# To Share or Not to Share?

Ryan Johnson

StagedDB Project Group

Nikos Hardavellas, Ippokratis Pandis, Naju Mancheril,
Stavros Harizopoulos, Kivanc Sabirli

Anastasia Ailamaki, Babak Falsafi

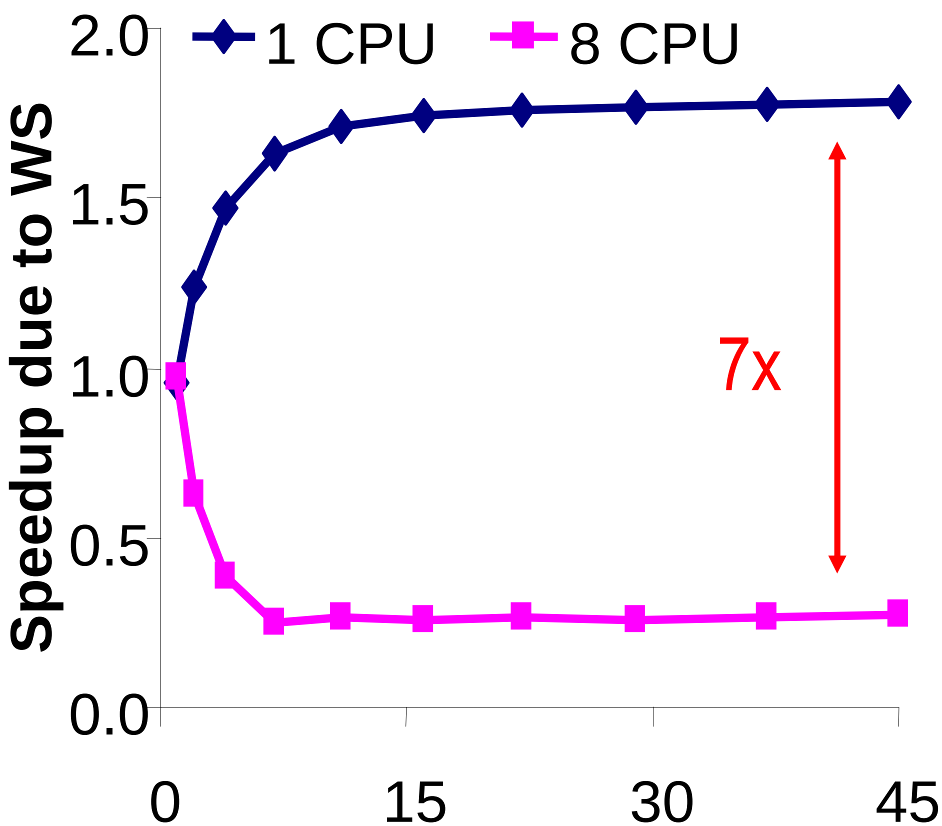
Databases
@ Carnegie Mellon

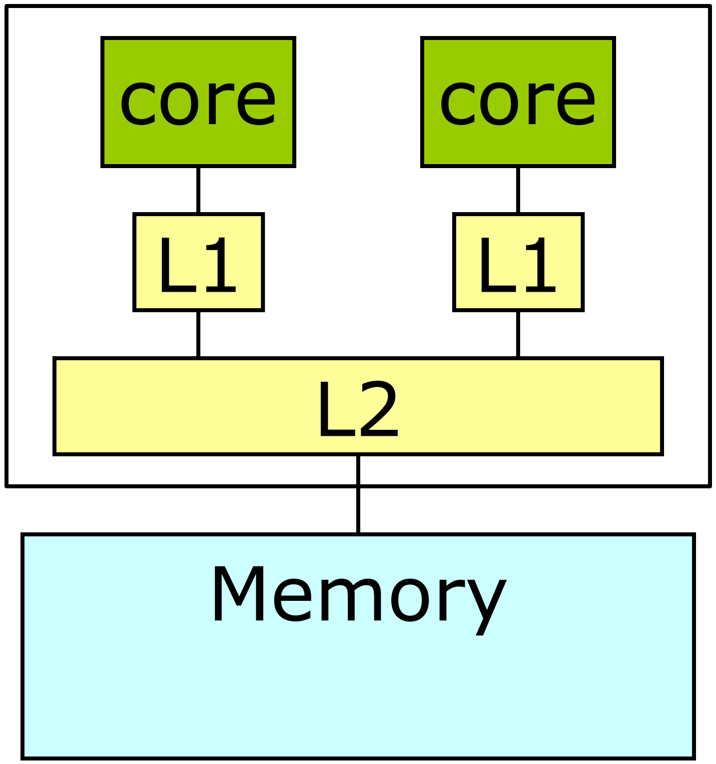
# Sharing Redundant Work in Queries

- Many queries in system
  - Many similar requests
  - Redundant work
- Work Sharing
  - Detect redundant work
  - Compute once and share
- Big win for uniprocessors, I/O

➡ Work sharing can hurt performance!

