Software Verification with Satisfiability Modulo Theories
- Verify Cryptographic Software -

Ming-Hsien Tsai

Institute of Information Science
Academia Sinica

FLOLAC 2017

Verifying Curve25519 Software

[CCS'14]

- Formal verification of the central hand-optimized assembly routine (ladderstep) of Curve25519 Diffie-Hellman key-exchange software [Ber06]
  - Two implementations [BDL+11] written in qhasm [Ber] (~1.5K LOC)
  - Speed-record holder
  - Reproduce a bug previously found by the developers
  - Verified the corrected version

a joint work with Yu-Fang Chen *et al.*

Applications of Curve25519

- OpenSSH
- iOS
- Apple HomeKit
- Tor
- ...

Ladderstep

Algorithm 2 Single Curve25519 Montgomery Ladderstep

function LADDERSTEP$(X_1, X_2, Z_2, X_3, Z_3)$

$T_1 \leftarrow X_2 + Z_2$
$T_2 \leftarrow X_2 - Z_2$
$T_7 \leftarrow T_2^2$
$T_6 \leftarrow T_1^2$
$T_5 \leftarrow T_6 - T_7$
$T_3 \leftarrow X_3 + Z_3$
$T_4 \leftarrow X_3 - Z_3$
$T_9 \leftarrow T_3 \cdot T_2$
$T_8 \leftarrow T_4 \cdot T_1$
$X_3 \leftarrow (T_8 + T_9)$
$Z_3 \leftarrow (T_8 - T_9)$

$X_3 \leftarrow X_3^2$
$Z_3 \leftarrow Z_3^2$
$Z_3 \leftarrow Z_3 \cdot X_1$
$X_2 \leftarrow T_6 \cdot T_7$
$Z_2 \leftarrow 121666 \cdot T_5$
$Z_2 \leftarrow Z_2 + T_7$
$Z_2 \leftarrow Z_2 \cdot T_5$
return (X_2, Z_2, X_3, Z_3)

end function

Ladderstep

Algorithm 2 Single Curve25519 Montgomery Ladderstep

function LADDERSTEP(X_1, X_2, Z_2, X_3, Z_3)

$T_1 \leftarrow X_2 + Z_2$
$T_2 \leftarrow X_2 - Z_2$
$T_7 \leftarrow T_2^2$
$T_6 \leftarrow T_1^2$
$T_5 \leftarrow T_6 - T_7$
$T_3 \leftarrow X_3 + Z_3$
$T_4 \leftarrow X_3 - Z_3$
$T_9 \leftarrow T_3 \cdot T_2$
$T_8 \leftarrow T_4 \cdot T_1$
$X_3 \leftarrow (T_8 + T_9)$
$Z_3 \leftarrow (T_8 - T_9)$

$X_3 \leftarrow X_3^2$
$Z_3 \leftarrow Z_3^2$
$Z_3 \leftarrow Z_3 \cdot X_1$
$X_2 \leftarrow T_6 \cdot T_7$
$Z_2 \leftarrow 121666 \cdot T_5$
$Z_2 \leftarrow Z_2 + T_7$
$Z_2 \leftarrow Z_2 \cdot T_5$
return (X_2, Z_2, X_3, Z_3)

end function

Arithmetic operations in F_p ($p = 2^{255}$ - 19)

Ladderstep

Algorithm 2 Single Curve25519 Montgomery Ladderstep

function LADDERSTEP$(X_1, X_2, Z_2, X_3, Z_3)$

255-bit variables

$$
\begin{aligned}
T_1 &\leftarrow X_2 + Z_2 \\
T_2 &\leftarrow X_2 - Z_2 \\
T_7 &\leftarrow T_2^2 \\
T_6 &\leftarrow T_1^2 \\
T_5 &\leftarrow T_6 - T_7 \\
T_3 &\leftarrow X_3 + Z_3 \\
T_4 &\leftarrow X_3 - Z_3 \\
T_9 &\leftarrow T_3 \cdot T_2 \\
T_8 &\leftarrow T_4 \cdot T_1 \\
X_3 &\leftarrow (T_8 + T_9) \\
Z_3 &\leftarrow (T_8 - T_9)
\end{aligned}
$$

$$
\begin{aligned}
X_3 &\leftarrow X_3^2 \\
Z_3 &\leftarrow Z_3^2 \\
Z_3 &\leftarrow Z_3 \cdot X_1 \\
X_2 &\leftarrow T_6 \cdot T_7 \\
Z_2 &\leftarrow 121666 \cdot T_5 \\
Z_2 &\leftarrow Z_2 + T_7 \\
Z_2 &\leftarrow Z_2 \cdot T_5
\end{aligned}
$$

return (X_2, Z_2, X_3, Z_3)

end function

Arithmetic operations in F_p $(p = 2^{255}$ - $19)$

Ladderstep

Algorithm 2 Single Curve25519 Montgomery Ladderstep

function LADDERSTEP$(X_1, X_2, Z_2, X_3, Z_3)$

255-bit variables

$$
\begin{aligned}
T_1 &\leftarrow X_2 + Z_2 \\
T_2 &\leftarrow X_2 - Z_2 \\
T_7 &\leftarrow T_2^2 \\
T_6 &\leftarrow T_1^2 \\
T_5 &\leftarrow T_6 - T_7 \\
T_3 &\leftarrow X_3 + Z_3 \\
T_4 &\leftarrow X_3 - Z_3 \\
&\boxed{T_9 \leftarrow T_3 \cdot T_2} \\
T_8 &\leftarrow T_4 \cdot T_1 \\
X_3 &\leftarrow (T_8 + T_9) \\
Z_3 &\leftarrow (T_8 - T_9)
\end{aligned}
$$

$T_9 \equiv T_3 T_2 \ (mod\ p)$

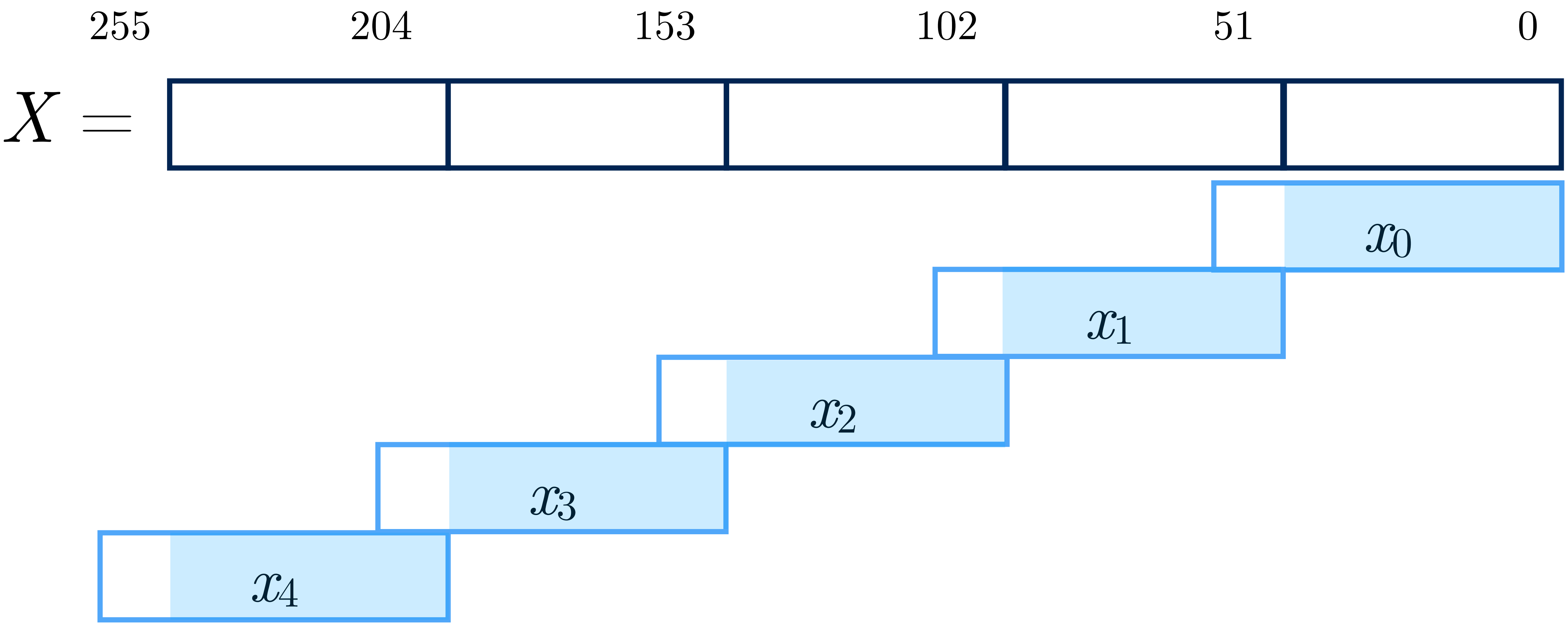
$$
\begin{aligned}
X_3 &\leftarrow X_3^2 \\
Z_3 &\leftarrow Z_3^2 \\
Z_3 &\leftarrow Z_3 \cdot X_1 \\
X_2 &\leftarrow T_6 \cdot T_7 \\
Z_2 &\leftarrow 121666 \cdot T_5 \\
Z_2 &\leftarrow Z_2 + T_7 \\
Z_2 &\leftarrow Z_2 \cdot T_5
\end{aligned}
$$

return (X_2, Z_2, X_3, Z_3)

end function

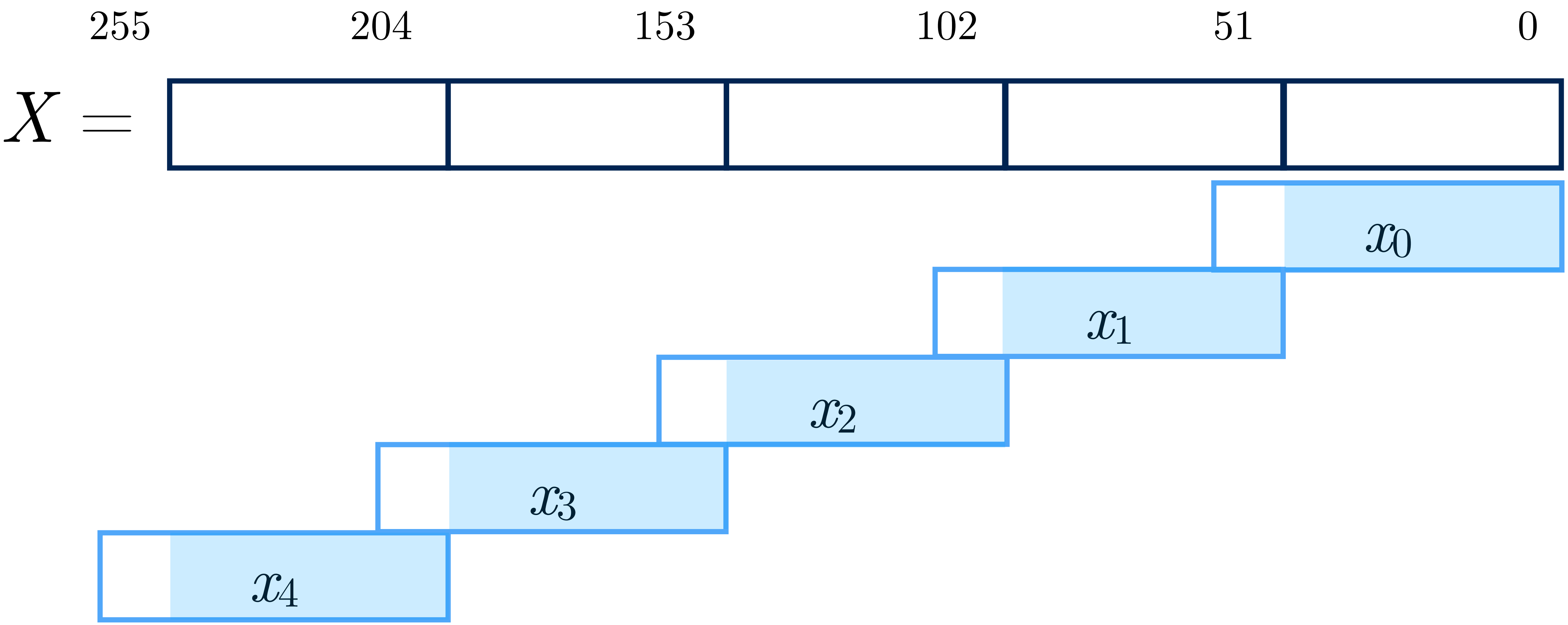
Arithmetic operations in F_p $(p = 2^{255}$ - $19)$

