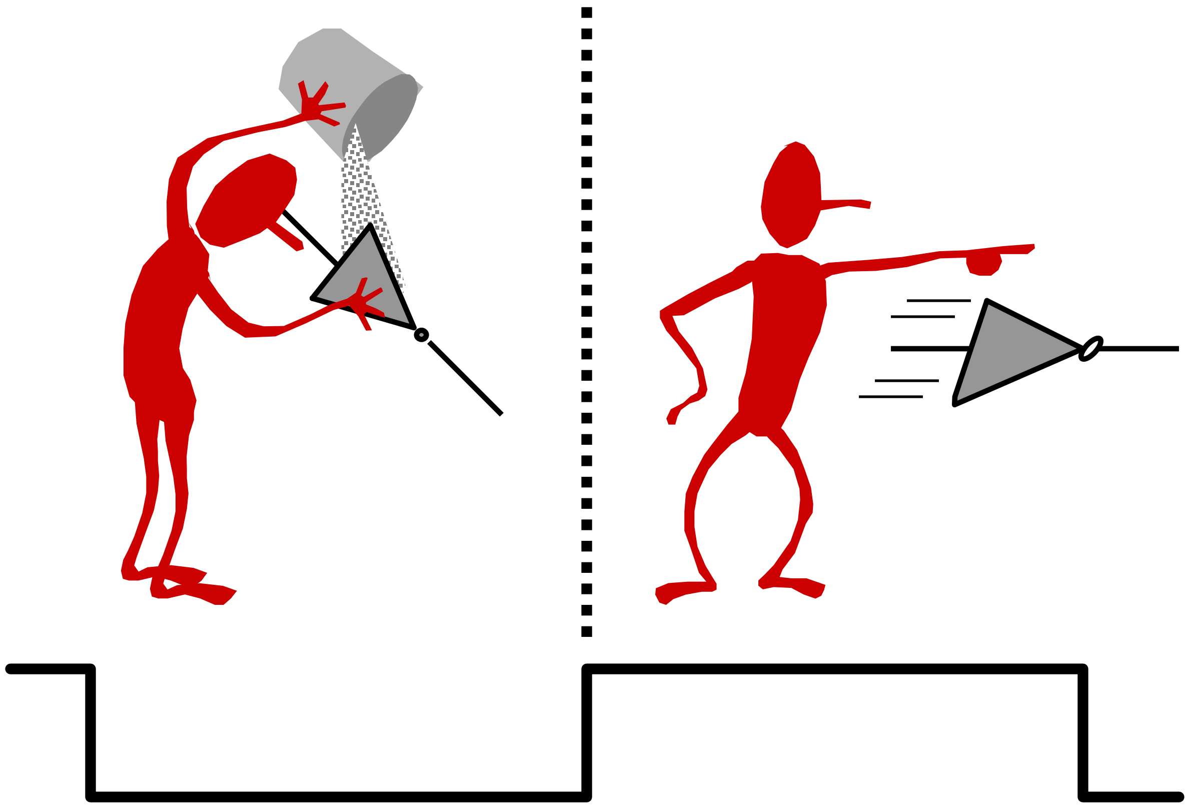
Domino Logic



Tinkering with Logic Gates

Things to like about CMOS gates:

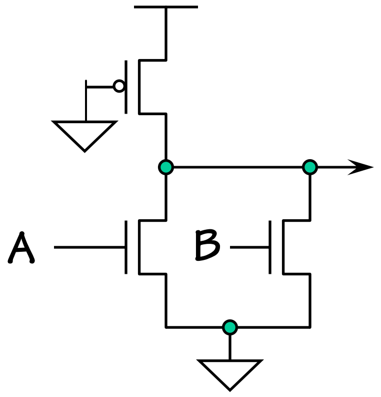
- easy to translate logic to fets
- rail-to-rail switching
  - good noise margins, no static power since fets are in cutoff
- sizing not critical to correct operation

Things not to like about CMOS gates:

- N inputs ⇨ 2N fets (i.e., one nfet and one pfet)
  - large circuit area, especially for pfets, "heavy" loading of inputs
- pfets are either large or slow relative to nfets
  - series connections can get very slow

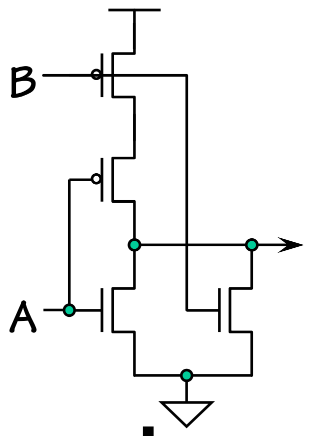
We can replace pfet pullup network with pseudo-NMOS load (pfet with grounded gate) but

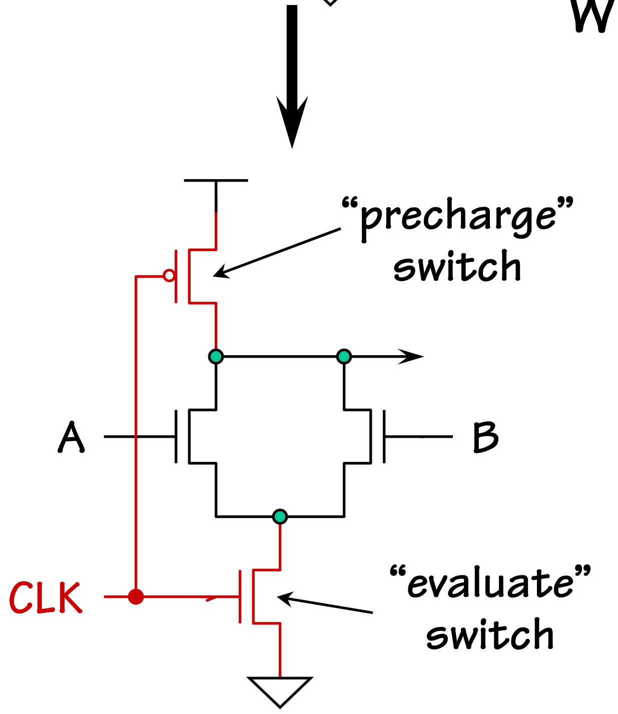
- dissipate static power when output is low
- have to make load fet small to ensure that V_{OL} is low enough to cut off nfets in next stage
  - reduces static power consumption (good!)
  - increases output rise time (bad!)



One alternative: dynamic CMOS gates

Dynamic CMOS Gates





When CLK is low

- evaluate nfet is off and precharge pfet is on
- output node is precharged to V_{DD}, other nodes may precharge to V_{DD} - $V_{th,n}$ depending on values of inputs

When CLK goes high

- evaluate nfet is on and precharge pfet is off
- output node may be discharged if inputs have configured a conducting path to GND, otherwise output node stays charged high.
- inputs must be stable before CLK goes high because once output has been discharged it won't go high again until next cycle
- for same reason, noise/glitches on inputs cannot exceed nfet threshold, a much more stringent requirement than for static CMOS gates.

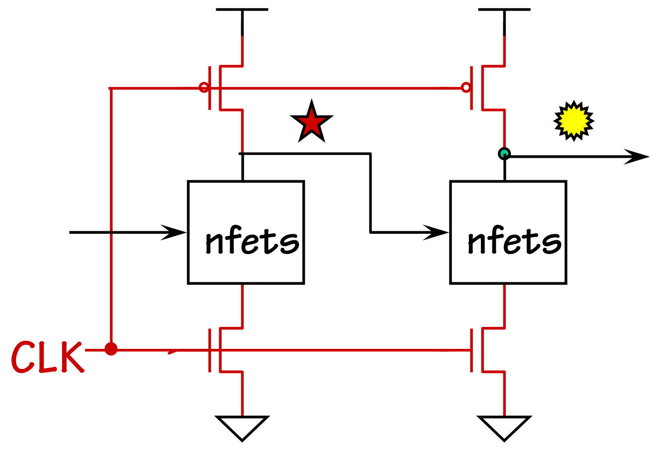
There's good news and bad news

The good news:

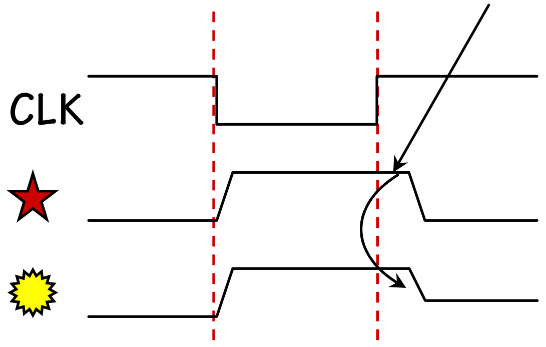
Dynamic gates are faster than static gates despite the extra "evaluate" fet in the pulldown path because of the reduction in self-loading and the elimination of the pullup short-circuit current during the first part of the output transition.

The bad news:

Dynamic gates cannot be cascaded.

