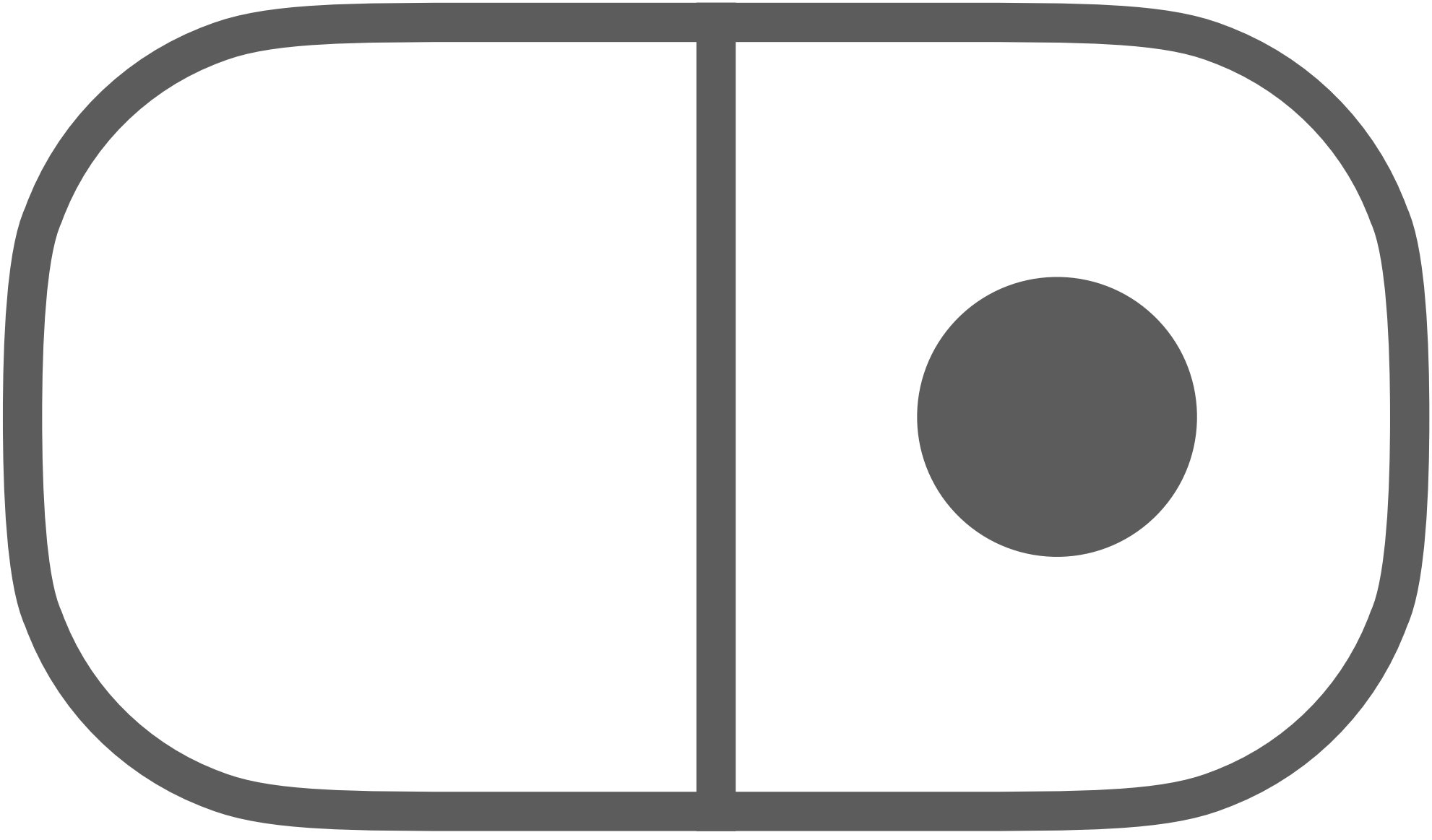
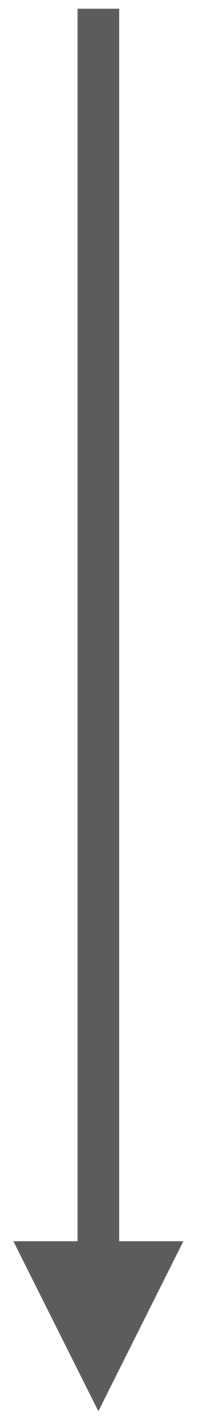
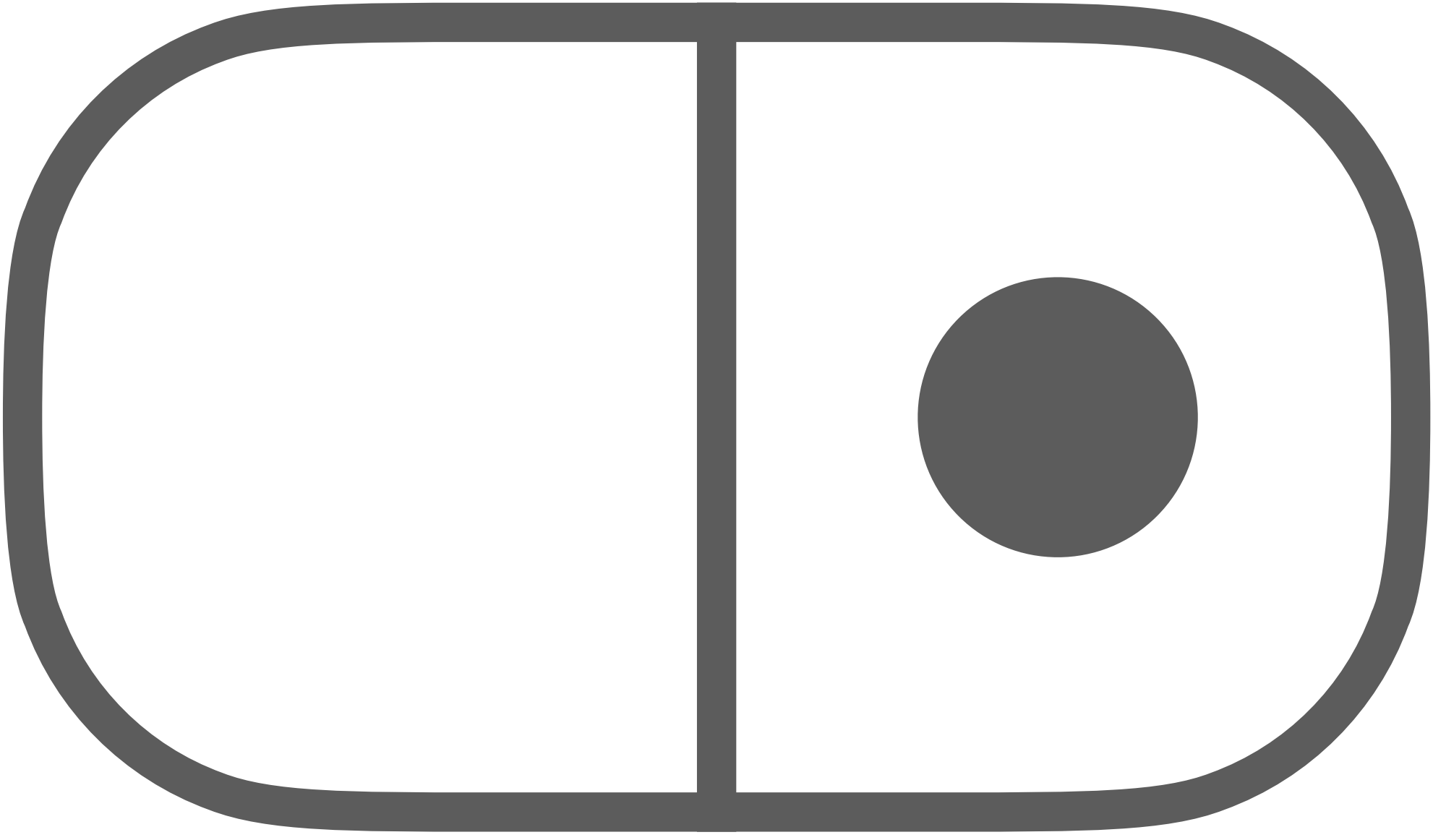
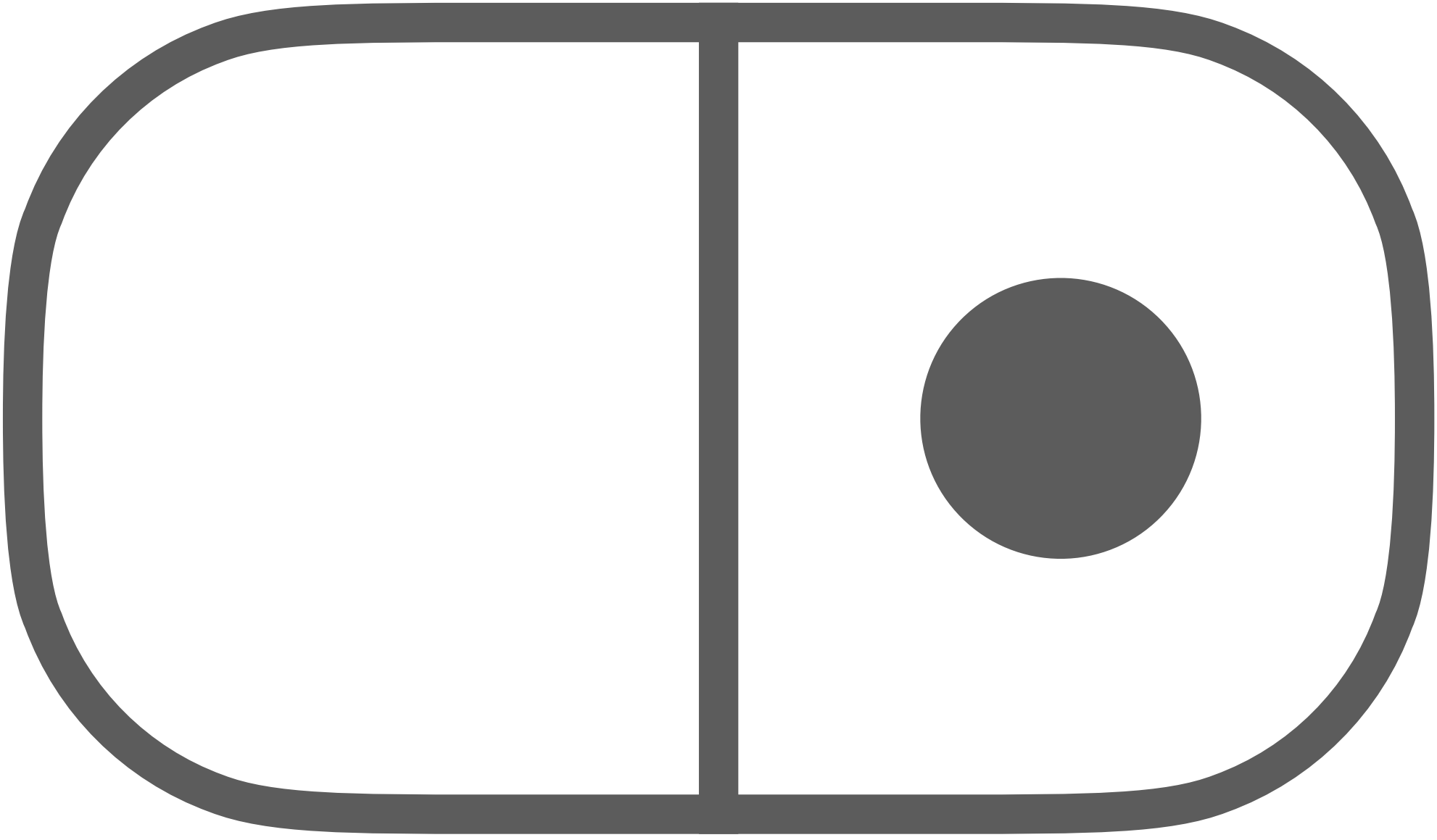
```
void dequeue() {
  while (true) {
    head = Head;
    protect(head);
    if(head != Head) continue;
    next = head->next;
    // ...
    if(CAS(Head,head,next)){
      retire(head);
      return;
}}}
```

