Optimizations

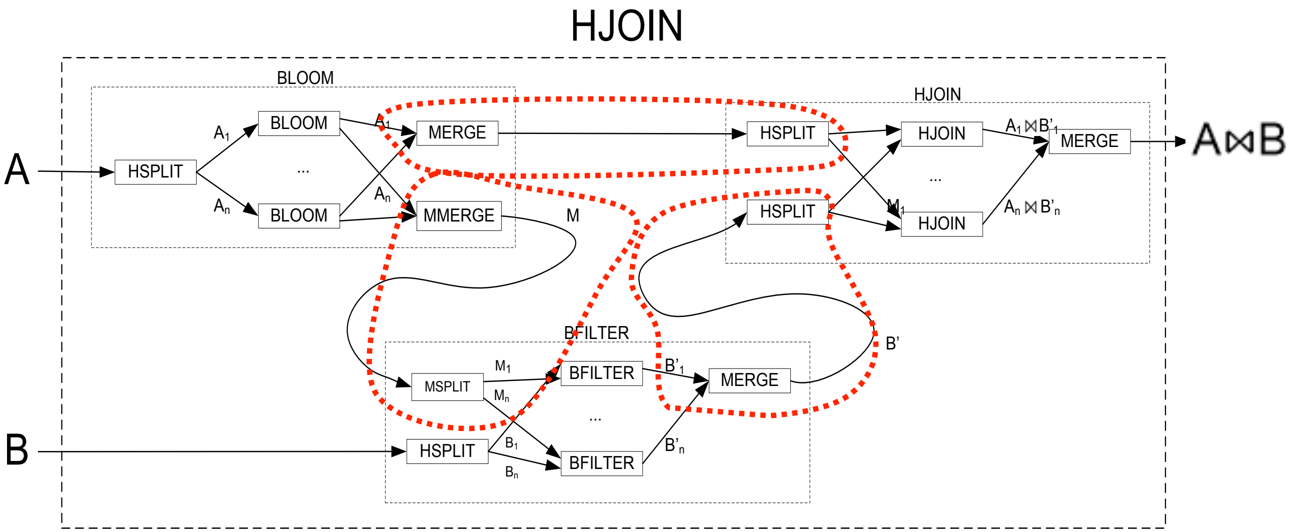
- Merge tuple streams $A\downarrow 1$ ...$A\downarrow n$ into A and then reconstitute them



- Merge bitmaps $M\downarrow 1$ ...$M\downarrow n$ into a single bitmap M and recreate bitmaps $M\downarrow 1$ ... $M\downarrow n$

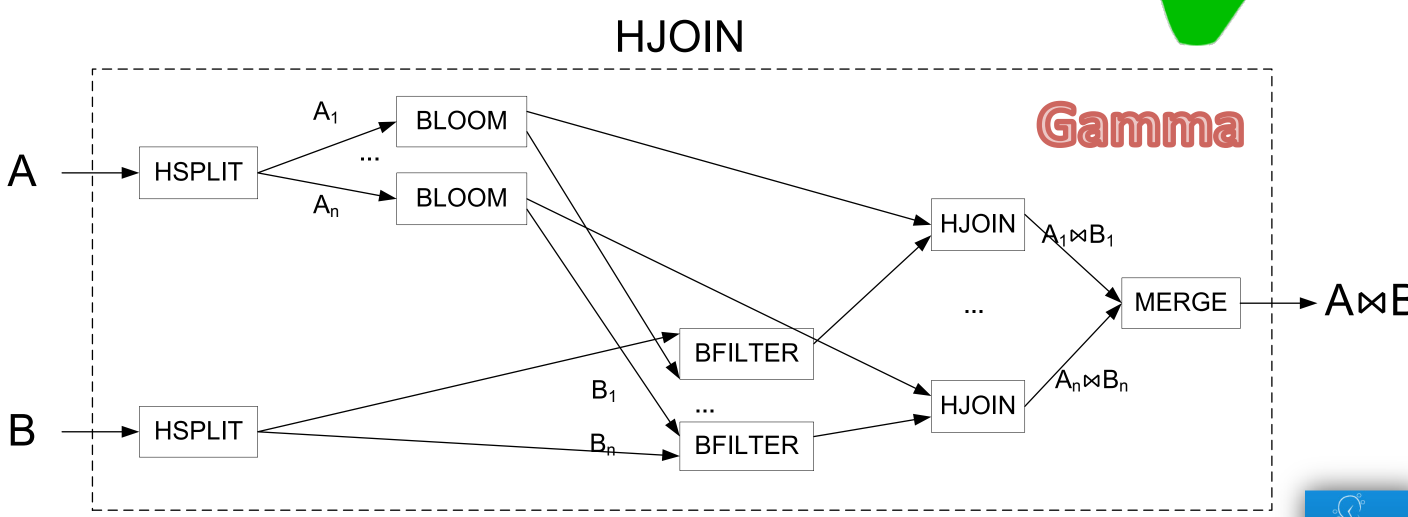


Derivation



This is another way to discover graph identities.

