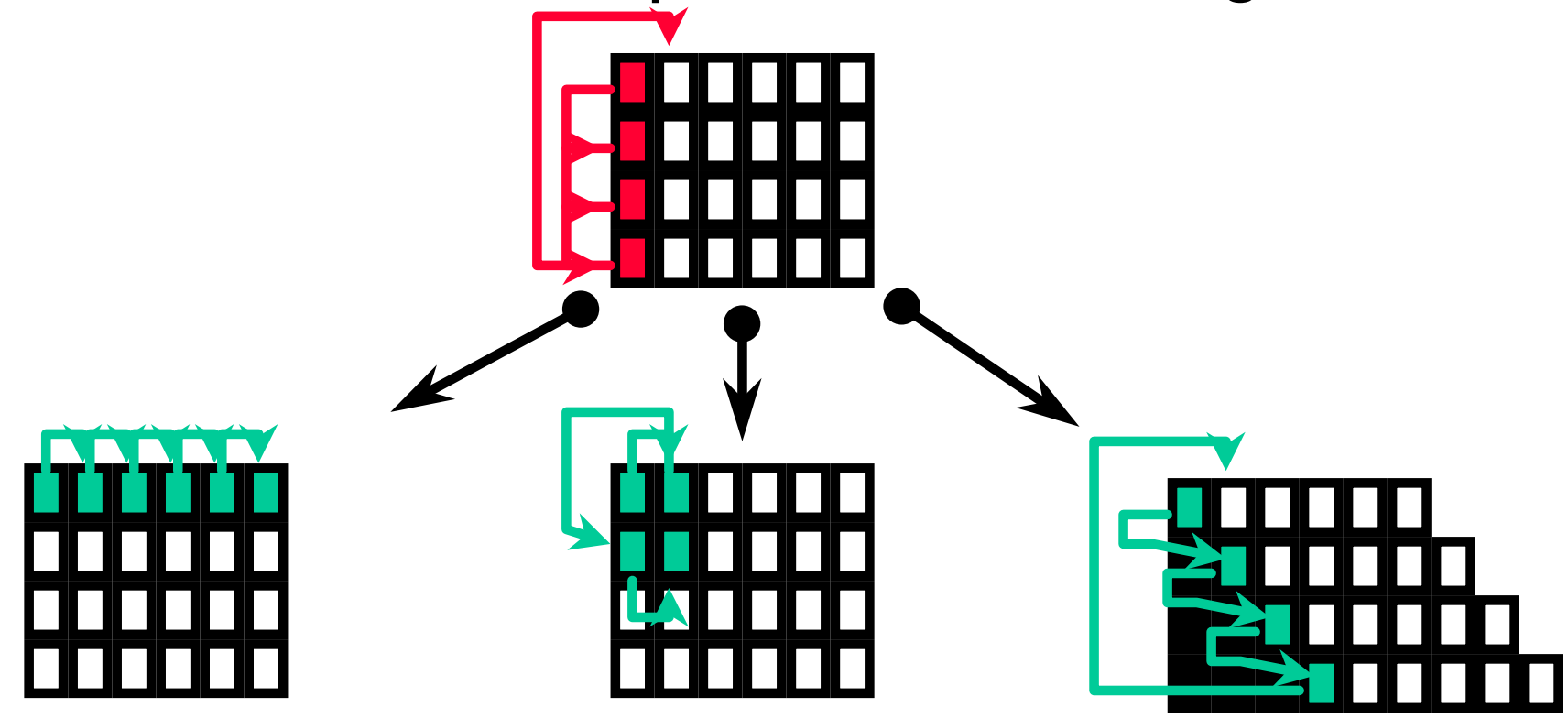
Improving Cache Behavior of Dynamically Allocated Data Structures

dtruong@irisa.fr | truong@enssat.fr
bodin@irisa.fr
seznec@irisa.fr

IRISA

Arrays have regular reference patterns: Solutions exist to improve cache usage...

