Computer Architecture

Performance Analysis

Debadatta Mishra, CSE, IITK

Recap (Abstraction)

- Abstraction is good …
  - Makes our life and interactions simple, and productive
  - Allows us not to be bogged down with underlying details
  - Plays a vital role in education and knowledge

Recap (Abstraction)

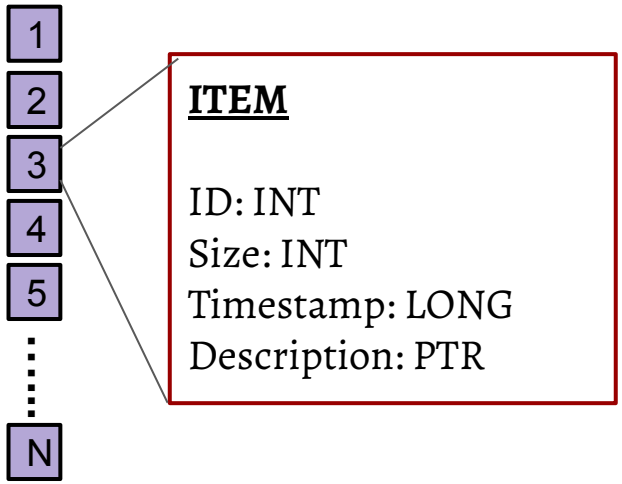
- Abstraction is good …
  - Makes our life and interactions simple, and productive
  - Allows us not to be bogged down with underlying details
  - Plays a vital role in education and knowledge
- However, understanding of the underlying details can open new doors to
  - Better understand the details (to learn how abstractions are built)
  - Improve the abstraction
  - Use the abstraction better

Recap (Abstraction)

- Abstraction is good …
  - Makes our life and interactions simple, and productive
  - Allows us not to be bogged down with underlying details
  - Plays a vital role in education and knowledge
- However, understanding of the underlying details can open new doors to
  - Better understand the details (to learn how abstractions are built)
  - Improve the abstraction
  - Use the abstraction better

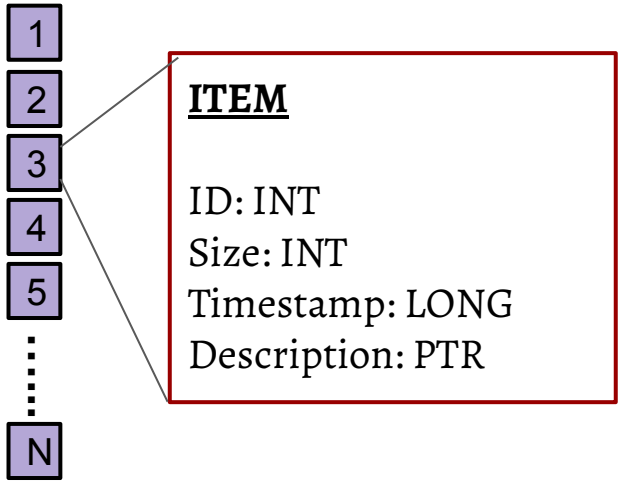
Agenda: Aspects of performance and analysis approaches

Structure of the Demo program




```
do_create(ITEM *I,  bool is_mmap, int size )
{
  if (is_mmap)
        I → Description = mmap(4096 );
  else
        I → Description = malloc(size);
}
```

