FAMU-FSU College of Engineering

Towards a More General Model of Reversible Logic Hardware

Invited talk presented Mar. 16th, 2012 at the *Superconducting Electronics Approaching the Landauer Limit and Reversibility (SEALeR)* workshop

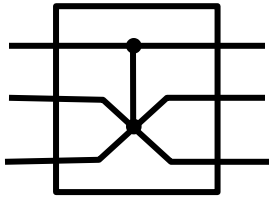
Michael P. Frank

Dept. of Elec. & Comp. Eng., FAMU-FSU College of Engineering
& Dept. of Physics, Florida A&M University

A Simple Question

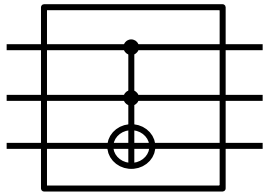
What is the simplest & most general universal set of primitive digital elements for implementing (adiabatic) reversible computing in CMOS?

A1: Fredkin gates (cSWAP)?



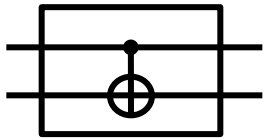
* Restricted to conservative logic (or dual-rail)

A2: Toffoli gates (ccNOT)?



* Still has six I/O terminals (3 input/3 output)

A3: cNOT?

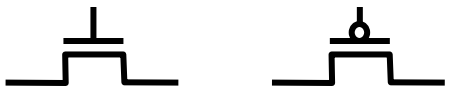


* Simpler, but not universal w/o also quantum gates

Also, all the above elements consume all of their inputs, & emit an equal number of outputs each time they are applied…

Is that truly the most general framework?

A4: CMOS (nFET & pFET) transistors!



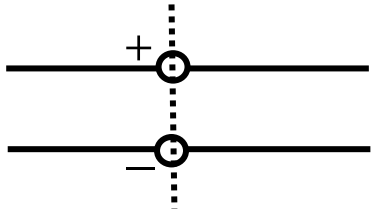
* Instances of a more general class of elements for reversible computing.

Transistors as Reversible Elements

- A (field-effect) transistor has 3 terminals only:
  - 1 input-only terminal (gate)
    - This input affects (but is not consumed by) the device
  - 2 bidirectional terminals (source/drain)
    - Generally these can act as inputs, or outputs, or both!
  - Actually, there is also a 4th (body) terminal
    - But we can ignore it for our present purposes
- Obviously, the operation of a transistor is not *always* reversible…
  - But, we'll see, it can be *conditionally* reversible
    - Under certain preconditions that we can define.
      - & this is sufficient for building any (classical) reversible digital computational functionality that we can imagine!

High-Impedance (Z) States

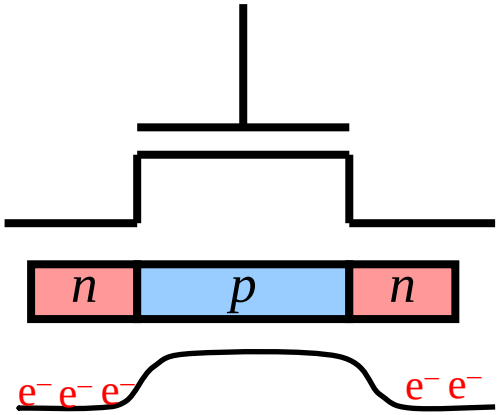
- In general, a given physical I/O terminal between devices does *not* need to *always* be supplying a bias from one side to the other.
  - Another option: The terminal can be configured (on either, or both sides) as an open circuit.
    - No voltage sourced / no current sunk



- 'Z' states are frequently used in digital design!
  - Bidirectional I/O ports
  - Shared buses
  - Dynamic logic families, dynamic RAM

FET Potential Energy Surface

- A FET provides a controllable potential energy surface for charge carriers, very similar to Landauer's bistable potential well model.
  - For an n-type FET:
    - Raise gate voltage → lower potential energy barrier for electrons to pass between source and drain terminals.
      - In pFET: effect is opposite, and for holes
  - Apply bias voltage between source & drain terminals to "tilt" the potential energy surface

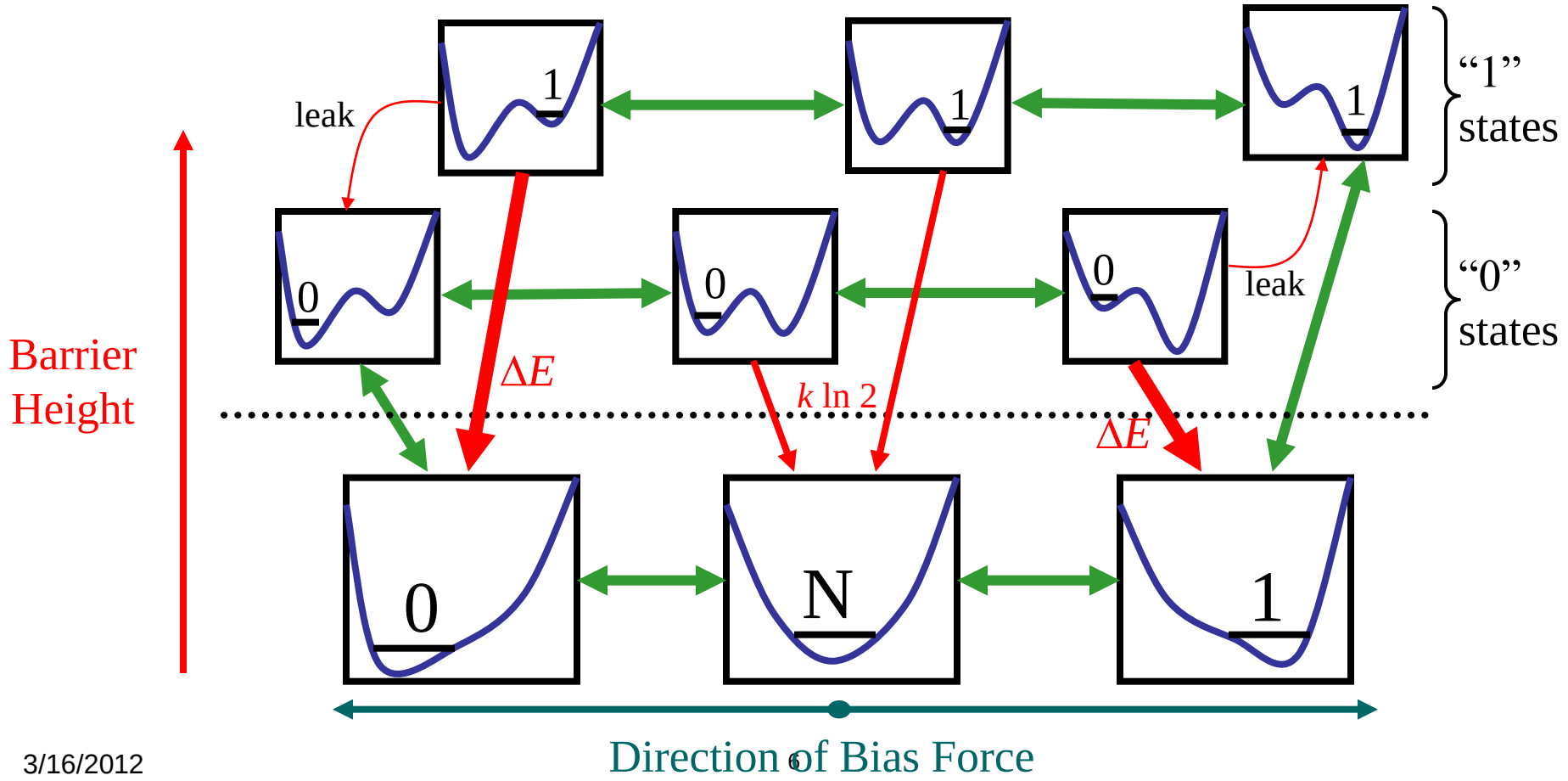


- Off transistor = open circuit
  - High-*Z* terminal (as seen from either side)

Possible Well Transitions

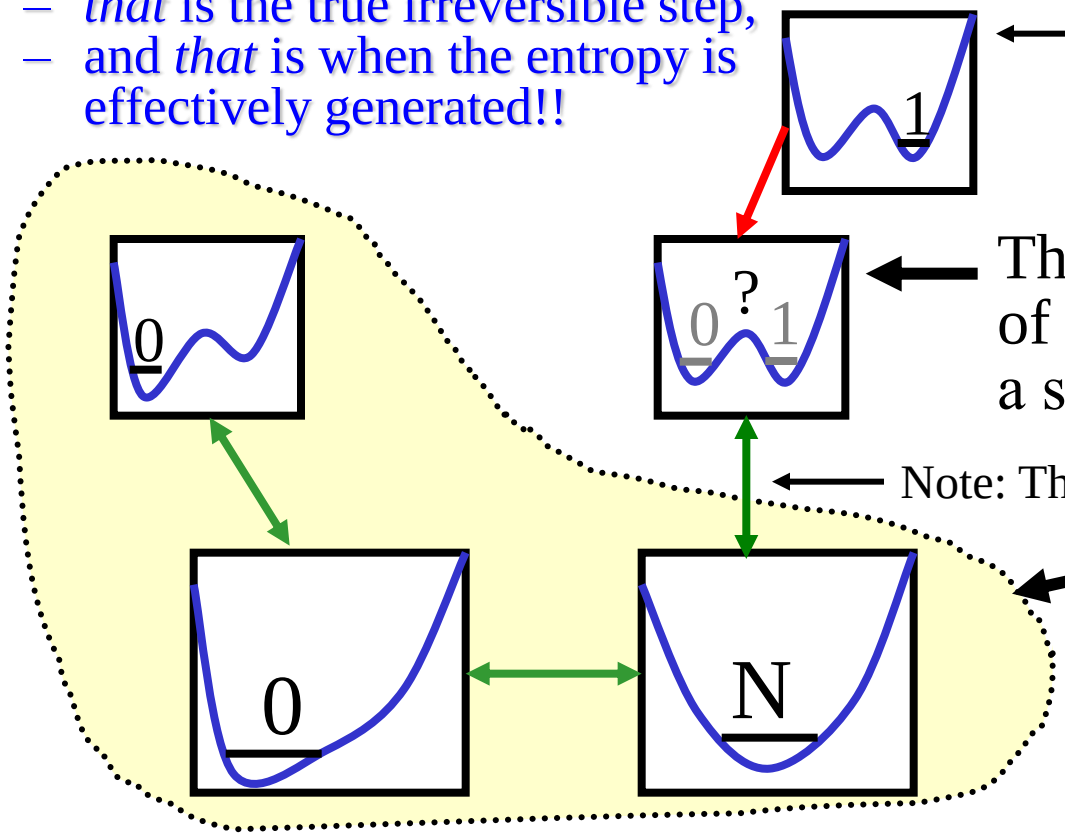
- Catalog of all the possible transitions in the bistable wells, adiabatic & not...
  - We can characterize a wide variety of digital logic and memory styles in terms of how their operation corresponds to subgraphs of this diagram.

(Ignoring superposition states.)



Erasing Digital *Entropy*

- Note that if the information in a bit-system is *already* entropy,
  - Then erasing it just *moves* this entropy to the surroundings.
  - This can be done with a thermodynamically reversible process, and does *not* necessarily *increase* total entropy!
- However, if/when we take a bit that is known, and irrevocably commit ourselves to thereafter treating it as if it were *unknown*,
  - *that* is the true irreversible step,
  - and *that* is when the entropy is effectively generated!!



This state contains 1 bit of decomputable information, in a stable, "digital" form

This state contains 1 bit of physical entropy, but in a stable, "digital" form

Note: This transformation is reversible!!

In these 3 states, there is *no* entropy in the digital state; it has all been pushed out into the environment.

Logic & Memory Styles

All describable within the potential-well paradigm!

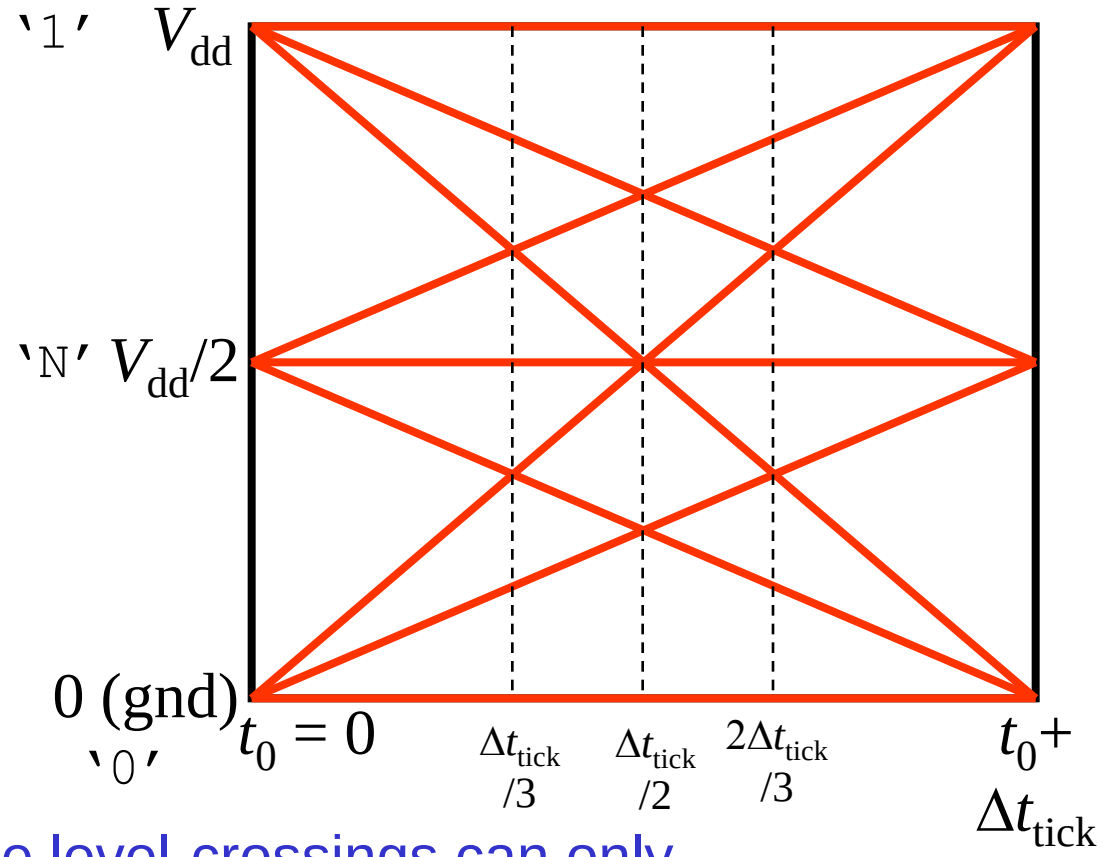
- Irreversible styles:
  - *Input-barrier*, *constant-bias* logic.
    - *E.g.* standard static CMOS inverters & combinational gates.
  - *Input-bias, clocked-barrier* latching.
    - Standard static CMOS latches, dynamic RAM cells, *etc*.
- Reversible styles:
  - **Type 1:** *Input-bias, clocked-barrier* latching.
  - **Type 2:** *Input-barrier, clocked-bias* logic.
  - **Type 3:** *Input-barrier, clocked-bias* latching logic.
- All of these are available in a very wide variety of different physical instantiations of the bistable well.
  - *E.g.*, CMOS, superconducting, quantum-dot, Y-branch switches, mechanical implementations, *etc.*

Rules for (Asymptotically) Reversible Operation of a FET

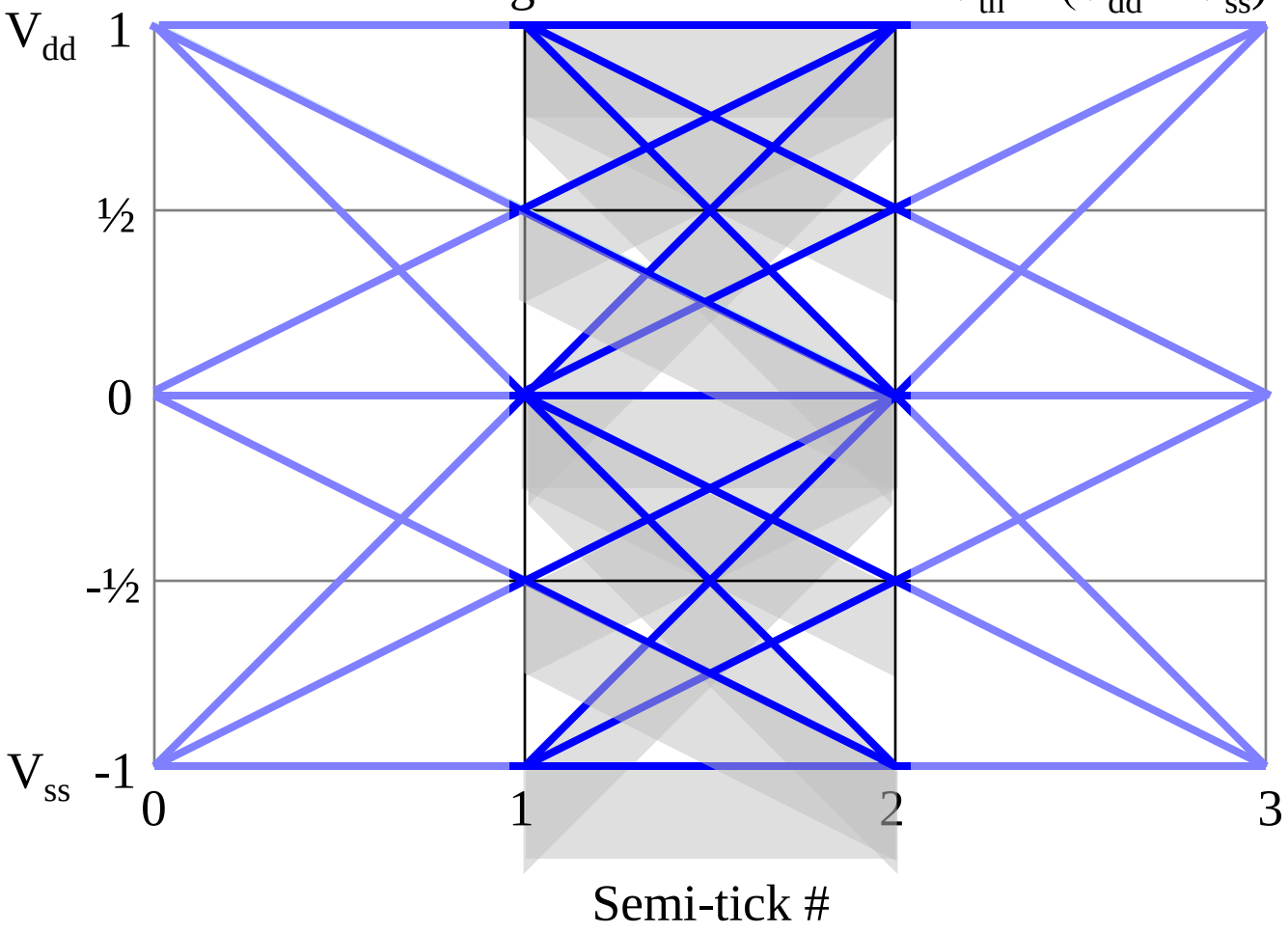
- **Rule 1:** Never turn *on* a transistor when there is a (non-negligible) *voltage* between source & drain terminals.
  - Leads to sudden order-CV^2 losses.
- **Rule 2:** (Obviously) never apply different biases simultaneously to source & drain of an "on" transistor.
  - Causes a (dissipative) short-circuit current across the device.
- **Rule 3:** Never change the voltage applied to any given terminal too rapidly (relative to *RC* of signal path)…
  - Especially to the source or drain of an "on" transistor!
    - Keeps the V_{DS} voltage drop (and the Q^2R/t losses) small
- **Rule 4:** Never turn *off* a transistor when there is a (non-negligible) *current* between source & drain terminals.
  - Exception: If there's an alternate path between the same nodes!
  - Several early "adiabatic" logic families unwittingly failed to obey this rule → not truly/fully adiabatic!

