



中国科学院大学  
University of Chinese Academy of Sciences

CS101

# Lab Slides

# Human Sorter

**z xu@ict.ac.cn**  
**zhangjialin@ict.ac.cn**

# Design a real computer to do quicksort

Read textbook 7.3 2.3 4.3.3 5.3.3

- However, this computer is made of a team of students
  - Students act as data and hardware components
- Process
  1. **Form teams**, each consisting of no more than 30 students and a team leader who serves the team
  2. Every student designs her/his own human sorter computer
  3. Each team decides on a design based on member designs
  4. Field run of the team design
  5. Team leader presents to the class
- Grading policy
  - Every student hands in a lab report
  - Total = Team leader's presentation  $\times 10\%$  + Student's report  $\times 90\%$

# How to do it?

- Read 7.3, 2.3, 4.3.3, 5.3.3
- Use your imagination, but note the following
  - Design a Human sorter rather an electronic von Neumann computer
  - The two may have many possibilities of similarities and differences

<b>Five features of the von Neumann Model of Computer (Textbook 2.3)</b>	<b>Human Sorter</b>
1. Binary representation	Not necessarily
2. P-M-I/O organization	May ignore I/O; P-M may have different organizations
3. Stored program	The program is not necessarily stored in memory, but in the Controller's brain
4. Instruction driven	Yes. May include powerful instructions, such as generating a random number in one instruction
5. Sequential execution	Yes

# Objective

- Design a team **computer** of students to execute a quicksort **program**, to sort the students in the team from **name order** to **height order**



# Criteria of each student's design

- A good lab report must include
  - The team computer organization
    - What components are used to form a computer system
  - The instruction set of the team computer
  - The quicksort program made of a sequence of such instructions
  - The evaluation record of program executions
    - Should include at least two executions, to show consistent results



# Criteria of each student's design

- The evaluation must show that the design satisfies three correctness properties:
  - Result correctness: the students are indeed ordered by height
  - Algorithmic correctness: the execution implements the quicksort algorithm
  - Systems correctness: the team computer executes the program sequentially, i.e., step-by-step, one instruction after another
    - Instruction-driven: every one in Data Group, NO ORDER NO MOVE!
    - Serial execution: the next instruction starts only when the current instruction finishes.





# Suggested structure of Lab report

- The lab report needs to include the
  1. Hardware design
  2. Instruction set design
  3. Quicksort program
  4. Summary of experiment (at least two executions)
    - Input and initial configuration
    - Output and final configuration
    - Number of steps executed
    - Total execution time of an execution (wall clock time)
    - Comments on distinct features of your design, and other observations
  5. Appendix: records of executions and other evidence

# Sample contents in Lab report

## ● Hardware design

- Use students and simple additional aids
  - E.g., paper marks on the lawn. Note: do not litter

Component	Implemented by	Duty of the component
Controller	1 student	Decode instruction, control other components to execute instruction, and determine next instruction
...	...	...

## ● Instruction set design

- The instruction set consists of 9 instructions (or 7, 12, 15)
- Assume the program is stored in the Controller's brain, who can control the execution of any instruction in one step

Opcode	Operand1	Operand2	Operand3	Explanatory remarks
MOV	Immediate value	Register1	None	Set Register1 to the Immediate value E.g., MOV 0, R1 means 0→R1
...	...	...	...	...



# Sample contents in Lab report

- The “assembly language” program of quicksort
  - E.g., my program is shown in the following table, which consists of a sequence of 12 instructions

No.	Instruction	Explanatory Comments
1	MOV 17, R1	Initialize R1 to N, the number of data items to be sorted
...	...	...
12	HALT	The program terminates. The sorted result is in the Operational Unit

- Record of execution (of 129 steps)

Step	Instruction Executed	Explanatory Comments
0	Initial configuration	Students of the data group are ordered by name; N=16
1	MOV 17, R1	Initialize R1 to N, the number of data items to be sorted
2	...	...
...	...	...
...	...	Start of the first iteration of the main loop
...	...	...; comparison #1; ...
...	...	...
129	Halt	End of program; the data group are now ordered by height

# Suggestions for team presentation

- Highlight the team design
- Highlight two executions in field runs
- Present evidence for three correctness properties:
  - Result correctness
  - Algorithmic correctness
  - Systems correctness
- Present observations on notable features and happenings in experiment
- Use photographs, videos, and tables to present the experiment

# Sample Q&A

- Q: Why bother doing this team sorter project?
  - After all, if the quicksort algorithm is correct and the input is correct, the resulting output must be correct.
- A: This project deepens understanding of algorithmic thinking and systems thinking
  - By doing human computation step-by-step personally, students will experience and appreciate the following
    - ① Time complexity  $O(N \log_2 N)$  matters.
      - When  $N=10$ ,  $N \log_2 N=34$ ; When  $N=30$ ,  $N \log_2 N=148$
      - The team of students could become tired when  $N>15$
    - ② Knuth's 5-point characterization of algorithms
      - A human computer could perform powerful instructions and violate the 5-th criteria (Effectiveness)

# Sample Q&A

- Q: Why bother doing this team sorter project?
  - **Assumption:** if the quicksort algorithm is correct and the input is correct, the resulting output must be correct.
    - Right? **No!**
- A: This project deepens understanding of algorithmic thinking and systems thinking
  - ③ The assumption in the question is incorrect. Given correct algorithm and input, a computer can generate a wrong output.
    - E. g., a human computer tends to violate the instruction-driven property, and data could move proactively without following commands (as found in the field run)
  - ④ The environment could affect program execution
    - Often discovered in the field run

# Appendix

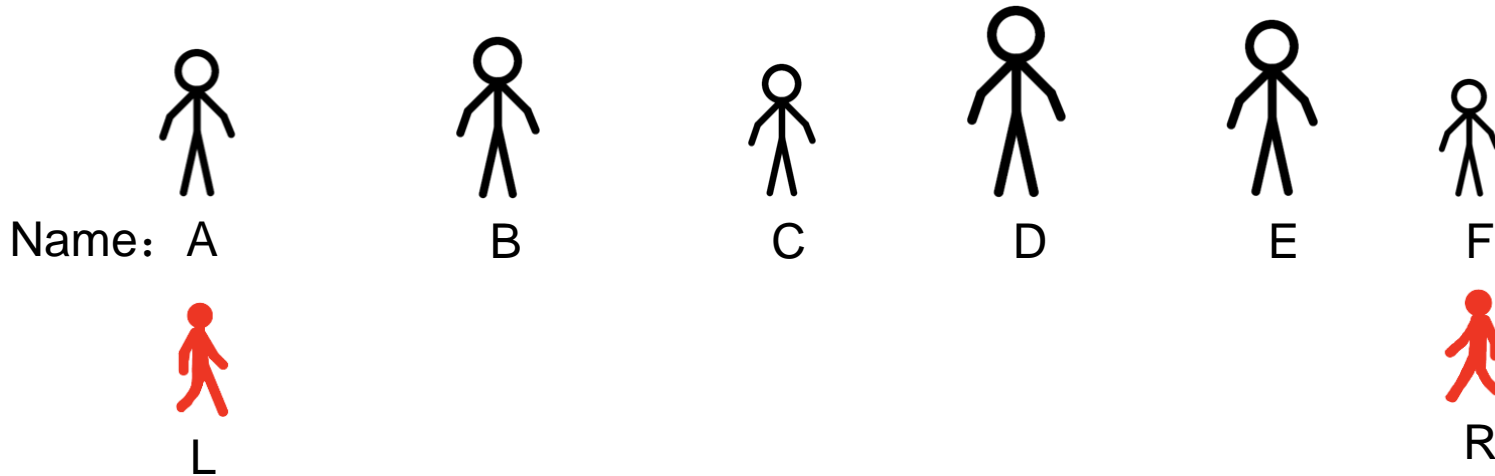
Some students may need more hints or help.

The following slides illustrate parts of a design.

Many other imaginative designs are possible,  
to realize the key concept of **recursion**.

# Hardware design

- Data Group: A, B, C, D, E, F
- Registers: L, R
  - L = 1, R = 6
- Others: Controller, Monitor, Overseer, Stepper
  - Not shown in this slice



# Instruction set design

Opcode	Operand1	Operand2	Explanatory remarks
select	label1	label2	Select the leftmost area as large as possible where everyone is standing. Let $L$ = index of the leftmost, $R$ = index of the rightmost. If $L$ equals to $R$ , jump to label1. If no such area exists, jump to label2.
pivot			Randomly select a student in $[L, R]$ as the pivot. The pivot needs to hold the flag up.
partition			Adjust students in $[L, R]$ so that students taller than the pivot are on the right side of the pivot, and students shorter than the pivot are on the left side of the pivot.
squat			Within $[L, R]$ : <ul style="list-style-type: none"><li>Let the student holding the flag (pivot) lay the flag down, and squat;</li><li>If no one holding the flag, let all students squat.</li></ul>
goto	label1		Jump to label1.
halt			All students in data group stand up and the program terminates.





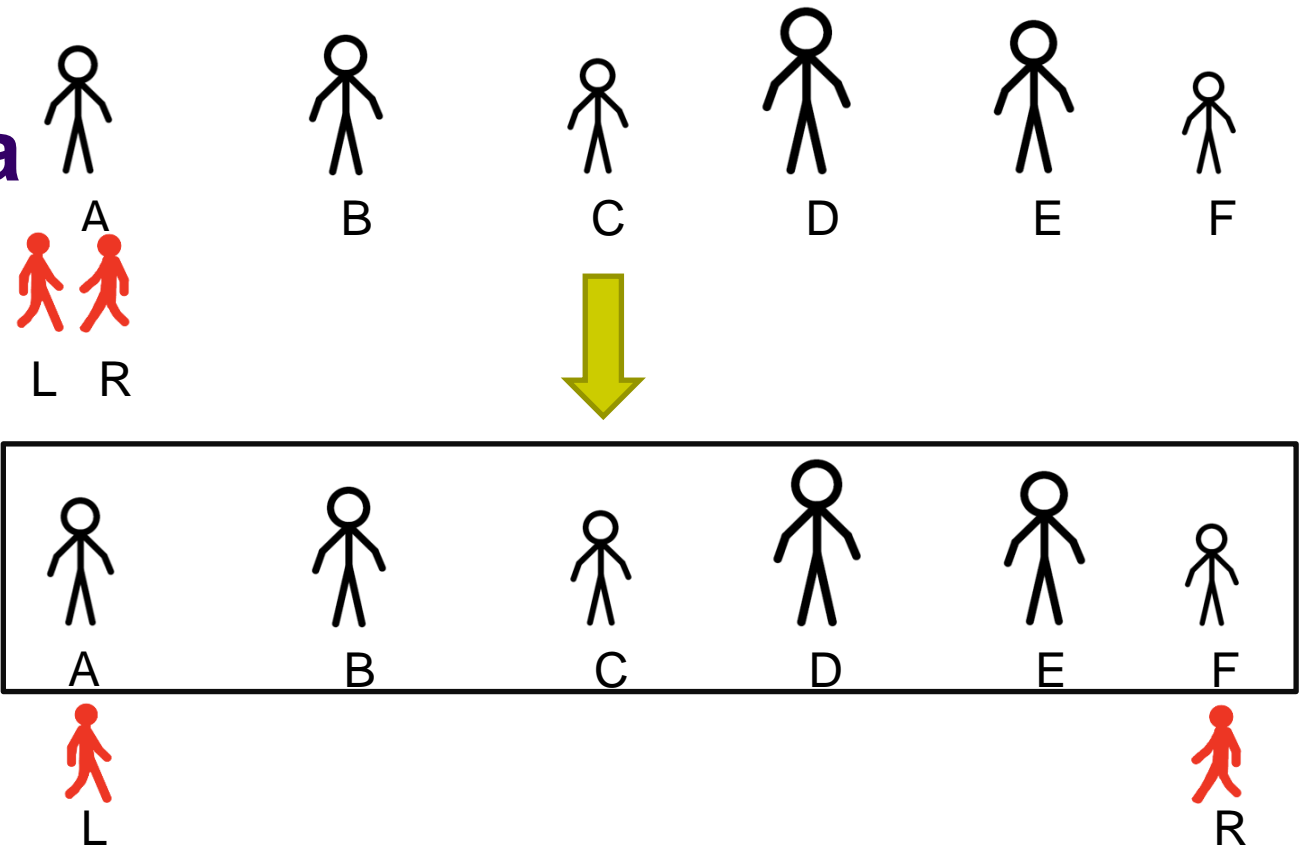
L=R=1 (point to the leftmost data)

L2: pivot # select a pivot

**L4:** squat # pivot squats. If no pivot, all students in [L, R] squat

**L6:** halt # the program terminates

# Step 1: Select a area



L1: select L4, L6

L2: pivot

L3: partition

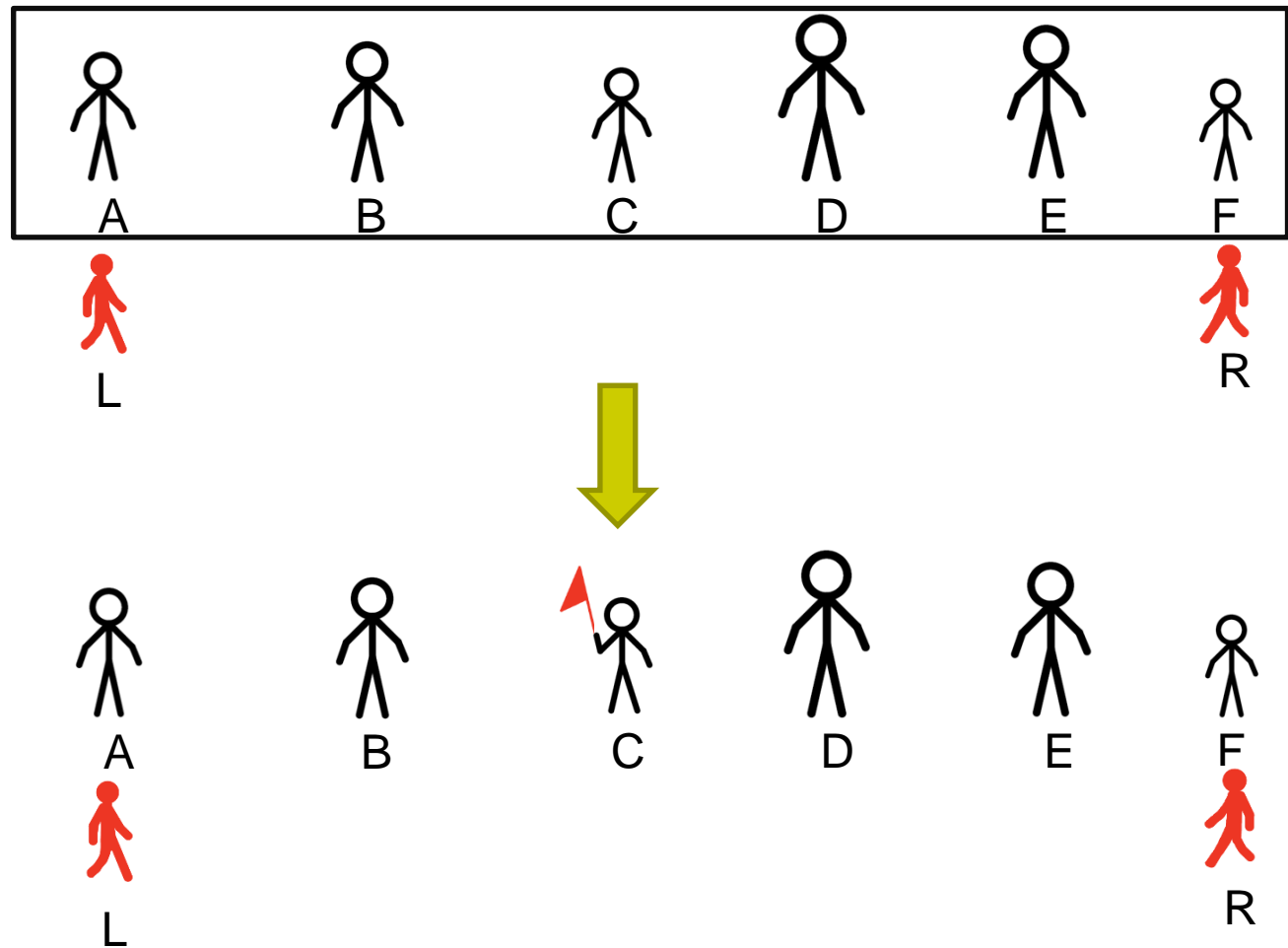
L4: squat

L5: goto L1

L6: halt

$L = 1, R = 6$

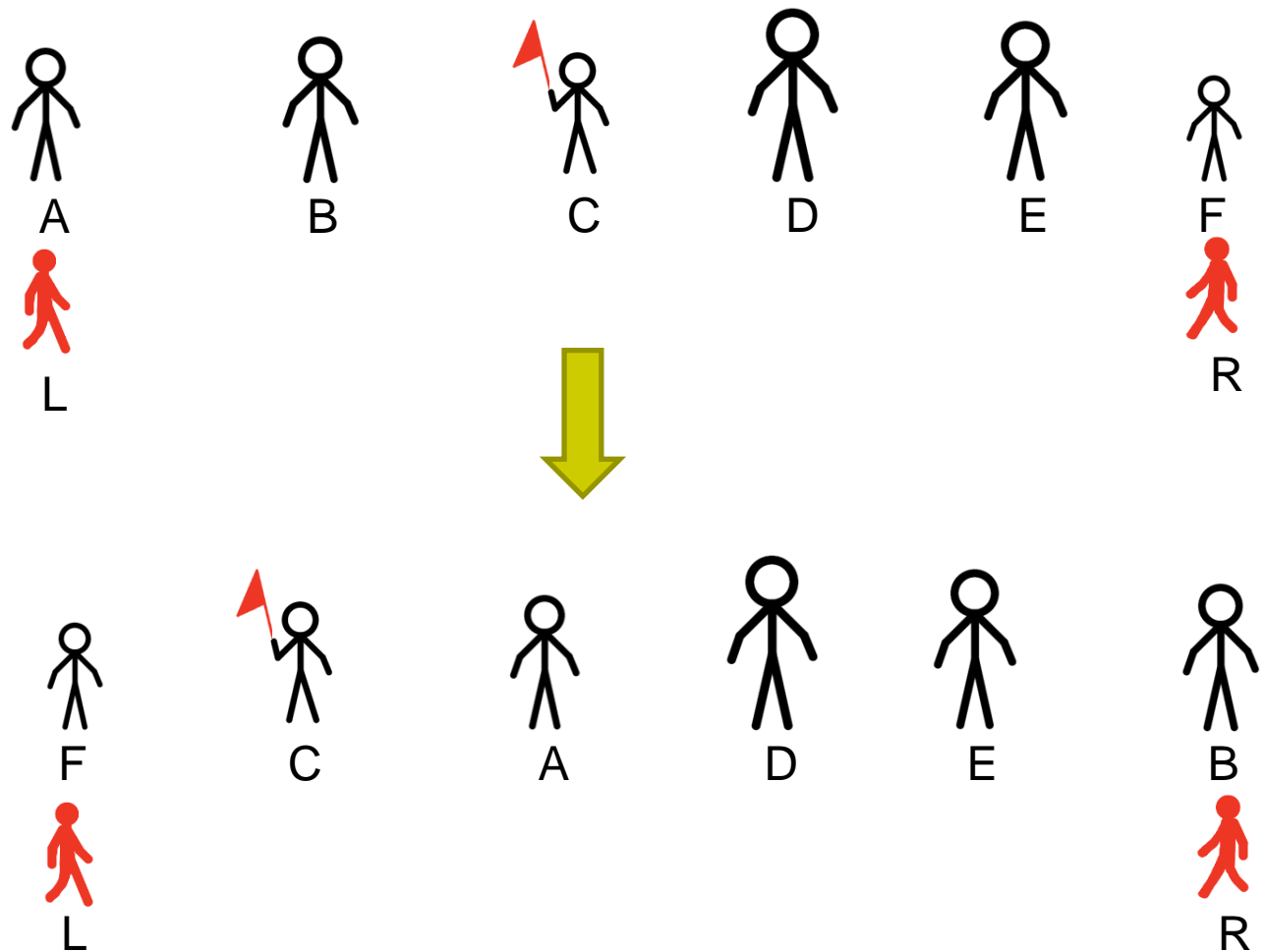
## Step 2: Select a pivot



L1: select L4, L6  
L2: **pivot**  
L3: partition  
L4: squat  
L5: goto L1  
L6: halt

C is selected as the pivot, and C holds the flag up.

