Lecture 11:
Adder Design

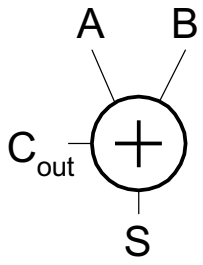
Mark McDermott
Electrical and Computer Engineering
The University of Texas at Austin

Single-Bit Addition

Half Adder

$$S = A \otimes B$$

$$C_{out} = A \cdot B$$

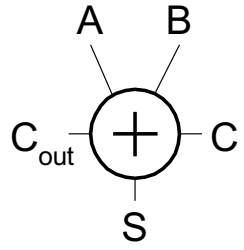


A	B	C_{out}	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

Full Adder

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

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



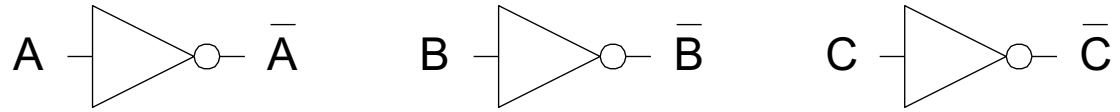
A	B	C	G	P	K	C_{out}	S
0	0	0	0	0	1	0	0
0	0	1	0	0	1	0	1
0	1	0	0	1	0	0	1
0	1	1	0	1	0	1	0
1	0	0	0	1	0	0	1
1	0	1	0	1	0	1	0
1	1	0	1	0	0	1	0
1	1	1	1	0	0	1	1

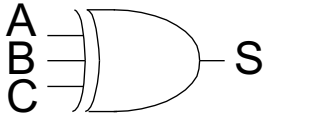
Full Adder Design I

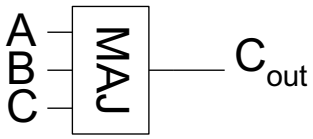
- **Brute force implementation from equations**

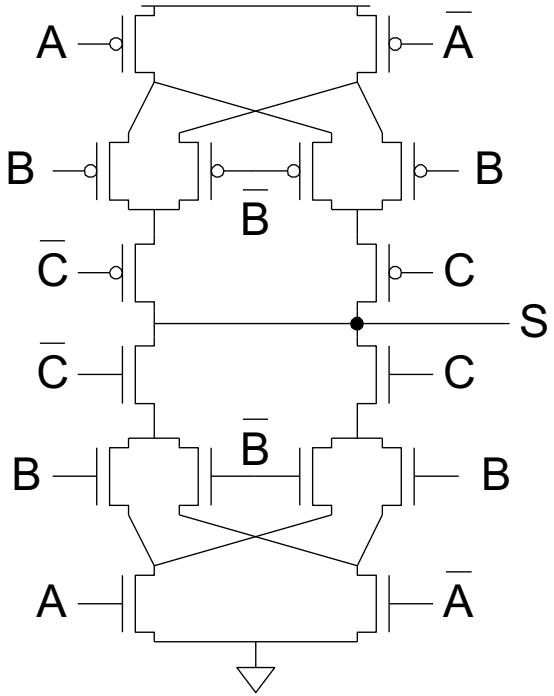
$$S = A \oplus B \oplus C$$

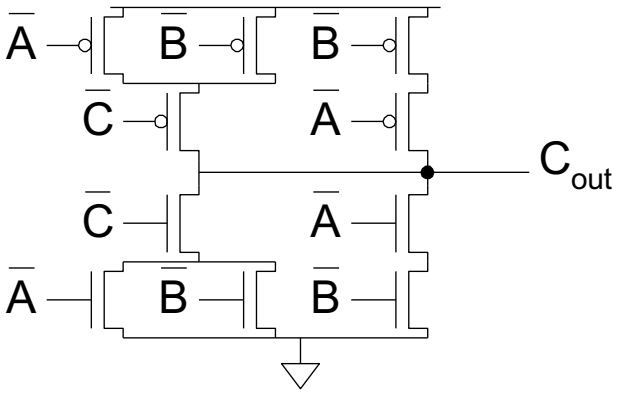
$$C_{out} = MAJ(A, B, C)$$





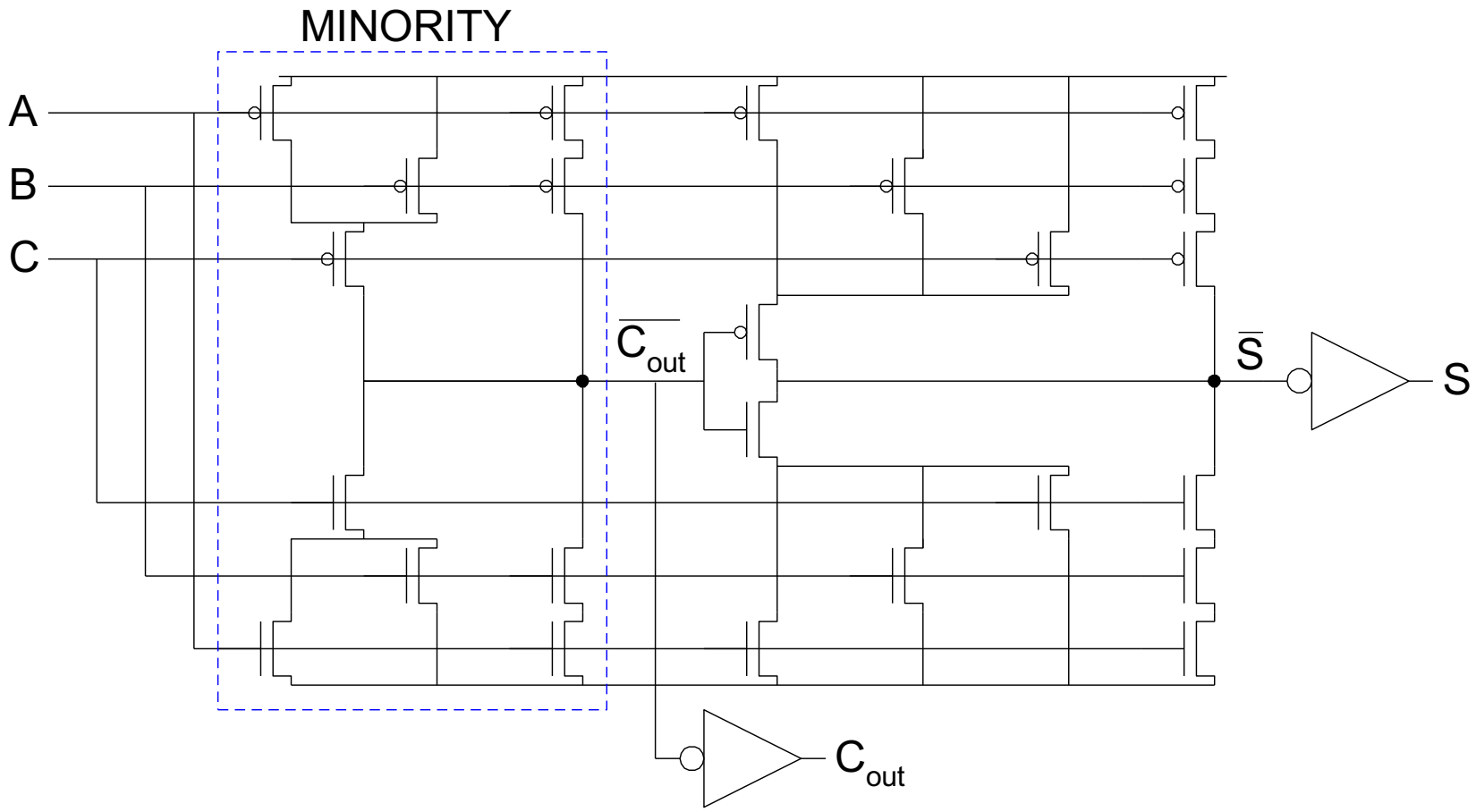






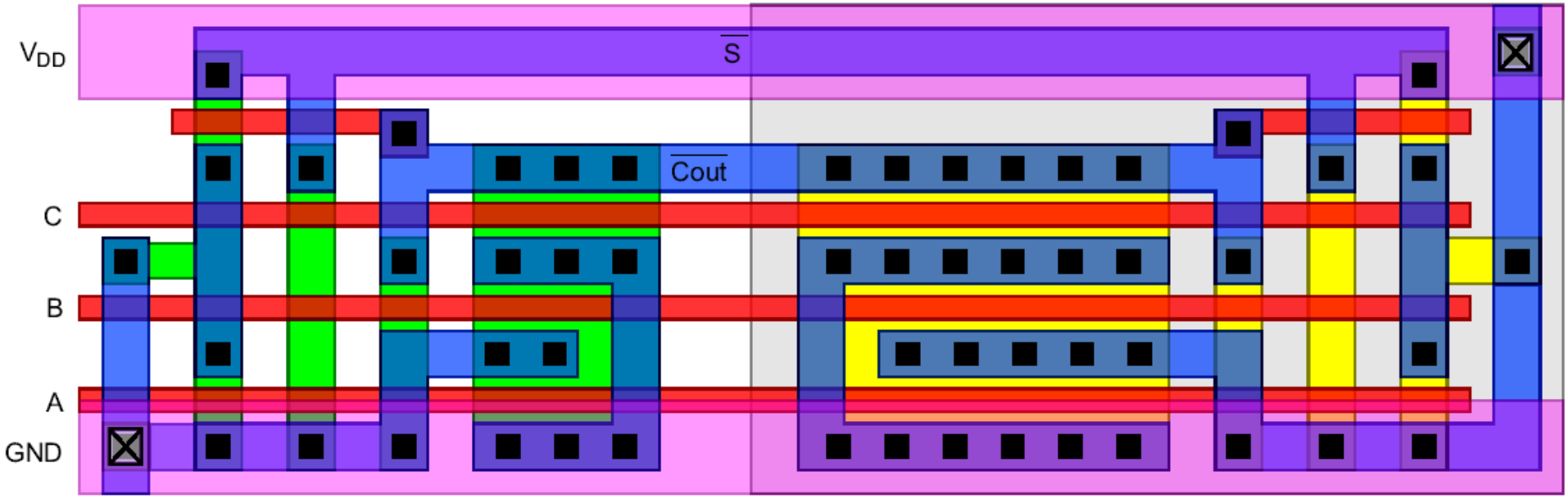
Full Adder Design II

- **Factor S in terms of Cout**
  - **S = ABC + (A + B + C)(~Cout)**
- **Critical path is usually C to Cout in ripple adder**