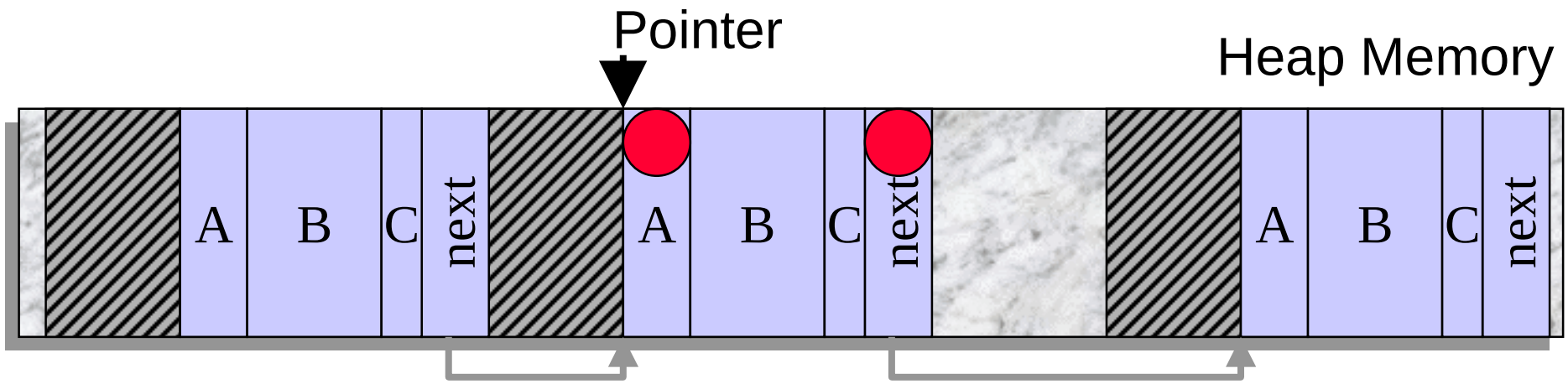


Interchange Tiling Padding

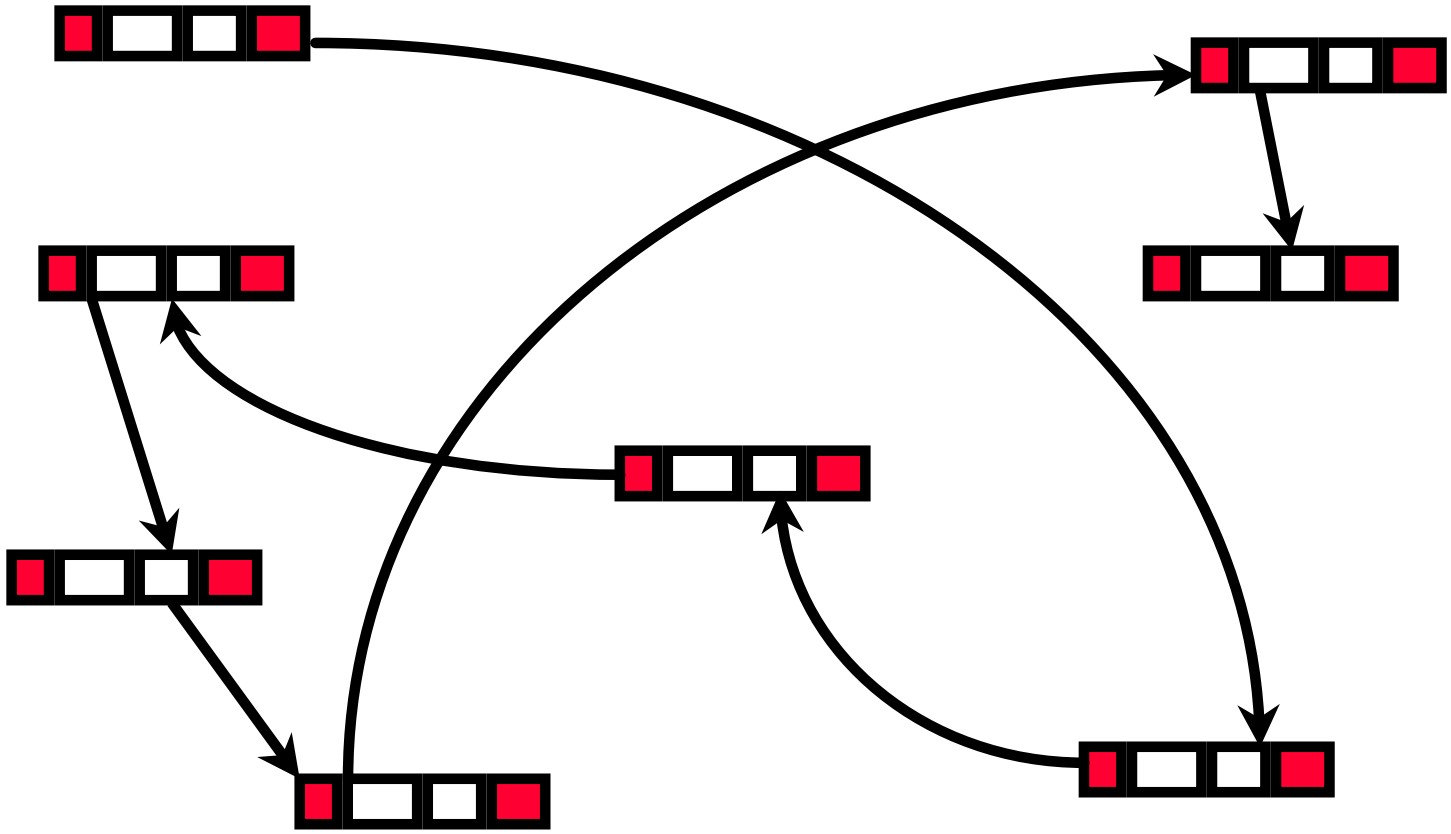
But for heterogenous data structures?..

```
struct Foo {
        int A;
        char B[8];
        short C;
        struct Foo * next;
};
```

