In [4]:
```
!pip install numpy==1.26
# Or simply:
# conda install 'numpy<2'
```

```
Collecting numpy==1.26
  Downloading numpy-1.26.0-cp312-cp312-win_amd64.whl.metadata (61 kB)
Downloading numpy-1.26.0-cp312-cp312-win_amd64.whl (15.5 MB)
   ---------------------------------------- 0.0/15.5 MB ? eta -:--:--
   ------ --------------------------------- 2.4/15.5 MB 12.2 MB/s eta 0:00:02
   ------------ --------------------------- 4.7/15.5 MB 11.9 MB/s eta 0:00:01
   ------------------ --------------------- 7.3/15.5 MB 11.6 MB/s eta 0:00:01
   ------------------------- -------------- 10.0/15.5 MB 11.7 MB/s eta 0:00:01
   ------------------------------- -------- 12.3/15.5 MB 11.7 MB/s eta 0:00:01
   ------------------------------------- -- 14.7/15.5 MB 11.7 MB/s eta 0:00:01
   ---------------------------------------- 15.5/15.5 MB 10.6 MB/s eta 0:00:00
Installing collected packages: numpy
  Attempting uninstall: numpy
    Found existing installation: numpy 2.2.6
    Uninstalling numpy-2.2.6:
      Successfully uninstalled numpy-2.2.6
Successfully installed numpy-1.26.0
```

```
  WARNING: Failed to remove contents in a temporary directory 'C:\Users\Student\AppD
ata\Local\anaconda3\Lib\site-packages\~umpy.libs'.
  You can safely remove it manually.
  WARNING: Failed to remove contents in a temporary directory 'C:\Users\Student\AppD
ata\Local\anaconda3\Lib\site-packages\~-mpy'.
  You can safely remove it manually.
ERROR: pip's dependency resolver does not currently take into account all the packag
es that are installed. This behaviour is the source of the following dependency conf
licts.
opencv-python 4.12.0.88 requires numpy<2.3.0,>=2; python_version >= "3.9", but you h
ave numpy 1.26.0 which is incompatible.
```

In [5]:
```python
import numpy as np
import matplotlib.pyplot as plt

# --- System Parameters (Section 5) ---
SYSTEM_BANDWIDTH_MHZ = 12.0  # B_sys in MHz
NUM_USERS_K = 6              # K for FDMA/TDMA comparison
CDMA_PROCESS_GAIN_L = 64     # L for CDMA DSSS
EB_N0_DB = 10                # Eb/N0 in dB for CDMA SIR calculation
EB_N0_LINEAR = 10**(EB_N0_DB / 10)

# Frame and Slot parameters (Conceptual for TDMA)
FRAME_TIME_MS = 6.0          # T_frame (as suggested by Figure 2)
```

In [6]:
```python
# --- FDMA Calculations ---
CHANNEL_BANDWIDTH_MHZ = SYSTEM_BANDWIDTH_MHZ / NUM_USERS_K  # 12 MHz / 6 = 2 MHz
# Assuming a Guard Band of 0.2 MHz per channel (200 kHz) for illustration
MESSAGE_BANDWIDTH_MHZ = 1.8
GUARD_BAND_MHZ = CHANNEL_BANDWIDTH_MHZ - MESSAGE_BANDWIDTH_MHZ # 0.2 MHz

TOTAL_GUARD_BAND_MHZ = GUARD_BAND_MHZ * NUM_USERS_K
GUARD_BAND_OVERHEAD_FRACTION = TOTAL_GUARD_BAND_MHZ / SYSTEM_BANDWIDTH_MHZ
```

```
print("\n--- FDMA Analysis ---")
print(f"B_sys: {SYSTEM_BANDWIDTH_MHZ} MHz, K: {NUM_USERS_K}")
print(f"Total Guard Bandwidth: {TOTAL_GUARD_BAND_MHZ:.1f} MHz")
print(f"Guard-Band Overhead (Q1 Data): {GUARD_BAND_OVERHEAD_FRACTION*100:.1f}%")

# (SS 3) Console Output: Printout of the calculated Guard-Band Overhead
```

```
--- FDMA Analysis ---
B_sys: 12.0 MHz, K: 6
Total Guard Bandwidth: 1.2 MHz
Guard-Band Overhead (Q1 Data): 10.0%
```

