Algorithms

FOURTH EDITION

ROBERT SEDGEWICK | KEVIN WAYNE

http://algs4.cs.princeton.edu

2.3 PARTITIONING DEMOS

▸ Sedgewick 2-way partitioning

▸ Dijkstra 3-way partitioning

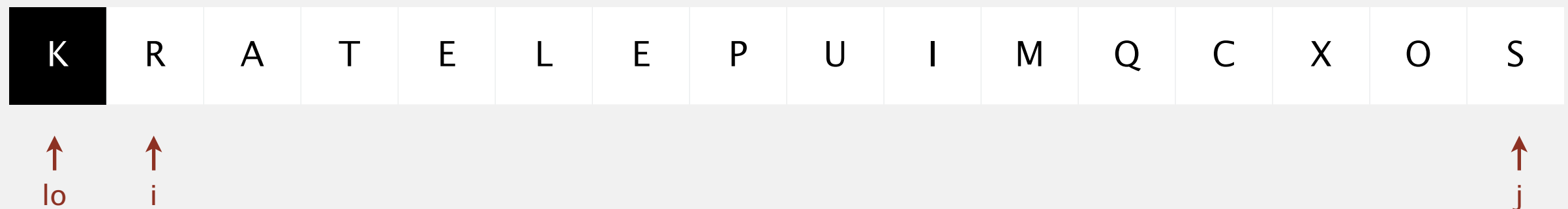▸ Bentley-McIlroy 3-way partitioning

▸ dual-pivot partitioning

2.3 PARTITIONING DEMOS

▸ **Sedgewick 2-way partitioning**

▸ Dijkstra 3-way partitioning

▸ Bentley-McIlroy 3-way partitioning

▸ dual-pivot partitioning

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

http://algs4.cs.princeton.edu

Quicksort partitioning demo

Repeat until `i` and `j` pointers cross.

- Scan `i` from left to right so long as `(a[i] < a[lo])`.
- Scan `j` from right to left so long as `(a[j] > a[lo])`.
- Exchange `a[i]` with `a[j]`.

K	R	A	T	E	L	E	P	U	I	M	Q	C	X	O	S

↑   ↑   ↑

lo   i   j

stop i scan because a[i] >= a[lo]

Quicksort partitioning demo

Repeat until `i` and `j` pointers cross.
- Scan `i` from left to right so long as `(a[i] < a[lo])`.
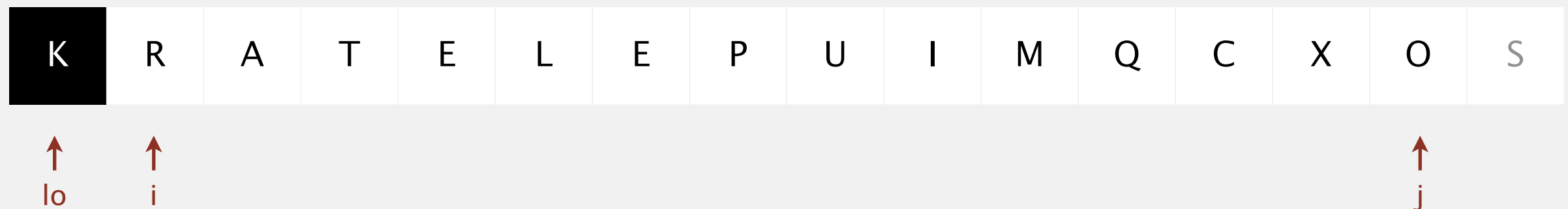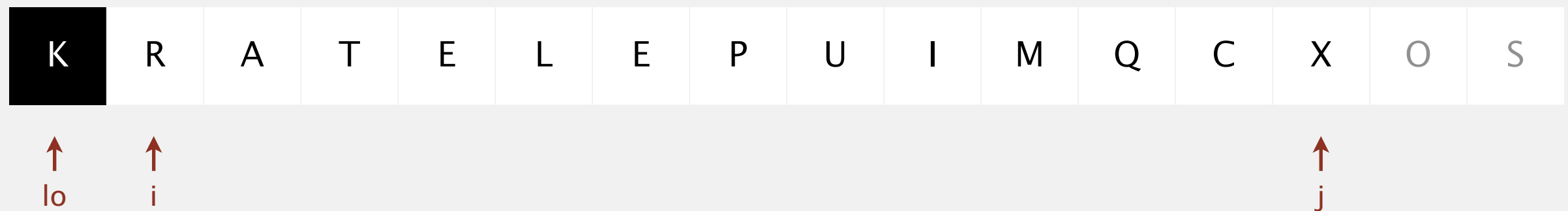- Scan `j` from right to left so long as `(a[j] > a[lo])`.
- Exchange `a[i]` with `a[j]`.

K	R	A	T	E	L	E	P	U	I	M	Q	C	X	O	S

lo   i   j

Quicksort partitioning demo

Repeat until `i` and `j` pointers cross.

- Scan `i` from left to right so long as `(a[i] < a[lo])`.
- Scan `j` from right to left so long as `(a[j] > a[lo])`.
- Exchange `a[i]` with `a[j]`.

K	R	A	T	E	L	E	P	U	I	M	Q	C	X	O	S

↑ lo   ↑ i   ↑ j

Quicksort partitioning demo

Repeat until `i` and `j` pointers cross.

- Scan `i` from left to right so long as `(a[i] < a[lo])`.
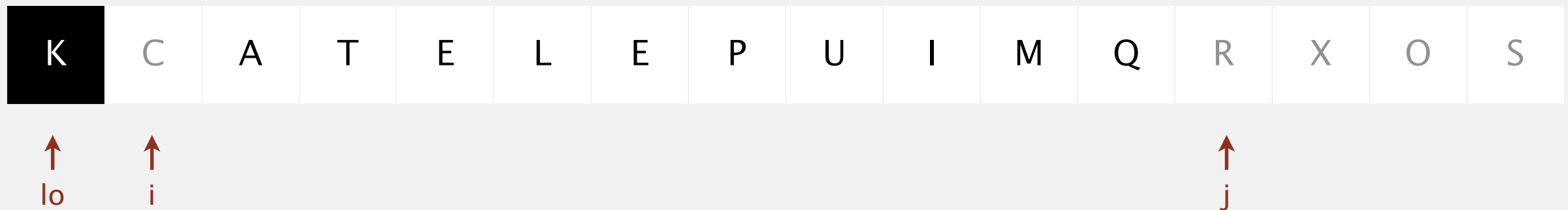- Scan `j` from right to left so long as `(a[j] > a[lo])`.
- Exchange `a[i]` with `a[j]`.

K	R	A	T	E	L	E	P	U	I	M	Q	C	X	O	S

↑   ↑   ↑

lo   i   j

stop j scan and exchange a[i] with a[j]

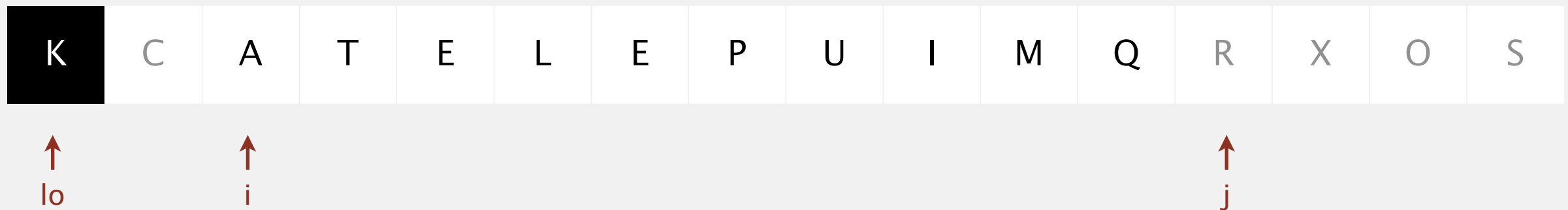# Quicksort partitioning demo

Repeat until `i` and `j` pointers cross.
- Scan `i` from left to right so long as `(a[i] < a[lo])`.
- Scan `j` from right to left so long as `(a[j] > a[lo])`.
- Exchange `a[i]` with `a[j]`.

K	C	A	T	E	L	E	P	U	I	M	Q	R	X	O	S

↑   ↑   ↑

lo   i   j

Quicksort partitioning demo

Repeat until `i` and `j` pointers cross.
- Scan `i` from left to right so long as `(a[i] < a[lo])`.
- Scan `j` from right to left so long as `(a[j] > a[lo])`.
- Exchange `a[i]` with `a[j]`.

K	C	A	T	E	L	E	P	U	I	M	Q	R	X	O	S

↑ lo   ↑ i   ↑ j

Quicksort partitioning demo

Repeat until `i` and `j` pointers cross.

- Scan `i` from left to right so long as `(a[i] < a[lo])`.
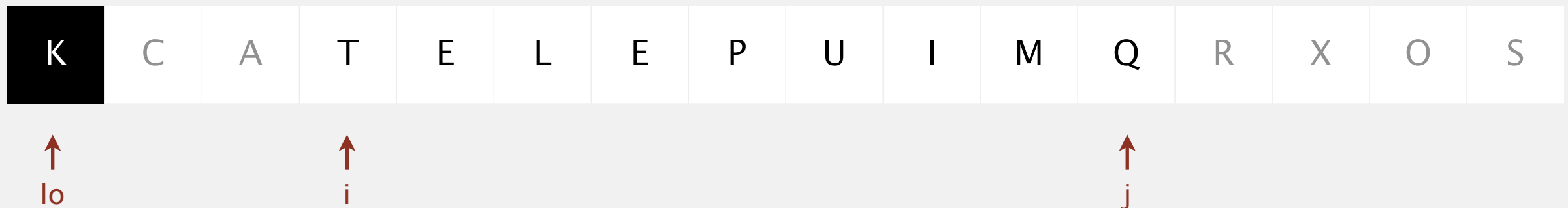- Scan `j` from right to left so long as `(a[j] > a[lo])`.
- Exchange `a[i]` with `a[j]`.

K	C	A	T	E	L	E	P	U	I	M	Q	R	X	O	S

↑   ↑   ↑

lo   i   j

stop i scan because a[i] >= a[lo]

Repeat until `i` and `j` pointers cross.

- Scan `i` from left to right so long as (`a[i] < a[lo]`).
- Scan `j` from right to left so long as (`a[j] > a[lo]`).
- Exchange `a[i]` with `a[j]`.

K	C	A	T	E	L	E	P	U	I	M	Q	R	X	O	S

↑ lo   ↑ i   ↑ j

Quicksort partitioning demo

Repeat until `i` and `j` pointers cross.
- Scan `i` from left to right so long as `(a[i] < a[lo])`.
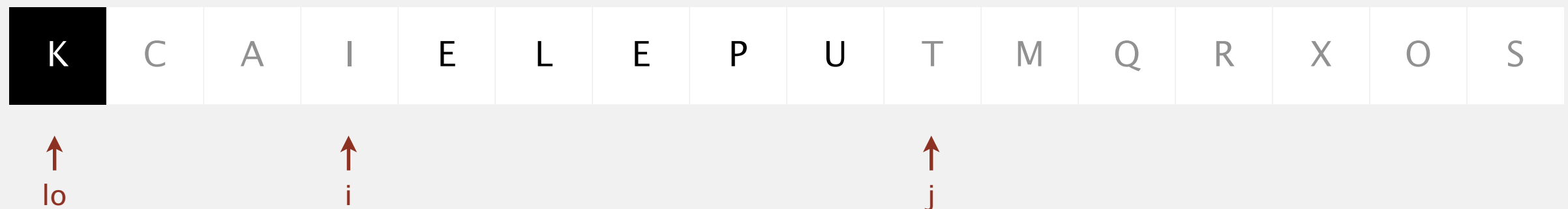- Scan `j` from right to left so long as `(a[j] > a[lo])`.
- Exchange `a[i]` with `a[j]`.

K	C	A	T	E	L	E	P	U	I	M	Q	R	X	O	S

↑ lo   ↑ i   ↑ j

Quicksort partitioning demo

Repeat until `i` and `j` pointers cross.

- Scan `i` from left to right so long as `(a[i] < a[lo])`.
- Scan `j` from right to left so long as `(a[j] > a[lo])`.
- Exchange `a[i]` with `a[j]`.

K	C	A	T	E	L	E	P	U	I	M	Q	R	X	O	S

↑ lo   ↑ i   ↑ j

stop j scan and exchange a[i] with a[j]

Quicksort partitioning demo

Repeat until `i` and `j` pointers cross.

- Scan `i` from left to right so long as `(a[i] < a[lo])`.
- Scan `j` from right to left so long as `(a[j] > a[lo])`.
- Exchange `a[i]` with `a[j]`.

K	C	A	I	E	L	E	P	U	T	M	Q	R	X	O	S

↑ lo   ↑ i   ↑ j

Quicksort partitioning demo

Repeat until `i` and `j` pointers cross.

- Scan `i` from left to right so long as `(a[i] < a[lo])`.
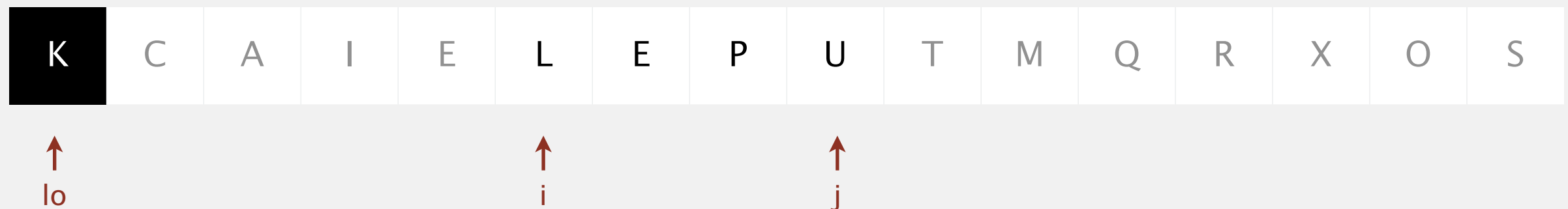- Scan `j` from right to left so long as `(a[j] > a[lo])`.
- Exchange `a[i]` with `a[j]`.

K	C	A	I	E	L	E	P	U	T	M	Q	R	X	O	S

↑
lo

↑
i

↑
j

Quicksort partitioning demo

Repeat until `i` and `j` pointers cross.

- Scan `i` from left to right so long as `(a[i] < a[lo])`.
- Scan `j` from right to left so long as `(a[j] > a[lo])`.
- Exchange `a[i]` with `a[j]`.

| K | C | A | I | E | L | E | P | U | T | M | Q | R | X | O | S |

↑
lo

↑
i

↑
j

stop i scan because a[i] >= a[lo]

Repeat until `i` and `j` pointers cross.

- Scan `i` from left to right so long as (`a[i] < a[lo]`).
- Scan `j` from right to left so long as (`a[j] > a[lo]`).
- Exchange `a[i]` with `a[j]`.

K	C	A	I	E	L	E	P	U	T	M	Q	R	X	O	S

↑   ↑   ↑

lo   i   j

Quicksort partitioning demo

Repeat until `i` and `j` pointers cross.

- Scan `i` from left to right so long as (`a[i] < a[lo]`).
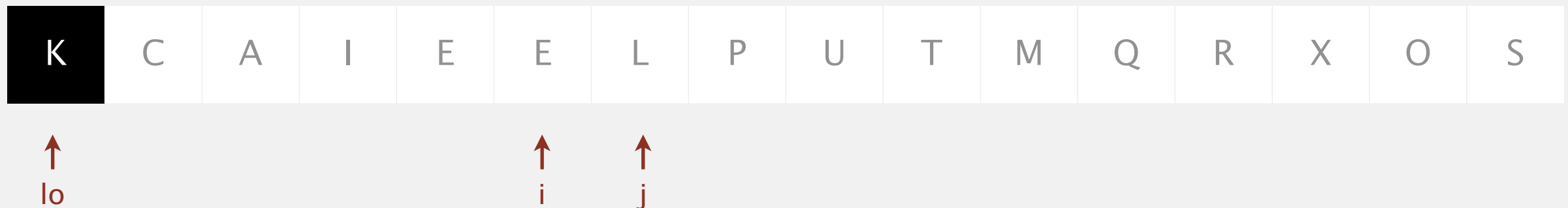- Scan `j` from right to left so long as (`a[j] > a[lo]`).
- Exchange `a[i]` with `a[j]`.

K	C	A	I	E	L	E	P	U	T	M	Q	R	X	O	S

↑
lo   ↑   ↑
                                      i   j

Quicksort partitioning demo

Repeat until `i` and `j` pointers cross.
- Scan `i` from left to right so long as `(a[i] < a[lo])`.
- Scan `j` from right to left so long as `(a[j] > a[lo])`.
- Exchange `a[i]` with `a[j]`.

K	C	A	I	E	L	E	P	U	T	M	Q	R	X	O	S

↑
lo

↑
i

↑
j

stop j scan and exchange a[i] with a[j]

Quicksort partitioning demo

Repeat until `i` and `j` pointers cross.
- Scan `i` from left to right so long as `(a[i] < a[lo])`.
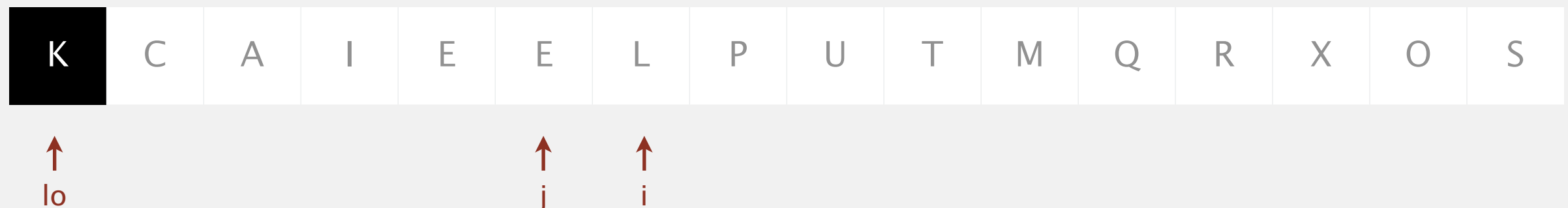- Scan `j` from right to left so long as `(a[j] > a[lo])`.
- Exchange `a[i]` with `a[j]`.

