

Industrial Strength Computational Science for DOE: A Verification and Validation Foundation

Tim Trucano

February 5, 1999

**Computational Physics Research and Development Department
Sandia National Laboratories
Albuquerque, NM 87185**

Phone: 844-8812, FAX: 844-0918

Email: tgtruca@sandia.gov

<http://sherpa.sandia.gov/9231home/compphys-frame.html>

Sandia is a multiprogram laboratory operated by Sandia Corporation, a Lockheed-Martin Company, for the United States Department of Energy under Contract DE-AC04-94AL85000.



A little understanding goes a long way...

“In terms of the significance which is disclosed in understanding the world, concerned Being-alongside the ready-to-hand gives itself to understand whatever involvement that which is encountered can have.”

Martin Heidegger, Being and Time

I want to discuss some technical issues for code verification and validation.

I will discuss:

- I. Software quality assurance (SQA)
- II. Software engineering (SE)
- III. Verification testing
- IV. Uncertainty analysis
- V. DOOMSDACE

There are viewpoints that underlie this talk:

- The goal is to provide information, not codes (don't confuse means with ends!)
- The focus of our code development is engineered software product rather than science

ASCI program managers are not the only ones who say controversial things about computational science.

“If the capabilities of computers continue to increase as they have in the past, the relative roles of experiment and computation in providing aerodynamic flow simulations will undergo profound changes.”

D. R. Chapman, H. Mark, and M. W. Pirtle (1975), “Computers vs Wind Tunnels for Aerodynamic Flow Simulations,” *Astronautics and Aeronautics*, April, 22-35.

They stressed that economics is a major component in pressure on wind tunnel facilities. [Does this sound familiar?]

The authors quantitatively estimated that ~ 400 GFlops would do the trick. Duh...

Practical definitions of Verification and Validation

Here are practical definitions for physics code development:

Verification: “Are we solving the equations correctly”

- This is a mathematics/computer science issue.

Validation: “Are we solving the correct equations”?

- This is a physics issue.

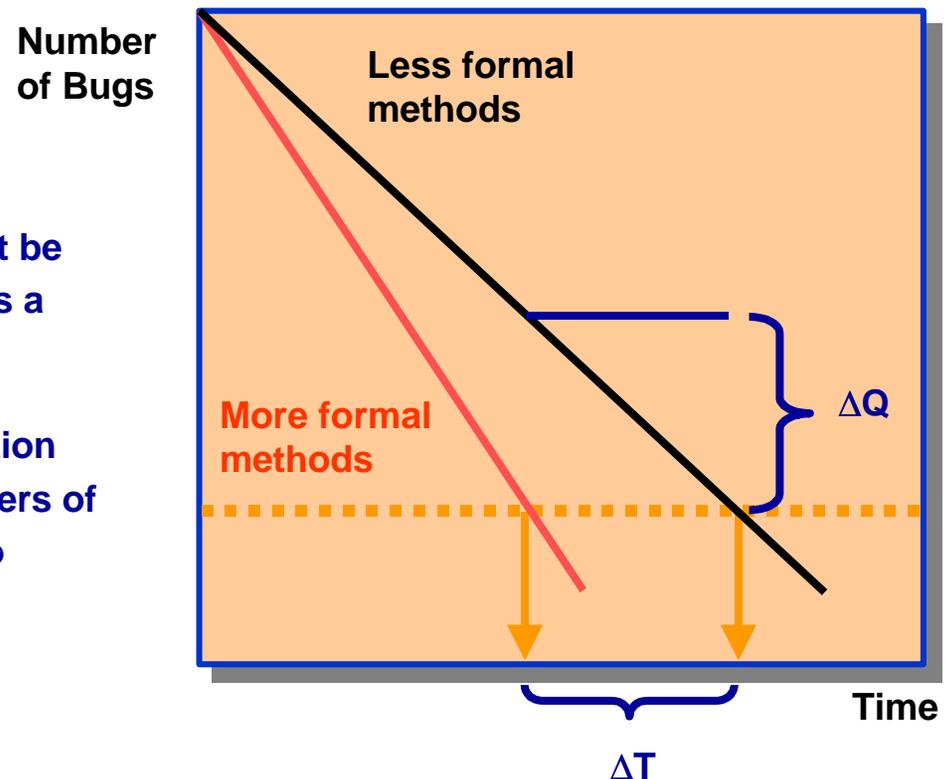
V&V must be coupled with code “accreditation” or “certification” for high consequence applications.

Verification should lead to the increase of code quality as a function of time, or cost, or effort.

Software quality is a multidimensional entity which evolves with multiple variables.

For example, software quality might be measured by the number of bugs as a function of the life cycle time.

The payoff for more formal verification processes would be reduced numbers of bugs at a given time, or less time to achieve a given bug status, or ...



We seek to verify multi-physics codes.

Multi-physics often implies general purpose

General purpose implies that users are a huge component in correct application of the code. Code verification does not address user error.

Multi-physics also often implies research models are present

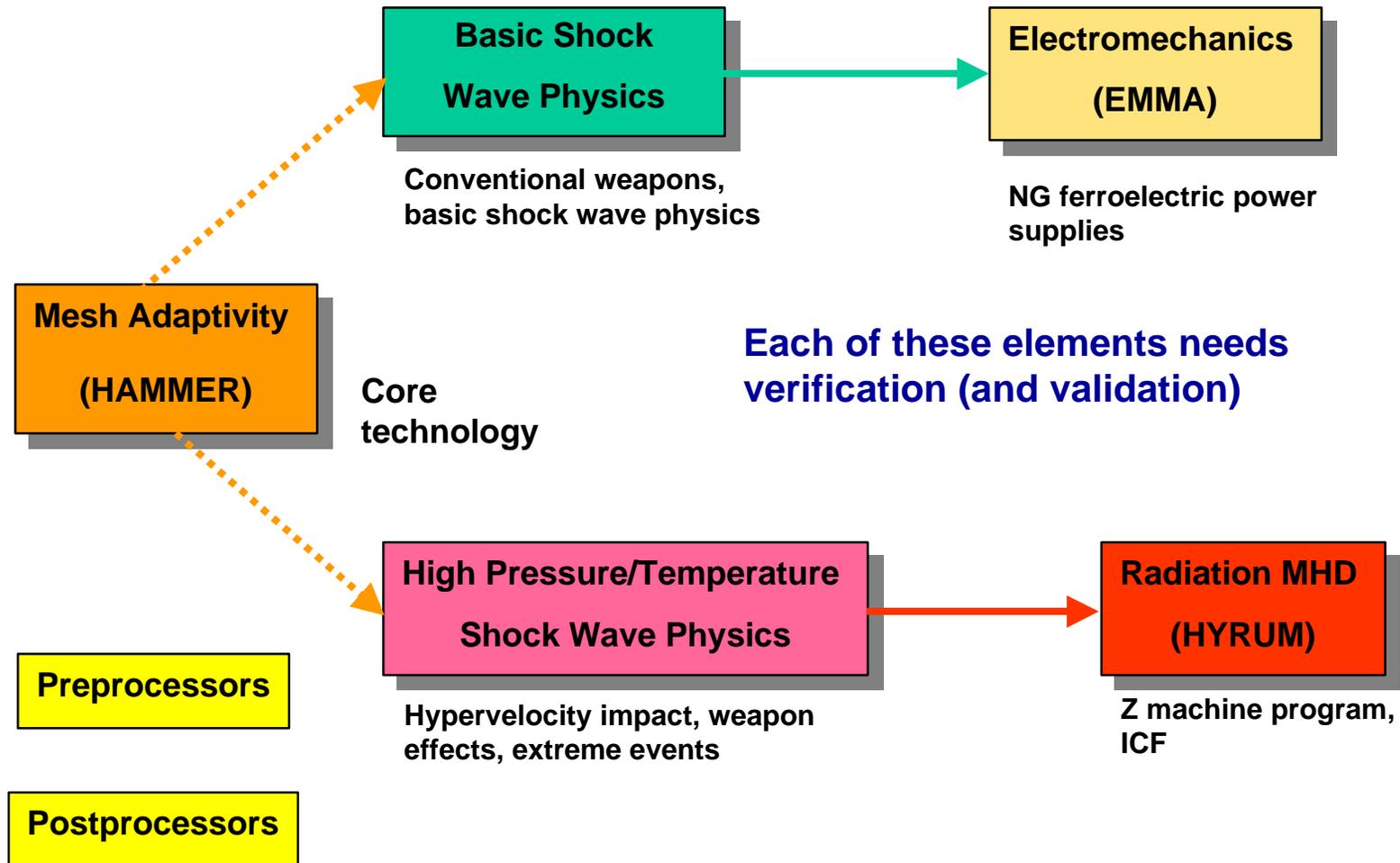
An example is the Sandia code ALEGRA:

3-D multi-material Arbitrary Lagrangian Eulerian Radiation-MHD shock wave physics code

C++ (150000 lines) + MPI + C + Fortran, etc. (1,000,000 total including special libraries)

Under development

Example: ALEGRA is a multi-physics code.



The context of the ASCI V&V program is broad.

Oberkampf has reviewed the not so recent work on formal V&V of scientific software:

- Society for Computer Modeling (1979)
- IEEE (1984)
- ASME (beginning 1986 and still evolving 1993)
- American Nuclear Society (1987)
- Military Operations Research Society (?)
- Defense Modeling and Simulation Organization (1994)
- AIAA (1994, 1998)
- • DOE DP [ASCI] (1998); other formal SQA activities prior to this date
- ISO standards (?)

Further information:

Pat Roache has written *the* book: P. J. Roache Verification and Validation in Computational Science and Engineering , Hermosa, 1998

Another interesting book is: P. L. Knepell and D. C. Arangno, Simulation Validation - A Confidence Assessment Methodology, IEEE Computer Society, 1993

Here is a smattering of bibliographies:

W. O. Oberkampf (1998), SAND98-2041, several hundred

O. Balci and R. G. Sargent (1984), Simuletter, Vol. 15, No. 3, several hundred

Date unknown, bibliography available at the DMSO Web Site

D. S. (Steve) Stevenson (1998), unpublished, ~ 150 references

T. G. Trucano (1998), unpublished, ~ 200 references

I. Software quality - is this an oxymoron?

How much do you believe my premise that we are delivering software product?

“...I’m here to make sure that they [V&V Program] don’t screw us up...”, ASCI code development program person...

