Argonne NATIONAL LABORATORY

Improving the **Trust** in Results of Numerical Simulations and Scientific Data Analytics

Franck Cappello
Argonne/MCS

Improving the Trust in Results of Numerical Simulations and Scientific Data Analytics Cappello, F, Constantinescu, EM, Hovland, PD, Peterka, T, Phillips, CL, Snir, M, Wild, SM, report ANL/MCS-TM-352

Why Trust is becoming important

- Solutions are identified for most of the hard problems in Fault Tolerance for HPC.
  - Checkpointing cost, fault tolerant protocols, optimization of ckpt interval, ABFT, etc.

- Fundamental problems that are still open or new:
  - Detection of Silent Data Corruption (SDCs) with minimal overhead
  - Forward recovery (from fail stop and transient errors)
  - New: Rollback recovery from approximate state (lossy compression)

→All relate to the **scientific data integrity** problem

→All relate to **result trustworthiness**

What is Trust (briefly)?

- Trust research aims to improve the confidence (with some quantification if possible) on the results (of numerical simulations and data analytics)

- Trust focuses on the product of the execution
  → direct connection to the applications and users
  → defines required execution properties based on the result expectations

- What could impair trust on scientific results: **corruptions**

- It is much more complicated problem than FT&Resilience:
  - Related to Validation and Verification, Uncertainty quantification, etc.
  - Errors + Bugs + Attacks
  - It involves users

Lack of Trust definition in HPC

- Avizienis, Laprie: "**the ability to deliver service that can justifiably be trusted**"

Concept	Dependability	Survivability	Trustworthiness
Goal	1) ability to deliver service that can justifiably be trusted 2) ability of a system to avoid failures that are more frequent or more severe than is acceptable to the user(s)	capability of a system to fulfill its mission in a timely manner	assurance that a system will perform as expected
Threats present	1) development faults (e.g., software flaws, hardware errata, malicious logic) 2) physical faults (e.g., production defects, physical deterioration) 3) interaction faults (e.g., physical interference, input mistakes, attacks, including viruses, worms, intrusions)	1) attacks (e.g., intrusions, probes, denials of service) 2) failures (internally generated events due to, e.g., software design errors, hardware degradation, human errors, corrupted data) 3) accidents (externally generated events such as natural disasters)	1) hostile attacks (from hackers or insiders) 2) environmental disruptions (accidental disruptions, either man-made or natural) 3) human and operator errors (e.g., software flaws, mistakes by human operators)

- In Sigsoft *Software Eng. Notes:* " **trust depends on many elements**: safety, correctness, reliability, availability, confidentiality/privacy, performance, certification, and security."
- In social sciences: "One party (trustor) is willing to rely on the actions of another party (trustee)" and "The trustor is uncertain about the outcome of the other's actions; **they can only develop and evaluate expectations**."

Why Trust research is important?

- There are many examples of execution producing bad results due to some form of result corruption.
- Let's start with an example in the space industry:

The Ariane 5 reused the inertial reference platform from the Ariane 4, but the Ariane 5's flight path differed considerably from the previous models. Specifically, the Ariane 5's greater horizontal acceleration caused the computers in both the back-up and primary platforms to crash (The greater horizontal acceleration caused a data conversion from a 64-bit floating point number to a 16-bit signed integer value to overflow and cause a hardware exception.).
A range check would have fixed the problem…

Explosion of Ariane 5
Loss of more than US$370 million
+population evacuation
+ loss of scientific results

Why Trust research is important?

- Other examples with catastrophic consequences:
  - See http://ta.twi.tudelft.nl/users/vuik/wi211/disasters.html for list of num. errors
  - See https://en.wikipedia.org/wiki/List_of_software_bugs for list of bugs
  - See http://www5.in.tum.de/~huckle/bugse.html for an even longer list of bugs.

- Consequences can be significant in the context of scientific simulations and data analytics
  - Wrong decisions may have been taken
  - Large number of executions may be corrupted before discovery
  - Post-mortem verification requires heavy checking
  - lead also to significant productivity losses.

The sinking of the Sleipner A offshore platform (inaccurate finite element approximation)

Agenda

- Corruption classification and origins
- Sources of corruptions (with examples!)
- Examples of corruption propagation
- Why existing techniques only help partially
- What strategies?
- Example of External Algorithmic Observer
- This is just a beginning

Agenda

- **Corruption classification and origins**
- Sources of corruptions (with examples!)
- Examples of corruption propagation
- Why existing techniques only help partially
- What strategies?
- Example of External Algorithmic Observer
- This is just a beginning

Not all corruptions are equal

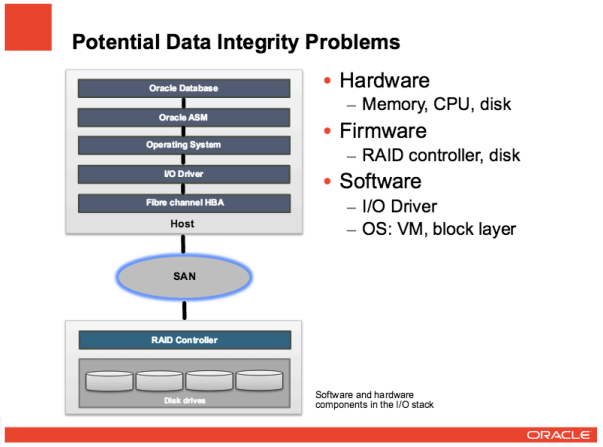
Note: All corruptions leading to the execution hanging or crashing or to results obviously wrong are beyond the scope of this keynote.

Some corruptions are expected, controlled, and accepted (modeling, discretization, truncation or round-off errors) → intrinsic to the methods and algorithms used in numerical simulations and data analytics.

Uncertainly quantification, verification, and validation help to quantify them.

We are interested only by unexpected corruptions that stay undetected by hardware, software, or the users.

This problem of silent data corruption is not limited to scientific computing. It is also a main concern in data bases



Corruption classification

- A **harmful corruption** is manifested as a silent alteration of one of more data elements.

- **Nonsystematic corruptions** affect data in a unique way; that is, the probability of repetition of the exact same corruption in another execution is very low.
- Origins: radiations (cosmic ray, alpha particles from package decay), bugs in some paths of nondeterministic executions, attacks targeting executions individually and other potential sources.

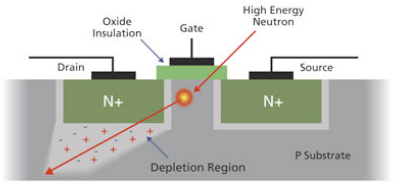
- **Systematic corruptions** affect data the same way at each execution. Executions do not need to be identical to produce the same corruptions.
- Origins : (1) bugs or defects (hardware or software) that are exercised the same way by executions and (2) attacks that will consistently affect executions the same way.

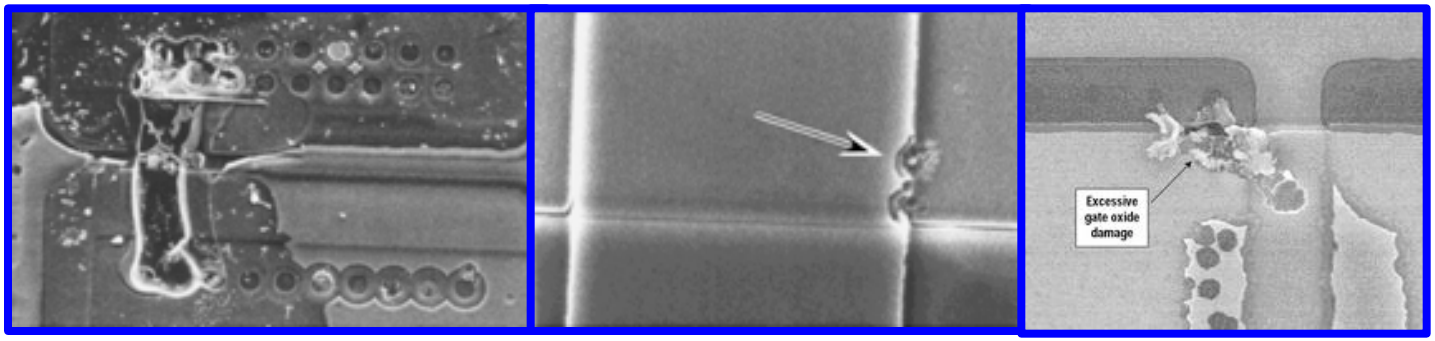
Agenda

- Corruption classification and origins
- **Sources of corruptions (with examples!)**
- Examples of corruption propagation
- Why existing techniques only help partially
- What strategies?
- Example of External Algorithmic Observer
- This is just a beginning

Hardware issues (usually called SDCs)

- Hard error: permanent damage to one or more elements of a device or circuit (e.g., gate oxide rupture, etc.).





metal melt, gate oxide damage

- Soft error (transient errors): An erroneous output signal from a latch or memory cell that can be corrected by performing one or more normal functions of the device containing the latch or memory cell:
  - Cause: Alpha particles from package decay, Cosmic rays creating energetic neutrons
  - Soft errors can occur on transmission lines, in digital logic, processor pipeline, etc.
  - **BW (1.5PB of memory): 1.5M memory errors in 261 days → 1 every 15s**

Bugs (hardware)

1986: Intel 386 processor's bug in the 32-bit multiply routine (fail stop).

1994: Bug of the FDIV instruction of the Pentium P5 processor.

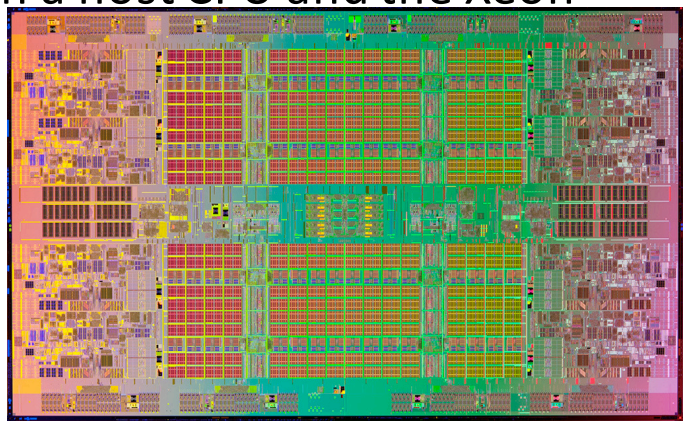
1990: ITT 3C87 chip incorrect computation of arctangent operation.