Finding Safe Adiabatic Transitions

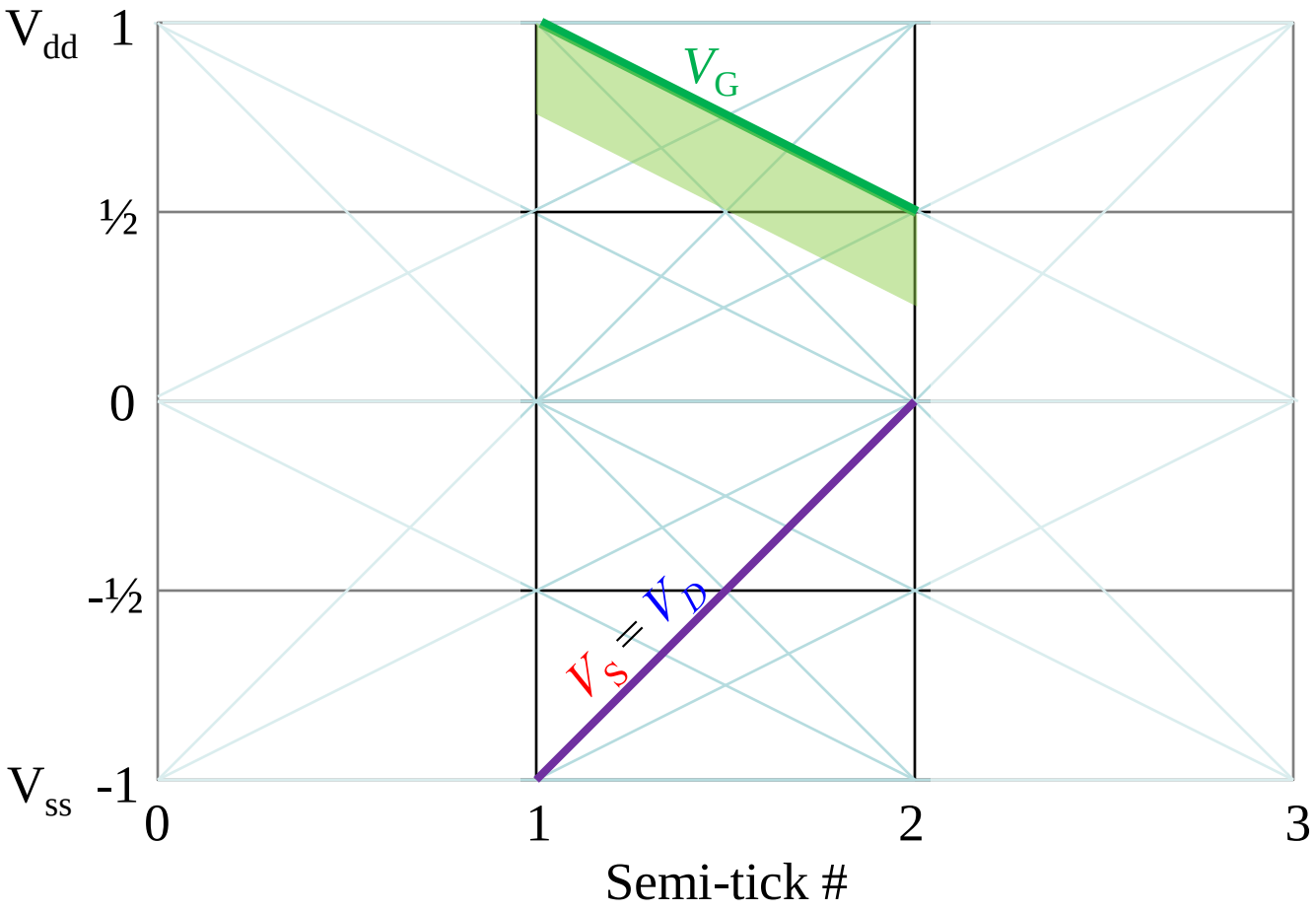
- Given a set of logic levels,
  - And linear ramps between them,
- We can determine which combinations of gate/source/drain transitions are adiabatic.
  - Made easier because level-crossings can only occur at certain discrete times.
- A discrete circuit simulator was developed at UF that checks arbitrary 3-level circuits for full adiabaticity.



Possible Linear Transitions in a Semi-Tick
for Three-Level Logic at an n-FET with $V_{th} \approx (V_{dd} - V_{ss})/8$

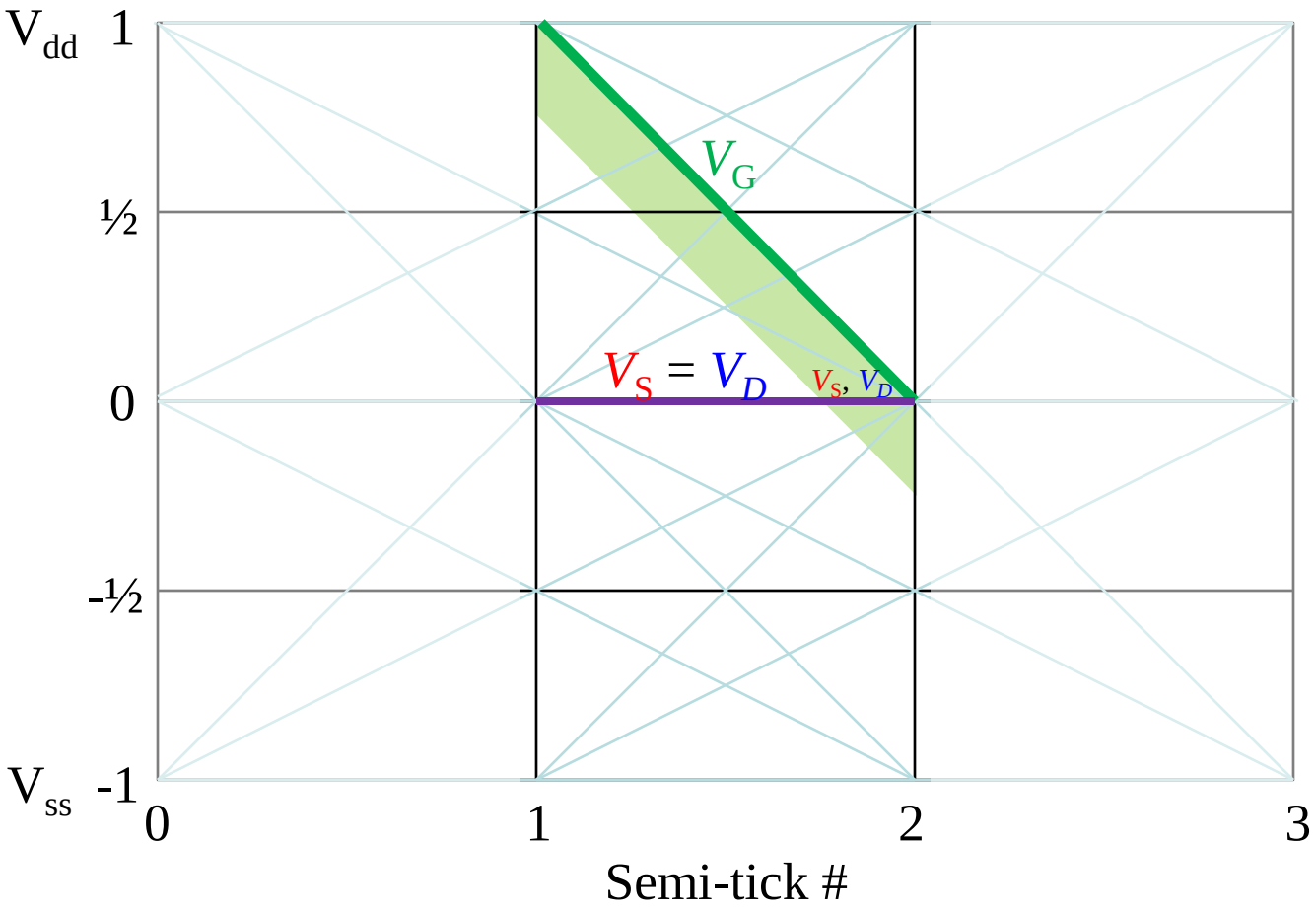


Examples of Adiabatic Transitions: Example #1



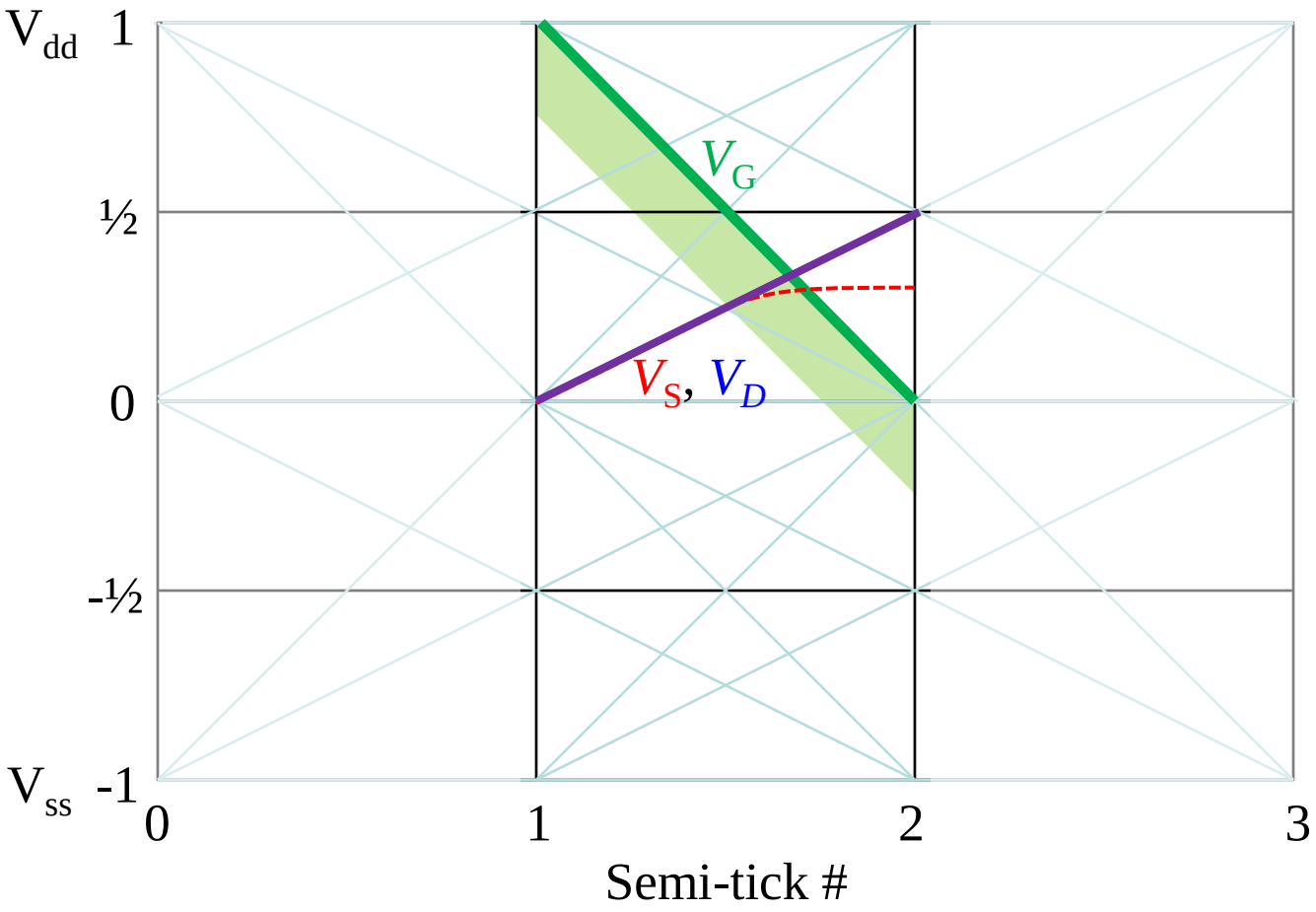
(Always adiabatic – Source & drain remain connected throughout transition)

Examples of Adiabatic Transitions: Example #2



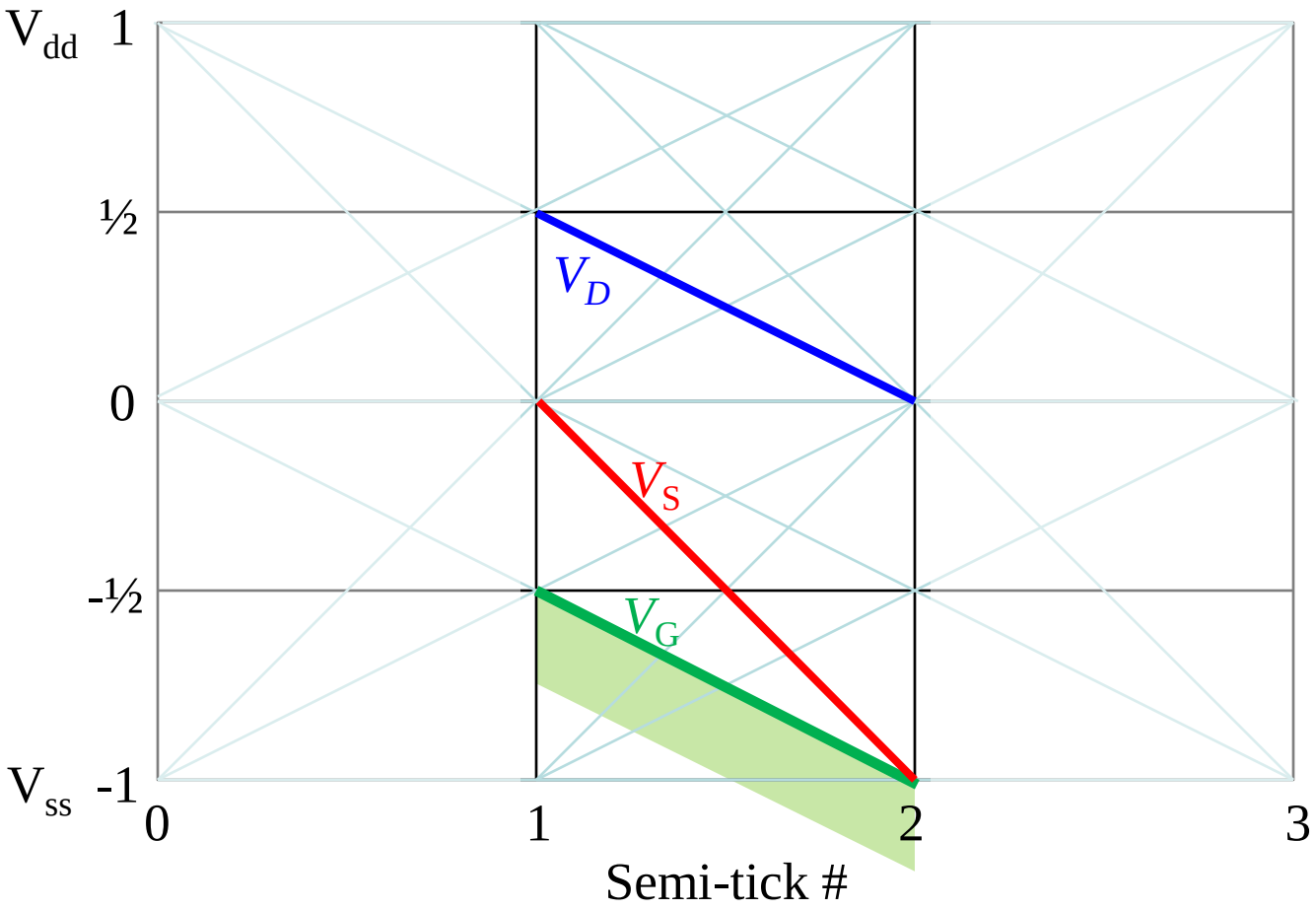
(Always adiabatic – Source & drain become disconnected, but only while $I_{DS} = 0$)

