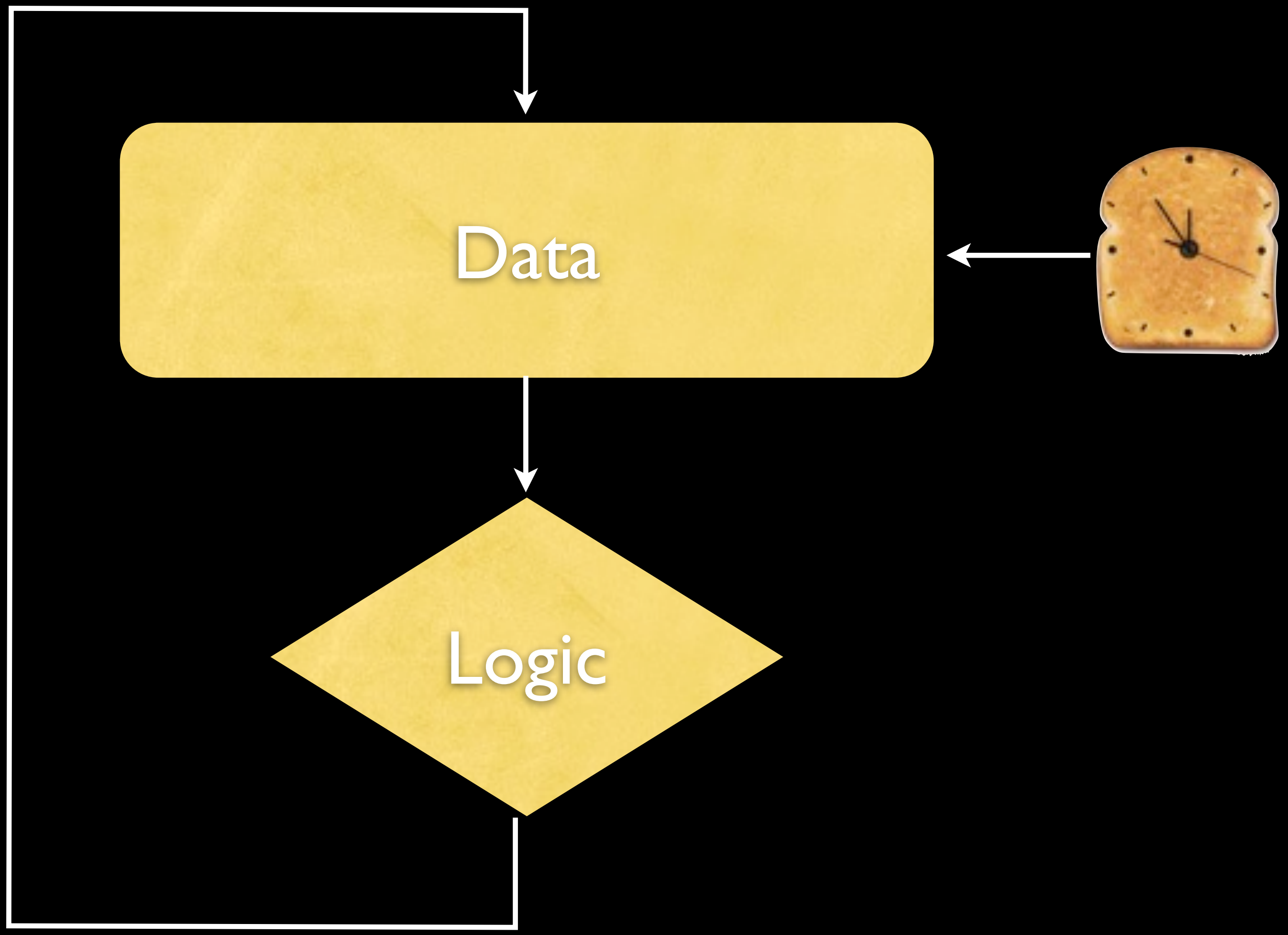
What a computer is



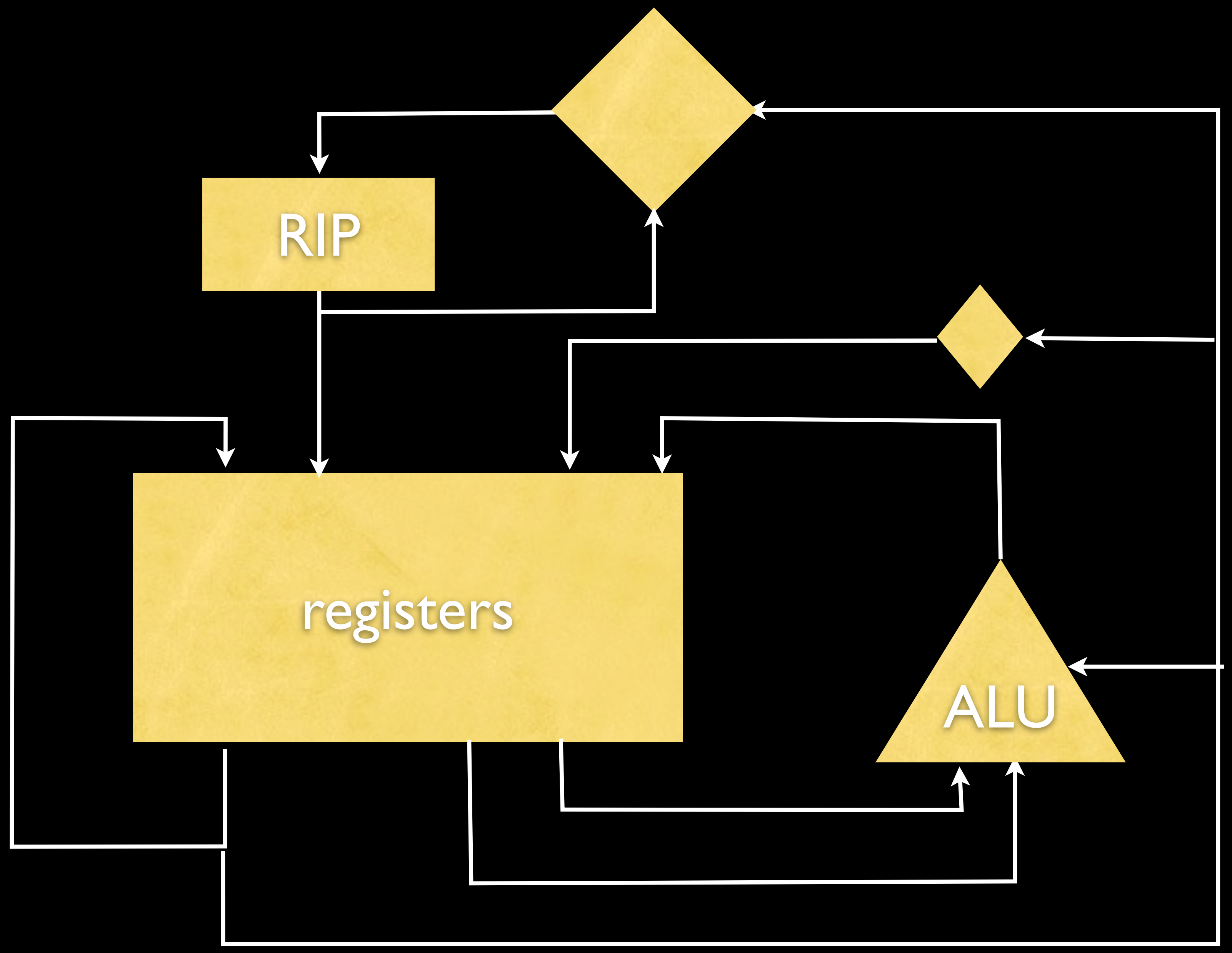
Computers can be really quite simple

- Need state (memory, registers, etc)
- Need some logic to compute (ALU)
- Need some instructions
  - *and here is the key:* instructions **are** data; they are just bits
- Need some wires to connect it all together
- Need a **clock** to precisely control when data is modified

