Appunti della Lezione 12/09/2025

Nel mondo classico i bit sono definiti in maniera artificiale: una tensione più bassa di una soglia viene interpretata come 0, una più alta come 1. È un approccio "passivo": l'hardware non ha intrinsecamente due stati, siamo noi a imporli con segnali elettrici.
Nel quantistico invece la situazione è diversa. Gli stati base (0 e 1) corrispondono a proprietà fisiche **intrinseche** di particelle o sistemi reali: due livelli energetici di un atomo, due orientazioni di spin, due correnti in un circuito superconduttore, due polarizzazioni di un fotone. Non "accendiamo" un 1: scegliamo due stati già esistenti della natura e li etichettiamo come 0/1. Da qui nasce la potenza, ma anche la fragilità del calcolo quantistico.

Un qubit è utile solo finché il suo stato resta stabile. L'ambiente circostante però tende a "disturbare" la particella, facendo decadere lo stato e mescolando 0 e 1. È il fenomeno della **decoerenza**, la ragione principale per cui il quantum computing è rimasto a lungo un sogno.
La **coerenza** misura quanto a lungo lo stato resta utilizzabile. A seconda della tecnologia, questa finestra di tempo varia moltissimo:

- Superconduttori: decine-centinaia di microsecondi.
- Ioni intrappolati: da millisecondi fino a secondi.
- Atomi neutri: da frazioni di millisecondo fino a **decine di secondi** in certi stati, un tempo enorme in questo contesto.
- Fotoni: coerenza praticamente indefinita, ma con interazioni molto difficili da gestire

Oltre alla coerenza c'è la **fedeltà di porta**, cioè quanto un'operazione (una "porta logica quantistica") sia eseguita correttamente. Anche partendo da errori minuscoli, la probabilità di sbagliare cresce con la complessità del circuito: una sequenza di dieci porte amplifica gli errori molto più di una singola. Per questo si lavora su tecniche di **fault tolerance** ed error correction, che però richiedono ulteriori qubit e aumentano la complessità.

Due domande fondamentali:

1. **Scalabilità** – quante unità fisiche posso mettere insieme?
2. **Connettività** – quali qubit possono interagire direttamente?

La seconda è cruciale: se i qubit possono interagire solo con i vicini immediati (come nei superconduttori, stampati su un wafer 2D), per collegare qubit lontani bisogna usare operazioni di "trasporto" (SWAP), che consumano tempo e fedeltà.
Gli atomi neutri, invece, offrono una **connettività programmabile** in 2D e 3D: gli atomi vengono intrappolati con pinzette ottiche in geometrie a piacere, permettendo di mappare direttamente il grafo del problema sul grafo fisico dei qubit.

Gli **atomi neutri** sono trattenuti da trappole ottiche (fascetti laser focalizzati). Ogni trappola può contenere un atomo, che diventa un qubit. La distanza tra le trappole e lo stato di eccitazione degli atomi determina l'intensità dell'interazione.

Quando un atomo è portato in uno **stato di Rydberg** (altissimo livello energetico), interagisce fortemente con i vicini. Questa interazione segue la legge di van der Waals, che decresce con la sesta potenza della distanza e cresce molto rapidamente con il numero quantico principale n.

Un fenomeno centrale è il **Rydberg blockade**: se due atomi sono troppo vicini, l'eccitazione di uno impedisce l'eccitazione dell'altro. Questo meccanismo viene usato per realizzare porte a due qubit, come la **Controlled-Z**, che è un mattoncino fondamentale per la computazione quantistica universale

Operazioni e ciclo di calcolo

- **Operazioni single-qubit:** tramite microonde o transizioni Raman (due fotoni), che cambiano lo stato iperfine dell'atomo.
- **Operazioni multi-qubit:** tramite eccitazione a Rydberg e blocco, per generare entanglement controllato.
- **Readout:** avviene con fluorescenza, misurando la luce emessa dagli atomi eccitati, con fedeltà molto alta.

Il ciclo tipico è: **raffreddamento → caricamento degli atomi nelle trappole → riarrangiamento → calcolo → lettura**. Il caricamento iniziale è probabilistico, ma con tecniche di riarrangiamento si arriva a configurazioni dense e programmabili

È importante legare la tecnologia a problemi reali, non solo a esercizi accademici. Un esempio concreto è quello di **ottimizzazione di campagne marketing**:

1. Un modello predice la probabilità che ogni utente risponda a una campagna.
2. Si formula il problema come **QUBO** (Quadratic Unconstrained Binary Optimization), cioè un Hamiltoniano di Ising.
3. Il quantum solver calcola la distribuzione ottimale del budget, massimizzando i ritorni e rispettando vincoli di costo.

In un caso reale con D-Wave, questa strategia ha portato a un incremento di oltre il 25% rispetto al metodo classico, con un ritorno economico multimilionario.

Analogamente, problemi di **routing (TSP)**, **scheduling aeroportuale** e **radioterapia oncologica** possono essere formulati e affrontati con approcci simili, sfruttando la possibilità di rappresentare vincoli e interazioni direttamente nella geometria degli atomi intrappolati.

Il cuore di questi sistemi non è solo il chip ma un'intera infrastruttura:

- **Sistemi laser** per raffreddare e intrappolare gli atomi a microkelvin.
- **Ottiche e modulatori spaziali (SLM, AOD)** per generare e riordinare le trappole.
- **Elettronica di controllo (FPGA)** per sincronizzare con precisione sub-microsecondo.
- **Isolamento termico e magnetico** per ridurre i disturbi.

Il risultato è una macchina che occupa l'equivalente di una stanza di alcuni metri, non molto diversa per dimensioni dai supercomputer quantistici superconduttori. Ogni dettaglio conta: una variazione di 0,1 °C può compromettere il calcolo.

La **scalabilità** è il punto forte degli **atomi neutri**: una volta che sai creare e posizionare poche **pinzette ottiche**, puoi estendere lo **stesso schema** a centinaia o migliaia di trappole, perché l'aggiunta di qubit richiede soprattutto **ottica programmabile** (SLM/AOD) più che nuove linee elettriche dedicate. In pratica, la "fatica" di scalare è diversa da piattaforme a **circuiti superconduttivi**, dove l'aumento di qubit implica anche un aumento di **cablaggio, controllo RF, lettura** e quindi complessità di sistema. Questo è coerente con l'offerta pubblica: **QuEra "Aquila"** espone array 2D di **256 atomi** accessibili via **AWS Braket**, proprio come dispositivo "field-programmable" (geometria riconfigurabile) per simulazione/ottimizzazione analogica.

Oggi l'operatività **2D** è matura; il **3D** è attivo in ricerca/sviluppo (prototipi e annunci), ma richiede più verifiche applicative prima di dichiararlo "prodotto". Anche qui, i comunicati recenti di **Pasqal** mostrano road-map e dimostratori, e soprattutto l'**integrazione in Azure Quantum** che rende il loro hardware disponibile via cloud.

