①量子・HPC連携のための遠隔手続き呼び出しシステムソフトウエアの研究開発
②量子・HPC連携プログラミング環境の研究開発

理化学研究所・計算科学研究センター
量子HPC連携プラットフォーム部門・量子HPCソフトウェア環境開発ユニット
辻美和子
2025/2/4計算科学コロキウム「HPCと量子コンピューティング」

RIKEN
Center for
Computational Science

開発体制

理化学研究所・計算科学研究センター

辻美和子
研究統括＆
ソフトウエア開発

部門長
佐藤 三久
遠隔手続き呼び出し・プログラミングモデル

副部門長
児玉 祐悦
スケジューラ

公募↓

研究員
Maxence Vandromme
Hybrid AI

特別研究員
Soratouch Pornmaneerattanatri
Quantum SDK

- 東京大学：マルチプラットフォームコスケジューラとシステムソフトウェアの連携
- ソフトバンク：PaaS, ユーザAPIとの連携
- 大阪大学：項目④ライブラリとプログラミング環境のインテグレーション
- 理研：項目⑥シミュレータと富岳上のプログラミング環境のインテグレーション
- …

NEDO外での関連共同研究や連携

- 量子コンピュータバックエンドサーバへのDirect Access API (IBM)
- SDKとワークフロー (Quantinuum)
- ワークフロー (京都橘大学)
- 遠隔手続き呼び出し (順天堂大学)
- …

量子HPC連携プラットフォームのコンセプト

- スーパーコンピュータ、量子コンピュータ、両方のスループットを最大化する
  - 2つの計算資源を同時刻に予約することはしない
    - どちらかが遊んでしまう
  - スーパーコンピュータ、量子コンピュータともにキューは埋まっていた方が良い
    - 量子HPCジョブに加えて、、、
    - スーパーコンピュータだけのジョブもOK
    - 量子コンピュータだけジョブもOK

- HPCから量子コンピュータへのオフロードの待ち時間を最小化する

量子HPC連携プラットフォームのコンセプト

- スーパーコンピュータ、量子コンピュータ、両方のスループットを最大化する
  - 2つの計算資源を同時刻に予約することはしない
    - どちらかが遊んでしまう
  - スーパーコンピュータ、量子コンピュータともにキューは埋まっていた方が良い
    - 量子HPCジョブに加えて、、、
    - スーパーコンピュータだけのジョブもOK
    - **量子コンピュータだけジョブもOK**

矛盾？

- **HPCから量子コンピュータへのオフロードの待ち時間を最小化する**

量子HPC連携プラットフォームのコンセプト

- スーパーコンピュータ、量子コンピュータ、両方のスループットを最大化する
  - 2つの計算資源を同時刻に予約することはしない
    - どちらかが遊んでしまう
  - スーパーコンピュータ、量子コンピュータともにキューは埋まっていた方が良い
    - 量子HPCジョブに加えて、、、
    - スーパーコンピュータだけのジョブもOK
    - **量子コンピュータだけジョブもOK**

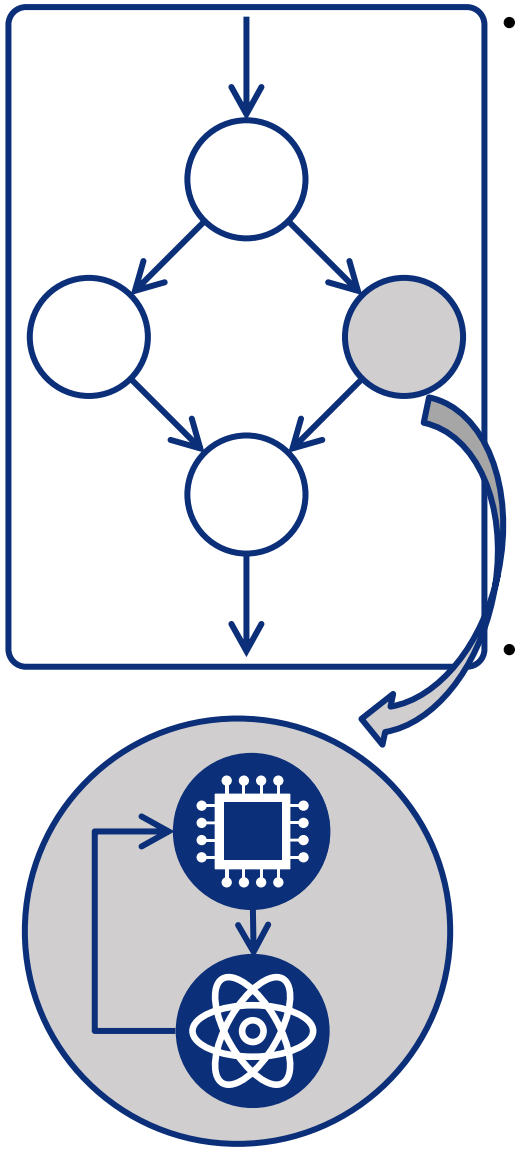
矛盾？

- **HPCから量子コンピュータへのオフロードの待ち時間を最小化する**

2階層のプログラミングモデル

二階層のプログラミングモデル

- ワークフローとタスクプログラミングからなるWorkflow & Task based parallelism



- ワークフロー
  - タスクは、スーパーコンピュータおよび量子コンピュータにおける“ジョブ”
  - タスクは、ワークフローエンジンにより管理され、ジョブの依存関係を考慮した適切な順番でスーパーコンピュータおよび量子コンピュータに投入される
  - ジョブは以下の3種類がある

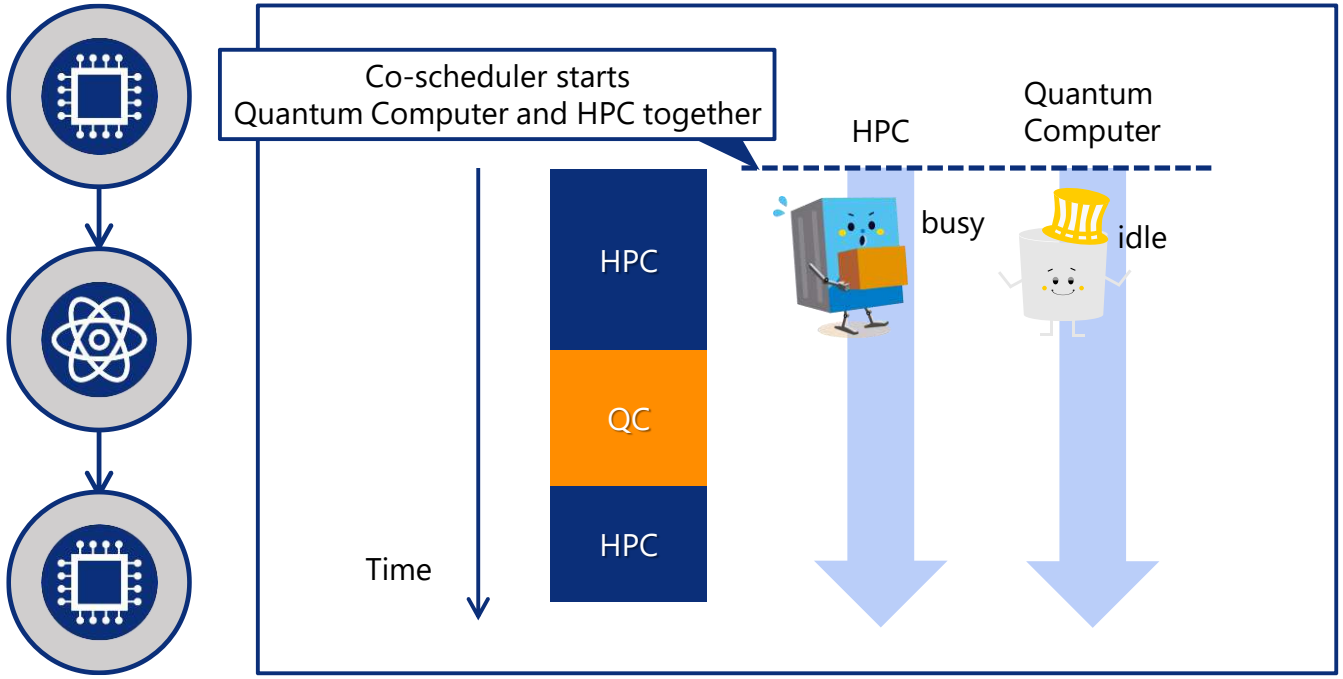
Simple HPC Job	Simple QC Job	HPC-QC Job
HPCのみで実行されるプログラム	量子コンピュータのみで実行されるプログラム	HPCでの計算と量子コンピュータへのオフロードをともなうプログラム

