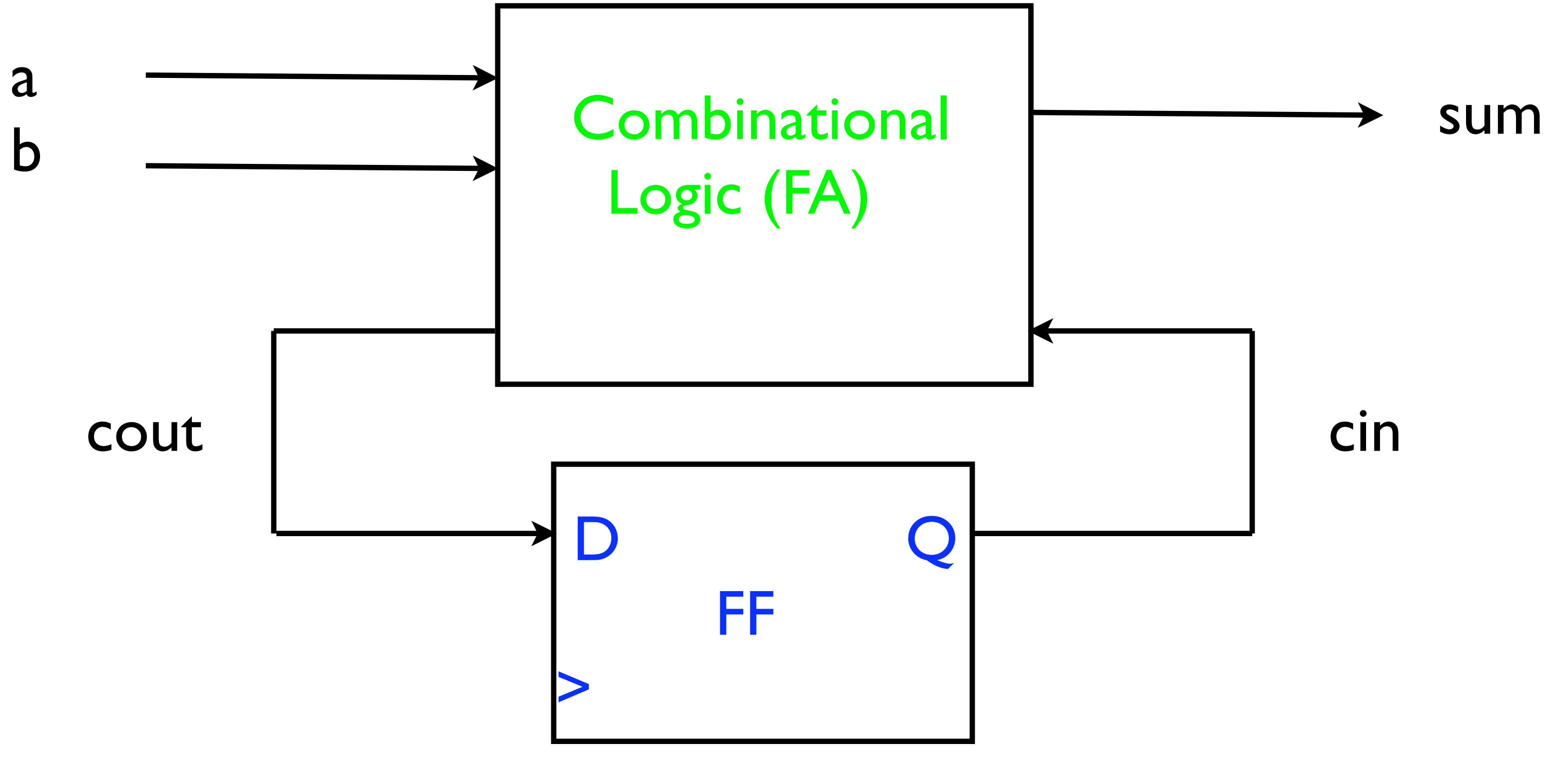
Serial Adder

1-bit "state" is carry-in bit

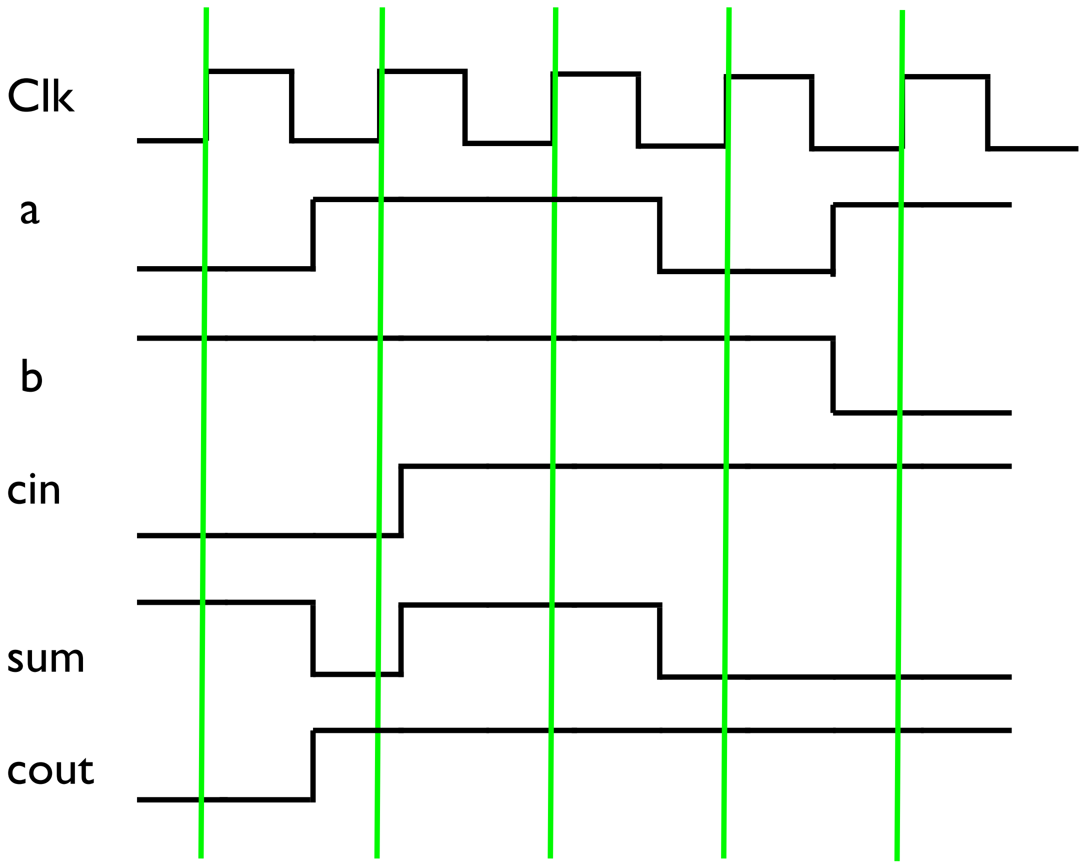
Mealy machine: output (sum) depends on state *and* inputs