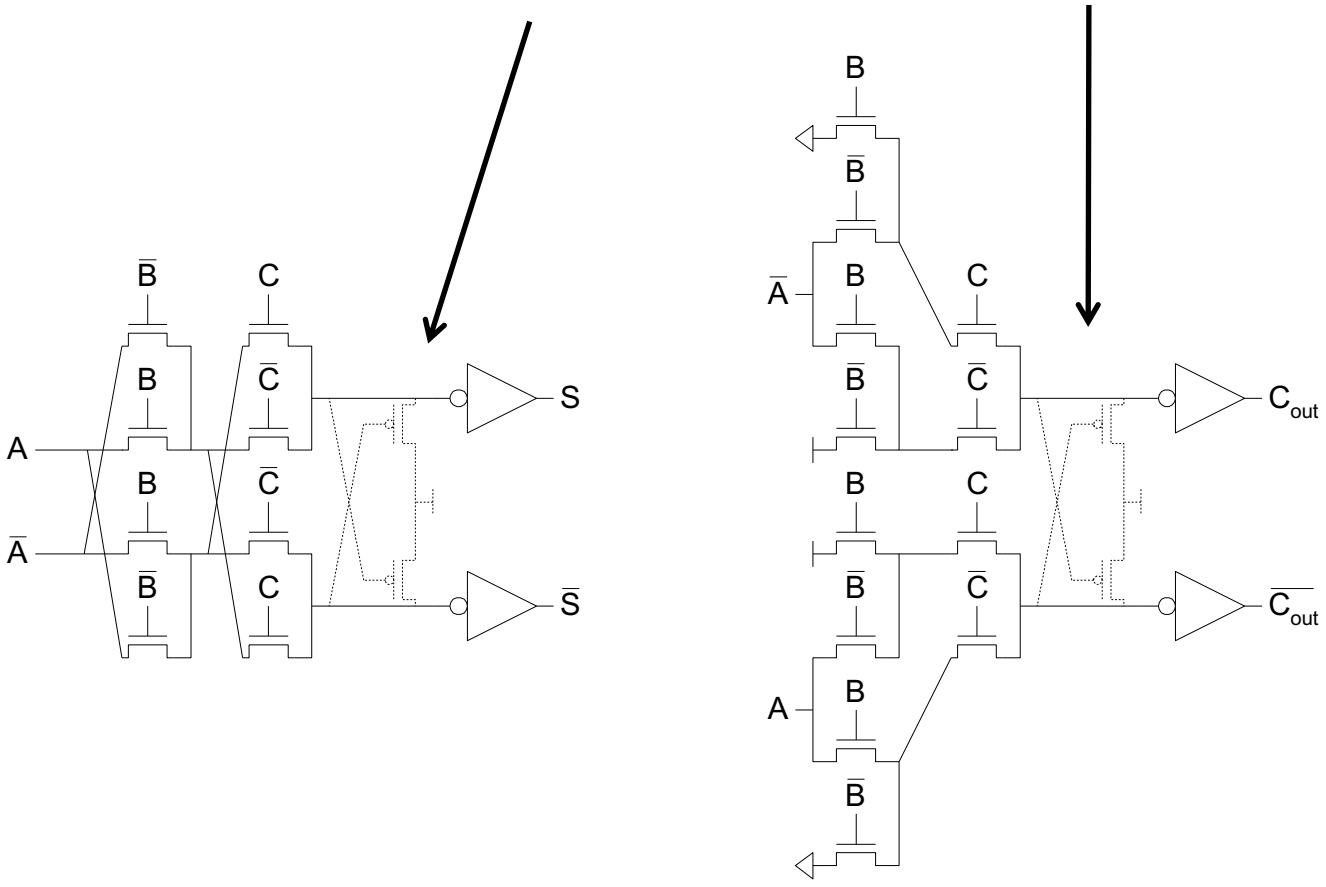
Layout

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



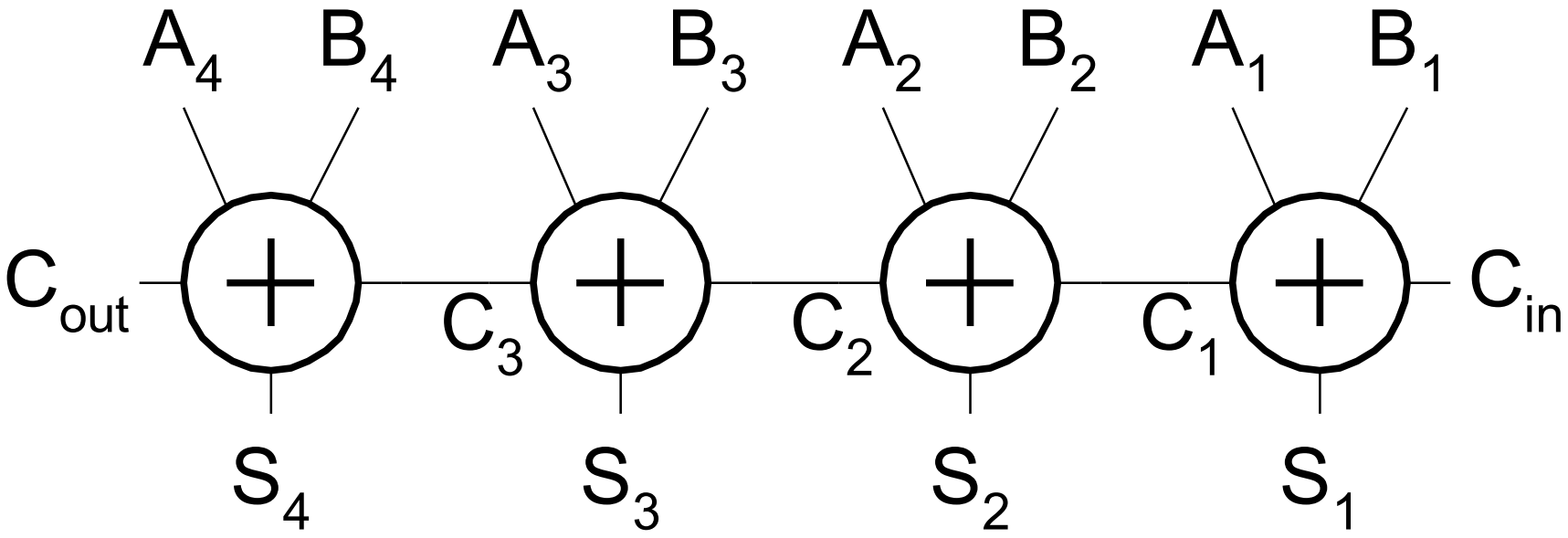
Full Adder Design III

- **Complementary Pass Transistor Logic (CPL)**
  - Slightly faster, but more area
  - Requires true and complement signals
  - There is some signal degradation through the pass transistors.
    - Cross coupled PMOS devices are used to restore logic levels.



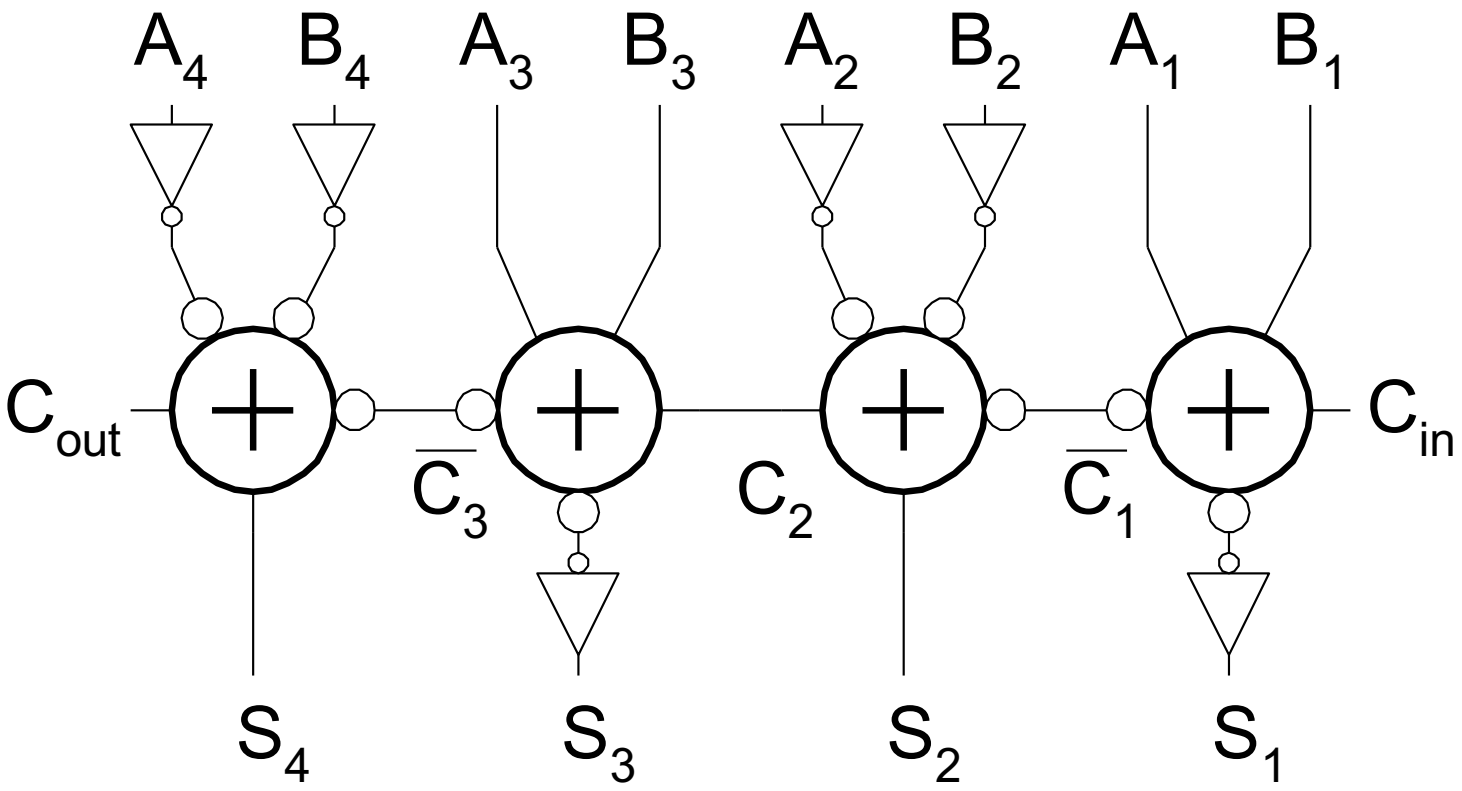
Ripple Carry Adder

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



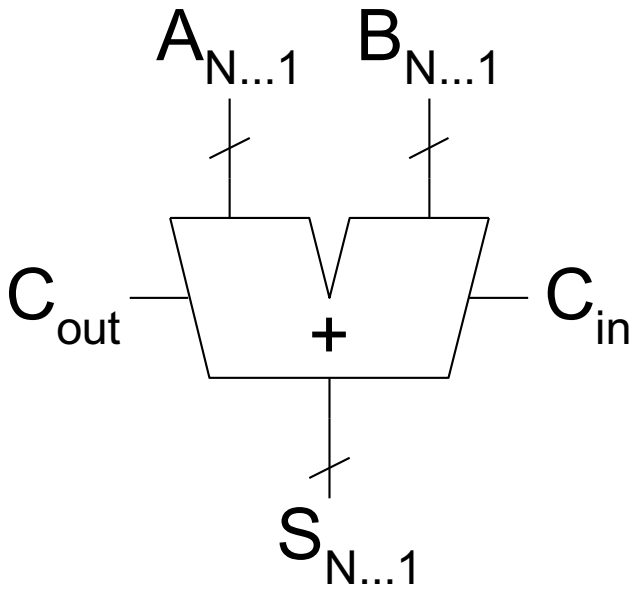
Use Inversions to speed up Carry Path

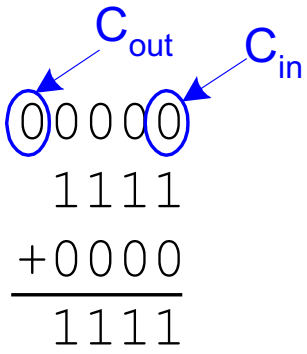
- **Critical path passes through majority gate**
  - **Built from minority + inverter**
  - **Eliminate inverter and use inverting full adder**

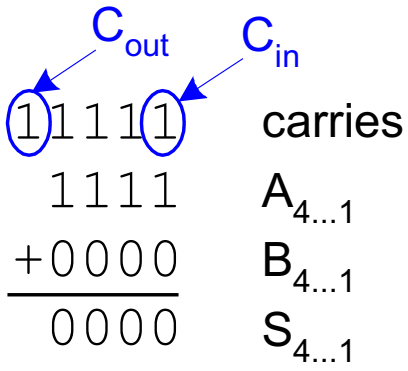


Carry Propagate Adders

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







Carry Propagate, Generate, Kill (PGK)

- **For a full adder, define what happens to carries**

  Generate: $C_{out} = 1$ independent of C

  G = A • B

  Propagate: $C_{out} = C$

  P = A ⊕ B

  Kill: $C_{out} = 0$ independent of C

  K = ~A • ~B (i.e., ~K = A + B)

A	B	C	G	P	K	C_{out}	S
0	0	0	0	0	1	0	0
0	0	1	0	0	1	0	1
0	1	0	0	1	0	0	1
0	1	1	0	1	0	1	0
1	0	0	0	1	0	0	1
1	0	1	0	1	0	1	0
1	1	0	1	0	0	1	0
1	1	1	1	0	0	1	1

Propagate / Generate (PG)

- **Equations often factored into P and G**
- **Propagate and generate for groups spanning i:j**

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

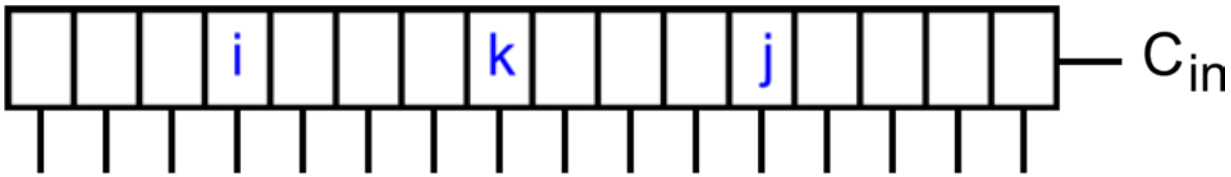
$$P_{i:j} = P_{i:k} \bullet P_{k-1:j}$$

- **Base case**

$$G_{i:i} \equiv G_i = A_i \bullet B_i \qquad G_{0:0} \equiv G_0 = C_{in}$$

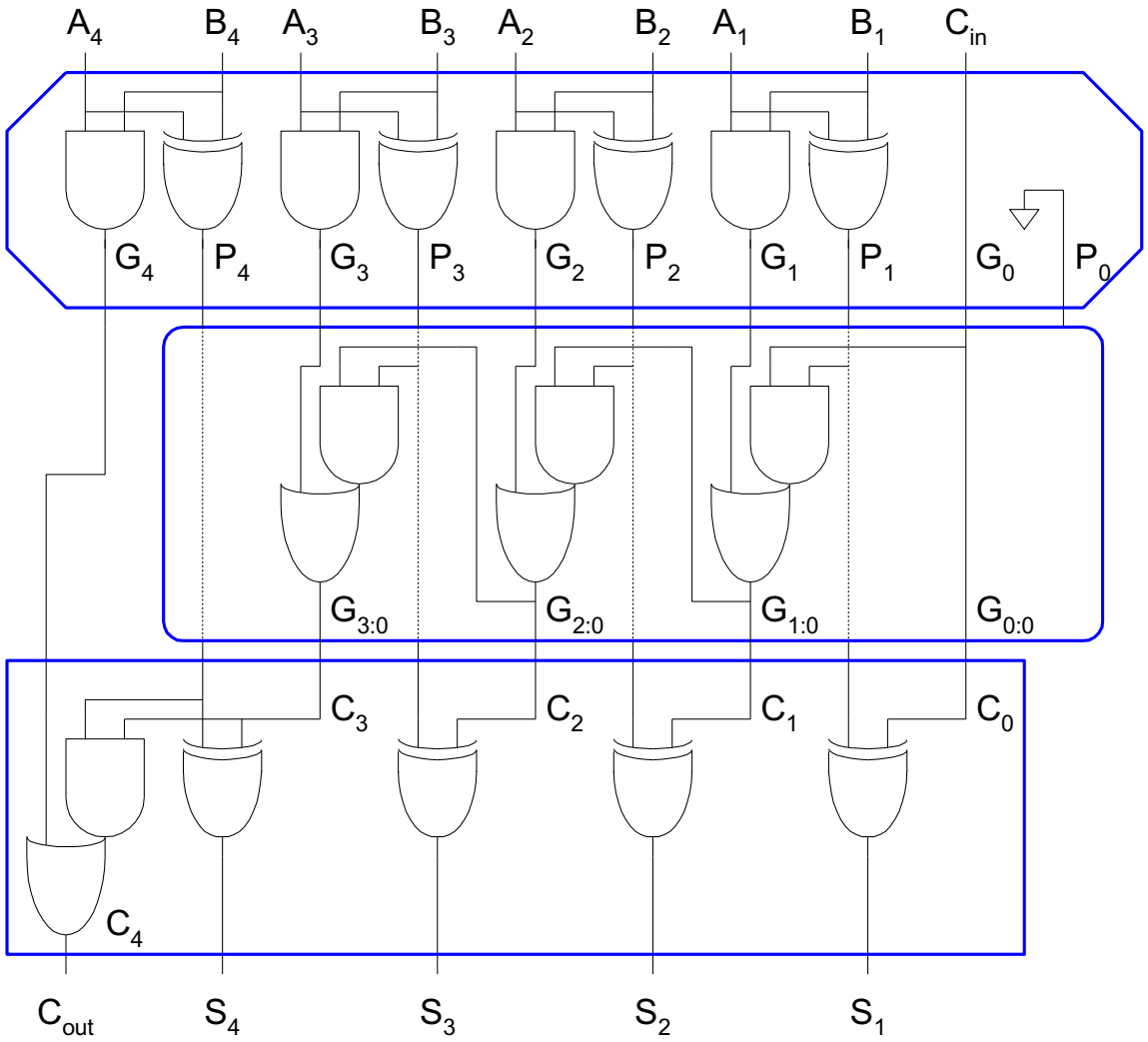
$$P_{i:i} \equiv P_i = A_i \oplus B_i \qquad P_{0:0} \equiv P_0 = 0$$

