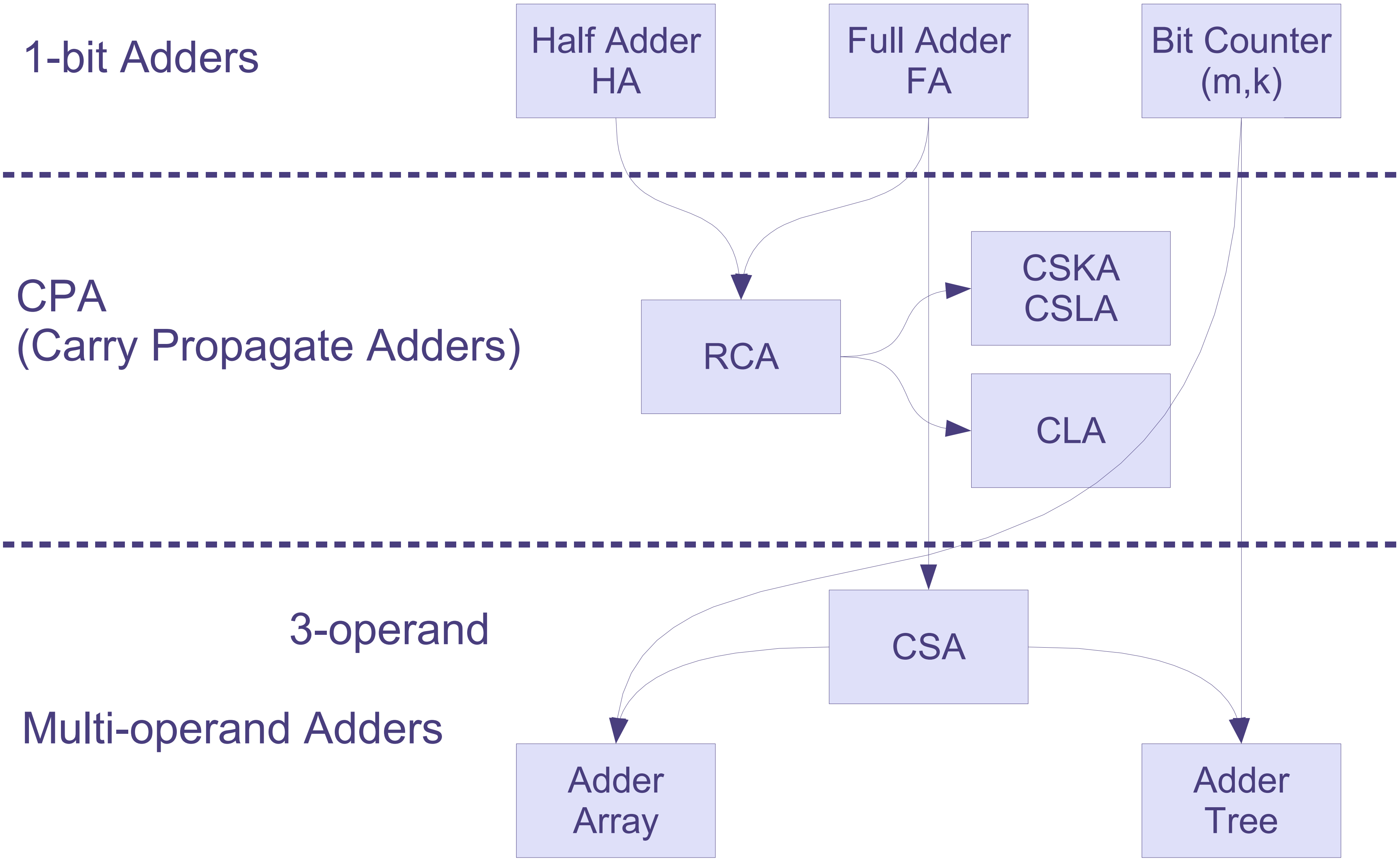
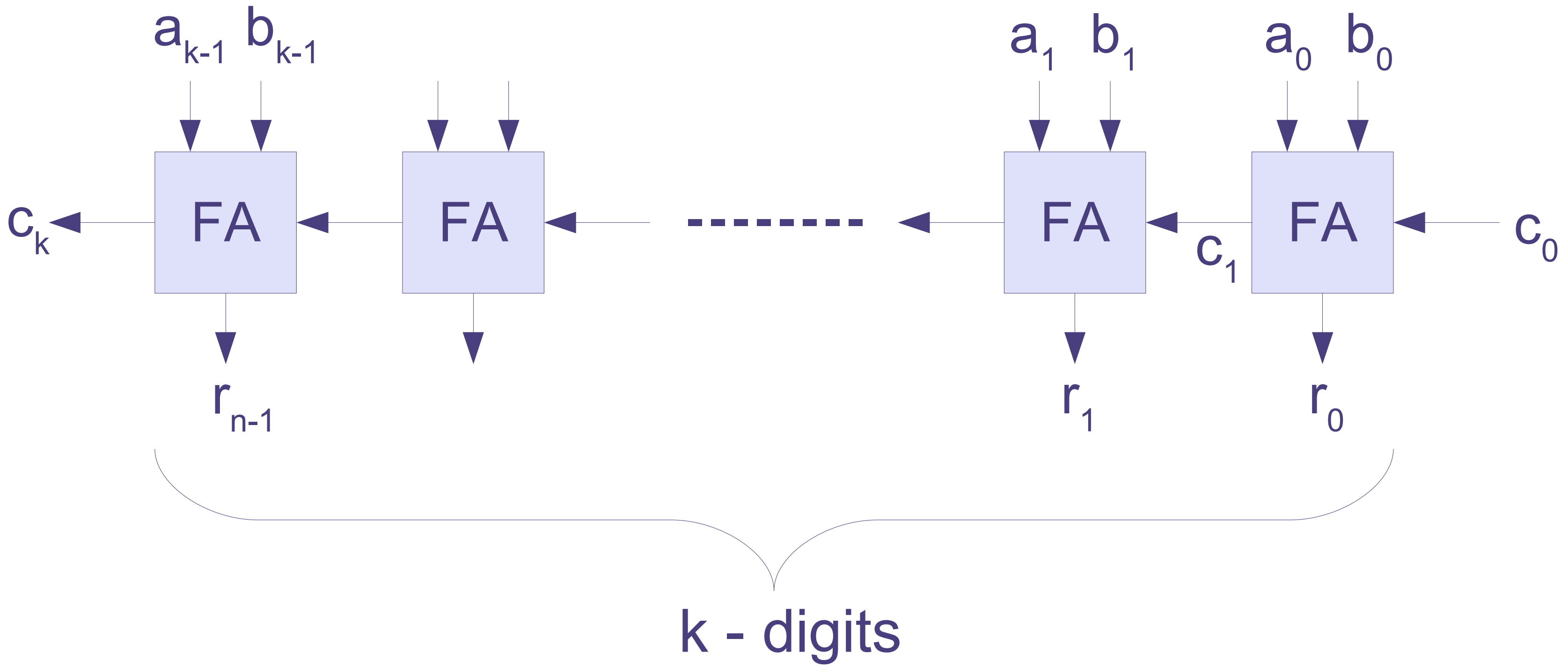
Fast Adders

Adders Overview (not complete)



RCA

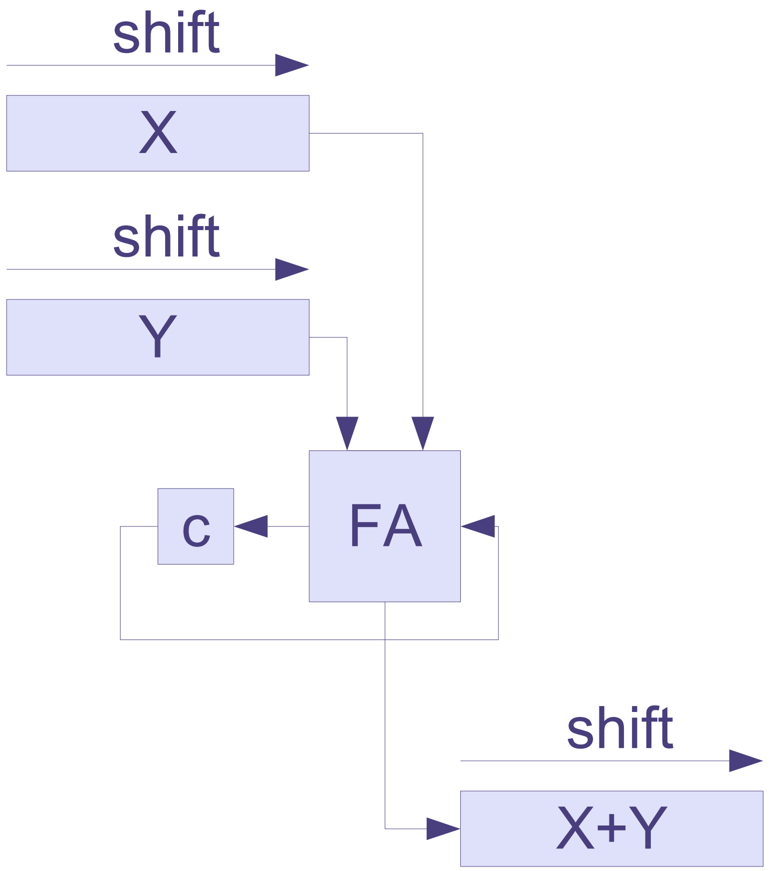
- Ripple Carry Adder *(Sumator z przeniesieniami szeregowymi)*
  - Simplest, Smallest and Slowest (SSS), but ...
  - Worst-case carry-propagation is Θ(k)



RCA: Bit-serial Implementation

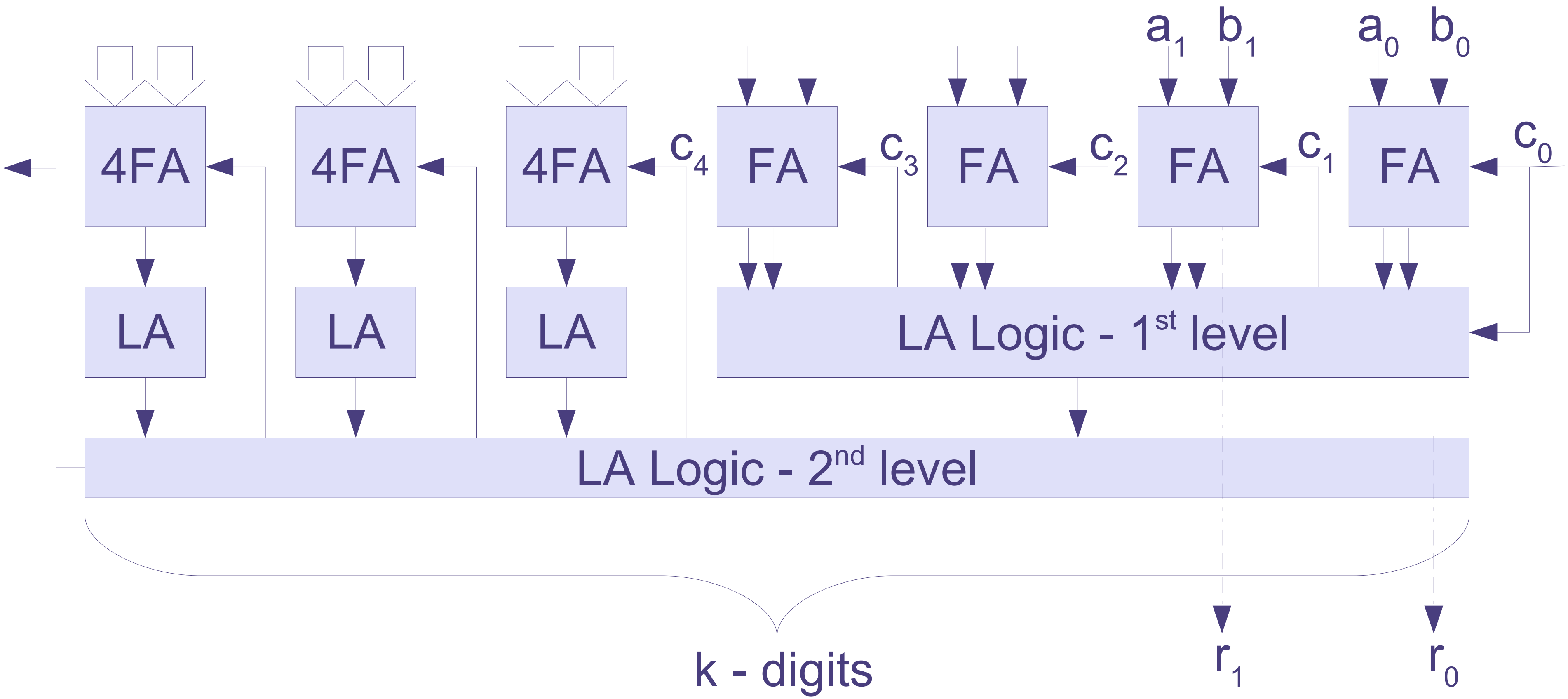
- VLSI implementation advantages:
  - small pin count
  - reduced wire length
  - high clock rate
  - small space
  - low power consumption
- Alternative to pipeline units for parallel processing
- Latency (the same): Θ(k)

