

# Memento: Architectural Support for Ephemeral Memory Management in Serverless Environments

Ziqi Wang, **Kaiyang Zhao**, Pei Li, Andrew Jacob, Michael Kozuch\*,  
Todd C. Mowry, Dimitrios Skarlatos



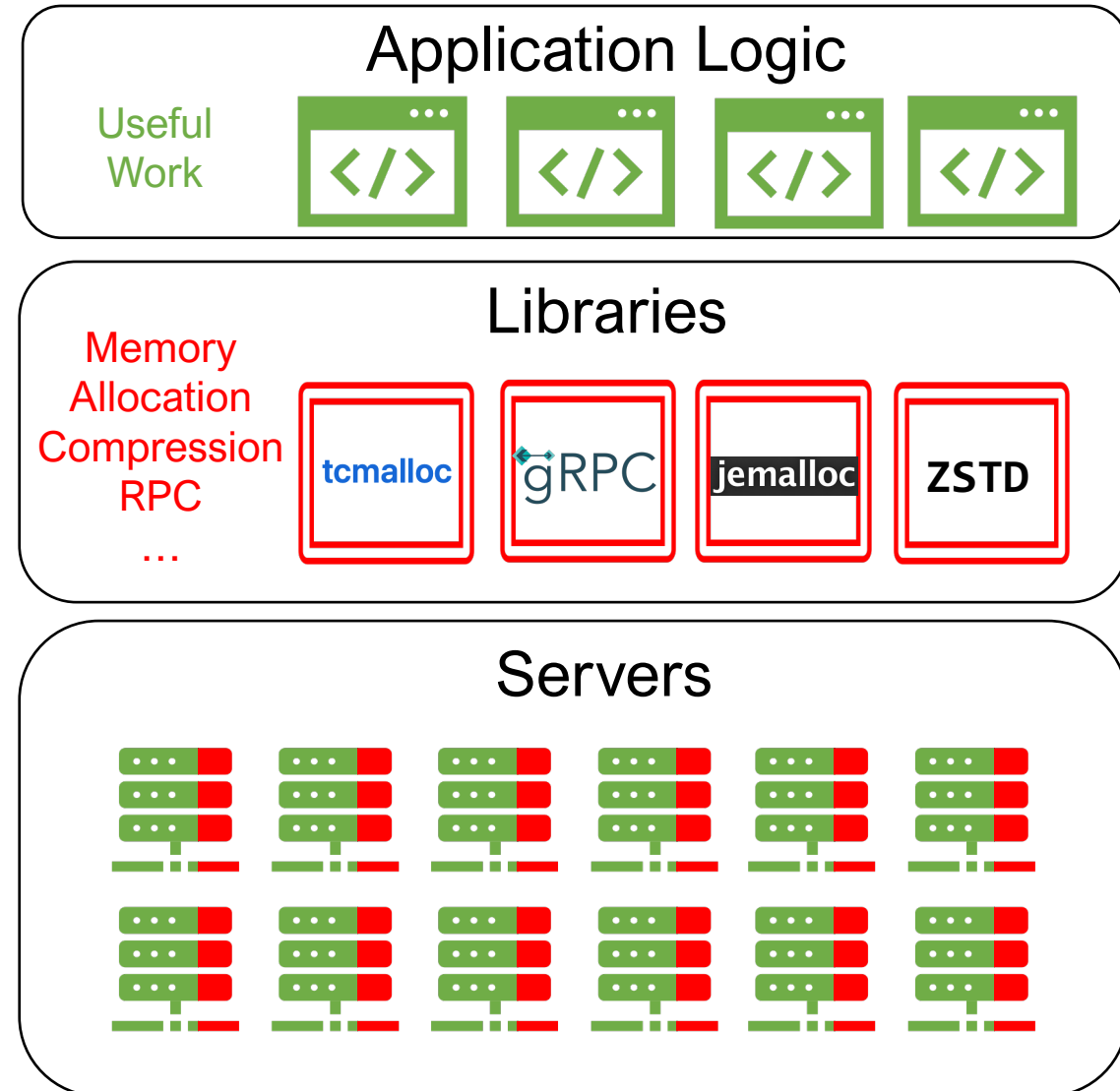
MICRO 2023

# Problem 1: Datacenter Tax Cycles!

Useful work by applications

## Datacenter Tax

Hotspots in the library layers  
of the software stack



# Problem 1: Datacenter Tax Cycles!

Useful work by applications

## Datacenter Tax

Hotspots in the library layers  
of the software stack

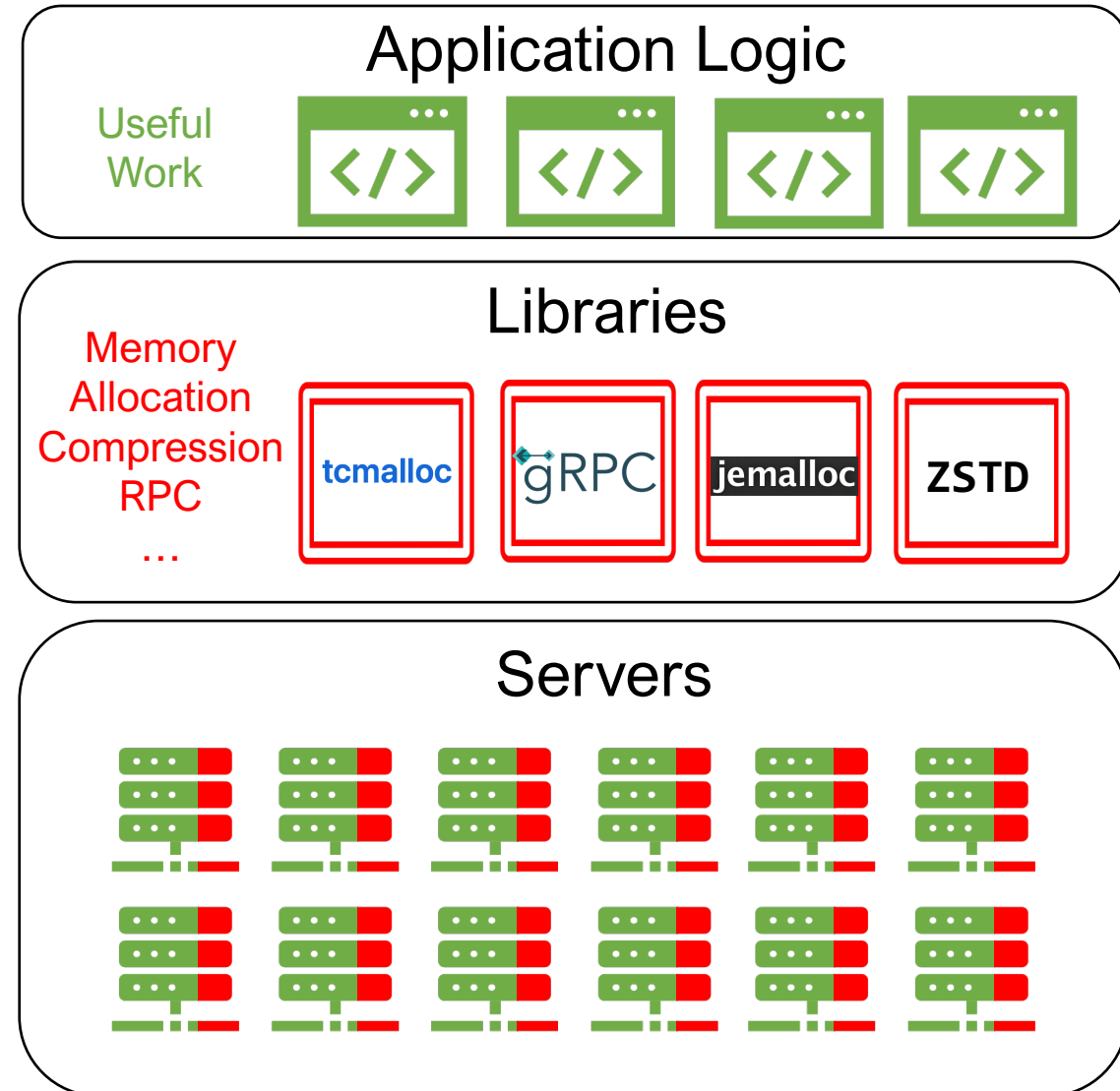


~30% of cycles are spent on tax!

S. Kanev et al [ISCA'15]

A. Gonzalez et al [ISCA'23]

Memory management is a major overhead



# Current State of Software in Datacenters

- Software architecture disaggregation



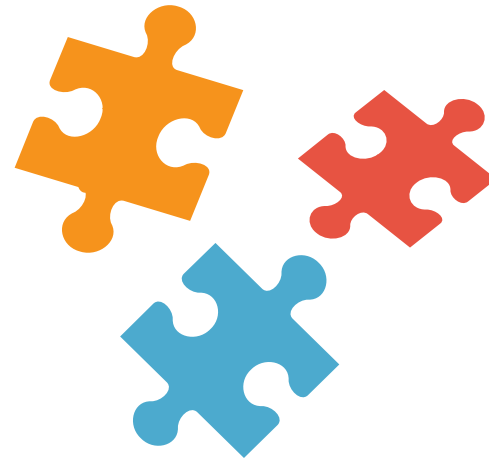
Monolith

# Current State of Software in Datacenters

- Software architecture disaggregation



Monolith



Microservices

# Current State of Software in Datacenters

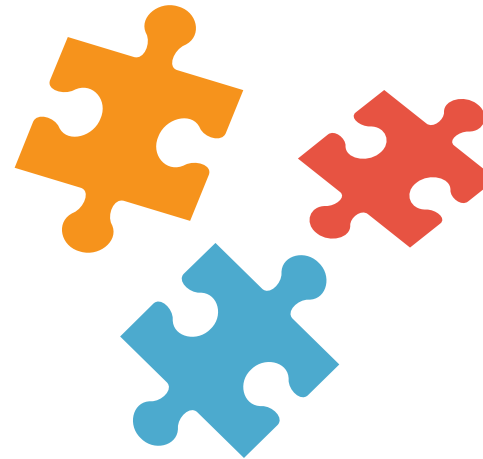
- Software architecture disaggregation



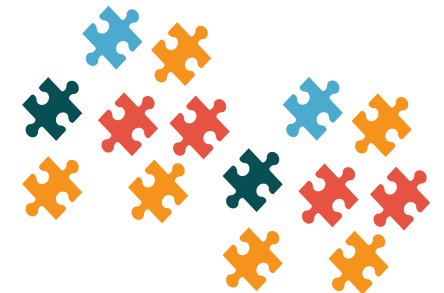
- ✓ Ease of Development
- ✓ Higher Resource Utilization
- ✓ High Scalability
- ✓ Ease of Deployment



Monolith



Microservices



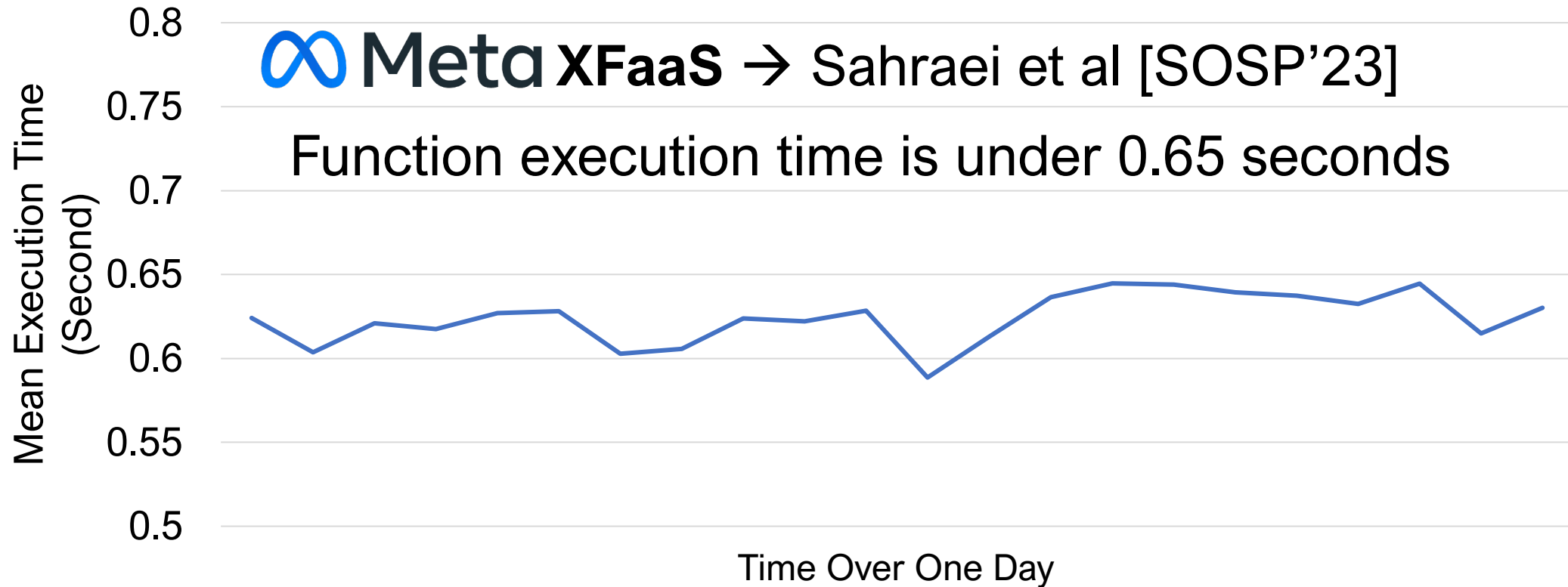
Serverless Functions

# Problem 2: Functions are short lived!



50% of functions runtime is under 1 second  
Shahrad et al [ATC'20]

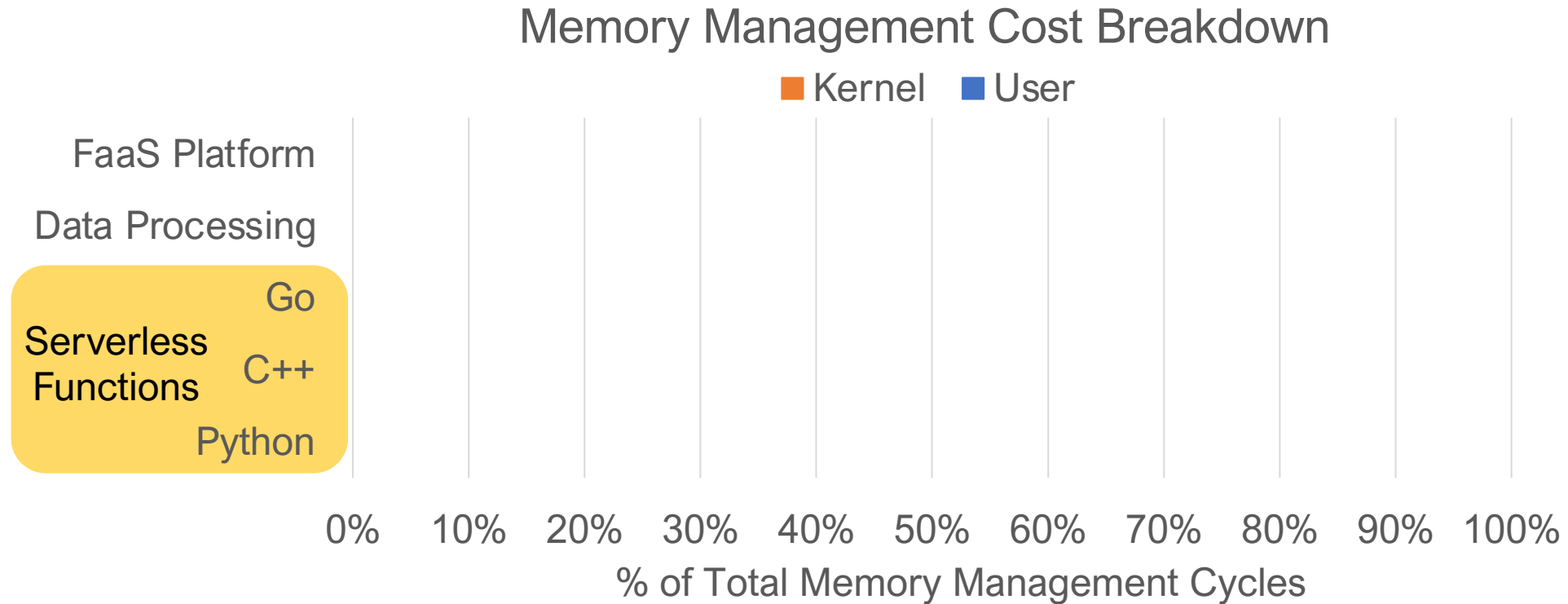
# Problem 2: Functions are short lived!



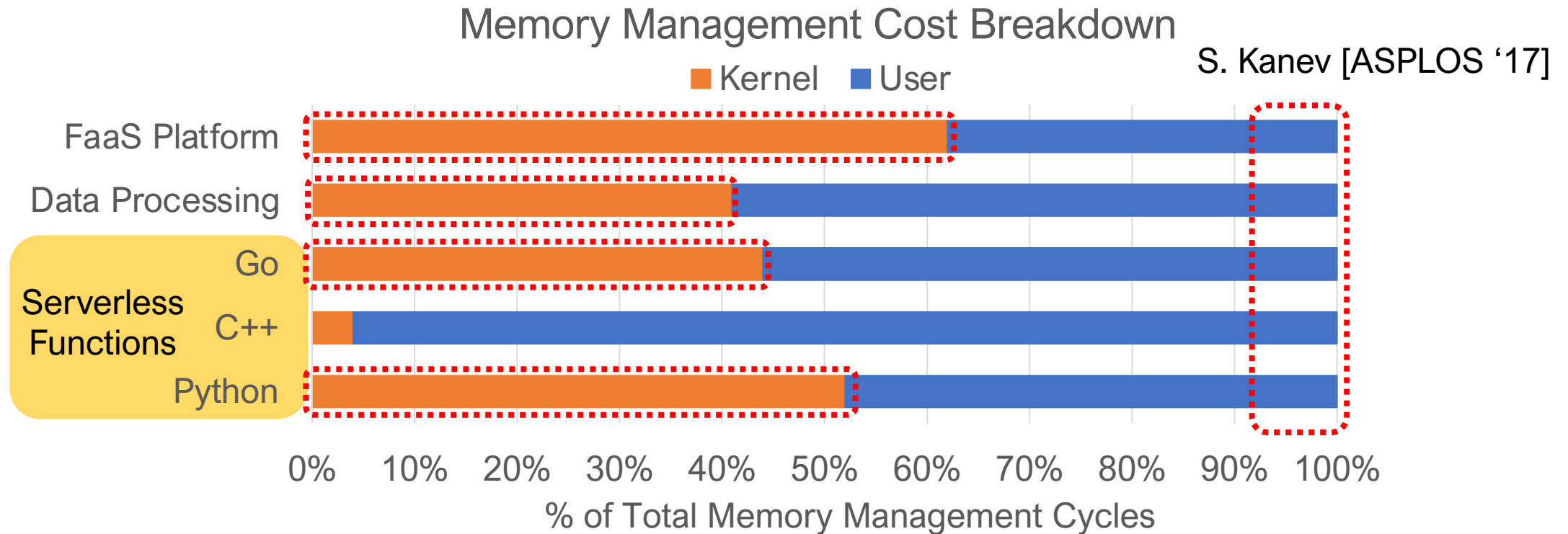
**Memory management tax up to 1/3 of function execution!**  
**Hard to amortize due to very short runtime**



# Problem 3: Kernel Memory Management Costs!



# Problem 3: Kernel Memory Management Costs!



1. Prior work focused on sub-malloc operations of userspace
2. Kernel memory management costs up to 62%!
3. Chain of allocs between userspace and kernel
4. Need to eliminate the complete allocation path

