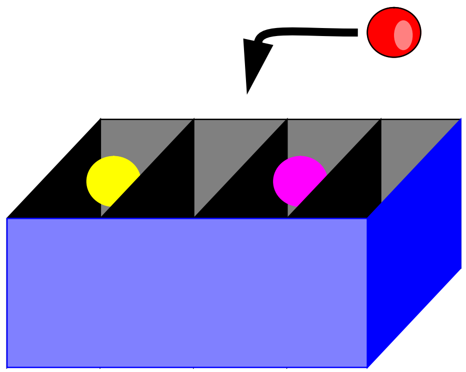
Accurate Data Distribution Into Blocks May Boost Cache Performance

Dan Truong, François Bodin, André Seznec

I R I SA - I N R I A (France)



Problem

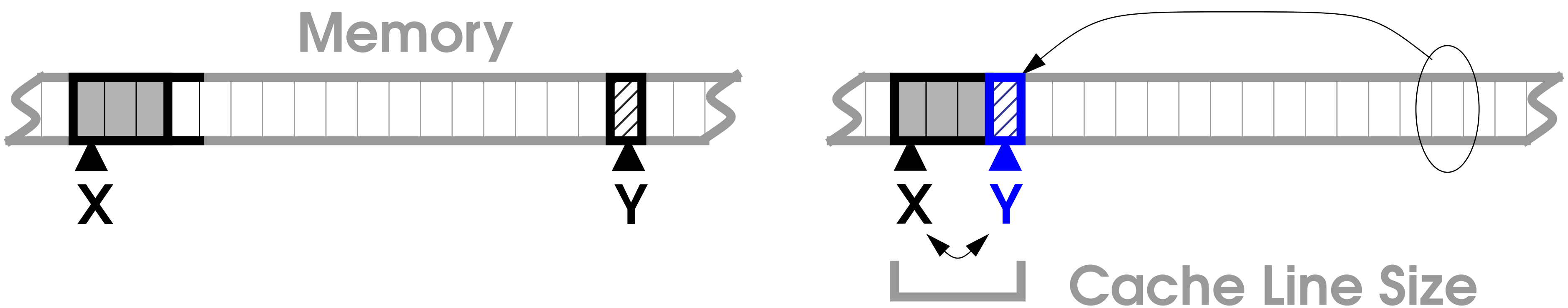
- ❑ Memory Wall
  - Limits CPU performance
- ❑ 90% cache space is wasted (Burger95)
  - Excessive miss ratio
- ❑ Compilers ignore cache
  - Data layout with other constraints or is arbitrary

Data layout needs more consideration

- ❑ Bigger, better hardware, tiling... are not sufficient
- ❑ Data distribution may help to fill the gap
  - It's data layout: Software becomes hardware aware
  - OK with Non-scientific codes

Data Distribution

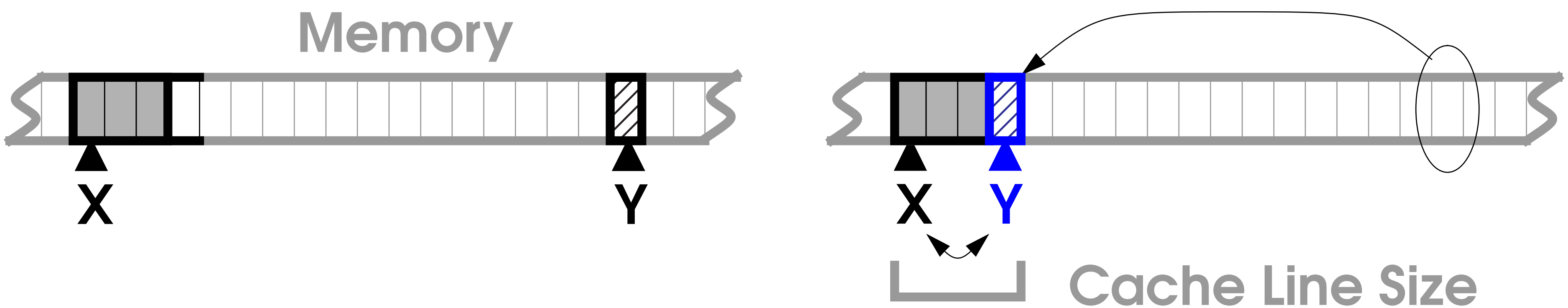
- Choose the address of data-elements
  - To improve reference patterns locality



- Only a preliminary study...
  - Cache tuning by moving data never tried:
  - Evaluating potential needs a new miss classification
  - Distribution heuristics need metrics to quantify locality
  - We show a heuristic for scalars

Data Distribution...

- Choosing the address of data-elements
    - Get better reference pattern locality



    - Untractable ?

- Our study
    - Are Maximum Expected Gains Worthwhile ?
    - Can We Find a Better Distribution ?

Background

- Block: memory chunk aligned to load in a Cache Line.

- Perfect Cache: Fully-Associative, Min policy
  - LRU is not optimal
  - Min is optimal (Belady66) -uses an oracle-

- Miss Classification (Sugumar93)
  - Cold
  - Capacity
  - Conflict
    - Mapping
    - Replacement

