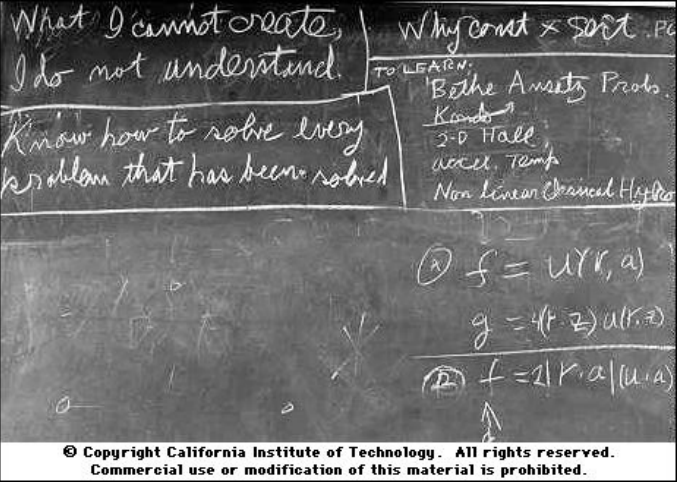
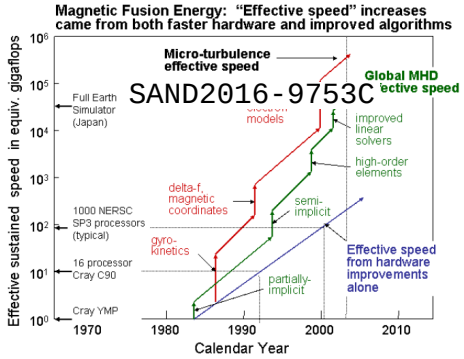
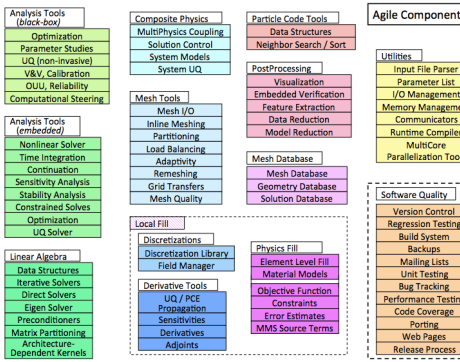


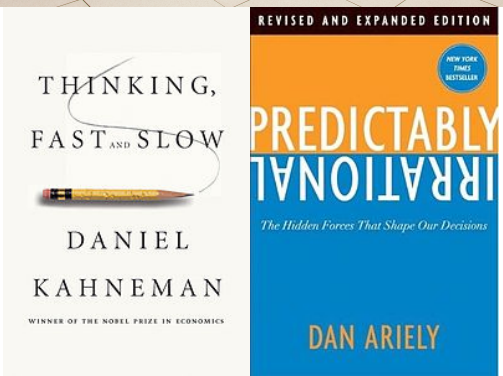


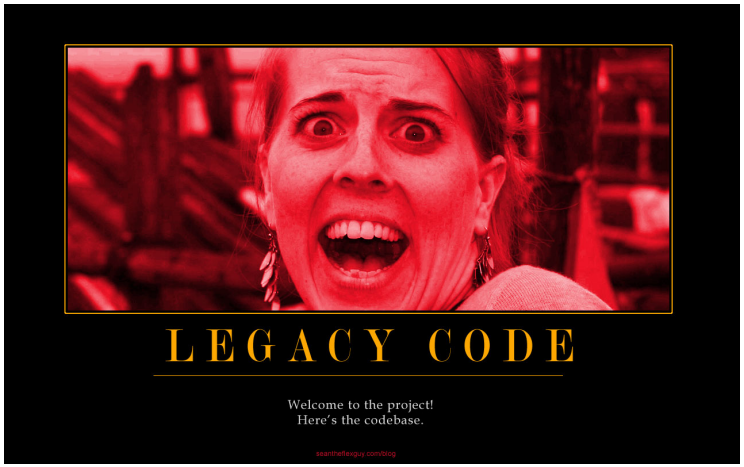
Are we actually modernizing our codes?

Bill Rider

CSSS January 12, 2016







Yes,

with a lot of caveats

“Dare to think for yourself.”

“Doubt is an uncomfortable condition, but certainty is a ridiculous one.”

“**Judge a man by his questions rather than by his answers**.”
— Voltaire

Maybe the first thing to do is define “modern”

Does modern mean it runs (well) on current processors?
Does modern mean it runs on a modern HPC platform?
Does modern mean its not written in Fortran?
…

Maybe I should ask a better question:
Are we creating modern codes?
Are we solving modern problems?
Are we using modern methods?
Are we using modern algorithms?
Are we computing modernly?
Or
Have we created a new generation of legacy codes?
Computing in legacy ways?

Is “modern” even good enough? Shouldn’t we be creating the future?

IMHO the answers to these questions, if tackled honestly, should trouble all of us.

We are not even creating modern codes much less the codes of the future.

The Three Goals for this Talk

- **Goal 1:** Take a critical look at the legacy of the ASC program so far, and what has been achieved.
- **Goal 2:** Discuss what constitutes a modern code all the way from problem being solved to hardware running the code.
- **Goal 3:** Discuss how we might take a more holistic strategy for developing a modern code base and the collateral intellectual expertise providing appropriate service to the stockpile.

"Code written in the 1970's is the bane of modern programming. It even has a special name: legacy code. Legacy code is feared, poorly understood, and worried over; most software professionals try to avoid making its maintenance part of their careers...”

- J. A. Whittaker and J. M. Voas (2002), "50 Years of Software: Key Principles for Quality," IT Pro, Nov/Dec issue.

ASC Road Map, SAND 2006-7535P

- Legacy code: Application codes that existed prior to the start of the ASC Program, before 1995. In many cases, Legacy codes are no longer being actively developed.
- Modern code: Application codes first developed under the ASC Program, starting after 1995. Some codes that would have been classified as Legacy codes have been significantly redesigned under the ASC Program and are therefore classified as modern codes.

Are these defintions fixed and immutable?

Are they correct? Are they useful?

LEGACY CODE

Welcome to the project!
Here's the codebase.

"Institutions will try to preserve the problem to which they are the solution."