Distinzione fra **come codifichi il qubit** e **quanto circuito puoi farci**:

- **Stati elettronici / Rydberg** ⇒ **porte veloci** (µs) ma **finestre di coerenza brevi** ($\sim 10^{-4}$–10^{-3} s). Ottimi per creare entanglement e interazioni forti, ma limitano la **profondità** di circuito.
- **Stati di base iperfini / spin nucleare** ⇒ **coerenze lunghe** (fino a secondi nelle migliori condizioni), ideali come **memoria** o per porzioni del calcolo che "stazionano" a lungo fra operazioni.

Ne deriva un'architettura "**eterogenea**": usi gradi di libertà **veloci ma fragili** per le interazioni, e **lenti ma robusti** per conservare informazione. Questa separazione dei ruoli è esattamente ciò che l'ingegneria dei sistemi Rydberg persegue oggi.

Error budget: cosa rompe davvero un calcolo

Gli errori non “pesano” tutti uguale. Quelli principali (e come si mitigano):

- **Perdita di atomo** (worst case): se l’atomo esce dalla trappola durante l’esecuzione, il qubit **non esiste più** ⇒ il circuito è irrimediabilmente compromesso. Va mitigata con **trappole stabili**, temperature basse, controllo di **heating** e sequenze che riducano tempi in **Rydberg** (dove la vita media è più breve).
- **Dephasing/decadimento in Rydberg**: limita la fedeltà delle CZ; si contrasta scegliendo **n** appropriato, **impulsi brevi** e **laser** stabili in ampiezza/fase.
- **Rumore di laser e “wrong pulse”**: deriva da ampiezza/detuning non ideali; serve **calibrazione continua** e controllo della catena ottica/RF.
- **Crosstalk**: un raggio o un campo pensato per A “sfiora” B; si progetta la **spaziatura** in modo da restare **sotto il raggio di blocco** per le coppie che **devono** interagire e **sopra** per quelle che **non devono**, cioè un vero **problema d’ottimizzazione nella progettazione** dell’array.

Questa visione “per priorità” è corretta: perdita > errore di fase > rumore di controllo > crosstalk. La progettazione di **distanze/tempi** è il mestiere quotidiano delle squadre hardware Rydberg.

Panorama industriale (corretto e aggiornato)

- **QuEra**: **Aquila (256)** su **AWS Braket**; forte accento su **ottimizzazione/ simulazione** con neutral atoms programmabili. Più recentemente il gruppo ha mostrato passi su **logica corretta** (dimostrazioni su pochi **qubit logici** con distanze e “magic state factory”), indice di traiettoria verso logica su scala maggiore.
- **Pasqal**: disponibilità tramite **Azure Quantum** e roadmap con **array 2D/3D**; messaggi pubblici parlano di “quantum advantage” *per casi specifici*, in linea con la prudenza scientifica (speedup **problema-dipendente**).
- **Atom Computing**: collaborazione con **Microsoft**; **24 qubit logici** dimostrati (creati/entangled) con neutral atoms e capacità di rilevare/correggere errori—un tassello importante verso il **fault tolerance**. Le dimensioni “512” citate in alcune conversazioni sono da trattare come **dichiarazioni** finché non supportate da **dati pubblici**; la comunicazione tecnica più solida è sui **24–28 logici**.
- **Infleqtion** (ex ColdQuanta): collabora con **NVIDIA (CUDA-Q)** su flussi ibridi e dimostrazioni con **qubit logici**; l’azienda è molto attiva anche in **sensing** (orologi atomici, navigazione).
- **D-Wave** (annealing ibrido): pur non essendo atomi neutri, è rilevante nell’ottimizzazione industriale; i **solver ibridi CQM** accettano **centinaia di migliaia** di

variabili (ordine **10^5–10^6** a seconda del solver/versione), decomponendo il problema e interrogando il QPU per i sottoproblemi "duri".

Nota di metodo: "**quantum advantage**" non significa "batte i classici su *tutto*"; significa **vantaggio misurabile** su **istanze/problemi ben definiti**. È un traguardo scientifico fondamentale, ma **non** garantisce **impatto business generalizzato**.

Il **QAOA** è un ciclo **ibrido**: il **quantum** prepara uno **stato parametrico** (profondità p), tu **misuri** un'**aspettazione** che implementa il **costo** del problema, poi un **ottimizzatore classico** aggiorna i parametri e ripeti. È corretto dire che, per problemi realistici, conviene **spendere i qubit** per rappresentare **la parte più dura** (decision function/configurazione) e lasciare al classico il grosso dell'**ottimizzazione**.

Sulla **stima del gradiente**: in generale puoi usare la **parameter-shift rule** (tutto **on-device**, ma richiede molte valutazioni) oppure **finite-difference**/**autodiff** sul **modello classico** che avvolge la chiamata quantistica. Nella pratica attuale, per limiti di **risorse** e di **rumore**, molte pipeline usano **ottimizzatori classici** (CMA-ES, SPSA, Adam…) e riservano al device la sola **valutazione del costo**. Framework come **PennyLane** forniscono entrambe le strade e unificano back-end diversi.

Chiarezza concettuale: dire che i dispositivi sono "**macchine di minimizzazione**" è una **buona metafora** per **annealer/simulatori analogici** (cercano ground state di un Hamiltoniano). Nel **gate model**, invece, si minimizza **per via variazionale**: è il **loop classico** che guida i parametri verso il minimo, mentre l'hardware fornisce **campioni/aspettazioni** del costo. Entrambi i mondi restano ibridi.

Cosa portarsi a casa (versione operativa)

- **Atomi neutri**: scala "con l'ottica", buon **match** con grafi di problemi, **CZ** via **Rydberg blockade**, e possibilità di **mixare** gradi di libertà veloci/lenti per ruoli diversi nel circuito.
- **Progetta l'array** come un problema a sé: **distanze** per massimizzare accoppiamenti utili e minimizzare **crosstalk**; **tempi** per stare entro le **coerenze** (breve in Rydberg, lunga negli stati di base).
- **Stack software**: evita lock-in quando possibile (p.es. **PennyLane** come front-end agnostico; **Pulser** per Pasqal quando vuoi sfruttare feature specifiche). Nel cloud oggi trovi **QuEra su Braket** e **Pasqal su Azure**.
- **Aspettative** realiste: "quantum advantage" è **mirato**; per **scopi industriali** spesso vince l'**approccio ibrido** (pre/post-processing classico + **quantum** sulla parte realmente refrattaria ai metodi classici).

Vengono a confronto i principali approcci basati su **atomi neutri** (e citando altri player). L'idea guida è che non basta "contare i qubit": servono **geometria delle interazioni** (connettività) e **tempi di coerenza** adeguati al tipo di operazioni.

- **QuEra (Aquila)**: array 2D fino a qualche centinaio di siti accessibili via cloud; interazioni Rydberg su distanze micrometriche ⇒ **connettività programmabile** (decidi chi interagisce con chi disegnando l'array e indirizzando i laser). I tempi di coerenza "effettivi" durante le operazioni Rydberg sono **brevi (~100 µs–ms)**, perché in Rydberg la vita media è limitata: per questo i **gate a due qubit** sono veloci (sub-µs–pochi µs).
- **Pasqal**: neutral atoms analog/digitale con forte accento su **simulazione/ottimizzazione**; si parla di **array 2D/3D**. "Fully connected" va inteso con cautela: fisicamente le interazioni decrescono con la distanza; l'"all-to-all" si ottiene **programmando** le distanze e/o usando schemi "globali" (non è un bus magico che collega tutto a tutto).
- **Atom Computing**: qubit memorizzati in **stati di base iperfini** (o spin nucleare) con **coerenze molto lunghe (fino a secondi)**: ottimi come **memoria** o per tratti "in sosta" del circuito; l'entanglement si ottiene attivando stati od operazioni più "veloci ma fragili".
- **Infleqtion** (ex ColdQuanta): approccio ibrido e filiera ampia (quantum + sensing), con discorsi su **scaling modulare** e integrazione software/hardware.