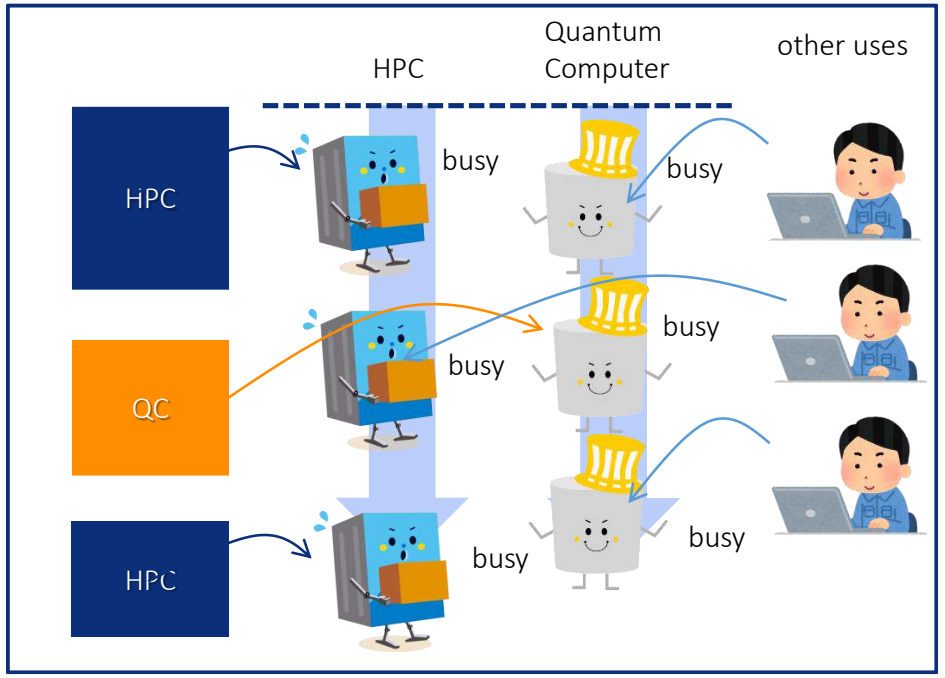
- タスクプログラミング (HPC-QC JOb)
  - スーパーコンピュータのプログラムから、遠隔手続き呼び出し (RPC) により量子コンピュータに回路の実行をオフロードする
  - 単一の HPC-QCジョブは、複数の量子回路のオフロードが可能
  - RPCは非同期のメカニズムであり、量子回路の実行とHPCの計算のオーバラップが可能

なぜ、第一階層ジョブスケジューラを用いるワークフローが必要なのか？

- 量子コンピュータとスーパーコンピュータのアイドル時間を最小化する
  - アプリケーション内で、量子・HPCのカーネルが独立に実行可能なときはできるだけ分離する
  - 実行順序を定義し、それに従ってジョブスケジューラに投入する
- ジョブスケジューラはそれぞれのシステムのポリシーに従い、また、スループットが最大になるよう、ジョブを実行する
- 前のジョブを「待つ」ことになるが、この「待ち」は計算リソースを無駄に消費しない

workflow manager



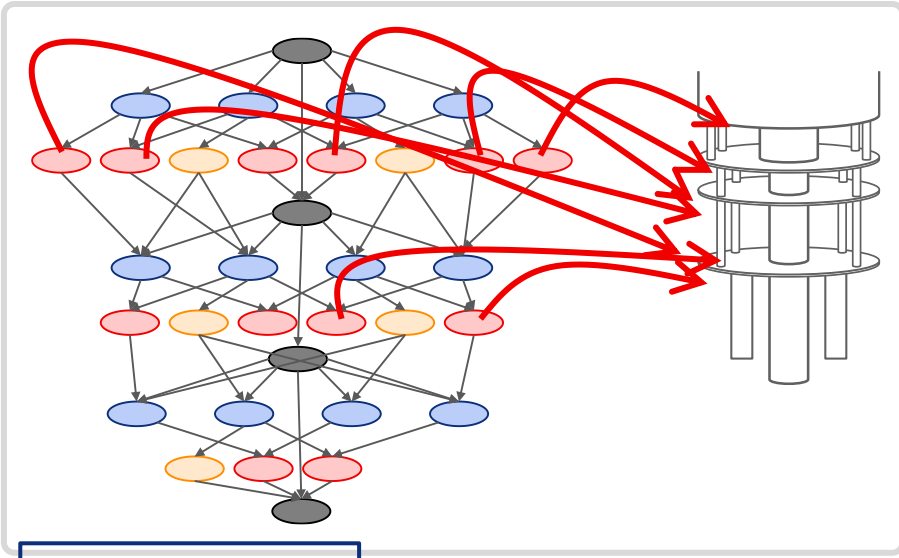


なぜ、第二階層タスクベースプログラミングが必要なのか

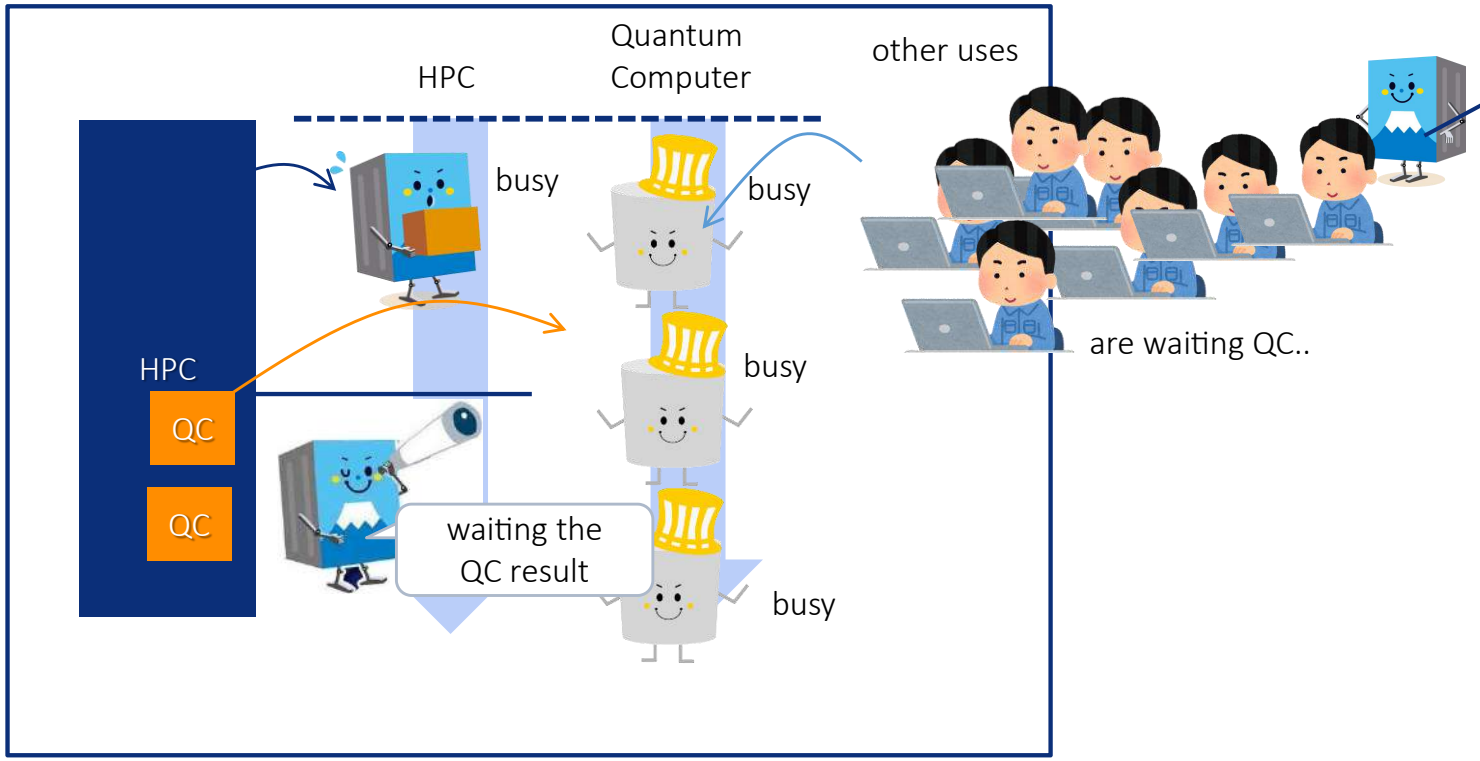
- 量子・HPCが密に連携し分離が難しい場合
- QAOAのように量子回路の繰り返しを含み、各繰り返しごとに
  ジョブとして投入することが現実的でない場合