– Clay Shirky

We are 20 years into ASC(I), perhaps it is time to examine its legacy

- ASC(I) was and is predicated on the prospect that hardware **speed** is the **key** to predictive simulation and modeling
  - **Is this a valid or good assumption?**
  - **If it were true wouldn't we have succeeded already?**
  - **Is anyone questioning this hypothesis?**
- Our code's models and methods are largely the same as when ASCI began–little or no progress.
- The technical composition of ATDM, being much like the original ASCI program – little has been learned
- ASCI needed to add PEM + V&V and these efforts are underfunded.
  - "Real" method & algorithm work are missing

Advanced Simulation and Computing

ASC started as ASCI in 1995 to develop simulation capabilities to analyze and predict the performance, safety and reliability of nuclear weapons

Program Elements

- Integrated Codes (IC)
- Physics and Engineering Models (P&EM)
- Verification and Validation (V&V)
- Computational Systems and Software Environment (CSSE)
- Facility Operations and User Support (FOUS)
- *Advanced Technology Development & Mitigation (ATDM)*
  - New program element started in FY14
  - Combination of IC and CSSE…

“V&V takes the fun out of computational simulation”
– Tim Trucano

What will be ASC(I)'s legacy?

Are we making a new generation of "legacy codes?"

- ASCI started in 1996.
- We are talking about where we will be in 2020
- 1972 is the same time difference .
- Where were we in 1972?
- Would we be happy using the models, methods and algorithms from 1972 in our 1996 codes?
- Will the progress from 1996 to 2020 look like that from 1972 to 1996?

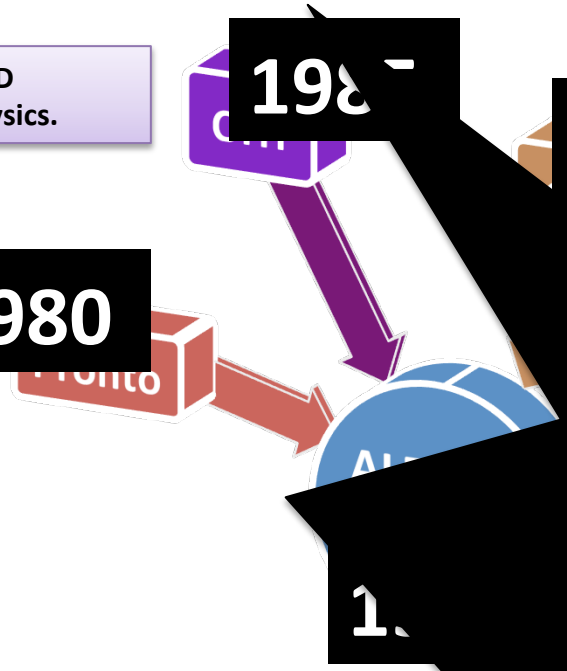
No it won't; not even a close competition.


I’ve seen a possible future
for our codes…

ALEGRA is a multi-physics hydrocode developed over the last 20-25 years

CTH is a fixture in the DoD community for shock physics.

SNL code capabilities: Pronto was a mid-1980 Lagrangian solid dynamics code and is a the basis of SIERRA-MECHANICS today (Presto).



There are new capabilities and algorithms in the code, but the core methods that define them are old, too old!

CTH is a 30 year old code representing a modern method at that time and SNL's corporate expertise in shock hydro. They did add AMR and lots of models

Defining

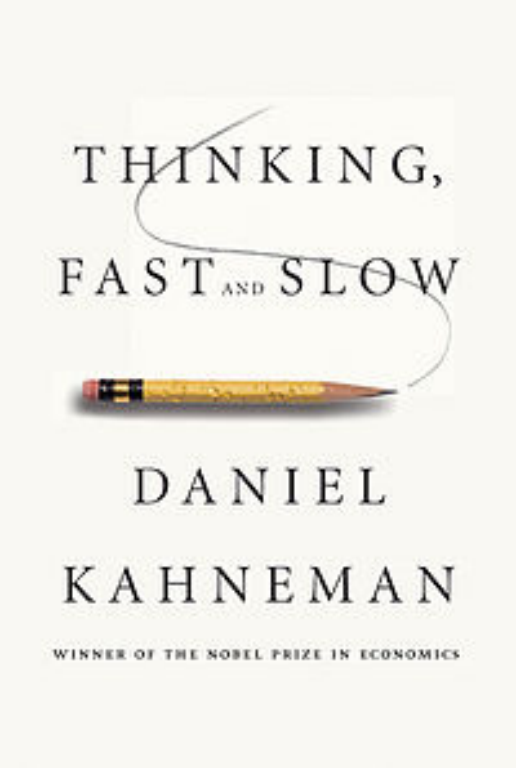
- Mi
- Robust mo
- XFEM

Let's talk about Methods & Algorithms in the ASC codes

- **SNL**
  - ✓Sierra – mixed bag, SM is legacy, FT is less so, …
  - ✓Shock/Aero – Legacy except for AMR/MHD stuff
  - ✓Emphasis – Legacy ATDM might change is a little
  - ✓Xyce/Charon/Rad – newer code, but legacy awaits
- **LLNL**
  - ✓Ares/ALE3D/Kull/Lasnex – All basically CALE++
  - ✓Blast/Miranda – new high order codes (the bright spot)
- **LANL**
  - ✓Rage – weird "Godunov" method with AMR
  - ✓Flag – free Lagrange method plus newer Lagrangian
  - ✓Mesa – old British code ported many times

Vast amounts of research over the last 25 years is waiting to be called upon to improve any and all of these codes. Making any of it stable, robust and reliable for production is very risky.

Preserve the code base



How much of our strategy is based on sunk cost?

What would a clean sheet strategy be?

Rather than consider the odds that an incremental investment would produce a positive return, people tend to "throw good money after bad" and continue investing in projects with poor prospects that have already consumed significant resources. In part this is to avoid feelings of regret.– Daniel Kahneman

"The validated and verified ASCI codes will eventually become the *production codes of the future* as such they will eventually transition to the custody of Stockpile Computing."
–1998 **Strategic Computing & Simulation Validation & Verification Program**

Now "Production Codes == Legacy Codes" ???

In my humble opinion “preserve the code base” is a terrible reason to do something. Why?

