Towards Coding for Max Errors in Interactive Communication

Mark Braverman [Princeton University]
Anup Rao [University of Washington]

Classical Error Correction

$x \in \{0,1\}^n$

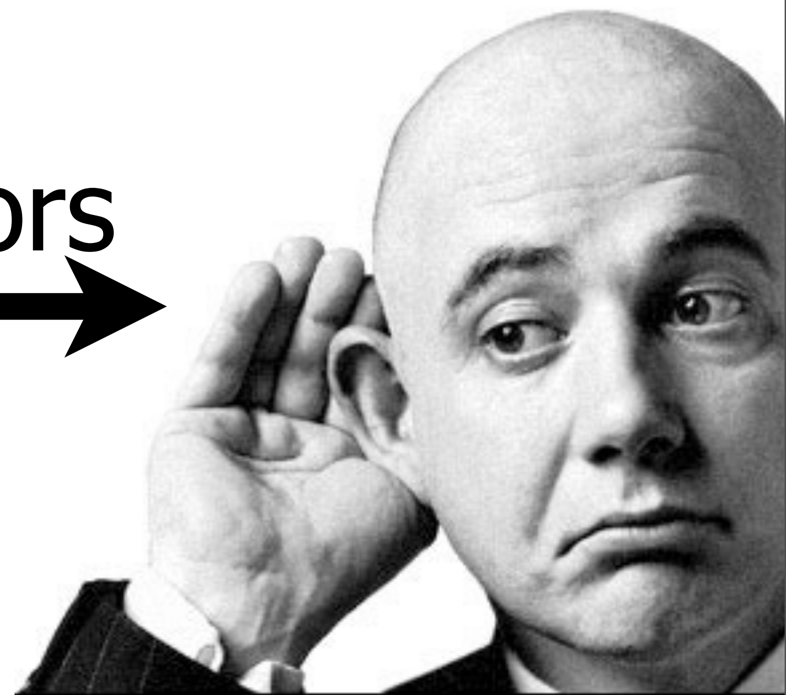
Classical Error Correction

$\xrightarrow{E(x)}$

$x \in \{0,1\}^n$

Classical Error Correction

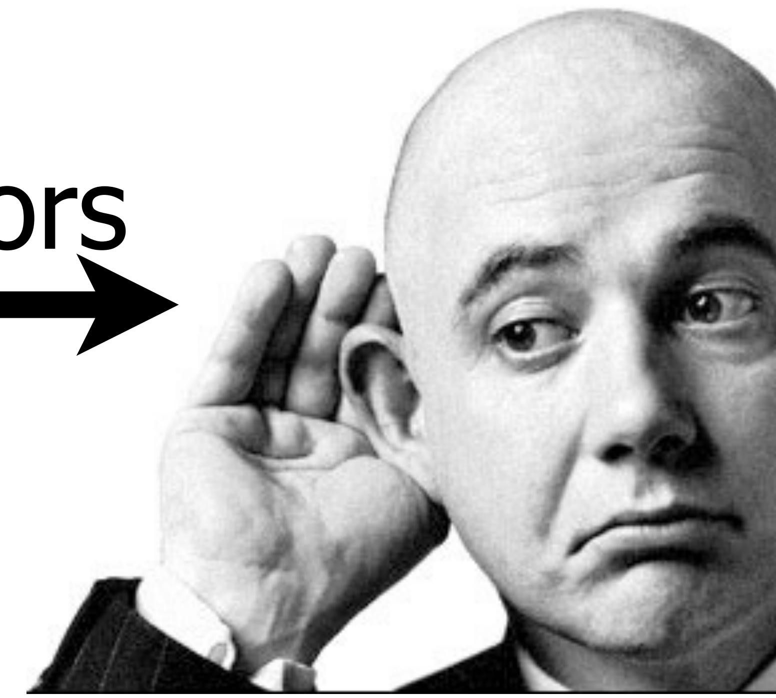
E(x) y=E(x) + errors



$x \in \{0,1\}^n$

Classical Error Correction

E(x) y=E(x) + errors



$x \in \{0,1\}^n$

$x = D(y)$

Classical Error Correction

$E(x)$ $\longrightarrow$ $y = E(x) + \text{errors}$

$x \in \{0,1\}^n$

$x = D(y)$

State of art: $E(x) = O_\varepsilon(n)$ bits

$D(y) = x$, if $(1/4-\varepsilon)$ errors

Classical Error Correction

E(x) $\longrightarrow$ y=E(x) + errors

NOTE: Throughout this talk: errors are 'adversarial'!

$x \in \{0,1\}^n$

$x = D(y)$

State of art: $E(x) = O_\varepsilon(n)$ bits

$D(y)=x$, if $(1/4-\varepsilon)$ errors

Classical Error Correction

$x \in \{0,1\}^n$

NOTE: Throughout this talk: errors are 'adversarial'!

Classical Error Correction

$x \in \{0,1\}^n$

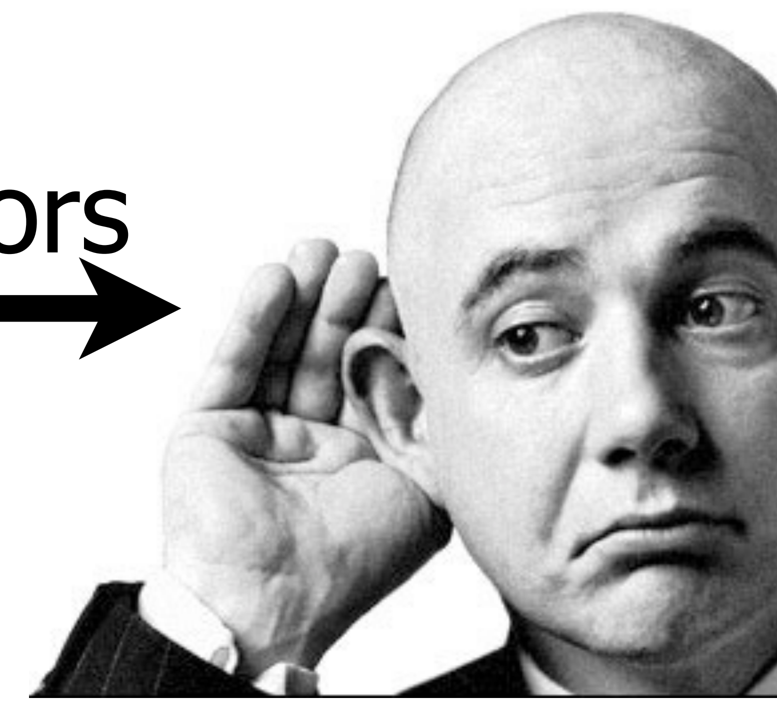
$E(x)$

NOTE: Throughout this talk: errors are 'adversarial'!

Classical Error Correction

E(x) $\longrightarrow$ y=E(x) + errors

NOTE: Throughout this talk: errors are 'adversarial'!

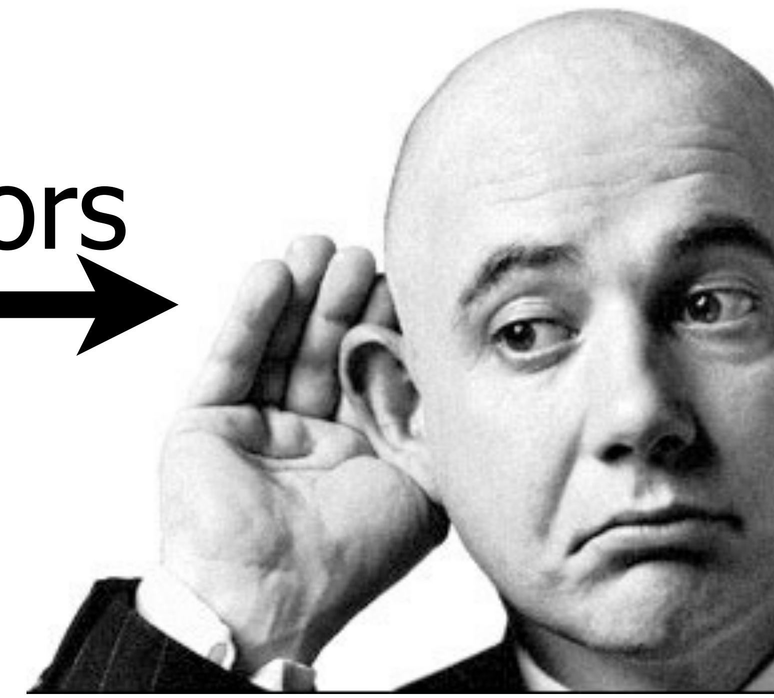


$x \in \{0,1\}^n$

Classical Error Correction

$E(x)$ $y=E(x) + \text{errors}$

NOTE: Throughout this talk: errors are 'adversarial'!

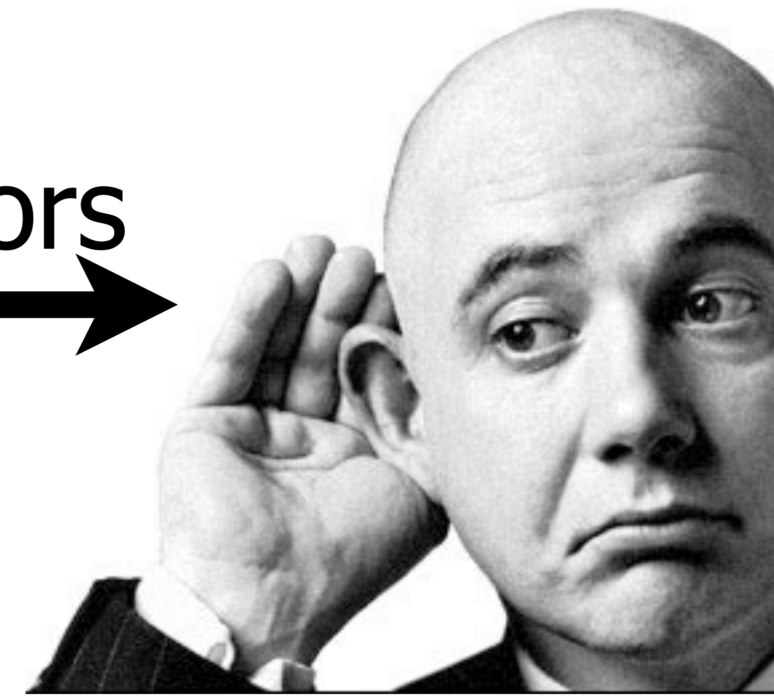


$x \in \{0,1\}^n$

$x = D(y)$

Classical Error Correction

$E(x)$ $y=E(x)$ + errors



NOTE: Throughout this talk: errors are 'adversarial'!

$x \in \{0,1\}^n$

$x = D(y)$

State of art: $|E(x)| = O_\varepsilon(n)$ $O_\varepsilon(1)$ size alphabet

$D(y)=x$, if $(1/2-\varepsilon)$ errors

Interaction given by: m1,m2,...

x  y

Interaction given by: m1,m2,...

x  y

$m_1(x)$ $\longrightarrow$

Interaction given by: m1,m2,...

x  y

$$\xrightarrow{m_1(x)}$$

$$\xleftarrow{m_2(y,m_1)}$$

Interaction given by: m1,m2,...

x  y

$m_1(x)$

$\longrightarrow$

$m_2(y,m_1)$

$\longleftarrow$

$\longrightarrow$

Interaction given by: m1,m2,...

x

y

$m_1(x)$ $\longrightarrow$

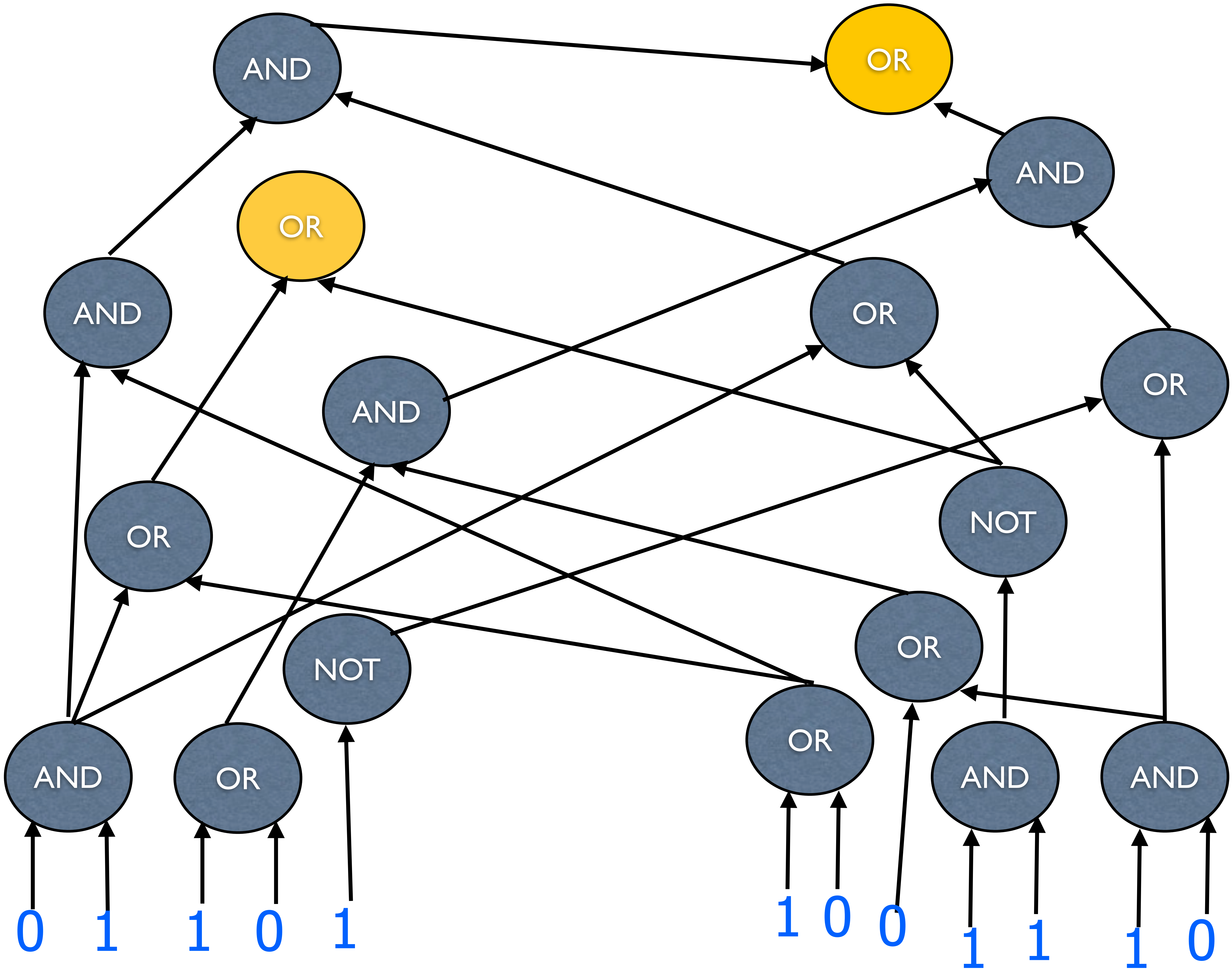
$m_2(y,m_1)$ $\longleftarrow$

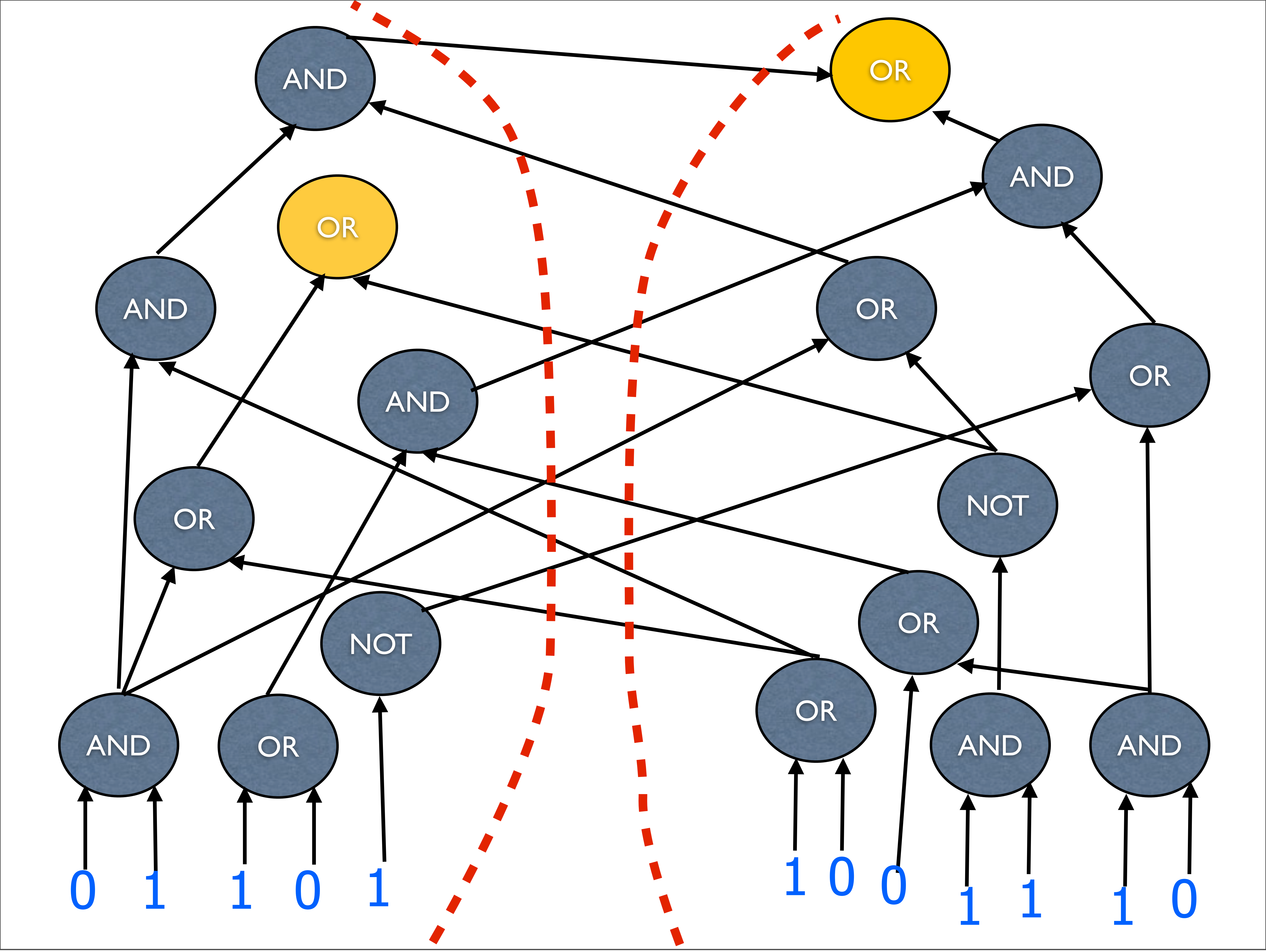
$\longrightarrow$

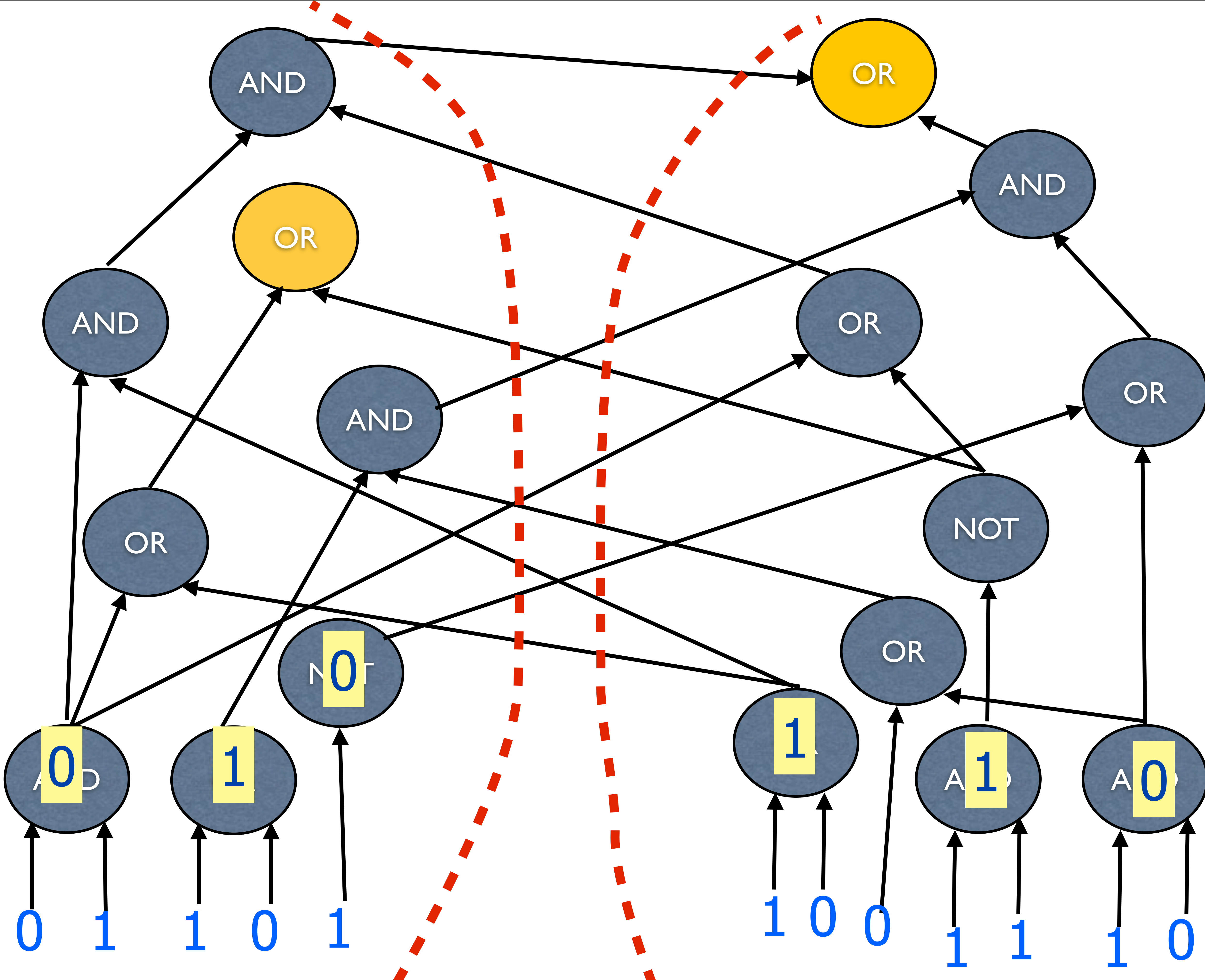
$\longleftarrow$

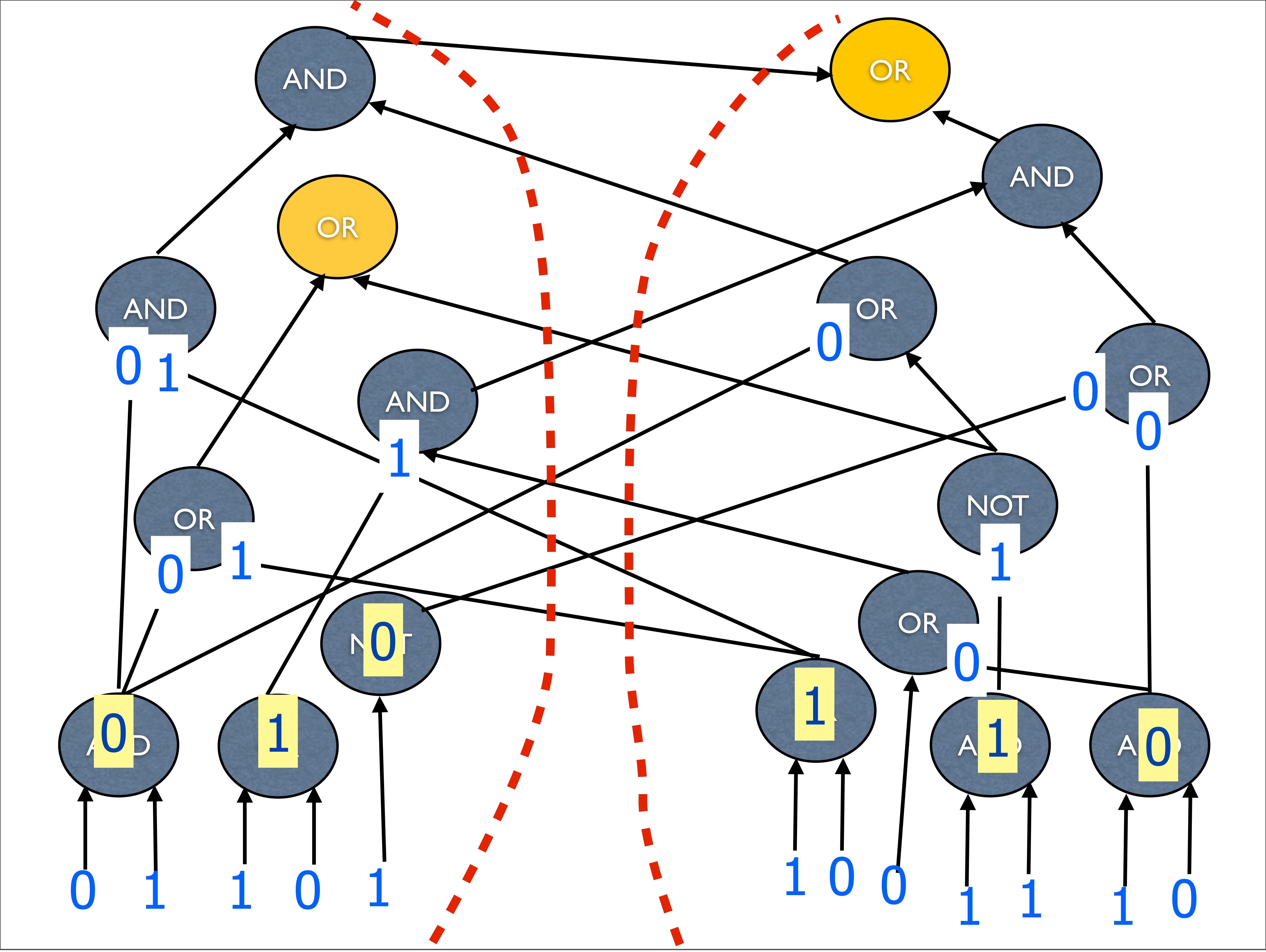
Aside

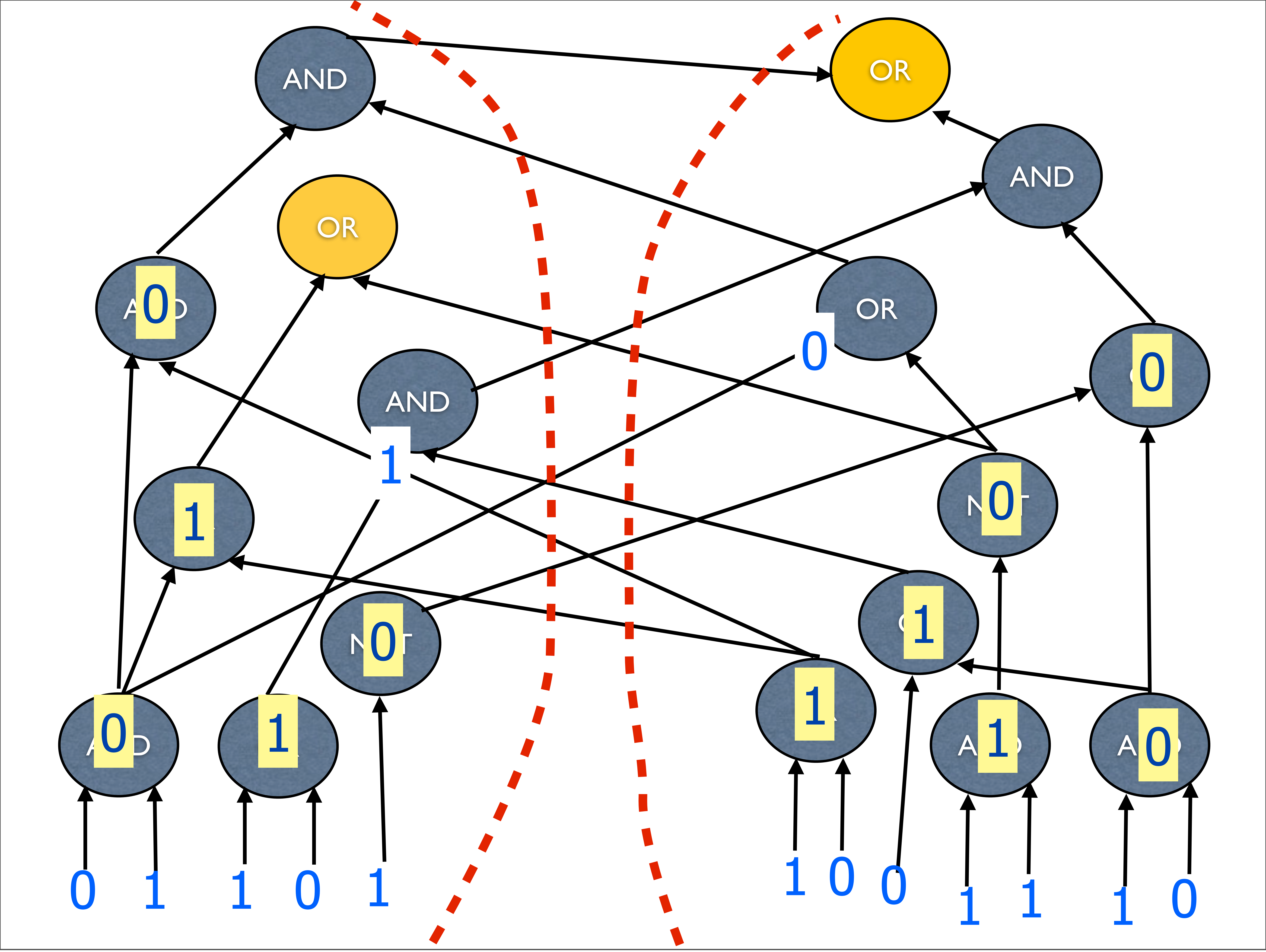
- How to compress interactive communication? - Applications to hardness amplification.