Example of Serial Adder

a = 10110 sum = (1)00101

b = 01111



Clock Rising Edge:

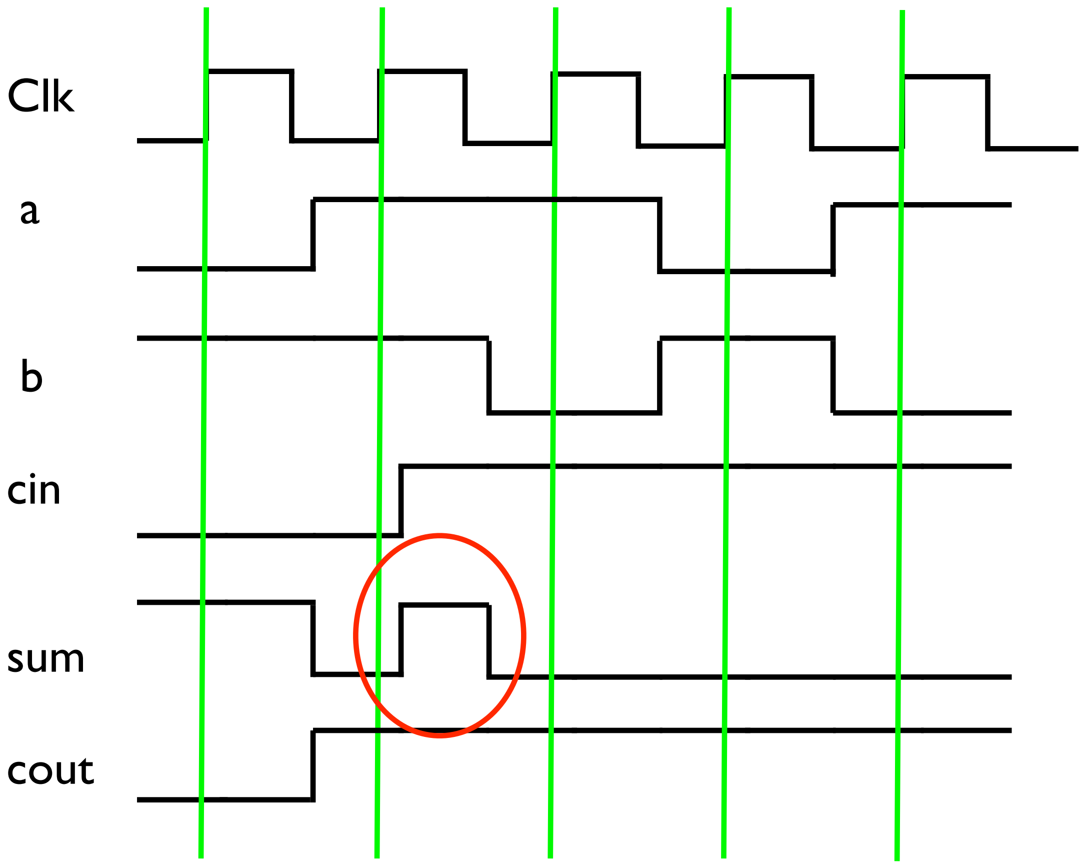
- cin ← cout
- recompute sum, cout

Before Next Clock:

- inputs a,b may change
- recompute sum, cout

Another Example

a = 10110 sum = (1)00001
b = 01011



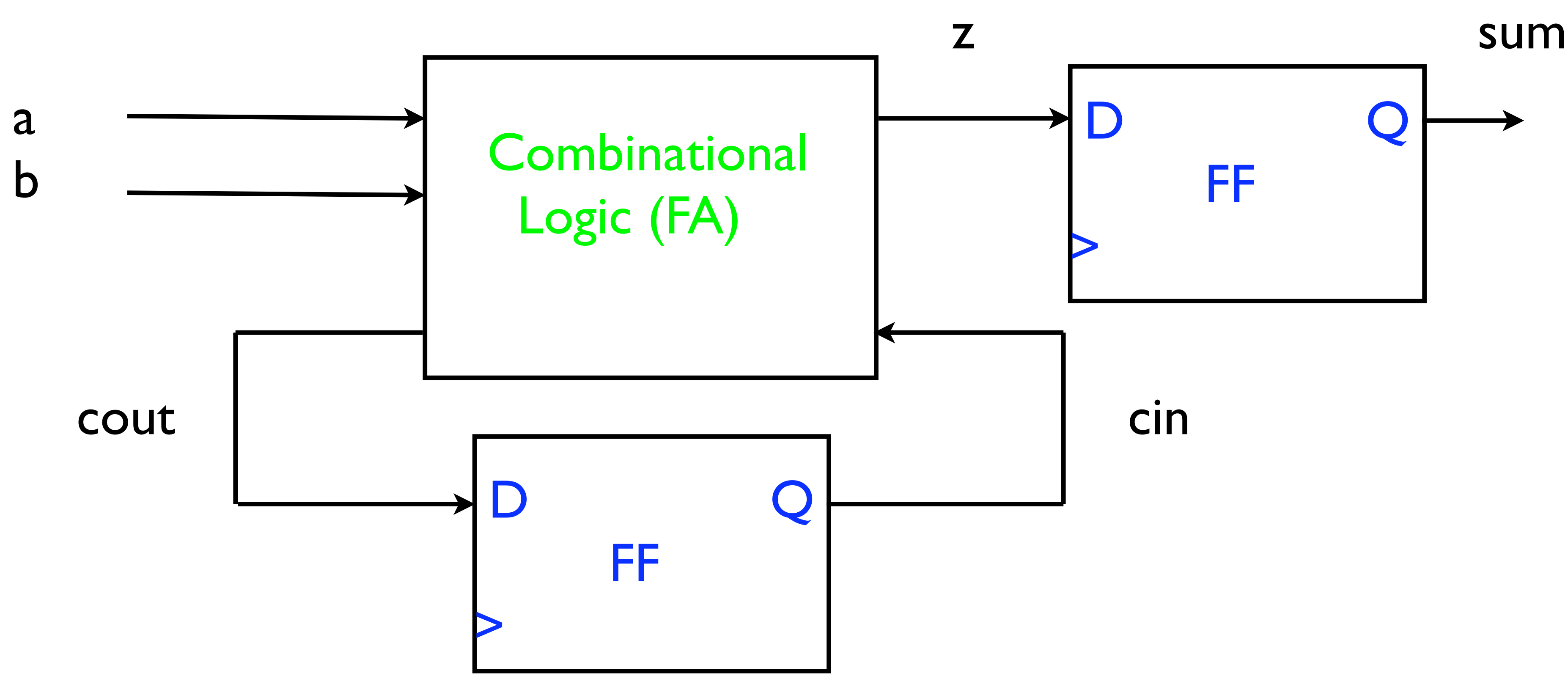
Clock Rising Edge:
cin ← cout
recompute sum, cout

