Towards hardware-software co-design for data analytics

The Wisconsin Quickstep Project

Jignesh M. Patel

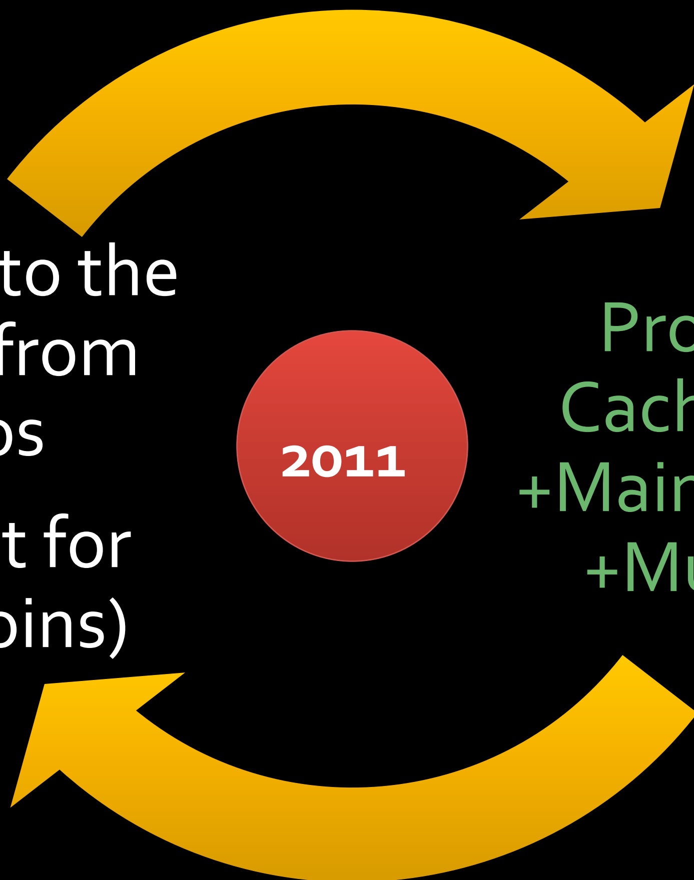
THE UNIVERSITY of WISCONSIN MADISON

Blog: http://bigfastdata.blogspot.com

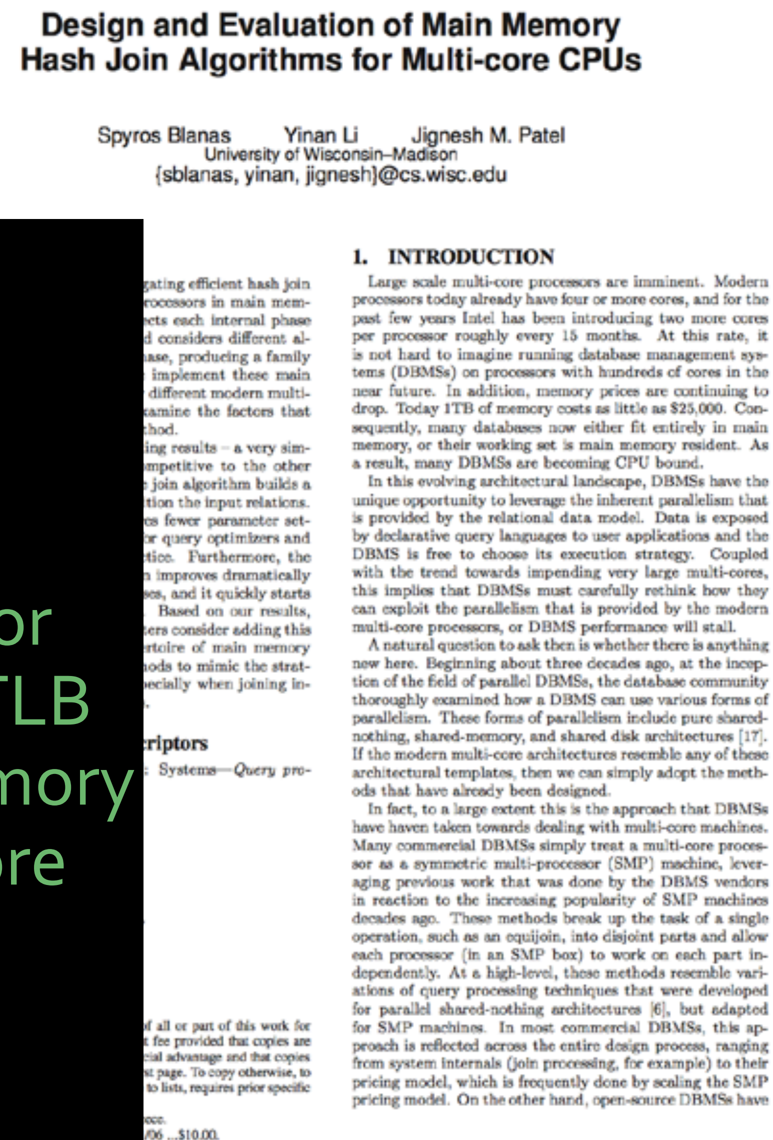
In a vicious cycle

Go back to the design from 1980s

(at least for Hash Joins)



Processor
Caches+TLB
+Main Memory
+Multicore



Design and Evaluation of Main Memory Hash Join Algorithms for Multi-core CPUs

Spyros Blanas Yinan Li Jignesh M. Patel
University of Wisconsin–Madison
{sblanas, yinan, jignesh}@cs.wisc.edu

1. INTRODUCTION

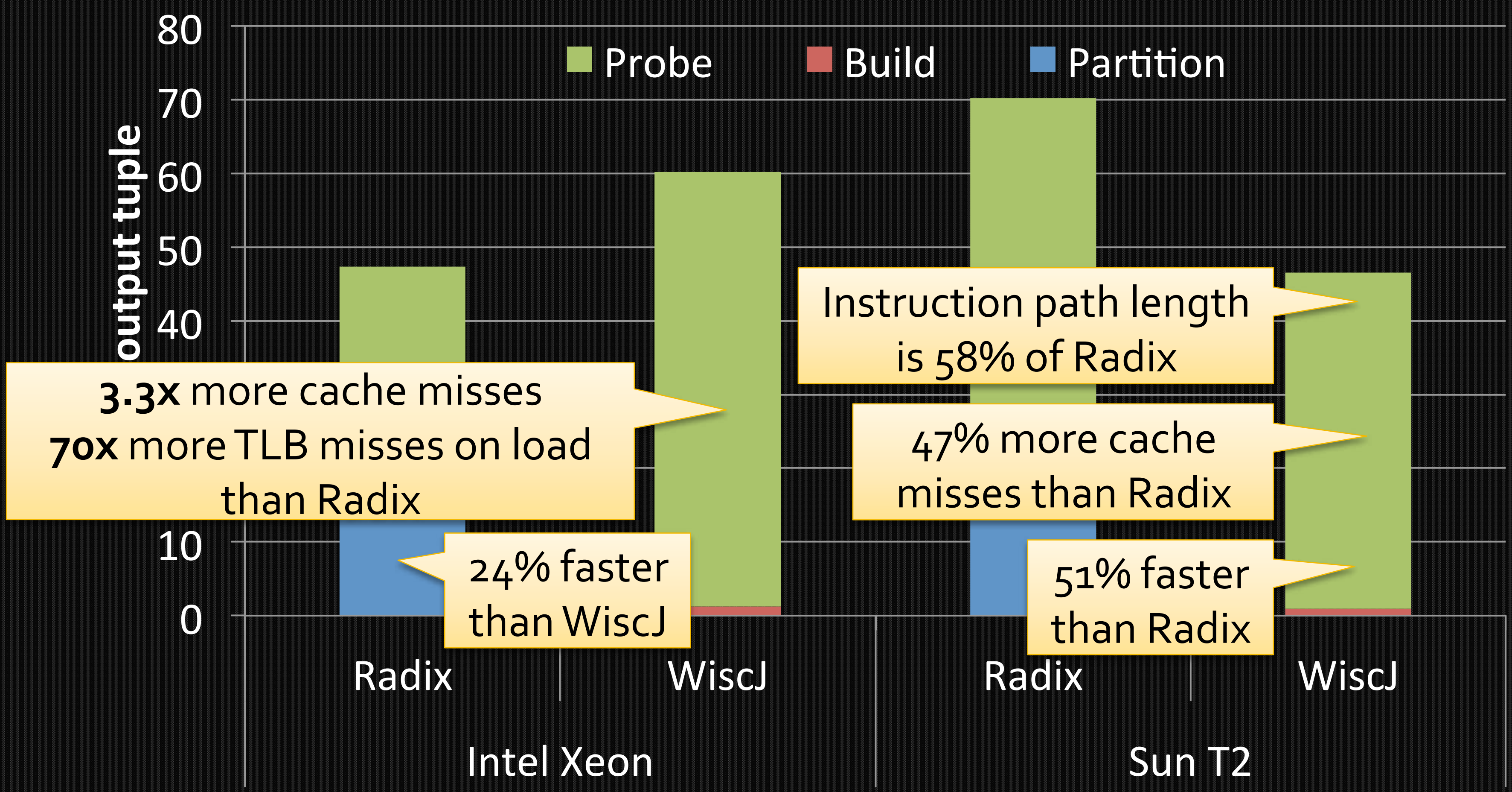
Large scale multi-core processors are imminent. Modern processors today already have four or more cores, and for the past few years Intel has been introducing two more cores per processor roughly every 15 months. At this rate, it is not hard to imagine running database management systems (DBMSs) on processors with hundreds of cores in the near future. In addition, memory prices are continuing to drop. Today 1TB of memory costs as little as $25,000. Consequently, many databases now either fit entirely in main memory, or their working set is main memory resident. As a result, many DBMSs are becoming CPU bound.

In this evolving architectural landscape, DBMSs have the unique opportunity to leverage the inherent parallelism that is provided by the relational data model. Data is exposed by declarative query languages to user applications and the DBMS is free to choose its execution strategy. Coupled with the trend towards impending very large multi-cores, this implies that DBMSs must carefully rethink how they can exploit the parallelism that is provided by the modern multi-core processors, or DBMS performance will stall.

A natural question to ask then is whether there is anything new here. Beginning about three decades ago, at the inception of the field of parallel DBMSs, the database community thoroughly examined how a DBMS can use various forms of parallelism. These forms of parallelism include pure shared-nothing, shared-memory, and shared disk architectures [17]. If the modern multi-core architectures resemble any of these architectural templates, then we can simply adopt the methods that have already been designed.

In fact, to a large extent this is the approach that DBMSs have haven taken towards dealing with multi-core machines. Many commercial DBMSs simply treat a multi-core processor as a symmetric multi-processor (SMP) machine, leveraging previous work that was done by the DBMS vendors in reaction to the increasing popularity of SMP machines decades ago. These methods break up the task of a single operation, such as an equijoin, into disjoint parts and allow each processor (in an SMP box) to work on each part independently. At a high-level, these methods resemble variations of query processing techniques that were developed for parallel shared-nothing architectures [6], but adapted for SMP machines. In most commercial DBMSs, this approach is reflected across the entire design process, ranging from system internals (join processing, for example) to their pricing model, which is frequently done by scaling the SMP pricing model. On the other hand, open-source DBMSs have

