

Introduction to CMOS VLSI Design

Lecture 11: Adders

Outline

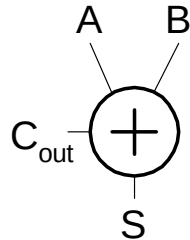
- ☐ Single-bit Addition
- ☐ Carry-Ripple Adder
- ☐ Carry-Skip Adder
- ☐ Carry-Lookahead Adder
- ☐ Carry-Select Adder
- ☐ Carry-Increment Adder
- ☐ Tree Adder

Single-Bit Addition

Half Adder

$S =$

$C_{out} =$

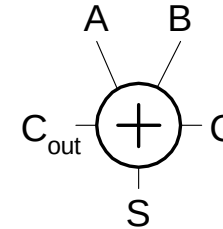


A	B	C_{out}	S
0	0		
0	1		
1	0		
1	1		

Full Adder

$S =$

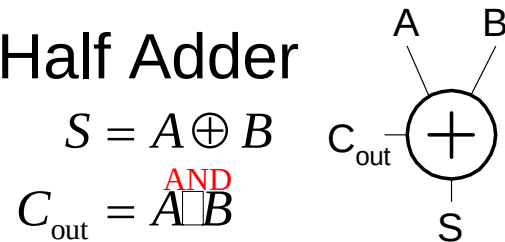
$C_{out} =$



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		

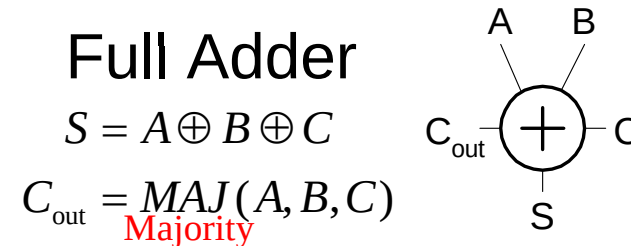
Single-Bit Addition

Half Adder



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

Full Adder



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)

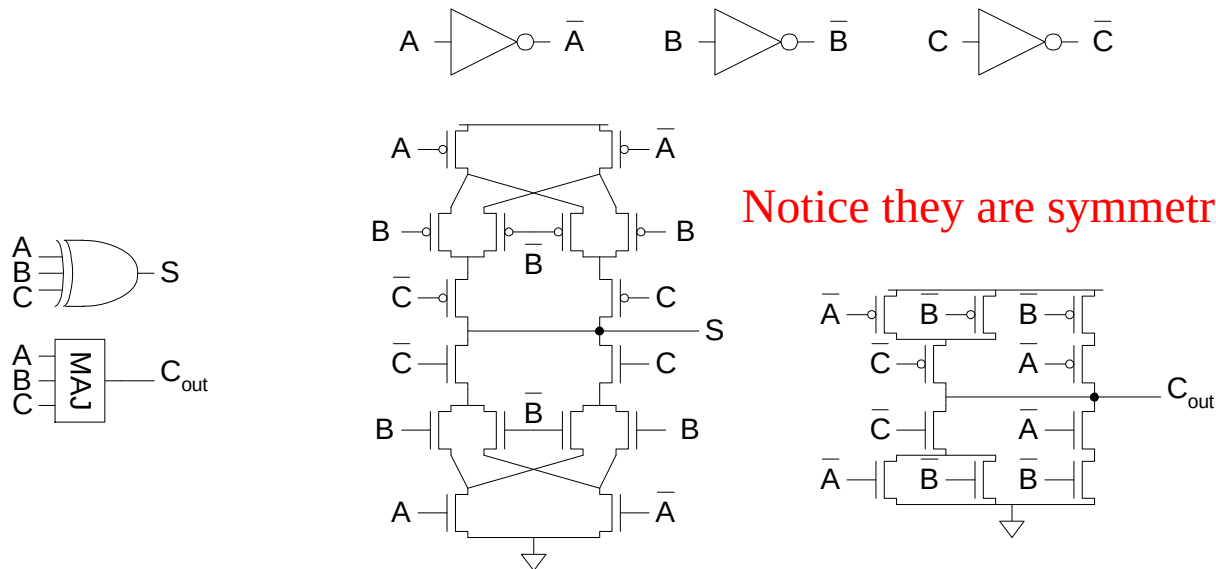
- For a full adder, define what happens to carries
 - Generate: $C_{out} = 1$ independent of C ($C = C_{in}$)
 - $G = A \cdot B$
 - Propagate: $C_{out} = C$
 - $P = A \oplus B$
 - Kill: $C_{out} = 0$ independent of C
 - $K = \sim A \cdot \sim B$

Full Adder Design I

- ❑ Brute force implementation from eqns

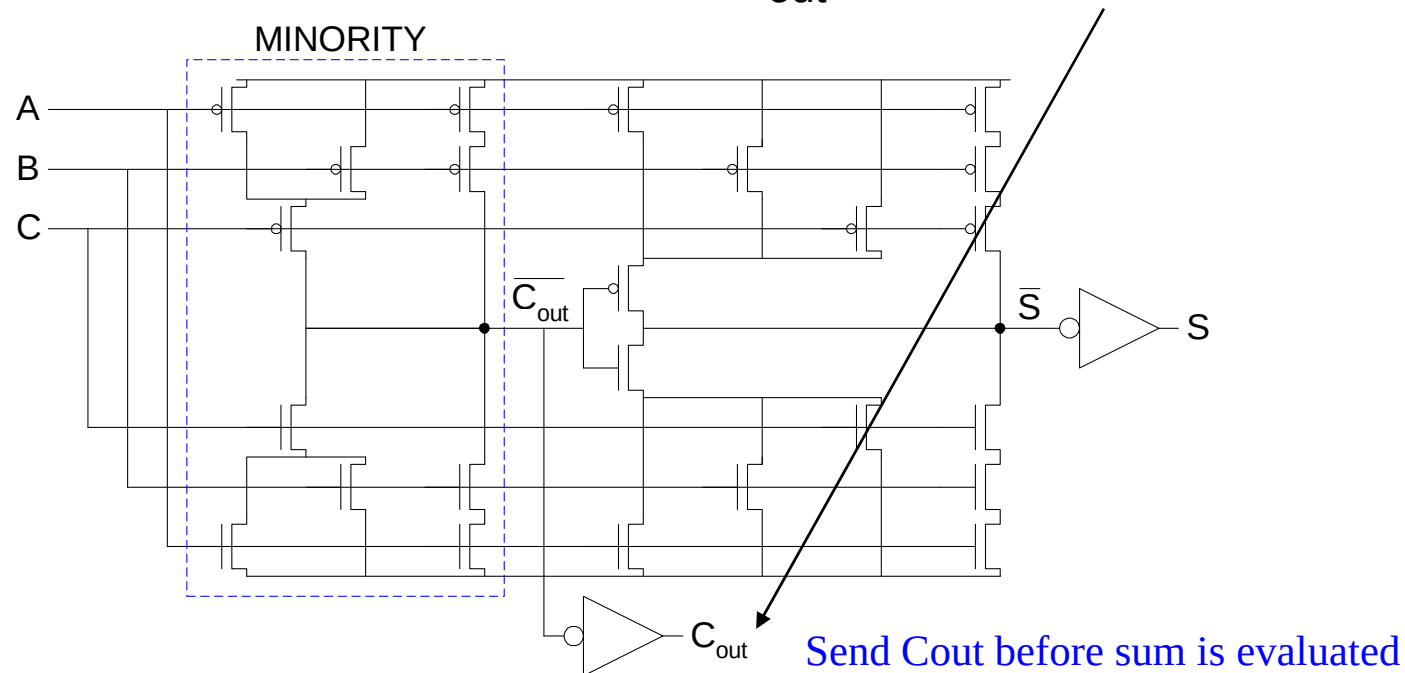
$$S = A \oplus B \oplus C = P \text{ exor } C$$

