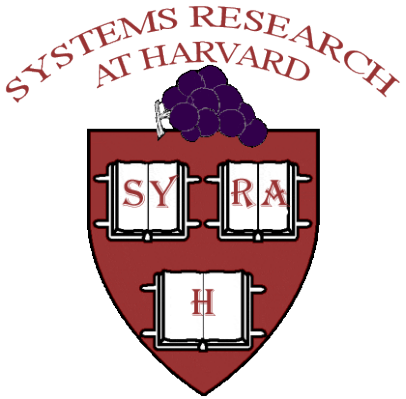
Computational Caches

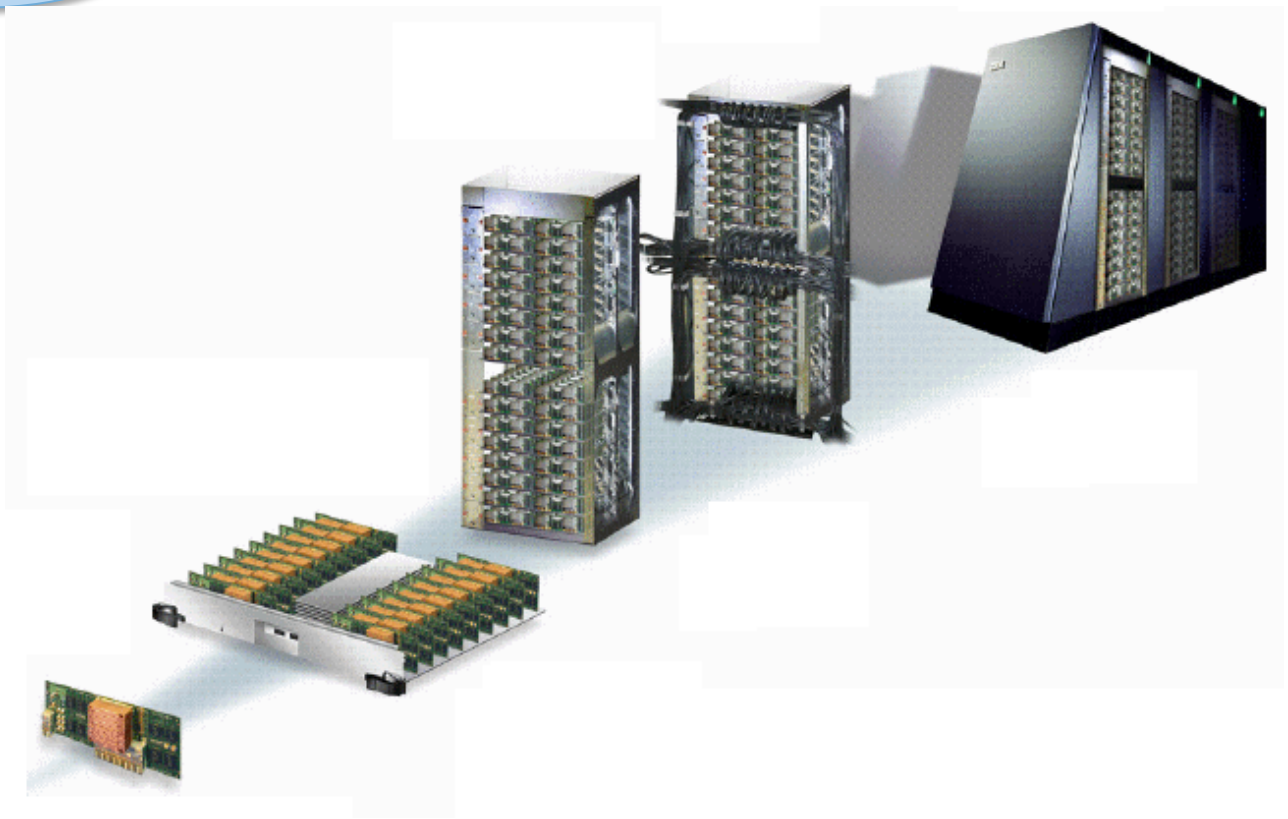


Amos Waterland,[1] Elaine Angelino,[1] Dogus Cubuk,[1]
Efthimios Kaxiras,[1] Ryan Adams,[1] Jonathan Appavoo,[2] and Margo Seltzer[1]