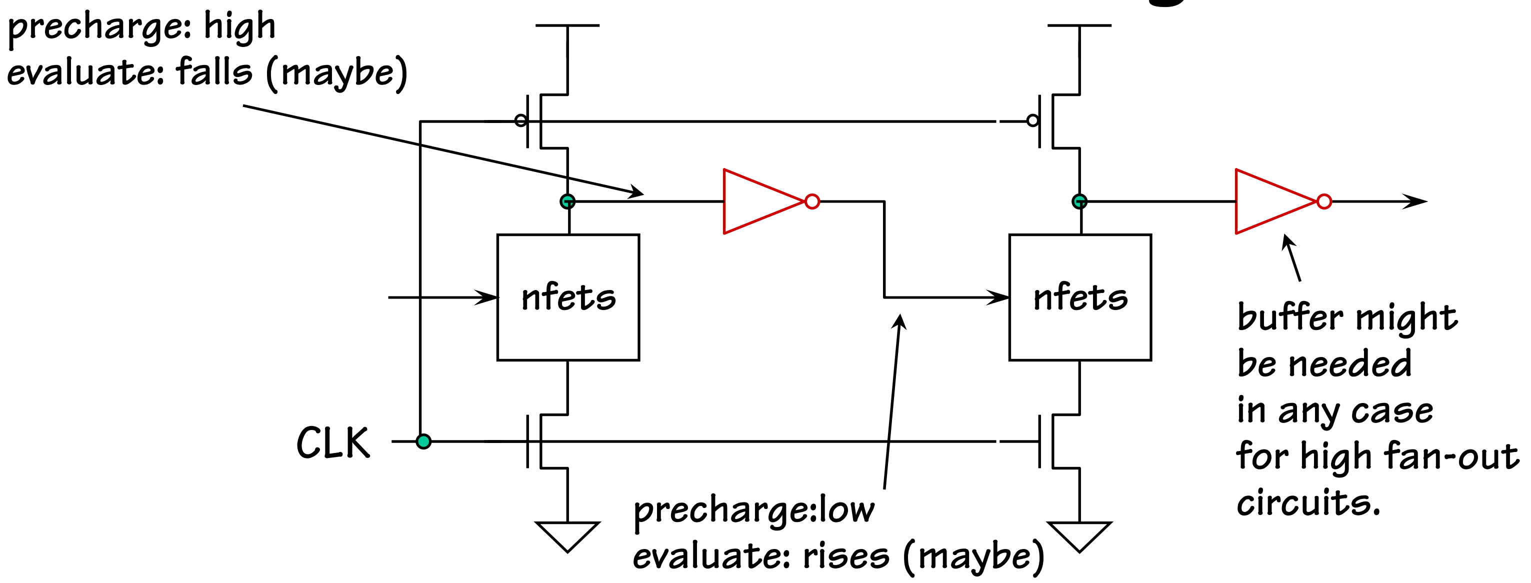


Because of finite pulldown time for node ★, node ✹ starts to discharge!



Solution – develop techniques that avoid races:
CMOS Domino logic, CMOS NORA (no race) logic

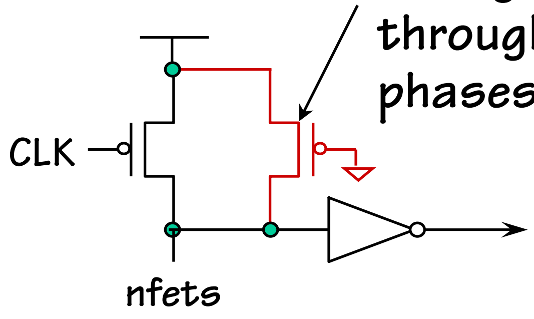
CMOS Domino Logic



When CLK is low, dynamic node is precharged high and buffer inverter output is low. Nfets in the next logic block will be off. When CLK goes high, dynamic node is conditionally discharged and the buffer output will conditionally go high. Since discharge can only happen once, buffer output can only make one low-to-high transition.

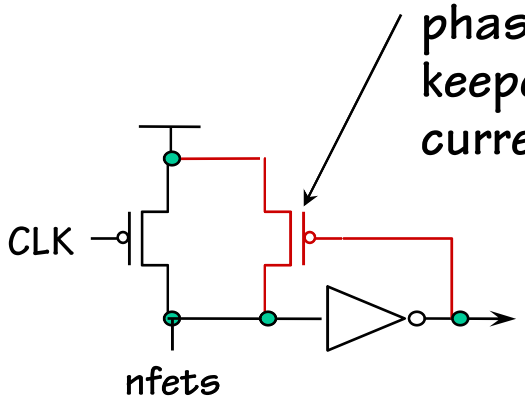
When domino gates are cascaded, as each gate "evaluates", if its output rises, it will trigger the evaluation of the next stage, and so on… like a line of dominos falling. Like dominos, once the internal node in a gate "falls", it stays "fallen" until it is "picked up" by the precharge phase of the next cycle. Thus many gates may evaluate in one eval cycle.

More Domino-style Circuits



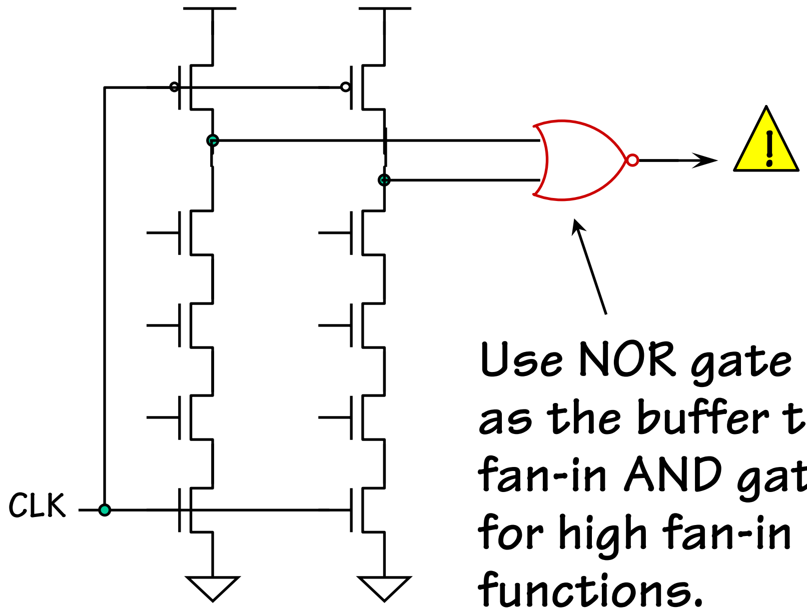
weak pfet "keeper" keeps dynamic node pulled high during evaluate phase if it's not being pulled down through nfets ⇨ gate is static in both clock phases.

"latching" pfet acts like keeper above unless dynamic node gets pulled down during evaluate phase. When buffer output goes high it switches keeper off saving static power. Good for leakage current problems...



Note that you can put an *even* number of static gates after the inverter and before the next domino gate.

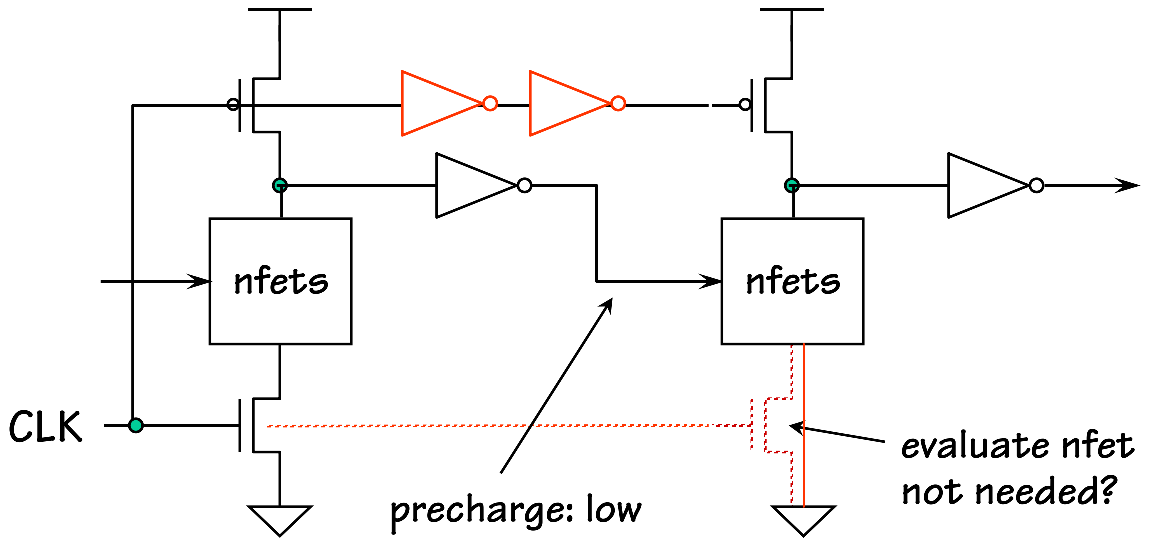
High-fanin Domino Circuits



Be careful of capacitive coupling to dynamic node (see later slide).

Use NOR gate instead of inverter as the buffer to make a faster high fan-in AND gate. Same trick works for high fan-in OR or MUX functions.

