FRAMER: Efficient Per-Object Metadata Management

Myoung Jin Nam (Korea Univ.)

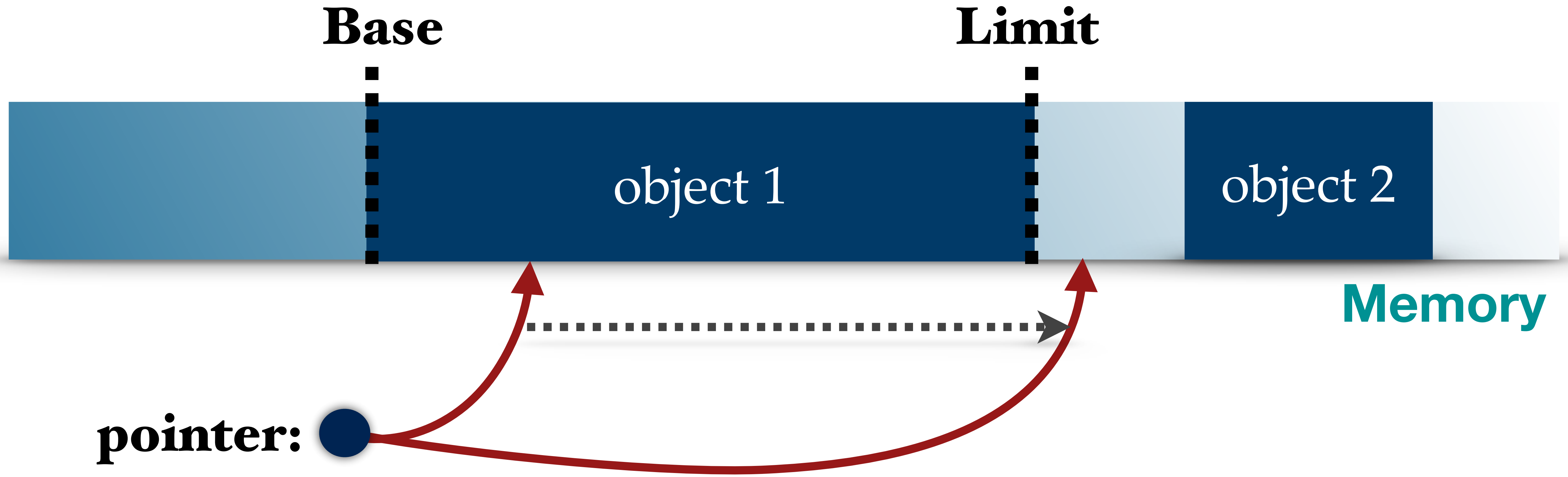
David Greaves (Cambridge Univ.)

Periklis Akritidis (Niometrics)

Memory Safety

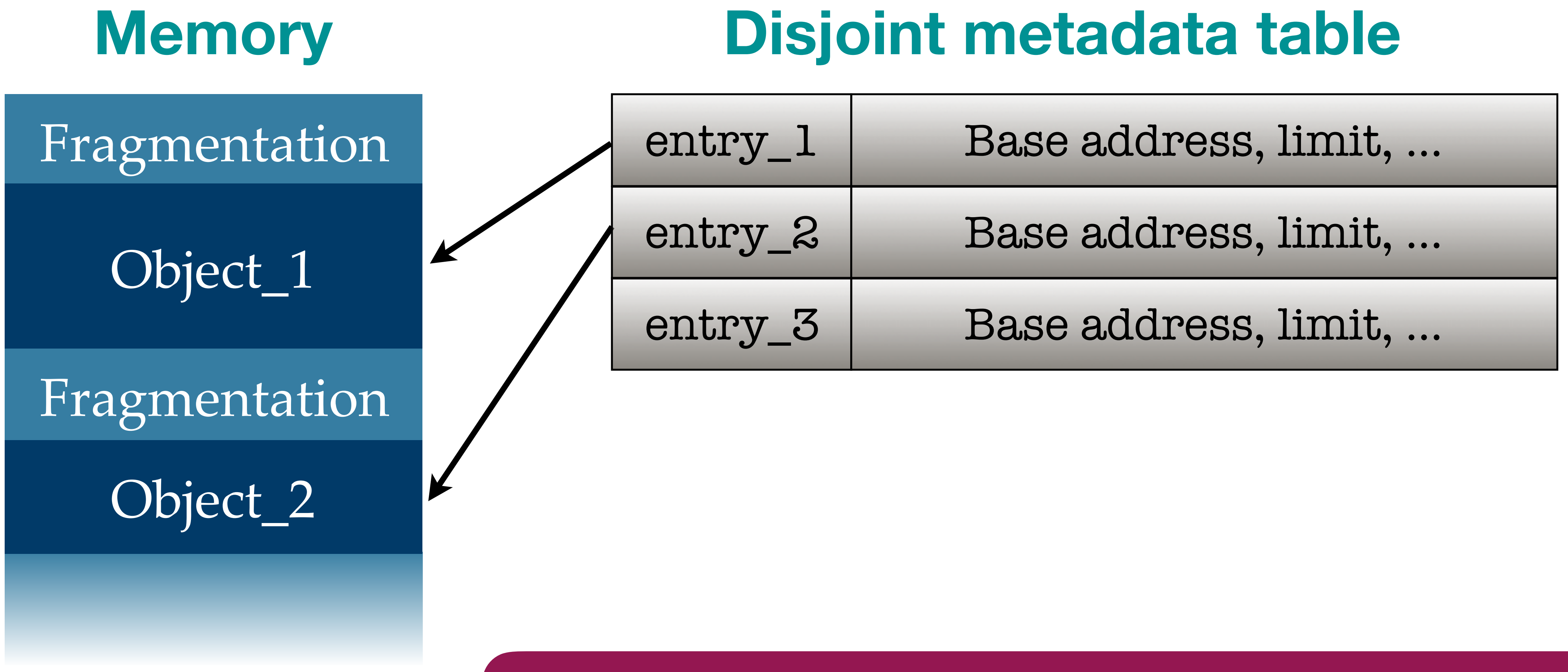
- A program execution is memory safe so long as memory access errors never occur:
  - Buffer overflows, null pointer dereference, use after free, use of uninitialized memory, illegal free
- Memory safety categories
  - Spatial memory safety
    - Stops out-of-bounds pointers. (buffer overflows)
  - Temporal memory safety
    - Stops dangling pointers (use-after-free, double-free)

Spatial Memory Safety



A pointer to be dereferenced at run-time is in-bound?

Tracking Pointers/Objects



Memory

Fragmentation

Object_1

Fragmentation

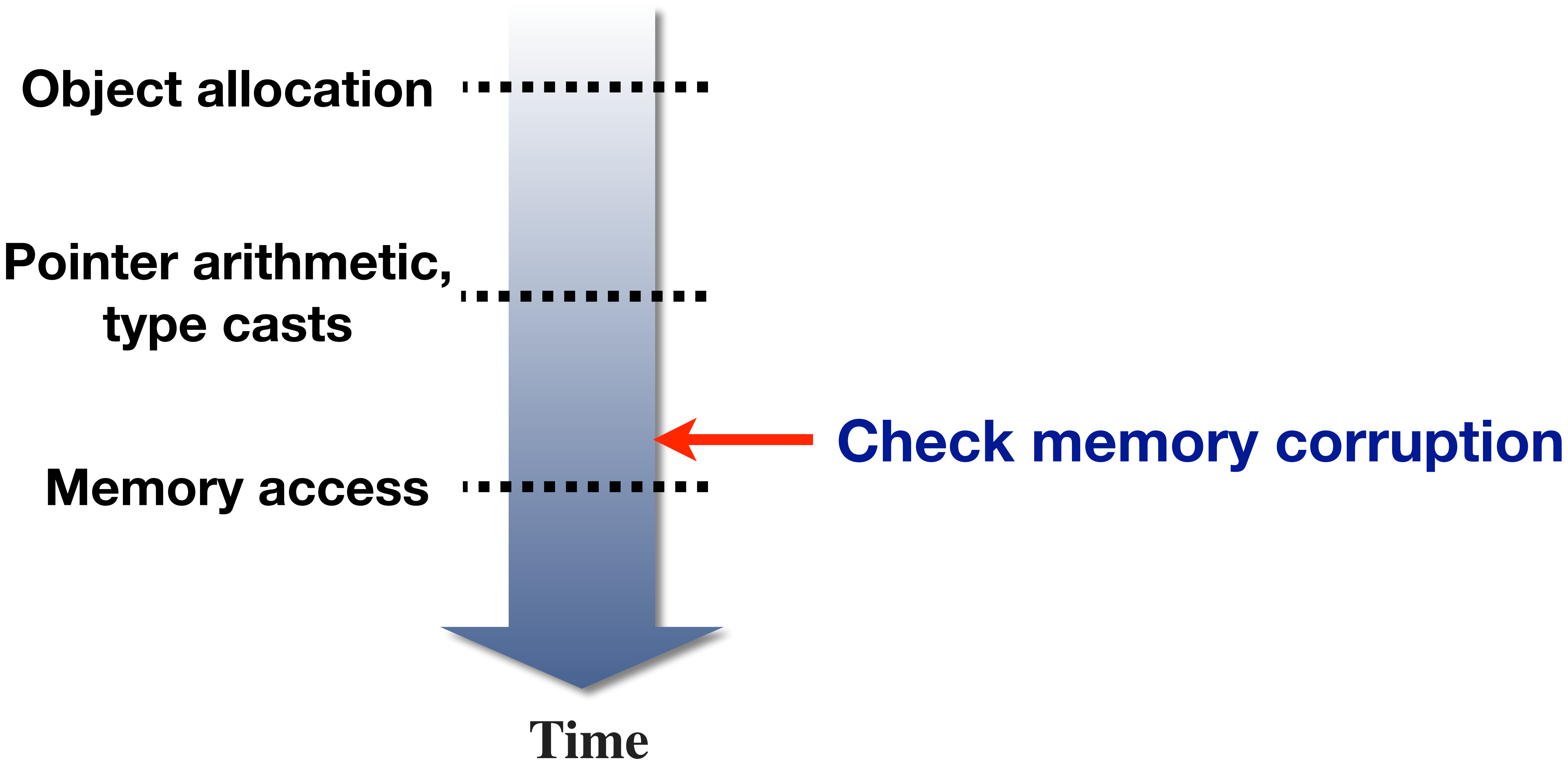
Object_2

Disjoint metadata table

entry_1	Base address, limit, …
entry_2	Base address, limit, …
entry_3	Base address, limit, …

Choice of data structure
➡ Zillions of objects (pointers) to track?

