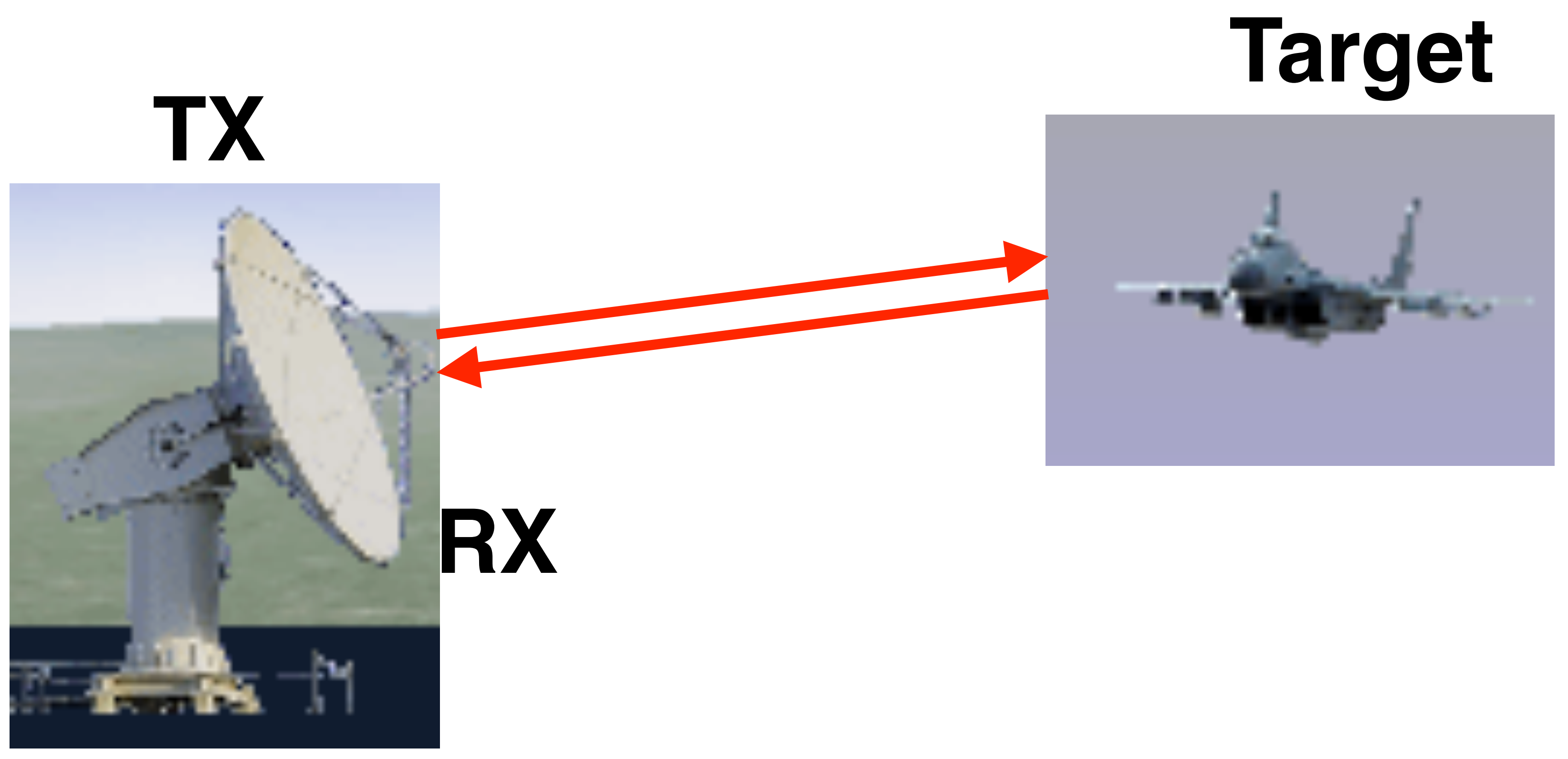
Project 1 | Monostatic pulse radar system

