WIT-Greedy: Hardware System Design of Weighted ITerative Greedy Decoder for Surface Code

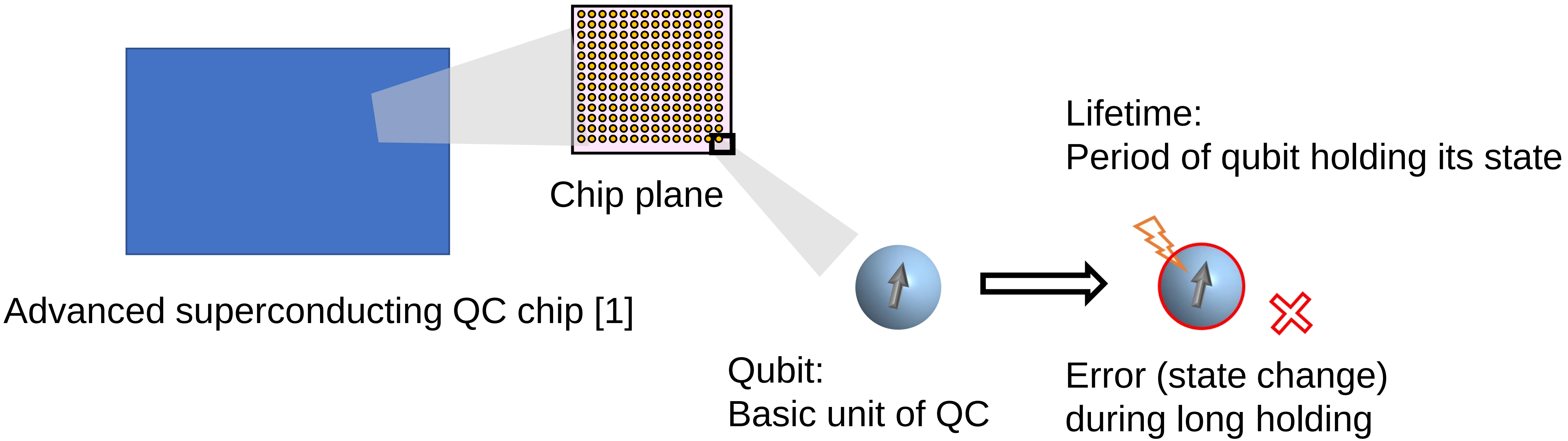
Wang LIAO[1], Yasunari Suzuki[2, 3], Teruo Tanimoto[3, 4], Yosuke Ueno[1], Yuuki Tokunaga[2]

1. The University of Tokyo
2. NTT Computer and Data Science Laboratories
3. JST PRESTO
4. Kyushu University

2023/01/17 @ ASPDAC

Quantum computing (QC) era is coming, but lifetime of qubits limits its application

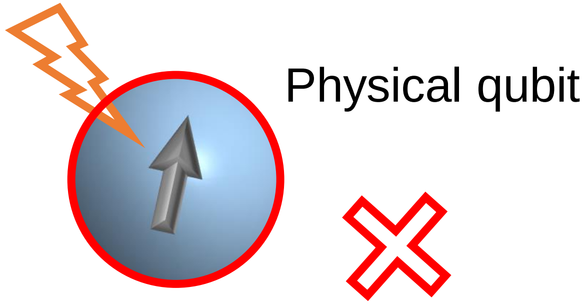
- QC enables exponential speed-up for several important information processing
- Error due to short lifetime of qubit is the current bottleneck
  - Only 100 μs, but we need…

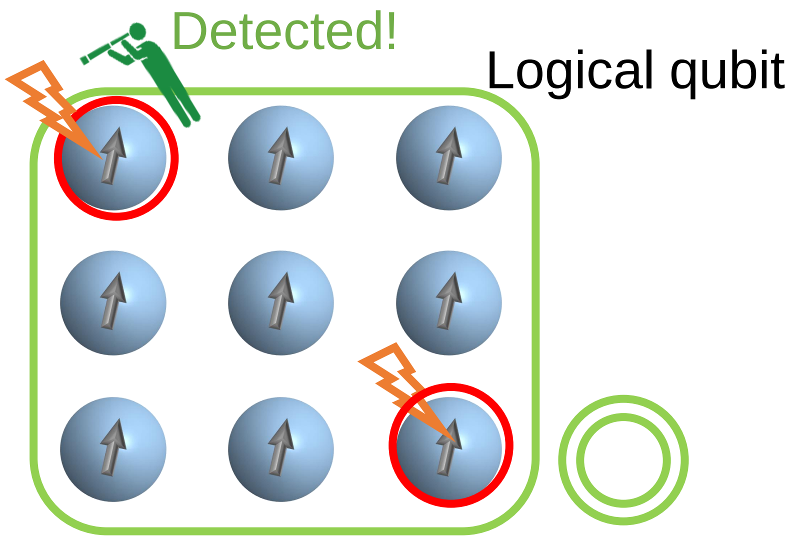


[1] Frank Arute, et al., Quantum supremacy using a programmable superconducting processor, Nature 574, 505–510 (2019).

QEC is necessary due to high error rate

- Error rate due to short lifetime of physical qubit is large
- Quantum error correction(QEC) reduces error rate
  - Physical qubits are encoded to logical qubits
  - Error rate of logical qubit can be reduced to any small value

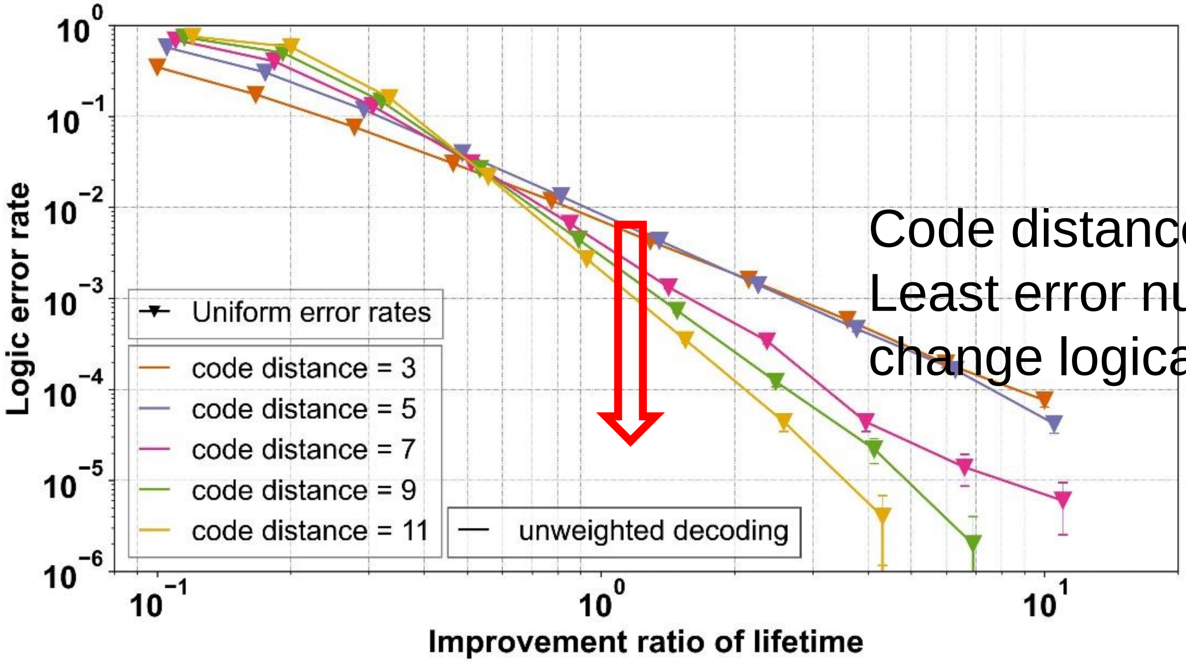




Logical error rate is reduced w/
the same single bit error rate

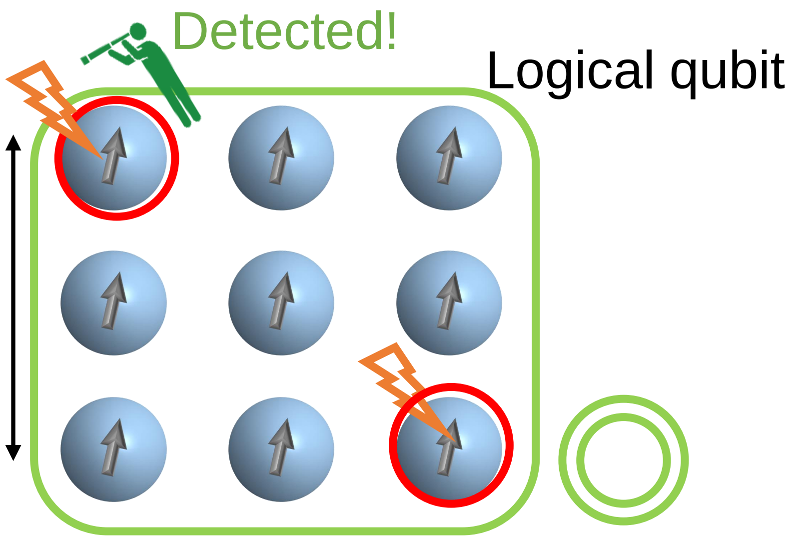
QEC is necessary due to high error rate

- Error rate due to short lifetime of physical qubit is large
- Quantum error correction(QEC) reduces error rate
  - Physical qubits are encoded to logical qubits
  - Error rate of logical qubit can be reduced to any small value



Code distance:
Least error number to change logical state

Error rate reduces to
any small value by increasing code distance

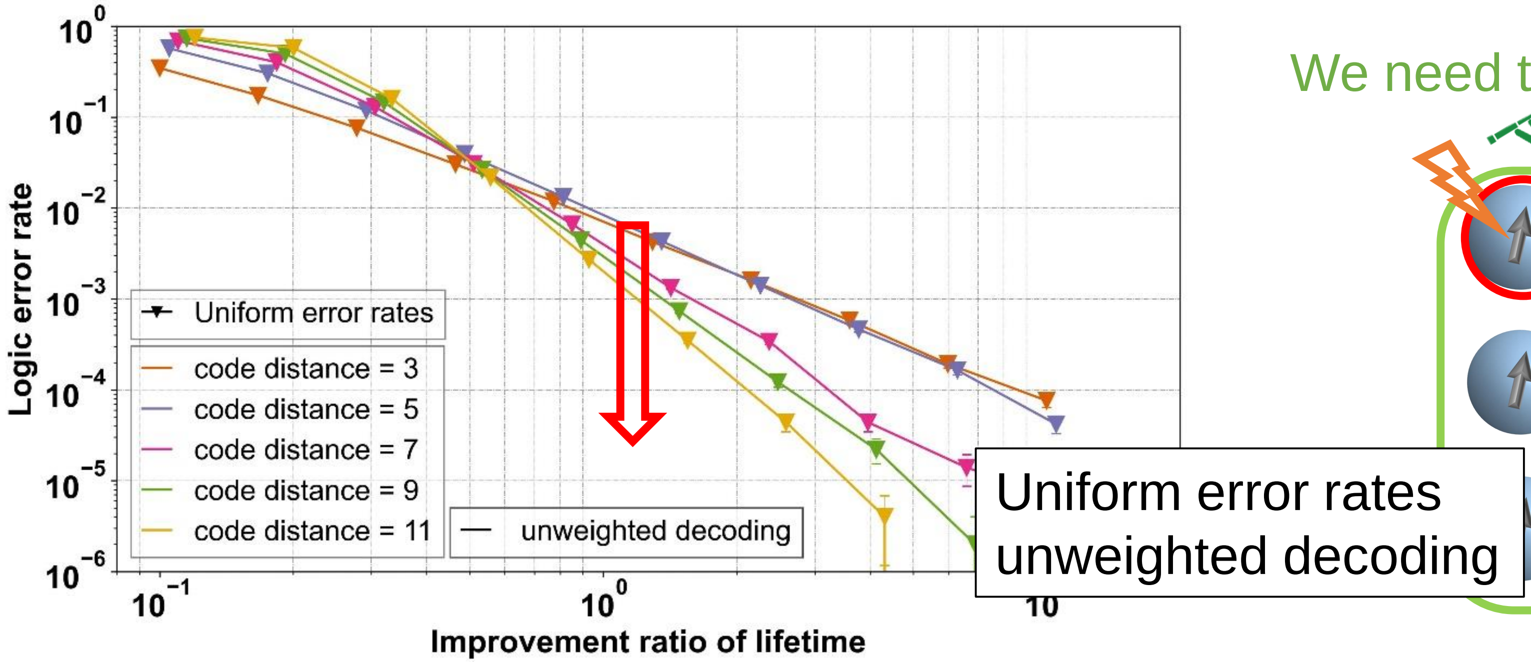


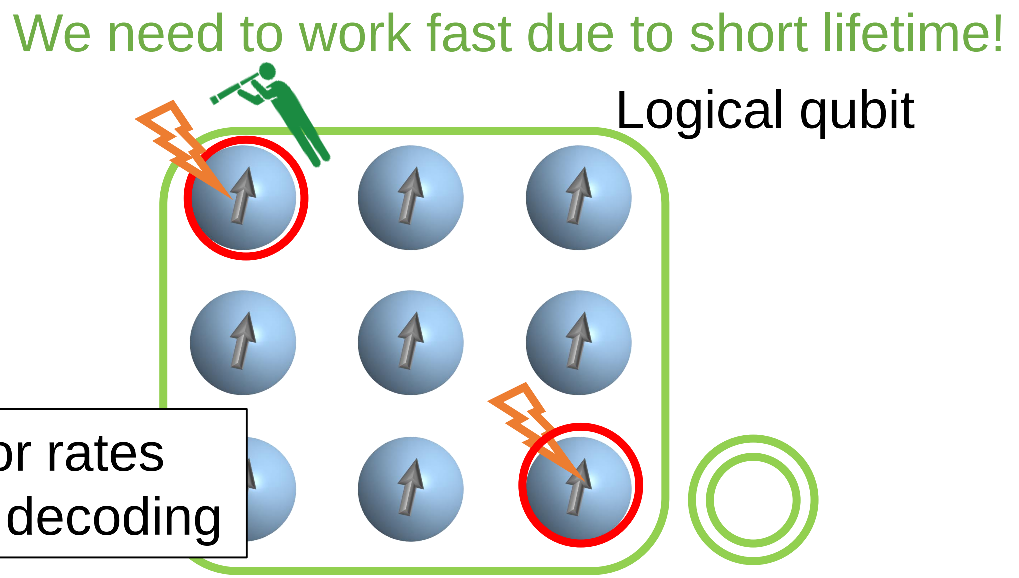
Logical error rate is reduced w/
the same single bit error rate

Non-uniform error rates jeopardize QEC

- Reason:
  - Fast correction is necessary due to short lifetime, but assumes ideal uniform error rates (unweighted decoding)

