Introduction to CMOS VLSI Design

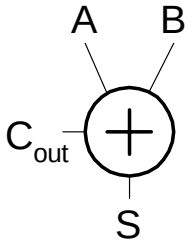
Lecture 11: Adders

Outline

- Single-bit Addition
- Carry-Ripple Adder
- Carry-Skip Adder
- Carry-Lookahead Adder
- Carry-Select Adder
- Carry-Increment Adder
- Tree Adder

Single-Bit Addition

Half Adder

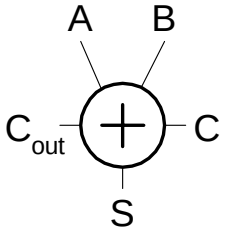


$S =$

$C_{out} =$

A	B	C_{out}	S
0	0		
0	1		
1	0		
1	1		

Full Adder



$S =$

$C_{out} =$

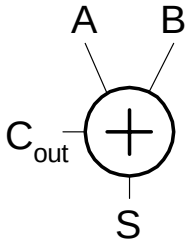
A	B	C	C_{out}	S
0	0	0		
0	0	1		
0	1	0		
0	1	1		
1	0	0		
1	0	1		
1	1	0		
1	1	1		

Single-Bit Addition

Half Adder

$$S = A \oplus B$$

$$C_{out} = A \cdot B$$ (AND)

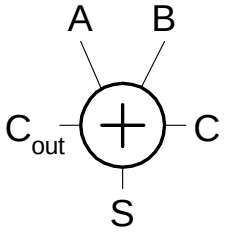


A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Full Adder

$$S = A \oplus B \oplus C$$

$$C_{out} = MAJ(A, B, C)$$ (Majority)



A	B	C	C_{out}	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

PGK

- For a full adder, define what happens to carries
  - Generate: $C_{out} = 1$ independent of C
    - G =
  - Propagate: $C_{out} = C$
    - P =
  - Kill: $C_{out} = 0$ independent of C
    - K =

PGK (carry 관련 terms)

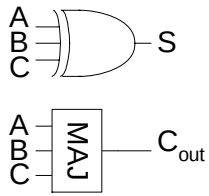
- For a full adder, define what happens to carries
  - Generate: C_{out} = 1 independent of C (C = C in)
    - G = A • B
  - Propagate: C_{out} = C
    - P = A ⊕ B
  - Kill: C_{out} = 0 independent of C
    - K = ~A • ~B

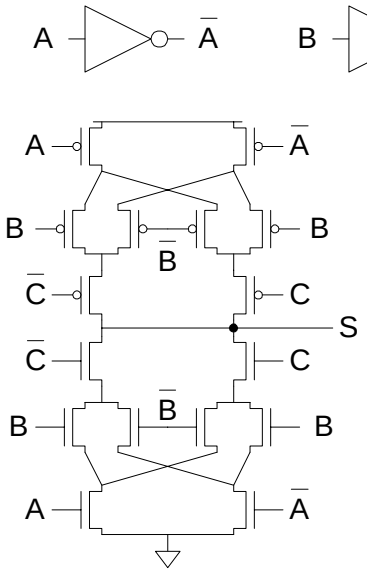
Full Adder Design I

- Brute force implementation from eqns

$$S = A \oplus B \oplus C \quad = \text{P exor C}$$

$$C_{out} = MAJ(A, B, C)$$





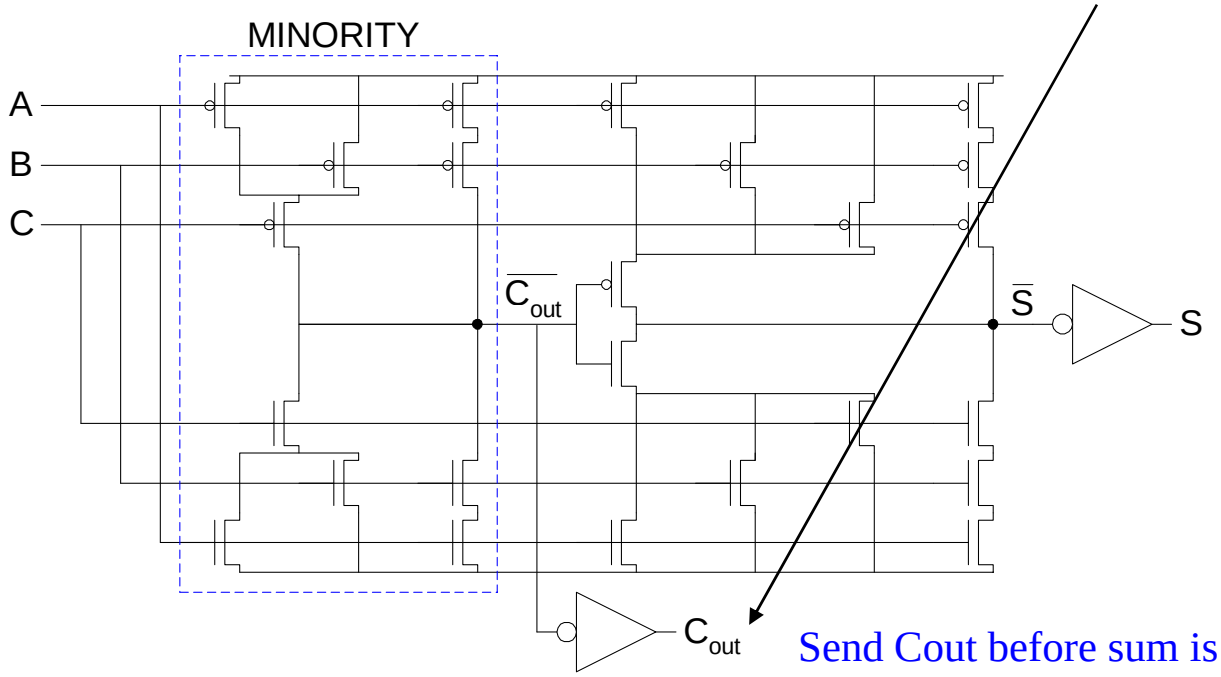
Notice they are symmetrical

Full Adder Design II

- Factor S in terms of C_{out}

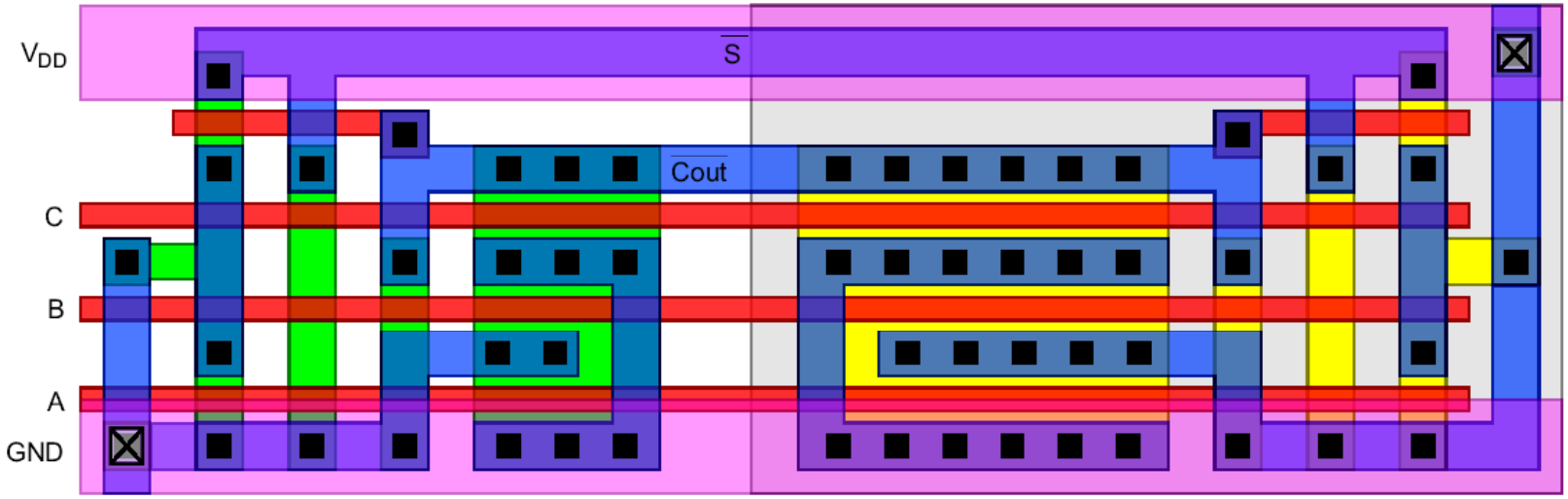
  $S = ABC + (A + B + C)(\sim C_{out})$

- Critical path is usually C to C_{out} in ripple adder



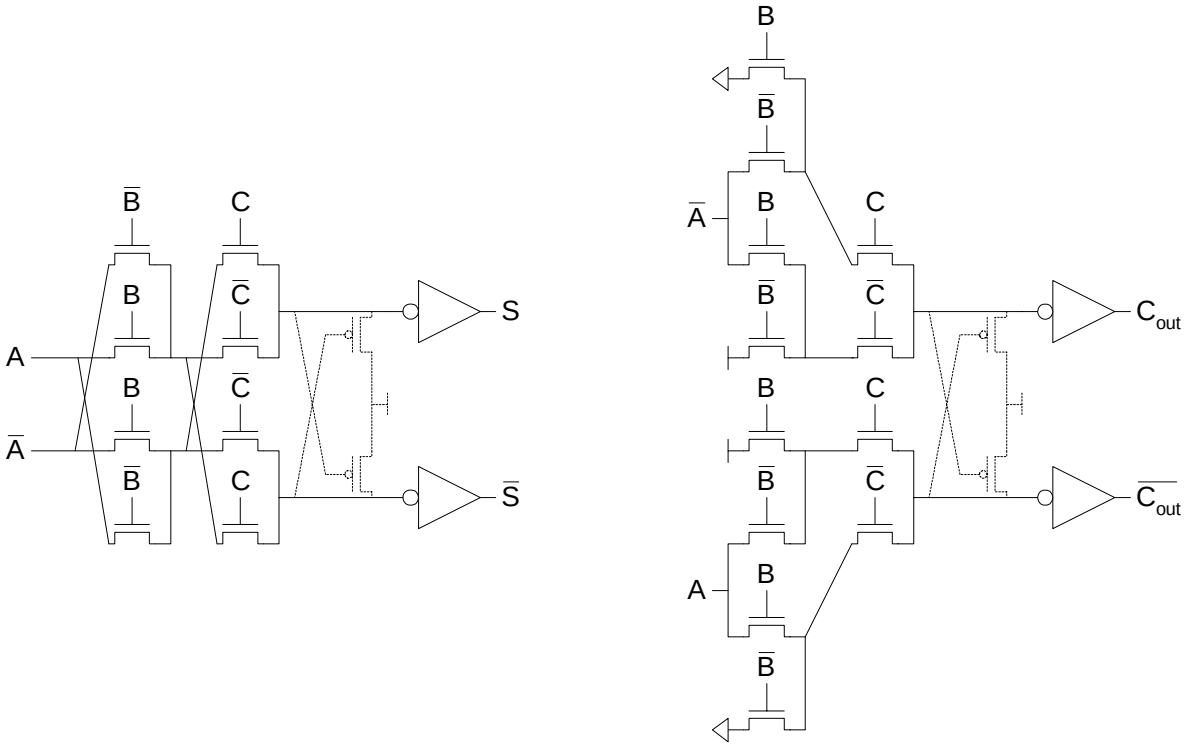
Layout

- Clever layout circumvents usual line of diffusion
  - Use wide transistors on critical path
  - Eliminate output inverters



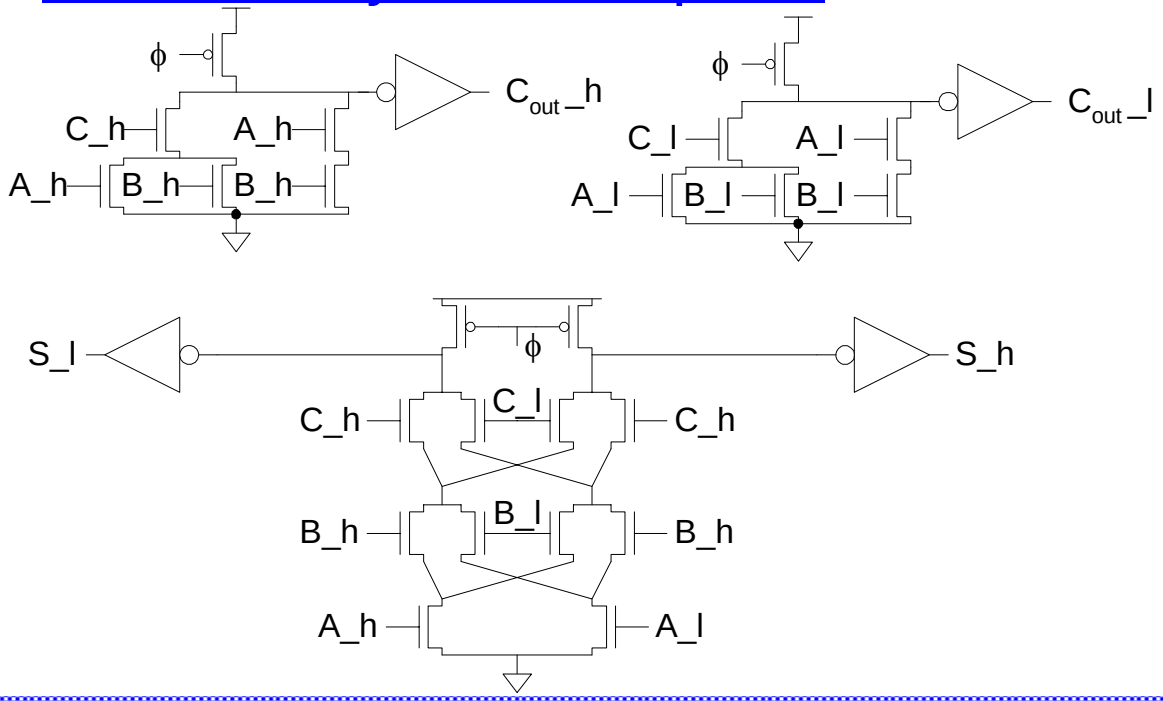
Full Adder Design III

- Complementary Pass Transistor Logic (CPL)
  - Slightly faster, but more area



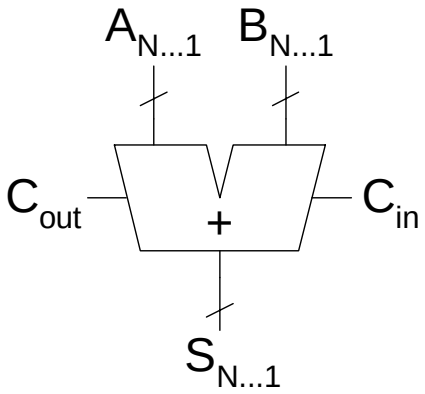
Full Adder Design IV

- Dual-rail domino
  - Very fast, but large and power hungry
  - Used in very fast multipliers



Carry Propagate Adders

- N-bit adder called CPA
  - Each sum bit depends on all previous carries
  - How do we compute all these carries quickly?



