SAND2014-20364C

Xyce: an open source SPICE engine

Eric Keiter, Tom Russo, Heidi Thornquist, et al.
Sandia National Laboratories

Jaijeet Roychowdhury, Tianshi Wang
UC-Berkeley

MOS-AK Workshop
Berkeley, CA
Dec. 12, 2014

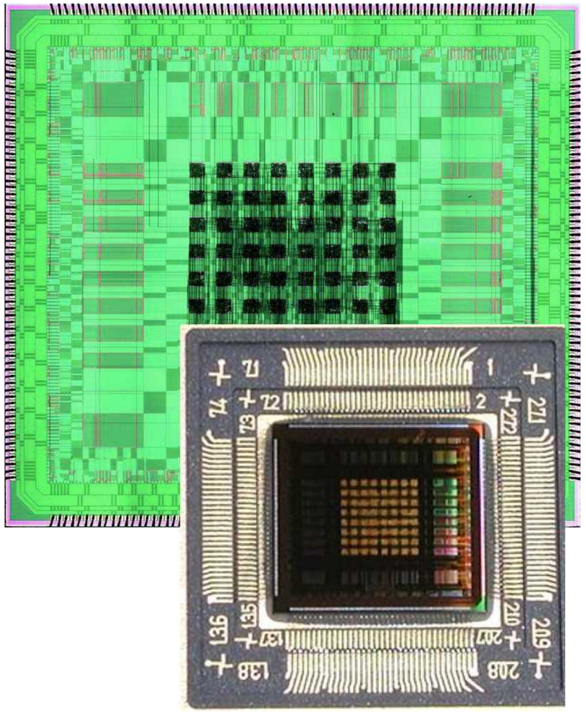
Outline

- Xyce intro
- Xyce requirements
- Xyce design
  - Object-oriented (C++ mostly)
  - Spice-compatible
  - But different than SPICE!
  - Parallel implementation
  - Analysis types (DC,Tran,AC, HB, MPDE, UQ)
  - Open-Source (as of v6.0)

- ADMS model compiler
- ModSpec-Xyce
- Xyce supports most SPICE models (BSIM, EKV, etc)

Xyce Parallel Circuit Simulator

- Xyce: Massively Parallel circuit simulator:
  - Distributed Memory Parallel (MPI-based)
  - Unique solver algorithms
  - SPICE-Compatible
  - Industry standard models (BSIM, PSP, EKV, VBIC, etc)
- Analysis types
  - DC, TRAN, AC
  - Harmonic Balance (HB)
  - Multi-time PDE (MPDE)
  - Model order reduction (MOR)
  - Direct and Adjoint sensitivity analysis
  - Uncertainty quantification (UQ) via Dakota.
- Sandia-specific models
  - Prompt Photocurrent
  - Prompt Neutron
  - Thermal
- Other, non-traditional models
  - Neuron/synapse
  - Reaction network
  - TCAD (PDE-based)
- Xyce Release 6.2
  - Open Source!
  - GPL v3 license



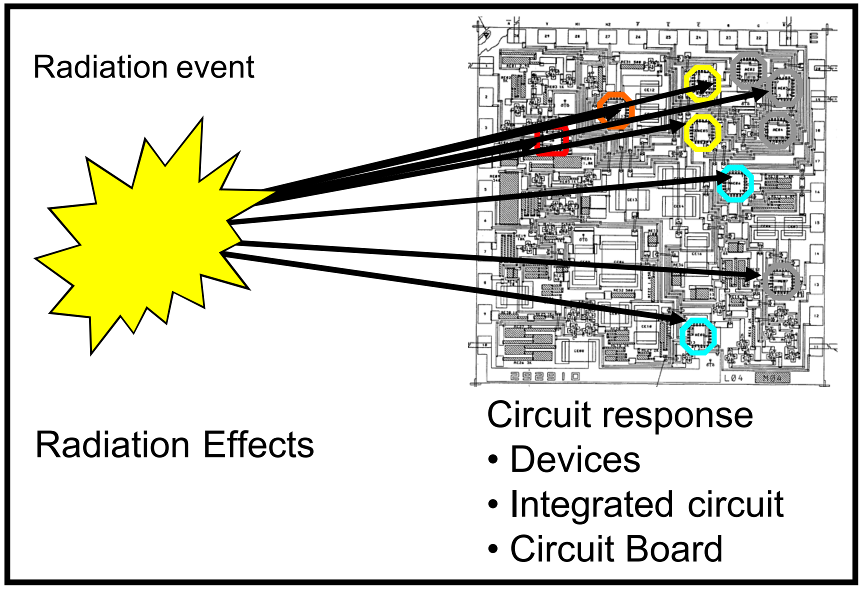
http://xyce.sandia.gov

Next release (v6.1) ~April 2014

Project Motivation: Why Xyce?

- Comprehensive Test Ban Treaty (CTBT), 1993
- Advanced Simulation & Computing (ASC), 1995
- Qualification Alternatives to SPR (QASPR), 2005
  - Offset lack of NW testing
  - Help qualify NW systems

- Unique Requirements ➡ Differentiating capabilities
  - Unique models: Radiation Effects
  - High fidelity: SPICE-level or higher
  - Large capacity: Massively-parallel
  - IP: Sandia owns it

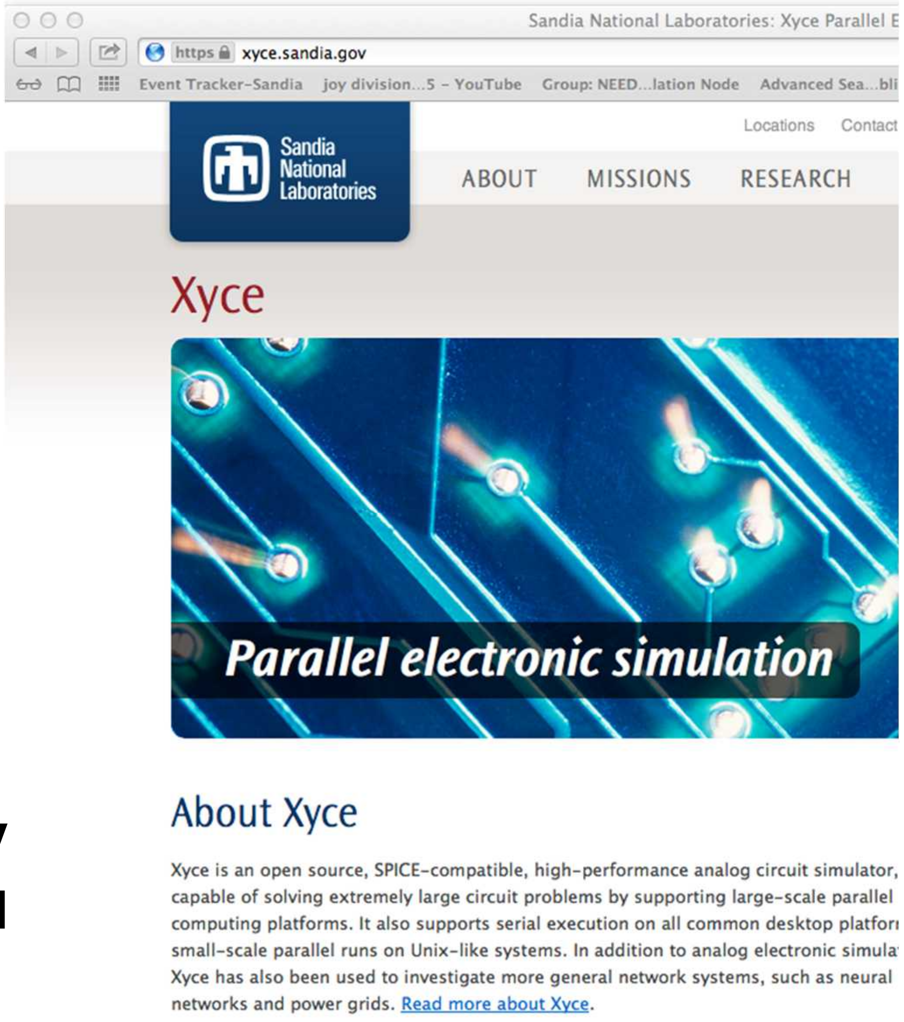


Why Open Source?

- First open source release, v6.0
- November 5, 2013.
- GPL license v3.0
- Source and binary downloads available
- xyce.sandia.gov

- Next release (v6.1) ~April 2014.

- Foster external collaboration
- Feedback from wider community
- Taxpayer funded, so encouraged to open source.


Sandia National Laboratories: Xyce Parallel E
https xyce.sandia.gov
Event Tracker-Sandia
joy division...5 - YouTube
Group: NEED...lation Node
Advanced Sea...bli
Locations
Contact
Sandia National Laboratories
ABOUT
MISSIONS
RESEARCH
Xyce
Parallel electronic simulation

About Xyce

Xyce is an open source, SPICE-compatible, high-performance analog circuit simulator, capable of solving extremely large circuit problems by supporting large-scale parallel computing platforms. It also supports serial execution on all common desktop platfor small-scale parallel runs on Unix-like systems. In addition to analog electronic simula Xyce has also been used to investigate more general network systems, such as neural networks and power grids. Read more about Xyce.

Open Source Releases

Release every 6 months.

Xyce v6.0 (first open source) October, 2013

Xyce v6.1 April 2014

- Parallel HB, MPDE and AC analysis.
- BSIM-CMG model
- New digital models
- Dynamically linkable devices (beta implementation)

Xyce v6.2 October, 2014

- Calculation of transient direct sensitivities.
- Support for lead current calculation and output for Harmonic Balance (HB) analysis.
- Improved interpolation for both the Trapezoid and Gear time integrators.
- Improved results output (.PRINT) capabilities.
- New models for a lumped transmission line and a digital latch.
- Fixes for nearly 60 bugs and enhancement requests.

Xyce v 6.3 ~April/May 2015

Near term new features

Some of these will be in 6.3, some 6.4.