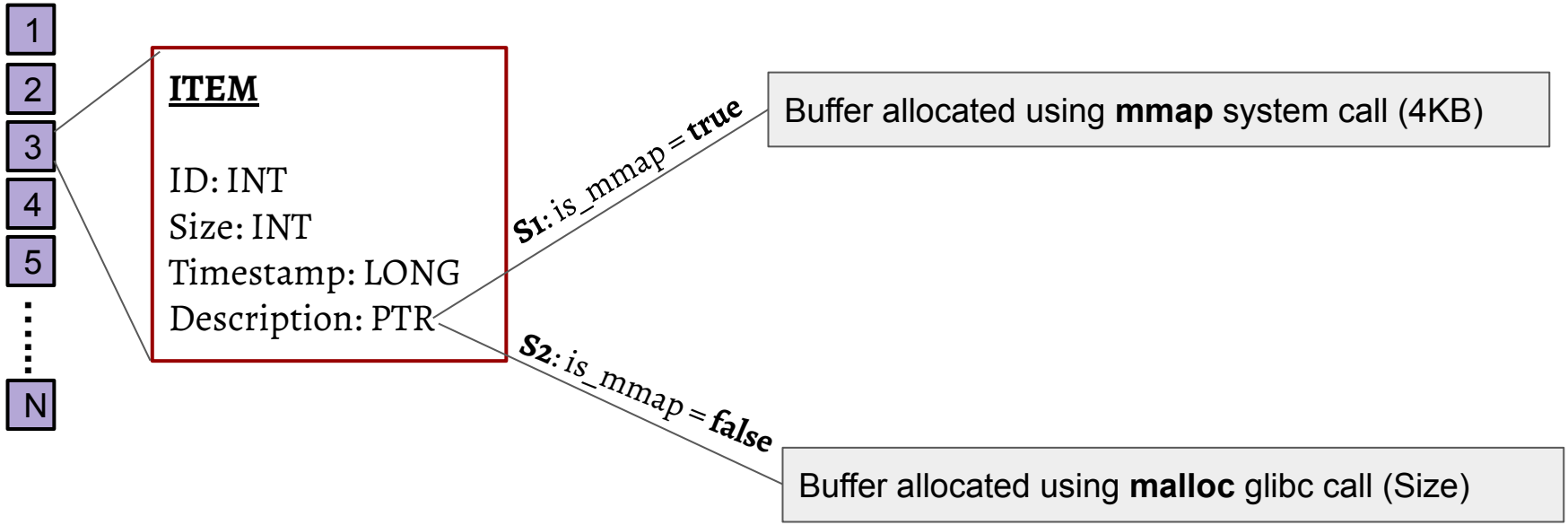
Structure of the Demo program




```
do_create(ITEM *I,  bool is_mmap, int size )
{
  if (is_mmap)
        I → Description = mmap(4096 );
  else
        I → Description = malloc(size);
}
```

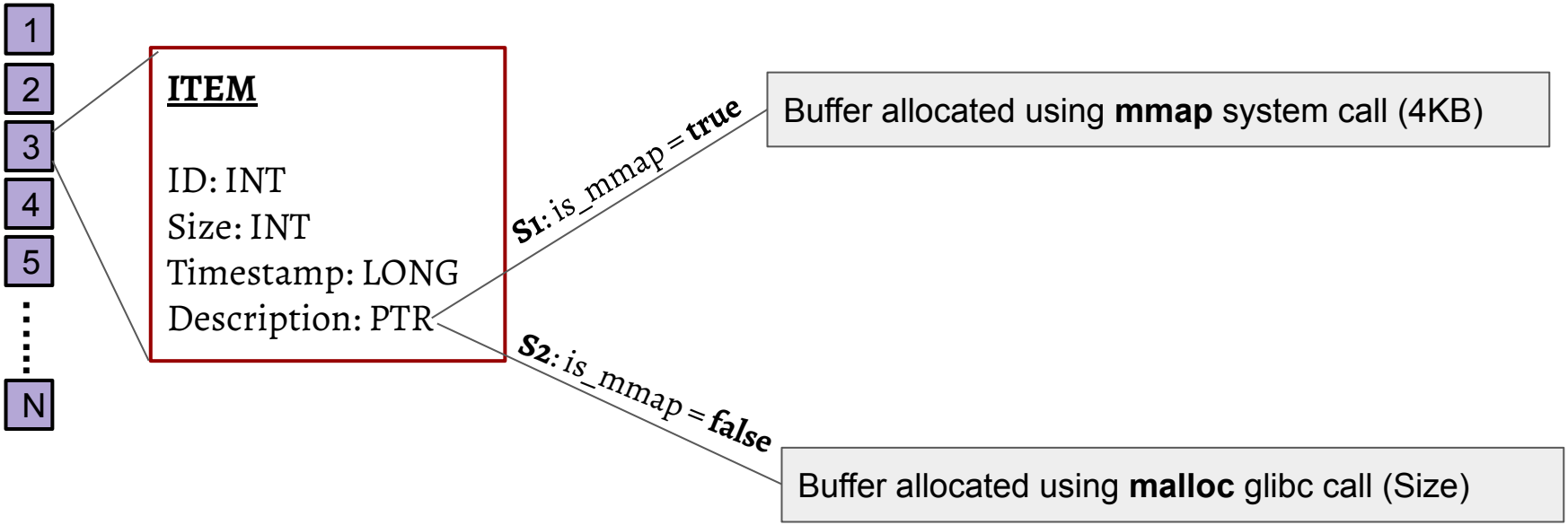
```
search_add( )
{
   ITEM *I = pickRandomItem( );
   Sum += I → Description [0];
}
```