Decoding: error estimation in QEC code



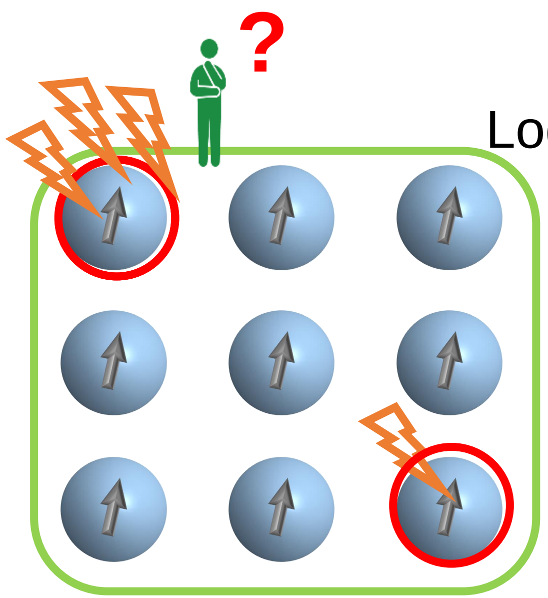


Ideal qubits: uniform error rates

Non-uniform error rates jeopardize QEC

- Reason:
  - Fast correction is necessary due to short lifetime, but assumes ideal uniform error rates (unweighted decoding)
  - Non-uniform error rates are the nature of real qubits

Decoding: error estimation in QEC code

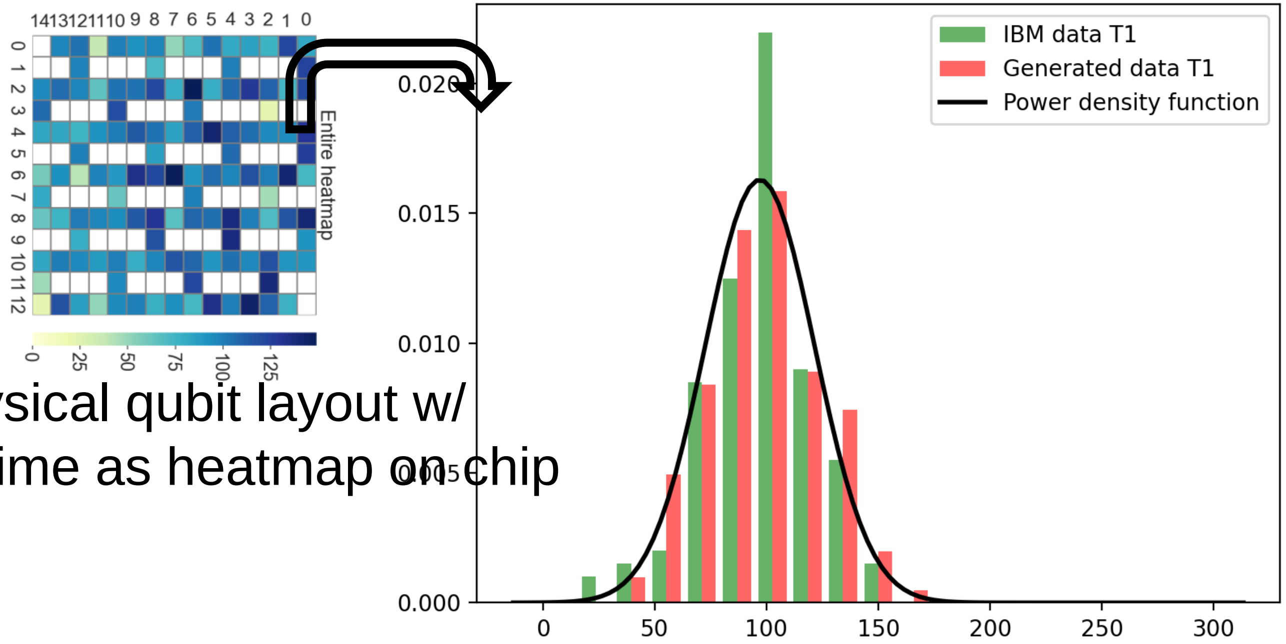


Ideal qubits: uniform error rates
Real qubits: Non-uniform err. rates

Non-uniform error rates jeopardize QEC

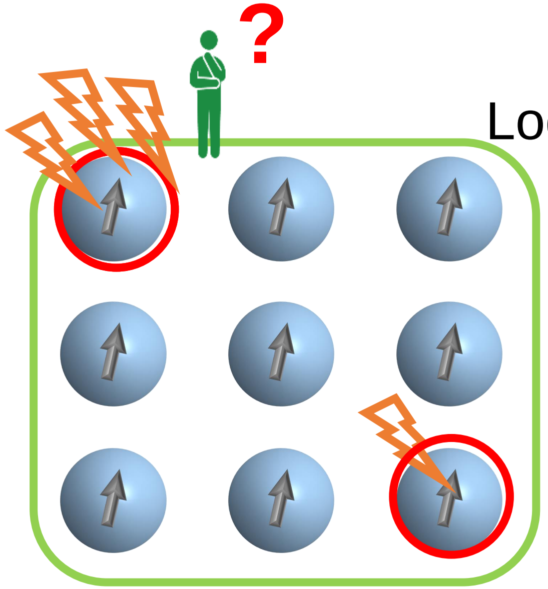
- Reason:
  - Fast correction is necessary due to short lifetime, but assumes ideal uniform error rates (unweighted decoding)
  - Non-uniform error rates are the nature of real qubits

Decoding: error estimation in QEC code



Physical qubit layout w/ lifetime as heatmap on chip

Profiling distribution of non-uniform error rates of real qubits in IBMQ chips (Eagle)[1]



Ideal qubits: uniform error rates
Real qubits: Non-uniform err. rates

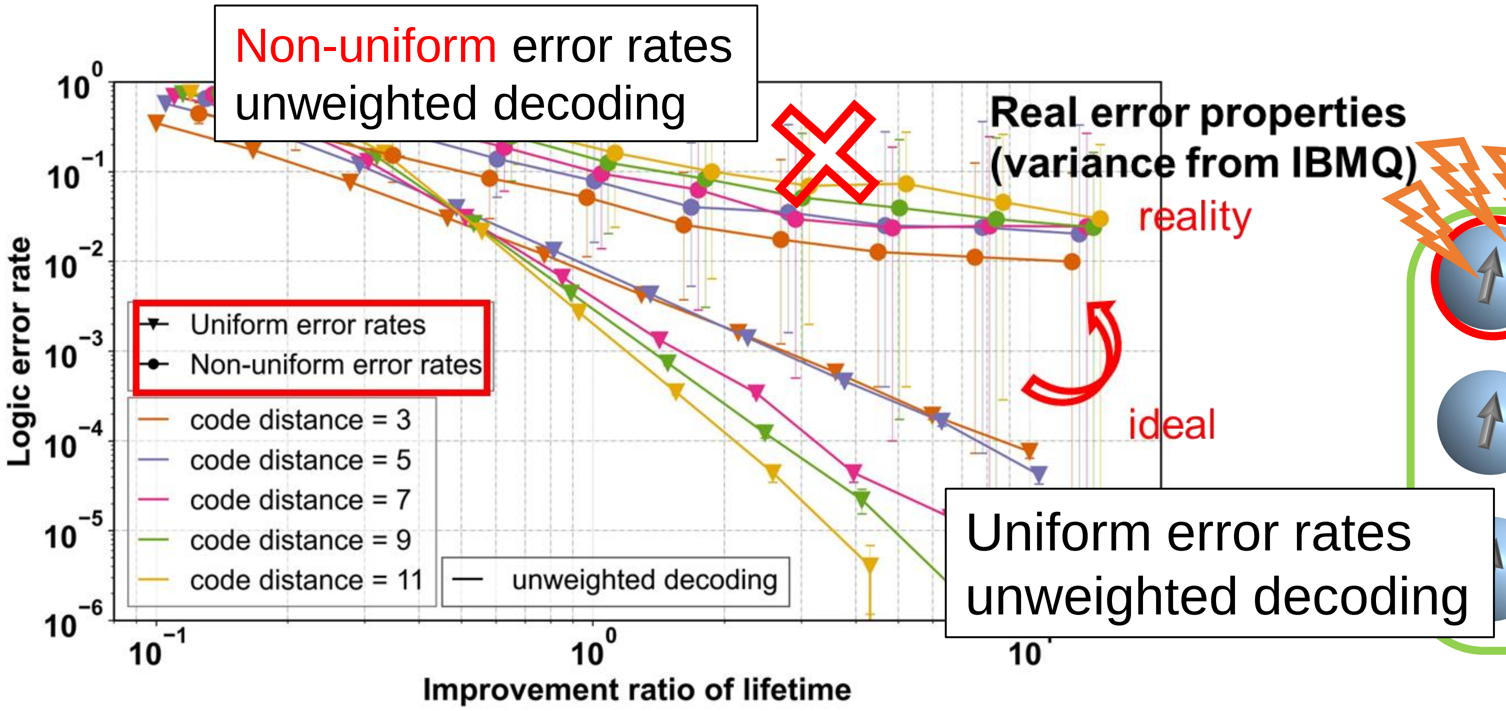
[1] IBMQ, https://quantum-computing.ibm.com/services/resources

Non-uniform error rates jeopardize QEC

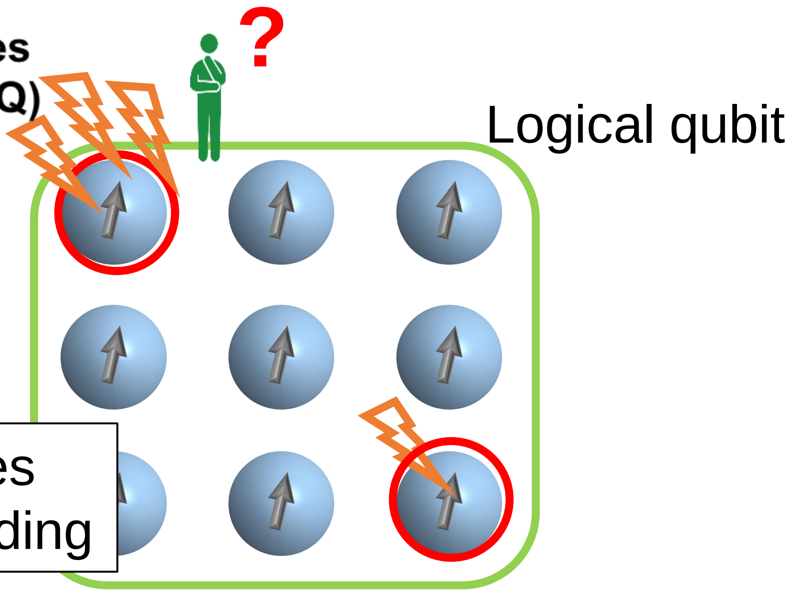
- Reason:

Decoding: error estimation in QEC code

  - Fast correction is necessary due to short lifetime, but assumes ideal uniform error rates (unweighted decoding)
  - Non-uniform error rates are the nature of real qubits



Monte Carlo simulation result:
no error rate reduction by larger code distance

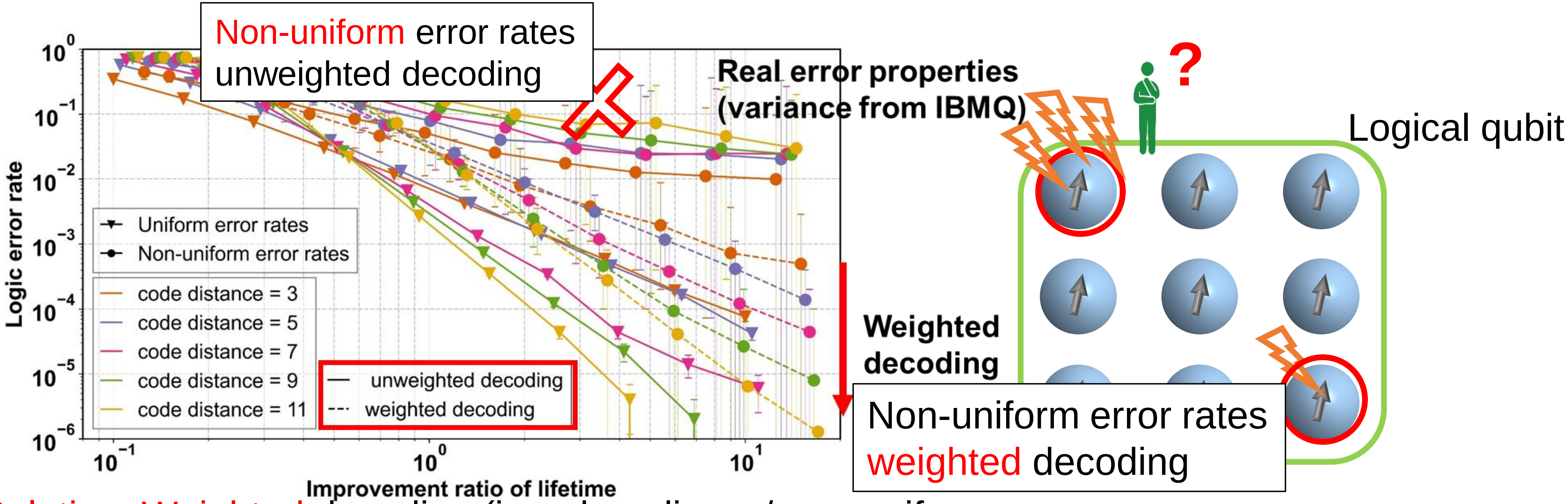


Ideal qubits: uniform error rates
Real qubits: Non-uniform err. rates

Non-uniform error rates jeopardize QEC

- Reason:
  - Fast correction is necessary due to short lifetime, but assumes ideal uniform error rates (unweighted decoding)
  - Non-uniform error rates are the nature of real qubits

Decoding: error estimation in QEC code



Solution: Weighted decoding (i.e., decoding w/ non-uniform error rates) can help to re-gain distance benefit, but…

Ideal qubits: uniform error rates
Real qubits: Non-uniform err. rates

Non-uniform error rates jeopardize QEC

- Reason:
  - Fast correction is necessary due to short lifetime, but assumes ideal uniform error rates (unweighted decoding)
  - Non-uniform error rates are the nature of real qubits
- Current issue: ✔ we solved in this paper
  - No fast error estimating device (i.e. decoder) for weighted decoding (i.e., error estimation w/ non-uniform error rates) large-distance code

TABLE I

COMPARISON OF HARDWARE IMPLEMENTATION OF SURFACE CODE DECODERS. OUR FRAMEWORK IS ABLE TO SCALE UP AND TAKE FABRICATION VARIANCE INTO CONSIDERATION.

Previous work — Small code distance only — No support for weighted

Decoding algorithm	d_{max}	Latency	Measurement error?	Decoding window length	Target device	Weighted matching?	Scalability
Look-up table [15]	5	42 ns	Y	3	FPGA	Y	Hard
Neural network [16]	5	194 ns	N	-	FPGA	Y	Hard
Union-find [17]	11	325 ns	Y	11	FPGA	N	Easy
Greedy [18]	9	162.72 ps	N	-	SFQ	N	Easy
Greedy [19]	13	215 ps	Y	3	SFQ	N	Easy
Greedy (Our proposal)	11	370 ns	Y	11	FPGA	Y	Easy

Highlight of this paper

- Current issue: No fast decoder for weighted decoding (i.e., error estimation w/ non-uniform error rates) large-distance code
- ✔ Our contribution: we solved it by designing a fast hardware decoder
  - Fast: average delay within time budget
  - Weighted decoding (i.e., error estimation w/ non-uniform error rates)
  - Large distance: support surface code up to d=11

TABLE I

COMPARISON OF HARDWARE IMPLEMENTATION OF SURFACE CODE DECODERS. OUR FRAMEWORK IS ABLE TO SCALE UP AND TAKE FABRICATION VARIANCE INTO CONSIDERATION.

