NEEDS-SPICE: A System for Easing the Development of Simulation-Ready Compact Models

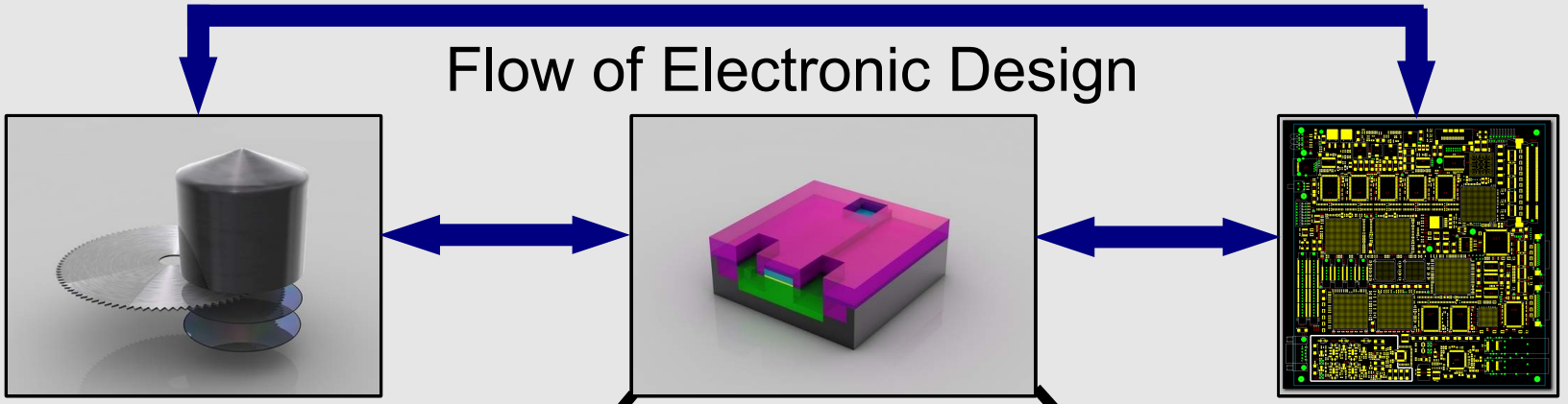
Tianshi Wang and Jaijeet Roychowdhury

EECS Department
University of California at Berkeley

Outline

- What is a compact model?
  - device equations written for circuit simulators
- How is a compact model used by a simulator?
  - ➔ how a simulator sets up equations
  - ➔ how it solves the equations
  - ➔ how simulation can fail (because of the compact model)
    - how a compact model can help avoid simulation failure
- Flows for developing compact models
  - Berkeley SPICE (direct C code)
  - Verilog-A and commercial simulators
  - NEEDS-SPICE & ModSpec (MATLAB and Verilog-A)

What is a Compact Model



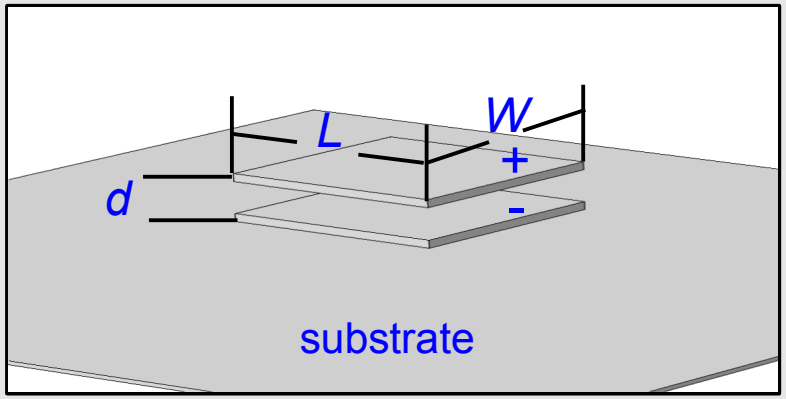
Analysis/Modelling for individual devices ("device simulation")

- Provides detailed information about device operation & characteristics
- Computationally intensive
  - EM simulation, drift-diffusion eqns., numerical solution of PDEs, etc.

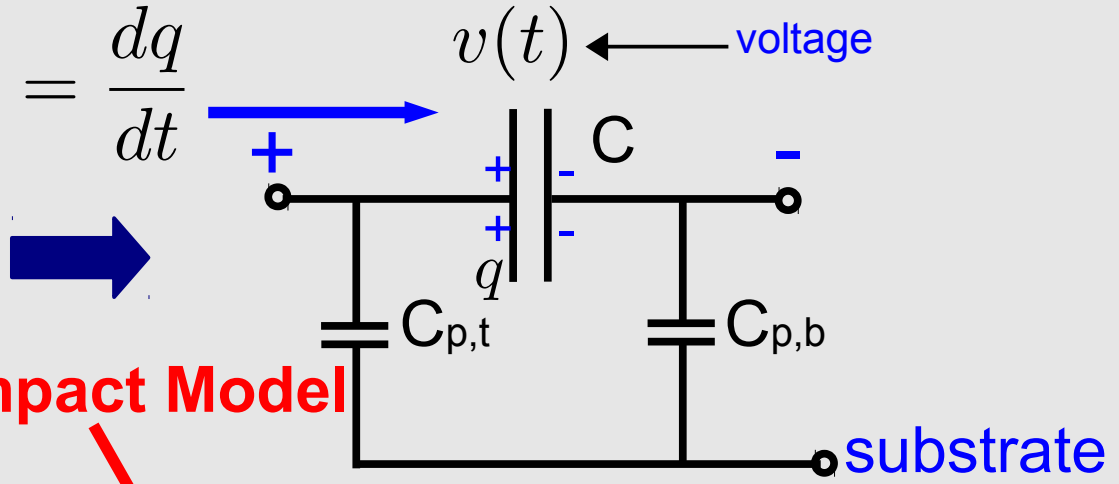
Compact Model of Device

- Simple enough to be incorporated in circuit simulators
- Accurate enough to have predictive value for circuits
- Terminal behaviour important
  - internal details less important
- Purpose: use in circuit design
  - via **circuit simulation**

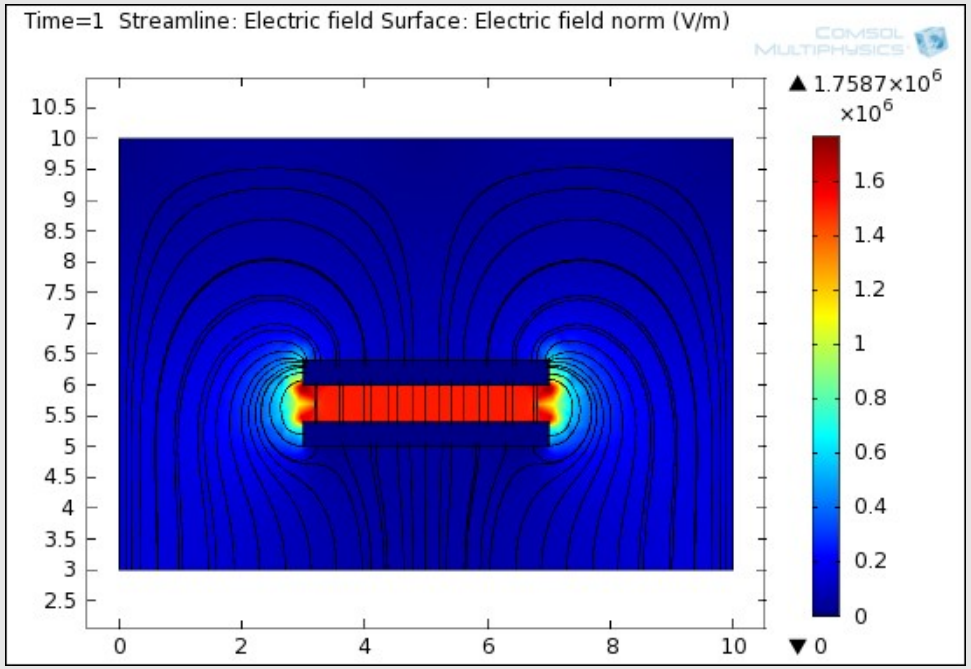
Example: Capacitor



$$i(t) = \frac{dq}{dt}$$



Compact Model



"device level" simulation
(eg, finite element electrostatic)