Bit-serial RCA



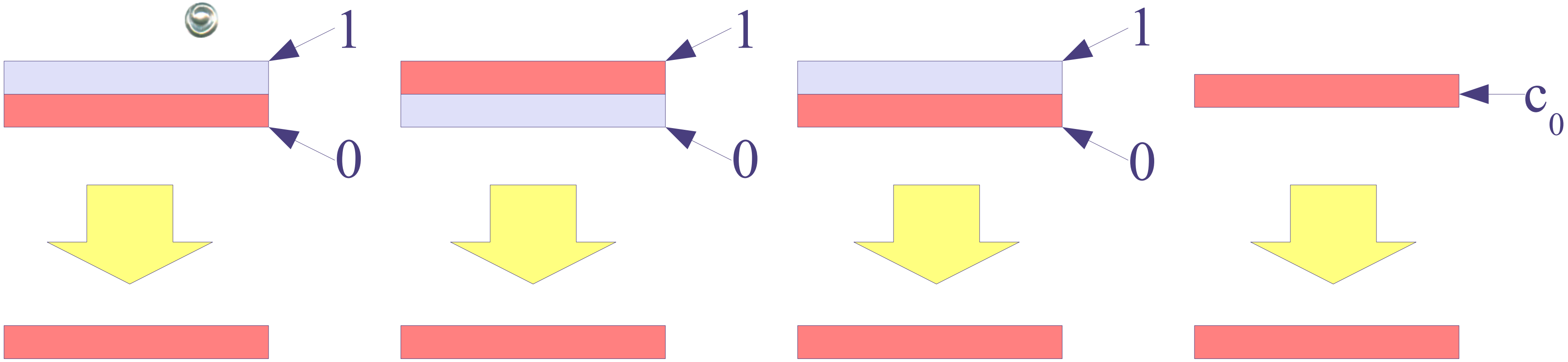
CLA

- Carry Lookahead Adder (Sumator z przeniesieniami równoległymi)
  - Fast but Complex (lookahead logic units)
  - Worst-case carry propagation is Θ(log k)

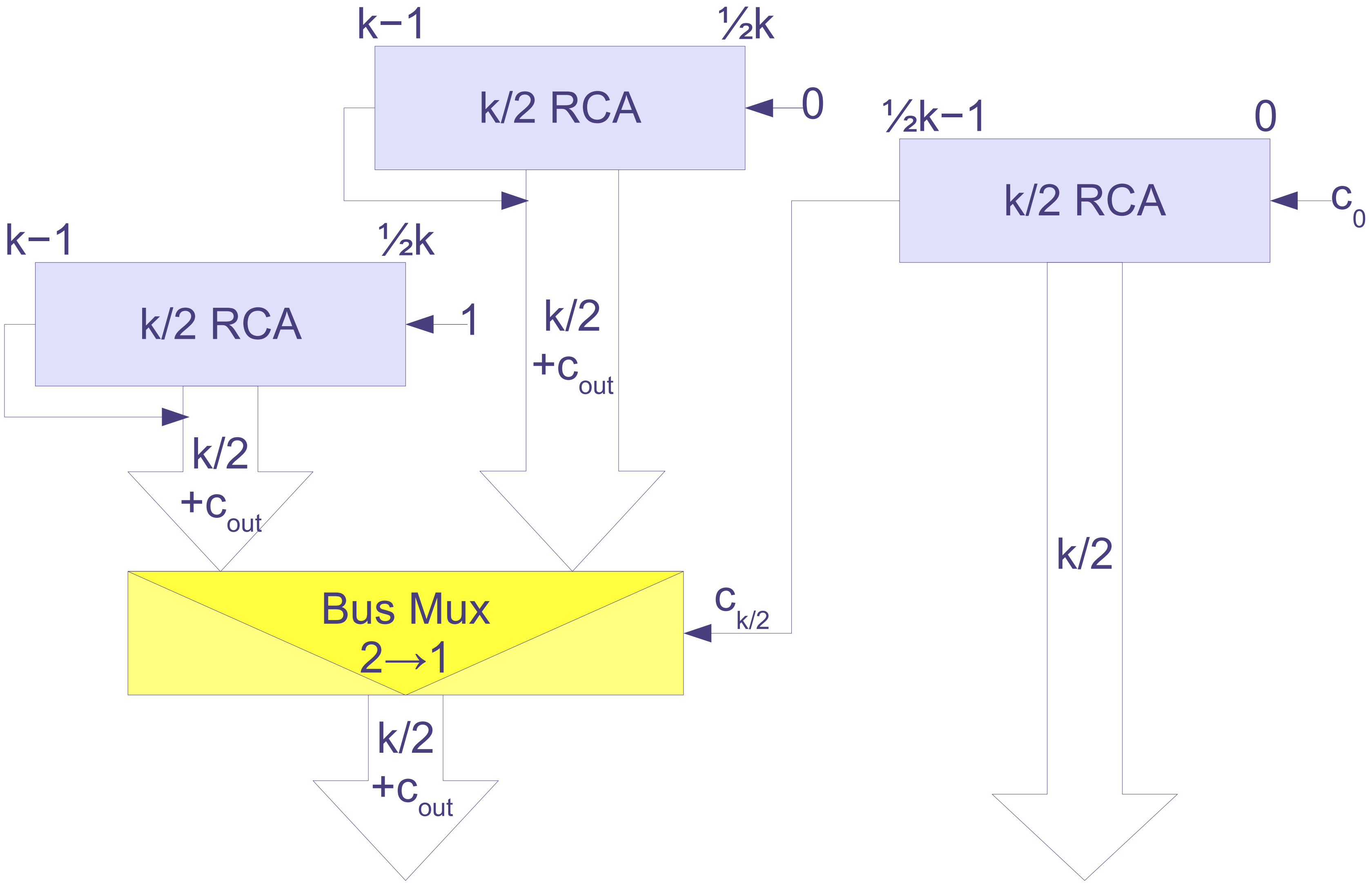


CSLA

- Carry Select Adder (Sumator z wyborem przeniesienia)
  - The oldest logarithmic-time adder Θ(log k)
  - The additions are performed in parallel according to alternative scenarios: carry= 0 or 1
  - The final selection of results is made with the computed value of carry
  - Fast, but with high complexity of (selection) hardware
  -



