Tools for Utility Beyond the SDK: Qiskit Addons

Qiskit Functions

軟体开發工具包

Qiskit SDK

software development Kit

Quantum Physicists

針对 physicist 設計

Use performant Qiskit SDK to run more and larger circuits

?

Quantum Computational Scientists

need 額外开發工具&技能

將底層量子电路

轉化為可用的工具

Stack capabilities to design new algorithms

Scientists

探索演算法

&

开發应用的原型

Discover algorithms and prototype applications

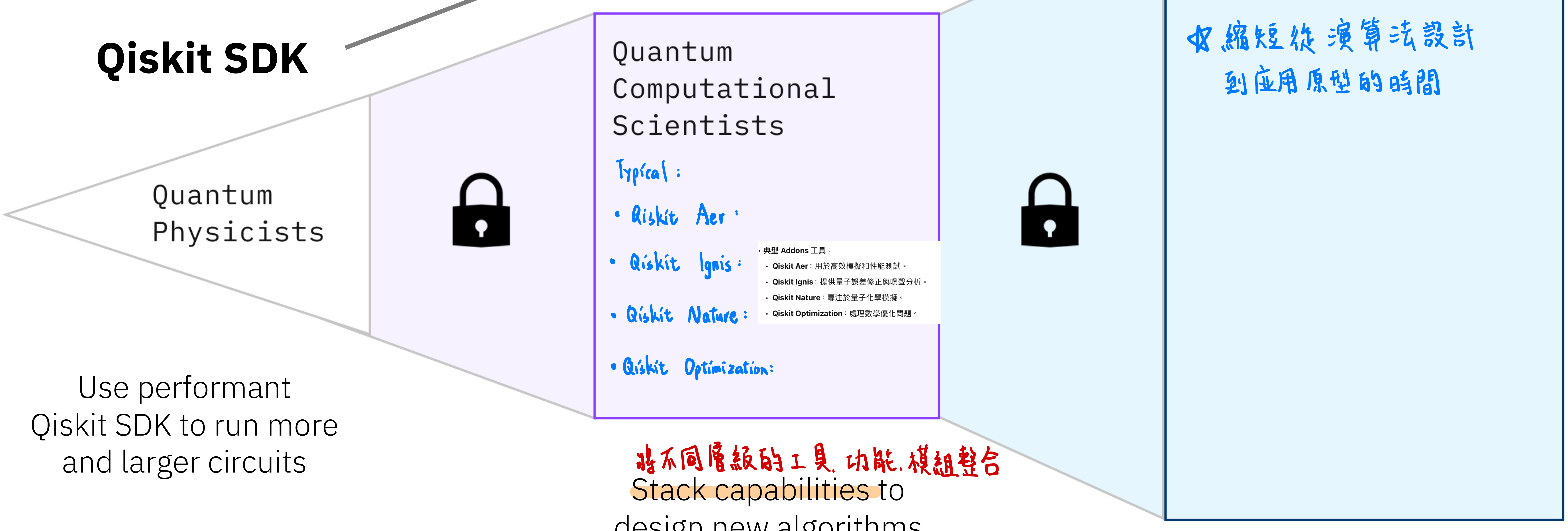
Qiskit Addons 是量子計算應用生態系的關鍵橋樑：

- **SDK 的強化工具**：通過 Addons，SDK 的功能可以得到進一步擴展，支援更多高階應用需求。
- **應用開發的催化劑**：Addons 幫助研究人員設計新演算法並連接至實際應用場景。

這張圖表清楚地展示了 Addons 的重要地位，使量子計算技術從基礎研究轉化為應用落地成為可能。

Qiskit Functions

Qiskit addons



Use performant Qiskit SDK to run more and larger circuits

將不同層級的工具、功能、模組整合

Stack capabilities to design new algorithms

Discover algorithms and prototype applications

Qiskit Functions

Pre-packaged workflows as abstracted services designed to accelerate **algorithm discovery** and **application prototyping** through advanced mapping and performant management techniques

Application function

↓
Domain-specific inputs

1. **Map (映射)**
   - **作用**：將問題轉化為適合量子計算的表示形式。
   - **輸入**：來自特定應用領域的數據 (Domain-specific inputs)。
   - **功能**：
     - 將經典問題 (如優化問題、化學模擬、機器學習模型) 映射到量子比特和電路結構。
     - 利用 Addons 工具 (如 Qiskit Nature 或 Qiskit Optimization) 完成問題建模。
2. **Optimize (優化)**
   - **作用**：優化量子電路和資源分配。
   - **功能**：
     - 縮減電路深度以減少噪聲影響。
     - 選擇適當的量子演算法或變分量子演算法 (如 VQE 或 QAOA)。
     - 提供執行前的電路調整和資源最佳化。
3. **Execute (執行)**
   - **作用**：在模擬器或實際量子硬體上運行電路。
   - **功能**：
     - 整合 Qiskit Aer (模擬器) 和 IBM Quantum 硬體，根據需要執行量子電路。
     - 收集執行結果，包括量子態和測量數據。
4. **Post-process (後處理)**
   - **作用**：處理和分析執行結果。
   - **輸出**：經過處理的數據與可視化結果 (Data/visualizations)。
   - **功能**：
     - 解碼量子測量結果並將其轉換回實際應用場景。
     - 生成波形圖、頻譜圖或其他可視化工具，協助結果解讀。

lizations

Map → Optimize → Execute → Post-process

Qiskit SDK sets the foundation for quantum workflows

↓
Domain-specific inputs

1. 整體架構

- **標題**：Qiskit SDK 設立了量子工作流程的基礎。
- **流程四階段**：**Map（映射）→ Optimize（優化）→ Execute（執行）→ Post-process（後處理）**。
- 每一階段都具體標示了所使用的 Qiskit SDK 功能模組。