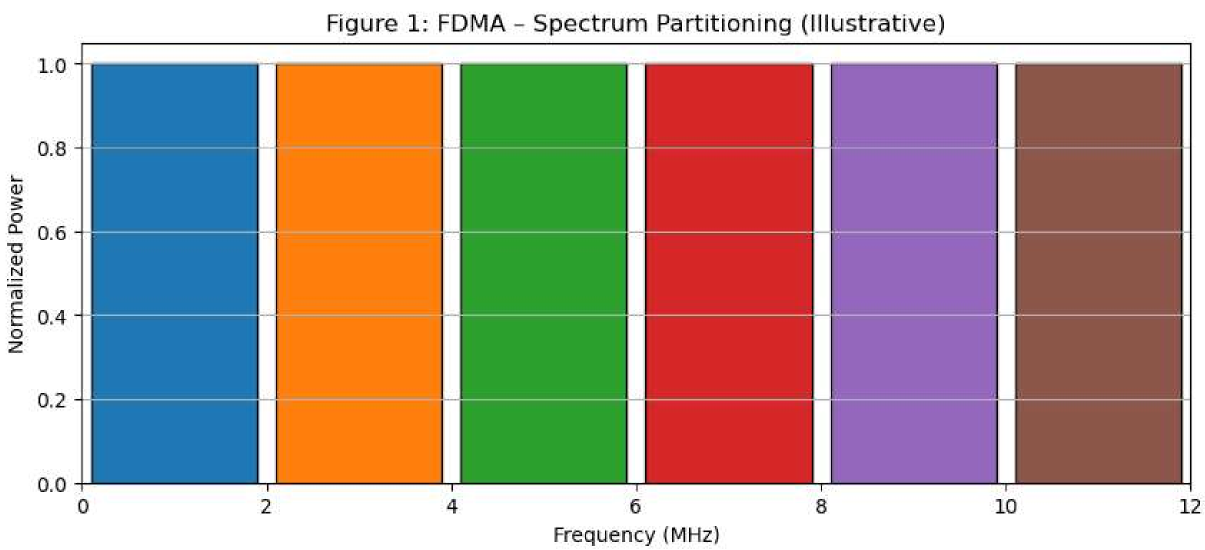
In [7]:

```
# --- FDMA Visualization (Figure 1) ---
plt.figure(figsize=(10, 4))
frequencies = np.arange(0, SYSTEM_BANDWIDTH_MHZ, CHANNEL_BANDWIDTH_MHZ)

for i, freq_start in enumerate(frequencies):
    # Plotting the main channel band
    plt.bar(freq_start + CHANNEL_BANDWIDTH_MHZ/2, 1.0, width=MESSAGE_BANDWIDTH_MHZ,
            bottom=0, align='center', edgecolor='black',
            label=f'User {i+1} Band' if i==0 else "")
    # Conceptual visualization of Guard Bands could be a small gap or different col

plt.title('Figure 1: FDMA – Spectrum Partitioning (Illustrative)')
plt.xlabel('Frequency (MHz)')
plt.ylabel('Normalized Power')
plt.xlim(0, SYSTEM_BANDWIDTH_MHZ)
plt.grid(axis='y')
plt.show()

# (SS 2) Figure 1 Printout: Screenshot of the FDMA plot
```