Radix-2^{51} Representation



a rectangle with 51 bits is called a limb

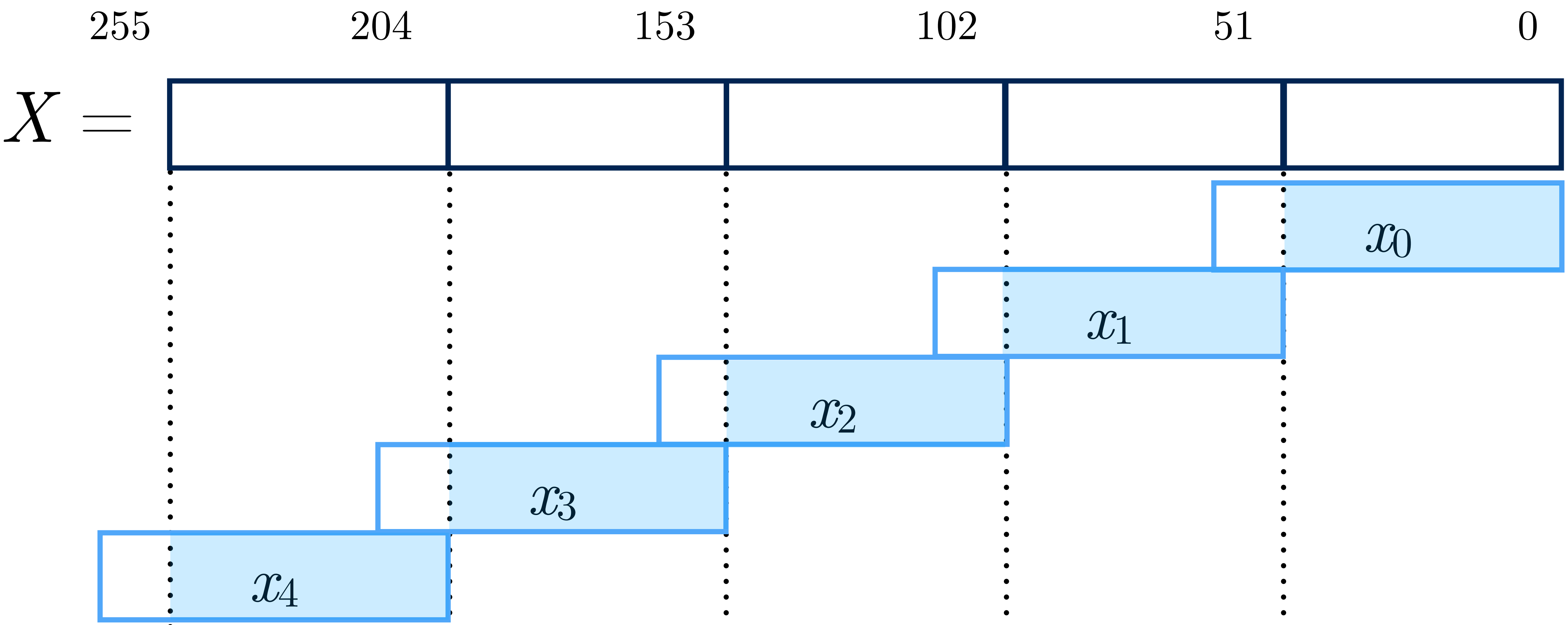
Radix-2^{51} Representation



$$X = x_4 2^{204} + x_3 2^{153} + x_2 2^{102} + x_1 2^{51} + x_0$$

a rectangle with 51 bits is called a limb

Radix-2^{51} Representation



$$X = x_4 2^{204} + x_3 2^{153} + x_2 2^{102} + x_1 2^{51} + x_0$$

a rectangle with 51 bits is called a limb

Carries can be stored in the extra bits

Multiplication (Radix-2^{51})

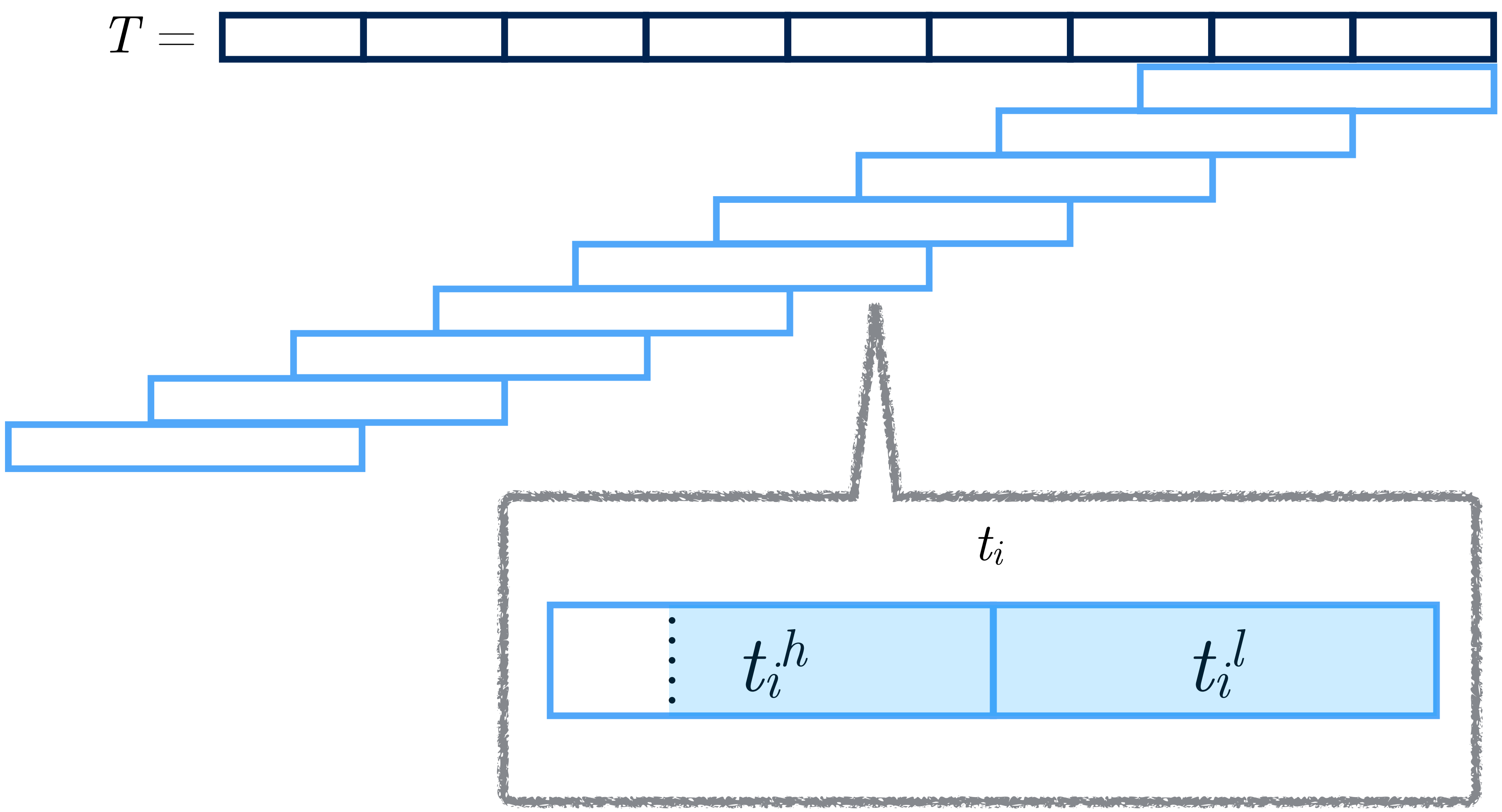
- Compute $R \equiv XY \pmod{2^{255}\text{-}19}$

$$
\begin{aligned}
X &= x_4 2^{204} + x_3 2^{153} + x_2 2^{102} + x_1 2^{51} + x_0 \\
Y &= y_4 2^{204} + y_3 2^{153} + y_2 2^{102} + y_1 2^{51} + y_0 \\
R &= r_4 2^{204} + r_3 2^{153} + r_2 2^{102} + r_1 2^{51} + r_0
\end{aligned}
$$

- The naive approach has three steps
  - Multiply
  - Reduce
  - Delayed carry
- The efficient implementation merges Multiply and Reduce

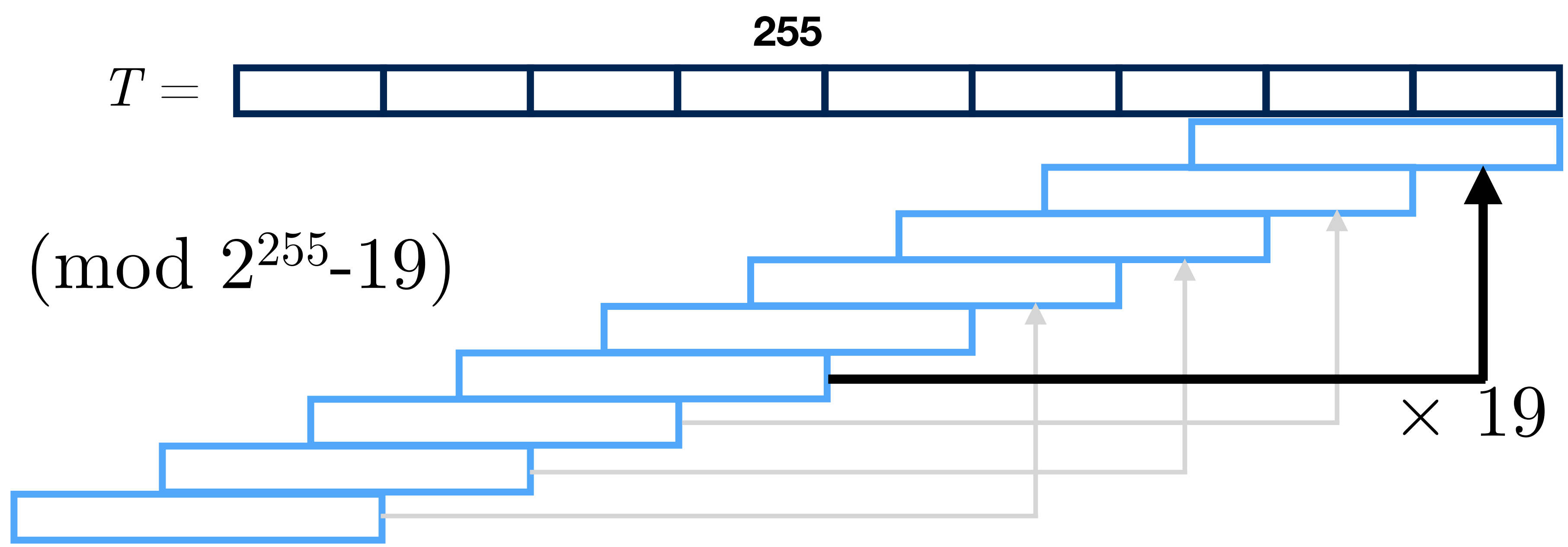
Multiply

- Compute $T = XY$
  - $t_0 = x_0 y_0$
  - $t_1 = x_0 y_1 + x_1 y_0$
  - …
  - $t_8 = x_4 y_4$
- T has 9 limbs
- A limb in T may have more than 64 bits



$$T = t_8 2^{448} + t_7 2^{357} + t_6 2^{306} + t_5 2^{255} + t_4 2^{204} + t_3 2^{153} + t_2 2^{102} + t_1 2^{51} + t_0$$

Reduce



- Compute $S \equiv T \pmod{2^{255}\text{-}19}$
- S has 5 limbs
- A limb in S may have more than 64 bits
- $n2^{255} \equiv 19n \pmod{2^{255}\text{-}19}$

$$T = t_8 2^{448} + t_7 2^{357} + t_6 2^{306} + t_5 2^{255} + t_4 2^{204} + t_3 2^{153} + t_2 2^{102} + t_1 2^{51} + t_0$$