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



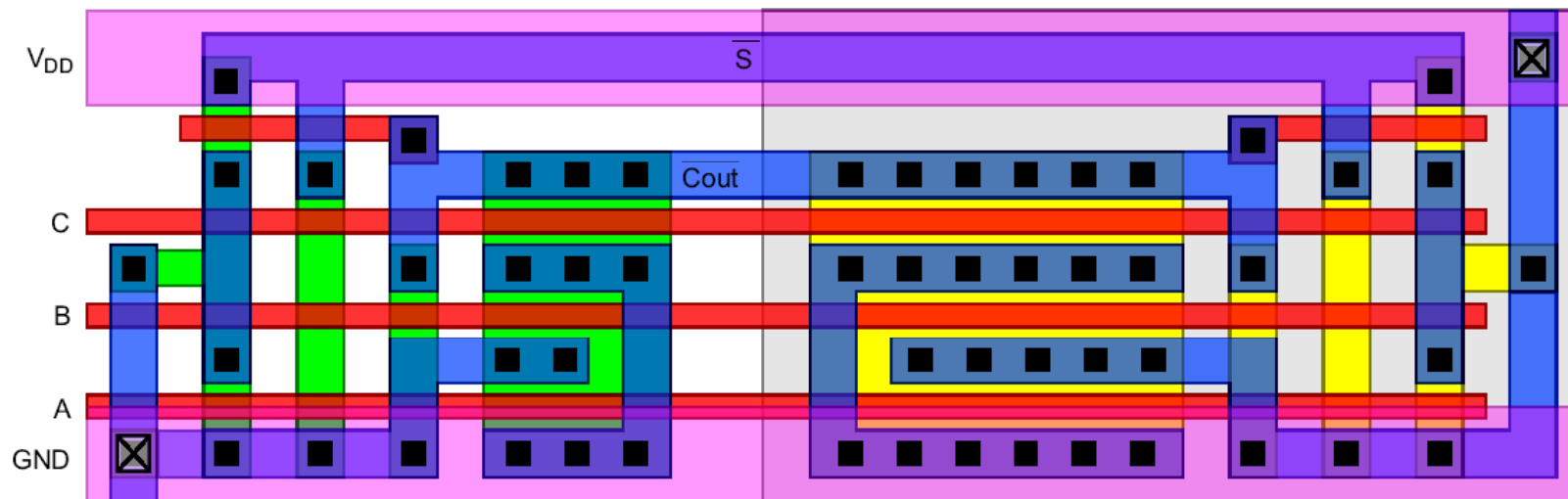
Full Adder Design II

- ❑ 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



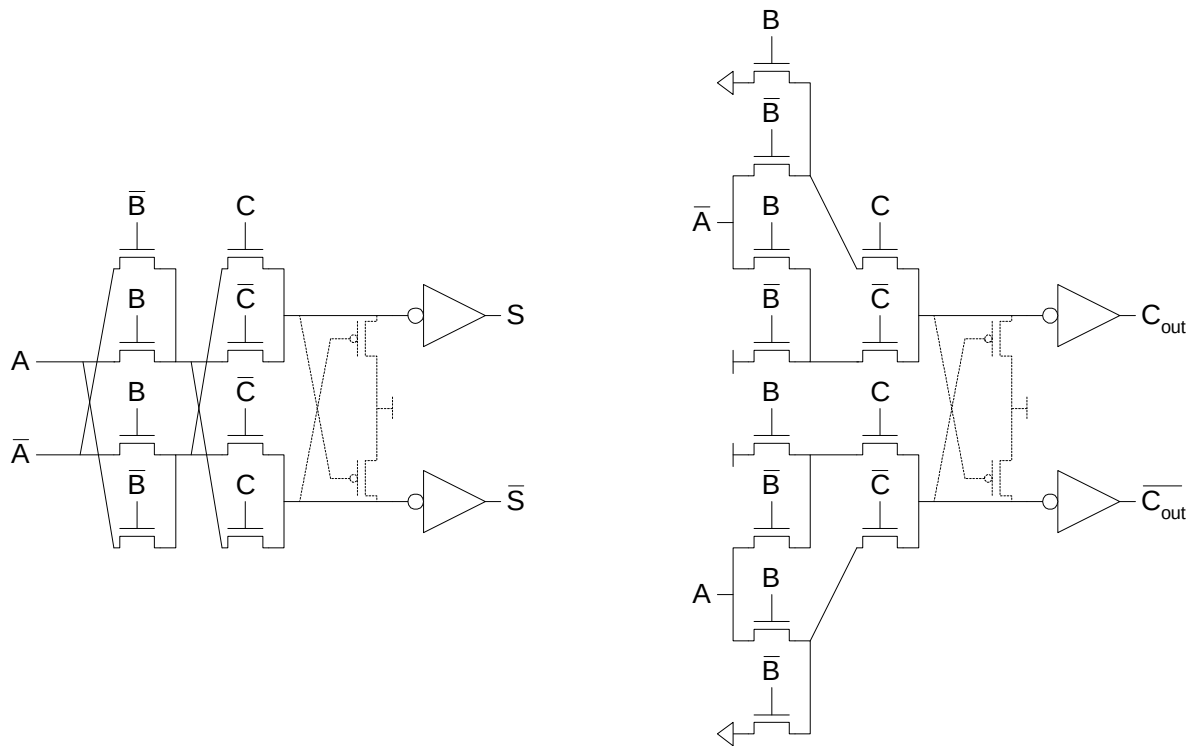
Layout

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



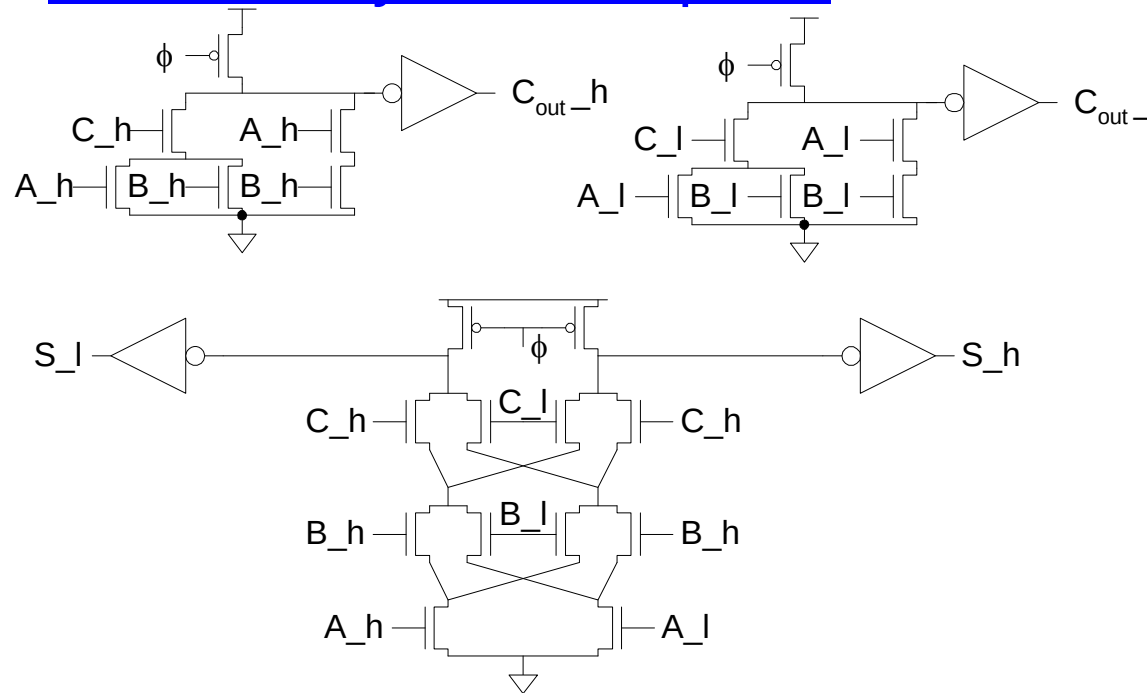
Full Adder Design III

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



Full Adder Design IV

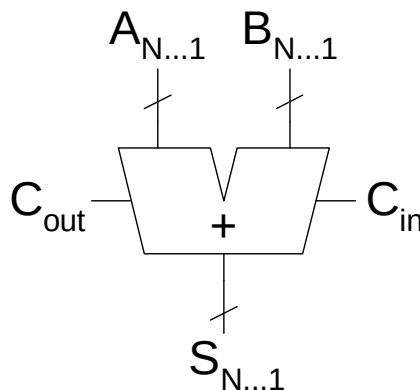
- ❑ Dual-rail domino
 - Very fast, but large and power hungry
 - Used in very fast multipliers



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 C_{in}
Conventional method



Case I

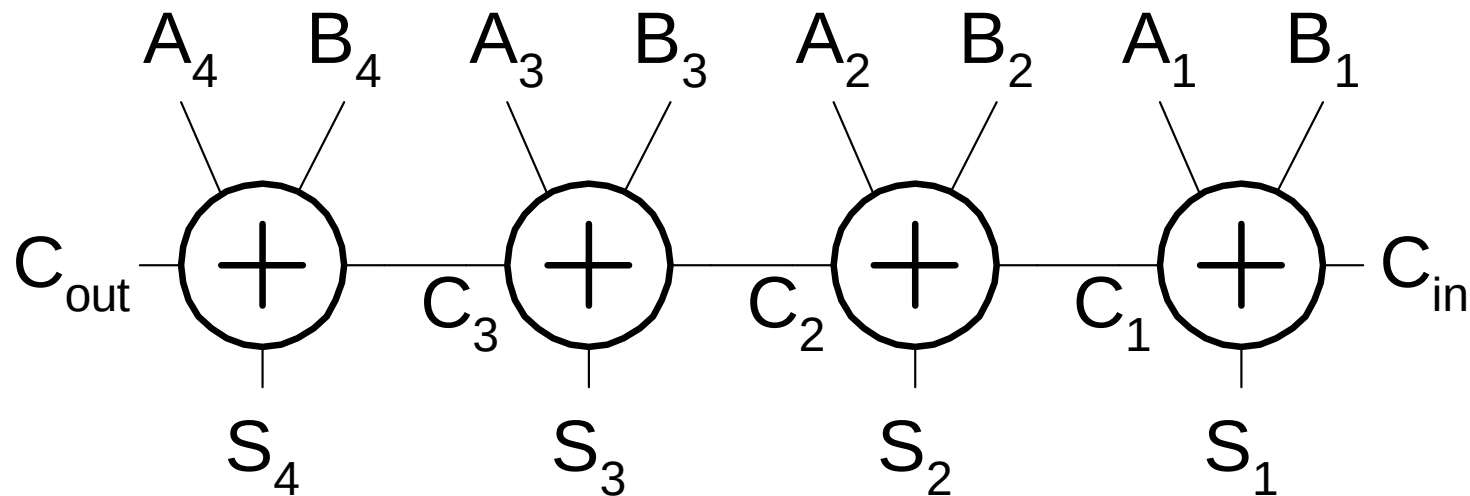
$$\begin{array}{r} \text{C}_{out} \swarrow \quad \nwarrow \text{C}_{in} \\ \textcircled{0}0000\textcircled{0} \\ 1111 \\ +0000 \\ \hline 1111 \end{array}$$

Case II

$$\begin{array}{r} \text{C}_{out} \swarrow \quad \nwarrow \text{C}_{in} \\ \textcircled{1}1111\textcircled{1} \\ 1111 \\ +0000 \\ \hline 0000 \end{array} \begin{array}{l} \text{carries} \\ A_{4...1} \\ B_{4...1} \\ S_{4...1} \end{array}$$

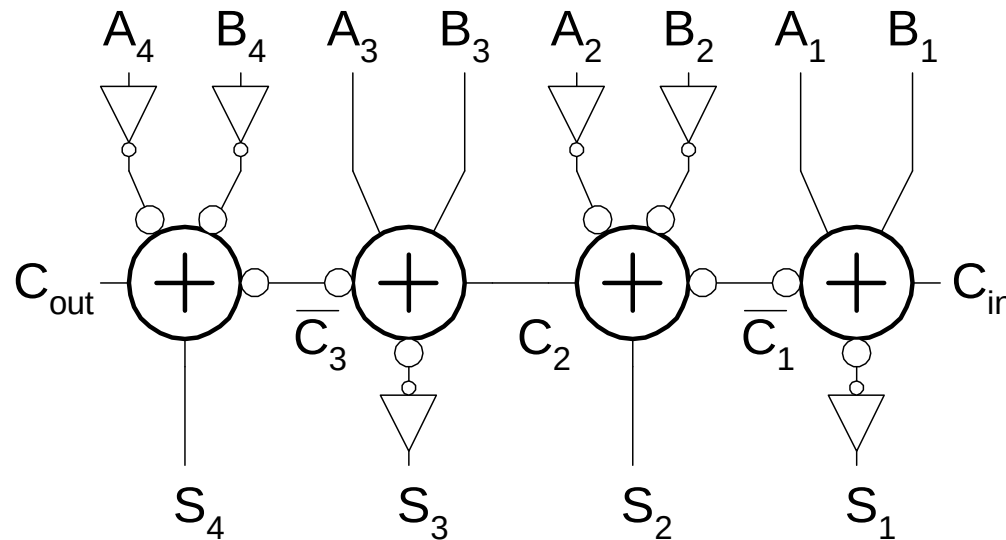
Carry-Ripple Adder

- ❑ Simplest design: cascade full adders
 - Critical path goes from C_{in} to C_{out}
 - 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



Using inverter, majority calculation becomes minority calculation.

Eliminate inverter between stages (See slide 8)

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

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

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

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

k can be any stage between i and k

- Base case

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

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

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

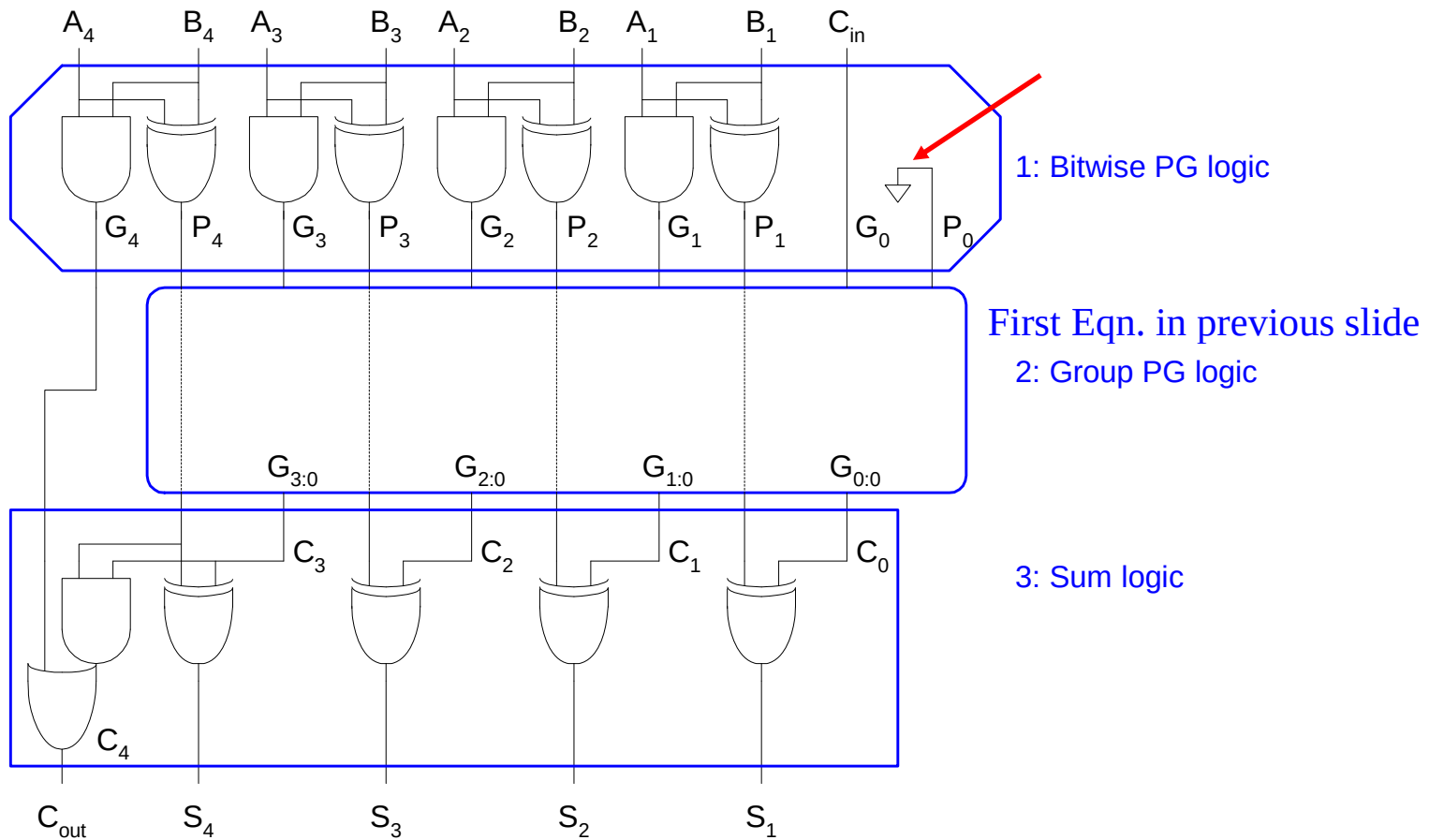
$$G_{0:0} \equiv G_0 = C_{in} \quad G_{i-1:0} = C_{i-1}$$

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

- Sum:

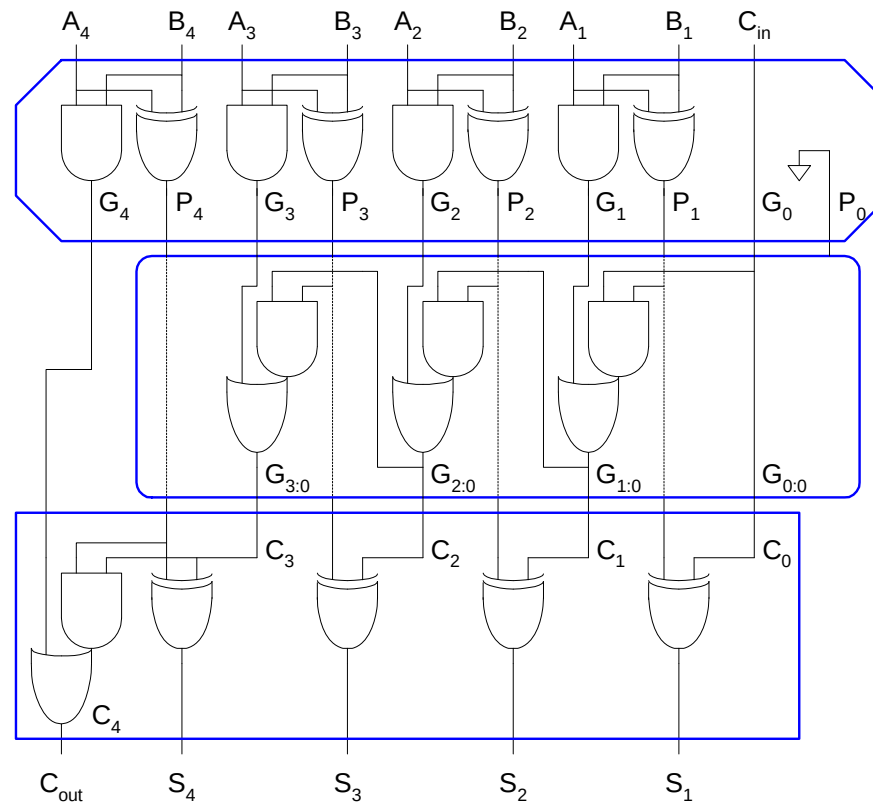
$$S_i = P_i \oplus G_{i-1:0} \text{ (As slide 7)}$$

PG Logic

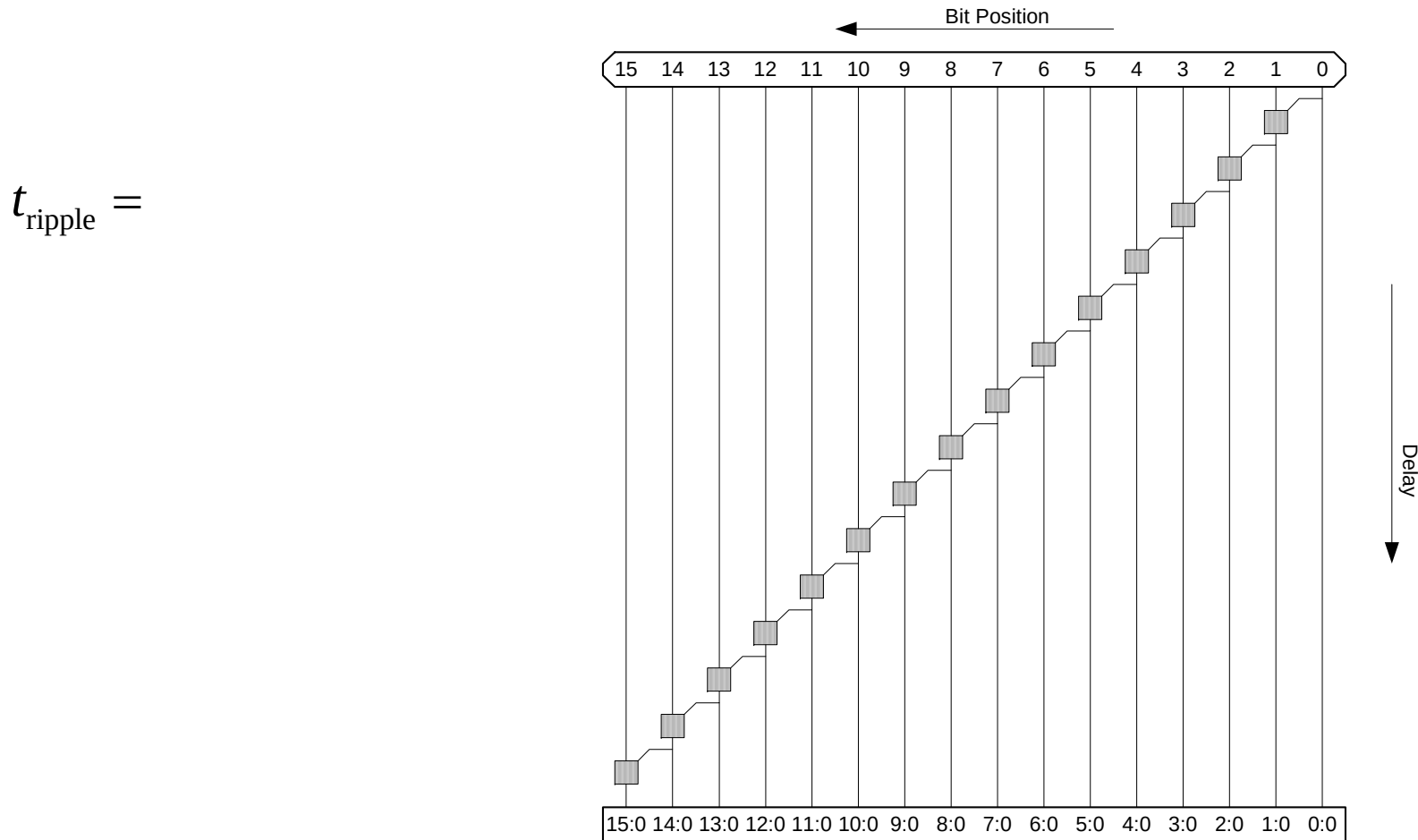


Carry-Ripple Revisited

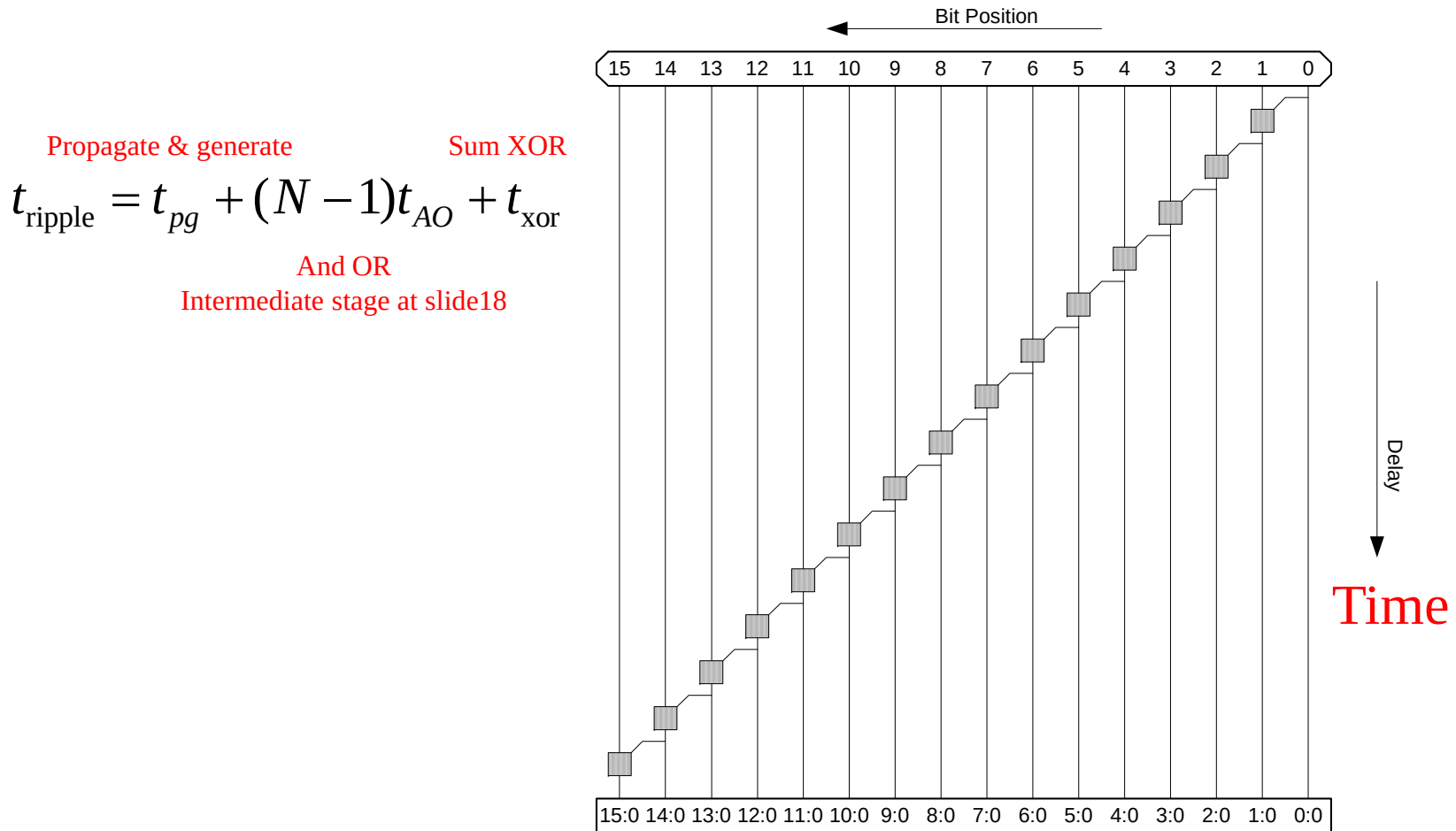
$$G_{i:0} = G_i + P_i \square G_{i-1:0}$$



Carry-Ripple PG Diagram

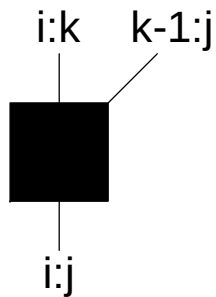


Carry-Ripple PG Diagram

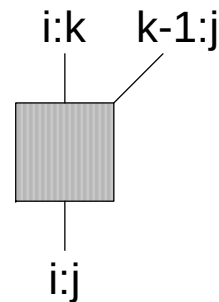


PG Diagram Notation

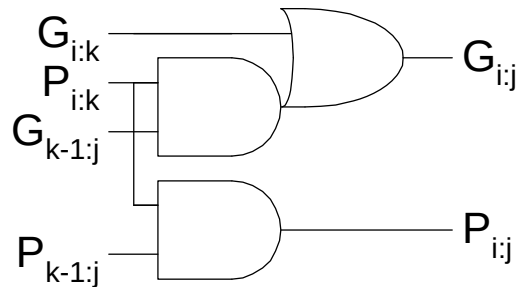
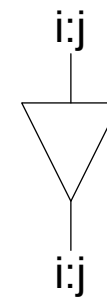
Black cell