In [8]:

```
# --- TDMA Calculations ---
TIME_PER_SLOT_MS = FRAME_TIME_MS / NUM_USERS_K # 6 ms / 6 = 1 ms
# Assuming a Timing Overhead of 0.05 ms (50 us) per slot for synchronization
GUARD_TIME_MS = 0.05
DATA_TIME_MS = TIME_PER_SLOT_MS - GUARD_TIME_MS # 0.95 ms
```

```
# Max delay is conceptually the frame time (ignoring half-duplex)
MAX_USER_DELAY_MS = FRAME_TIME_MS
USER_DUTY_CYCLE = TIME_PER_SLOT_MS / FRAME_TIME_MS # 1/6

print("\n--- TDMA Analysis ---")
print(f"T_frame: {FRAME_TIME_MS} ms, K: {NUM_USERS_K}")
print(f"Max User Delay (Q3 Data): {MAX_USER_DELAY_MS:.1f} ms")
print(f"User Duty Cycle: {USER_DUTY_CYCLE:.2f}")

# (SS 6) Console Output: Printout of the calculated Max User Delay and Duty Cycle
```

```
--- TDMA Analysis ---
T_frame: 6.0 ms, K: 6
Max User Delay (Q3 Data): 6.0 ms
User Duty Cycle: 0.17
```

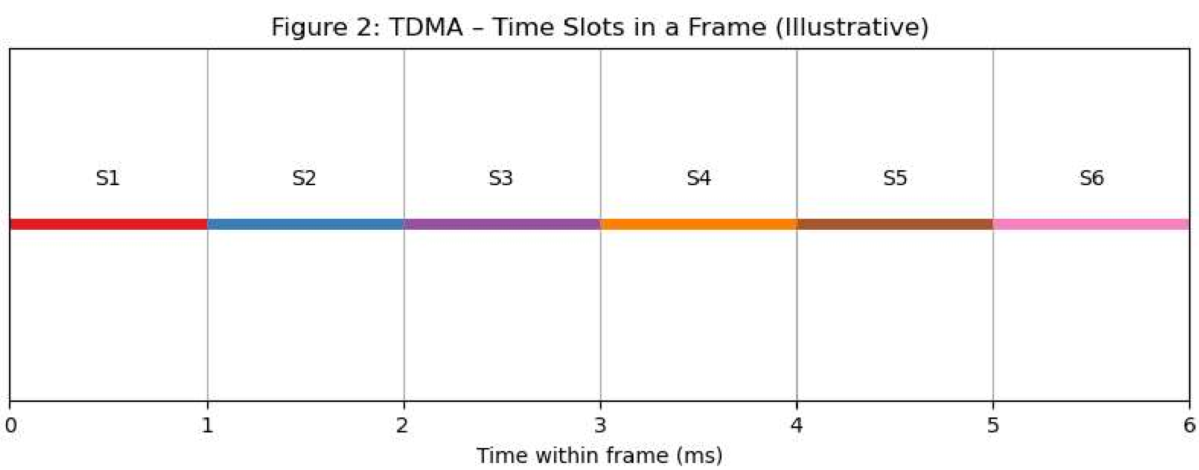
In [9]:

```
# --- TDMA Visualization (Figure 2) ---
plt.figure(figsize=(10, 3))

for i in range(NUM_USERS_K):
    slot_start = i * TIME_PER_SLOT_MS
    # Visualize the slot as a horizontal line segment
    plt.hlines(y=0.5, xmin=slot_start, xmax=slot_start + TIME_PER_SLOT_MS,
               color=plt.cm.Set1(i/NUM_USERS_K), linewidth=5)
    plt.text(slot_start + TIME_PER_SLOT_MS/2, 0.6, f'S{i+1}',
             ha='center', va='bottom')

plt.title('Figure 2: TDMA – Time Slots in a Frame (Illustrative)')
plt.xlabel('Time within frame (ms)')
plt.yticks([]) # Hide y-axis
plt.xlim(0, FRAME_TIME_MS)
plt.ylim(0, 1)
plt.grid(axis='x')
plt.show()

# (SS 5) Figure 2 Printout: Screenshot of the TDMA plot
```