K	C	A	I	E	E	L	P	U	T	M	Q	R	X	O	S

↑
lo   ↑   ↑
                                 i   j

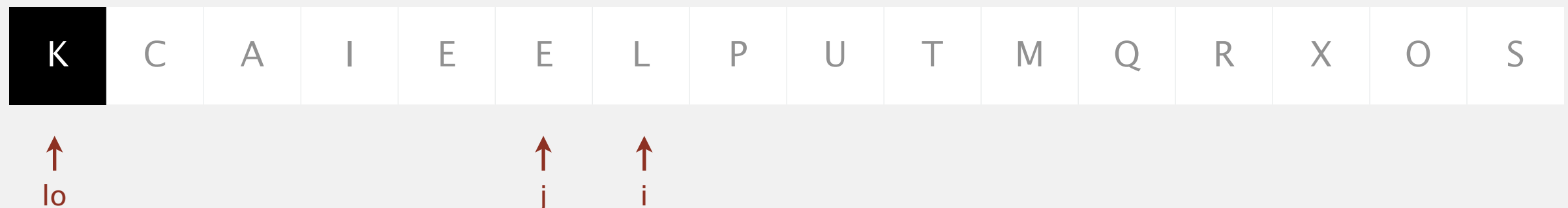# Quicksort partitioning demo

Repeat until `i` and `j` pointers cross.

- Scan `i` from left to right so long as `(a[i] < a[lo])`.
- Scan `j` from right to left so long as `(a[j] > a[lo])`.
- Exchange `a[i]` with `a[j]`.

K	C	A	I	E	E	L	P	U	T	M	Q	R	X	O	S

↑
lo

       ↑ ↑
       i j

stop i scan because a[i] >= a[lo]

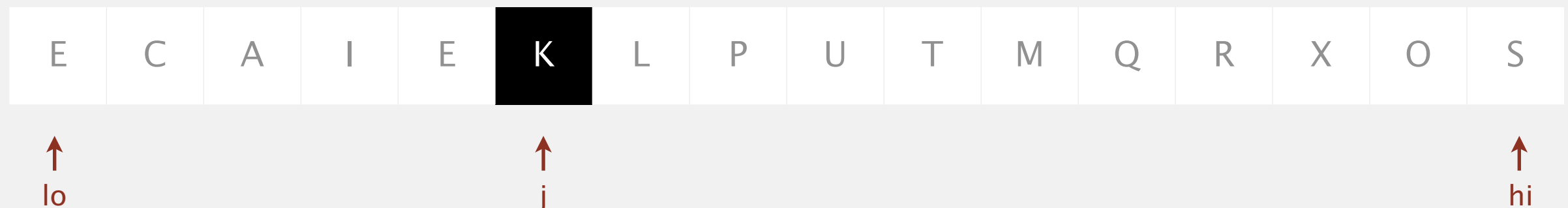# Quicksort partitioning demo

Repeat until `i` and `j` pointers cross.
- Scan `i` from left to right so long as `(a[i] < a[lo])`.
- Scan `j` from right to left so long as `(a[j] > a[lo])`.
- Exchange `a[i]` with `a[j]`.

K	C	A	I	E	E	L	P	U	T	M	Q	R	X	O	S

↑
lo

↑
j

↑
i

stop j scan because a[j] <= a[lo]

Quicksort partitioning demo

Repeat until `i` and `j` pointers cross.
- Scan `i` from left to right so long as (a[i] < a[lo]).
- Scan `j` from right to left so long as (a[j] > a[lo]).
- Exchange a[i] with a[j].
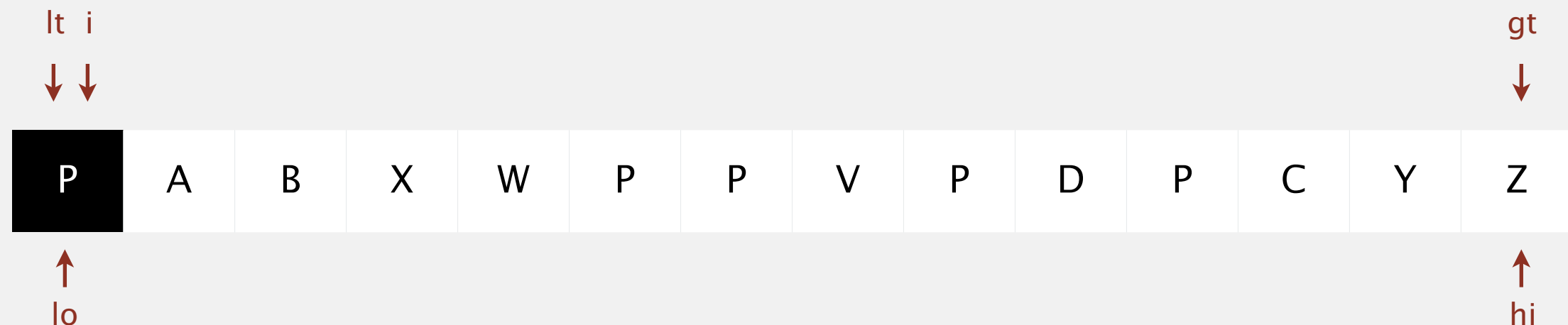
When pointers cross.
- Exchange a[lo] with a[j].

K	C	A	I	E	E	L	P	U	T	M	Q	R	X	O	S

↑
lo

↑ ↑
j   i

pointers cross: exchange a[lo] with a[j]

Quicksort partitioning demo

Repeat until `i` and `j` pointers cross.

- Scan `i` from left to right so long as `(a[i] < a[lo])`.
- Scan `j` from right to left so long as `(a[j] > a[lo])`.
- Exchange `a[i]` with `a[j]`.

When pointers cross.

- Exchange `a[lo]` with `a[j]`.

E	C	A	I	E	K	L	P	U	T	M	Q	R	X	O	S

↑
lo

↑
j

↑
hi

partitioned!

2.3 PARTITIONING DEMOS

▸ *Sedgewick 2-way partitioning*

▸ **Dijkstra 3-way partitioning**

▸ *Bentley-McIlroy 3-way partitioning*

▸ *dual-pivot partitioning*

Algorithms

Robert Sedgewick | Kevin Wayne
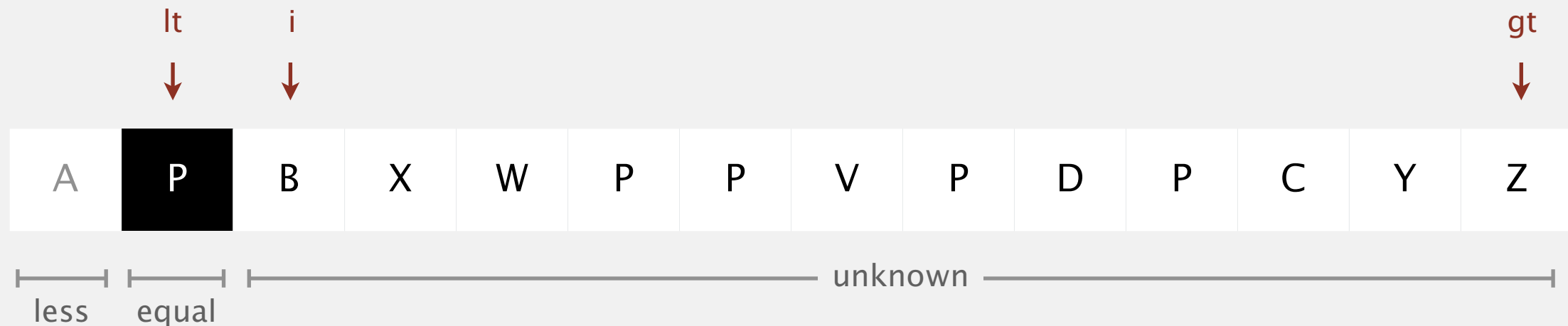
http://algs4.cs.princeton.edu

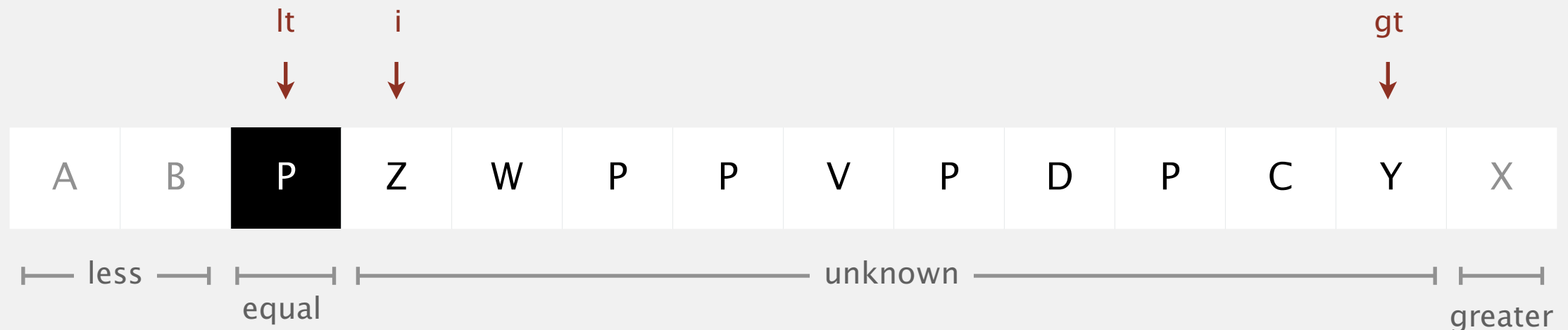# Dijkstra 3-way partitioning demo

- Let `v` be partitioning item `a[lo]`.
- Scan `i` from left to right.
  - `(a[i] < v)`: exchange `a[lt]` with `a[i]`; increment both `lt` and `i`
  - `(a[i] > v)`: exchange `a[gt]` with `a[i]`; decrement `gt`
  - `(a[i] == v)`: increment `i`

lt i													gt
↓ ↓													↓
P	A	B	X	W	P	P	V	P	D	P	C	Y	Z

lo   hi

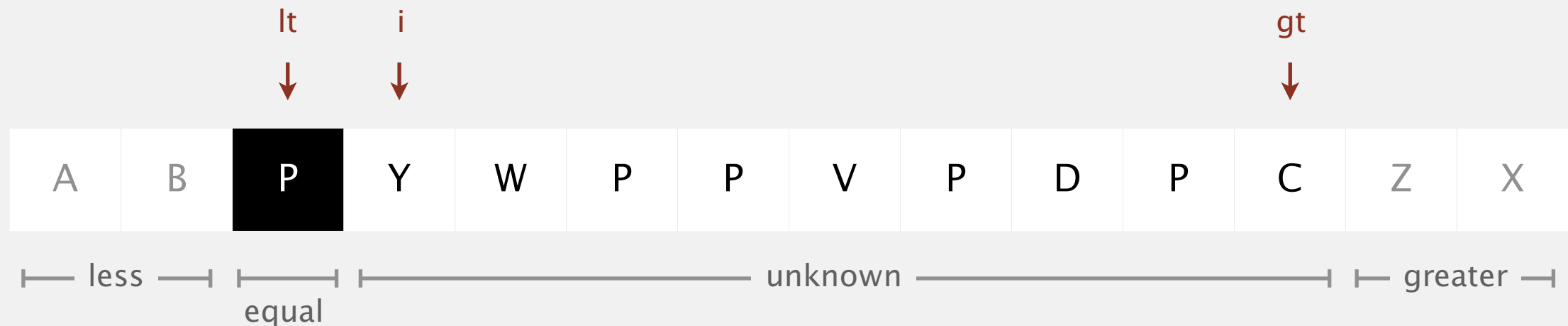# Dijkstra 3-way partitioning demo

- Let `v` be partitioning item `a[lo]`.
- Scan `i` from left to right.
  - `(a[i]  < v)`: exchange `a[lt]` with `a[i]`; increment both `lt` and `i`
  - `(a[i]  > v)`: exchange `a[gt]` with `a[i]`; decrement `gt`
  - `(a[i] == v)`: increment `i`

Dijkstra 3-way partitioning demo

- Let `v` be partitioning item `a[lo]`.
- Scan `i` from left to right.
  - `(a[i]  < v)`: exchange `a[lt]` with `a[i]`; increment both `lt` and `i`
  - `(a[i]  > v)`: exchange `a[gt]` with `a[i]`; decrement `gt`
  - `(a[i] == v)`: increment `i`

lt   i   gt

| A | P | B | X | W | P | P | V | P | D | P | C | Y | Z |

less   equal   unknown

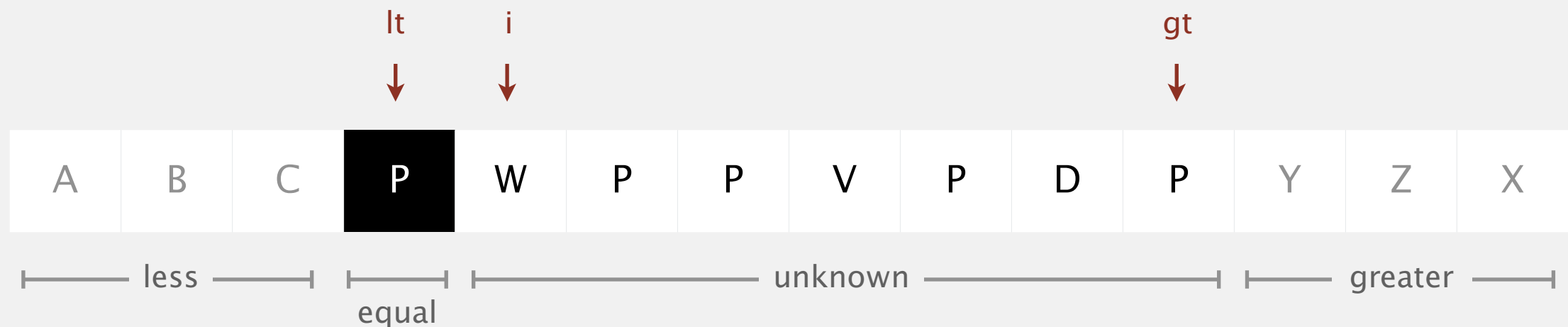# Dijkstra 3-way partitioning demo

- Let `v` be partitioning item `a[lo]`.
- Scan `i` from left to right.
  - `(a[i]  < v)`: exchange `a[lt]` with `a[i]`; increment both `lt` and `i`
  - `(a[i]  > v)`: exchange `a[gt]` with `a[i]`; decrement `gt`
  - `(a[i] == v)`: increment `i`

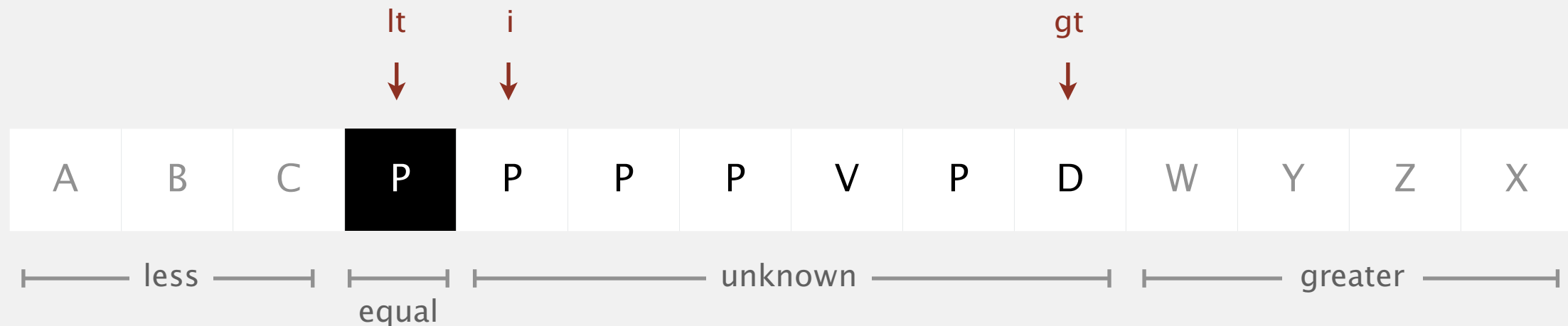# Dijkstra 3-way partitioning demo

- Let `v` be partitioning item `a[lo]`.
- Scan `i` from left to right.
  - `(a[i]  < v)`: exchange `a[lt]` with `a[i]`; increment both `lt` and `i`
  - `(a[i]  > v)`: exchange `a[gt]` with `a[i]`; decrement `gt`
  - `(a[i] == v)`: increment `i`

Dijkstra 3-way partitioning demo

- Let `v` be partitioning item `a[lo]`.
- Scan `i` from left to right.
  - (`a[i] < v`): exchange `a[lt]` with `a[i]`; increment both `lt` and `i`
  - (`a[i] > v`): exchange `a[gt]` with `a[i]`; decrement `gt`
  - (`a[i] == v`): increment `i`

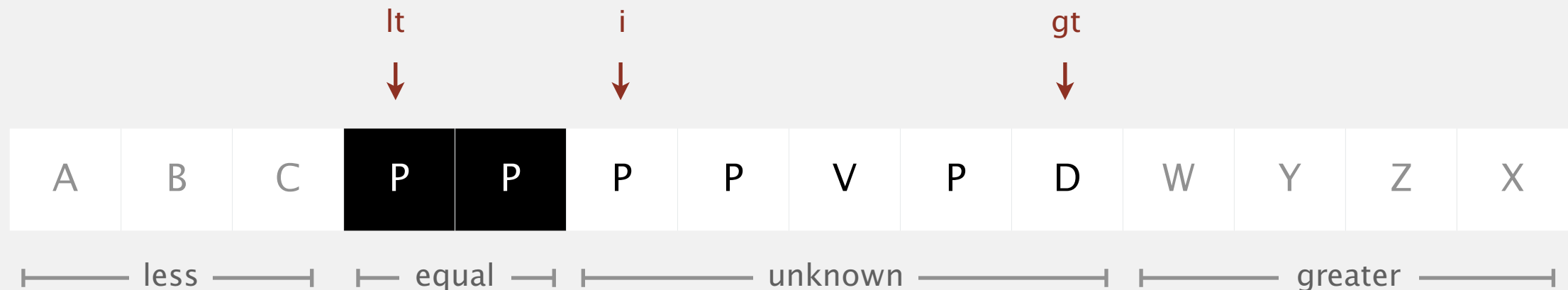# Dijkstra 3-way partitioning demo

- Let `v` be partitioning item `a[lo]`.
- Scan `i` from left to right.
  - `(a[i]  < v)`: exchange `a[lt]` with `a[i]`; increment both `lt` and `i`
  - `(a[i]  > v)`: exchange `a[gt]` with `a[i]`; decrement `gt`
  - `(a[i] == v)`: increment `i`