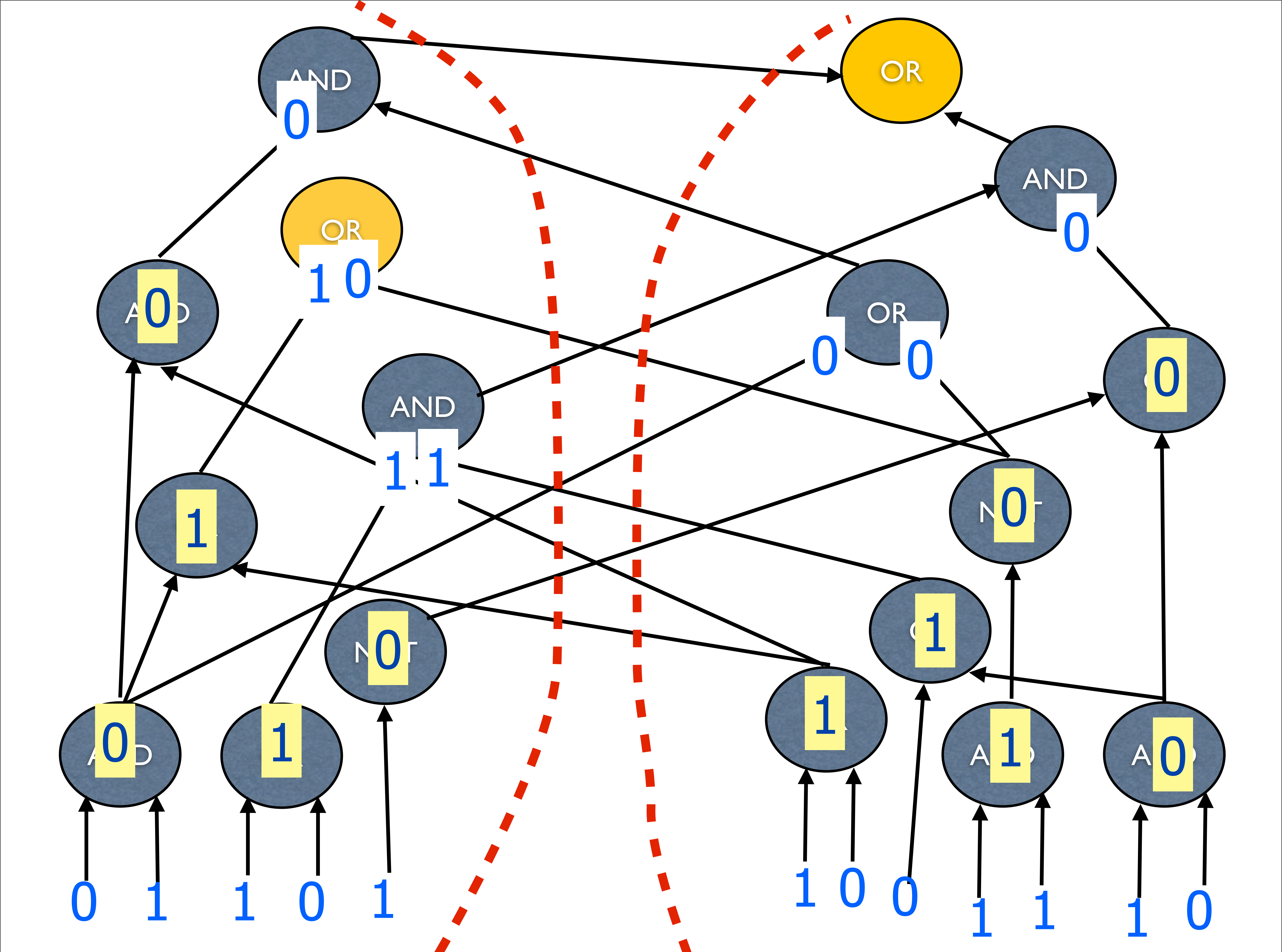

AND
OR
AND
OR
AND
OR
OR
AND
OR
NOT
OR
NOT
OR
AND
OR
OR
AND
AND
0 1 1 0 1
1 0 0 1 1 1 0

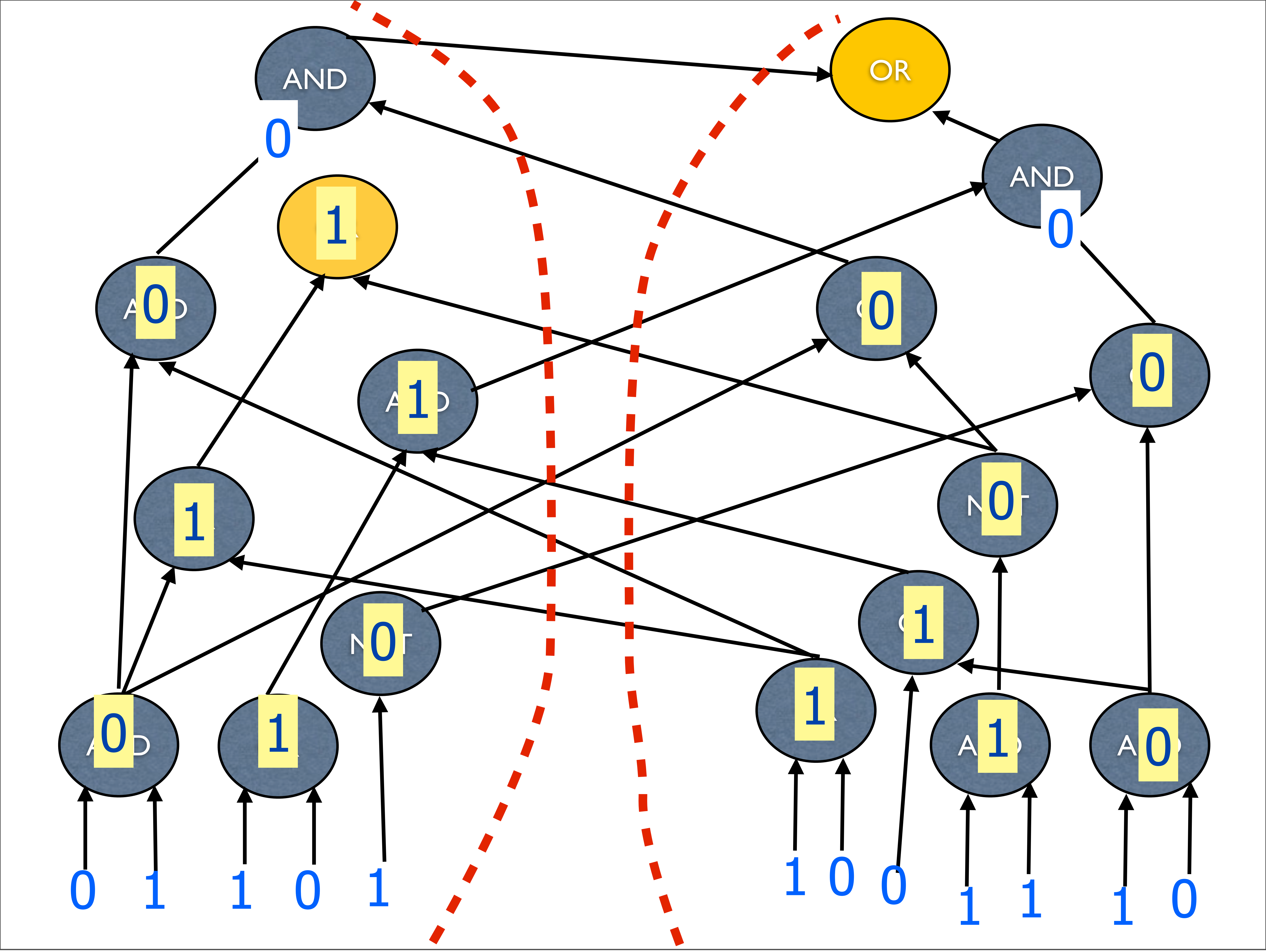

AND
OR
AND
OR
AND
OR
AND
OR
NOT
OR
NOT
OR
AND
OR
OR
AND
AND
0 1 1 0 1
1 0 0 1 1 1 0

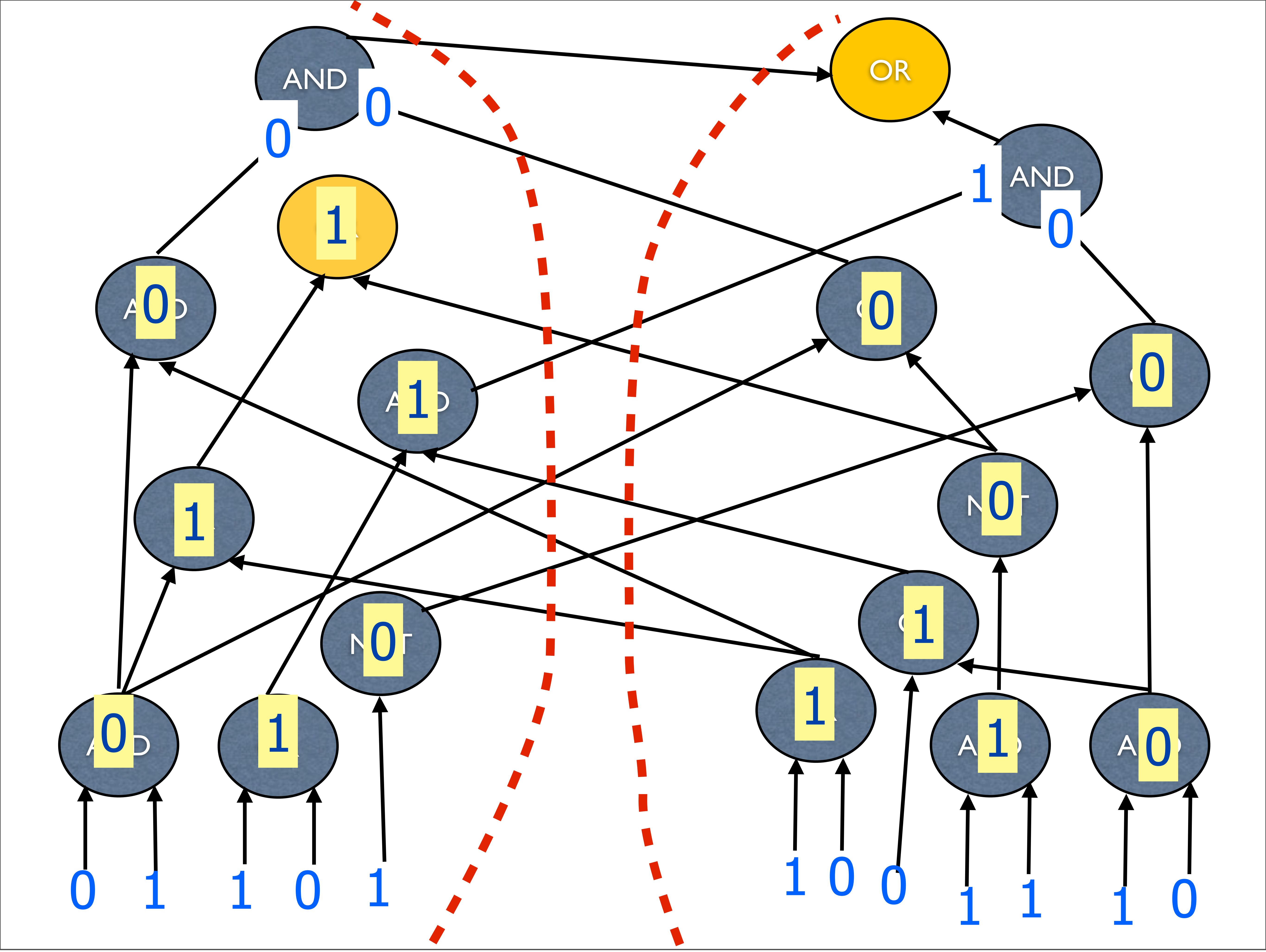

AND
OR
AND
OR
AND
OR
AND
OR
OR
OR
NOT
OR
0
1
1
1
0
0 1 1 0 1
1 0 0 1 1 1 0

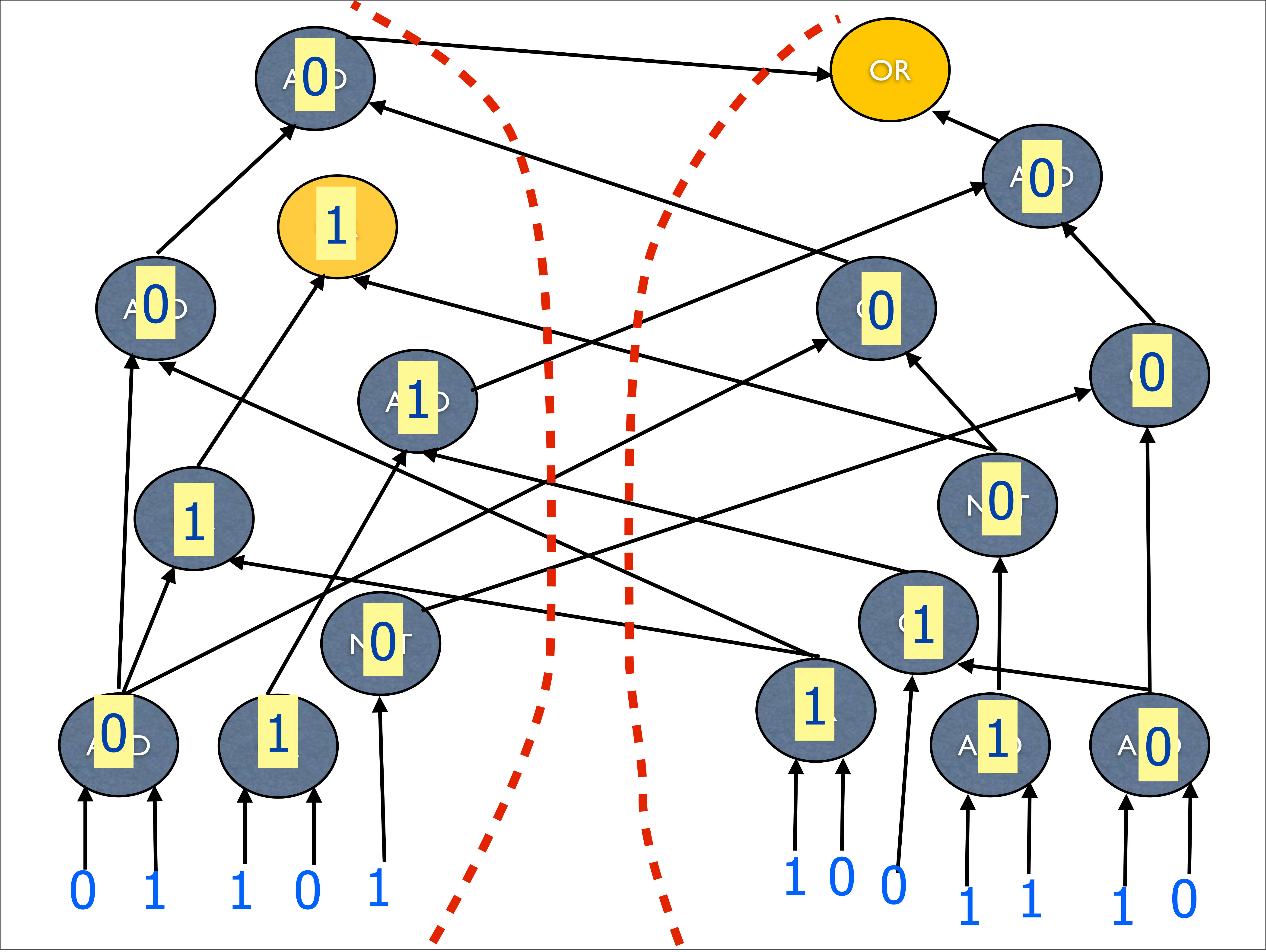

AND
OR
AND
OR
AND
OR
AND
OR
0 1
0
OR
0
0
1
NOT
OR
0 1
0
1
0
1
0
1
1
0
0 1 1 0 1
1 0 0 1 1 1 0

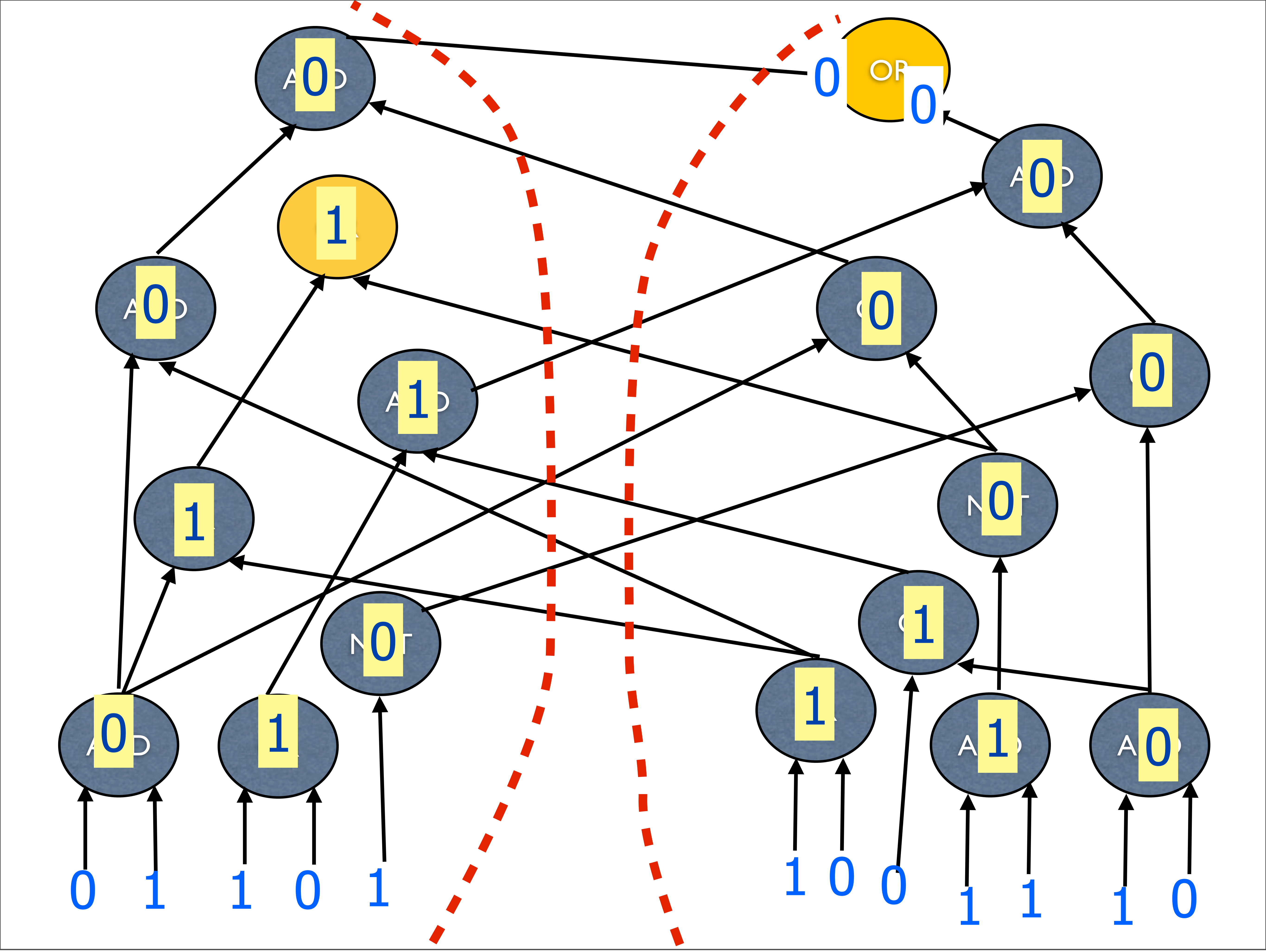

AND
OR
OR
AND
OR
AND
0
1
0
0
1
0
1
0
1
0
1
1
0
0 1 1 0 1
1 0 0 1 1 1 0

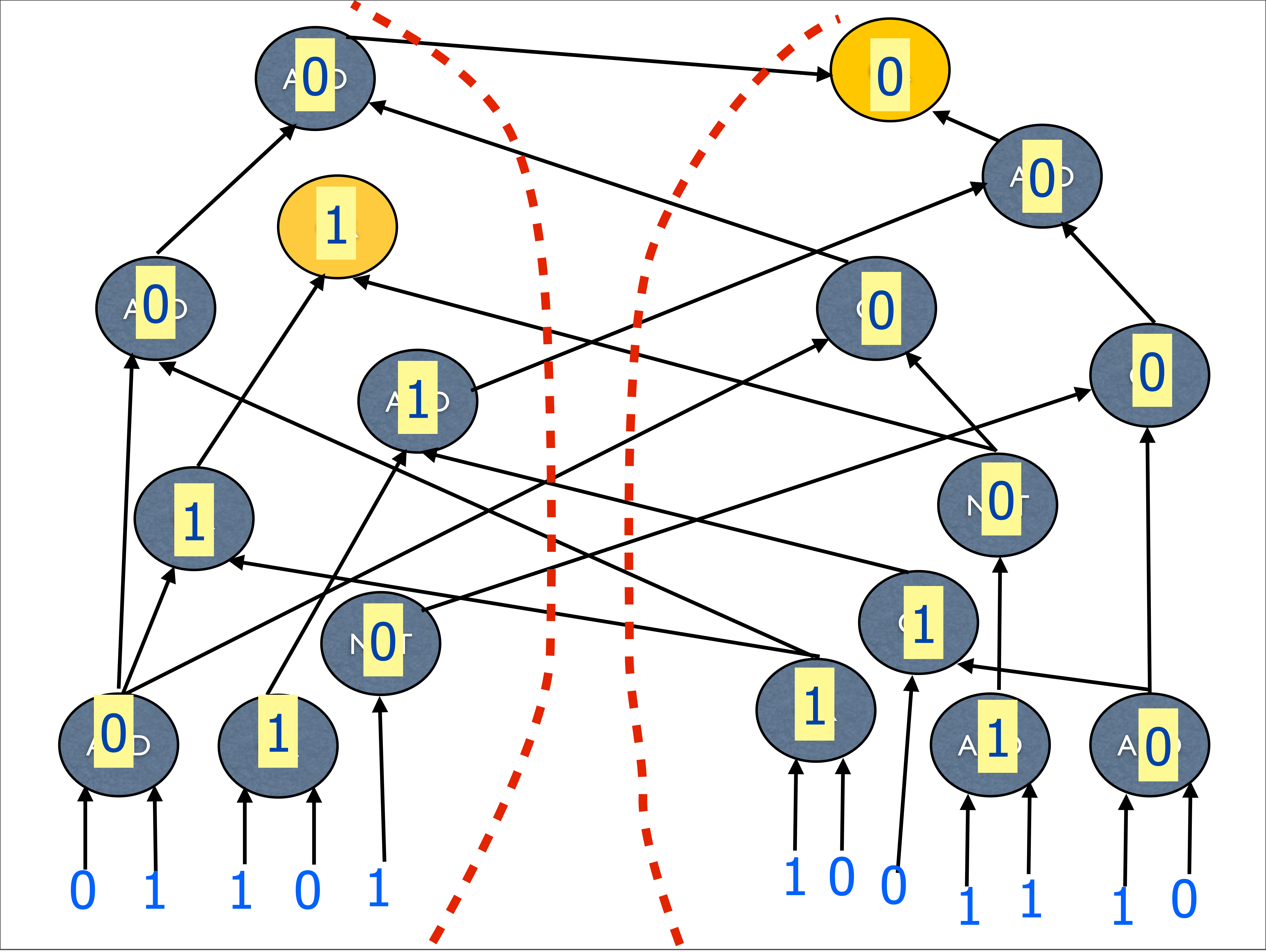

OR
AND
OR
AND
OR
AND
0 1 1 0 1
1 0 0 1 1 1 0

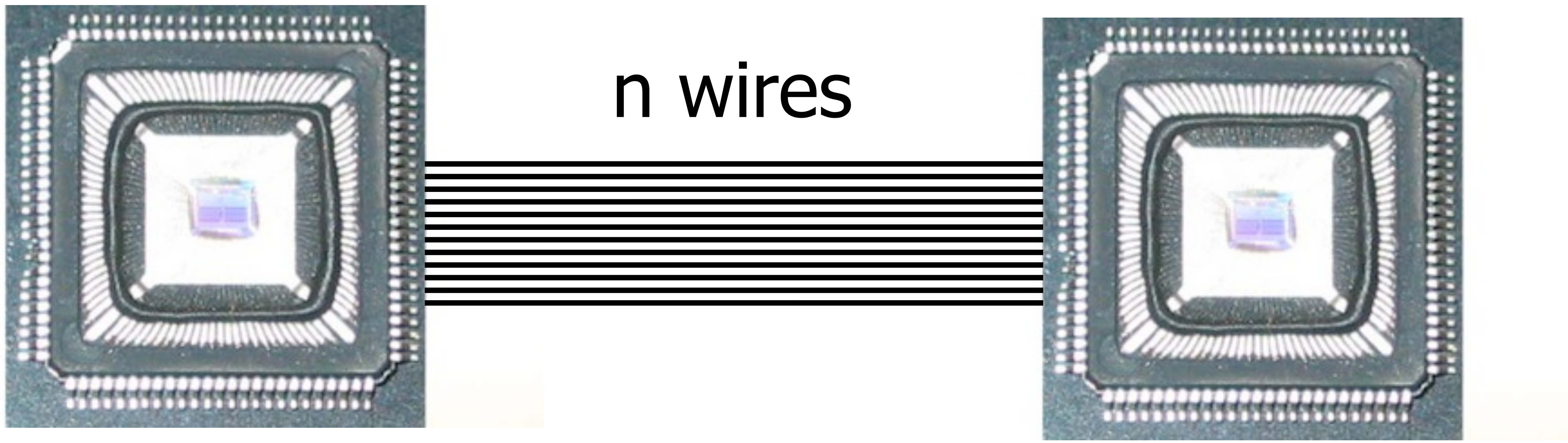

AND
OR
0
AND
0
1
0
0
0
1
0
1
0
1
1
0
1
1
0
0 1 1 0 1
1 0 0 1 1 1 0

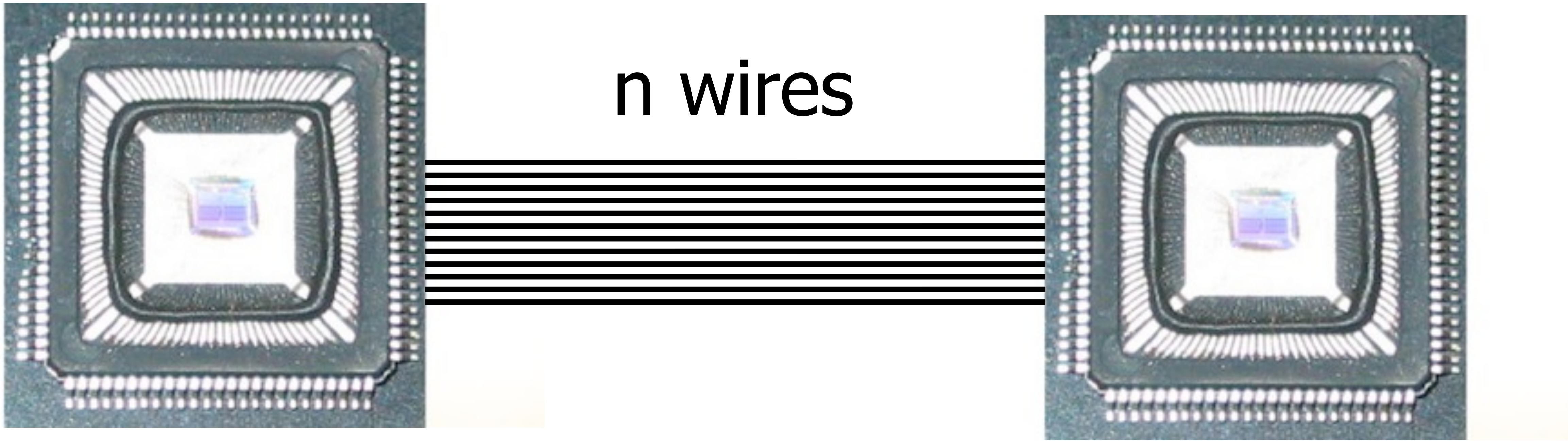

AND
OR
AND
0
0
1
0
1
0
0
0
1
1
0
0
1
0
1
1
0
1
1
0
0
1
1
0
1
1
0
1
1
1
0


OR
0 1 1 0 1
1 0 0 1 1 1 0

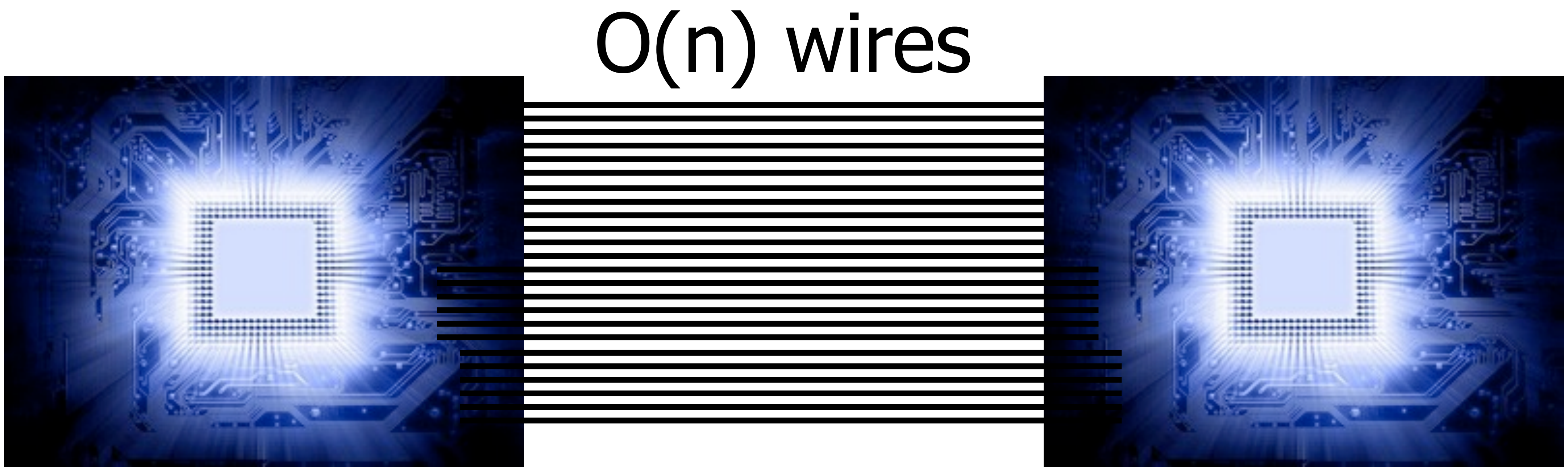

0 1 1 0 1
1 0 0 1 1 1 0
0
0


0 1 1 0 1
1 0 0 1 1 1 0


n wires



Want resilient version:



Want circuit to work even if 10% of wires fail

Error Correction

- First attempt: use code for each round of communication.
- Adversary can corrupt single round completely, to ruin entire outcome. If #rounds is $\omega(1)$, subconstant fraction of corruption.