# Performance Impact of Work Sharing




➡ Must apply work sharing adaptively

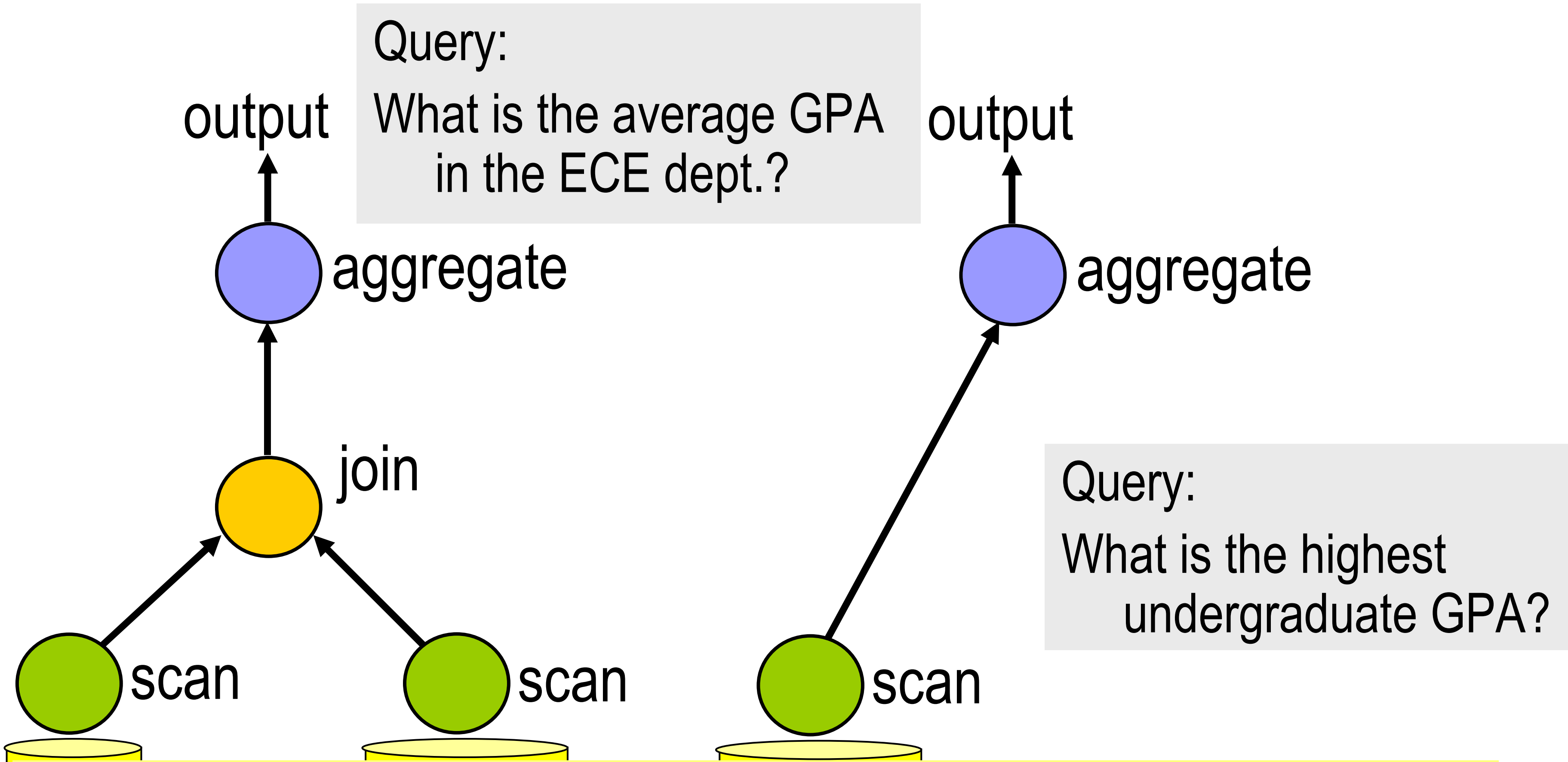
# Contributions

- **Observation**
  - Work sharing can hurt performance on parallel hardware
- **Analysis**
  - Develop intuitive analytical model of work sharing
- **Application**
  - Model-based policy outperforms static ones by up to 6x

# Outline

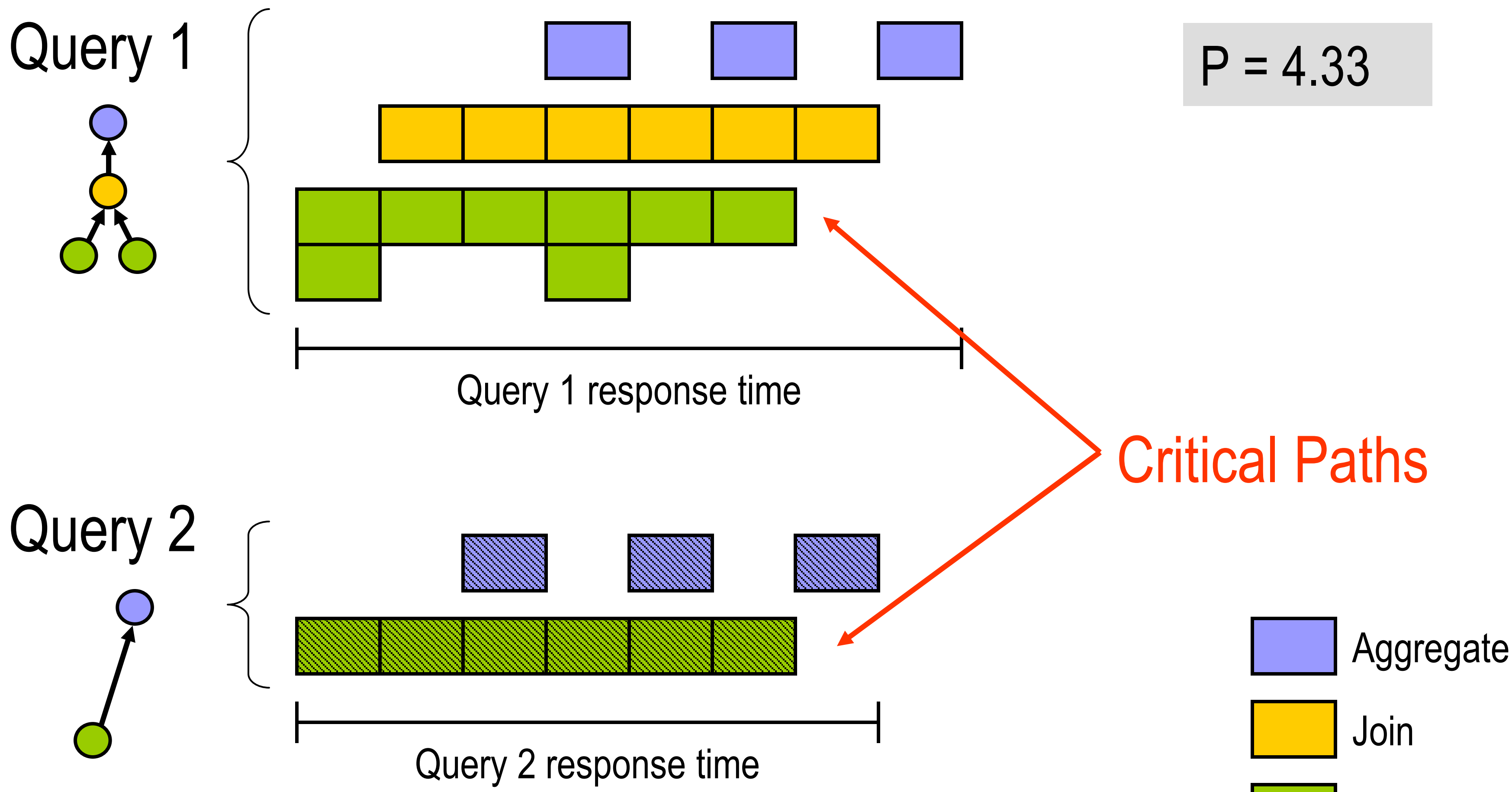
- Introduction
- **Motivation**
- Model and Validation
- Analysis and Experiments

