
VOLTRON DATA

We've been building data systems for years

Darren Haas

Josh Patterson

RAPIDS

Rodrigo Aramburu

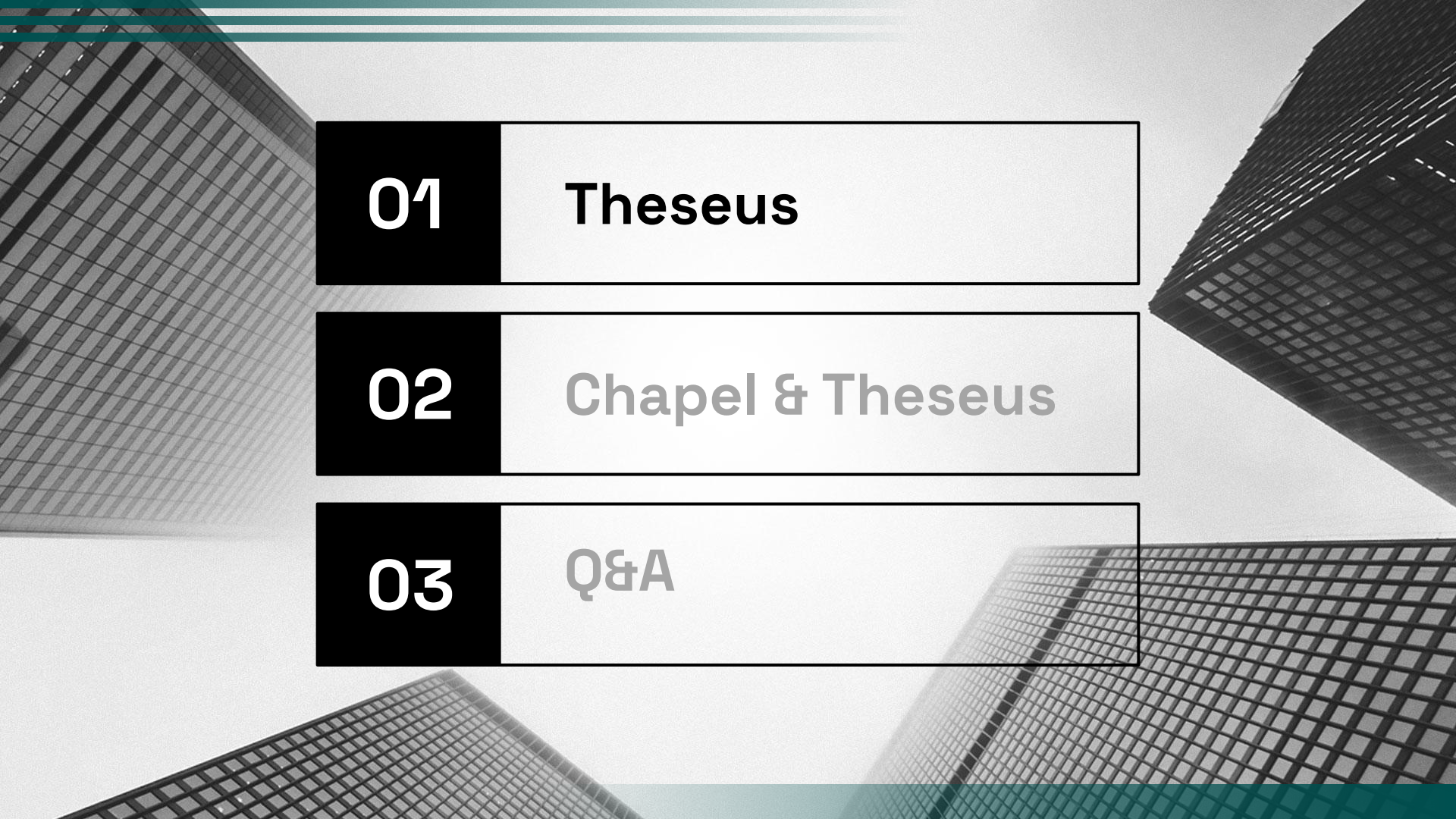
blazingSQL

Keith Kraus

RAPIDS NVIDIA

$110M Seed and Series A

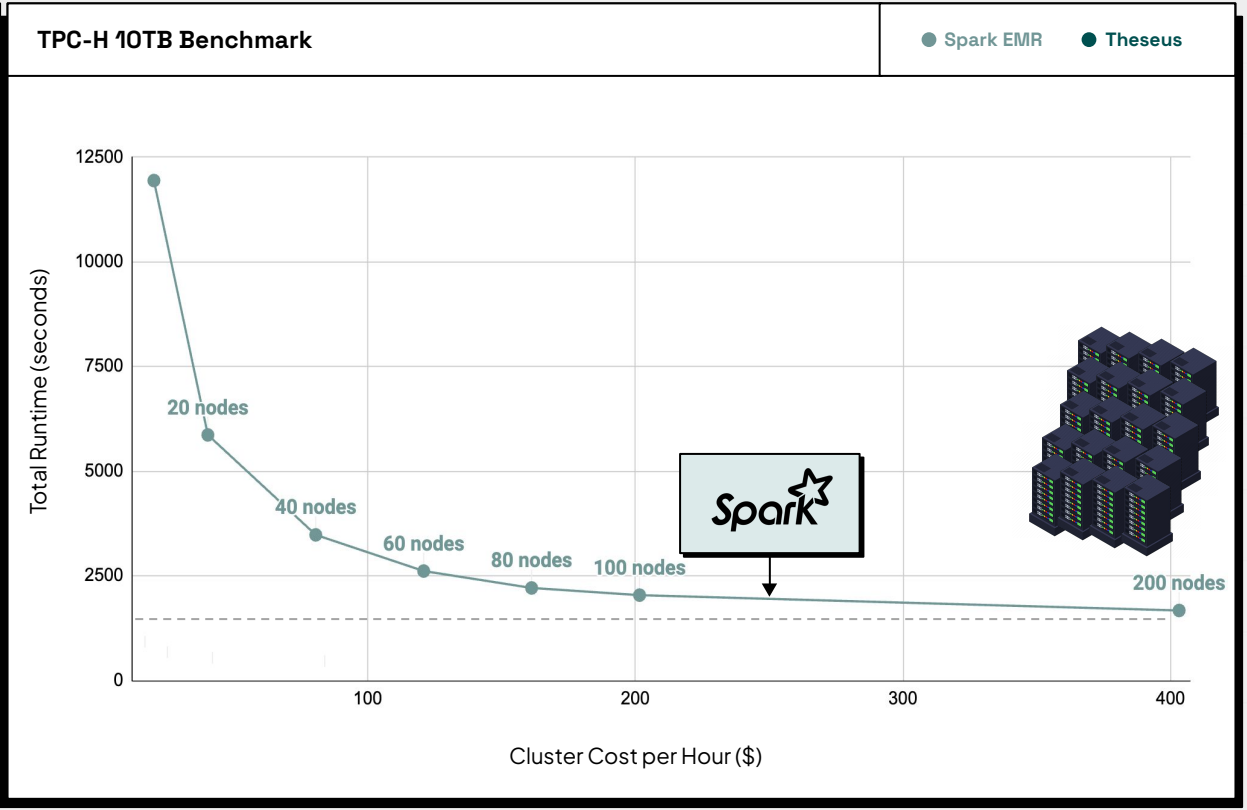
BlackRock FACTORY iqt IN-Q-TEL (w)C G/


01
Theseus
02
Chapel & Theseus
03
Q&A

The Wall

Spark vs. Theseus:

CPU performance is capped. No amount of money will jump over this wall.



Note: Theseus: 1 Node 8 x A100 80 GB, Spark: 1 Node r5.8xlarge (AWS) 32 VCPU 32 GB