[Schulman]

x  y

x  y

n bit interaction

O(n) interaction
using constant sized
alphabet

encoded protocol has same effect,
as long
as errors are at most 1/240

[Schulman]

x  y

n bit interaction

x  y

O(n) interaction using constant sized alphabet

encoded protocol has same effect,
as long
as errors are at most 1/240

for good reasons

Our Results

x

y

n bit interaction

x

y

$O(n/\varepsilon)$ interaction
using binary alphabet

encoded protocol has same effect,
as long
as errors are at most $1/8 - \varepsilon$

Our Results

x  y

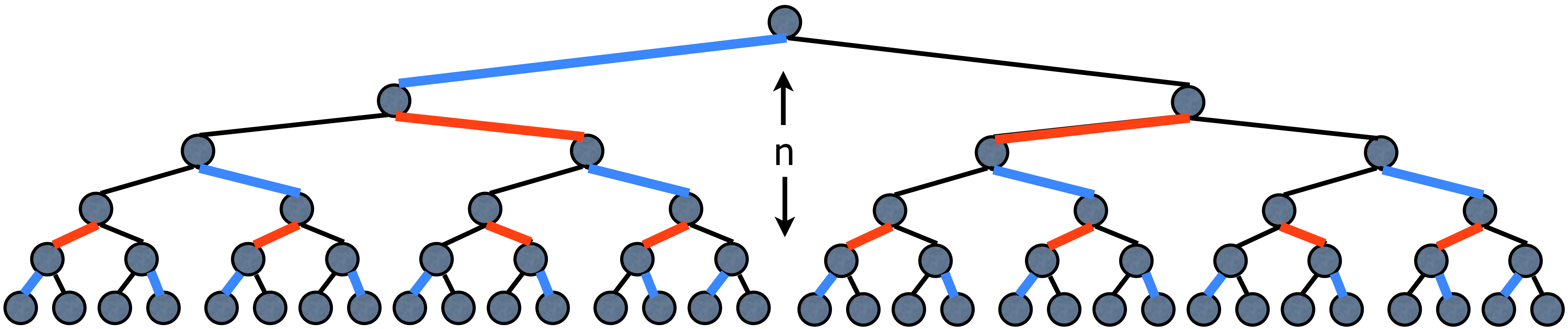
n bit interaction

x  y

O(n/ε) interaction using $O_\varepsilon(1)$ size alphabet

encoded protocol has same effect,
as long
as errors are at most 1/4 - ε

Pointer Jumping

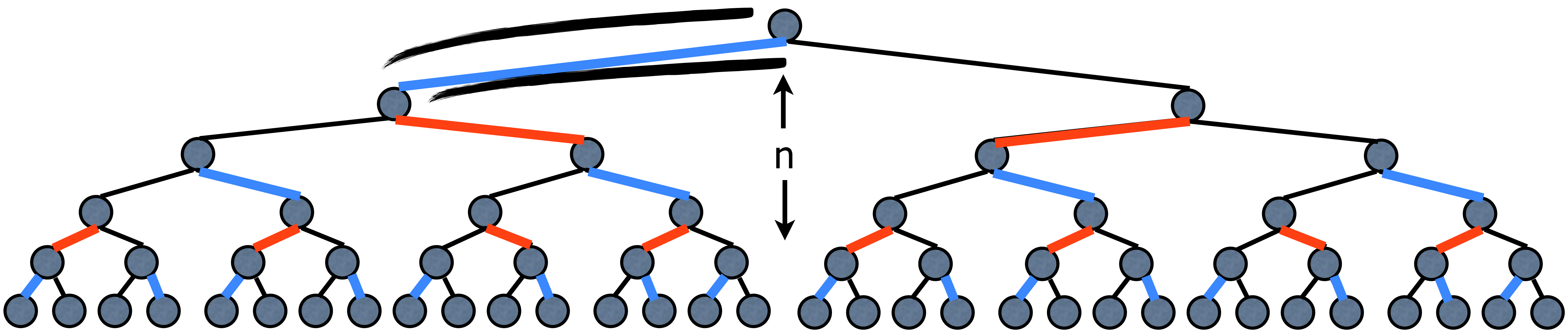


Goal: find the red-blue path

Party 1
(knows even edges)

Party 2
(knows odd edges)

Pointer Jumping



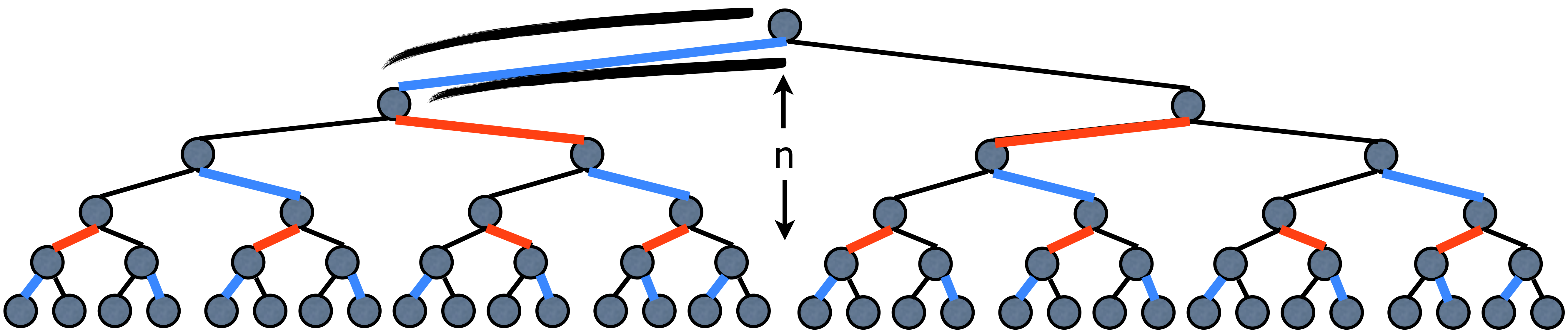
Goal: find the red-blue path

Party 1
(knows even edges)

0

Party 2
(knows odd edges)

Pointer Jumping



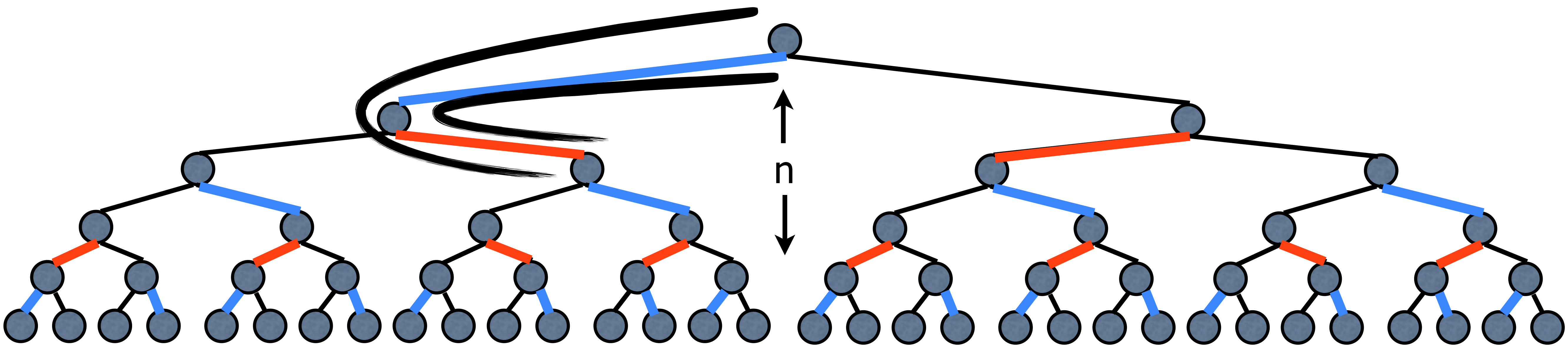
Goal: find the red-blue path

Party 1 (knows even edges)

0

Party 2 (knows odd edges)

Pointer Jumping



Goal: find the red-blue path

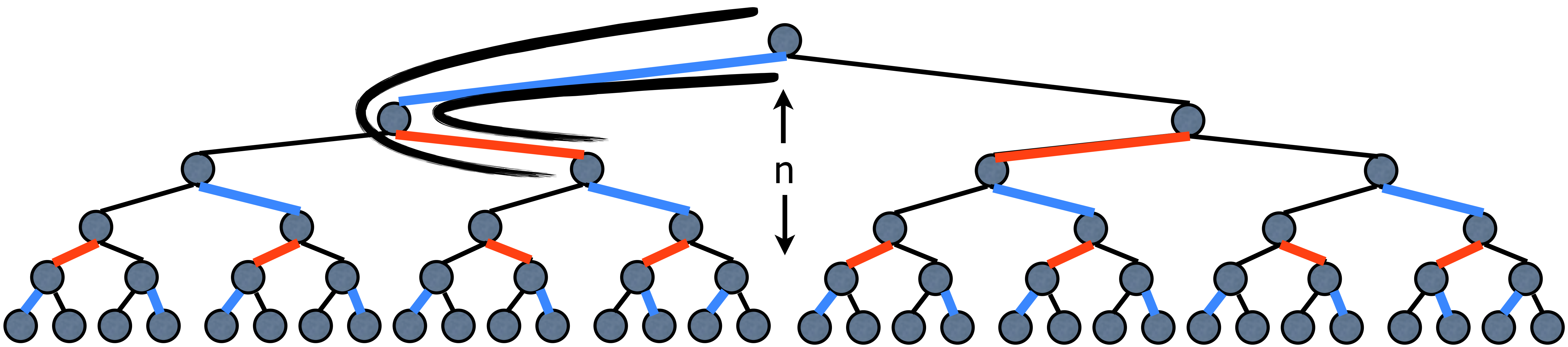
Party 1 (knows even edges)

Party 2 (knows odd edges)

0

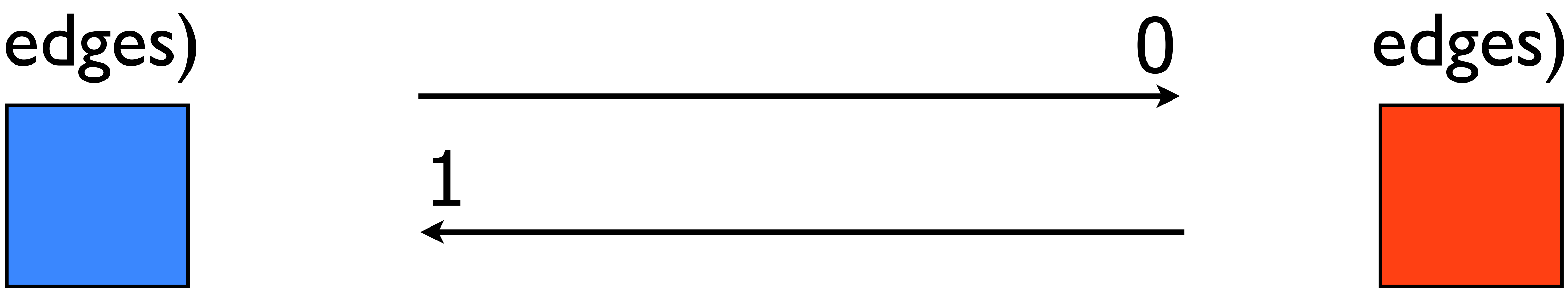
1

Pointer Jumping



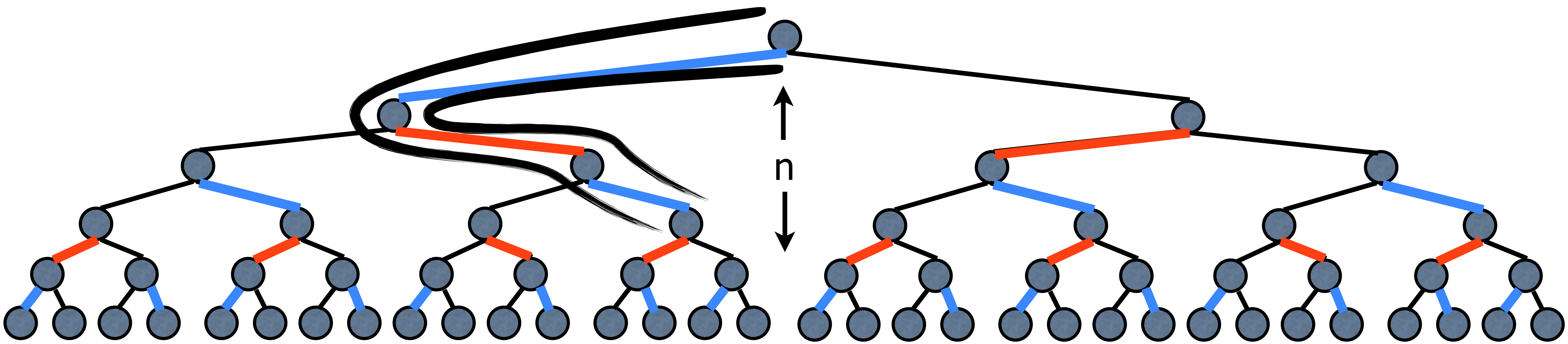
Goal: find the red-blue path

Party 1 (knows even edges)

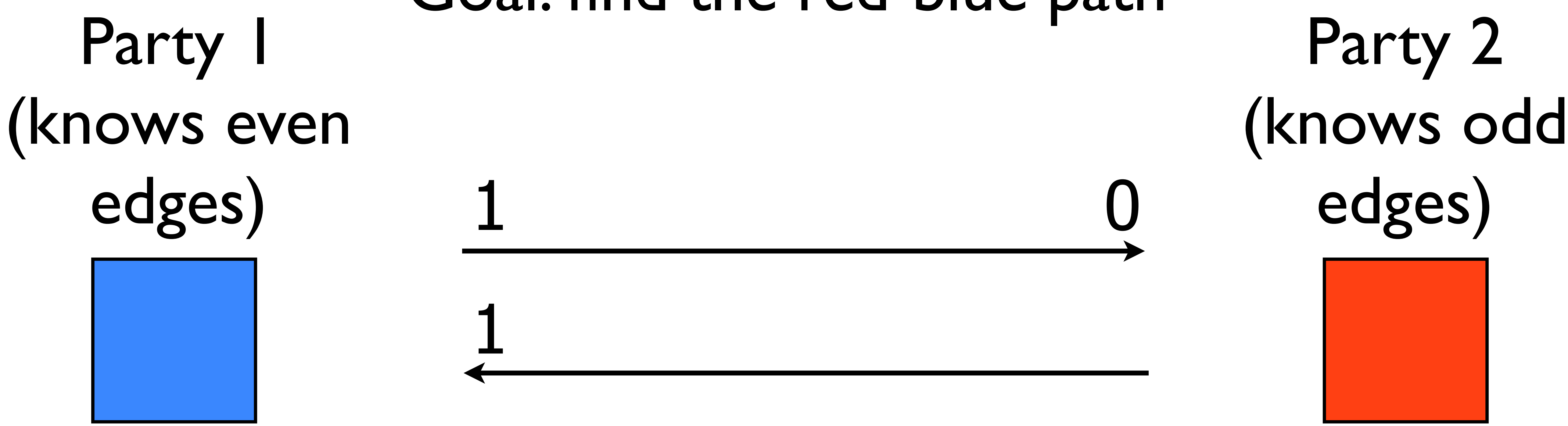


Party 2 (knows odd edges)

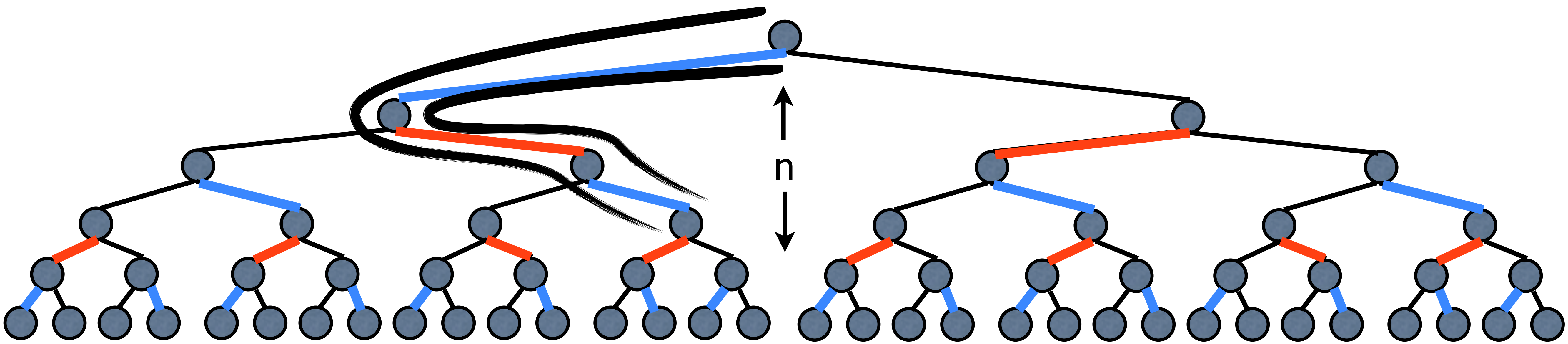
Pointer Jumping



Goal: find the red-blue path



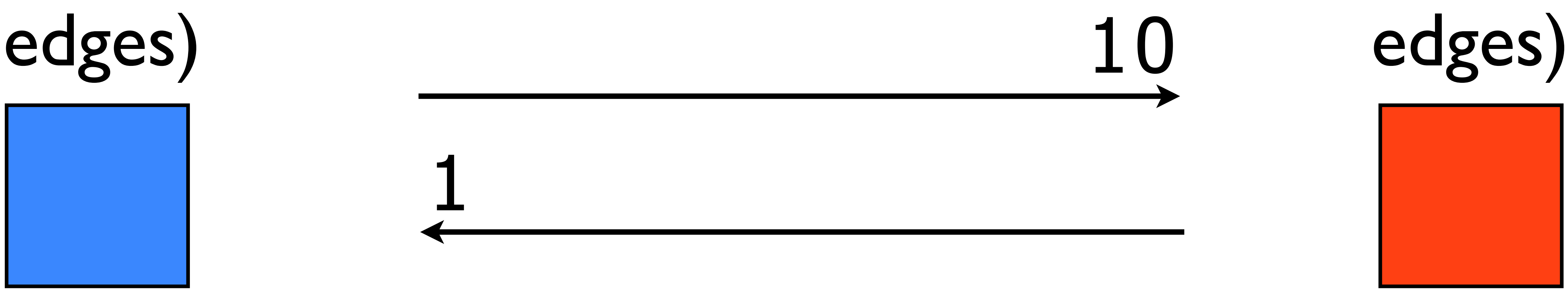
Pointer Jumping



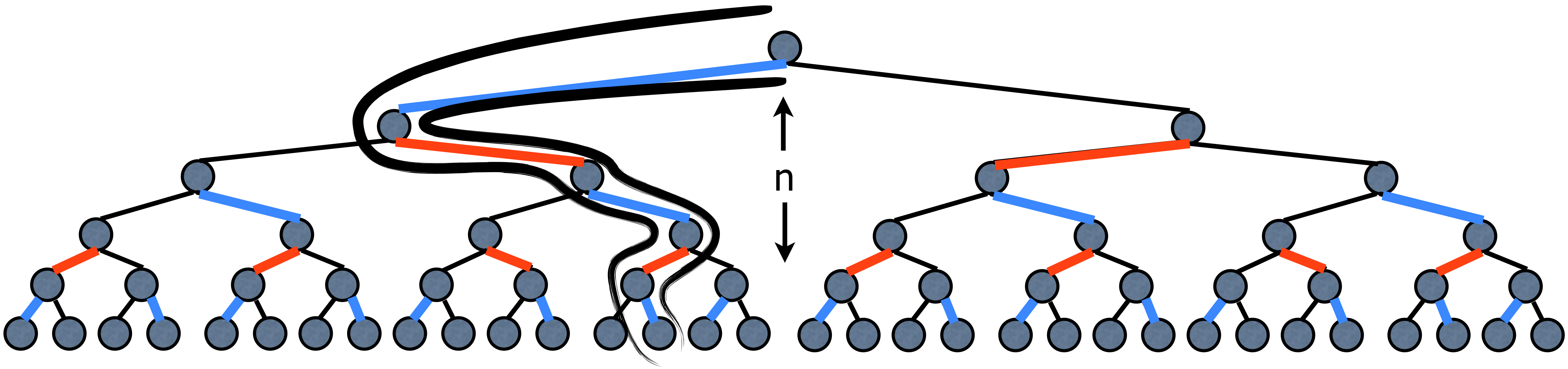
Goal: find the red-blue path

Party 1
(knows even
edges)

Party 2
(knows odd
edges)



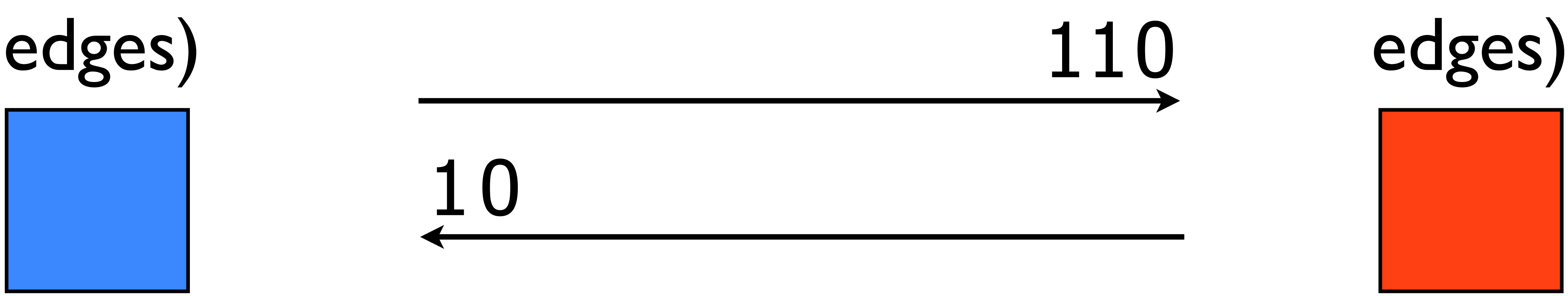
Pointer Jumping



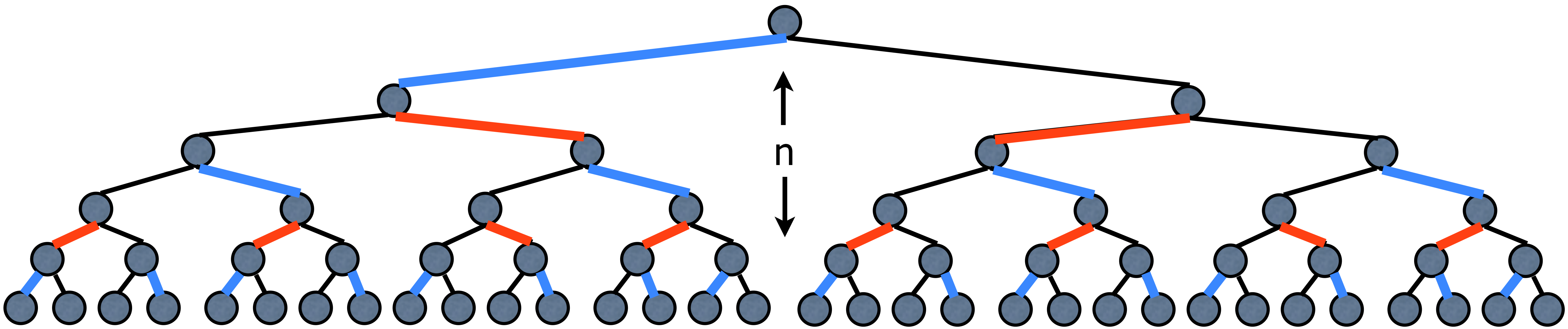
Goal: find the red-blue path

Party 1 (knows even edges)

Party 2 (knows odd edges)



Jumping Over Errors



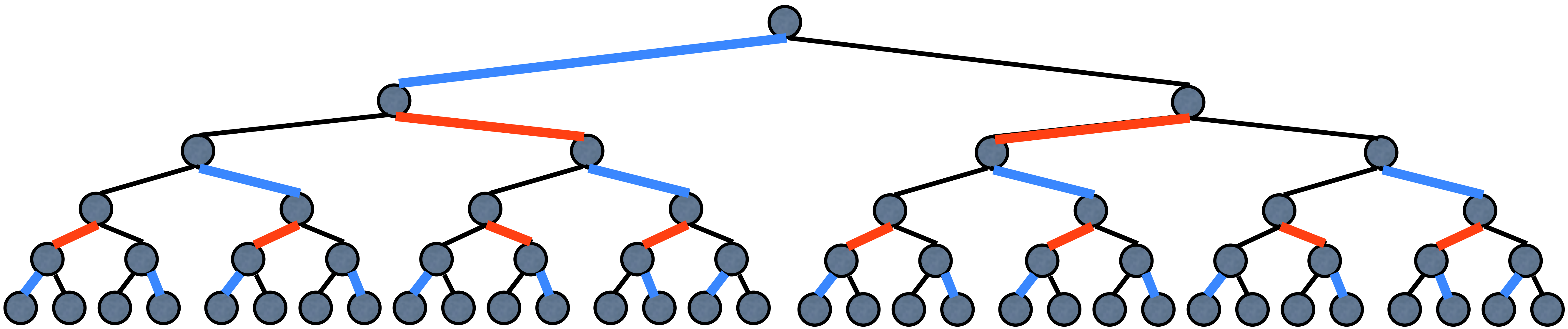
Party 1
(knows even edges)

Goal: find the red-blue path despite errors: transmitted symbols may be corrupted.

Solve this, and you get results for *every* protocol!

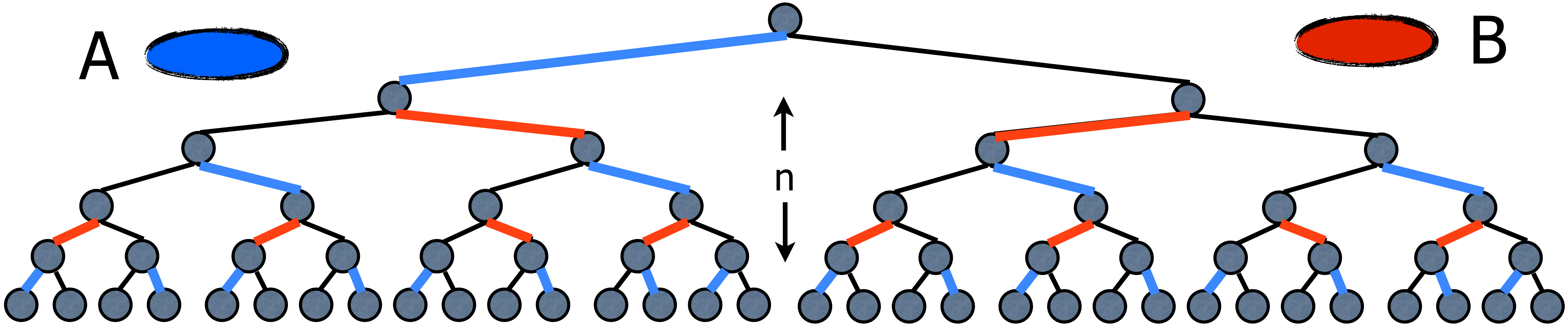
Party 2
(knows odd edges)

Jumping Over Errors



Plan

1. Solve the problem with huge alphabet
2. Solve the problem with reasonable alphabet
3. Solve the problem with constant sized alphabet



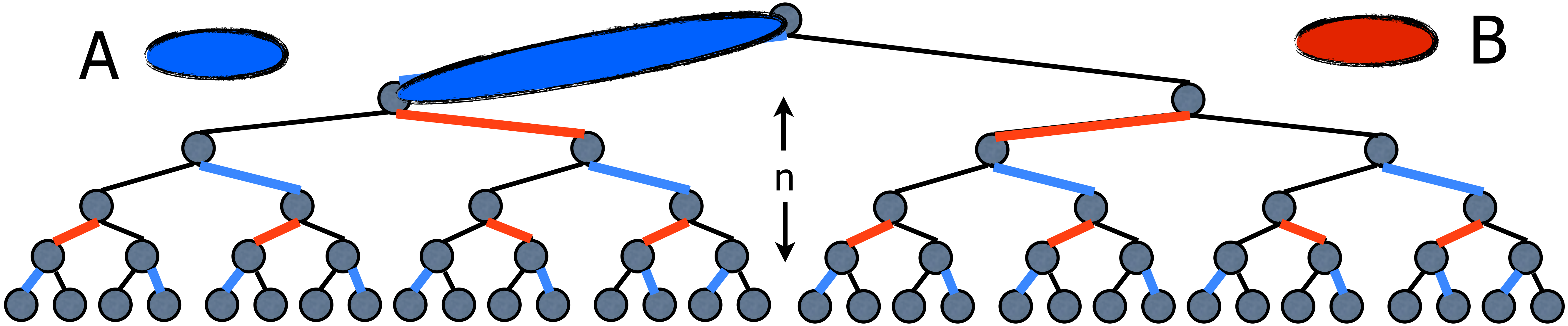
Alphabet: each symbol represents distinct edge (size 2^n)

<u>Protocol for Party 1</u>

A: edges announced by Player 1

B: edges announced by Player 2

Repeat: Announce the edge that extends path in $A \cup B$, if such an edge exists. Else send NULL.



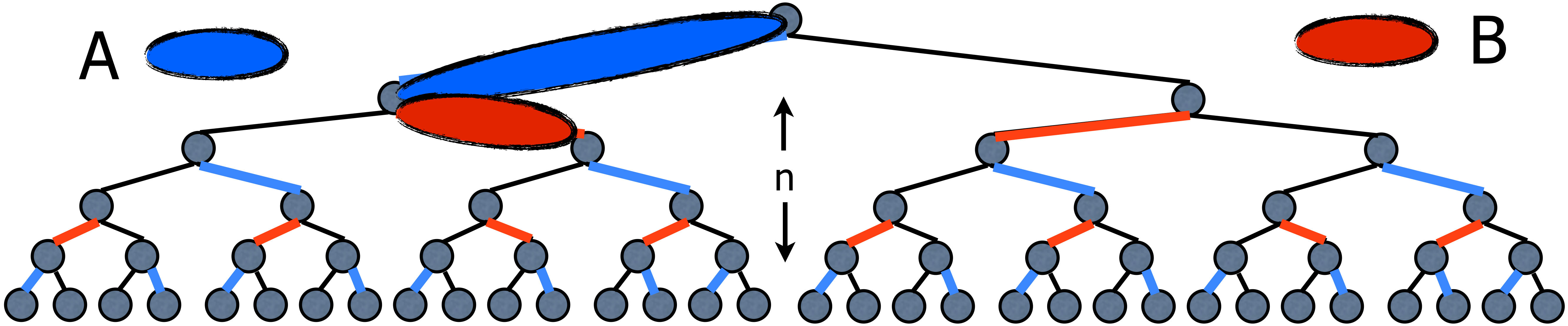
Alphabet: each symbol represents distinct edge (size 2^n)

Protocol for Party 1

A: edges announced by Player 1

B: edges announced by Player 2

Repeat: Announce the edge that extends path in A∪B, if such an edge exists. Else send NULL.



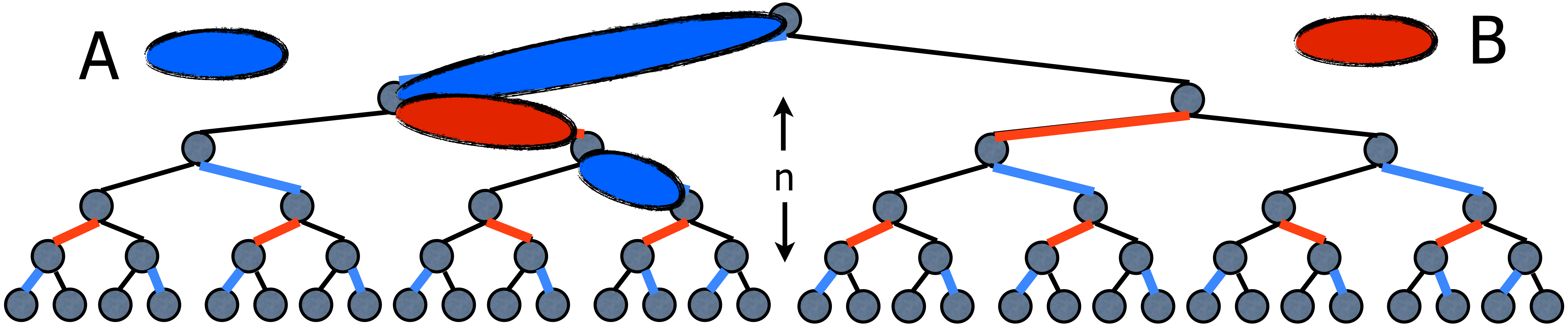
Alphabet: each symbol represents distinct edge (size 2^n)

Protocol for Party 1

A: edges announced by Player 1

B: edges announced by Player 2

Repeat: Announce the edge that extends path in A∪B, if such an edge exists. Else send NULL.



