Microsoft Azure

Decoding Hardware Requirements for Fault-Tolerant Quantum Computation

Nicolas Delfosse - Microsoft

Quantum Resource Estimation Workshop – May 30th, 2020

with Poulami Das, Chris Pattison, Bobbie Manne, Doug Carmean, Krysta Svore, Moin Qureshi

Quantum Physics

[Submitted on 18 Jan 2020]

A Scalable Decoder Micro-architecture for Fault-Tolerant Quantum Computing

Poulami Das, Christopher A. Pattison, Srilatha Manne, Douglas Carmean, Krysta Svore, Moinuddin Qureshi, Nicolas Delfosse

Quantum computation promises significant computational advantages over classical computation for some problems. However, quantum hardware suffers from much higher error rates than in classical hardware. As a result, extensive quantum error correction is required to execute a useful quantum algorithm. The decoder is a key component of the error correction scheme whose role is to identify errors faster than they accumulate in the quantum computer and that must be implemented with minimum hardware resources in order to scale to the regime of practical applications. In this work, we consider surface code error correction, which is the most popular family of error correcting codes for quantum computing, and we design a decoder micro-architecture for the Union-Find decoding algorithm. We propose a three-stage fully pipelined hardware implementation of the decoder that significantly speeds up the decoder. Then, we optimize the amount of decoding hardware required to perform error
resources between logical qubits, we obtain a
Moreover, we reduce the bandwidth required
Finally, we provide numerical evidence that ou
computer.

Our decoder is fast enough

Quantum Physics

[Submitted on 30 Jan 2020]

Hierarchical decoding to reduce hardware requirements for quantum computing

Nicolas Delfosse

Extensive quantum error correction is necessary in order to scale quantum hardware to the regime of practical applications. As a result, a significant amount of decoding hardware is necessary to process the colossal amount of data required to constantly detect and correct errors occurring over the millions of physical qubits driving the computation. The implementation of a recent highly optimized version of Shor's algorithm to factor a 2,048-bits integer would require more 7 TBit/s of bandwidth for the sole purpose of quantum error correction and up to 20,000 decoding units. To reduce the decoding hardware requirements, we propose a fault-tolerant quantum computing architecture based on surface codes with a cheap hard-decision decoder, the lazy decoder, combined with a sophisticated decoding unit that takes care of complex error configurations. Our design drops the decoding hardware requirements by several orders of magnitude assuming that good enough qubits are provided. Given qubits and quantum gates with a physical error rate $p = 10^{-4}$, the lazy decoder drops both the bandwidth requirements and the number of decoding units by a factor 50x. Provided very good qubits with error rate $p = 10^{-5}$, we obtain a 1,500x reduction in bandwidth and decoding hardware thanks to the lazy decoder. Finally, the lazy decoder can be used as a decoder accelerator. Our simulations show a 10x speed-up of the Union-Find decoder and a 50x speed-up of the Minimum Weight Perfect Matching decoder.

The lazy decoder can save 99.9% of the decoding hardware

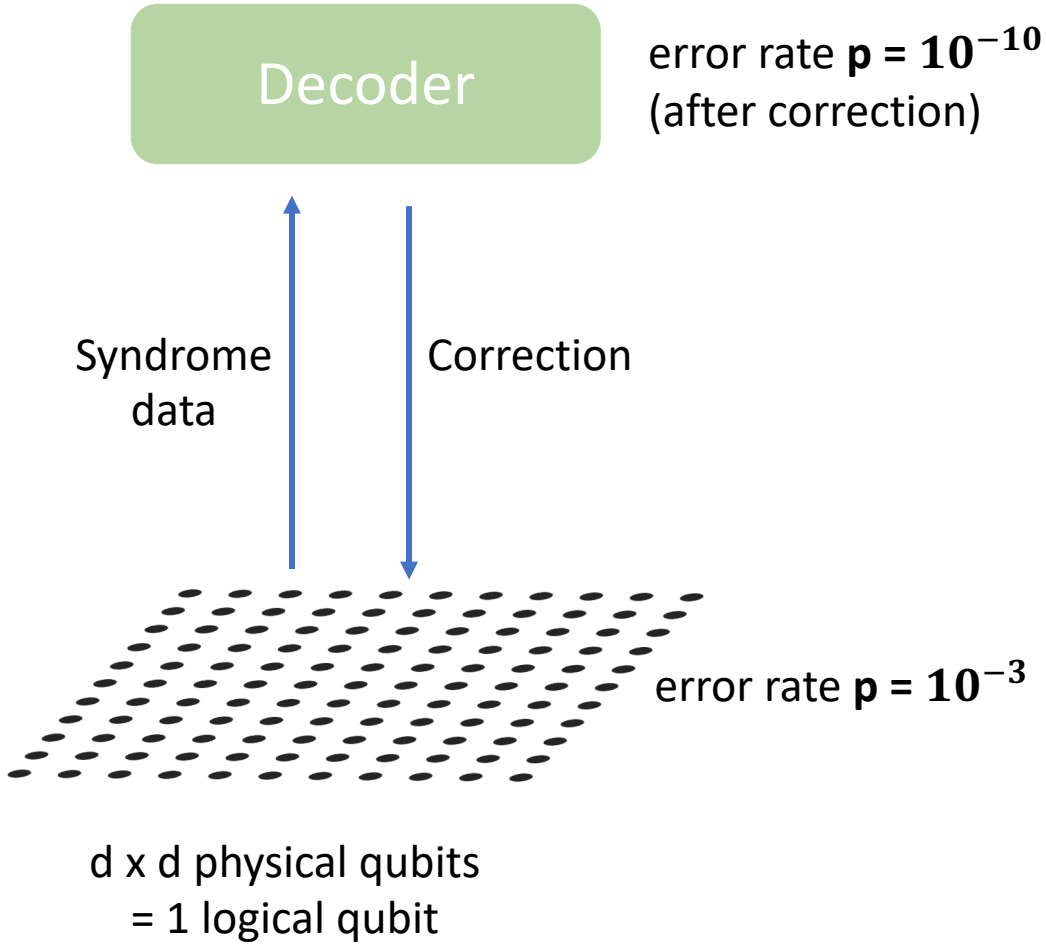
The Decoding Problem

Objective:

Design a decoder for **1,000 logical qubits**[1,2,3].

In this talk:

- Surface code[4,5]
- Qubit / gate error rate **p = 10^{-3}**



1. Gidney, Ekera (2020) https://arxiv.org/abs/1905.09749
2. Reiher et al. (2016) https://arxiv.org/abs/1605.03590
3. Campbell, Khurana, Montanaro https://arxiv.org/abs/1810.05582
4. Fowler et al. (2012) https://arxiv.org/abs/1208.0928
5. Litinski (2018) https://arxiv.org/pdf/1808.02892.pdf

Three Decoding Constaints

- **(A) Acuracy:** Identify the error with high probability.
- **(L) Latency:** Decoder runtime < 1 logical cycle
- **(S) Scalability:** Decode **1,000 logical qubits**[1] [2,3] with low hardware requirements.

Our Results:

- We design a decoder with **(A)(L)(S)**

Decoder	Acuracy	Latency	Scalability
LUT	Very High		
TN	Very high		
MWPM	High to Very high		
ML	High		
UF	High		

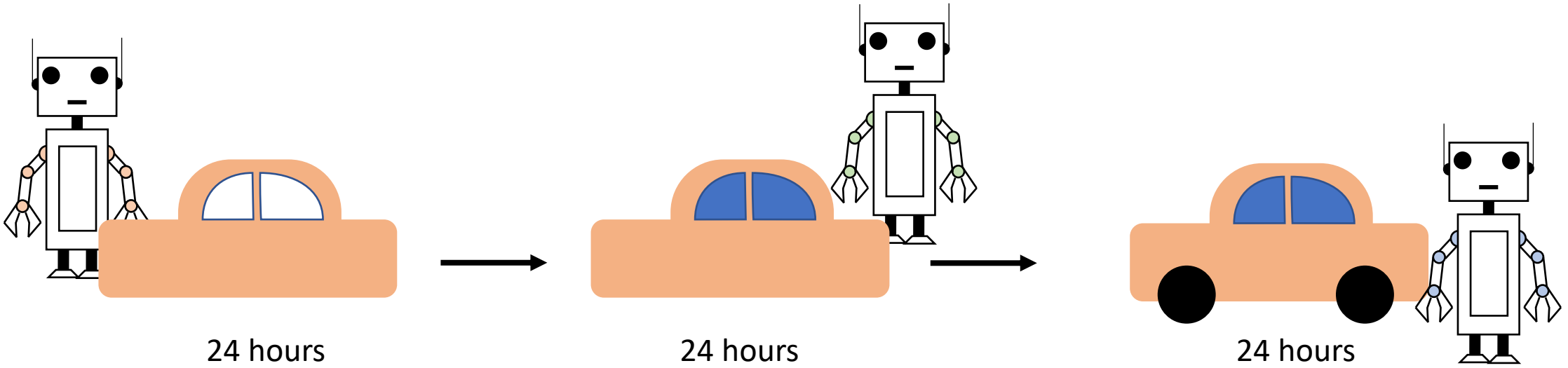
Too Slow

?