---

2. 分階段分析

1. **Map（映射）**
   - **功能**：將特定領域的輸入數據轉化為量子電路。
   - **模組**：
     - **Circuit Library（電路庫）**：提供一組預建的量子電路，針對常見的量子操作或演算法（如 QAOA、VQE）進行快速構建。
   - **應用**：
     - 將化學分子、優化問題或機器學習模型轉換為量子比特的邏輯結構。

2. **Optimize（優化）**
   - **功能**：減少電路運行資源並優化運算效率。
   - **模組**：
     - **Transpiler（轉譯器，含 AI 支持）**：負責將高層電路轉化為適合硬體執行的低層電路。
     - **AI 支援**：使用機器學習方法來選擇最佳優化策略。
   - **應用**：
     - 減少量子閘深度，減低量子誤差。
     - 提供針對不同硬體的量身定制電路。

3. **Execute（執行）**
   - **功能**：在量子硬體或模擬器上運行電路。
   - **模組**：
     - **Primitives（基元操作）**：一組核心的基礎運算工具（如測量、狀態準備），用於簡化硬體調用。
   - **應用**：
     - 執行量子模擬，並收集測量數據。
     - 適配不同類型的量子計算資源（如 IBM Quantum 硬體或 Qiskit Aer 模擬器）。

Data / visualizations
↑

4. **Post-process（後處理）**
   - **功能**：分析執行結果並進行誤差修正。
   - **模組**：
     - **Fitting（數據擬合）**：從測量數據中提取結果，將量子結果轉換為實際應用數據。
     - **Error Mitigation（誤差緩解）**：應用誤差校正技術來提高結果的可靠性。
   - **應用**：
     - 分析量子計算結果，生成可視化圖表（如波形或頻譜圖）。
     - 減少硬體噪聲對計算結果的影響。

---

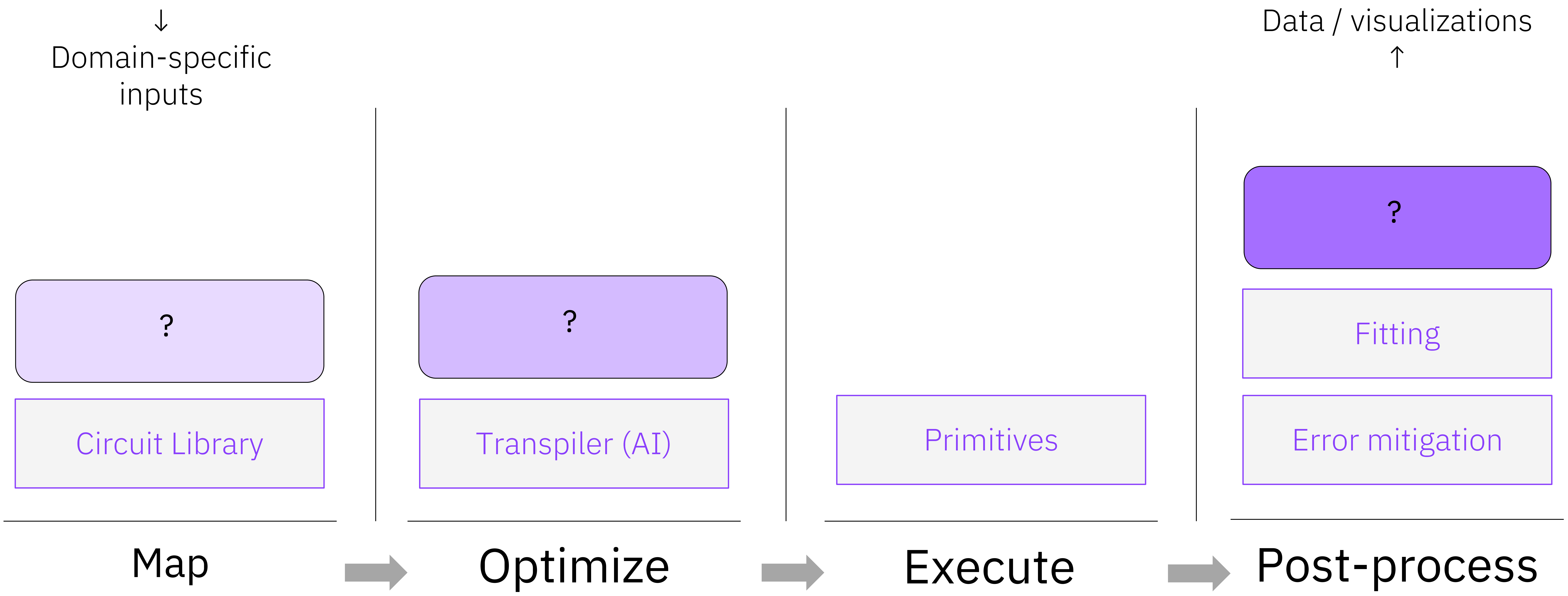
3. 核心信息

- **SDK 的基礎角色**：Qiskit SDK 作為所有量子工作流程的核心基礎，提供從電路構建到結果分析的完整工具鏈。
- **模組化架構**：每個階段都對應特定的功能模組，讓用戶可以靈活地選擇適合需求的工具。
- **應用驅動**：這些工具和功能模組旨在支援不同領域的實際應用，促進量子技術的落地。

Map	Optimize	Execute	Post-process
Circuit Library	Transpiler (AI)	Primitives	Fitting Error mitigation