- **System Setting**

Sampling frequency 6MHz

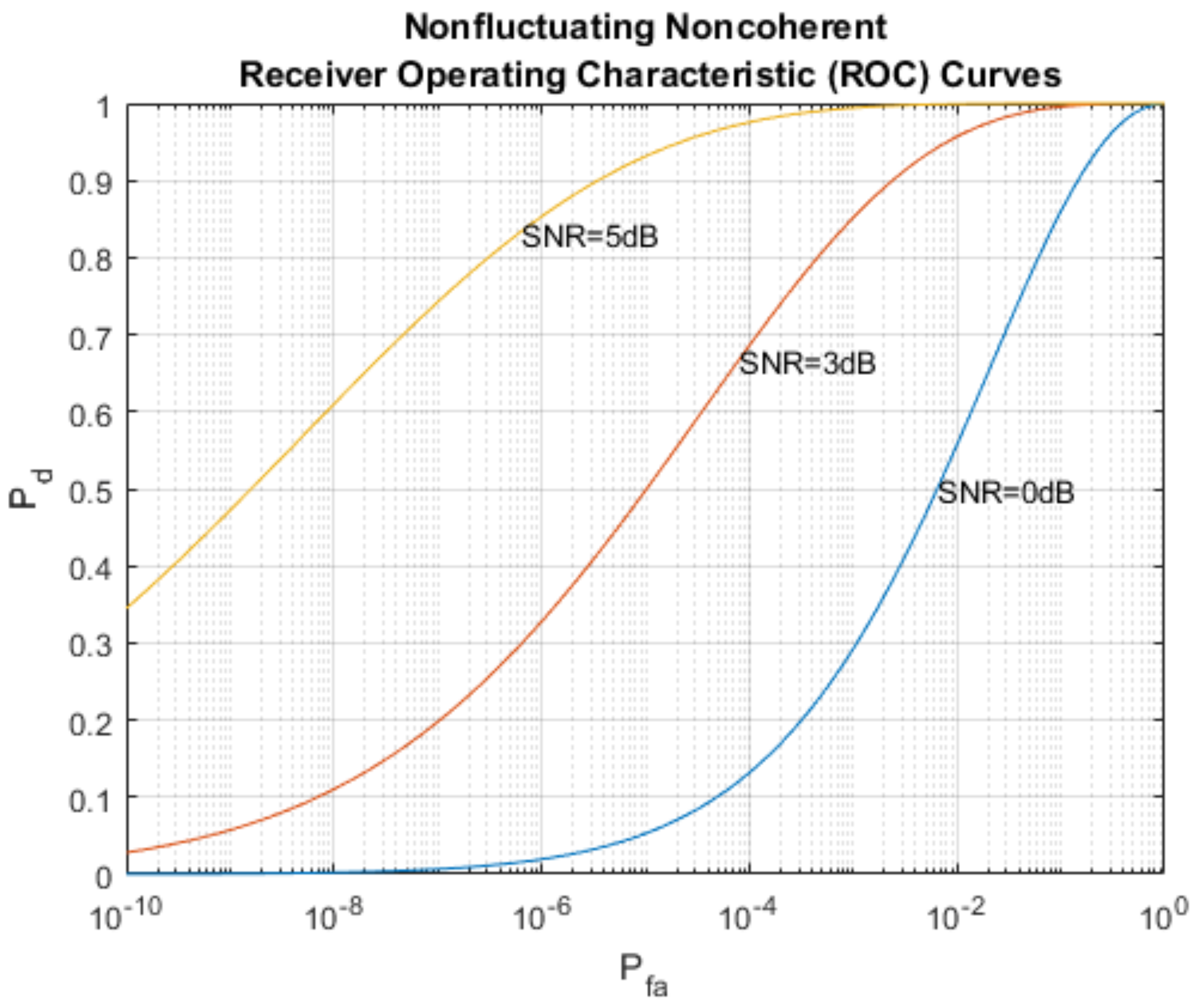
Maximum range 5000m → Pulse bandwidth 33us

Range resolution 50m → Pulse repetition frequency 30kHz



System model

Probability of detection 0.9
Probability of false alarm 1e-6



→ RX SNR requirement 5dB with 10 pulsed integrated

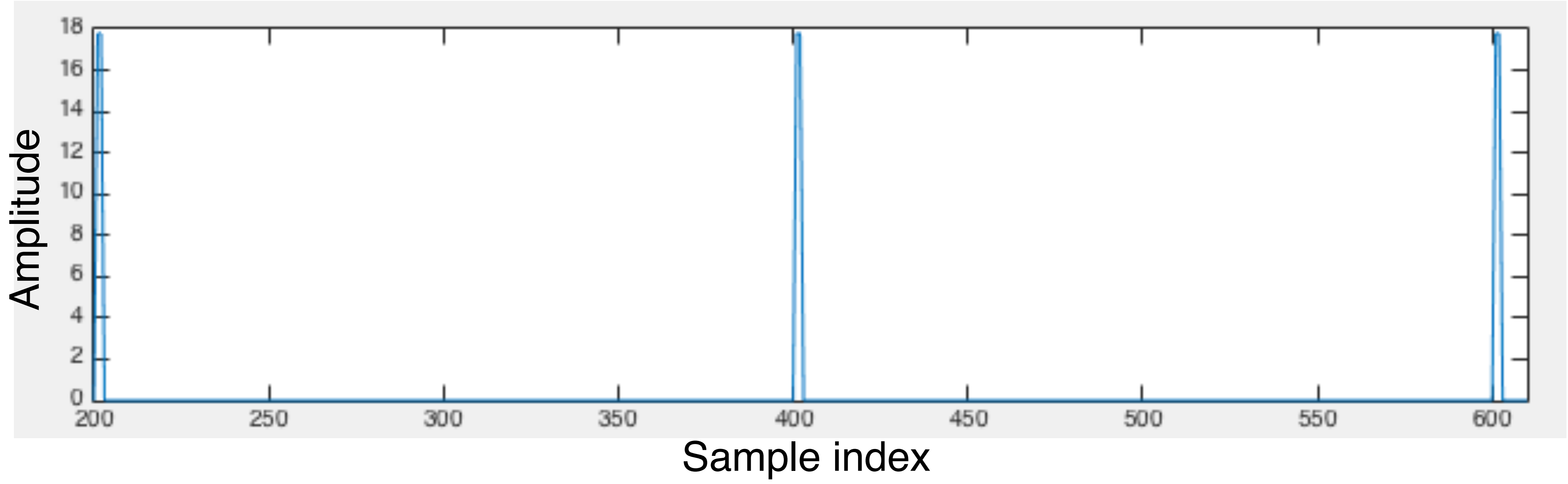
- Assume thermal noise power 0dB, RX SNR is 5dB, we can derive TX amplitude:

$$A_{tx} = \sqrt{\frac{prt}{pulse\_width} * P_{noise} * SNR_{rx}}$$

Project 1 | Monostatic pulse radar system

- **Transmitter**

- Transmission signal: Periodic rectangular wave with pulse width 2 samples and period 200 samples at sampling frequency 6MHz.



- **Target**

- Three targets with distance: 2024m 3518m 3845m.

- Assume the targets are static (no velocity).