Optimizing Domino Logic (I)



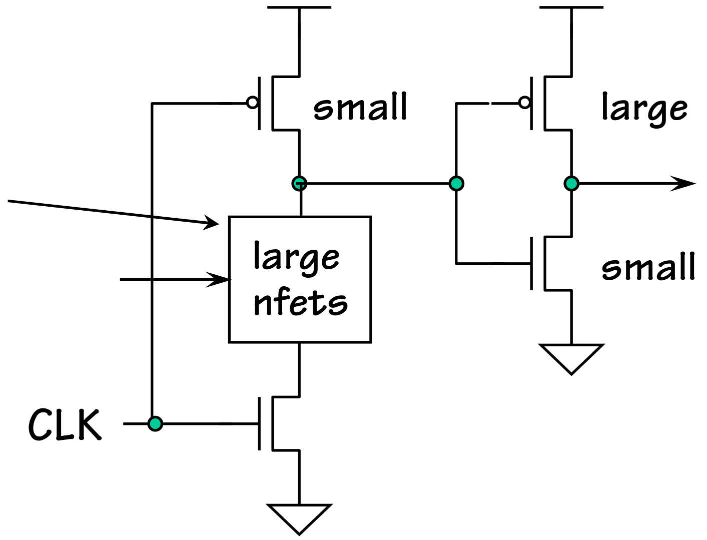
Since domino gate outputs are low during the precharge phase, gates which have only domino output nodes as inputs don't need the "evaluate" nfet since all the nfets in the pulldown will be off anyway.

But remember: if evaluate nfet is removed, precharge will "ripple" through cascaded gates just like evaluates do. Maybe only remove for gates where nfet stack is tall (i.e. resistive) enough that pullup will start to "win" anyway before ripple reaches gates and turns off pulldowns. To avoid short-circuit current: delay arrival of precharge signal to downstream gate.

Optimizing Domino Logic (II)

In domino logic circuits we want evaluate to happen as quickly as possible. We can tweak fet sizes to optimize evaluate speed.

Some designers also "grade" the sizes of the nfets, smallest at the top (increase in R offset by decrease in C)



If we make the nfet in the output inverter much smaller than the pfet then

- the load on the internal node decreases, and
- the switching threshold of the inverter increases

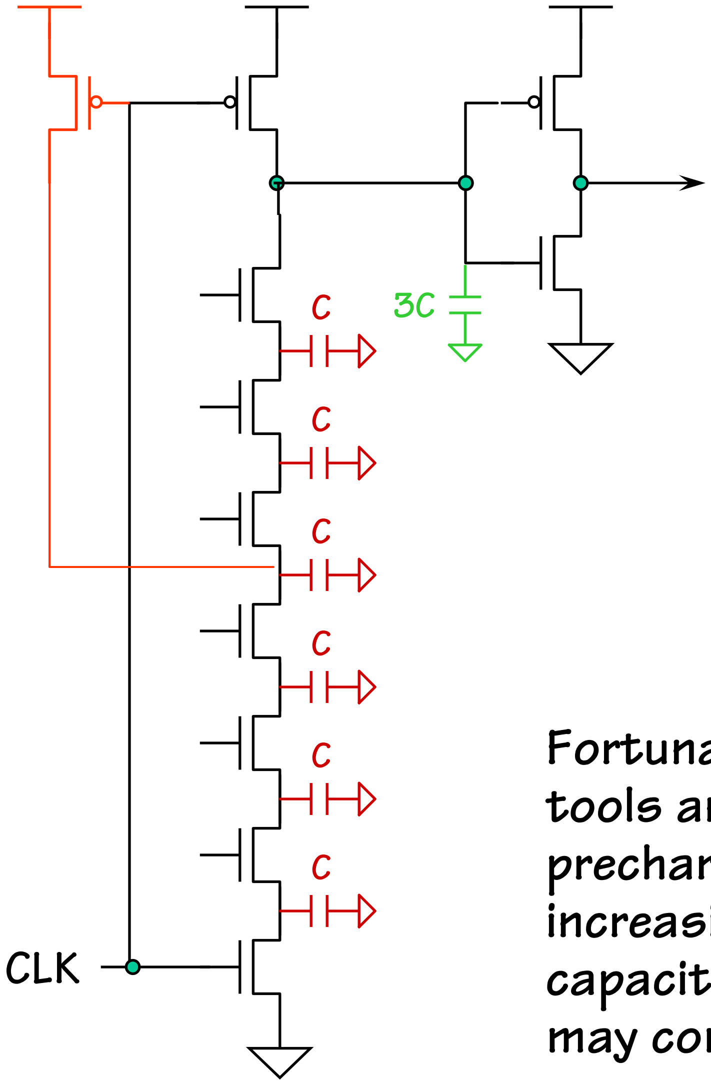
Both effects make the gate evaluate sooner. If large >> small, the gate delay can be cut almost in half! However, the other edge is very slow, so ripple precharge is a problem.

"Flies in the ointment"

There are a few "little" difficulties:

- "charge sharing" between nodes in the pulldown network and the dynamic node can unintentionally reduce the voltage of the dynamic node enough to switch output buffer
- the addition of the output inverter makes domino gates non-inverting. One can often design around this limitation, but some circuits cannot be implemented solely using domino logic unless both polarities (true and complement) of the inputs are available. If both polarities of inputs are available then we can generate both polarities of internal signals with two domino gates so subsequent stages will have both polarities of their inputs available too.

Charge Sharing



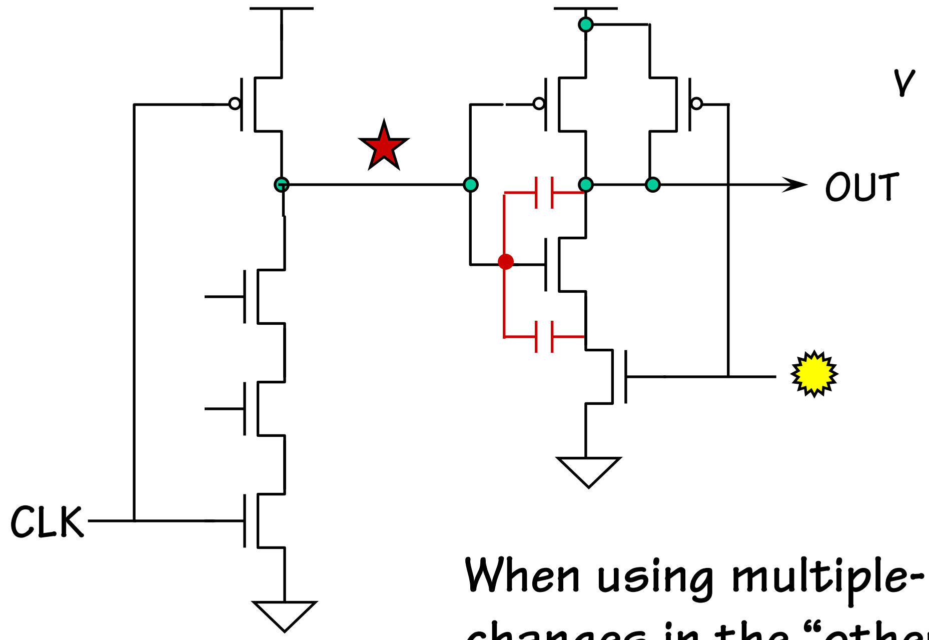
S'pose the dynamic node had been discharged during the previous evaluate cycle. Then during precharge, all the intermediate nodes in the pulldown chain will remain discharged while the dynamic node is precharged. When CLK goes high, the voltage on the dynamic node goes to

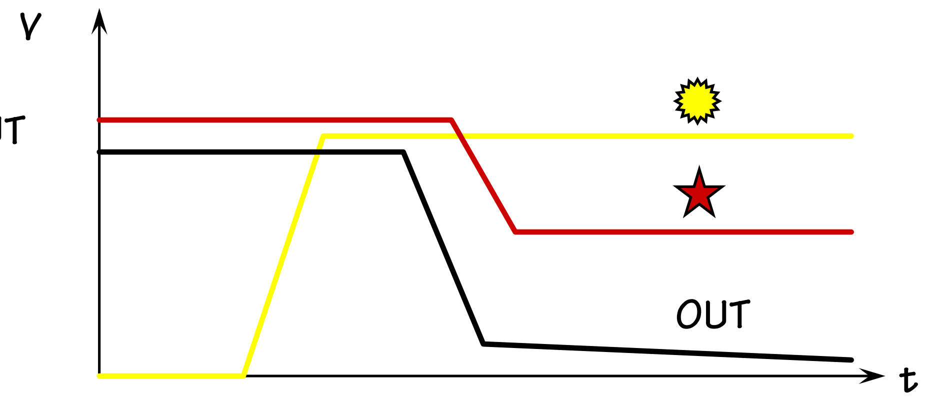
$$\frac{3C}{3C + 6C} V_{DD} = .3V_{DD} = 1.5V$$

which is low enough to switch the output inverter.

Fortunately this situation is easily detected by CAD tools and can be resolved by (1) adding additional precharge devices to intermediate nodes or (2) increasing size of output buffer which will increase capacitance of dynamic node (faster output buffer may compensate for larger internal capacitance).