# Contribution: Memento

Goal: memory allocations spend their lifetime in caches

An arena-based hardware object allocator

- Enables full HW management of objects

A hardware page allocator

- Eliminates the OS costs of memory management

Exposes allocation semantics in HW

- Main memory bypass for new objects

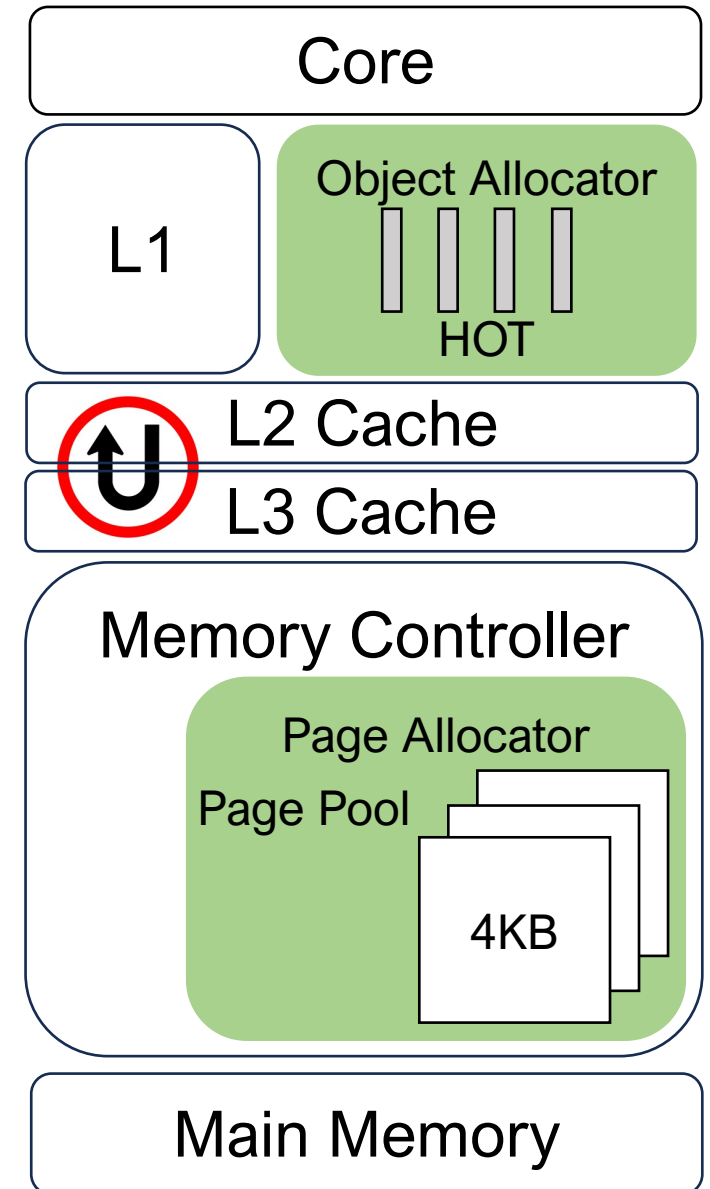
8%-28% speedup of function execution

4%-11% speedup of long-running workloads

23% reduction of memory usage

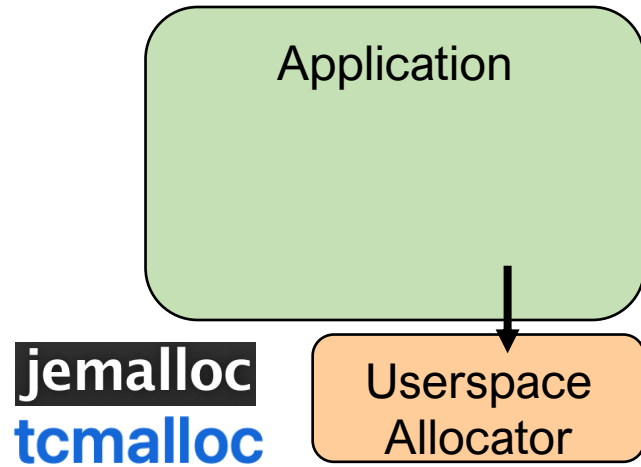
30% reduction of memory bandwidth needs

29% reduction in FaaS costs



# Identifying Challenges & Opportunities in Serverless Memory Management

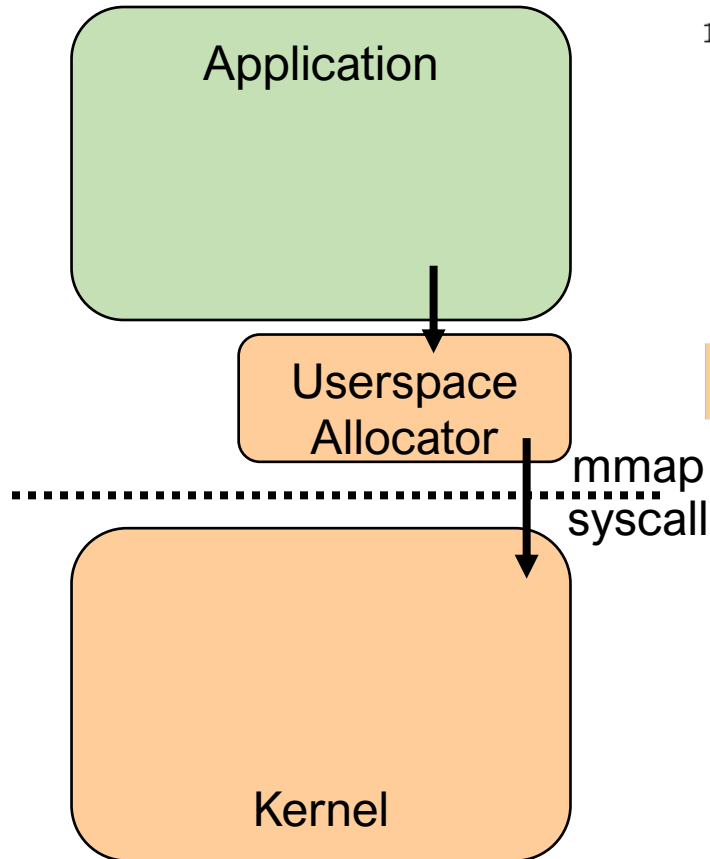
# Userspace Memory Management is Slow



```
1 int *arr = malloc(32);
```

**10-100s of Cycles**

# Kernel Memory Management is Even Slower



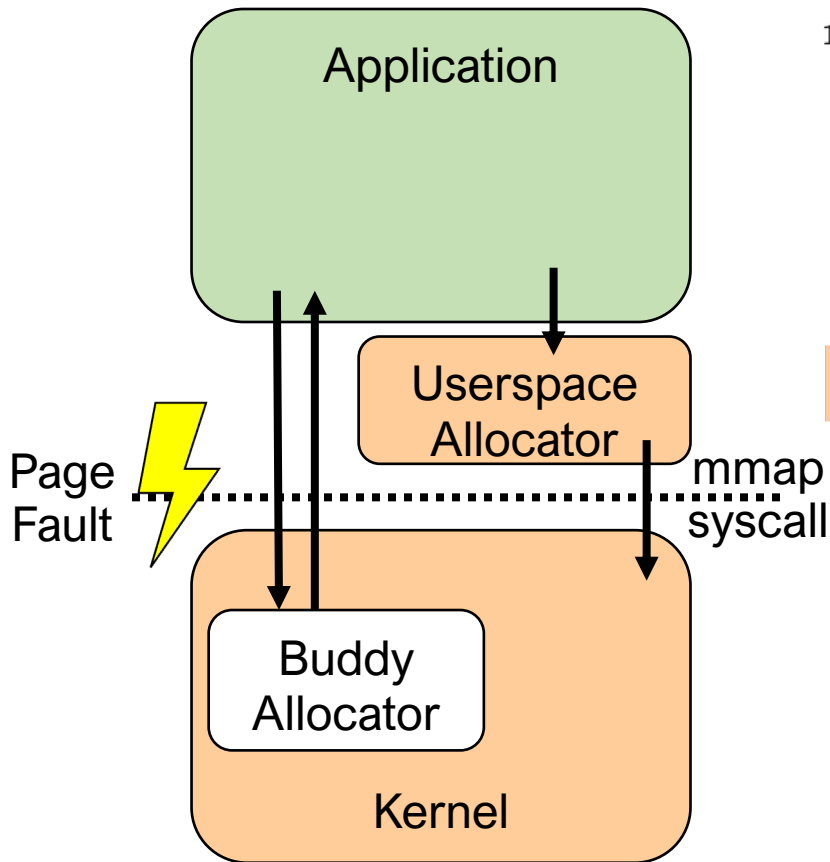
```
1 int *arr = malloc(32);
```

10-100s of Cycles

```
1 arena = mmap(...);
```

100-1000s of Cycles

# Kernel Memory Management is Even Slower



```
1 int *arr = malloc(32);
```

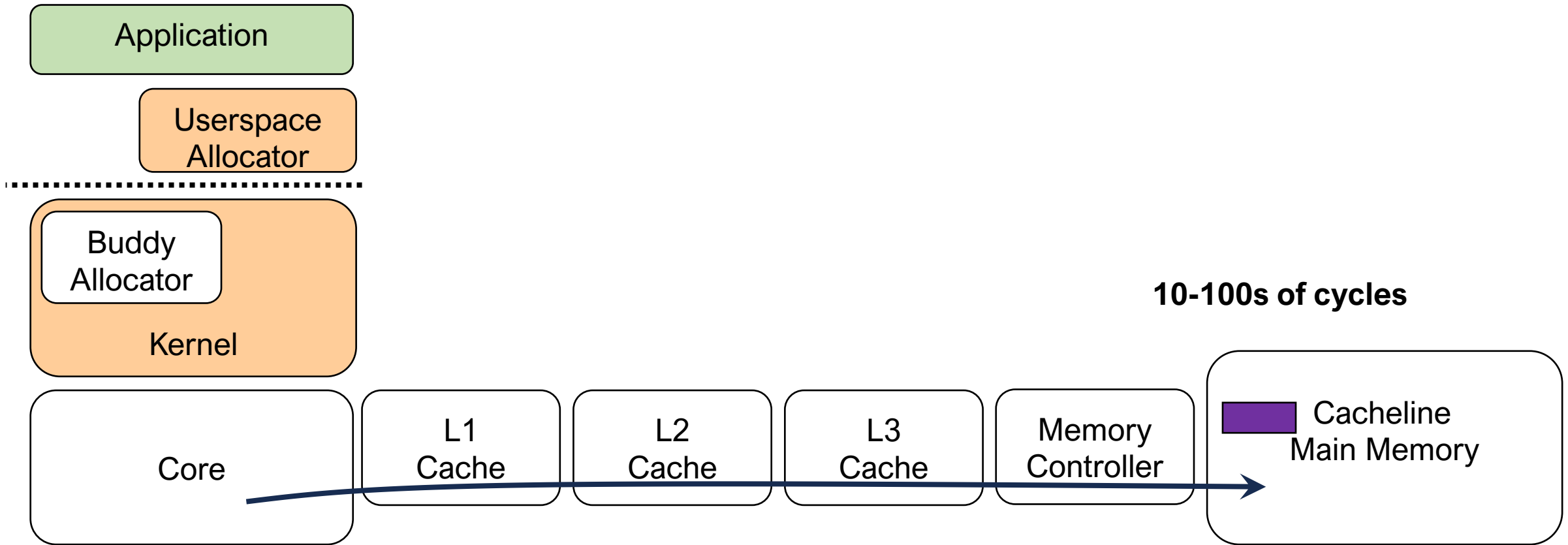
**10-100s of Cycles**

```
1 arena = mmap(...);
```

**100-1000s of Cycles**

**1000-10,000s of Cycles**

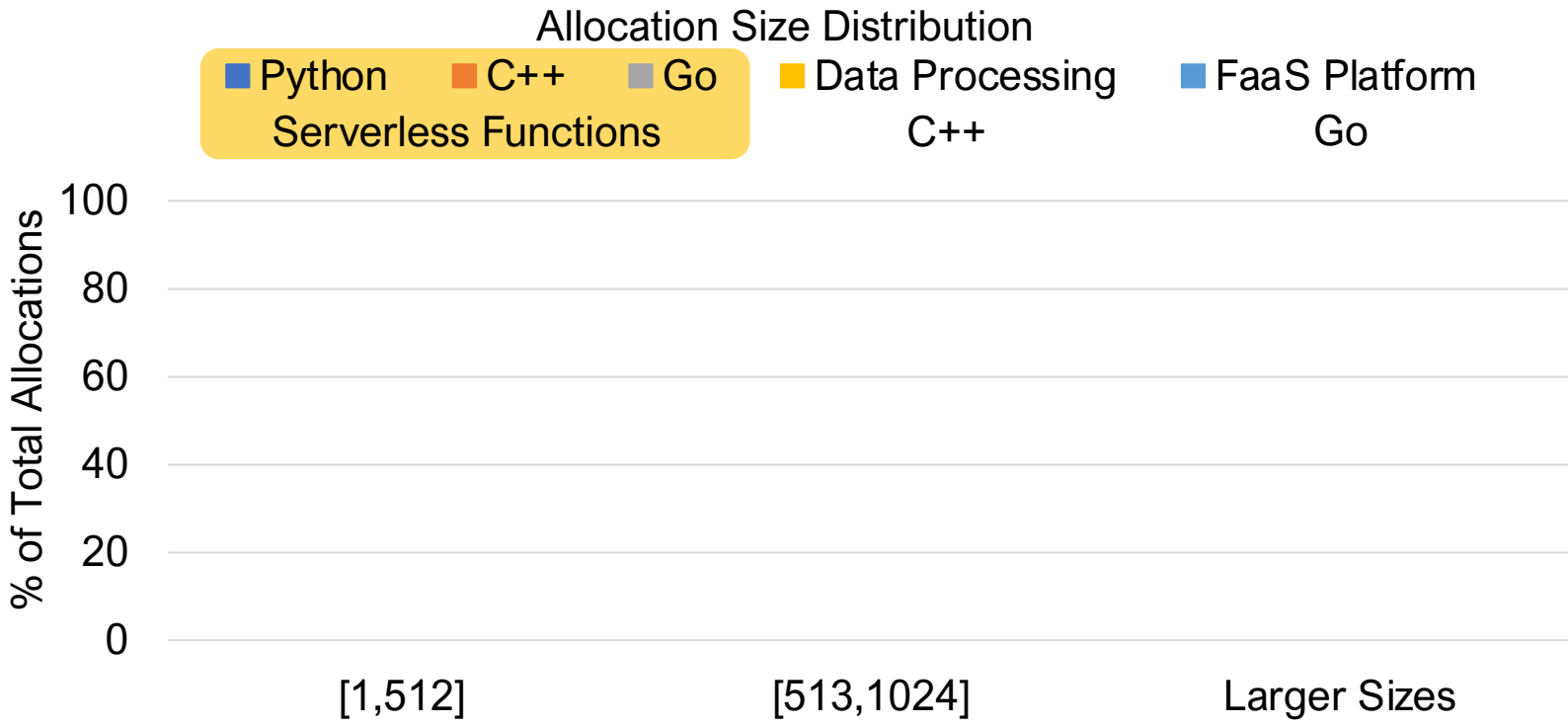
# Unnecessary Hardware Operations



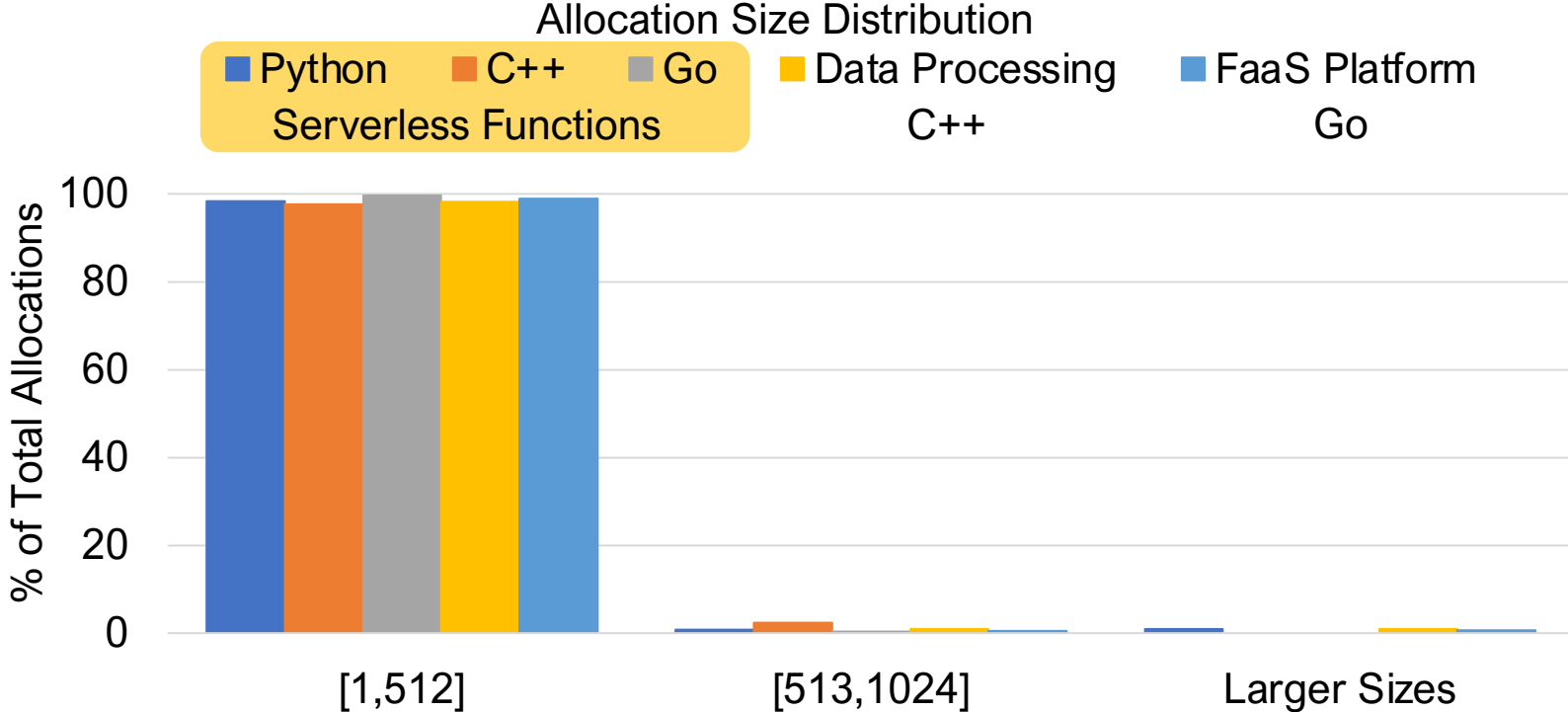
Fetch cacheline all the way from main memory