Decoding algorithm	d_{max}	Latency	Measurement error?	Decoding window length	Target device	Weighted matching?	Scalability
Look-up table [15]	5	42 ns	Y	3	FPGA	Y	Hard
Neural network [16]	5	194 ns	N	-	FPGA	Y	Hard
Union-find [17]	11	325 ns	Y	11	FPGA	N	Easy
Greedy [18]	9	162.72 ps	N	-	SFQ	N	Easy
Greedy [19]	13	215 ps	Y	3	SFQ	N	Easy
Greedy (Our proposal)	11	370 ns	Y	11	FPGA	Y	Easy

Previous work

Small code distance only

No support for weighted

Contents

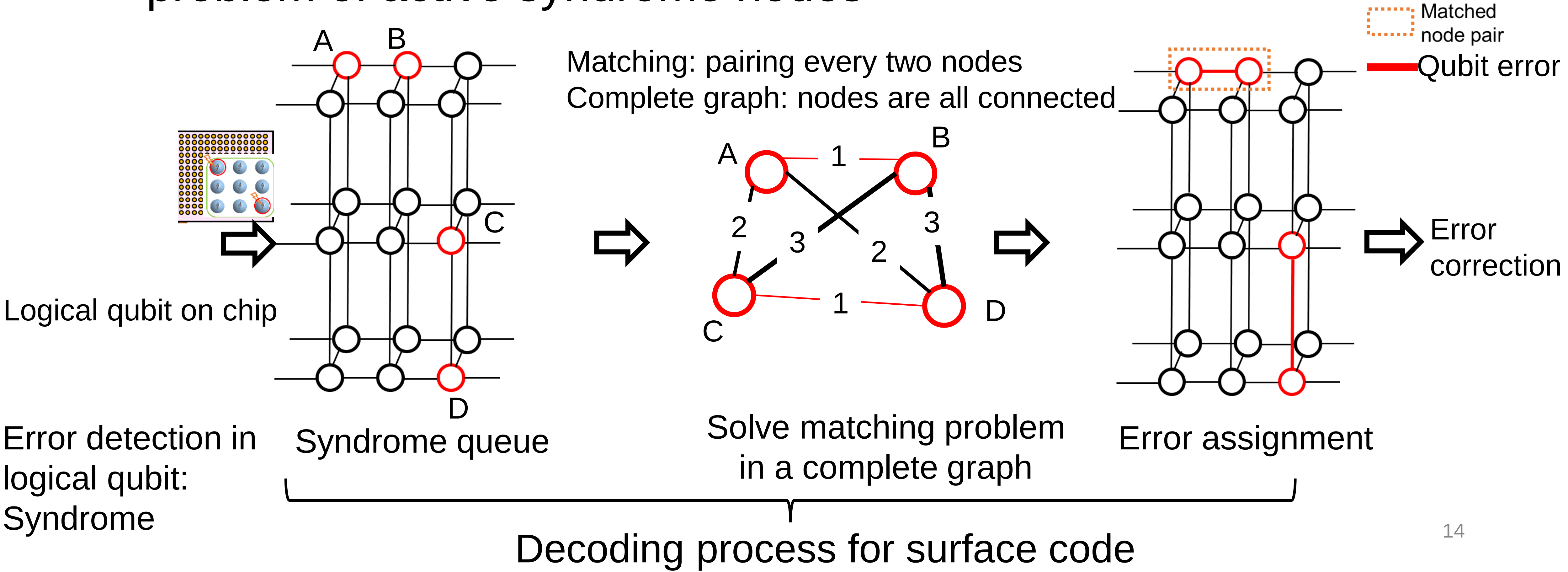
- Introduction of decoder for surface code
- Proposed weighted hardware decoder of WIT-Greedy
- Conclusion

Contents

- Introduction of decoder for surface code
- Proposed weighted hardware decoder of WIT-Greedy
- Conclusion

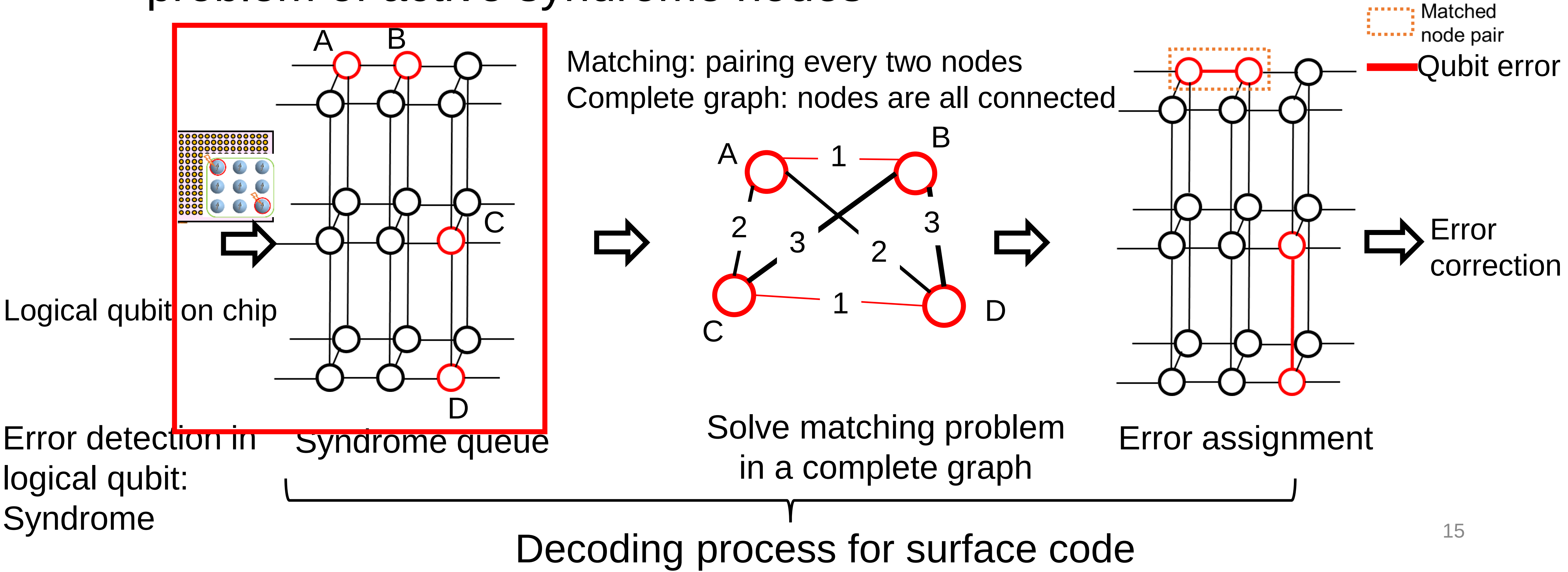
Overview of decoding for surface code

- Surface code is currently the most promising QEC code
- Decoding: estimation of error allocation inside code
- Decoding task is essentially a classical problem: solve matching problem of active syndrome nodes



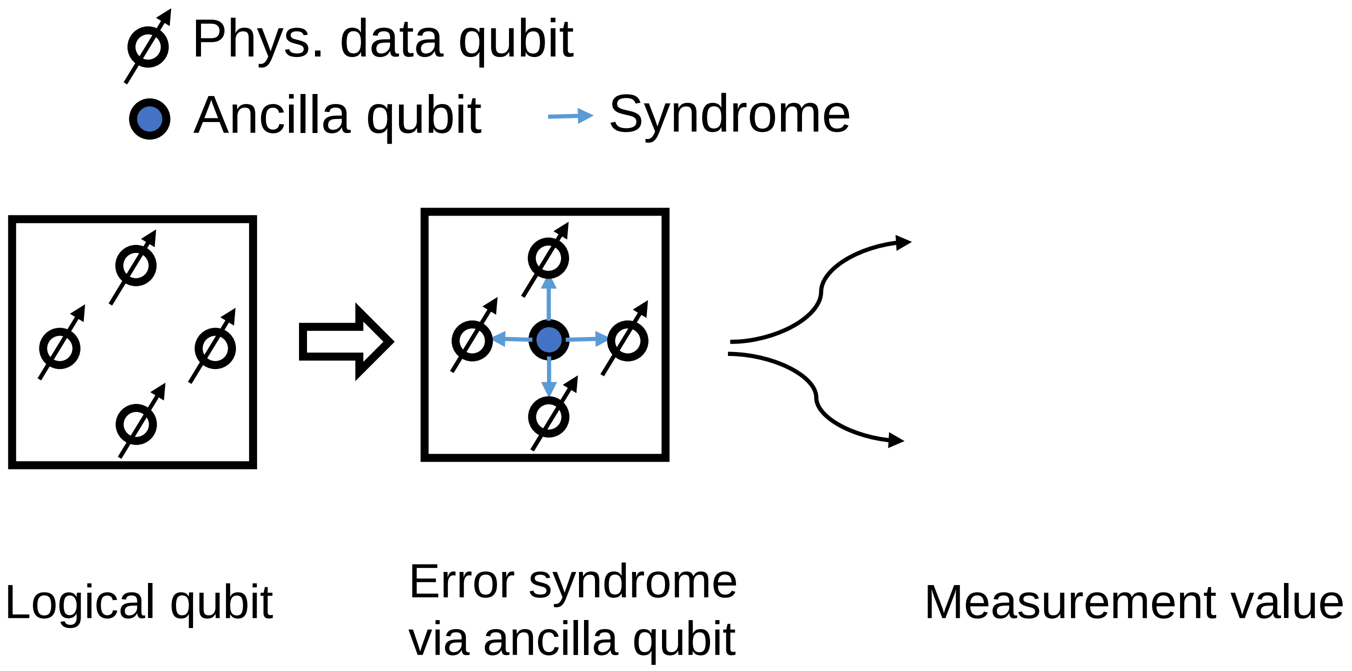
Overview of decoding for surface code

- Surface code is currently the most promising QEC code
- Decoding: estimation of error allocation inside code
- Decoding task is essentially a classical problem: solve matching problem of active syndrome nodes



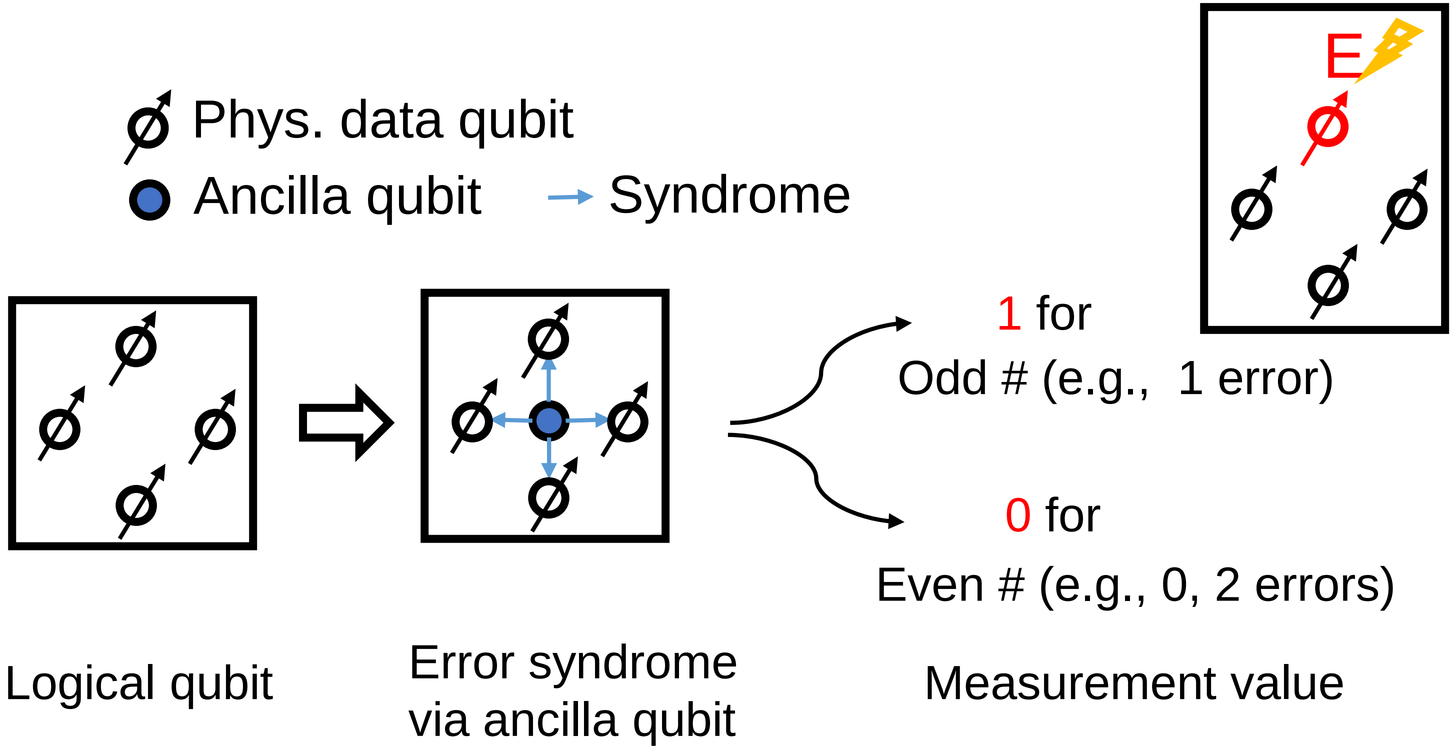
Error syndrome using surface code

- Error syndrome: parity check # of errors in neighbor data qubits → error detection in QEC



Error syndrome using surface code

- Error syndrome: parity check # of errors in neighbor data qubits → error detection in QEC