<u>それでも、HPCのアイドル時間は最小化しなければならない</u>

- 特に大規模なHPCプログラムの場合



20～30MW/h

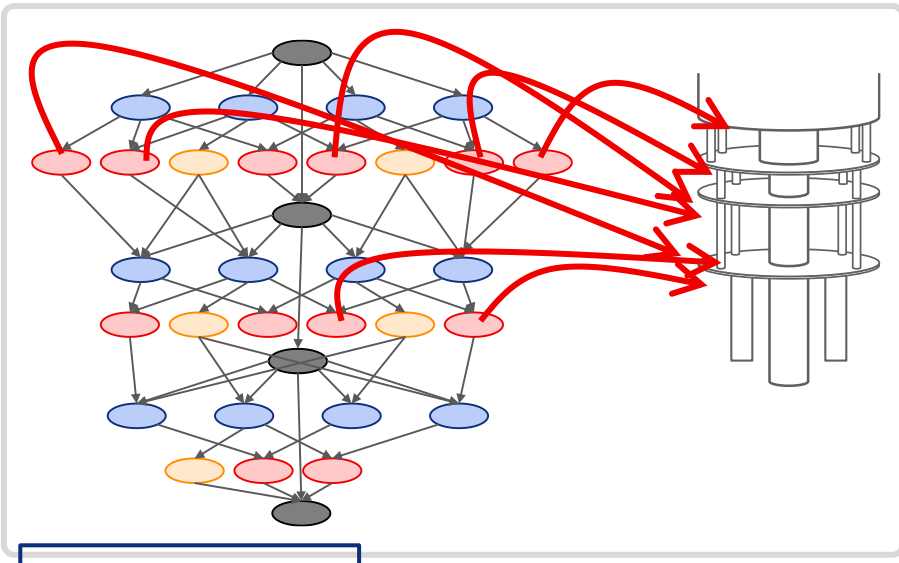


なぜ、第二階層タスクベースプログラミングが必要なのか

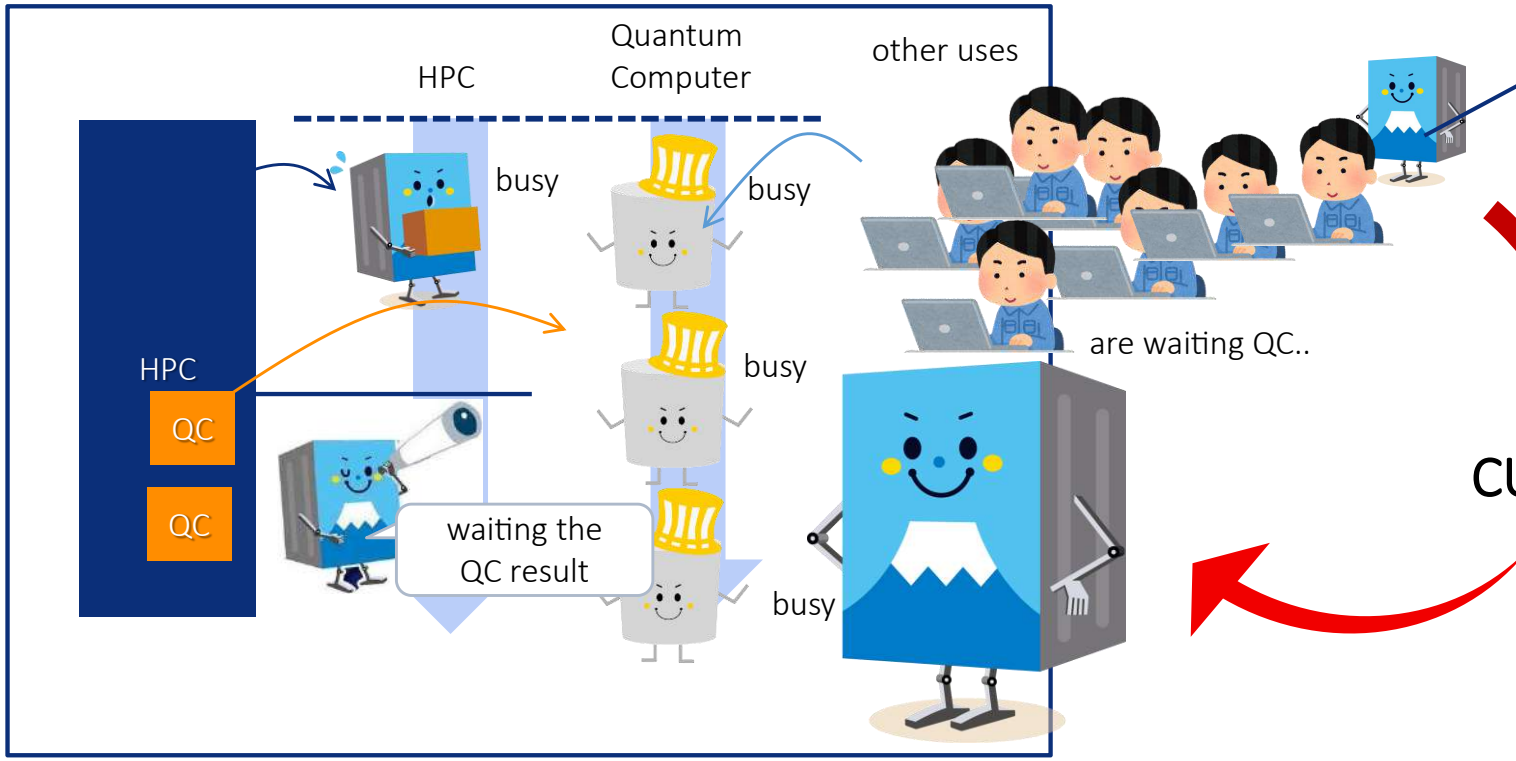
- 量子・HPCが密に連携し分離が難しい場合
- QAOAのように量子回路の繰り返しを含み、各繰り返しごとに
  ジョブとして投入することが現実的でない場合

