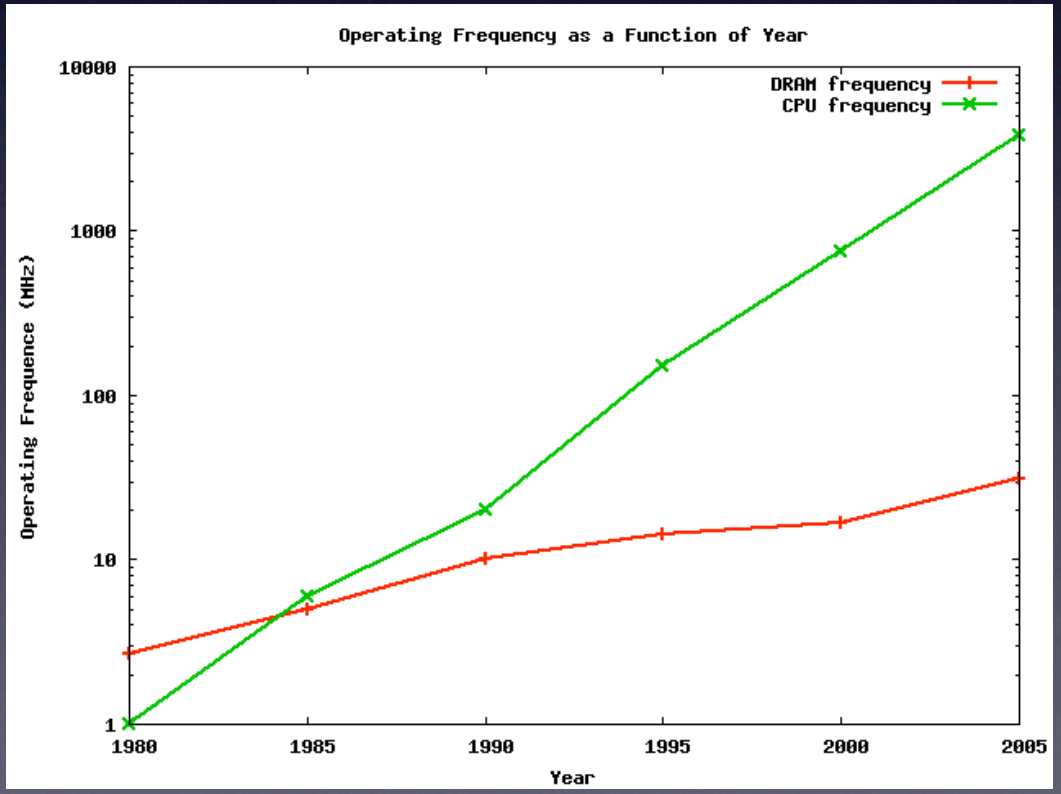
Interactive Visualization for Memory Reference Traces

A.N.M. Imroz Choudhury
Kristin C. Potter
Steven G. Parker

Trends in Computing

Year	1980	1985	1990	1995	2000	2005
DRAM access time (MHz)	2.7	5	10	14.3	16.7	31.3
CPU clock rate (MHz)	1	6	20	150	750	3800

- DRAM speedup: 11.6x
- CPU speedup: 3800x

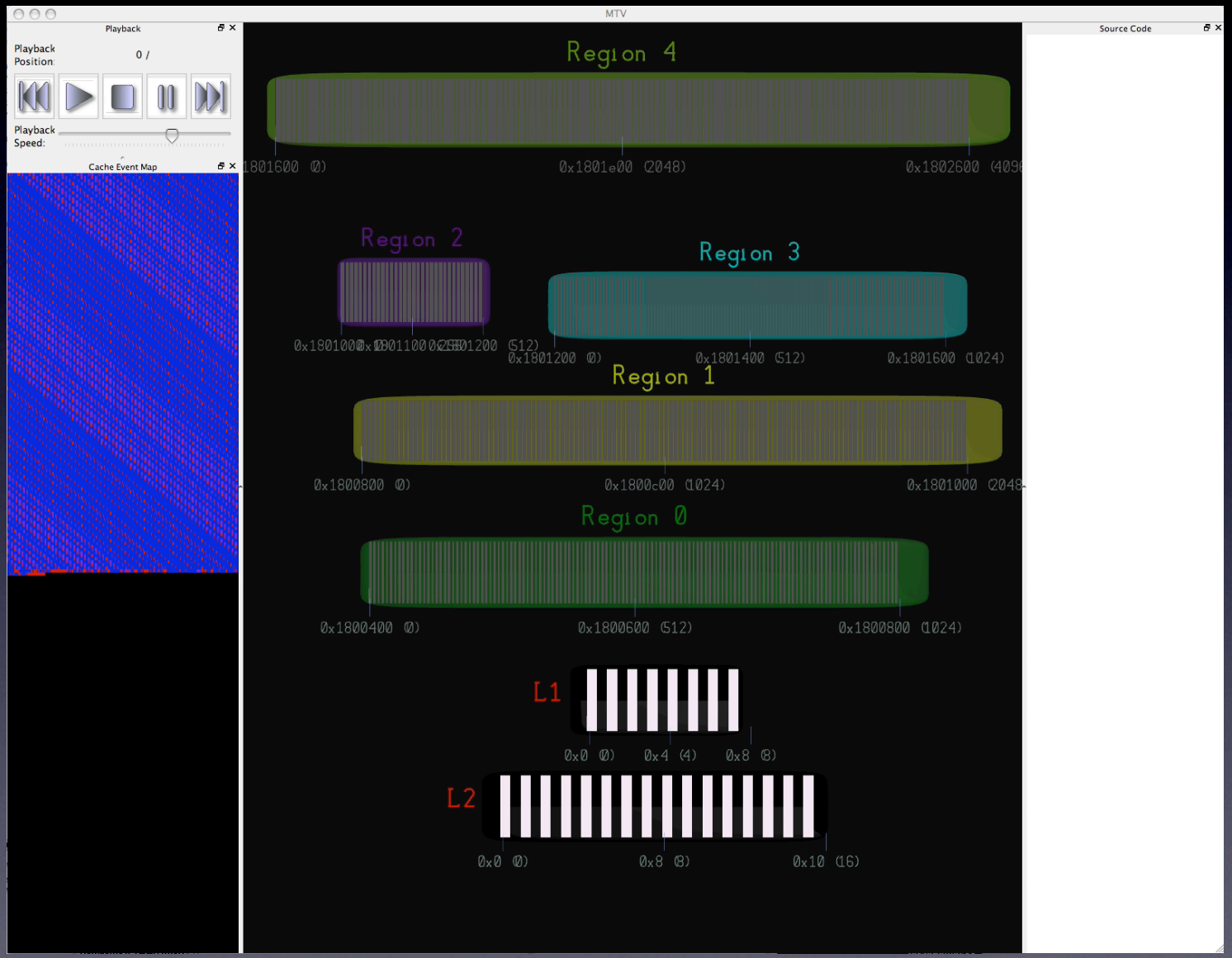


Reference Traces

- List of all memory references made by a program as it runs
- Tools: Pin, CHUD (`amber, acid`)
- Clean, full abstraction of memory interaction
- But...hard to understand!

```
R 2800600 W 2800400 W 2800600
W 2800608 W 2800408 W 2800608
W 2800610 R 2800410 W 2800610
R 2800618 W 2800418 W 2800618
R 2800620 R 2800420 W 2800620
```

Memory Trace Visualizer (MTV)

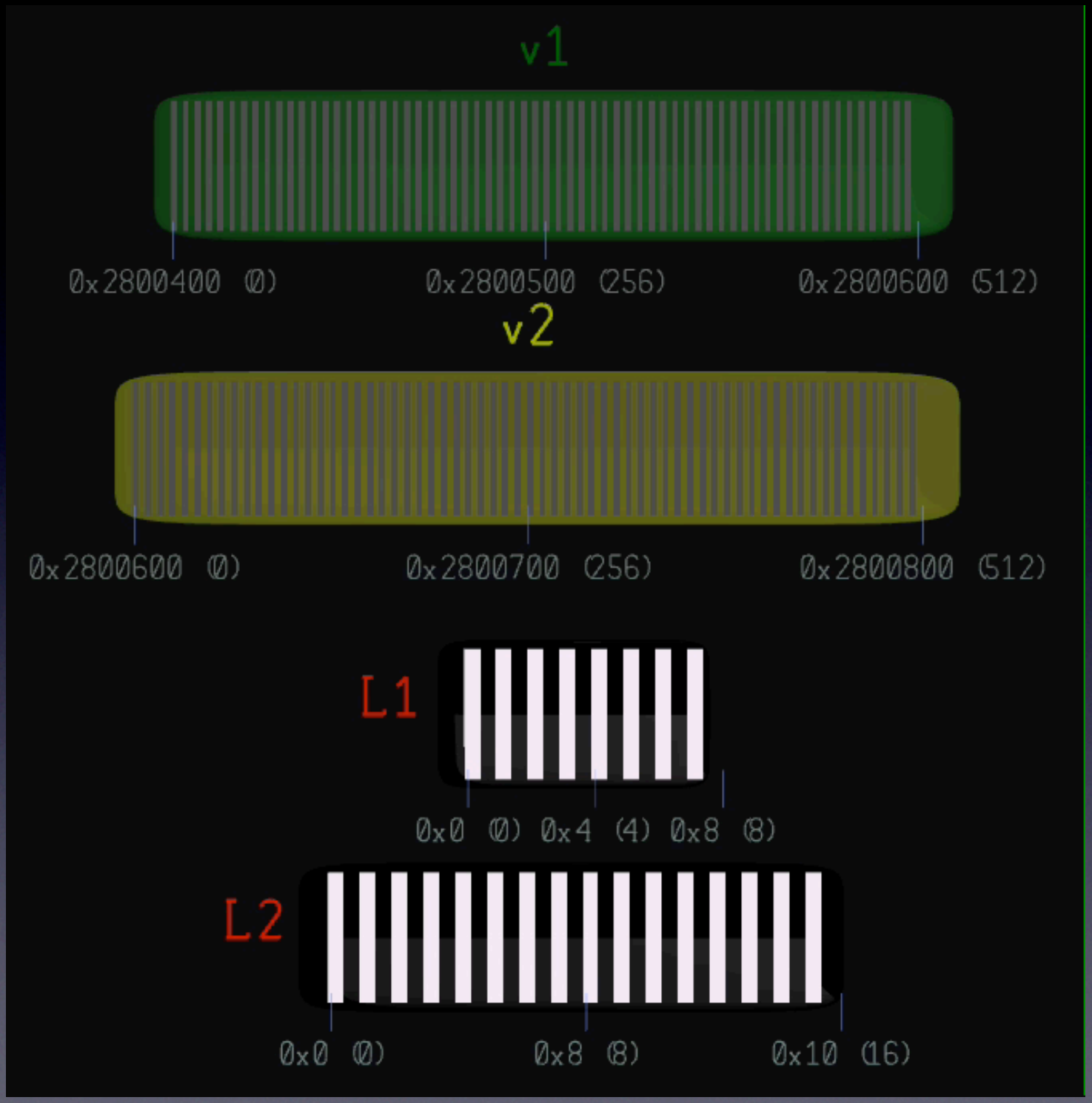


Data Structures

- Interesting memory regions are registered
- MTV creates on-screen glyphs, each with unique color
- Trace is processed ⟶ glyphs light up
  - Orange = write, Cyan = read
  - Red = cache miss, Blue = cache hit
- Colors fade with passage of time
- Common patterns are easily seen

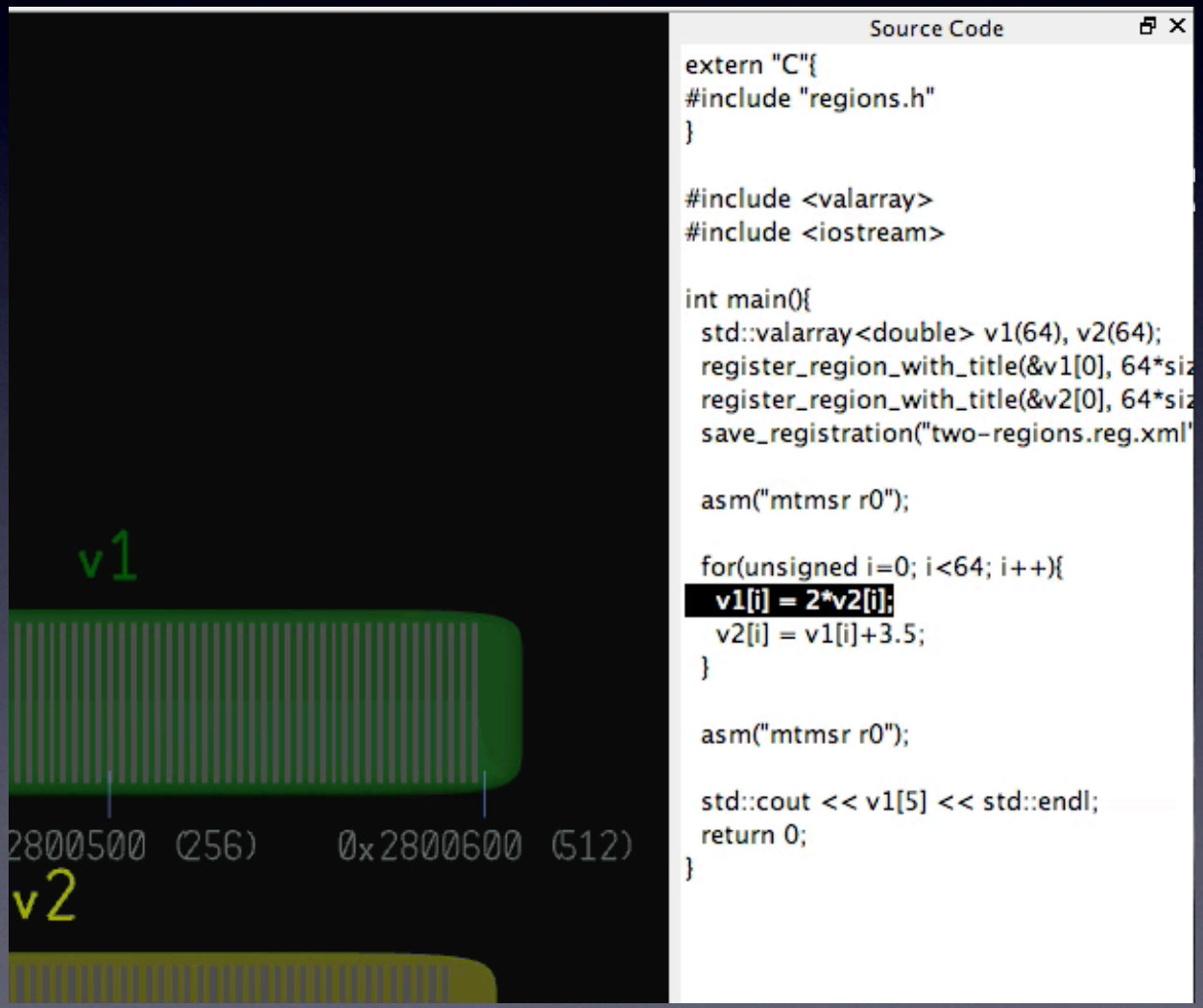
Cache

- Cache simulation with each reference record
- Residency indicated by region colors
- Hit level indicated by red/purple/blue
- Shell color indicates “temperature”
- “Laser” lines connect region events with cache events


v1
0x2800400 (0)
0x2800500 (256)
0x2800600 (512)
v2
0x2800600 (0)
0x2800700 (256)
0x2800800 (512)
L1
0x0 (0)
0x4 (4)
0x8 (8)
L2
0x0 (0)
0x8 (8)
0x10 (16)

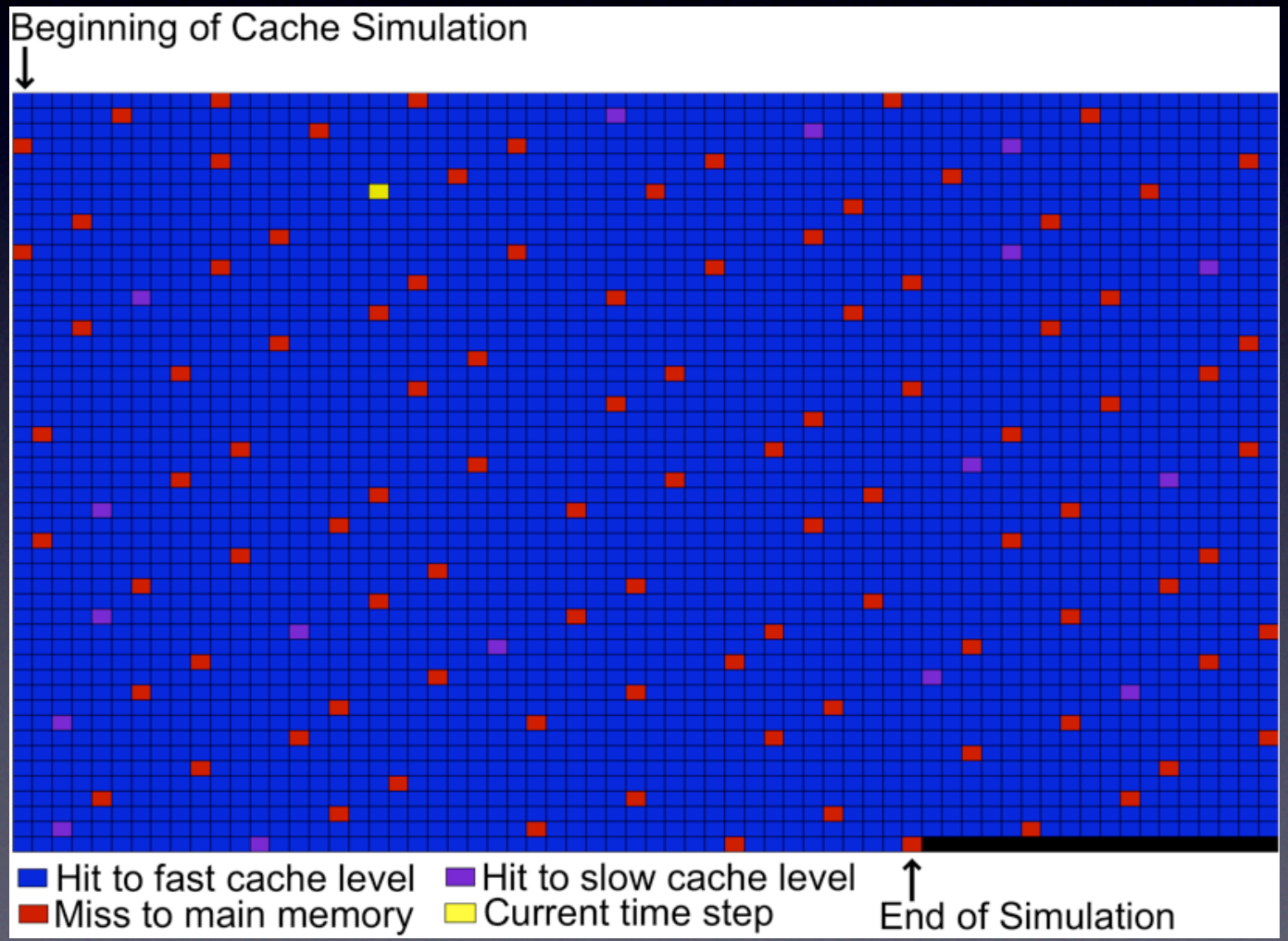
Orientation and Navigation

Source Code View



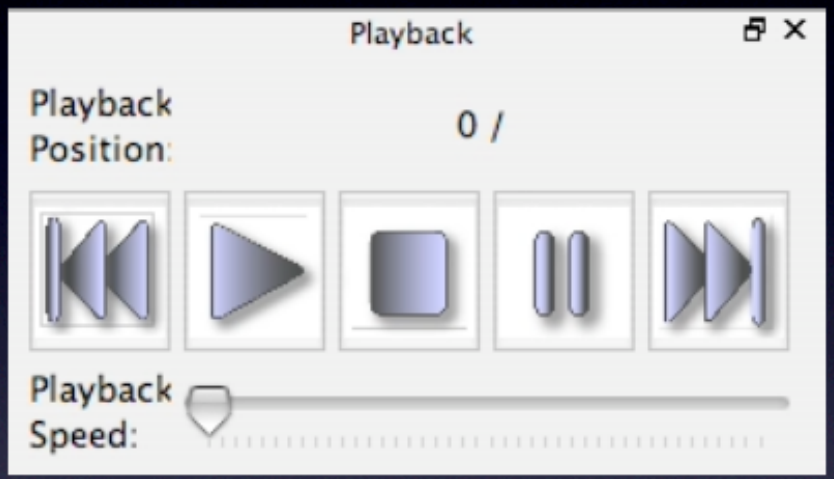
Orientation and Navigation

Cache Event Map



Orientation and Navigation

Playback Panel



- Pretty self-explanatory
- Gives the user a fine level of control over playback

MTV provides insight by...

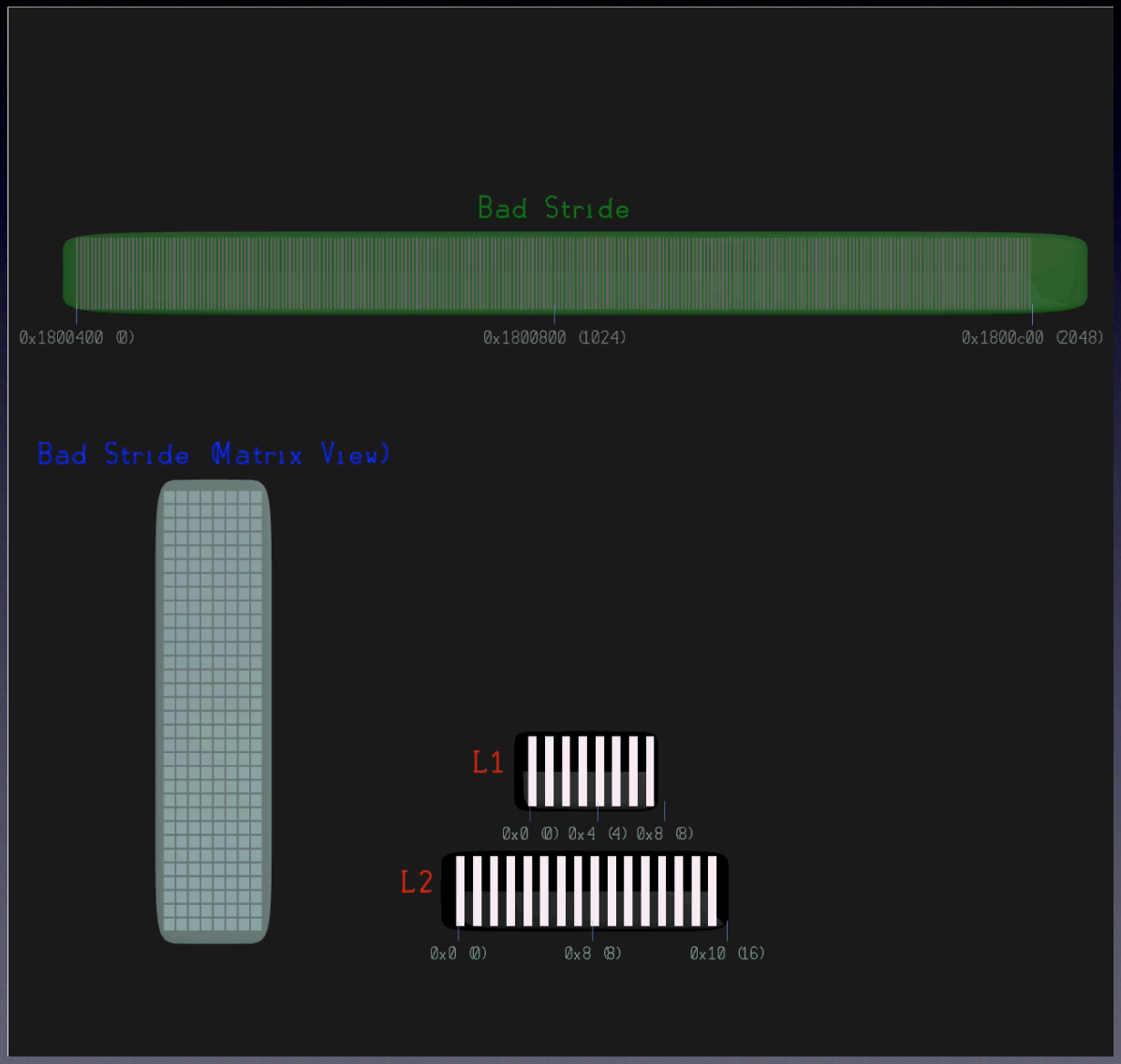
- correlating access patterns to high-level operations
- exposing reasons for poor performance
- allowing investigation of complex codes
- suggesting new abstractions

Example: Loop Interchange

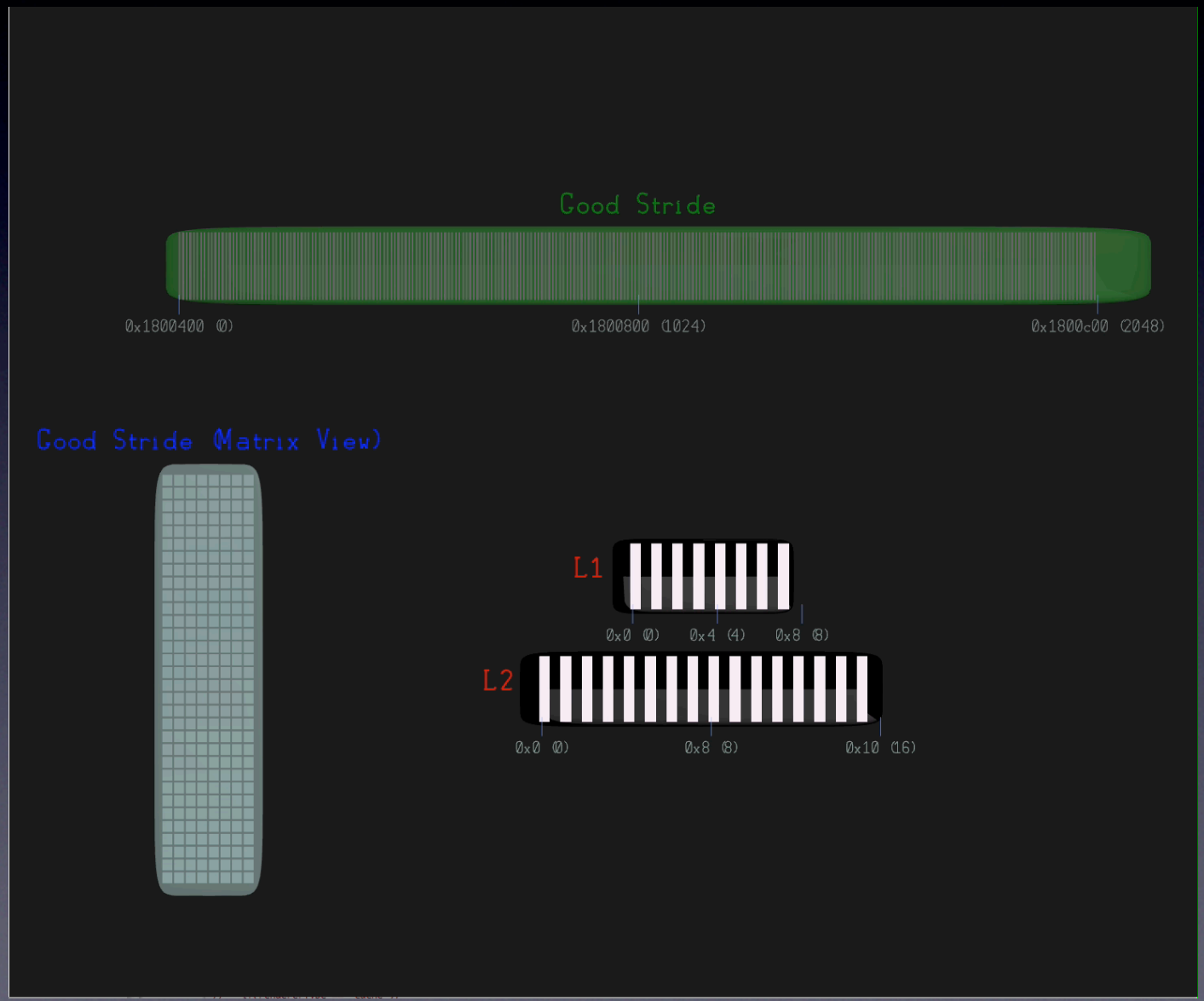
```
/* Bad */
double sum = 0.0;
for(j=0; j<4; j++)
  for(i=0; i<32; i++)
    sum += A[i][j];
```

```
/* Good */
double sum = 0.0;
for(i=0; i<32; i++)
  for(j=0; j<4; j++)
    sum += A[i][j];
```

Example: Loop Interchange



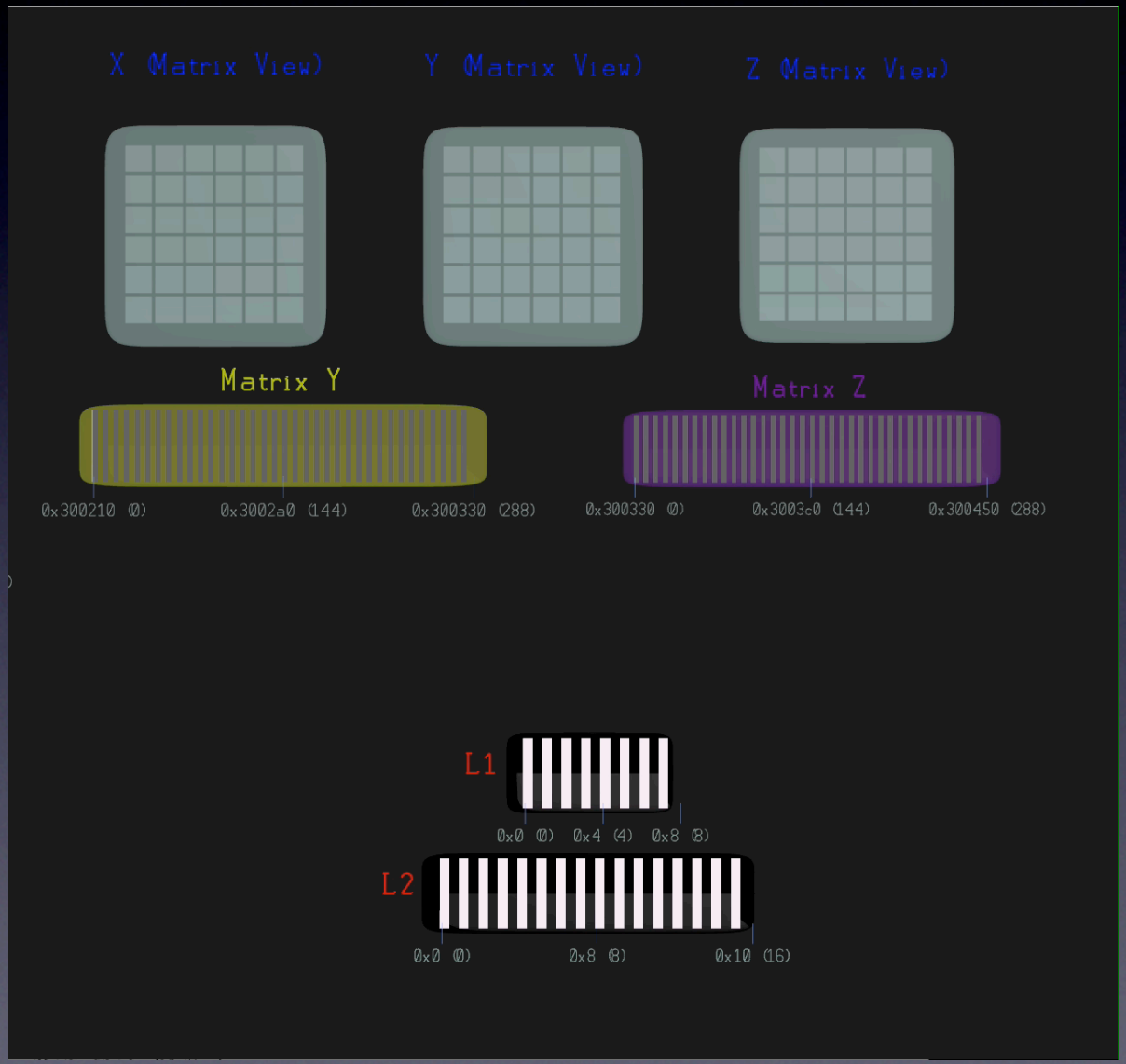
Example: Loop Interchange



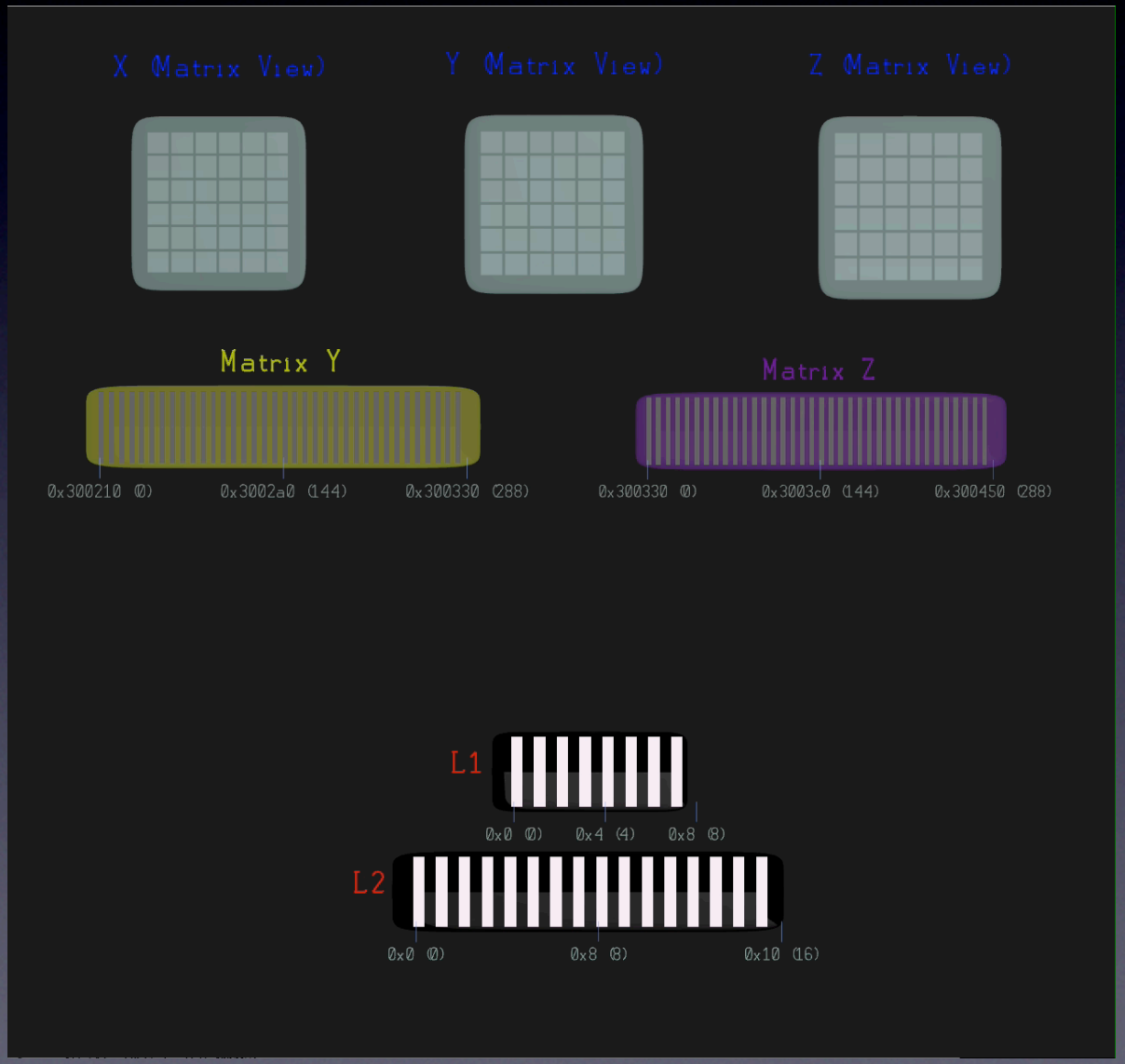
Example: Matrix Multiplication

```
for(i=0; i<N; i++)
  for(j=0; j<N; j++){
    r = 0.0;
    for(k=0; k<N; k++)
      r += Y[i*N+k] * Z[k*N+j];
    X[i*N+j] = r;
   }
```

Example: Matrix Multiplication (standard)



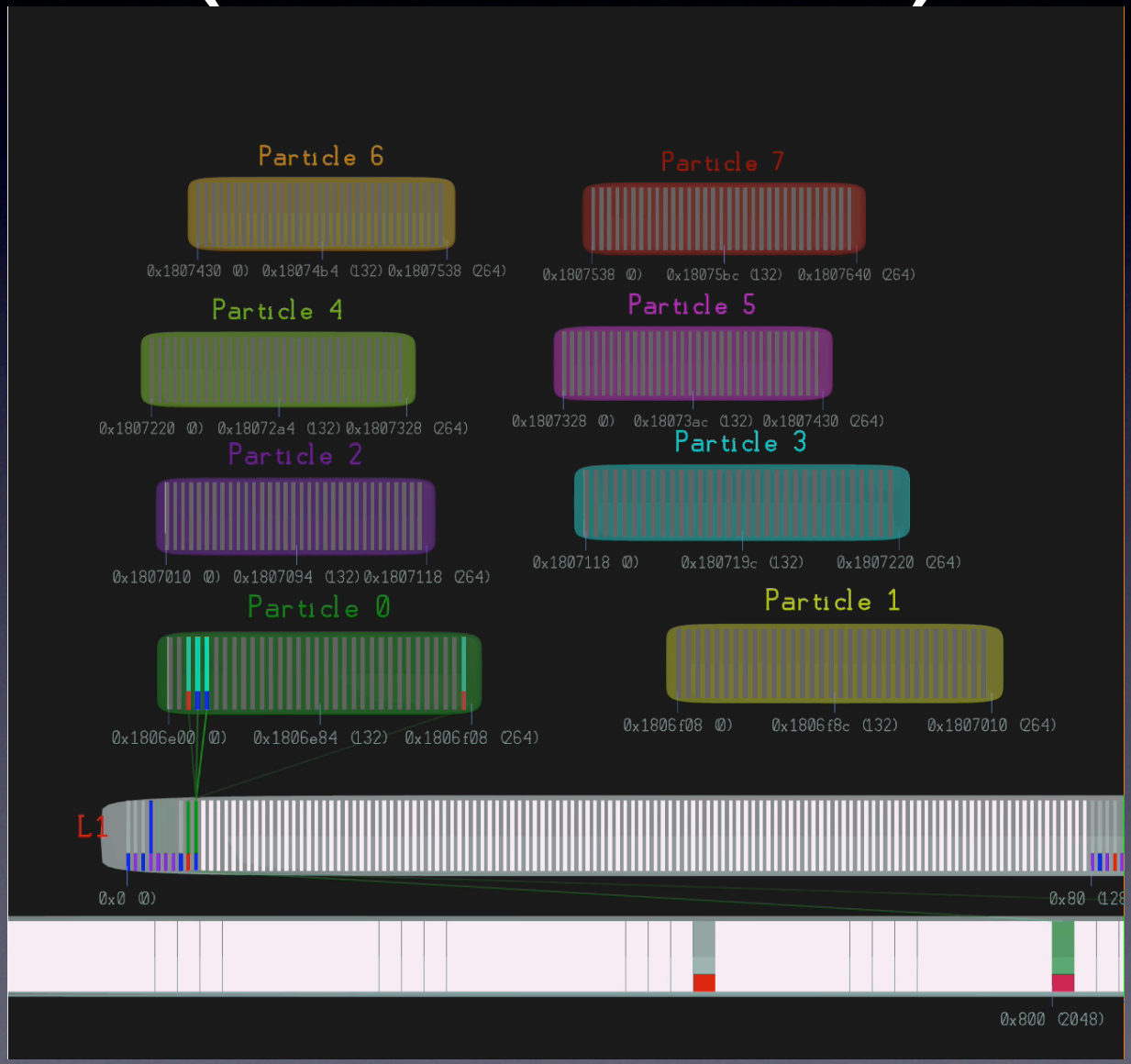
Example: Matrix Multiplication (blocked)



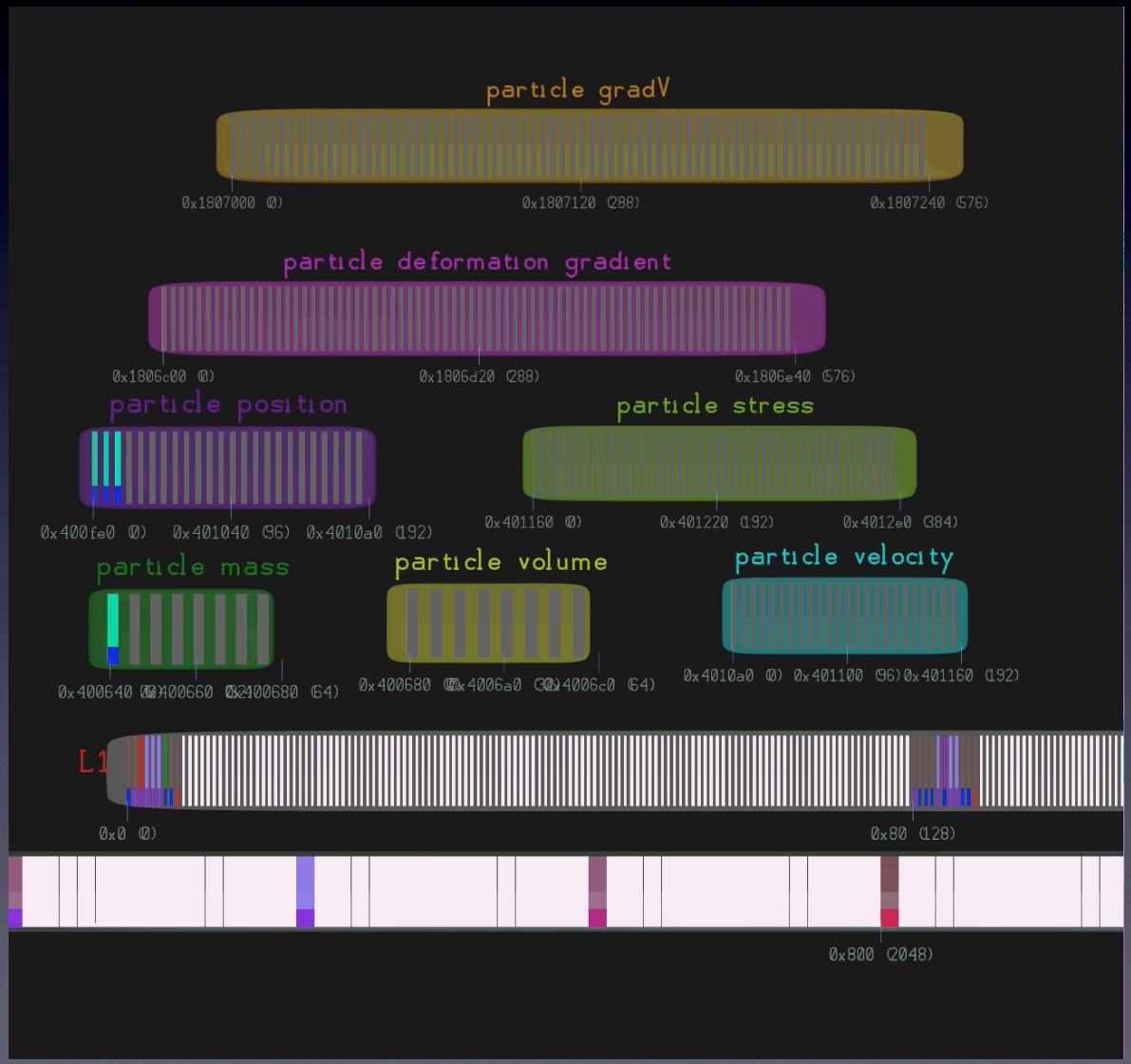
Example: Material Point Method

- A real code running on a real cache
- MPM is a method for mechanical simulation
- "Material points" carry physical parameters with it
- Data layout concerns: Horizontal vs. Vertical Storage
  - Horizontal: "array of structs"
  - Vertical: "parallel arrays"

Example: MPM (Horizontal)



Example: MPM (Vertical)



Conclusions

- MTV makes memory reference traces viewable through *cache simulation, cache visualization,* and *access pattern visualization*
- Helps the programmer derive *insight* about what is going on at a low level in a program
- This type of analysis is important to achieving *high performance*
- We hope it becomes more widespread, as interactive debuggers are today

Future Work

- Make MTV more informative: more detailed annotation of regions, better zooming (focus + context) for large objects like real-world caches, more systems information (heap, malloc/new traces, etc.)
- Make MTV more independent: remove instrumentation requirement
- Expand tracing model to parallel machines, multi-core, GPU, etc.

Questions?