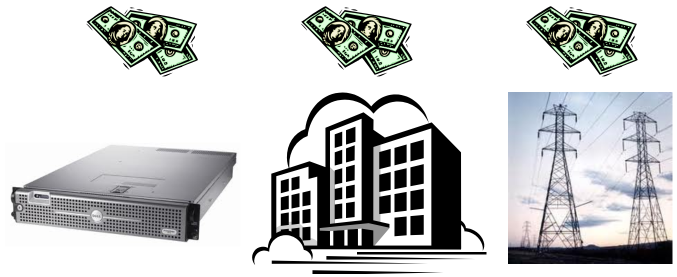
Dark Silicon Accelerators for Database Indexing

Onur Kocberber, Kevin Lim, Babak Falsafi,
Partha Ranganathan, Stavros Harizopoulos

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

Dark Silicon and Big Data Challenges

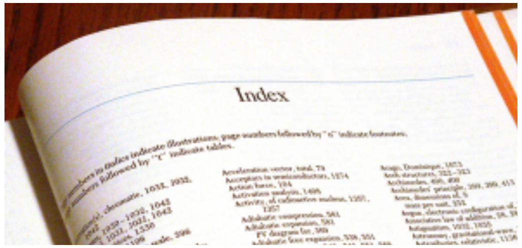
- Data explosion
  - Data growing faster than technology
- End of "Free energy"
  - Higher density ⇒ higher energy



- Challenge: CPUs ill-matched to server workloads
  - Most of time waiting for data rather than computing

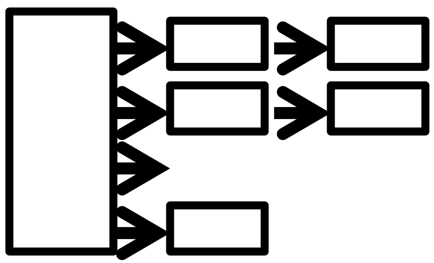
Need to specialize for data-centric workloads

How Do Data-Centric Workloads Access Data?

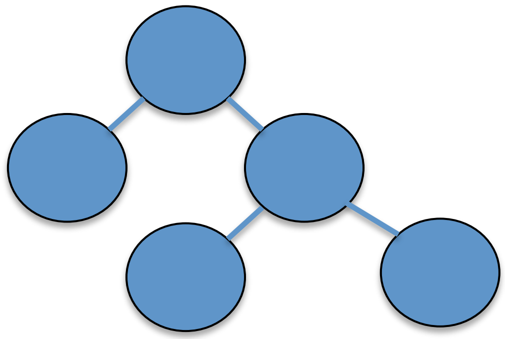


- Databases create and use an **index**
  - Data structures for fast data lookup
  - Most often balanced tree or hash table
  - Frequently accessed

Hash Table



Tree



- Indexing is pointer-intensive
  - Underutilize general-purpose CPUs
  - IPCs as low as **0.25** on OoO core

Contribution: Database Indexing Widget

- Index lookups on general-purpose CPUs:
  - Pointer-intensive → low IPC
  - Time-intensive → poor energy-efficiency
- Database Indexing Widget
  - Dedicated hardware for database index lookups
  - Full-service offload: core sleeps when widget runs
  - Up to 65% less energy per query

Outline

Introduction

Indexing in Databases

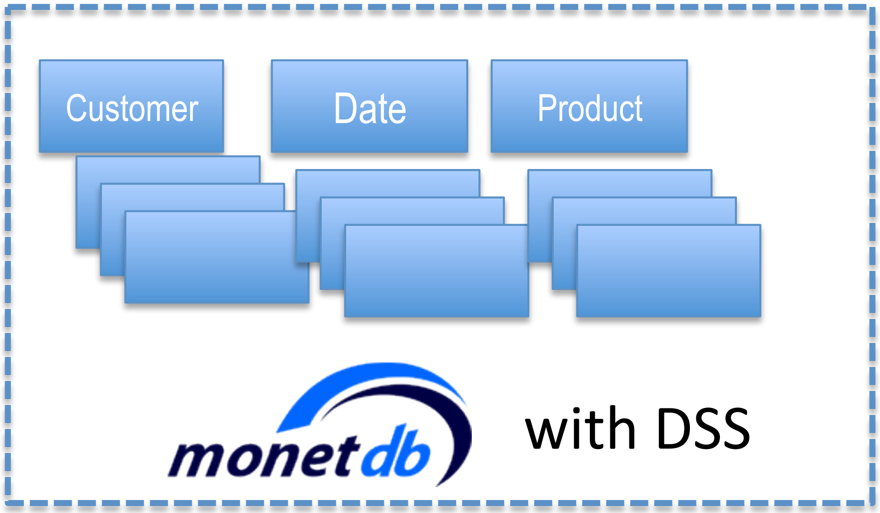
Indexing Widget

Results

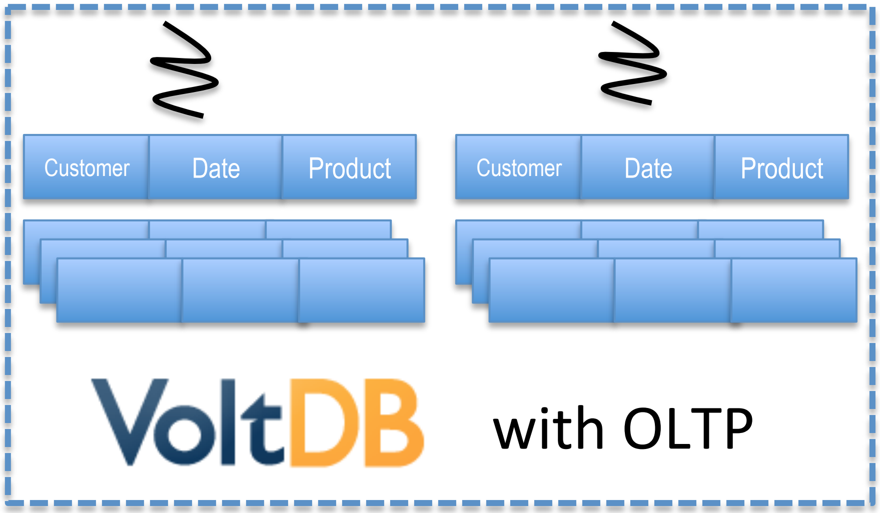
Modern Databases and Indexing

Two types of contemporary in-memory databases:

Column-store analytical processing



Scale-out transaction processing

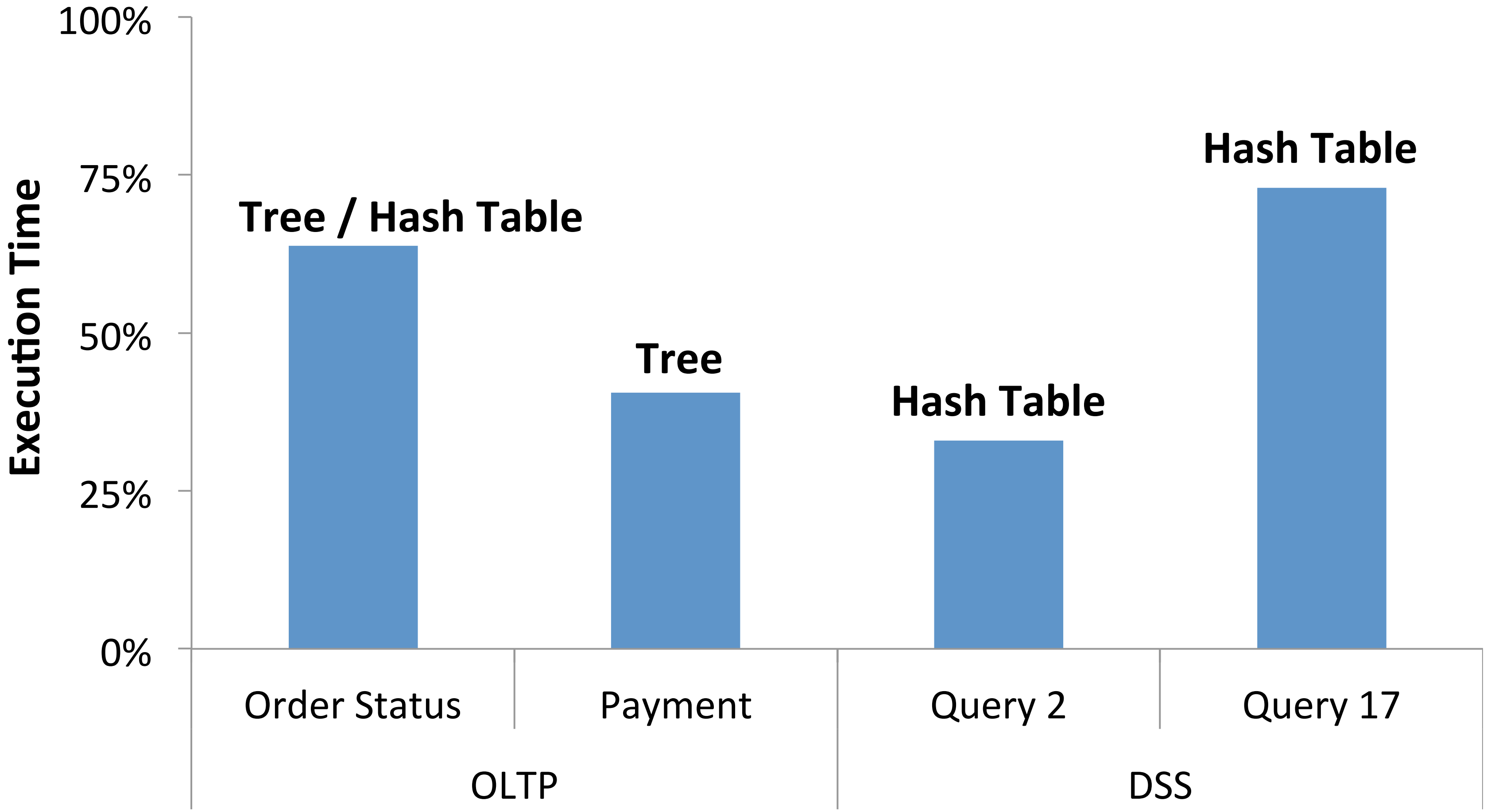


- Two fundamental indexing operations
  - –Hash table probe
  - –Tree traversal

ecocloud

How Much Time is Spent Indexing?