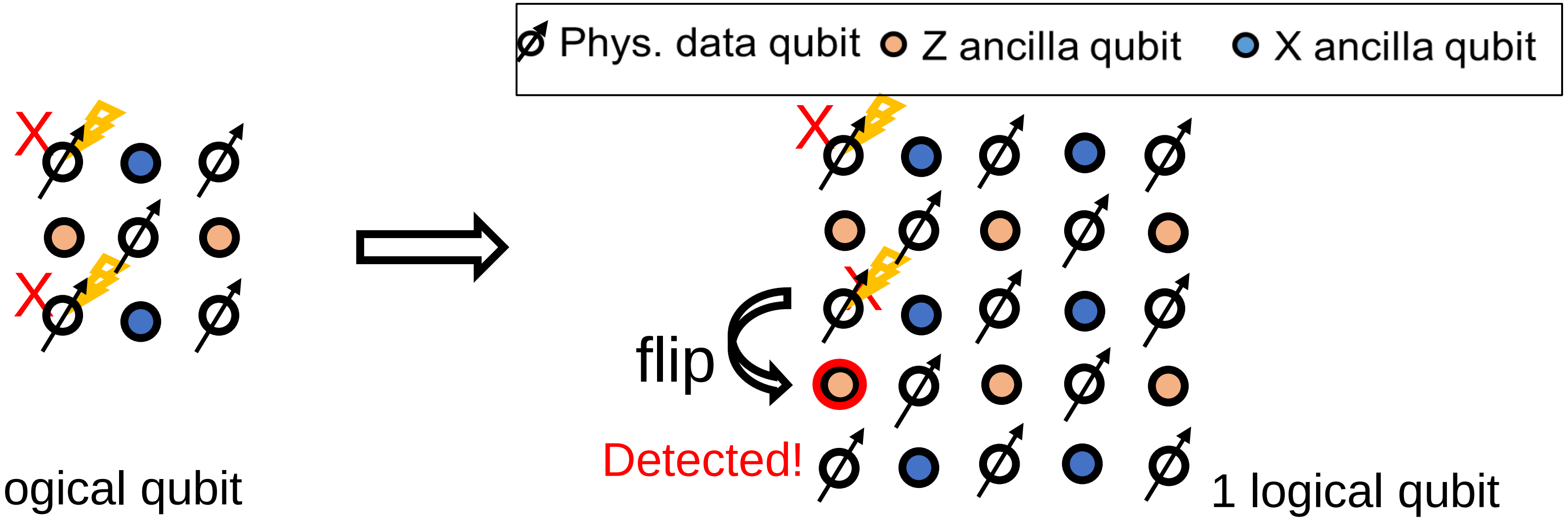


Error syndrome using surface code

- Error syndrome: parity check # of errors in neighbor data qubits → error detection in QEC
- Code distance *d*: error detection ability

d = maximum # of errors to detect

*: Two types of errors exist in QC as X and Z types

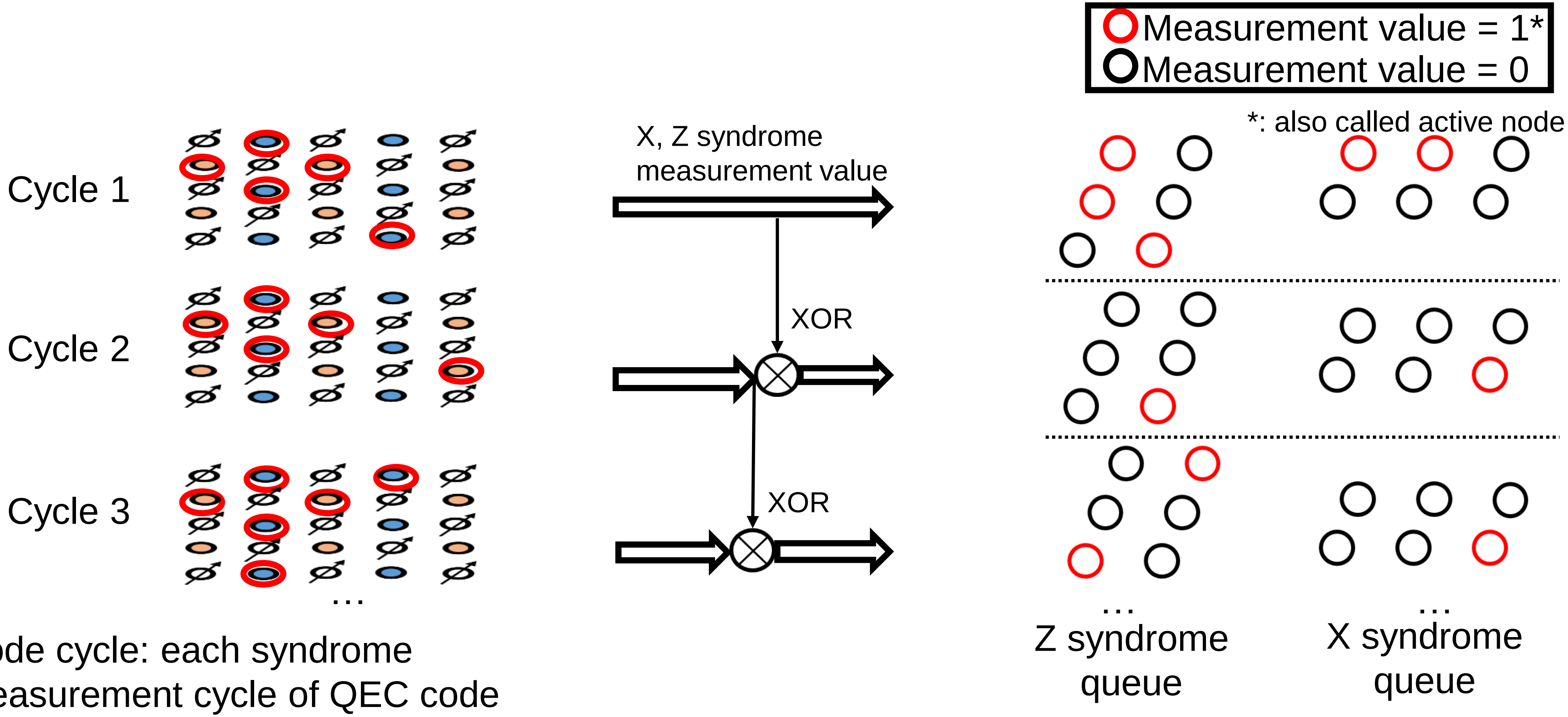


1 logical qubit
(*d* = 2)

1 logical qubit
(*d* = 3)

Successive syndrome during computation

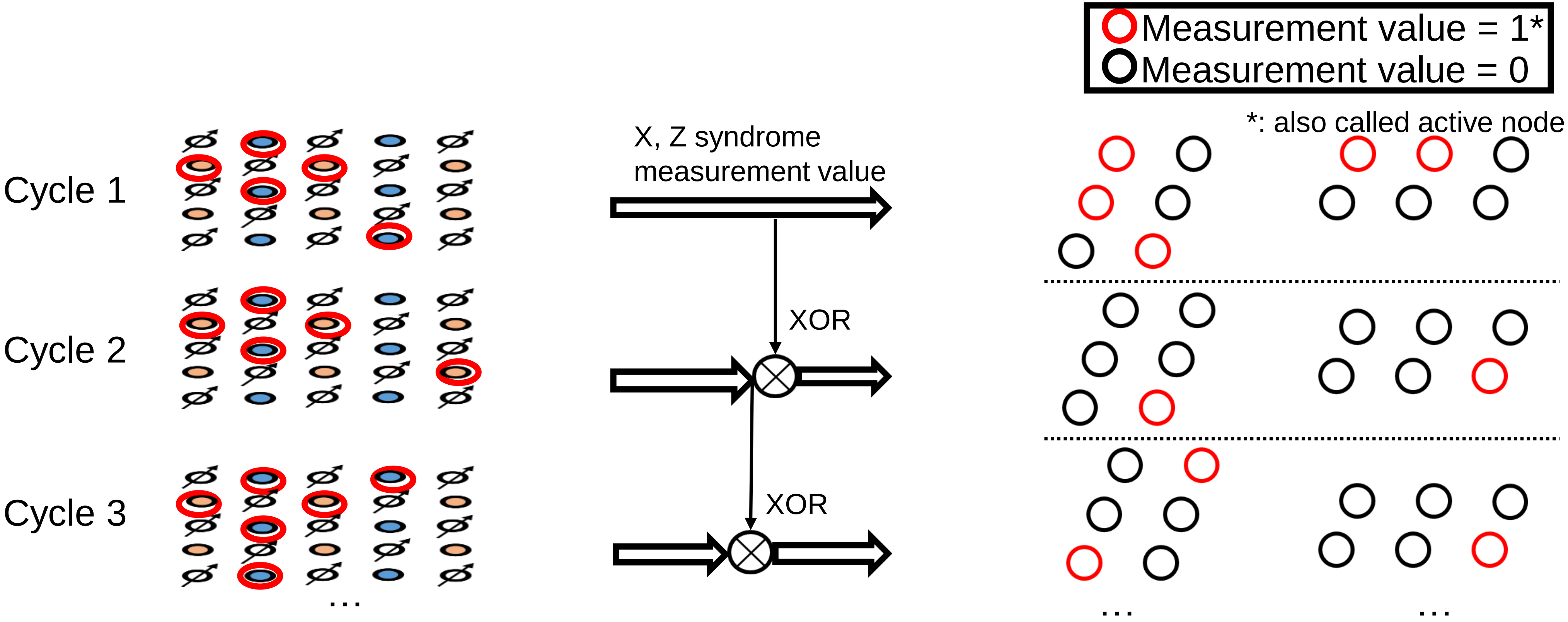
- Syndrome measurement is conducted every computation code cycle due to large error rates



Code cycle: each syndrome measurement cycle of QEC code

Successive syndrome during computation

- Syndrome measurement is conducted every computation code cycle due to large error rates

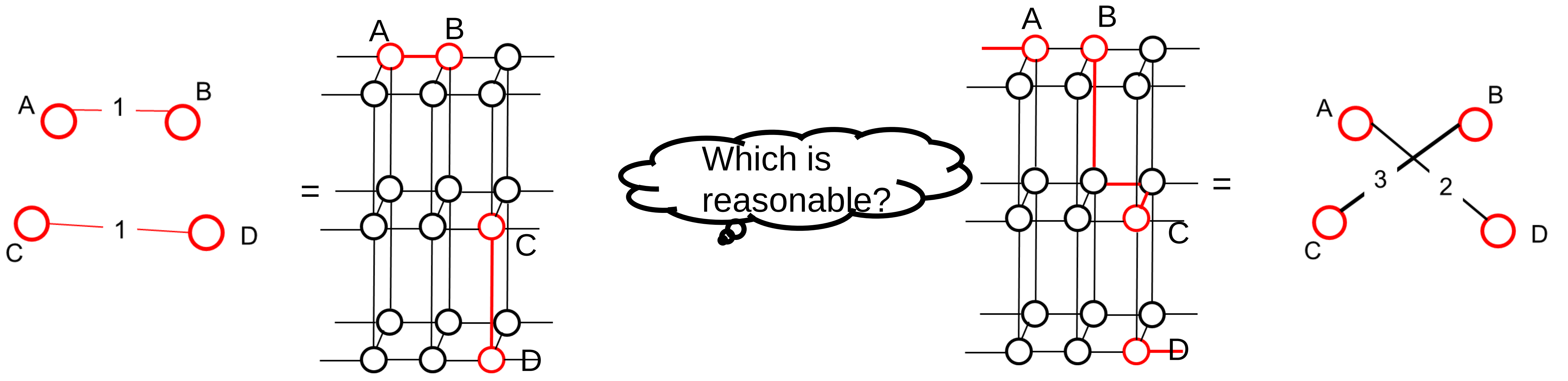


Code cycle: each syndrome measurement cycle of QEC code

How can we estimate errors using syndrome queue?

Convert decoding task of syndrome queue to a matching problem

- Most probable error estimation -> find minimum weight perfect matching (i.e. most probable error pattern)



Total weight of edges: 2 ✓

Total weight of edges: 5

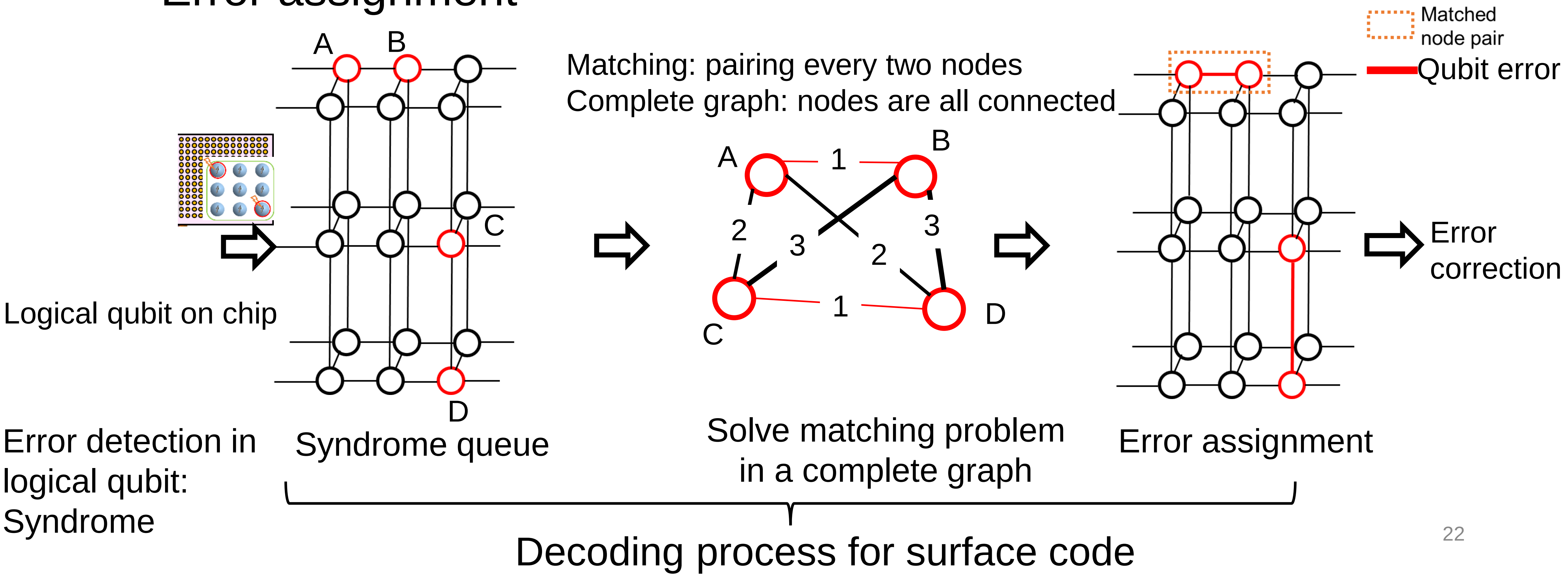
Weight of edge: related to the error rate of each qubit

Meaning of edges in graph:

- Horizontal edge > data qubit error
- Vertical edge -> ancilla qubit error

Summary of decoding process

- Pick up all active syndrome node (i.e. measurement value = 1)
- Re-construct complete graph with weights of path
- Solve matching problem of active syndrome nodes
- Error assignment



Contents

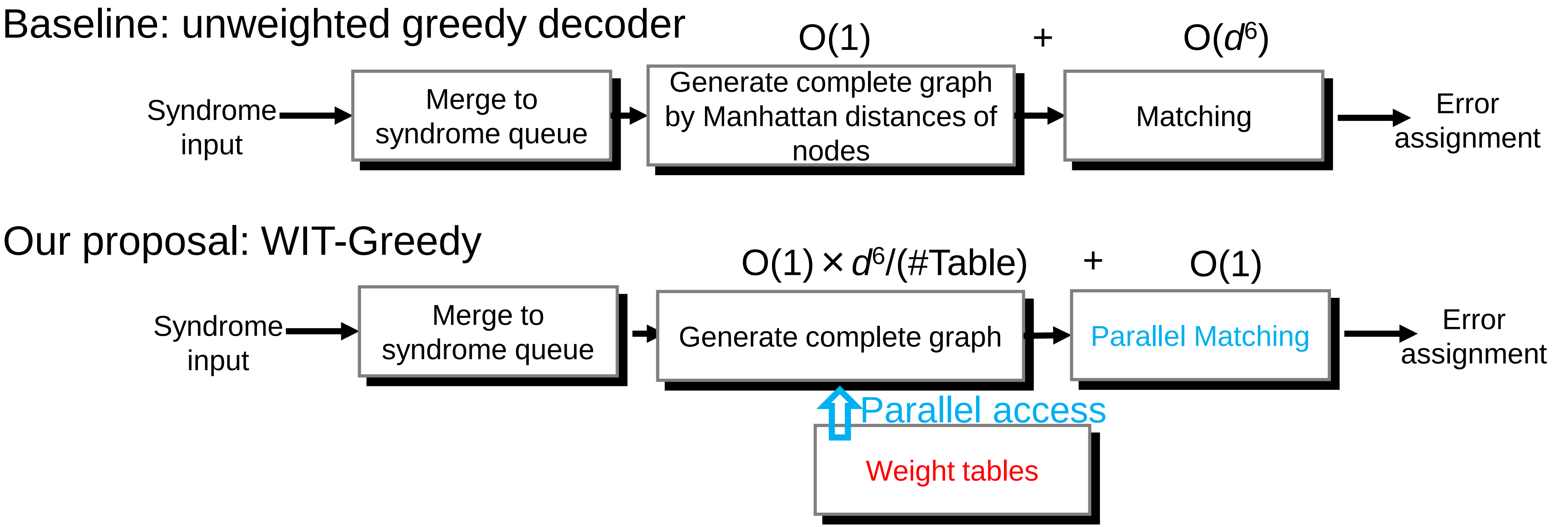
- Introduction of decoder for surface code
- Proposed weighted hardware decoder of WIT-Greedy
- Conclusion