Messaggio chiave: gli **stati di base** servono per **conservare** informazione a lungo; gli **stati di Rydberg** per **interagire velocemente**. Molte architetture combinano consapevolmente i due regimi.

Metriche:

- **Soglia di tolleranza agli errori (fault-tolerance threshold).** Non è "quanti errori commetto", ma il **limite massimo per porta** sotto il quale la **correzione d'errore** può, in principio, far crescere l'affidabilità all'aumentare della ridondanza. Dire "99,9%" vuol dire **errore 1e-3 per porta**: sotto certe soglie si può costruire un qubit **logico** affidabile da molti qubit fisici.
- **Rearrangement success.** È la **probabilità di riempire e riordinare** l'array dopo il caricamento probabilistico (tante trappole, non tutte piene al primo colpo). Serve >99% per avere array densi e ripetibili.
- **Quantum Volume (QV).** Spesso frainteso. Un **QV=2^k** indica che il sistema esegue **circuiti casuali "quadrati"** (larghezza = profondità = k) con fedeltà statistica accettabile. Non vuol dire "12 qubit logici" né "compongo 2^12 variabili": misura **congiuntamente** larghezza, profondità, fedeltà e compilazione.

- **CLOPS (Circuit Layer Operations Per Second).** È un **throughput**: quante "layer" di porte effettive per secondo, includendo overhead di controllo/lettura. Paragonarlo in modo diretto ai GHz di una CPU è fuorviante: il valore utile è **quante valutazioni per unità di tempo** riesci a fare in un **workflow ibrido** (es. QAOA).

QAOA, Ising e grafi:

Flusso: **problema combinatorio → QUBO/Ising → mapping su chip**. Nel grafo:

- i **nodi** sono le **variabili** (qubit);
- gli **archi** rappresentano **interazioni** o **vincoli** (pesi J_{ij}).

Esempi didattici ricorrenti: **MaxCut**, **Graph Coloring**, **Independent Set**, **Vertex Cover**. L'esempio "antenne con minimo overlap" assomiglia più a un **problema di copertura/massima copertura** (o a un Vertex-Cover con vincoli geometrici): si può **QUBO-izzare**, ma va formulato con attenzione (penalità per sovrapposizione, vincoli di budget, ecc.). La cosa bella degli **atomi neutri** è che puoi riflettere **la struttura del grafo** nella **geometria fisica** riducendo i **SWAP**.

Ottimizzazione vs simulazione: due modalità diverse

- **Ottimizzazione (annealing/variazionale):** cerchi **un punto** (tipicamente il minimo dell'Hamiltoniano di costo).
- **Simulazione di Hamiltoniani:** cerchi di **campionare** il **comportamento** di un sistema fisico (es. magnetismo quantistico, transizioni di fase), "specchiandolo" nel chip. Qui le **array grandi** (centinaia di atomi) sono già state usate per studiare **transizioni di fase** e **modelli di spin**: il valore è scientifico/industriale quando vuoi capire **la dinamica**, non solo il minimo.

Limiti attuali:

- **Profondità utile**: con Rydberg e rumore attuale, si resta nell'ordine delle **decine di layer** realmente affidabili su circuiti piccoli/medi (dipende dal device).
- **Rumore e perdite**: **perdita di atomo** è il caso peggiore; seguono **decoerenza/dephasing** (specie in Rydberg), **rumore di ampiezza/fase dei laser**, **crosstalk**.
- **Connettività imperfetta ⇒ SWAP ⇒ errori**: se il grafo del problema non "entra" bene nella geometria fisica, i costi di routing crescono.

Roadmap:

- **QEC in pratica.** Codici e **qubit logici** sono passati dalla teoria a **dimostrazioni su scala "decine"**: la traiettoria da qui a **50+ logici** (stabili) è l'obiettivo dichiarato da più gruppi nel medio periodo.
- **Scalabilità fisica.** Array da **~10^3 qubit** sono nella portata ingegneristica; **modularità** + **link fotonici** promettono di interconnettere più "rack" quantistici: più semplice gestire **blocchi** da 10k che un unico monolite da 20k.
- **"Simulated qubits".** Quando i **qubit fisici** non bastano, si ibrida con **HPC/GPU** per emulare parti del problema: non è perfetto, ma spesso **meglio che niente** se il requisito è scala massiva.
- **Aspettative per domini specifici.** In aree dove la **fisica microscopica** è il problema (chimica, materiali, parte della biotech), ridurre le **approssimazioni classiche** può cambiare la partita; in altre (es. finanza) tolleri approssimazioni più spinte e il vantaggio è più sottile/contestuale

Di seguito viene illustrato come si programma l'hardware ad **atomi neutri**, come si mappa un problema in **Ising/QAOA**, quali sono le **modalità di lettura**, le **sorgenti d'errore**, e come leggere **metriche** e **case study.**

Il ciclo operativo: dal raffreddamento alla soluzione

La macchina lavora a "cicli": **raffreddo → carico → intrappolo → ri-arrangio → eseguo → leggo**. Si parte da un **MOT** (trappola magneto-ottica) che porta gli atomi a ~10^{-4} K; poi **polarization-gradient cooling** rifinisce fino a decine di μK. Ogni pinzetta ottica (tweezer) dovrebbe contenere **un solo atomo**: questo si ottiene grazie a **collisioni luce-assistite** (collisional blockade) durante il caricamento; non è il Rydberg blockade, che invece useremo più avanti per i **gate**. Una volta caricata una frazione delle trappole, si fa **imaging** per sapere quali sono piene e si effettua il **rearrangement** con SLM/AOD per popolare esattamente la geometria desiderata.

Perché il ri-arrangiamento è centrale. In un processore a atomi neutri la **topologia delle interazioni** si programma **spostando gli atomi** (e scegliendone la distanza). Questo consente di far "assomigliare" il chip al **grafo del problema** e ridurre SWAP/overhead. La cosa importante, sottolineata anche a lezione, è che il **ri-arrangiamento può avvenire dinamicamente** dentro lo stesso job: se cambia la topologia utile a metà algoritmo, posso riposizionare atomi e proseguire, senza ripartire da zero. È proprio qui che la piattaforma si differenzia dagli **annealer** con **connettività fissa** (es. D-Wave): lì si "congela" un embedding per ogni istanza; cambiare topologia significa **re-embed** e riallineare tutto.