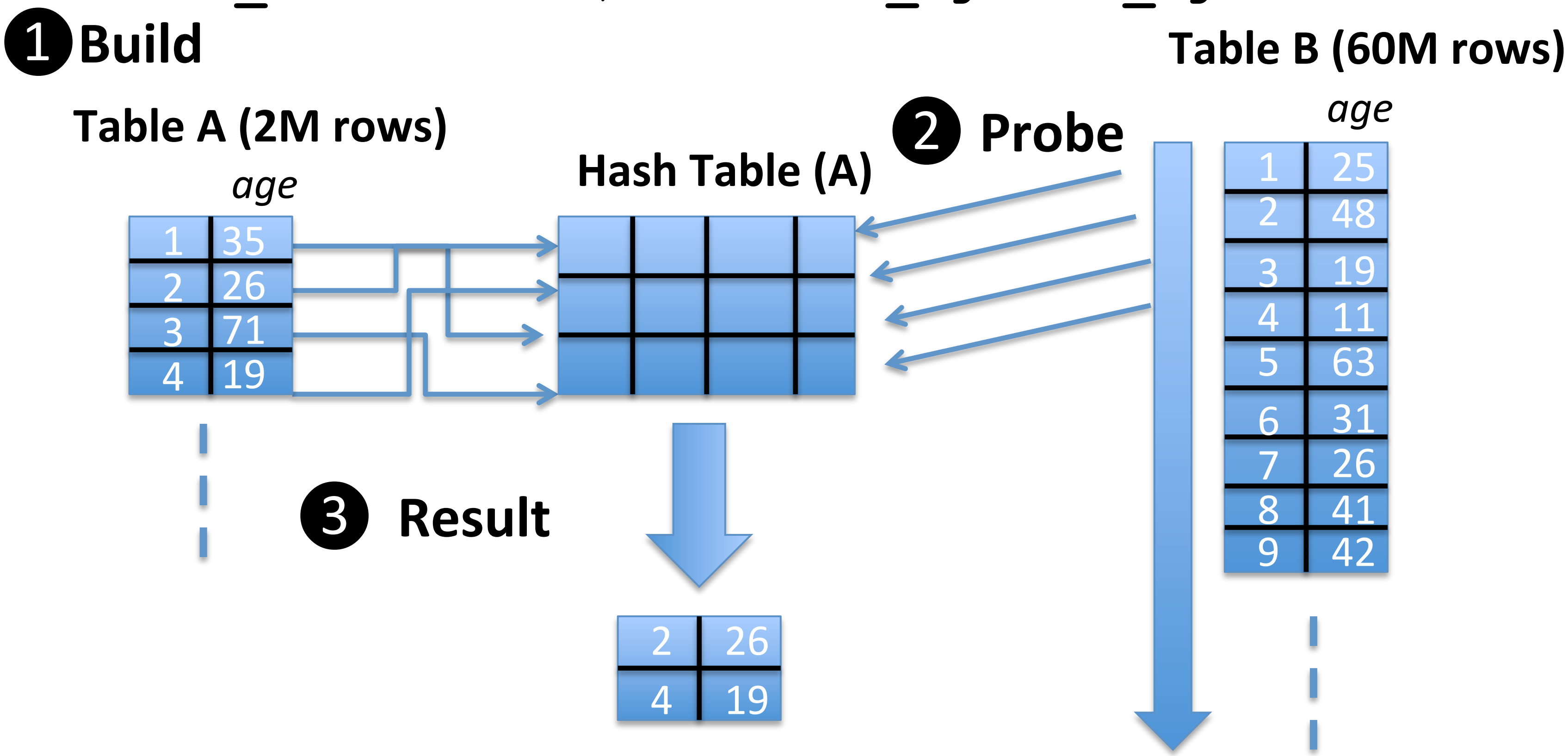
Measurement on Xeon 5670 CPU with HW Counters



Indexing can account for up to 73% of execution

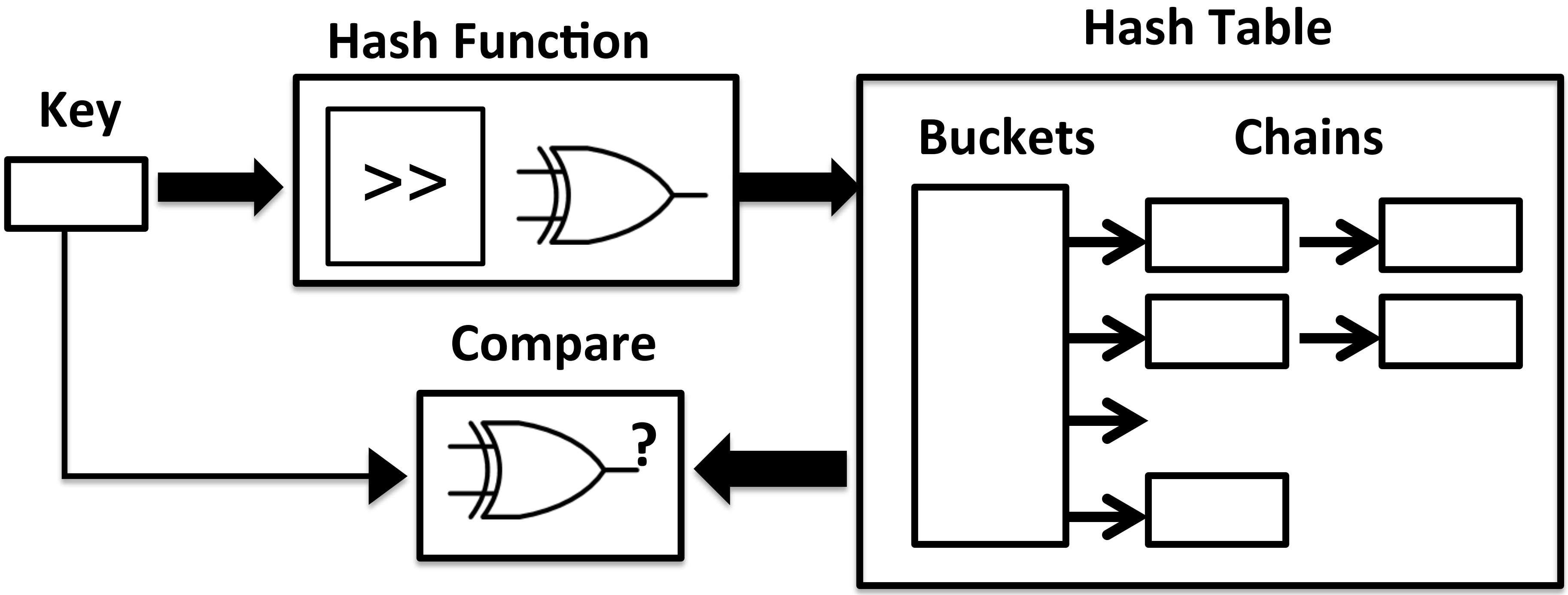
Example: Hash Join

```
SQL :
SELECT A_name FROM A,B WHERE A_age = B_age
```