Map → Optimize → Execute → Post-process

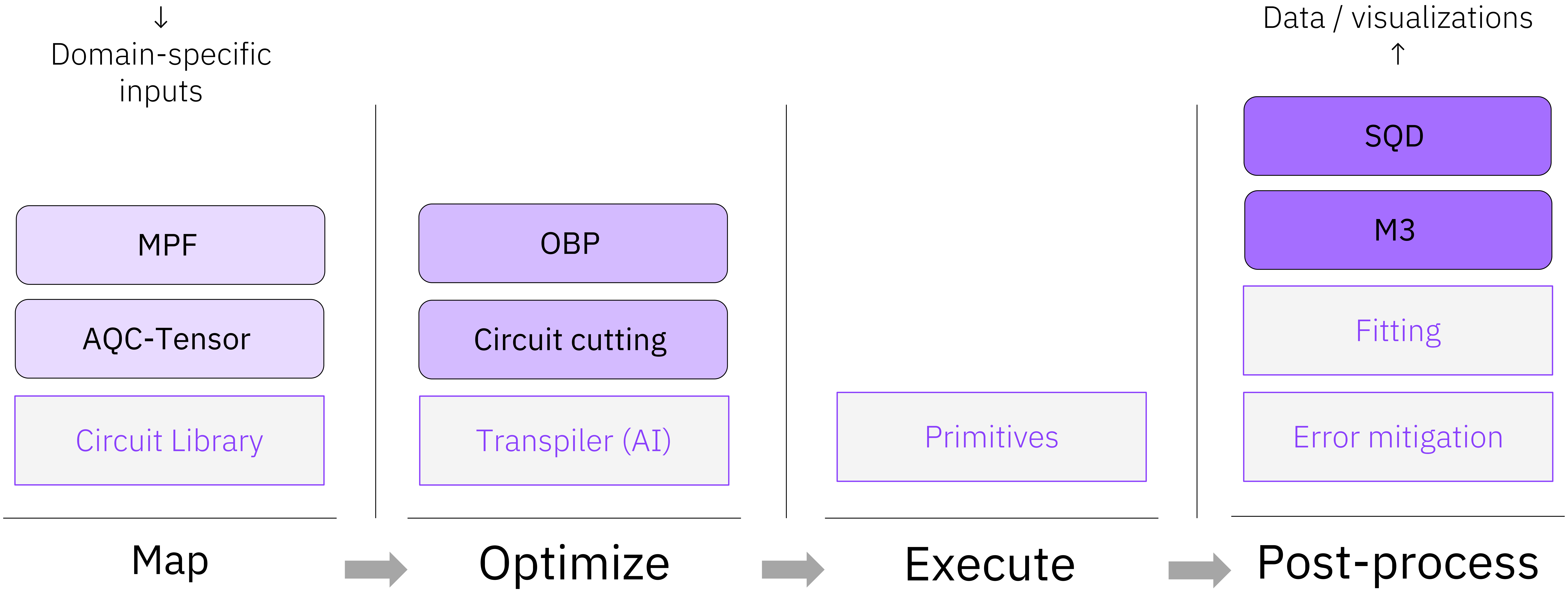
Qiskit SDK sets the foundation for quantum workflows



Qiskit addons build on the Qiskit SDK

A collection of research capabilities developed as modular tools that can plug into a workflow to scale or design new algorithms at the utility scale.

Starting with **multi-product formulas** (MPF), **approximate quantum compilation** (AQC-Tensor), **operator backpropagation** (OBP), and **sample-based quantum diagonalization** (SQD)

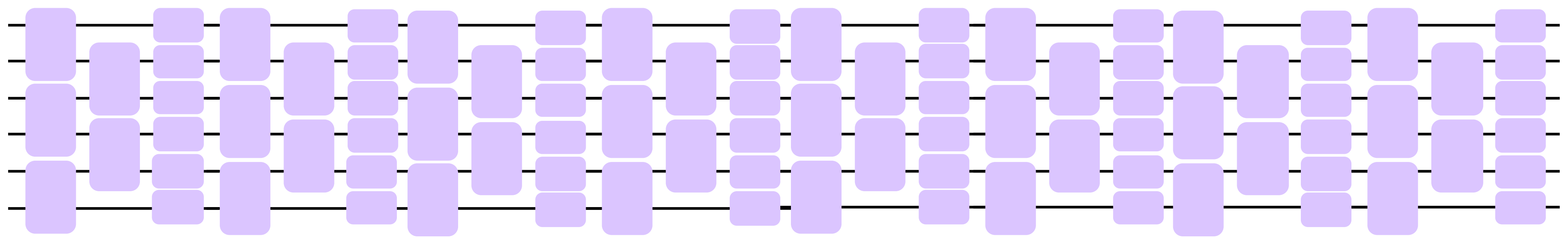


Multi-product formulas

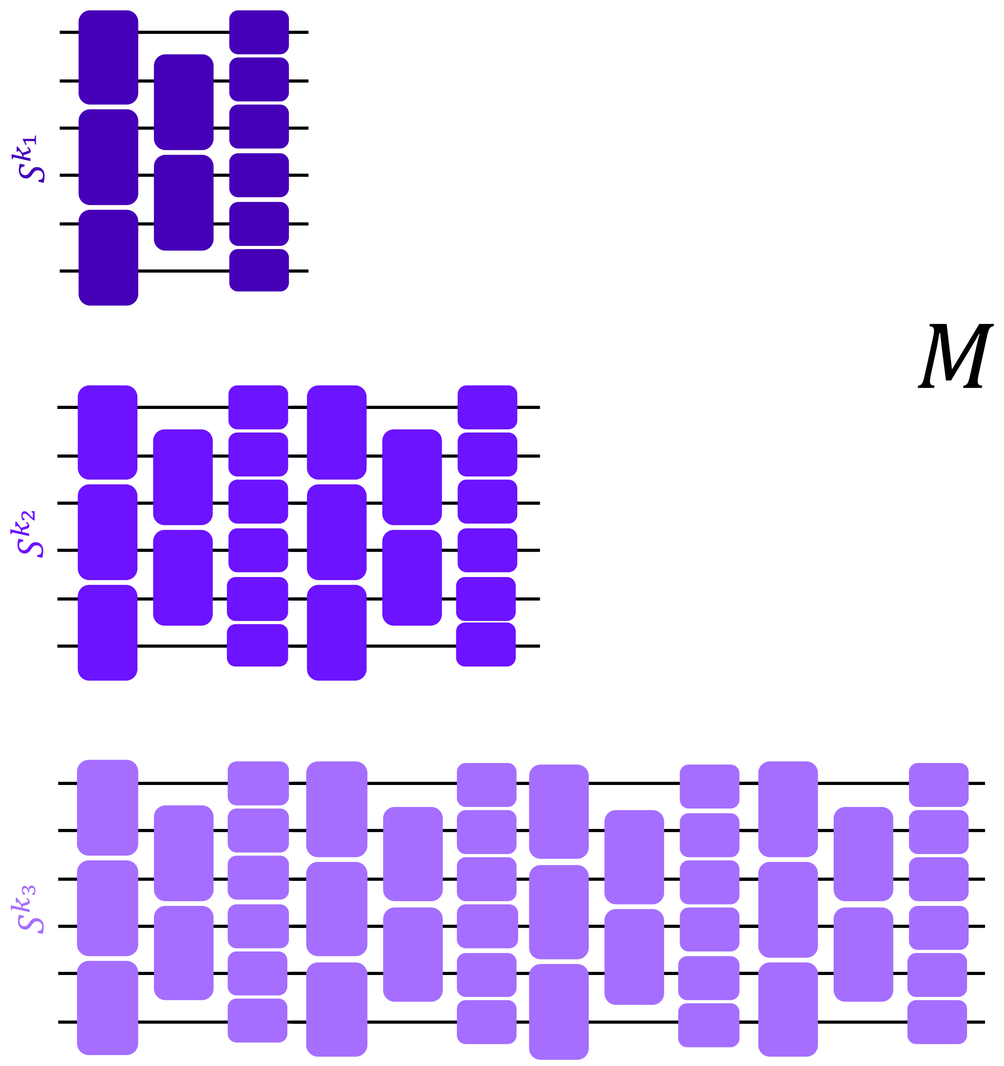
A technique to reduce algorithmic (Trotter) errors through a weighted combination of several circuit executions