Runtime Checks



Object allocation

Pointer arithmetic, type casts

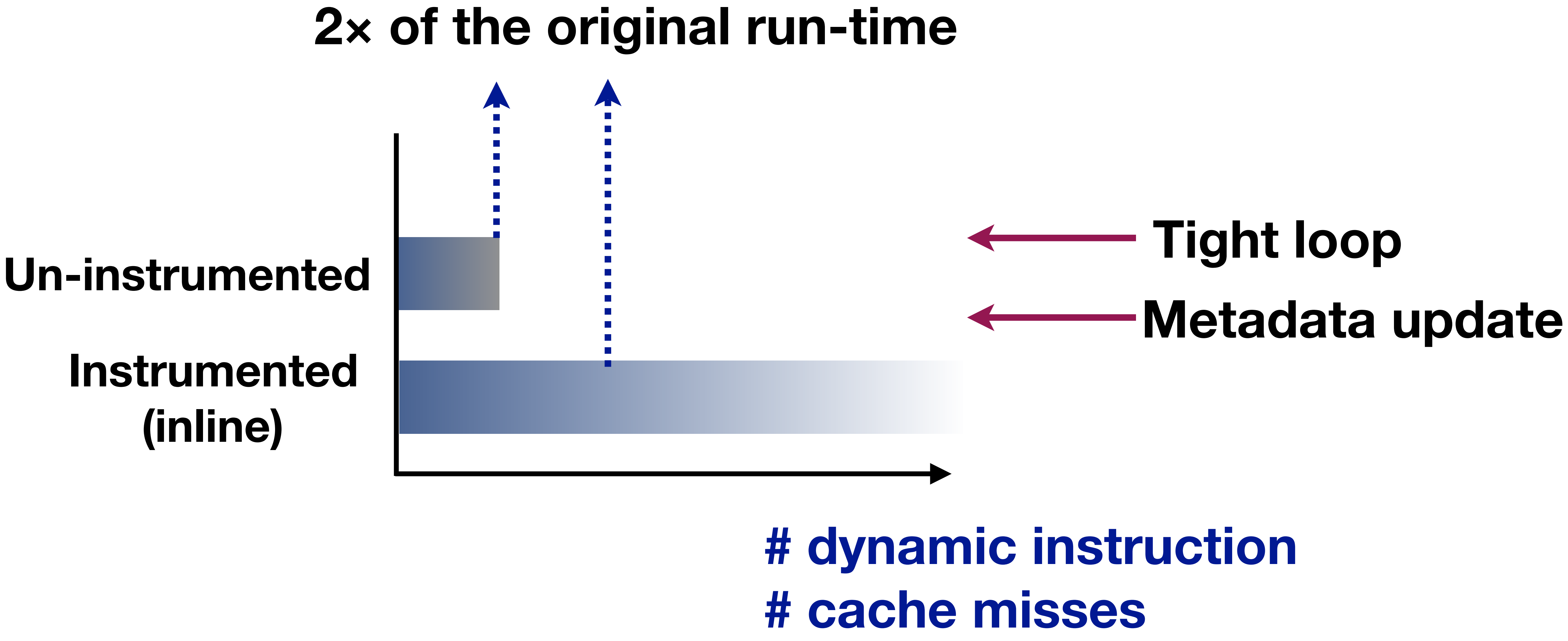
Memory access

Check memory corruption

Time

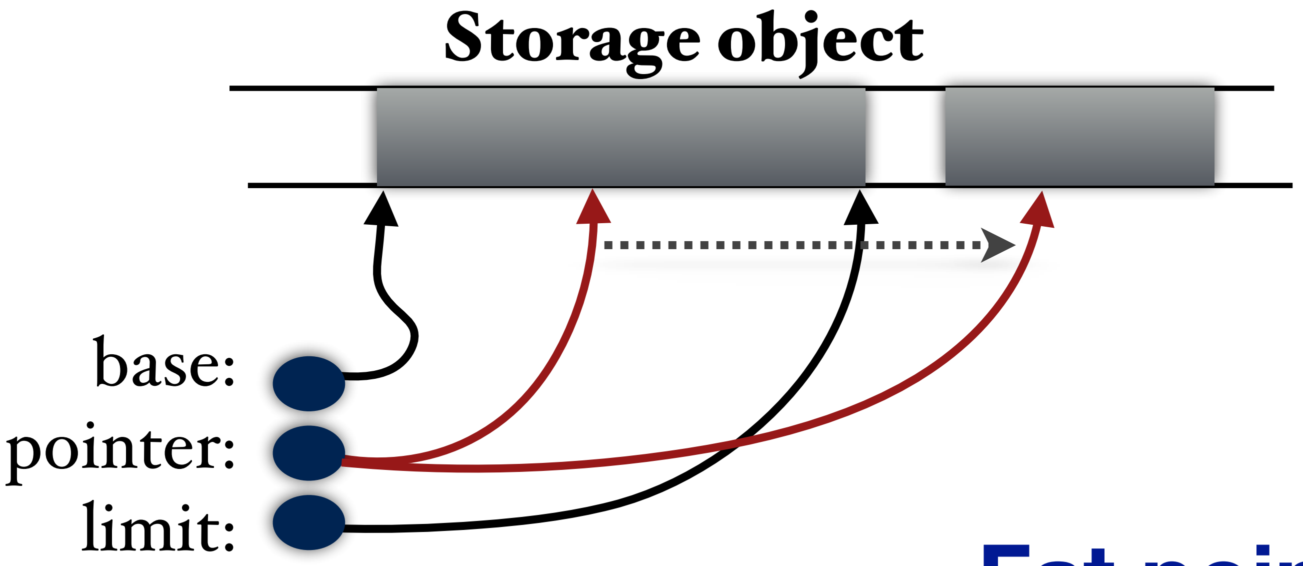
Halt right before out-of-bound pointers are dereferenced.

Runtime Overheads



High and unpredictable overhead

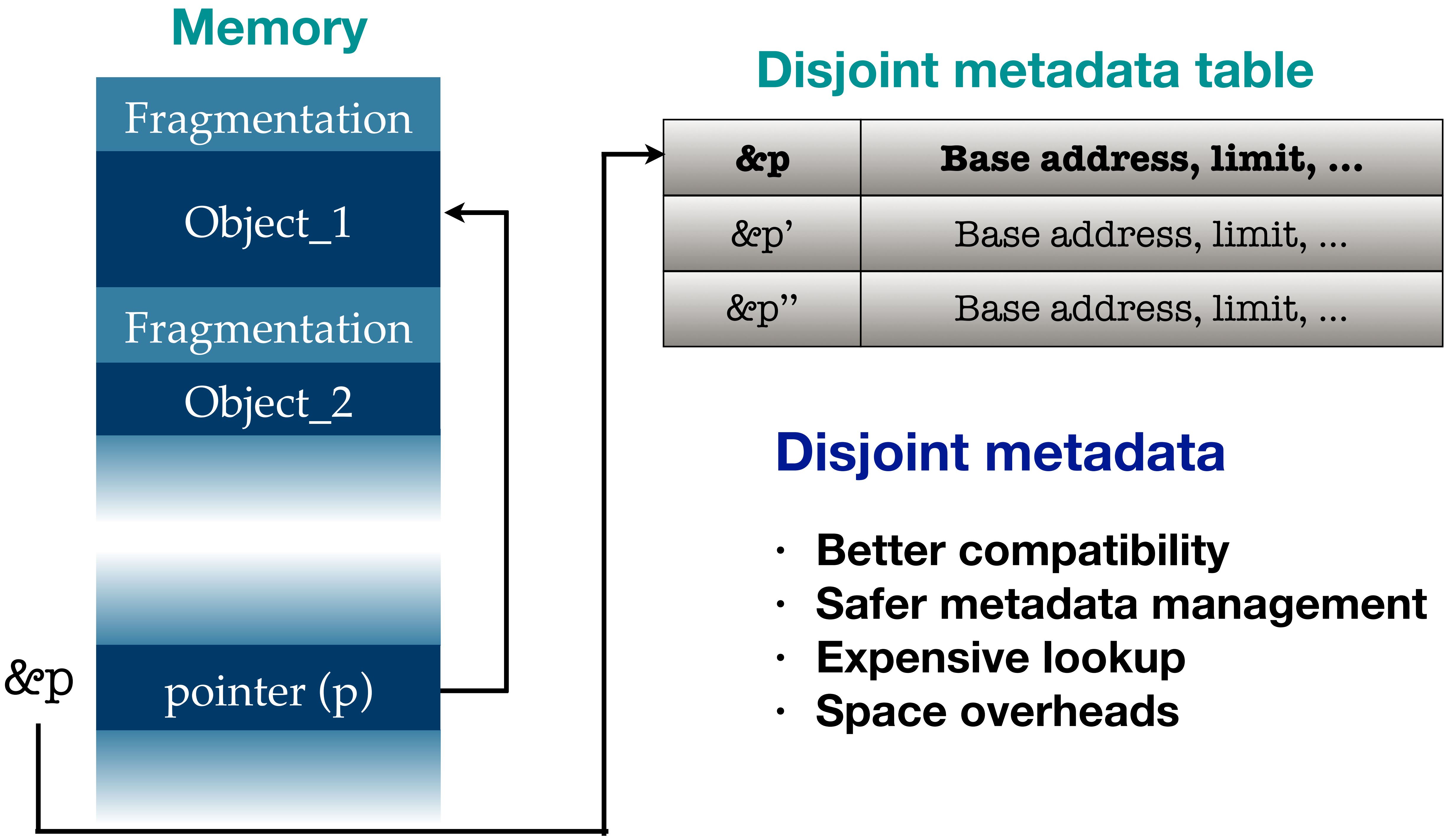
Metadata Storage 1/2



Fat pointer

- FAST (high locality of references)
- Low compatibility with precompiled libs
- Metadata overwritten by unsafe typecast