Frequently used



Many instances allocated/used



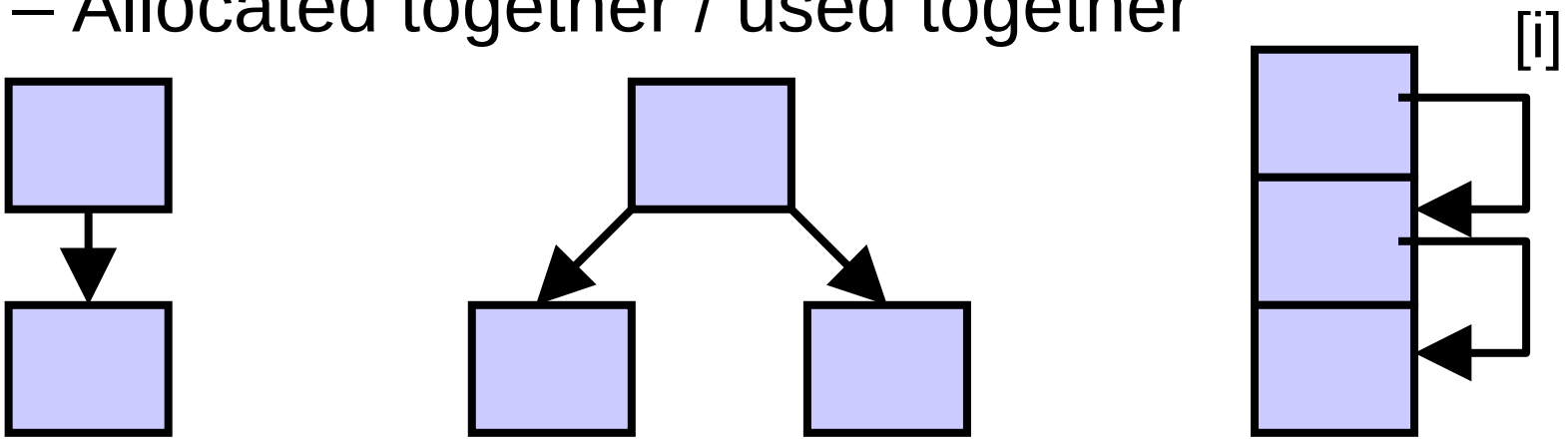
- Can we improve locality?

Our solutions

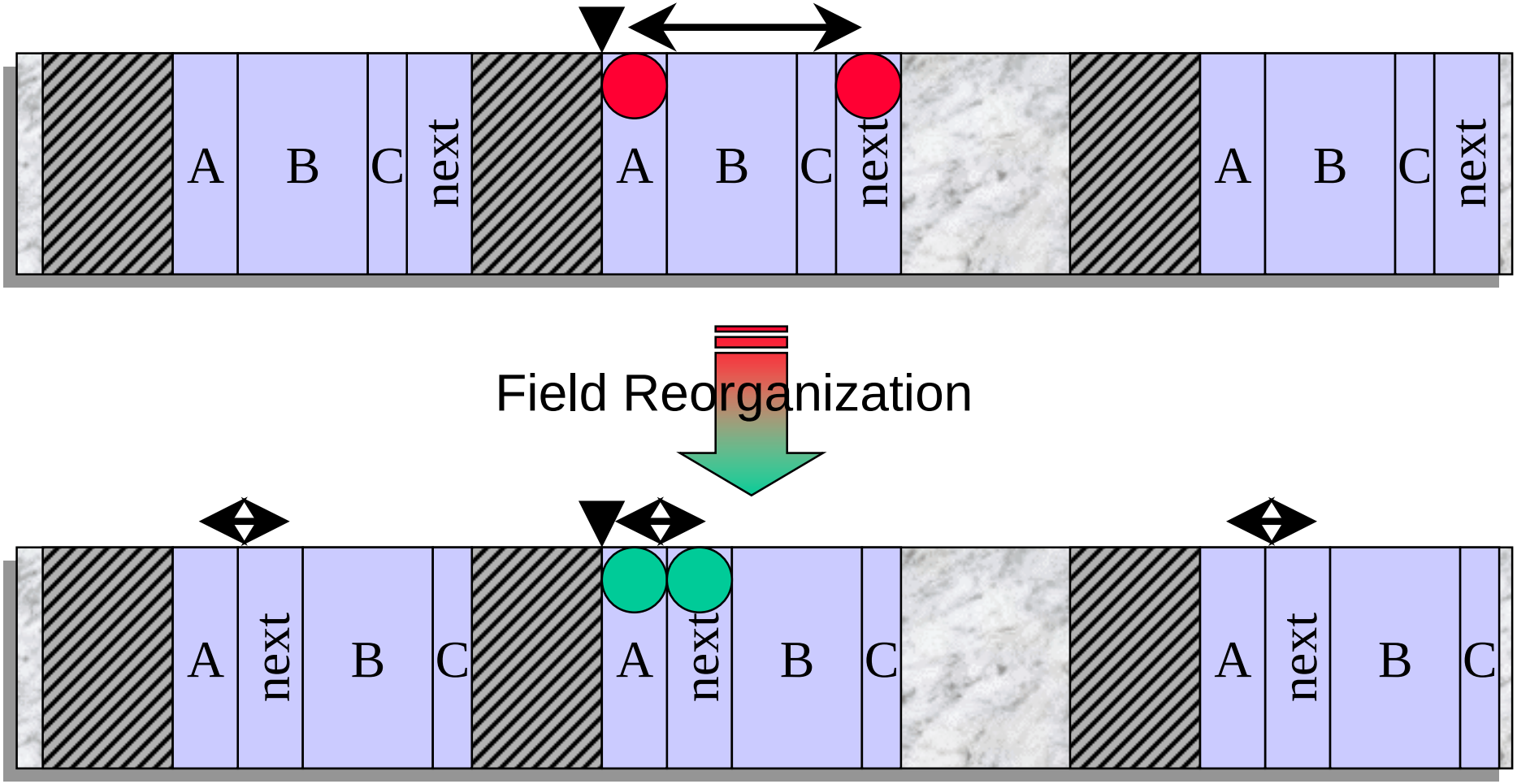
- Layout optimizations
  - Field reorganization
  - Instance interleaving
- Toolbox implementation approach in C
  - Profile based
  - Special allocation library
  - Programmer responsability

Locality of programs

- Few code lines run frequently:
  - Few structures fields are touched
  - Same fields in all instances
  - Instances referenced in sequence
  - Allocated together / used together