Issues

- 90% cache is wasted (Burger95)
- Compilers ignore cache

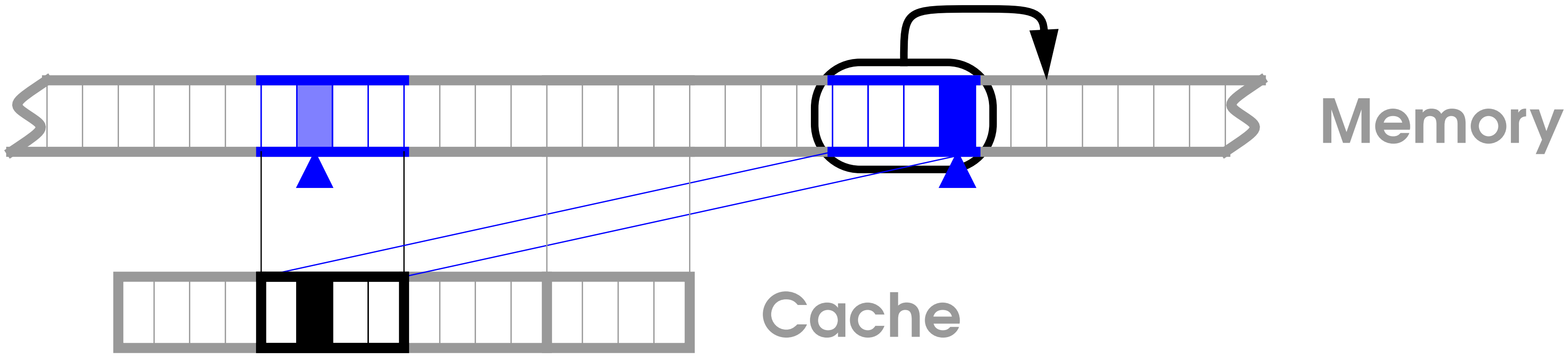
Solutions

- Smarter hardware ?
  - "Dumb" software may defeat optimizations
- Smarter software ?
  - Loop transforms only for scientific code
  - Data distribution for all codes

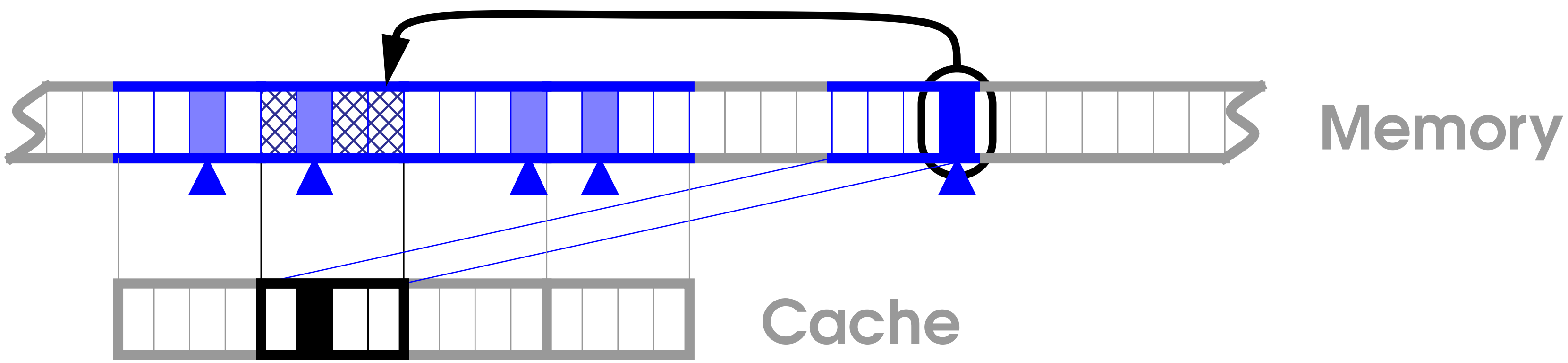
Data distribution gains ?

- **Fewer Conflicts**

  - Less mapping-conflict misses



- **Fewer Capacity Misses**



New Miss Classification

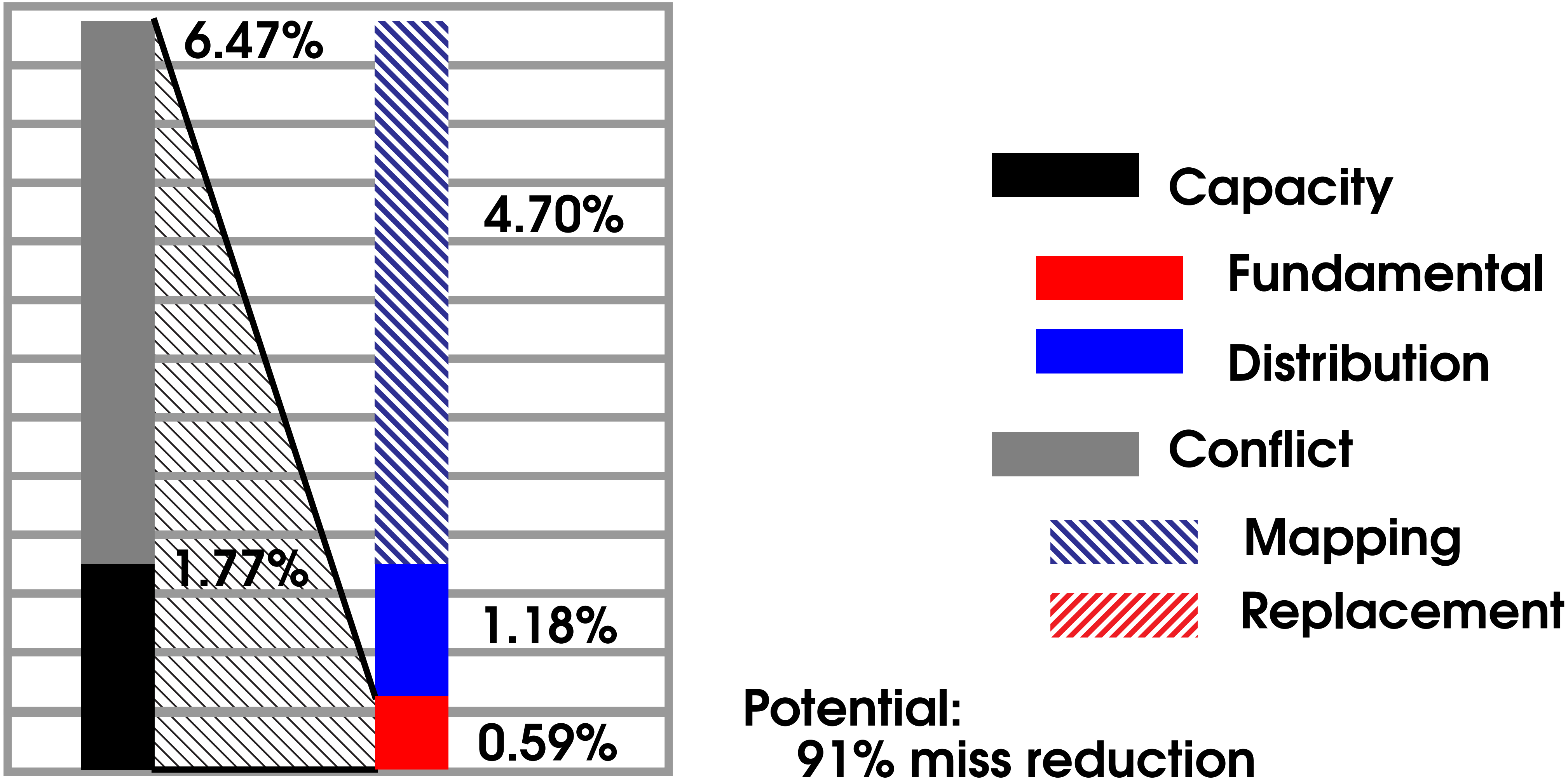
- There exists an optimal data-layout
  - Cannot find it
  - We can estimate its miss rate (lower bound)
  - Fundamental miss ratio