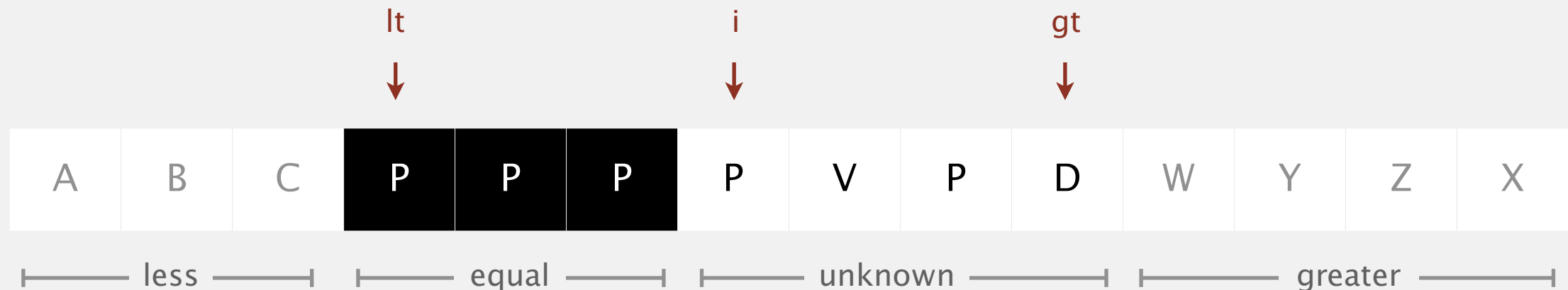# Dijkstra 3-way partitioning demo

- Let `v` be partitioning item `a[lo]`.
- Scan `i` from left to right.
  - `(a[i]  < v)`: exchange `a[lt]` with `a[i]`; increment both `lt` and `i`
  - `(a[i]  > v)`: exchange `a[gt]` with `a[i]`; decrement `gt`
  - `(a[i] == v)`: increment `i`

Dijkstra 3-way partitioning demo

- Let `v` be partitioning item `a[lo]`.
- Scan `i` from left to right.
  - `(a[i] < v)`: exchange `a[lt]` with `a[i]`; increment both `lt` and `i`
  - `(a[i] > v)`: exchange `a[gt]` with `a[i]`; decrement `gt`
  - `(a[i] == v)`: increment `i`

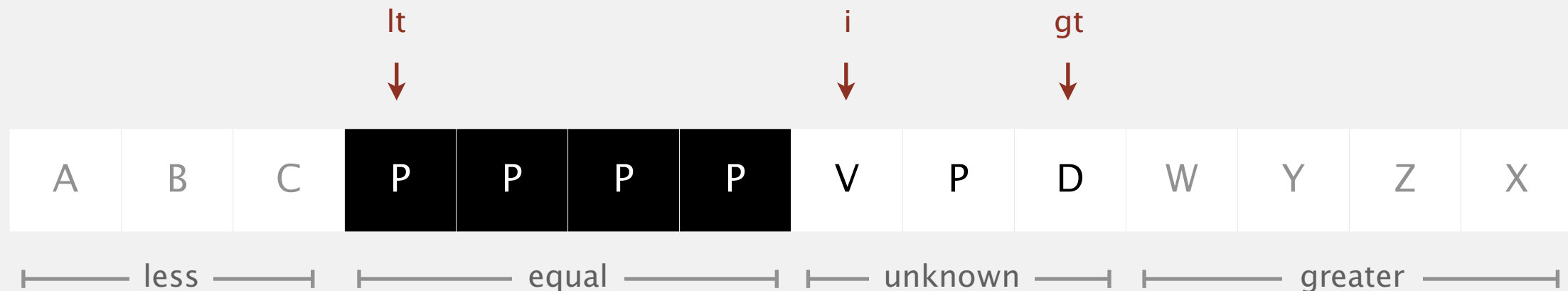# Dijkstra 3-way partitioning demo

- Let `v` be partitioning item `a[lo]`.
- Scan `i` from left to right.
  - `(a[i]  < v)`: exchange `a[lt]` with `a[i]`; increment both `lt` and `i`
  - `(a[i]  > v)`: exchange `a[gt]` with `a[i]`; decrement `gt`
  - `(a[i] == v)`: increment `i`

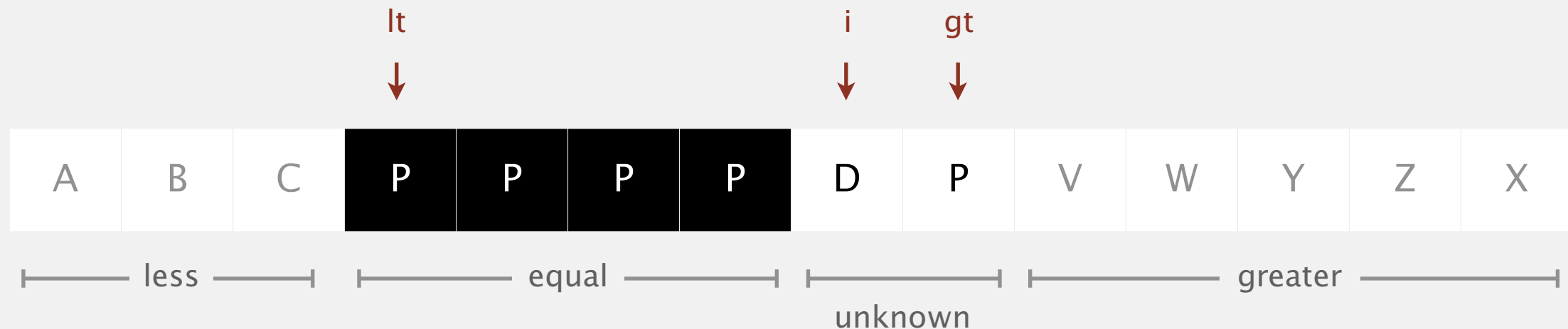# Dijkstra 3-way partitioning demo

- Let `v` be partitioning item `a[lo]`.
- Scan `i` from left to right.
  - `(a[i]  < v)`: exchange `a[lt]` with `a[i]`; increment both `lt` and `i`
  - `(a[i]  > v)`: exchange `a[gt]` with `a[i]`; decrement `gt`
  - `(a[i] == v)`: increment `i`

Dijkstra 3-way partitioning demo

- Let `v` be partitioning item `a[lo]`.
- Scan `i` from left to right.
  - `(a[i]  < v)`: exchange `a[lt]` with `a[i]`; increment both `lt` and `i`
  - `(a[i]  > v)`: exchange `a[gt]` with `a[i]`; decrement `gt`
  - `(a[i] == v)`: increment `i`

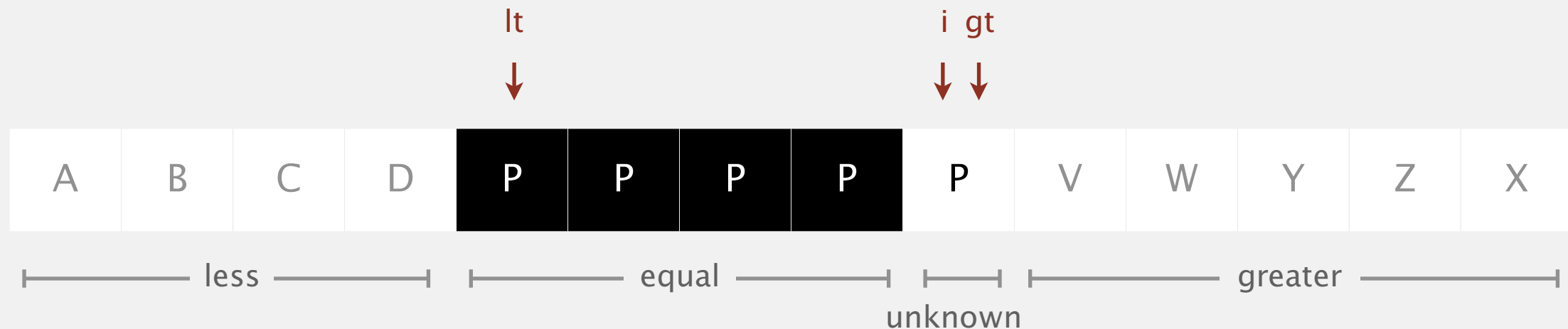# Dijkstra 3-way partitioning demo

- Let `v` be partitioning item `a[lo]`.
- Scan `i` from left to right.
  - `(a[i]  < v)`: exchange `a[lt]` with `a[i]`; increment both `lt` and `i`
  - `(a[i]  > v)`: exchange `a[gt]` with `a[i]`; decrement `gt`
  - `(a[i] == v)`: increment `i`

- Let `v` be partitioning item `a[lo]`.
- Scan `i` from left to right.
  - `(a[i]  < v)`: exchange `a[lt]` with `a[i]`; increment both `lt` and `i`
  - `(a[i]  > v)`: exchange `a[gt]` with `a[i]`; decrement `gt`
  - `(a[i] == v)`: increment `i`

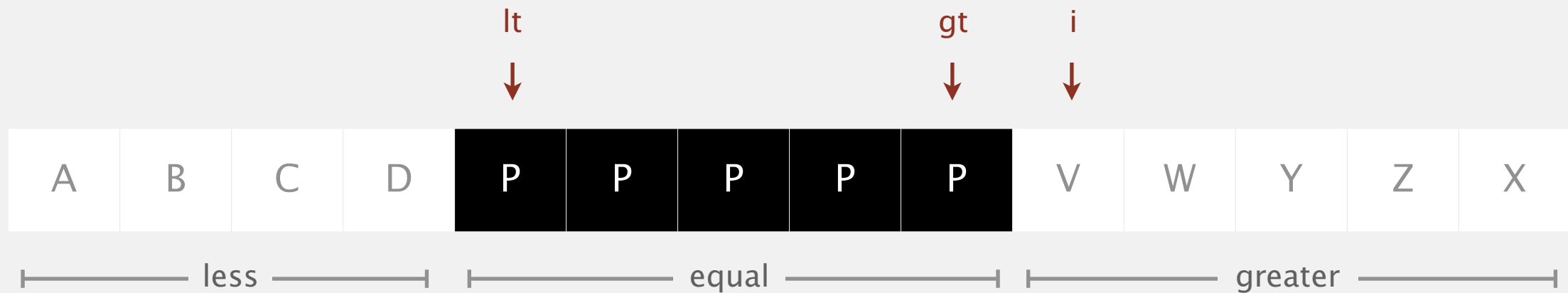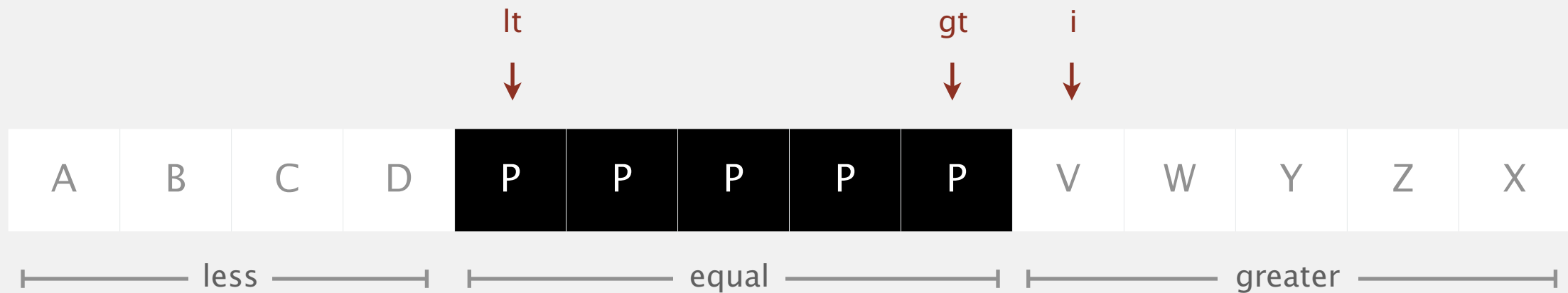# Dijkstra 3-way partitioning demo

- Let `v` be partitioning item `a[lo]`.
- Scan `i` from left to right.
  - `(a[i]  < v)`: exchange `a[lt]` with `a[i]`; increment both `lt` and `i`
  - `(a[i]  > v)`: exchange `a[gt]` with `a[i]`; decrement `gt`
  - `(a[i] == v)`: increment `i`

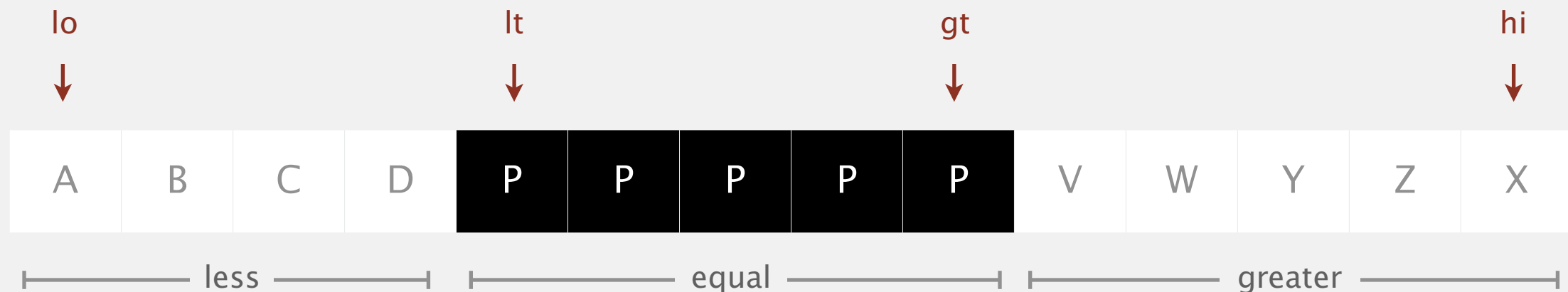# Dijkstra 3-way partitioning demo

- Let `v` be partitioning item `a[lo]`.
- Scan `i` from left to right.
  - `(a[i]  < v)`: exchange `a[lt]` with `a[i]`; increment both `lt` and `i`
  - `(a[i]  > v)`: exchange `a[gt]` with `a[i]`; decrement `gt`
  - `(a[i] == v)`: increment `i`

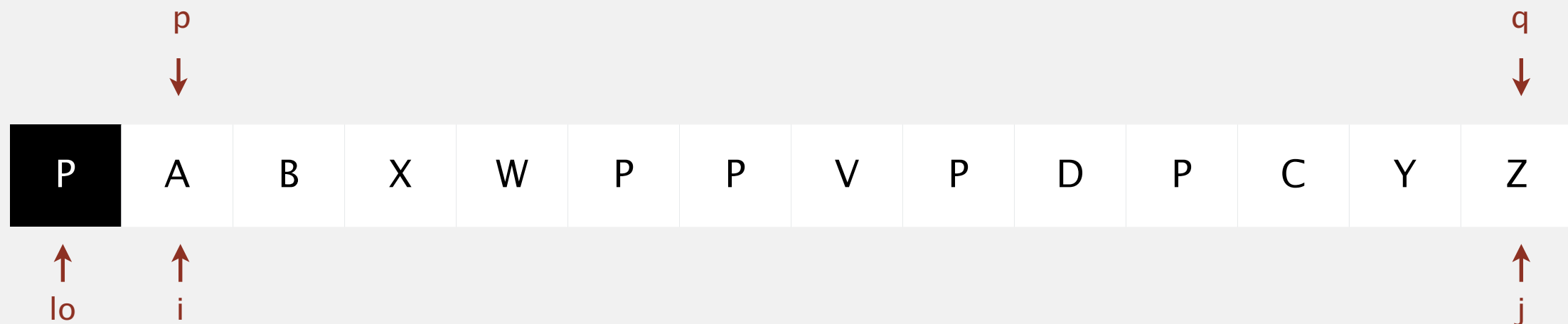# 2.3 PARTITIONING DEMOS

▸ *Sedgewick 2-way partitioning*

▸ *Dijkstra 3-way partitioning*

▸ *Bentley-McIlroy 3-way partitioning*

▸ *dual-pivot partitioning*

Bentley-McIlroy 3-way partitioning demo

Phase I. Repeat until `i` and `j` pointers cross.

- Scan `i` from left to right so long as `(a[i] < a[lo])`.
- Scan `j` from right to left so long as `(a[j] > a[lo])`.
- Exchange `a[i]` with `a[j]`.
- If `(a[i] == a[lo])`, exchange `a[i]` with `a[p]` and increment `p`.
- If `(a[j] == a[lo])`, exchange `a[j]` with `a[q]` and decrement `q`.

p													q
P	A	B	X	W	P	P	V	P	D	P	C	Y	Z
lo	i												j

Phase I. Repeat until `i` and `j` pointers cross.

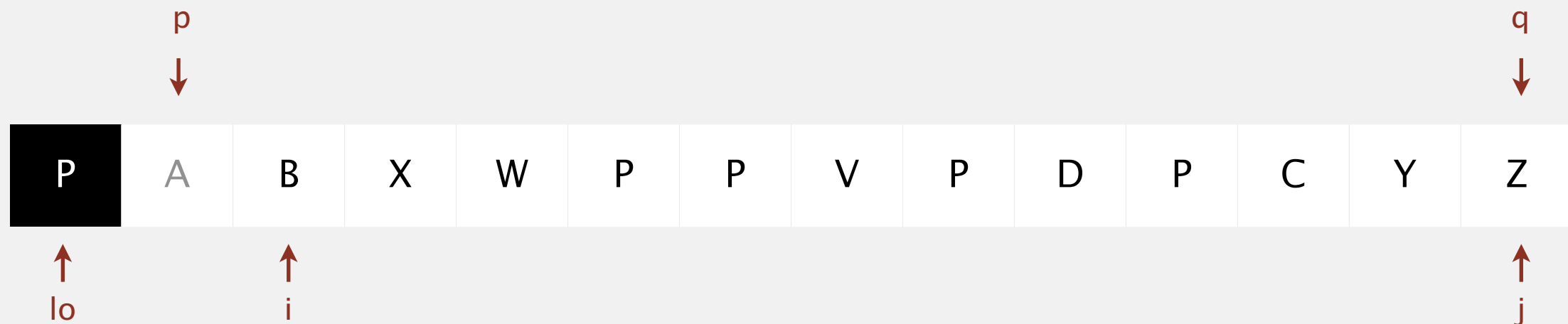- Scan `i` from left to right so long as (`a[i] < a[lo]`).
- Scan `j` from right to left so long as (`a[j] > a[lo]`).
- Exchange `a[i]` with `a[j]`.
- If (`a[i] == a[lo]`), exchange `a[i]` with `a[p]` and increment `p`.
- If (`a[j] == a[lo]`), exchange `a[j]` with `a[q]` and decrement `q`.

| P | A | B | X | W | P | P | V | P | D | P | C | Y | Z |

Bentley-McIlroy 3-way partitioning demo

Phase I. Repeat until `i` and `j` pointers cross.

- Scan `i` from left to right so long as (`a[i] < a[lo]`).
- Scan `j` from right to left so long as (`a[j] > a[lo]`).
- Exchange `a[i]` with `a[j]`.
- If (`a[i] == a[lo]`), exchange `a[i]` with `a[p]` and increment `p`.
- If (`a[j] == a[lo]`), exchange `a[j]` with `a[q]` and decrement `q`.