SQA with current ASCI code development - one view:

Good application of the principles to begin with

Lax review approach during the software life cycle

No **metrics** of the software life cycle process

Example: ASCI SQA activities in FY99:

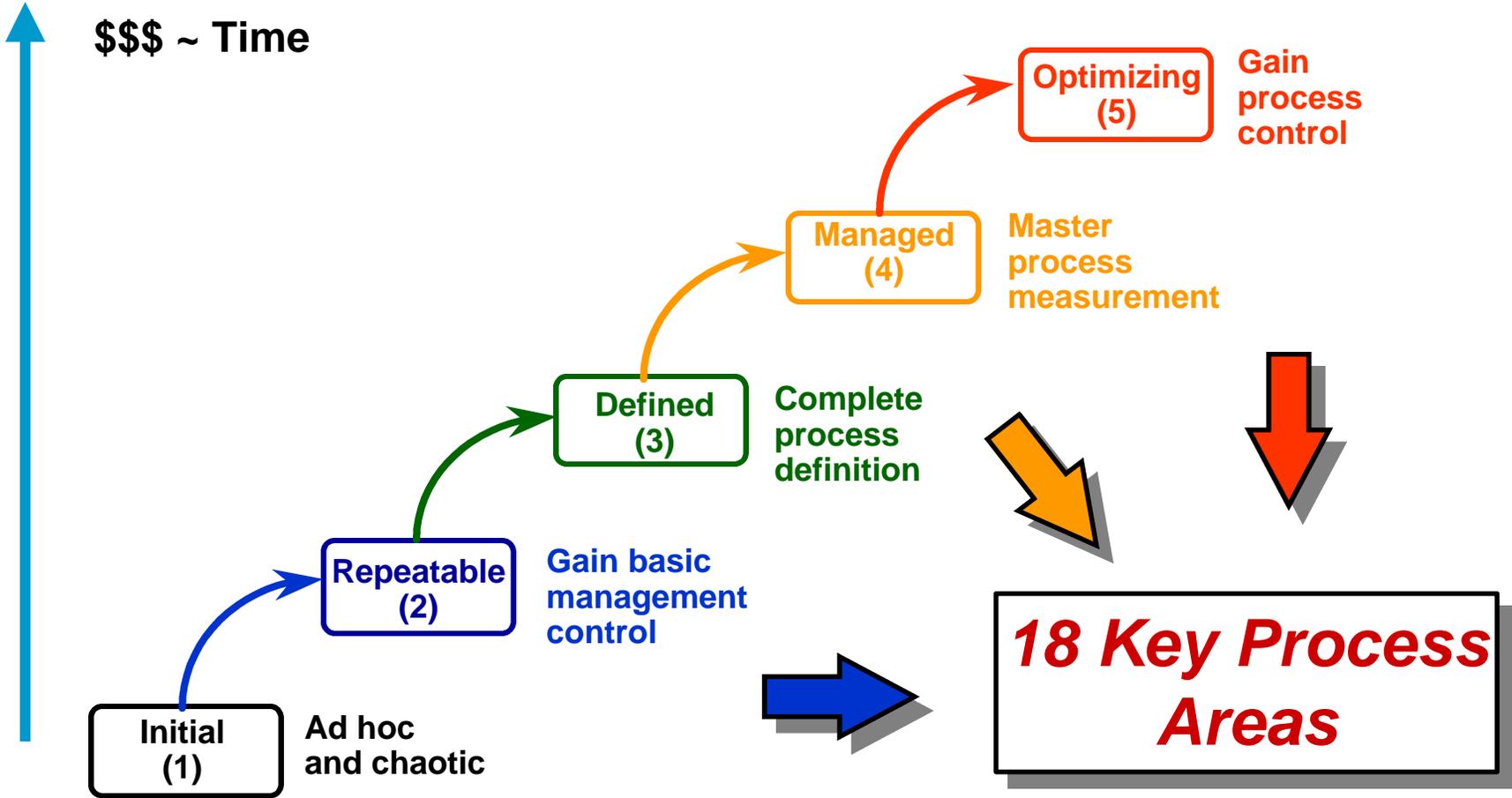
Quality surveys of code development activities are being performed in a Tri-Lab activity. This will be completed ~ June 1999.

Investigations into the utility of the SEI Capability Maturity Model (CMM) are being performed:

- Contract to SEI studying relevance of CMM (LLNL)
- Personnel training for internal CMM work (LANL)
- SQA study and guidance for use of CMM (Sandia)

Issue: cost/effort tradeoffs; metrics

The SEI Capability Maturity Model is a multilevel software quality tool.



We estimate ASCI code projects to be concentrated between Levels 2 and 3.

Transition to Level 2 requires:

Configuration management **s**

SQA **T**

Sub-contract management (suppliers) **T**

Project tracking and oversight **s**

Project planning **s**

Requirements management **T**

Transition to Level 3 requires:

Peer reviews **s**

Intergroup coordination **?**

Product engineering **T**

Integrated software management **T**

Training program **s**

Organization process definition **?**

Organization process focus **?**

s - doing it; **T** - worrying about it; **?** - don't know

II. Delivering computational science as product requires software engineering (SE).

What software engineering principles and techniques are applicable to large systems of floating point multi-physics code?

Many recommendations are called but less are chosen:

Configuration management systems and rules

Program development environments (is the ideal to have no human write any code, but only specs, equations, etc?)

Coding standards

Formal software inspections

Regression testing

Documentation - plans, requirements, theory, implementation, algorithms, process

**The
Crux!**

“...the author cannot find any long term empirical study on the efficacy of the various software engineering techniques...”, D. E. Stevenson (1991), “1001 Reasons for not Proving Programs Correct: A Survey,” Clemson preprint.

Still true. Recent comments from NIS study group are similar

III. ASCI verification testing issues: If only it were true that ...



Verification logically fits this model relatively well, except...

this still doesn't "prove" that the code implementation is "correct."

The historical approaches to testing are performed in ASCI verification programs.

Test problem classification:

“Consistency Checks”
(spherical blast waves)

Analytic test problems
(Whalen test suite)

Very precise numerical
solutions (Sedov blast
waves)

Code comparisons
(ALEGRA vs CTH, etc)

Mesh generation

Initial conditions

Boundary conditions

Physics algorithm design

Algorithm accuracy,
convergence, stability,
consistency

Algorithm coupling

Material models (EOS,
opacity, strength, fracture)

Post-processing

The “God” of testing speaks:

B. Beizer, Software Testing Techniques, International Thomson Computer Press, 1990

For example (**software**): [Beizer]

- Flowgraphs and path testing
- Transaction-flow testing
- Data-flow testing
- Domain testing
- Syntax testing
- Logic-based testing
- Transition testing

For example (**novel**): [Trucano]

- Spelling
- Grammar
- Punctuation
- Paragraph structure
- Syntax
- Logic

At the end of all of this you still don't have a novel.

Verification for a multi-physics code is more than code testing.

What tests?

How many of them?

What is the benefit?

What is the cost?

What are they telling you?

What are they not telling you?

What are the **metrics**?

Regression test suites need not be verification test suites, but they should be.

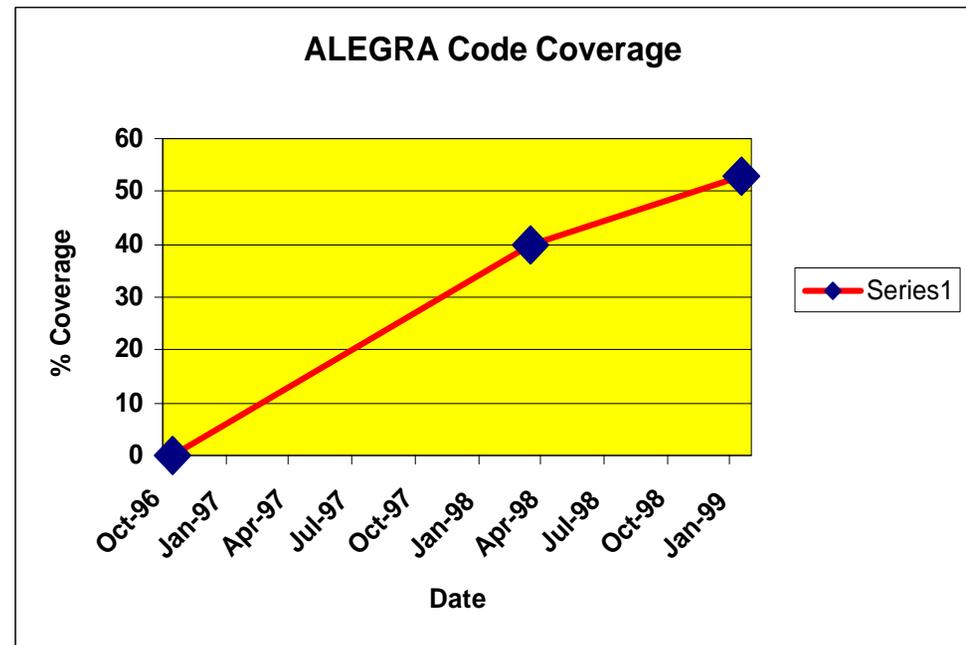
Regression testing measures implementation stability, not correctness. It is a software engineering practice, not a verification activity per se.

We do it and it leads to much interesting insight.

Our regression test set currently covers less than 60% of the functioning software (not including pre- and post-processors).

The appropriate metric is: **we spend much less time fixing bad checkins than we did before - but they still creep in, even in parts of the code covered by the test suite.**

Serves as the kernel for a much larger benchmark (verification) test suite.



Component testing and coverage metrics are linked for efficient high granularity testing.

What level of code component granularity is most effective for techniques like regression testing?

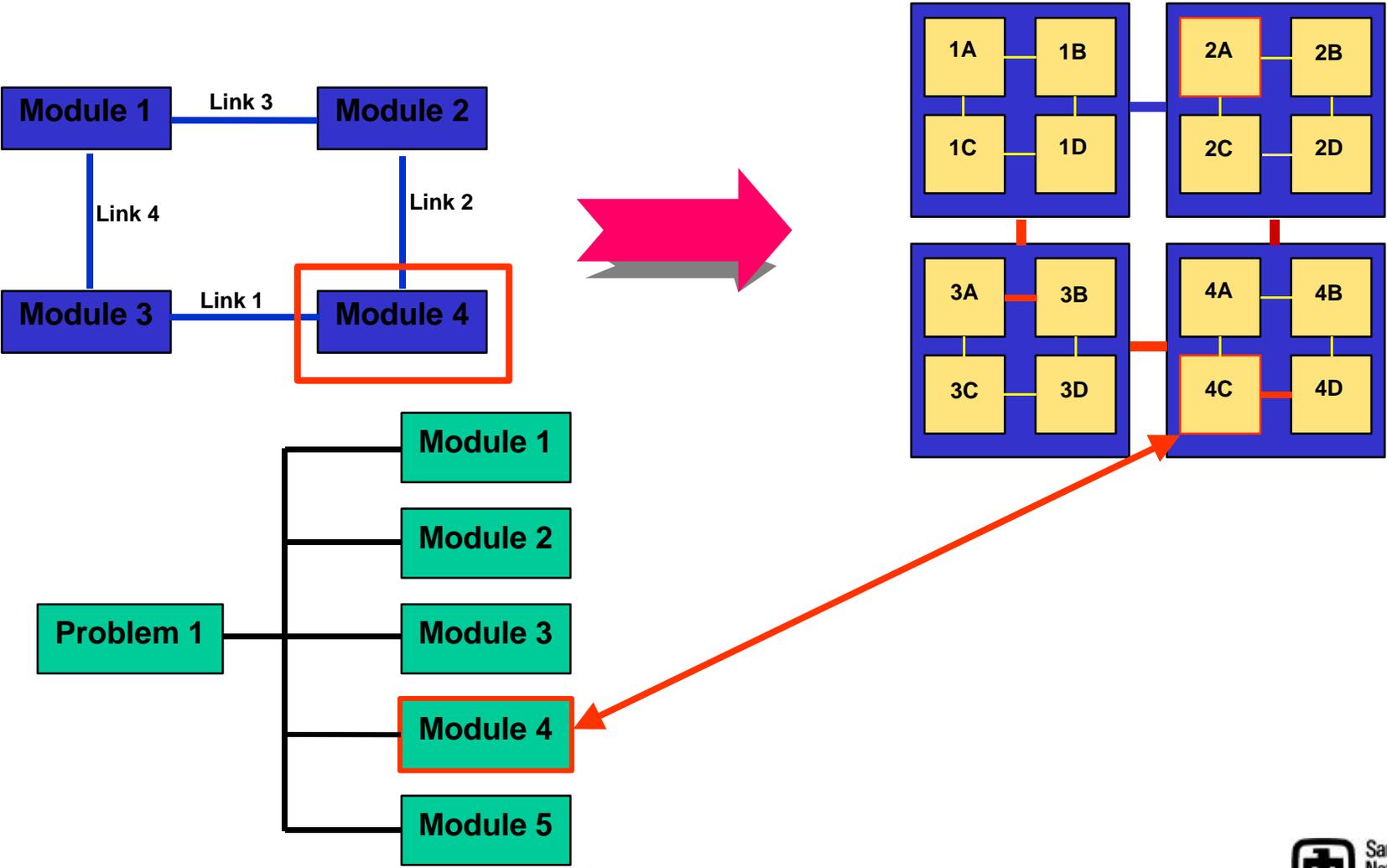
We need to assess both module and path coverage for specific problems. This requires tools.