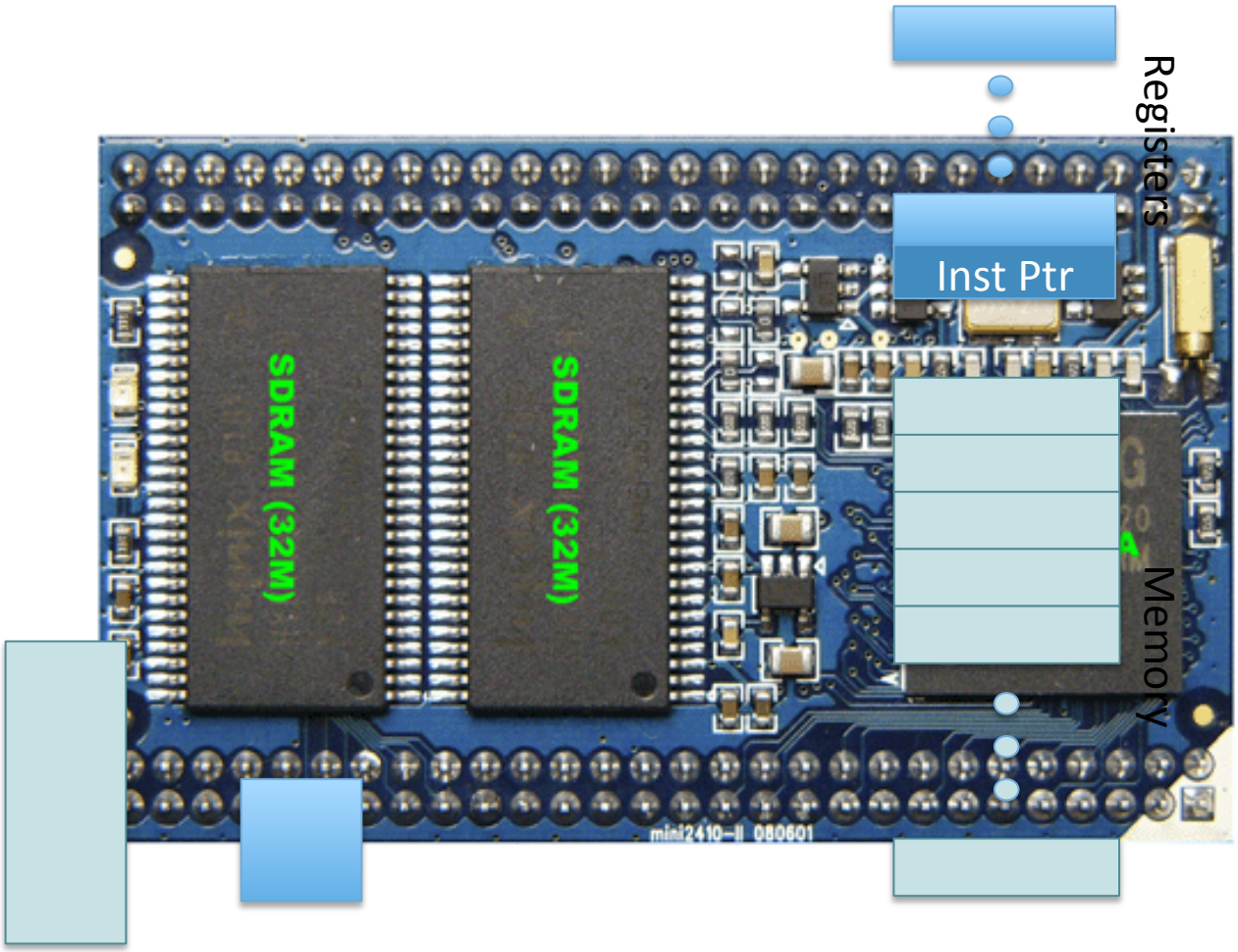
[1] Harvard University
[2] Boston University

More resources = more speedup

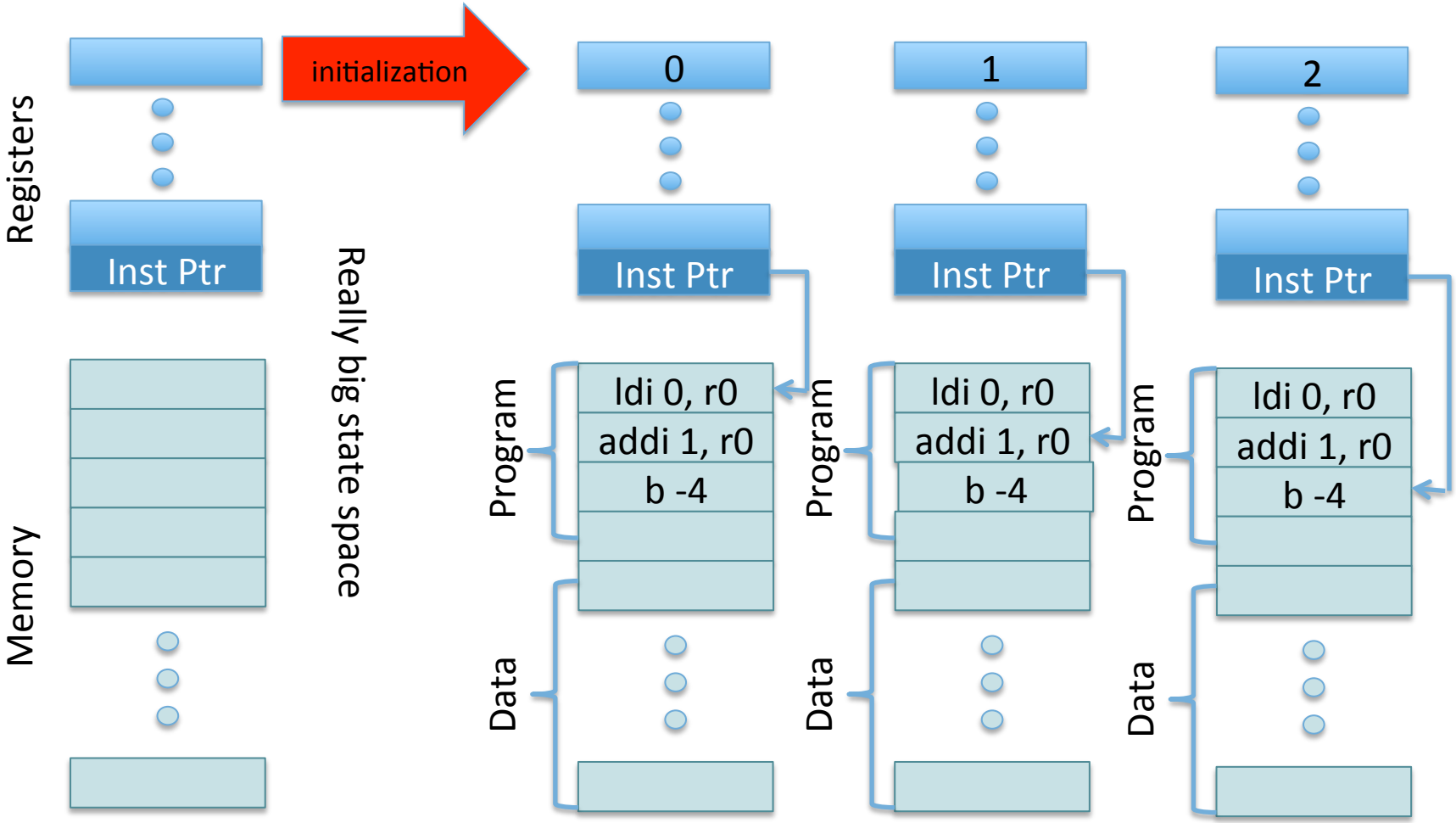
Hmmm, here's my nice *sequential* program. Sure wish I could run it on **sixteen thousand** cores.