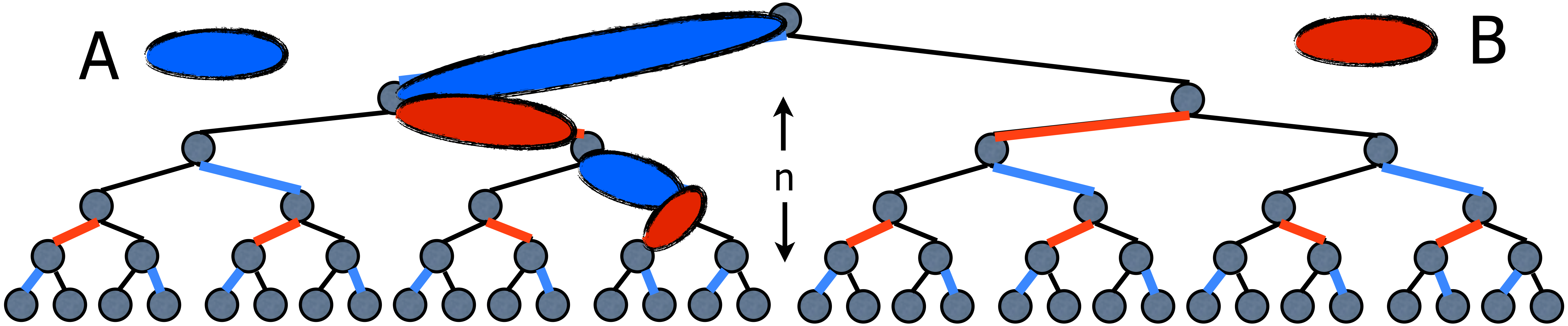
Alphabet: each symbol represents distinct edge (size 2^n)

Protocol for Party 1

A: edges announced by Player 1

B: edges announced by Player 2

Repeat: Announce the edge that extends path in A∪B, if such an edge exists. Else send NULL.



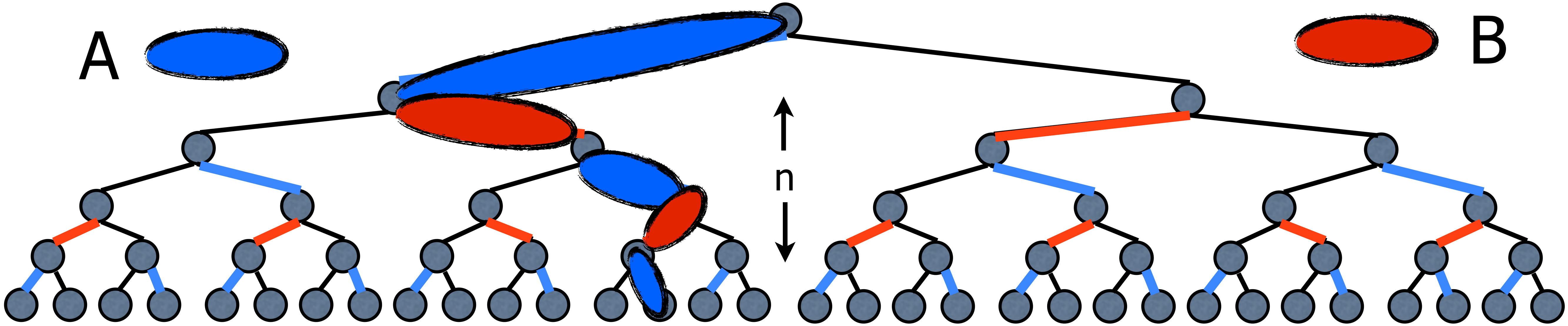
Alphabet: each symbol represents distinct edge (size 2^n)

Protocol for Party 1

A: edges announced by Player 1

B: edges announced by Player 2

Repeat: Announce the edge that extends path in A∪B, if such an edge exists. Else send NULL.



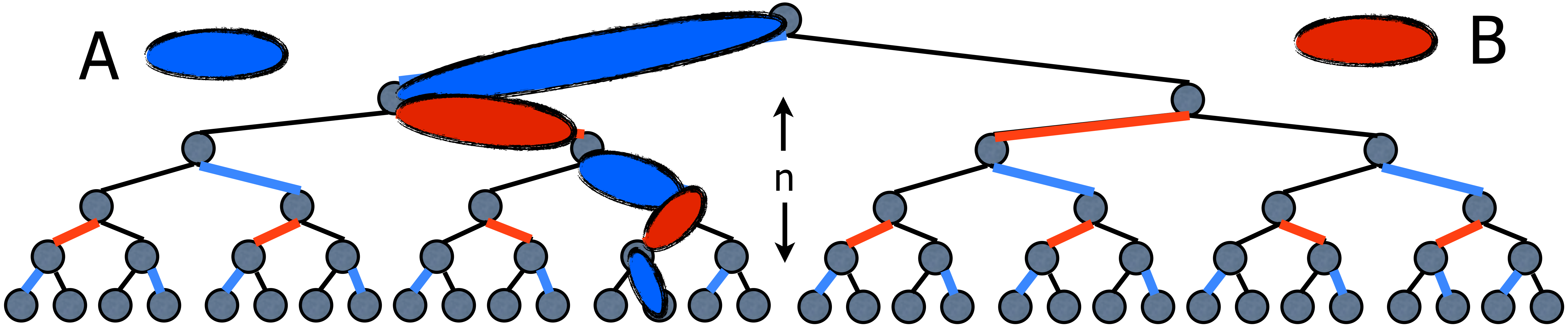
Alphabet: each symbol represents distinct edge (size 2^n)

Protocol for Party 1

A: edges announced by Player 1

B: edges announced by Player 2

Repeat: Announce the edge that extends path in A∪B, if such an edge exists. Else send NULL.



Alphabet: each symbol represents distinct edge (size 2^n)

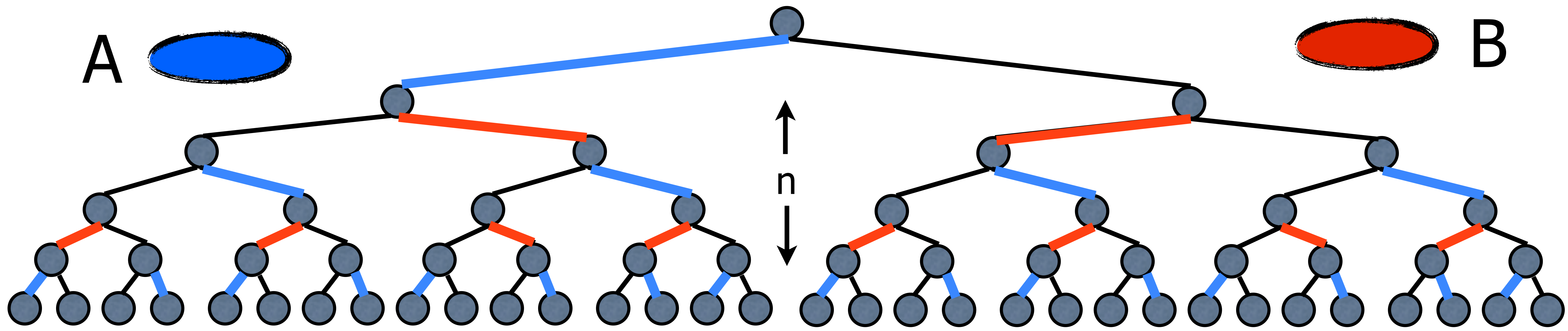
Protocol for Party 1

A: edges announced by Player 1

B: edges announced by Player 2

Repeat: Announce the edge that extends path in A∪B, if such an edge exists. Else send NULL.

Problem: B is not known, since there can be errors!



Alphabet: each symbol represents distinct edge (size 2^n)

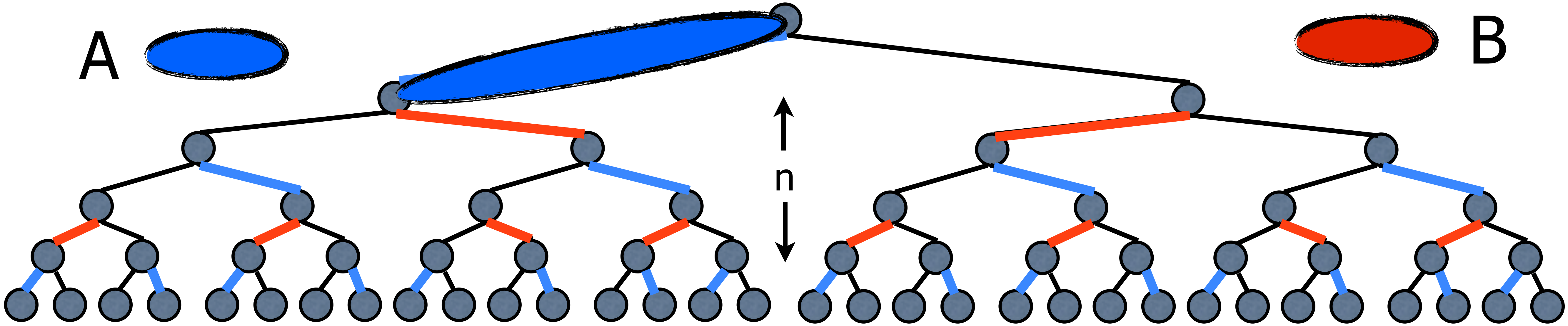
Protocol for Party 1

A: edges announced by Player 1

B: edges announced by Player 2

Repeat: Announce the edge that extends path in $A \cup B$, if such an edge exists. Else send NULL.

Problem: B is not known, since there can be errors!



Alphabet: each symbol represents distinct edge (size 2^n)

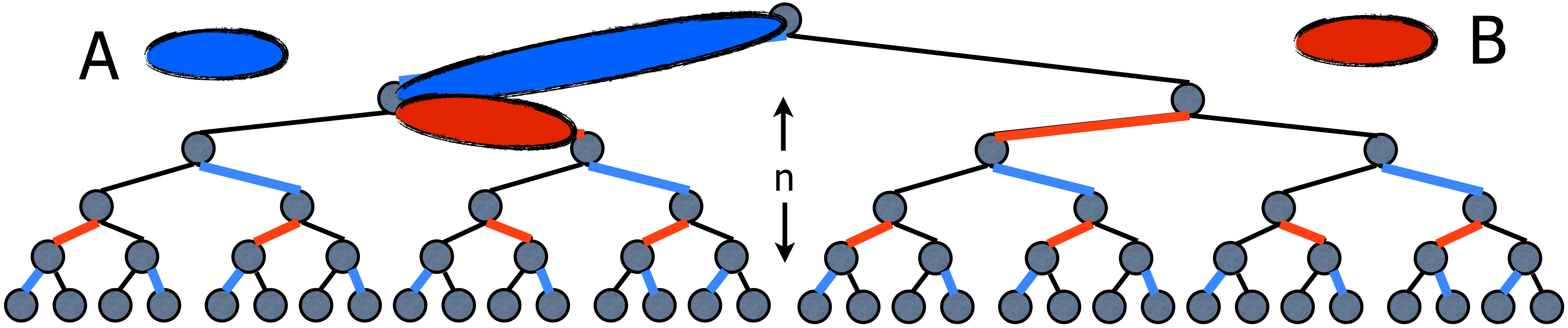
Protocol for Party 1

A: edges announced by Player 1

B: edges announced by Player 2

Repeat: Announce the edge that extends path in A∪B, if such an edge exists. Else send NULL.

Problem: B is not known, since there can be errors!



Alphabet: each symbol represents distinct edge (size 2^n)

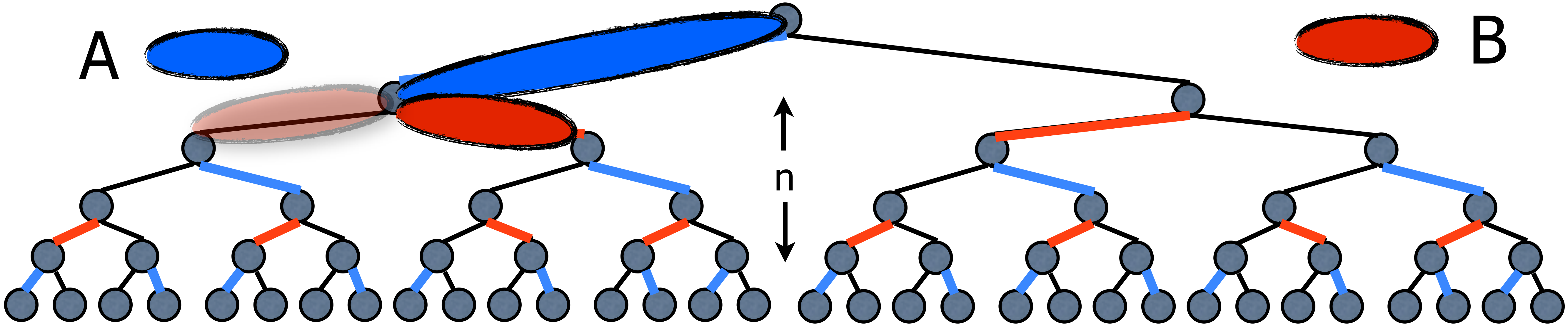
Protocol for Party 1

A: edges announced by Player 1

B: edges announced by Player 2

Repeat: Announce the edge that extends path in $A \cup B$, if such an edge exists. Else send NULL.

Problem: B is not known, since there can be errors!



Alphabet: each symbol represents distinct edge (size 2^n)

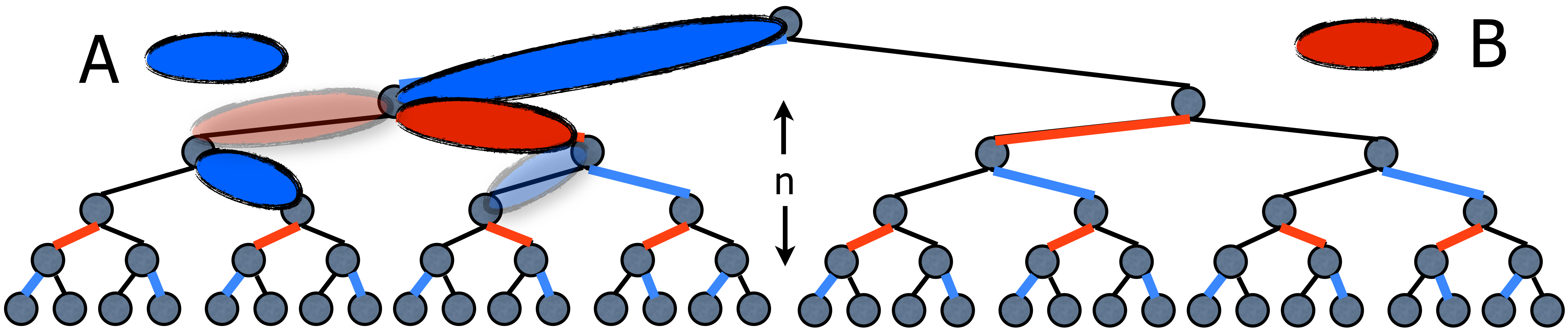
Protocol for Party 1

A: edges announced by Player 1

B: edges announced by Player 2

Repeat: Announce the edge that extends path in $A \cup B$, if such an edge exists. Else send NULL.

Problem: B is not known, since there can be errors!



Alphabet: each symbol represents distinct edge (size 2^n)

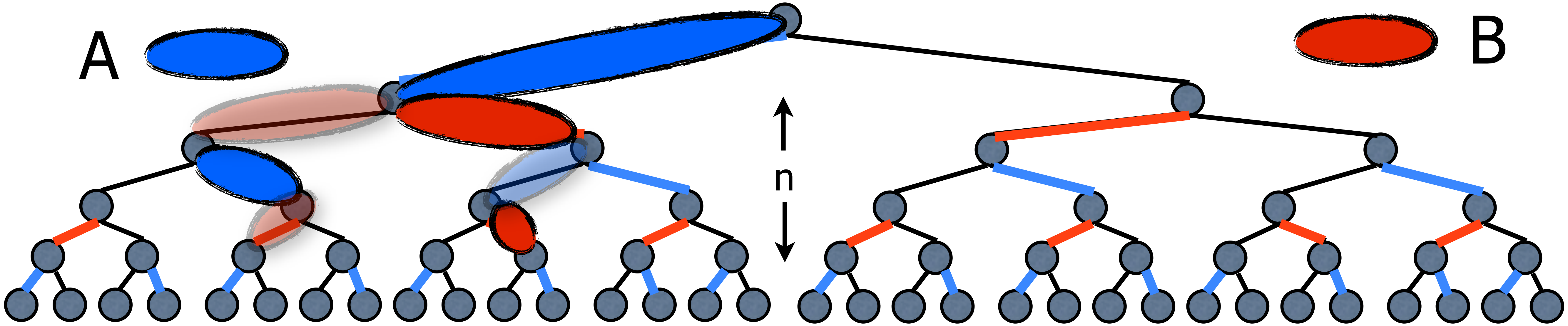
Protocol for Party 1

A: edges announced by Player 1

B: edges announced by Player 2

Repeat: Announce the edge that extends path in A∪B, if such an edge exists. Else send NULL.

Problem: B is not known, since there can be errors!



Alphabet: each symbol represents distinct edge (size 2^n)

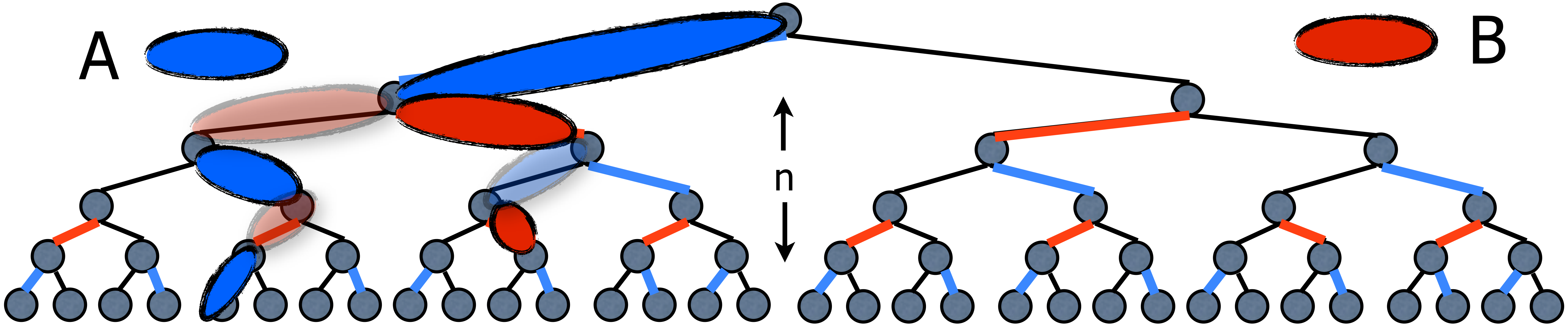
Protocol for Party 1

A: edges announced by Player 1

B: edges announced by Player 2

Repeat: Announce the edge that extends path in $A \cup B$, if such an edge exists. Else send NULL.

Problem: B is not known, since there can be errors!



Alphabet: each symbol represents distinct edge (size 2^n)

Protocol for Party 1

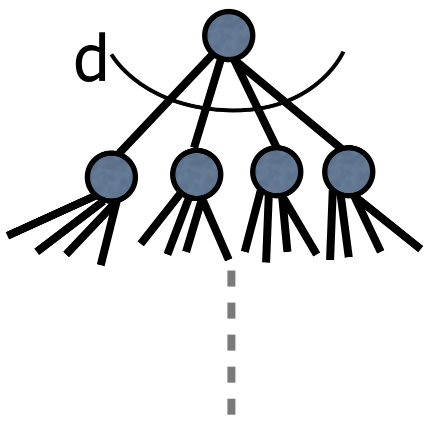
A: edges announced by Player 1

B: edges announced by Player 2

Repeat: Announce the edge that extends path in A∪B, if such an edge exists. Else send NULL.

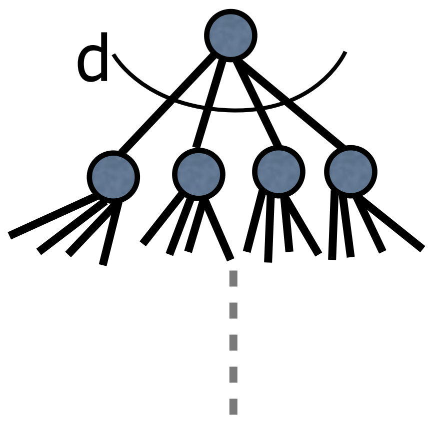
Problem: B is not known, since there can be errors!

d-ary Tree Codes [Schulman]



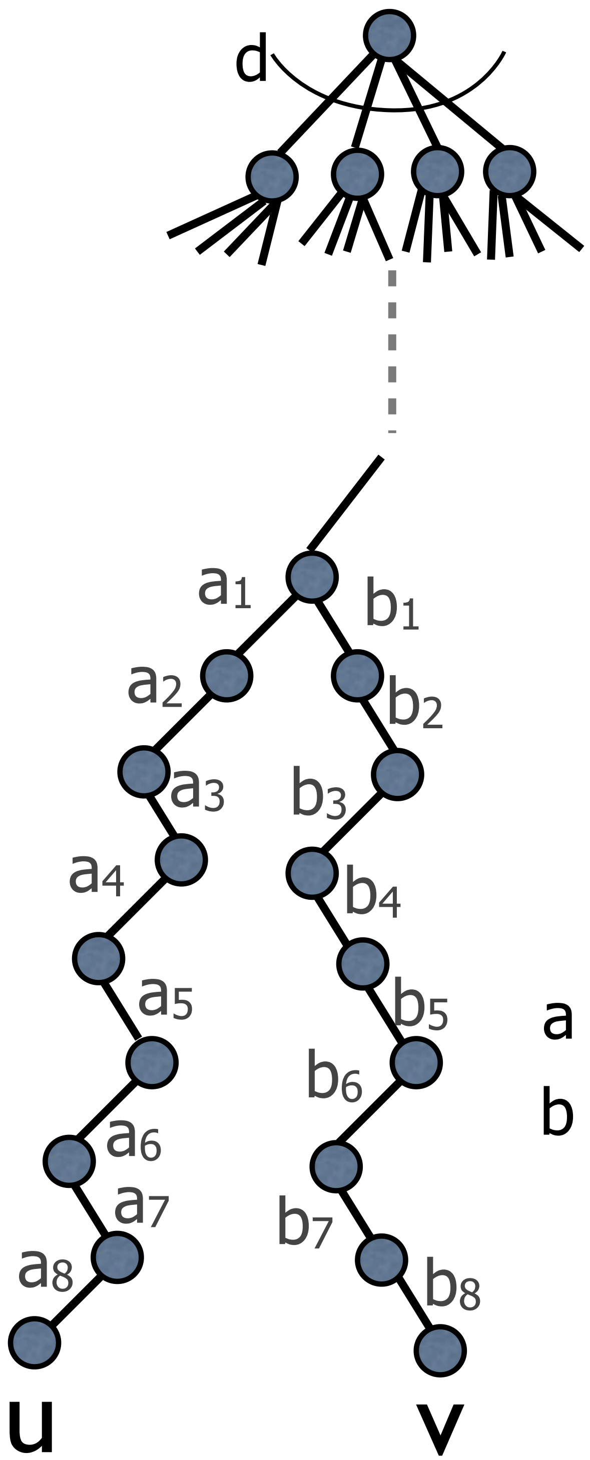
- Edges labeled by symbols from alphabet

d-ary Tree Codes [Schulman]



- Edges labeled by symbols from alphabet

d-ary Tree Codes [Schulman]

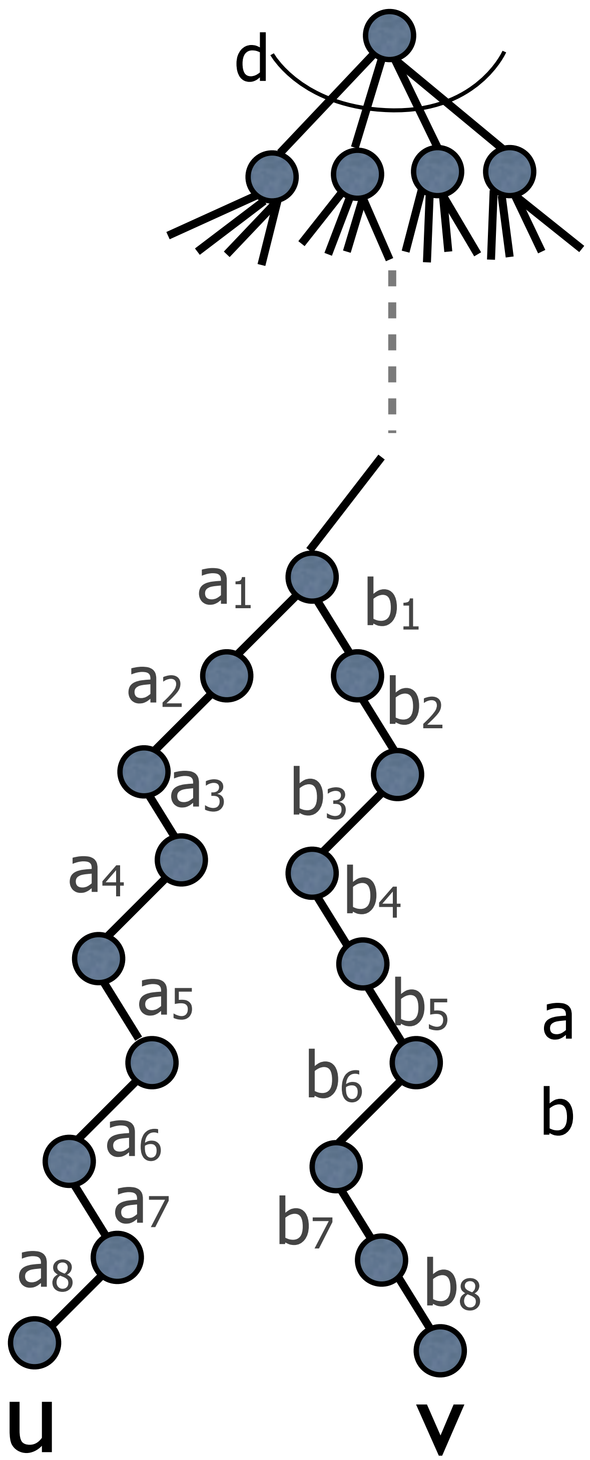


$a = a_1 a_2 \ldots a_8$

$b = b_1 b_2 \ldots b_8$

- Edges labeled by symbols from alphabet
- Distance = 1-ε means for every u,v at same depth $\Delta(a,b) > (1-\varepsilon)\ |a|$

d-ary Tree Codes [Schulman]

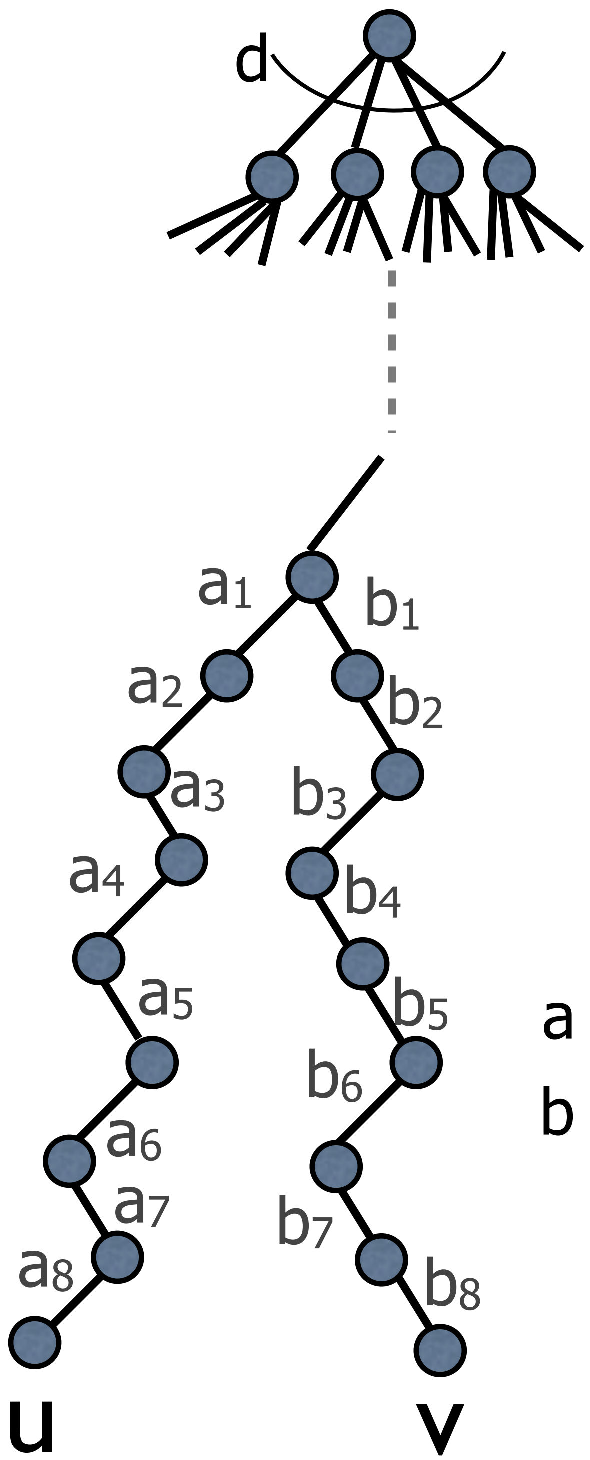


$a = a_1 a_2 \ldots a_8$

$b = b_1 b_2 \ldots b_8$

- Edges labeled by symbols from alphabet
- Distance = $1-\varepsilon$ means for every u,v at same depth $\Delta(a,b) > (1-\varepsilon)\,|a|$
- alphabet of size $d^{O(1/\varepsilon)}$ enough!

d-ary Tree Codes



$a = a_1 a_2 \ldots a_8$

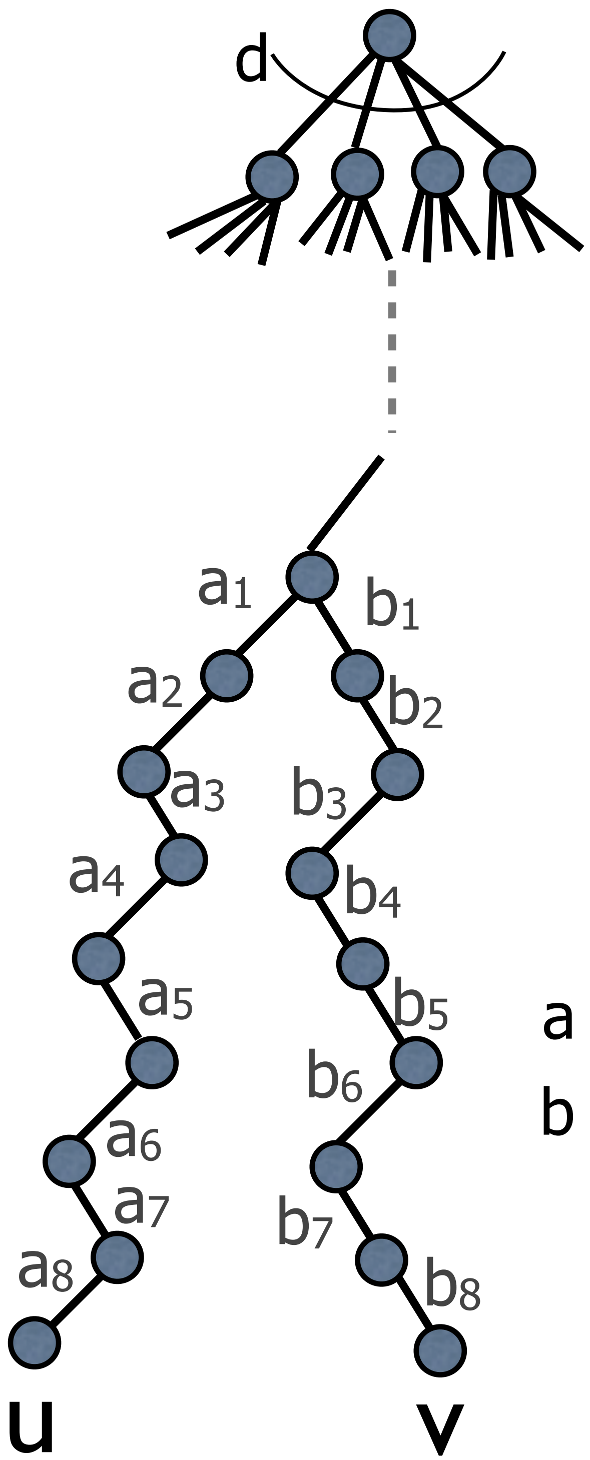
$b = b_1 b_2 \ldots b_8$

- Distance = 1-ε means for every u,v
  $\Delta(a,b) > (1-\varepsilon)\,|a|$
- alphabet of size $d^{O(1/\varepsilon)}$ enough!

