# CMSC 714 – Cache Tools

**Chau-Wen Tseng**

**(Subbing for Jeff Hollingsworth)**

**Department of Computer Science**
**University of Maryland, College Park**

# Two Keys To Performance

- **Parallelism**
  - **too expensive to speed up single processor**
  - **combine power of multiple processors**

- **Locality**
  - **processors faster than memory, network**
    - **In cache $\Rightarrow$ avoid memory latency**
    - **on processor $\Rightarrow$ avoid network latency**

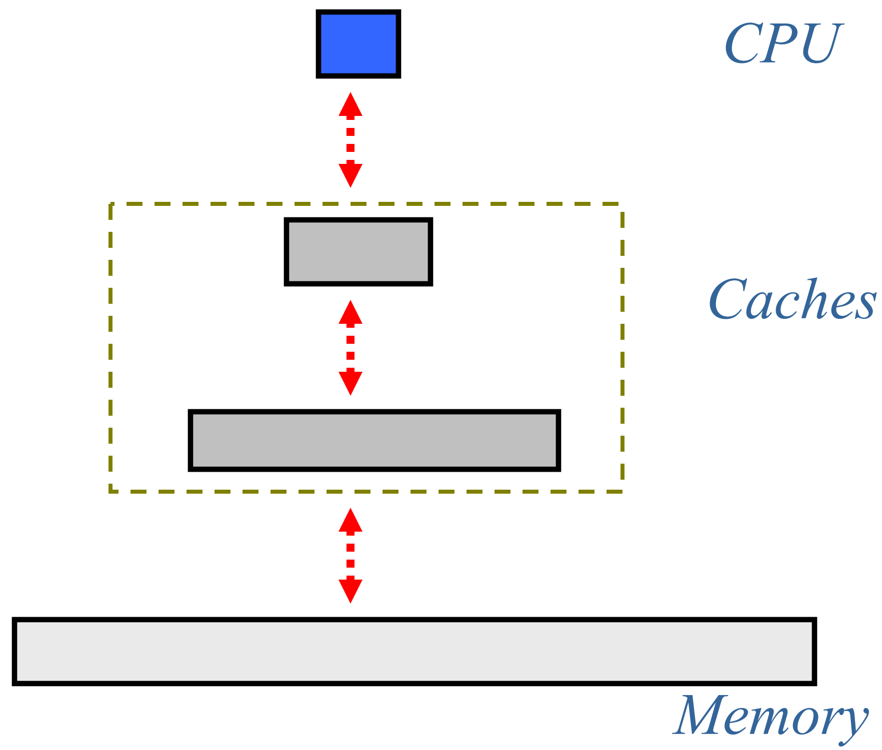
# 2 Papers

- John Mellor-Crummey, David Whalley, Ken Kennedy, "Improving Memory Hierarchy Performance for Irregular Applications Using Data and Computation Reorderings," International Journal of Parallel Programming, 29(3), June 2001
  - Examine impact of locality for scientific applications

- Margaret Martonosi, Anoop Gupta, Thomas Anderson, "MemSpy: analyzing memory system bottlenecks in programs", SIGMETRICS 92
  - Tool for analyzing multiprocessor cache locality

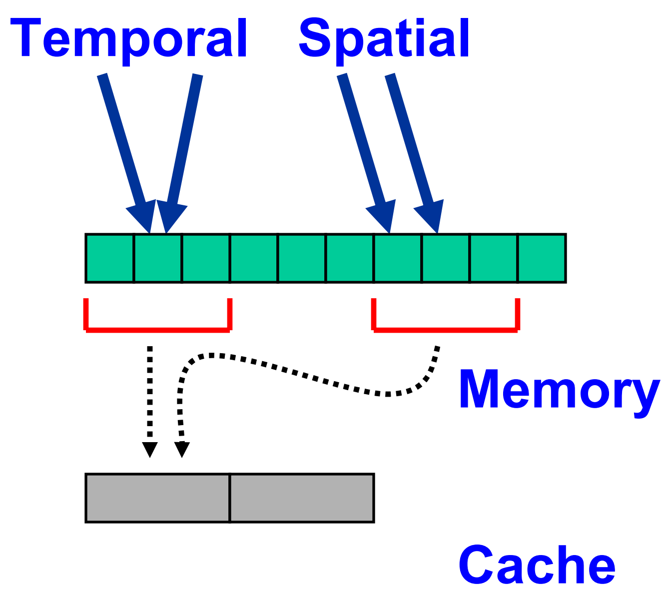
# Memory Hierarchy

- **Levels**
  - **Registers, cache, TLB, DRAM, disk…**
- **Higher levels smaller but faster**
  - **Disparity increasing**



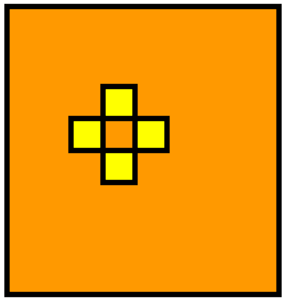
# Locality

- **Types of locality**
  - Temporal (reuse same data)
  - Spatial (reuse nearby data)

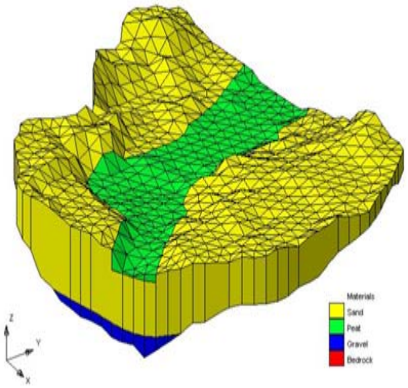


# Science & Engineering Applications

- **Two types of computations**
  - **Regular (dense matrix)**

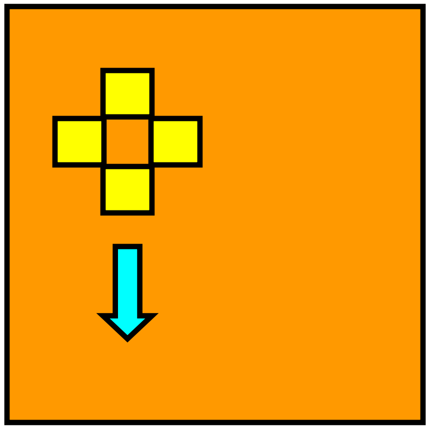


  - **Irregular (sparse matrix)**

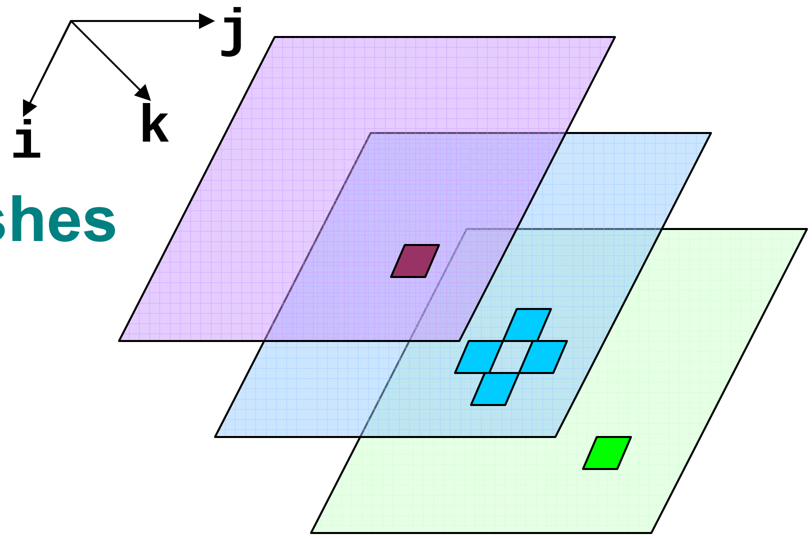


# Regular Computations

- **Characteristics**
  - Multidimensional arrays
  - Multiple loop nests
  - Regular access patterns

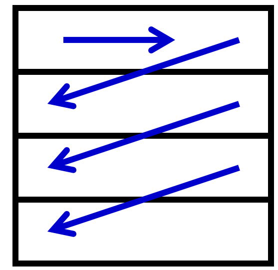


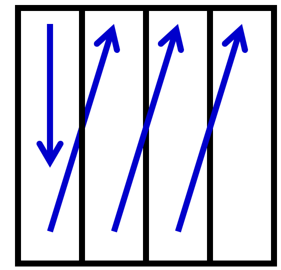
- **Examples**
  - Linear algebra
  - Simulations w/ uniform meshes
  - Image processing
  - Relational databases



# Array Layout

- **Multidimensional arrays**
  - **Linearized for memory storage**
    - **Row major (C, C++, Java)**
    - **Column major (Fortran)**