Before Next Clock:
inputs a,b may change
recompute sum, cout

Sum is correct *only*
just before clock

Serial Adder - Moore

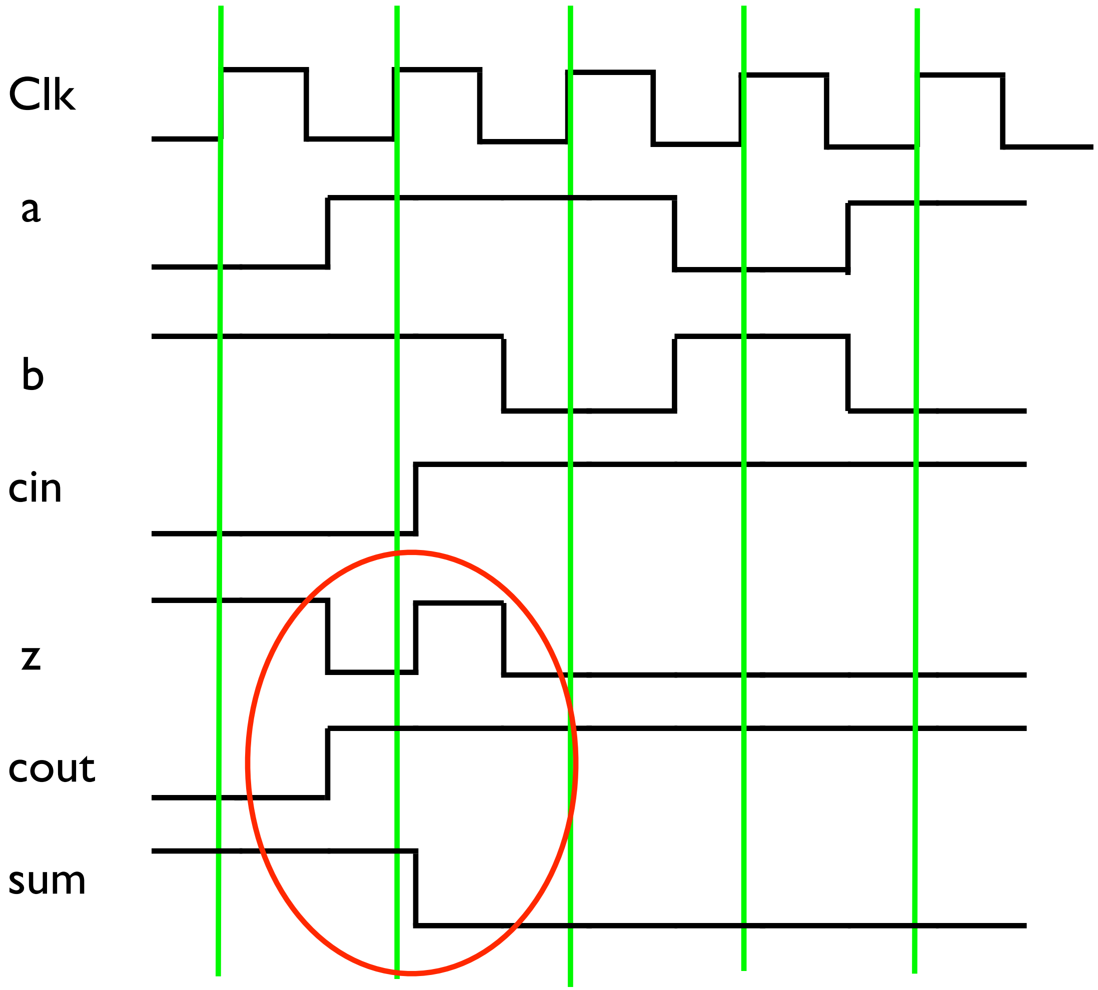
2-bit "state" is carry-in bit and (previous) sum
Moore machine: output (sum) depends on state *only*