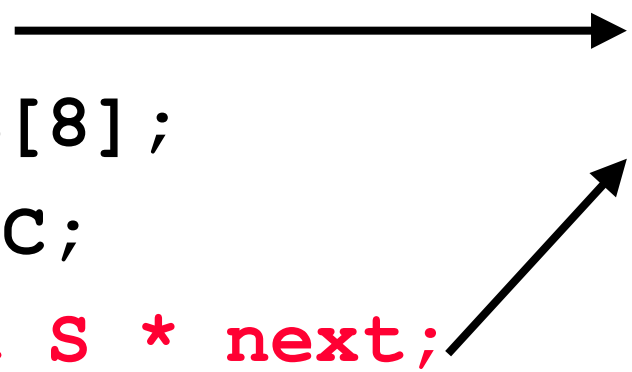


Field Reorganization

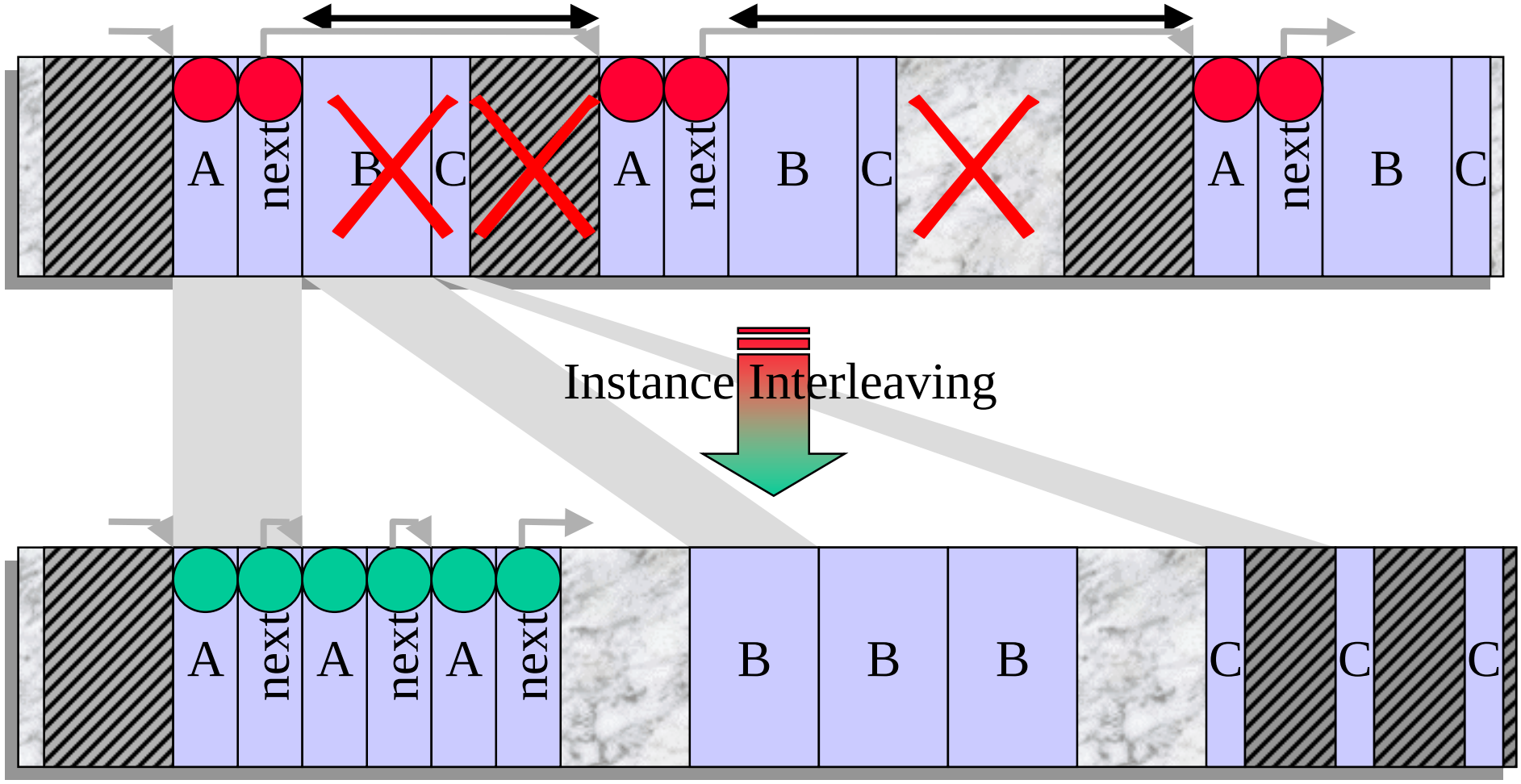


Field Reorganization

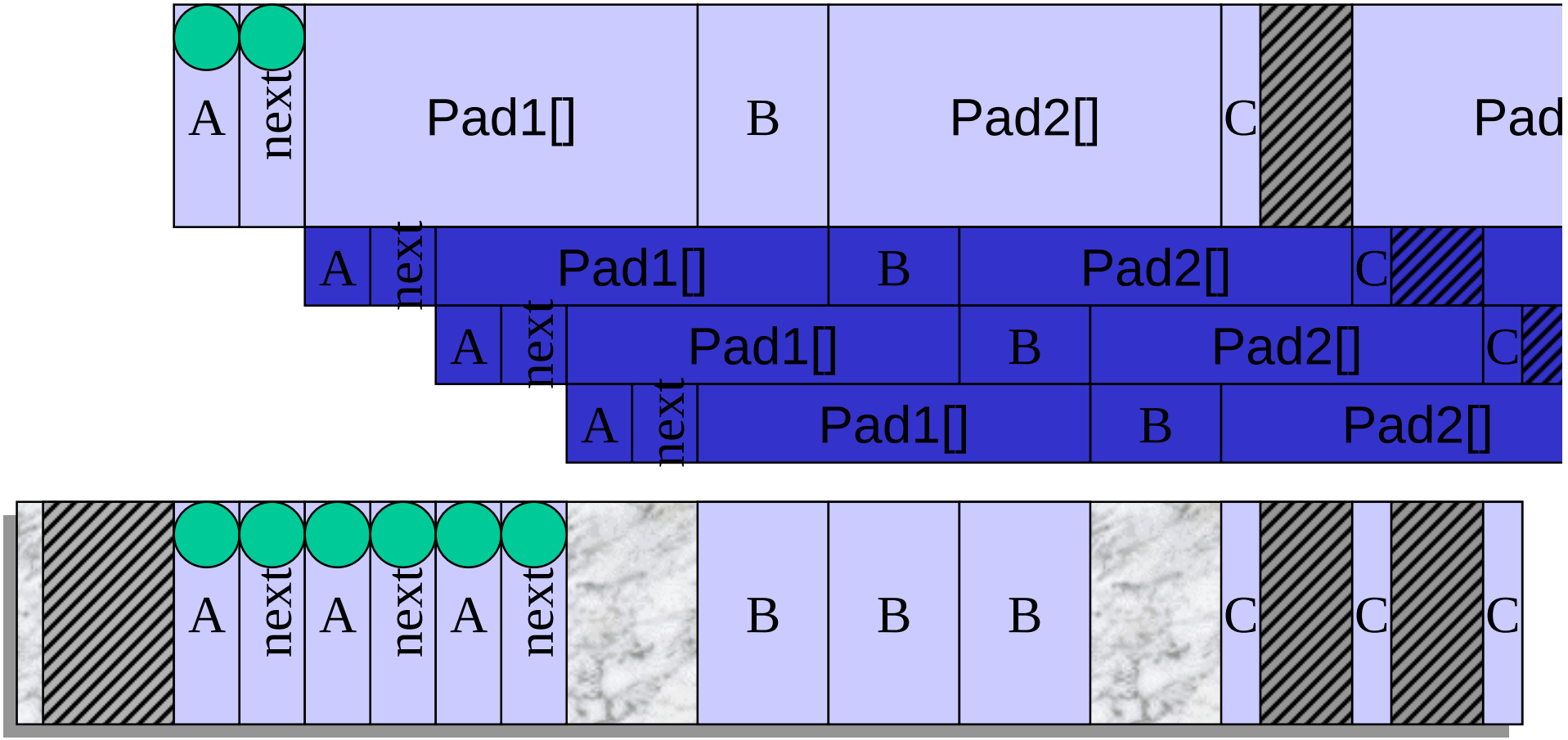
```
struct S {                    struct S {
   int A;                        int A;
   char B[8];                    struct S * next;
   short C;                      char B[8];
   struct S * next;              short C;
};                            };

struct S * Ptr;               struct S * Ptr;
Ptr= (struct S*) malloc();    Ptr= (struct S*) malloc();
```