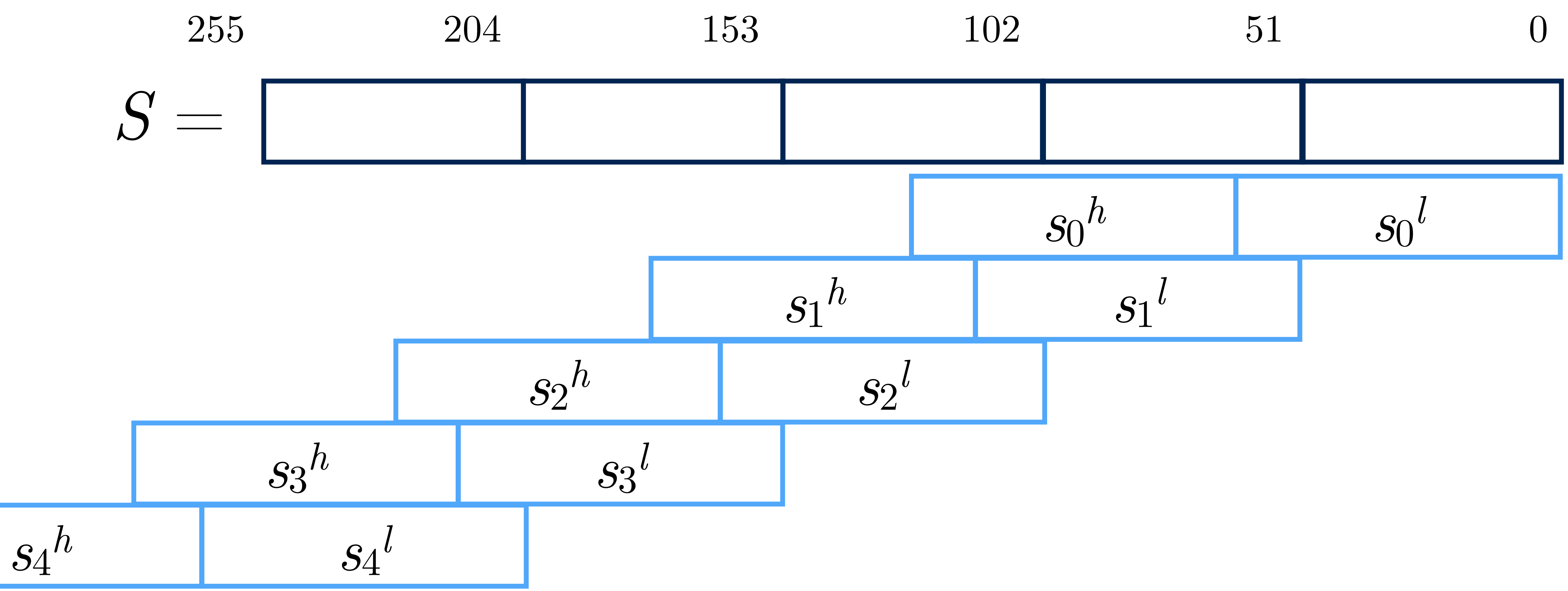
$$S = 19 t_8 2^{153} + 19 t_7 2^{102} + 19 t_6 2^{51} + 19 t_5 + t_4 2^{204} + t_3 2^{153} + t_2 2^{102} + t_1 2^{51} + t_0$$

$$= s_4 2^{204} + s_3 2^{153} + s_2 2^{102} + s_1 2^{51} + s_0$$

Delayed-Carry (Simplified)

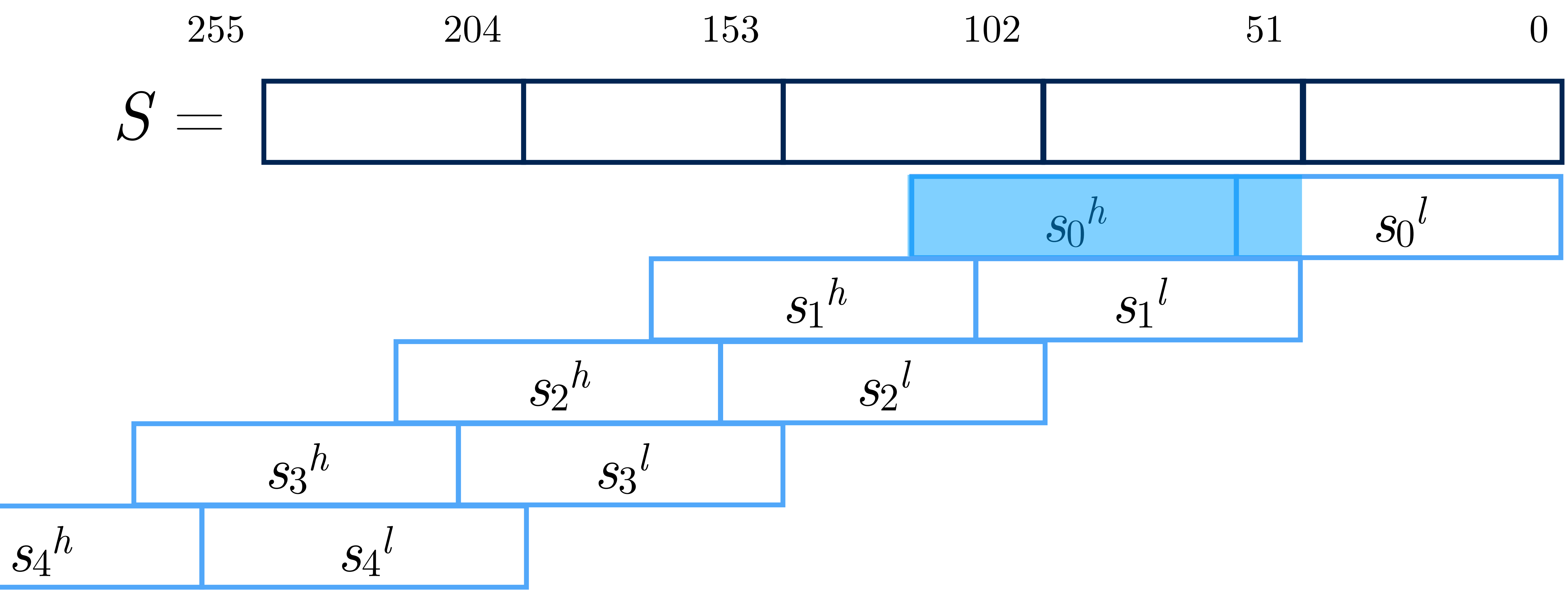
- Compute $R \equiv S \pmod{2^{255}\text{-}19}$

- A limb in R has at most 52 bits



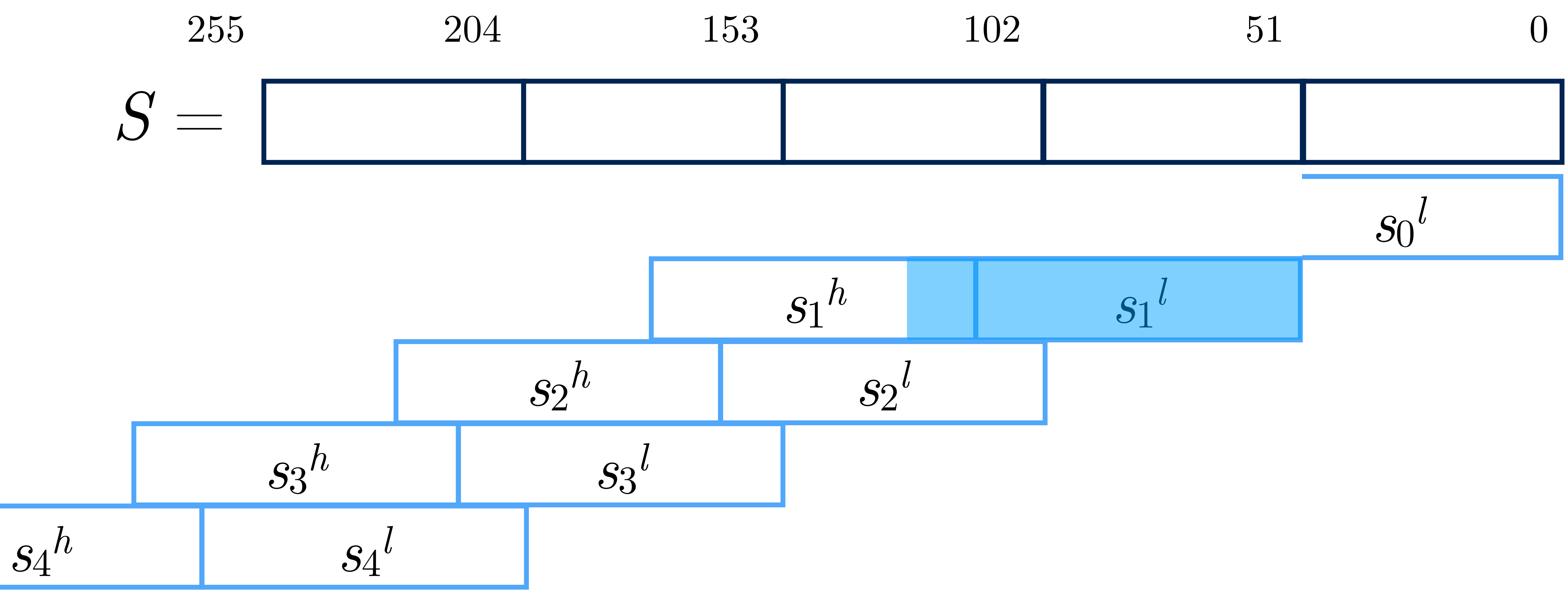
Delayed-Carry (Simplified)

- Compute $R \equiv S \pmod{2^{255}\text{-}19}$
- A limb in R has at most 52 bits



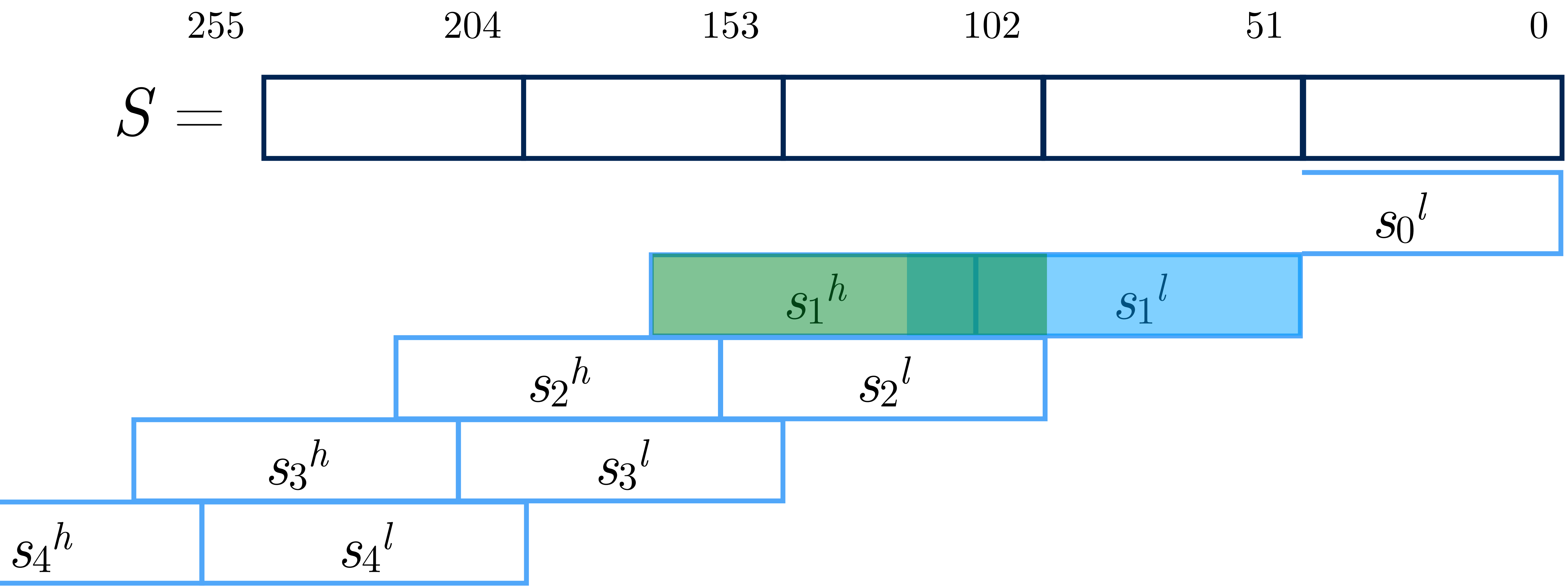
Delayed-Carry (Simplified)

- Compute $R \equiv S \pmod{2^{255}\text{-}19}$
- A limb in R has at most 52 bits