Creation of the list of items



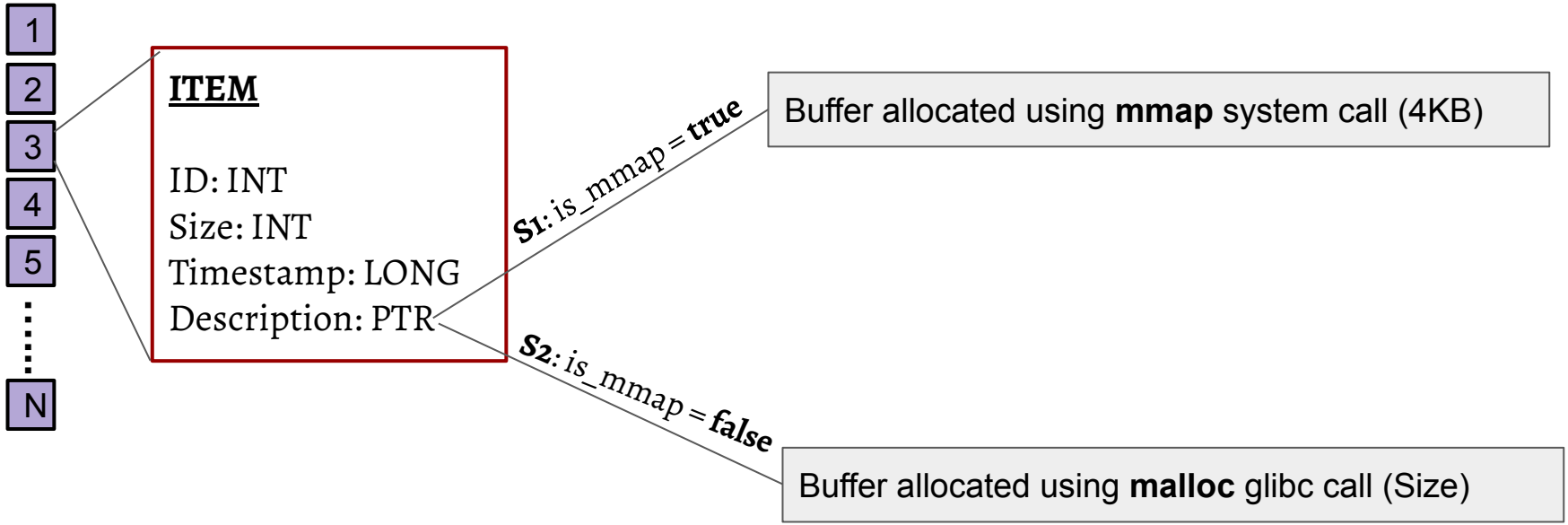
- Observation: To create a list of 1M items, S2 consumes 0.54 sec while S1 consumes 1.5 sec. Why?
- Even before answering, how do we know if the observation is correct?