2002: Itanium processor's bug that could corrupt the data integrity

2004: AMD Opteron's bug that could result in succeeding instructions being skipped or an incorrect address size or data size being used.

2013: Difference in floating-point accuracy between a host CPU and the Xeon Phi used in the TACC Stampede

2014: Opteron's random jump/branch into code.



Detection time and notification time is a major issue:

- It took 6 months for Intel to inform Pentium users about the FDIV bug.
- It took 4 months for HP to communicate the Itanium bug to its customers.

All these examples are documented in the white paper.

Bugs (Numerical libs)

2009: Wrong calculation of Matlab when solving a linear system of equations with the transpose.

2010-2012: Other examples of corruptions (wrong results) have appeared in the Intel MKL library.

2014: cuBLAS DGEMM provided by NVIDIA CUDA 5.5 on Blue Waters' sm_35 Kepler GPUs: case of a silent error where under specific circumstances the results of the cuBLAS DGEMM matrix-matrix multiplication are incorrect but no error is reported.

2014: Issues have been reported for the latest version of MKL on the MIC:

- DSYGVD (eigenvalues) returning incorrect results for a given number of threads.
- DGEQRF (QR fact.) giving wrong results with mkl_set_dynamic(false)

All these examples are documented in the white paper.

Bugs (Compiler-Apps)

Compilers:

2010: Intel Fortran IA-64 compiler optimizer skipped some statements. The bug was difficult to locate and reproduce.

2012: IntelFortran compiler: Several bugs affecting numerical results (in particular, in vectorization and OpenMP): “Loop vectorization causes incorrect results”.

NCAR maintains a list of bugs for CESM. Some of the bugs may lead to corruptions (wrong results, wrong code, call to wrong procedure).

“fortran 95: PGIWith FMA instructions enabled, runs on bluewaters do not give reproducible answers.”

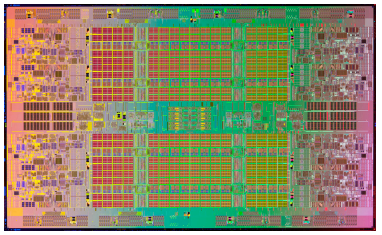
“Fortran 2003 NAG: Functions that return allocatable arrays of type character cause corruption on the stack.”

2014: Bugs in optimization source-to-source compilers (PolyOpt/C 0.2).

Frameworks:

2008: Bug in Nmag micromagnetic simulation package leading to significant corruptions: “Calculation of exchange energy, demag energy, Zeeman energy and total energy had wrong sign.”

Bugs



Hardware

1994: Bug of the FDIV instruction of the Pentium P5 processor.

2014: Opteron's random jump/branch into code.

Libraries

2014: cuBLAS DGEMM (CUDA 5.5) on Kepler GPUs: silent error: results of the cuBLAS DGEMM matrix-matrix multiplication are incorrect

2014: Issues have been reported for the latest version of MKL on the MIC: DSYGVD (eigenvalues): incorrect results for a given number of threads.

Compilers

2012: IntelFortran: bugs affecting numerical results (in particular, in OpenMP vectorization and): "Loop vectorization causes incorrect results".

Frameworks

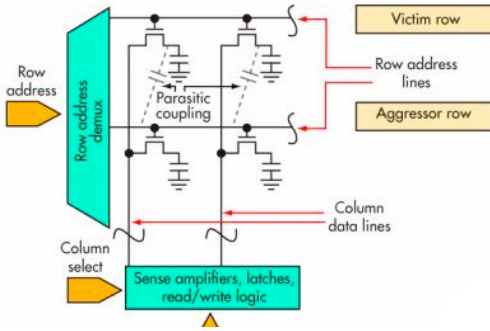
2008: Bug in Nmag micromagnetic simulation package leading to: "Calculation of energy (exchange , demag, Zeeman, total) energies, had wrong result"

Many more examples are documented in the white paper.

Attacks (example)

2014 (ISCA) a group of Carnegy Mellon and Intel show how to flip bits without accessing the victim DRAM row.

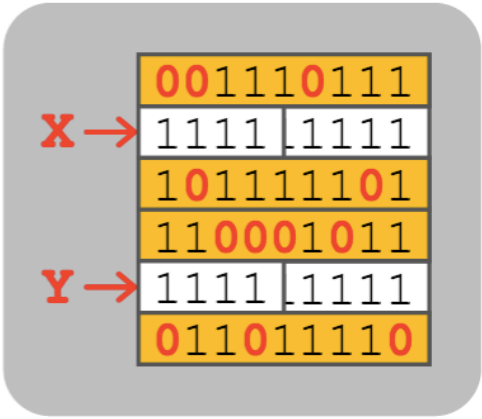
Observation: Toggling a row accelerates charge leakage in adjacent rows, because of row-to-row coupling



Technique:

- DRAM is refreshed every 64ms
- Accelerate charge leakage by writing on the same data at high frequency
- Flush caches to hit DRAM
- Victim rows will be corrupted before the next refresh

```
loop:
  mov (X), %eax
  mov (Y), %ebx
  clflush (X)
  clflush (Y)
  mfence
  jmp loop
```



All modules manufactured in the past two years (2012 and 2013) were vulnerable

As many as 4 errors per cache line: simple ECC (SECDED) cannot prevent all errors

2015 Googleprojectzero published a Linux attack based on row hammer

http://googleprojectzero.blogspot.com/2015/03/exploiting-dram-rowhammer-bug-to-gain.html

Lessons

- Hardware issues (defect, radiation induced bit flips) happen (usual SDCs)
- Bugs are reported everywhere in the stack from the hardware to the users
- Software upgrades often introduce new functionalities that bring new sets of bugs and potential corruptions.
- Attacks exploiting technology weaknesses

- We run simulations and data analytics over very complicated, evolving and fragile stacks

- What can we do to improve the trust in scientific computing results?

Agenda

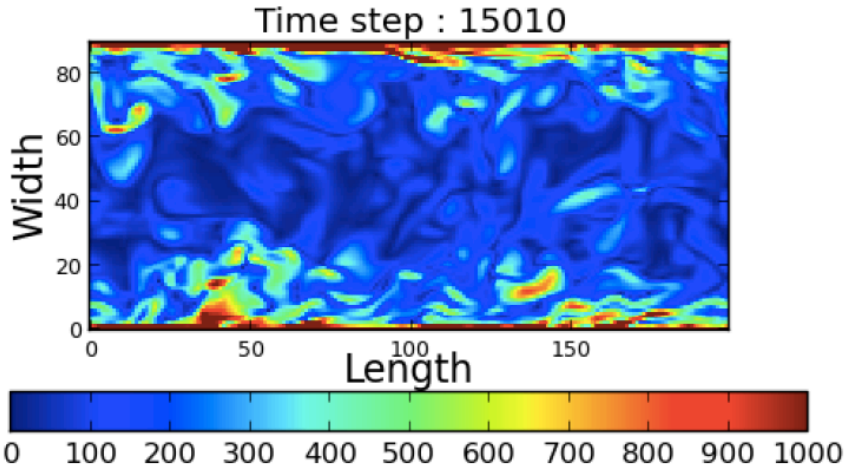
- Corruption classification and origins
- Sources of corruptions (with examples!)
- **Examples of corruption propagation**
- Why existing techniques only help partially
- What strategies?
- Example of External Algorithmic Observer
- This is just a beginning

Example of corruption propagation

- A turbulent flow in a 3D duct modeled as a large eddy simulation… using Navier-Stokes equations.
- Error Injection (1 bit of the exponent) at position 40X40 in the velocity field.

Vorticity (computed from velocity field) of the fluid on a 2D cut of the 3D duct

Difference between the clean and corrupted simulation



Corruption propagation on a CFD code
Bit flip injection in bit position: 24

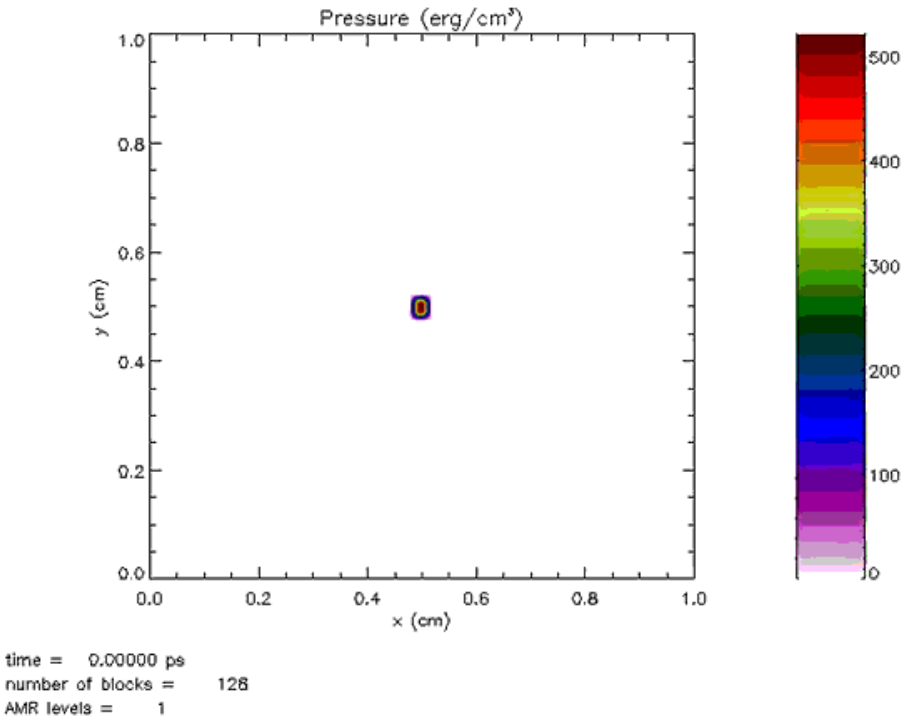
Leonardo Bautista-Gomez
More info: leobago.com