Lettura (readout): distruttiva, non-distruttiva e "parallela"

Si distinguono modalità di **readout**:

- **Finale/distruttiva**: imaging aggressivo per massima fedeltà, ma **collassa e perturba** il registro; tipico per l'output finale.
- **Non-distruttiva / mid-circuit**: misure **selettive** durante il circuito per **diagnostica/feedback** con impatto minimo sugli altri qubit; resta comunque una misura quantistica (il qubit misurato collassa), ma l'obiettivo è **limitare** l'effetto sul resto e poter **continuare** l'esecuzione.
- **Parallela**: combinazioni/configurazioni per coprire più regioni o bilanciare fedeltà vs velocità.
  Messaggio pratico: la possibilità di **mid-circuit readout** indirizzato è uno dei punti forti della piattaforma, pur non essendo "magico" (zero impatto non esiste).

Dove nascono gli errori (e come si mitigano)

Il "budget d'errore" tipico include:

- **Laser** (ampiezza/frequenza instabili, fase, *addressing* imperfetto) → si combatte con stabilizzazione, calibrazione continua e shaping degli impulsi.
- **Porte a due qubit**: **decadimento Rydberg**, **leakage** da blockade non perfetto, rumore meccanico (vibrazioni), detuning non ideale.
- **SPAM** (State Preparation And Measurement): preparazione dello stato, microonde, purezza iniziale.
- **Readout**: errori della camera, rumore, **perdita di atomi** se la configurazione è troppo aggressiva nell'ultimo scatto.
  La **perdita di atomo** è il caso peggiore: quel qubit non esiste più. Per questo si ottimizzano **tempi** (impulsi brevi in Rydberg), **temperature** e **profili d'impulso** (composite/echo/adiabatici) per ridurre decoerenza e leakage.

Programmare l'hardware: Pulser (Pasqal) e PennyLane

Due livelli di astrazione coesistono e sono **complementari**:

Pulser (low-level, hardware-centric). Si definisce un **Register** (coordinate 2D/3D dei siti → geometria/grafo) e poi una **Sequence** di canali/impulsi: ampiezza, detuning, fase, durata. È vicino alla macchina: controlli davvero i **laser**, il **timing** e perfino sequenze **composite** (adiabatico, echo, phase compensation) per mitigare errori e mantenere la coerenza. È

potente ma “impegnativo”: ottimo per sfruttare al massimo l’hardware o testare profili d’impulso robusti.

PennyLane (circuit-level, device-agnostic). Fornisce primitive **variazionali** (VQE, **QAOA**, QML) e dispositivi/simulatori; puoi scrivere il problema come **grafo**, ottenere $\mathbf{H_c}$ e **mixer** già pronti (es. `qaoa.maxcut(graph)`), e far girare il ciclo **ottimizzatore classico ↔ device quantistico**. Il vantaggio è l’**orchestrazione multi-provider**: in teoria puoi comporre pipeline che chiamano hardware diversi per sotto-compiti differenti, restando nello stesso framework di autodifferenziazione e training (anche con PyTorch).

Nota di accuratezza: in QML **“automatic differentiation”** è una tecnica software (reverse-mode, parameter-shift, ecc.) per calcolare gradienti; **non** significa che i “pesi cadono da soli al minimo” per legge fisica. Quella è una metafora utile per capire il ruolo del **ground state**, ma i gradienti si ottengono comunque via **autodiff/parameter-shift** in un loop classico-quantum.

Dal grafo al circuito: MaxCut/QAOA in pratica

Flusso tipico (anche nello snippet mostrato):

1. **Definisci il grafo** (lista di archi).
2. **Ottieni** H_C **e** H_M (cost & mixer) già implementati.
3. **Cost function** = $\langle H_C \rangle$ sullo stato preparato da p strati $[\exp(-i\gamma_k H_C)\exp(-i\beta_k H_M)]$.
4. **Ottimizzazione classica** (SGD/Adam/SPSA/CMA-ES…) delle $\{\gamma, \beta\}$.
5. **Compilazione su atomi neutri**: mappo i qubit del grafo su **siti fisici** vicini quando devono interagire (CZ via blockade) e lontani quando devono essere indipendenti → meno SWAP, profondità ridotta.

Strategia di compilazione e connettività

Prima di lanciare il job:

- **Mapping logico→fisico** per minimizzare **SWAP**.
- **Inserimento SWAP** dove inevitabile e **ri-arrangiamenti dinamici** quando conviene cambiare topologia in corsa.
- **Generazione degli impulsi** coerente con i vincoli temporali e di potenza del dispositivo.
  Tutta questa parte è **classica** (pre-compilazione), poi il circuito/sequence viene inviato al QPU.

Tempi, ripetizioni (shots) e perché serve campionare

Anche quando il **tempo puro** di "move+verify" è dell'ordine **ms–decine di ms** per un'istanza semplice, in pratica occorrono **più run** per via del rumore: si parla di **shots** (es. 10^3) per stimare correttamente la soluzione o campionare lo **spazio delle soluzioni** intorno all'ottimo. Nel paradigma **NISQ** si confronta sempre **tempo-alla-soluzione** + **qualità** vs baseline classica.

Metriche: CLOPS, Quantum Volume, Q-Score (come leggerle)

- **CLOPS**: throughput (quante "layer di circuito" al secondo inclusi overhead di controllo).
- **Quantum Volume (QV)**: dimensione del più grande **circuito casuale "quadrato"** (larghezza = profondità) implementabile con confidenza; non misura "qubit logici".
- **Q-Score**: ampiezza/profondità massime che raggiungono una data **probabilità di successo** su circuiti di riferimento.
  Le slide mostrano numeri d'ordine (es. QV ~2^{12}, range CLOPS per piattaforme, Q-Score di esempio): sono utili come **indicazioni** ma **dipendono da setup/benchmark**; servono più per confronti **intra-piattaforma nel tempo** che per "classifiche assolute" tra tecnologie.

Case study & stack cloud

È stato citato un case study (es. **Pasqal Orion Gamma**): **array 2D/3D** di atomi di Rb-87, **>250 qubit**, modalità **digitale-analogica** e applicazioni (MaxCut, coloring, 3-SAT, QML, simulazione di materia condensata, finanza). Sul lato software si è vista l'**integrazione API**: PennyLane, Pulser, AWS Braket (QuEra), Qiskit/Cirq, con l'idea di orchestrare **workflow eterogenei** (hardware diversi per sottoproblemi diversi) – tenendo presente, però, che oggi c'è **overhead** di latenza/costi se si fa tutto da cloud.

Cosa portarsi a casa (versione "operativa")

1. **Programmare la geometria è programmare l'algoritmo**: con gli atomi neutri **il grafo del problema diventa letteralmente il layout** (distanze ⇒ interazioni). Il **ri-arrangiamento dinamico** è un vantaggio reale rispetto a connettività fissa.
2. **Pulser vs PennyLane**: il primo dà **controllo fine degli impulsi** (perfetto per fisica/gating/mitigazione), il secondo **astrazione circuitale** e **ibrido ML-friendly** (ottimo per QAOA/VQE/QML). Non sono concorrenti: si usano **a livelli diversi**.
3. **Errori e misure**: i veri nemici sono **perdita di atomi**, **leakage/decadimento Rydberg** e **rumore nei laser**; le **misure mid-circuit** sono utili ma vanno dosate (mai zero impatto).

4. **Benchmark**: usali come **bussola**, non come verità assoluta; ciò che conta per il tuo caso d'uso è **profondità utile**, **tempo-alla-soluzione** e **qualità/costo** rispetto alla baseline classica.