Xyce v6.3 ~April/May 2015
- Multi-tone HB (unlimited # of tones)
- Small-signal noise analysis
- Auto-detection/application of MOR to linear subcircuits
- BSIM6 model
- MEXTRAM model

Xyce v6.4 ~October/November 2016
- Adjoint transient parameter sensitivities
- New direct linear solver (BASKER)
  - serial version
  - threaded version

Hybrid-Hybrid solver result: 19x Speedup for Challenging IC

Necessary for efficient simulation of a CMOS logic circuit:

- 1.6M total devices, ~2M unknowns
- Xyce w/ KLU solver takes ~ **2 weeks**, w/ ShyLU solver takes ~ **1 day**

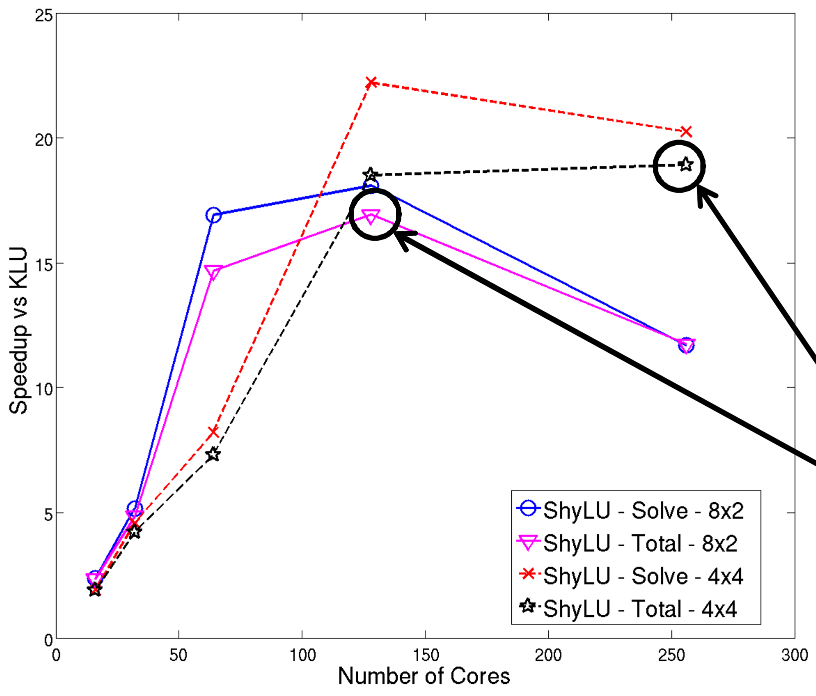


TABLE III
COMPARISON OF TOTAL LINEAR SOLVE TIME (SEC.) OF VARIOUS SPARSE DIRECT SOLVERS FOR OUR TEST CIRCUITS; (-) INDICATES SIMULATION FAILED TO COMPLETE.

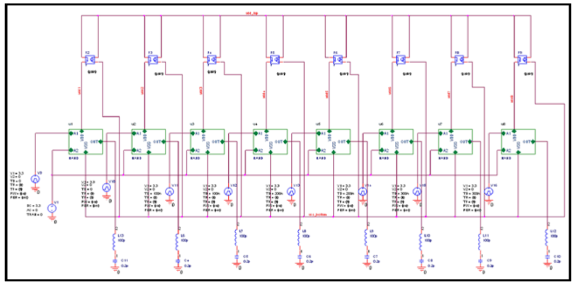
	ckt1	ckt2	ckt3	ckt4	ckt5
KLU	**80.8**	162.2	9381.3	7060.8	**14222.7**
PARDISO (16)	128.6	**105.3**	**715.0**	**6690.5**	-
SuperLU	-	10294.1	-	-	72176.8
SuperLU_Dist (16)	-	-	-	-	-

Strong scaling of Xyce's simulation time and ShyLU linear solve time for different configurations of MPI Tasks X Threads per node.

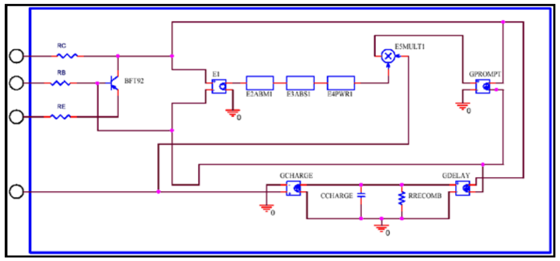
A Hybrid Approach for Parallel Transistor-Level Full-Chip Circuit Simulation, H. K. Thornquist, S. Rajamanickam, VECPAR 2014

Transforming Design Capabilities

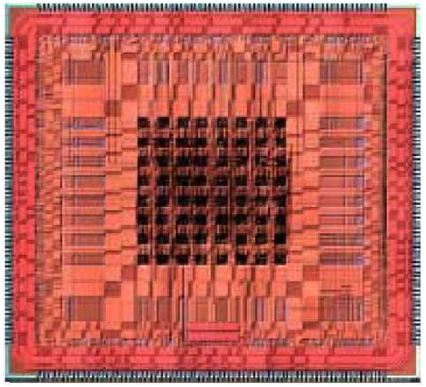
then ~ 2001
PSpice circuit model:
~300 devices



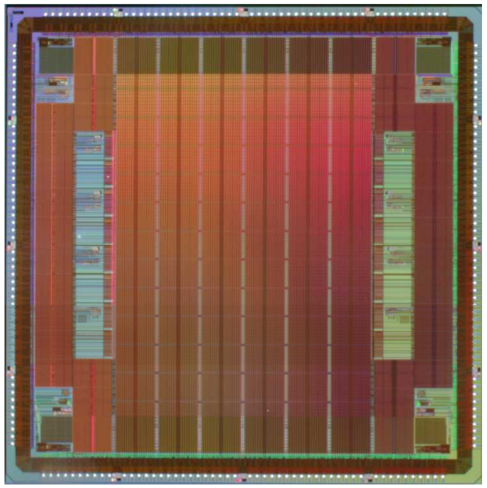
rad. model:
empirical/behavioral



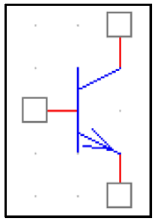
2006
Xyce circuit model:
300K+ devices



201
2
Xyce circuit model:
40M+ devices



rad. model:
physics-based/built-in



Xyce Requirements

Xyce started in 1999. Requirements/goals:

- SPICE-compatible
- Massively parallel
- Include special Sandia-specific physics models
- Useful to both sophisticated and unsophisticated users
- Object-oriented, modular (C++)
- Extensible, lots of developers

Another way to put it:

"Be the same as SPICE, but also be much better"

"I never want to see the "Time step too small" error again" – Ken Marx, Sandia Electrical Analyst

Why not use SPICE?

SPICE: the original open-source simulator

- de-facto standard

To be useful: modular, well-structured, flexible

- separated numerics, algorithms, models, I/O
- simple, clean interfaces
- **short, easy to read, easy to modify**

i.e., **not SPICE!**

excerpt from *dioload.c* (SPICE3)

```
#ifdef SENSDEBUG
  printf("vd = %.7e \n",vd);
#endif /* SENSDEBUG */
  goto next1;
}
if(ckt->CKTmode & MODEINITSMSIG) {
  vd= *(ckt->CKTstate0 + here->DIOvoltage);
} else if (ckt->CKTmode & MODEINITTRAN) {
  vd= *(ckt->CKTstate1 + here->DIOvoltage);
} else if ( (ckt->CKTmode & MODEINITJCT) &&
   (ckt->CKTmode & MODETRANOP)
        && (ckt->CKTmode & MODEUIC) ) {
                vd=here->DIOinitCond;
} else if ( (ckt->CKTmode & MODEINITJCT) && here->DIOoff)
{
        vd=0;
} else if ( ckt->CKTmode & MODEINITJCT) {
        vd=here->DIOtVcrit;
} else if ( ckt->CKTmode & MODEINITFIX && here->DIOoff) {
        vd=0;
} else {
#ifndef PREDICTOR
        if (ckt->CKTmode & MODEINITPRED) {
```

Sensitivity analysis code

AC analysis code

Transient analysis related code

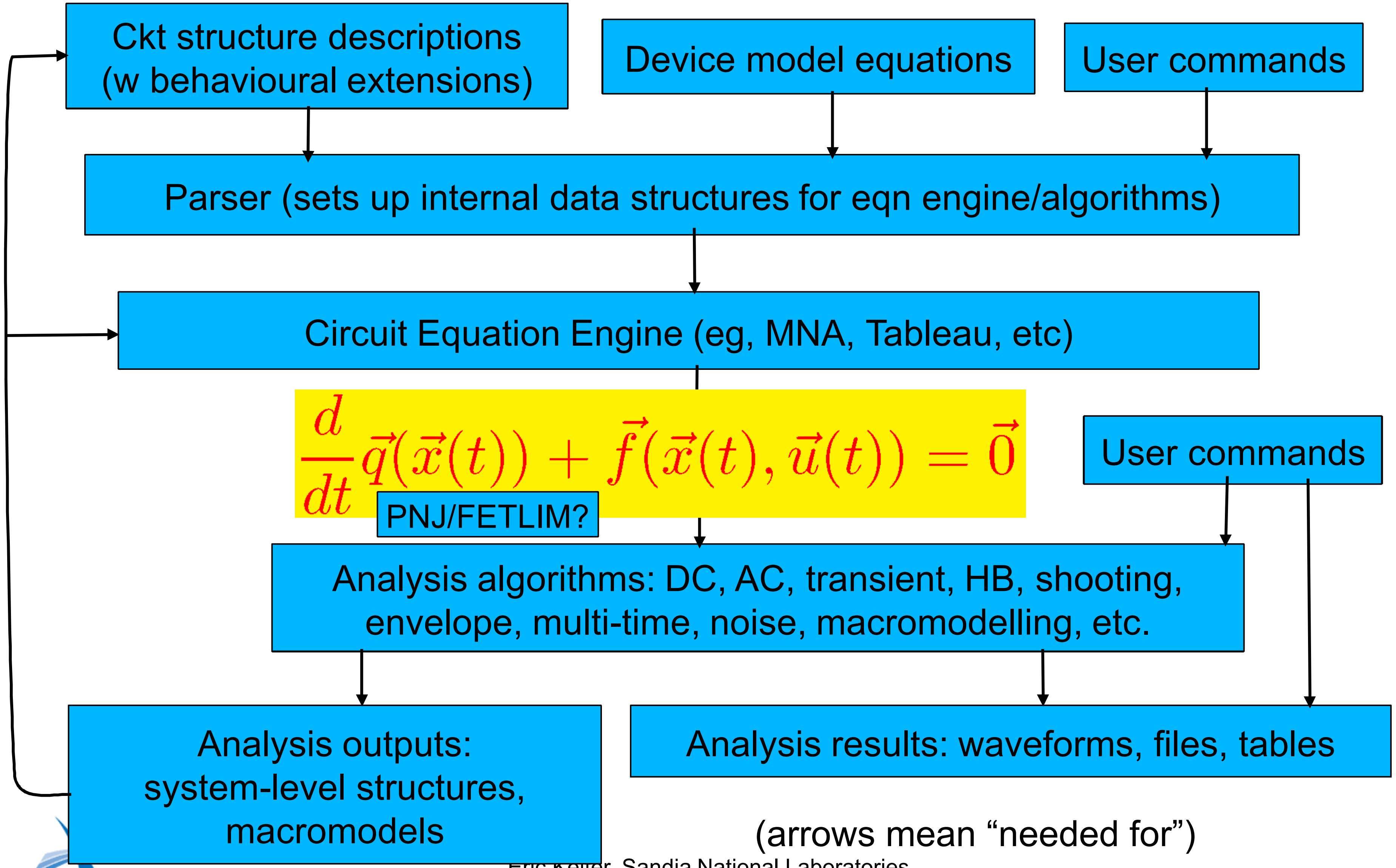
Why Modularity is Necessary

Key to managing complexity

- limits what each developer needs to know about the system
  - the "API" encapsulates rest of the simulator
- Eg, to implement BSIM6.4, you don't need to know all about:
  - trapezoidal, Gear, etc. algorithms
  - shooting, harmonic balance, sensitivity, ...
  - Newton-Raphson, homotopy, ...