Examples of Adiabatic Transitions: Example #3



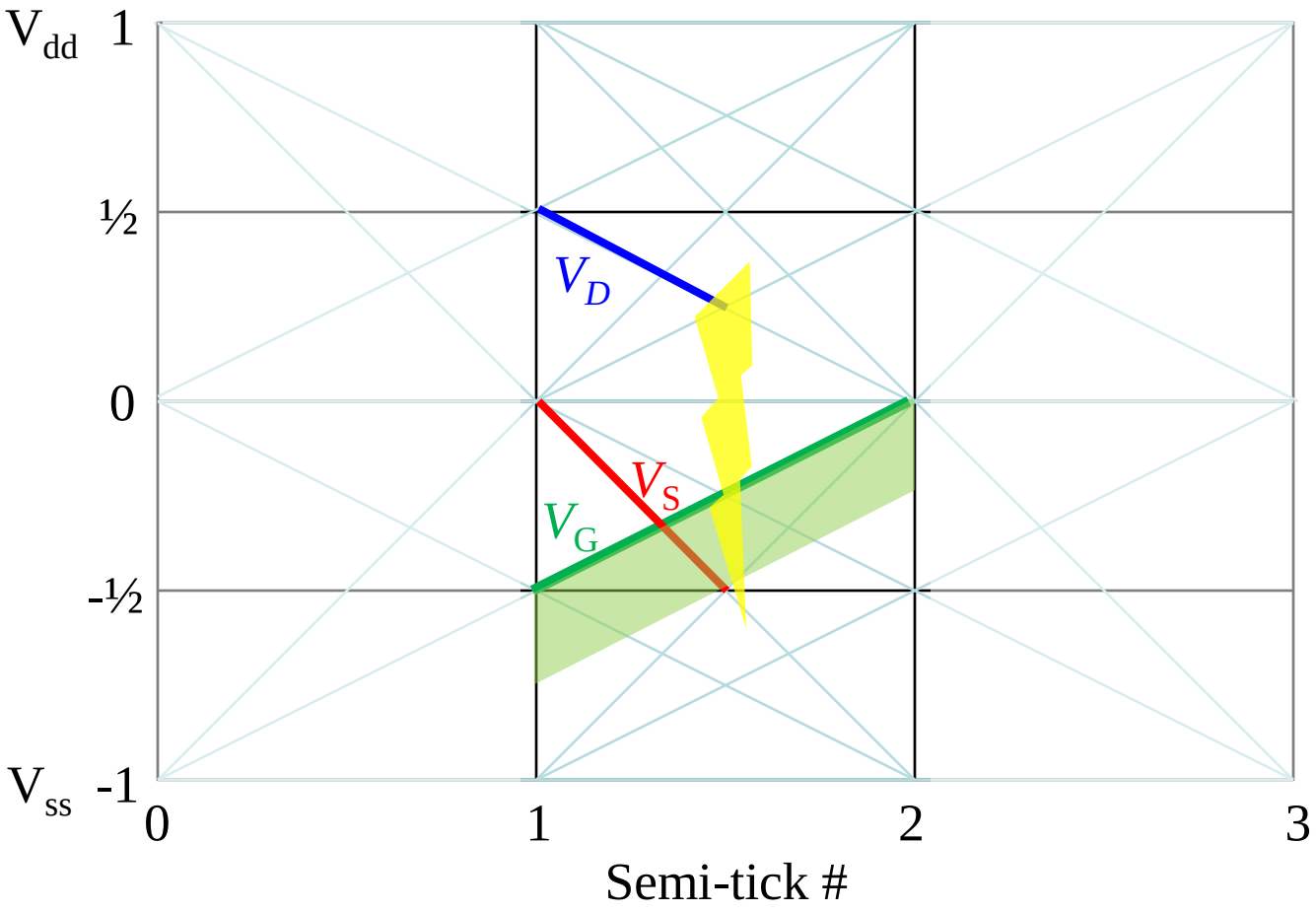
(Possibly adiabatic – But only if source & drain are *separately* being driven along identical trajectories)

Examples of Adiabatic Transitions: Example #3



(Always adiabatic – Source & drain disconnected throughout transition)

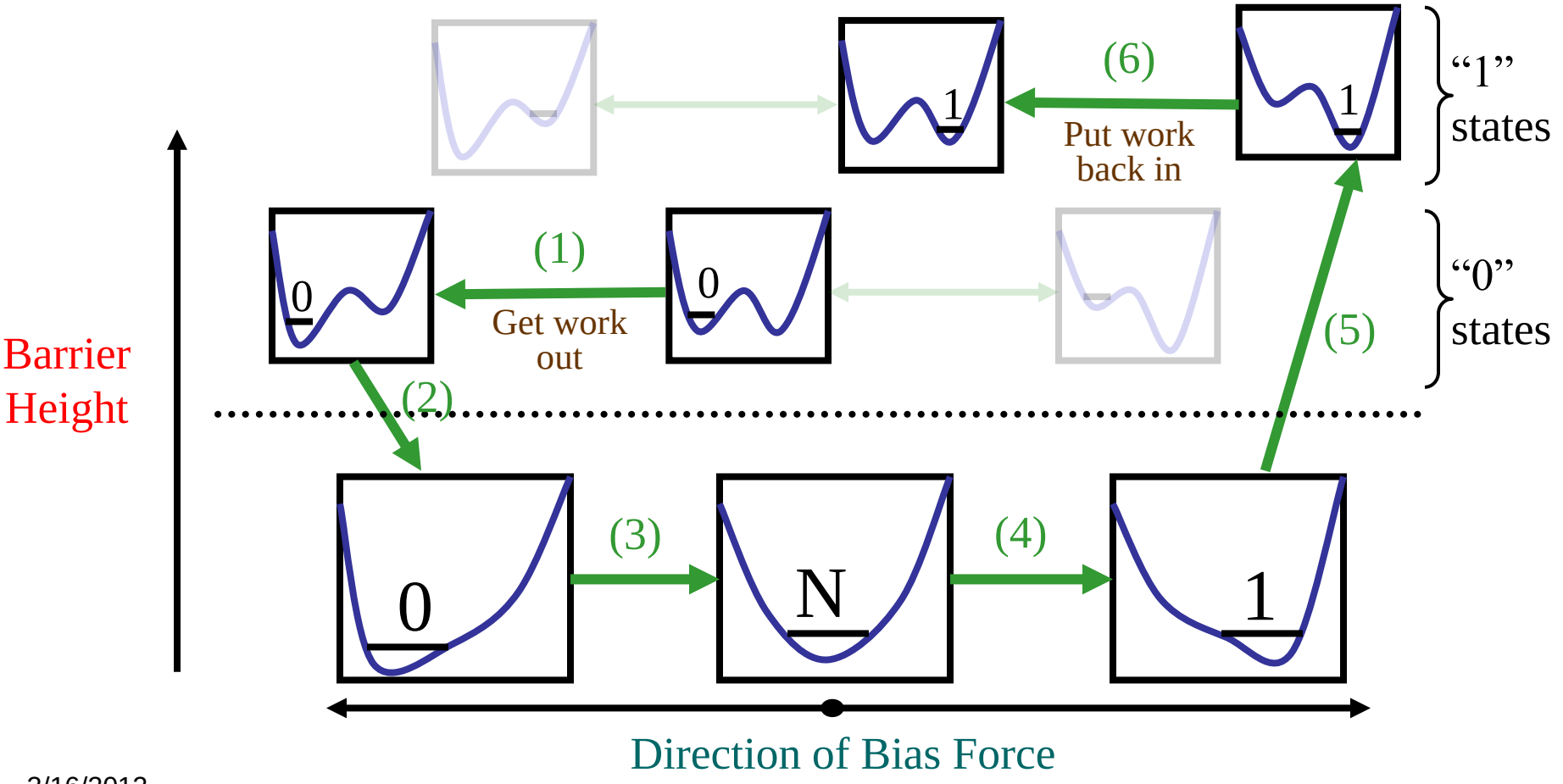
Examples of **Non-**Adiabatic Transitions: Example #1



(Never adiabatic – Transistor turns on when $V_{DS} \neq 0$)

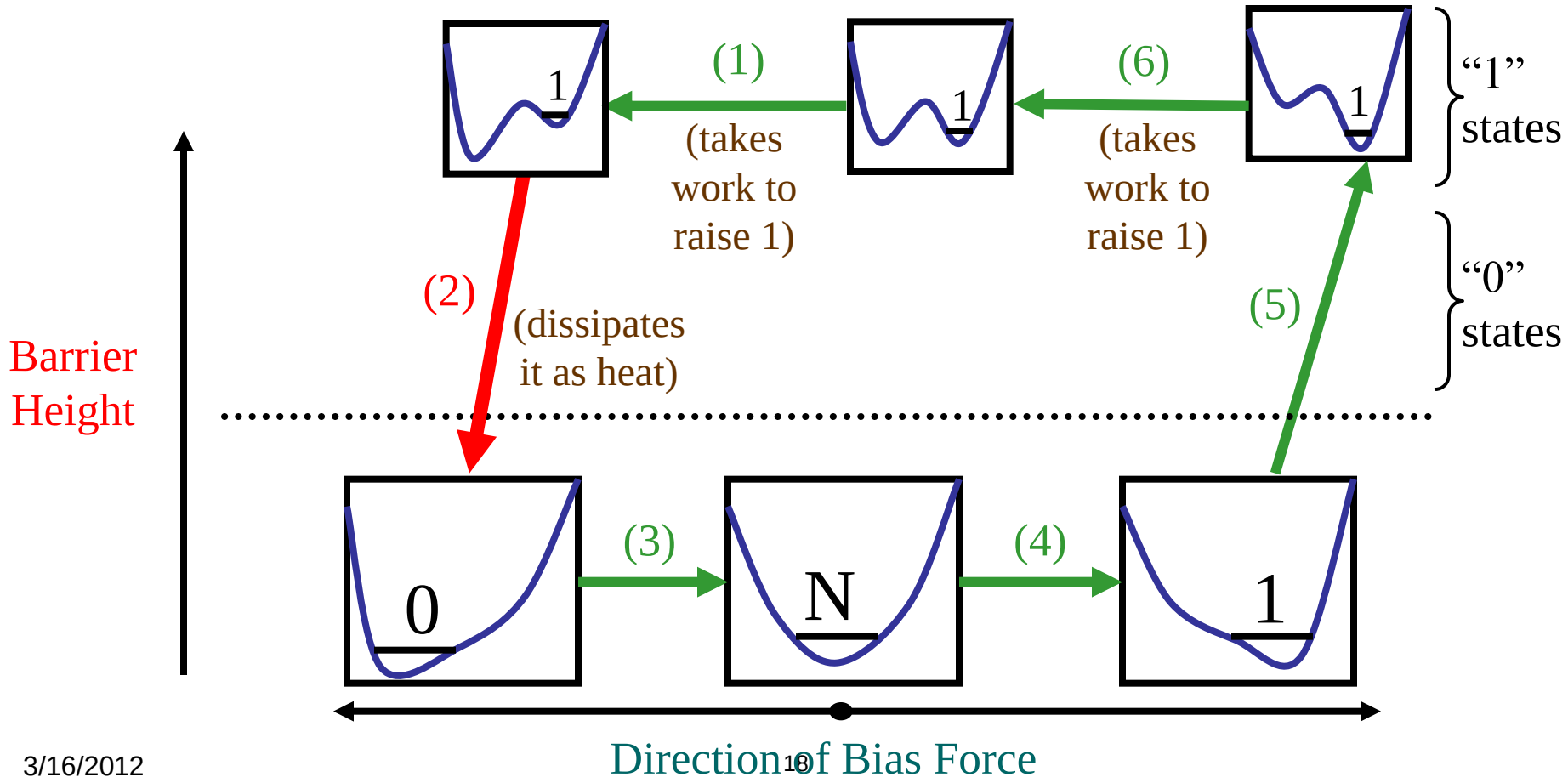
Reversible Set (`rSET`) & Clear (`rCLR`)

- **`rSET`** operation semantics: Given assurance that a bit is initially 0, unconditionally change it to 1.
  - To implement: Traverse the adiabat (reversible trajectory) shown below.
- Reverse this path to perform **`rCLR`**.



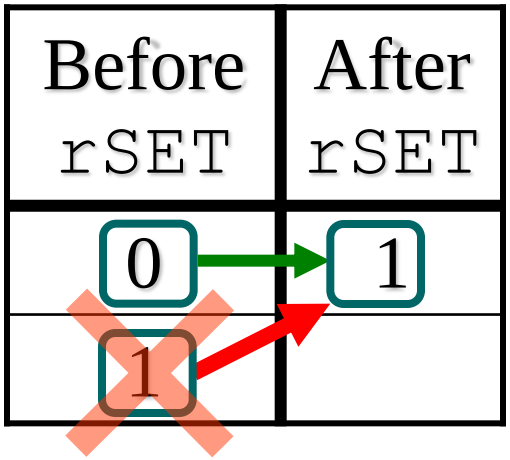
Taking `rSET`/`rCLR` out of context

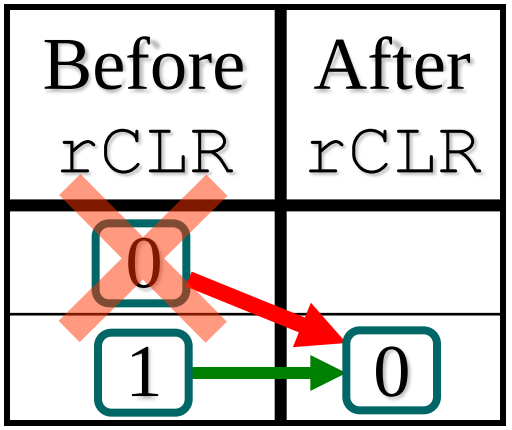
- What happens if we attempt to perform `rSET` on a bit that is already a 1?
  - It still ends up with the right value (1), but…
  - Irreversible dissipation occurs in step 2 (when barrier is lowered), as shown below.
- Similarly if we try to `rCLR` a 0.



rSET/rCLR transition tables

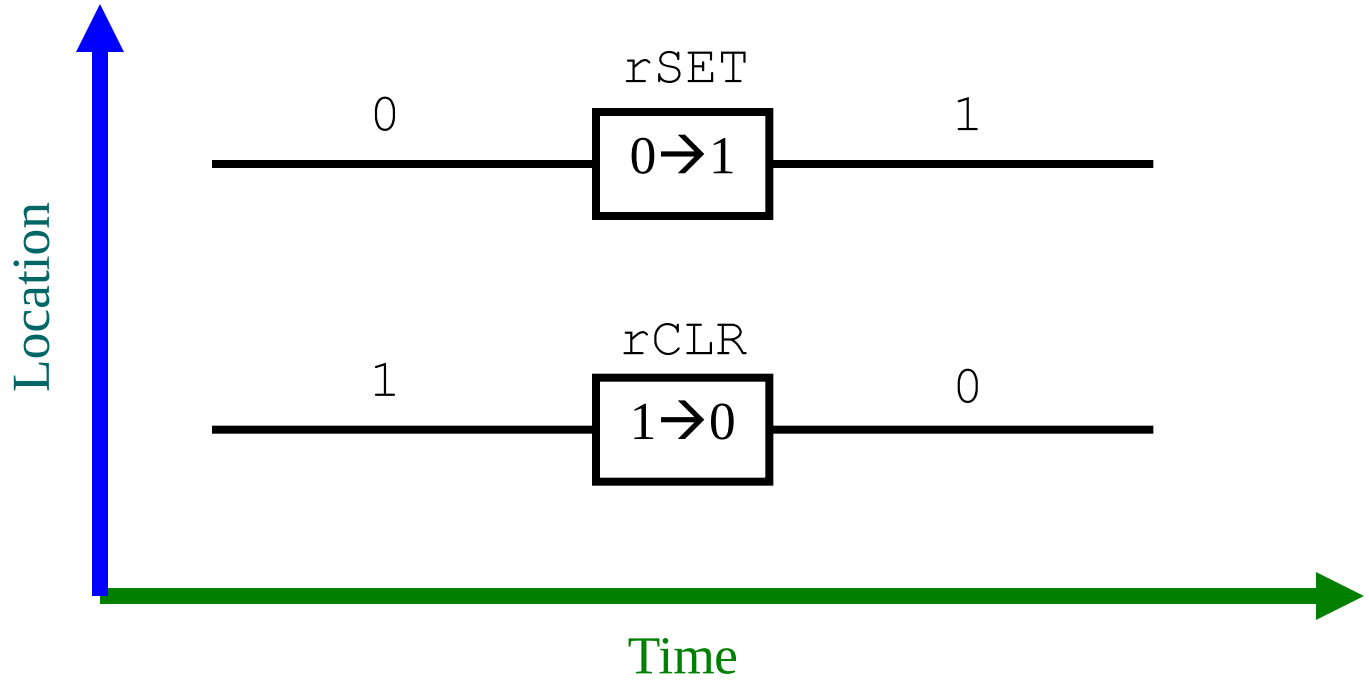
- Note that these tables are *not* logically reversible (invertible) according to the strict traditional definition…
  - Since they don't represent a one-to-one transformation of *all* possible initial states. (Some final states have >1 predecessor.)
- *However*, if we restrict our use of these operations so as to always *avoid* the input states that actually result in dissipation,
  - *Then*, we obtain a one-to-one transformation of *the subset of the possible initial states that are actually used* in the design,
  - And that is the correct statement of the minimum logical requirement for avoiding Landauer's limit!





rSET/rCLR in spacetime diagram

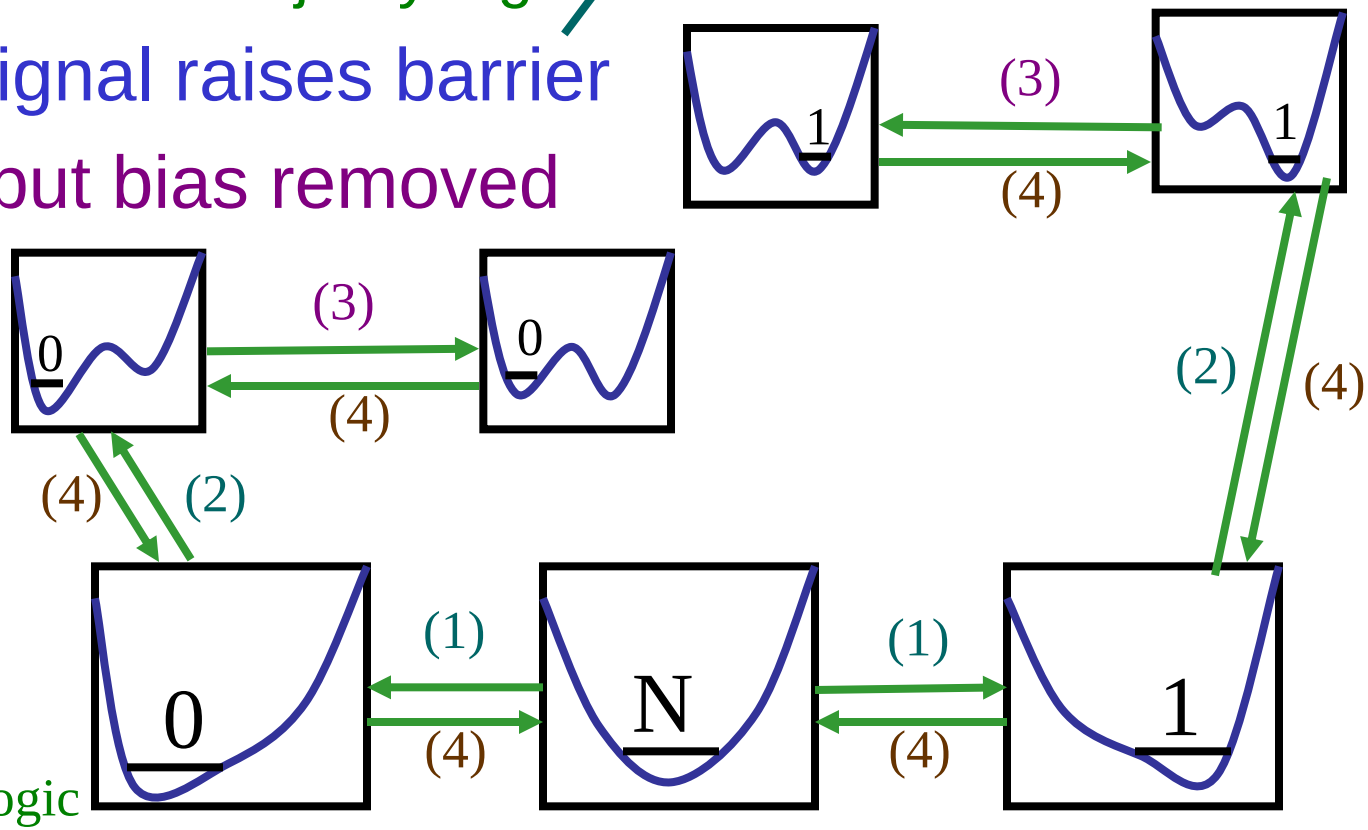
- Here is a suggested graphical notation for rSET/rCLR in the spacetime diagram picture.
  - However, keep in mind that the spacetime diagram formalism omits representation of the control signal that determines exactly *when* the operation occurs.