The Same Example - Moore

a = 10110 sum = (1)00001

b = 01011



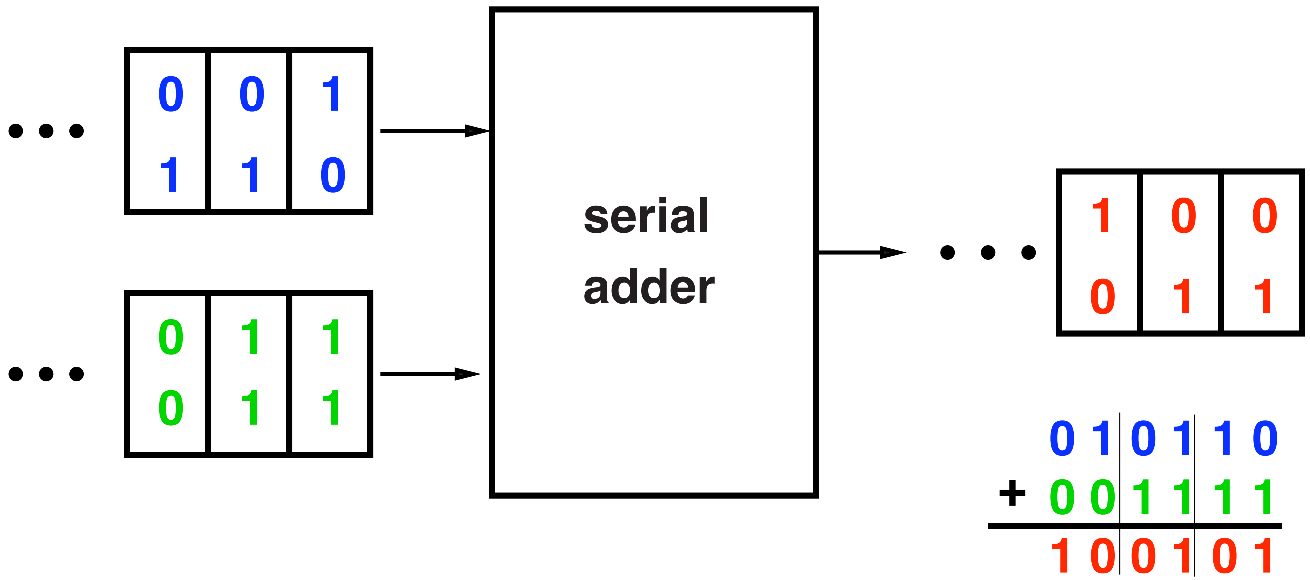
Clock Rising Edge:
- cin ← cout
- sum ← z
- recompute z, cout

Before Next Clock:
- inputs a,b may change
- recompute z, cout

Sum is available at next clock

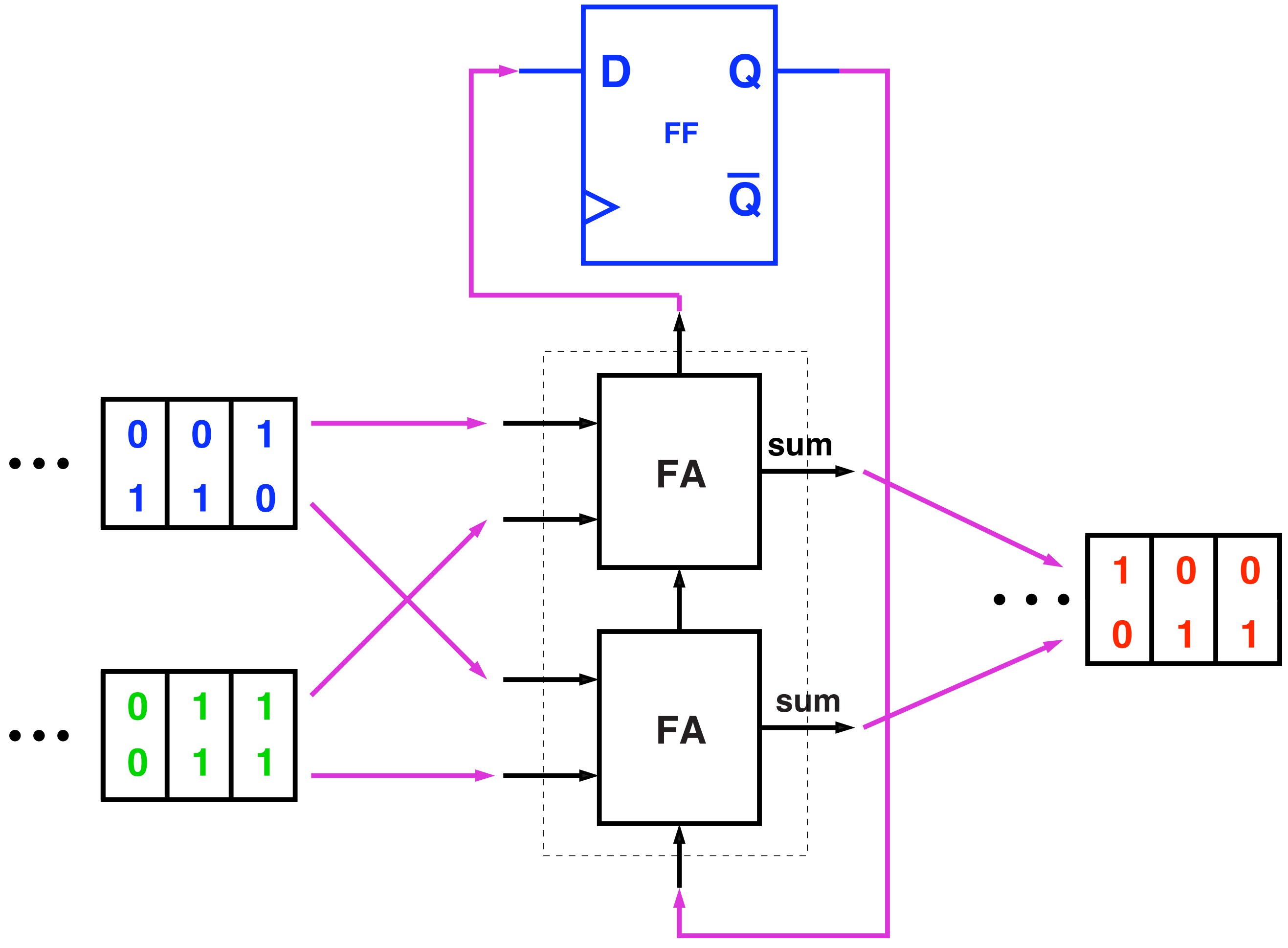
More Bits At A Time

Let's add two bits at a time...



Is this faster?