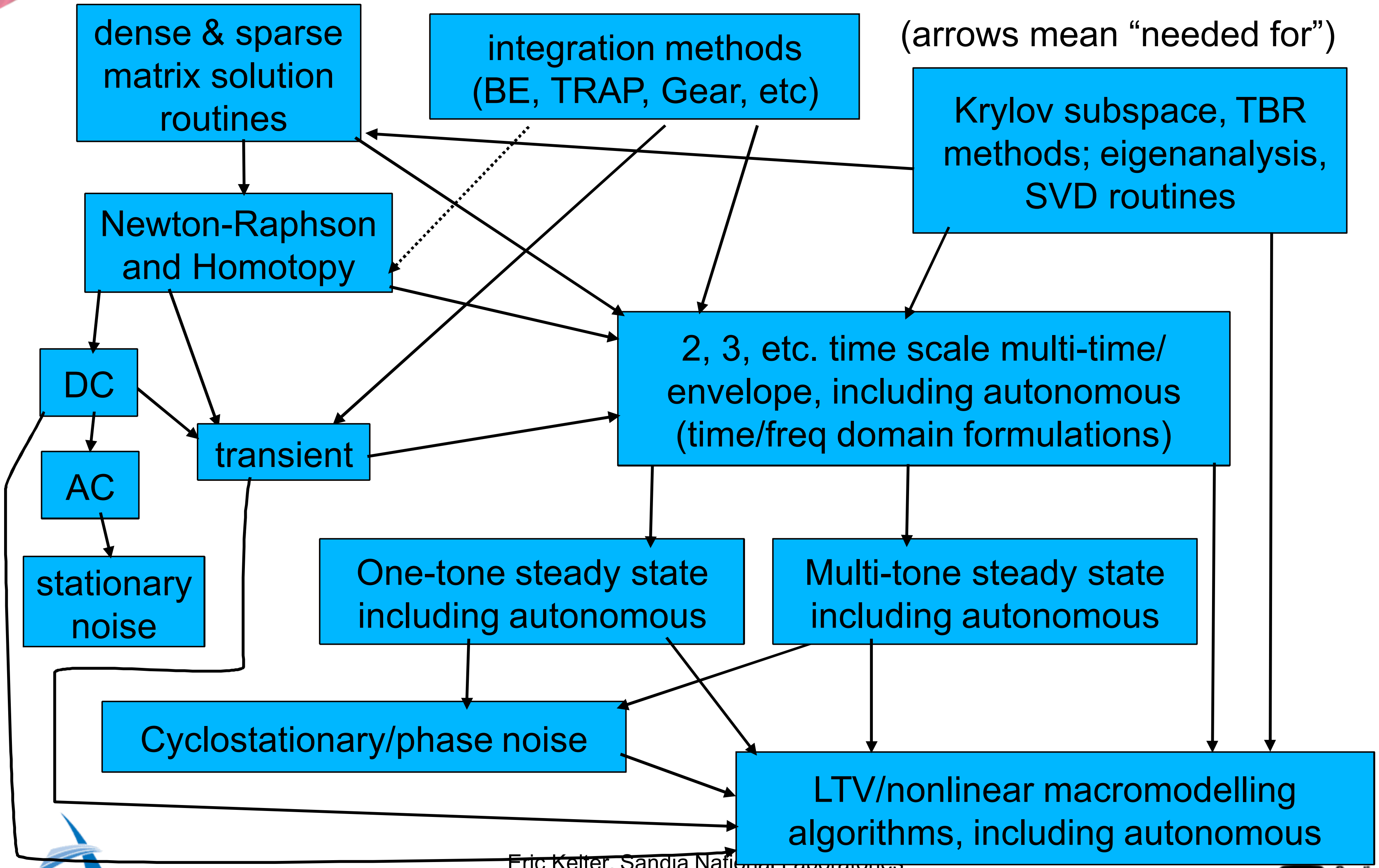
Errors and mistakes reduced

Modular Organization: The Big Picture

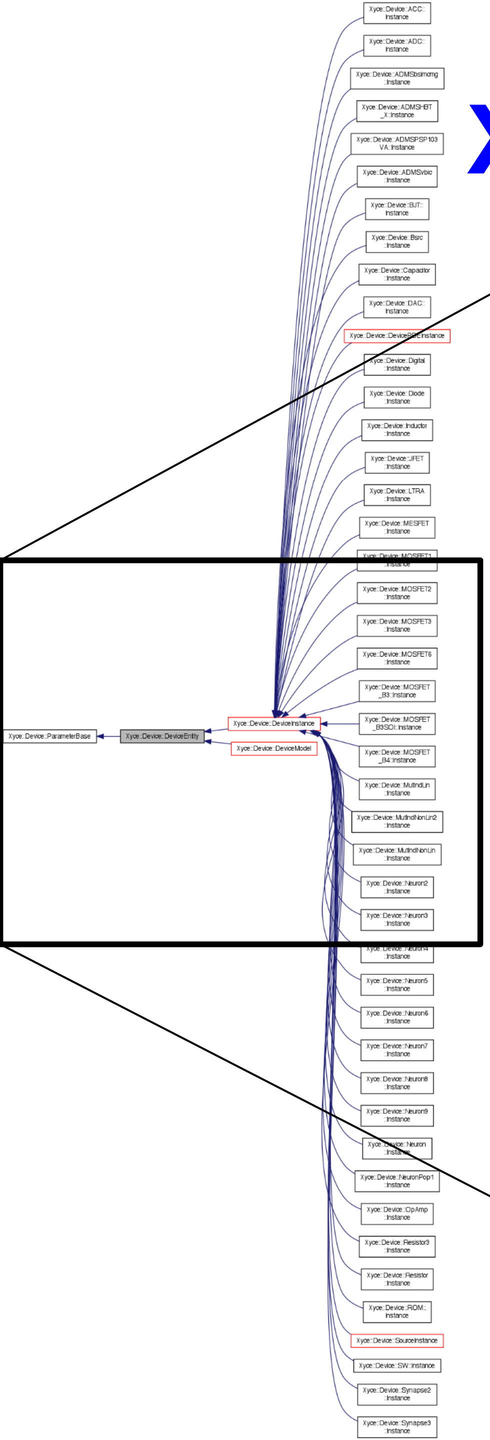

Ckt structure descriptions (w behavioural extensions)
Device model equations
User commands
Parser (sets up internal data structures for eqn engine/algorithms)
Circuit Equation Engine (eg, MNA, Tableau, etc)
$\frac{d}{dt}\vec{q}(\vec{x}(t)) + \vec{f}(\vec{x}(t), \vec{u}(t)) = \vec{0}$
PNJ/FETLIM?
User commands
Analysis algorithms: DC, AC, transient, HB, shooting, envelope, multi-time, noise, macromodelling, etc.
Analysis outputs: system-level structures, macromodels
Analysis results: waveforms, files, tables

(arrows mean “needed for”)