- New Miss Classification
  - refines capacity misses

- Cold
- Capacity
  - Fundamental
  - Distribution
- Conflict
  - Replacement
  - Mapping

Potential (GO, specint95, 16KB)

- 16KB Direct-Mapped Cache, Original data layout



Potential:
91% miss reduction

Feasability

- Trying all layouts is not possible

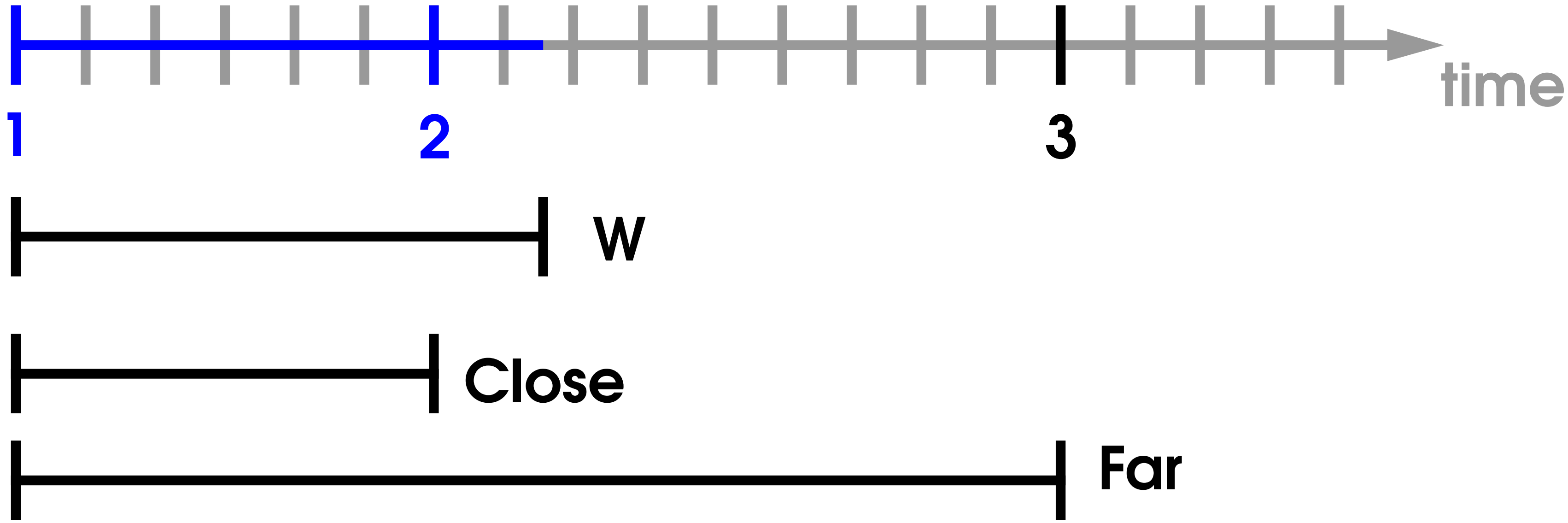
- We Need a Method for Data-Distribution
  - Automatic
  - Select a good layout
  - One pass

- We must evaluate potential Spatial Locality between elements
  - We must define Metrics