<u>Using Tree Codes</u>

Party 1 $\xrightarrow{\text{path to u} \qquad \text{^&^*#&(}}$ Party 2

d-ary Tree Codes



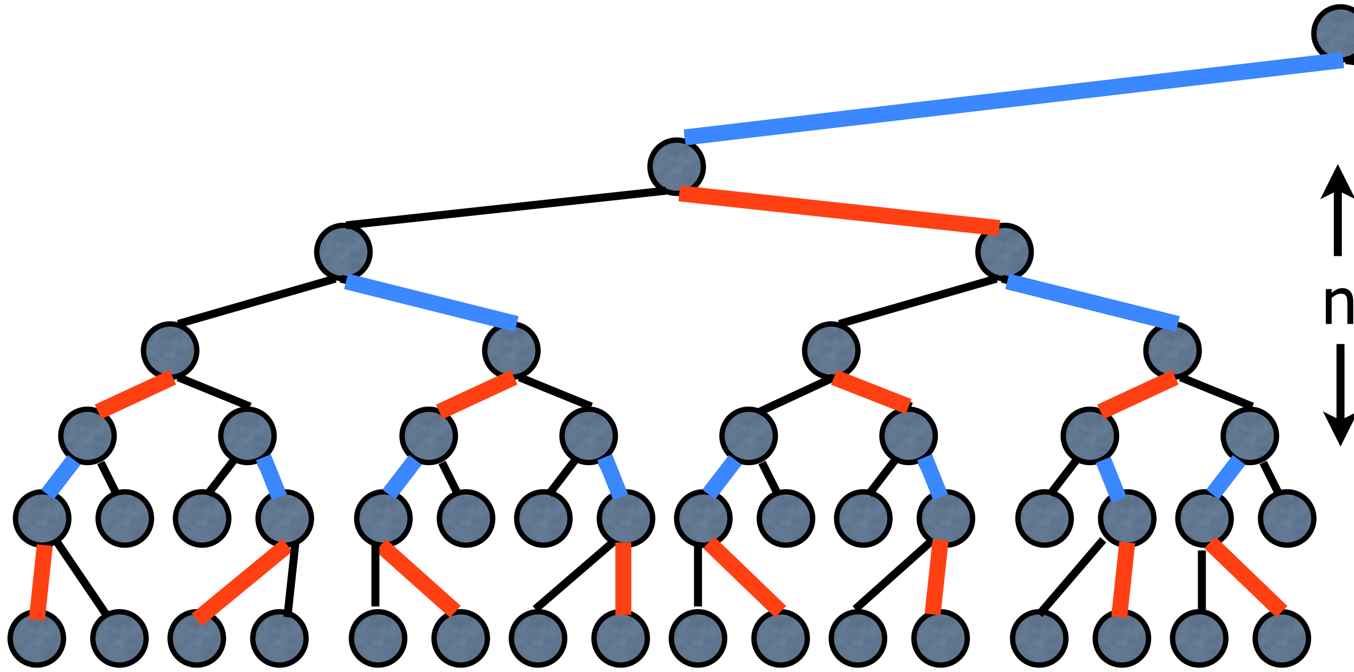
$a = a_1 a_2 \ldots a_8$

$b = b_1 b_2 \ldots b_8$

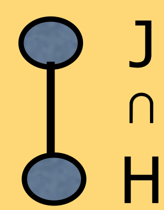
- Distance = $1-\varepsilon$ means for every u,v
  $\Delta(a,b) > (1-\varepsilon)\,|a|$
- alphabet of size $d^{O(1/\varepsilon)}$ enough!

Using Tree Codes

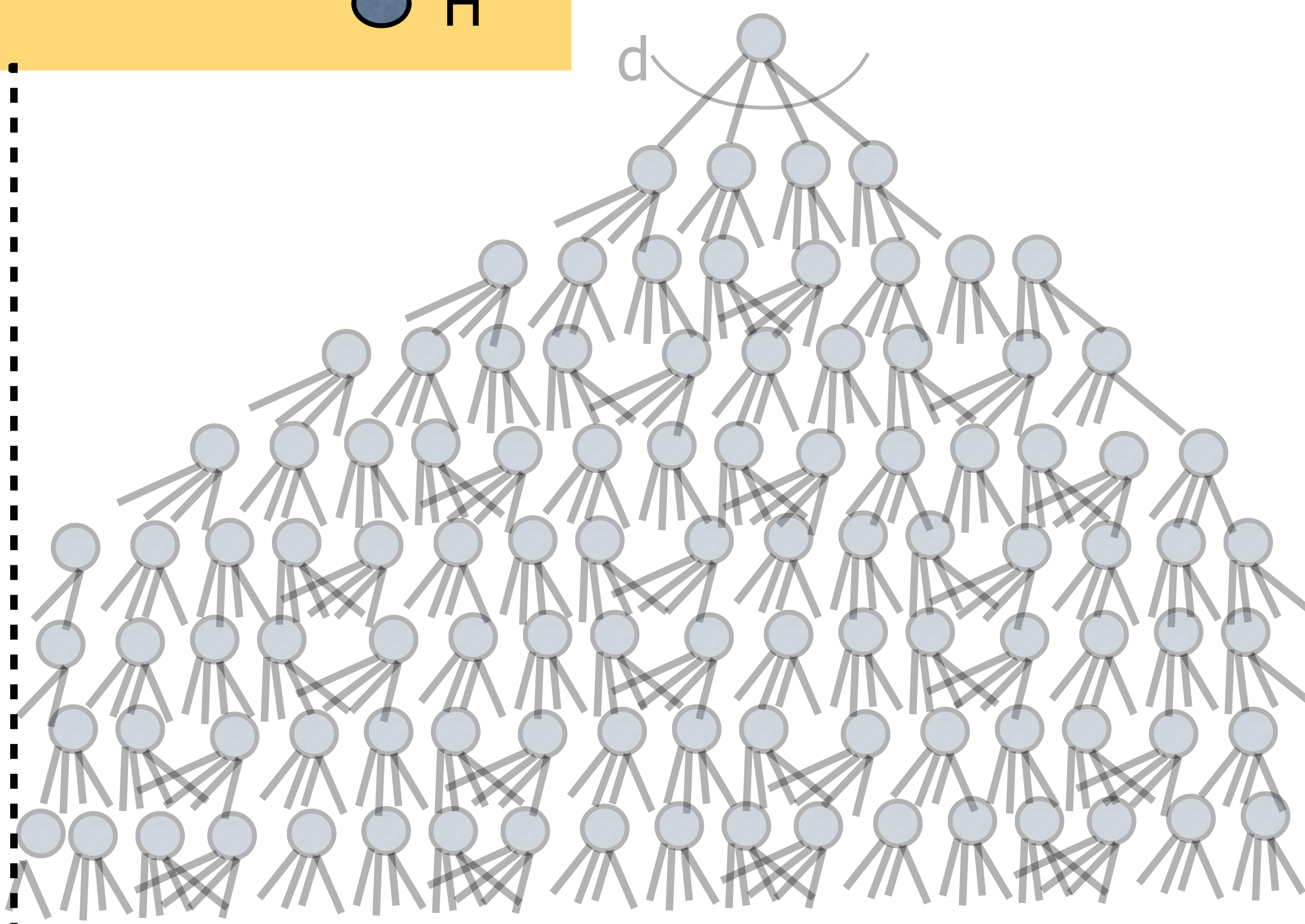
If v is decoded instead of u, #errors in last |a| transmissions must exceed $(1-\varepsilon)|a|/2$



Distance = 1-ε, Depth $O(n/\varepsilon)$. $d = 2^n$
Vertex at depth k = set of at most k edges.



Tree Code

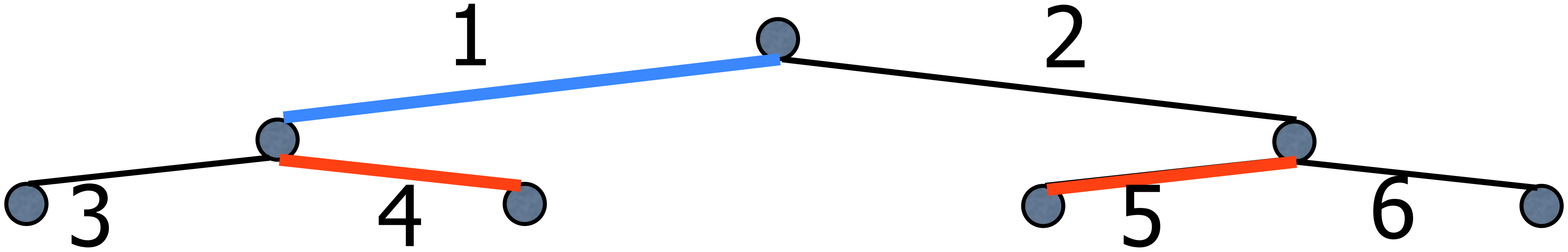


Protocol for Party 1

A: edges announced by Player 1

B′: decoding of edges announced by Player 2

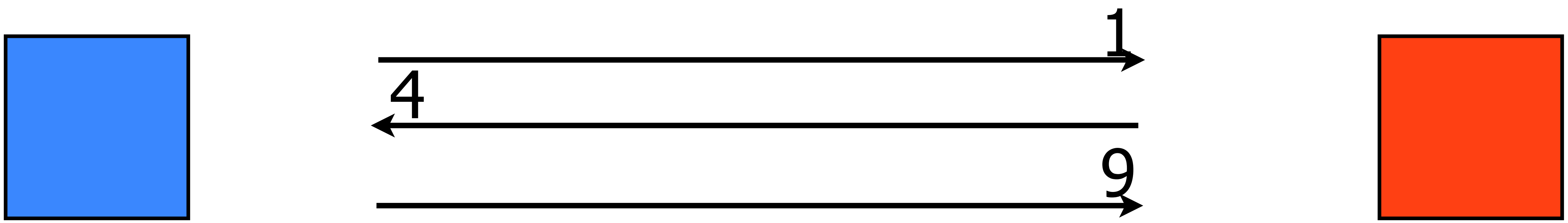
Repeat: Send edge that extends path in A∪B′, if possible.



Party 1

Before

Party 2

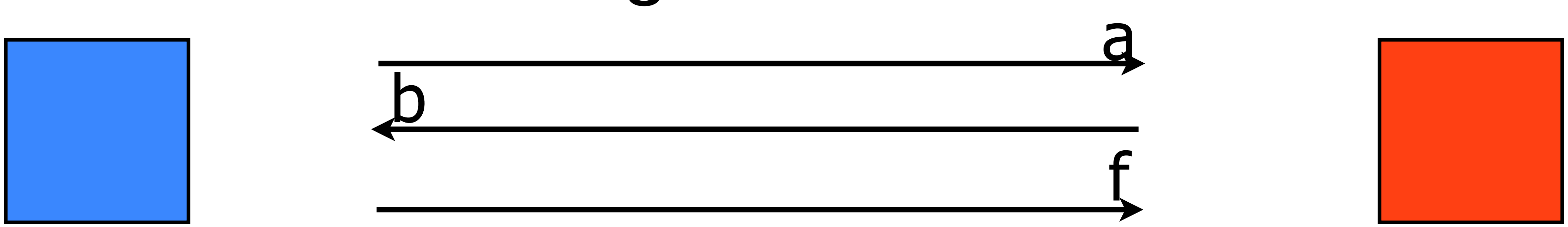


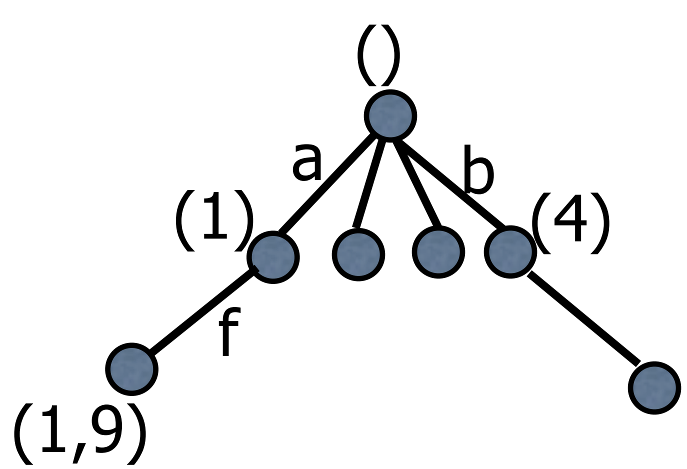
Assuming no errors....

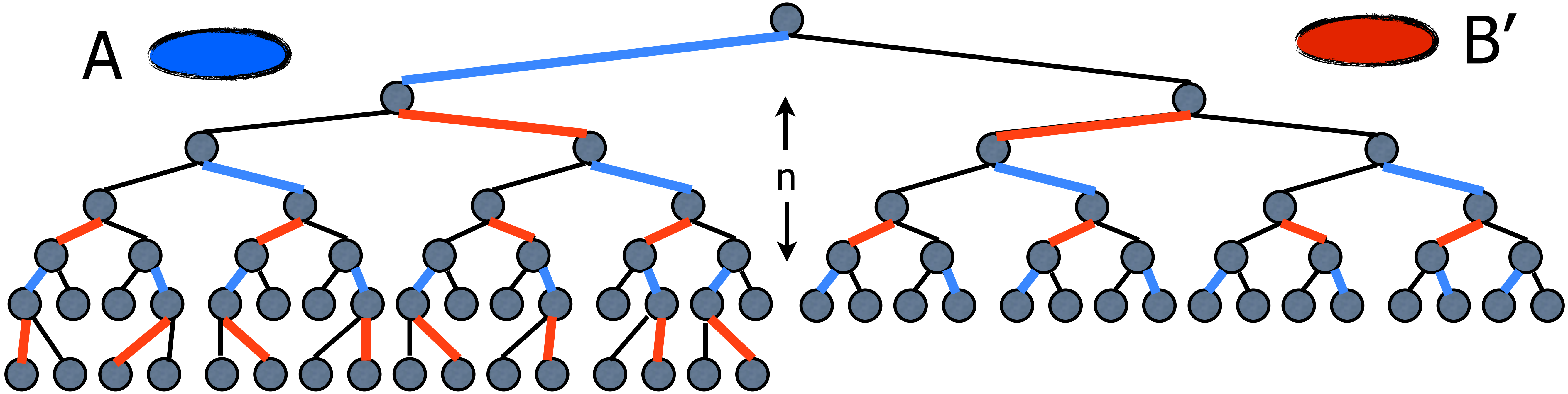
Party 1

Using Tree Codes

Party 2



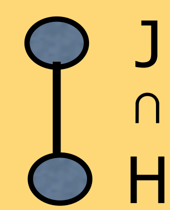


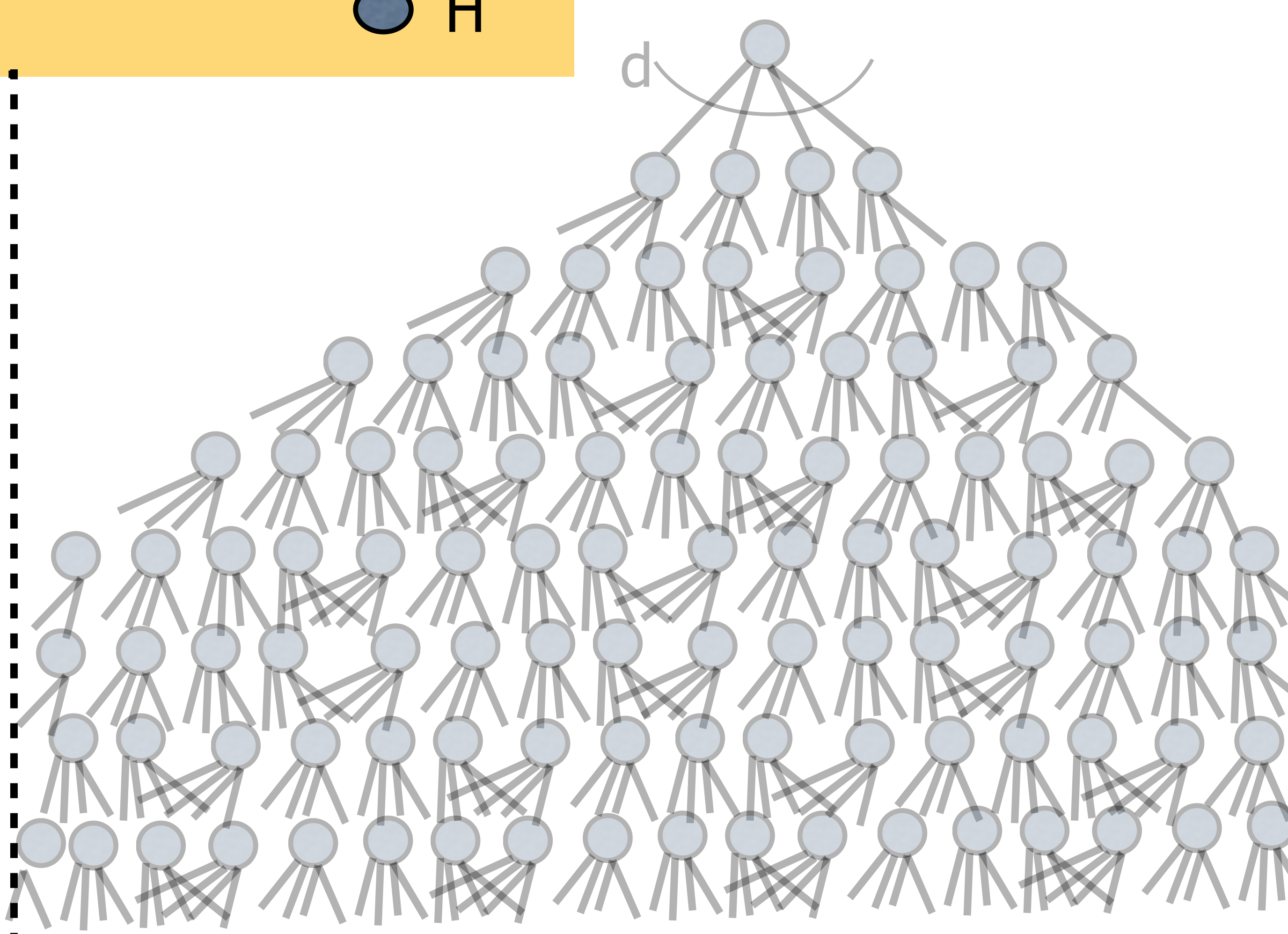


Tree Code

Distance = $1-\varepsilon$, Depth $O(n/\varepsilon)$. $d = 2^n$

Vertex at depth k = list of at most k edges.



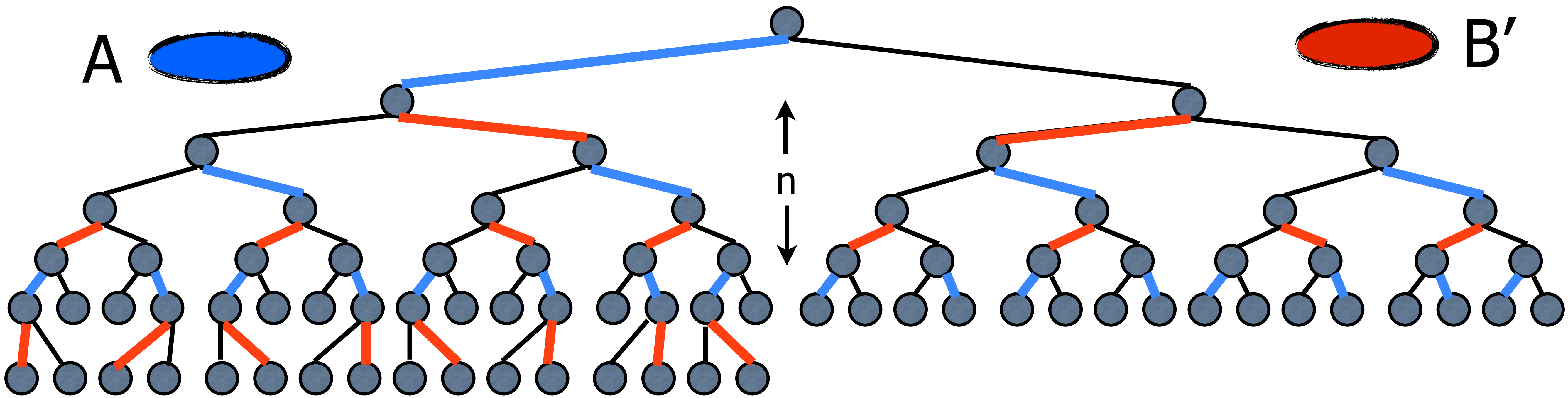


Protocol for Party I

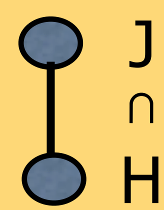
A: edges announced by Player I

B′: decoding of edges announced by Player 2

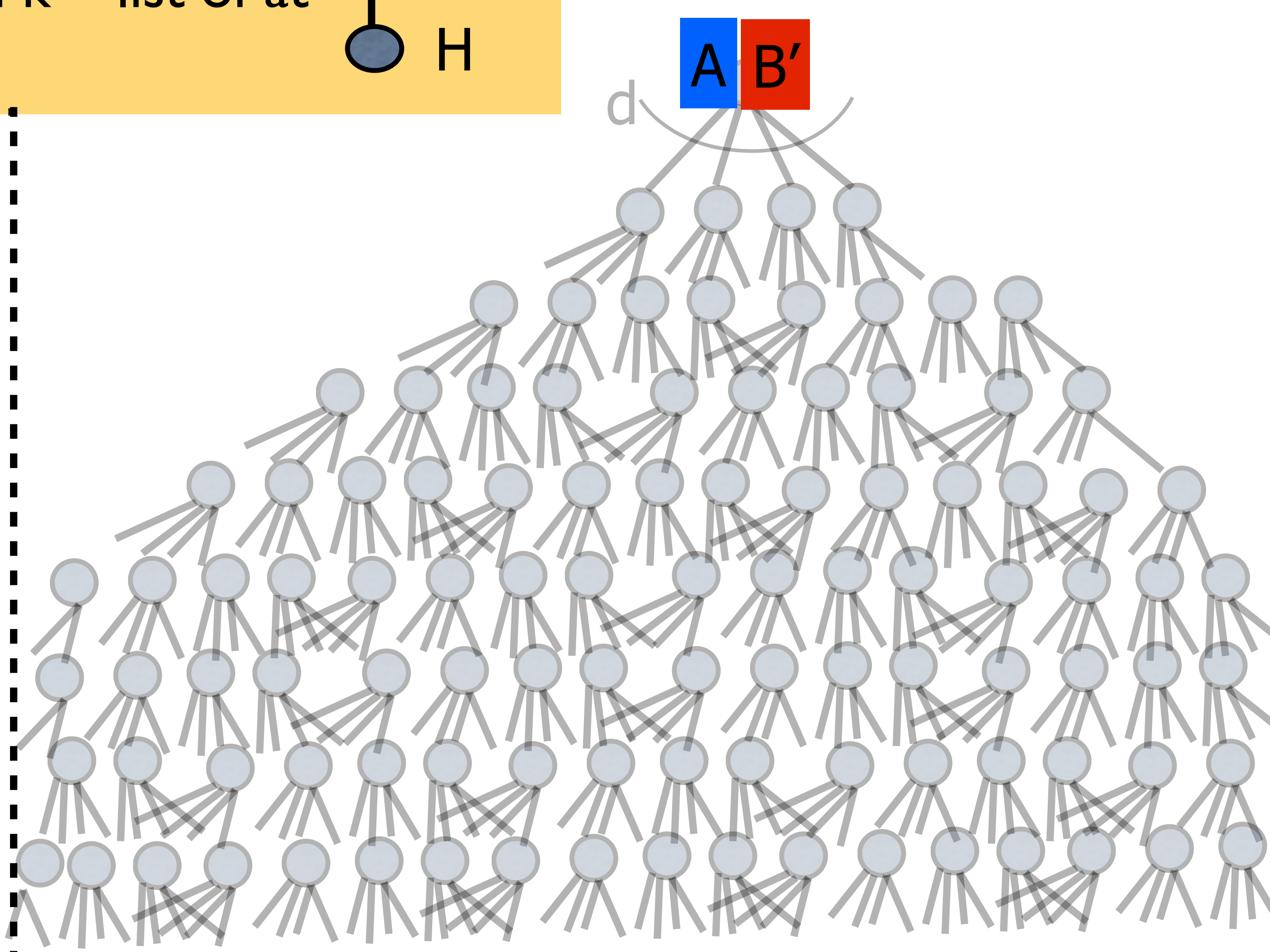
Repeat: Send edge that extends path in A∪B′, if possible.



Distance = $1-\varepsilon$, Depth $O(n/\varepsilon)$. $d = 2^n$
Vertex at depth k = list of at most k edges.



Tree Code

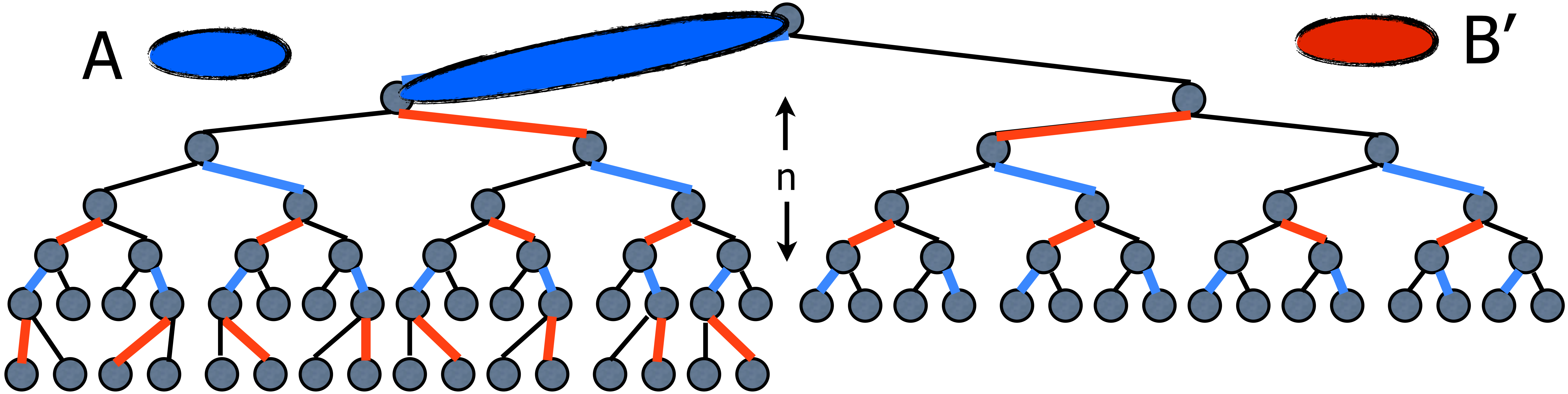


Protocol for Party I

A: edges announced by Player I

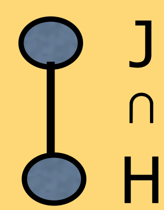
B′: decoding of edges announced by Player 2

Repeat: Send edge that extends path in A∪B′, if possible.

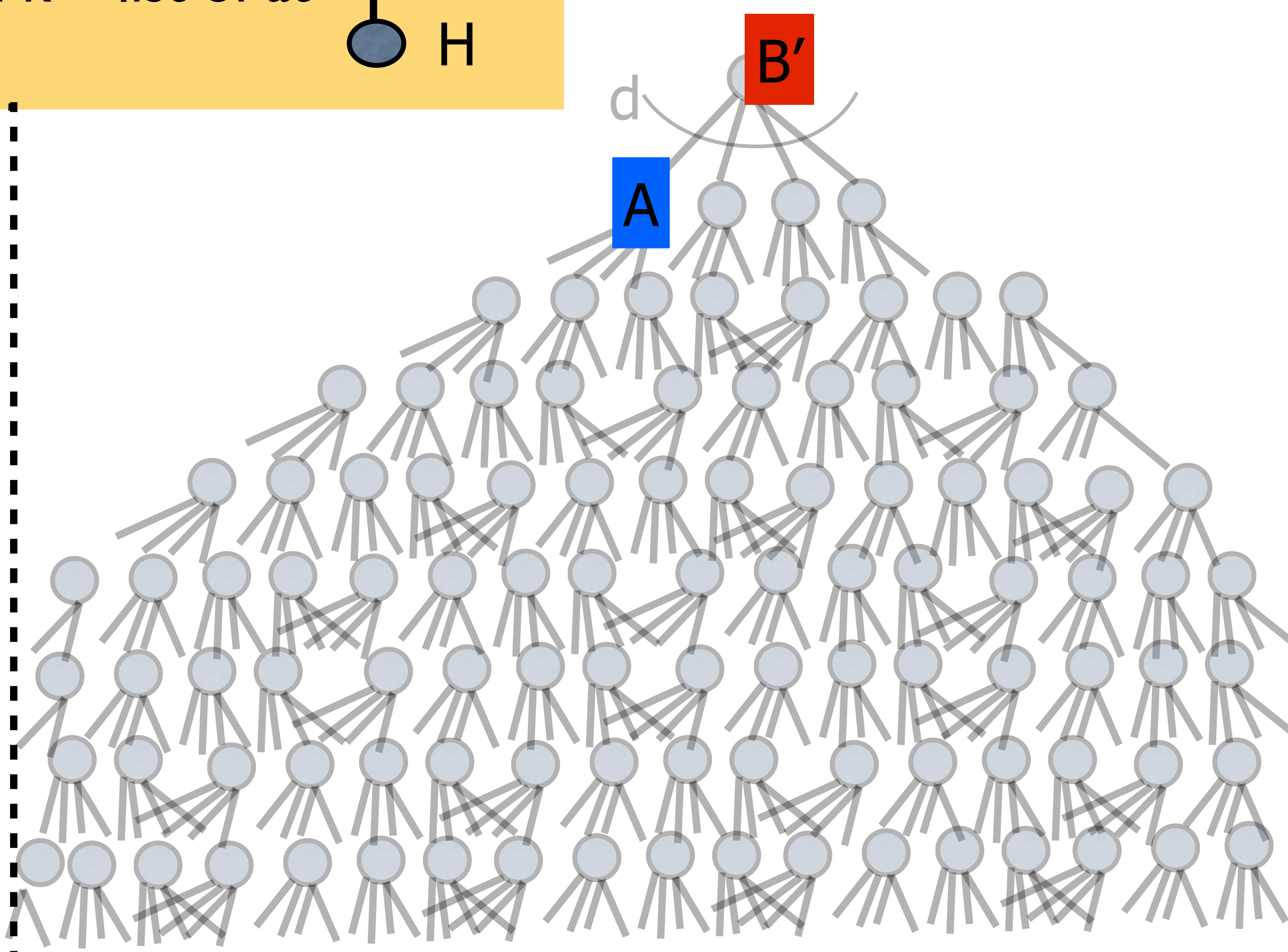


Distance = 1-ε, Depth $O(n/\varepsilon)$. $d = 2^n$

Vertex at depth k = list of at most k edges.