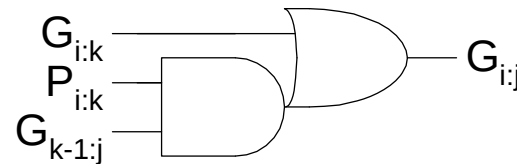
Gray cell



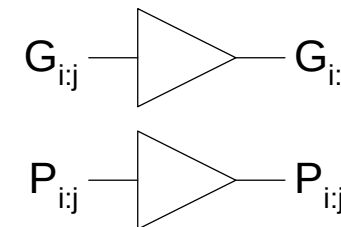
Buffer



P & G terms

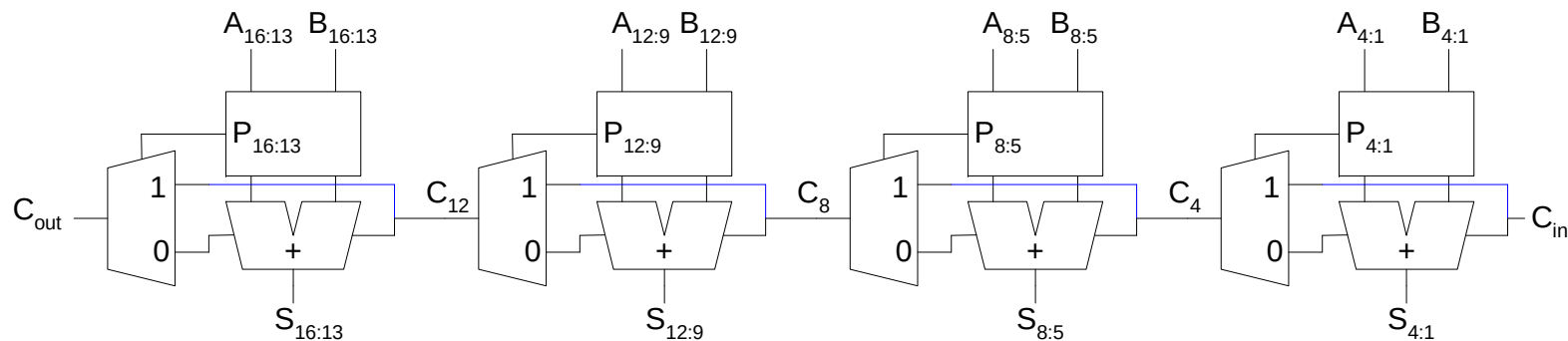


Only G term



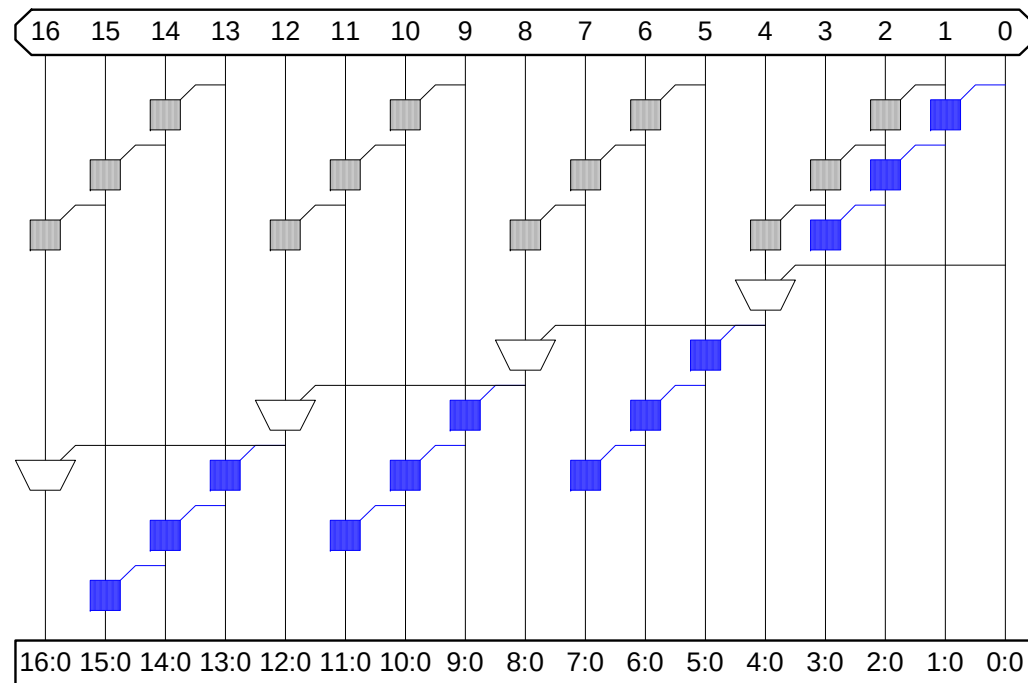
Carry-Skip Adder

- ❑ 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 corresponding 4bit adder result

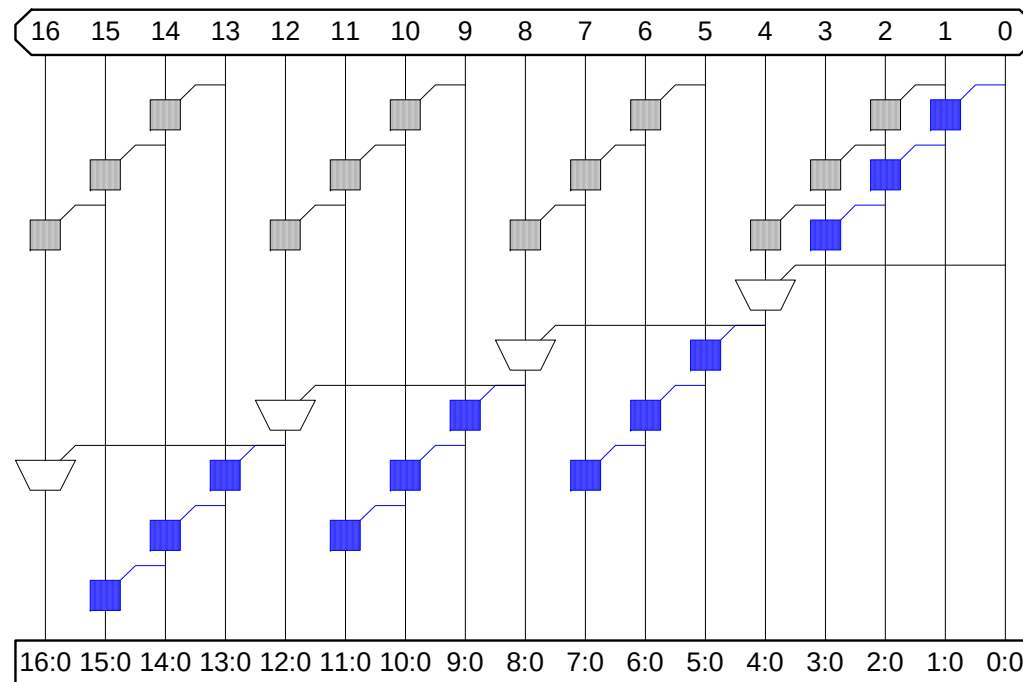
Carry-Skip PG Diagram



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

$$t_{\text{skip}} =$$

Carry-Skip PG Diagram

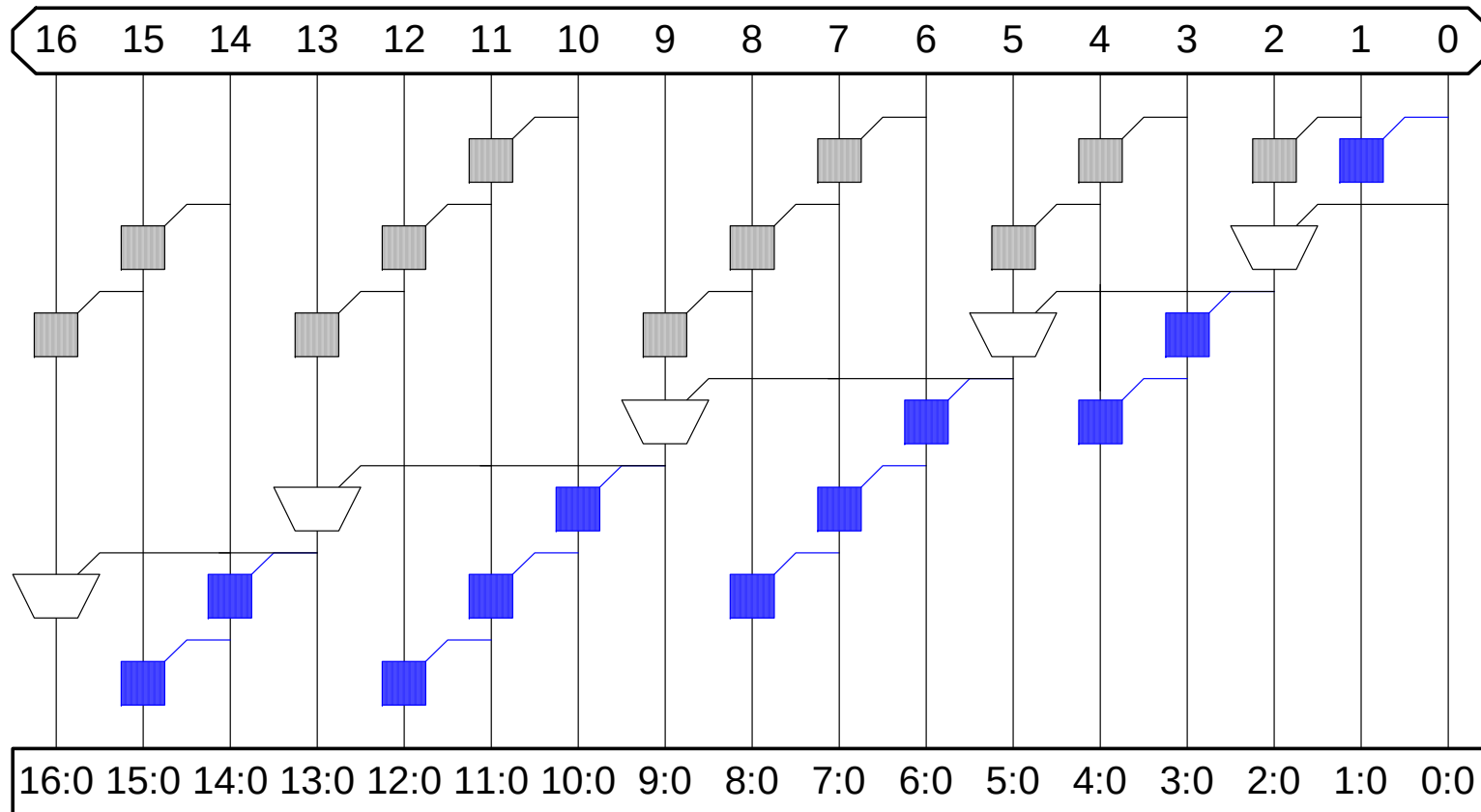


Black: P logic
Blue: Propagate
inside 4 bit blocks

For k n-bit groups ($N = n \cdot k$) ($k \cdot n$ bit groups)

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

Variable Group Size

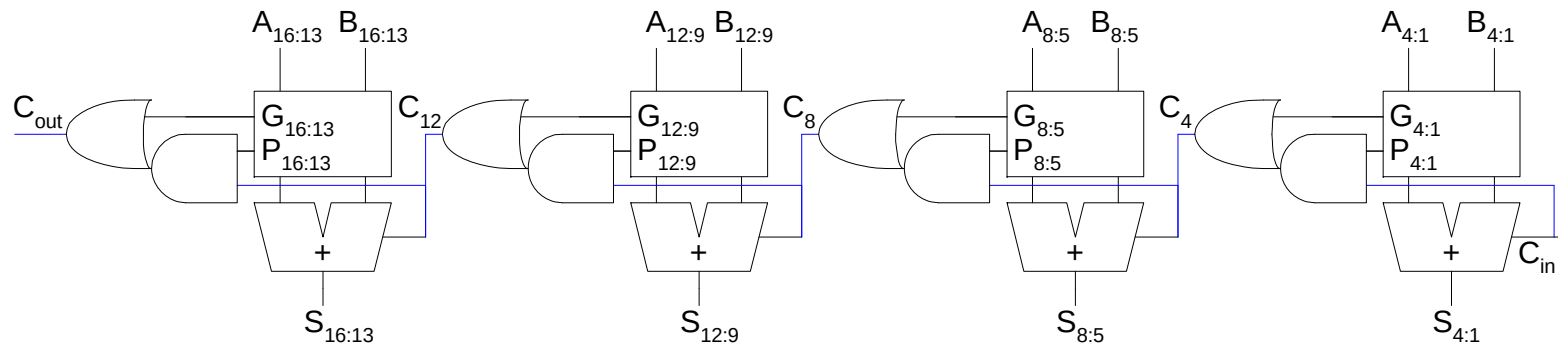


Delay grows as $O(\sqrt{N})$

Carry-Lookahead Adder

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

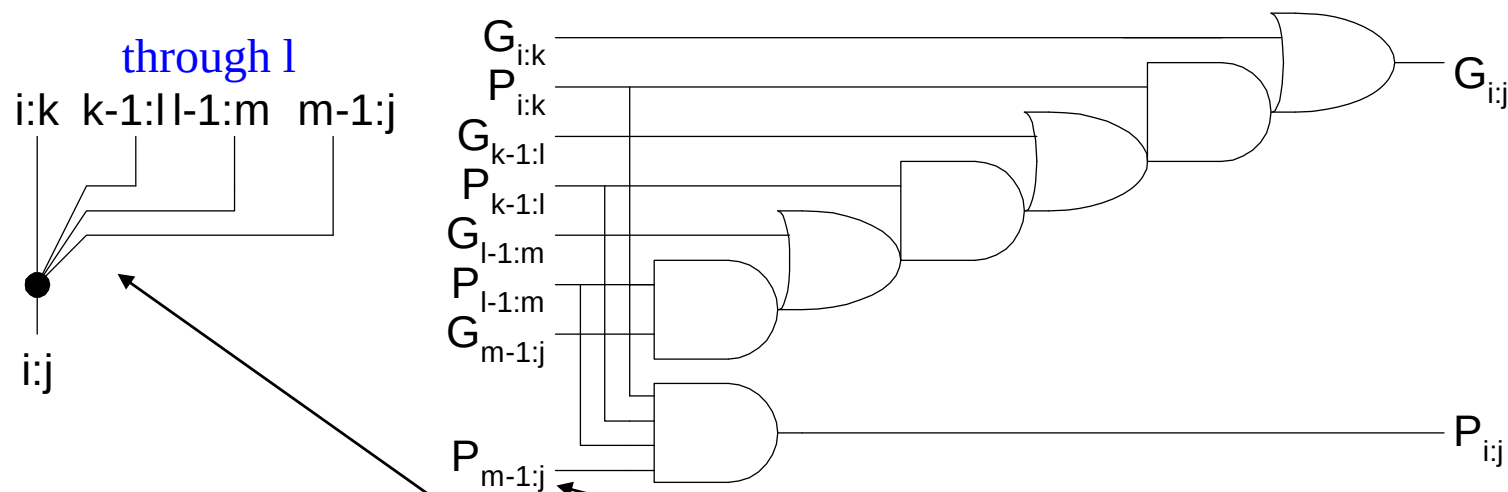
Carry skip waits for the first 4 bit adder
The carry out propagation eliminated in CLA