So let’s “build” one

- State: Let’s keep it simple, and just use a big register file.
- Logic: We’ll keep it simple and use a single ALU with a giant MUX at the end to select the operation.
- We’ll keep 1 register off to the side as something special and we’ll call it the %rip
  - We’ll do a little custom logic around %rip too, in order to support branches
- We’ll use a single clock entering all the register bits every cycle, and use selective write-enable on them

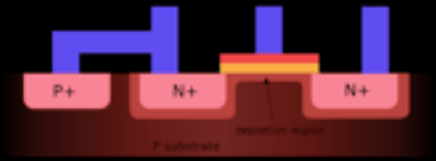
The 351 Workhorse :-)



Components

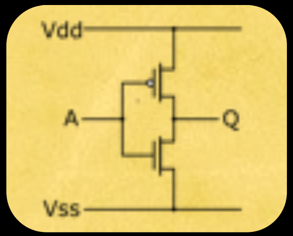
- For now we are not going to focus on being efficient; our focus is on providing a confidence in you that it *can be done*.
  - Efficiency will come later... but under the hood computers are not all that efficient (100's of Watts have to go somewhere :-)

Logic



A transistor is a switch
CMOS has 2 types NPN and PNP

Vdd (positive)

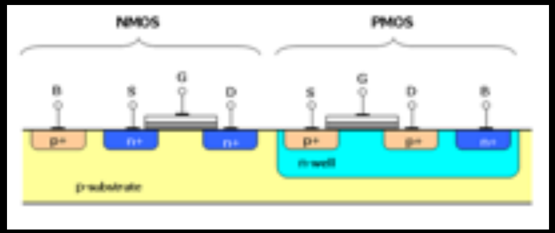


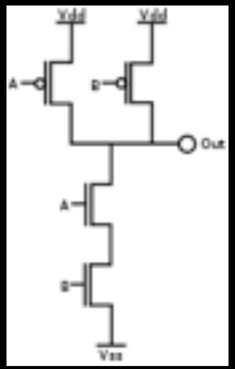
input

output

Ground (negative)