# Motivation Behind Work Sharing



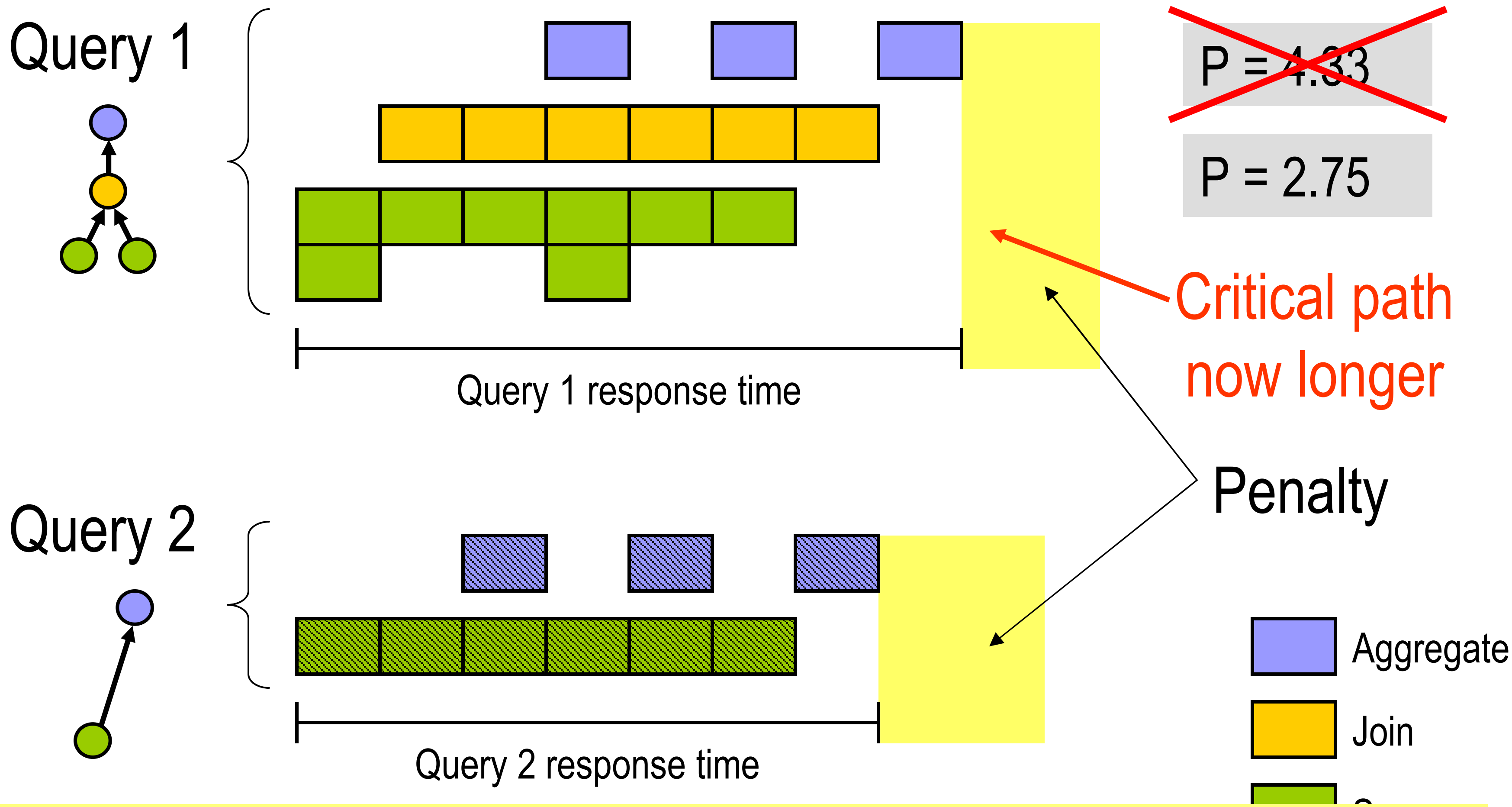

- 2x speedup or better on uniprocessors [hariz05]

# Work Sharing vs. Parallelism




Independent Execution

# Work Sharing vs. Parallelism




- Need to predict changes in critical path length

# Challenges of Exploiting Work Sharing

- Independent execution
  - Load reduction from work sharing can be useful
- Work sharing
  - Indiscriminate application can hurt performance
- To share or not to share?
  - System and workload dependent
  - Must make decision at runtime

➡ Need lightweight model of work sharing

# Outline

- Introduction
- Motivation
- **Model and Validation**
- Analysis and Experiments

# Basis for a Model

- "Closed" system
  - Consistent high load
  - Throughput computing
  - Assumed in most benchmarks
- Little's Law governs throughput
  - Total work *not* a direct factor
  - Higher response time = lower throughput

➡ Load reduction secondary to response time

# Predicting Response Time

m = #Queries
n = #CPUs

- Case 1: Compute-bound

$$T(m,n) = \frac{u(m)}{n} = \frac{\text{Requested Utilization}}{\text{Available Processors}}$$

- Case 2: Critical path-bound

$$T(m,n) = p_{\max}(m) = \text{Delay at slowest pipe stage}$$

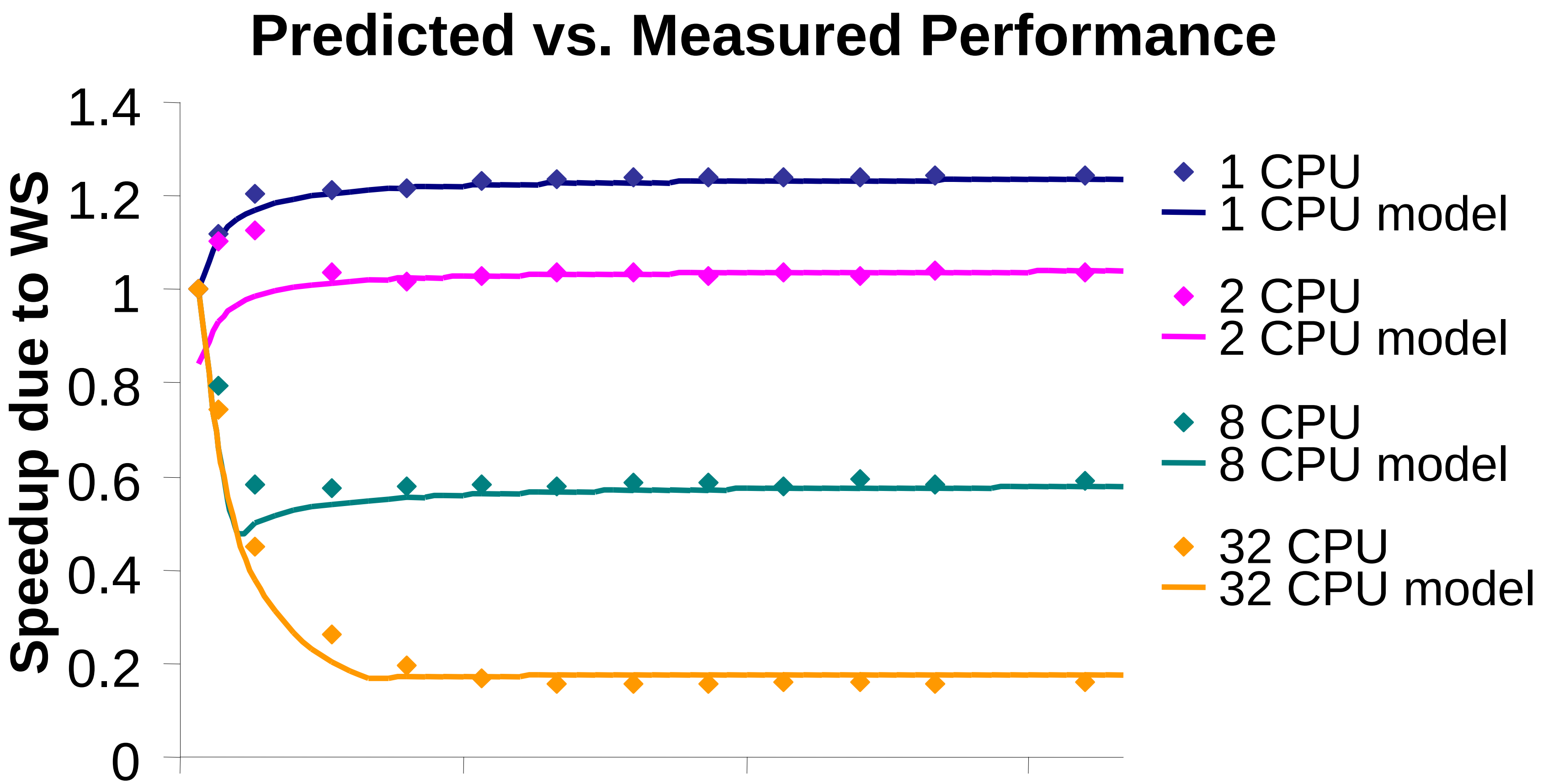
- Larger bottleneck determines response time