# Allocation Sizes are Small

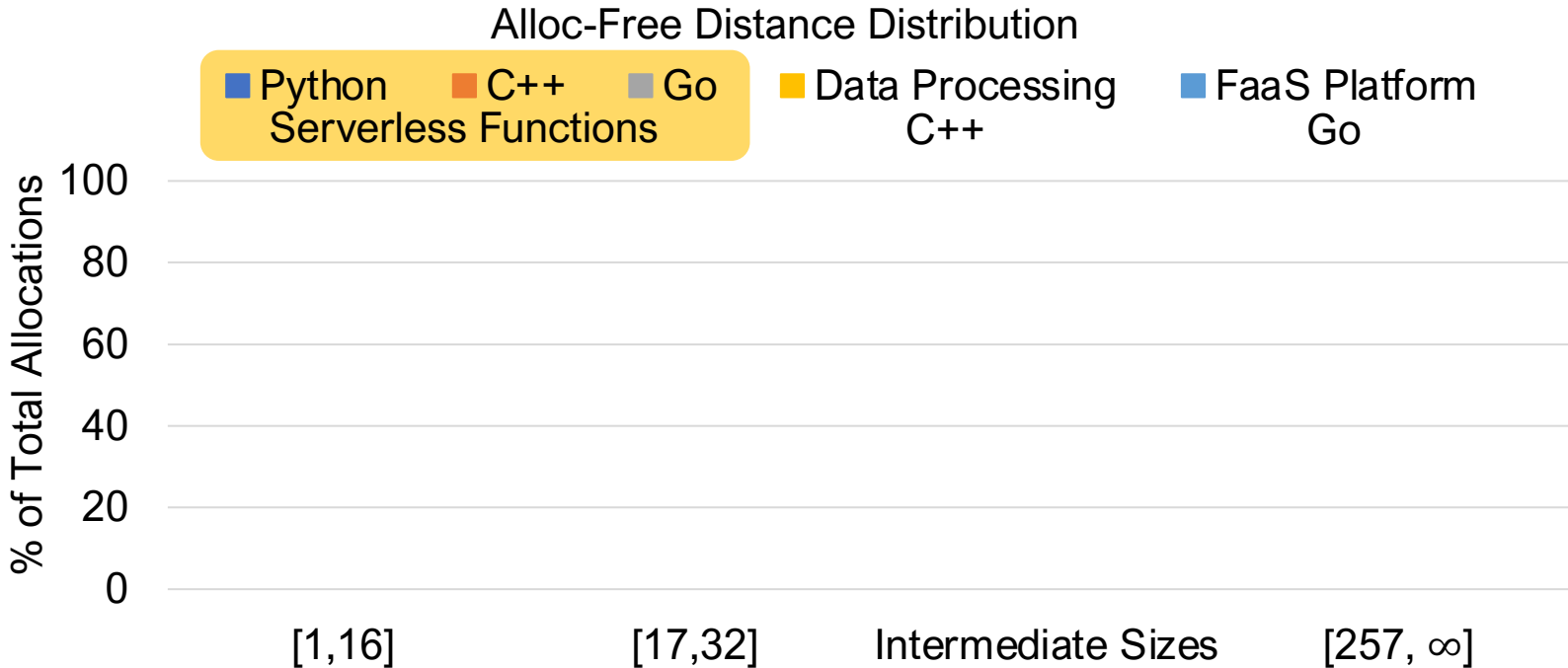


# Allocation Sizes are Small

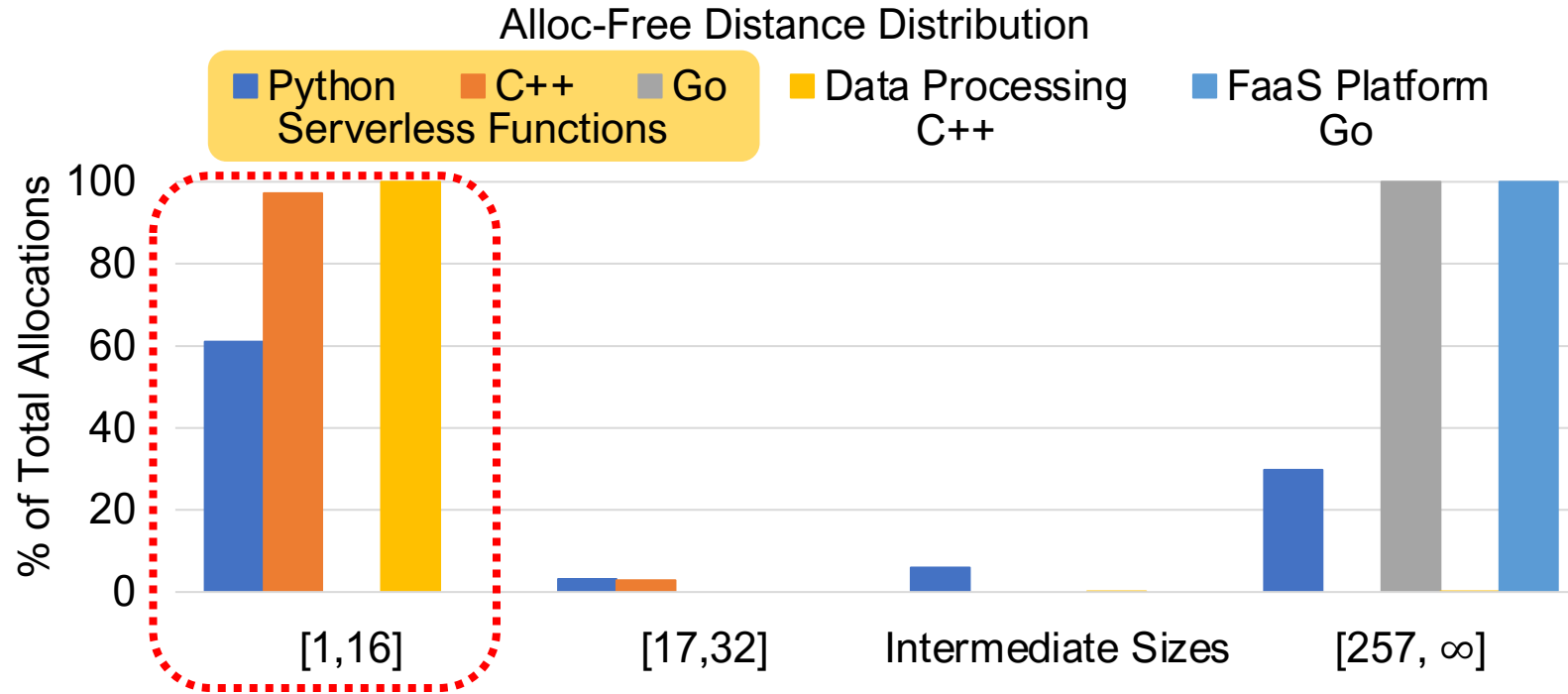


Most allocations are small → Less than 512 bytes

# Alloc-Free Distances are Bimodal

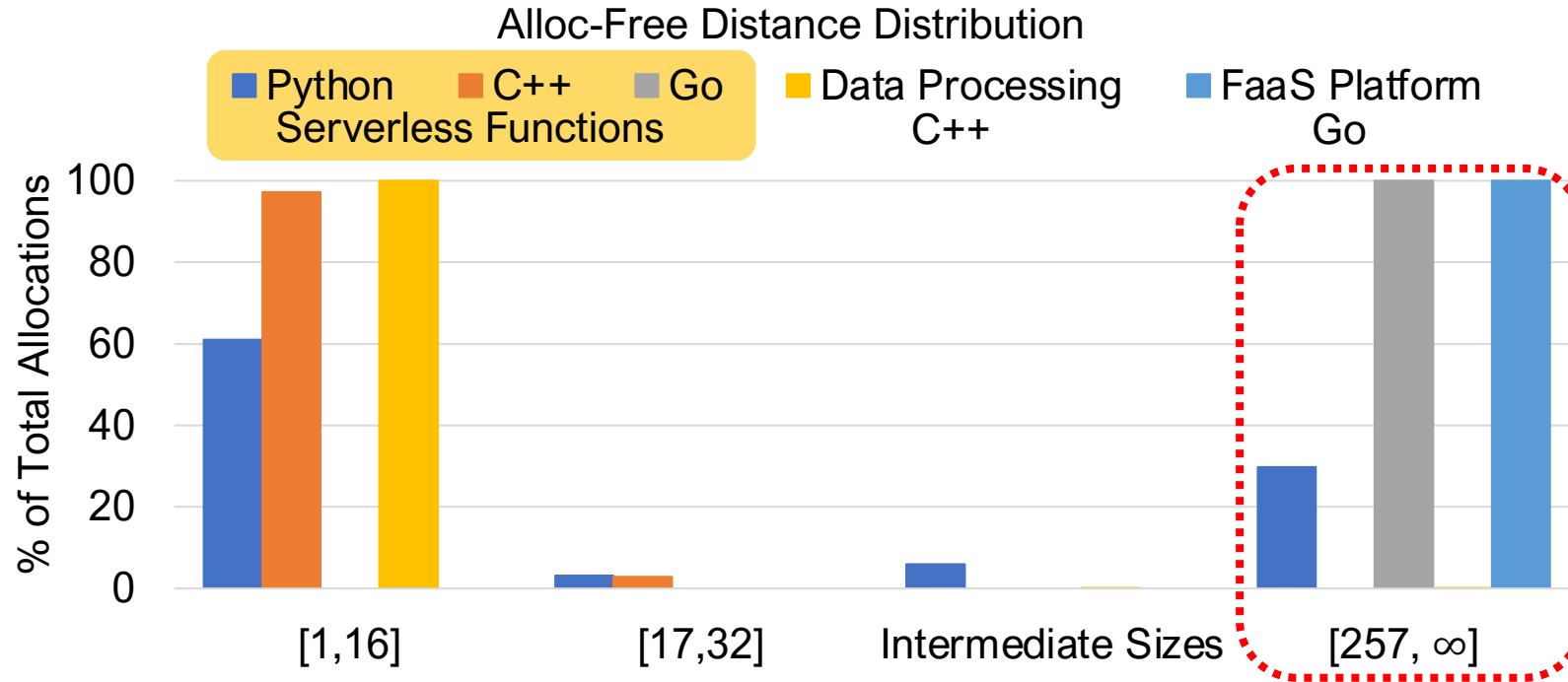


# Alloc-Free Distances are Bimodal



Allocations are quickly freed  
→ 71% within 16 allocations

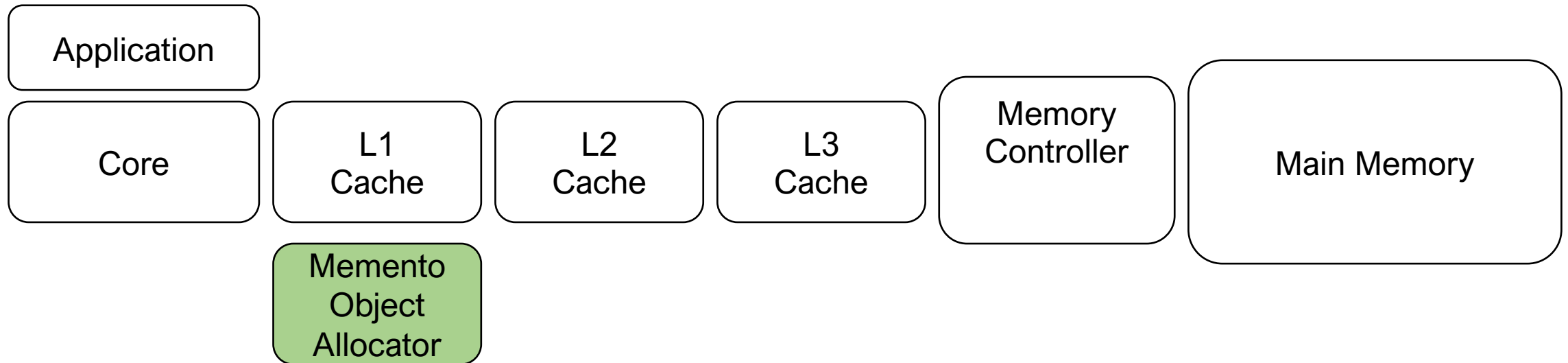
# Alloc-Free Distances are Bimodal



Allocations are quickly freed  
→ 71% within 16 allocations  
Otherwise batch-freed

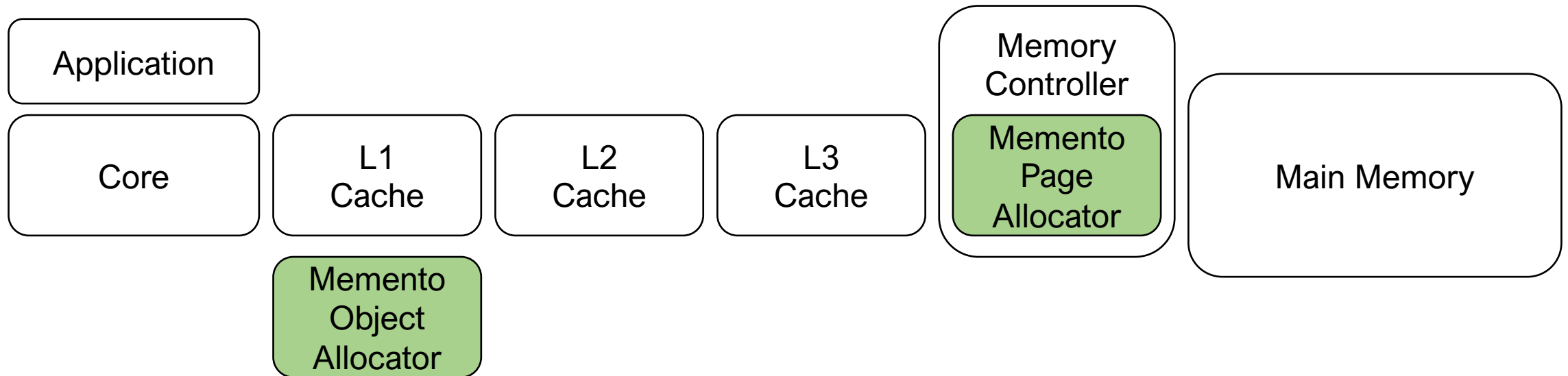
# Memento Overview

- Arena-based hardware **object allocator**



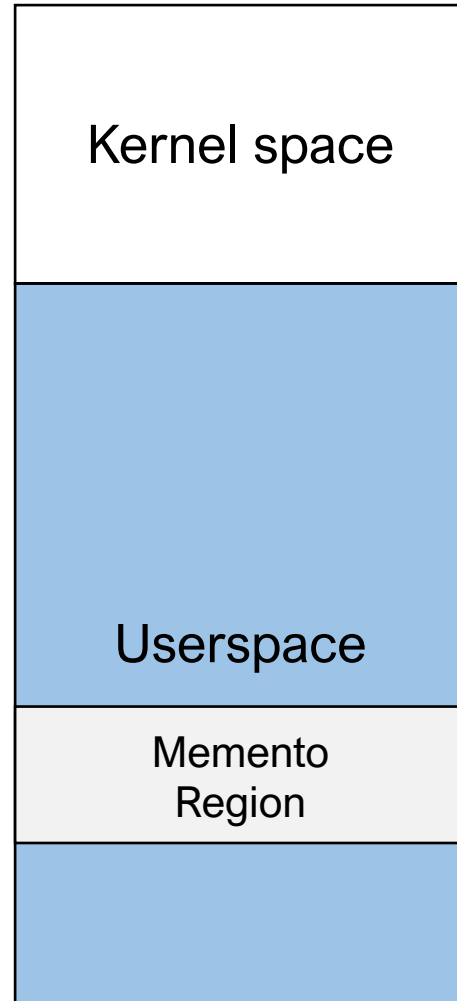
# Memento Overview

- Arena-based hardware **object allocator**
- Hardware **page allocator** for page management



# Memento Region in Virtual Address Space

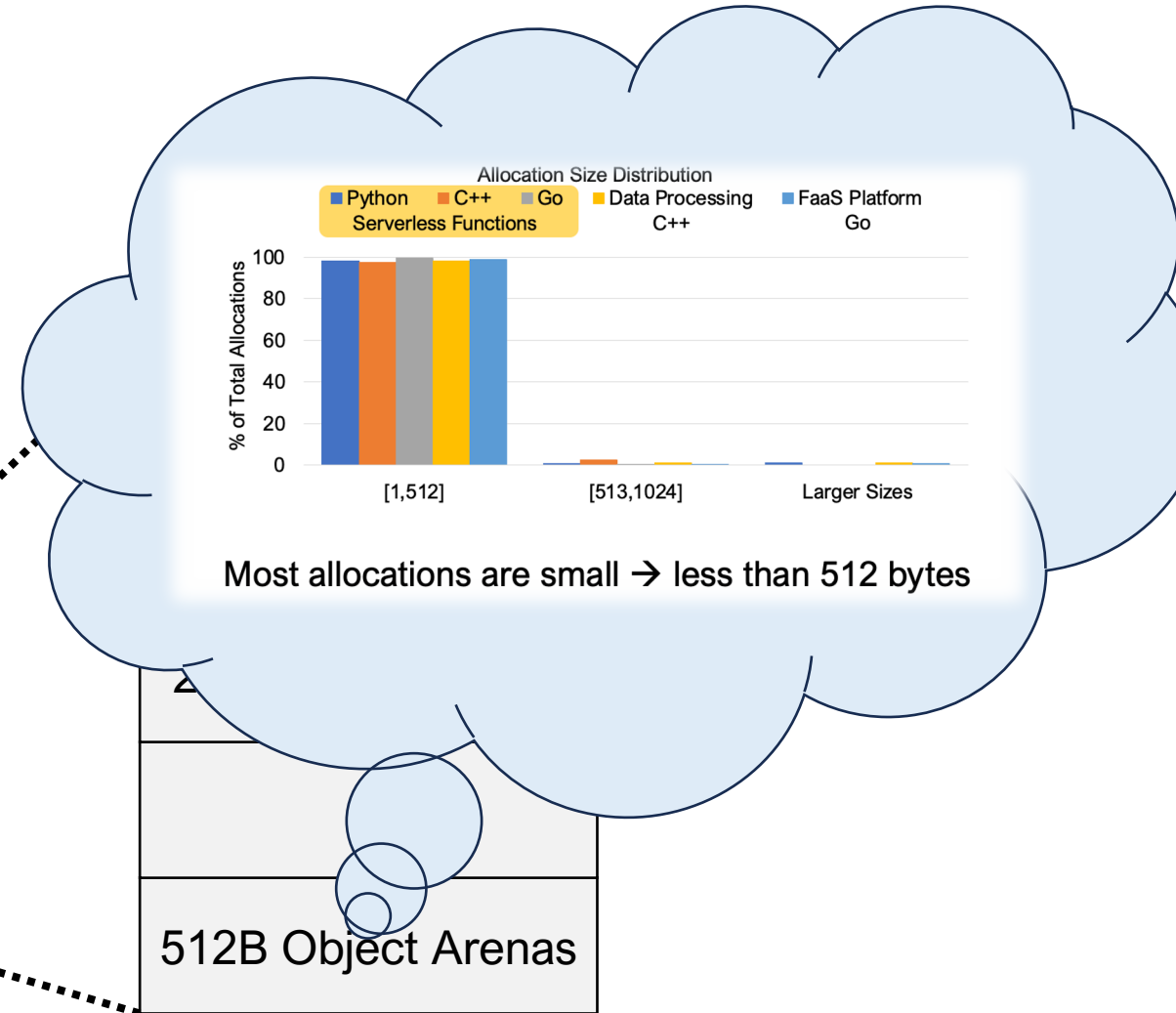
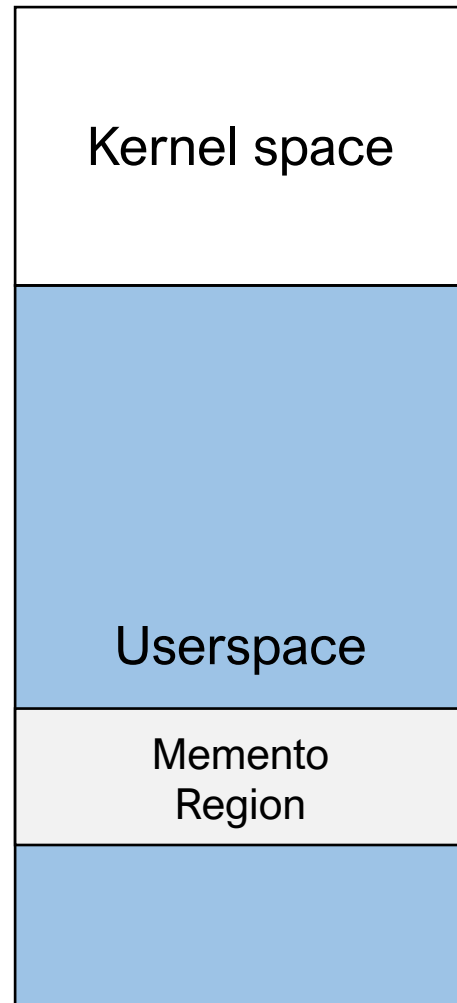
Virtual Address Space





