

The background is a deep blue gradient. It features several horizontal lines of glowing green and yellow binary code (0s and 1s) that appear to be receding into the distance. Overlaid on this are several bright, white, curved light streaks that sweep across the frame, creating a sense of motion and energy. A bright, circular lens flare or light burst is visible in the center-right area.

SQL+GPU+SSD= ∞

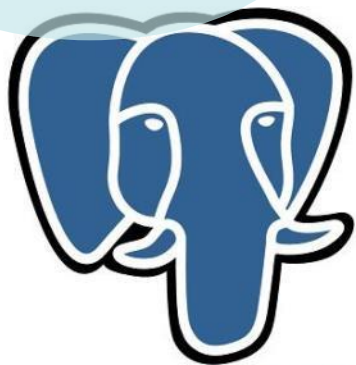
かぴばら@品川

自己紹介



- Name: かぴばら@品川
- PostgreSQL歴: 9年ほど(2006~)
- works: セキュリティ周り、FDW周り、諸々...
- 趣味: 別業界のテクノロジーを持ってきて
PostgreSQLに組み合わせる事
(混ぜるな危険?)

Very functional
& well-used
database



PostgreSQL

PG-Strom:

なるものを作っています。

GPGPU



Very powerful
computing
capability

What's PG-Strom – ざっくり説明すると

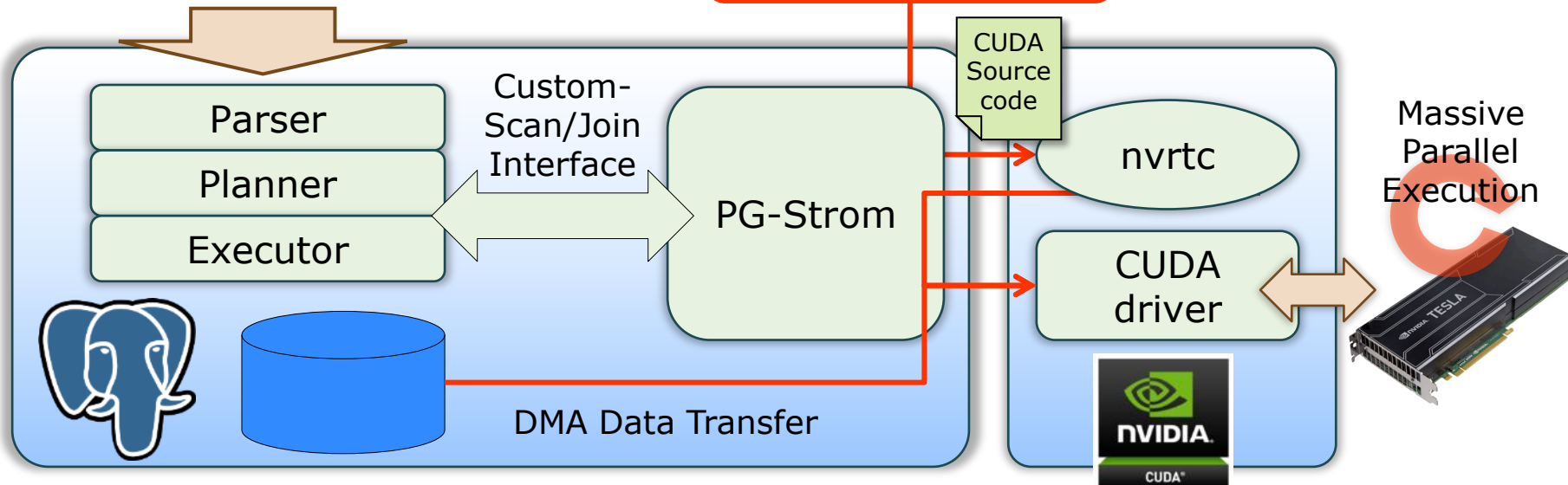
コア機能

- ① SQLからGPUネイティブバイナリを動的に生成する。
- ② GPUによる“超”並列処理を非同期に実行する。

利点

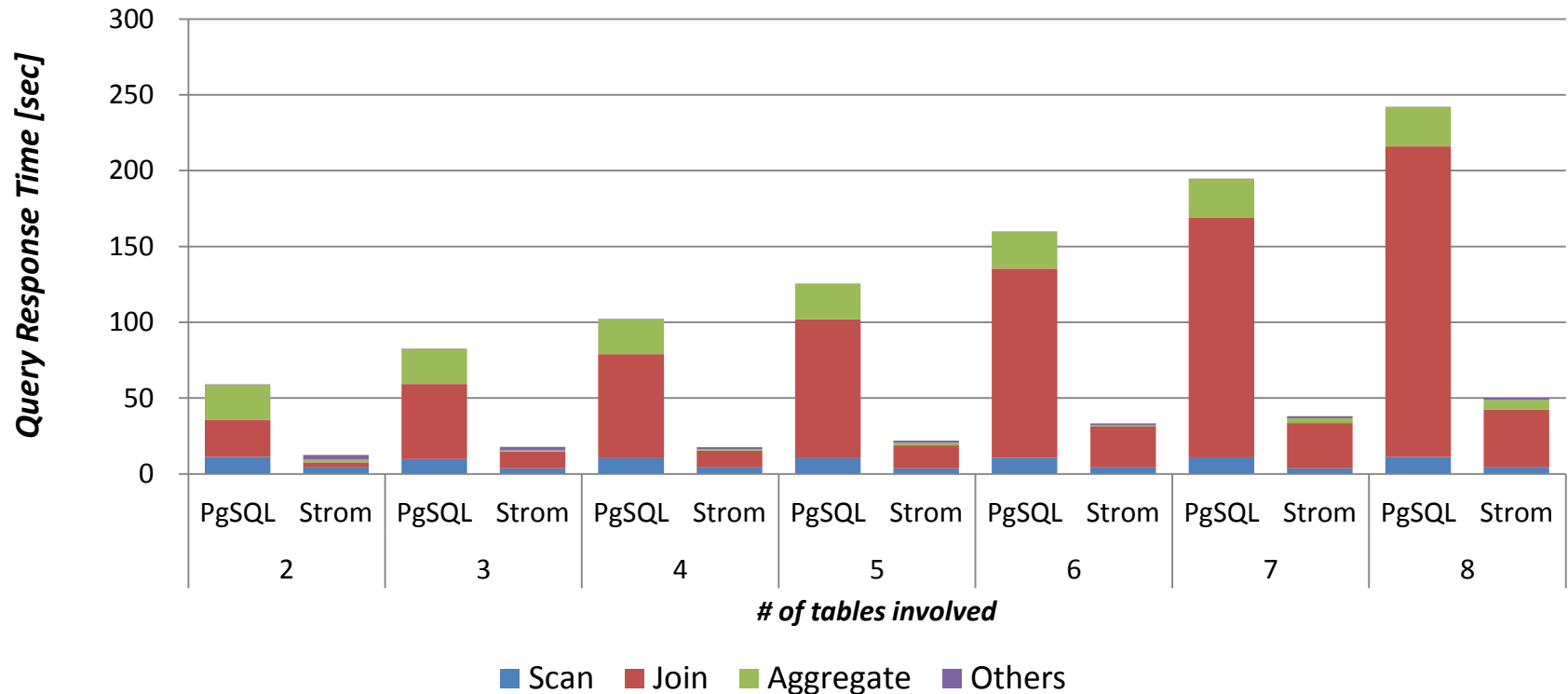
- SQLクエリを透過的にアクセラレーション可能。
- 一般的なH/Wを使って構成可能。安価。

```
Query: SELECT * FROM l_tbl JOIN r_tbl on l_tbl.lid = r_tbl.rid;
```



対応ワークロード – Scan, Join, Aggregation

Time consumption per component (PostgreSQL v9.5b vs PG-Strom)



SELECT cat, AVG(x) FROM t0 NATURAL JOIN t1 [, ...] GROUP BY cat;

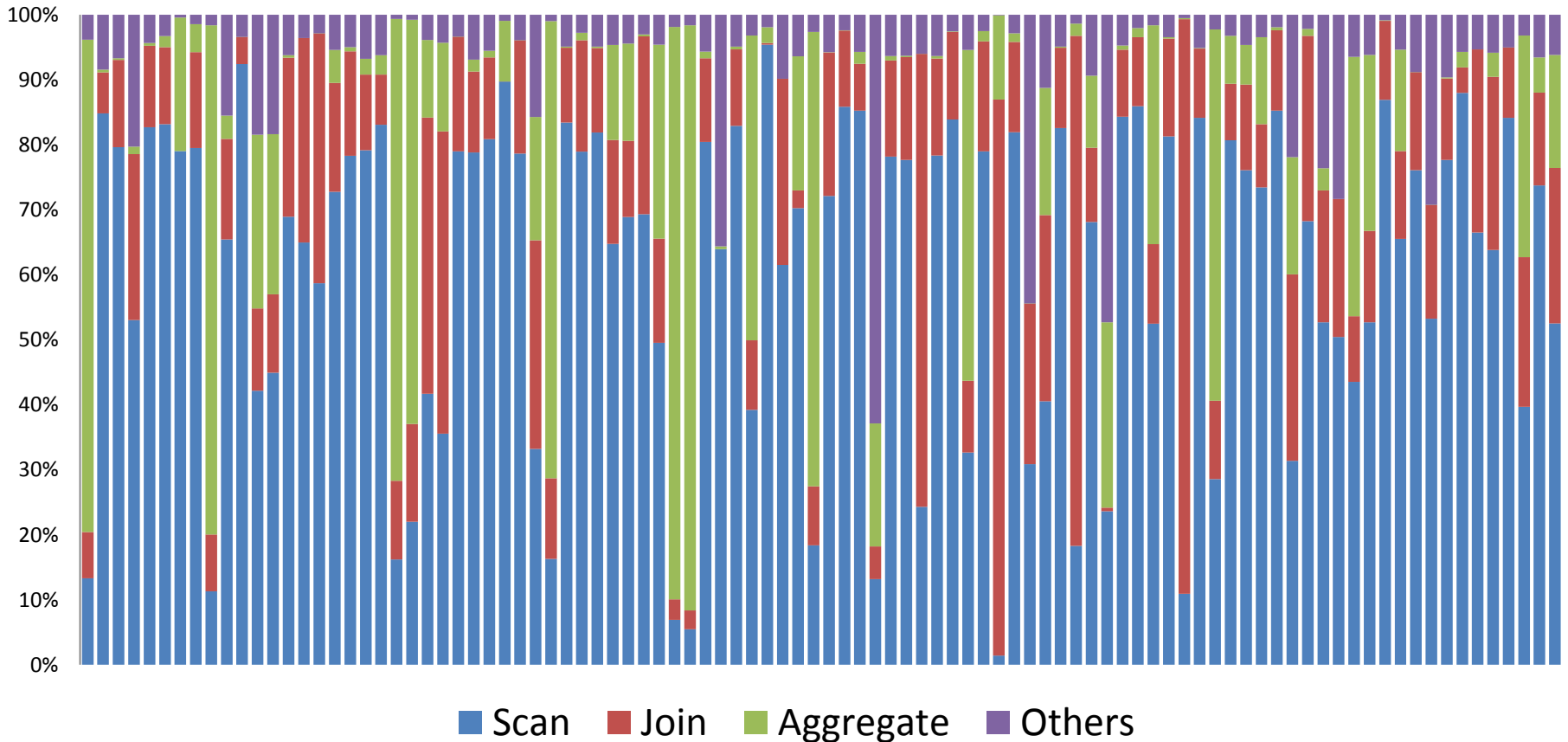
- t0: 100M rows, t1~t10: 100K rows for each, all the data was preloaded.