Creation of the list of items



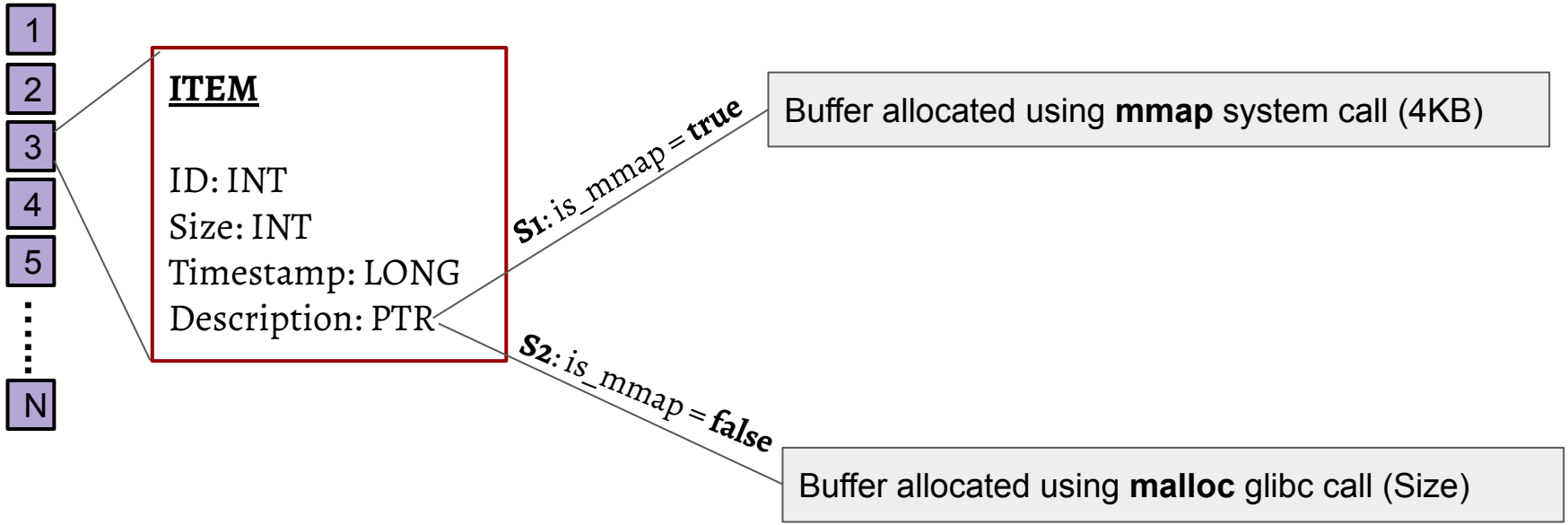
- Observation: To create a list of 1M items, S2 consumes 0.54 sec (avg.) while S1 consumes 1.5 sec (avg). Why?