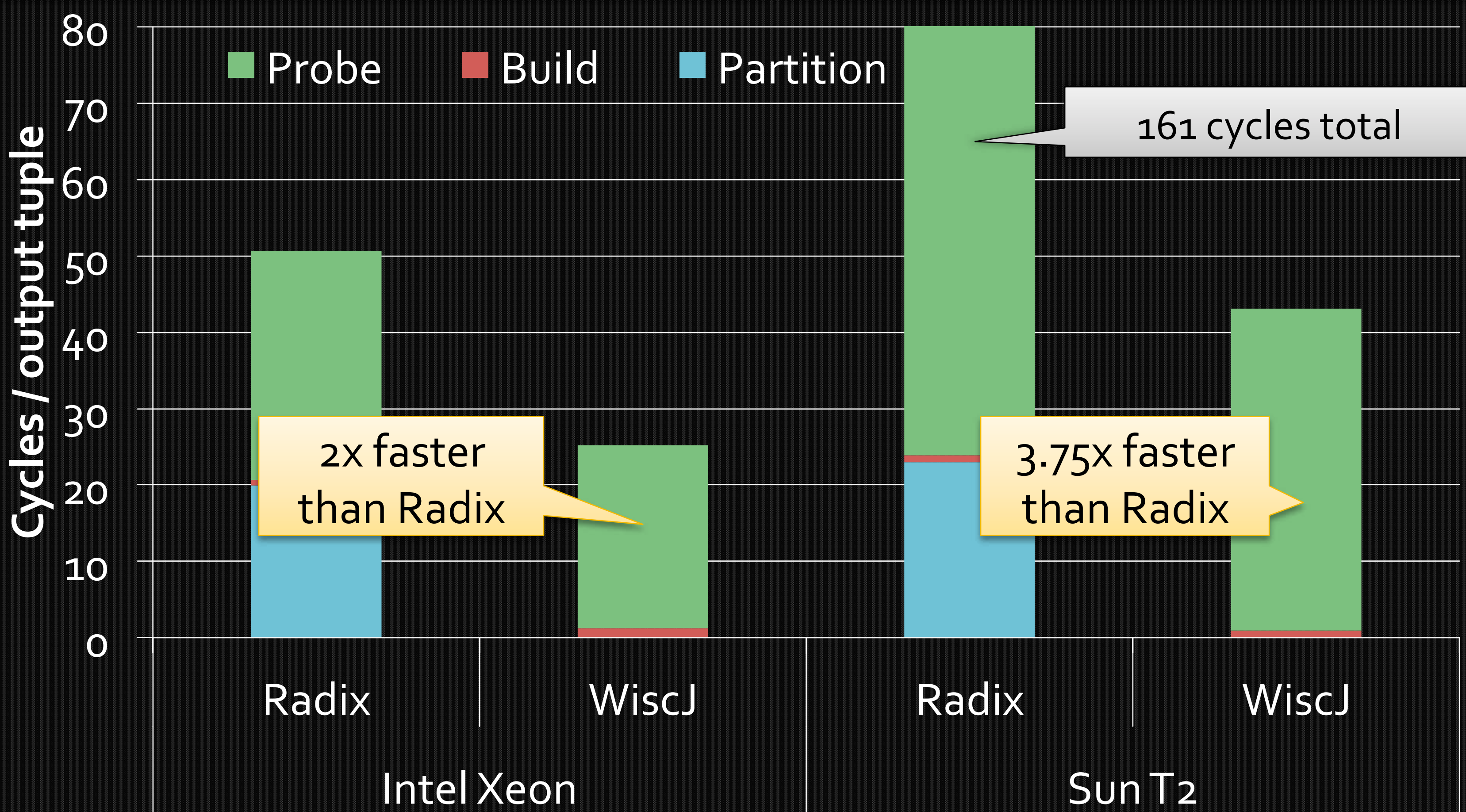
Results – uniform dataset



Results – high skew dataset

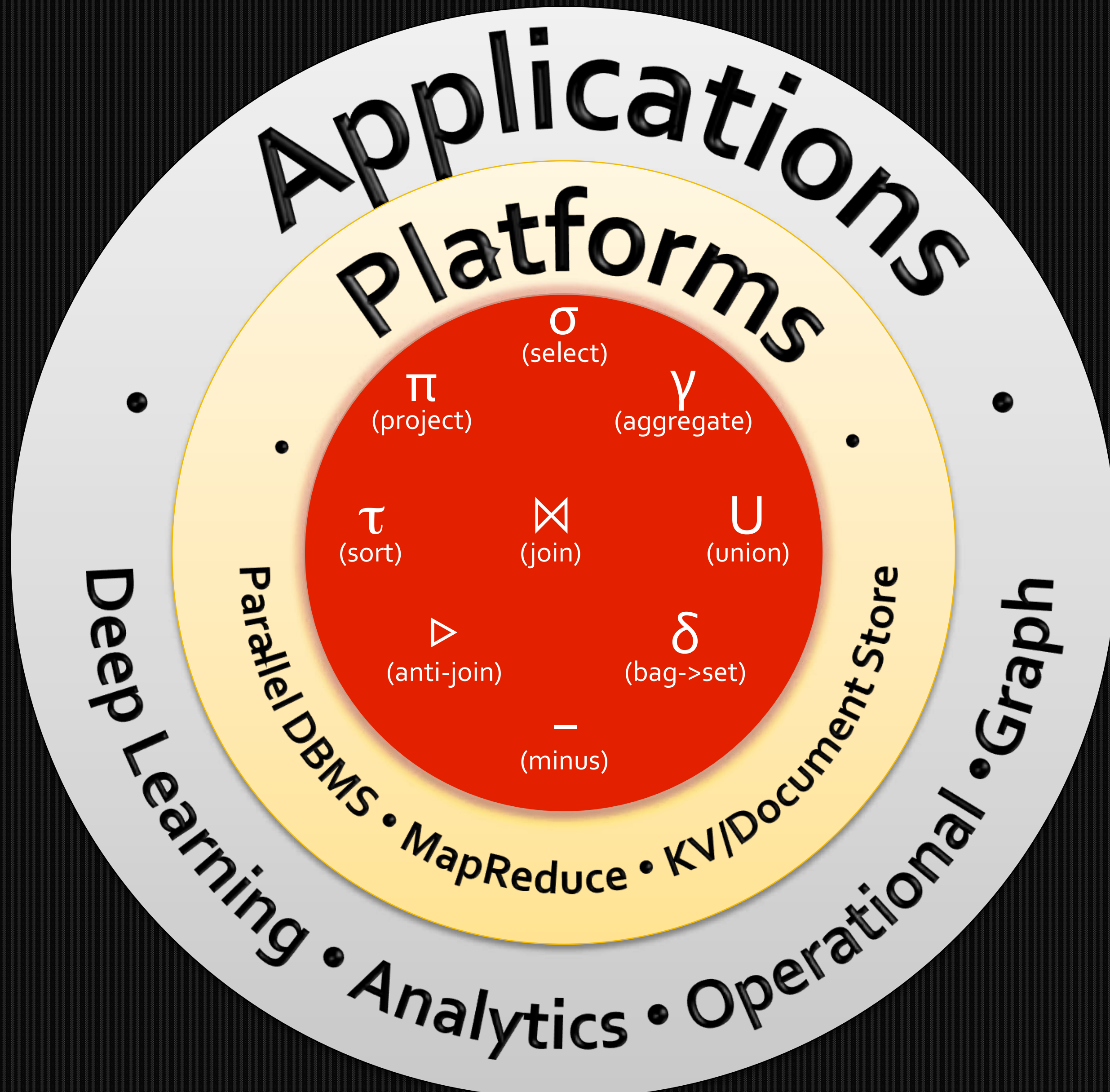
Skew in partitioning-based hash join algorithms causes partition size skew → work imbalance

Non-partitioned (Wisconsin) hash join improves with higher skew!

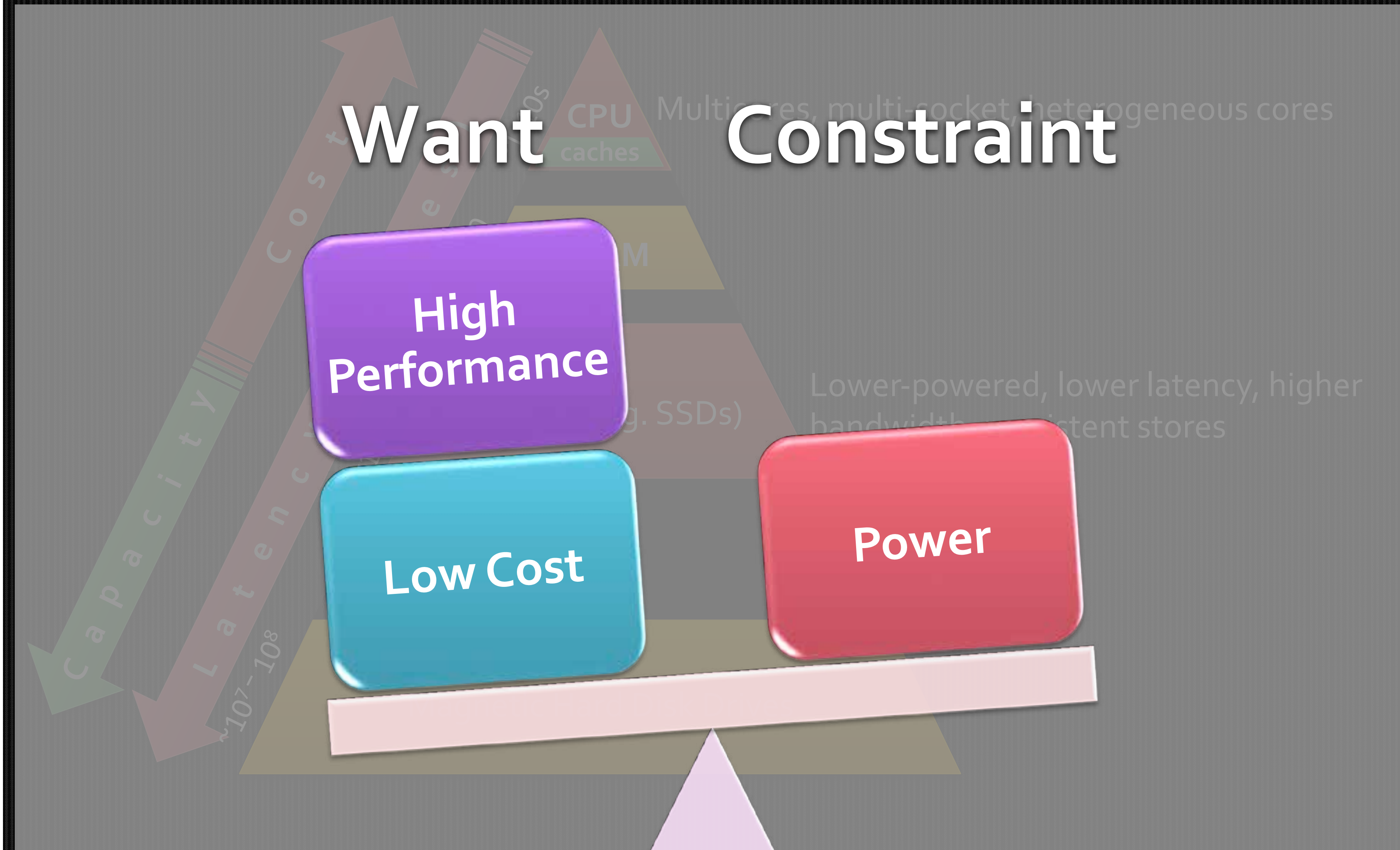


Hash Joins (2011): Summary

- Hash join algorithm started simple, and with each architectural turn, it adapted.
- We have come full circle: The simple hash join is now very competitive. And, in many cases more efficient than the more complex methods!


Applications
Platforms
σ
(select)
π
(project)
γ
(aggregate)
τ
(sort)
⋈
(join)
∪
(union)
▷
(anti-join)
δ
(bag->set)
−
(minus)
Parallel DBMS • MapReduce • KV/Document Store
Deep Learning • Analytics • Operational • Graph

Disruptive hardware trends



Quickstep

http://quickstep.cs.wisc.edu

Goal	• Run data analytics @ hardware speeds
Short-term	• Run @ the speed of hardware today
Long-term	• Hardware-software co-design for data kernels

Li and Patel, SIGMOD'13

Scan: A Key Data Processing Kernel

What?

- Scan a column of a table applying some predicate

Why?

- A key primitive in database
- "The" critical kernel in main memory analytic systems

How?

- Conserve memory bandwidth: **BitWeaving** the data
- Use every bit of data that is brought to the processer efficiently using **intra-cycle parallelism**

Li and Patel, SIGMOD'13

Focus on Column Scan (can be generalized)

Traditional Row Store

shipdate	...	discount	quantity
Mar-12-2013		5%	5
Jan-08-2013		2%	4
Apr-29-2013		10%	3
May-14-2013		0%	6
...	...	...	...
Feb-28-2013		5%	0

One big file

Column Store

shipdate	...	discount	quantity
Mar-12-2013		5%	5
Jan-08-2013		2%	4
Apr-29-2013		10%	3
May-14-2013		0%	6
...		...	...
Feb-28-2013		5%	0
File: 1		File: n-1	File: n

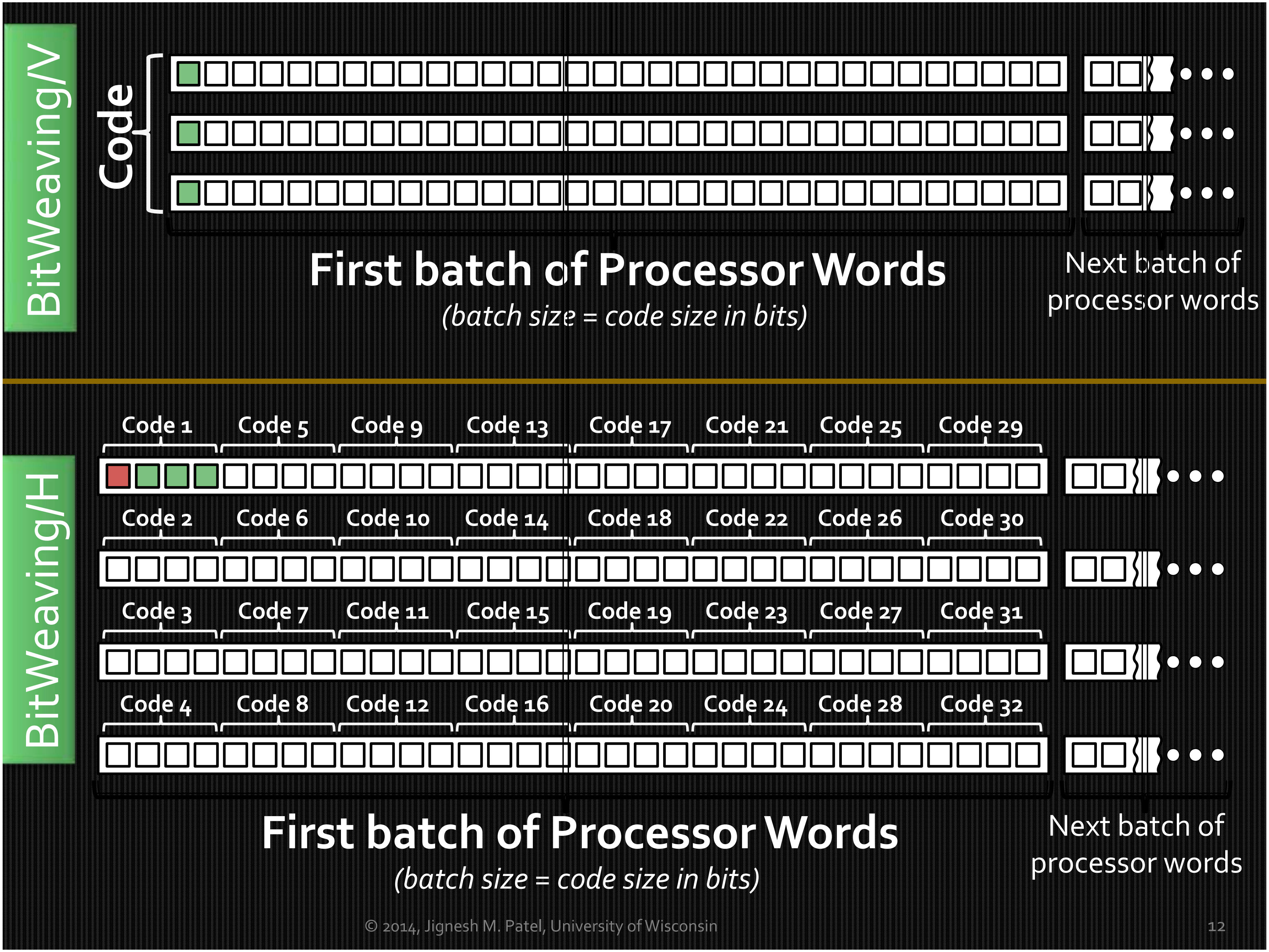
16 bits

Order-preserving compression

Column Codes:

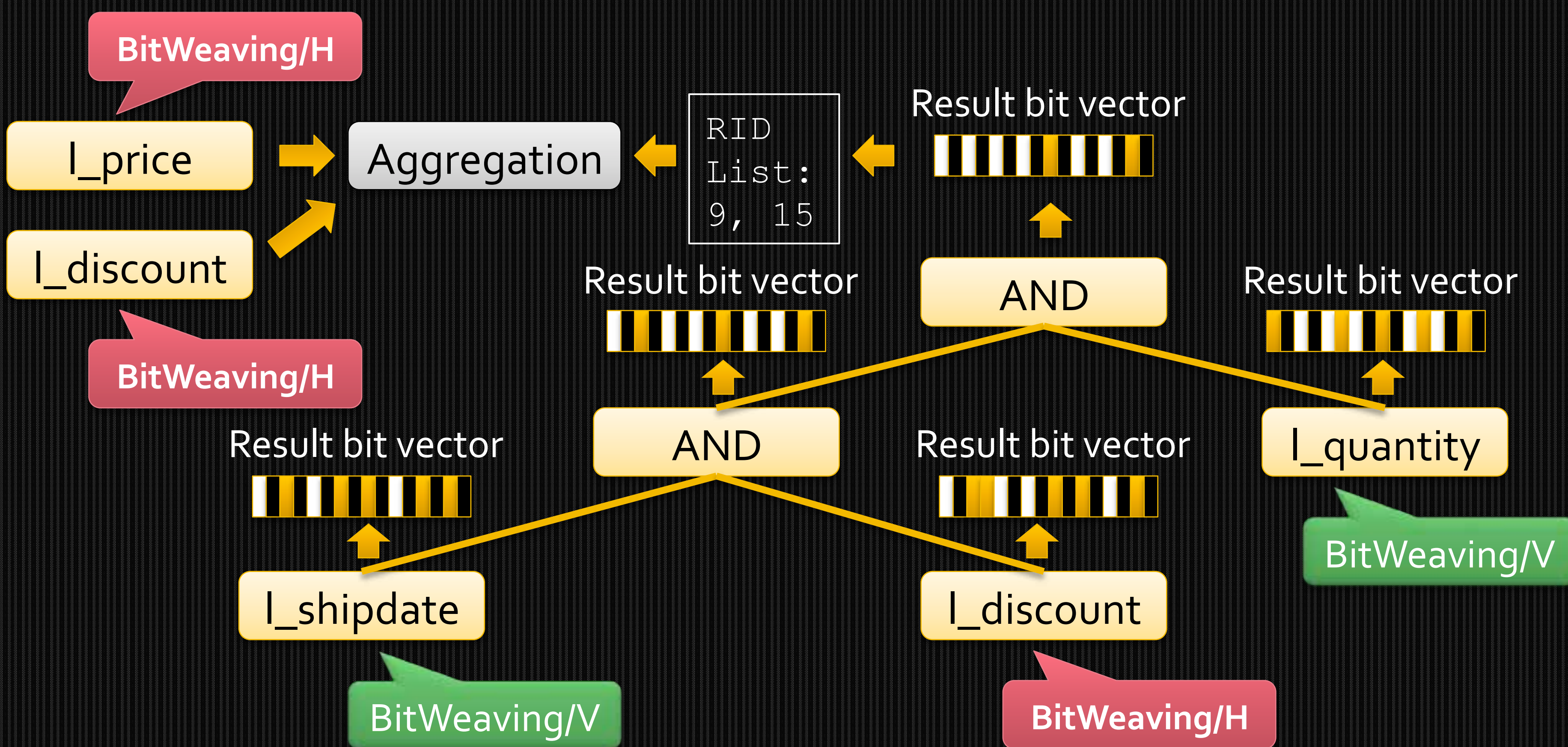
5	4	3	6	2	7	1	0	...

3 bits

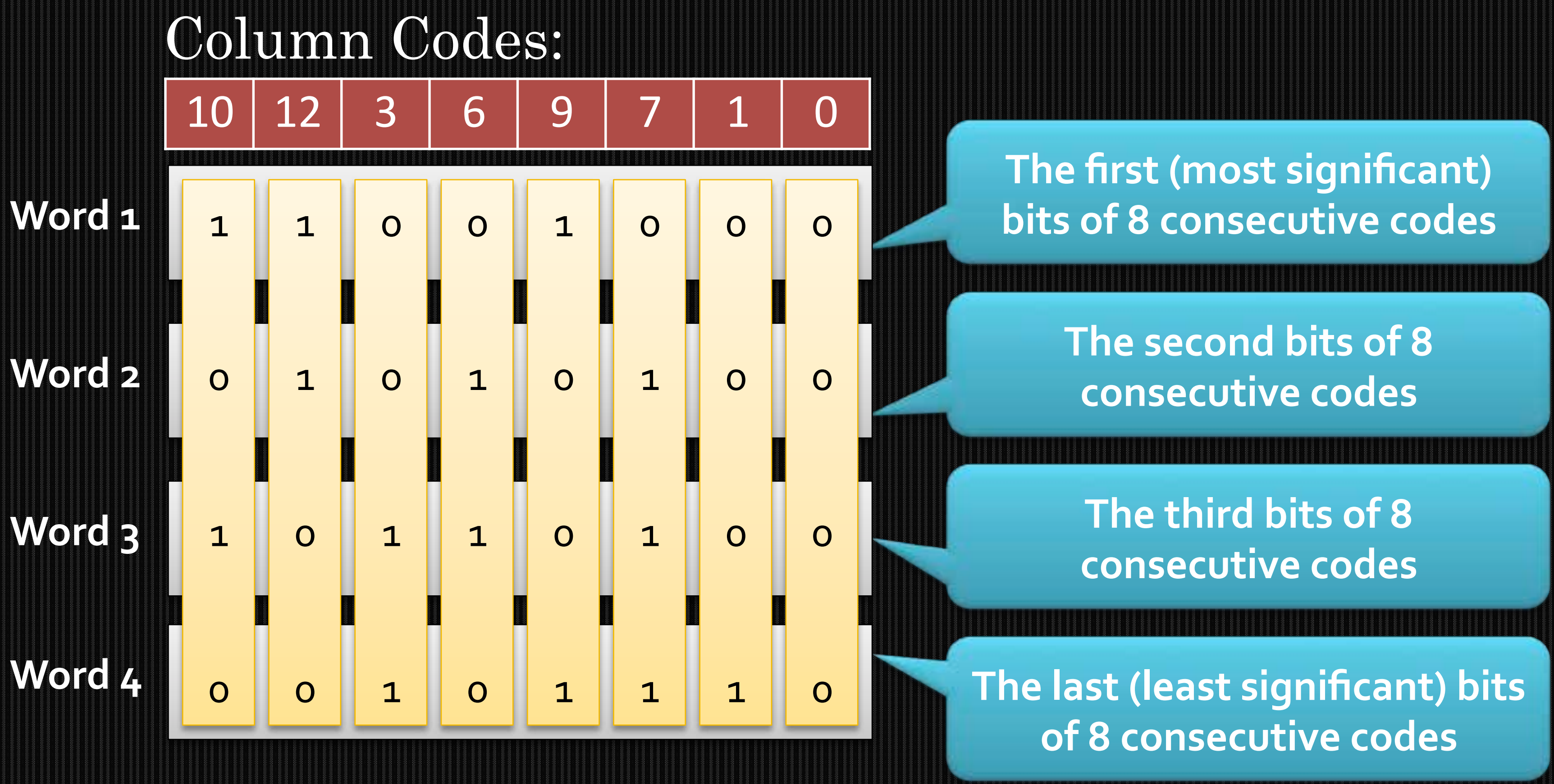

BitWeaving/V
Code
First batch of Processor Words
(batch size = code size in bits)
Next batch of processor words
BitWeaving/H
Code 1 Code 5 Code 9 Code 13 Code 17 Code 21 Code 25 Code 29
Code 2 Code 6 Code 10 Code 14 Code 18 Code 22 Code 26 Code 30
Code 3 Code 7 Code 11 Code 15 Code 19 Code 23 Code 27 Code 31
Code 4 Code 8 Code 12 Code 16 Code 20 Code 24 Code 28 Code 32
First batch of Processor Words
(batch size = code size in bits)
Next batch of processor words
© 2014, Jignesh M. Patel, University of Wisconsin
12

Framework – Example

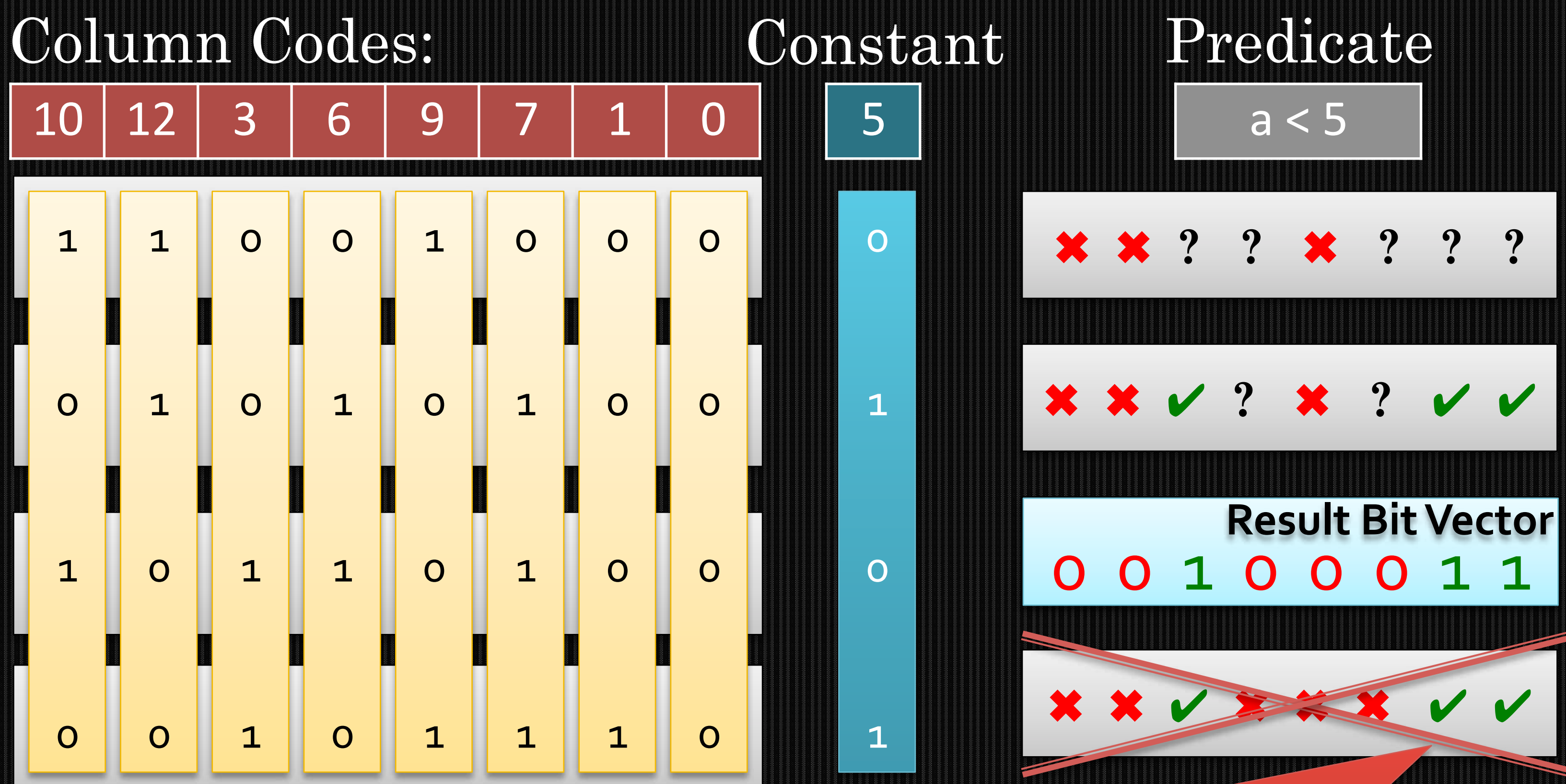
```
SELECT SUM(l_discount * l_price) FROM lineitem
WHERE l_shipdate BETWEEN Date AND Date + 1 year
  AND l_discount BETWEEN Discount – 0.01 AND Discount + 0.01
  AND l_quantity < Quantity
```