# Step 3: Partition

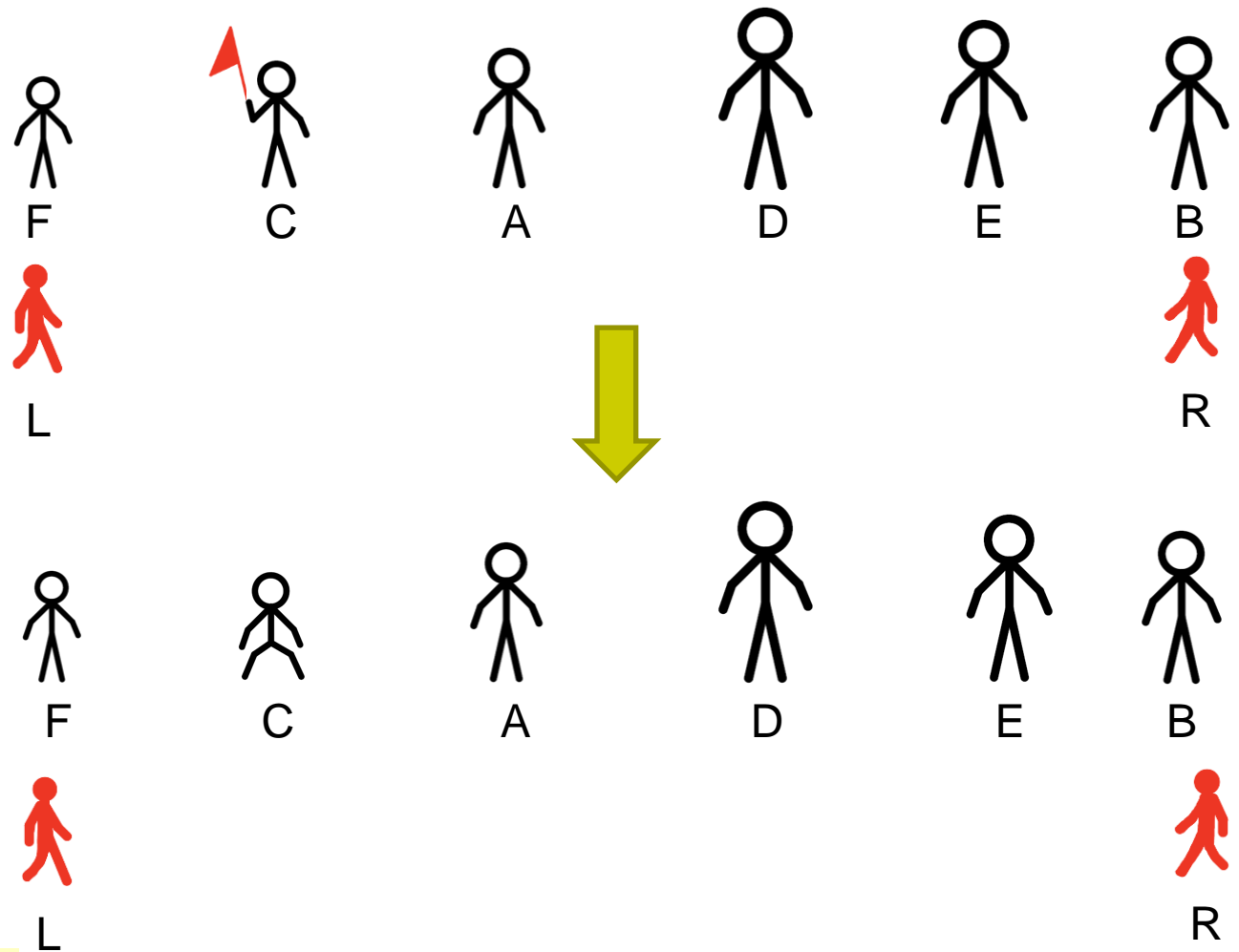


L1: select L4, L6  
L2: pivot  
L3: **partition**  
L4: squat  
L5: goto L1  
L6: halt

Adjust students in  $[L, R]$ , students

- taller than the pivot are on the right side of the pivot
  - shorter than the pivot are on the left side of the pivot
- L and R keep unchanged

## Step 4: Squat



L1: select L4, L6  
L2: pivot  
L3: partition  
**L4: squat**  
L5: goto L1  
L6: halt

Let the student holding the flag (C) lay the flag down and squat

## Step 5: Jump

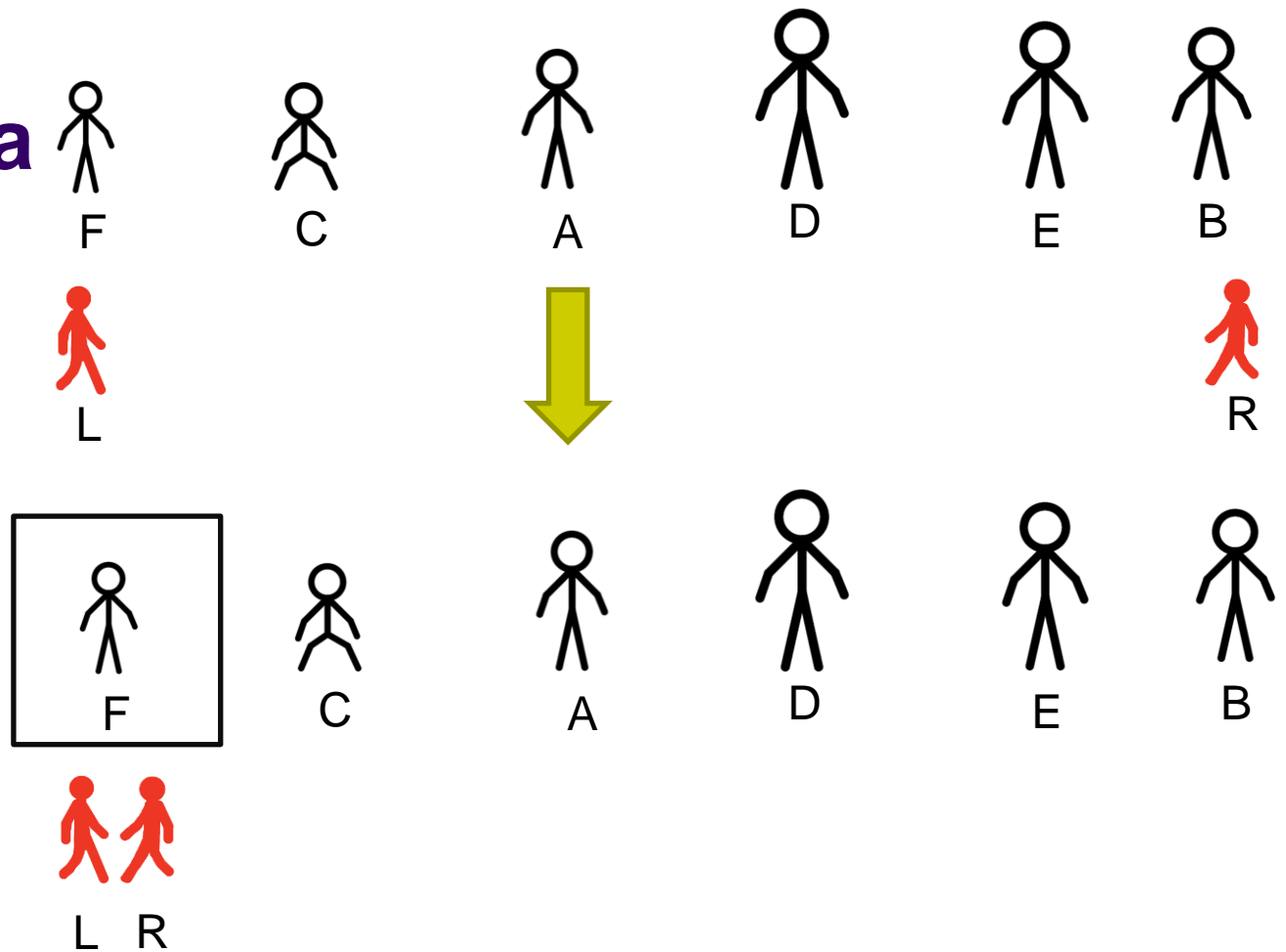


L1: select L4, L6  
L2: pivot  
L3: partition  
L4: squat  
L5: goto L1  
L6: halt

Jump to L1

Data group, L and R keep unchanged

## Step 6: Select a area



L1: select L4, L6

L2: pivot

L3: partition

L4: squat

L5: goto L1

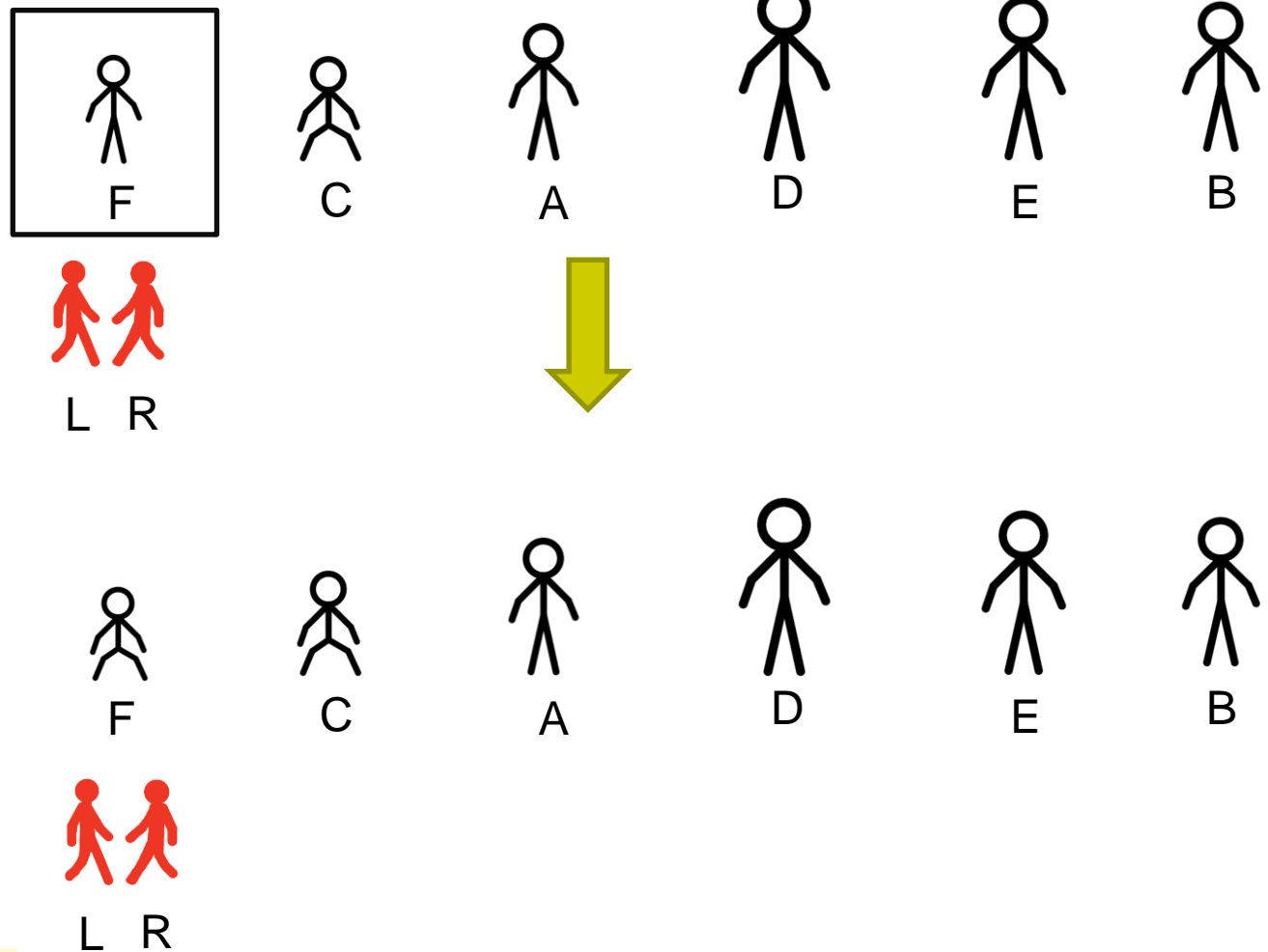
L6: halt

Select a new area

L equals to R, jump to L4



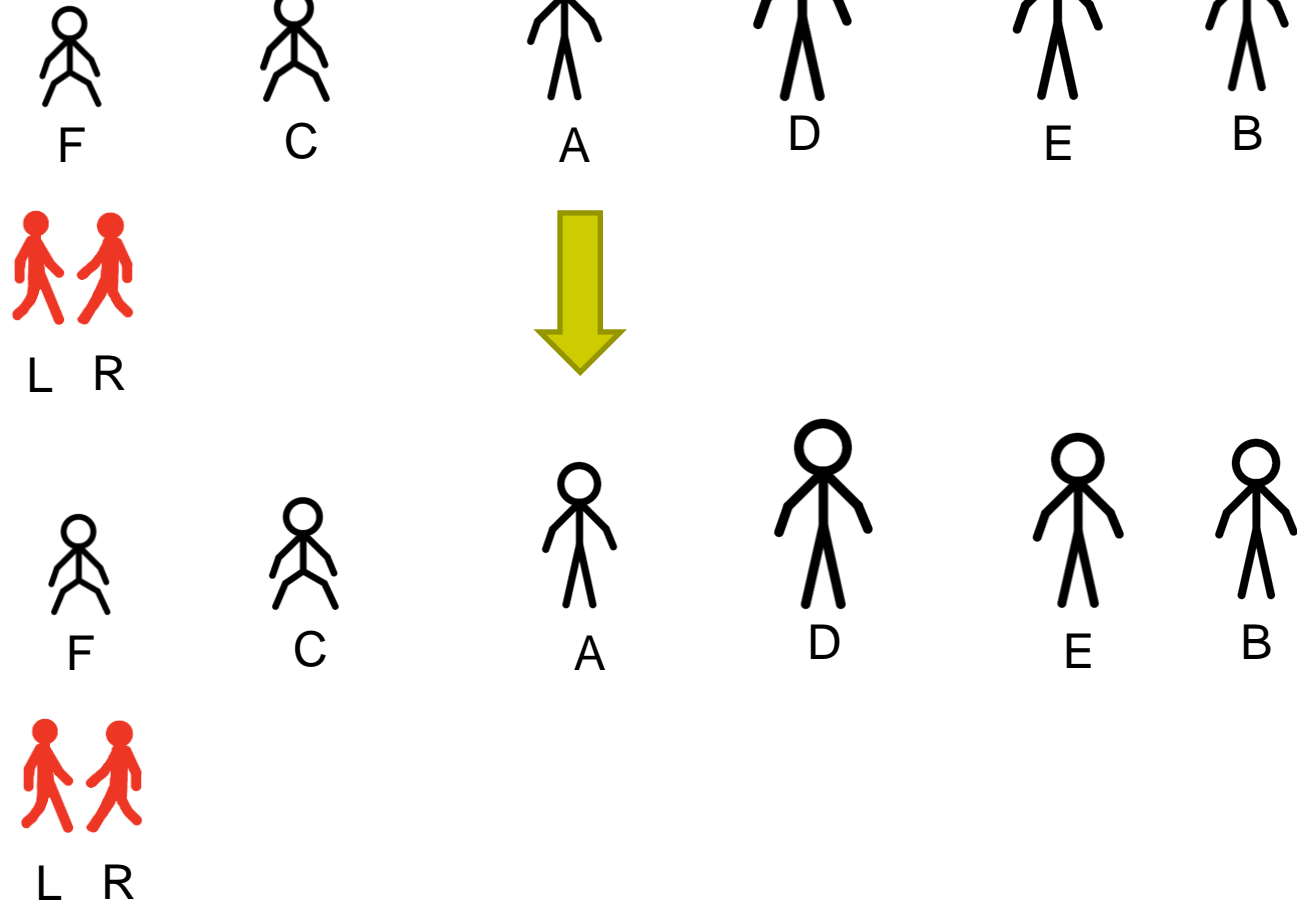
## Step 7: Squat



L1: select L4, L6  
L2: pivot  
L3: partition  
**L4: squat**  
L5: goto L1  
L6: halt

No one in [L, R] holds the flag up, so let all students (F) squat

# Step 8: Jump

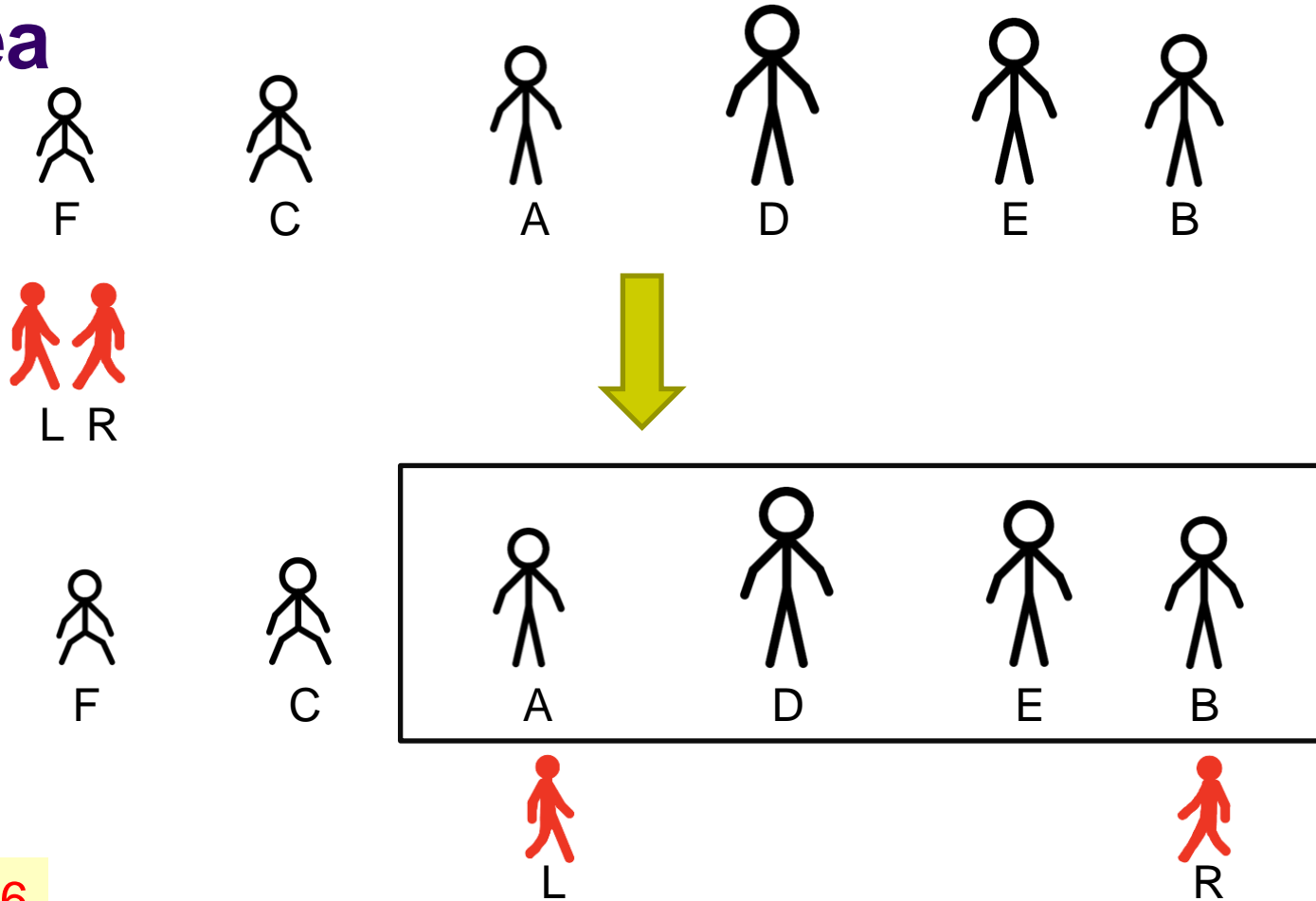


L1: select L4, L6  
L2: pivot  
L3: partition  
L4: squat  
**L5: goto L1**  
L6: halt

Jump to L1

Data group, L and R keep unchanged

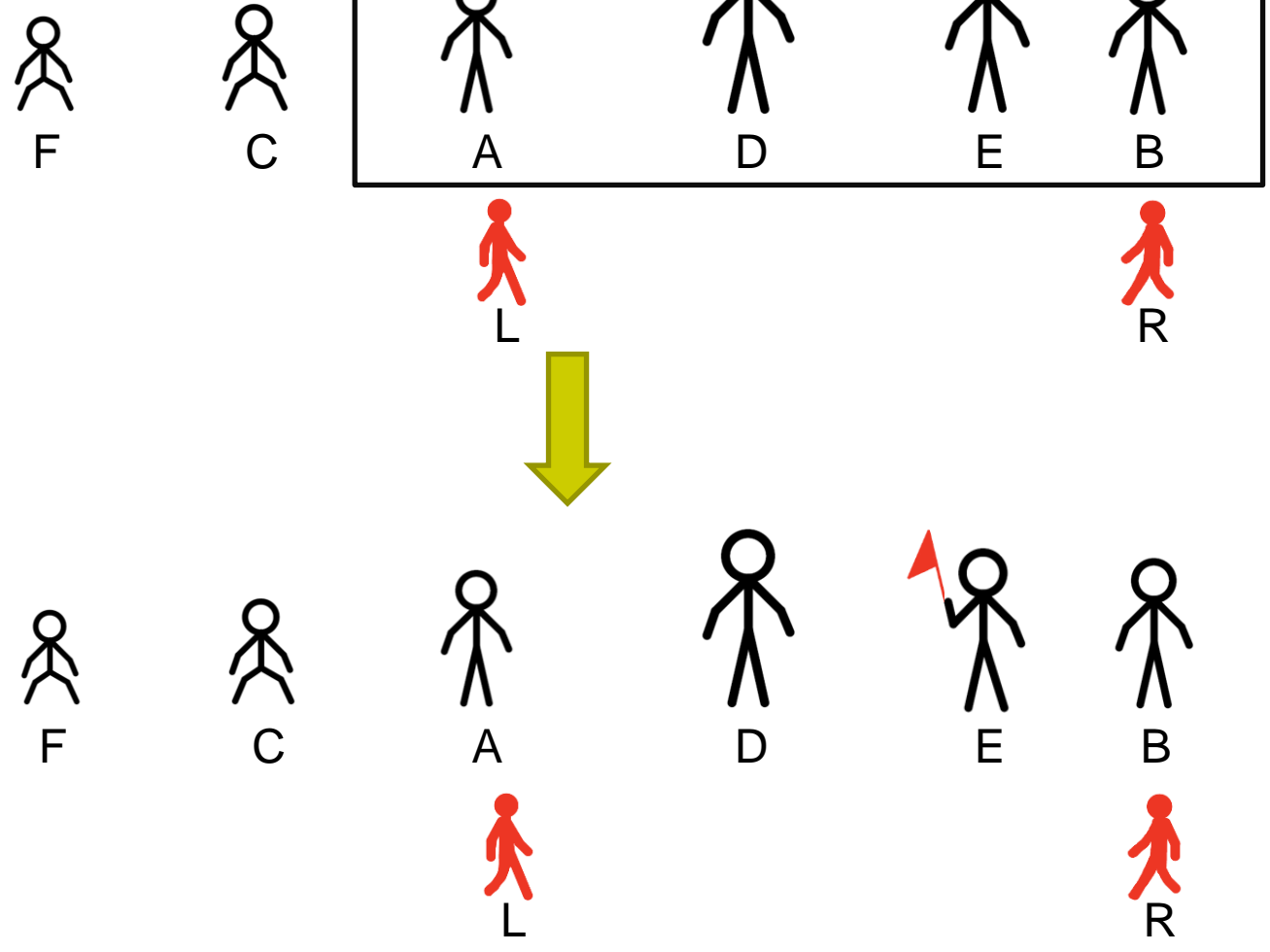
# Step 9: Select a area



$L = 3, R = 6$

L1: select L4, L6  
L2: pivot  
L3: partition  
L4: squat  
L5: goto L1  
L6: halt

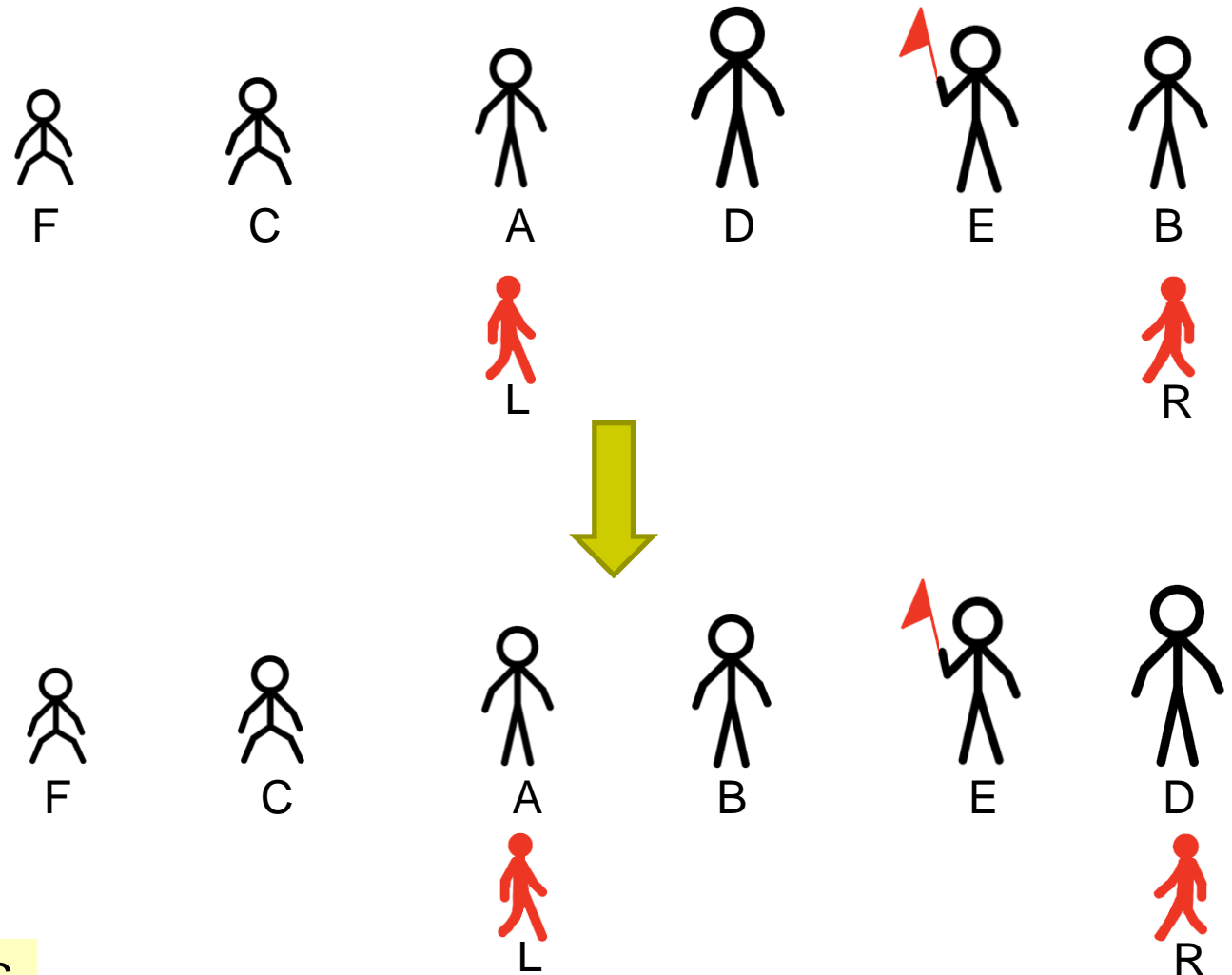
# Step 10: Select a pivot



- L1: select L4, L6
- L2: pivot
- L3: partition
- L4: squat
- L5: goto L1
- L6: halt