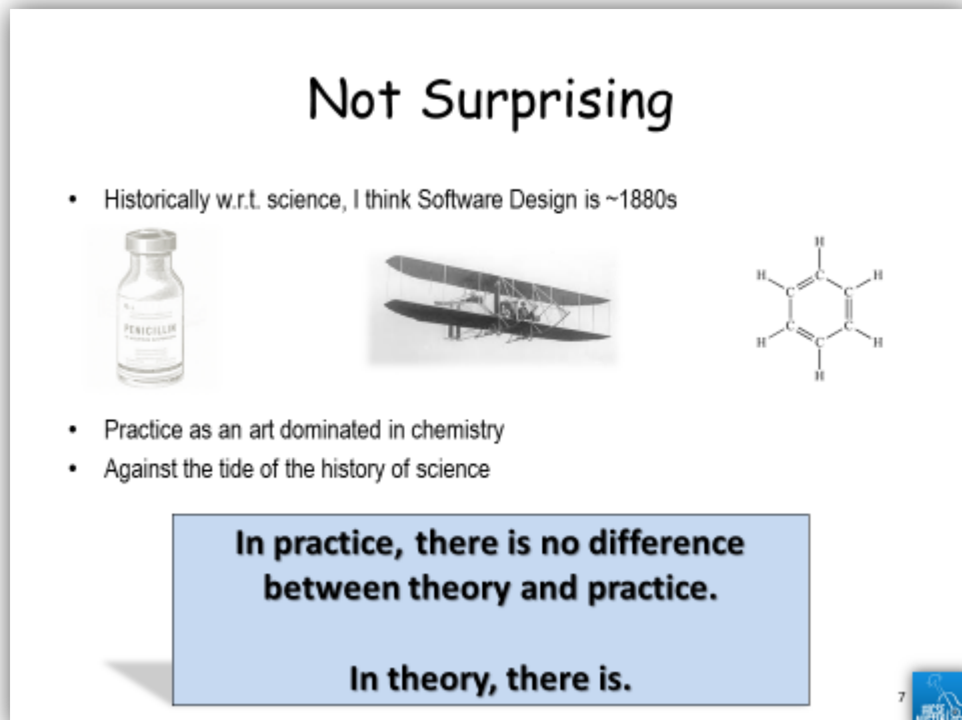
optimize



DON
TOO FANCY FOR ME;
I PREFER MY WAY

I Understand…

- That's exactly what Chemists said in the 1880s…
  - but this will change and will take time…
  - if it were easy, would have been done years ago



Let me show you a difference between manual software development and automated design in another fundamental area of CS

Dense Linear Algebra (DLA)

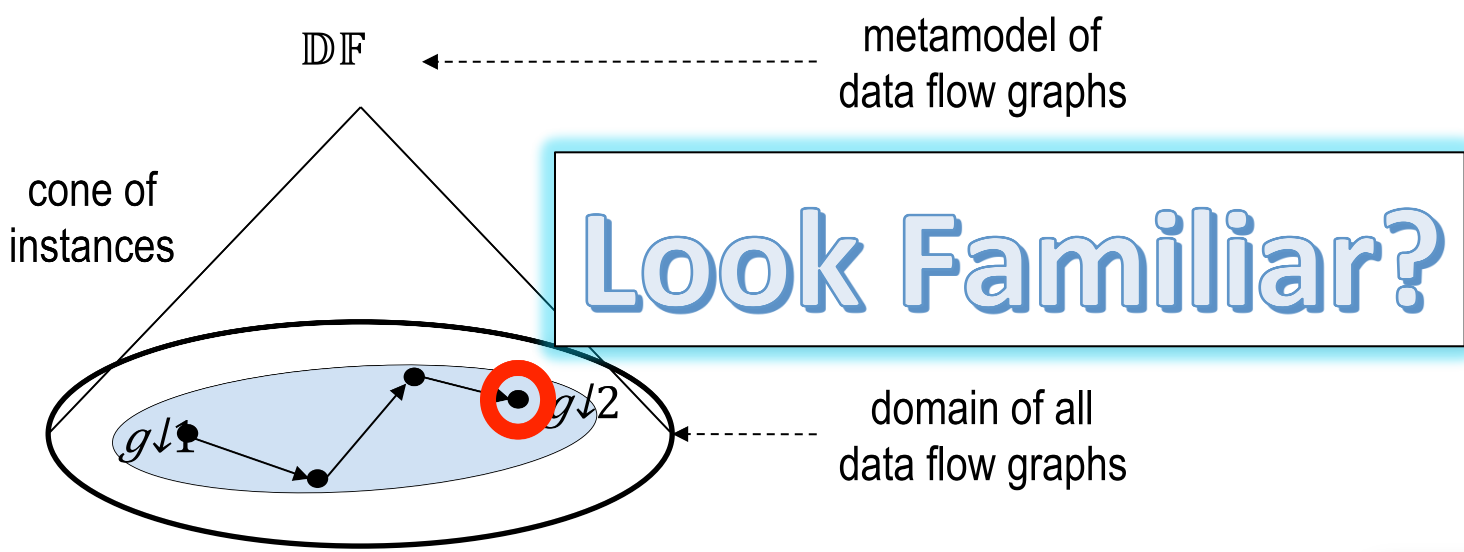
- Robert van de Geijn
  - last 30 years creating elegant mathematically layered designs of DLA computations

- Jack Paulson created Elemental Distributed DLA package
  - standard BLAS3 matrix-matrix operations
  - solvers
  - decomposition functions (Cholesky factorization)
  - eigenvalue problems

- Bryan Marker mechanized the above work as application of dataflow identities
  - DxTer name of his tool

What DxTer Does

- Starts with a simple DLA dataflow graph g_1 specified by library designer or DLA user
- Applies algebraic identities & creates a subdomain of equivalent graphs, incl g_2
- Ranks graphs by their estimated efficiency and choses the cheapest, ex: g_2
- That's the implementation that is translated to code