Metadata Storage 2/2



Memory

Fragmentation

Object_1

Fragmentation

Object_2

pointer (p)

&p

Disjoint metadata table

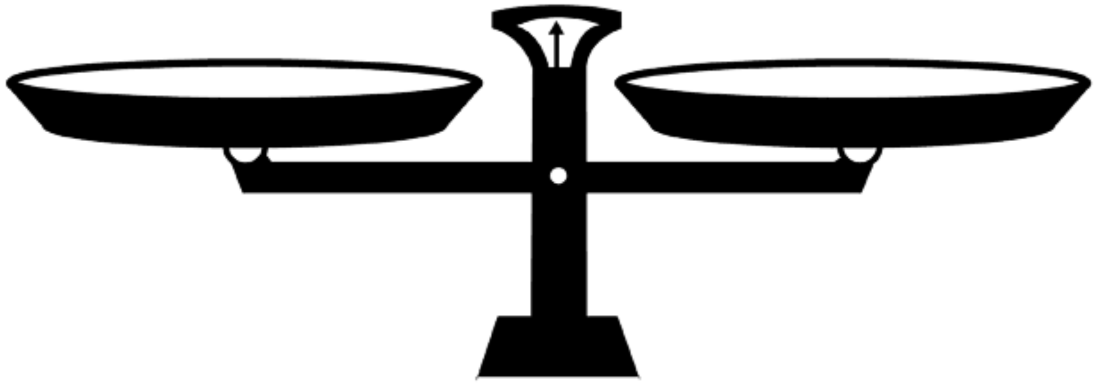
&p	**Base address, limit, …**
&p’	Base address, limit, …
&p”	Base address, limit, …

Disjoint metadata

- **Better compatibility**
- **Safer metadata management**
- **Expensive lookup**
- **Space overheads**

Trade-offs

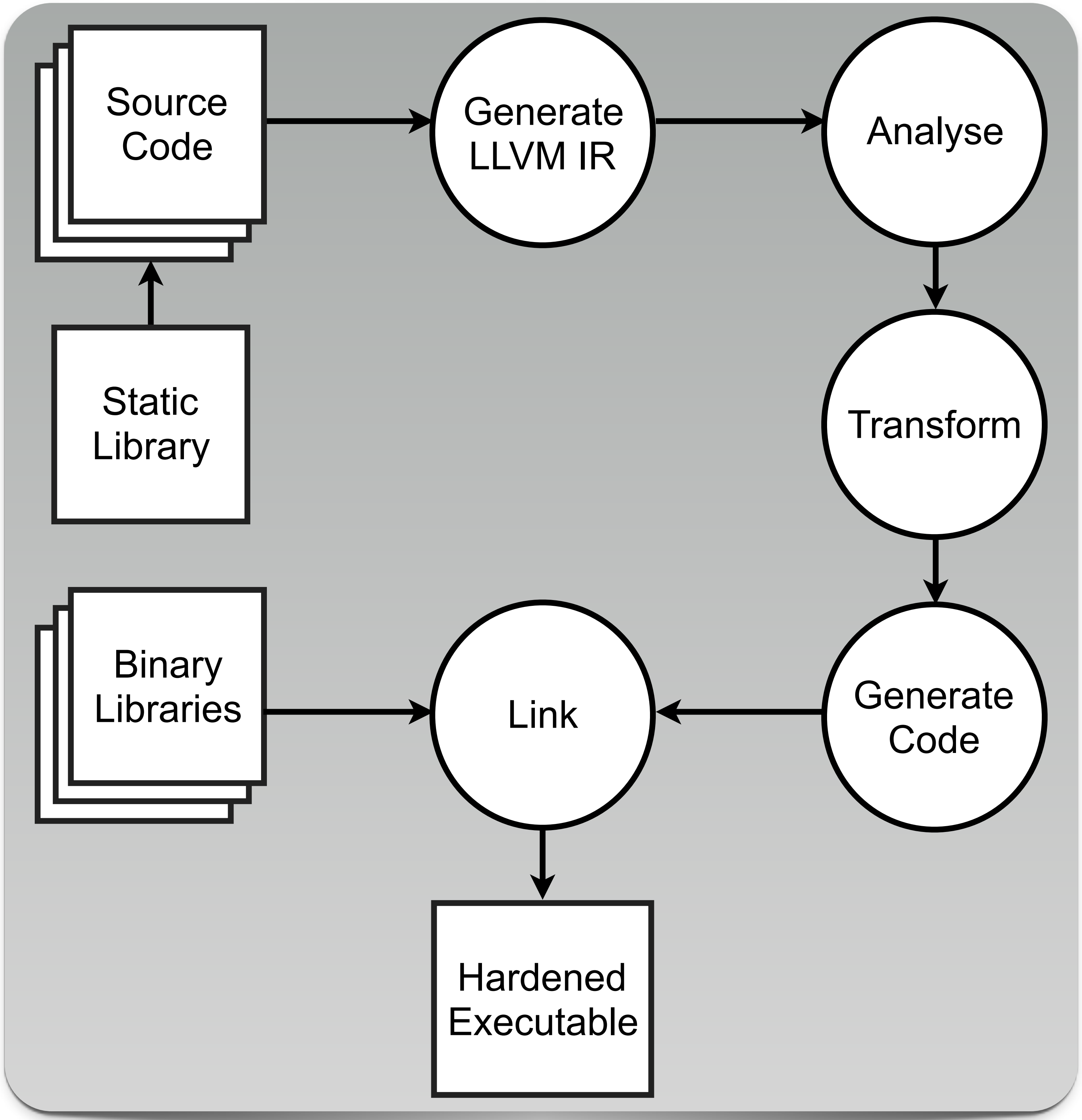
Runtime overheads	Space overheads (shadow space, padding)
Complete checking	Incomplete checking (e.g. internal overflows)
Precise checking	Approximate checking
Better compatibility	Less compatibility



FRAMER

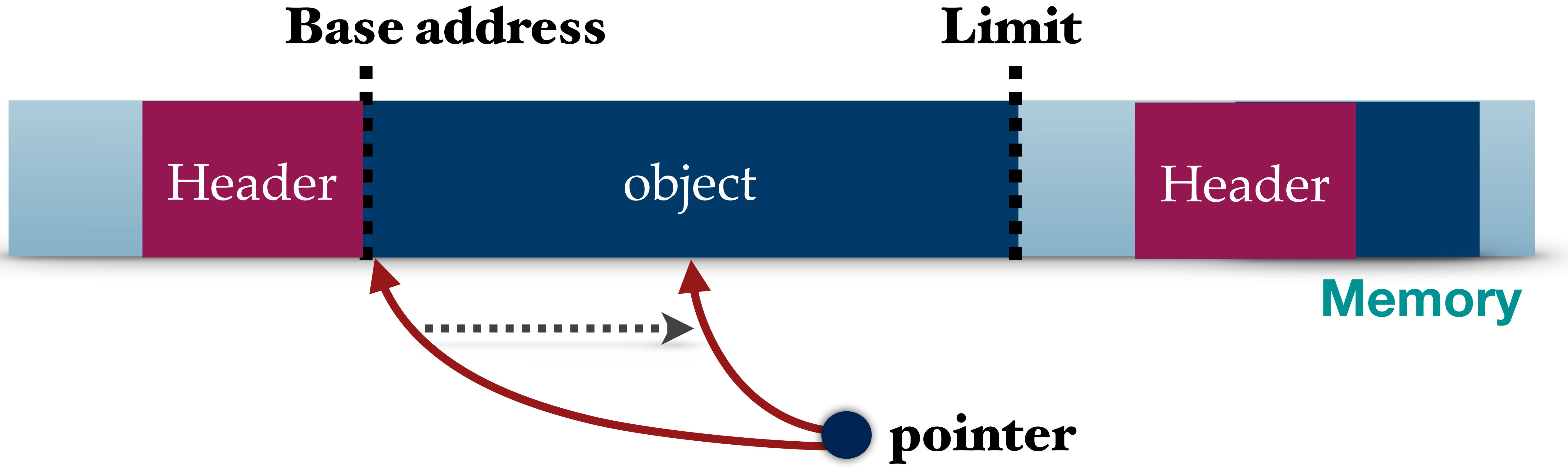
- High locality of references
  - Having an object carry its own metadata
  - Using a supplementary table
- Streamlined metadata lookup in the data structure
  - The worst case: O(1)
- Compatibility
  - Avoiding internal memory layout change or superfluous padding
- Scalability
  - Extending its usage to type safety, thread safety or garbage collection using per-object information

