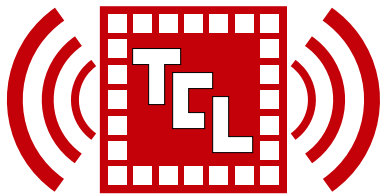
LLR-Based Successive Cancellation List Decoding of Polar Codes

Alexios Balatsoukas-Stimming,[1] Mani Bastani Parizi,[2] Andreas Burg[1]

1: Telecommunications Circuits Laboratory, EPFL
2: Information Theory Laboratory, EPFL



ICASSP 2014, Florence

Arıkan's Polar Coding Paradigm

$\mathcal{A} = \{3, 5, 6, 7\}$

0
1
2
3
4
5
6
7

- Construction:
  - Information Indices: $\mathcal{A} \subset \{0, 1, \ldots, N-1\}$, $|\mathcal{A}| = NR$, $N = 2^n$.

Arıkan's Polar Coding Paradigm

$$\mathbf{u} = \begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \\ u_6 \\ u_7 \end{bmatrix}$$

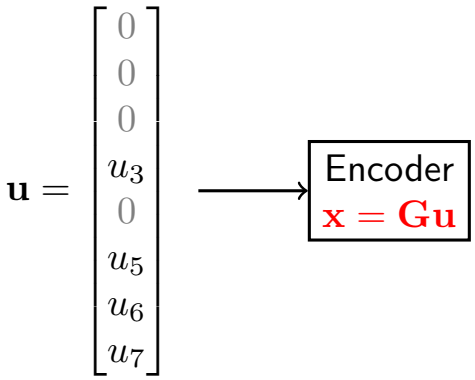
- Construction:
  - Information Indices: $\mathcal{A} \subset \{0, 1, \ldots, N-1\}$, $|\mathcal{A}| = NR$, $N = 2^n$.
- Encoding:
  - $\mathbf{u}_{\mathcal{A}} \triangleq [u_i, i \in \mathcal{A}]^T \leftarrow$ data bits,

Arıkan's Polar Coding Paradigm

$$\mathbf{u} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ u_3 \\ 0 \\ u_5 \\ u_6 \\ u_7 \end{bmatrix}$$

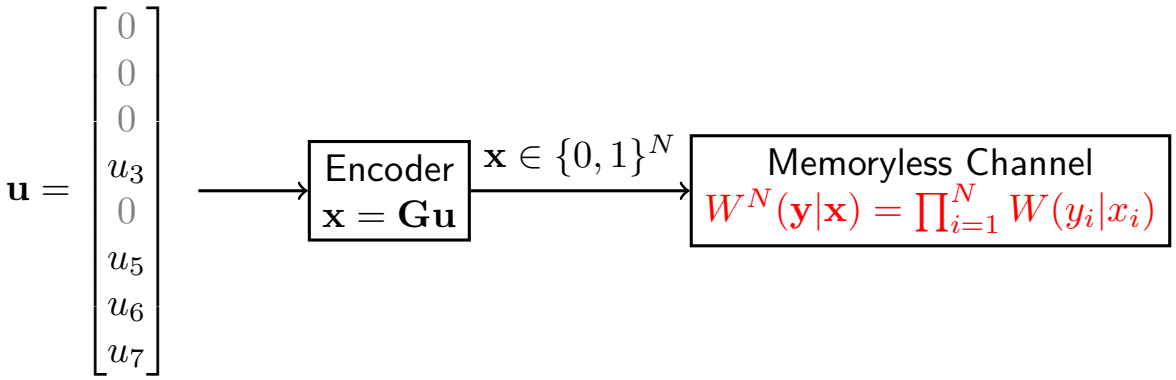
- Construction:
  - Information Indices: $\mathcal{A} \subset \{0, 1, \ldots, N-1\}$, $|\mathcal{A}| = NR$, $N = 2^n$.
- Encoding:
  - $\mathbf{u}_{\mathcal{A}} \triangleq [u_i, i \in \mathcal{A}]^T \leftarrow$ data bits,
  - $\mathbf{u}_{\mathcal{A}^C} \leftarrow$ known-to-receiver frozen bits (say all-zero).

Arıkan's Polar Coding Paradigm



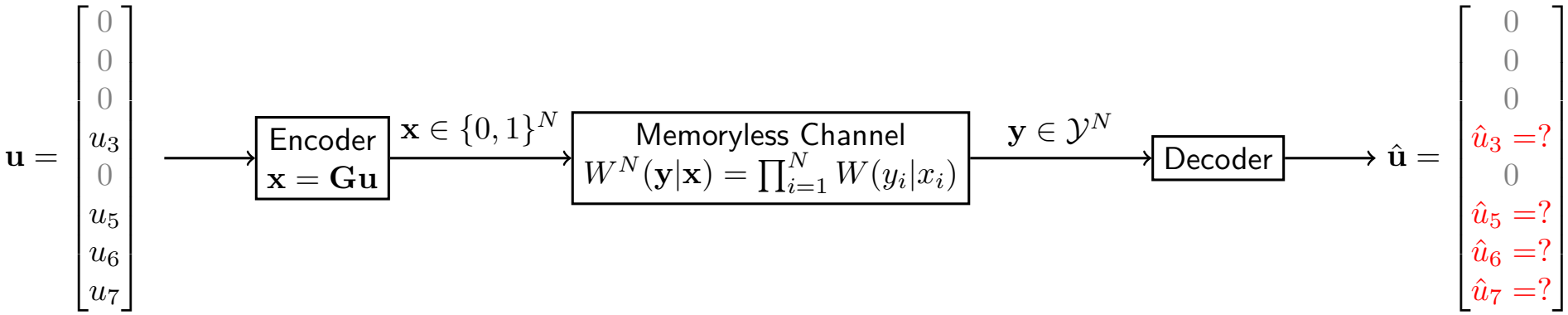
- Construction:
  - Information Indices: $\mathcal{A} \subset \{0, 1, \ldots, N-1\}$, $|\mathcal{A}| = NR$, $N = 2^n$.
- Encoding:
  - $\mathbf{u}_{\mathcal{A}} \triangleq [u_i, i \in \mathcal{A}]^T \leftarrow$ data bits,
  - $\mathbf{u}_{\mathcal{A}^C} \leftarrow$ known-to-receiver frozen bits (say all-zero).
  - $\mathbf{x} \leftarrow \mathbf{G}\mathbf{u}$ (using $O(N \log N)$ binary additions).

Arıkan's Polar Coding Paradigm



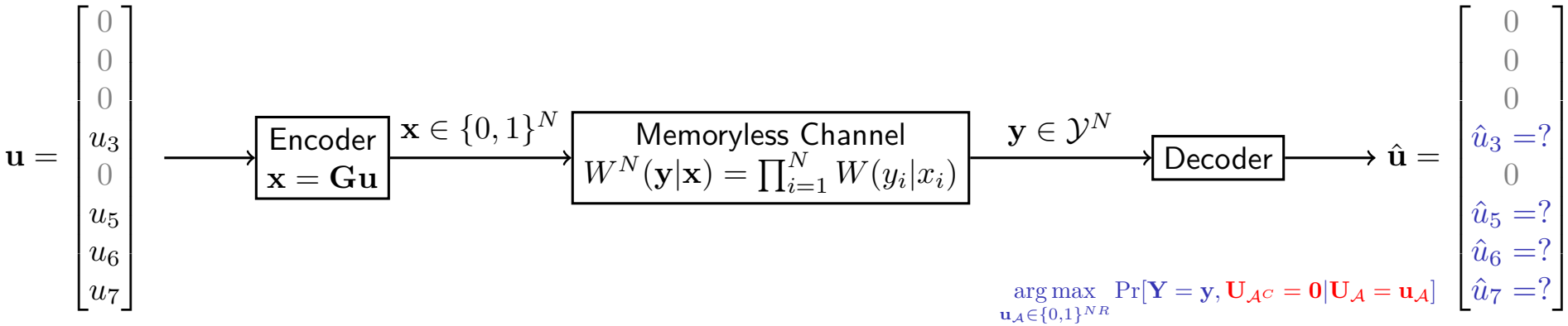
- Construction:
  - Information Indices: $\mathcal{A} \subset \{0, 1, \ldots, N-1\}$, $|\mathcal{A}| = NR$, $N = 2^n$.
- Encoding:
  - $\mathbf{u}_{\mathcal{A}} \triangleq [u_i, i \in \mathcal{A}]^T \leftarrow$ data bits,
  - $\mathbf{u}_{\mathcal{A}^C} \leftarrow$ known-to-receiver frozen bits (say all-zero).
  - $\mathbf{x} \leftarrow \mathbf{G}\mathbf{u}$ (using $O(N \log N)$ binary additions).

Arıkan's Polar Coding Paradigm



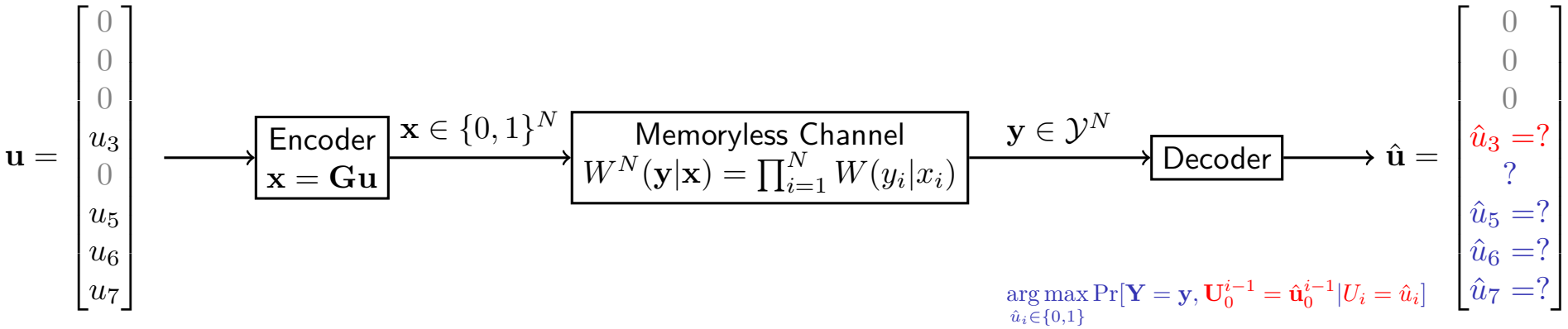
- Construction:
  - Information Indices: $\mathcal{A} \subset \{0, 1, \ldots, N-1\}$, $|\mathcal{A}| = NR$, $N = 2^n$.
- Encoding:
  - $\mathbf{u}_{\mathcal{A}} \triangleq [u_i, i \in \mathcal{A}]^T \leftarrow$ data bits,
  - $\mathbf{u}_{\mathcal{A}^C} \leftarrow$ known-to-receiver frozen bits (say all-zero).
  - $\mathbf{x} \leftarrow \mathbf{G}\mathbf{u}$ (using $O(N \log N)$ binary additions).
- Decoding: Estimate the information bits $\hat{\mathbf{u}}_{\mathcal{A}}$.

Arıkan's Polar Coding Paradigm



- Construction:
  - Information Indices: $\mathcal{A} \subset \{0, 1, \ldots, N-1\}$, $|\mathcal{A}| = NR$, $N = 2^n$.
- Encoding:
  - $\mathbf{u}_\mathcal{A} \triangleq [u_i, i \in \mathcal{A}]^T \leftarrow$ data bits,
  - $\mathbf{u}_{\mathcal{A}^C} \leftarrow$ known-to-receiver frozen bits (say all-zero).
  - $\mathbf{x} \leftarrow \mathbf{G}\mathbf{u}$ (using $O(N \log N)$ binary additions).
- Decoding: Estimate the information bits $\hat{\mathbf{u}}_\mathcal{A}$.
  - (Optimal) ML Decoding: Consider all 2^{NR} possible paths on the binary tree of height N and choose the best $\rightarrow$ **Too complex!**

Arıkan's Polar Coding Paradigm



- Construction:
  - Information Indices: $\mathcal{A} \subset \{0, 1, \ldots, N-1\}$, $|\mathcal{A}| = NR$, $N = 2^n$.
- Encoding:
  - $\mathbf{u}_{\mathcal{A}} \triangleq [u_i, i \in \mathcal{A}]^T \leftarrow$ data bits,
  - $\mathbf{u}_{\mathcal{A}^C} \leftarrow$ known-to-receiver frozen bits (say all-zero).
  - $\mathbf{x} \leftarrow \mathbf{Gu}$ (using $O(N \log N)$ binary additions).
- Decoding: Estimate the information bits $\hat{\mathbf{u}}_{\mathcal{A}}$.
  - (Optimal) ML Decoding: Consider all 2^{NR} possible paths on the binary tree of height N and choose the best $\rightarrow$ **Too complex!**
  - (Suboptimal) Successive Cancellation (SC) Decoding: At each level $i \in \mathcal{A}$, choose the best possible value of u_i given the past estimations and frozen bits.

SC Decoding and SCL Decoding in Details ...

for $i = 0, 1, \ldots, N-1$ **do** // Start from the root and go down...
  if $i \notin \mathcal{A}$ **then**
    $\hat{u}_i \leftarrow 0$;
  else
    $\hat{u}_i \leftarrow \underset{u_i \in \{0,1\}}{\arg\max}\, W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1} | u_i)$;

return $\hat{\mathbf{u}}_{\mathcal{A}}$;

SC Decoding and SCL Decoding in Details ...

for $i = 0, 1, \ldots, N-1$ **do** // Start from the root and go down...

  if $i \notin \mathcal{A}$ **then**

    $\hat{u}_i \leftarrow 0$; // If i is a frozen index the you know where to go

  else

    $\hat{u}_i \leftarrow \underset{u_i \in \{0,1\}}{\arg\max}\, W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1} | u_i)$;

return $\hat{\mathbf{u}}_{\mathcal{A}}$;

SC Decoding and SCL Decoding in Details ...

for $i = 0, 1, \ldots, N-1$ **do** // Start from the root and go down...

  if $i \notin \mathcal{A}$ **then**

    $\hat{u}_i \leftarrow 0$; // If i is a frozen index the you know where to go

  else

    $\hat{u}_i \leftarrow \underset{u_i \in \{0,1\}}{\arg\max}\, W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1} | u_i)$; // Follow the best direction given the past

return $\hat{\mathbf{u}}_{\mathcal{A}}$;

SC Decoding and SCL Decoding in Details ...

for $i = 0, 1, \ldots, N-1$ **do** // Start from the root and go down...

  if $i \notin \mathcal{A}$ **then**

    $\hat{u}_i \leftarrow 0$; // If i is a frozen index the you know where to go

  else

    $\hat{u}_i \leftarrow \underset{u_i \in \{0,1\}}{\arg\max}\, W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1} | u_i)$; // Follow the best direction given the past

return $\hat{\mathbf{u}}_{\mathcal{A}}$;

SC Decoding and SCL Decoding in Details ...

for $i = 0, 1, \ldots, N-1$ **do** // Start from the root and go down...
  if $i \notin \mathcal{A}$ **then**
    $\hat{u}_i \leftarrow 0$; // If i is a frozen index the you know where to go
  else
    $\hat{u}_i \leftarrow \underset{u_i \in \{0,1\}}{\arg\max}\, W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1} | u_i)$; // Follow the best direction given the past

return $\hat{\mathbf{u}}_{\mathcal{A}}$;

- Sub-optimal, but still asymptotically powerful; $P_e \approx O(2^{-\sqrt{N}})$ [Arıkan and Telatar'09] $\Rightarrow$ Polar codes are *capacity-achieving*.