Appendice A - QUBO (Quadratic Unconstrained Binary Optimization)

Quando vogliamo risolvere un problema reale – ad esempio distribuire un budget marketing, pianificare turni, o scegliere il miglior percorso in una rete – dobbiamo trasformarlo in una forma che un computer quantistico possa capire. I qubit non "capiscono" direttamente fatture, voli o utenti: capiscono funzioni matematiche da minimizzare.

L'approccio standard è tradurre il problema in una **funzione obiettivo quadratica su variabili binarie**:

Variabili $x_i \in \{0,1\}$ rappresentano scelte o decisioni (es. "assegno budget a questo canale = 1, altrimenti = 0").

La funzione da minimizzare ha la forma:

$$\mathrm{QUBO}(x) = \sum_i a_i x_i + \sum_{i<j} b_{ij} x_i x_j$$

Qui a_i sono i pesi lineari (quanto conviene scegliere una variabile da sola), e b_{ij} sono i pesi quadratici (quanto conviene scegliere due variabili insieme o quanto si penalizza se entrambe sono 1).

Molti problemi hanno **vincoli** ("il budget totale deve essere ≤ 100"). Nella formulazione QUBO i vincoli non sono scritti a parte, ma **inseriti nella funzione obiettivo come penalità**. Esempio: se supero il budget, aggiungo un termine molto grande nella funzione che rende quella soluzione svantaggiosa. In questo modo ogni problema vincolato si trasforma in uno **senza vincoli** (da qui "Unconstrained"), ma con una funzione costruita per "punire" le soluzioni non ammissibili.

I computer quantistici spesso lavorano in termini di **spin** o **stati ±1**, non di 0/1. La formulazione QUBO si può tradurre direttamente in un **Hamiltoniano di Ising**, che è una funzione energetica tipica della fisica dei materiali magnetici:

$$H = \sum_i h_i s_i + \sum_{i<j} J_{ij} s_i s_j$$

dove $s_i \in \{-1,+1\}$.
Il legame è semplice: basta mappare $x_i = (1+s_i)/2$. I coefficienti a_i e b_{ij} si trasformano rispettivamente in campi locali h_i e interazioni J_{ij} .

Così, un problema di ottimizzazione diventa trovare lo **stato fondamentale** (energia minima) di un sistema di spin: un compito naturale per macchine quantistiche basate su **annealing** o su **evoluzioni controllate**.

Esempio concreto

Supponiamo di voler allocare un budget pubblicitario su tre canali.

- Variabili: x_1, x_2, x_3 (0 = non investo, 1 = investo).
- Vincolo: il budget massimo consente al massimo due canali.
- Obiettivo: massimizzare il ritorno (supponiamo r_1, r_2, r_3).

La funzione QUBO potrebbe essere:

$$Q(x) = -(r_1x_1 + r_2x_2 + r_3x_3) + P(x_1 + x_2 + x_3 - 2)^2$$

Il termine negativo premia i canali redditizi, il termine con P (penalità grande) punisce se più di due canali sono attivi. Risolvere il QUBO equivale a scegliere la combinazione di canali migliore rispettando il vincolo.

Perché è importante nel quantistico

- È una **forma standardizzata**: molti problemi diversi finiscono nello stesso formalismo.
- Si mappa bene sugli **hardware fisici** (annealing con Ising, o gate model con qubit che simulano spin).
- Permette di usare **ibridi classico-quantum**: un algoritmo classico prepara il QUBO, il quantistico propone soluzioni, e un classico rifinisce/valida.

Appendice B - QAOA, Ising e grafi

Come visto precedentemente, la forma QUBO è

$$\min_{x\in\{0,1\}^n} Q(x) = \sum_i a_i\, x_i + \sum_{i<j} b_{ij}\, x_i x_j,$$

dove:

- a_i pesa l'**utilità/costo** del singolo x_i,
- b_{ij} pesa l'**effetto combinato** di scegliere sia i che j (bonus o penalità).

I **vincoli** si inglobano come **penalità**. Due ricette ricorrenti:

- **Budget (uguaglianza):** $\sum_i w_i x_i = B$.
  Penalizza con $P(\sum_i w_i x_i - B)^2$. Espandendo, ottieni contributi lineari e quadratici:

  $$P\Big(\sum_i w_i^2 x_i + 2\sum_{i<j} w_i w_j x_i x_j - 2B\sum_i w_i x_i + B^2\Big).$$

  (Il termine B^2 è costante: si può ignorare.)
- **Cardinalità (al più K scelte):** $\sum_i x_i \le K$.
  Versione "dura": introduci variabili **slack** binarie e trasformi in uguaglianza.
  Versione "morbida" (spesso efficace): penalizza $P\big(\max(0, \sum_i x_i - K)\big)^2$ **approssimandola** con una forma quadratica che punisce l'eccesso (o usa equality con slack pesati).

Scelta di P (i "moltiplicatori di Lagrange"): abbastanza grande da rendere *sempre* peggiore una soluzione non ammissibile, ma **non** così grande da schiacciare la dinamica numerica. Euristica: scegli PPP maggiore della **somma del massimo guadagno** che otterresti violando il vincolo di poco. In pratica si provano 2–3 valori crescenti e si verifica la **feasibility rate**.

Da QUBO a Ising (per QAOA/annealing)

Molti dispositivi "pensano" in variabili di **spin** $s_i \in \{-1, +1\}$. La mappa è

$$x_i = \frac{1 + s_i}{2}, \qquad x_i x_j = \frac{1 + s_i + s_j + s_i s_j}{4}.$$

Sostituendo in $Q(x)$ ottieni la forma di **Ising**:

$$E(s) = \sum_{i<j} J_{ij}\, s_i s_j + \sum_i h_i\, s_i + \text{costante},$$

con

$$J_{ij} = \frac{b_{ij}}{4}, \qquad h_i = \frac{a_i}{2} + \frac{1}{4} \sum_{j \neq i} b_{ij}.$$

A differenza della QUBO, aggiungere una costante non cambia il problema (serve solo a riallineare energie).

Normalizzazione ai limiti hardware. Su ogni device i coefficienti hanno un range (es. $J_{ij}, h_i \in [-1,1]$). Si riscalano J,h dividendo per il massimo assoluto e si riassorbe lo scale in tempi/pulse (gate model) o in accoppiamenti/field (annealing/analogico).

QAOA: che cosa mappa davvero e come "rispetti" i vincoli

Per un problema su grafo (es. **MaxCut** con pesi w_{ij}), costruisci:

- **Hamiltoniano di costo** $H_C = \sum_{(i,j)\in E} w_{ij} \frac{1 - Z_i Z_j}{2}$ (o la variante Ising equivalente),
- **Hamiltoniano di mixing** $H_M = \sum_i X_i$ (standard) **oppure** un **mixer "vincolo-preservante"** se vuoi restare *sempre* nello spazio delle soluzioni ammissibili (utile per $\sum x_i = K$, colorazioni, matching, ecc.).

Il circuito QAOA di profondità p è

$$|\gamma, \beta\rangle = \left(e^{-i\beta_p H_M}\, e^{-i\gamma_p H_C}\right) \cdots \left(e^{-i\beta_1 H_M}\, e^{-i\gamma_1 H_C}\right) |+\rangle^{\otimes n},$$

e un **ottimizzatore classico** sceglie $\{\gamma_k, \beta_k\}$ per minimizzare $\langle H_C \rangle$.