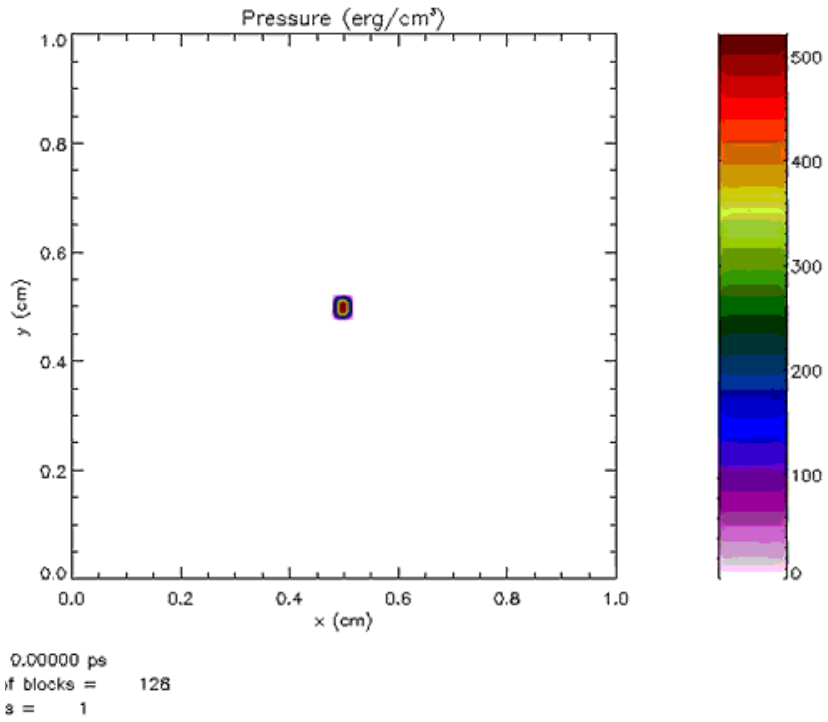
Argonne NATIONAL LABORATORY

- Injections in the lower bit of the representation (mantissa) also have a significant influence (up to bit 18)

Another one

- Hydrodynamical test code involving strong shock and non planar symmetry using the FLASH code.
- Value range on the entire data set: 1200 (diff between largest and smallest)
- Error modifying a value 1 to much lower than 1 (so error << value range)
- Injection at time step 20, in the middle of the bottom edge





Agenda

- Corruption classification and origins
- Sources of corruptions (with examples!)
- Examples of corruption propagation
- **Why existing techniques only help partially**
- What strategies?
- Example of External Algorithmic Observer
- This is just a beginning

Why replication and ABFT helps only partially

Harmful **nonsystematic** corruptions:

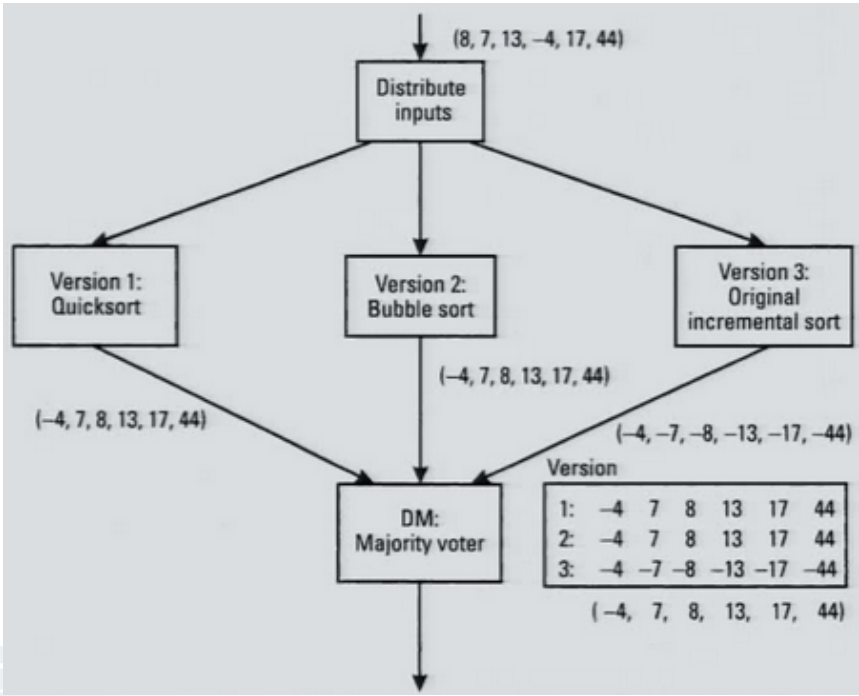
- **Replication** works but is too expensive to be applied on all executions,
- **ABFT** covers only the data protected by the ABFT scheme: other application data are not protected.
- **Ensemble computations:** statistical analysis of the ensemble results may detect or absorb the corruptions. BUT Ensemble can be expensive, and thus not all executions can afford to include ensemble computations.

Harmful **systematic** corruptions:

- **Replication** does not work because it detects corruptions by comparing identical (or comparable) executions.
- **ABFT** may not detect corruptions affecting the ABFT calculation itself. ABFT is also not a solution for attacks because a sophisticated attack could target data sets not protected by ABFT or alter the ABFT calculation itself.
- **Ensemble computations** will suffer the same limitations as replication

Why Multi-version does not help

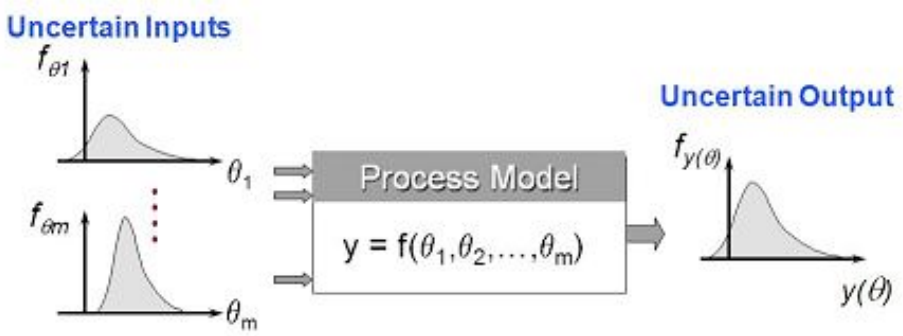
- **N-version programming** was proposed almost three decades ago.
- It was proposed to detect bugs (systematic corruptions).
- Similar to the notion of "alternates" in "recovery blocks",
- Principle: **compare the results of the executions of multiple different code versions responding to the same specification.**
- The higher the diversity of the versions (from hardware to application), the higher is the chance of detecting corruptions.
- This approach **does not seem applicable in our domain** because of the cost of developing multiple versions of all levels of the stacks, from the hardware to the application.
- Moreover, it has been demonstrated experimentally that **different versions may suffer the same bugs** (and lead to the same corruptions).



Why V&V and UQ help only partially

Some background:

- **Validation** compares the output of a simulation with experimental data
- **Verification** checks that the simulation code respects its specification (solution verification, code verification, unit and regression testing,…)
- **UQ** tries to quantity and reduce model uncertainties (the model cannot capture all aspects of a real life phenomenon), using parametric variability and producing output probability distributions.



Limitations:

- Formal **validation and verification** presuppose a correct reference solution. Formal methods are limited to simpler or smaller subsystems than the apps. No solution for complex simulations performed for DOE.
- **UQ** considers hardware and software stack produce correct results. Numerical errors or incorrect software can lead to biases in UQ.

Agenda

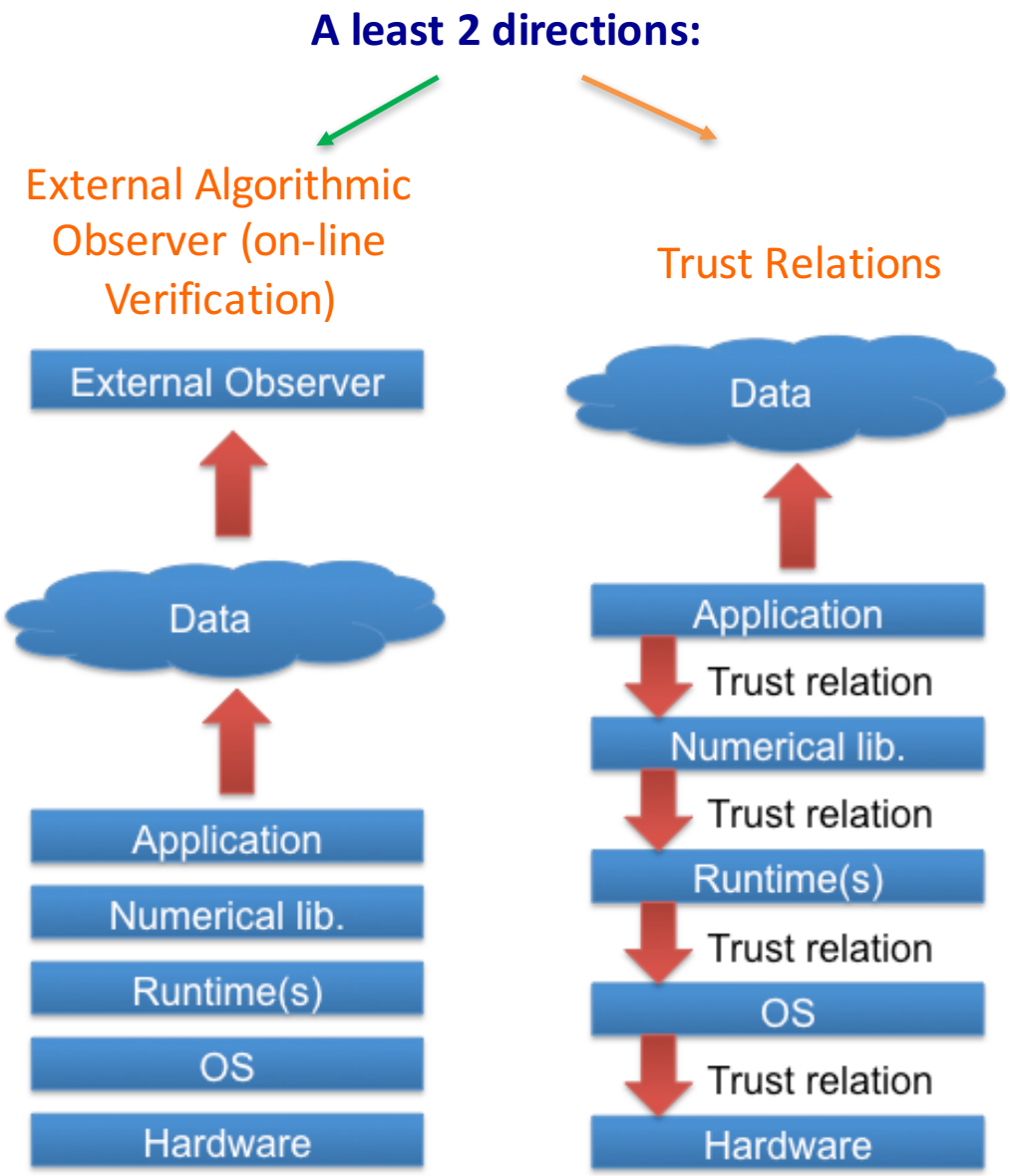
- Corruption classification and origins
- Sources of corruptions (with examples!)
- Examples of corruption propagation
- Why existing techniques only help partially
- **What strategies?**
- Example of External Algorithmic Observer
- This is just a beginning