SC Decoding and SCL Decoding in Details ...

for $i = 0, 1, \ldots, N-1$ **do** // Start from the root and go down...
  if $i \notin \mathcal{A}$ **then**
    $\hat{u}_i \leftarrow 0$; // If i is a frozen index the you know where to go
  else
    $\hat{u}_i \leftarrow \underset{u_i \in \{0,1\}}{\arg\max}\, W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1} | u_i)$; // Follow the best direction given the past

return $\hat{\mathbf{u}}_{\mathcal{A}}$;

- Sub-optimal, but still asymptotically powerful; $P_e \approx O(2^{-\sqrt{N}})$ [Arıkan and Telatar'09] $\Rightarrow$ Polar codes are *capacity-achieving*.
- Efficient: $O(N \log N)$ arithmetic operations [Arıkan'09]

SC Decoding and SCL Decoding in Details ...

for $i = 0, 1, \ldots, N-1$ **do** // Start from the root and go down...
  if $i \notin \mathcal{A}$ **then**
    $\hat{u}_i \leftarrow 0$; // If i is a frozen index the you know where to go
  else
    $\hat{u}_i \leftarrow \underset{u_i \in \{0,1\}}{\arg\max}\, W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1} | u_i)$; // Follow the best direction given the past

return $\hat{\mathbf{u}}_{\mathcal{A}}$;

- Sub-optimal, but still asymptotically powerful; $P_e \approx O(2^{-\sqrt{N}})$ [Arıkan and Telatar'09] $\Rightarrow$ Polar codes are *capacity-achieving*.
- Efficient: $O(N \log N)$ arithmetic operations [Arıkan'09]
- Successive Cancellation List (SCL) decoding to improve the poor finite block-length performance [Tal and Vardy'11].

SC Decoding and SCL Decoding in Details ...

for $i = 0, 1, \ldots, N-1$ **do** // Go down like before ...
  if $i \notin \mathcal{A}$ **then**
    $\hat{u}_i[\ell] \leftarrow u_i, \forall \ell \in \{0, 1, \ldots, L-1\}$; // Frozen bits are the same for all L paths
  else // Continue along the L best paths on the tree
    if *Less than L paths are active* **then**
      Duplicate all the paths and continue with both possible values of u_i;
    else
      Sort the likelihoods $\left\{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1}[\ell]|u_i) : \forall \ell \in \{0, 1, \ldots, L-1\}, \forall u_i \in \{0, 1\}\right\}$;
      Continue along the L most likely paths;
$\ell^* \leftarrow$ The index of the most-likely path;
return $\hat{\mathbf{u}}_{\mathcal{A}}[\ell^*]$;

- Sub-optimal, but still asymptotically powerful; $P_e \approx O(2^{-\sqrt{N}})$ [Arıkan and Telatar'09] $\Rightarrow$ Polar codes are *capacity-achieving*.
- Efficient: $O(N \log N)$ arithmetic operations [Arıkan'09]
- Successive Cancellation List (SCL) decoding to improve the poor finite block-length performance [Tal and Vardy'11].
  - Idea: keep track of L paths on the binary tree concurrently.

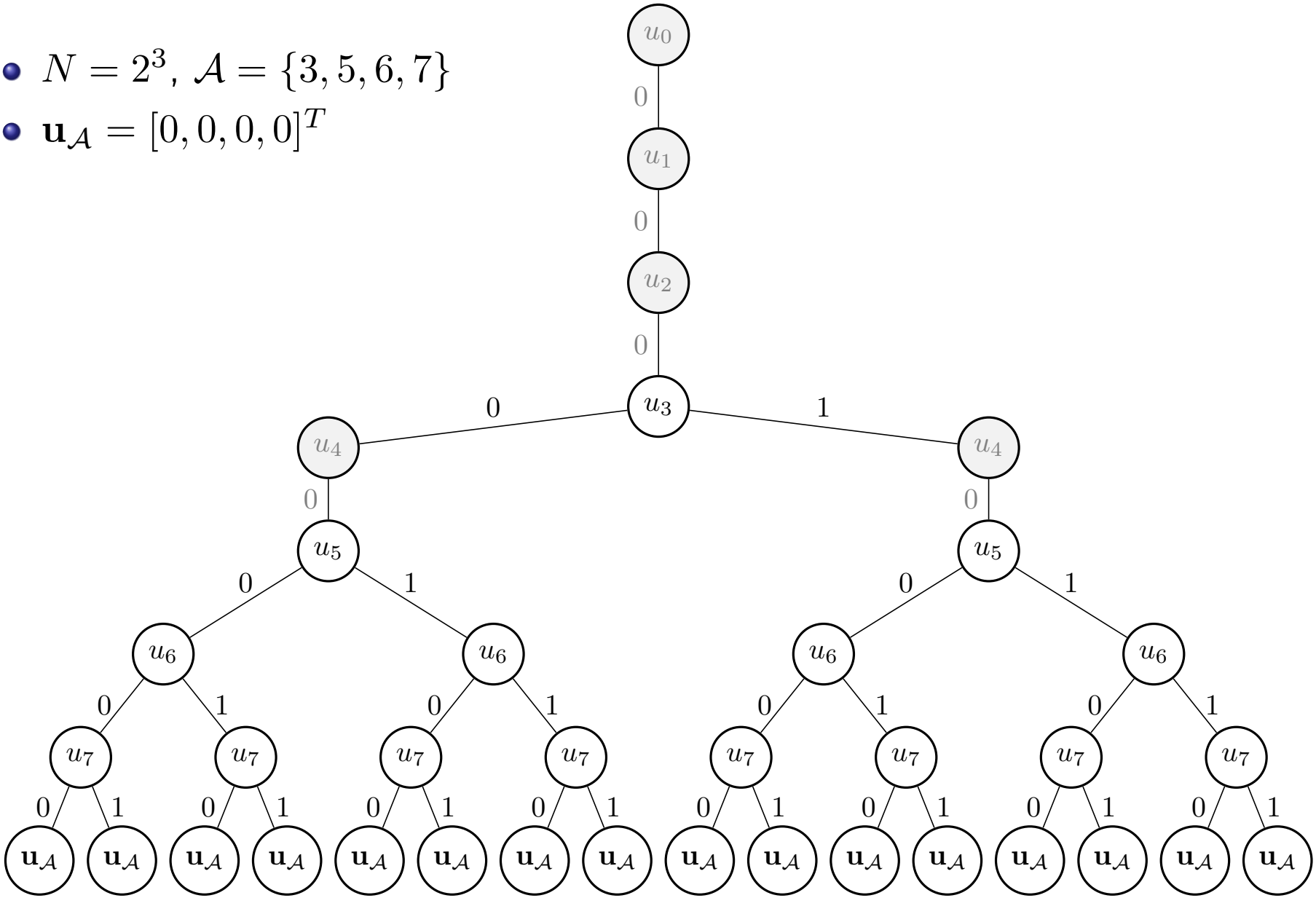
SC Decoding and SCL Decoding in Details ...

for $i = 0, 1, \ldots, N-1$ **do** // Go down like before ...
  if $i \notin \mathcal{A}$ **then**
    $\hat{u}_i[\ell] \leftarrow u_i, \forall \ell \in \{0, 1, \ldots, L-1\}$; // Frozen bits are the same for all L paths
  else // Continue along the L best paths on the tree
    if *Less than L paths are active* **then**
      Duplicate all the paths and continue with both possible values of u_i;
    else
      Sort the likelihoods $\left\{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1}[\ell]|u_i) : \forall \ell \in \{0, 1, \ldots, L-1\}, \forall u_i \in \{0, 1\}\right\}$;
      Continue along the L most likely paths;
$\ell^* \leftarrow$ The index of the most-likely path;
return $\hat{\mathbf{u}}_{\mathcal{A}}[\ell^*]$;

- Sub-optimal, but still asymptotically powerful; $P_e \approx O(2^{-\sqrt{N}})$ [Arıkan and Telatar'09] $\Rightarrow$ Polar codes are *capacity-achieving*.
- Efficient: $O(N \log N)$ arithmetic operations [Arıkan'09]
- Successive Cancellation List (SCL) decoding to improve the poor finite block-length performance [Tal and Vardy'11].
  - Idea: keep track of L paths on the binary tree concurrently.
  - Time complexity: $O(L\, N \log N)$.

Example of SC versus SCL Decoding

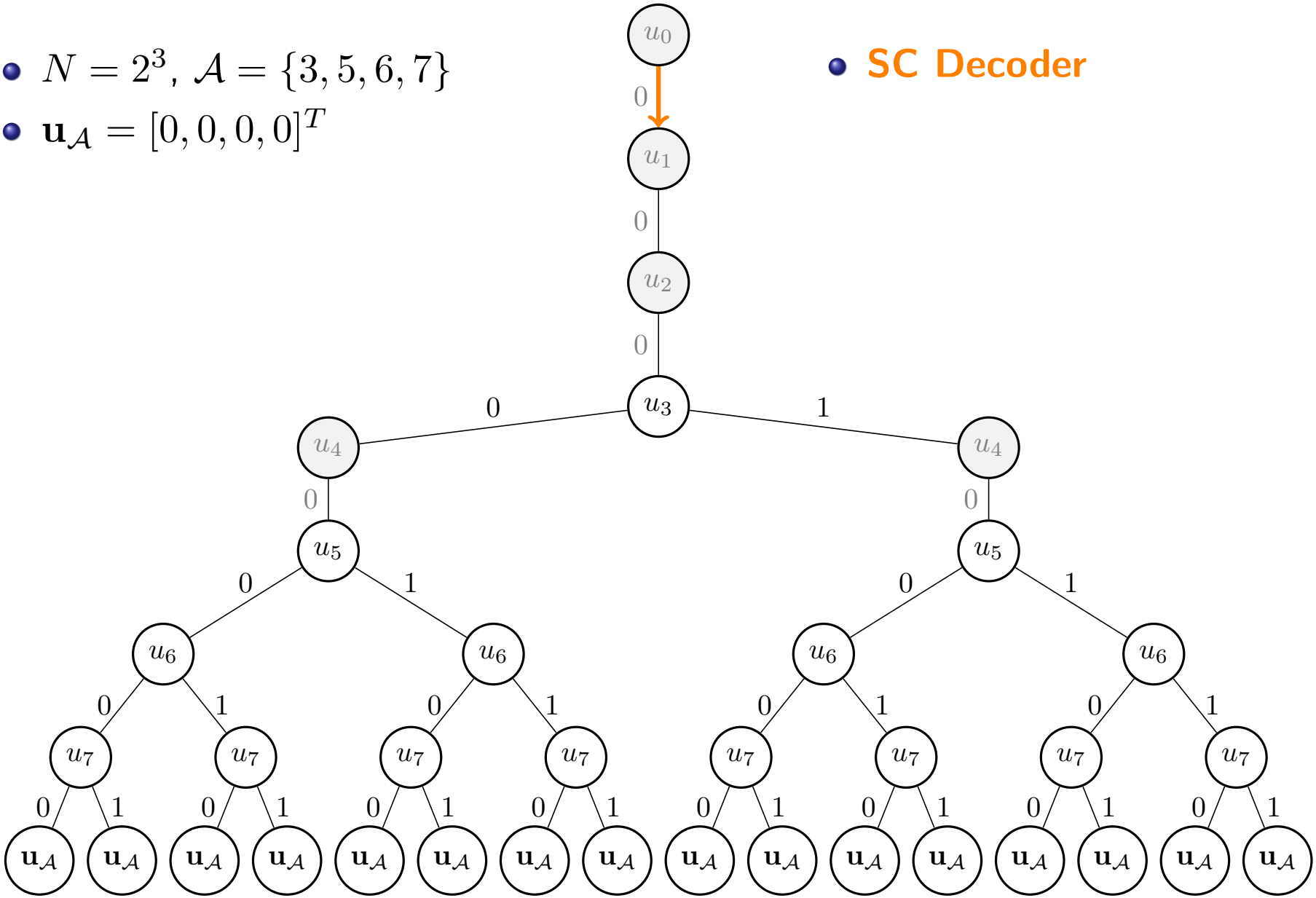
- $N = 2^3$, $\mathcal{A} = \{3, 5, 6, 7\}$
- $\mathbf{u}_{\mathcal{A}} = [0, 0, 0, 0]^T$



Example of SC versus SCL Decoding

- $N = 2^3$, $\mathcal{A} = \{3, 5, 6, 7\}$
- $\mathbf{u}_{\mathcal{A}} = [0, 0, 0, 0]^T$