- And Make an Heuristic

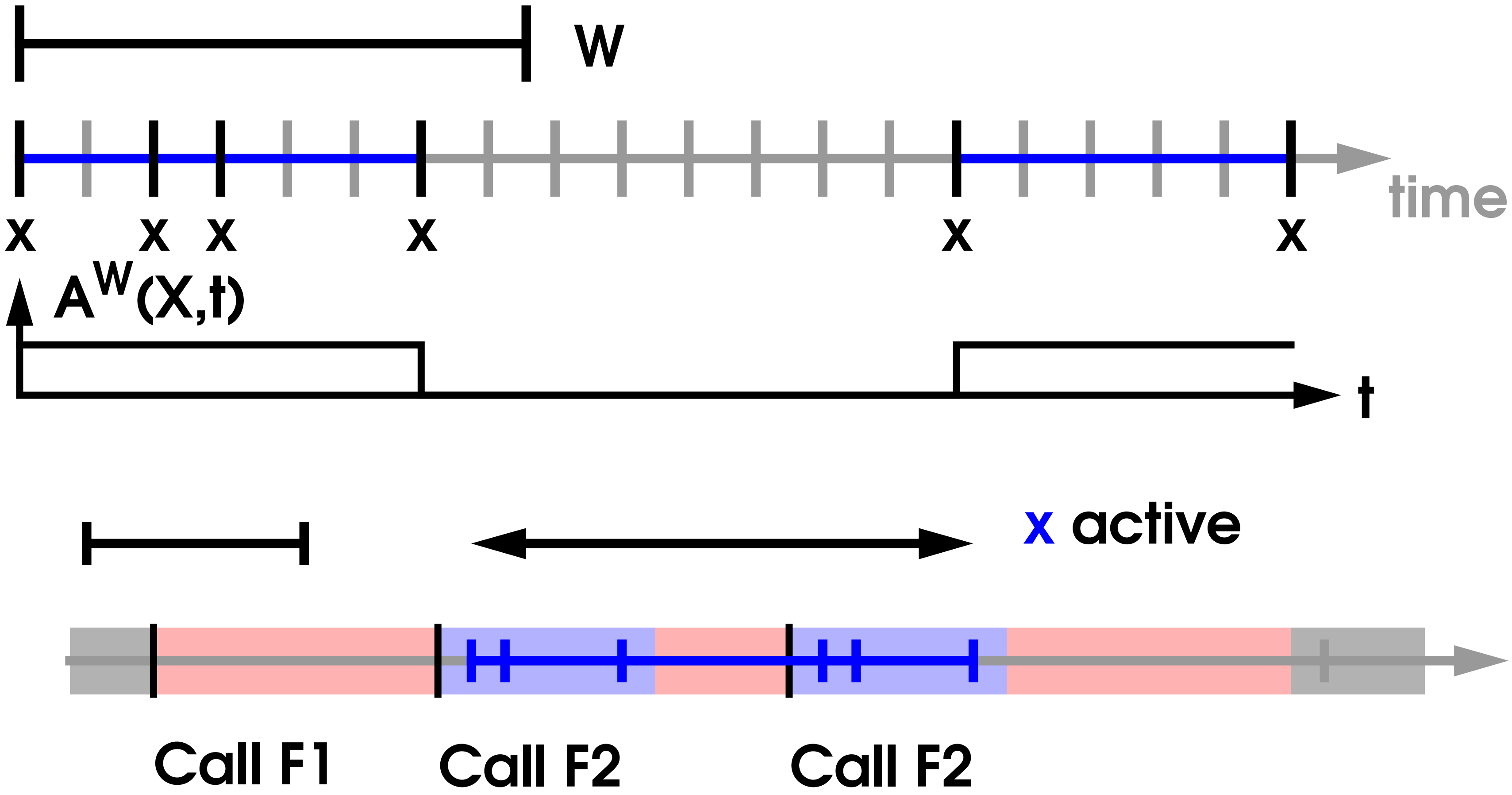
Time Window

- ||W|| determines distance in time
  - Ref. 2 is Close to Ref. 1
  - Ref. 3 is Far to Ref. 1



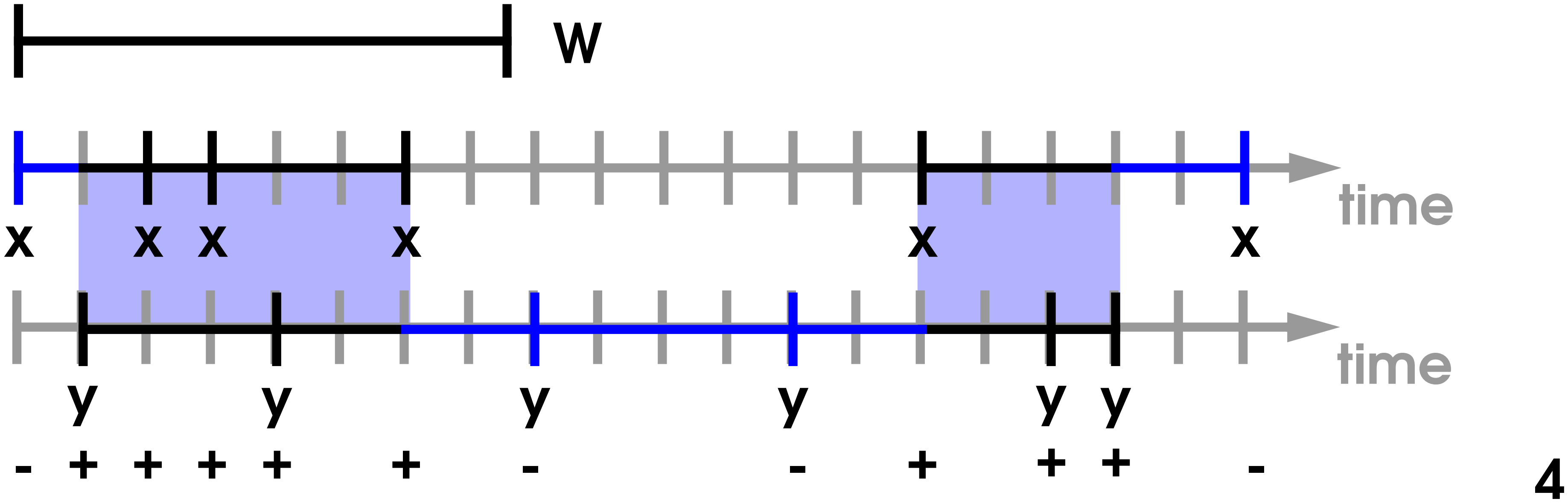
Activity

- $A^W(X,t)$=1 if an element is in use, 0 otherwise.
  - Liveness not OK(between R-W, X is usefull)

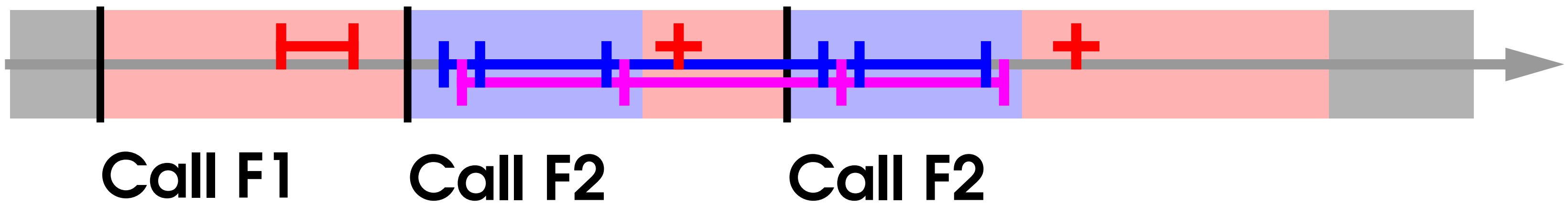


Correlation

- $C^W(X,Y) = \text{Sum}_t\ \text{Ref}(X)\ \text{while}\ A^W(Y,t) - \text{Sum}_t\ \text{Ref}(X)\ \text{while}\ !A^W(Y,t)\ldots$