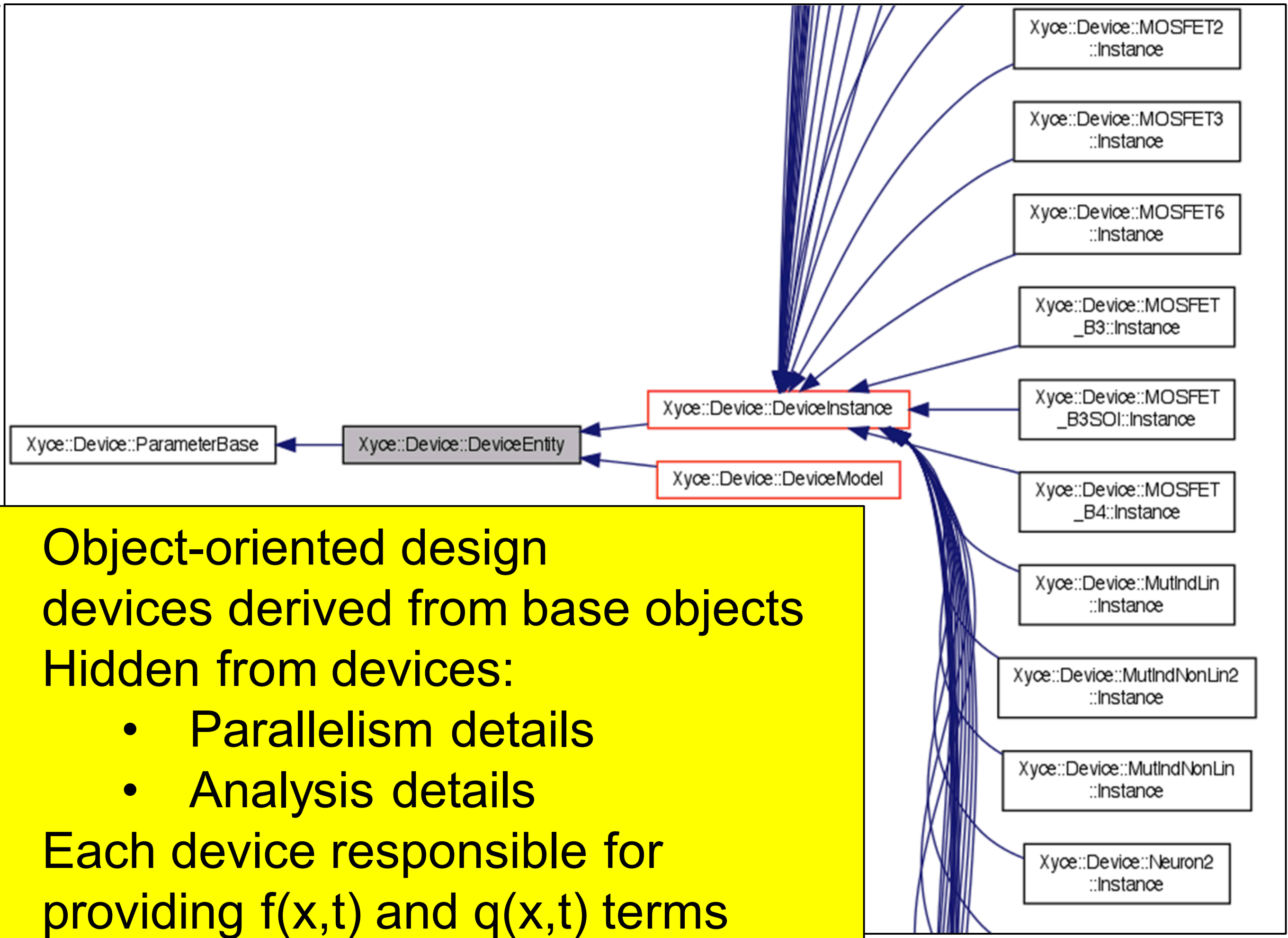
Analysis Algorithms: Structure


dense & sparse matrix solution routines
integration methods (BE, TRAP, Gear, etc)
(arrows mean "needed for")
Krylov subspace, TBR methods; eigenanalysis, SVD routines
Newton-Raphson and Homotopy
2, 3, etc. time scale multi-time/ envelope, including autonomous (time/freq domain formulations)
DC
transient
AC
stationary noise
One-tone steady state including autonomous
Multi-tone steady state including autonomous
Cyclostationary/phase noise
LTV/nonlinear macromodelling algorithms, including autonomous

Xyce device hierarchy



Xyce::Device::ACC::Instance
Xyce::Device::ADC::Instance
Xyce::Device::ADMSbsimcmg::Instance
Xyce::Device::ADMSHBT_X::Instance
Xyce::Device::ADMSPSP103VA::Instance
Xyce::Device::ADMSvbic::Instance
Xyce::Device::BJT::Instance
Xyce::Device::Bsrc::Instance
Xyce::Device::Capacitor::Instance
Xyce::Device::DAC::Instance
Xyce::Device::DeviceBlockInstance
Xyce::Device::Digital::Instance
Xyce::Device::Diode::Instance
Xyce::Device::Inductor::Instance
Xyce::Device::JFET::Instance
Xyce::Device::LTRA::Instance
Xyce::Device::MESFET::Instance
Xyce::Device::MOSFET1::Instance
Xyce::Device::MOSFET2::Instance
Xyce::Device::MOSFET3::Instance
Xyce::Device::MOSFET6::Instance
Xyce::Device::MOSFET_B3::Instance
Xyce::Device::MOSFET_B3SOI::Instance
Xyce::Device::MOSFET_B4::Instance
Xyce::Device::MutIndLin::Instance
Xyce::Device::MutIndNonLin2::Instance
Xyce::Device::MutIndNonLin::Instance
Xyce::Device::Neuron2::Instance
Xyce::Device::Neuron3::Instance
Xyce::Device::Neuron4::Instance
Xyce::Device::Neuron5::Instance
Xyce::Device::Neuron6::Instance
Xyce::Device::Neuron7::Instance
Xyce::Device::Neuron8::Instance
Xyce::Device::Neuron9::Instance
Xyce::Device::Neuron::Instance
Xyce::Device::NeuronPop1::Instance
Xyce::Device::OpAmp::Instance
Xyce::Device::Resistor3::Instance
Xyce::Device::Resistor::Instance
Xyce::Device::ROM::Instance
Xyce::Device::SourceInstance
Xyce::Device::SW::Instance
Xyce::Device::Synapse2::Instance
Xyce::Device::Synapse3::Instance
Xyce::Device::ParameterBase
Xyce::Device::DeviceEntity
Xyce::Device::DeviceInstance
Xyce::Device::DeviceModel



Xyce::Device::MOSFET2::Instance
Xyce::Device::MOSFET3::Instance
Xyce::Device::MOSFET6::Instance
Xyce::Device::MOSFET_B3::Instance
Xyce::Device::MOSFET_B3SOI::Instance
Xyce::Device::MOSFET_B4::Instance
Xyce::Device::MutIndLin::Instance
Xyce::Device::MutIndNonLin2::Instance
Xyce::Device::MutIndNonLin::Instance
Xyce::Device::Neuron2::Instance
Xyce::Device::DeviceInstance
Xyce::Device::ParameterBase
Xyce::Device::DeviceEntity
Xyce::Device::DeviceModel

- Object-oriented design
- devices derived from base objects
- Hidden from devices:
  - Parallelism details
  - Analysis details
- Each device responsible for providing f(x,t) and q(x,t) terms

$$\frac{d}{dt}\vec{q}(\vec{x}(t)) + \vec{f}(\vec{x}(t),\vec{u}(t)) = \vec{0}$$

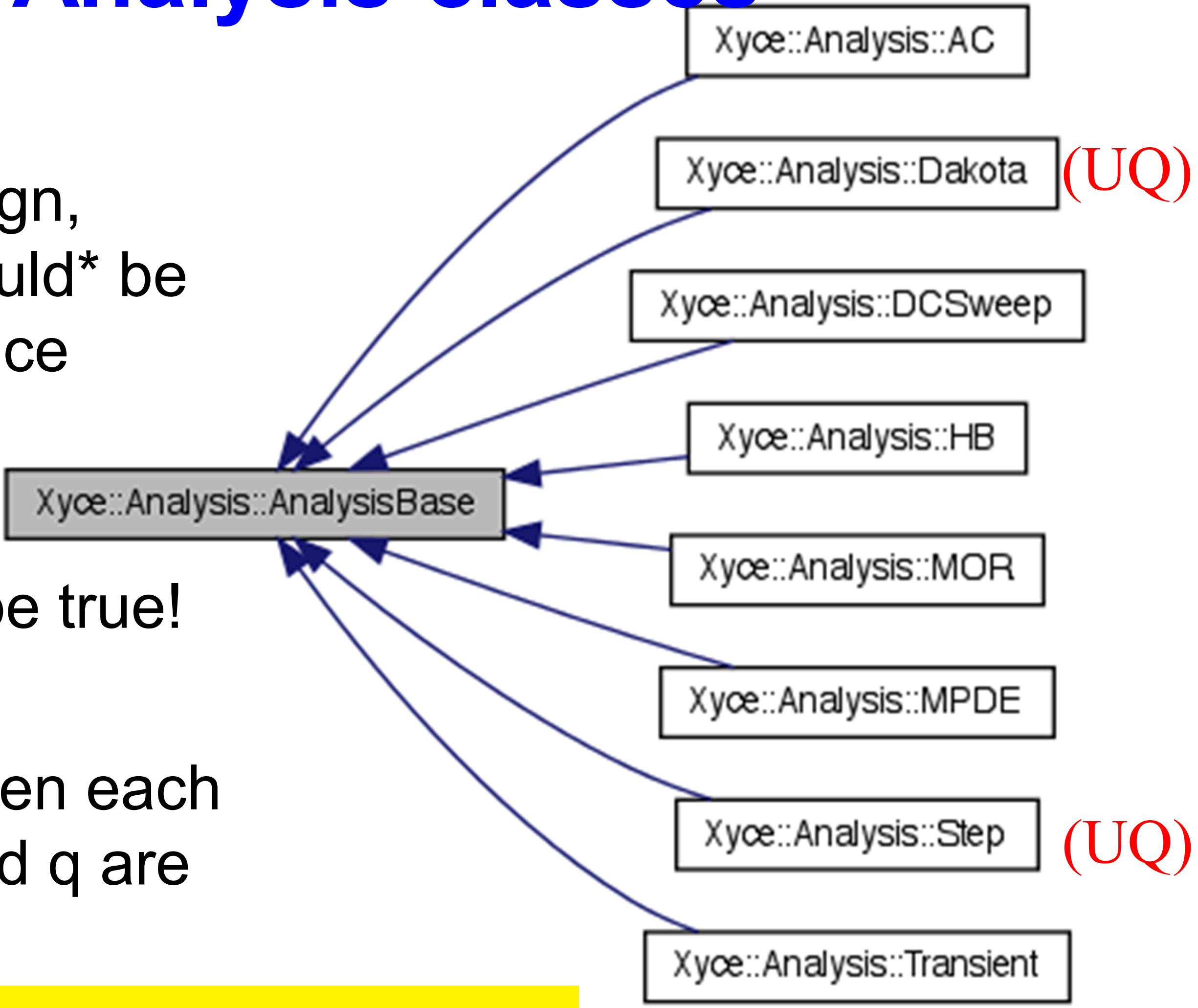
Xyce Analysis classes

With proper code design, analysis types *should* be independent of device models

Reverse should also be true!