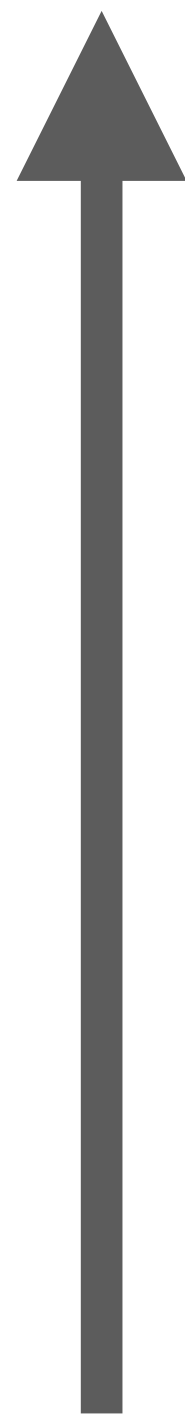
Safe Memory Reclamation (SMR)

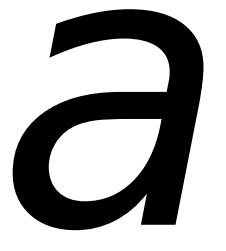


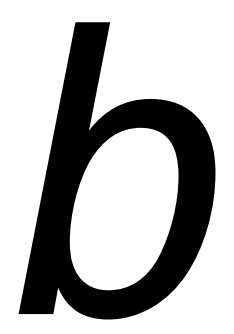


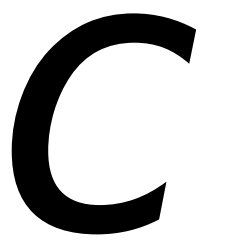













Head


head2


head1


next1

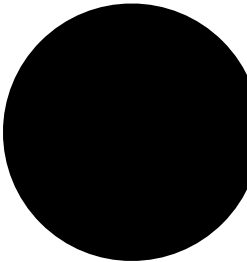

next2

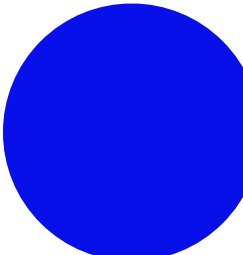

head3

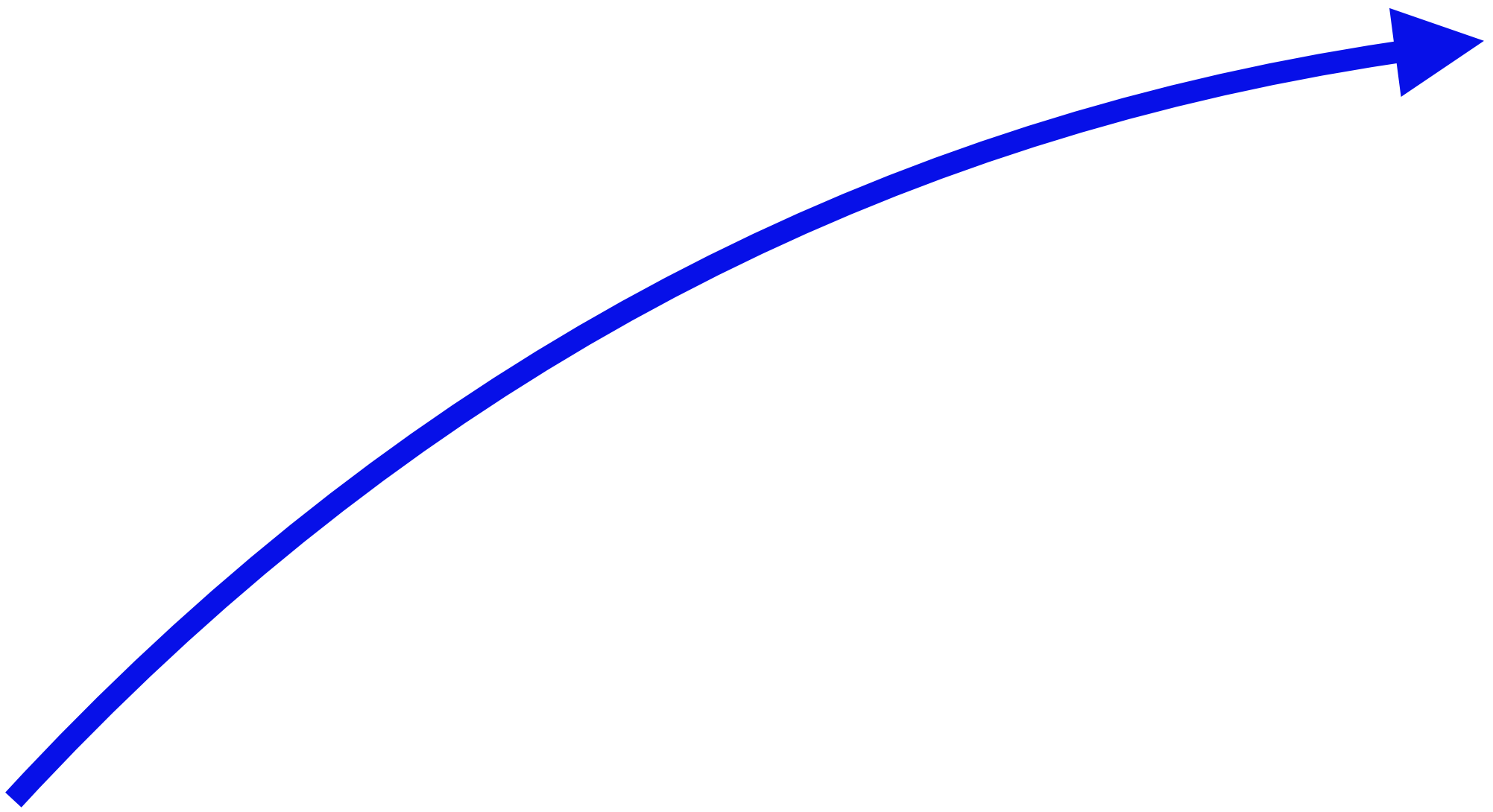

1

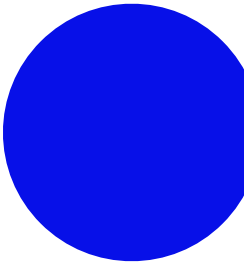