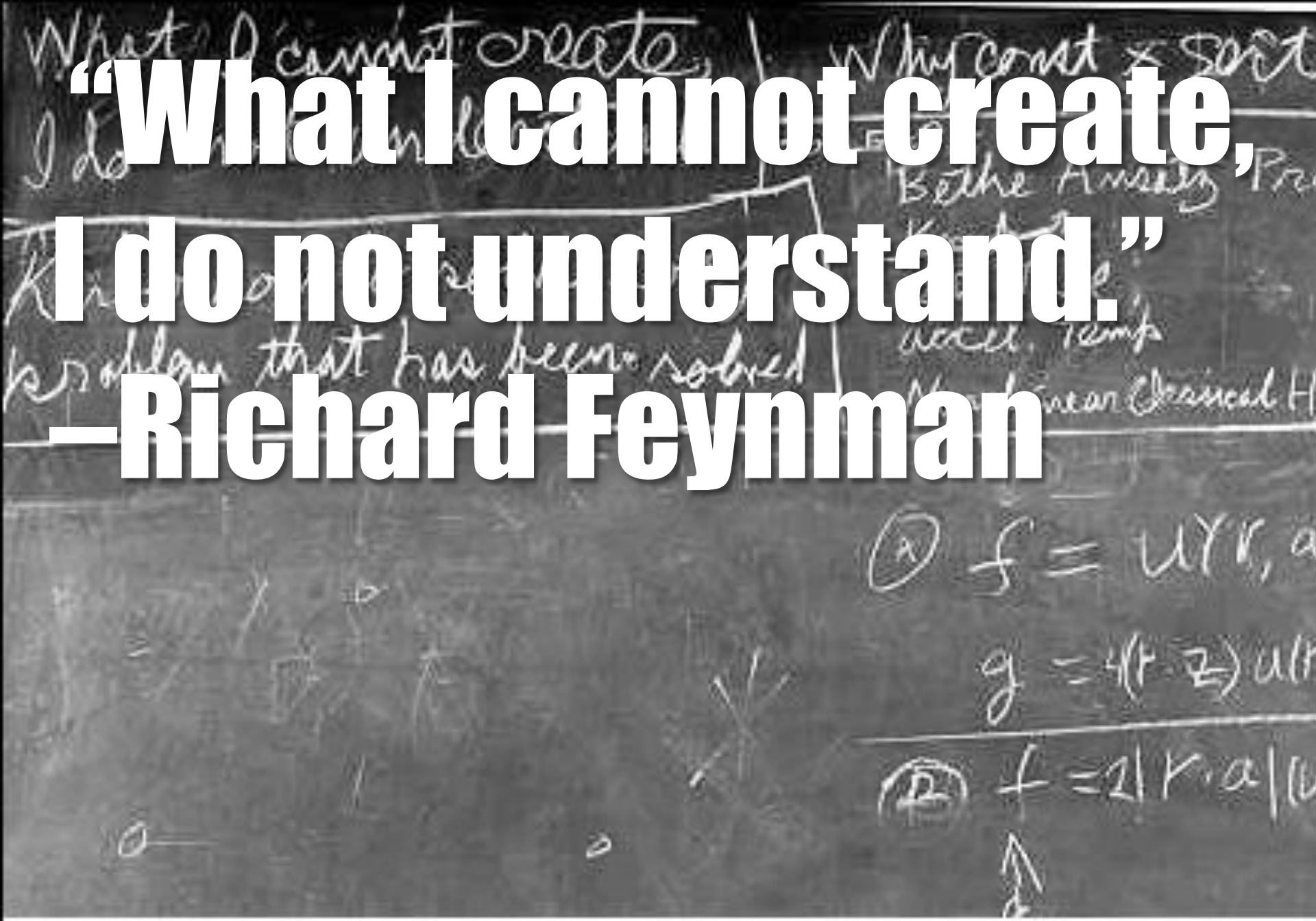
First, legacy codes come from preserving a code base successfully.

In other words preserving the code base creates legacy code.

What is code?

So, what is code?

- Code is a concrete expression of intellectual thought, math & ideas, i.e., models, procedures, methods and algorithms that can execute on a computer.
- The intellectual thoughts, math & ideas are more important than the code. Code effectively becomes legacy when the thoughts, math & ideas expressed are no longer clearly understood.
  - We run the risk of producing a vast expanse of code that we don't understand, yet depend upon.
  - We are already there in many cases.
  - This is a disaster.
- If we don't understand the code, should we be using it?
- Should we be using it for the purpose of National Security?



“In the twilight of Moore’s Law, the transitions to multicore processors, GPU computing, and HaaS cloud computing are not separate trends, but aspects of a single trend – mainstream computers from desktops to ‘smartphones’ are being permanently transformed into heterogeneous supercomputer clusters. Henceforth, a single compute-intensive application will need to harness different kinds of cores, in immense numbers, to get its job done.

The free lunch is over. Now welcome to the hardware jungle.” — Herb Sutter 2011

Exascale Applications Respond to DOE/NNSA Missions in Discovery, Design, and National Security

Scientific Discovery

- **Mesoscale materials and chemical sciences**
- **Improved climate models with reduced uncertainty**

Engineering Design

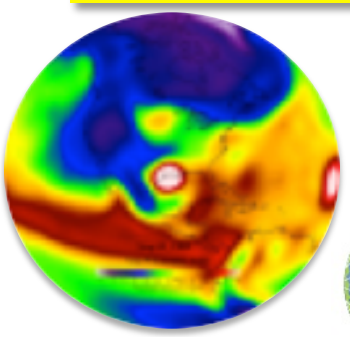
- **Nuclear power reactors**
- Advanced energy technologies
- Resilient power grid

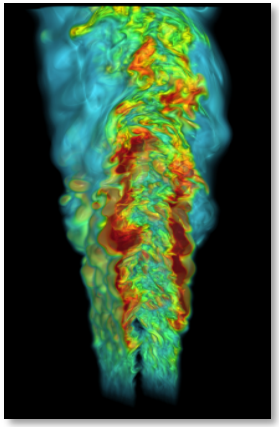
National Security

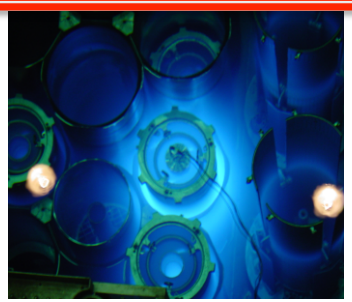
- **Stockpile stewardship**
- Real-time cybersecurity and incident response
- Advanced manufacturing

The Exascale drivers are terribly weak

Blue Bold Text indicates planned or existing exascale application projects







Making the drivers for Exascale better is a function of models, methods and algorithms plus genuine innovation in how modeling & simulation is utilized in Science & Engineering

The difficulty lies not so much in developing new ideas as in escaping from old ones.

— John Maynard Keynes

Was massively parallel computing a "disruptive innovation" ?

“People don’t want to buy a quarter-inch drill. They want a quarter-inch hole.”

