# A Trace-driven Comparison of Algorithms for Parallel Prefetching and Caching


Tracy Kimbrel
Brian Bershad
Anna R. Karlin
University of Washington

Andrew Tomkins
R. Hugo Patterson
Garth A. Gibson
Carnegie Mellon University

Pei Cao
University of Wisconsin-Madison

Edward W. Felten
Kai Li
Princeton University

# Motivation

Overcoming the I/O bottleneck by telling the I/O system what you want before you want it

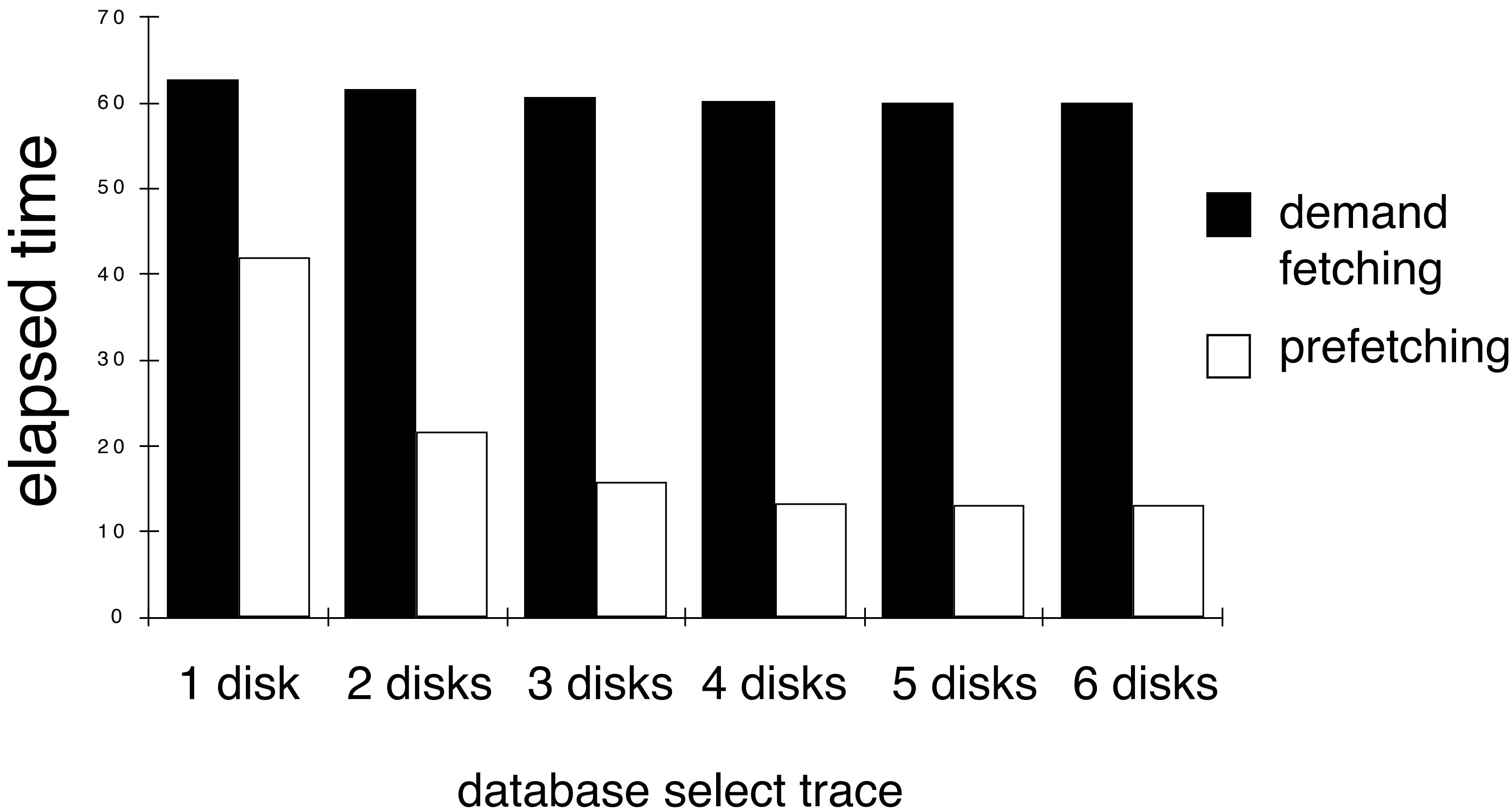
# Outline

- Background
- Previous algorithms for prefetching and caching
- Simulation framework and results
- A new approach that always wins
- Future directions

# Techniques to overcome I/O bottleneck

- caching for re-use
- disk arrays for parallel I/O
- deep prefetching to hide latency