LUT: Tomita, Svore (2014) https://arxiv.org/abs/1404.3747
TN: Bravyi, Sushara, Vargo (2014) https://arxiv.org/abs/1405.4883
MWPM: Dennis, et al. (2001) https://arxiv.org/abs/quant-ph/0110143
ML: Torlai, Melko (2016) https://arxiv.org/pdf/1610.04238.pdf
UF: Delfosse, Nickerson (2017) https://arxiv.org/abs/1709.06218

Computer Architecture Toolbox

Pipelining

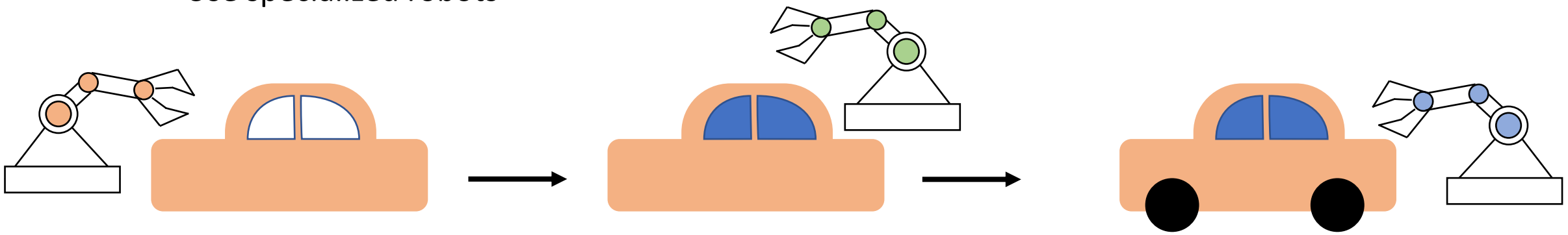


3 robots ⇒ 3x speedup:

- Building 1 car takes 3 days.
- Building 1,000 cars takes 1,003 days ≈ 1 day per car

Hardware Specialization

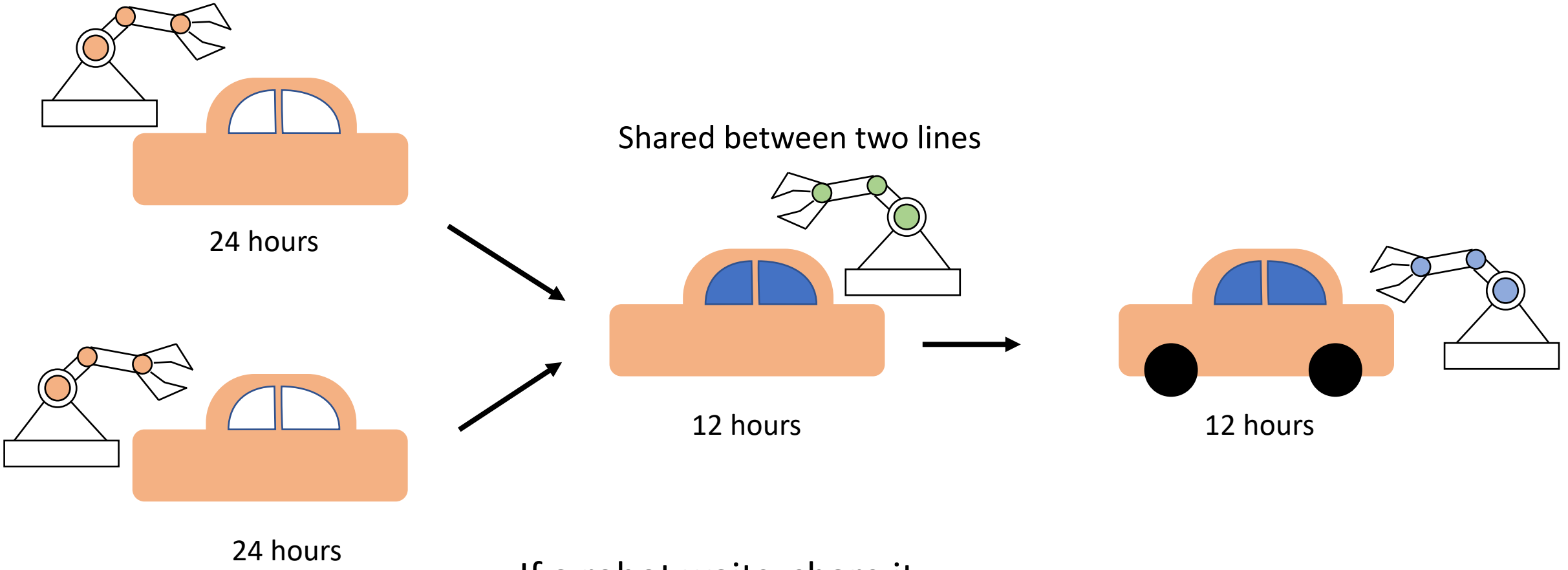
Use specialized robots



Specialized robots:

- **Faster processing** using precomputed movements.
- **Energy efficient.**

Ressource Sharing



If a robot waits, share it:

- **Save hardware** (# robots)
- **Save energy** (waiting robots consume energy)

Hardware Accelerator for the Union-Find Decoder

The Union-Find Decoder[1]

Why the Union-Find decoder?

- **Fast**: Complexity $O(n\,\alpha(n))$ with $\alpha(n) < 5$ for all practical applications*
- **Performant**: Correct any set of (d-1)/2 faults
- **Flexible**: works for surface codes and color codes on any lattice (even hyperbolic)

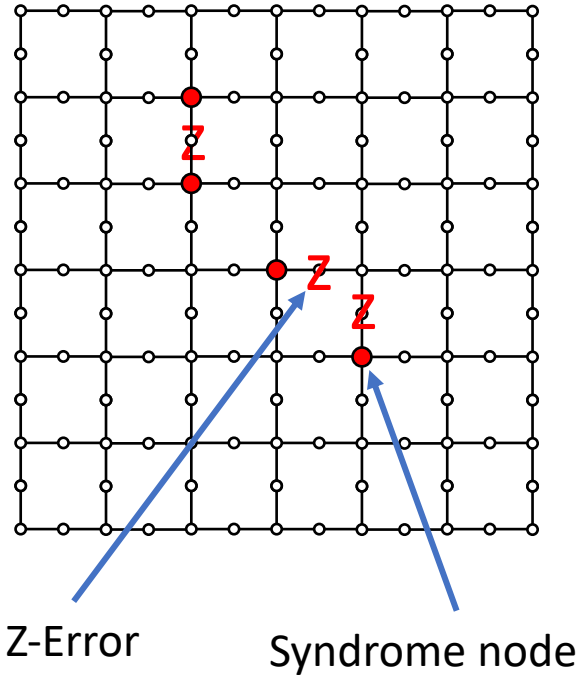
But also:

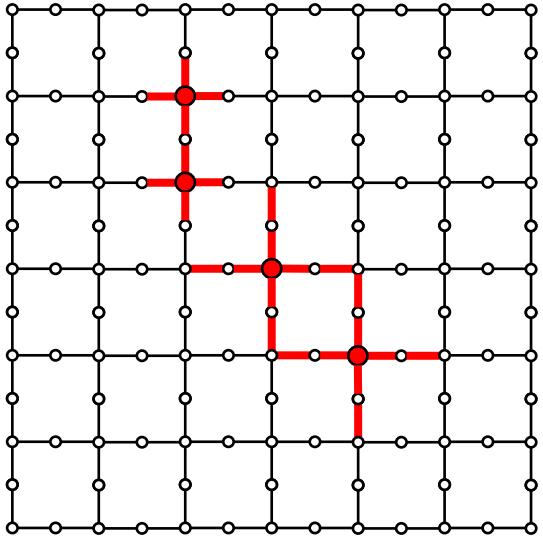
- **Simple**

* $\alpha(n) < 5$ if n is smaller than the number of atoms in the universe (10^{80})