1-Level CSLA

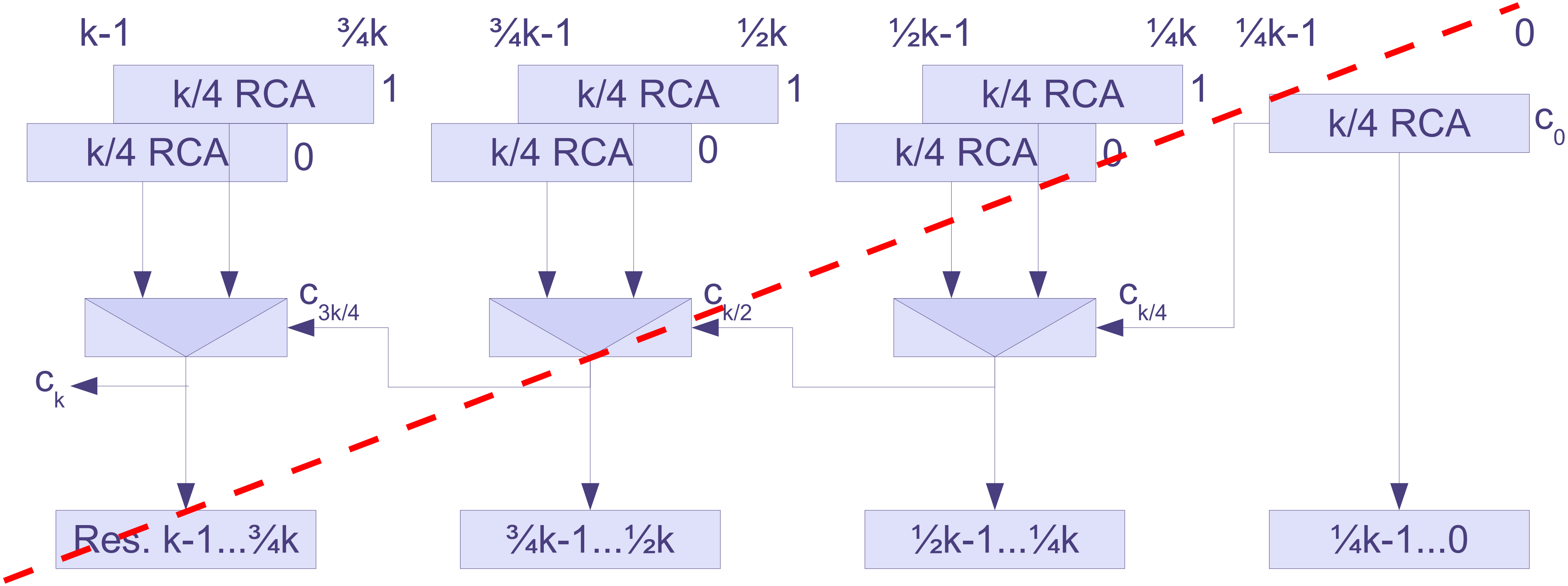


c_{out} + high k/2 bits of result

k/2 low bits of result

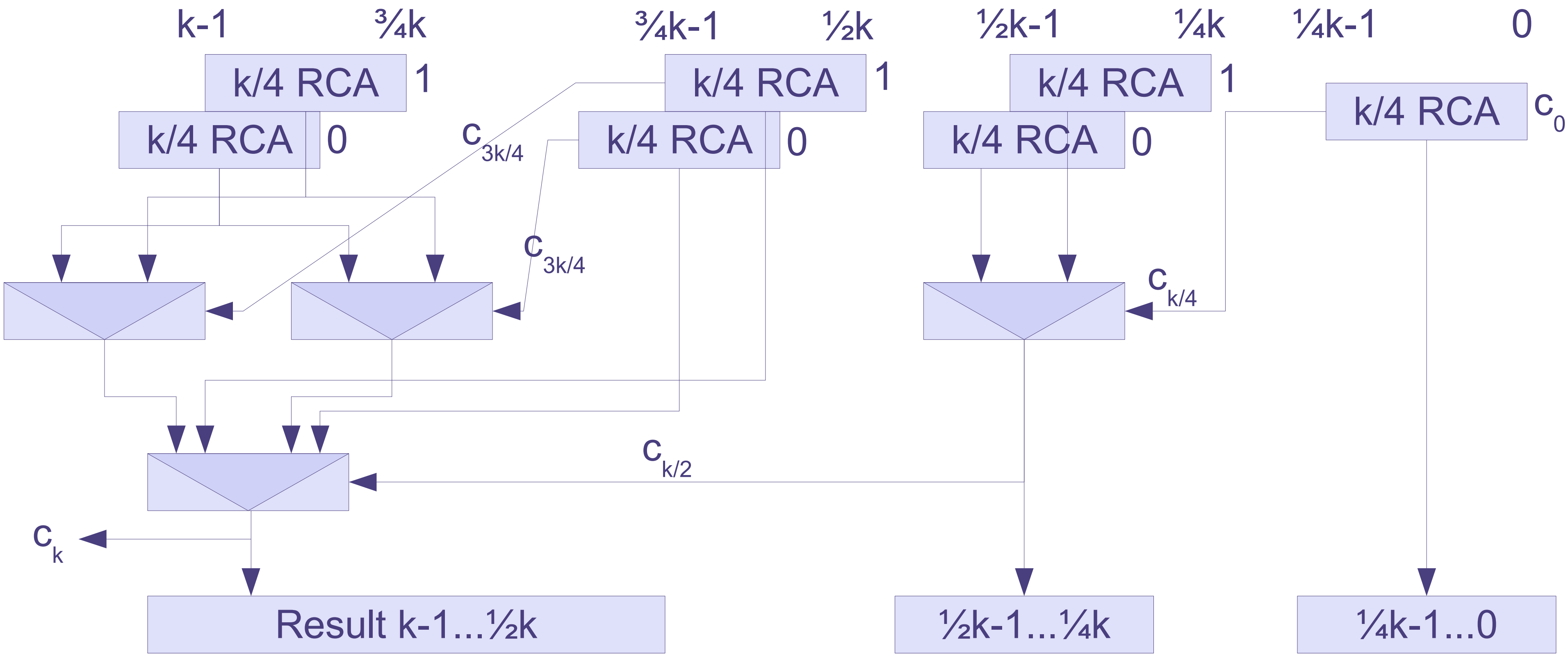
Block-carry Propagation in CSLA

- 1-level architecture cannot be applied to multiple blocks due to block-carry propagation delay, which gives effect similar to RCA



Two-Level CSLA

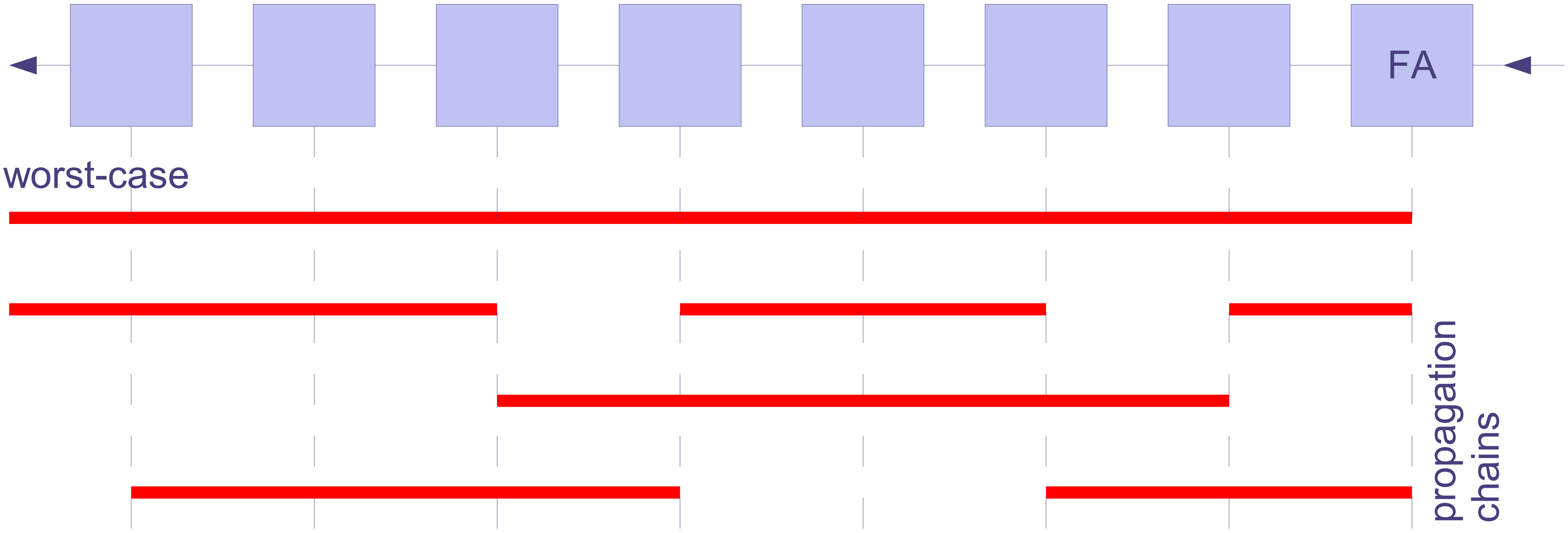
- Block-carry propagation is fast, but design complex
- Carry-bits at 2nd level select longer parts of a results
- Concept may be extended to more levels (complex)



Computer Architecture, IFE CS and T&CS, 4th sem

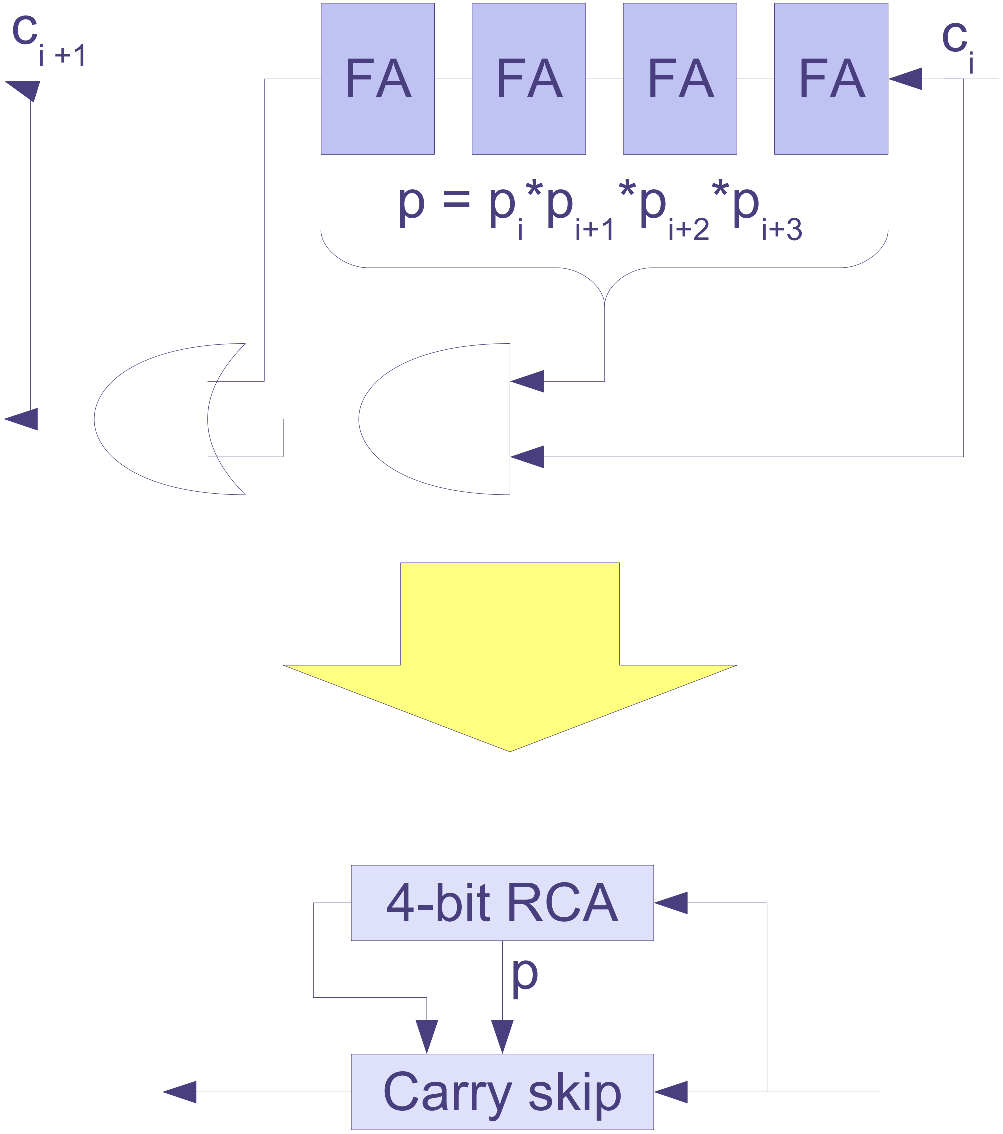
CSKA

- Carry Skip Adder (Sumator z przeskokiem przeniesień)
- Carries can be generated, propagated or absorbed
- Propagation chains are evaluated in parallel
- How to speed up the worst-case propagation: ?
  allow for carry propagation over propagation chains



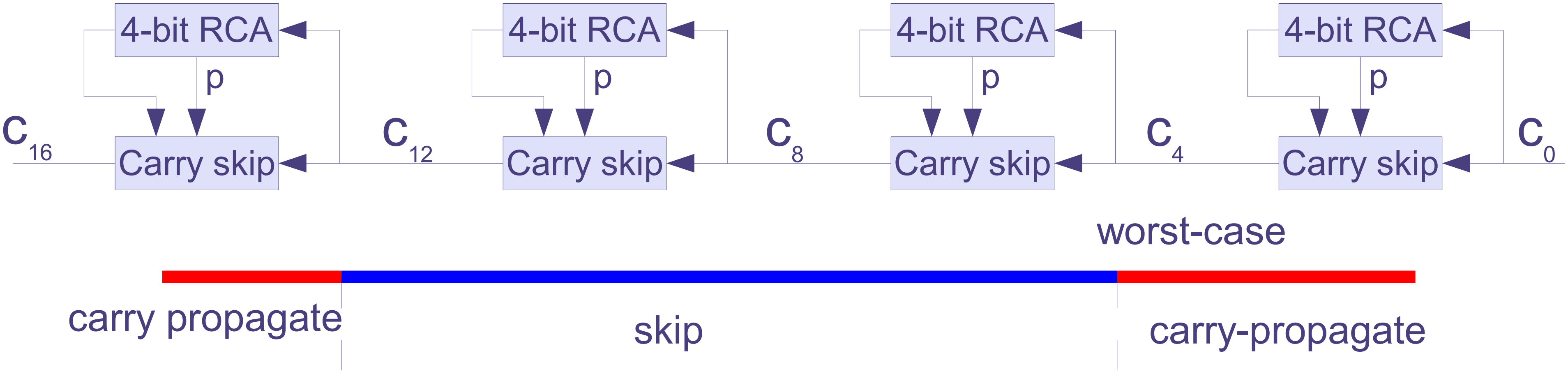
Skip Block Logic

- Carry-in is propagated through n-stages if p=1
- Propagate condition p is easily computable: $p_i=a_i+b_i$