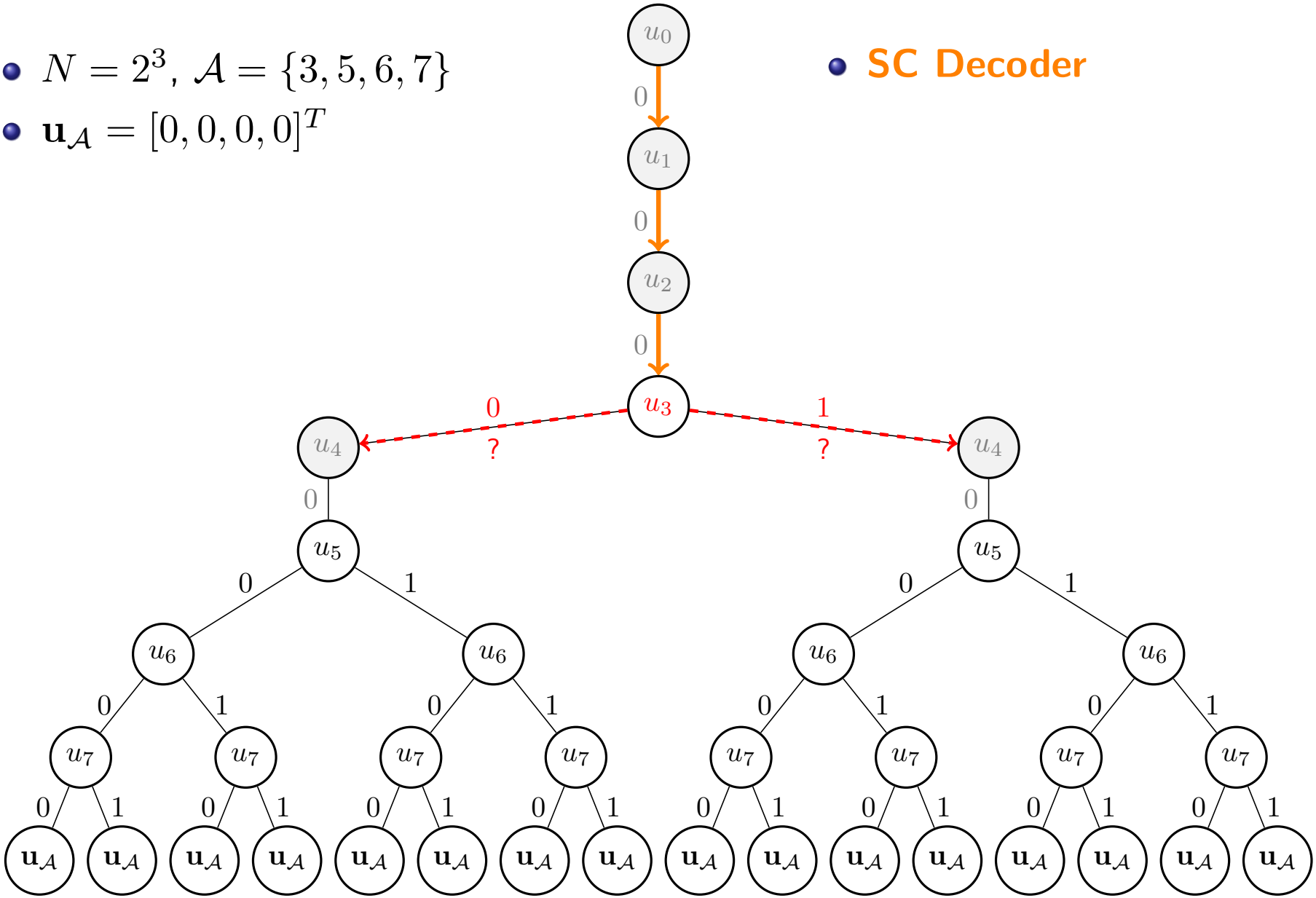
- **SC Decoder**



Example of SC versus SCL Decoding

- $N = 2^3$, $\mathcal{A} = \{3, 5, 6, 7\}$
- $\mathbf{u}_{\mathcal{A}} = [0, 0, 0, 0]^T$

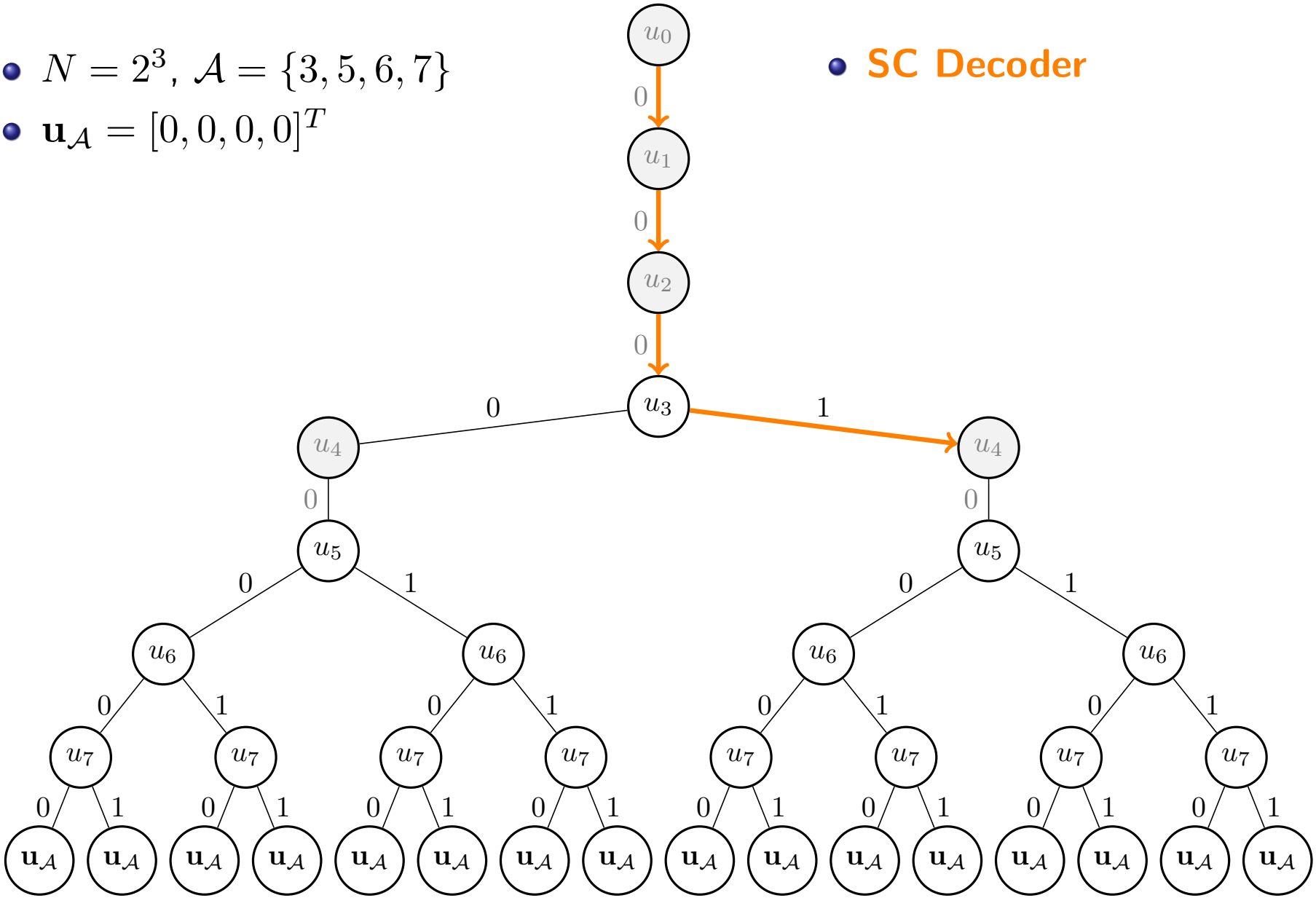
- **SC Decoder**



Example of SC versus SCL Decoding

- $N = 2^3$, $\mathcal{A} = \{3, 5, 6, 7\}$
- $\mathbf{u}_{\mathcal{A}} = [0, 0, 0, 0]^T$

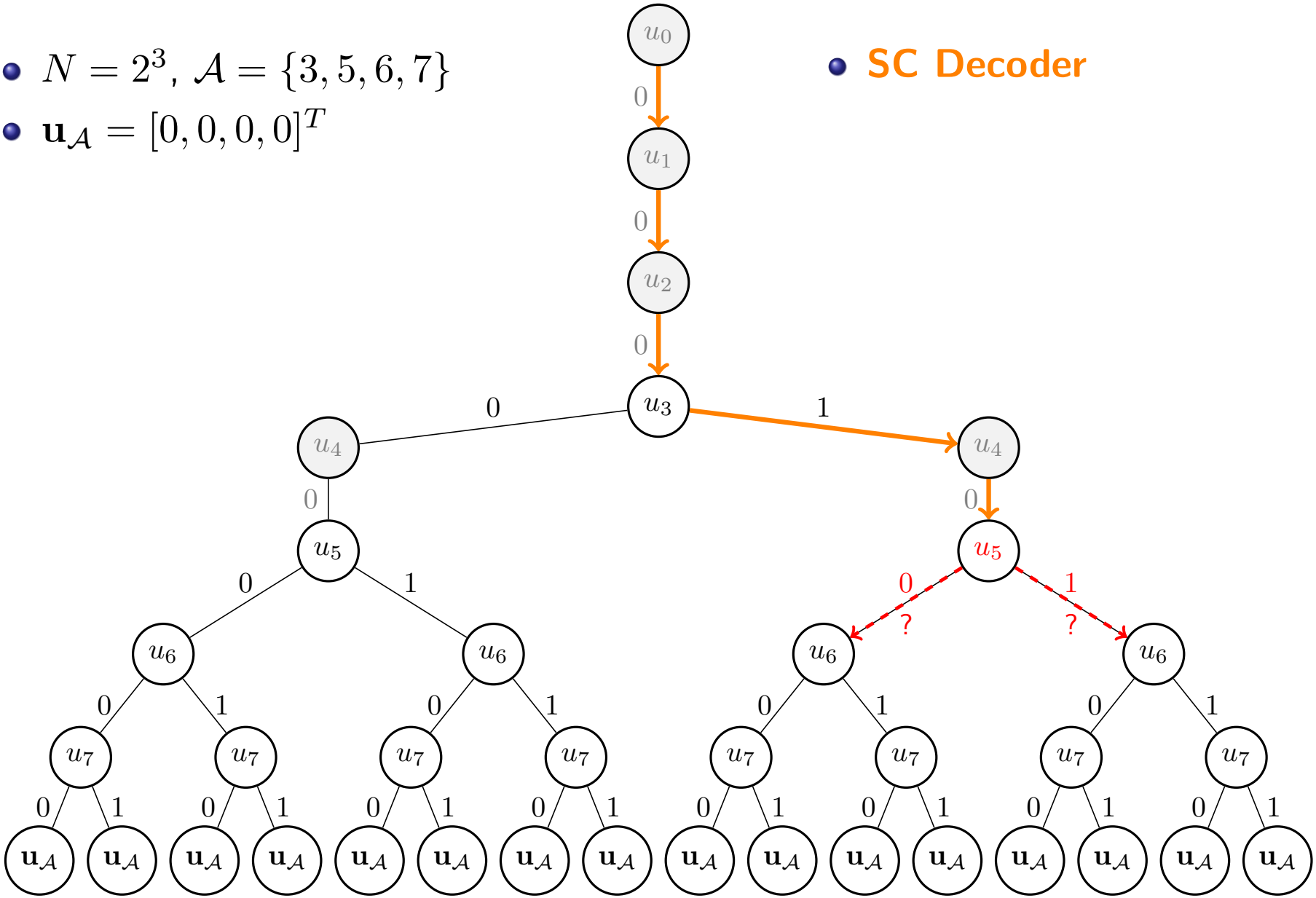
- **SC Decoder**



Example of SC versus SCL Decoding

- $N = 2^3$, $\mathcal{A} = \{3, 5, 6, 7\}$
- $\mathbf{u}_{\mathcal{A}} = [0, 0, 0, 0]^T$

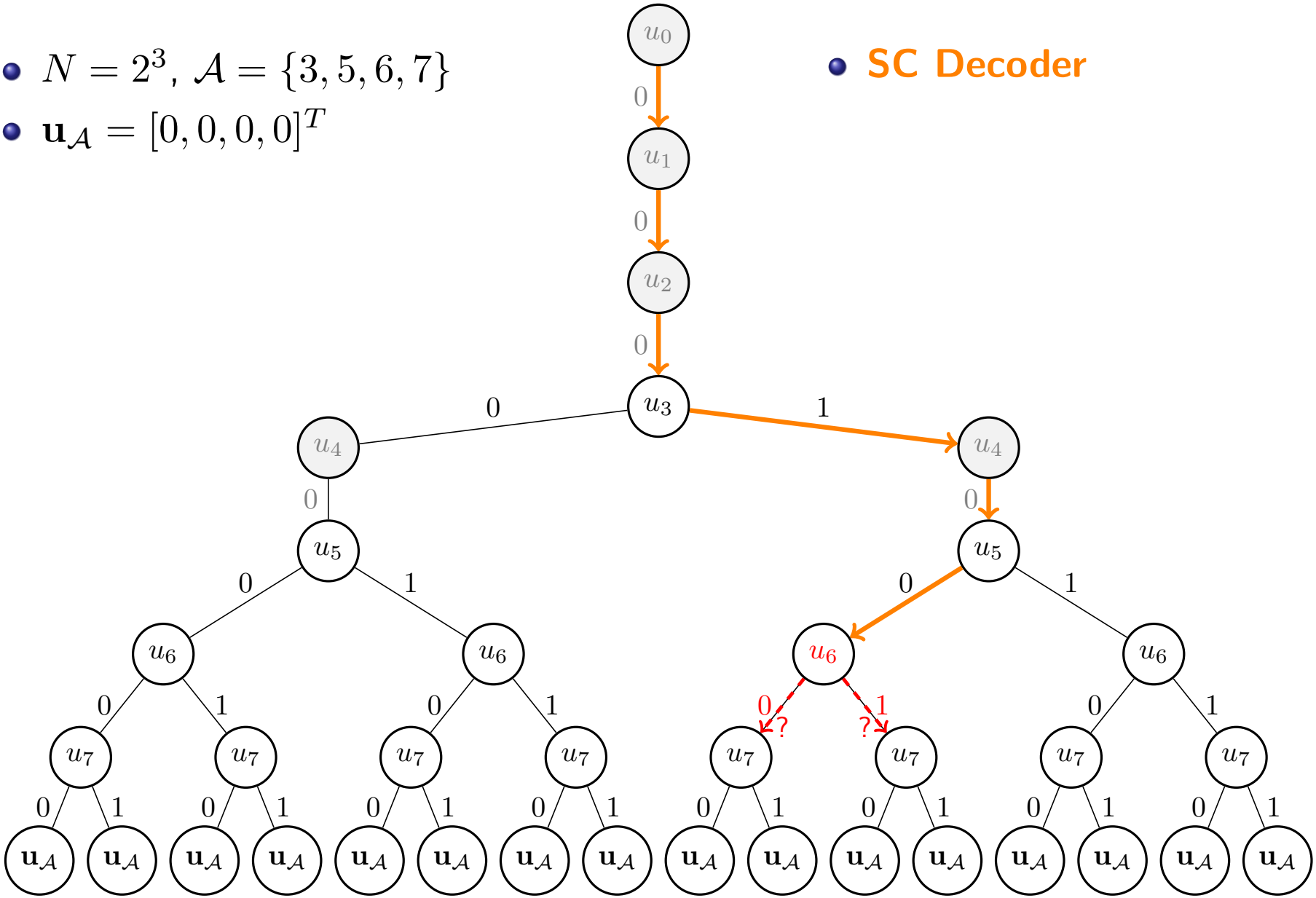
- **SC Decoder**



Example of SC versus SCL Decoding

- $N = 2^3$, $\mathcal{A} = \{3, 5, 6, 7\}$
- $\mathbf{u}_\mathcal{A} = [0, 0, 0, 0]^T$