# Deep prefetching is a big win



# Deep prefetching requires advance knowledge

Acquiring advance knowledge (hints) of I/O accesses

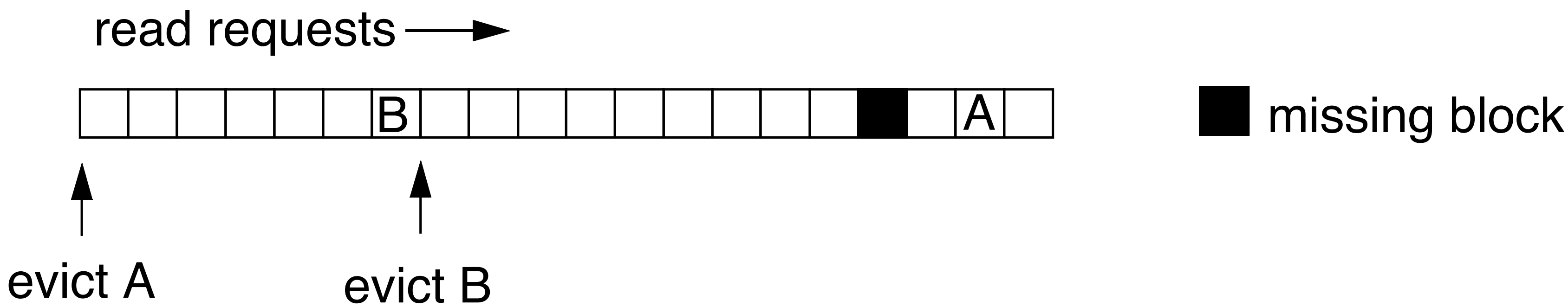
- Programmer-supplied hints (Patterson Gibson et. al 1995)
- Compiler techniques (Mowry et. al 96)

# Using advance knowledge

- “smart” prefetching
- optimal cache replacement

# Problem is hard even with advance knowledge

- Prefetching too eagerly can increase I/O load




- Caching too long reduces effectiveness of prefetching

# Problem is even harder with parallel disks

Cache replacement decisions can affect load balance among the disks

# Focus of this study

- Single process
- Read-only
- Fully hinted request sequence

These simplify evaluation; but the algorithms work without them

# Tradeoffs faced by algorithm designer

- Aggressive prefetching vs. optimal cache replacement

- Does load balance depend on careful cache replacement decisions?

- Amount of lookahead information to use

# Main result

- None of the previous algorithms performs well for all workloads and is practical

- There is a single, practical algorithm for prefetching and caching that performs well for all workloads in our setting

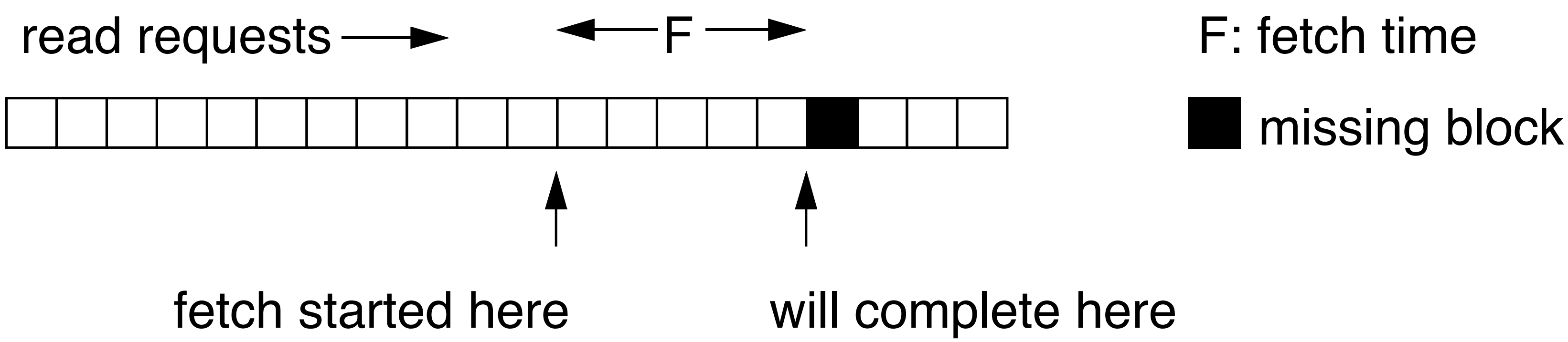
# Outline

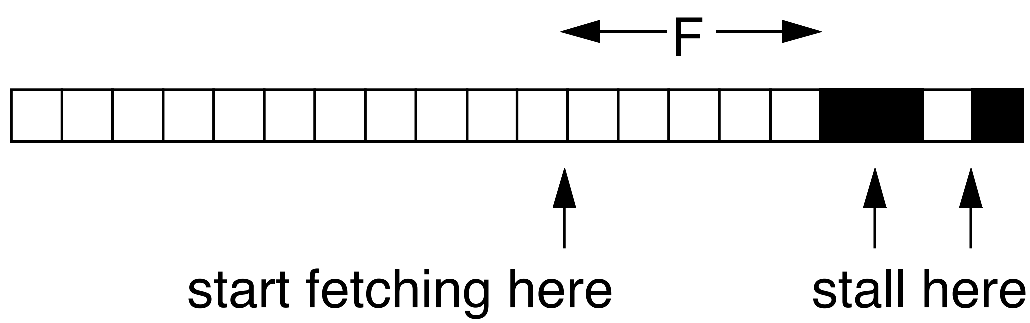
- Background
- Previous algorithms for prefetching and caching
- Simulation framework and results
- A new approach that always wins
- Future directions

# Algorithm #1: Fixed Horizon

(Patterson, Gibson, Ginting, Stodolsky, Zelenka)

- Prefetch “just in time” to serve request



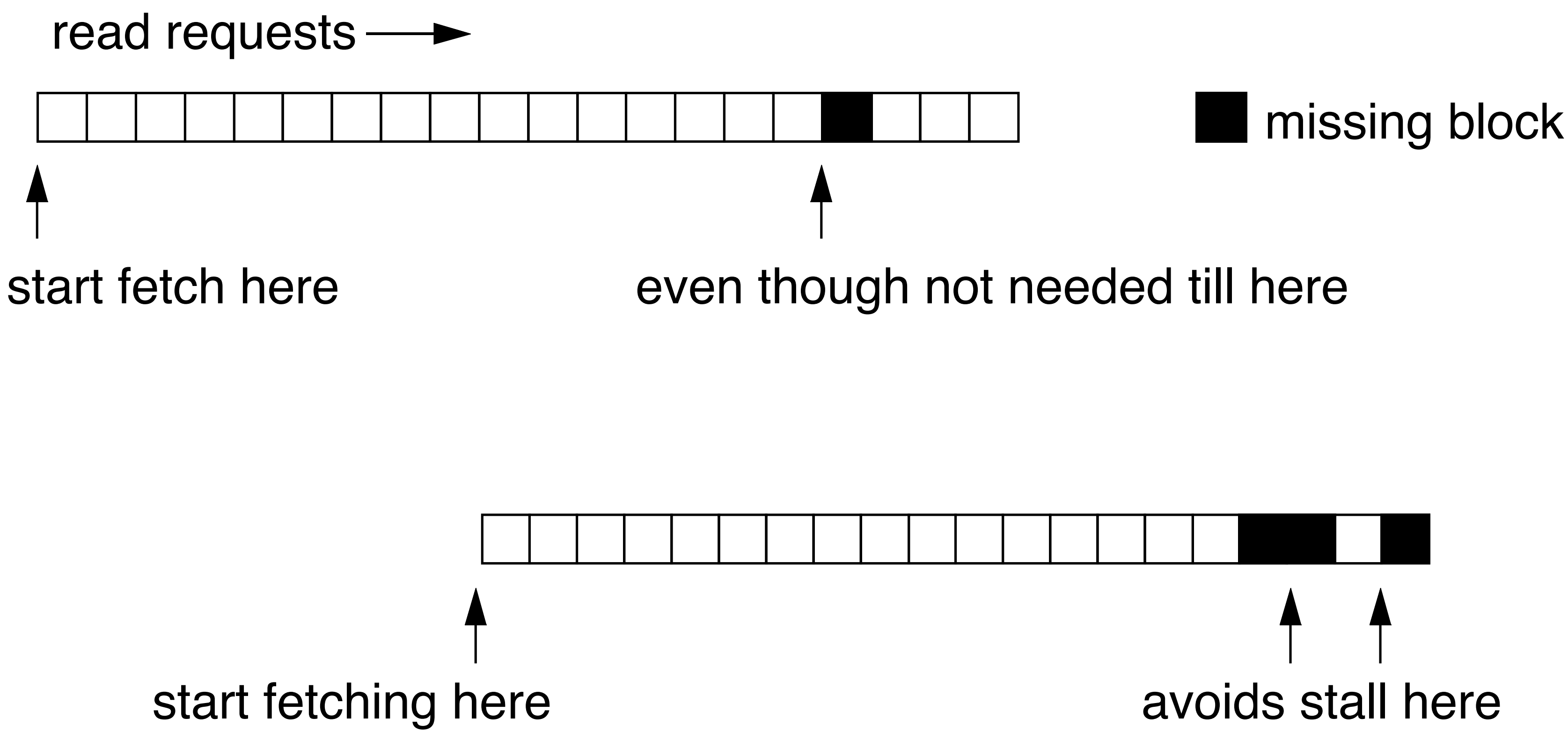



# Algorithm #1: Fixed Horizon

- Late prefetching (just-in-time) => near-optimal replacement decisions
- Makes no effort to balance disk loads through replacement decisions
- Uses least lookahead

# Algorithm #2: Aggressive
(Cao, Felten, Karlin, Li)

Prefetch whenever cache holds block not needed until after first missing block



# Algorithm #2: Aggressive

- Aggressive prefetching => sub-optimal replacement decisions => increased I/O load
- Makes no effort to balance disk loads through replacement decisions
- Uses more lookahead

# Algorithm #3: Reverse Aggressive

(Kimbrel and Karlin)

Construct schedule as does Aggressive but on reverse of input sequence; convert to schedule for original.

Impractical, but only algorithm with proven performance guarantee

# Algorithm #3: Reverse Aggressive

In theoretical model assuming constant fetch and process compute times:

**Theorem**

Elapsed time is always within $1 + \varepsilon$ of optimal

Worst case performance of other two algorithms is much worse.

# Algorithm #3: Reverse Aggressive

- Near-optimal balance between prefetching and cache replacement decisions
- Only algorithm to balance disk loads through careful replacement decisions - unimportant in practice
- Uses complete lookahead

# Outline

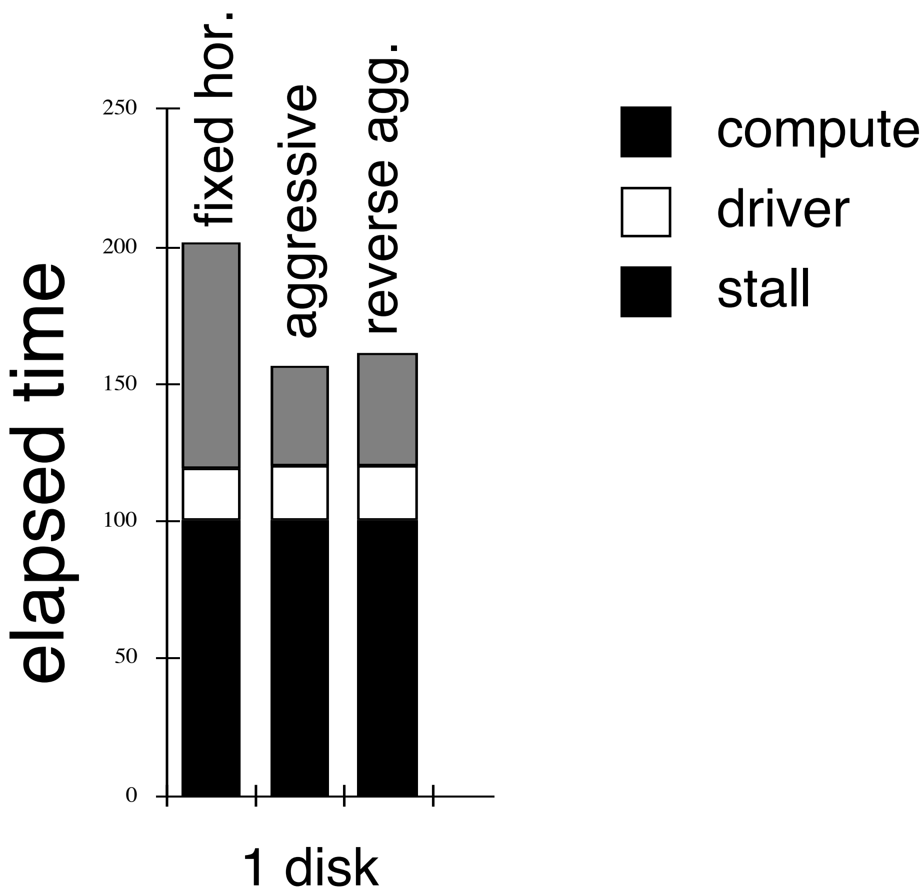
- Background
- Previous algorithms for prefetching and caching
- Simulation framework and results
- A new approach that always wins
- Future directions

# Trace-driven Simulations

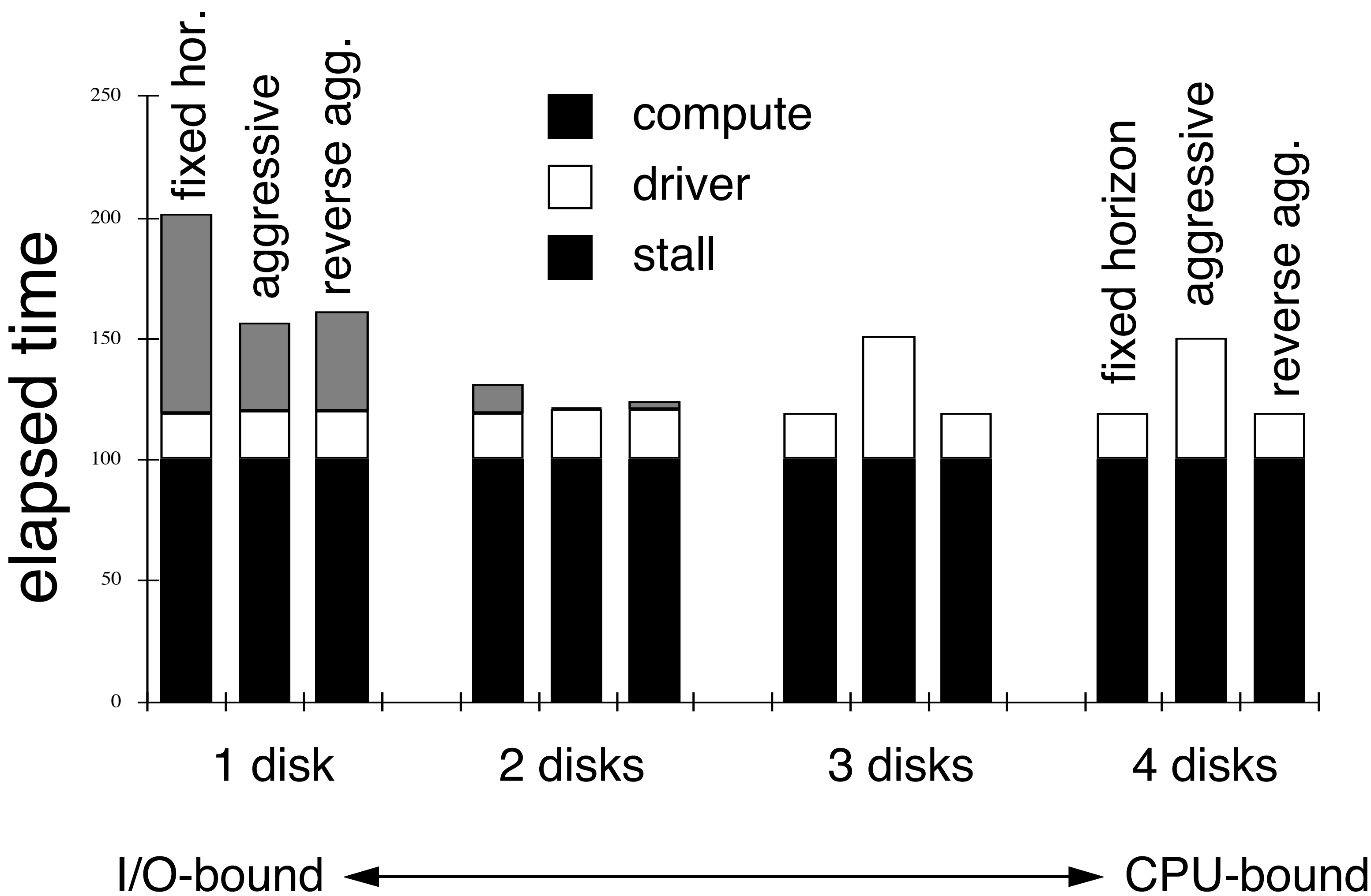
- cyclic accesses, text search, cache simulator, database join/select, 3D data visualization, source code analysis
- traces contain <read request, compute time> pairs
- accurate disk modelling (Kotz et. al, Ruemmler &Wilkes)
- data striped over disks
- algorithms incorporate disk scheduling
- Performance measure:

  Elapsed time = compute + I/O driver + stall

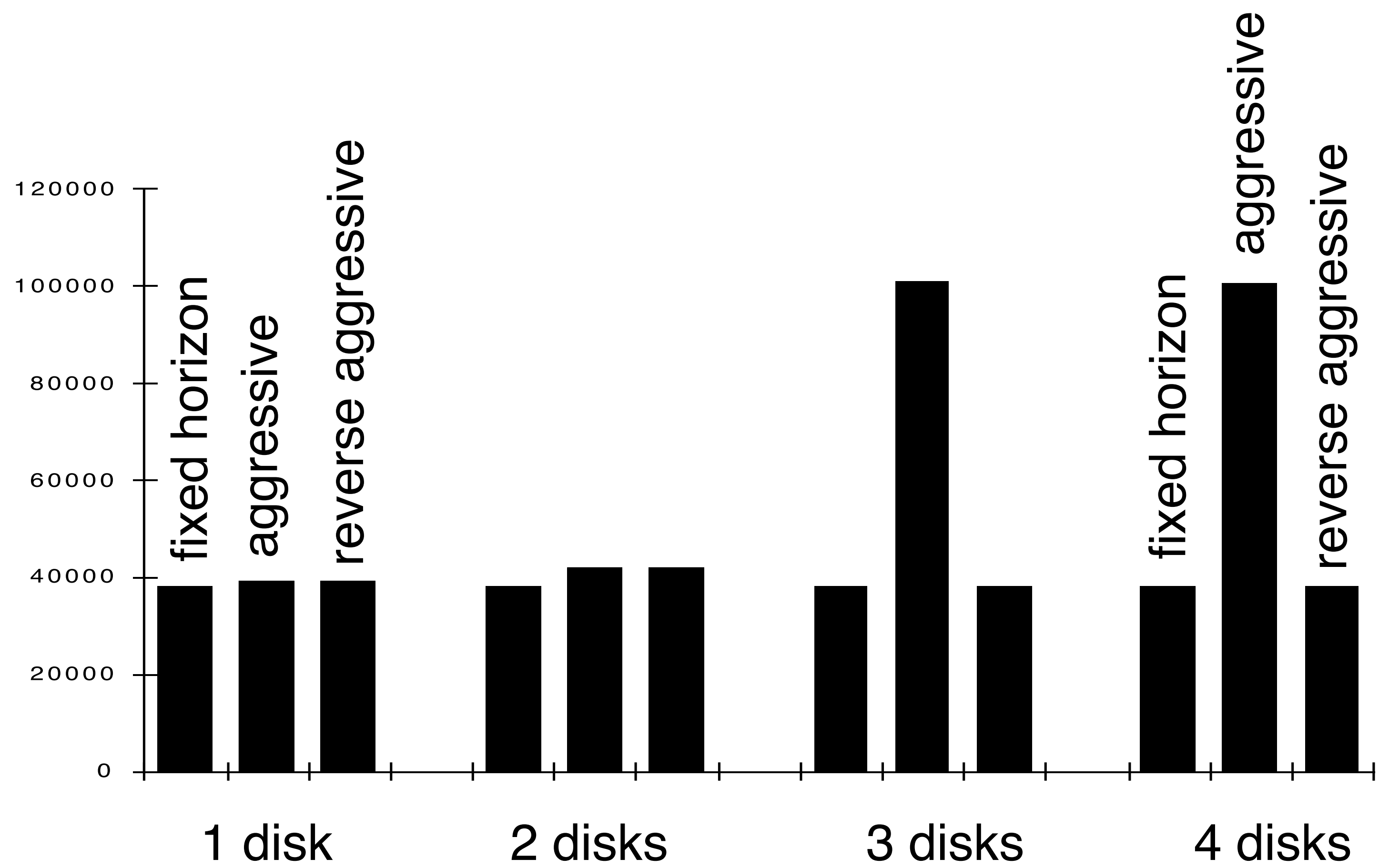
# Performance on cyclic accesses



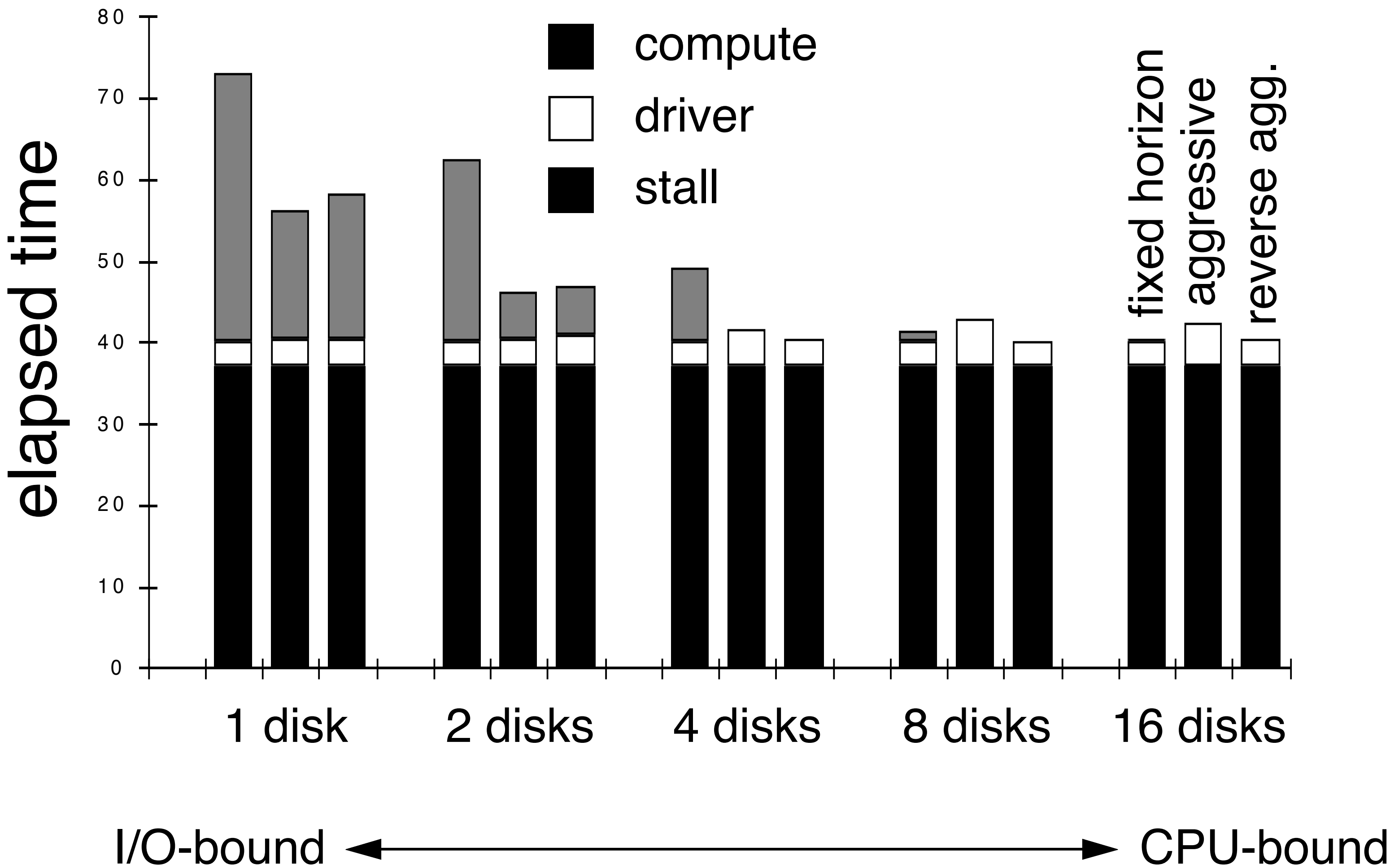
# Performance on cyclic accesses



# Number of fetches on cyclic access pattern



# Example application performance (text search)



# Problem

- Fixed horizon is not aggressive enough when application is I/O bound

- Aggressive performs too many fetches when application is CPU bound

- Reverse aggressive is impractical

**Solution:** new algorithm

# Outline

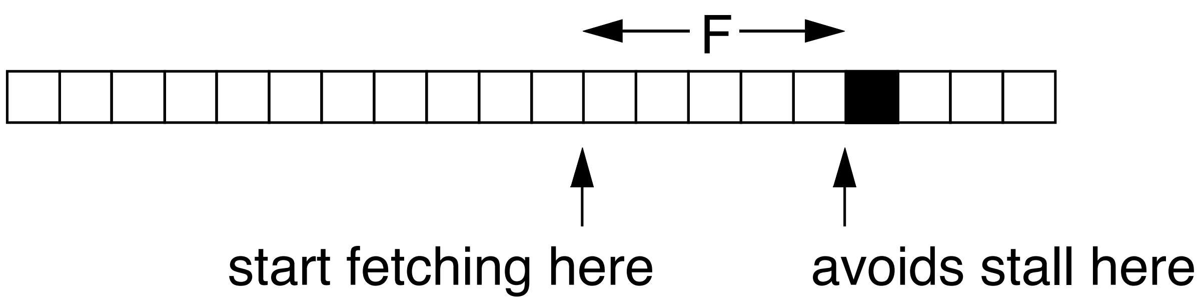
- Background
- Previous approaches
- Simulation framework and results
- A new approach that always wins
- Future directions

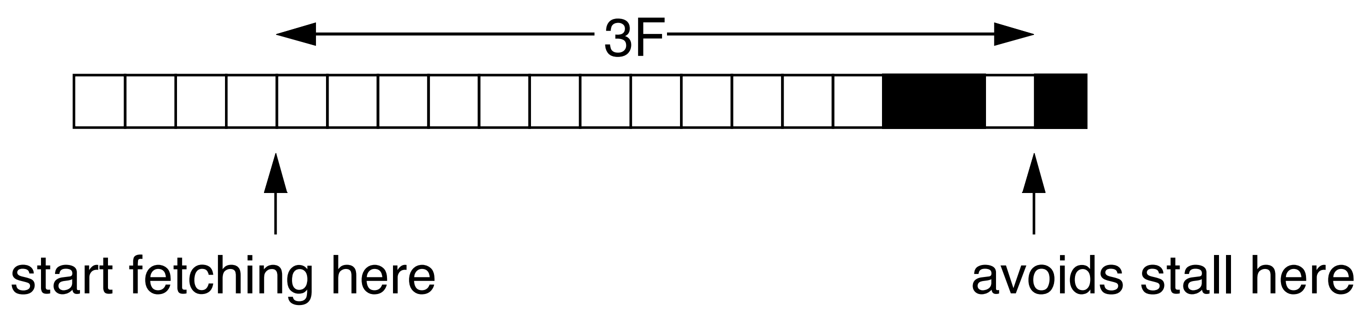
# New Algorithm: Forestall

For each disk, for each i, prefetch if $i^{th}$ missing block is within iF references

F: fetch time

missing block

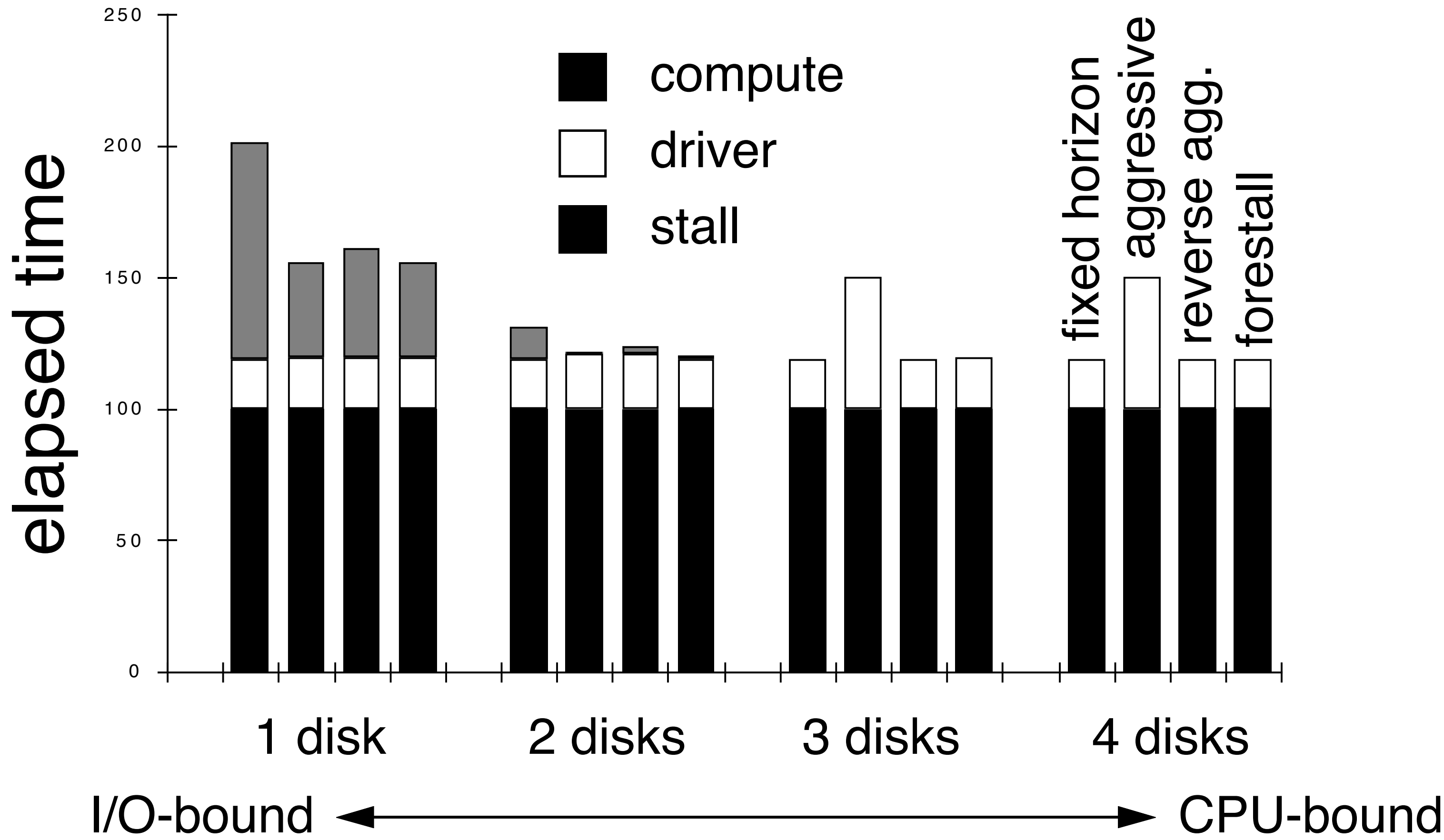


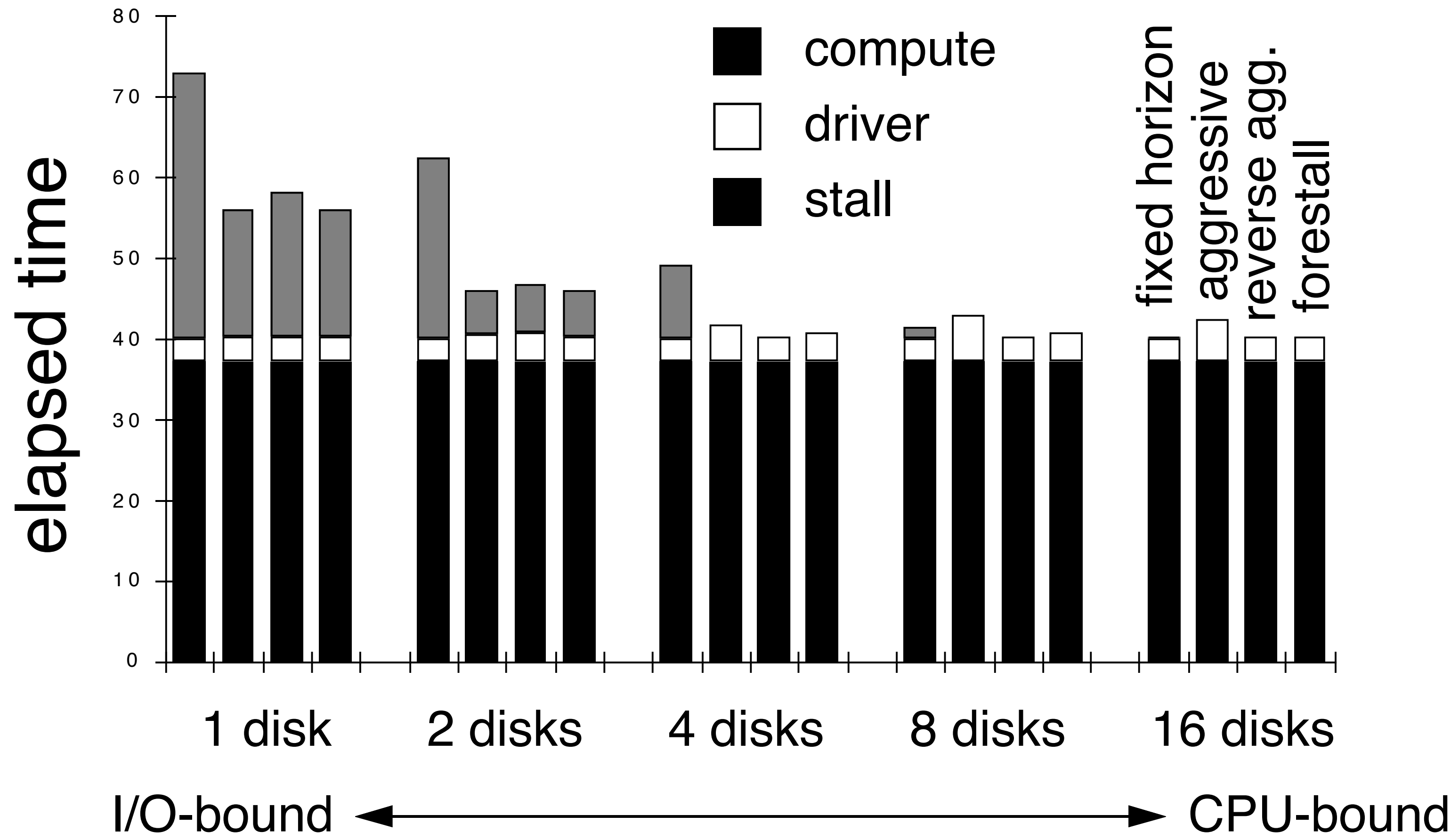
# New Algorithm: Forestall

- Performs aggressive prefetching when application is I/O bound and conservative prefetching when application is CPU-bound
- Makes no effort to balance disk loads through replacement decisions
- Intermediate use of lookahead

# Performance on cyclic accesses



# Example application performance (text search)



# Summary of results

| | Fixed horizon | Aggressive | Reverse Agg. | Forestall |
|---|---|---|---|---|
| CPU-bound perf. | √ | | √ | √ |
| I/O-bound perf. | | √ | √ | √ |
| Practical | √ | √ | | √ |

# Conclusions

- Deep prefetching is possible and is a big win for I/O-bound applications
- Existing practical algorithms have weaknesses, prefetching either too deep or not deep enough
- A hybrid algorithm has near-optimal performance

# Future directions

- Evaluate performance with incomplete and less accurate lookahead
- Multiple processes
- Develop programming interface and hinting applications