2 complementary directions

The Trust problem:

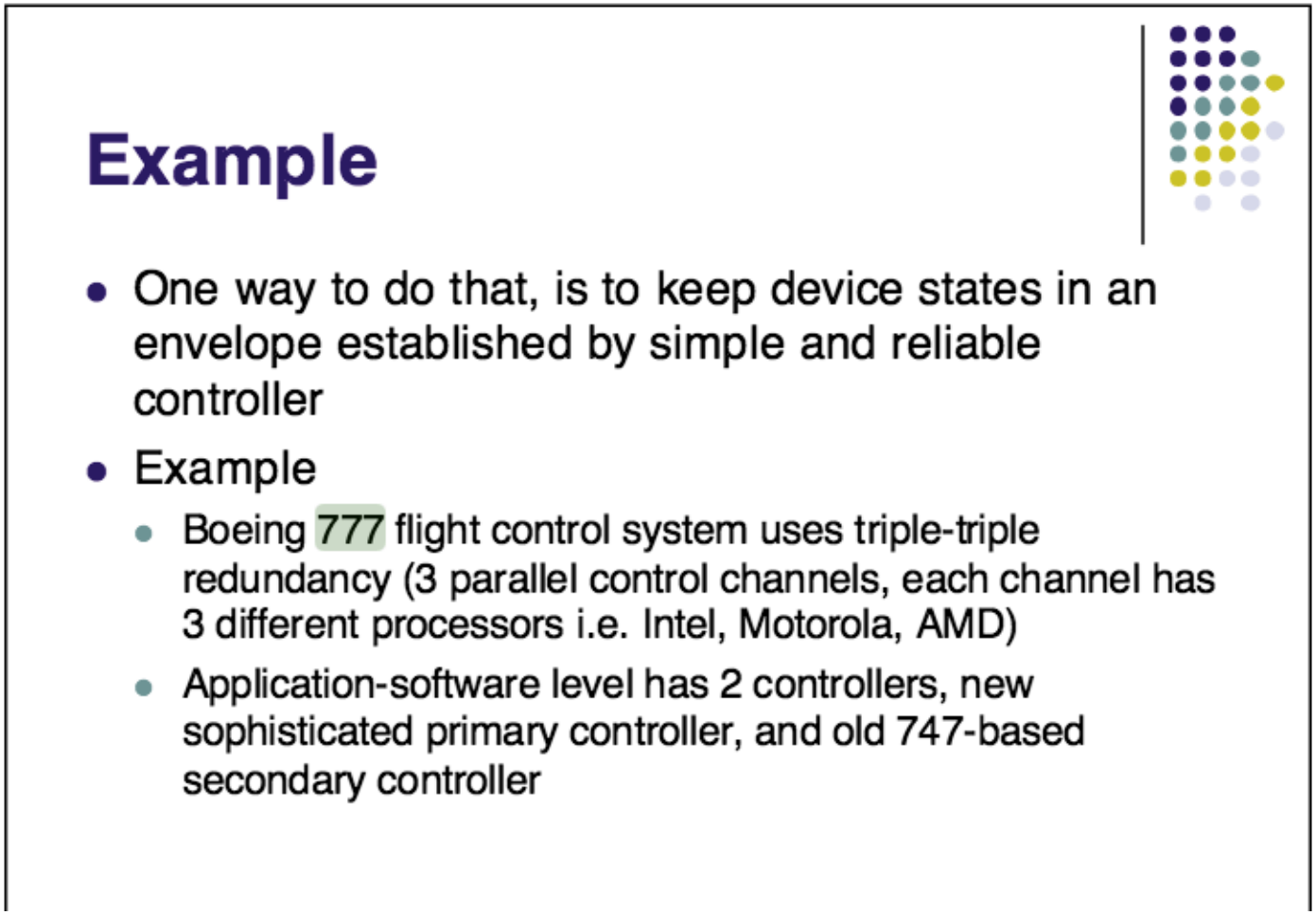
- spans over all layers between hardware and users.
- Is related to many aspects of numerical simulation and data analytics (modeling, initial conditions, numerical accuracy, parametric settings, etc.).

Only a holistic approach has a chance of succeeding.



External Algorithmic Observer Concept

Main idea follows Lui Sha's proposal for "Simplex" architecture for critical systems

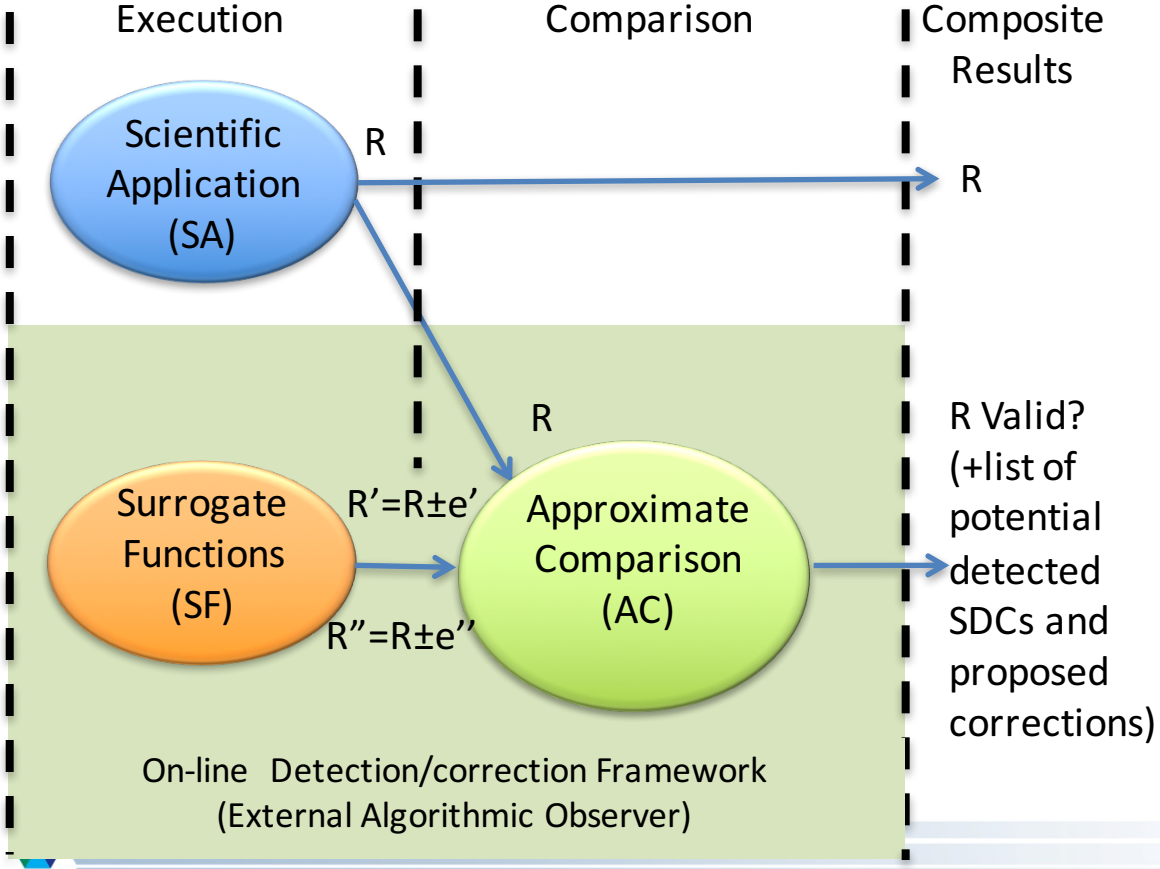


Lui Sha (UIUC), Using Simplicity to Control Complexity -– EEE Software, Jan 2001

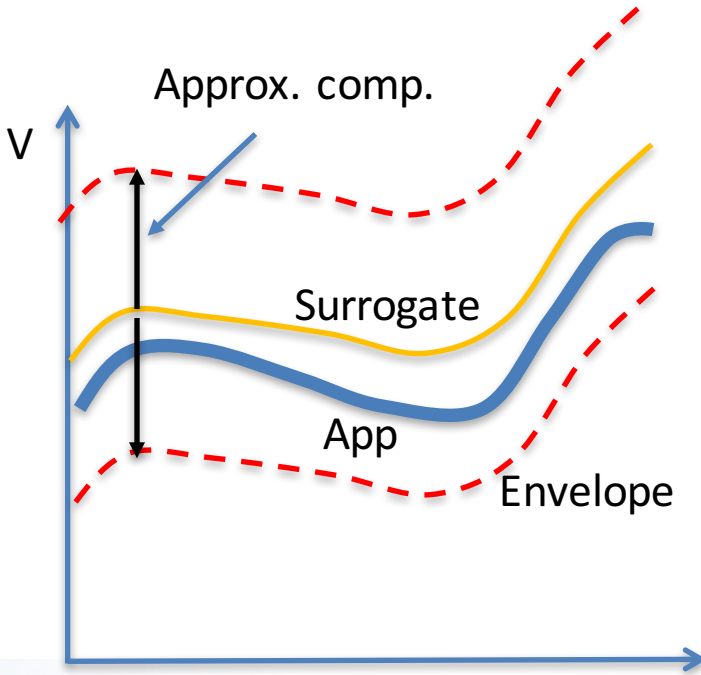
External Algorithmic Observer Principles