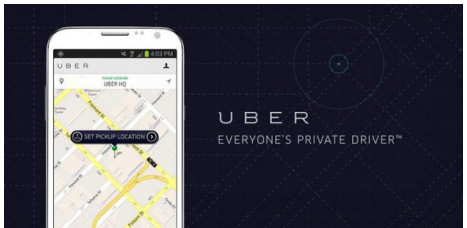
“the best way to get a good idea is to get a lot of ideas.”

“This is one of the innovator’s dilemmas: Blindly following the maxim that good managers should keep close to their customers can sometimes be a fatal mistake.”
— Clayton M. Christensen

Disruptive Innovation is a “hot” concept

- Introduced by Clayton Christenson in 1993
- Basically it is an innovation that changes the competitive landscape. Think of examples:
  - iPhone
  - DOS/Windows
  - Google
  - Facebook, Airbnb, Lyft, Uber, Tinder, …

- The old way of doing things can no longer compete and often goes out of business
- You want to be on the right side of the disruption, you’re rich on one side, out of the game on the other.
- Can we apply it to HPC?



So, what if the disruption isn't good, but instead it is bad?

Could massively parallel computing have been a negative disruption?

Exascale, what could go wrong?

End of life for Moore's law will be expensive and messy.

Perhaps its like the end of a human life..

lots of $$$ for little benefit!

Very expensive and low quality

We have massive technical debts. Our code bases are too old, we haven't invested enough in algorithms, methods and models, our talent base is weak.

The soul of the code (or skeleton) are mesh & solver/algorithms. **Where we aren't investing**

Our approach to computing is not resonant with the trends in value of the broader computing industry, but sub-serviant to them. – beyond big data, look at iPhones!

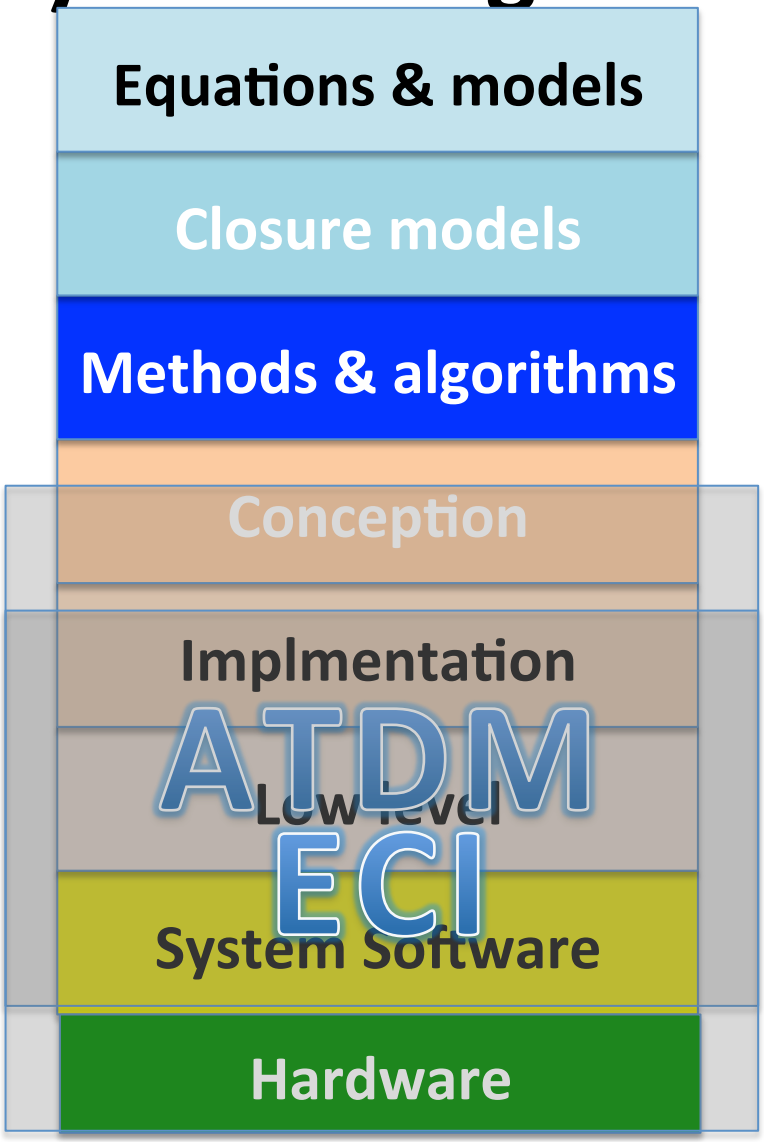
Risk and fear with administrative control are killing the future. Technical issues are not the problem!

If failure is not an option, then neither is success.
— Seth Godin

“Change almost never fails because it's too early. It almost always fails because it's too late.” — Seth Godin

The discussion of the "software stack" doesn't go nearly far enough

- We get lots of discussion of the various software stacks
  - System & programming
  - Solver stack
  - UQ stack
  - Discretization
- Does this go far enough?
- **What about a predictive simulation stack?**



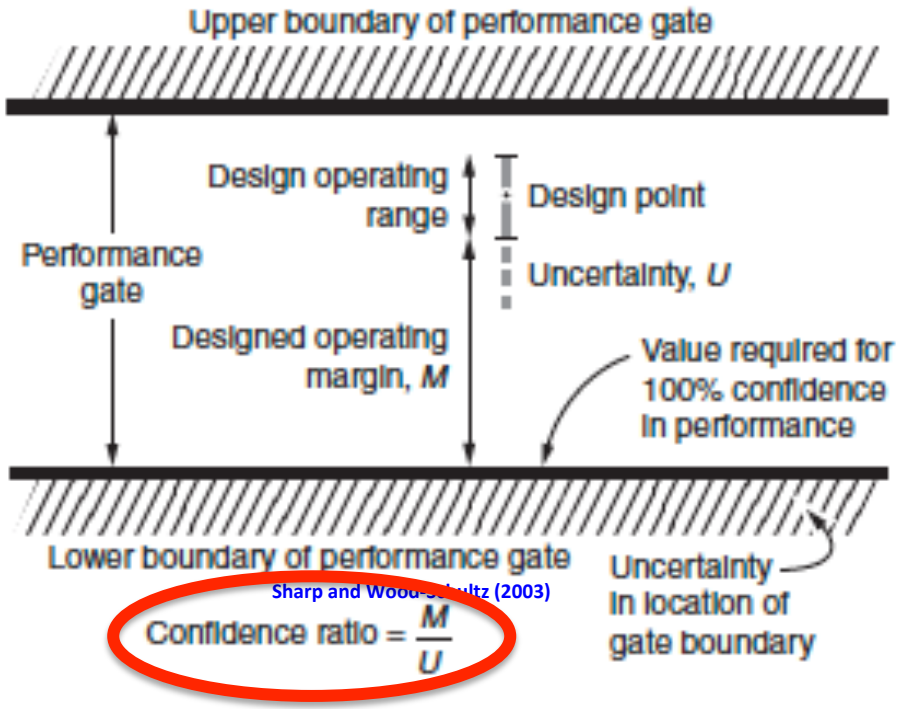
No amount of mesh refinement, numerical accuracy, or computer speed can rescue a model that is incorrect.