Bentley-McIlroy 3-way partitioning demo

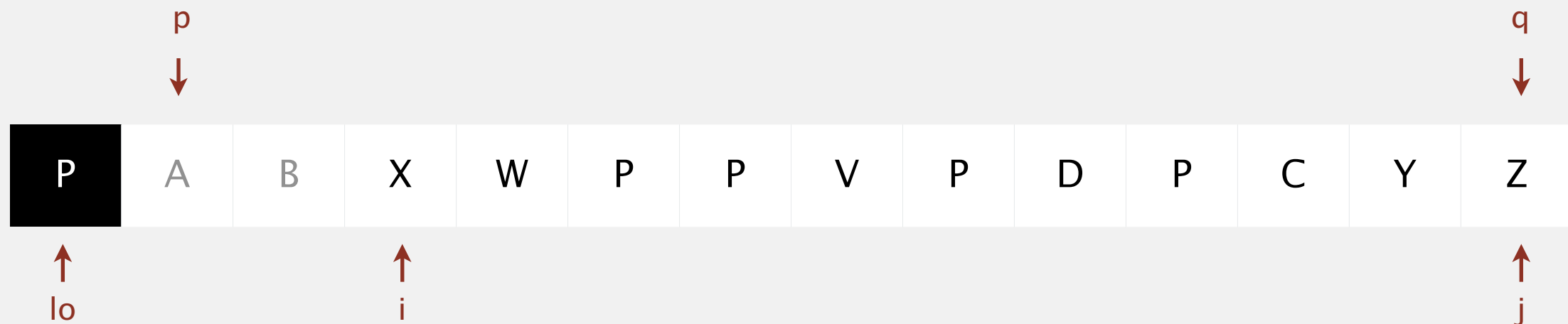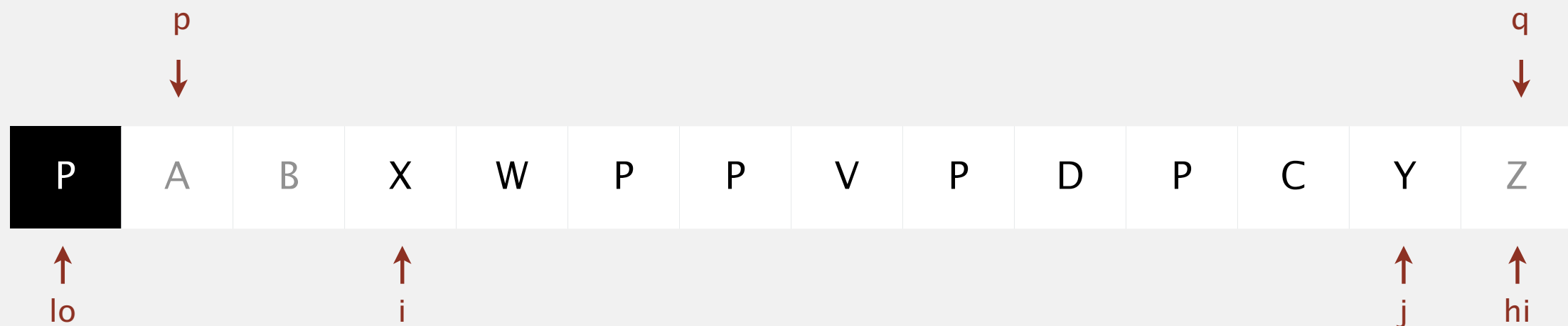Phase I. Repeat until `i` and `j` pointers cross.

- Scan `i` from left to right so long as (`a[i] < a[lo]`).
- Scan `j` from right to left so long as (`a[j] > a[lo]`).
- Exchange `a[i]` with `a[j]`.
- If (`a[i] == a[lo]`), exchange `a[i]` with `a[p]` and increment `p`.
- If (`a[j] == a[lo]`), exchange `a[j]` with `a[q]` and decrement `q`.

P	A	B	X	W	P	P	V	P	D	P	C	Y	Z

p → (above A)
q → (above Z)

lo ↑ (below P)
i ↑ (below X)
j ↑ (below Y)
hi ↑ (below Z)

Bentley-McIlroy 3-way partitioning demo

Phase I. Repeat until `i` and `j` pointers cross.

- Scan `i` from left to right so long as (`a[i] < a[lo]`).
- Scan `j` from right to left so long as (`a[j] > a[lo]`).
- Exchange `a[i]` with `a[j]`.
- If (`a[i] == a[lo]`), exchange `a[i]` with `a[p]` and increment `p`.
- If (`a[j] == a[lo]`), exchange `a[j]` with `a[q]` and decrement `q`.

exchange a[i] with a[j]

Bentley-McIlroy 3-way partitioning demo

Phase I. Repeat until `i` and `j` pointers cross.

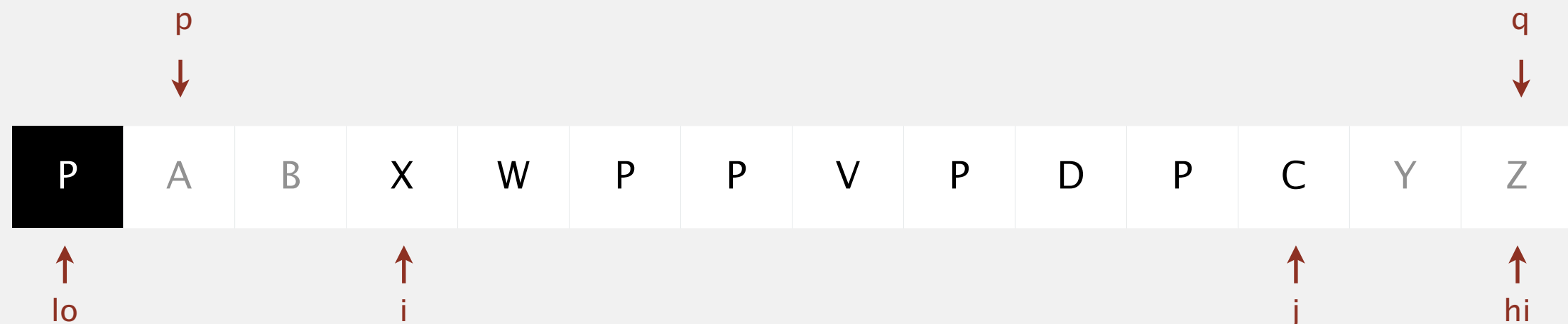- Scan `i` from left to right so long as (`a[i] < a[lo]`).
- Scan `j` from right to left so long as (`a[j] > a[lo]`).
- Exchange `a[i]` with `a[j]`.
- If (`a[i] == a[lo]`), exchange `a[i]` with `a[p]` and increment `p`.
- If (`a[j] == a[lo]`), exchange `a[j]` with `a[q]` and decrement `q`.

Bentley-McIlroy 3-way partitioning demo

Phase I. Repeat until `i` and `j` pointers cross.

- Scan `i` from left to right so long as (`a[i]` < `a[lo]`).
- Scan `j` from right to left so long as (`a[j]` > `a[lo]`).
- Exchange `a[i]` with `a[j]`.
- If (`a[i]` == `a[lo]`), exchange `a[i]` with `a[p]` and increment `p`.
- If (`a[j]` == `a[lo]`), exchange `a[j]` with `a[q]` and decrement `q`.

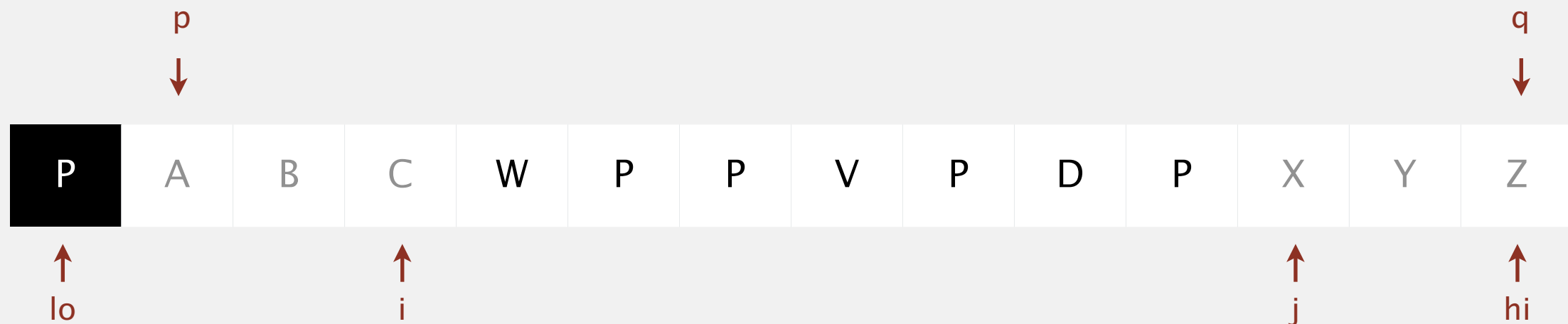| p											q
P	A	B	C	W	P	P	V	P	D	P	X	Y	Z
lo				i							j		hi

Bentley-McIlroy 3-way partitioning demo

Phase I. Repeat until `i` and `j` pointers cross.

- Scan `i` from left to right so long as (`a[i]` < `a[lo]`).
- Scan `j` from right to left so long as (`a[j]` > `a[lo]`).
- Exchange `a[i]` with `a[j]`.
- If (`a[i]` == `a[lo]`), exchange `a[i]` with `a[p]` and increment `p`.
- If (`a[j]` == `a[lo]`), exchange `a[j]` with `a[q]` and decrement `q`.

exchange `a[i]` with `a[j]`

Bentley-McIlroy 3-way partitioning demo

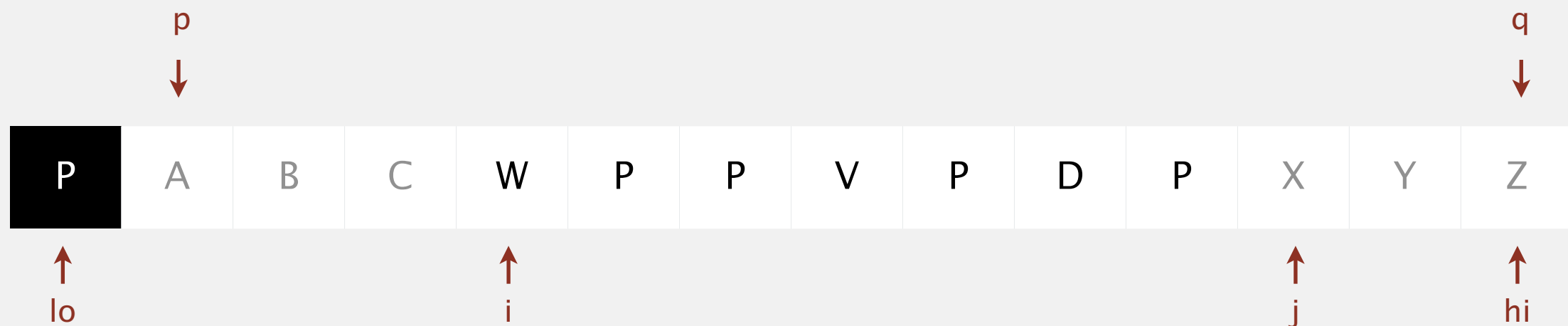Phase I. Repeat until `i` and `j` pointers cross.

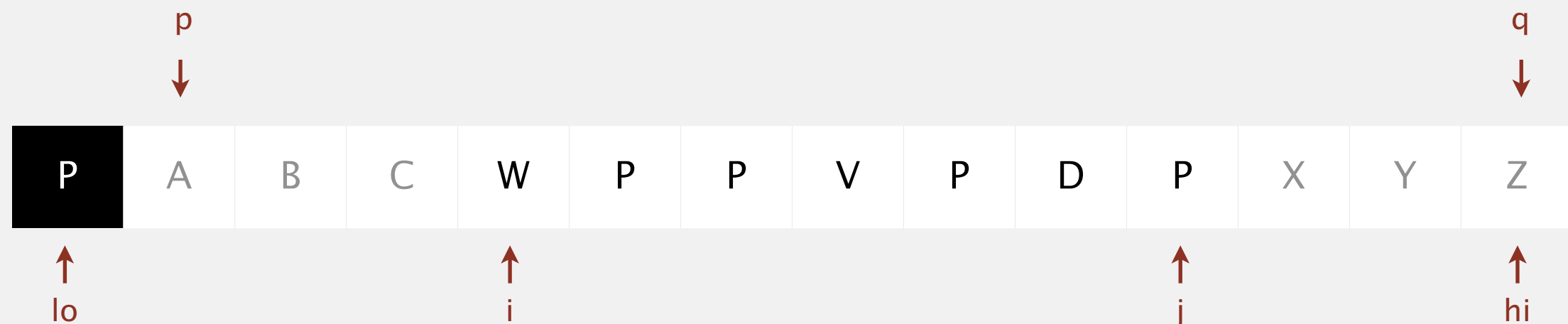- Scan `i` from left to right so long as `(a[i] < a[lo])`.
- Scan `j` from right to left so long as `(a[j] > a[lo])`.
- Exchange `a[i]` with `a[j]`.
- If `(a[i] == a[lo])`, exchange `a[i]` with `a[p]` and increment `p`.
- If `(a[j] == a[lo])`, exchange `a[j]` with `a[q]` and decrement `q`.

exchange a[i] with a[p] and increment p

Bentley-McIlroy 3-way partitioning demo

Phase I. Repeat until `i` and `j` pointers cross.

- Scan `i` from left to right so long as (`a[i] < a[lo]`).
- Scan `j` from right to left so long as (`a[j] > a[lo]`).
- Exchange `a[i]` with `a[j]`.
- If (`a[i] == a[lo]`), exchange `a[i]` with `a[p]` and increment `p`.
- If (`a[j] == a[lo]`), exchange `a[j]` with `a[q]` and decrement `q`.

P	P	B	C	A	P	P	V	P	D	W	X	Y	Z

p → (above position 2)
q → (above position 13)

lo ↑ (position 0)
i ↑ (position 4)
j ↑ (position 10)
hi ↑ (position 13)
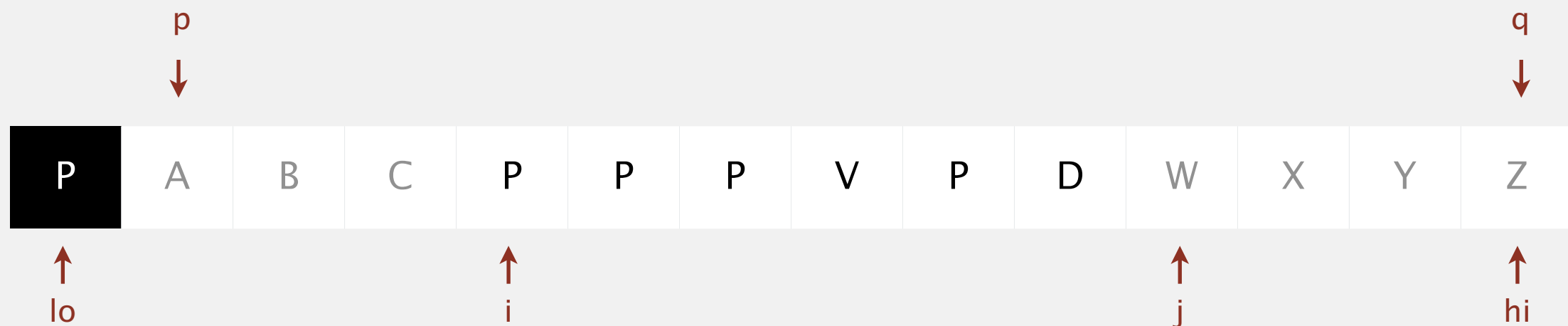
Bentley-McIlroy 3-way partitioning demo

Phase I. Repeat until `i` and `j` pointers cross.

- Scan `i` from left to right so long as (`a[i] < a[lo]`).
- Scan `j` from right to left so long as (`a[j] > a[lo]`).
- Exchange `a[i]` with `a[j]`.
- If (`a[i] == a[lo]`), exchange `a[i]` with `a[p]` and increment `p`.
- If (`a[j] == a[lo]`), exchange `a[j]` with `a[q]` and decrement `q`.
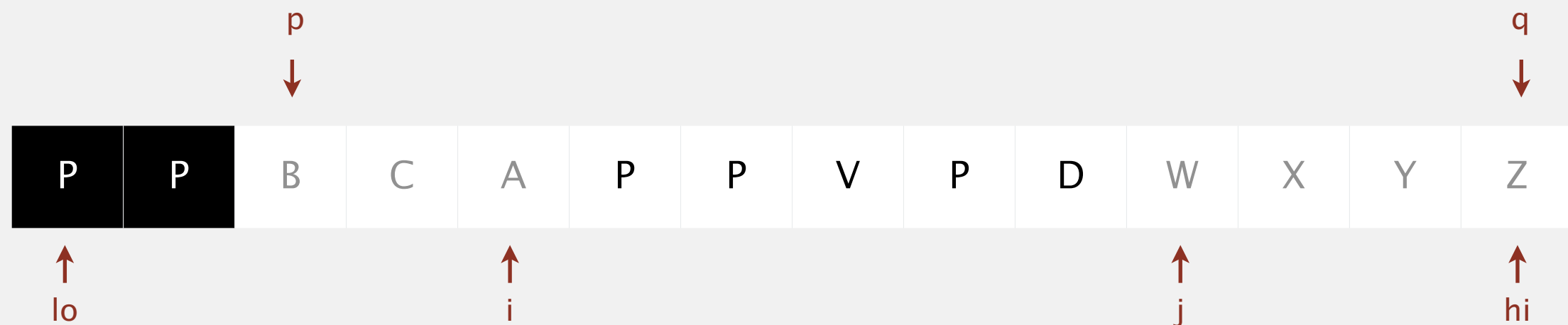
p													q
P	P	B	C	A	P	P	V	P	D	W	X	Y	Z

lo   i   j   hi

Bentley-McIlroy 3-way partitioning demo

Phase I. Repeat until `i` and `j` pointers cross.