Scelte pratiche:

- Se i vincoli sono cruciali, preferisci mixer vincolo-preservanti (XY-mixer, ring-mixer, swap-network) per non doverli "far rispettare" con penalità gigantesche.
- Warm-start QAOA: inizializza vicino a una buona soluzione classica (riduce varianza e tempi di convergenza).
- Sparsità: compila H_C in CZ (o Rydberg-CZ) solo dove ci sono archi del grafo → profondità minore.

Mapping su atomi neutri: due strade

(A) Digitale (gate-based) con porte CZ via Rydberg

- Posizioni gli atomi in modo che le coppie che devono interagire siano dentro il raggio di blocco, e quelle indipendenti fuori.
- Compili H_C (Ising) in una rete di CZ e rotazioni Z/X.
- Vantaggi: flessibilità algoritmica (QAOA, VQE, ecc.).
- Costi: ogni CZ richiede una sequenza π–2π–π e devi gestire crosstalk e budget d'errore (coerenza Rydberg limitata).

(B) Analogico (Rydberg-Ising "naturale")

Usi direttamente l'Hamiltoniano fisico dei Rydberg,

$$H = \sum_i (-\Delta_i)\, n_i + \sum_{i<j} V_{ij}\, n_i n_j + \sum_i \frac{\Omega_i}{2}\, (\sigma_i^x),$$

con $n_i = |r\rangle\langle r|_i$.

Nel regime di blockade ($V_{ij} \gg \Omega$ per vicini) e con detuning Δ scelto ad hoc, l'evoluzione "spinge" verso un insieme di eccitazioni mutuamente non adiacenti: è una realizzazione fisica di Maximum Independent Set (MIS) sul grafo determinato dalla geometria (archi tra atomi più vicini di una soglia).

- Se i nodi hanno pesi, li implementi modulando Δ_i (local detuning) → MWIS.
- Vantaggio: pochi parametri, usa interazione nativa (profondo vantaggio di connettività).
- Limite: è "fatto su misura" per problemi tipo MIS/packing su grafi geometrici (unit-disk/near-neighbors). Per grafi generali servono gadget/embedding, con overhead.

L'esempio delle antenne

Problema: attivare un sottoinsieme di antenne massimizzando copertura/valore ma **riducendo overlap/interferenza**.

Passo 1 – Variabili. $x_i=1$ se attivo l'antenna i.
Passo 2 – Grafo di conflitto. Crea un arco (i,j) se le antenne i,j **si sovrappongono troppo** (oltre soglia).
Passo 3 – Formulazione.

- Se l'obiettivo è **selezionare tante antenne NON sovrapposte** → **MIS/MWIS** sul grafo di conflitto (pesi = valore/capacità dell'antenna).
- Se ammetti **un po'** di overlap con costo → QUBO con penalità $b_{ij}>0$ per $x_i x_j$ su archi "in conflitto", più termini positivi $a_i<0$ che premiano l'attivazione.
- Vincoli (budget, numero massimo, copertura minima) → penalità o mixer vincolo-preservante.

Passo 4 – Scelta hardware.

- **Rydberg analogico:** disponi gli atomi come i **centri** delle antenne; scegli il **blockade radius** uguale alla soglia di conflitto → il chip **è** il grafo di conflitto. Metti pesi con Δi\Delta_iΔi.
- **Gate-based/QAOA:** stessa geometria per ridurre SWAP; compili H_C e scegli p (profondità) compatibile con coerenza.

Nota pratica. Se il tuo grafo **non** è "unit-disk-like" (non è determinato da distanze) puoi:

1. **deformare la geometria** (layout ottico) per approssimarne la struttura,
2. usare **ancilla/gadget** per simulare archi "lunghi" (costo = più qubit e più CZ),
3. mappare su **annealer** con embedding dedicato (altra storia, ma utile se la geometria fisica è rigida).

Errori comuni e "sanity check"

- Penalità troppo piccole: il solver "bara" violando vincoli. Alza P finché tutte le migliori soluzioni campionate sono ammissibili.
- Penalità troppo grandi: numerica instabile; i termini utili spariscono. Scala i coefficienti per stare nel range hardware e bilancia P.
- QUBO denso senza motivo: elimina/rafforza solo le interazioni rilevanti (sparsità = meno gate, meno crosstalk).
- Mixer sbagliato: su problemi con feasible set piccolo, il mixer standard X esplora troppo lo spazio inammissibile → usa un mixer constraint-preserving.
- Embedding che ignora la geometria: su atomi neutri, la geometria è il tuo super-potere: usa layout per rappresentare il grafo e tagliare SWAP.

Mini-ricetta operativa

1. Definisci x_i e l'obiettivo;

2. scrivi i vincoli e scegli se farli duri (mixer) o morbidi (penalità);
3. costruisci $Q(x)=\sum a_i x_i+\sum b_{ij} x_i x_j$;
4. converti in Ising (J,h), riscalando nei limiti hardware;
5. progetta geometria (atomi) che rispecchi il grafo (minimizza SWAP/crosstalk);
6. scegli modalità (QAOA o analogico Rydberg) e i parametri iniziali;
7. fai warm-start (buona soluzione classica) e ottimizzazione ibrida;
8. verifica feasibility e confronta con baseline classica (gap di qualità e/o tempo-alla-soluzione).

Appendice C - Atomi neutri e stati di Rydberg

Gli atomi neutri sono trattenuti da trappole ottiche (fascetti laser focalizzati). Ogni trappola può contenere un atomo, che diventa un qubit. La distanza tra le trappole e lo stato di eccitazione degli atomi determina l'intensità dell'interazione.

Quando un atomo è portato in uno stato di Rydberg (altissimo livello energetico), interagisce fortemente con i vicini. Questa interazione segue la legge di van der Waals, che decresce con la sesta potenza della distanza e cresce molto rapidamente con il numero quantico principale n.

Un fenomeno centrale è il Rydberg blockade: se due atomi sono troppo vicini, l'eccitazione di uno impedisce l'eccitazione dell'altro. Questo meccanismo viene usato per realizzare porte a due qubit, come la Controlled-Z, che è un mattoncino fondamentale per la computazione quantistica universale

1) Come si "fanno" i qubit con gli atomi neutri

L'idea è usare **pinzette ottiche**: un fascio laser molto focalizzato crea un piccolo pozzo di potenziale (una trappola) tramite lo **spostamento di Stark AC**. Con luce "red-detuned" (frequenza leggermente più bassa di una risonanza atomica) l'atomo è attratto nel massimo d'intensità: il fuoco del laser diventa una "ciotolina" dove l'atomo si siede.
Si generano **migliaia di trappole** in una volta sola modulando il fronte d'onda con un **SLM** (Spatial Light Modulator) o "spazzolando" il fuoco con **AOD** (deflettori acusto-ottici). Il **caricamento** degli atomi è probabilistico (nuvola fredda → alcune trappole prendono 0/1 atomo): si fa un'**immagine fluorescente** per sapere quali sono piene e poi si **riordina** (rearrangement) spostando gli atomi con pinzette mobili fino a ottenere l'array desiderato (una griglia 2D o una piccola architettura 3D).