Both sum and carry are influenced by Cin
Conventional method

Case I

C_{out} C_{in}

```
 00000
  1111
 +0000
 ------
  1111
```

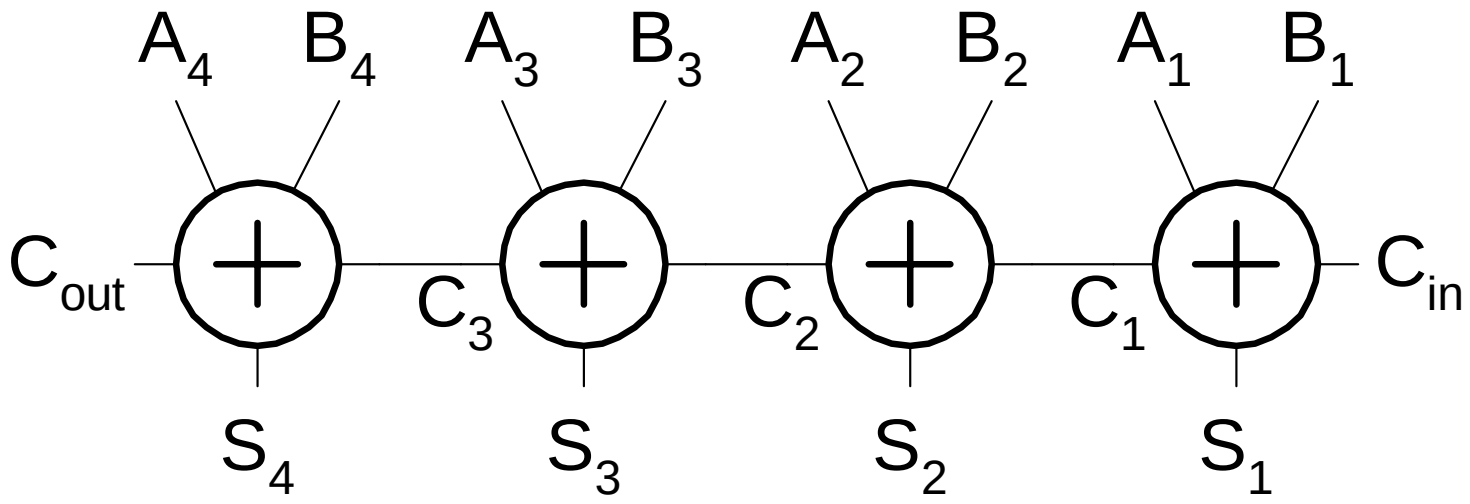
Case II

C_{out} C_{in}

```
 11111   carries
  1111   A4...1
 +0000   B4...1
 ------
  0000   S4...1
```

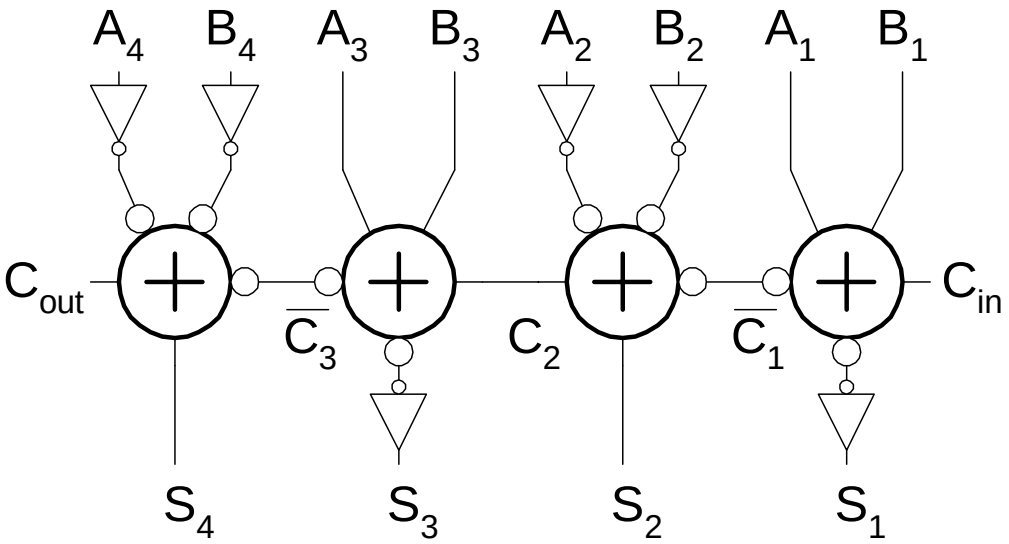
Carry-Ripple Adder

- Simplest design: cascade full adders
  - Critical path goes from Cin to Cout
  - Design full adder to have fast carry delay



Inversions

- Critical path passes through majority gate
  - Built from minority + inverter
  - Eliminate inverter and use inverting full adder



Eliminate inverter between stages

Using inverter, majority calculation becomes minority calculation.

Generate / Propagate

- Equations often factored into G and P
- Generate and propagate for groups spanning i:j

$$G_{i:j} =$$

$$P_{i:j} =$$

- Base case

$$G_{i:i} \equiv G_i =$$

$$P_{i:i} \equiv P_i =$$

$$G_{0:0} \equiv G_0 =$$

$$P_{0:0} \equiv P_0 =$$

- Sum:

$$S_i =$$

Generate / Propagate

- Equations often factored into G and P
- Generate and propagate for groups spanning i:j

$$G_{i:j} = G_{i:k} + P_{i:k} \cdot G_{k-1:j}$$

AND

$$P_{i:j} = P_{i:k} \cdot P_{k-1:j}$$

(Either stages from k to i generates carry
Or stages from k-1 to j generates carry
propagating all the way up to i stage
from k stage.

k can be any stage between i and k

- Base case

$$G_{i:i} \equiv G_i = A_i \cdot B_i$$

$$P_{i:i} \equiv P_i = A_i \oplus B_i$$

$G_{i-1:0}$ means that the carry
can be either generated or propagated
as the def. $G_{i;j}$

$$G_{0:0} \equiv G_0 = C_{in}$$

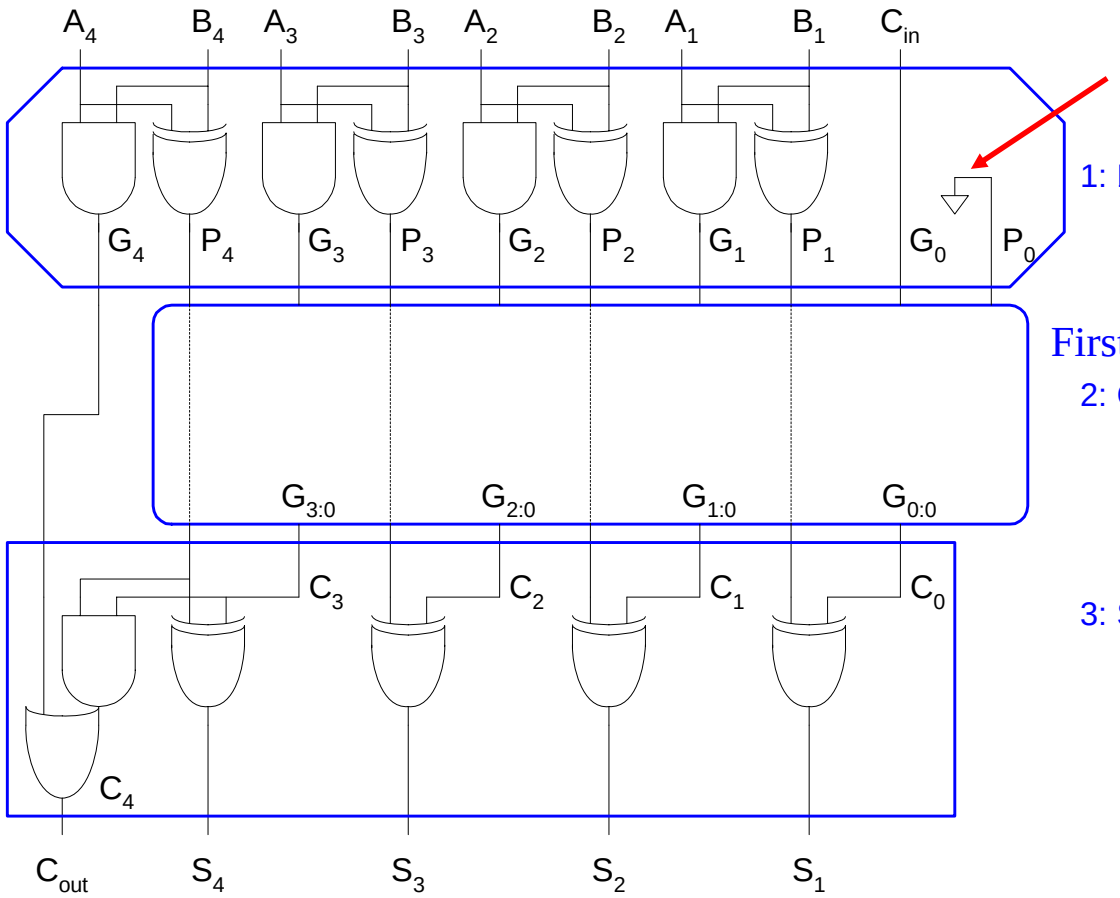
$G_{i-1:0} = C_{i-1}$

$$P_{0:0} \equiv P_0 = 0$$

- Sum:

$$S_i = P_i \oplus G_{i-1:0}$$ (As slide 7)