Worst-Case Analysis CSKA

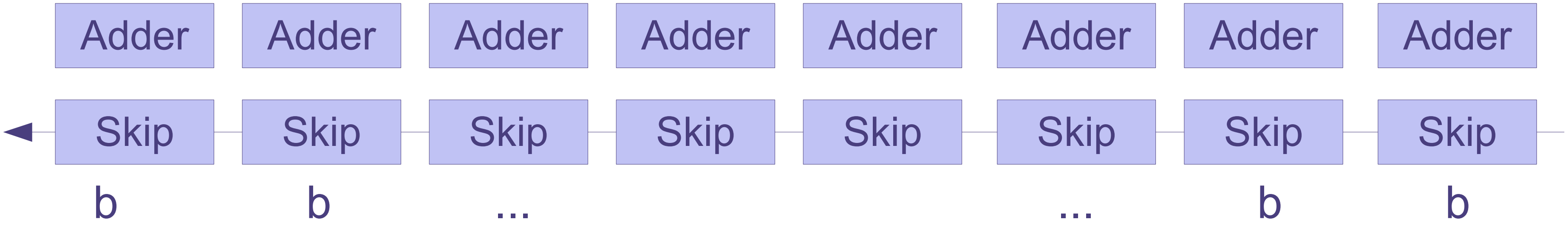
- 16-bit CSKA
- The longest delay due to carry propagation:
    - propagation through bits 1-3 and OR
    - skip bits 4-11
    - propagation through bits 12-14



(block 0 and last do not contribute to carry-propagation delay)

CSKA Architecture Variations

- fixed-size skip blocks (block size b)

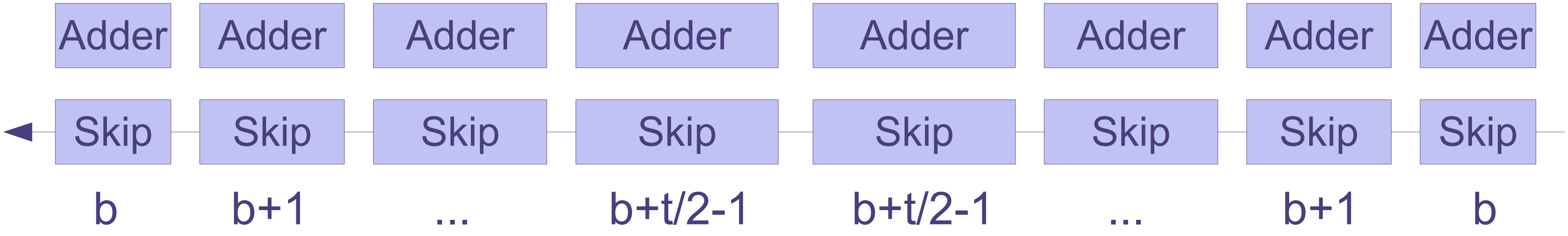


Adder Adder Adder Adder Adder Adder Adder Adder

Skip Skip Skip Skip Skip Skip Skip Skip

b b ... ... b b

- Variable-size skip blocks (t – number of blocks)



Adder Adder Adder Adder Adder Adder Adder Adder

Skip Skip Skip Skip Skip Skip Skip Skip

b b+1 ... b+t/2-1 b+t/2-1 ... b+1 b

CSKA - Results

- T - Carry-propagation delay in CSKA
  - fixed-size blocks (b):
    - $b_{opt} = (k/2)^{1/2}$
    - $T_{fixCSKA\text{-}opt} = 2(2k)^{1/2} - 3.5$
  - variable-size blocks (t):
    - $b_{opt} = 1,\ t_{opt} = 2\ k^{1/2}$
    - $T_{varCSKA\text{-}opt} = 2\ k^{1/2} - 2.5$

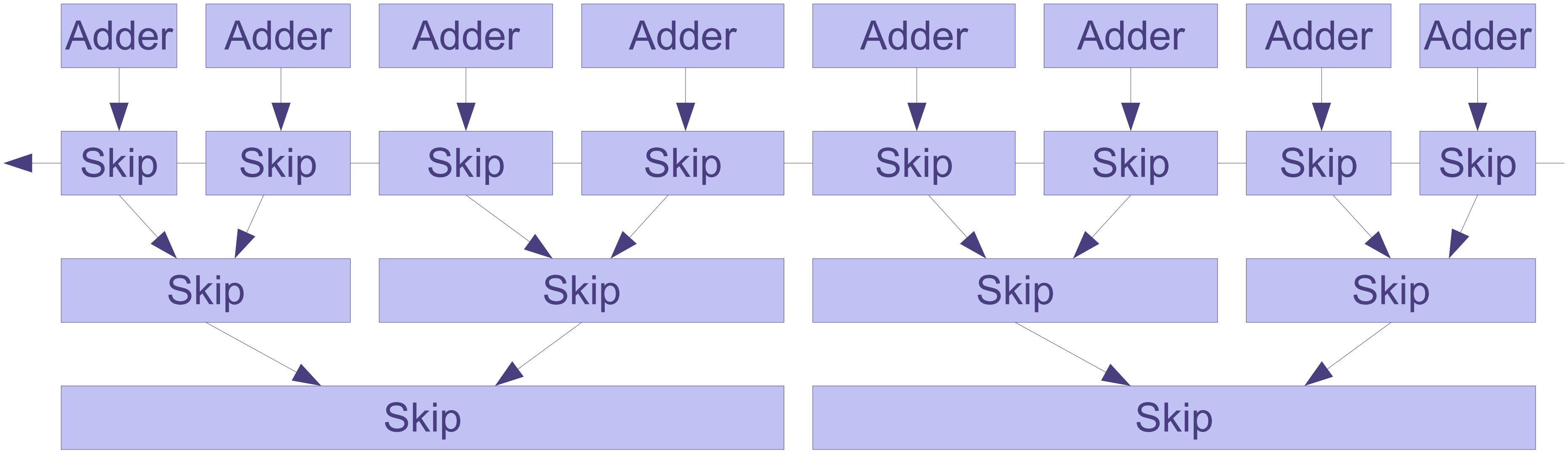
CSKA - Results

Examples (FA dalay - 1 unit of time):

- fixed-size (b) blocks:
  - 16-bit adder → $b_{opt} \approx 3$, $T_{fixCSKA} \approx 8$
  - 32-bit adder → $b_{opt} = 4$, $T_{fixCSKA} = 12.5$
  - 64-bit adder → $b_{opt} \approx 6$, $T_{fixCSKA} \approx 19$
- (t) variable-size blocks ($b_{opt} = 1$):
  - 16-bit adder → $t_{opt} = 8$, $T_{fixCSKA} \approx 5.5$
  - 32-bit adder → $t_{opt} \approx 12$, $T_{fixCSKA} \approx 9$
  - 64-bit adder → $t_{opt} = 16$, $T_{fixCSKA} \approx 13.5$

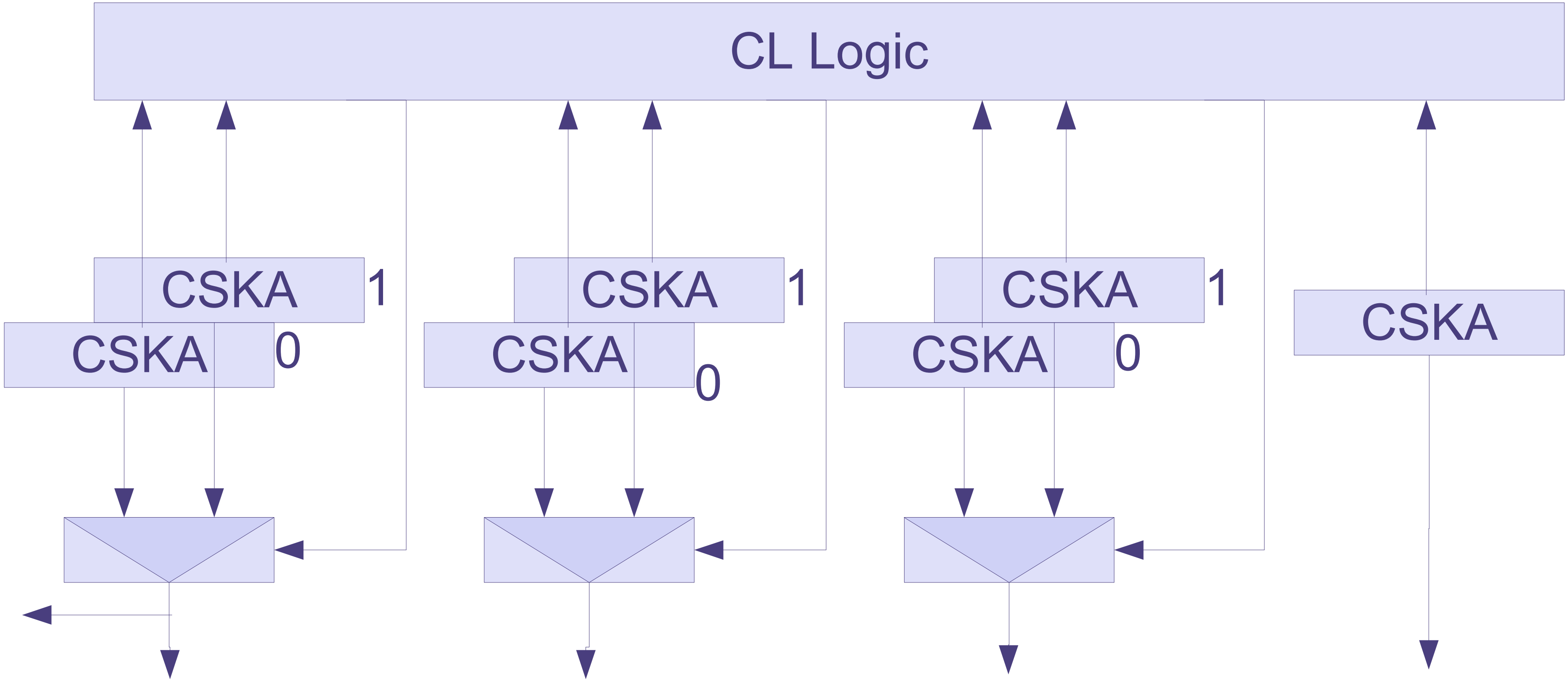
Multilevel CSKA

- First-level skip blocks get propagate-condition signal from block adders, second-level from first-level, etc.
- Carry can be propagated over longer distances, to shorten the worst-case propagation time