E is selected as the pivot

# Step 11: Partition

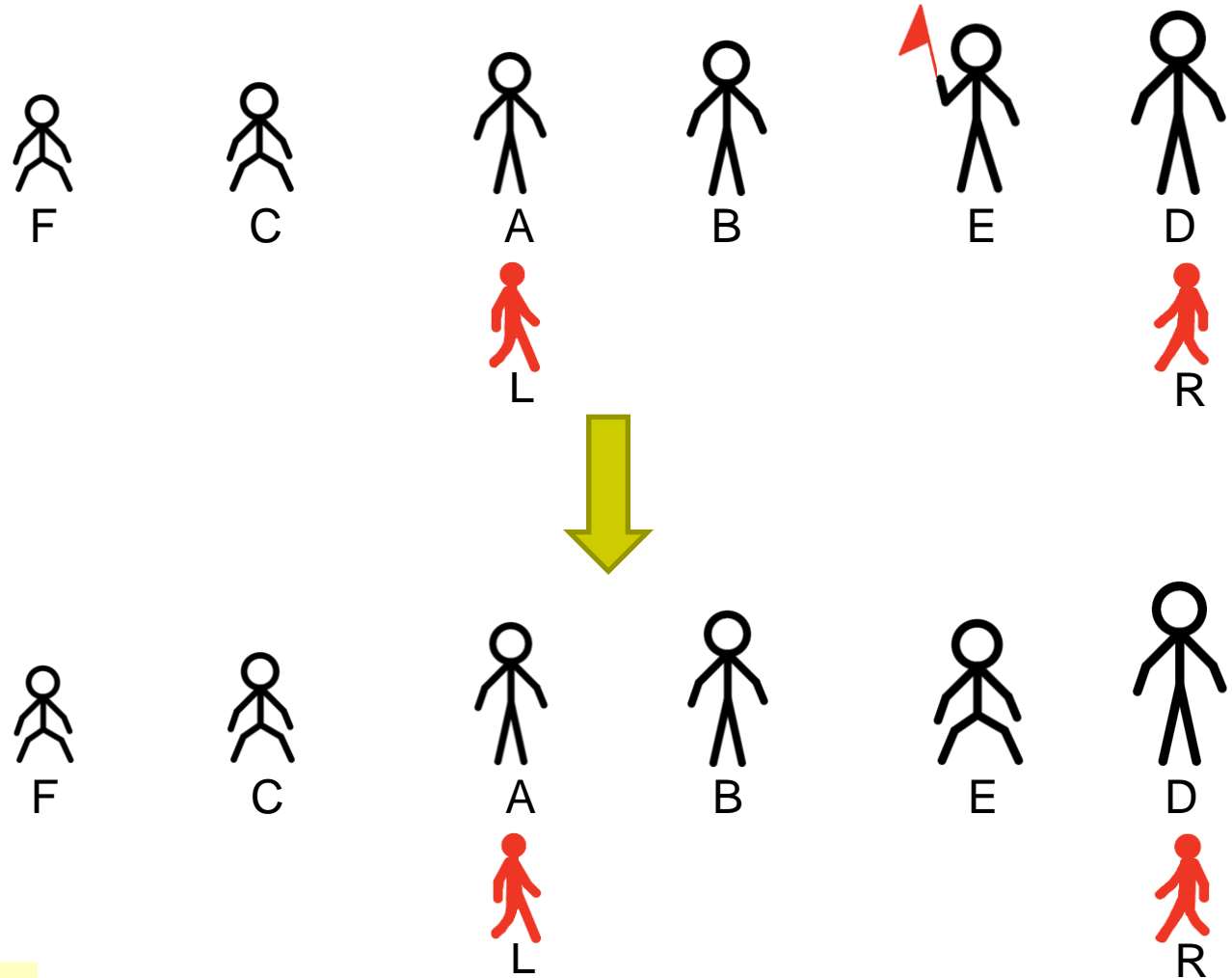


L1: select L4, L6  
L2: pivot  
L3: **partition**  
L4: squat  
L5: goto L1  
L6: halt

Adjust students in [L, R], students

- taller than E are on the right side of E
  - shorter than E are on the left side of E
- L and R keep unchanged

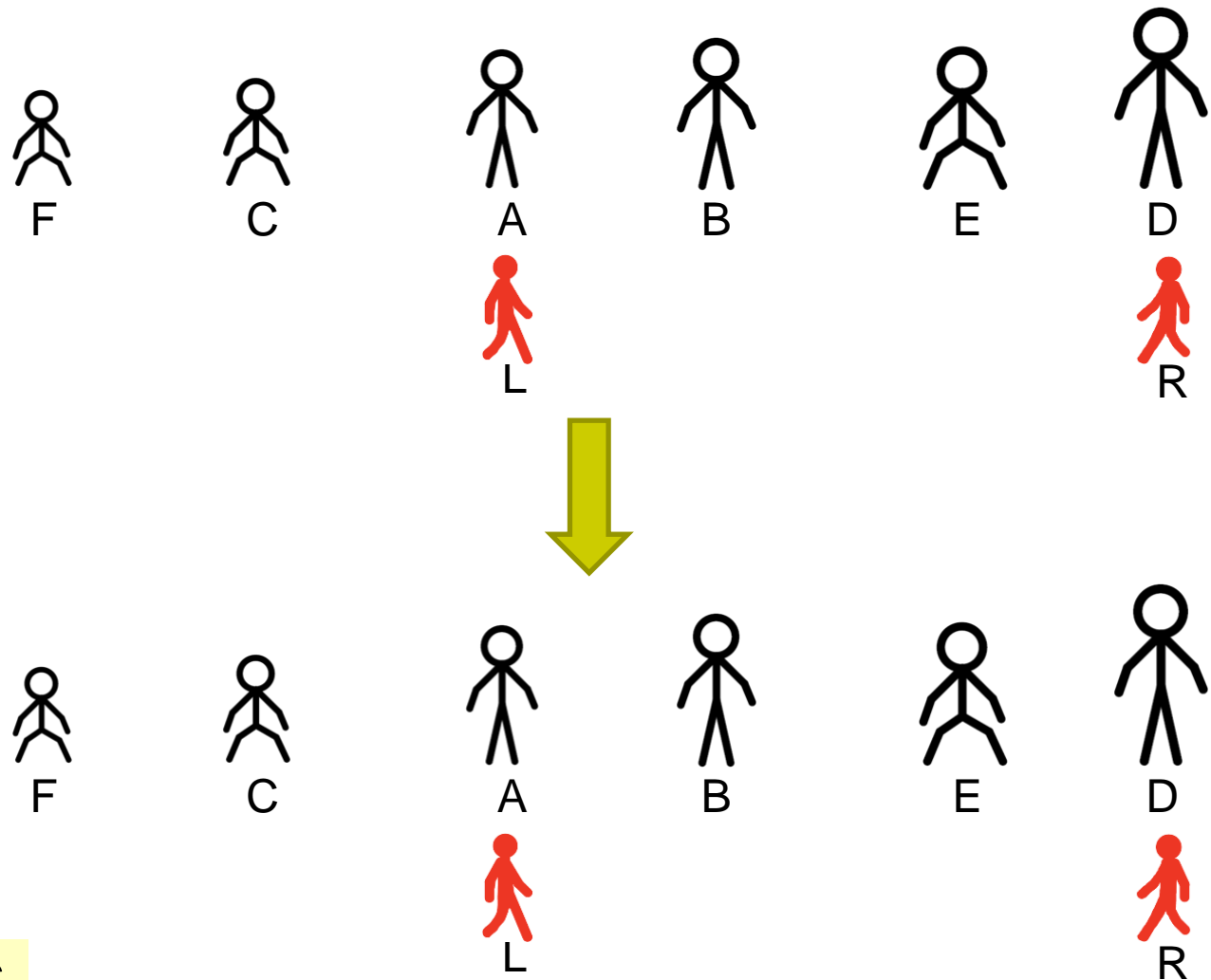
# Step 12: Squat



L1: select L4, L6  
L2: pivot  
L3: partition  
**L4: squat**  
L5: goto L1  
L6: halt

E lays the flag down and squats

# Step 13: Jump



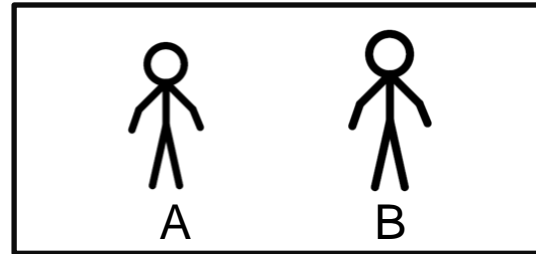
L1: select L4, L6  
L2: pivot  
L3: partition  
L4: squat  
**L5: goto L1**  
L6: halt

Jump to L1

Data group, L and R keep unchanged



# Step 14: Select a area



L1: select L4, L6

L2: pivot

L3: partition

L4: squat

L5: goto L1

L6: halt

L = 3, R = 4

# Step 15: Select a pivot

F

C

A

B

E

D

L

R



F

C

A

B

E

D

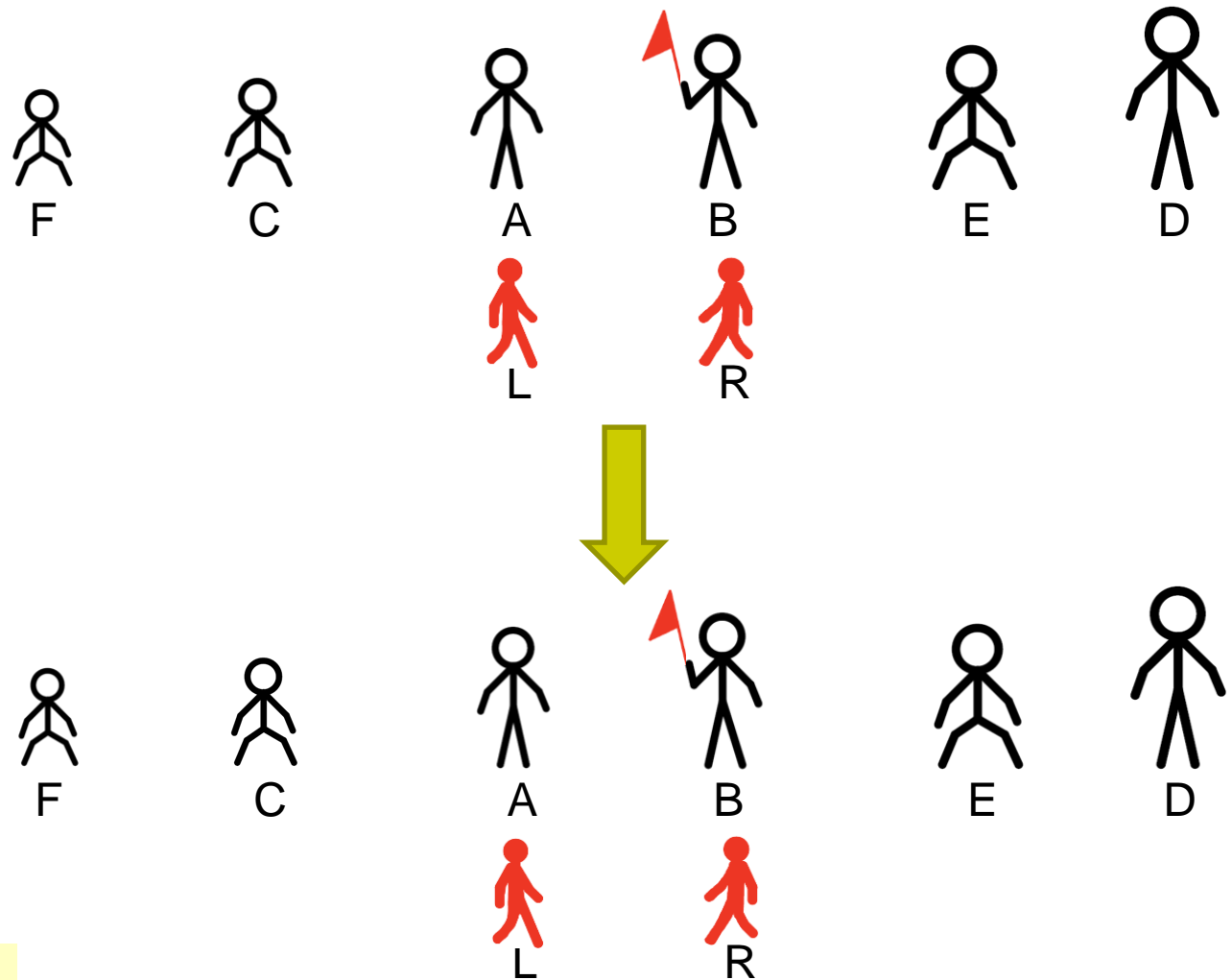
L

R

L1: select L4, L6  
L2: pivot  
L3: partition  
L4: squat  
L5: goto L1  
L6: halt

E is selected as the pivot

# Step 16: Partition

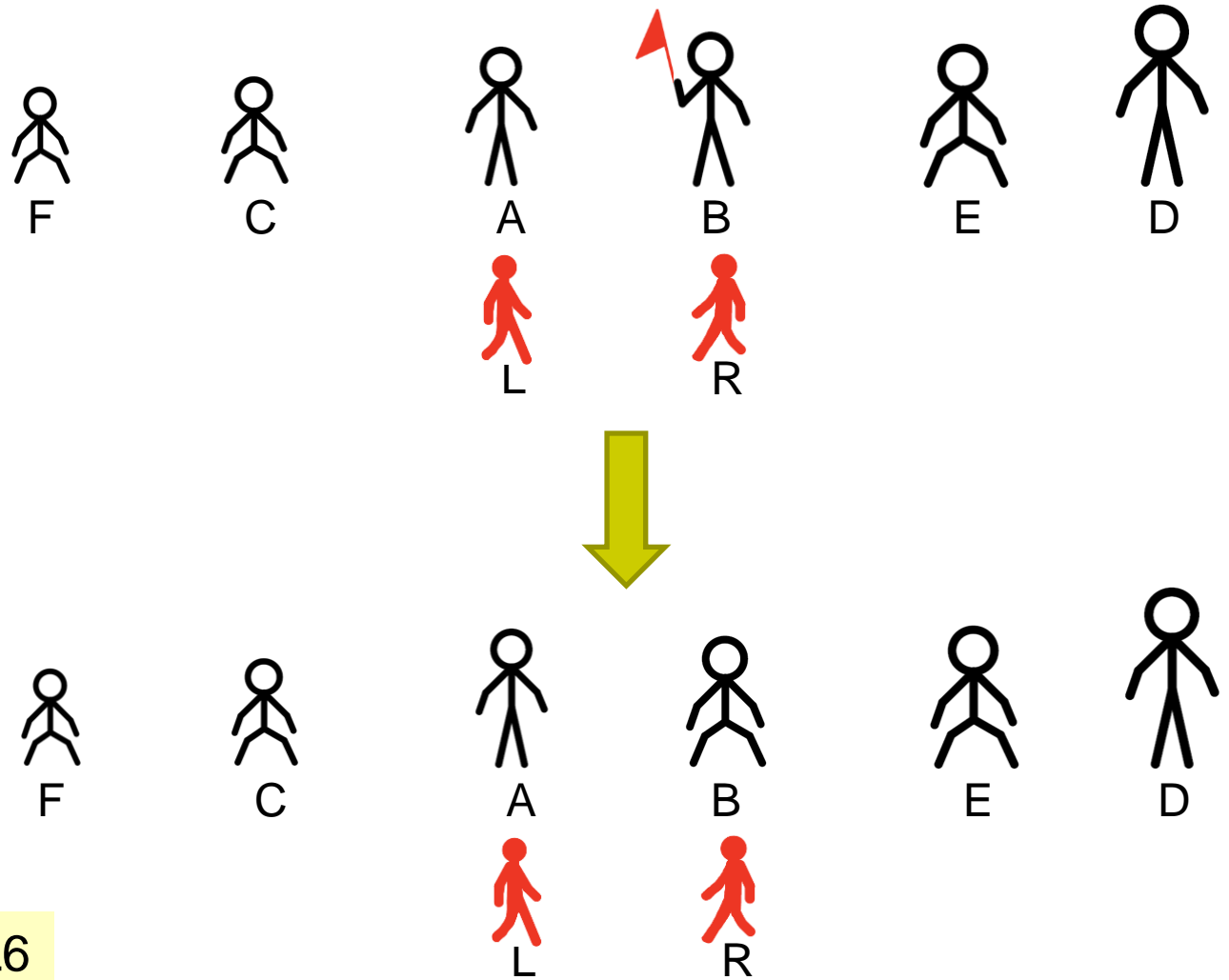


L1: select L4, L6  
L2: pivot  
L3: partition  
L4: squat  
L5: goto L1  
L6: halt

Adjust students in  $[L, R]$ , students

- taller than B are on the right side of B
  - shorter than B are on the left side of B
- L and R keep unchanged

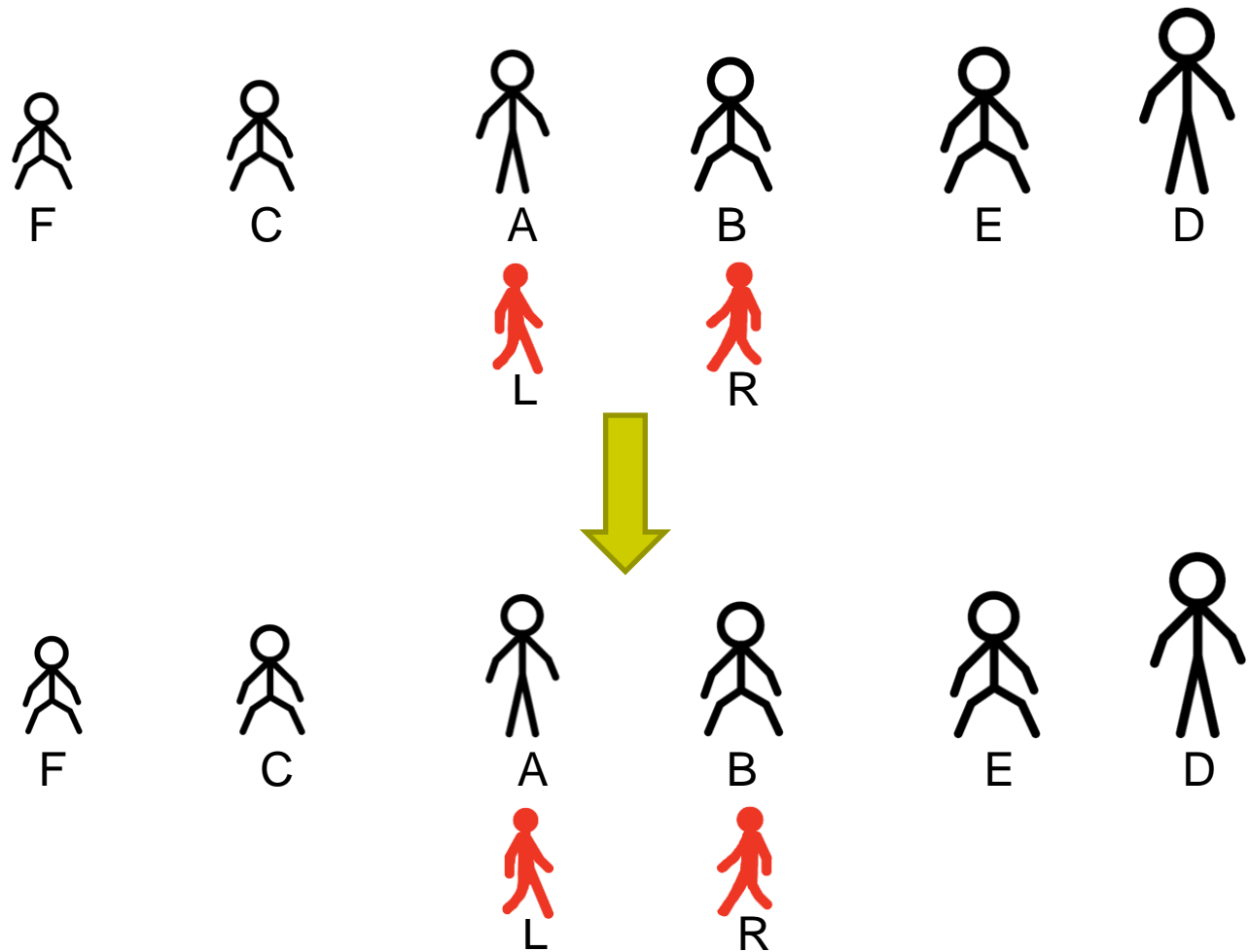
# Step 17: Squat



L1: select L4, L6  
L2: pivot  
L3: partition  
**L4: squat**  
L5: goto L1  
L6: halt