PG Logic



1: Bitwise PG logic

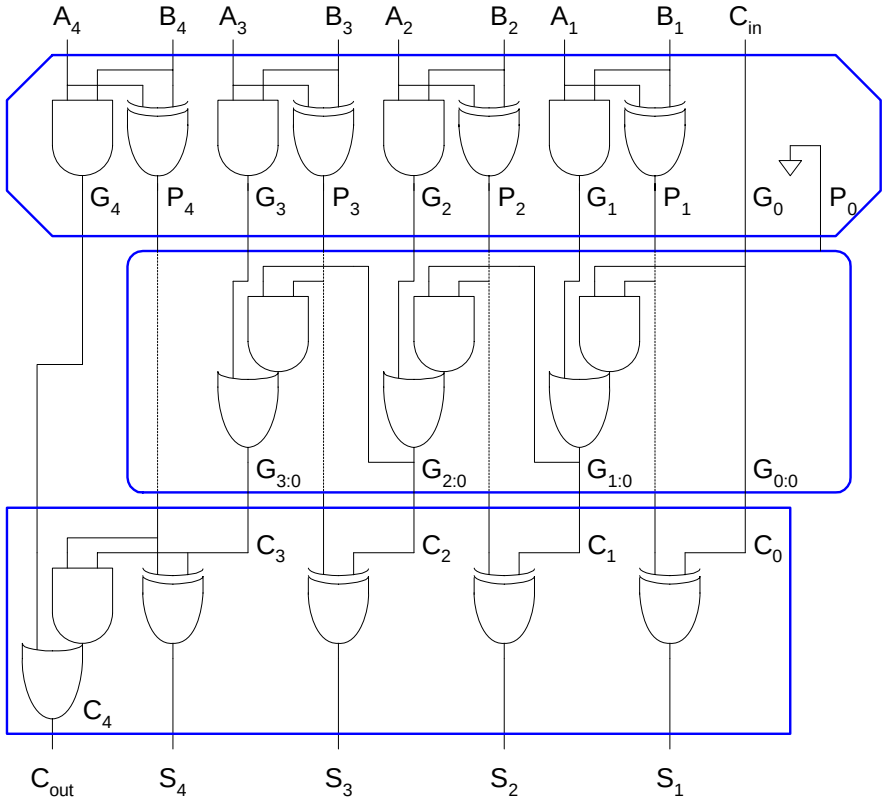
First Eqn. in previous slide

2: Group PG logic

3: Sum logic

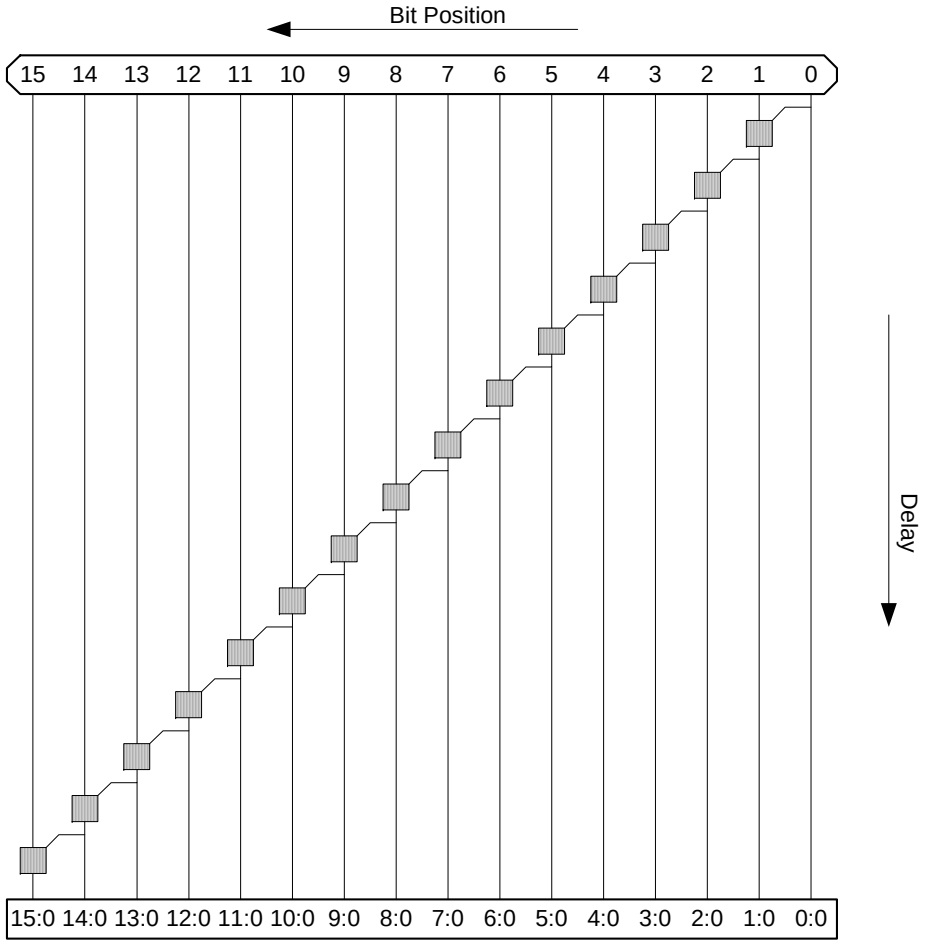
Carry-Ripple Revisited

$$G_{i:0} = G_i + P_i \cdot G_{i-1:0}$$



Carry-Ripple PG Diagram

$t_{ripple} =$

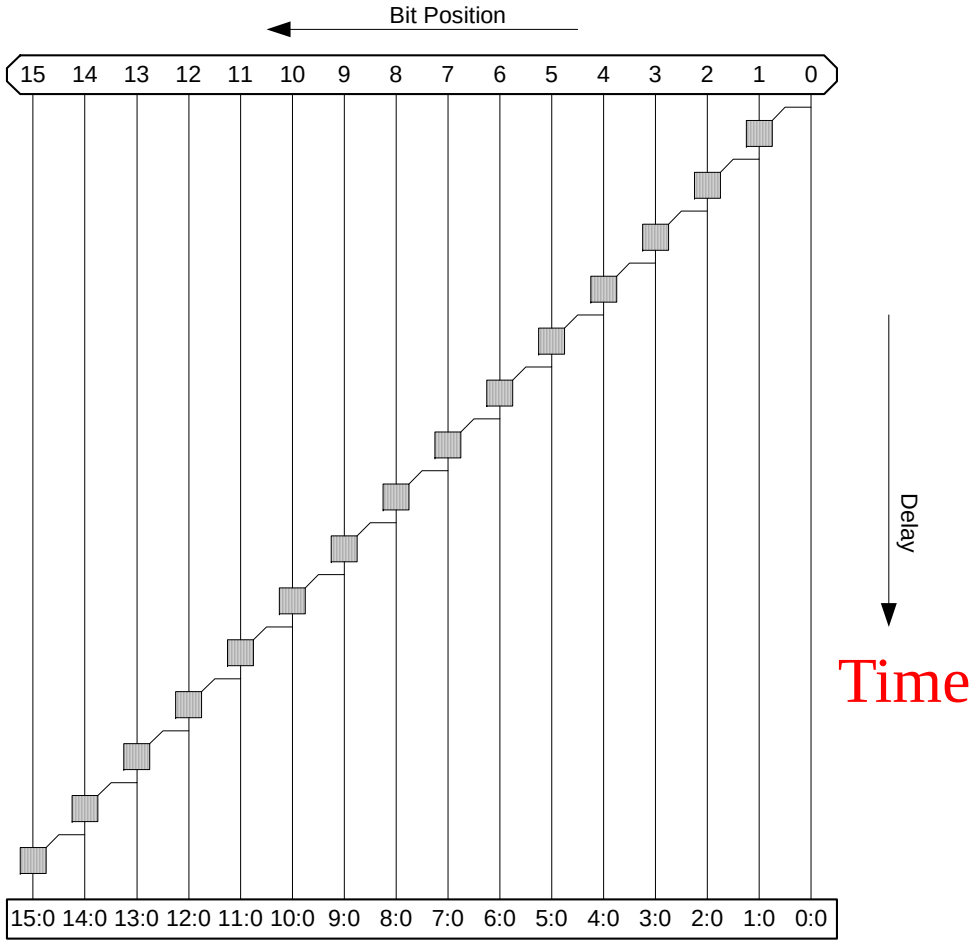


Carry-Ripple PG Diagram

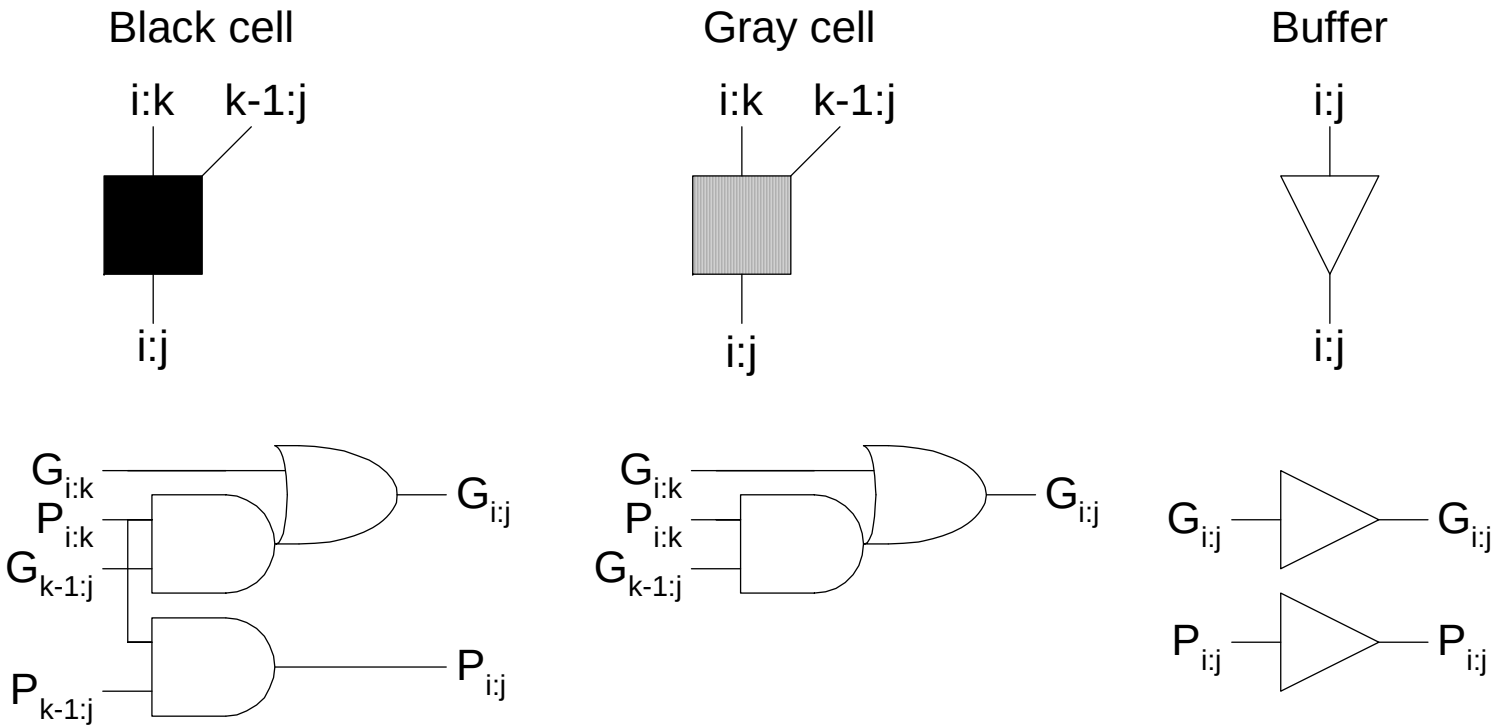
Propagate & generate Sum XOR

$$t_{\text{ripple}} = t_{pg} + (N-1)t_{AO} + t_{\text{xor}}$$

And OR

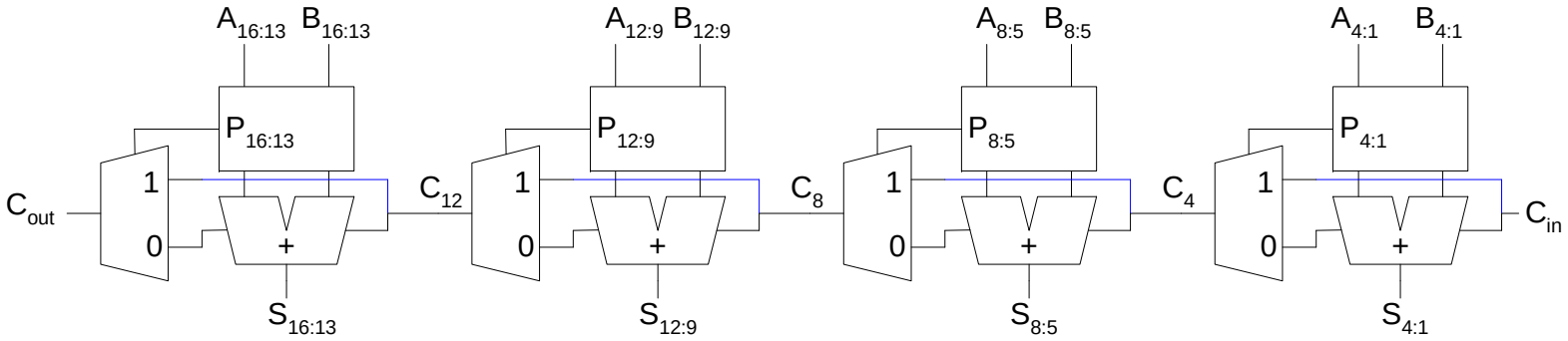


PG Diagram Notation



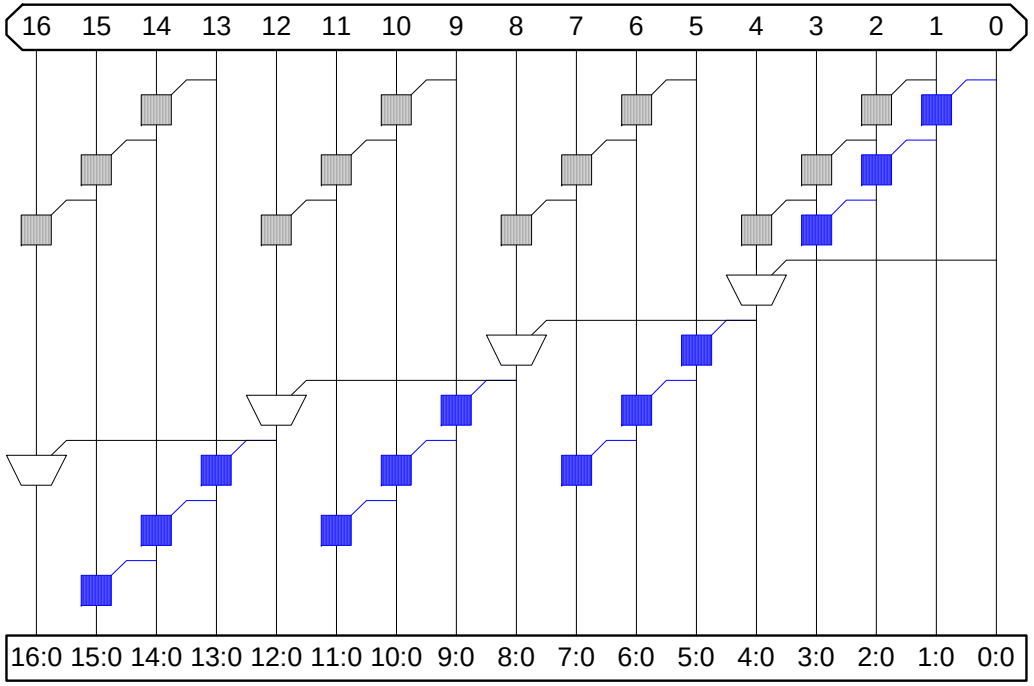
Carry-Skip Adder

- Carry-ripple is slow through all N stages
- Carry-skip allows carry to skip over groups of n bits
  - Decision based on n-bit propagate signal



MUX selects group carry in if group propagate is true otherwise 4bit adder result