Our contribution

- We developed fast weighted decoder with the same computation complexity of the baseline unweighted decoder
- Key idea: weight tables and parallel processing

Complexity and flow of hardware decoder

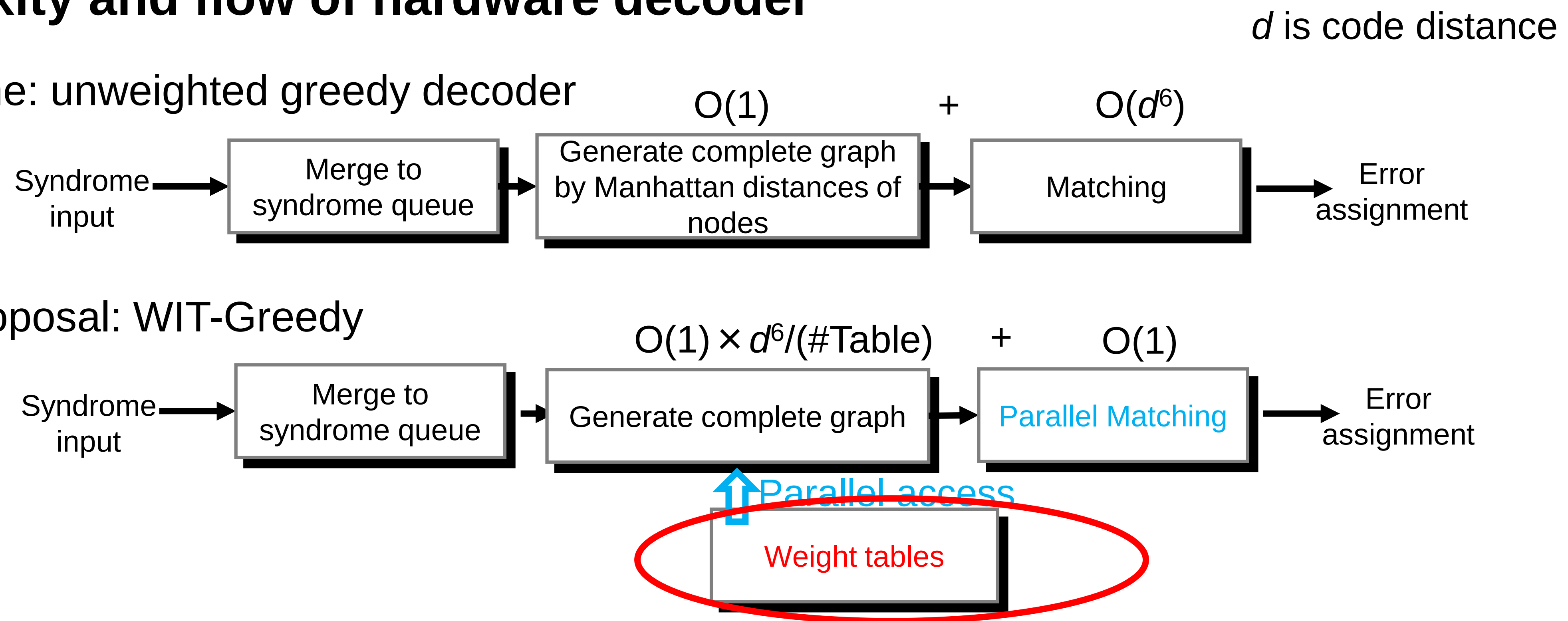
d is code distance



Our contribution

- We developed fast weighted decoder with the same computation complexity of the baseline unweighted decoder
- Key idea: weight tables and parallel processing

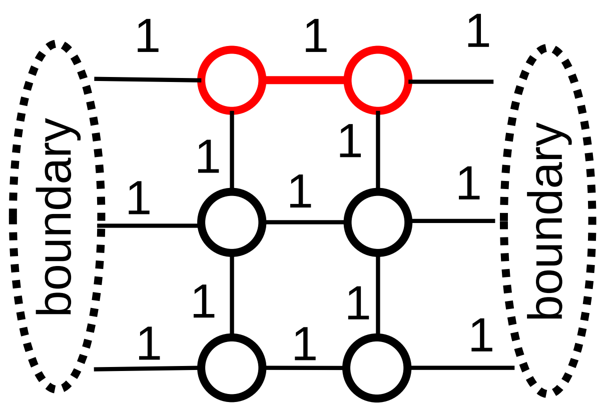
Complexity and flow of hardware decoder



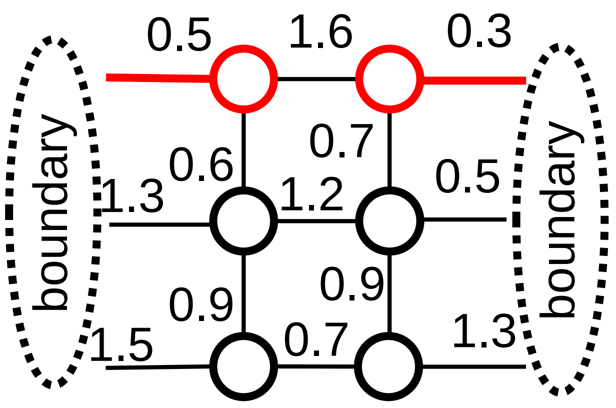
Difficulty to consider non-uniform weights

- Hard to find shortest path betw. nodes
  - Unweighted one can be calculated by Manhattan distance
- Shortest path needs repeated enumeration
  - Complexity of O(d^3) for d^6 times

d is code distance



Matching result of
uniform weight



Matching result of
non-uniform weight

Shortest path in uniform weight can be replaced by Manhattan distance

However, for non-uniform case, we can not use Manhattan distance

Our proposal: Pre-calculated weight table of shortest paths

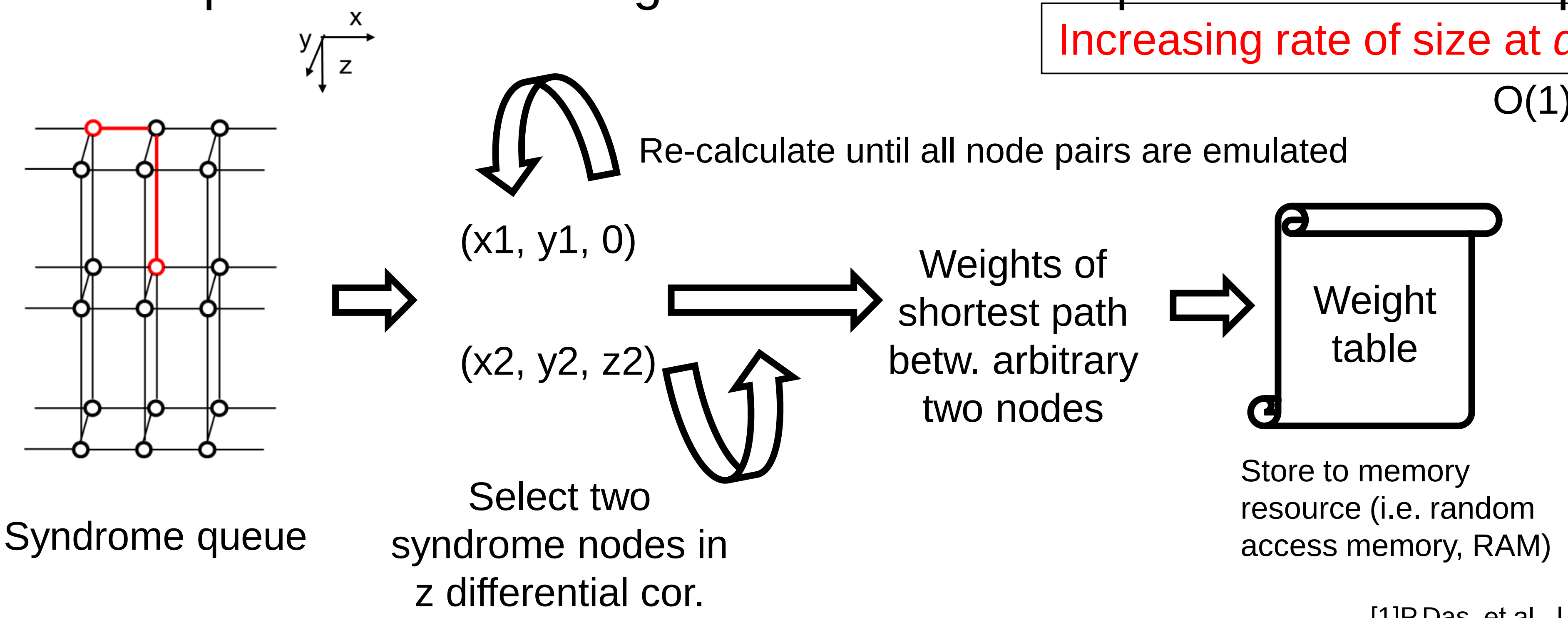
- Previous solutions: Complexity O(d^3)
  - 1. Perform shortest-path finding in each iteration -> time consuming
  - 2. Full look-up table for matching -> mem. resource exhaust at d=5 [1]

Increasing rate of size at 2^{d^3}

- Our solution:
  - Construct pre-calculated weight table of shortest path of each node pair

Increasing rate of size at d^5

O(1)



[1]P.Das, et al., LILLIPUT, ASPLOS'22

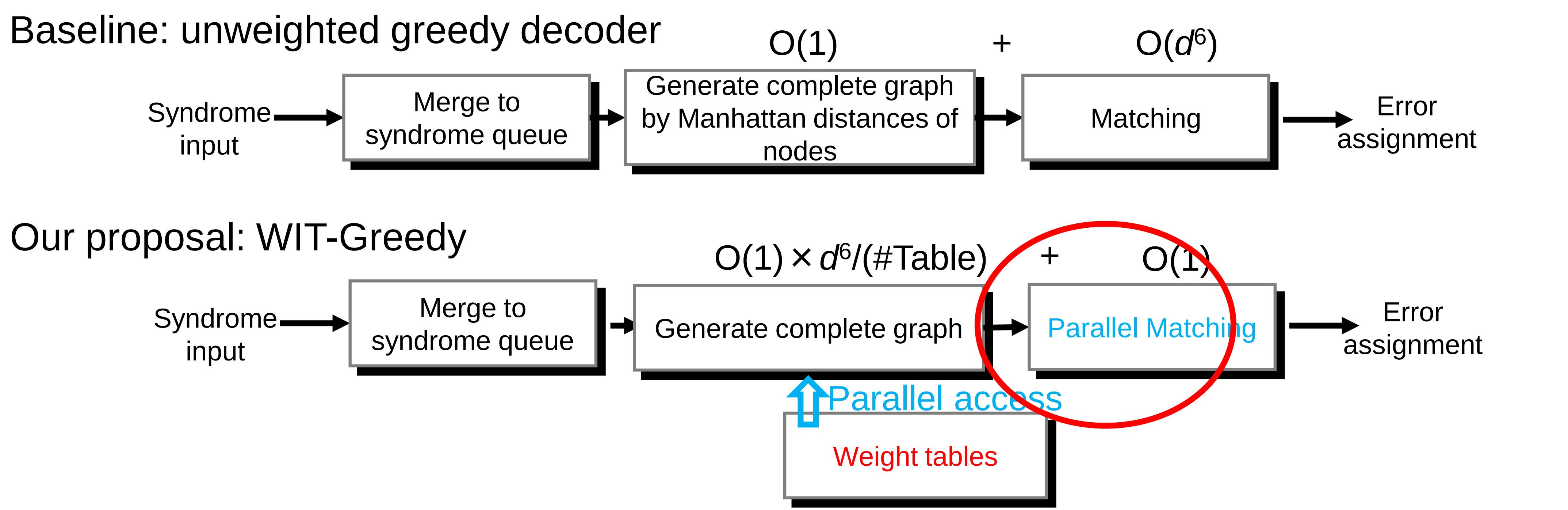
Our contribution

- We developed fast weighted decoder with the same computation complexity of the baseline unweighted decoder
- Key idea: weight tables and parallel processing