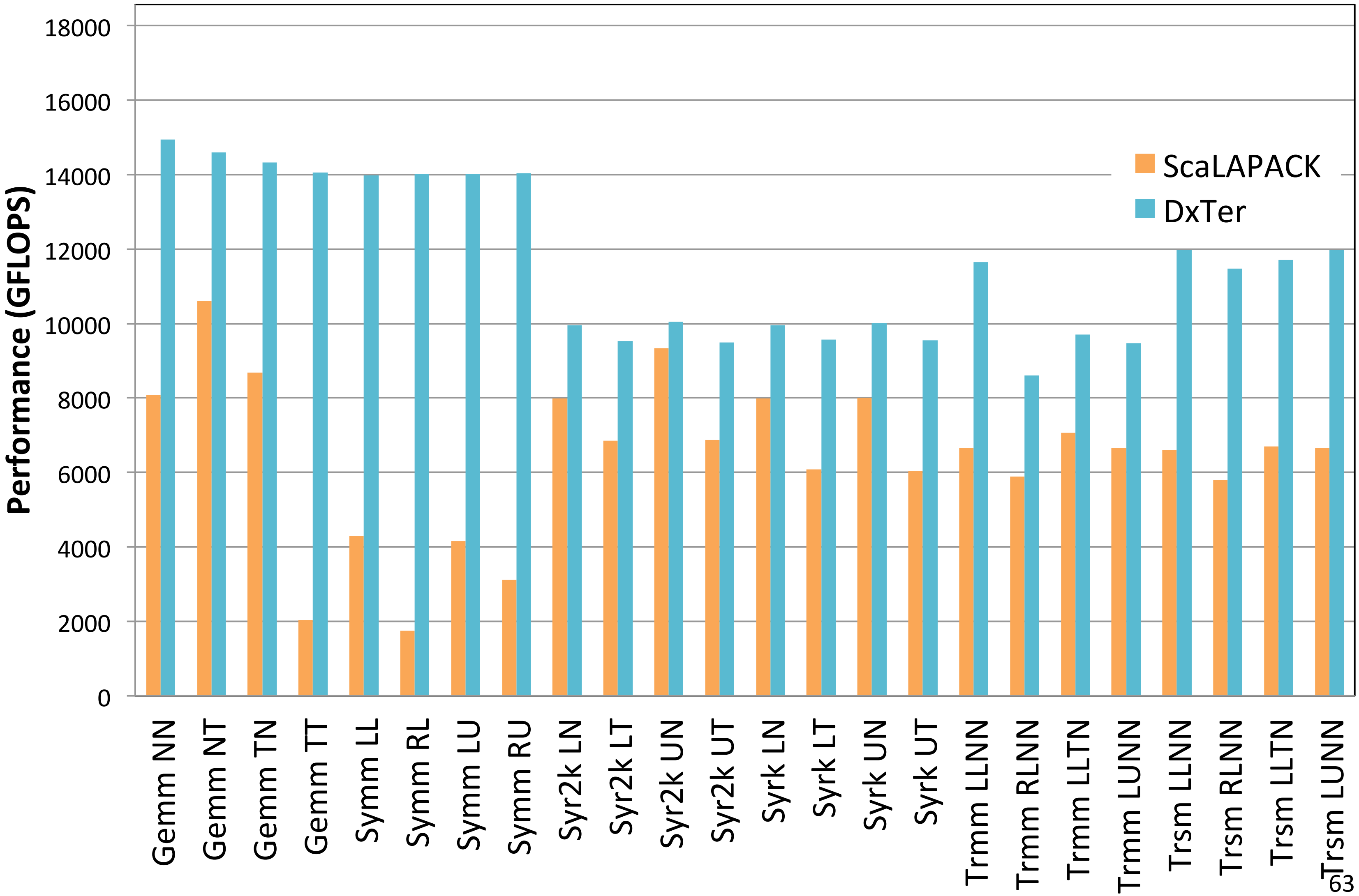


Performance Results

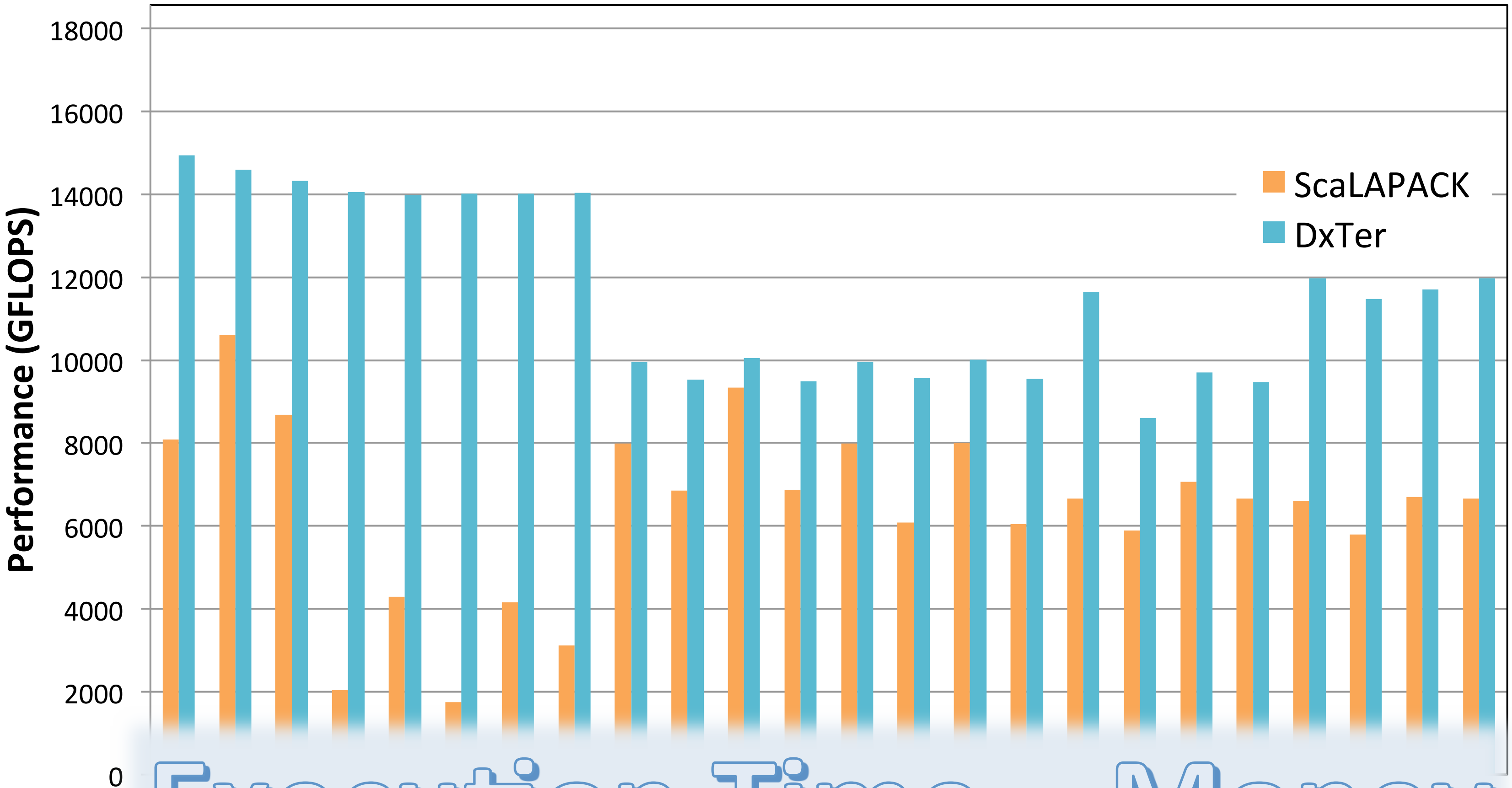
Machine	# of Cores	Peak Performance
Argonne's BlueGene/P (Intrepid)	8,192	27+ TFLOPS
Texas Advanced Computing Center (Lonestar)	240	3.2 TFLOPS