B lays the flag down and squats

# Step 18: Jump

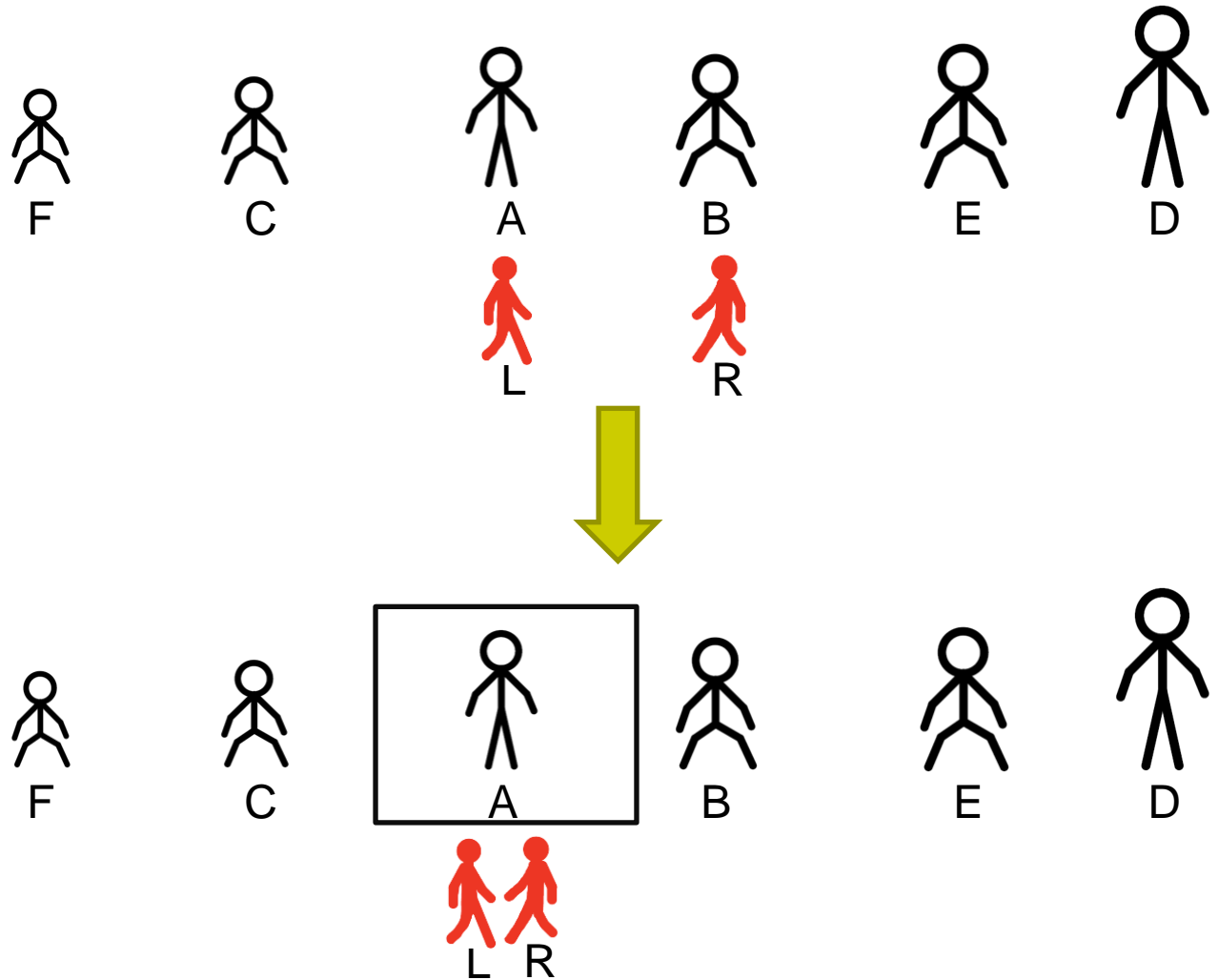


L1: select L4, L6  
L2: pivot  
L3: partition  
L4: squat  
**L5: goto L1**  
L6: halt

Jump to L1

Data group, L and R keep unchanged

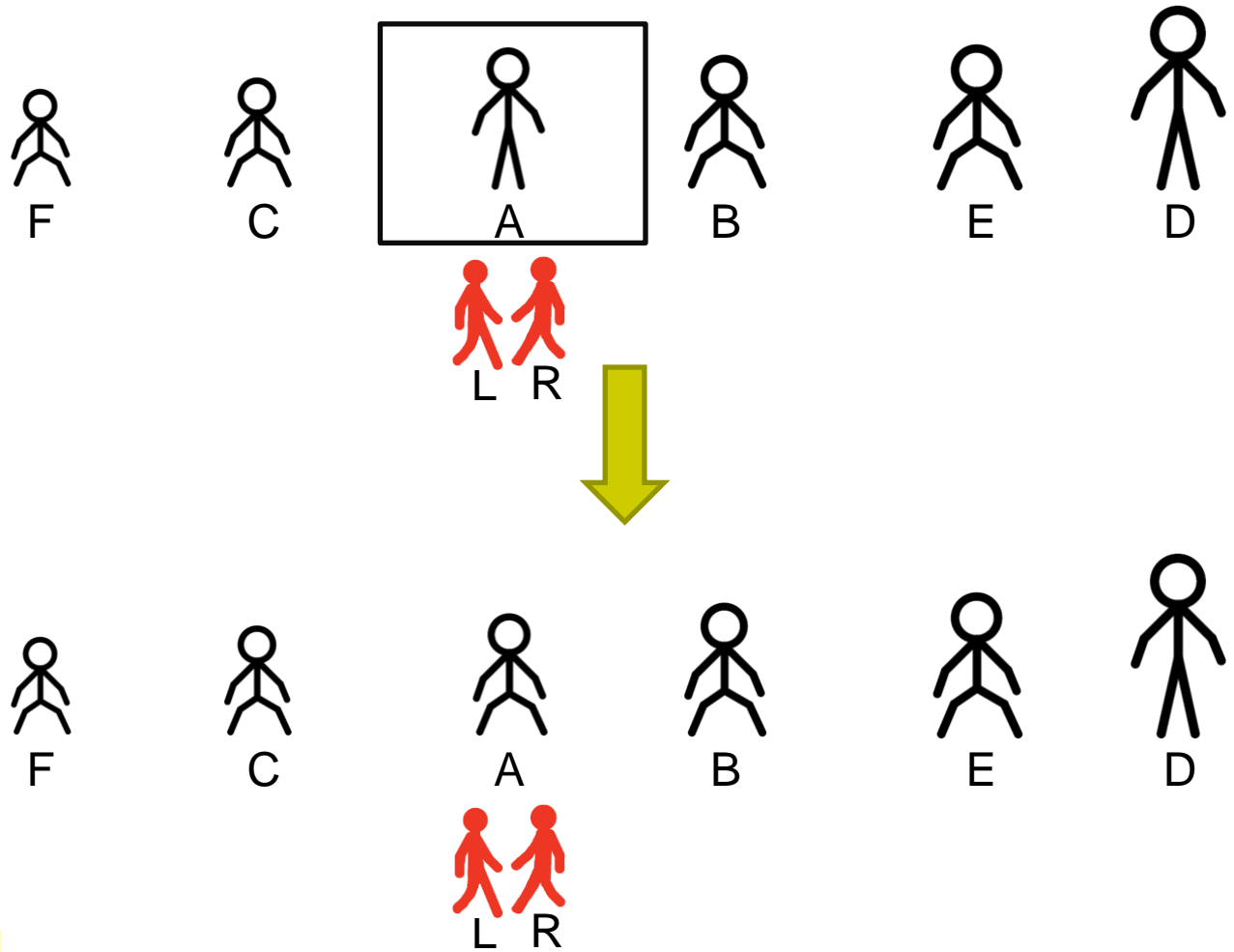
# Step 19: Select a area



L1: select L4, L6  
L2: pivot  
L3: partition  
L4: squat  
L5: goto L1  
L6: halt

Select a new area  
L equals to R, jump to L4

# Step 20: Squat

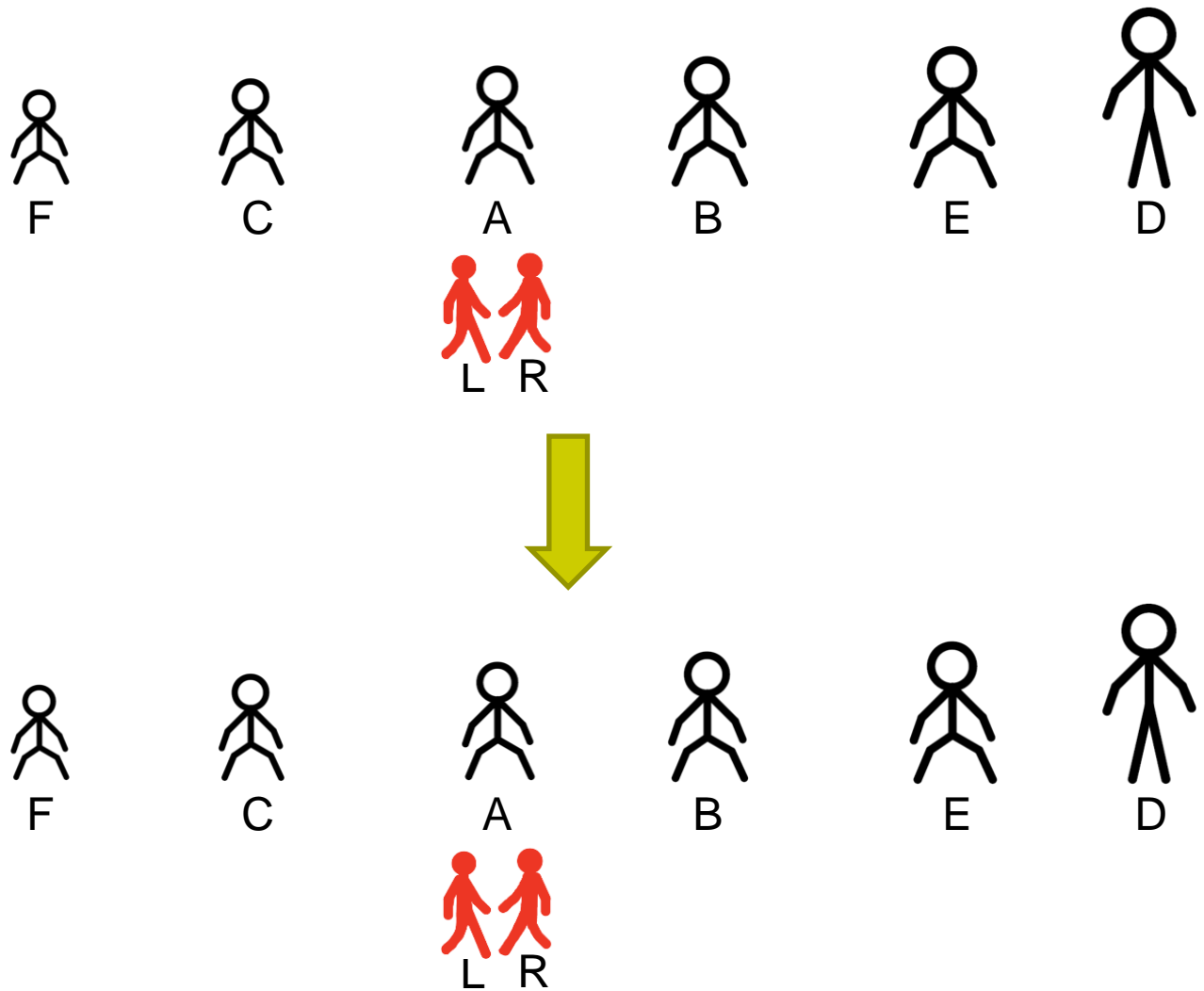


L1: select L4, L6  
L2: pivot  
L3: partition  
**L4: squat**  
L5: goto L1  
L6: halt

A squats



# Step 21: Jump

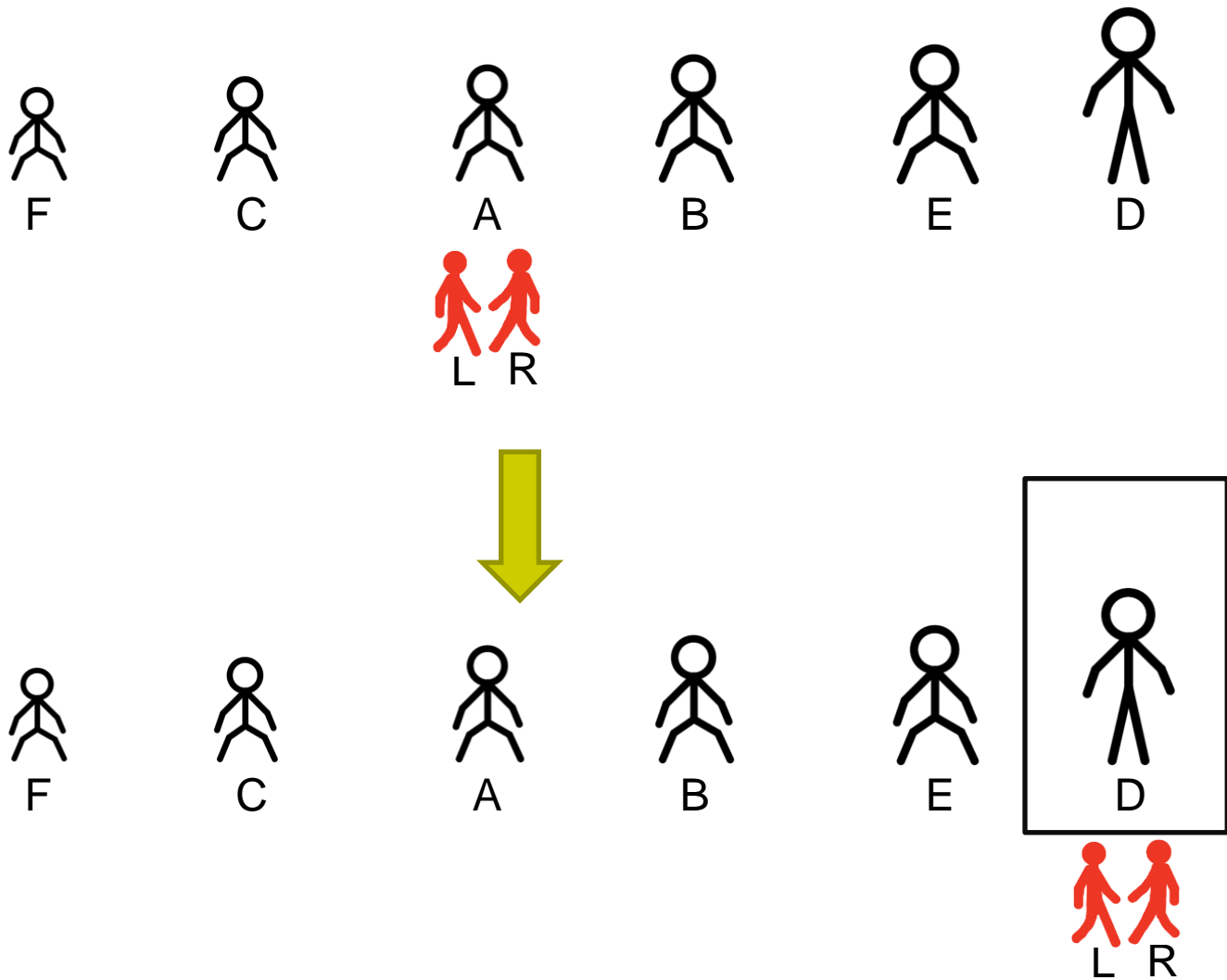


L1: select L4, L6  
L2: pivot  
L3: partition  
L4: squat  
**L5: goto L1**  
L6: halt

Jump to L1

Data group, L and R keep unchanged

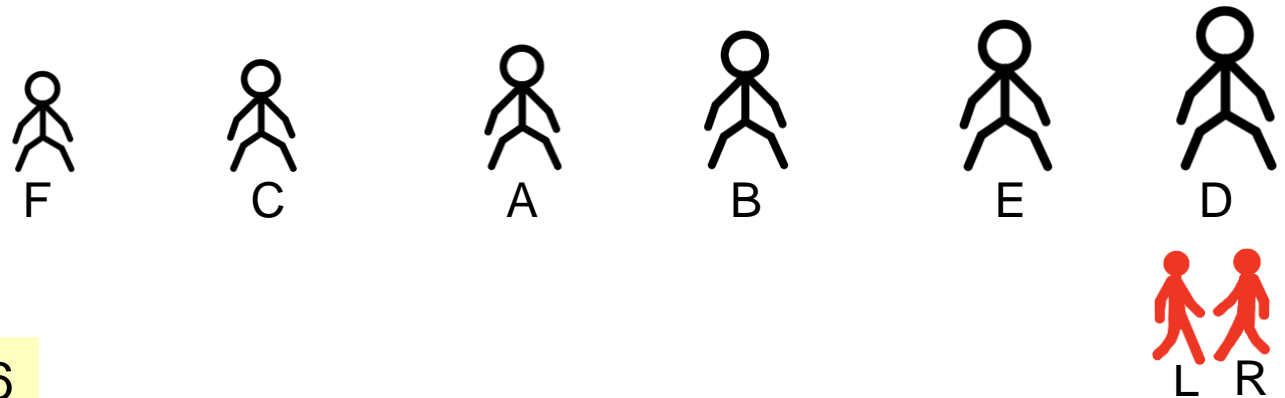
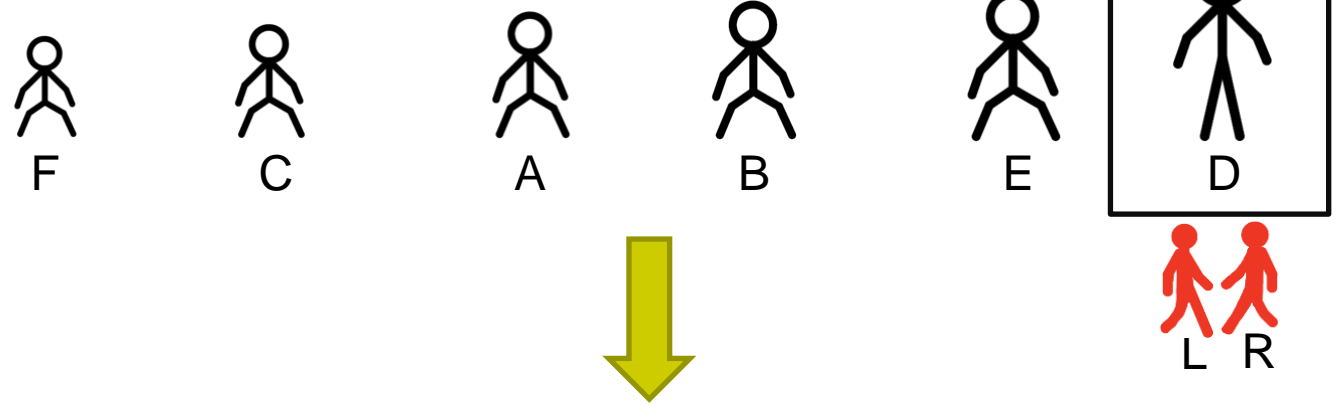
# Step 22: Select a area



- L1: select L4, L6
- L2: pivot
- L3: partition
- L4: squat
- L5: goto L1
- L6: halt

$L = R = 6$ , jump to L4

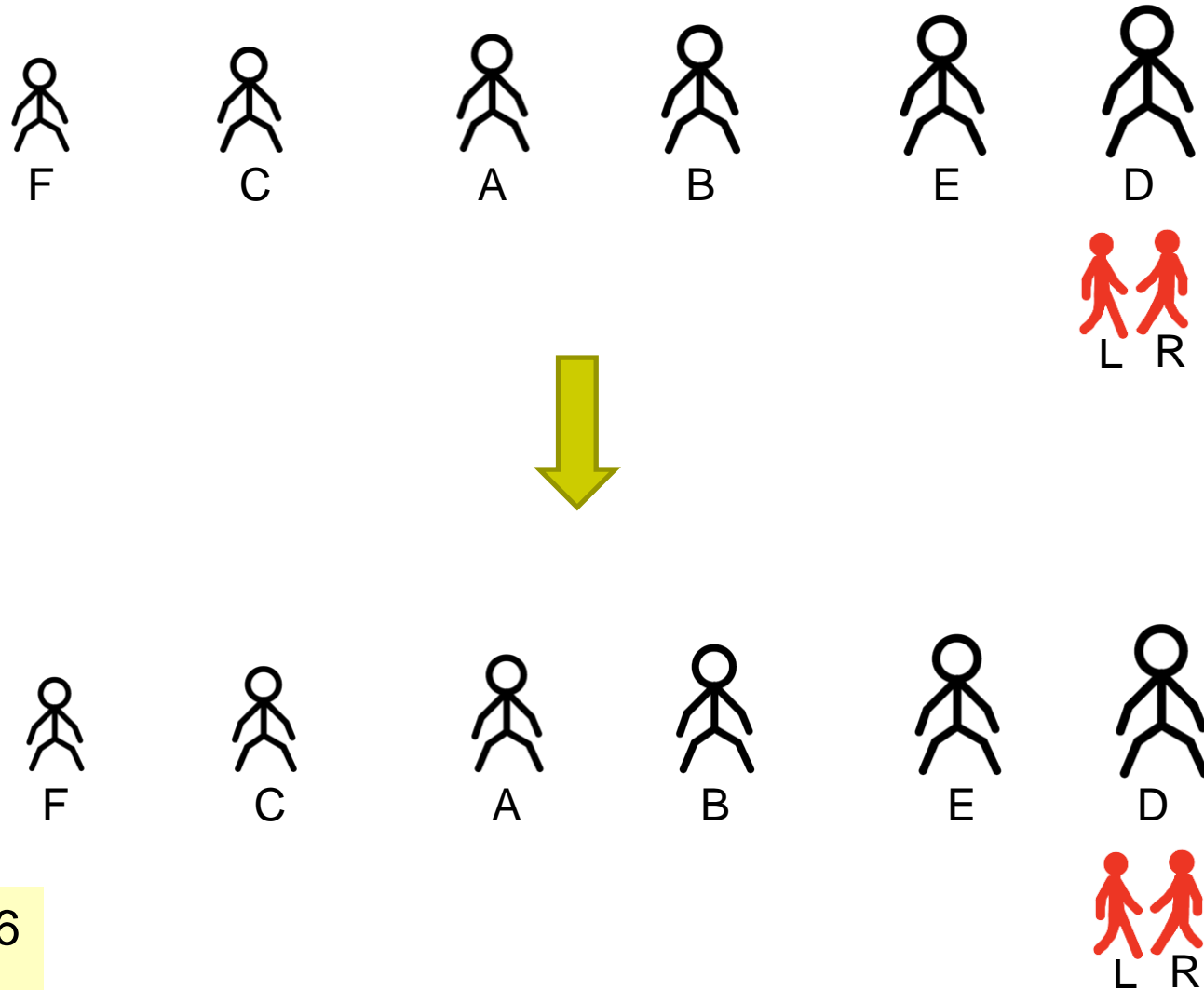
# Step 23: Squat



L1: select L4, L6  
L2: pivot  
L3: partition  
**L4: squat**  
L5: goto L1  
L6: halt

D squats

# Step 24: Jump



L1: select L4, L6  
L2: pivot  
L3: partition  
L4: squat  
**L5: goto L1**  
L6: halt