Il **qubit** vive in due livelli interni dell'atomo, tipicamente **stati iperfini** "clock" (insensibili al primo ordine a campi magnetici) — per esempio in Rb-87 si usano due sotto-livelli della fondamentale.

- **Giri single-qubit**: si fanno con **microonde** (transizione iperfina diretta) o con **Raman a due fotoni** (due laser lontani dalla risonanza P intermedia per evitare emissione spontanea).
- La **lettura** si fa per **fluorescenza**: illumini in modo stato-selettivo e "vedi" se l'atomo emette ($|1\rangle$) o resta scuro ($|0\rangle$), o viceversa usando tecniche "push-out".

Nota pratica: la **singola occupazione** della trappola durante il loading non è dovuta al Rydberg blockade; è dovuta a **collisioni luce-assistite** che espellono automaticamente la

seconda particella (“collisional blockade”). Il **Rydberg blockade** entra in gioco più avanti, quando ecciti agli stati di Rydberg per far interagire i qubit.

2) Stati di Rydberg e perché “accendono” l’interazione

Gli **stati di Rydberg** sono livelli eccitati con **numero quantico principale** n molto alto (30–100+): il raggio elettronico cresce ~n2n^2n2, i dipoli elettrici diventano enormi e gli atomi **si “sentono” forte**.
Due regimi importanti:

- **Fuori risonanza (van der Waals)**: l’interazione efficace tra due atomi entrambi in uno stato Rydberg scala

$$V(r) = \frac{C_6}{r^6},$$

con C6C_6C6 che **cresce molto rapidamente con n** (approssimativamente ~n^11. Risultato: basta spostare gli atomi di **pochi micron** per cambiare l’accoppiamento di **ordini di grandezza**.

- **Quasi risonante (dipolo-dipolo)**: vicino a una **risonanza di Förster** l’interazione può comportarsi come V∝1/r^3, ancora più lunga portata. In pratica si “sintonizza” la specie/il livello per ottenere il profilo desiderato.

Questa ingegnerizzabilità è l’arma segreta degli atomi neutri: con **geometria (distanze)** e **livello n** controlli **forza e raggio** dell’interazione.

3) Rydberg blockade: l’idea in una riga

Se due atomi sono abbastanza vicini, l’**eccitazione Rydberg** del primo **sposta** (di V(r)) i livelli del secondo **fuori risonanza**: il secondo **non può** essere eccitato *con lo stesso laser*. È come se al secondo mancasse il “canale” per salire allo stesso stato.
Il criterio è:

$$V(r) \gg \hbar\Omega$$

dove Ω è la **frequenza di Rabi** del laser di eccitazione. Definisce un **raggio di blocco** r_b tramite

$$V(r_b) = \hbar\Omega \quad \Rightarrow \quad r_b = \left(\frac{C_6}{\hbar\Omega}\right)^{1/6}.$$

Sotto r_b **doppia eccitazione proibita**; sopra r_b **consentita**. Con n alti e Ω nell'intervallo tipico dei gate, r_b è nell'ordine dei **pochi micron** — perfetto perché le trappole in array hanno spaziature comparabili.

Intuizione: il primo atomo "gonfia" il suo dipolo e **spinge** i livelli del vicino: il laser tarato per l'atomo isolato **non è più alla frequenza giusta** per il vicino, quindi l'eccitazione simultanea viene *bloccata*.

4) Dalla fisica alla porta Controlled-Z (CZ)

Con il blockade, costruire una **porta a due qubit** diventa un gioco di **fasi condizionate**. Sequenza standard (schema di Jaksch et al., molto usato in varianti):

1. **π-pulse sul controllo**: se il controllo è in |1⟩, lo **porti a |r⟩** (stato di Rydberg); se è in |0⟩|0\rangle|0⟩, resta dov'è.
2. **2π-pulse sul target**:
    - Se **il controllo è in |r⟩**, il **blockade** impedisce al target di andare/ritornare da |r⟩: **il target non evolve**, quindi **non accumula fase**.
    - Se **il controllo non è in |r⟩**, il target esegue un **giro completo 2π** su |r⟩ e ritorna allo stato di partenza ma con una **fase geometrica** (tipicamente −1 su |1⟩, a seconda del dettaglio della base logica).
3. **π-pulse di de-eccitazione** sul controllo: |r⟩→|1⟩

Risultato netto: **solo la base |11⟩**prende una **fase −1** (o viceversa a seconda delle convenzioni), cioè una **CZ**. Con rotazioni locali la converti in **CNOT**. Tutto dipende da **Ω, detuning, durata degli impulsi, fase dei laser** e dal fatto che **il blockade sia "profondo"** (errore di *leakage* se V non è abbastanza grande).

5) Connettività nativa e grafi del problema

Poiché l'interazione dipende dall'**inter-distanza**, puoi "**disegnare**" il **grafo di interazione mettendo gli atomi** nella geometria voluta:

- Per **ottimizzazione combinatoria** (MAX-independent set, QUBO/Ising…), colleghi con archi gli atomi che **non devono** essere eccitati insieme; scegli una spaziatura **sotto rbr_brb** sugli archi per avere **forte penalità** (blockade), **sopra rbr_brb** quando vuoi **indipendenza**.
- Per **gate-model digitale**, scegli distanze tali da ottenere **coppie "forti"** (facili CZ) e **croci-parlare minimo** con il resto.

Questa **connettività programmabile 2D/3D** riduce drasticamente i **gate SWAP** rispetto a piattaforme con reti fisse: meno "trasporto logico", meno profondità di circuito, **meno errori**.

6) "Numeri che contano" (senza fissarli alla seconda cifra)

- **Spaziatura tipica** delle trappole: **3–10 µm**.
- **Raggi di blocco** per n attorno a 50–70: **pochi µm** (sintonizzabili via n e intensità).
- **Tempi di gate Rydberg**: **centinaia di ns → pochi µs**; single-qubit **molto più rapidi/robusti** (microonde/Raman).
- **Lifetime Rydberg** scala ~n^3 (limitano la fedeltà); gli **stati di base** iperfini hanno coerenze molto più lunghe (fino a secondi in condizioni buone).
- **Error budget** per CZ: **imperfetto blockade** (V non infinito), **decoerenza Rydberg** (decadimento/BBR), **Doppler** residuo, **rumore di fase dei laser**, **fluttuazioni d'intensità** e **campi elettrici parassiti** (Stark).

L'arte ingegneristica sta nel **trade-off**: Ω alta velocizza il gate ma alza il requisito $V \gg \hbar\Omega$; salire in n aumenta C_6 ma **accorcia** la vita Rydberg e lo **rende più sensibile** a campi parassiti.

7) Due modi di "usare" l'interazione: digitale e analogico

- **Digitale (gate-based)**: si eseguono sequenze precise di π/2, π, 2π, ecc. su atomi selezionati; si ottengono **CZ** ad alta selettività spaziale e temporale.
- **Analogico (quantum simulation/Ising)**: si eccitano/dressing più atomi insieme e si lascia che l'**Hamiltoniano fisico** (con i suoi $J_{ij} \sim C_6/r^6$ evolva lo stato verso un **minimo energetico** che rappresenta la soluzione (es. MAX-Independent-Set). È molto efficiente per certe classi di problemi.