QMU is a good way to look at modeling

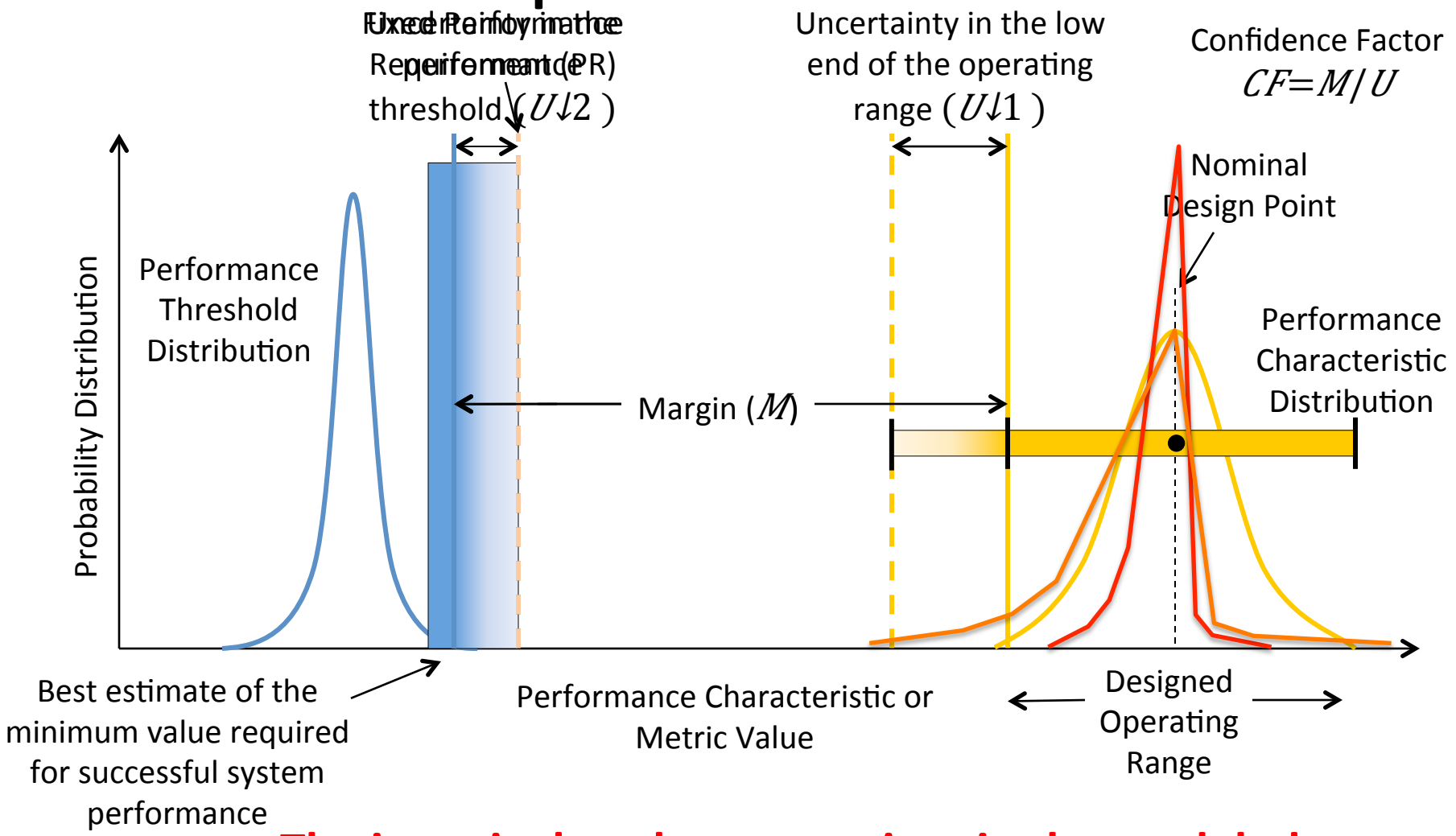
- This illustrates a performance threshold, uncertainty in the threshold, and resulting uncertainty in the performance margin M.
- A single formalism defines confidence:

$$\mathbf{CR \equiv M/U}$$

- This is unlikely to be the whole story.
  - Ex: Epistemic uncertainty characterization creates a family o these ratios (Pilch et al., 2011)
  - Therefore at least information has been collapsed to create a single ratio

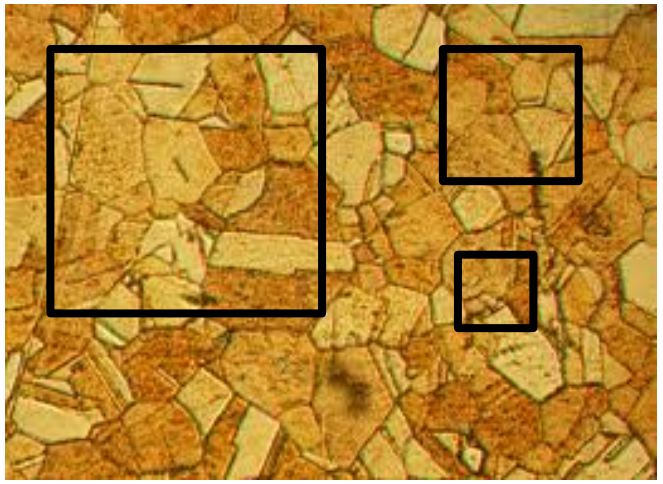


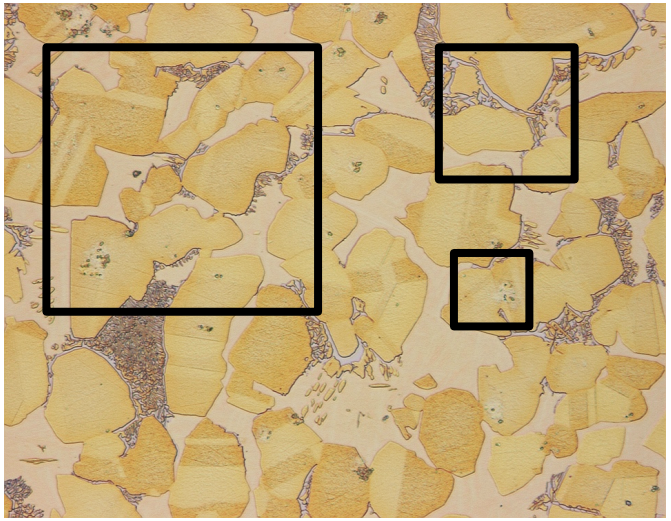
Conceptual Framework



The issue is that the uncertainty in the modeled average is not the same as the true distribution of the design!

