Discussion Section 3

- HW1 Comments
- HW2 questions
- Maximal D-segment
- Coding practices
- Useful data structures

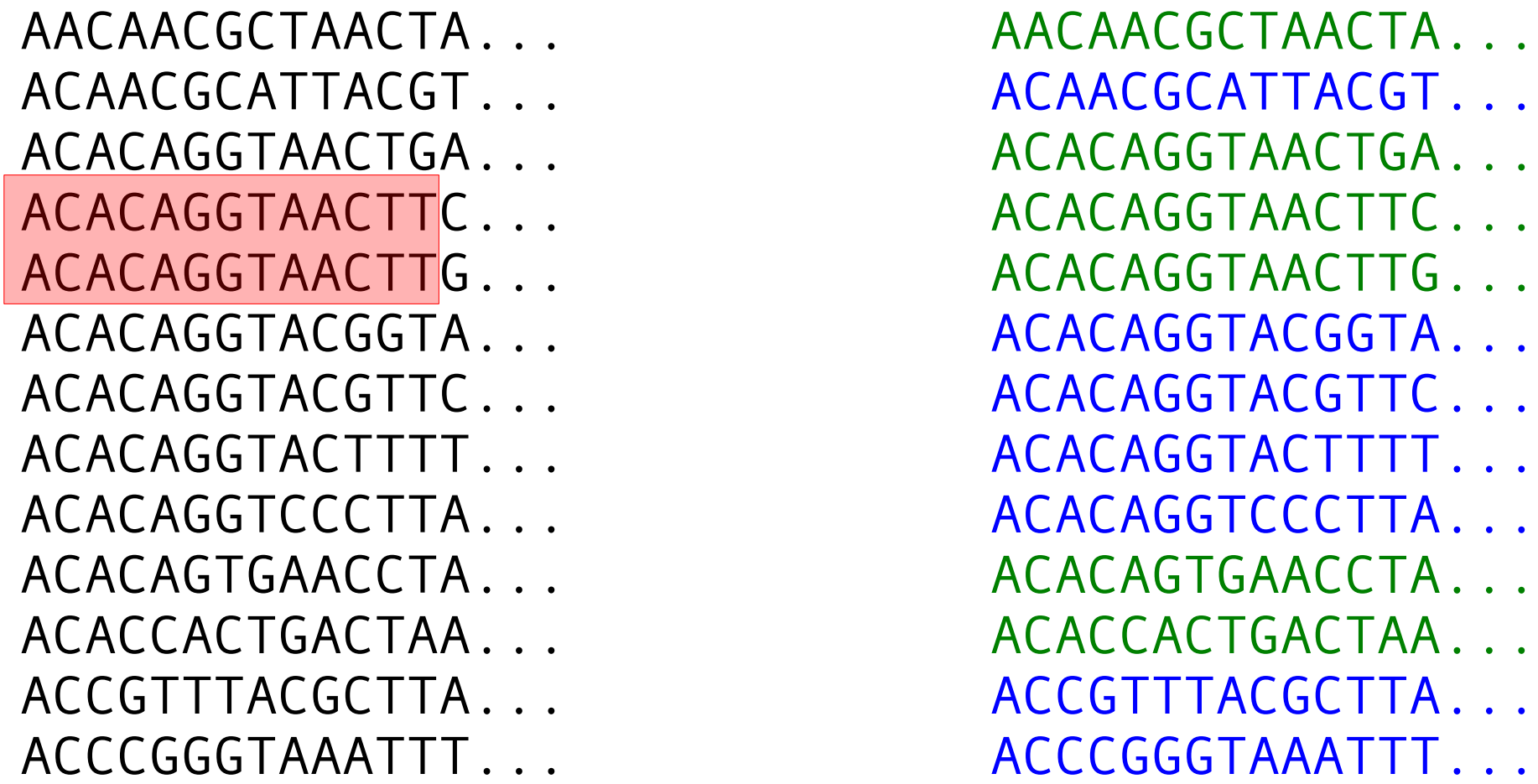
How many possible matches are there?

```
AACAACGCTAACTA...
ACAACGCATTACGT...
ACACAGGTAACTGA...
ACACAGGTAACTTC...
ACACAGGTAACTTG...
ACACAGGTACGGTA...
ACACAGGTACGTTC...
ACACAGGTACTTTT...
ACACAGGTCCCTTA...
ACACAGTGAACCTA...
ACACCACTGACTAA...
ACCGTTTACGCTTA...
ACCCGGGTAAATTT...
```

How many possible matches are there?

```
AACAACGCTAACTA...
ACAACGCATTACGT...
ACACAGGTAACTGA...
ACACAGGTAACTTC...
ACACAGGTAACTTG...
ACACAGGTACGGTA...
ACACAGGTACGTTC...
ACACAGGTACTTTT...
ACACAGGTCCCTTA...
ACACAGTGAACCTA...
ACACCACTGACTAA...
ACCGTTTACGCTTA...
ACCCGGGTAAATTT...
```

How many possible matches are there?



How many possible matches are there?

```
AACAACGCTAACTA...
ACAACGCATTACGT...
ACACAGGTAACTGA...
ACACAGGTAACTTC...
ACACAGGTAACTTG...
ACACAGGTACGGTA...
ACACAGGTACGTTC...
ACACAGGTACTTTT...
ACACAGGTCCCTTA...
ACACAGTGAACCTA...
ACACCACTGACTAA...
ACCGTTTACGCTTA...
ACCCGGGTAAATTT...
```

```
AACAACGCTAACTA...
ACAACGCATTACGT...
ACACAGGTAACTGA...
ACACAGGTAACTTC...
ACACAGGTAACTTG...
ACACAGGTACGGTA...
ACACAGGTACGTTC...
ACACAGGTACTTTT...
ACACAGGTCCCTTA...
ACACAGTGAACCTA...
ACACCACTGACTAA...
ACCGTTTACGCTTA...
ACCCGGGTAAATTT...
```

How many possible matches are there?

```
AACAACGCTAACTA...
ACAACGCATTACGT...
ACACAGGTAACTGA...
ACACAGGTAACTTC...
ACACAGGTAACTTG...
ACACAGGTACGGTA...
ACACAGGTACGTTC...
ACACAGGTACTTTT...
ACACAGGTCCCTTA...
ACACAGTGAACCTA...
ACACCACTGACTAA...
ACCGTTTACGCTTA...
ACCCGGGTAAATTT...
```

```
AACAACGCTAACTA...
ACAACGCATTACGT...
ACACAGGTAACTGA...
ACACAGGTAACTTC...
ACACAGGTAACTTG...
ACACAGGTACGGTA...
ACACAGGTACGTTC...
ACACAGGTACTTTT...
ACACAGGTCCCTTA...
ACACAGTGAACCTA...
ACACCACTGACTAA...
ACCGTTTACGCTTA...
ACCCGGGTAAATTT...
```

HW2 Questions?

- Notes:

HW2 Questions?

- Notes:
  - Assume the graph lists vertices in depth order
    - Write your representation of the graph we give in depth order
    - Make sure you write the sequence graph file in depth order

HW2 Questions?

- Notes:
  - Assume the graph lists vertices in depth order
    - Write your representation of the graph we give in depth order
    - Make sure you write the sequence graph file in depth order
  - How do you find the vertex at the beginning of your path?

Maximal segment vs. Maximal D-segment

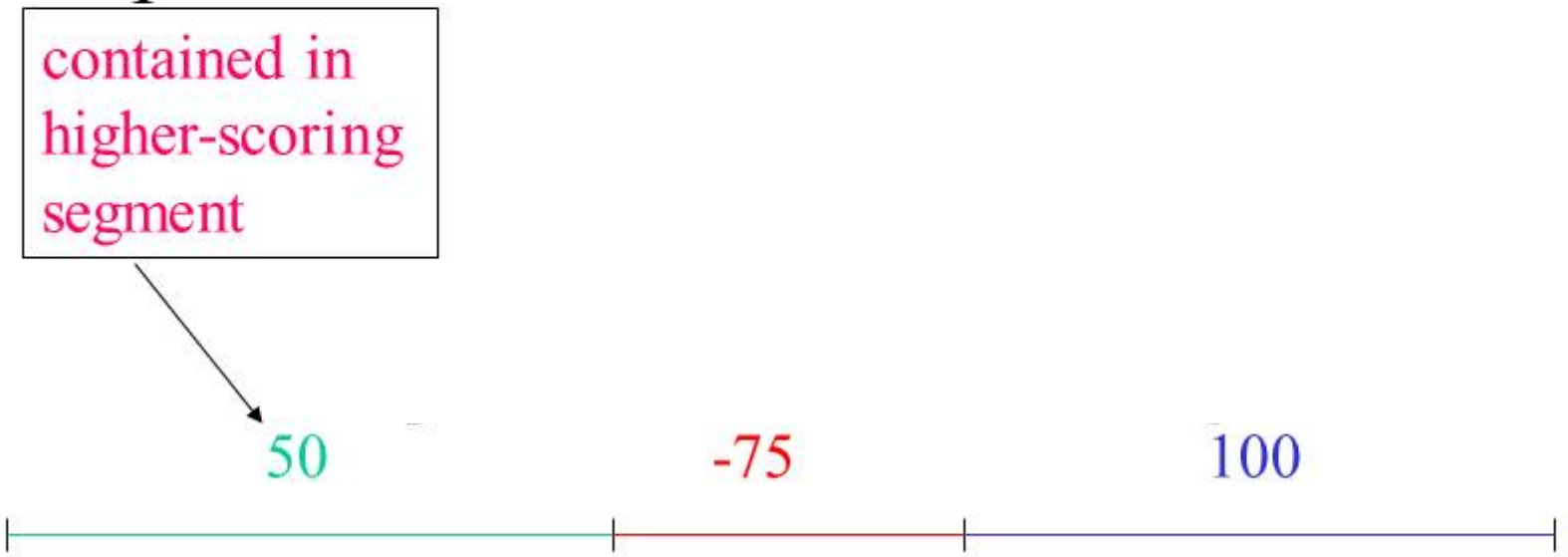
- Maximal segment
  - No subsegment has higher score
  - No segment properly containing the segment satisfies the above

Maximal segment vs. Maximal D-segment

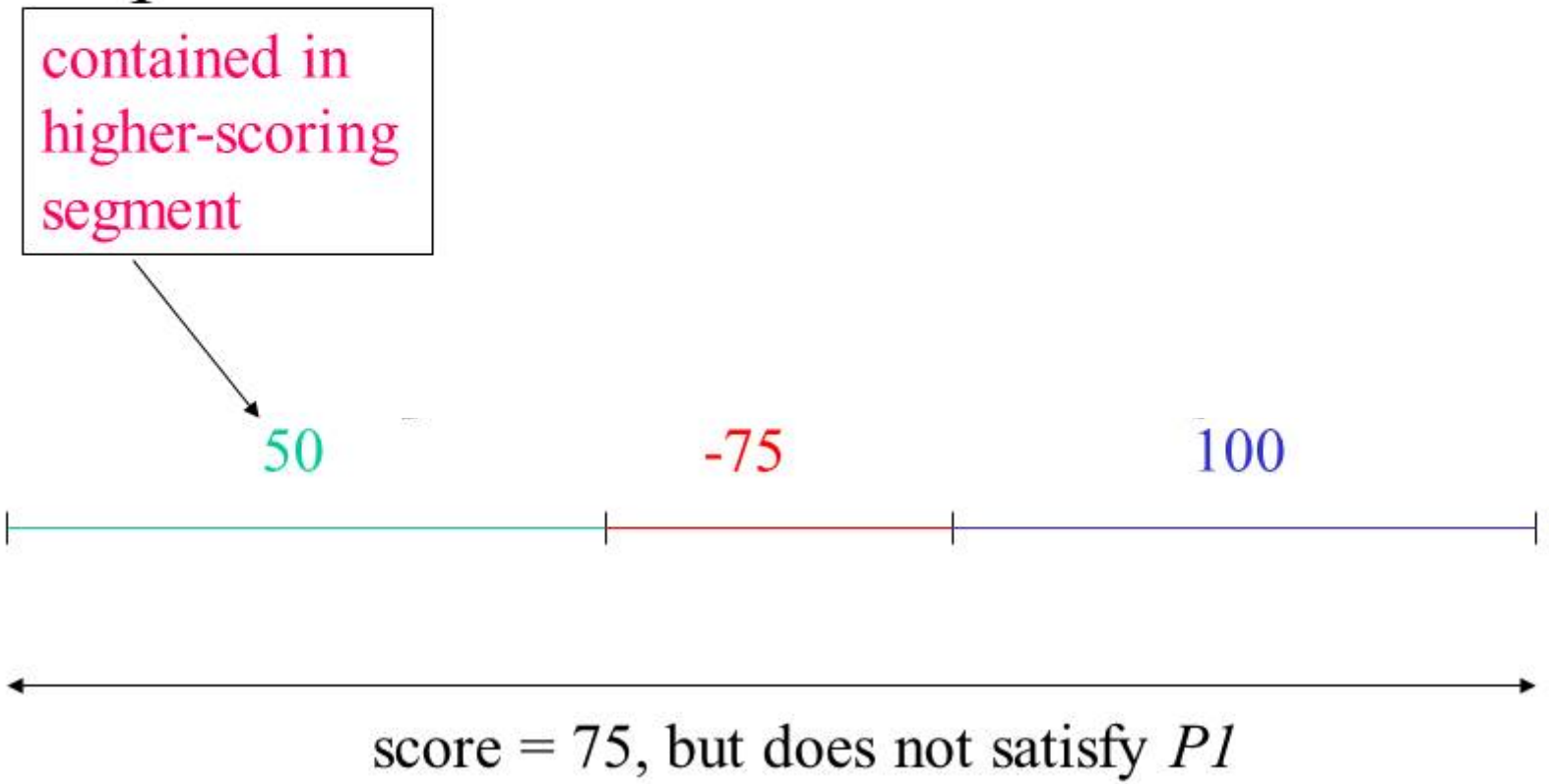
- Maximal segment
  - No subsegment has higher score
  - No segment properly containing the segment satisfies the above
- Maximal D-segment
  - No subsegment has score < D
  - We also require the segment score to be >= S
    - S >= -D

- A *maximal(-scoring) segment* I is one such that
  - *P1:* no subsegment of I has a higher score than I
  - *P2:* no segment properly containing I satisfies *P1*
- Example:

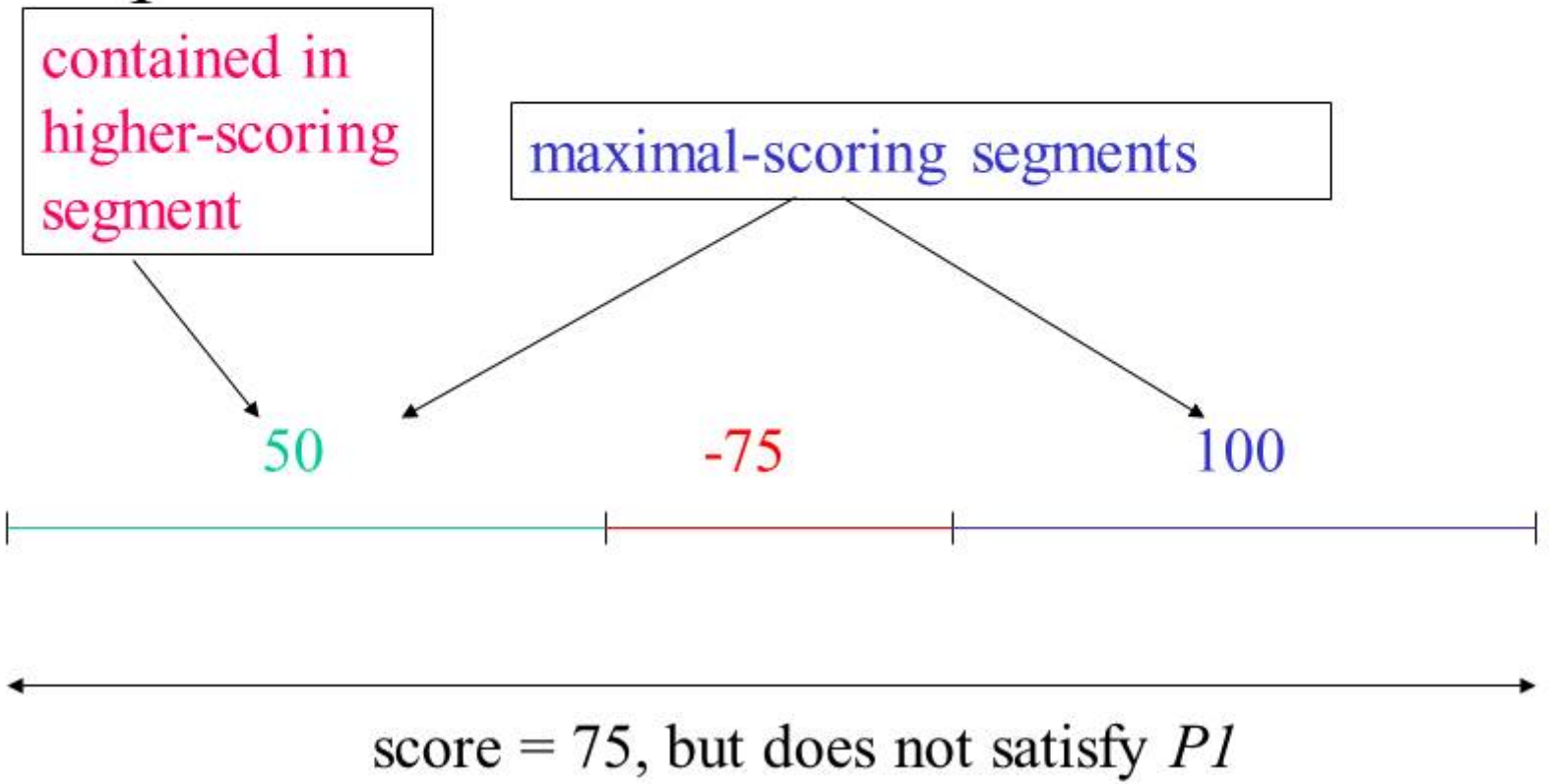
- A *maximal(-scoring) segment* I is one such that
  - *P1:* no subsegment of I has a higher score than I
  - *P2:* no segment properly containing I satisfies *P1*
- Example:

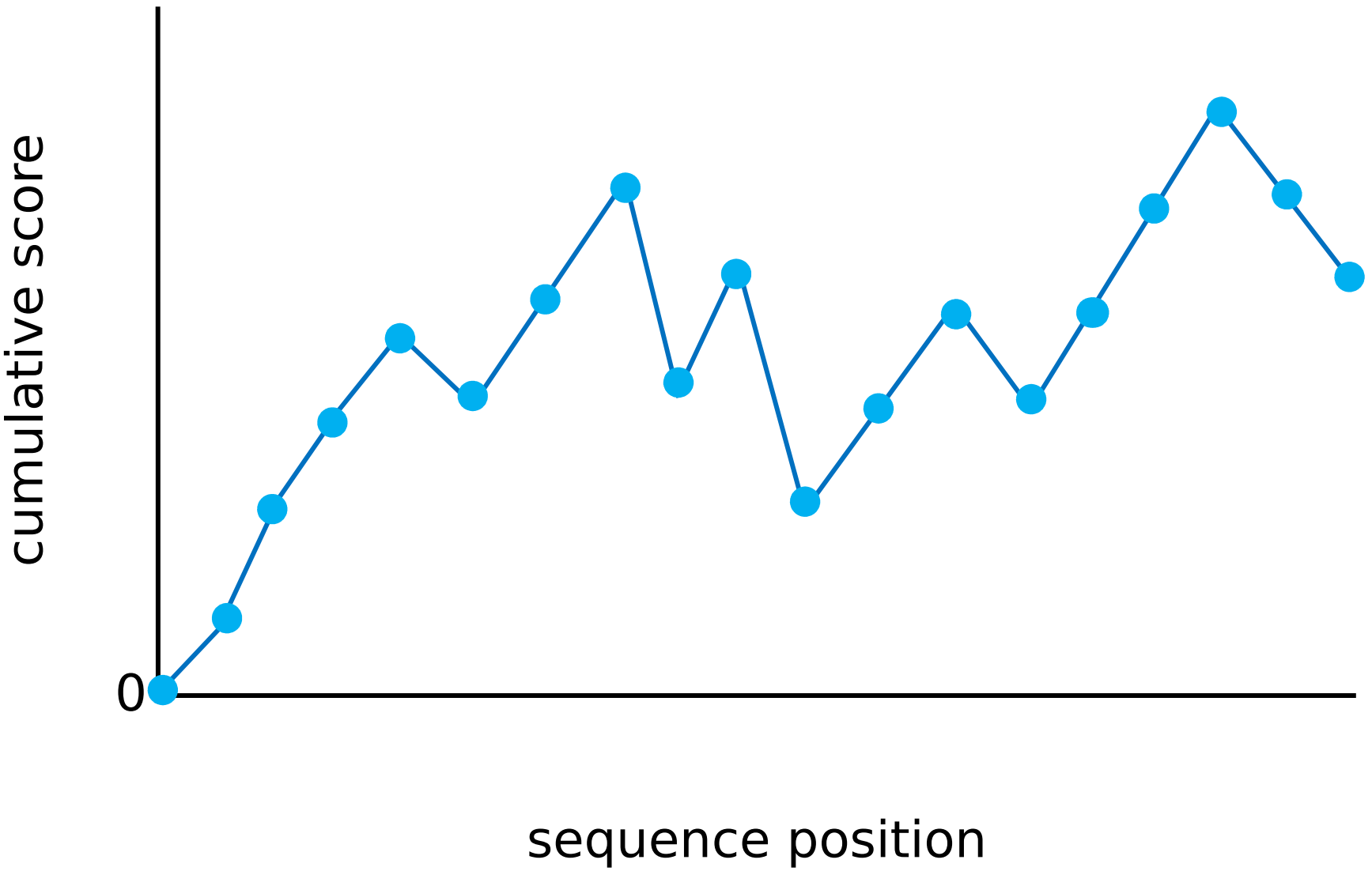


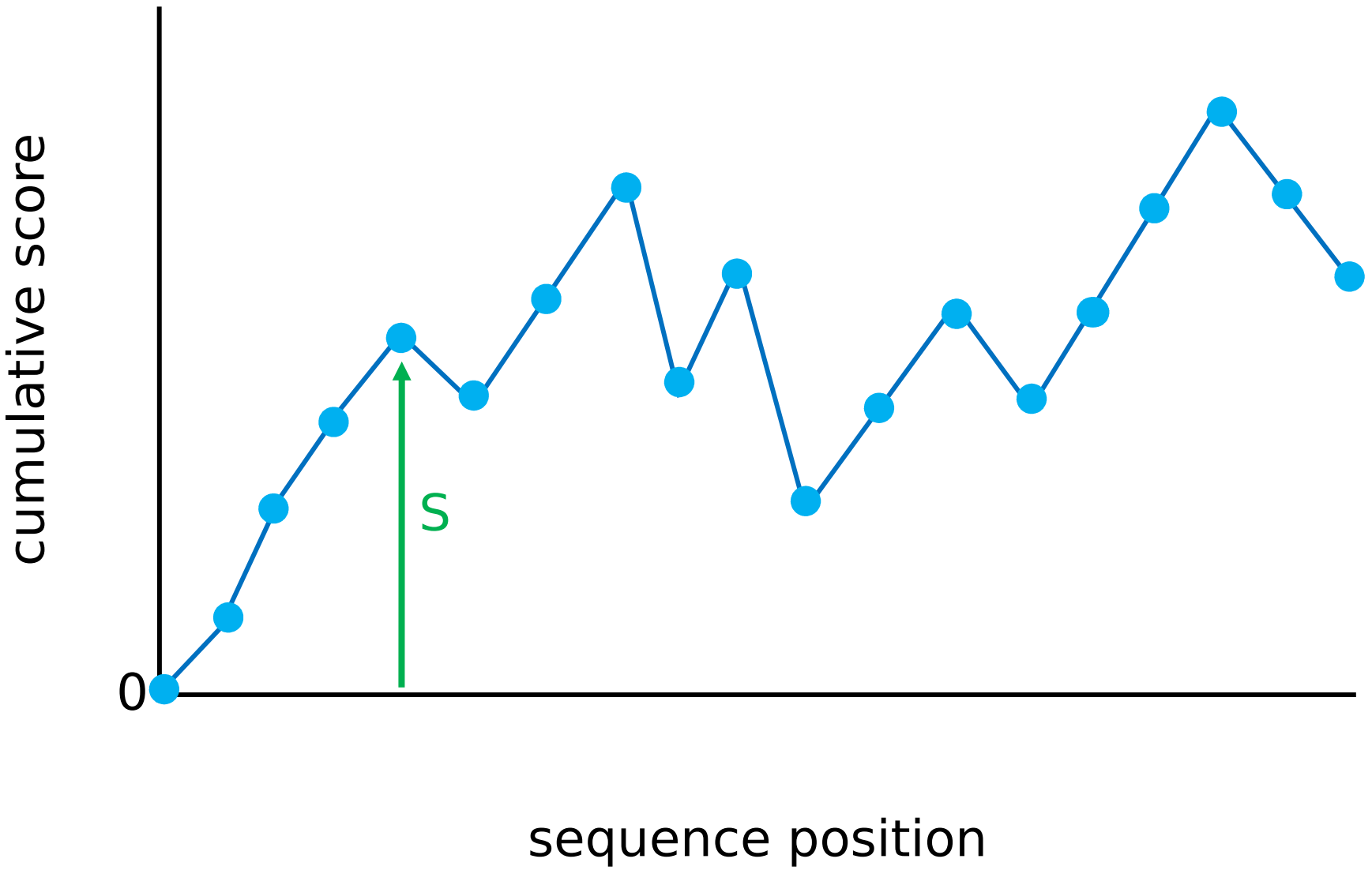
- A *maximal(-scoring) segment* I is one such that
  - *P1:* no subsegment of I has a higher score than I
  - *P2:* no segment properly containing I satisfies *P1*
- Example:

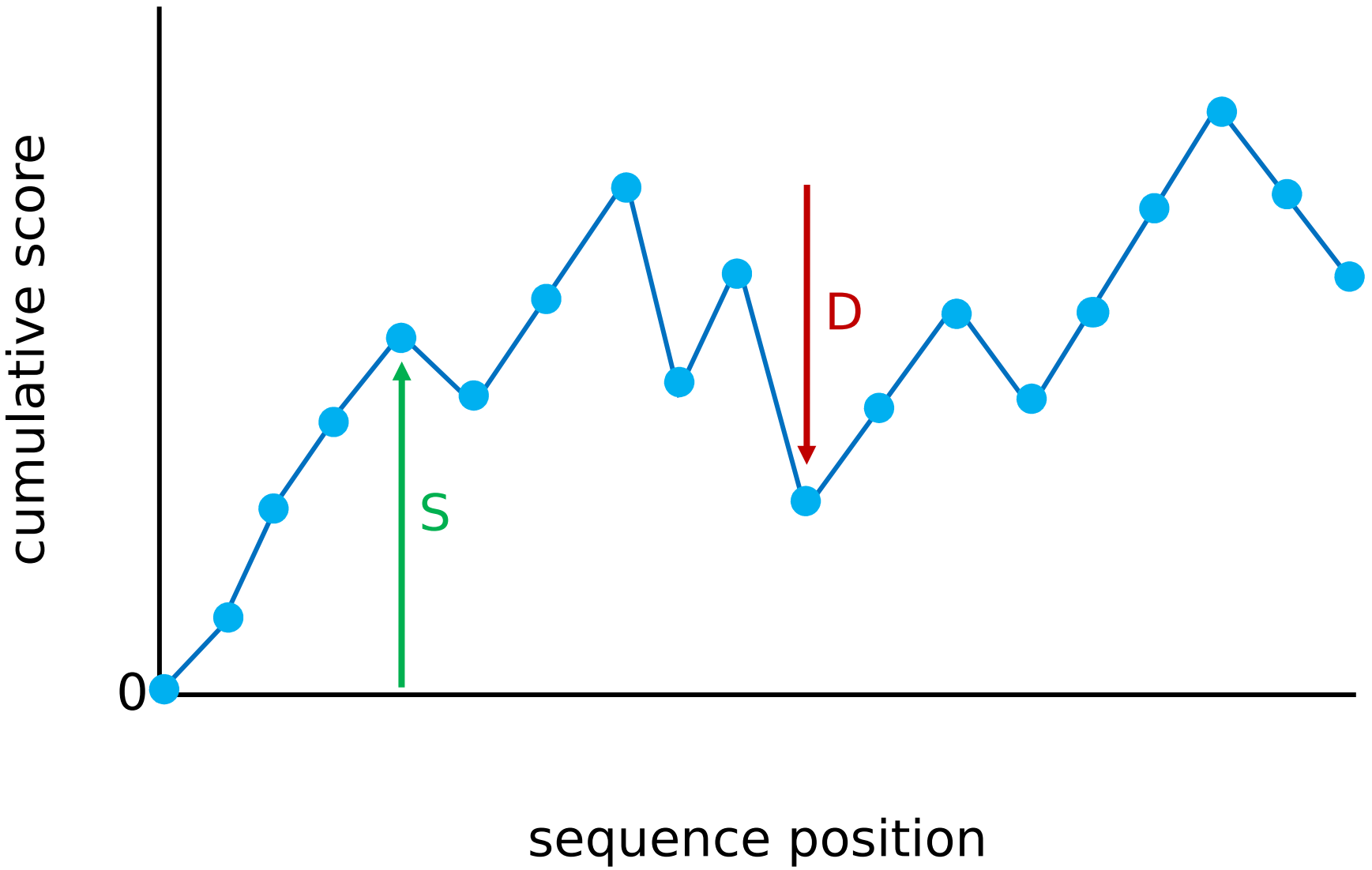


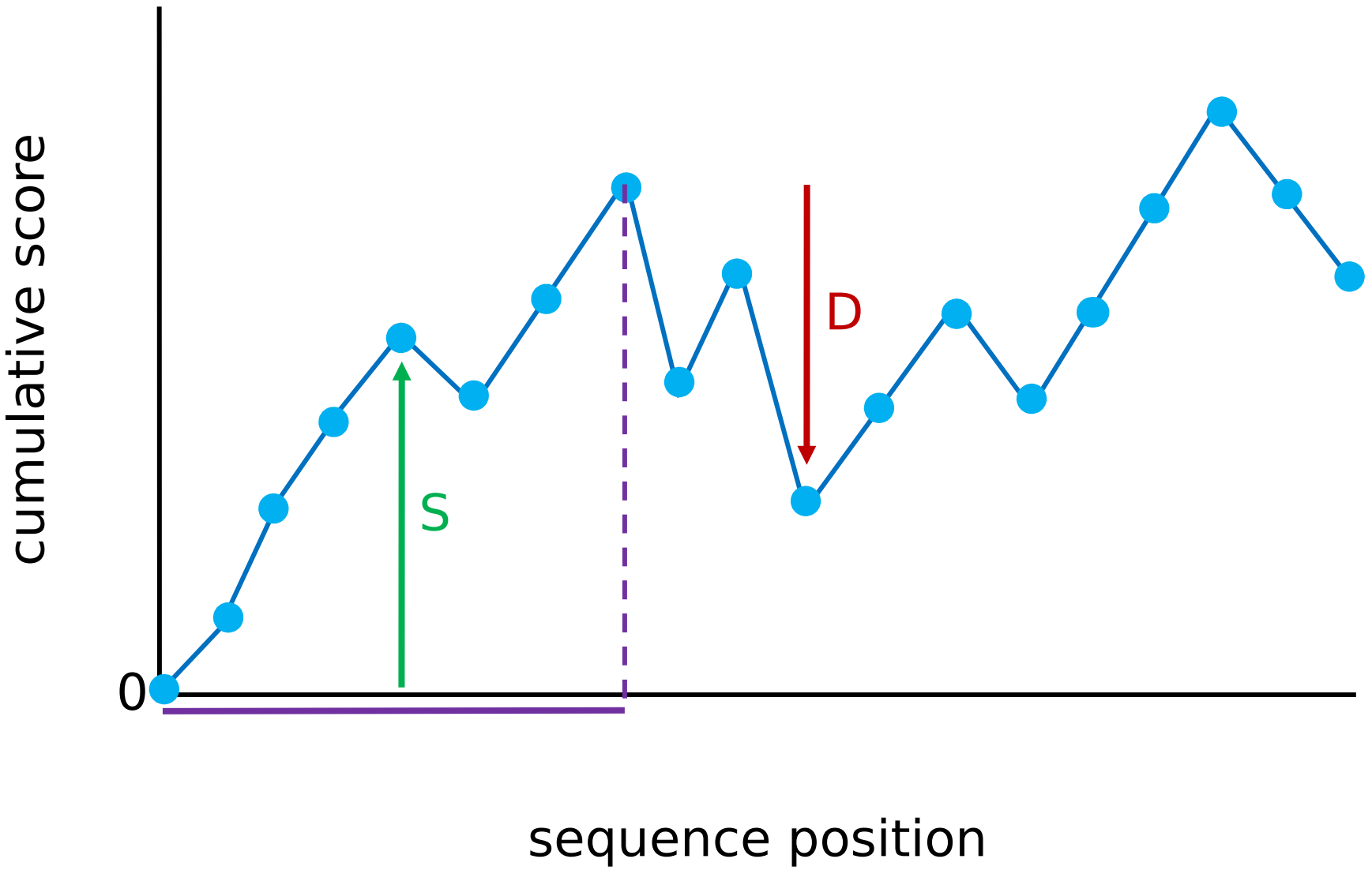
- A *maximal(-scoring) segment* I is one such that
  - *P1:* no subsegment of I has a higher score than I
  - *P2:* no segment properly containing I satisfies *P1*
- Example:

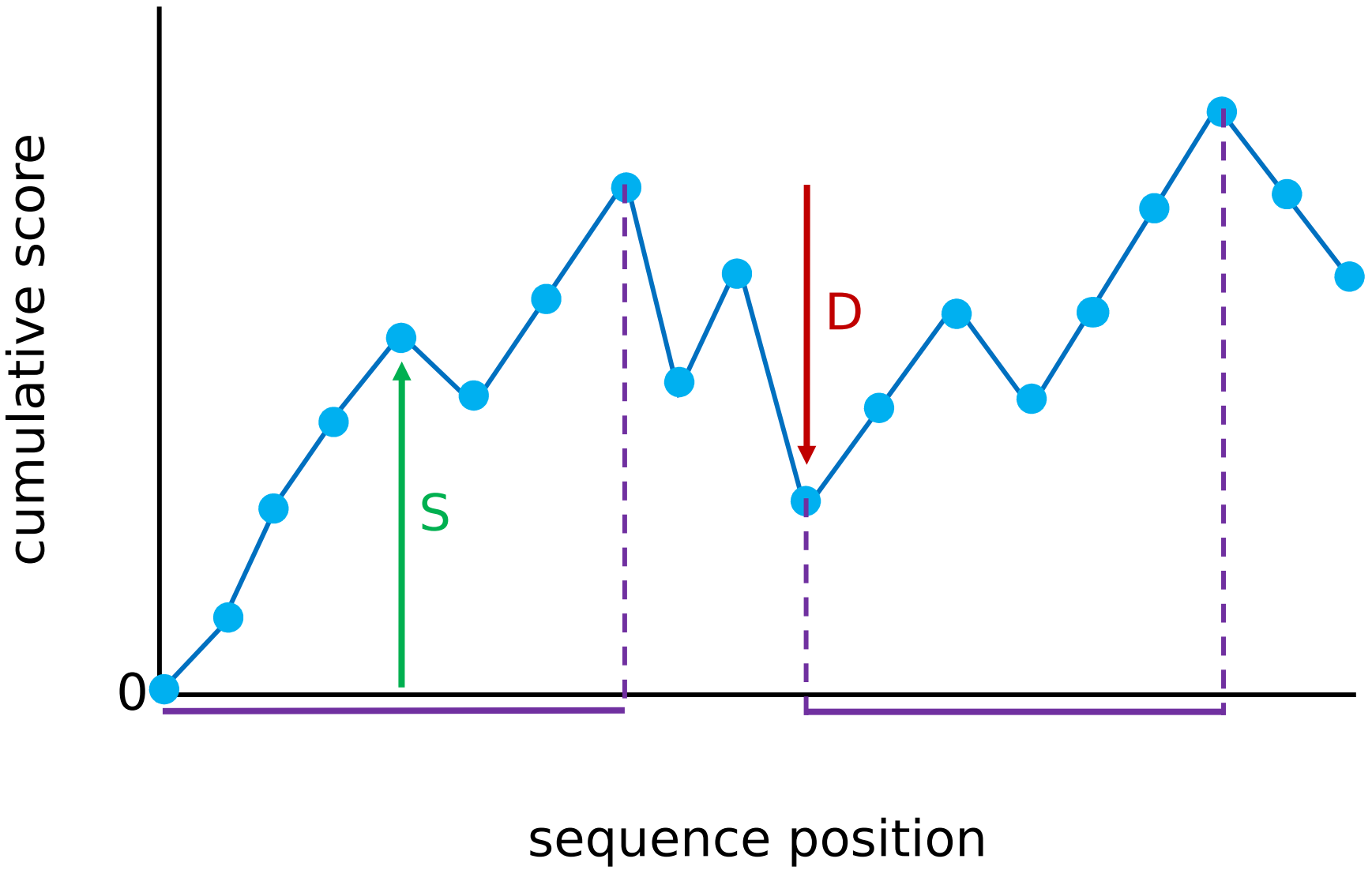



cumulative score
0
sequence position


cumulative score
0
S
sequence position


cumulative score
0
S
D
sequence position


cumulative score
0
S
D
sequence position


cumulative score
0
S
D
sequence position

```
cumul = max = 0; start = 1;
for (i = 1; i ≤ N; i++) {
    cumul += s[i];
    if (cumul ≥ max)
        {max = cum; end = i;}
    if (cumul ≤ 0 or cumul ≤ max + D or i == N) {
        if (max ≥ S)
            {print start, end, max; }
        max = cumul = 0; start = end = i + 1; /* NO BACKTRACKING
            NEEDED! */
    }
}
```

position	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# read starts	0	0	0	0	1	2	0	4	1	2	0	0	0	0
score	-0.5	-0.5	-0.5	-0.5	0.52	1.1	-0.5	1.7	0.52	1.1	-0.5	-0.5	-0.5	-0.5

D = -3

max = 0

start = 1

end = 1

cumul = -0.5

position	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# read starts	0	0	0	0	1	2	0	4	1	2	0	0	0	0
score	-0.5	-0.5	-0.5	-0.5	0.52	1.1	-0.5	1.7	0.52	1.1	-0.5	-0.5	-0.5	-0.5

D = -3

max = 0

start = 2

end = 2

cumul = -0.5

position	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# read starts	0	0	0	0	1	2	0	4	1	2	0	0	0	0
score	-0.5	-0.5	-0.5	-0.5	0.52	1.1	-0.5	1.7	0.52	1.1	-0.5	-0.5	-0.5	-0.5

D = -3

max = 0

start = 2

end = 2

cumul = -0.5

position	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# read starts	0	0	0	0	1	2	0	4	1	2	0	0	0	0
score	-0.5	-0.5	-0.5	-0.5	0.52	1.1	-0.5	1.7	0.52	1.1	-0.5	-0.5	-0.5	-0.5

D = -3

max = 0

start = 3

end = 3

cumul = -0.5

position	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# read starts	0	0	0	0	1	2	0	4	1	2	0	0	0	0
score	-0.5	-0.5	-0.5	-0.5	0.52	1.1	-0.5	1.7	0.52	1.1	-0.5	-0.5	-0.5	-0.5

D = -3

max = 0

start = 3

end = 3

cumul = -0.5

position	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# read starts	0	0	0	0	1	2	0	4	1	2	0	0	0	0
score	-0.5	-0.5	-0.5	-0.5	0.52	1.1	-0.5	1.7	0.52	1.1	-0.5	-0.5	-0.5	-0.5

D = -3

max = 0

start = 4

end = 4

cumul = -0.5

position	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# read starts	0	0	0	0	1	2	0	4	1	2	0	0	0	0
score	-0.5	-0.5	-0.5	-0.5	0.52	1.1	-0.5	1.7	0.52	1.1	-0.5	-0.5	-0.5	-0.5

D = -3

max = 0

start = 4

end = 4

cumul = -0.5

position	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# read starts	0	0	0	0	1	2	0	4	1	2	0	0	0	0
score	-0.5	-0.5	-0.5	-0.5	0.52	1.1	-0.5	1.7	0.52	1.1	-0.5	-0.5	-0.5	-0.5

D = -3

max = 0

start = 5

end = 5

cumul = -0.5

position	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# read starts	0	0	0	0	1	2	0	4	1	2	0	0	0	0
score	-0.5	-0.5	-0.5	-0.5	0.52	1.1	-0.5	1.7	0.52	1.1	-0.5	-0.5	-0.5	-0.5

D = -3

max = 0

start = 5

end = 5

cumul = -0.5

position	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# read starts	0	0	0	0	1	2	0	4	1	2	0	0	0	0
score	-0.5	-0.5	-0.5	-0.5	0.52	1.1	-0.5	1.7	0.52	1.1	-0.5	-0.5	-0.5	-0.5

D = -3

max = 0.52

start = 5

end = 5

cumul = 0.52

position	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# read starts	0	0	0	0	1	2	0	4	1	2	0	0	0	0
score	-0.5	-0.5	-0.5	-0.5	0.52	1.1	-0.5	1.7	0.52	1.1	-0.5	-0.5	-0.5	-0.5

D = -3

max = 0.52

start = 5

end = 5

cumul = 0.52

position	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# read starts	0	0	0	0	1	2	0	4	1	2	0	0	0	0
score	-0.5	-0.5	-0.5	-0.5	0.52	1.1	-0.5	1.7	0.52	1.1	-0.5	-0.5	-0.5	-0.5

D = -3

max = 1.62

start = 5

end = 6

cumul = 1.62

position	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# read starts	0	0	0	0	1	2	0	4	1	2	0	0	0	0
score	-0.5	-0.5	-0.5	-0.5	0.52	1.1	-0.5	1.7	0.52	1.1	-0.5	-0.5	-0.5	-0.5

D = -3

max = 1.62

start = 5

end = 6

cumul = 1.62

position	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# read starts	0	0	0	0	1	2	0	4	1	2	0	0	0	0
score	-0.5	-0.5	-0.5	-0.5	0.52	1.1	-0.5	1.7	0.52	1.1	-0.5	-0.5	-0.5	-0.5

D = -3

max = 1.62

start = 5

end = 6

cumul = 1.12

position	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# read starts	0	0	0	0	1	2	0	4	1	2	0	0	0	0
score	-0.5	-0.5	-0.5	-0.5	0.52	1.1	-0.5	1.7	0.52	1.1	-0.5	-0.5	-0.5	-0.5

D = -3

max = 1.62

start = 5

end = 6

cumul = 1.12

position	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# read starts	0	0	0	0	1	2	0	4	1	2	0	0	0	0
score	-0.5	-0.5	-0.5	-0.5	0.52	1.1	-0.5	1.7	0.52	1.1	-0.5	-0.5	-0.5	-0.5

D = -3

max = 2.82

start = 5

end = 8

cumul = 2.82

position	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# read starts	0	0	0	0	1	2	0	4	1	2	0	0	0	0
score	-0.5	-0.5	-0.5	-0.5	0.52	1.1	-0.5	1.7	0.52	1.1	-0.5	-0.5	-0.5	-0.5

D = -3

max = 2.82

start = 5

end = 8

cumul = 2.82

position	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# read starts	0	0	0	0	1	2	0	4	1	2	0	0	0	0
score	-0.5	-0.5	-0.5	-0.5	0.52	1.1	-0.5	1.7	0.52	1.1	-0.5	-0.5	-0.5	-0.5

D = -3

max = 3.34

start = 5

end = 9

cumul = 3.34

position	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# read starts	0	0	0	0	1	2	0	4	1	2	0	0	0	0
score	-0.5	-0.5	-0.5	-0.5	0.52	1.1	-0.5	1.7	0.52	1.1	-0.5	-0.5	-0.5	-0.5

D = -3

max = 3.34

start = 5

end = 9

cumul = 3.34

position	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# read starts	0	0	0	0	1	2	0	4	1	2	0	0	0	0
score	-0.5	-0.5	-0.5	-0.5	0.52	1.1	-0.5	1.7	0.52	1.1	-0.5	-0.5	-0.5	-0.5

D = -3

max = 4.44

start = 5

end = 10

cumul = 4.44

position	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# read starts	0	0	0	0	1	2	0	4	1	2	0	0	0	0
score	-0.5	-0.5	-0.5	-0.5	0.52	1.1	-0.5	1.7	0.52	1.1	-0.5	-0.5	-0.5	-0.5

D = -3

max = 4.44

start = 5

end = 10

cumul = 4.44

position	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# read starts	0	0	0	0	1	2	0	4	1	2	0	0	0	0
score	-0.5	-0.5	-0.5	-0.5	0.52	1.1	-0.5	1.7	0.52	1.1	-0.5	-0.5	-0.5	-0.5

D = -3

max = 4.44

start = 5

end = 10

cumul = 3.94

position	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# read starts	0	0	0	0	1	2	0	4	1	2	0	0	0	0
score	-0.5	-0.5	-0.5	-0.5	0.52	1.1	-0.5	1.7	0.52	1.1	-0.5	-0.5	-0.5	-0.5

D = -3

max = 4.44

start = 5

end = 10

cumul = 3.94

position	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# read starts	0	0	0	0	1	2	0	4	1	2	0	0	0	0
score	-0.5	-0.5	-0.5	-0.5	0.52	1.1	-0.5	1.7	0.52	1.1	-0.5	-0.5	-0.5	-0.5

D = -3

max = 4.44

start = 5

end = 10

cumul = 3.44

position	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# read starts	0	0	0	0	1	2	0	4	1	2	0	0	0	0
score	-0.5	-0.5	-0.5	-0.5	0.52	1.1	-0.5	1.7	0.52	1.1	-0.5	-0.5	-0.5	-0.5

D = -3

max = 4.44

start = 5

end = 10

cumul = 3.44

position	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# read starts	0	0	0	0	1	2	0	4	1	2	0	0	0	0
score	-0.5	-0.5	-0.5	-0.5	0.52	1.1	-0.5	1.7	0.52	1.1	-0.5	-0.5	-0.5	-0.5

D = -3

max = 4.44

start = 5

end = 10

cumul = 2.94

position	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# read starts	0	0	0	0	1	2	0	4	1	2	0	0	0	0
score	-0.5	-0.5	-0.5	-0.5	0.52	1.1	-0.5	1.7	0.52	1.1	-0.5	-0.5	-0.5	-0.5

D = -3

max = 4.44

start = 5

end = 10

cumul = 2.94

position	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# read starts	0	0	0	0	1	2	0	4	1	2	0	0	0	0
score	-0.5	-0.5	-0.5	-0.5	0.52	1.1	-0.5	1.7	0.52	1.1	-0.5	-0.5	-0.5	-0.5

D = -3

max = 4.44

start = 5

end = 10

cumul = 2.44

position	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# read starts	0	0	0	0	1	2	0	4	1	2	0	0	0	0
score	-0.5	-0.5	-0.5	-0.5	0.52	1.1	-0.5	1.7	0.52	1.1	-0.5	-0.5	-0.5	-0.5

D = -3

max = 4.44

start = 5

end = 10

cumul = 2.44

position	1	2	3	4	5	6	7	8	9	10	11	12	13	14
# read starts	0	0	0	0	1	2	0	4	1	2	0	0	0	0
score	-0.5	-0.5	-0.5	-0.5	0.52	1.1	-0.5	1.7	0.52	1.1	-0.5	-0.5	-0.5	-0.5