Capacitive Coupling

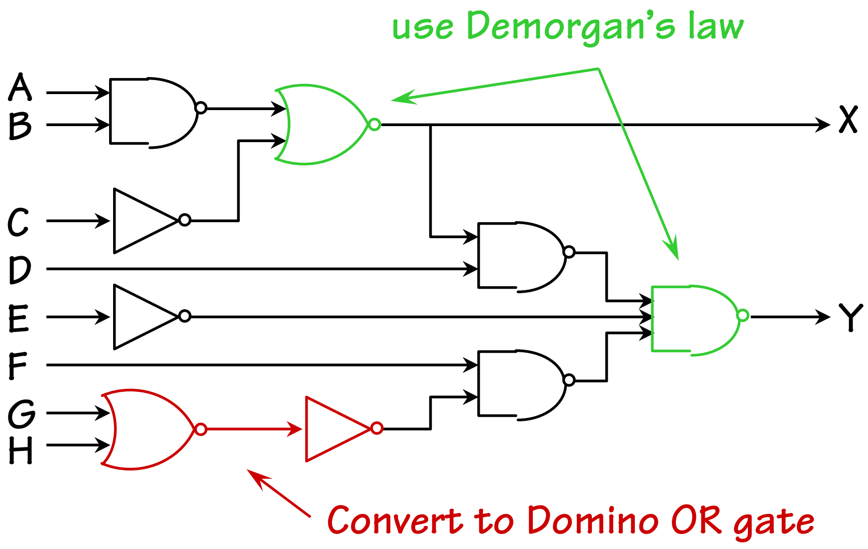




When using multiple-input gates as the Domino buffer changes in the "other" input during evaluate phase can cause dynamic node voltage to sag due to capacitive coupling, leading to unintended transition on OUT

Coupling can also occur between other signal wires and long dynamic nodes (e.g., ones that span multiple bits in a datapath). Solutions: on long routes add "twists" to avoid continguous routes or route dynamic signals between mutually exclusive or complementary signals.

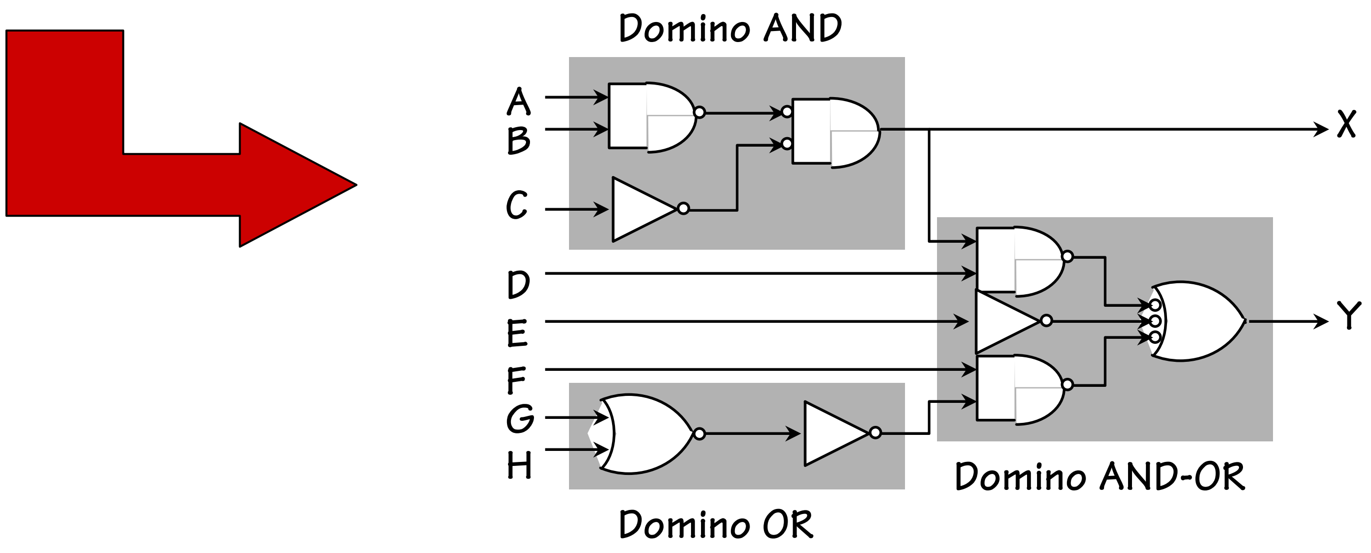
Domino Logic Design (I)



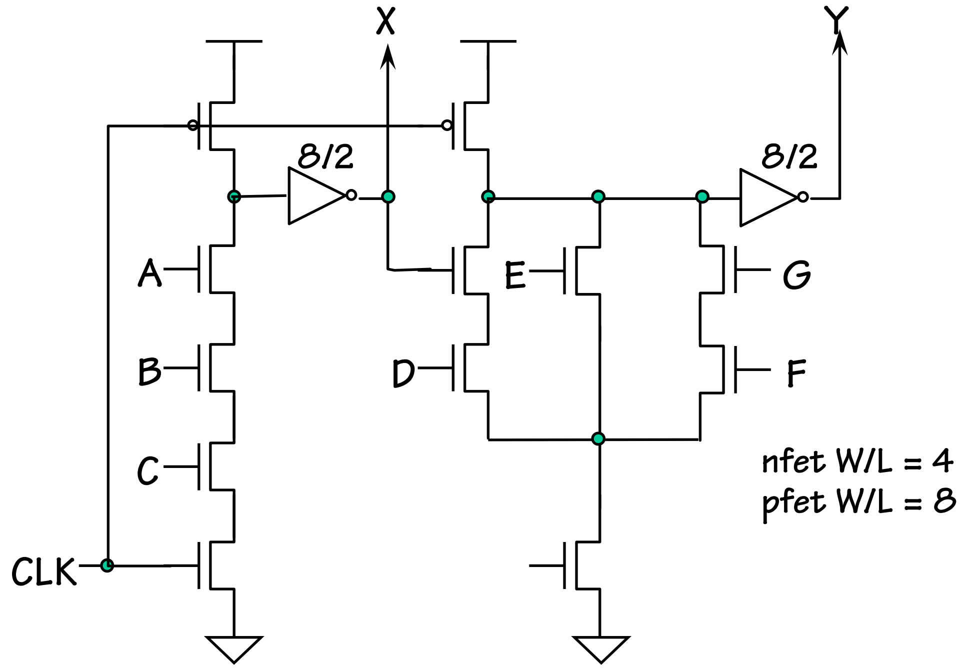
To convert to Domino-style design we need to create schematic that uses non-inverting gates:

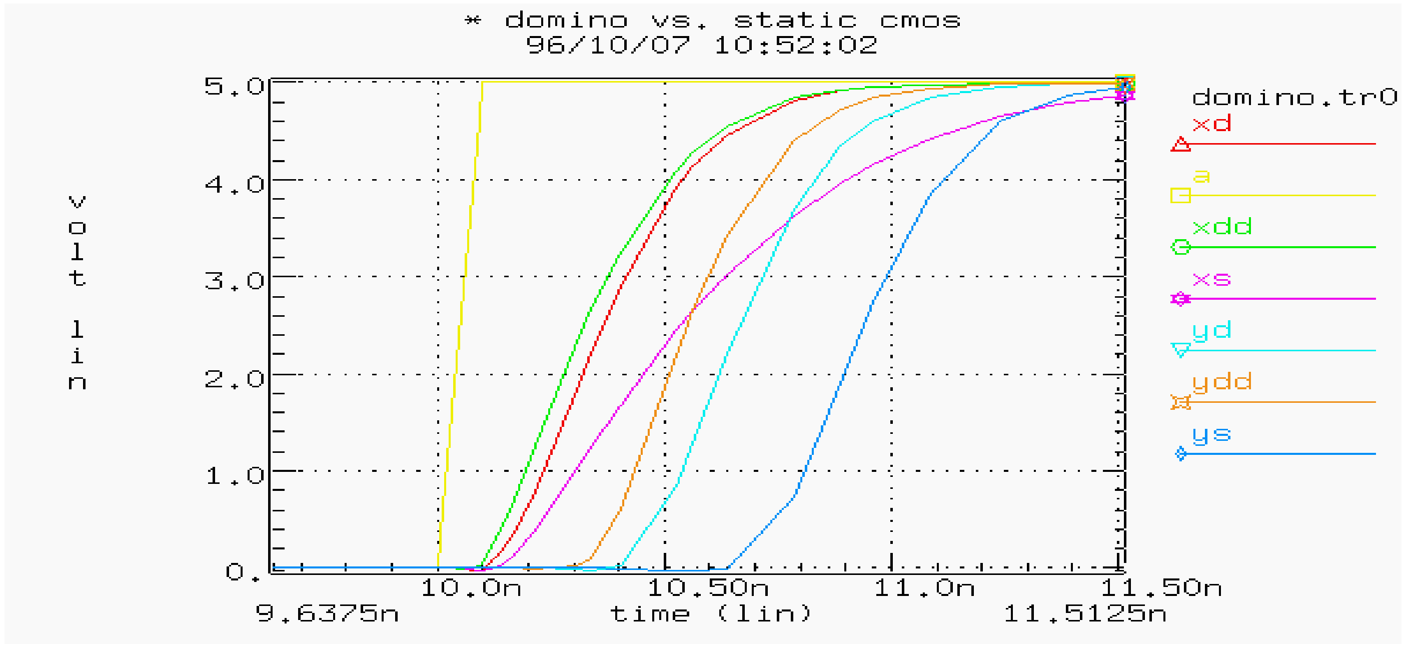
(1) look for CMOS gates followed by inverter