Creation of the list of items



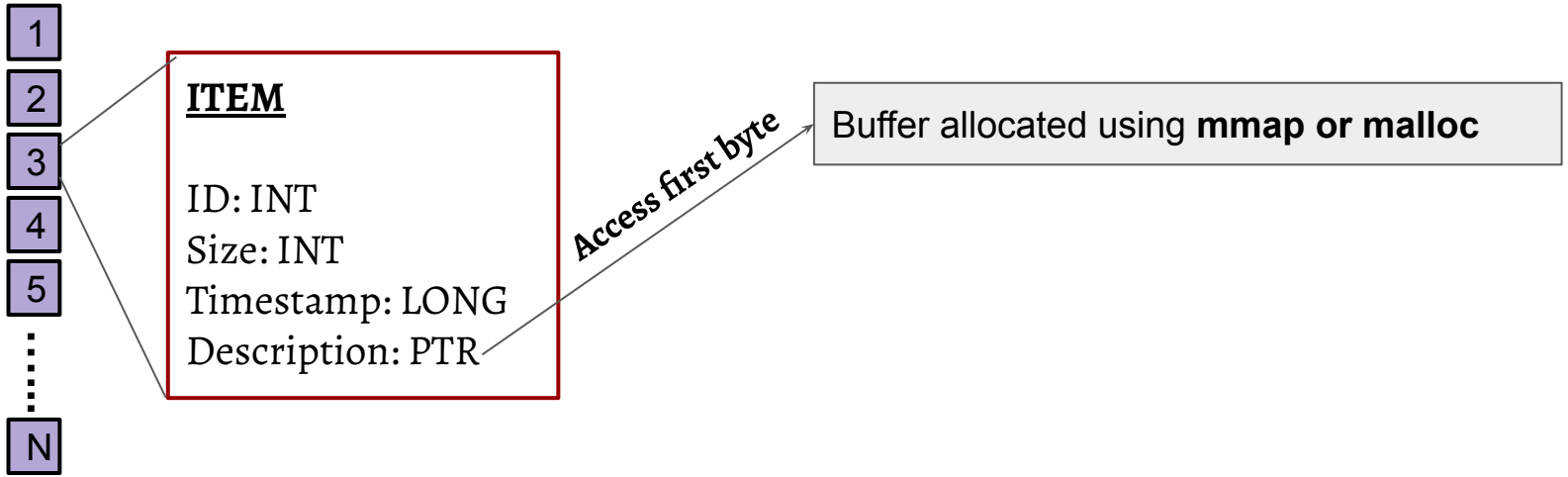
- Observation: To create a list of 1M items, S2 consumes 0.54 sec (avg.) while S1 consumes 1.5 sec (avg). Why?
- More system call invocations in case of S1, **malloc** benefits due to coalescing

Creation of the list of items (Homework)



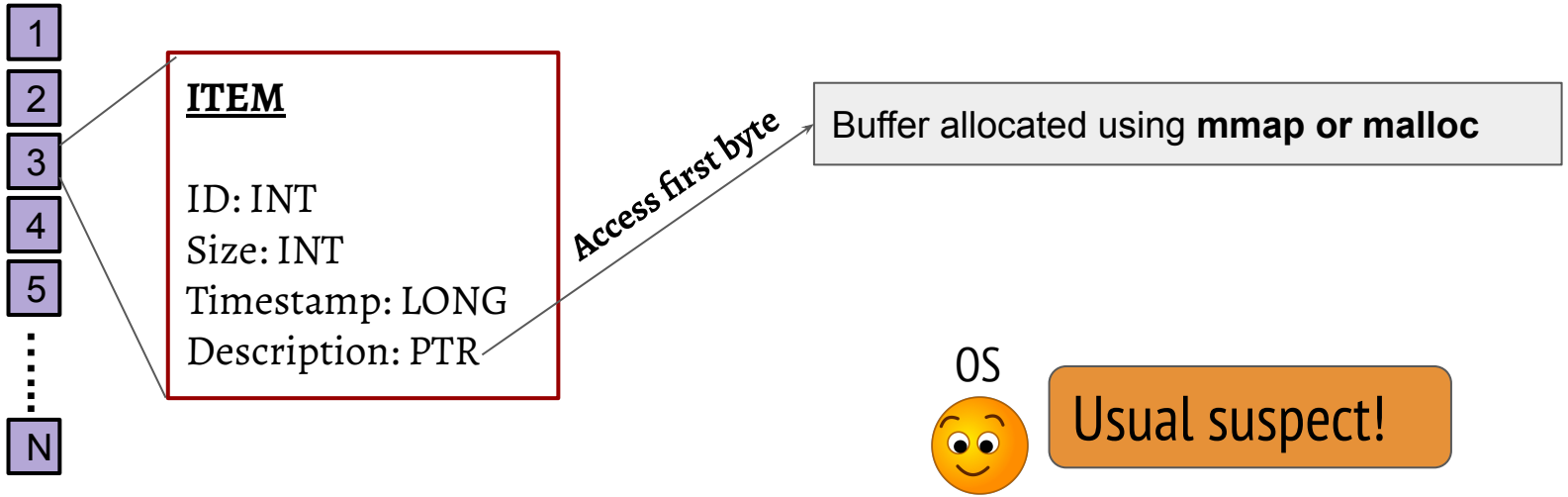
- Execute the program on your machines to reproduce the behavior
- Compare the time taken for all invocations of **malloc** and **mmap**
- Does your observation match the conclusions?