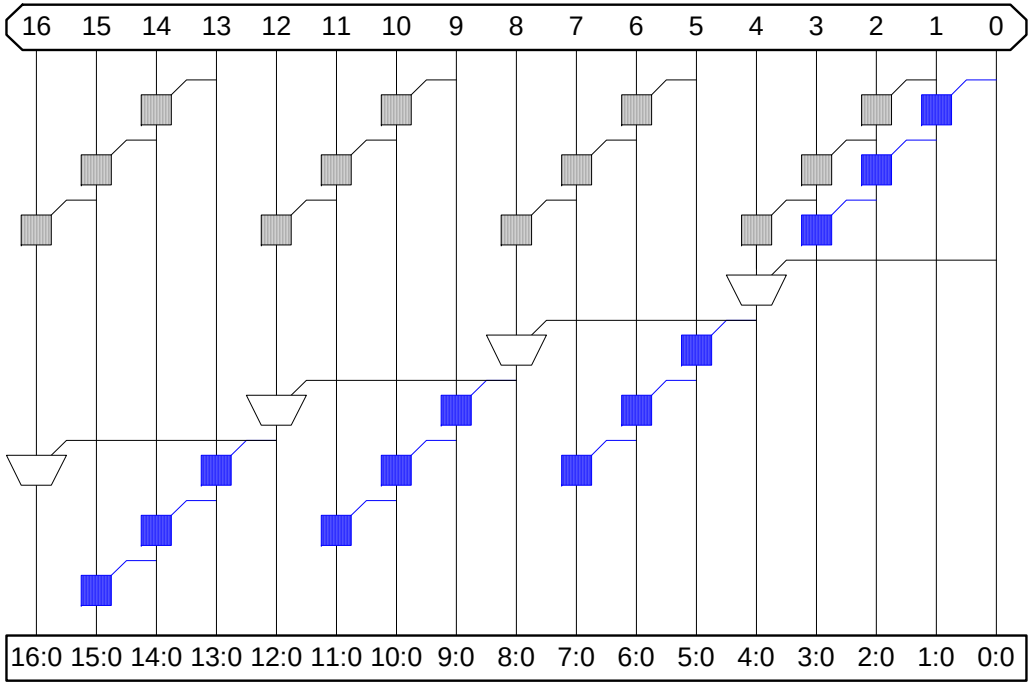
Carry-Skip PG Diagram



For k n-bit groups (N = nk)

$$t_{skip} =$$

Carry-Skip PG Diagram

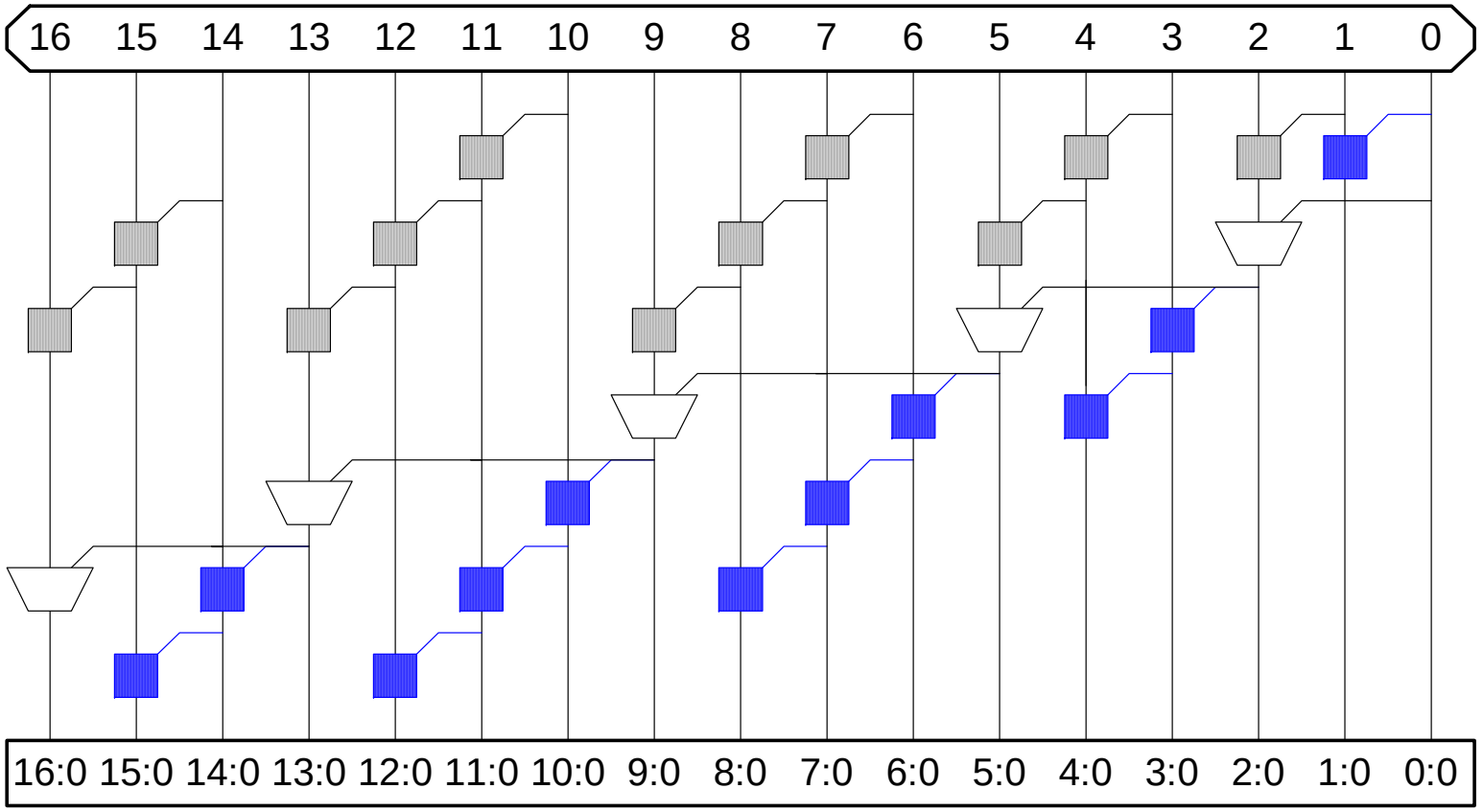


Black: P logic
Blue: Propagate inside 4 bit blocks

For k n-bit groups (N = n (4) * k (4)) (k * n bit groups)

$$t_{\text{skip}} = t_{pg} + \left[2(n-1) + (k-1)\right] t_{AO} + t_{\text{xor}}$$

Variable Group Size

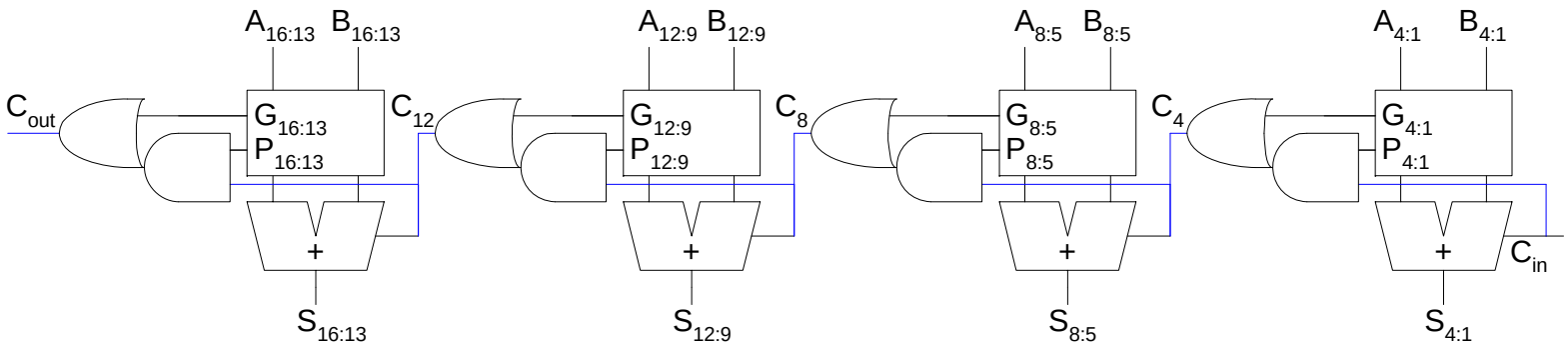


Delay grows as O(sqrt(N))

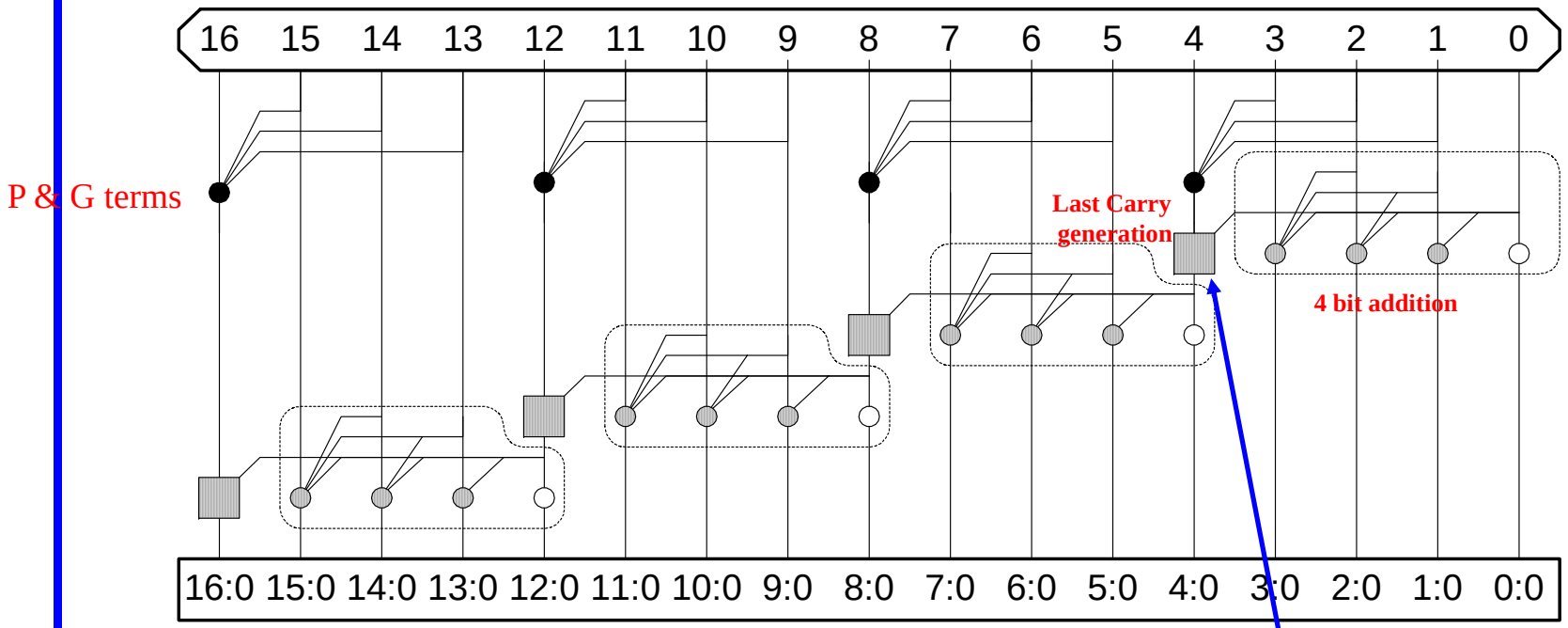
Carry-Lookahead Adder

- Carry-lookahead adder computes $G_{i:0}$ for many bits in parallel.
- Uses higher-valency cells with more than two inputs.