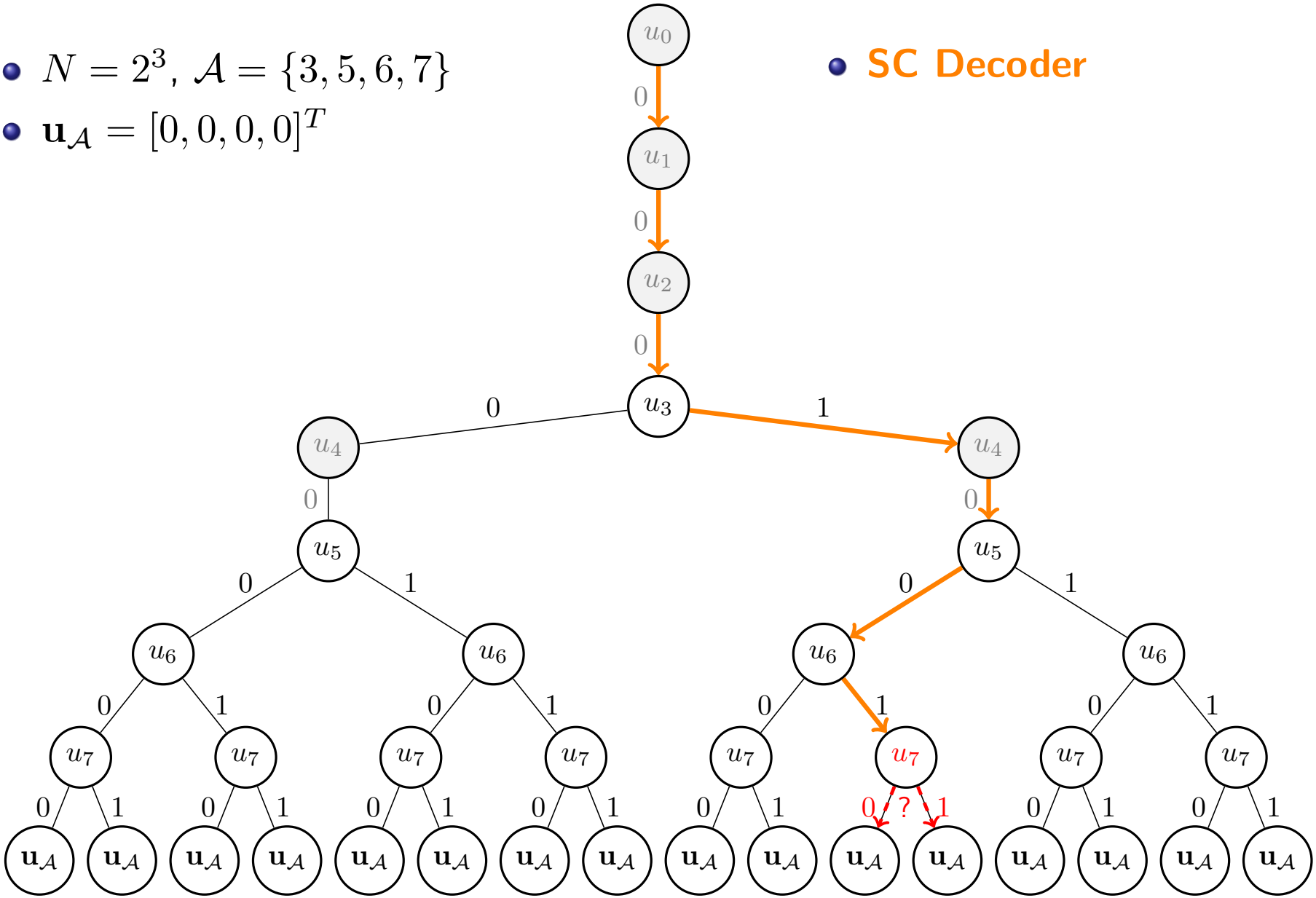
- **SC Decoder**



Example of SC versus SCL Decoding

- $N = 2^3$, $\mathcal{A} = \{3, 5, 6, 7\}$
- $\mathbf{u}_{\mathcal{A}} = [0, 0, 0, 0]^T$

- **SC Decoder**

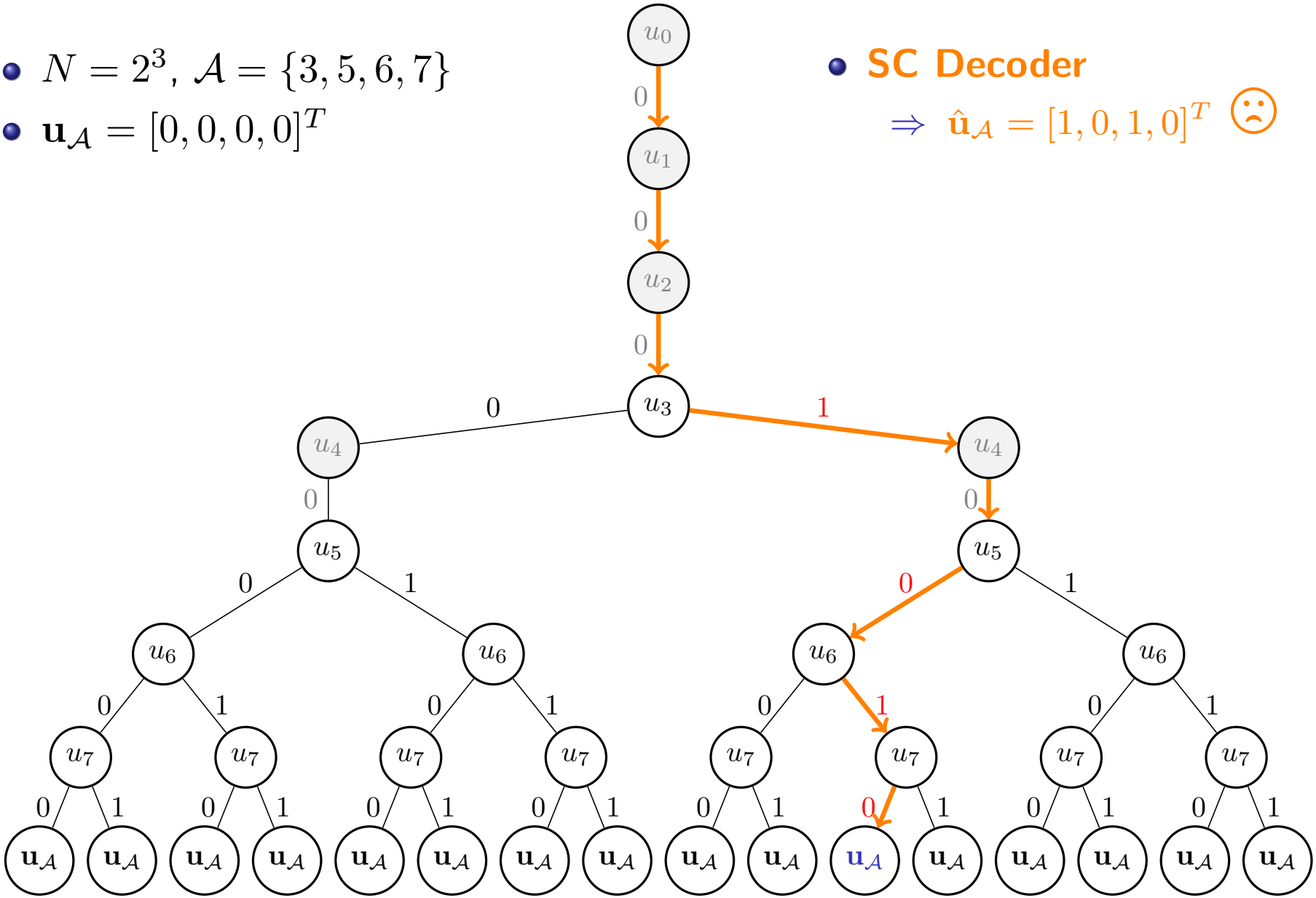


Example of SC versus SCL Decoding

- $N = 2^3$, $\mathcal{A} = \{3, 5, 6, 7\}$
- $\mathbf{u}_{\mathcal{A}} = [0, 0, 0, 0]^T$

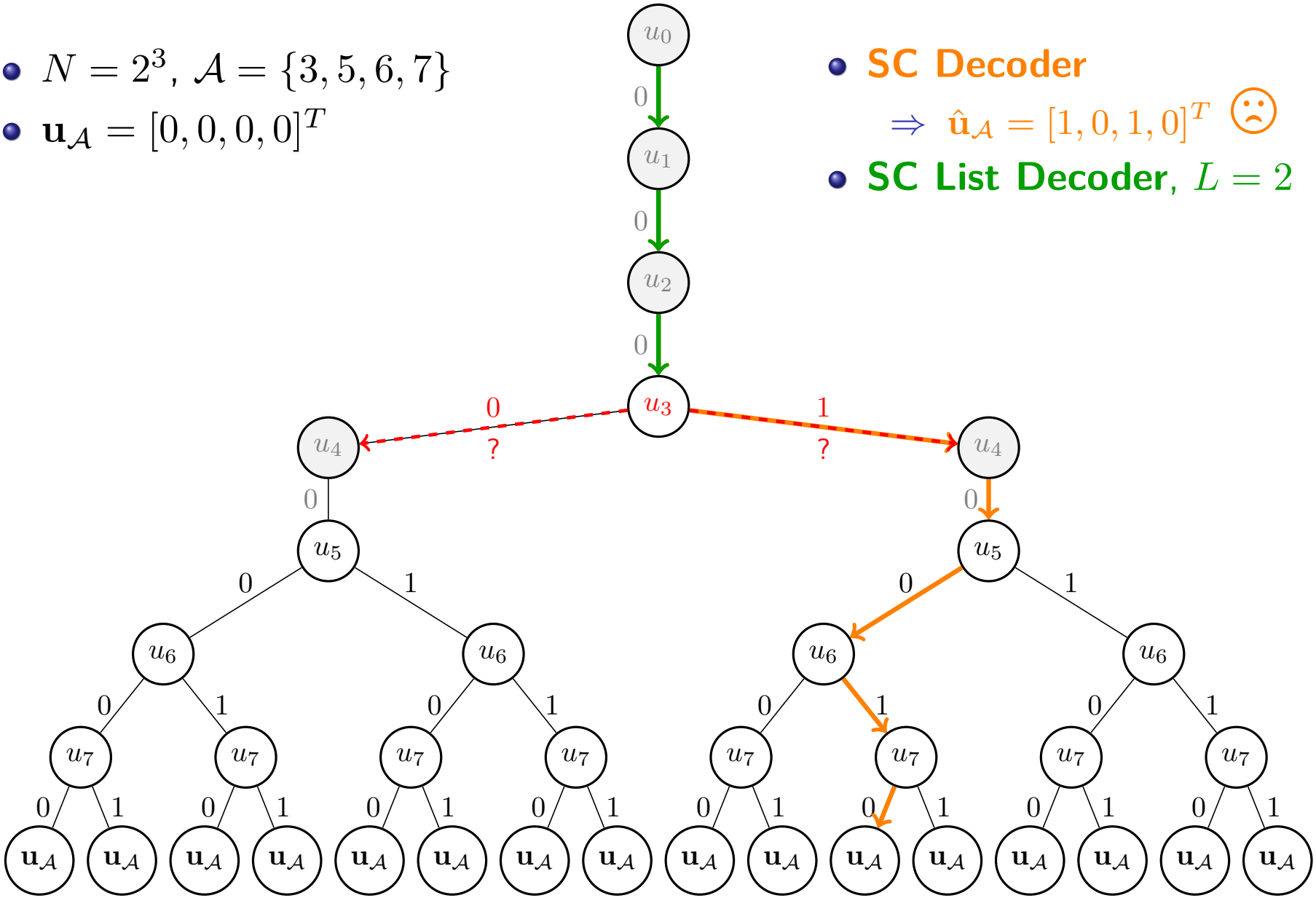
- **SC Decoder**

  $\Rightarrow \hat{\mathbf{u}}_{\mathcal{A}} = [1, 0, 1, 0]^T$ ☹



Example of SC versus SCL Decoding

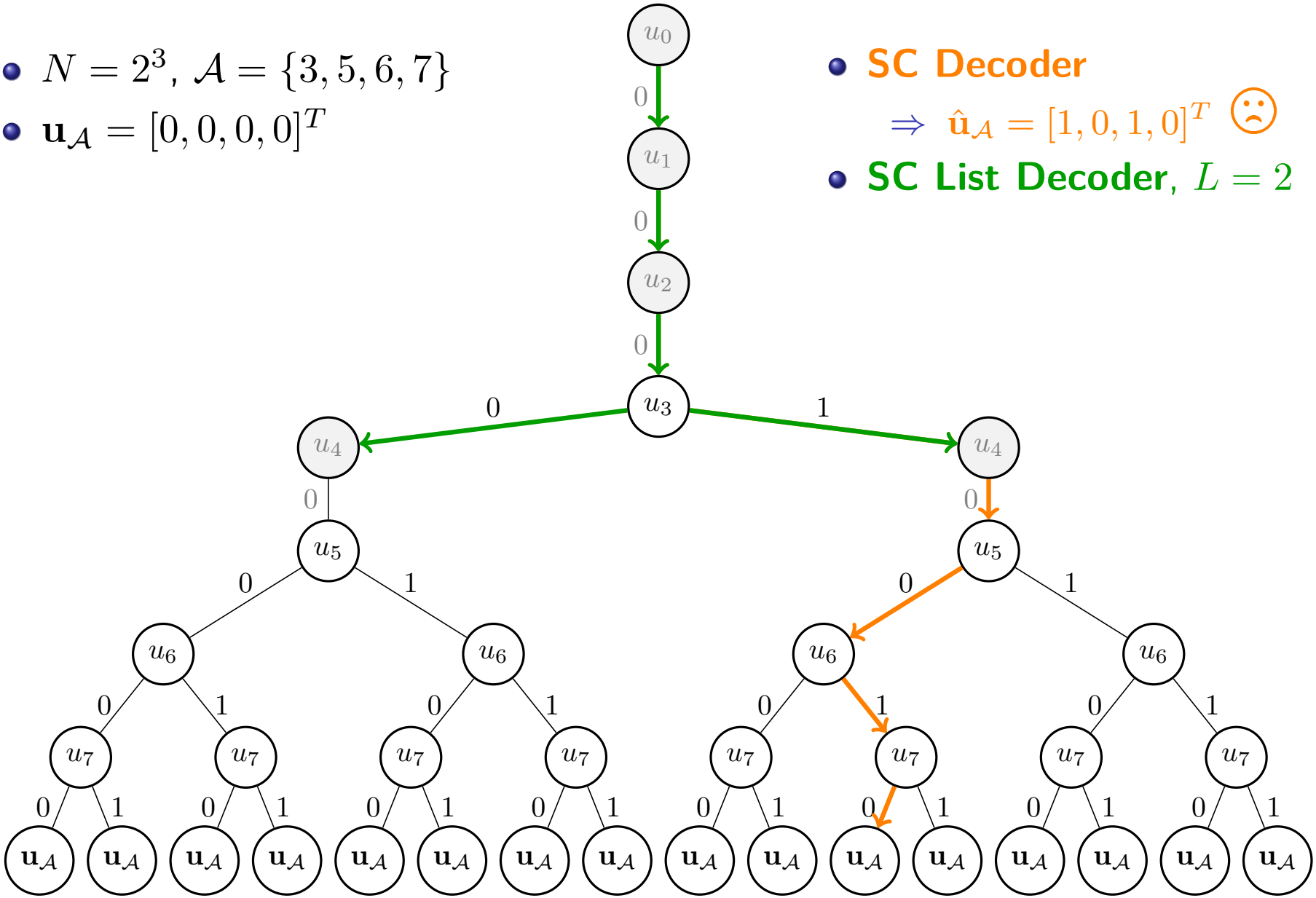
- $N = 2^3$, $\mathcal{A} = \{3, 5, 6, 7\}$
- $\mathbf{u}_{\mathcal{A}} = [0, 0, 0, 0]^T$