Overall Architecture



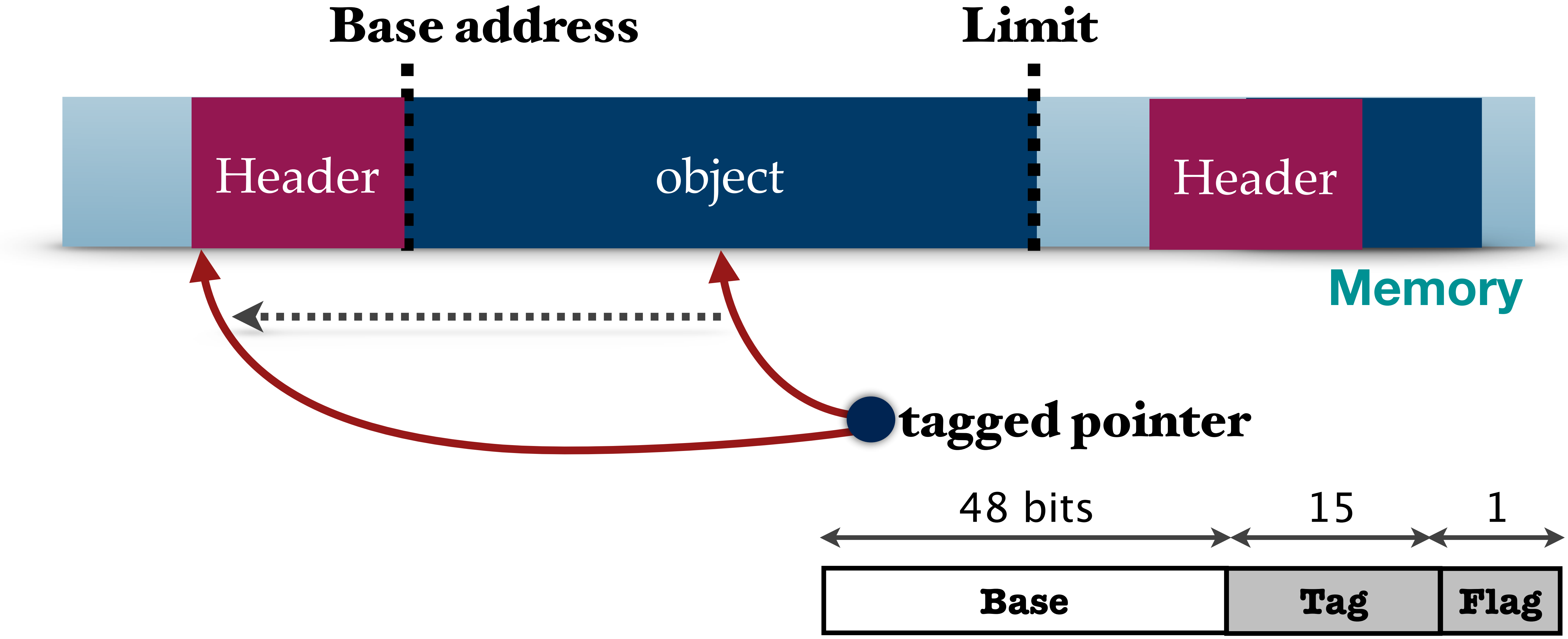
Framer is Implemented as a LLVM LTO Pass for whole program analysis

Metadata Storage



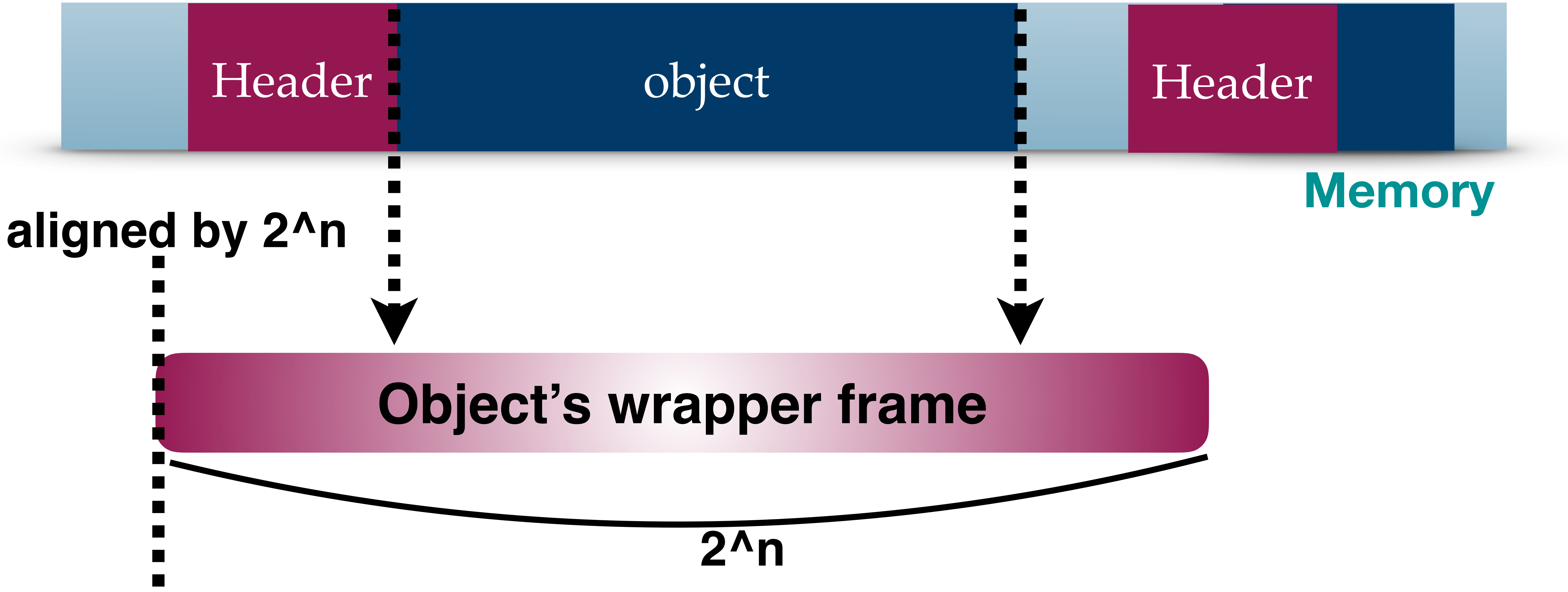
For the higher locality of references, we attach a header.

Metadata Retrieval



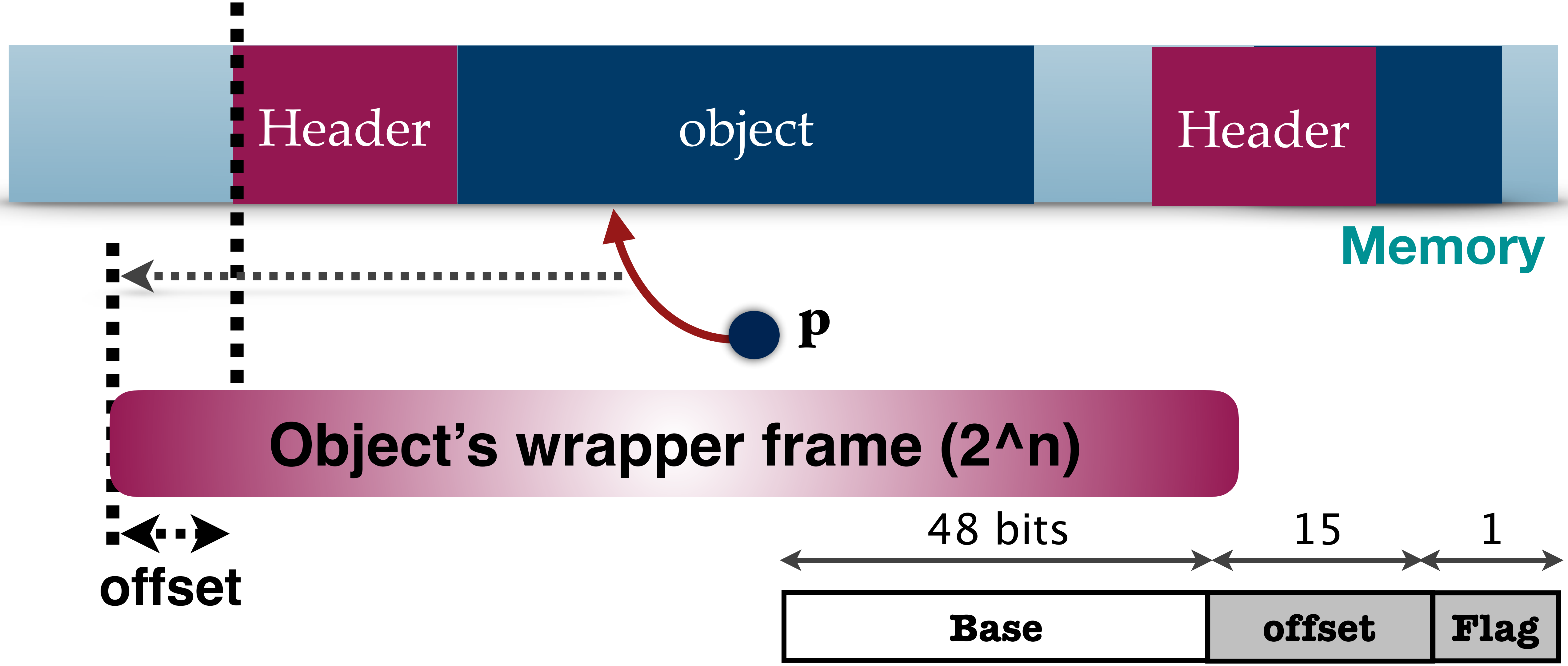
The address of a header is derived from a tagged pointer.