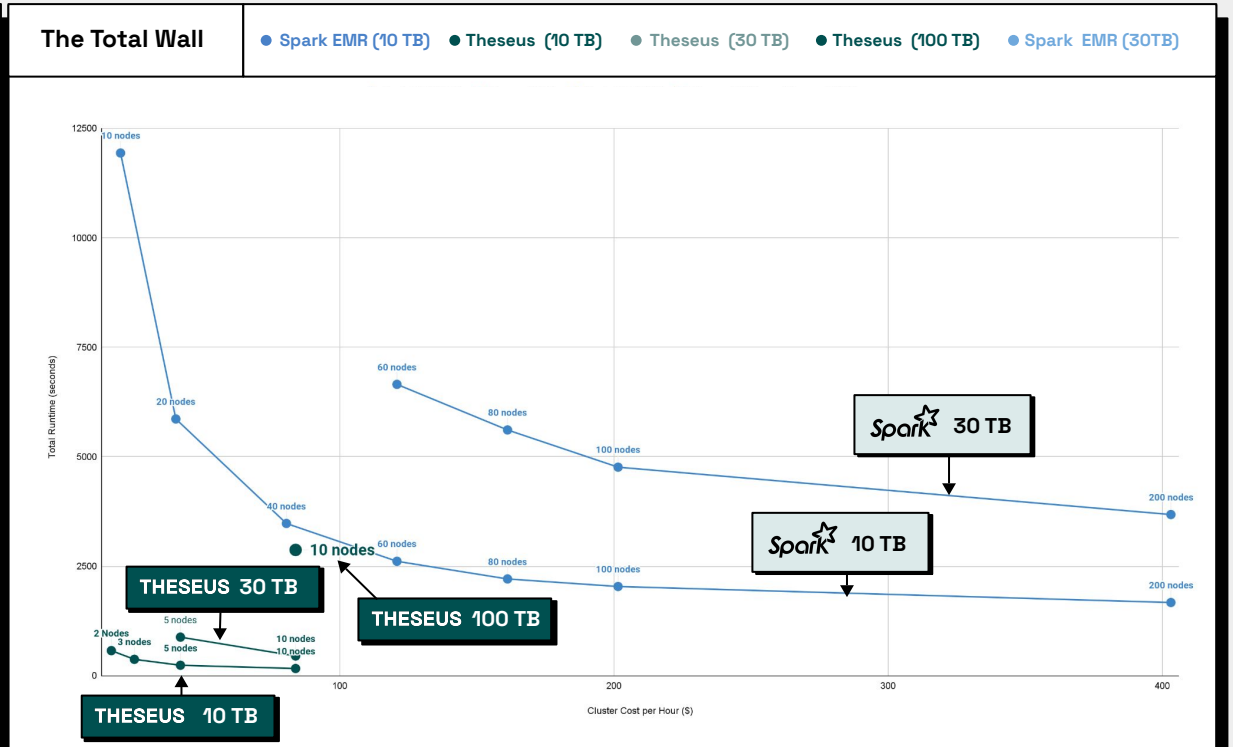
Scaling Datasets

TPC-H (10TB, 30TB, 100TB)

- ✓ **Up to 10 DGX Servers**
- ✓ **Parquet Files**
- ✓ **Remote File System**
- ✓ **Lots of Spilling**

- ✘ **No** Sorting
- ✘ **No** Indexing
- ✘ **No** Caching
- ✘ **No** Warm Up (Cold Queries)

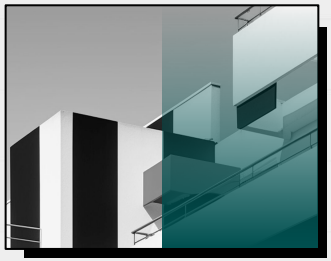
Note: Theseus: 1 Node 8 x A100 80 GB, Spark: 1 Node r5.8xlarge (AWS) 32 VCPU 32 GB



Theseus

A composable, accelerated data processing engine built on GPUs.

Scale to problems too big for Spark



Support efficient spilling out of GPU memory

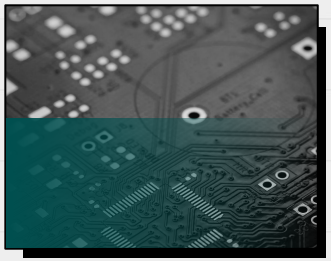
The only distributed GPU engine

72x faster, 71x cheaper

Linearly scale to massive data problems

Seamlessly move from Dev to Prod with the same code

Built from the ground up for accelerators



Upgrade and diversify hardware by leveraging GPUs, x86, ARM, Infiniband, RoCE, NVLink, and more

Harness underutilized generally available GPUs to improve economics

Reduce data center footprint by 6x

Evolve as enterprise needs change

Confidently leverage GPUs, x86, ARM, and future hardware innovations for data preprocessing

Process analytics and AI workflows within the same semiconductor

Use multiple programming languages (Python, R, Java, Rust, C++)