- Reducing Trotter error can require deep circuits
- MPF combines experiments with different Trotter errors to produce an estimate with lower overall Trotter error
- Can leverage TN methods to further reduce Trotter error
- Demonstrated on
  - 50 qubits
  - 27 2q-depth

time split 時間切割 (每段都有 Trotter error)



Replace a single product formula with multi-product formula of shorter depth



number of Trotter steps

fix

$$M(t) = \sum_{j=1}^{l} a_j S^{k_j}\left(\frac{t}{k_j}\right)$$

加权 時間分割

maybe → linear combination

Domain inputs

MPF: Pre-process

Map

Many circuits, observable

Transpile circuit

Optimize

Many ISA circuits, observable

Estimator primitive

Execute

Many expectation values

MPF: Post-process

Post-process

Expectation value

Research

Quantum 7, 1067 (2023)
arxiv.org/abs/2407.17405
arxiv.org/abs/2306.12569

Development

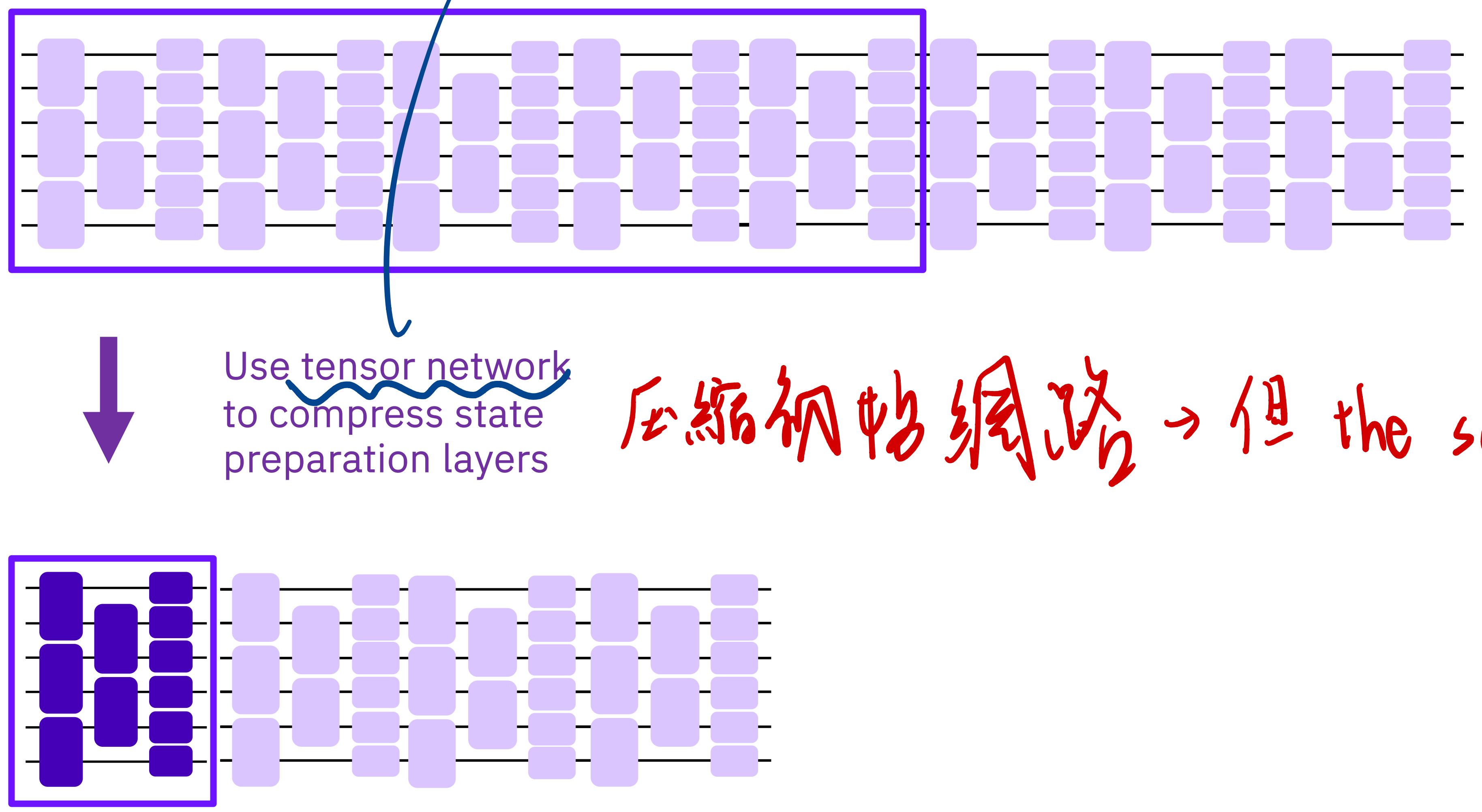
github.com/Qiskit/qiskit-addon-mpf

Approximate quantum compilation with tensor networks

A technique to produce shorter-depth Trotter circuits for time evolution problems by classically compressing the initial layers