- DxTer automatically generated & optimized Elemental code for BLAS3 and Cholesky operations
- Benchmarked with manually-written ScaLAPACK
  - vendors standard option for distributed memory machines; auto-tuned or manually-tuned
  - only alternative available for target machines

BLAS3 Performance on Intrepid



BLAS3 Performance on Intrepid



Execution Time = Money
this is Huge

Bryan Found

- Error(s) in Elemental Library
- Instances where the Domain Expert Jack forgot to apply an optimization
- Or used the wrong algorithm (performance error)

DxTer-Generated code is being shipped with Elemental

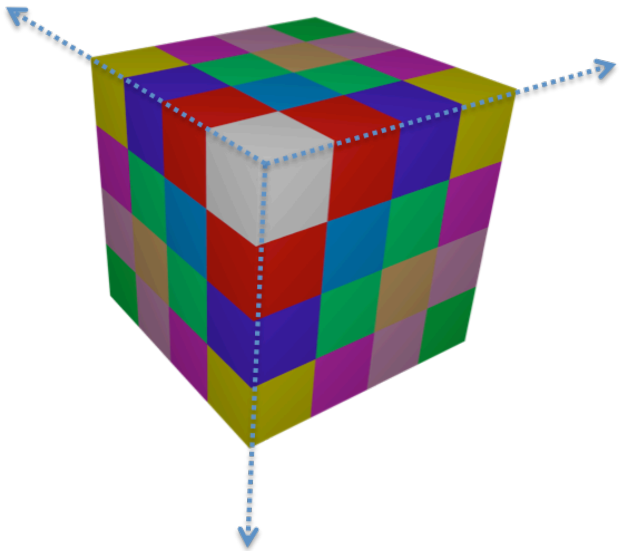
What is Really Important…

- New hardware architectures are invented every year
- DLA algorithms that are optimized for 1 architecture are not optimized for another…
- Porting DLA libraries either
    - runs slower than optimal – which is costly
    - rewrite much from scratch – which is costly

DxTer is a long-term, cost-effective, & scalable way to customize DLA libraries

Next Stop - Tensors!

$$\mathcal{A} \in \mathbb{R}^{[3,4]}$$



- Tensor
  - n – dimensional array
- Tensor Contraction
  - generalization of matrix multiplication
- Tensors = matrices on steroids
- Based on ROTE Library of Martin Schatz

Next Stop

- Generalized computations to tensor equations in Computational Chemistry
- Here are the CCSD coupled cluster single double equations for accurate reproduction of experimental results on electron correlation for molecules
- Bryan & Martin created a dataflow graph of these equations and DxTer to optimize their implementation

$$
\begin{aligned}
W^{bm}_{je} &= (2w^{bm}_{je} - x^{bm}_{ej}) + \sum_{f}(2r^{bm}_{fe} - r^{bm}_{ef})t^{f}_{j} - \sum_{n}(2u^{nm}_{je} - u^{mn}_{je})t^{b}_{n} \\
&\quad + \sum_{fn}(2v^{fe}_{nm} - v^{fe}_{mn})(T^{bf}_{jn} + \frac{1}{2}T^{bf}_{nj} - \tau^{bf}_{nj}) \\
X^{bm}_{ej} &= x^{bm}_{ej} + \sum_{f} r^{bm}_{ef}t^{f}_{j} - \sum_{n} u^{mn}_{je}t^{b}_{n} - \sum_{fn} v^{fe}_{mn}(\tau^{bf}_{nj} - \frac{1}{2}T^{bf}_{nj}) \\
U^{mn}_{ie} &= u^{mn}_{ie} + \sum_{f} v^{fe}_{mn}t^{f}_{i} \\
Q^{mn}_{ij} &= q^{mn}_{ij} + (1 + P^{mi}_{nj}) \sum_{e} u^{mn}_{ie}t^{e}_{j} + \sum_{ef} v^{ef}_{mn}\tau^{ef}_{ij} \\
P^{ji}_{mb} &= u^{ji}_{mb} + \sum_{ef} r^{bm}_{ef}\tau^{ef}_{ij} + \sum_{e} w^{bm}_{ie}t^{e}_{j} + \sum_{e} x^{bm}_{ej}t^{e}_{i} \\
H^{m}_{e} &= \sum_{fn}(2v^{ef}_{mn} - v^{ef}_{nm})t^{f}_{n} \\
F^{a}_{e} &= -\sum_{m} H^{m}_{e}t^{a}_{m} + \sum_{fm}(2r^{am}_{ef} - r^{am}_{fe})t^{f}_{m} - \sum_{fmn}(2v^{ef}_{mn} - v^{ef}_{nm})T^{af}_{mn} \\
G^{m}_{i} &= \sum_{e} H^{m}_{e}t^{e}_{i} + \sum_{en}(2u^{mn}_{ie} - u^{nm}_{ie})t^{e}_{n} + \sum_{efn}(2v^{ef}_{mn} - v^{ef}_{nm})T^{ef}_{in} \\
z^{a}_{i} &= -\sum_{m} G^{m}_{i}t^{a}_{m} - \sum_{emn}(2U^{mn}_{ie} - U^{nm}_{ie})T^{ae}_{mn} + \sum_{em}(2w^{am}_{ie} - x^{am}_{ei})t^{e}_{m} \\
&\quad + \sum_{em}(2T^{ae}_{im} - T^{ae}_{mi})H^{m}_{e} + \sum_{efm}(2r^{am}_{ef} - r^{am}_{fe})\tau^{ef}_{im} \\
Z^{ab}_{ij} &= v^{ab}_{ij} + \sum_{mn} Q^{mn}_{ij}\tau^{ab}_{mn} + \sum_{ef} y^{ab}_{ef}\tau^{ef}_{ij} + (1 + P^{ai}_{bj}) \Big\{ \sum_{e} r^{ej}_{ab}t^{e}_{i} \\
&\quad - \sum_{m} P^{ij}_{mb}t^{a}_{m} + \sum_{e} F^{a}_{e}T^{eb}_{ij} - \sum_{m} G^{m}_{i}T^{ab}_{mj} + \frac{1}{2}\sum_{em} W^{bm}_{je}(2T^{ae}_{im} - T^{ae}_{mi}) \\
&\quad - (\frac{1}{2} + P^{i}_{j}) \sum_{em} X^{bm}_{ej}T^{ae}_{mi} \Big\}
\end{aligned}
$$