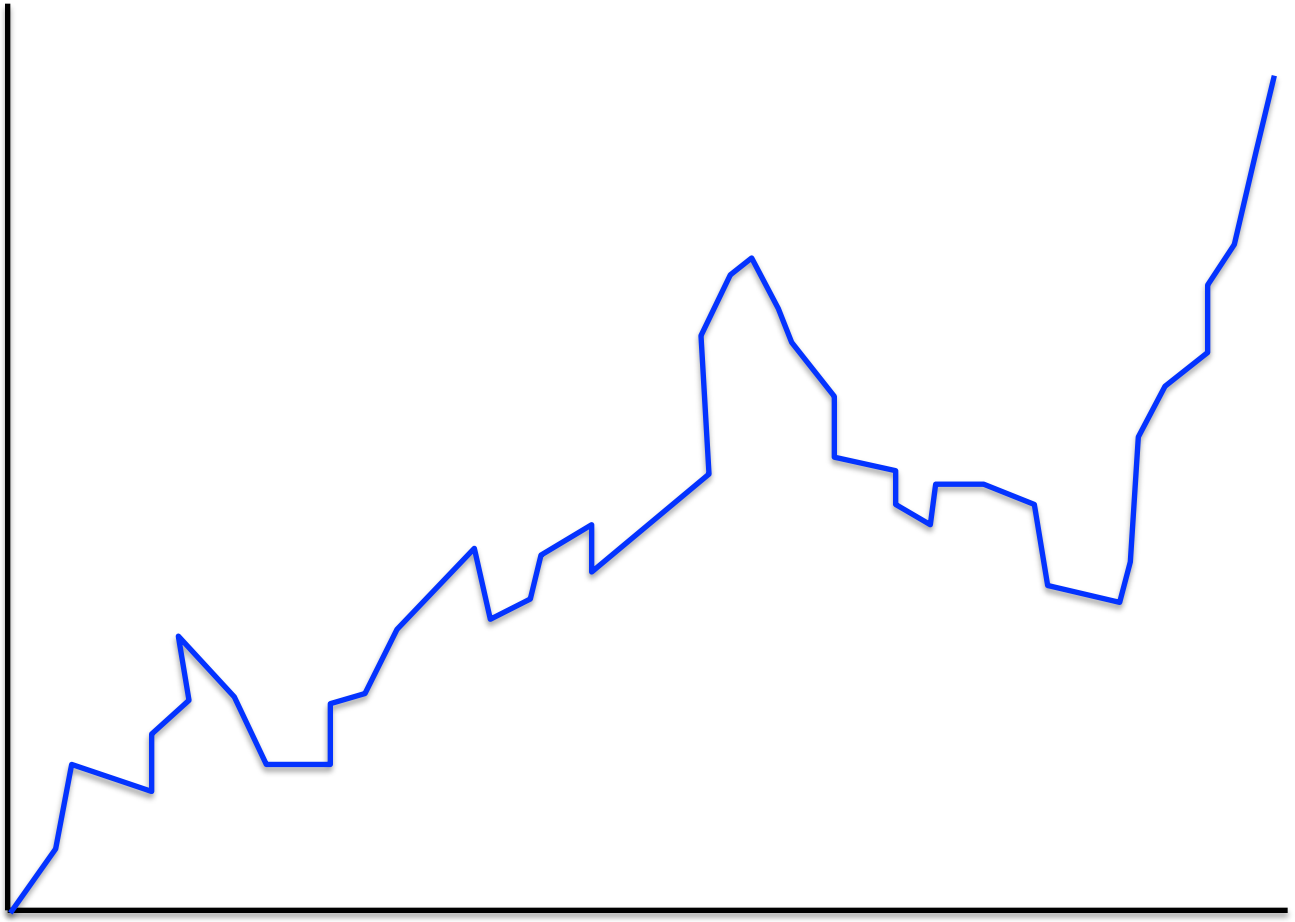
Join me in a thought experiment…



Execution in a Really Big State Space