We paint materials into regions most (i.e., all) of the time–used since the 1960's!





The governing equations themselves are a problem in this regard – mean field approach

- Most equations simply evolve the mean of the field

$$\frac{\partial U}{\partial t}+\nabla\cdot F\left(U\right)=0 \quad \Rightarrow \quad \frac{\partial \bar{U}}{\partial t}+\nabla\cdot F\left(\bar{U}\right)=0$$

- One could start to take a philosophy more similar to LES in turbulence, and consider the nonlinear effect of fluctuations (the mean is the mean of a random field too)

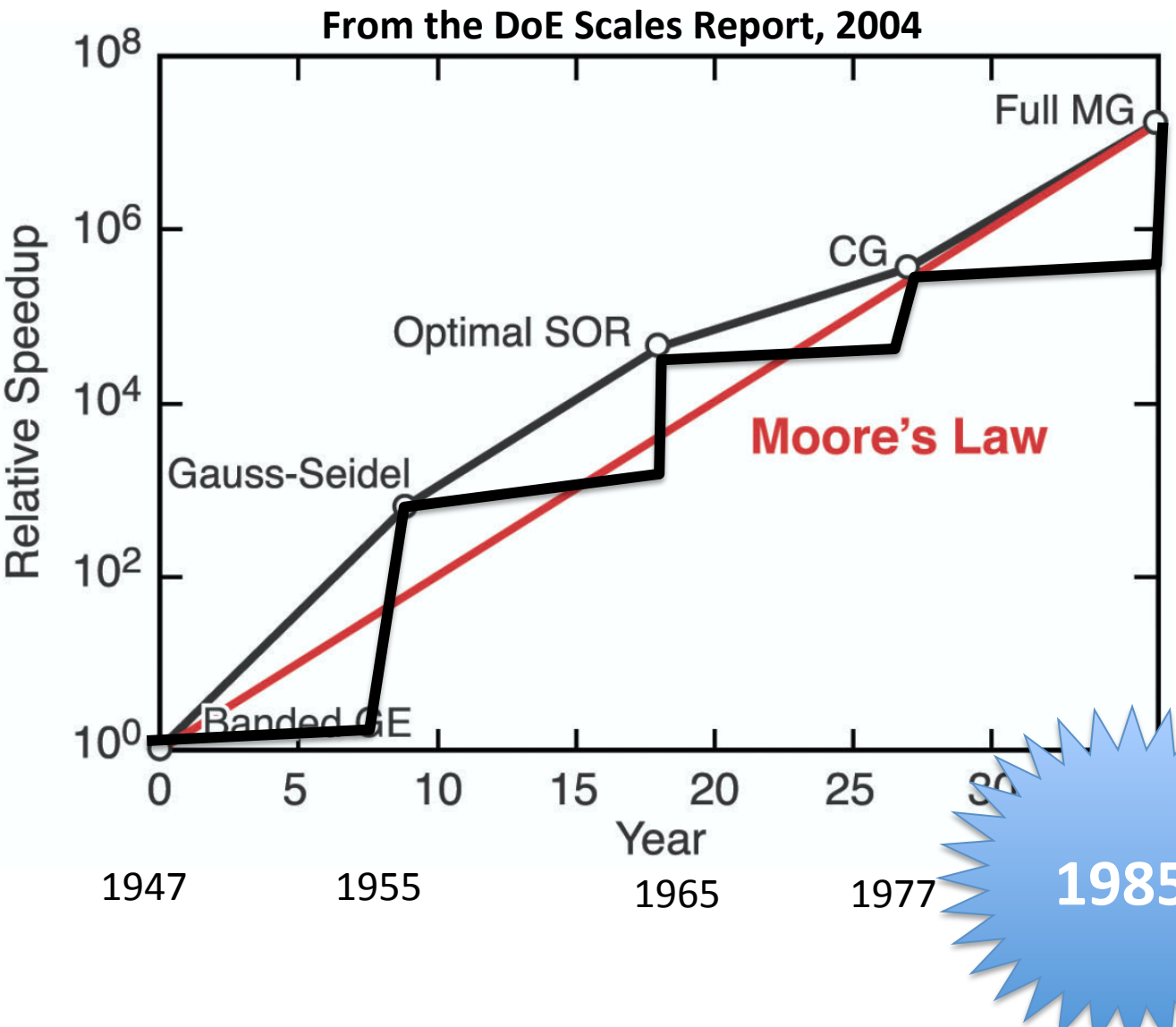
$$\frac{\partial \bar{U}}{\partial t}+\overline{\nabla\bullet F\left(U\right)}=0 \quad \Rightarrow \quad \frac{\partial \bar{U}}{\partial t}+\nabla\bullet F\left(\bar{U}+\zeta\right)+\nabla\bullet\tau\left(\bar{U}+\zeta\right)=0$$

- The question is then the structure of the subgrid stress both the physics, but also the randomness of the field (the unresolved or unresolvable portion). **This change renders the solution non-deterministic**.

“The fundamental law of computer science: As machines become more powerful, the efficiency of algorithms grows more important, not less.“

– Nick Trefethen

Comparing performance improvements between hardware and algorithms.