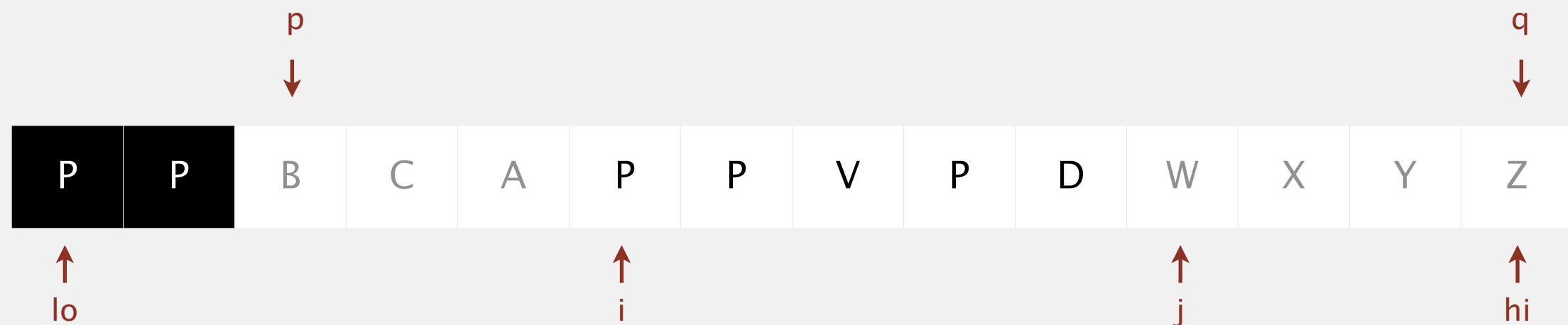- Scan `i` from left to right so long as (`a[i] < a[lo]`).
- Scan `j` from right to left so long as (`a[j] > a[lo]`).
- Exchange `a[i]` with `a[j]`.
- If (`a[i] == a[lo]`), exchange `a[i]` with `a[p]` and increment `p`.
- If (`a[j] == a[lo]`), exchange `a[j]` with `a[q]` and decrement `q`.

exchange `a[i]` with `a[j]`

Bentley-McIlroy 3-way partitioning demo

Phase I. Repeat until `i` and `j` pointers cross.
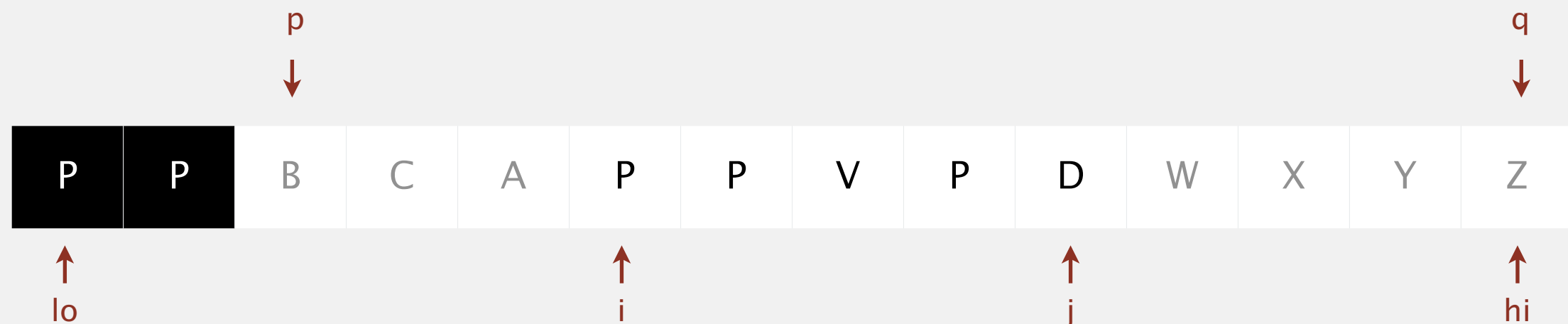- Scan `i` from left to right so long as `(a[i] < a[lo])`.
- Scan `j` from right to left so long as `(a[j] > a[lo])`.
- Exchange `a[i]` with `a[j]`.
- If `(a[i] == a[lo])`, exchange `a[i]` with `a[p]` and increment `p`.
- If `(a[j] == a[lo])`, exchange `a[j]` with `a[q]` and decrement `q`.

	p								q	
P	P	B	C	A	D	P	V	P	P	W	X	Y	Z
lo				i				j					hi

exchange a[j] with a[q] and decrement q

Phase I. Repeat until `i` and `j` pointers cross.

- Scan `i` from left to right so long as (`a[i] < a[lo]`).
- Scan `j` from right to left so long as (`a[j] > a[lo]`).
- Exchange `a[i]` with `a[j]`.
- If (`a[i] == a[lo]`), exchange `a[i]` with `a[p]` and increment `p`.
- If (`a[j] == a[lo]`), exchange `a[j]` with `a[q]` and decrement `q`.

p   q

P   P   B   C   A   D   P   V   P   Z   W   X   Y   P
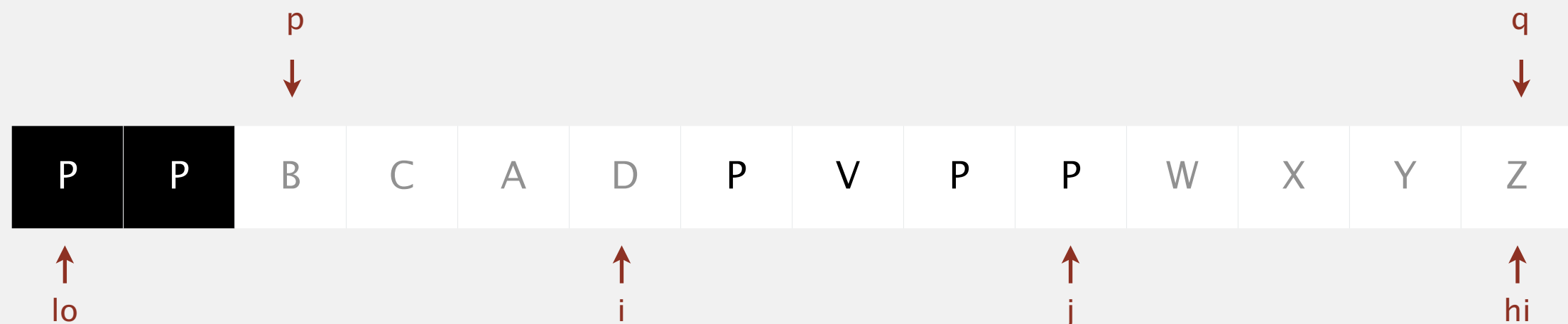
lo   i   j   hi

Bentley-McIlroy 3-way partitioning demo

Phase I. Repeat until `i` and `j` pointers cross.

- Scan `i` from left to right so long as (`a[i]` `<` `a[lo]`).
- Scan `j` from right to left so long as (`a[j]` `>` `a[lo]`).
- Exchange `a[i]` with `a[j]`.
- If (`a[i]` `==` `a[lo]`), exchange `a[i]` with `a[p]` and increment `p`.
- If (`a[j]` `==` `a[lo]`), exchange `a[j]` with `a[q]` and decrement `q`.

Phase I. Repeat until `i` and `j` pointers cross.

- Scan `i` from left to right so long as (`a[i] < a[lo]`).
- Scan `j` from right to left so long as (`a[j] > a[lo]`).
- Exchange `a[i]` with `a[j]`.
- If (`a[i] == a[lo]`), exchange `a[i]` with `a[p]` and increment `p`.
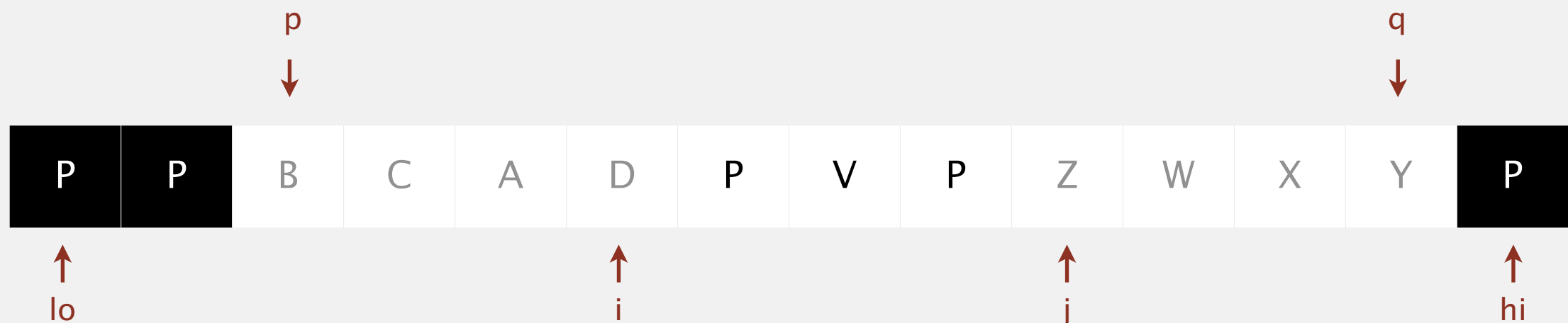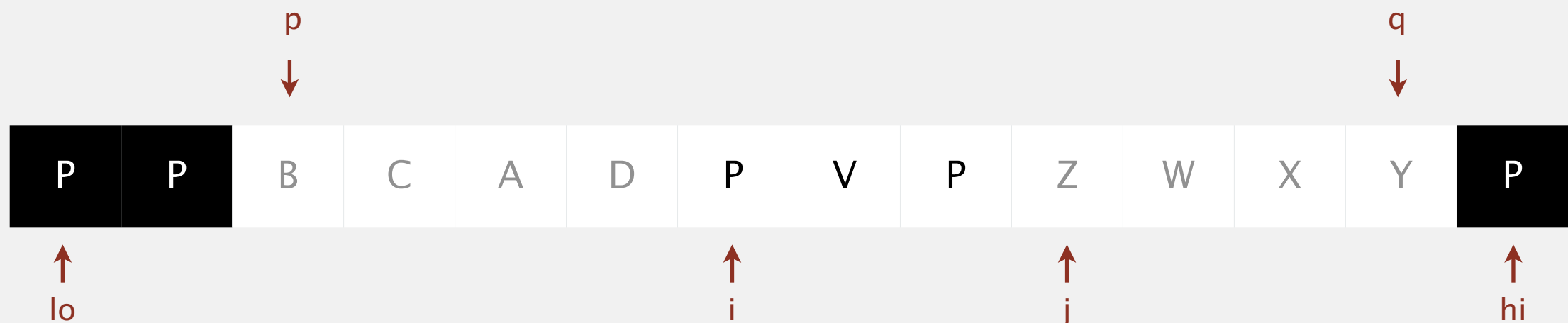- If (`a[j] == a[lo]`), exchange `a[j]` with `a[q]` and decrement `q`.

p   q
↓   ↓

| P | P | B | C | A | D | P | V | P | Z | W | X | Y | P |

↑   ↑   ↑   ↑
lo   i   j   hi

exchange `a[i]` with `a[j]`

Bentley-McIlroy 3-way partitioning demo

Phase I. Repeat until `i` and `j` pointers cross.

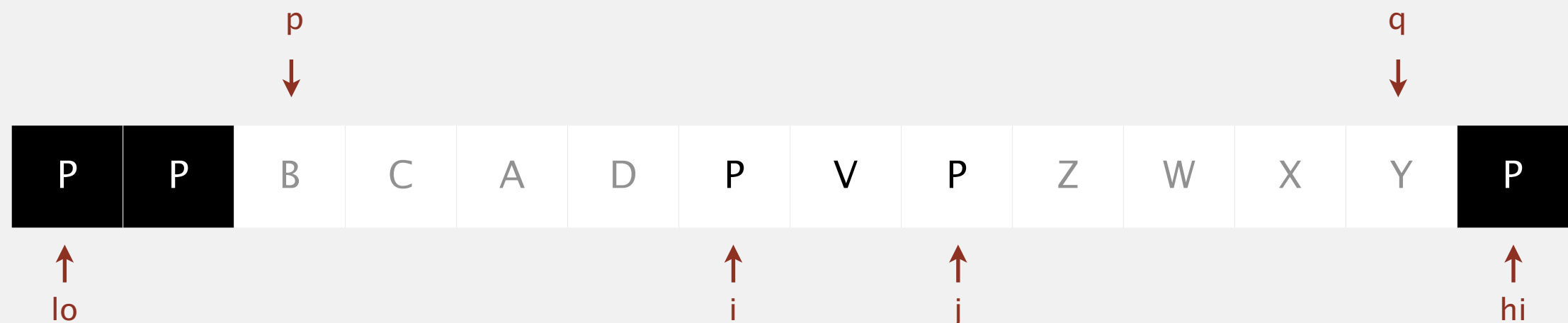- Scan `i` from left to right so long as `(a[i] < a[lo])`.
- Scan `j` from right to left so long as `(a[j] > a[lo])`.
- Exchange `a[i]` with `a[j]`.
- If `(a[i] == a[lo])`, exchange `a[i]` with `a[p]` and increment `p`.
- If `(a[j] == a[lo])`, exchange `a[j]` with `a[q]` and decrement `q`.

p											q	
P	P	B	C	A	D	P	V	P	Z	W	X	Y	P

lo   i   j   hi

exchange `a[i]` with `a[p]` and increment p
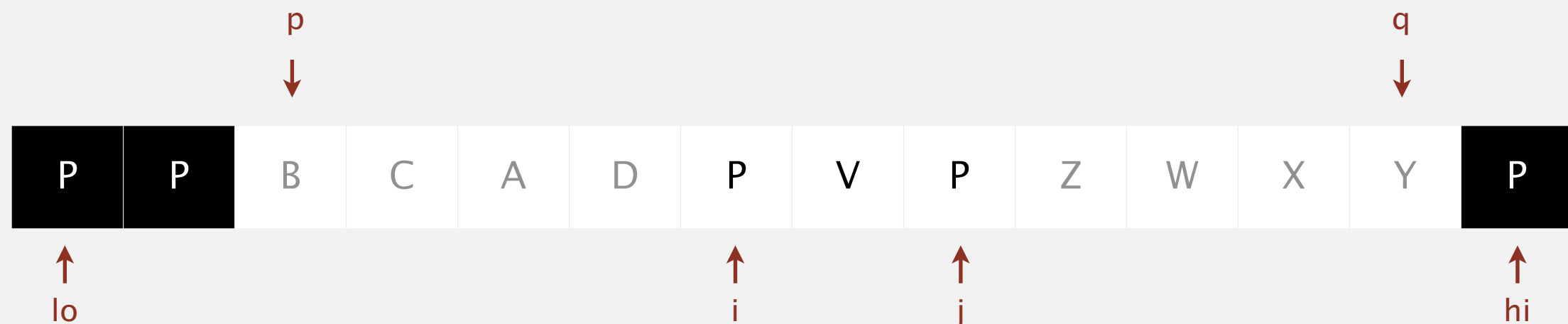
Bentley-McIlroy 3-way partitioning demo

Phase I. Repeat until `i` and `j` pointers cross.

- Scan `i` from left to right so long as (`a[i] < a[lo]`).
- Scan `j` from right to left so long as (`a[j] > a[lo]`).
- Exchange `a[i]` with `a[j]`.
- If (`a[i] == a[lo]`), exchange `a[i]` with `a[p]` and increment `p`.
- If (`a[j] == a[lo]`), exchange `a[j]` with `a[q]` and decrement `q`.

exchange a[j] with a[q] and decrement q

Phase I. Repeat until `i` and `j` pointers cross.

- Scan `i` from left to right so long as (`a[i] < a[lo]`).
- Scan `j` from right to left so long as (`a[j] > a[lo]`).
- Exchange `a[i]` with `a[j]`.
- If (`a[i] == a[lo]`), exchange `a[i]` with `a[p]` and increment `p`.
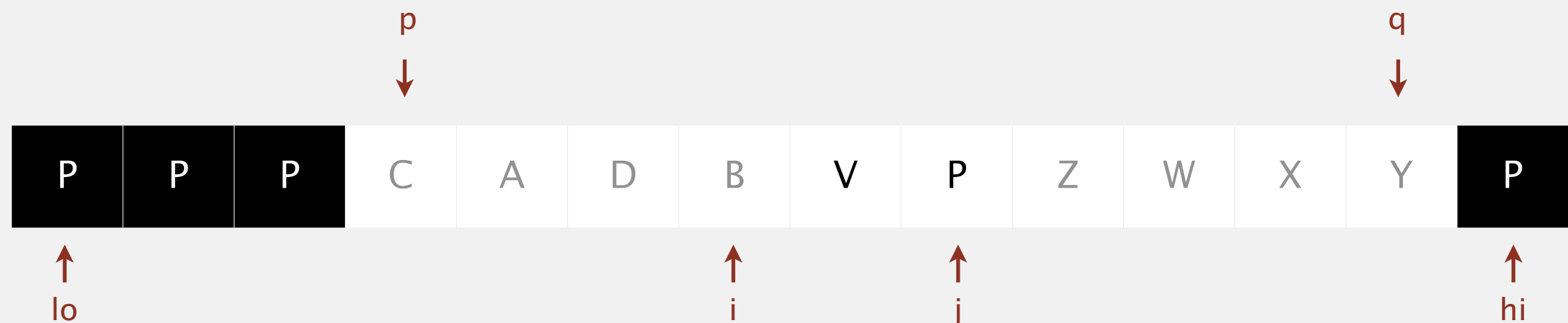- If (`a[j] == a[lo]`), exchange `a[j]` with `a[q]` and decrement `q`.

p										q	
P	P	P	C	A	D	B	V	Y	Z	W	X	P	P

lo   i   j   hi

Phase I. Repeat until `i` and `j` pointers cross.