BitWeaving/V



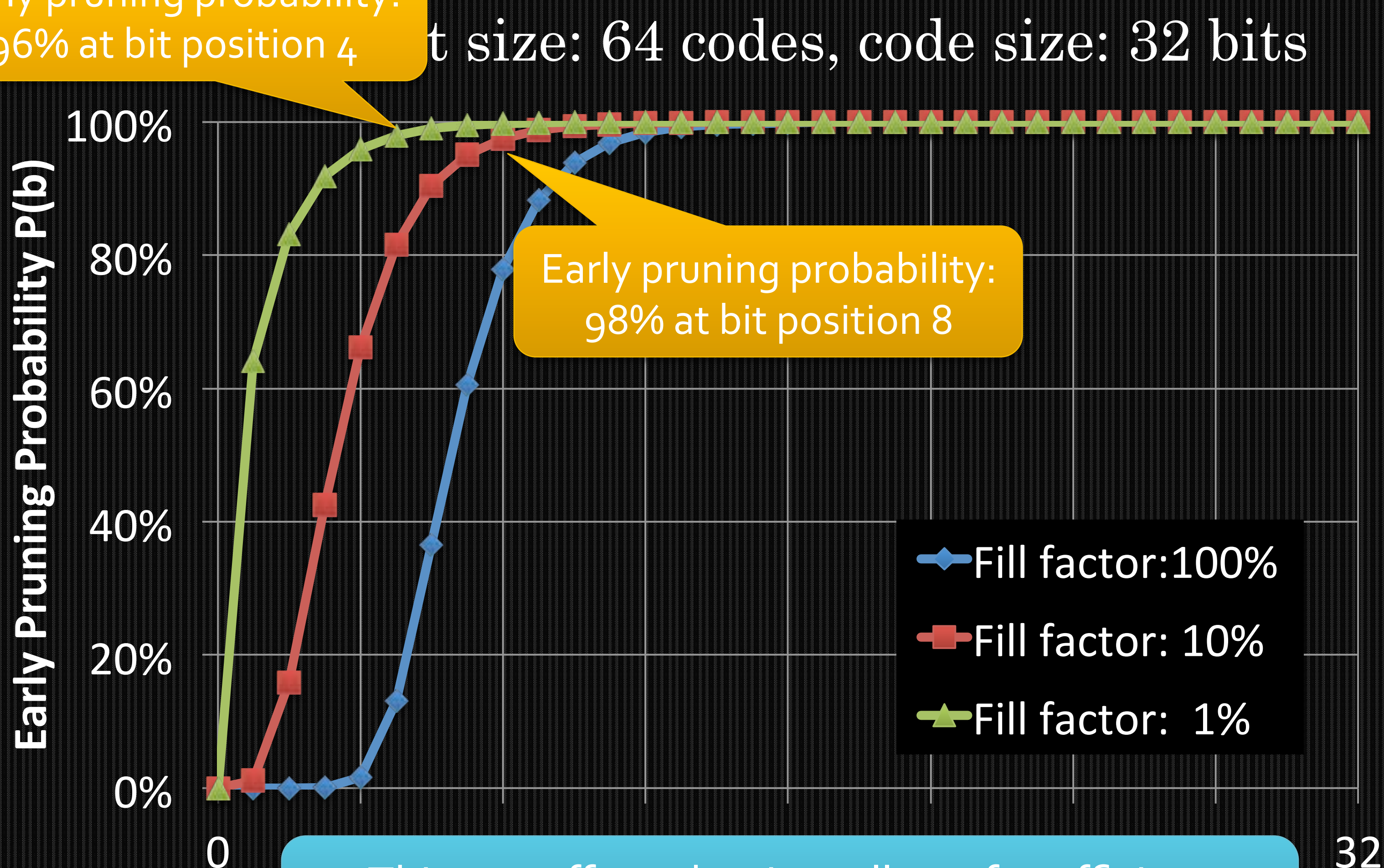
BitWeaving/V - early pruning



Early Pruning: terminate the predicate evaluation on a segment, when all results have been determined.

BitWeaving/V - Early Pruning Model

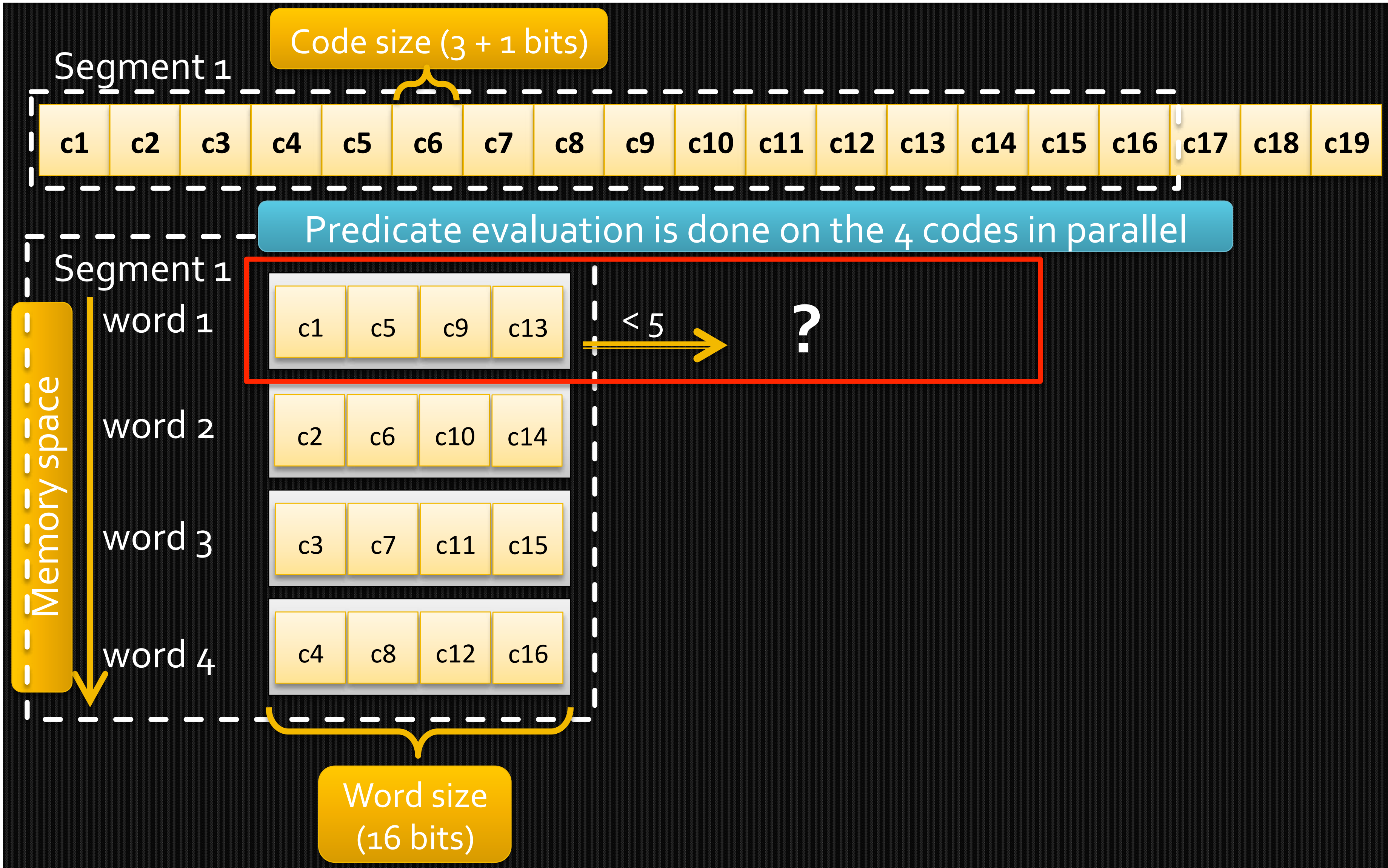
Early pruning probability: 96% at bit position 4



Early pruning probability: 98% at bit position 8

This cut-off mechanism allows for efficient evaluation of conjunction/disjunction predicates

BitWeaving/H - Example



BitWeaving/H: Less Than Predicate

Uses only 3 instructions! Without the delimiter, we would need ~12 instructions…

	c5=7	c9=6	c13=2
0001	0111	0110	0010
0101	0101	0101	0101
1000	0000	0000	1000

$$X = (c_1 c_5 c_9 c_{13})$$

$$Y = (5555)$$

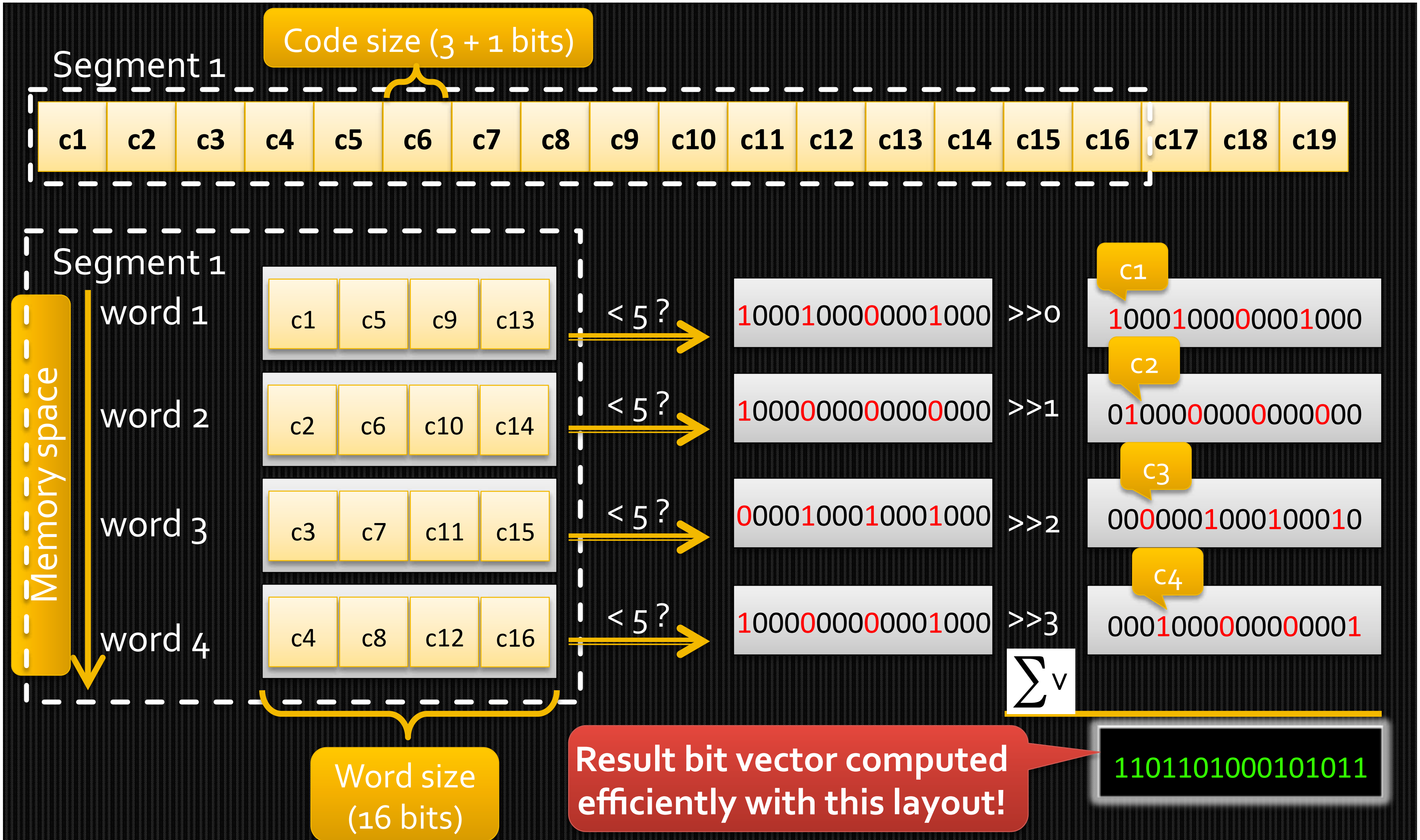
$$(Y + (X \oplus M1)) \wedge M2$$

$$M1 = 0111\ 0111\ 0111\ 0111$$

$$M2 = 1000\ 1000\ 1000\ 1000$$

Works for arbitrary code sizes & word sizes!

BitWeaving/H - Example



Evaluation

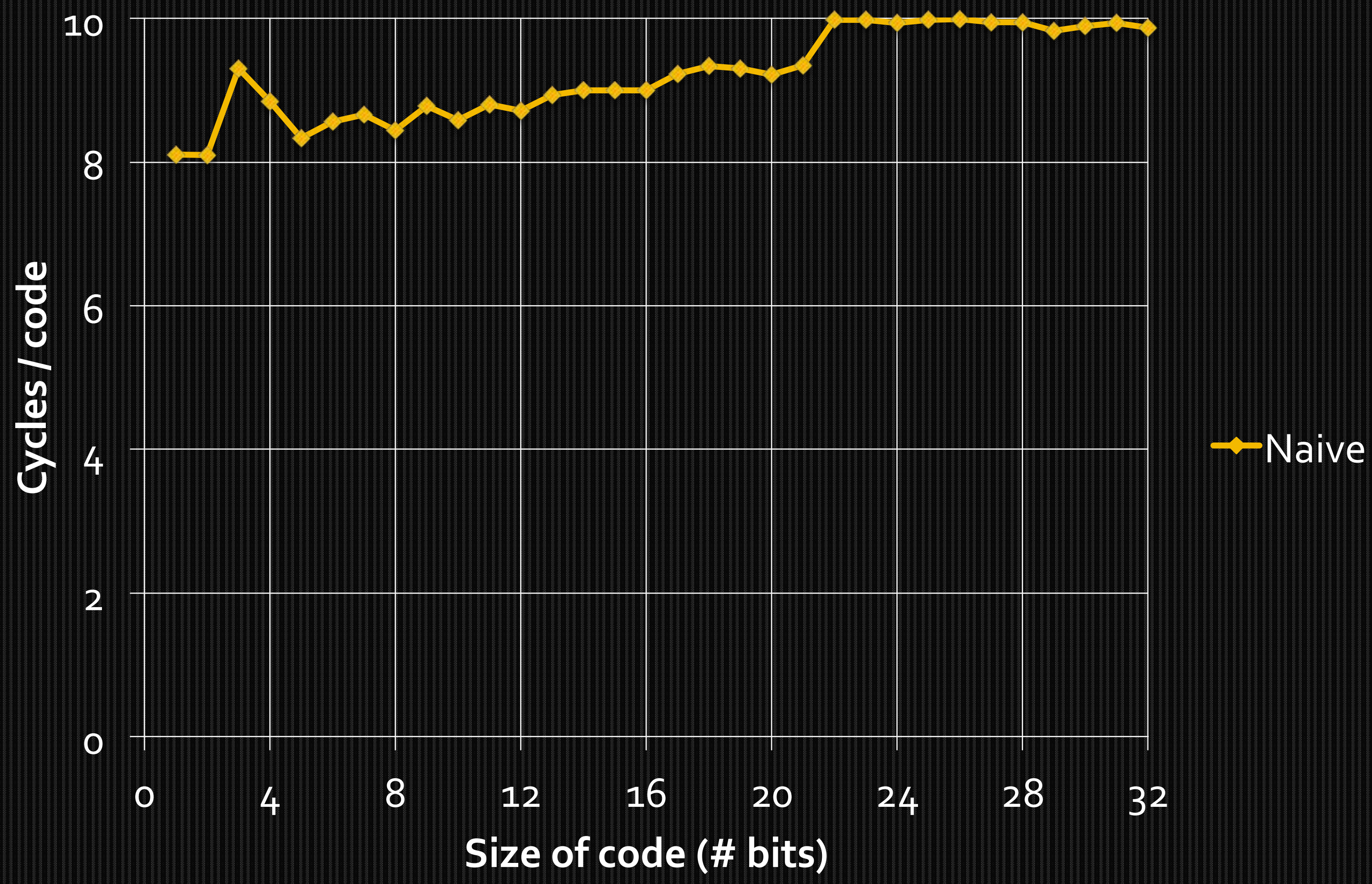
SYSTEM

- Intel Xeon X5650
  - 64 bits ALU
  - 128 bits SIMD
  - 12MB L3 Cache
- 24GB memory
- Single threaded execution