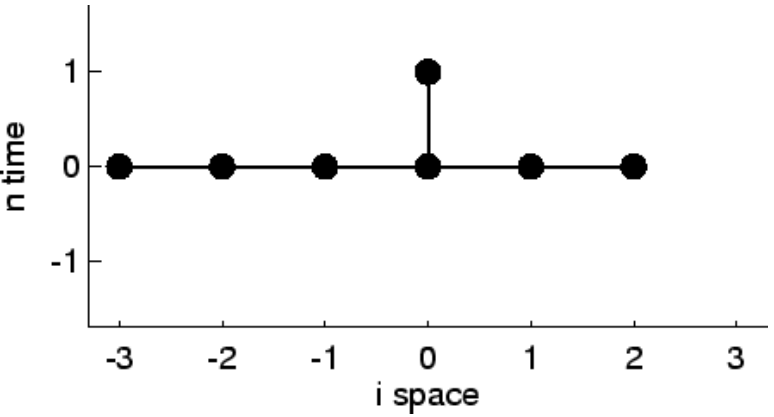
The jumps in performance are actually more discrete… "quantum"

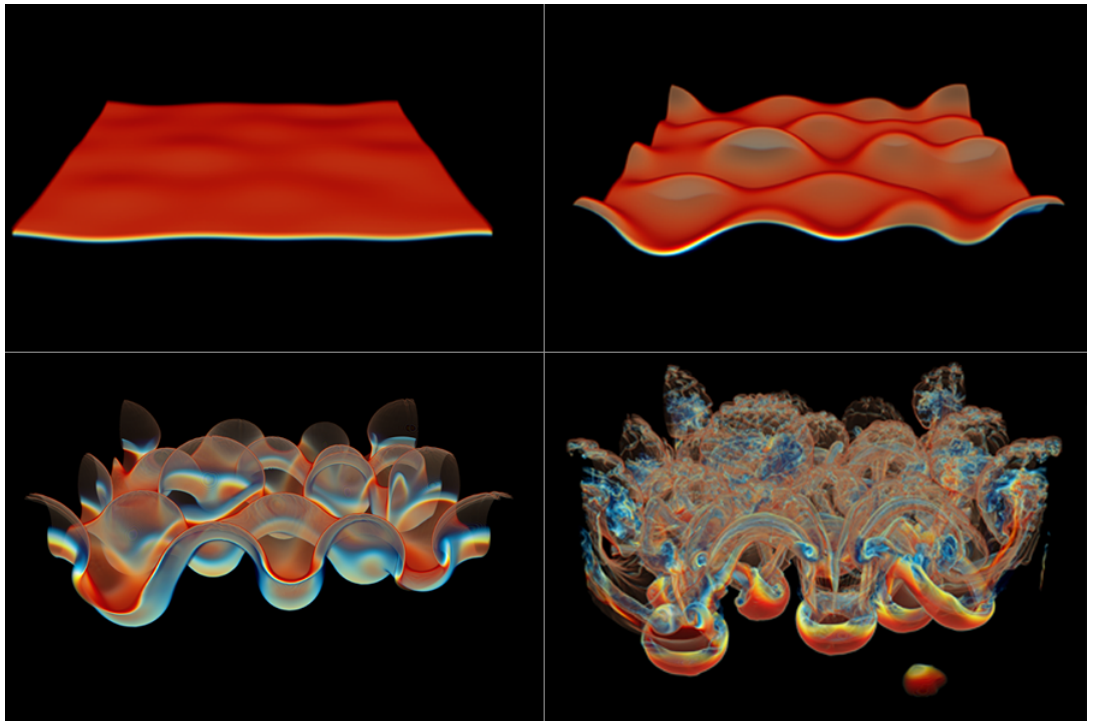
We are *overdue* for a breakthough, but what will it be? sublinear? A nonlinear method for a linear problem, or maybe multigrid is it?

… model solved using linear programming would have taken 82 years to solve in 1988... Fifteen years later… this same model could be solved in roughly 1 minute, an improvement by a factor of roughly 43 million… a factor of roughly **1,000 was due to increased processor speed**, … a factor of roughly 43,000 was due to improvements in algorithms! – DESIGNING A DIGITAL FUTURE: FEDERALLY FUNDED R&D IN NETWORKING AND INFORMATION TECHNOLOGY

“FLOPS are free” – Community Wisdom

Paul Woodward's PPM method performs amazingly well on a variety of modern computing.

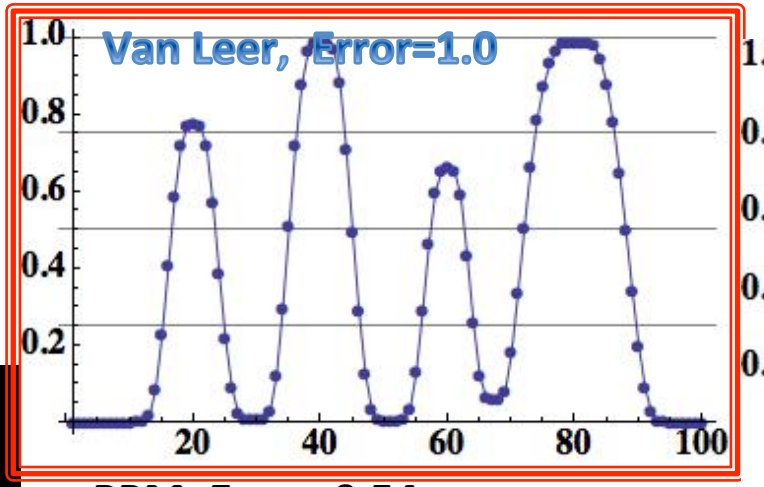


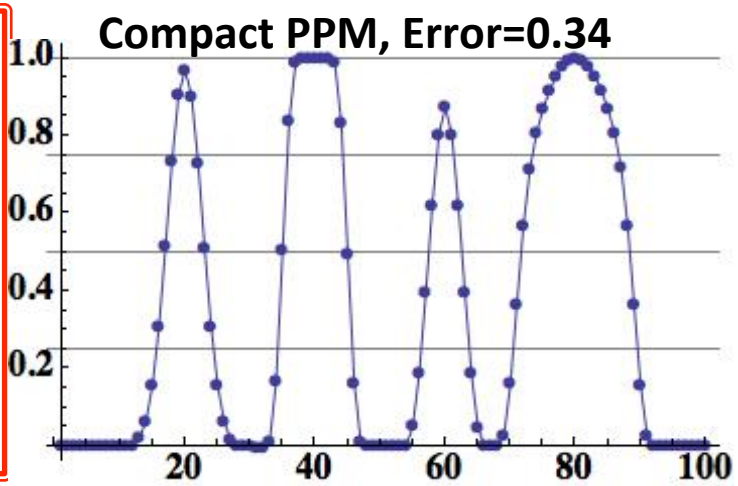


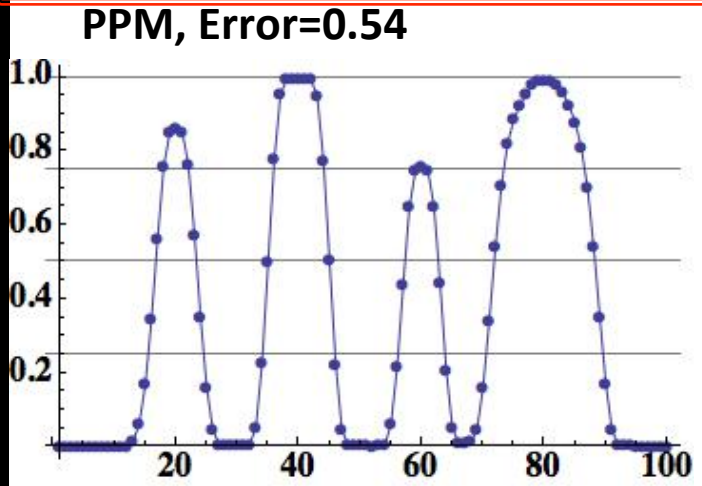
Why?