(Wrapper) Frame



An object's wrapper frame is defined as the smallest frame.

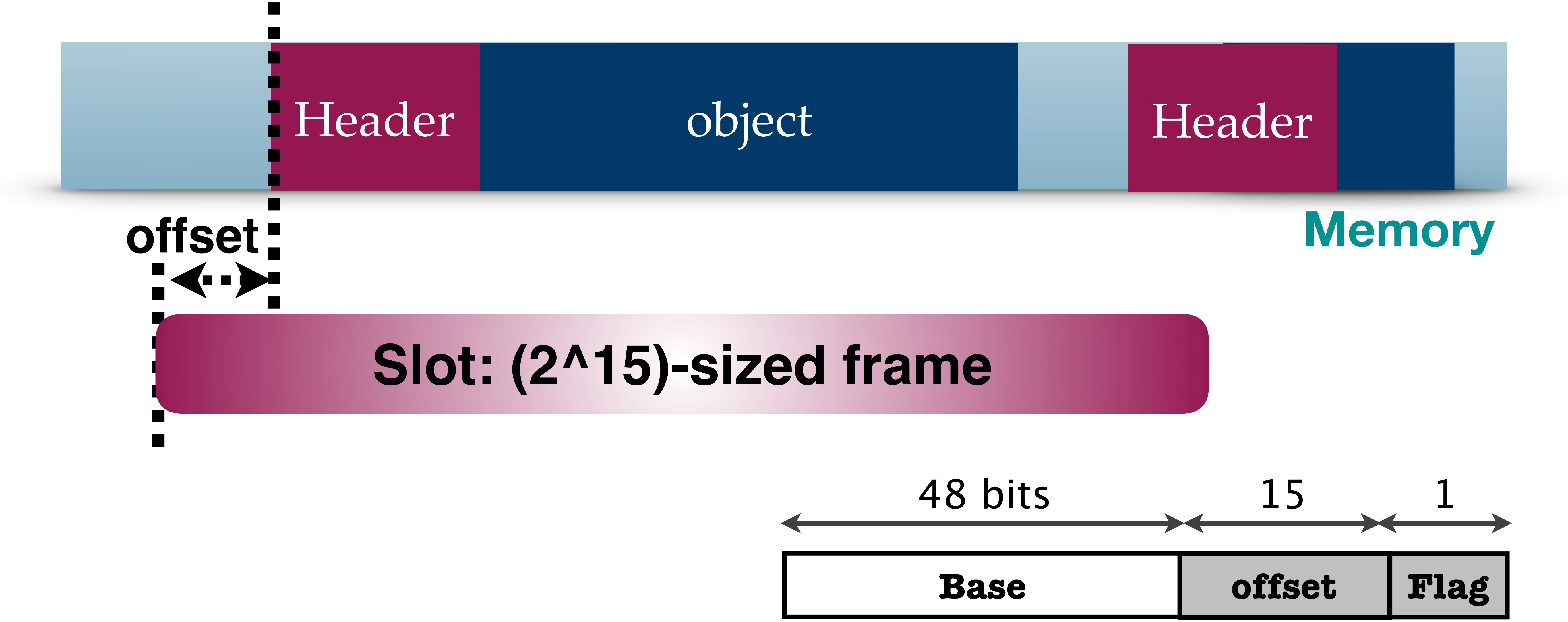
Derivation of Header Location



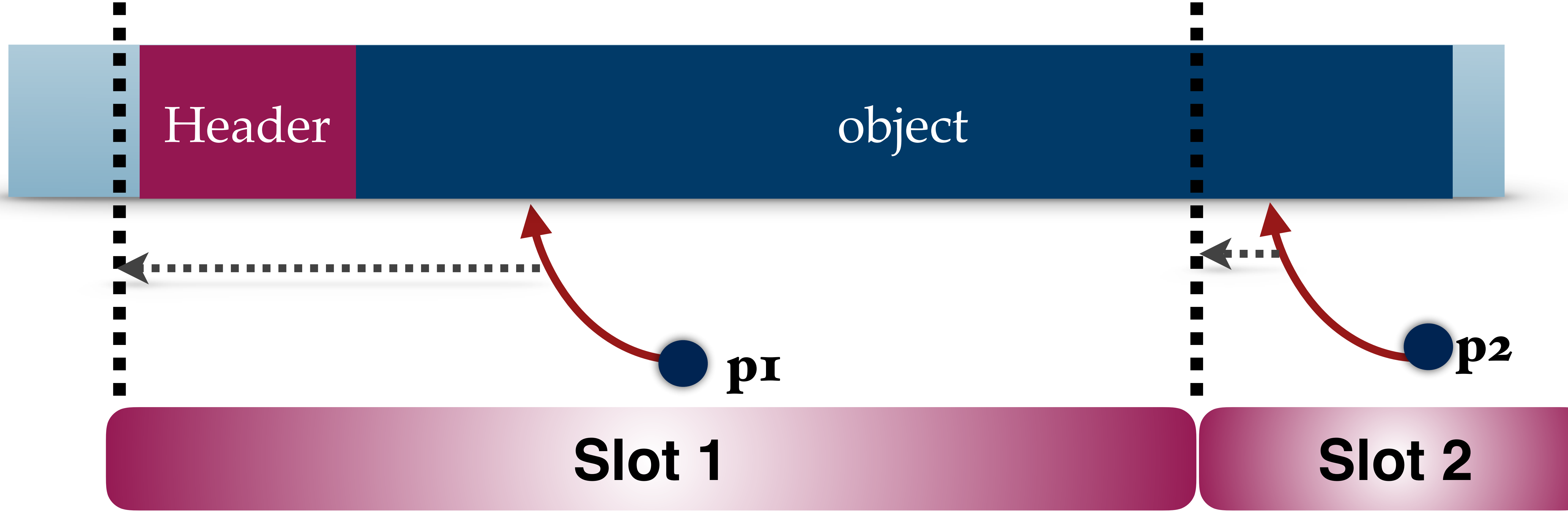
48 bits	15	1
Base	offset	Flag

The base of the wrapper frame= p & ((˜0)<<n)

Slot

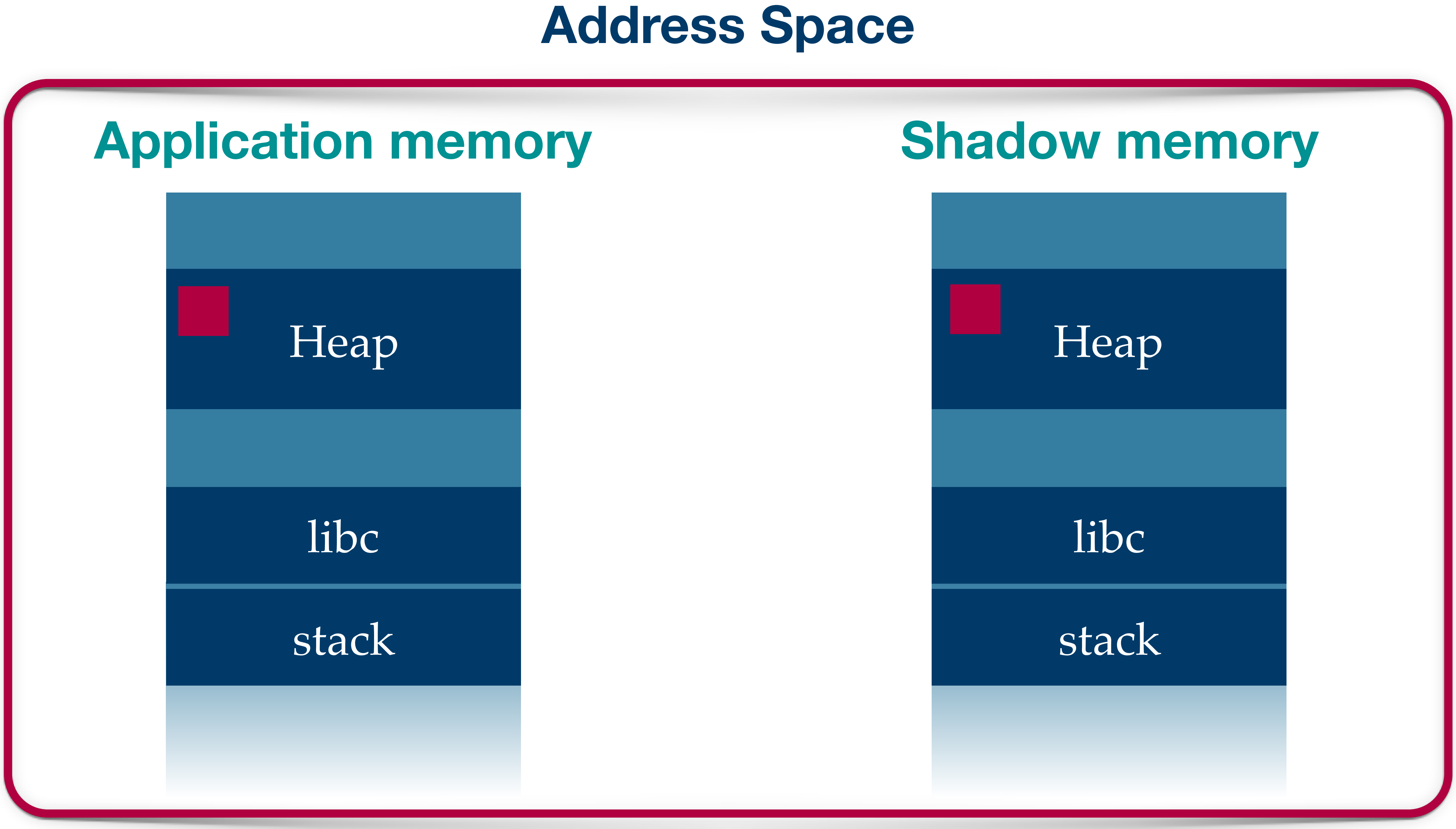

Header
object
Header
Memory
offset
Slot: (2^15)-sized frame
48 bits
15
1
Base
offset
Flag