We can't test every single "module" and path at the lowest level. We are driven to larger modules, which implicitly contain more paths.

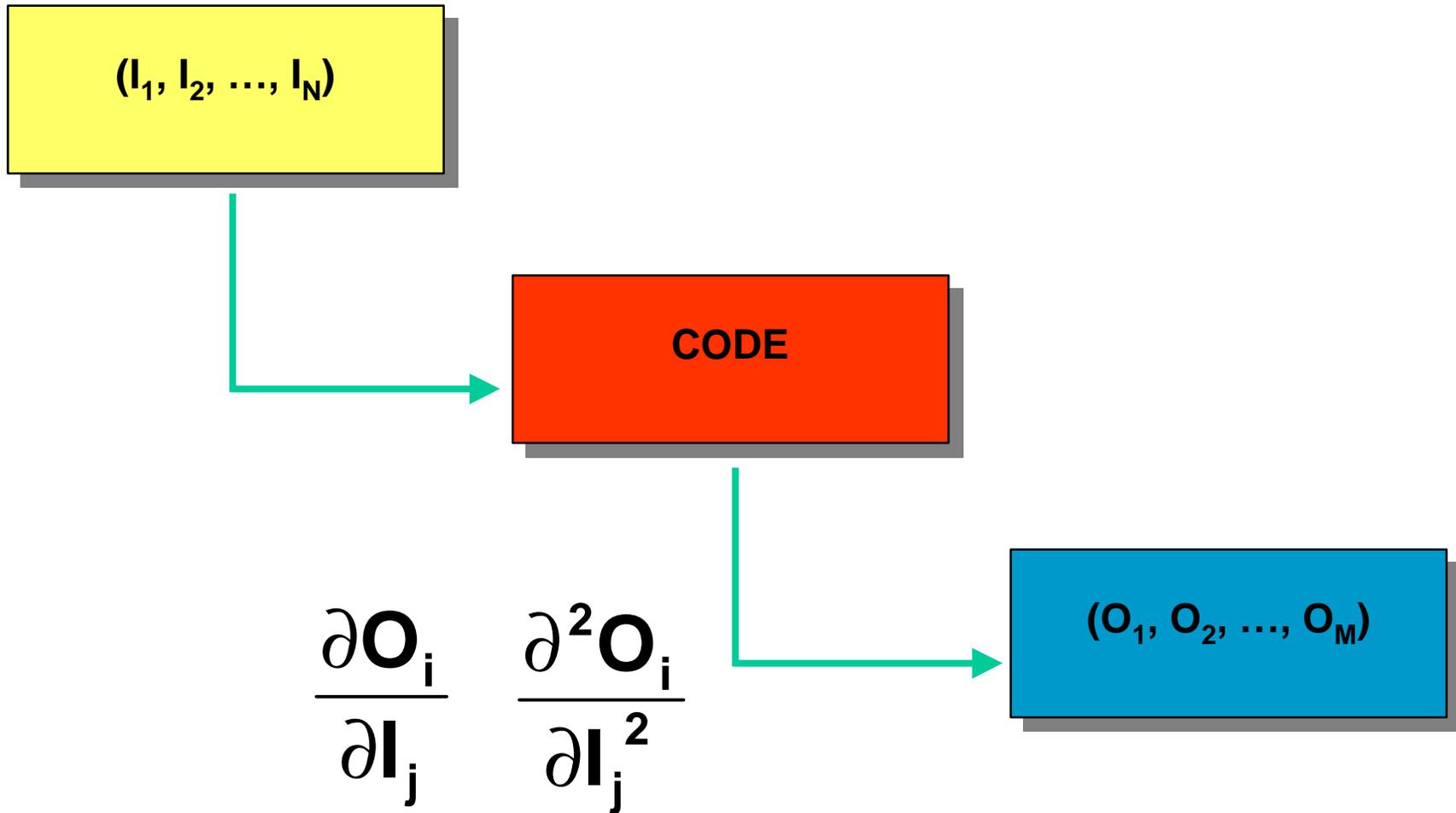
What is an appropriate module size for efficient testing (so it does not take two weeks of running before you can do a code check in?)

What is an appropriate measure of module importance? Of path importance?

Links between test problems and code modules/paths are needed for component testing and deciding granularity.



Recall that sensitivity analysis is like calculating partial derivatives.



Sensitivity analysis can be used to determine the most important modules for verification.

Sensitivity analysis can be used to refine the regression test suite, as well as the more intensive verification problem suites.

Coupled with suitable coverage tools, sensitivity analysis tools can suggest important paths as well as modules.

Sensitivity analysis can be used to develop measures of “importance” which help define metrics.

Footnote: How should sensitivity coefficients be calculated?

Deterministic methods (brute force numerical differentiation of the output is not recommended)

“Symbolic methods” - automatic differentiation (not yet applicable to us)

Probabilistic methods (regression based, for example)

Miscellaneous verification issues within ASCI code development.

How do we increase the fidelity of static testing?

Are “formal methods” worth the cost?

Is “Design by Contract” worth pursuing?

How do we “verify” the hardware?

Can we develop self-testing codes?

Static testing - Consider “The T Experiments”

L. Hatton (1997), “The T Experiments: Errors in Scientific Software,”
Computational Science and Engineering, Vol. 4, No. 2, 1997, 27-38

“The results were horrifying.” - Roache

Here are the lessons to me:

Well designed *static testing* is critical.

You need *tools* to do this.

You need *metrics* to do this.

The proper language to understand what Hatton is saying
is *reliability*, not physics.

Formal methods.

They are out there: NASA, Bell Labs, etc.

They are expensive.

They don't prove that your code implementation is correct.

Nobody in ASCI applications development is using them.

What are the metrics?

“Design by Contract”

Originates with Bertrand Meyer and Eiffel?

**Requires language support for full implementation ... or ...
appropriate program development environments?**

LLNL using this (?)

**Budge at Sandia has reported reported success in using the
concepts in C++ development in ALEGRA.**

While we are about it, let's not forget the hardware environment.

Should we be performing verification tests to confirm the acceptable performance of our hardware?

For example, should every “important” calculation (such as the verification suite acceptance testing) be preceded by a super-Paranoia-for-supercomputers (Stevenson) that insures that the compute environment on the machine is ready for that calculation?

Is this more important for heterogeneous computing systems, such as CPlant at Sandia? For distance computing?

“How expensive is it?”, Unknown supercomputer user...

What about self-generated test problems?

Simple:

Invert sources, say, to determine coefficients, then solve the resulting direct problem and confirm you match the source. [FUEGO doing this]

Heuristic:

Synthesize test problems from a sampling of user problems.

Do this continuously?

AI:

More sophisticated approach to sensing code weaknesses and designing test problems to attack those weaknesses

IV. Do we need validation “science”?

“...we never (least of all in science) draw inferences from mere observational experience to the prediction of future events. Rather, each such inference is based upon observational experience...plus some universal theories...”

Karl Popper, Objective Knowledge

What is “uncertainty quantification” and why do I care?

Uncertainty quantification: an anti-reductionist measure of “error”.

The forward prediction problem:

Characterize the “input” uncertainty (stochastic, fuzzy, etc)

Propagate this uncertainty through the code

Characterize the resulting output uncertainty

Refine this characterization via comparison with data

Develop “code reliability” metrics and statements (need requirements)

(Most of this is not rocket science for an initial implementation.)

Now follow it with backward prediction:

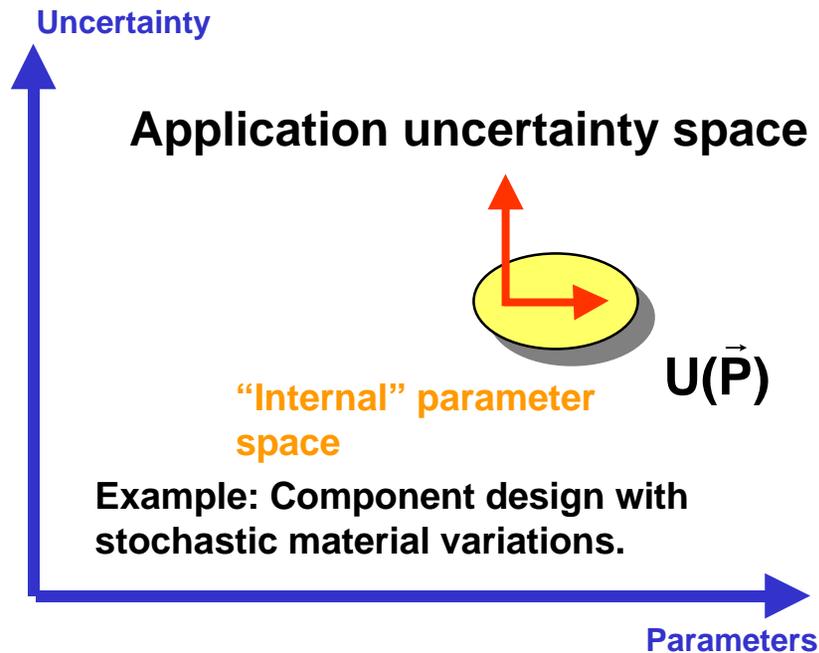
Reduce the code uncertainty via the output uncertainty characterization (Bayesian?).

(This IS rocket science.)

Now optimize:

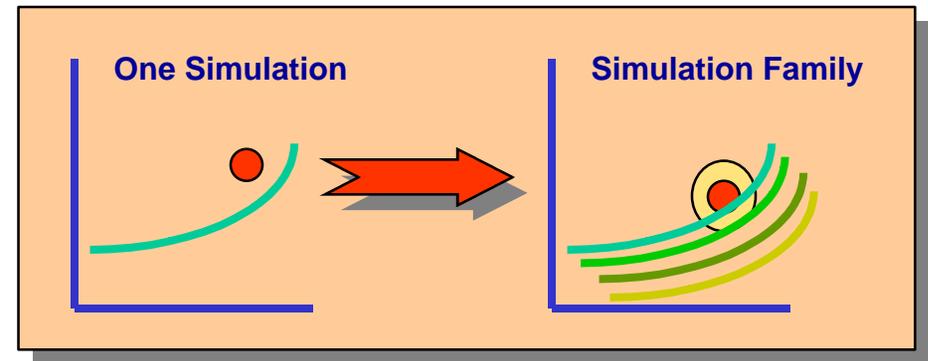
Perform forward/backward prediction sweeps to increase “code reliability” and guide new experiments.

Local validation begins with local uncertainty quantification, a high dimensional problem.



Local uncertainty quantification performs systematic studies of code uncertainty from stochastic treatments of parameter uncertainty. U is a random variable, P is a random vector.

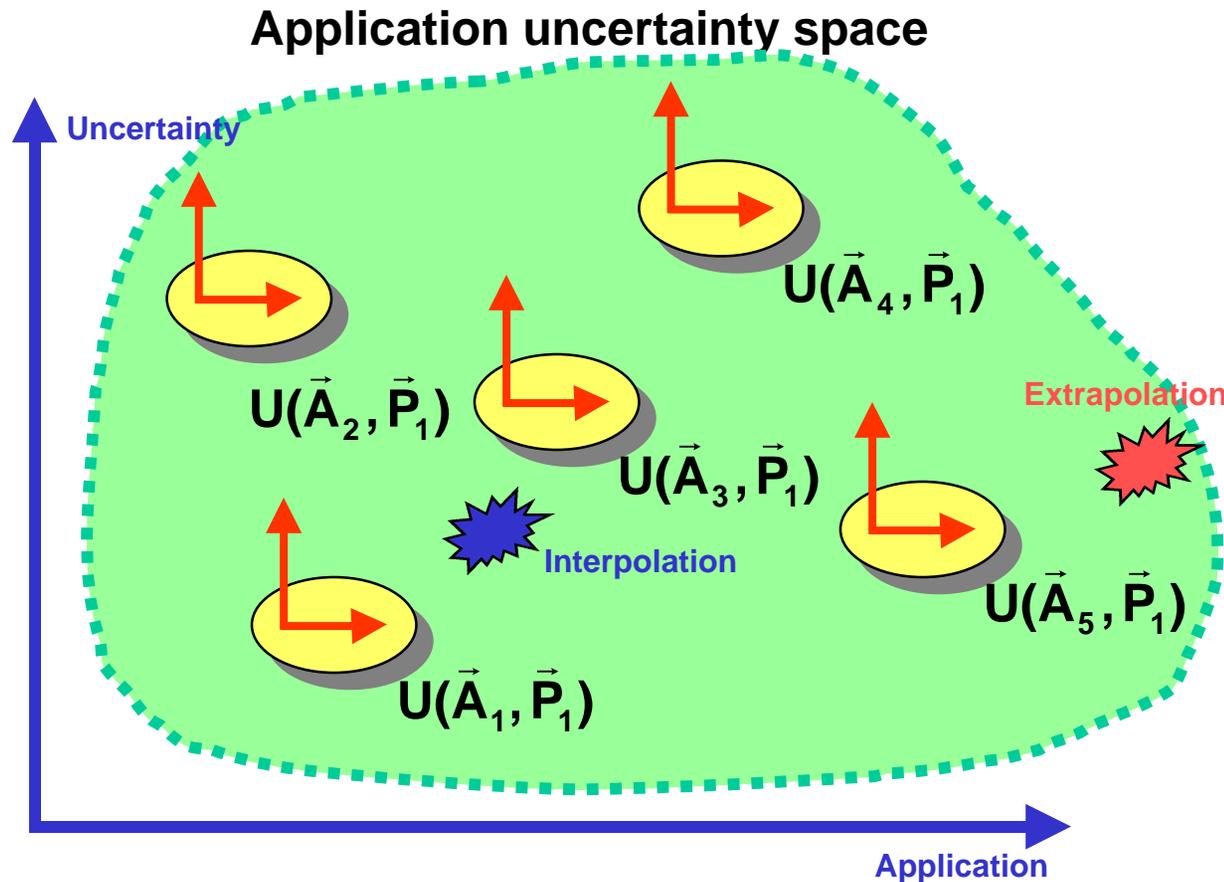
Notice that how uncertainty U is defined is an issue.



Alternatively, think of results being replaced by probability distributions of results. Then, do response surface methods, or other things, to characterize the result family stochastically. The most important issue is this characterization.

The number of parameters can be enormous - *parsimony* is hoped for (but unlikely?).

“Global” validation leads to global uncertainty quantification, an even higher dimensional problem with “internal” and “applications” parameters.



Uncertainty has two components:

“Local” - uncertainty quantification and local data comparisons.

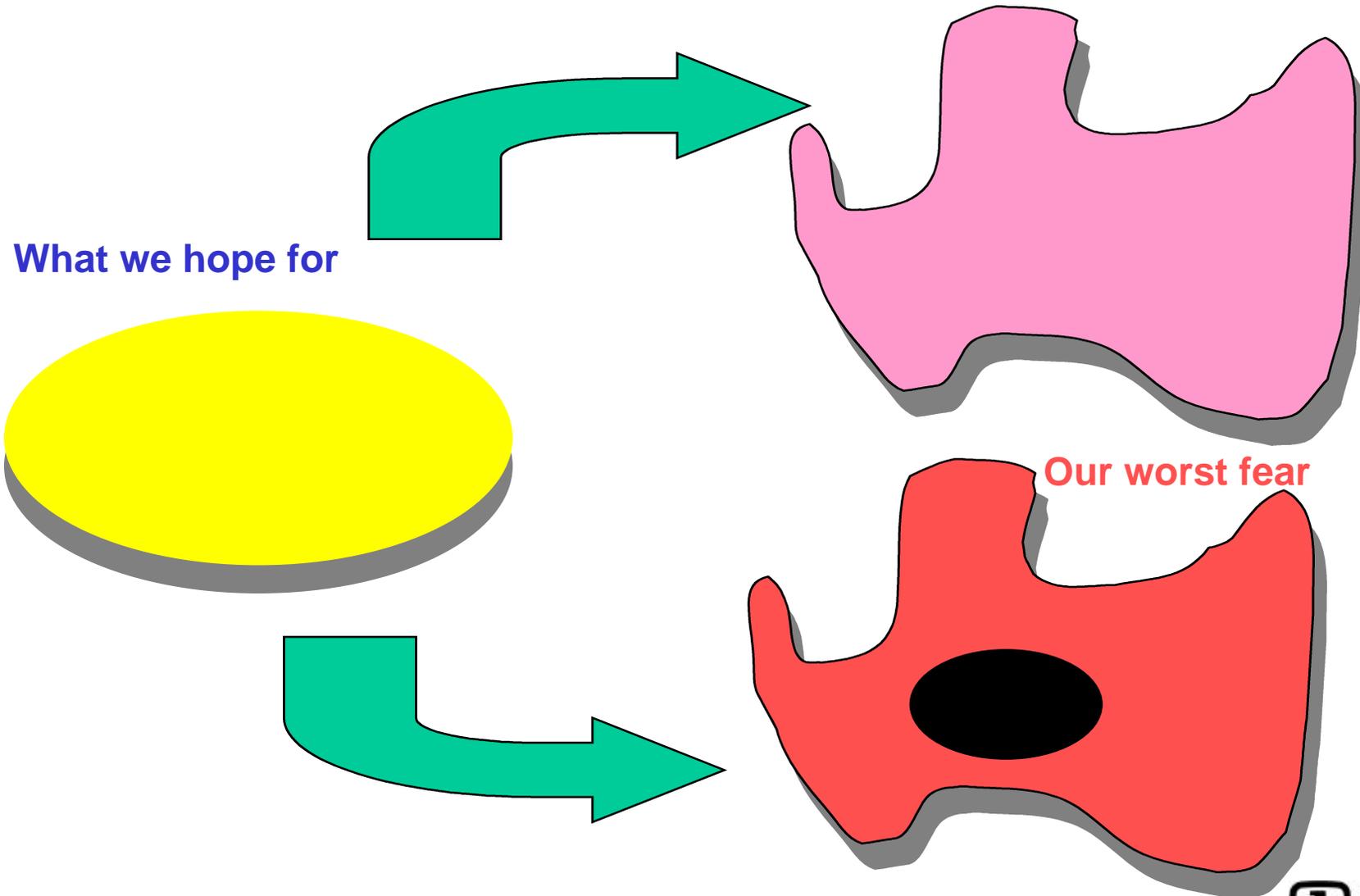
“Global” - system scale uncertainty quantification and global data comparisons.

Can we use spatial statistics methods (like kriging) to characterize $U(A,P)$?

Note that we have simplified the problem by assuming that the internal parameter functional dependence is constant over application space. Is this true?

Example: NIF ignition capsule design confidence levels based on uncertainty quantified NOVA design calculation experience.

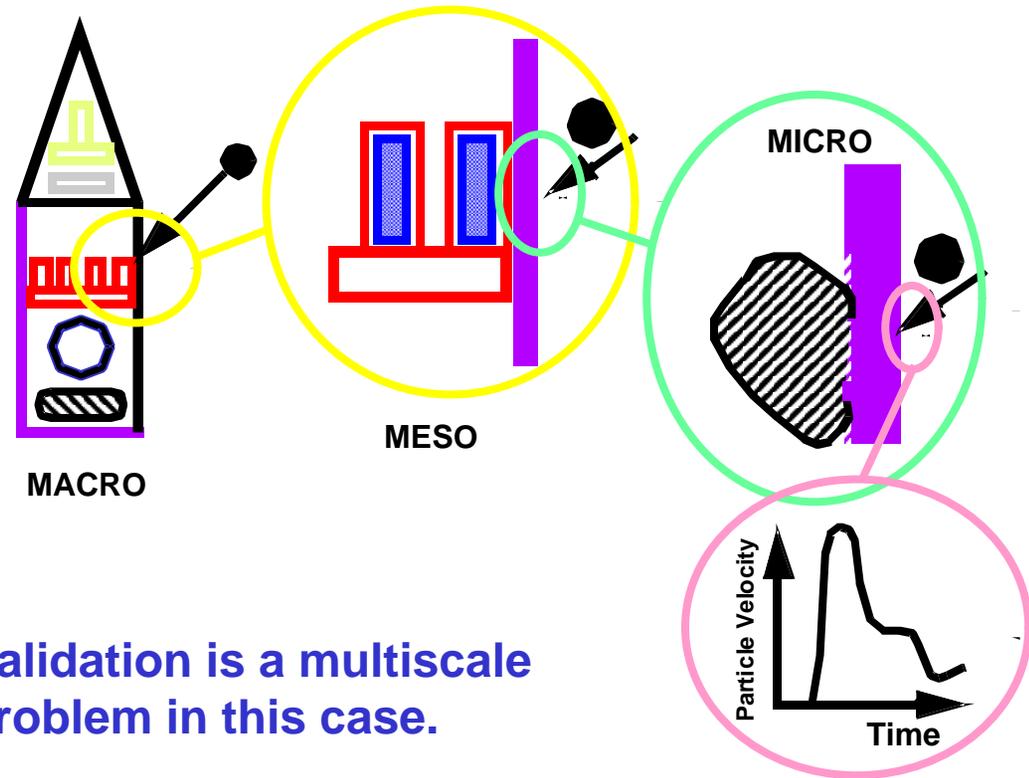
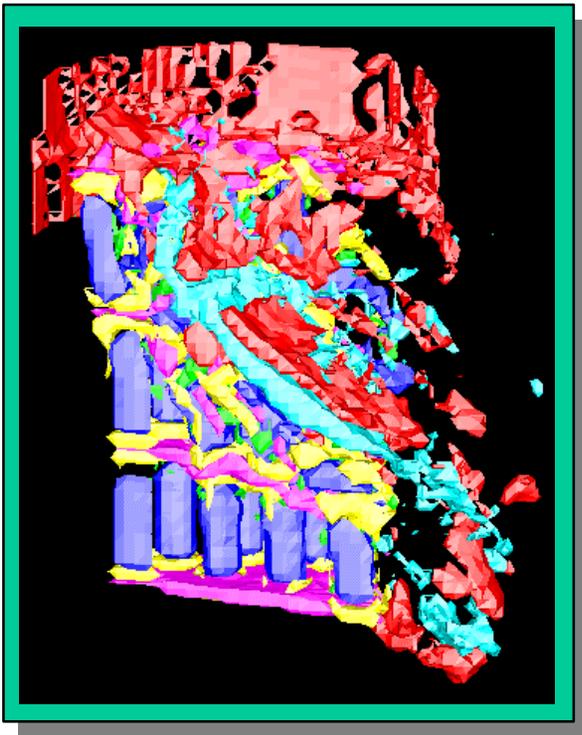
“Uncertainty” probably looks a lot more complex than suggested by the previous figures.



