AMOS and UPC

Phil Merkey
Michigan Tech

Definition

A memory operation is atomic in case the only observable states for the memory are the state before the operation began or the state after the operation has completed.

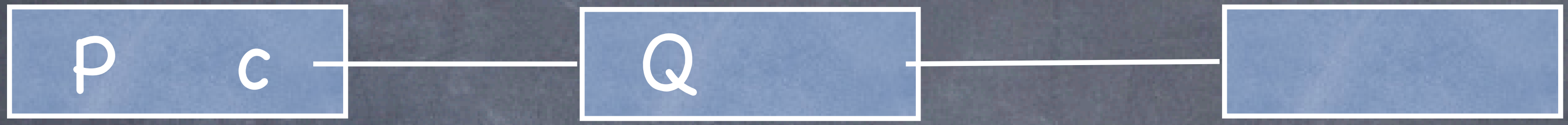
It is as if it happens at the memory location.

Definition

- Link Load / Store Conditional
- Fetch_and_Add afadd(&p, v)
  - return p, set p = p + v
- Compare and Swap acswap(&p, cmp, replace)
  - return p, if(cmp==p) { p = replace }

Link Lists

Standard use of compare and swap



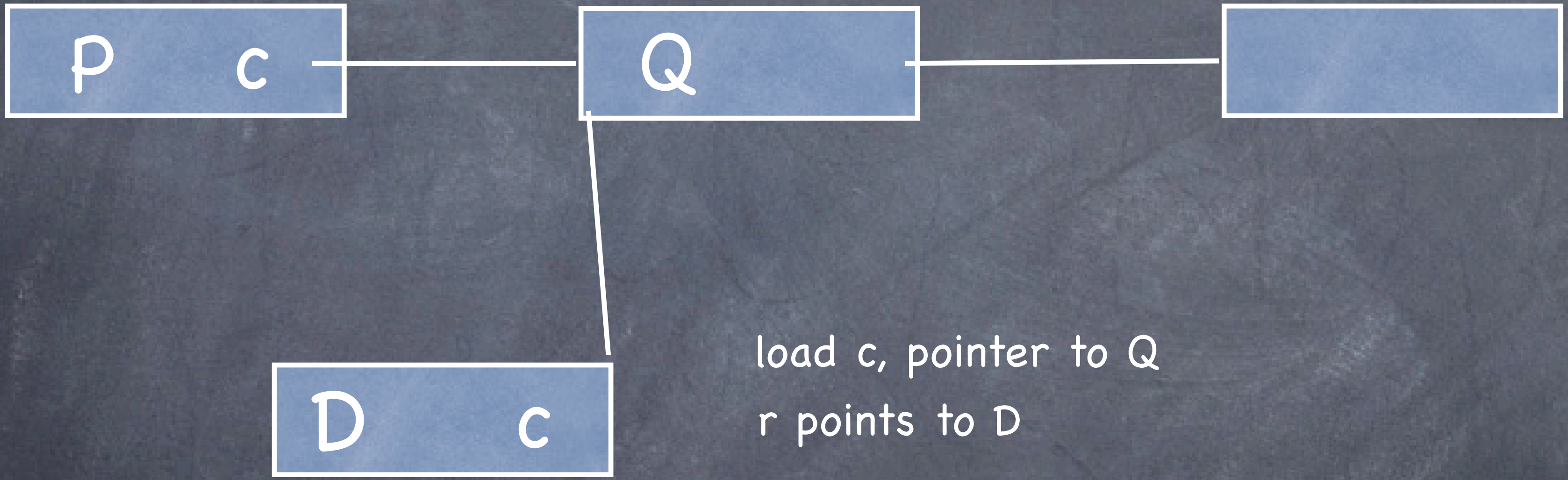
Link Lists

Standard use of compare and swap

load c, pointer to Q

Link Lists

Standard use of compare and swap

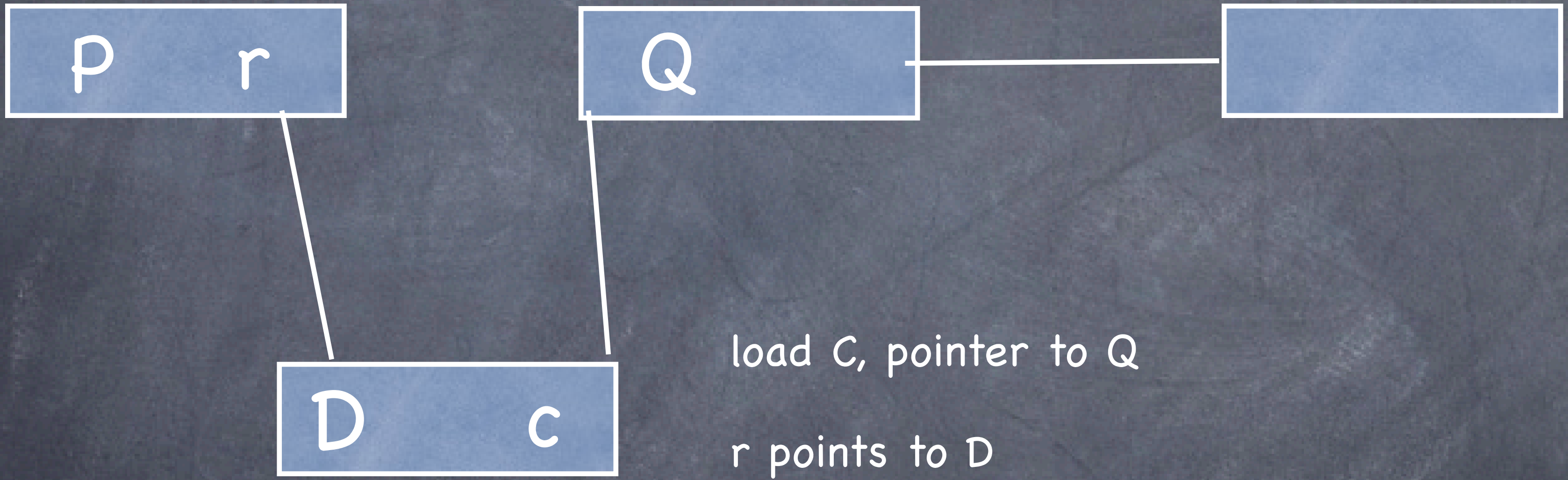


load c, pointer to Q

r points to D

Link Lists

Standard use of compare and swap



load C, pointer to Q

r points to D

cswap(&P, c, r)