D = -3

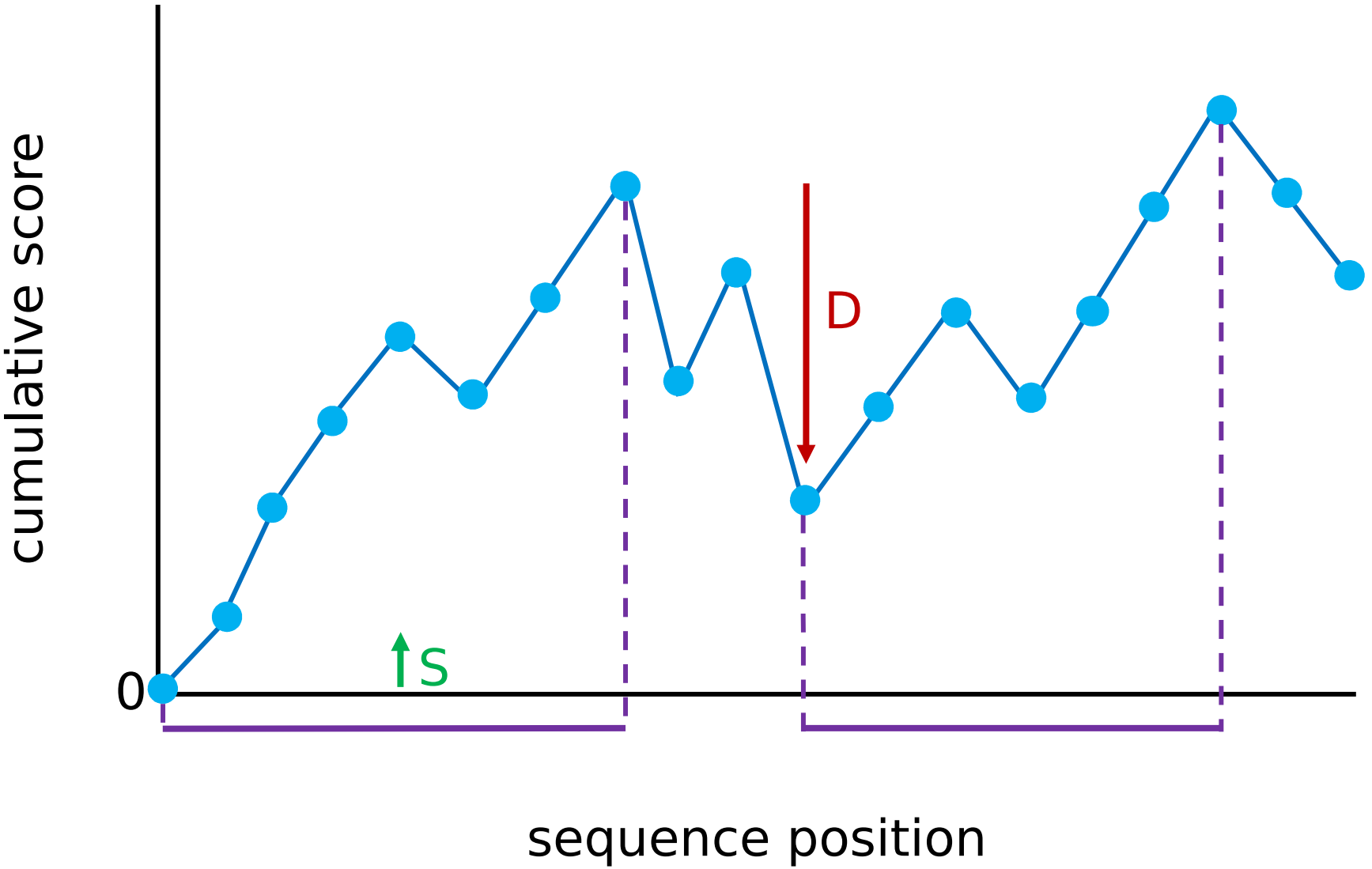
max = 4.44

start = 5

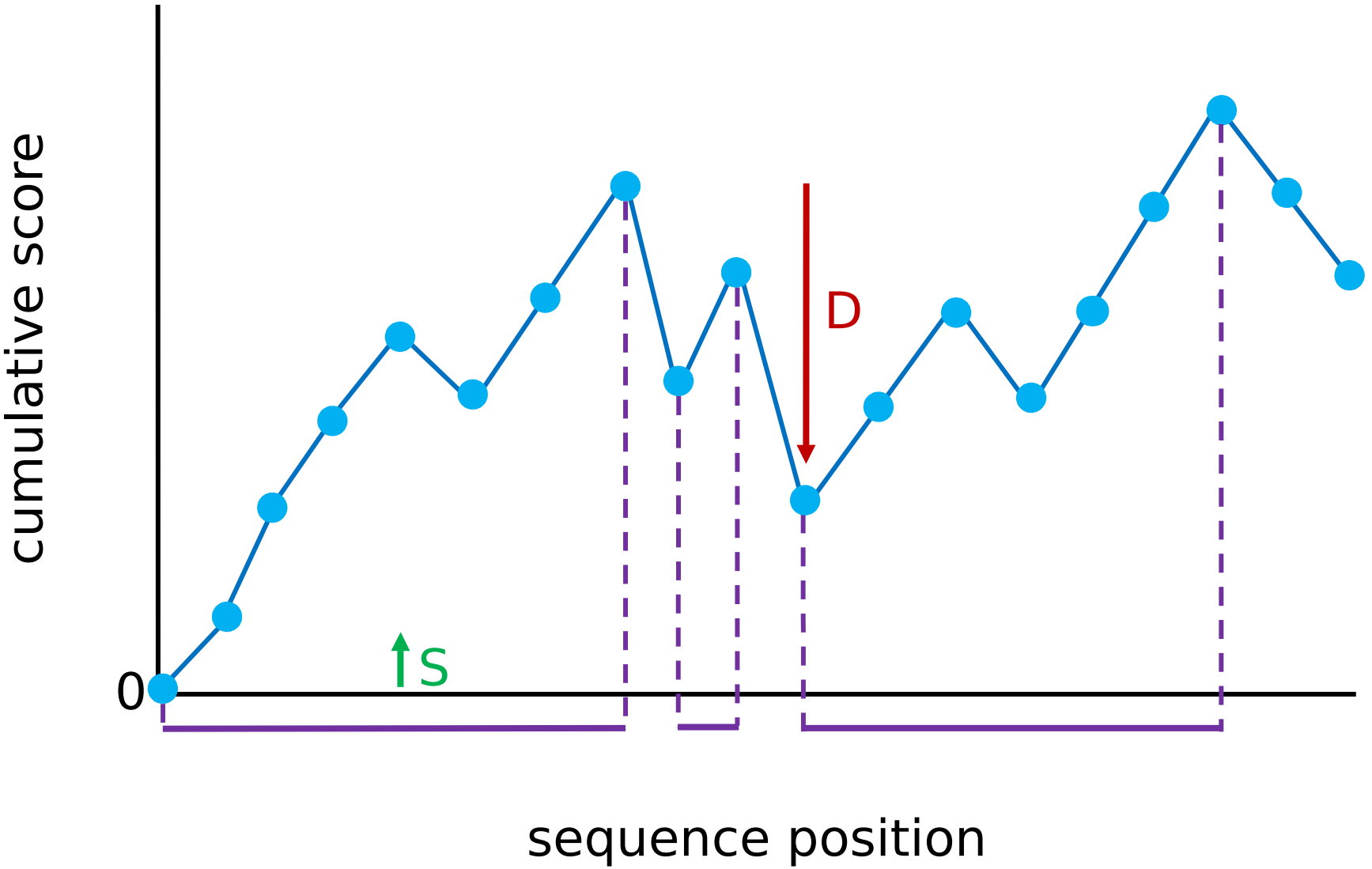
end = 10

cumul = 2.44

Why should we set S >= -D?



Why should we set S >= -D?



Coding Practices (Inspired by HW1)

- Taking in arguments
- for vs. while loops
- Global variables are hurting the environment
- Passing by reference
- Constant variables (not necessarily an oxymoron)

Taking Arguments

Taking Arguments

In C++:

```
24
25  int main(int argc, const char* argv[])
26  {
27      string fn = argv[1];
```

Taking Arguments

In C++:

```
25  int main(int argc, const char* argv[])
26  {
27      string fn = argv[1];
```

In Python:

```
import sys

input_filename = sys.argv[1]
input_threshold = float(sys.argv[2])
```

for vs. while loops

- for loop
  - more intuitive/readable with a known range
  - keeps important iteration information in one place

for vs. while loops

- for loop
  - more intuitive/readable with a known range
  - keeps important iteration information in one place

- while loop
  - number of iterations is hard to define
  - expects side effects from loop operations

for vs. while loops

```
for (int i = 0; i < 10; i++){
    do cool stuff that doesn't affect i;
}
```

for vs. while loops

```
for (int i = 0; i < 10; i++){
    do cool stuff that doesn't affect i;
}

int i = 0;
while (i < 10){
    do cool stuff that doesn't affect i;
    i++;
}
```

for vs. while loops

```
int i = 5;
while (i > 0){
    do cool stuff that changes i;
}
```

for vs. while loops

```
int i = 5;
while (i > 0){
    do cool stuff that changes i;
}

for (int i = 5; i > 0;){
    do cool stuff that changes i;
}
```

Global variables are hurting the environment

string cool_string = “cool”

Global variables are hurting the environment

```
string cool_string = “cool”

void make_string_cool(string &uncool_string){
    uncool_string = cool_string + uncool_string;
}
```

Global variables are hurting the environment

```
string cool_string = “cool”

void make_string_cool(string &uncool_string){
    uncool_string = cool_string + uncool_string;
}

bool check_if_string_is_cool(string possibly_cool_string){
    return(possibly_cool_string == cool_string);
}
```

Global variables are hurting the environment

```
string cool_string = “cool”

void make_string_cool(string &uncool_string){
    uncool_string = cool_string + uncool_string;
}

bool check_if_string_is_cool(string possibly_cool_string){
    return(possibly_cool_string == cool_string);
}

int main(){
    string test_string = “cool”;
    cout<<check_if_string_is_cool(test_string)<<endl;
    make_string_cool(cool_string);
    cout<<check_if_string_is_cool(test_string)<<endl;
}
```

Passing by reference

```
string cool_string = “cool”

void make_string_cool(string &uncool_string){
    uncool_string = cool_string + uncool_string;
}

bool check_if_string_is_cool(string possibly_cool_string){
    return(possibly_cool_string == cool_string);
}

int main(){
    string test_string = “cool”;
    cout<<check_if_string_is_cool(test_string)<<endl;
    make_string_cool(cool_string);
    cout<<check_if_string_is_cool(test_string)<<endl;
}
```

By reference

A copy

Constant variables

```
string cool_string = “cool”

void make_string_cool(string &uncool_string){
    uncool_string = cool_string + uncool_string;
}

bool check_if_string_is_cool(string possibly_cool_string){
    return(possibly_cool_string == cool_string);
}

int main(){
    string test_string = “cool”;
    cout<<check_if_string_is_cool(test_string)<<endl;
    make_string_cool(cool_string);
    cout<<check_if_string_is_cool(test_string)<<endl;
}
```

Constant variables

```
const string cool_string = “cool”

void make_string_cool(string &uncool_string){
    uncool_string = cool_string + uncool_string;
}

bool check_if_string_is_cool(string possibly_cool_string){
    return(possibly_cool_string == cool_string);
}

int main(){
    string test_string = “cool”;
    cout<<check_if_string_is_cool(test_string)<<endl;
    make_string_cool(cool_string);
    cout<<check_if_string_is_cool(test_string)<<endl;
}
```

Constant variables

```
const string cool_string = “cool”

void make_string_cool(string &uncool_string){
    uncool_string = cool_string + uncool_string;
}

bool check_if_string_is_cool(string possibly_cool_string){
    return(possibly_cool_string == cool_string);
}

int main(){
    string test_string = “cool”;
    cout<<check_if_string_is_cool(test_string)<<endl;
    make_string_cool(cool_string); ← will throw an error
    cout<<check_if_string_is_cool(test_string)<<endl;
}
```

Constant variables

- The main use of “const” is to make sure your code isn't doing anything it shouldn't be doing (like assert statements)
- Unfortunately, it can also cause code to be a bit messier (see using built-in C++ library functions)

Some useful data structures

Some useful data structures

- Arrays
  - Fast, pointer math is easy

Some useful data structures

- Arrays
  - Fast, pointer math is easy
- Linked lists
  - Inserting/deleting/reordering is easy

Some useful data structures

- Arrays
  - Fast, pointer math is easy
- Linked lists
  - Inserting/deleting/reordering is easy
- Hash tables/maps
  - Good for looking up things

Some useful data structures

- Arrays
  - Fast, pointer math is easy
- Linked lists
  - Inserting/deleting/reordering is easy
- Hash tables/maps
  - Good for looking up things
- Trees
  - Useful for sorting/searching

Some useful data structures

- Arrays
  - Fast, pointer math is easy
- Linked lists
  - Inserting/deleting/reordering is easy
- Hash tables/maps
  - Good for looking up things
- Trees
  - Useful for sorting/searching
- Heaps
  - Keeping track of extreme values

Arrays

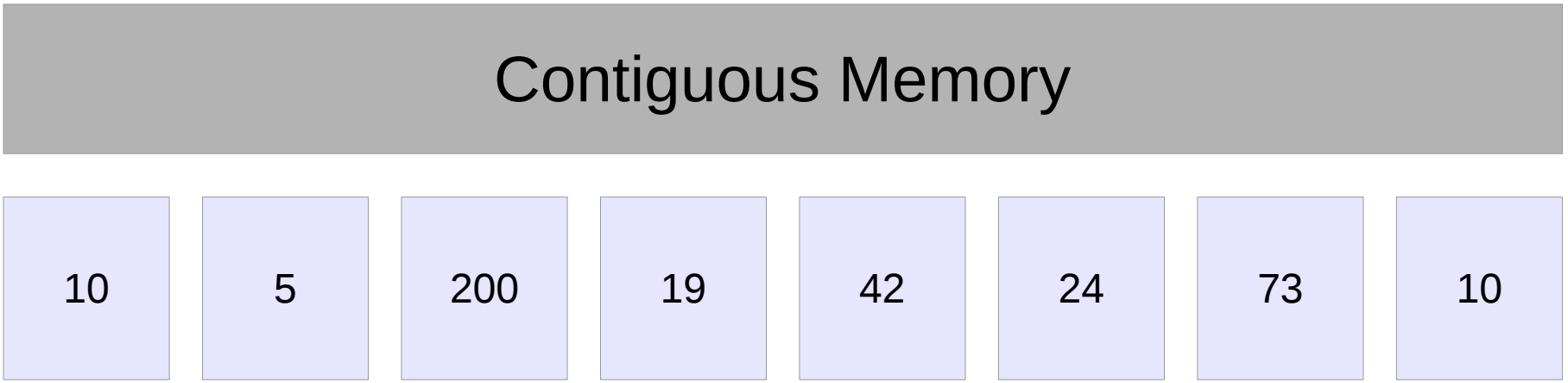
- Getting the element at a particular index is fast

Arrays

- Getting the element at a particular index is fast

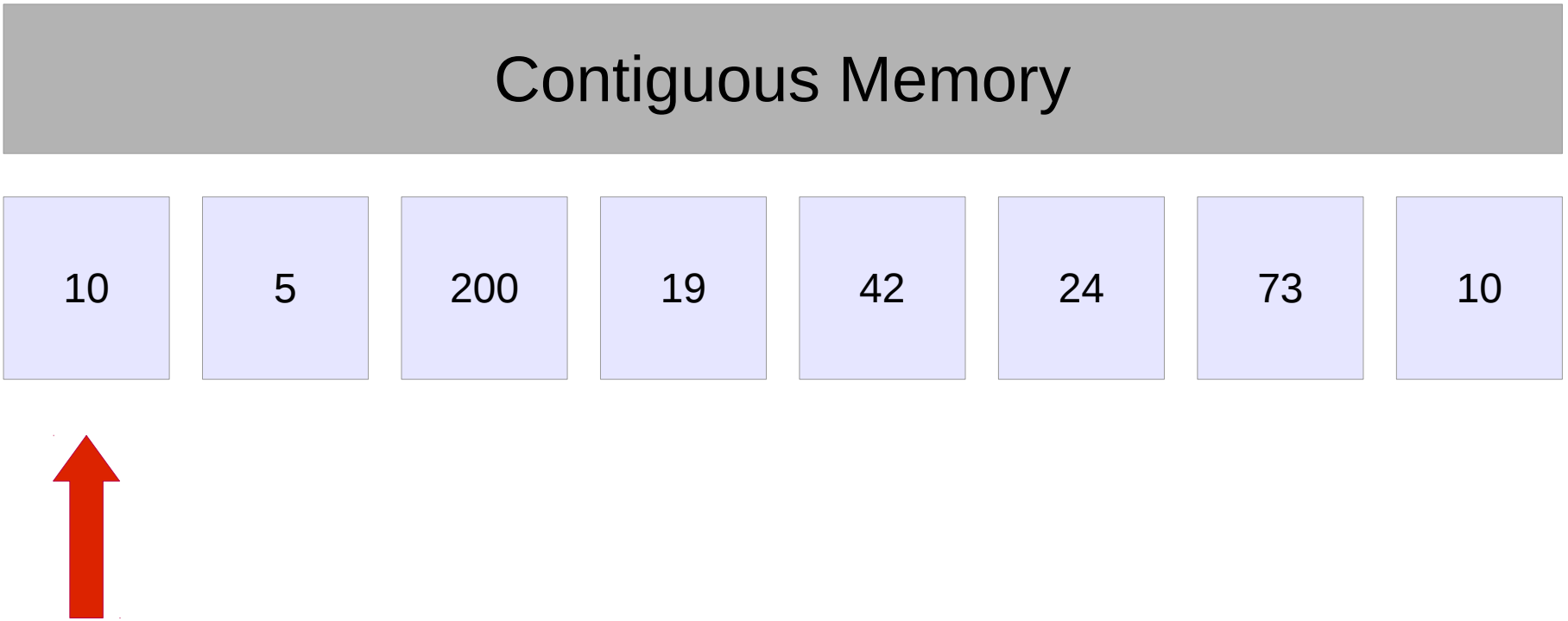
Arrays

- Getting the element at a particular index is fast



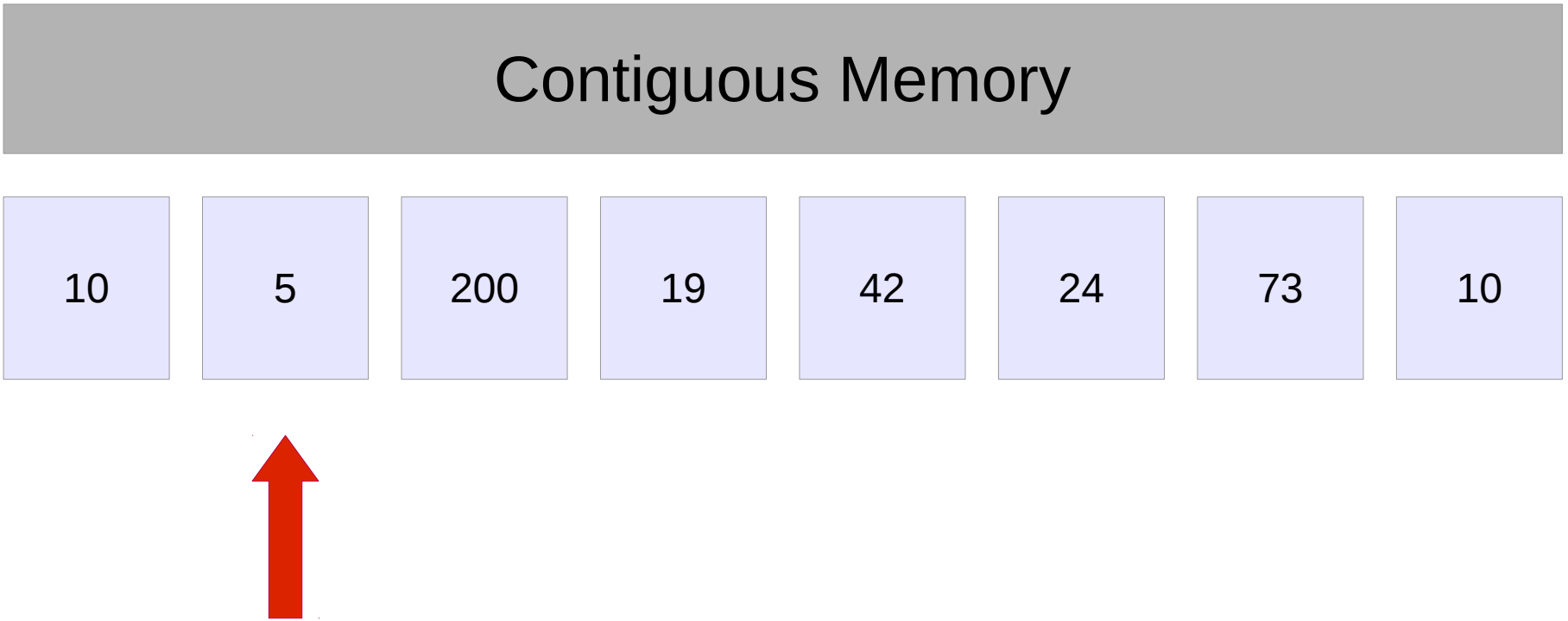
Arrays

- Getting the element at a particular index is fast



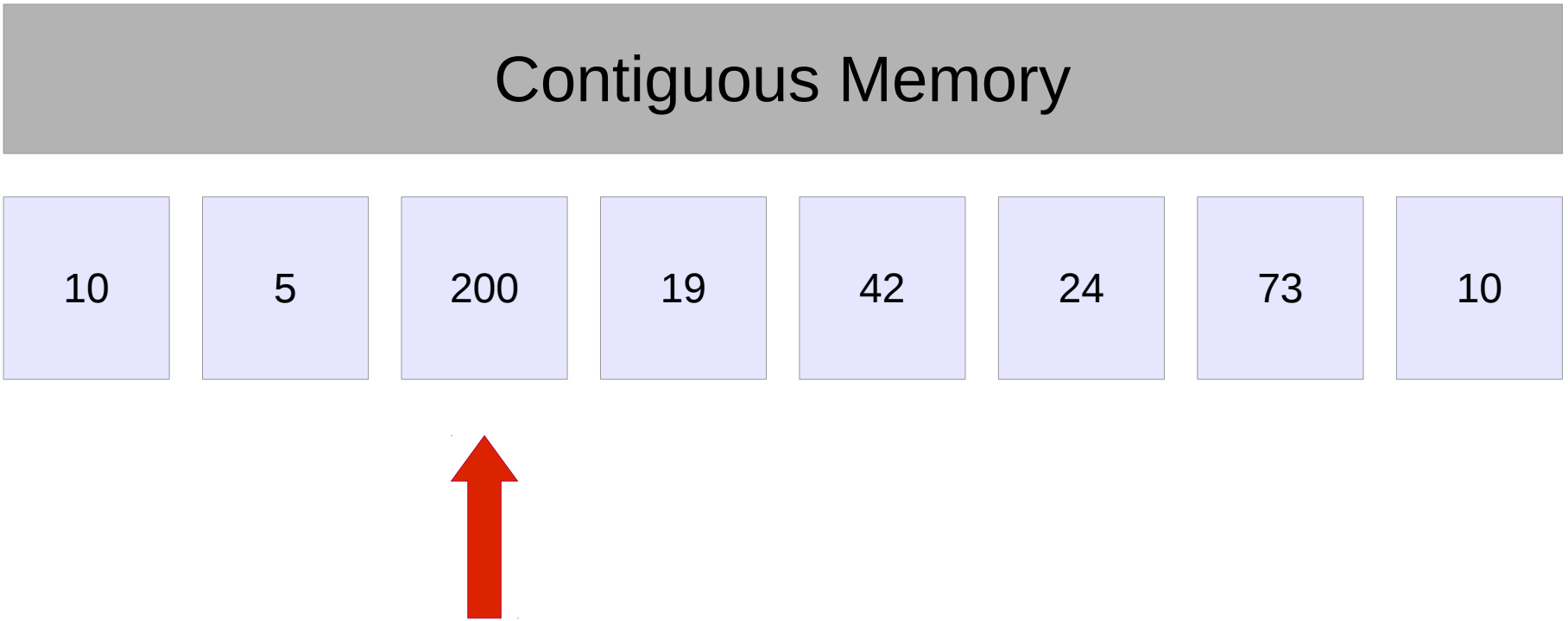
Arrays

- Getting the element at a particular index is fast



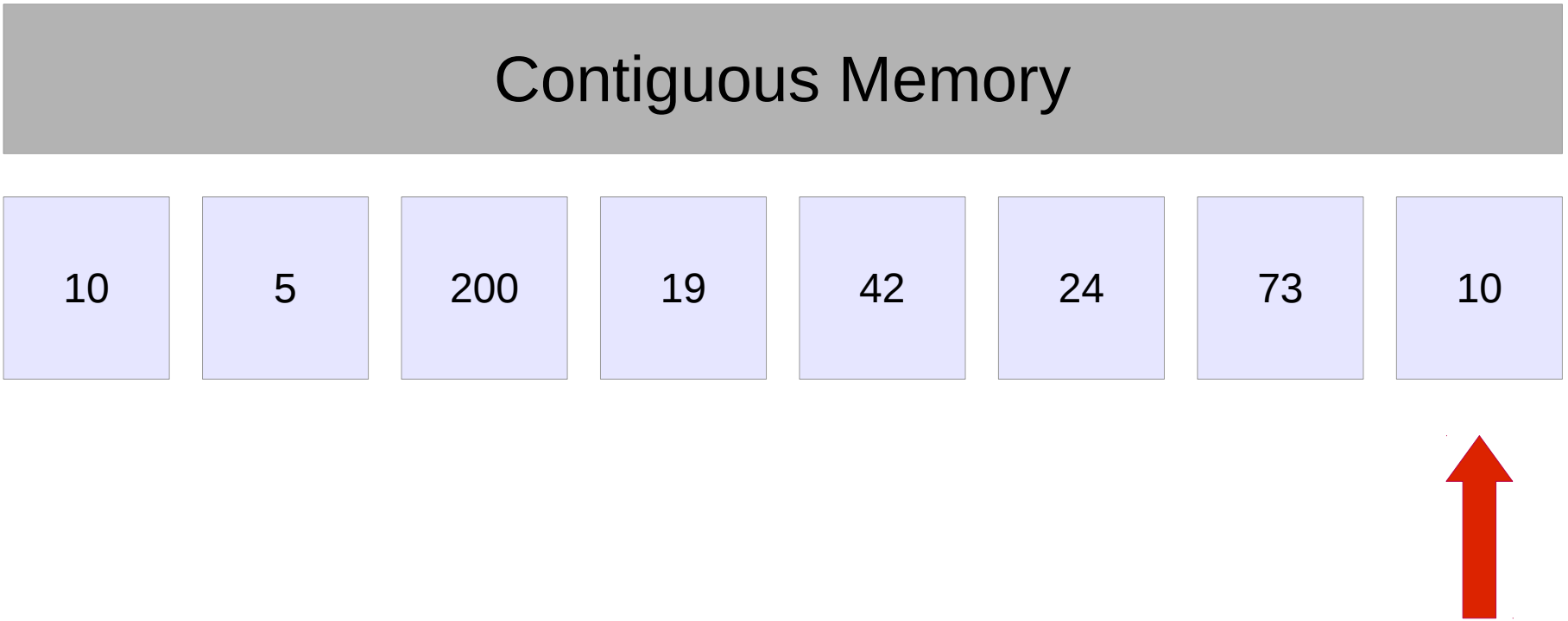
Arrays

- Getting the element at a particular index is fast



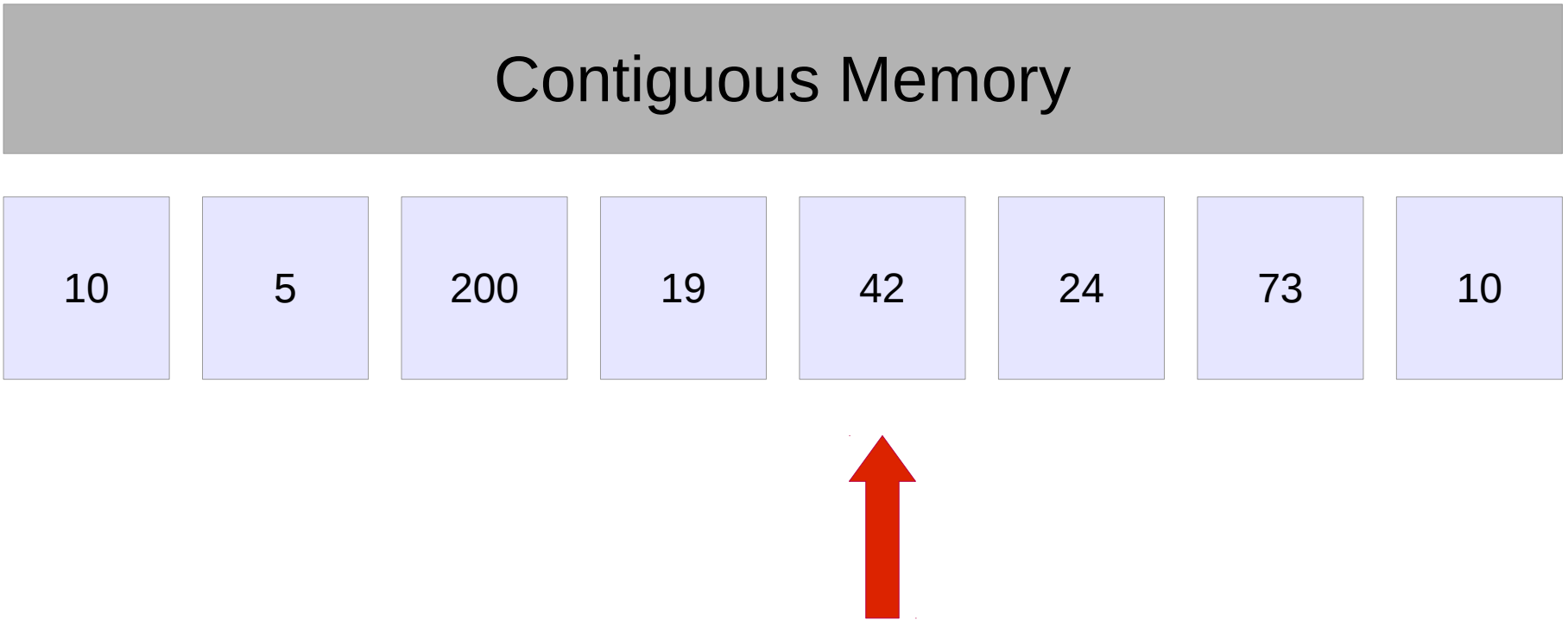
Arrays

- Getting the element at a particular index is fast



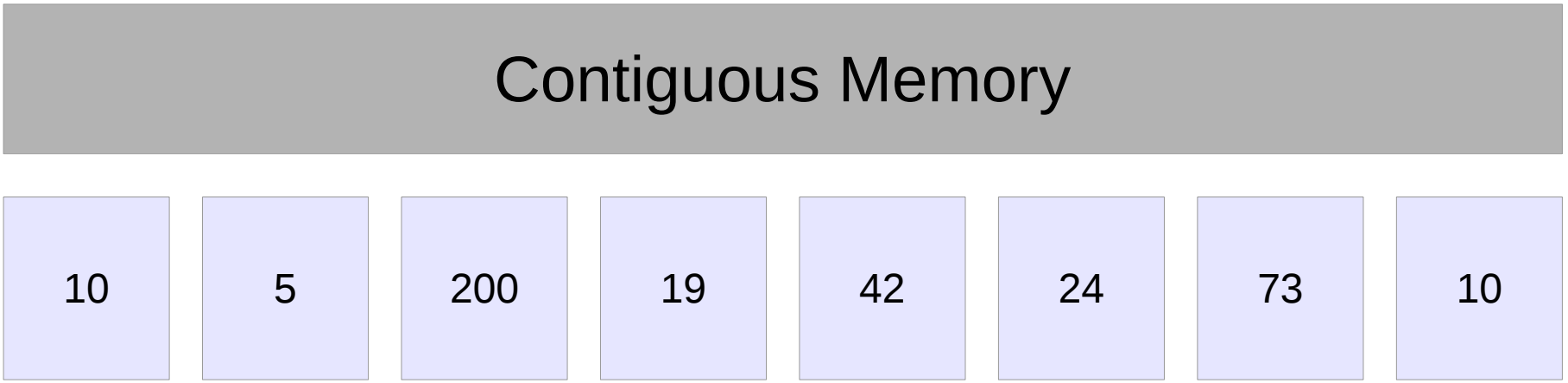
Arrays

- Getting the element at a particular index is fast



Arrays

- Getting the element at a particular index is fast

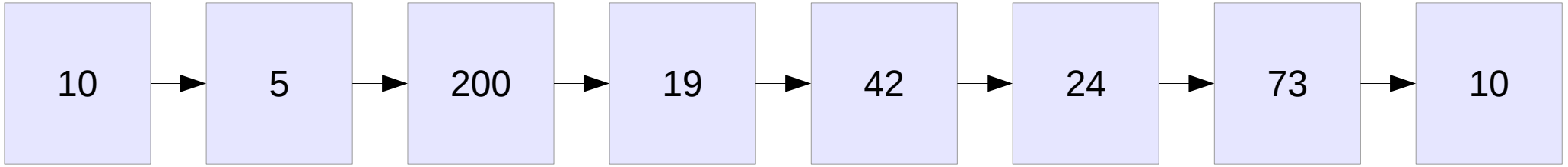


Linked Lists

- Easier to modify than an array

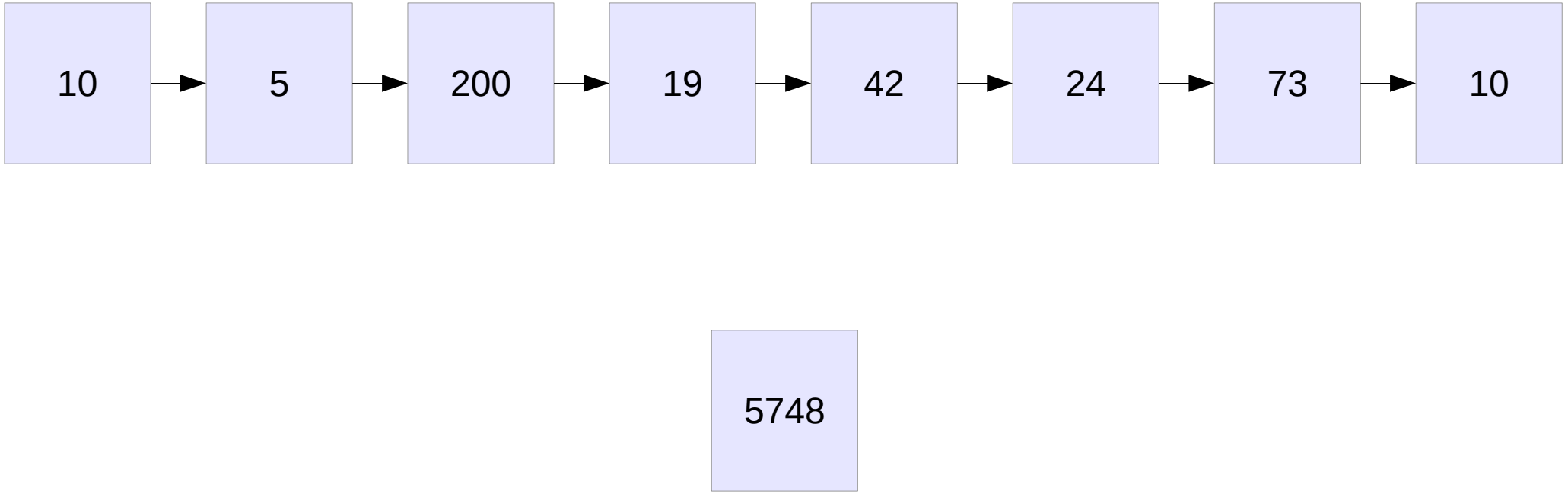
Linked Lists

- Easier to modify than an array



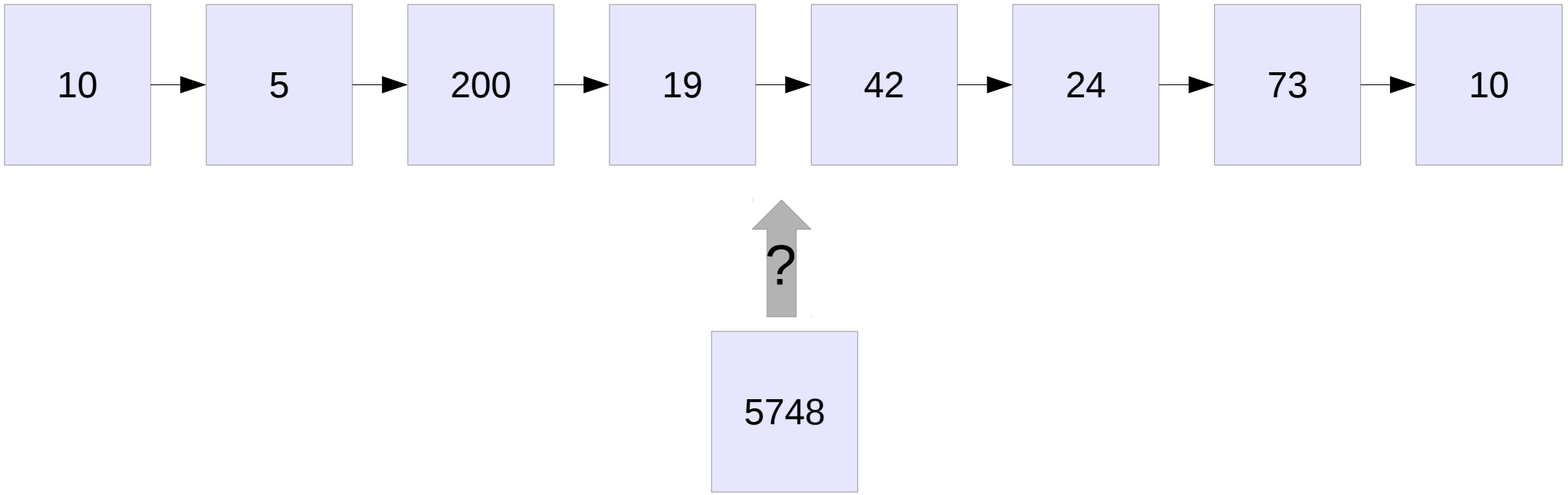
Linked Lists

- Easier to modify than an array



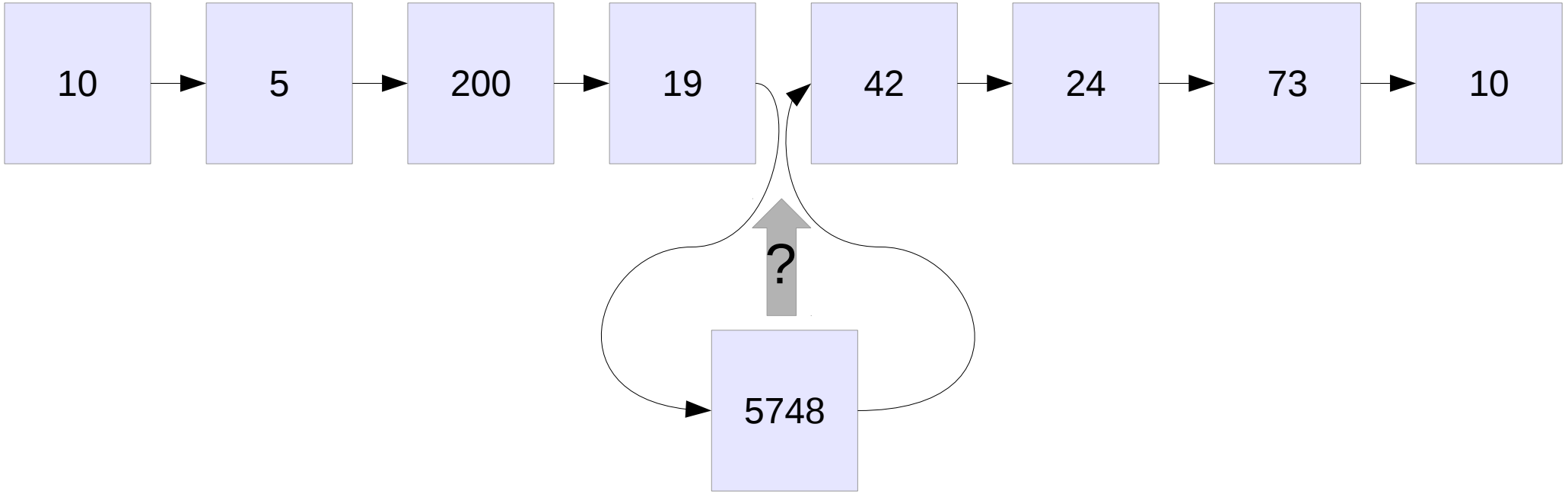
Linked Lists

- Easier to modify than an array



Linked Lists

- Easier to modify than an array

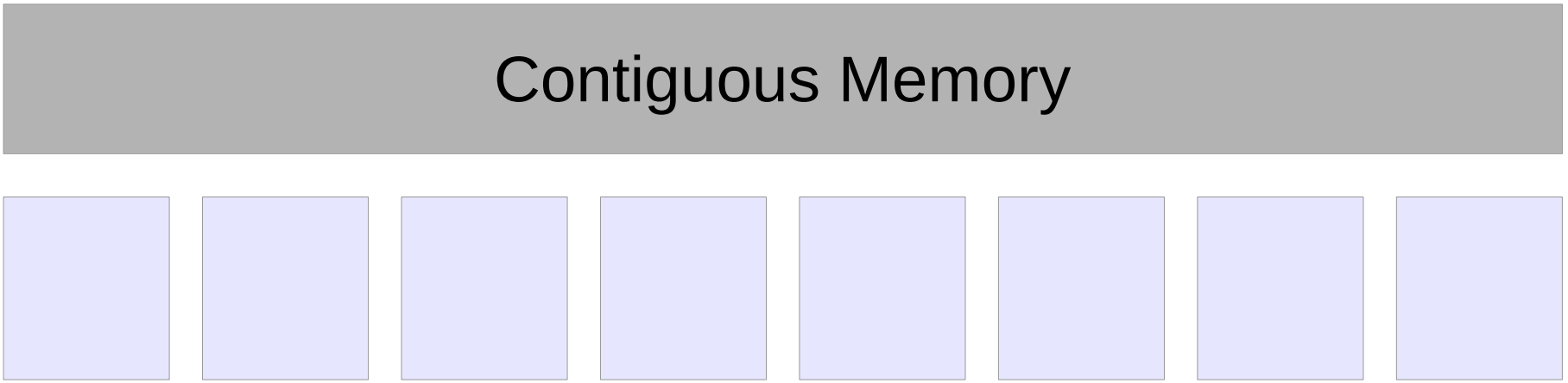


Hash Tables and Hash Maps

- Fast for looking up values

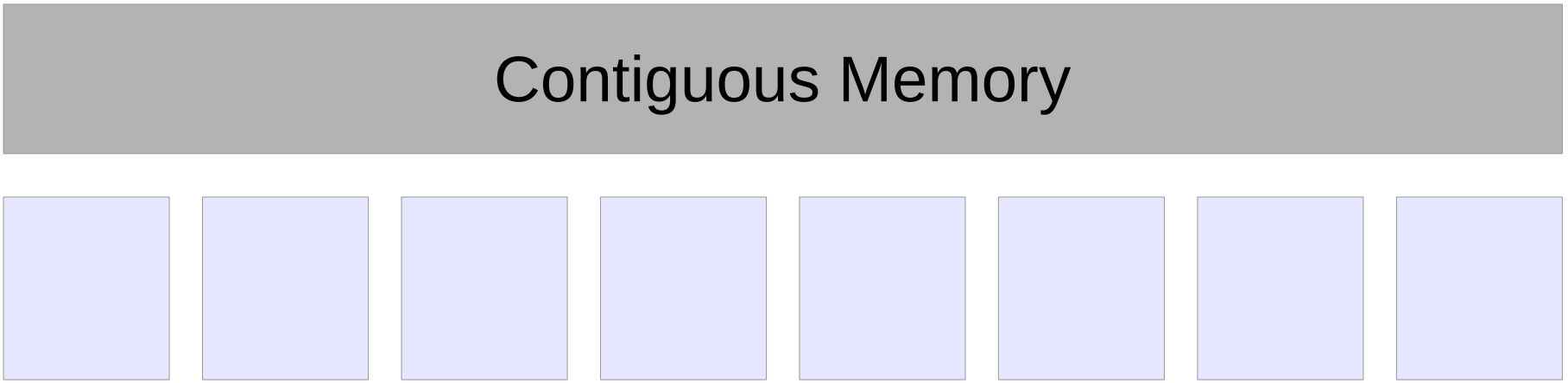
Hash Tables and Hash Maps

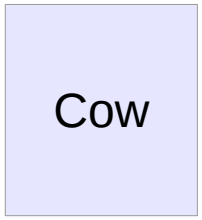
- Fast for looking up values



Hash Tables and Hash Maps

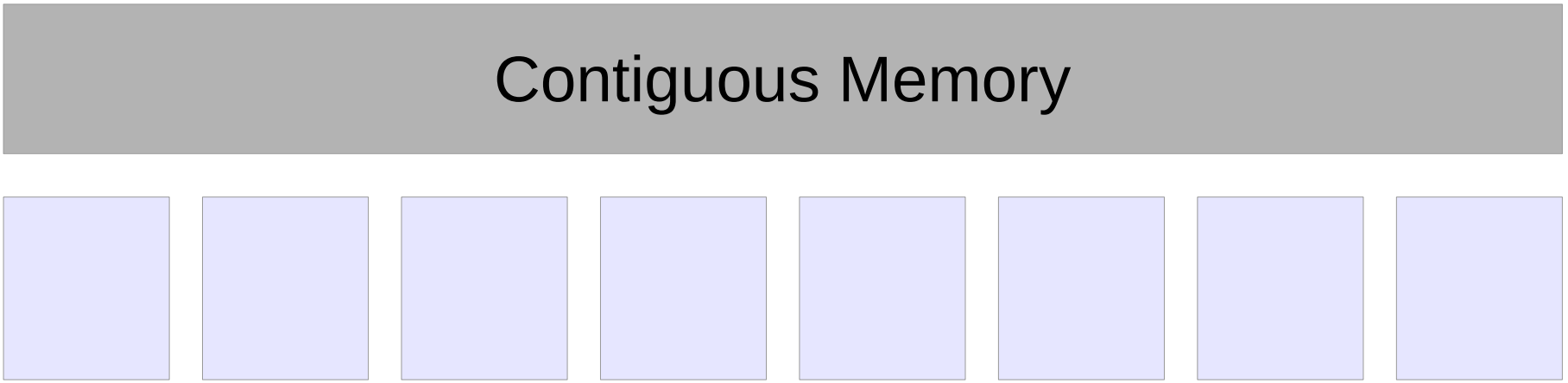
- Fast for looking up values

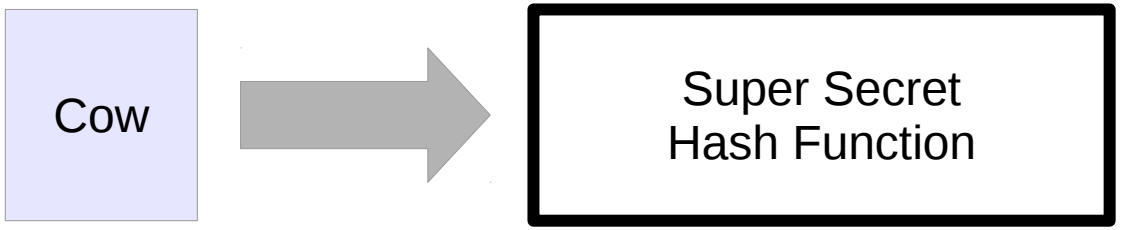




Hash Tables and Hash Maps

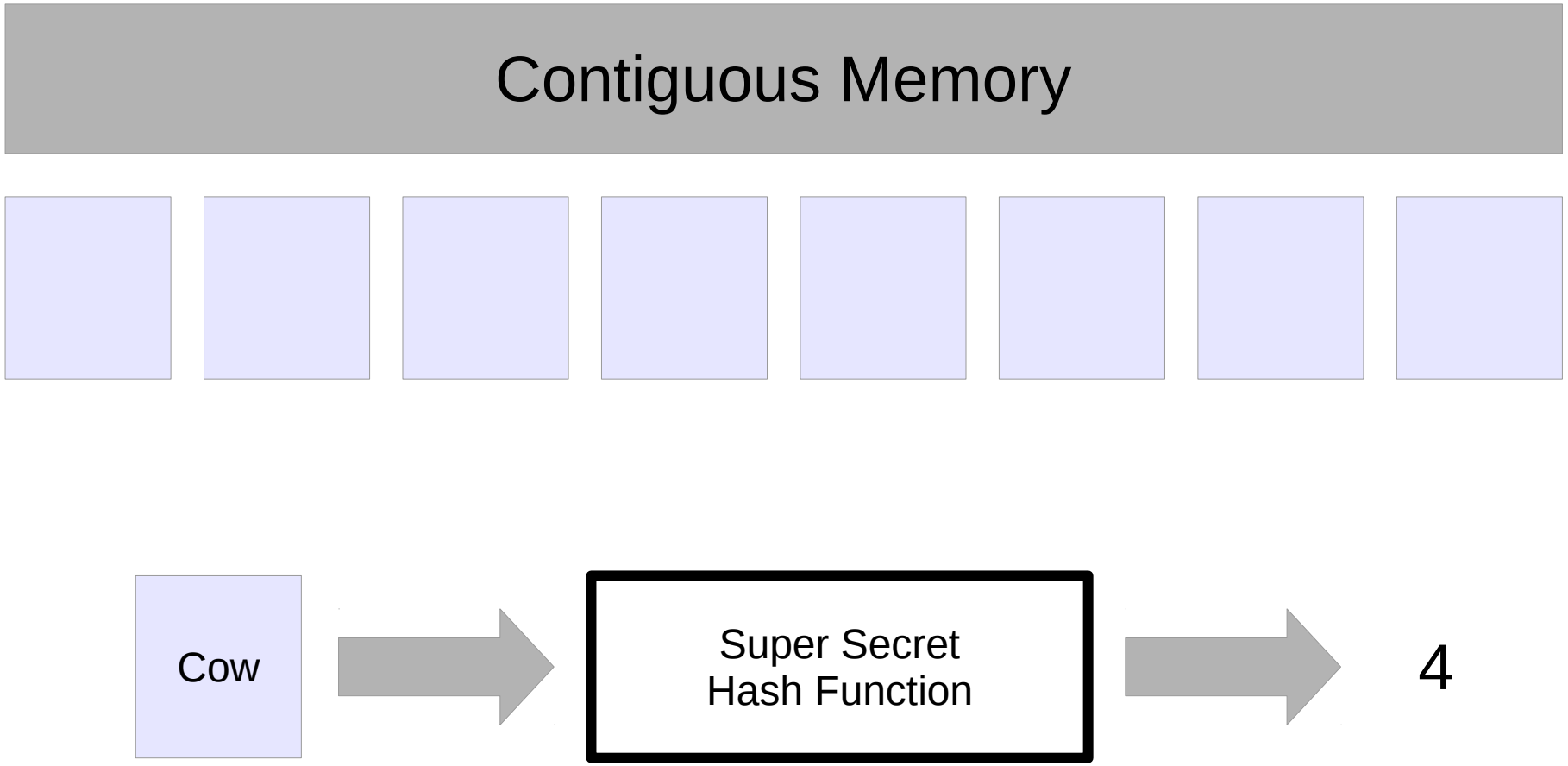
- Fast for looking up values





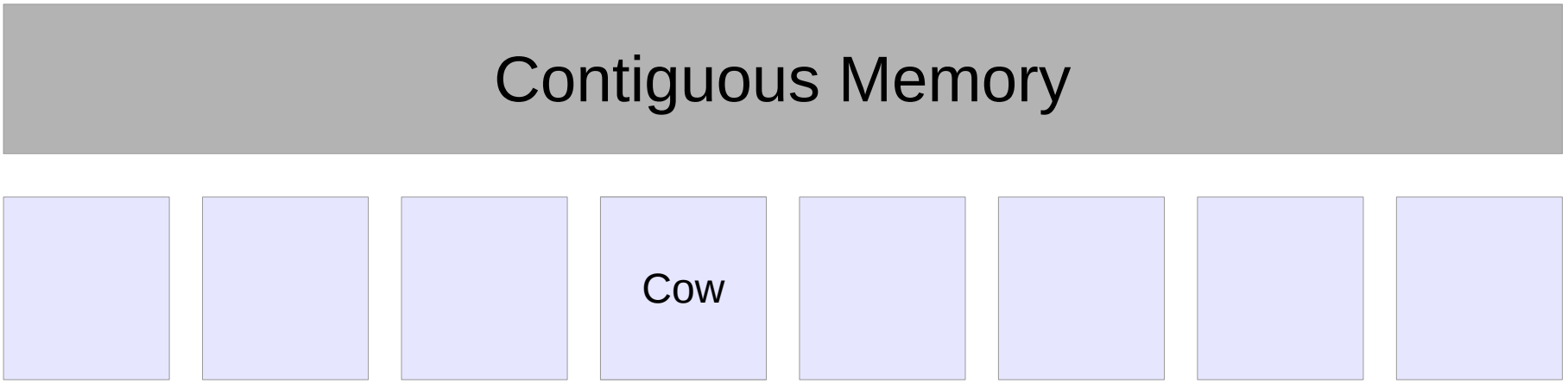
Hash Tables and Hash Maps

- Fast for looking up values



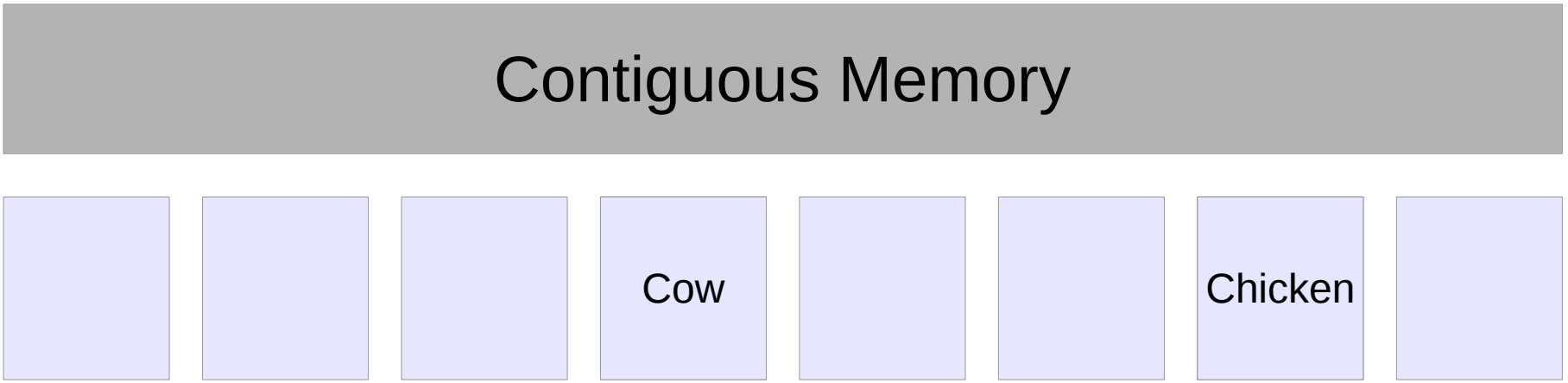
Hash Tables and Hash Maps

- Fast for looking up values



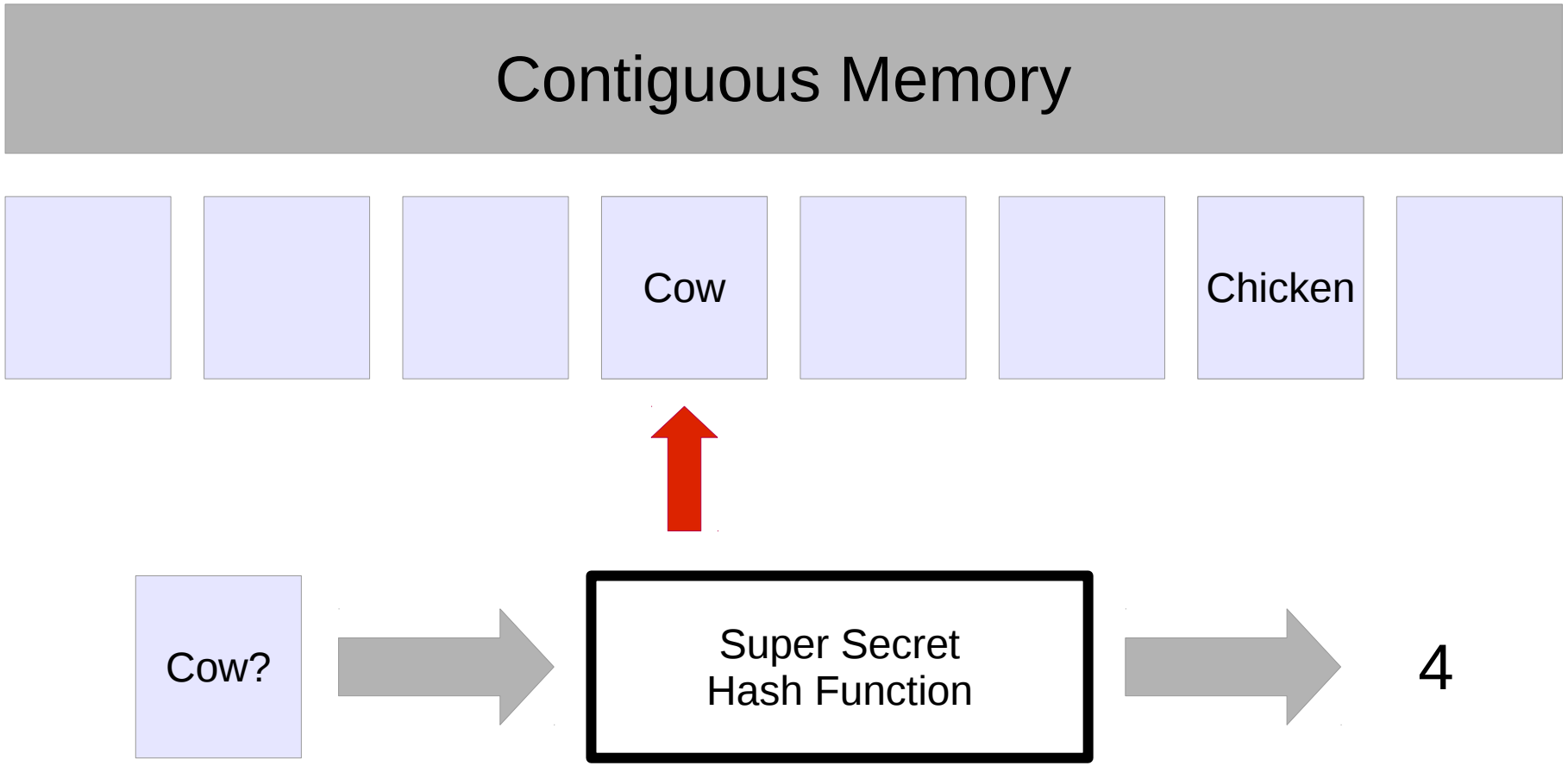
Hash Tables and Hash Maps

- Fast for looking up values



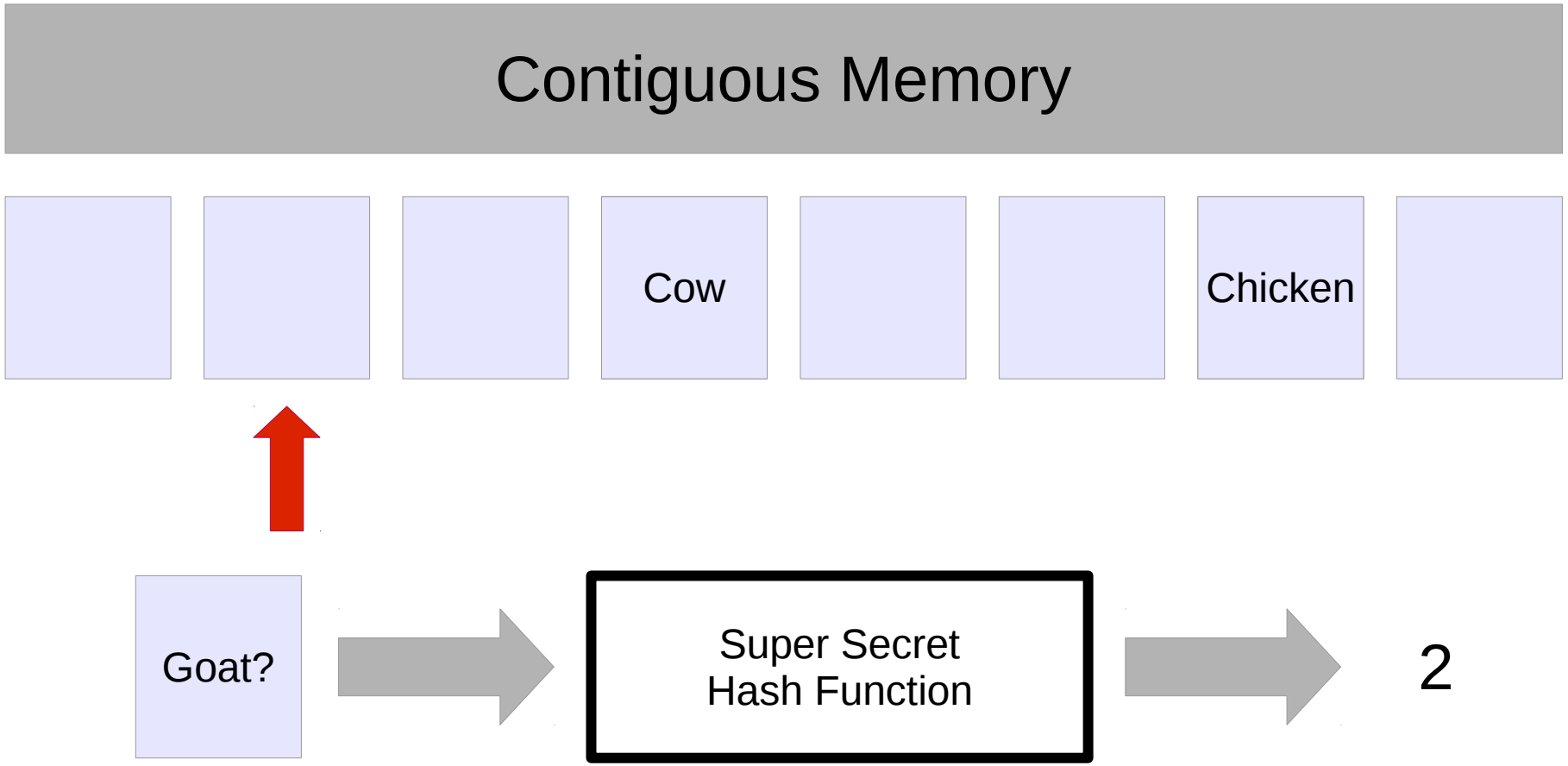
Hash Tables and Hash Maps

- Fast for looking up values



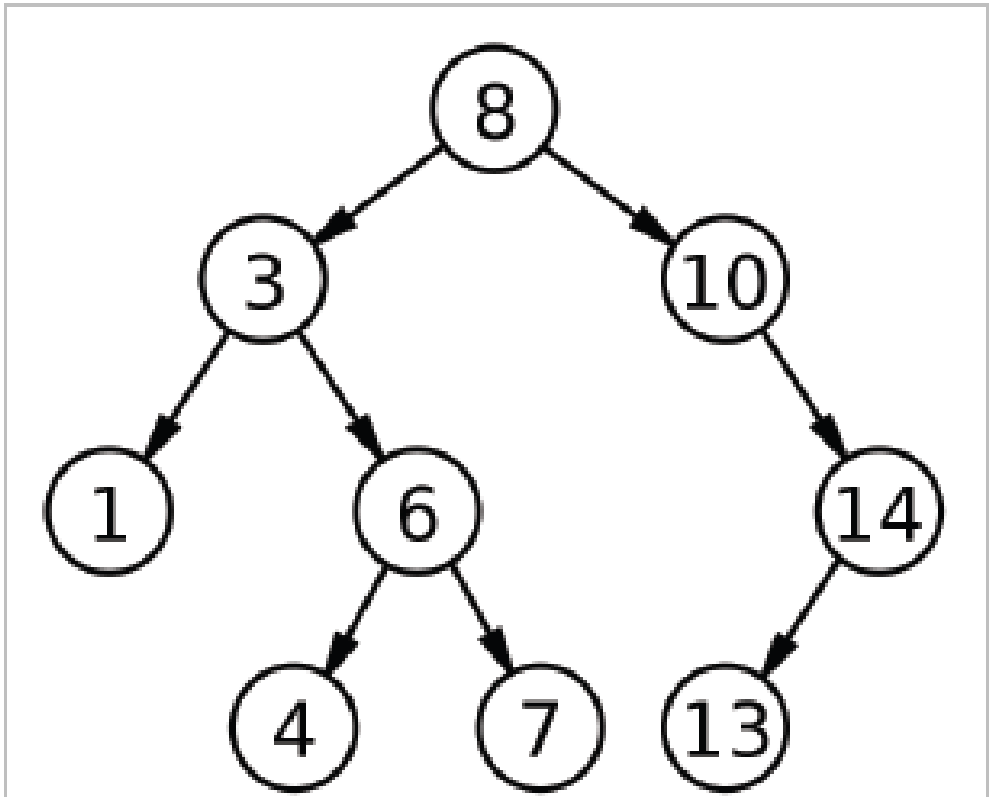
Hash Tables and Hash Maps

- Fast for looking up values



Trees

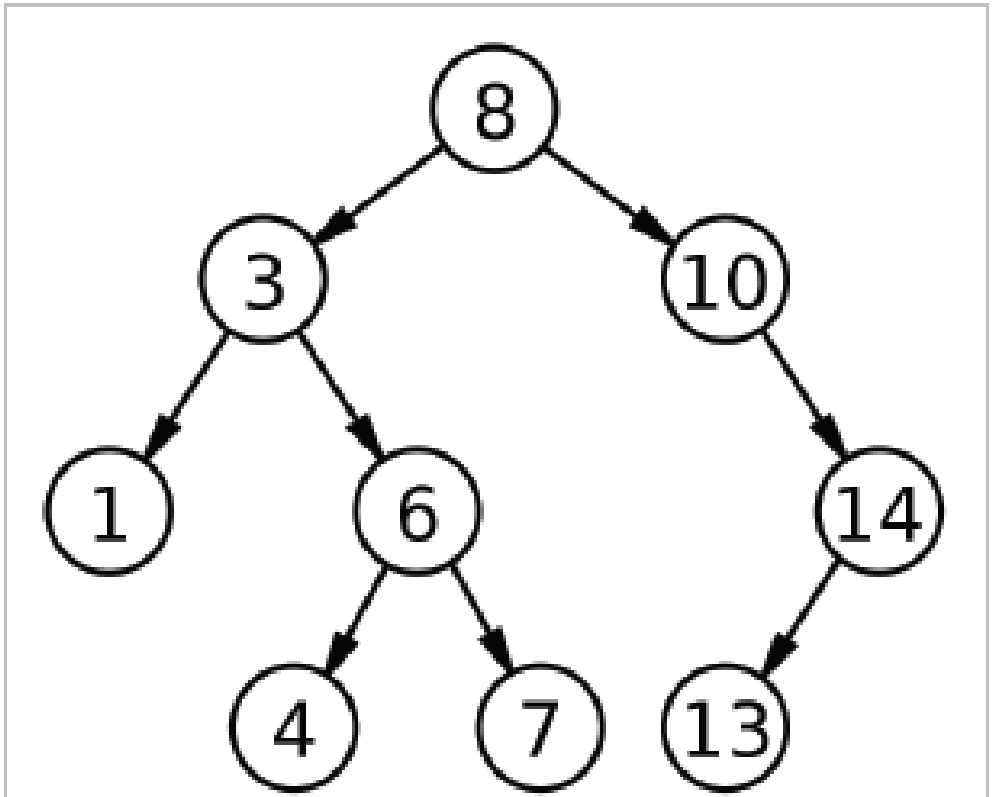
- Good for searching ranges



A binary search tree of size 9 and depth 3, with root 8 and leaves 1, 4, 7 and 13

Trees

- Good for searching ranges

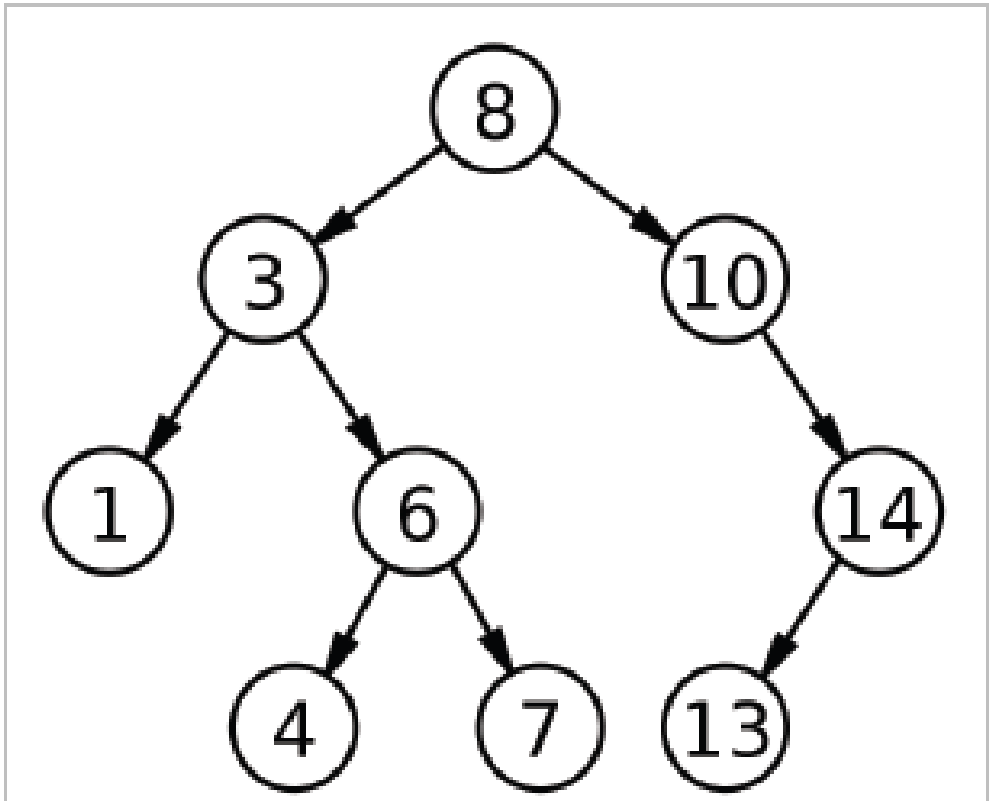


A binary search tree of size 9 and depth 3, with root 8 and leaves 1, 4, 7 and 13

Any values between 5 and 8 inclusive?

Trees

- Good for searching ranges

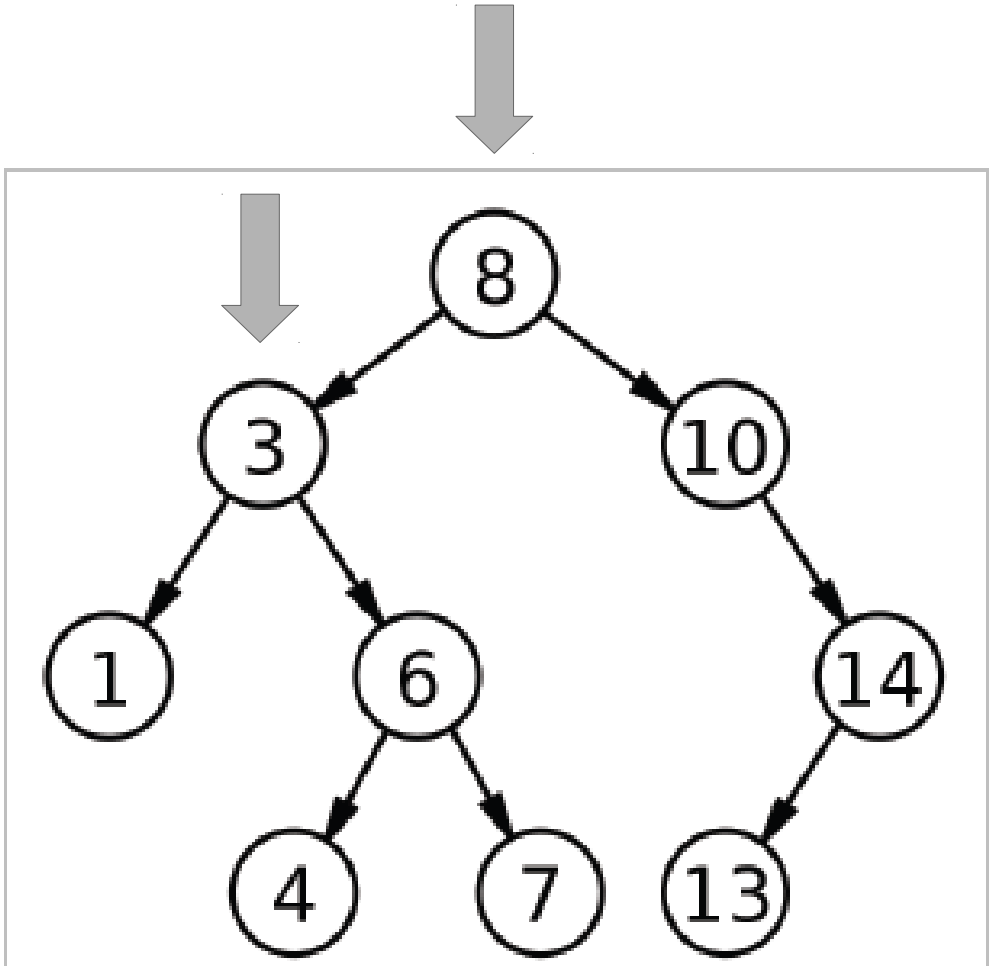


A binary search tree of size 9 and depth 3, with root 8 and leaves 1, 4, 7 and 13

Any values between 5 and 8 inclusive?

Trees

- Good for searching ranges

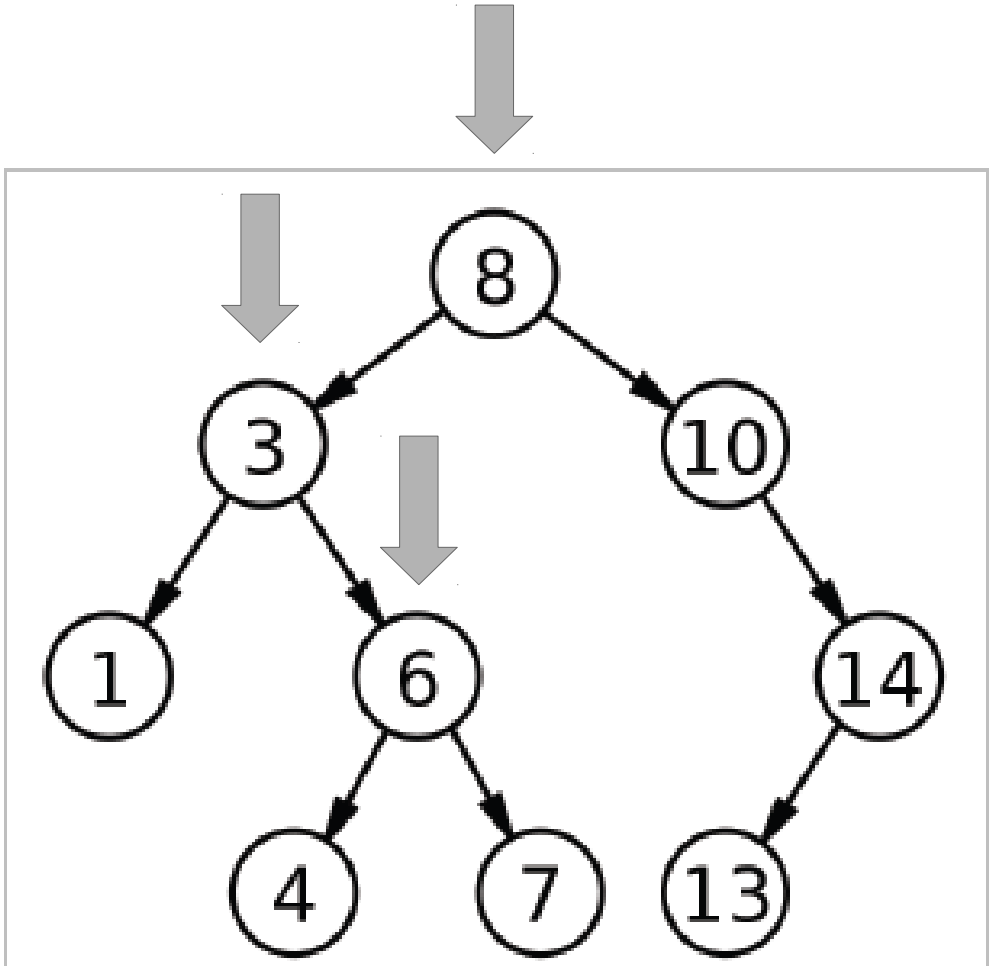


A binary search tree of size 9 and depth 3, with root 8 and leaves 1, 4, 7 and 13

Any values between 5 and 8 inclusive?

Trees

- Good for searching ranges

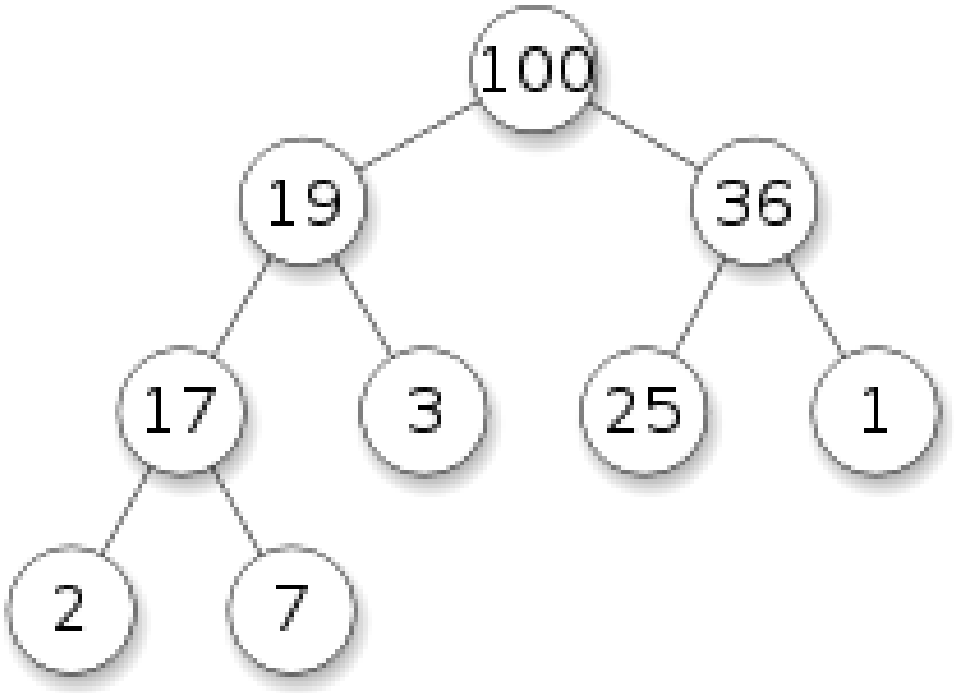


A binary search tree of size 9 and depth 3, with root 8 and leaves 1, 4, 7 and 13

Any values between 5 and 8 inclusive?

Heaps

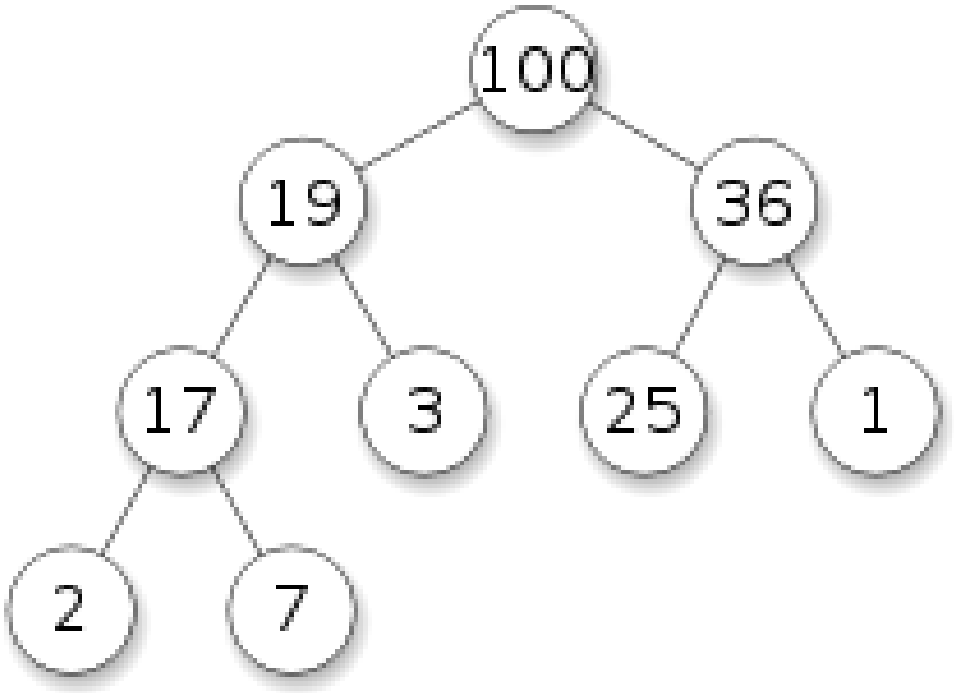
- Good for keeping track of extreme values



Heaps

- Good for keeping track of extreme values

What's the largest value?



Trees and Heaps

- Similar in structure, but different rules