Type 1: Input-Bias Clocked-Barrier Reversible Latching (& Logic)

- **Cycle of operation:**
  - (1) Data input applies bias
    - Add forces to do majority logic
  - (2) Clock signal raises barrier
  - (3) Data input bias removed

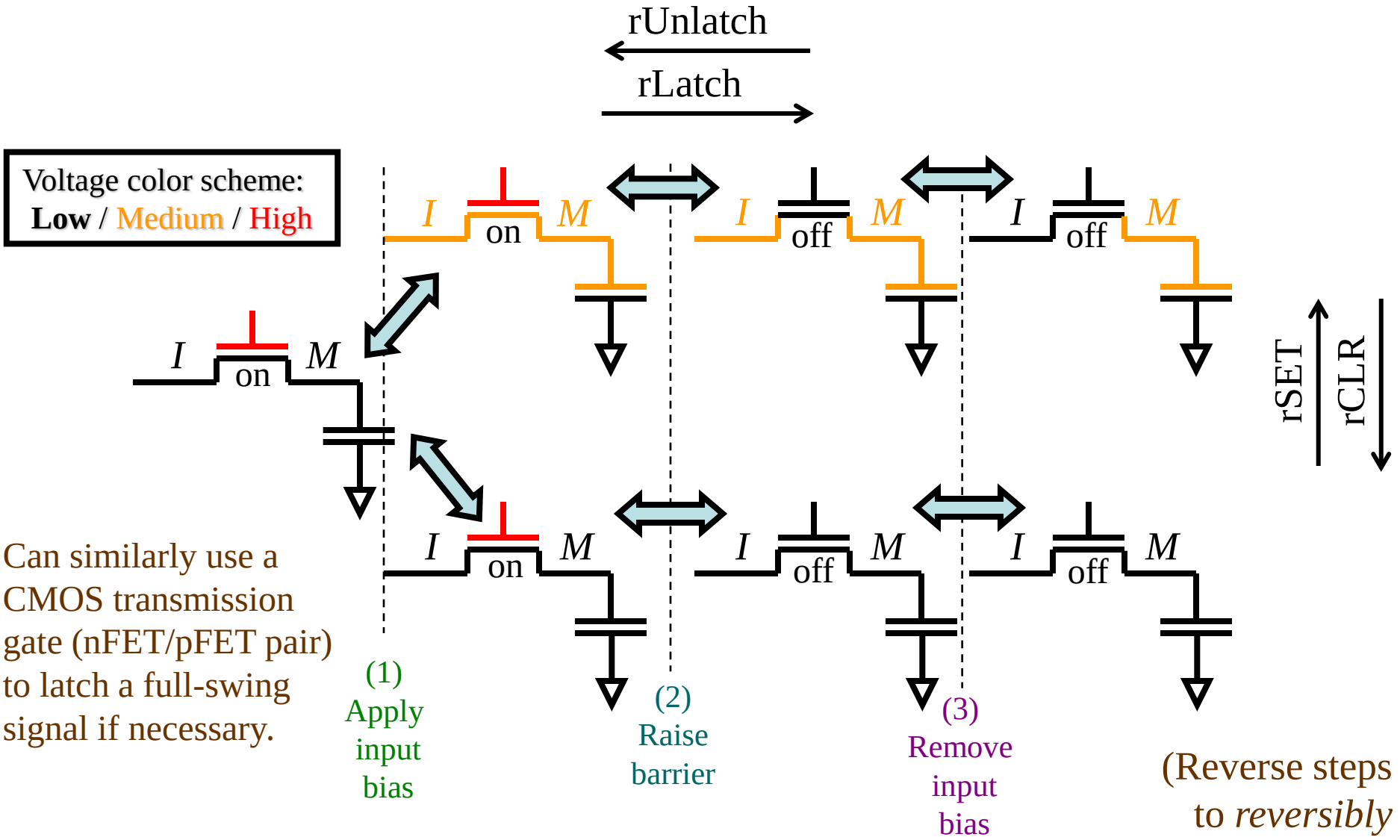
(Can amplify/restore input signal in the barrier-raising step.)



Can reset latch reversibly (4) given copy of contents.

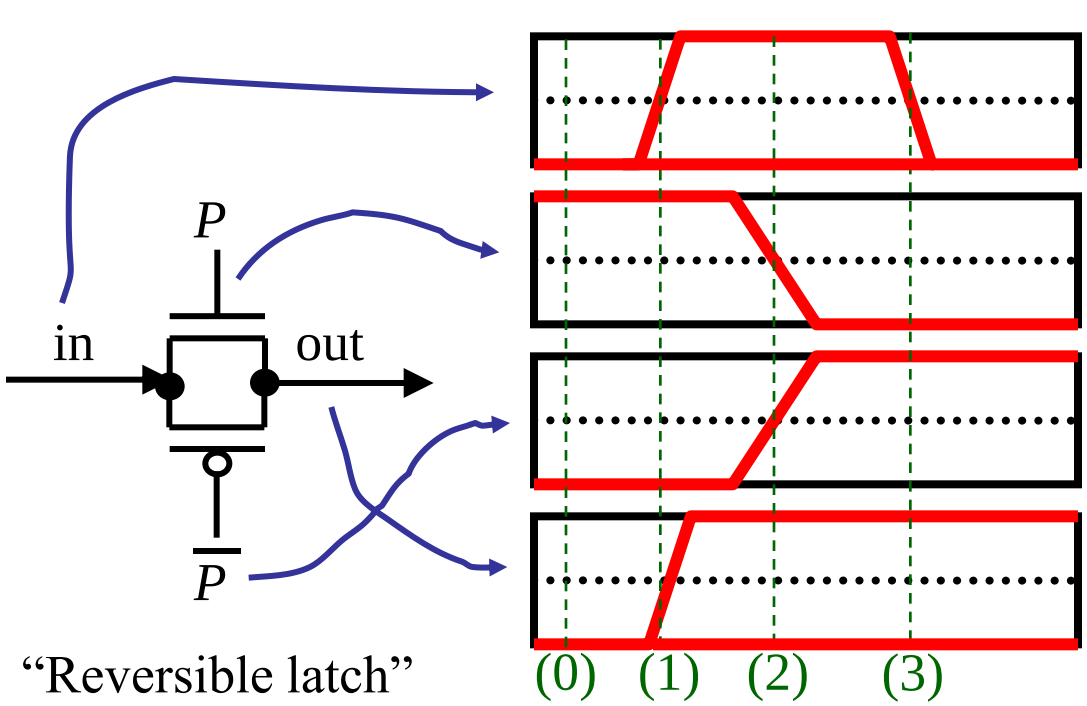
Examples: Adiabatic QCA, SCRL latch, Rod logic latch, PQ logic, Buckled logic, Helical logic

1-transistor Adiabatic rSET/rCLR/latch/DRAM cell



A Simple Reversible CMOS Latch

- Uses a single standard CMOS *transmission gate* (T-gate).
- Sequence of operation:
  - (0) input level initially tied to latch 'contents' (output);
  - (1) input changes gradually → output follows closely;
  - (2) latch closes, charge is stored dynamically (node floats);
  - (3) afterwards, the input signal can be removed.

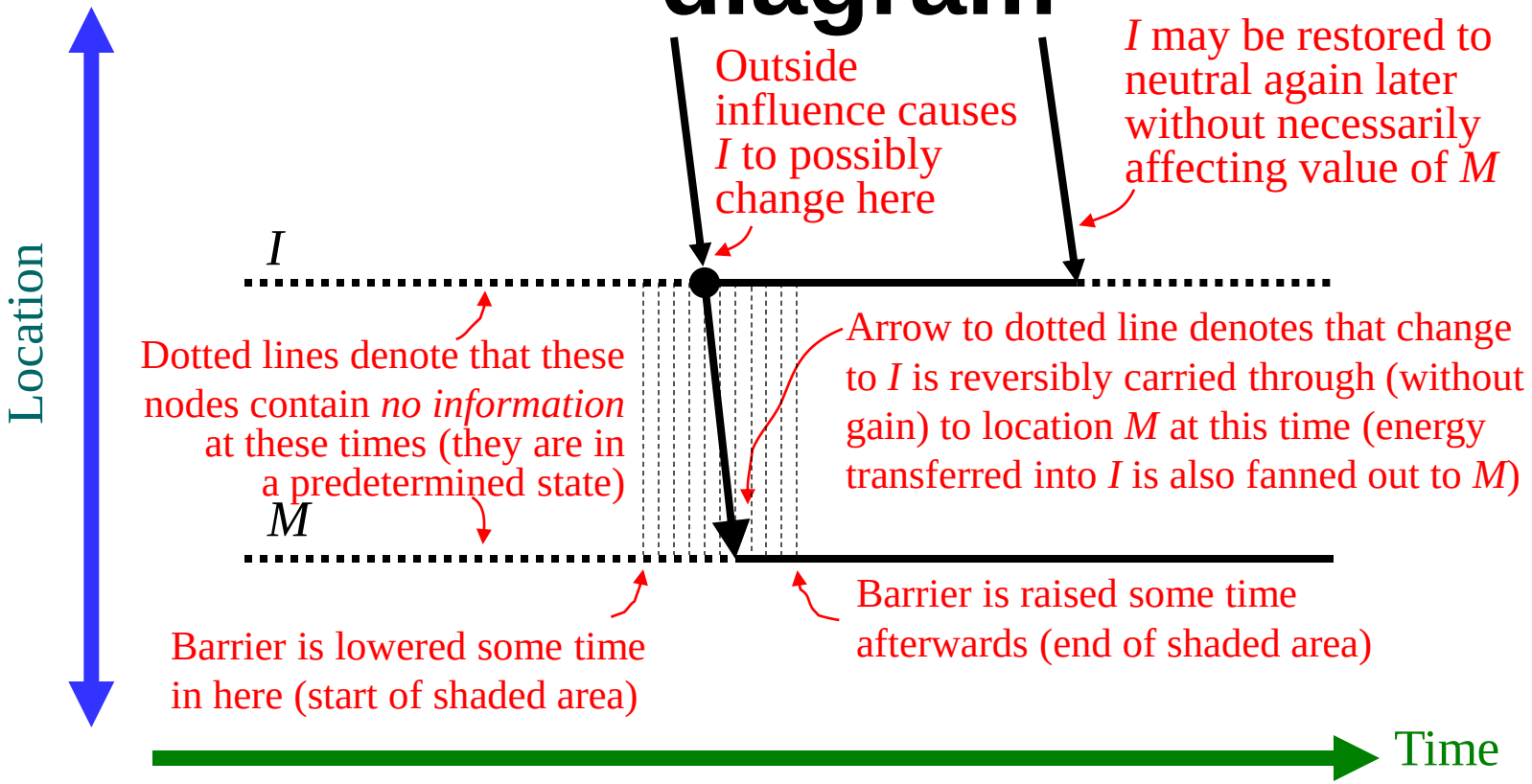


"Reversible latch"

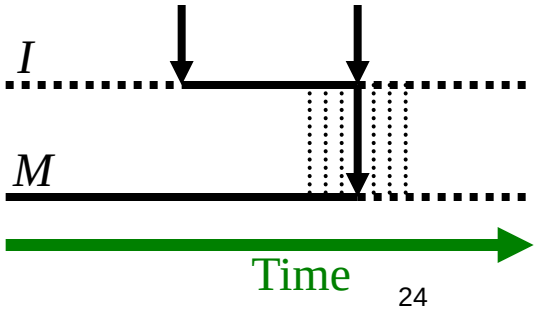
Before input:		*Input arrived:*		*Input removed:*	
in	out	in	out	in	out
0	0	0	0	0	0
		1	1	0	1

- Later, we can reversibly "unlatch" the data with an exactly time-reversed sequence of steps.

Reversible latch in spacetime diagram



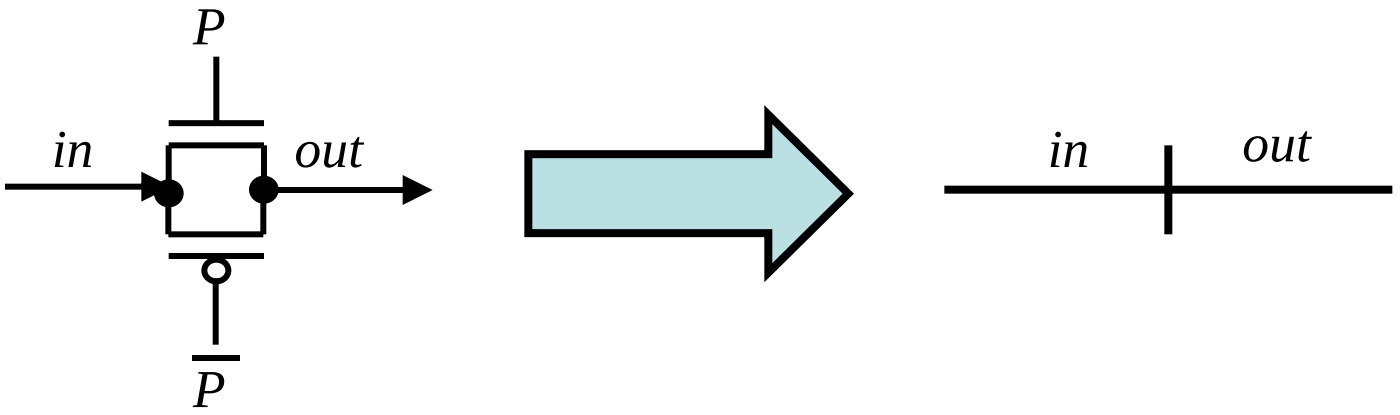
Unlatching sequence:



Note this operation is adiabatic only if *I* and *M* match up exactly when they are first connected together!

Icon for Latch

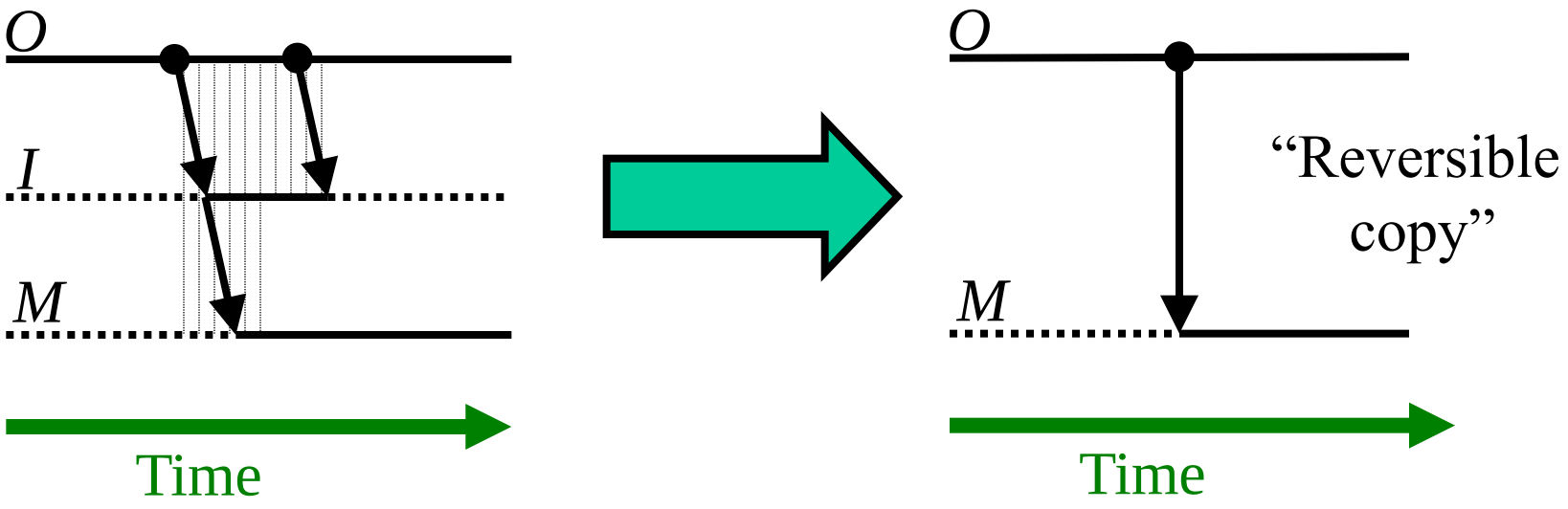
- Extremely simple notation:
  - Just draw a short orthogonal line across the wire to indicate the presence of a latch at that point.
    - Control of latch timing is implicit.



- Note the same latch hardware can actually be used to latch signals being passed in either direction. → It's symmetric in space and time.

rCOPY - Spacetime Diagram

- Suppose the signal on the input node *I* was produced as a temporary copy of some origin node *O*.
  - We will see how to implement this reversibly later.
- Then for simplicity of our diagrams, we may wish to omit explicit representation of the intermediate node *I*.
  - However, we must keep in mind that there is then a small additional space usage not explicitly shown in the diagram.



Operation Naming Conventions

- Clarification of our naming conventions for operations:
  - **`r`** – "reversible"
    - Means operation is a conditionally-reversible variant of an operation that would traditionally have been irreversible.
  - **`l`** – "latching"
    - Operation includes latching of result; *i.e.*, input operands aren't required to be held after output operand is modified.
  - **`c`** – "controlled"
    - First operand is a "control" input; operation is only performed if it is a 1.
  - **`Un`** – "undo"
    - Operation does the time-reversal of the operation without the "Un" prefix.