2

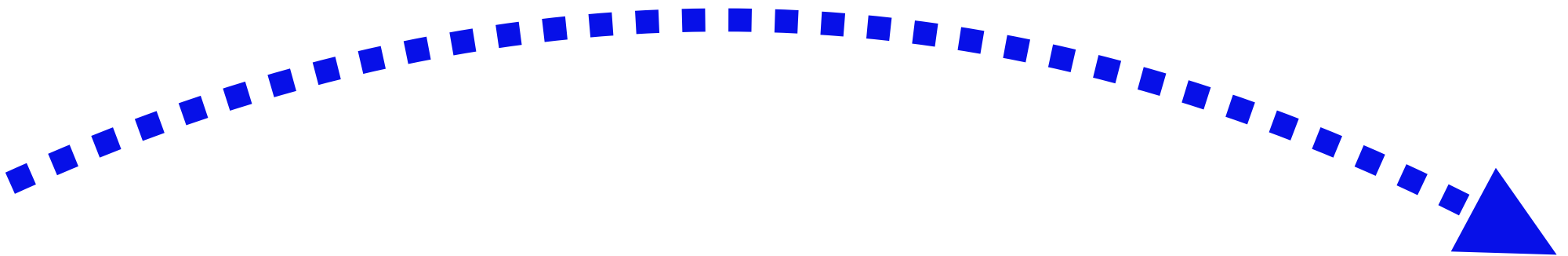

3

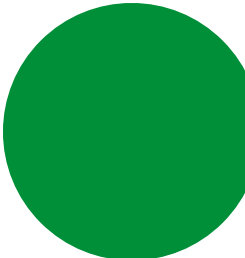


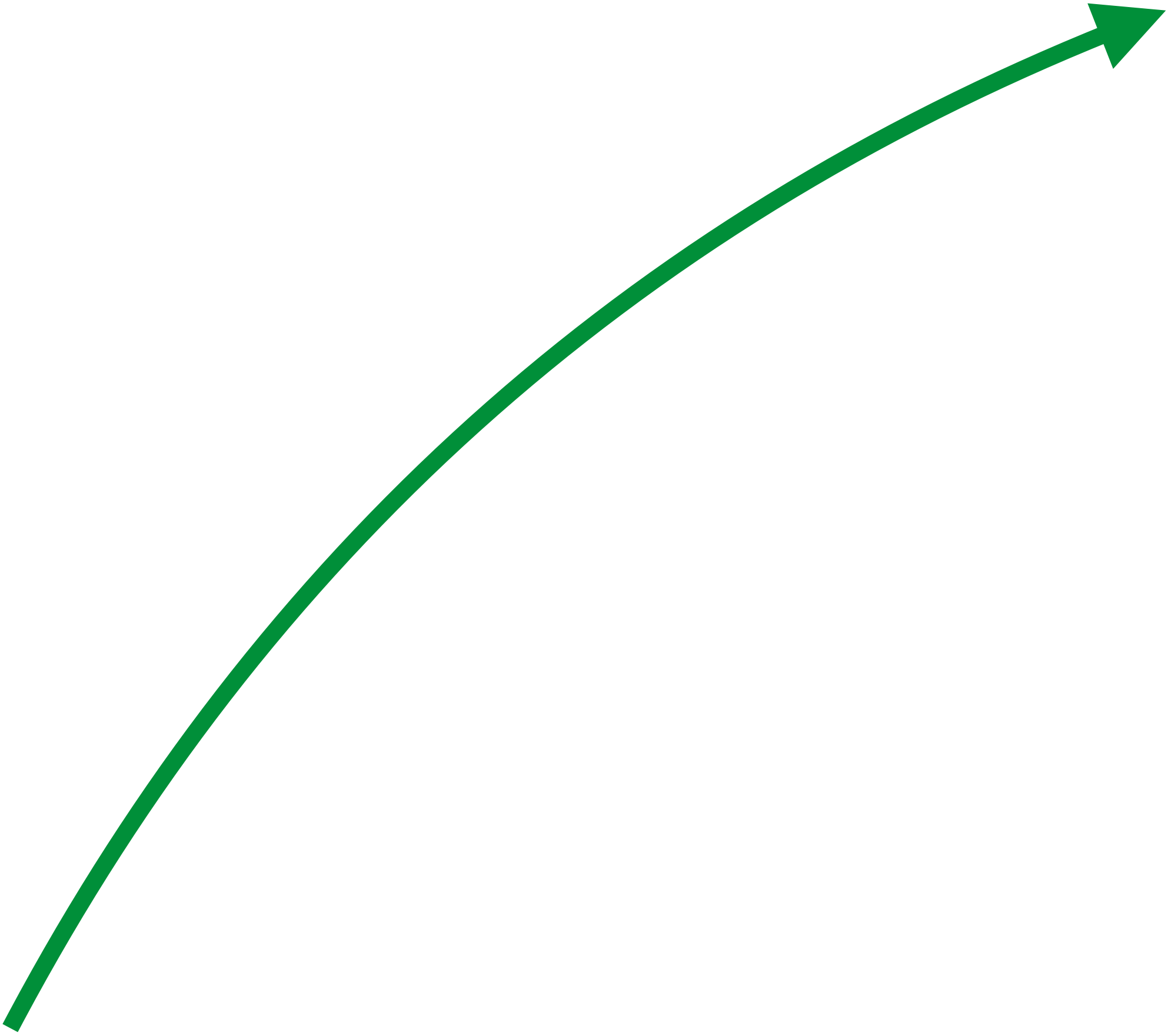


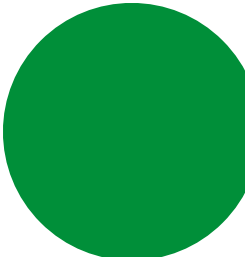


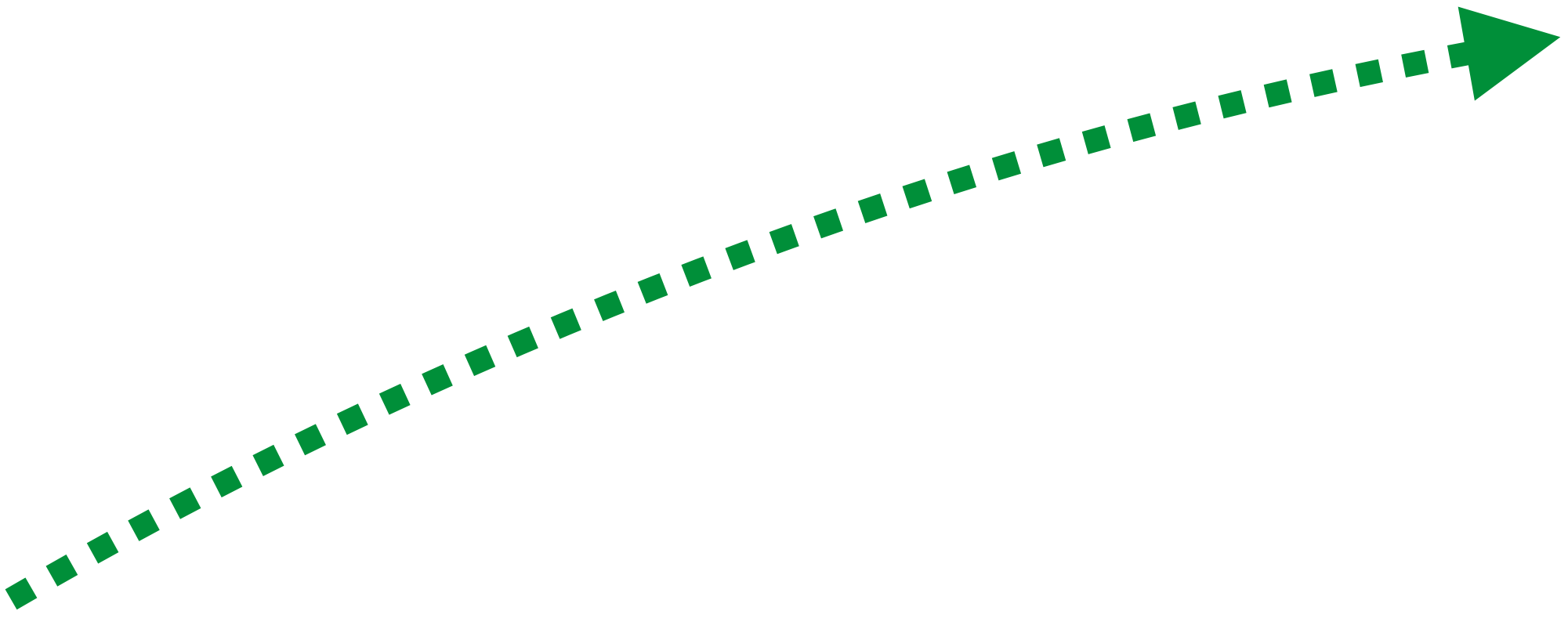


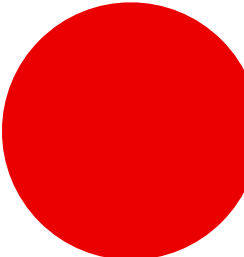


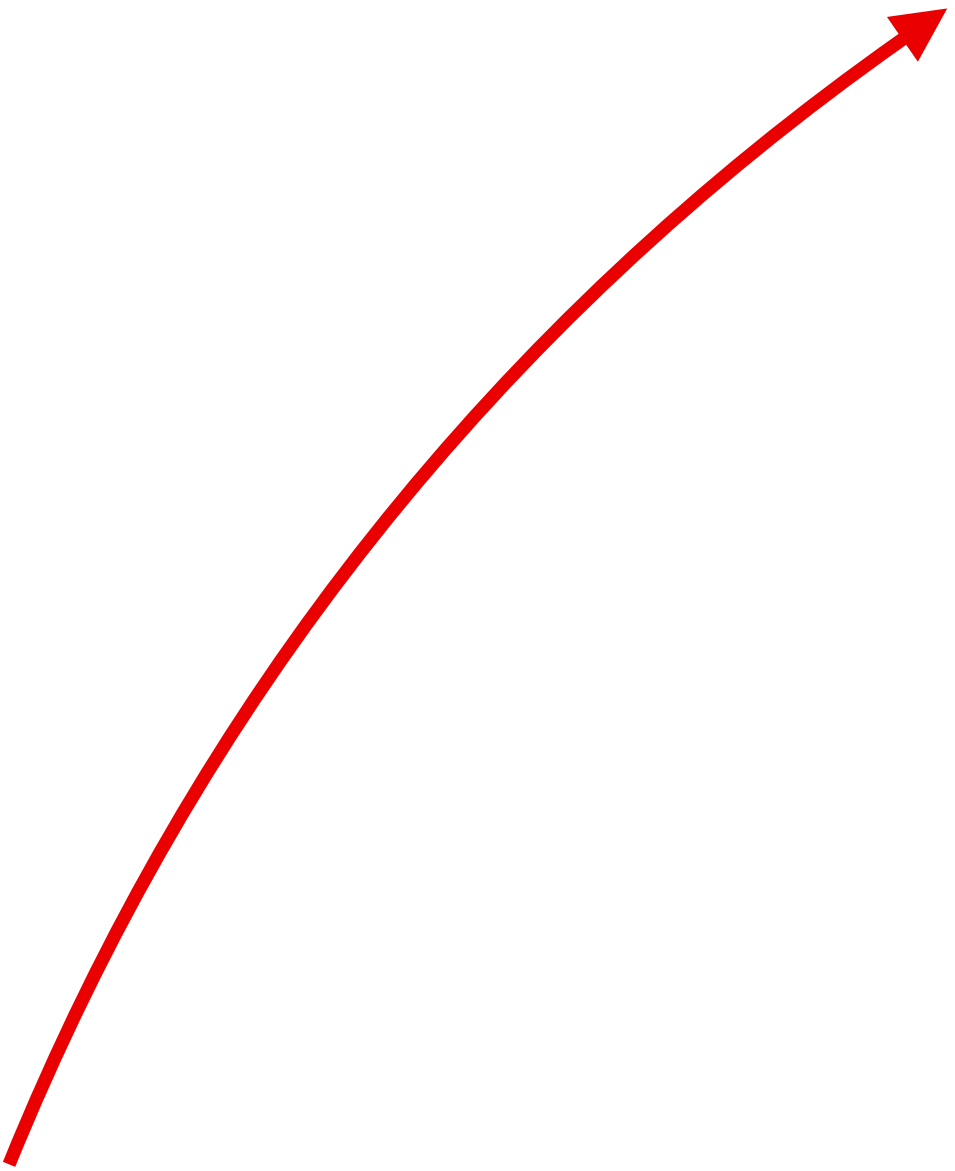


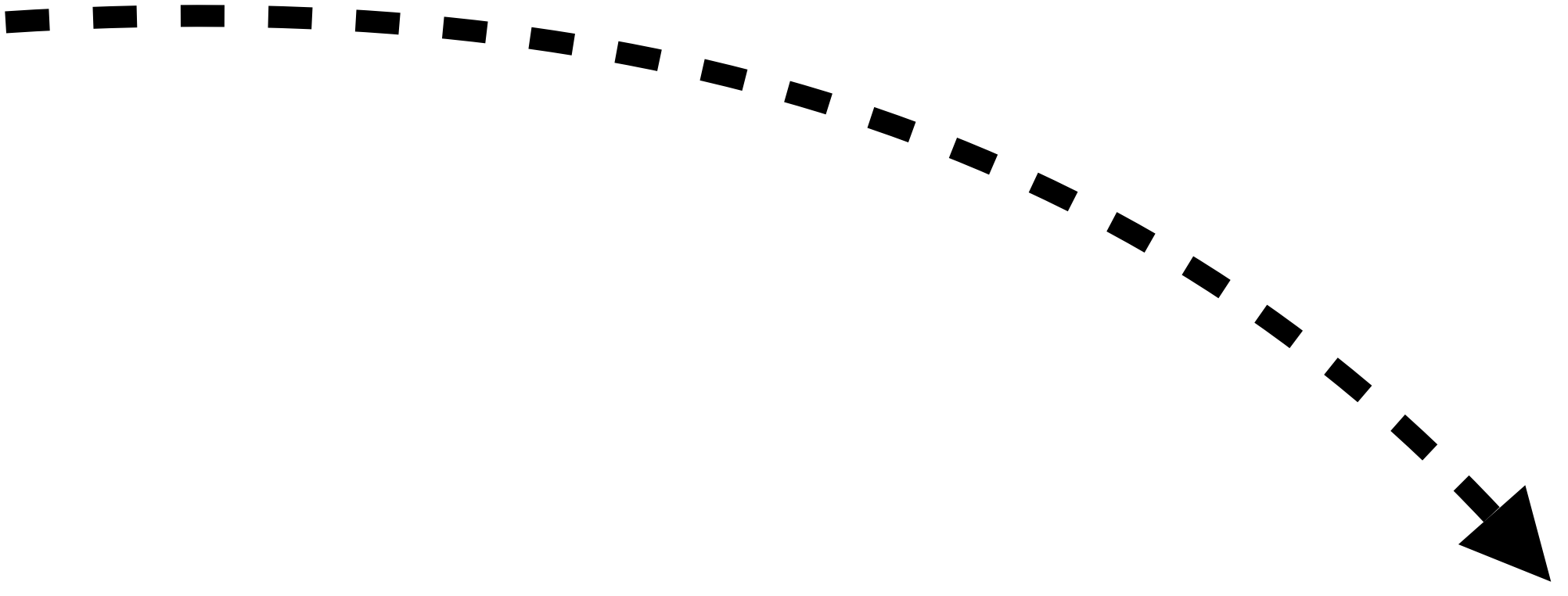


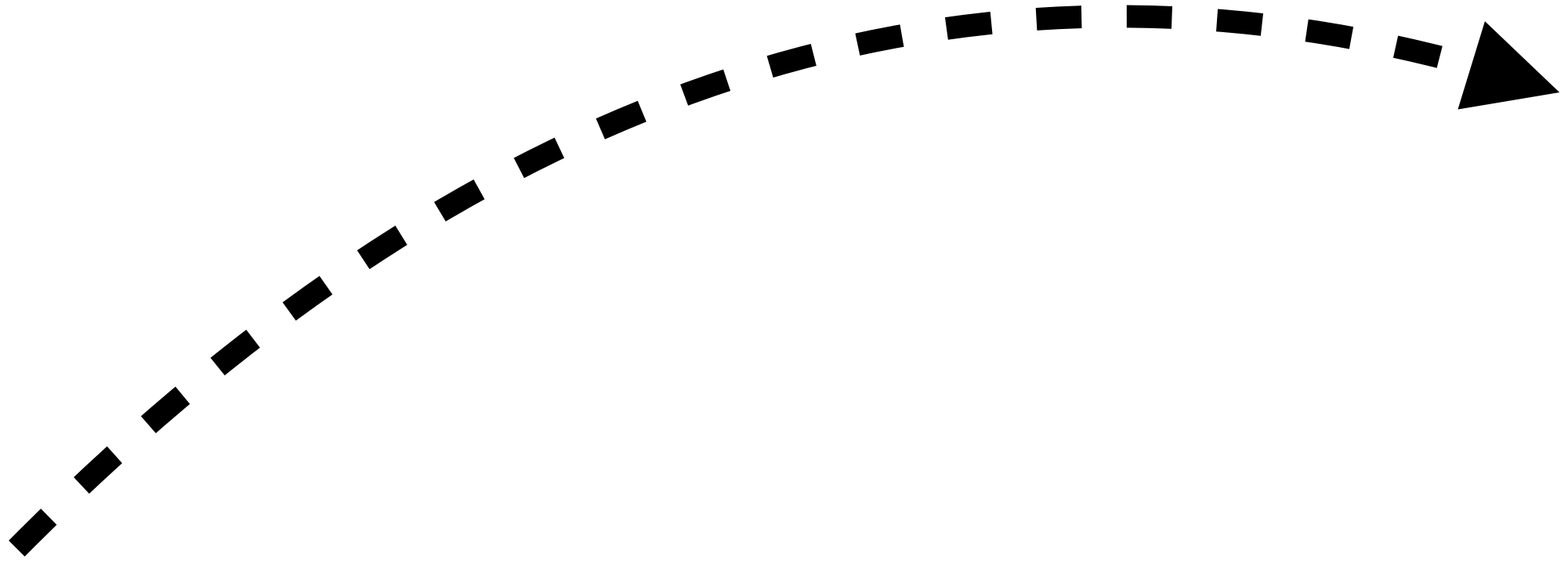