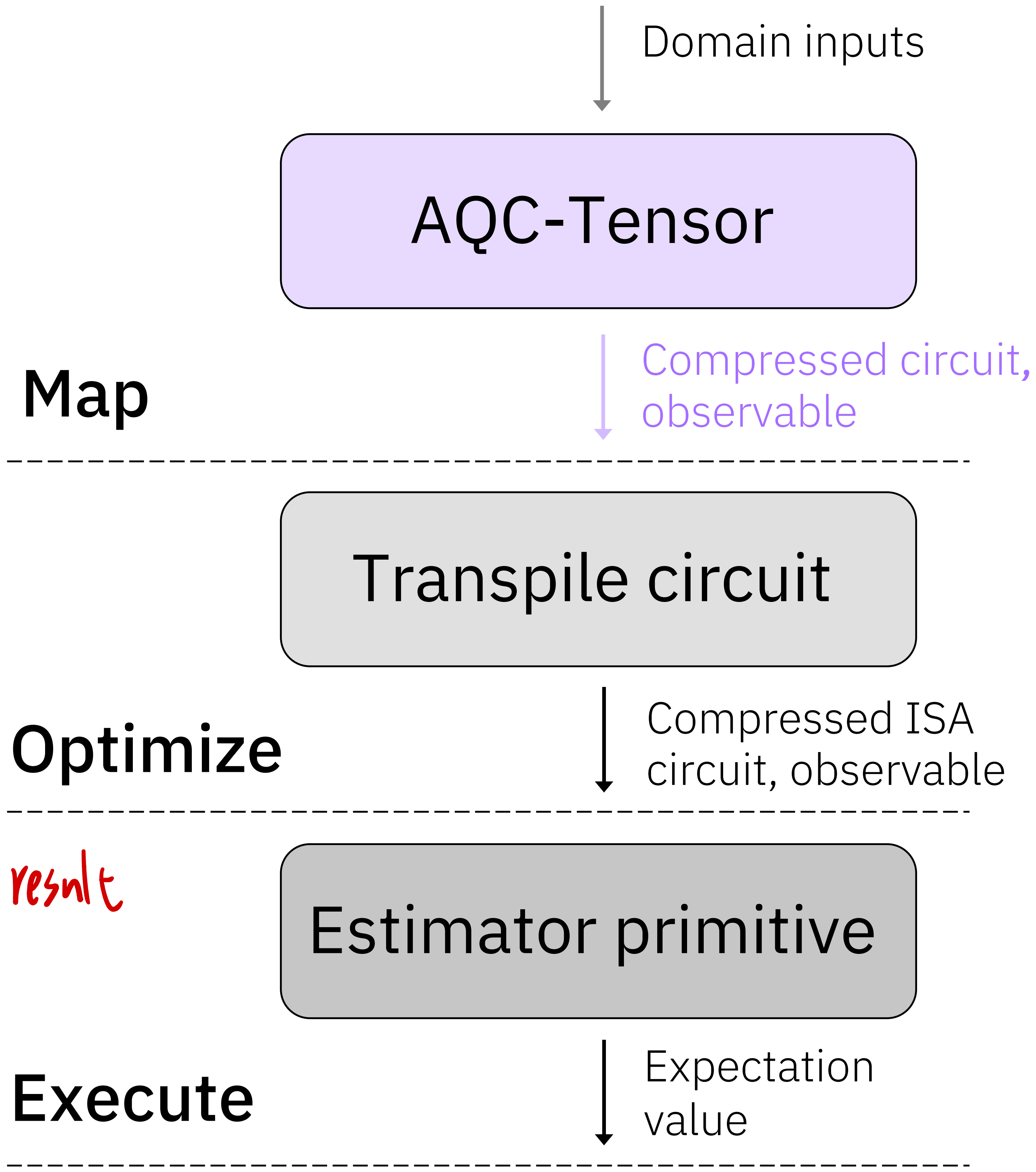
- Trotterized time evolution is often classically tractable for small times, but evolving further becomes classically hard
- AQC uses a tensor network to compress initial circuit layers, allowing more circuit depth to be spent on further time evolution
- Demonstrated on
  - 50 qubits
  - 27 2q-depth