(2) use Demorgan's Law to create non-inv gates



Domino Logic Design (II)

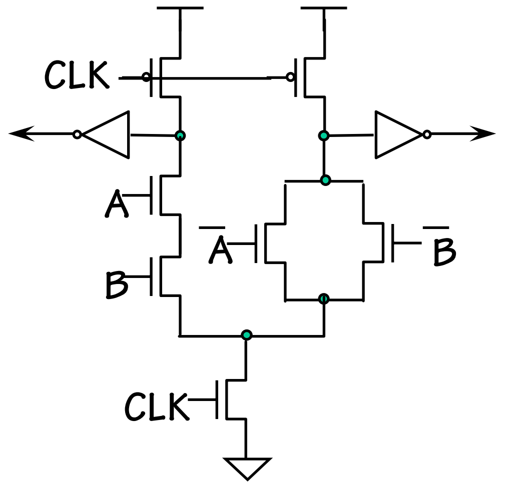


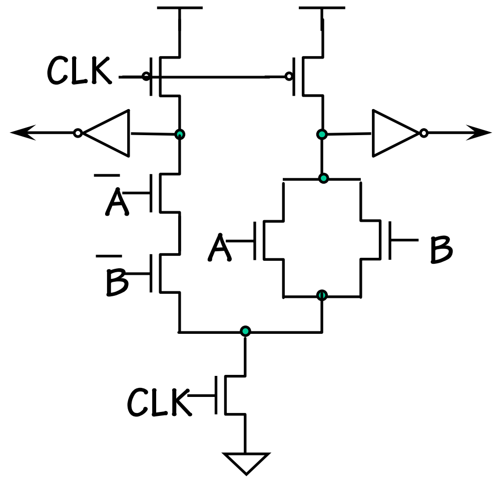


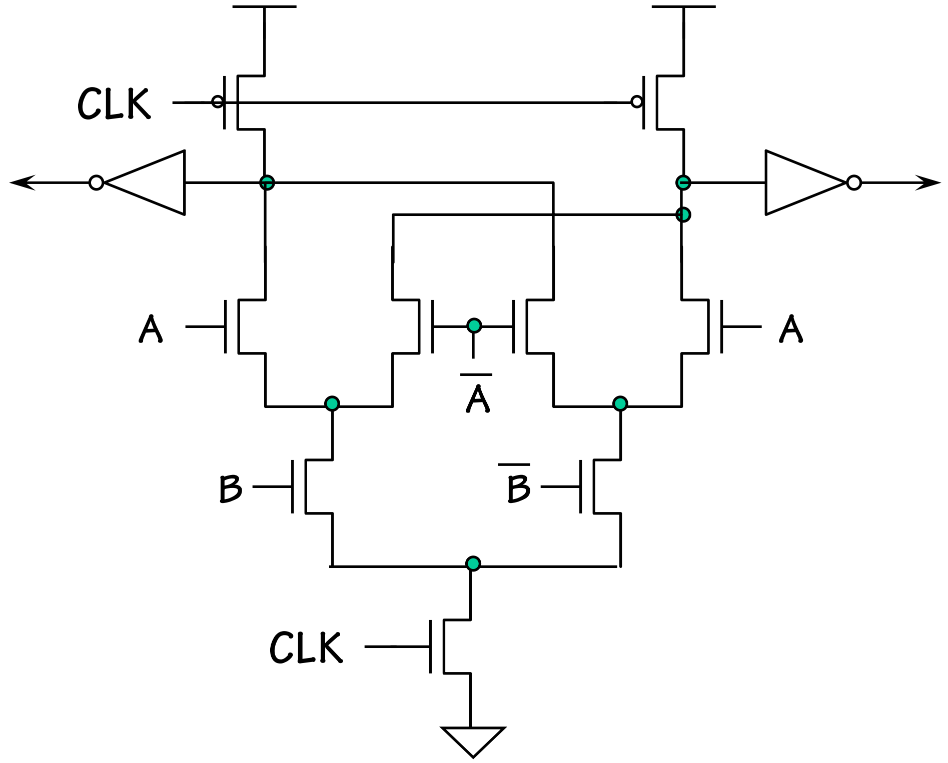
s = static
d = domino (W/L = 4)
dd = domino (W/L = 8)

Dual-rail Domino Logic

Domino circuits that generate both polarities of output







Multiple-output Domino

Why stop at complementary outputs? There are interesting multiple-output functions where there is a lot of sharing of nfets in the evaluate logic. For example, in a carry-lookahead adder