WORKLOAD

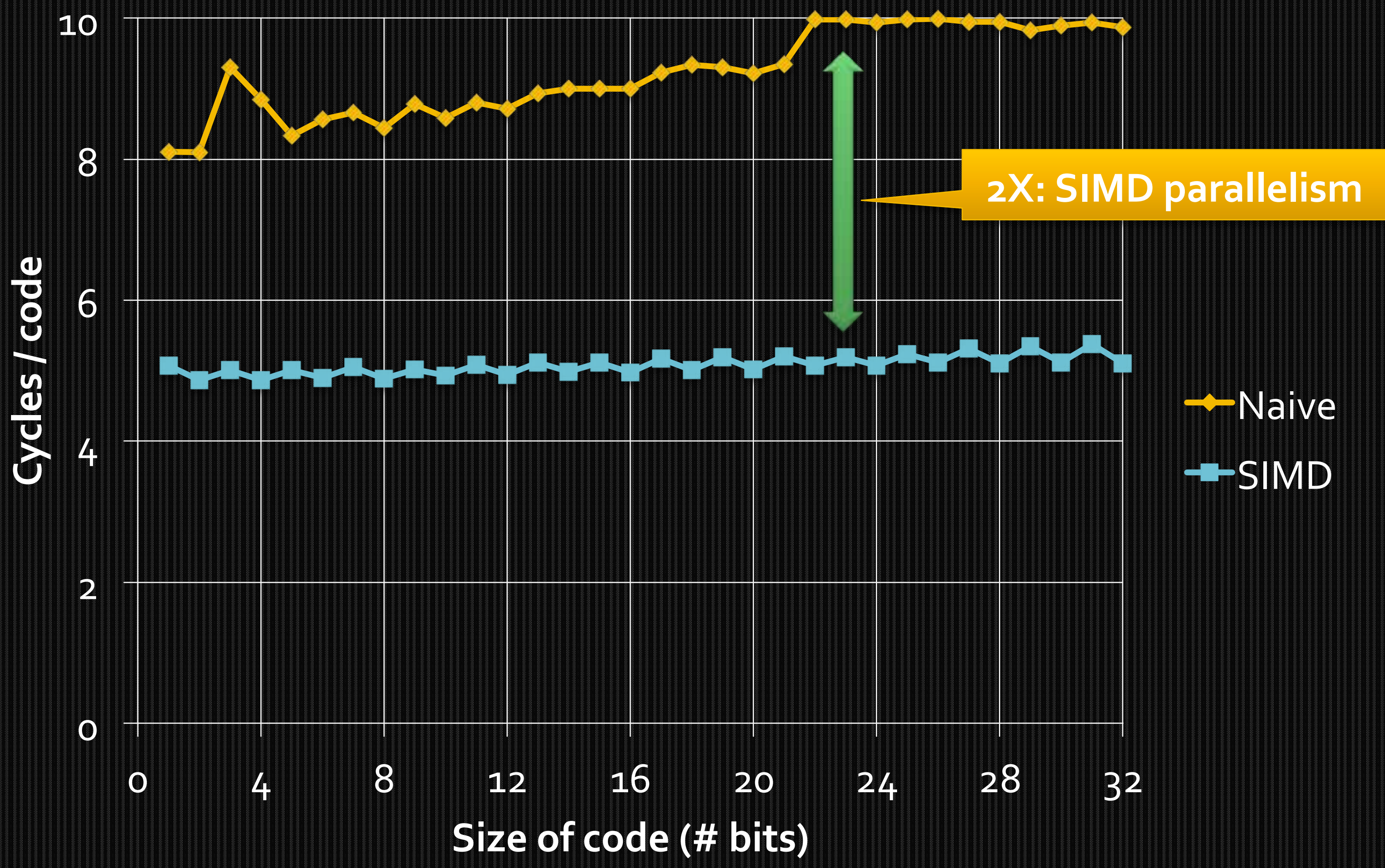
1. Synthetic
   - ```
     SELECT COUNT(*)
     FROM R
     WHERE R.a < C
     ```
   - 1 billion tuples
   - Uniform distribution
   - Selectivity: 10%
2. TPC-H @ SF=10
   - scan only with materialized join results
     ```

Evaluation: Micro-benchmark



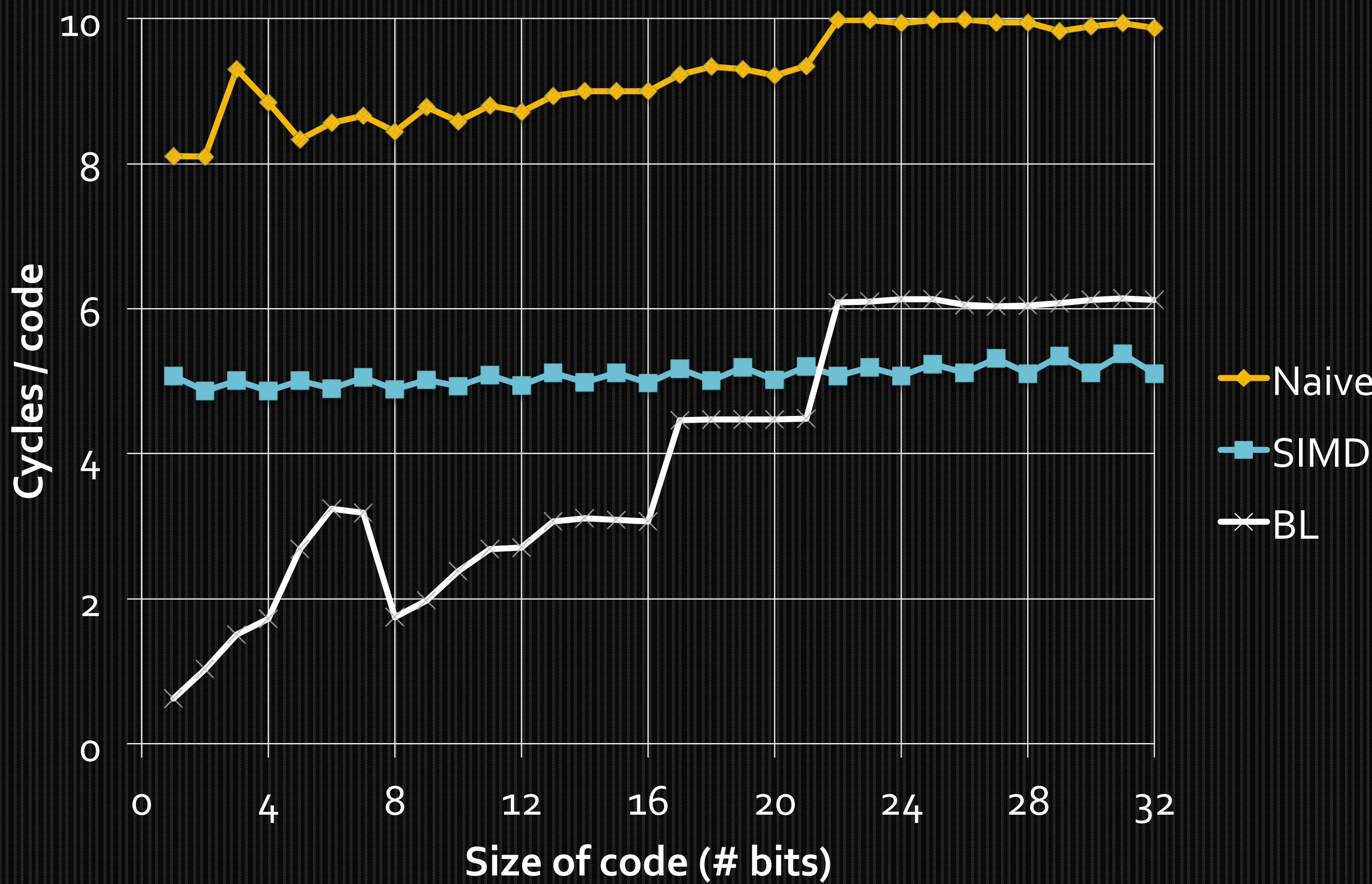
SIMD Paper: T. Willhalm, N. Popovici, Y. Boshmaf, H. Plattner, A. Zeier, and J. Schaffner.
SIMD-Scan: Ultra fast in-memory table scan using on-chip vector processing units. PVLDB'09

Evaluation: Micro-benchmark

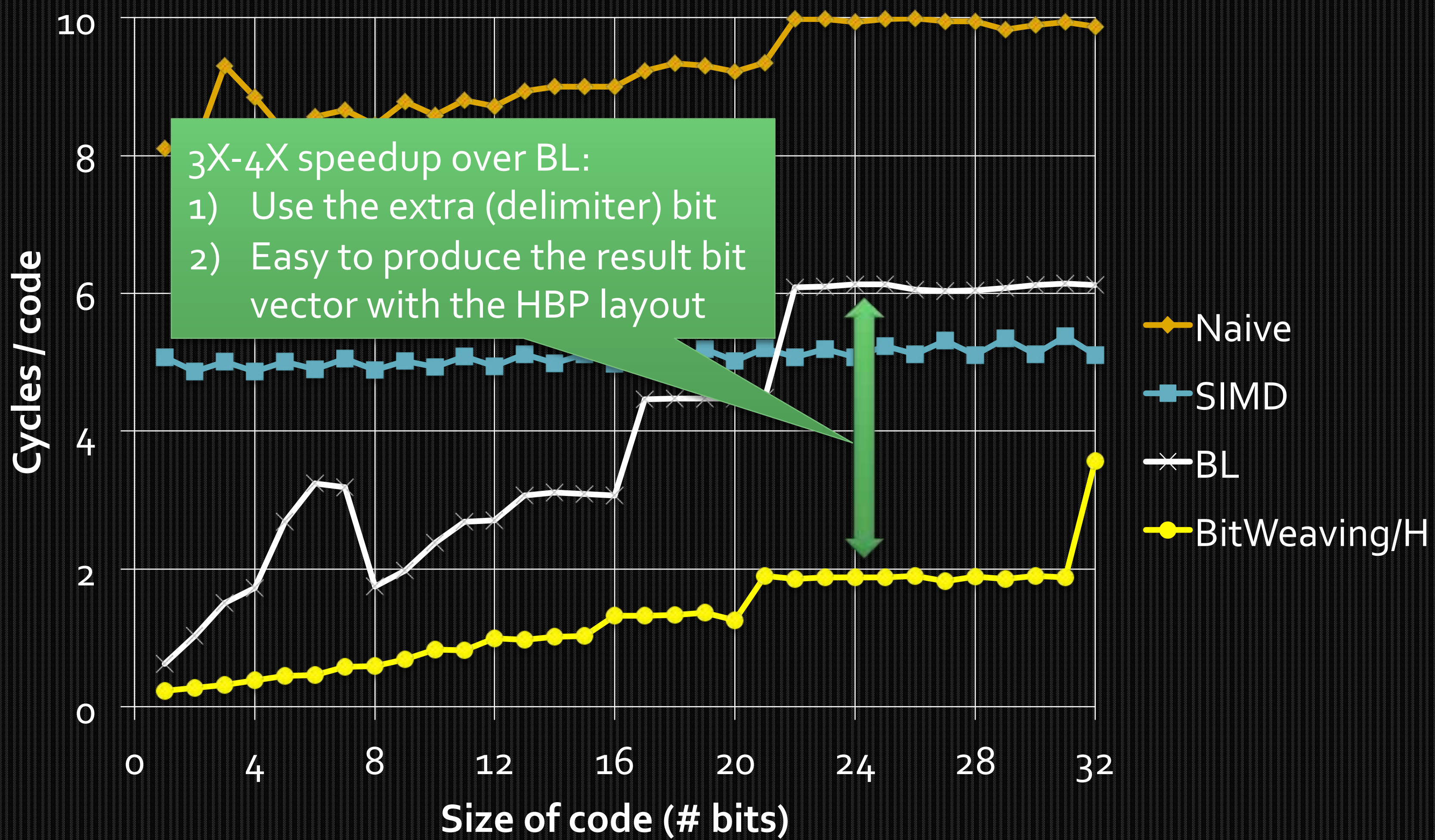


Blink Paper: R. Johnson, V. Raman, R. Sidle, and G. Swart. Row-wise parallel predicate evaluation. VLDB'08

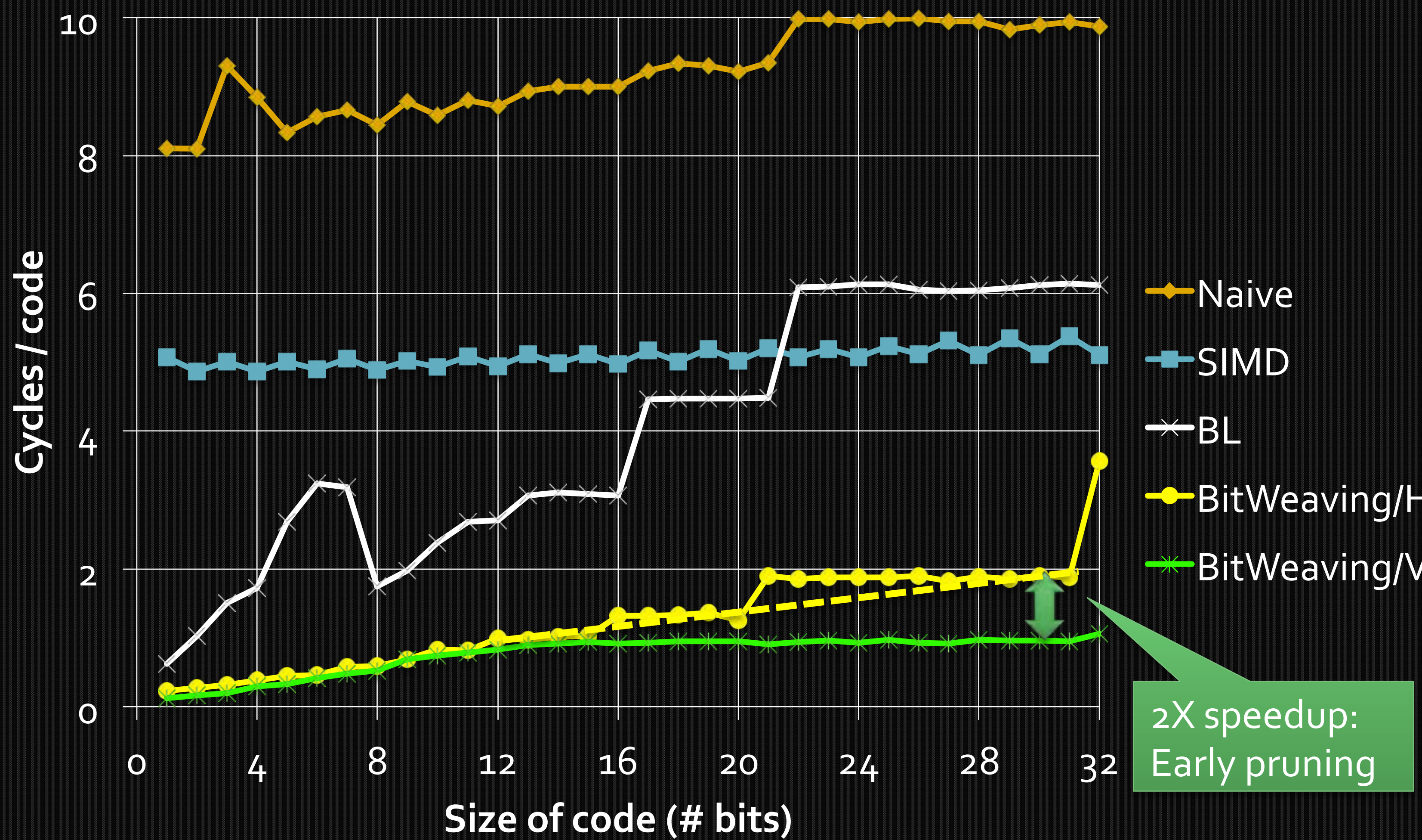
Evaluation: Micro-benchmark



Evaluation: Micro-benchmark



Evaluation: Micro-benchmark



Many more experiments in the paper

WideTable

Customer