In [10]:

```
# --- CDMA Processing Gain Definition (Q2) ---
PROCESSING_GAIN_DB = 10 * np.log10(CDMA_PROCESS_GAIN_L)

print("\n--- CDMA Analysis ---")
print(f"Processing Gain L: {CDMA_PROCESS_GAIN_L}")
```

```
print(f"Processing Gain in dB (Q2 Data): {PROCESSING_GAIN_DB:.2f} dB")
# (SS 8) Console Output: Definition and calculation of Processing Gain L
```

```
--- CDMA Analysis ---
Processing Gain L: 64
Processing Gain in dB (Q2 Data): 18.06 dB
```

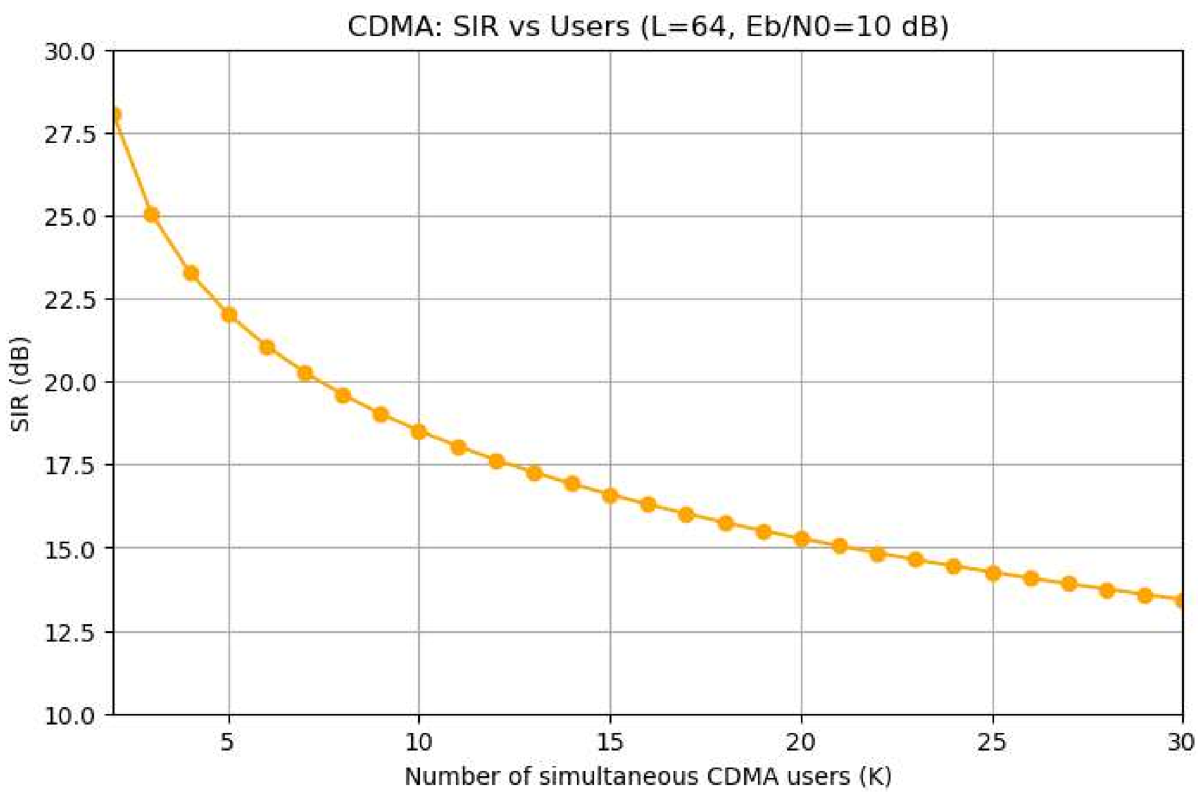
In [11]:
```
# --- CDMA SIR Calculation and Visualization (Figure 4) ---
NUM_USERS_RANGE = np.arange(2, 31) # K from 2 to 30

# Calculate SIR using the simplified model (linear scale)
# SIR = (L / (K - 1)) * (Eb/N0)
SIR_LINEAR = (CDMA_PROCESS_GAIN_L / (NUM_USERS_RANGE - 1)) * EB_N0_LINEAR

# Convert to dB
SIR_DB = 10 * np.log10(SIR_LINEAR)

plt.figure(figsize=(8, 5))
plt.plot(NUM_USERS_RANGE, SIR_DB, marker='o', linestyle='-', color='orange')
plt.title(f'CDMA: SIR vs Users (L={CDMA_PROCESS_GAIN_L}, Eb/N0={EB_N0_DB} dB)')
plt.xlabel('Number of simultaneous CDMA users (K)')
plt.ylabel('SIR (dB)')
plt.grid(True)
plt.xlim(2, 30)
plt.ylim(10, 30)
plt.show()

# (SS 9) Figure 4 Printout: Screenshot of the SIR vs Users plot
```