Example - Hypervelocity Impact

Missile defense simulations provide an excellent application for studying predictive complexity.

There are at least six stochastic parameters to begin with: the hit point and the engagement velocity

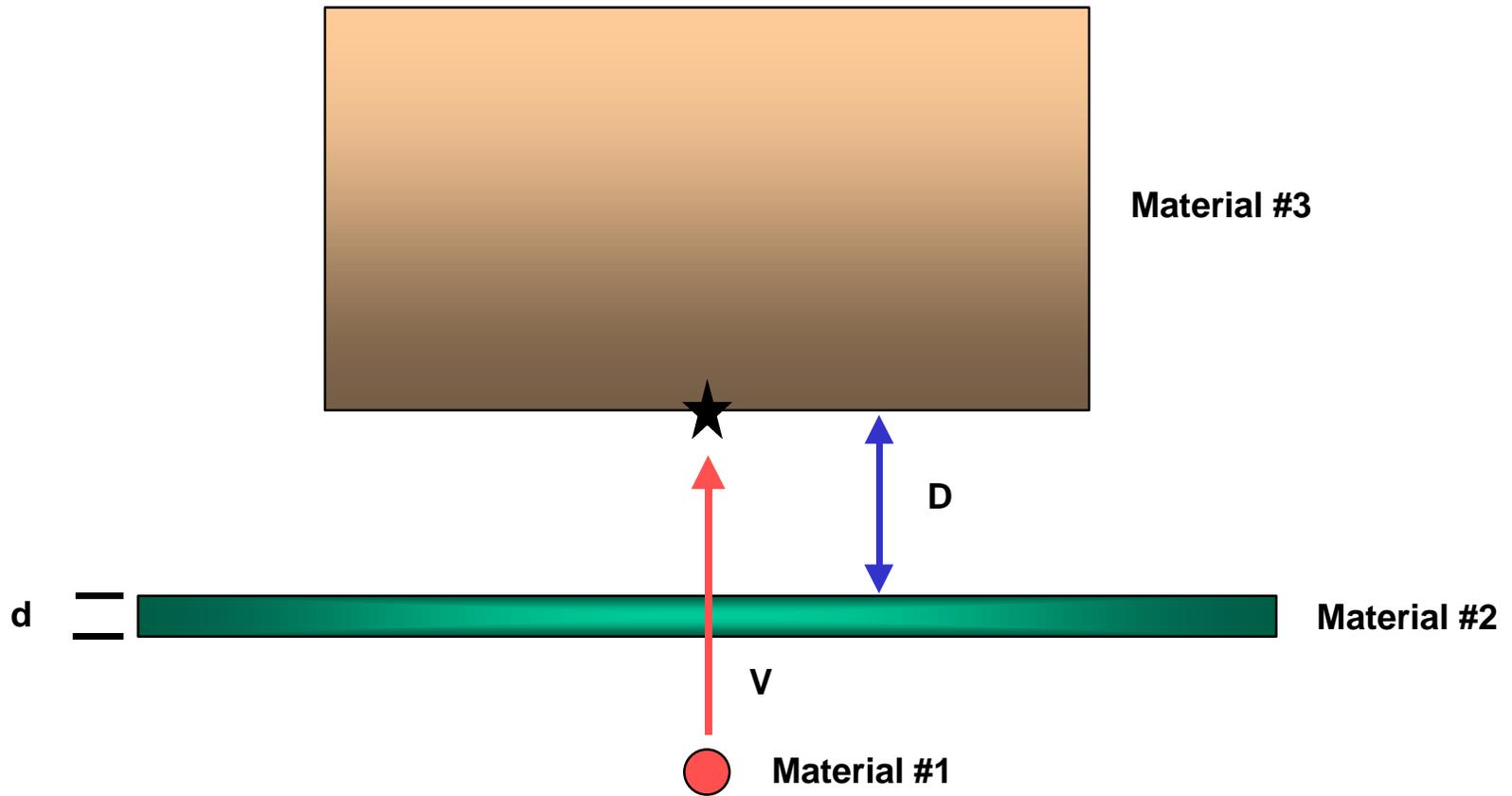


Validation is a multiscale problem in this case.

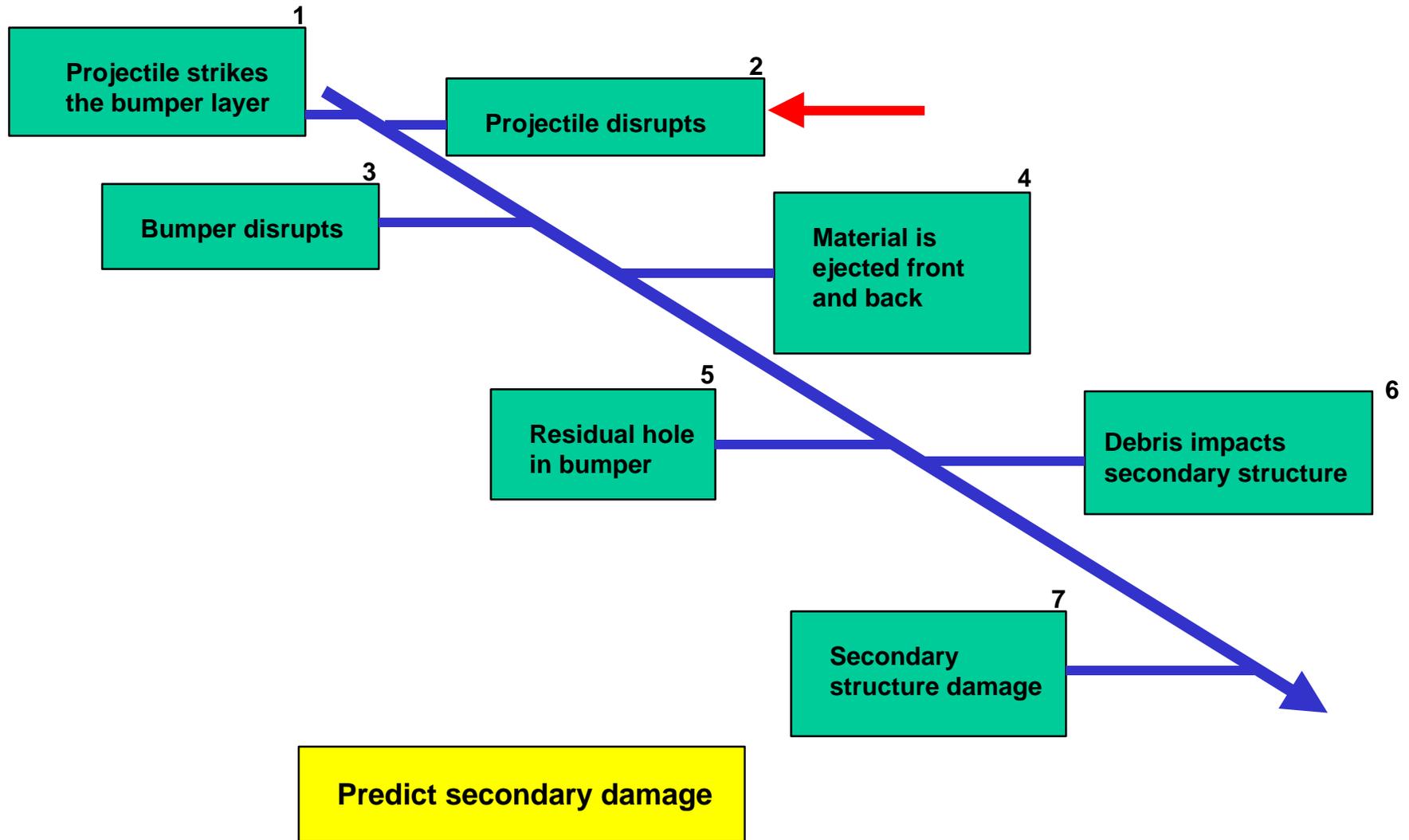
The necessary information is a mix of qualitative and quantitative.

“Validation data”.