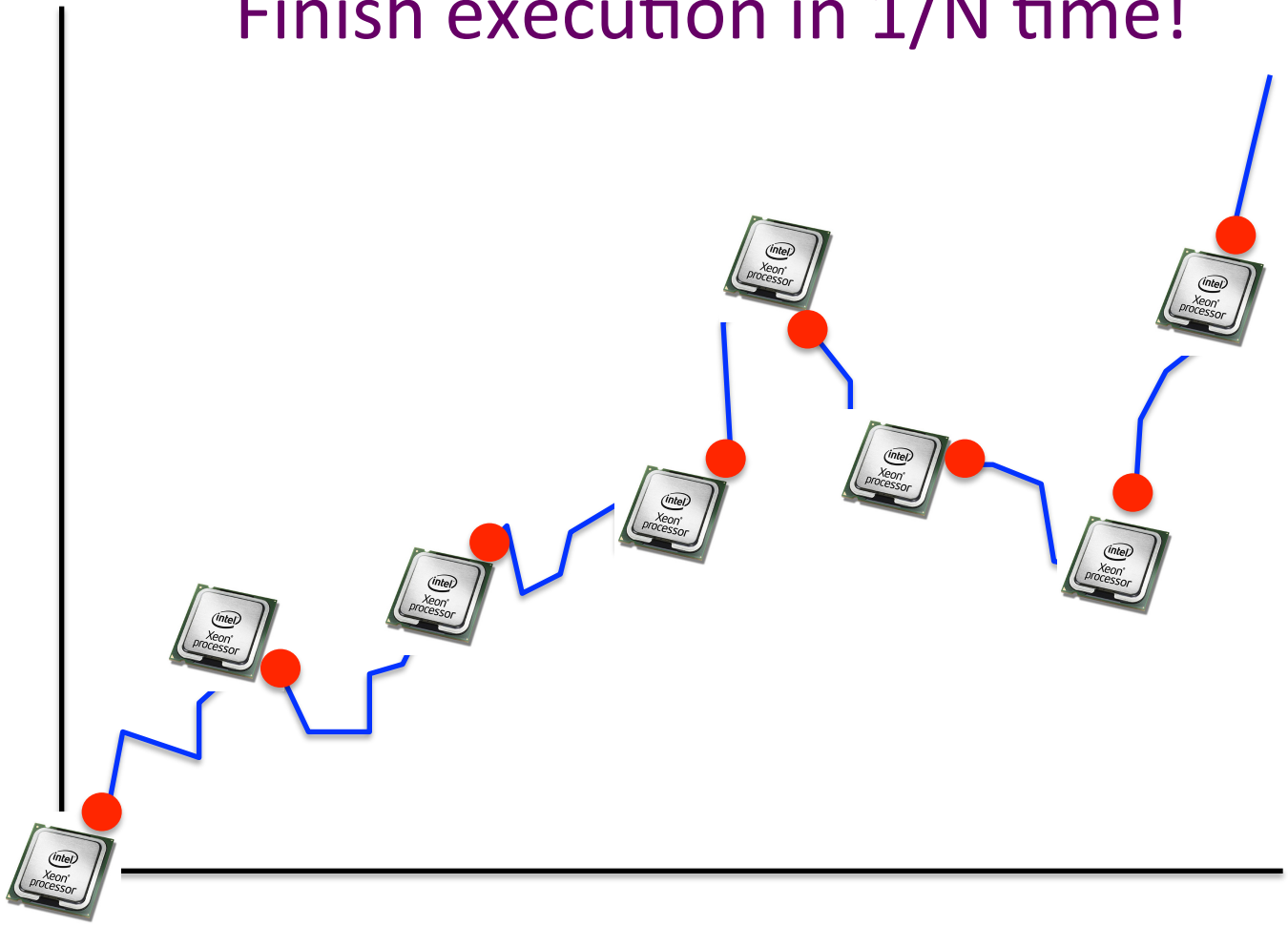


Trajectory-Based Execution

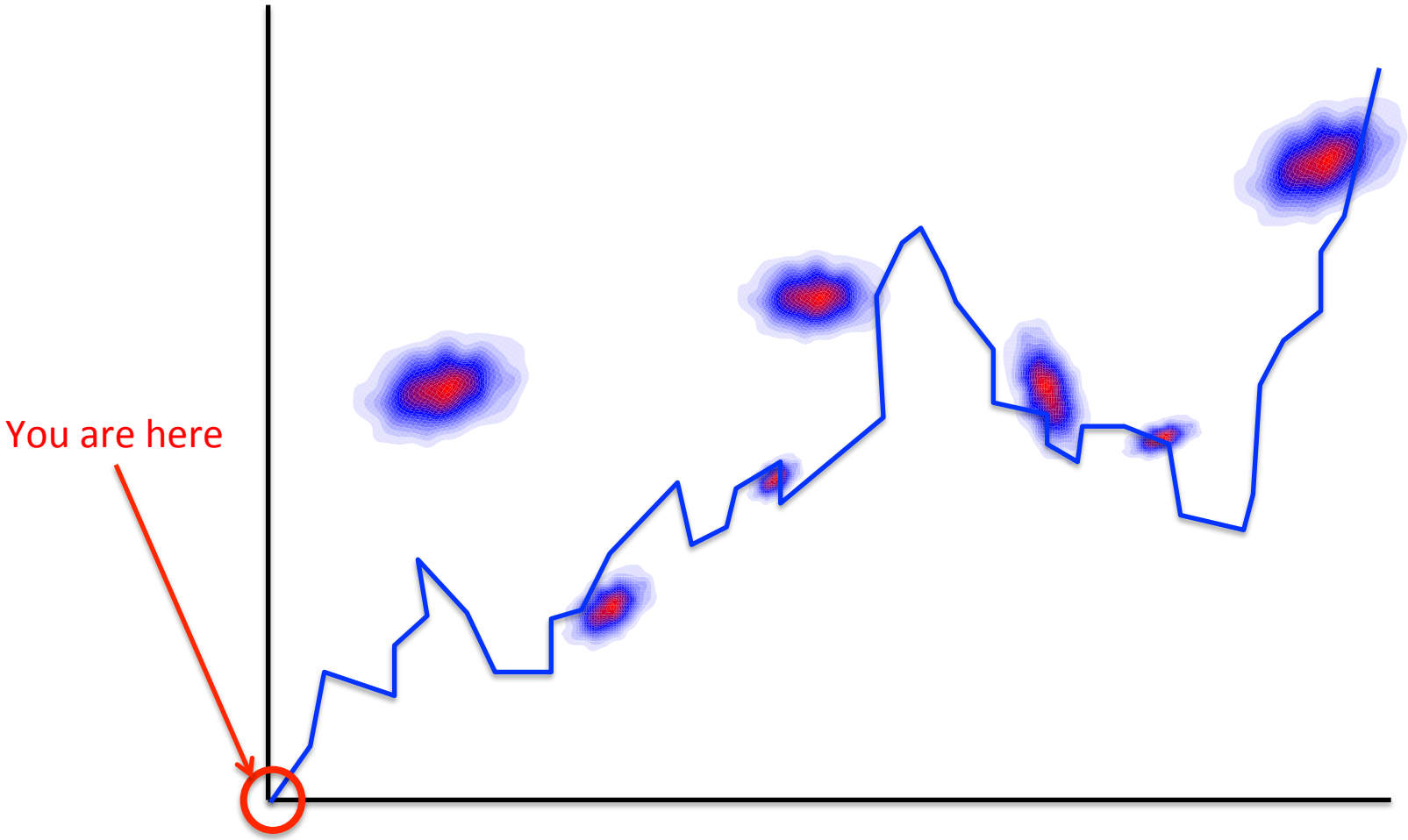