External Algorithmic Observer for scientific applications:

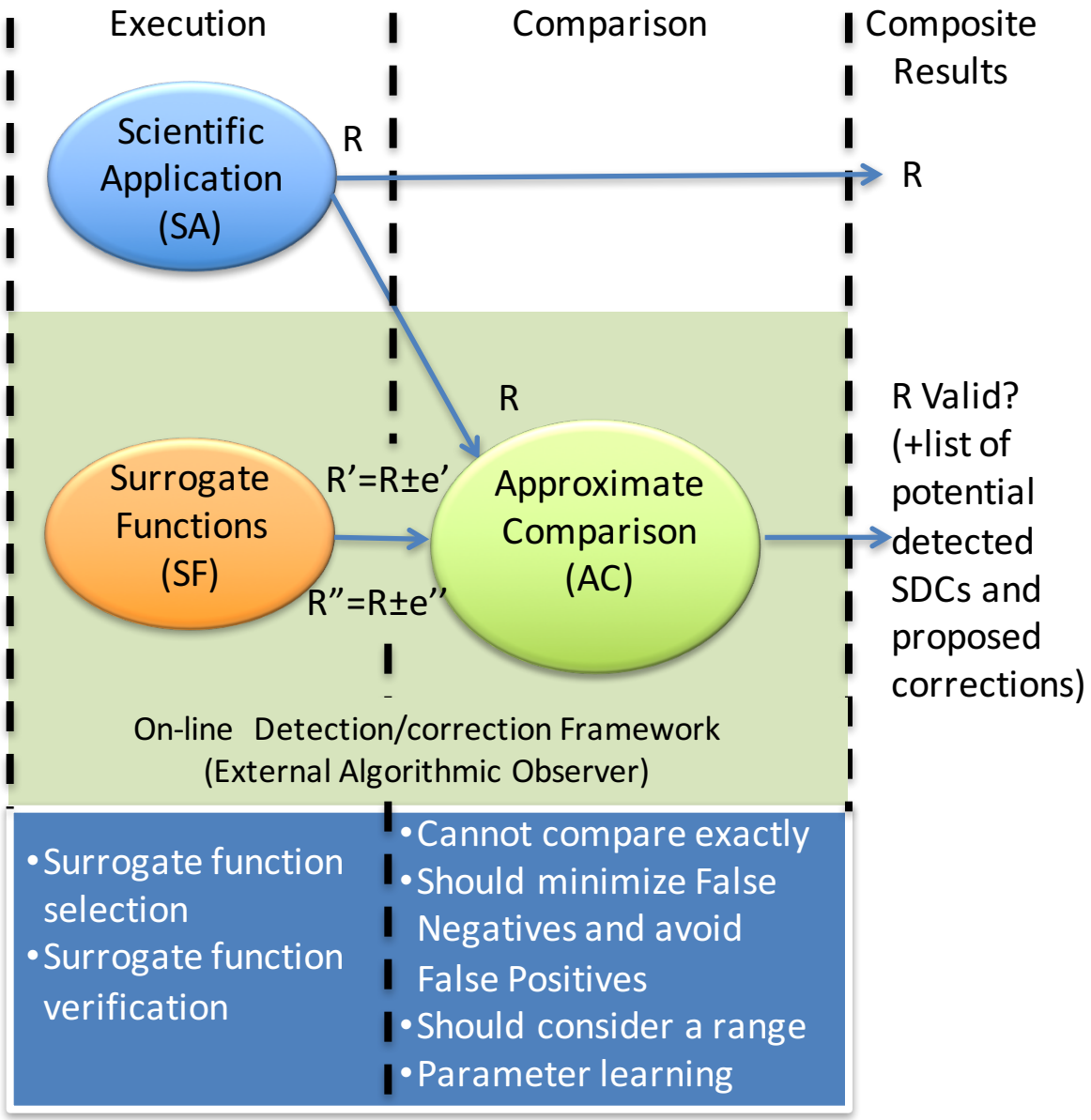
- **Executes a surrogate function that models** the data transformation performed by the application.
- **Approximately compares** the result of the application and the surrogate function



Spatial or temporal trajectory of a simulation variable



External Algorithmic Observer Research



External Algorithmic Observer When/where to check data?

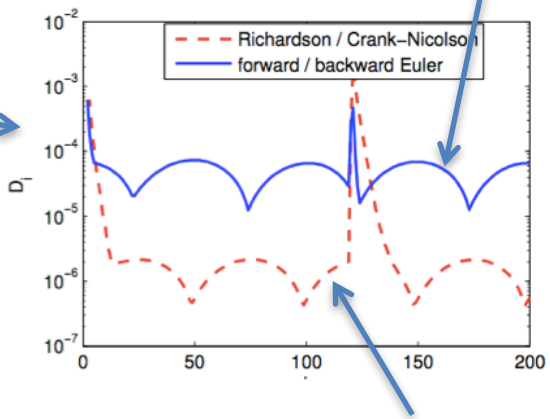
Example of interface for time stepping / iterative computations:

```
MPI_Init(&argc, &argv);
SDC_Init(cfgfile, MPI_COMM_WORLD);
/*Register state variables.*/
SDC_Protect("data", data, SDC_DOUBLE, size);
Loop:
   {Main Application Loop Body}
    isDetected=SDC_Snapshot();
   if (isDetected) take action for data recovery.
end
SDC_Finalize();
MPI_Finalize();
```

External Algorithmic Observer Results

- Very few published research results (3 known groups)

- Two models so far (all for time stepping simulations):
  - **Auxiliary numerical method**:
    - Benson, Schmit, Schreiber 2014
    - Guhur, Zhang, Peterka, Constantinescu, Cappello

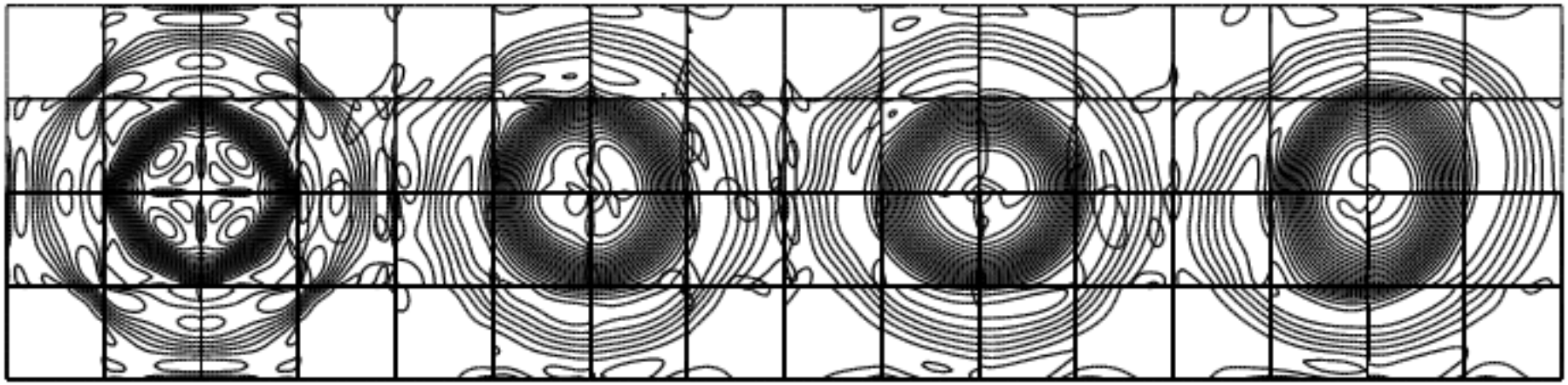


  - **Prediction based method**:
    - Gomez, Di, Berrocal, Cappello, 2014, 2015, 2016
    - Sharma, Bronevetski, Gopalakrishnan, 2015
    - Di, Cappello 2016
    - Subasi, Di, Gomez, Balaprakash, Unsal, Cristal, Labarta, Cappello 2016

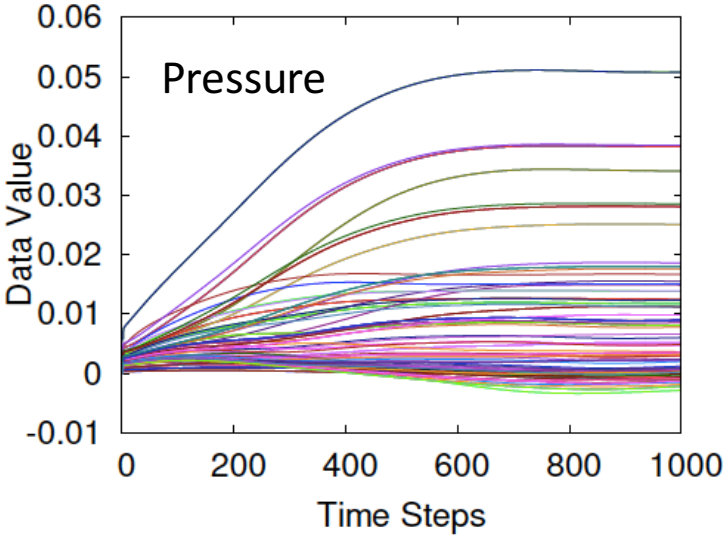
A simple example of External Algorithmic Observer (Simulation)

For time stepping schemes and rather smooth data evolution

- Example: Nek5000 framework, **Vortex** test problem of finite volume methods



Contours of vorticity at t=0,1,2,3
(Hundreds of time steps)
Vorticity: local rotation of the fluid



