COE 530 Final Exam Study Guide

Quantum Computers and Architecture — Term 252

Lecture 12 — Circuit Optimization

Compilation Flow & Two Optimization Levels

- **Generic compilation flow:** Quantum algorithm → programming language → high-level program → automated compilation (resolve syntax, function cloning, loop unrolling, gate decomposition, arithmetic synthesis, scheduling, memory management, error correction, dynamic compilation) constrained by qubit connectivity, coherence, gate error, parallelism → classical+quantum coprocessor.
- **Execution pipeline:** Problem → algorithm → circuit → **Optimizer Level I** → simulator → mapping → **Optimizer Level II** → execute on QPU.
  - **Level I (hardware-independent):** gate cancellation, fusion, depth reduction (pre-simulation).
  - **Level II (hardware-aware):** qubit mapping to physical layout, SWAP insertion (post-simulation).

Metrics & Objectives

- **Metrics:** circuit depth, gate count, qubit count, two-qubit gate count (high error), gate fidelity, parallelism.
- **Two objectives:** (1) **Circuit Simplification** — minimize gate count & depth; (2) **Circuit Layout Optimization** — rearrange execution order & qubit mapping while preserving correctness.

Core Methodologies for Simplification

- **Pattern Matching:** Replace common sub-structures with optimized equivalents using DAGs (gates=vertices, dependencies=edges).
- **Unitary Synthesis:** Decompose complex unitaries into smaller pieces via ML / algorithmic approaches.

Equivalent Quantum Circuit Rules

- **Rule I — Null gates:** $HH = I$, $ZZ = I$; CNOT with $|+\rangle$ target → identity; controlled-U with $|0\rangle$ control → identity.
- **Rule II — Control Reversal:** Direction of CZ swappable; CNOT = $H \cdot CZ \cdot H$ on target; CNOT reversal via H on both wires.
- **Rule III — Quantum-Classical Substitution:** Replace a CNOT before measurement with classical XOR of measurement outcomes that conditions a later U.
- **Rule V — Distributed CNOT** and **Parallel CNOT Gates:** rewrite a 3-qubit CNOT pattern into an equivalent sequence using additional CNOTs.
- **Rule VI — CNOT Mirror:** mirror identity over a CNOT gate.

Commuting Gates

- Gates G_1, G_2 commute iff $U_1U_2|\phi\rangle = U_2U_1|\phi\rangle$. Rearranging enables parallelism + cancellations. Examples: control-R_z commutes with another control on same qubit; X on target commutes with CNOT target.

Gate Cancellation

- Self-inverse (real unitary, $U \cdot U = I$): NOT, Z, CNOT, SWAP, CCNOT (Toffoli), CSWAP. **Not self-inverse:** T, S.

- Implemented programmatically using a DAG; two adjacent identical real-unitary nodes cancel.

Specific Reductions

- **Hadamard reduction:** $H\,\text{CNOT}\,H = \text{CZ}$ pattern; $H\,P^\dagger(\pi/2)\,P(\pi/2)\,H = P(\pi/2)\,P^\dagger(\pi/2)$.
- **Rz merging:** $Rz(\theta_1) \cdot Rz(\theta_2) = Rz(\theta_1 + \theta_2)$; reduces gate count & depth without changing behavior.
- **Linear reversible circuits** (CNOT-only / stabilizer): $E_{ij} = I + e_{ij}$. Two adjacent $E(c_k, t_k), E(c_{k+1}, t_{k+1})$ can swap iff $c_k \neq t_{k+1}$ and $c_{k+1} \neq t_k$.
- **Ancilla-assisted depth reduction:** Precompute & store phase shifts on ancillas applied in parallel; QFT depth $O(n^2) \to O(n \log n)$. Trade-off: more qubits $\Rightarrow$ more NISQ noise.