Hash table probes dominate execution

Indexing with Hash Table Probes



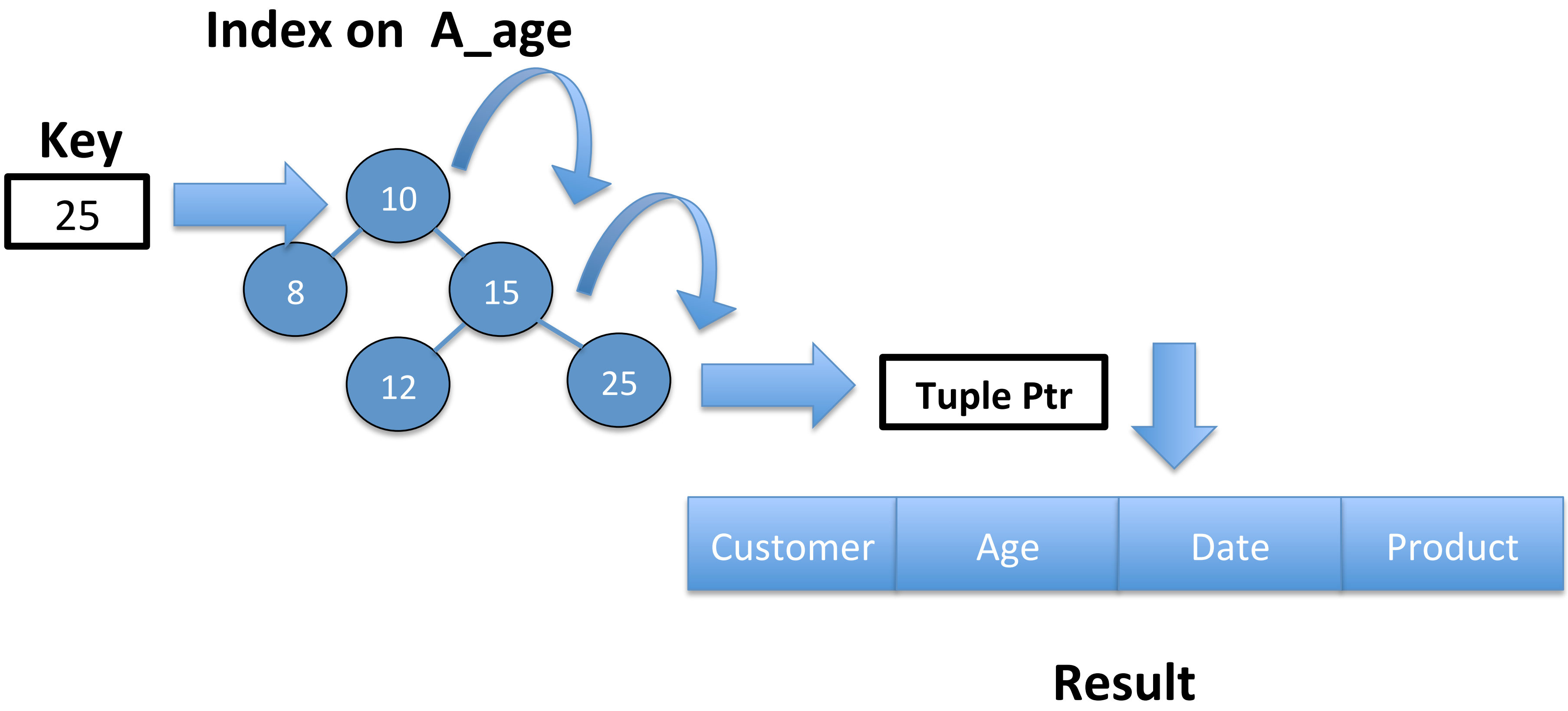
Each hash probe operation:

→100-200 dynamic instructions: hash, then chase pointers

→50% memory ref.

Indexing with Tree Traversals

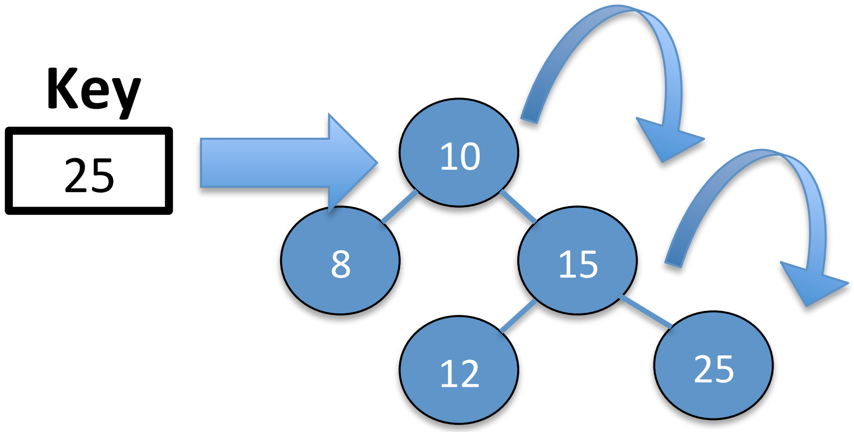
```
SQL :
SELECT A_Product,A_Customer FROM A WHERE A_age = 25
```



Indexing with Tree Traversals

```
SQL :
SELECT A_Product,A_Customer FROM A WHERE A_age = 25
```

Index on A_age



Each index traversal :

→10K-15K dynamic instructions: lots of pointer chasing

→50-60% memory ref.