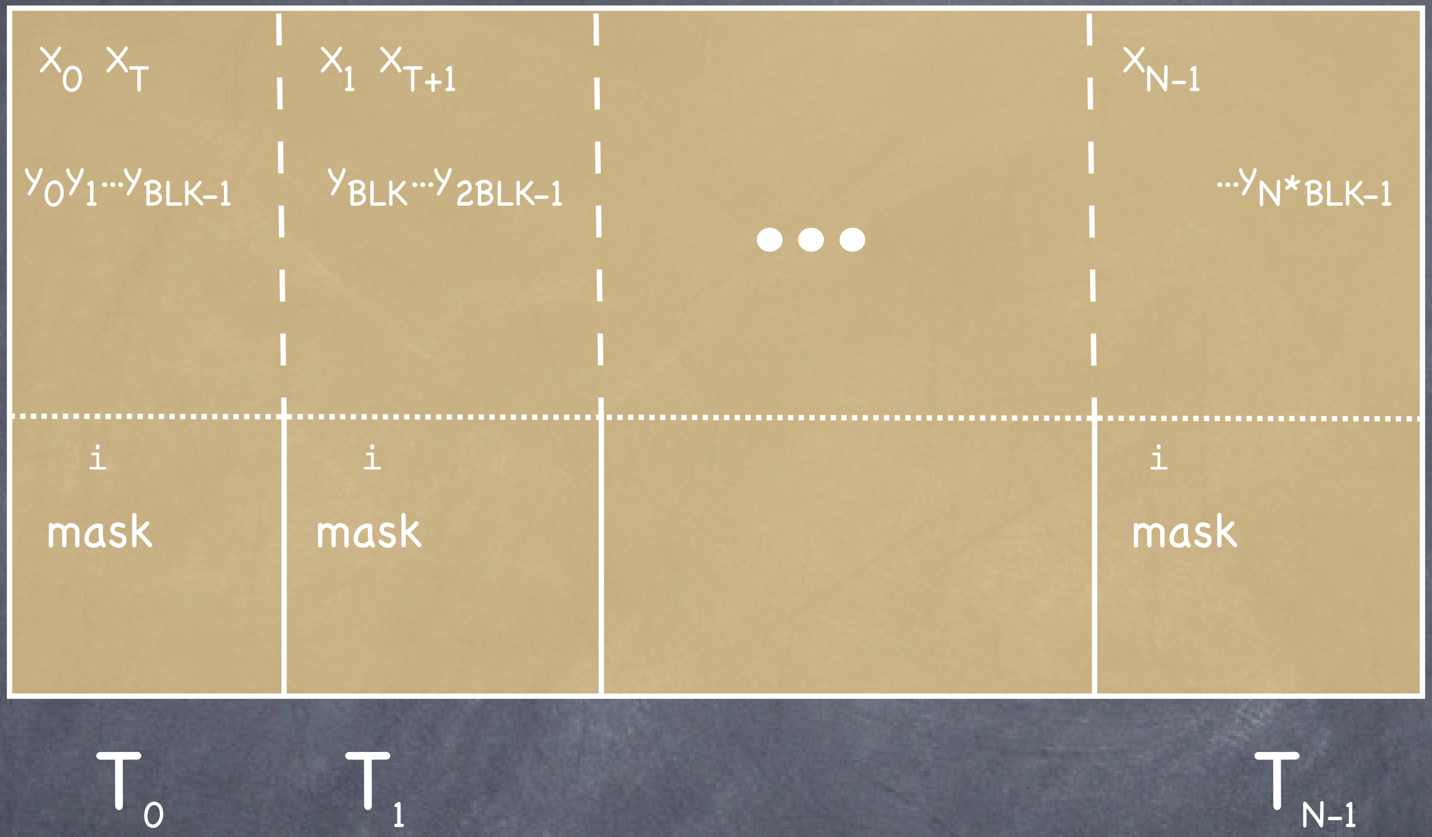
UPC machines with AMOS

- Cray T3E: fetch_and_add(&p,v)
- Cray X1:
  - _amo_aadd(&p, v) //atomic add (long v)
  - _amo_afadd(&p,v) // fetch and add
  - _amo_aax(&p, A, X) // p=p&A^X
  - _amo_afax(&p,A,X)