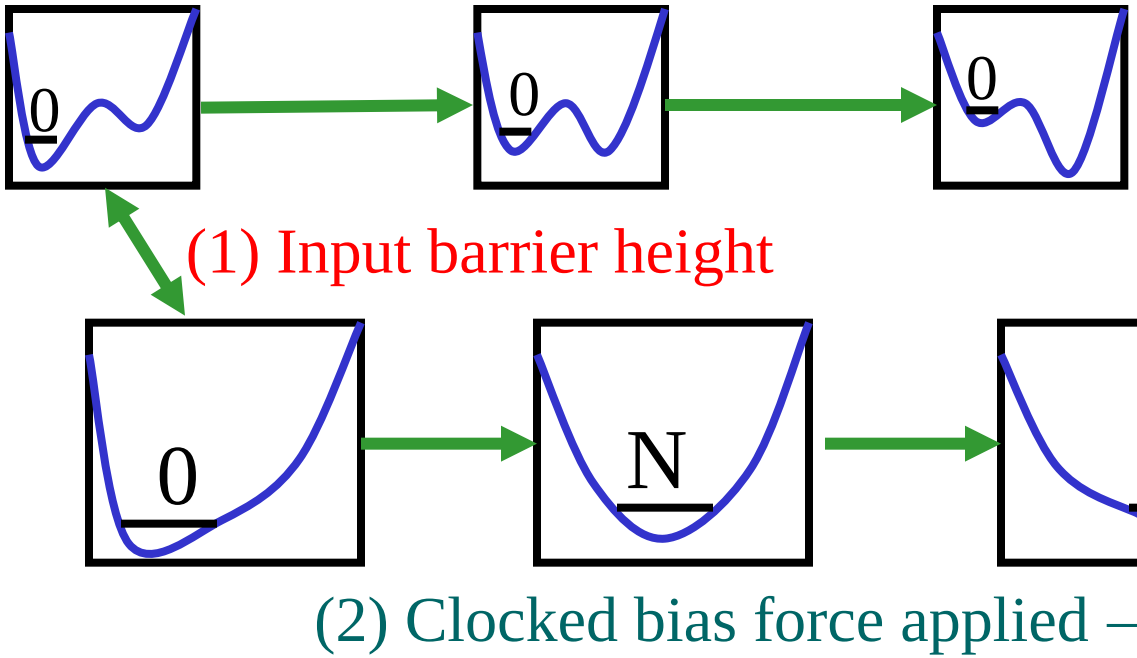
Operations Encountered So Far

- Ordinary irreversible operations:
  - `CLR`(a): $a := 0$. Clear operation.
  - `Invert`(a, b): $b := \underline{a}$. Inverter operation.
  - `AND`(a, b, c): $c := ab$. AND gate operation.
  - `OR`(a, b, c): $c := a+b$. OR gate operation.
  - `XOR`(a, b, c): $c := a \oplus b$. XOR gate operation.
- Unconditionally reversible operations:
  - `NOT`(a): $a := \underline{a}$. In-place NOT operation.
  - `cNOT`(a, b): $b := a \oplus b$. Controlled-NOT operation.
  - `ccNOT`(a, b, c): $c := ab \oplus c$. Toffoli gate operation.
  - `SWAP`(a, b): $a \leftrightarrow b$. Swap operation.
  - `cSWAP`(a, b): if a, $a \leftrightarrow b$. Fredkin gate operation.
- Conditionally reversible operations:
  - `rCLR`(a): (a) $a := 0$. Reversible clear operation.
  - `rcSET`(a, b): ($\underline{a} + \underline{b}$) if a, $b := 1$. Controlled rSET operation.
  - `rCOPY`(a, b): ($\underline{b}$) $b := a$. Reversible copy operation.

Type 2: Input-Barrier, Clocked-Bias Reversible Retractile Logic

- **Cycle of operation:**
  - (1) Inputs raise or lower barriers
    - Do logic w. series/parallel barriers
  - (2) Clock applies bias force, which changes state, or not

- Barrier signal is amplified! Gain, restoring logic, fan-out.
- Must reset output prior to changing input.
- Combinational logic only!

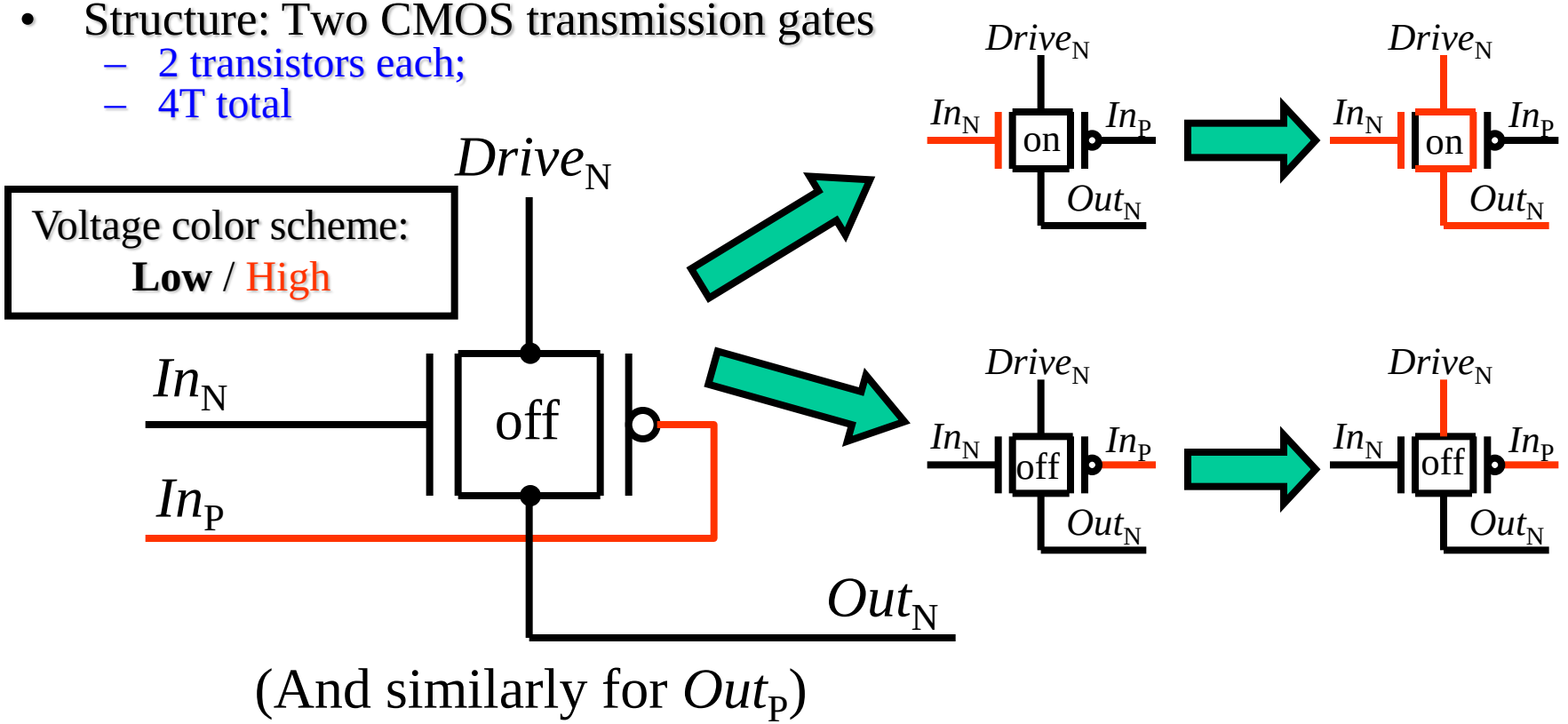


Examples:
Hall's logic,
SCRL gates,
Rod logic interlocks

Type 2 example: Adiabatic CMOS "buffer" (really, a cSET/cCLR gate)

- Controlled-SET / controlled-CLEAR.
  - Using dual-rail signaling, we can reversibly set or clear a bit on an unoccupied logic node (pair of voltage nodes),
    - conditionally on an input node.
- Structure: Two CMOS transmission gates
  - 2 transistors each;
  - 4T total

- Features:
  - Amplifies input signal.
  - Fully restores logic levels.



(And similarly for Out_P)

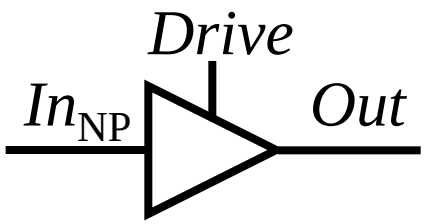
Transition Table for **cSET**

- It is not *always reversible*,
  - Not a one-to-one transformation of *all* possible local states,
- But, it is *conditionally reversible*
  - *I.e.*, on condition that input state 1,1 is avoided.
- Transition table for **cCLR** is similar.

Before **cSET**		After **cSET**	
Source	Destination	Source	Destination
0	0	0	0
0	1	0	1
1	0	1	1
1	1	1	0

Icon for **cSET/cCLR** gate

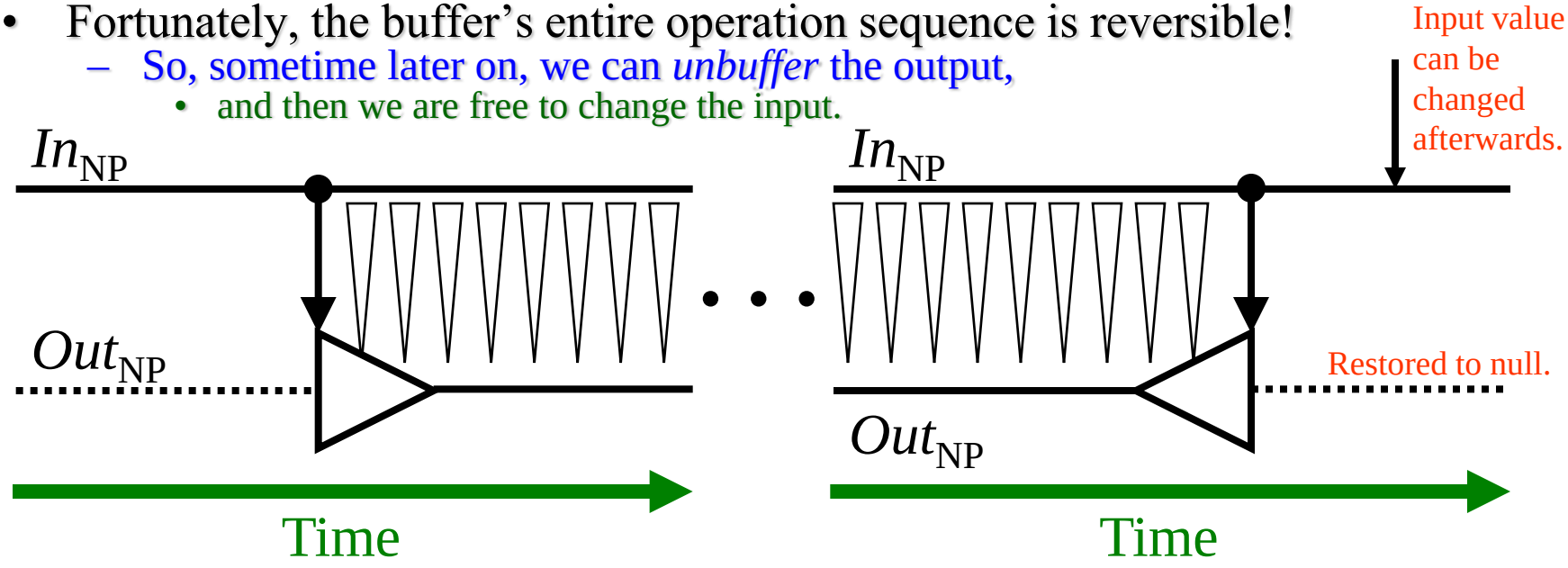
- Represents a gate that can perform either **cSET** or **cCLR** on the *Out* node, with either operation conditioned on In_{NP} being a logic 1.



- Constraints on use (in simple CMOS impl.):
  - Input must be a dual-rail pair.
  - The *Drive* control signal must have the same bus width as the *Out* signal.

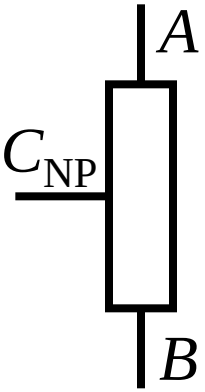
Spacetime diagram for buffer

- Subscript $_{NP}$ notation denotes shorthand for dual-rail NP pair of wires.
  - Still denotes a single *logical* bit.
- Diagram emphasizes that the buffer copies In_{NP}'s value to a *new* location.
  - The value simultaneously remains available in the old location.
- Dotted horizontal line shows that Out_{NP} is empty prior to the operation.
  - The absence of "×" icon shows that the operation is reversible.
- Buffer icon indicates that the input signal is being amplified and restored.
  - Note that the input comes from In_{NP}, *not* from previous value of Out_{NP}.
- Downward wedges remind us the output remains dependent on the input.
  - Input can't be changed without (possibly) irreversibly destroying output.
- Fortunately, the buffer's entire operation sequence is reversible!
  - So, sometime later on, we can *unbuffer* the output,
    - and then we are free to change the input.



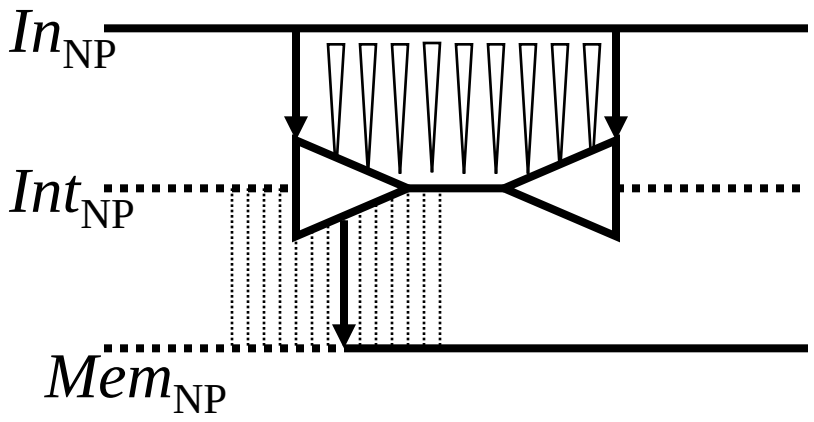
Reversible Buffered Latch

This is our icon for a CMOS transmission gate (T-gate). It says that nodes *A* and *B* are connected whenever the control signal C_{NP} has logic value 1.

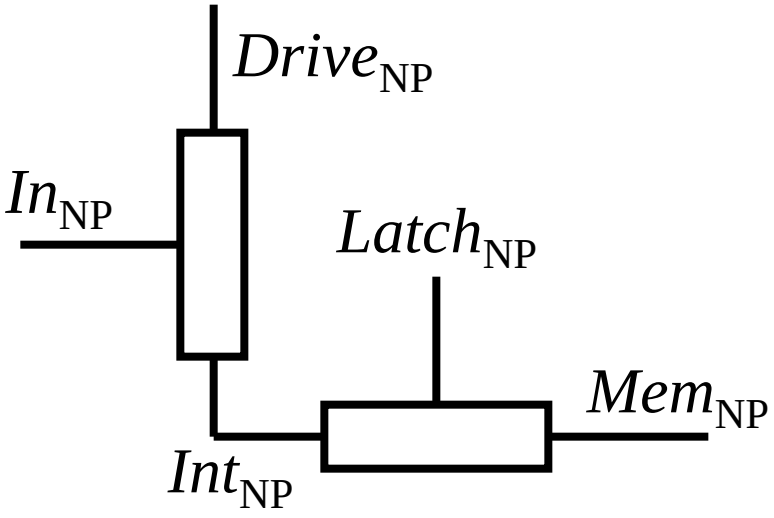


- Uses two dual-rail T-gates.
- Combines a buffer and latch.
  - Reversibly copies In_{NP} to Mem_{NP} when operated.

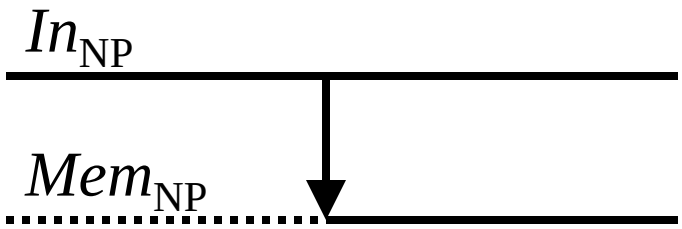
Spacetime diagram for operation sequence:



Physical structure:

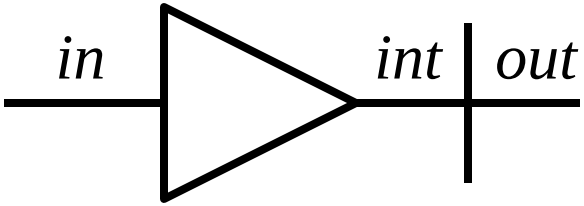


Implements "reversible copy":



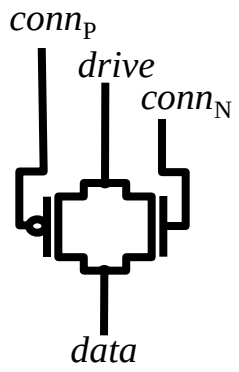
Hardware Icon for Buffered Latch

- Looks a little bit like a diode icon, but isn't.
- Composed of our previous icons for:
  - Reversible buffer
  - Reversible latch

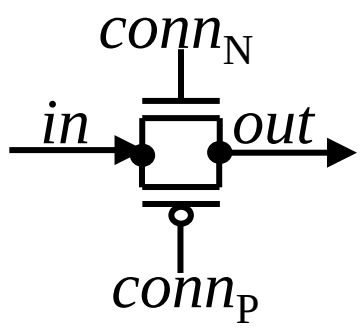


- This gate properly implements the **`rlcSET`**(*in*,*out*) and **`rlcCLR`**(*in*,*out*) operations!
  - The buffer alone does not quite do it, because of the constraint that *in* must be stable while *out* is driven.