- **SC Decoder**
  $\Rightarrow \hat{\mathbf{u}}_{\mathcal{A}} = [1, 0, 1, 0]^T$ ☹
- **SC List Decoder**, $L = 2$

Example of SC versus SCL Decoding

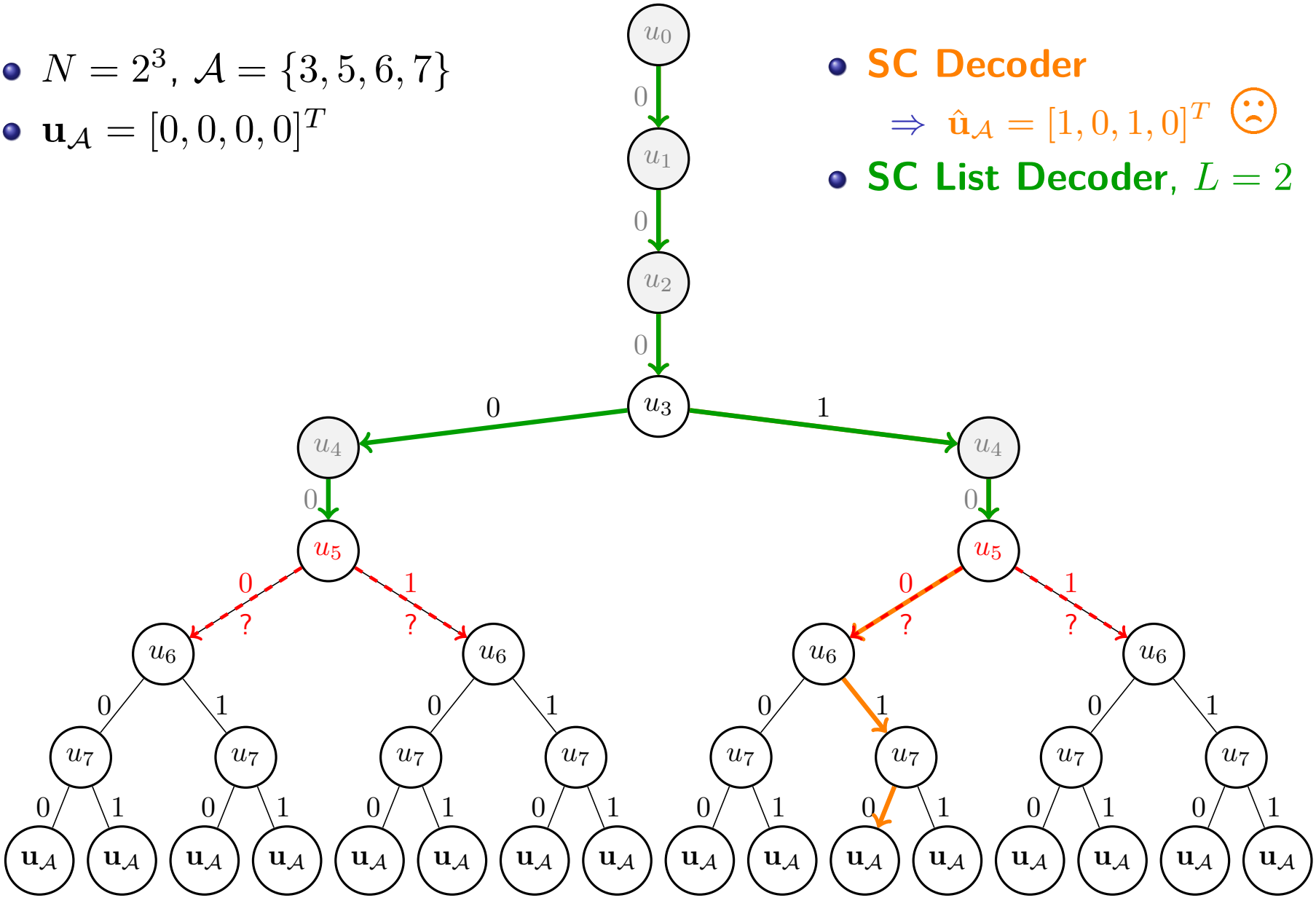
- $N = 2^3$, $\mathcal{A} = \{3, 5, 6, 7\}$
- $\mathbf{u}_{\mathcal{A}} = [0, 0, 0, 0]^T$



- **SC Decoder**
  $\Rightarrow \hat{\mathbf{u}}_{\mathcal{A}} = [1, 0, 1, 0]^T$ ☹
- **SC List Decoder**, $L = 2$

Example of SC versus SCL Decoding

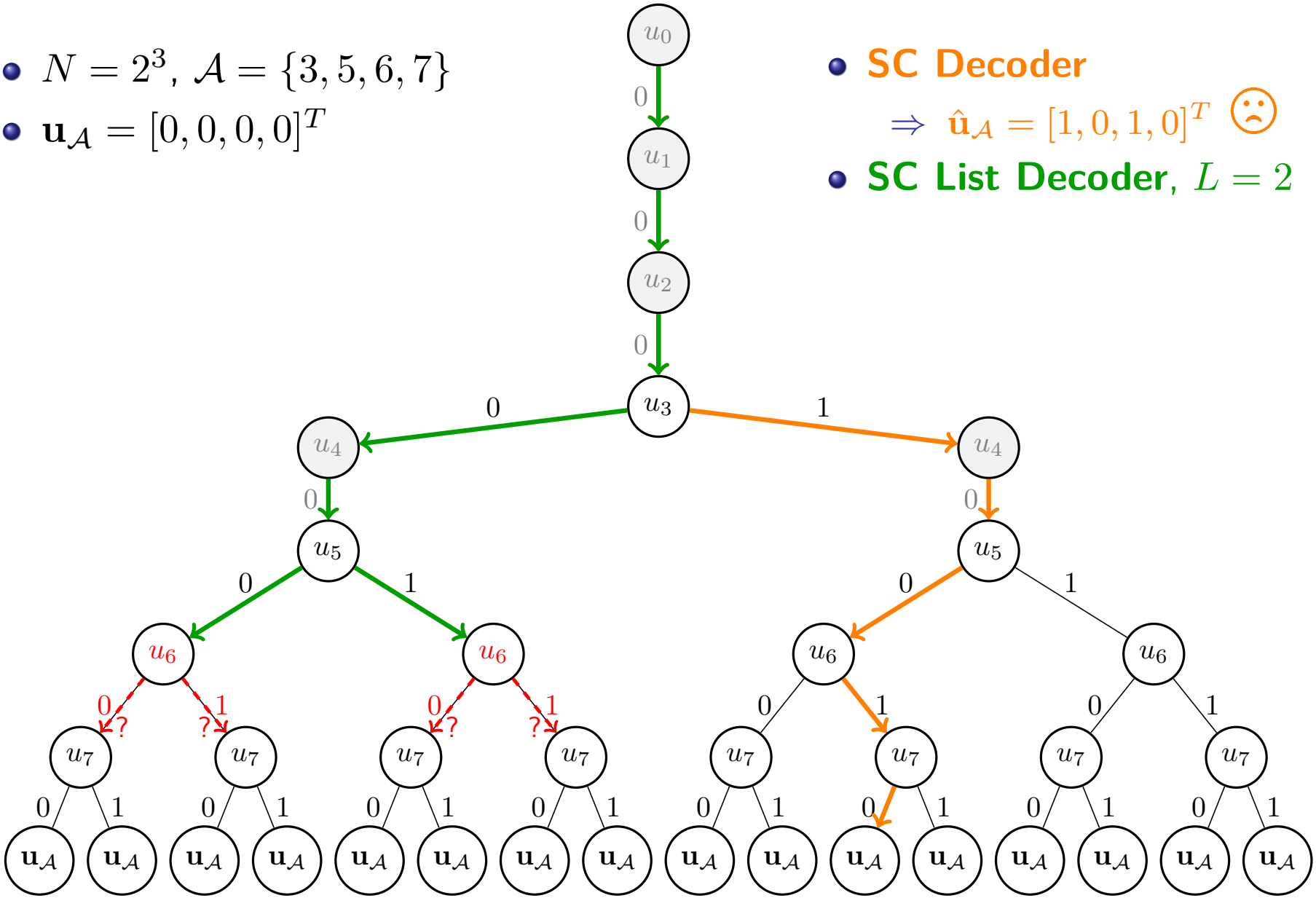
- $N = 2^3$, $\mathcal{A} = \{3, 5, 6, 7\}$
- $\mathbf{u}_{\mathcal{A}} = [0, 0, 0, 0]^T$



- **SC Decoder**
  $\Rightarrow \hat{\mathbf{u}}_{\mathcal{A}} = [1, 0, 1, 0]^T$ ☹
- **SC List Decoder**, $L = 2$

Example of SC versus SCL Decoding

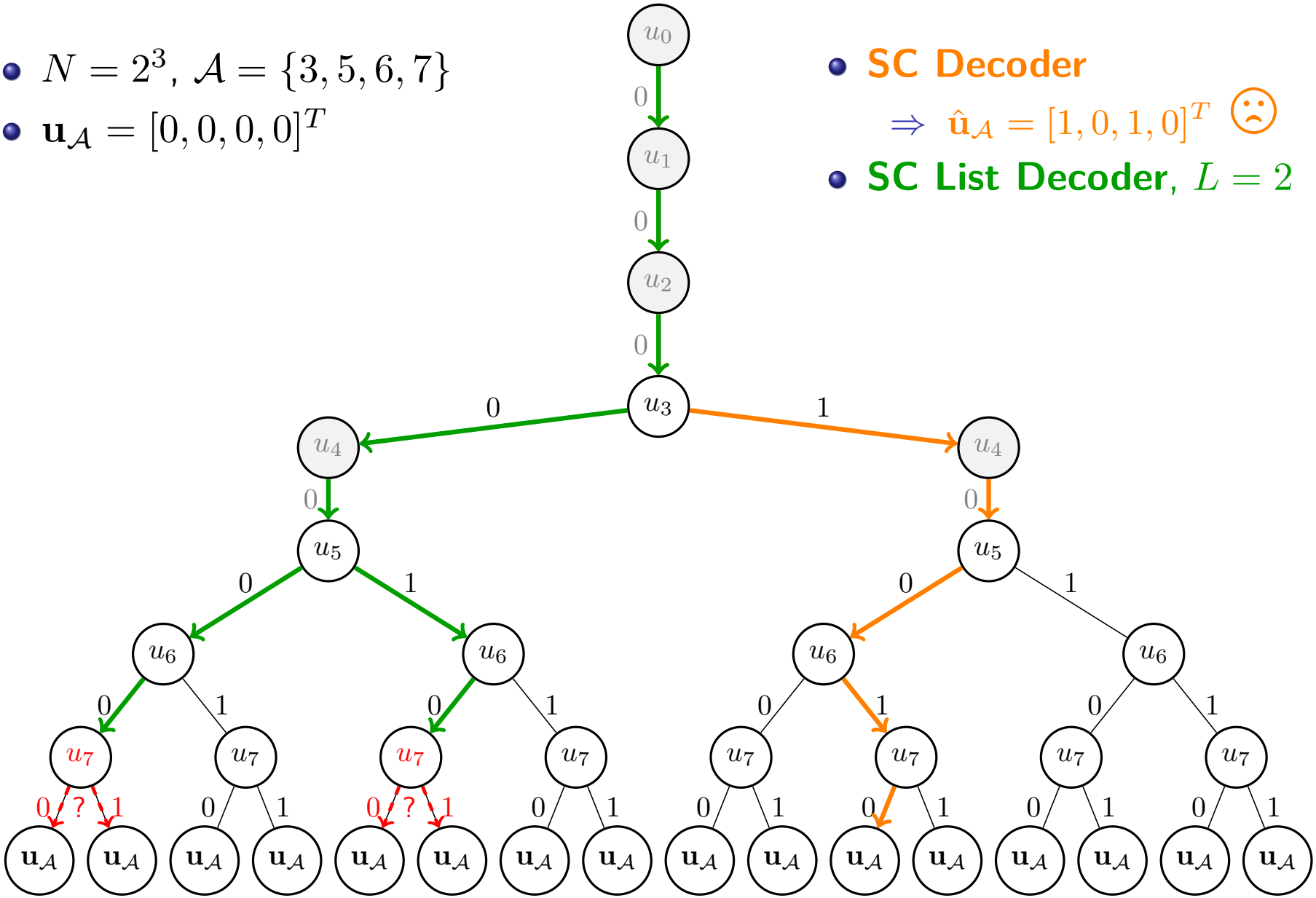
- $N = 2^3$, $\mathcal{A} = \{3, 5, 6, 7\}$
- $\mathbf{u}_{\mathcal{A}} = [0, 0, 0, 0]^T$