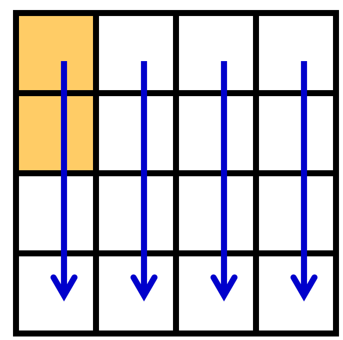




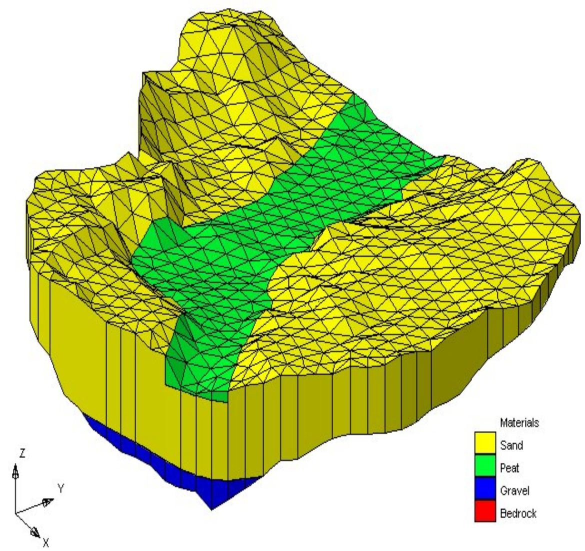
- **Contiguous accesses exploit spatial locality**

**<u>Regular codes</u>**

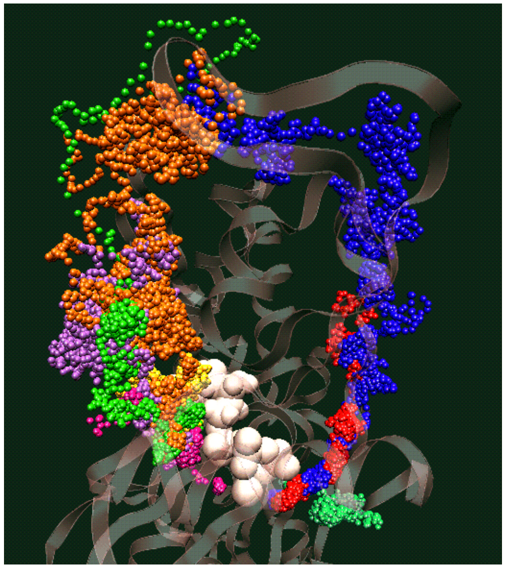
```
do j = 1, N
    do i = 1, N
     … = node[i, j]
```



# Irregular Computations

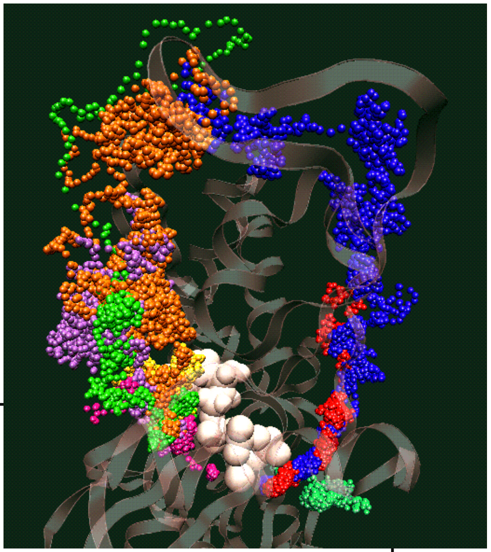



- **Characteristics**
  - **1D or 2D arrays**
  - **Multiple loop nests**
  - **Irregular, dynamic access patterns**
- **Examples**
  - **Sparse linear algebra**
  - **Simulations w/ sparse meshes**
    - **N-body**
    - **Molecular dynamics**



# Irregular Computation

- **Molecular dynamics**
  - **Example algorithm for Moldyn**



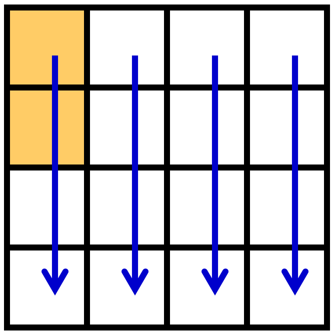
```
Initialize coordinates of particles
For N time steps DO
    Update particle coordinates based on velocity
    Build interaction list of nearby particles
    For each pair of interacting particles DO
        Update force on each particle
    Update velocity of each particle
```

# Problem

- **Irregular memory accesses** $\Rightarrow$ **poor locality**
  - **Unable to take advantage of memory hierarchy**

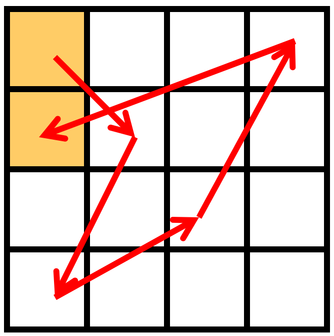
**Regular codes**

```
do i = 1, N
    do j = 1, N
     … = node[i, j]
```



**Irregular codes**

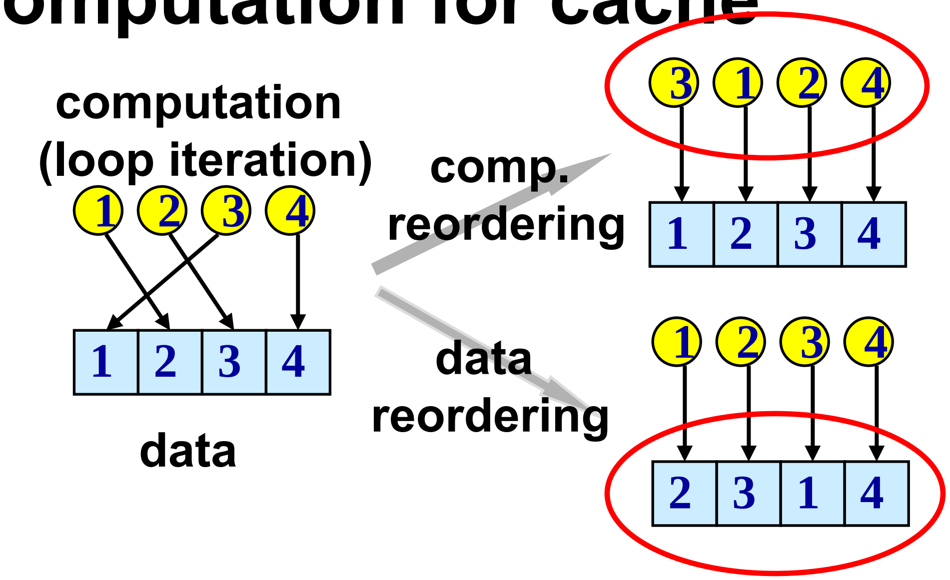
```
do i = 1, M
     … = node[ edge1[i] ]
     … = node[ edge2[i] ]
```