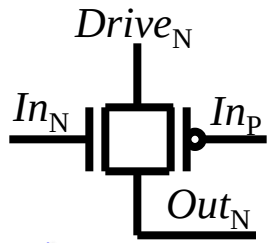
Three Ways to Use a Transmission Gate Reversibly



- **`rSET`/`rCLR`** – Drive signal and gate signals are both considered to be control.
  - – Unconditional, not data-dependent.
- **`rLatch`/`rUnlatch`** – Drive signal is a data input, gate signals are control.
  - – **`rLatch`**(*in*,*out*) just isolates *in* from *out*



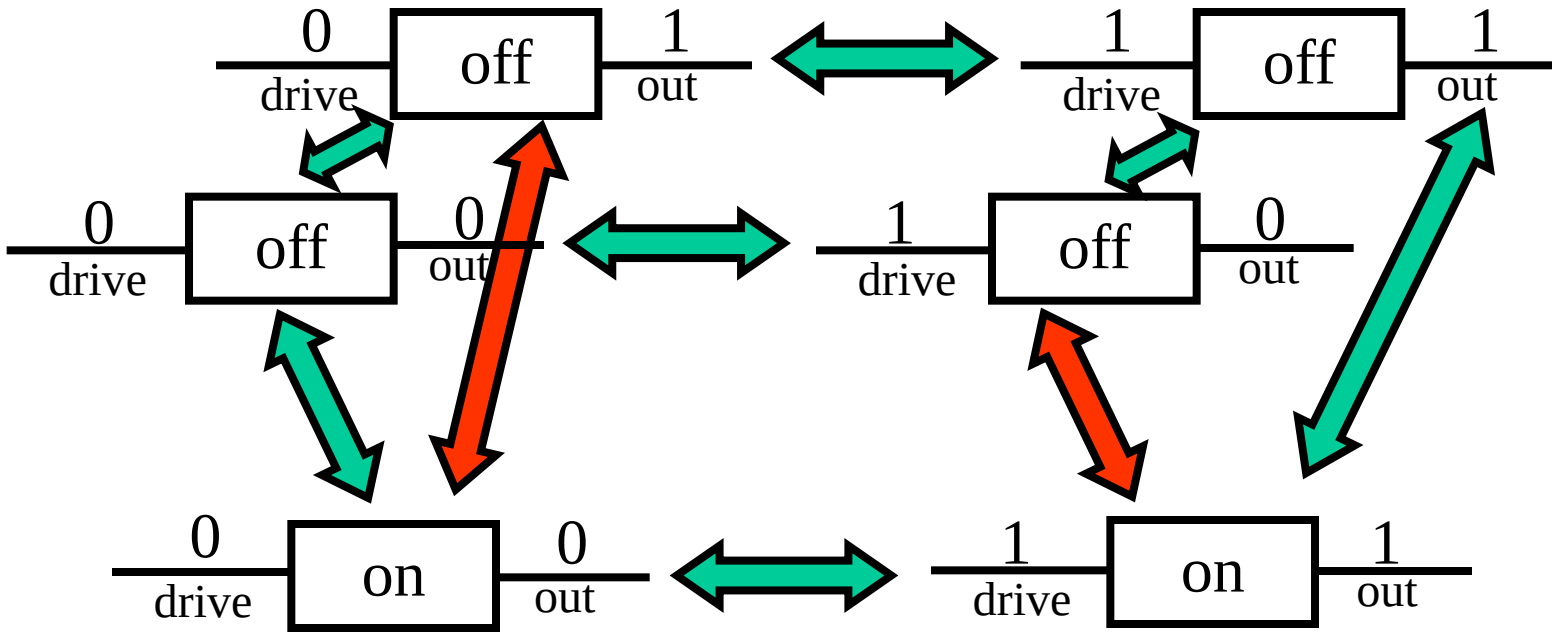
- **`cSET`/`cCLR`** – Drive signal is a control signal, gate signals comprise data input.
  - – **`cSET`**(*in*,*out*) presents a copy of *in* on *out*



- What if everything is a data input?
  - – Some data transitions are reversible, others not

Reversible & Irreversible T-gate Transitions

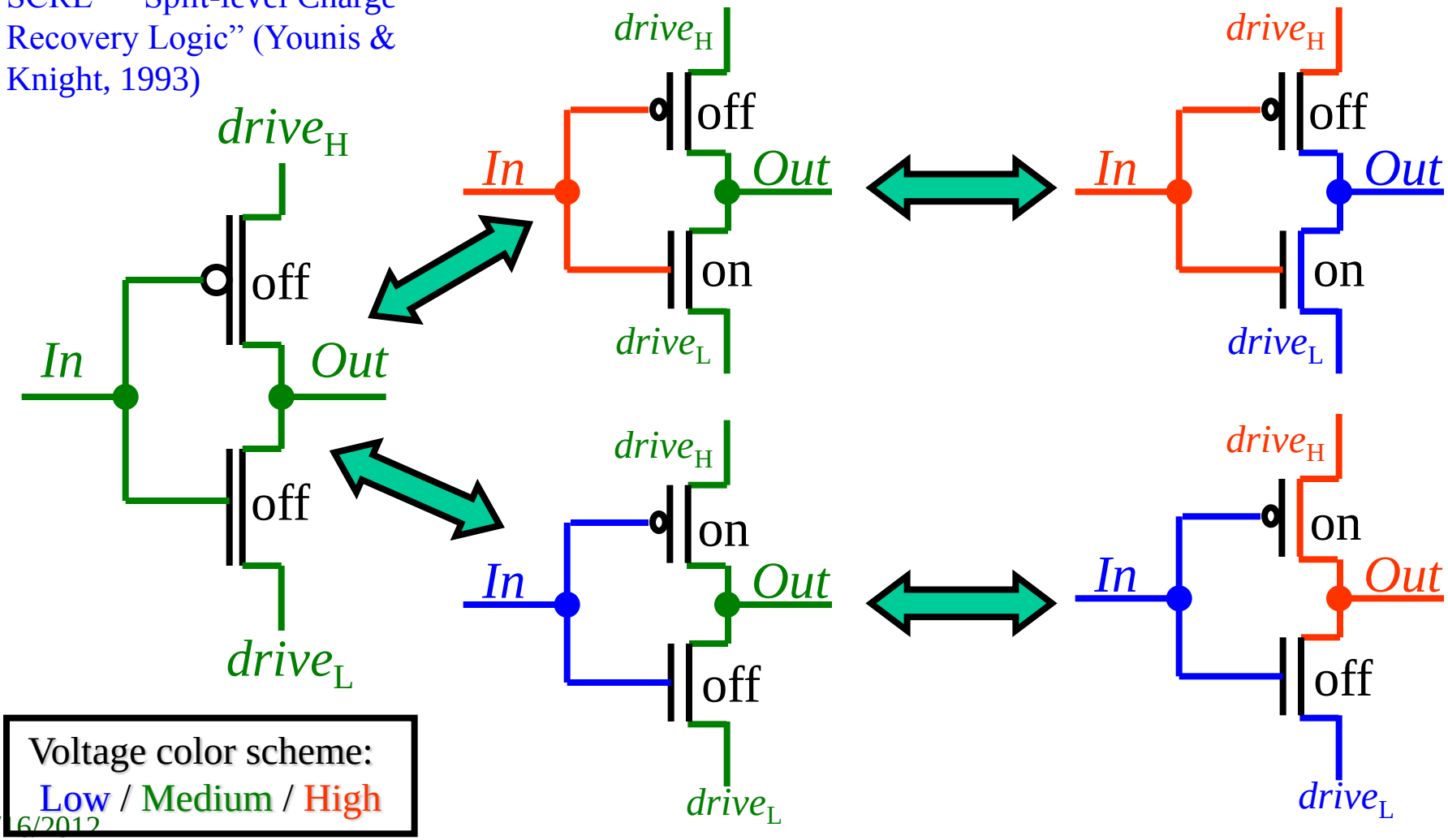
- When drive and gate are both data-dependent
  - Certain data transitions must be avoided…



Type 2 example: SCRL inverter (w/o latch)

- Same structure as static CMOS inverter, but used reversibly.
- Produces a fully-restored, amplified output signal.
- Inverters can be cascaded, but need latches to get feedback.

SCRL = "Split-level Charge Recovery Logic" (Younis & Knight, 1993)



Voltage color scheme:
Low / Medium / High

SCRL Inverter Transition Table

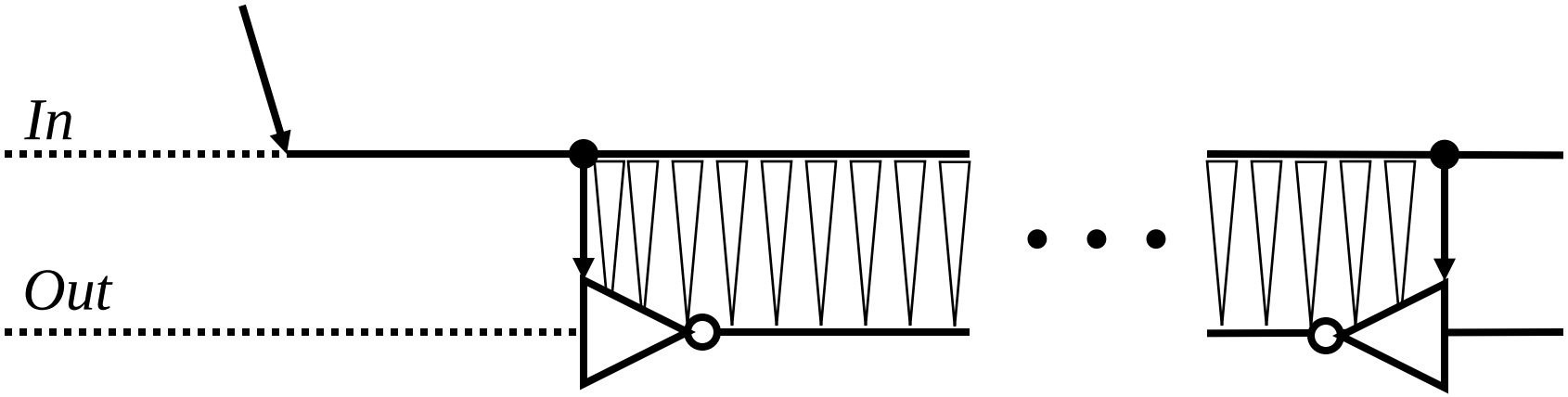
Before SCRL-Inv		After SCRL-Inv	
In	*Out*	*In*	*Out*
0	0		
0	½ →	0	1
0	1		
½	0		
½	½		
½	1		
1	0		
1	½ →	1	0
1	1		

- Conditionally reversible, if input is valid and output is ½ just before drivers do their thing.
- No point in even listing the table entries that don't occur; can summarize operation below.

Before SCRL-Inv		After SCRL-Inv	
In	*Out*	*In*	*Out*
0	½ →	0	1
1	½ →	1	0

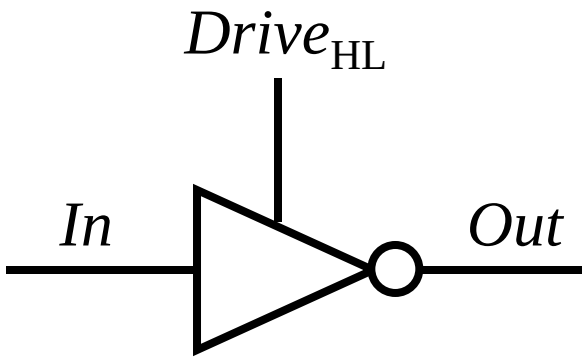
Spacetime Diagram for SCRL Inverter

- Note that the notation shows that *Out* is being computed from *In* on a separate wire.
  - *In* is explicitly not being inverted "in place."
- Wedge symbols show ongoing dependence.
  - Of course, we can always undo the op later.

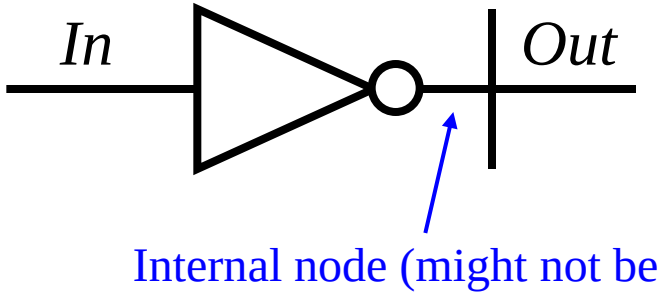


Icon for SCRL Inverter

- Same as normal inverter icon
  - Can (optionally) also show control (drive) bus.



- Note we can build a latched SCRL inverter very easily:

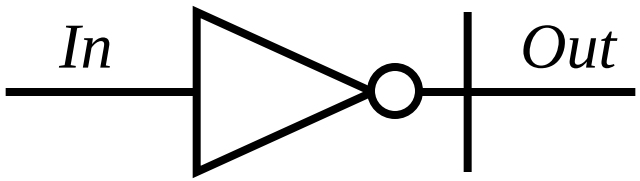


`rsCopyInv`(*In*,*Out*)

Reversible Split-level Copy with Inversion

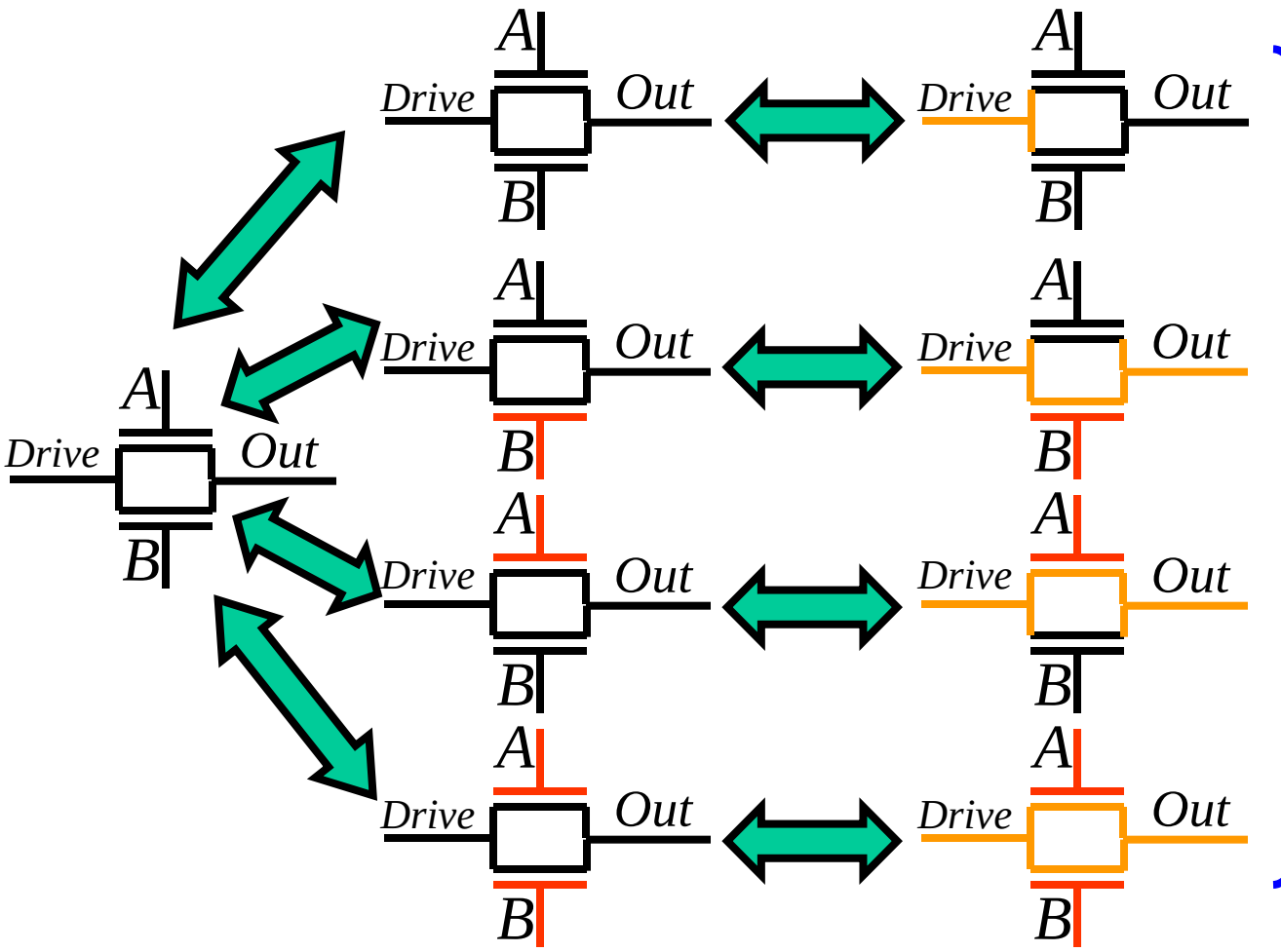
- Preconditions:
  - *Out* is initially clear (logic N - neutral).
- Semantics:
  - *Out* := ¬ *In*

- Gate icon in hardware diagrams:
  - (same gate also performs `rUnCopyInv`.)



Simple Logic Example: Adiabatic NMOS OR gate

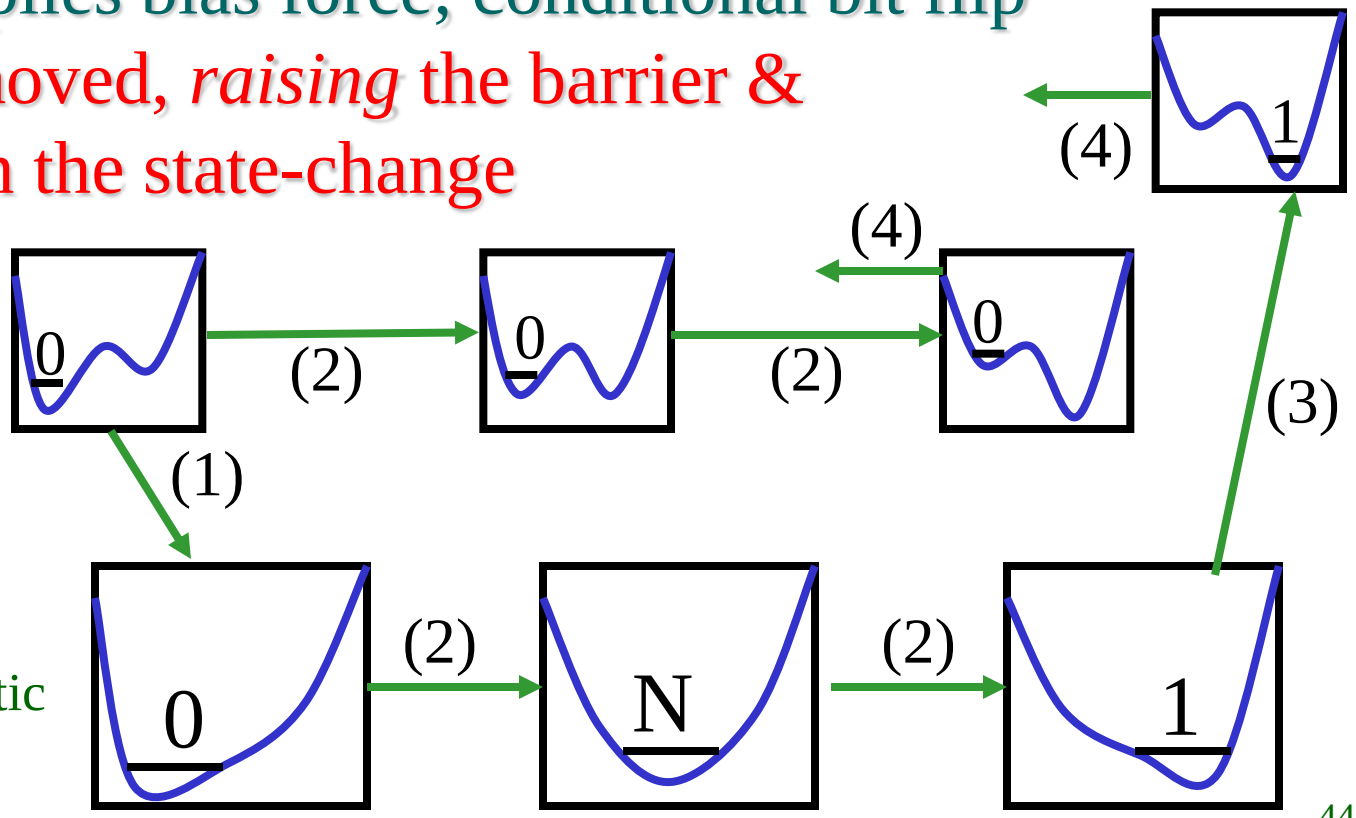
- Input barriers along two parallel paths



- Reverse sequence decomputes *Out*.
  - Can't change *A*,*B* freely until then.
- With NMOS, *Out* is weak (orange).
- Can use an SCRL inverter to restore the signal levels.
  - If appropriately biased…
  - Or, just use CMOS transmission gates instead (8T OR)

Type 3: Input-Barrier, Clocked-Bias Latching Logic

- **Cycle of operation:**
  1. Input *conditionally lowers* barrier
     - Do logic w. series/parallel barriers
  2. Clock applies bias force; conditional bit flip
  3. Input removed, *raising* the barrier & locking in the state-change
  4. Clock bias can retract

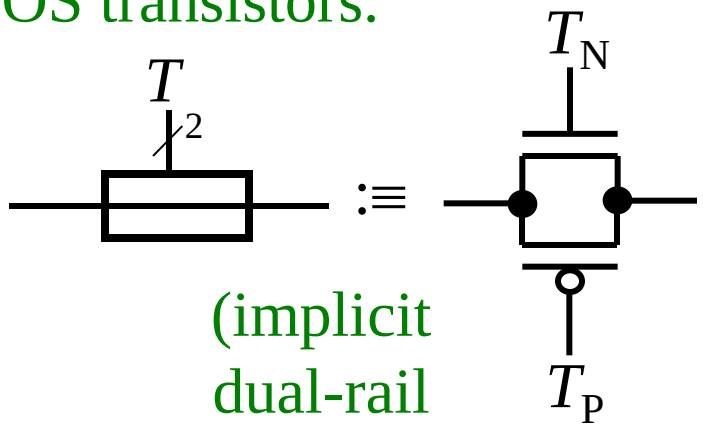


Examples: Mike's 4-cycle 2-level adiabatic CMOS logic (2LAL)

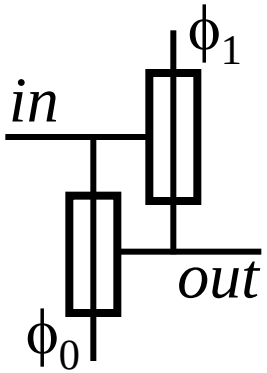
2LAL: 2-level Adiabatic Logic

A pipelined fully-adiabatic logic invented at UF (Spring 2000), implementable using ordinary CMOS transistors.