測定環境:

- PostgreSQL v9.5beta1 + PG-Strom (22-Oct), CUDA 7.0 + RHEL6.6 (x86_64)
- CPU: Xeon E5-2670v3, RAM: 384GB, GPU: NVIDIA TESLA K20c (2496cores)

次のターゲットはI/O – TPC/DSワークロードの解析より

Time consumption per workloads (PostgreSQL v9.5beta+PG-Strom)



では、どうやってGPUでI/Oを高速化しようというのか？

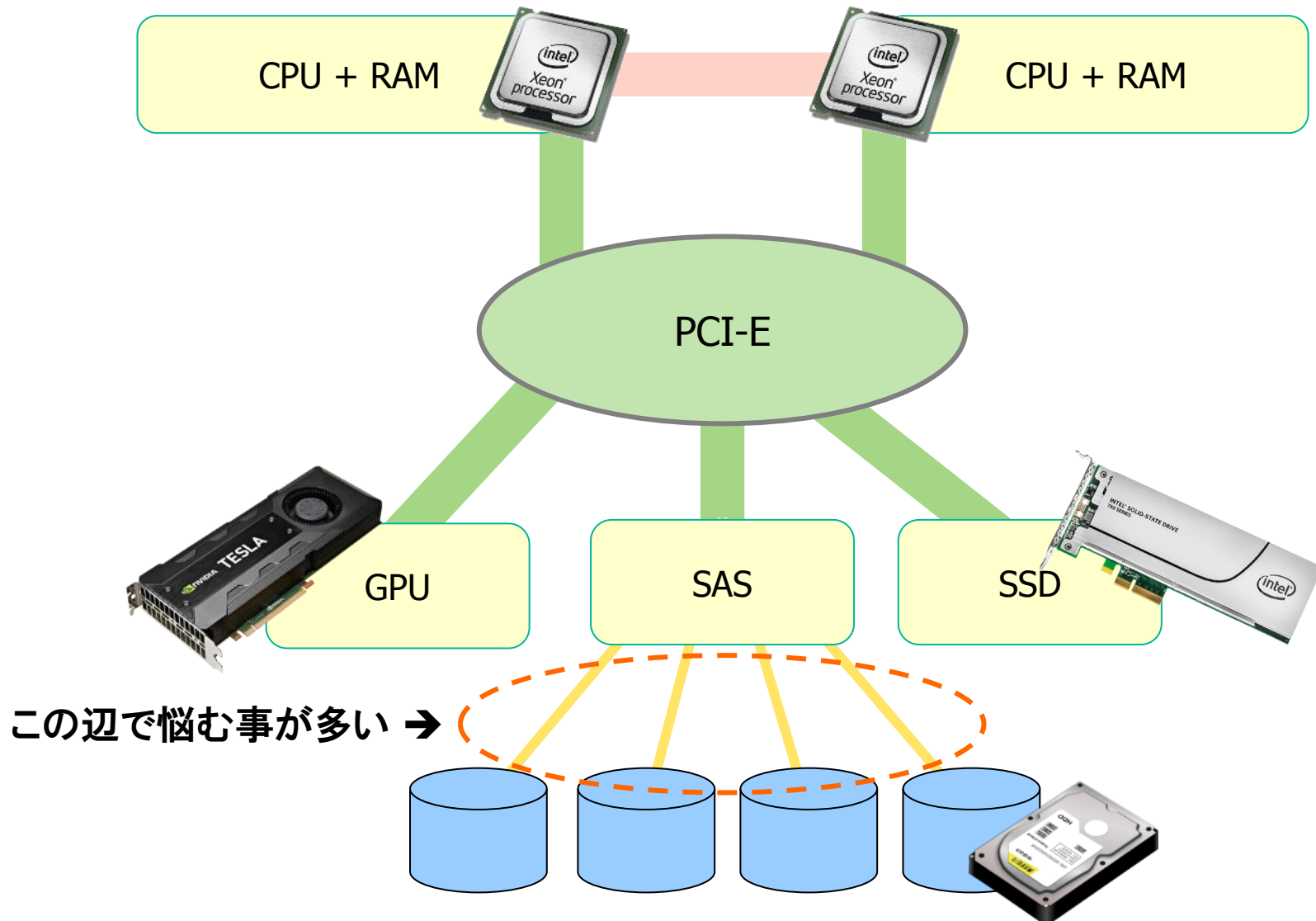
お断り。

今からご紹介する話は、

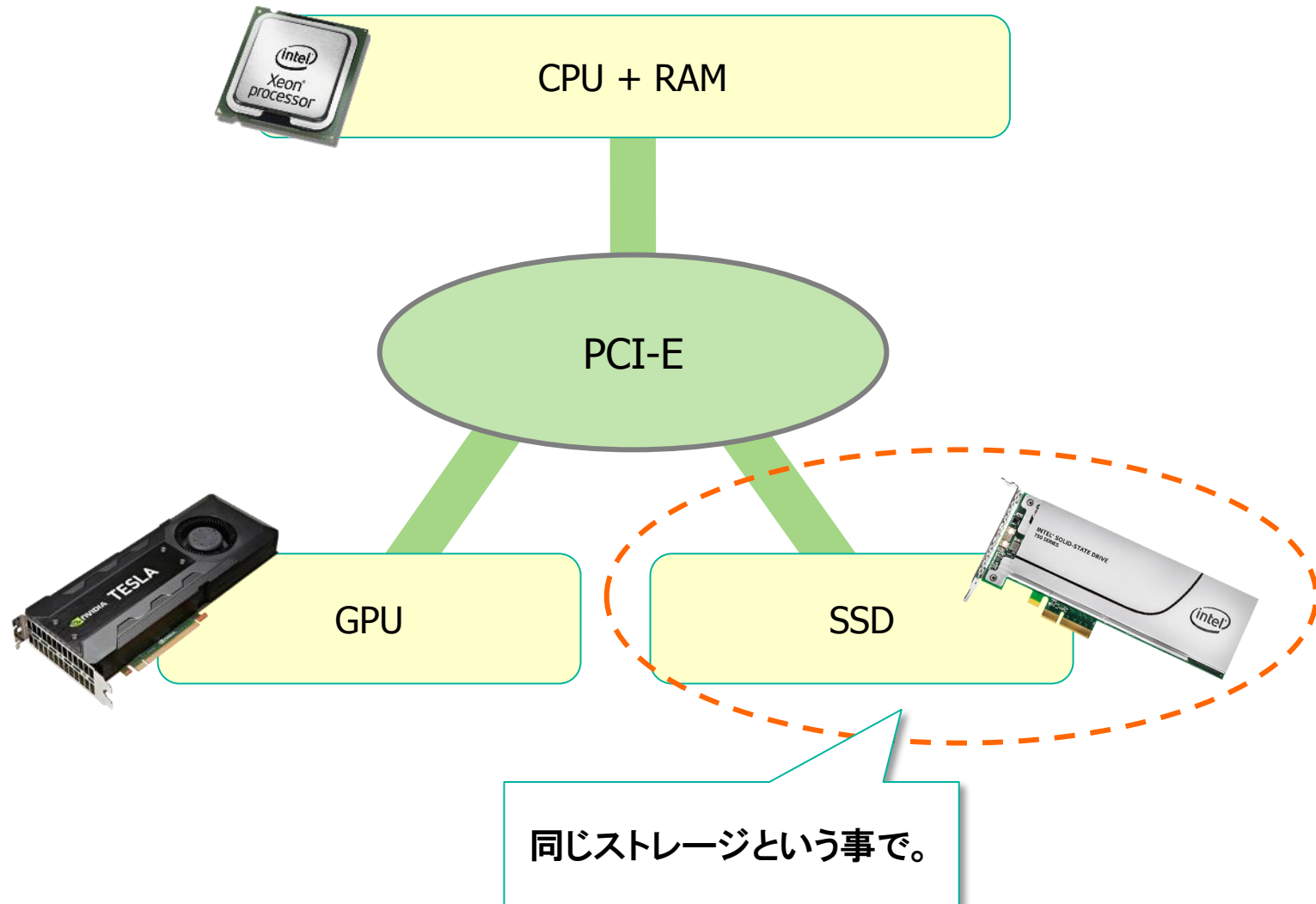
現在のところ**実装アイデア**です。

.....これから頑張って作りますよ。