voltage-current relationships

$$q(v) = Cv \qquad i(t) = \frac{d}{dt}q\big(v(t)\big)$$

charge capacitance

$$C = \frac{\epsilon_r \epsilon_0 L \cdot W}{d}$$

electrical parameter

geometrical parameters

or

$$C = \frac{\epsilon_r \epsilon_0 L \cdot W}{d}\left[1 + \frac{d}{\pi W} + \frac{d}{\pi W}\ln\left(\frac{2\pi W}{d}\right)\right] + \cdots$$

H. B. Palmer 1927

$$I_{ts}(t) = C_{p,t} \cdot dV_{ts}(t)/dt$$
$$I_{bs}(t) = C_{p,b} \cdot dV_{bs}(t)/dt$$

complicated equations for $C_{p,t}$ and $C_{p,b}$

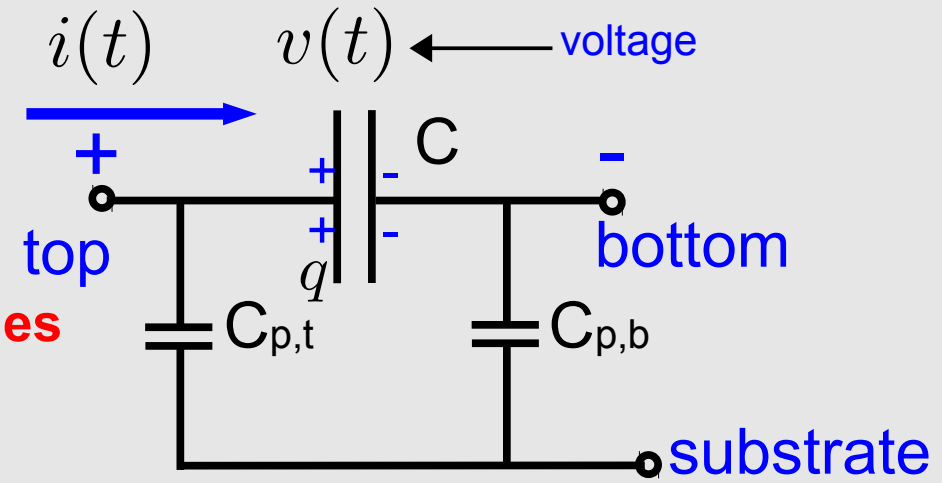
Example: Capacitor

Verilog-A Code:

```
 1 `include "disciplines.vams"
 2 `include "constants.vams"
 3
 4 module Horizontal_Cap(t, b, s);
 5
 6     inout t, b, s;
 7     electrical t, b, s;
 8
 9     branch (t ,b )              cap;
10     branch (t ,s )              ts;
11     branch (b ,s )              bs;
12
13     parameter real L = 4.0e-6   from (0:inf);
14     parameter real W = 4.0e-6   from (0:inf);
15     parameter real D = 0.6e-6   from (0:inf);
16     parameter real EPSR = 3.9   from (0:inf);
17     // other parameters
18
19     real C, Cpt, Cpb;
20
21     analog begin
22         C = `P_EPS0 * EPSR * L * W / D;
23         Cpt = ...; Cpb = ...;
24         I(cap) <+ ddt(C * V(cap));
25         I(ts)  <+ ddt(Cpt * V(ts));
26         I(bs)  <+ ddt(Cpb * V(bs));
27     end
28 endmodule
```

Nodes/Branches

Parameters



$$q(v) = Cv \qquad i(t) = \frac{d}{dt} q\big(v(t)\big)$$

$$C = \frac{\epsilon_r \epsilon_0 L \cdot W}{d}$$

or

$$C = \frac{\epsilon_r \epsilon_0 L \cdot W}{d}\left[1 + \frac{d}{\pi W} + \frac{d}{\pi W}\ln\left(\frac{2\pi W}{d}\right)\right] + \cdots$$

H. B. Palmer 1927

$$I_{ts}(t) = C_{p,t} \cdot dV_{ts}(t)/dt$$

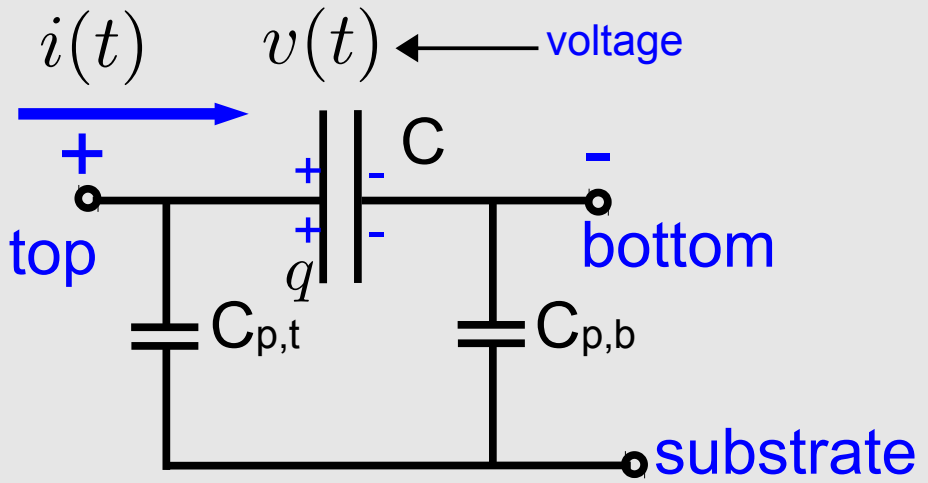
$$I_{bs}(t) = C_{p,b} \cdot dV_{bs}(t)/dt$$

complicated equations for $C_{p,t}$ and $C_{p,b}$

Example: Capacitor

ModSpec code (MATLAB)

```
 1 ... % Defining names, ID, etc
 2
 3 vecX = [vts; vbs];
 4 vecY = [];
 5 vecU = [];
 6
 7 ... % Defining parameters, branches, etc
 8
 9 function feout = fe(vecX, vecY, vecU, MOD)
10     feout = zeros(2,1); % [its; ibs]
11 end % fe
12
13 function qeout = qe(vecX, vecY, MOD)
14     ... % set up variable names from vecXY
15     Cpt = ...; Cpb = ...;
16     C = EPS0 * EPSR * L * W / D;
17     qeout(1,1) = Cpt * vts + C * vtb; %its
18     qeout(2,1) = Cpb * vbs + C * vtb; %ibs
19 end % qe
20
21 function fiout = fi(vecX, vecY, vecU, MOD)
22     fiout = [];
23 end % fi
24
25 function qiout = qi(vecX, vecY, MOD)
26     qiout = [];
27 end % qi
```



$$q(v) = Cv \qquad i(t) = \frac{d}{dt} q\big(v(t)\big)$$

$$C = \frac{\epsilon_r \epsilon_0 L \cdot W}{d}$$

or

$$C = \frac{\epsilon_r \epsilon_0 L \cdot W}{d}\left[1 + \frac{d}{\pi W} + \frac{d}{\pi W}\ln\left(\frac{2\pi W}{d}\right)\right] + \cdots$$