Tree Code

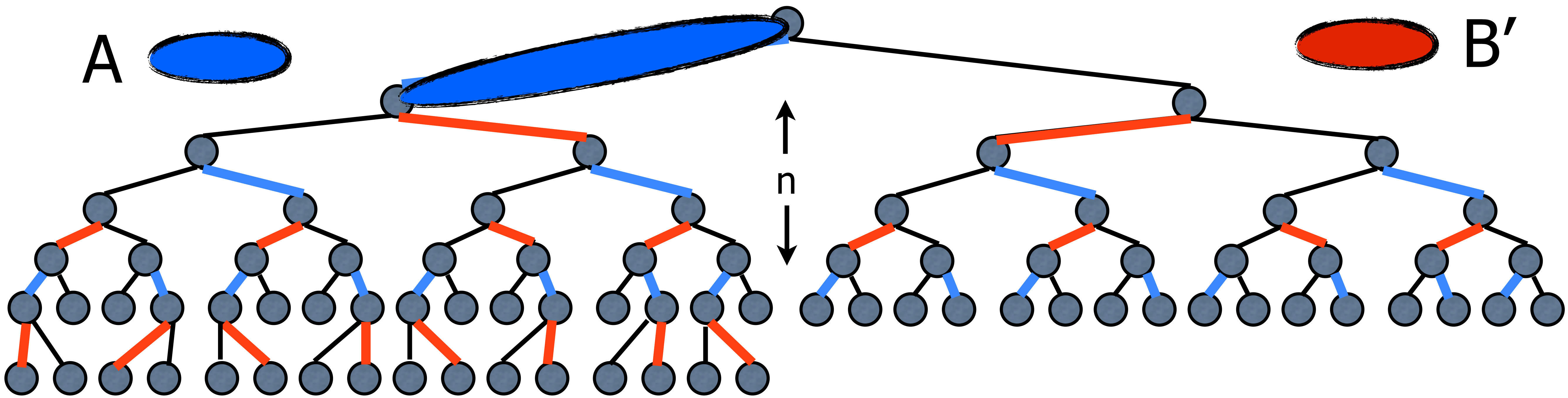


Protocol for Party I

A: edges announced by Player I

B′: decoding of edges announced by Player 2

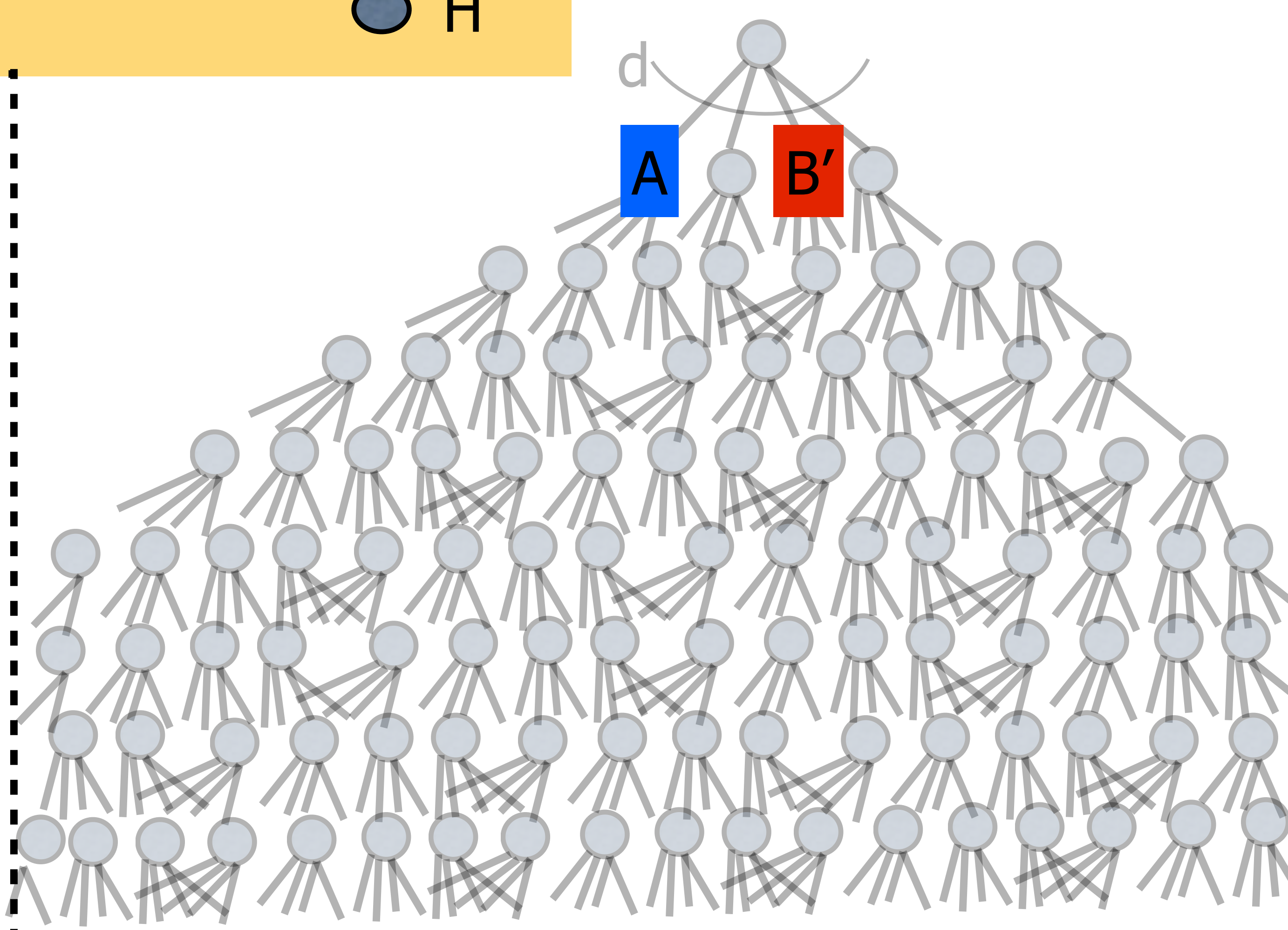
Repeat: Send edge that extends path in A∪B′, if possible.



Distance = 1-ε, Depth O(n/ε). d = 2^n
Vertex at depth k = list of at most k edges.

J
∩
H

Tree Code

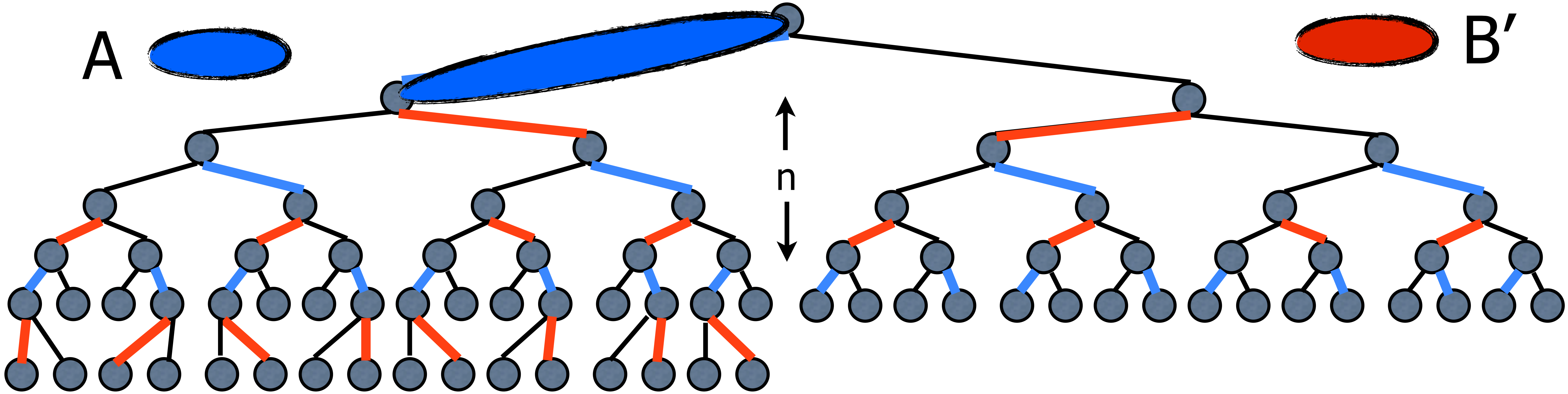


Protocol for Party 1

A: edges announced by Player 1

B′: decoding of edges announced by Player 2

Repeat: Send edge that extends path in A∪B′, if possible.

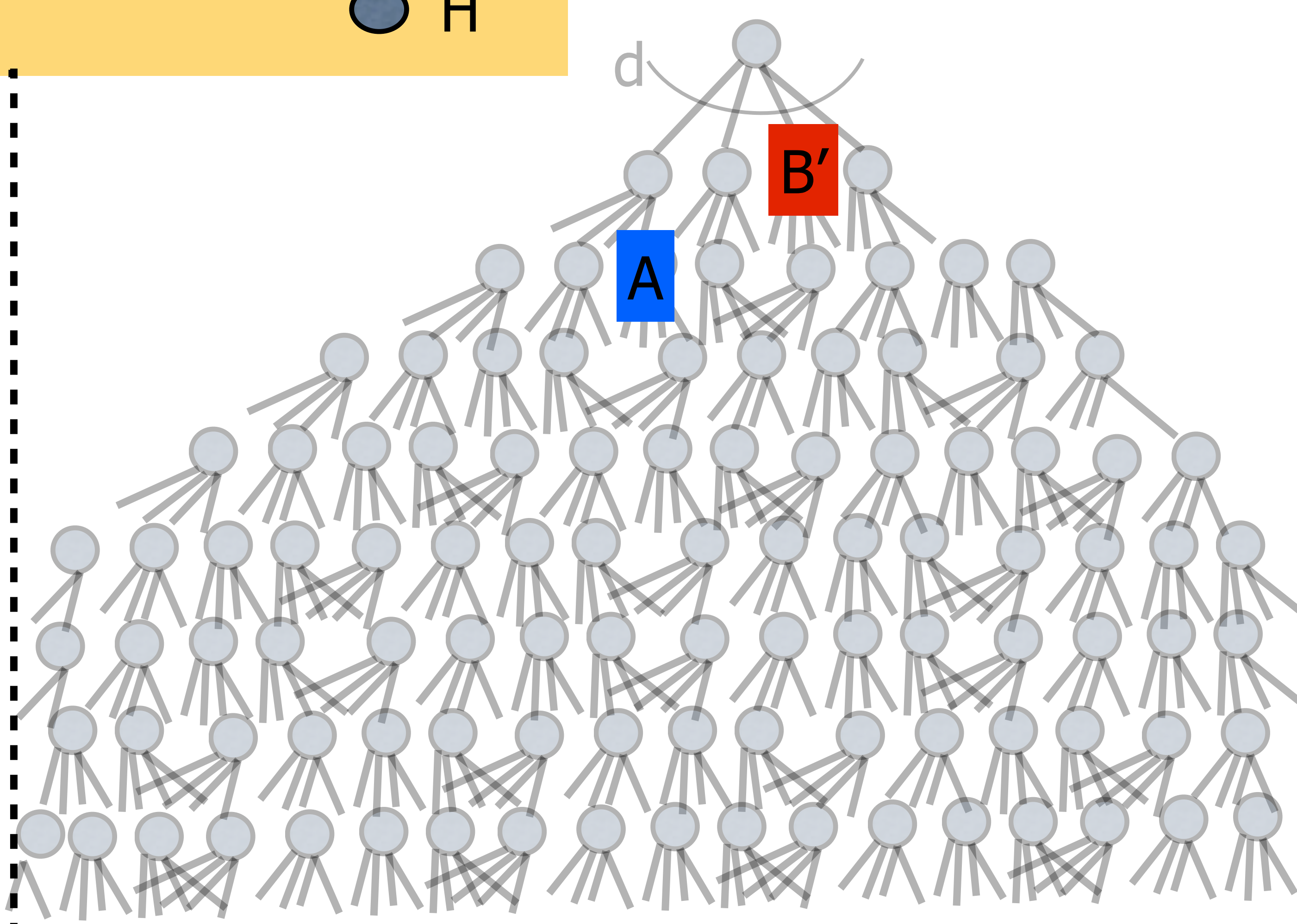


Distance = 1-ε, Depth O(n/ε). $d = 2^n$

Vertex at depth k = list of at most k edges.

J ∩ H

Tree Code

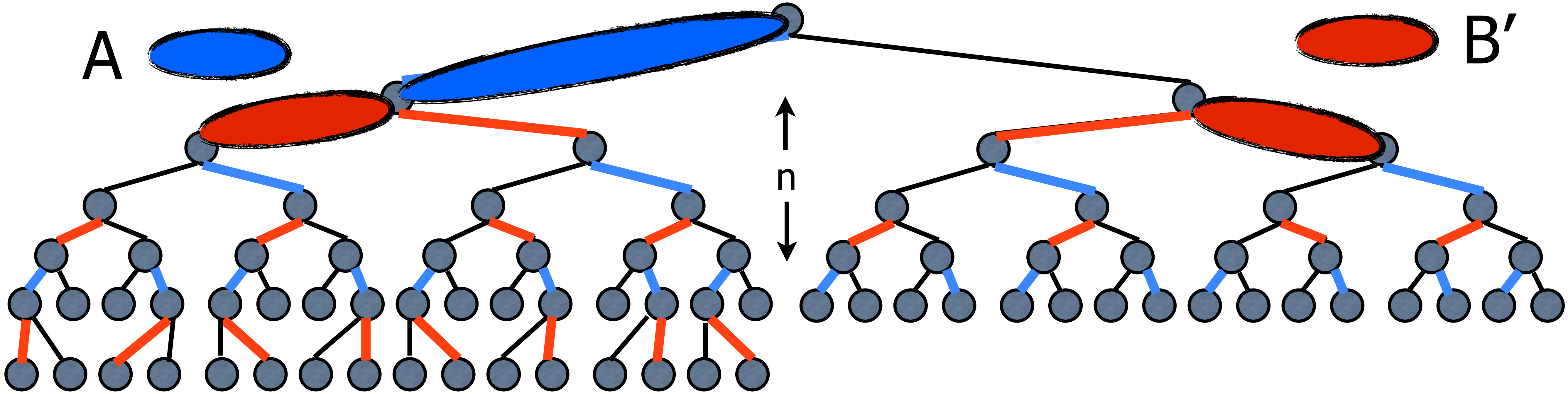


Protocol for Party 1

A: edges announced by Player 1

B′: decoding of edges announced by Player 2

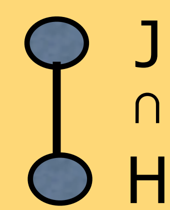
Repeat: Send edge that extends path in A∪B′, if possible.



Tree Code

Distance = 1-ε, Depth O(n/ε). $d = 2^n$

Vertex at depth k = list of at most k edges.

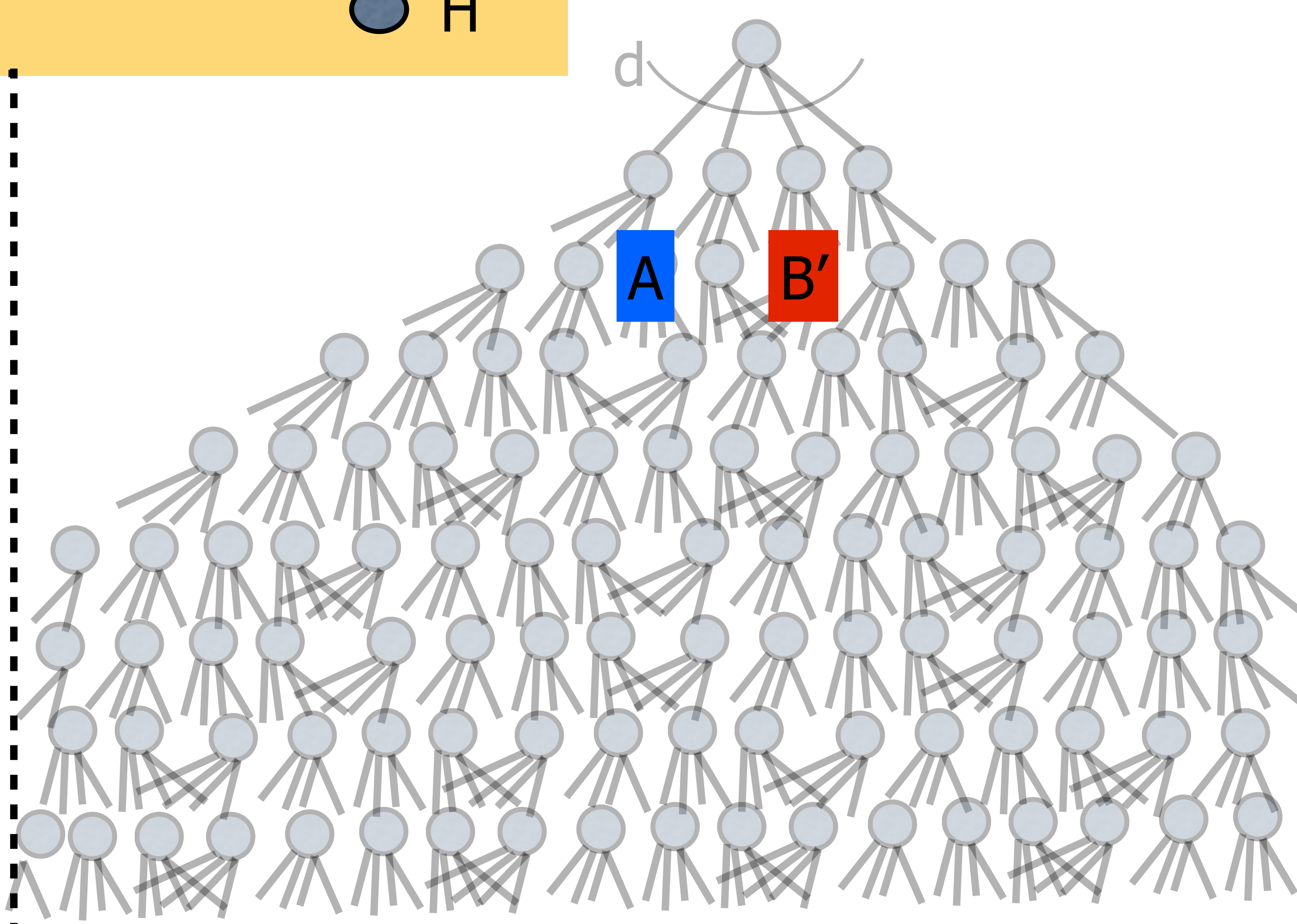


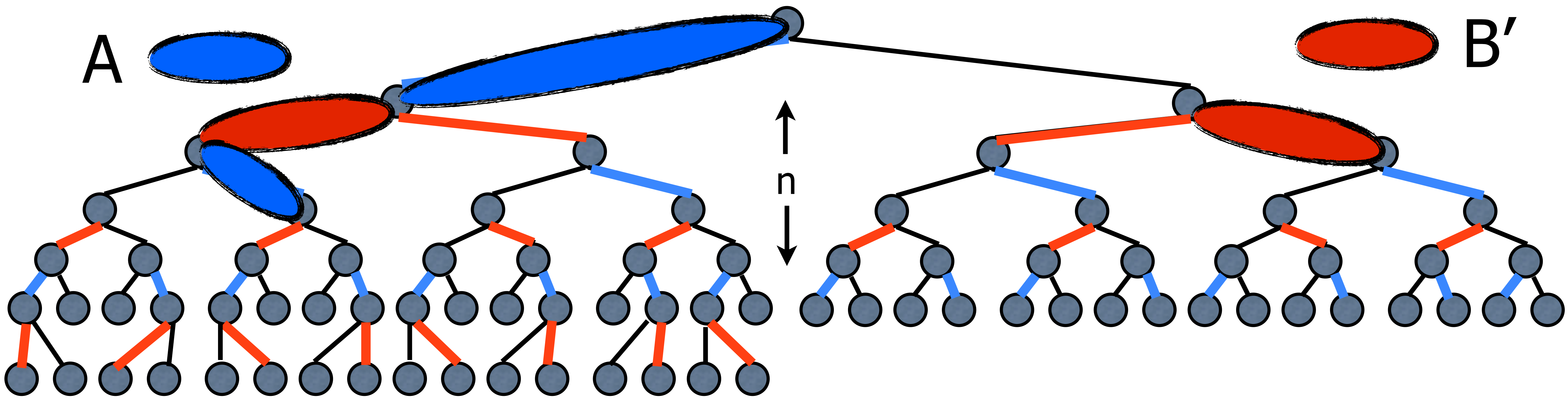
Protocol for Party I

A: edges announced by Player I

B′: decoding of edges announced by Player 2

Repeat: Send edge that extends path in A∪B′, if possible.

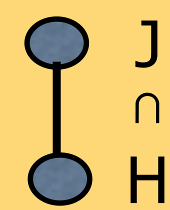




Tree Code

Distance = 1-ε, Depth O(n/ε). $d = 2^n$

Vertex at depth k = list of at most k edges.

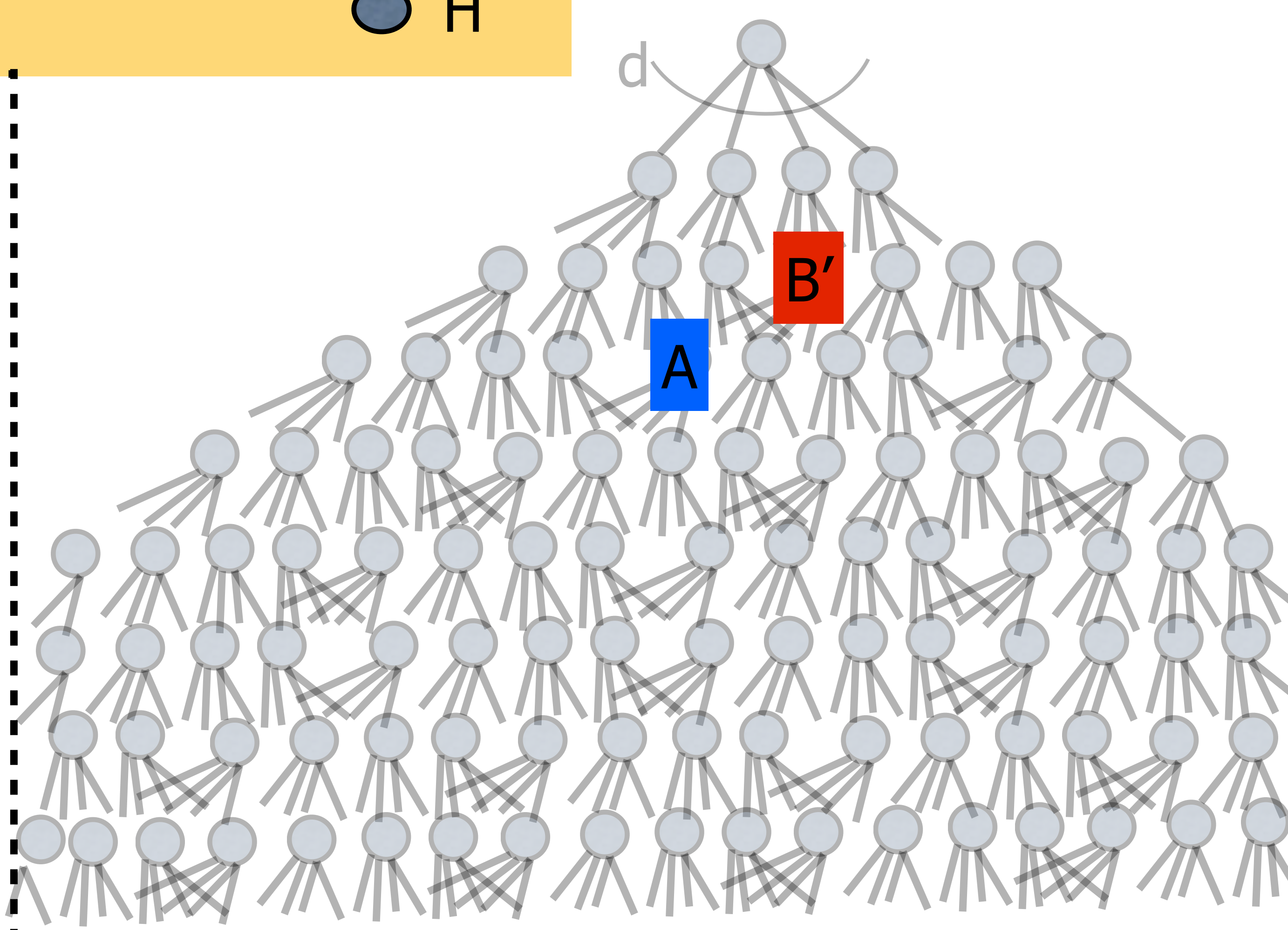


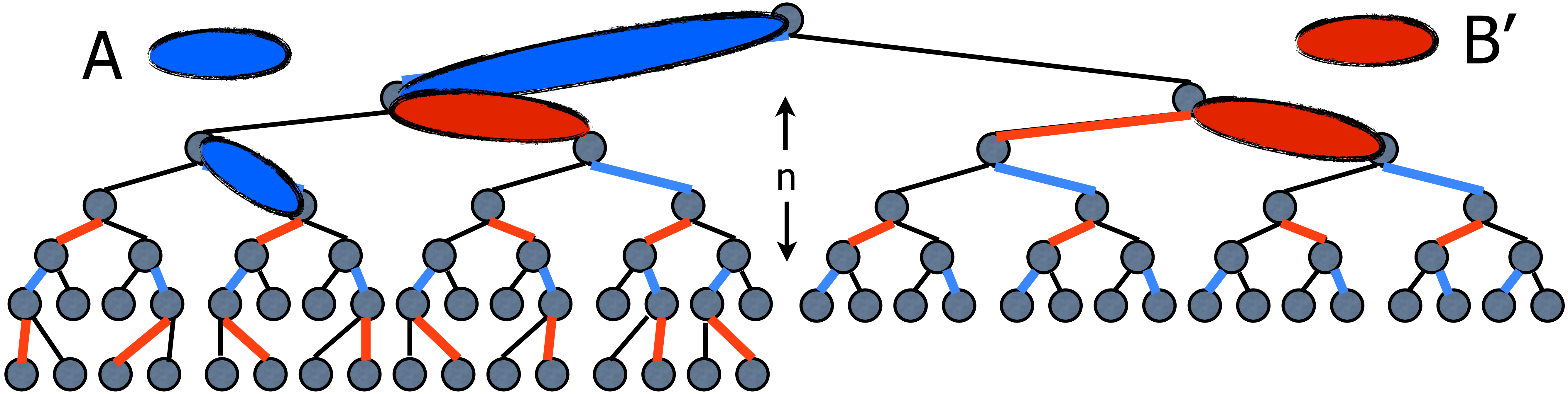
Protocol for Party I

A: edges announced by Player I

B′: decoding of edges announced by Player 2

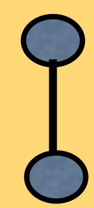
Repeat: Send edge that extends path in A∪B′, if possible.



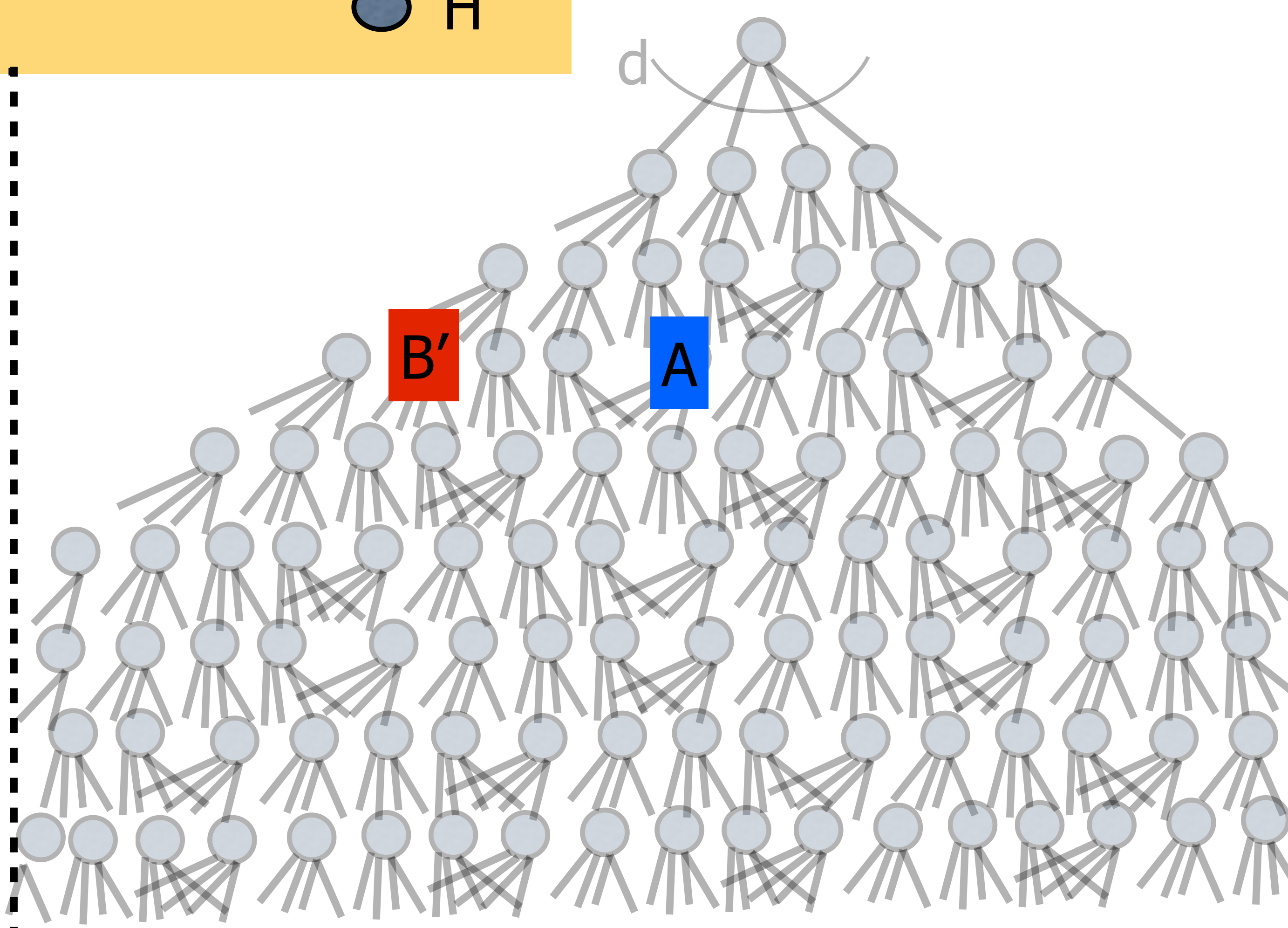


Distance = 1-ε, Depth O(n/ε). d = 2^n

Vertex at depth k = list of at most k edges.