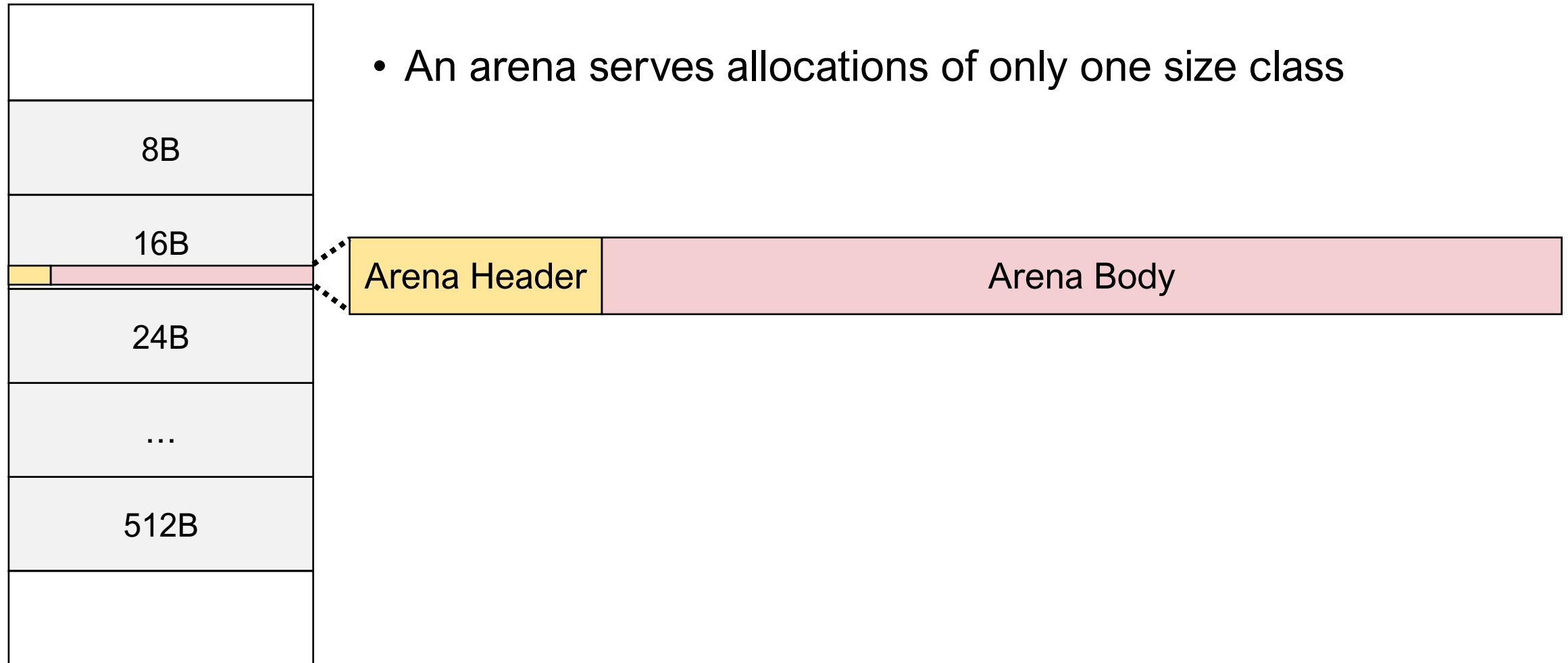
# Evenly Divided Between Size Classes

Virtual Address Space



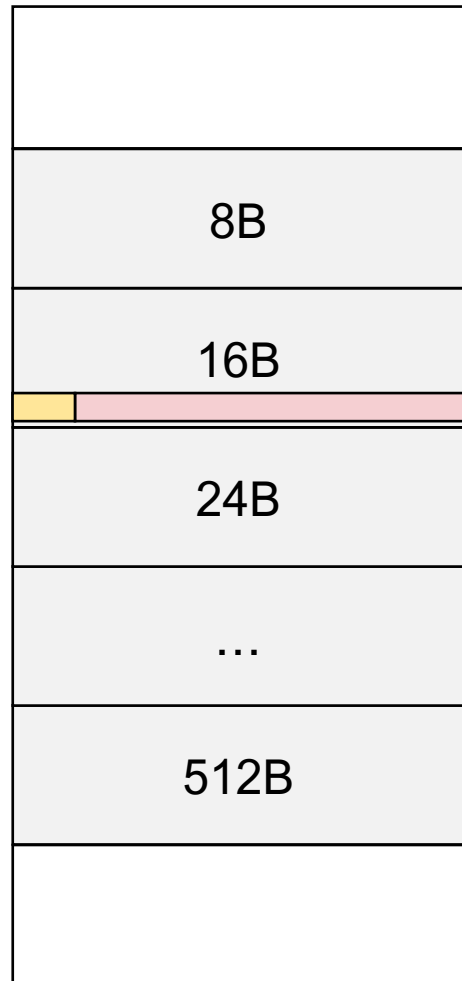
# Memento Arena

Virtual Address Space

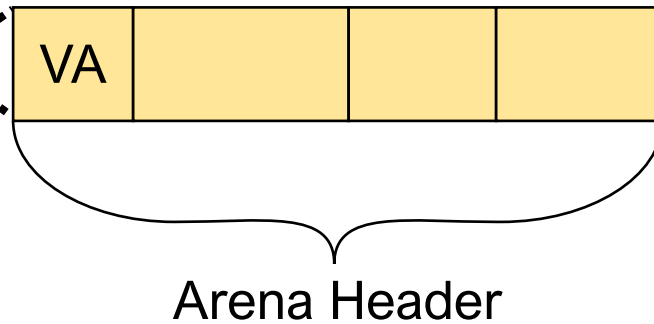


# Memento Arena Header

Virtual Address Space

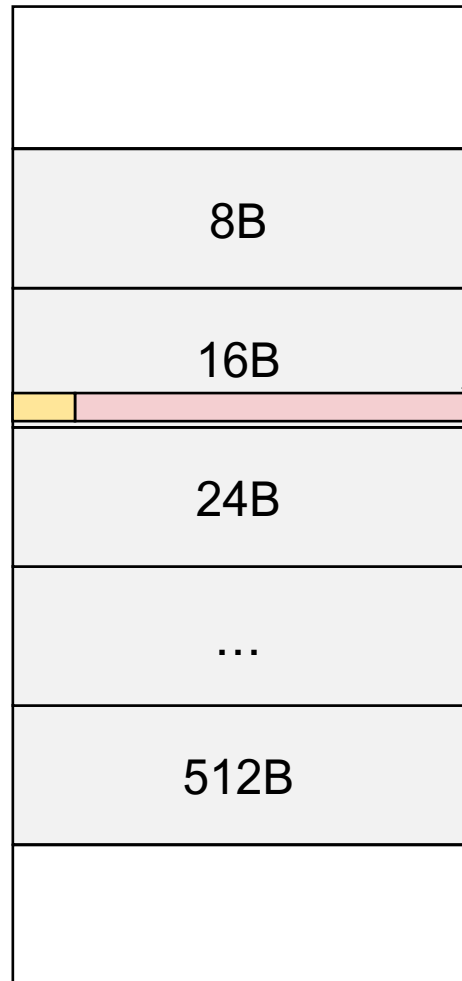


- VA: base virtual address of the arena

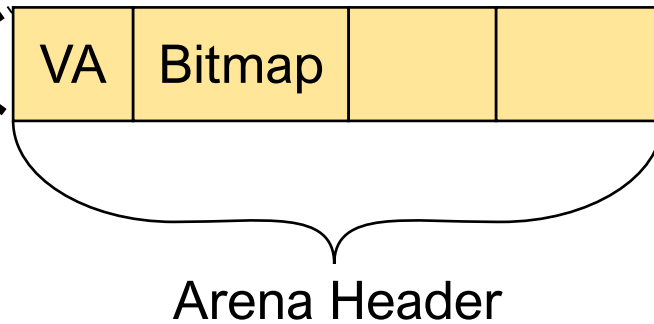


# Memento Arena Header

Virtual Address Space

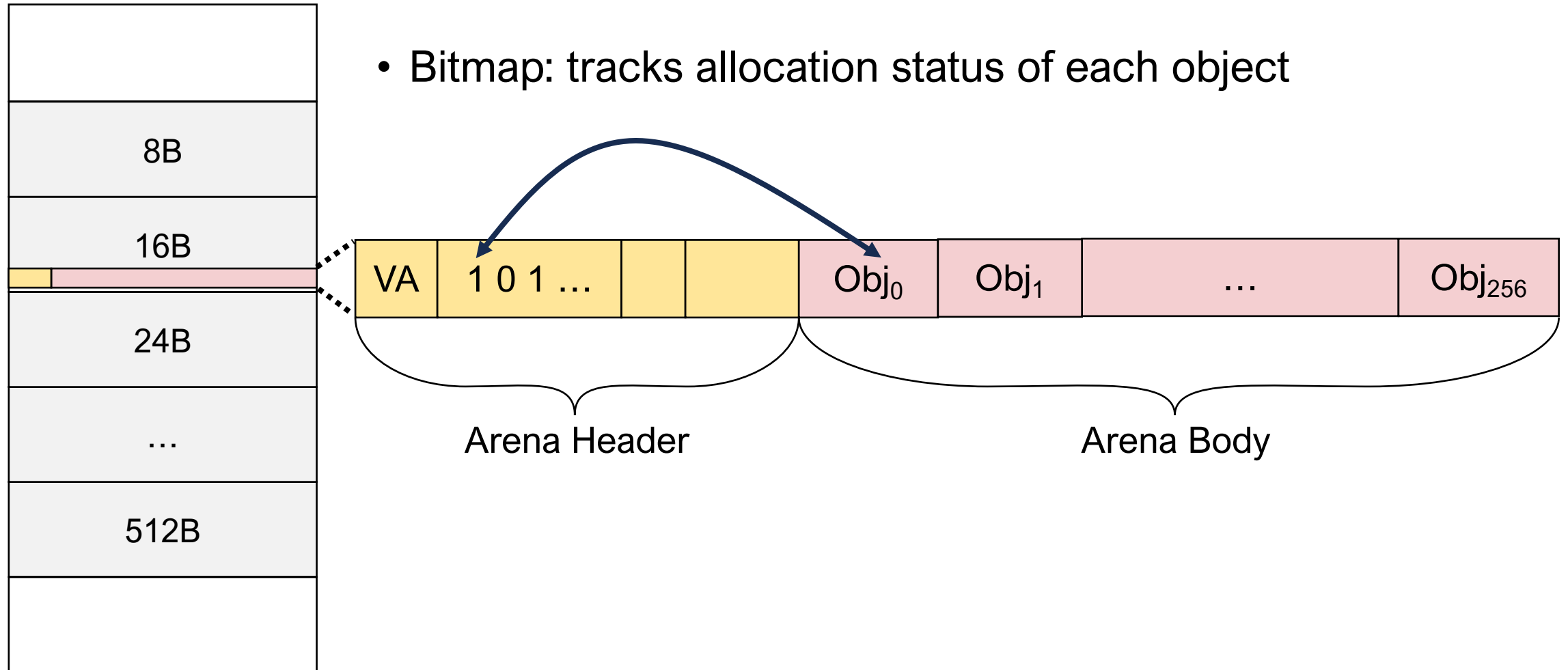


- Bitmap: tracks allocation status of each object



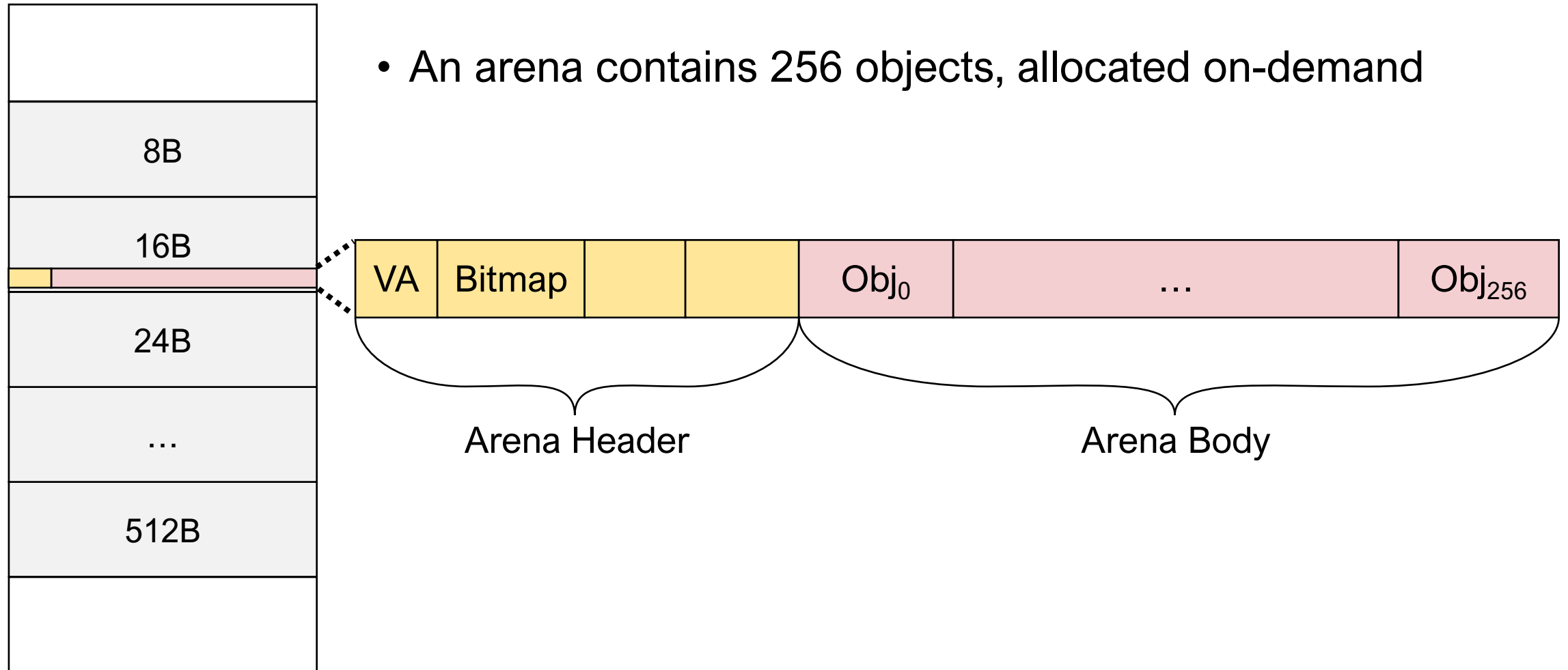
# Memento Arena Header

Virtual Address Space



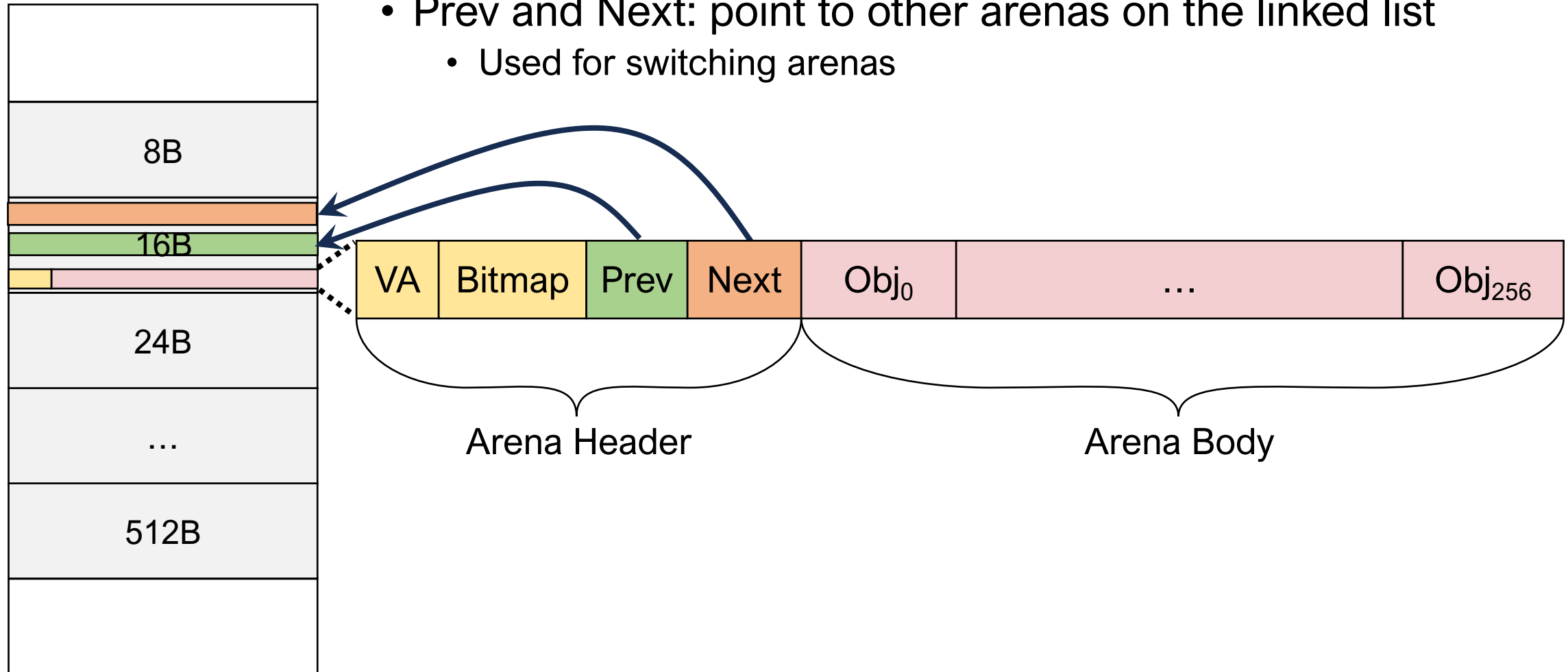
# Memento Arena

Virtual Address Space



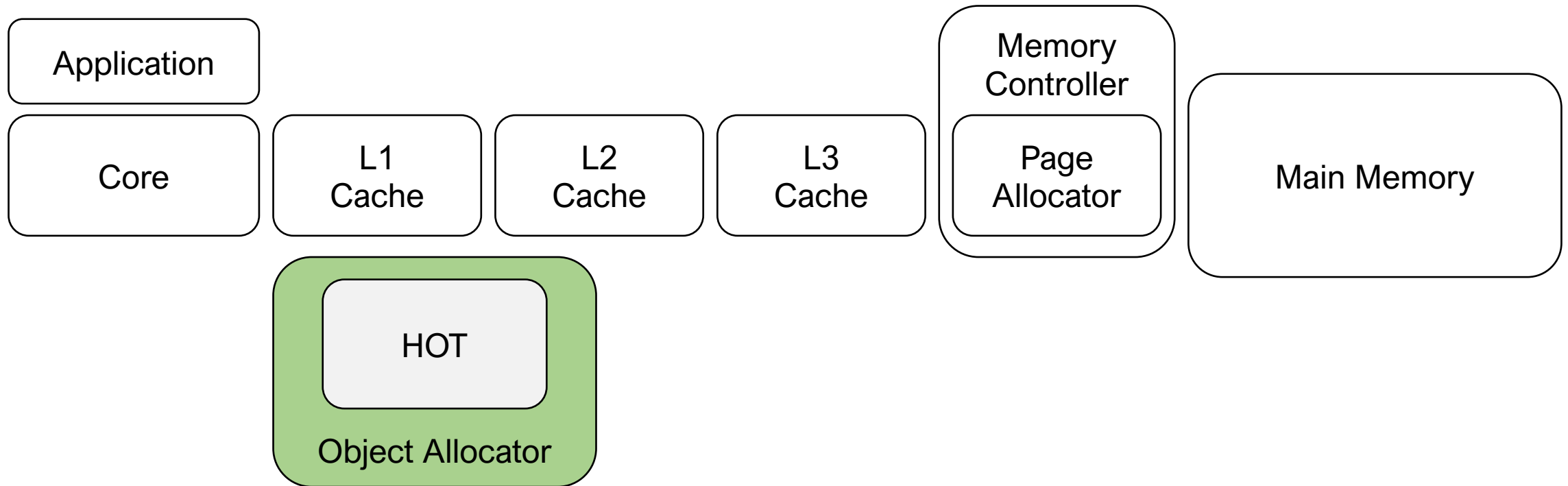
# Memento Arena Header

Virtual Address Space



# Caching Allocation Metadata in Hardware

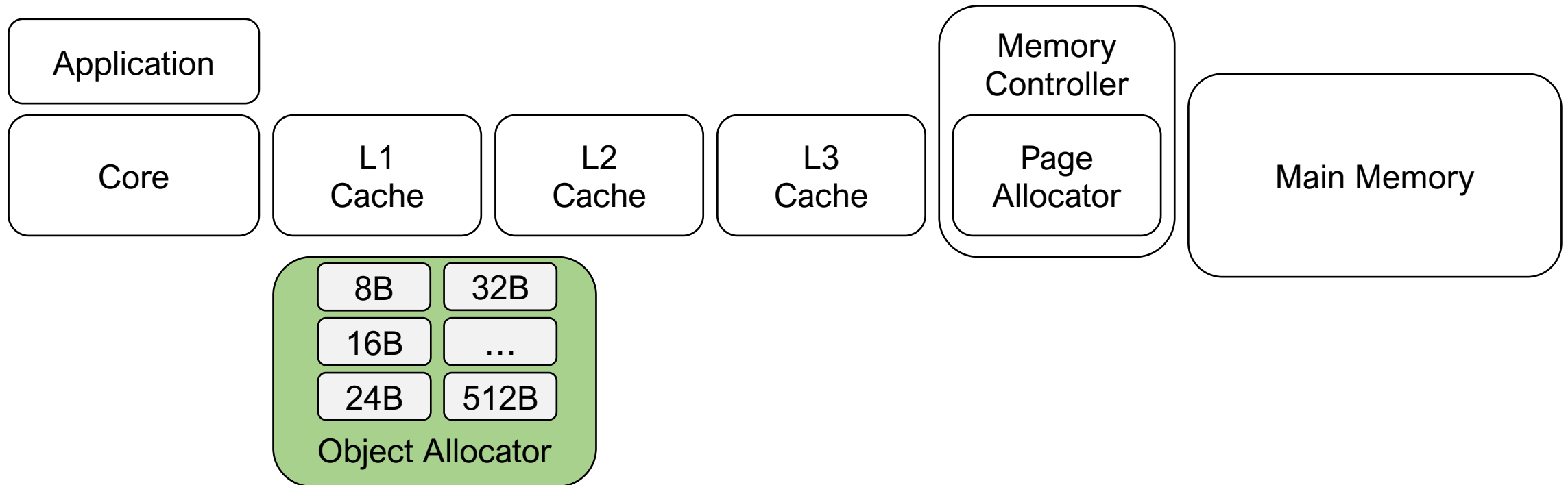
- Hardware Object Table (HOT) caches the most recently used arena headers