➡ Model provides u(m) and $p_{max}$(m)

# Experimental Setup

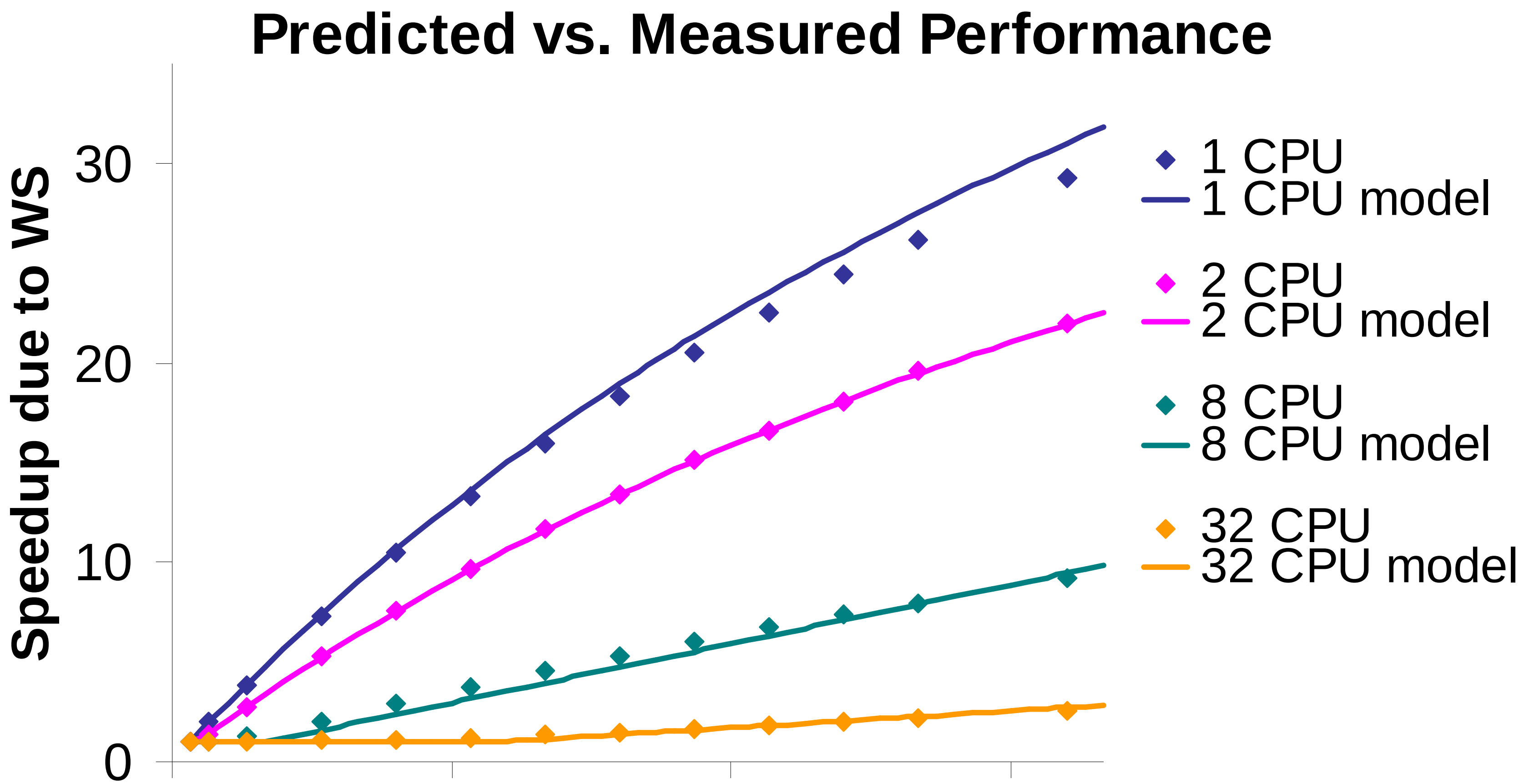
- Hardware
  - Sun T2000 “Niagara” with 8 GB RAM
  - 8 cores (32 threads)
  - Solaris processor sets vary effective CPU count
- Cordoba
  - Staged DBMS
  - Naturally exposes work sharing
  - Flexible work sharing policies
- 1GB TPCH dataset