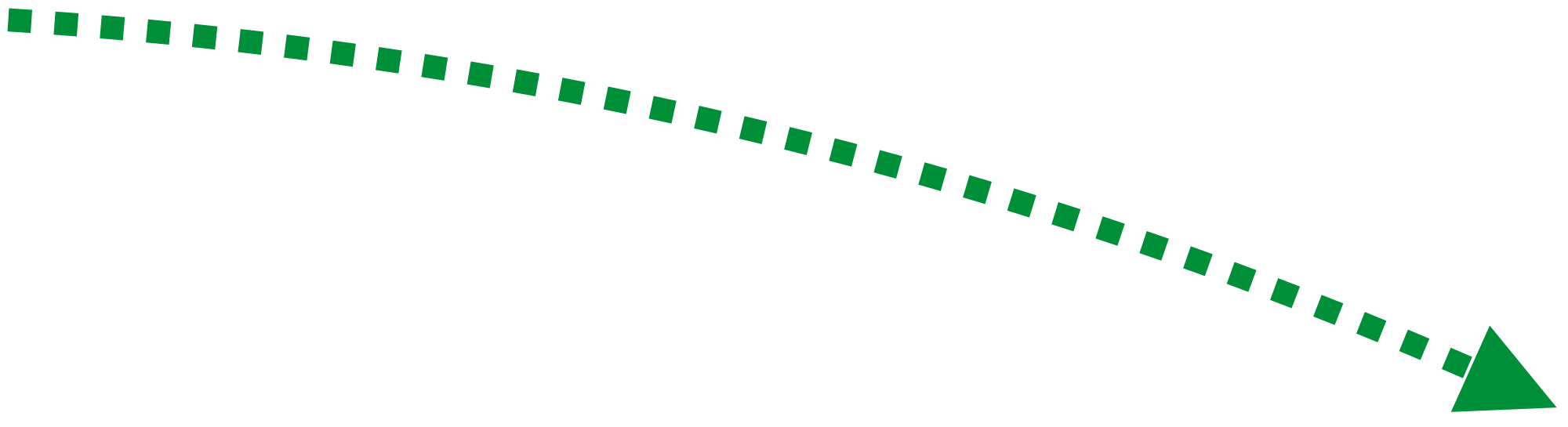




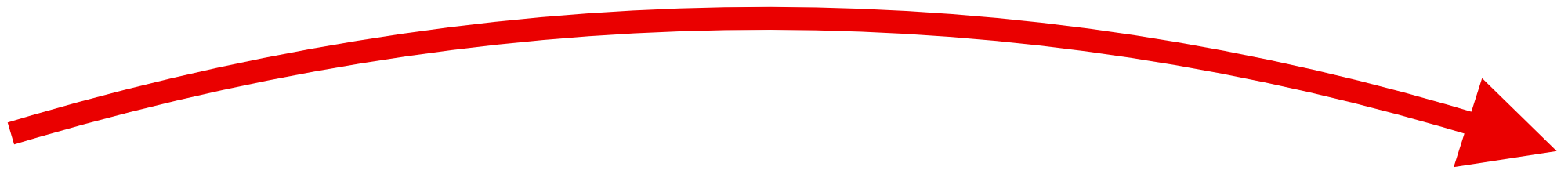









2

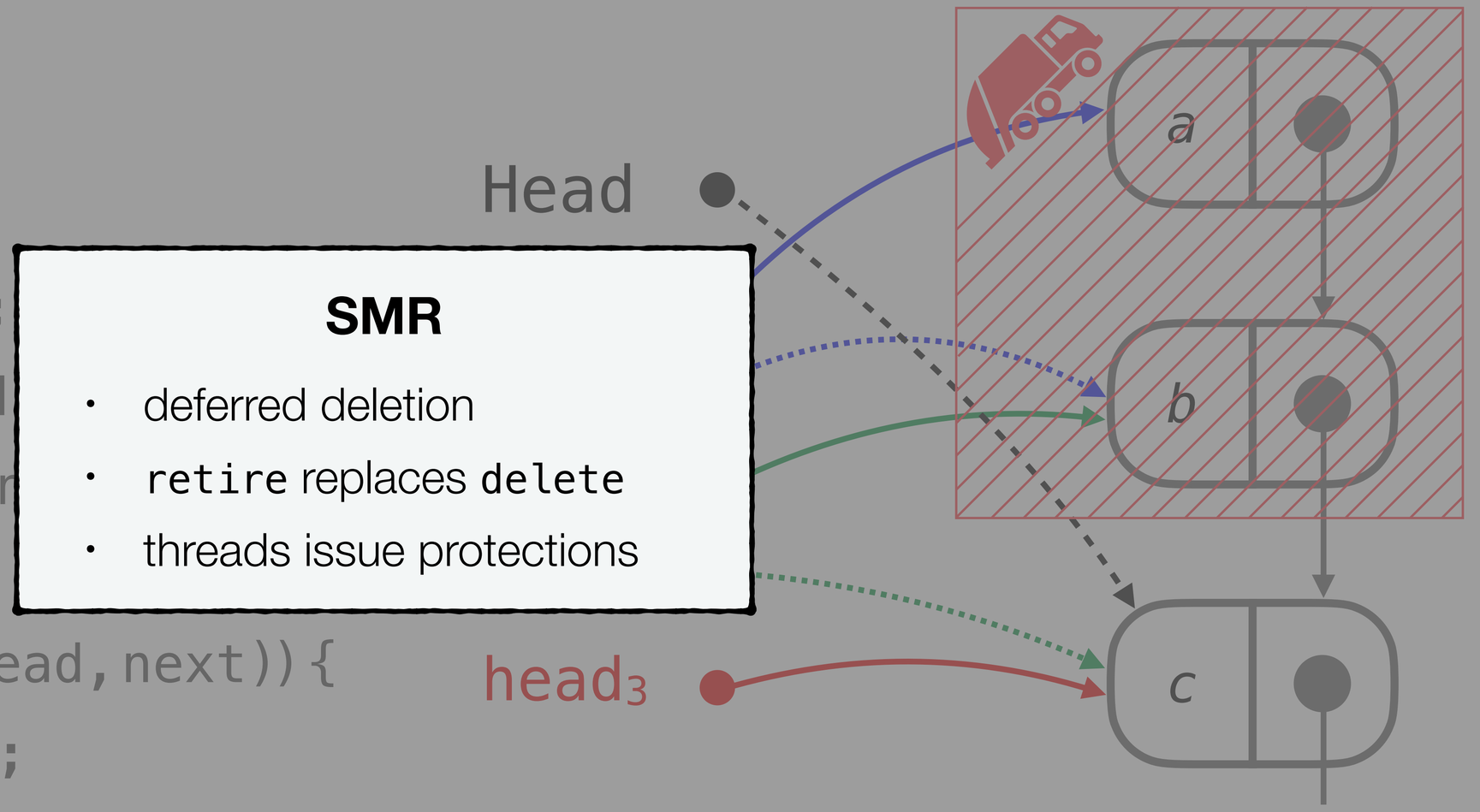


SMR

- deferred deletion
- `retire` replaces `delete`
- threads issue protections

Safe Memory Reclamation (SMR)

```
void dequeue() {
  while (true) {
    head = Head;
3   protect(head);
    if(head != Head
    next = head->n
    // ...
    if(CAS(Head,head,next)){
      retire(head);
1 2   return;
}}}
```



SMR

- deferred deletion
- `retire` replaces `delete`
- threads issue protections

Non-blocking Queue (Michael&Scott)

```
struct Node {
    data_t data;
    Node* node;
}
```

```
shared:
    Node* Head;
    Node* Tail;
```

```
void init() {
    Head = new Node();
    Head->next = null;
    Tail = Head;
}
```

```
void enqueue(data_t val) {
    Node* node = new Node();
    node->data = val;
    node->next = null;
    while (true) {
        Node* tail = Tail;


        Node* next = tail->next;
        if (Tail != tail) continue;
        if (next == null) {
            if (CAS(tail->next, null, node)) {
                CAS(Tail, tail, node);
            }
        } else {
            CAS(Tail, tail, next);
        }
    }
}
```

```
data_t dequeue() {
    while (true) {
        Node* head = Head;


        Node* tail = Tail;
        Node* next = head->next;

        if (Head != head) continue;
        if (head == tail) {
            if (next == null) return empty_t;
            else CAS(Tail, tail, next);
        } else {
            data = head->data;
            if (CAS(Head, head, next)) {

                return data;
            }
        }
    }
}
```

queue

lock-free

GC

37 LOC