Main difference between each of these is how f and q are handled.

$$\frac{d}{dt}\vec{q}(\vec{x}(t)) + \vec{f}(\vec{x}(t),\vec{u}(t)) = \vec{0}$$



Harmonic Balance

- Expands state variables as a Fourier series; solves for the Fourier coefficients
  - Insensitive to widely spaced spectral components
  - Excellent for dealing with complex high-frequency passive (linear) components
  - Directly captures the large-signal quasi-periodic steady-state
- Standard set of circuit equations:

Circuit DAE: $$g(x(t)) + \frac{d}{dt} q(x(t)) = u(t)$$

Approximate solution with Fourier expansion: $$x(t) = \sum_{h=-M}^{M} X_h \exp(j\omega_h t)$$

$$\Downarrow$$

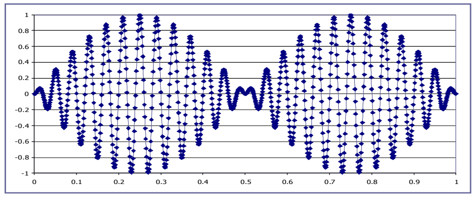
Frequency domain equation: $$G(X) + \Omega Q(X) = U$$

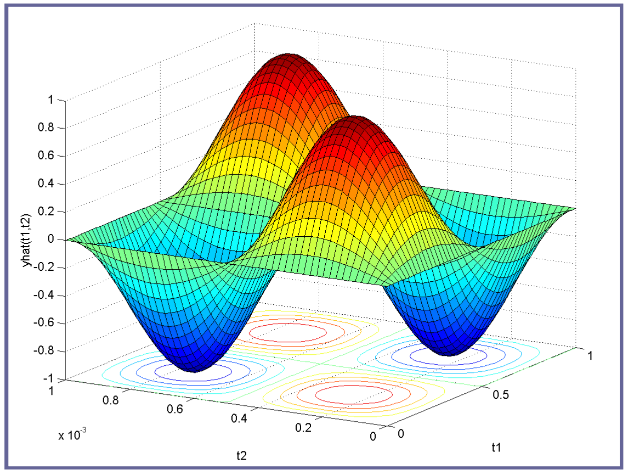
- Block linear system:
  - N=original system size
  - M=number of Fourier terms
  - Total size = N*M

- New for Xyce v6.1: Parallel HB
- New for Xyce v6.3:
  - Multi-tone HB
  - Unlimited number of tones
  - Various stability enhancements

MPDE Analysis

- Multi-time PDE (MPDE)
- Useful for problems with widely differing time scales
  - Slow envelope
  - Fast periodic
- Original DAE transformed into a PDE-like problem, where fast timescale solved all at once:





Original DAE $$\frac{d}{dt}q(x,t) + f(x,t) = b(t)$$

MPDE $$\frac{\partial}{\partial t_1}q(x,t_1,t_2) + \frac{\partial}{\partial t_2}q(x,t_1,t_2) + f(x,t_1,t_2) = b(t)$$

$$x(t) = \sin(2\pi t)\sin(2\pi 10^9 t)$$

$$\hat{x}(t_1,t_2) = \sin(2\pi t_1)\sin(2\pi 10^9 t_2)$$

$$x(t) = \hat{x}(t,t)$$

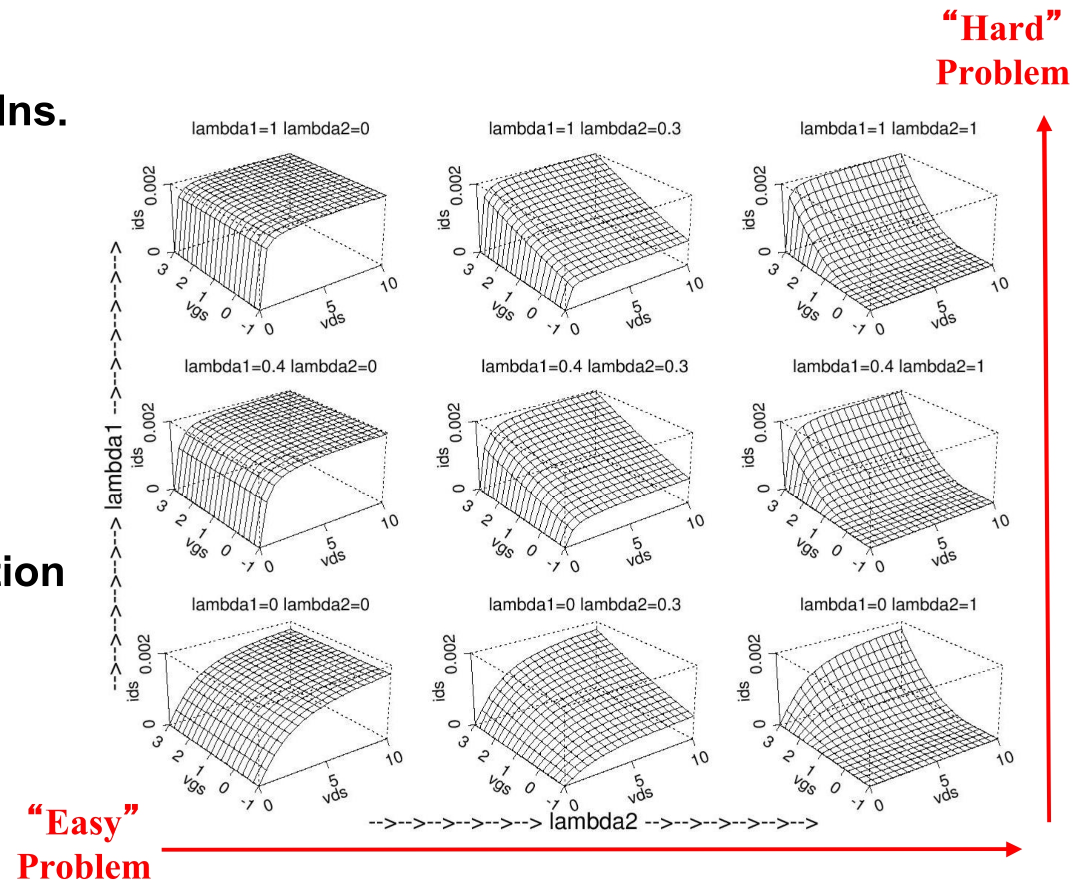
DCOP Non-linear solution techniques

- DCOP solution can be difficult
  - No initial guess!
- Some circuits have multiple DC solns.
- Need to find stable solution.
- DCOP is initial condition to most analysis methods:
  - TRAN, AC, HB, etc

- Homotopy:
  - o Relax stiffness
  - o continuation params
  - o Obtain solution via continuation

- Common examples (Xyce & SPICE)
  - o Voltage source stepping
  - o GMIN stepping

- Less common (Xyce):
  - o Pseudo-transient
  - o MOSFET homotopy (see right)

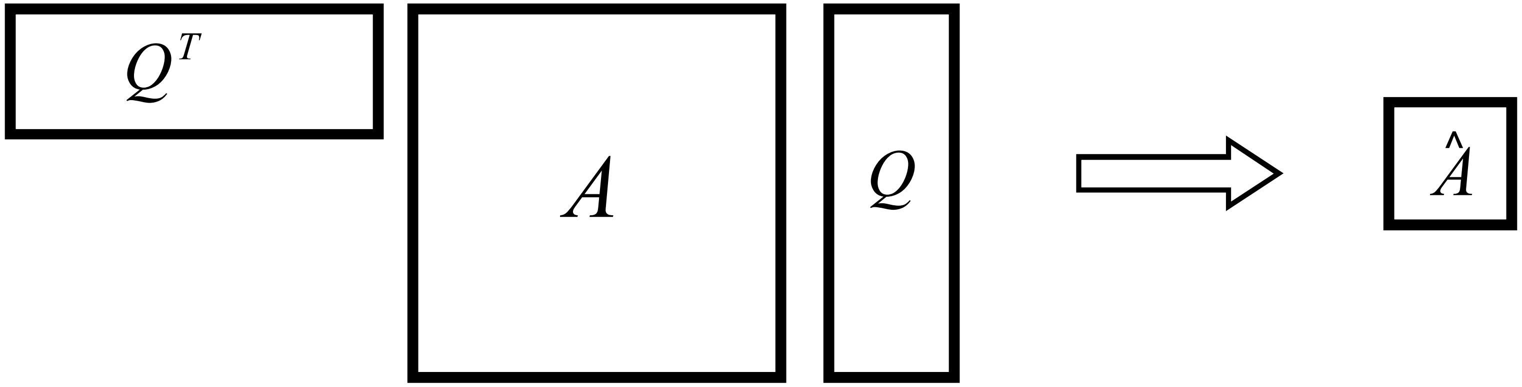


MOSFET continuation example

* Figure 12 from, Jaijeet Roychowdhury and Robert Melville, "Delivering Global DC Convergence for Large Mixed-Signal Circuits via Homotopy/Continuation Methods" IEEE Trans. CAD of ICAS, vol 25, no 1, 2006.

Model Order Reduction(MOR)

- Model reduction: in a rigorous manner, generate a system of the same form, but smaller dimension, with input-output behavior approximately the same.
- Very important for large linear networks, such as post-layout parasitics.
- Several projection methods are implemented in Xyce.
- Projection squashes matrices to smaller size



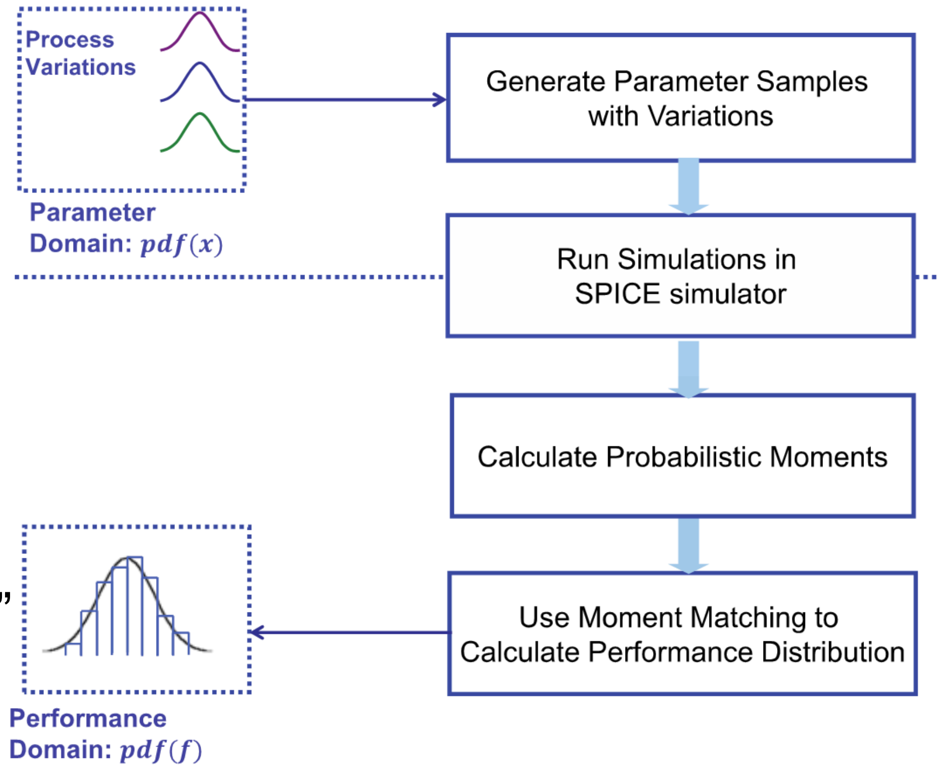
- Several MOR methods for linear systems implemented in Xyce
- New auto-detection code being implemented to automatically detect and reduce large linear subcircuits. (should be in Xyce v6.3)

Uncertainty Quantification (UQ)

- Xyce has a number of capabilities to support uncertainty quantification (UQ)
- Mostly these come via the DAKOTA framework.