Two-Bit Adder



Performance

First bit-serial adder:

- takes $2N$ clock cycles to add $2N$ bits
- smaller cycle time

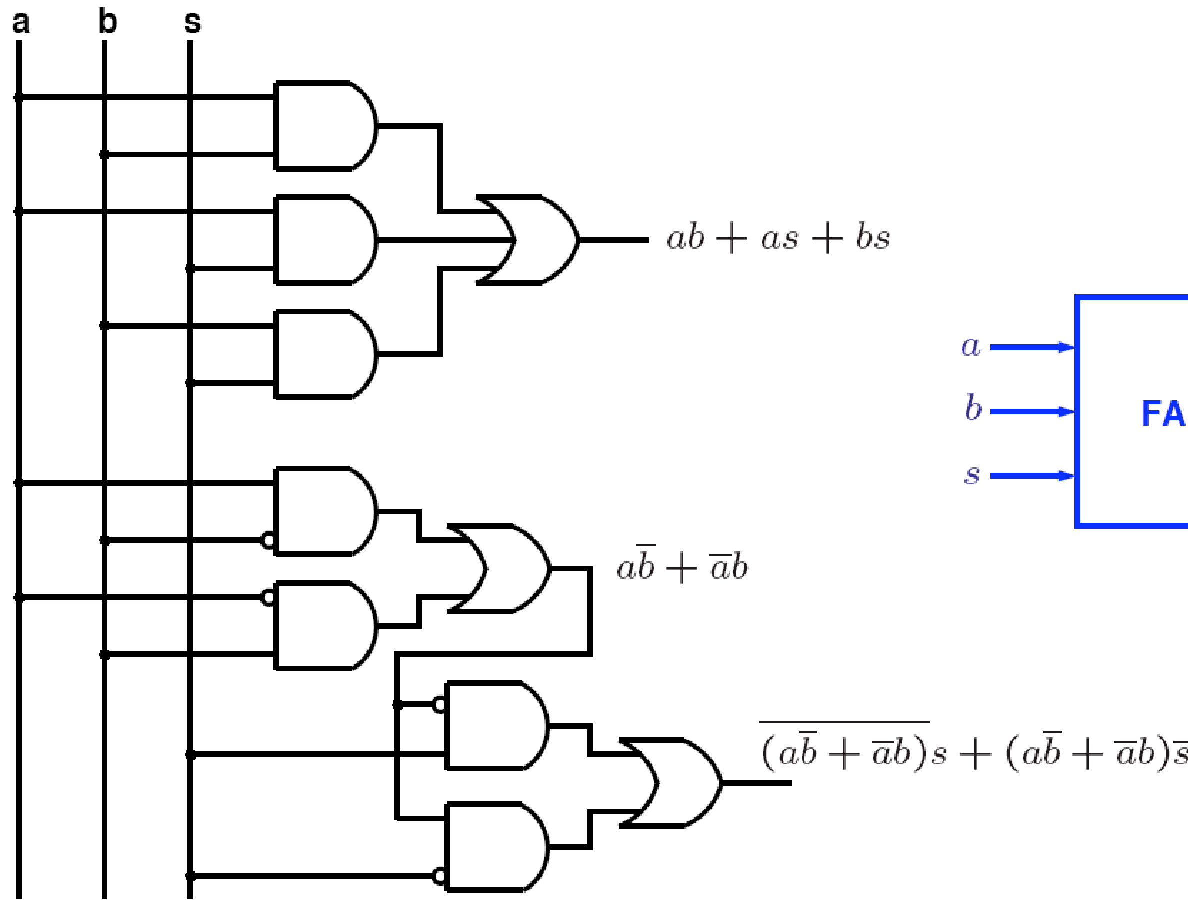
Adding two bits at a time:

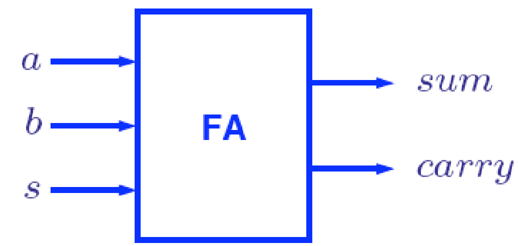
- takes N clock cycles to add $2N$ bits
- larger cycle time

Total time = (number of cycles) $\times$ (cycle period)

Building Blocks For Arithmetic

Binary Addition: recall the full-adder design.





Integer Addition

Full-adder:

- Three input bits a, b, s
- Output: two bits sum and $carry$

Logic equations and gate diagram derived from truth-tables.

What about 4-bit addition?

Integer Addition

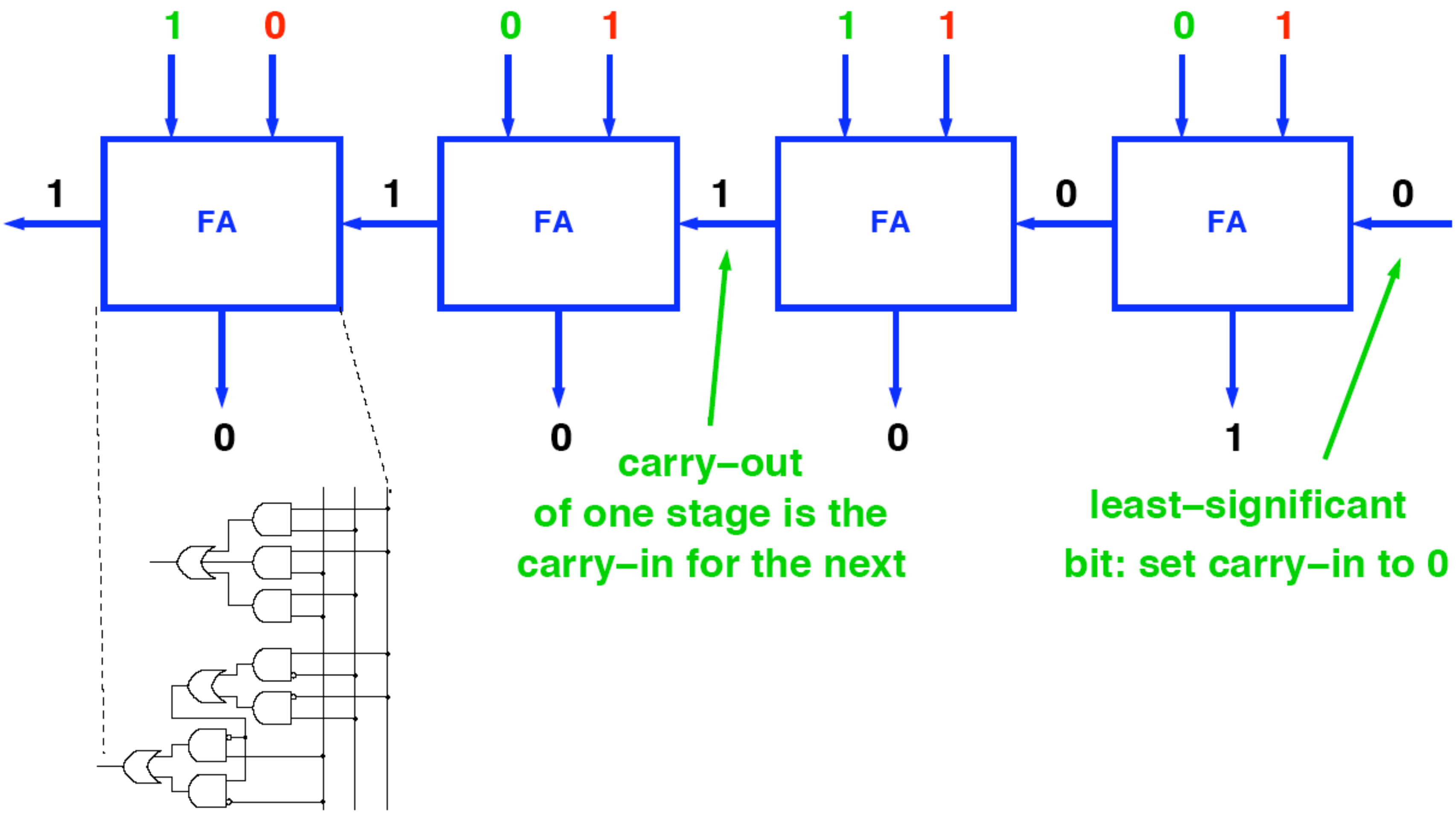
Solution 1: write truth-table, derive logic equations, draw gate diagram.