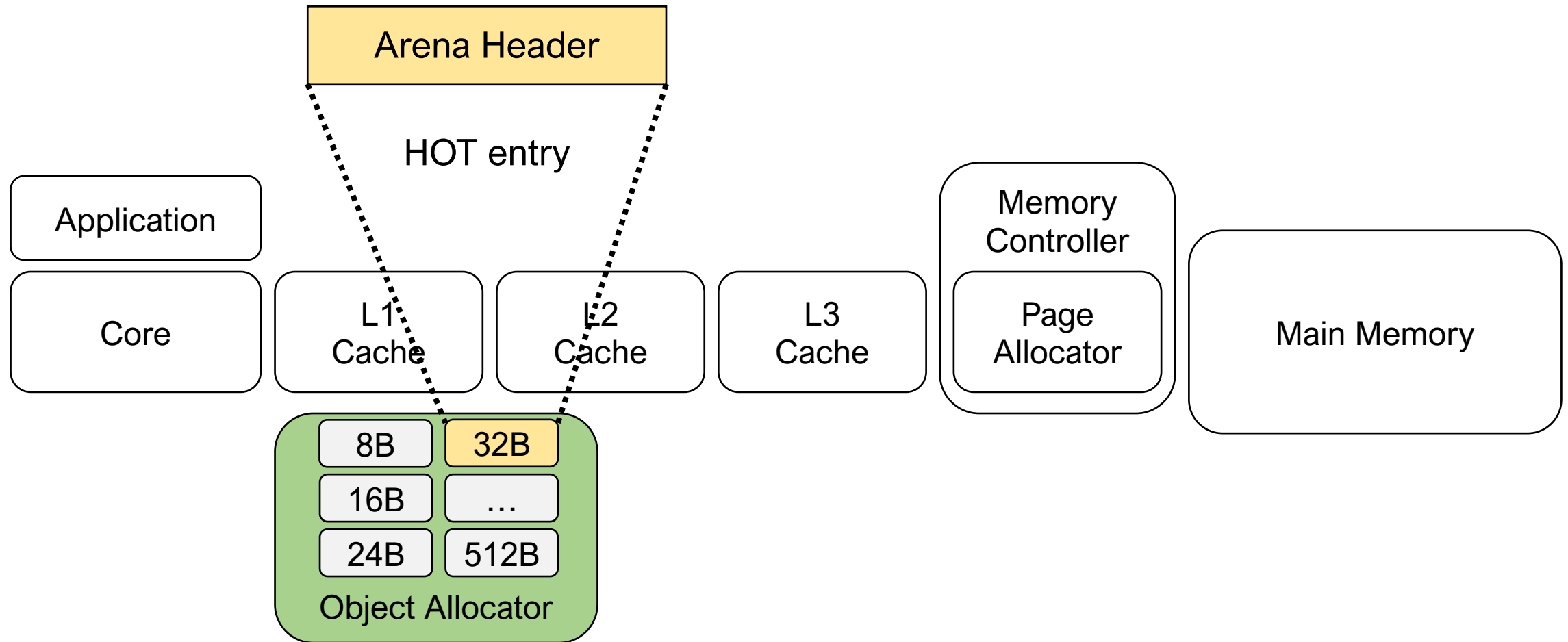


# Hardware Object Table (HOT)

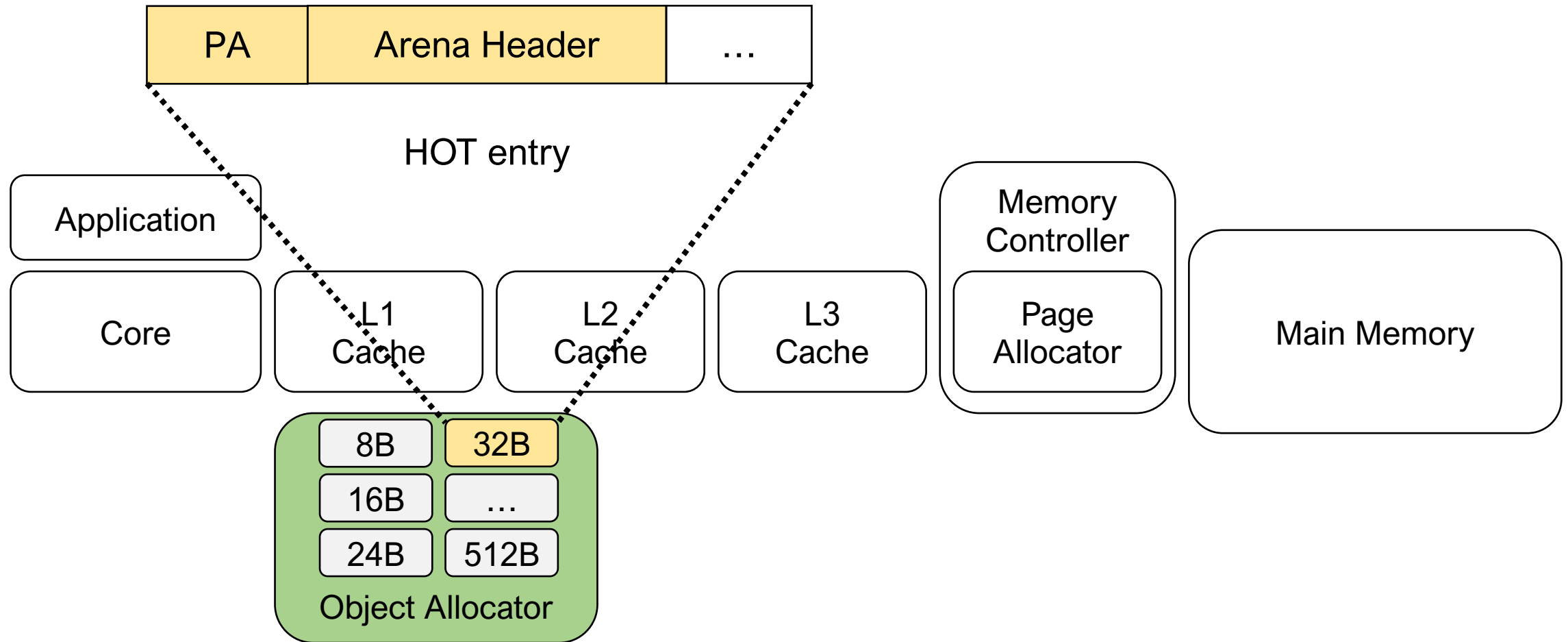
- HOT caches the most recently used arena headers
- One entry for each size class