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



Ripple Carry Revisited in PG Framework

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

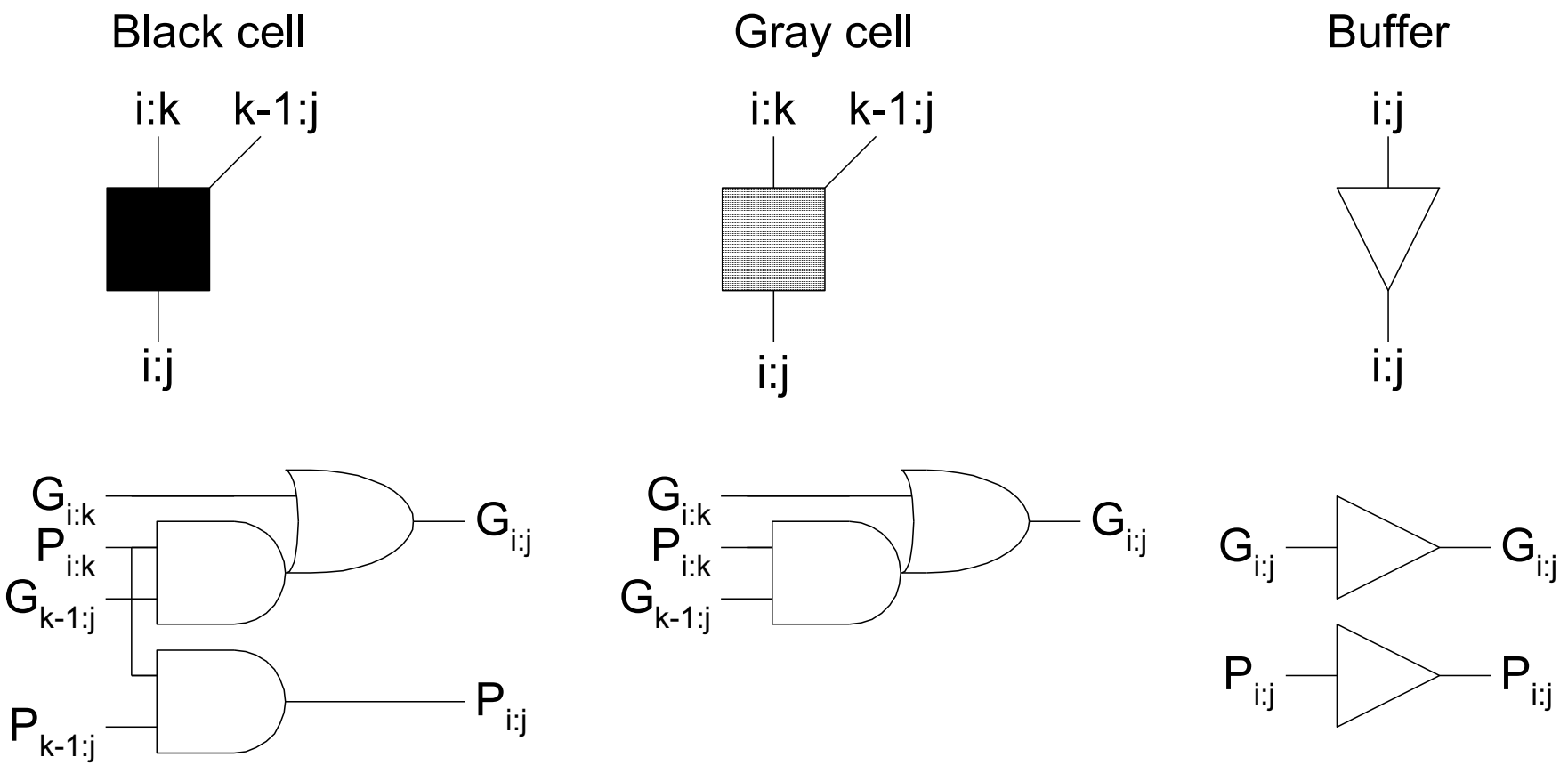


1: Bitwise PG logic

2: Group PG logic

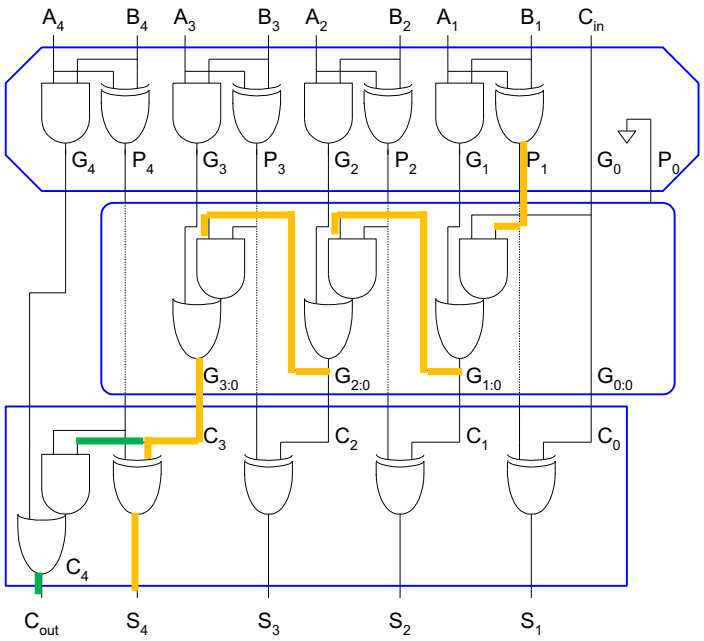
3: Sum logic

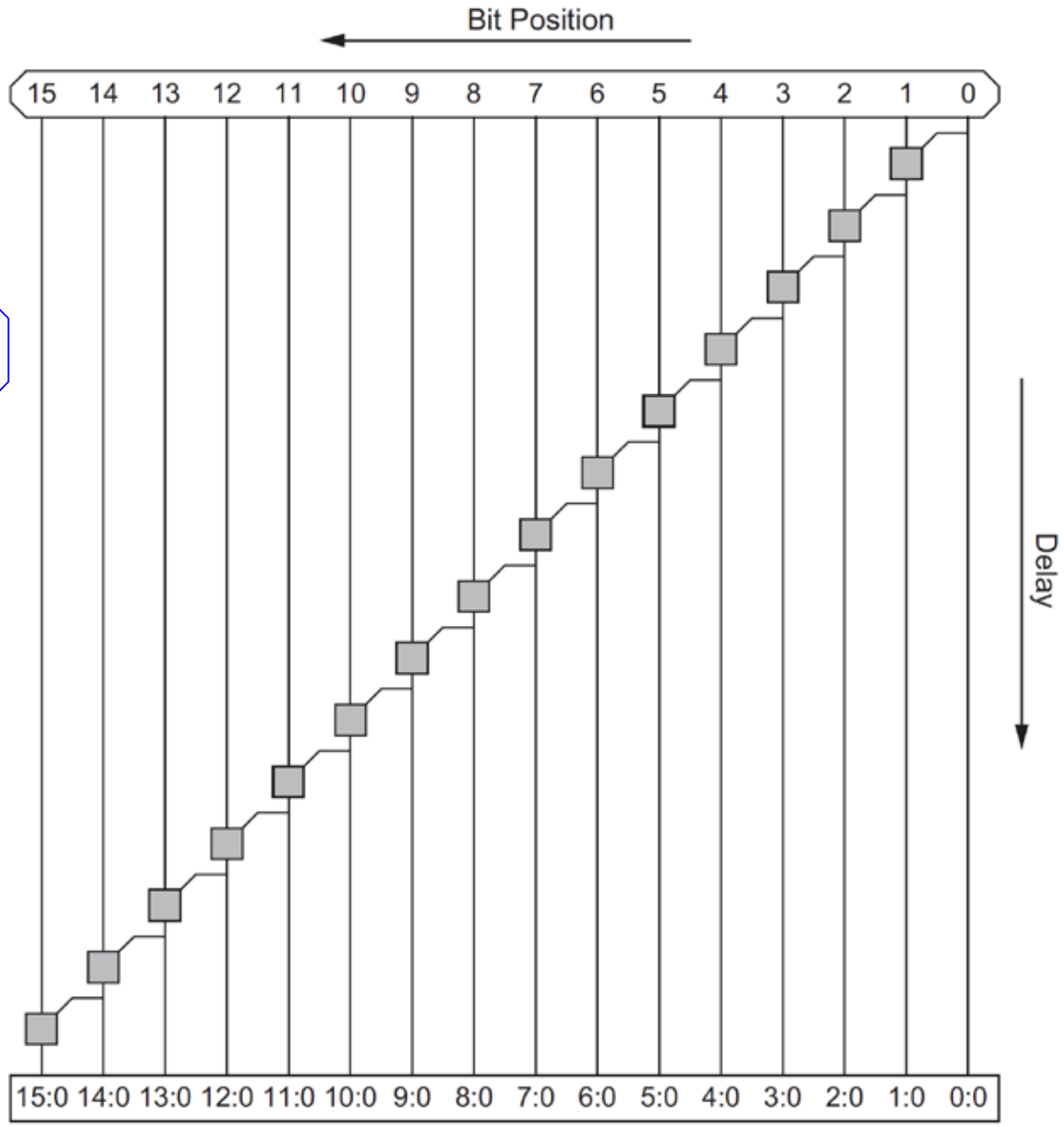
PG Diagram Notation



Ripple Carry PG Diagram

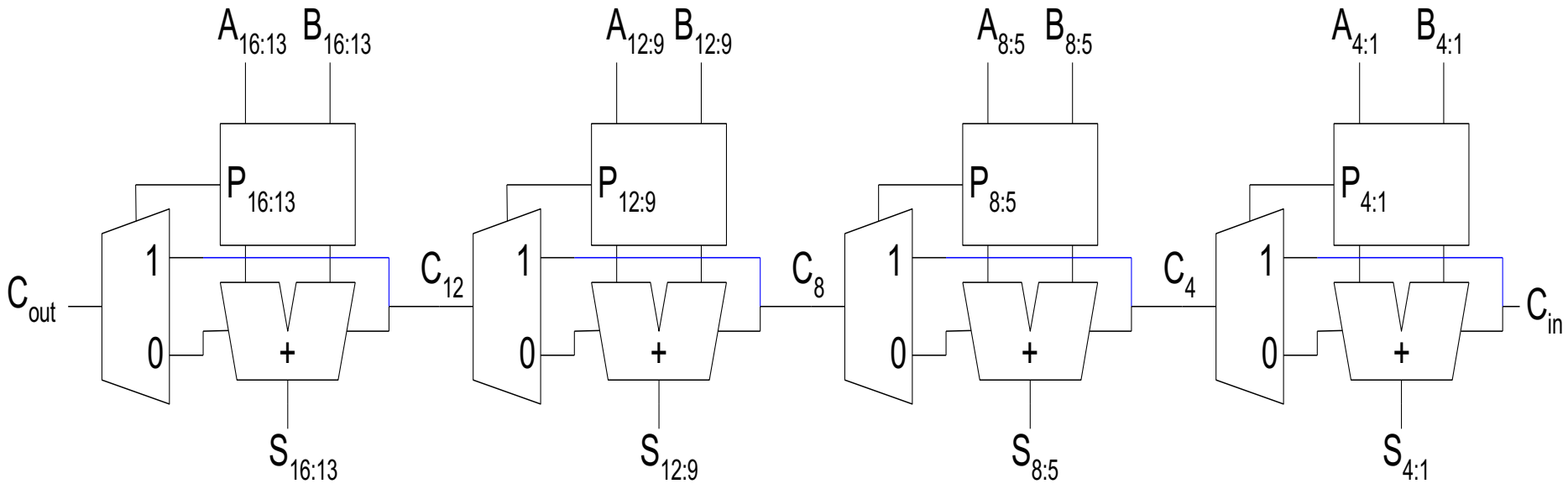
$$t_{\text{ripple}} = t_{pg} + (N-1)t_{AO} + t_{\text{xor}}$$