Parallel Trajectory-Based Execution

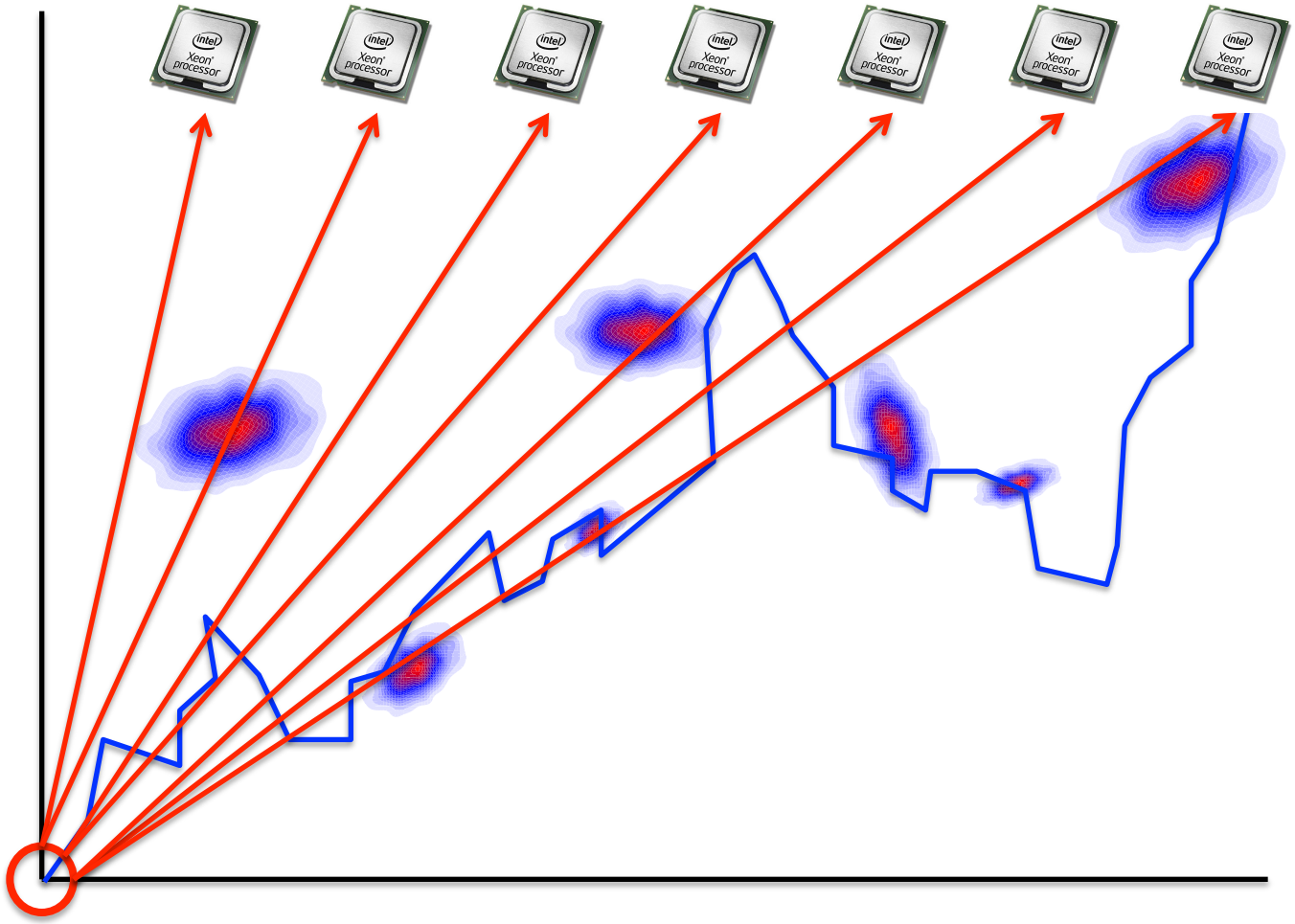
Finish execution in 1/N time!