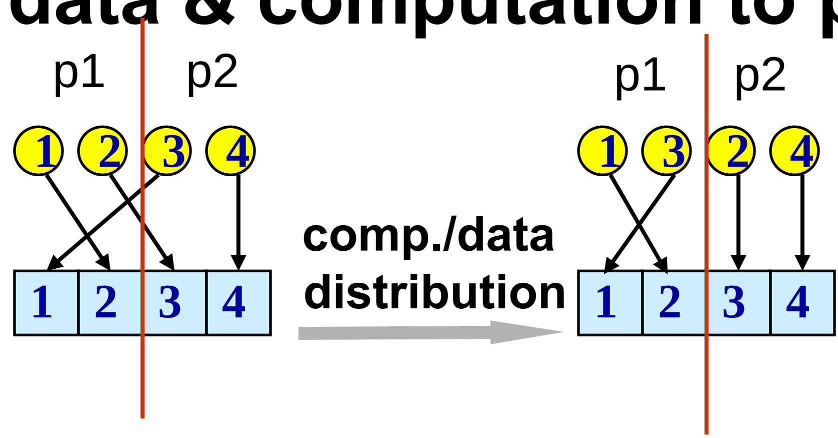
# Transformations for Irregular Codes

- **Reorder data & computation for cache**

```
do i = 1,E
  x[idx[i]] =
```




- **Distribute data & computation to processors**



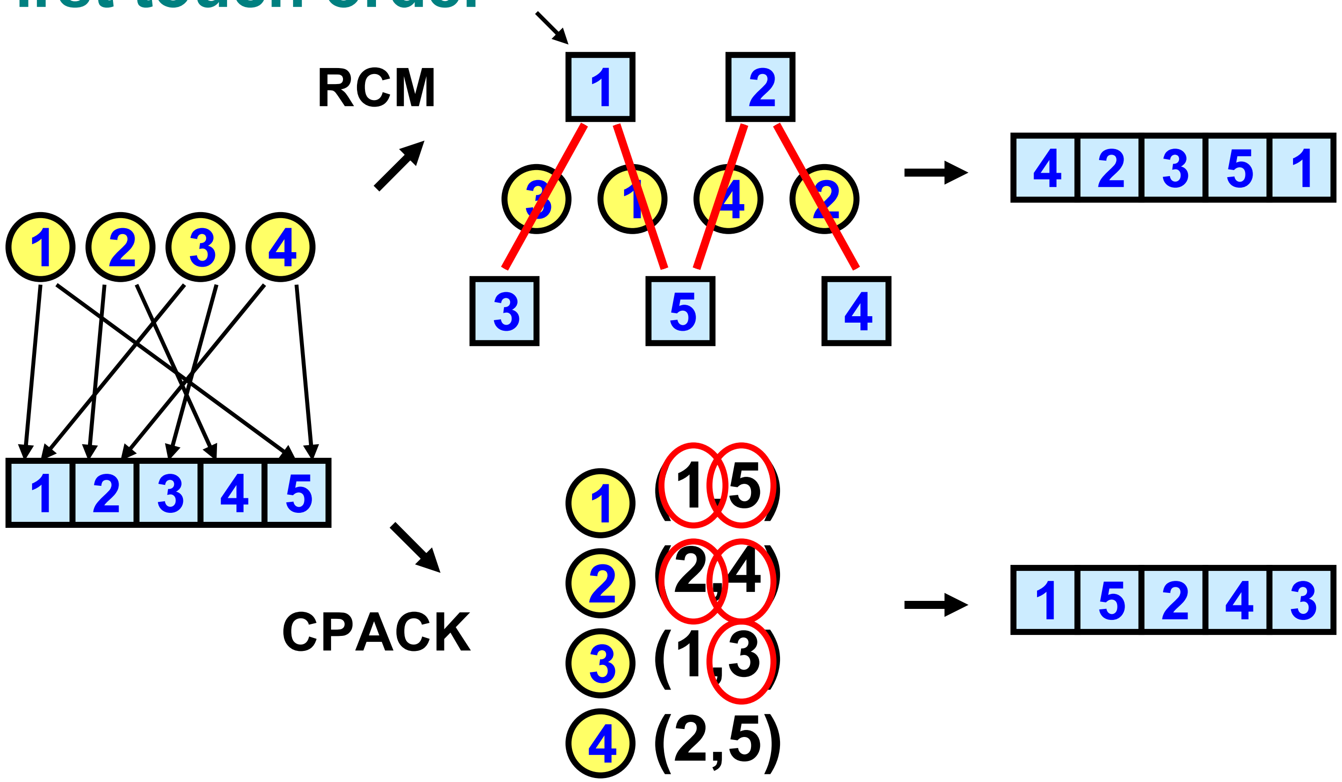
# Locality Optimizations

- **Data reordering**
  - Traversal algorithms (RCM, CPACK)
  - Geometric partitioning algorithms (RCB, Morton)
    - Use real coordinates or array index
  - Graph partitioning algorithms (METIS)
    - View accesses as a graph
    - Coordinates not needed

- **Computation reordering**
  - Bucket sort
  - Lexicographic sort
  - Space filling curves

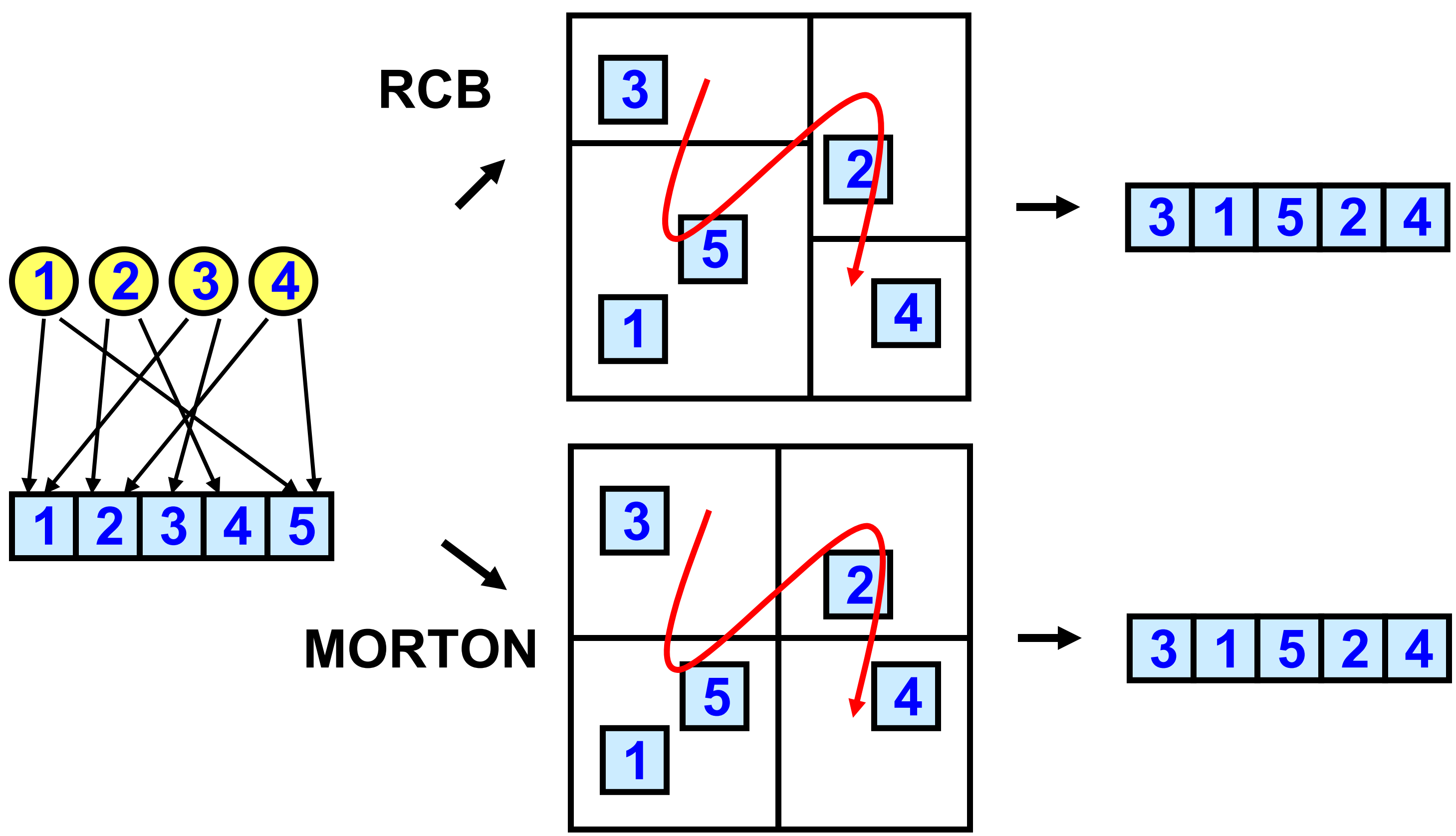
# Data Reordering - Traversal Algorithms

- **Reverse Cuthill-McKee (RCM)**
  - **Reverse BFS order**
- **Consecutive packing (CPACK)**
  - **First touch order**



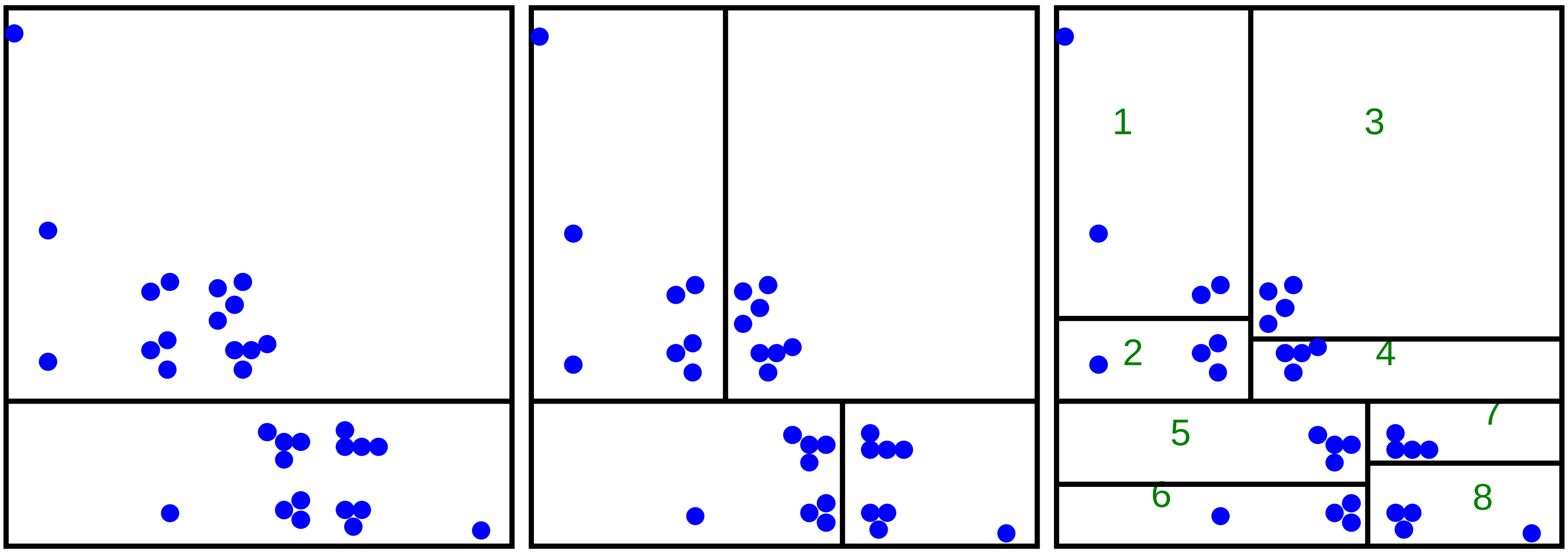
# Data Reordering - Partitioning Algorithms