- **Channel**

- Delay from radar TX back to radar RX induced by targets: 80 samples, 140 samples, 153 samples.

- Assume the ideal channel: no fading at all.

- **Receiver**

- Three reflected signals from targets merge at the receiver.

- Thermal noise at receiver with noise power 0dB (AWGN with mean 0 and variance 1)

$$y(t) = x(t+d_1) + x(t+d_2) + x(t+d_3) + n$$

- Non-coherent demodulation (convolution)

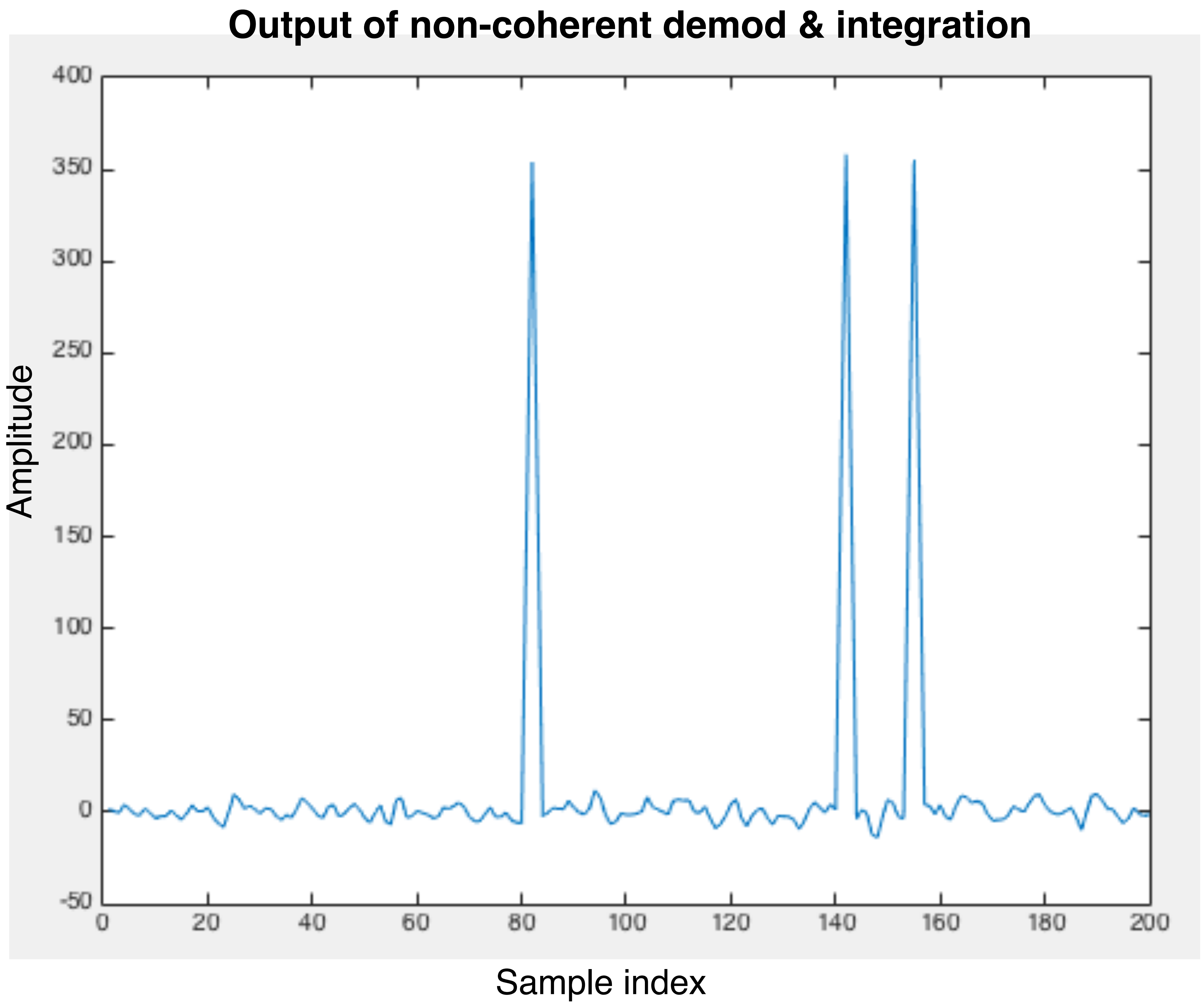
$$demod(t) = y(t) * Pulse(t)$$

- 10 periods integration

$$output(t) = \sum_{i=1}^{10} demod(t + (i-1) * T), t \in [0, T)$$

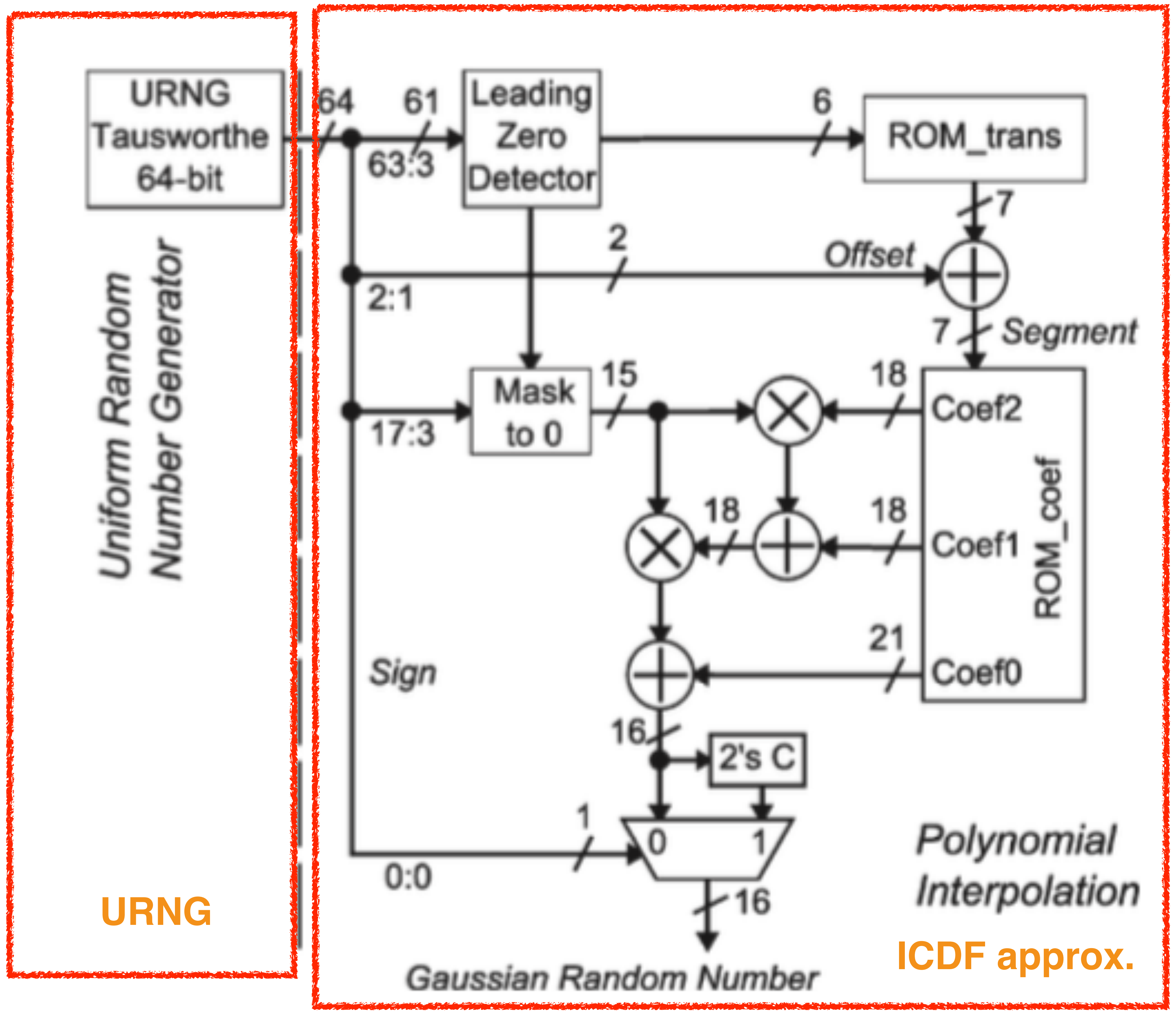
Project 1 | Monostatic pulse radar system

- **Simulation result**



	Target 1	Target 2	Target 3
Actual distance	2025	3518	3845
Estimated distance	2000	3500	3825

- **Gaussian noise generator consists of two blocks**
  - Tausworthe uniform random number generator (URNG)
  - ICDF approximation via polynomial interpolation



- **64-bit Tausworthe URNG implemented as [1]**

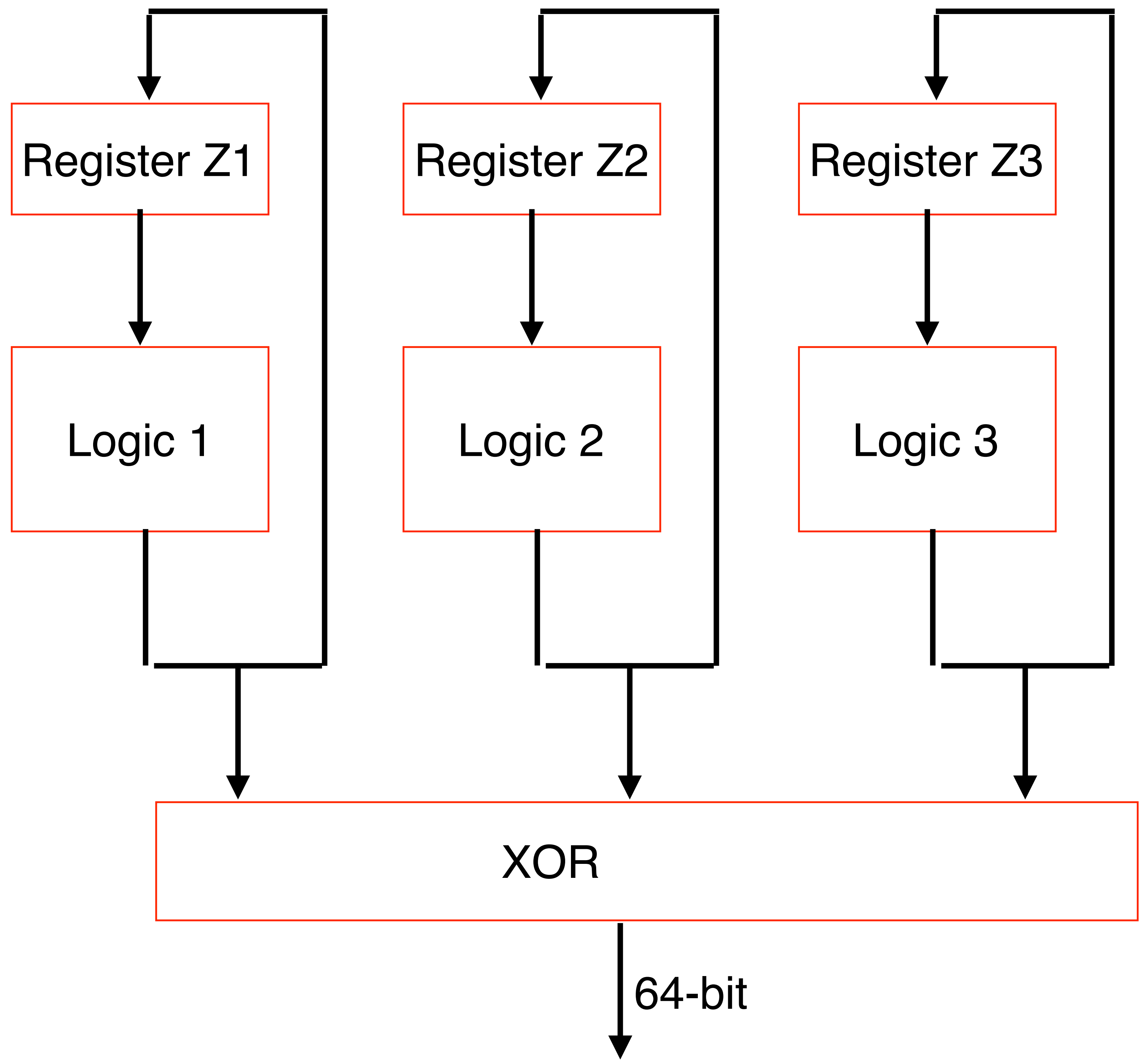


TABLE 2. ME-CF generators with $L = 64$ and $J = 3$.

	k_1	k_2	k_3	q_1	q_2	q_3	s_1	s_2	s_3	k	$\lg \rho$	N_1
1	63	58	55	5	19	24	24	13	7	176	176	17
2	63	55	52	1	24	3	27	22	14	170	170	27
3	63	55	47	5	24	5	22	18	21	165	165	21
4	63	55	47	31	24	21	17	21	5	165	165	21
5	63	58	57	31	19	22	20	26	13	178	175	27
6	63	58	57	31	19	22	26	14	15	178	175	27
7	63	58	57	31	19	22	20	11	16	178	175	27
8	63	58	57	31	19	22	29	26	20	178	175	27
9	63	58	57	31	19	22	11	25	27	178	175	27
10	63	57	55	5	22	24	51	18	19	175	172	27

From [2]

- Logic 1, Logic 2 and Logic 3 described as follows

$$z_{1,k} = \{z_{1,k-1}[22:1], z_{1,k-1}[58:17] \oplus z_{1,k-1}[63:22]\}$$

$$z_{2,k} = \{z_{2,k-1}[45:7], z_{2,k-1}[41:17] \oplus z_{2,k-1}[63:39]\}$$

$$z_{3,k} = \{z_{3,k-1}[44:19], z_{3,k-1}[39:2] \oplus z_{3,k-1}[63:26]\}$$

- Output of URNG

$$output = z_{1,k} \oplus z_{2,k} \oplus z_{3,k}$$

[1] P.L'Ecuyer. Maximally Equidistributed Combined Tausworthe Generators. Math. Computation, 65(213):203–213, 1996.

[2] P. L'Ecuyer. Tables of Maximally Equidistributed Combined LFSR Generators. Math. Computation, 68(225):261–269, 1999.

- **Hierarchical segmentation**
  - Outer segmentation: P2SL segmentation scheme for the first pass

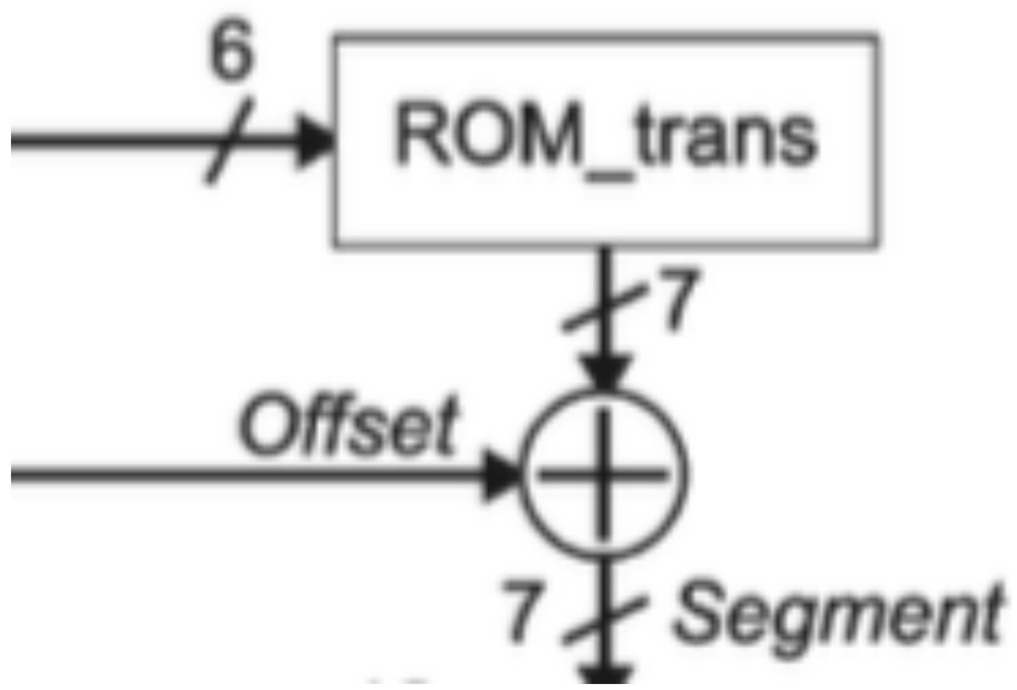
    (-) 61 bits feed into Leading zero detector, which leads to 62 outer segments for P2SL.

  - Inner segmentation: US segmentation scheme for the second pass

    (-) 2 bits for US segmentation leads to 4 inner segments.

  - Total segments: 62*4 = 248 segments

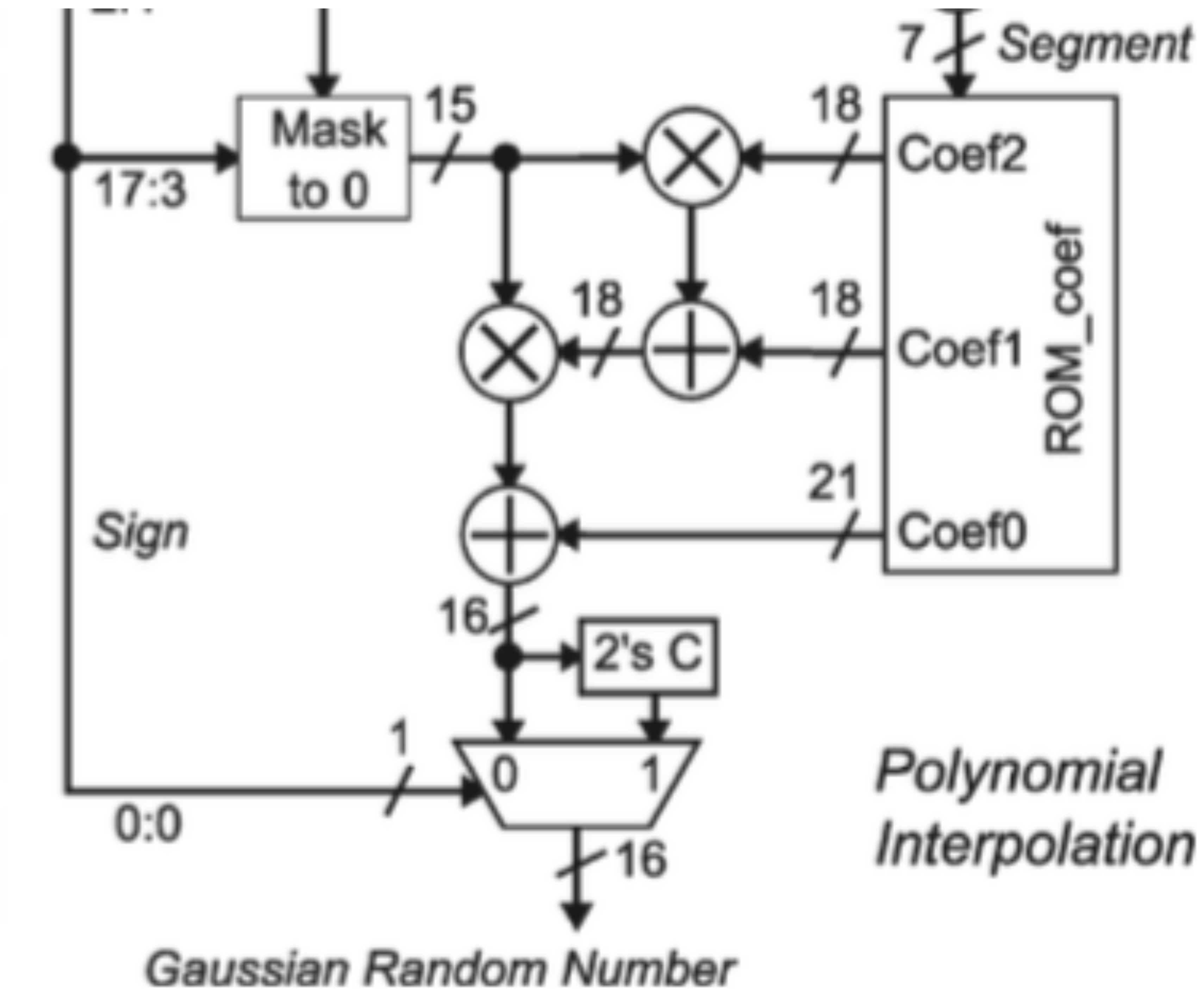
- **Address decoding**
  - No barrel-shifter, and use simple combination: outer_seg * 4 + offset.



- **Polynomial interpolation**
  - Pre-calculated polynomial coefficients: C0, C1, C2
  - Second-order polynomial evaluation by

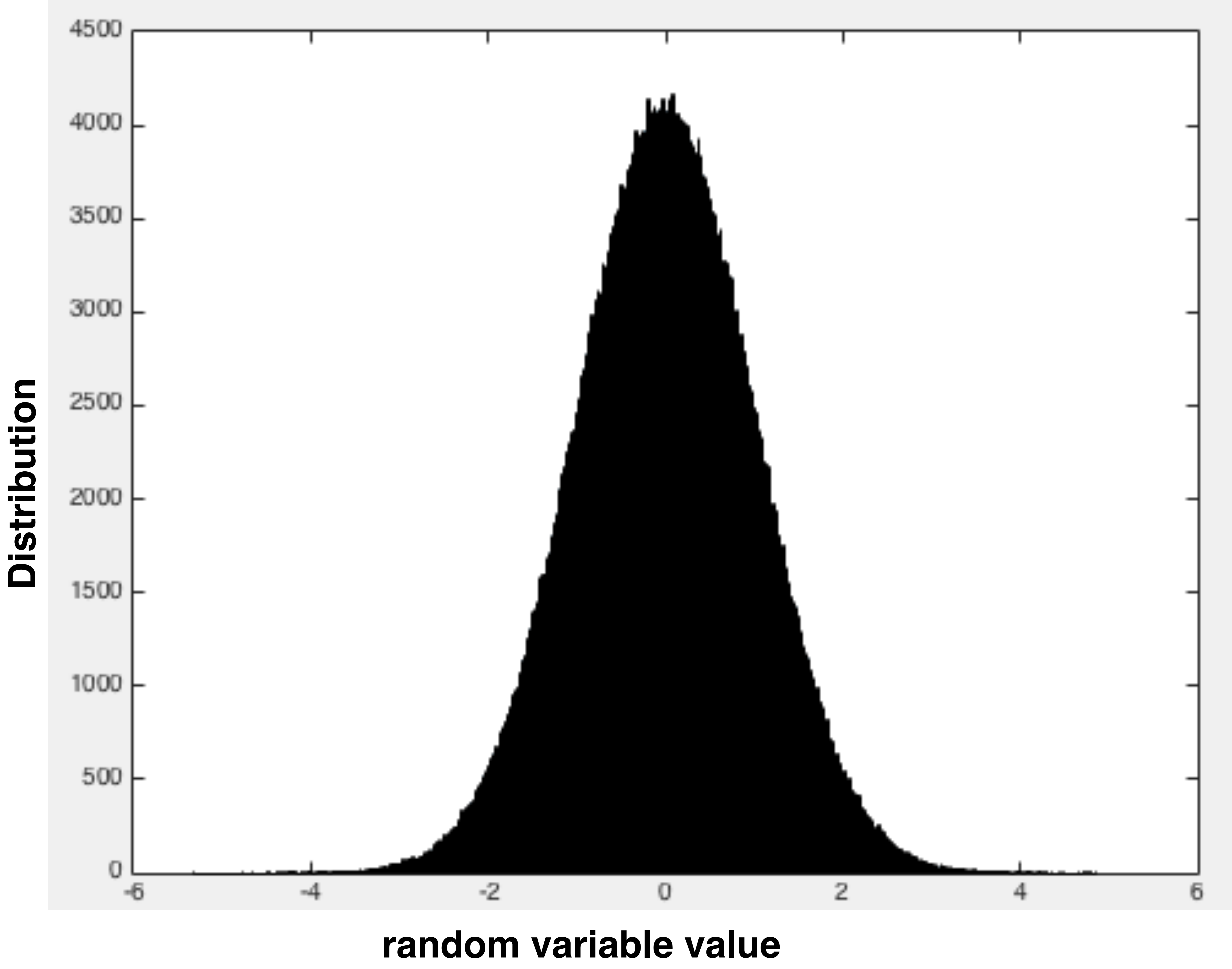
$$y = (C_2 x + C_1)x + C_0$$

  - Simple masking+Reverse order makes segment selection and interpolation mutually independent.
  - Bit 0 from URNG determine the sign for Gaussian random number



Project 2 | ICDF approximation

- **Simulation result**
  - Setup: 1M samples



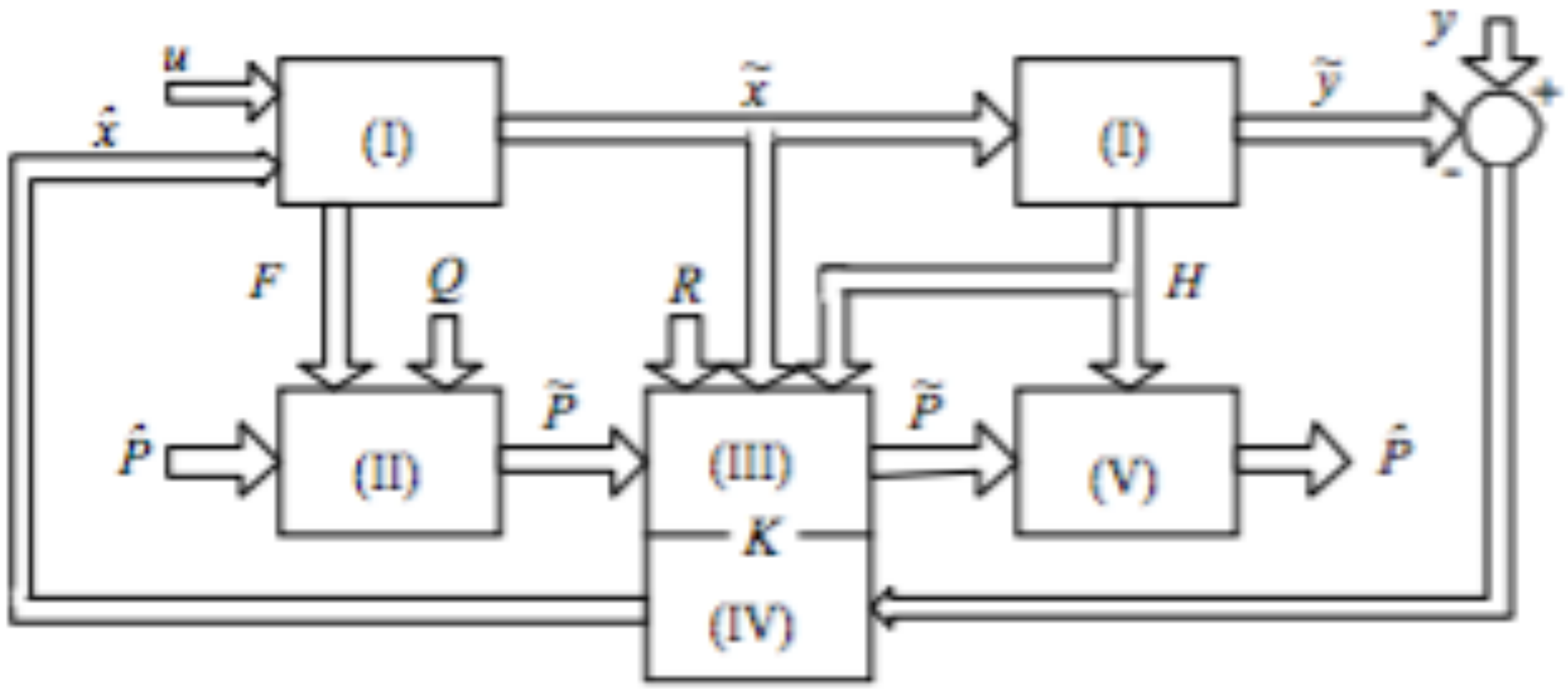
Project 1 | Kalman filter

- **Simplified Kalman filter model**

$$x_k = f(x_{k-1}, k-1) + w_{k-1} \xrightarrow{f(x_{k-1}, k-1) = Fx_{k-1}} x_k = Fx_{k-1} + w_{k-1}$$

$$y_k = h(x_k, k) + v_k \xrightarrow{h(x_k, k) = Hx_k} y_k = Hx_k + v_k$$

- Kalman filter and Extended Kalman filter share the same core part for implementation, so we use the simplified model



- **Computational procedure**

Time update

$$\tilde{x}_k = F\hat{x}_{k-1}$$

$$\tilde{P}_k = F\hat{P}_{k-1}F^T + Q$$

State update

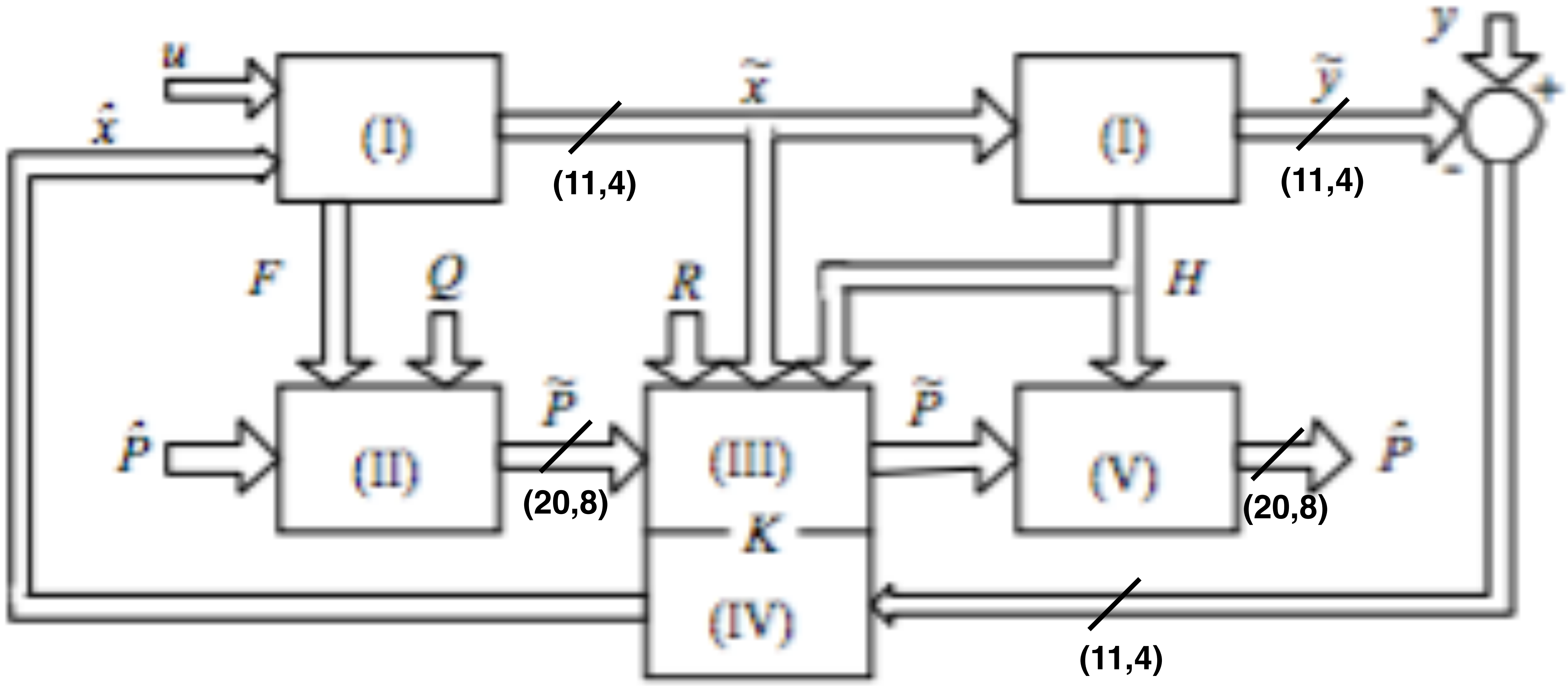
$$K_k = \tilde{P}_k H^T [H\tilde{P}_k H^T + R]^{-1}$$

$$\tilde{y}_k = H\tilde{x}_k$$

$$\hat{x}_k = \tilde{x}_k + K_k(y_k - \tilde{y}_k)$$

$$\hat{P}_k = \tilde{P}_k - K_k H\tilde{P}_k$$

- **Fixed point implementation depending on application**
  - For our simulation, the following fixed point setting gives the same performance as floating point.



- **Simulation results**