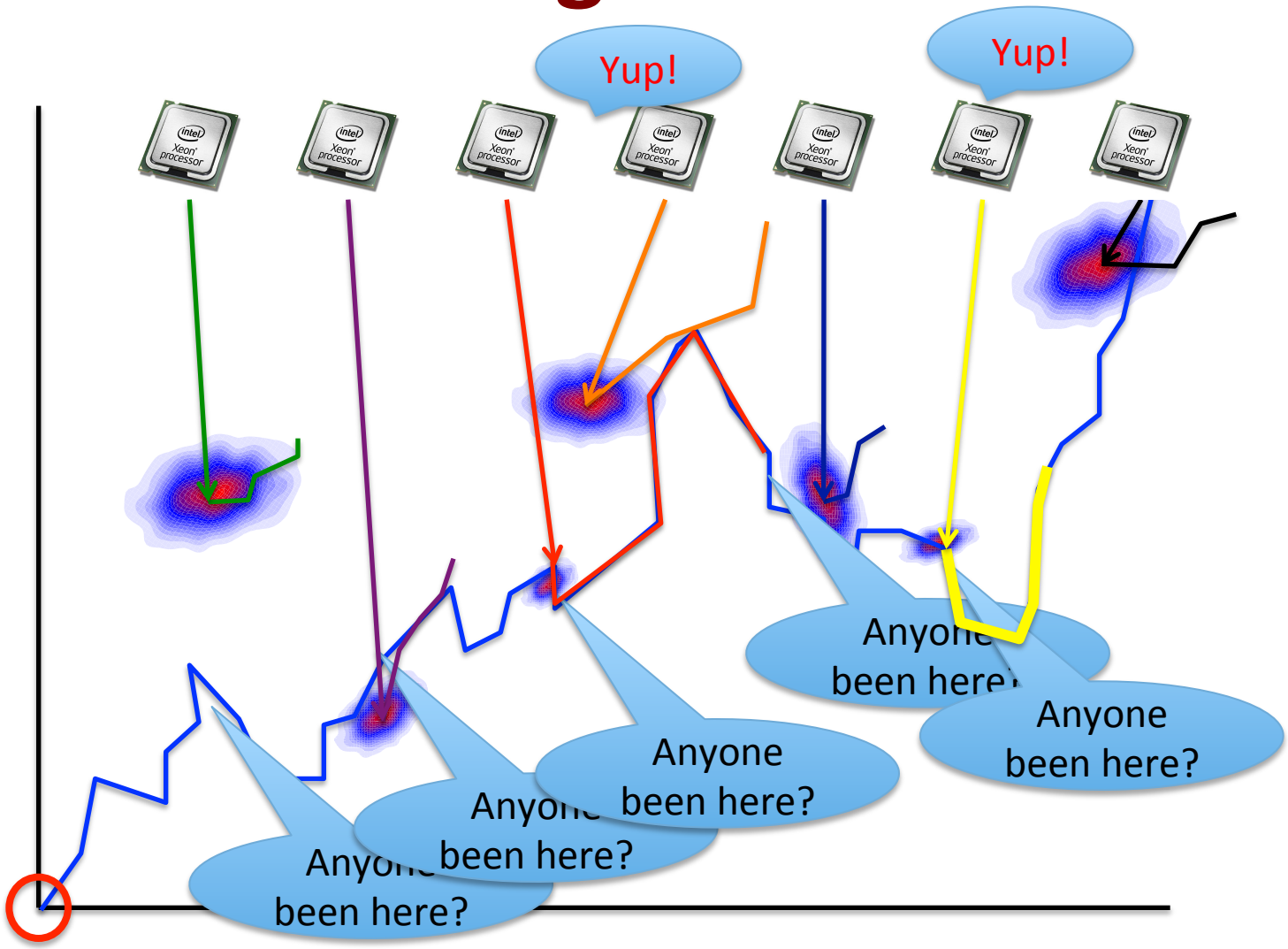
Oracle approximation



Prefetching execution

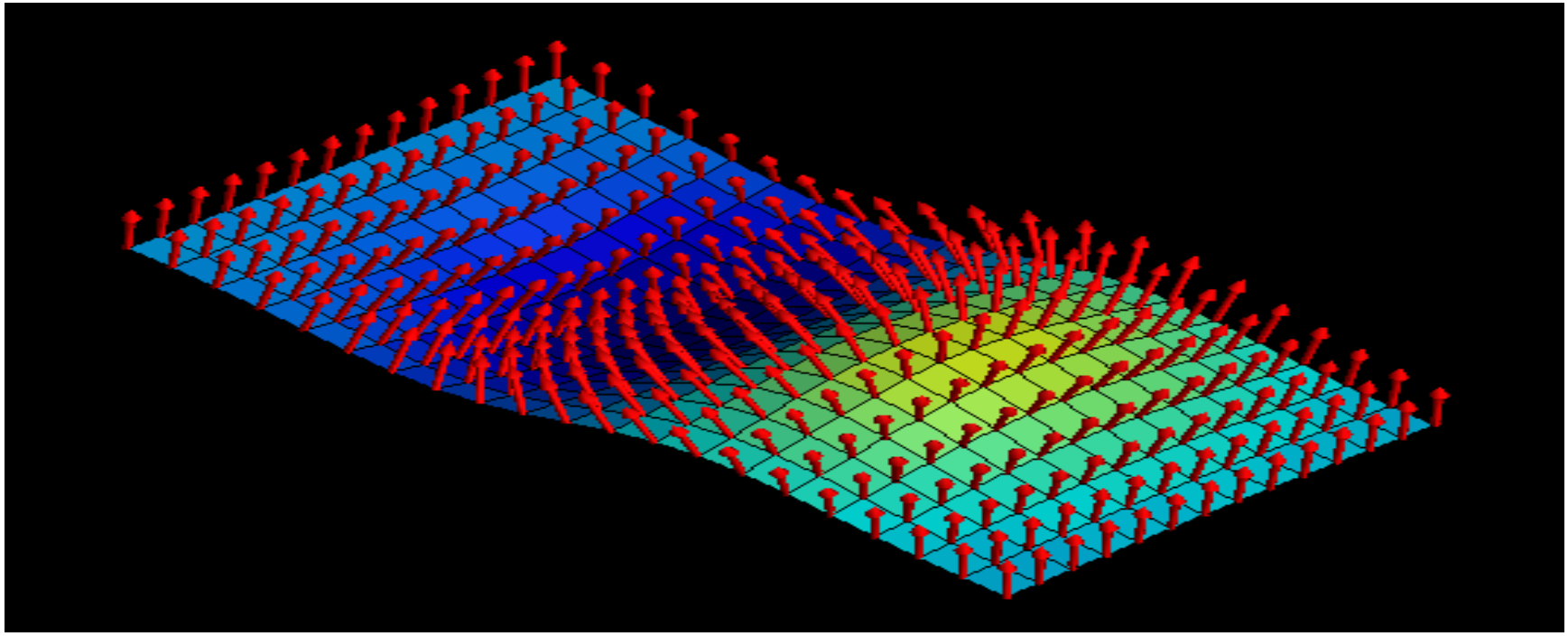


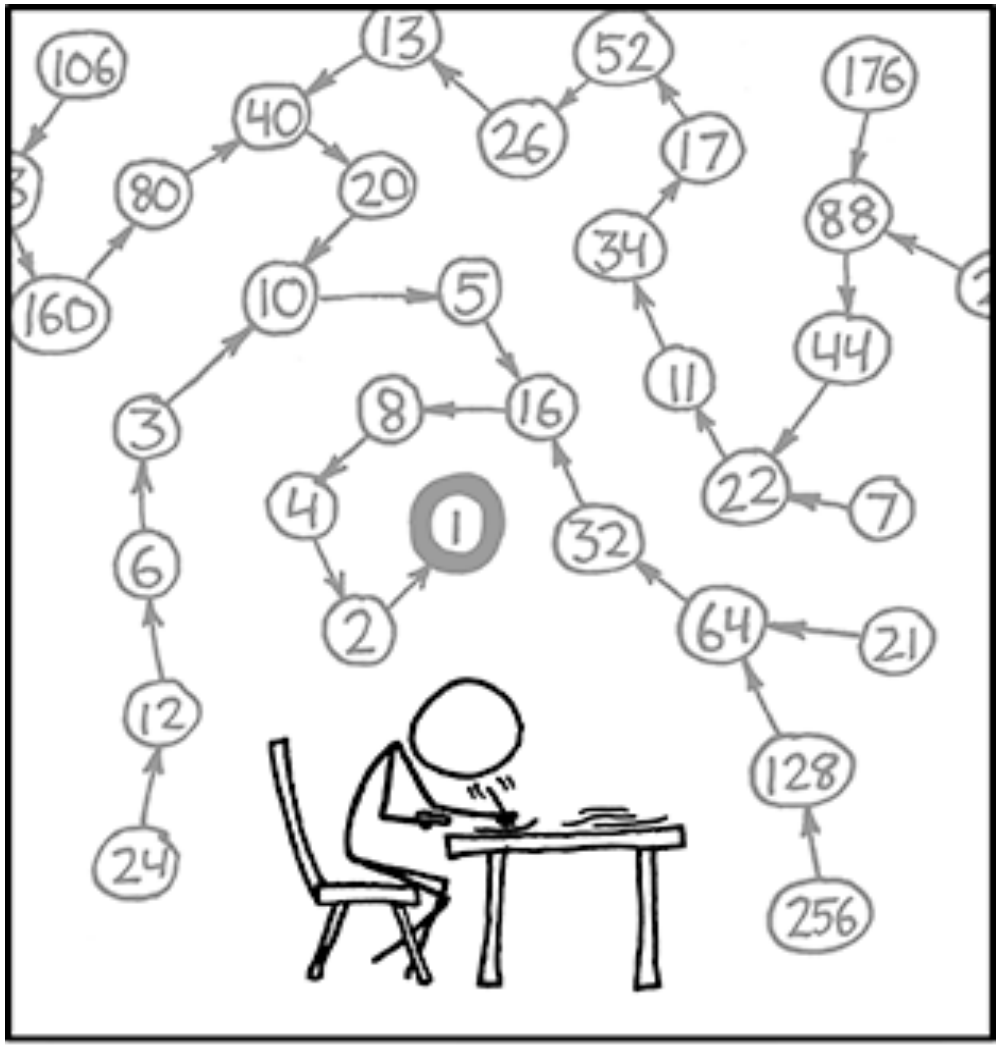
Prefetching execution



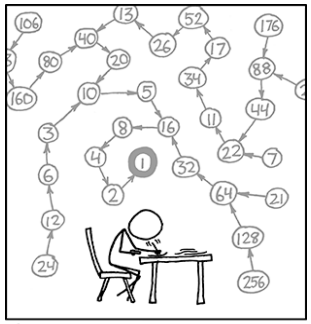
Generalized, speculative memoization

- Cache arbitrary sequences of execution.
- "Prefetch" entries with speculative execution.
- Symmetries play a crucial role:





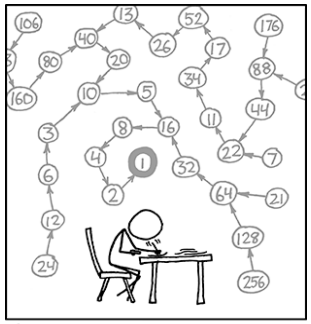
THE COLLATZ CONJECTURE STATES THAT IF YOU PICK A NUMBER, AND IF IT'S EVEN DIVIDE IT BY TWO AND IF IT'S ODD MULTIPLY IT BY THREE AND ADD ONE, AND YOU REPEAT THIS PROCEDURE LONG ENOUGH, EVENTUALLY YOUR FRIENDS WILL STOP CALLING TO SEE IF YOU WANT TO HANG OUT.