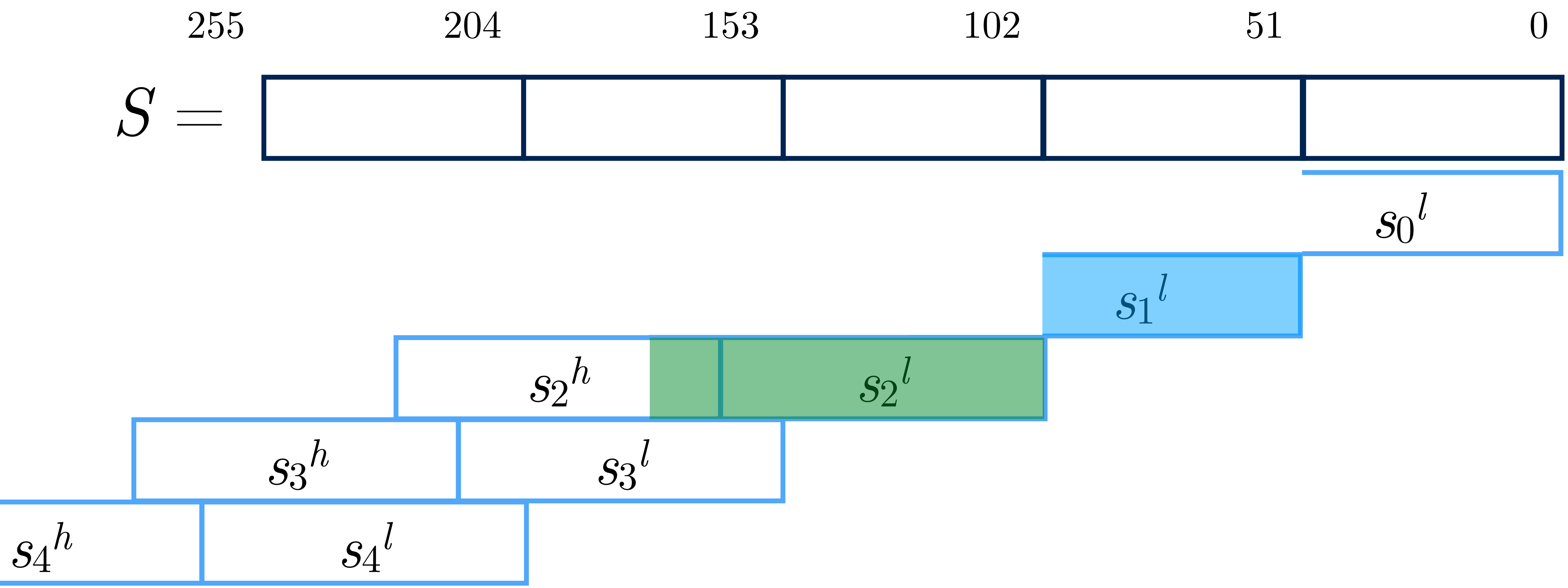
Delayed-Carry (Simplified)

- Compute $R \equiv S \pmod{2^{255}\text{-}19}$
- A limb in R has at most 52 bits



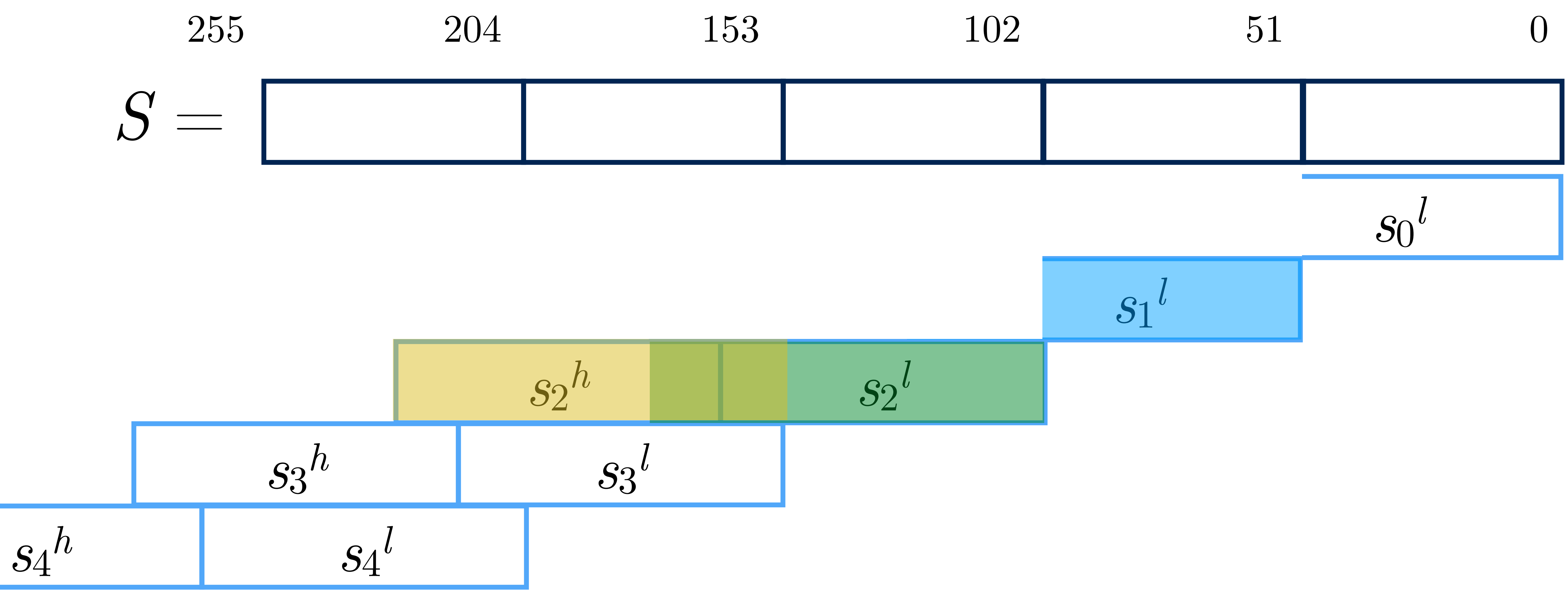
Delayed-Carry (Simplified)

- Compute $R \equiv S \pmod{2^{255}\text{-}19}$
- A limb in R has at most 52 bits



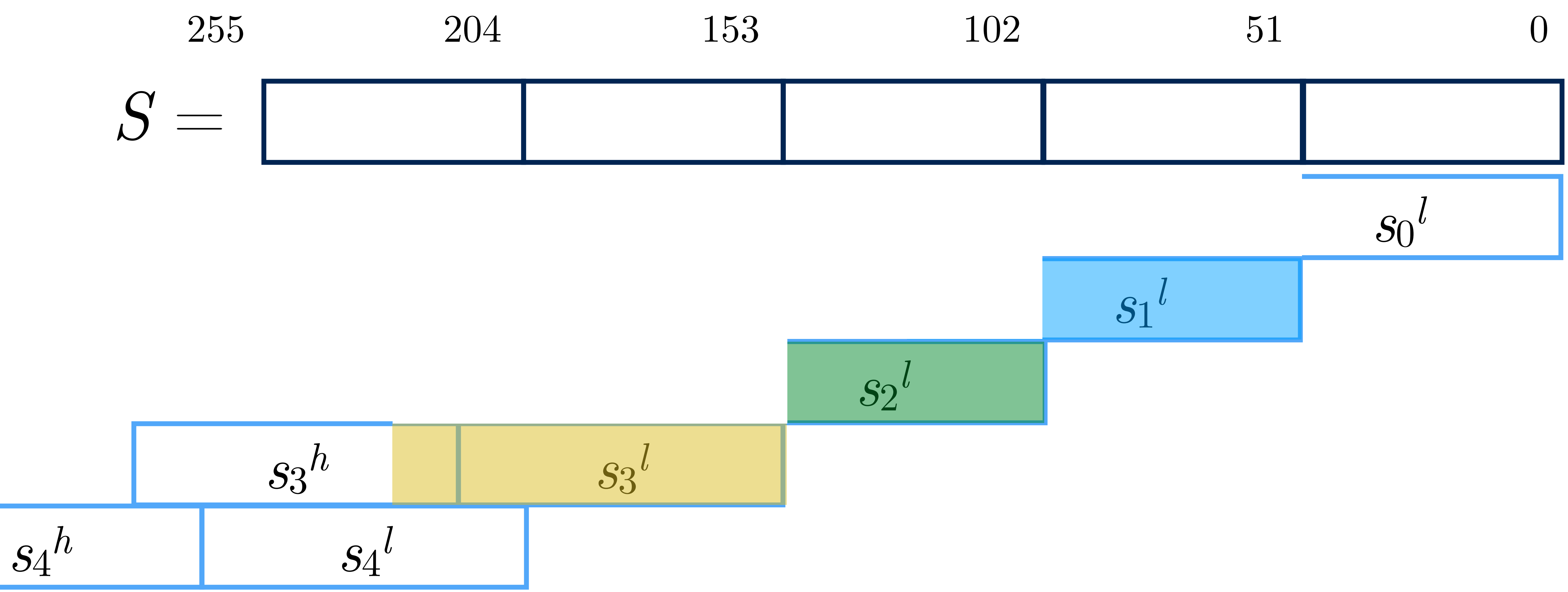
Delayed-Carry (Simplified)

- Compute $R \equiv S \pmod{2^{255}\text{-}19}$
- A limb in R has at most 52 bits



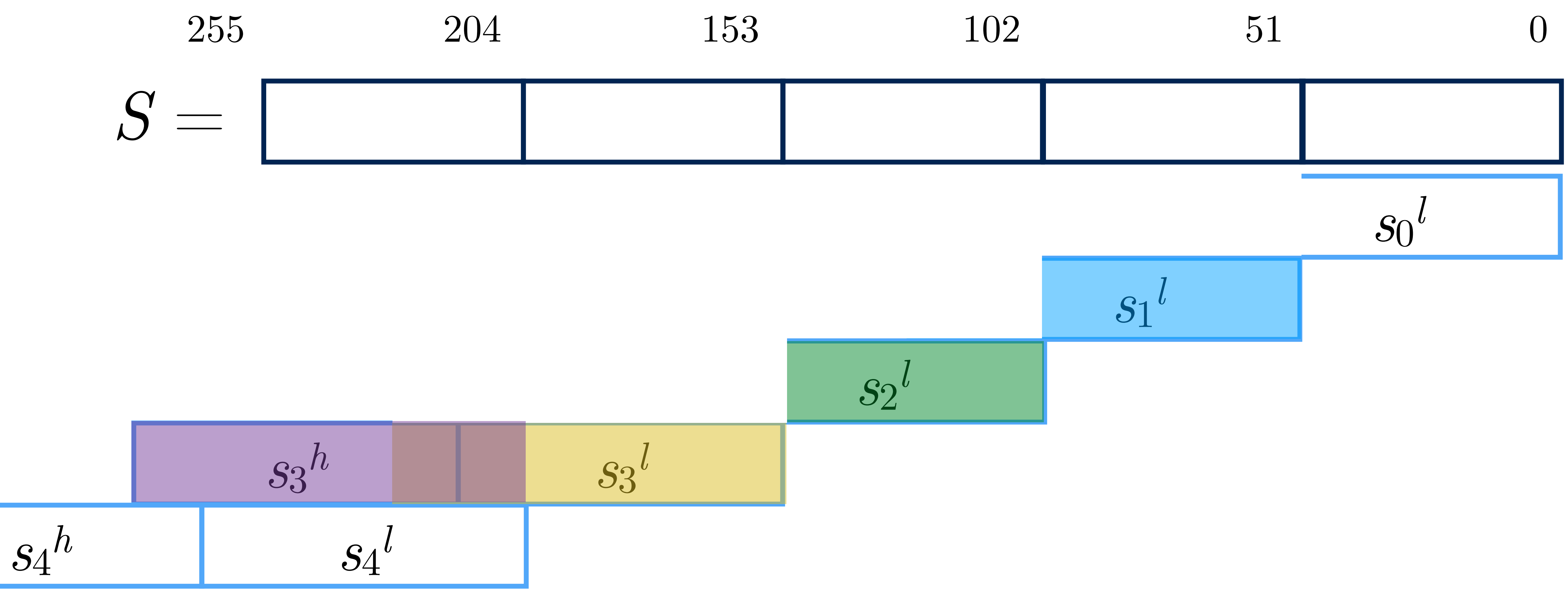
Delayed-Carry (Simplified)

- Compute $R \equiv S \pmod{2^{255}\text{-}19}$
- A limb in R has at most 52 bits