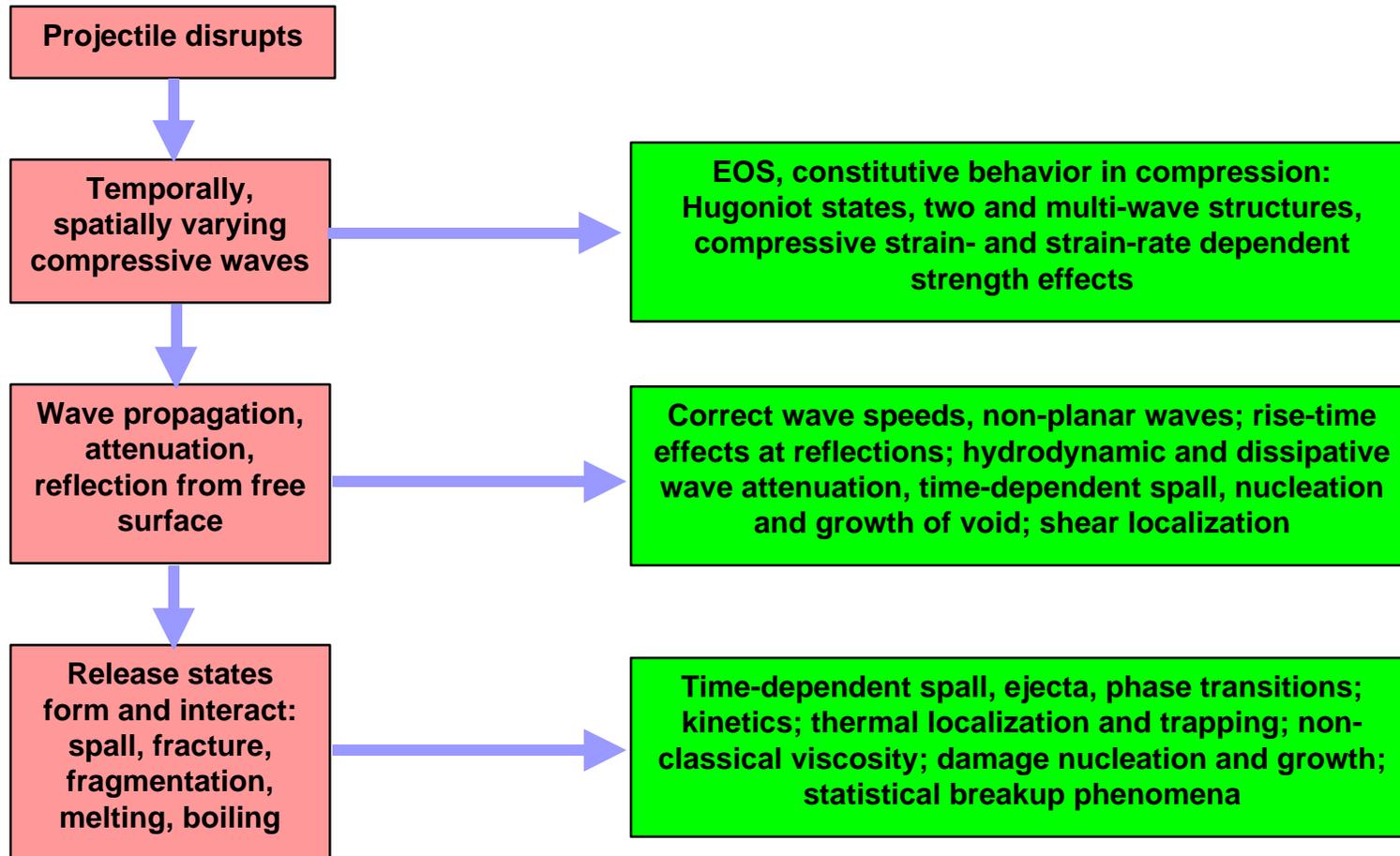
Consider validating the microscale with laboratory hypervelocity impact experiments.



Levels of phenomenology at the microscale:



Further levels of phenomenology in this experiment:



The parameter space for these experiments:

“Experimental” parameters:

- Impact location (2)
- Velocity vector (3)
- D, d (2)
- Projectile geometry (1)
- Materials (none)

Total = 8

“Internal” parameters:

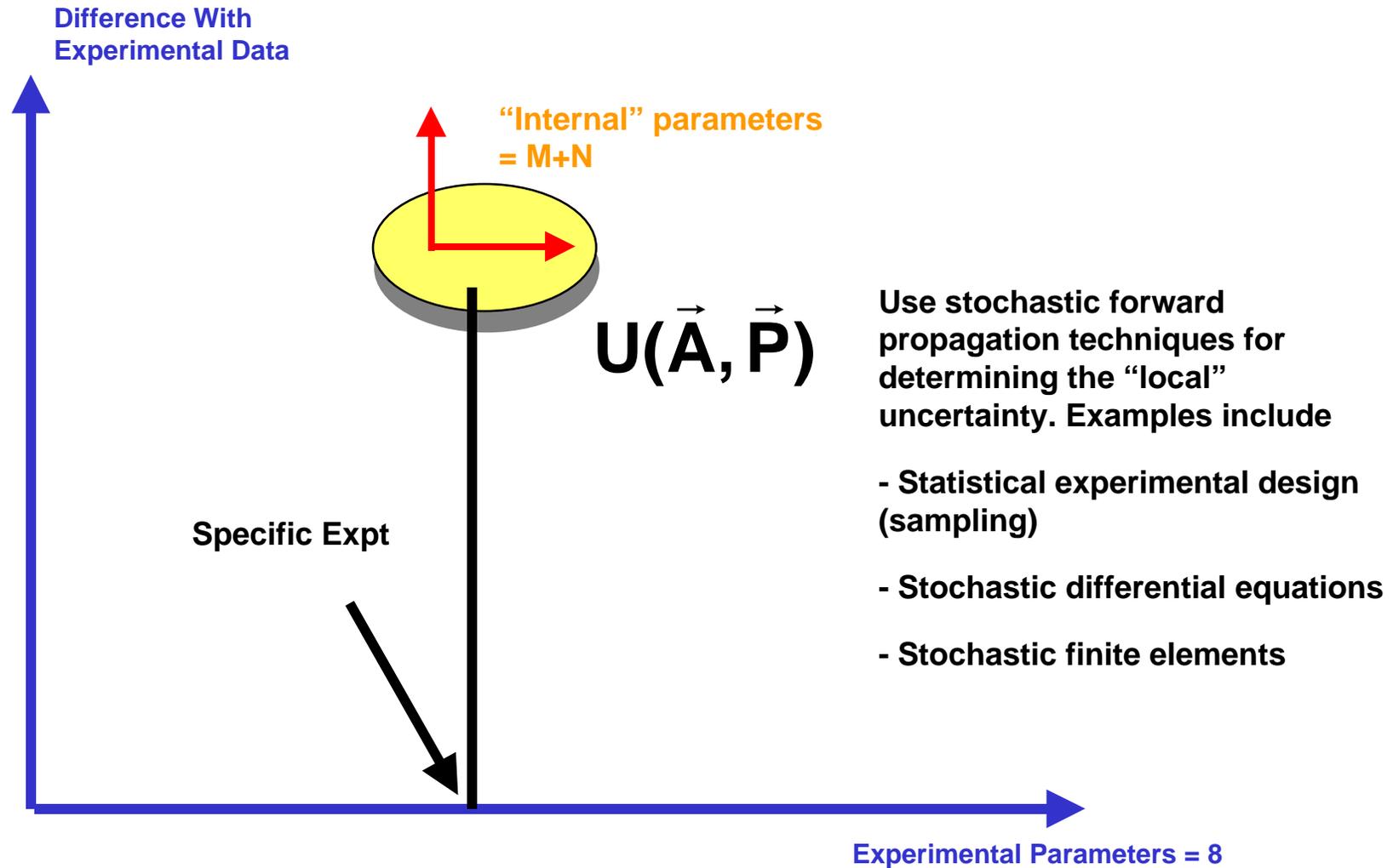
- Hydrodynamic parameters (H)
- Material Models (M)

Total = H+M

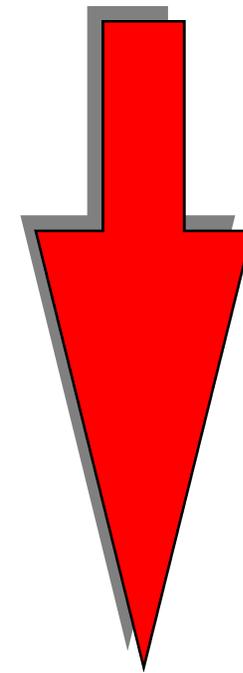
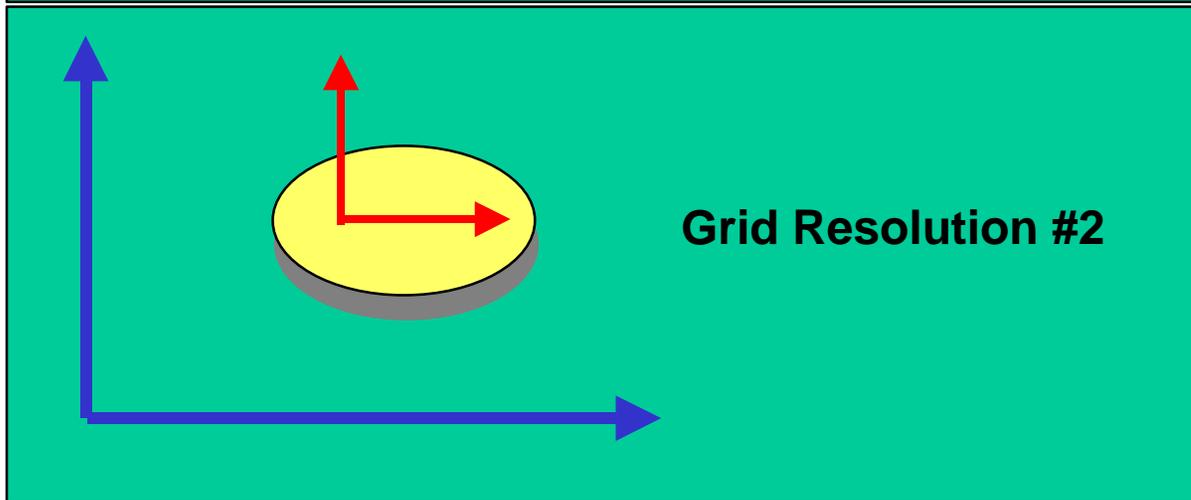
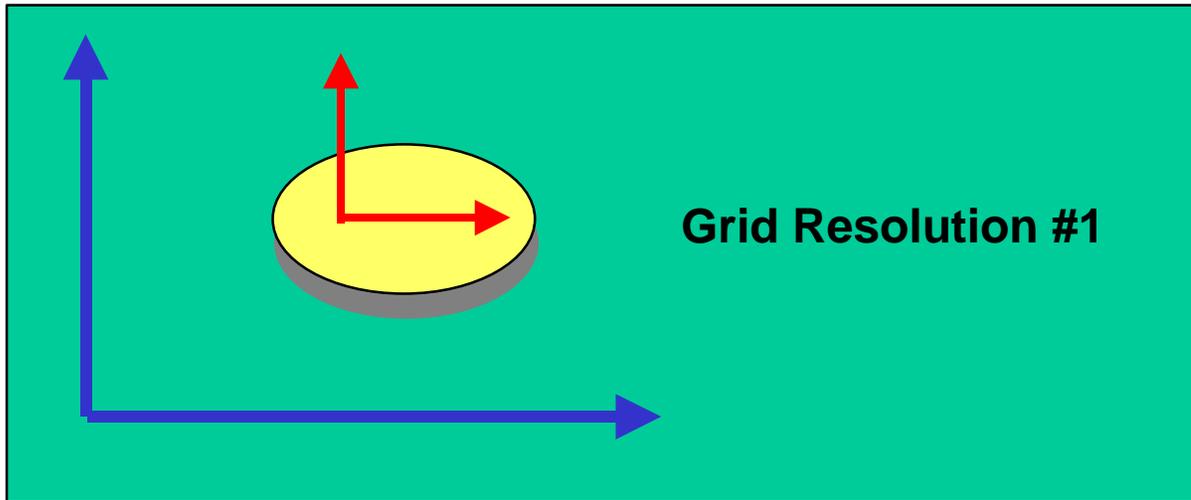
“Uncertainty” Metrics:

- Time-resolved data in witness material (PVF gauges)
- Time- and spatially-resolved debris cloud data (radiography)
- Target recovery and inspection