Jump to L1

Data group, L and R keep unchanged

# Step 25:

Select a area



F



C



A



B



E



D



L

R



F



C



A



B



E



D



L

R

L1: select L4, L6

L2: pivot

L3: partition

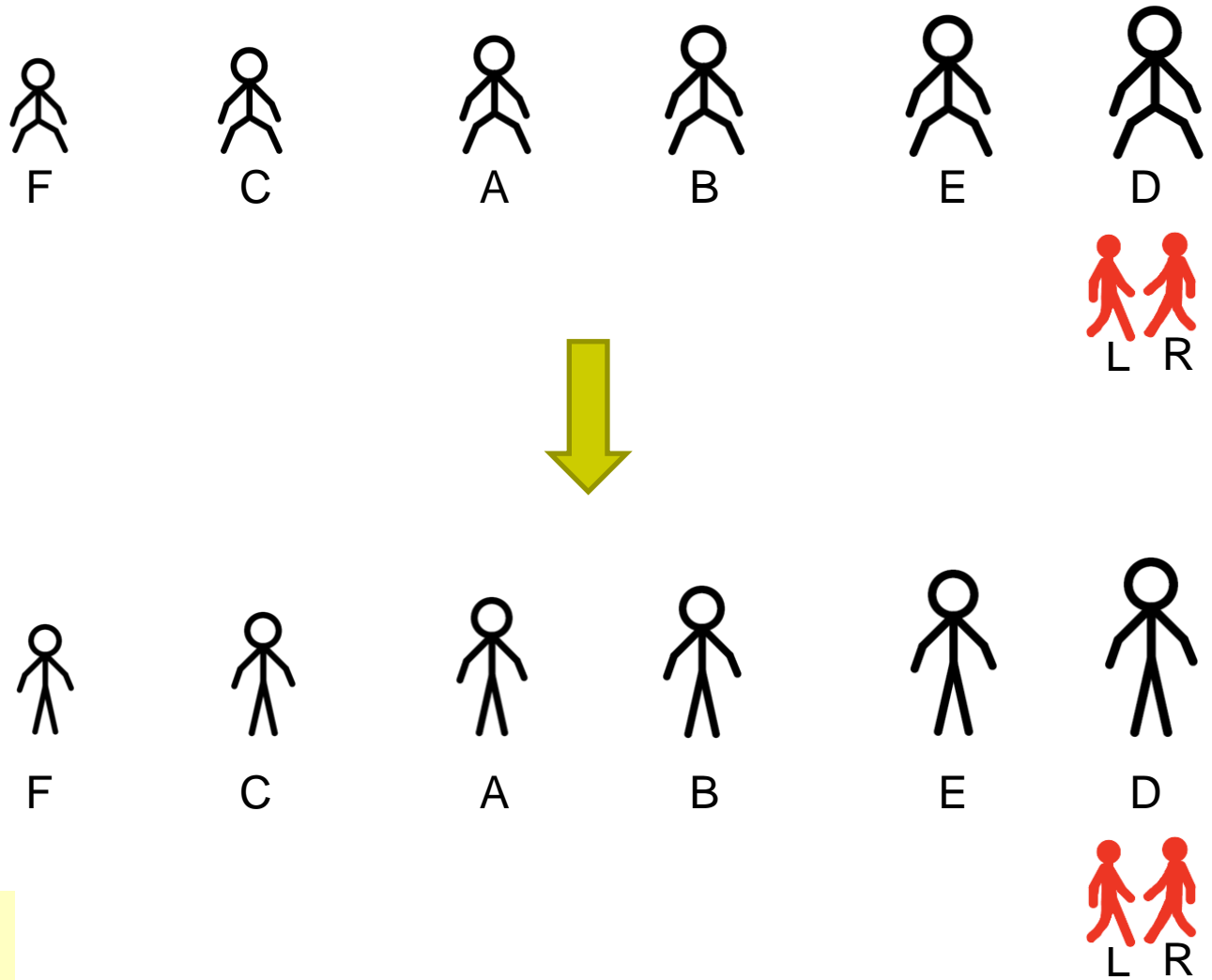
L4: squat

L5: goto L1

L6: halt

Because all students in data group squat, jump to L6

# Step 26: Jump

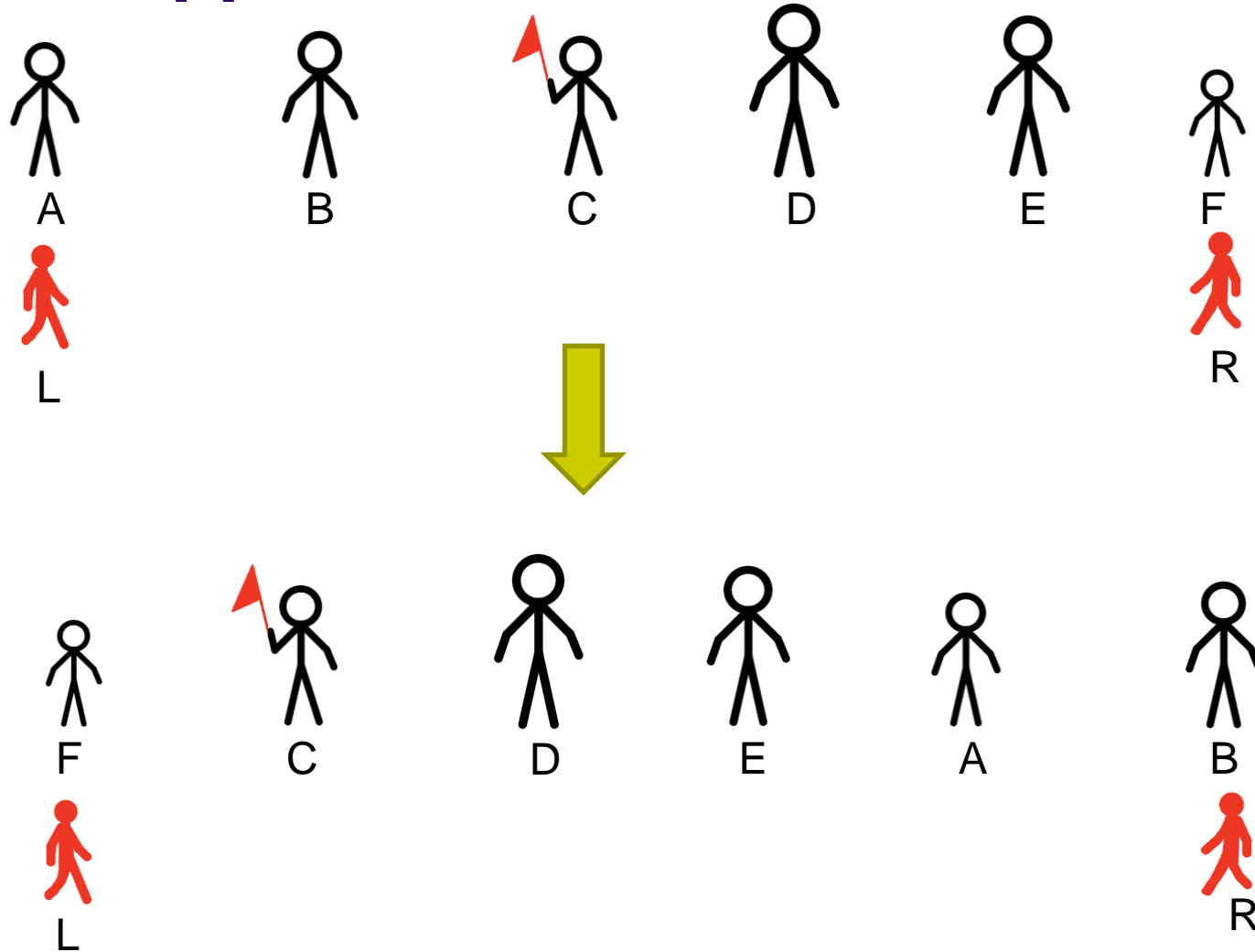


L1: select L4, L6  
L2: pivot  
L3: partition  
L4: squat  
L5: goto L1  
L6: halt

All students in data group stand up and the program terminates.

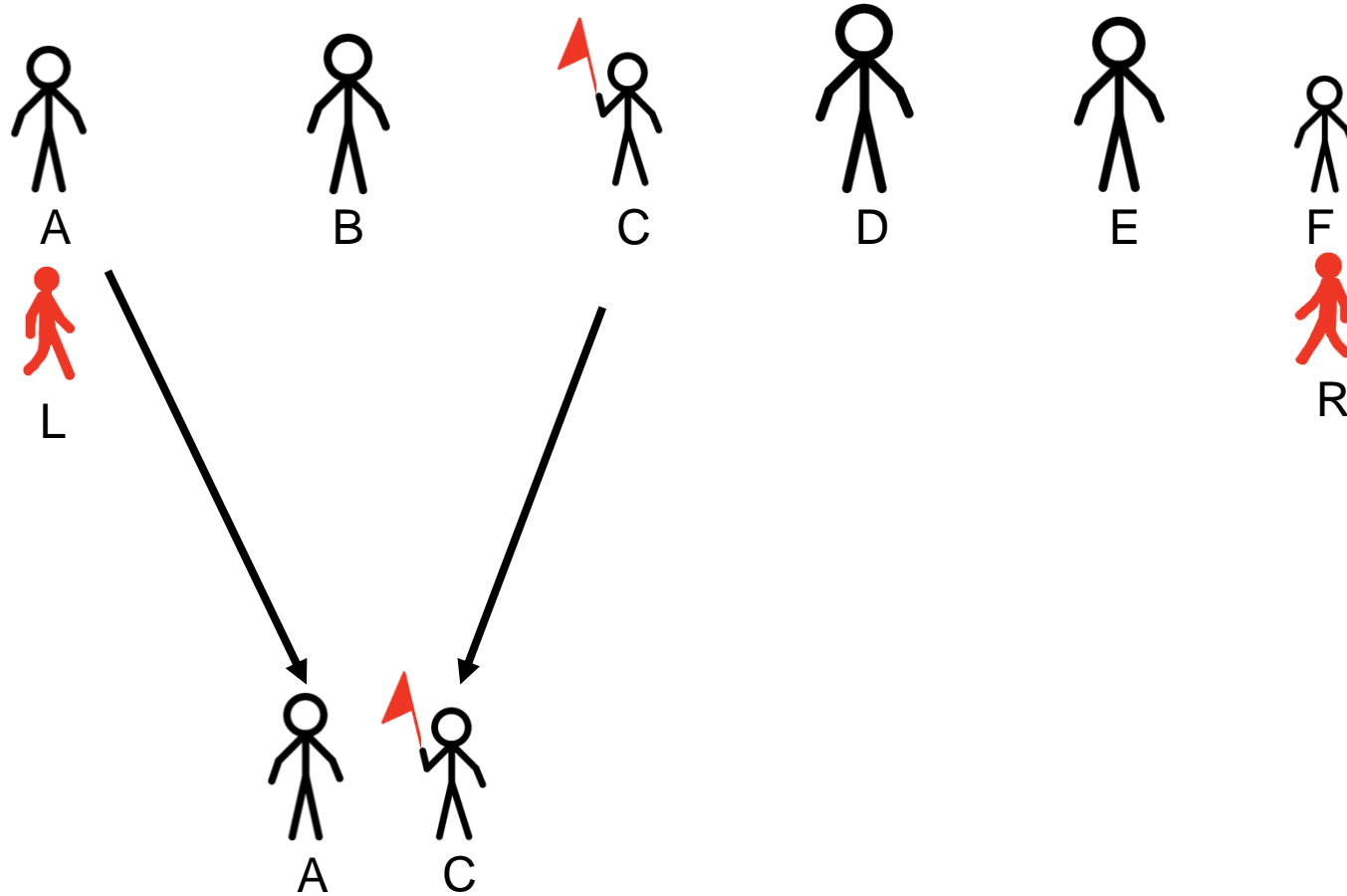
# Revisit partition:

What happens when there are 100 students?



99 students are needed to compare with the pivot at one step, which will cause disorder!!

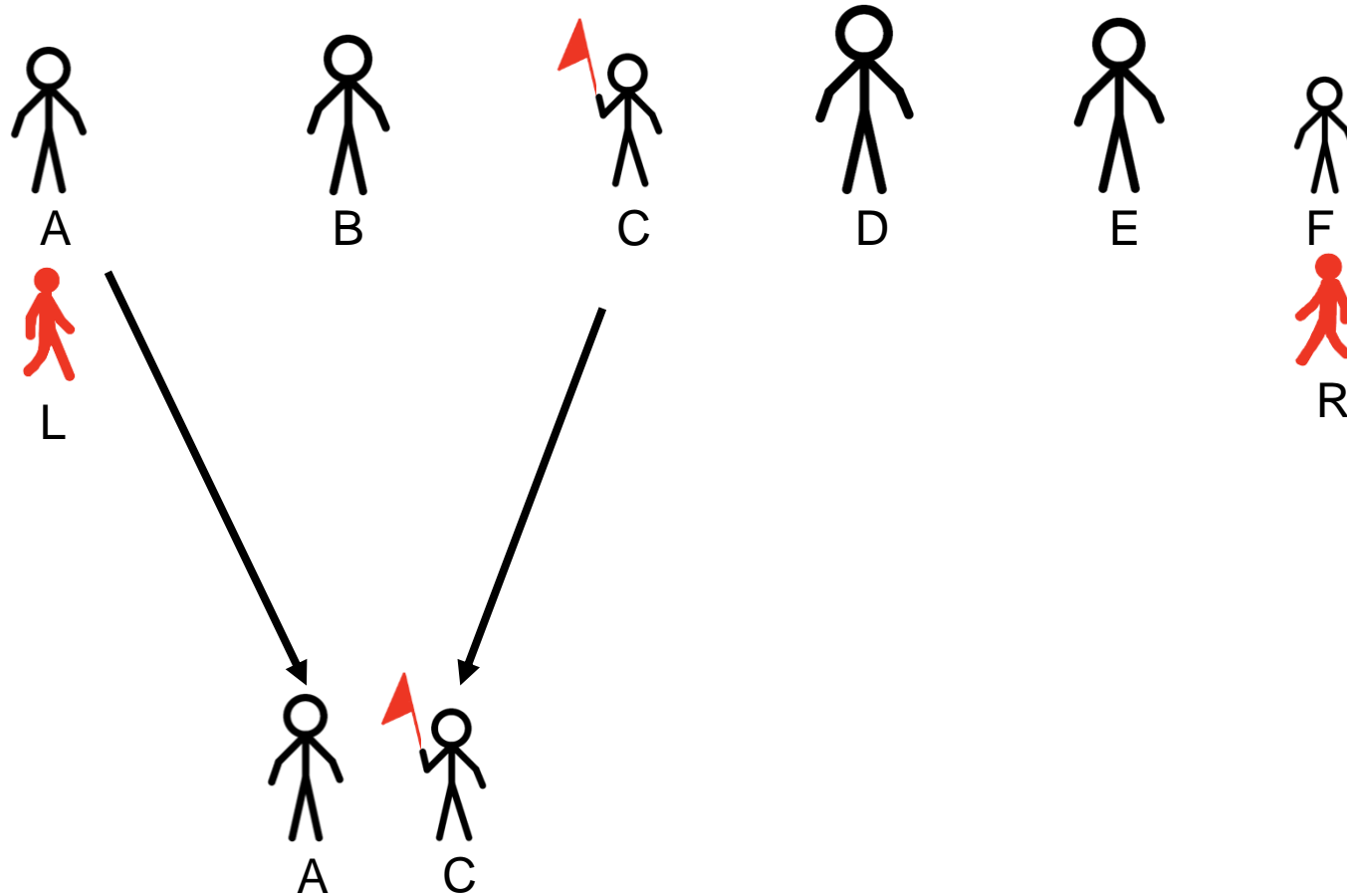
# Pairwise comparison



A is higher than C and needs to move to the right side of C



# Pairwise comparison



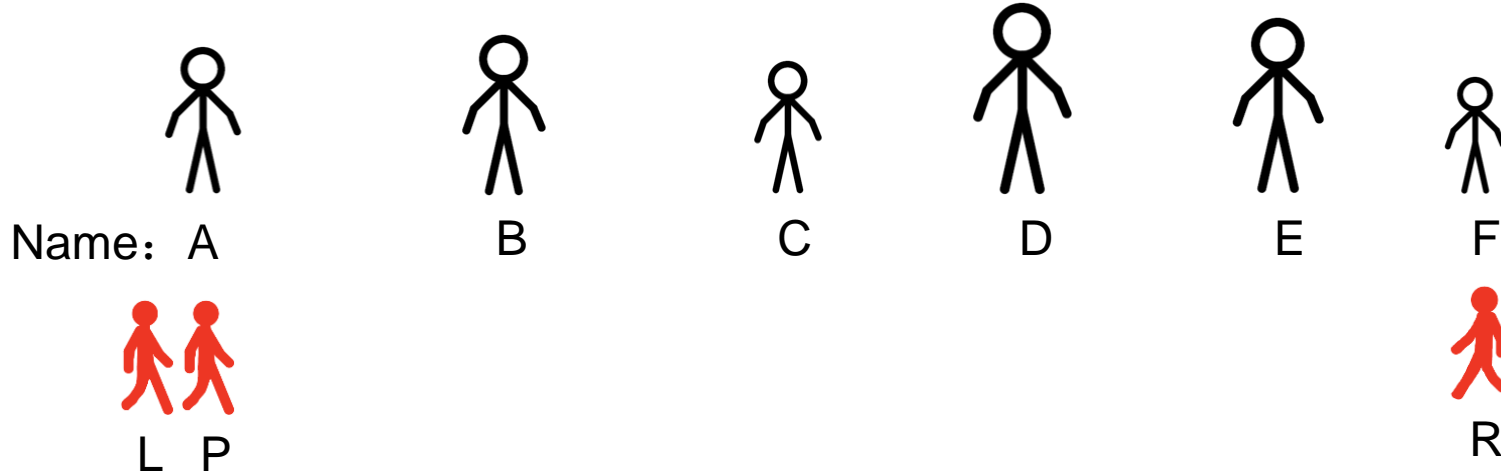
How A move to the right side of C?

Does the position of the others in data group change?

Do the values of L and R change?

# Hardware design

- Data Group: A, B, C, D, E, F
- Registers: L, R, **P**
  - $L = P = 1, R = 6$
- Others: Controller, Monitor, Overseer, Stepper
  - Not shown in this slice



**Improve the design**

# Instruction set design

- Original instructions:
  - *partition* is deleted

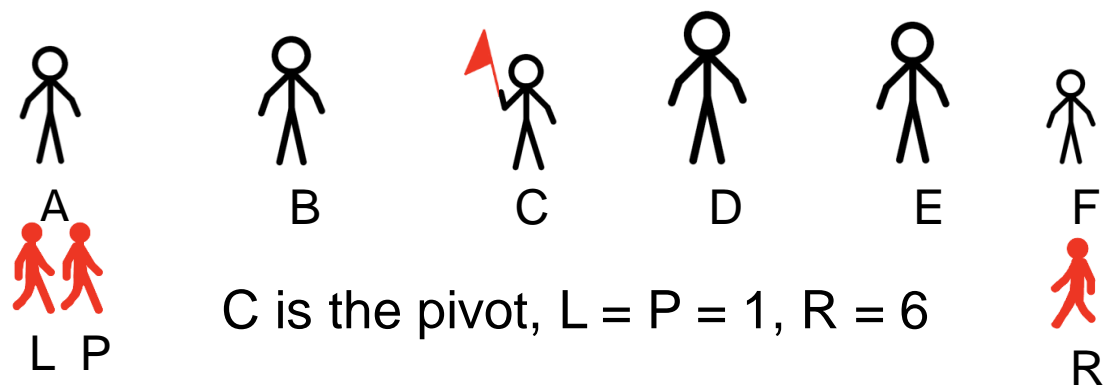
Opcode	Operand1	Operand2	Explanatory remarks
select	label1	label2	Select the leftmost area as large as possible where everyone is standing, let $L = P$ = index of the leftmost, $R$ = index of the rightmost. If $L$ equals to $R$ , jump to label 1. If no such area exists, jump to label2.
pivot			Random select a student in $[L, R]$ as the pivot. The pivot need hold the flag up.
squat			Within $[L, R]$ : <ul style="list-style-type: none"><li>• Let the student holding the flag (pivot) lay the flag down, and squat;</li><li>• If no one holding the flag, let all students squat.</li></ul>
goto	label1		Jump to label1.
halt			All students in data group stand up and the program terminates.

# Instruction set design

- New instructions

Opcode	Operand1	Operand2	Explanatory remarks
cmp	label1	label2	If P equals to R, jump to label1; If the student that P points to is not shorter than the student that R points to, jump to label2.
inc	reg		$\text{reg} = \text{reg} + 1$
swap	reg1	reg2	Swap the students that reg1 and reg2 point to. reg1 could be “flag”, which points to the pivot.

# Improved Quicksort Program



L1:	select	L10, L12	# select a area
L2:	pivot		# select a pivot
L3:	swap	flag, R	# swap the pivot to where R point to
L4:	cmp	L9, L7	# if P equals to R, jump to L9; # If the student that P points to is not shorter # than the student that R points to, jump to L7
L5:	swap	P, L	# swap the students that P and L point to
L6:	inc	L	# $L = L + 1$
L7:	inc	P	# $P = P + 1$
L8:	goto	L4	# jump to L4
L9:	swap	flag, L	# swap the pivot to where R point to
L10:	squat		# squat
L11:	goto	L1	# jump to L1
L12:	halt		# All students stand up and the program halts.