Derivation Fails



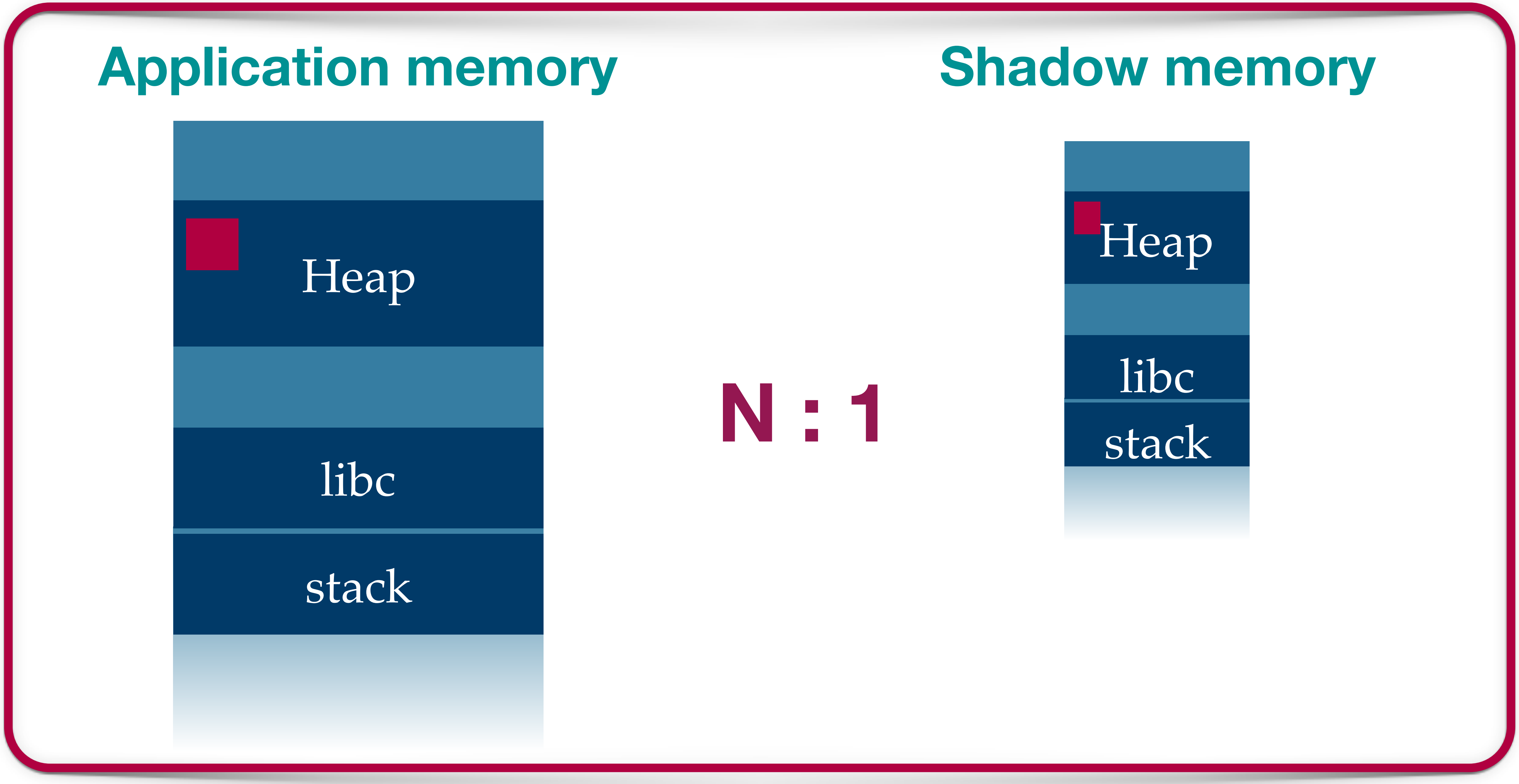
Offset cannot be used as relative location information.