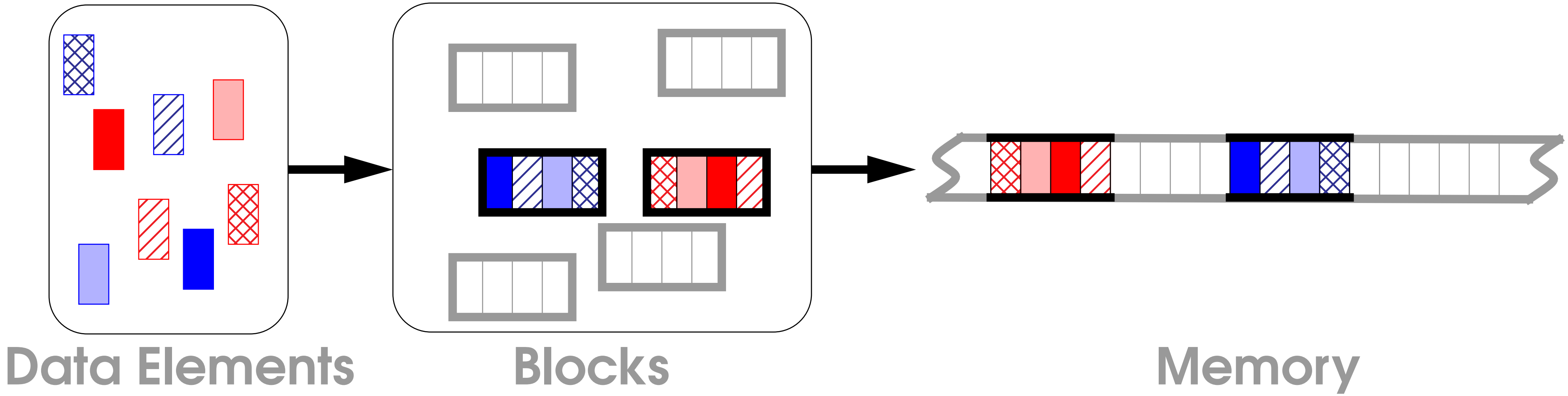


$C^W(x,y) > C^W(y,z)$



Scalar Data Distribution Heuristic

- Analyse Trace & Compute $A^W(X,t)$ for all X
- Group elements into blocks using $C^W(X,Y)$
  - Most Correlated elements
  - Minimize Distribution-capacity misses
- Place blocks in memory
  - Use cache mapping properties (Direct-Mapping)
  - Most Correlated map to distinct lines
  - Minimizes Mapping-conflict misses