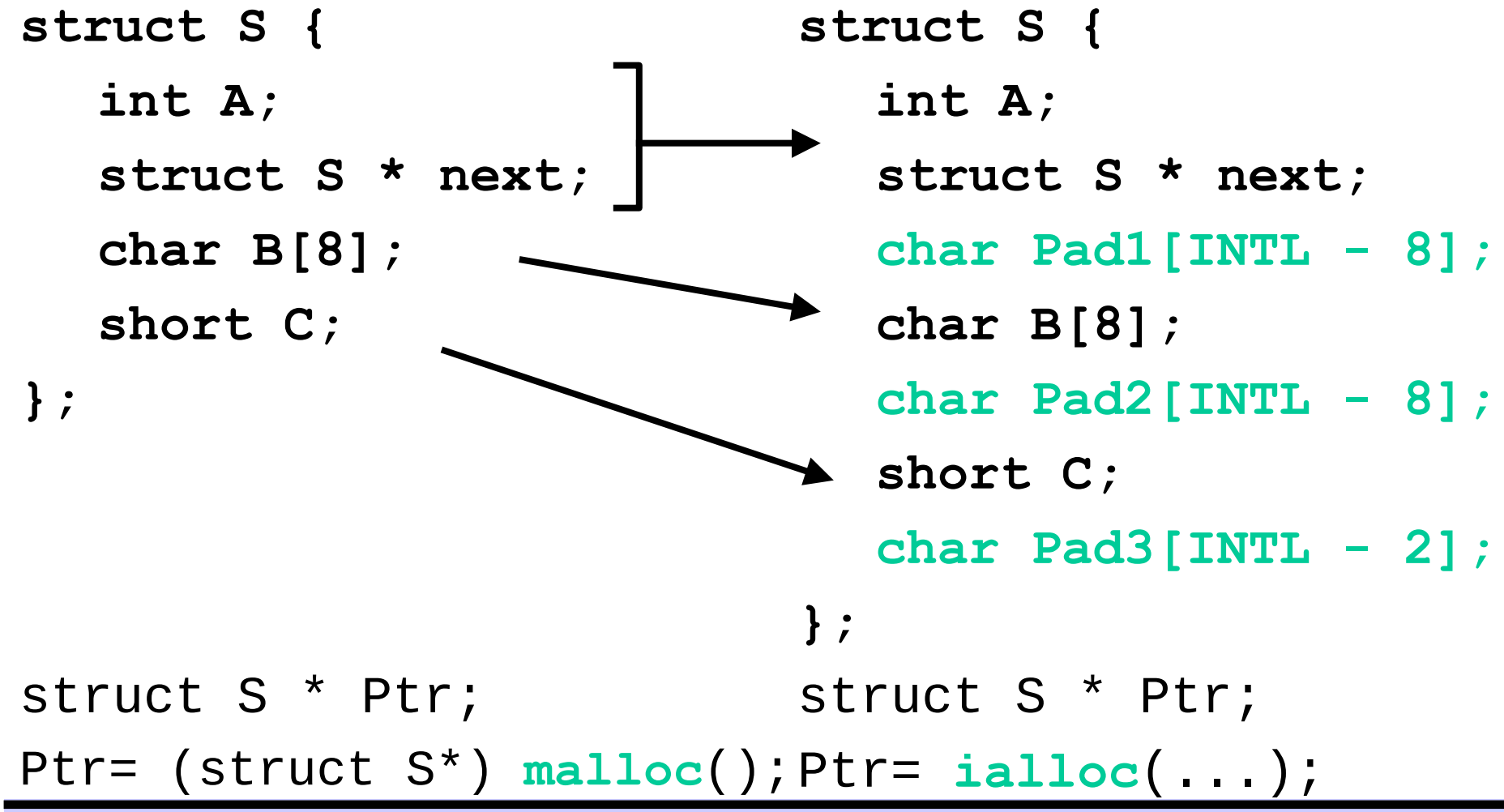
Instance Interleaving



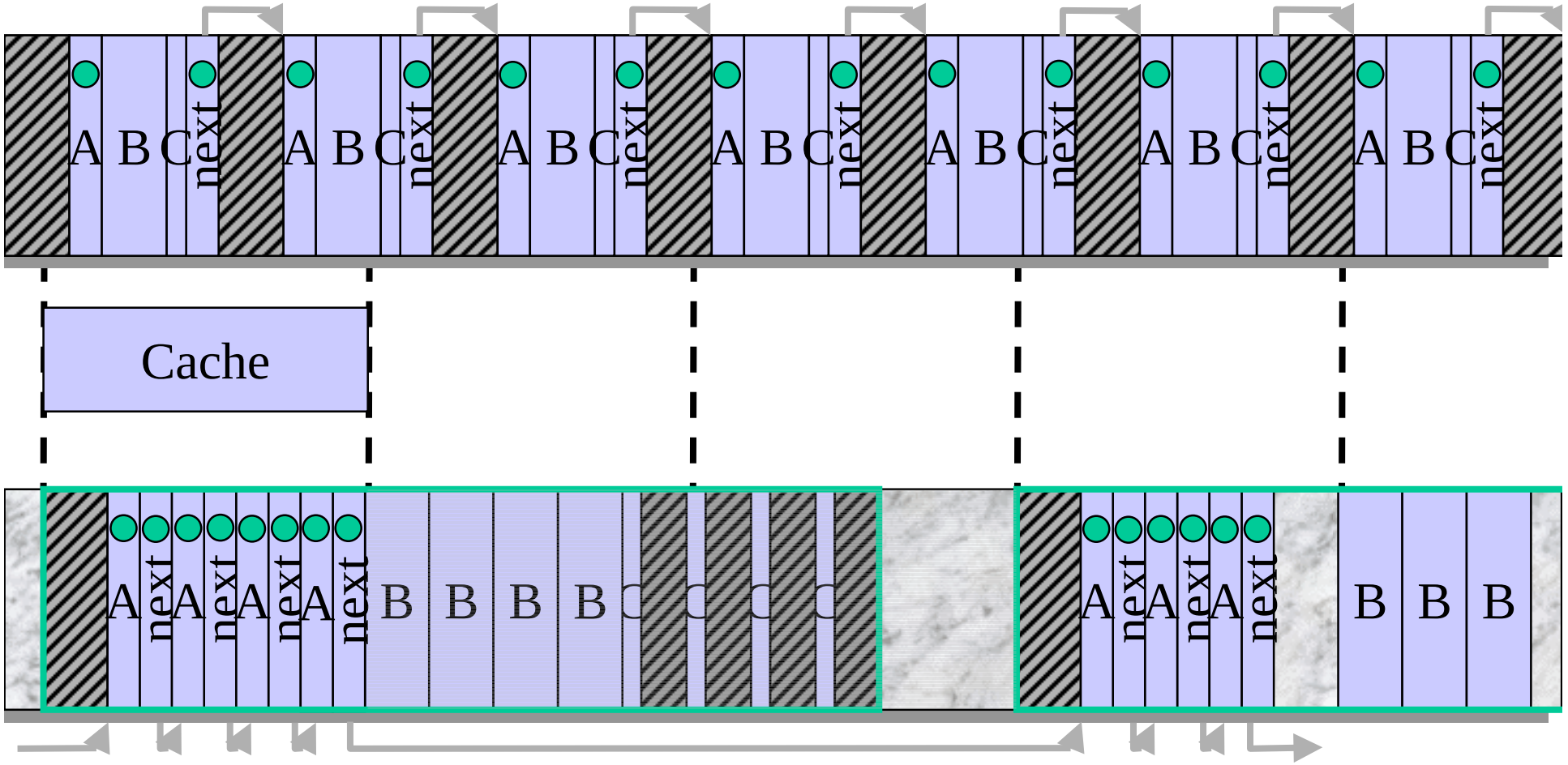
Instance Interleaving



Instance Interleaving



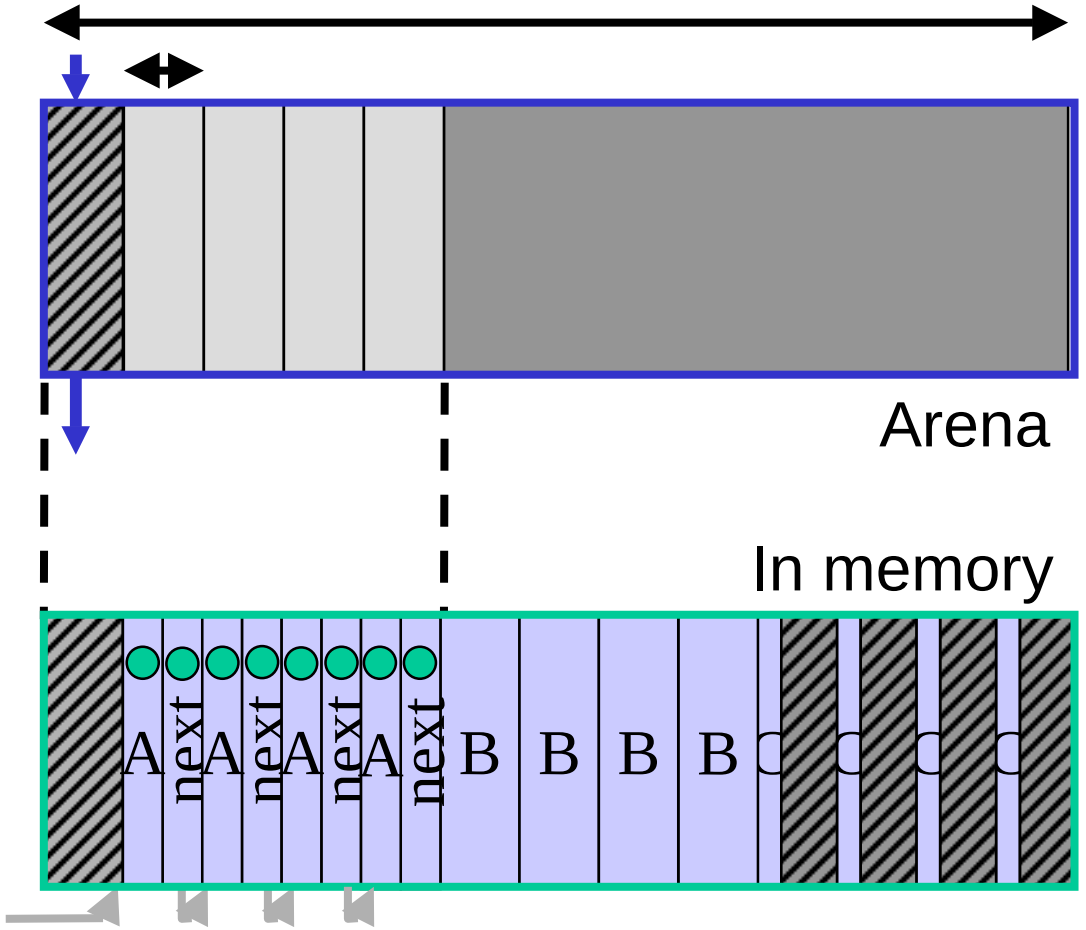
Instance Interleaving



Ialloc library

```
Ialloc(type,chunk,idx);
```

- Auto. mem. Managt
  - built with arenas
  - Auto arena usage
  - Auto arena creation
- Toolbox
  - Ialloc.c, ialloc.h…
  - Manual usage