# Model Validation: TPCH Q1




- Avg/max error: 5.7% / 22%

# Model Validation: TPCH Q4

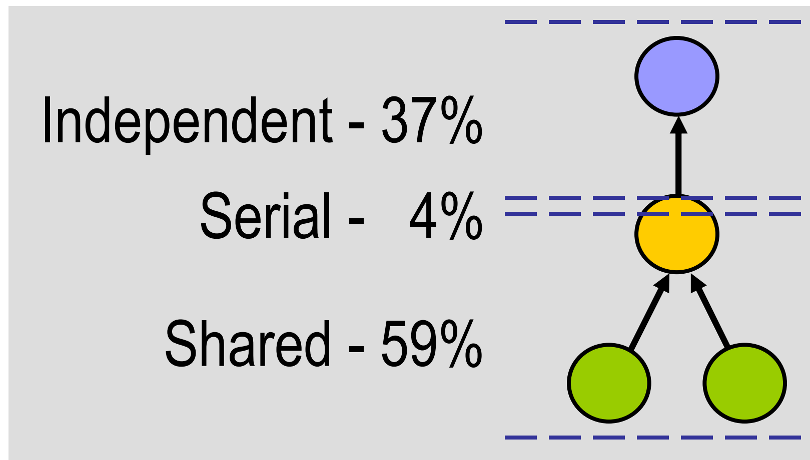



➡ Behavior depends on both system and workload

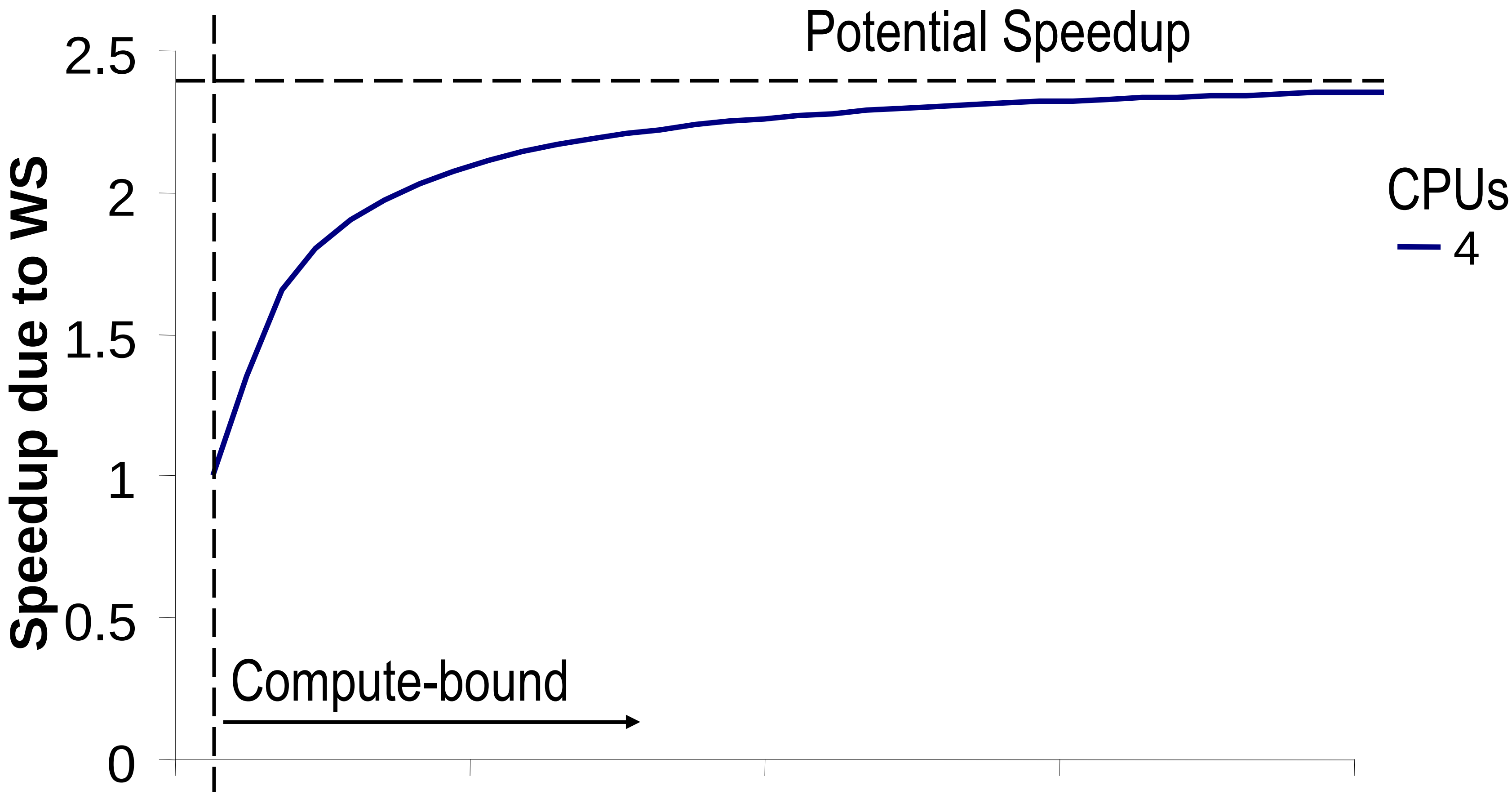
# Outline

- Introduction
- Motivation
- Model and Validation
- **Analysis and Experiments**