Accelerate matching

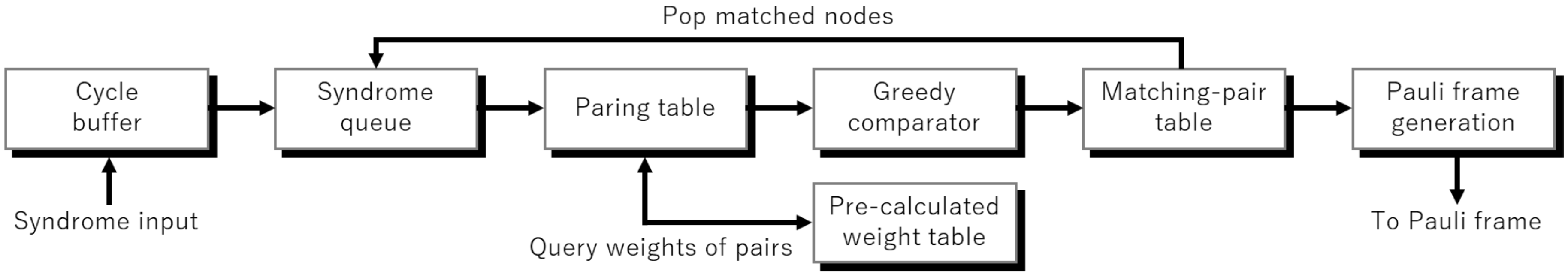
Complexity and flow of hardware decoder

d is code distance



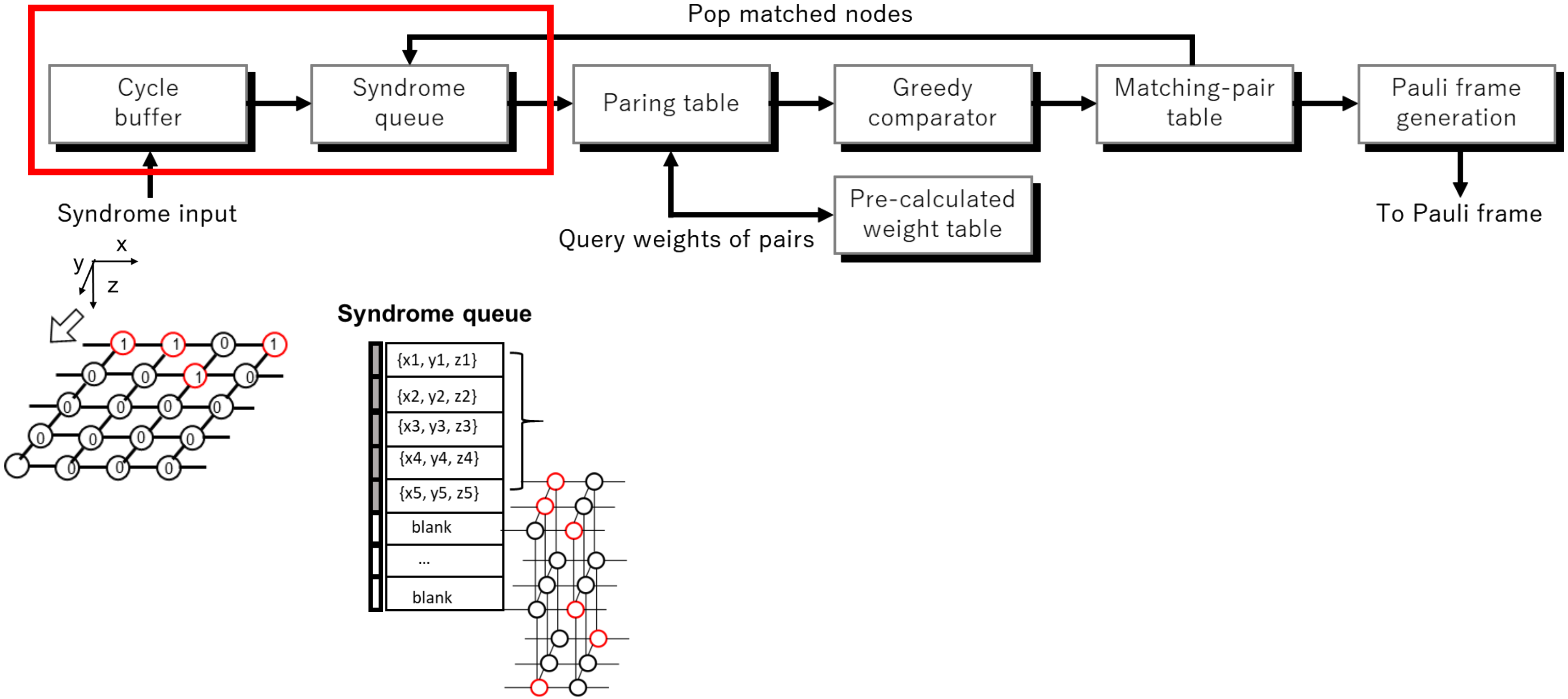
Overall HW design of WIT-Greedy for parallel processing

- Processing flow chart



Overall HW design of WIT-Greedy for parallel processing

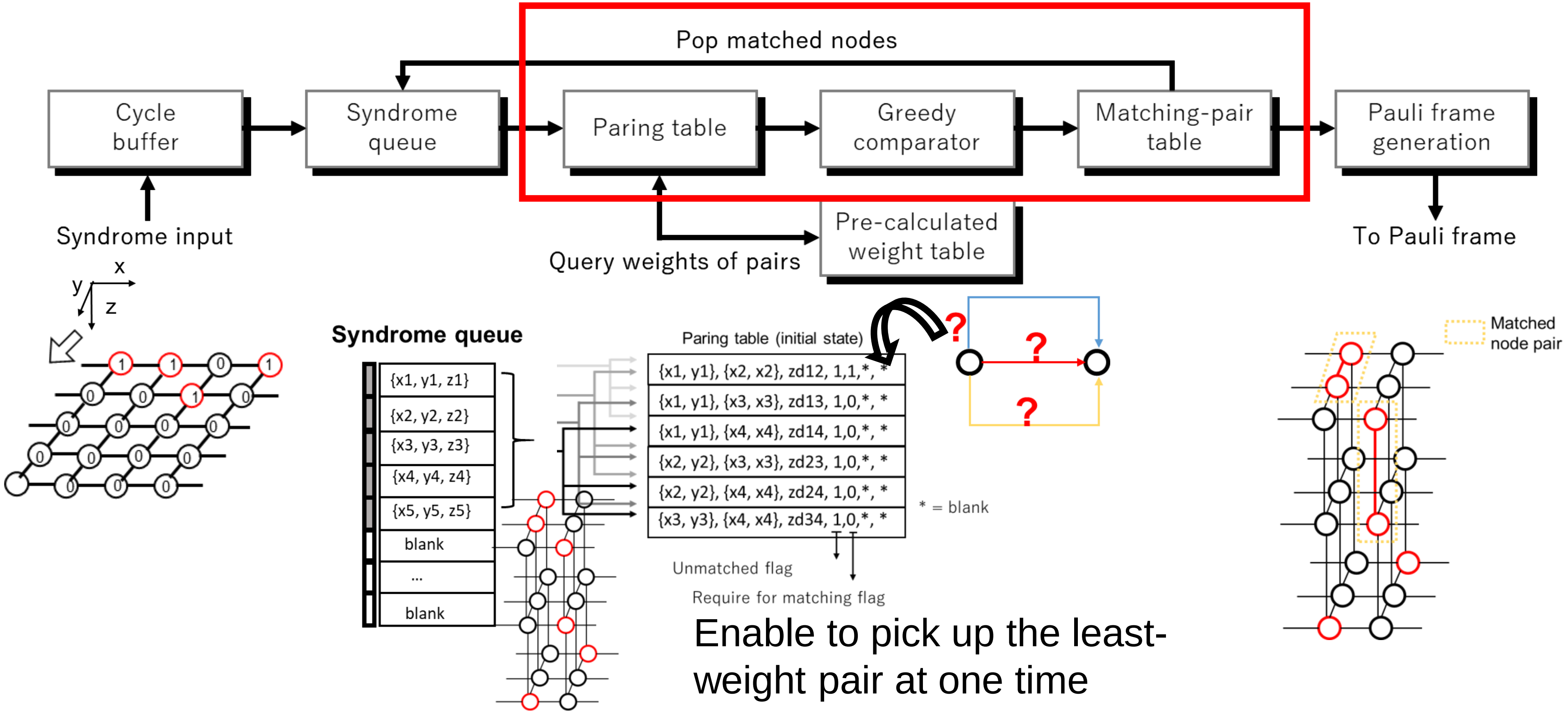
- Processing flow chart
  - Active syndrome node encoding → Generating address to query weight tables
  - Pair table based parallel comparator



Overall HW design of WIT-Greedy for parallel processing

- Processing flow chart
  - Active syndrome node encoding
  - Pair table based parallel comparator

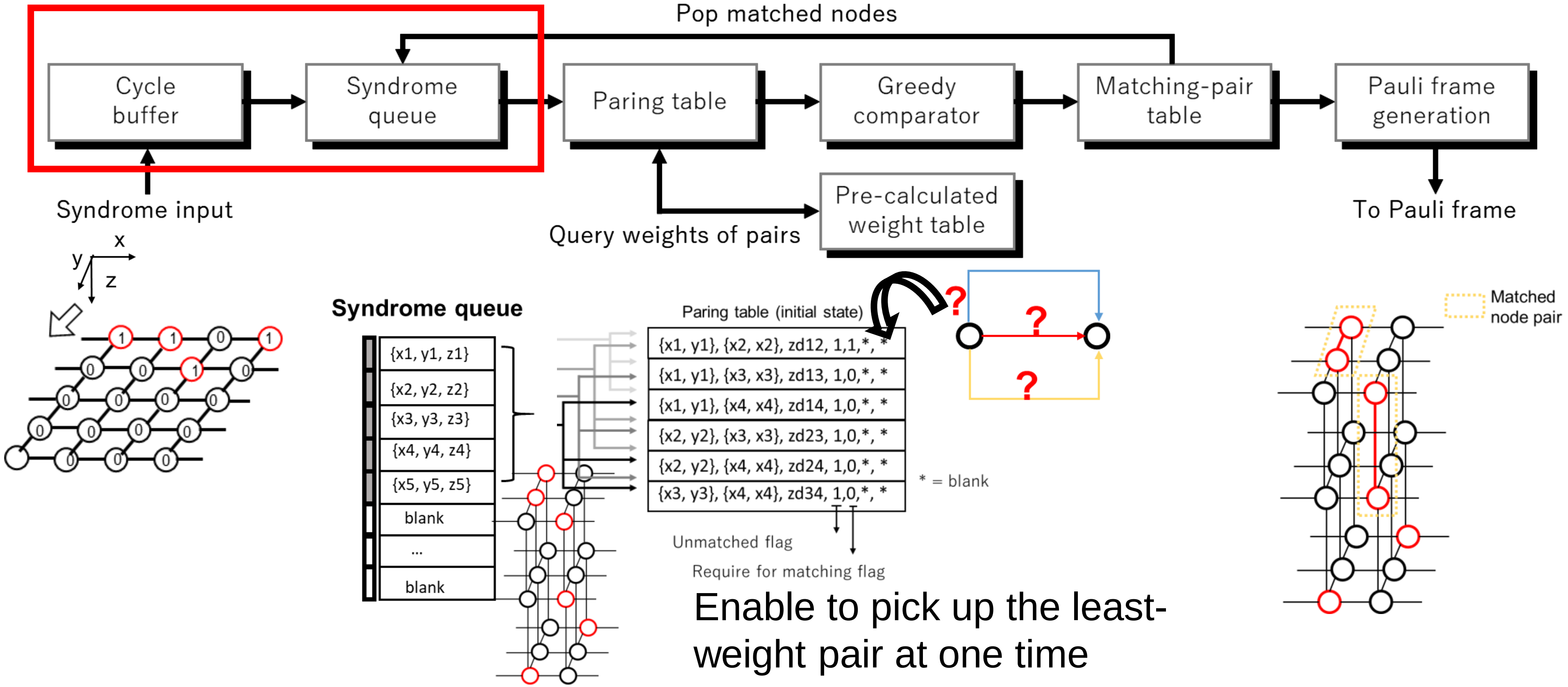
→ Matching nodes



Enable to pick up the least-weight pair at one time

Overall HW design of WIT-Greedy for parallel processing

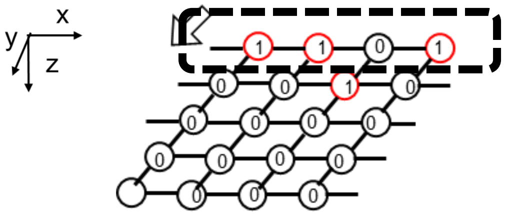
- Processing flow chart
  - Active syndrome node encoding
  - Pair table based parallel comparator



Enable to pick up the least-weight pair at one time

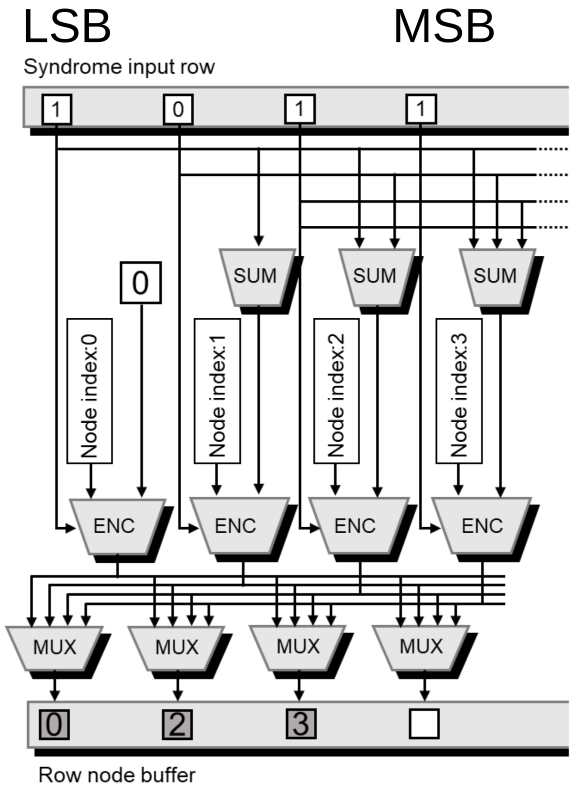
Parallel processing in active syndrome encoding

- Encoding rows: 1 clock delay for entire row
- Merge to syndrome queue



input

Array of syndrome measurement value:
1101 0010 0000 0000 0000…



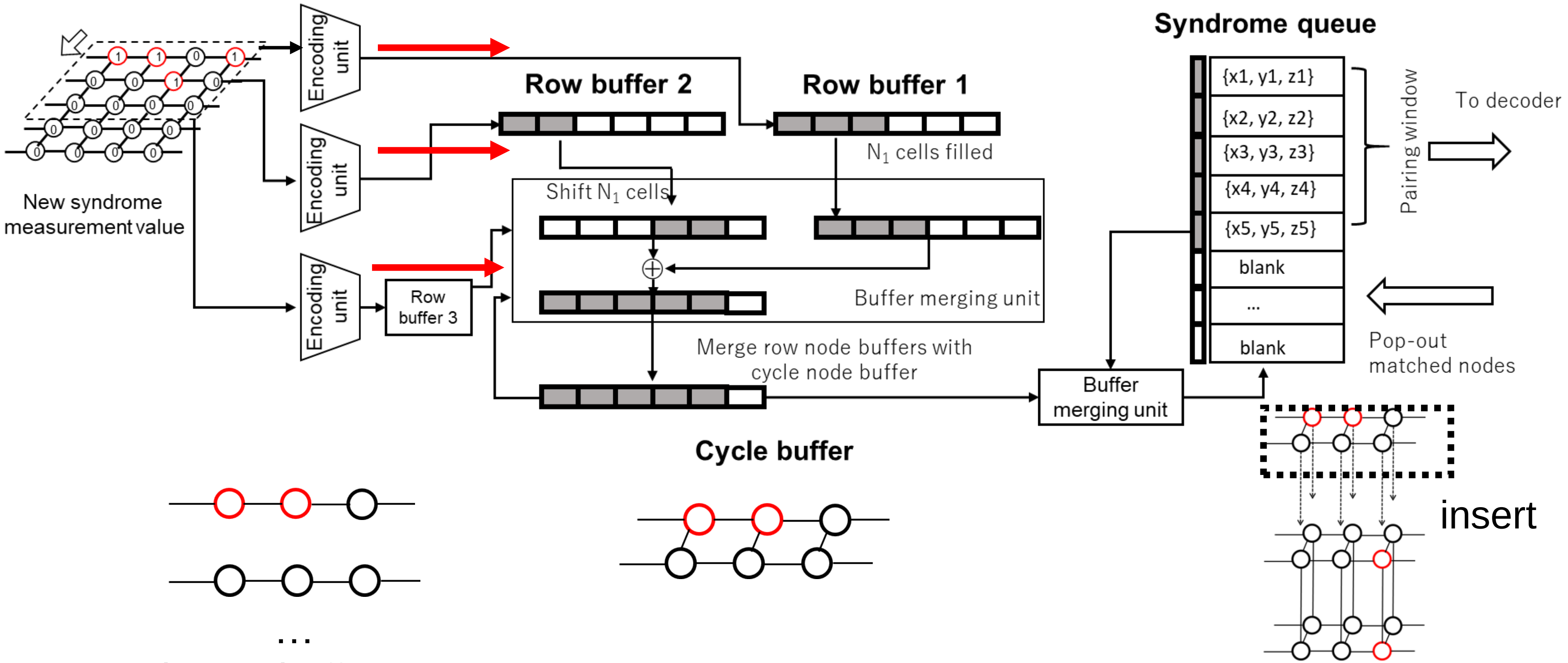
x offset of active syndrome

output

Row encoding unit

Parallel processing in active syndrome encoding

- Encoding rows: 1 clock delay for entire row
- Merge to syndrome queue: 3-stage pipeline