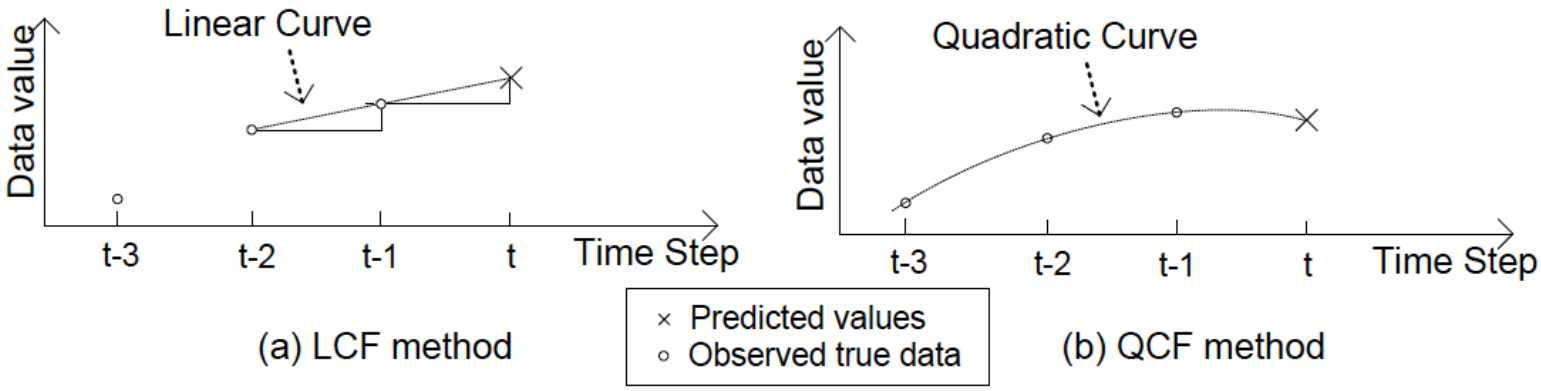
Evolution of 100 Pressure data points
Over 1000 time steps

A simple example of External Algorithmic Observer (Model)

Model and Surrogate function:

Principle: leverage the smoothness of data variation in time to detect corruptions

→ Simple prediction from curve fitting using previous data values:



→Prediction error (Vortex): -Linear Curve fitting: 4×10^{-6}

-Quadratic CF: 2.8×10^{-8}

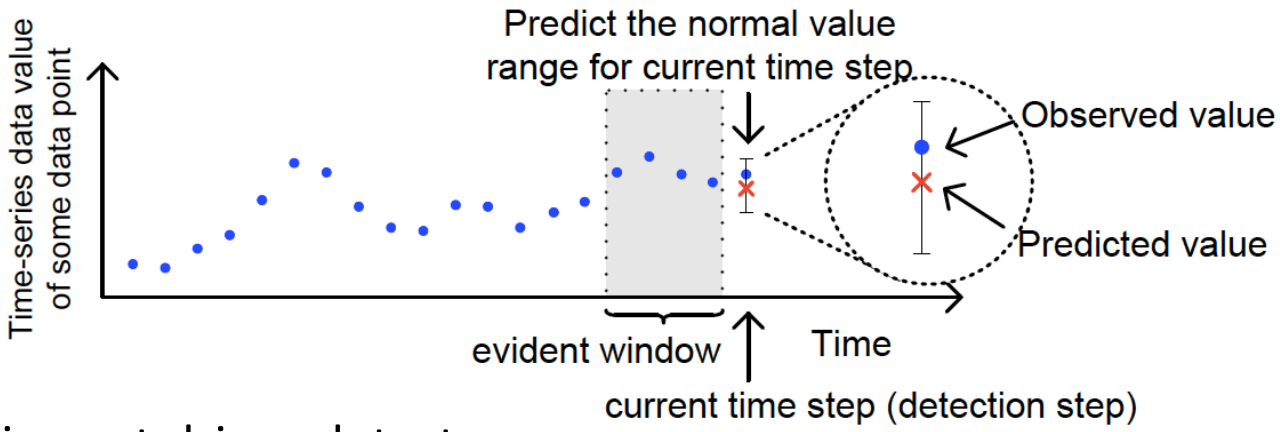
A simple example of External Algorithmic Observer (Comparison)

Approximate comparison:

→ Since exact prediction is not possible, predict a value and a "range"

→ Several ways to define the range.

→ This is itself a prediction (static, based on prediction errors, etc.)



→ Adaptive impact-driven detector

→ Adaptive: Select the curve fitting function based on previous prediction errors.

→ Impact driven:

→Study the impact of SDCs on final results

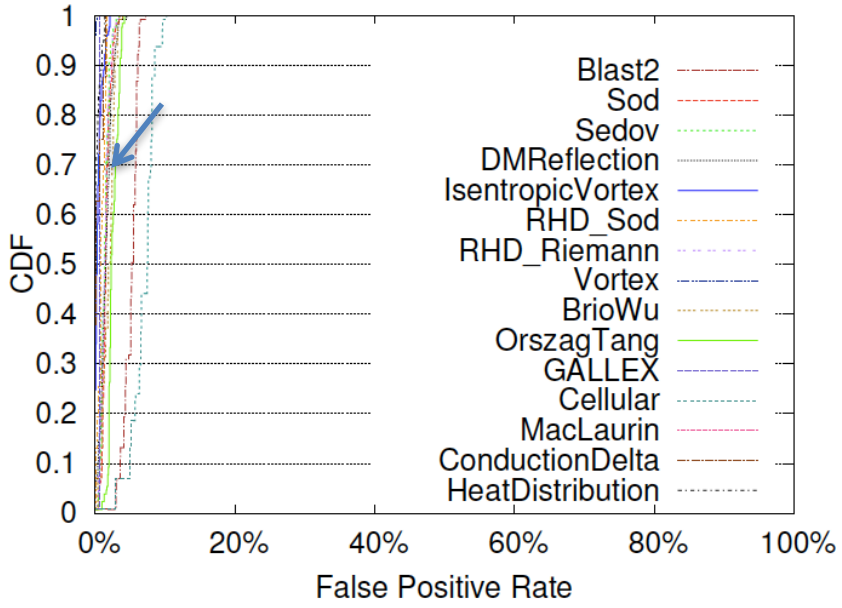
→Establish the value range to avoid impactful SDCs

A simple example of External Algorithmic Observer (Performance)

Injection: single bit corruptions in all possible bit locations (that impact results)
(So no multi-bits corruptions)

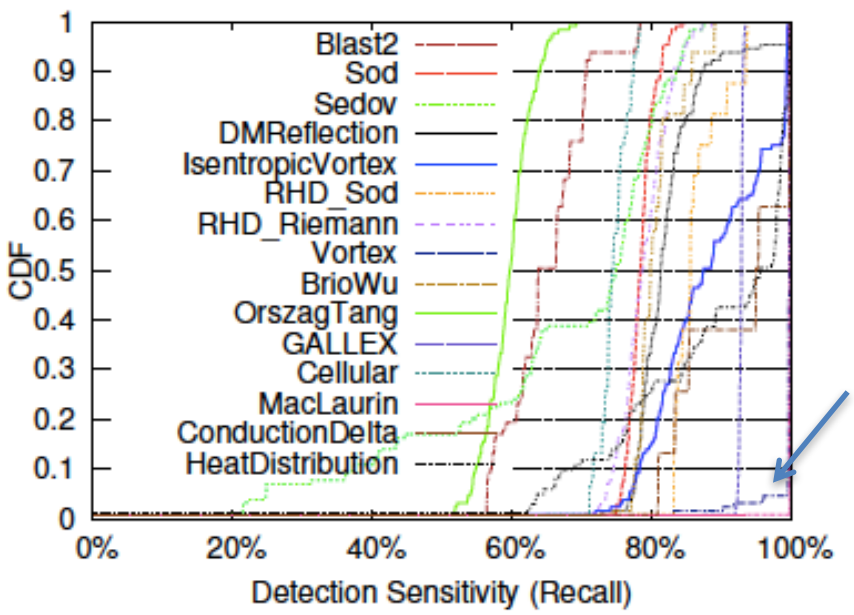
Vortex: False positive rate: ~1%

(1% of iterations trigger unnecessary checking)



CDF computed over all time steps

Vortex: True positive rate: 90%



CDF computed over data points and all time steps

Encouraging results for large set of apps.

This simple detector is called AID (Adaptive Impact Driven)
It has been tested over 24 benchmarks from Nek5000, Flash and other codes
It is our reference detector so far for data set with smooth trajectories
It outperforms all other tentative detectors in terms of accuracy and overhead

For more details: Sheng Di, Franck Cappello, "Adaptive-Impact Driven Detection of Silent Data Corruption for HPC Applications, " to appear in IEEE Transactions on Parallel and Distributed Computing (IEEE TPDS), 2016.

Application	Memory Cost Ratio			Execution Overhead Ratio		
RHD_Sod	1.6%	3.9%	4.1%	3.11%	3.4%	3.4%
RHD_Riemann	1.29%	1.27%	3.9%	4.35%	3.6%	3.7%
Eddy	25.3%	10.6%	23.9%	12%	8.6%	5%
Vortex	44%	52%	48%	1.2%	0.5%	0.4%
BrioWu	9.4%	10.1%	1.9%	2.5%	3.74%	2.85%
OrszagTang	1.12%	1.98%	1.77%	2.46%	5.1%	3%
BlastBS	2.83%	0.87%	0.71%	3.1%	3%	2.98%
Cellular	2.16%	1.66%	3.38%	3.02%	3.44%	3.38%
DustCollapse	13.7%	14.6%	14.1%	1.4%	1.13%	1.12%
MacLaurin	12.8%	13%	9.15%	1.55%	1%	1.3%
Condu.Delta	1.17%	3.84%	1.8%	3.1%	3.6%	2.97%
HeatDist.	0.9%	0.72%	0.85%	13.5%	8.5%	6.3%