1. UF: Delfosse, Nickerson (2017) https://arxiv.org/abs/1709.06218

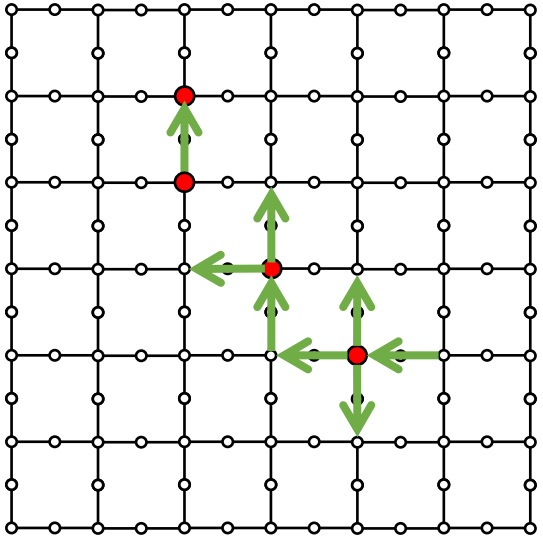
The Union-Find Decoder[1]





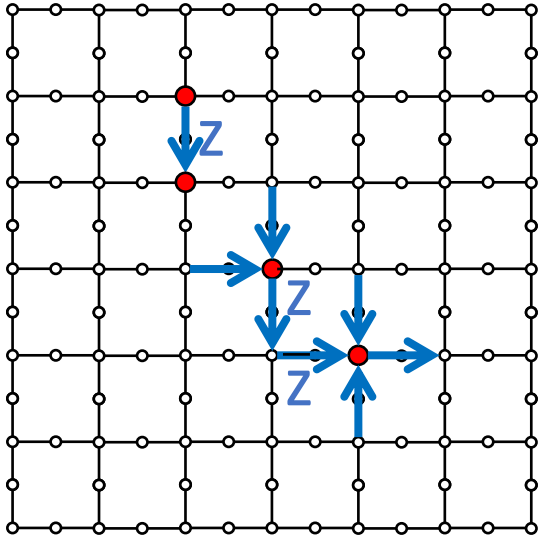
Step 1: Graph Generator

- Grow clusters around syndrome
- Stop when the cluster contains an even number of syndrome



Step 2: DFS

- Build a spanning tree for each cluster

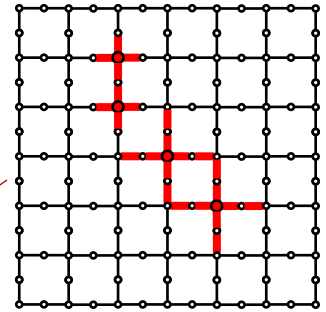


Step 3: Correction

- Reverse the spanning tree to correct

1. UF: Delfosse, Nickerson (2017) https://arxiv.org/abs/1709.06218

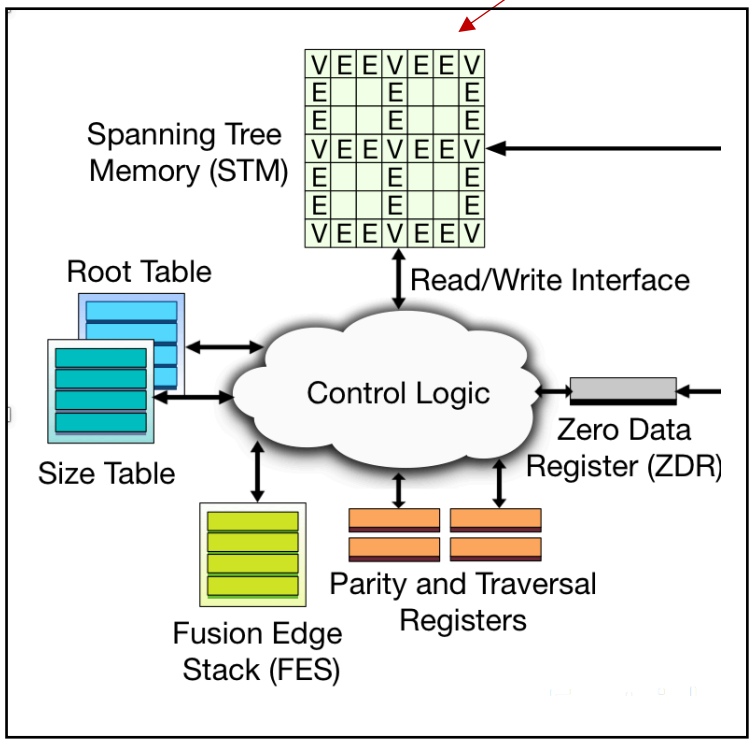
Hardware Unit for the Graph-Generator



Store and grow
Clusters in the STM

Memory requirement for 1,000 logical qubits:

Design Component	Baseline	Optimized Design	Savings
STM (Gr-Gen)	1.97 MB	0.99 MB	(2X)
Root Table (Gr-Gen)	3.17 MB	0.79 MB	(4X)
Size Table (Gr-Gen)	3.46 MB	0.87 MB	(4X)
Stacks (DFS Engine)	1.35 MB	0.34 MB	(4X)
Total	9.96	2.81	(3.5X)



Hardware accelerator

Hardware acceleration:

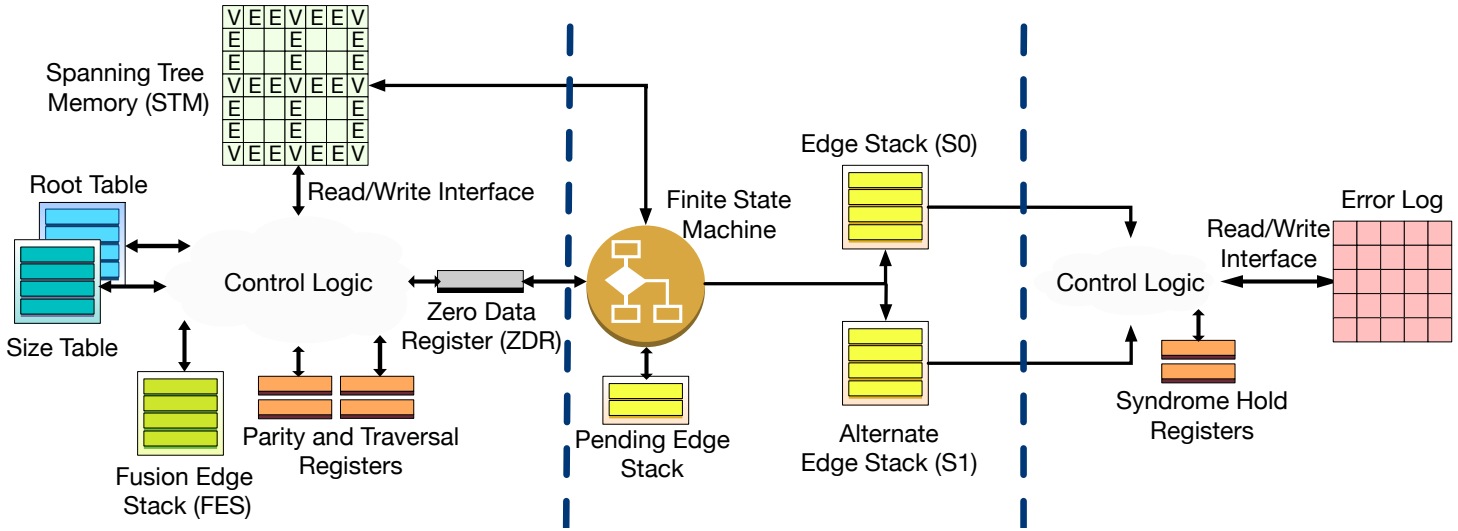
- Memory read without fetching from off-chip memory
- Speedup by pipelining

Pipepline speedup:

- Consider 4 clusters C_1, C_2, C_3, C_4 through DFS and Cor:

Time step	DFS	Cor
1	C_1	
2	C_2	C_1
3	C_3	C_2
4	C_4	C_3
5		C_4

5 steps instead of 8

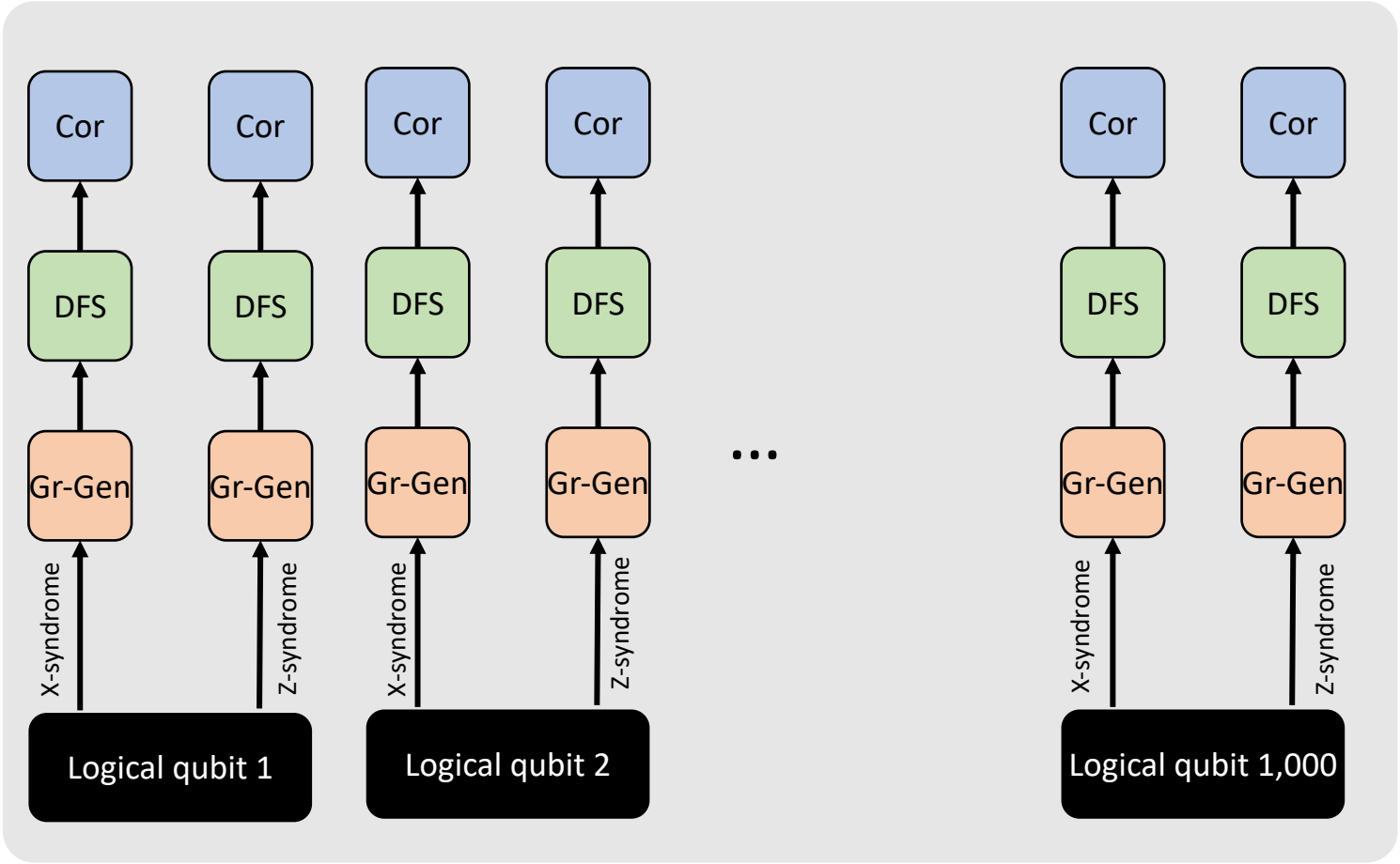


Resource Optimization

Baseline design

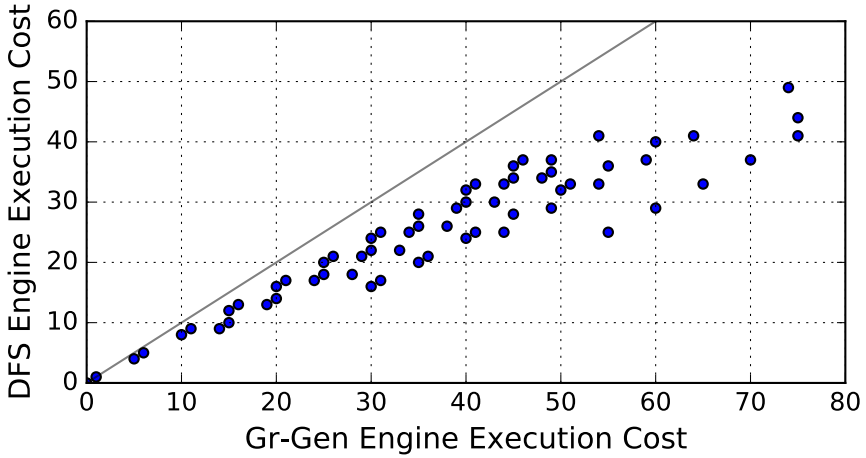
For 1,000 logical qubits:

- 2,000 Gr-Gen units
- 2,000 DFS units
- 2,000 Cor units



Pipeline Optimization

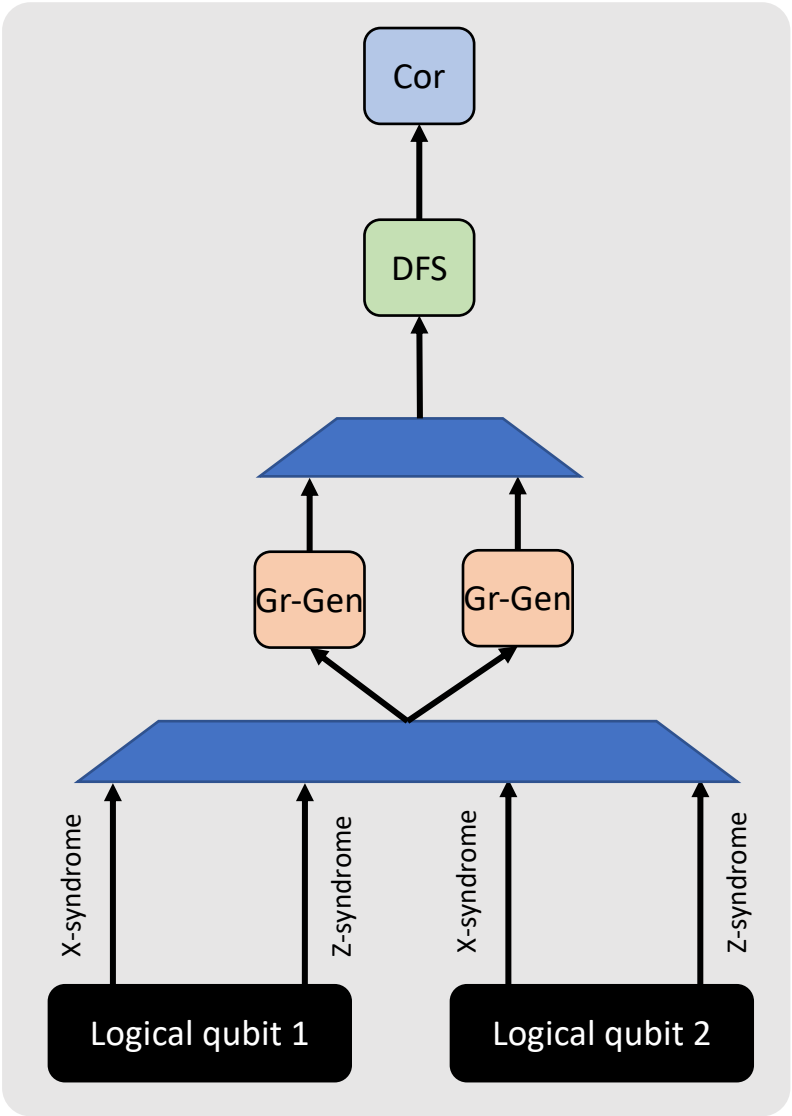
Estimate runtimes by monte-carlo simulation:



- Gr-Gen is twice slower than DFS
- DFS and Cor have similar runtime

⇒

(2,1,1)-pipeline



Hardware saving:

- 50 % of Gr-Gen
- 75 % of DFS
- 75% of Cor
- 70 % of total memory

Results for of d=11 surface code with $p=10^{-3}$

Is the decoder fast enough?