Tree Code

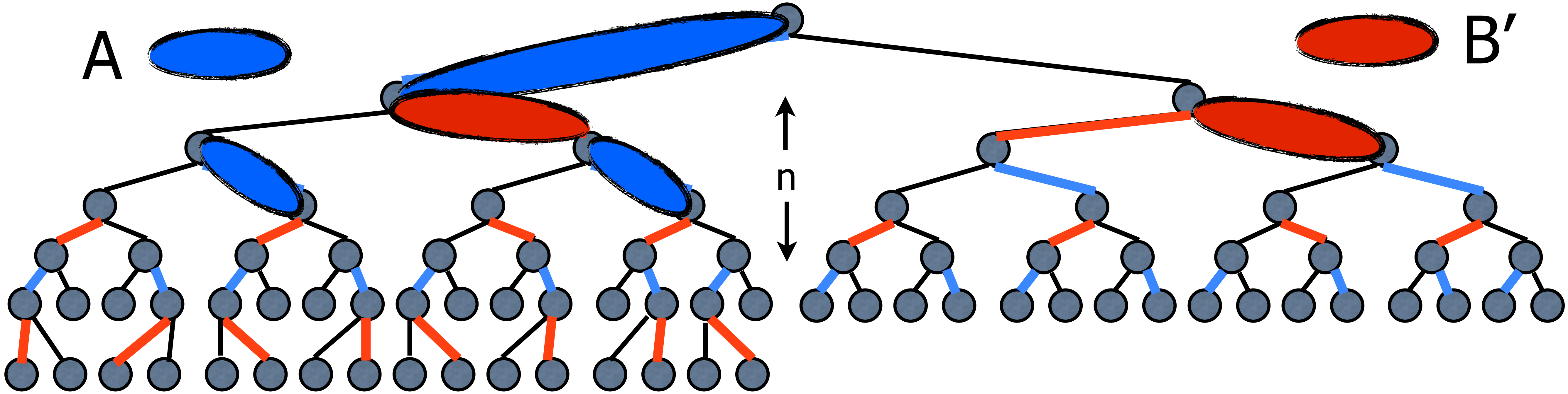


Protocol for Party I

A: edges announced by Player I

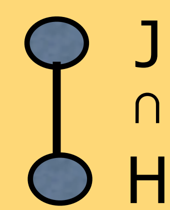
B′: decoding of edges announced by Player 2

Repeat: Send edge that extends path in A∪B′, if possible.



Tree Code

Distance = 1-ε, Depth O(n/ε). d = 2^n
Vertex at depth k = list of at most k edges.

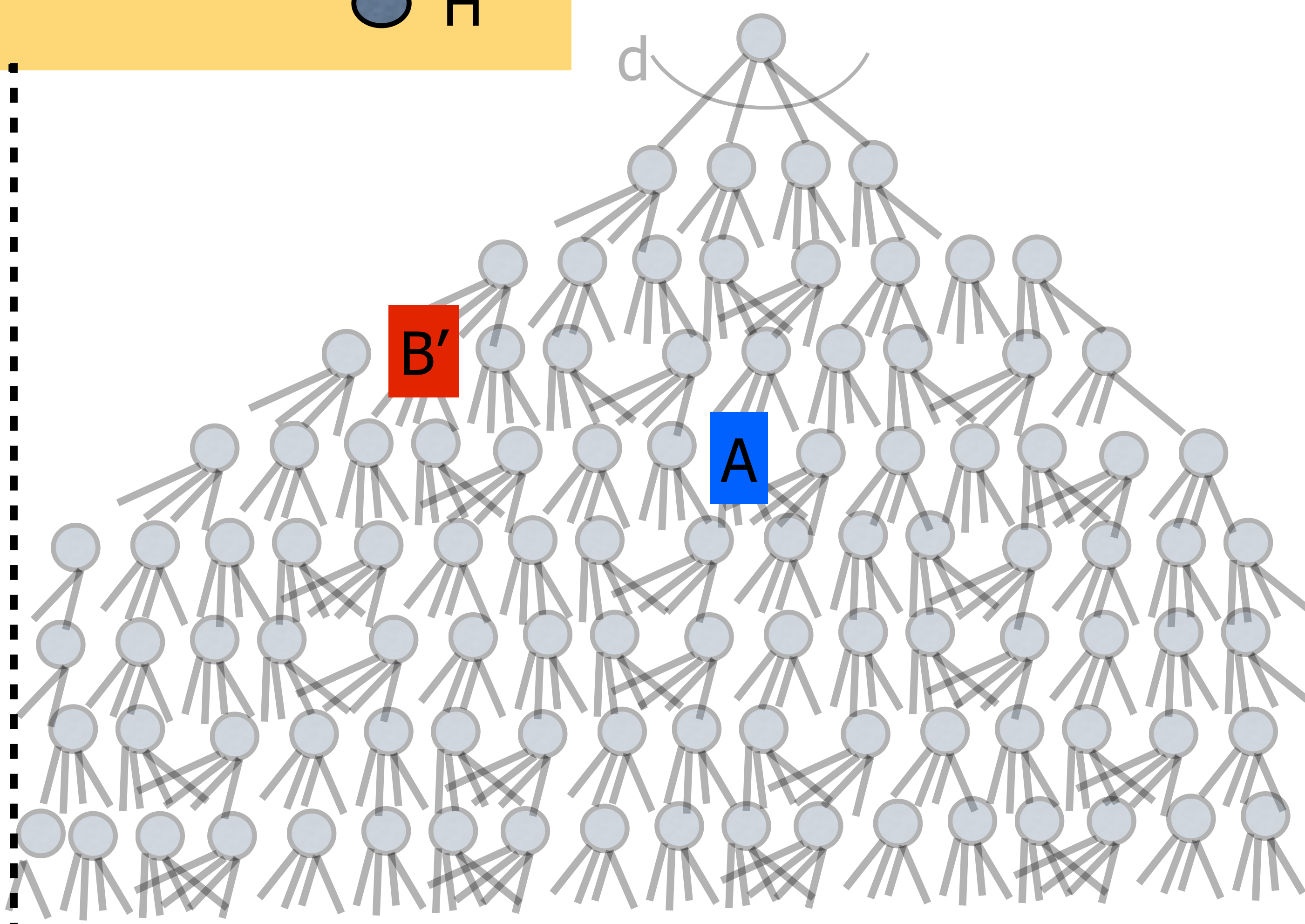


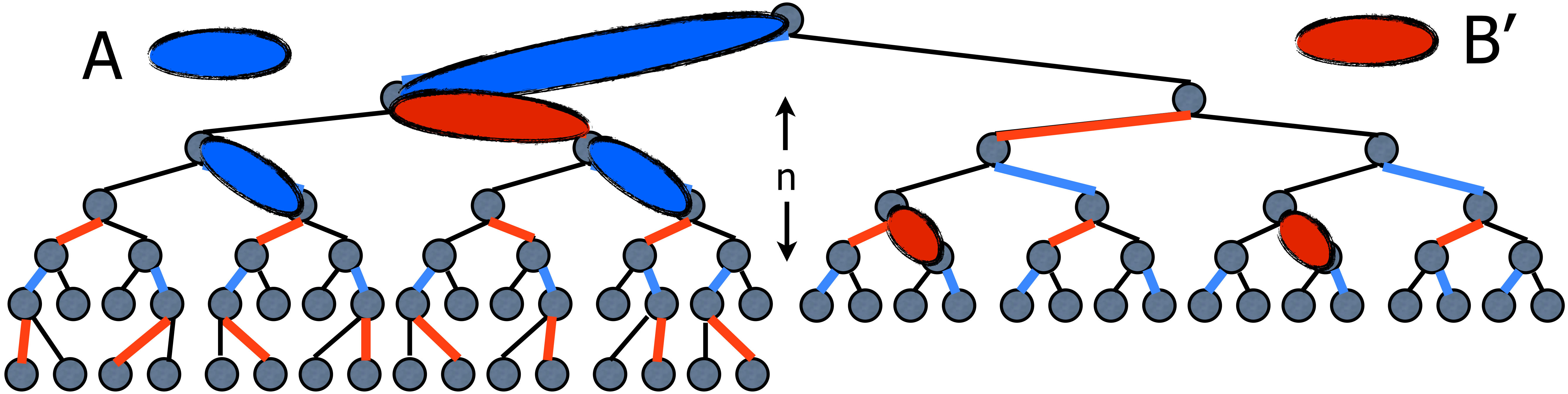
Protocol for Party 1

A: edges announced by Player 1

B′: decoding of edges announced by Player 2

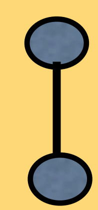
Repeat: Send edge that extends path in A∪B′, if possible.



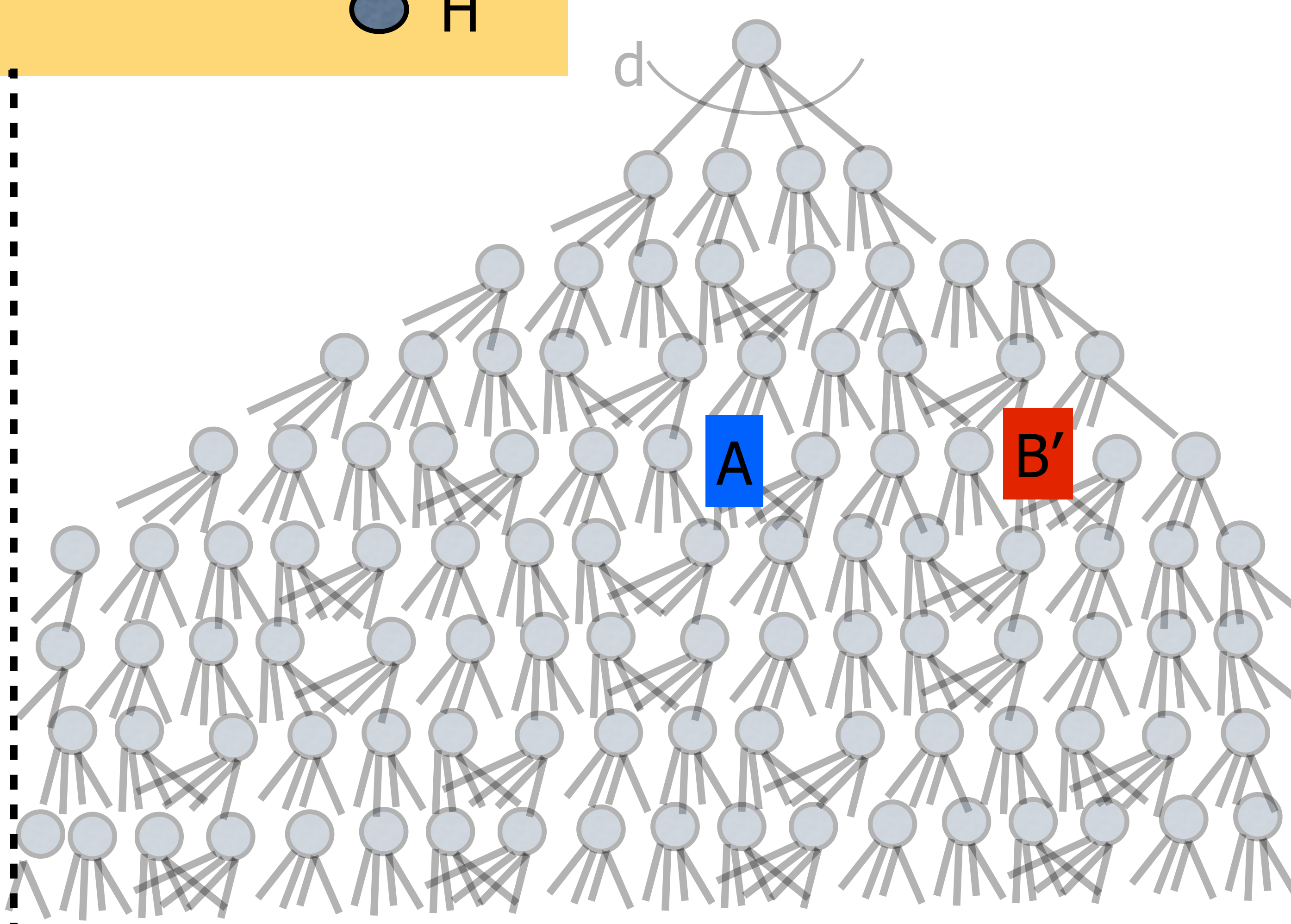


Distance = 1-ε, Depth O(n/ε). d = 2^n

Vertex at depth k = list of at most k edges.



Tree Code

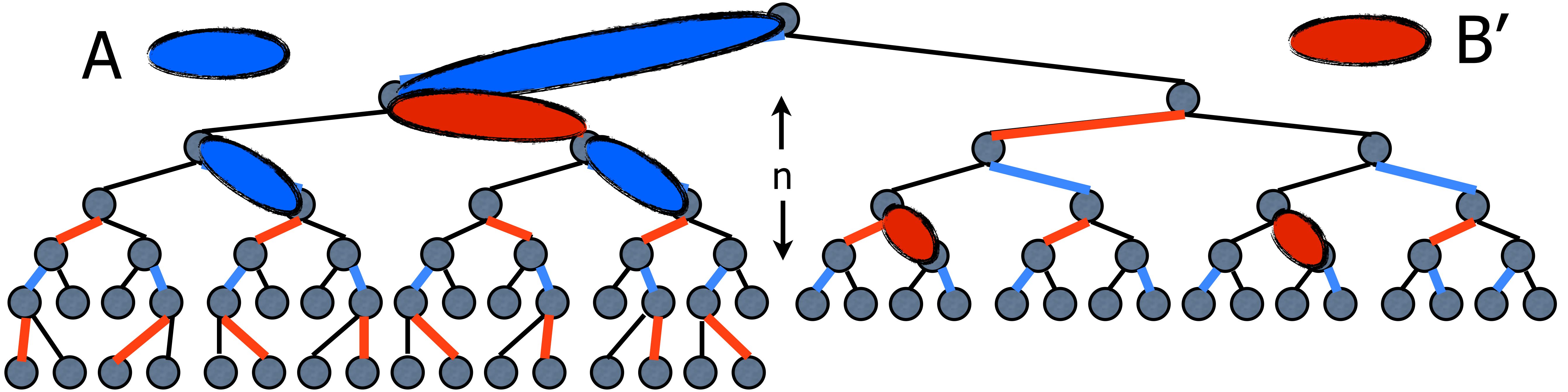


Protocol for Party 1

A: edges announced by Player 1

B′: decoding of edges announced by Player 2

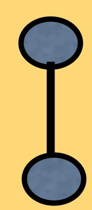
Repeat: Send edge that extends path in A∪B′, if possible.



Tree Code

Distance = 1-ε, Depth O(n/ε). $d = 2^n$

Vertex at depth k = list of at most k edges.

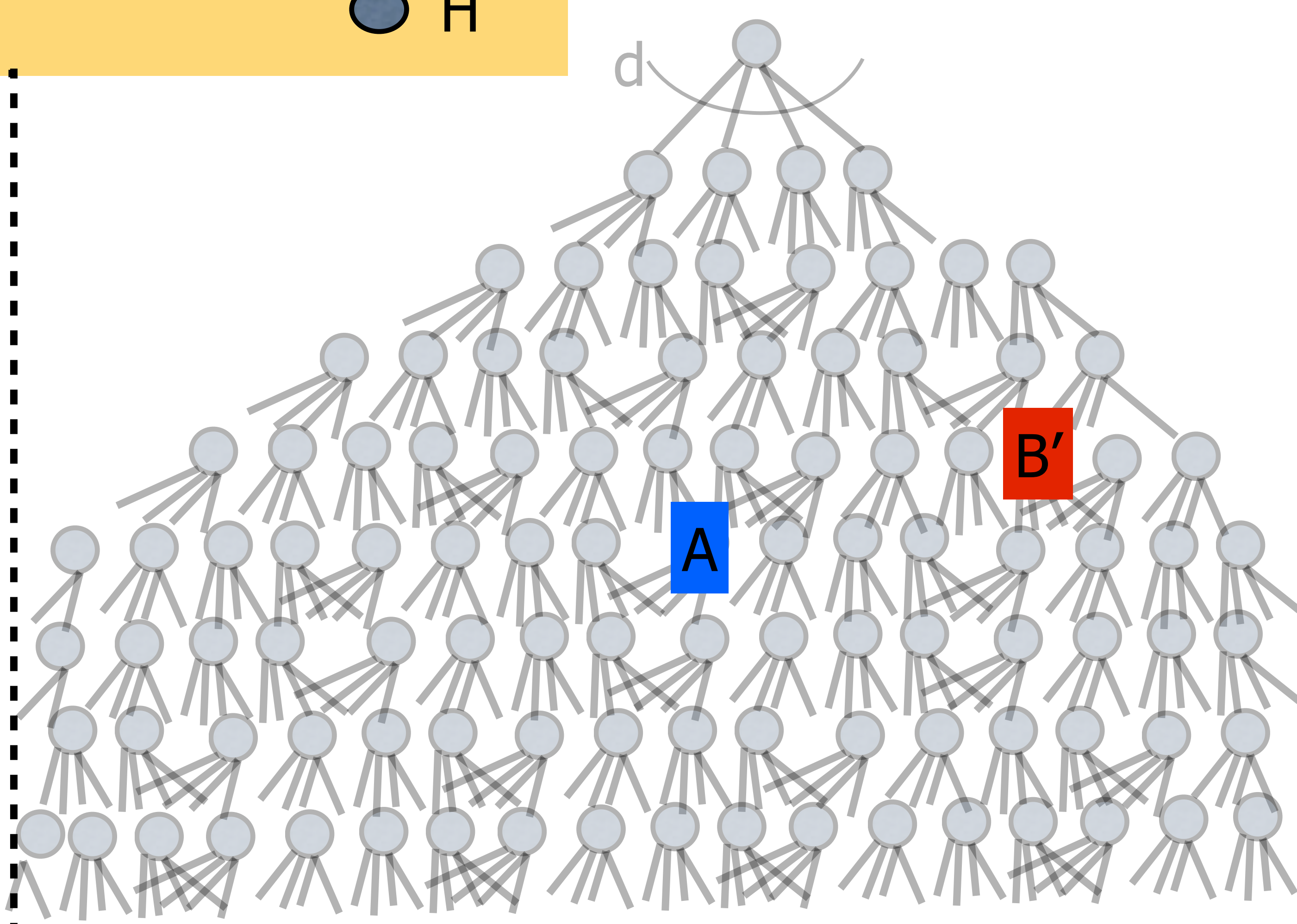


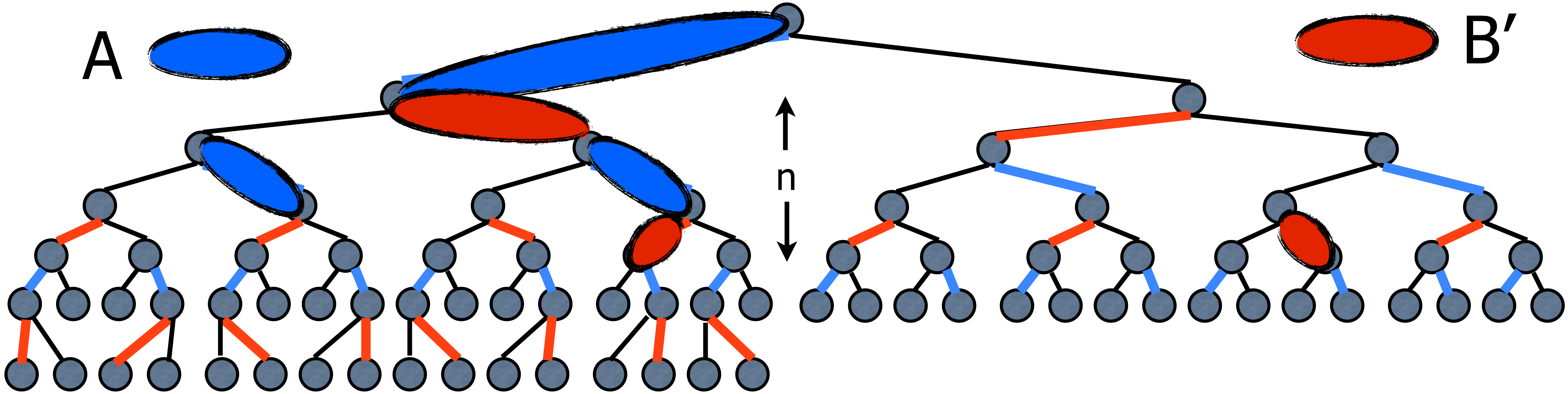
Protocol for Party 1

A: edges announced by Player 1

B′: decoding of edges announced by Player 2

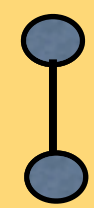
Repeat: Send edge that extends path in A∪B′, if possible.



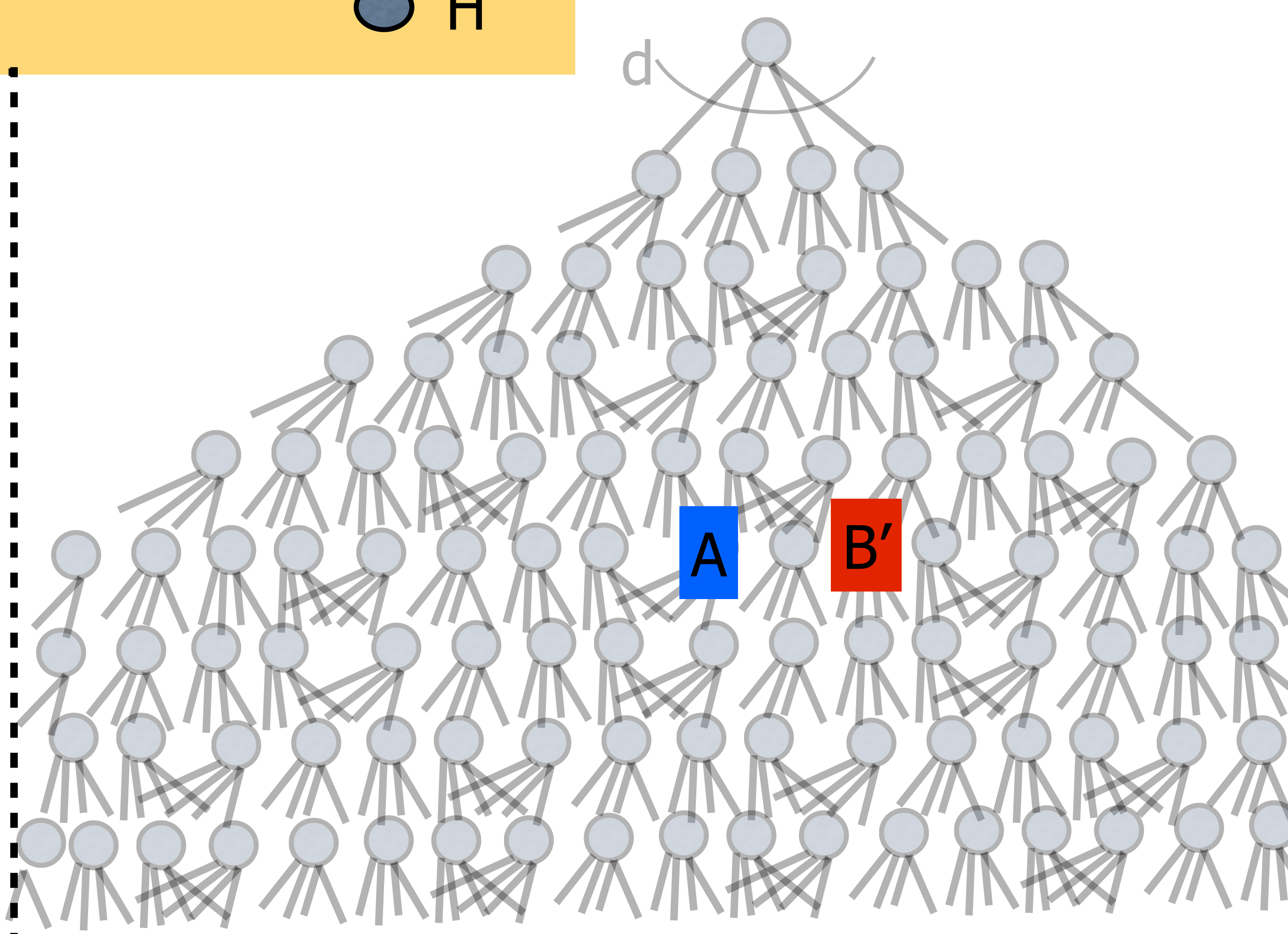


Distance = 1-ε, Depth O(n/ε). d = 2^n

Vertex at depth k = list of at most k edges.



Tree Code

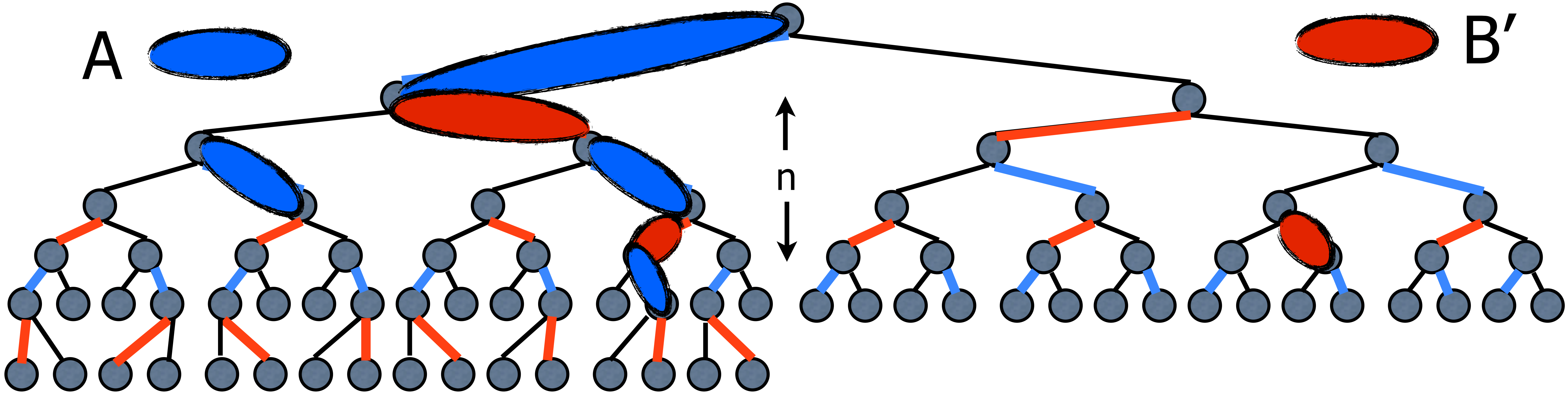


Protocol for Party I

A: edges announced by Player I

B′: decoding of edges announced by Player 2

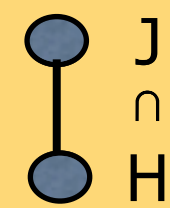
Repeat: Send edge that extends path in A∪B′, if possible.



Tree Code

Distance = 1-ε, Depth $O(n/\varepsilon)$. $d = 2^n$

Vertex at depth k = list of at most k edges.

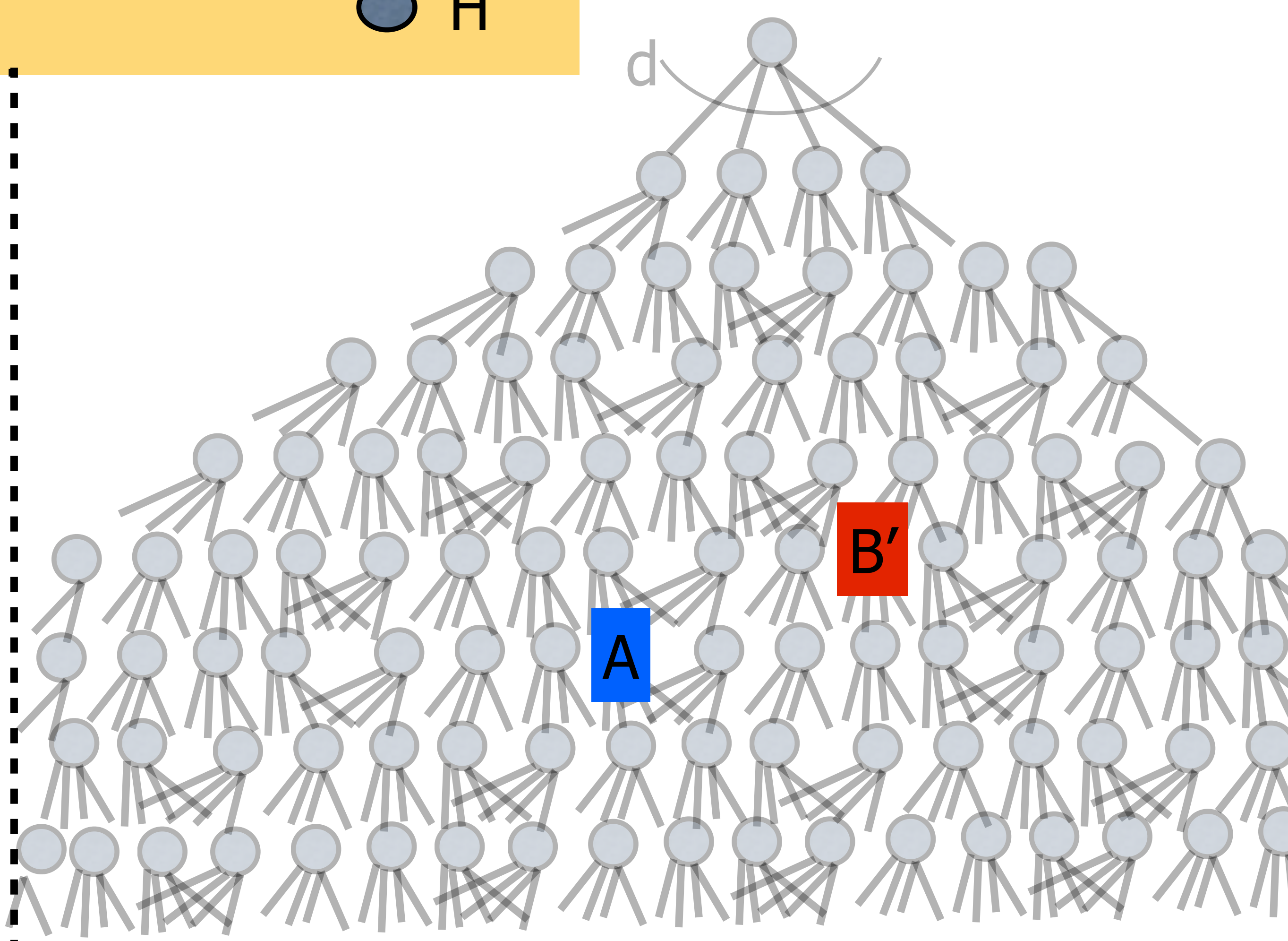


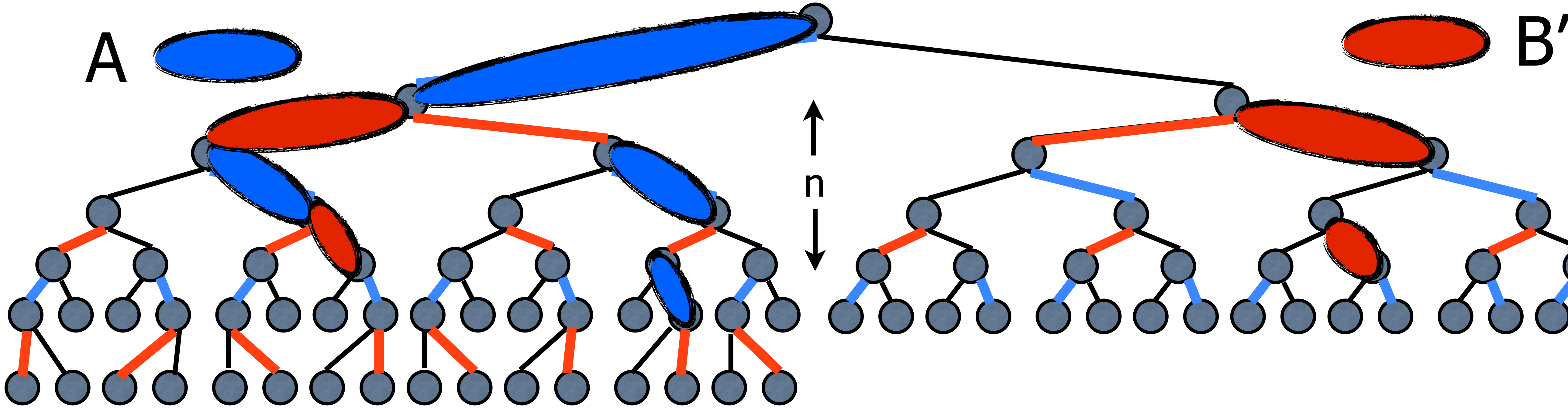
Protocol for Party 1

A: edges announced by Player 1

B′: decoding of edges announced by Player 2

Repeat: Send edge that extends path in A∪B′, if possible.