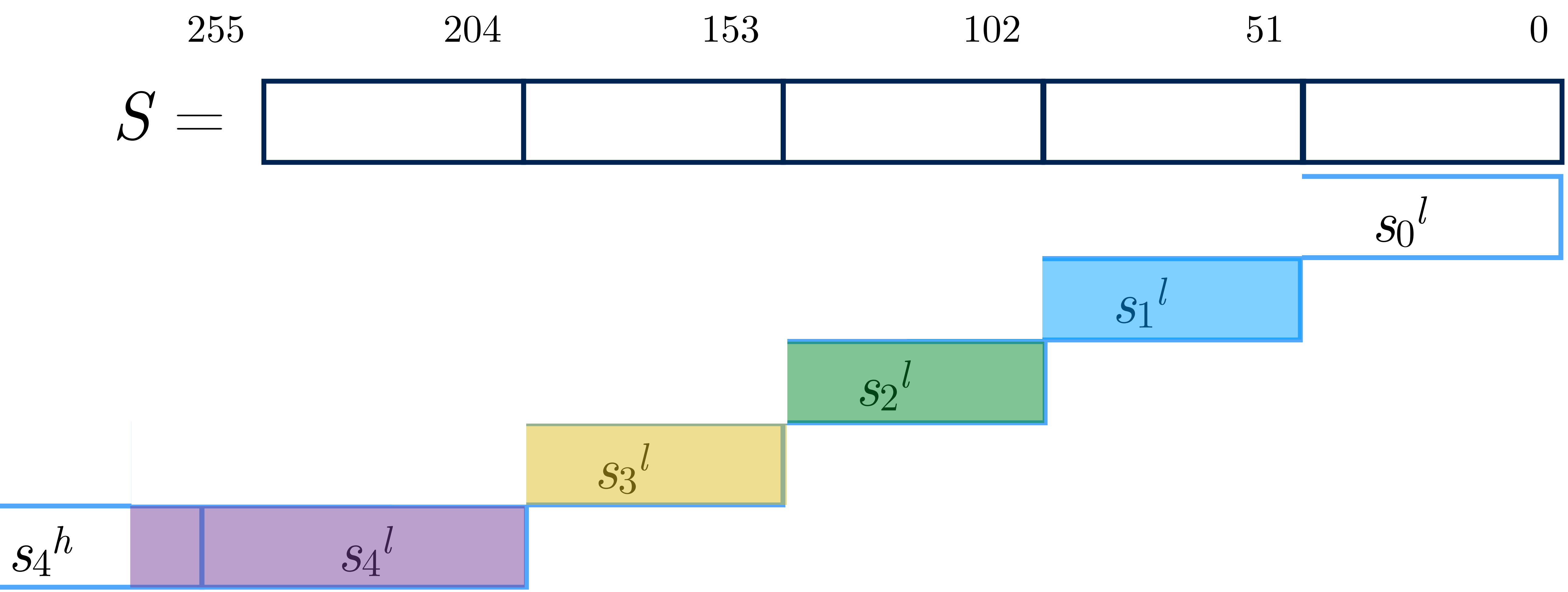
Delayed-Carry (Simplified)

- Compute $R \equiv S \pmod{2^{255}\text{-}19}$
- A limb in R has at most 52 bits



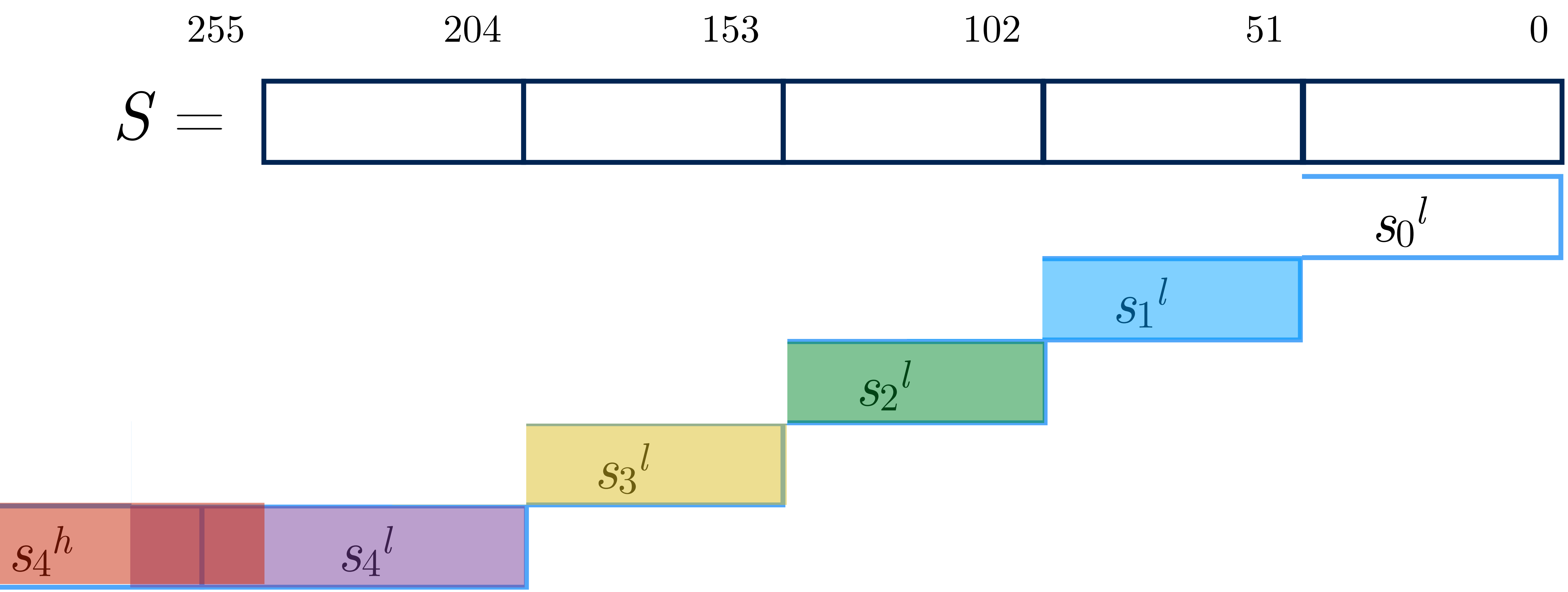
Delayed-Carry (Simplified)

- Compute $R \equiv S \pmod{2^{255}\text{-}19}$
- A limb in R has at most 52 bits



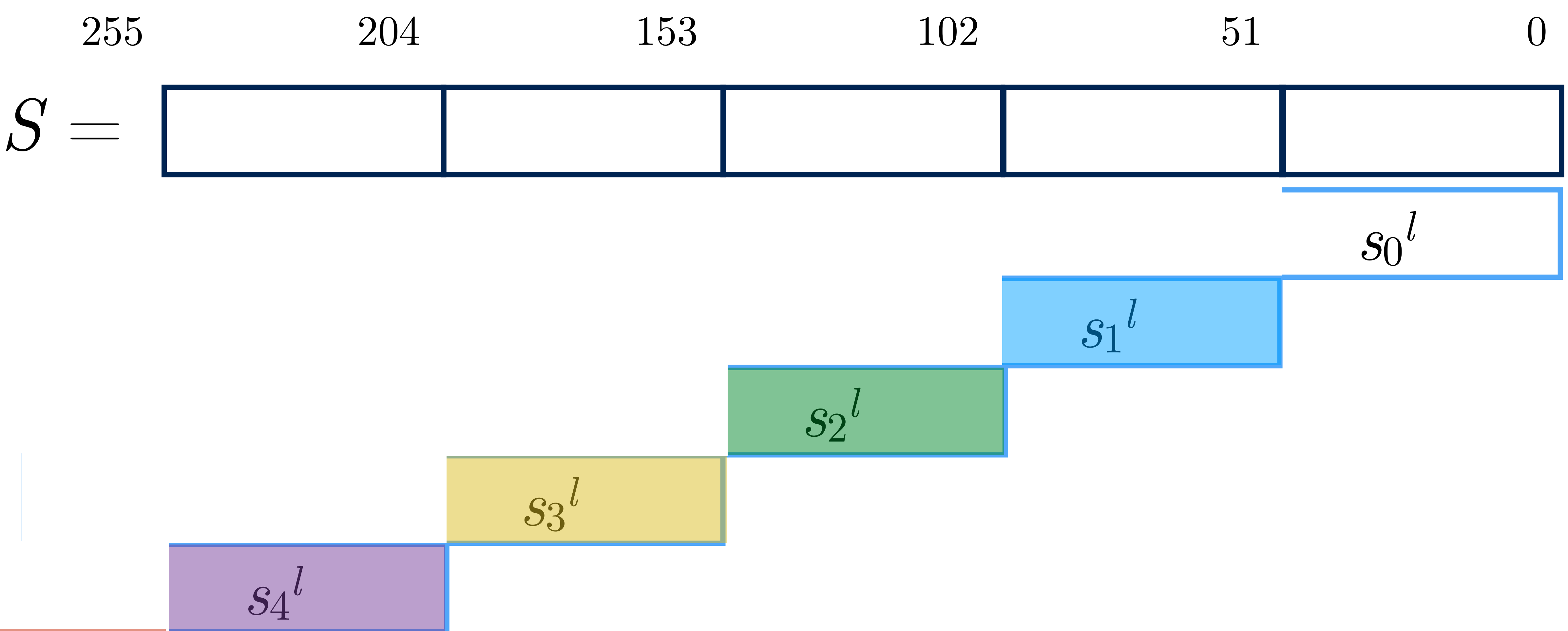
Delayed-Carry (Simplified)

- Compute $R \equiv S \pmod{2^{255}\text{-}19}$
- A limb in R has at most 52 bits



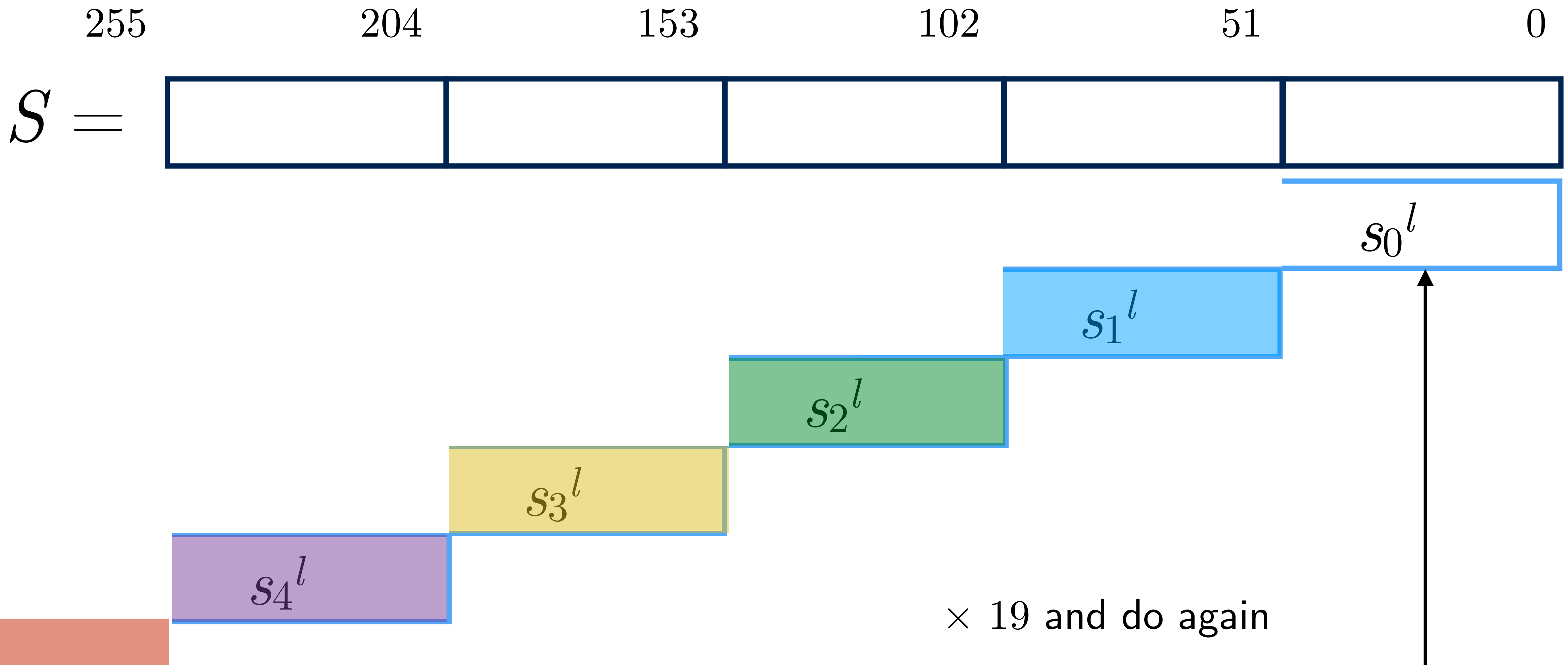
Delayed-Carry (Simplified)

- Compute $R \equiv S \pmod{2^{255}\text{-}19}$
- A limb in R has at most 52 bits



Delayed-Carry (Simplified)

- Compute $R \equiv S \pmod{2^{255}\text{-}19}$
- A limb in R has at most 52 bits