- Use simplified T-gate symbol:



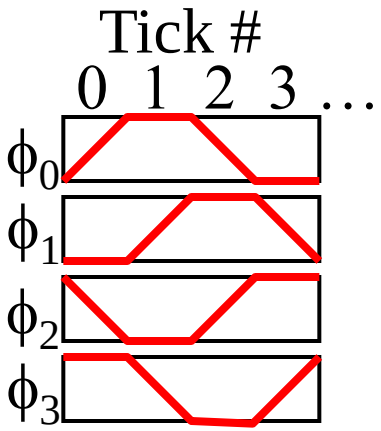
- Basic buffer element:
  - cross-coupled T-gates:
    - need 8 transistors to buffer 1 dual-rail signal



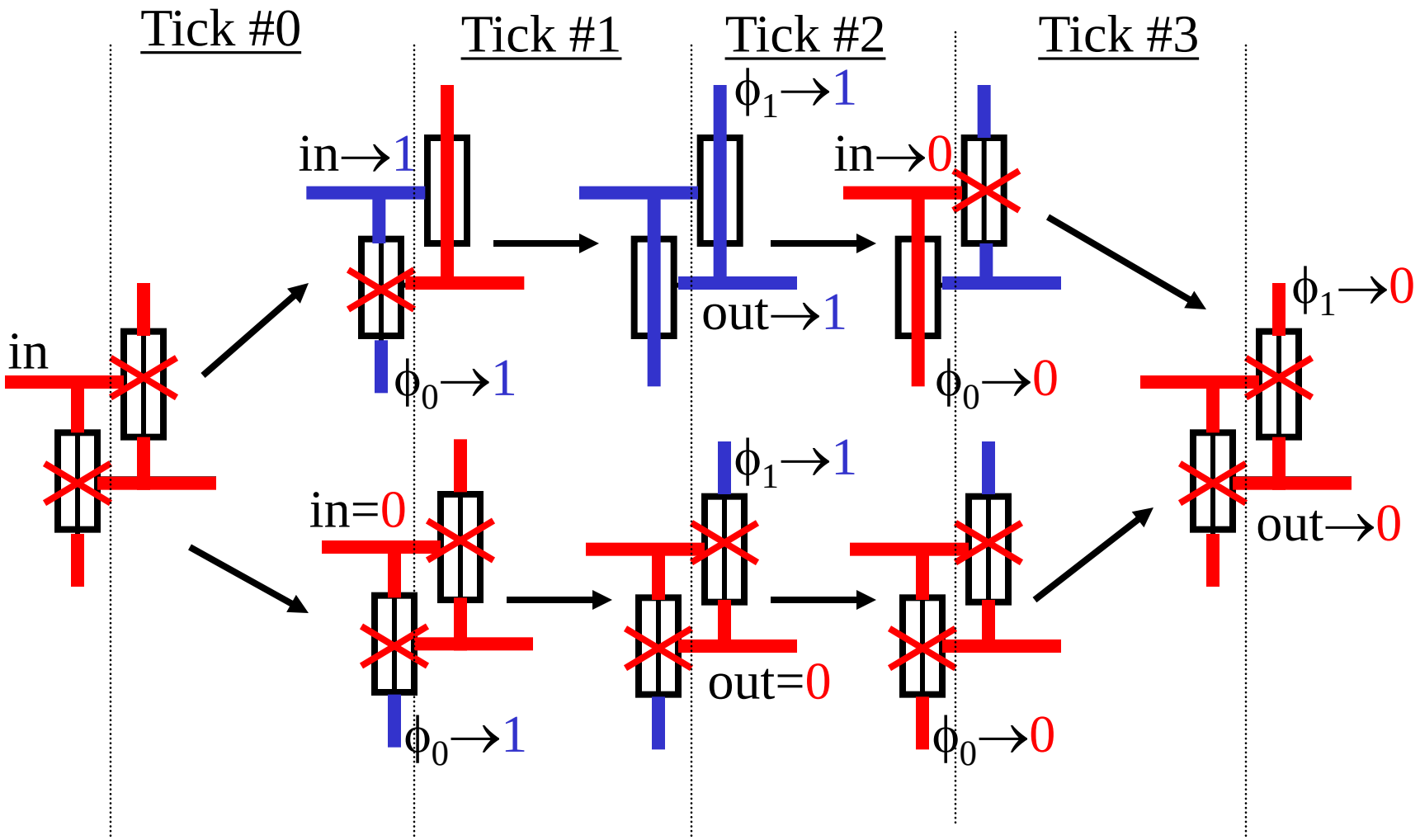
(implicit dual-rail encoding everywhere)

- Only 4 timing signals $\phi_{0\text{-}3}$ are needed. Only 4 ticks per cycle:
  - ϕ_i rises during ticks $t \equiv i \pmod 4$
  - ϕ_i falls during ticks $t \equiv i+2 \pmod 4$

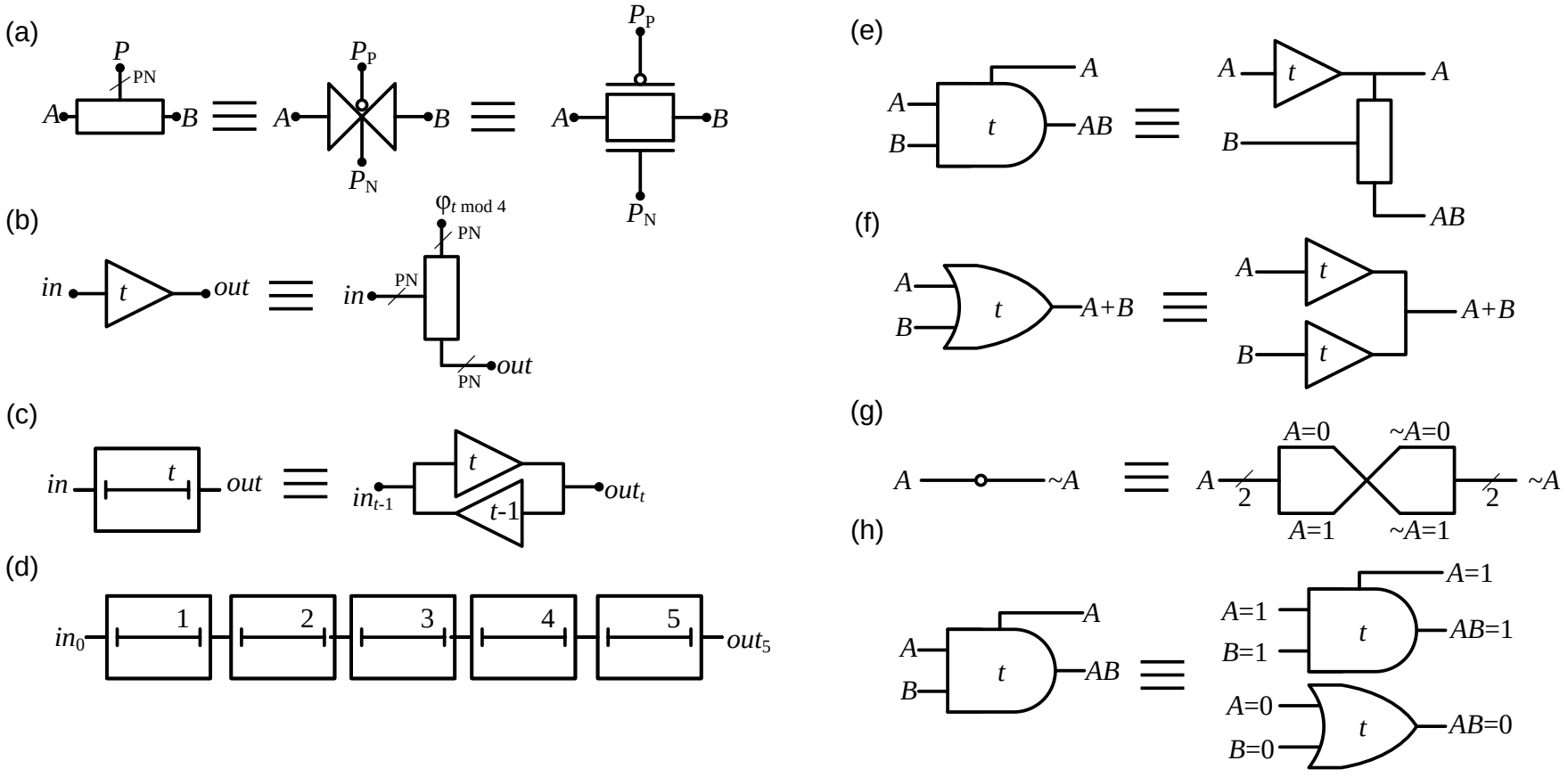
Animation:



2LAL Cycle of Operation



A Schematic Notation for 2LAL

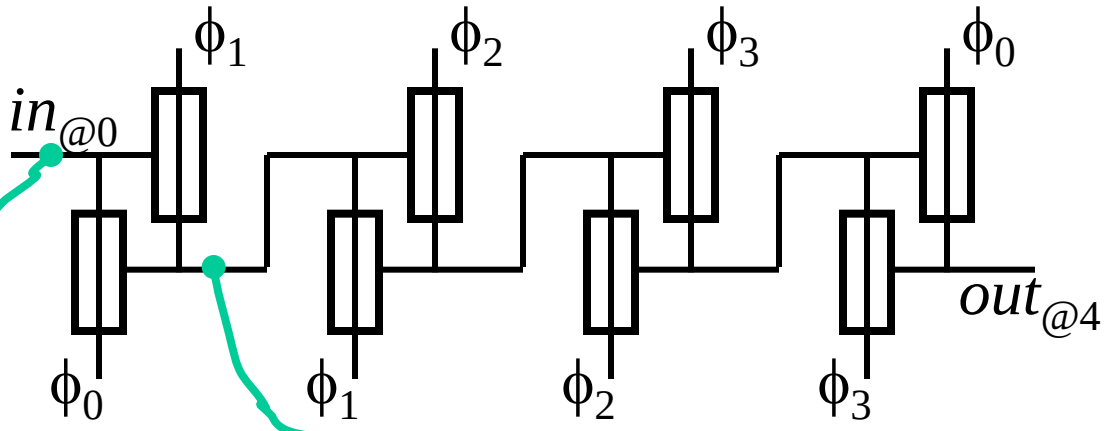


2LAL Shift Register Structure

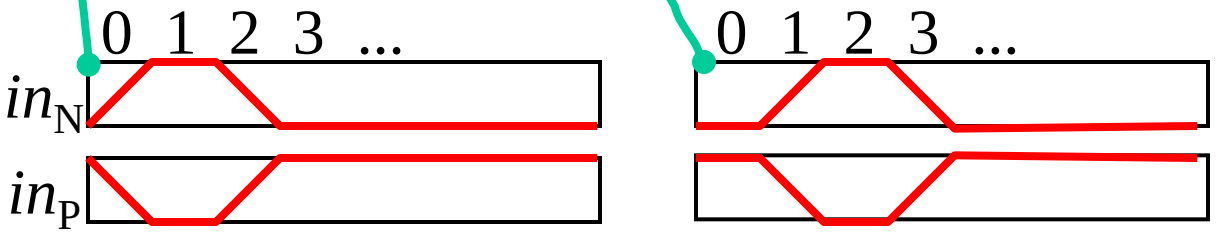
Animation:

2lal.swf

- 1-tick delay per logic stage:

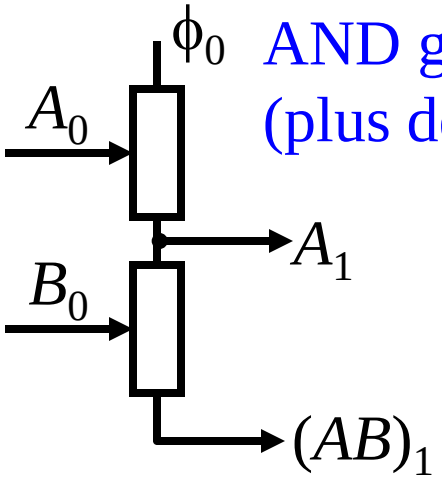


- Logic pulse timing and signal propagation:



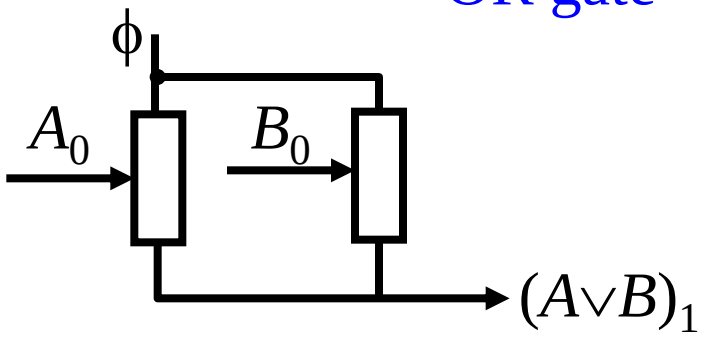
More Complex Logic Functions

- Non-inverting multi-input Boolean functions:

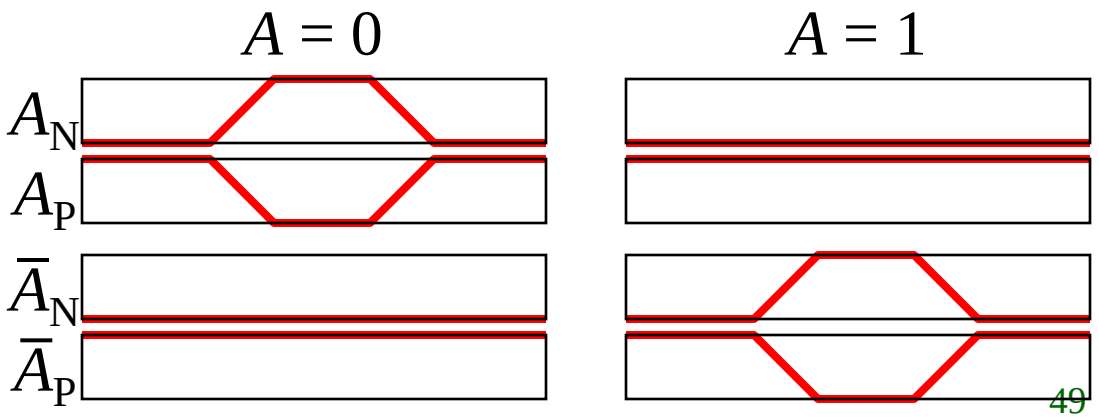


AND gate
(plus delayed *A*)

OR gate

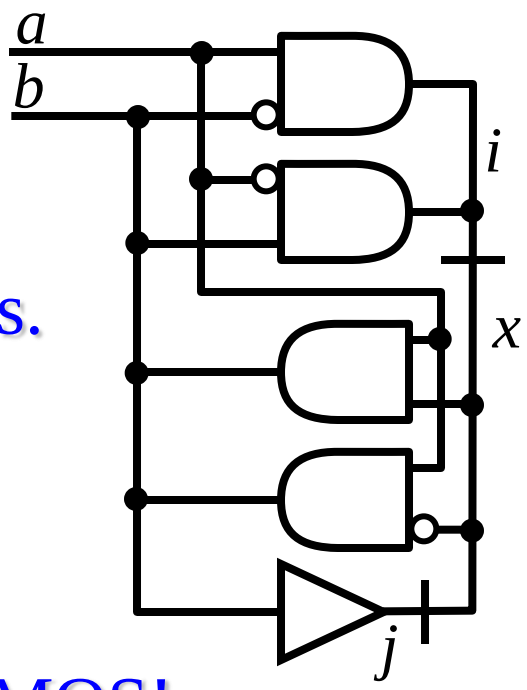


- One way to do inverting functions in pipelined logic is to use a quad-rail logic encoding:
  - To invert, just swap the rails!
    - Zero-transistor "inverters."



cNOT hardware diagram

- Here is an implementation of in-place **cNOT**(*a*,*b*) (controlled-NOT)
  - In terms of reversible AND or OR, reversible buffers, reversible latches, and (0T dual-rail) complement bubbles.