- General idea: given uncertainty on parameter inputs, how to estimate that on circuit outputs.

- Most simulators support “brute force” approaches based on sampling.
- Many much more sophisticated methods exist, including stochastic collocation methods, etc.


Process Variations
Parameter Domain: $pdf(x)$
Generate Parameter Samples with Variations
Run Simulations in SPICE simulator
Calculate Probabilistic Moments
Use Moment Matching to Calculate Performance Distribution
Performance Domain: $pdf(f)$

Transient Direct Sensitivities

- New in Xyce v6.2
- Solves for dO/dp, where O=objective function, p=parameter.
- Can be applied to *any* device model parameter.
- In general, direct form solves

$$\frac{dO}{dp} = \frac{dO}{dx}\left[\left(\frac{dF}{dx}\right)^{-1}\frac{dF}{dp}\right] + \frac{dO}{dp}$$

- For transient, the linear system being solved is specific to the integration method. For backward Euler, this gives:

$$\left[\frac{1}{h}\frac{dq}{dx} + \frac{dj}{dx}\right]\frac{dx}{dp}_{n+1} = -\frac{1}{h}\left[\frac{dq}{dp}_{n+1} - \frac{dq}{dp}_{n}\right] - \frac{dj}{dp} + \frac{db}{dp} + \frac{1}{h}\left[\frac{dq}{dx}\right]\frac{dx}{dp}_{n}$$

- The dq/dp, dj/dp and db/dp terms come from compact device models.
- Xyce will compute these automatically; method depends on device implementation
  - Hand-coded derivatives in simple devices (resistors, etc)
  - Automatic-differentiation in devices that have been instrumented for it
  - Finite difference derivatives if neither are available.
- ADMS Back end currently doesn't instrument for AD parameter derivatives, but this is planned for this year.

Transient Direct Sensitivities

Example circuit.

Simple diode, driven by sinewave source.

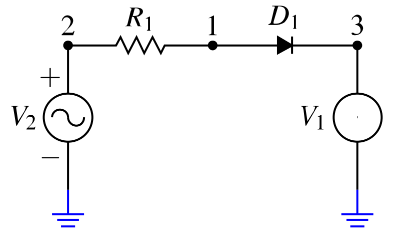


Figure 4.9. Transient Diode Circuit

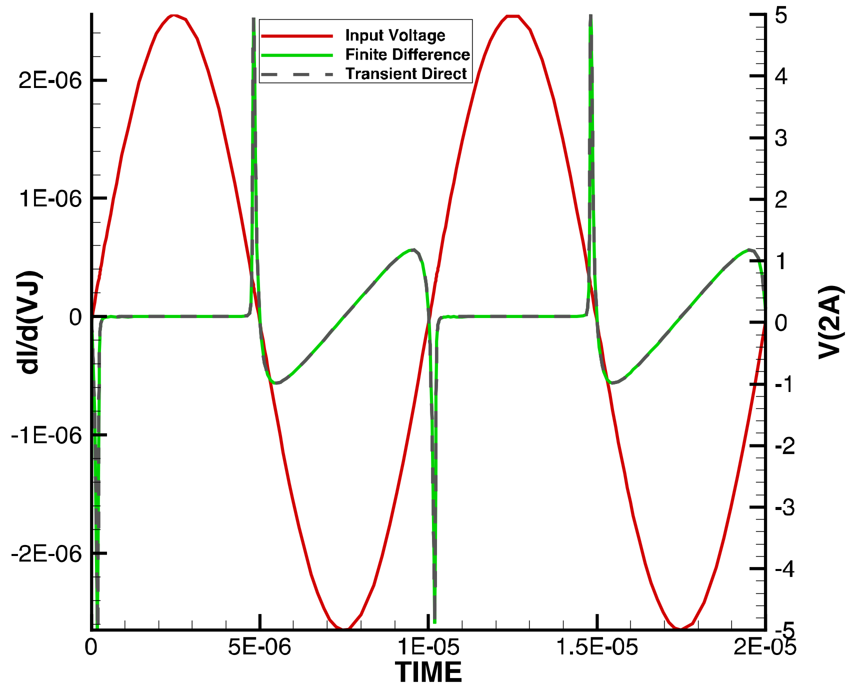
```
transient diode circuit sensitivity calculation
R 1 2 0.0001
V2 2 0 0.0 SIN(0 5 100K)
V1 3 0 0.0

D2 1 3 DZR
.MODEL DZR D( level=2 IS = 1E-14 RS = 10.8 N = 1 TT = 0
+ CJO = 1P VJ = 1 M = .5 EG = 1.11
+ XTI = 3 KF = 0 AF = 1 FC = .5
+ BV = 7.255 IBV = .001 tbv1 = 0.00013 tbv2 = -5e-8 )

.SENS objfunc={I(V1)}
+ param=DZR:VJ,DZR:CJO,DZR:EG,DZR:XTI,DZR:M,DZR:IS,DZR:RS,DZR:N

.options SENSITIVITY adjoint=0 direct=1

.options timeint method=gear
.TRAN 0 2e-5
.print TRAN format=tecplot v(2) I(V1)
.print SENS format=tecplot v(2)
.options device temp=25
.END
```



What about adjoints?

Xyce supports adjoint parameter sensitivites for steady-state (DCOP)

$$\frac{dO}{dp} = \left[\frac{dO}{dx}\left(\frac{dF}{dx}\right)^{-1}\right]\frac{dF}{dp} + \frac{dO}{dp}$$

$$\left[\frac{dO}{dx}\left(\frac{dF}{dx}\right)^{-1}\right] = \left(\frac{dj}{dx}\right)^{-T}\frac{dO}{dx} \longrightarrow \left(\frac{dj}{dx}\right)^{T}\frac{dO}{dF} = \frac{dO}{dx}$$

$$\frac{dO}{dp} = \frac{dO}{dF}\cdot\left(\frac{dj}{dp} - \frac{db}{dp}\right)$$

Transient adjoints are planned for this year.

ADMS-Xyce

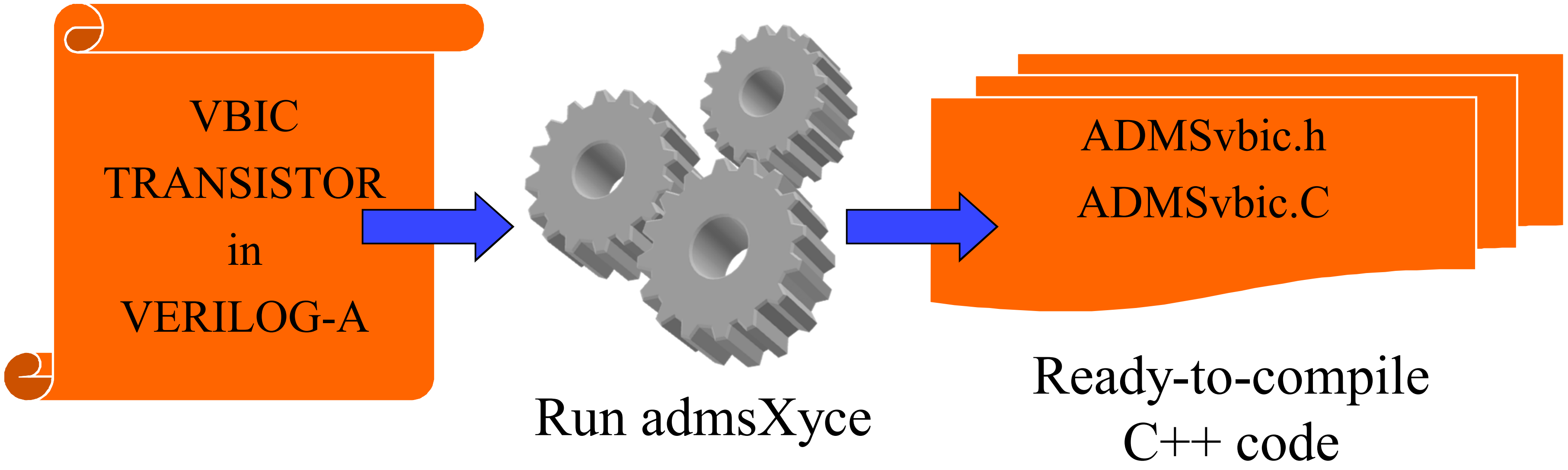
Verilog-A interface, via ADMS model compiler

- VBIC, Mextram, EKV, HiCUM, etc.

Verilog-A: industry standard format for new models

ADMS translates Verilog-A to compilable C/C++ code;

API automatically handles data structures, matrices, tedious details.



ADMS-Xyce

ADMS-converted models in Xyce:

- VBIC (the first model we did)
- EKV (can't distribute thanks to NDA, so it's not in Xyce open-source)
- PSP (version 103)
- BSIM-CMG 107 (only one of the many variants)
- FBH HBT_X version 2.1
- FBH HBT_X version 2.3

- BSIM6 (development branch)
- MEXTRAM (pending)
- MVS Silicon virtual source model

ADMS-Xyce

What is unique about Xyce's use of ADMS?

- Sacado AD library: By using Sacado, we are able to do away with an enormous amount of "admst" code for computing derivatives (compare with ng-spice's or qucs ADMS back-ends). It is difficult to overstate how much back-end work this saves.
- Have added "$limit" support (simple mod to ADMS required to let "$limit" be recognized as a legal function, but otherwise all the work is in the back-end)
- Had to add some "attributes" for parameters and modules to provide certain metadata (descriptions, units, "model" vs. "instance", level number, whether it's a Q, M, Y device, etc.).