The Implementation

```
…
mulrax = *(uint64 *)(xp + 0)
(uint128) mulrdx mulrax = mulrax * *(uint64 *)(yp + 0)
carry? r0 += mulrax
mulr01 += mulrdx + carry
…
mulrax = *(uint64 *)(xp + 8)
(uint128) mulrdx mulrax = mulrax * *(uint64 *)(yp + 0)
carry? r1 += mulrax
mulr11 += mulrdx + carry
…
mulr01 = (mulr01.r0) << 13
r0 &= mulredmask
…
```

- Sequential
- Bit-vector operations
- Good for SMT solvers

SAT

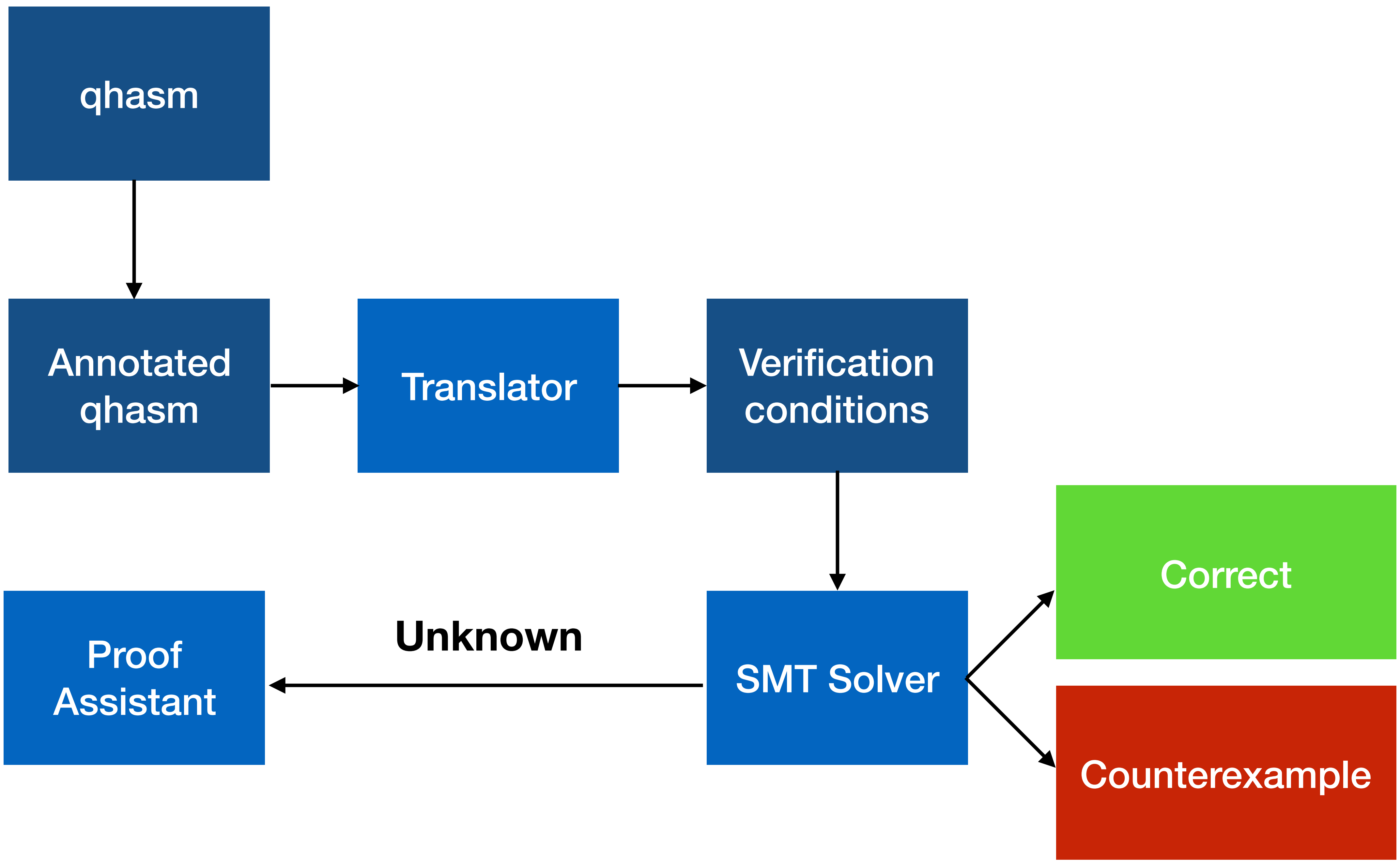
- Satisfiability of Boolean expressions (usually in CNF)
- Can be effectively solved by modern SAT solvers
  - minisat
  - glucose
  - lingeling $(a \vee b) \wedge (\neg b \vee c) \wedge (a \vee d)$

SMT

- Satisfiability modulo theories
- A Boolean variable corresponds to an expression under a theory
- SMT solvers
  - Z3, Boolector, MathSAT, STP, Yices

$$\overbrace{(a[0] = 1)}^{\textbf{array}} \land \overbrace{(3.07 + f1 < f2)}^{\textbf{floating point}} \land \overbrace{((b1 \,\&\, b2) << 2 = b3)}^{\textbf{bit-vector}}$$

Hybrid Methodology



Annotation

- Hoare logic style annotation: $\{\ P\ \}\ C\ \{\ Q\ \}$ where
  - P: the precondition
  - C: the program
  - Q: the postcondition
- $\{\ P\ \}\ C\ \{\ Q\ \}$ is valid $\leftrightarrow$ for every initial state s satisfying P, if C runs from s and ends with a final state t, then t satisfies Q

Translation

{ precondition } program { postcondition }

$\{x - 1 > 0\}$ `x := x - 1` $\{x > 0\}$

$\{x - 1 > 0\}$ `x := x - 1` $\{x > 0\}$

Translation

$\{x - 1 > 0\}$ `x := x - 1` $\{x > 0\}$

$$\{x_0 - 1 > 0\}\ x_1 = x_0 - 1\ \{x_1 > 0\}$$

1. Convert to SSA (Static Single Assignment) form

Translation

$\{x - 1 > 0\}$ `x := x - 1` $\{x > 0\}$

$$(x_0 - 1 > 0) \wedge (x_1 = x_0 - 1) \wedge (x_1 > 0)$$

2. Make conjunction

Translation

$\{x - 1 > 0\}$ `x := x - 1` $\{x > 0\}$

$(x_0 - 1 > 0) \wedge (x_1 = x_0 - 1) \wedge \neg(x_1 > 0)$

3. Take negation of the post-condition

Translation

$\{x - 1 > 0\}$ `x := x - 1` $\{x > 0\}$

$$(x_0 - 1 > 0) \wedge (x_1 = x_0 - 1) \wedge \neg(x_1 > 0)$$

Unsatisfiable: Valid specification
Satisfiable: Counterexample found

Specification of Multiplication

$\{\ 0 \leq x_0,\ x_1,\ x_2,\ x_3,\ x_4 < 2^{52} \wedge 0 \leq y_0,\ y_1,\ y_2,\ y_3,\ y_4 < 2^{52}\ \}$

`Multiply`

`Reduce`

`Delayed-Carry`

$\{$

$R \equiv XY\ (mod\ 2^{255}\text{-}19) \wedge$

$0 \leq r_0 < 2^{52} \wedge$

$0 \leq r_1 < 2^{52} \wedge$

$0 \leq r_2 < 2^{52} \wedge$

$0 \leq r_3 < 2^{52} \wedge$

$0 \leq r_4 < 2^{52}$

$\}$

Specification of Multiplication

$\{\ 0 \leq x_0, x_1, x_2, x_3, x_4 < 2^{52} \wedge 0 \leq y_0, y_1, y_2, y_3, y_4 < 2^{52}\ \}$

`Multiply`

`Reduce`

`Delayed-Carry`

$\{$

$R \equiv XY\ (mod\ 2^{255}\text{-}19) \wedge$

$0 \leq r_0 < 2^{52} \wedge$

$0 \leq r_1 < 2^{52} \wedge$

$0 \leq r_2 < 2^{52} \wedge$

$0 \leq r_3 < 2^{52} \wedge$

$0 \leq r_4 < 2^{52}$

$\}$

Not proven!

Problems

- The SMT solver failed to verify the multiplication operation
  - { pre } multiplication { post }
- Verify the three steps (multiply, reduce, carry) separately

$$\frac{\vdash \{\ P\ \}\ C_1\ \{\ Q\ \} \qquad \vdash \{\ Q\ \}\ C_2\ \{\ R\ \}}{\vdash \{\ P\ \}\ C_1;\ C_2\ \{\ R\ \}}\ \text{Seq}$$

Specification of Multiplication Revisited

$\{\ P\ \}$

`Multiply`

$\{\ R_1\ \}$

`Reduce`

$\{\ R_2\ \}$

`Delayed-Carry`

$\{\ Q\ \}$

Specification of Multiplication Revisited

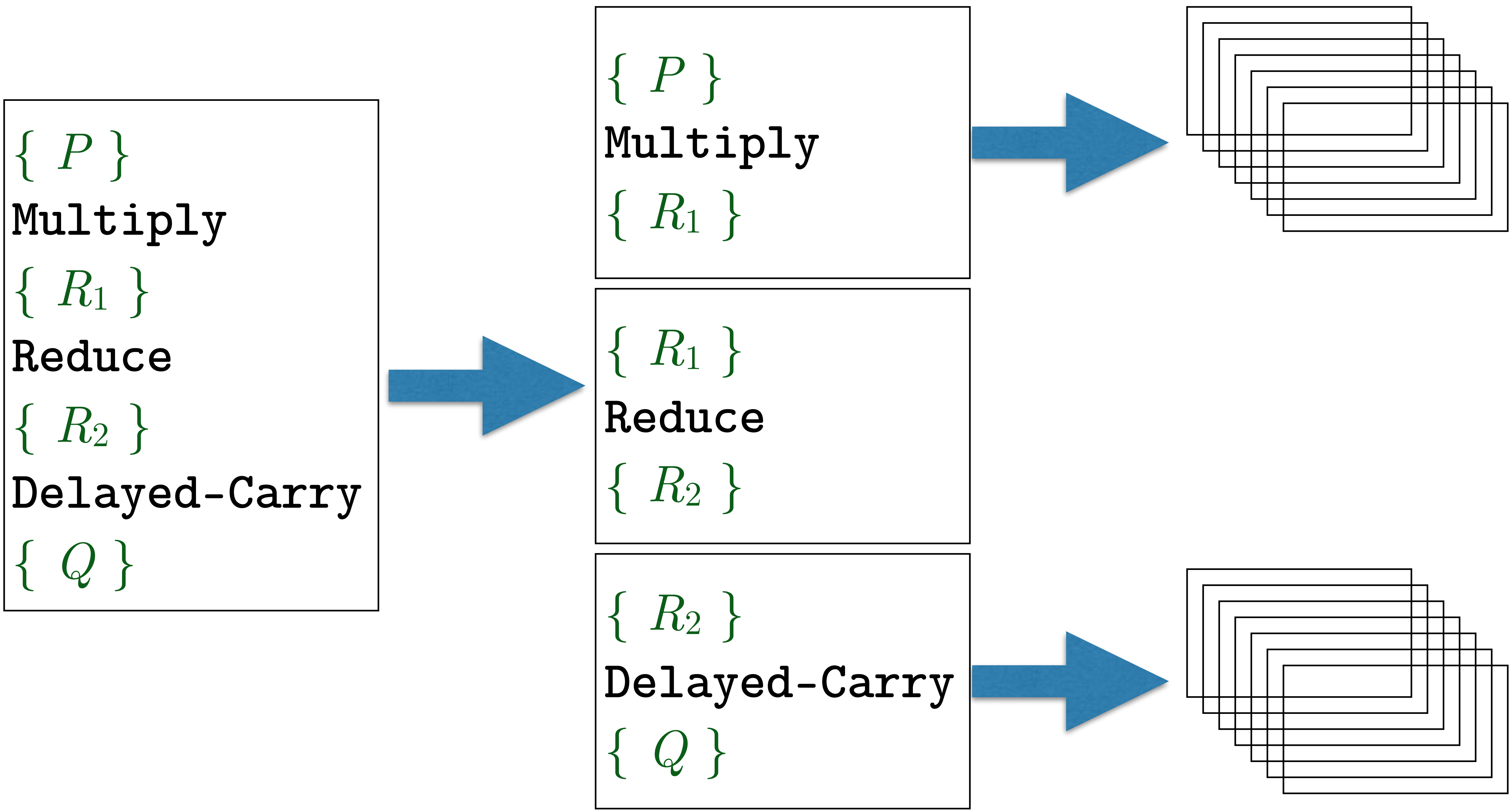
$\{\ P\ \}$
`Multiply`
$\{\ R_1\ \}$
`Reduce`
$\{\ R_2\ \}$
`Delayed-Carry`
$\{\ Q\ \}$