Molti workflow moderni sono **ibridi**: mappano problema→grafo, regolano geometria e parametri, fanno "anneal" analogico o piccole sequenze digitali, e **post-elaborano** classicamente.

9) Riassunto operativo della CZ (passo-per-passo, vista "ingegnere")

1. **Preparazione**: scegli due atomi a distanza $r < r_b$.
2. **Selezione**: indirizza il **controllo** con un fascio locale per la transizione $|1\rangle \leftrightarrow |r\rangle$
3. **π-pulse** controllo: $|1\rangle \rightarrow |r\rangle$ (se era $|1\rangle$).
4. **2π-pulse** target: se il controllo è in $|r\rangle$ il target **non** può salire (blockade) → niente fase; se il controllo *non* è in $|r\rangle$, il target fa il giro completo e accumula la fase.

5. **π-pulse** di ritorno sul controllo: $|r\rangle \to |1\rangle$.
6. **Compensazioni locali**: piccole rotazioni per rifasare le basi e ottenere esattamente la **CZ** desiderata.
7. **Mitigazioni**: scegli Ω, Δ, e la durata degli impulsi per **minimizzare leakage e decoerenza**; compensa **Doppler** (atomi ben raffreddati, "magic trapping", geometrie di fasci contro-propaganti).

Appendice D – Snippet

Di seguito l'esempio visto a lezione:

```
import pennylane as qml
import numpy as np
# Define neutral atom device
dev = qml.device("default.qubit", wires=4)

# Define variational circuit
@qml.qnode(dev)
def variational_circuit(params):
  # Prepare initial state
  for i in range(4):
    qml.Hadamard(wires=i)

  # Variational layers
  qml.StronglyEntanglingLayers(params, wires=range(4))

  # Measurement
  return qml.expval(qml.PauliZ(0) @ qml.PauliZ(1))

# Optimize parameters
params = np.random.uniform(0, 2*np.pi, (2, 4, 3))
```

1. **Dispositivo.** Viene creato un *simulatore* (`"default.qubit"`) con 4 qubit. Quindi lo snippet gira in locale, non su un QPU a atomi neutri. Nella lezione si sottolinea che in pratica si "alloca un device" e poi si crea il circuito—con PennyLane o con backend specifici come Pulser/Braket/Azure; qui è solo un esempio minimale.
2. **QNode.** `@qml.qnode(dev)` trasforma la funzione Python in un circuito eseguibile sul device scelto: dentro ci sono **preparazione**, **strati variazionali**, **misura**.
3. **Stato iniziale.** Quattro **Hadamard** creano $|+\rangle^{\otimes 4}$: è proprio lo *starting state* standard per QAOA (superposizione uniforme), che a lezione veniva introdotto nel flusso "cost–mix–measure–update".
4. **Strato variazionale.** StronglyEntanglingLayers è un *template* di PennyLane: per ogni *layer* applica rotazioni a 3 parametri per qubit (la terza dimensione del tensore params è 3) e una rete di entanglement (tipicamente una catena di CNOT). La forma di params (L, n_wires, 3) indica: **L = 2 layer**, **4 qubit**, **3 rotazioni per qubit**.
5. **Misura.** `qml.expval(qml.PauliZ(0) @ qml.PauliZ(1))` restituisce l'**attesa del correlatore Z_0Z_1**: è **un solo termine** di una tipica funzione di costo stile Ising (come MaxCut). In un QAOA reale si misura l'**intera Hamiltoniana di costo**, somma pesata di molti termini Z_iZ_j. Nella lezione si spiegava che QAOA alterna *cost layer* e *mixer* e che le librerie forniscono già queste componenti.
6. **Inizializzazione dei parametri.** np.random.uniform(0, 2π, (2,4,3)): parametri casuali per 2 layer. In pratica poi li si **ottimizza** con un *optimizer* classico (gradient descent, Adam, SPSA ecc.) come discusso in lezione.

Cosa non fa (ed è bene sapere)

- Non c'è **ciclo di ottimizzazione** (nessun `Optimizer.step` né *loop* di training): nella lezione l'idea era proprio "misura costo → aggiorna parametri → ripeti p volte".
- Non costruisce un **Hamiltoniano di costo completo** (per MaxCut, p.es. $\sum_{(i,j)\in E}((1-Z_iZ_j)/2)$ misura solo un termine Z_0Z_1. In pratica si sommano tutte le attese dei termini rilevanti del grafo definito.
- Il commento "Define neutral atom device" è fuorviante: il device è un **simulatore generico**. Per un QPU di **atomi neutri** serve un **backend appropriato** e la relativa **compilazione** verso il set di gate/impulsi nativi, cosa che la lezione ha messo in scaletta (compilazione → esecuzione)

Come si porta questo su atomi neutri (correttamente)

1. **Scegli il device giusto** in PennyLane (o usi **Pulser** quando vuoi controllare gli impulsi): stessa logica di "allocare un device" vista in aula. Imposti anche i **shots** (serve campionare, non c'è stato perfetto)
2. **Entanglement nativo.** I Rydberg implementano in modo naturale **CZ** (via blockade); i template che usano **CNOT** vengono **traspile** in CZ + rotazioni a 1 qubit (CNOT = $(I\otimes H)\cdot CZ\cdot(I\otimes H)$). La catena di entanglement del template va **adattata alla connettività fisica** (chi è entro raggio di blocco). Questo è esattamente il punto su cui il docente ha insistito: **mappare il grafo del problema nella geometria** per ridurre SWAP e crosstalk.
3. **Costo completo.** Per il piccolo grafo dell'esempio (anello 0–1–2–3–0 più la corda 0–2 citata a voce) costruisci H_C sommando **tutti** i Z_iZ_j degli archi, non solo Z_0Z_1. PennyLane ha già funzioni/templati QAOA che preparano automaticamente *cost layer* e *mixer* sul grafo dato (come ricordato in lezione)
4. **Ottimizzazione ibrida.** Lanci un *loop* classico: esegui il circuito, **valuti $\langle H_C\rangle$**, aggiorni $\{\gamma,\beta\}$ e ripeti. È lo schema "cost–mix–measure–update".
5. **Compilazione e tempi reali.** La parte "hardware compilation for neutral atoms" significa: scegliere **quali coppie** far interagire (entro raggio di blocco), generare **impulsi Rydberg** π–2π–π per le CZ, programmare **addressing**/sequencing, e rispettare i **vincoli di timing** per stare dentro la **coerenza**

Se volessimo "sistemare" lo snippet per QAOA

- **Device**: usare un backend compatibile e impostare `shots`.

- **Costo**: creare H_C dal **grafo** (gli archi detti a lezione) e misurarne l'intera attesa, non solo Z_0Z_1 .
- **Template**: sostituire *StronglyEntanglingLayers* con i layer **QAOA** (*cost layer* + *mixer*) oppure mantenere il template ma assicurarsi che l'entanglement rispetti la **mappa fisica**.
- **Training loop**: aggiungere un ottimizzatore (come nella descrizione "gradient descent" citata) e iterare finché la **funzione di costo** non converge.