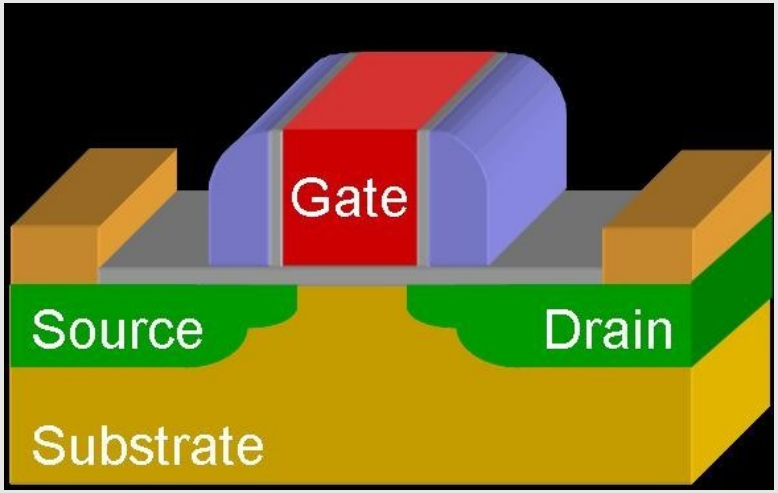
H. B. Palmer 1927

$$I_{ts}(t) = C_{p,t} \cdot dV_{ts}(t)/dt$$

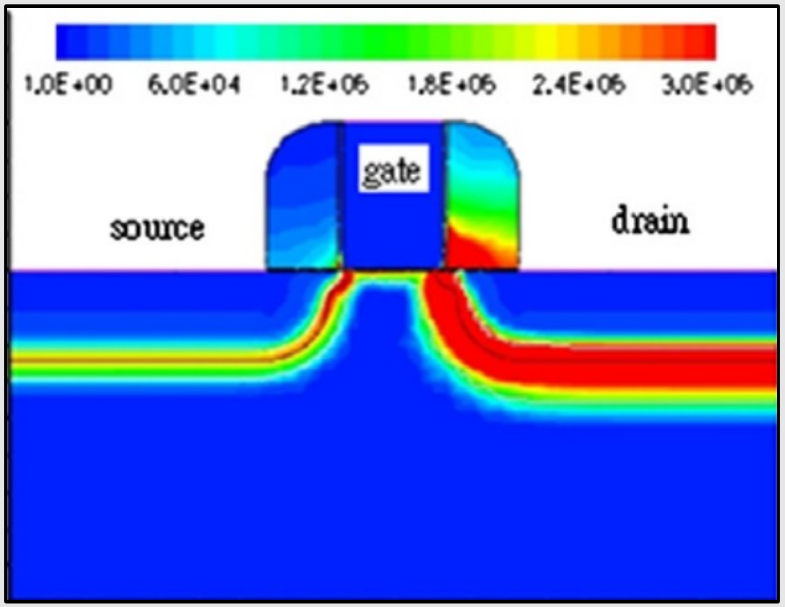
$$I_{bs}(t) = C_{p,b} \cdot dV_{bs}(t)/dt$$

complicated equations for $C_{p,t}$ and $C_{p,b}$

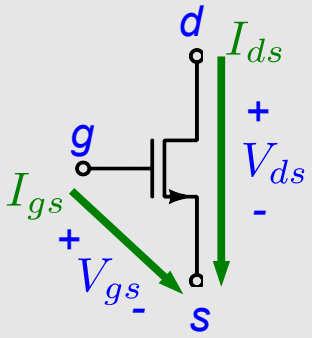
Example: MOSFET



University of Southern California, EE 327 website
http://www.isi.edu/~vernier/EE327/mosfet_3d_nm_king_ucb.jpg



S.L. Jang, J.S. Yuan, S.D. Yen, E. Kritchanchai, G.W. Huang "Experimental evaluation of hot electron reliability on differential Clapp-VCO" Microelectronics Reliability, Volume 53, Issue 2, February 2013, Pages 254–258



Compact Model

- Schichman-Hodges (core)

$$
\begin{aligned}
I_{ds} &= f(V_{gs}, V_{ds}) \\
&= \begin{cases} \beta \left[(V_{gs} - V_T) - \frac{V_{ds}}{2} \right] V_{ds}, & \text{if } V_{ds} < V_{gs} - V_T \\ \frac{\beta}{2} (V_{gs} - V_T)^2, & \text{if } V_{ds} \geq V_{gs} - V_T \\ 0 & \text{if } V_{gs} < V_T \end{cases} \\
I_{gs} &= 0
\end{aligned}
$$

- many other MOS models
  - BSIM, EKV, PSP, ...
  - MVS
    - ➔ MIT Virtual Source compact FET model

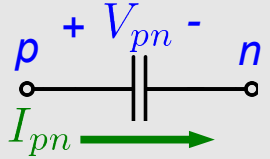
Using Compact Models in Simulation

Simulation Basics Compact Modellers Need to Know

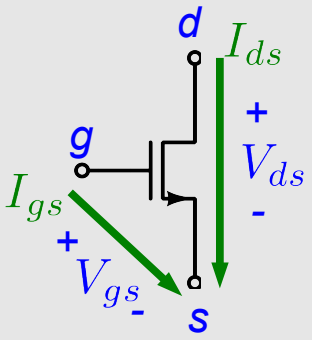
Outline

- What is a compact model?
  - writing device equations in a way useful for circuit design (via circuit simulation)
- **What does one do with a compact model?**
  - **simulation (within a circuit simulator)**
    - → **how a simulator sets up equations**
    - → **how it solves the equations**
    - → **how a compact model can make simulation fail**
      - **how to work around it when designing a compact model**
- Flows for developing compact models
  - Berkeley SPICE (direct C code)
  - Verilog-A and commercial simulators
  - ModSpec (in MATLAB or via Verilog-A)

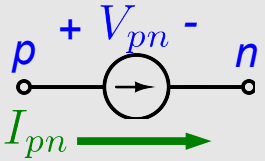
Circuit Equations



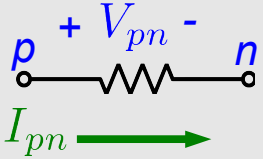
$$\text{Cap} : I_{pn} = C \cdot \frac{d}{dt} V_{pn}$$