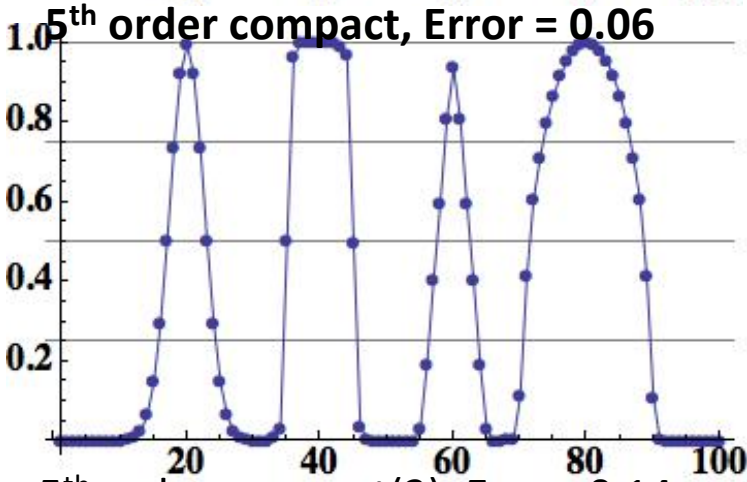
It is very FLOP intensive per memory reference.
How? It uses "high-order" methods and does an intense amount of analysis to adapt the discretization used to the local structure of the solution.

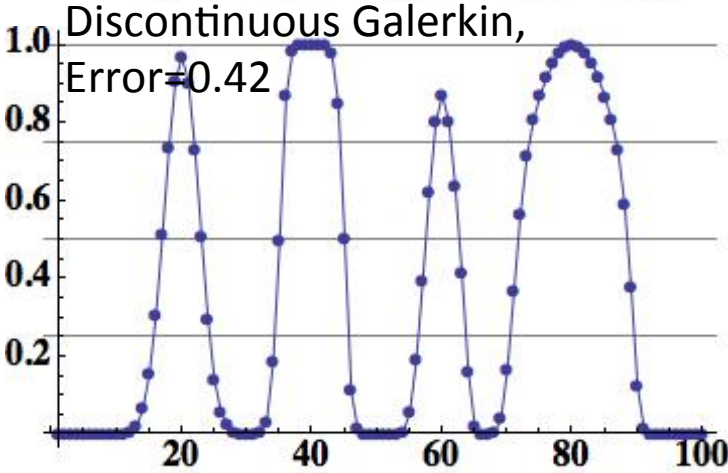
Methods and algorithms will improve the performance of a code on every single platform: laptop, to desktop, to cluster to capacity machine to capability machine

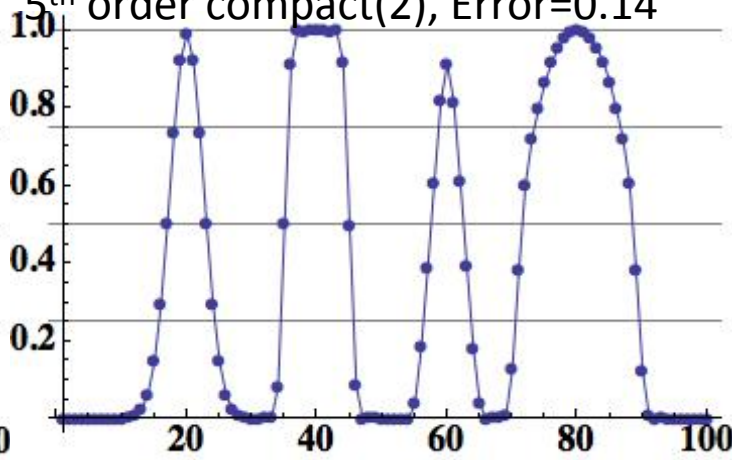










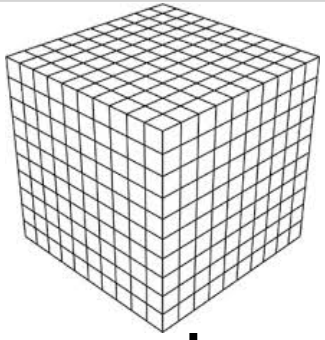


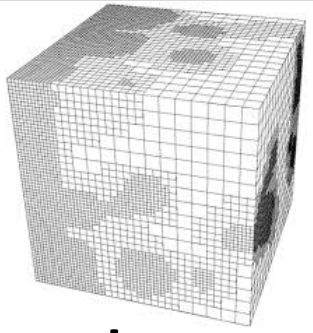
Accuracy can produce the same quality answer with much less effort

For shock-hydro problems we can estimate the impact

First-order accuracy (convergence), for problems containing shocks, and if cost is *effectively* equal.

Error Ratio	3-D N^4	AMR N^3
1	1	1
1/2 (PPM)	16x	8x
1/3 (CPPM)	81x	27x
1/6 (5^{th} O)	1296x	216x
1/16	65,536x	4096x





The advantage is larger for other problems, where high-order methods have a slightly higher rate of convergence

“An expert is someone who knows some of the worst mistakes that can be made in his subject, and how to avoid them.”

– Werner Heisenberg

If we don’t allow mistakes will we have any experts?

To sum up my thoughts

- ASC Has focused on hardware for 20 years, ECI simply follows that path.
  - ✓ Time to change? Is ECI really bold? Have we learned?
- For the most part we replaced on generation of legacy codes with another
  - ✓ What is the cost on our intellectual foundation?
- The gains from hardware have been modest
  - ✓ What is the opportunity cost?
- Progress is needed in more important areas
  - ✓ Models, Methods and Algorithms have been shorted

E-mail: wjrider@sandia.gov

Twitter: @TotalFutbalNow

Blog: http://wjrider.wordpress.com
The Regularized Singularity