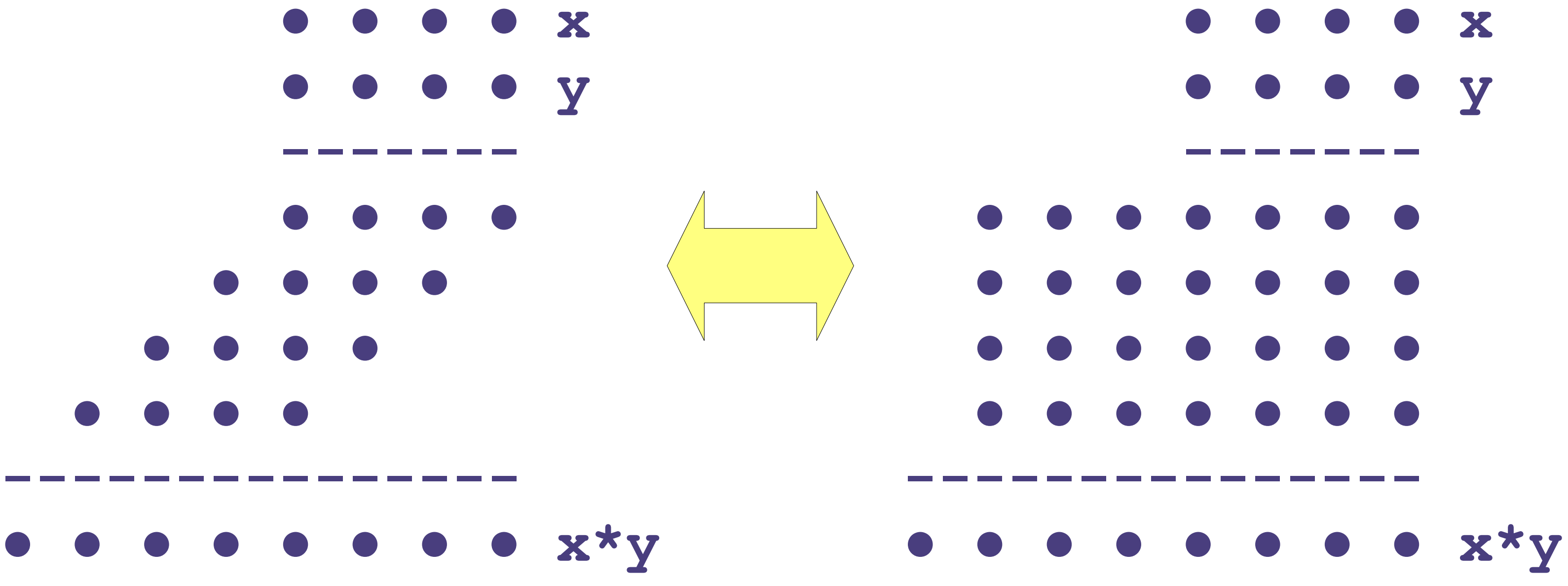
Hybrid Architectures

- example: CSLA + CLA + CSKA



Multioperand Adders

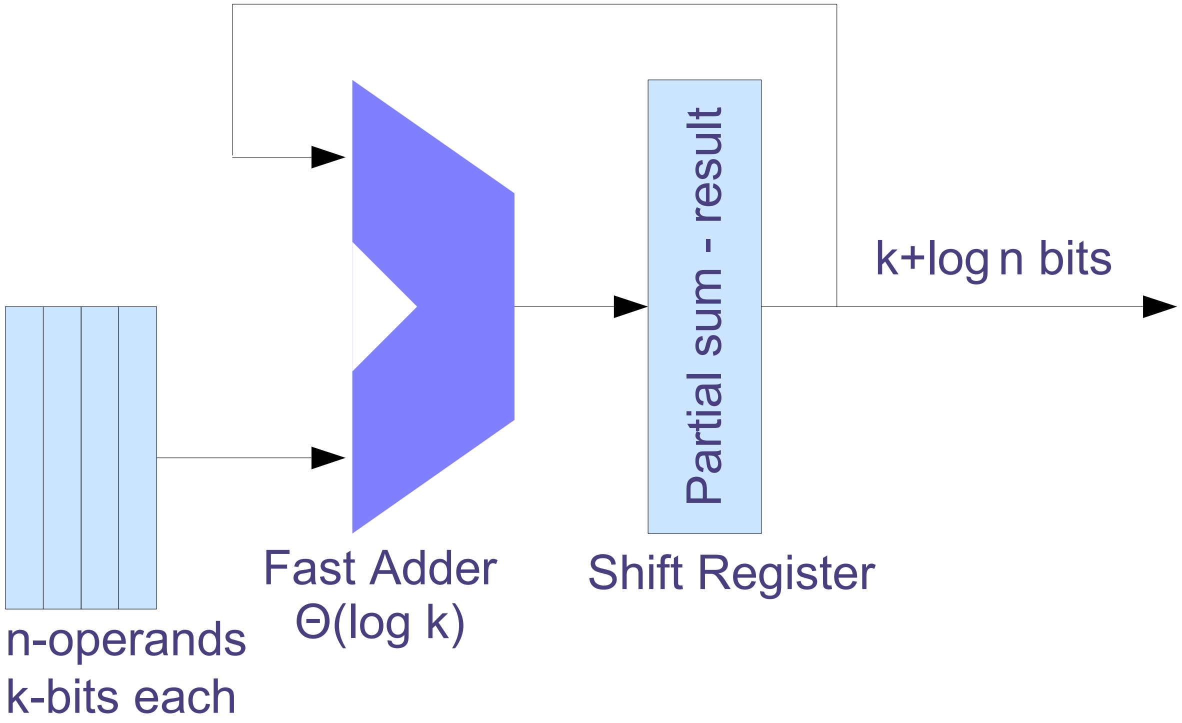
- Applications in multiplication, vector and matrix arithmetics and others



Adding n-numbers of k-bits, total sum has k+$\log_2$n bits

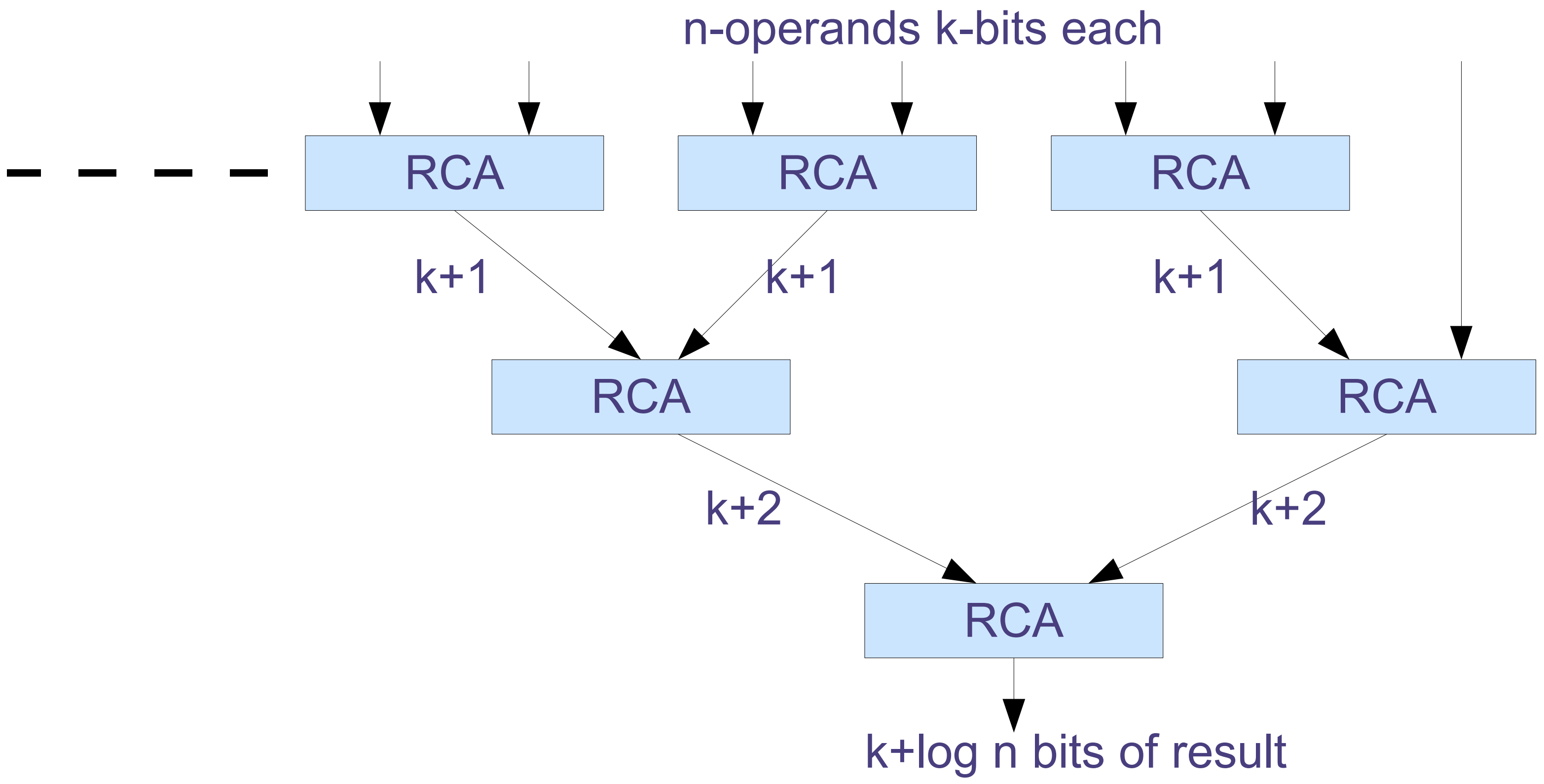
Serial Implementation

- latency: Θ(n log k)
  - faster than linear dependence on number of operands
  - logarithmic dependence for scaling the operand size



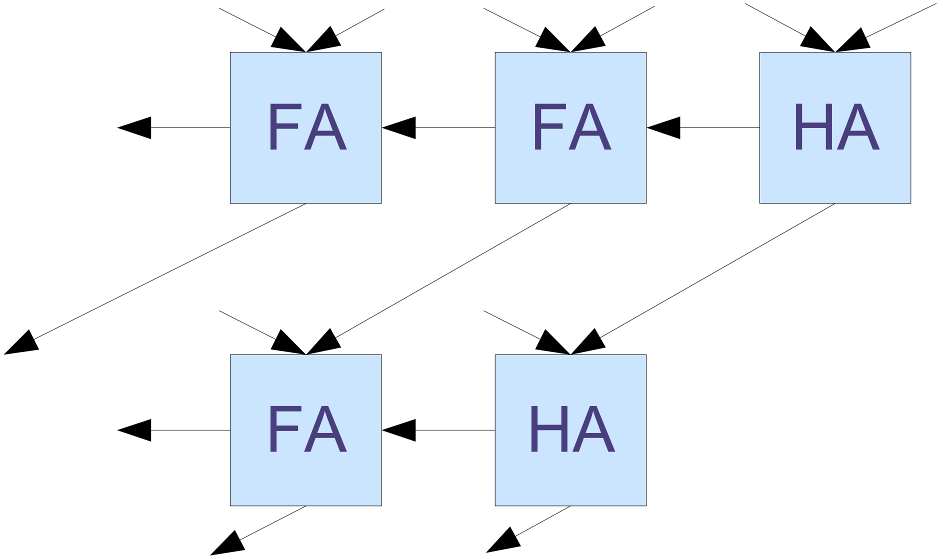
Tree Implementation with CPA (2→1)

- Tree of 2-operand adders (RCA are the best here!)
- For n-operands, n-1 adders are needed (costly)
- Latency Θ(k+log n) – scales well with n



Look into Tree of RCAs

- Adders at higher levels need not wait for full carry propagation from lower-level adders
- All adders start with just one FA-delay after previous level