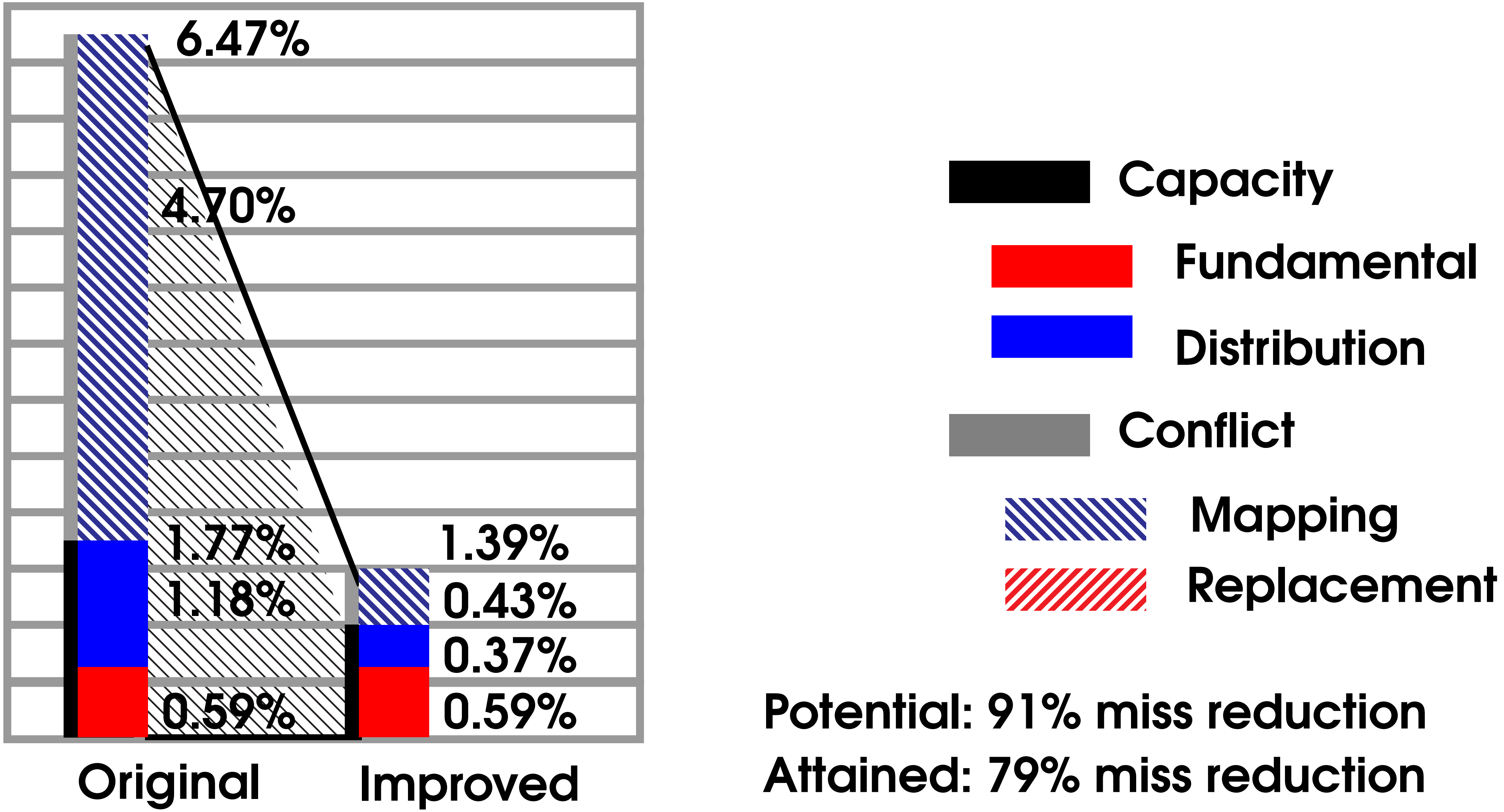
Heuristic Pros & Cons

- Problems
  - Profile based
  - C^W computed for all pairs (n^2),(b^2)
  - Scalars only
- But
  - Not necessarily profile: needs future
  - Compiler constraits reduce choice
  - Principle can be adapted to complex structures
- Advantages
  - Software only
  - Layout found after 1 analysis
  - Automatic
  - Works for D.M, S.A, F.A. Caches
  - Works for larger cache sizes

Experimentation

- 16KB cache, 32B line, D.M.
- Cache Simulation
- On Trace
  - No compiler layout constraints
  - All references are considered as scalar references
- Go (SpectInt'95)
  - fixplaylevel=20
  - gcc -O
  - go 10 10
  - full trace

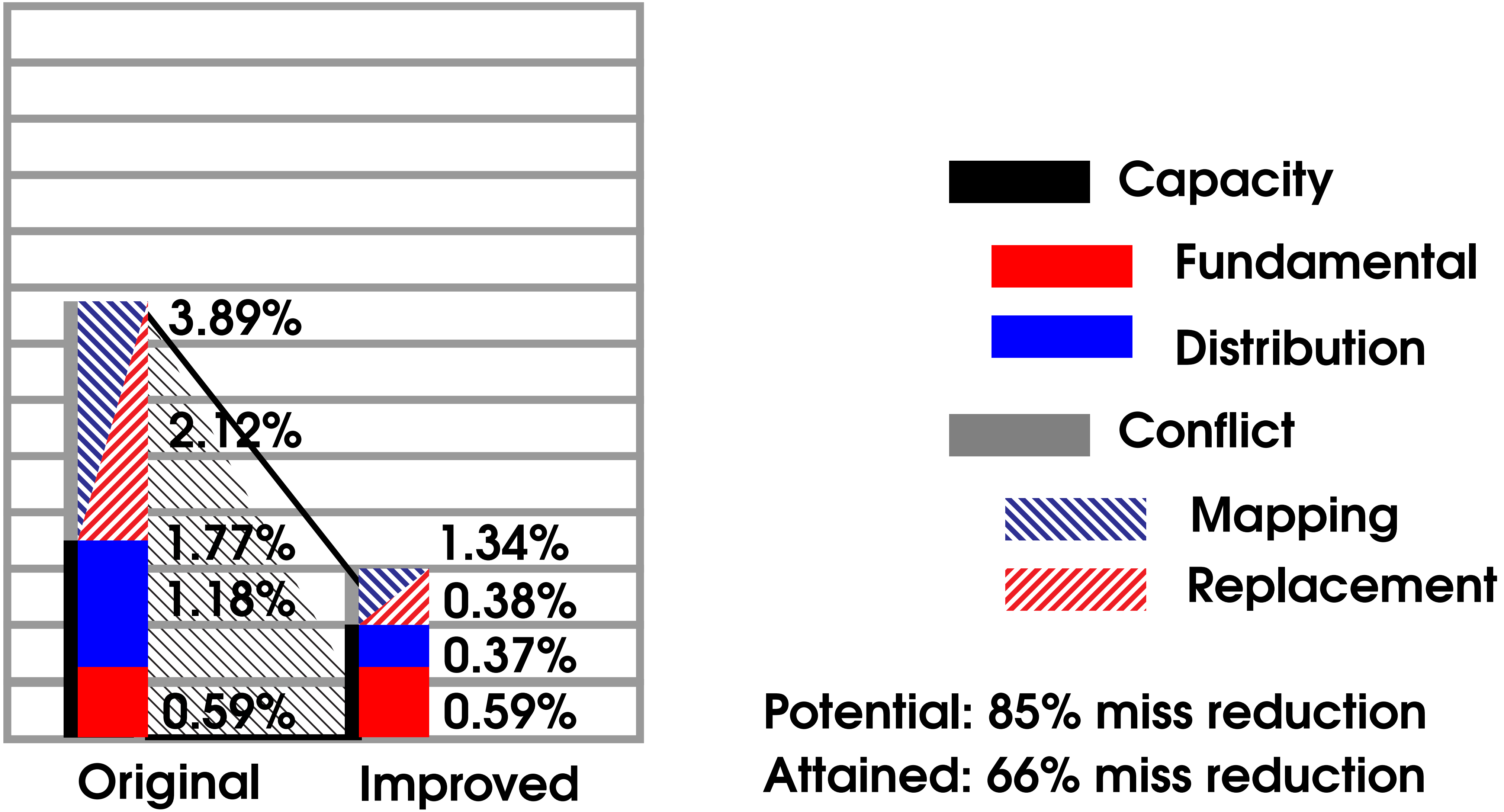
Optimization results (D.M.)