# Compare the Two Quicksort Programs

L1: select L4, L6  
L2: pivot  
L3: partition  
L4: squat  
L5: goto L1  
L6: halt



L1: select L10, L12

L2: pivot

L3: swap flag, R

**L4:** cmp **L9, L7**

L5: swap P, L

L6: inc L

**L7:** inc P

L8: goto **L4**

**L9:** swap flag, L

L10: squat

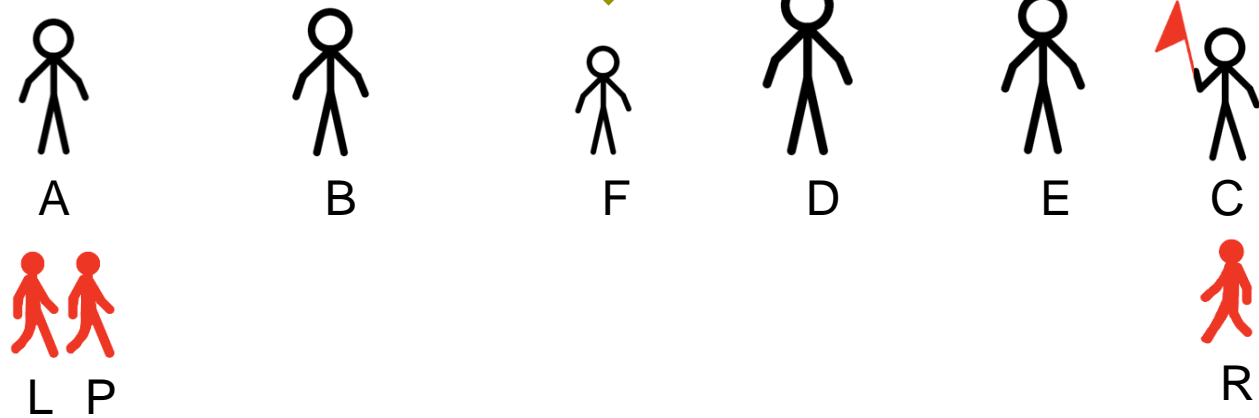
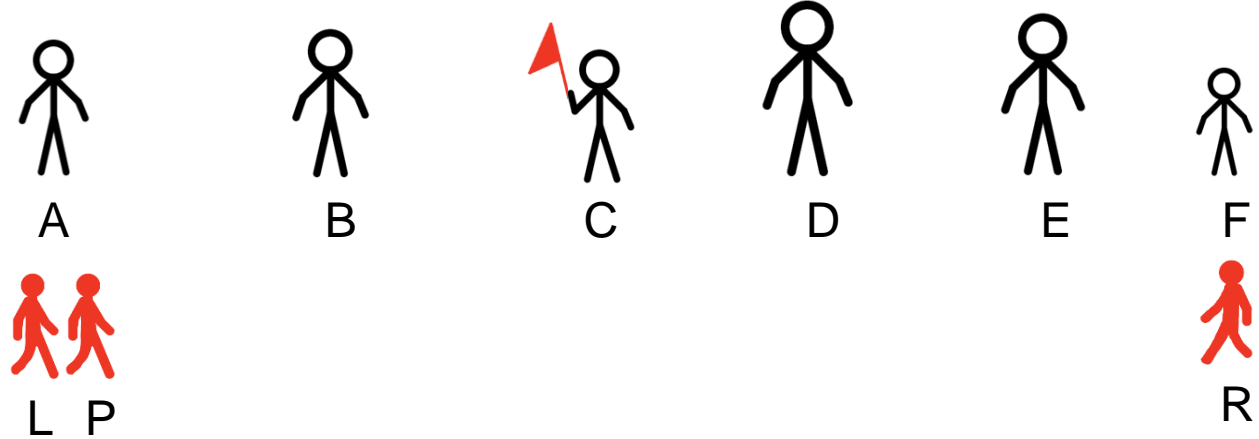
L11: goto L1

L12: halt

**Code in the box implements the partition instruction**  
The following slices show how to complete one partition

# Step 3: Swap

Already executed:  
Step 1: select a area  
Step 2: select a pivot

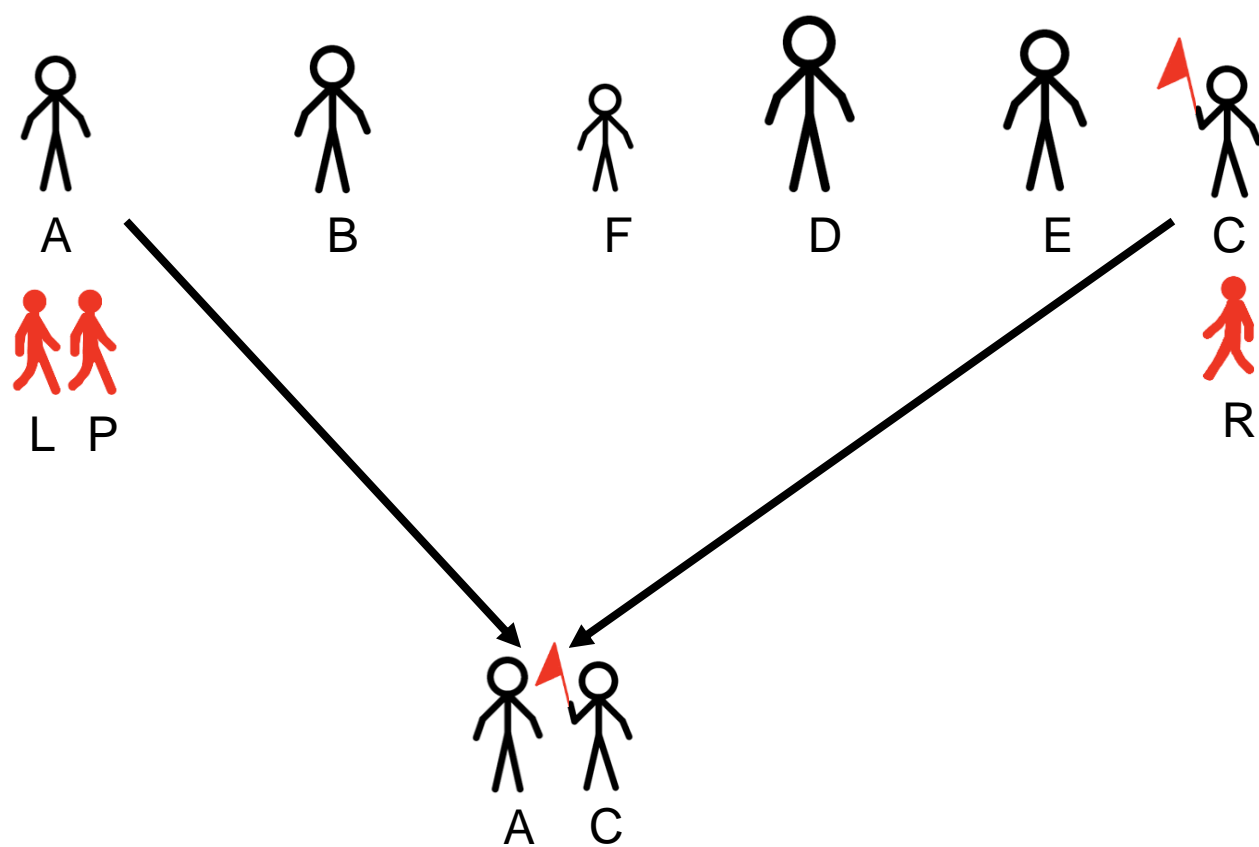


```
L3: swap  flag, R
L4: cmp   L9, L7
L5: swap  P, L
L6: inc   L
L7: inc   P
L8: goto  L4
L9: swap  flag, L
```

The pivot C is swapped to R



## Step 4: Compare



L3: swap flag, R

L4: cmp L9, L7

L5: swap P, L

L6: inc L

L7: inc P

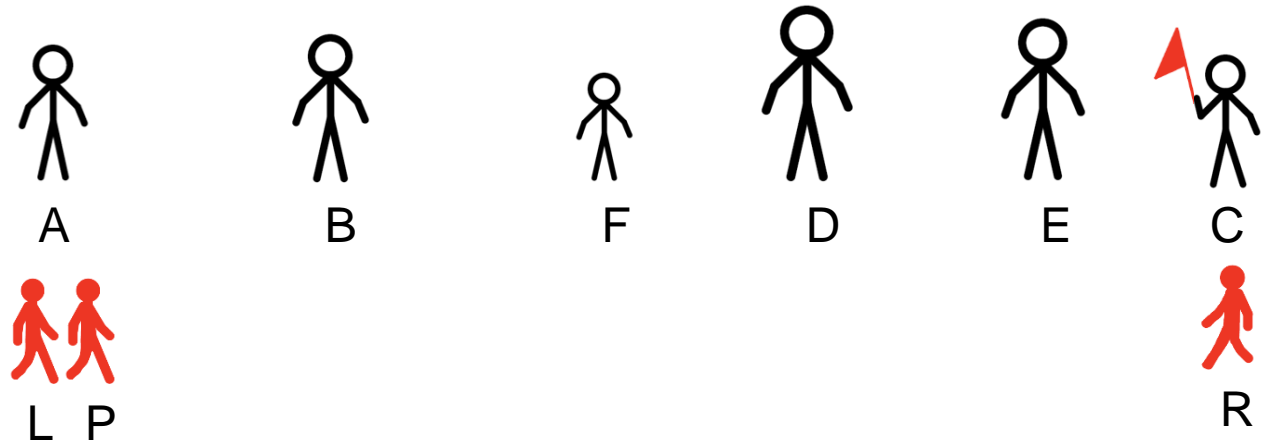
L8: goto L4

L9: swap flag, L

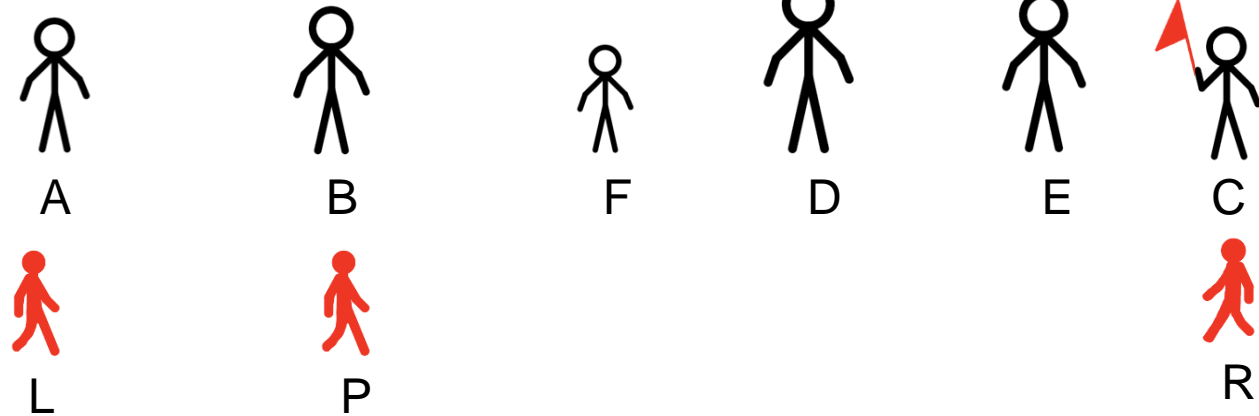
$P \neq R$  and A is taller than C, jump to L7

# Step 5:

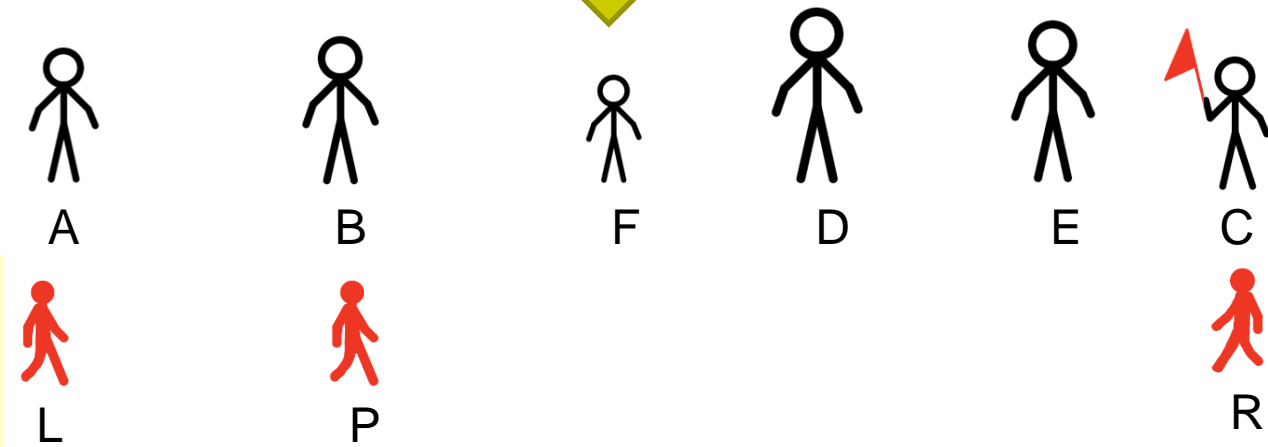
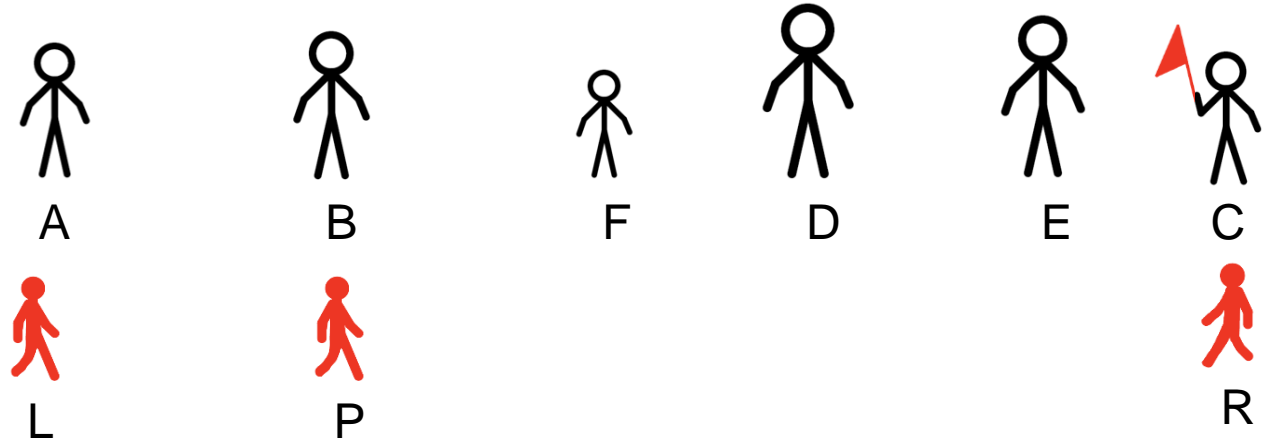
## P moves 1 position to right



L3: swap flag, R  
L4: cmp L9, L7  
L5: swap P, L  
L6: inc L  
**L7: inc P**  
L8: goto L4  
L9: swap flag, L



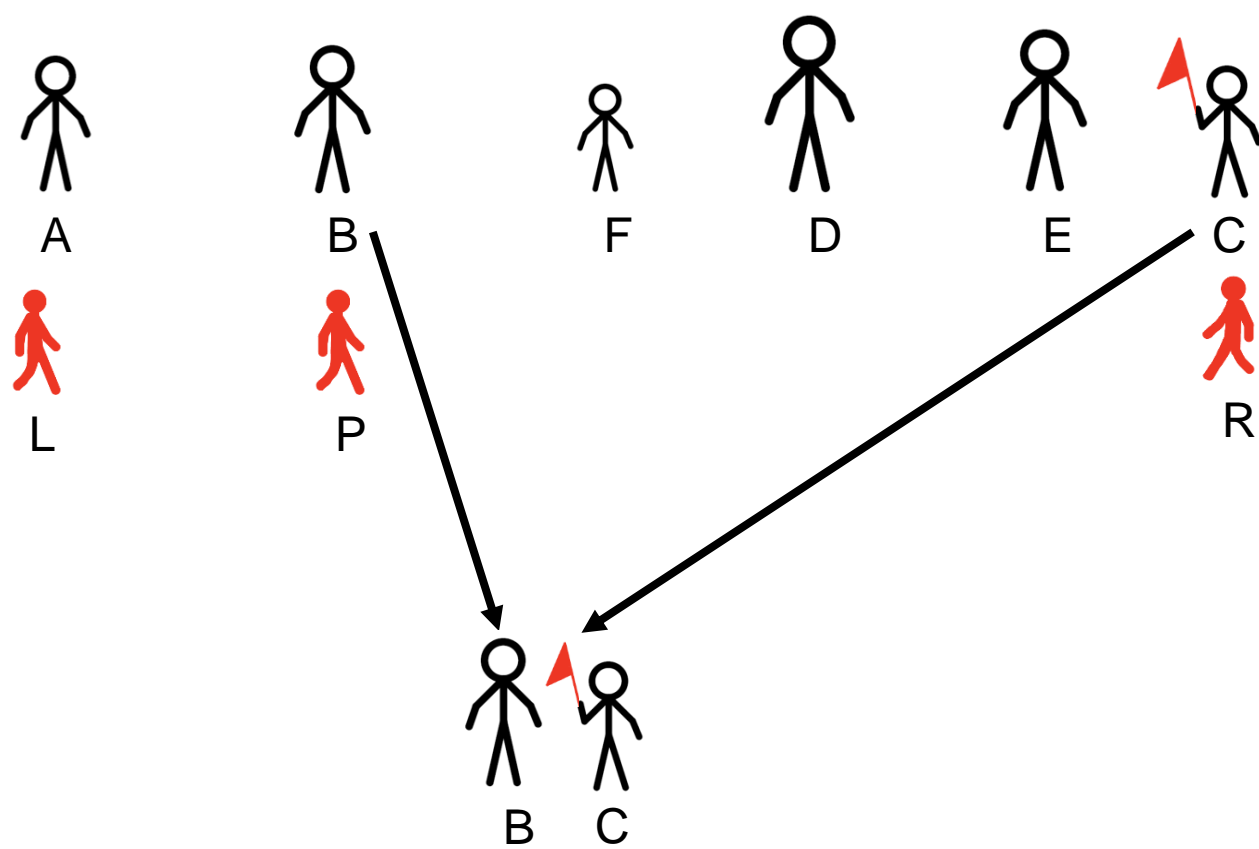
# Step 6: Jump



```
L3: swap  flag, R
L4: cmp    L9, L7
L5: swap  P, L
L6: inc    L
L7: inc    P
L8: goto   L4
L9: swap  flag, L
```

Jump to L4  
L, P, R and data group keep unchanged

# Step 7: Compare



L3: swap flag, R

L4: cmp L9, L7

L5: swap P, L

L6: inc L

L7: inc P

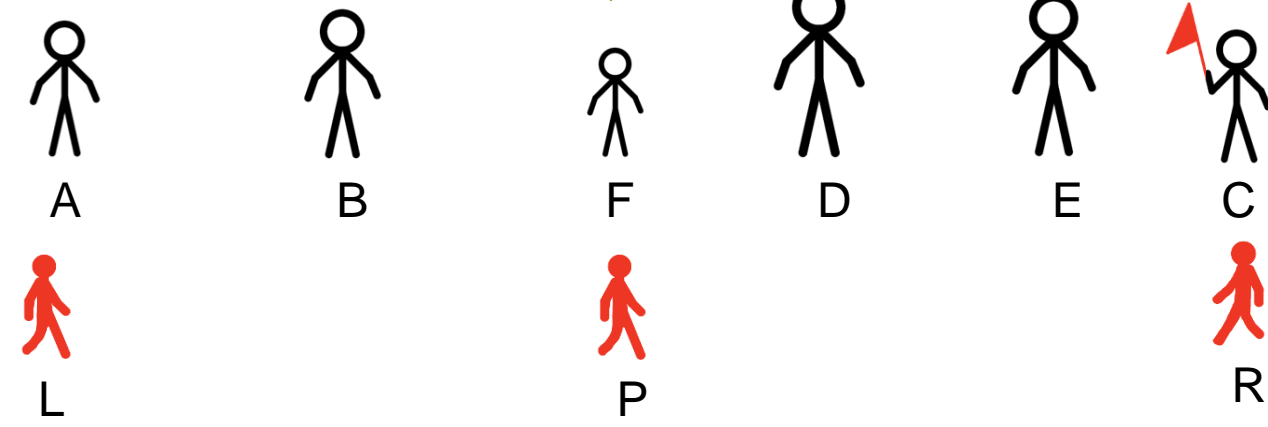
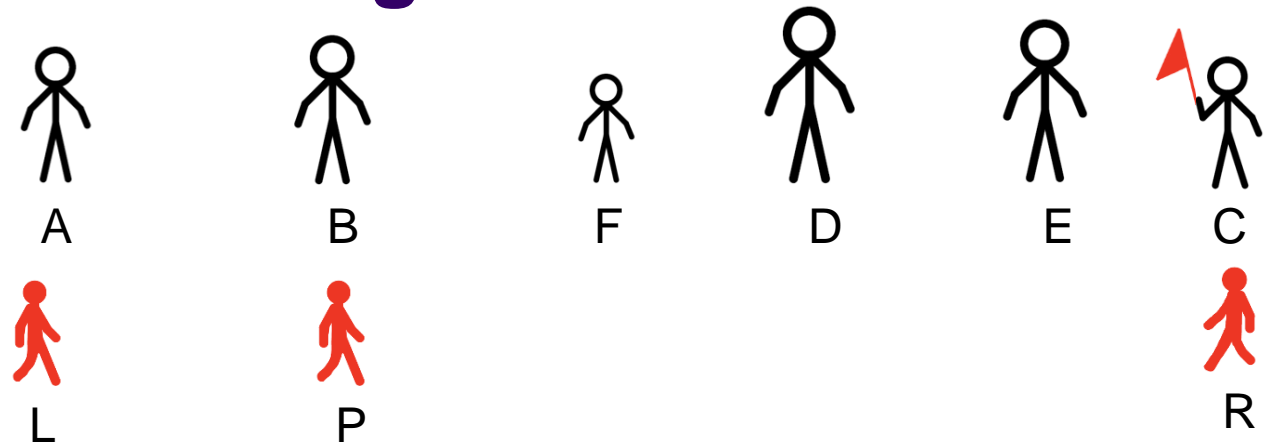
L8: goto L4

L9: swap flag, L

$P \neq R$  and B is taller than C, jump to L7

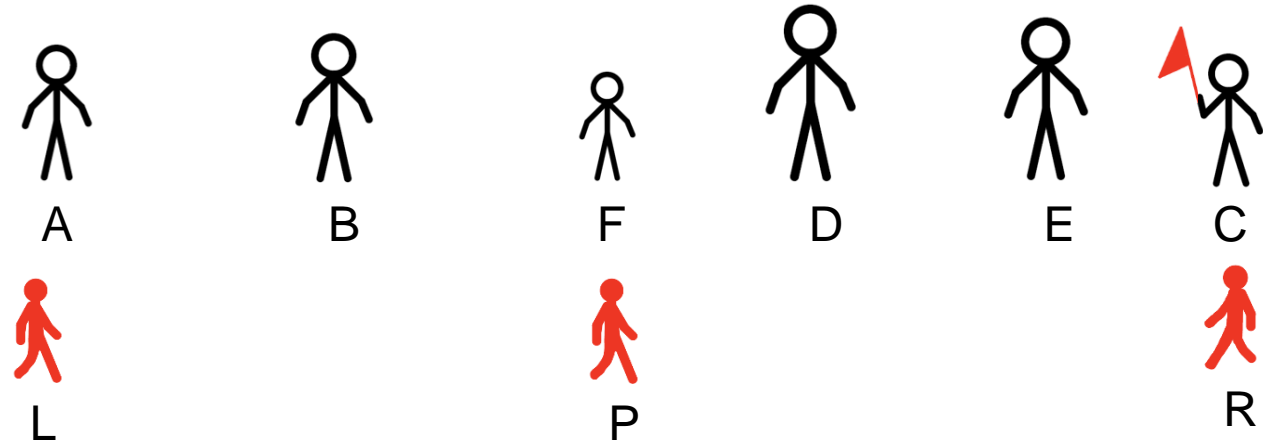
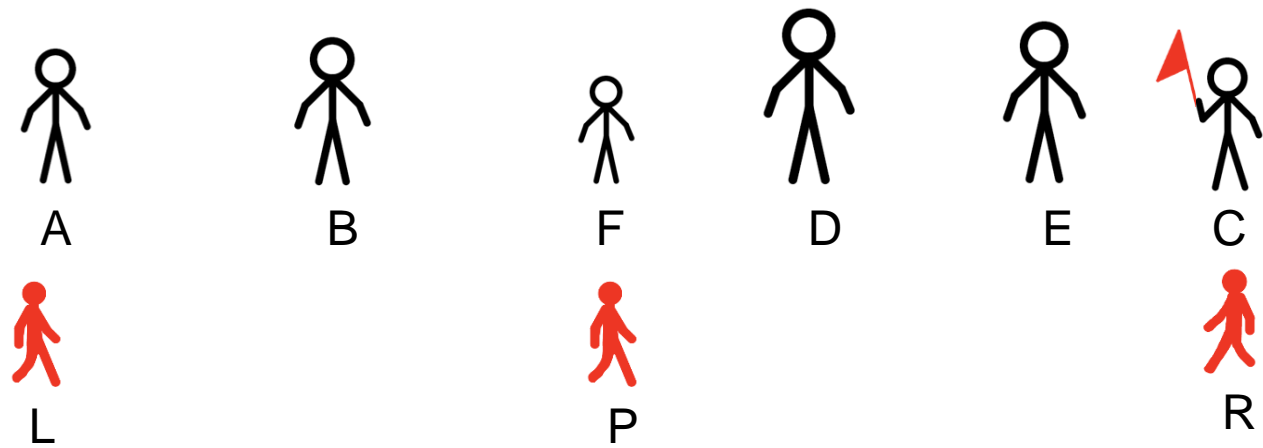
# Step 8:

## P moves 1 position to right



- L3: swap flag, R
- L4: cmp L9, L7
- L5: swap P, L
- L6: inc L
- L7: inc P**
- L8: goto L4
- L9: swap flag, L

## Step 9: Jump

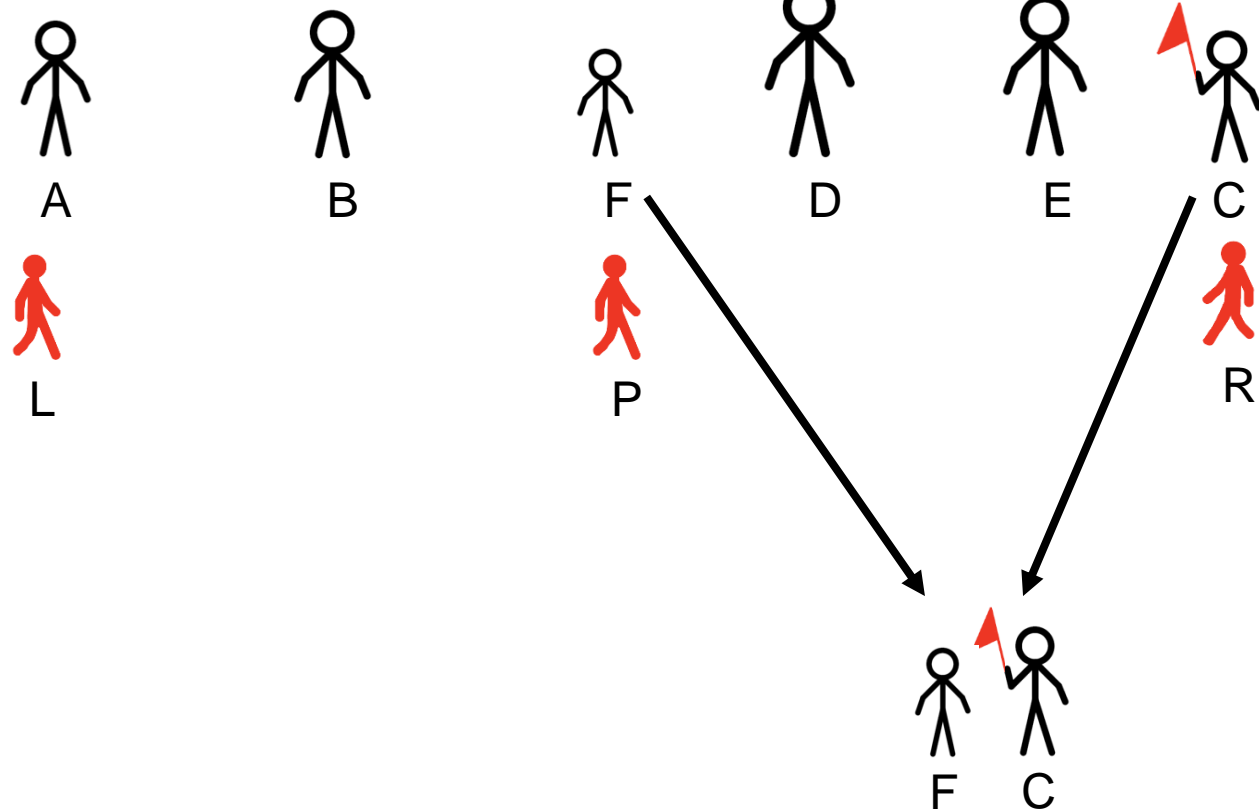


L3: swap flag, R  
L4: cmp L9, L7  
L5: swap P, L  
L6: inc L  
L7: inc P  
**L8: goto L4**  
L9: swap flag, L

Jump to L4

L, P, R and data group keep unchanged

# Step 10: Compare



L3: swap flag, R

L4: cmp L9, L7

L5: swap P, L

L6: inc L

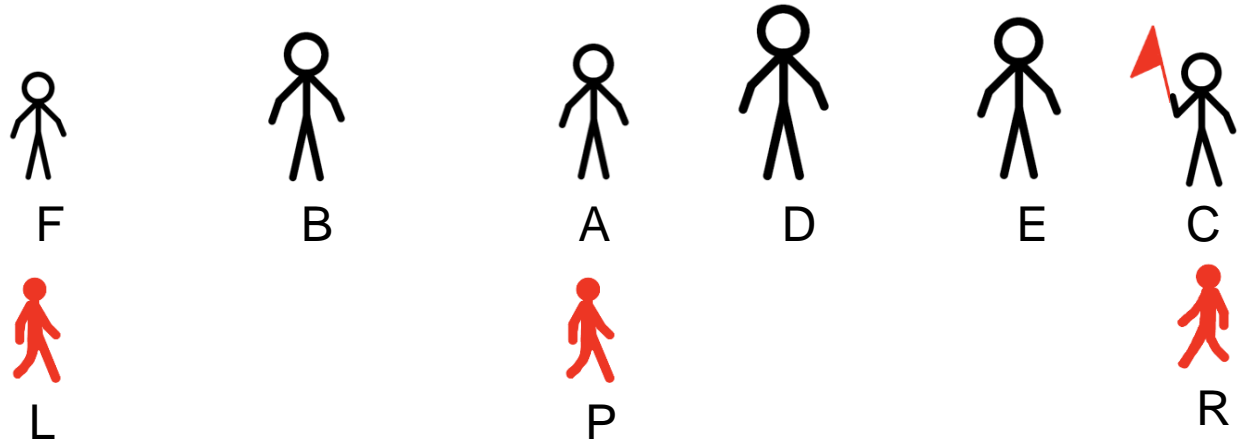
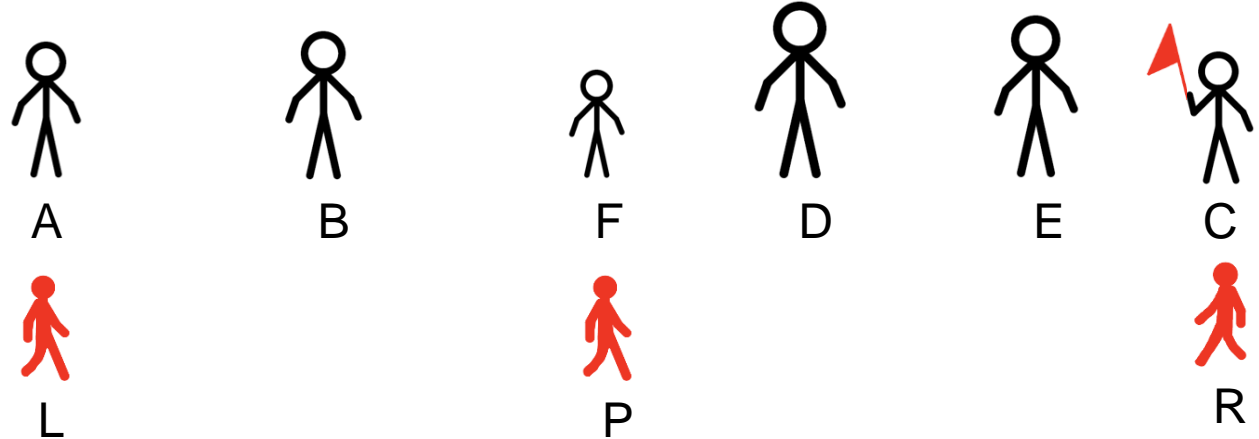
L7: inc P

L8: goto L4

L9: swap flag, L

$P \neq R$  and F is shorter than C, execute the next instruction in sequence (L5)

# Step 11: Swap



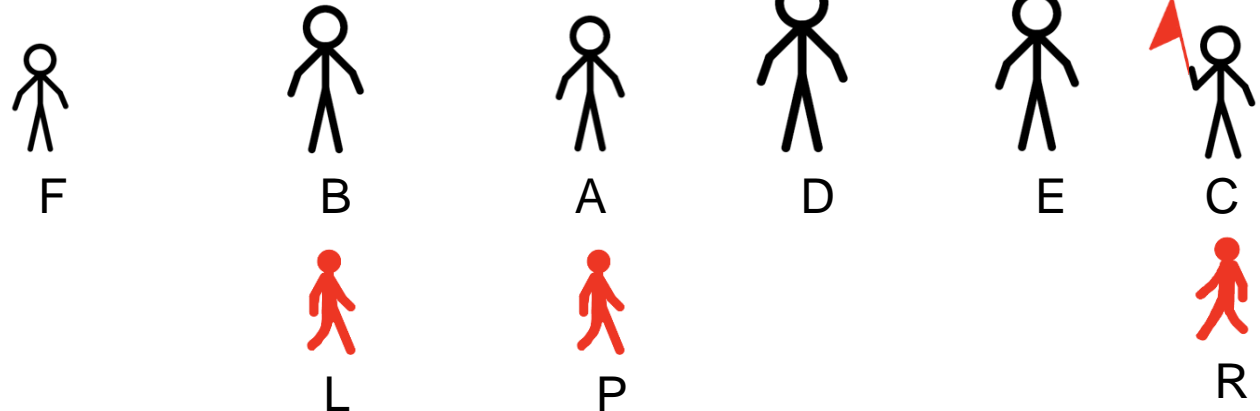
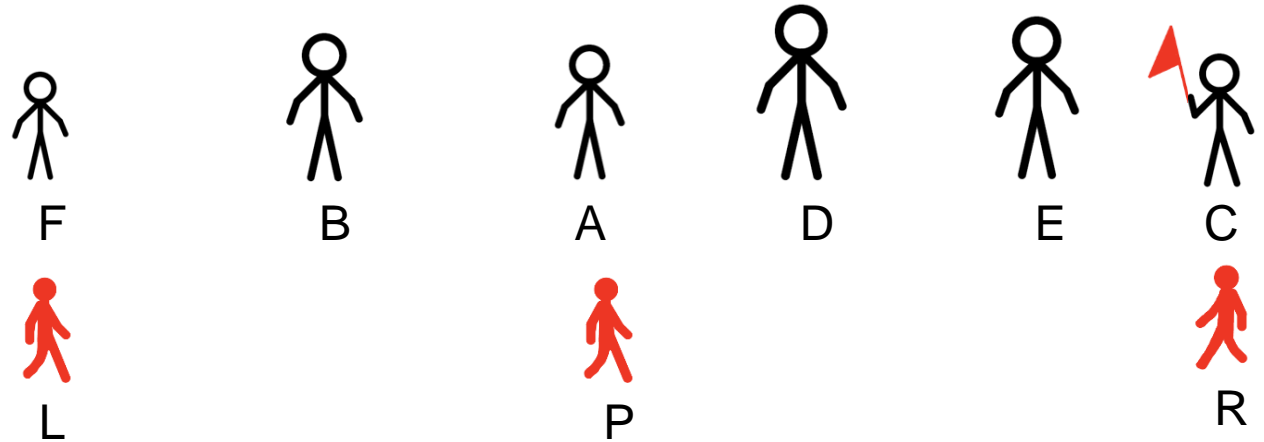
L3: swap flag, R  
L4: cmp L9, L7  
**L5: swap P, L**  
L6: inc L  
L7: inc P  
L8: goto L4  
L9: swap flag, L

Swap A and F



# Step 12:

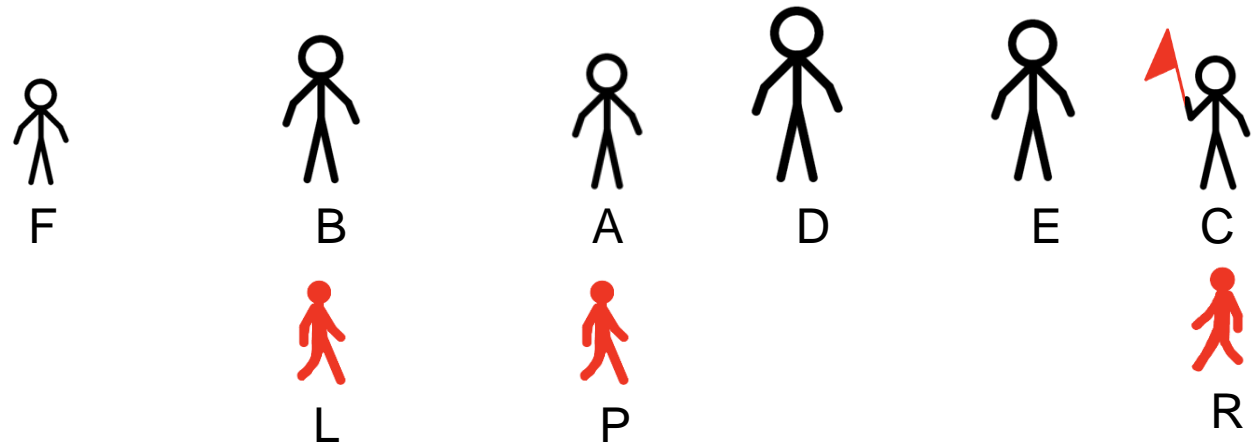
## L moves 1 position to right



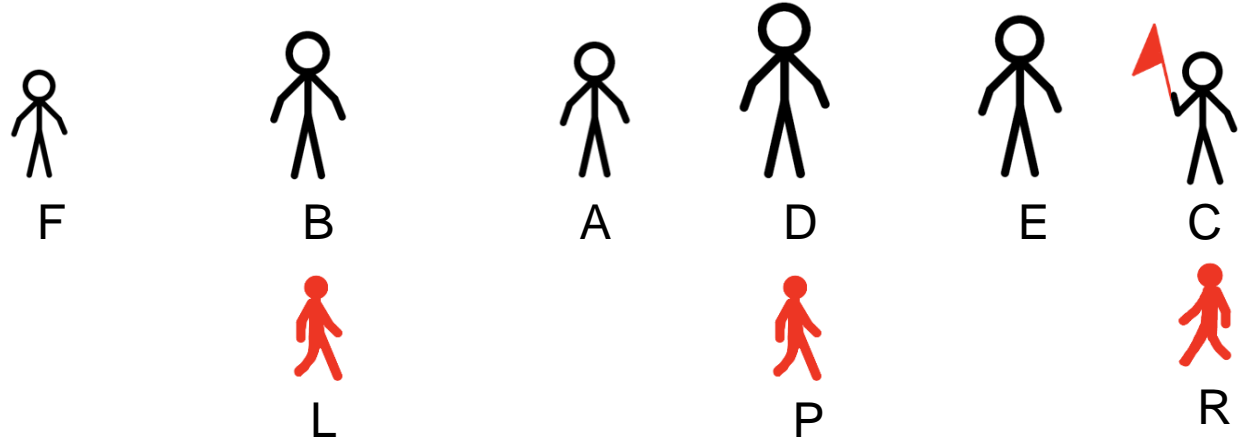
L3: swap flag, R  
L4: cmp L9, L7  
L5: swap P, L  
**L6: inc L**  
L7: inc P  
L8: goto L4  
L9: swap flag, L

# Step 13:

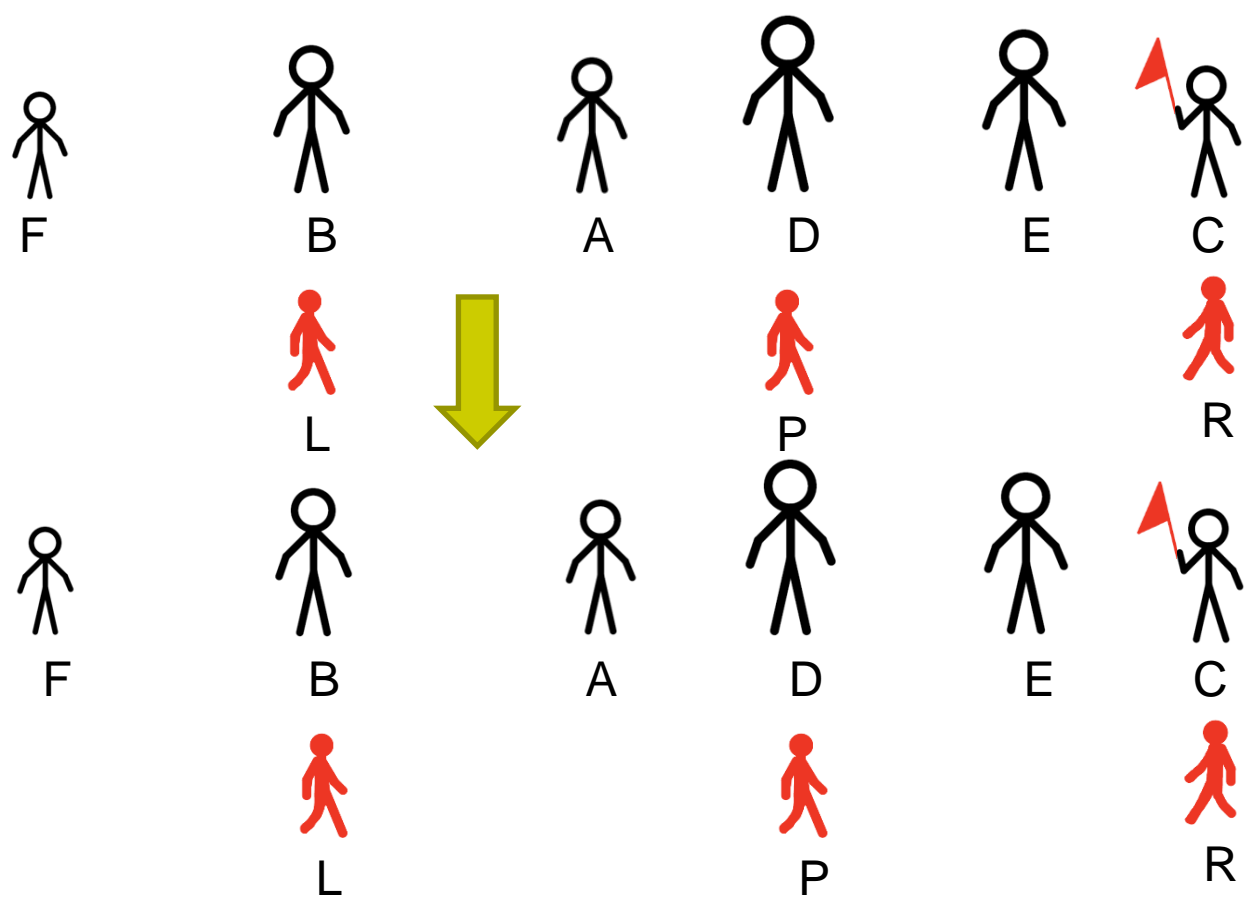
## P moves 1 position to right



L3: swap flag, R  
L4: cmp L9, L7  
L5: swap P, L  
L6: inc L  
**L7: inc P**  
L8: goto L4  
L9: swap flag, L



# Step 14: Jump



L3: swap flag, R

L4: cmp L9, L7

L5: swap P, L

L6: inc L

L7: inc P

**L8: goto L4**

L9: swap flag, L

Jump to L4.

L, P, R and data group keep unchanged.

# Step 15: Compare

F

B

L

A

D

E

C

R

D

C

L3: swap flag, R

L4: cmp L9, L7

L5: swap P, L

L6: inc L

L7: inc P

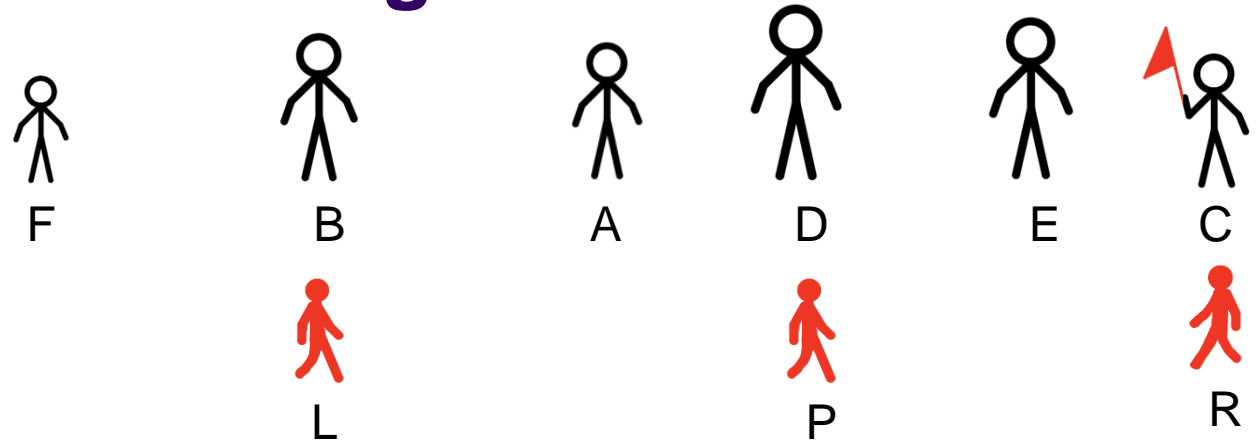
L8: goto L4

L9: swap flag, L

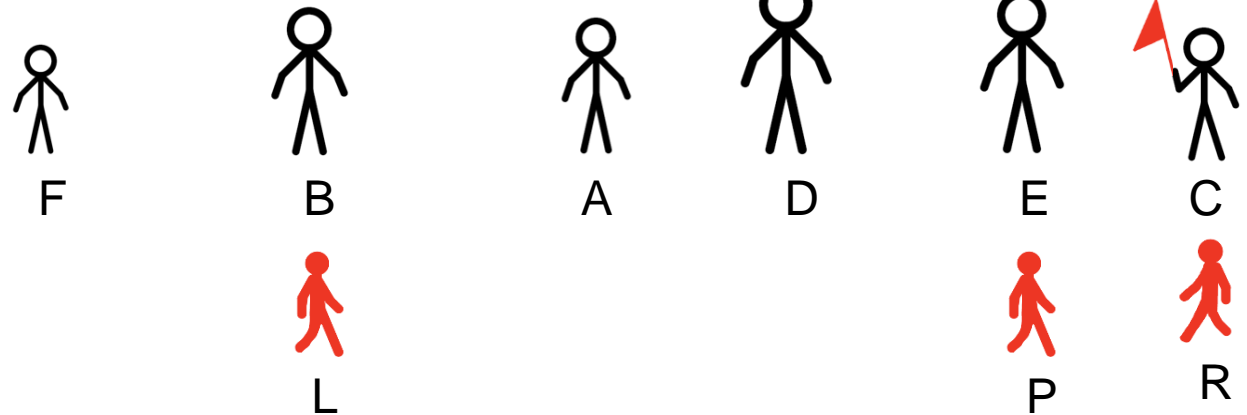
$P \neq R$ , D is taller than C, jump to L7