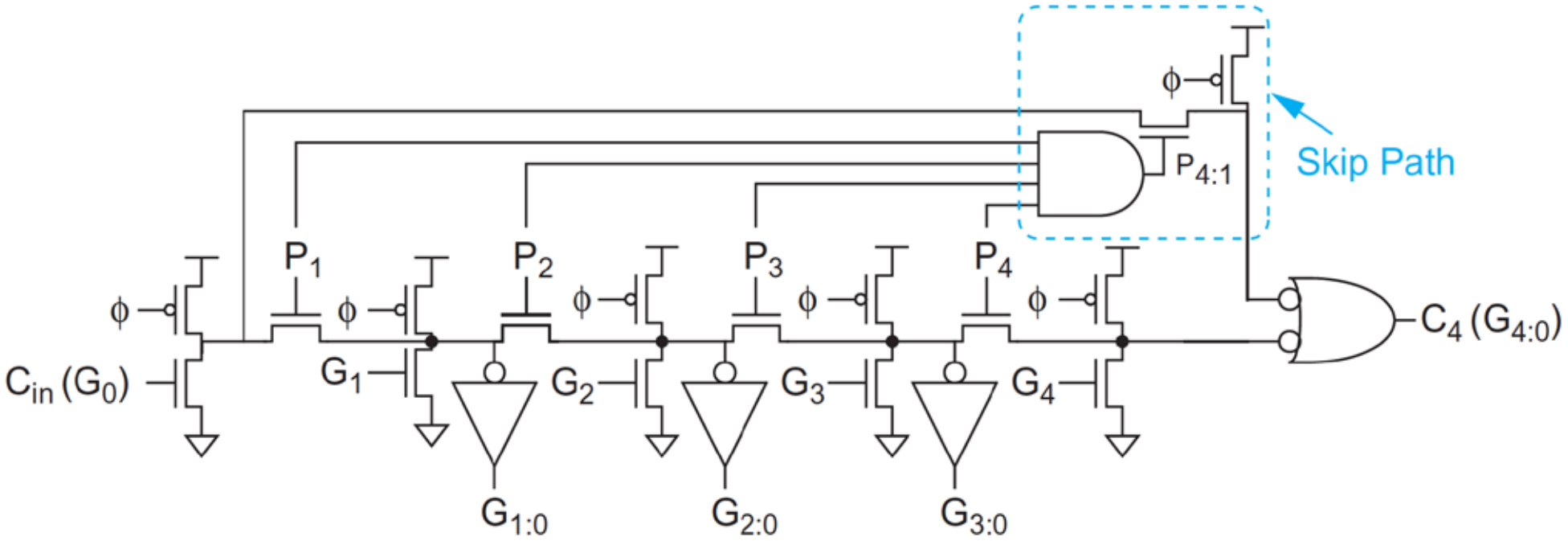
Carry-Skip Adder

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



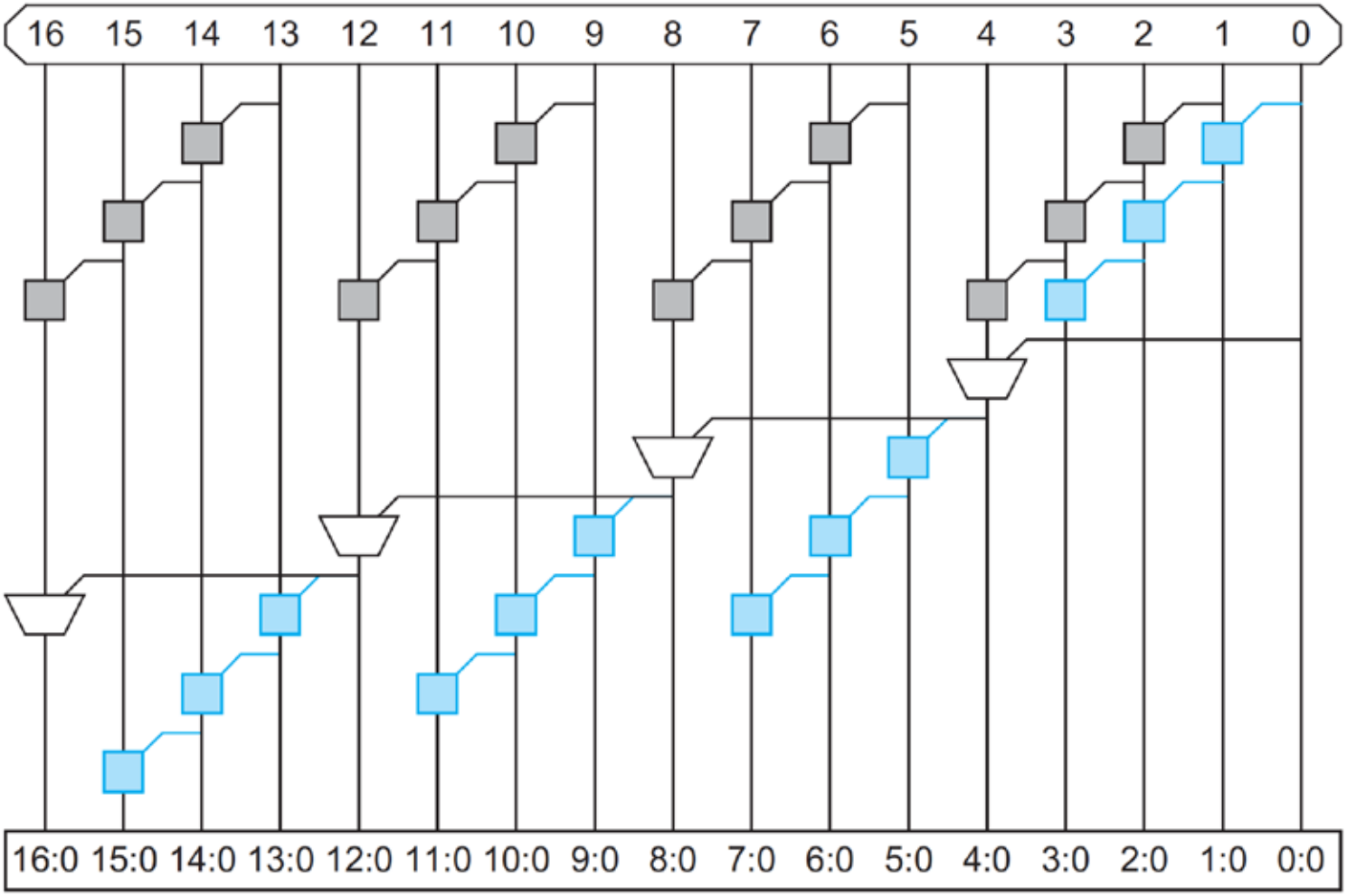
Manchester Carry-Skip Adder

- Uses dynamic logic and n-channel pass gates
- Can skip across 4 bits



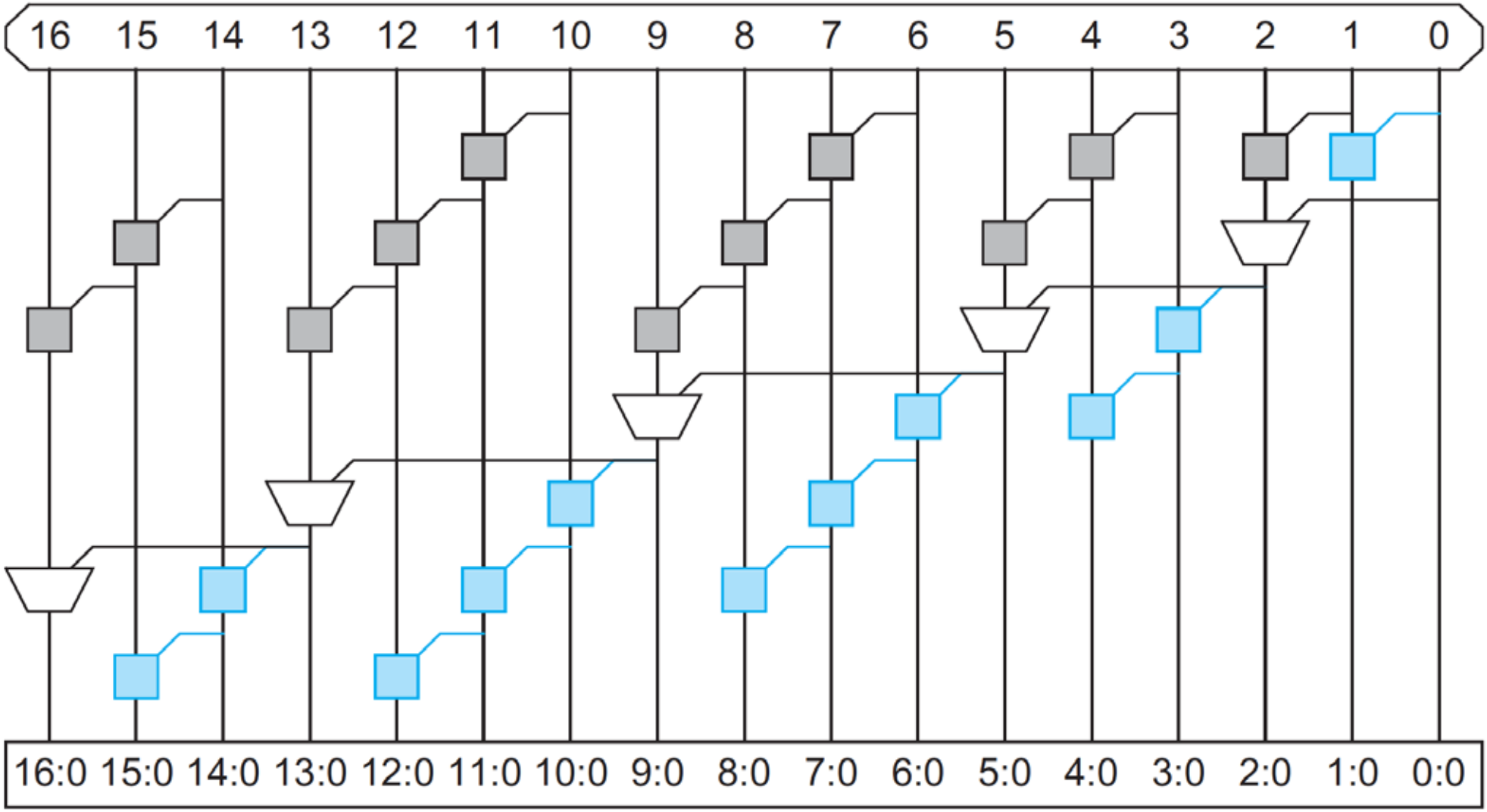
Carry-Skip PG Diagram

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



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

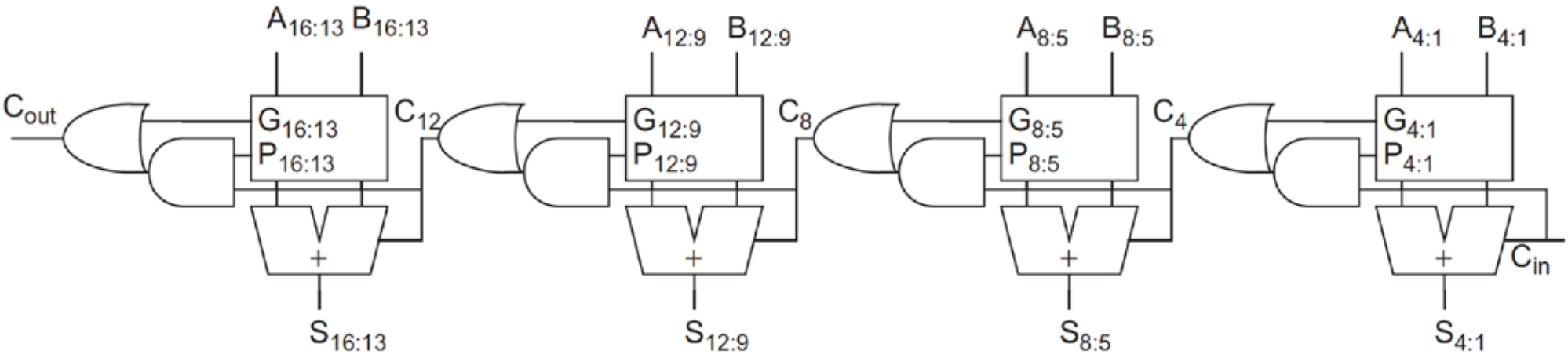
Variable Group Size



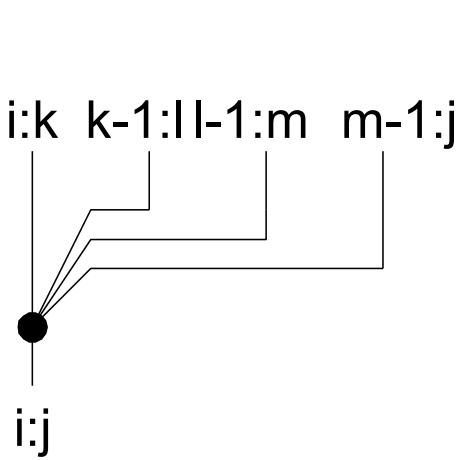
Delay grows as O(sqrt(N))