可 choose mps



压縮張量網路 → 但 the same result

★ 節省 circuit depth



Research → **Development**

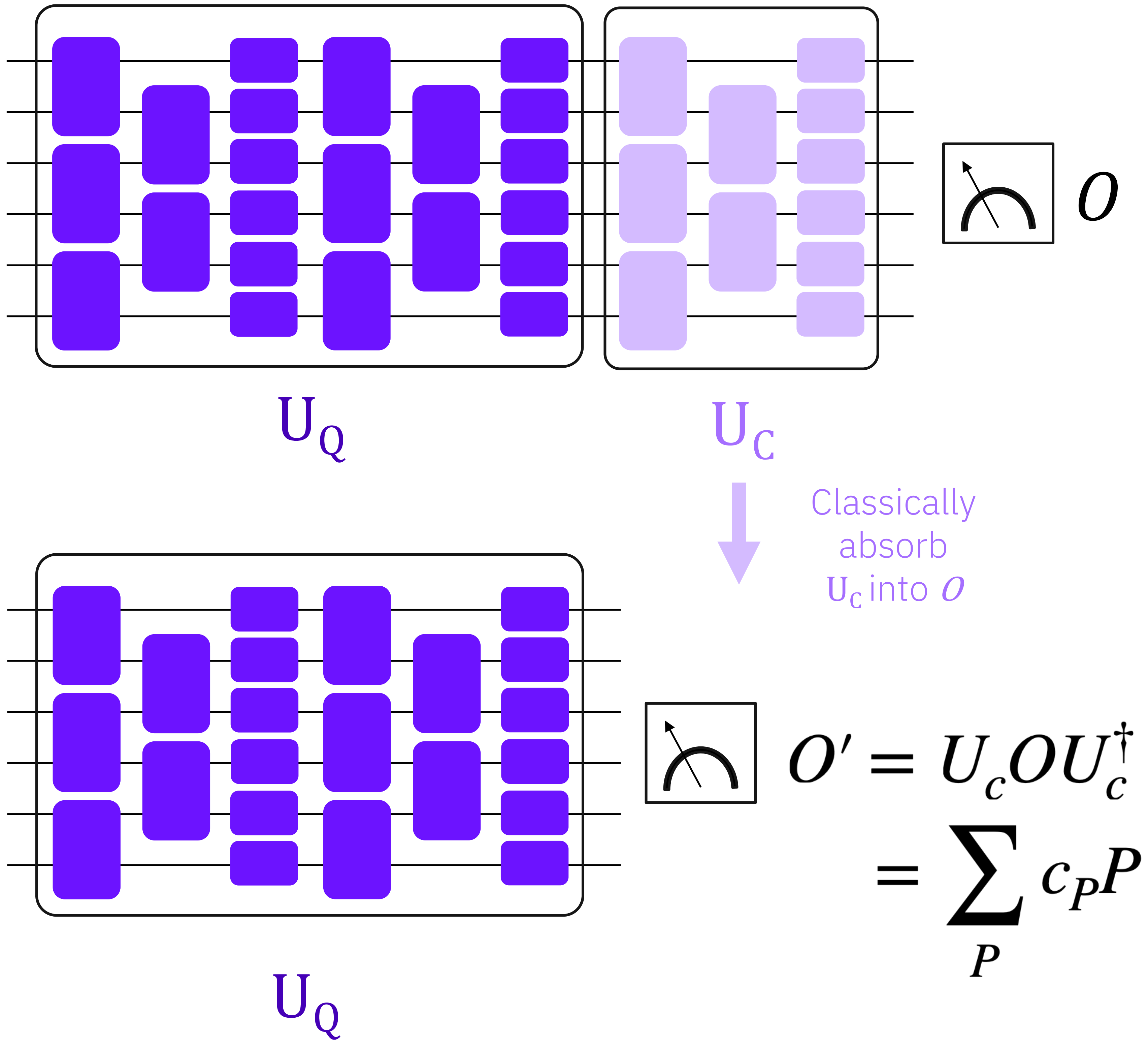
arxiv.org/abs/2301.08609
arxiv.org/abs/2407.17405

github.com/Qiskit/qiskit-addon-aqc-tensor

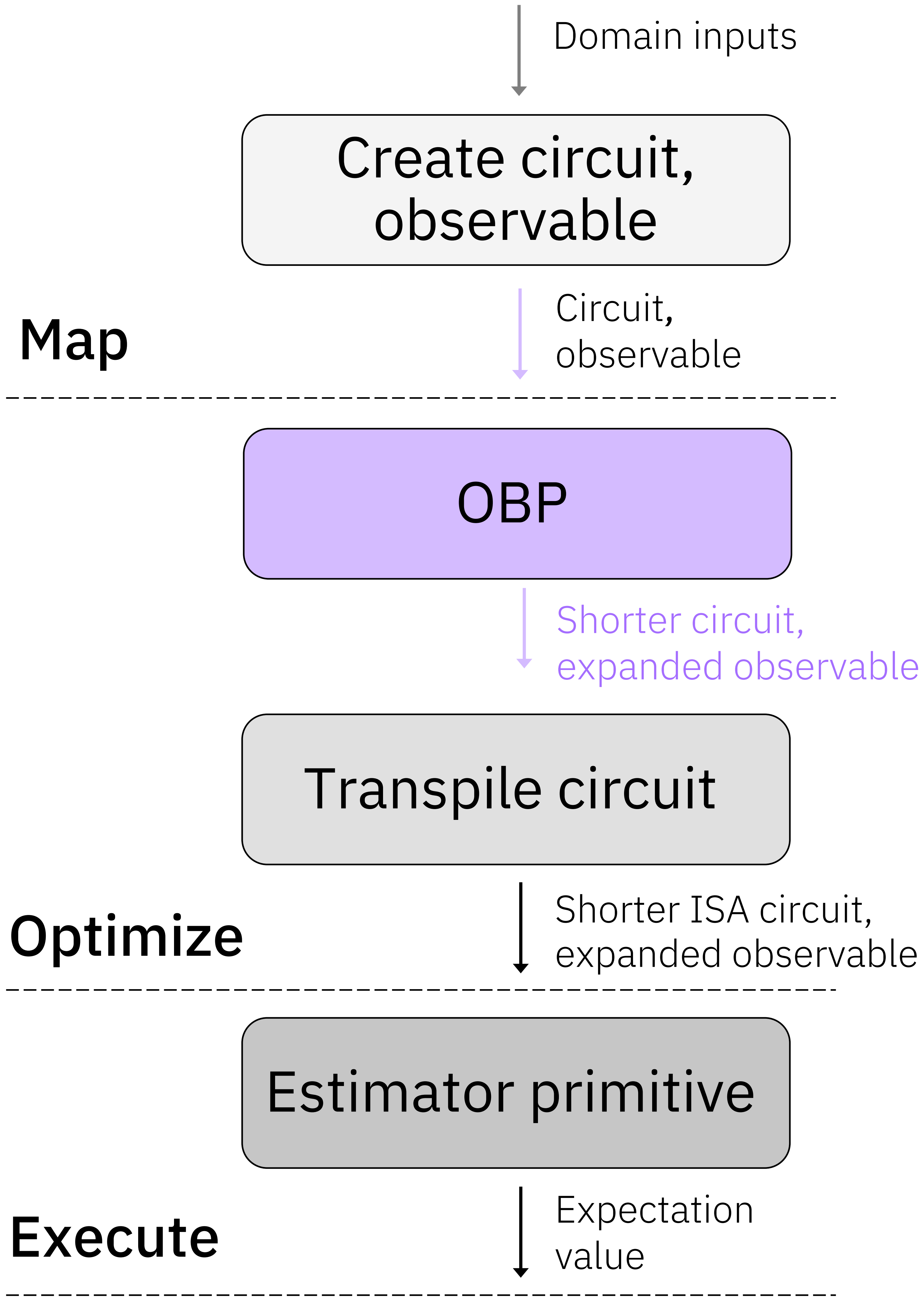
Operator backpropagation → Circuit depth ↓ → noise ↓