Solution 2:

	1	1	0	0
	1	0	1	0
+	0	1	1	1
1	0	0	0	1

Use a number of full-adders!

Integer Addition



2's complement? Addition time for N bits?

Integer Addition

Observation: all we need is the carry-out...

$\Rightarrow$ **compute carry-out** *cout* **for** *blocks*

- **input:** 0 0, $cout = 0$ *kill*
- **input:** 1 1, $cout = 1$ *generate*
- **input:** 0 1 **or** 1 0, $cout$**=carry-in** (cin) *propagate*

$$\begin{aligned} cout &= cin \cdot P + G \\ G &= a \cdot b \\ P &= a + b \end{aligned}$$

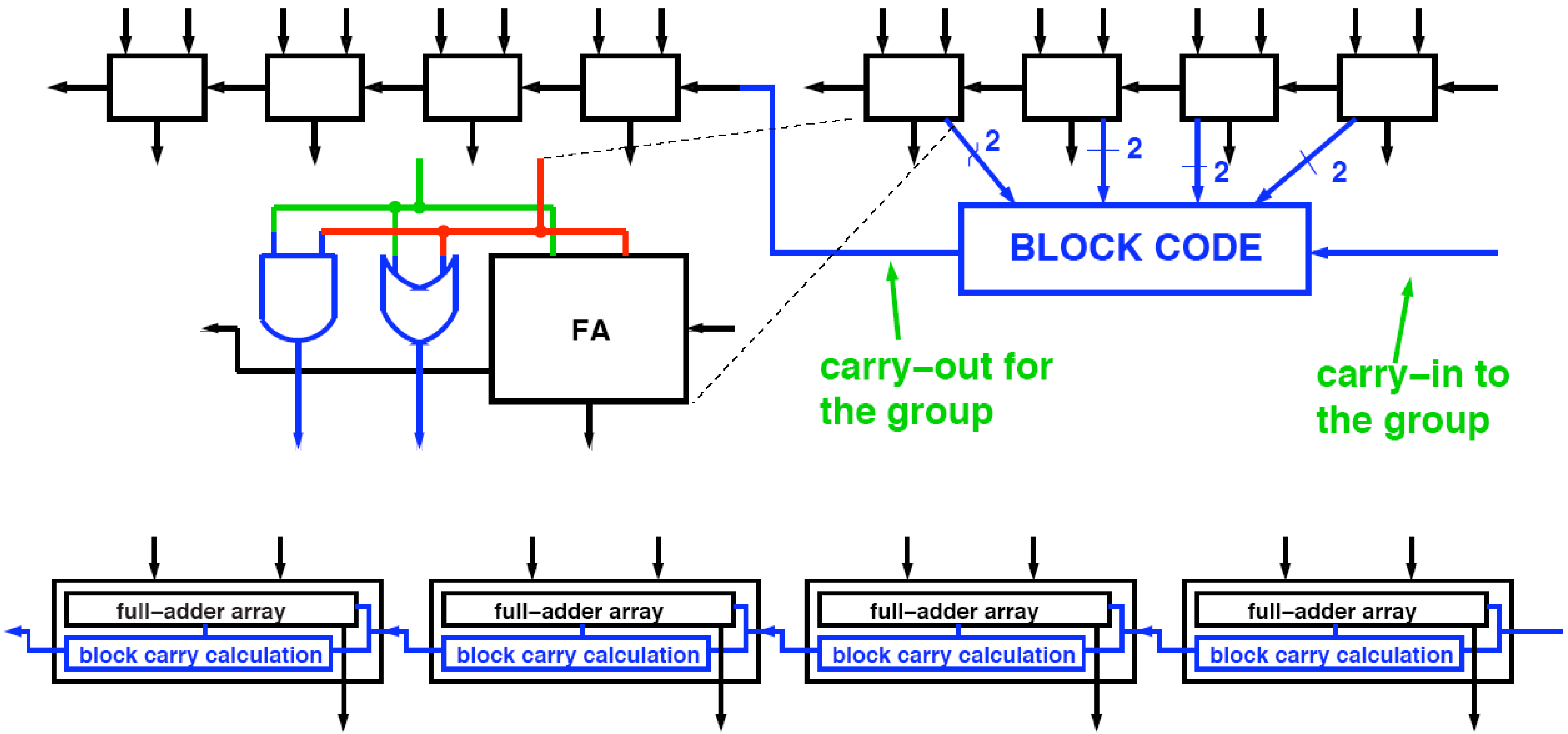
Block codes:

$$G_{01} = G_1 + G_0 P_1$$

$$P_{01} = P_0 P_1$$

Integer Addition

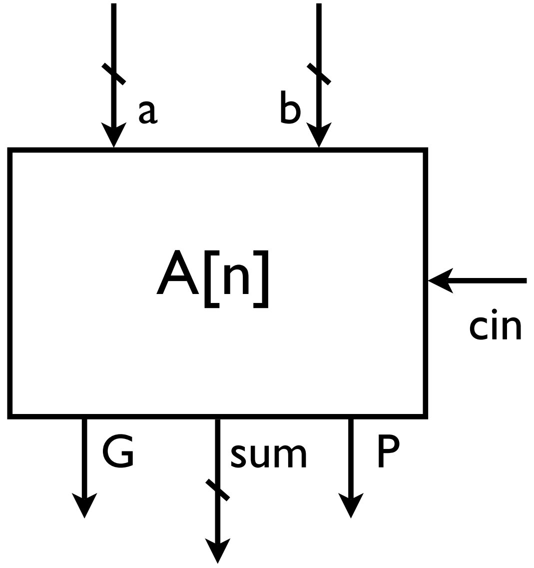
Carry Lookahead adder: compute block codes to speed up carry computation.