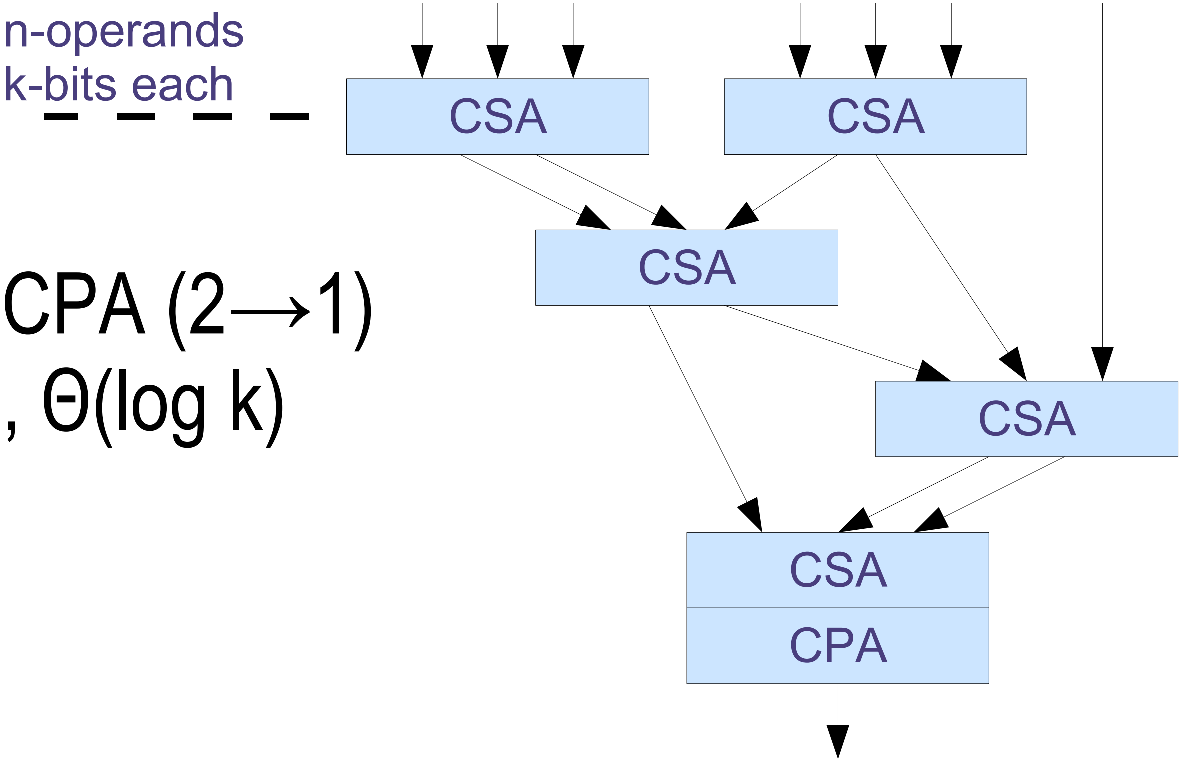


single RCA at level i

single RCA at level i+1

Tree Implementation with CSA (3→2)

- Tree of 3-operand carry-save adders
- CSA reduce n-operands to 2-operands, Θ(log n)
- Latency Θ(log n + log k) – scales well with n & k



- Final fast CPA (2→1) is needed, Θ(log k)

Computer Architecture, IFE CS and T&CS, 4th sem

Redundant Number Systems

- Conventional radix-r systems use [0,r-1] digit set
  - radix-10 → 0,1,2,3,4,5,6,7,8,9
- If the digit set (in radix-r system) contains more than r digits, the system is redundant
  - radix-2 → 0,1,2 or -1,0,1
  - radix-10 → 0,1,2,3,4,5,6,7,8,9,10,11,12,13
  - radix-10 → 0,1,2,3,4,5,6,7,8,9,♠,♣,♥,♦
  - radix-10 → -6,-5,-4,-3,-2,-1,0,1,2,3,4,5
- Conversions between Redundant Number Systems is a simple digit-serial process

Redundancy

- Redundancy may result from adopting the digit set wider than the radix
- The number interpretation is conventional (positional)
- Representation of numbers is not unique
- Physical representation of redundant numbers may be not trivial

Conversions

Conversion is a carry-propagate (slow) process - Θ(k)

	11	9	17	10	12	18	radix-10, digit set [0,18]
	11	9	17	10	12	**18**	18 = 10+8
	11	9	17	10	**13**	8	13 = 10+3
	11	9	17	**11**	3	8	11 = 10+1
	11	9	**18**	1	3	8	18 = 10+8
	11	**10**	8	1	3	8	10 = 10+0
	12	0	8	1	3	8	12 = 10+2
1	2	0	8	1	3	8	radix-10, digit set [0,9]

	11	9	17	10	12	18	radix-10, digit set [0,18]
	11	9	17	10	12	**18**	18 = 20-2
	11	9	17	10	**14**	-2	14 = 10+4
	11	9	17	**11**	4	-2	11 = 10+1
	11	9	**18**	1	4	-2	18 = 20-2
	11	**11**	-2	1	4	-2	11 = 10+1
	12	1	-2	1	4	-2	12 = 10+2
1	2	1	-2	1	4	-2	radix-10, digit set [-6,5]

Decomposition

- Redundant numbers with [0,m] digit set can be represented by two numbers of [0,k] digit sets, where m=2k
- Conversion requires ordinary addition of two such numbers with [0,k] digit set representation

	11	9	17	10	12	18	radix-10, digit set [0,18]
	9	9	9	9	9	9	
+	2	0	8	1	3	9	
1	2	0	8	1	3	8	radix-10, digit set [0,9]

- Decomposed representation is, of course, not unique, but the sum amounts to the correct result

Computer Architecture, IFE CS and T&CS, 4th sem

Radix-2 [0,2] digit set numbers (BSC)

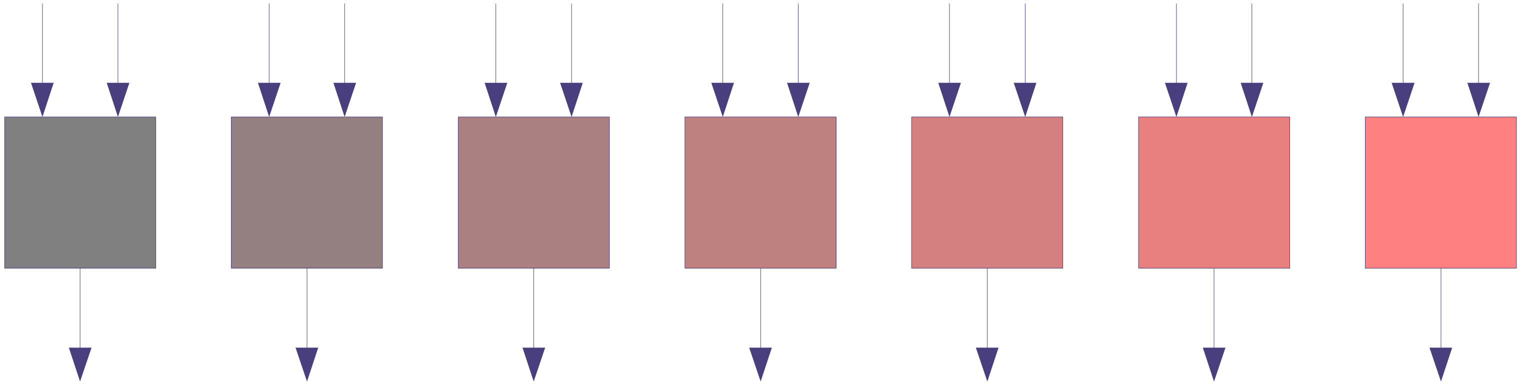
- Redundant binary numbers may be coded with bit-fields, e.g:
  0: (0,0),
  1: (0,1) or (1,0),
  2: (1,1)
- Decomposed form is convenient and efficient representation of redundant binary numbers

	1	1	2	0	2	0	radix-2, digit set [0,2]
	1	1	1	0	1	0	
+	0	0	1	0	1	0	
1	0	0	0	1	0	0	radix-2, digit set [0,1]

Carry-Free Addition (Naive)

- Carry-free → no carry propagation, all digit additions can be done simultaneously!!! - Θ(1)
- Carry-free addition is possible with widening the digit set for the result (carry is always absorbed)

1	2	3	4	5	6	radix-10, digit set [0,9]
4	5	6	7	8	9	radix-10, digit set [0,9]
5	7	9	11	13	15	radix-10, digit set [0,18]



Computer Architecture, IFE CS and T&CS, 4th sem

Carry-Free Addition

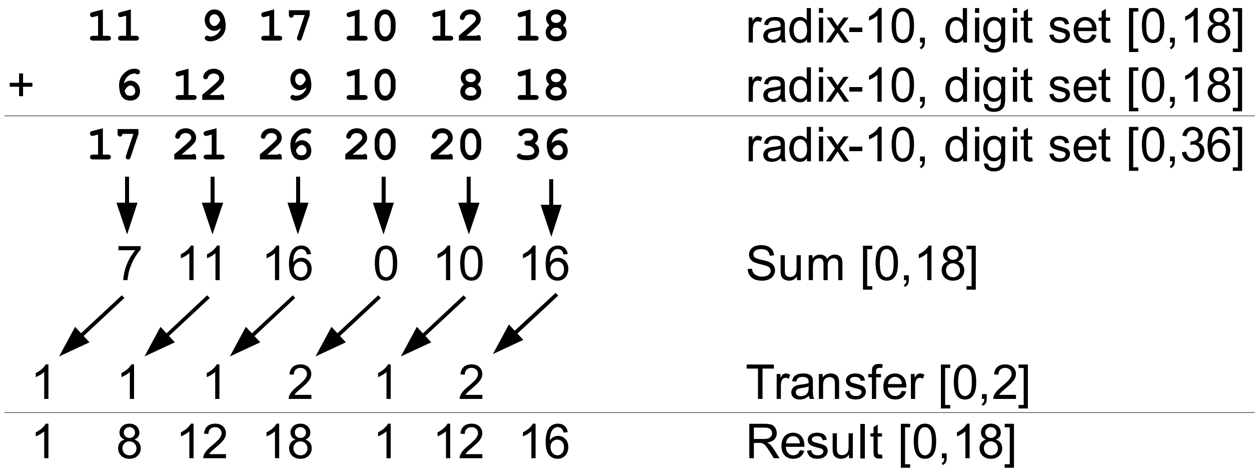
- Carry-free → no carry propagation, all digit additions can be done simultaneously!!!
- The digit set of the result must be the same as digit set of operands (the same redundant number system)

```
    11   9  17  10  12  18          radix-10, digit set [0,18]
+        6  12   9  10   8  18          radix-10, digit set [0,18]
----------------------------------------------------------------
 1   8  12  18   1  12  16          Result, radix-10, [0,18]
```

- Is this always possible and how?
- Is is still a carry-free operation?