Data in row buffer

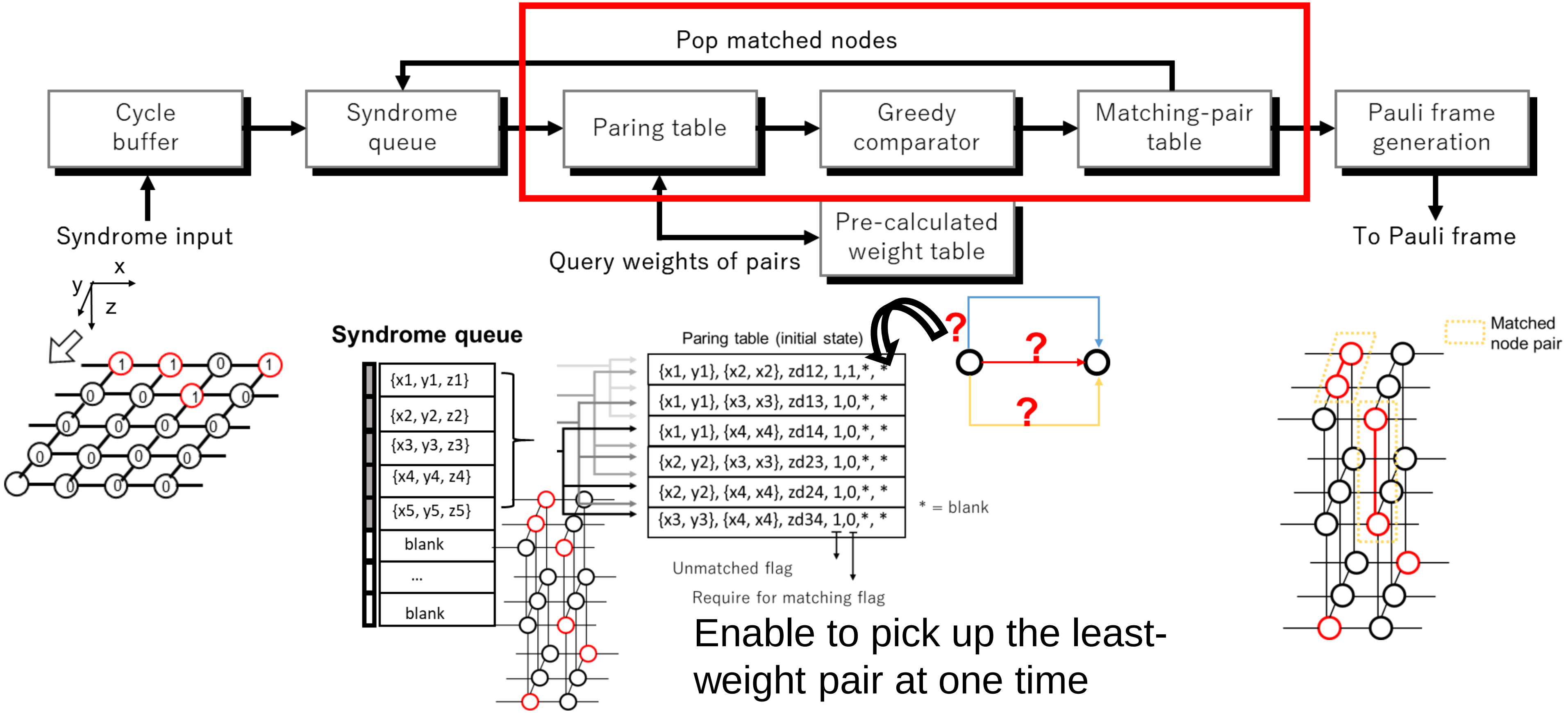
Data in cycle buffer

Data in syndrome queue

*: These buffers are all registers (FFs)

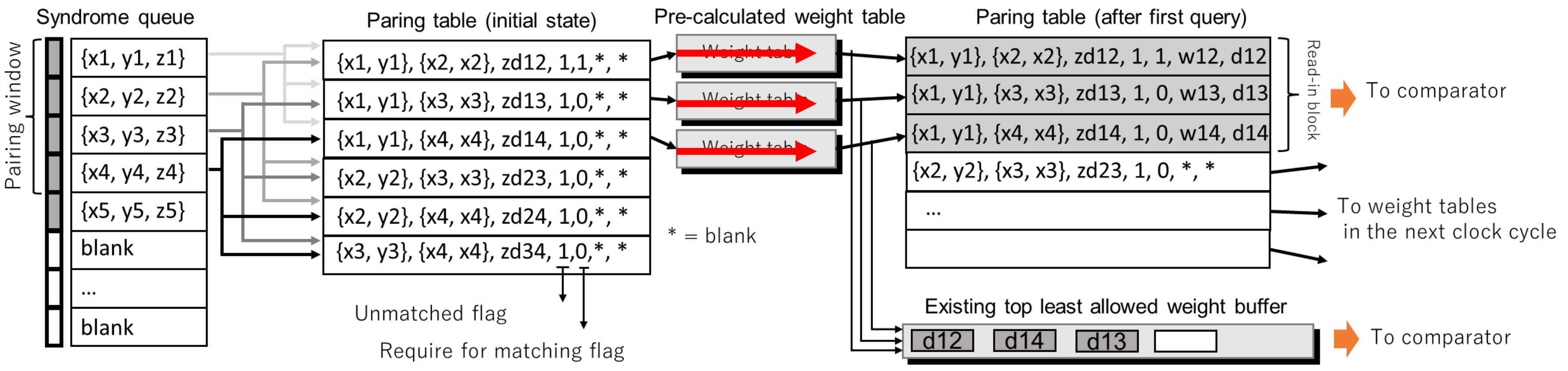
Overall HW design of WIT-Greedy for parallel processing

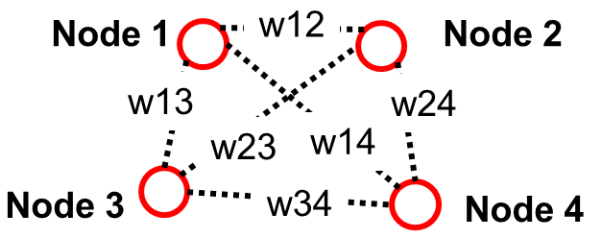
- Processing flow chart
  - Active syndrome node encoding
  - Pair table based parallel comparator



Generate pair table for parallel matching

- Large pair table of registers is created for comparing all the pairs simultaneously
  - Parallel accessing weight tables to read weight of each node pairs
  - Record existing top least weight

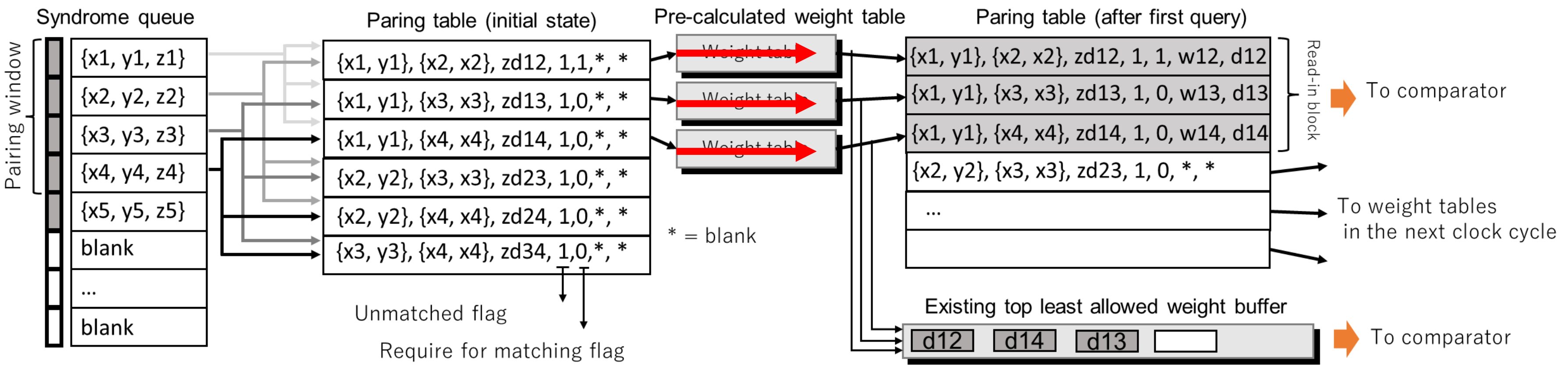


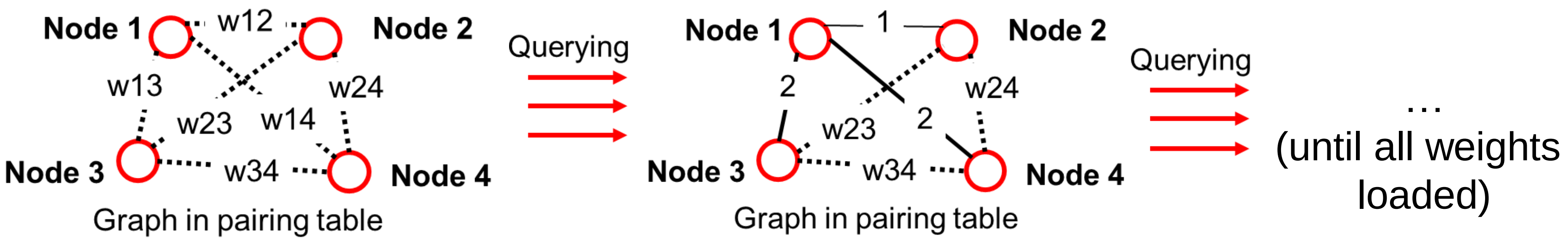


Graph in pairing table

Generate pair table for parallel matching

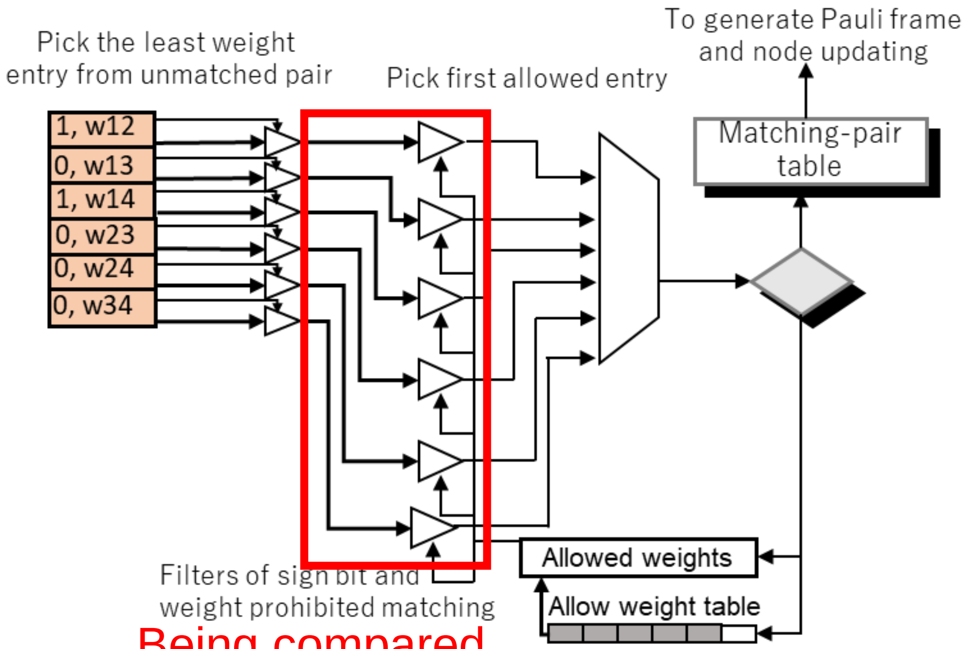
- Large pair table of registers is created for comparing all the pairs simultaneously
  - Parallel accessing weight tables to read weight of each node pairs
  - Record existing top least weight



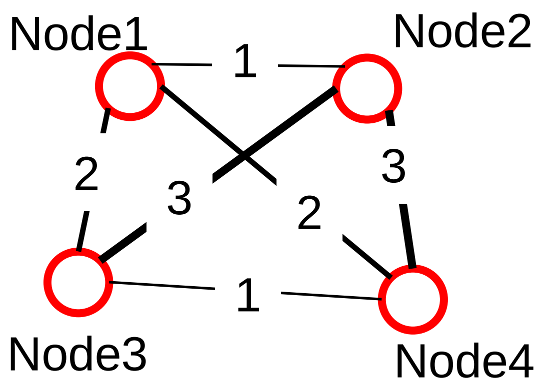


Parallel processing for matching

- Parallel greedy algorithm:
  - pick up the first pair of the current least weight @minimum of 1 clock
  - Turn off flags of pairs containing matched nodes
  - Pop-up pairs for Pauli frame generation in parallel