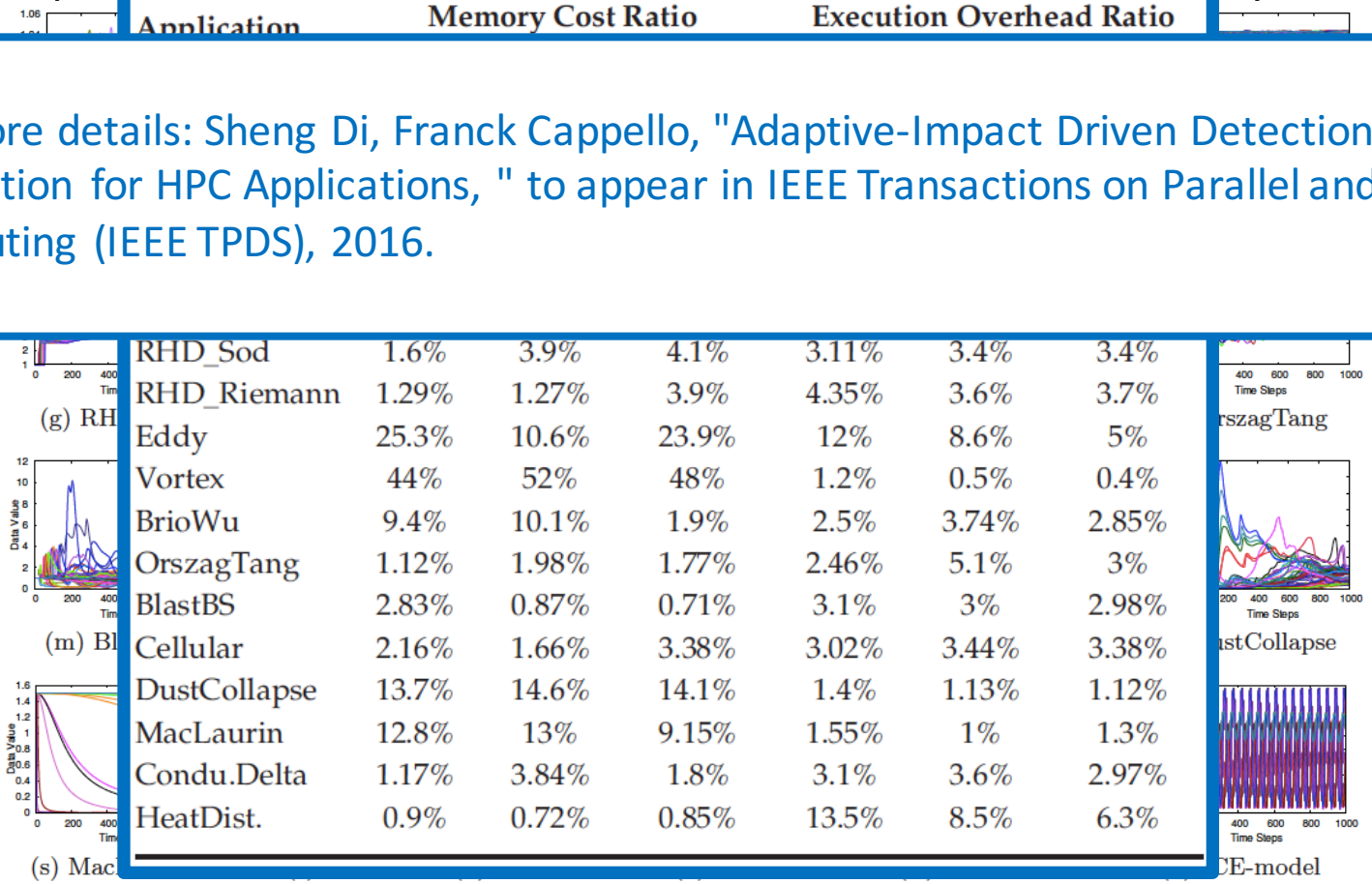
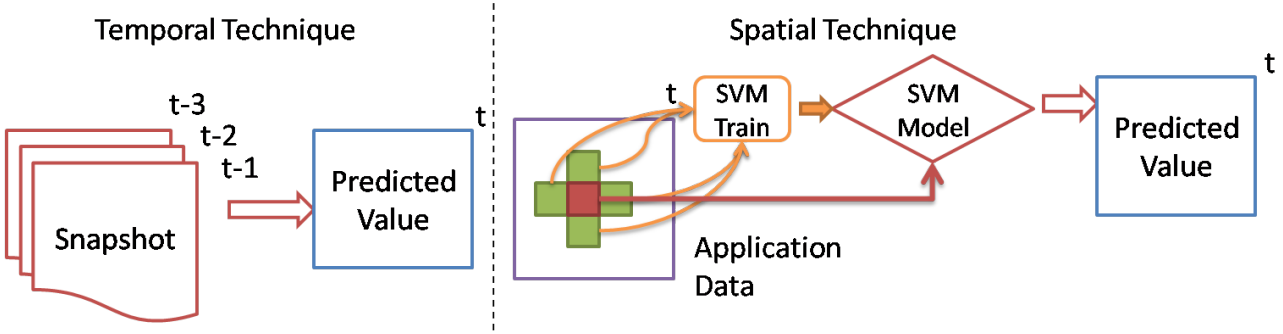


Figure 1: Sampled Time Series Data of The 24 HPC Applications

Another prediction method leveraging spatial property

- SSD: spatial support-vector-machine detector
- Surrogate: predict the value of the variable based on the value of its neighbor using Support Vector Machine



- Experimented with different size of learning sets: 1, 2, 4 neighboring points.
- Experimented four different kernels: linear, polynomial with degree 2, radial basis, and sigmoid functions.
- SSD has a lower memory overhead but also a lower true positive rate
- Details were presented at IEEE CCGRID 16

→ Both are good enough when data evolution is smooth enough

Another direction closer to the numerical method

- Ordinary differential equation

$$x'(t) = f(t, x(t)), \quad x(t_0) = x_0,$$

- Runge Kutta method

$$\forall i \leq s, k_i = f\left(t_n + c_i h, x_n + h\sum_{j=1}^{i-1} a_{ij} k_i\right)$$

$$x_{n+1} = x_n + h\sum_{j=1}^{s} b_i k_i.$$

- Local Truncation Error (LTE)

$$LTE_{n+1} = x_{n+1} - \tilde{x}(t_{n+1}, x_n)$$

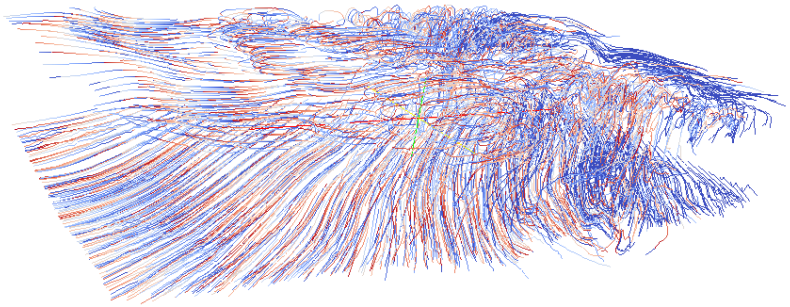
- Surrogate function: A second estimate of the LTE

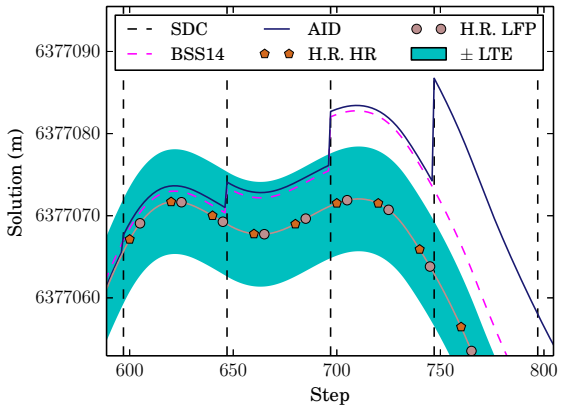
$$\begin{aligned}\Delta_n &= \left(LTE_n + O(h^{p+2})\right) - \left(LTE_n + O(h^{p+2})\right), \\ &= O(h^{p+2}).\end{aligned}$$

- The 2 LTEs are chosen in a way that they are not correlated in case of SDC

Another direction closer to the numerical method

1408 streamlines are computed from a velocity field measured in WRF.





		Name	Replic.	AID	BSS14	HR LFP	HR HR
True positive Rate	TPR	Singlebit (%)	100.0	14.3	18.8	23.1	28.6
		Multibit (%)	100.0	43.2	49.5	64.6	69.6
		Signif. (%)	100.0	86.7	91.2	99.9	99.9
False Positive Rate	FPR (%)		100.0	1.6	0.6	0.01	1.2
Injected relative errors	IRE 95%		0.0	$7e^{-6}$	$4e^{-6}$	$7e^{-8}$	$5e^{-9}$
Overhead	Over.	Comput. (%)	+100	+4.6	+3.7	+3.8	+4.4
		Memory (%)	+100	+62	+67	+105	+105

Coverage of the Algorithmic Observer

Why does it cover (partially) non-systematic corruptions:

- Very unlikely to have the same non-systematic corruptions twice in the simulation and in the model

Why does it cover (partially) systematic corruptions:

- The simulation and model do not perform the same computations
- However data is very close.
- There is a very low probability that a same operation (FPADD, FPMUL, etc.) is executed with close data in both the simulation and model.
  - → If the operation is bogus, similar corruptions may happen.
  - → Recommendation: execute the model in a different hardware (CPU+GPU)

→ More study is needed on the coverage

Our experience with Algorithmic Obs.

Different surrogate functions (run by the observer):

- Temporal property of the data transformation
- Spatial property of the data transformation
- Exploiting estimates of local truncation error

The approximate comparison:

- Fixed value range
- Adaptive value range using the prediction errors
- Point based adaptive value range based on error impact

Corruption detection Metrics:

- Not so straightforward methodology (lack of common practices):
  - What corruptions to inject: (single bits, multi-bits, multi-data, systematic)?
  - How to inject: where (in registers, memory), when (iteration, library)?
  - What temporal/spatial distributions for the corruption injection?
  - How to report true positive rate (if multi-bit corruptions)?

3 Remarks on Algorithmic Observer Features

1) The surrogate function **cannot replace the application**. Its predictions are valid only from one step to the next one.

2) Low-complexity models implement **trade-offs between complexity, accuracy, and other properties**:

-Benson, Schmit, Schreiber relaxes numerical stability assuming:

- model should be restarted at each step from application verified results
- corruptions happening in one step are detected in the same step

-Prediction models compute only local predictions from the application results at the current step (one step prediction). Also assumes:

- corruptions happening in one step are detected in the same step

3) Important advantage: **model is easier to verify and to protect than the** application. → Amenable to formal verification, multi-version programming and execution on a secure processor (FPGA for example).

Trust Relations

More mature: a large body of research in computer science

DOE report on Cybersecurity for Scientific Computing Integrity [1] covers issues and approaches.