- Recursive coordinate bisection (RCB)
- Space filling curves (MORTON)



# Data Reordering Algorithms

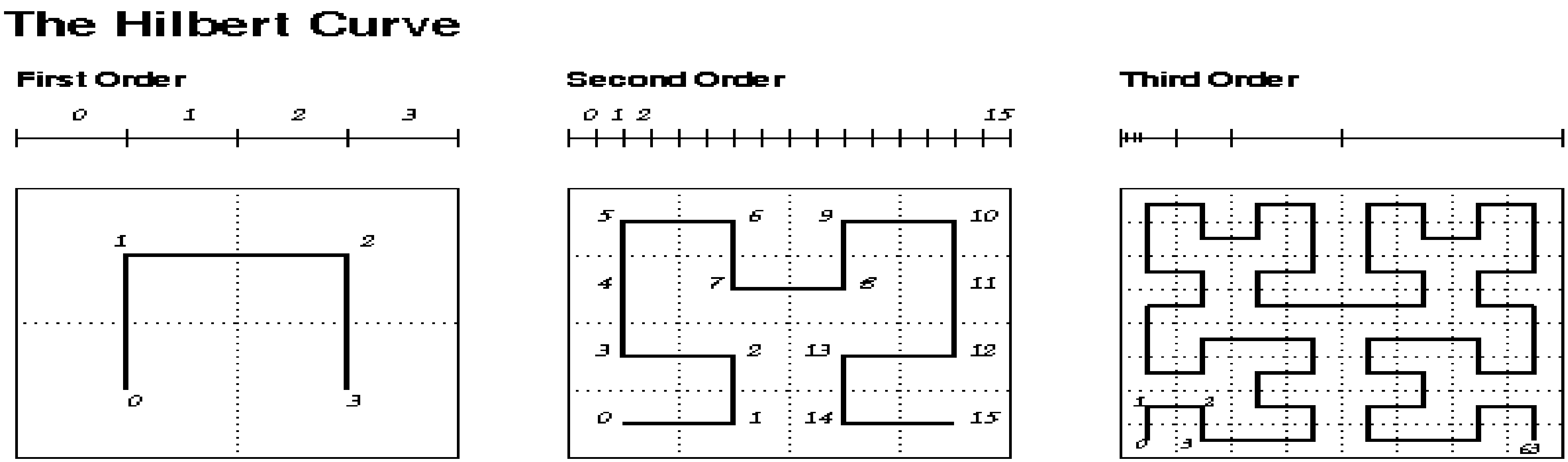
- **Recursive coordinate bisection**
  - **Recursively select median for dimension**

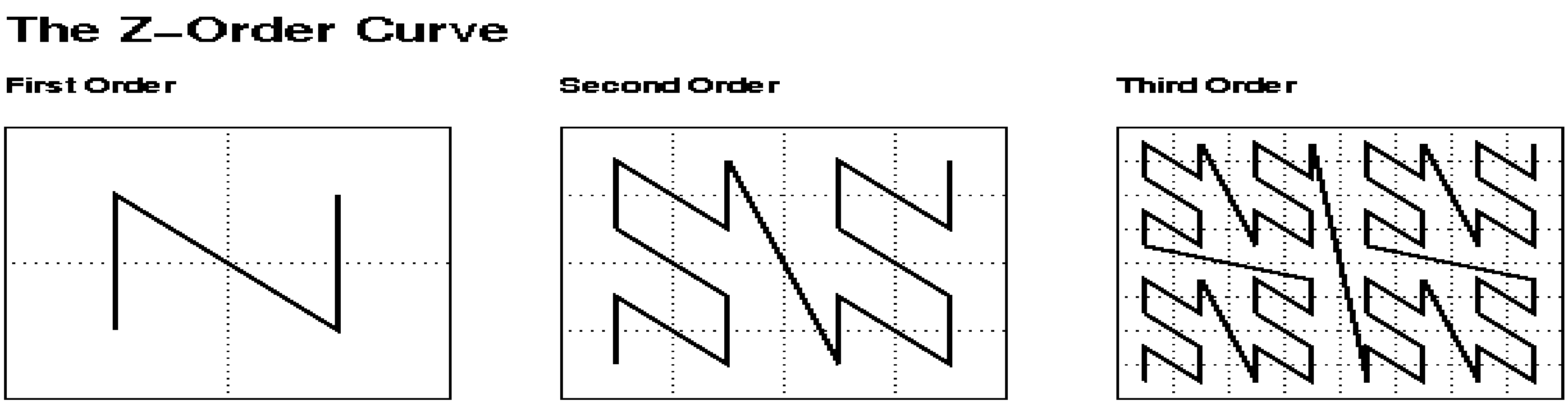


# Space Filling Curves

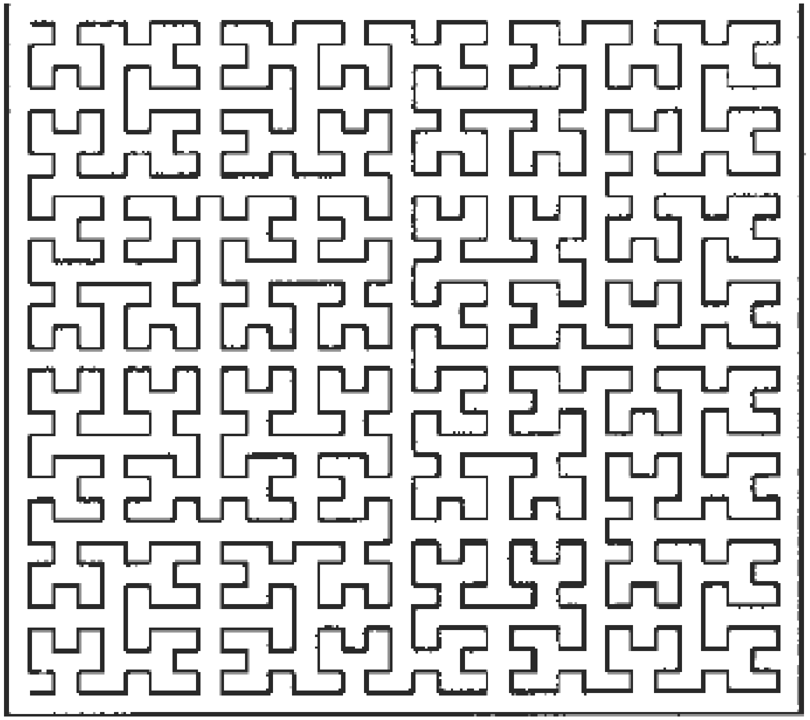
- **Characteristics**
  - Curve whose range contains every point in square
  - Used to map multidimensional data structures to 1D
  - Preserves locality
    - (5,5,5) likely to be close to (4,5,5), (5,4,5), (5,5,4)
  - Several types
    - Hilbert
    - Morton (Z-order)
      - Computed by interleaving binary coordinates
  - Can select granularity
    - Match to memory hierarchy (e.g., page size)