Accessing random elements of the list



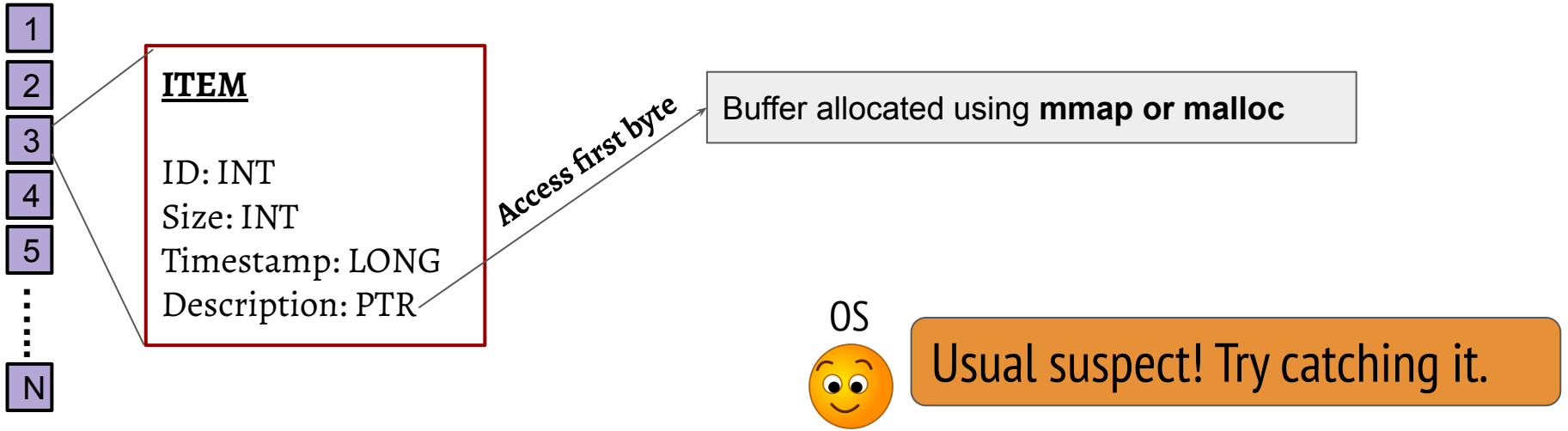
- Observations: For 200K random accesses to the list, S2 consumes 19.3 ms (avg) while S1 consumes 27.6 ms (avg). Why?