  - _amo_acswap(&p, c, r) //compare and swap

UPC machines with AMOS

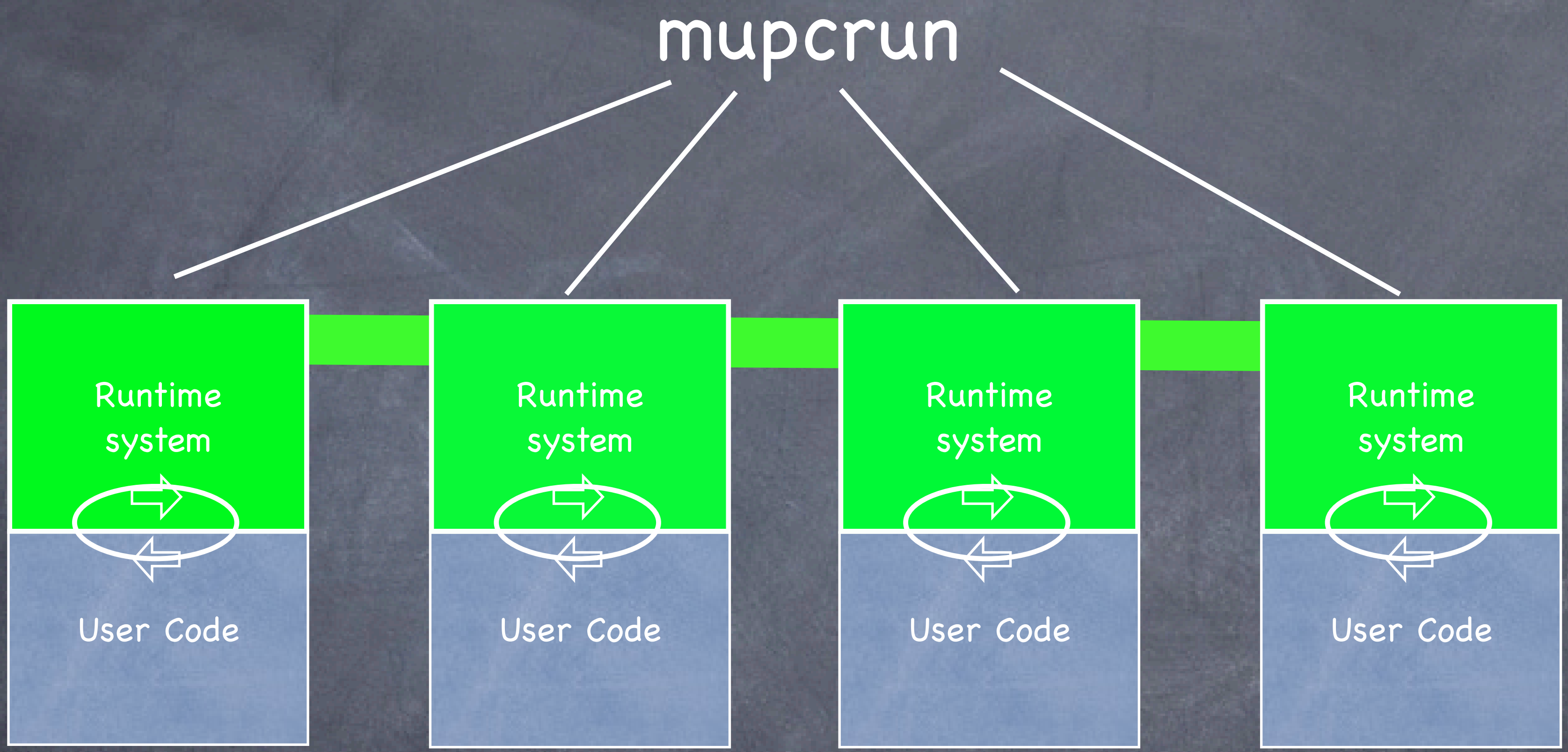
- MuPC:
  - _upc_faop(opcode, &p, v)
  - _upc_cas(&p, &c, &r)
  - _upc_dcas(&p, &c1, &r2, &c2, &r2)
  - _upc_maskswap(&p, v, mask)
  - _upc_ffaop(opcode, &p, v)

