Memroy fluctuations are common:
- Irregular parallel programs
- Parallel dynamic programs
- Computational biology
- Cloud virtual machine servers
- Database servers
- Any time-sharing system

Systems sacrifice throughput and resource utilization to avoid thrashing:
- Batch scheduling
- Static allocation
- Memory “reserves”
- These heuristics waste memory

**Example: Protein Docking**
Docking software combines
- Highly regular FFTs
- Highly irregular octree-based fast-multipole type methods

- Memory demands of each thread vary widely during execution
- Threads contend for RAM

**Theoretical Foundations:** Models of computation assume fixed memory size.

We introduce the **Cache Adaptive (CA)** model, which allows memory to change size dynamically.

**Key insight:** Cache-Oblivious (CO) algorithms are a good foundation for designing cache-adaptive algorithms.

Cache-oblivious algorithms:
- Optimal for any fixed memory size
- Many good CO algorithms: matrix multiply, sorting, FFT, all-pairs shortest path, edit distance, longest common substring

**Toolbox for cache-adaptive analysis:**
- Prove algorithms optimally adaptive
- Bound non-optimality
- Analyze adaptivity of algorithms that are not known to be optimal

**Major results:**
- Theorem characterizing optimality of many CO algorithms in CA model, or how far they are from optimal
- Optimally adaptive algorithms
- Lazy Funnel Sort
- CO Matrix Multiply
- CO Edit distance
- CO all-pairs shortest path
- and many others
- CO FFT is at most $O(\log \log N)$ slower than optimal in CA model
- Also show that LRU is still a good paging algorithm in CA model

**Applied Results:** When the size of memory fluctuates, cache adaptivity can have significant performance benefits.

**Example: Sorting**
- Cache-adaptive Lazy Funnel Sort
- Highly-optimized non-adaptive merge sort
- Memory fluctuates between 90MB and 2GB

LFS finishes faster and uses less disk bandwidth when memory changes size.

**Example: Parallel Dynamic Programs**
Many parallel computations are dynamic programs:
- Optimal chain matrix multiplication
- Optimal sequence alignment
- All-pairs shortest path

Traditionally implemented iteratively or as tiled loops.
- Not cache adaptive
- Not even very cache efficient

**Conclusion:** Cache-adaptive design principles can yield real performance benefits for irregular parallel programs.