Accessing random elements of the list



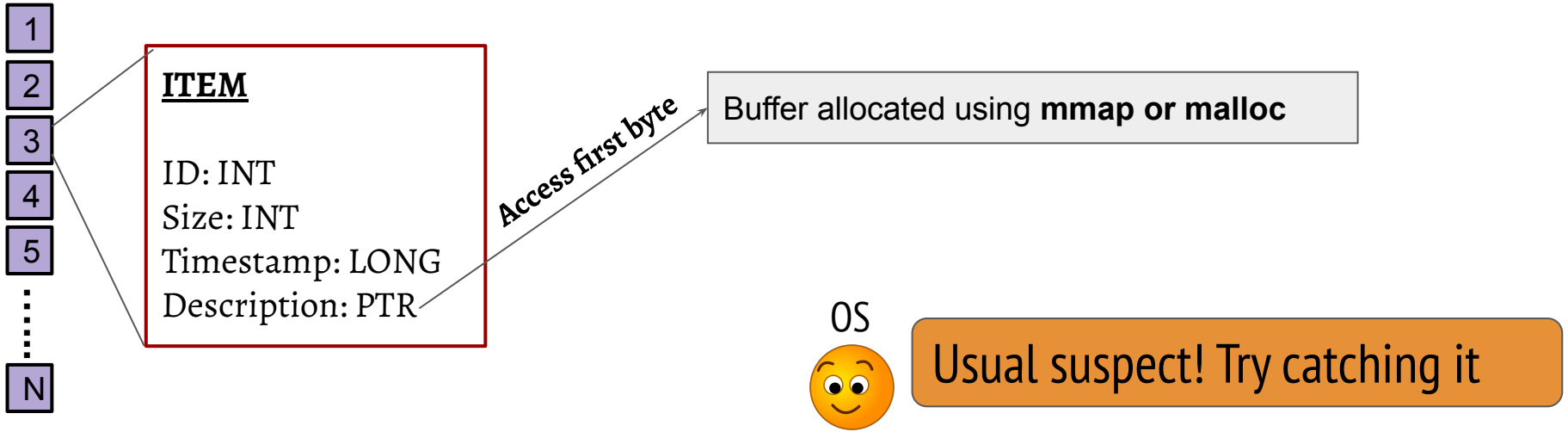
- Observations: For 200K random accesses to the list, S2 consumes 19.3 ms (avg) while S1 consumes 27.6 ms (avg). Why?

Is OS responsible?



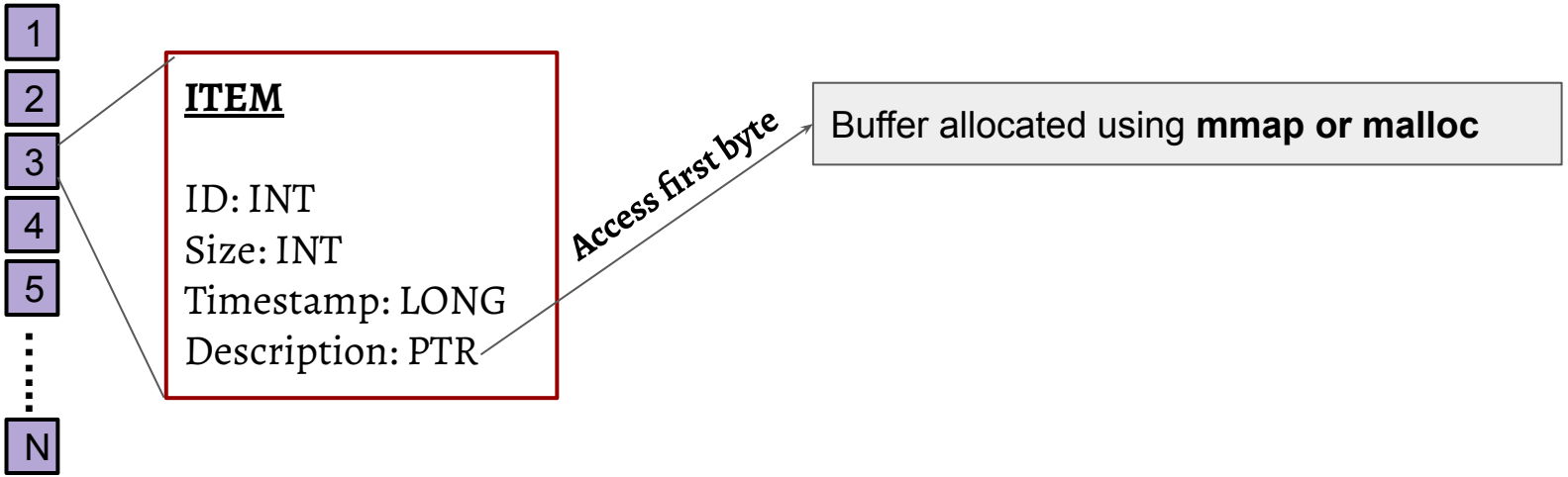
- Probe directions
  - Paging and page faults play some role (more likely)
  - Scheduling plays a role (less likely)
  - Anything else …

Is OS responsible?