# Space Filling Curves

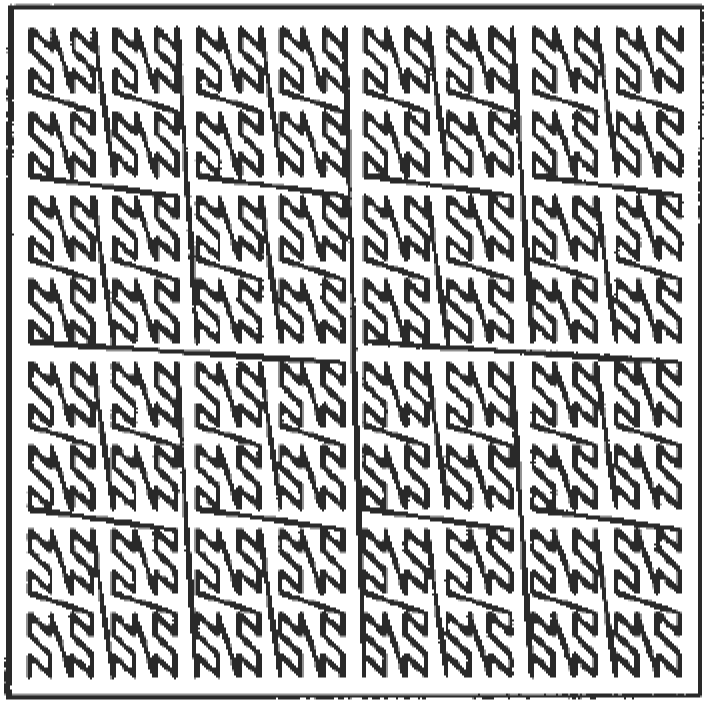


# Space Filling Curves

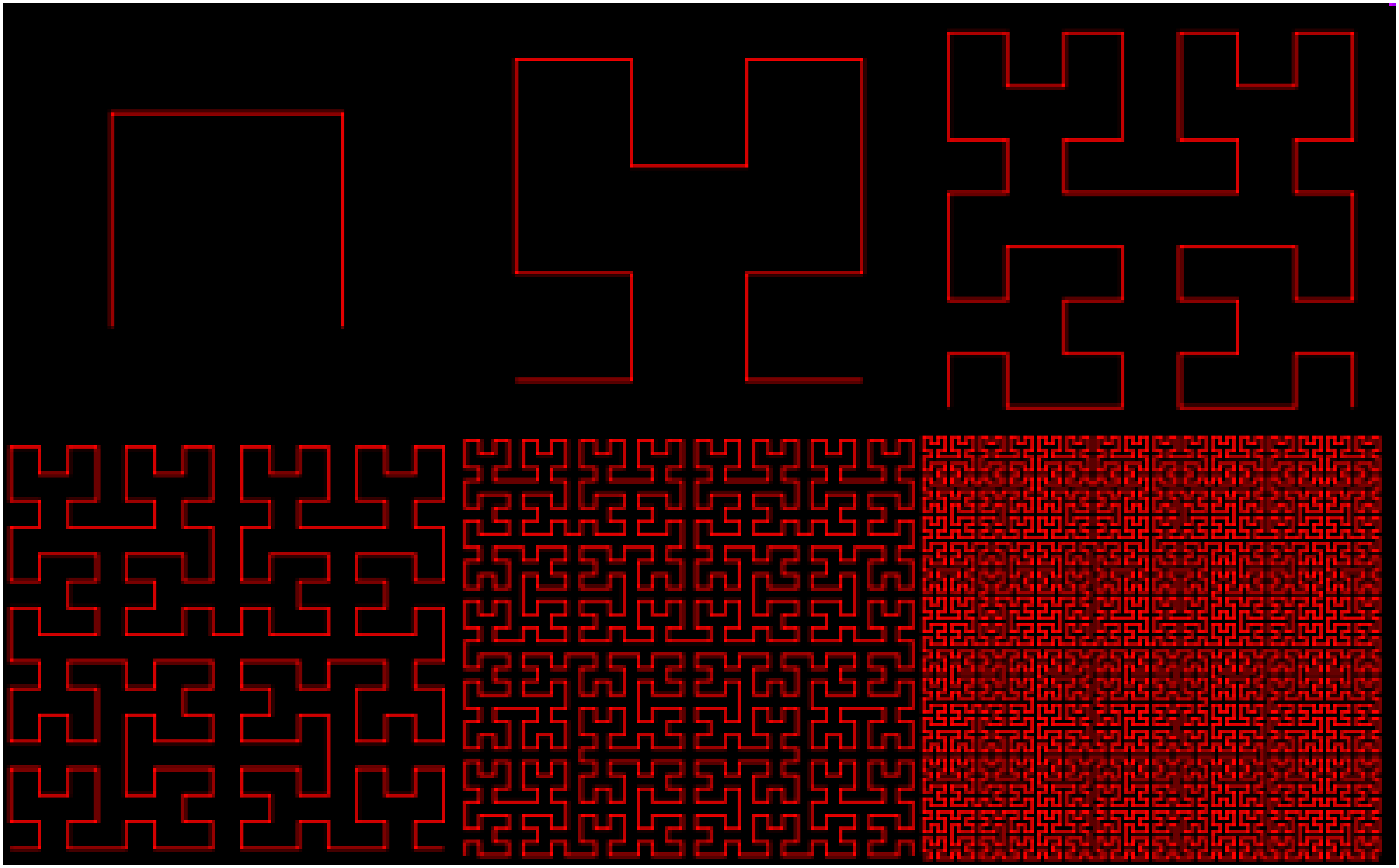


**Hilbert**

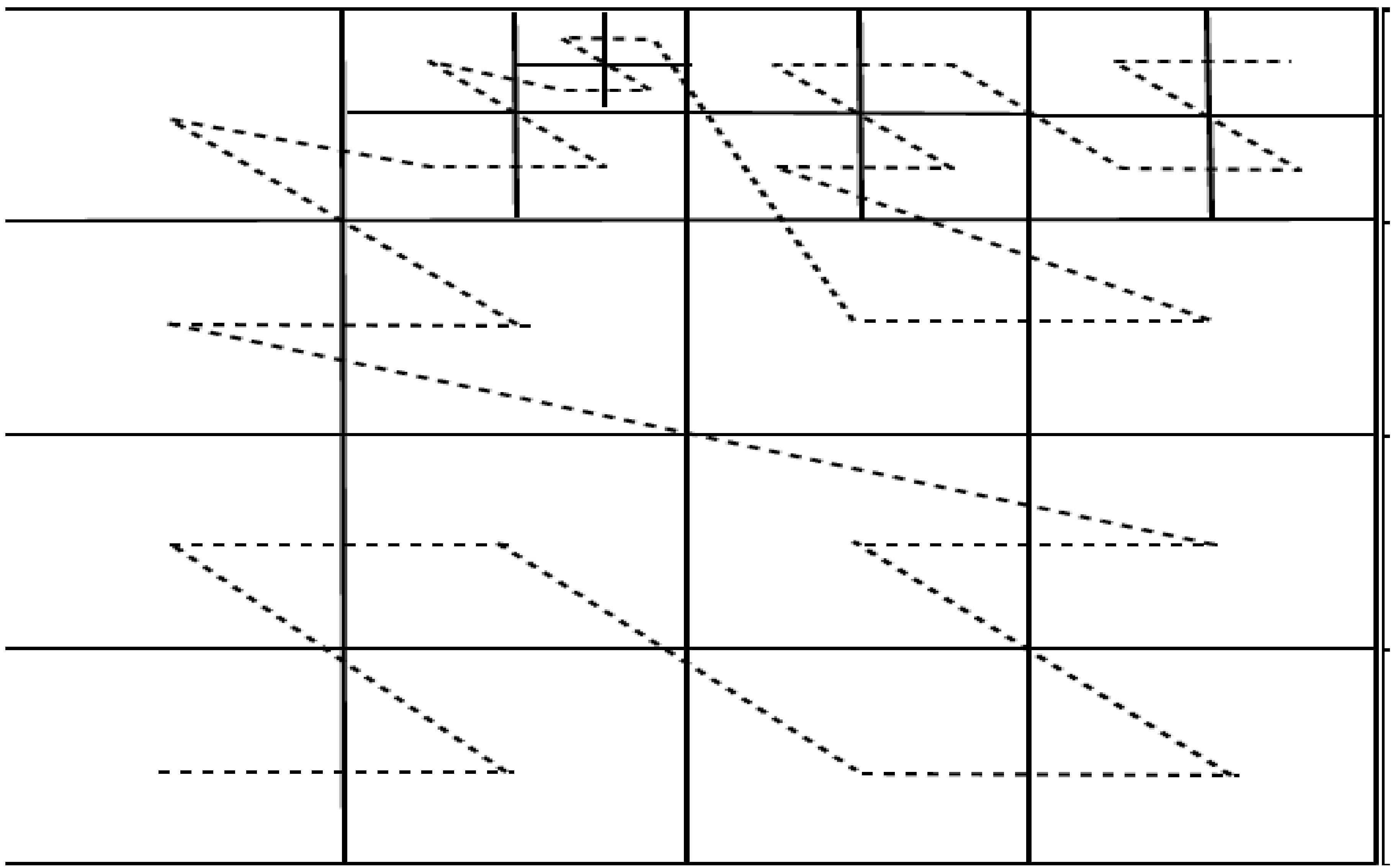


**Morton (Z)**

# Hilbert Space Filling Curve

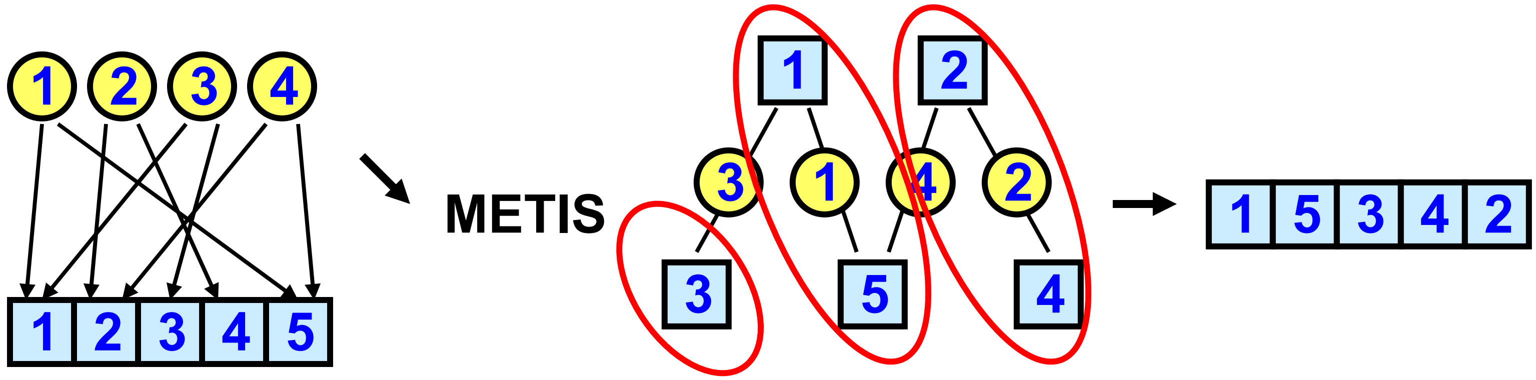


# Morton Curve For Adaptive Mesh



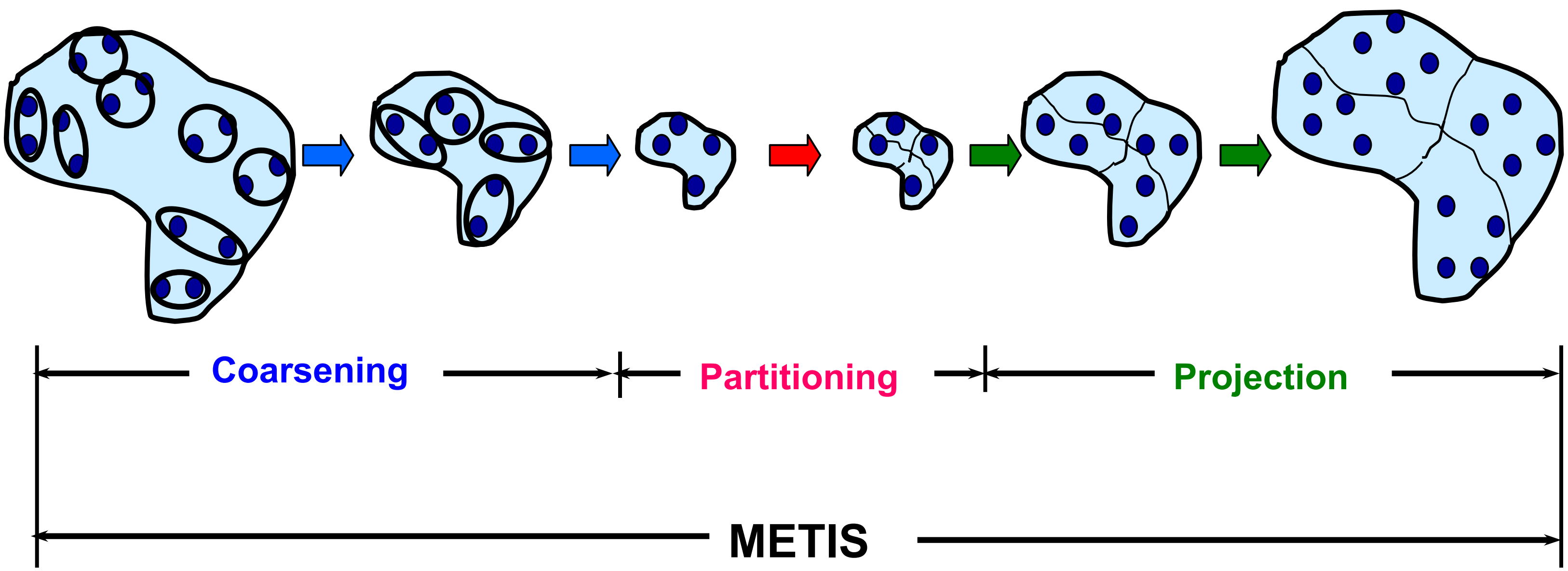
# Data Reordering - Partitioning Algorithms

- **Multi-level graph partitioning library (METIS)**



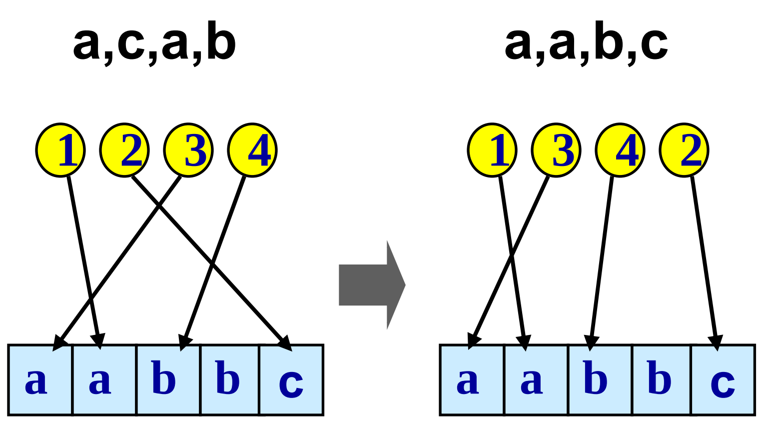
# Data Reordering Algorithms

- **Multi-level graph partitioning (METIS)**
  - **Simplify graph in phases**
    - **Merge neighboring nodes**
  - **Partition simplified graph**
  - **Project partition back to original graph**



# Computation Reordering

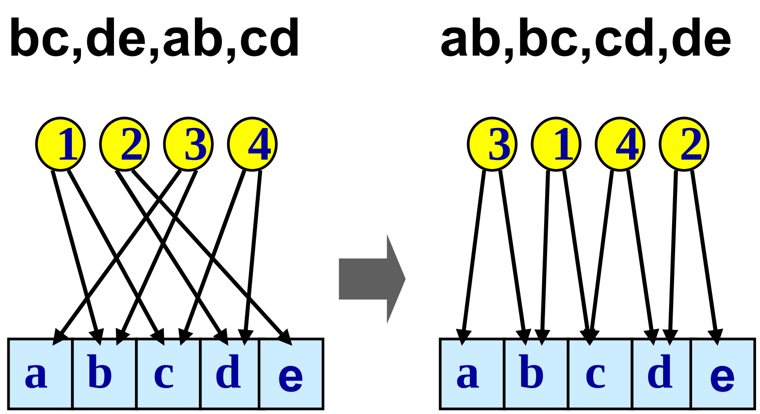
- **Bucket sort**
  - **Assign data to buckets (similar to tiling)**
  - **Label iterations based on bucket of data accessed**
  - **Reorder iterations using labels**




**Assumes 1 access per iteration**

# Computation Reordering (cont.)

- **Lexicographic sort / space filling curve**
  - **Assign vector label to iteration**
    - **Based on data accesses**