Potential: 91% miss reduction

Attained: 79% miss reduction

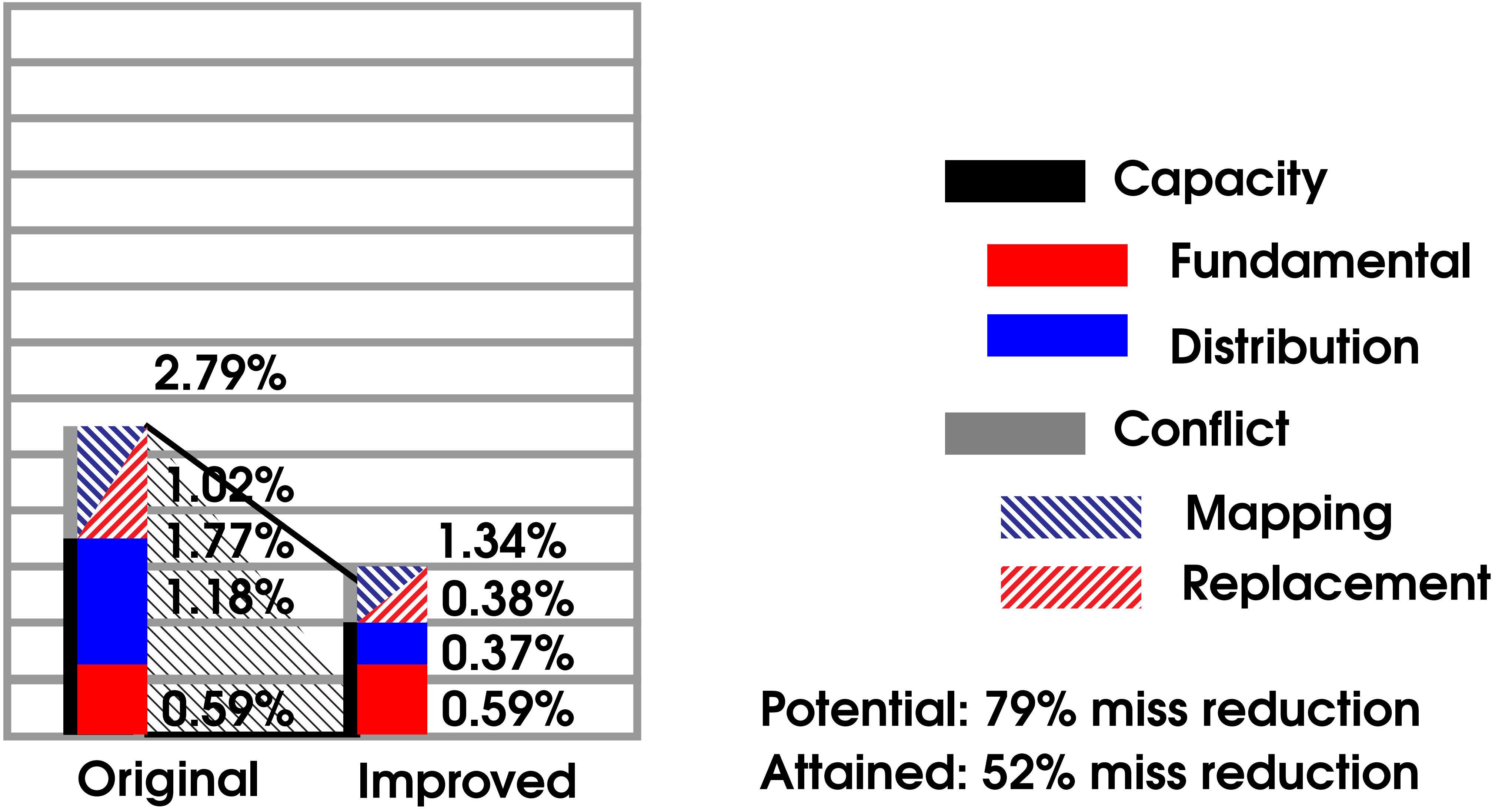
Optimization results (2 S.A.)



Potential: 85% miss reduction

Attained: 66% miss reduction

Optimization Results (4 S.A.)



Potential: 79% miss reduction

Attained: 52% miss reduction

Conclusion

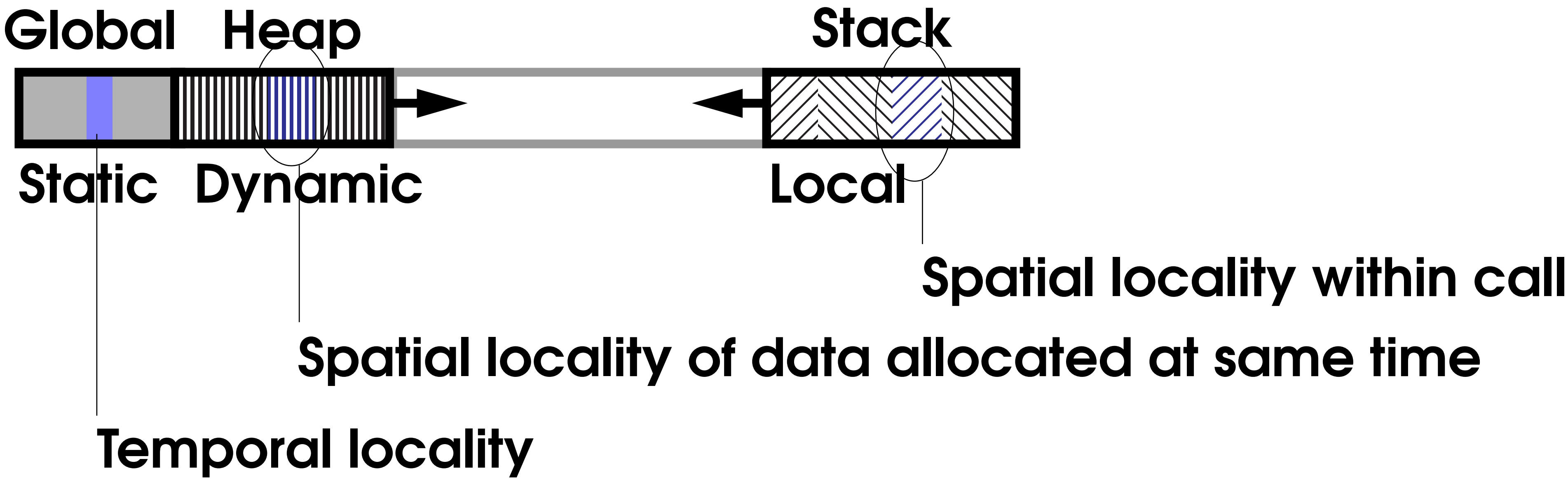
- We
  - Defined Data Distribution
  - Updated the miss classification for data layout evaluation
  - Gave metrics to quantify locality
  - Showed an automatic/1 pass scalar layout heuristic
- Data distribution
  - Has a big potential to tap
  - Complement to loop transformation techniques
  - Works with D.M./S.A./F.A. caches
  - Works with bigger caches
- To be done...
  - Prove it can be applied in real applications
  - Find heuristics for compatible data types
  - Replace profiling by source code analysis or ...