Next Step

- Generalized computations to tensor equations in Computa
- Here are coupled equation reproduc results o for molecules
- Marker created a dataflow graph of these equations and DxTer to optimize their implementation

$$W^{bm}_{je} = (2w^{bm}_{je} - x^{bm}_{ej}) + \sum_f (2r^{bm}_{fe} - r^{bm}_{ef}) t^f_j - \sum_n (2u^{nm}_{je} - u^{mn}_{je}) t^b_n + \sum_{fn} (2v^{fe}_{nm} - v^{fe}_{mn})(T^{bf}_{jn} + \frac{1}{2} T^{bf}_{nj} - \tau^{bf}_{nj})$$

$$X^{bm}_{ej} = x^{bm}_{ej} + \sum_f r^{bm}_{ef} t^f_j - \sum_n u^{mn}_{je} t^b_n - \sum_{fn} v^{fe}_{mn}(\tau^{bf}_{nj} - \frac{1}{2} T^{bf}_{nj})$$

$$U^{mn}_{ie} = u^{mn}_{ie} + \sum_f v^{fe}_{mn} t^f_i$$

Nontrivial Space

O(10↑22

Searched in ~11sec

$$z^a_i = -\sum_m G^m_i t^a_m - \sum_{emn} (2U^{mn}_{ie} - U^{nm}_{ie}) T^{ae}_{mn} + \sum_{em} (2w^{am}_{ie} - x^{am}_{ei}) t^e_m + \sum_{em} (2T^{ae}_{im} - T^{ae}_{mi}) H^m_e + \sum_{efm} (2r^{am}_{ef} - r^{am}_{fe}) \tau^{ef}_{im}$$

$$Z^{ab}_{ij} = v^{ab}_{ij} + \sum_{mn} Q^{mn}_{ij} \tau^{ab}_{mn} + \sum_{ef} y^{ab}_{ef} \tau^{ef}_{ij} + (1 + P^{ai}_{bj}) \Big\{ \sum_e r^{ej}_{ab} t^e_i - \sum_m P^{ij}_{mb} t^a_m + \sum_e F^a_e T^{eb}_{ij} - \sum_m G^m_i T^{ab}_{mj} + \frac{1}{2} \sum_{em} W^{bm}_{je} (2T^{ae}_{im} - T^{ae}_{mi}) - (\frac{1}{2} + P^i_j) \sum_{em} X^{bm}_{ej} T^{ae}_{mi} \Big\}$$