<u>それでも、HPCのアイドル時間は最小化しなければならない</u>

- 特に大規模なHPCプログラムの場合



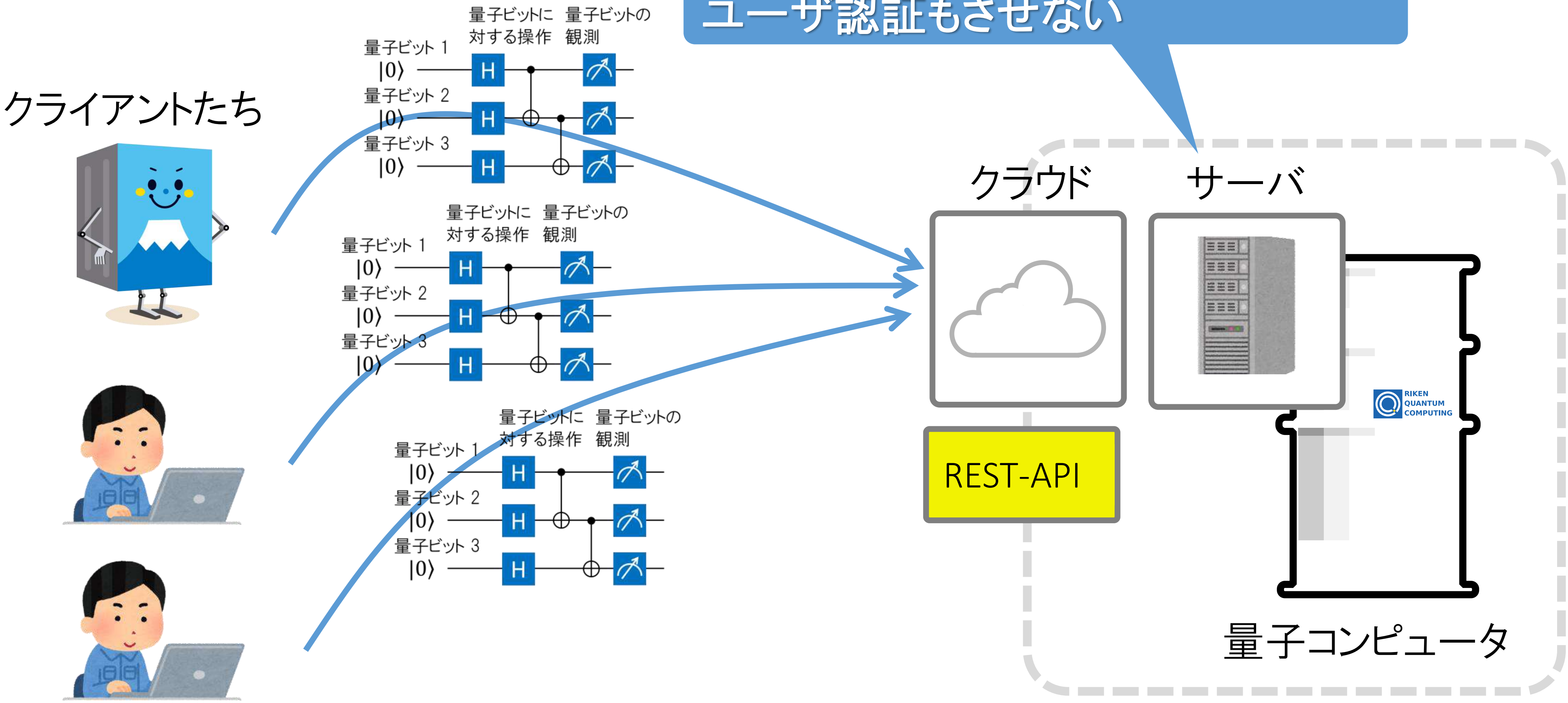
20～30MW/h



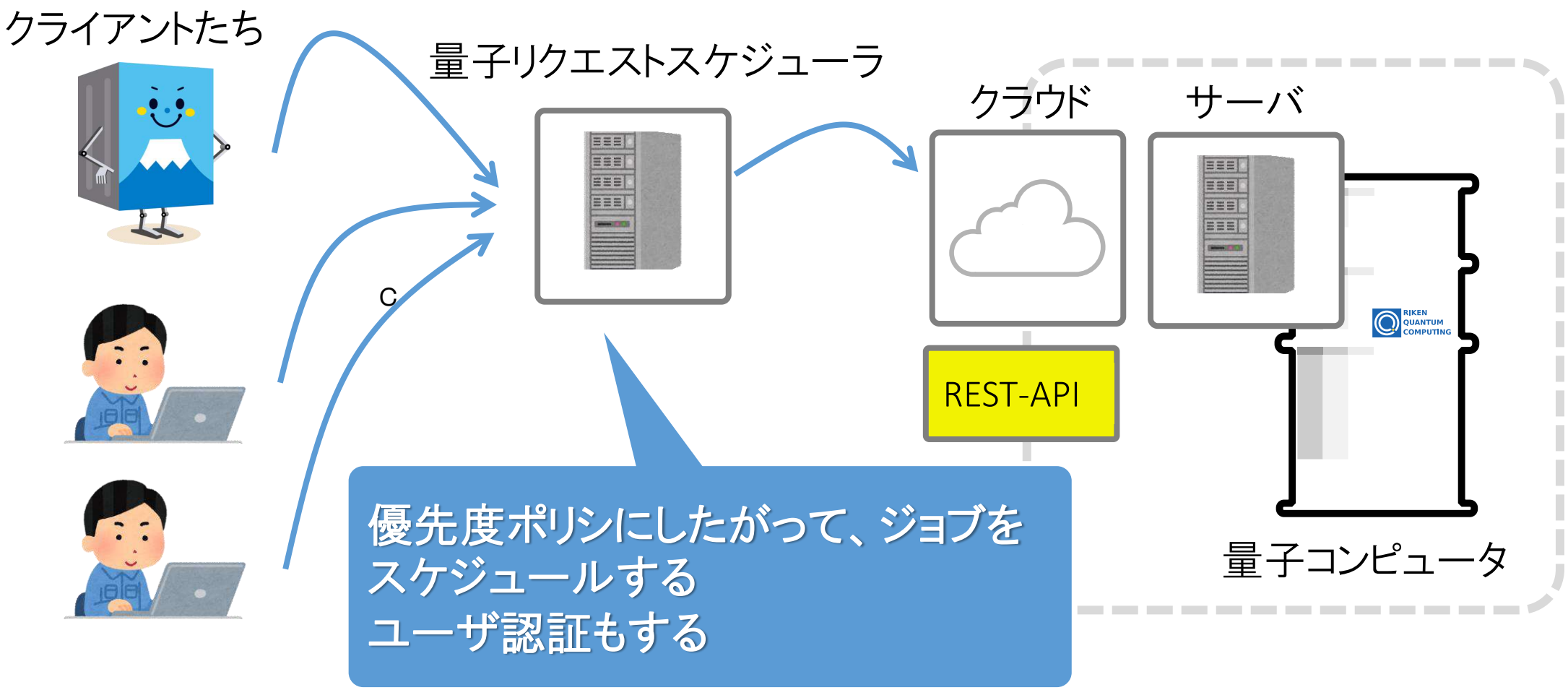
HPCから来た量子回路の実行依頼は、最優先で処理

量子リクエストスケジューラ

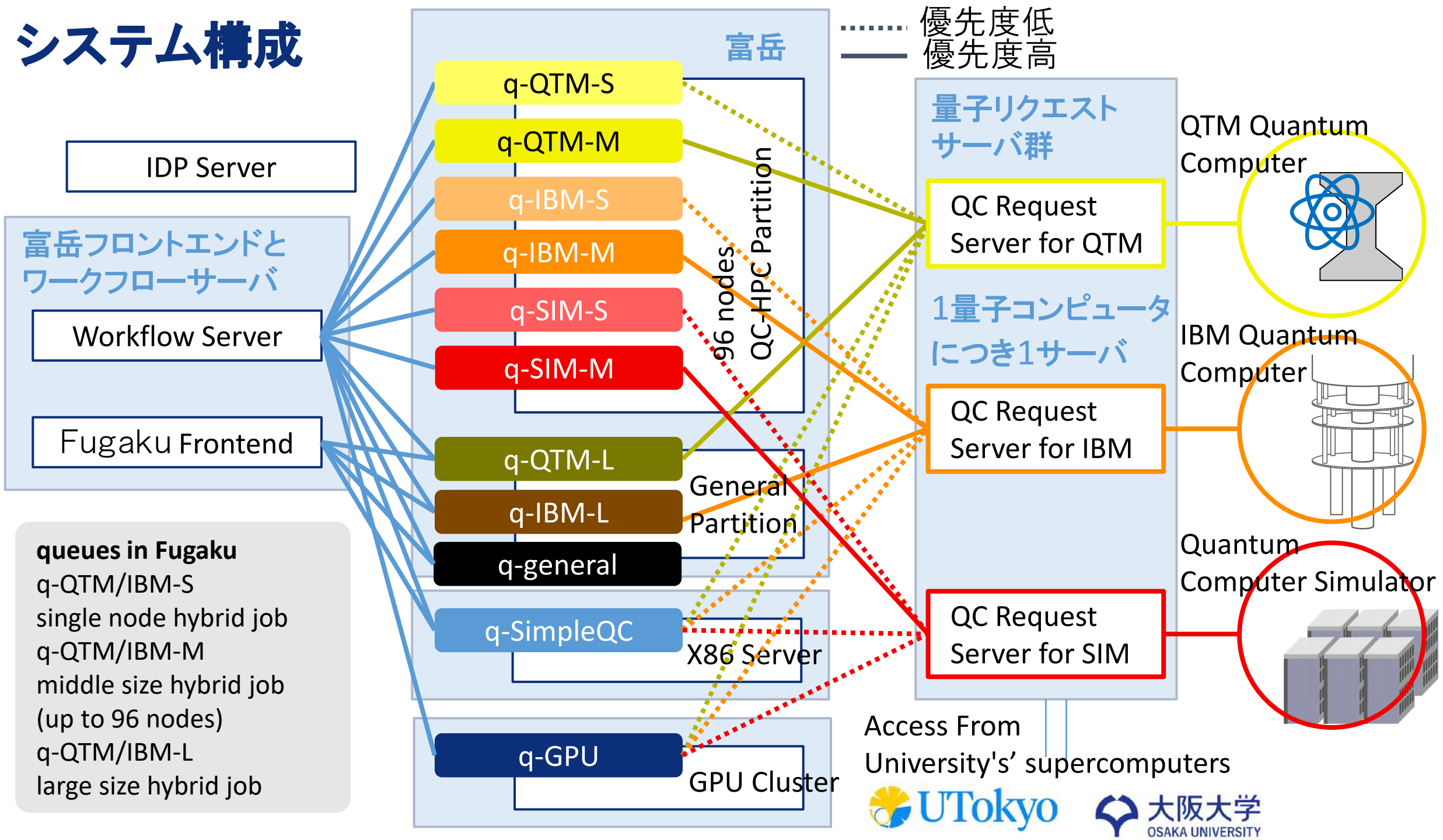
通常はここでスケジュールされるが、
ここにはスケジュールさせない
ユーザ認証もさせない