# Hardware Object Table (HOT) Entries

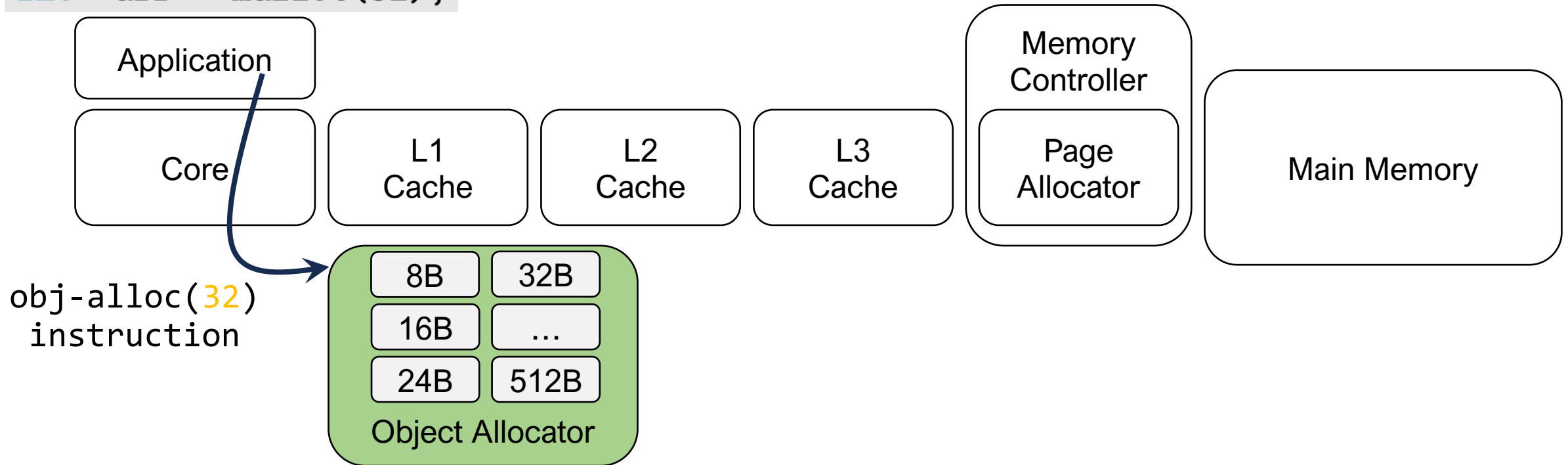


# Hardware Object Table (HOT) Entries

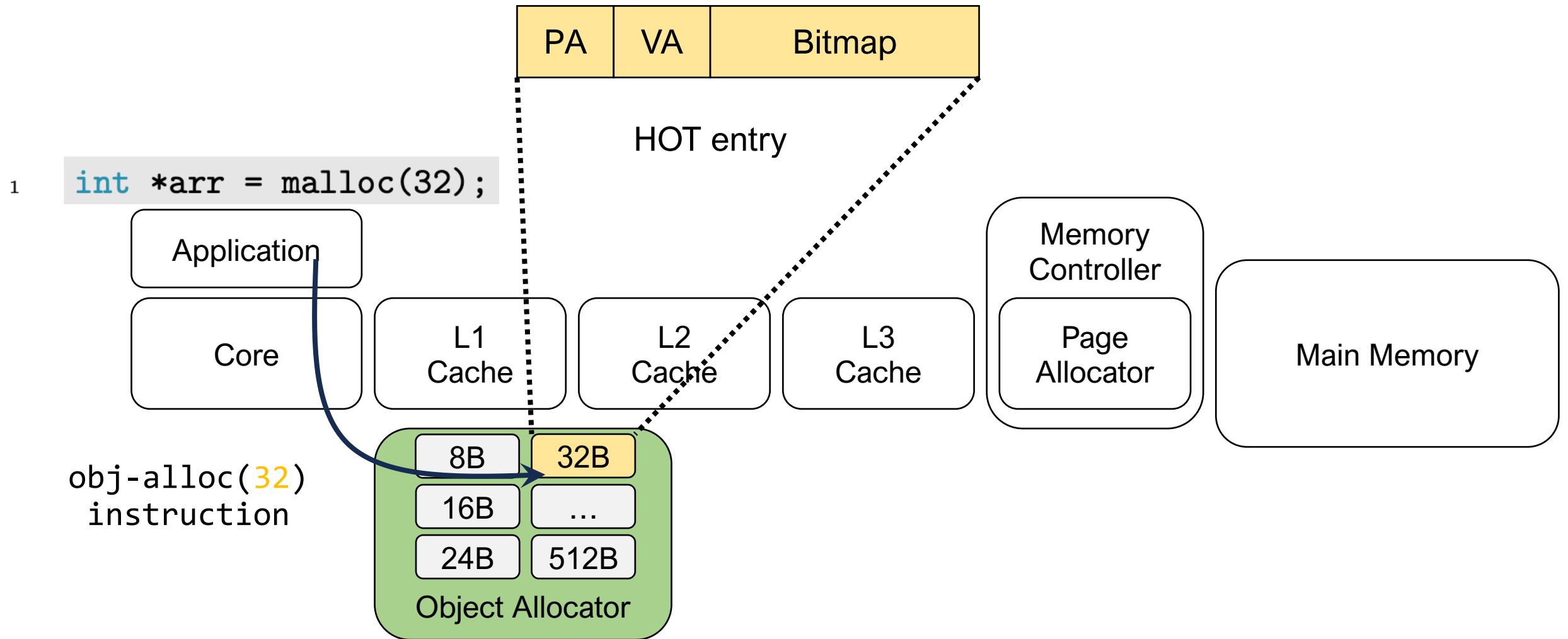


# Memento Object Allocation

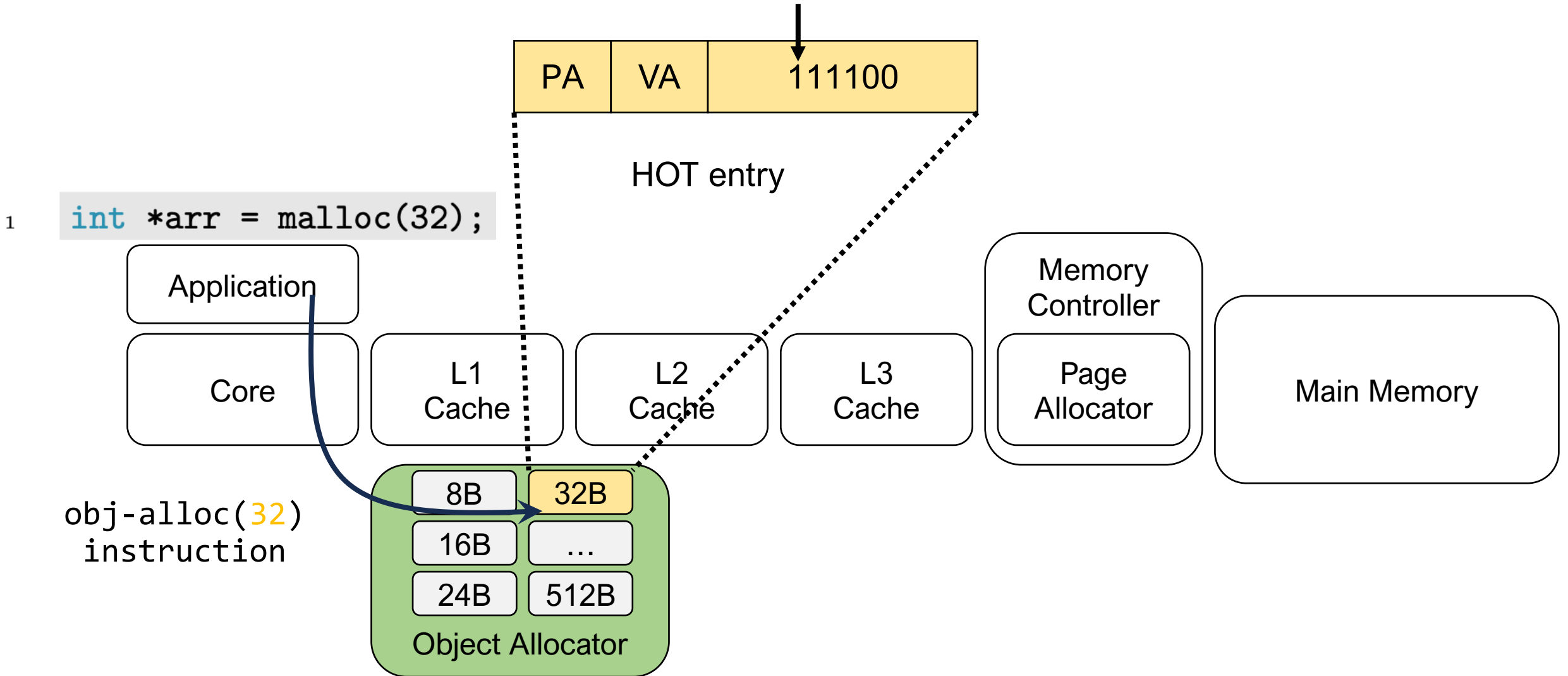
```
1 int *arr = malloc(32);
```



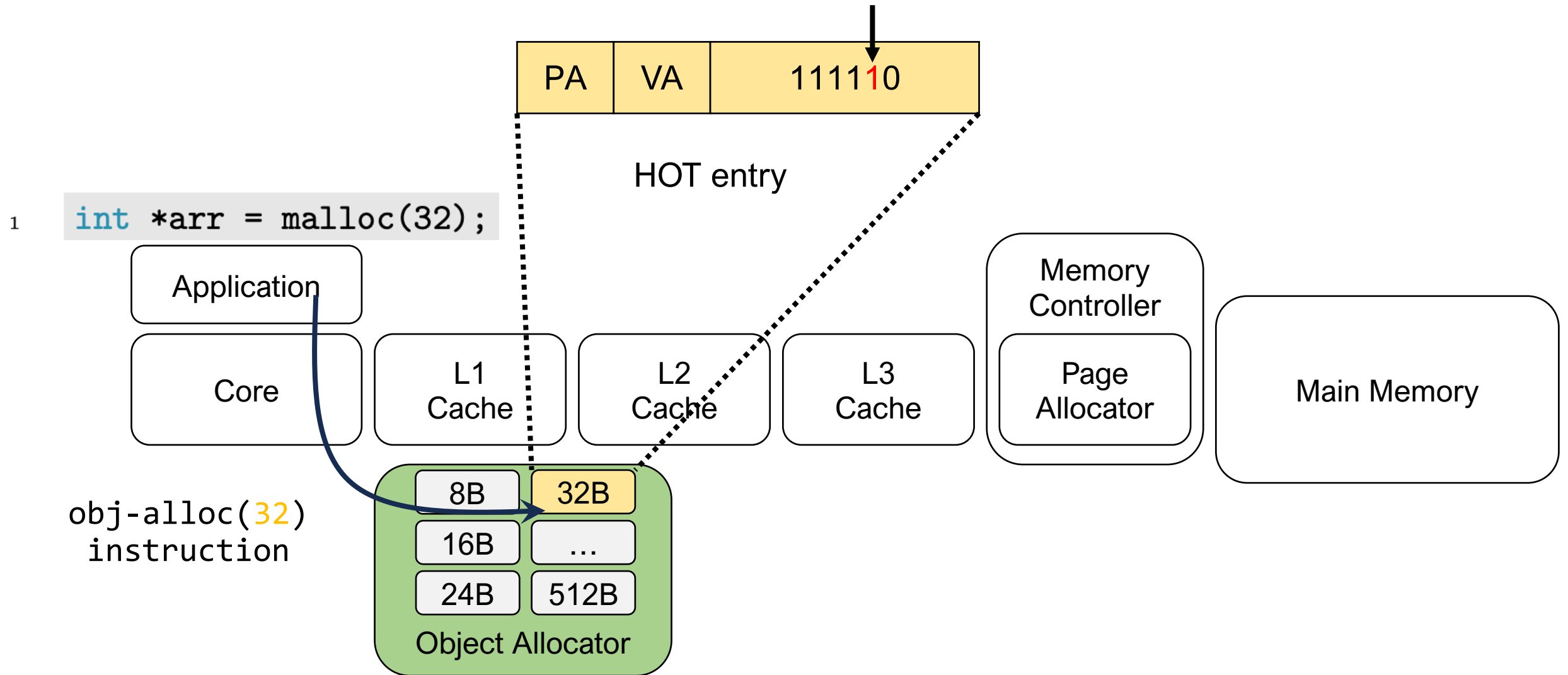
# Memento Object Allocation



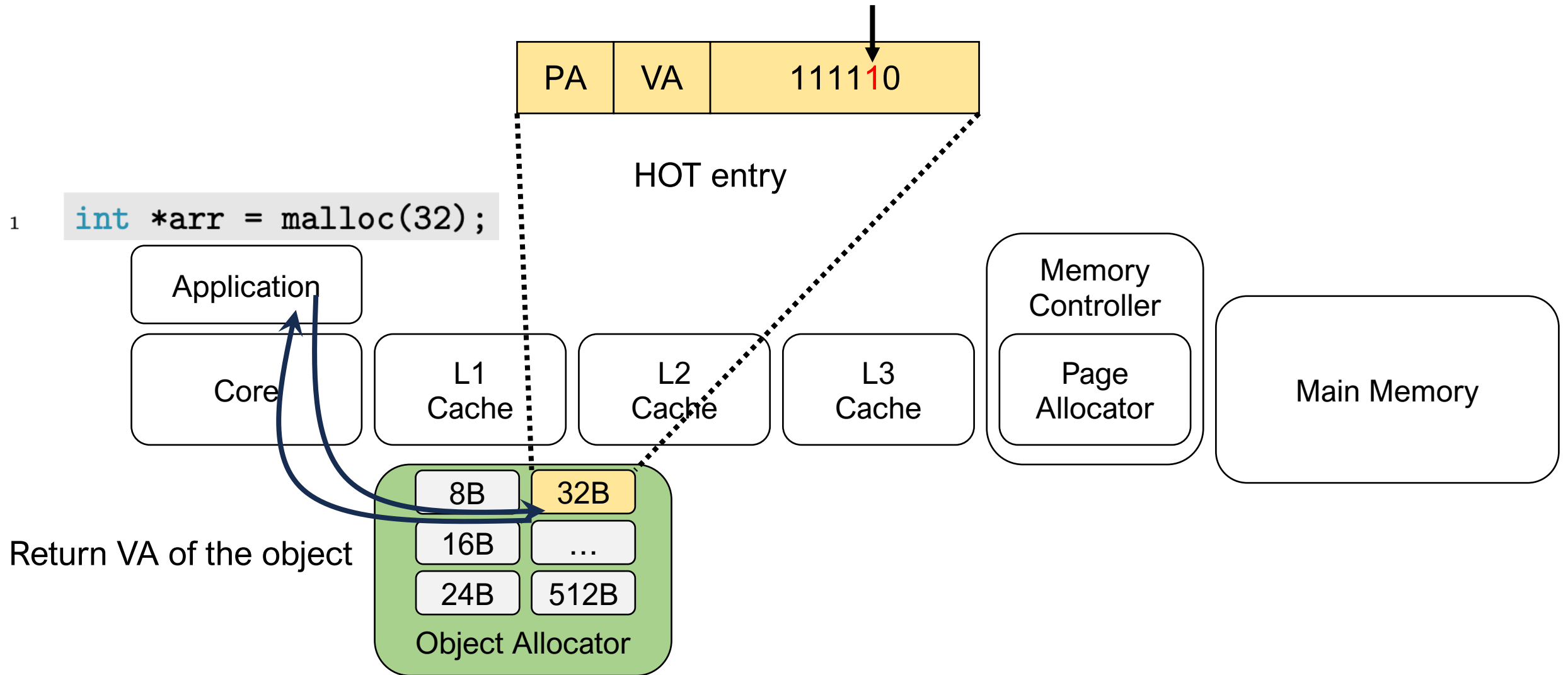
# Memento Object Allocation



# Memento Object Allocation



# Memento Object Allocation



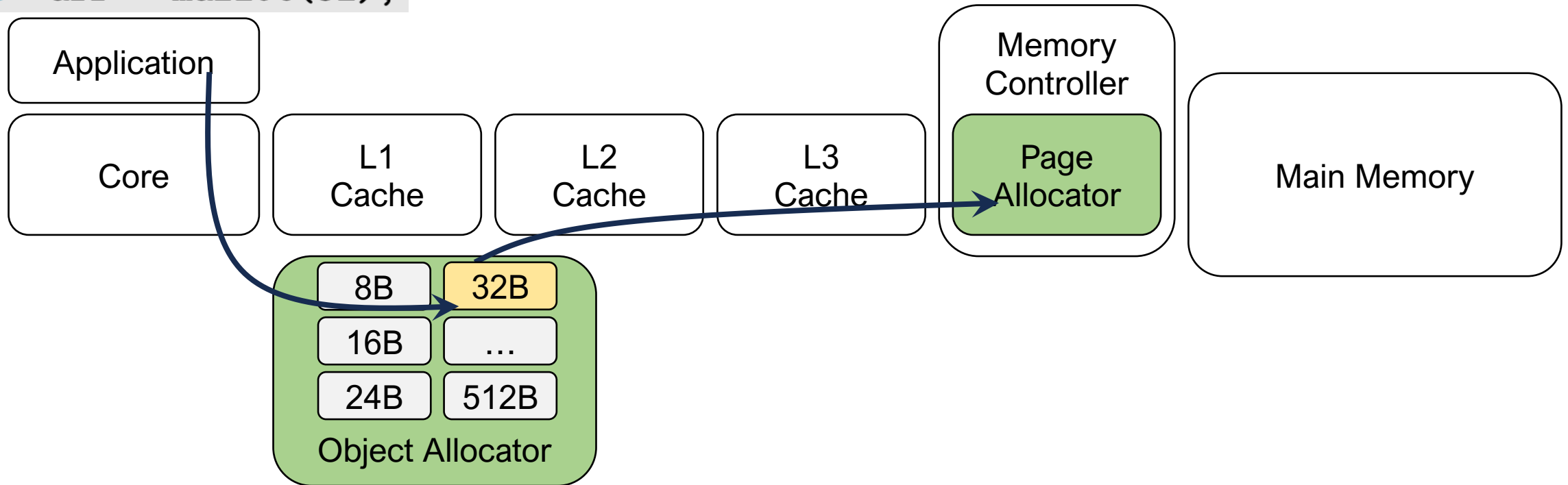
Return VA of the object



# Allocate New Arenas from Page Allocator

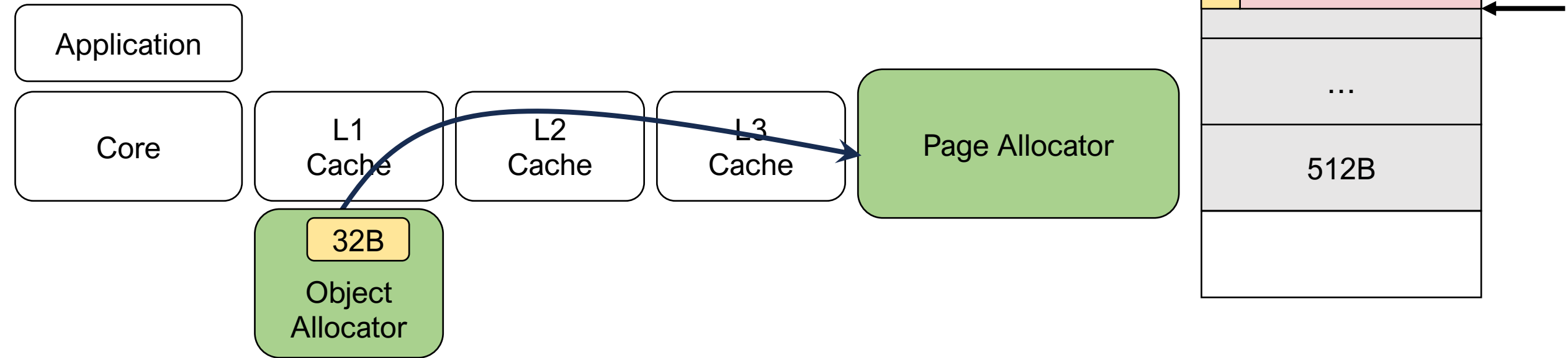
- Arenas are allocated by Memento's page allocator on-demand

```
1 int *arr = malloc(32);
```



# Memento Page Allocation

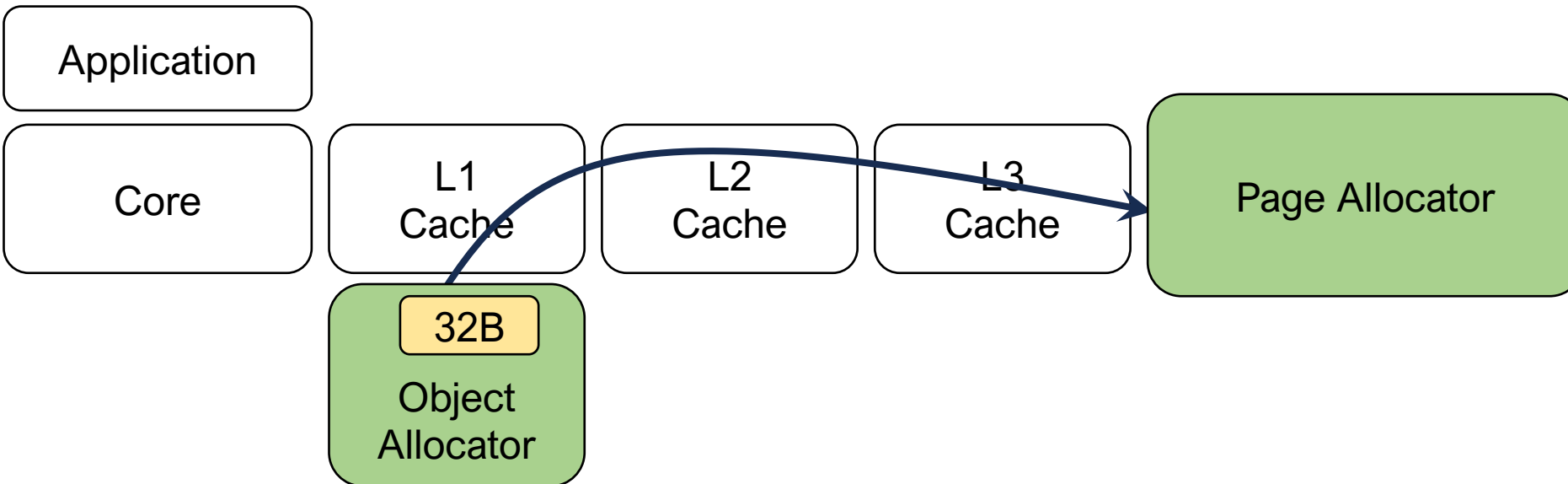
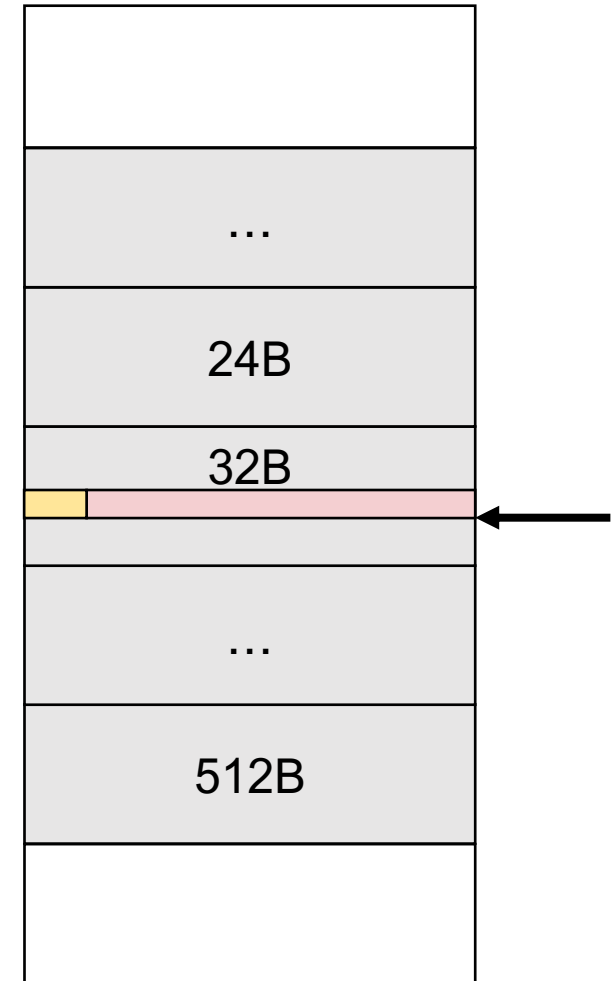
- Memento maintains a per size class allocation pointer



# Memento Page Allocation

- Memento maintains a per size class allocation pointer
- Increment pointer to get new arena

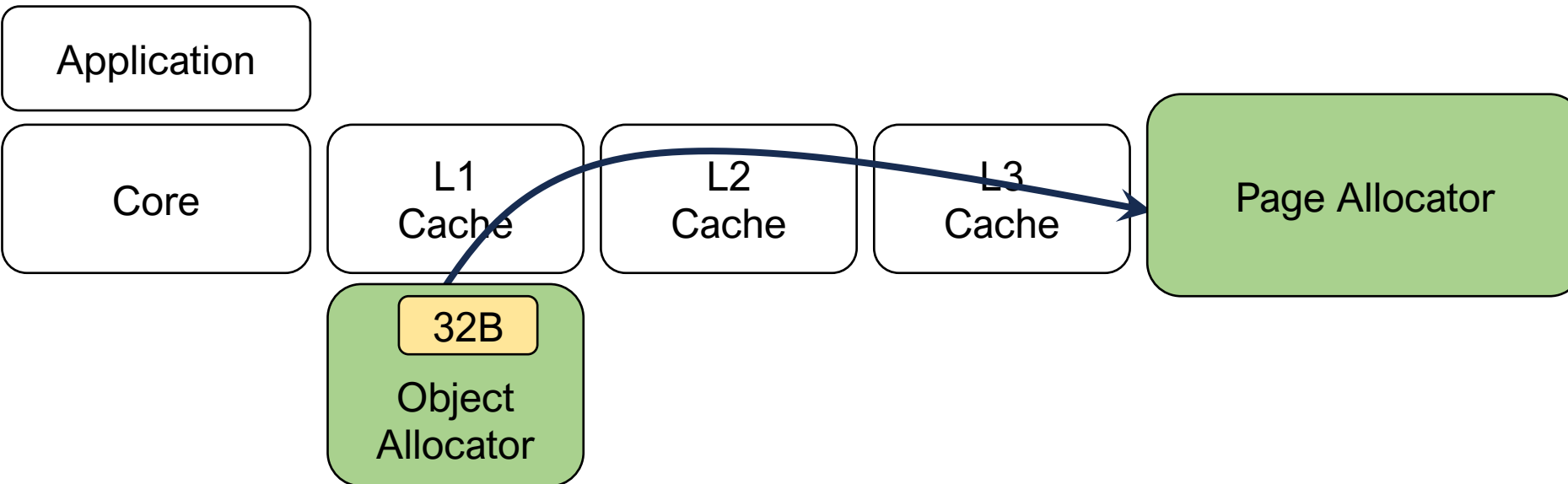
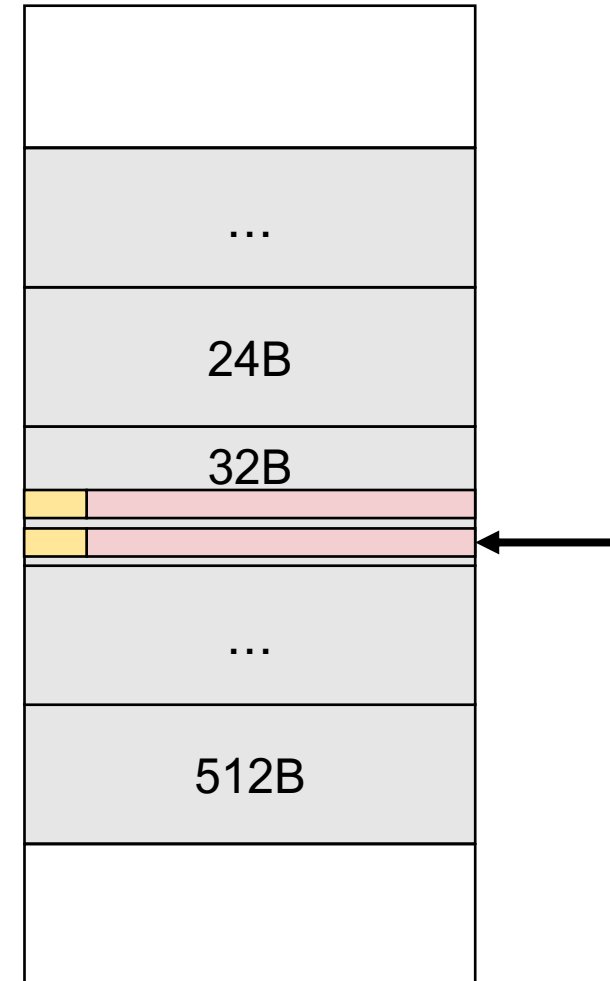
Virtual Address Space



# Memento Page Allocation

- Memento maintains a per size class allocation pointer
- Increment pointer to get new arena

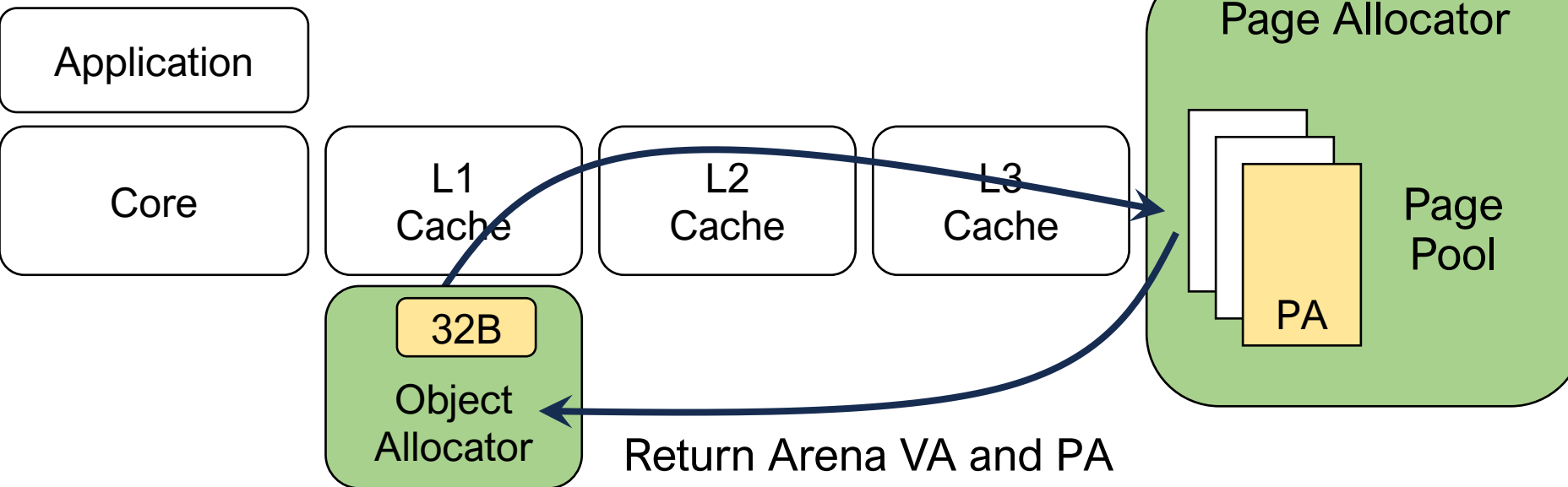
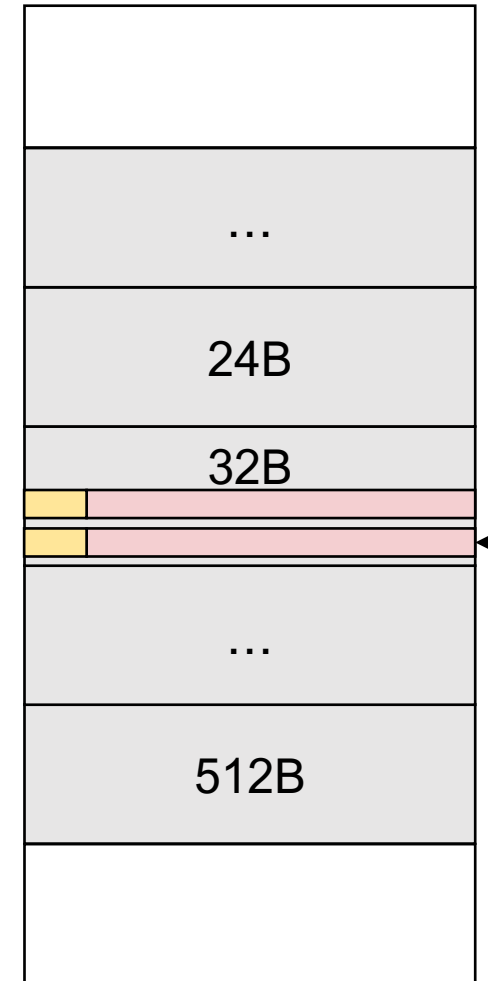
Virtual Address Space



# Memento Page Allocation

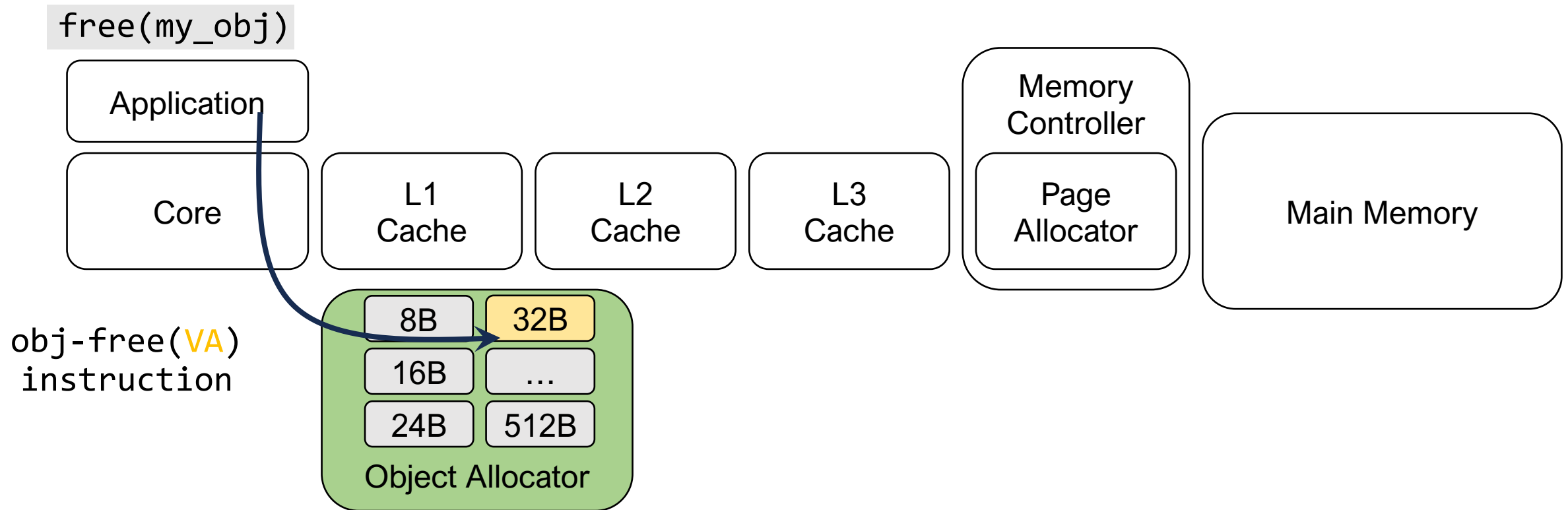
- Memento maintains a per size class allocation pointer
- Increment pointer to get new arena
- Allocate *only first* page for the arena