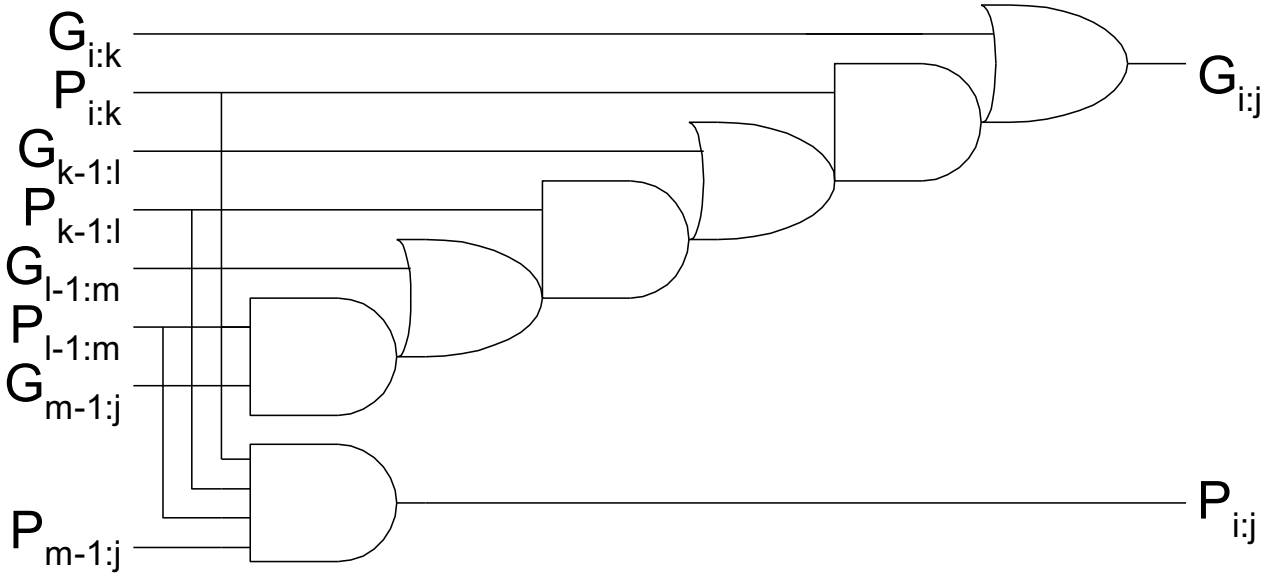
Carry-Lookahead Adder

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

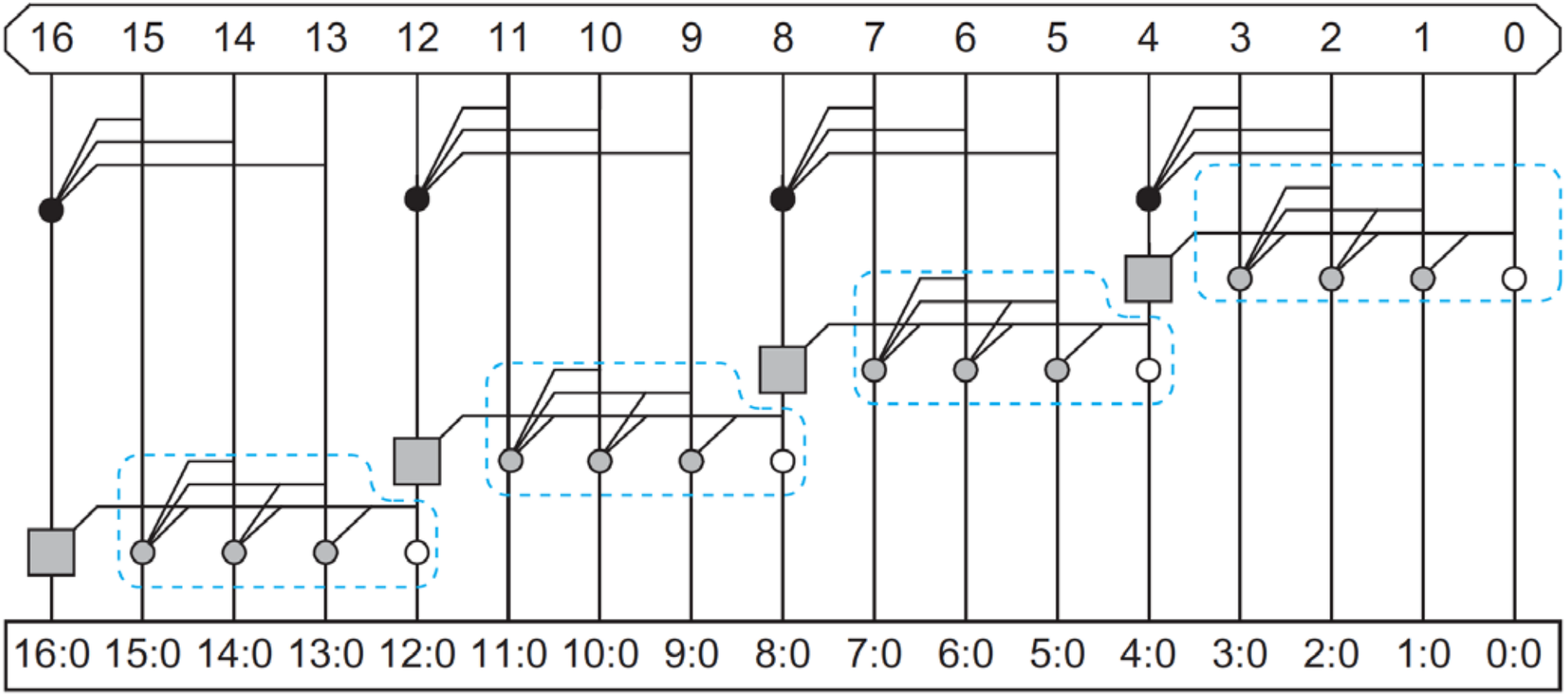


Higher-Valency Cells



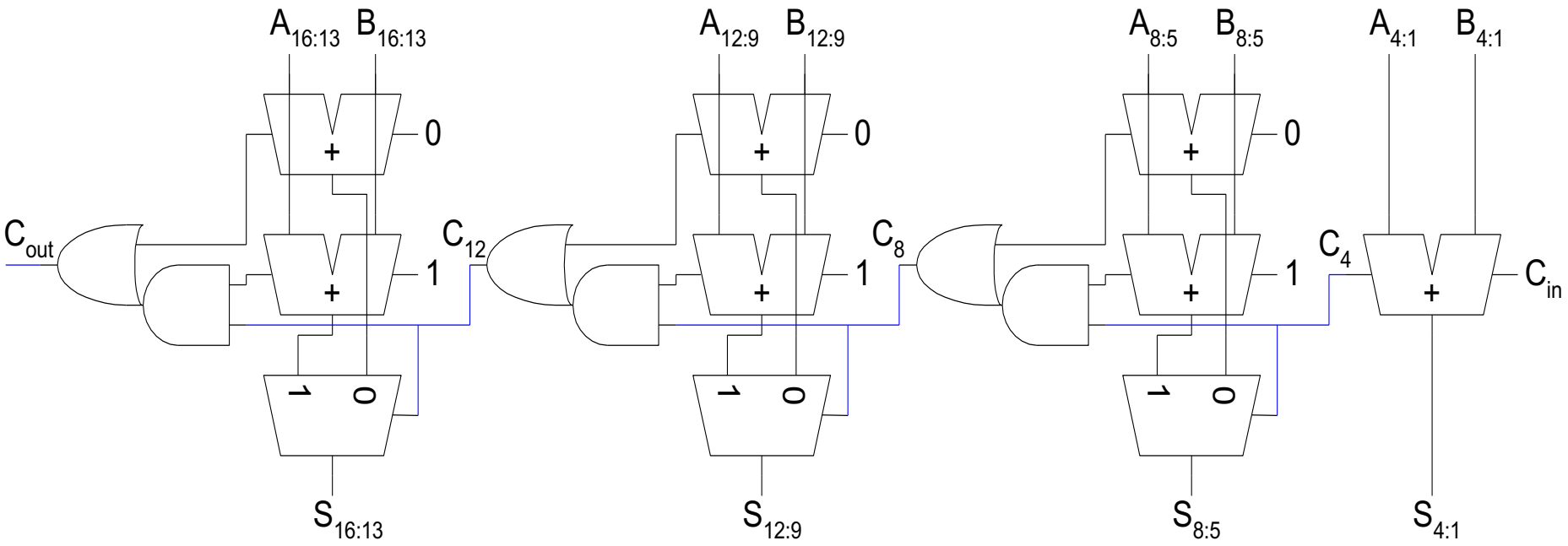


CLA PG diagram using higher valency cells



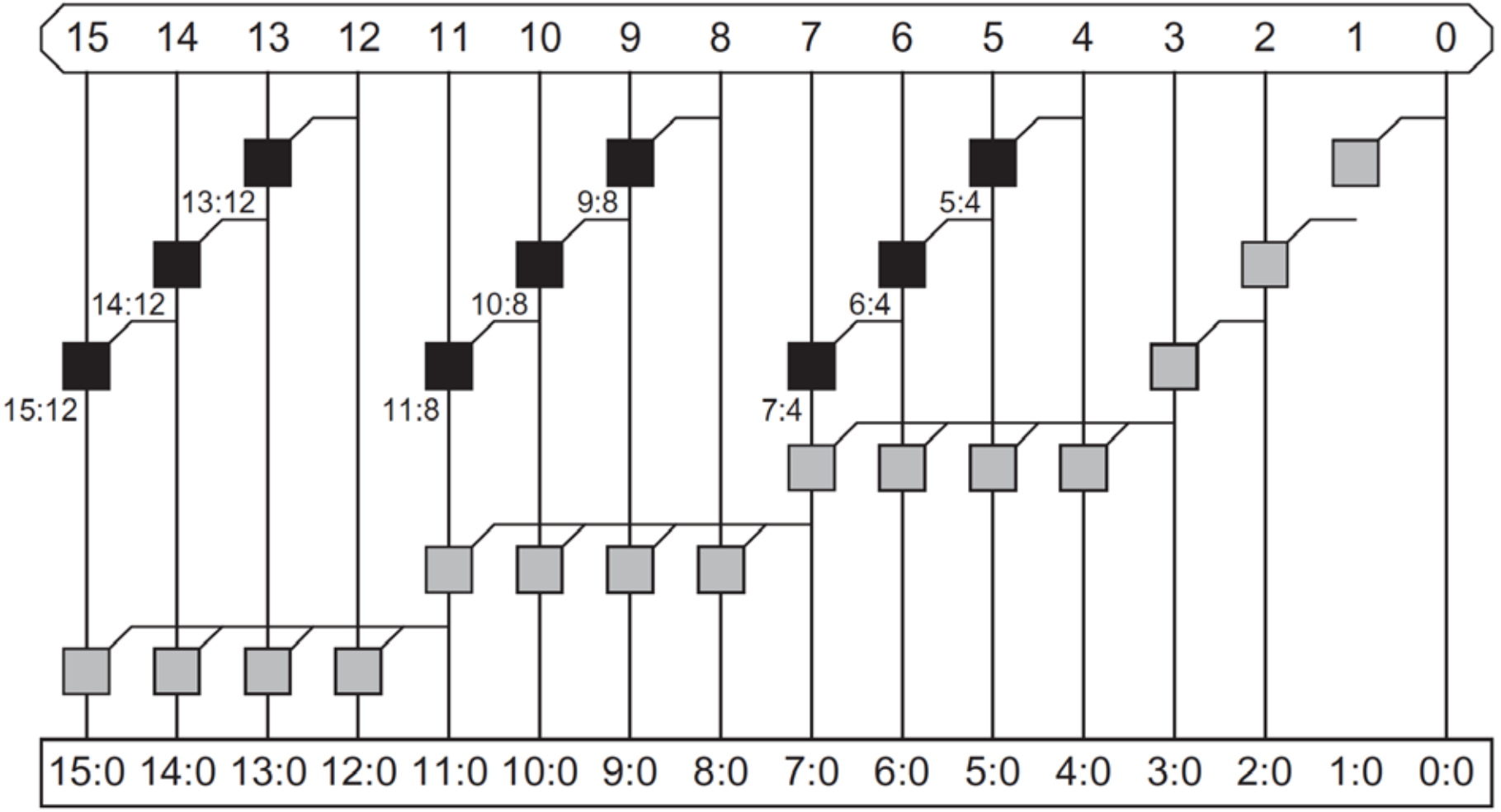
Carry-Select Adder

- **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

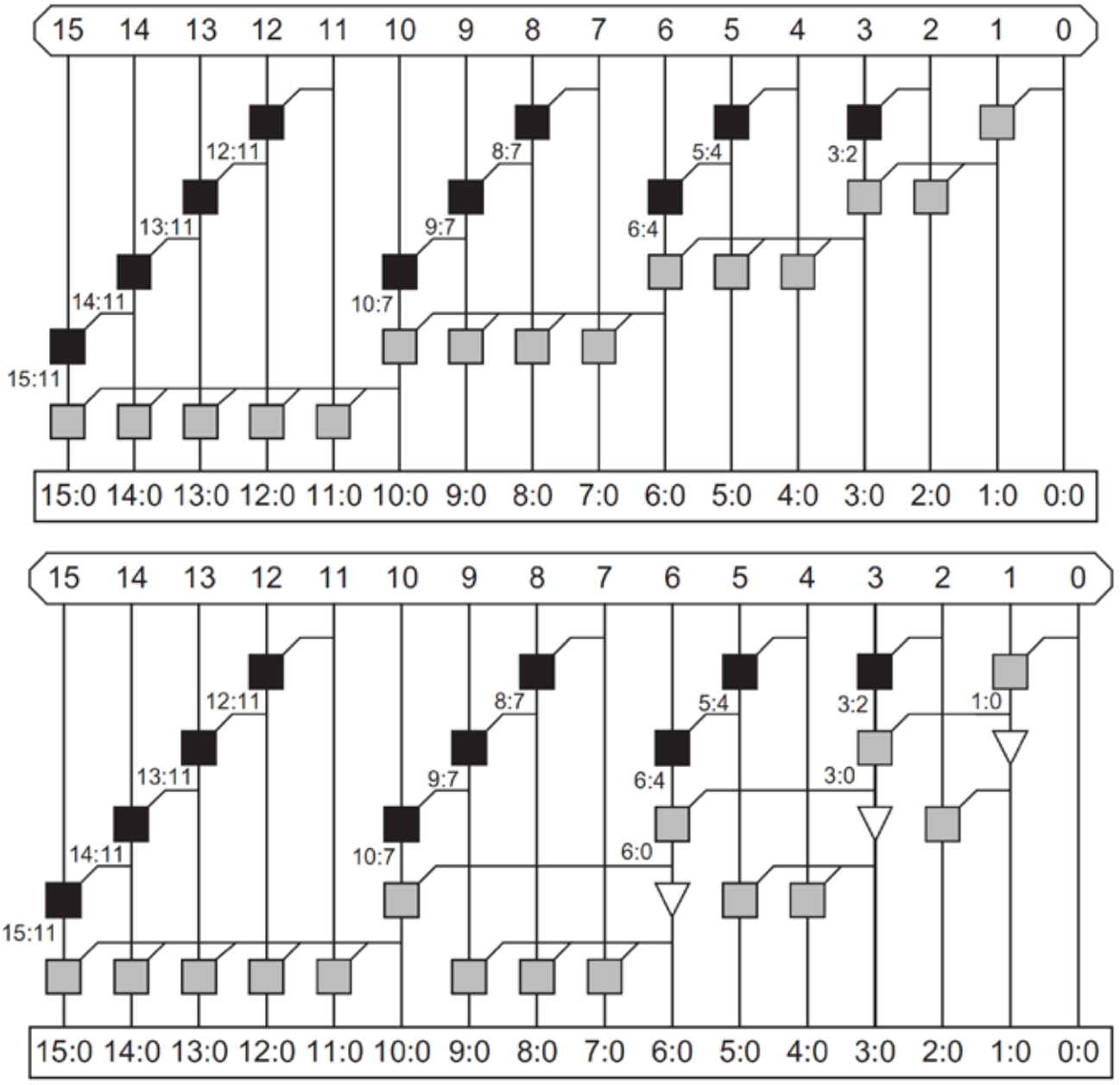
- **Factor initial PG and final XOR out of carry-select**