Affinity and Blocking



```
shared [1] float x[MaxN];
shared [BLK] float y[MaxN];
```

MuPC



mupcrun

Runtime system

User Code

Runtime system

User Code

Runtime system

User Code

Runtime system

User Code

Why not just use locks?

```
old = _amo_cswap(&A[k], C, R);
```

Is sort of equivalent to:

```
_upc_lock( lockforA[k]);
    old = A[k];
    if(C==old)
        A[k] = R;
_upc_unlock( lockforA[k]);
```

But there are issues.

Why not just use locks?

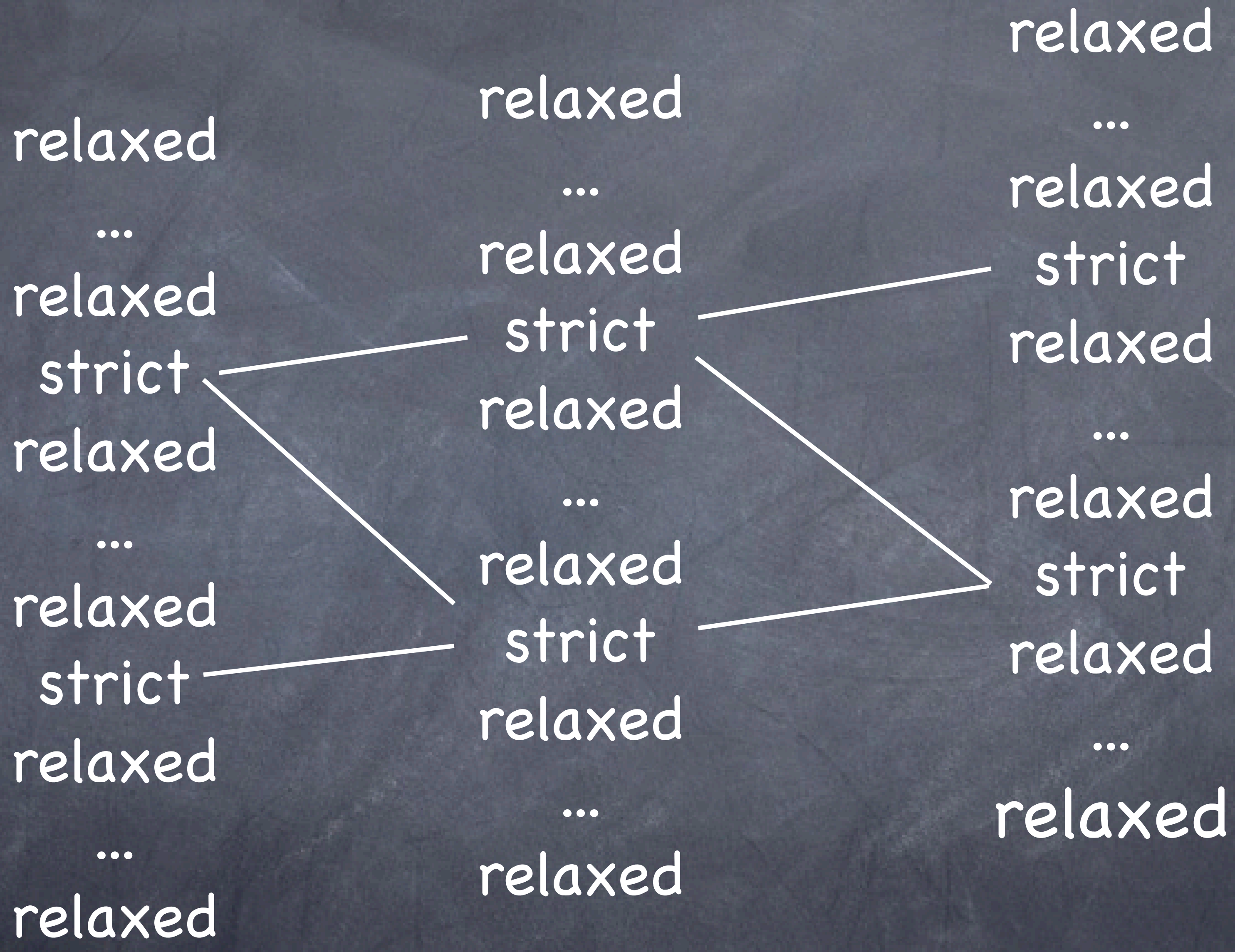
- Locks are ugly and not cool
- You need a lock for each element of A[]
    - Or a lock per thread if you can partition the writes by thread
    - Or some other way to hash into a table of locks
- There are issues with the memory model.