cid	cname	gender	address
1	Andy	M	100 Main st.
2	Kate	F	20 10th blvd.
3	Bob	M	300 5th ave.

Product

pid	pname
1	Milk
2	Coffee
3	Tea

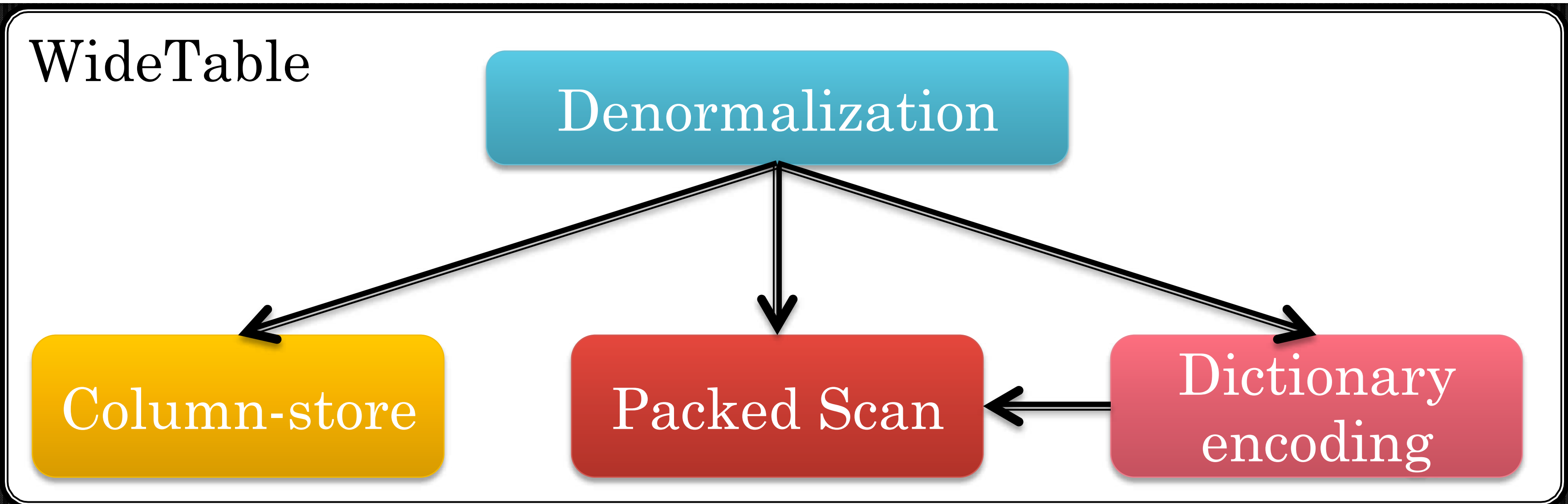
Buy

cid	pid	status
1	2	S
2	2	F
3	3	S
1	2	S

cid	cname	gender	address	pid	pname	status
1	Andy	M	100 Main st.	2	Coffee	S
2	Kate	F	20 10th blvd.	2	Coffee	F
3	Bob	M	300 5th ave.	3	Tea	S
1	Andy	M	100 Main st.	2	Coffee	S
NULL	NULL	NULL	NULL	1	Milk	NULL

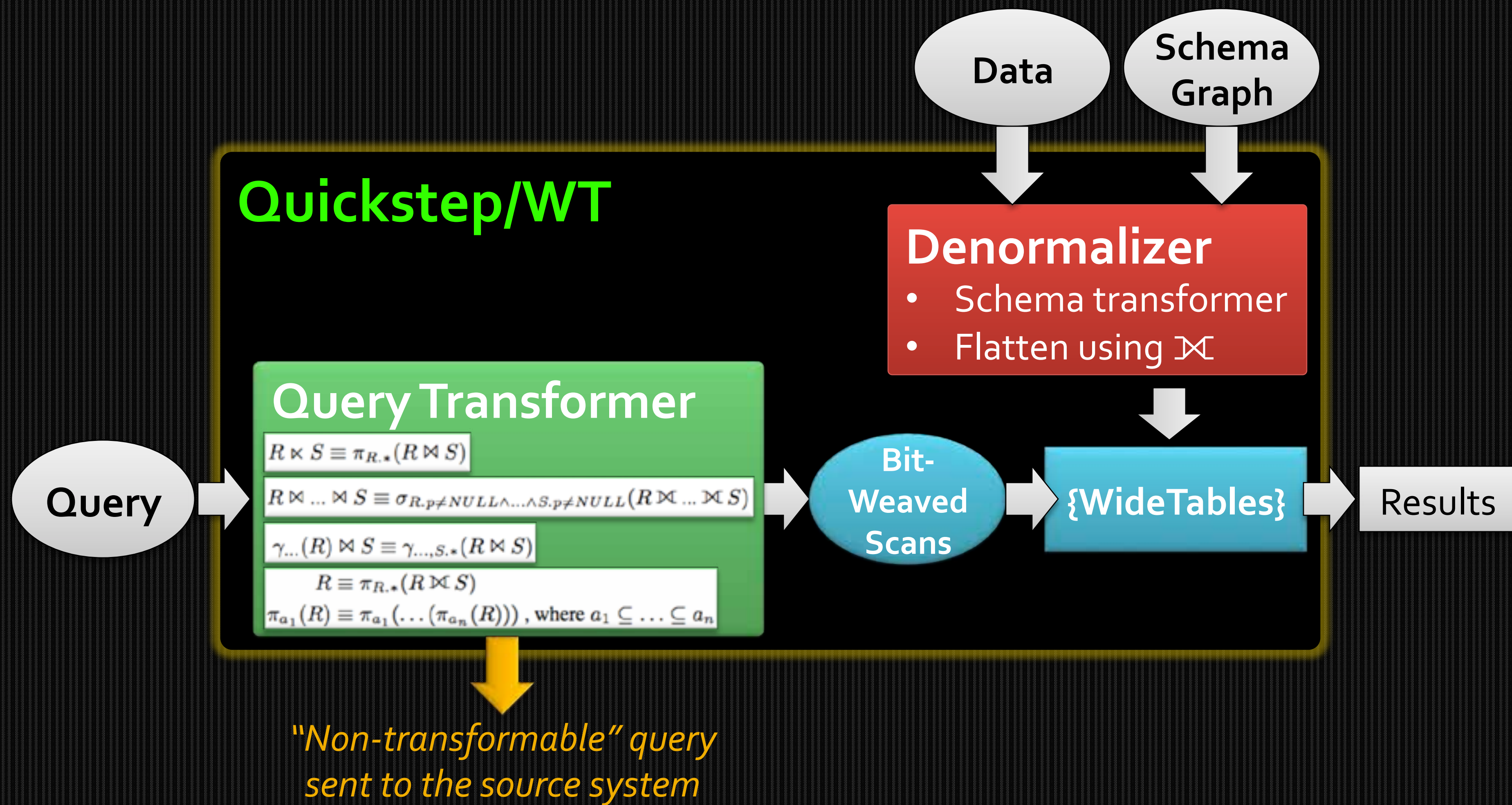
WideTable

WideTable

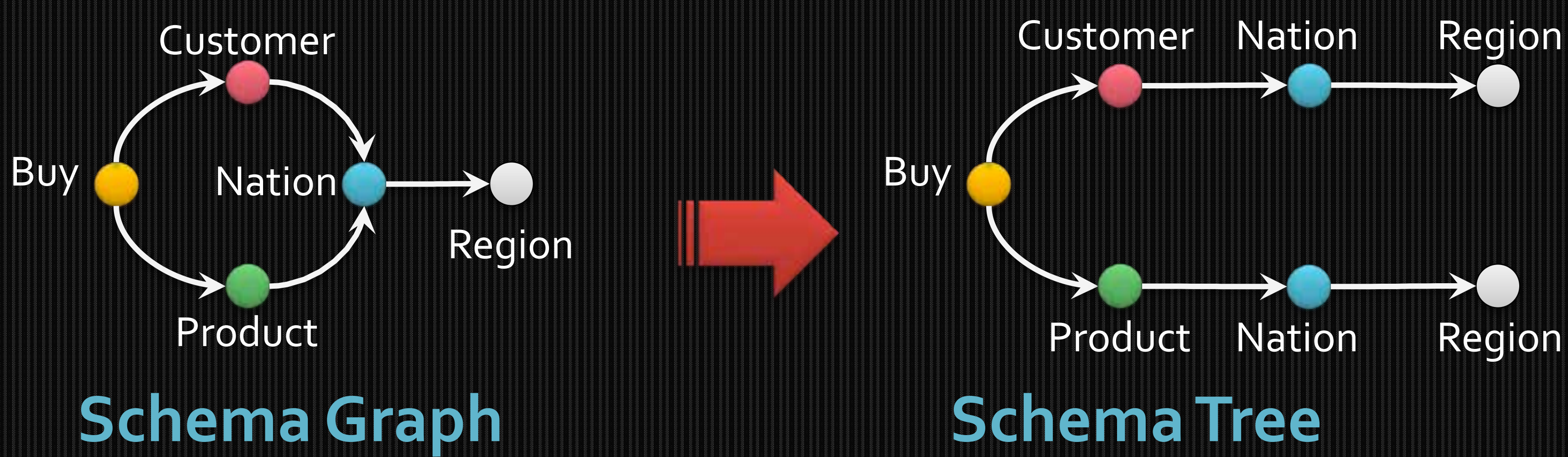


Now we can run analytical workloads (e.g. TPC-H) using simple BitWeaved scans

WideTable Techniques



Schema Graph



WideTable = (Region $\bowtie$ Nation $\bowtie$ Customer) $\bowtie$ (Region $\bowtie$ Nation $\bowtie$ Product $\bowtie$ Buy)

SMW = {WideTables}
e.g. for TPC-H, SMW={lineItemWT, ordersWT, partsuppWT, customerWT}

TPC-H Queries

TPC-H Queries	Joins	Nested Queries	Non-FK joins	WideTable