Carry skip waits for the first 4 bit adder
carry out eliminated in CLA

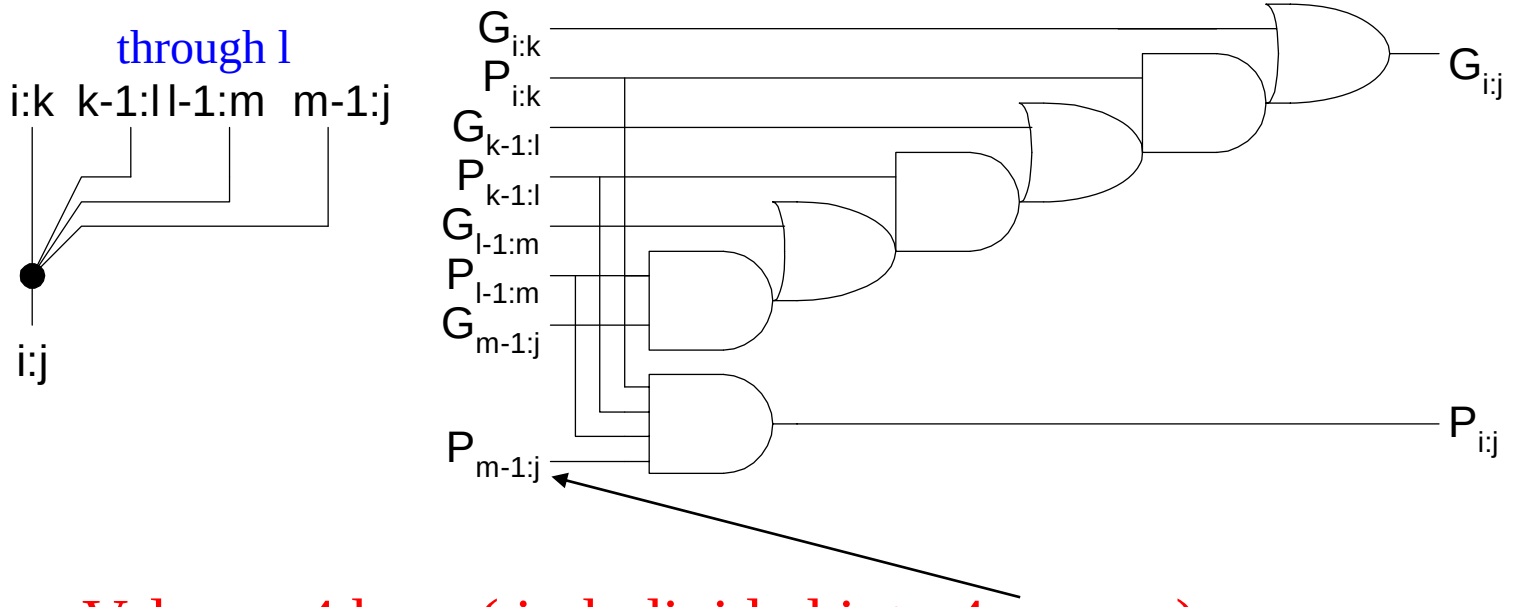


CLA PG Diagram



Higher-Valency Cells

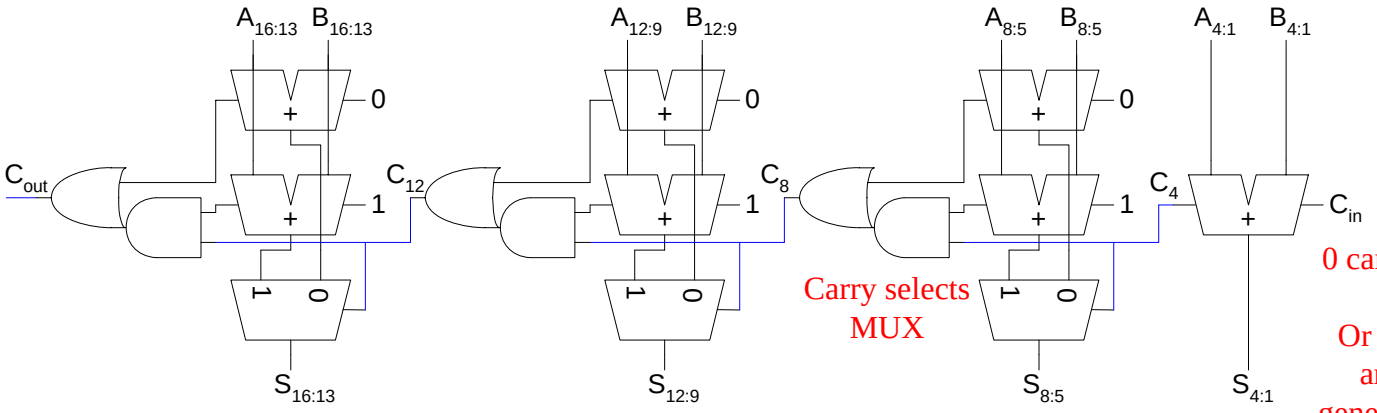
Valency : Relative capacity to unite



Valency 4 here (i : k divided into 4 groups)

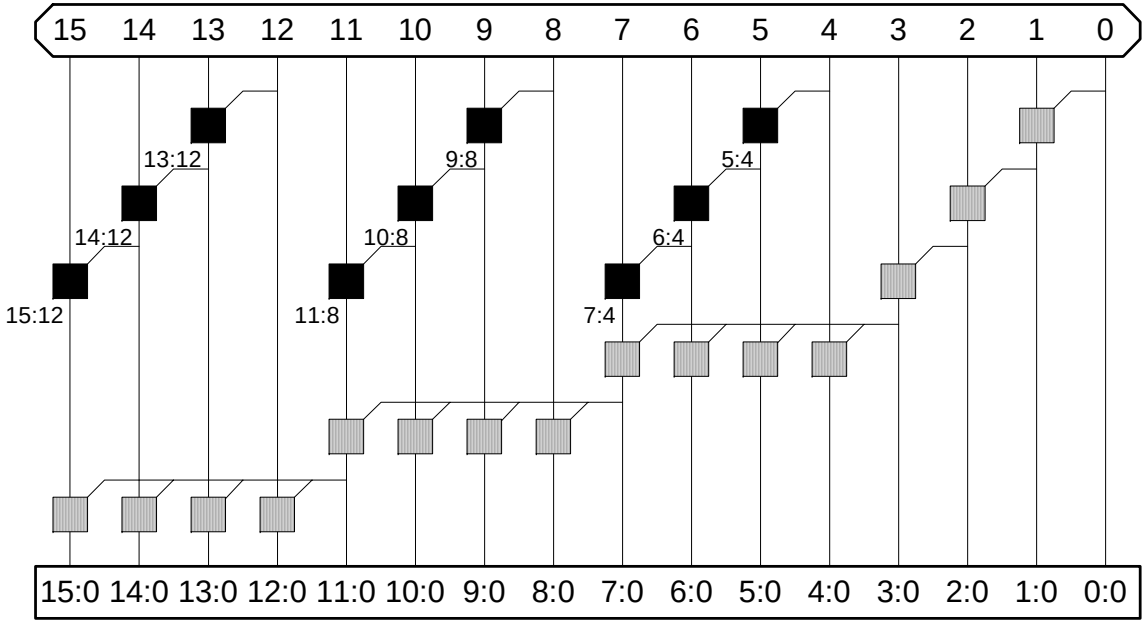
Carry-Select Adder

- Trick for critical paths dependent on late input X
  - Precompute two possible outputs for X = 0, 1
  - Select proper output when X arrives
- Carry-select adder precomputes n-bit sums
  - For both possible carries into n-bit group



Carry-Increment Adder

- Factor initial PG and final XOR out of carry-select

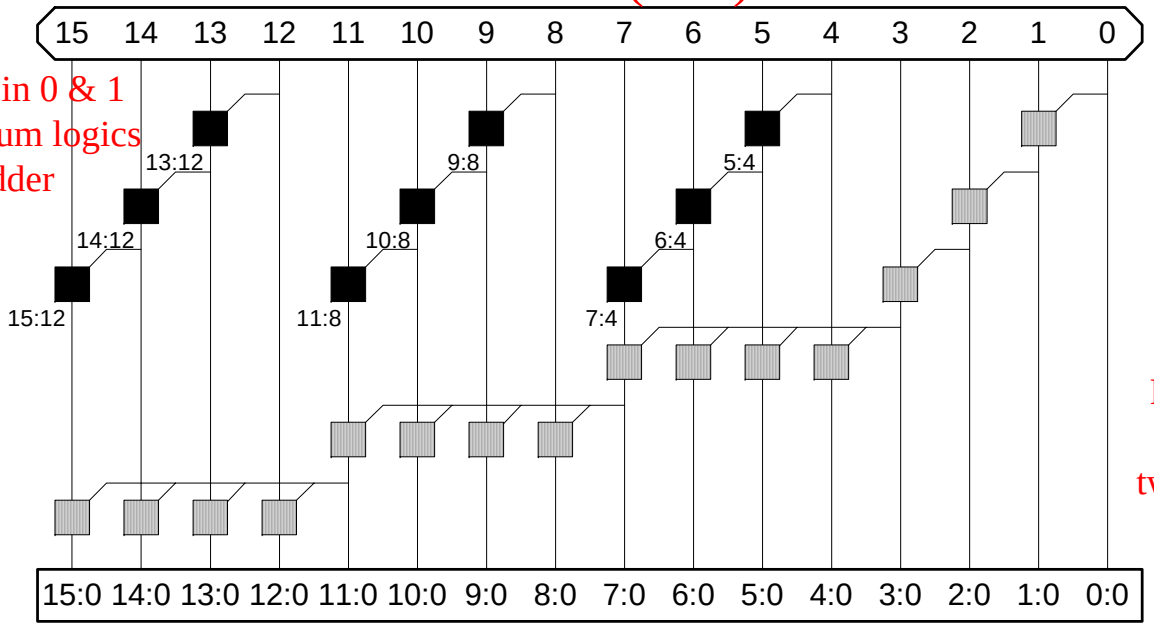


$$t_{\text{increment}} =$$

Carry-Increment Adder

- Factor initial PG (t_{pg}) and final XOR (t_{xor}) out of carry-select (sum)

Two adders for carry in 0 & 1
can share the PG & sum logics
→ carry increment adder



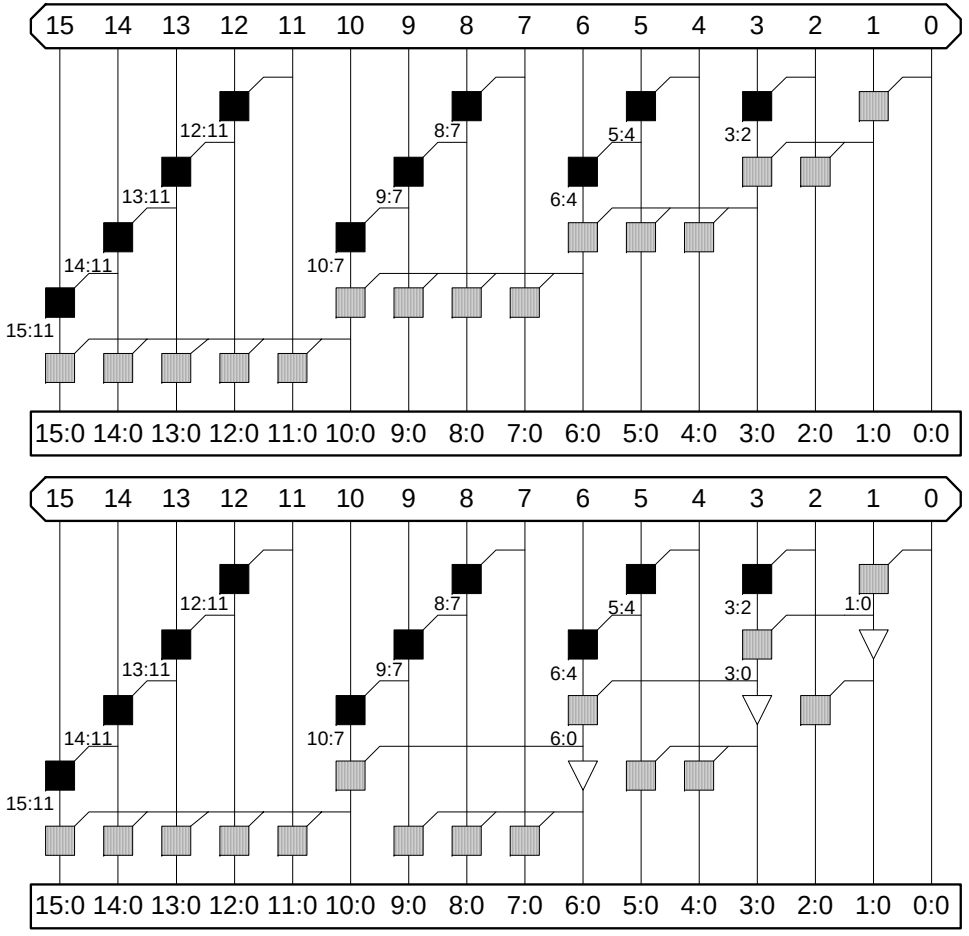
First 4bit addition in series
In parallel with
two possible addition results
(carry in 0 and 1 cases)

$$t_{\text{increment}} = t_{pg} + \left[(n-1) + (k-1) \right] t_{AO} + t_{\text{xor}}$$

Variable Group Size

- Also buffer noncritical signals

Buffered version



Tree Adder

- If lookahead is good, lookahead across lookahead!
  - Recursive lookahead gives O(log N) delay

    (Usually 2 bits P, G generation in parallel and sum them up from lsb to msb considering delay, higher bit sum up in parallel with lower bit sum up)
- Many variations on tree adders

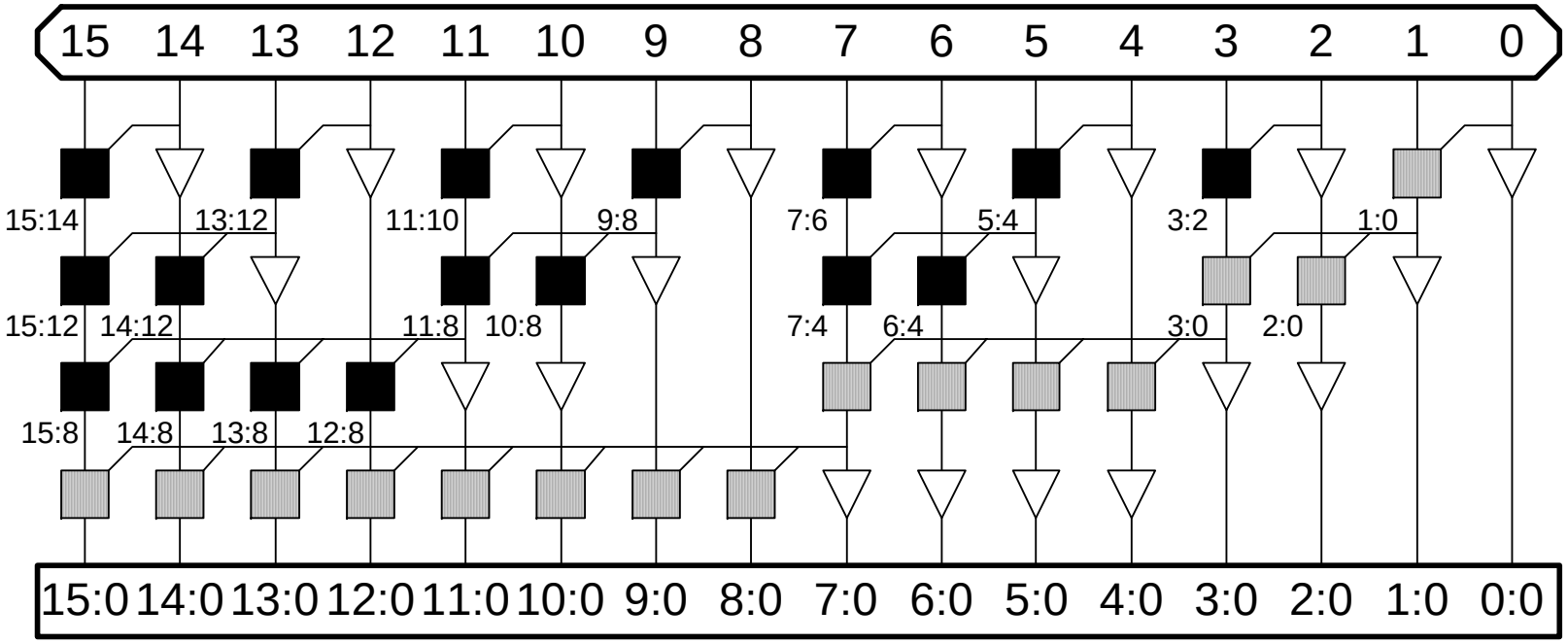
Tree Adder Taxonomy

(Taxonomy : Division into ordered groups or categories)