Virtual Address Space



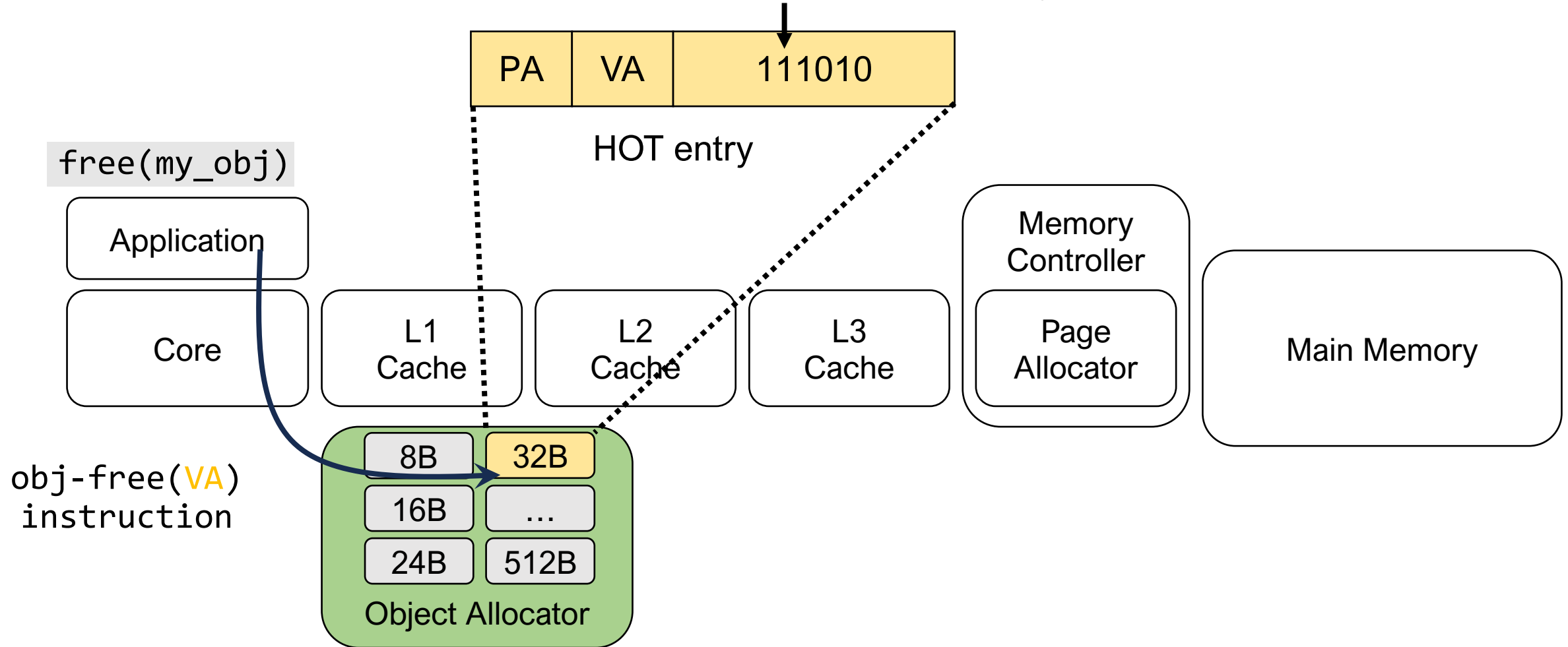
# Memory Free with Memento

- Calculate arena VA and compare it to the HOT entry



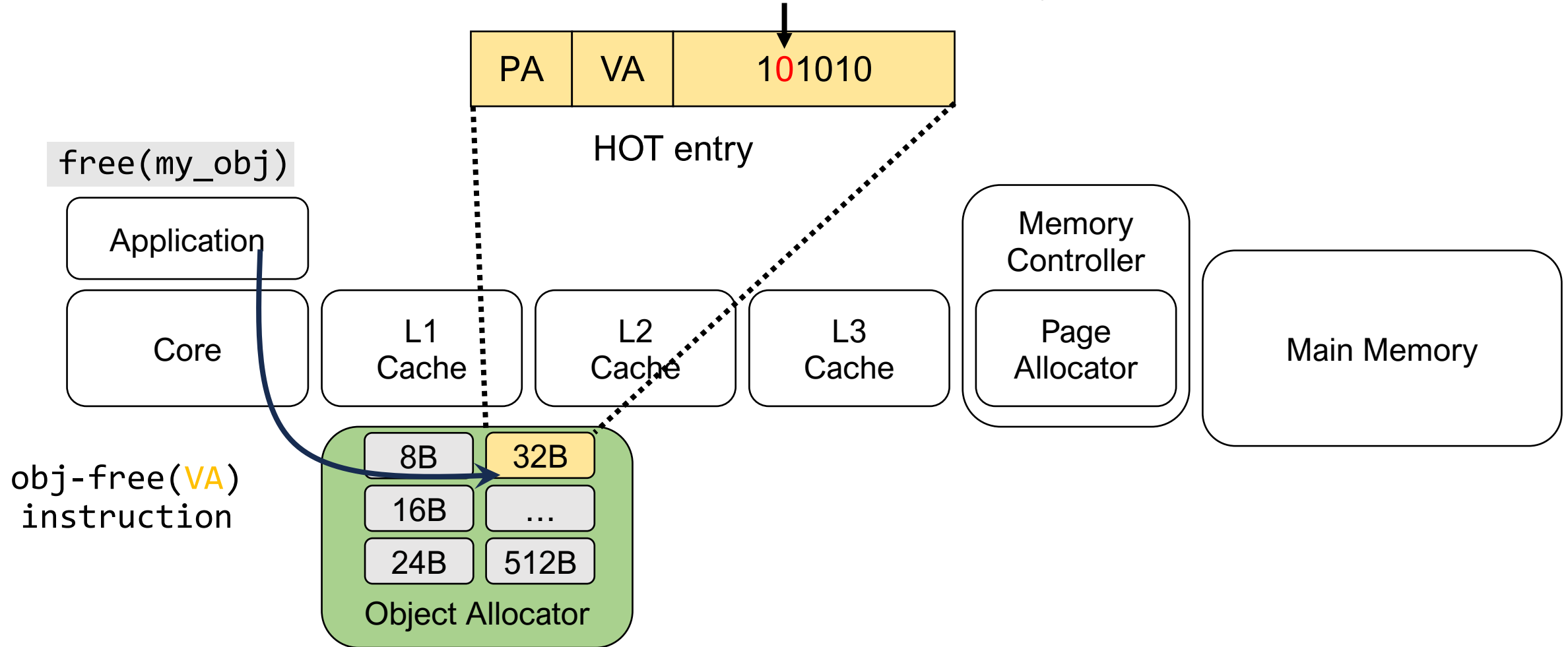
# Memory Free with Memento

- Calculate arena VA and compare it to the HOTA entry



# Memory Free with Memento

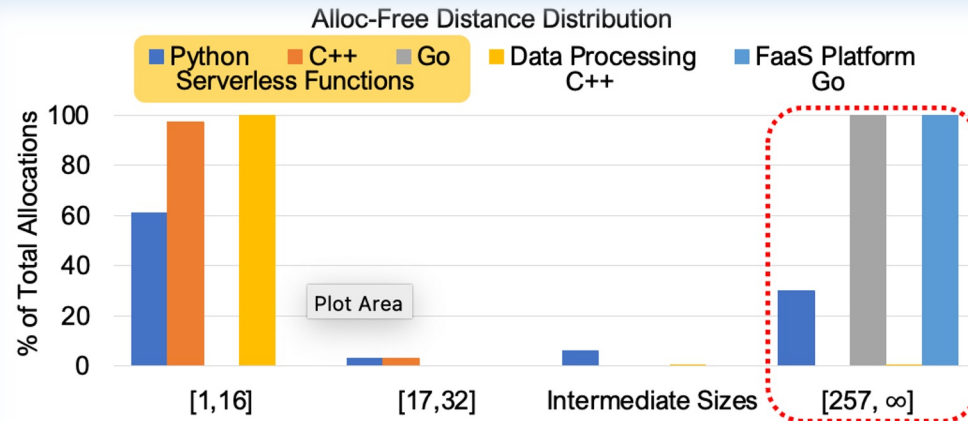
- Calculate arena VA and compare it to the HOT entry





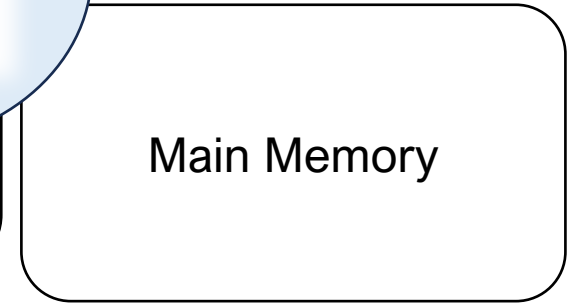
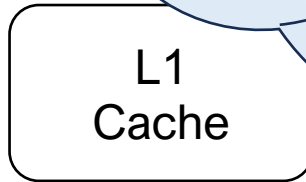
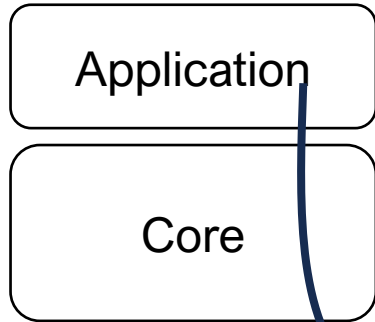
# Memory Free with Me

- Calculate arena VA

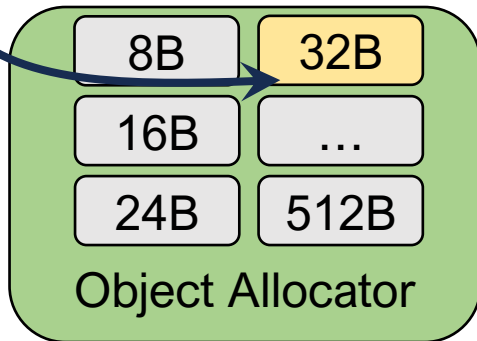


71% are freed within 16 allocations  
Or are freed when the process exits

`free(my_obj)`

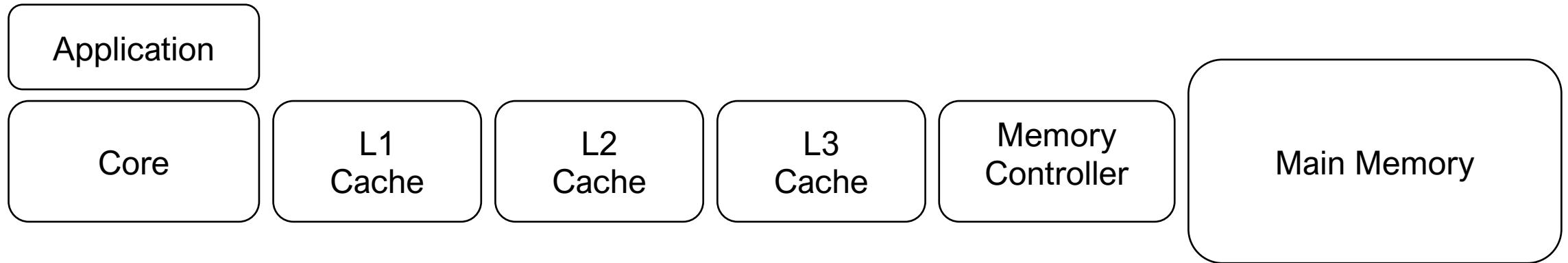


`obj-free(VA)`  
instruction



# Unnecessary Hardware Operations

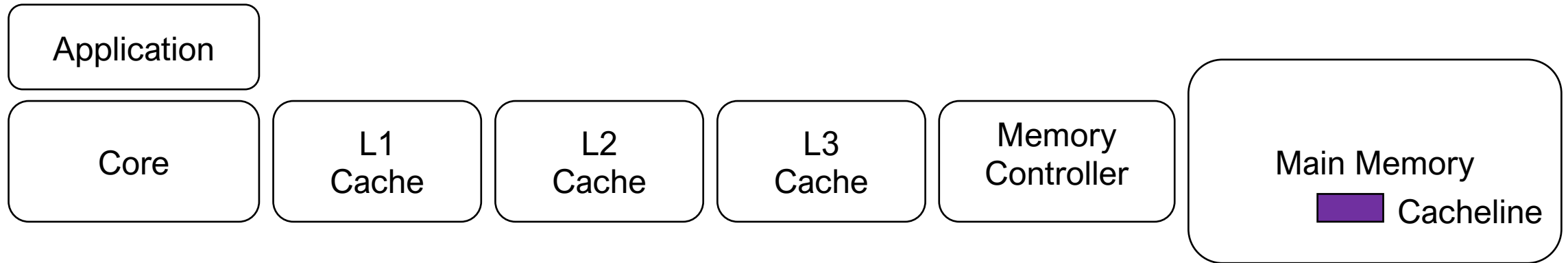
```
1 int *arr = malloc(32);  
2 arr[2] = 5;
```



# Wasted Fetches from Main Memory

- Applications write immediately to allocated objects
- Cachelines may be fetched all the way from memory just to be rewritten

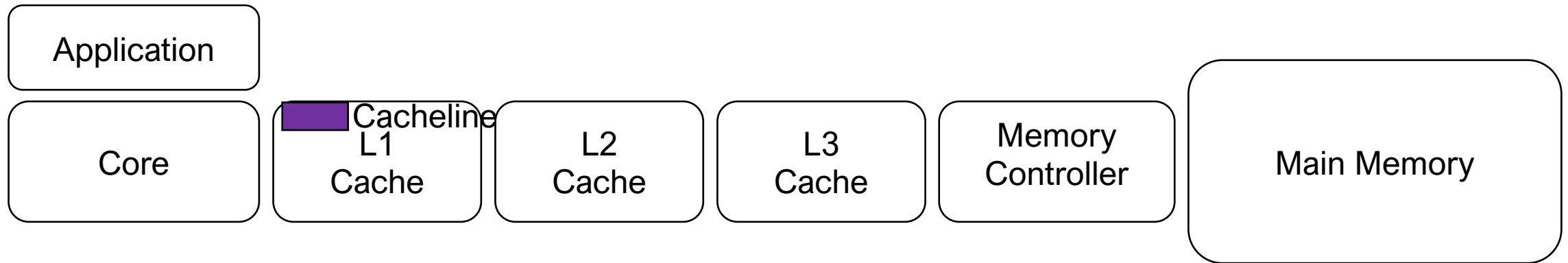
```
1 int *arr = malloc(32);  
2 arr[2] = 5;
```



# Wasted Fetches from Main Memory

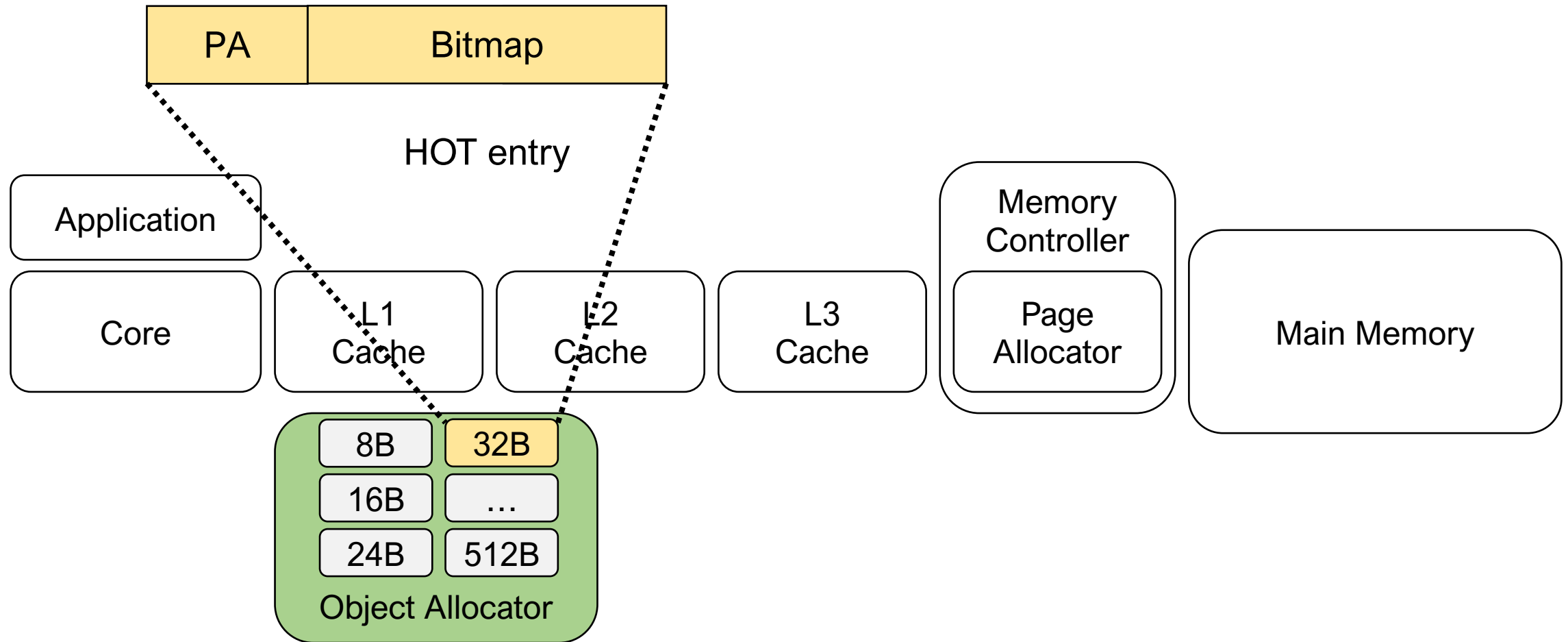
- Applications write immediately to allocated objects
- Cachelines may be fetched all the way from memory just to be written

```
1 int *arr = malloc(32);  
2 arr[2] = 5;
```



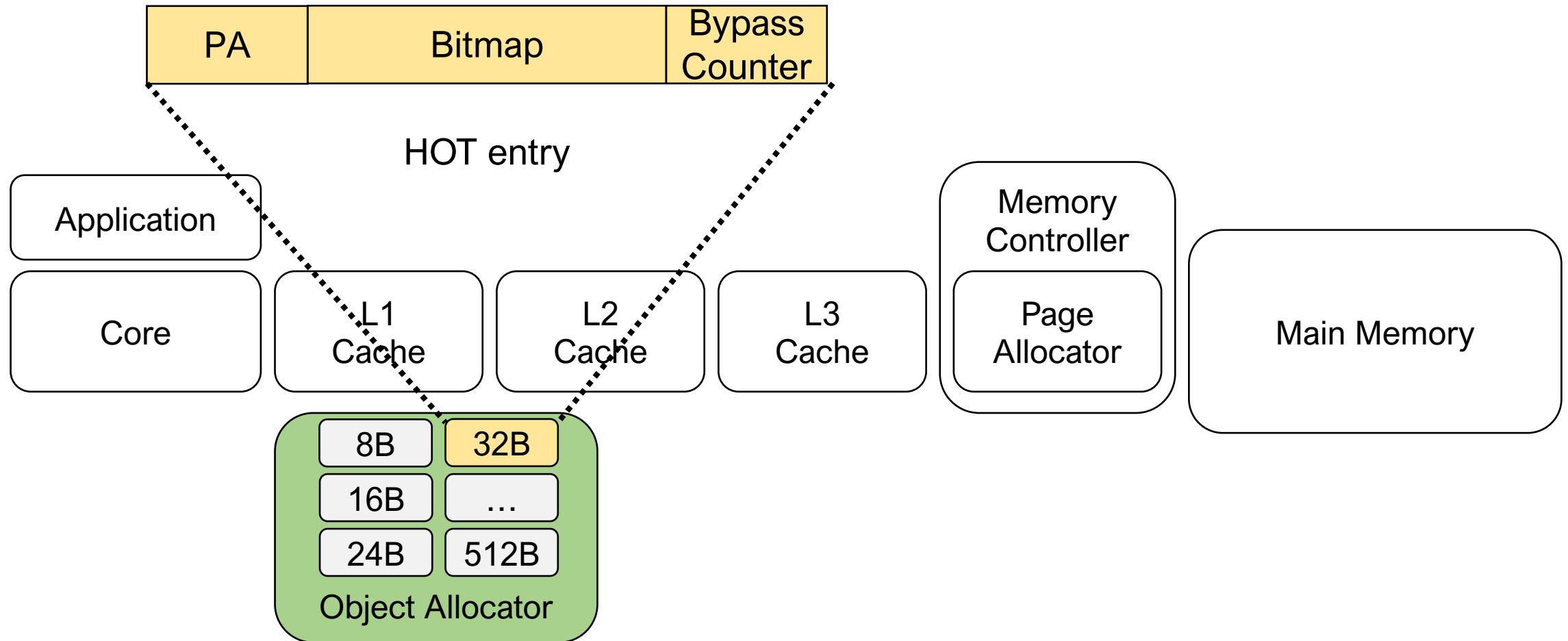
Exploit memory allocation semantics to reduce wasted fetches