Memory Model

Every memory reference is either strict or relaxed.

- `strict` ~= sequential consistency
  "every thread sees this memory operation in program order wrt to other strict references"
- `relaxed` ~= weak consistency
  "just a C program within a thread"
- Important to implementors---legal issue
- Performance issues
- Interaction with I/O and Collectives and AMOs

Memory Models



relaxed
…
relaxed
strict
relaxed
…
relaxed
strict
relaxed
…
relaxed

relaxed
…
relaxed
strict
relaxed
…
relaxed
strict
relaxed
…
relaxed

relaxed
…
relaxed
strict
relaxed
…
relaxed
strict
relaxed
…
relaxed

Why not just use locks?

```
old = _amo_cswap(&A[k], C, R);
```

Is sort of equivalent to:

```
strict null reference
_upc_lock( lockforA[k]);
    old = A[k];
    if(C==old)
            A[k] = R;
_upc_unlock( lockforA[k]);
strict null reference
```

Relative Costs

No Head-to-Head Comparison is Meaningful

- T3E
  - afadd == aadd
  - upc_lock is done with afadd
  - a afadd about the same as an add
- X1
  - amos don't vectorize
  - aadd twice as fast as afadd
  - don't know how upc_lock works
- Beowulf (think simulator)
  - the current caching strategies are ⊥ amos

Relative Costs

Pick a modest number of threads 16-64

For array sizes from 1 to 1024

```
loop                                    // baseline
   array[ random ]++
-----------
loop                                    // atomic
   afaad(&array[ random ] , 1)
-----------
loop                                    // locks
   upc_lock( lockarray[ random ] )
   array[random]++
   upc_unlock( lockarray[ random ] )
```

Relative Costs

T3E:

buckets	updates raw +=1	slow down w/ locks
1024	98%	3x
32	50%	5x
1	20%	20x

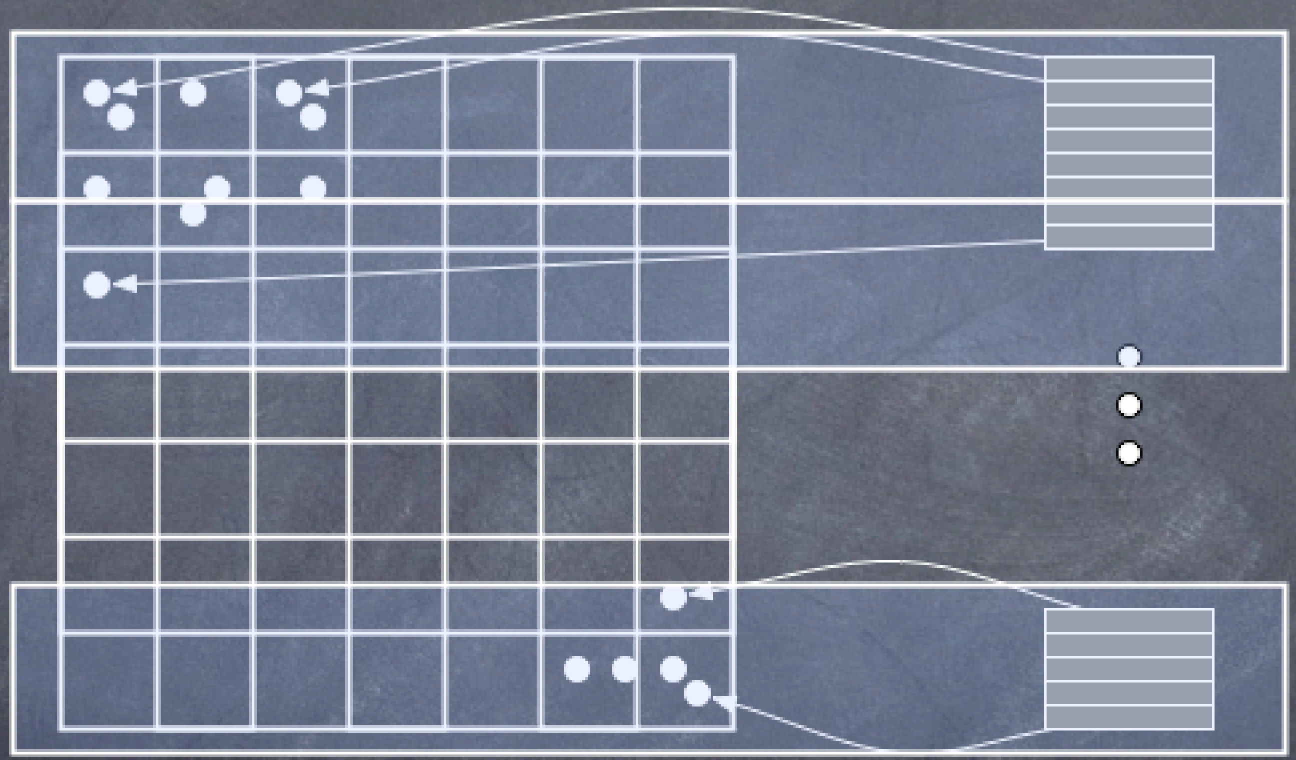
afadd is 2x over raw +=

X1:

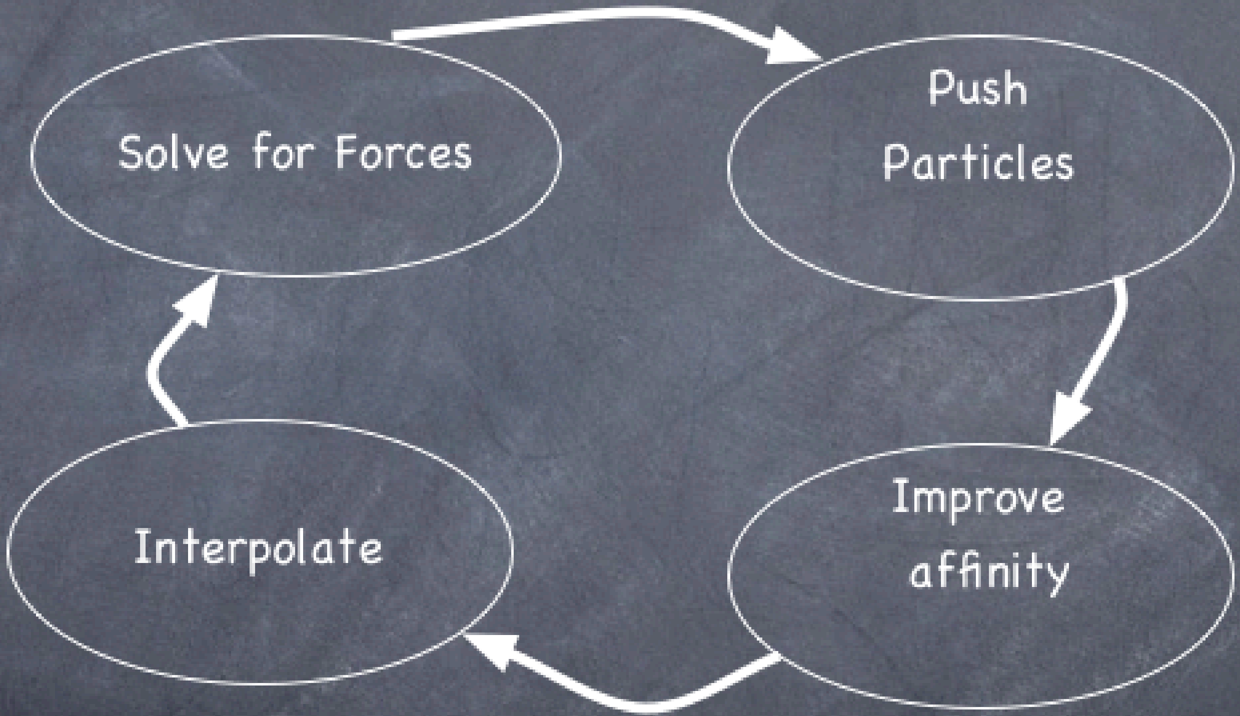
buckets	updates raw +=1	slow down w/ locks
1024	98%	3x
32	50%	5x
1	20%	100x

amos don't vectorize, afadd is 2x over aadd

PIC Code



PIC Code



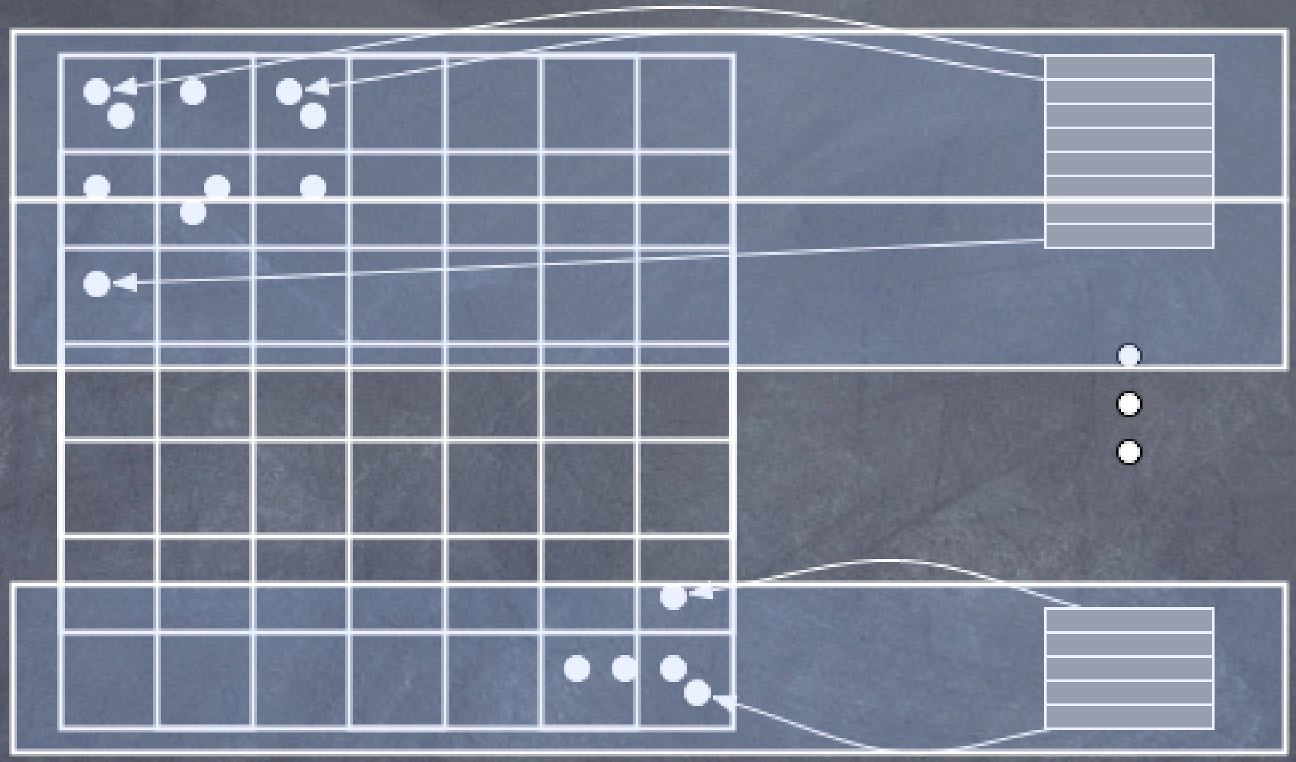
Solve for Forces

Push Particles

Improve affinity

Interpolate

PIC Code



As the particles move their affinity changes

Improving Affinity

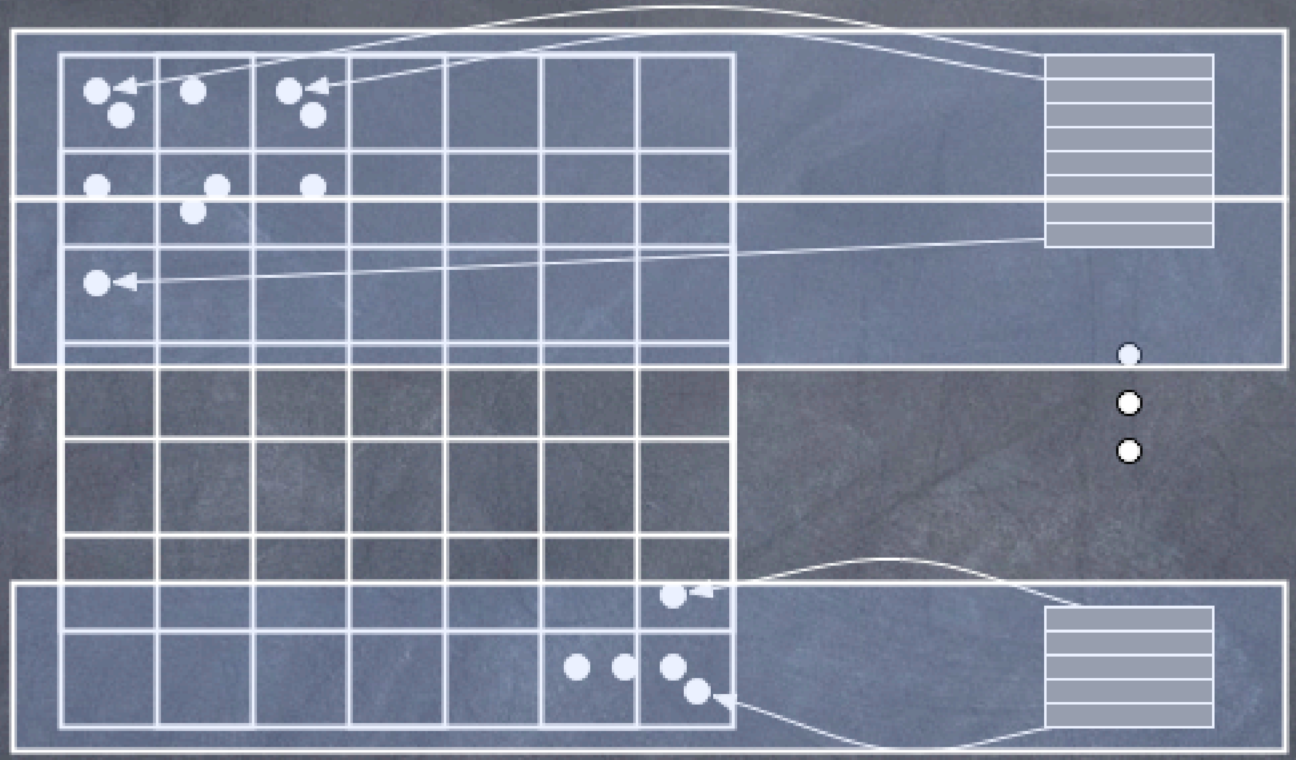
n_T = desired affinity of particle new position

newindex[0:THREADS-1] = counter for each thread

```
for all my particles, P
    n_T = desired affinity of P
    n_idx[P] = amo_afadd( &newindex[n_T], 1 )
    if out of bounds, work harder

for all my particles, P
    move P to new home
```

PIC Code



Each particle contributes (mass or charge) to the cell that contains it.

Interpolation

Fract(corner, particle) = the contribution to said corner

```
for each particle, P
    ffaop( Op, Grid[nw], Fract(nw, P) )
    ffaop( Op, Grid[ne], Fract(ne, P) )
    ffaop( Op, Grid[sw], Fract(sw, P) )
    ffaop( Op, Grid[se], Fract(se, P) )

barrier
```

Announcement

MuPC's atomic floating point fetch and add is the fastest on the planet.