$\{\ P\ \}$
`Multiply`
$\{\ R_1\ \}$

$\{\ R_1\ \}$
`Reduce`
$\{\ R_2\ \}$

$\{\ R_2\ \}$
`Delayed-Carry`
$\{\ Q\ \}$

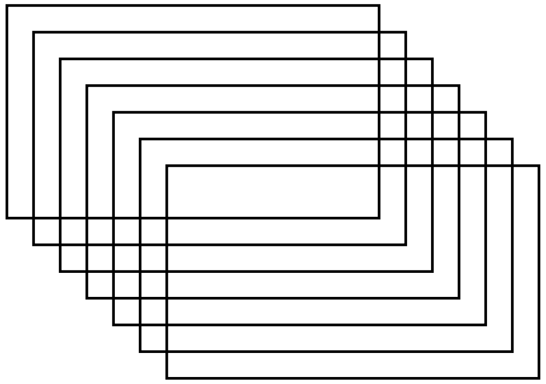
Specification of Multiplication Revisited



Specification of Multiplication Revisited

$\{\ P\ \}$
Multiply
$\{\ R_1\ \}$
Reduce
$\{\ R_2\ \}$
Delayed-Carry
$\{\ Q\ \}$

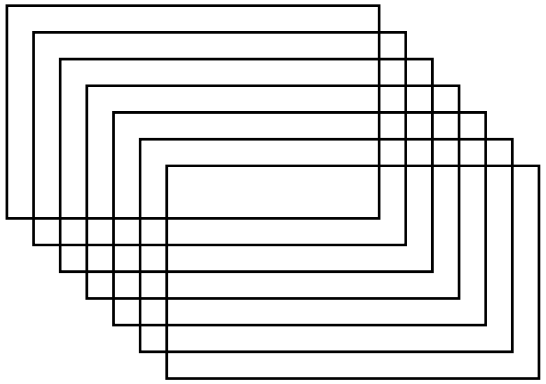
$\{\ P\ \}$
Multiply
$\{\ R_1\ \}$



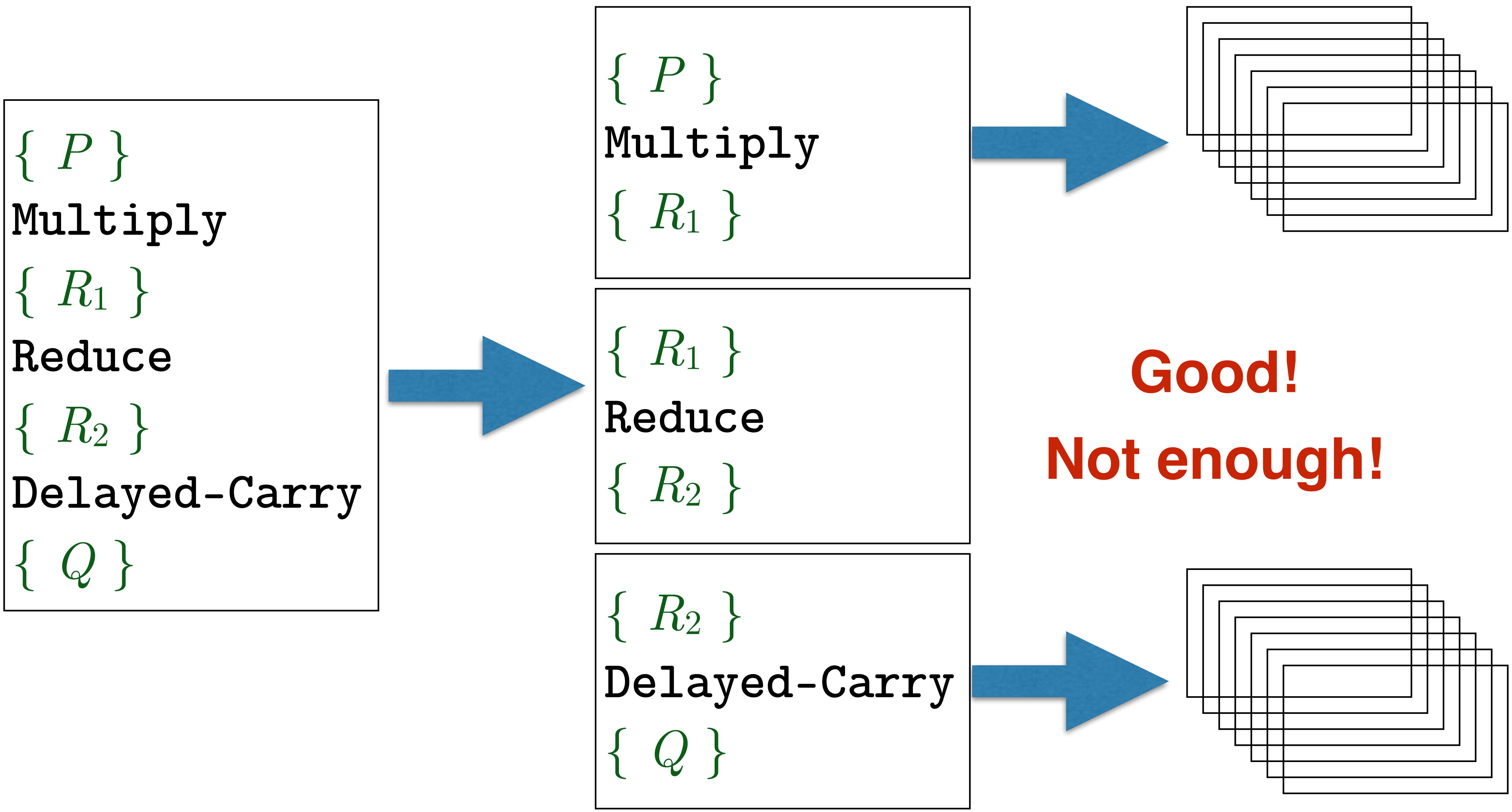
$\{\ R_1\ \}$
Reduce
$\{\ R_2\ \}$

Good!

$\{\ R_2\ \}$
Delayed-Carry
$\{\ Q\ \}$



Specification of Multiplication Revisited



Simple but Failed

$\{ 0 \leq xp[0], xp[8], xp[16] < 2^{54} \land r11.r1 = 2 * xp[0]@128 * xp[8]@128 \}$

read memory	`rax = xp[0]`
shift left	`rax <<= 1`
128-bit multiplication	`(uint128) rdx rax = rax * xp[16]`
	`r2 = rax`
	`r21 = rdx`

$\{ r21.r2 = 2 * xp[0]@128 * xp[16]@128 \}$

(qhasm syntax simplified)

$x@n$: extension of x to n bits

Simple but Failed

$\{ 0 \leq xp[0], xp[8], xp[16] < 2^{54} \wedge r11.r1 = 2 * xp[0]@128 * xp[8]@128 \}$

read memory	`rax = xp[0]`
shift left	`rax <<= 1`
128-bit multiplication	`(uint128) rdx rax = rax * xp[16]`
	`r2 = rax`
	`r21 = rdx`

$\{ r21.r2 = 2 * xp[0]@128 * xp[16]@128 \}$

(qhasm syntax simplified)

$x@n$: extension of x to n bits

Need more heuristics to reduce the complexity

Heuristic 1
- Split Conjunctions -

$\{\ P\ \}$
Code
$\{\ Q_1 \wedge Q_2\ \}$

$\{\ P\ \}$
Code
$\{\ Q_1\ \}$

$\{\ P\ \}$
Code
$\{\ Q_2\ \}$

Heuristic 2
- Delayed Extension -

$$R = (x@256 * \text{y}@256)\ 2^{64} + \ldots$$

($\boxed{x}$) * ($\boxed{y}$)

$$R = (x@128 * y@128)@256\ 2^{64} + \ldots$$

(($\boxed{x}$) * ($\boxed{y}$))

R: 256-bit vector

$x,\ y$: 64-bit vectors

Heuristic 2
- Delayed Extension -

$$R = (x@256 * \mathrm{y}@256)\ 2^{64} + \dots$$

(0000…0000 $\boxed{\quad x \quad}$) * (0000…0000 $\boxed{\quad y \quad}$)

$$R = (x@128 * y@128)@256\ 2^{64} + \dots$$

(($\boxed{\quad x \quad}$) * ($\boxed{\quad y \quad}$))

R: 256-bit vector

x, y: 64-bit vectors

Heuristic 2
- Delayed Extension -

$$R = (x@256 * \mathrm{y}@256)\ 2^{64} + \ldots$$

(0000…0000 $\boxed{\quad x \quad}$) * (0000…0000 $\boxed{\quad y \quad}$)

$$R = (x@128 * y@128)@256\ 2^{64} + \ldots$$

((00…00 $\boxed{\quad x \quad}$) * (00…00 $\boxed{\quad y \quad}$))

R: 256-bit vector

x, y: 64-bit vectors

Heuristic 2
- Delayed Extension -

$$R = (x@256 \ * \ \mathrm{y}@256)\ 2^{64} + \ldots$$

(0000…0000 $\boxed{\quad x \quad}$) * (0000…0000 $\boxed{\quad y \quad}$)

$$R = (x@128 \ * \ y@128)@256\ 2^{64} + \ldots$$

00…00 ((00…00 $\boxed{\quad x \quad}$) * (00…00 $\boxed{\quad y \quad}$))

R: 256-bit vector

$x,\ y$: 64-bit vectors

Heuristic 3
- Match Code -

$\{\ P\ \}$

$\texttt{rh.r1 = 19x}_0\texttt{y}_1$

$\texttt{rh.rl += 19x}_1\texttt{y}_0$

$\{\ rh.rl = 19(x_0 y_1\ +\ x_1 y_0)\ \}$

$\{\ P\ \}$

$\texttt{rh.rl = 19x}_0\texttt{y}_1$

$\texttt{rh.rl += 19x}_1\texttt{y}_0$

$\{\ rh.rl = 19x_0 y_1\ +\ 19x_1 y_0\ \}$

Heuristic 4
- Over-approximation -

$\{\ 0 \leq xp[0], xp[8], xp[16] < 2^{54} \wedge r11.r1 = 2 * xp[0]@128 * xp[8]@128\ \}$

```
rax = xp[0]
rax <<= 1
(uint128) rdx rax = rax * xp[16]
r2 = rax
r21 = rdx
```

$\{\ r21.r2 = 2 * xp[0]@128 * xp[16]@128\ \}$

Heuristic 4
- Over-approximation -

$\{\ 0 \leq xp[0], xp[8], xp[16] < 2^{54} \wedge r11\cancel{.r1 = 2 * xp[0]@128 * xp[8]@128\ \}}$

```
rax = xp[0]
rax <<= 1
(uint128) rdx rax = rax * xp[16]
r2 = rax
r21 = rdx
```

$\{\ r21.r2 = 2 * xp[0]@128 * xp[16]@128\ \}$

Heuristic 4
- Over-approximation -

$\{\ 0 \leq xp[0], xp[8], xp[16] < 2^{54} \wedge r11 \cancel{.r1 = 2 * xp[0]@128 * xp[8]@128\ \}}$

```
rax = xp[0]
rax <<= 1
(uint128) rdx rax = rax * xp[16]
r2 = rax
r21 = rdx
```