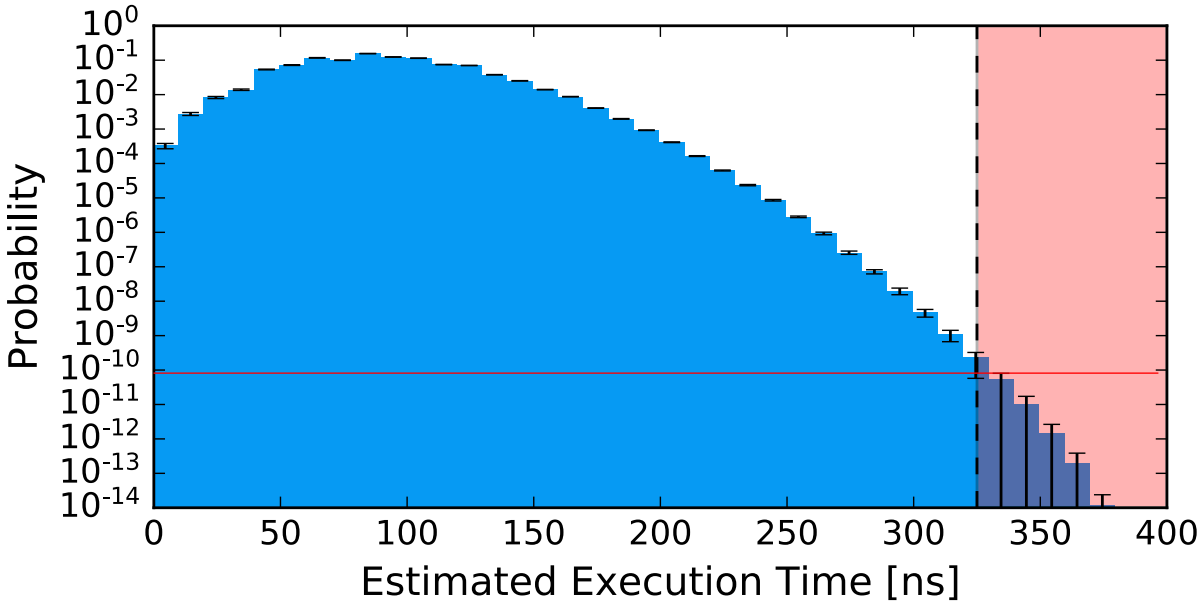
Decoding time:

- **(L)** We must have decoding time < 11 μs.
- We obtain max decoding time = **325 ns.**

Conclusion:

- **Our decoder is fast enough.**
- **Even with shared resources.**

with (2,1,1)-pipeline.



Runtime Estimation Model:

- Count reads
- 4GHz frequency and 4 cycles per 32-bit read
- Assume our cluster model
- Ignore write, some latency, queue processing

Results for of d=11 surface code with $p=10^{-3}$

The lazy decoder

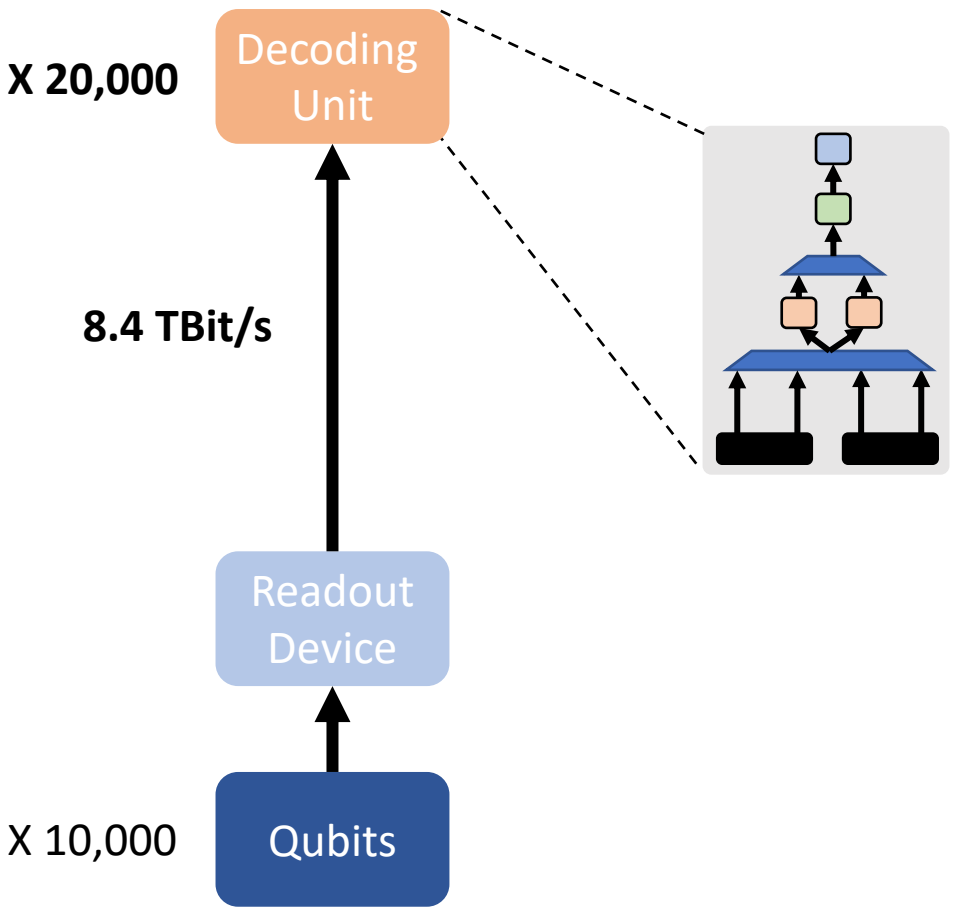
Requirements for RSA 2048 factorization

Resource estimation by Gidney and Ekera[1]:

- 20 millions qubits with error rate 10^{-3}
- 10,000 logical qubits
- 8 hours

But decoding requirements are colossal[2]:

- **8.4 TBit/s** of bandwidth
- **20,000** decoding units



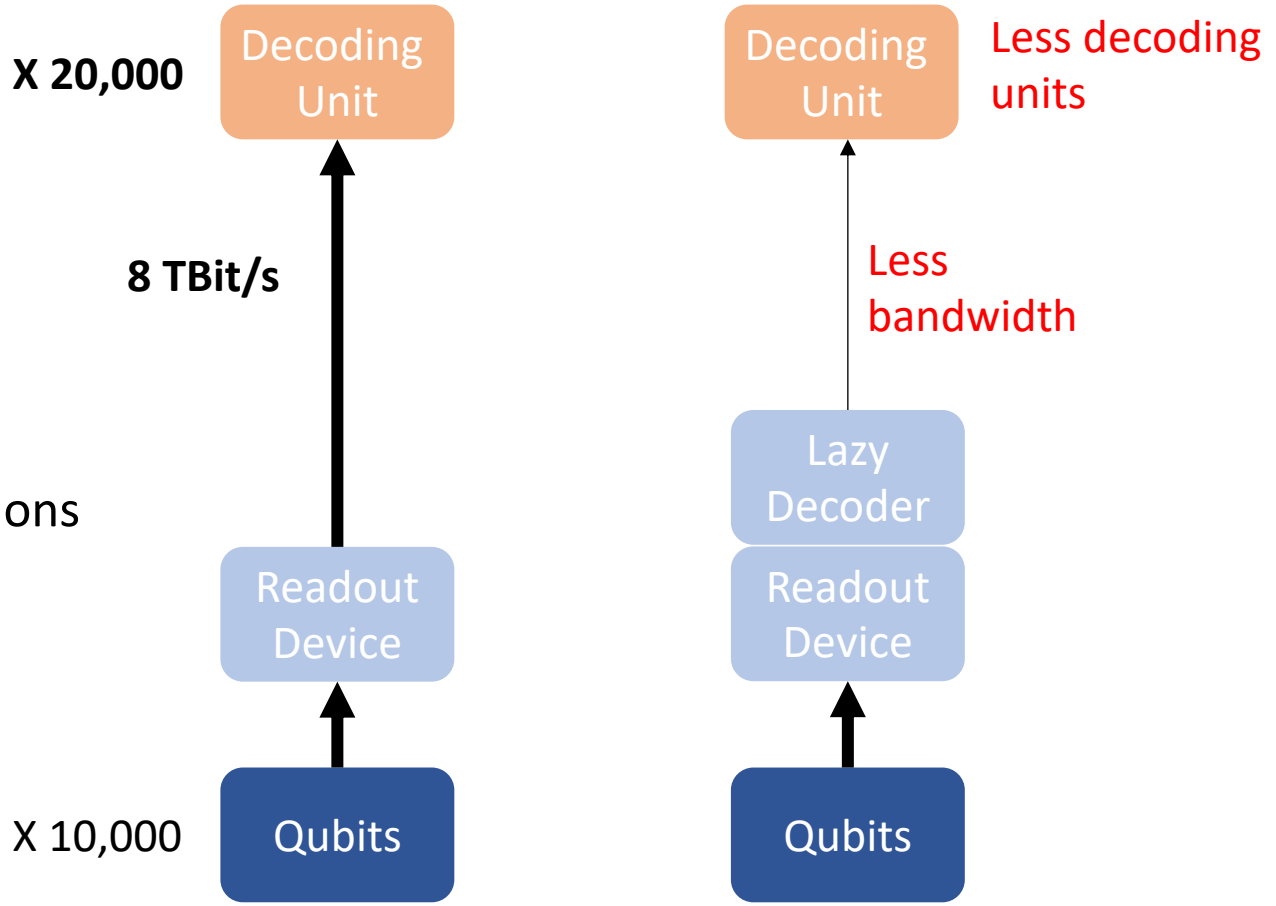
1. Gidney, Ekera (2020) https://arxiv.org/abs/1905.09749
2. Delfosse (2020) https://arxiv.org/abs/2001.11427