# Exploring WS vs. Parallelism

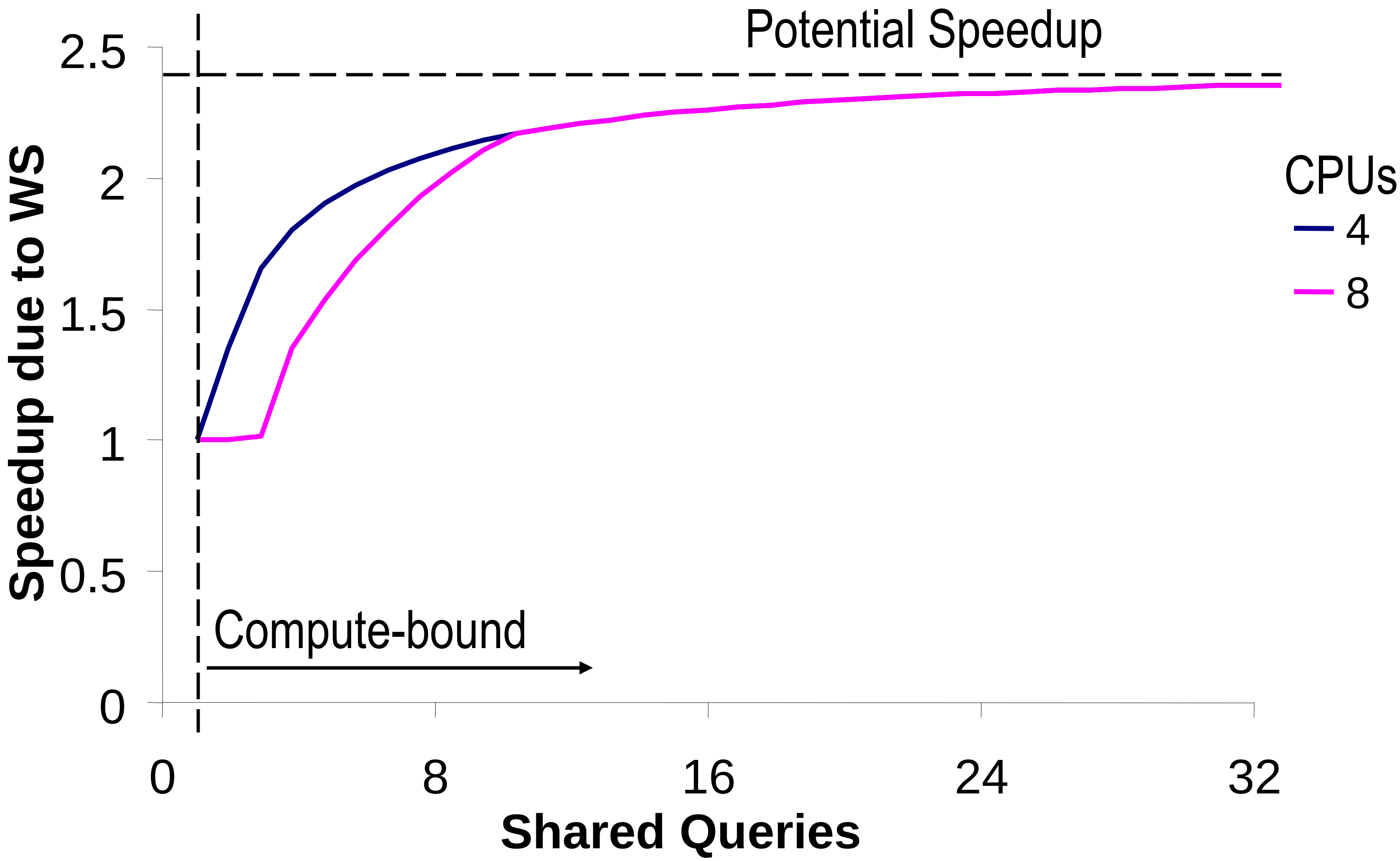


# Exploring WS vs. Parallelism

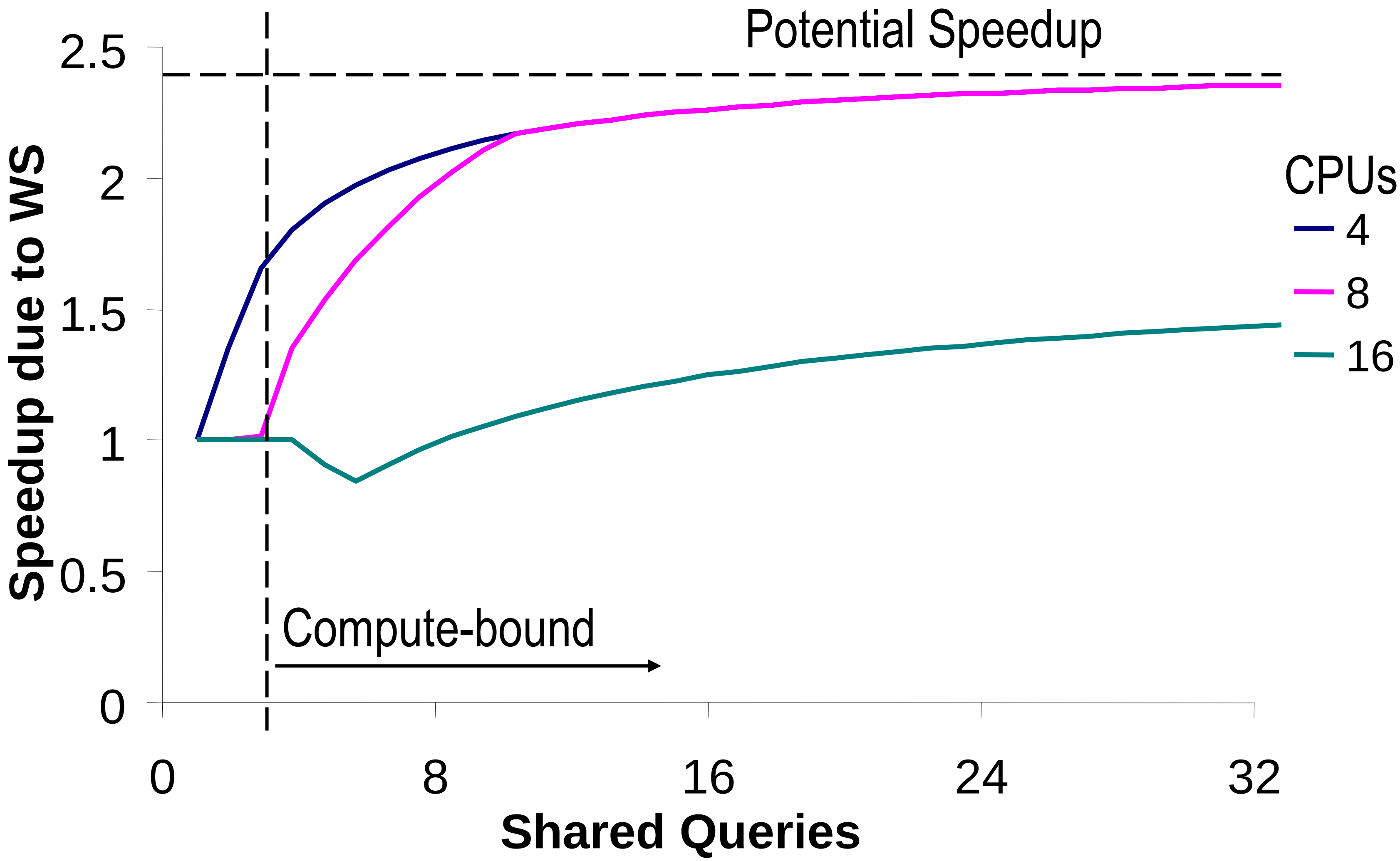



➡ Behavior corroborates previously published results

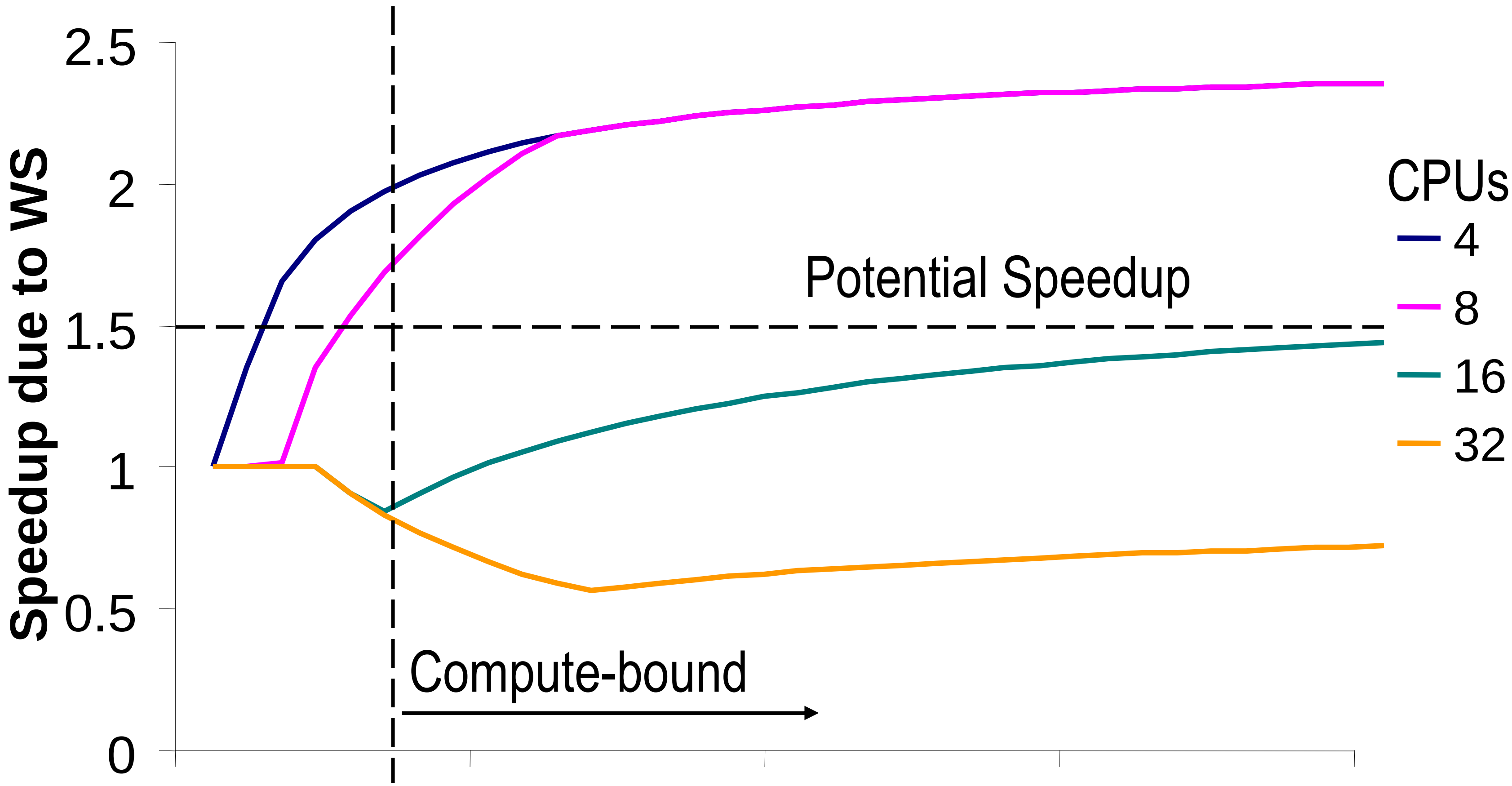
# Exploring WS vs. Parallelism



# Exploring WS vs. Parallelism

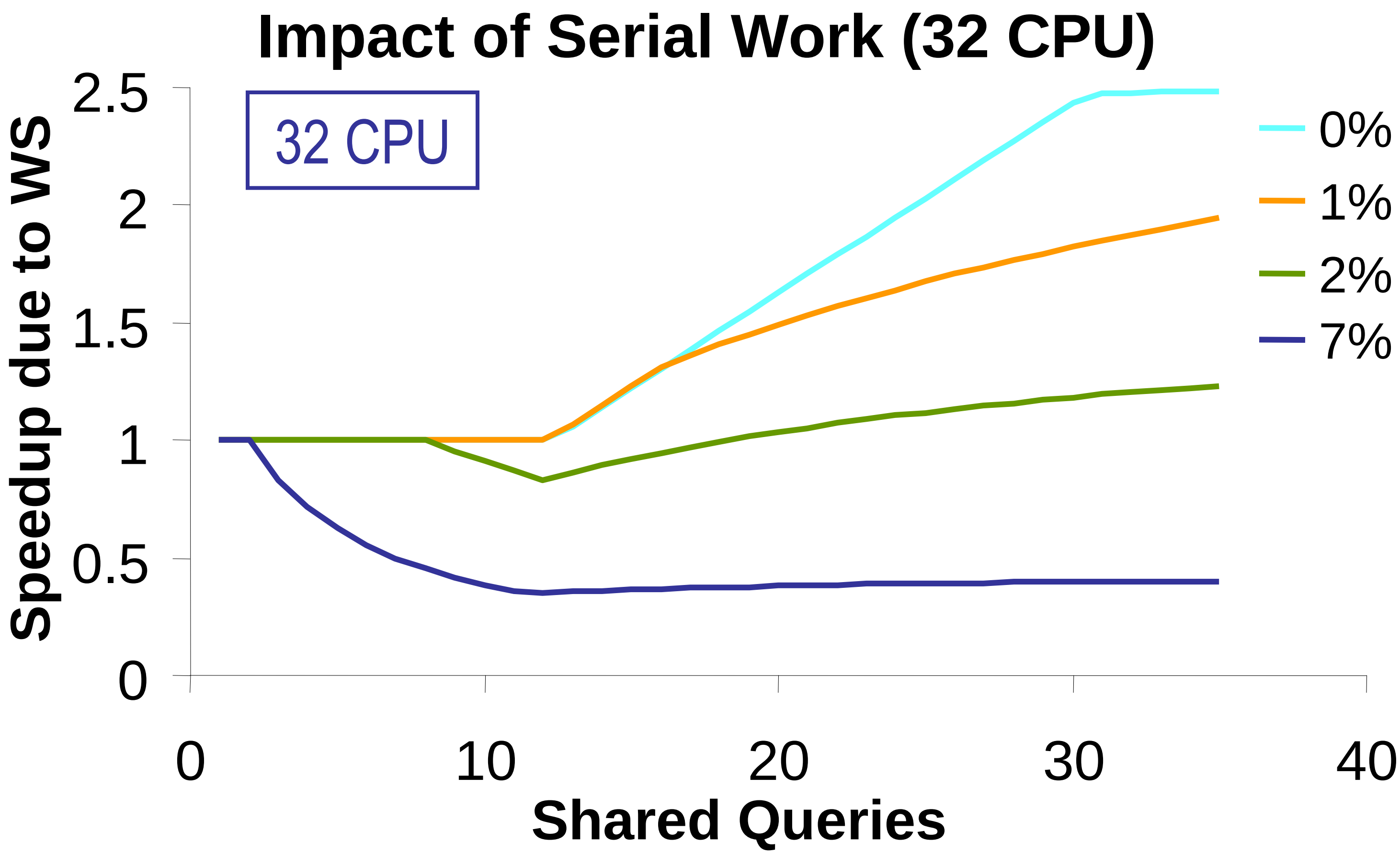


# Exploring WS vs. Parallelism




➡ More processors shift bottleneck to critical path

# Performance Impact of Serial Work



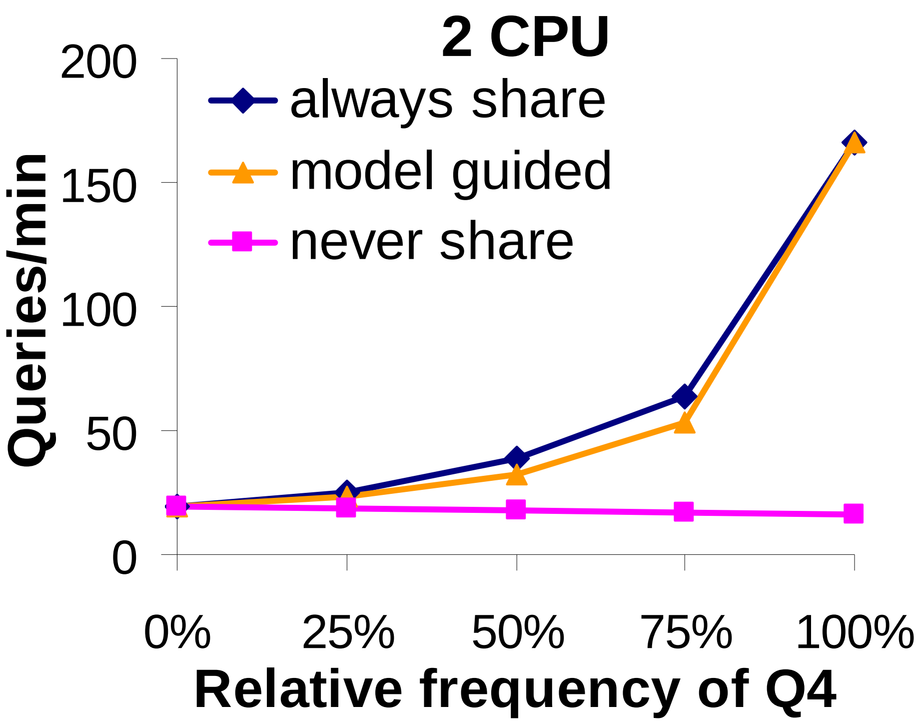

➡ Longer critical path causes major bottlenecks

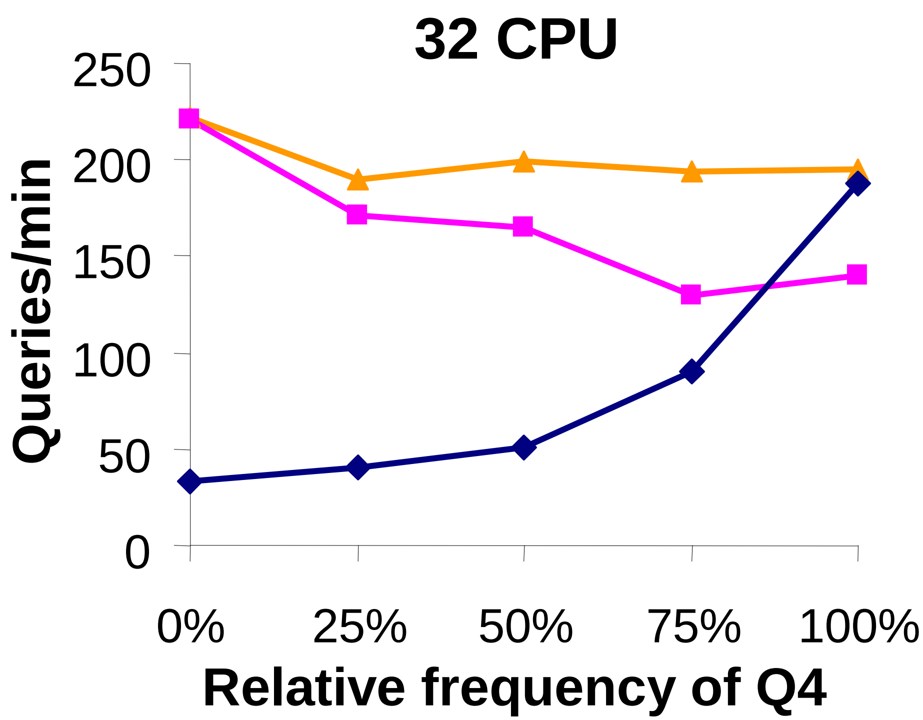
# Model-guided Work Sharing

- Integrate predictive model into Cordoba
  - Predict benefit of work sharing for each new query
- Consider multiple groups of queries at once
  - Shorter critical path, increased parallelism