- **SC Decoder**
  $\Rightarrow \hat{\mathbf{u}}_{\mathcal{A}} = [1, 0, 1, 0]^T$ ☹
- **SC List Decoder**, $L = 2$

Example of SC versus SCL Decoding

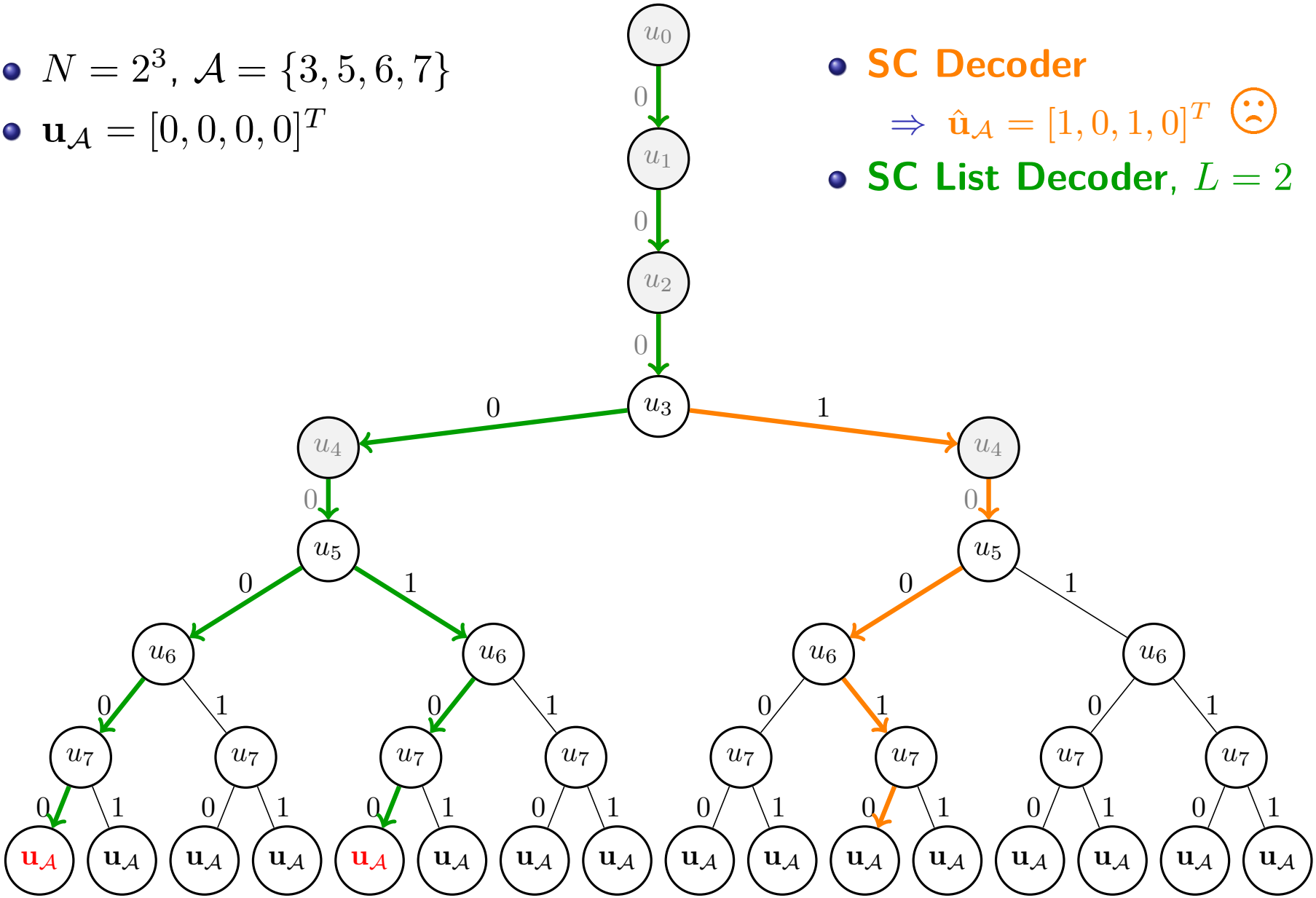
- $N = 2^3$, $\mathcal{A} = \{3, 5, 6, 7\}$
- $\mathbf{u}_{\mathcal{A}} = [0, 0, 0, 0]^T$



- **SC Decoder**
  $\Rightarrow \hat{\mathbf{u}}_{\mathcal{A}} = [1, 0, 1, 0]^T$ ☹
- **SC List Decoder**, $L = 2$

Example of SC versus SCL Decoding

- $N = 2^3$, $\mathcal{A} = \{3, 5, 6, 7\}$
- $\mathbf{u}_{\mathcal{A}} = [0, 0, 0, 0]^T$

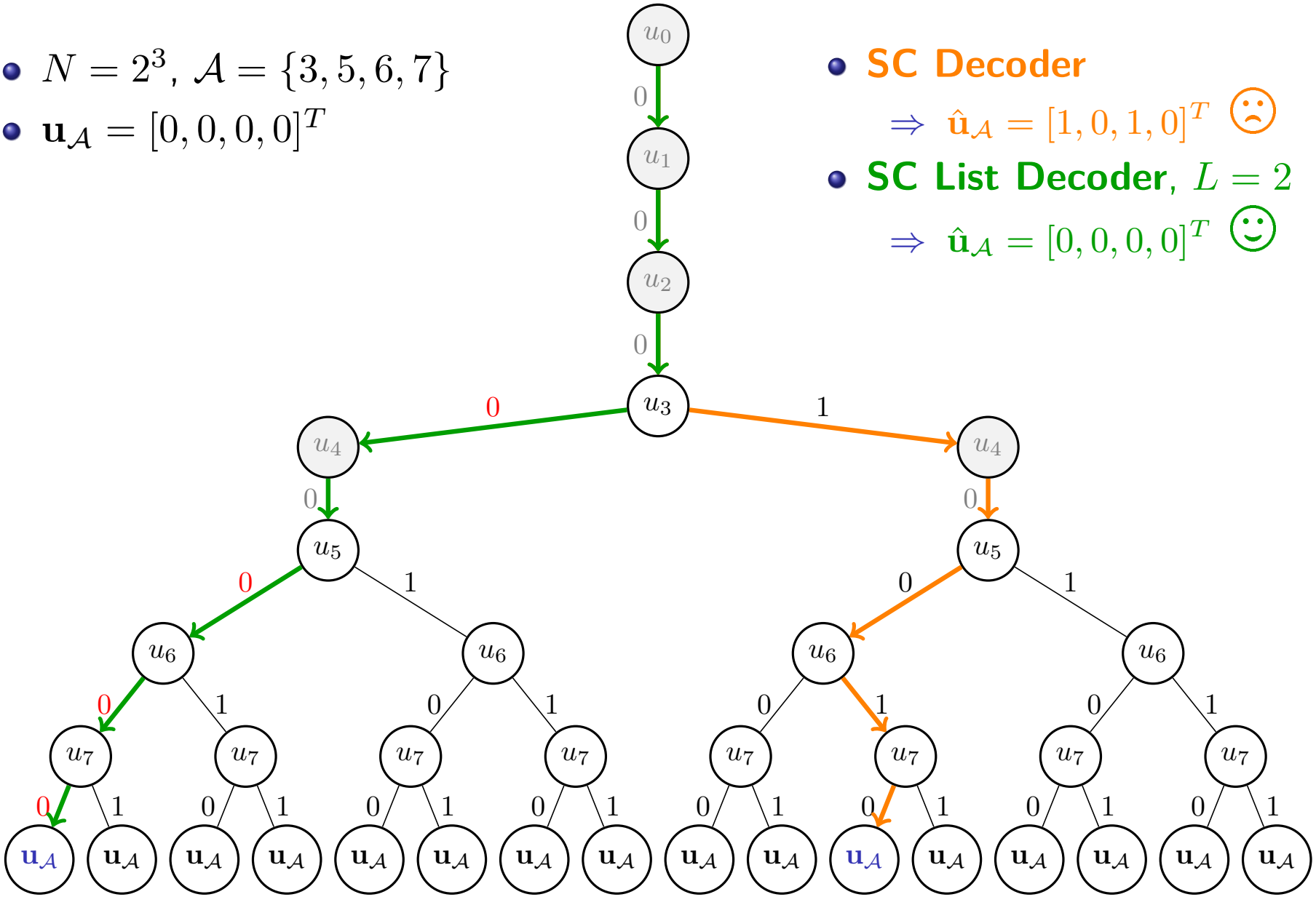


- **SC Decoder**
  $\Rightarrow \hat{\mathbf{u}}_{\mathcal{A}} = [1, 0, 1, 0]^T$ ☹
- **SC List Decoder**, $L = 2$

Example of SC versus SCL Decoding

- $N = 2^3$, $\mathcal{A} = \{3, 5, 6, 7\}$
- $\mathbf{u}_{\mathcal{A}} = [0, 0, 0, 0]^T$

- **SC Decoder**
  $\Rightarrow \hat{\mathbf{u}}_{\mathcal{A}} = [1, 0, 1, 0]^T$ ☹
- **SC List Decoder**, $L = 2$
  $\Rightarrow \hat{\mathbf{u}}_{\mathcal{A}} = [0, 0, 0, 0]^T$ ☺



Implementing the Decoders

- SC Decoder: binary decisions

Implementing the Decoders

- SC Decoder: binary decisions
  - Computing the **log-likelihood ratios** (LLRs),

$$\mathsf{L}_n^{(i)} \triangleq \ln \frac{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1} | u_i = 0)}{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1} | u_i = 1)}$$

    is sufficient for the decisions.

Implementing the Decoders

- SC Decoder: binary decisions
  - Computing the **log-likelihood ratios** (LLRs),

$$\mathsf{L}_n^{(i)} \triangleq \ln \frac{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1} | u_i = 0)}{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1} | u_i = 1)}$$

    is sufficient for the decisions.
  - **LLR-based** implementation:
    - Simple arithmetics (Min-Sum approximation),
    - Numerical stability.

Implementing the Decoders

- SC Decoder: binary decisions
  - Computing the **log-likelihood ratios** (LLRs),

$$\mathsf{L}_n^{(i)} \triangleq \ln \frac{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1} | u_i = 0)}{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1} | u_i = 1)}$$

    is sufficient for the decisions.
  - **LLR-based** implementation:
    - Simple arithmetics (Min-Sum approximation),
    - Numerical stability.
- SC List Decoding: Decisions of type **best L out of $2L$ possibilities**.

Implementing the Decoders

- SC Decoder: binary decisions
  - Computing the **log-likelihood ratios** (LLRs),

$$\mathsf{L}_n^{(i)} \triangleq \ln \frac{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1} | u_i = 0)}{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1} | u_i = 1)}$$

    is sufficient for the decisions.
  - **LLR-based** implementation:
    - Simple arithmetics (Min-Sum approximation),
    - Numerical stability.
- SC List Decoding: Decisions of type **best L out of $2L$ possibilities**.
  - The **LLR**s

$$\mathsf{L}_n^{(i)}[\ell] \triangleq \ln \frac{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1}[\ell] | u_i = 0)}{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1}[\ell] | u_i = 1)}, \ \ell \in \{0, 1, \ldots, L-1\},$$

    seem not to be useful a priori.

Implementing the Decoders

- SC Decoder: binary decisions
  - Computing the **log-likelihood ratios** (LLRs),

$$\mathsf{L}_n^{(i)} \triangleq \ln \frac{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1} | u_i = 0)}{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1} | u_i = 1)}$$

    is sufficient for the decisions.
  - **LLR-based** implementation:
    - Simple arithmetics (Min-Sum approximation),
    - Numerical stability.
- SC List Decoding: Decisions of type **best L out of $2L$ possibilities**.
  - The **LLR**s

$$\mathsf{L}_n^{(i)}[\ell] \triangleq \ln \frac{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1}[\ell] | u_i = 0)}{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1}[\ell] | u_i = 1)}, \ \ell \in \{0, 1, \ldots, L-1\},$$

    seem not to be useful a priori.
  - Computing the likelihoods directly is practically impossible because of underflows.

How to Implement a SCL Decoder?

- [Tal and Vardy'11] Re-normalize the likelihoods at intermediate steps.
  - Suitable for software implementation but still requires floating point numbers and complicated arithmetic operations (multiplications).

How to Implement a SCL Decoder?

- [Tal and Vardy'11] Re-normalize the likelihoods at intermediate steps.
  - Suitable for software implementation but still requires floating point numbers and complicated arithmetic operations (multiplications).
- [Balatsoukas-Stimming *et. al*'13] Compute the **log-likelihoods (LLs)**

$$-\ln W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1}[\ell]|u_i), \qquad \forall \ell = 0, 1, .., L-1, u_i \in \{0, 1\}$$

How to Implement a SCL Decoder?

- [Tal and Vardy'11] Re-normalize the likelihoods at intermediate steps.
  - Suitable for software implementation but still requires floating point numbers and complicated arithmetic operations (multiplications).
- [Balatsoukas-Stimming *et. al*'13] Compute the **log-likelihoods (LLs)**

$$-\ln W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1}[\ell]|u_i), \qquad \forall \ell = 0, 1, .., L-1, u_i \in \{0, 1\}$$

  - Some numerical stability,
  - Relatively simple arithmetics (Min-Sum update rules for **LLs**),

How to Implement a SCL Decoder?

- [Tal and Vardy'11] Re-normalize the likelihoods at intermediate steps.
  - Suitable for software implementation but still requires floating point numbers and complicated arithmetic operations (multiplications).
- [Balatsoukas-Stimming *et. al*'13] Compute the **log-likelihoods (LLs)**

$$-\ln W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1}[\ell]|u_i), \qquad \forall \ell = 0, 1, .., L-1, u_i \in \{0, 1\}$$

  - Some numerical stability,
  - Relatively simple arithmetics (Min-Sum update rules for **LLs**),
  - Large and irregular memory elements

How to Implement a SCL Decoder?

- [Tal and Vardy'11] Re-normalize the likelihoods at intermediate steps.
  - Suitable for software implementation but still requires floating point numbers and complicated arithmetic operations (multiplications).
- [Balatsoukas-Stimming *et. al*'13] Compute the **log-likelihoods (LLs)**

$$-\ln W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1}[\ell]|u_i), \qquad \forall \ell = 0, 1, .., L-1, u_i \in \{0, 1\}$$

  - Some numerical stability,
  - Relatively simple arithmetics (Min-Sum update rules for **LLs**),
  - Large and irregular memory elements

Our Contribution

- The **SCL Decoder** can still be implemented using only the **LLR**s (at each decoding path) and an additional path-metric (used for sorting the paths).