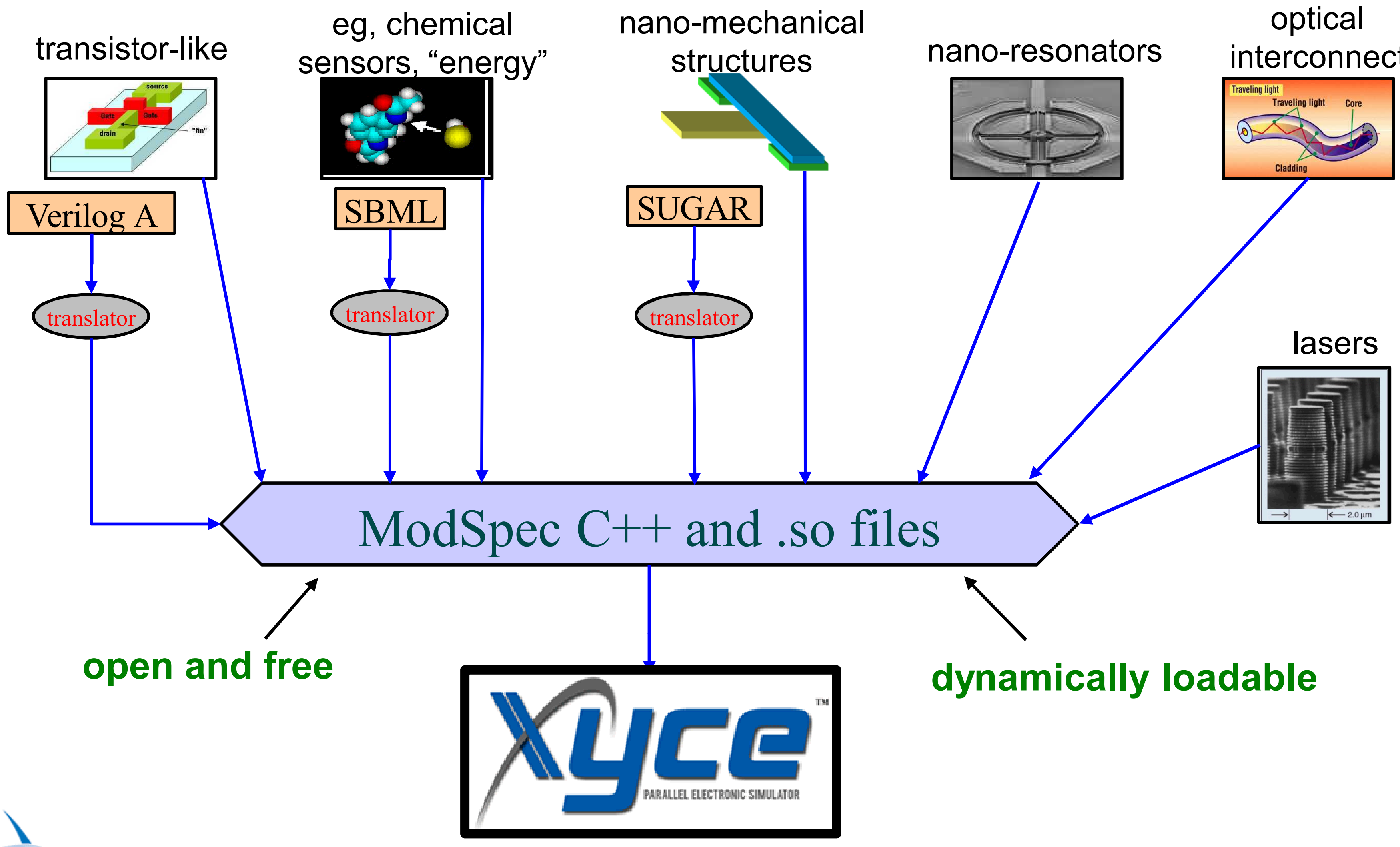
ADMS Challenges

- ADMS back-ends are written in ADMST, a sort of XSLT superset.
- Xyce's linear system differs from SPICE so generating code for voltage limiting ($limit) is not as simple as generating a function call.
- ADMS has NO support for current branches. It generates datastructures representing the Jacobian resulting when all variables are nodal voltages, but you must do all the work yourself if you throw a branch current in the mix.
- Node collapse: since this has to happen at the time the device is instantiated (rather than the time its equations are evaluated), it is necessary to generate (separately from the rest of the evaluation) the parts of code needed to compute all the variables required to determine whether or not to collapse.
- Need to extend ADMS back-end to produce parameter sensitivities with Sacado.
- Dynamic linking: this has been done, but is still immature.

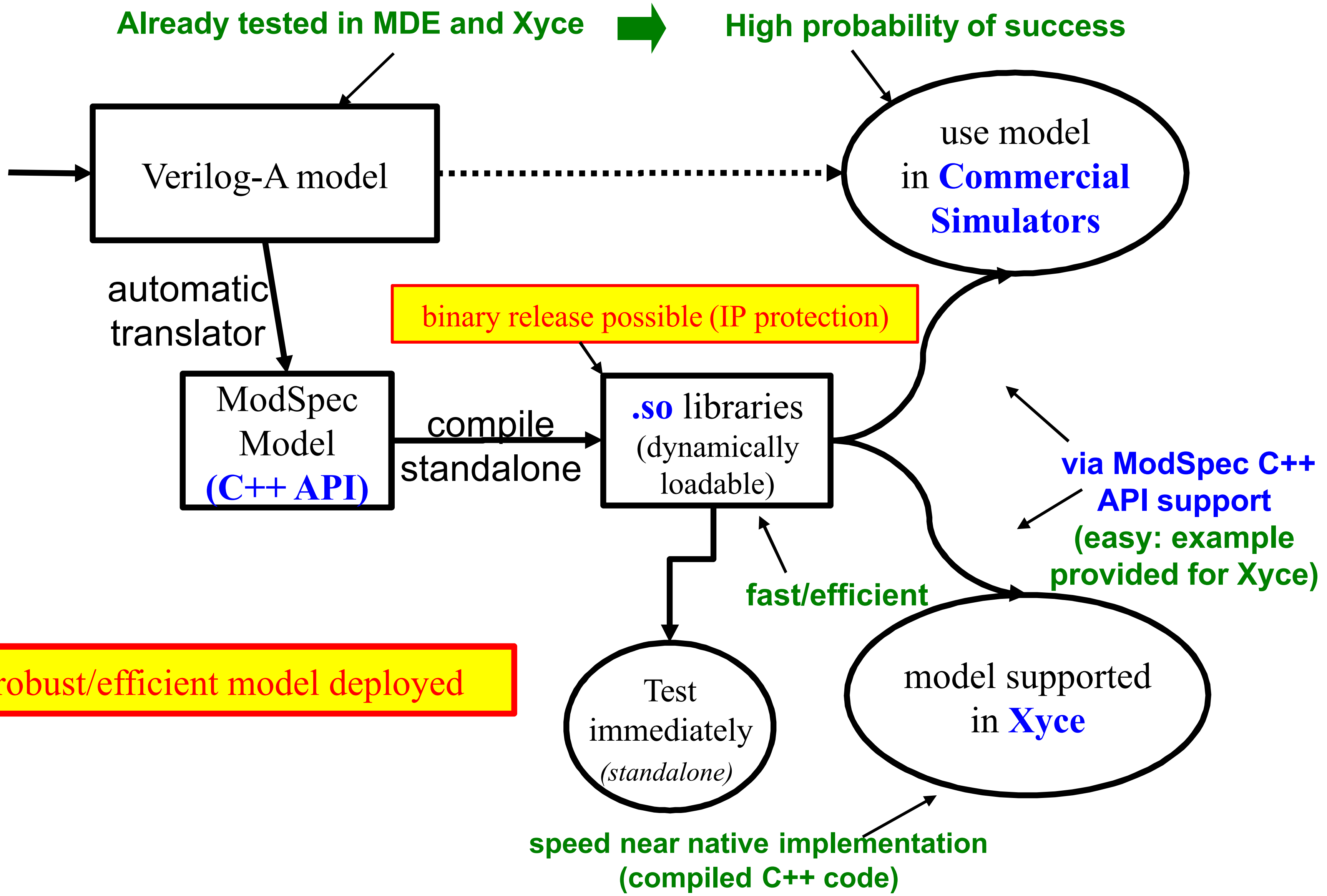
ADMS Challenges

- not well suited for multi-physics domains
  - even for electrical domain, doesn't support structures like internal current unknowns
- no longer open-source
  - ADMS's website: “*Noovela Consulting offers - at interesting rate - support to implement your compact models into spice simulators*”
  - future support uncertain
- written in XSLT
  - difficult to work with or maintain
  - e.g. took 6-7 years by Tom Russo (working off and on on it) even with ADMS already existing