Peephole Optimization (Re-writing)

- Local rewrites within a sliding window of 2–5 gates; iterative until fixed-point; rule-based ($HH = I$, CNOT $\cdot$ CNOT $= I$). Analogous to classical compilers.
- **Common rules:** (1) self-inverse cancellation; (2) rotation merging $Rz(\theta_1) \cdot Rz(\theta_2) = Rz(\theta_1 + \theta_2)$ (two T's $\to$ one S; if sum $= 2\pi$ eliminate); (3) gate substitution (H; CNOT; $H \to CZ$; X; CNOT; $X \to$ CNOT with swapped control/target).
- **Worked example:** H; CNOT; H; $Rz(\pi/4)$; $Rz(\pi/4)$; X; $X \Rightarrow CZ$; $Rz(\pi/2)$. 7 gates $\to$ 2, depth 7 $\to$ 2.

Loop Optimization

- Quantum algorithms (Grover iterations, Trotter steps, variational ansatze) repeat sub-circuits. Savings multiply by iteration count; critical for NISQ.
- **Loop Unrolling:** Expand k iterations $\to$ cross-iteration peepholes. *Grover diffusion:* $H \cdots H \cdot H \cdots H$ adjacent at boundaries cancel $HH = I$, saves $2n$ gates per boundary. *Trotter:* k separate $Rz(\theta/k)$ merge into one $Rz(\theta)$.
- **Loop-Invariant Code Motion (LICM):** Hoist iteration-independent gates outside loop (must commute with body). *Variational ansatz with basis change:* hoist H outside, $H \cdot Rz \cdot H = Rx$; saves $2(k-1)$ Hadamards. *State prep inside Trotter:* hoist X on uninvolved qubit.
- **Boundary optimization:** cancel gates at iteration boundaries.

Lecture 13 — Qubit Mapping and Reuse

The Mapping Problem

- Circuits assume all-to-all connectivity; real NISQ hardware is sparse. Two-qubit gates require physical adjacency.
- **Objective:** initial mapping logical $\to$ physical + SWAP insertion that minimizes SWAP count (each SWAP = 3 CNOTs). Poor mapping can grow gates 2–10×.

QPU Topologies

- **Grid/square lattice:** Google Sycamore (~4 neighbors).
- **Heavy-Hex:** IBM Eagle/Heron (degree-3, sparse).
- **Trapped-Ion:** dynamically reconfigurable zones; all-to-all within zone; shuttling between.
- **Neutral atom:** blockade radius defines connectivity; atoms rearranged via optical tweezers.

Two Graphs

	Device Connectivity Graph	Program Interaction Graph
Represents	Physical HW topology	Circuit's qubit-pair interactions
Nodes	Physical qubits	Logical qubits
Edges	Directly executable 2Q gates	Required 2Q interactions
Nature	Fixed per device	Varies per program
Typically	Sparse	Can be dense

Routing & Cost

- **Process:** pick next 2Q gate → if mapped physical qubits not adjacent, SWAP along shortest path → execute and update mapping.
- **Cost model:** weighted edges on device graph; longer SWAP chains = more error & congestion. Cost components: edge distance (latency), edge-density uniformity, edge-crossings minimization.
- **Hardware-specific routing:** superconducting → SWAP chain; trapped ion → shuttling; neutral atom → optical-tweezer rearrangement.

Formalization & Hardness

- **Initial mapping π:** injective $L \to P$. **Dynamic mapping:** π updates as SWAPs permute.
- **Coupling graph $G_p = (V, E)$:** V physical qubits, E allowed 2Q interactions.
- **QMP is NP-complete** (reduces to subgraph isomorphism). Optimal initial mapping alone is NP-hard.
- **Exact (SAT/SMT):** optimal but only ~20–30 qubits. **Heuristics (SABRE):** scale to 1000+ qubits, no optimality guarantee.

DAG of Quantum Circuit

- Nodes = gates, edges = data deps (qubit wires), direction = temporal order.
- **Front layer:** gates with no unresolved deps, ready to execute.
- **Look-ahead:** routing inspects future layers; **critical path:** longest chain = min depth.
- Single-qubit gates often **neglected** in DAGs (don't add cross-qubit dependencies). Two-qubit gate executes when all prior 2Q gates on its qubits done. Complexity $O(g)$.

SABRE — SWAP-based Bidirectional Heuristic

- Qiskit default routing heuristic.
- **Look-ahead heuristic:** score SWAPs by distance reduction in front layer + upcoming gates.
- **Decay effect:** penalize frequently swapped qubits to spread parallelism.
- **Reverse traversal:** run mapper backward through reversed DAG to refine initial placement; iterate forward/backward to converge.
- Polynomial-time, scales to 1000+ qubits.