Q1, Q6				LineitemWT
Q3, Q5, Q7-Q10, Q12, Q14, Q19	×			LineitemWT
Q4, Q15, Q17, Q18, Q20	×	×		LineitemWT
Q21	×	×	×	---
Q2, Q11, Q16	×	×		PartsuppWT
Q13	×			OrdersWT
Q22	×	×		OrdersWT

Evaluation

SYSTEM

- Intel Xeon E5-2620 × 2
- 2.0 GHz
- 12 cores / 24 threads
- 15MB L3 Cache
- 32G, 1600MHz DDR3

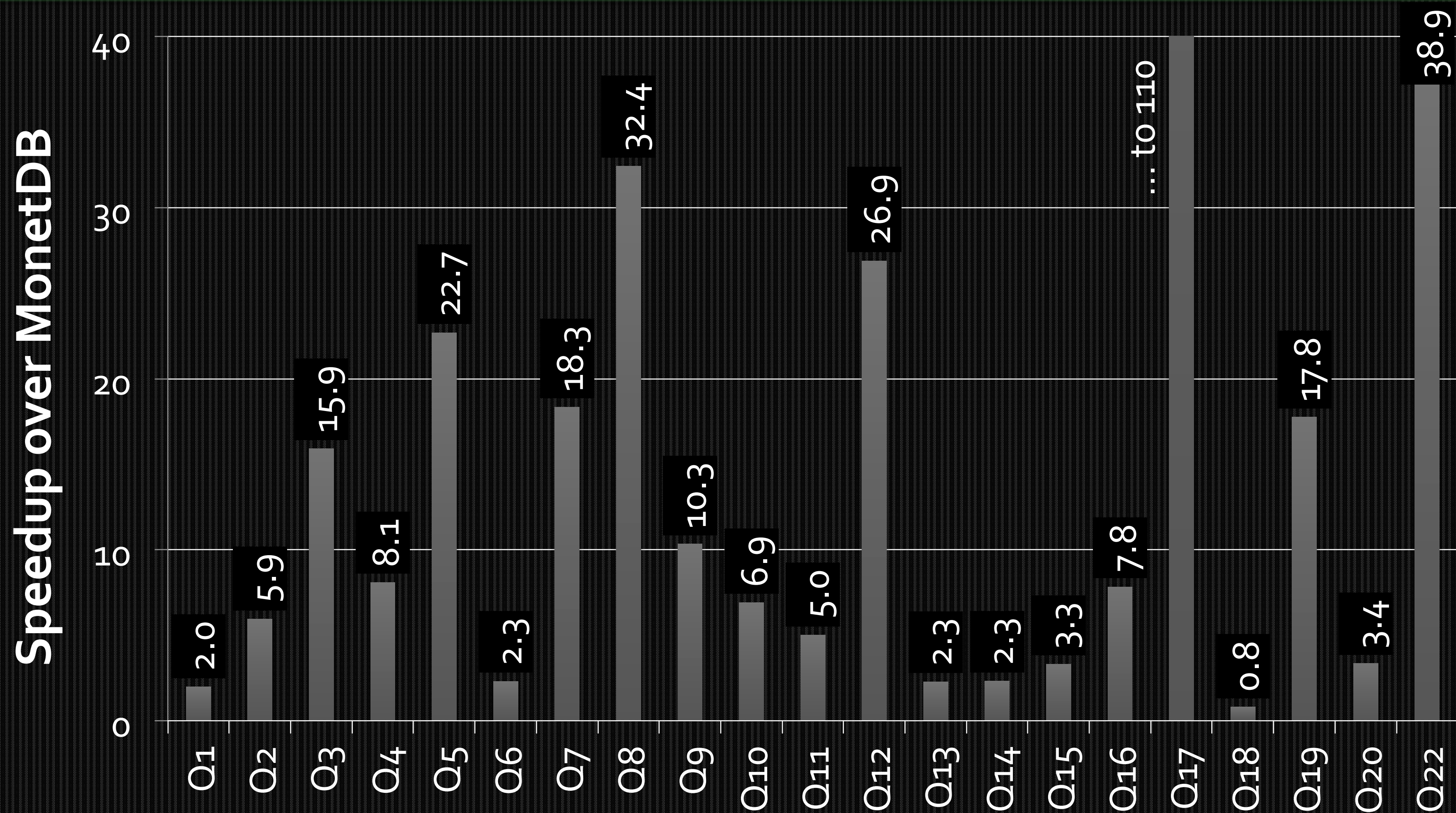
BENCHMARK

- SF: 10 (~10GB)
- SMW =

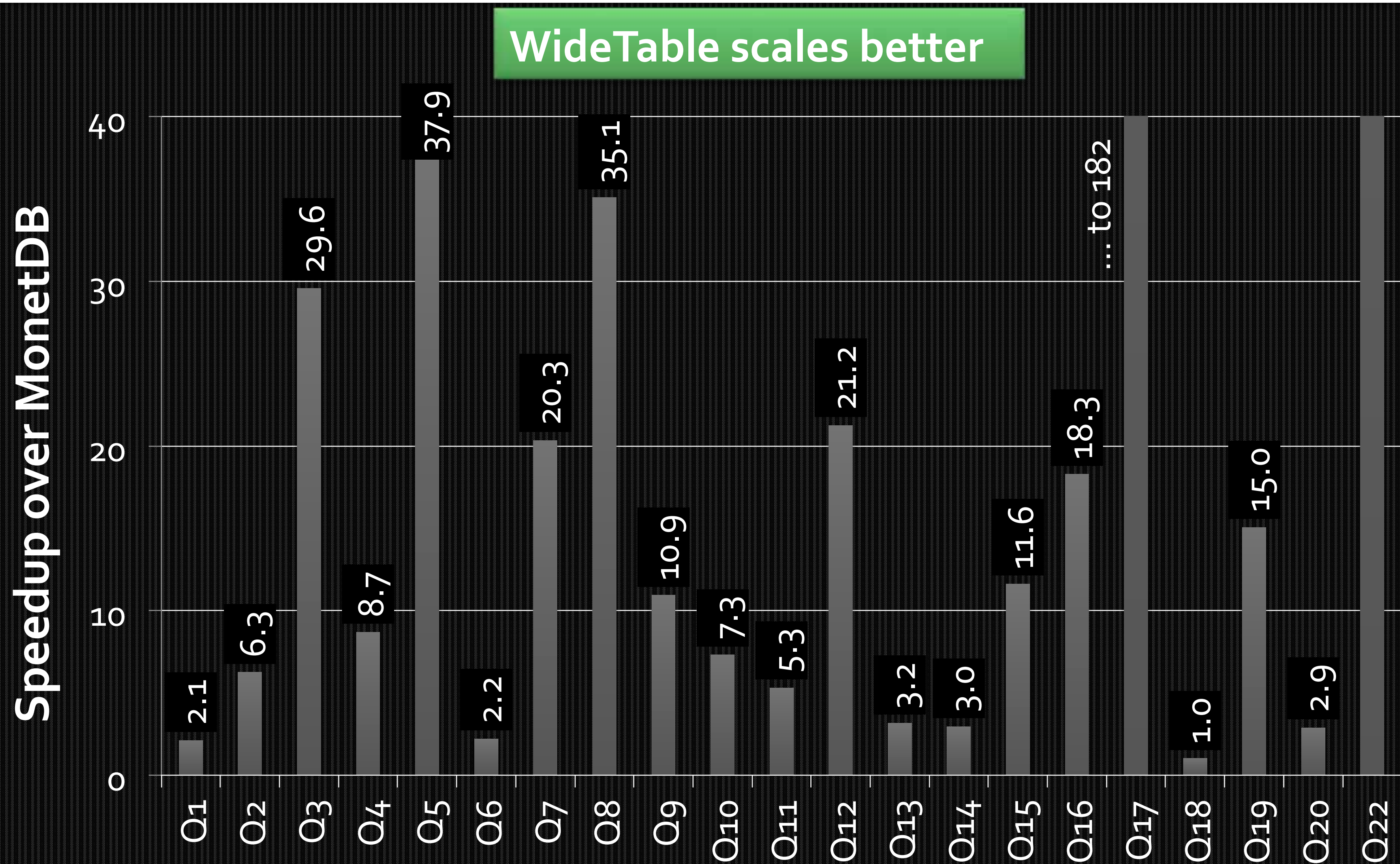
lineItemWT	5.4 GB
ordersWT	0.7 GB
partsuppWT	0.2 GB
customerWT	0.05 GB
dictionaries	0.8 GB
filter columns	1.3 GB
TOTAL	8.5GB

Speedup over MonetDB: Single Thread

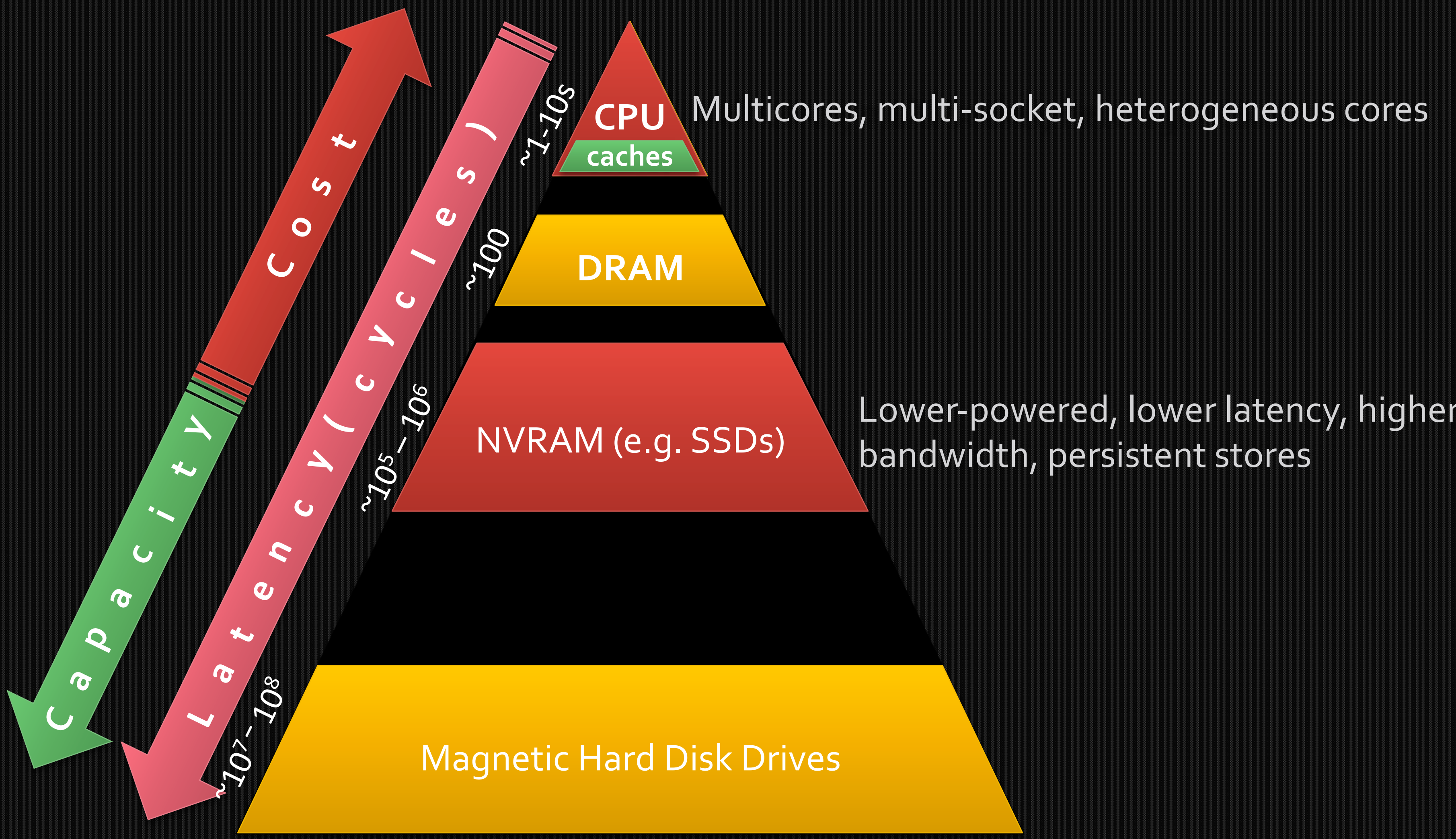
WideTable over 10X faster than MonetDB for about half of the 21 queries



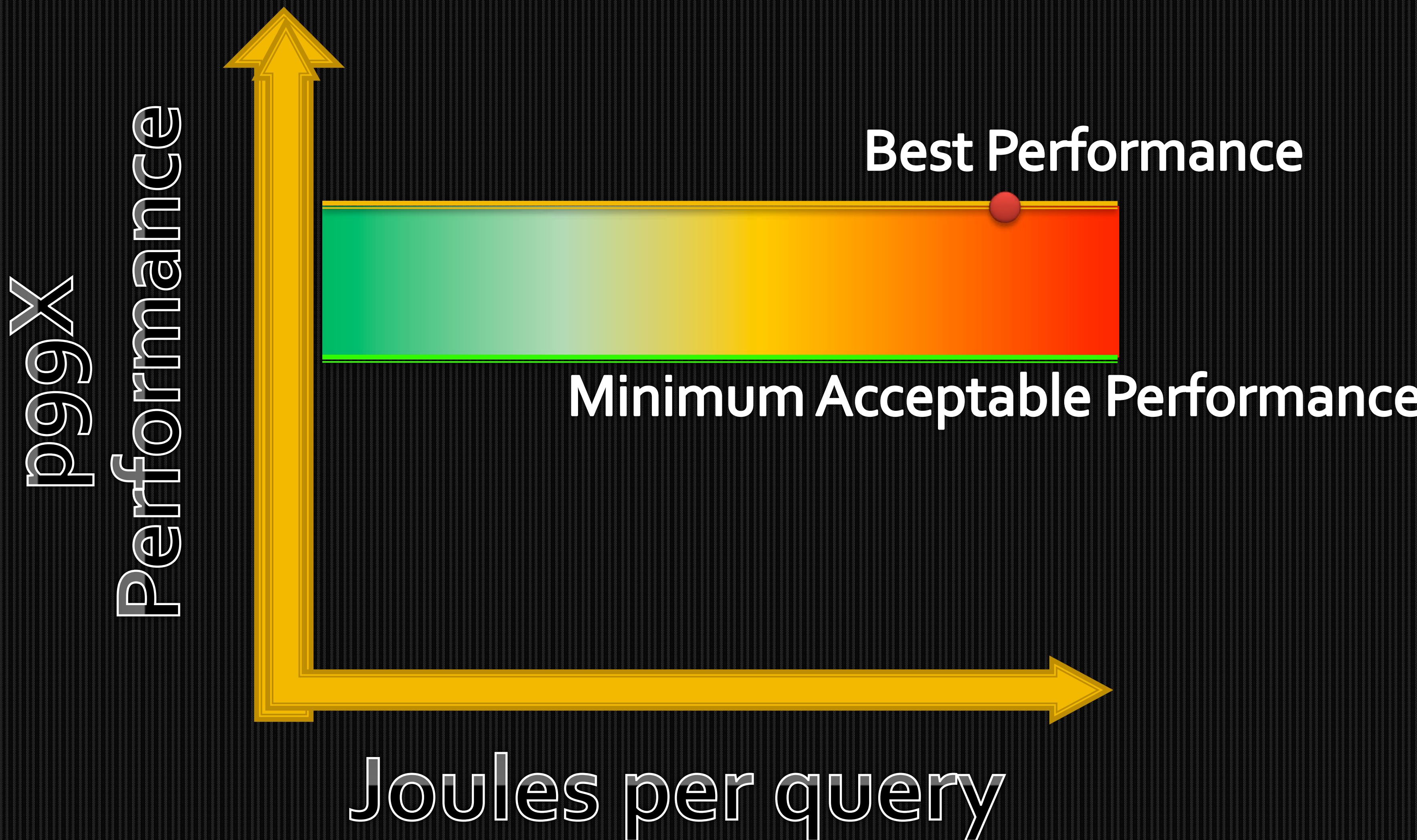
Speedup over MonetDB: 12 Threads



Disruptive hardware trends



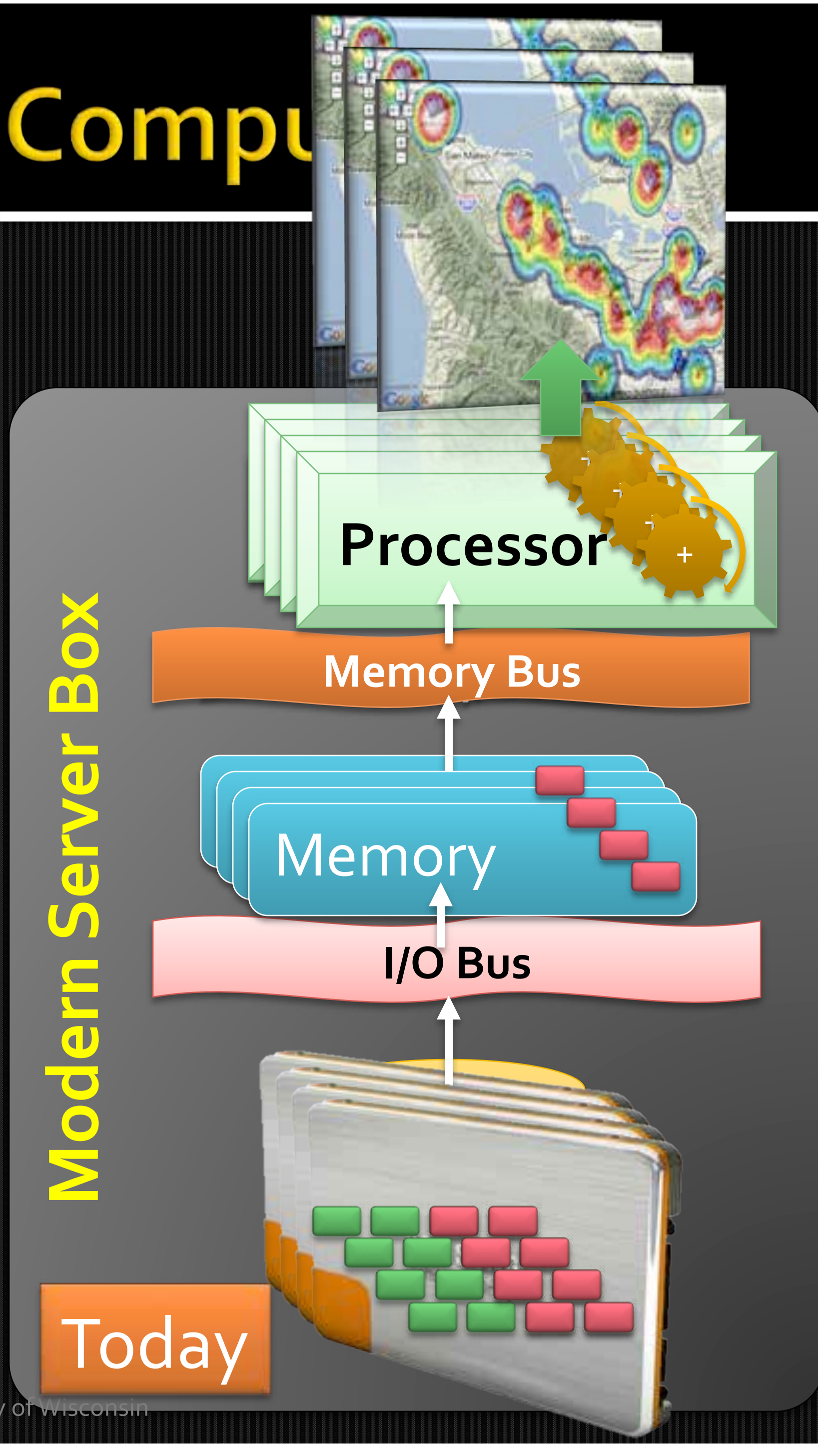
Rethinking Performance Goals



Data and Compu