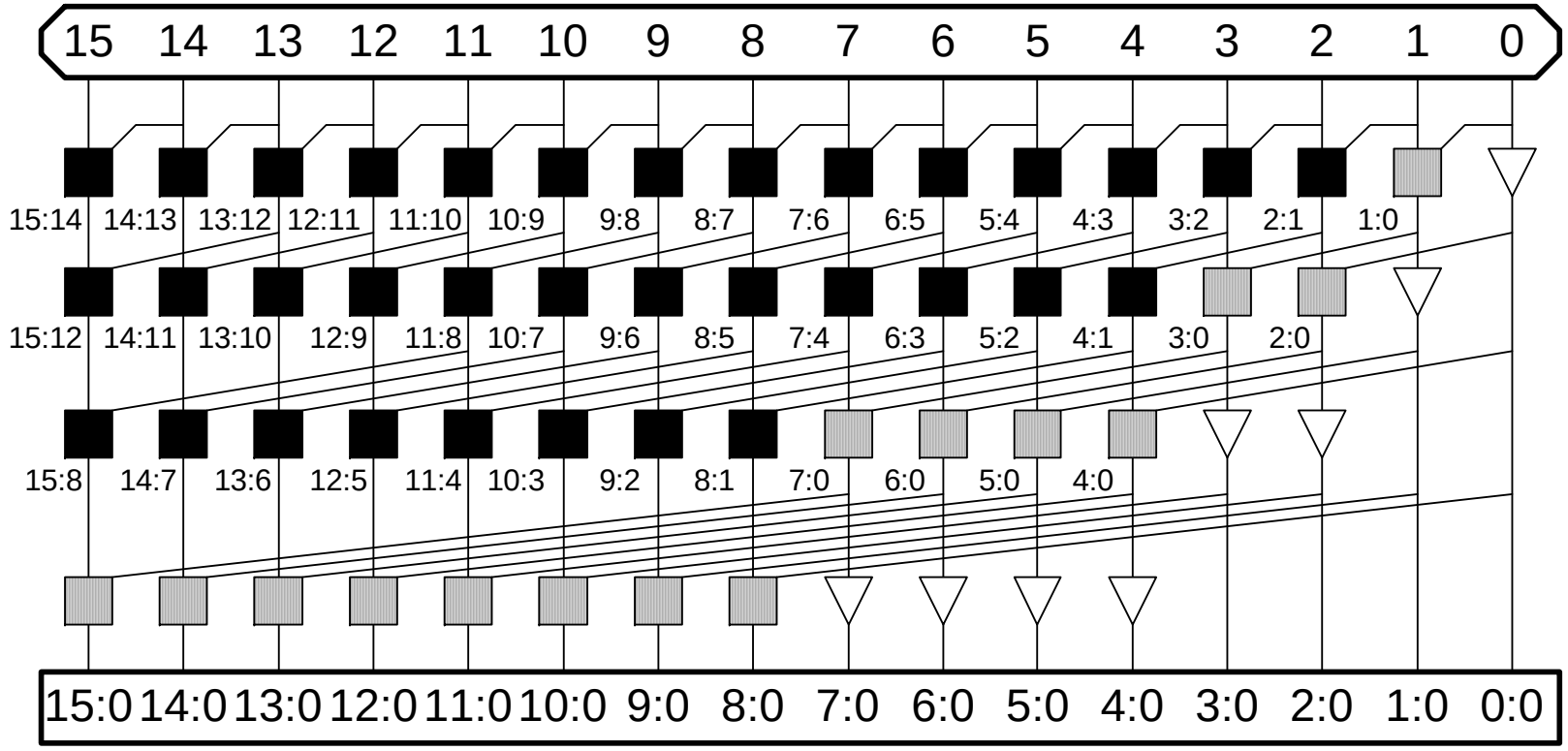
- Ideal N-bit tree adder would have
  - $L = \log N$ logic levels
  - Fanout never exceeding 2
  - No more than one wiring track between levels
- Describe adder with 3-D taxonomy (l, f, t)
  - Logic levels: $L + l$ *(not 1)* (MAX. # of logic levels to the result)
  - Fanout: $2^f + 1$ (MAX. # of gates one cell drives)
  - Wiring tracks: 2^t (MAX. # of wires in a level)
- Known tree adders sit on plane defined by

$$l + f + t = L\text{-}1$$

Sklansky


15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
15:14
13:12
11:10
9:8
7:6
5:4
3:2
1:0
15:12
14:12
11:8
10:8
7:4
6:4
3:0
2:0
15:8
14:8
13:8
12:8
15:0 14:0 13:0 12:0 11:0 10:0 9:0 8:0 7:0 6:0 5:0 4:0 3:0 2:0 1:0 0:0

Kogge-Stone

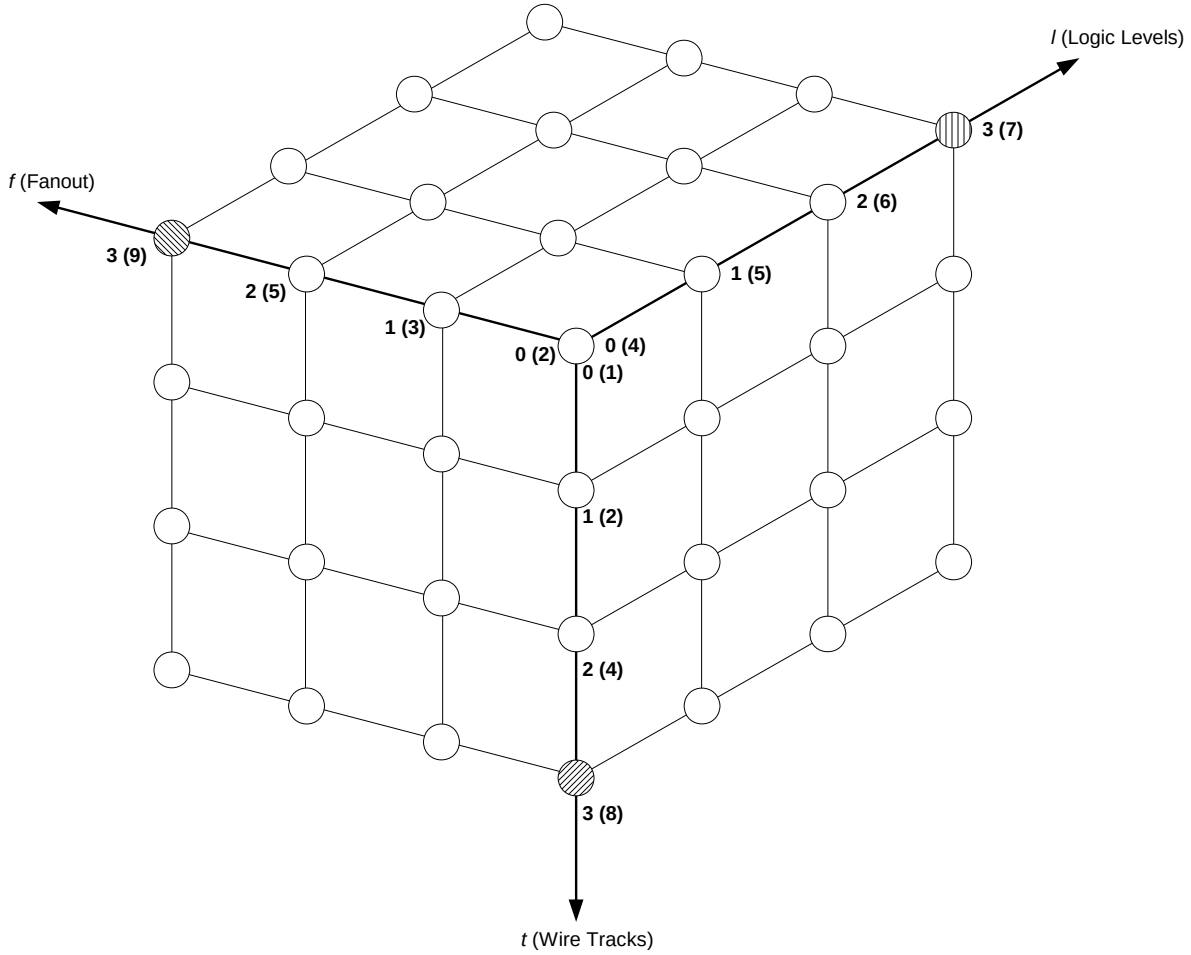

15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
15:14 14:13 13:12 12:11 11:10 10:9 9:8 8:7 7:6 6:5 5:4 4:3 3:2 2:1 1:0
15:12 14:11 13:10 12:9 11:8 10:7 9:6 8:5 7:4 6:3 5:2 4:1 3:0 2:0
15:8 14:7 13:6 12:5 11:4 10:3 9:2 8:1 7:0 6:0 5:0 4:0
15:0 14:0 13:0 12:0 11:0 10:0 9:0 8:0 7:0 6:0 5:0 4:0 3:0 2:0 1:0 0:0

Tree Adder Taxonomy

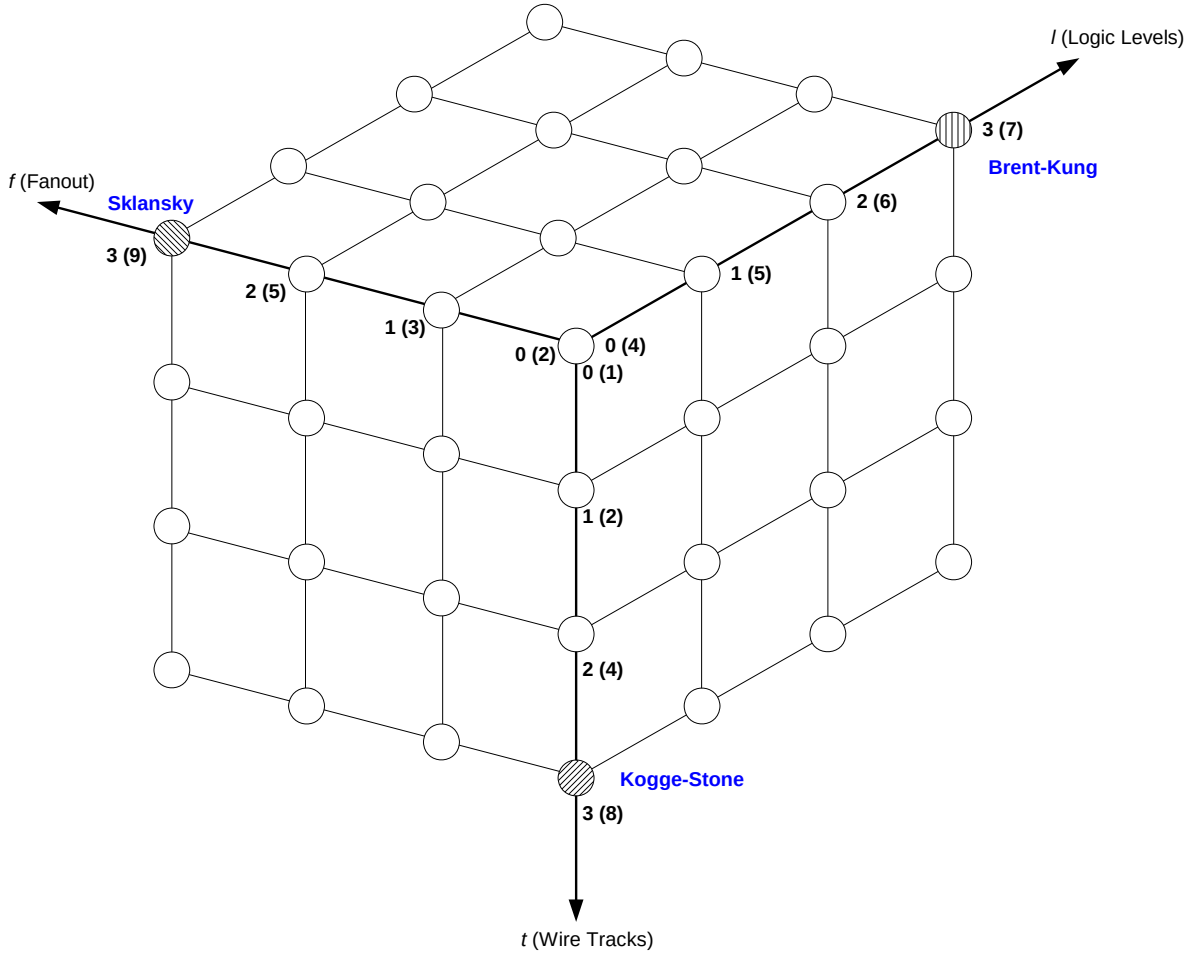
- Ideal N-bit tree adder would have
  - $L = \log N$ logic levels
  - Fanout never exceeding 2
  - No more than one wiring track between levels
- Describe adder with 3-D taxonomy (l, f, t)
  - Logic levels: $L + l$
  - Fanout: $2^f + 1$
  - Wiring tracks: 2^t
- Known tree adders sit on plane defined by