- Scan `i` from left to right so long as (`a[i] < a[lo]`).
- Scan `j` from right to left so long as (`a[j] > a[lo]`).
- Exchange `a[i]` with `a[j]`.
- If (`a[i] == a[lo]`), exchange `a[i]` with `a[p]` and increment `p`.
- If (`a[j] == a[lo]`), exchange `a[j]` with `a[q]` and decrement `q`.

p   q
↓   ↓

| P | P | P | C | A | D | B | V | Y | Z | W | X | P | P |

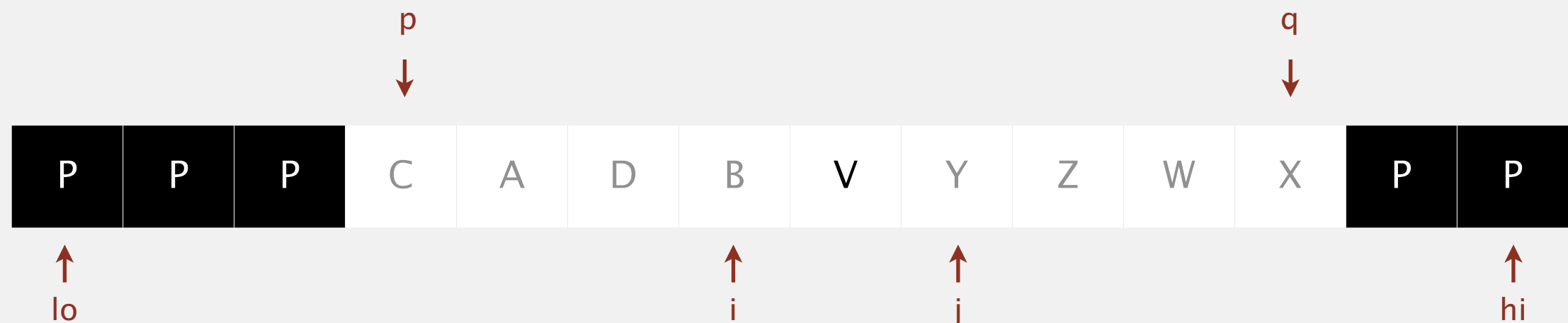↑   ↑   ↑   ↑
lo   i   j   hi

Bentley-McIlroy 3-way partitioning demo

Phase I. Repeat until `i` and `j` pointers cross.
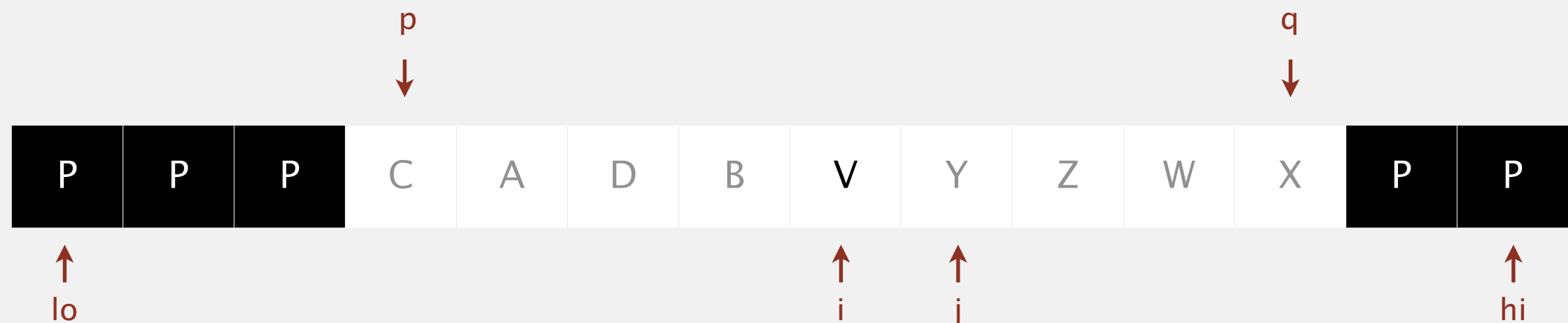- Scan `i` from left to right so long as (`a[i] < a[lo]`).
- Scan `j` from right to left so long as (`a[j] > a[lo]`).
- Exchange `a[i]` with `a[j]`.
- If (`a[i] == a[lo]`), exchange `a[i]` with `a[p]` and increment `p`.
- If (`a[j] == a[lo]`), exchange `a[j]` with `a[q]` and decrement `q`.

Bentley-McIlroy 3-way partitioning demo

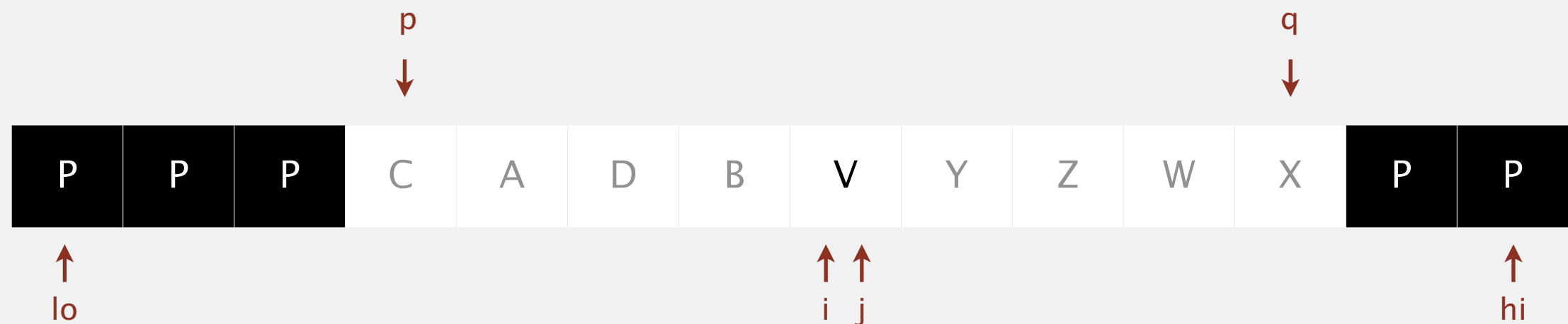Phase I. Repeat until `i` and `j` pointers cross.

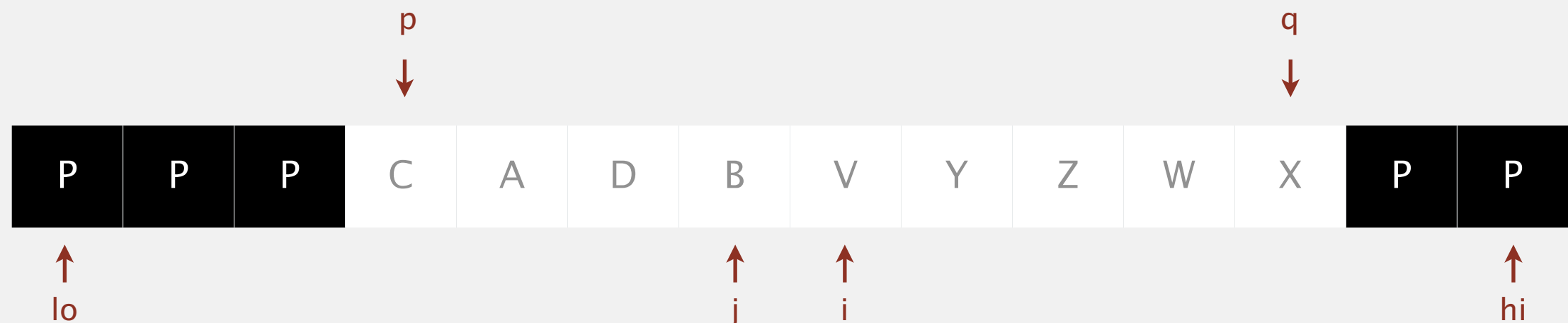- Scan `i` from left to right so long as (`a[i] < a[lo]`).
- Scan `j` from right to left so long as (`a[j] > a[lo]`).
- Exchange `a[i]` with `a[j]`.
- If (`a[i] == a[lo]`), exchange `a[i]` with `a[p]` and increment `p`.
- If (`a[j] == a[lo]`), exchange `a[j]` with `a[q]` and decrement `q`.

pointers cross

Bentley-McIlroy 3-way partitioning demo

Phase II. Swap equal keys to the center.
- Scan `j` and `p` from right to left and exchange `a[j]` with `a[p]`.
- Scan `i` and `q` from left to right and exchange `a[i]` with `a[q]`.

p
↓

| P | P | P | C | A | D | B | V | Y | Z | W | X | P | P |

↑ lo

↑ j   ↑ i

q
↓

↑ hi

exchange `a[j]` with `a[p]`

Bentley-McIlroy 3-way partitioning demo

Phase II. Swap equal keys to the center.

- Scan `j` and `p` from right to left and exchange `a[j]` with `a[p]`.
- Scan `i` and `q` from left to right and exchange `a[i]` with `a[q]`.

p											q	

P	P	B	C	A	D	P	V	Y	Z	W	X	P	P

lo   j   i   hi

exchange `a[j]` with `a[p]`

Bentley-McIlroy 3-way partitioning demo

Swap equal keys to the center.

- Scan `j` and `p` from right to left and exchange `a[j]` with `a[p]`.
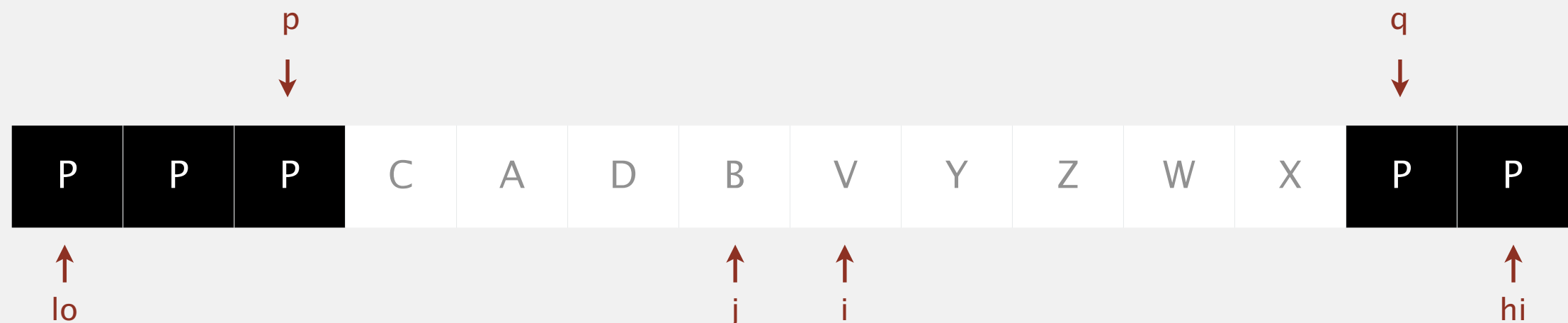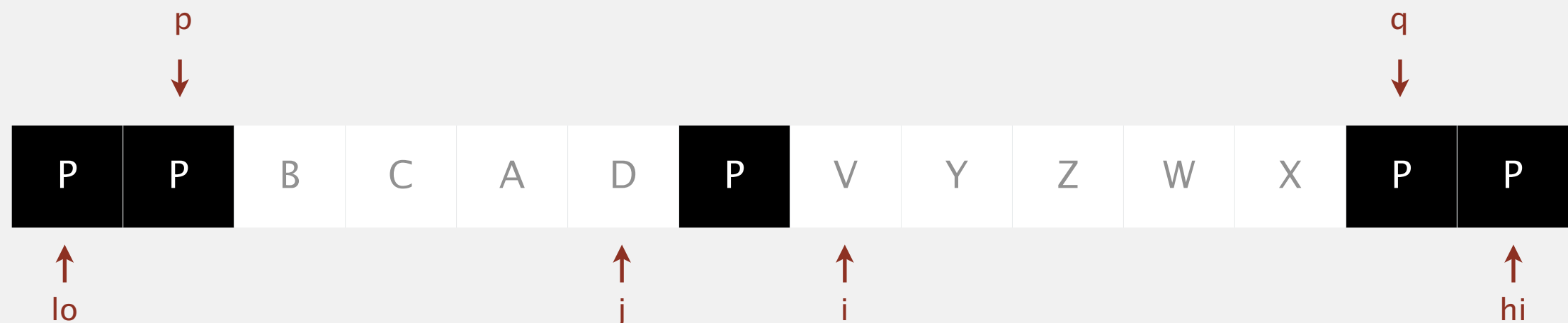- Scan `i` and `q` from left to right and exchange `a[i]` with `a[q]`.

exchange `a[j]` with `a[p]`

Bentley-McIlroy 3-way partitioning demo

Phase II. Swap equal keys to the center.
- Scan `j` and `p` from right to left and exchange `a[j]` with `a[p]`.
- Scan `i` and `q` from left to right and exchange `a[i]` with `a[q]`.

A	D	B	C	P	P	P	V	Y	Z	W	X	P	P

lo   j   i   q   hi

exchange `a[i]` with `a[q]`

Phase II. Swap equal keys to the center.

- Scan `j` and `p` from right to left and exchange `a[j]` with `a[p]`.
- Scan `i` and `q` from left to right and exchange `a[i]` with `a[q]`.

exchange a[i] with a[q]

Phase II. Swap equal keys to the center.

- Scan `j` and `p` from right to left and exchange `a[j]` with `a[p]`.
- Scan `i` and `q` from left to right and exchange `a[i]` with `a[q]`.
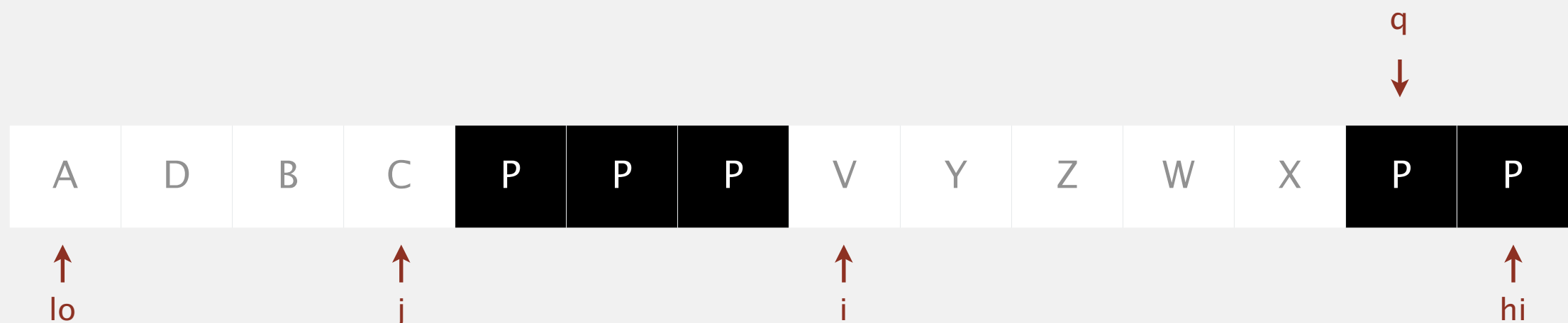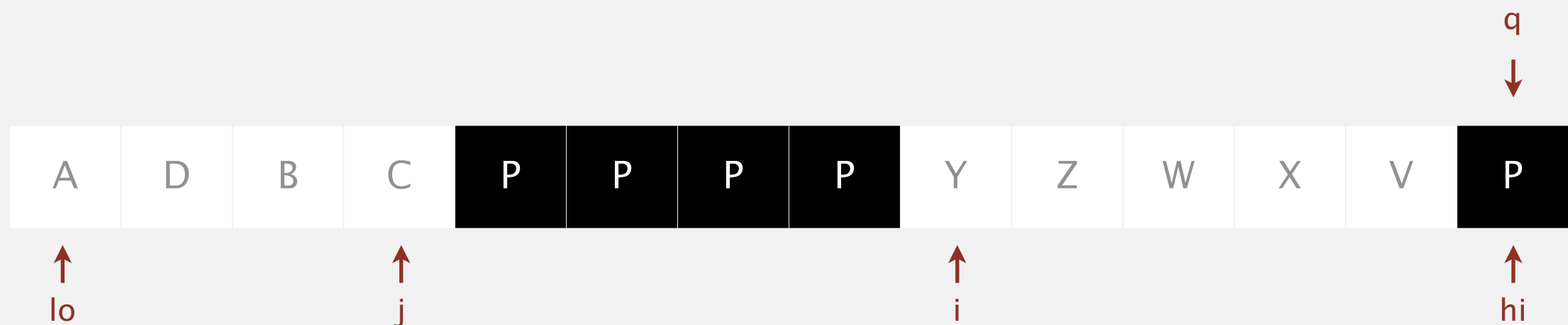

3-way partitioned

2.3 PARTITIONING DEMOS

‣ Sedgewick 2-way partitioning

‣ Dijkstra 3-way partitioning

‣ Bentley-McIlroy 3-way partitioning

‣ **dual-pivot partitioning**

Algorithms

ROBERT SEDGEWICK | KEVIN WAYNE

Dual-pivot partitioning demo

Initialization.

- Choose `a[lo]` and `a[hi]` as partitioning items.
- Exchange if necessary to ensure `a[lo]` ≤ `a[hi]`.

S	E	A	Y	R	L	F	V	Z	Q	T	C	M	K

↑
lo

↑
hi
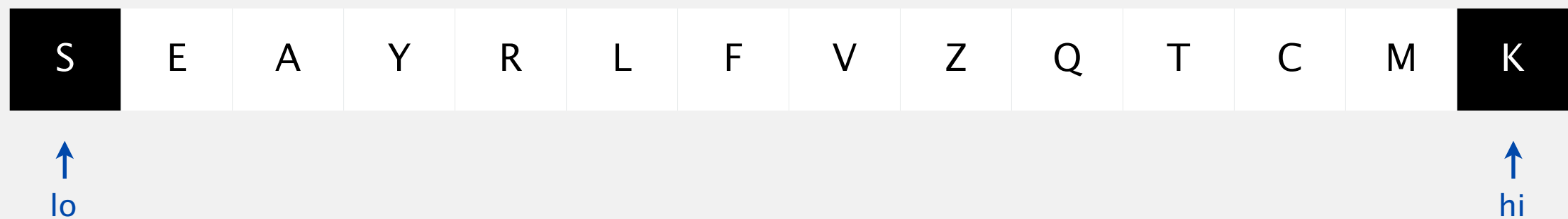
exchange a[lo] and a[hi]

Dual-pivot partitioning demo

Initialization.

- Choose `a[lo]` and `a[hi]` as partitioning items.
- Exchange if necessary to ensure `a[lo]` ≤ `a[hi]`.

K	E	A	Y	R	L	F	V	Z	Q	T	C	M	S

↑
lo

↑
hi

Dual-pivot partitioning demo

Main loop. Repeat until `i` and `gt` pointers cross.

- If   `(a[i] < a[lo])`, exchange `a[i]` with `a[lt]` and increment `lt` and `i`.
- Else if `(a[i] > a[hi])`, exchange `a[i]` with `a[gt]` and decrement `gt`.
- Else, increment `i`.

p₁	< p₁	p₁ ≤ and ≤ p₂	?	> p₂	p₂

$$p_1 \quad < p_1 \quad p_1 \leq \text{ and } \leq p_2 \quad ? \quad > p_2 \quad p_2$$

↑ lo   ↑ lt   ↑ i   ↑ gt   ↑ hi

| K | E | A | Y | R | L | F | V | Z | Q | T | C | M | S |

↑ lo   ↑↑ lt i   ↑ gt   ↑ hi

exchange a[i] and a[lt]; increment lt and i

Dual-pivot partitioning demo

Main loop. Repeat until `i` and `gt` pointers cross.

- If   (`a[i] < a[lo]`), exchange `a[i]` with `a[lt]` and increment `lt` and `i`.
- Else if (`a[i] > a[hi]`), exchange `a[i]` with `a[gt]` and decrement `gt`.
- Else, increment `i`.

p₁	< p₁	p₁ ≤ and ≤ p₂	?	> p₂	p₂
↑	↑	↑		↑	↑
lo	lt	i		gt	hi

K	E	A	Y	R	L	F	V	Z	Q	T	C	M	S
↑		↑ ↑										↑	↑
lo		lt i										gt	hi

exchange a[i] and a[lt]; increment lt and i

Dual-pivot partitioning demo

Main loop. Repeat until `i` and `gt` pointers cross.