$G_i = A_iB_i$

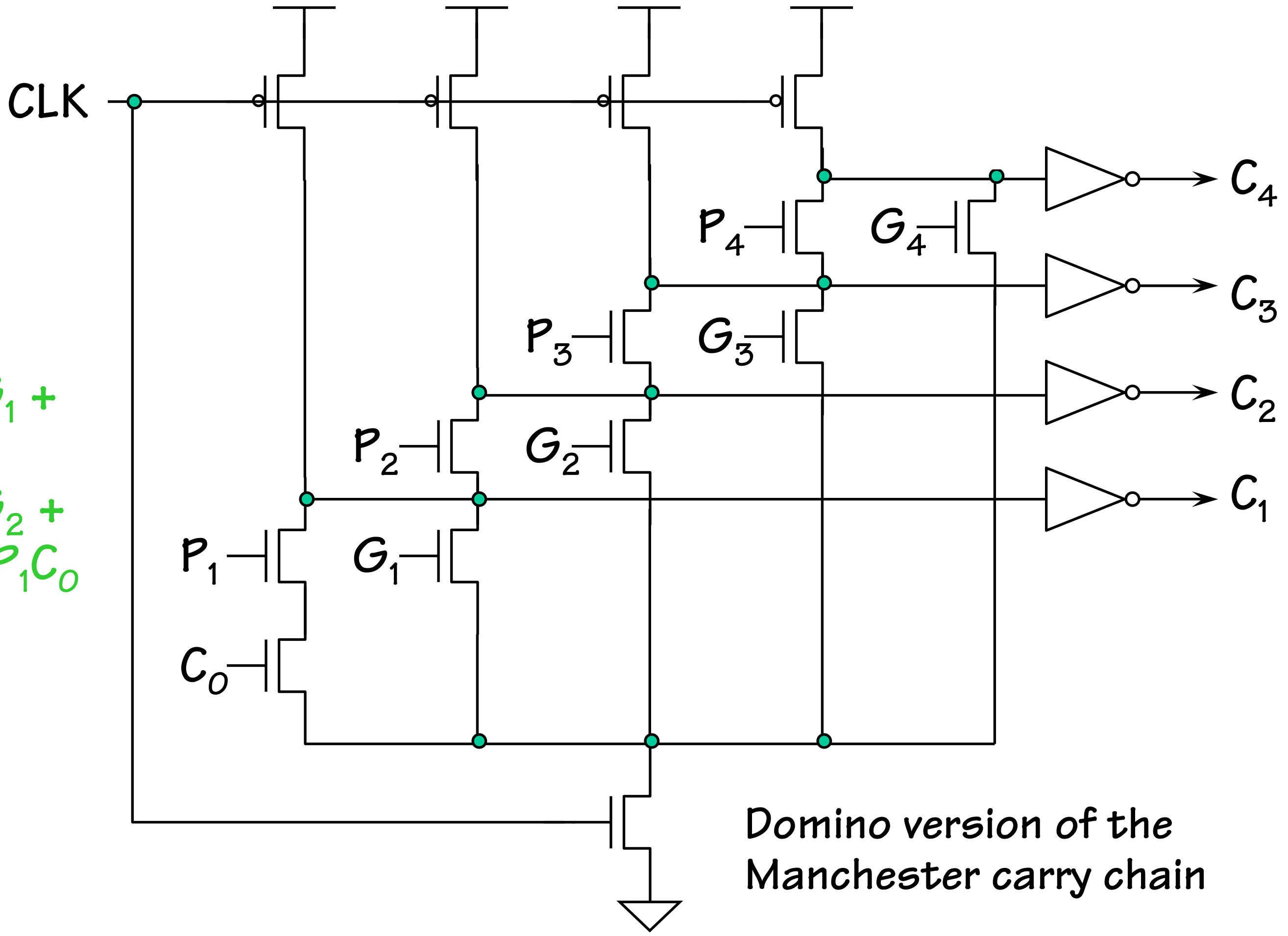
$P_i = A_i + B_i$

$C_1 = G_1 + P_1C_0$

$C_2 = G_2 + P_2G_1 + P_2P_1C_0$

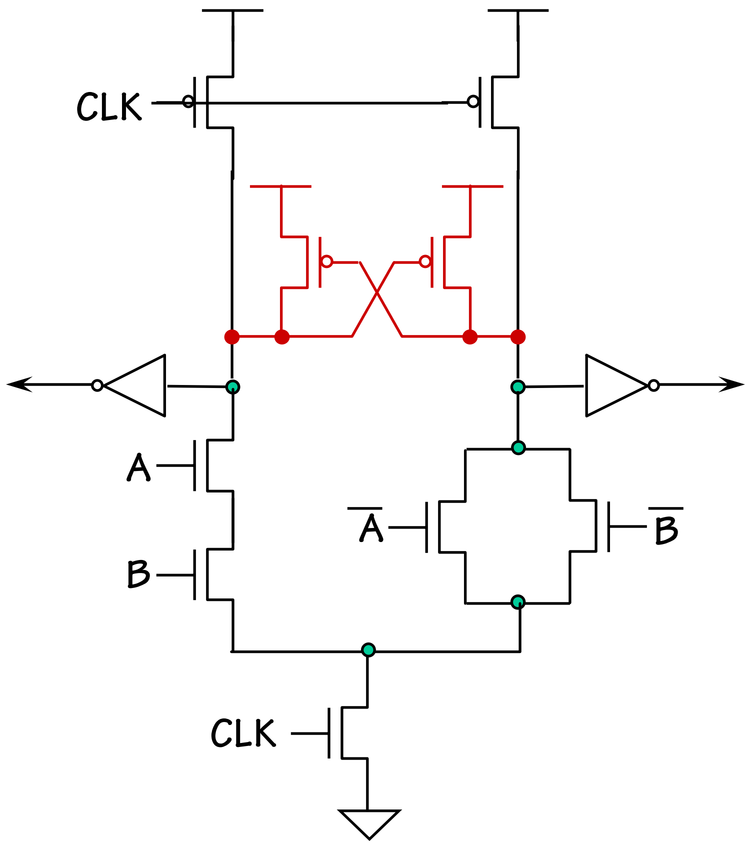
$C_3 = G_3 + P_3G_2 + P_3P_2G_1 + P_3P_2P_1C_0$

$C_4 = G_4 + P_4G_3 + P_4P_3G_2 + P_4P_3P_2G_1 + P_4P_3P_2P_1C_0$



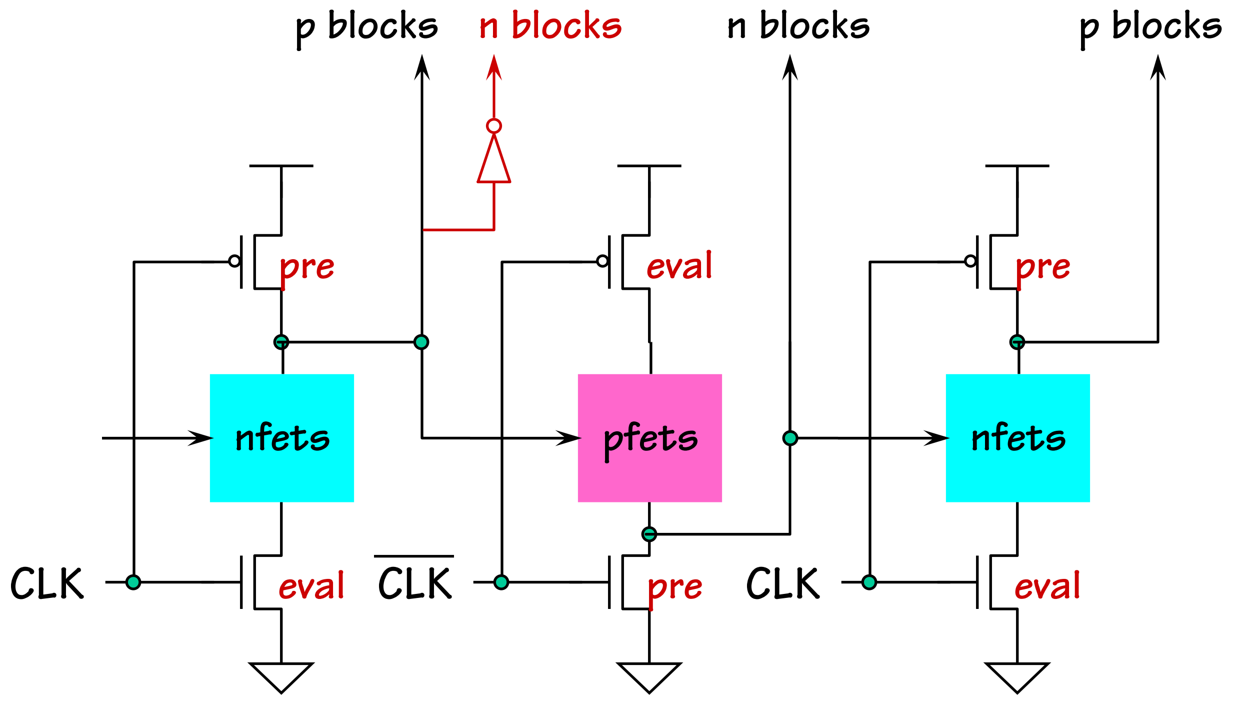
Domino version of the Manchester carry chain

Dual-rail "Keeper" Circuit



The cross-coupled pfets serve as "keepers" for the output which is high making the gate static rather than dynamic! During precharge both keepers are off; during the evaluate phase, the output that goes low switches on the keeper for the output that is staying high. Really solves capacitive coupling problems with dynamic logic in datapaths.

CMOS NORA Logic



If we turn a dynamic gate "upside down" and use pfets to build the logic block, we get a logic gate that "precharges" low and "discharges" high. By using these gates in an alternating sequence with regular nfet dynamic gates we can eliminate the race problem we had with nfet-only dynamic gate sequences and hence we don't need the buffer inverter present in domino gates.

Removing the buffer is a mixed blessing since we may need it for drive reasons and to keep compatibility with other domino gates. It also makes NORA logic very susceptible to noise since during the evaluate phase all information is stored dynamically.

Dynamic Logic Summary

Advantages of dynamic logic:

- smaller area than fully static gates
- smaller parasitic capacitances hence higher speed
- reliable operation if correctly designed.

This makes dynamic logic a good choice for those parts of a circuit where the extra engineering investment is justified, e.g., along the critical timing paths.

Concerns:

capacitive coupling to dynamic nodes

charge sharing with dynamic nodes

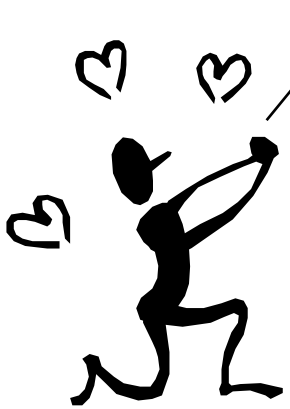
subthreshold leakage currents in eval logic

minority carrier injection and latchup

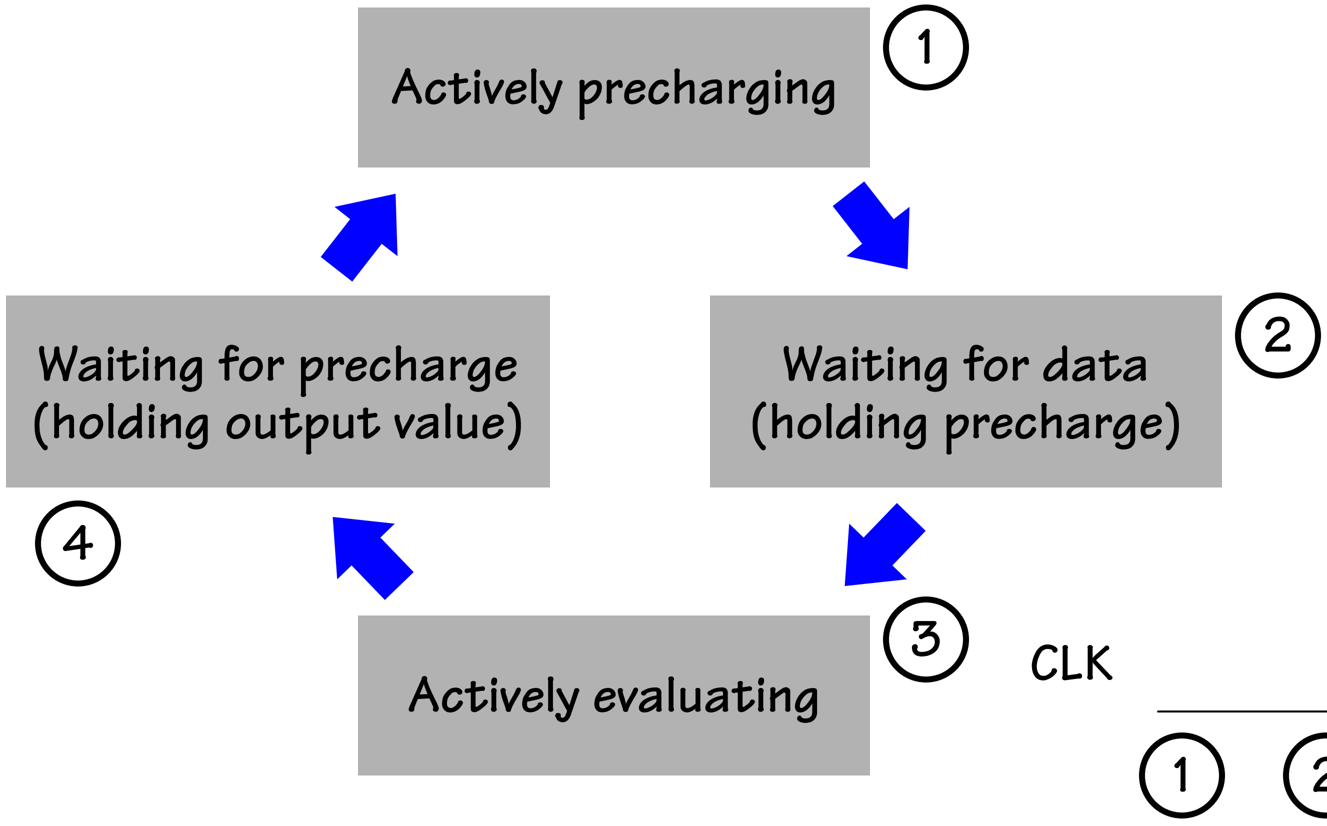
alpha particle immunity

vdd/gnd noise and resistance

Engineers who like this sort of design will find this the sort of design they like!