Outline

Introduction

Indexing in Databases

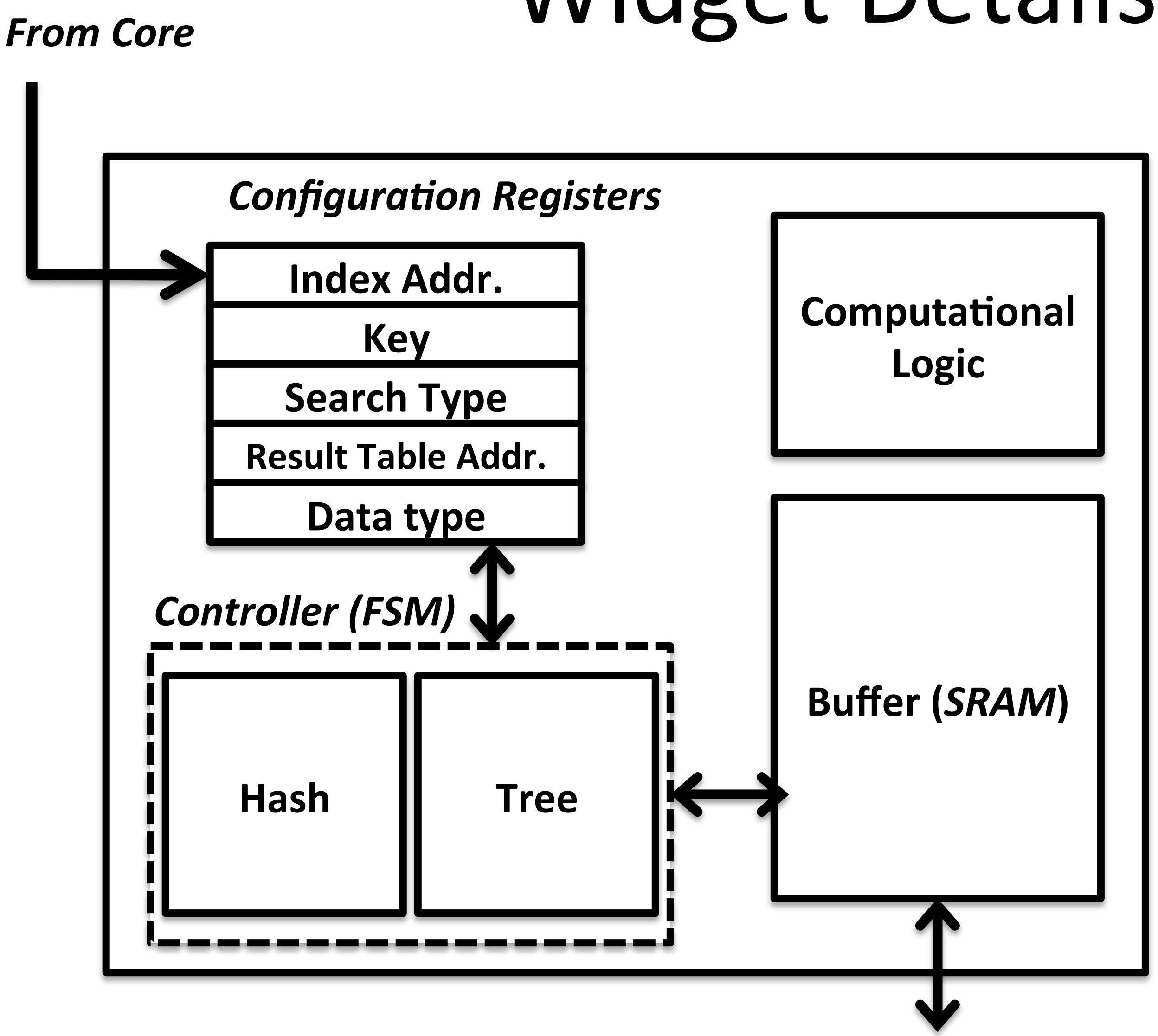
Indexing Widget

Results

Indexing Widget Overview

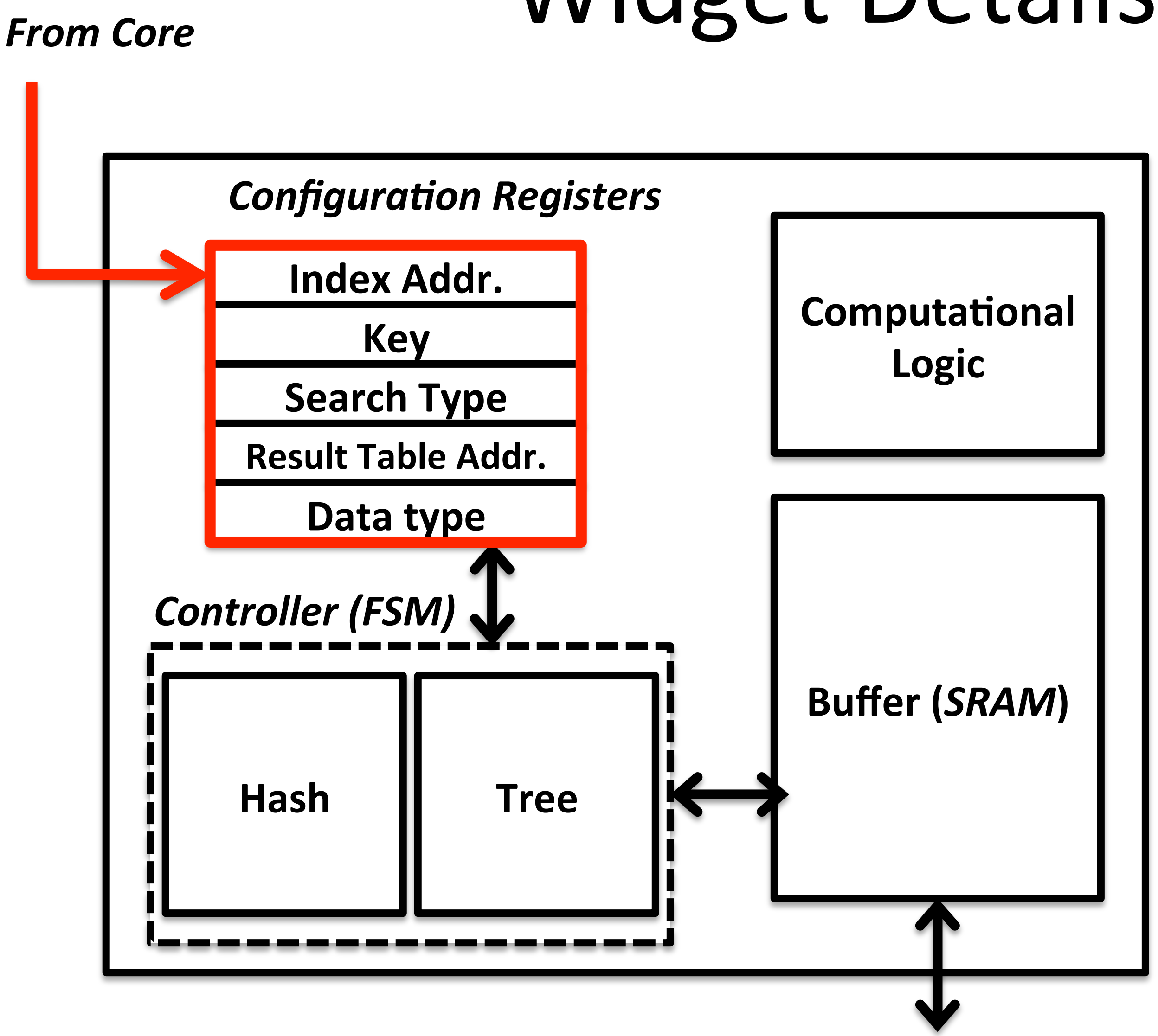
- Dedicated offload engine for index lookups
  - Activated on-demand by the core
  - Full-service index lookup
  - Core sleeps when widget runs
- Widget features
  - Efficient: Specialized control and functional units
  - Low-latency: Caches frequently-accessed index data
  - Tightly-integrated: Uses core's L1-D and TLB