Parameter versus uncertainty space



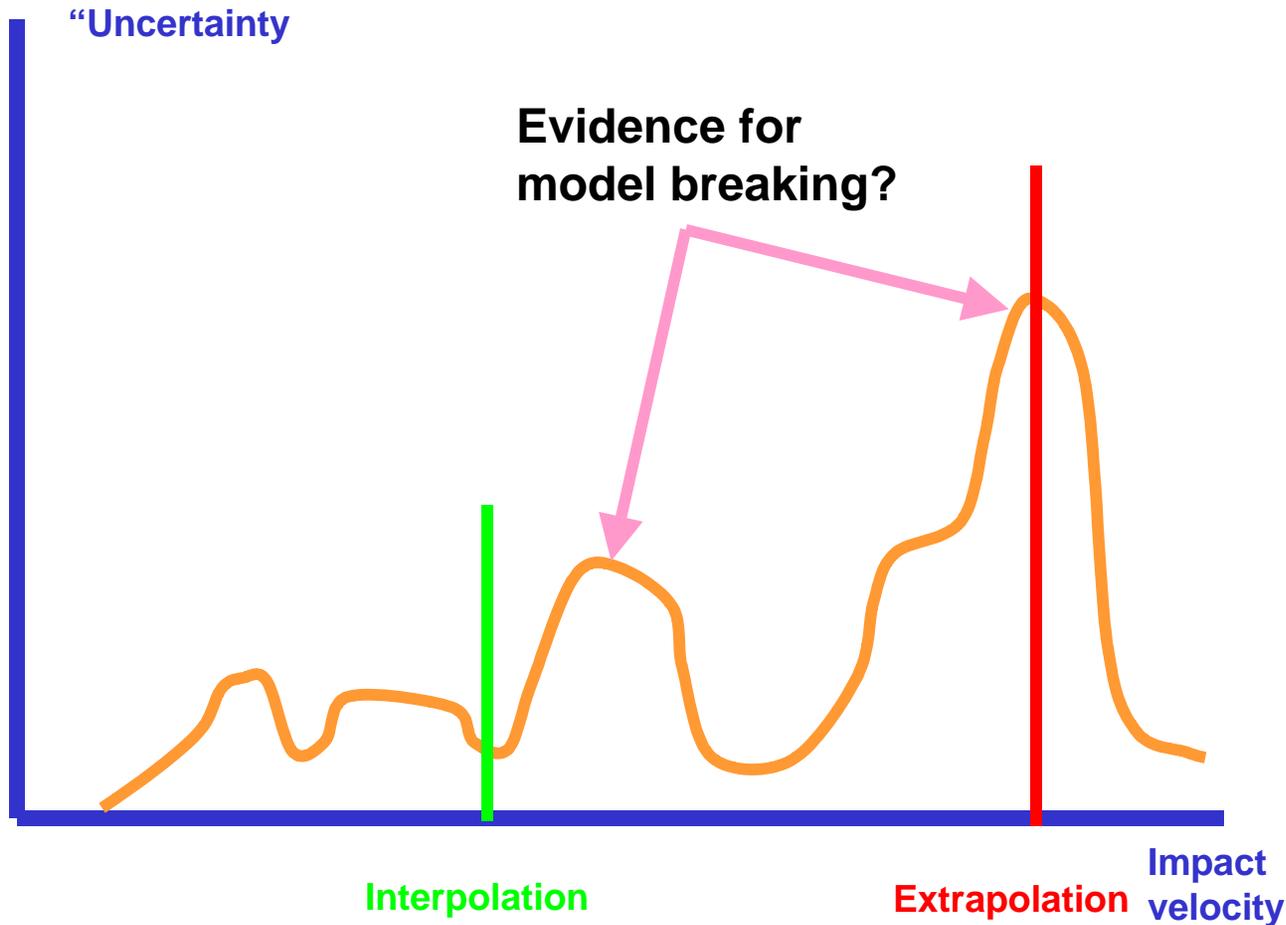
One way to worry about grid resolution:



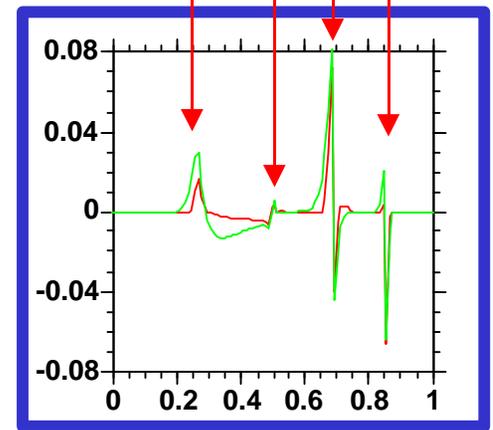
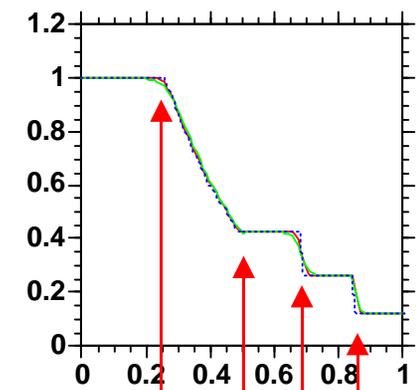
Increasing grid resolution

Increasing grid resolution does not mean uniform refinement (ALE, adaptivity, geometry constraints). Algorithm parameters controlling dynamic grid resolution are included in the internal parameters.

A thought experiment: projecting the uncertainty



Sod Problem



Thanks to Kamm and Rider

Some speculation and reasonable opportunities for research: Uncertainty as a spatial random field

I. Let the uncertainty be represented by a $(M+H+8)$ dimensional spatial stochastic process

For example, we end up worrying about properties of something like the variogram:

$$\text{var}[U(\vec{A}_j, \vec{P}_j) - U(\vec{A}_i, \vec{P}_i)] = 2\gamma((\vec{A}_j, \vec{P}_j) - (\vec{A}_i, \vec{P}_i))$$

Then, we can develop predictors for U at other points, such as the BLUP (**Best Linear Unbiased Predictor**):

$$\hat{U}(\vec{A}_0, \vec{P}_0) = \sum_{i=1}^n \lambda_i \hat{U}(\vec{A}_i, \vec{P}_i)$$

Or (μ = mean of random field, S = fine structure, and ε = “measurement error”):

$$\tilde{U}(\vec{A}, \vec{P}) = \mu(\vec{A}, \vec{P}) + S(\vec{A}, \vec{P}) + \varepsilon(\vec{A})$$

Comments:

See J. Sacks, W. J. Welch, T. J. Mitchell and H. P. Wynn (1989), “Design and Analysis of Computer Experiments,” Statistical Science, Vol. 4, No. 4, 409-435; N. Cressie (1988), “Spatial Prediction and Ordinary Kriging,” Mathematical Geology, Vol. 20, No. 4, 405-421.

Do we need another framework other than probability to do this?

What is the appropriate way to partition this random field among the experimental parameters and the internal parameters? Does this question even make sense?

What structure do we require on the projected random field $U(A,P)$ to facilitate piecing together the various uncertainties?

Speculation: Sensitivity coefficients

II. Sensitivity studies define which of the H+M+8 parameters is most important. Probabilistic evaluation of the sensitivities is of interest.

Is parsimony really true?

Does the sensitivity structure projected onto the internal parameter space remain invariant as the experimental parameters alone vary? If no, does this imply model invalidity?

Does the sensitivity structure remain invariant over grid variations?

Don't assume that the parameters are all uncorrelated. Then we need "interaction" coefficients.

The literature on sensitivity analysis is huge. See M. D. McKay (1995), "Evaluating Prediction Uncertainty," Los Alamos Report, LA-12915-MS for one important approach.

Speculation: Model calibration

III. Some understanding of U should lead to improvement in the model. “Calibration” reduces U locally in the application space by optimizing the internal parameters.

How does the calibration vary with the experimental parameters?

Comments:

A Bayesian approach can be applied to the formal study of improving model uncertainty in the presence of parameters derived via comparison with experimental data. See, for example, K. M. Hanson (1998), “A Framework for Assessing Uncertainties in Simulation Predictions,” Los Alamos preprint.

Statistically rigorous comparisons between uncertain calculations and uncertain data with the intent of providing code validation are the subject of a recent tutorial report by R. G. Hills (1998), “Statistical Validation of Engineering and Scientific Models: Background,” Sandia National Laboratories Contract Report.

This question also leads to the use of “surrogates” for studying parameter calibration, as well as other optimization questions associated with code uncertainty. Consider the important work A. J. Booker, et al (1998), “A Rigorous Framework for Optimization of Expensive Functions by Surrogates,” Boeing Shared Services Group Report, SSGTECH-98-005.

Other papers that the reader might find of interest are D. D. Cox, J. Park, and C. E. Singer (1996), “A Statistical Method for Tuning a Computer Code to a Data Base,” Rice University Department of Statistics Report 96-3 and M. B. Beck (1987), “Water Quality Modeling: A Review of the Analysis of Uncertainty,” Water Resources Journal, Vol. 23, No. 8, 1393-1442.

Speculation: Structural (Model) uncertainty

IV. Is there anyway to deal with “structural” uncertainty?

A Bayesian structure can be developed for considering structural uncertainty. See D. Draper (1995), “Assessment and Propagation of Model Uncertainty,” J. R. Statist. Soc. B, Vol. 57, No. 1, 45-97.

This involves developing posteriors via conditioning over the space of models, a rather hopeless endeavor on the face of it. Additional structure might make this more feasible.

Model uncertainty is often treated in multi-physics code through the introduction of tuning parameters. If a code (sub)model is built out of sub-submodels:

$$M = \sum_j M_j$$

Uncertainty about the overall model is then treated by modifying this equation to:

$$M = \sum_j \alpha_j M_j$$

Add $(\alpha_1, \dots, \alpha_m)$ to the parameter list and proceed as before.

Speculation: Is probability appropriate for this discussion?

How do we treat “variability” in the assumed distributions on the parameters to do experimental design, sensitivity analysis, and forward propagation?

Is probability the canonical way to capture “uncertainty?”