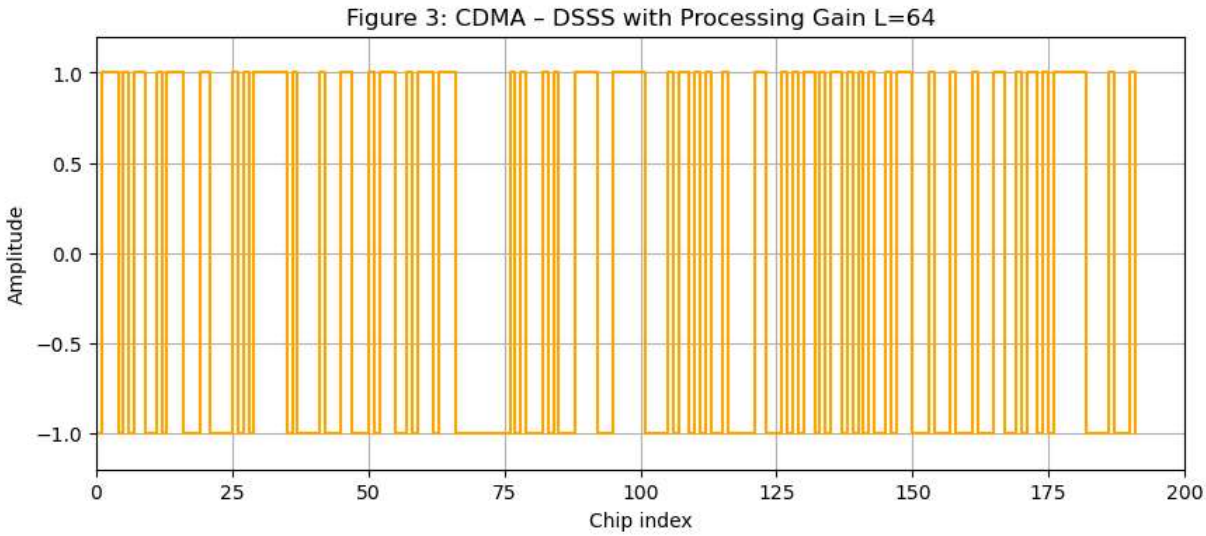


In [12]:
```
# --- DSSS Spreading Visualization (Figure 3) ---
# Create a conceptual spreading sequence (e.g., a random binary sequence of length
SPREAD_SEQUENCE = np.random.choice([-1, 1], size=CDMA_PROCESS_GAIN_L * 3)
```

```
CHIP_INDEX = np.arange(len(SPREAD_SEQUENCE))

plt.figure(figsize=(10, 4))
plt.step(CHIP_INDEX, SPREAD_SEQUENCE, where='post', color='orange')
plt.title(f'Figure 3: CDMA – DSSS with Processing Gain L={CDMA_PROCESS_GAIN_L}')
plt.xlabel('Chip index')
plt.ylabel('Amplitude')
plt.ylim(-1.2, 1.2)
plt.xlim(0, 200)
plt.grid(True)
plt.show()

# (SS 7) Figure 3 Printout: Screenshot of the DSSS spreading example
```



In [ ]:

In [ ]: