Distributed Prefetch-buffer/Cache Design for High Performance Memory Systems

Thomas Alexander & Gershon Kedem

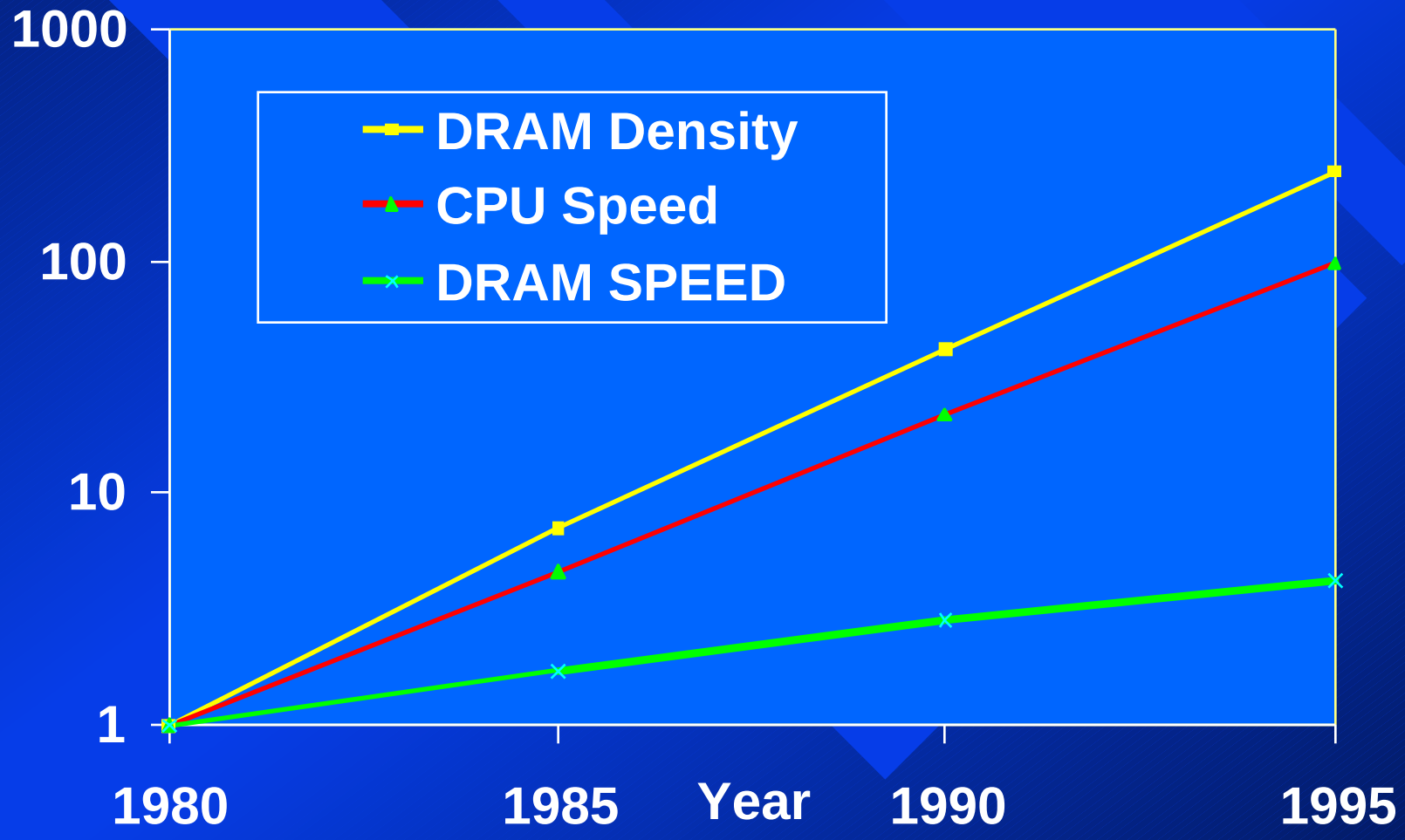
Duke University

Supported in part by Sun Microsystems

Objective

- Reduce CPU main memory speed gap
  - Hide DRAM access latency
  - Increase DRAM transfer bandwidth
- Price/Performance target for medium/high end computer systems

CPU & DRAM Improvement Trends



Memory Latency Overhead

Processor Cycle time	10ns
DRAM Access latency (t_{rac})	60ns
DRAM Column cycle time (t_{cac})	20ns

- 100MHz µP wastes ~14 (8,2,2,2) cycles on cache miss

$$CPI = P_{hit} \times C_{access} + (1-P_{hit}) \times M_{access}$$

$$CPI = 0.9 \times 1 + (1-0.9) \times 14 = 0.9+1.4 = 2.3$$

- 90% hit rate results in 130% memory access penalty.
- DRAM access penalty must be reduced

Current Solutions

L2 Caches

- Expensive to provide both capacity and speed.
- Cache heuristic ineffective for programs that traverse large data structures
  - cache capacity miss

SDRAM/RAMBUS Memories

- Do not reduce large DRAM latency

Main Idea

- **Predict** future CPU requests to main memory
- **Prefetch** data into SRAM prefetch buffers that are **integrated with the DRAM**

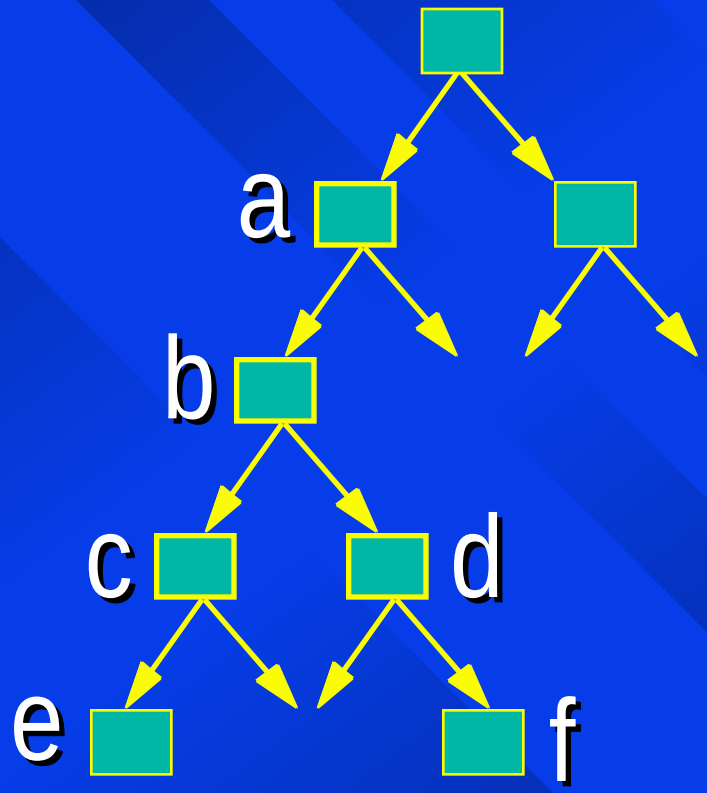
Prediction & Prefetch Problems

- How to **PREDICT** the CPU memory requests?
  - Complex request patterns.
  - Large prediction alphabet.
- How to **PREFETCH** the data in time?
- Reduce wrong prediction penalty.
- Conserve CPU-Memory bandwidth.

Observation

- Many programs **repeatedly** traverse large data structures.
  - link-list based programs.
  - linear and 2D array based programs.
- The CPU **read-request patterns** repeat over time.
- Patterns are used for the prediction scheme

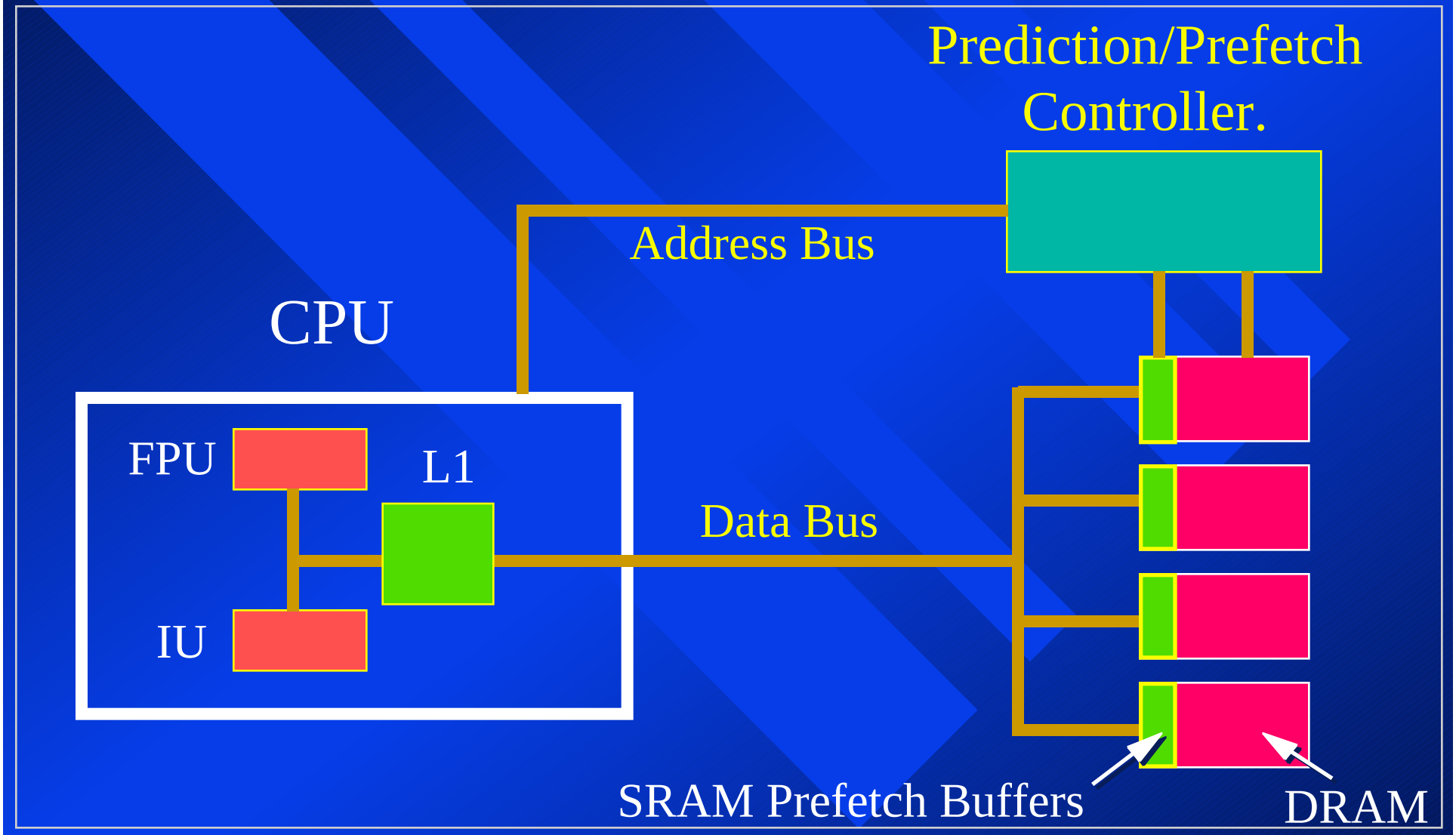
Example:
Binary Tree Data Structure.



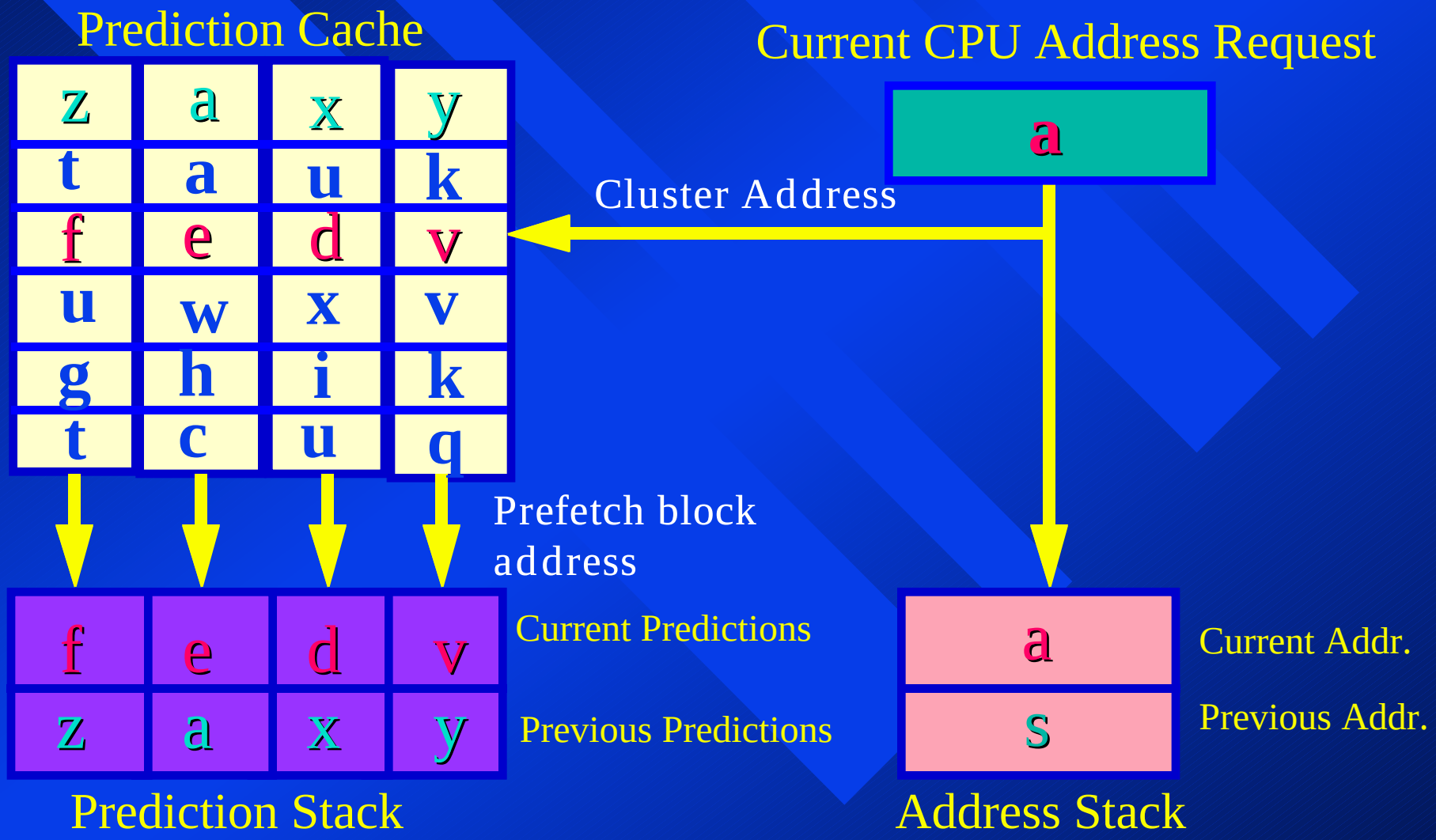
Program repeatedly traverse data structure

Address **c** or **d** will be requested after **b**

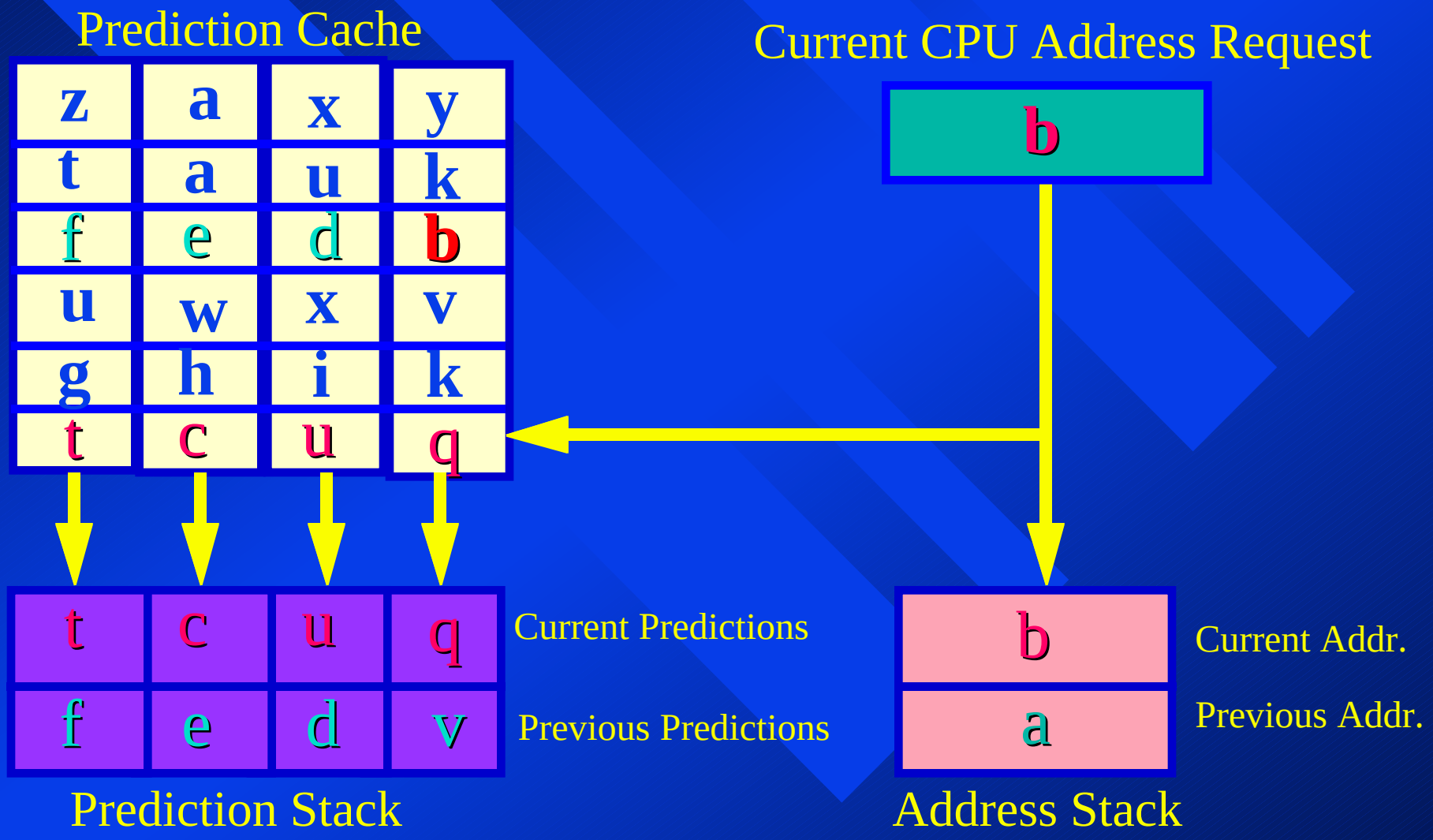
Memory System Architecture



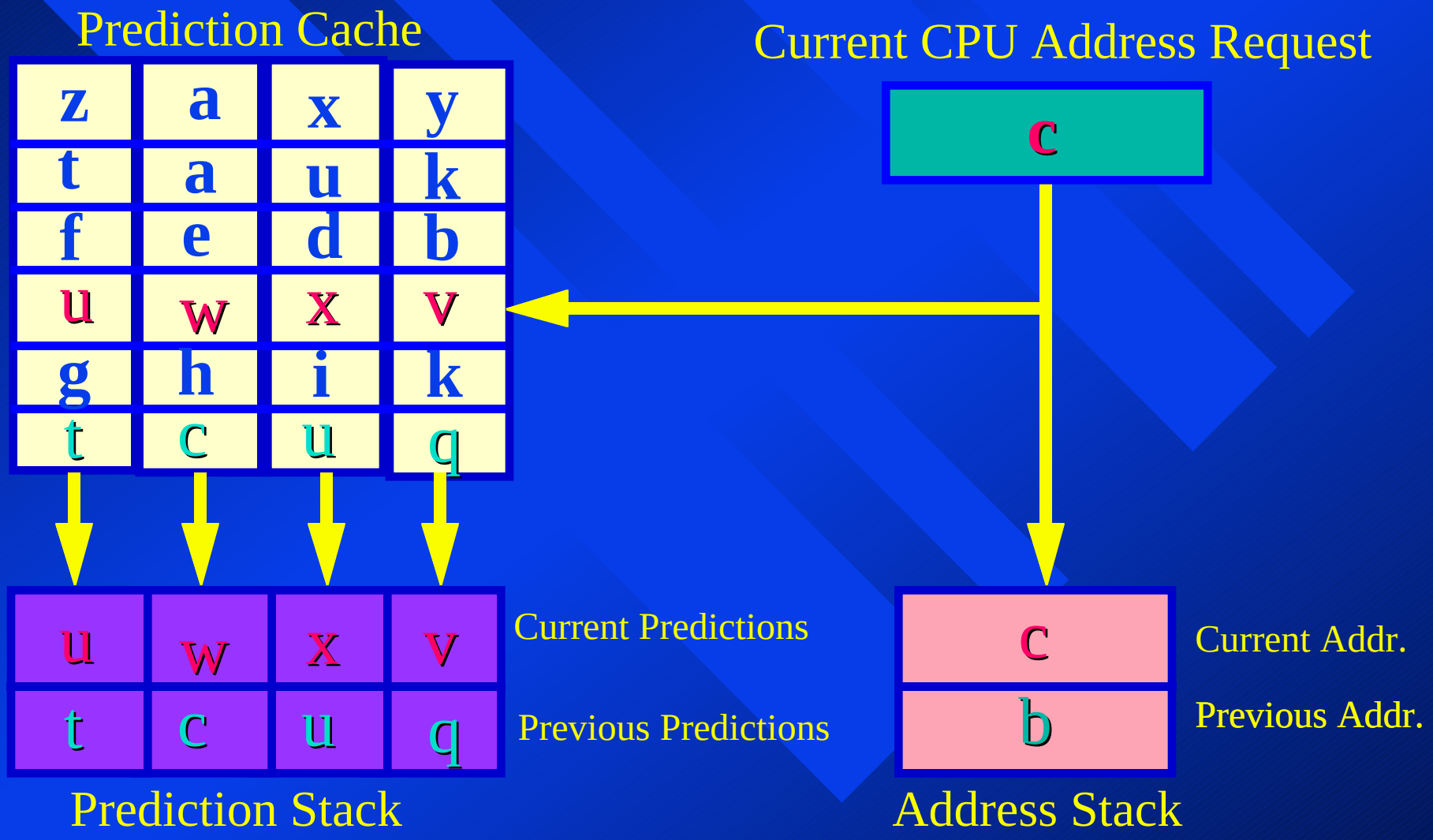
Prediction Example 1



Prediction Example 2



Prediction Example 3



The Prediction Cache

- Store physical addresses.
- 4 predictions per entry.
- Caching addresses rather than data!
- Clustering reduces the number of unique addresses.
- Large prefetch buffers reduce the number of address bits for each prediction.
- Update mechanism tracks program execution

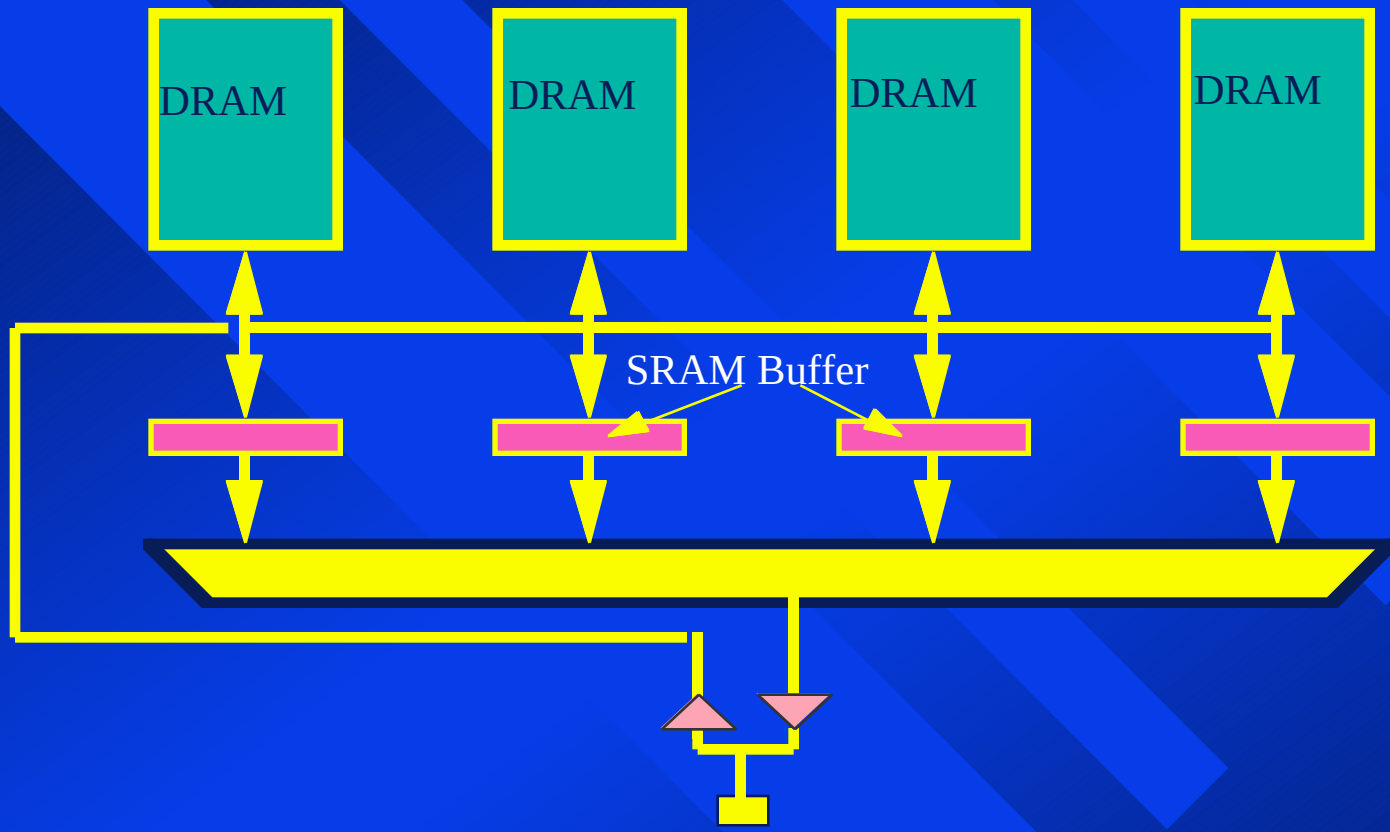
Prediction & Prefetch Algorithm

- Use integrated prefetch buffers (internal to the IC).
- Cluster Addresses into blocks
- Store the address-request patterns in a cache.
- Use the patterns to predict future requests.
- Use **4** predictions to capture program branching
- Prefetch simultaneously the **4** predictions into the SRAM prefetch buffers (Interleave memory)

Design Example:
(Using EDRAM memory)

- EDRAM IC with integrated SRAM buffers
- **Not IDEAL** IC but provides sufficient functionality to test our ideas.
- 4-way interleaved memory
- Memory Access time (t_{RAS}): 45ns
- Buffer Access time (t_{CAS}): 15ns
- µP cycle time: 10ns
- Memory bus cycle time: 10ns

EDRAM Memory



- Limitation: - No concurrent accesses to banks

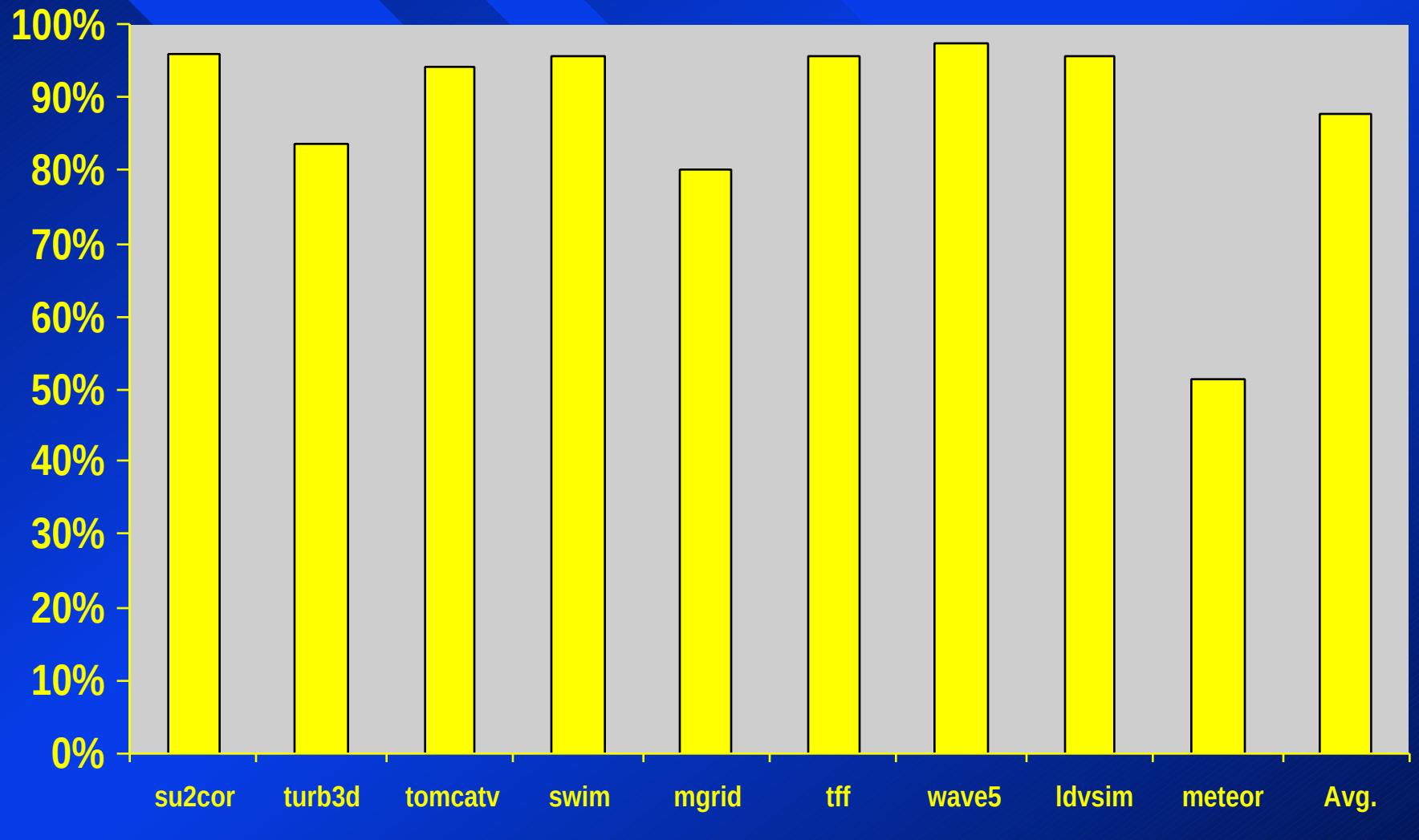
Design Example: (continue)

- Total memory size: 64 MB
- Prediction cache size: 32 KB
- Address Cluster Size: 256 B
- Prefetch Block Size: 2 KB
- Number of Guesses: 4

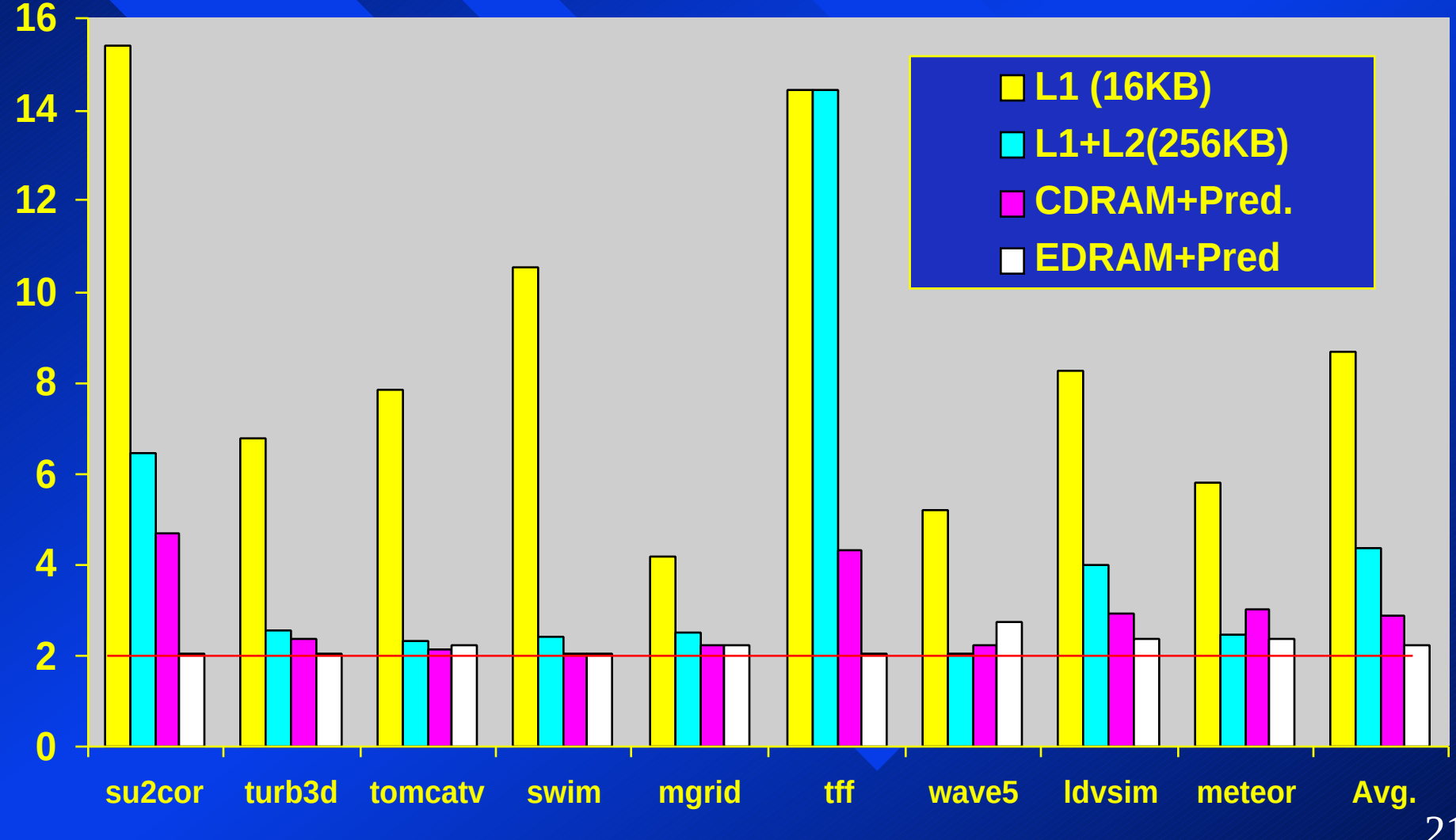
Simulation Setup

- Use SHADE (SPARC simulator) to extract data references and cycle timing.
- Use LDVSIM to simulate the memory system.
- Use 9 benchmark programs (7 from SPEC95)
- Simulated 2×10^9 cycles.

Simulation Results:
Prediction Accuracy

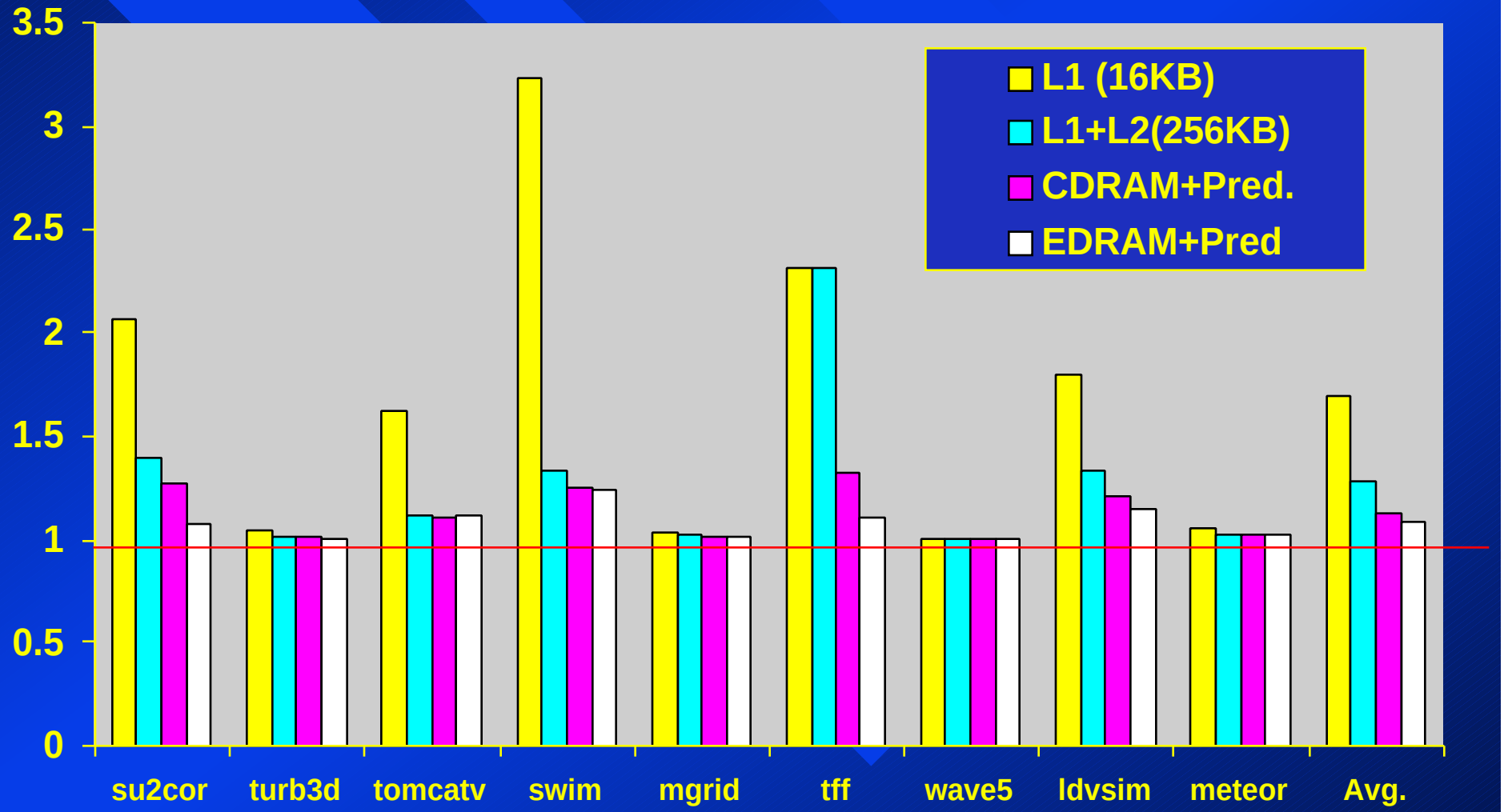


Simulation Results:
Cycles Per Memory Reference



Simulation Results:

Cycles Per Instruction



Conclusions

- Accurate predictions of CPU read requests is both possible and practical.
- Main memory built with integrated DRAM/prefetch buffers ICs
  - Use the huge bandwidth inherent in DRAM architecture to hide latency.
  - Have very low prediction miss penalty.
  - Do not consume precious bus resources.

Conclusions (cont.)

- Overall system performance improvement is obtained.
- Better performance will be achieved with DRAM ICs designed for prediction & prefetching.
- The architecture is compatible with conventional memory hierarchy.