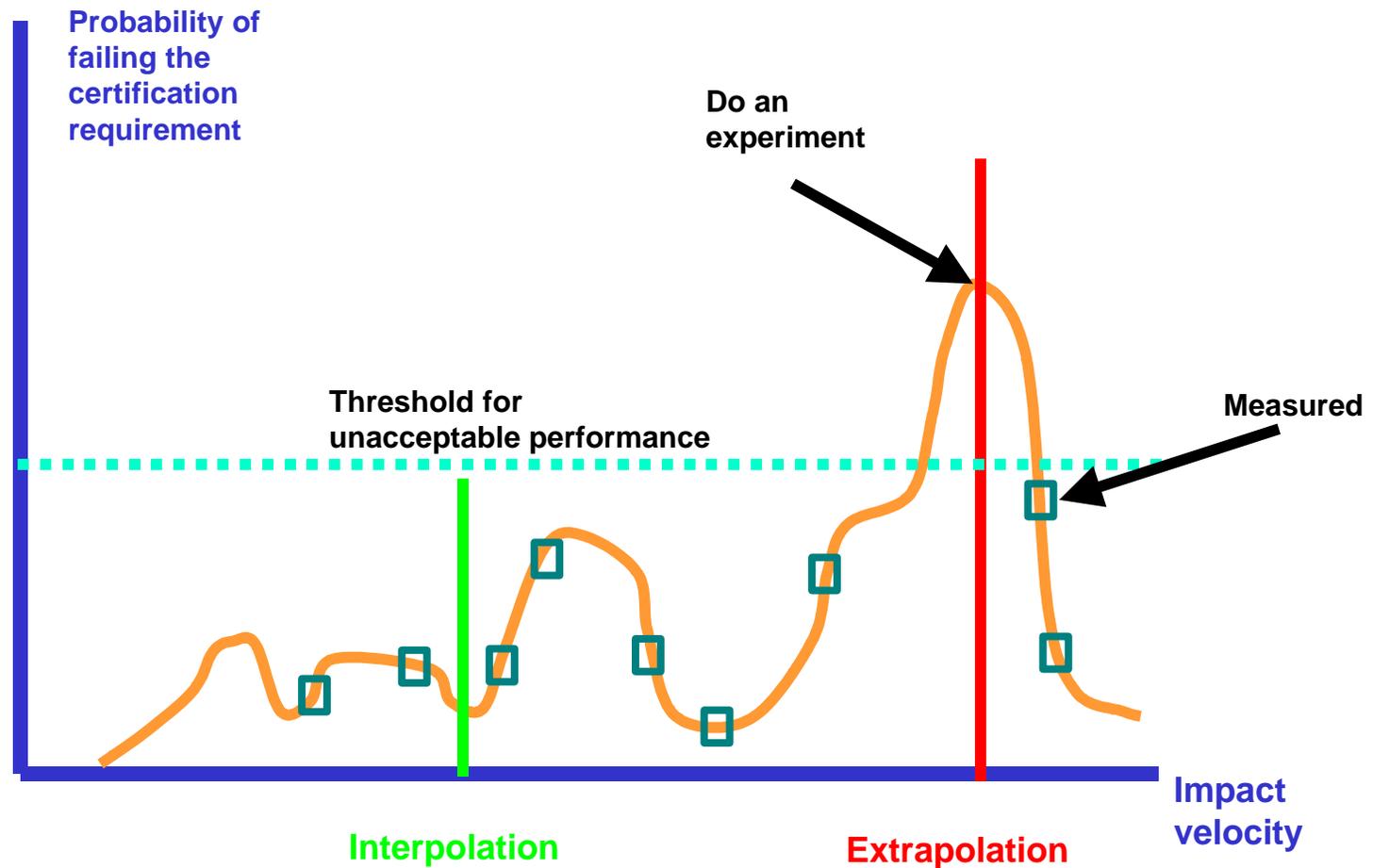
Is saying “I don’t know what the value of a parameter is” the same as placing a probability distribution on it?

We need to design validation processes to attack the weak points of codes.

"...the theorist is interested in explanation as such, that is to say, in testable explanatory theories: applications and predictions interest him only for theoretical reasons - because they can be used as tests of theories..."

Karl Popper, The Logic of Scientific Discovery

Speculation: “Certification” leads to quantitative “reliability” analysis.



DOOMSDACE: Distributed Object-Oriented Software With Multiple Samplings for the Design and Analysis of Computer Experiments

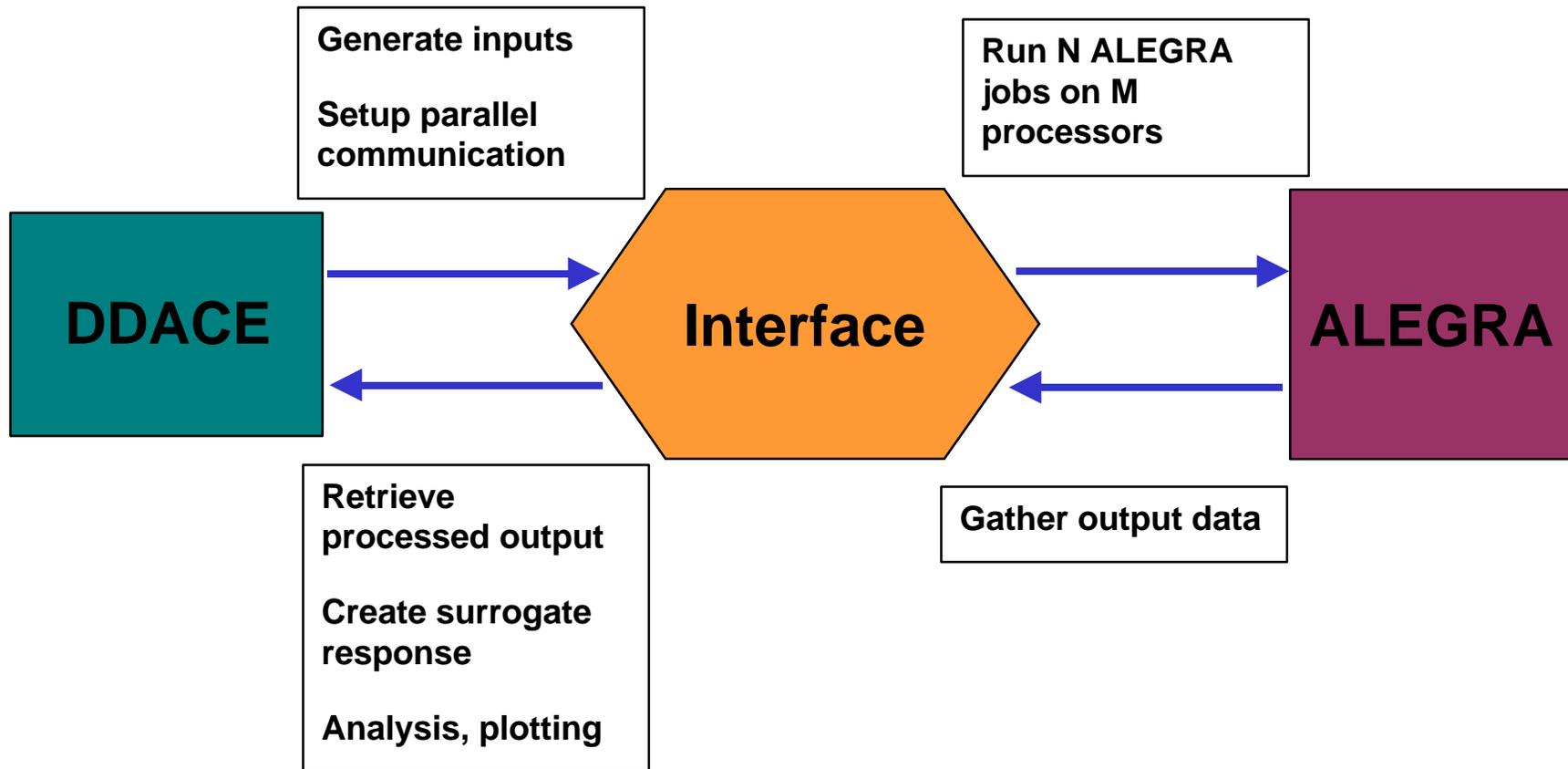
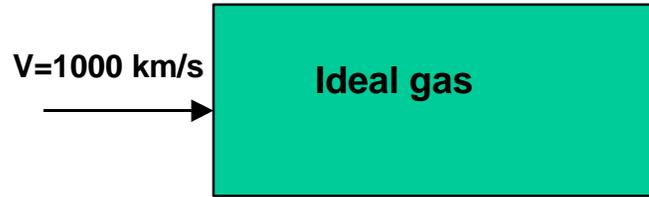
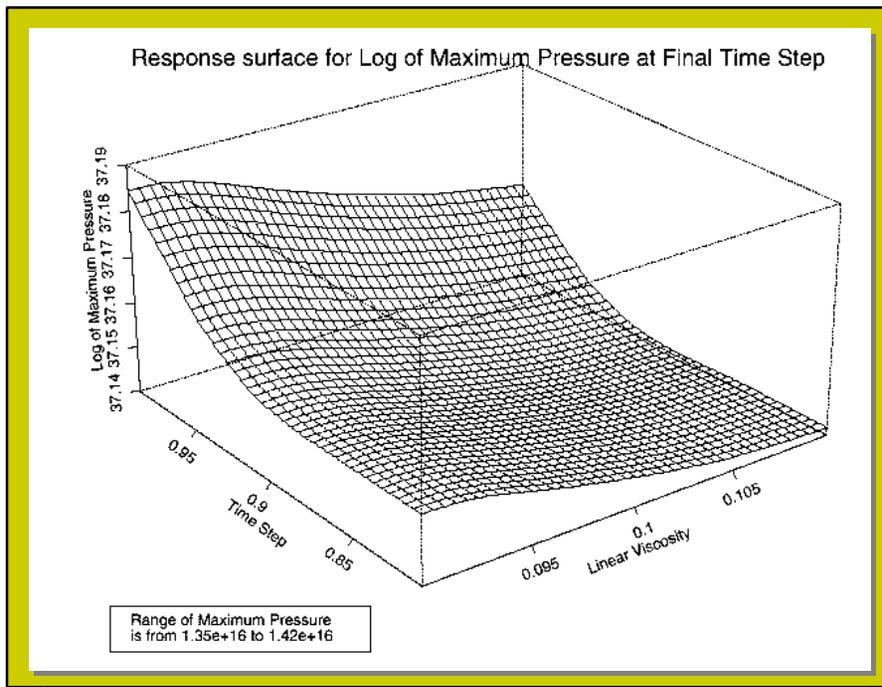


Illustration of current work: 1-D “Saltzman” problem



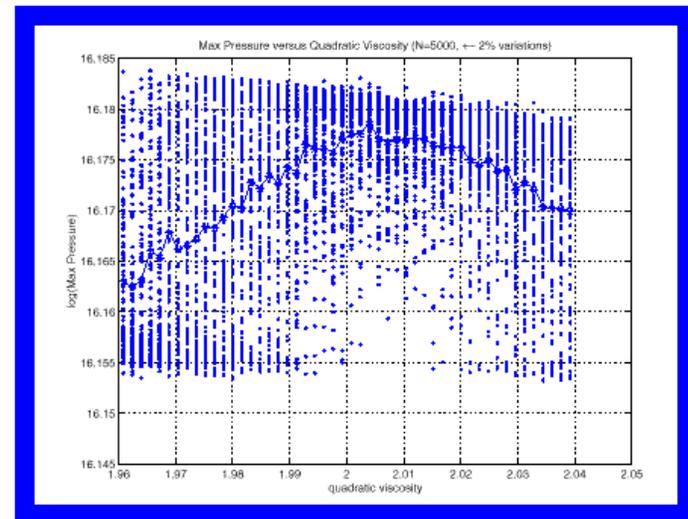
McKay “correlation ratio” analysis

Linear Viscosity	0.36
Quadratic viscosity	0.04
Hourglassing	0.02
Courant Limit	0.3



Response surface for $\log P_{\max}$ vs linear viscosity vs Δt Courant limiting

P_{\max} vs quad viscosity



In conclusion:

"Summary: Computers Are Here To Stay. They Endanger Thought, Language, Science, and the Survival of Man. Like Any Other Dangerous Tool, They Should Be Put Under Strict Controls."

Clifford Truesdell, "The Computer: Ruin of Science, Threat to Man" in An Idiot's Fugitive Essays on Science