  - **Reorder iterations using labels**
    - **Lexicographic sort**
    - **Space filling curve**

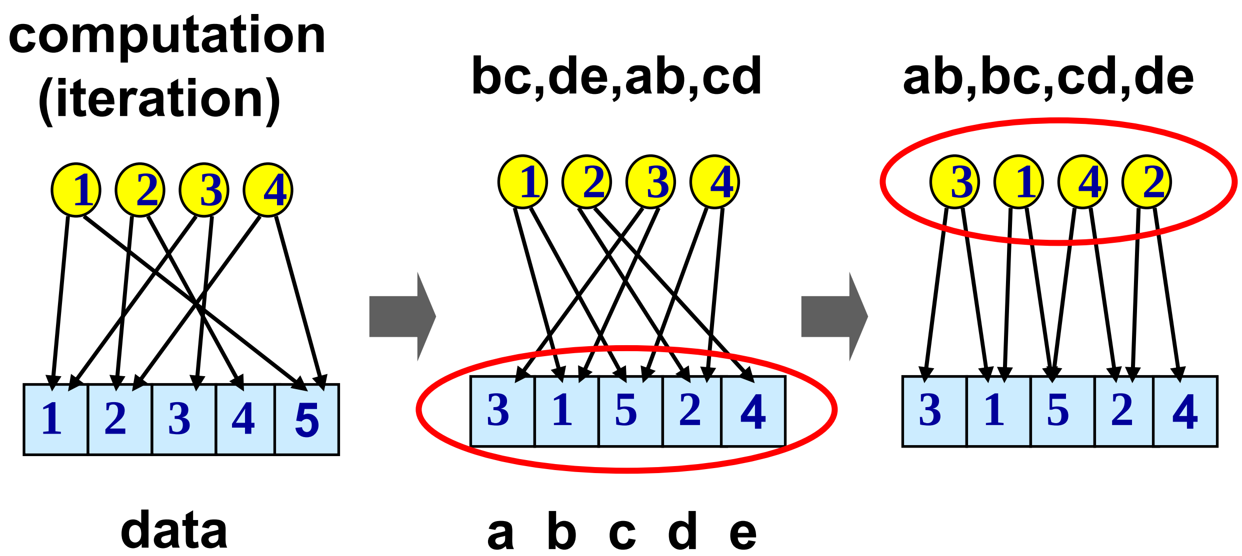



**Allows multiple access per iteration**

# Locality Optimization Algorithm

- **Framework**

  **1) Reorder data**
  **2) Reorder computation**




Must also decide whether benefit of improved locality is worth overhead of reordering data & computation

# Chronology

- **Locality reordering**
  - **Das *et al.* : RCM & *Lexicographical Sort* [AIAA'94]**
  - **Al-Furaih and Ranka : METIS & BFS [IPPS'98]**
  - **Ding and Kennedy :**
    **CPACK & *Lexicographical Sort* [PLDI'99]**
  - **Mellor-Crummey *et al.* : Space Filling Curve [ICS'99]**

# Runtime Transformation

- **Inspector / executor approach**
  - **Insert call to inspector in run-time library**
  - **Original computation transformed to executor**
  - **At run time, inspector can**
    - **reorder data & computation**
    - **partition computation for parallel execution**

| original code | | transformed code |
|---|---|---|
| do i = 1,E<br>x[idx[i]] = | → | inspector(x, idx)<br>// executor<br>do i = 1,E<br>x[idx[i]] = |