Higher-Valency Cells

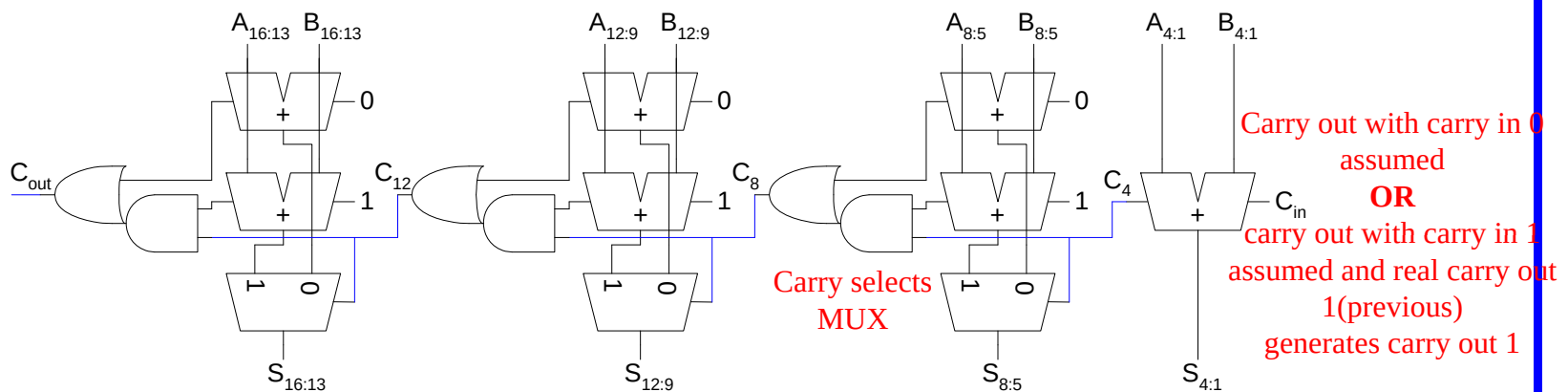
Valency : Relative capacity to unite



Valency 4 here ($i : k$ divided into 4 groups)
For each P , G paths are divided as above respectively.

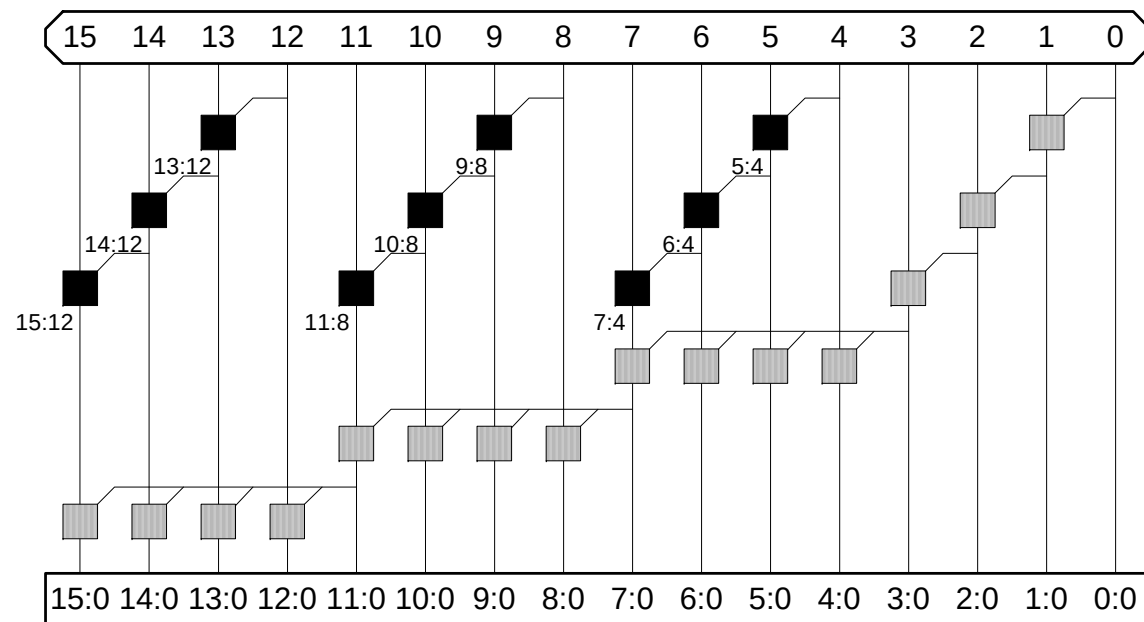
Carry-Select Adder

- ❑ 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

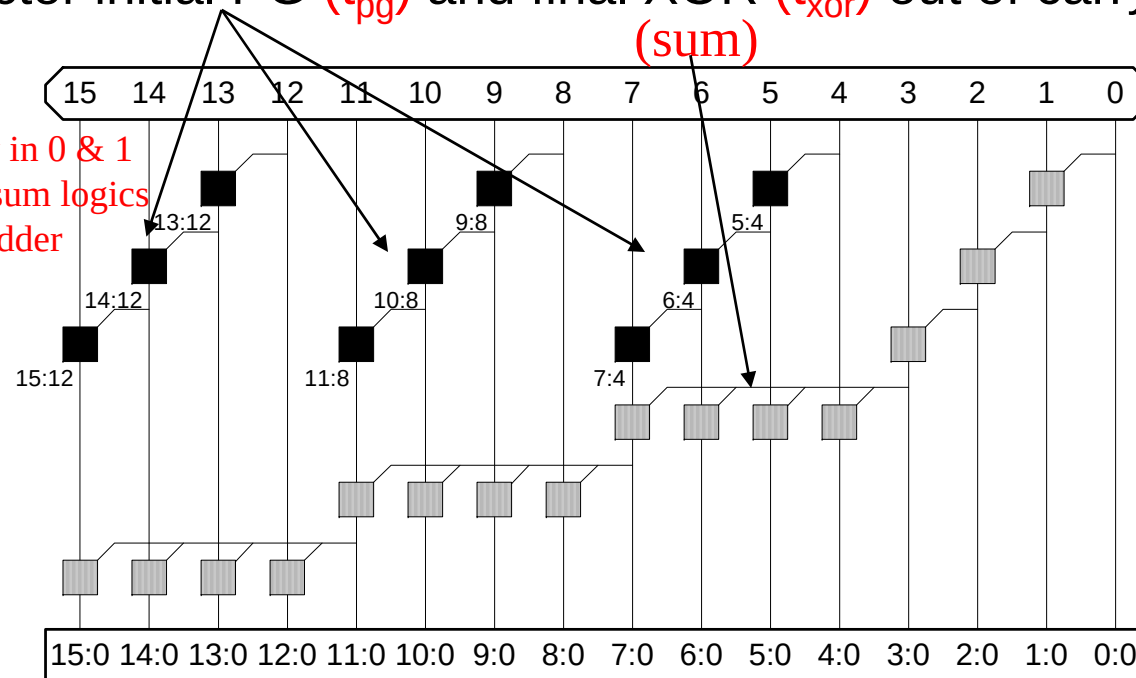


$$t_{\text{increment}} =$$

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



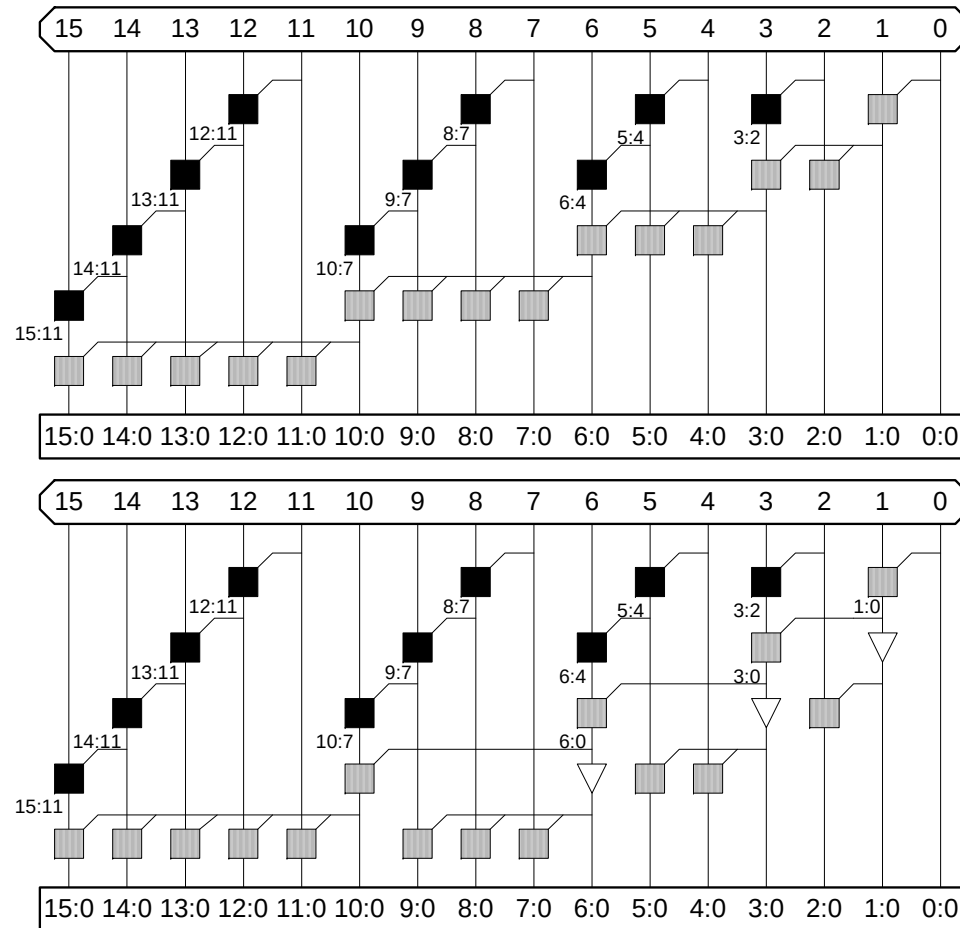
First 4bit addition in series
In parallel with
two possible addition results
(carry in 0 and 1 cases)

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

Variable Group Size (skip)

- Also buffer noncritical signals

Buffered version



Tree Adder

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

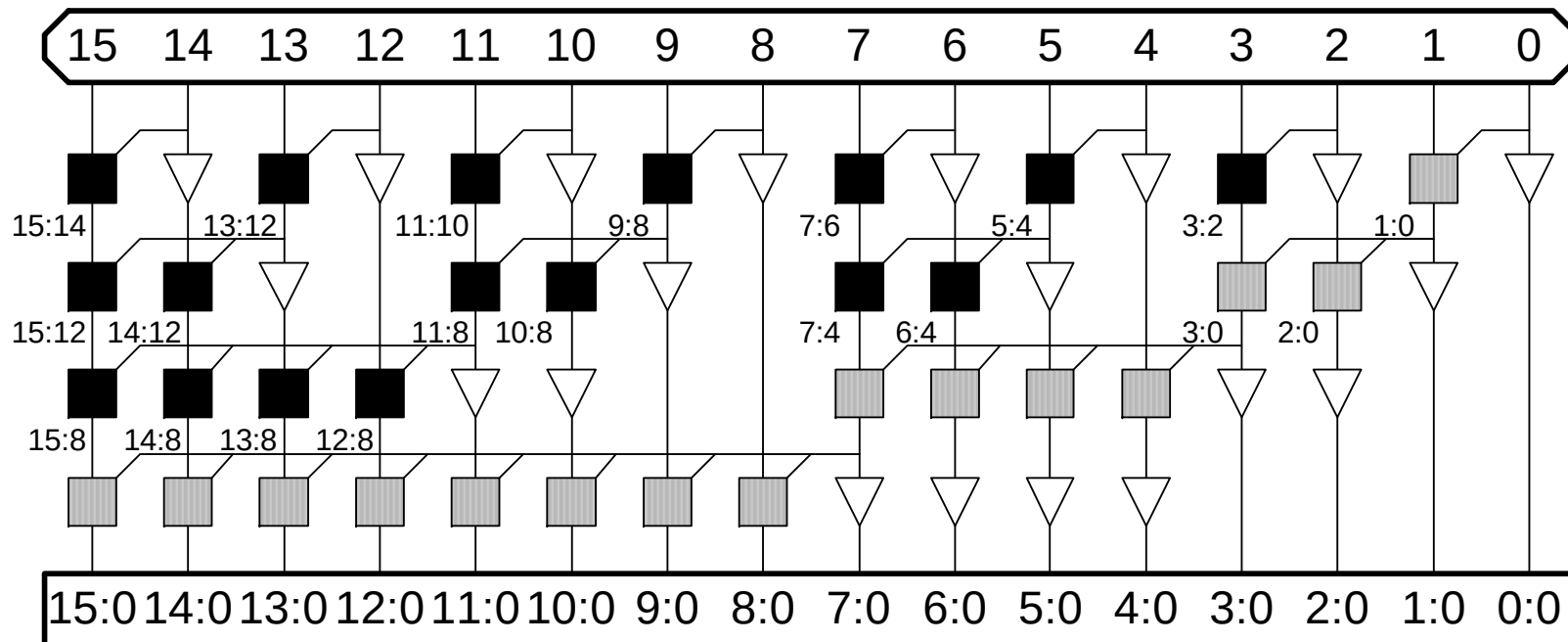
(Usually 2 bits P, G generation in parallel and sum them up from lsb to msb considering delay, higher bit sum up in parallel with lower bit sum up)
- ❑ Many variations on tree adders

