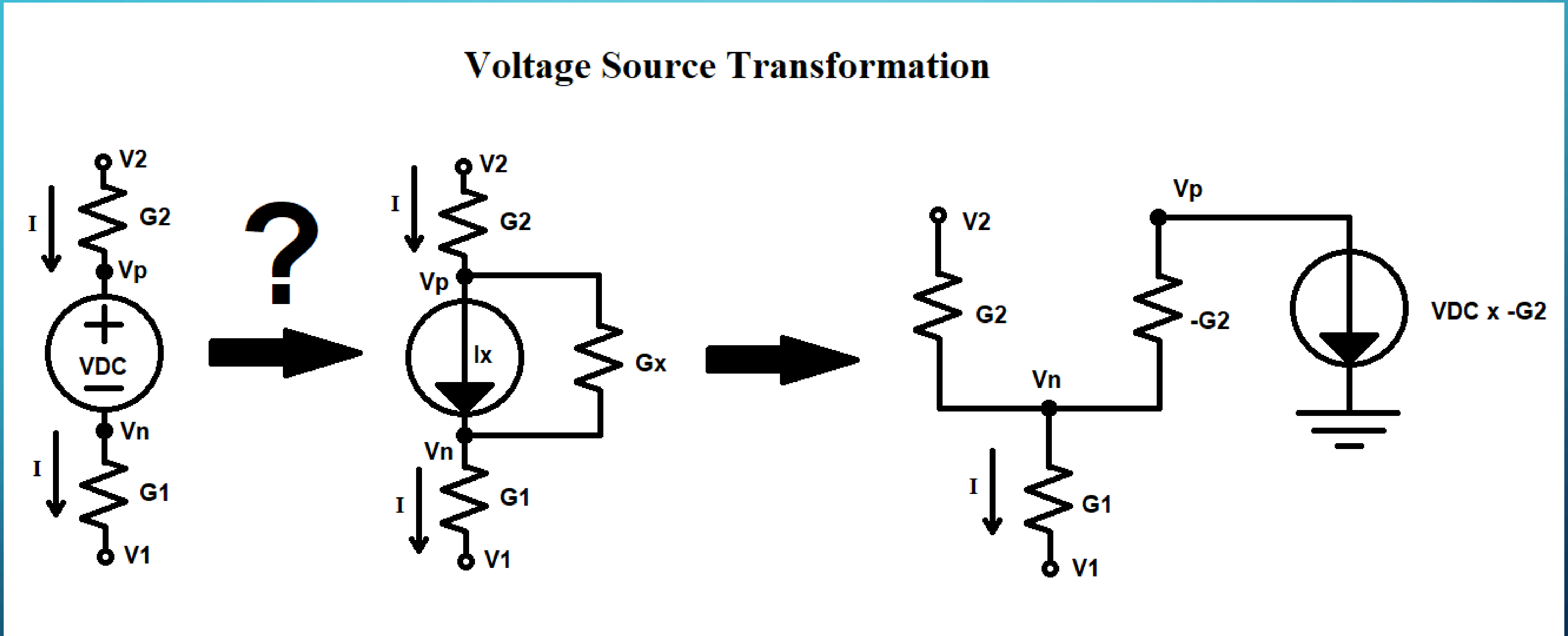
CUDA SPICE CIRCUIT SIMULATOR

MILESTONE 2

ANGELINA RISI, EE 2018

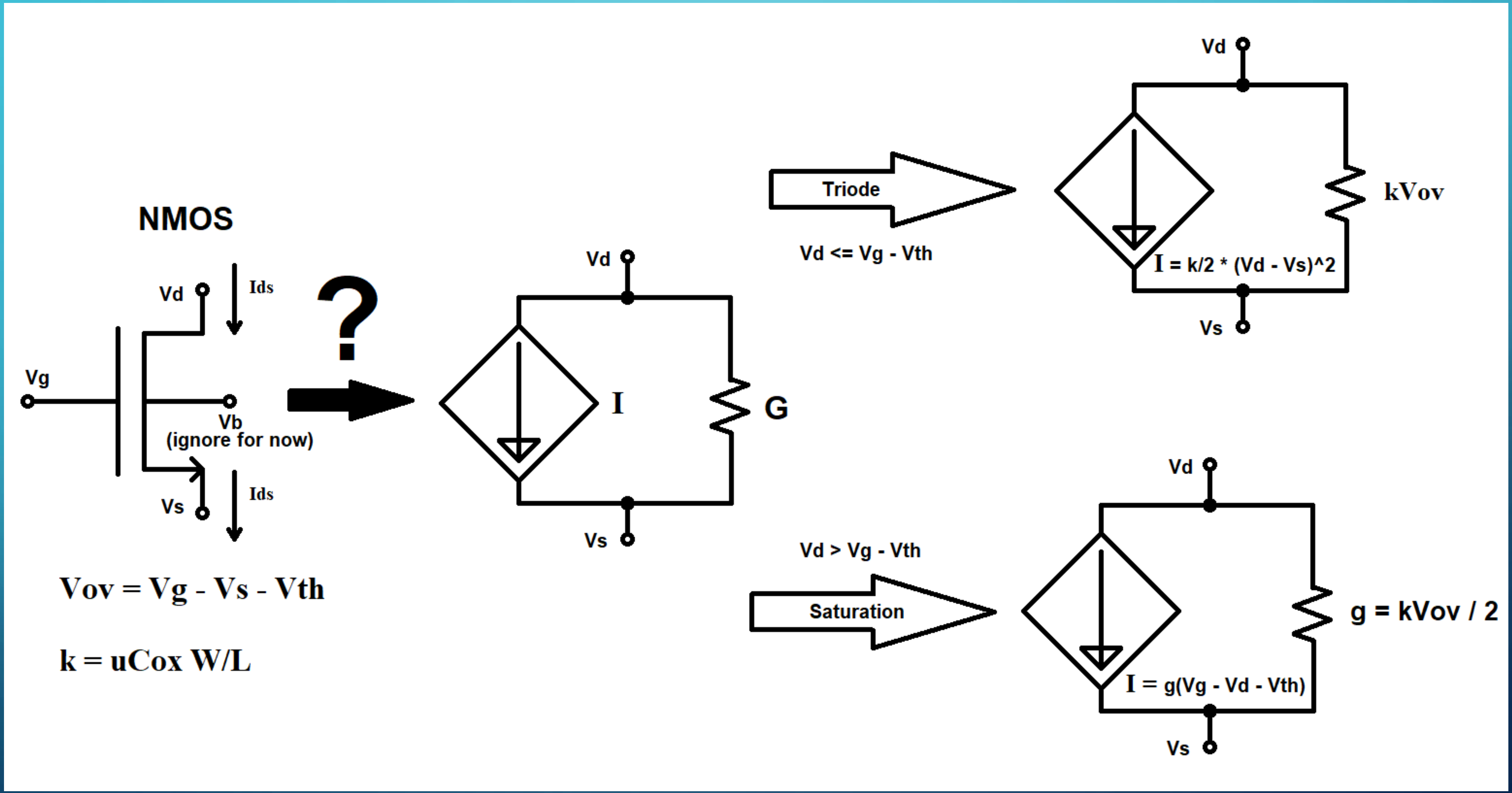
PROGRESS SINCE MILESTONE 1:

- Code cleanup: overhaul on how we handle circuit elements
  - Instead of separate structs and lists for each type of element, all passives are the same Element with node and parameter arrays (instead of hardcoding how many they have)
    - Other than voltage sources, all passives can be stored on the same list (because VDC needs to be applied last to matrices for reasons described below
  - Will need further modification to properly port netlist to GPU (currently using std::vectors for dynamic arrays)
- More edge case testing to locate and correct bugs
  - Located a problem with VDC modelling (problem with conversion to I and G when neither node grounded)
    - Before: Used the fact that the currents flowing out of both positive and negative node are equal to generate G and I matrix entries
    - Problem: Even though Vp = Vn + Vdc used to replace on of the voltages in one of the equations to assert Vdc dependence, still both voltages dependent on G seen at each node
    - Solution: seems really obvious, but Vp – Vn = Vdc, => Vp(Gx) – Vn(Gx) = Vdc(Gx) => divide by Gx. So, I use the previous method to populate the matrix for Vn but this new equation for Vp


Voltage Source Transformation
V2
I
G2
Vp
VDC
Vn
I
G1
V1
?
V2
I
G2
Vp
Ix
Gx
Vn
I
G1
V1
V2
G2
Vp
-G2
VDC x -G2
Vn
I
G1
V1

PROGRESS SINCE MILESTONE 1: TRANSISTORS

- Added transistor parsing, not including any extra geometry/layout parameters
- Currently use a default hard-coded MOSFET models, need to add model parsing
- Transistor linearization to G and I elements
- Added channel-length modulation parameter
  - Adds dependence on drain-source voltage in saturation
  - Otherwise, for example for a simple inverter w/ Vdd/2 input there are 3 different solutions possible and it converges to one of those, not necessarily the one we want
- Solution of circuits including transistors by:
  - Using previous solution as “Guess” voltages to linearize model
  - Iterating solving circuit until new solution and previous guess converge to within some absolute tolerance (currently 1µV)
  - This implicitly performs Newton-Raphson method
  - Currently very serial loop method, only matrix solving on GPU



NMOS
Vd
Ids
Vg
Vb
(ignore for now)
Vs
Ids
?
Vd
I
G
Vs
Vov = Vg - Vs - Vth
k = uCox W/L
Triode
Vd <= Vg - Vth
Vd
kVov
I = k/2 * (Vd - Vs)^2
Vs
Vd > Vg - Vth
Saturation
Vd
g = kVov / 2
I = g(Vg - Vd - Vth)
Vs

CURRENTLY WORKING ON:

- Model parsing and multiline content parsing
  - Models generally specified in separate .incude files, but can be in netlist file, .model line
  - Parameters spilling into multiple lines start new line with '+' character, need to look ahead to next line
- DC Sweep – already have idea of how to setup
  - Find swept element, save original parameter value, overwrite with start value
  - Solve, increment value by step size, repeat until stop value reached
- Porting matrix setup to GPU:
  - Parallelize by elements
  - Atomics for race conditions between elements on same node
  - Current issue with this is passing netlist to GPU, needs modifications

NEXT MILESTONE:

- Transient (Time step) Simulation
  - Add in capacitors, inductors, AC sources, etc. (time-dependent elements)
- More second-order effects (improved transistor modelling)
- Graphical interface or data export and plotting
- Performance optimizations & comparison
- Stream compaction of matrices (expected to be sparse in large circuits)
  - For better memory usage, as ultimate goal is to be useful for VLSI