$\{\ r21.r2 = 2 * xp[0]@128 * xp[16]@128\ \}$

Success: done

Fail: prove the original one

Experimental Results

File Name	Description		# of *limb*	# of *MC*	Time
radix-2^{64} representation					
fe25519r64_mul-1	$r = x * y \pmod{2^{255} - 19}$, a buggy version		4	1	0m8.73s
fe25519r64_add	$r = x + y \pmod{2^{255} - 19}$	Operations of Algorithm 2	4	0	0m3.15s
fe25519r64_sub	$r = x - y \pmod{2^{255} - 19}$		4	0	0m16.24s
fe25519r64_mul-2	$r = x * y \pmod{2^{255} - 19}$, a fixed version of fe25519r64_mul-1		4	19	73m55.16s
fe25519r64_mul121666	$r = x * 121666 \pmod{2^{255} - 19}$		4	2	0m2.03s
fe25519r64_sq	$r = x * x \pmod{2^{255} - 19}$		4	15	3m16.67s
ladderstepr64	The implementation of Algorithm 2		4	14	0m3.23s
fe19119_mul	$r = x * y \pmod{2^{191} - 19}$		3	12	8m43.07s
mul1271	$r = x * y \pmod{2^{127} - 1}$		2	1	141m22.06s
radix-2^{51} representation					
fe25519_add	$r = x + y \pmod{2^{255} - 19}$	Operations of Algorithm 2	5	0	0m16.35s
fe25519_sub	$r = x - y \pmod{2^{255} - 19}$		5	0	3m38.62s
fe25519_mul	$r = x * y \pmod{2^{255} - 19}$		5	27	5658m2.15s
fe25519_mul121666	$r = x * 121666 \pmod{2^{255} - 19}$		5	5	0m12.75s
fe25519_sq	$r = x * x \pmod{2^{255} - 19}$		5	17	463m59.5s
ladderstep	The implementation of Algorithm 2		5	14	1m29.05s
mul25519	$r = x * y \pmod{2^{255} - 19}$, a 3-phase implementation		5	3	286m52.75s
mul25519-p2-1	The delayed carry phase of $r = x * y \pmod{2^{255} - 19}$		5	1	2723m16.56s
mul25519-p2-2	The delayed carry phase of $r = x * y \pmod{2^{255} - 19}$ with two sub-phases		5	2	263m35.46s
muladd25519	$r = x * y + z \pmod{2^{255} - 19}$		5	7	1569m11.06s
re15319	$r = x * y \pmod{2^{153} - 19}$		3	3	2409m16.89s

Experimental Results

File Name	Description		# of *limb*	# of *MC*	Time
	radix-2^{64} representation				
fe25519r64_mul-1	$r = x * y \pmod{2^{255} - 19}$, a buggy version		4	1	0m8.73s
fe25519r64_add	$r = x + y \pmod{2^{255} - 19}$	Operations of Algorithm 2	4	0	0m3.15s
fe25519r64_sub	$r = x - y \pmod{2^{255} - 19}$		4	0	0m16.24s
fe25519r64_mul-2	$r = x * y \pmod{2^{255} - 19}$, a fixed version of fe25519r64_mul-1		4	19	73m55.16s
fe25519r64_mul121666	$r = x * 121666 \pmod{2^{255} - 19}$		4	2	0m2.03s
fe25519r64_sq	$r = x * x \pmod{2^{255} - 19}$		4	15	3m16.67s
ladderstepr64	The implementation of Algorithm 2		4	14	0m3.23s
fe19119_mul	$r = x * y$ (m				3.07s
mul1271	$r = x * y$ (m				22.06s
fe25519_add	$r = x + y$ (m				6.35s
fe25519_sub	$r = x - y \pmod{2^{255} - 19}$	Operations of Algorithm 2	5	0	3[illegible]38.62s
fe25519_mul	$r = x * y \pmod{2^{255} - 19}$		5	27	5658m2.15s
fe25519_mul121666	$r = x * 121666 \pmod{2^{255} - 19}$		5	5	0m12.75s
fe25519_sq	$r = x * x \pmod{2^{255} - 19}$		5	17	463m59.5s
ladderstep	The implementation of Algorithm 2		5	14	1m29.05s
mul25519	$r = x * y \pmod{2^{255} - 19}$, a 3-phase implementation		5	3	286m52.75s
mul25519-p2-1	The delayed carry phase of $r = x * y \pmod{2^{255} - 19}$		5	1	2723m16.56s
mul25519-p2-2	The delayed carry phase of $r = x * y \pmod{2^{255} - 19}$ with two sub-phases		5	2	263m35.46s
muladd25519	$r = x * y + z \pmod{2^{255} - 19}$		5	7	1569m11.06s
re15319	$r = x * y \pmod{2^{153} - 19}$		3	3	2409m16.89s

5	27	5658m2.15s

Experimental Results

File Name	Description		# of *limb*	# of *MC*	Time
radix-2^{64} representation					
fe25519r64_mul-1	$r = x * y \pmod{2^{255} - 19}$, a buggy version		4	1	0m8.73s
fe25519r64_add	$r = x + y \pmod{2^{255} - 19}$	Operations of Algorithm 2	4	0	0m3.15s
fe25519r64_sub	$r = x - y \pmod{2^{255} - 19}$		4	0	0m16.24s
fe25519r64_mul-2	$r = x * y \pmod{2^{255} - 19}$, a fixed version of fe25519r64_mul-1		4	19	73m55.16s
fe25519r64_mul121666	$r = x * 121666 \pmod{2^{255} - 19}$		4	2	0m2.03s
fe25519r64_sq	$r = x * x \pmod{2^{255} - 19}$		4	15	3m16.67s
ladderstepr64	The implemen[illegible]		[illegible]	[illegible]	[illegible]3.23s
fe19119_mul	$r = x * y$ (m[illegible]		[illegible]	[illegible]	[illegible]3.07s
mul1271	$r = x * y$ (m[illegible]		[illegible]	[illegible]	[illegible]22.06s
fe25519_add	$r = x + y$ (m[illegible]		[illegible]	[illegible]	[illegible]6.35s
fe25519_sub	$r = x - y \pmod{2^{255} - 19}$	Operations of [illegible]	5	0	3[illegible]38.62s
fe25519_mul	$r = x * y$ (mo[illegible]		[illegible]	[illegible]	[illegible]2.15s
fe25519_mul121666	$r = x * 12166$[illegible]		[illegible]	[illegible]	[illegible]2.75s
fe25519_sq	$r = x * x$ (mc[illegible]		[illegible]	[illegible]	[illegible]59.5s
ladderstep	The impleme[illegible]		[illegible]	[illegible]	[illegible]9.05s
mul25519	$r = x * y \pmod{2^{255} - 19}$, a 3-phase implementation		5	3	28[illegible]52.75s
mul25519-p2-1	The delayed carry phase of $r = x * y \pmod{2^{255} - 19}$		5	1	2723m16.56s
mul25519-p2-2	The delayed carry phase of $r = x * y \pmod{2^{255} - 19}$ with two sub-phases		5	2	263m35.46s
muladd25519	$r = x * y + z \pmod{2^{255} - 19}$		5	7	1569m11.06s
re15319	$r = x * y \pmod{2^{153} - 19}$		3	3	2409m16.89s

5	27	5658m2.15s

5	1	2723m16.56s
5	2	263m35.46s

What Remains

- The translator may be incorrect
- The SMT solver may be incorrect

What Remains

- The translator may be incorrect
- The SMT solver may be incorrect

Could we provide higher guarantee?

What Remains

- The translator may be incorrect
- The SMT solver may be incorrect

Could we provide higher guarantee?

Certify our verification approach in Coq

Coq

- A proving assistant based on Calculus of Inductive Constructions (CIC)
- Proofs in Coq are type checked

Natural Numbers in Coq

```
Inductive nat : Set :=
  | O : nat
  | S : nat -> nat.
```

```
0: O
1: S O
2: S S O
3: S S S O
```

Addition in Coq

```
Fixpoint add n m :=
  match n with
  | 0 => m
  | S p => S (p + m)
  end
```

```
Lemma plus_n_O : forall n:nat, n = n + 0.
Proof.
  …
Qed.
Lemma plus_comm : forall n m:nat, n + m = m + n.
Lemma plus_assoc : forall n m p:nat, n + (m + p) = (n + m) + p.
```

What to be Certified?

- The qhasm translator ✔
- The SMT solver ✘
- Use Coq tactics with automatic proof generation

Solving Polynomials

$$P_0 = 0 \land P_1 = 0 \land \dots \land P_n = 0 \to \exists\ k,\ x - y = k * p$$

where P_0, ..., P_n are polynomials over integers

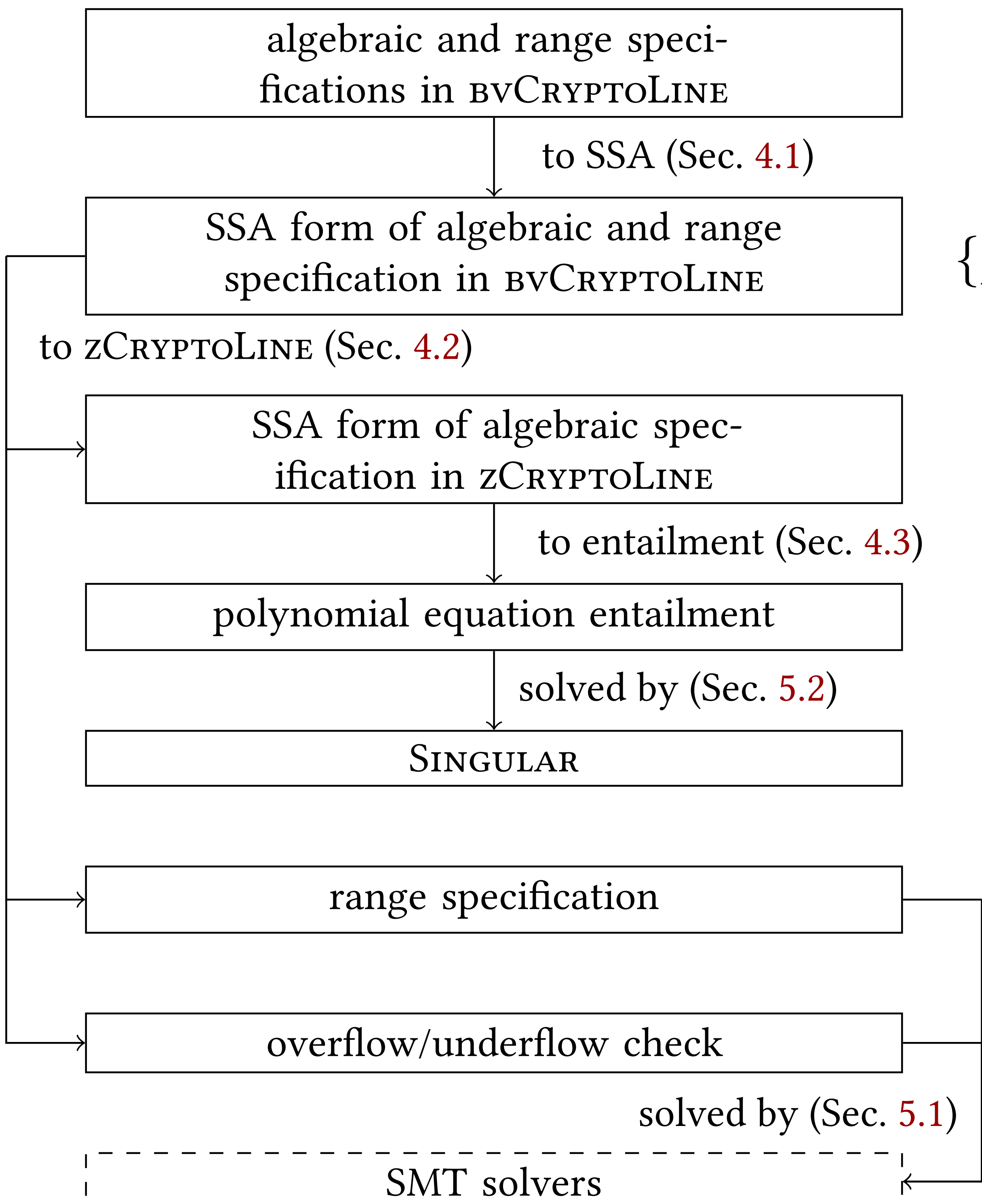
Can be solved automatically in Coq

John Harrison. Automating Elementary Number-Theoretic Proofs Using Gröbner Bases. CADE 2007.
Loïc Pottier. Connecting Groöner Bases Programs with Coq to do Proofs in Algebra, Geometry and Arithmetics.

Done in Coq

- Formalize a simplified version of qhasm: bvCryptoLine
- Translate bvCryptoLine programs with specifications to
  - polynomials via zCryptoLine (for algebraic properties)
  - SMT problems (for range properties)
- Prove the translation is correct
- Verify the solution to the polynomials

New Approach Overview



$$\{p_a \land p_r\}\ C_{bv}\ \{q_a \land q_r\}$$

$$\{{p_a}^{SSA} \land {p_r}^{SSA}\}\ {C_{bv}}^{SSA}\ \{{q_a}^{SSA} \land {q_r}^{SSA}\}$$

$$\{{p_a}^{SSA}\}\ {C_{bv}}^{SSA}\ \{{q_a}^{SSA}\}$$

$${p_a}^{SSA} \land {C_{bv}}^{SSA} \rightarrow {q_a}^{SSA}$$

$$\{{p_r}^{SSA}\}\ {C_{bv}}^{SSA}\ \{{q_r}^{SSA}\}$$

Experimental Results

Certified:

fe25519_mul	1m31.21s
fe25519_sq	0m51.754s

Previous:

fe25519_mul	5658m2.15s
fe25519_sq	463m59.5s