$$l + f + t = L\text{-}1$$

Tree Adder Taxonomy



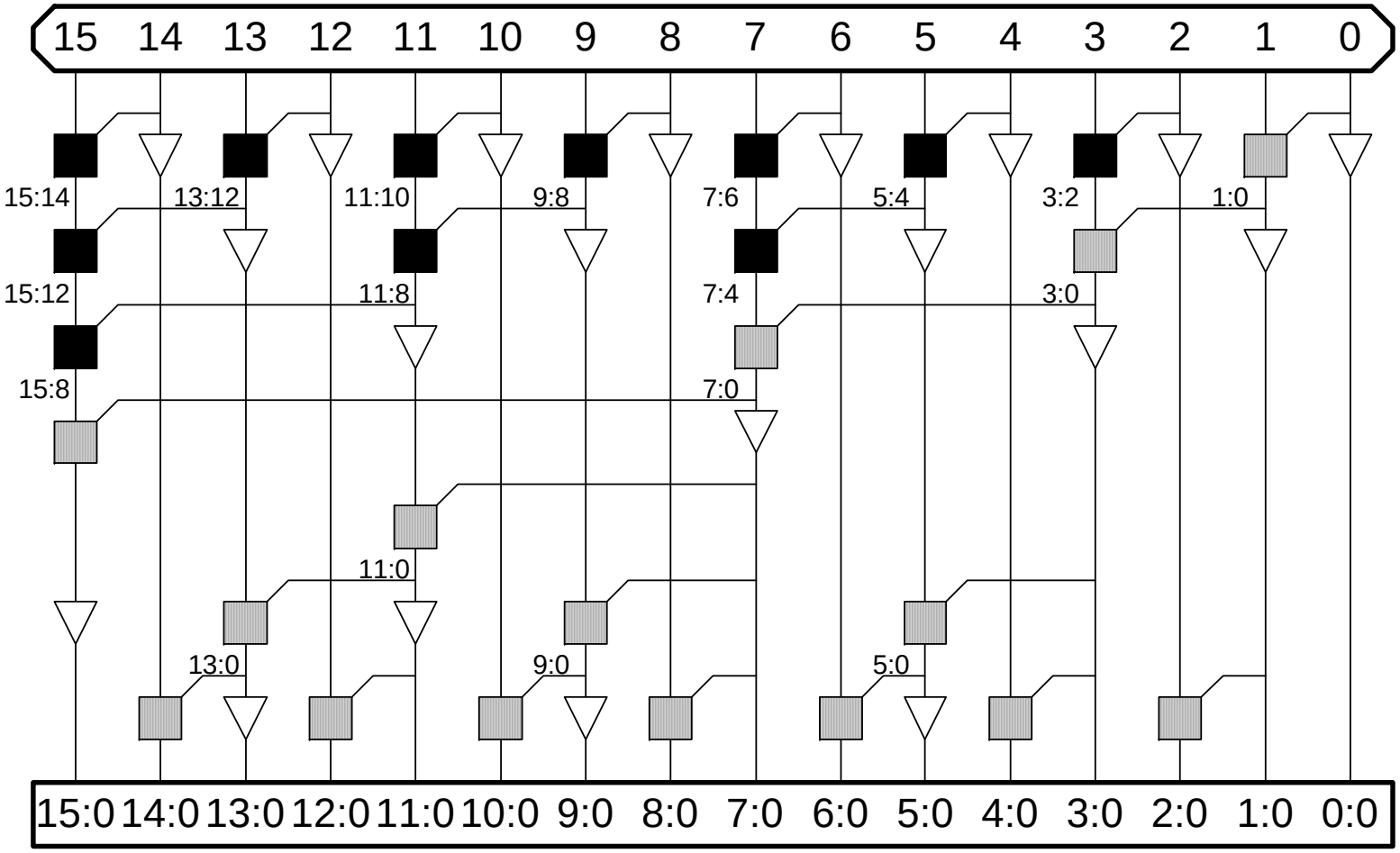
Tree Adder Taxonomy



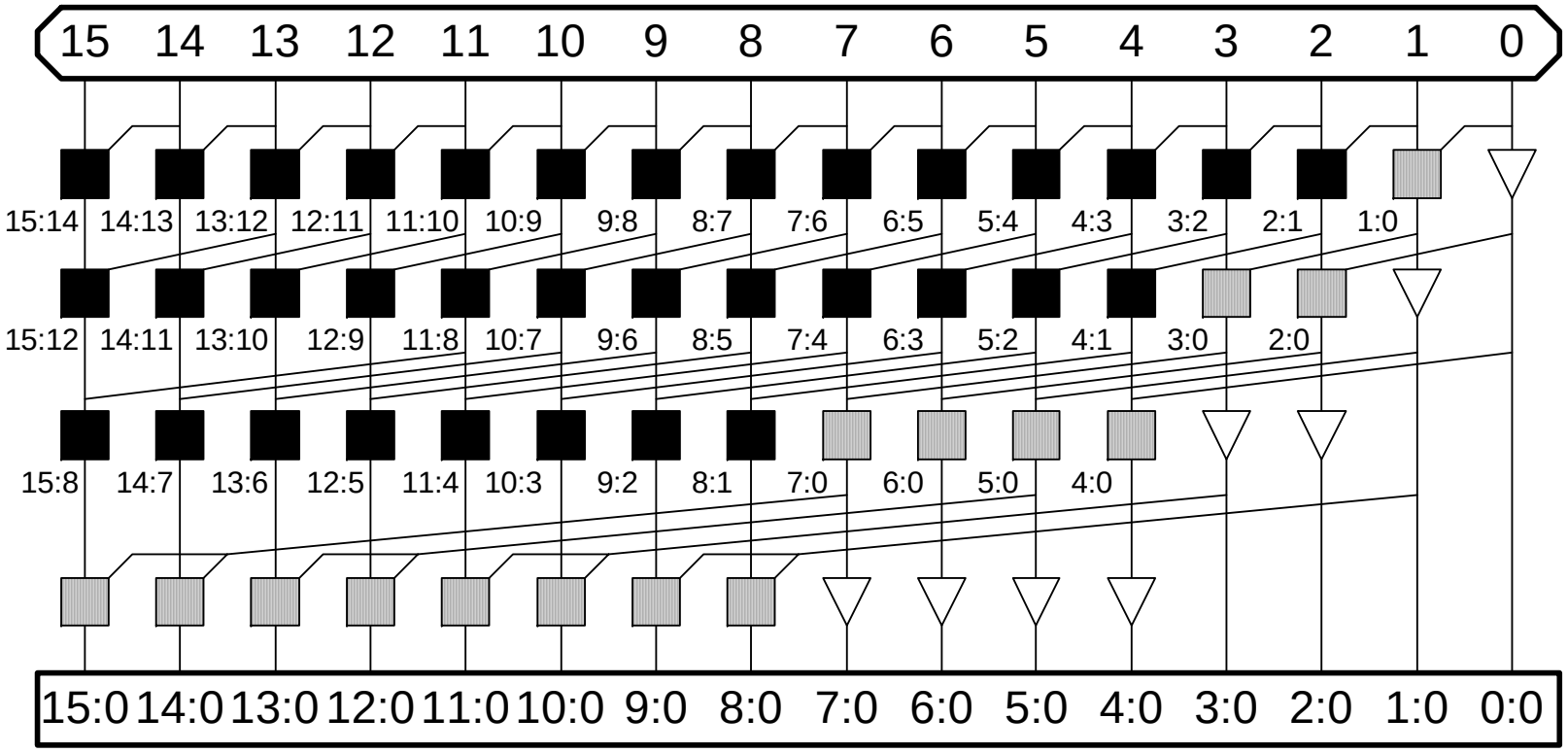
$$S_i = P_i \oplus G_{i-1:0}$$

Brent-Kung (moved after pp.34)

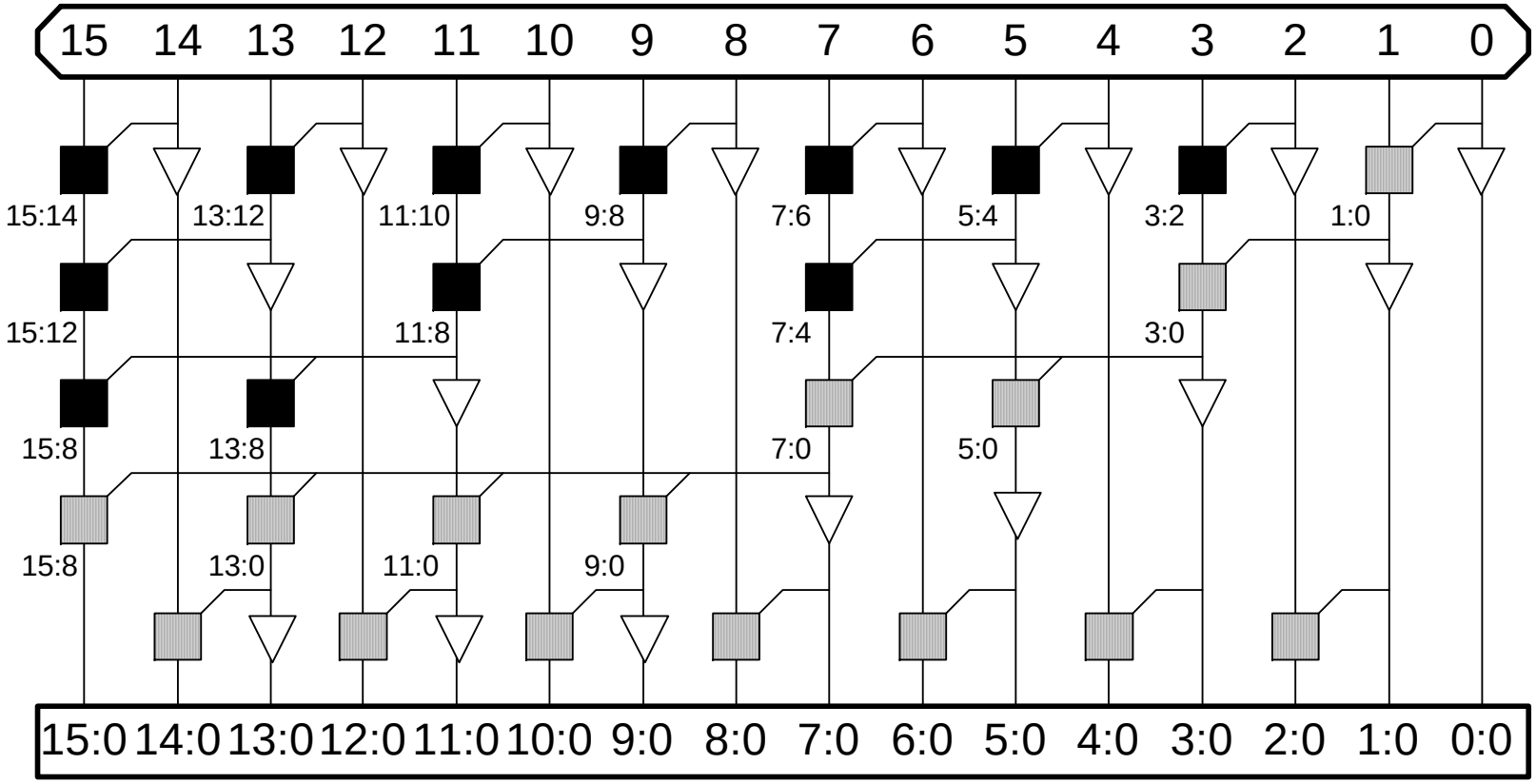
(prefix 2 → prefix 4 → ...)



Knowles [2, 1, 1, 1]