Heuristics vs Exact

- **SABRE:** greedy stochastic, polynomial, near-optimal, no guarantee.
- **SMT/SAT:** provably optimal, exponential worst-case, ~20–30 qubits.
- **Hybrid:** SMT on critical subcircuits, SABRE on the rest.

Pre-Mapping Optimizations

- **Assertion-based:** use pre/post conditions to delete redundant gates (e.g., SWAP on qubits known equal).
- **Gate merging:** combine consecutive single-qubit rotations Rx, Ry, Rz into one unitary; helps variational (VQE, QAOA).
- **Subroutine boundary optimization:** cross-boundary opts shrink qubit width → less constrained mapping; trade-off: breaks modularity.

Qubit Reuse

- After measurement / when no longer needed, reset qubit to $|0\rangle$ and reuse. Benefits: fewer physical qubits, reduced decoherence (less idle time), enables larger algorithms beyond device width.

Uncomputation

- Reverses an intermediate computation to return ancillas to $|0\rangle$ so they can be safely discarded/reused. Critical for coherence (no leftover entanglement). Analog to garbage collection.
- **Three steps:** (1) Compute U in ancilla, (2) Use result with gate V on output, (3) Uncompute with $U^\dagger$.

- Possible because gates are unitary (reversible): $U^\dagger U = I$.

Lecture 14 — Gate Scheduling and Resource Allocation

Why Scheduling Matters

- **NISQ:** decoherence budget tight; idle cycles cost fidelity; depth must be minimized.
- **Fault-tolerant:** T-gates and magic states dominate cost; depth = wall-clock & qubit-seconds.
- Both regimes share abstractions: circuit DAG, gate set, hardware graph.

Data Dependency Graph (DDG)

- Schedule = sequence of gate ops on logical qubits; g_2 depends on g_1 if share a qubit and g_2 comes later.
- **DDG:** $G = (V, E)$, V=gates, edge (v_i, v_j) if v_j depends on v_i.
- **Build:** node per gate, scan each qubit L-to-R, add edge between consecutive gates on same qubit. Resulting DAG captures partial ordering.
- **Metrics:** critical path = longest path = logical depth; width = qubits = max parallelism; gate count = total nodes.

Scheduling Considerations

- **Commutator** $[A, B] = AB - BA$; if 0, gates commute (swappable).
- **Conjugate gates:** if $UAU^\dagger = B$ then $UA = BU$.
- **Common commutations:** two CNOTs sharing only the control qubit commute; Z commutes through CNOT control; diagonal gates on same qubit commute.
- **Commutation-aware DAG reduction:** gate merging (adjacent unitaries combined), inverse pair cancellation, template-based rewriting.
- **Commutation-relaxed DDG (CRDDG):** weakens DDG edges where gates commute, enabling more reordering.
- **Hardware:** 2Q gates only on adjacent qubits (else SWAP); crosstalk increases error with simultaneous nearby gates; scheduler balances parallelism vs noise.

Goals

- **Circuit depth:** minimize longest critical path; if depth $<$ qubit lifetime, safe from decoherence.
- **Space-time volume:** $V = \sum_{q \in Q} \sum_{(t_i, t_f) \in T_q} (t_f - t_i)$ — active usage of each qubit; sum of intervals between first and last gate per qubit. Parallelism reduces both depth and V.

ASAP vs ALAP

- **ASAP (As Soon As Possible):** schedule each gate at earliest time all deps met; greedy forward; minimizes start times; preferred when reducing idle time matters.
- **ALAP (As Late As Possible):** schedule at latest time without delaying successors; backward from outputs; defers ops to reduce qubit lifetime.
- ASAP and ALAP have the same depth; differ in slack distribution.

Slack & Critical Gates

- $\text{slack}(g) = \text{ALAP}(g) - \text{ASAP}(g) \geq 0$.
- slack $= 0 \Rightarrow$ **critical**; critical path = all slack-0 gates.
- Slack $> 0 \Rightarrow$ schedule anywhere in $[\text{ASAP}, \text{ALAP}]$ without changing depth $\Rightarrow$ secondary objectives (crosstalk, lifetime).