量子リクエストスケジューラ

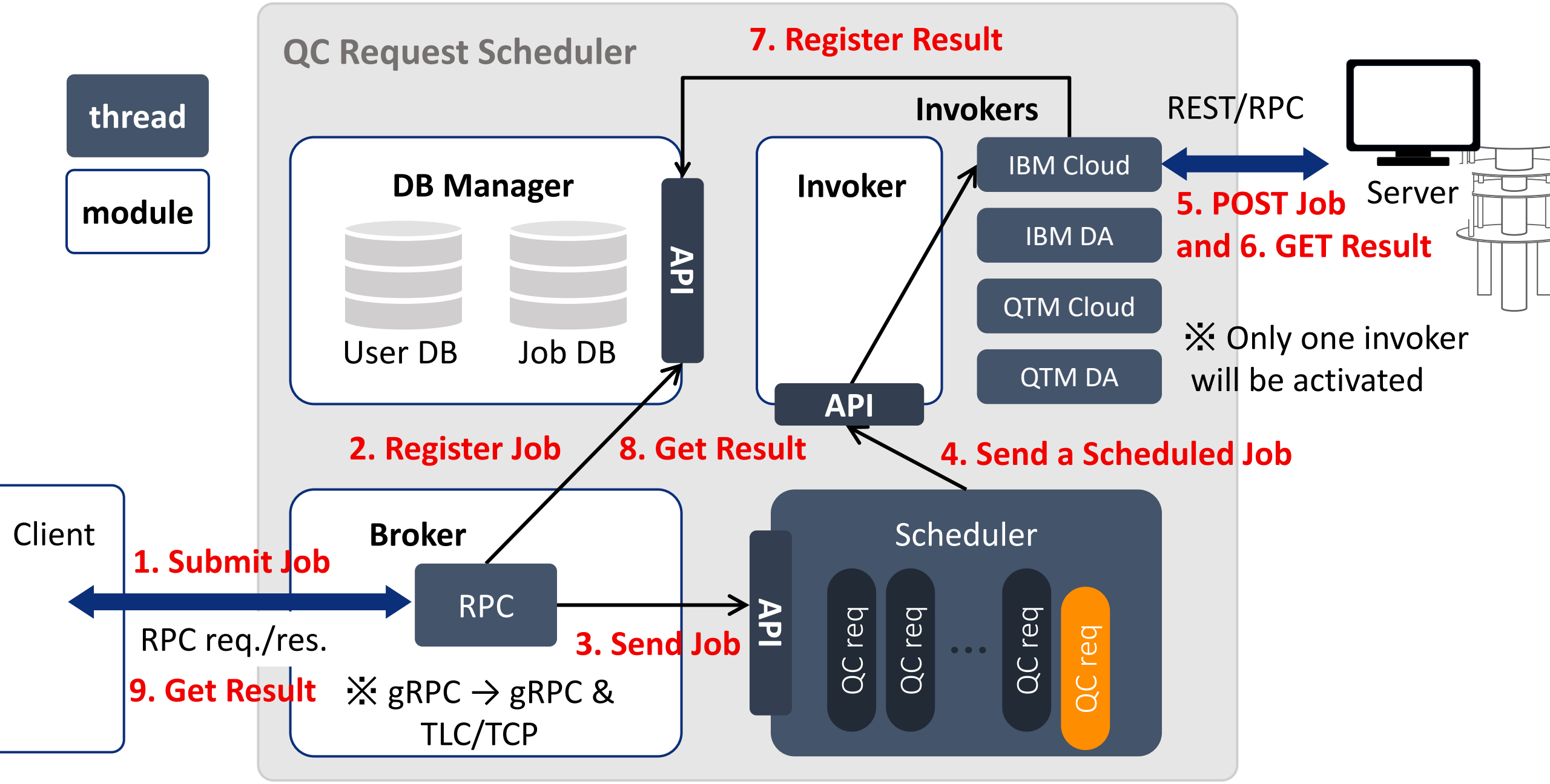


システム構成



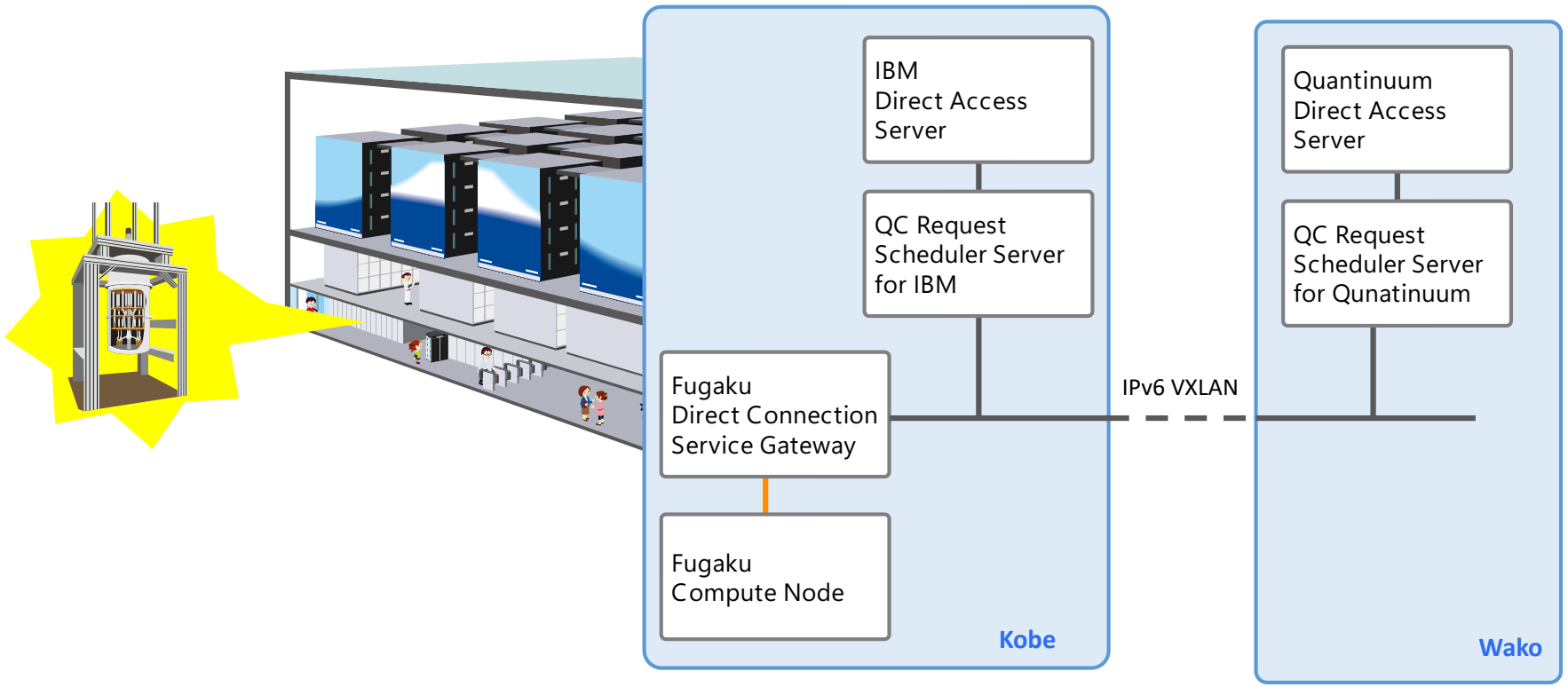
queues in Fugaku
q-QTM/IBM-S
single node hybrid job
q-QTM/IBM-M
middle size hybrid job
(up to 96 nodes)
q-QTM/IBM-L
large size hybrid job