How to Implement a SCL Decoder?

- [Tal and Vardy'11] Re-normalize the likelihoods at intermediate steps.
  - Suitable for software implementation but still requires floating point numbers and complicated arithmetic operations (multiplications).
- [Balatsoukas-Stimming *et. al*'13] Compute the **log-likelihoods (LLs)**

$$-\ln W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1}[\ell]|u_i), \qquad \forall \ell = 0, 1, .., L-1, u_i \in \{0, 1\}$$

  - Some numerical stability,
  - Relatively simple arithmetics (Min-Sum update rules for **LLs**),
  - Large and irregular memory elements

Our Contribution

- The **SCL Decoder** can still be implemented using only the **LLR**s (at each decoding path) and an additional path-metric (used for sorting the paths).

$\Rightarrow$ Combining the improved performance of **SCL Decoder** and the ease of implementation of **SC Decoder**.

LLR-based Path Metric Computation

Theorem

- *For each path* $\ell \in \{0, 1, \ldots, L-1\}$ *and each index* $i \in \{0, 1, \ldots, N-1\}$ *let the path-metric be defined as:*

$$\mathsf{PM}_{\ell}^{(i)} \triangleq \sum_{j=0}^{i} \ln\left(1 + e^{-(1-2\hat{u}_j[\ell])\cdot \mathsf{L}_n^{(j)}[\ell]}\right),$$

where $\mathsf{L}_n^{(i)}[\ell] = \ln \dfrac{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1}[\ell]|0)}{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1}[\ell]|1)}$.

LLR-based Path Metric Computation

Theorem

- *For each path* $\ell \in \{0, 1, \ldots, L-1\}$ *and each index* $i \in \{0, 1, \ldots, N-1\}$ *let the path-metric be defined as:*

$$\mathsf{PM}_{\ell}^{(i)} \triangleq \sum_{j=0}^{i} \ln\left(1 + e^{-(1-2\hat{u}_j[\ell]) \cdot \mathsf{L}_n^{(j)}[\ell]}\right),$$

where $\mathsf{L}_n^{(i)}[\ell] = \ln \frac{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1}[\ell]|0)}{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1}[\ell]|1)}$.

- *If all the information bits are uniformly distributed in* $\{0, 1\}$*, for any pair of paths* $\ell, \ell' \in \{0, 1, \ldots, L-1\}$*,*

$$W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1}[\ell]|\hat{u}_i[\ell]) < W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1}[\ell']|\hat{u}_i[\ell']) \iff \mathsf{PM}_{\ell}^{(i)} > \mathsf{PM}_{\ell'}^{(i)}.$$

LLR-based Path Metric Computation

Theorem

- *For each path* $\ell \in \{0, 1, \ldots, L-1\}$ *and each index* $i \in \{0, 1, \ldots, N-1\}$ *let the path-metric be defined as:*

$$\mathsf{PM}_{\ell}^{(i)} \triangleq \sum_{j=0}^{i} \ln\left(1 + e^{-(1-2\hat{u}_j[\ell]) \cdot \mathsf{L}_n^{(j)}[\ell]}\right),$$

where $\mathsf{L}_n^{(i)}[\ell] = \ln \frac{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1}[\ell]|0)}{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1}[\ell]|1)}$.

- *If all the information bits are uniformly distributed in* $\{0, 1\}$*, for any pair of paths* $\ell, \ell' \in \{0, 1, \ldots, L-1\}$*,*

$$W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1}[\ell]|\hat{u}_i[\ell]) < W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1}[\ell']|\hat{u}_i[\ell']) \iff \mathsf{PM}_{\ell}^{(i)} > \mathsf{PM}_{\ell'}^{(i)}.$$

$\Rightarrow$ Implement a **SCL Decoder** using L, **LLR-based** SC decoders + L memories for the path-metrics.

Properties of the LLR-Based Path Metric

$$\mathsf{PM}_\ell^{(i)} \triangleq \sum_{j=0}^{i} \ln\left(1 + e^{-(1-2\hat{u}_j[\ell])\cdot \mathsf{L}_n^{(j)}[\ell]}\right)$$

$$\mathsf{L}_n^{(i)}[\ell] = \ln \frac{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1}[\ell]|0)}{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1}[\ell]|1)}.$$

- Iterative update:

$$\mathsf{PM}_\ell^{(i)} = \mathsf{PM}_\ell^{(i-1)} + \ln\left(1 + e^{-(1-2\hat{u}_i[\ell])\mathsf{L}_n^{(i)}[\ell]}\right).$$

Properties of the LLR-Based Path Metric

$$\mathsf{PM}_\ell^{(i)} \triangleq \sum_{j=0}^{i} \ln\left(1 + e^{-(1-2\hat{u}_j[\ell])\cdot \mathsf{L}_n^{(j)}[\ell]}\right)$$

$$\mathsf{L}_n^{(i)}[\ell] = \ln \frac{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1}[\ell]|0)}{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1}[\ell]|1)}.$$

- Iterative update:

$$\mathsf{PM}_\ell^{(i)} = \mathsf{PM}_\ell^{(i-1)} + \ln\left(1 + e^{-(1-2\hat{u}_i[\ell])\mathsf{L}_n^{(i)}[\ell]}\right).$$

- Approximation $\ln(1+e^x) \approx \begin{cases} x, & \text{if } x \geq 0, \\ 0, & \text{if } x < 0. \end{cases}$

$$\mathsf{PM}_\ell^{(i)} \approx \begin{cases} \mathsf{PM}_\ell^{(i-1)} & \text{if } \hat{u}_i[\ell] = \frac{1}{2}[1 - \operatorname{sign}(\mathsf{L}_n^{(i)}[\ell])], \\ \mathsf{PM}_\ell^{(i-1)} + |\mathsf{L}_n^{(i)}[\ell]|, & \text{otherwise.} \end{cases}$$

Properties of the LLR-Based Path Metric

$$\mathsf{PM}_\ell^{(i)} \triangleq \sum_{j=0}^{i} \ln\left(1 + e^{-(1-2\hat{u}_j[\ell])\cdot \mathsf{L}_n^{(j)}[\ell]}\right)$$

$$\mathsf{L}_n^{(i)}[\ell] = \ln \frac{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1}[\ell]|0)}{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1}[\ell]|1)}.$$

- Iterative update:

$$\mathsf{PM}_\ell^{(i)} = \mathsf{PM}_\ell^{(i-1)} + \ln\left(1 + e^{-(1-2\hat{u}_i[\ell])\mathsf{L}_n^{(i)}[\ell]}\right).$$

- Approximation $\ln(1+e^x) \approx \begin{cases} x, & \text{if } x \geq 0, \\ 0, & \text{if } x < 0. \end{cases}$

$$\mathsf{PM}_\ell^{(i)} \approx \begin{cases} \mathsf{PM}_\ell^{(i-1)} & \text{if } \hat{u}_i[\ell] = \frac{1}{2}[1 - \operatorname{sign}(\mathsf{L}_n^{(i)}[\ell])], \\ \mathsf{PM}_\ell^{(i-1)} + |\mathsf{L}_n^{(i)}[\ell]|, & \text{otherwise.} \end{cases}$$

  - **Hardware-friendly:** only addition and comparison required

Properties of the LLR-Based Path Metric

$$\mathrm{PM}_\ell^{(i)} \triangleq \sum_{j=0}^{i} \ln\left(1 + e^{-(1-2\hat{u}_j[\ell])\cdot \mathsf{L}_n^{(j)}[\ell]}\right)$$

$$\mathsf{L}_n^{(i)}[\ell] = \ln \frac{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1}[\ell]|0)}{W_n^{(i)}(\mathbf{y}, \hat{\mathbf{u}}^{i-1}[\ell]|1)}.$$

- Iterative update:

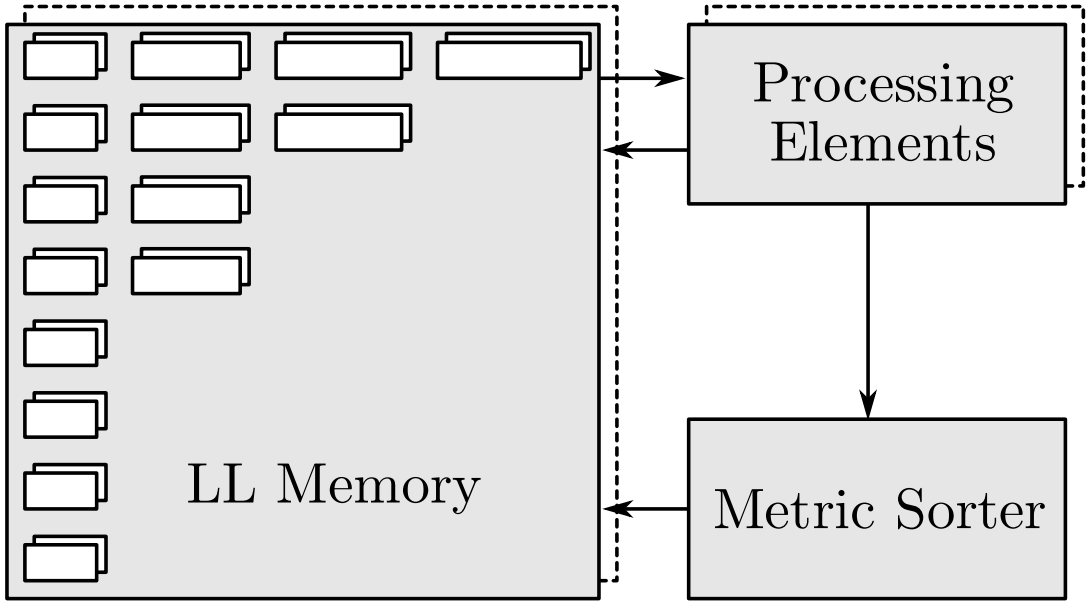
$$\mathrm{PM}_\ell^{(i)} = \mathrm{PM}_\ell^{(i-1)} + \ln\left(1 + e^{-(1-2\hat{u}_i[\ell])\mathsf{L}_n^{(i)}[\ell]}\right).$$

- Approximation $\ln(1+e^x) \approx \begin{cases} x, & \text{if } x \geq 0, \\ 0, & \text{if } x < 0. \end{cases}$

$$\mathrm{PM}_\ell^{(i)} \approx \begin{cases} \mathrm{PM}_\ell^{(i-1)} & \text{if } \hat{u}_i[\ell] = \frac{1}{2}[1 - \mathrm{sign}(\mathsf{L}_n^{(i)}[\ell])], \\ \mathrm{PM}_\ell^{(i-1)} + |\mathsf{L}_n^{(i)}[\ell]|, & \text{otherwise.} \end{cases}$$

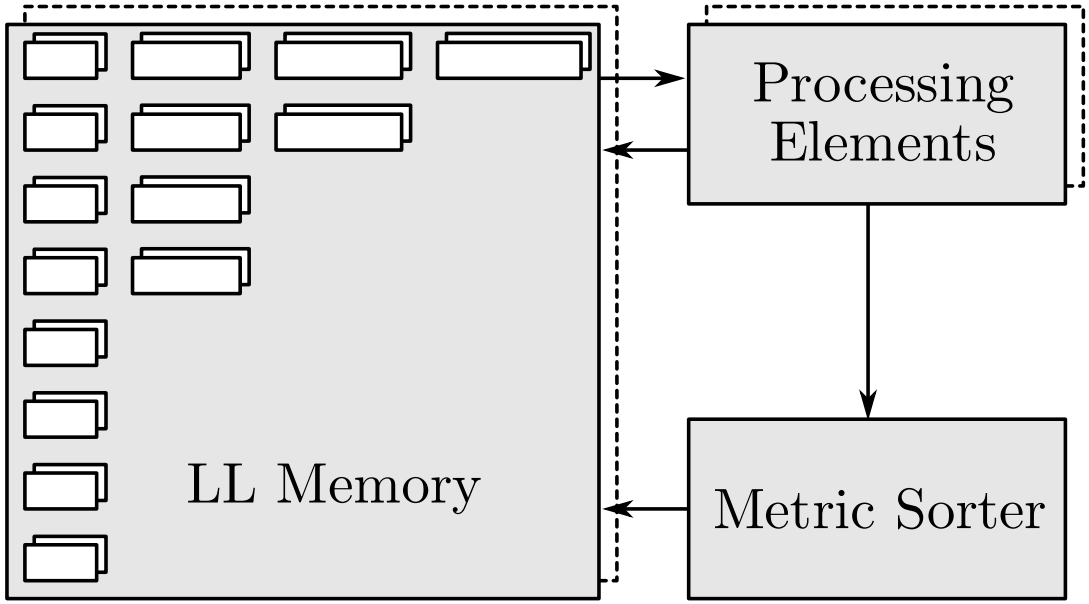
  - **Hardware-friendly:** only addition and comparison required
  - **Natural interpretation:** paths "disobeying" $\mathsf{L}_n^{(i)}[\ell]$ are penalized by an amount $\approx |\mathsf{L}_n^{(i)}[\ell]|$.