Widget Details



1. Configure
2. Run
3. Return

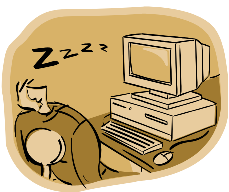
Widget Details

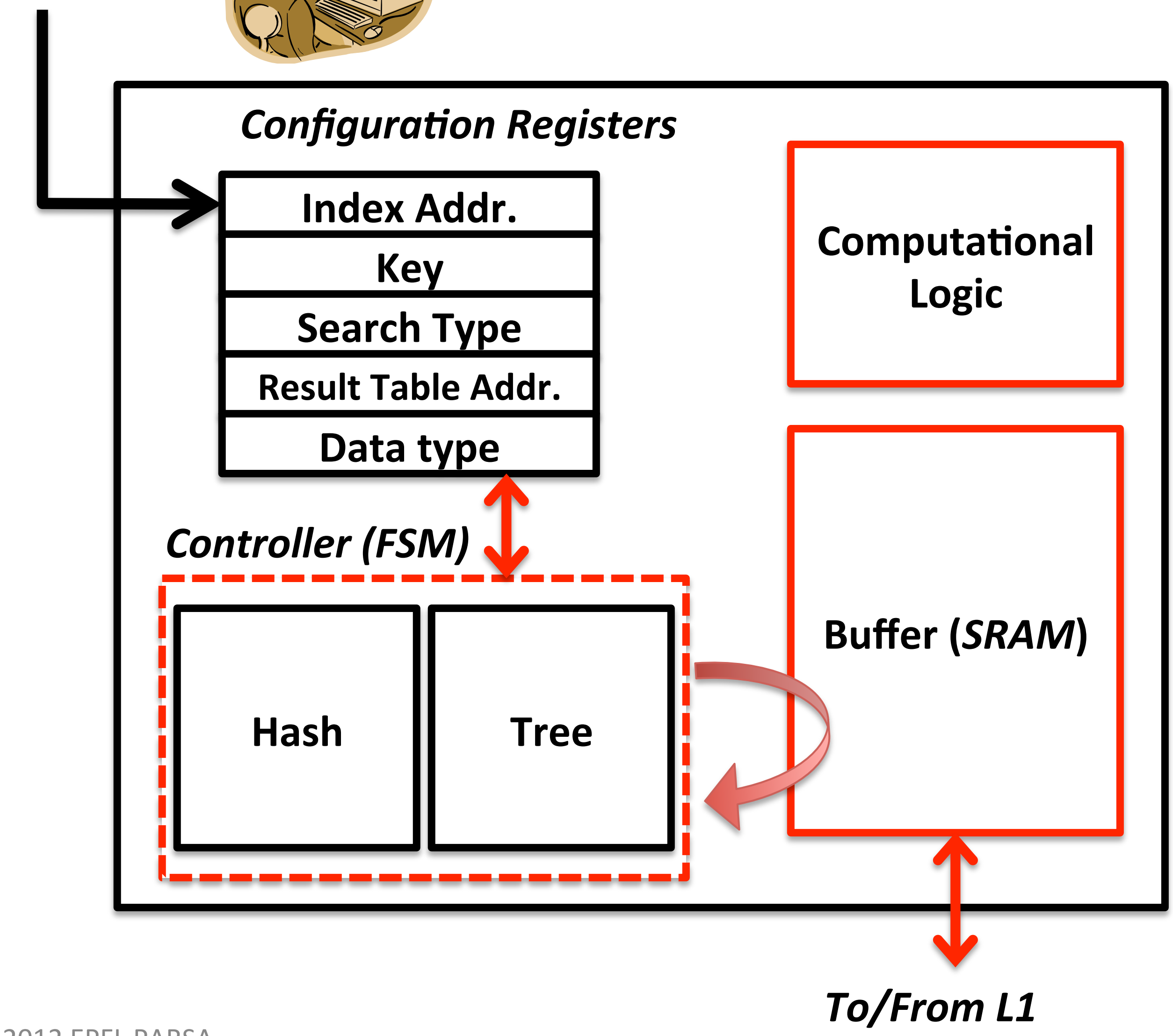


❶ Configure

```
If (hasWidget) {
widget.index=&A;
widget.key=&B;
widget.type=EQUAL;
widget.result=&R;
widget.data= int;
…
…
widget.run();
} else {
Hashprobe();
}
```