A technique to reduce circuit depth by trimming operations from the end at the cost of more operator measurements.

- Some circuits, such as Trotterized time-evolution, get deeper and nearer to Clifford as they become more accurate
- OBP can reduce maximum depth of these circuits, reducing impact of noise
- Demonstrated on
  - 127 qubits
  - 82 2q-depths



$$O' = U_c O U_c^\dagger$$
$$= \sum_P c_P P$$



Research → **Development**

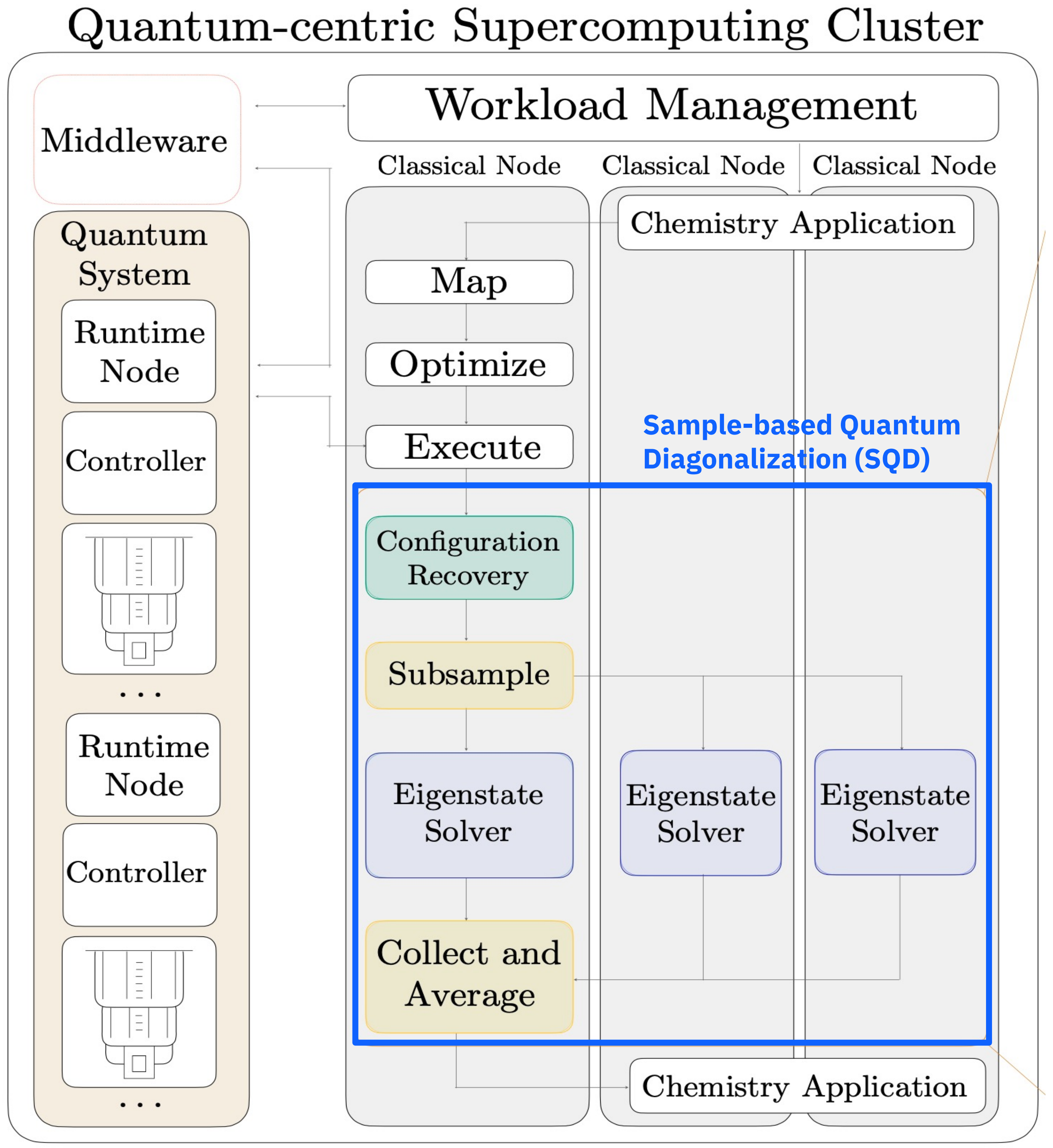
In preparation

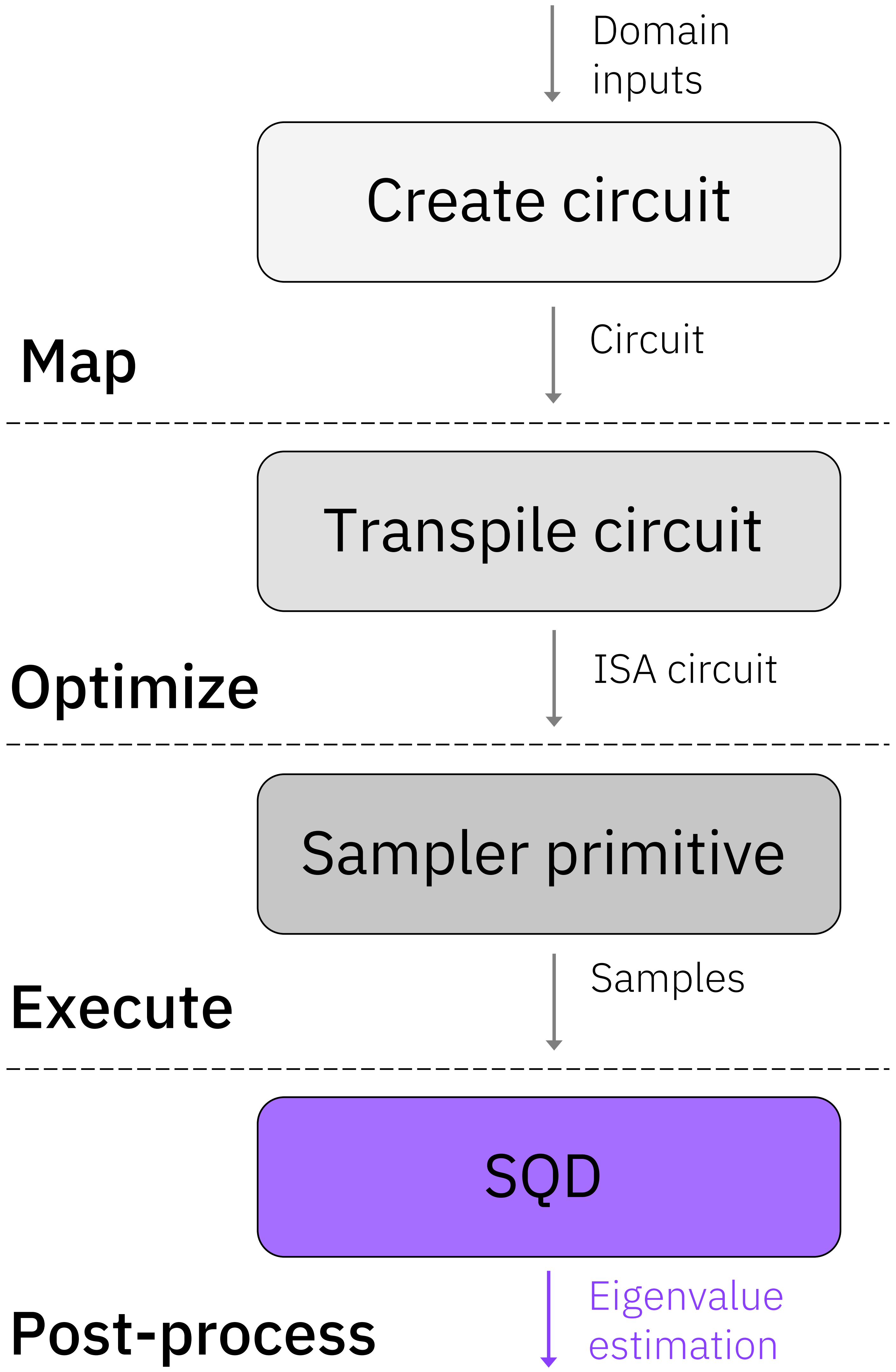
github.com/Qiskit/qiskit-addon-obp

Sample-based quantum diagonalization

A technique using classical distributed computing to produce more accurate eigenvalue estimations from noisy quantum samples

- Energy estimation on current hardware is limited by circuit depth, cost of error mitigation, overhead of evaluating expectation values
- SQD refines noisy samples with classical distributed computing to address large Hamiltonians
- Demonstrated on
  77-qubit chemistry Hamiltonian
  3590 CNOTs





Research → **Development**

arxiv.org/abs/2405.05068
arxiv.org/abs/2410.09209

github.com/Qiskit/qiskit-addon-sqd

Qiskit addons

- Advanced research capabilities made available as easy-to-use modular software tools

- Designed to plug into workflows to build new algorithms at utility scale

Qiskit addons workshop

- Pick one or more of the following addons:
  - Multi-product formulas (MPF)
  - Approximate quantum compilation with tensor networks (AQC-Tensor)
  - Operator backpropagation (OBP)