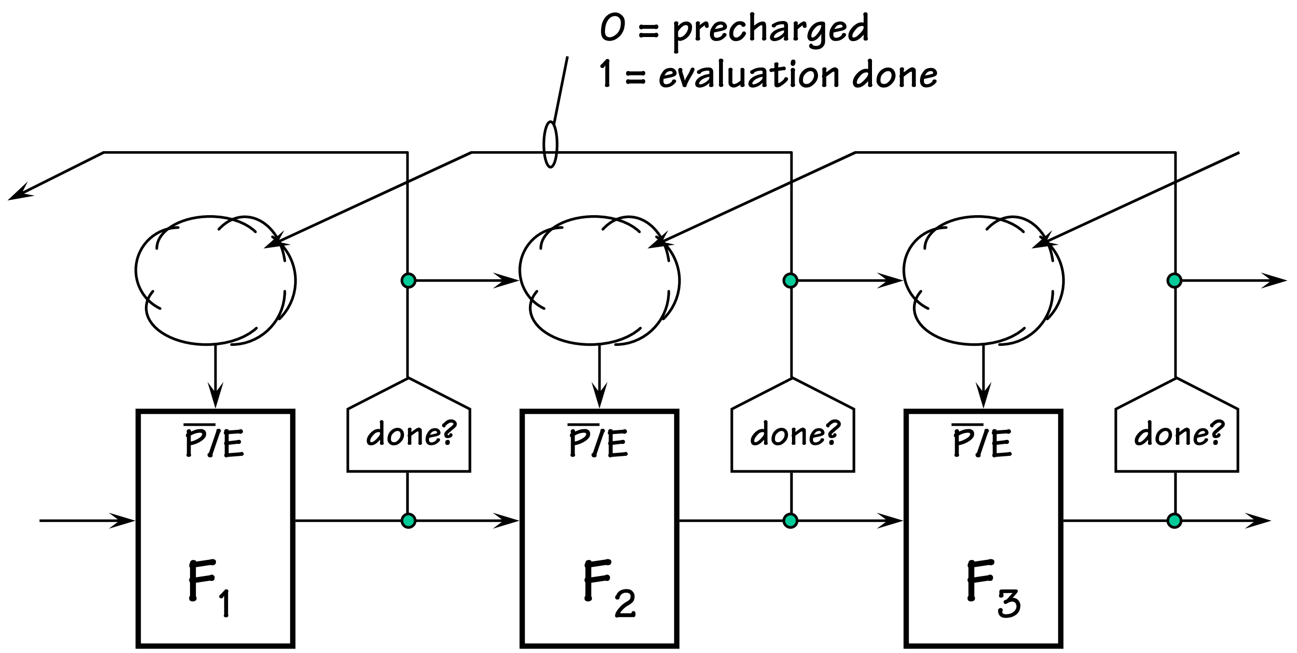


Domino Life Cycle


Actively precharging
1
Waiting for data (holding precharge)
2
Actively evaluating
3
Waiting for precharge (holding output value)
4
CLK
1
2
3
4

The "9 O'clock" state is very interesting: once a Domino gate has finished evaluating, the gate's immediate predecessors can start to precharge (forcing the gate's inputs low) without affecting the value of the gate's output. The gate is acting as latch so long as its predecessors don't start another evaluate cycle. Perhaps we can build a pipeline of domino stages where each stage serves as both logic and latch depending on where it is in its cycle. Need to have each stage supply its own precharge/evaluate timing dependent on what its neighbors are doing...

Self-timed Pipelines



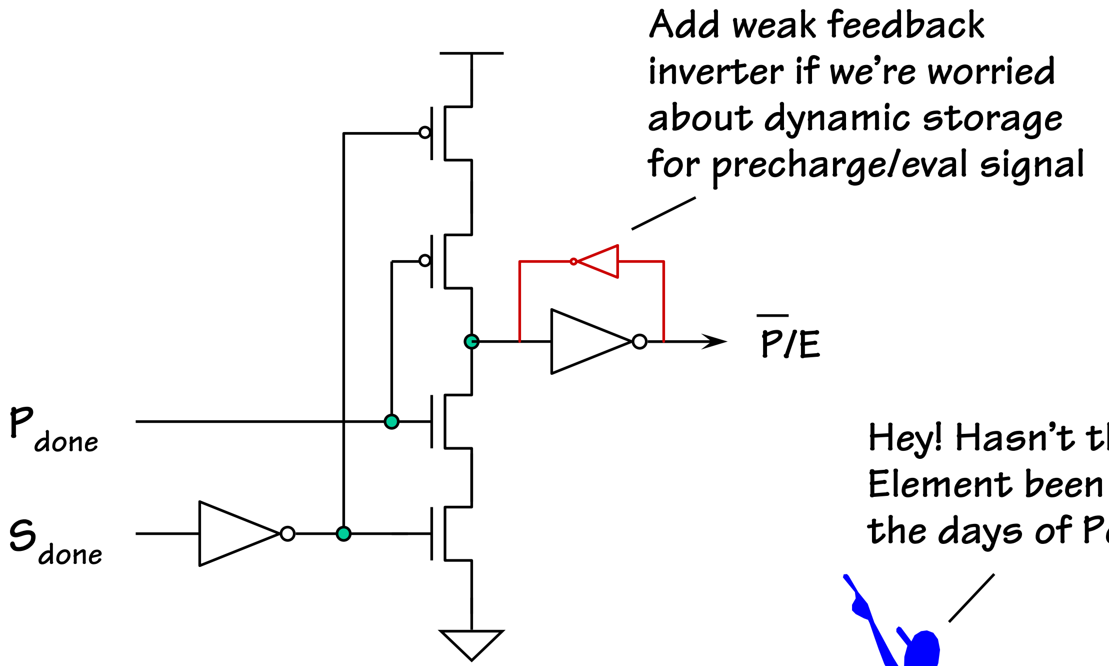
Simplest correctness rules:

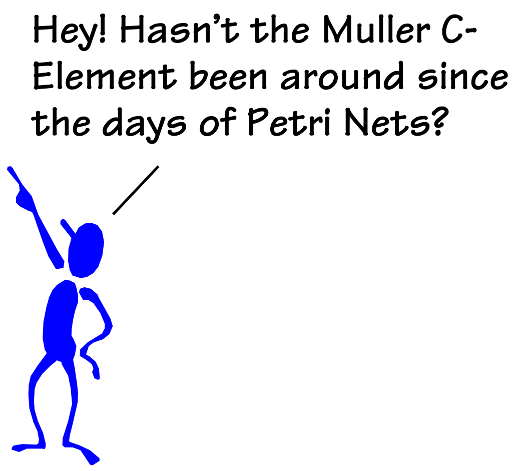
- a stage only precharges when both
  - (a) its successor has finished evaluating (it's done with our values) ← $S_{done} = 1$
  - (b) its predecessor has finished precharging (old values are gone so we can't use 'em twice!) ← $P_{done} = 0$
- a stage only evaluates when both
  - (a) its successor has finished precharging (our new output won't affect its stored value) ← $S_{done} = 0$
  - (b) its predecessor has finished evaulating (there are new inputs for us to consider) ← $P_{done} = 1$

So, what logic goes in the clouds? And how do we build the "done?" boxes?

Muller C-Element

The Muller C-Element is the "AND" gate for self-timed logic because it changes its output only after both inputs have changed. As shown here, it's an elegant implementation for both sets of rules on the previous slide.


Add weak feedback inverter if we're worried about dynamic storage for precharge/eval signal
$\overline{P}/E$
P_{done}
S_{done}


Hey! Hasn't the Muller C-Element been around since the days of Petri Nets?