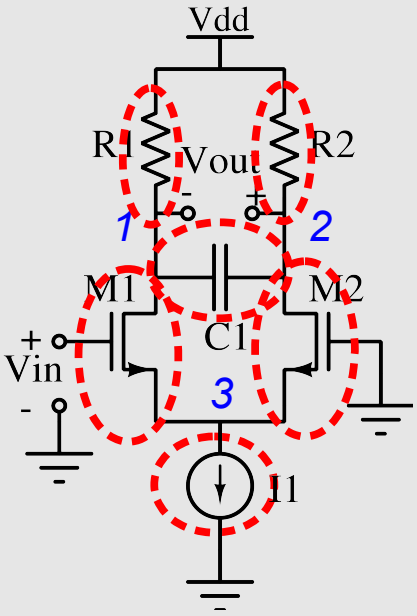
$$\begin{aligned} \text{Mos} : I_{ds} &= f(V_{gs}, V_{ds}) \\ &= \begin{cases} \beta \left[(V_{gs} - V_T) - \frac{V_{ds}}{2} \right] V_{ds}, & \text{if } V_{ds} < V_{gs} - V_T \\ \frac{\beta}{2} (V_{gs} - V_T)^2, & \text{if } V_{ds} \geq V_{gs} - V_T \\ 0 & \text{if } V_{gs} < V_T \end{cases} \\ I_{gs} &= 0 \end{aligned}$$



$$\text{Isrc} : I_{pn} = I$$



$$\text{Res} : I_{pn} = \frac{V_{pn}}{R}$$



Circuit Equations

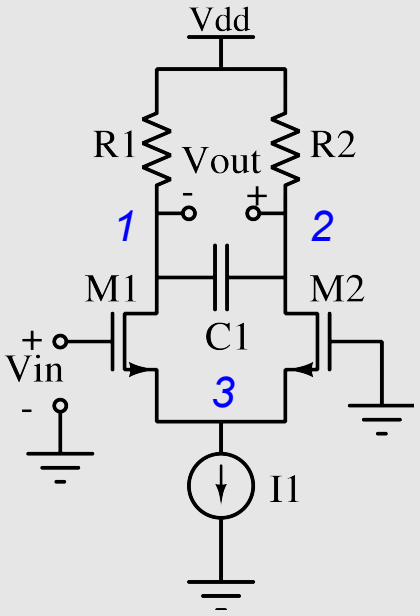
(made by composing device equations)

C1 M1 M2

$$\begin{aligned} \text{KCL}_1 &: \frac{V_{dd} - e_1}{R_1} - C \cdot \frac{d}{dt}(e_1 - e_2) - f_1(V_{in} - e_3, e_1 - e_3) = 0 \\ \text{KCL}_2 &: \frac{V_{dd} - e_2}{R_2} + C \cdot \frac{d}{dt}(e_1 - e_2) - f_2(-e_3, e_2 - e_3) = 0 \\ \text{KCL}_3 &: f_1(V_{in} - e_3, e_1 - e_3) + f_2(-e_3, e_2 - e_3) - I = 0 \end{aligned}$$

R1, R2 M1 M2 I1

Circuit Equations are DAEs



$$\begin{aligned}
\text{KCL}_1 \quad &: \quad \frac{V_{dd}-e_1}{R_1} - C\frac{d}{dt}(e_1-e_2) - f_1(V_{in}-e_3, e_1-e_3) = 0 \\
\text{KCL}_2 \quad &: \quad \frac{V_{dd}-e_2}{R_2} + C\frac{d}{dt}(e_1-e_2) - f_2(-e_3, e_2-e_3) = 0 \\
\text{KCL}_3 \quad &: \quad f_1(V_{in}-e_3, e_1-e_3) + f_2(-e_3, e_2-e_3) - I = 0
\end{aligned}$$

Differential Equations

Algebraic

DAEs

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

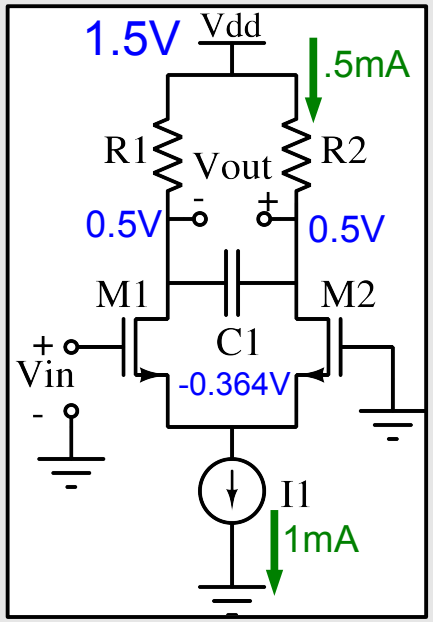
$$\vec{x}(t) = \begin{bmatrix} e_1(t) \\ e_2(t) \\ e_3(t) \end{bmatrix} \qquad \vec{q}(\vec{x}) = \begin{bmatrix} -C(e_1-e_2) \\ +C(e_1-e_2) \\ 0 \end{bmatrix}$$

$$\vec{u}(t) = [V_{in}(t)] \qquad \vec{f}(\vec{x},\vec{u}) = \begin{bmatrix} \frac{V_{dd}-e_1}{R_1} - f_1(V_{in}-e_3, e_1-e_3) \\ \frac{V_{dd}-e_2}{R_2} - f_2(-e_3, e_2-e_3) \\ f_1(V_{in}-e_3, e_1-e_3) + f_2(-e_3, e_2-e_3) \end{bmatrix}$$

Analyses on DAEs: DC/AC/TRAN

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

- **DC Analysis**
  - solve for $\vec{x}(t)$ assuming **nothing changes with time** (given constant $\vec{u}(t)$)



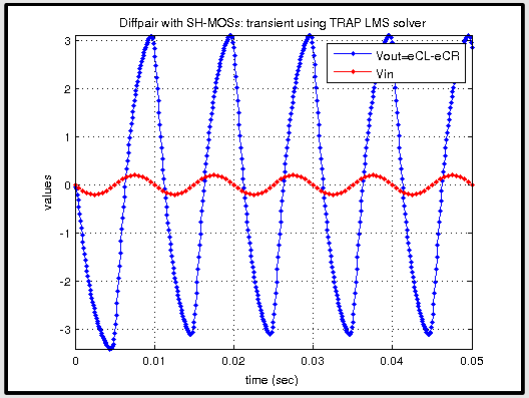
$Vin \equiv 0V$

solve using Newton-Raphson method

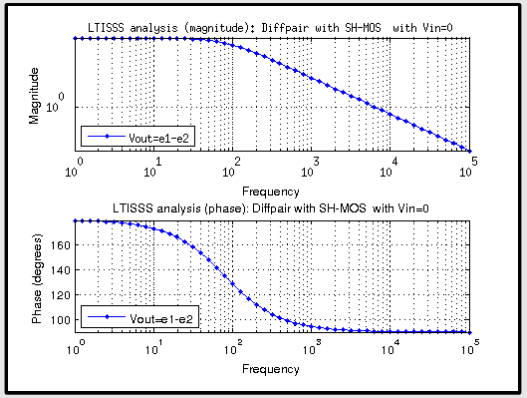
$\vec{g}(\vec{x}) \triangleq \vec{f}(\vec{x}, \vec{u}) = \vec{0}$