Limited-Carry Propagation

- The result must be decomposed into:
  - sum
  - transfer digits
- Transfer digits must be absorbed by sum digits
- Limited carry propagation is a parallel process - Θ(1)

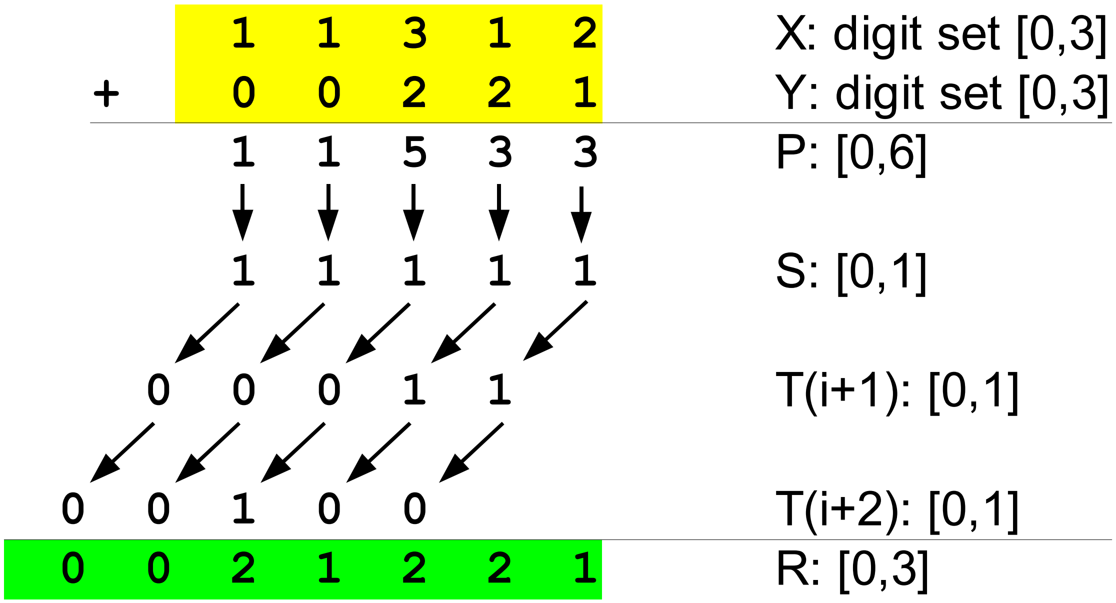


```
      11   9  17  10  12  18      radix-10, digit set [0,18]
+      6  12   9  10   8  18      radix-10, digit set [0,18]
      17  21  26  20  20  36      radix-10, digit set [0,36]
       7  11  16   0  10  16      Sum [0,18]
   1   1   1   2   1   2          Transfer [0,2]
   1   8  12  18   1  12  16      Result [0,18]
```

Computer Architecture, IFE CS and T&CS, 4th sem

Limited-Carry Algorithm

- If redundancy is low (<3) or radix is small (<3), the absorption requires decomposition into a sum and two transfer digits
- Limited-carry addition algorithm can be implemented for all redundant DSG numbers with digit set [−α,β]



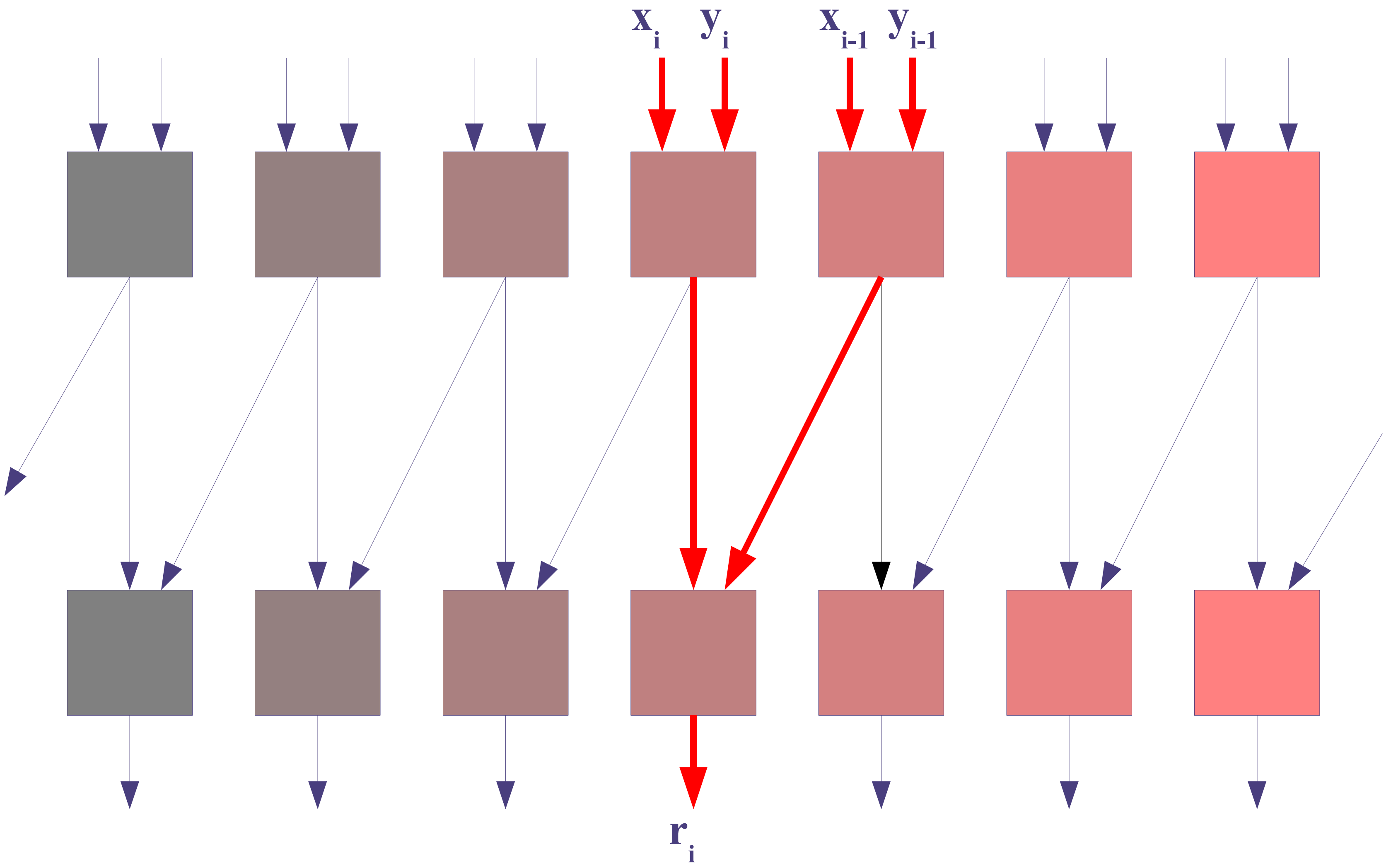
```
         1  1  3  1  2     X: digit set [0,3]
 +       0  0  2  2  1     Y: digit set [0,3]
         1  1  5  3  3     P: [0,6]
         1  1  1  1  1     S: [0,1]
      0  0  0  1  1        T(i+1): [0,1]
   0  0  1  0  0           T(i+2): [0,1]
   0  0  2  1  2  2  1     R: [0,3]
```