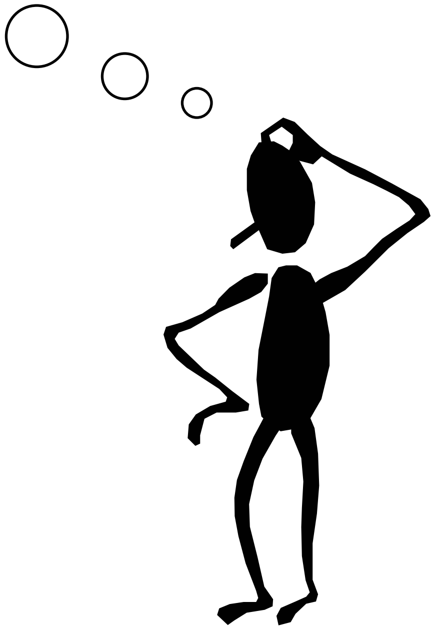
Completion Detectors

Self-timed logic

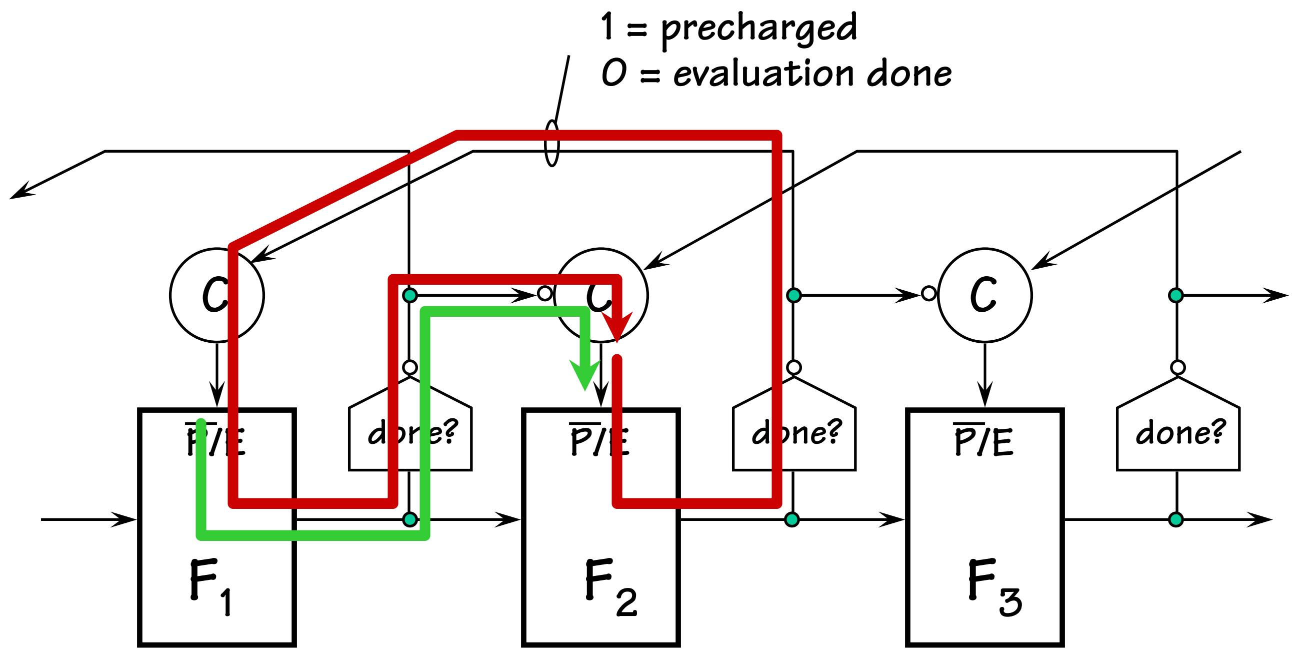
use dual-rail signaling (*i.e.*, two wires) to encode

reset (not yet evaluated)	00
ready with value 0	01
ready with value 1	10

and then build handshake logic that starts next stage when current stage is done and next stage has completed its previous computation and delivered its values...



Self-timed Pipeline Latency



Propagation through self-timed pipelines is constrained in both directions:

In the forward direction by how long it takes for the evaluate edge in one stage to trigger the evaluate edge in the next stage:

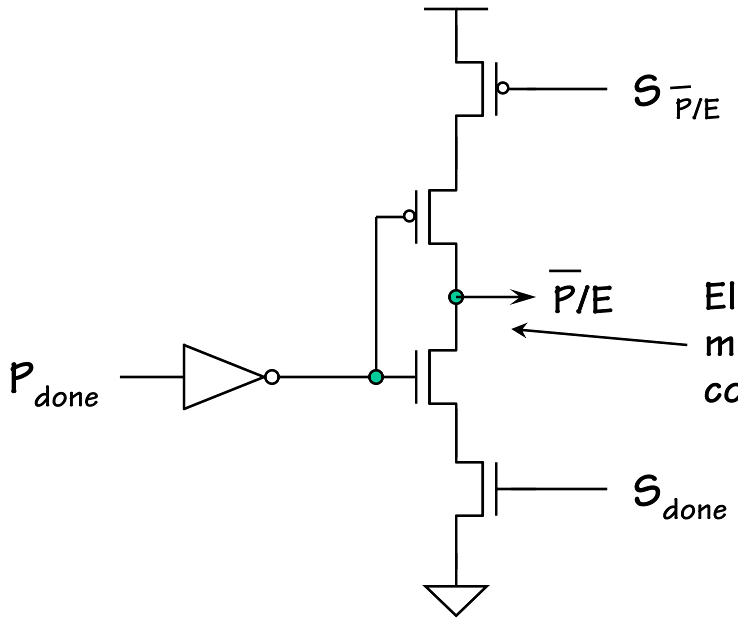
$$L_F = t_{F\uparrow} + t_{D\downarrow} + t_{C\uparrow}$$

In the reverse direction by how long it takes for the precharge in one stage to trigger a new evaluate in the stage after first evaluating the previous stage (remember not double count!):

$$L_R = 0.5^*(t_{C\downarrow} + t_{F\downarrow} + t_{D\uparrow} + t_{C\uparrow} + t_{F\uparrow} + t_{D\downarrow})$$

Further Improvements

We don't have to delay evaluation until successor has finished its precharge (signaling that it's finished with our values). We can just check that successor has started precharging… Even with this improvement, the correct sequencing will still happen for any combination of precharge and evaluate times for all the gates. We can modify the control element like so:

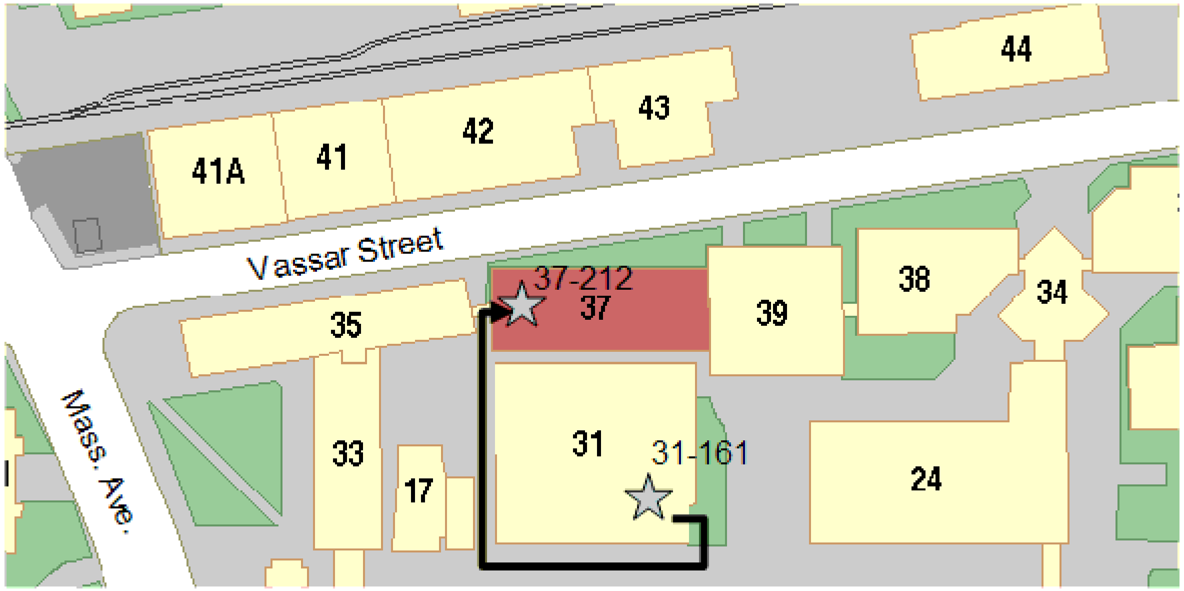


Eliminate the "extra" inverter for good measure and use dynamic storage as control element memory

We're going to stop here, but there are other improvements that can be made. Hint: do we have to wait until the predecessor is done computing new values before starting our eval? etc., etc., etc.

Quiz #1 Info

- Friday, 10/11, 80 mins: 11:05 – 12:25
- Two rooms: 31-161 [A – Li], 37-212 [Lu – Z]
- Open book (suggestion: bring lecture handouts)
- Calculator may come in handy



Getting to 37-212: Arrow shows route from current classroom (31-161) to 37-212. Enter 37 from street level at west end, go up stairs to second floor, classroom entrance is just to the left as you exit the stairs.