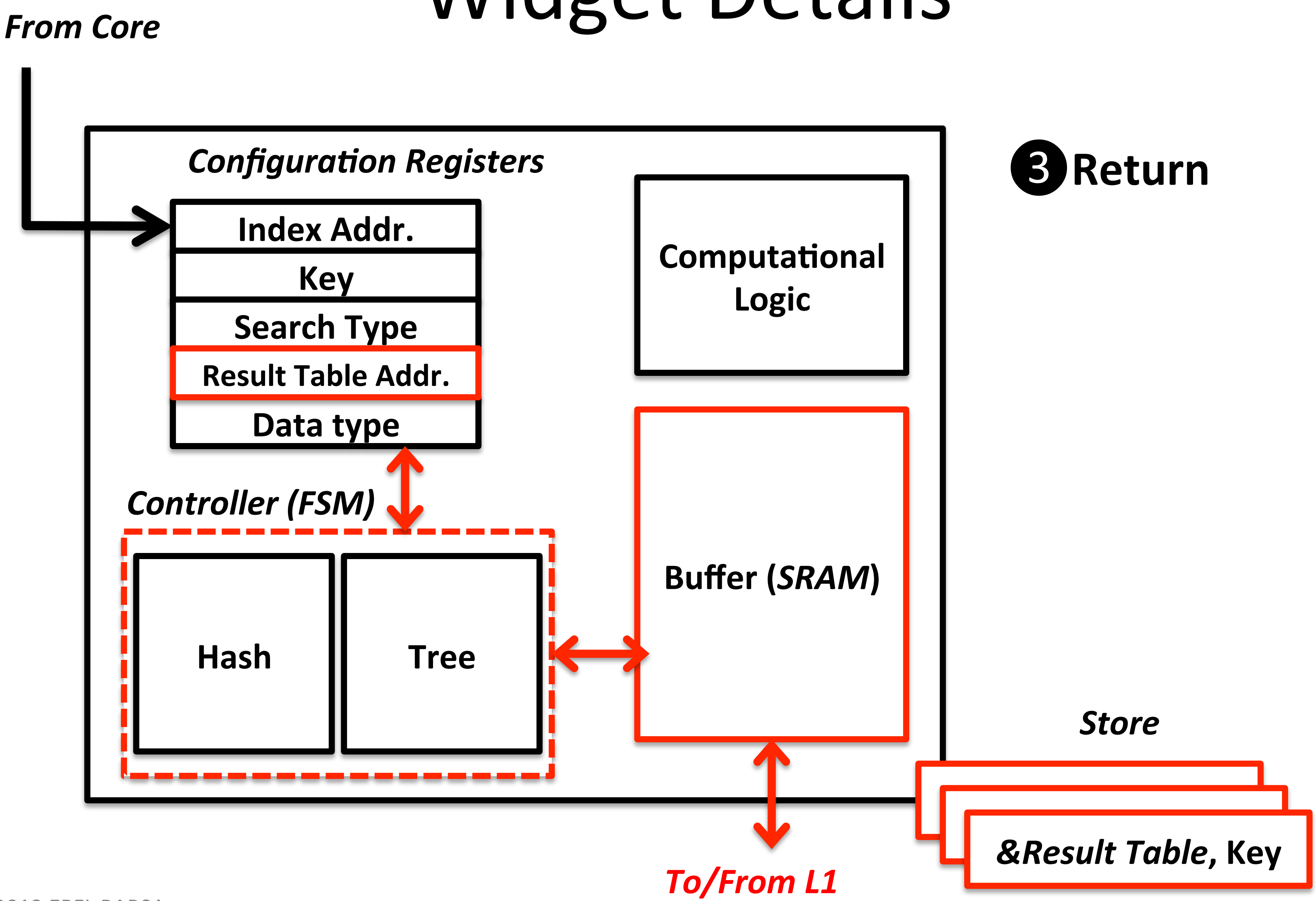
Widget Details





❷ Run

Widget Details



Methodology

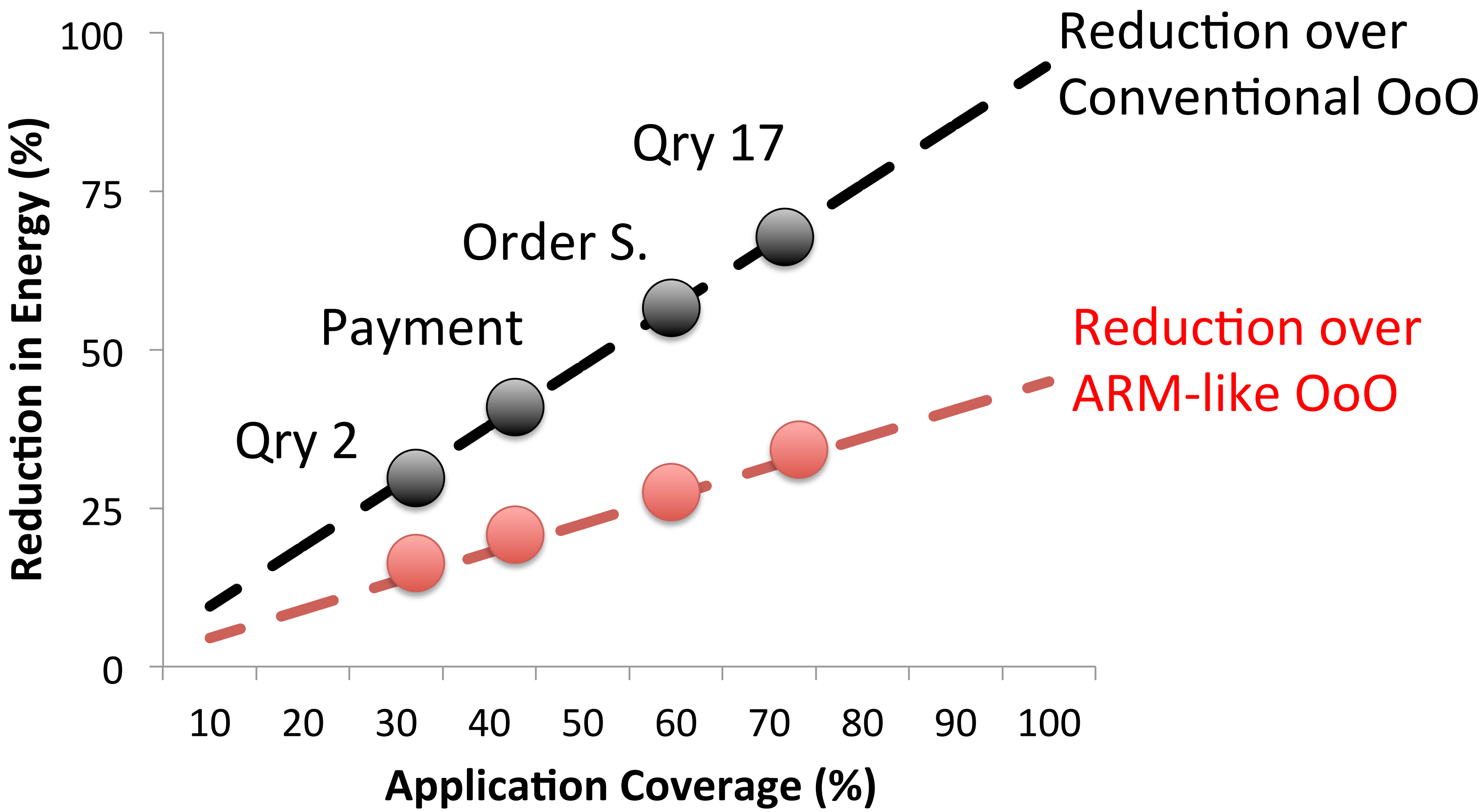
- First-order analytical model
  - Execution traces: Pin
  - Execution profiling: Vtune, Oprofile