nonlinear

Transient Analysis

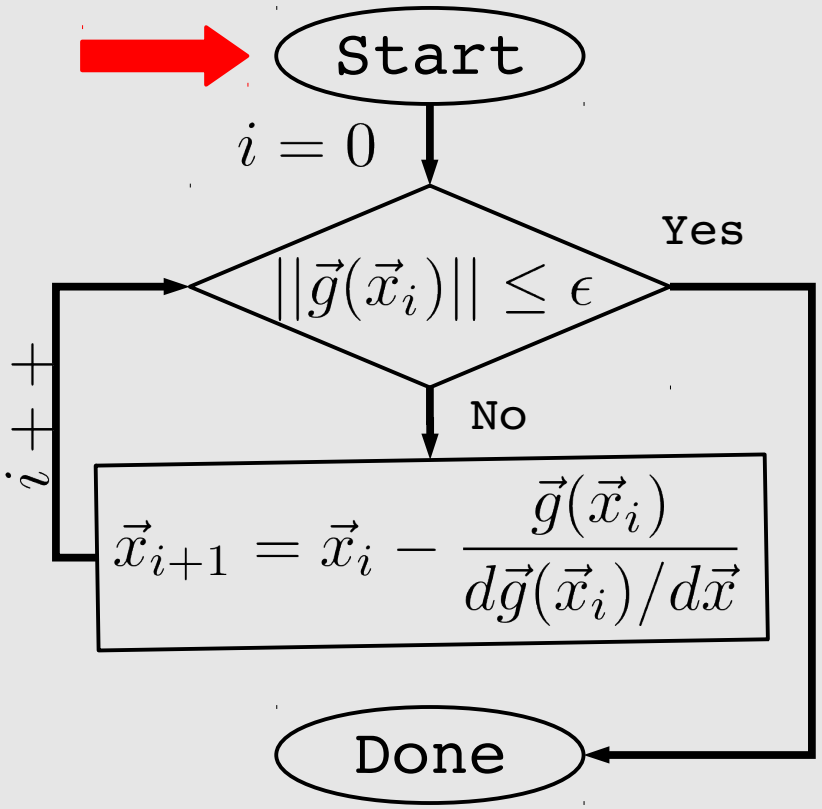


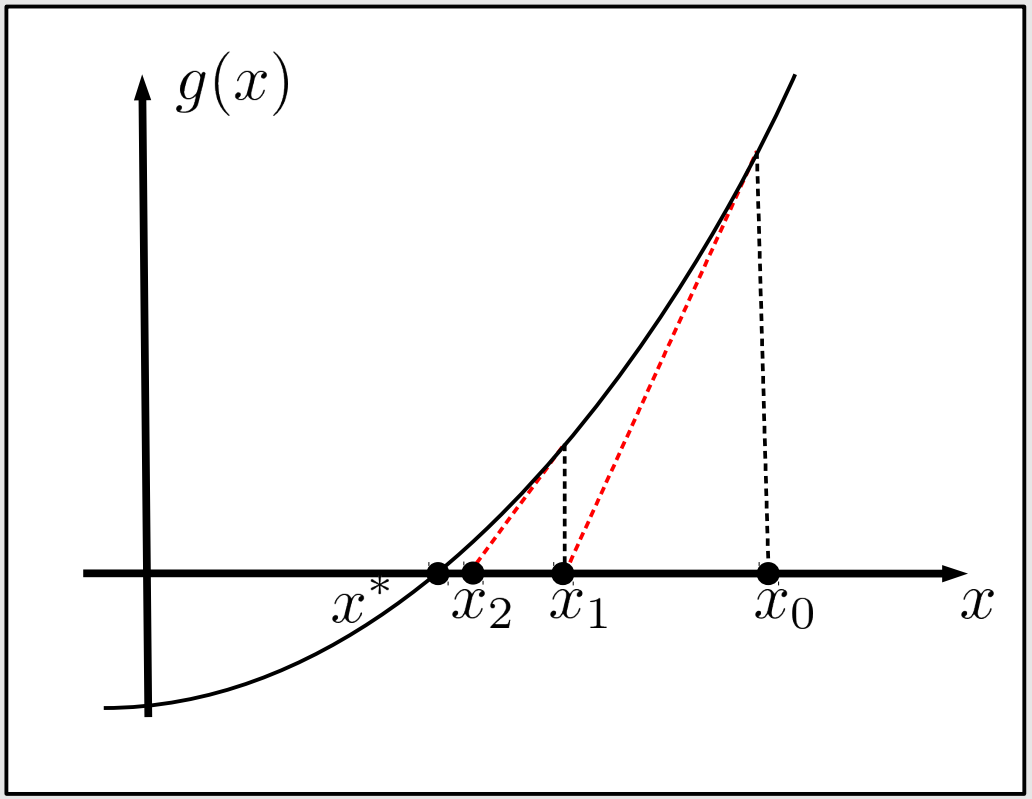
AC Analysis



The Newton Raphson Method

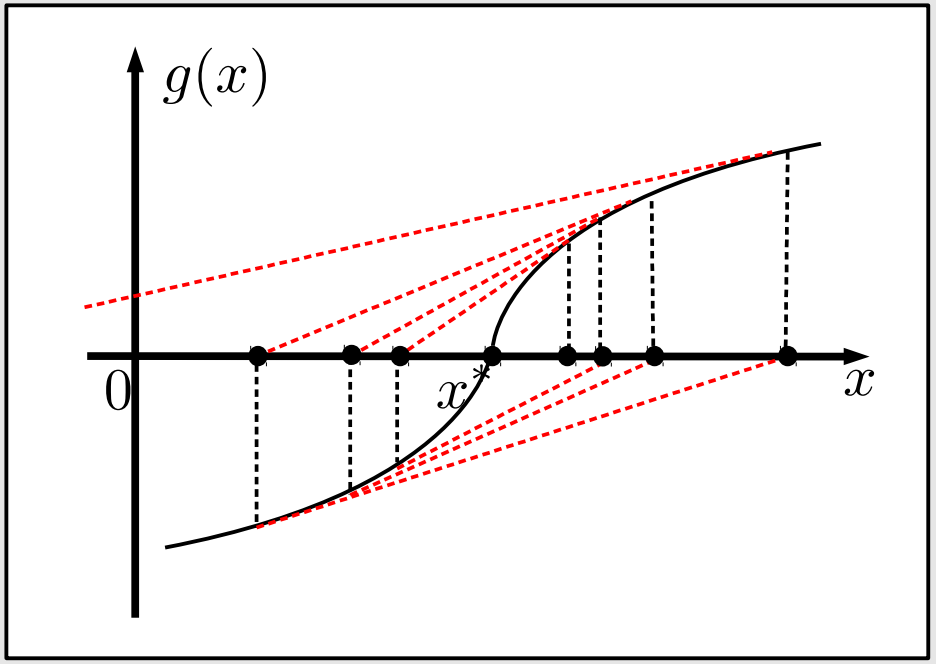
- **Iterative** numerical algorithm to solve $\vec{g}(\vec{x}) = \vec{0}$
  - Start with $\vec{x}_0$, update $\vec{x}_i$ with derivative information





NR: Convergence

- Conditions for NR to converge reliably
  - g(x) must be **"smooth"**: continuous, differentiable
  - starting guess must be "close enough" to solution
  - if not "close enough", NR is not guaranteed to converge



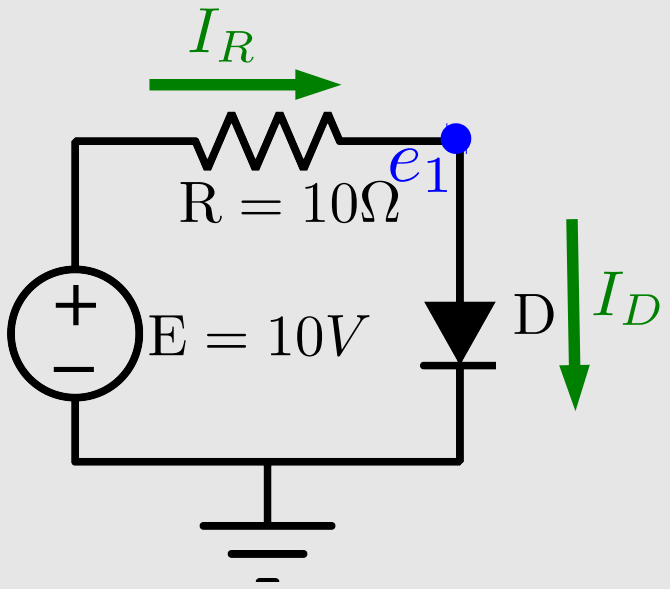
- However, there are things we can do to improve convergence:
  - **initialization and limiting**