Other Scheduling Concepts

- **List scheduling:** ready list, priority heuristics (critical-path length, mobility/slack, gate type); pick highest-priority ready gate.

- **Depth minimization** = constraint satisfaction; related to **classical job-shop scheduling (NP-hard).** Heuristics near-optimal in practice.
- **Parallelization:** disjoint-qubit gates can co-execute; max parallelism = anti-chain width in DAG.

5-Qubit QFT Worked Example (key takeaways)

- 17 gates total (5 H + 10 controlled-rotations + 2 SWAPs); QFT has tight dependencies; ASAP depth=10 vs sequential 17 (1.7× speedup); max parallelism 3 at $t = 5$; 16/17 gates critical (slack 0). ASAP $V = 35$, ALAP $V = 37$; ASAP preferred for minimizing qubit resources.

Hardware-Aware Scheduling

- **Coupling constraints, qubit mapping/placement** (NP-hard, SABRE).
- **SWAP gates** = 3 CNOTs each; routing changes DAG (adds SWAP nodes).
- **Joint optimization** mapping+scheduling, iterative schedule → route → re-schedule.
- **Noise-aware:** gate fidelities vary; pick higher-fidelity qubits/links using calibration data.
- **Crosstalk-aware:** trade parallelism vs crosstalk error; add serialization constraints.

Pulse-Level Scheduling

- Gate durations differ (1Q ~20 ns vs 2Q ~200–600 ns). Pulse-level packs into continuous time, exploits idle windows.
- **OpenPulse:** interface with calibration-based durations; 10–20% extra depth reduction over gate-level.

Scheduling vs Resource Allocation

- Scheduling = "when"; allocation = "what/who".
- **Logical resources:** qubit count, T-count, T-depth, CNOT count, Toffoli count, measurement count.
- **Physical resources:** physical qubit count, wall-clock runtime, code distance, magic-state-factory count, energy.

Lecture 16 — Quantum Software Stack (QSS)

Why a QSS?

- Bridges algorithms ↔ hardware; translates high-level problems into low-level hardware-specific instructions; abstracts diverse back-ends (SC, ion, neutral atom) and integrates QPUs into classical HPC for hybrid workflows.
- Analogous to classical accelerators: needs control interfaces, JIT compilation, resource scheduling.
- **Without QSS:** every user wires to every device. **With QSS:** a single shared stack via shared IRs.

MQSS Architecture (Munich Quantum Software Stack)

- **Front-end:** user-facing interfaces / high-level abstractions; adapters for Qiskit, PennyLane, C/C++ HPC.
- **Middle-end (toolchain & orchestration):** compile to IR; device-agnostic opts (gate reduction); device-specific transpilation (routing); dynamic resource scheduling.
- **Back-end:** unified HW abstraction (QDMI/QPM) standardizing job submission, pulse generation, hardware querying.

Core Design Principles

- **Hardware heterogeneity** (modular, extensible).
- **HPC integration** (tightly-coupled hybrid workflows).
- **Open interface** (community ecosystem via QDMI).

QDMI — Three Components

- **Session:** user mgmt, access control, resource mgmt.
- **Query:** device, site, operation properties.

- **Job:** configuration, submission, result retrieval.
- Clients consume; drivers and devices implement the interface.

Targeted Groups

- Domain experts (chemistry, finance), HPC experts (QPU as accelerator), algorithm & software developers, hardware providers / physicists, system admins.

Comparing Existing Stacks

- **Vendor-specific (Qiskit, Quil):** rich but tied to one vendor; naive lexicographic mapping in Qiskit yields lower success vs noise-aware.
- **Hardware-agnostic (TriQ):** multi-platform (IBM, Rigetti, UMD ions); up to 28× success-rate over Qiskit, 2.3× over Quil via noise-aware mapping.
- **HPC-integrated (MQSS):** open, modular; uses MLIR for hybrid compilation, QDMI for vendor-agnostic interfaces.

QOS — Quantum Operating System

- **What:** cloud OS managing quantum resources as a service; hardware-agnostic API for transparent job execution.
- **Problem:** QPUs noisy, small, heterogeneous; rudimentary cloud interfaces force tensions between fidelity, utilization, waiting time, load balance.
- **Why:** QPU demand $\gg$ supply; existing tools optimize one metric at a time. QOS balances tradeoffs across users/devices.