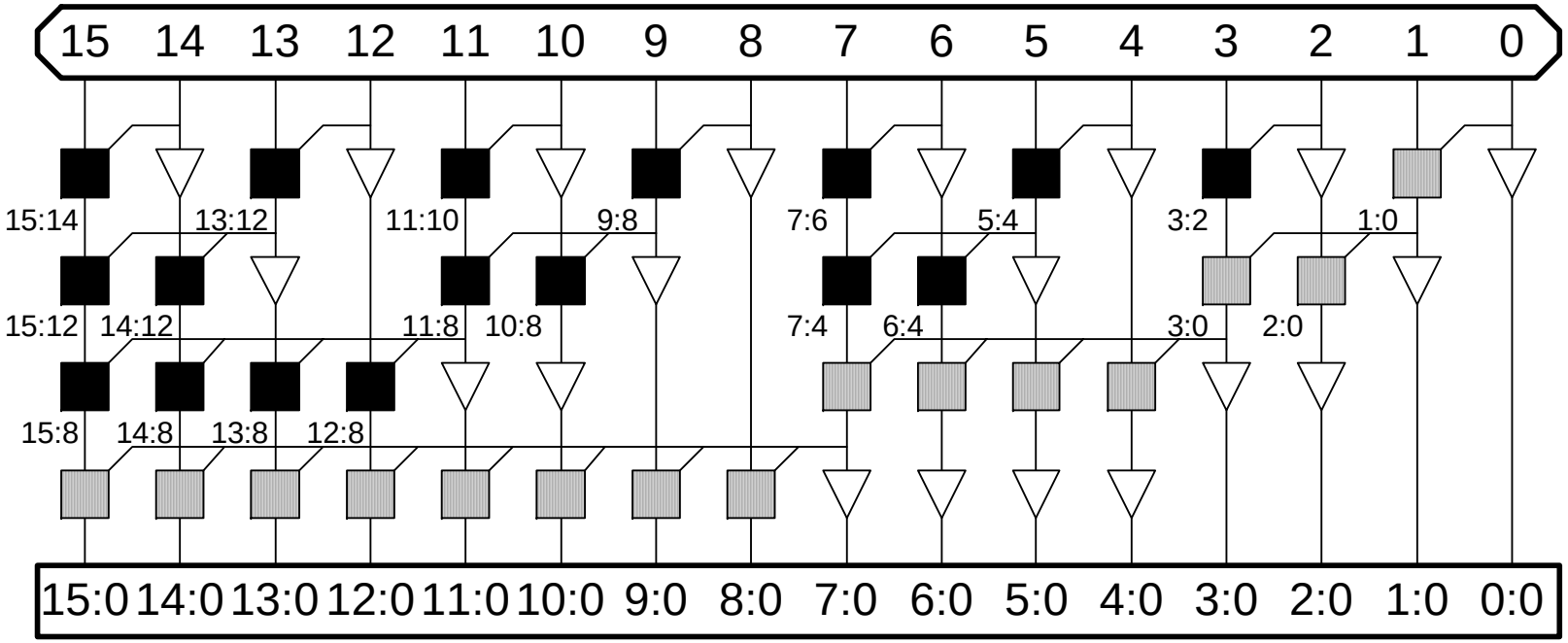


Ladner-Fischer



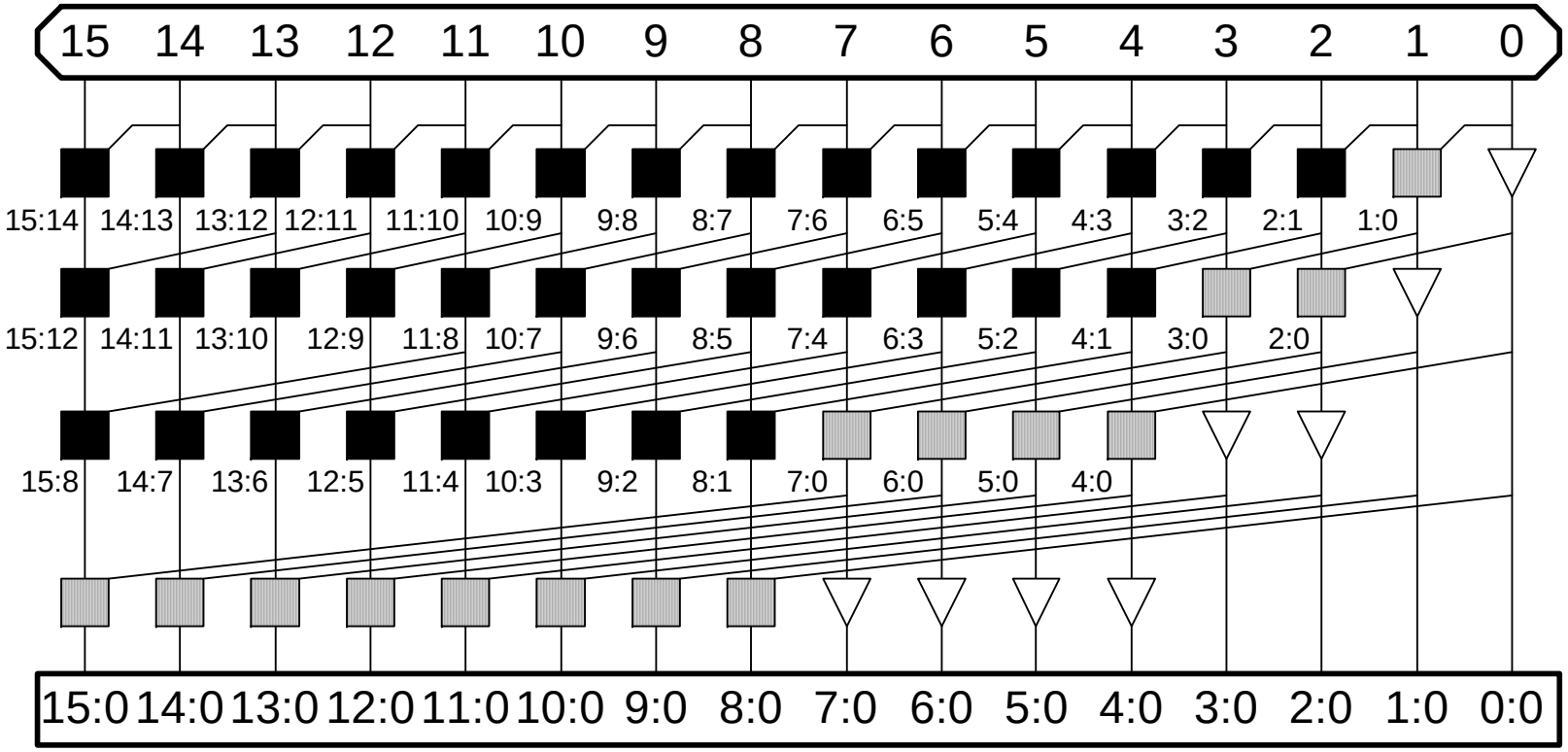
Sklansky

(moved after pp.34) (go to pp.43)



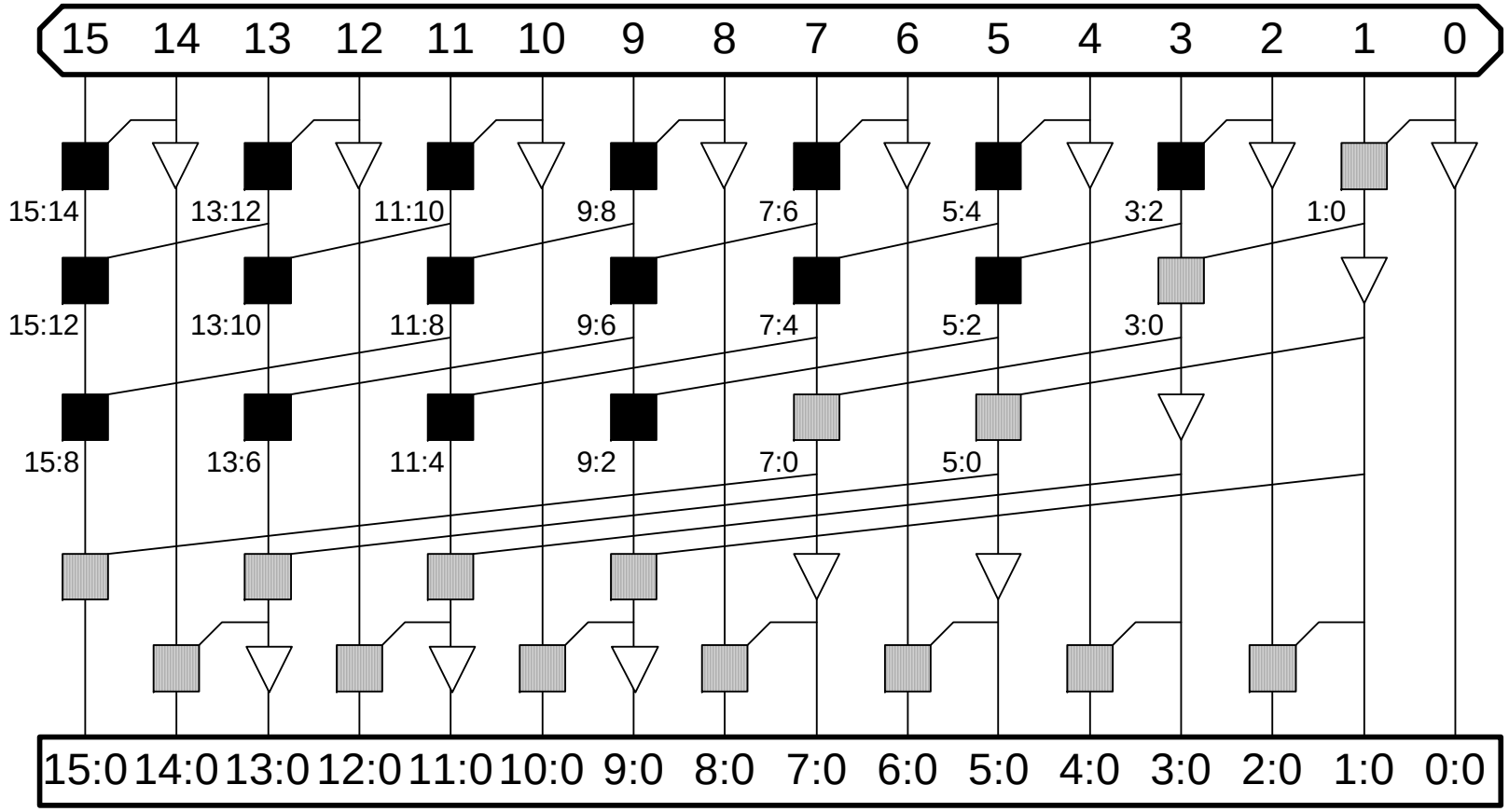
Intermediate Prefix
with large fan out

Kogge-Stone (moved after pp.34)

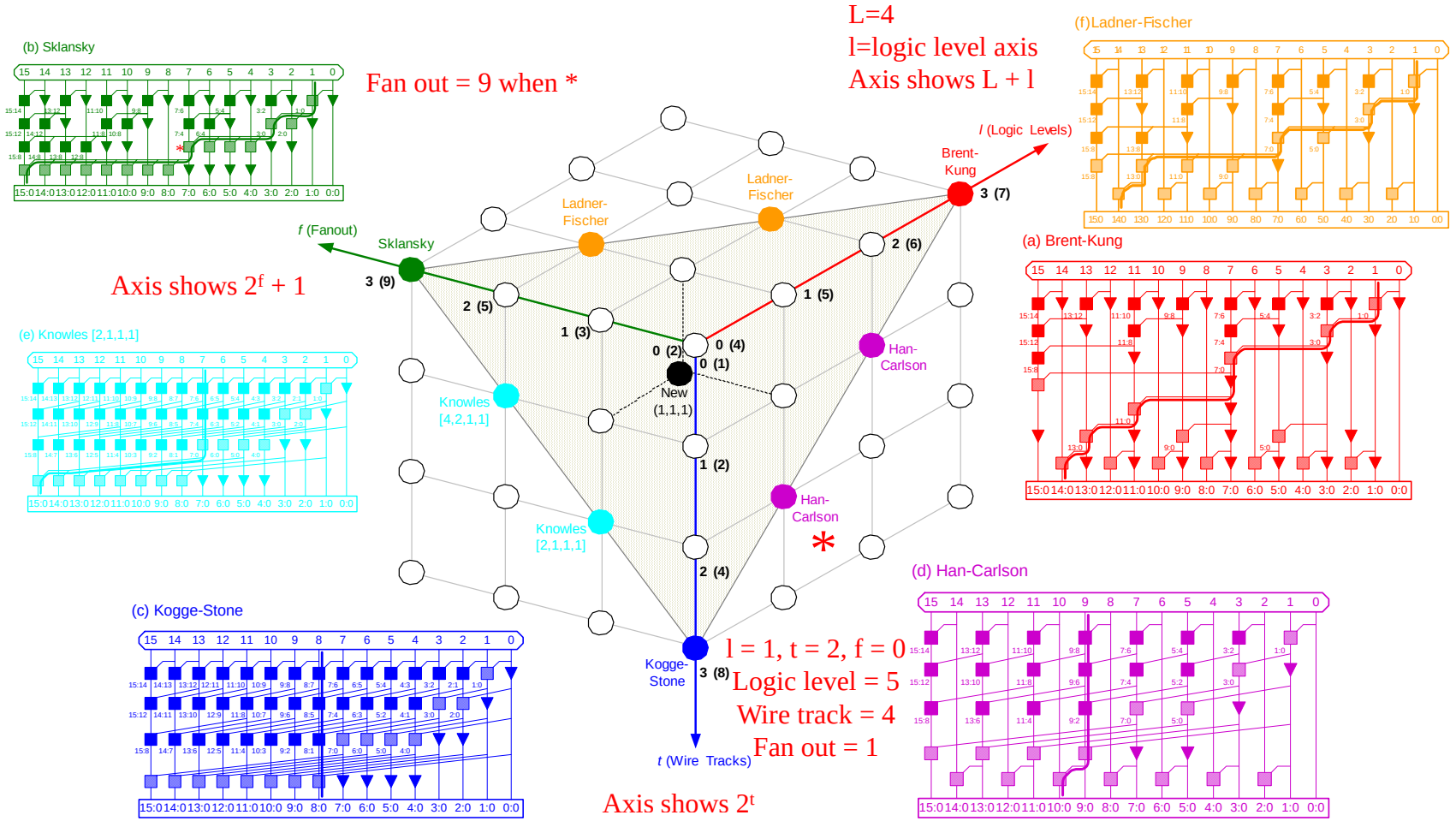


Many long wires, large power consumption

Han-Carlson


15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
15:14 13:12 11:10 9:8 7:6 5:4 3:2 1:0
15:12 13:10 11:8 9:6 7:4 5:2 3:0
15:8 13:6 11:4 9:2 7:0 5:0
15:0 14:0 13:0 12:0 11:0 10:0 9:0 8:0 7:0 6:0 5:0 4:0 3:0 2:0 1:0 0:0

Taxonomy Revisited



Summary

Adder architectures offer area / power / delay tradeoffs.

Choose the best one for your application.

Architecture	Classification	Logic Levels	Max Fanout	Tracks	Cells
Carry-Ripple		N-1	1	1	N
Carry-Skip n=4		N/4 + 5	2	1	1.25N
Carry-Inc. n=4		N/4 + 2	4	1	2N
Brent-Kung	(L-1, 0, 0)	$2\log_2 N - 1$	2	1	2N
Sklansky	(0, L-1, 0)	$\log_2 N$	N/2 + 1	1	$0.5\ N\log_2 N$
Kogge-Stone	(0, 0, L-1)	$\log_2 N$	2	N/2	$N\log_2 N$