Shadow Space

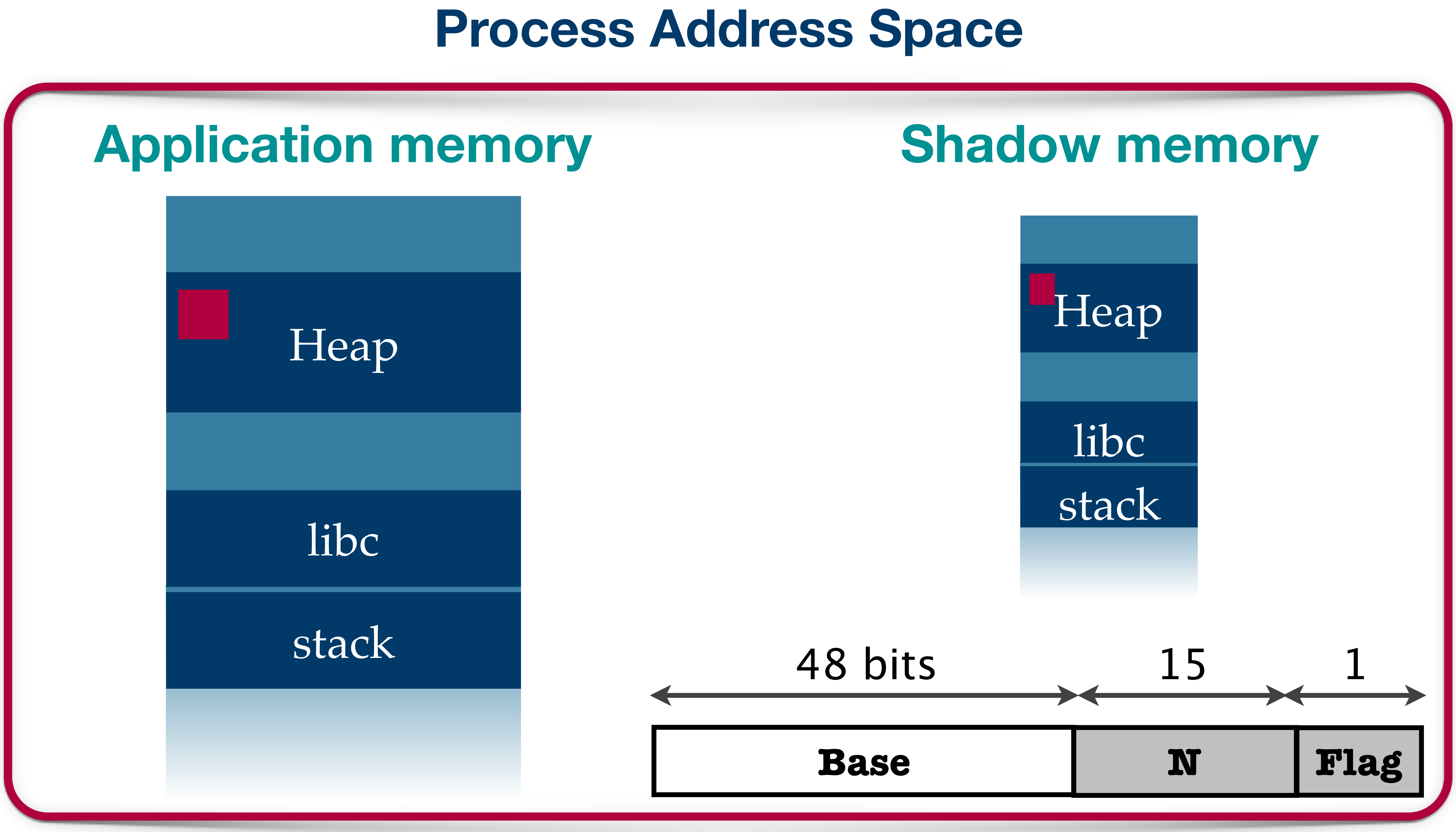


Compact Shadow Space

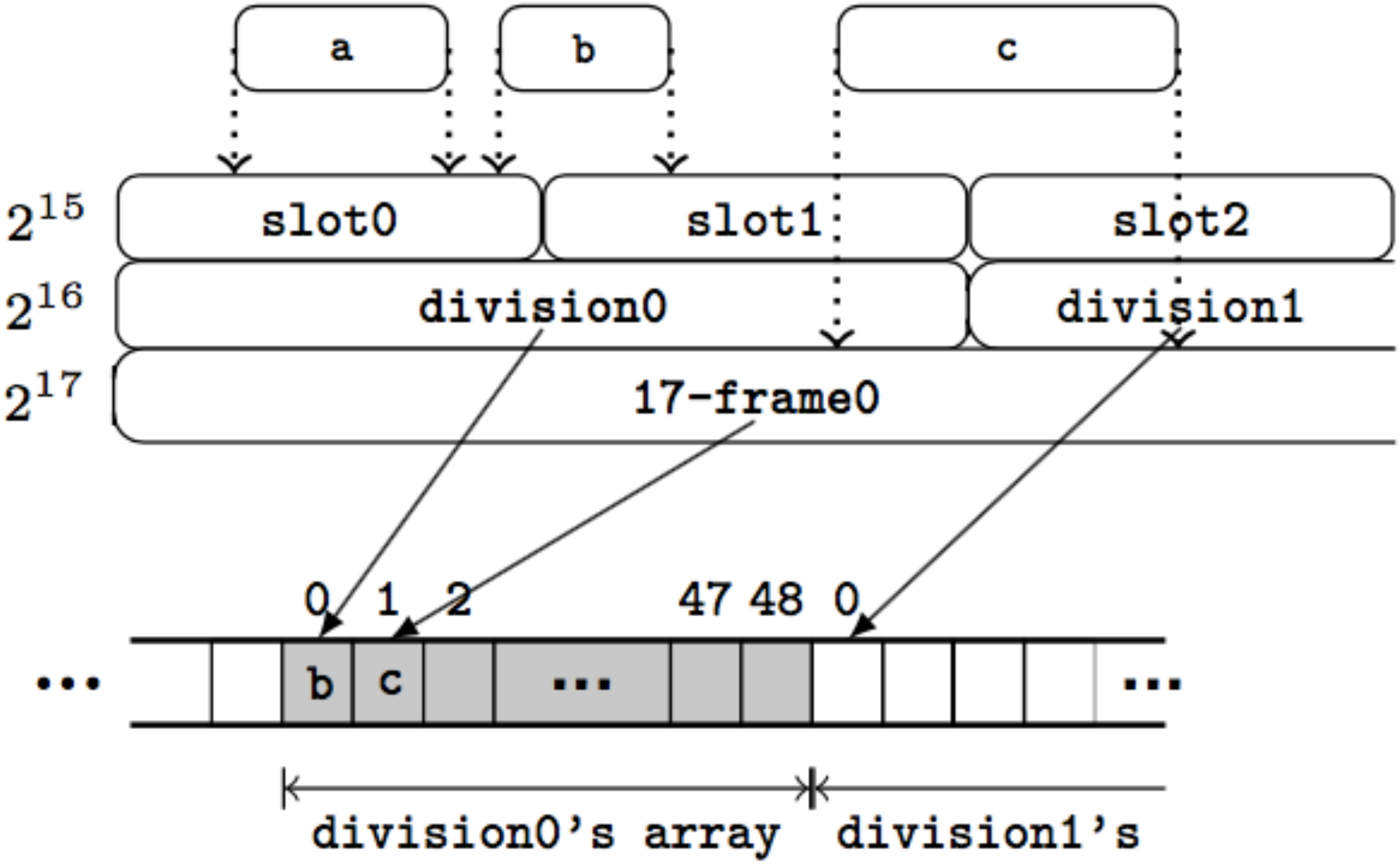
Process Address Space



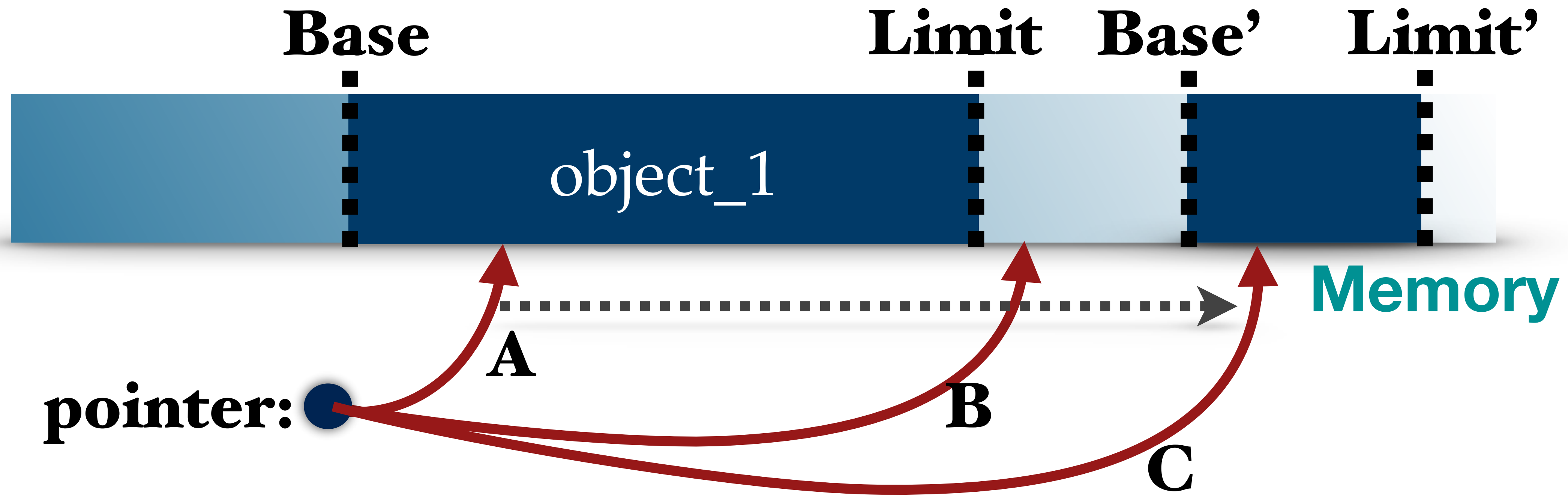
Framer’ s Shadow Space

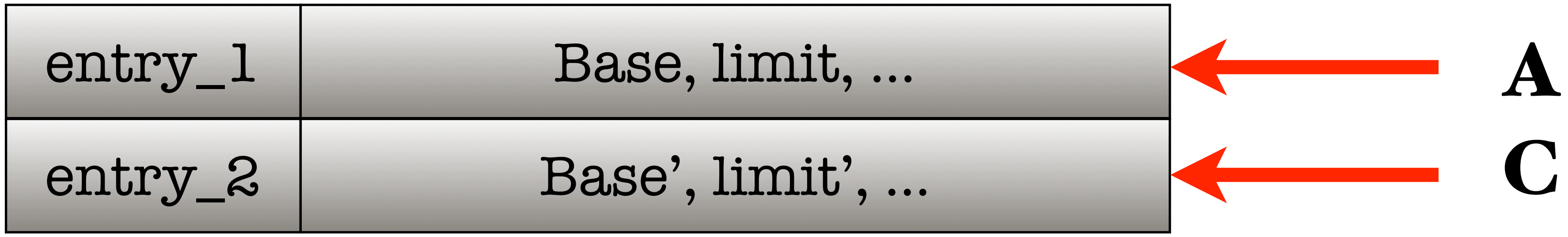


Mapping Table Entries



False Negatives





entry_1	Base, limit, …
entry_2	Base', limit', …

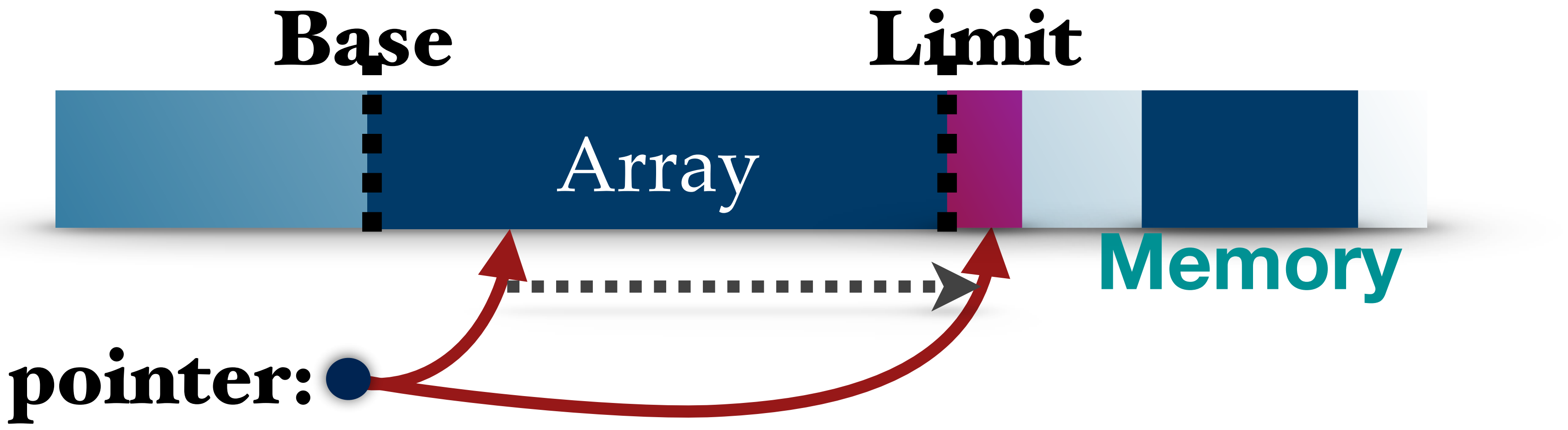
Tracking objects requires checks at pointer arithmetic to keep track of intended referents.