Technical Foundations

- Qubits, circuits, measurement; **NISQ** = small, high error, limited coherence (no error correction yet); **fidelity** as quality metric (drops 98.9% from 4 to 24 qubits in paper's data); **QAOA** drives the workload; **quantum-as-a-service** offered via IBM/AWS/Azure/Google.

Three Key Tensions & QOS Responses

- **A. Heterogeneity:** ~38% fidelity gap best vs worst IBM QPU on 12-qubit GHZ + day-to-day variance. *QOS: analytical Estimator predicts per-circuit fidelity on each QPU → scheduler picks best now.*
- **B. Utilization:** single job uses fraction of qubits; rest idle. *QOS: Multi-programmer spatially multiplexes compatible jobs onto same QPU, quantifying compatibility.*
- **C. Load balancing:** heavy QPUs queue while others idle; naive balance pushes to low-fidelity. *QOS: fidelity-aware multi-objective Scheduler trades 1–3% fidelity for up to 5× shorter waits + balanced load.*

QOS Architecture

- **QOS API:** `run`, `results`, `status`, `backends`, `backend_props`; transforms circuits to Qernels, hides QPU complexity.
- **Qernel:** common execution unit; static (qubits, depth, gates, Supermarq features) + dynamic (status, fidelity estimate, results).
- **Error Mitigator:** pre/post-processing pipeline composing complementary mitigation techniques.
- **Estimator:** lightweight analytical fidelity prediction.
- **Multi-programmer:** bundles compatible Qernels for spatial co-execution.
- **Scheduler:** fidelity-aware multi-objective over heterogeneous QPUs.

In-house QSS Discussion (Strategic Questions)

- Strategic control (vendor lock-in vs roadmap divergence); hardware optimality (TriQ 28×); HPC integration (SLURM, identity, storage); vendor lock-in vs multi-QPU heterogeneity (QDMI-style abstraction); talent / IP development; cost of ownership; community vs custom (extend MQSS/QFw vs build from scratch).

Lecture 17 — Quantum Benchmarking

Why Benchmark?

- Assess gate performance/fidelity; ensure fault-tolerance thresholds; identify & quantify errors.
- **Two main techniques:** Randomized Benchmarking (RB), Deterministic Benchmarking (DB).

Randomized Benchmarking (RB)

- Apply sequences of randomly chosen Clifford gates; measure fidelity decay to estimate average error.
- Ideal: $\rho \mapsto (C_m \circ ... \circ C_1)(\rho)$ then apply inverse $C_{m+1} = C_1^\dagger \circ ... \circ C_m^\dagger \Rightarrow$ identity, return to ρ.
- In practice each Clifford carries error $\Lambda_{i_j,j}$: $S_{i_m} = \odot_{j=1}^{m+1} \Lambda_{i_j,j} \circ C_{j_i}$; final $\tilde{\rho} \neq \rho$.

Standard RB Protocol

1. Prepare known initial state.
2. Apply random Clifford sequence.
3. Apply inversion gate.
4. Measure final-state fidelity.
5. Repeat for varying lengths m and average.

- **Outcome:** exponential fidelity decay vs $m \rightarrow$ extract average gate error rate.

RB Advantages & Challenges

- **Pros:** scalable to multi-qubit; robust to SPAM (state-prep & measurement) errors; gives global error rate.
- **Cons:** less sensitive to coherent/systematic errors; may underestimate certain error types $\rightarrow$ affects fault-tolerance assessments.

RB Variants

- **Interleaved RB:** measures fidelity of a specific gate by interleaving it within random sequences.
- **Simultaneous RB:** assess crosstalk via parallel RB on multiple qubits.
- **Correlated RB:** detect correlated errors across qubits.
- **Gate-set Shadow Protocol:** RB + shadow tomography for comprehensive characterization.

Deterministic Benchmarking (DB)

- Carefully chosen small set of gate-pair sequences targeting the real noise on single-qubit gates.
- **Procedure:** prepare state $U|0\rangle \rightarrow$ apply n reps of fixed two-pulse $P_1P_2 \rightarrow$ unprepare via $U^\dagger \rightarrow$ readout. Extract coherent + incoherent error parameters.