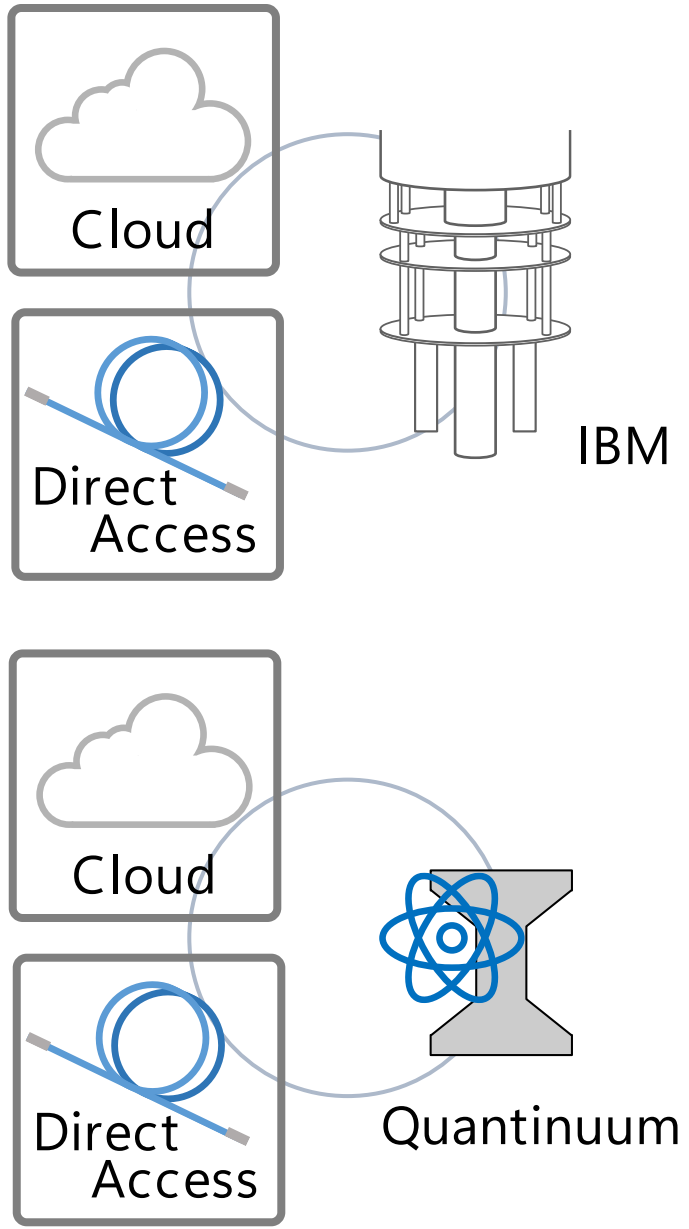
量子コンピュータ・リクエストスケジューラ



量子リクエストスケジューラ⇔量子バックエンド

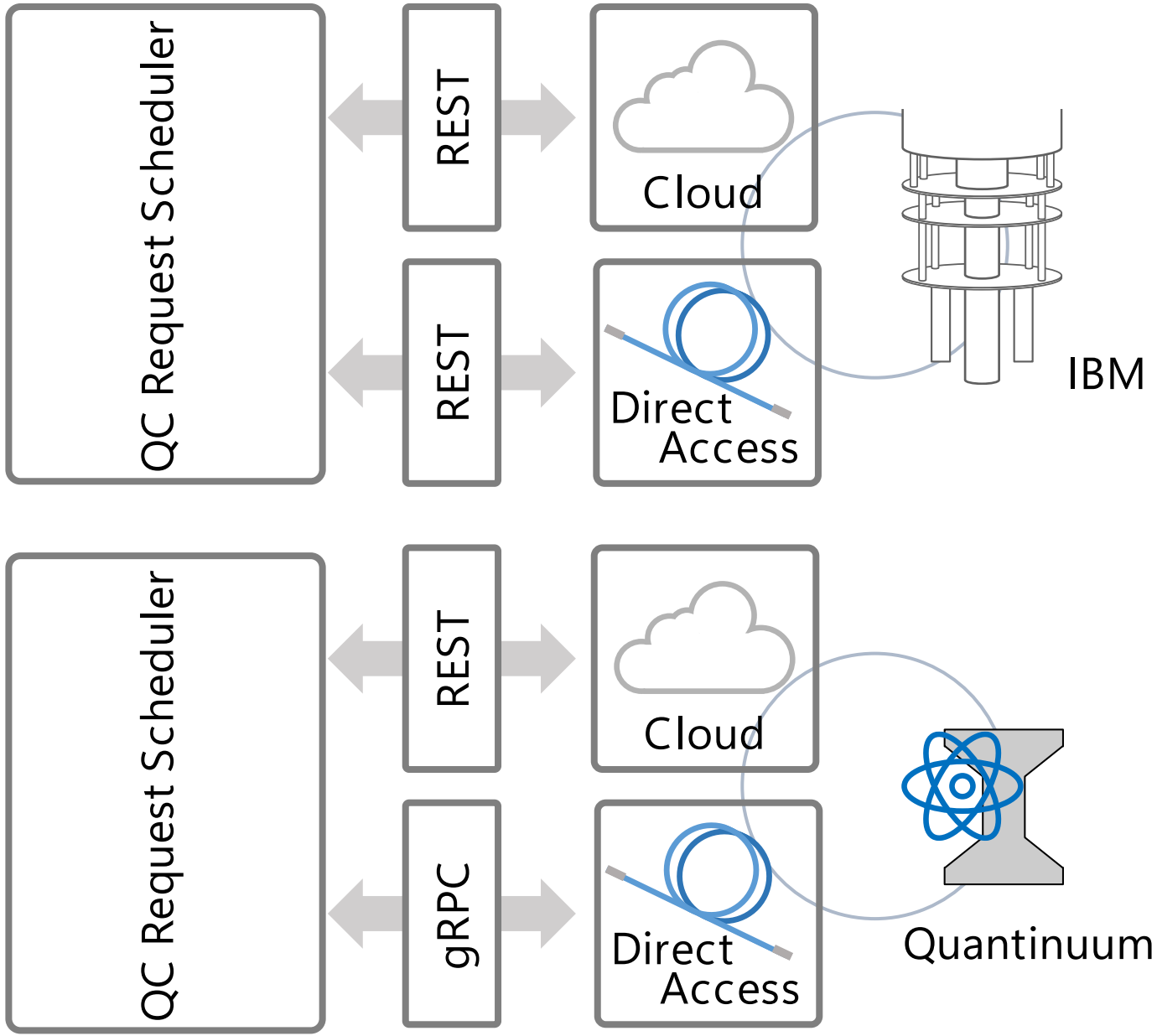
- 量子コンピュータのバックエンドサーバ
  - クラウド：既存のクラウドベースサービス。サーバは世界のどこかにある
  - ダイレクトアクセス：より高速なアクセスを提供。IBM向けサーバは神戸に、Quantinuum向けは神戸とバーチャルvLABNで接続された和光メインキャンパスに設置（予定）