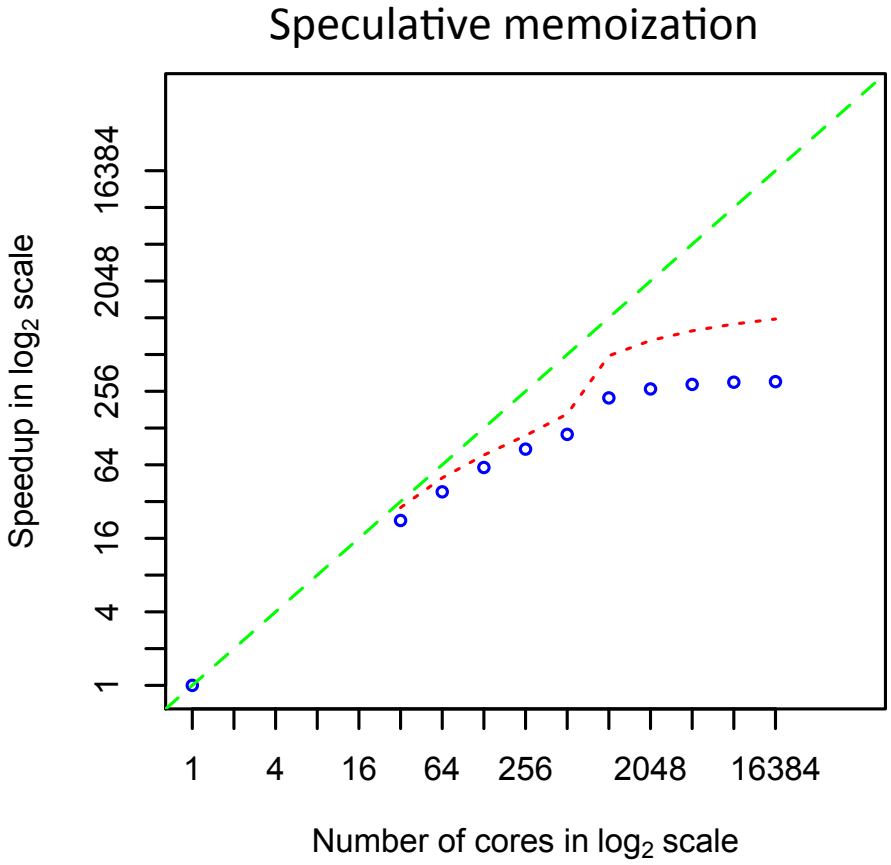
Source code

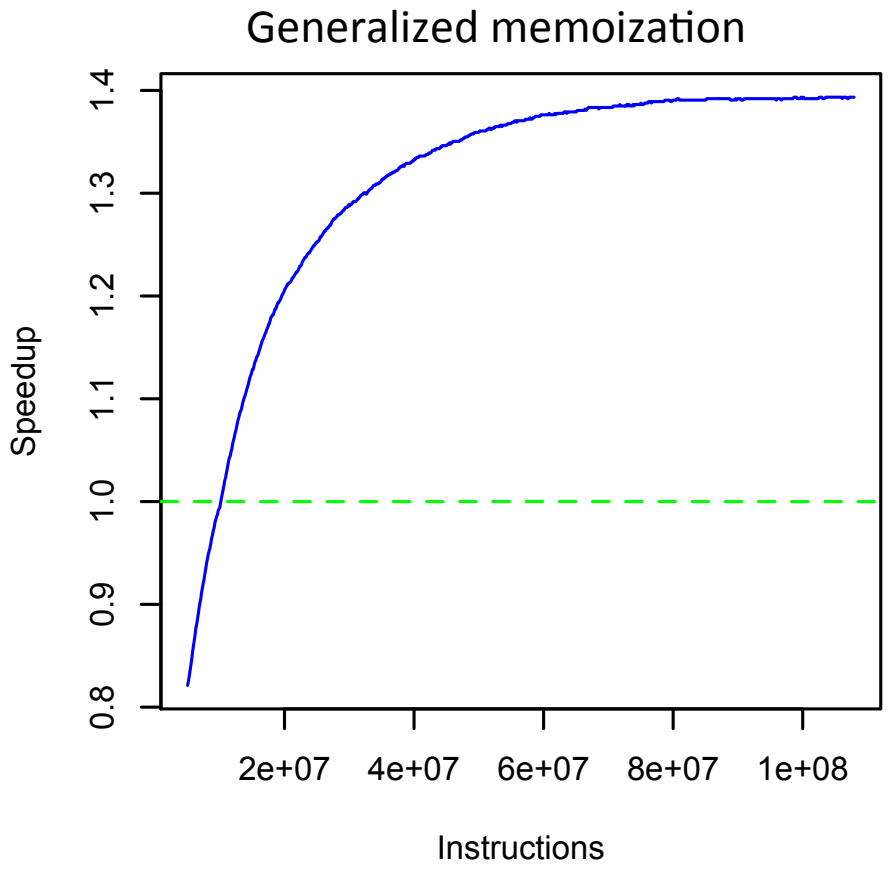
apw — 53×17 — ⌘1

```
int main()
{
    unsigned int i, j;

    for (i = 1; i < 100000000; i++) {
        for (j = i; j > 1; ) {
            if (j % 2 == 0)
                j = j / 2;
            else
                j = 3 * j + 1;
        }
    }

    return i;
}
~
"collatz.c" 15L, 242C
```



Observed speedups





Conclusions

1) Automatic speedup by caching computation, including future computation.

2) Amplify trajectories into equivalence classes using don't care bits.

3) Adaptively learn the structure of trajectories to allow efficient cache fill and query.

Speedup = adaptive learning + cache search.

Supported by:

U.S. DEPARTMENT OF ENERGY

NATIONAL SCIENCE FOUNDATION NSF