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

Variable Group Size

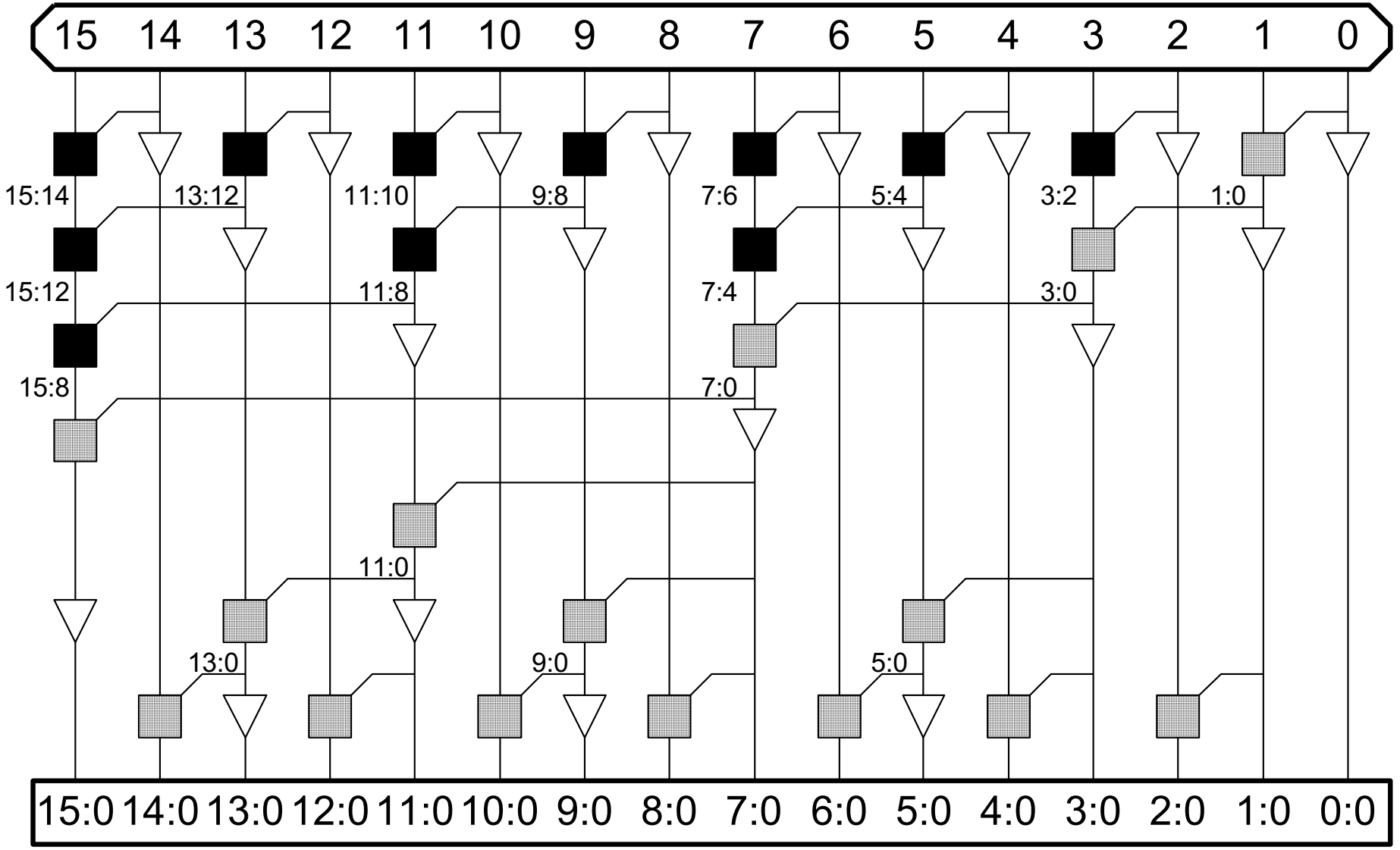
- **Buffer noncritical signals**



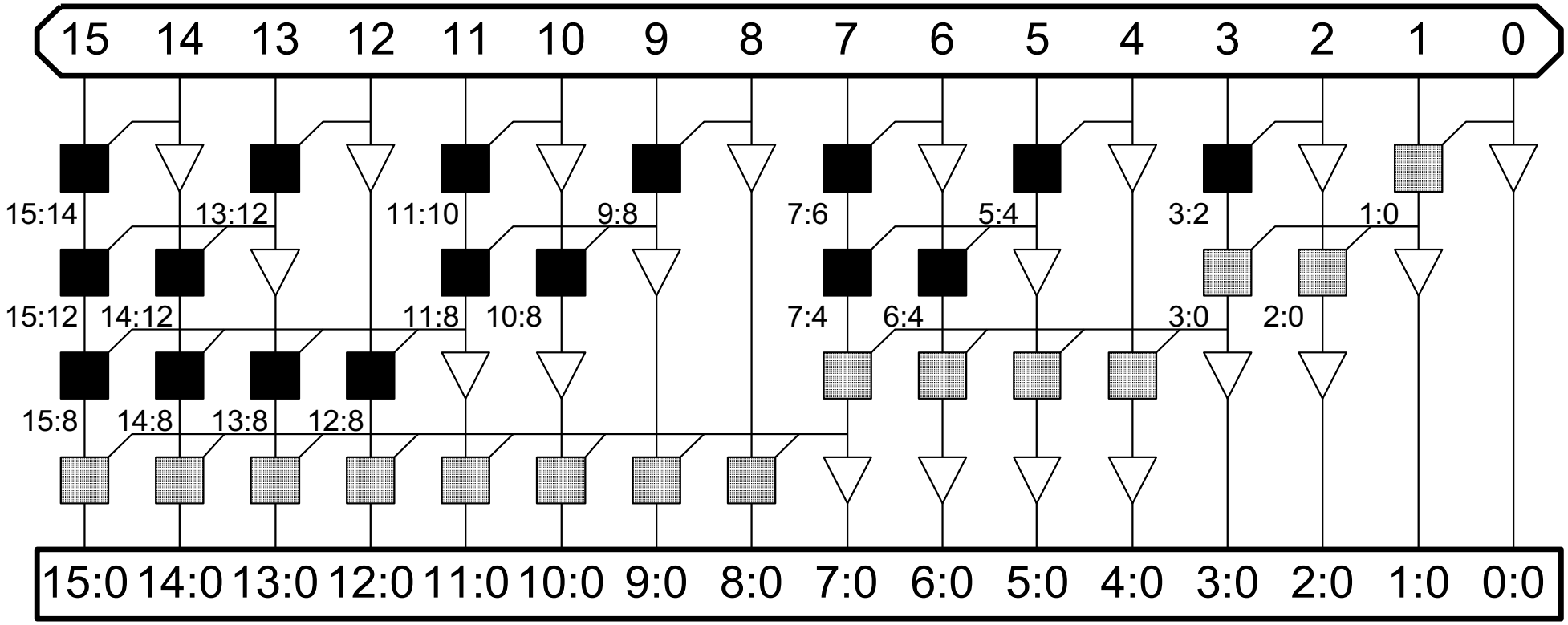
Tree Adder

- If look-ahead is good, look-ahead across lookahead!
  - Recursive lookahead gives O(log N) delay
- Many variations on tree adders
- Three metrics to worry about:
  - Wiring tracks
  - Levels of logic
  - Fanout

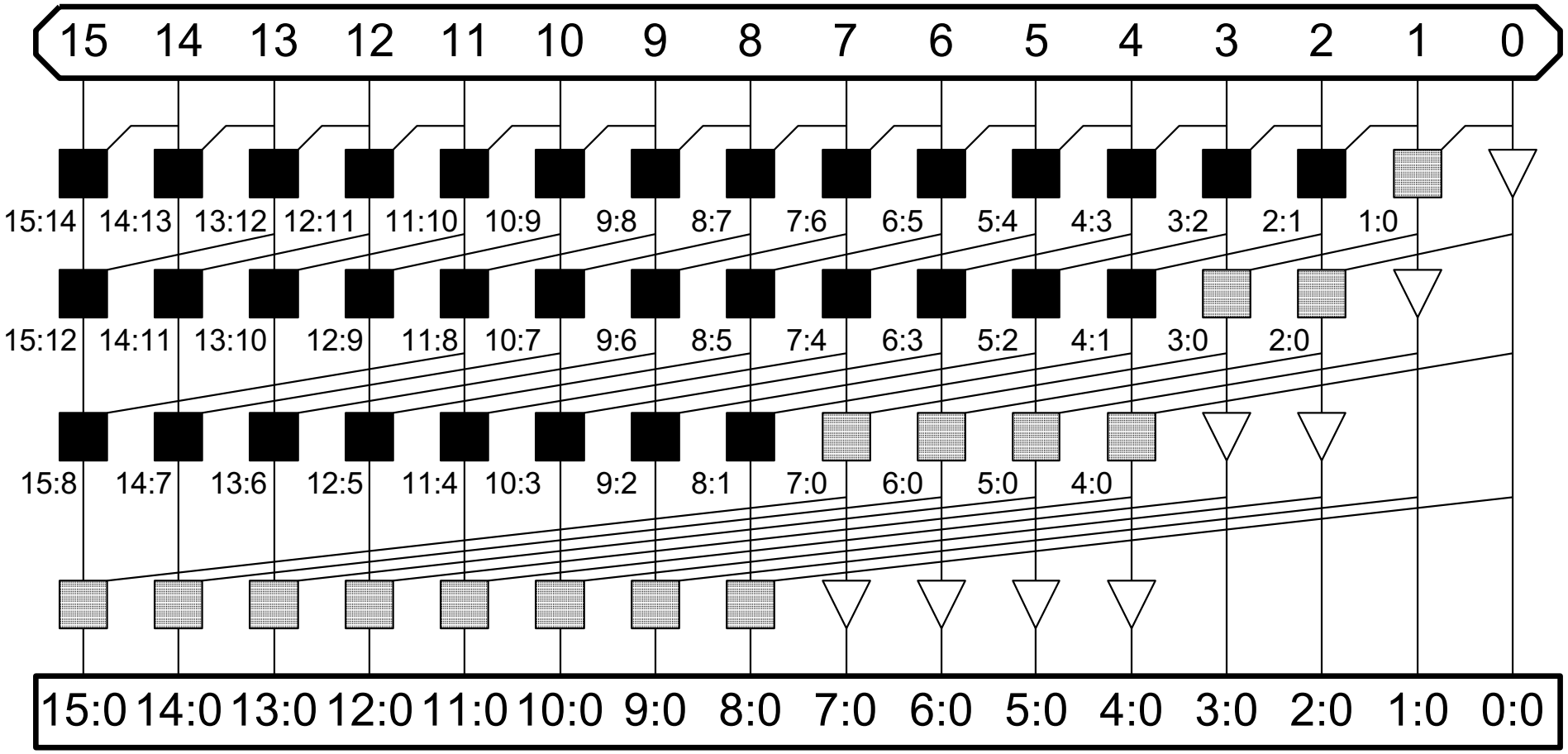
Brent-Kung


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
11:8
7:4
3:0
15:8
7:0
11:0
13:0
9:0
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

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

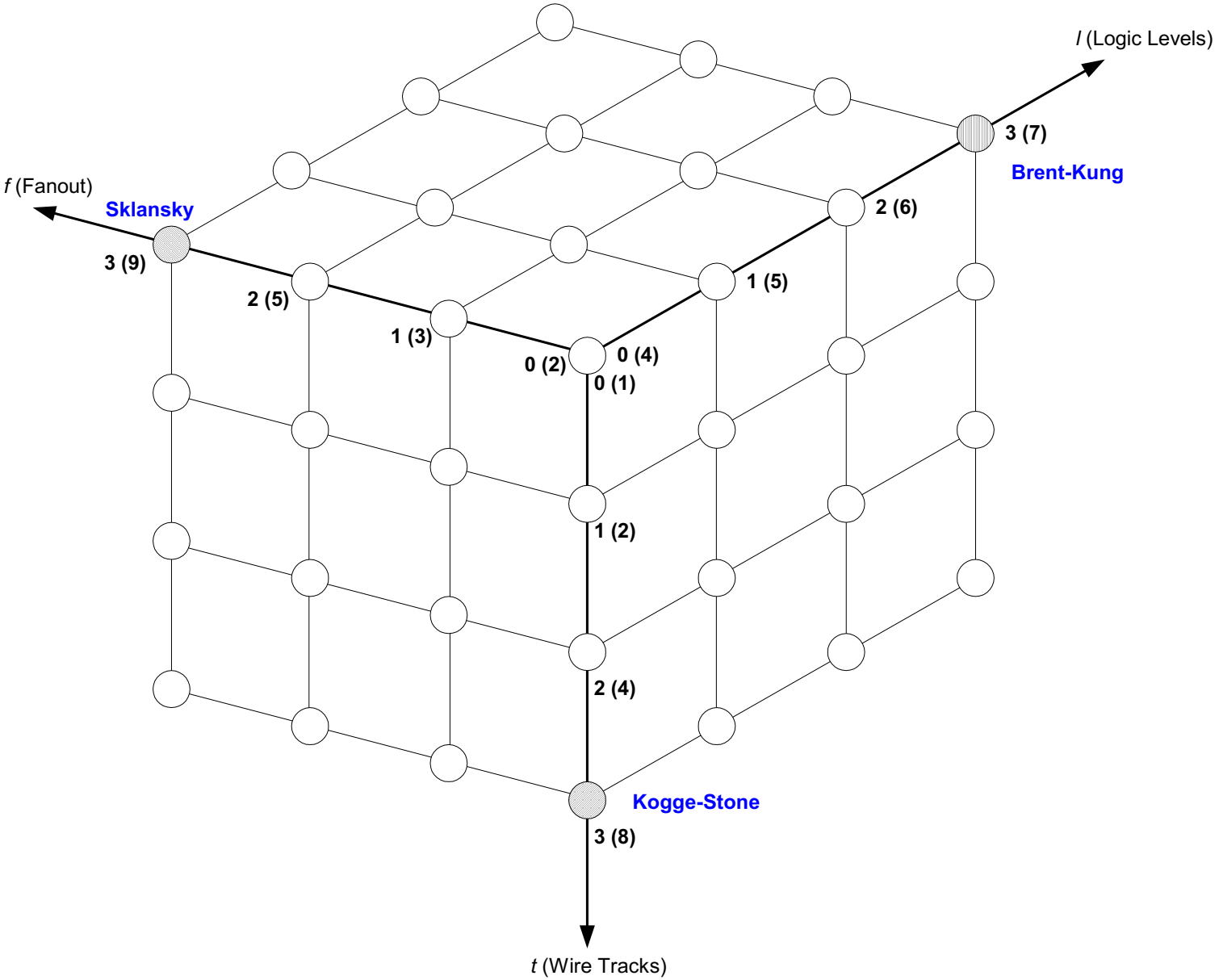
- **Ideal N-bit tree adder would have**
  - $L = \log N$ logic levels
  - Fanout never exceeds 2
  - No more than one wiring track between levels