Tree Adder Taxonomy

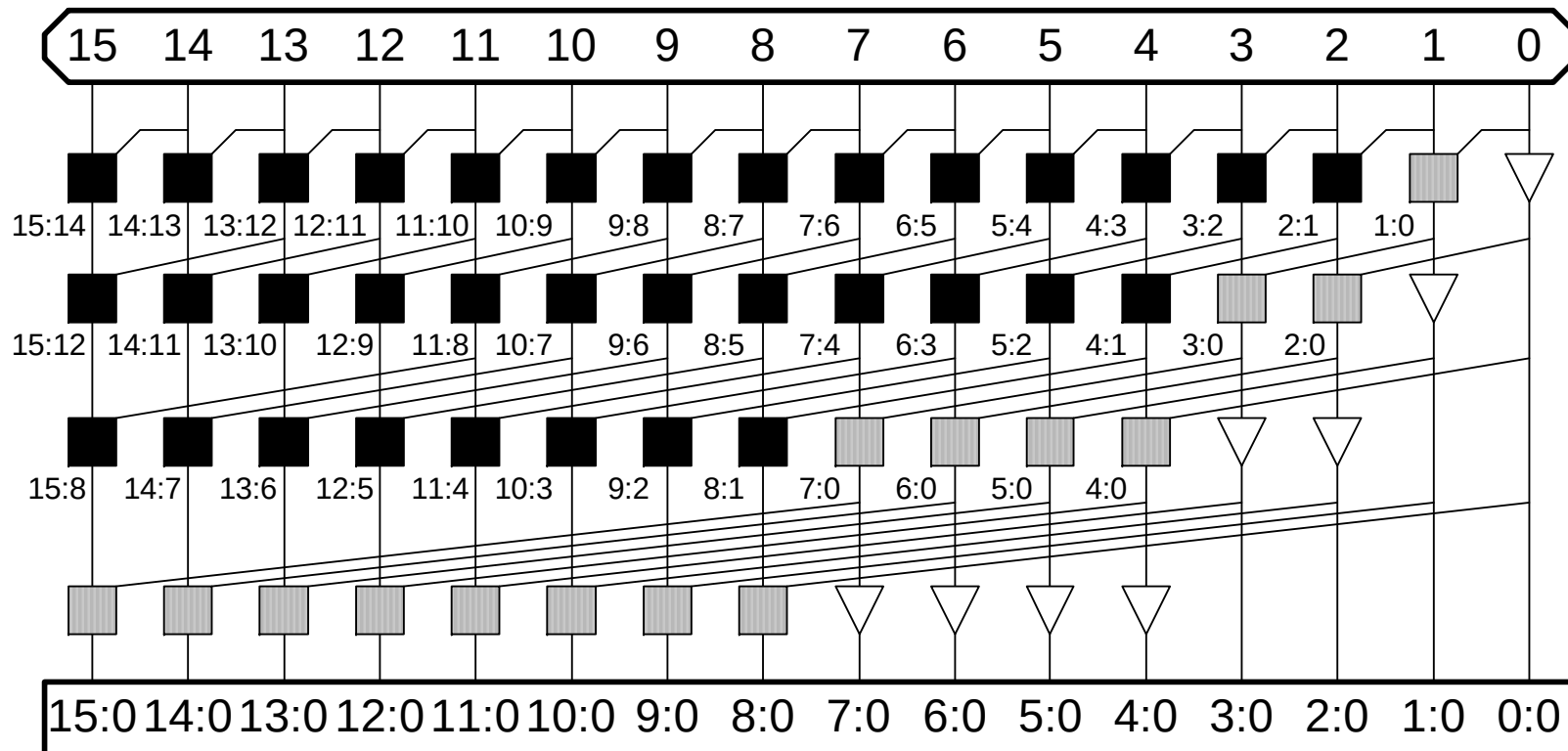
(Taxonomy : Division into ordered groups or categories)

- ❑ 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: $L + l$ (MAX. # of logic levels to the result)
(not 1)
 - 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
$$l + f + t = L - 1$$

Sklansky



Kogge-Stone

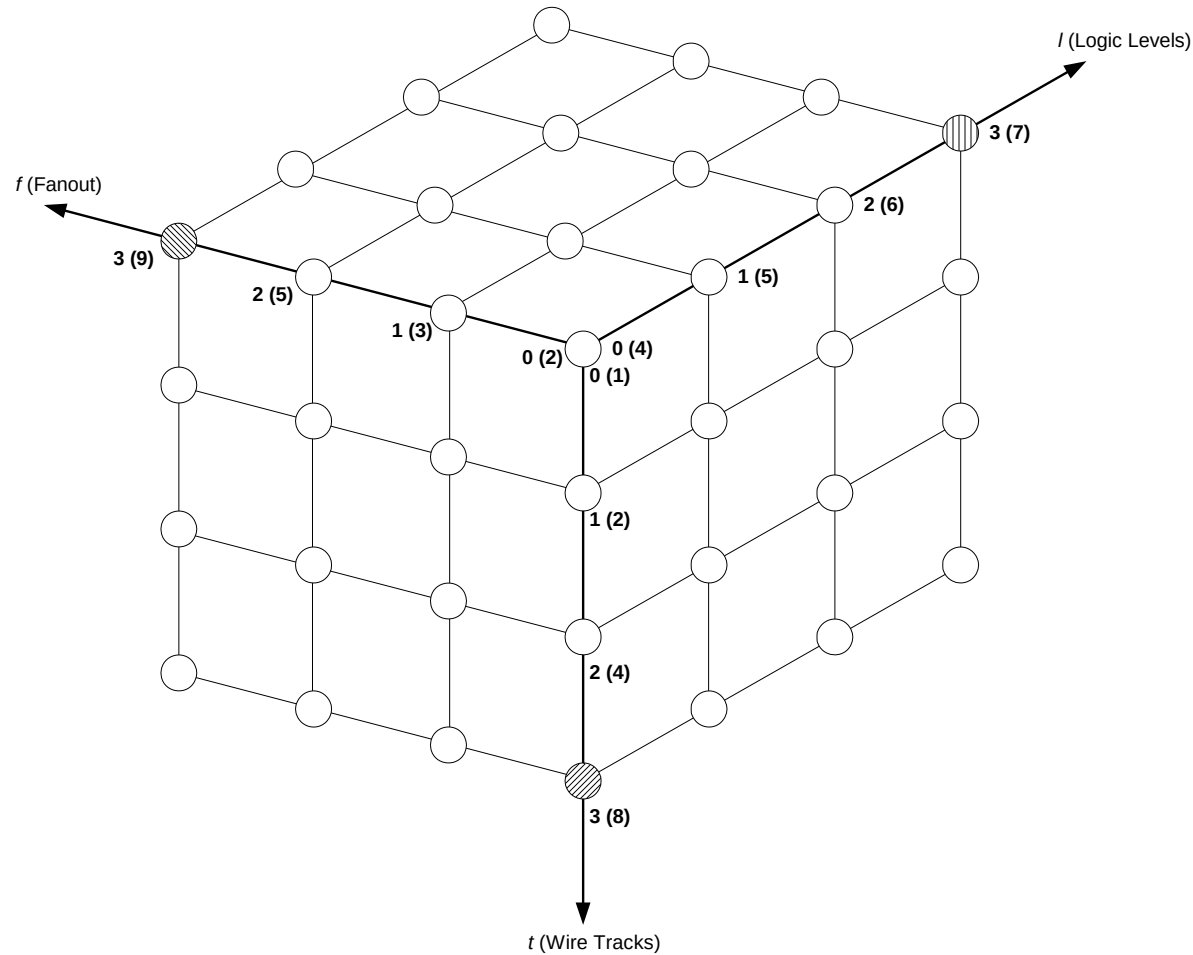


Tree Adder Taxonomy

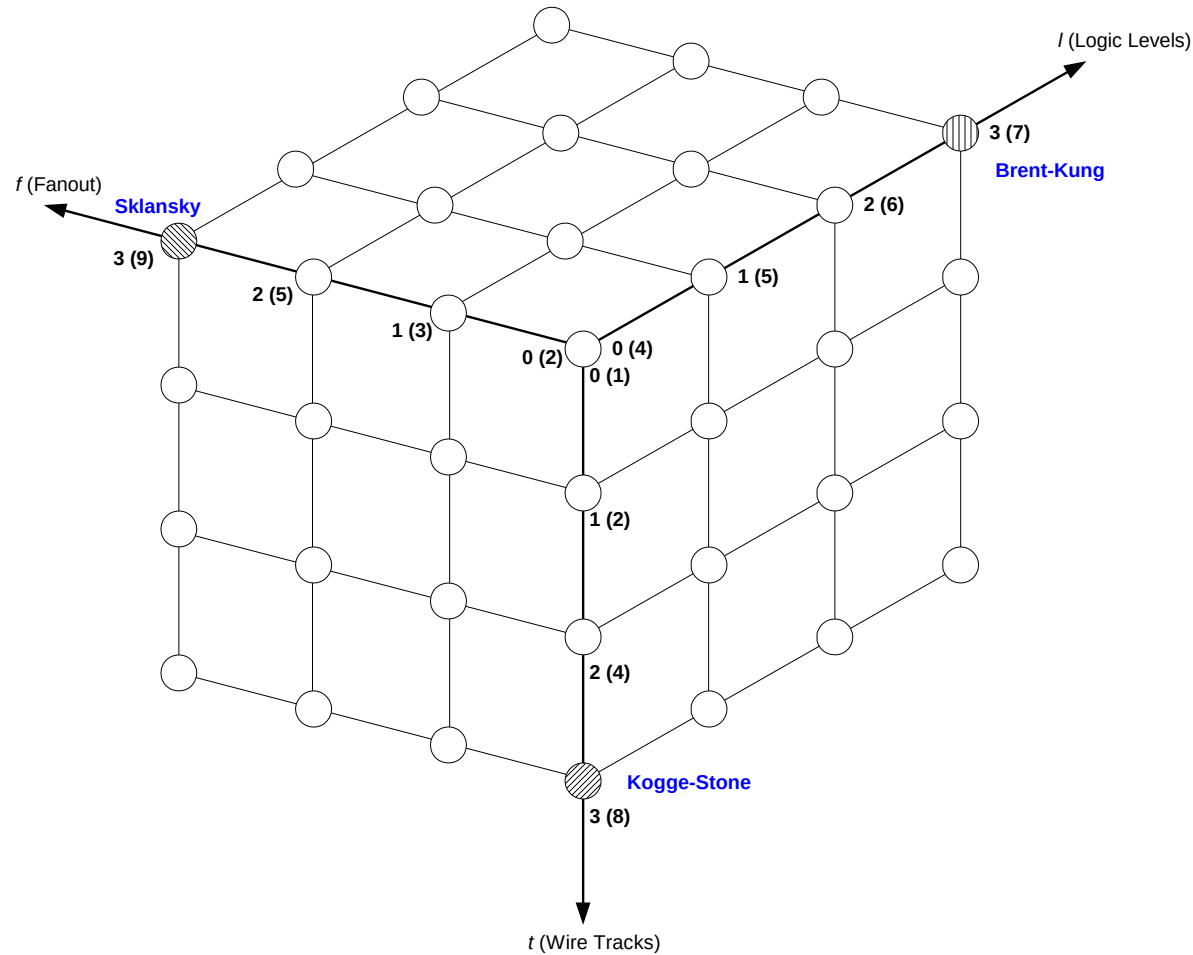
- ❑ Ideal N-bit tree adder would have
 - $L = \log N$ logic levels
 - Fanout never exceeding 2
 - No more than one wiring track between levels
- ❑ Describe adder with 3-D taxonomy (l, f, t)
 - Logic levels: $L + l$
 - Fanout: $2^f + 1$
 - Wiring tracks: 2^t
- ❑ Known tree adders sit on plane defined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