  - Sample-based quantum diagonalization (SQD)
- Download and go through the tutorial
- Note any questions, issues, or feedback that you have
- Reconvene and discuss

https://docs.quantum.ibm.com/guides/addons

IBM

Code example for the OBP addon

```python
# STEP 1: Map
# define Hamiltonian time evolution problem

circuit = get_circuits_for_experiment( ... )
observable =  ...

# STEP 2: Optimize
# transpile the final circuit
transpiled_circuit = transpile( ... )

# STEP 3: Execute
job = estimator.run(transpiled_circuit, observable)
result = job.result()

# STEP 4: Postprocess
final_expval = result[0].data.evs
```

```python
# STEP 1: Map
# define Hamiltonian time evolution problem

circuit = get_circuits_for_experiment( ... )
observable =  ...

# STEP 2: Optimize
# slice circuit into layers for backpropagation
from qiskit_addon_utils.slicing import slice_by_gate_types
slices = slice_by_gate_types(circuit)

# specify an operator budget for backpropagation
from qiskit_addon_obp.utils.simplify import OperatorBudget
op_budget = OperatorBudget(max_qwc_groups=10)

# OBP: Backpropagate slices onto the observable
from qiskit_addon_obp import backpropagate
bp_obs, remaining_slices, metadata = backpropagate(observable, slices, op_budget)

from qiskit_addon_utils.slicing import combine_slices
bp_circuit = combine_slices(remaining_slices)

# transpile the final circuit
transpiled_circuit = transpile( ... )

# STEP 3: Execute
job = estimator.run(transpiled_circuit, bp_obs)
result = job.result()

# STEP 4: Postprocess
final_expval = result[0].data.evs
```