大雑把な x86 ハードウェアの構成



説明のため図を単純化します



NVM EXPRESS SSD

PCI-Eダイレクト接続タイプのSSD – 低レイテンシと広帯域が特長



HGST
Ultrastar SN100



Intel
SSD DC P3700



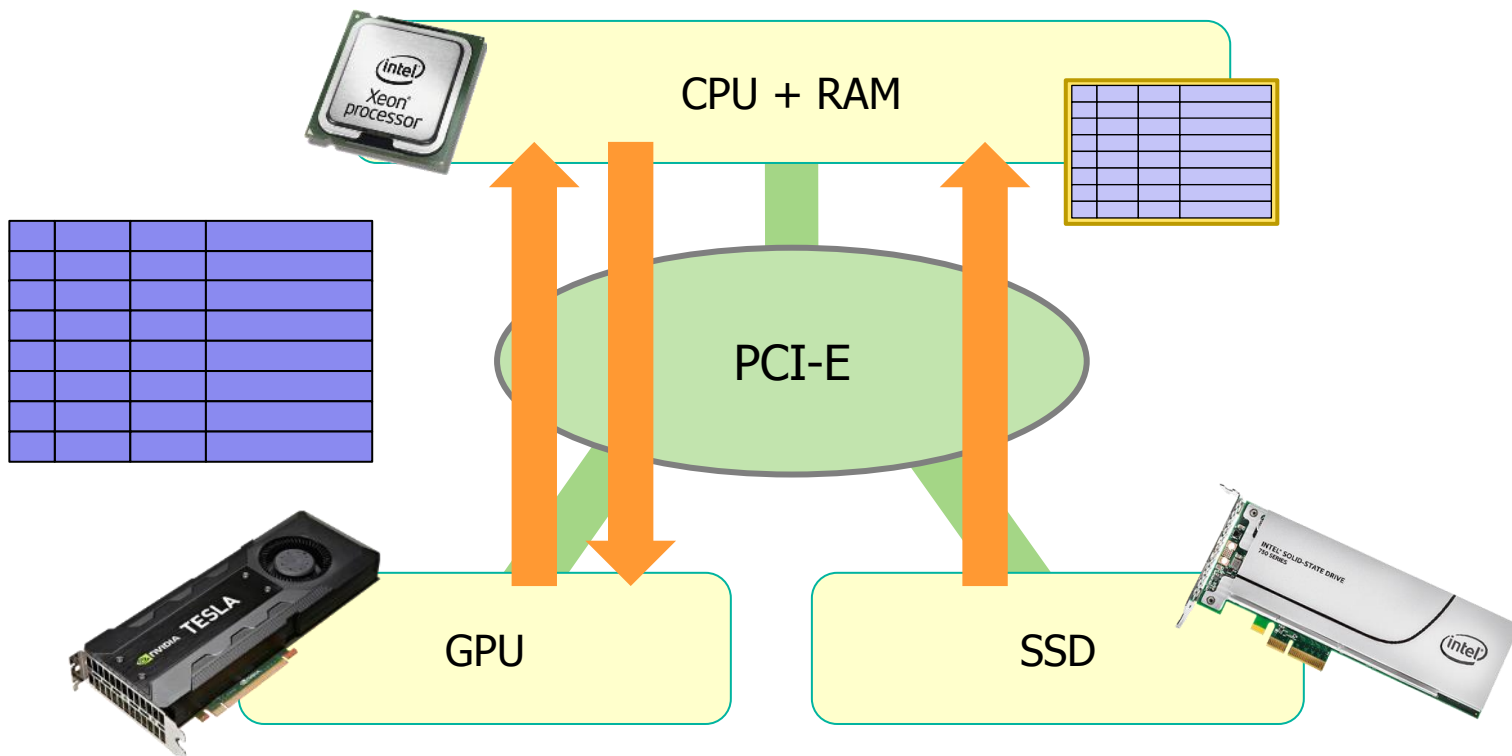
Intel SSD 750



Samsung
SSD 950 PRO



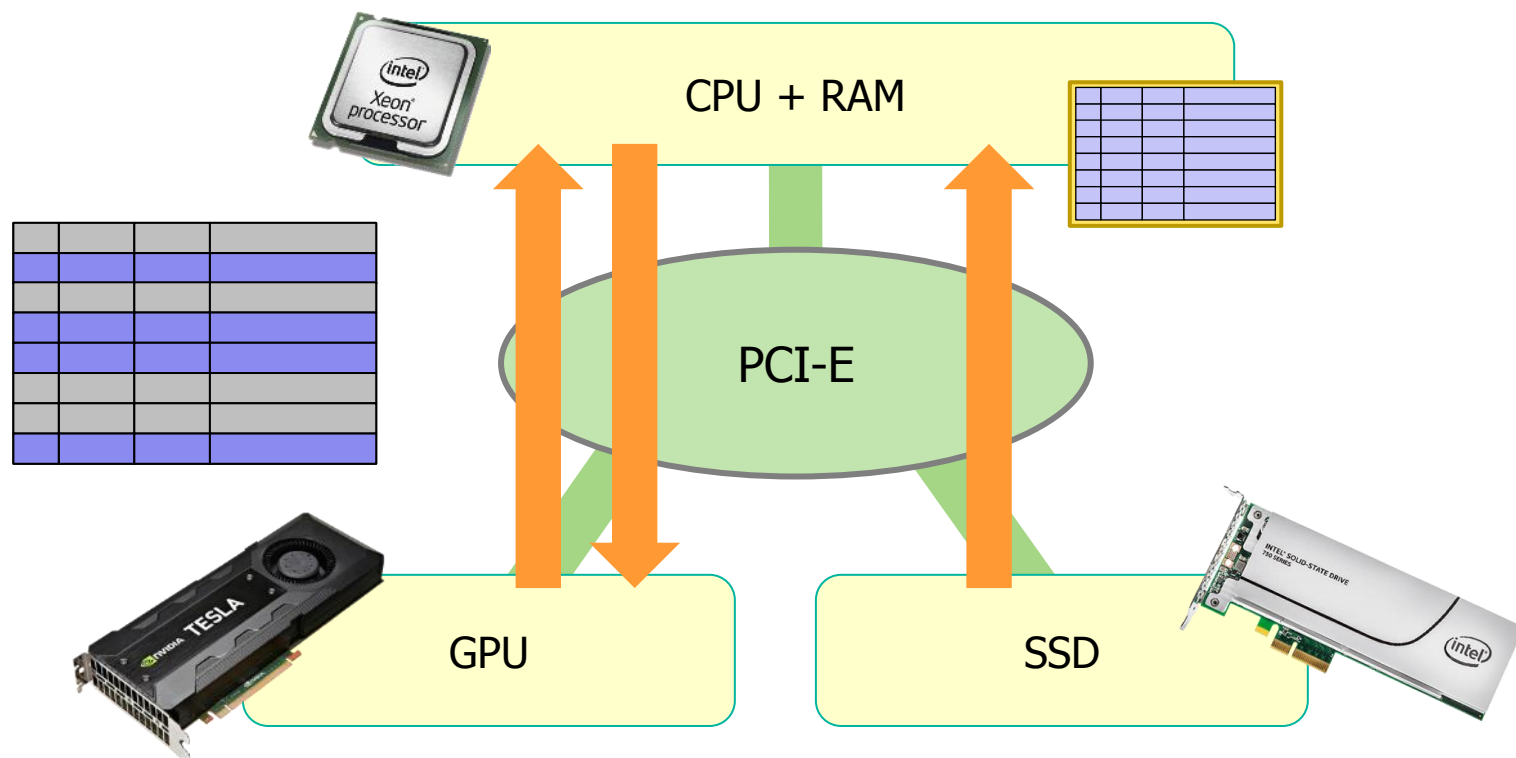
解析系クエリ実行時のデータフロー



① ストレージ → CPU/RAMへデータをロード

Table			

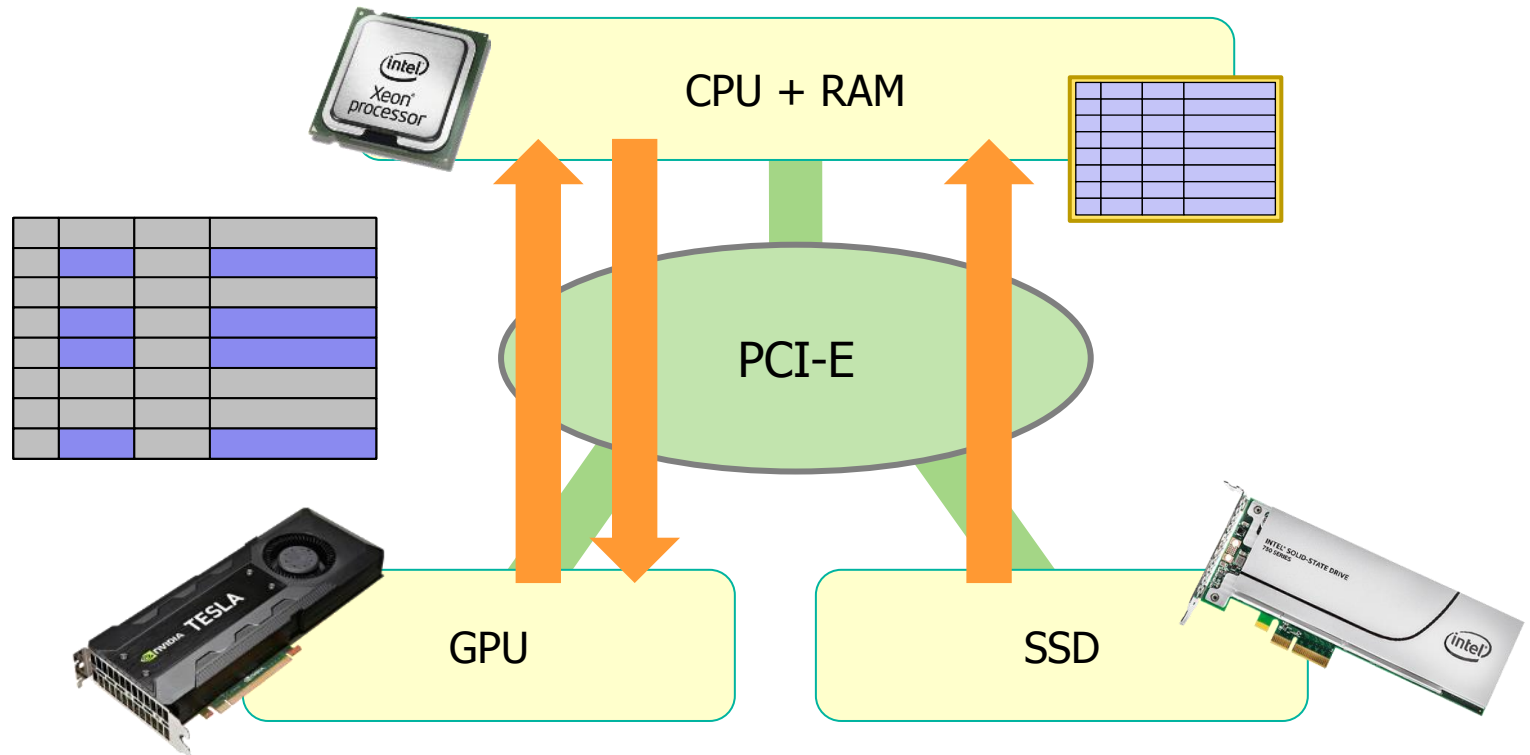
解析系クエリ実行時のデータフロー



- ① ストレージ → CPU/RAMへデータをロード
- ② 条件句で行をフィルタリング(Select)