SKIP

Tree Adder Taxonomy



Tree Adder Taxonomy

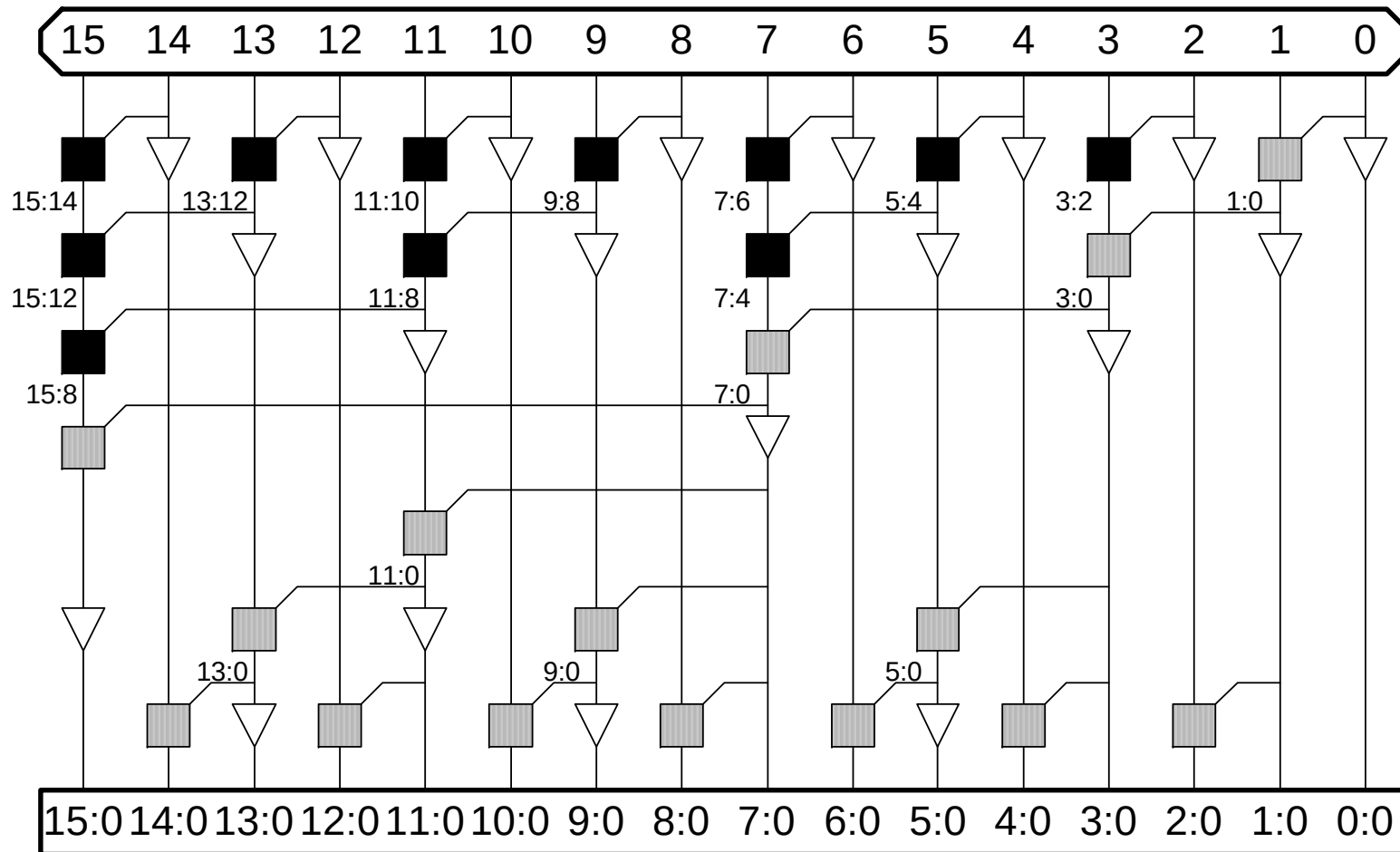


SKIP

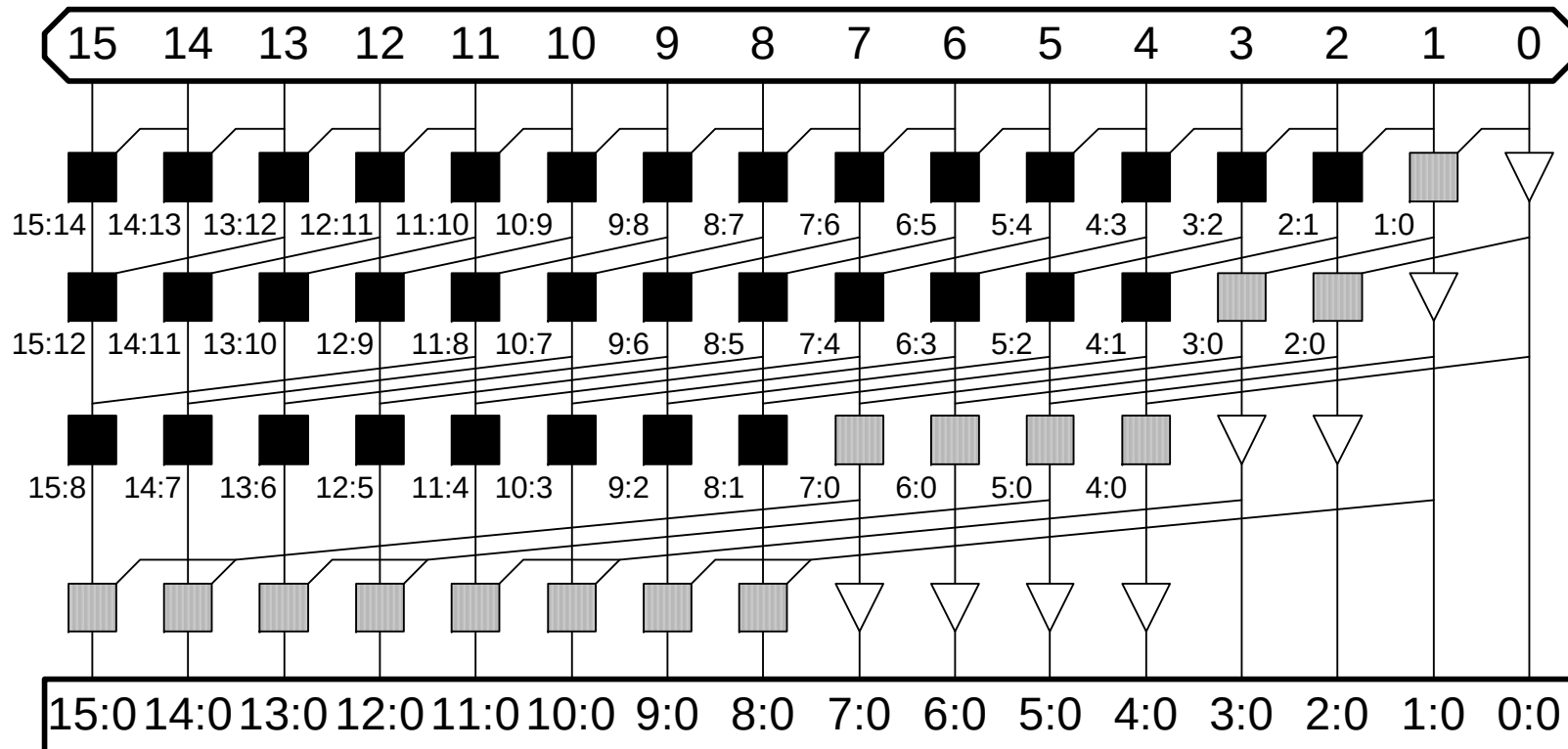
$$S_i = P_i \oplus G_{i-1:0}$$

Brent-Kung (moved after pp.34)

(prefix 2 → prefix 4 → ...)

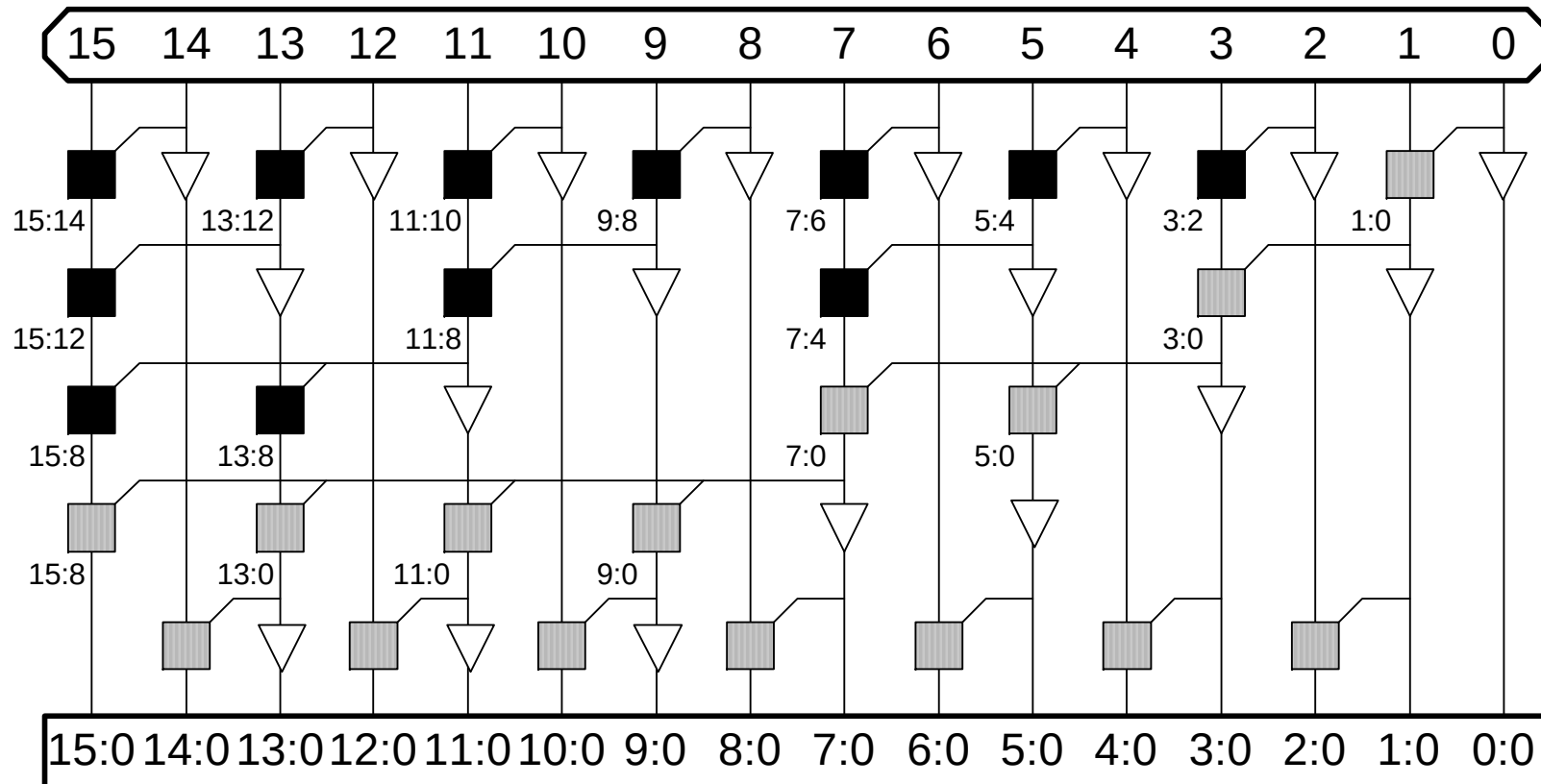


Knowles [2, 1, 1, 1]