- If   (`a[i] < a[lo]`), exchange `a[i]` with `a[lt]` and increment `lt` and `i`.
- Else if (`a[i] > a[hi]`), exchange `a[i]` with `a[gt]` and decrement `gt`.
- Else, increment `i`.

p₁	< p₁	p₁ ≤ and ≤ p₂	?	> p₂	p₂
↑ lo	↑ lt		↑ i	↑ gt	↑ hi

K	E	A	Y	R	L	F	V	Z	Q	T	C	M	S

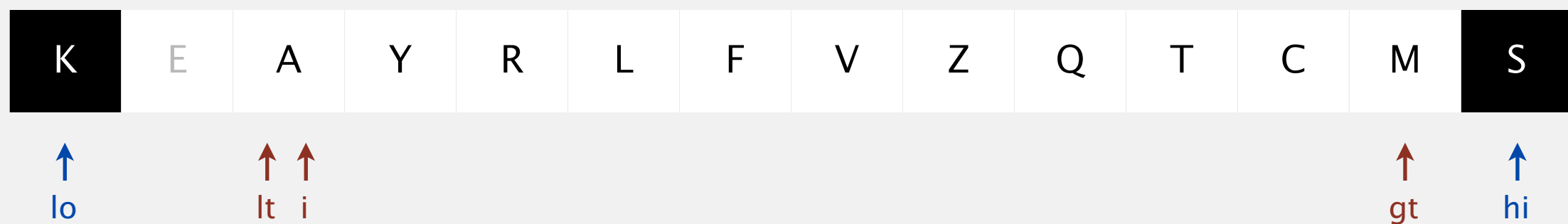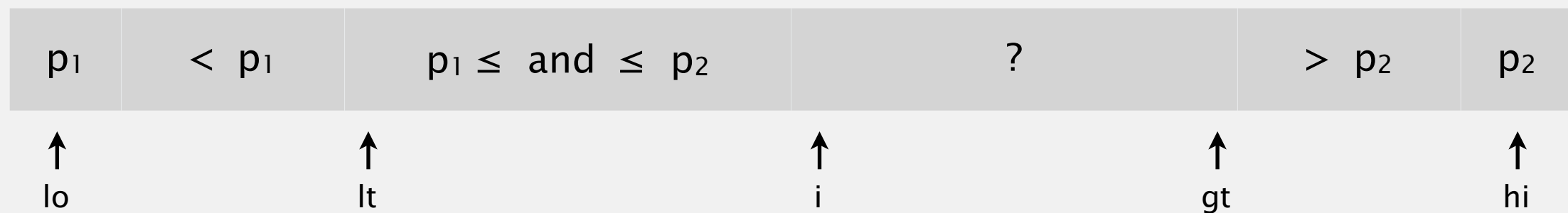lo → lt i   gt hi

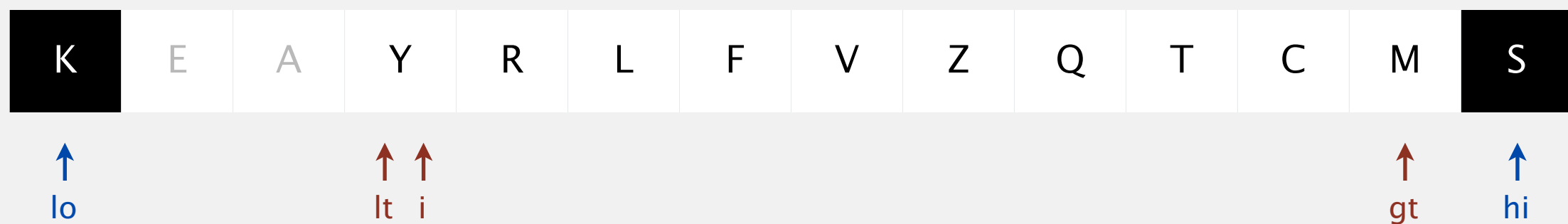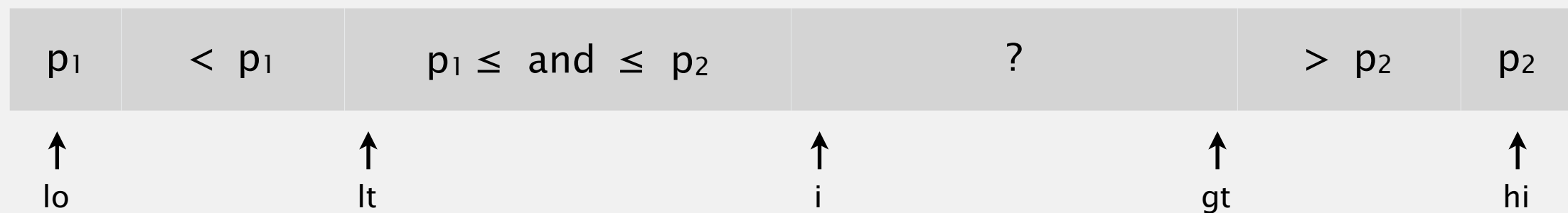**exchange a[i] and a[gt]; decrement gt**

Dual-pivot partitioning demo

Main loop. Repeat until `i` and `gt` pointers cross.

- If   `(a[i] < a[lo])`, exchange `a[i]` with `a[lt]` and increment `lt` and `i`.
- Else if `(a[i] > a[hi])`, exchange `a[i]` with `a[gt]` and decrement `gt`.
- Else, increment `i`.

p₁	< p₁	p₁ ≤ and ≤ p₂	?	> p₂	p₂

$$p_1 \qquad < p_1 \qquad p_1 \leq \text{ and } \leq p_2 \qquad ? \qquad > p_2 \qquad p_2$$

↑ lo   ↑ lt   ↑ i   ↑ gt   ↑ hi

K	E	A	M	R	L	F	V	Z	Q	T	C	Y	S

↑ lo   ↑↑ lt i   ↑ gt   ↑ hi

increment i

Dual-pivot partitioning demo

Main loop. Repeat until `i` and `gt` pointers cross.

- If   (`a[i] < a[lo]`), exchange `a[i]` with `a[lt]` and increment `lt` and `i`.
- Else if (`a[i] > a[hi]`), exchange `a[i]` with `a[gt]` and decrement `gt`.
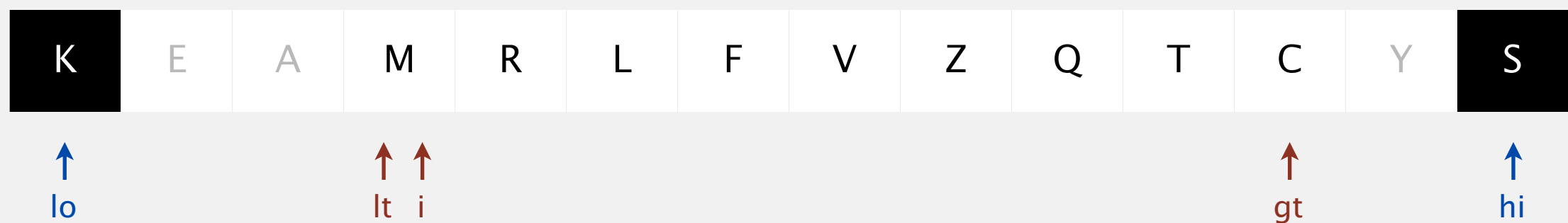- Else, increment `i`.

p₁	< p₁	p₁ ≤ and ≤ p₂	?	> p₂	p₂
↑	↑		↑	↑	↑
lo	lt		i	gt	hi

K	E	A	M	R	L	F	V	Z	Q	T	C	Y	S
↑			↑	↑								↑	↑
lo			lt	i								gt	hi

increment i

Dual-pivot partitioning demo

Main loop. Repeat until `i` and `gt` pointers cross.

- If   `(a[i] < a[lo])`, exchange `a[i]` with `a[lt]` and increment `lt` and `i`.
- Else if `(a[i] > a[hi])`, exchange `a[i]` with `a[gt]` and decrement `gt`.
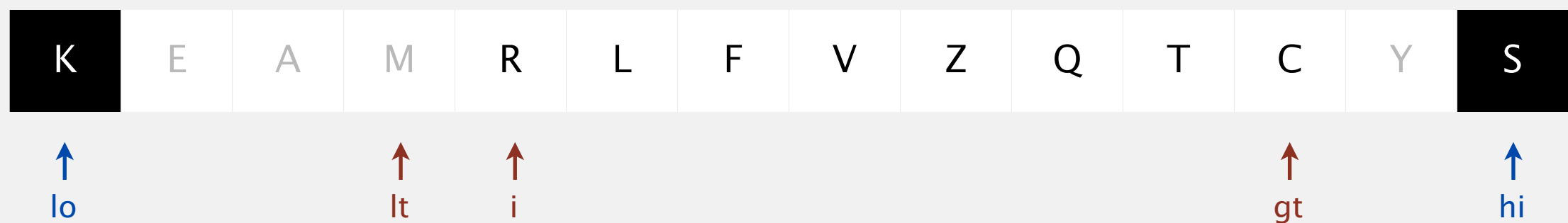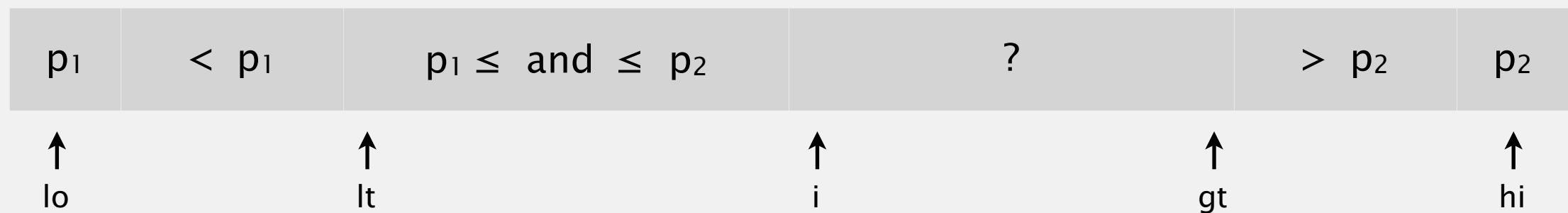- Else, increment `i`.

p₁	< p₁	p₁ ≤ and ≤ p₂	?	> p₂	p₂

p_1 $< p_1$ $p_1 \le$ and $\le p_2$? $> p_2$ p_2

↑ lo   ↑ lt   ↑ i   ↑ gt   ↑ hi

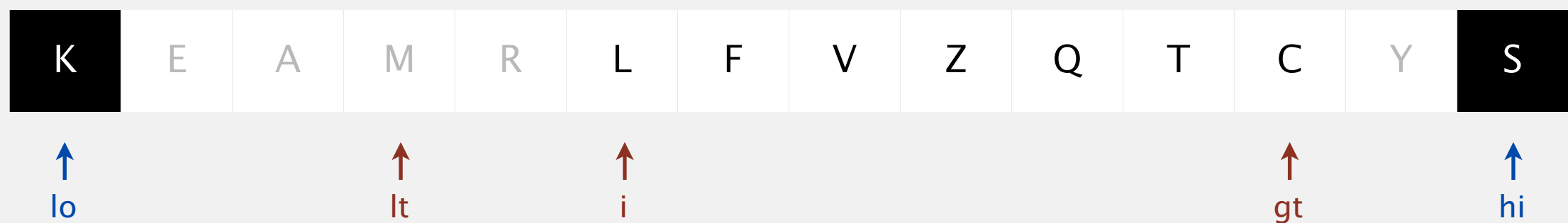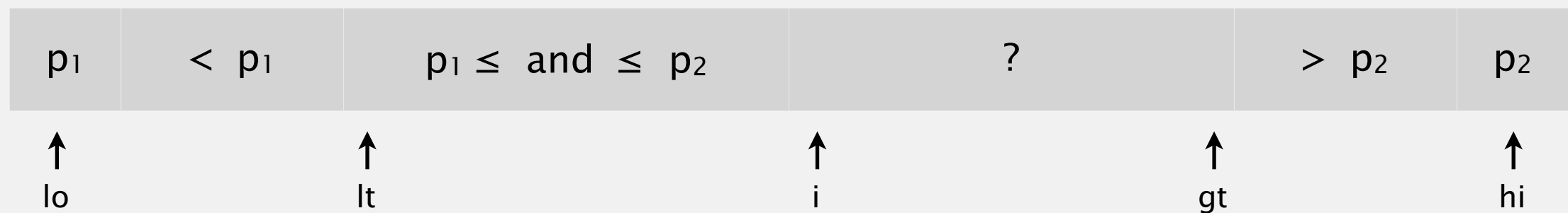| K | E | A | M | R | L | F | V | Z | Q | T | C | Y | S |

↑ lo   ↑ lt   ↑ i   ↑ gt   ↑ hi

increment i

Dual-pivot partitioning demo

Main loop. Repeat until `i` and `gt` pointers cross.

- If   (`a[i] < a[lo]`), exchange `a[i]` with `a[lt]` and increment `lt` and `i`.
- Else if (`a[i] > a[hi]`), exchange `a[i]` with `a[gt]` and decrement `gt`.
- Else, increment `i`.

p₁	< p₁	p₁ ≤ and ≤ p₂	?	> p₂	p₂
↑ lo	↑ lt		↑ i	↑ gt	↑ hi

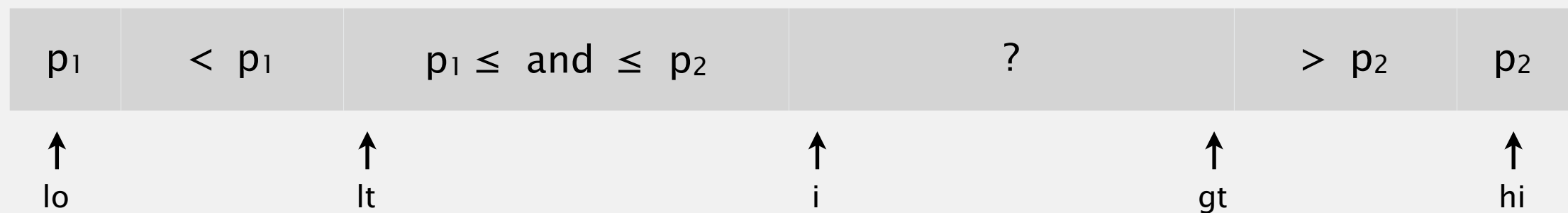| K	E	A	M	R	L	F	V	Z	Q	T	C	Y	S
↑ lo		↑ lt				↑ i						↑ gt	↑ hi

exchange a[i] and a[lt]; increment lt and i

Dual-pivot partitioning demo

Main loop. Repeat until `i` and `gt` pointers cross.

- If   (a[i] < a[lo]), exchange a[i] with a[lt] and increment `lt` and `i`.
- Else if (a[i] > a[hi]), exchange a[i] with a[gt] and decrement `gt`.
- Else, increment `i`.

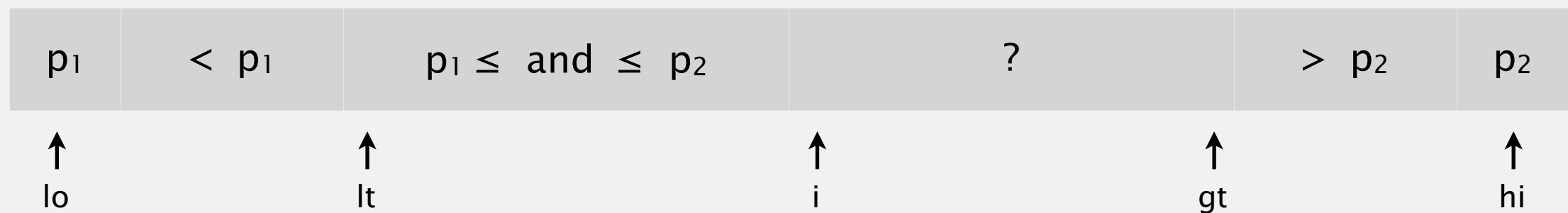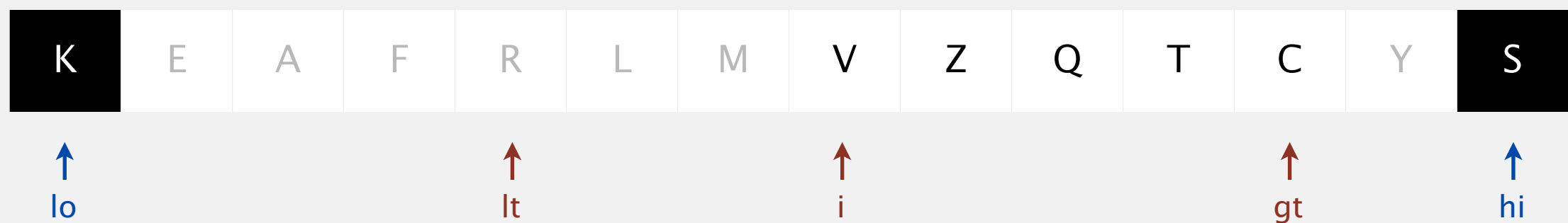| p₁	< p₁	p₁ ≤ and ≤ p₂	?	> p₂	p₂

p_1 $< p_1$ $p_1 \leq$ and $\leq p_2$? $> p_2$ p_2

↑ lo   ↑ lt   ↑ i   ↑ gt   ↑ hi

K	E	A	F	R	L	M	V	Z	Q	T	C	Y	S

↑ lo   ↑ lt   ↑ i   ↑ gt   ↑ hi

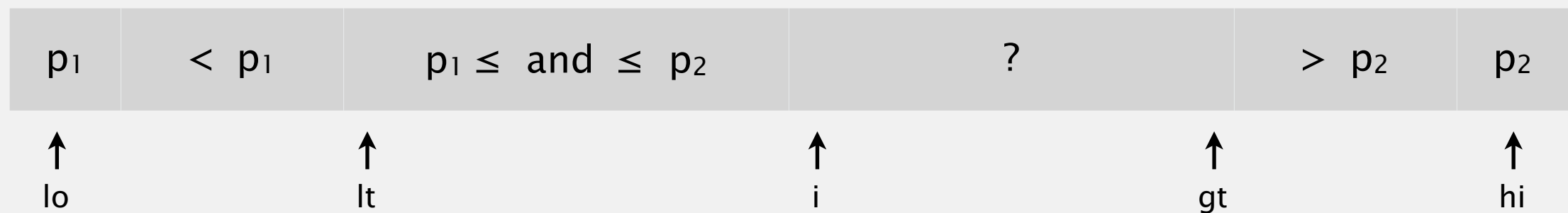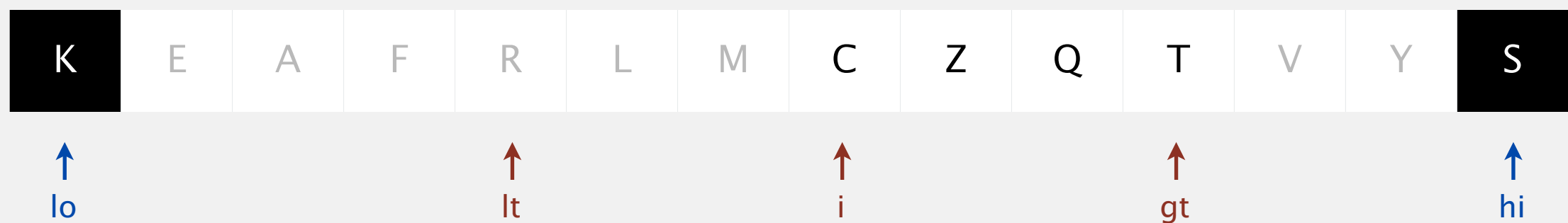**exchange a[i] and a[gt]; decrement gt**

Dual-pivot partitioning demo

Main loop. Repeat until `i` and `gt` pointers cross.