Hardware Architecture - LL based



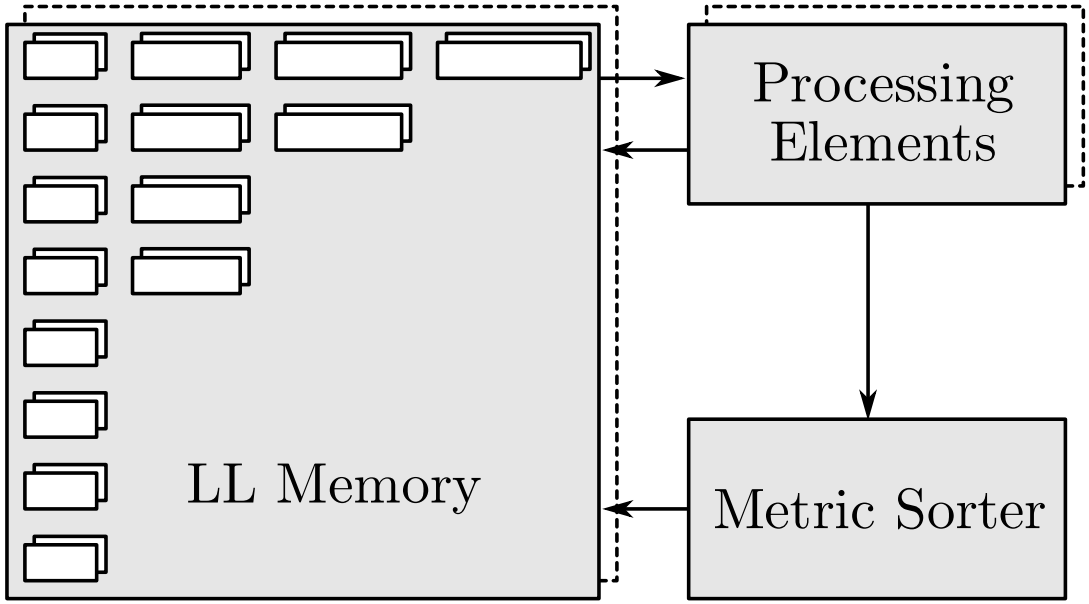
- **LL Quantization:**
  - **Channel**: Q^{ch}
  - **Internal**: $Q^{\text{ch}} + j$ bits for stage j $\Rightarrow$ irregular memory

Hardware Architecture - LL based



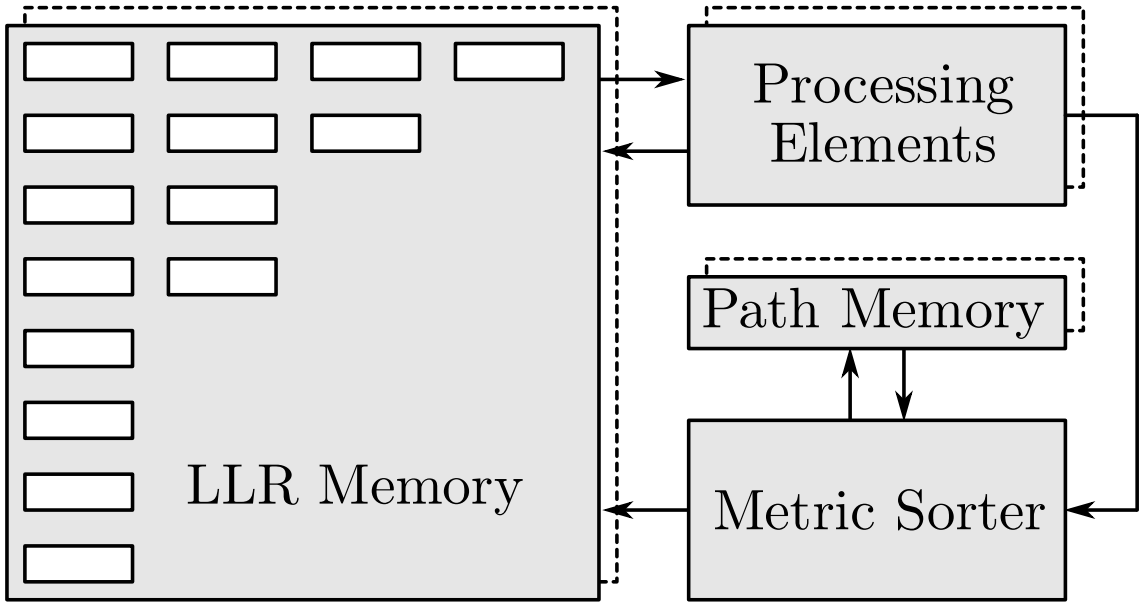
- **LL Quantization:**
  - **Channel**: Q^{ch}
  - **Internal**: $Q^{\text{ch}} + j$ bits for stage j $\Rightarrow$ irregular memory
- **Processing Elements**: Maximum LL bit-width: $Q^{\text{LL}} = Q^{\text{ch}} + n$ bits

Hardware Architecture - LL based



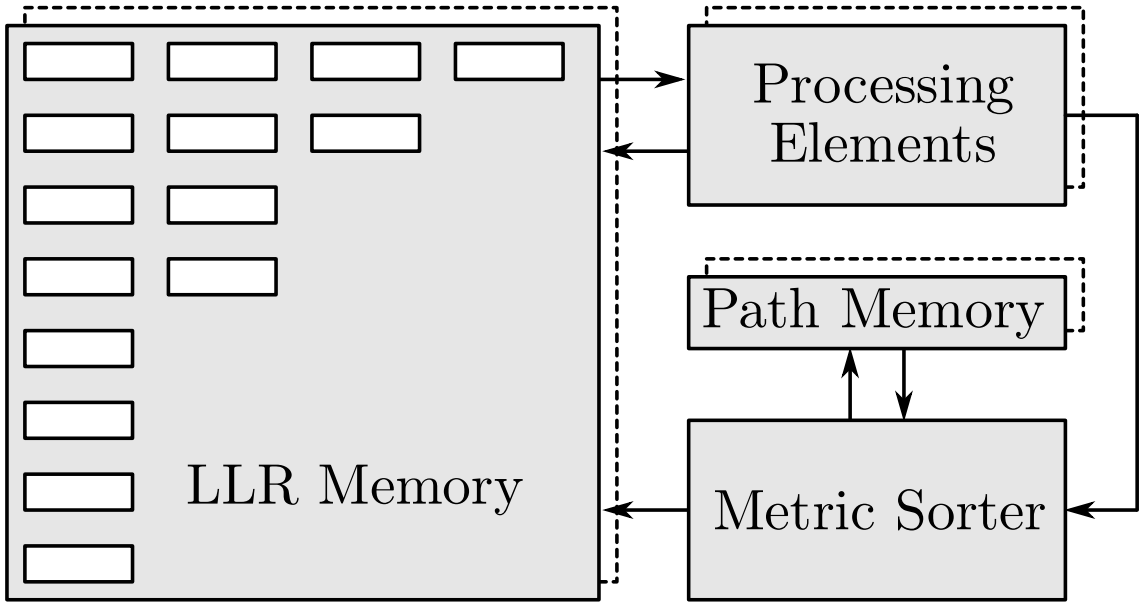
- **LL Quantization:**
  - **Channel**: Q^{ch}
  - **Internal**: $Q^{\text{ch}} + j$ bits for stage j $\Rightarrow$ irregular memory
- **Processing Elements**: Maximum LL bit-width: $Q^{\text{LL}} = Q^{\text{ch}} + n$ bits
- **Metric Sorter**: Sorts $2L$ path metrics of bit-width Q^{LL}

Hardware Architecture - LLR based



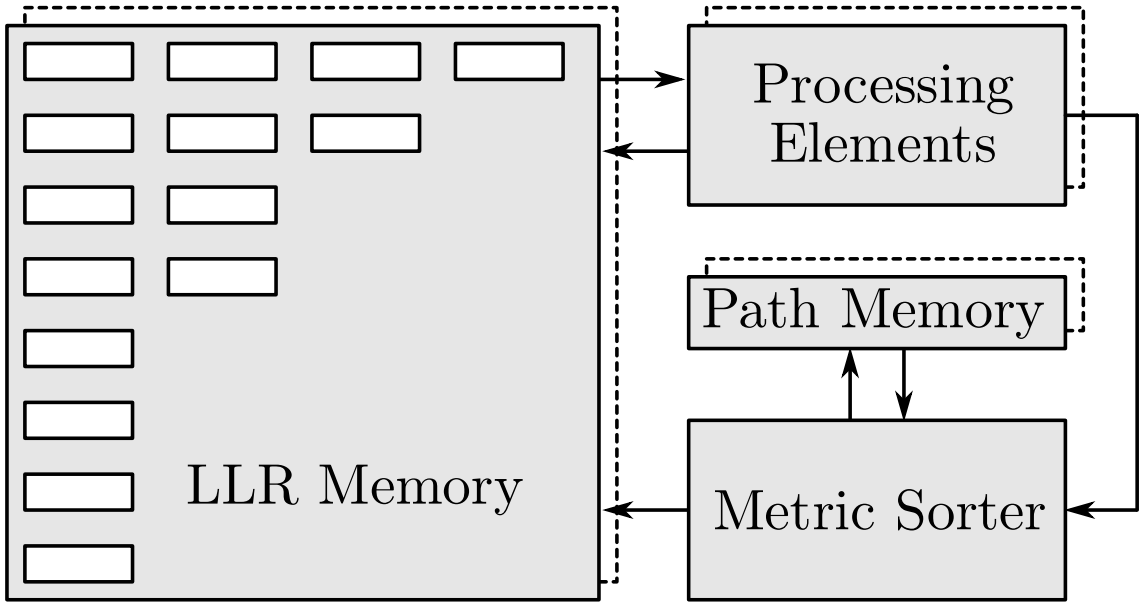
- **LLR Quantization:**
  - Q^{LLR} bits for all stages $\Rightarrow$ regular memory

Hardware Architecture - LLR based



- **LLR Quantization:**
  - Q^{LLR} bits for all stages $\Rightarrow$ regular memory
- **Processing Elements**: LLR bit-width: Q^{LLR} bits

Hardware Architecture - LLR based



- **LLR Quantization:**
  - Q^{LLR} bits for all stages $\Rightarrow$ regular memory
- **Processing Elements**: LLR bit-width: Q^{LLR} bits
- **Path Metrics**: Worst-case $Q^{\text{M}} = Q^{\text{LLR}} + n$
- **Metric Sorter**: Sorts $2L$ metrics of bit-width Q^{M}

Performance of Quantized SC List Decoding

- Polar code with $N = 1024$ and $R = 0.5$, designed for $E_b/N_0 = 2$dB.

Performance of Quantized SC List Decoding

- Polar code with $N = 1024$ and $R = 0.5$, designed for $E_b/N_0 = 2$dB.

- **LL-based**:
  - $Q^{\text{ch}} = 4$ bits
  - $Q^{\text{LL}} = 14$ bits

Performance of Quantized SC List Decoding

- Polar code with $N = 1024$ and $R = 0.5$, designed for $E_b/N_0 = 2$dB.

- **LL-based**:
  - $Q^{\text{ch}} = 4$ bits
  - $Q^{\text{LL}} = 14$ bits

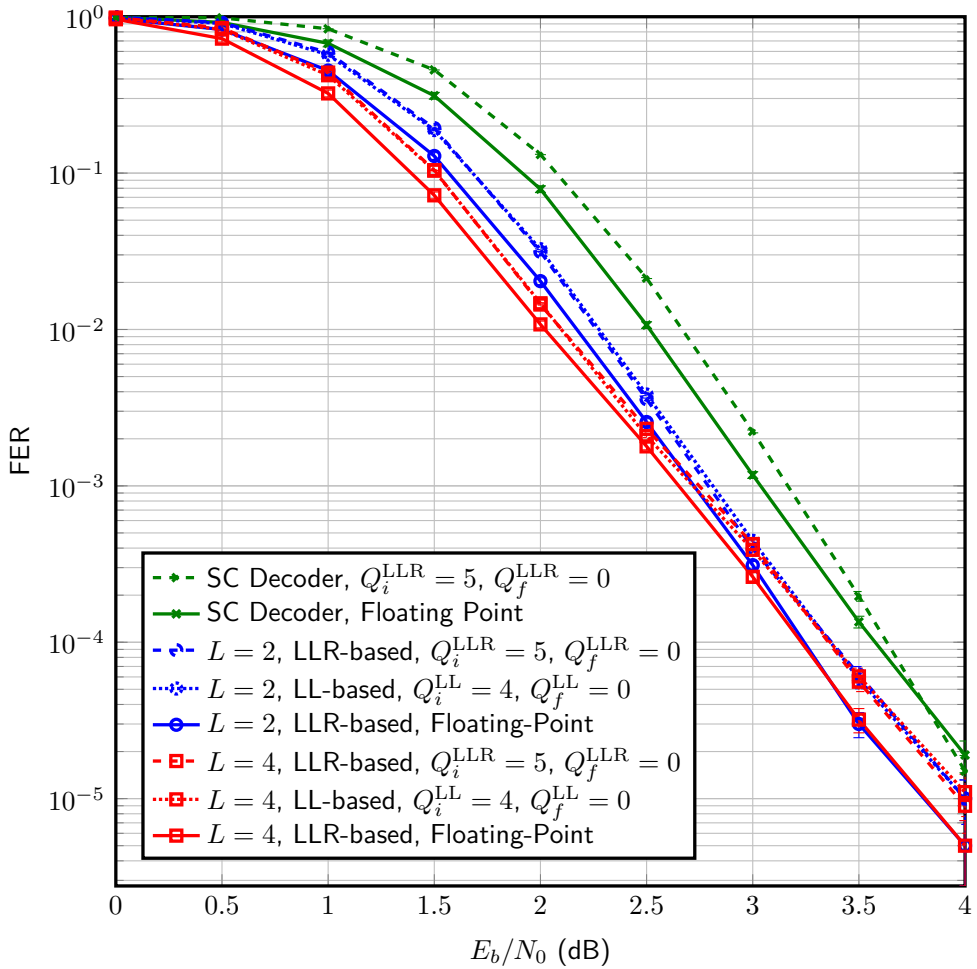
- **LLR-based**:
  - $Q^{\text{LLR}} = 6$ bits
  - $Q^{\text{M}} = 8$ bits

Performance of Quantized SC List Decoding

- Polar code with $N = 1024$ and $R = 0.5$, designed for $E_b/N_0 = 2$dB.

- **LL-based**:
  - $Q^{\text{ch}} = 4$ bits
  - $Q^{\text{LL}} = 14$ bits

- **LLR-based**:
  - $Q^{\text{LLR}} = 6$ bits
  - $Q^{\text{M}} = 8$ bits



Synthesis Area and Timing Results

Cell Area	LLR-based	LL-based	Reduction
List Size		$L = 2$	
Total	$\mathbf{0.977\ mm^2}$	$\mathbf{1.668\ mm^2}$	$\mathbf{41.4\,\%}$
List Size		$L = 4$	
Total	$\mathbf{1.743\ mm^2}$	$\mathbf{3.708\ mm^2}$	$\mathbf{53.0\,\%}$

Synthesis Area and Timing Results

Cell Area	LLR-based	LL-based	Reduction
List Size		$L = 2$	
Total	**0.977 mm²**	**1.668 mm²**	**41.4 %**
List Size		$L = 4$	
Total	**1.743 mm²**	**3.708 mm²**	**53.0 %**

	LLR-based	LL-based	Speedup
List Size		$L = 2$	
Clock Frequency	**558 MHz**	**427 MHz**	**30.7 %**
List Size		$L = 4$	
Clock Frequency	**412 MHz**	**386 MHz**	**6.7 %**

Summary and Conclusion

- Proved an **LLR-based path metric** for SC list decoding

Summary and Conclusion

- Proved an **LLR-based path metric** for SC list decoding
- Provided **hardware-friendly** approximation to metric update rule

Summary and Conclusion

- Proved an **LLR-based path metric** for SC list decoding
- Provided **hardware-friendly** approximation to metric update rule
- Hardware implementation results show:
  1. Up to **50%** area reduction
  2. Up to **30%** operating frequency increase