Doing Carry-Lookahead Top-Down

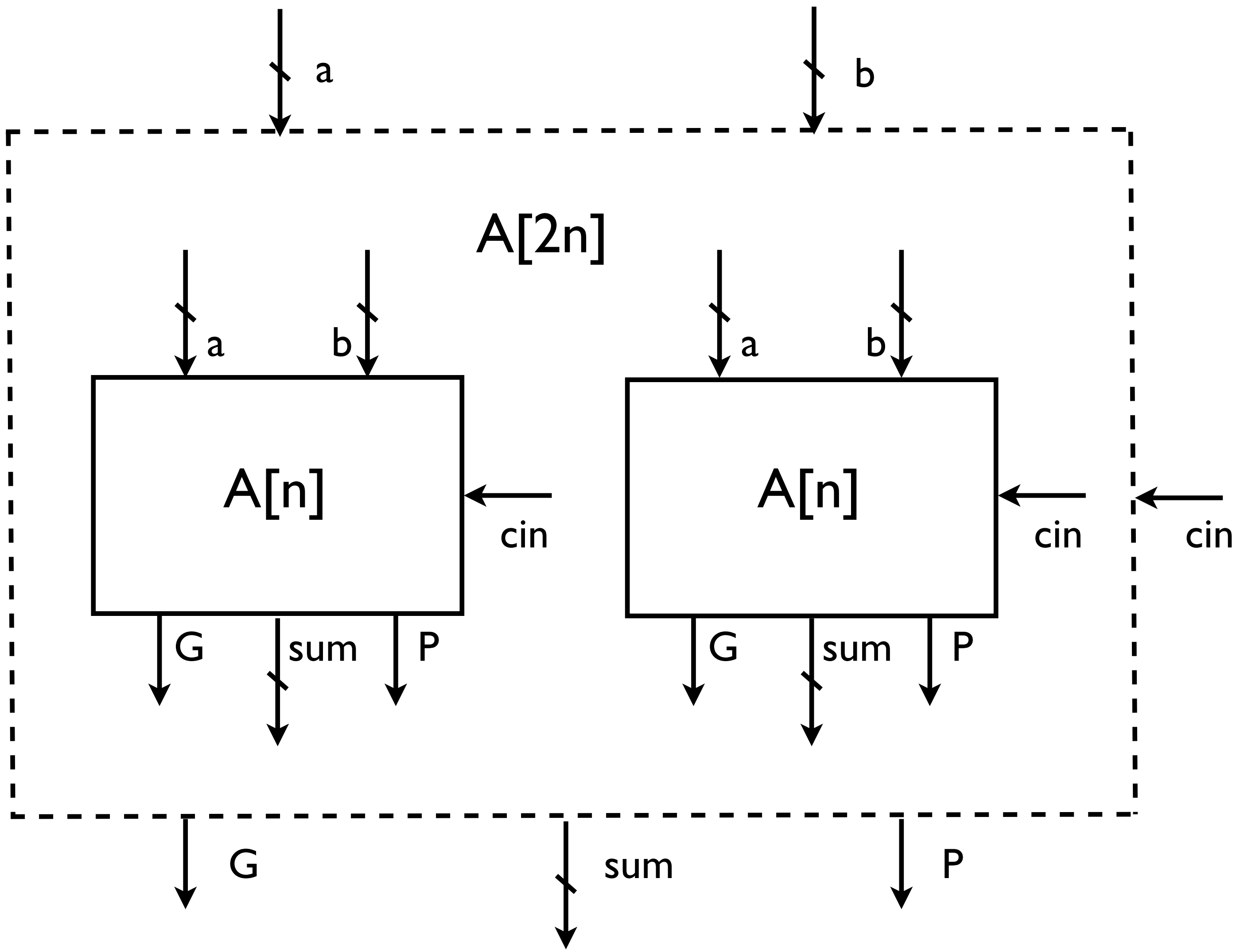
We want to build an n-bit carry-lookahead adder ...

- a, b, cin are the inputs
- G, P, sum are the outputs

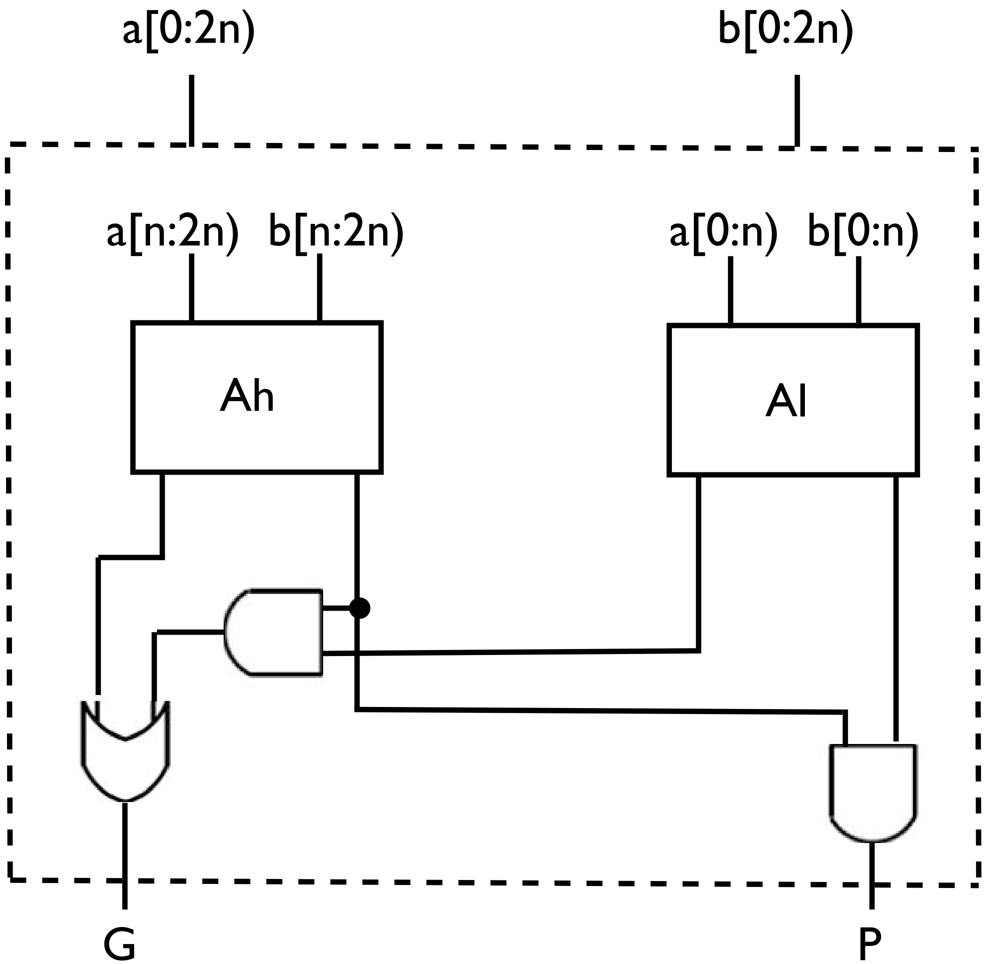


Build a 2n-bit adder from two n-bit ones

Use "divide-and-conquer" approach



Carry Lookahead

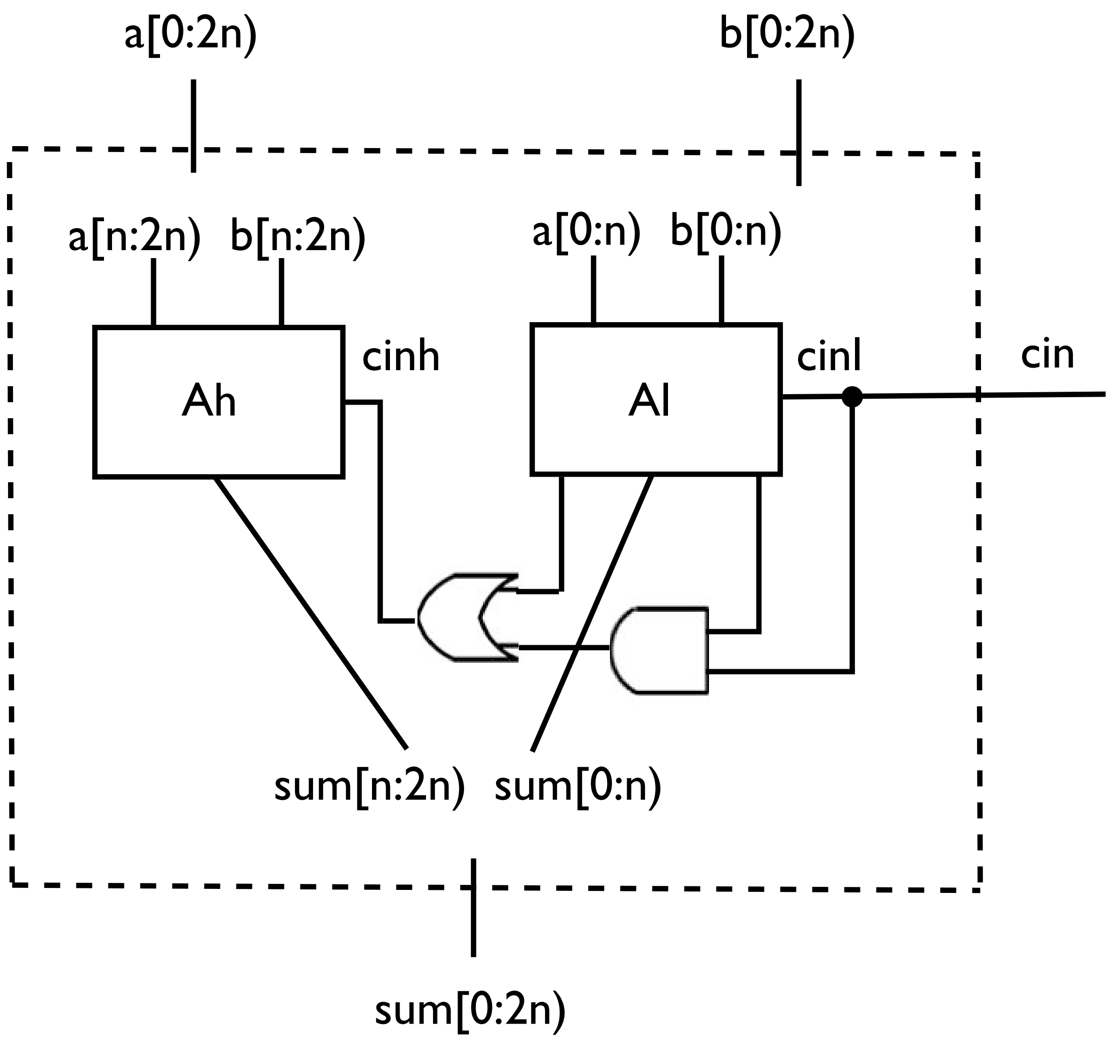


Equations for G,P:

$G = Gh + Ph \cdot Gl$

$P = Ph \cdot Pl$

Carry-in and Sum



Equation for cin:

cinl = cin

cinh = Gl + cinl • Pl

Asymptotic Time

A crude approximation: *phases*

Phase 1: compute all G,P values

$T(1) = \text{constant}$

$T(2n) = T(n) + \text{constant}$

Solution: T(n) is O(log n)

Phase 2: now compute sum ... how much longer?

$S(1) = \text{constant}$

$S(2n) = \text{constant} + S(n)$

Solution: S(n) also is O(log n)

CSL