Table			

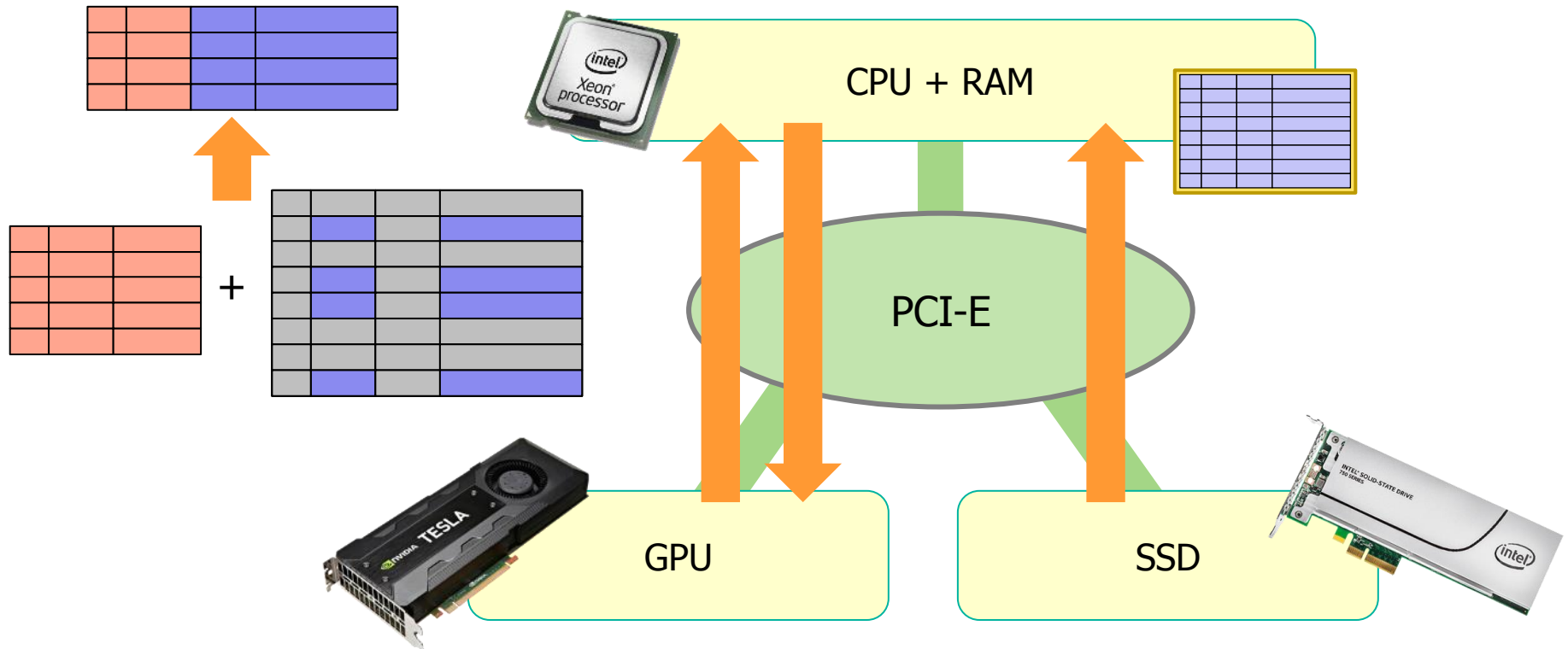
解析系クエリ実行時のデータフロー



- ① ストレージ → CPU/RAMへデータをロード
- ② 条件句で行をフィルタリング(Select)
- ③ 参照されない列を除去(Projection)

Table			

解析系クエリ実行時のデータフロー

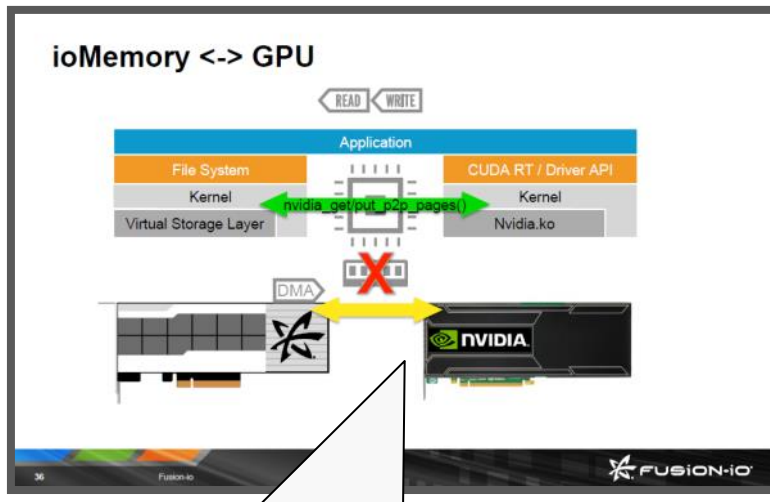
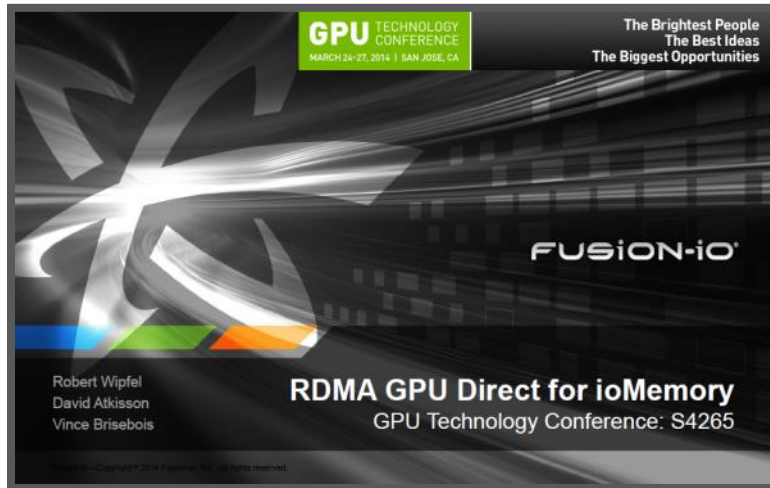


- ① ストレージ → CPU/RAMへデータをロード
- ② 条件句で行をフィルタリング(Select)
- ③ 参照されない列を除去(Projection)
- ④ 他のテーブルと結合(Join)