Conclusion

- High potential for Miss reduction
- Range of hardware & software optimizations reduce miss
- Data Distribution is a necessary complement
  - Fills the gap between memory hierarchy and compilers
  - Complement to other software techniques (tiling...)
- This is a preliminary study
  - Heuristics for all data types
  - Use Static Code Analysis or Reference Prop. Knowledge

Future Work

- Improve Compiler “Accidental” Locality

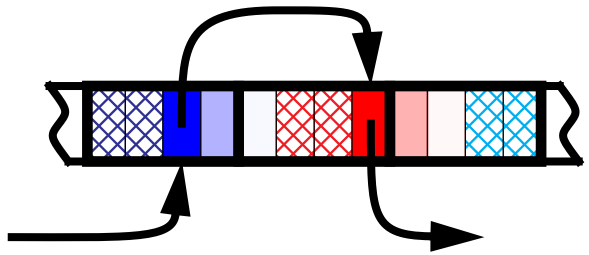


- Example: Dynamic Allocation
  - Use knowledge on data usage (Linked List)
  - Modified allocation library
  - Can be automated

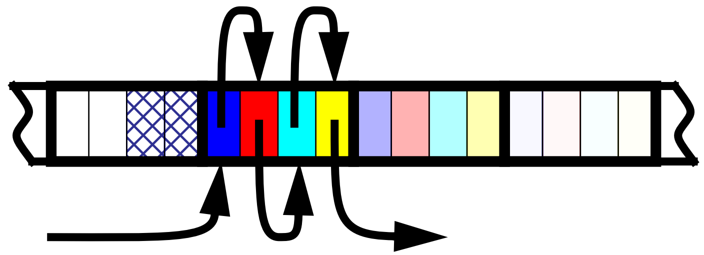
Interleaving Example

User Defined Struct:

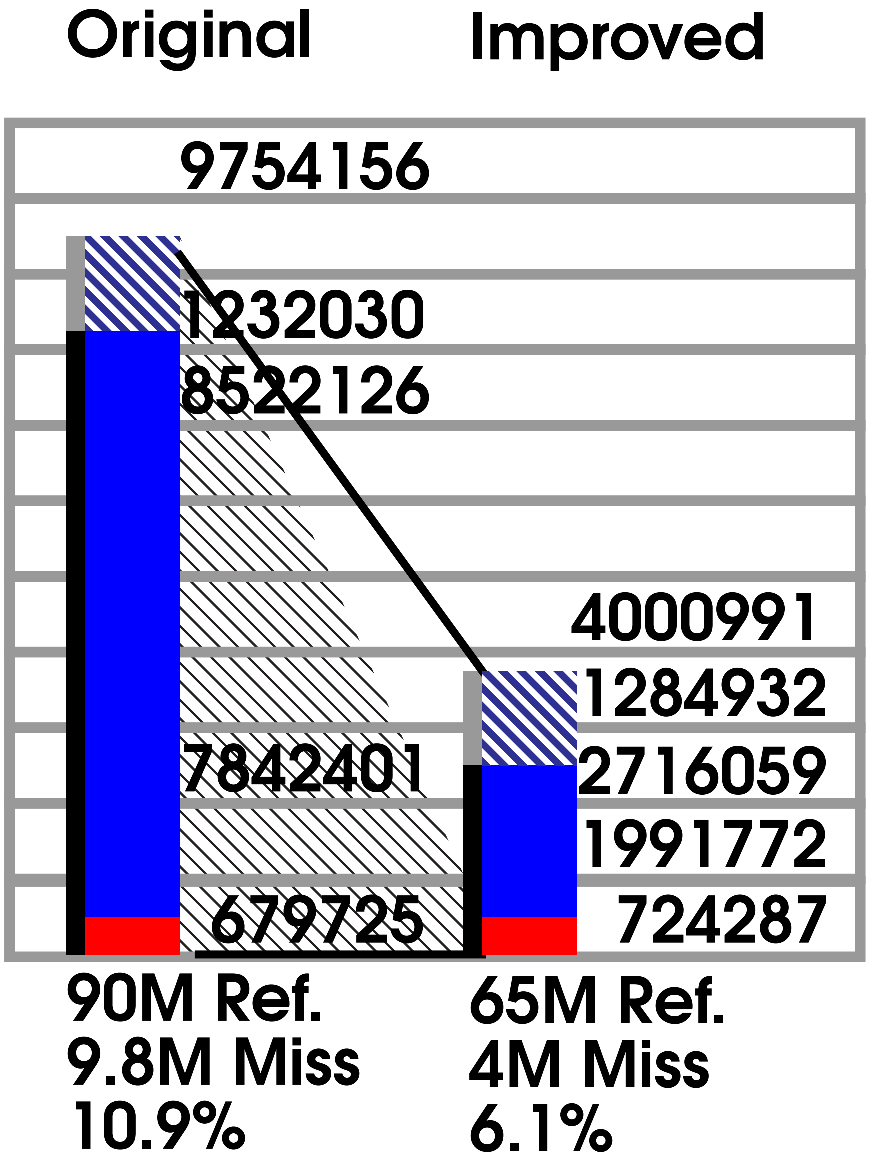
Malloc Layout:



Improved Alloc.:



Interleaving Results (D.M. 16K)



Potential: 93% miss reduction

Attained: 59% miss reduction

Speedup: 1.5 (9.63s to 6.44s)

Experimental Results

- 16KB Direct-Mapped Cache, 32B Lines