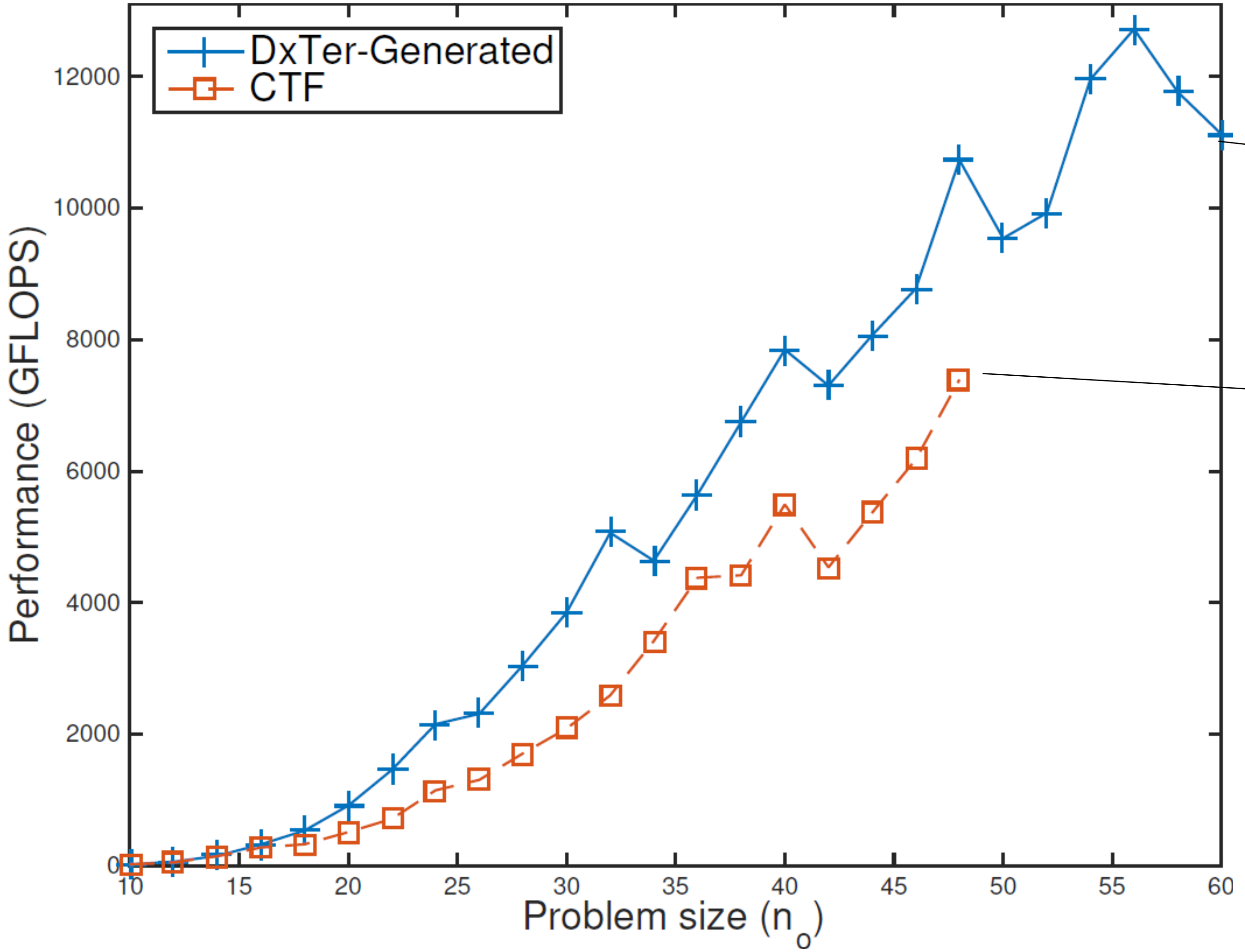
State of the Art

- Contestant: **CTF** – Cyclops Tensor Framework
  - state-of-the art distributed library for tensor computations
  - performs one contraction (tensor multiply) at a time
  - chooses among different algorithms

- Machine: Benchmark on BlueGene/Q
  - 16 shared-memory cores of IBM's 64-bit Power A2 architecture @ 1600MHz
  - each node has 16 GB of memory
  - ran CCSD on 256 of these nodes, for a total of 4096 cores

Performance of Full CCSD

on 4096 cores, ¼ peak on top



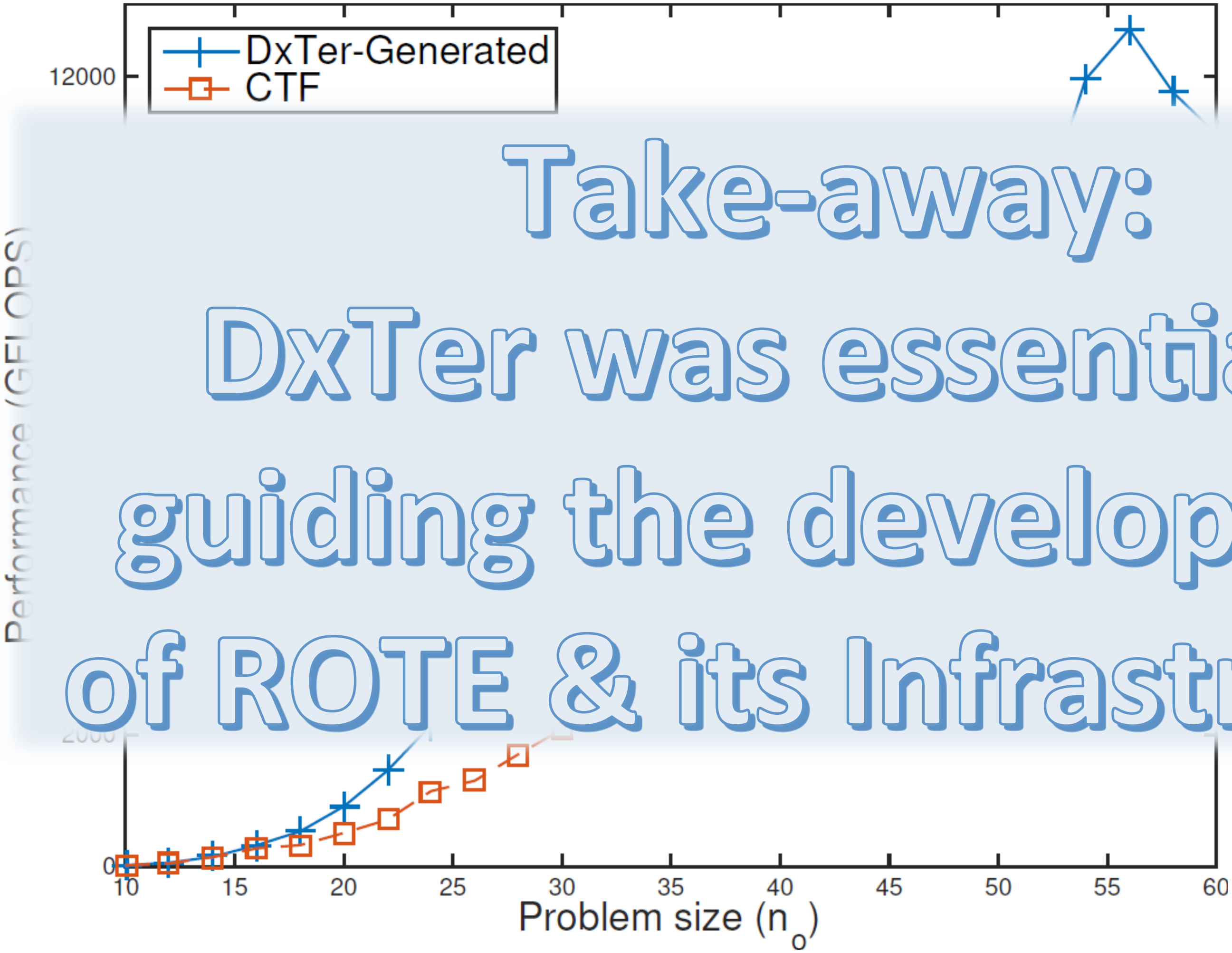
approx 1.3x larger problems

approx 1.4x faster

This is good!