RB vs DB

Feature	RB	DB
Error sensitivity	Stochastic	Both coherent & incoherent
Sequence	Random Cliffords	Fixed deterministic
SPAM robustness	High	Moderate
Measurement reqs	High	Low
Characterization	Global error rates	Detailed decomposition

System-Level Benchmarking

- Evaluates entire system on realistic/algorithmic tasks (not single gates) — application-oriented utility.

Aspect	Gate-Level	System-Level
Focus	Single gate fidelity	Overall device
Methods	RB, T_1/T_2	Quantum Volume, full algorithm runs
SPAM sensitivity	Usually robust	Not robust (SPAM included)
Use case	HW tuning	User-facing perf evaluation

Quantum Volume (QV)

- IBM-introduced holistic metric of computational power. $V_Q = 2^d$ where d is the largest square circuit (depth = width = d) the device runs successfully (above threshold fidelity ~2/3).
- Reflects largest reliable problem size — useful for comparing devices with same qubit count but different errors/topology.

Lecture 18 — Distributed Quantum Computing (DQC)

Limits of Monolithic QC

- Practical algorithms need 10^6–10^7 physical qubits; today's NISQ chips 10^2–10^3.
- **Bottlenecks:** cryogenic & control (fridge dissipates only mW at 15 mK, wiring scales linearly); crosstalk & yield (cross-resonance, ZZ; yield falls super-linearly with die area); connectivity collapse (long SWAP chains); manufacturing (reticle ~26\$×\$33 mm caps die size).
- **Vendor roadmaps:** IBM (Heron 5K → Flamingo 7.5K/10K/15K → Starling 200 logical qubits 100M gates 2029 → Blue Jay 2000 logical 1B gates 2033); IonQ interconnect roadmap 2026 (datacenter) → 2027 (metropolitan 30 km) → 2028 (regional 100+ km, fault-tolerant link).

DQC — Definition

- $k \geq 2$ small/moderate QPUs interconnected via classical+quantum network cooperatively run a circuit too wide/deep for any single QPU.
- Involves **non-local gates** between qubits on different QPUs.
- **No-cloning:** cannot copy $|q\rangle$ from one QPU to another.

Three Archetypes

- **Multi-core (Quantum Networks-on-Chip):** same chip/fridge; homogeneous QPUs; mm–cm distance; low latency ~tens of ns.
- **Multi-computer (same room/farm):** multiple QCs via Quantum LAN (QLAN); heterogeneous OK; meters; ~μs latency.
- **Multi-farm:** geographically distributed; significantly heterogeneous; km scale (Quantum Internet); ms latency.

Jargon

- **Network primitives:** Bell-pair (EPR) distribution, entanglement swapping, entanglement distillation, classical 2-bit signaling.
- **Computation primitives:** local gates, non-local via **TeleGate**, qubit movement via **TeleData**, reversible message passing.
- **Resources:** data qubits + **ebits** (entangled bit) as first-class resource.

Bell Pairs, Swapping, Distillation

- $|\Phi^+\rangle = \frac{1}{2}(|00\rangle+|11\rangle)$ = unit of quantum communication; generated via photonic emitters, atom-photon interfaces, microwave-optical transducers.
- Each non-local gate consumes ≥ 1 ebit; ebit/s used as QoS for a quantum link.
- **Entanglement swapping:** if $A \leftrightarrow R$ and $R \leftrightarrow B$ share Bell pairs, a Bell-state measurement (BSM) at R + 2 classical bits creates direct $A \leftrightarrow B$ entanglement. Basis of quantum repeaters/long-range ebits.
- **Entanglement distillation:** multiple noisy ebits → one high-fidelity ebit via local ops + measurement.

Quantum Networking Primitives

- **Direct qubit transmission:** entangle a qubit to a photon, send over fiber.
- **TeleData:** quantum teleportation of an unknown $|\psi\rangle$ between QPUs — consumes 1 EPR pair + 2 classical bits.
- **TeleGate:** apply a non-local 2-qubit gate (e.g., CNOT) between data qubits on different QPUs **without moving them** — consumes a Bell state + classical comms; uses cat-entangler/cat-disentangler.