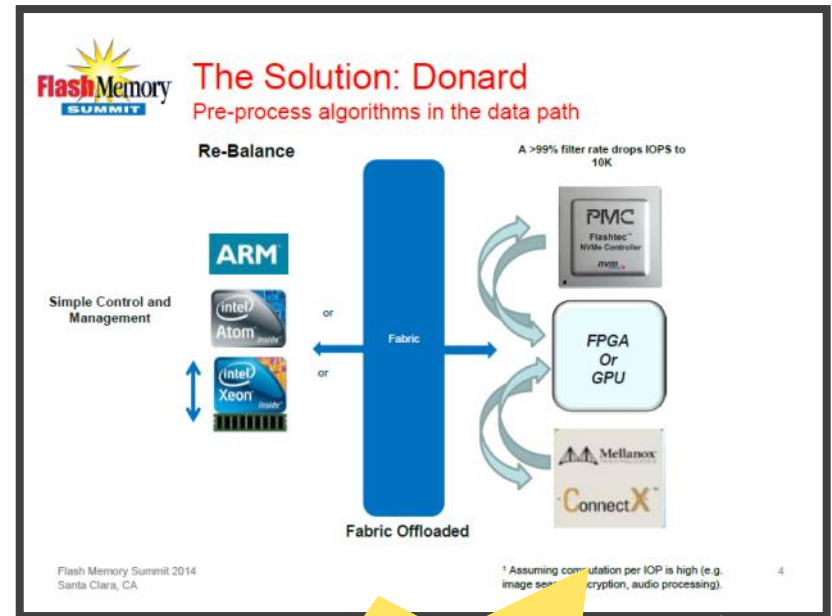
↑ CPUの仕事

Table			

SSD-to-GPU Direct

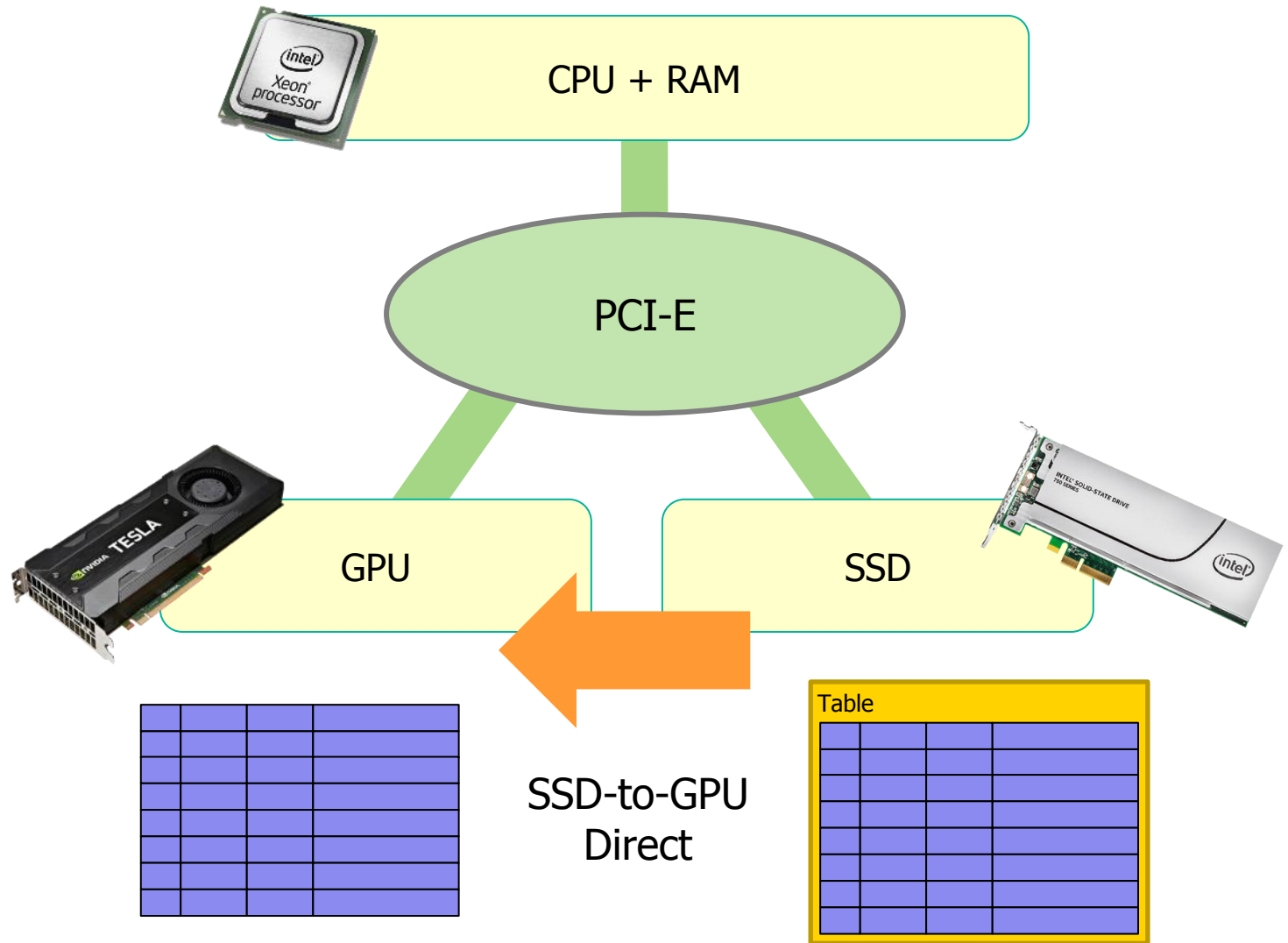


CPU/RAMを介さずに
SSD→GPU間でデータ転送

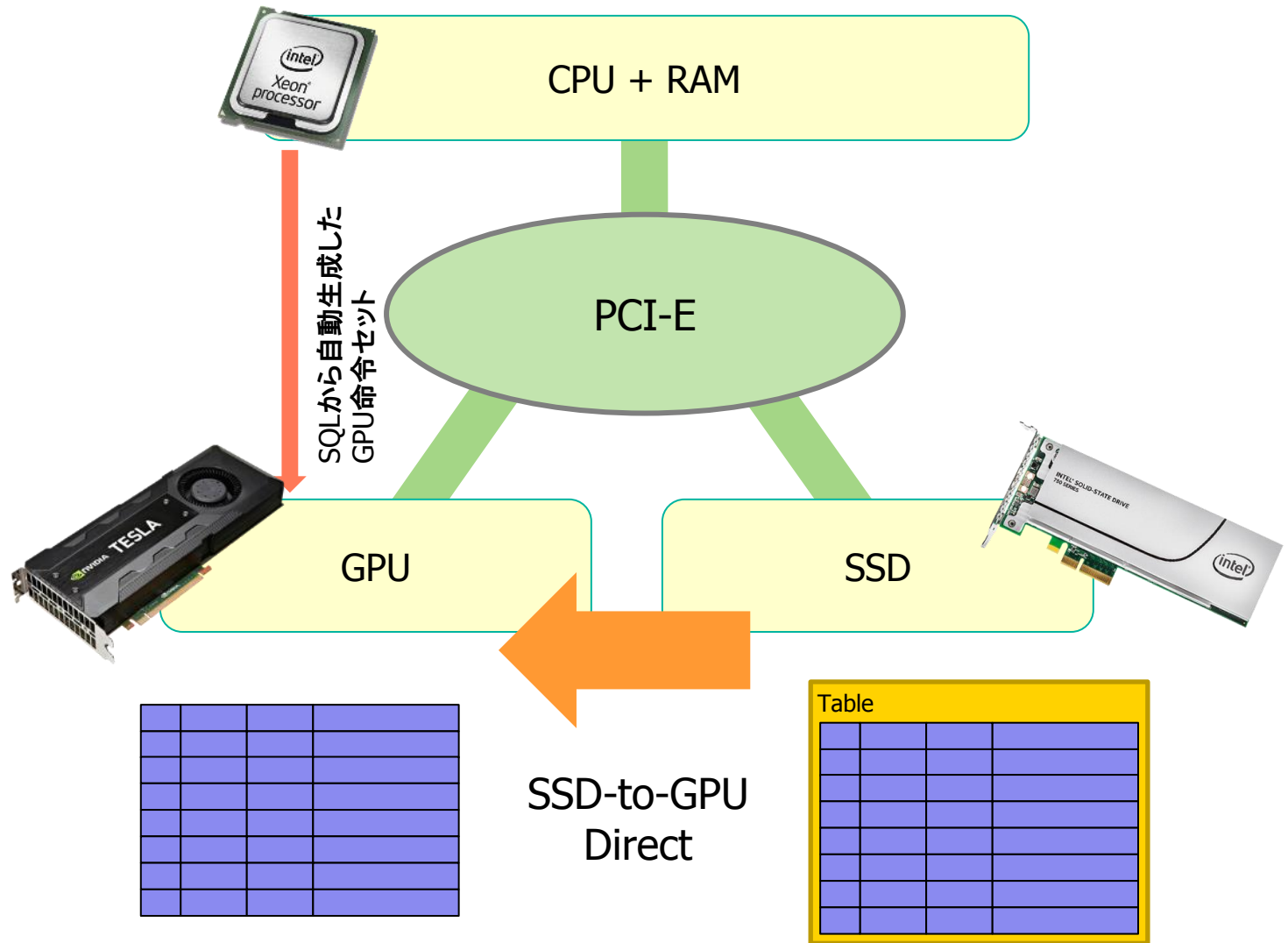


同じ事が標準規格の
NVMeでも可能！

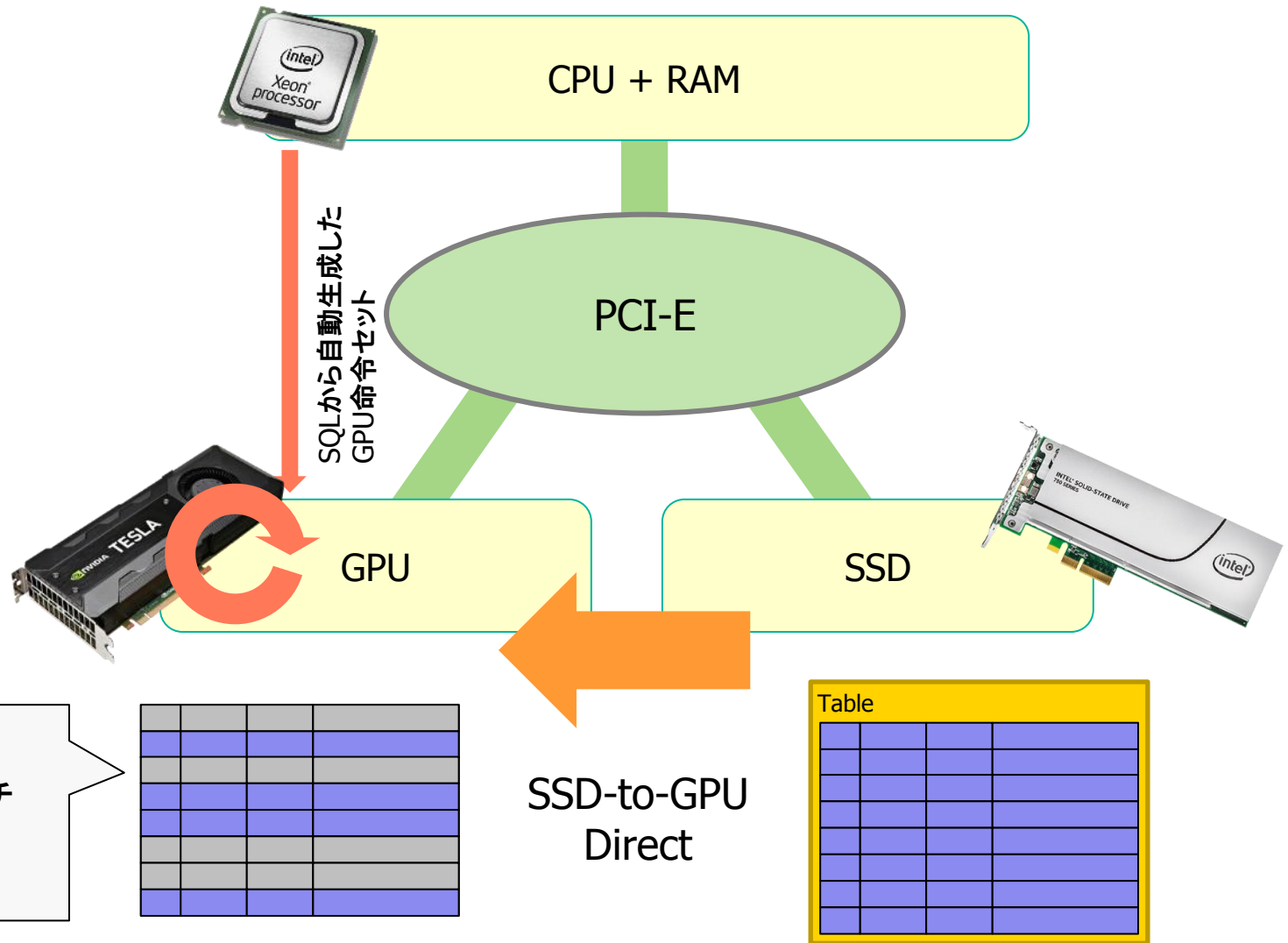
解析系クエリ実行時のデータフロー (1/3) – 基本パターン