Tree Code

Distance = 1-ε, Depth $O(n/\varepsilon)$. $d = 2^n$
Vertex at depth k = list of at most k edges.

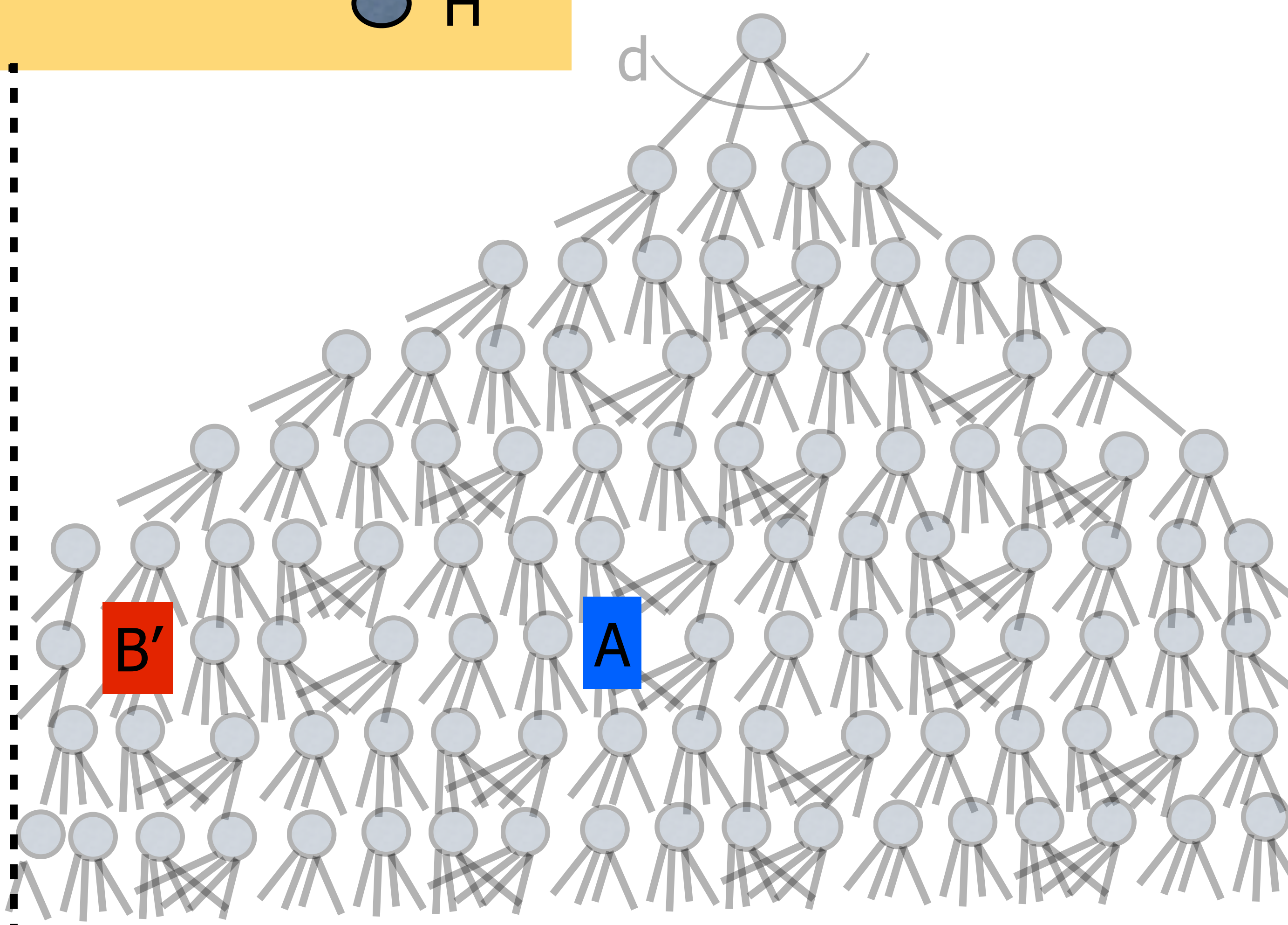
J ∩ H

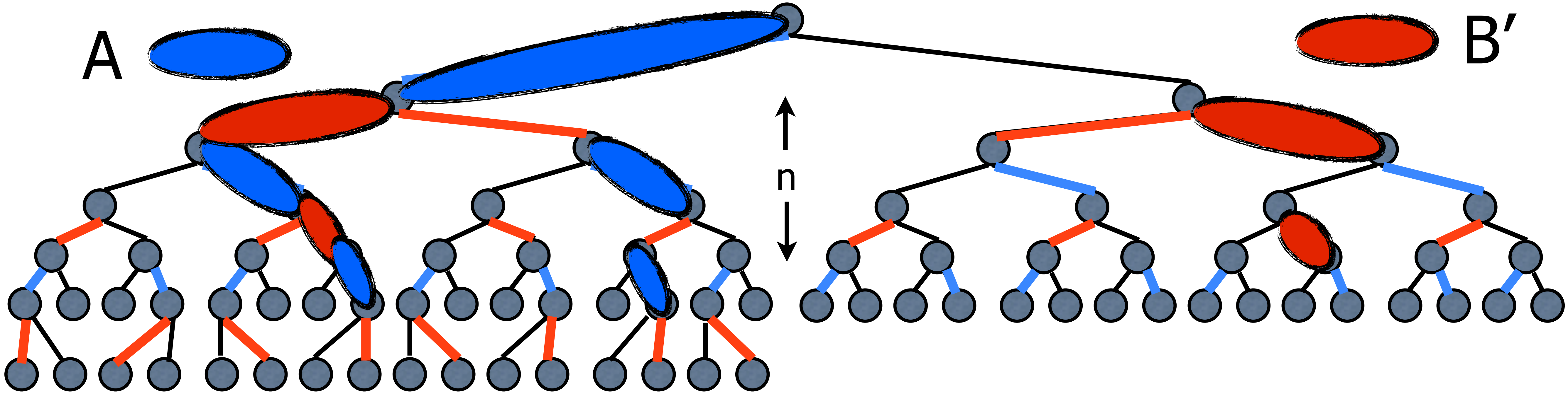
Protocol for Party I

A: edges announced by Player I

B′: decoding of edges announced by Player 2

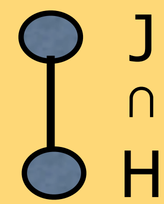
Repeat: Send edge that extends path in A∪B′, if possible.





Tree Code

Distance = 1-ε, Depth O(n/ε). $d = 2^n$
Vertex at depth k = list of at most k edges.

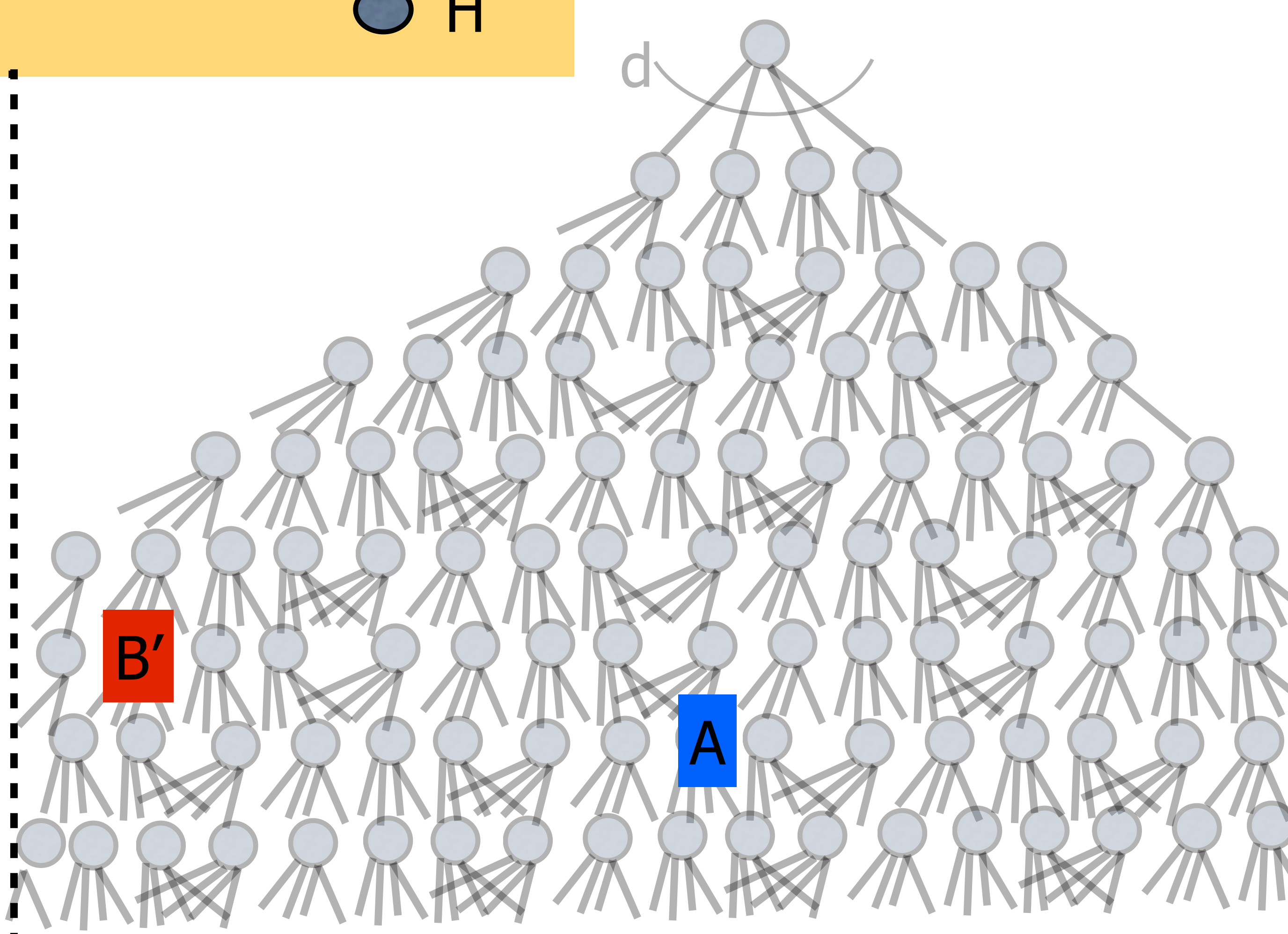


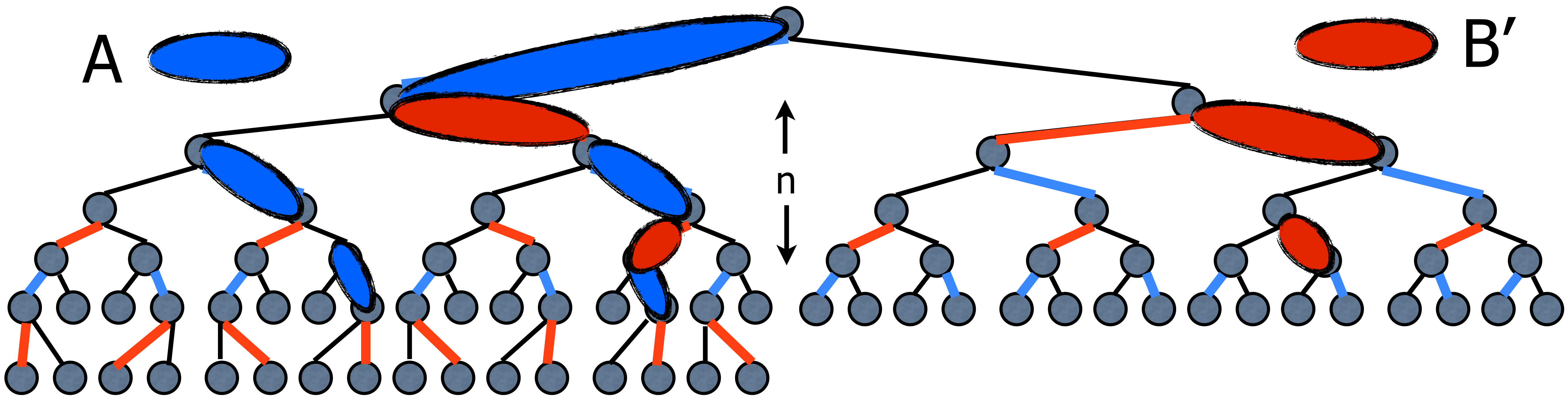
Protocol for Party I

A: edges announced by Player I

B′: decoding of edges announced by Player 2

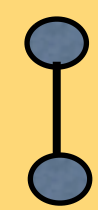
Repeat: Send edge that extends path in A∪B′, if possible.





Distance = 1-ε, Depth O(n/ε). d = 2^n

Vertex at depth k = list of at most k edges.



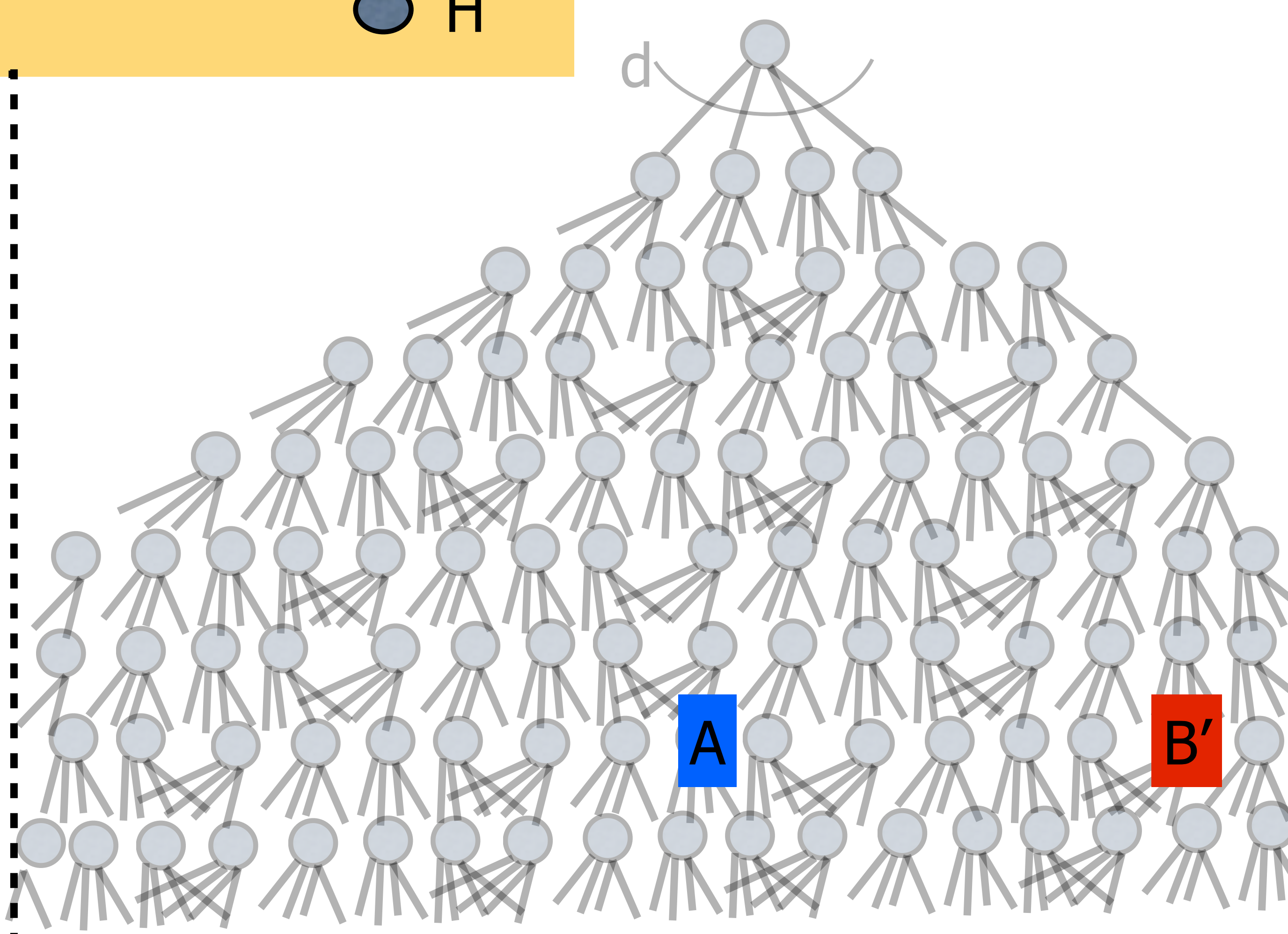
Tree Code

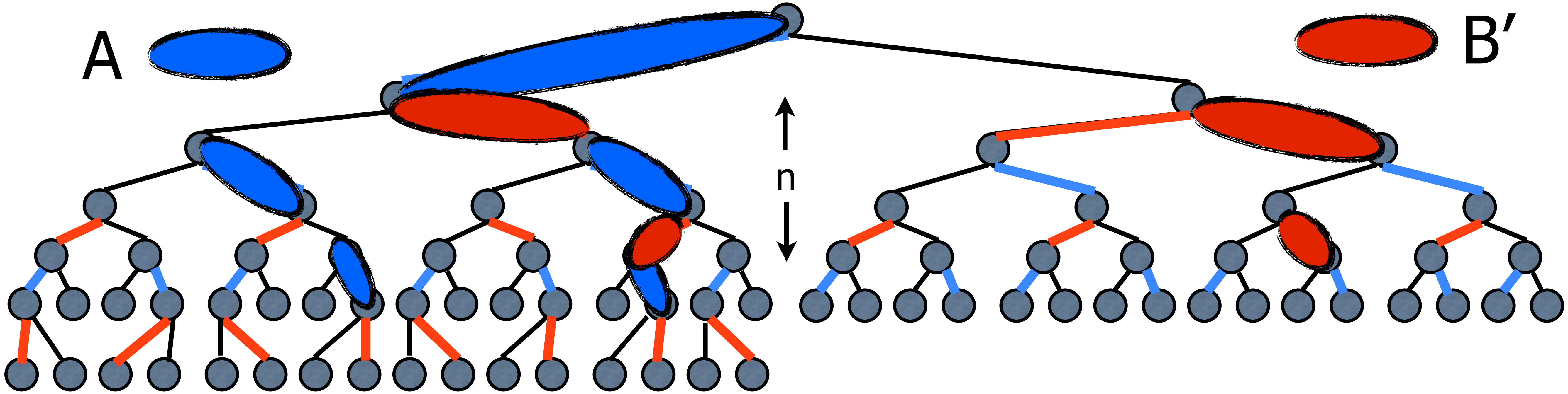
Protocol for Party 1

A: edges announced by Player 1

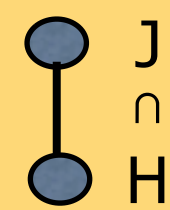
B′: decoding of edges announced by Player 2

Repeat: Send edge that extends path in A∪B′, if possible.

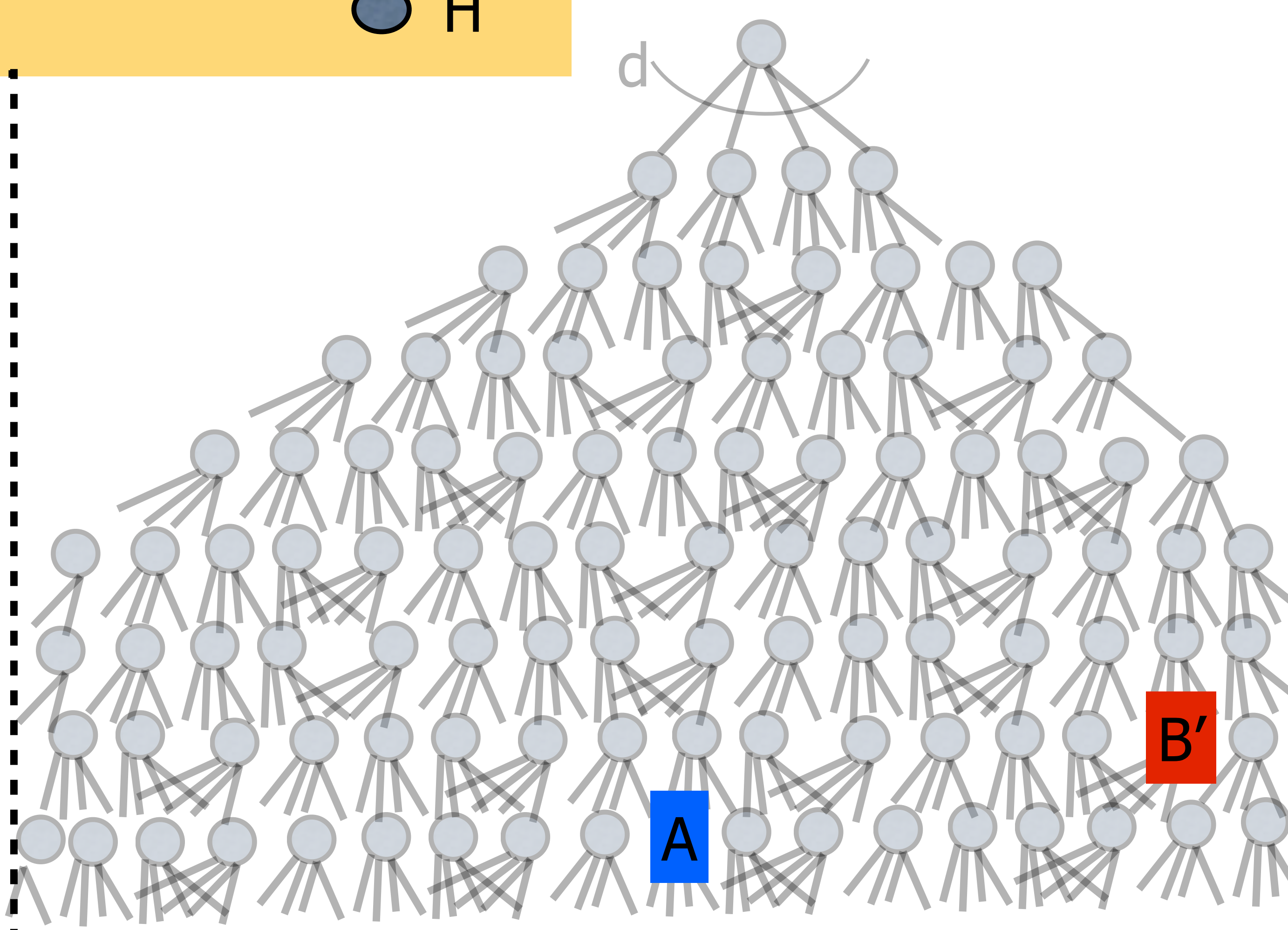




Distance = 1-ε, Depth O(n/ε). d = 2^n
Vertex at depth k = list of at most k edges.



Tree Code

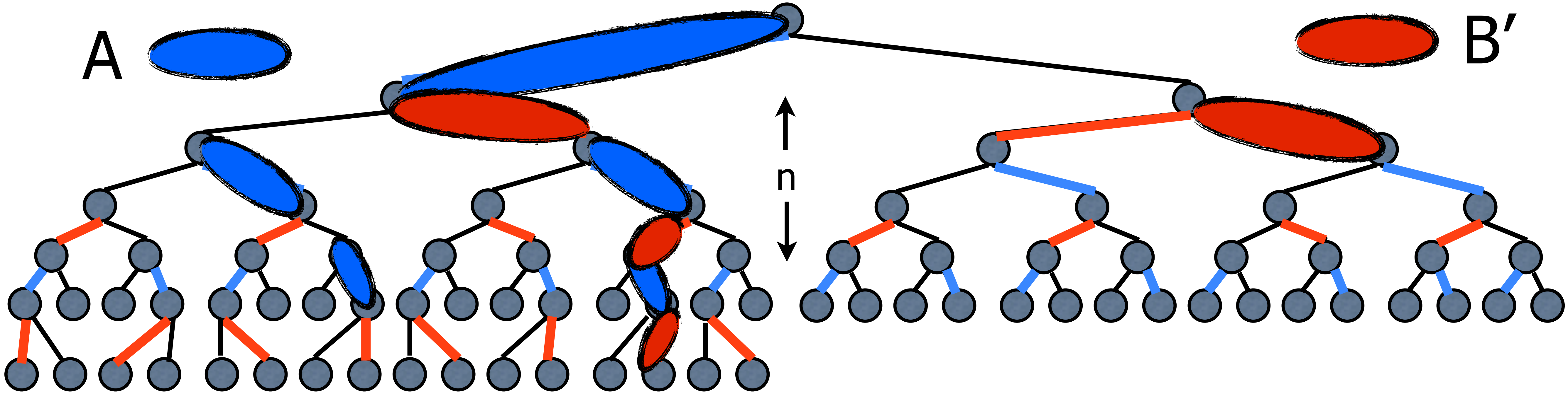


Protocol for Party 1

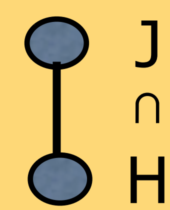
A: edges announced by Player 1

B′: decoding of edges announced by Player 2

Repeat: Send edge that extends path in A∪B′, if possible.



Distance = 1-ε, Depth O(n/ε). $d = 2^n$
Vertex at depth k = list of at most k edges.



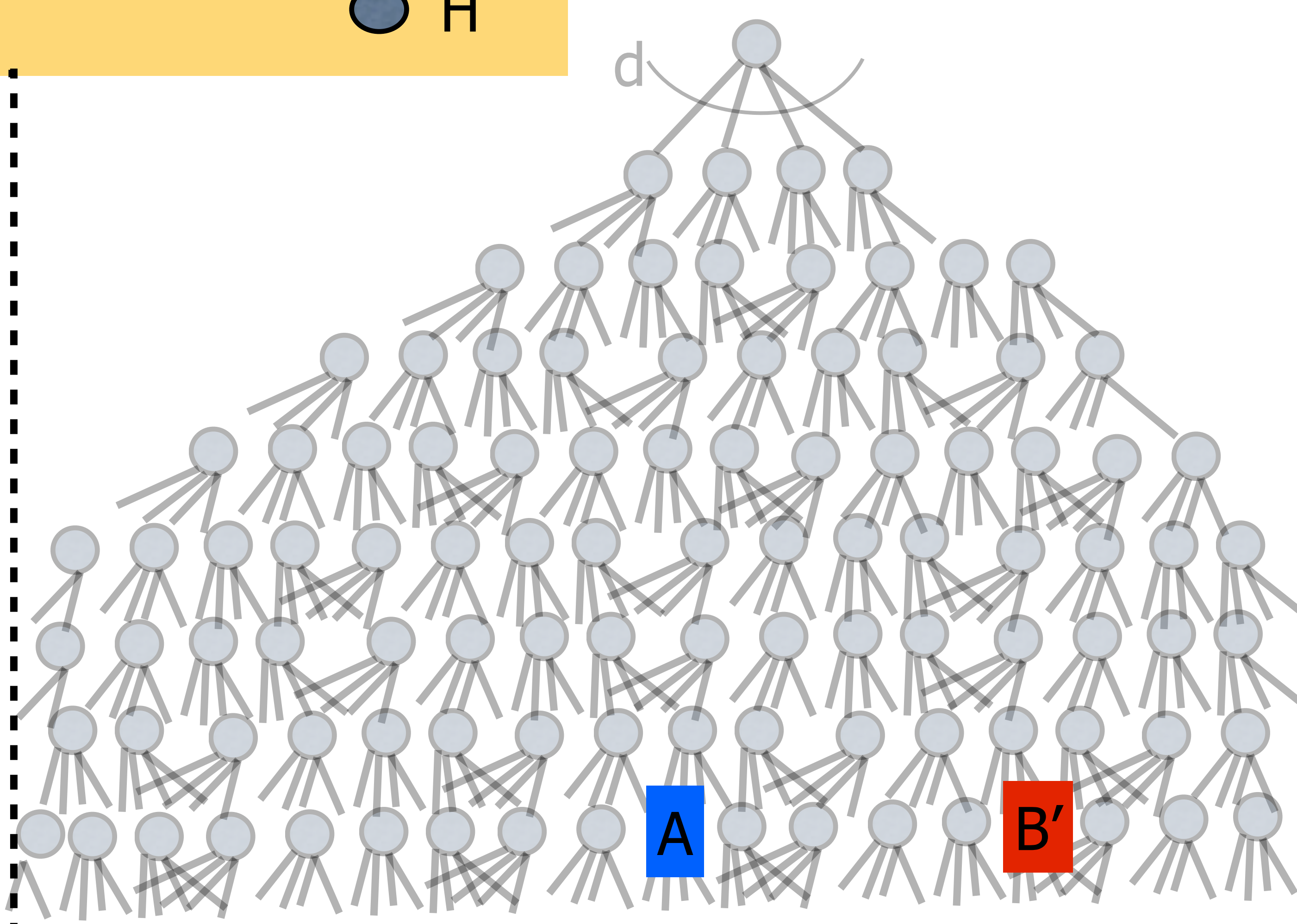
Tree Code

Protocol for Party 1

A: edges announced by Player 1

B′: decoding of edges announced by Player 2

Repeat: Send edge that extends path in A∪B′, if possible.



Open Problems

- Explicitly encodable and decodable Tree Codes? (This would make everything explicit). poly(n) alphabet possible [EKS]
- What if error rate is bounded per party, not globally?


Questions?