Now, False Positives

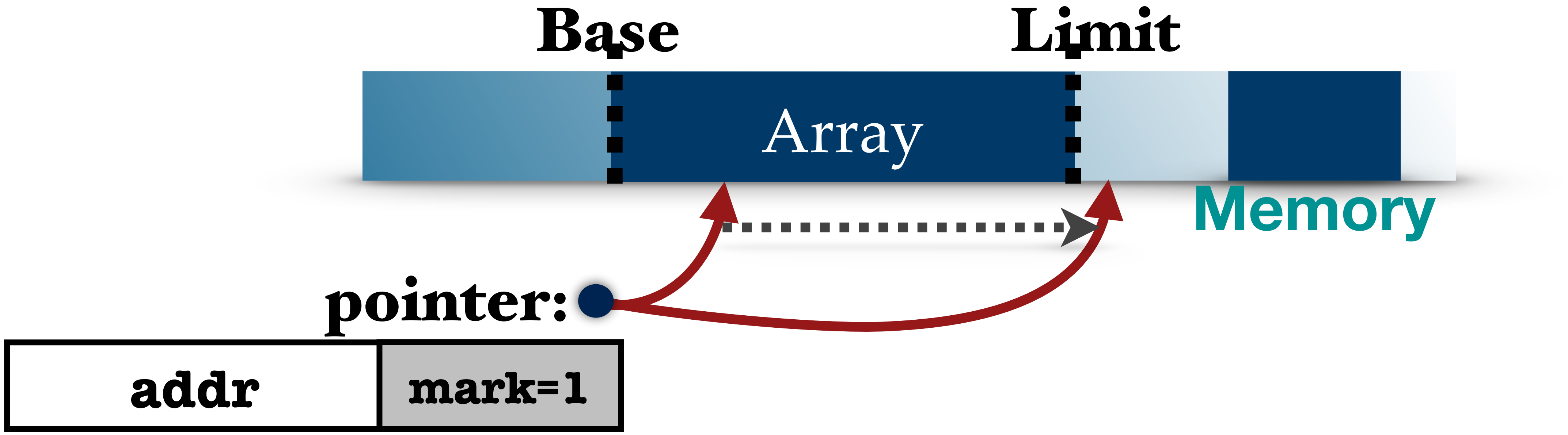
```
  int *p;

  int *a= (int*)malloc(100*sizeof(int));

  for (p=a; p<&a[100];++p)

     *p=0;

/*  p == &a[100]  */
```

Should we check bounds at pointer arithmetic
AND
memory read/write??

Previous Solutions

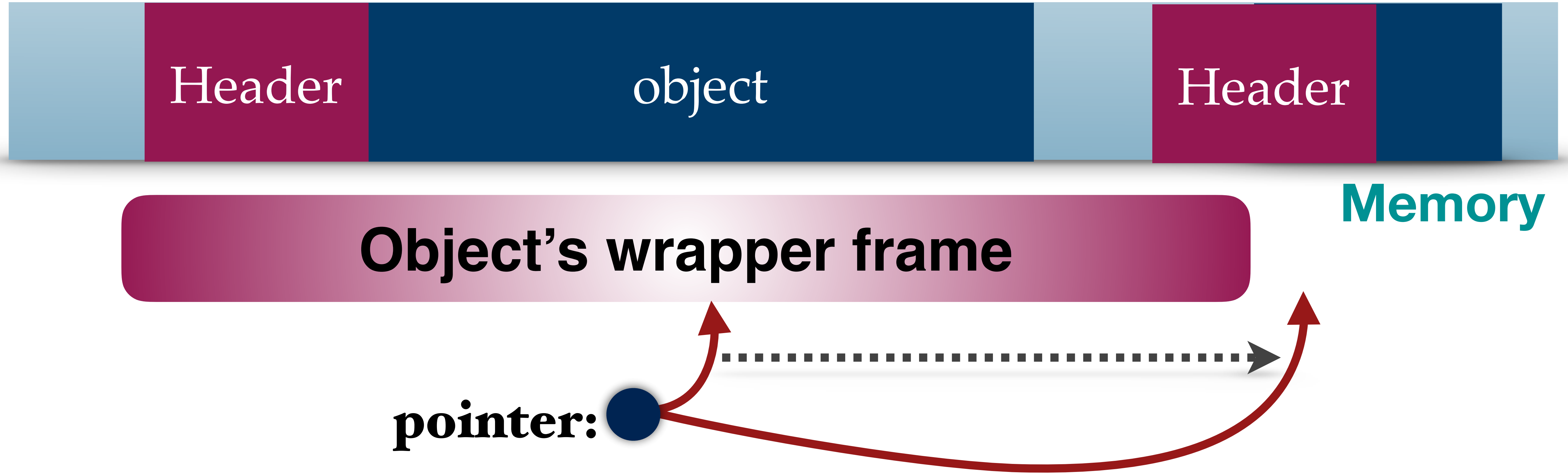


1. Pad an off-by-one byte.



2. Mark out-of-bound pointer at pointer arithmetic.

In-frame Checking



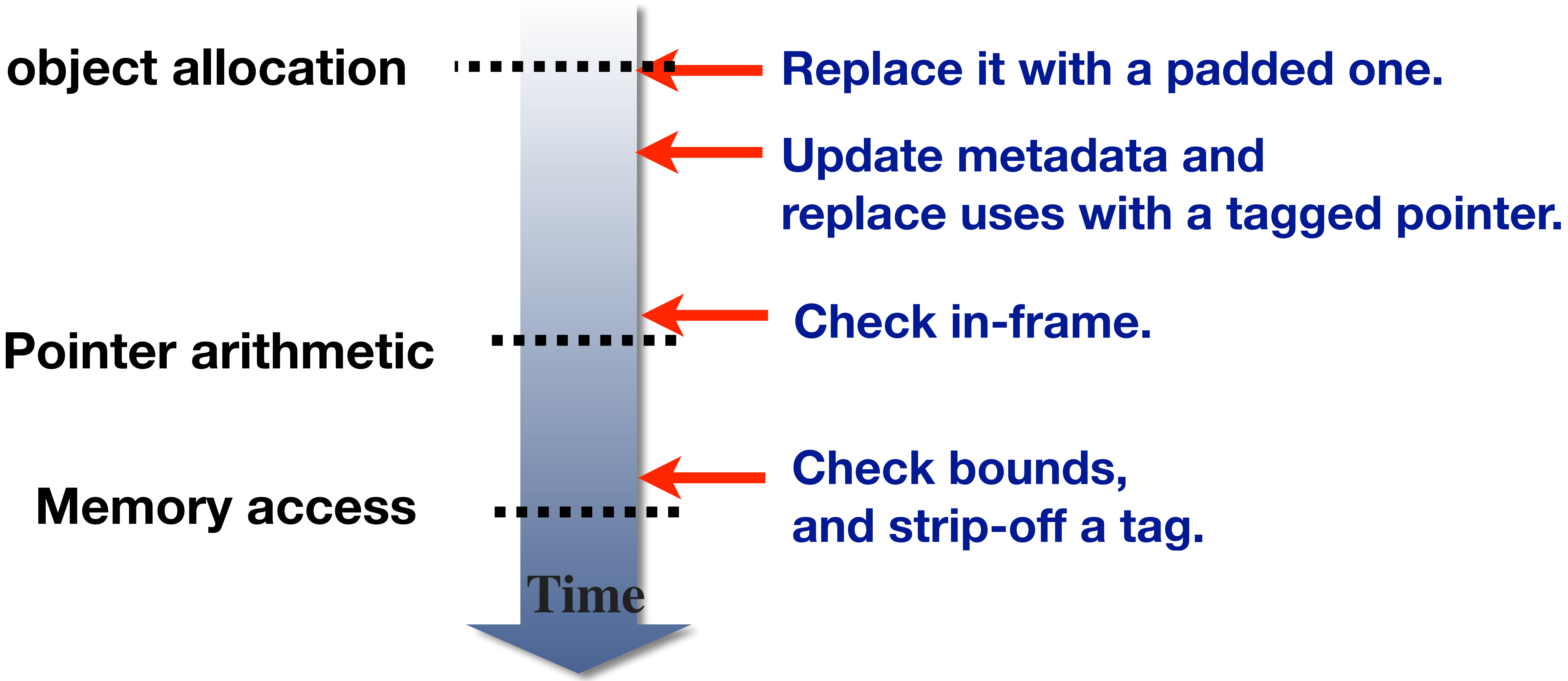
Check only in-frame at pointer arithmetic.

Interoperability

- Framer ensures compatibility with un-instrumented libs
  - Strip-off tagged pointers passed to pre-compiled libs
  - Header attached does not damage compatibility

Program Transformation

```
int myarray [10];    /* object allocation*/
int * p= myarray;   /* pointer creation && assignment */
p= p+4;             /* pointer arithmetic */
*p=10;              /* pointer dereference */
```



object allocation — Replace it with a padded one.

Update metadata and replace uses with a tagged pointer.

Pointer arithmetic — Check in-frame.

Memory access — Check bounds, and strip-off a tag.

Optimization

- Reduce objects to be tracked.
  - Use the compiler’s variable range analysis
  - Minimise the penalty of using tagged pointers
- Reduce run-time checks
  - Hoist runtime checks outside loops
  - Remove redundant checks due to a previous check
  - Remove checks for pointers statically determined safe

Advantage

- High locality of references
  - Storing per-object metadata in the header
  - Supplementary table in the form of a contiguous array.
  - Low, stable cache misses compared to other approaches
- Streamlined metadata lookup
  - Direct access to the corresponding header or entry < hash table management
- Low space overhead
- Compact encoding of addresses
  - 4 bytes of size information < 1 word (the base) + alpha

Discussion

- Losing high locality for big-sized arrays
- Vulnerable to overwrites on metadata by user program's unsafe type casts like fat pointers
- More compact encoding for supplementary metadata table
- Reducing dynamic instruction counts using static analysis