量子リクエストスケジューラ⇔量子バックエンド

- 量子バックエンドサーバは、ベンダが定義し提供する、それぞれ異なるAPIで量子回路実行を受け付ける。
- 量子リクエストスケジューラには、これに合わせて個別のinvokerが実装される。
  ※ RESTの使用も異なる
- REST向けのinvokerは、Microsoft C++ REST SDKでそれぞれの使用に合わせて実装、gRPCはgRPCで実装
- 以下の機能がサポート(予定)
  - **submit** a job
  - check a job **status**
  - **cancel** a job
  - **delete** a job
- 量子リクエストスケジューラから見て、量子回路実行はブロッキング処理である:
  - 1度に1つの回路のみ送る
  - バックエンドサーバ側には、スケジュールの余地はない
- 量子コンピュータから見て「ユーザ」は、スケジューラであり、個別ユーザの認証は行わない



量子リクエストスケジューラ⇔量子バックエンド

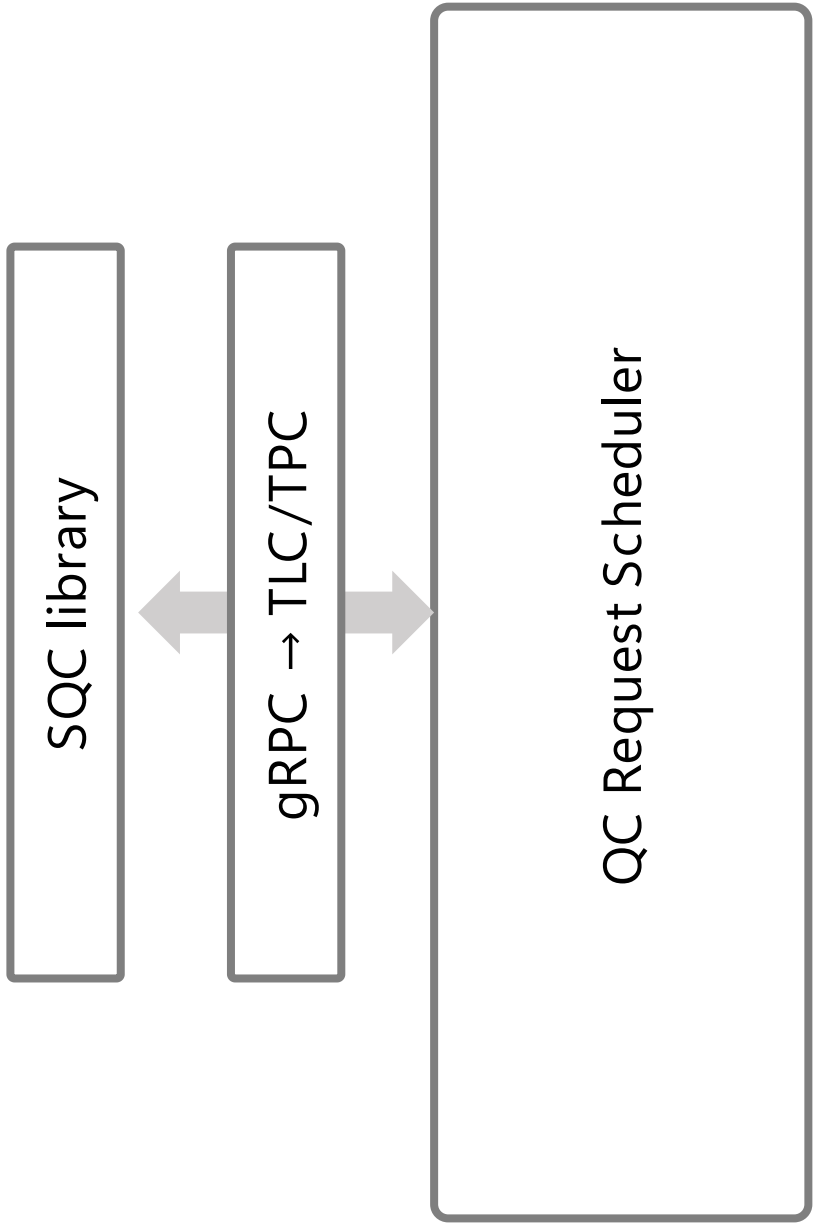
- APIは、量子コンピュータプロバイダの定義に従う

	RQC	IBM	Quantinuum
Cloud Service	REST	REST	REST
Direct Access	-	REST	gRPC

- これらの差は、ユーザからは完全に隠蔽される
  - ユーザは利用したい量子コンピュータを選択し、引数として与える
  - 適切なinvokerがランタイムに選択される
  - 回路は、OpenQASM形式で、量子バックエンドサーバに送られる
    - これらの3つの量子コンピュータで共通してサポートされている中間形式はOpenQASMのみ
  - 必要に応じてトランスパイルされる
    - 現状はIBMのみ投入前にトランスパイルが必須
    - ユーザが陽に行うことも可能
    - トランスパイルが必要なのにされていない場合は、ライブラリが勝手にやってくれる