Operate on data where it is

What it's NOT

Framework

Database

Data Lake

Walled Garden

Monolith

SaaS

Data Warehouse

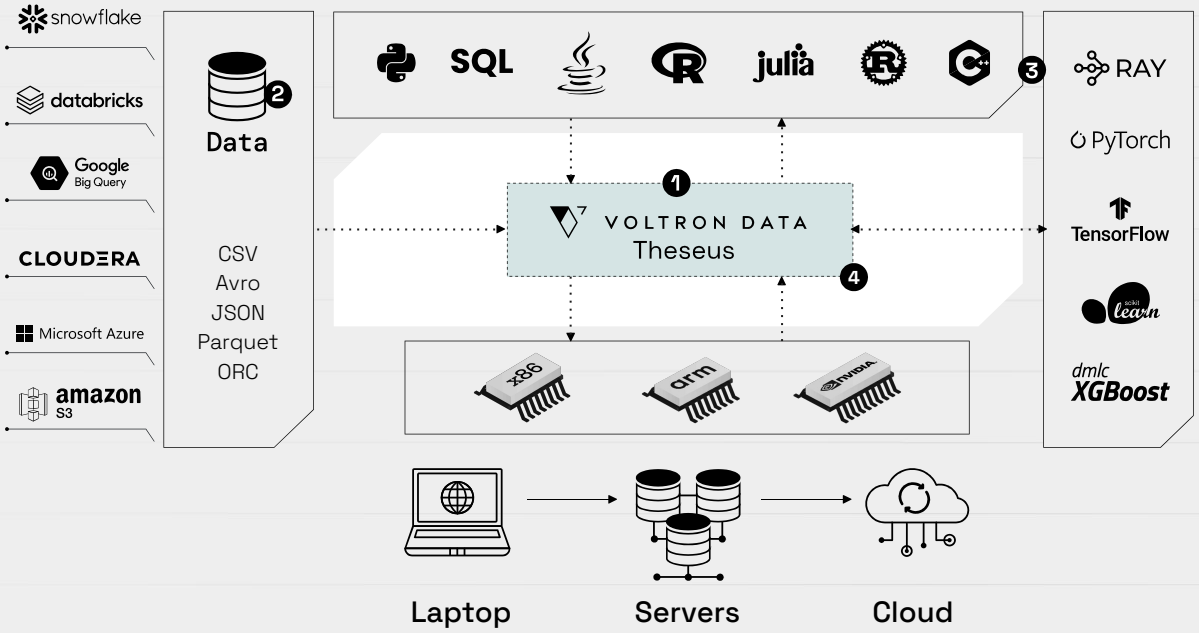
File System

Cloud

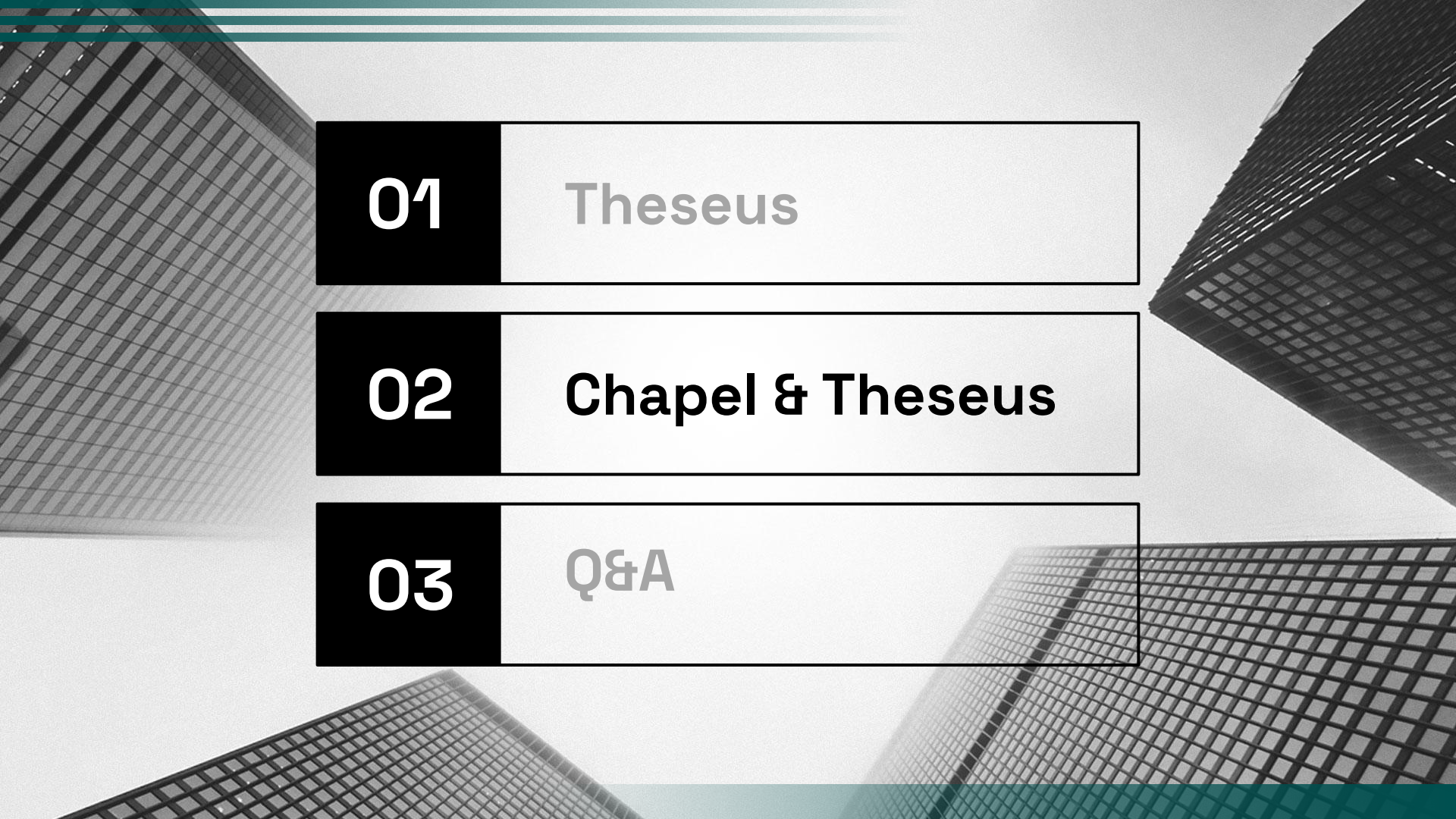
Support service

Voltron Data Theseus

A Compute Mesh unifying hardware, languages, and applications



1. **Accelerator-Native**: Distributed query engine built from the ground up to take advantage of full system hardware acceleration.
2. **Petabyte Scale**: Focusing on problems too big and time sensitive for Spark
3. **Composable:** Built on open source standards that enables interoperability from storage to application
4. **Evolutionary:** A composable engine that seamlessly adapts to new hardware and languages


01
Theseus
02
Chapel & Theseus
03
Q&A

Theseus: User Defined Functions

- At its heart, Theseus is a GPU-accelerated SQL engine
- Its bread and butter is relational database operations; joins, filtering, aggregation, projections, window functions, etc.
- HPC, on the other hand, has far more diverse computational requirements
- Enter: User Defined Functions (UDFs). Theseus has robust UDF support that allows any columnar-oriented routines to be called directly from SQL, e.g. `SELECT l_itemnum, foo.postprocess(l_itemdetails) FROM …"
- Theseus UDF framework is in C++; you write a C++ module in line with our guide and voila, you're now a first-class SQL citizen.

Theseus & Chapel

Leveraging the power of Chapel from within Theseus

- Early-access customers expressed a strong interest in having Theseus support Chapel as a first-class SQL citizen
- Two things that made this very easy to implement:
  - Chapel has great C interoperability!
  - The `chapel` Python module has great AST introspection support!
- So you can write your columnar-oriented routines (i.e. parameters are arrays) in Chapel, run our helper Python script against that `.chpl` file, and voila, we auto-gen the necessary C++ scaffolding to bridge the gap between C++ and Chapel at runtime.

Chapel Support

- Write your Chapel routine in one of two ways, depending on how you want to “return” data back to Theseus