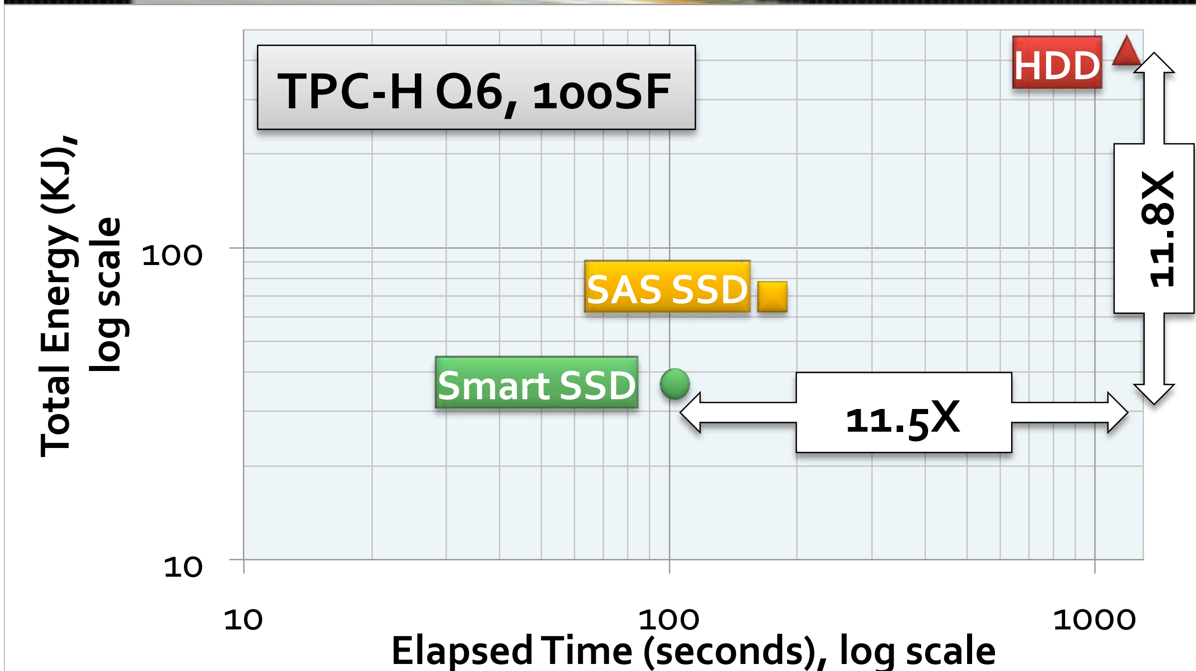
<u>Long Term</u>:
Raw computing and storage costs tends to zero!

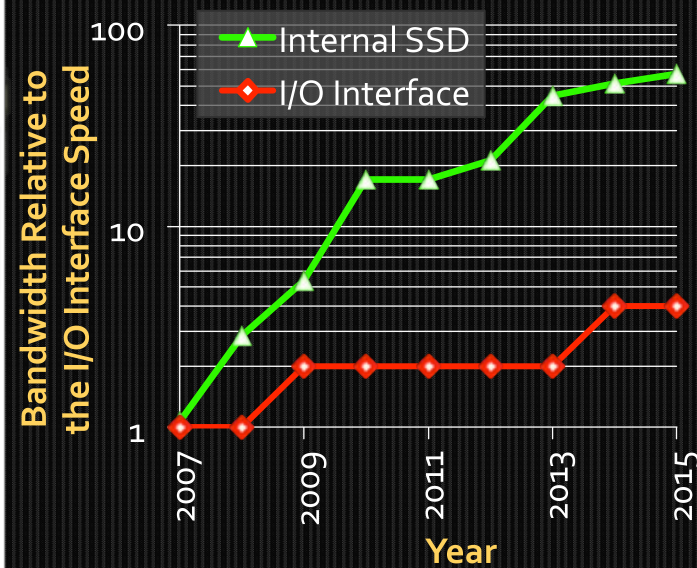
The cost is in moving data and powering the circuits/devices



Do et al., SIGMOD'13

Example: Flash SSD Architecture

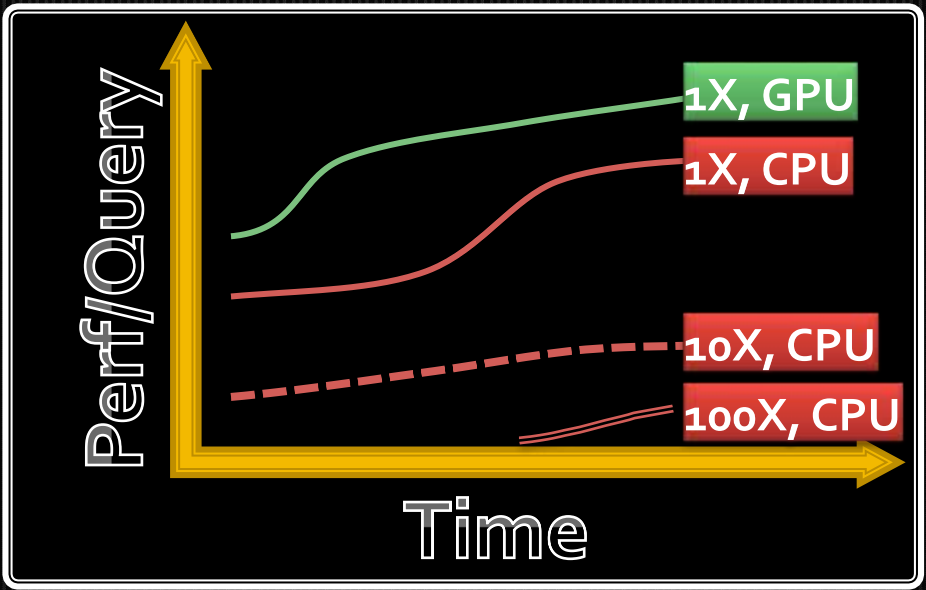


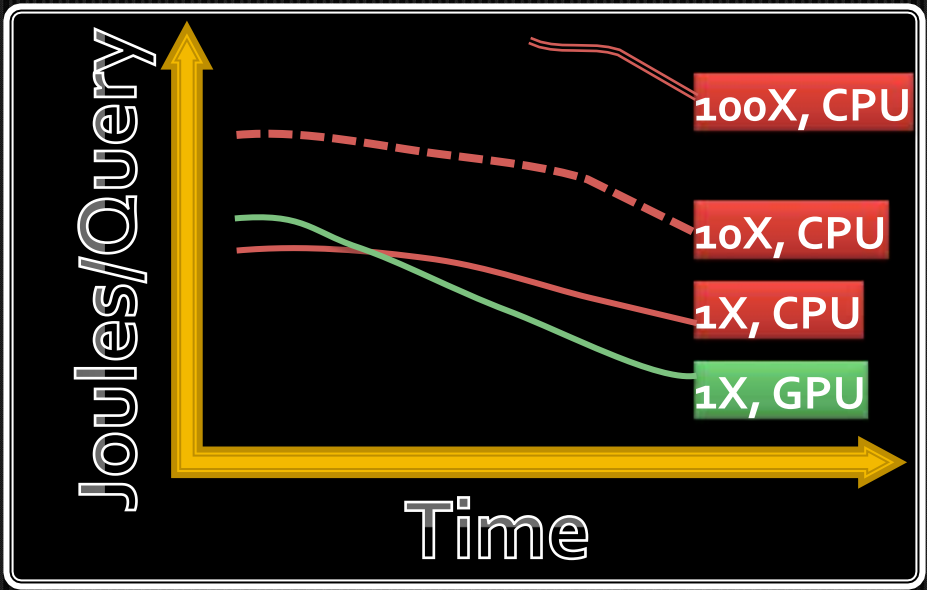


There are similar ways of using hardware creatively, e.g. IDISKs, ASICs, CGRA, FPGAs, or GPUs.

Basically, need hardware and software synergy!

Hardware Software Co-design: A Good Starting Point





Starting point:
Two queries

Scan	• Sequential read kernel
Scattered Read/Write	• Index access kernel

Conclusions and Future Work

Transformative architectural changes at all levels (CPU, memory subsystem, I/O subsystem) is underway

Need to rethink data processing kernels

- Run @ current bare metal speed

Need to think of hardware software co-design