- **We describe adders with 3-D taxonomy (l, f, t)**
  - Logic levels: $L + l$
  - Fanout: $2^f + 1$
  - Wiring tracks: 2^t

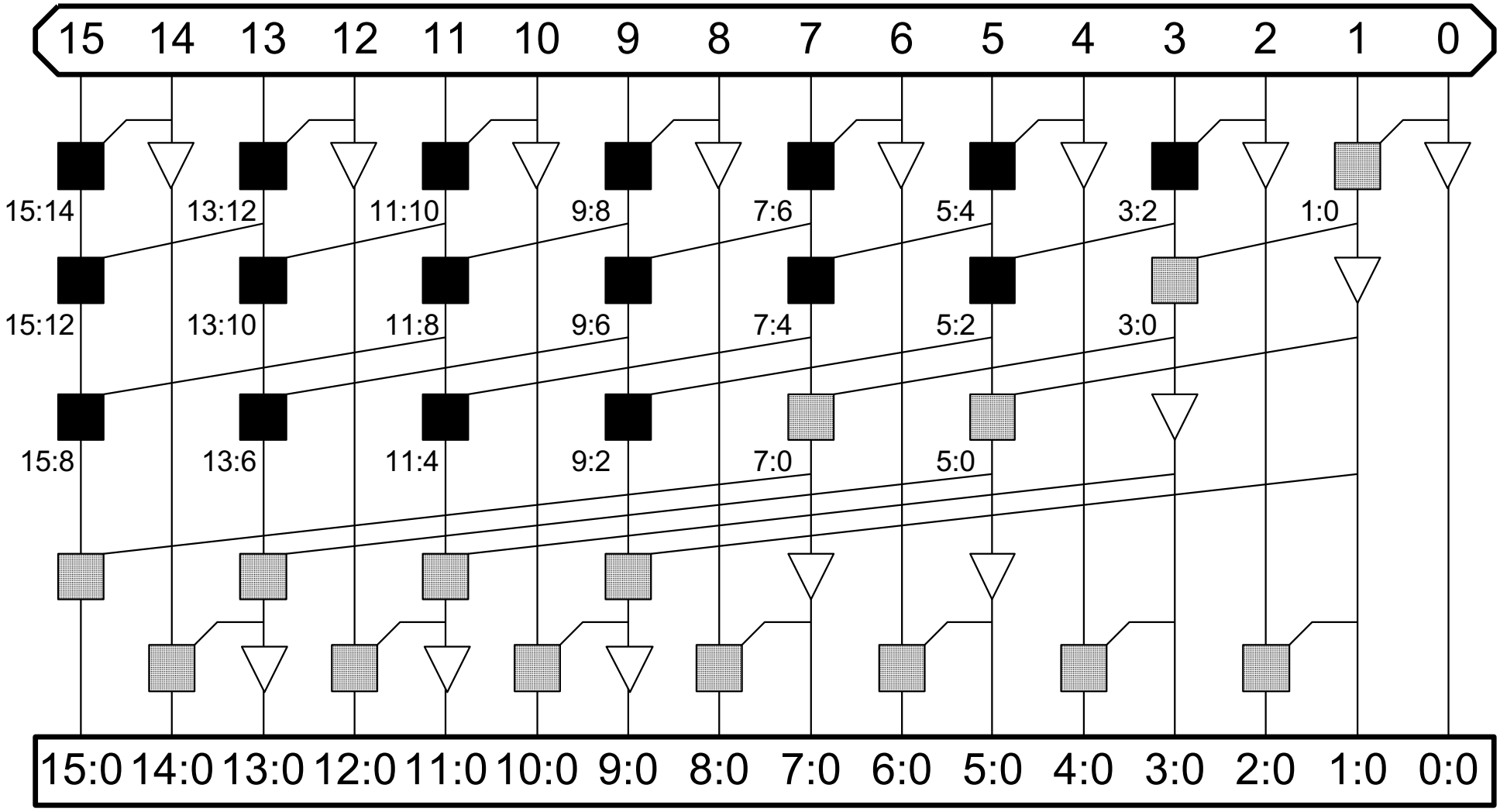
- **Known tree adders sit on plane defined by**