- As you can see, it is rather complicated!
  - Illustrates that cNOT might not be a very good primitive for reversible CMOS!
- This structure can be properly called a **cNOT** gate (as opposed to a **cNOT** operation)

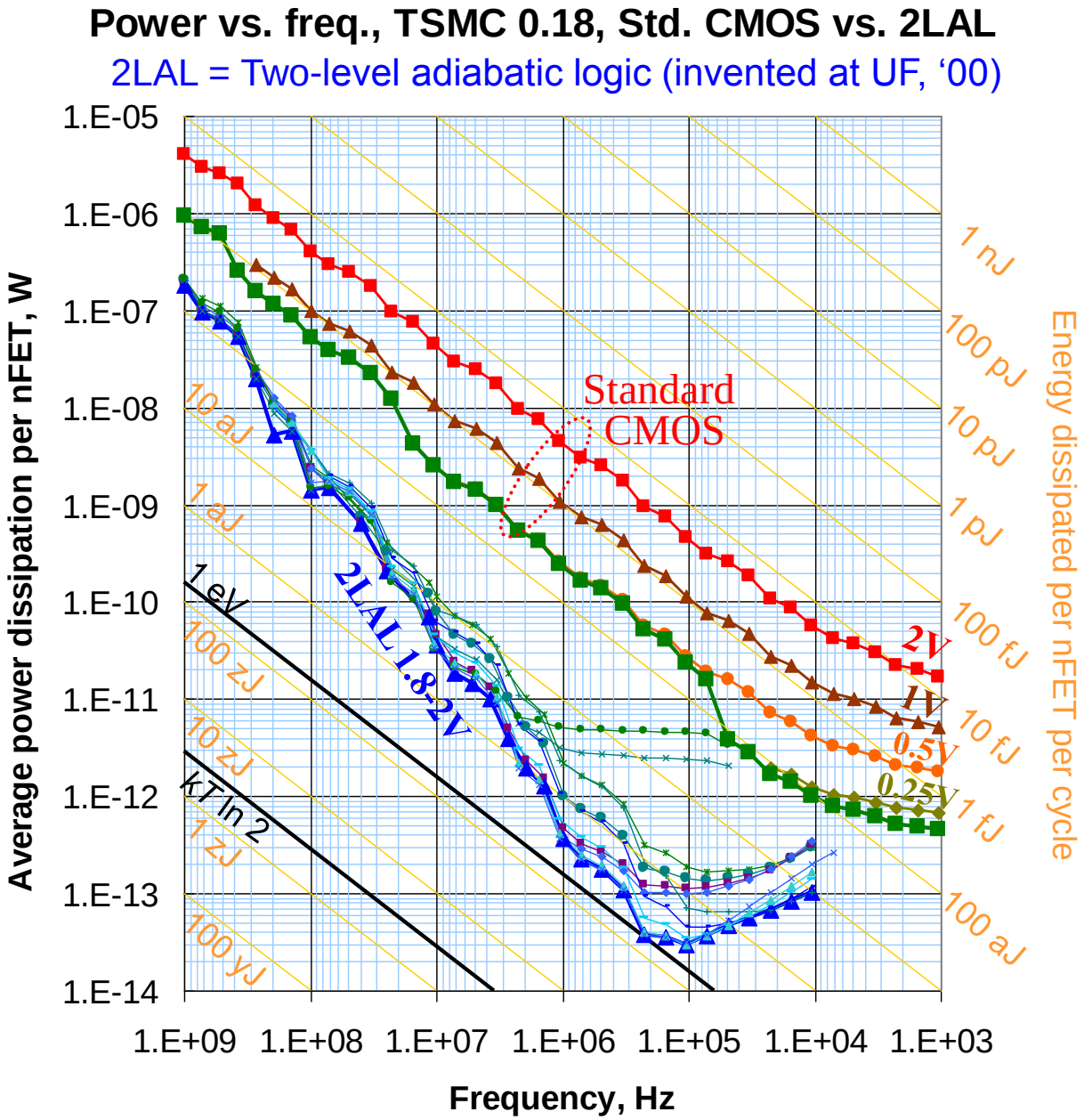
cNOT operation sequence

- Steps to implement cNOT(a,b):

	$\underline{a}$	$\underline{b}$	$\underline{i}$	$\underline{x}$	$\underline{j}$
	a	b	0	0	0
1. rlXOR(a, b, x):					
1a. rUnLatch(i, x)	a	b	(	0	) 0
1a. rAND(a, $\underline{b}$, i), rAND($\underline{a}$, b, i)	a	b	($a\oplus b$)		0
1b. rLatch(i, x)	a	b	$a\oplus b$	$a\oplus b$	0
1c. rUnAND(a, $\underline{b}$, i), rUnAND($\underline{a}$, b, i)	a	b	0	$a\oplus b$	0
2. rAND(a, x, b), rUnAND(a, $\underline{x}$, b)	a	$a\oplus b$	0	$a\oplus b$	0
3. rUnCopy(b, x):					
3a. rcSET(b, j)	a	$a\oplus b$	0	$a\oplus b$	$a\oplus b$
3b. rUnLatch(j, x)	a	$a\oplus b$	0	($a\oplus b$)	
3c. rcUnSet(b, j)	a	$a\oplus b$	0	(0)	
3d. rLatch(j, x)	a	$a\oplus b$	0	0	0

- Note it takes 9 full steps!

Shift Register Simulation Results (Cadence/Spectre)



- Graph shows power dissipation vs. frequency
  - in 8-stage shift register.
- At moderate frequencies (1 MHz),
  - Reversible uses < 1/100th the power of irreversible!
- At ultra-low power (1 pW/transistor)
  - Reversible is 100× faster than irreversible!
- Minimum energy dissip. per nFET is < 1 eV!
  - 500× lower than best irreversible!
    - 500× higher computational energy efficiency!
- Energy transferred is still ~10 fJ (~100 keV)
  - So, energy recovery efficiency is 99.999%!
    - Not including losses in power supply, though

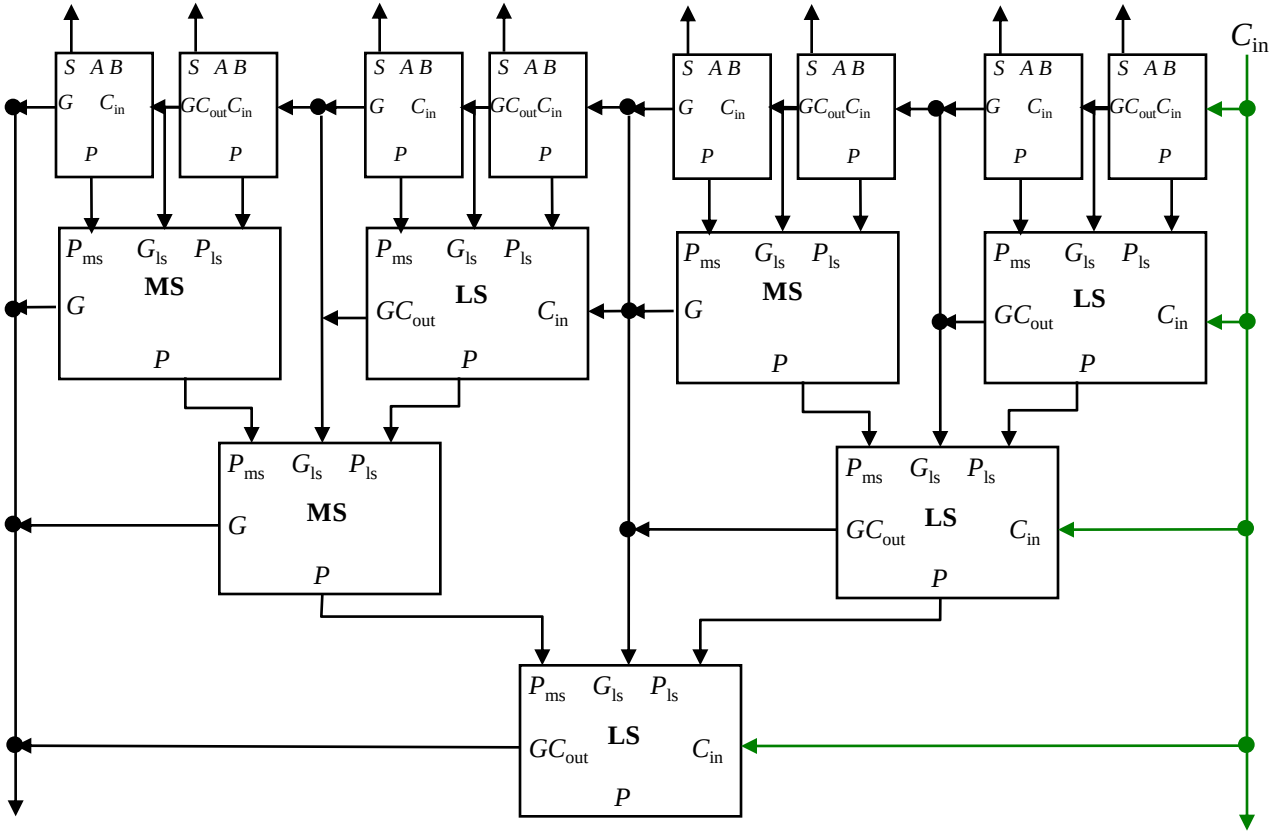
M. Frank, IEEE DCAS Workshop, Oct. 2006

Θ(log *n*)-time Recursive Adiabatic Wired-OR Carry-Skip Adder

(8 bit segment shown)

With this recursive structure, we can do a 2^n-bit add in $2(n+1)$ logic levels.

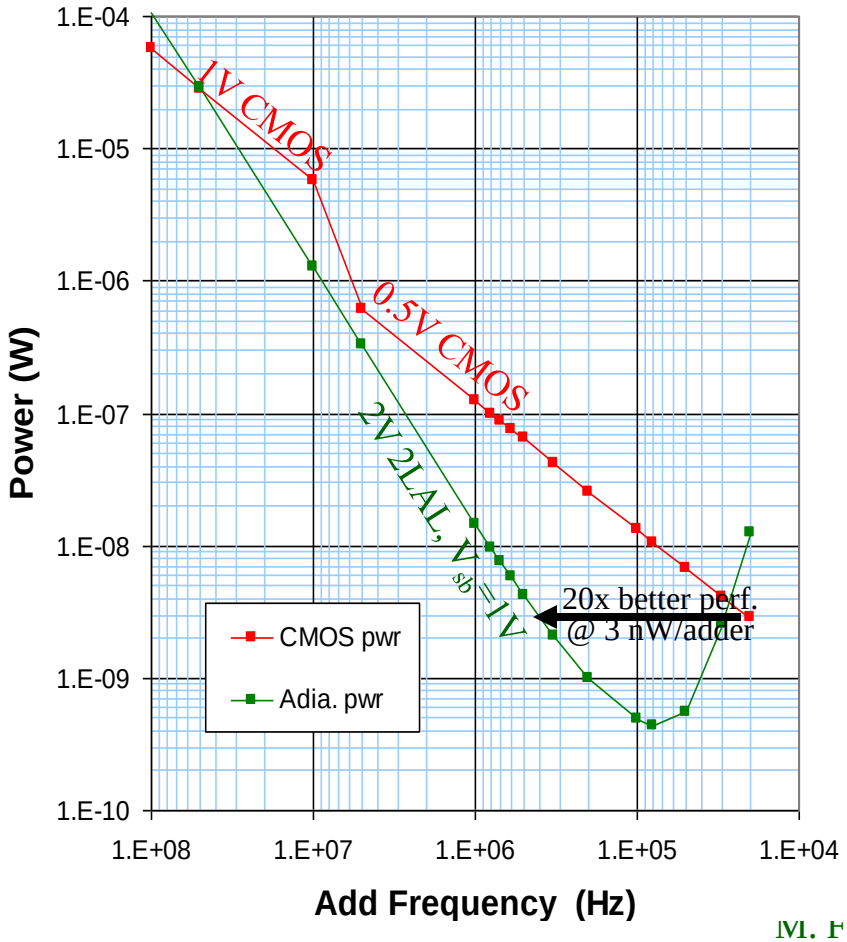
Hardware overhead is < 2× regular ripple-carry!



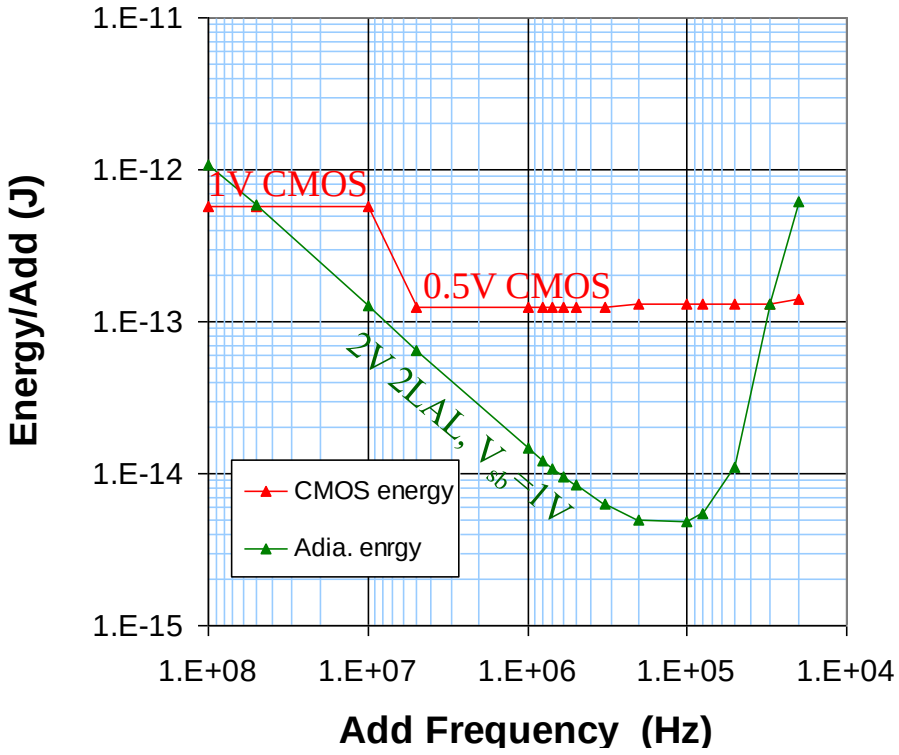
32-bit Adder Simulation Results

Further improvements may be attainable through more pipelining, carry-save adders, etc.

32-bit adder power vs. frequency



32-bit adder energy vs. frequency

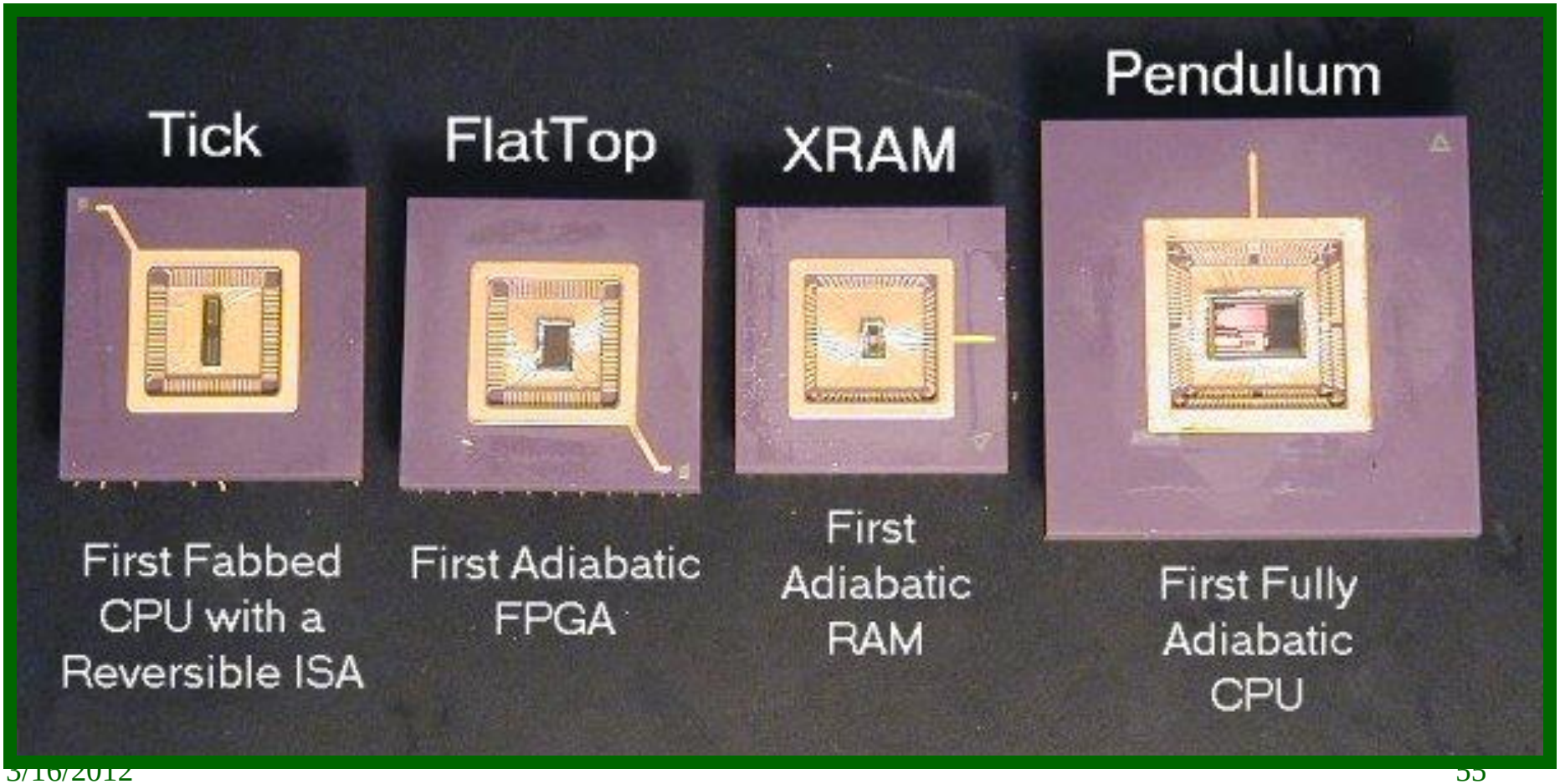


(Results are normalized to a throughput level of 1 add/cycle)

M. Frank, IEEE DCAS Workshop, Oct. 2006

Reversible and/or Adiabatic Full-Custom VLSI Chips Designed @ MIT, 1996-1999

By EECS grad students Josie Ammer, Mike Frank, Nicole Love, Scott Rixner, and Carlin Vieri under CS/AI lab members Tom Knight and Norm Margolus.



Things to Do

- Explore whether this more-general paradigm for conditionally-reversible logic primitives might be helpful in developing reversible designs in technologies other than CMOS.
  - In particular, superconducting technologies.
    - May facilitate porting designs between domains.
- Build up a much more comprehensive variety of larger functional-unit designs based on this general approach.
  - And teach more designers how to work with it!