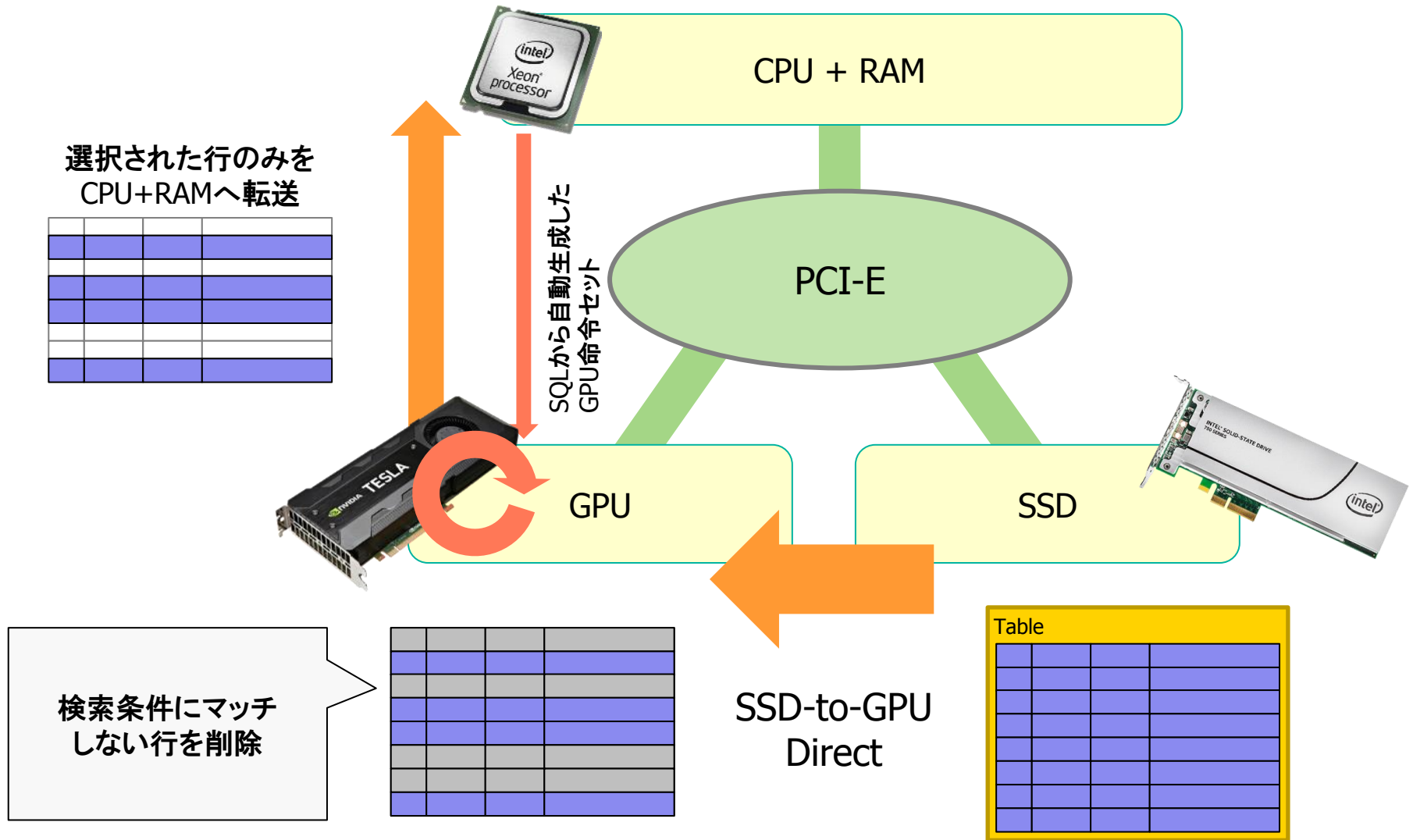
解析系クエリ実行時のデータフロー (1/3) – 基本パターン



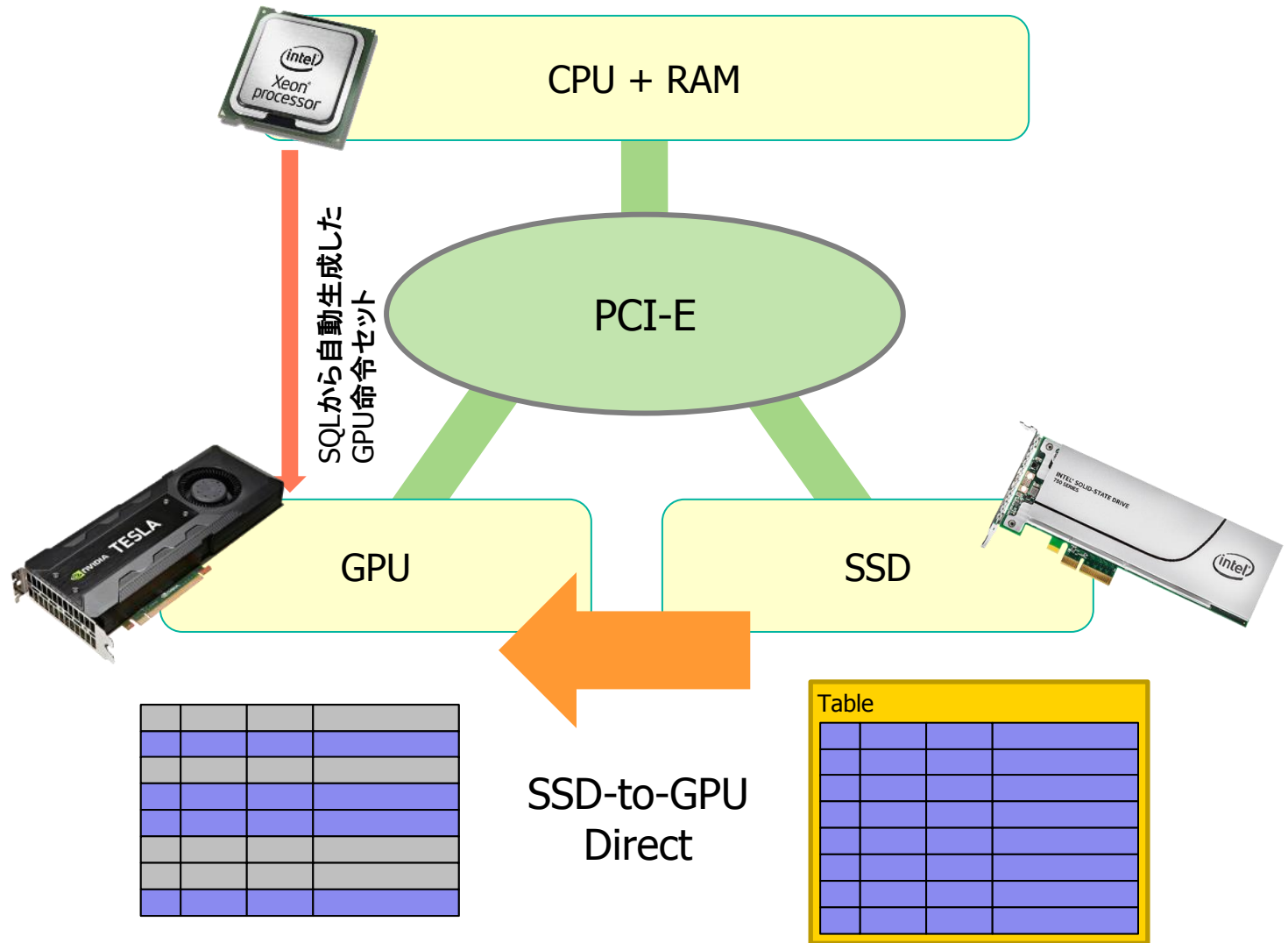
解析系クエリ実行時のデータフロー (1/3) – 基本パターン



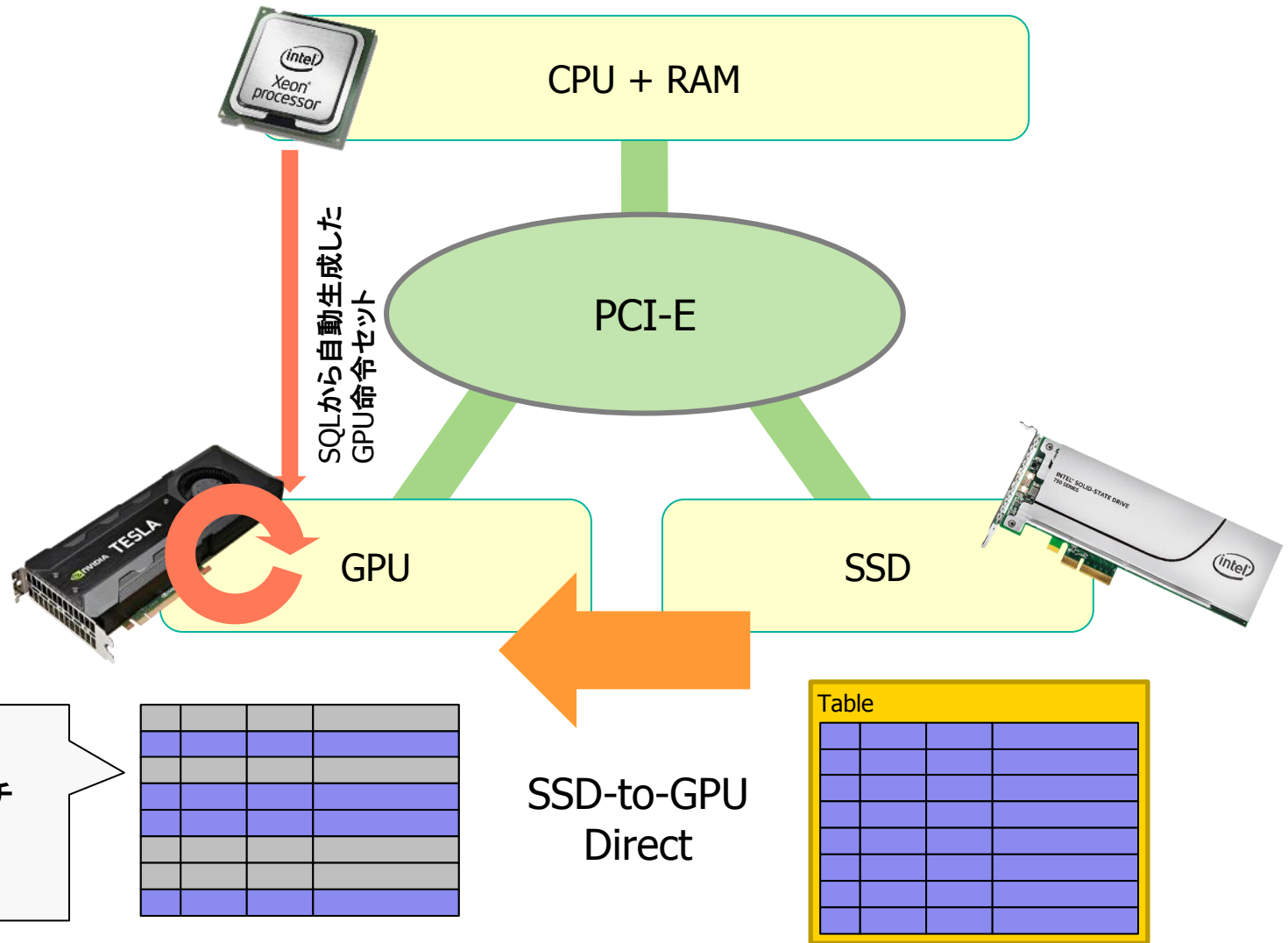
解析系クエリ実行時のデータフロー (1/3) – 基本パターン



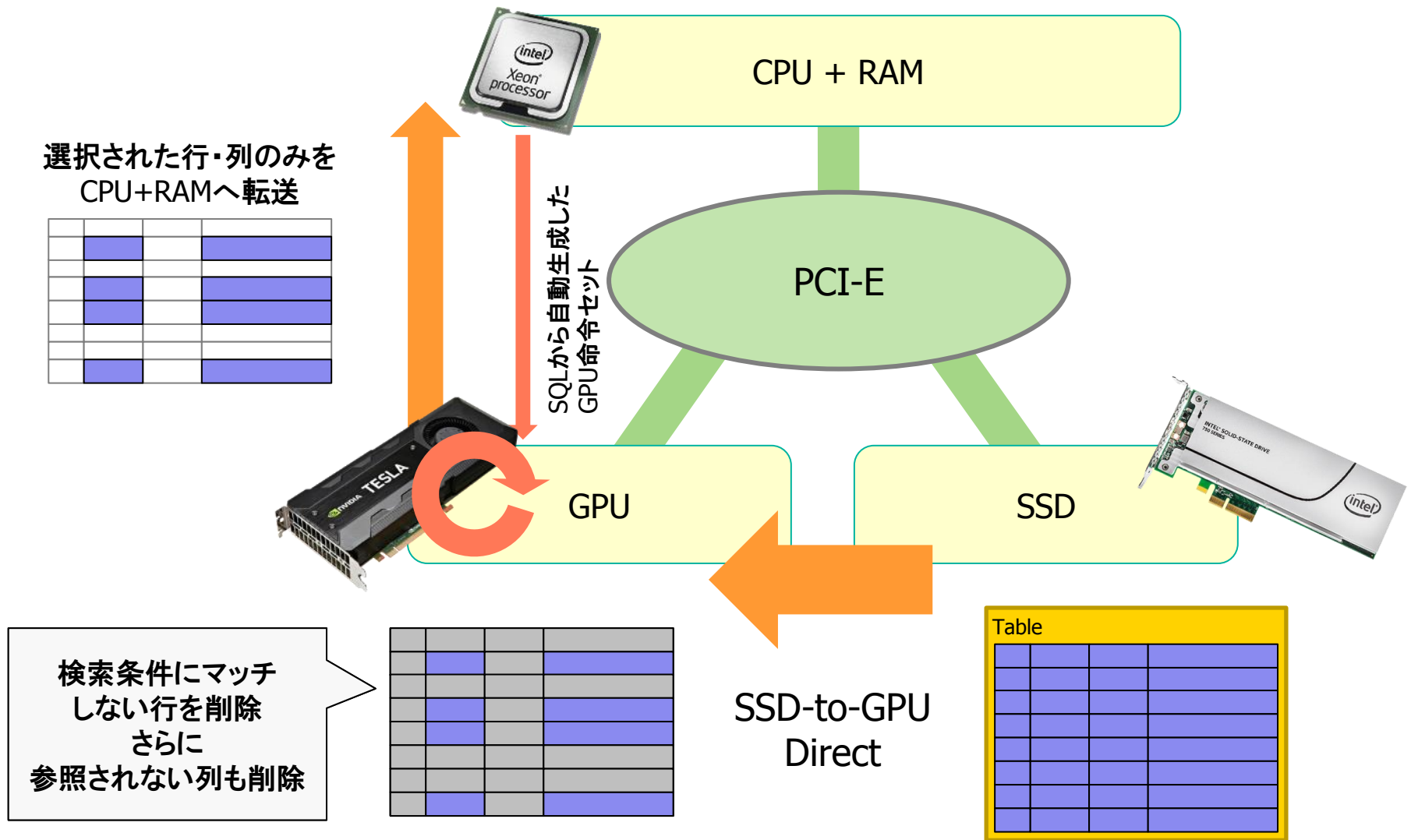
解析系クエリ実行時のデータフロー (2/3) – アレンジ版