Convergence Problems in a Simple Circuit

$$I_D = I_S\left(e^{\frac{e_1}{V_T}} - 1\right)$$

$$I_S \approx 1 \times 10^{-12}, V_T \approx 0.025$$

$$I_R = \frac{\mathrm{E} - e_1}{\mathrm{R}}$$



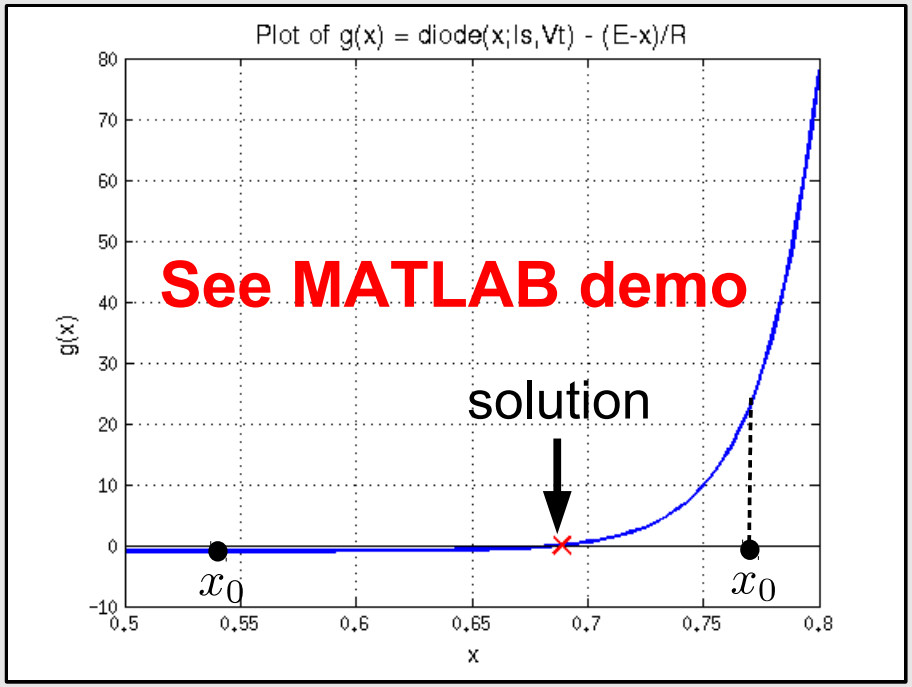
Circuit Equation: $I_D - I_R = 0$

$$\Longrightarrow \boxed{g(e_1) \triangleq I_S\left(e^{\frac{e_1}{V_T}} - 1\right) - \frac{\mathrm{E} - e_1}{\mathrm{R}} = 0}$$

Let $x = e_1$, draw $g(x)$ →

What will happen if x_0 is small?

NR fails for initial guess 0.5



NR: Initialization and Limiting

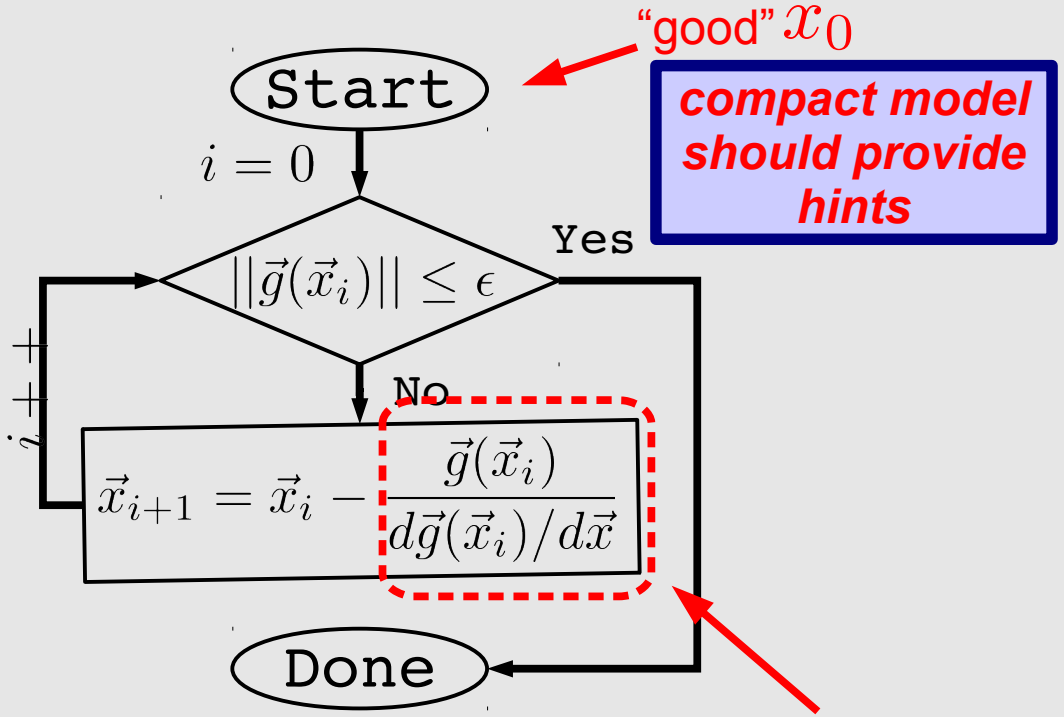
Initialization and limiting are crucial for NR success

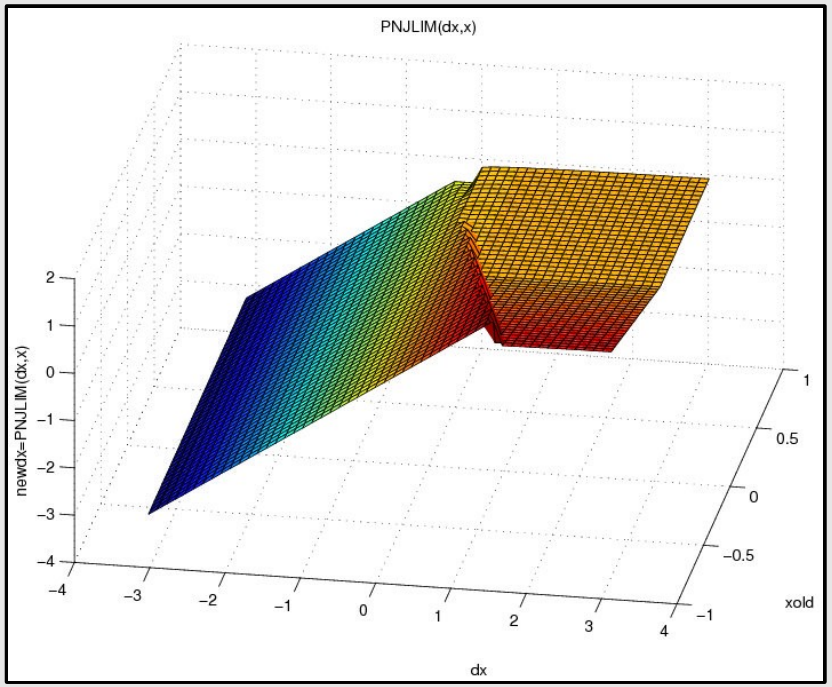
- Initialization: each "problematic" nonlinear device suggests initial branch voltage/current
  - e.g. diode, V(branch) ~0.7*V*

Compact model provides these

- Limiting: modify NR update step
  - e.g. diode, $\Delta V_d = \text{pnjlim}\left(\Delta V_d, V_d\right)$

See MATLAB demo





Compact Model Development Flows

Outline

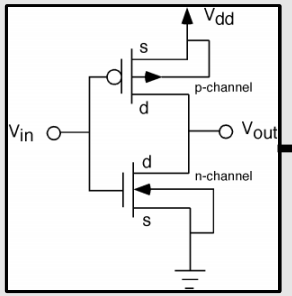
- What is a compact model?
  - writing device equations in a way useful for circuit design (via circuit simulation)
- What does one do with a compact model?
  - simulation (within a circuit simulator)
    - → how a simulator sets up equations
    - → how it solves the equations
    - → how a compact model can make simulation fail
      - how to work around it when designing a compact model
- **Flows for developing compact models**
  - **Berkeley SPICE (direct C code)**
  - **Verilog-A and commercial simulators**
  - **NEEDS-SPICE & ModSpec (MATLAB, Verilog-A)**

SPICE-based Model Development Flow

Device Simulation

takes months

not many can do this



Analytical Equations for Device

Write SPICE Model Code (in C)

Compile SPICE

Run Small Circuits in SPICE

tweak init/limiting

weeks/months

check derivatives

Debug code

modify

No

robust convergence?