- Benchmark Applications
  - OLTP: TPC-C on VoltDB
  - DSS: TPC-H on MonetDB

- Model Parameters
  - L1 / L2 / Off-chip latency: 2 / 12 / 200 cycles
  - Widget buffer: 2-way set associative cache

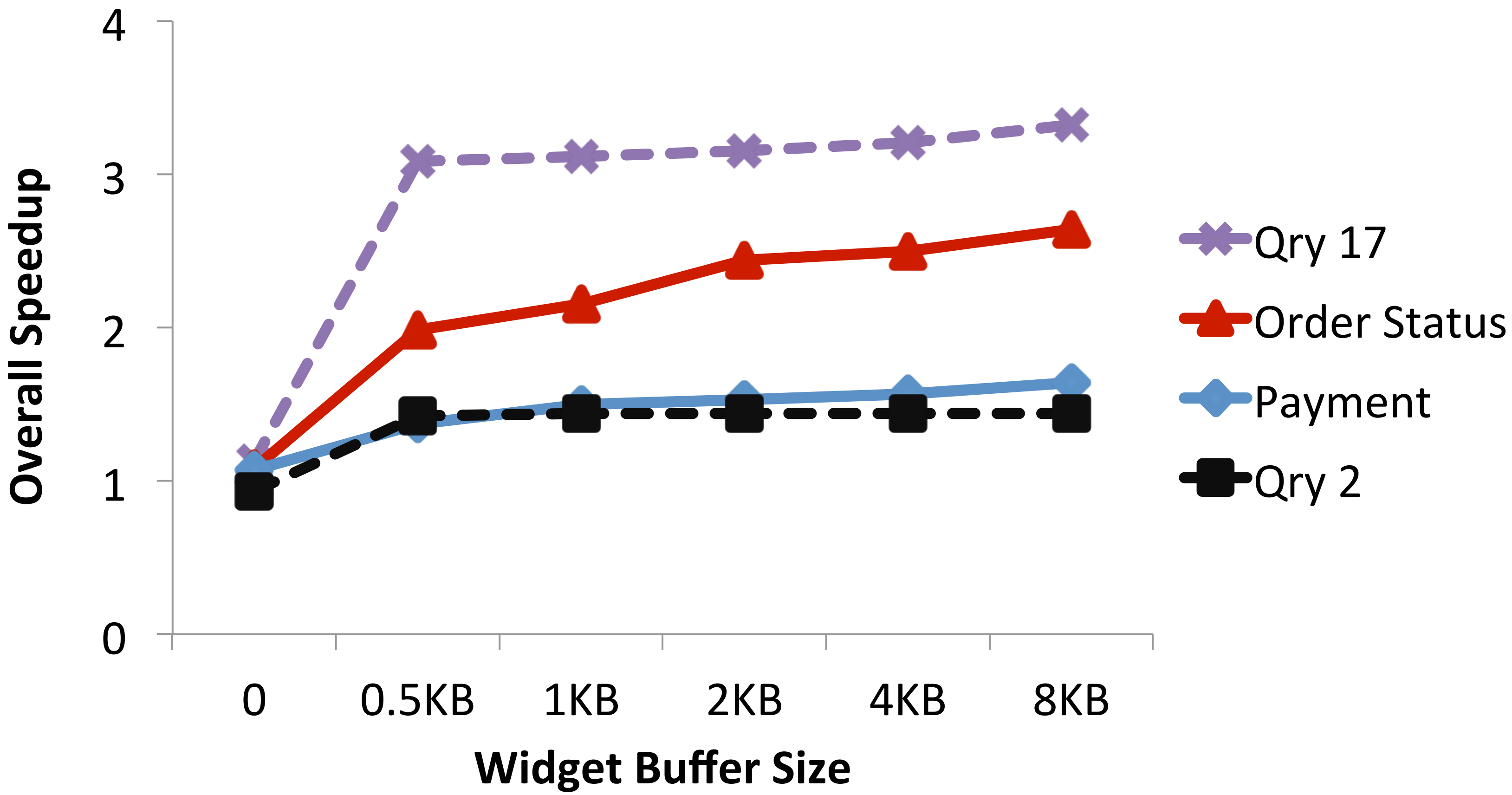
- Energy Estimations
  - Mcpat

Energy Efficiency with Indexing Widget



Up to 65% reduction in energy

Performance with Indexing Widget



Widget does not hurt performance

Conclusions

- Data explosion, dark silicon trends call for specialization
  - Rethinking of architectures to achieve efficiency
- Databases spend significant time in indexing
  - Mostly pointer chasing: general purpose CPUs are poorly suited
- Augment CPU with indexing widget
  - Dedicated offload engine: core sleeps when widget runs
  - Improves efficiency: 65% less energy, 3x faster query execution

More challenges:

Data types, data sharing, generalization…

Thanks!