  - **Used for both locality optimizations & parallelization**

# Compiler Support

- **Identify irregular reductions**
- **Locate access pattern changes**
- **Insert library call - reorder data & computation**
- **Reinvoke inspector if access pattern changes**

**original code**

```
idx[ ] = …   // init idx[ ]

do t = 1, time
    if (change)
        idx[…] = ...

    do i = 1, M
        … = x[ idx[i] ]
```

**transformed code**

```
idx[ ] = …   // init idx[ ]
inspect(x, idx)
do t = 1, time
    if (change)
        idx[…] = ...
        inspect(x, idx)
    do i = 1, M
        … = x[ idx[i] ]
```

# Experimental Evaluation

- **Benchmarks**
  - **Two particle kernels - Moldyn, Magi**
  - **Unstructured mesh application – CHAD**

- **Measurements**
  - **Cache simulator**
  - **Hardware counters on SGI Origin 10000 (SMP)**

# Experimental Evaluation (cont.)

- **Results (data/computation)**
  - Moldyn
    - Hilbert/Hilbert best (25% L1 misses)
  - Magi
    - Hilbert/Hilbert best (28% L1 misses)
  - CHAD
    - none/lexicographic best (96% L1 misses)
    - Hilbert increased cache misses due to overhead
- **Conclusions**
  - Locality opts. needed for some irregular computations
  - Particle codes (Moldyn, Magi) have more temporal locality, thus benefit more than mesh codes (CHAD)?

# 2 Papers

- John Mellor-Crummey, David Whalley, Ken Kennedy, "Improving Memory Hierarchy Performance for Irregular Applications Using Data and Computation Reorderings," International Journal of Parallel Programming, 29(3), June 2001
  - Examine impact of locality for scientific applications

- Margaret Martonosi, Anoop Gupta, Thomas Anderson, "MemSpy: analyzing memory system bottlenecks in programs", SIGMETRICS 92
  - Tool for analyzing multiprocessor cache locality

# MemSpy

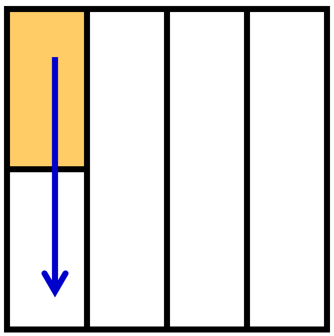
- Simulator tool for analyzing cache performance
- Features
  - Data structure-specific cache statistics
    - % total memory stall time due to each heap object
  - Supports multithreaded codes
  - Reports cause of cache miss
    - Cold ($1^{st}$ reference) miss
    - Invalidate miss
    - Replacement miss
- Combination of features
  - Helps explain memory behavior
  - Aids in performance tuning

# Multimensional Array Layout

- **Contiguous accesses exploit spatial locality**

**Column accesses**

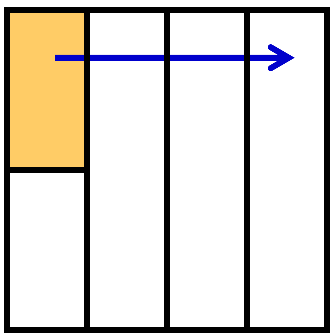
```
do j = 1, N
    do i = 1, N
     … = node[i, j]
```



- **Non-contiguous accesses waste cache lines**

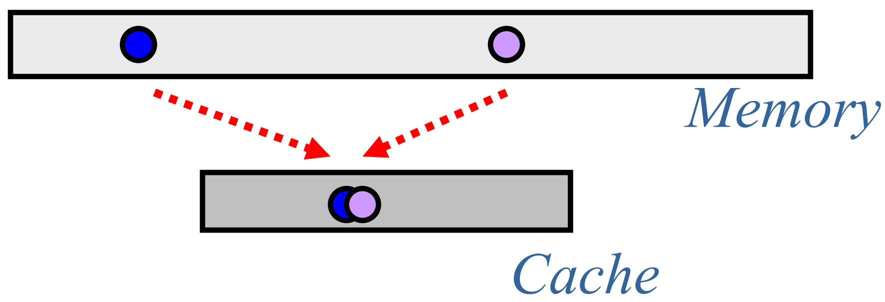
**Row accesses**

```
do i = 1, N
    do j = 1, N
     … = node[i, j]
```



# Cache Misses

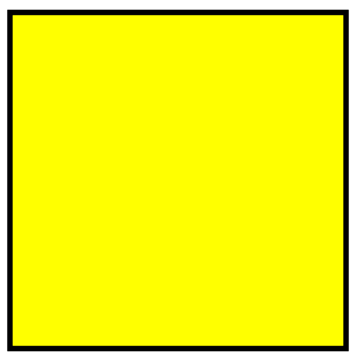
- **Capacity misses: limited cache size**
- **Conflict misses: limited set associativity**
  - **Referred to as self-interference misses**
  - **50% conflict misses (McKinley & Temam, [ASPLOS'96])**



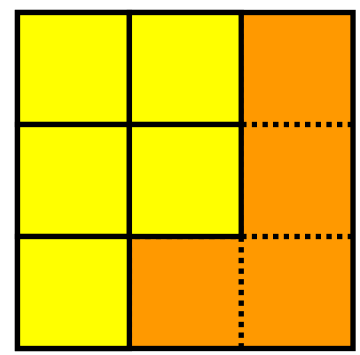
# Tiling / Blocking Regular Codes

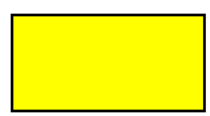
- **Computation reordering transformation**
  - **Bring reuses closer in time**
  - **Iteration broken into tiles (blocks)**
  - **Reduces capacity misses**
  - **Can introduce conflict misses**

*N sweeps for entire array → Too large to fit in cache*



*N sweeps for each tile → Tile fits in cache*



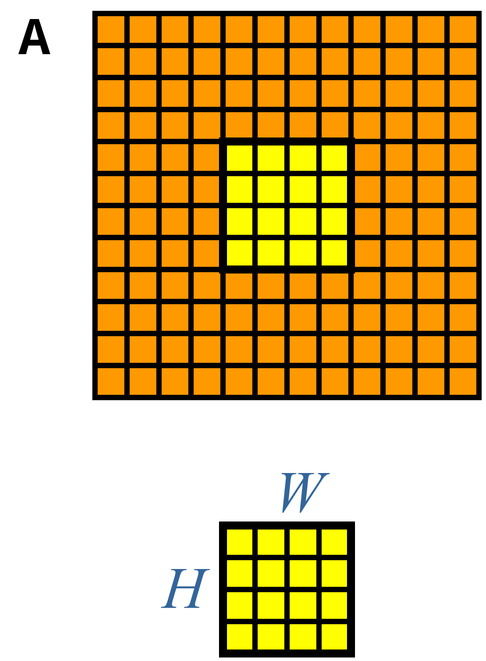


*cache*

# Tiled 2D Codes

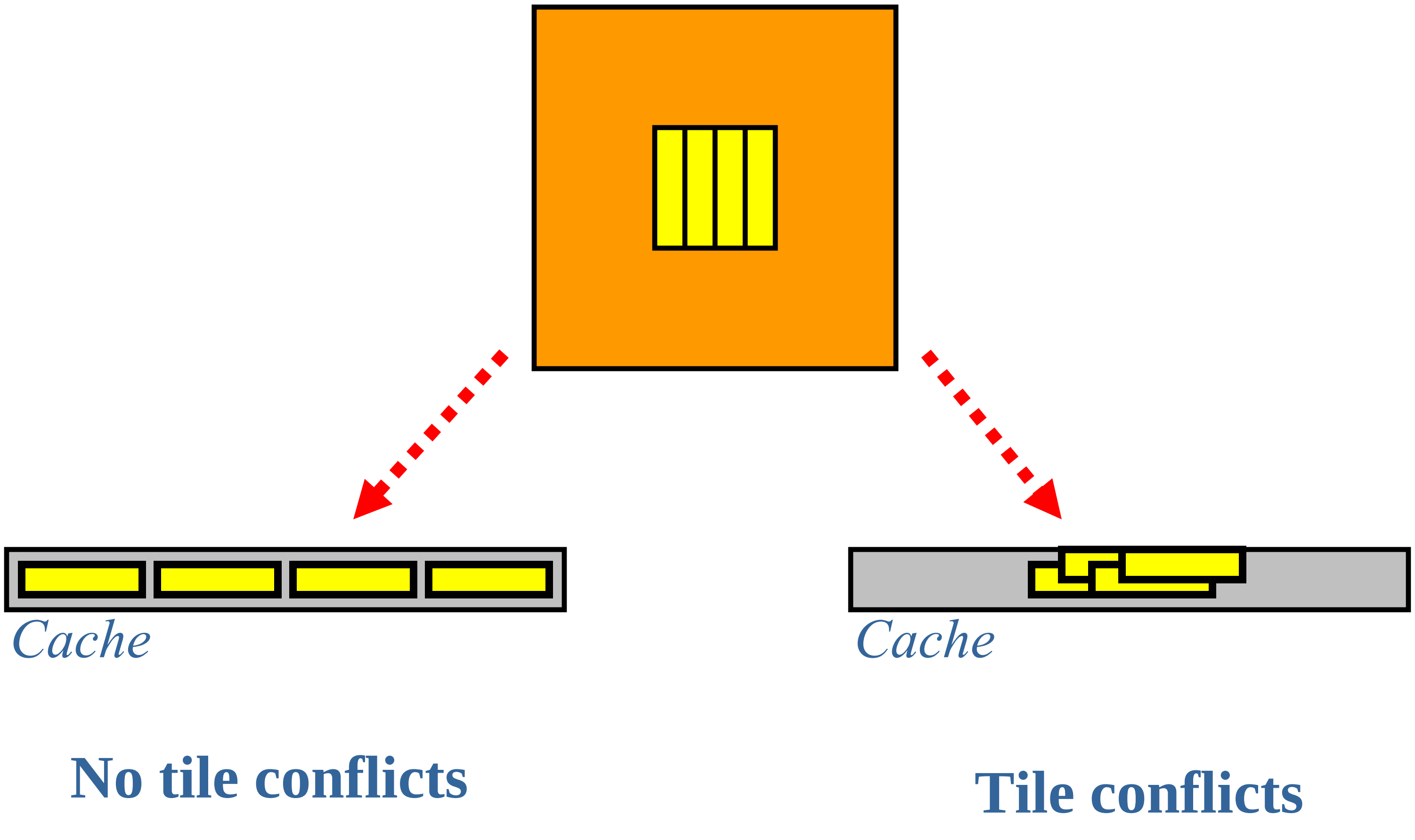
- ***Mult*** **example (C = A*B):**

```
do KK=1,N,W
do II=1,N,H
do J=1,N
do K=KK,min(KK+W-1,N)
do I=II,min(II+H-1,N)
do J=1,N
do K=1,N
do I=1,N
   C(I,J) = C(I,J) + A(I,K)*B(K,J)
```