Data to reach pLog = 10^{-15}

The lazy decoder[1]

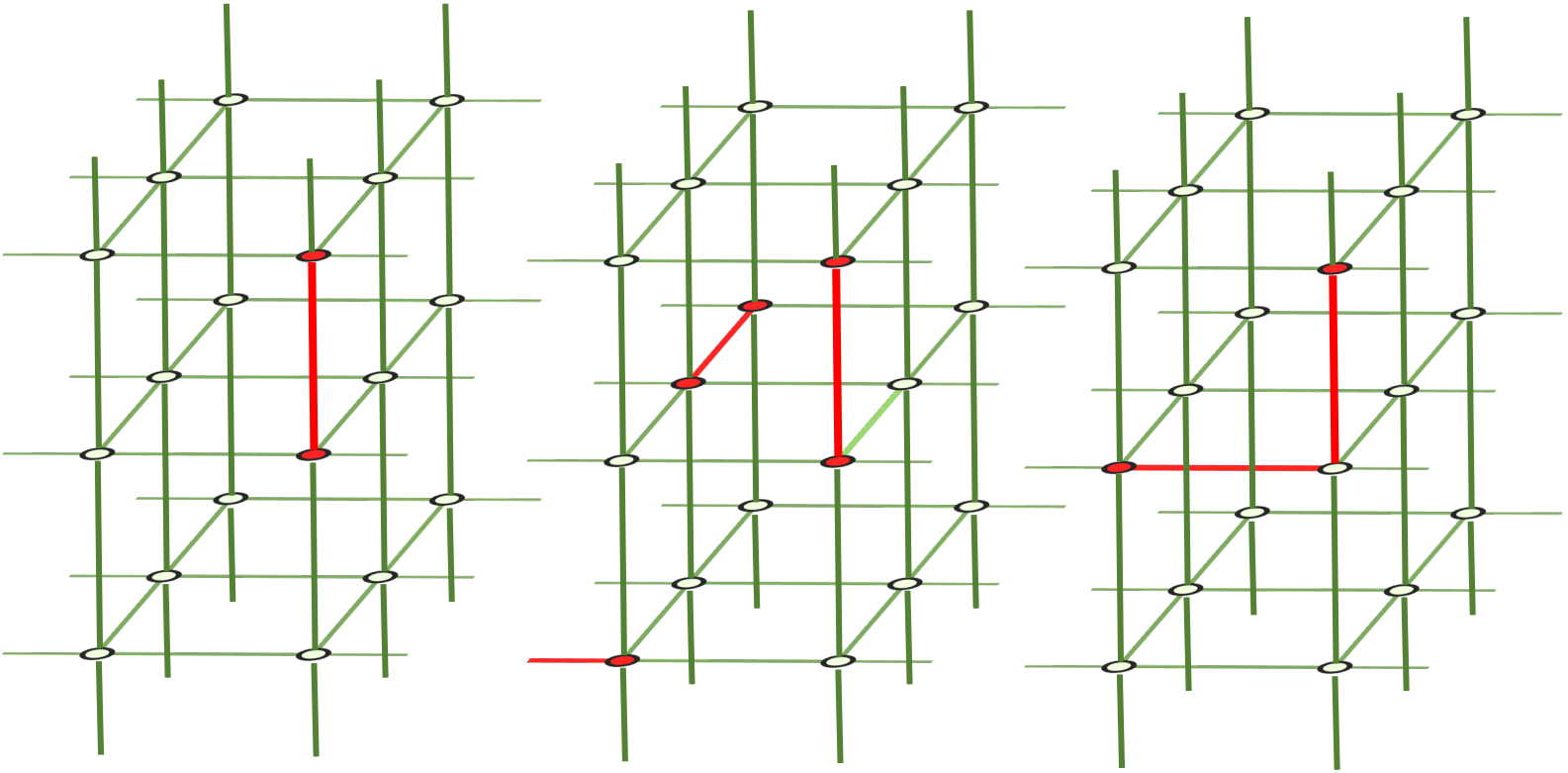
Lazy decoder:

- It is a pre-decoder
- It only corrects easy error configurations



1. Delfosse (2020) https://arxiv.org/abs/2001.11427

The Lazy Decoder



success

success

failure

Basic idea:

- Try to pair each syndrome node with a neighbor
- If not possible, abort

Main feature:

- If the lazy decoder succeeds, the correction retuned is guaranteed to be optimal.

Lazy decoding algorithm

Input: Set S of highlighted syndrome nodes
Output: Either a set E of edges pairing s or **failure.**

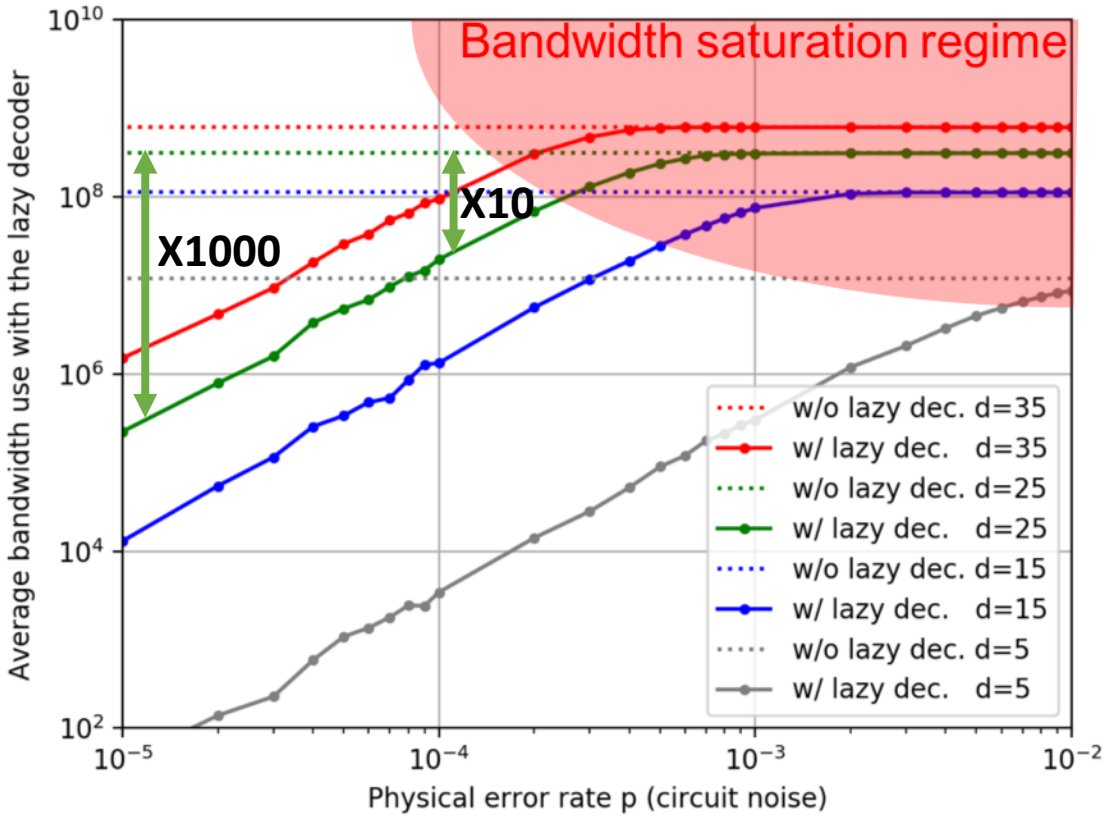
1. Initialize $\mathrm{E} = \emptyset$.
2. Loop over highligted nodes $v \in S$ and do:
3. If v has a neighbor u in S, add $\{u, v\}$ to E
4. Else if v is a boundary add the half edge $\{u, -\}$ to E
5. Else return **failure**
6. If the number of ambiguous pairing is > 1 return **failure**

Fully local
Easy to parallelize
Easy hardware implementation

The only global data
Easy to compute

Ambiguous: Vertex paired to a boundary but could have been paired to a neighbor.