Performance of Full CCSD

on 4096 cores, ¼ peak on top



Take-away:
DxTer was essential in guiding the development of ROTE & its Infrastructure

is good!

There is much, much more to say…
but enough for today

DON,
I DON'T WANT TO WORRY ABOUT MATH ☺

30 Years Ago

- This leap forward could not have been done or would not believable
  - simply didn't have the "observational" data to propel us forward
  - each "experiment" to derive programs from graph identities took ~4 years
  - had to look at several domains to see the commonalities → more years
  - took years to bring the pieces together → doesn't happen over night

- And this is Hard: If there is anything I've learned from programming and my career:

We are geniuses at making the simplest things complicated; Finding the simplicity and elegance behind what we do is hard

And If You Do It Right…

- You will know you are successful when people ask…

> **So what was the problem?**
> **What is so hard about this?**

- Here, in summary, are my take-away ideas…

Finding Domain-Specific Identities

- Is a fundamental activity in Science, like Physics

$$L = L_0 \sqrt{1 - v^2/c^2}$$

$$\begin{aligned} \nabla \cdot \mathbf{D} &= \rho \\ \nabla \cdot \mathbf{B} &= 0 \\ \nabla \times \mathbf{E} &= -\frac{\partial \mathbf{B}}{\partial t} \\ \nabla \times \mathbf{H} &= \mathbf{J} + \frac{\partial \mathbf{D}}{\partial t} \end{aligned}$$

$$G_{\mu\nu} = 8\pi T_{\mu\nu}$$

$$E = mc^2$$

$$F = ma = d\boldsymbol{p}/dt$$

$$\sigma_x \sigma_y \geq \hbar/2$$

$$F = G m_1 m_2 / d^2$$

$$i\hbar \Psi(\boldsymbol{r},t) = -\hbar^2/2m \, \nabla^2 \Psi(\boldsymbol{r},t) + V(\boldsymbol{r},t)\Psi(\boldsymbol{r},t)$$