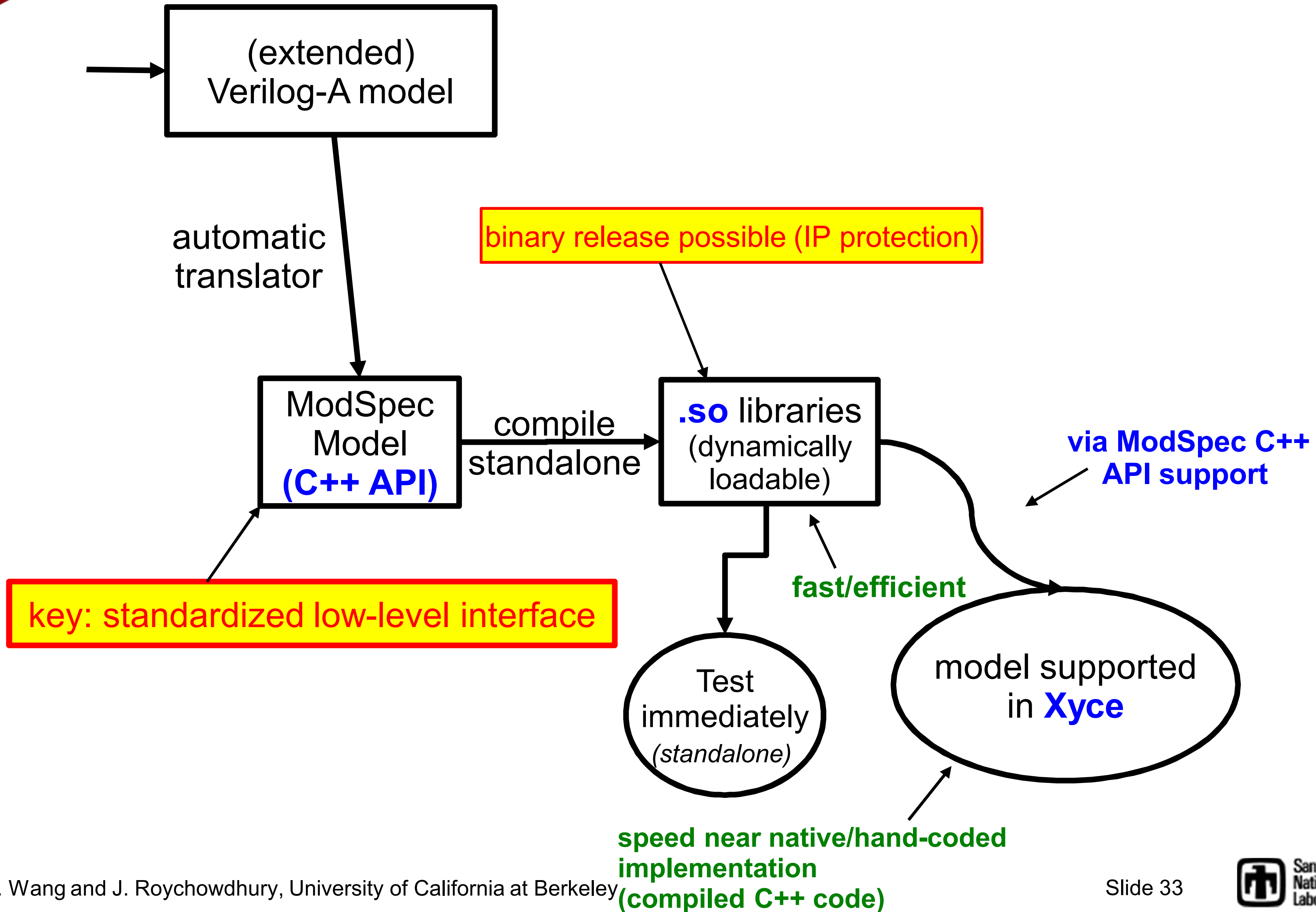
Multi-physics support in Xyce through ModSpec



Model Development: Verilog-A → ModSpec → Xyce


Already tested in MDE and Xyce
High probability of success
Verilog-A model
use model in Commercial Simulators
automatic translator
binary release possible (IP protection)
ModSpec Model (C++ API)
compile standalone
.so libraries (dynamically loadable)
via ModSpec C++ API support (easy: example provided for Xyce)
fast/efficient
robust/efficient model deployed
Test immediately (standalone)
model supported in Xyce
speed near native implementation (compiled C++ code)

Using ModSpec in Xyce


(extended) Verilog-A model
automatic translator
binary release possible (IP protection)
ModSpec Model (C++ API)
compile standalone
.so libraries (dynamically loadable)
via ModSpec C++ API support
fast/efficient
key: standardized low-level interface
Test immediately (standalone)
model supported in Xyce
speed near native/hand-coded implementation (compiled C++ code)

Xyce Summary

- Mature code; under development for 14 years
  - Large regression test suite
  - Regular, formal code releases
- Under active development
- Open source GPL v3.
- Modular object-oriented design
  - DAE form: $\frac{d}{dt}q(x,t) + f(x,t) = b(t)$
  - Quick, independent development of models and algorithms
- Model support
  - Legacy spice models
  - Non-electrical: neuron, reaction networks
  - CMC models
  - ADMS integration
  - Mod-Spec integration

Acknowledgements

Xyce team

xyce.sandia.gov

- Scott Hutchinson
- Tom Russo
- Heidi Thornquist
- Jason Verley
- Rich Schiek
- Ting Mei
- Dave Baur
- Sivasankaran Rajamanickam

Trilinos team

trilinos.sandia.gov

Dakota team

dakota.sandia.gov

Thanks!

Publications / Presentations

Publications

- *"A Hybrid Approach for Parallel Transistor-Level Full-Chip Circuit Simulation", VECPAR 2014*
- *"Electrical Modeling and Simulation for Stockpile Stewardship",* ACM XRDS, 2013
- *"ShyLU: A Hybrid-Hybrid Solver for Multicore Platforms"*, IPDPS 2012
- *"Parallel Transistor-Level Circuit Simulation",* Simulation and Verification of Electronic and Biological Systems, Springer, 2011
- *"A Parallel Preconditioning Strategy for Efficient Transistor-Level Circuit Simulation"*, ICCAD 2009

Presentations

- *"Sparse Matrix Techniques for Next-Generation Parallel Transistor-Level Circuit Simulation"*

  Heidi K. Thornquist, Parallel Matrix Algorithms and Applications 2012
- *"Partitioning for Hybrid Solvers: ShyLU and HIPS"*

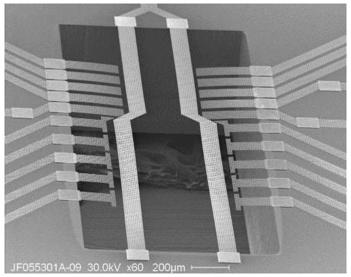
  Erik G. Boman, Siva Rajamanickam, and Jeremie Gaidamour, Copper Mtn. 2012
- *"Efficient Preconditioners for Large-Scale Parallel Circuit Simulation"*

  Heidi K. Thornquist, SIAM Computational Science & Engineering 2011
- *"Advances in Parallel Transistor-Level Circuit Simulation"*

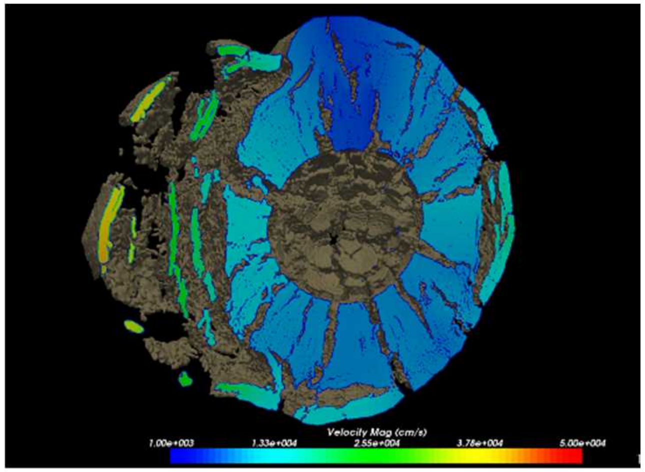
  Heidi K. Thornquist, Scientific Computing in Electrical Engineering 2010
- *"Large Scale Parallel Circuit Simulation"*

  Heidi Thornquist and Eric Keiter, Circuit and Multi-Domain Simulation Workshop, ICCAD 2009

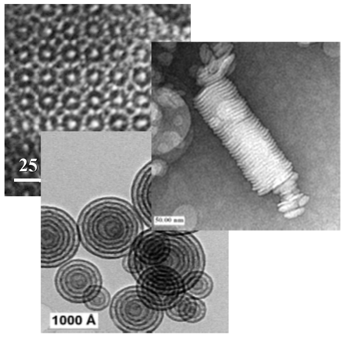
SNL has six core technical capabilities



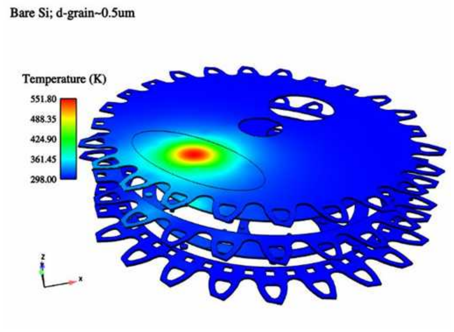
Microelectronics and Photonics



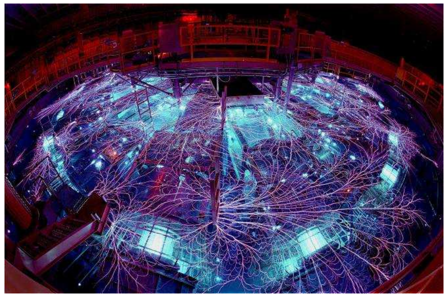
Computational & Informational Sciences

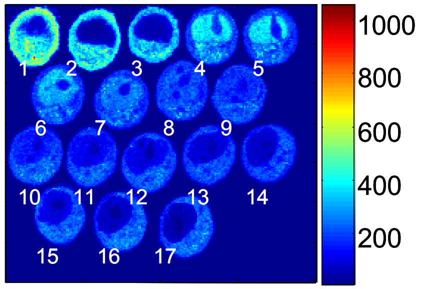


Materials Science & Technology



Engineering Sciences





Bioscience

Pulsed Power

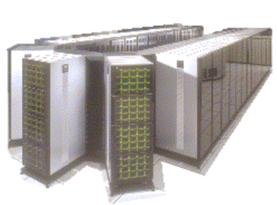
Eric Keiter, Sandia National Laboratories
2014 NEEDS Workshop, Berkeley, CA

CIS has a rich history in the development and maturation of high performance computing hardware and software technology

CM-2

nCUBE-2

iPSC-860


Paragon

ASCI Red

Cplant

Red Storm

1987 1988 1989 1990 1991 1992 1993 1994 1995 1996 1997 1998 1999 2000 2001 2002 2003 2004 2005 2006 2007

Gordon Bell Prize

R&D 100
Parallel Software

Karp Challenge

Patent
Parallel Software

R&D 100
Meshing

Patent
Meshing

R&D 100
Signal Processing

Gordon Bell Prize

World Record
143 GFlops

World Record
281 GFlops

R&D 100
Dense Solvers

R&D 100
Storage

Patent
Paving

R&D 100
Aztec

SC96 Gold Medal
Networking

World Record
Teraflops

Gordon Bell Prize

R&D 100
Salvo

Patent
Decomposition

Patent
Data Mining

Mannheim
SuParCup

R&D 100
Trilinos

R&D 100
Allocator

R&D 100
Xyce

R&D 100
3D-Touch

Fernbach
Award

Eric Keiter, Sandia National Laboratories
2014 NEEDS Workshop, Berkeley, CA