# Tracking Cacheline Alloc/Access Status



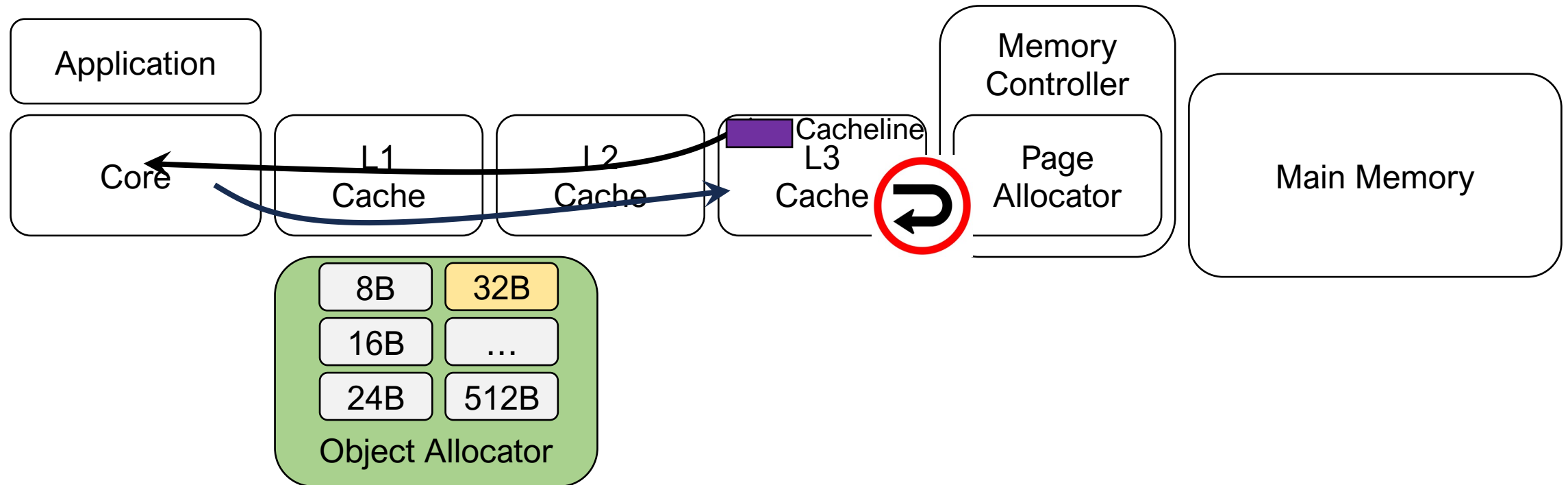
# Tracking Cacheline Alloc/Access Status

- Accesses with a bypass counter greater than bitmap count are new


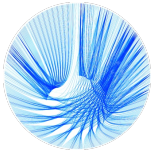






# Bypass Main Memory Accesses

- Accesses with a bypass counter greater than bitmap count are new
- Instantiate new cacheline in L3 cache
- Bypass main memory

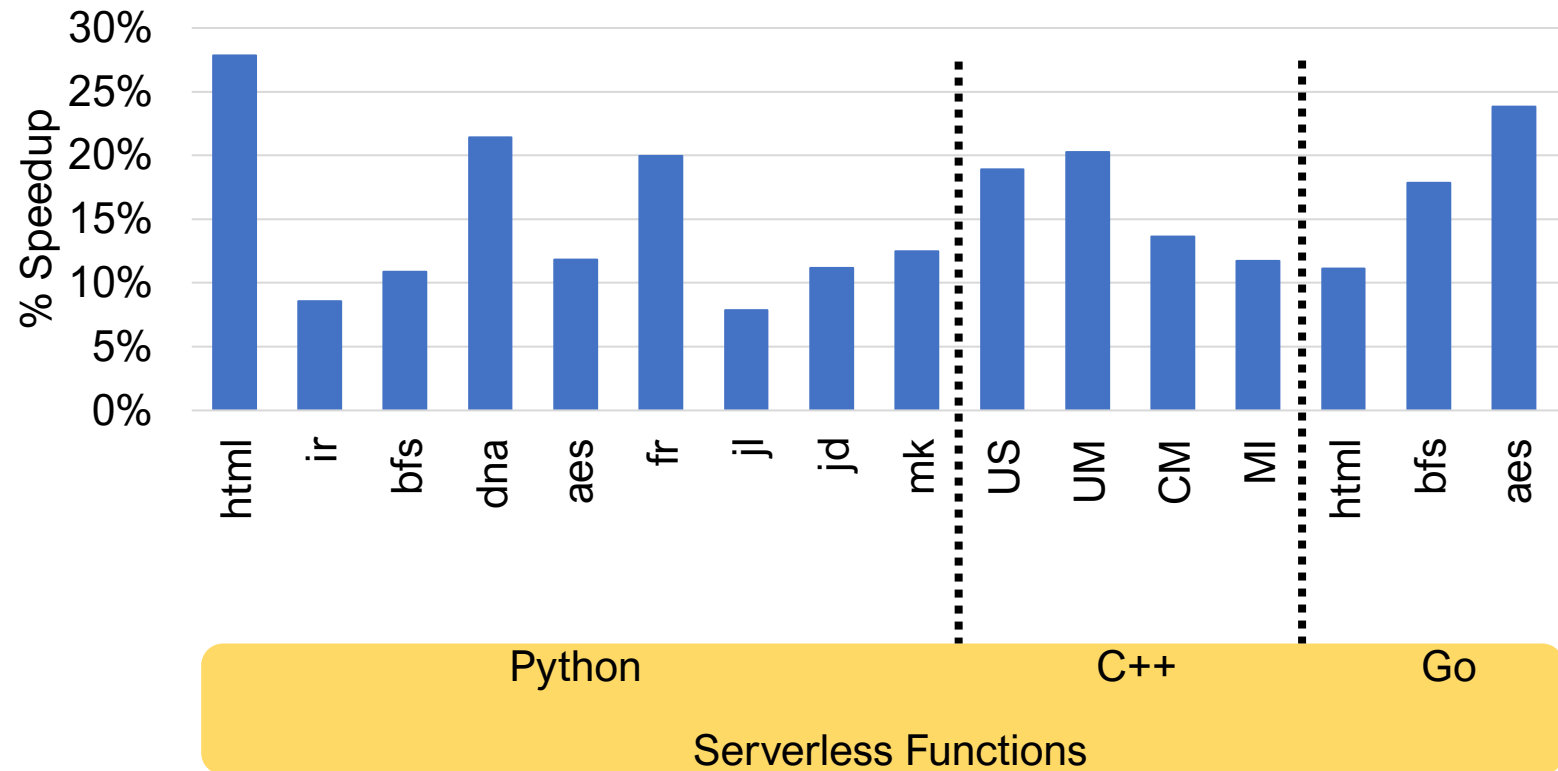


# Evaluation Methodology

- Full-system cycle-accurate simulation
- Linux kernel 5.18
- Serverless functions executed in containers  
  - 9 Python functions from SeBS, FunctionBench and pyperformance
  - 4 C++ functions from DeathStarBench
  - 3 Go functions for encryption, html generation and graph processing
- Key-value stores and in-memory databases    SQLite
  - Redis and Memcached. Silo and SQLite3.
- OpenFaaS platform that deploys and invokes functions   
OPENFAAS

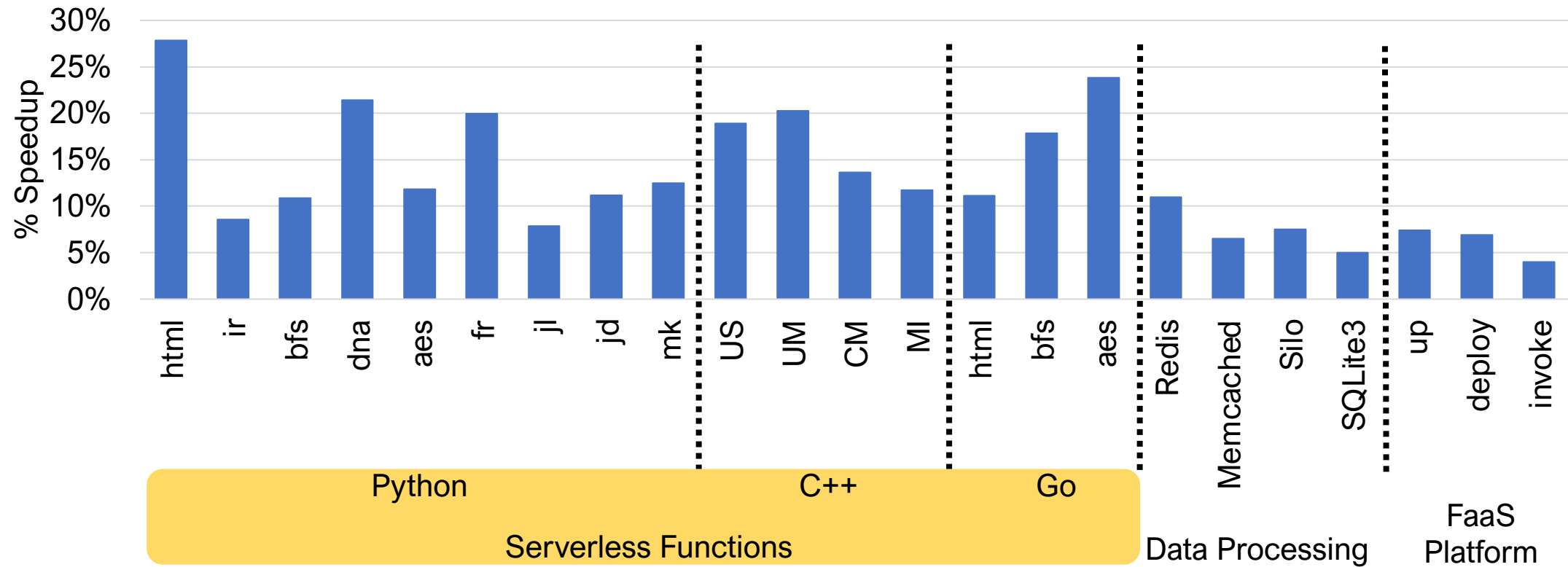


# Speedup



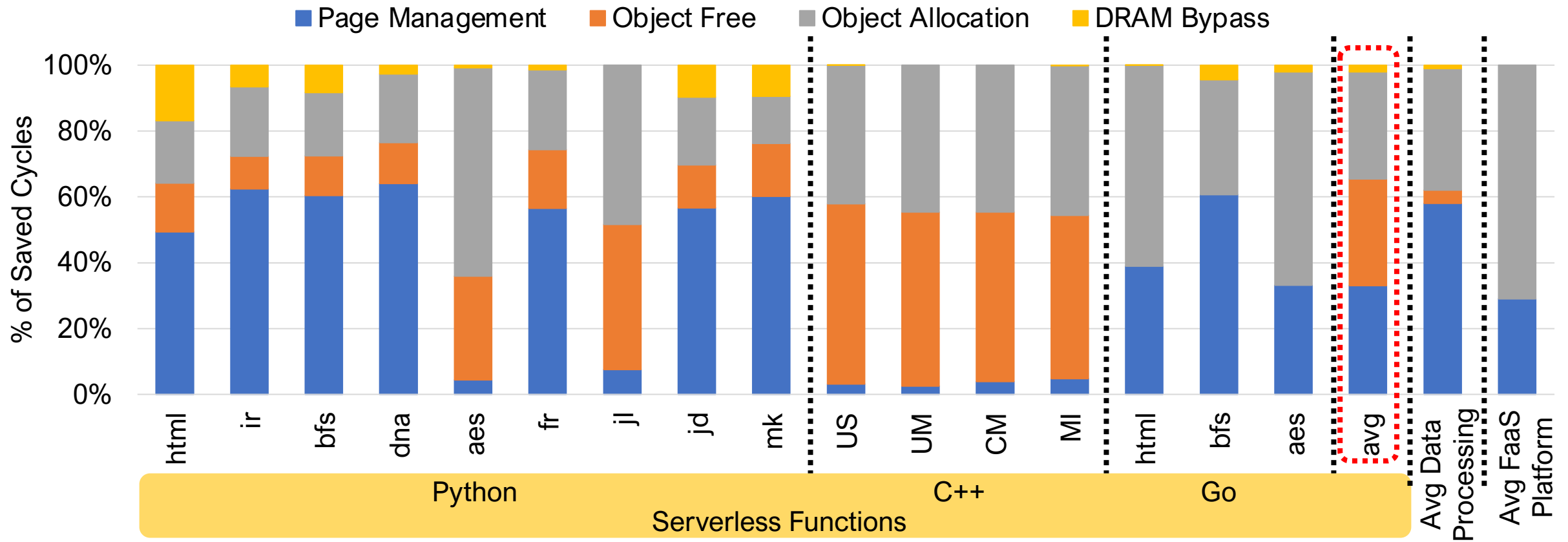
8-28% speedup for serverless functions

# Speedup



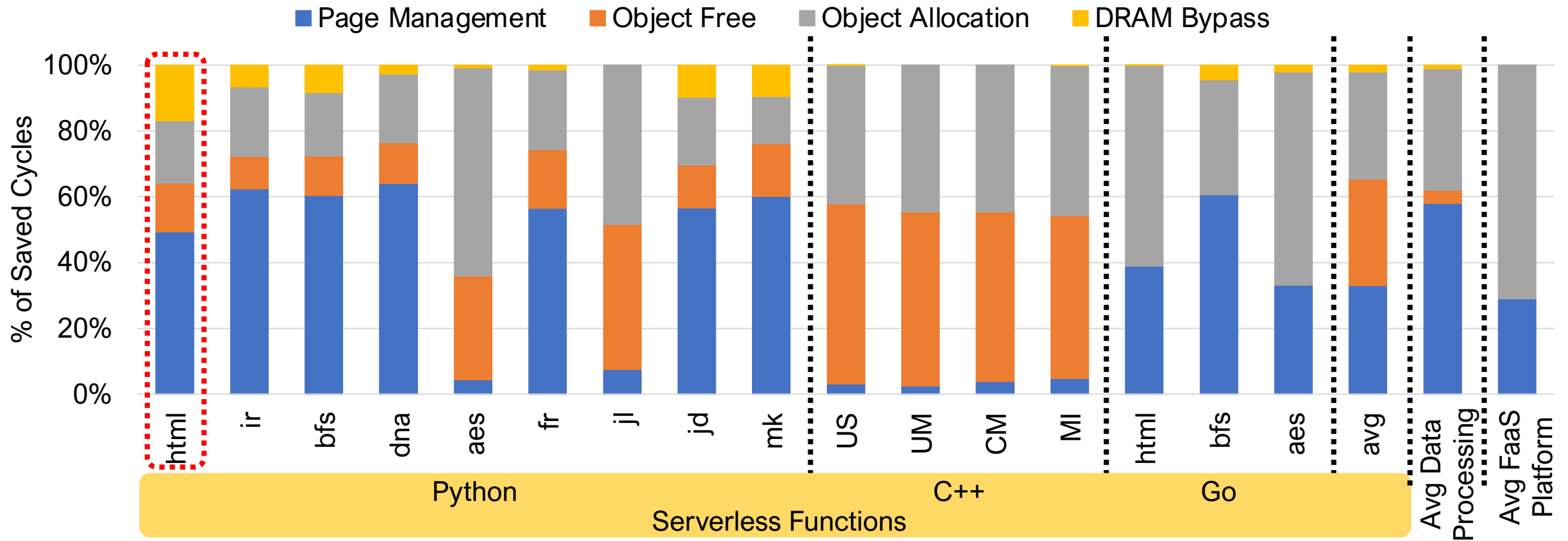
8-28% speedup for serverless functions  
4-11% speedup for Data Processing and FaaS platform workloads

# Speedup Breakdown



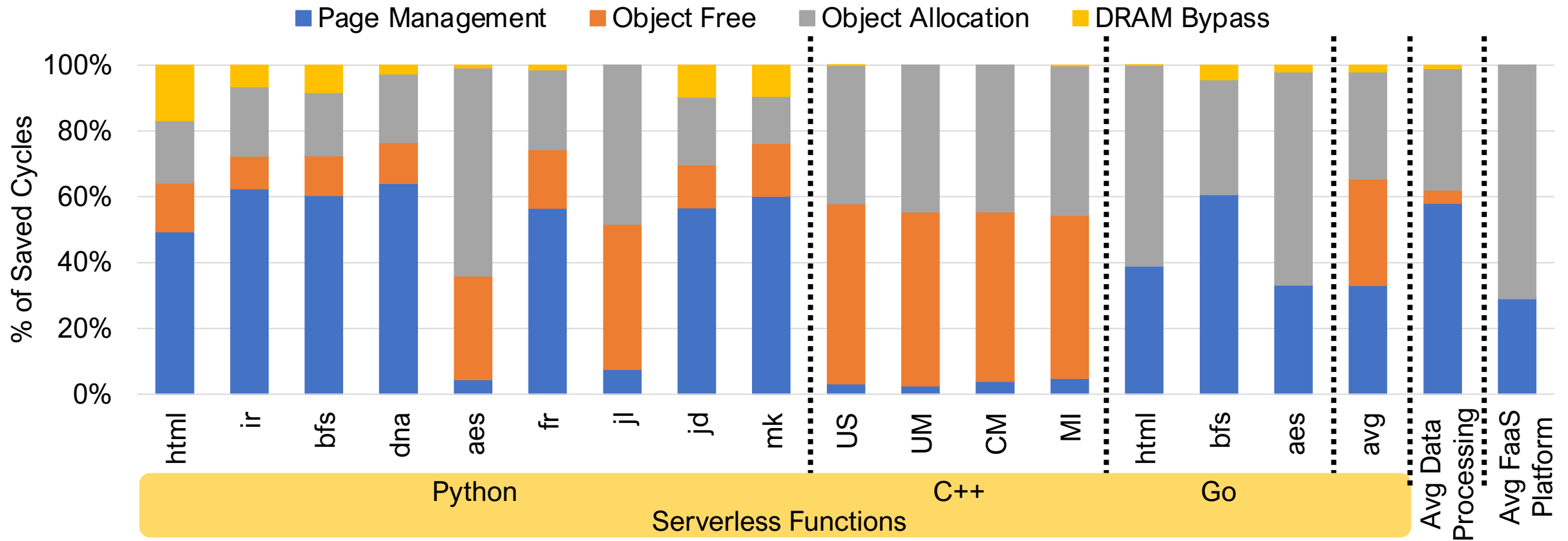
On average equal gains come from Page Management, Obj Alloc, and Obj Free

# Speedup Breakdown



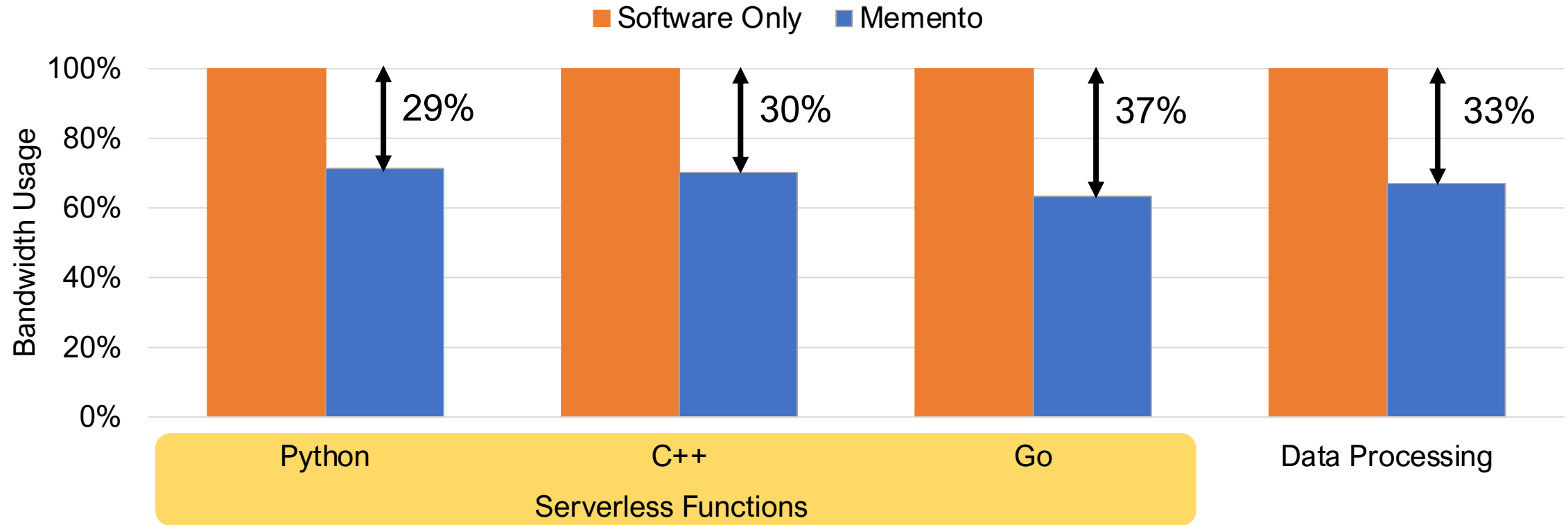
On average equal gains come from Page Management, Obj Alloc, and Obj Free  
Up to 17% of the gains come from Main Memory Bypass

# Speedup Breakdown



On average equal gains come from Page Management, Obj Alloc, and Obj Free  
 Up to 17% of the gains come from Main Memory Bypass  
 All components of Memento are crucial for performance

# Normalized Memory Bandwidth Usage



Reduces memory bandwidth usage by 30% on average

# More in the Paper

- Multicore support
- Software effects
  - Prepopulating pages on OS paths
  - Tuning software allocators
  - Fragmentation
  - Container Warmup vs Cold-start
- Detailed hardware characterization
  - HOT hit rates
  - Arena list operation frequency
  - Iso-storage and related work
  - Area and Power

# Takeaway: Memento

Datacenter tax is a long-standing challenge  
Memory management is major component

**Memento** introduces holistic arch extensions

Eliminate both **userspace** and **kernel** costs

Expose allocation semantics w/ main memory bypass

8%-28% speedup of function execution

4%-11% speedup of long-running workloads

23% reduction of memory usage

30% reduction of memory bandwidth needs

29% reduction in FaaS costs