Average bandwidth use per logical qubit



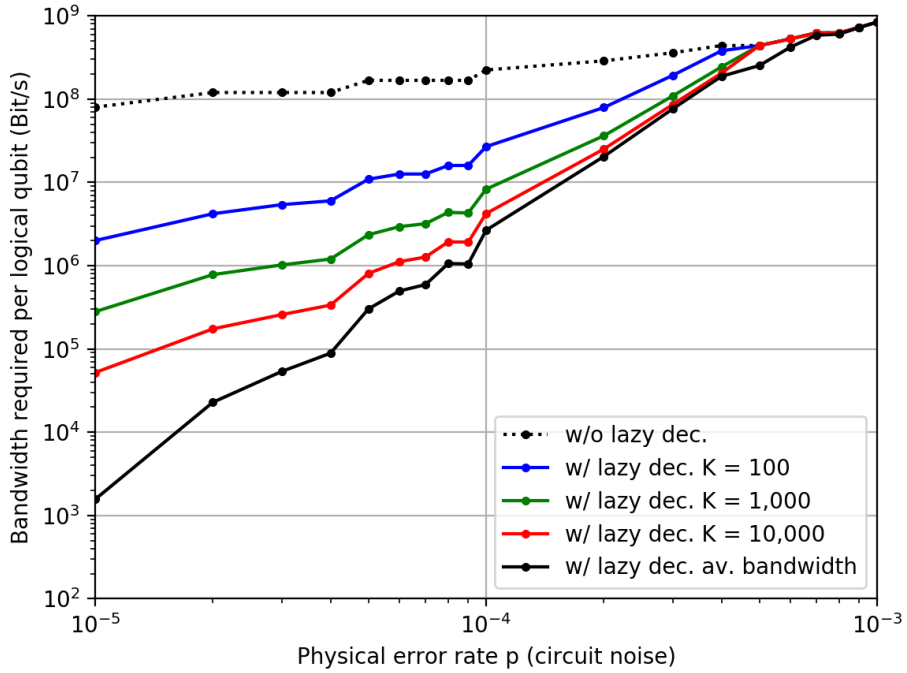
Bandwidth saturation:

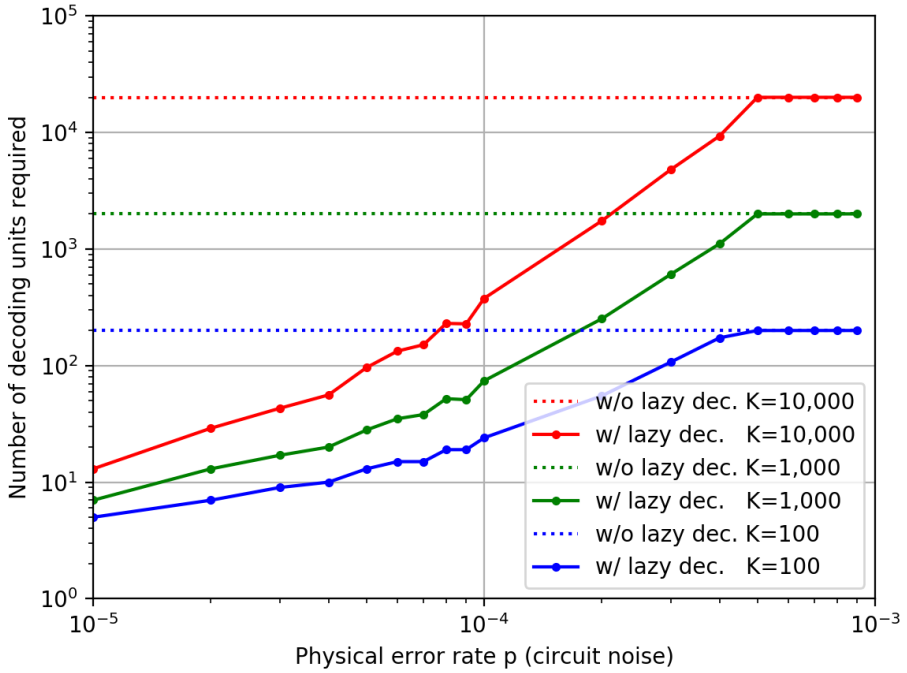
- No improvement with Lazy decoder
- Qubits and gates are too noisy

Lazy decoder saving:

- Large saving for $p < 10^{-4}$

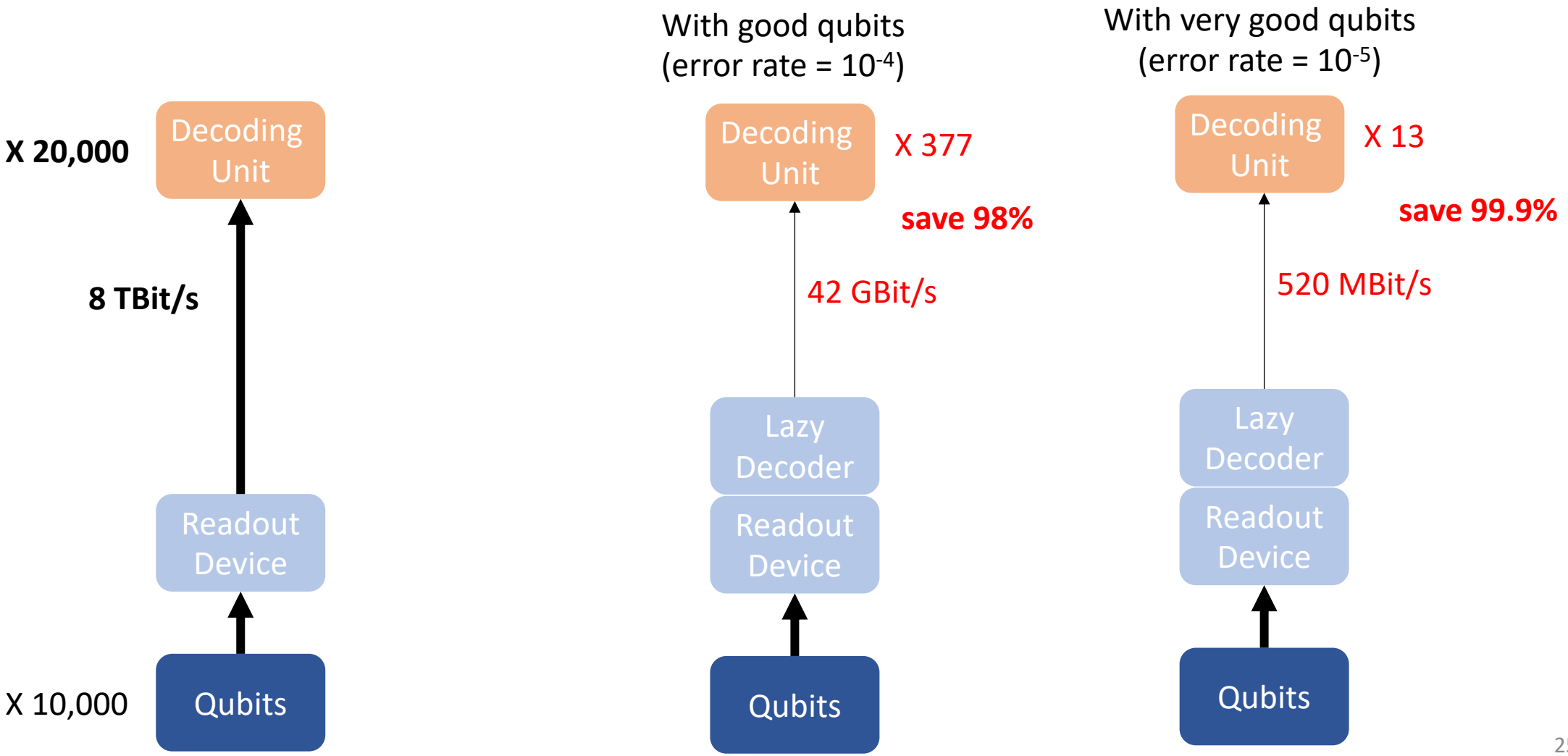
Hardware required to reach $p_{Log} = 10^{-12}$