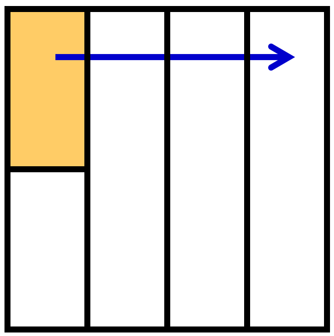
# Conflicts in Tiled 2D Codes

- **2D subarray (HxW) overlaps on cache**

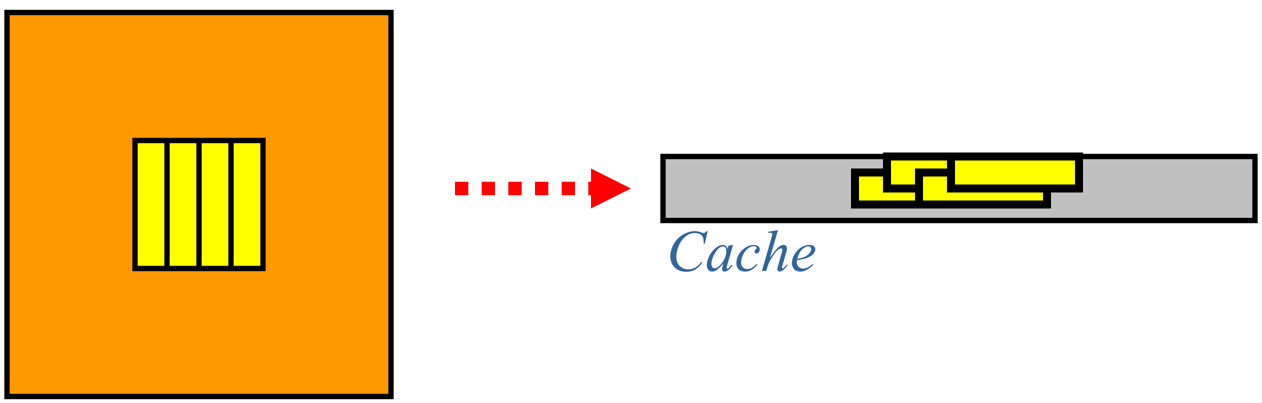


# MemSpy Case Studies

- **Examples of how to analyze cache performance**
  - **High cold miss rate → poor spatial locality**

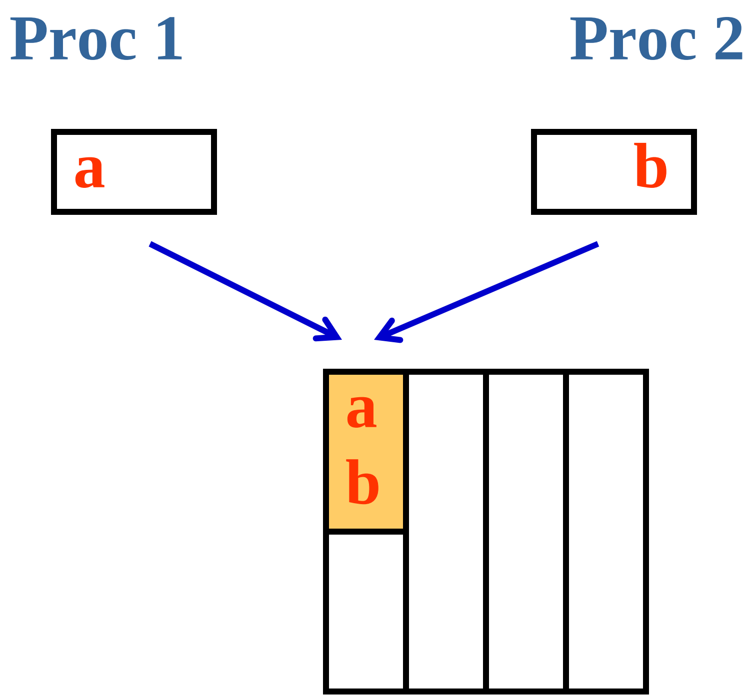


  - **High self replacement rate → conflict misses**



# MemSpy Case Studies (cont.)

- **Examples of how to analyze cache performance**
  - **High invalidate misses → poor multiprocessor locality (possibly false sharing)**



# MemSpy Design

- **Implemented using Tango simulator**
  - Inserts procedure call per memory reference
  - 40% increase in execution time
- **Data structure specific statistics**
  - Heap allocated data structures aggregated into bins
  - Same bin if allocated
    - at same point in program w/ identical call path
  - % of total stall time used to prioritize data structures
- **Cause of cache miss is recorded**
  - Maintain and use 1D array of state bits

# Student Questions – Locality Opts

- **Q**
  - **Is there an intuitive explanation for why space-filling curves improve temporal and spatial locality better than more simple orderings?**
- **A**
  - **Actually only improves spatial locality. Simple orderings (e.g., row/column) have large jumps going from 1 column/row to the next. I.e., with row-major ordering two neighboring points 1 row apart are separated by the size of the entire row.**

# Student Questions – Locality Opts

- **Q**
  - **What is the breakdown of regular vs. irregular applic ations?**
- **A**
  - **Not sure, but trend is towards irregular applications as problem size & complexity increase**

# Student Questions – Locality Opts

- **Q**
  - **They often mention that their re-ordering improvements are x times better than a random ordering. I would think that a more natural baseline would be some sort of row-based or column-based ordering. It seems like a random ordering would just be inherently wasteful in terms of spatial locality benefits. Is there any reason why a random ordering was used as the baseline comparison?**
- **A**
  - **I think random is just one example. Baseline is with respect to the original particle order, I believe.**

# Student Questions – Locality Opts

- **Q**
  - **Are the Hilbert and Morton curves pretty much the only space-filling curves currently used? I notice that points in the very center of the space that are spatially very close to each other, are very far apart on the curve.**
- **A**
  - **Hilbert is better than Morton in avoiding big jumps. Other space filling curves exist, though I'm not sure whether they are used at all.**

# Student Questions - MemSpy

- **Q**
  - **The paper failed to address a few of my questions ab out the role of the simulator in MemSpy, e.g. why is t he simulator needed at all and what exactly is its pur pose?**
- **A**
  - **Some mechanism is needed to be able to predict cache behavior. Without hardware counters a simulator is the only way to be able to track the stream of memory references.**

# Student Questions - MemSpy

- **Q**
  - **Has hardware tracing proven more effective than usi ng a simulator?**
- **A**
  - **Depends on what you mean by effective. Hardware cache counters are much faster, but provide less detailed information and cannot be used to test different cache configurations.**

# Student Questions – MemSpy

- **Q**
  - **MemSpy seems like a good tool for analyzing programs that run on dedicated hardware. But, it seems like if the program were intended to run within an OS environment, context switching and OS data structures would change the behavior of the cache. So, I wonder whether the simulations that MemSpy uses would accurately reflect the execution if the program in its actual environment.**
- **A**
  - **Application-level cache simulators ignore the impact of context switching on cache behavior.**

# Student Questions – MemSpy

- **Q**
  - **MemSpy seems like a good analysis tool to use when targeting a single architecture (homogeneous cluster or single computer ). However, would such a cache analysis tool be useful at all in a heterogeneous grid computing environment?**
- **A**
  - **Only shared-memory architectures need to worry about shared caches. Grids communicate via messages, in effect making copies of nonlocal data as needed. This eliminates invalidate misses.**