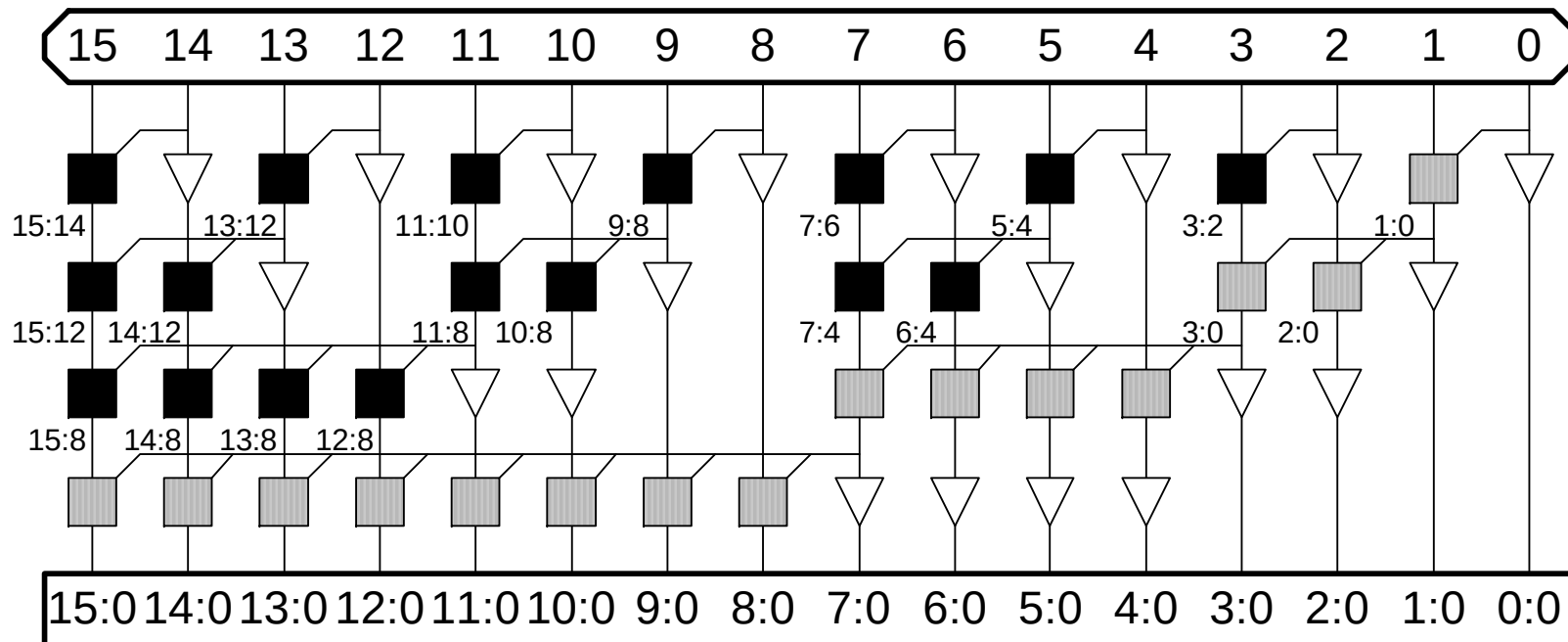
SKIP

Ladner-Fischer



SKIP

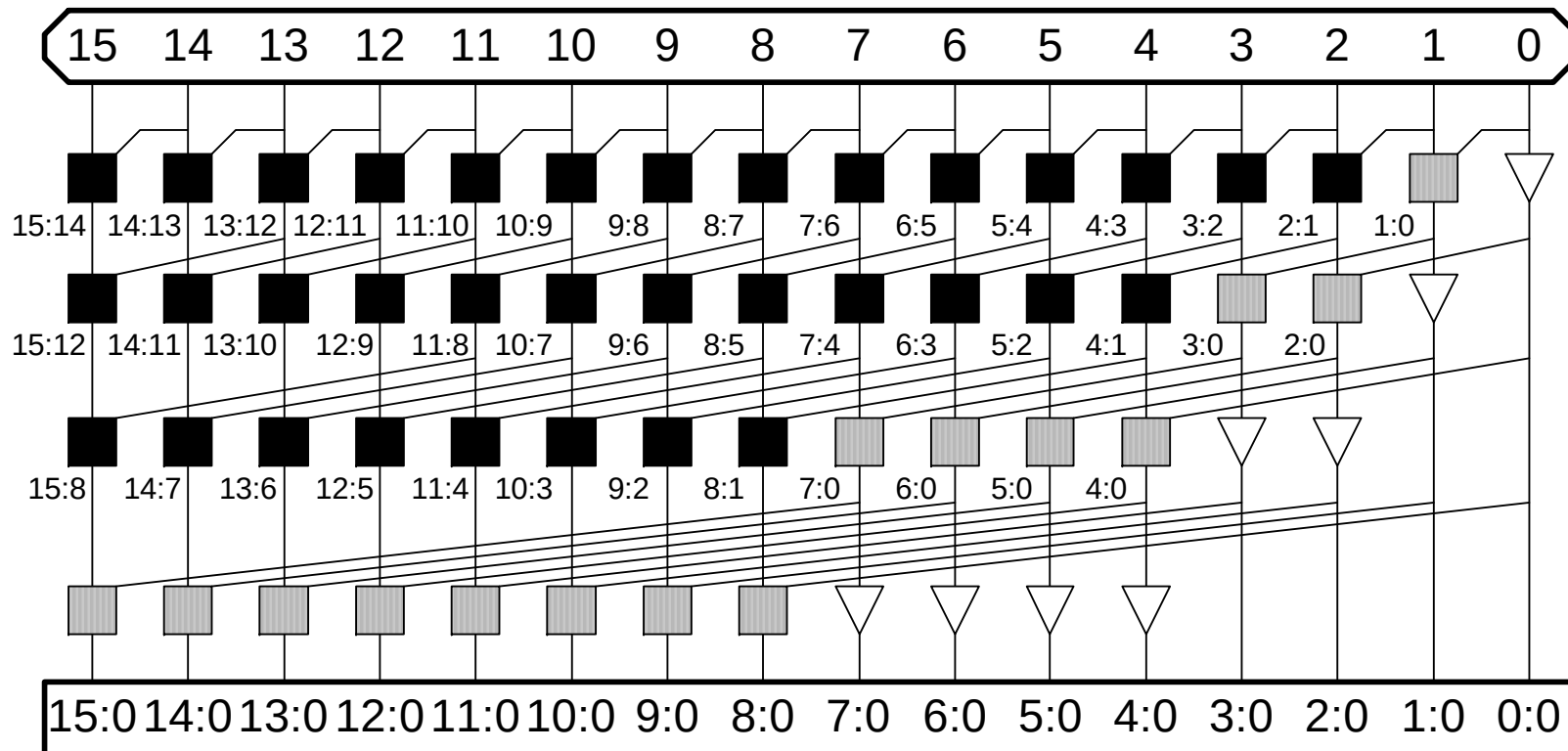
Sklansky (moved after pp.34) (go to pp.43)



Intermediate Prefix
with large fan out

SKIP

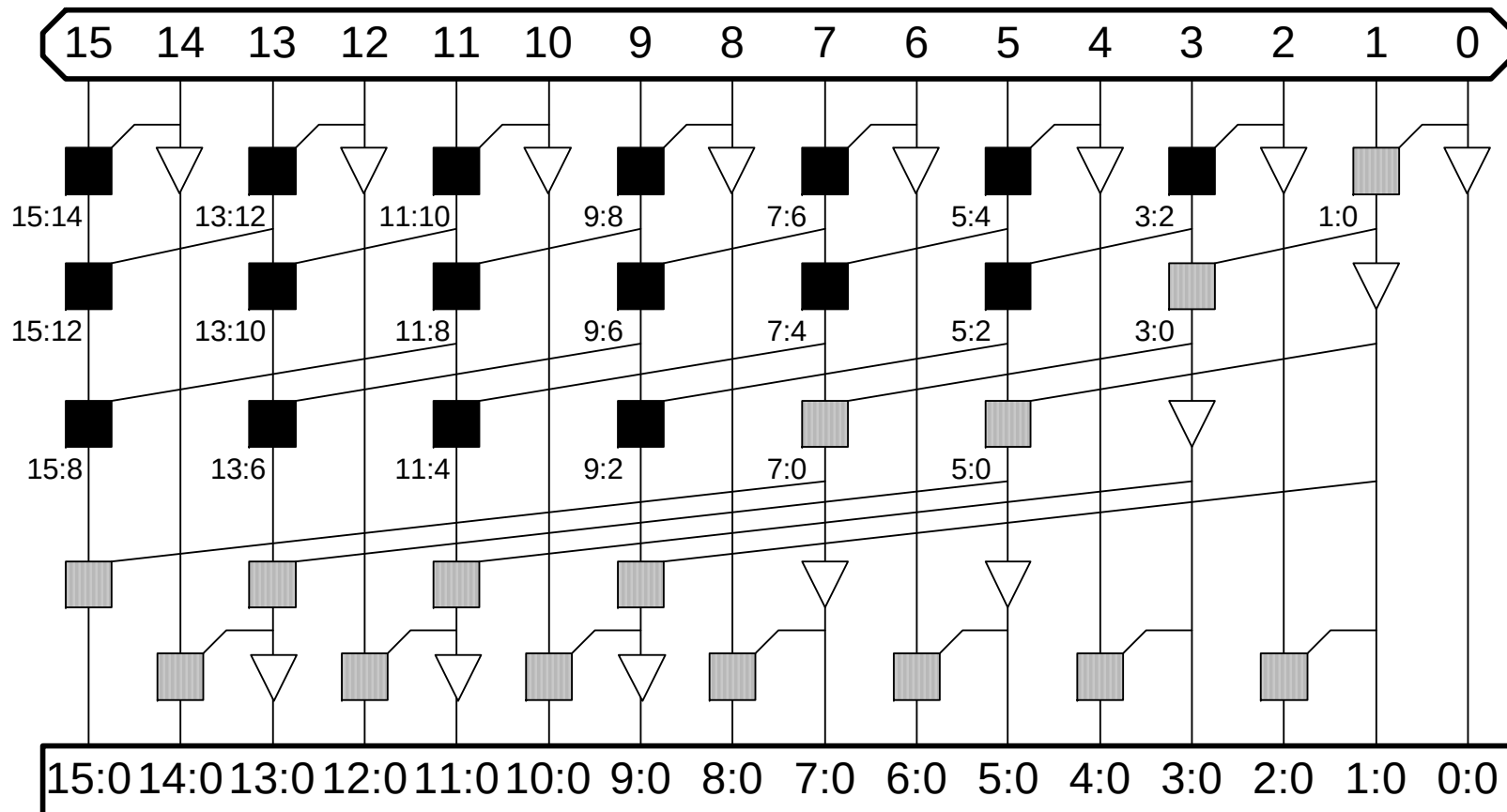
Kogge-Stone (moved after pp.34)



Many long wires, large power consumption

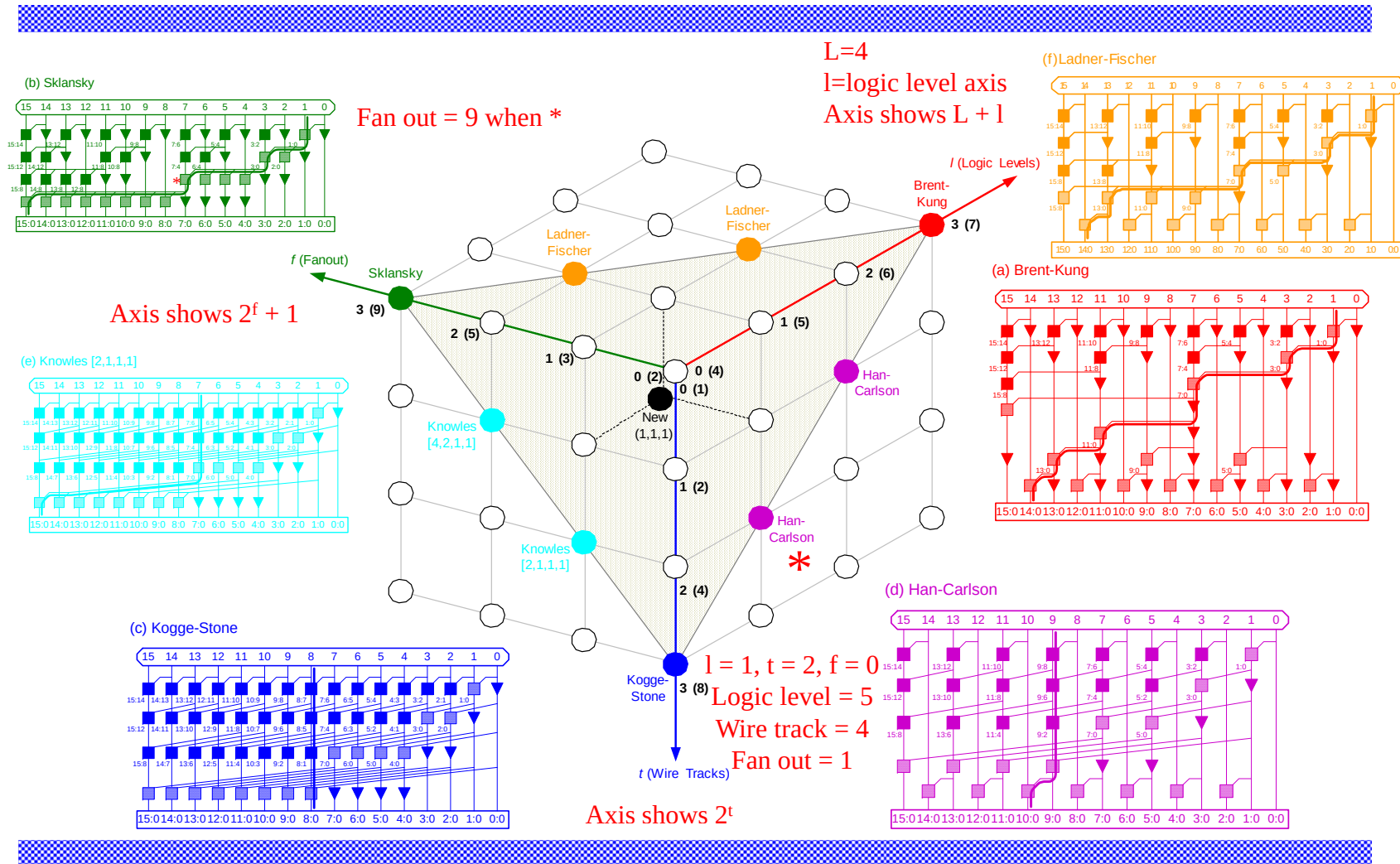
SKIP

Han-Carlson



Logic level = delay, fan out = power, wire track = area

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	$N/2$	$N \log_2 N$