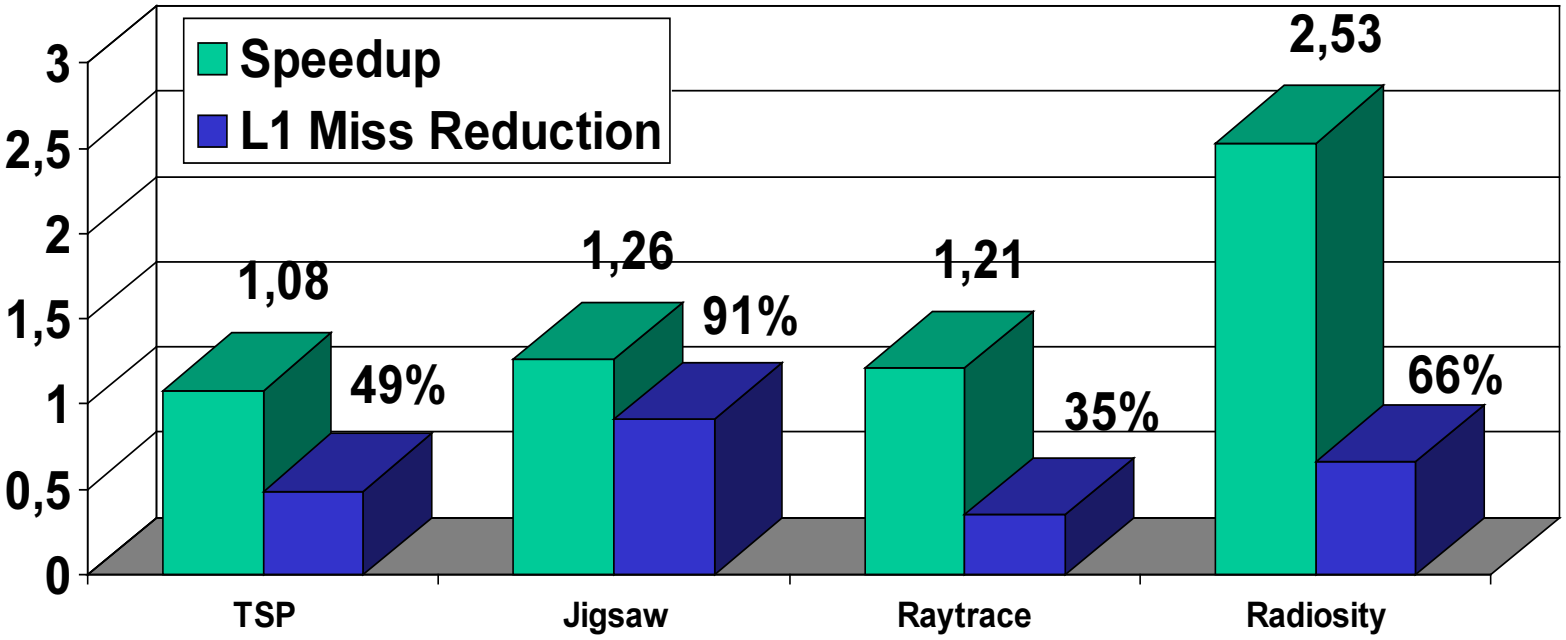


Profile-based optimization

- We use SGI 's Speedshop to profile
  - tells if & which cache slows program
  - shows source lines causing most misses
- We optimize layout by hand
  - to improve locality of refce of these lines
  - we change struct declaration & malloc calls

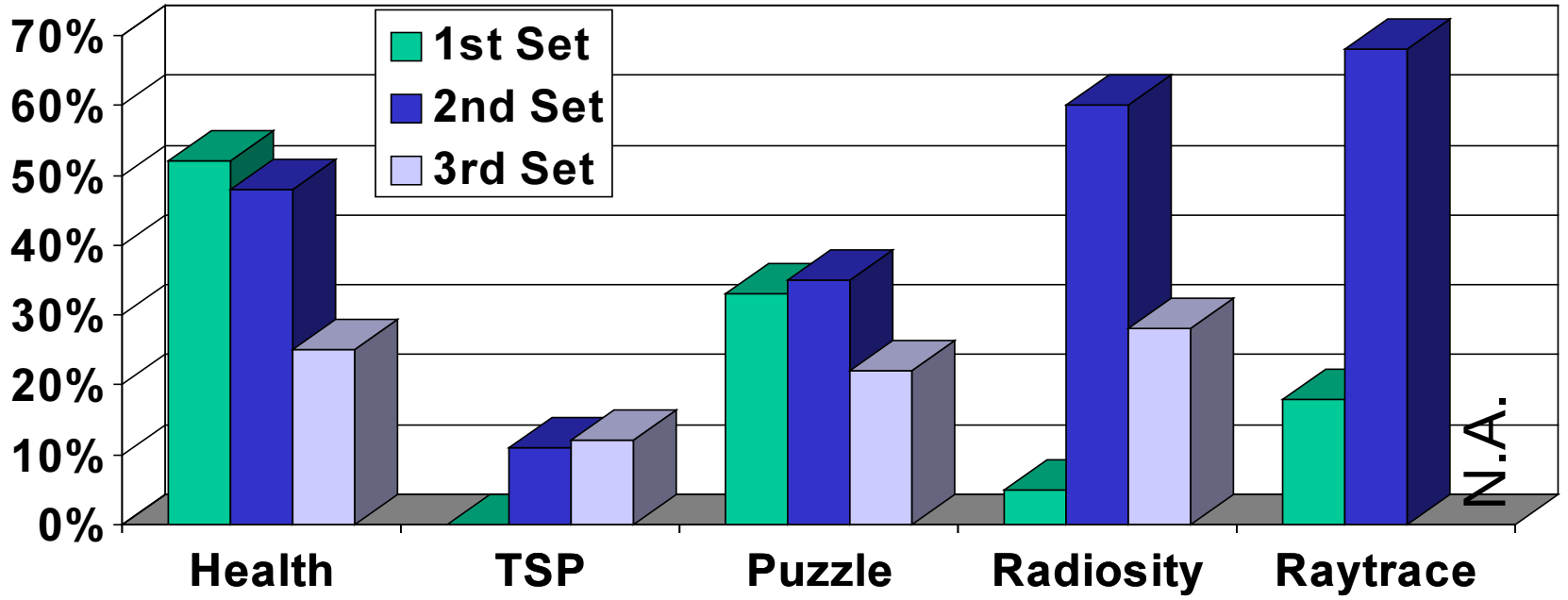
Results

- Speedups from 1.08 to 2.53 (user time)
- L1 Miss Reduction from 49% to 91%



Input Set Independance

- Gains with different input sets
- Larger data sets often yield more gain



Conclusion

- Hw independant
- Input independant
- Avail. profile tools
  - Hardware counters
- Easy to do if...
  - code built for *ialloc*
  - struct are simple
- Real speedups

- Manual optimization
- Syntax limitations
  - array of struct
  - sub-structures
- Hard to do if...
  - complex code
  - user custom alloc.
  - struct casting

What next ?

- Automation of structure layout
  - Hiding optim. in libraries (like STL?)
  - Precompiler tool to help manual optim.
  - Profile-feedback compiler
- New languages
  - Higher expressivity / decoupled from mach.
  - Compiler know data organization & usage
  - Removes pointer alias / casting