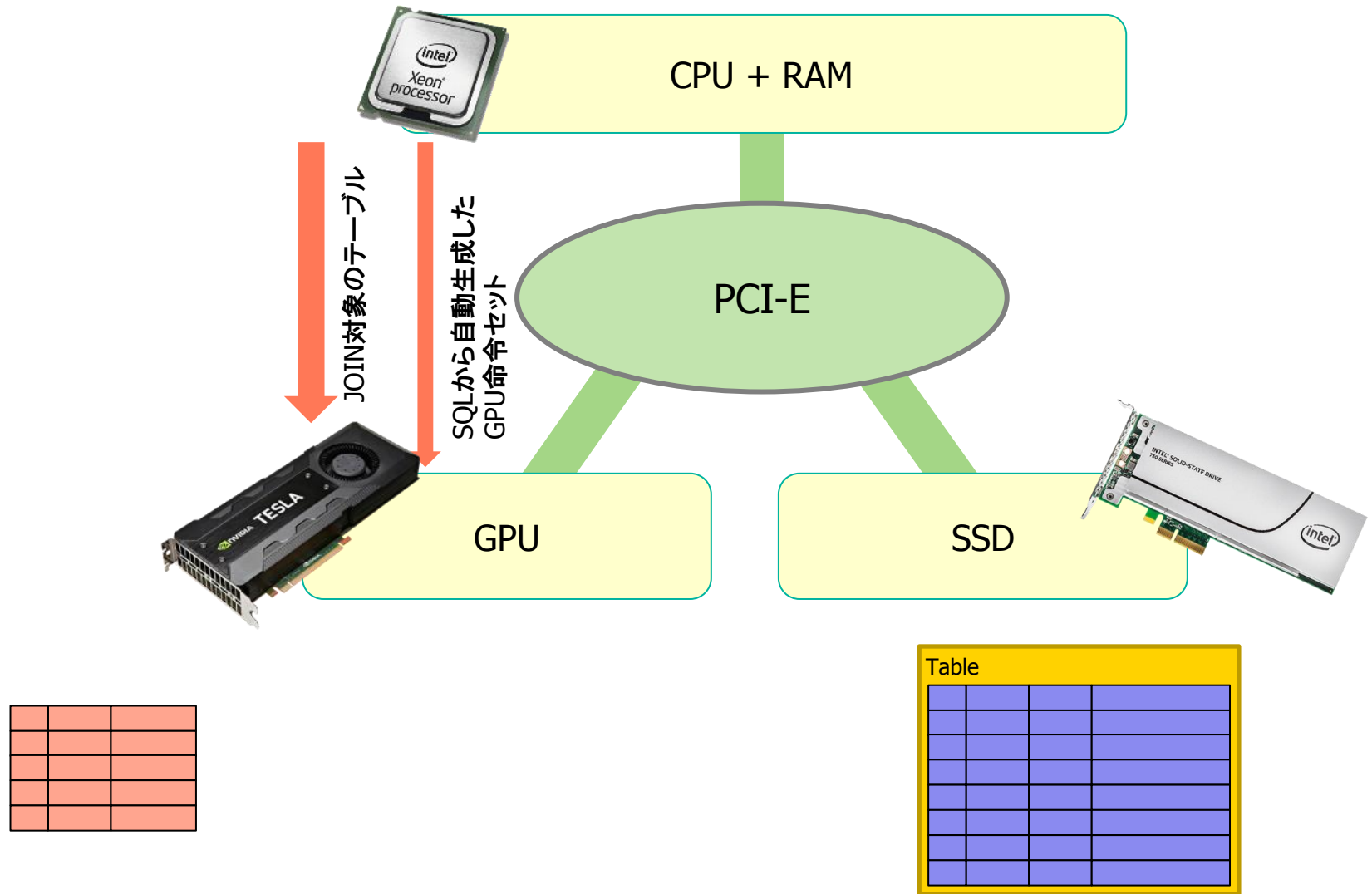
解析系クエリ実行時のデータフロー (2/3) – アレンジ版



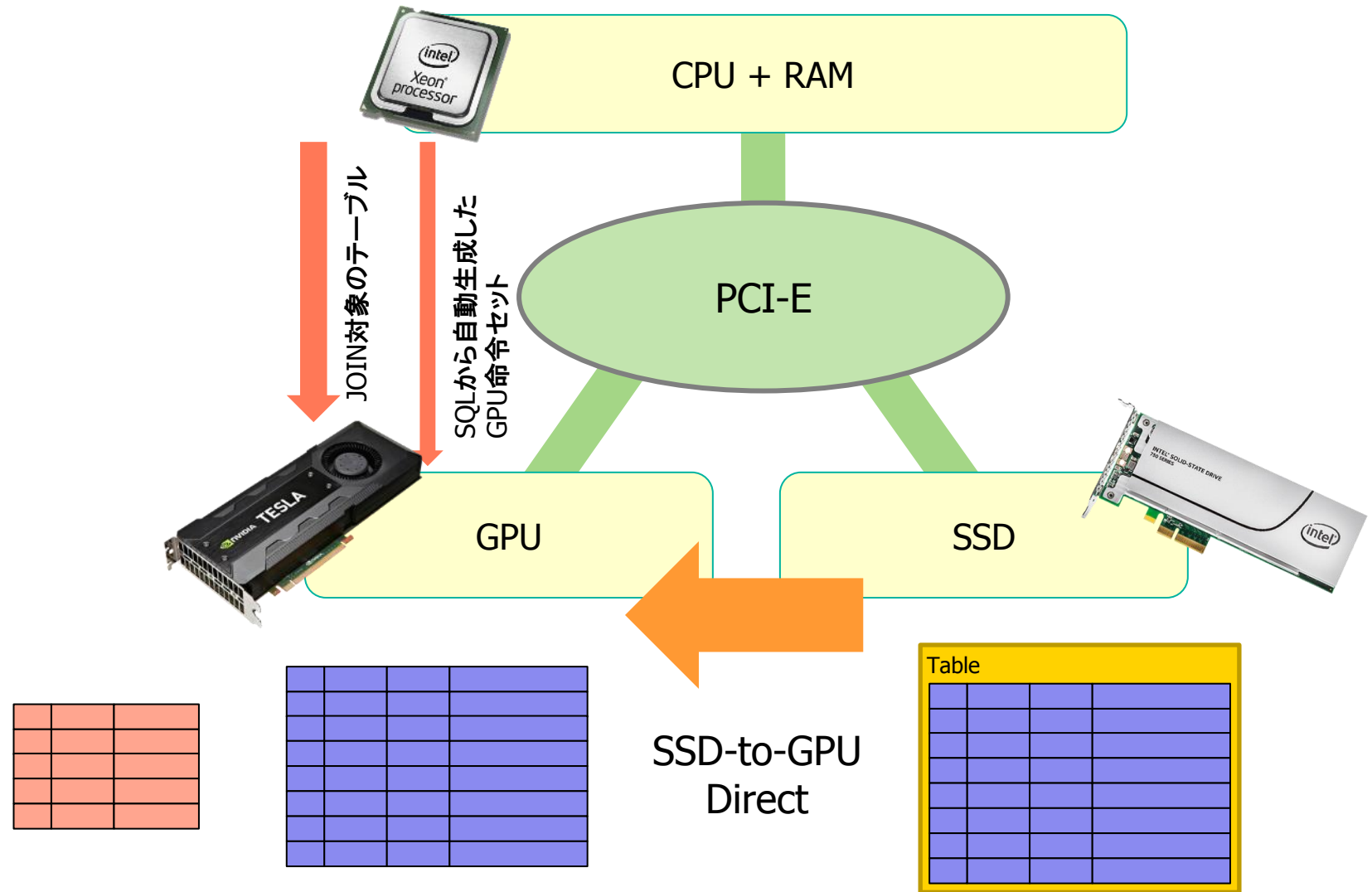
解析系クエリ実行時のデータフロー (2/3) – アレンジ版



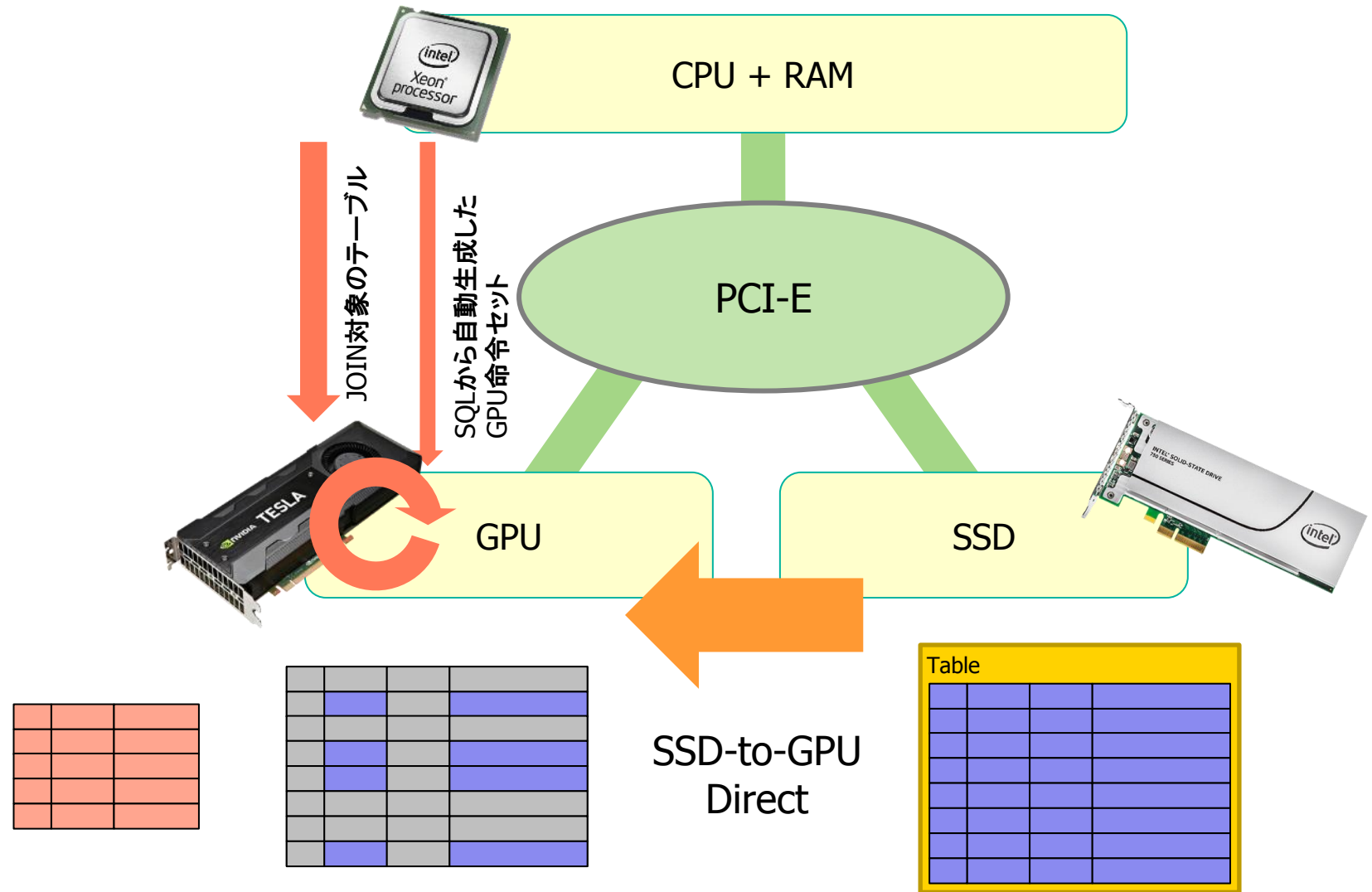
解析系クエリ実行時のデータフロー (3/3) – 鬼アレンジ版



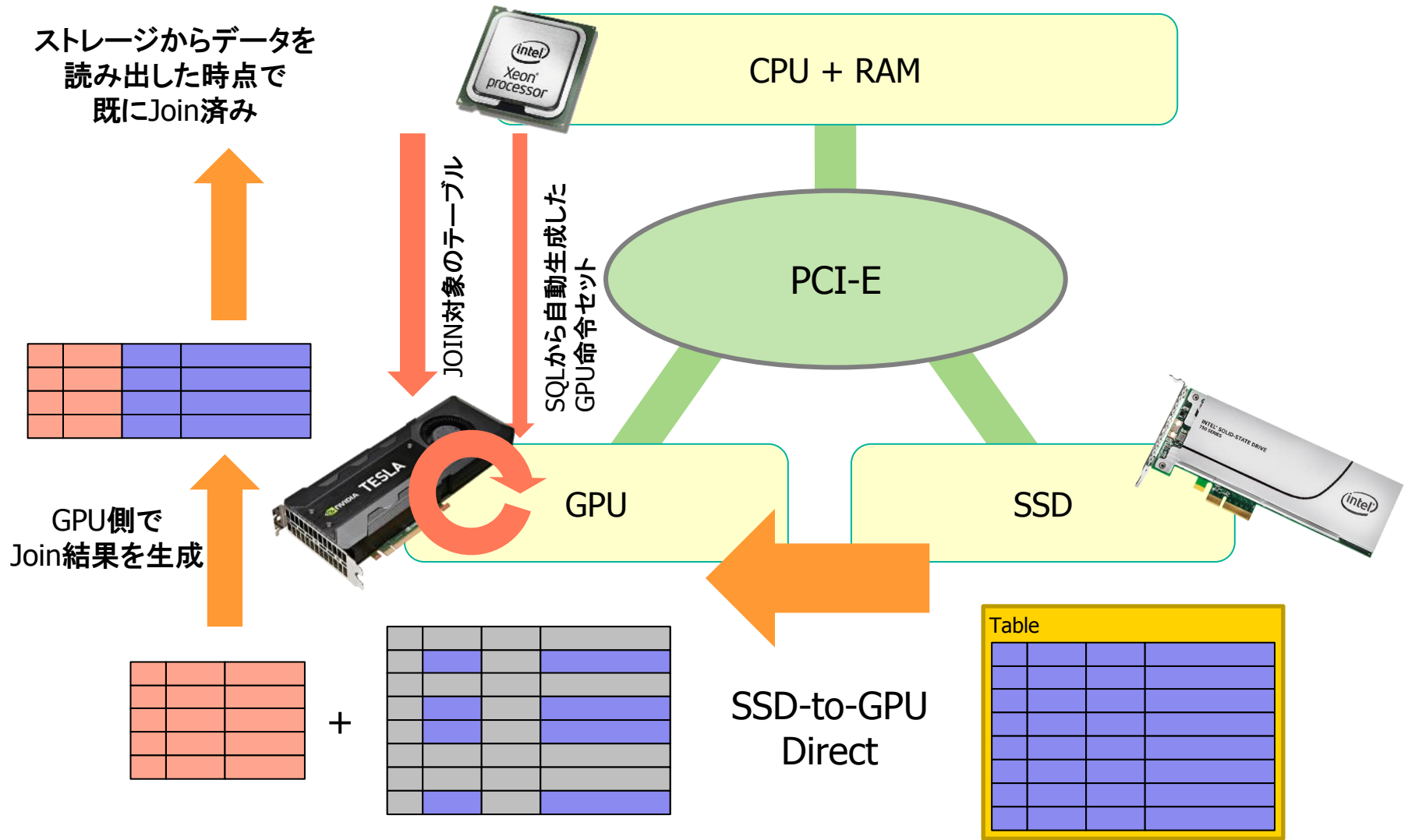
解析系クエリ実行時のデータフロー (3/3) – 鬼アレンジ版



解析系クエリ実行時のデータフロー (3/3) – 鬼アレンジ版



解析系クエリ実行時のデータフロー (3/3) – 鬼アレンジ版



必要な要素技術

■ NVMeドライバへの NVIDIA GPUDirect 機能の追加

- Linux kernel driverとnvidia driverの間にやり取りが必要....。

■ shared_bufferの利用率統計情報採取

- テーブルの大半が既にオンメモリなら意味はないので。

■ shared_bufferのアクセスモード追加

- SSD→GPU Direct転送が終わるまでは誰も書き込んじゃダメ。

PG-Stromプロジェクトへの参加者
熱烈歓迎大募集中

乞うご期待。