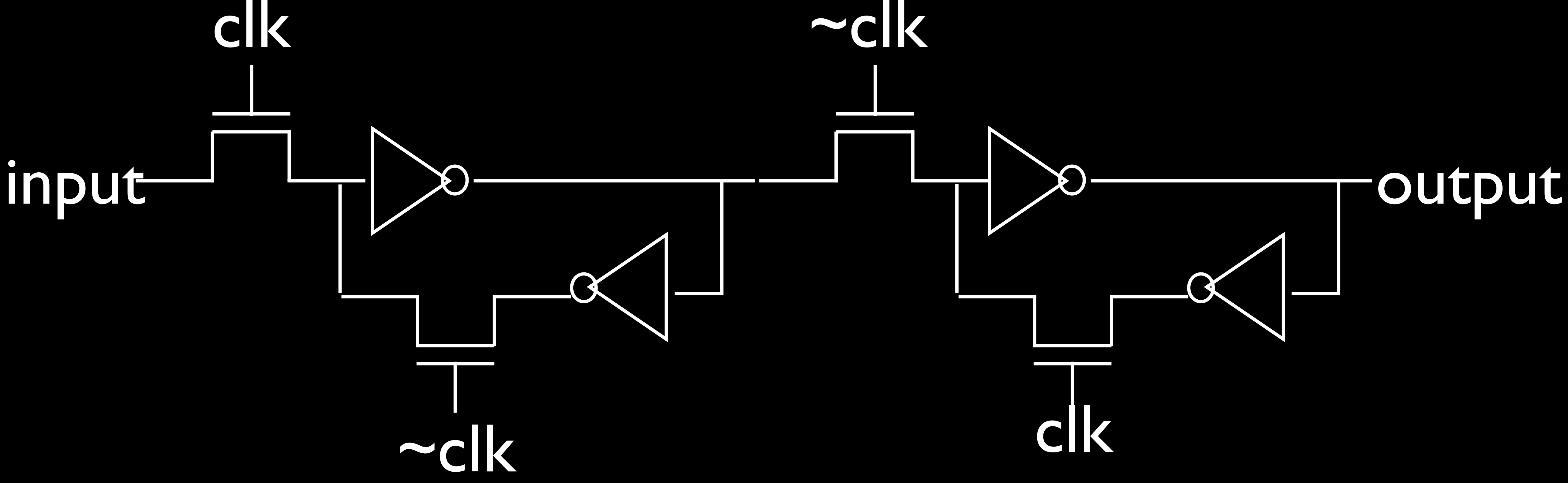
An inverter



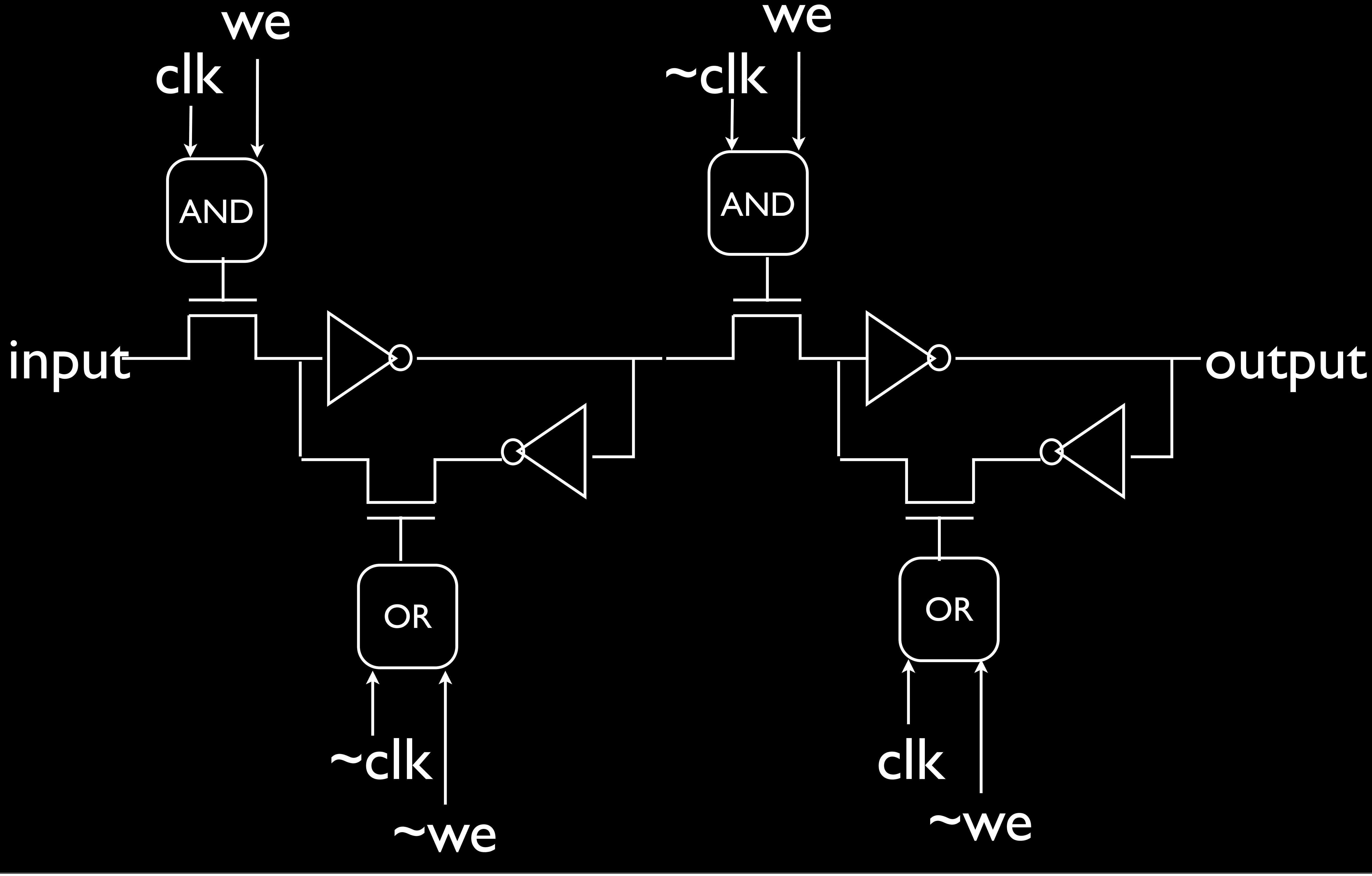


A NAND gate.
Theoretically NAND gates are "universal" and all other combinatorial circuits can be synthesized from them. Of course, it is not the most efficient way to make complex gates, but it sufficient for our exercise here...

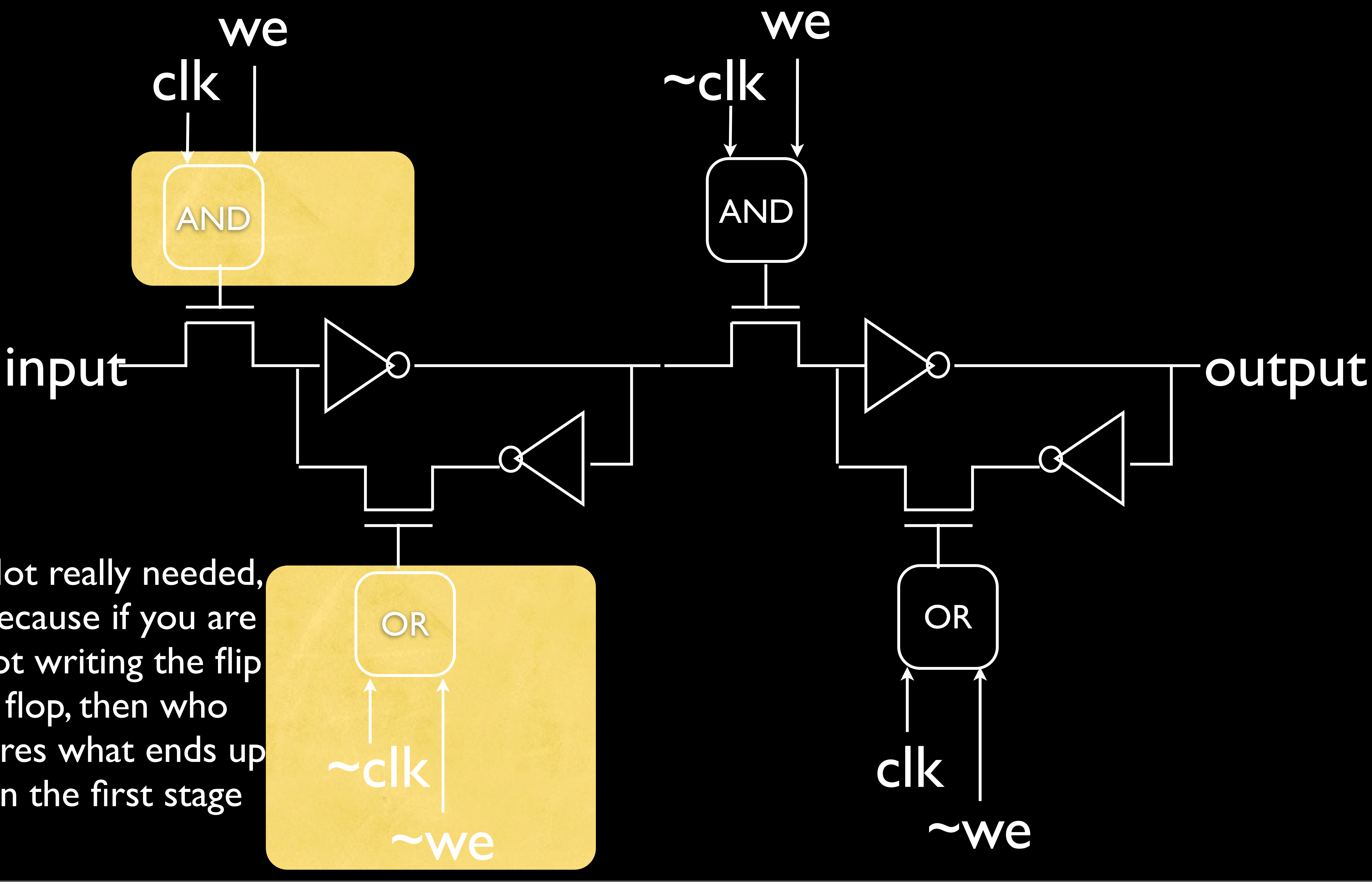
A bit



A bit

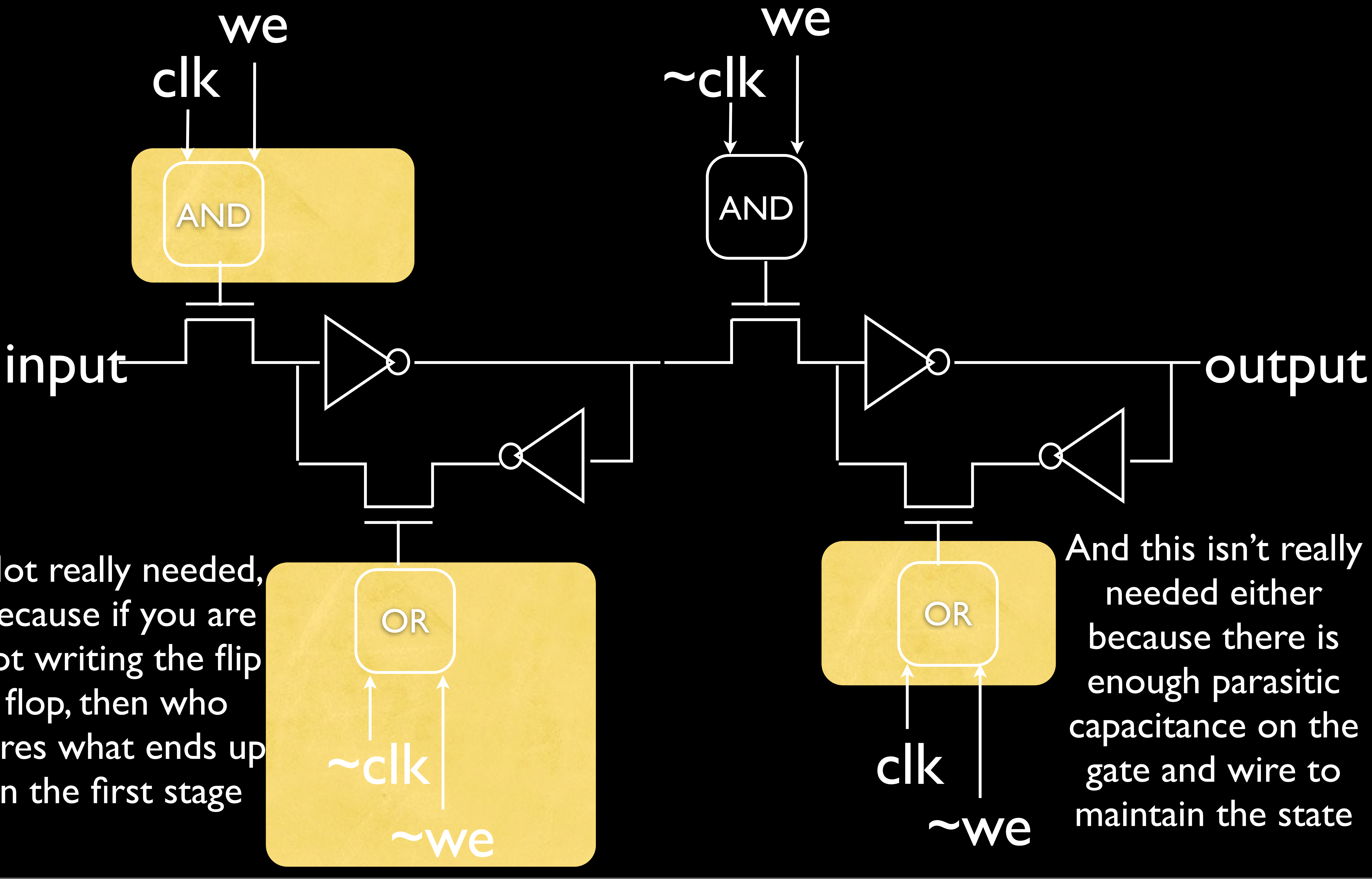


A bit



Not really needed, because if you are not writing the flip flop, then who cares what ends up in the first stage

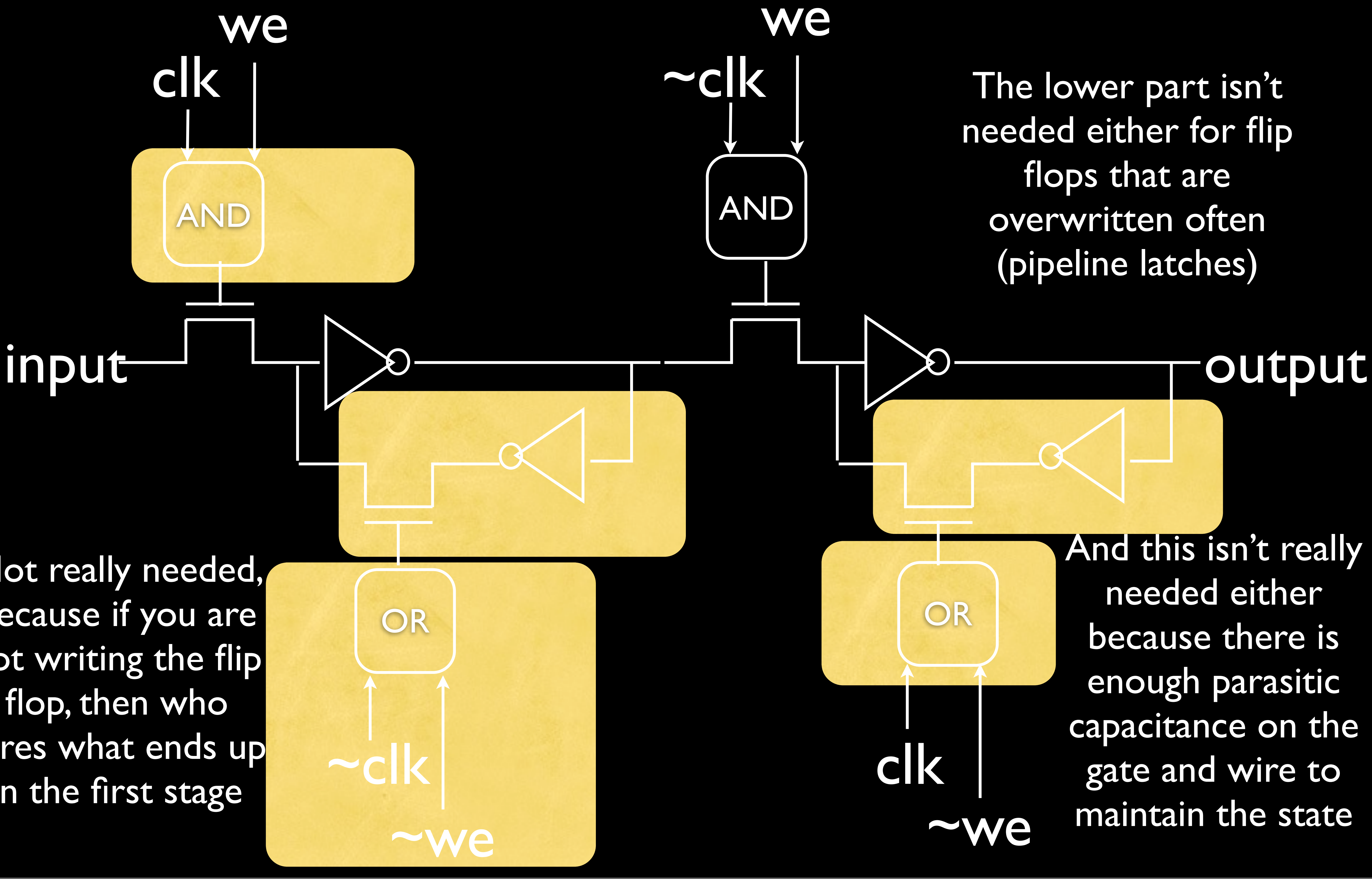
A bit



Not really needed, because if you are not writing the flip flop, then who cares what ends up in the first stage

And this isn't really needed either because there is enough parasitic capacitance on the gate and wire to maintain the state

A bit

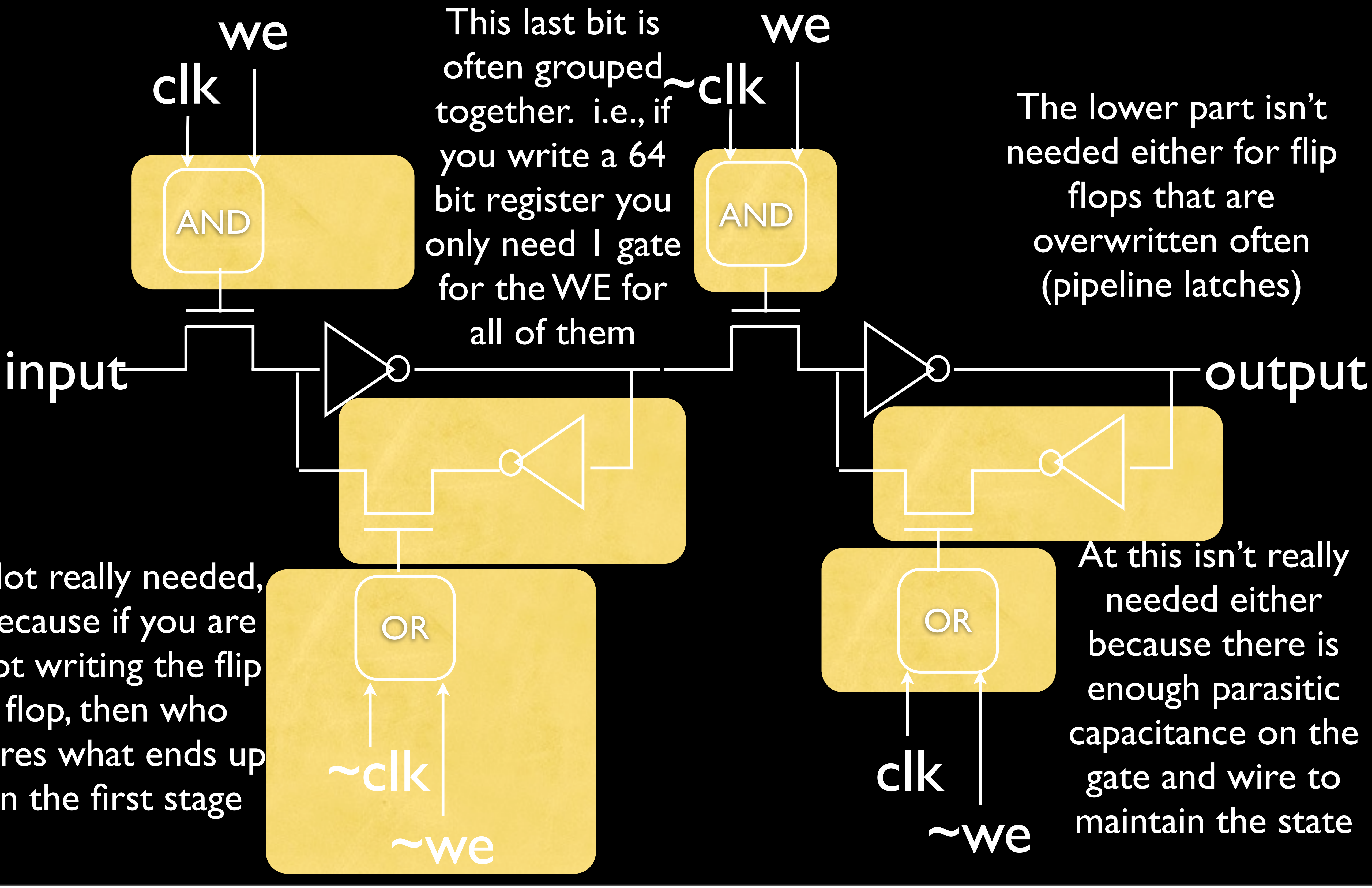


The lower part isn't needed either for flip flops that are overwritten often (pipeline latches)

Not really needed, because if you are not writing the flip flop, then who cares what ends up in the first stage

And this isn't really needed either because there is enough parasitic capacitance on the gate and wire to maintain the state

A bit



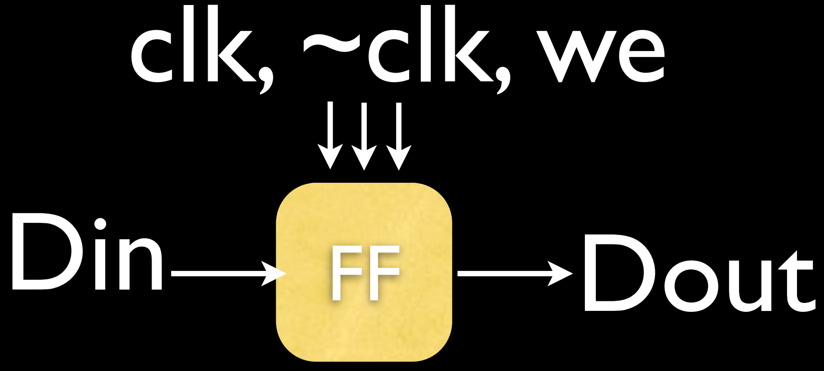
This last bit is often grouped together. i.e., if you write a 64 bit register you only need 1 gate for the WE for all of them

The lower part isn't needed either for flip flops that are overwritten often (pipeline latches)

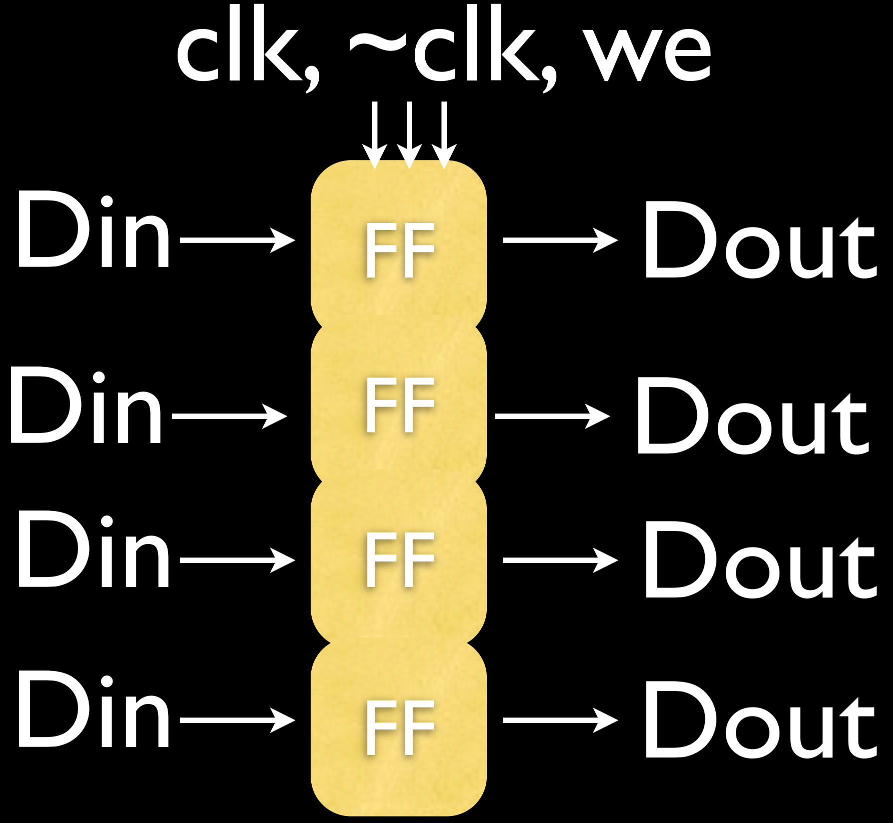
Not really needed, because if you are not writing the flip flop, then who cares what ends up in the first stage

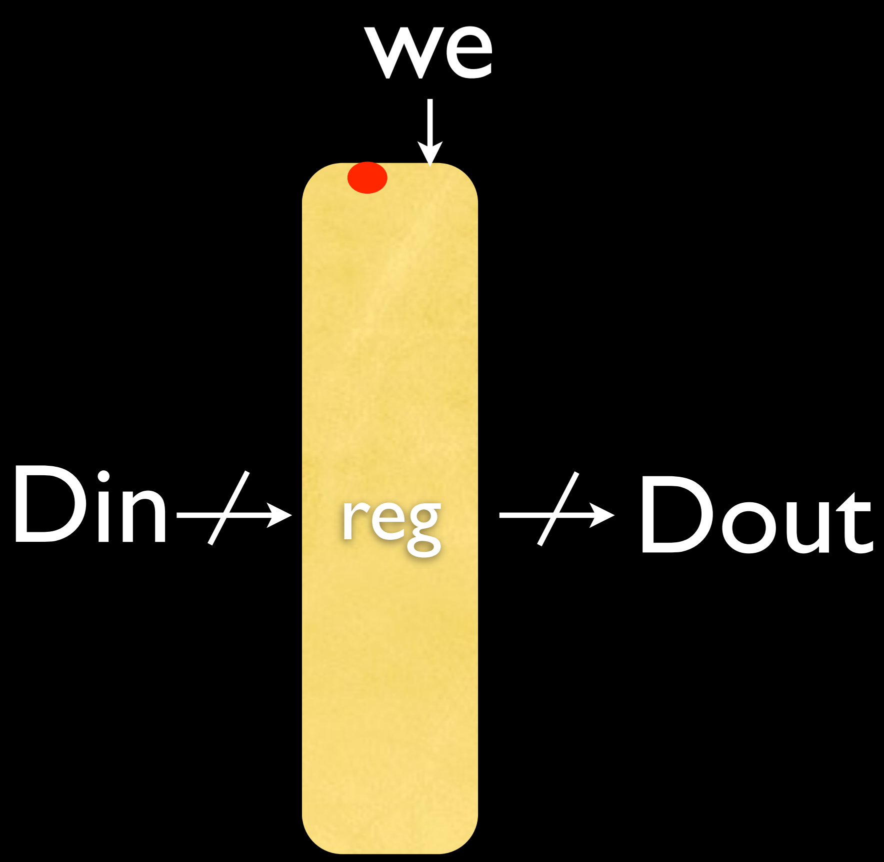
At this isn't really needed either because there is enough parasitic capacitance on the gate and wire to maintain the state

A register

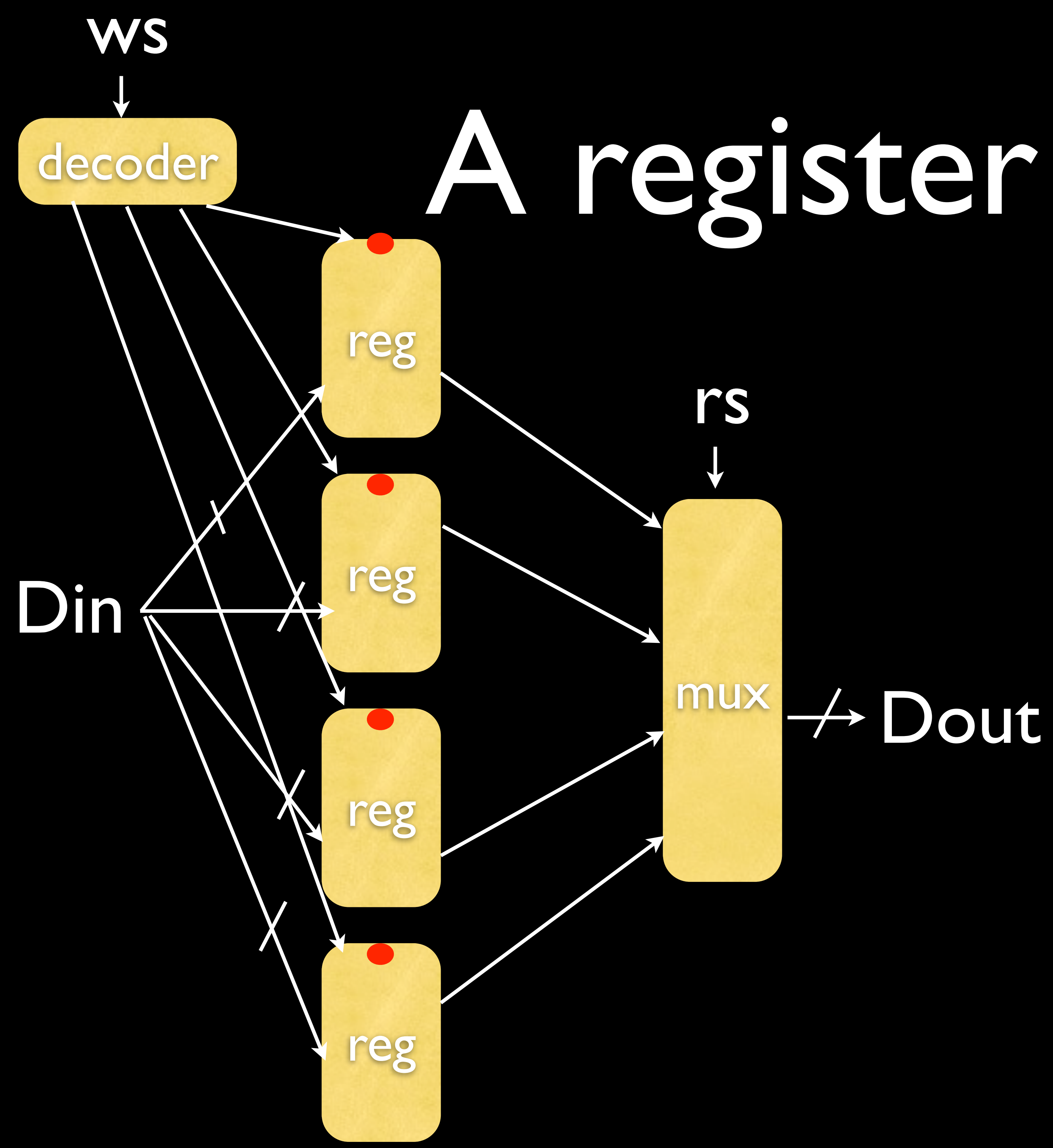


Let’s take our flip flop from before and use that as a basic component

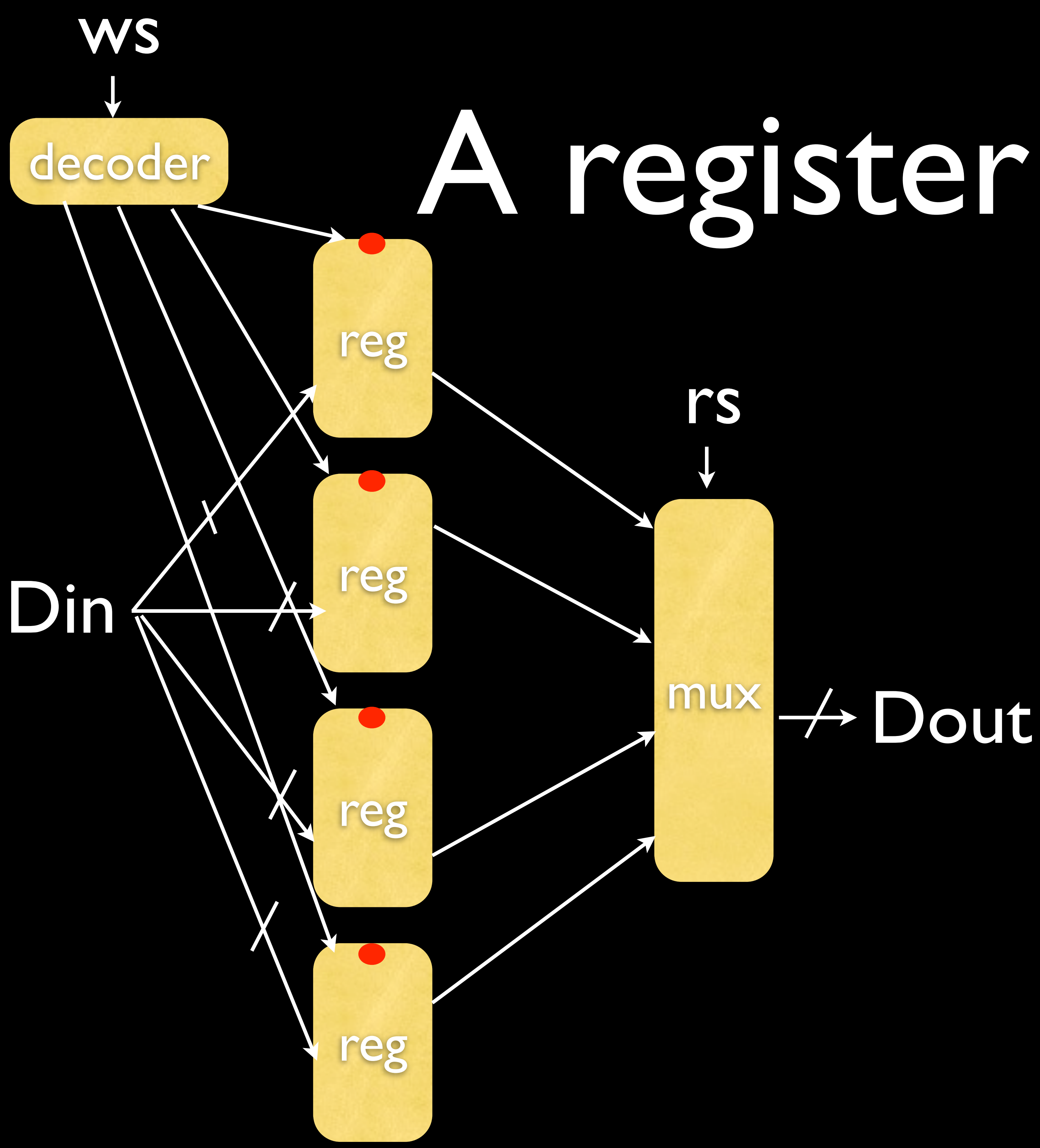




A register file

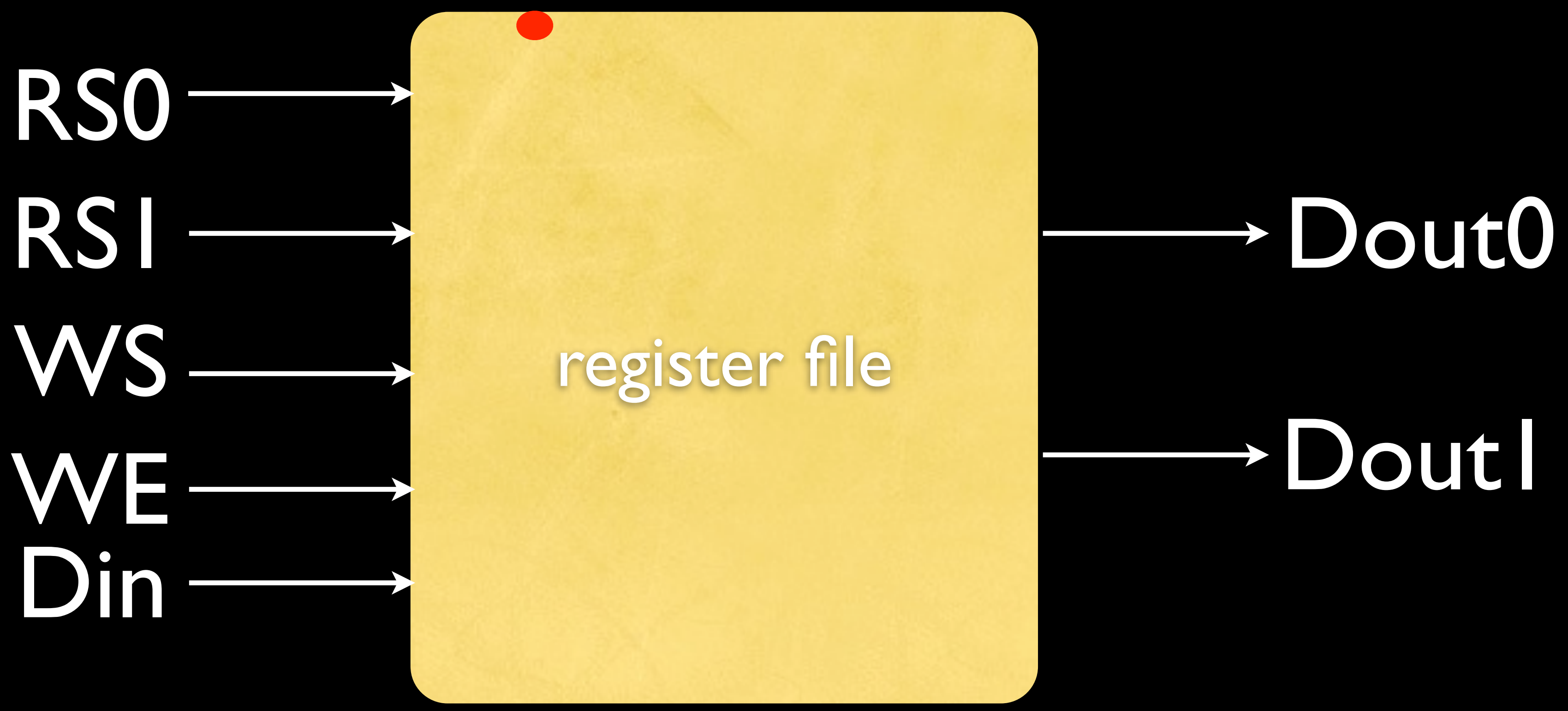


A register file

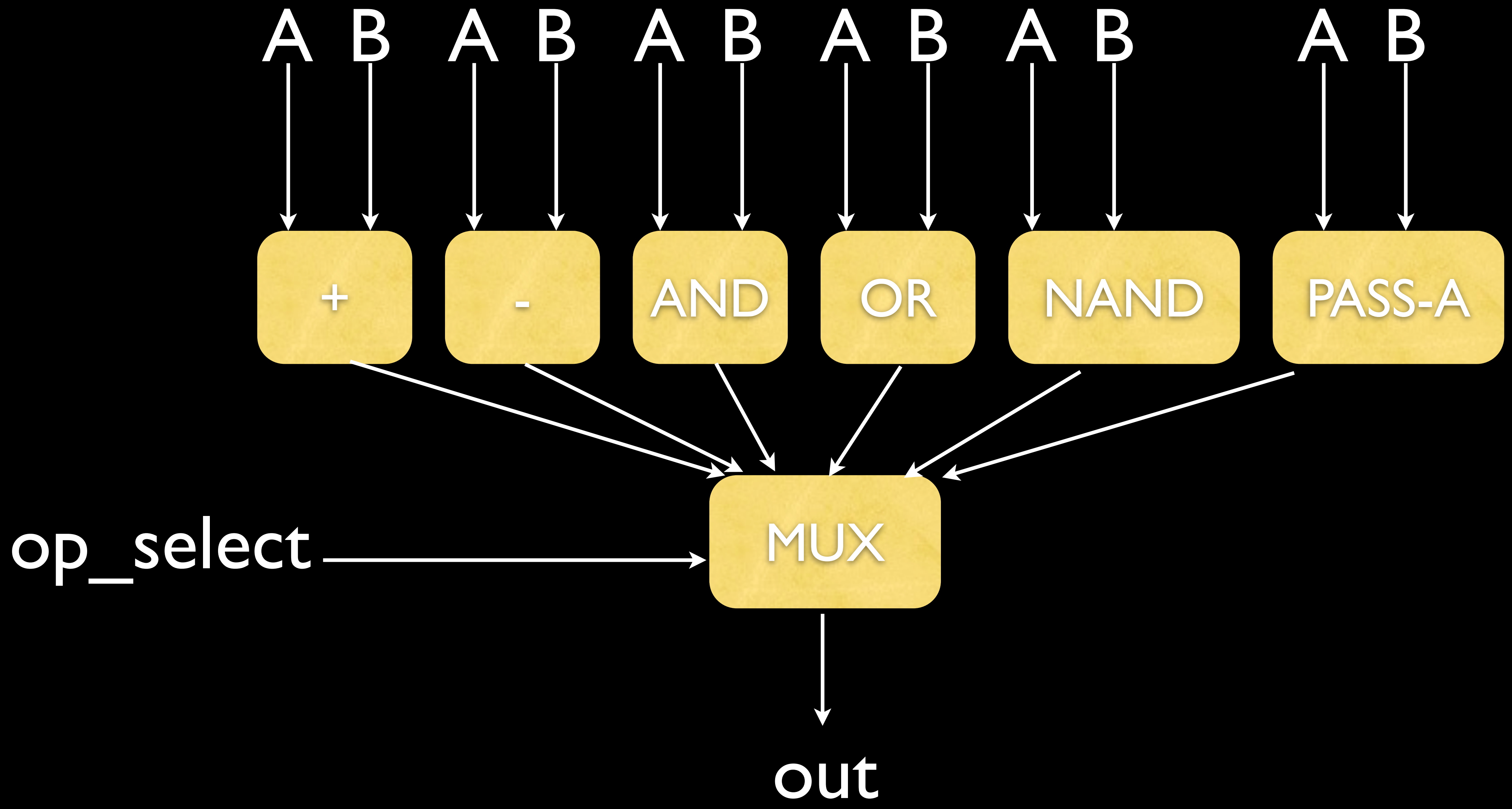