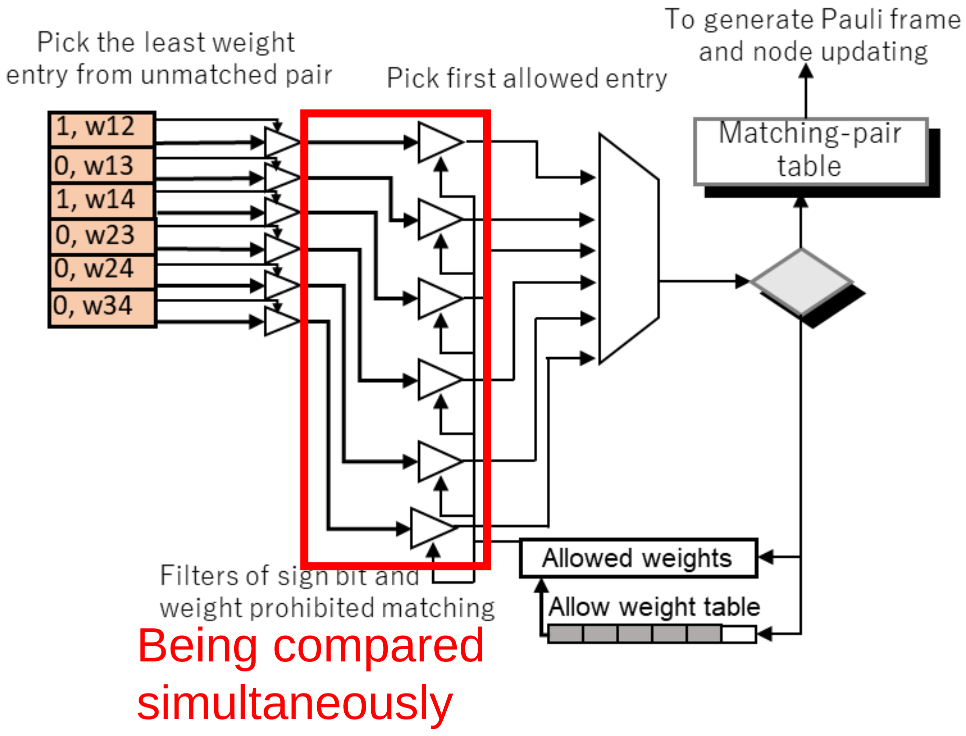


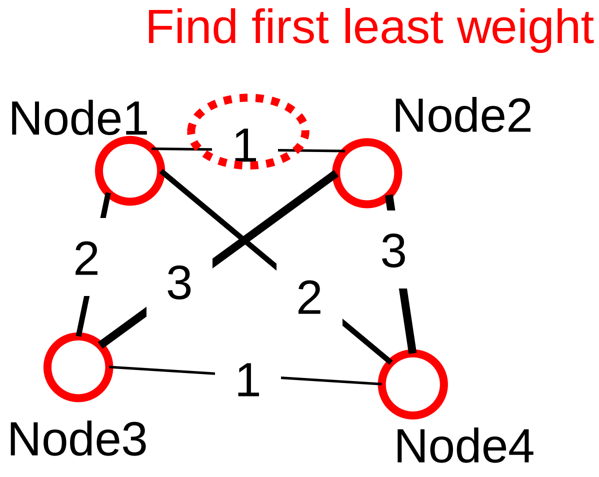
Being compared
simultaneously



Parallel processing for matching

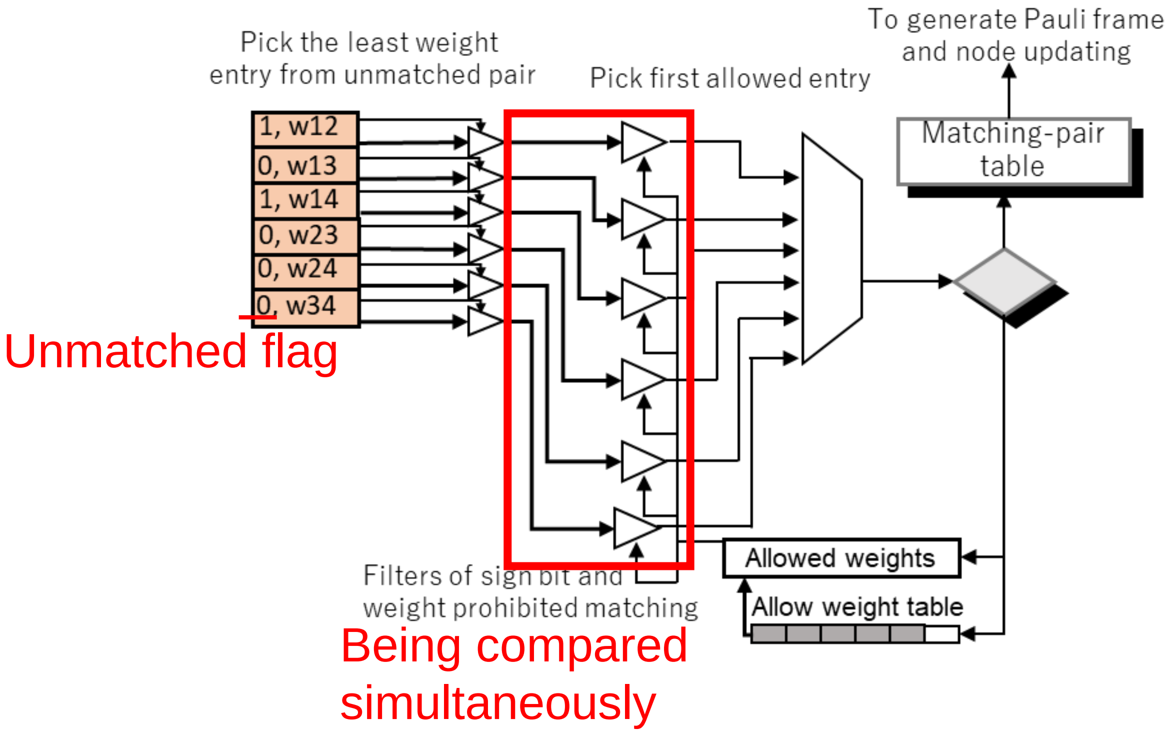
- Parallel greedy algorithm:
  - pick up the first pair of the current least weight @minimum of 1 clock
  - Turn off flags of pairs containing matched nodes
  - Pop-up pairs for Pauli frame generation in parallel

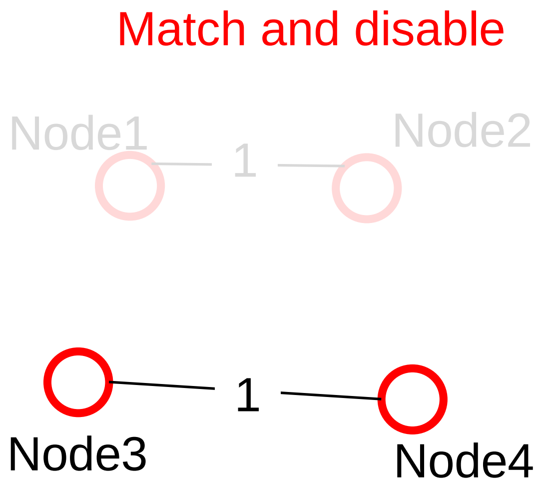




Parallel processing for matching

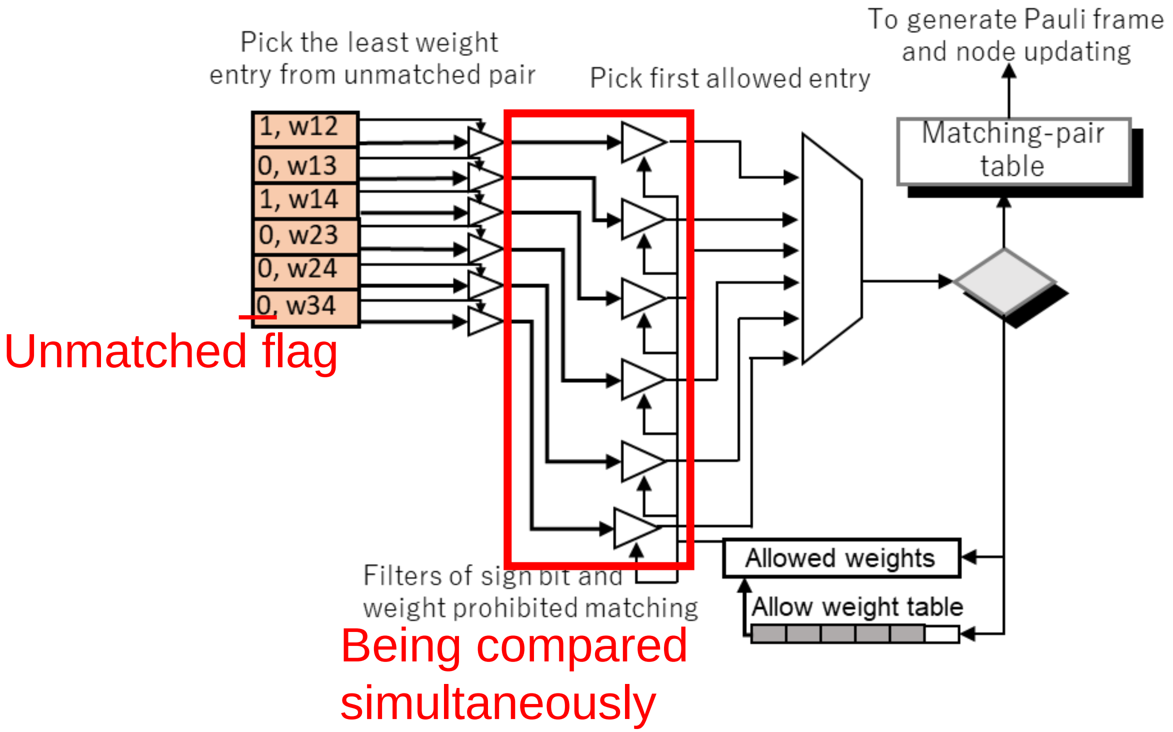
- Parallel greedy algorithm:
  - pick up the first pair of the current least weight @minimum of 1 clock
  - Turn off flags of pairs containing matched nodes
  - Pop-up pairs for Pauli frame generation in parallel

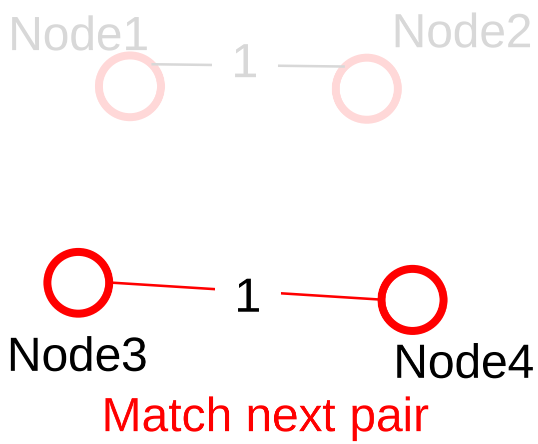




Parallel processing for matching

- Parallel greedy algorithm:
  - pick up the first pair of the current least weight @minimum of 1 clock
  - Turn off flags of pairs containing matched nodes
  - Pop-up pairs for Pauli frame generation in parallel

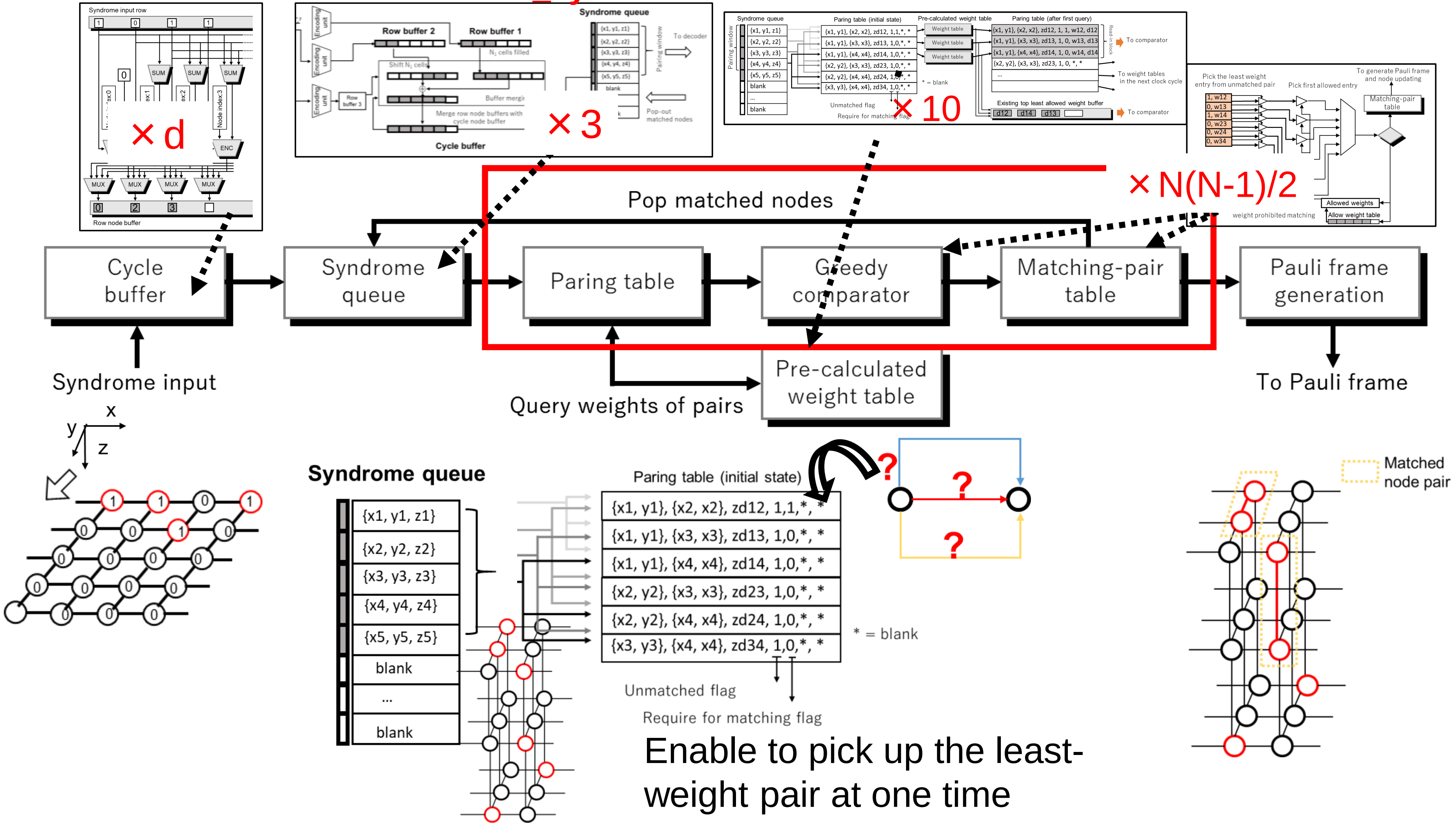




Overall HW design of WIT-Greedy for parallel processing

Parallel processing acceleration

(*d* is code distance, N = N_{active_syn} is paring window size)



Enable to pick up the least-weight pair at one time

Implementation result using FPGA

- Requirement of latency is satisfied
  - We have an average delay of 370 ns at d=11
- Scale of circuit implementation is reasonable

d	Decoding cylces			Buffer overflow times
	min	average	max	
5	18	19	71	0
7	19	22	97	0
9	24	29	105	0
11	33	37	103	0

Latency fits into 1µs time budget of code cycle

Condition of latency evaluation:

- RTL simulation
- Write random syndrome flipping testbench and monitor the delay
- Flip rate = 6*average error rate
- 10^4 random syndrome cycle sample

TABLE III
RESOURCE UTILIZATION OVERHEAD FOR DECODING LOGIC

d	Decoding entry node # N_{active_syn}	Buffer size of syndrome queue	LUT	FF	DSP	BRAM
5	9	15	5.5k	1.6k	10	2.5
7	12	21	11.0k	2.4k	10	10
9	15	29	16.2k	3.4k	10	35
11	20	39	22.6k	4.8k	10	126

Reasonable resource utilization of circuits for d=11

Condition of resource evaluation

- Queue overflow probability < 1e-15
- Clock constraint = 100 MHz
- placement&routing in FPGA (field programmable gate array, user reconfigurable circuits)

Conclusion

- Weighted decoding with non-uniformity is a necessity
- We made it with results of
  - Iterative greedy matching within average 370 ns
  - Support surface code up to d=11
  - Implementable with mid-class classical commercial device

TABLE I

COMPARISON OF HARDWARE IMPLEMENTATION OF SURFACE CODE DECODERS. OUR FRAMEWORK IS ABLE TO SCALE UP AND TAKE FABRICATION VARIANCE INTO CONSIDERATION.

Previous work

Small code only

No support weighted

Decoding algorithm	d_{max}	Latency	Measurement error?	Decoding window length	Target device	Weighted matching?	Scalability
Look-up table [15]	5	42 ns	Y	3	FPGA	Y	Hard
Neural network [16]	5	194 ns	N	-	FPGA	Y	Hard
Union-find [17]	11	325 ns	Y	11	FPGA	N	Easy
Greedy [18]	9	162.72 ps	N	-	SFQ	N	Easy
Greedy [19]	13	215 ps	Y	3	SFQ	N	Easy
Greedy (Our proposal)	11	370 ns	Y	11	FPGA	Y	Easy