Let's call **"object"** any software of hardware that needs to be trusted.

→ Trust relation supposes at least:

- a way to **certify** that **each used object** is actually the object it is supposed to be,
- a method to **evaluate a level of trust** for each object involved in the execution (reputation for example)
- a **metric of the level of trust**, and
- a way to **protect the trust level acquired** by an object

Trust Relations (nothing new here)

Certification and protection of trust level:

→ **Trust Computing Group** produced Trusted Platform Module (TPM) specification

→ Specifies Embedded crypto capability for user, apps., machine authentication

- More than 500 million PCs have shipped with TPM.
- Vulnerable to sophisticated attacks + TPM circuits showed vulnerability

Trust evaluation:

→ Trust level could rely on **verification and validation** of that object by a combination of formal verification when applicable and empirical methods.

→ In principle, **external observer approach** can be applied for each object.

Trust Metrics:

→ **Not a new problem** in security and networking domains (solutions)

→ **Metrics with multiple dimensions**: time since first trusted, time since last verification, number of independent verifications, number of validations, etc.

All these precautions will not avoid corruptions from a highly trusted object

http://www.trustedcomputinggroup.org/

Comparing the 2 approaches

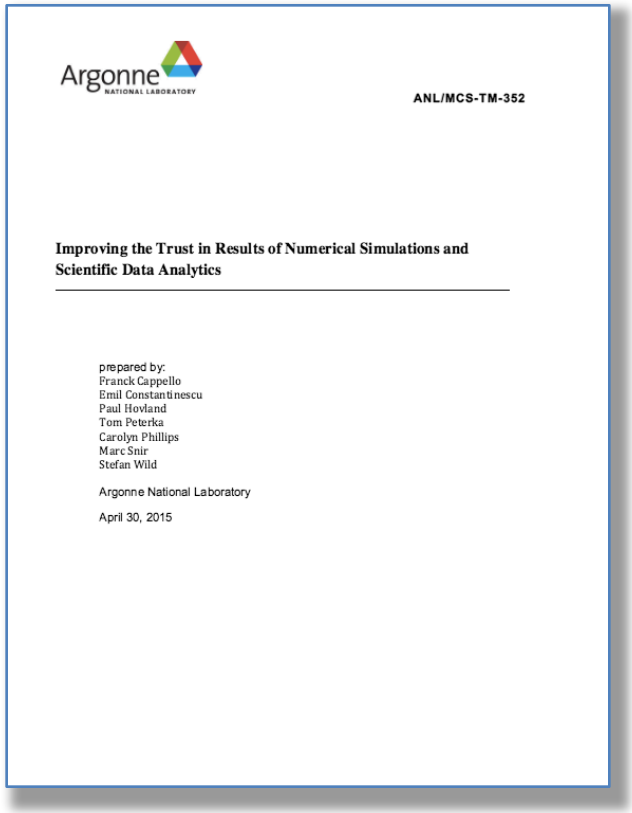
	External Observer	Trust Relations
Detection Approach	Simulation and observer are checking each other	Checking object results
Detection Assumptions	**External observer is correct** (should be verified, validated)	All **verifications and reputation calculations are correct**
Detection Latency	**Short** (depends on sampling rate, typically 1 application iteration)	**Long** (actual detection could be long: months)
Timeliness of Notification after Detection	**Short** (from one iteration to the next)	**Short** (immediate upper layer)
Time to build trust	**Low** (trust depends on verisimilitude of results not on components)	**High** (hard and soft components need to acquire trust level)
Targeted Level of Trust	**User-expected accuracy**	**Machine precision** (modulo round-off errors)
Development Time and Cost	**Low** (requires only to develop the observer)	**High** (affects all layers of the stack)
Tolerance	**High** (corruptions of the application data lower than user-expected accuracy are tolerated)	**Low** (any corruption at object level is suspicious since the consequence on application data is unknown)

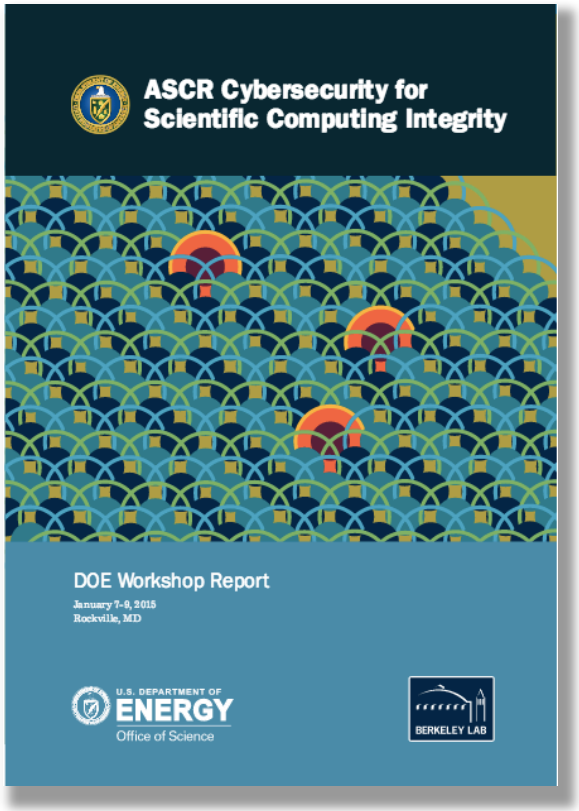
Conclusion

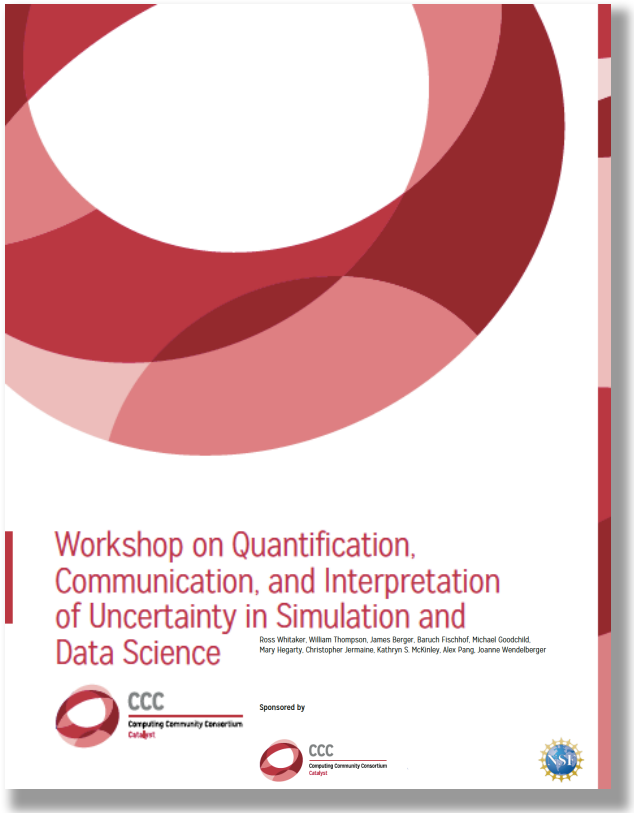
- Trust in results of numerical simulation and data analytics is serious and insufficiently recognized problem in our community
- Trust is a harder problem than FT and resilience because it relates also to bugs and attacks
- Lack of research and results in this domain
- Two directions (identified so far, more probably exist):
  - Algorithmic external observer:
    - Model of data transformation
    - Approximate comparison
  - Trust relation (much more mature in other domains)

→ it's a fascinating and pretty open research problem!

Questions?







Improving the Trust in Results of Numerical Simulations and Scientific Data Analytics Cappello, F, Constantinescu, EM, Hovland, PD, Peterka, T, Phillips, CL, Snir, M, Wild, SM, report ANL/MCS-TM-352

Applications

Domain	Name	Code	Description
HD	Blast2 [18]	Flash	Strong shocks and narrow features
	SodShock [19]	Flash	Sodshock tube for testing compressible code's ability with shocks & contact discontinuities
	Sedov [20]	Flash	Hydrodynamical test code involving strong shocks and non-planar symmetry
	DMReflection [18]	Flash	Double Mach reflection: an evolution of an unsteady planar shock on an oblique surface
	IsentropicVortex [21]	Flash	2D isentropic vortex problem: a benchmark of comparing numerical methods for fluid dynamics
	RHD_Sod [22]	Flash	Relativistic Sod Shock-tube: involving the decay of 2-fluids into 3-elementary wave structures
	RHD_Riemann2D [23]	Flash	Relativistic 2D Riemann: exploring interactions of four basic waves consisting of shocks, etc.
	Eddy [24]	Nek5k	2D solution to Navier-Stokes equations with an additional translational velocity
	Vortex [25]	Nek5k	Inviscid Vortex Propagation: tests the problem in earlier studies of finite volume methods [26]
MHD	BrioWu [27]	Flash	Coplanar magneto-hydrodynamic counterpart of hydrodynamic Sod problem
	OrszagTang [28]	Flash	Simple 2D problem that has become a classic test for MHD codes
	BlastBS [29]	Flash	3D version of the MHD spherical blast wave problem
	GALLEX [30]	Nek5k	Simulation of gallium experiment (a radiochemical neutrino detection experiment)
BURN	Cellular [31]	Flash	Burn simulation: cellular nuclear burning problem
GRAV	DustCollapse [32]	Flash	Selfgravitating problems in which the flow geometry is spherical without gas pressure
	MacLaurin [16]	Flash	MacLaurin spheroid (gravitational potential at the surface/inside a spheroid)
DIFF	ConductionDelta [16]	Flash	Delta-function heat conduction problem: examining the effects of Viscosity
	HeatDistribution [33]	customized	Steady-state heat distribution with Laplace's equation by Jacobi iterative method

Lack of Trust metrics in HPC

- Many metrics in e-commerce are related to the notion of **reputation** built from external evaluations.

- BUT reputation does not seem sufficient in our domain:
  - reputation built from the apparent corruption-free usage of a hardware or software artifact does not mean that this artifact has not produced incorrect results in the past and does not inform users about the potential of producing incorrect results in the future.

- In numerical simulation and scientific data analytics, there is a **lack of trust metrics** that could be used to quantitatively compute and express the trustworthiness of the execution results.