- If   `(a[i] < a[lo])`, exchange `a[i]` with `a[lt]` and increment `lt` and `i`.
- Else if `(a[i] > a[hi])`, exchange `a[i]` with `a[gt]` and decrement `gt`.
- Else, increment `i`.

p₁	< p₁	p₁ ≤ and ≤ p₂	?	> p₂	p₂

p_1 $< p_1$ $p_1 \leq$ and $\leq p_2$? $> p_2$ p_2

↑ lo   ↑ lt   ↑ i   ↑ gt   ↑ hi

K	E	A	F	R	L	M	C	Z	Q	T	V	Y	S

↑ lo   ↑ lt   ↑ i   ↑ gt   ↑ hi

exchange a[i] and a[lt]; increment lt and i

Dual-pivot partitioning demo

Main loop. Repeat until `i` and `gt` pointers cross.

- If   (`a[i] < a[lo]`), exchange `a[i]` with `a[lt]` and increment `lt` and `i`.
- Else if (`a[i] > a[hi]`), exchange `a[i]` with `a[gt]` and decrement `gt`.
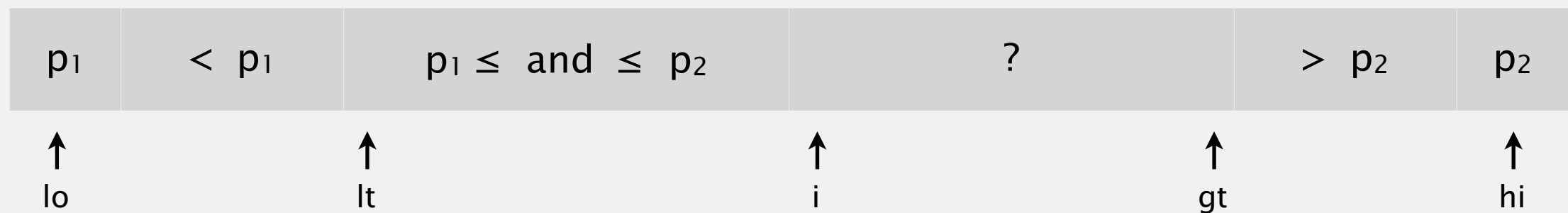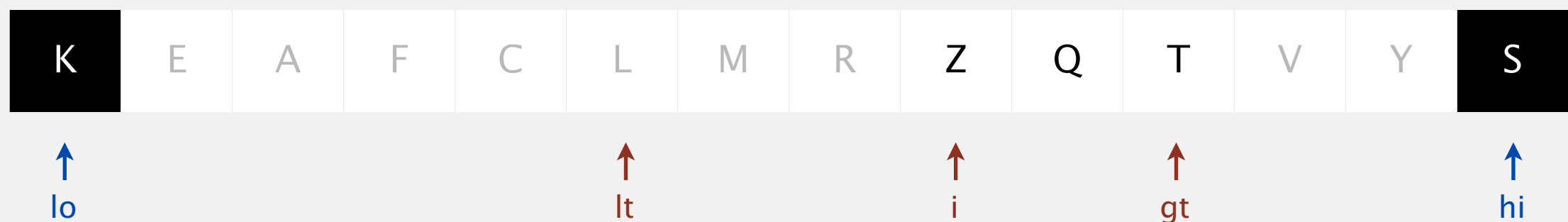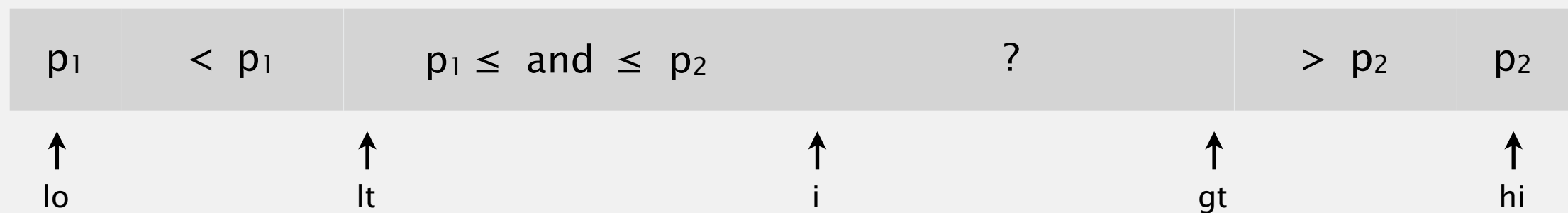- Else, increment `i`.

p₁	< p₁	p₁ ≤ and ≤ p₂	?	> p₂	p₂

p_1 < p_1 $p_1 \le$ and $\le p_2$?   > p_2 p_2

↑   ↑   ↑   ↑   ↑
lo   lt   i   gt   hi

K	E	A	F	C	L	M	R	Z	Q	T	V	Y	S

↑   ↑   ↑   ↑   ↑
lo   lt   i   gt   hi

exchange a[i] and a[gt]; decrement gt

Dual-pivot partitioning demo

Main loop. Repeat until `i` and `gt` pointers cross.

- If $\quad$ (`a[i] < a[lo]`), exchange `a[i]` with `a[lt]` and increment `lt` and `i`.
- Else if (`a[i] > a[hi]`), exchange `a[i]` with `a[gt]` and decrement `gt`.
- Else, increment `i`.

p₁	< p₁	p₁ ≤ and ≤ p₂	?	> p₂	p₂
↑ lo	↑ lt		↑ i	↑ gt	↑ hi

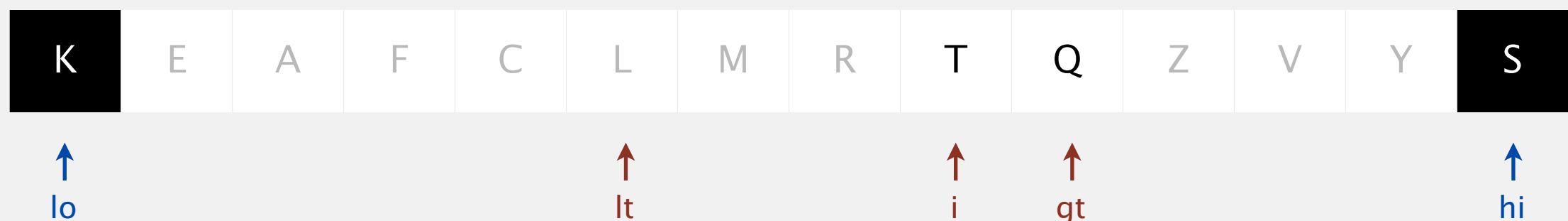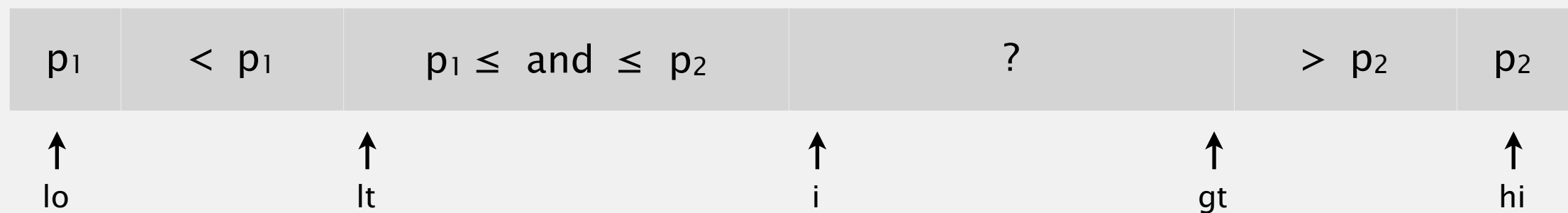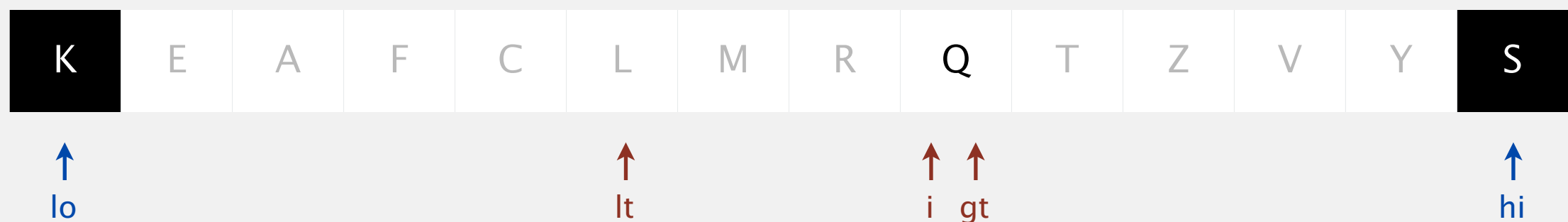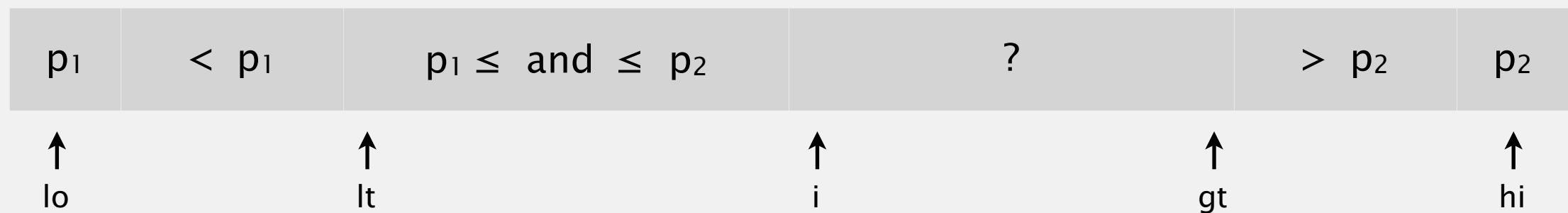| K	E	A	F	C	L	M	R	T	Q	Z	V	Y	S
↑ lo					↑ lt			↑ i	↑ gt				↑ hi

exchange a[i] and a[gt]; decrement gt

Dual-pivot partitioning demo

Main loop. Repeat until `i` and `gt` pointers cross.

- If   `(a[i] < a[lo])`, exchange `a[i]` with `a[lt]` and increment `lt` and `i`.
- Else if `(a[i] > a[hi])`, exchange `a[i]` with `a[gt]` and decrement `gt`.
- Else, increment `i`.

p₁	< p₁	p₁ ≤ and ≤ p₂	?	> p₂	p₂

p_1	$< p_1$	$p_1 \leq$ and $\leq p_2$	?	$> p_2$	p_2
↑ lo	↑ lt		↑ i	↑ gt	↑ hi

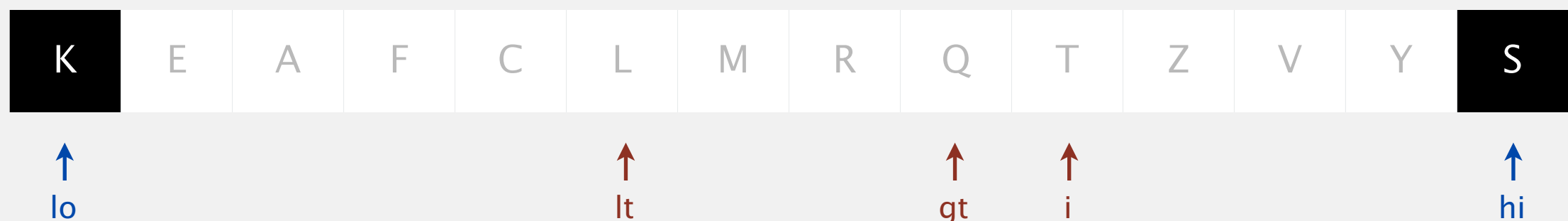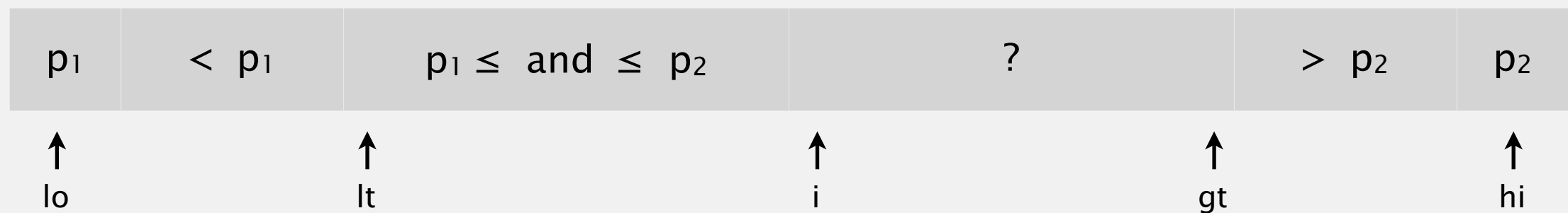| K	E	A	F	C	L	M	R	Q	T	Z	V	Y	S
↑ lo					↑ lt			↑ i	↑ gt				↑ hi

increment i

Dual-pivot partitioning demo

Main loop. Repeat until `i` and `gt` pointers cross.

- If   (`a[i] < a[lo]`), exchange `a[i]` with `a[lt]` and increment `lt` and `i`.
- Else if (`a[i] > a[hi]`), exchange `a[i]` with `a[gt]` and decrement `gt`.
- Else, increment `i`.

p₁	< p₁	p₁ ≤ and ≤ p₂	?	> p₂	p₂

$$p_1 \quad < p_1 \quad p_1 \leq \text{ and } \leq p_2 \quad ? \quad > p_2 \quad p_2$$

↑ lo   ↑ lt   ↑ i   ↑ gt   ↑ hi

K	E	A	F	C	L	M	R	Q	T	Z	V	Y	S

↑ lo   ↑ lt   ↑ gt   ↑ i   ↑ hi

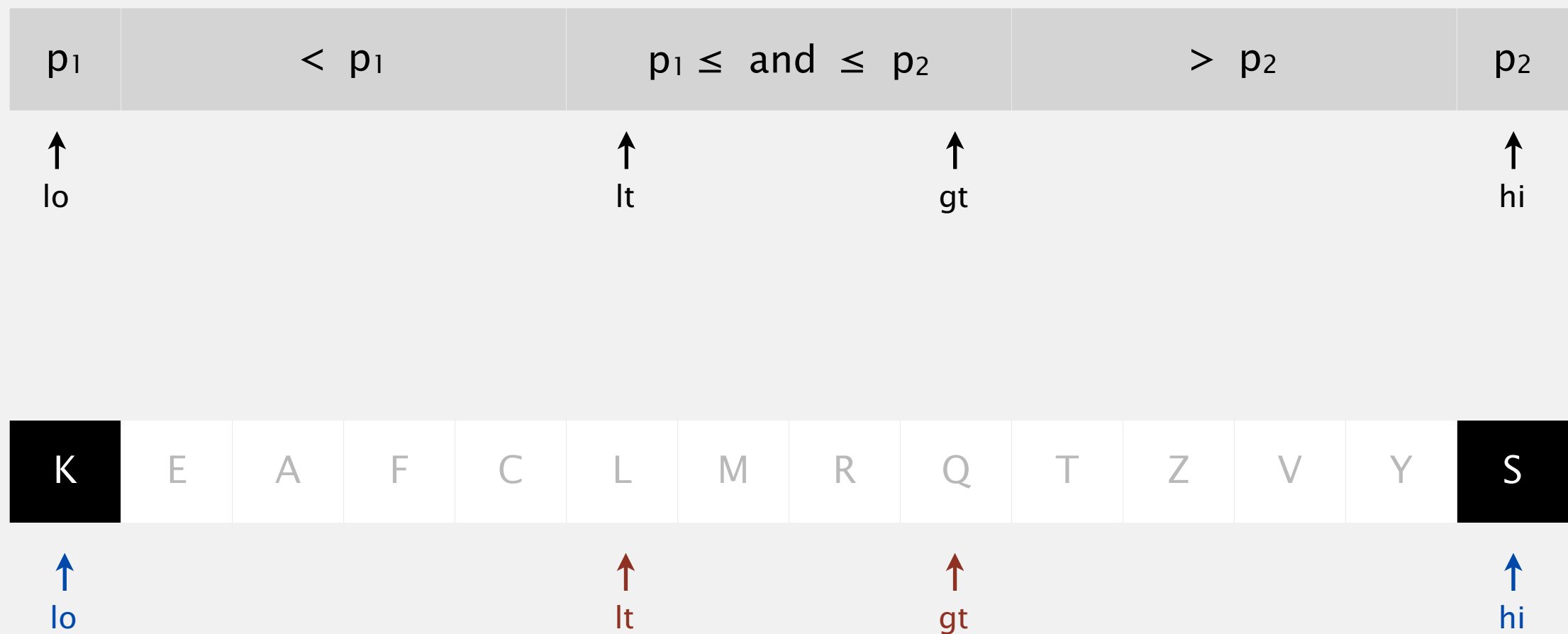**stop when pointers cross**

Dual-pivot partitioning demo

Finalize.

- Exchange `a[lo]` with `a[--lt]`.
- Exchange `a[hi]` with `a[++gt]`.

p₁	< p₁	p₁ ≤ and ≤ p₂	> p₂	p₂

↑ lo ↑ lt ↑ gt ↑ hi

K	E	A	F	C	L	M	R	Q	T	Z	V	Y	S

↑ lo ↑ lt ↑ gt ↑ hi

Dual-pivot partitioning demo

Finalize.

- Exchange `a[lo]` with `a[--lt]`.
- Exchange `a[hi]` with `a[++gt]`.

< p₁	p₁	p₁ ≤ and ≤ p₂	p₂	> p₂

↑ lo   ↑ lt   ↑ gt   ↑ hi

C	E	A	F	**K**	L	M	R	Q	**S**	Z	V	Y	T

↑ lo   ↑ lt   ↑ gt   ↑ hi

3-way partitioned