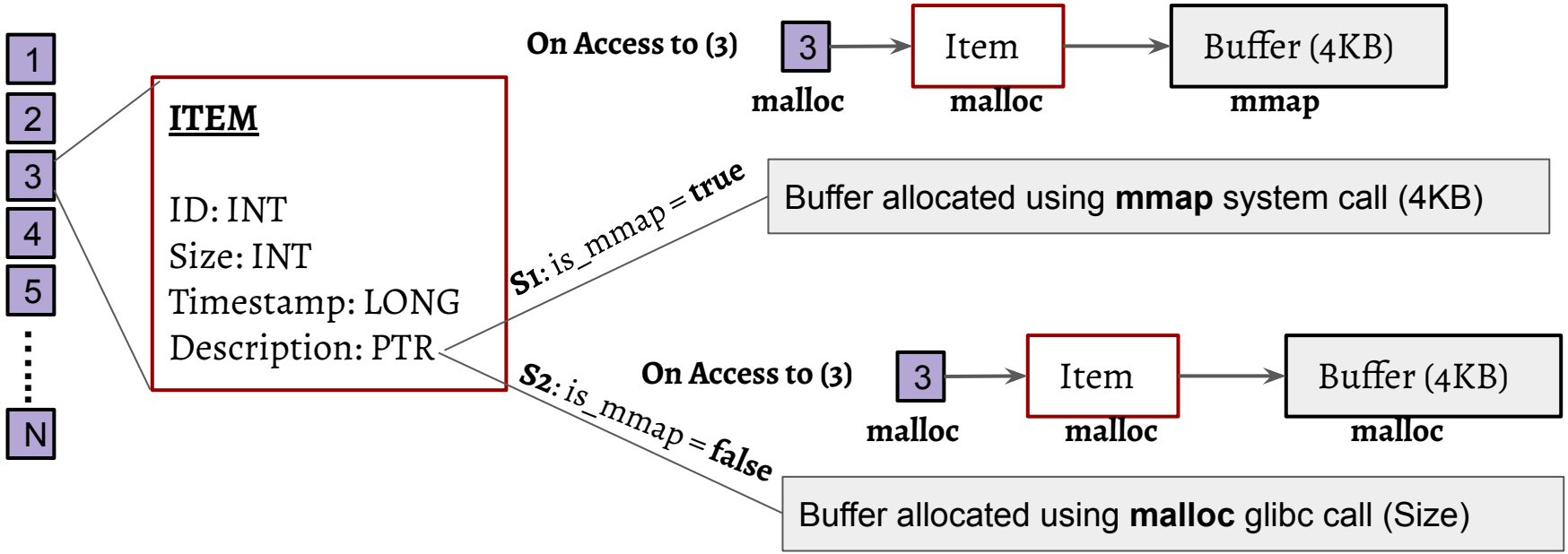
- Probe outcome
  - No page faults observed, OS memory management not guilty!
  - Plenty of CPUs in the machine, OS scheduling not guilty!
  - Anything else …

Anything other than OS?



- The hardware is the same… that can not be the issue (or can it be?)
- The code fragment is the same, compiler is out of the equation
- Could something at the hardware layer change because of how the list is created?

Let us zoom into the creation and analyze access



- In one case all indirections are accessing memory allocated by **malloc** while in the other case the last access is to memory allocated by **mmap**
- So what?

Performance Analysis in three situations