R=X+Y: [0,3], r=2, ρ=2

p_i	t_{i+2}	t_{i+1}	s_i
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0

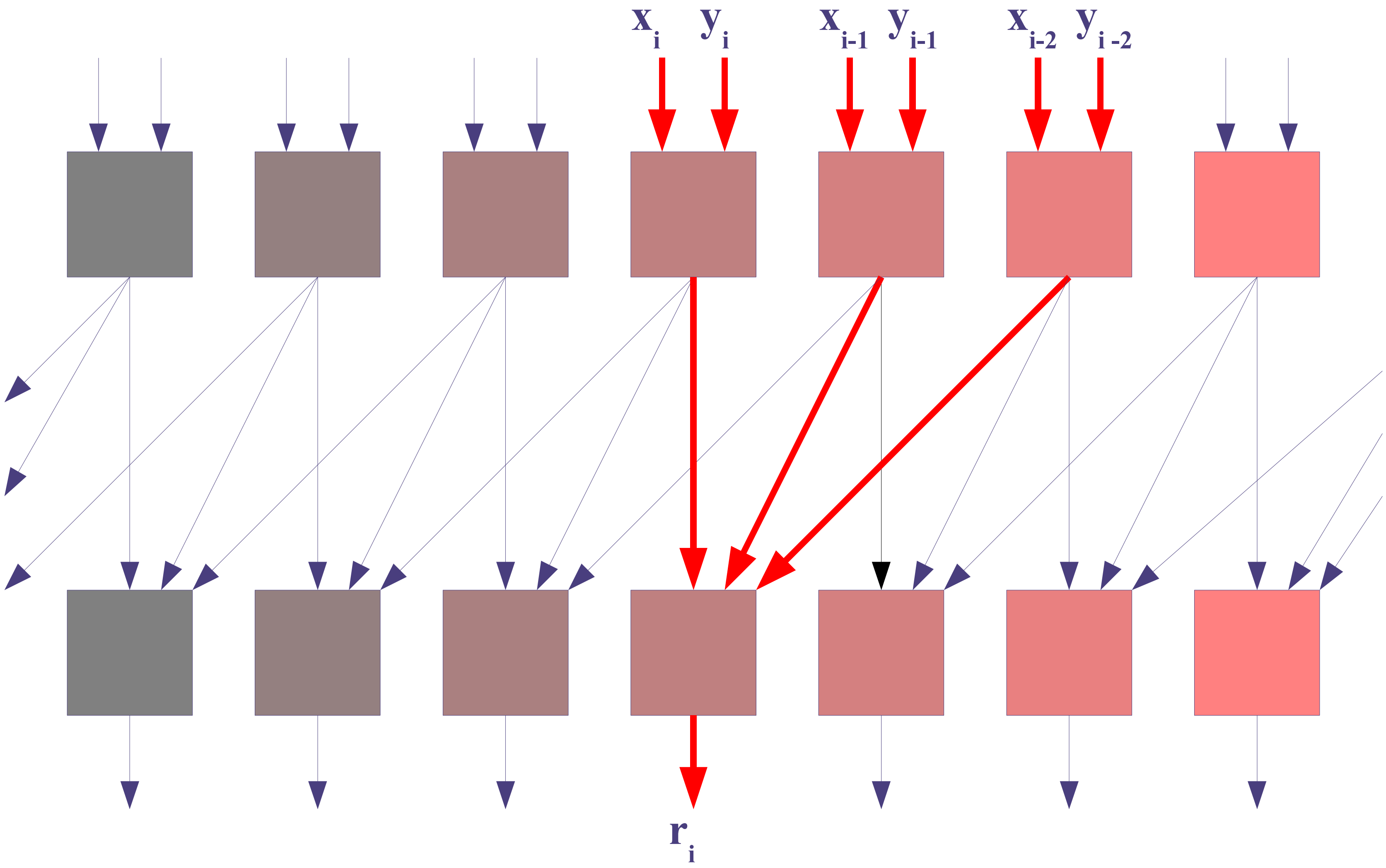
Two-stage Parallel-Carry Hardware

- High redundancy and radix

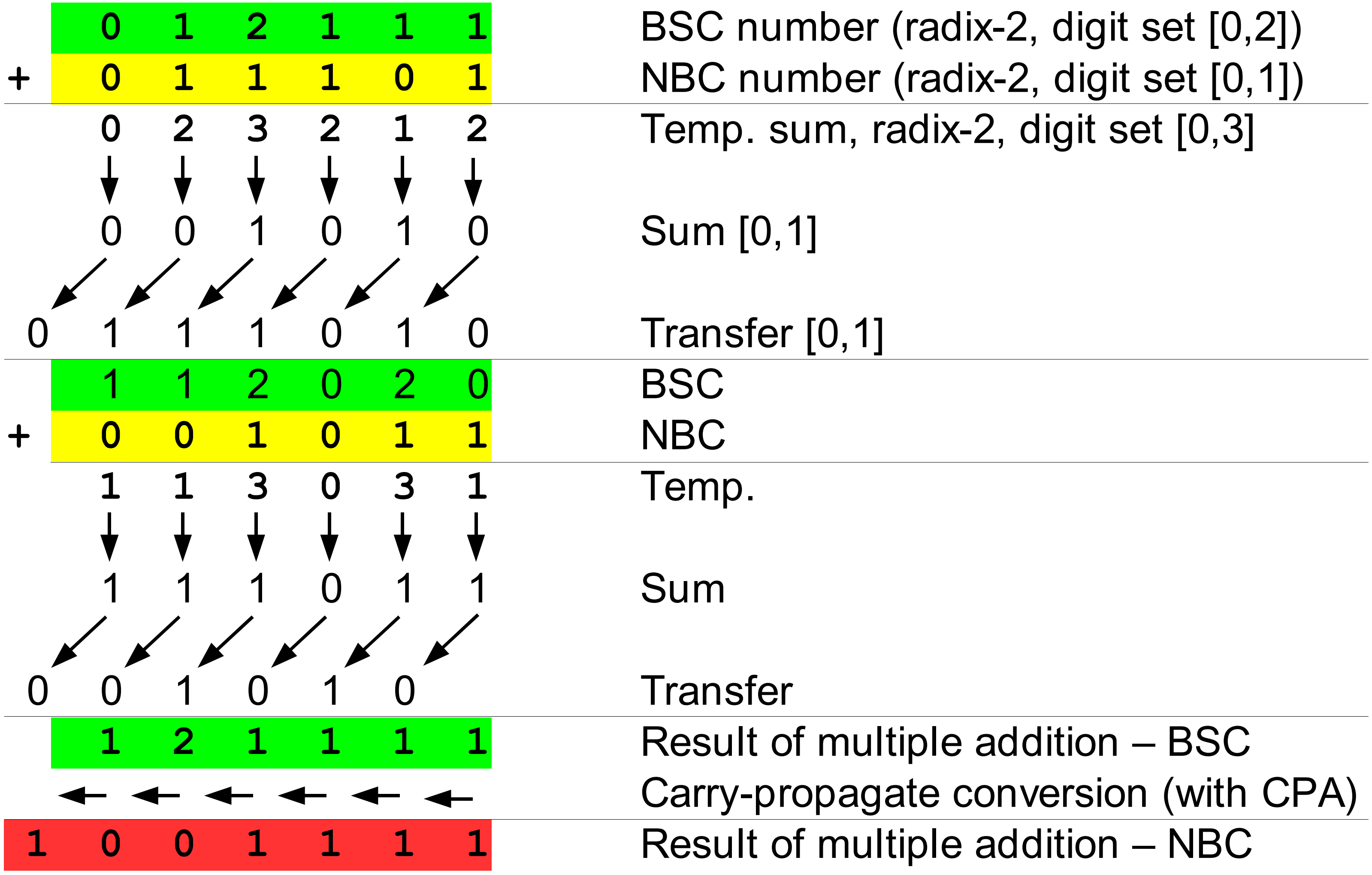


Two-stage Parallel-Carry Hardware

- Low redundancy or small radix



Redundant Binary Addition with BSC



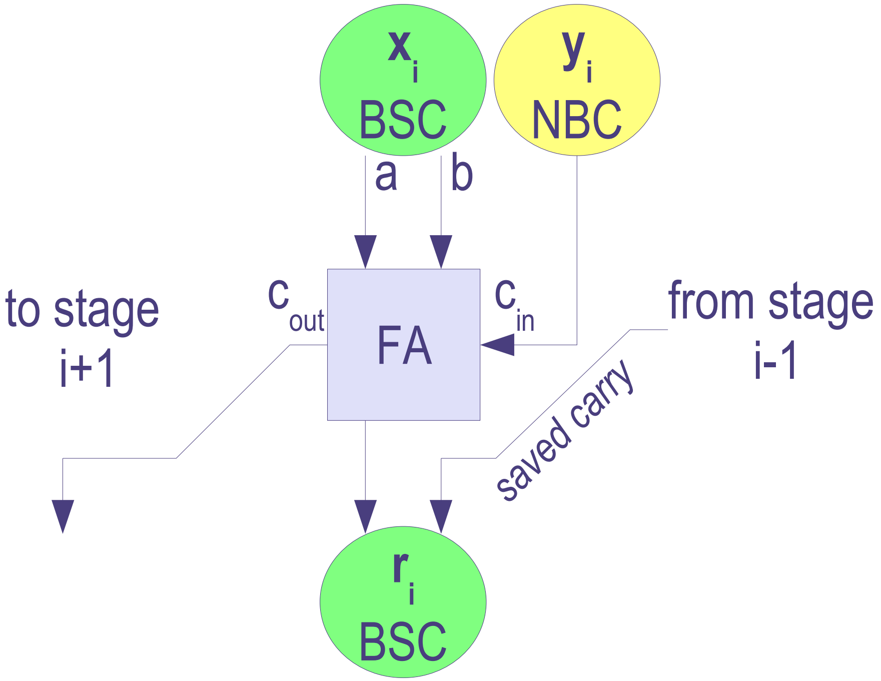
		0	1	2	1	1	1	BSC number (radix-2, digit set [0,2])
+		0	1	1	1	0	1	NBC number (radix-2, digit set [0,1])
		0	2	3	2	1	2	Temp. sum, radix-2, digit set [0,3]
		↓	↓	↓	↓	↓	↓	
		0	0	1	0	1	0	Sum [0,1]
	↙	↙	↙	↙	↙	↙		
	0	1	1	1	0	1	0	Transfer [0,1]
		1	1	2	0	2	0	BSC
+		0	0	1	0	1	1	NBC
		1	1	3	0	3	1	Temp.
		↓	↓	↓	↓	↓	↓	
		1	1	1	0	1	1	Sum
	↙	↙	↙	↙	↙	↙		
	0	0	1	0	1	0		Transfer
		1	2	1	1	1	1	Result of multiple addition – BSC
	←	←	←	←	←	←		Carry-propagate conversion (with CPA)
	1	0	0	1	1	1	1	Result of multiple addition – NBC

$$X_{BSC} + C_{NBC} = A_{NBC} + B_{NBC} + C_{NBC} = R_{BSC} = S_{NBC} + T_{NBC}$$

reduction $3_{NBC} \longrightarrow 2_{NBC}$

1-bit Carry Save Adder (CSA)

- FA in 3→2 configuration
- Carry is saved
  in next stage
  transfer register
- 3/2 reduction circuit

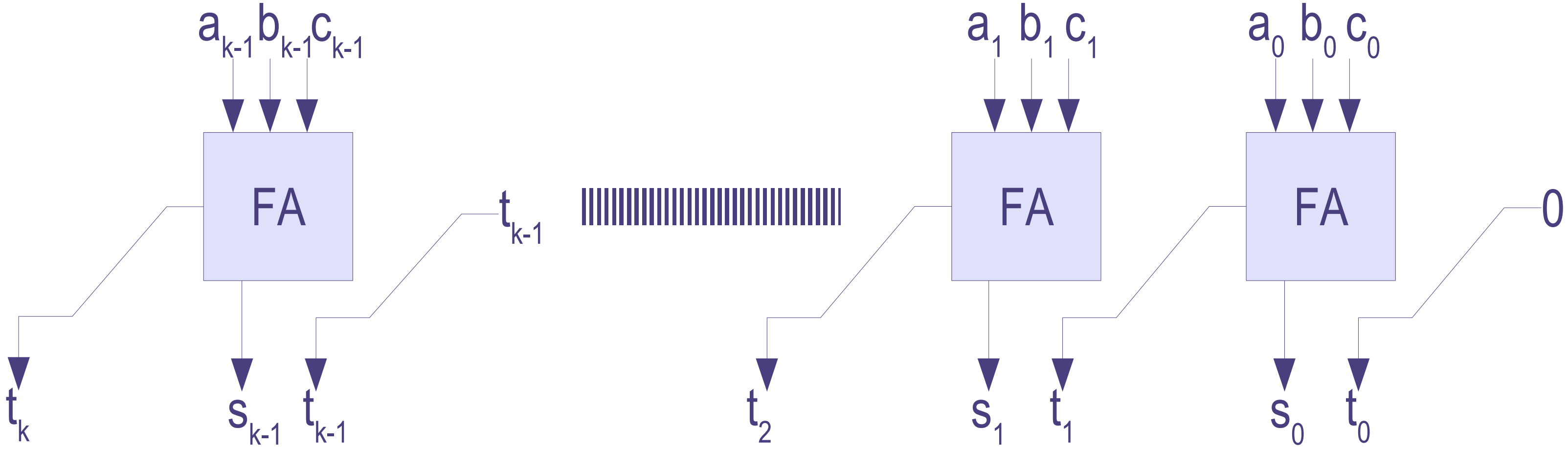


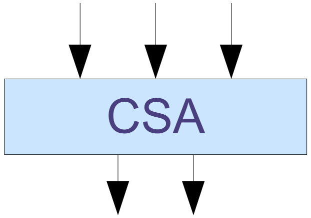
	1	1	2	0	2	0	BSC
+	0	0	1	0	1	1	NBC
	1	1	3	0	3	1	Temp.
	1	1	1	0	1	1	Sum
0	0	1	0	1	0		Transfer
	1	2	1	1	1	1	BSC

Computer Architecture, IFE CS and T&CS, 4th sem

Carry Save Adder (CSA)

- Latency Θ(1)
- Length of operands does not matter

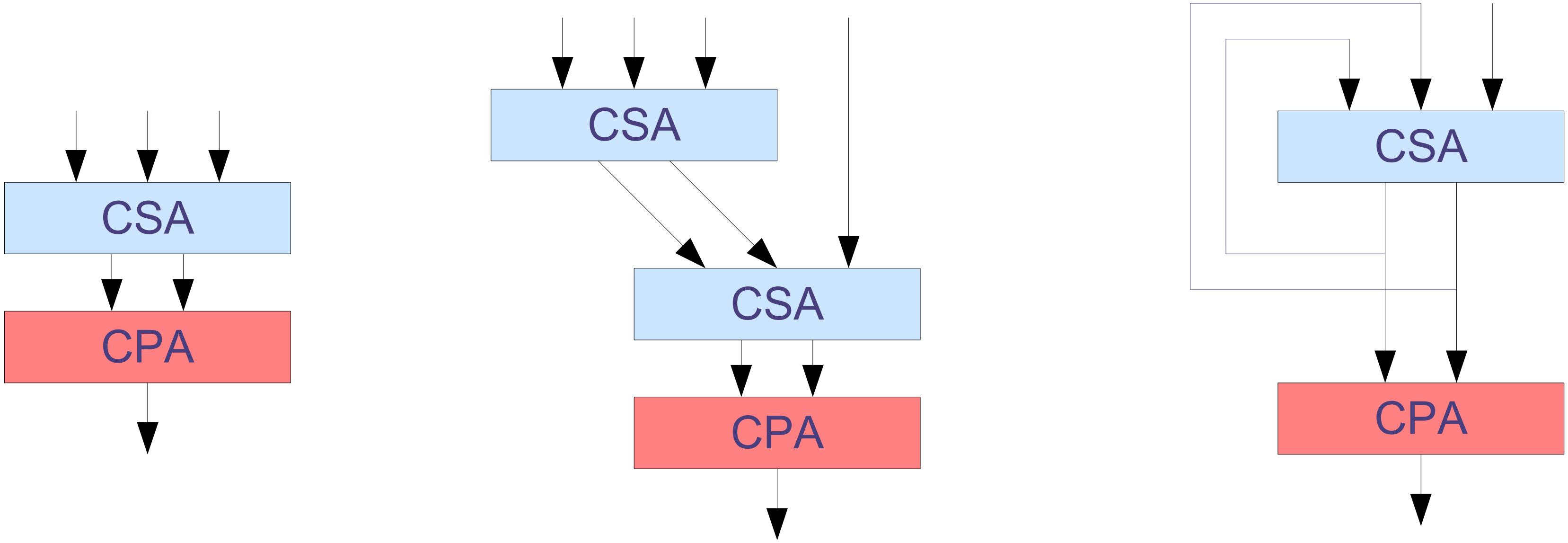




Multioperand Addition with CSA

- Final result needs CPA conversion - Θ(k)
- Tree implementations Θ(log n + log k)
- Sequential implementations Θ(n + log k)

(n-operands, k-bits)


CSA
CPA
CSA
CSA
CPA
CSA
CPA