Hardware Layer — Inter-QPU Communication

- **All-microwave (cryo coax):** QPUs share fridge or cryo-coaxial wiring.
- **Microwave-optical transducer:** SC qubit → telecom photon → fiber → re-convert; efficiency ~5–30% today (DQC bottleneck).
- **Native photonic / matter-flying:** trapped ions, NV centers, neutral atoms emit photons directly.
- Fault-tolerance is the binding constraint, not the link.
- Physical 2-qubit error today $\approx 10^{-3}$; algorithms need $\approx 10^{-10}$ logical $\Rightarrow$ ~10^3–10^4 physical per logical (surface code).
- HW engineers optimize **ebit-per-second** and **ebit-fidelity**; compilers must treat both as budgets.

Data vs Communication Qubits

- Each QPU partitions physical qubits into two disjoint sets (hyperparameter).
- **Data qubits n_d:** carry algorithmic state; drive WIDTH of distributable circuits; need long coherence; bigger $n_d \rightarrow$ fewer SWAPs / partitioning overhead.
- **Communication qubits n_c:** hold one half of a peer EPR pair; drive THROUGHPUT of remote gates; need high-bandwidth photonic interface; bigger $n_c \rightarrow$ more parallel TeleGate/TeleData per layer.
- **First-order model:** $n_{\text{phys}} = n_d + n_c$; $R_{\text{remote}} \leq \min(n_c, BW_{\text{link}} \cdot t_{\text{layer}})$; $T_{\text{circuit}} \approx T_{\text{local}} + (G_{\text{remote}} - R_{\text{remote}}) \cdot T_{\text{ebit}}$.

DQC Compiler — 3 Stages

1. **Qubit assignment (partitioning):** map n logical $\rightarrow$ k QPU partitions to minimize inter-partition gates. Models: hypergraph k-cut, weighted K-L. Static vs dynamic (re-partition on moving window via TeleData). NP-hard.
2. **Non-local gate handling:** per cut edge decide TeleGate vs TeleData; TeleGate via cat-entangler/cat-disentangler. Objective: minimize #ebits and depth.
3. **Local circuit compilation + scheduling:** per-QPU gate decomp + routing + noise-aware mapping; cross-QPU schedule remote ops in time, respect ebit-generation rate, hide latency behind local clock; co-design treats ebit availability as scheduler constraint.

Distributed Algorithm Examples

- **QFT:** monolithic QFT has $m-i$ controlled-R ops per qubit; distributed version performs n non-local operations using a single shared entangled state (cat-entangler pattern).
- **VQA:** 3-qubit variational form distributed across two devices by replacing inter-device CNOTs with TeleGate using one Bell state and classical comms.

DQC Challenges

- Decoherence vs latency (SC ~100 μs, ion ~1 s vs inter-farm RTT ms).
- NP-hard partitioning, search grows combinatorially with k + time-varying coupling.
- No end-to-end noise-aware multi-QPU compiler in production.
- Distillation vs throughput: higher-fidelity ebits need more rounds $\rightarrow$ more comm qubits $\rightarrow$ less compute.
- Multi-tenant scheduling over shared entanglement fabric.
- Verification/debugging — no copy/peek of quantum state.

DQC Opportunities

- Exponential power scaling (linear interconnected QPUs $\Rightarrow$ exp. joint Hilbert space).
- Quantum-centric supercomputing (CPU + GPU + QPU + EPR fabric within SLURM-style schedulers).
- Multi-vendor heterogeneity (ions for fidelity, SC for fast Clifford, photonic memory).
- Measurement-based one-way QC natural in networked setting.
- Geographic + secure compute (DQC + QKD = secure delegated computation).
- Modular fault-tolerance via lattice surgery across modules.

Five Take-Aways

1. Monolithic QPUs hit physical/thermal/yield walls long before practical algorithm sizes.
2. DQC scales out: theoretically equivalent to monolithic up to polylog depth, paying real ebit + classical signaling cost.
3. Three archetypes (multi-core / multi-computer / multi-farm) trace deployment path.
4. TeleData + TeleGate + ebit-aware scheduler = minimal compiler primitives.
5. Bottleneck is no longer "enough qubits" but "enough high-fidelity ebits/second".