Yes

Check/fix coding

No

Improve equations

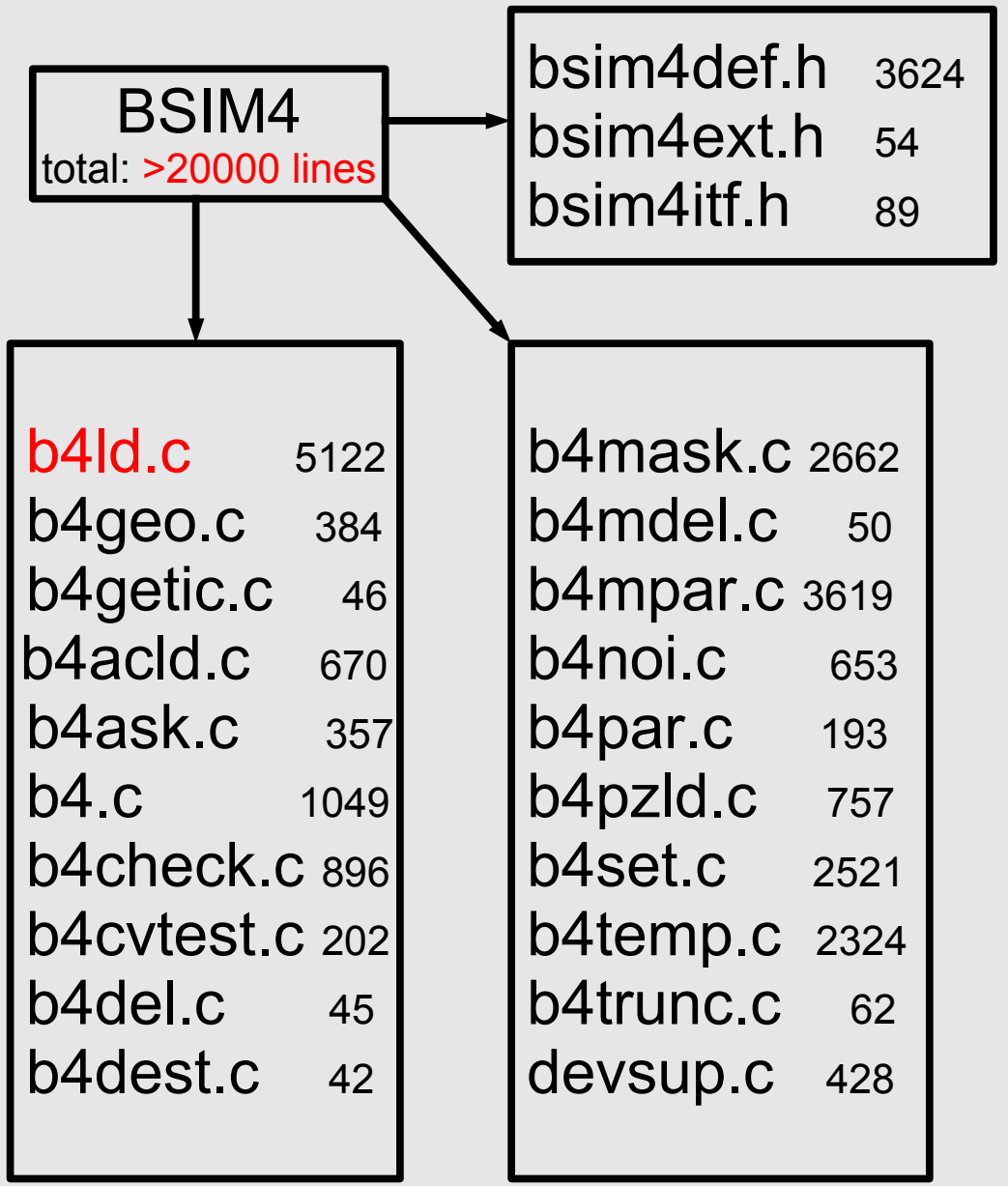
results OK?

Yes

Deploy model

Writing SPICE Model Code

- SPICE: **open source** and **freely available**
  - **can tweak init/limiting**
- **Model code is complex**
  - ➔ many files
- **Intricate dependencies**
  - ➔ easy to break
- **Derivatives done by hand**
  - ➔ error-prone and extremely tedious
- **DC/AC/TRAN analysis code inside model!**
  - must know SPICE algorithms
- *Typically, takes a PhD student years to deploy compact model*
  - *needs to learn sim. algorithms*
  - *needs to learn SPICE coding*
    - ➔ *poorly done; no documentation, ...*