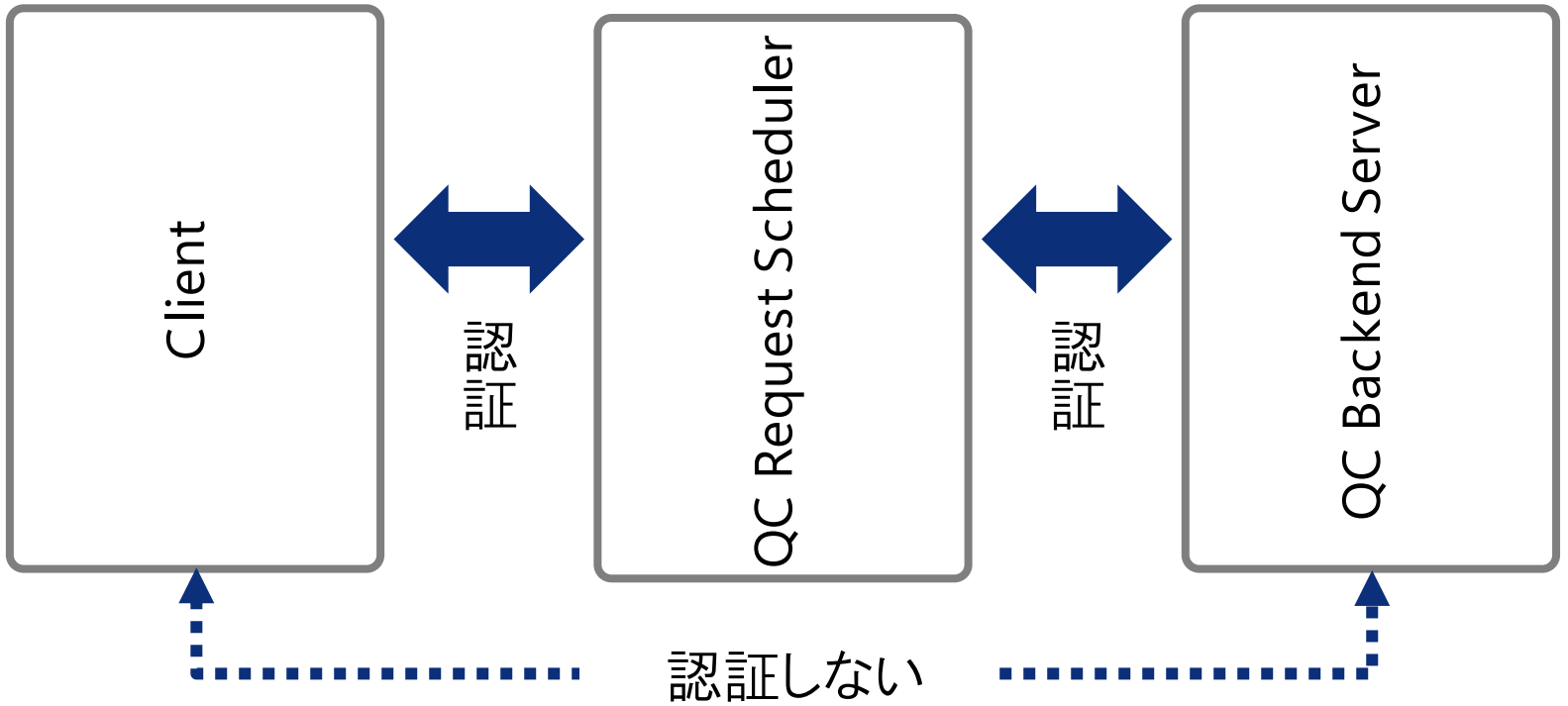
クライアント ⇔ 量子リクエストスケジューラ

- ユーザの認証は量子リクエストスケジューラで行う
- SQC ライブラリのAPIにより回路の構築や、回路の実行のオフロードが可能
- SQC ライブラリでは以下がサポートされる（予定）
  - circuit construction
  - **submit** a job (sync and async)
  - check a job **status**
  - **cancel** a job
  - **delete** a job
- C/C++ インターフェス
- Fortran ラッパ（by 東大）も提供予定
- Pythonは、Qiskit等の既存のツールを量子リクエストスケジューラが利用できるように拡張することでサポート
- 非同期オフロードを活用するプログラミングを強く推奨
  - 例：OpenMP



認証

- クライアント⇔量子リクエストスケジューラ
  - JSON Web Token (JWT) 認証＋TLS クライアント認証
    - TLS サーバ認証による通信秘匿性と健全性を保証した通信路上でセッション開始時にJWT認証を行う
- 量子リクエストスケジューラ⇔量子バックエンドサーバ
  - 量子コンピュータプロバイタの指定した方法による
    - 基本はJWT



Programming with SQC, Supercomputer Quantum computer Computing

```
const int nqubits = 4;
sqcBackend backend = qtmc_h1_1;
sqcRunOptions ropt = {0, 0};
sqcInitialize(NULL);
```

```
// Construct circuit
sqcQC* circ;
circ = sqcQuantumCircuit(nqubits);

sqcHGate(circ, 0);
sqcCXGate(circ, 0, 1);
for(int i=0; i<nqubits; i++)
   sqcMeasure(circ, i, i, NULL);
// You can provide a text file name or
string of a qasm circuit
```

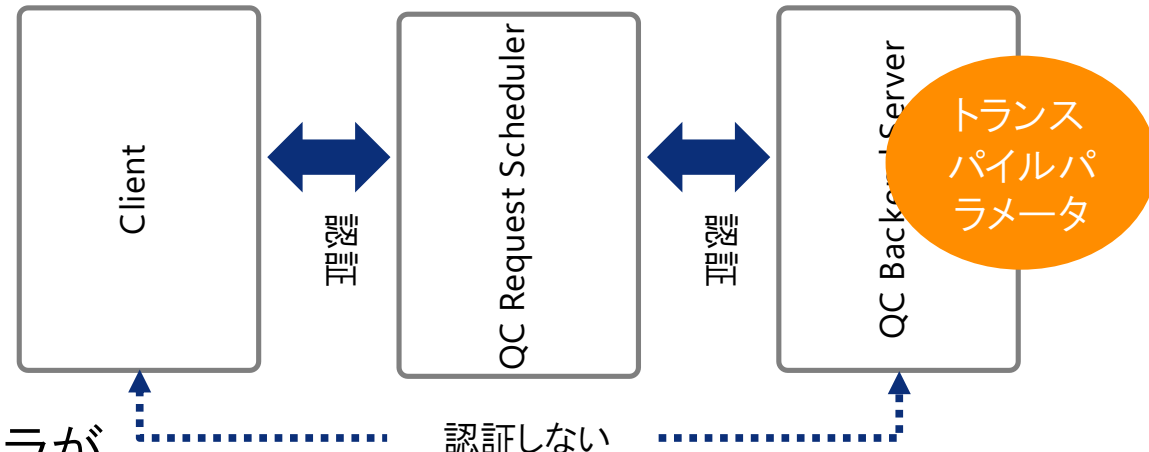
```
circ->qasm = (char *)calloc
            (sizeof(char), MAX_QASM_LEN);
int len = sqcConvQASMtoMemory
        (circ, circ->qasm, MAX_QASM_LEN);
sqcTranspile(circ, backend, &ropt);
```

```
// Run the circuit and wait the result
char *result = NULL;
char *job_id = NULL;
ropt.nshots = 1024;
int ret1 = sqcQCRunAsync
        (circ, backend, ropt, &job_id);
// do something ...
int ret2 = sqcQCWait
            (backend, job_id, &result);
```

```
if(result!=NULL){
  printf("Result: %s¥n",result);
  free(result);
}
if(job_id!=NULL){
  free(job_id);
}
sqcDestroyQuantumCircuit(circ);
sqcFinalize();
```

Optionally, you can transpile your code explicitly in your code for a certain system

プログラミング環境



- トランスパイラ
  - 現状はIBMのみローカルでトランスパイル必要あり / 可能
  - Qiskit TranspilerのPython C APIを用意
    - トランスパイルに必要な情報は量子スケジューラが量子バックエンドに問い合わせ、ユーザに提供する
  - ユーザがトランスパイルしていない場合はライブラリがトランスパイルする
- Batch スケジューラ･ワークフローエンジン
  - Xcrypt, Python base (Prefect?), Tierkreis by Quantinuum 提供予定
  - GUI (by Softbank)
  - 好きなものを持ち込んでもOK
- 量子SDK
  - Qiskit, TKET, CUDA-Qを提供
  - 量子リクエストスケジューラとの通信機能を追加（予定）

シミュレータ向けプログラミング環境

- RIKEN Braket
  - https://github.com/jhpc-quantum/RIKEN-braket
  - AWS Braket とは異なる
  - 大規模分散計算に特化した状態ベクトルシミュレータ
  - Bra : オリジナルの量子アセンブラおよびそのインタプリタ
    - 本プロジェクトの成果として OpenQASM に対応！
  - Ket : シミュレーションのためのC++ヘッダライブラリ
  - GPUへの対応を準備中
    - Lawrence Berkeley National Laboratory との共同研究

まとめと本年度の進捗

① 2023年度に設計した実行環境の実現に向けて
- 量子リクエストスケジューラの実装（ほぼ終了）
- 量子リクエストスケジューラへの認証機能の追加（進行中）

② プログラミング環境の実現に向けて
- ①で開発した量子リクエストスケジュラへのジョブの投入などをライブラリ化
- ①のスケジューラと連携するトランスパイル用のAPIとライブラリを開発
  - IBMのみ、量子回路投入前に陽なトランスパイルが必要
- 第一階層プログラミングのワークフロー
  - Xcrypt 基本的にそのまま富岳で使える。Simple QCサーバ向けのジョブ投入の拡張が進行中
  - Tierkreis Quantinuum社により移植
  - Prefect 調査中