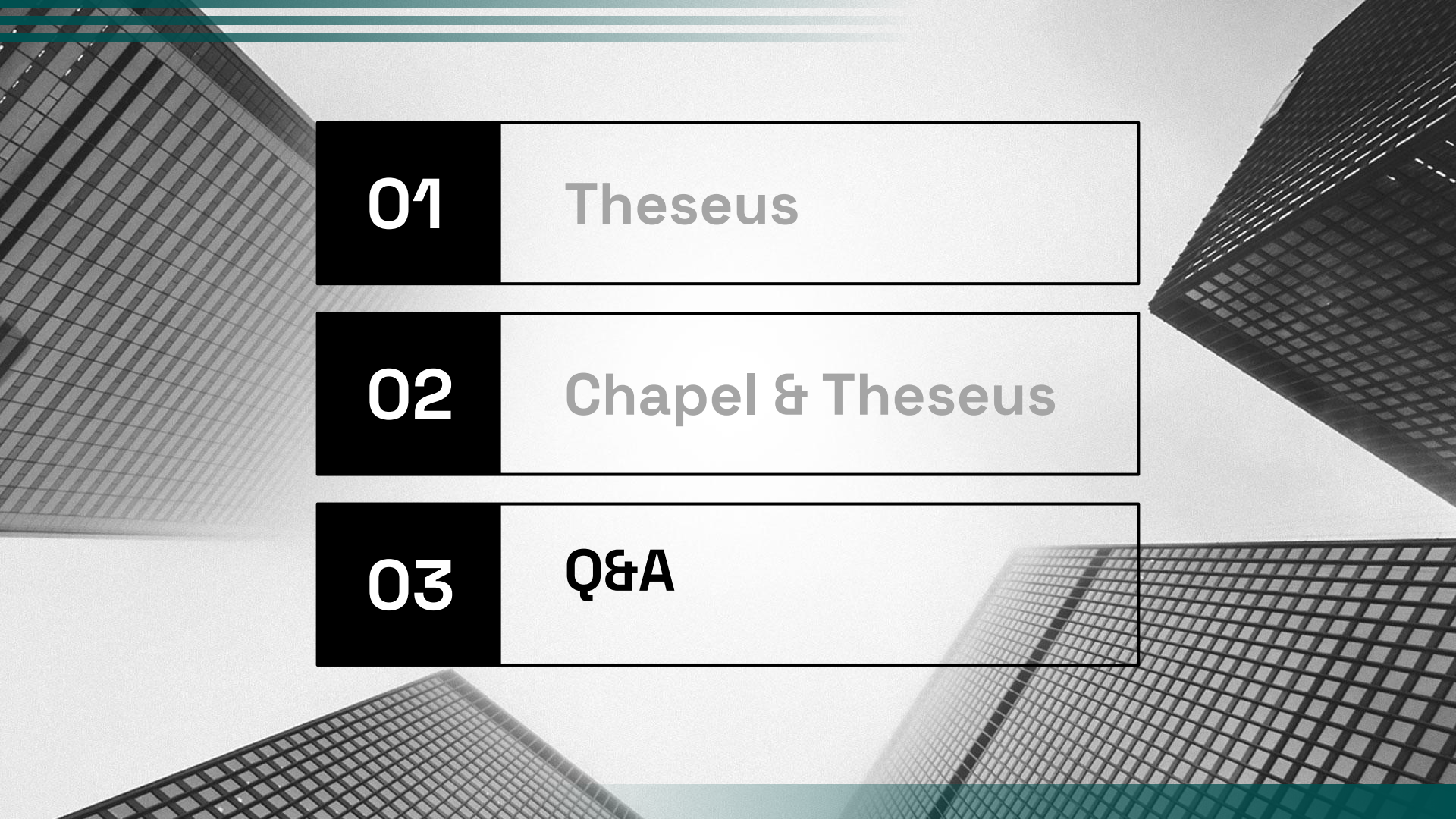
We allocate your `result` array and you just write into it like normal:

```
export proc add_int64(ref result: [] int(64),
                      const ref a: [] int(64),
                      const ref b: [] int(64)) {
  on here.gpus[0] {
    forall i in a.domain {
      result[i] = a[i] + b[i];
    }
  }
}
```

Or, you return us an array you allocated, by way of chpl_externay_array:

```
export proc add_int64(const ref a: [] int(64),
                      const ref b: [] int(64)) : chpl_external_array {
  var retval: chpl_external_array;
  on here.gpus[0] {
    var result: [a.domain] a.type;
    forall i in a.domain {
      result[i] = a[i] + b[i];
    }
    retval = convertToExternalArray(result);
  }
  return retval;
}
```

- Then just call your routine directly from SQL:
  `SELECT ChapelPlugin.add_int64(t.foo, t.bar) FROM table t`


01
Theseus
02
Chapel & Theseus
03
Q&A

Theseus - Infrastructure Recommendations

What would an ideal data system look like?

Minimum Hardware	Better Hardware	Ideal Hardware
NVIDIA DGX V100 (32GB GPUs) Servers	NVIDIA DGX A100 (40GB GPUs) Servers	NVIDIA DGX H100/A100 80GB Servers
64GB RAM per NVIDIA GPU	80GB RAM per NVIDIA GPU	160GB RAM per NVIDIA GPU
100 Gbps Ethernet Networking, 1 NIC shared b/w 2 GPUs	100 Gbps RoCE Networking, 1 NIC per GPU	200+ Gbps Infiniband Networking
	RoCE Network Attached Storage	Infiniband Network Attached Storage