DC/AC/TRAN code in Compact Model!
(SPICE diode code)

```
#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

Verilog-A-based Model Development Flow

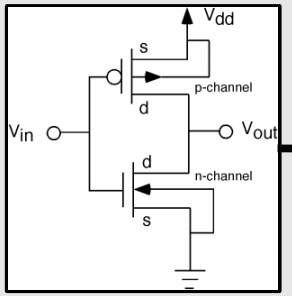
Device Simulation

Analytical Equations for Device

takes days or weeks

much easier than coding in SPICE

un-natural for many; confusing, dangerous constructs



Write Verilog-A Model

Run Circuits in **Commercial Simulators**

no ability to debug code

CANNOT TWEAK init/limiting (support obscure/missing)

X

No

robust convergence?

trade off physics for convergence

Yes

improve equations

trade off physics for convergence

Check/fix coding

No

results OK?

inadequate V-A support in open-source simulators

hand-translated+tweaked "native" implementation needed for efficiency

Yes

slow deploy model

NEEDS-SPICE Model Development Flow

Device Simulation

Analytical Equations for Device

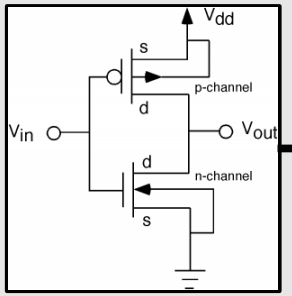
takes hours or days

much more natural than Verilog-A (once learned)

Write ModSpec Model (in MATLAB)

Test immediately *(standalone)*

debug/improve



DC/AC/TRAN in MATLAB

Run Small Circuits in **MDE**
(Model Development Environment)

code available for inspection and debugging

add/tweak init/limiting functions

robust convergence?

No

Yes

check/fix coding
easier/faster in MATLAB

init/limiting → easier physics vs convergence tradeoff

improve equations

init/limiting → easier physics vs convergence tradeoff

results OK?

No

Yes

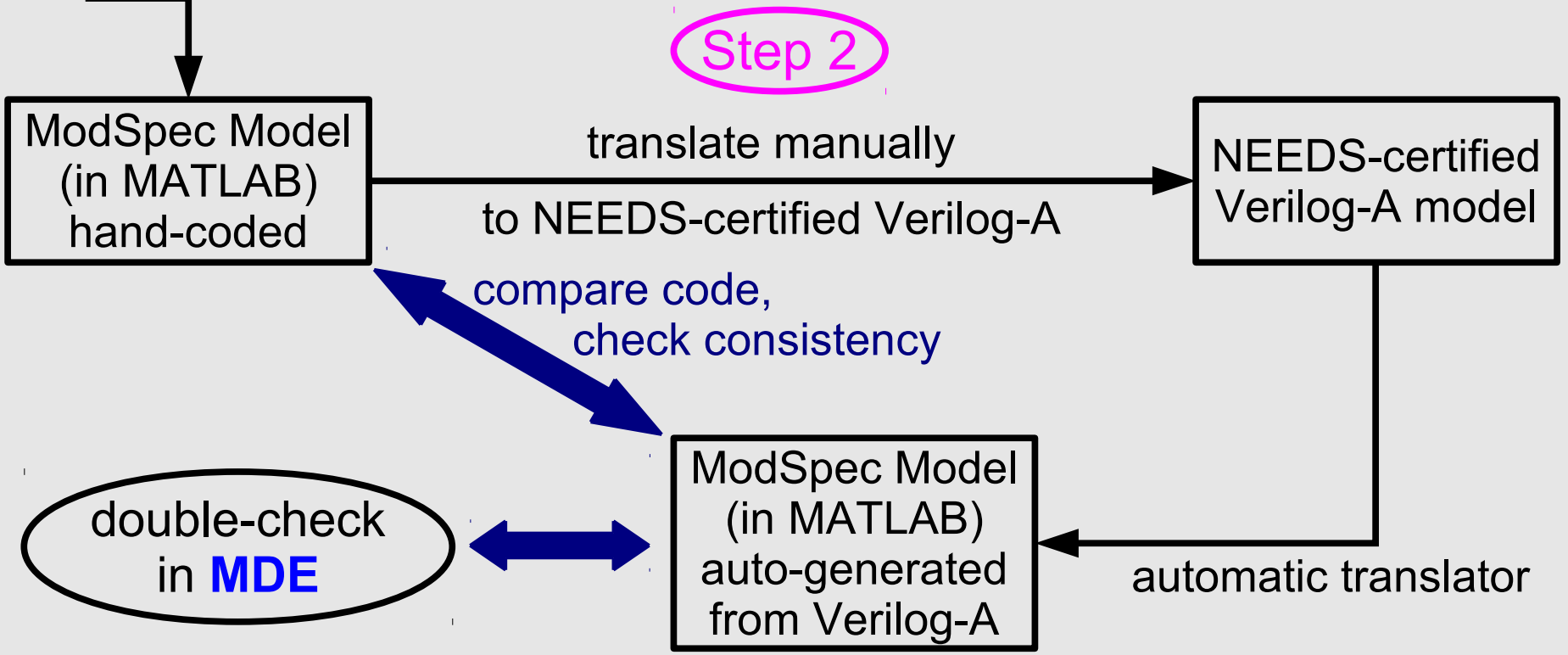
go to next step in flow

satisfactory ModSpec model

NEEDS-SPICE Model Development Flow (2)

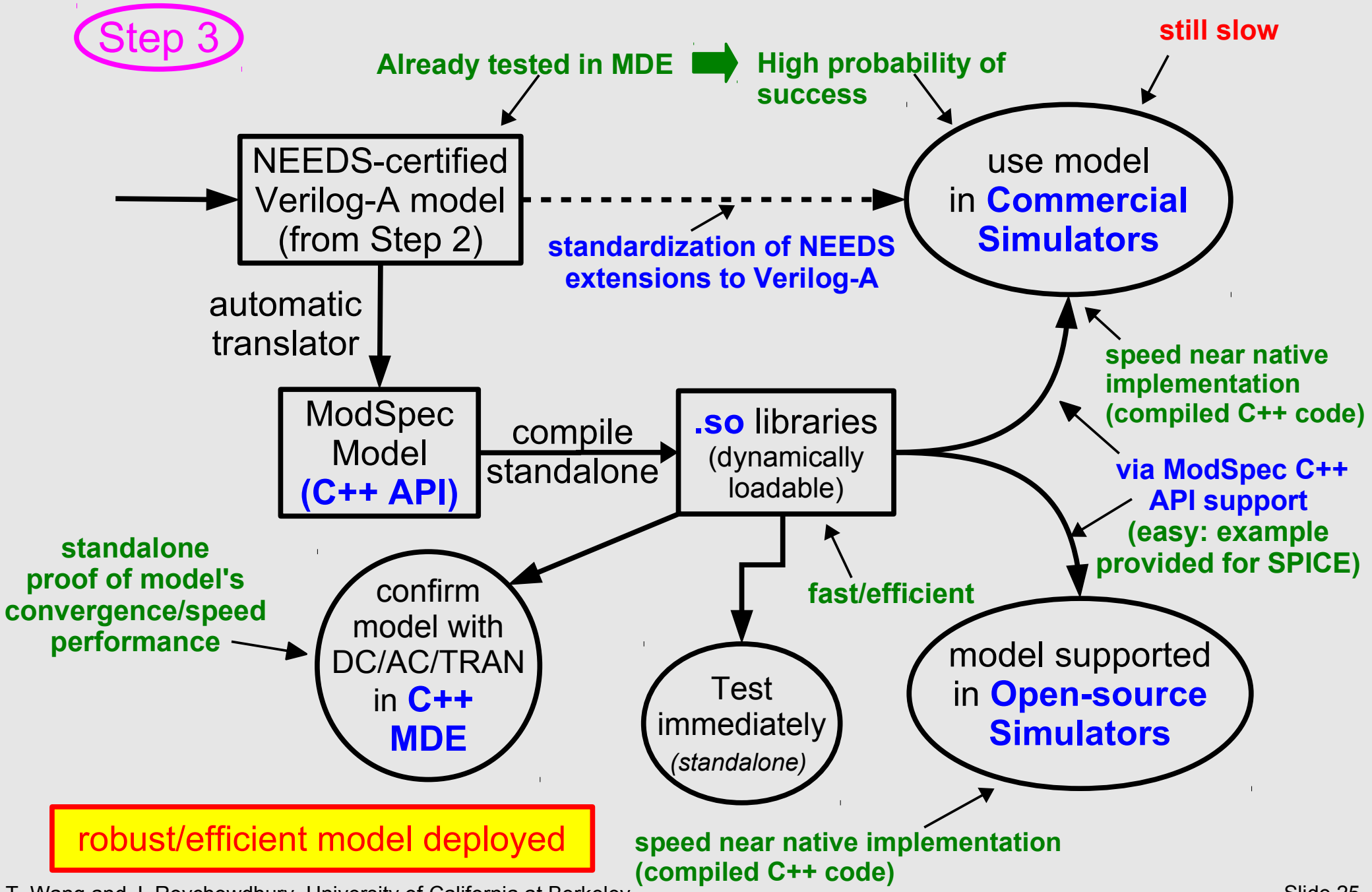
Step 1: model developed in ModSpec/MDE

- Takes hours or days
- All in **MATLAB**
- High level of confidence that model works in simulation
  - → because you have run DC/AC/TRAN in MDE



end of Step 2: high level of confidence Verilog-A model is correct/debugged

NEEDS-SPICE Model Development Flow (3)



Summary and Conclusion