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

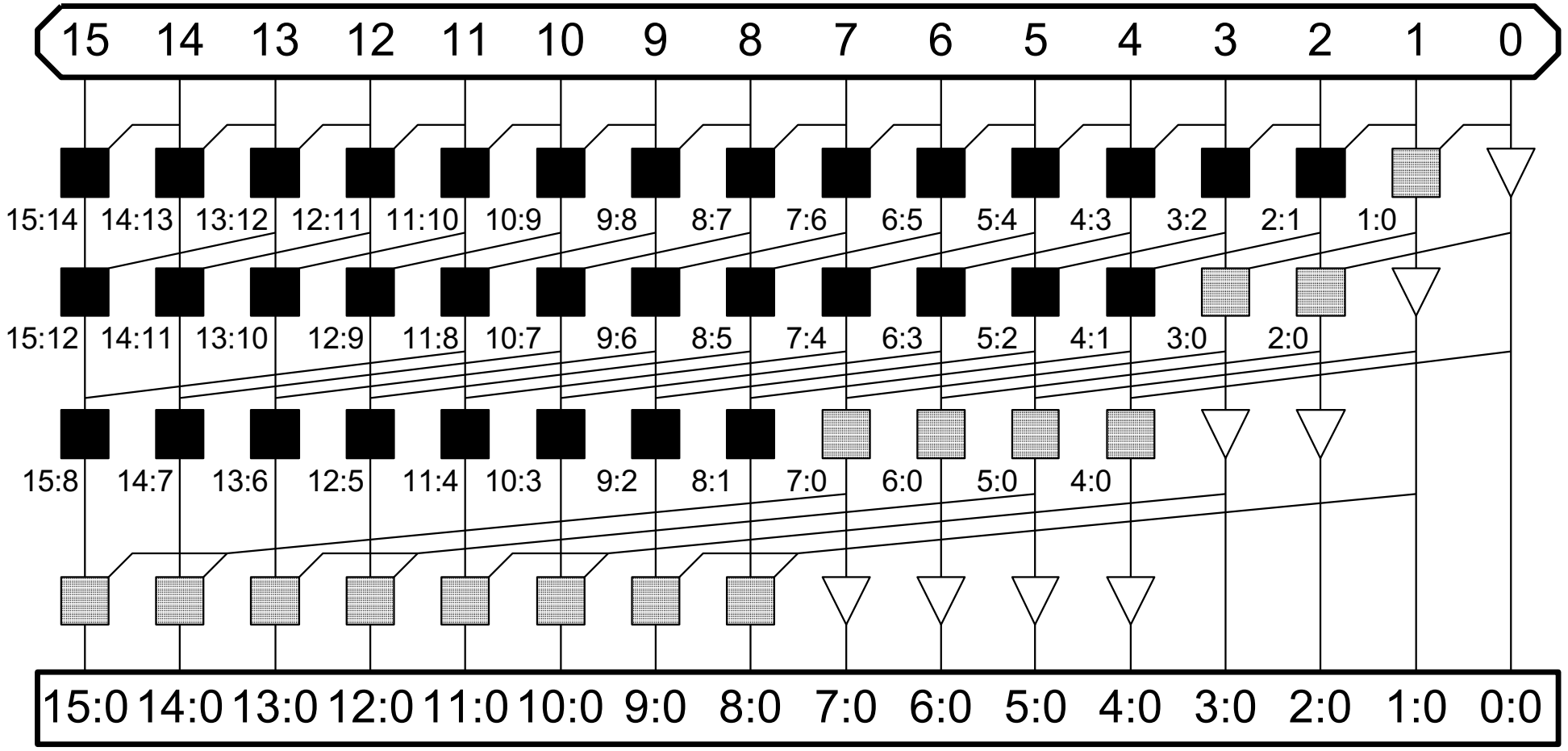
Basic Tree Adder Taxonomy



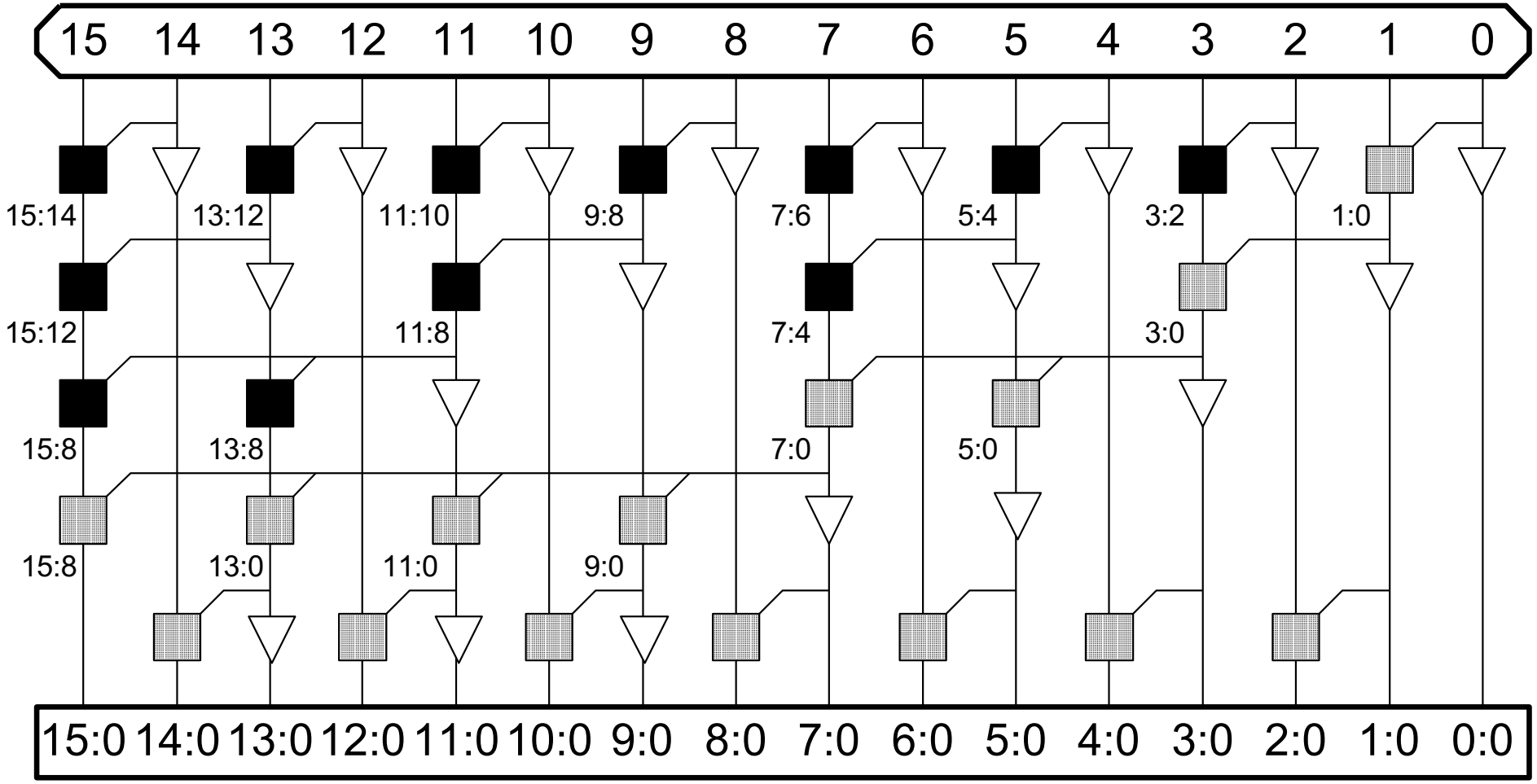
Han-Carlson


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

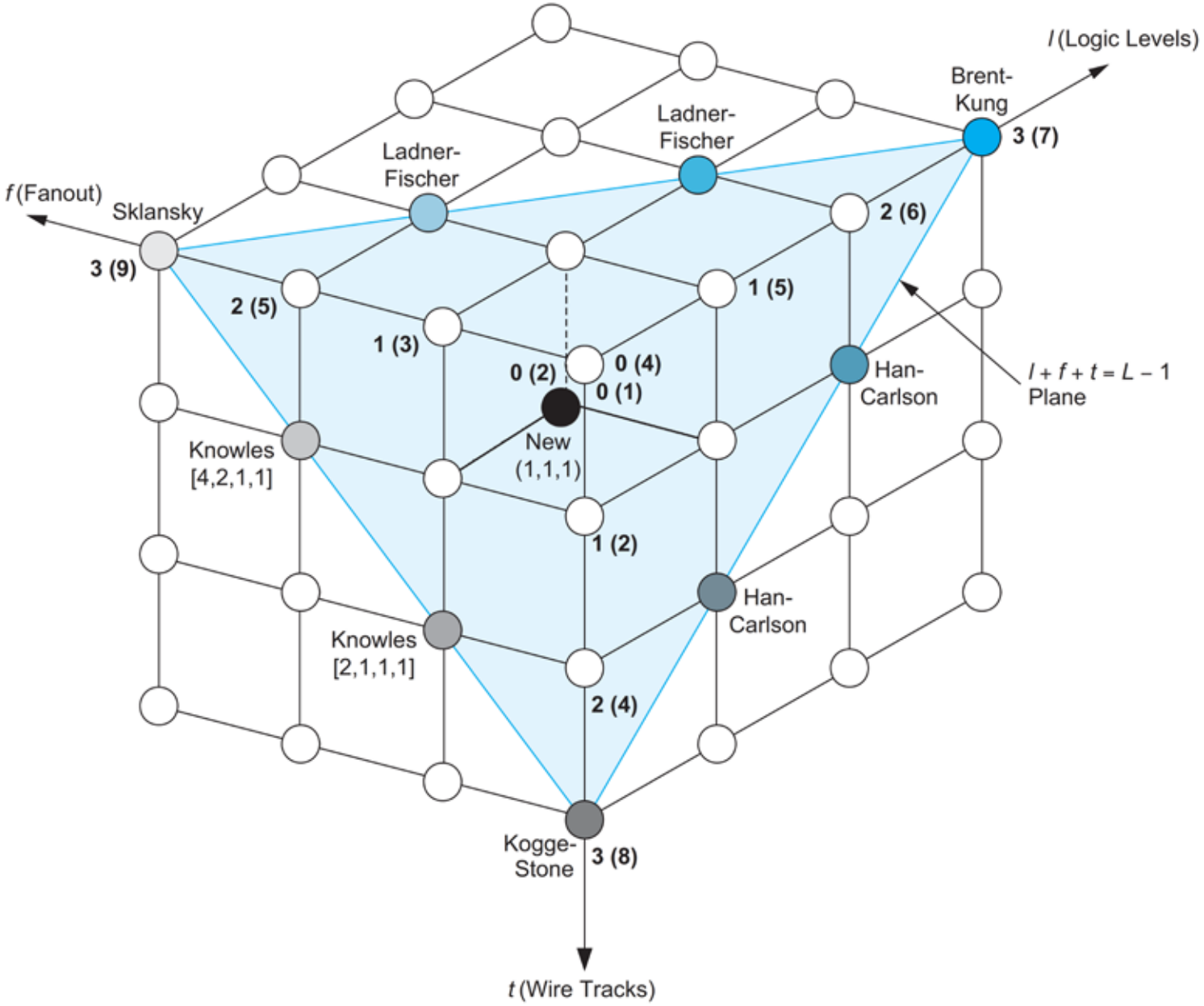
Knowles


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

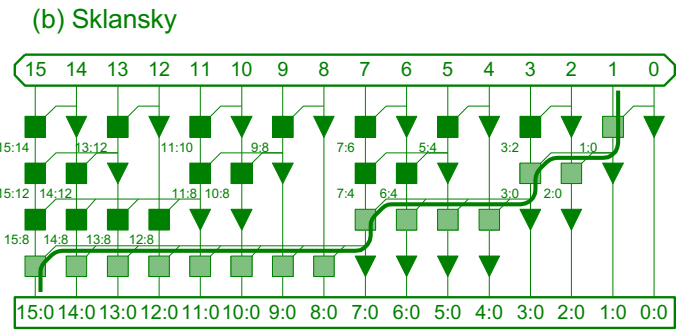
Ladner-Fischer

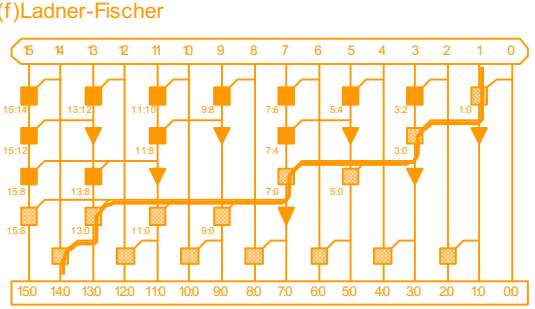


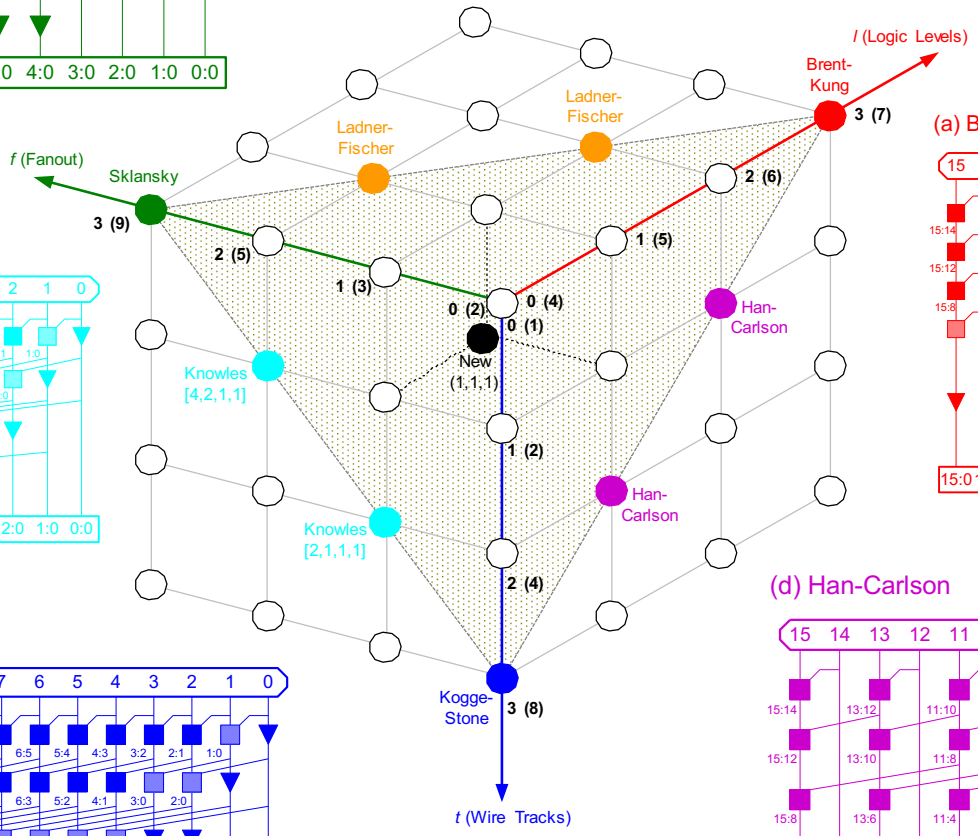
Revised Tree Adder Taxonomy

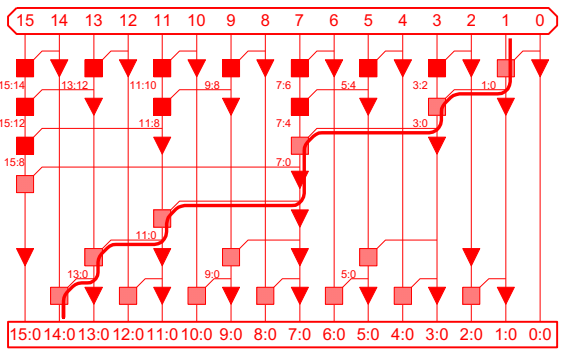


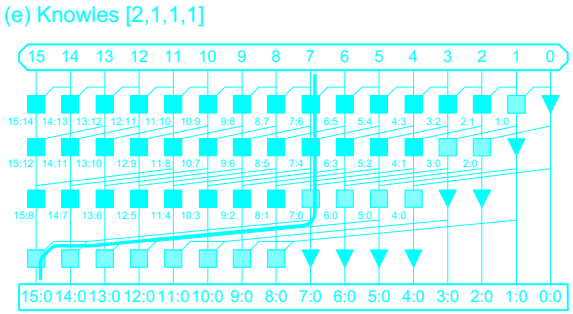
Tree Adder Taxonomy Overview

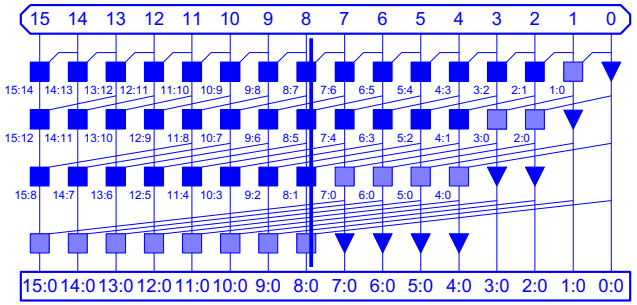


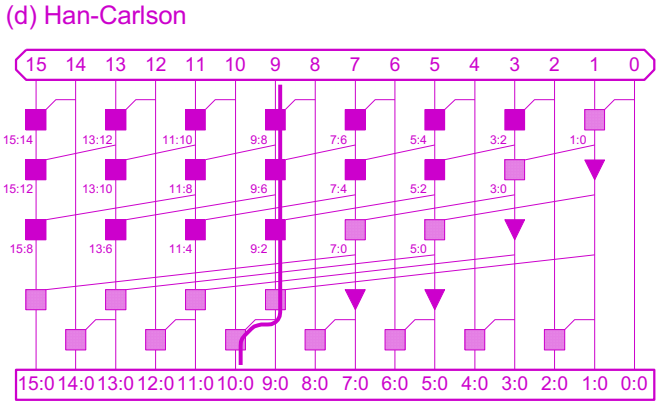




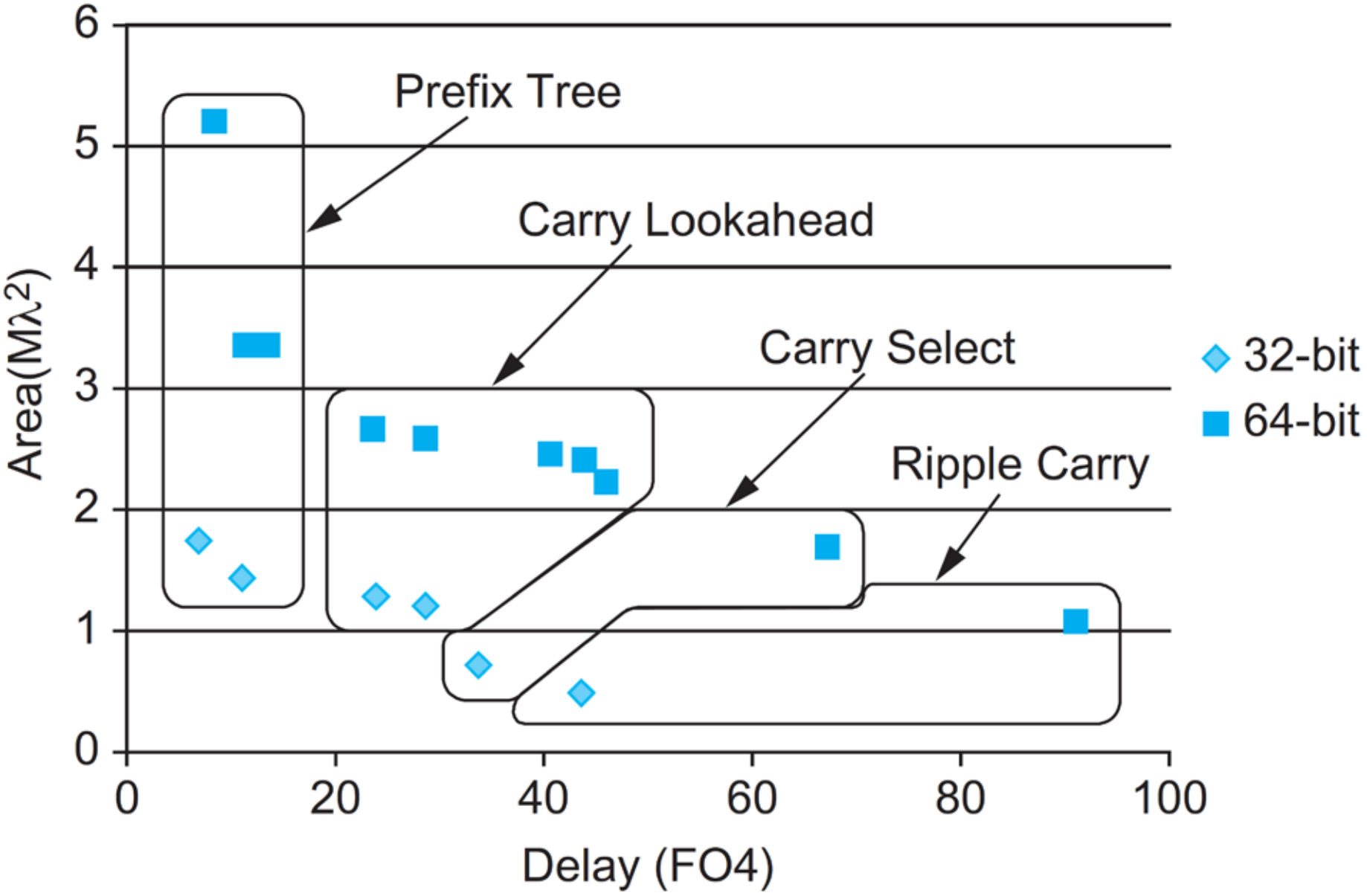








Area vs. delay of synthesized adders



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-Increment ($n=4$)		$N/4+2$	4	1	$2N$
Carry-Increment (variable group)		$\sqrt{2N}$	$\sqrt{2N}$	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	$N/2$	$N\log_2 N$
Han-Carlson	$(1, 0, L-2)$	$\log_2 N+1$	2	$N/4$	$0.5\,N\log_2 N$
Ladner Fischer ($l=1$)	$(1, L-2, 0)$	$\log_2 N+1$	$N/4+1$	1	$0.25\,N\log_2 N$
Knowles [2,1,…,1]	$(0, 1, L-2)$	$\log_2 N$	3	$N/4$	$N\log_2 N$