- More qubits ⟹ Less bandwidth / decoding units per qubit

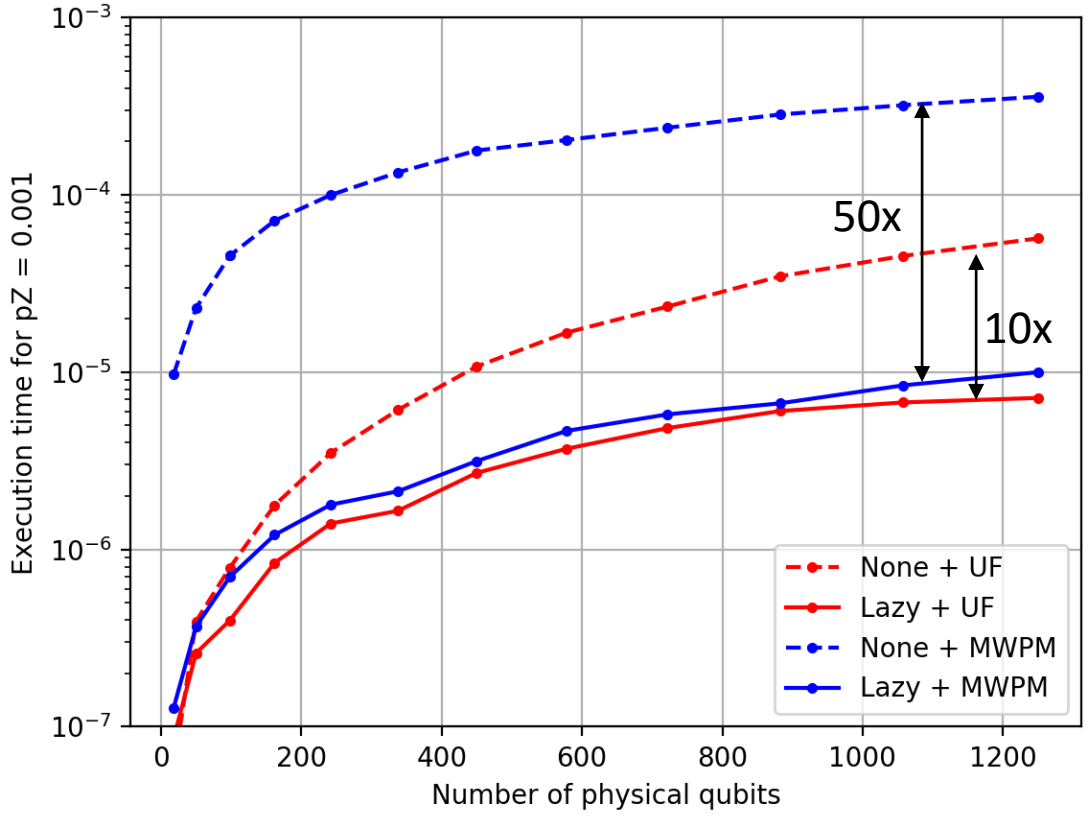
Back to RSA 2048 factorization



The lazy decoder as a decoder accelerator

The lazy decoder:

- Speeds up any decoding algorithm
- Without deteriorating the performance



Conclusion

We design a decoder that is:

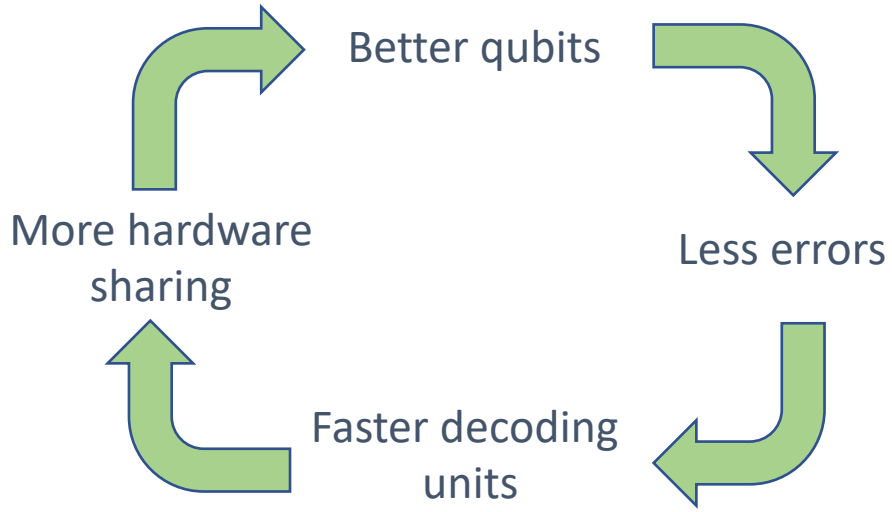
- Acurate
- **Fast enough**
- **Scalable**

Observations:

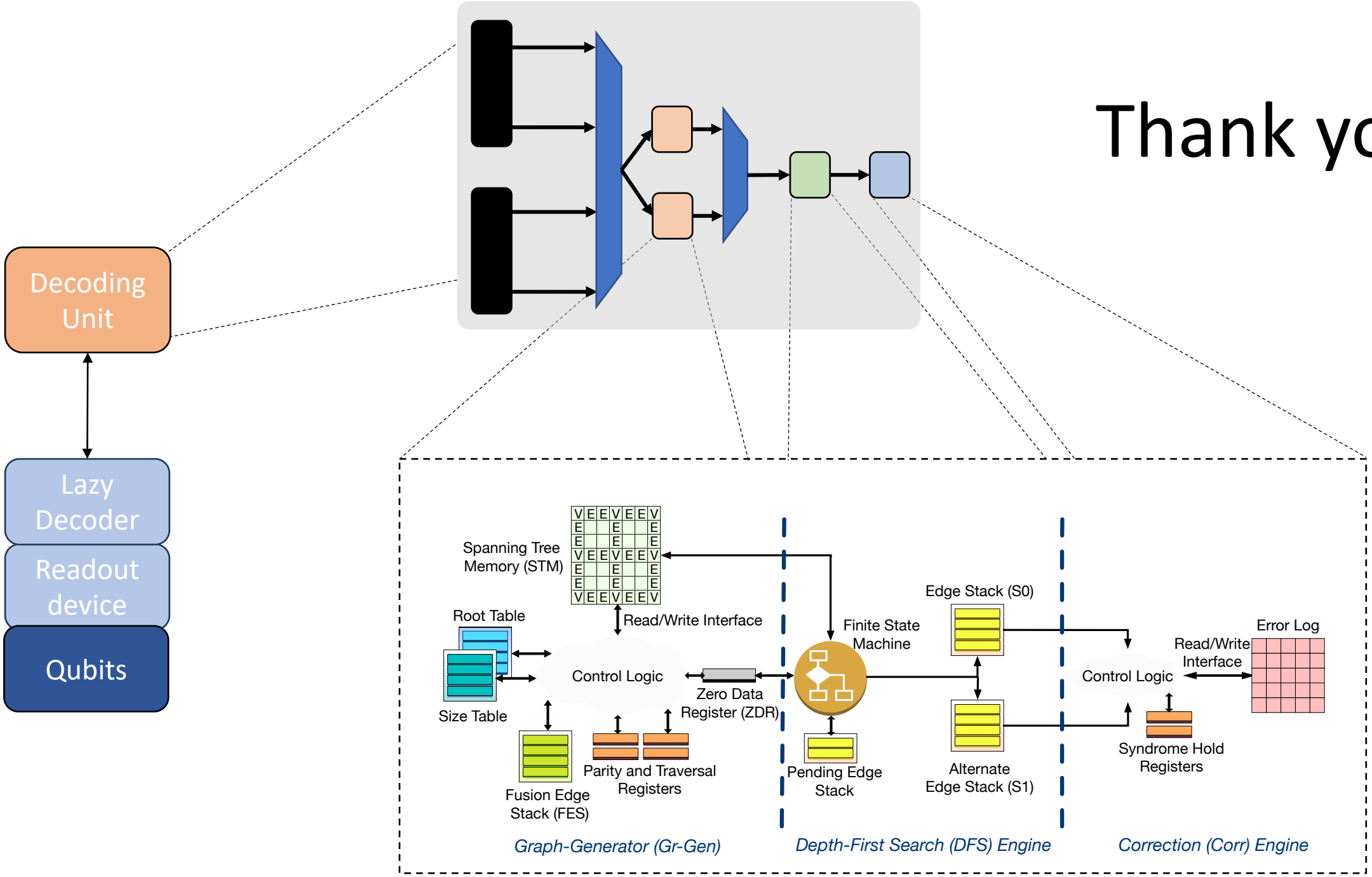
- Current qubits are not good enough for scalability.
- **We need better qubits**: aim at gate error rate $p < 10^{-4}$

Future directions:

- Explore more precise noise models
- Adapte our micro-architecture to the recent version of the UF decoder of Huang et al. (arxiv:2004.04693)



Thank you!


Decoding Unit
Lazy Decoder
Readout device
Qubits
Spanning Tree Memory (STM)
Read/Write Interface
Root Table
Size Table
Control Logic
Zero Data Register (ZDR)
Fusion Edge Stack (FES)
Parity and Traversal Registers
Finite State Machine
Pending Edge Stack
Edge Stack (S0)
Alternate Edge Stack (S1)
Control Logic
Read/Write Interface
Error Log
Syndrome Hold Registers
Graph-Generator (Gr-Gen)
Depth-First Search (DFS) Engine
Correction (Corr) Engine