- Experimental setup
  - Extract model parameters with profiling tools
  - 20 clients submit mix of TPCH Q1 and Q4

➡ Compare against always-, never-share policies

# Comparison of Work Sharing Strategies




Model-guided policy balances critical path and load

# Related Work

- Many existing work sharing schemes
  - Identification occurs at different stages of query lifetime
  - All allow pipelined query execution

| | | | | | |
|---|---|---|---|---|---|
| | Materialized Views [rouss82] | Multiple Query Optimization [roy00] | Staged DBMS [hariz05] | Cooperative Scanning [lang07] | |
| Early | Schema design | Query compilation | Query execution | Buffer Pool Access | Late |

➡ Model describes all types of work sharing

# Conclusions

- Work sharing can hurt performance
  - Highly parallel, memory resident machines
- Intuitive analytical model captures behavior
  - Trade-off between load reduction and critical path
- Model-guided work sharing highly effective
  - Outperforms static policies by up to 6x

# References

[hariz05] S. Harizopoulos, V. Shkapenyuk, and A. Ailamaki. "QPipe: A Simultaneously Pipelined Relational Query Engine." In Proc. SIGMOD, 2005.

[lang07] C. Lang, B. Bhattacharjee, T. Malkemus, S. Padmanabhan, and K. Wong. "Increasing Buffer-Locality for Multiple Relational Table Scans through Grouping and Throttling." In Proc. ICDE, 2007.

[rouss82] N. Roussopoulos. "View Indexing in Relational databases." In ACM TODS, 7(2):258-290, 1982.

[roy00] P. Roy, S. Seshadri, S. Sudarshan, and S. Bhobe. "Efficient and Extensible Algorithms for Multi Query Optimization." In Proc. SIGMOD, 2000.

http://www.cs.cmu.edu/~StagedDB/