Real register files use a pass-gate mux -- or for very large register files they are actually SRAM structures. But the model here isn't that far off ...

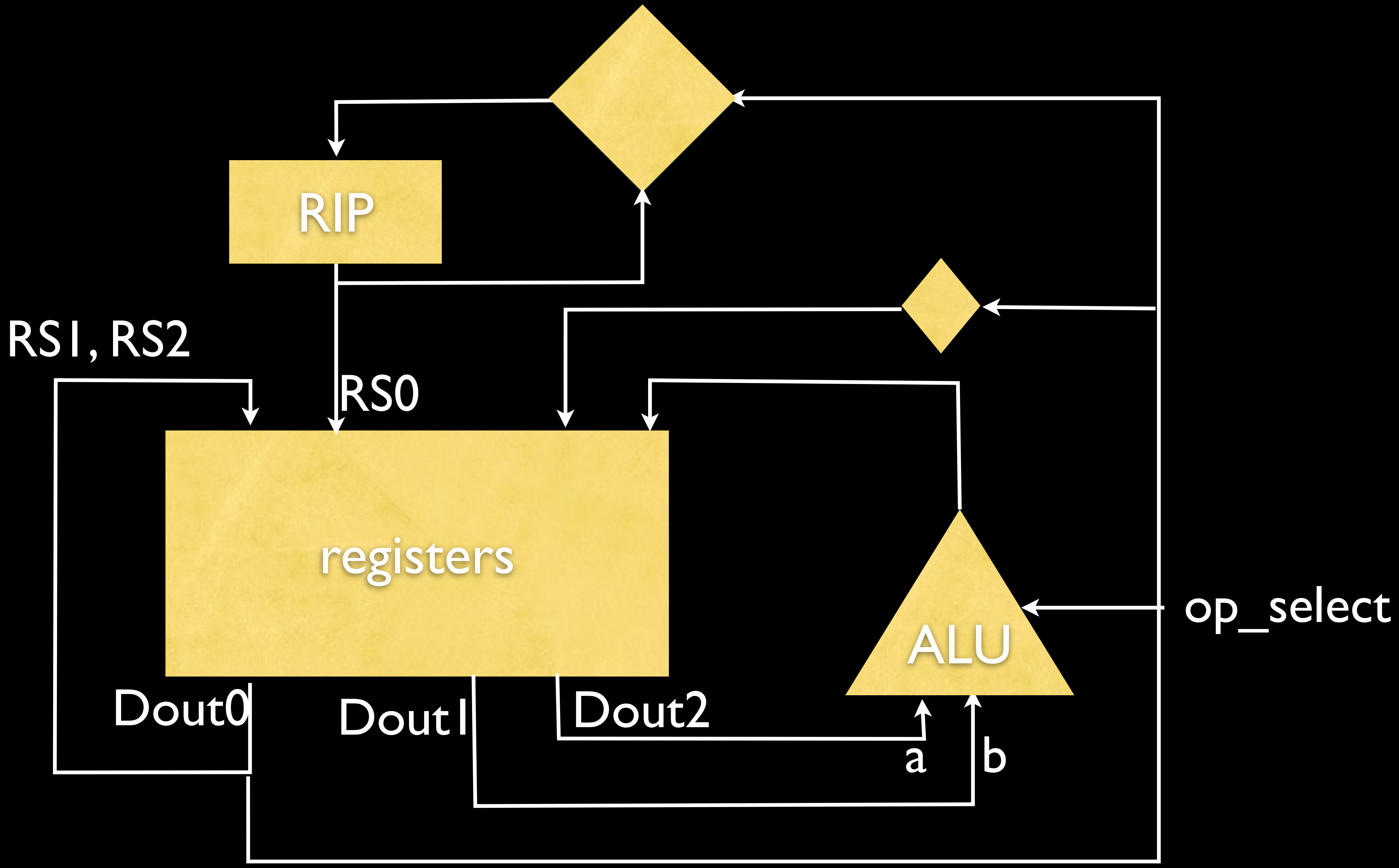
A register file



ALU



The 351 Workhorse :-)


RIP
RS1, RS2
RS0
registers
ALU
op_select
Dout0
Dout1
Dout2
a
b