Finding Domain-Specific Identities

- Is a fundamental activity in Science, like Physics

Physics is discovering the structural identities of the Universe

$\sigma_x \sigma_y \geq \hbar/2$

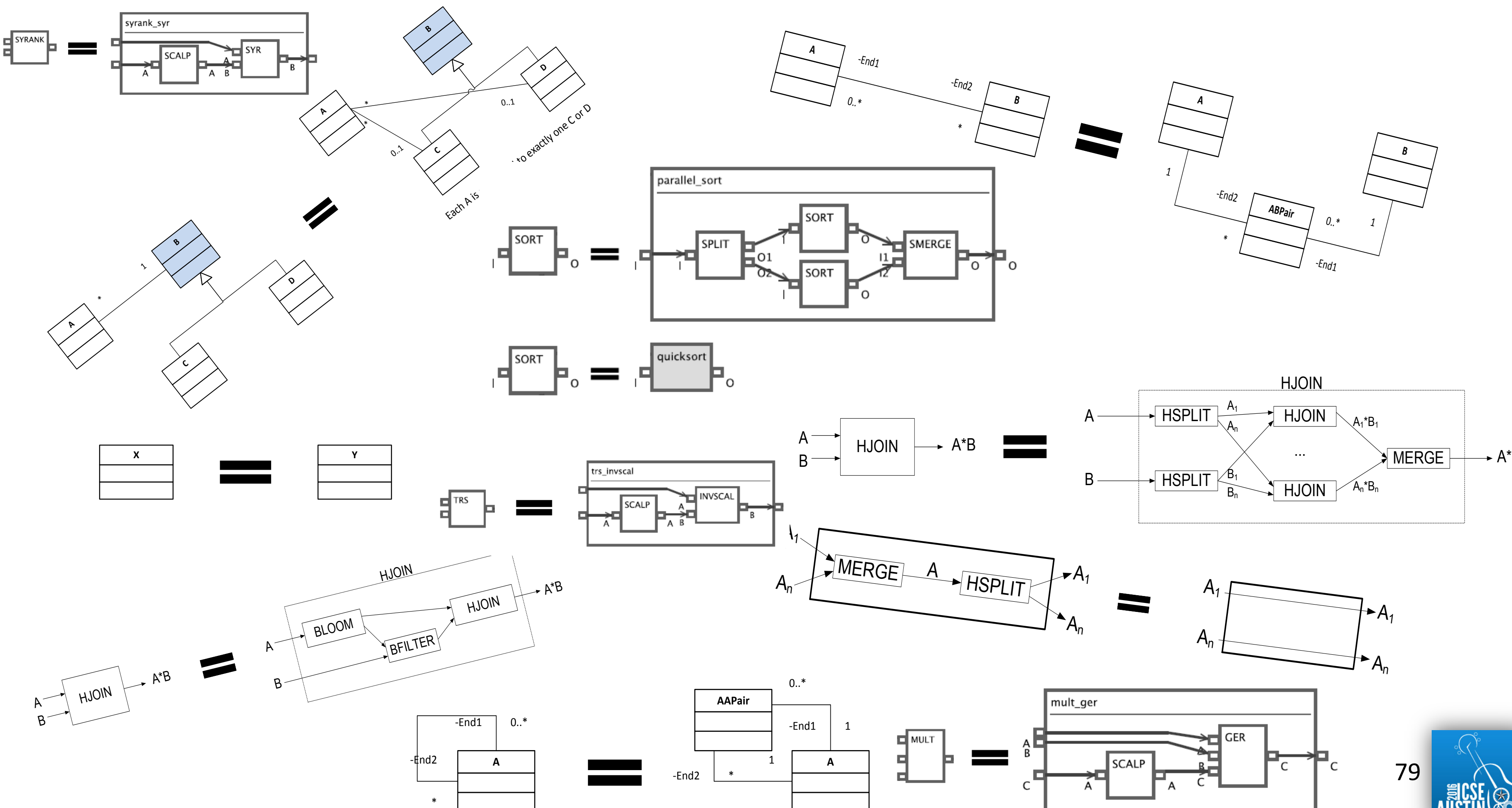
$F = ma = d\mathbf{p}/dt$

$F = G m_1 m_2 / d^2$

$i\hbar \Psi(\mathbf{r},t) = -\hbar^2/2m \; \nabla^2 \Psi(\mathbf{r},t) + V(\mathbf{r},t)\Psi(\mathbf{r},t)$

Finding Domain-Specific Identities

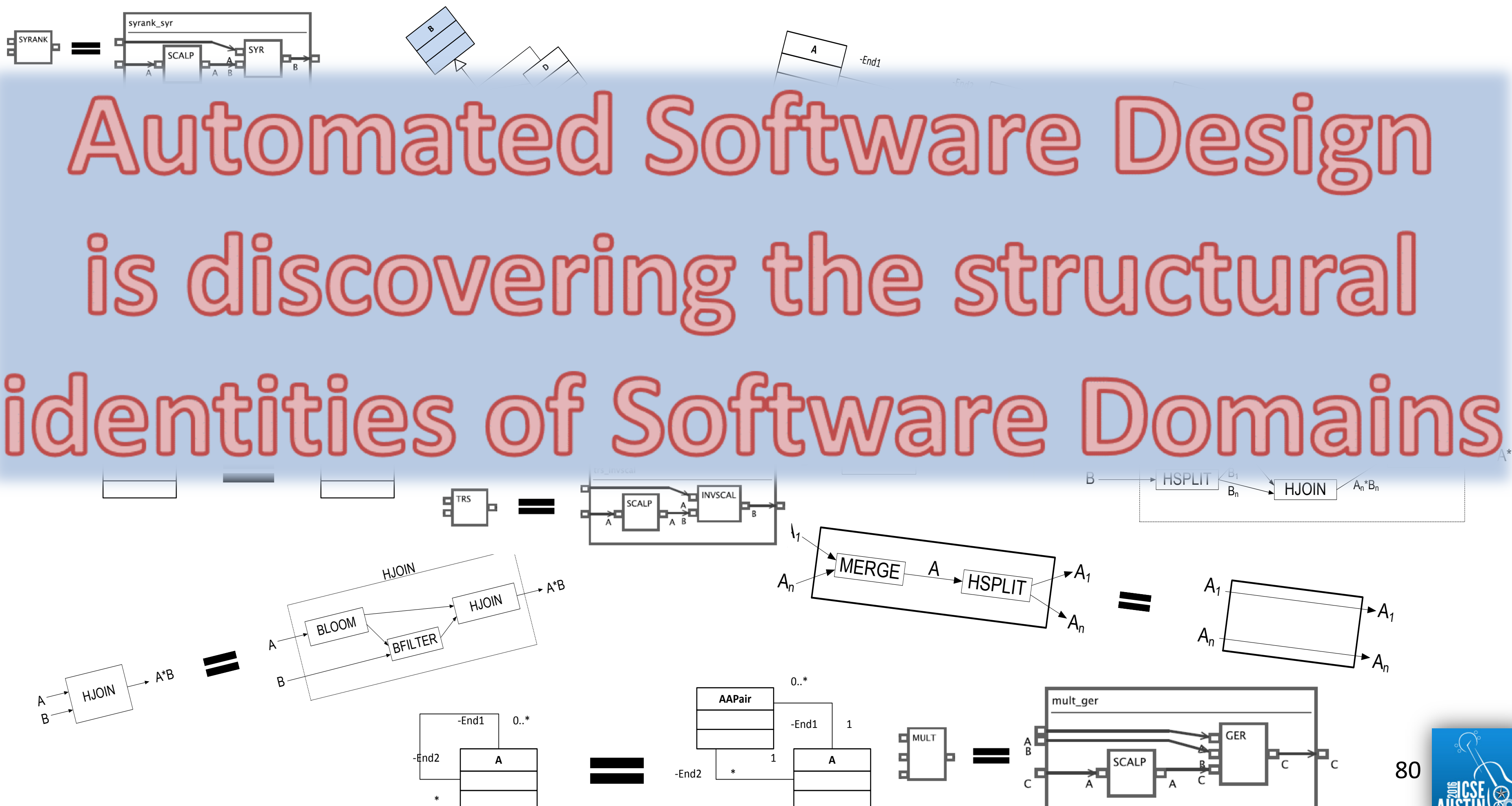
- Is also a fundamental activity in Automated Software Design



2016 ICSE AUSTIN

Finding Domain-Specific Identities

- Is also a fundamental activity in Automated Software Design



Automated Software Design is discovering the structural identities of Software Domains

Teaching Math in Software Design is Important

- Foundation of Science
- Shows algebraic foundations of advances in last 25 years in Software Design:

Model Driven Engineering
Refactorings
Design Patterns
Parallel Architectures
Product Lines

Remember History of Science

- Greatest technical advances in last century were via science
- It will be no difference for software design
- It is now time to prepare our students for the future, not to continue the past

Even if you are a dog... ☺

Thank You!