# Step 16:

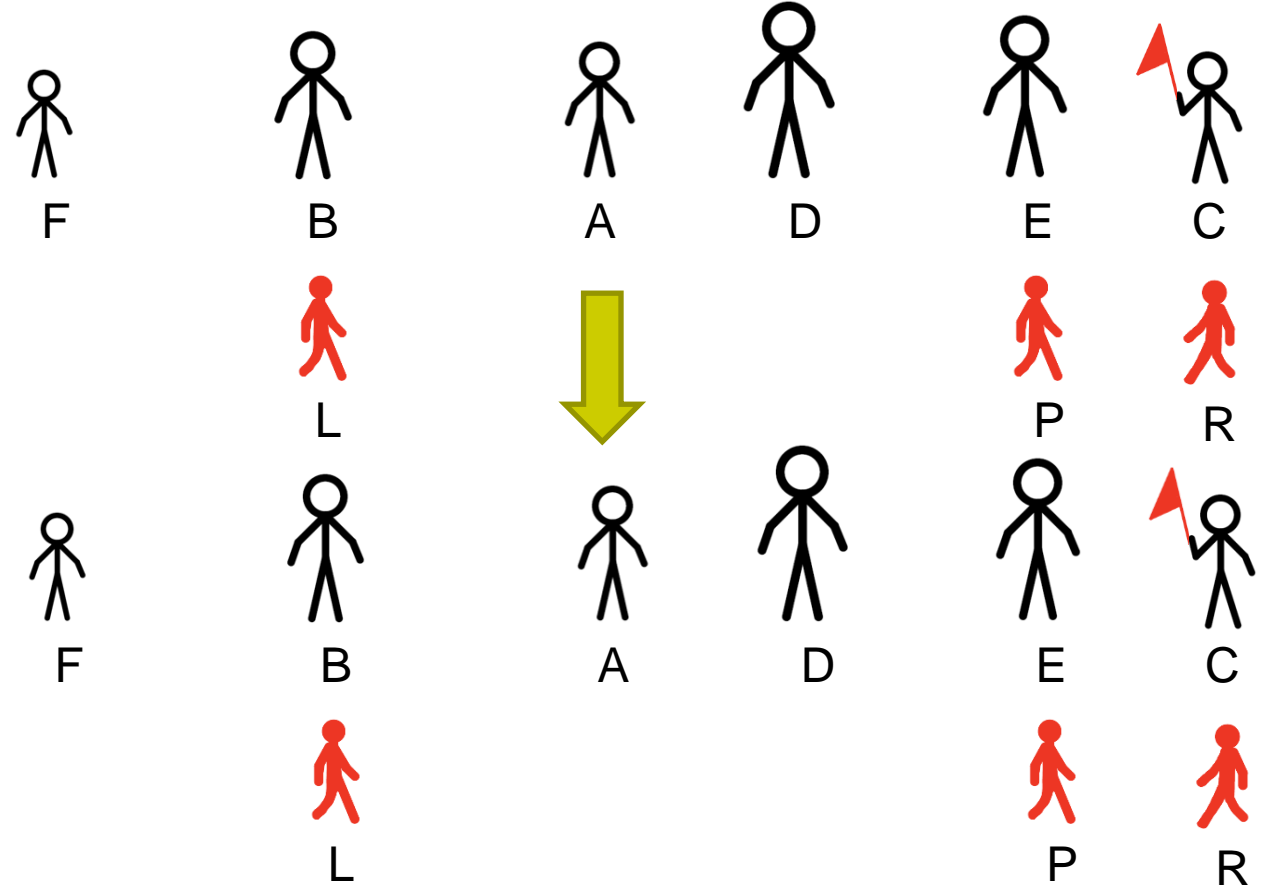
## P moves 1 position to right



L3: swap flag, R  
L4: cmp L9, L7  
L5: swap P, L  
L6: inc L  
**L7: inc P**  
L8: goto L4  
L9: swap flag, L



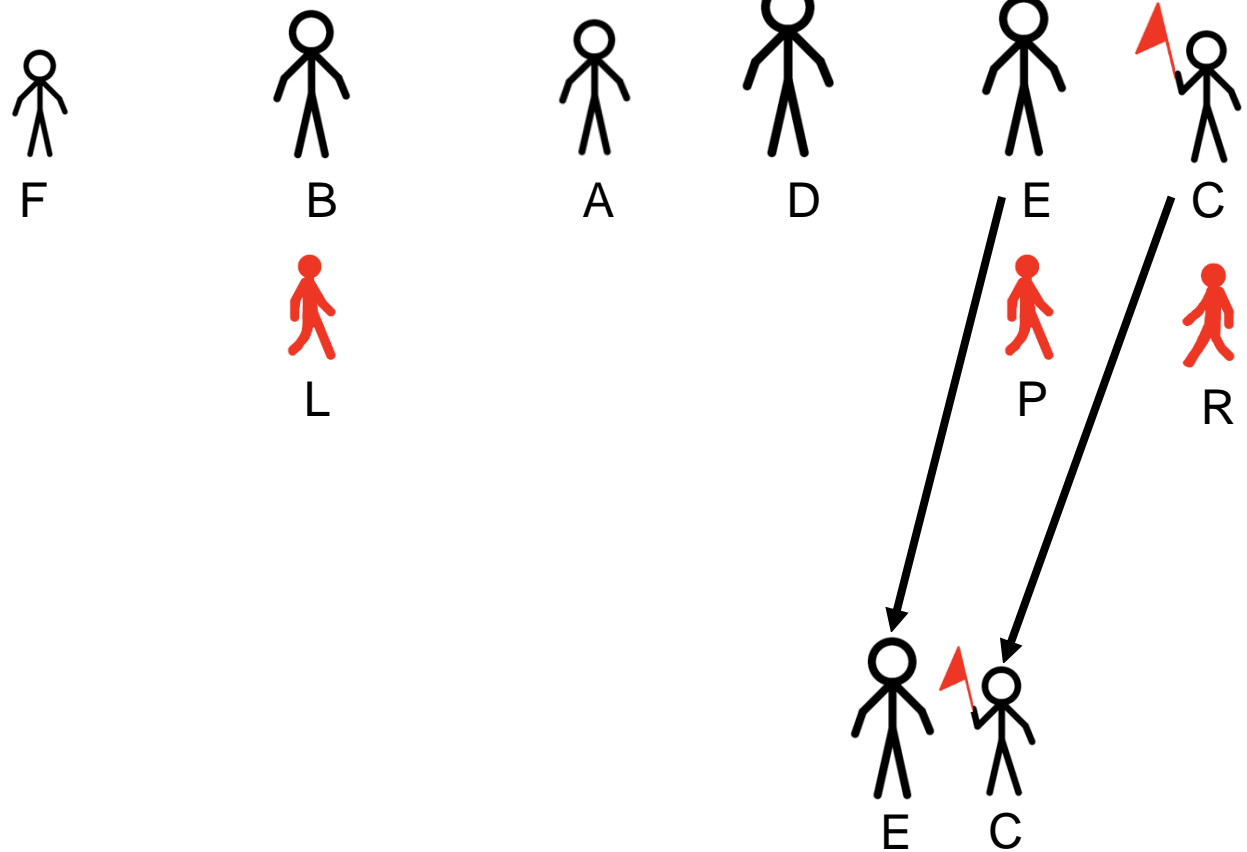
# Step 17: Jump



L3: swap flag, R  
L4: cmp L9, L7  
L5: swap P, L  
L6: inc L  
L7: inc P  
**L8: goto L4**  
L9: swap flag, L

Jump to L4.  
L, P, R and data group keep unchanged.

# Step 18: Compare



L3: swap flag, R

L4: cmp L9, L7

L5: swap P, L

L6: inc L

L7: inc P

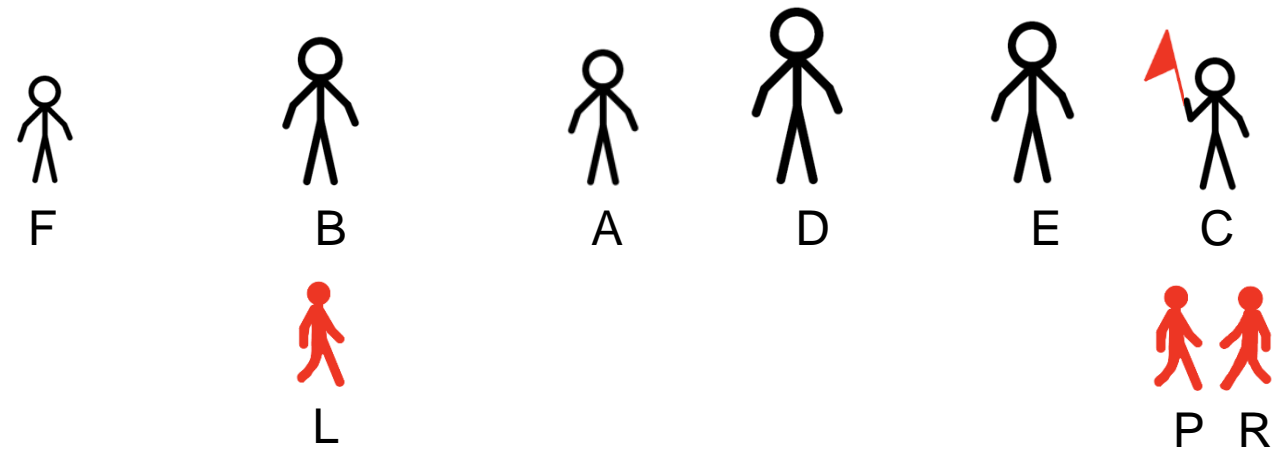
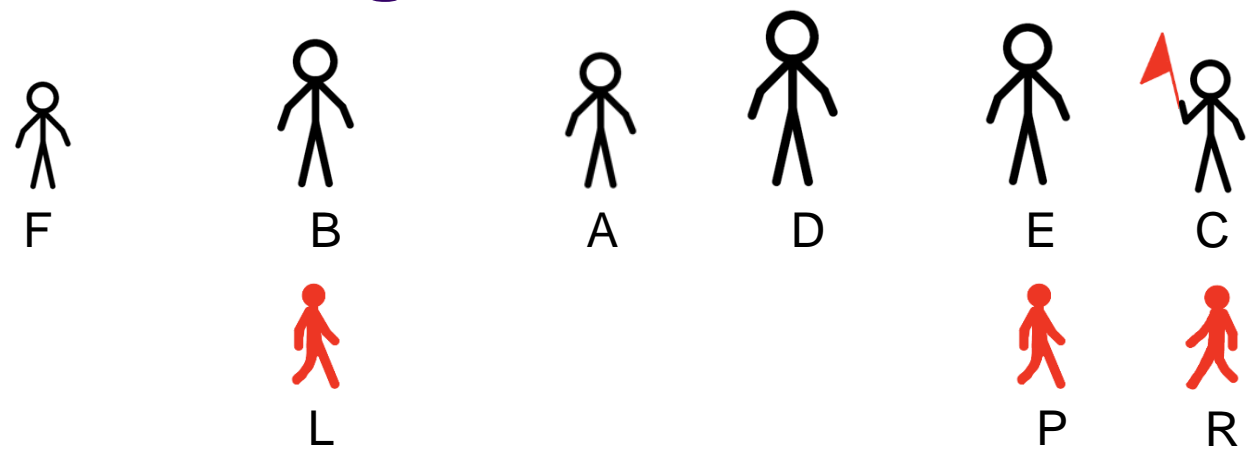
L8: goto L4

L9: swap flag, L

$P \neq R$  and E is taller than C, jump to L7

# Step 19:

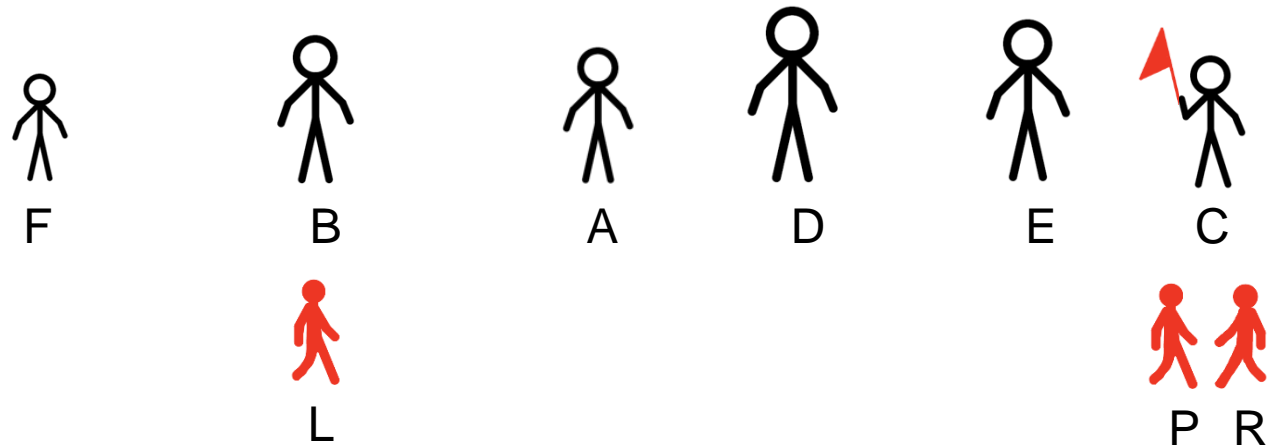
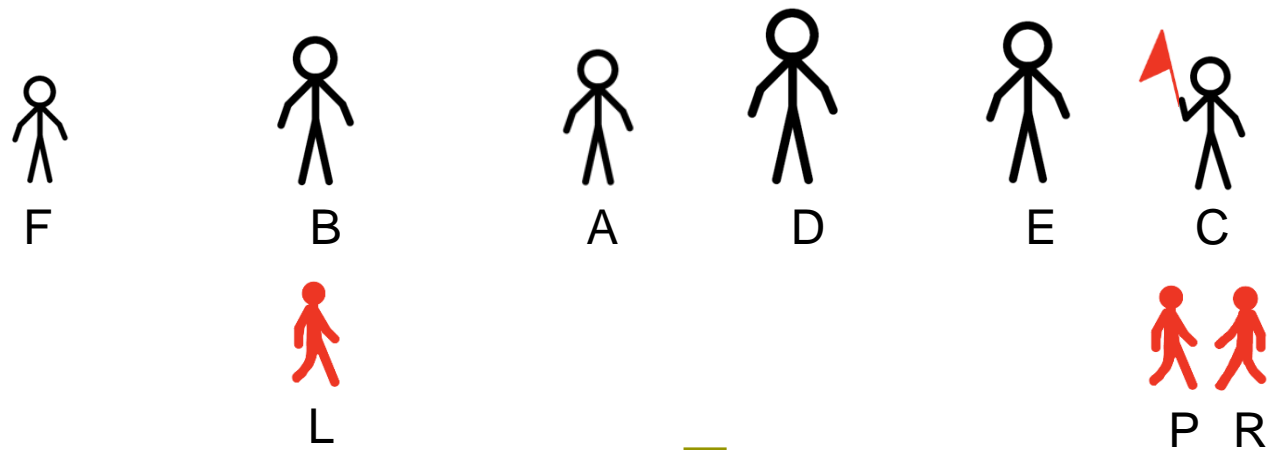
## P moves 1 position to right



- L3: swap flag, R
- L4: cmp L9, L7
- L5: swap P, L
- L6: inc L
- L7: inc P
- L8: goto L4
- L9: swap flag, L



# Step 20: Jump

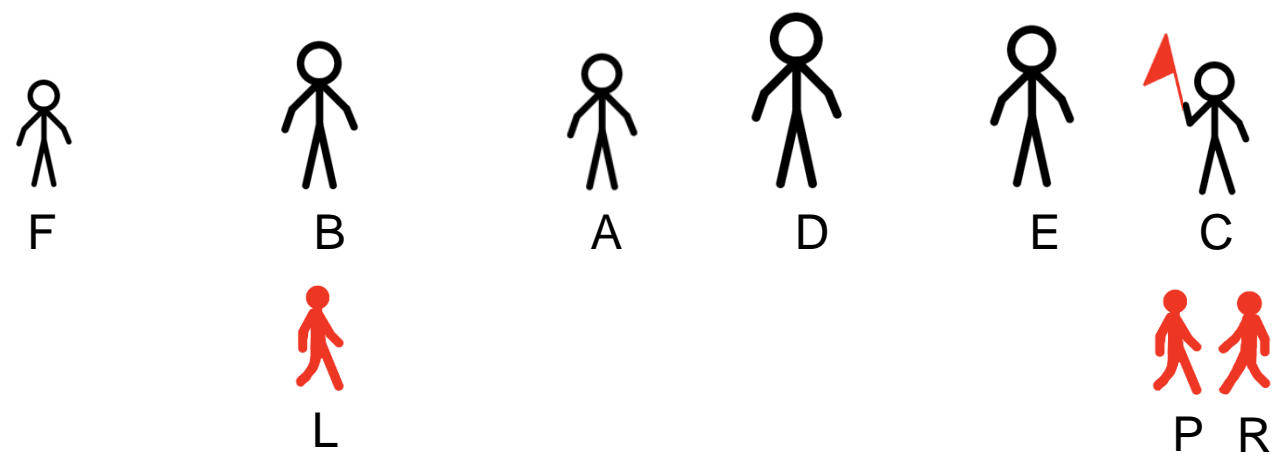
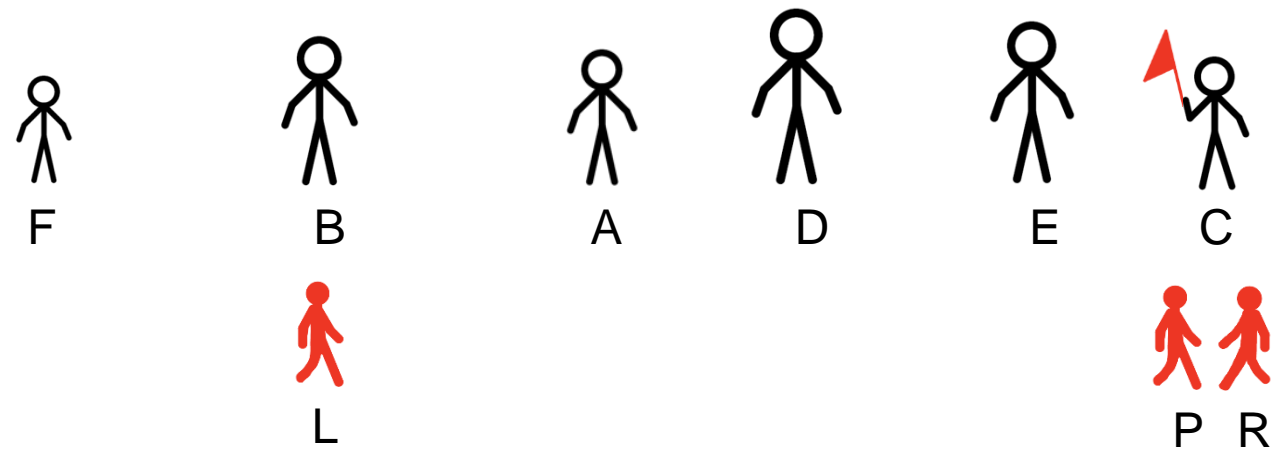


L3: swap flag, R  
L4: cmp L9, L7  
L5: swap P, L  
L6: inc L  
L7: inc P  
**L8: goto L4**  
L9: swap flag, L

Jump to L4.

L, P, R and data group keep unchanged.

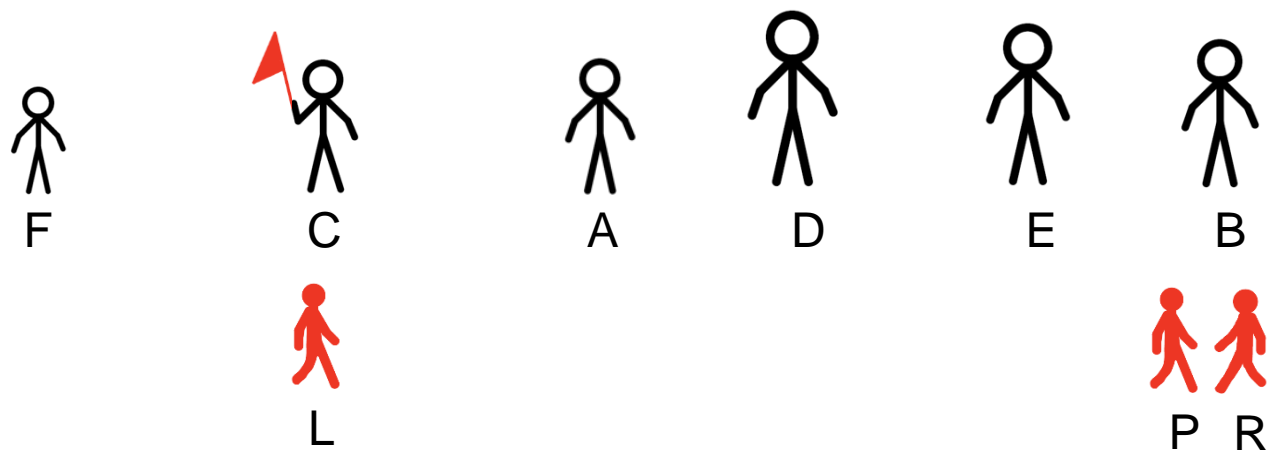
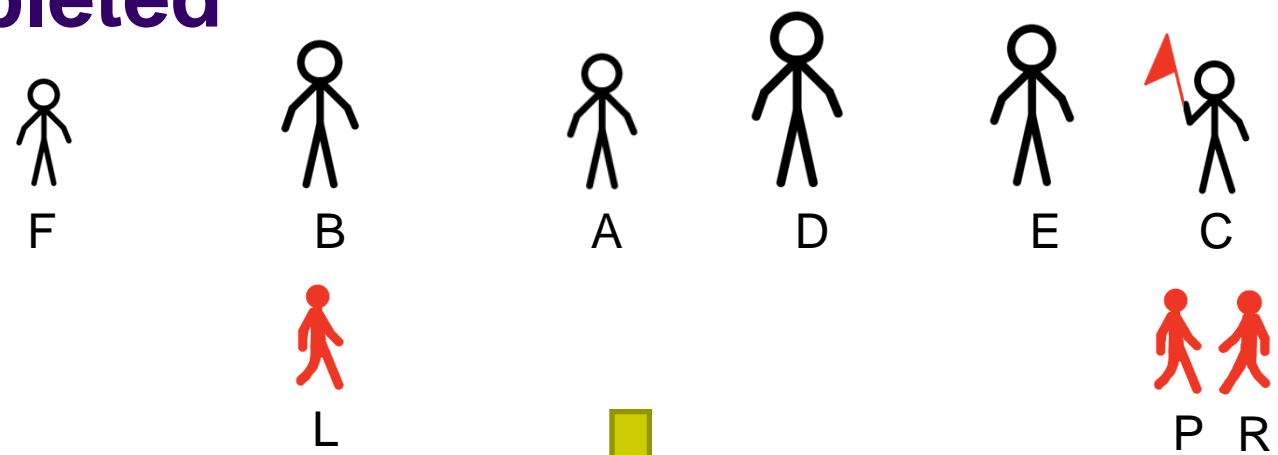
# Step 21: Compare



```
L3: swap  flag, R
L4: cmp   L9, L7
L5: swap  P, L
L6: inc   L
L7: inc   P
L8: goto  L4
L9: swap  flag, L
```

P = R, jump to L9

# Step 22: Partition completed



- L3: swap flag, R
- L4: cmp L9, L7
- L5: swap P, L
- L6: inc L
- L7: inc P
- L8: goto L4
- L9: swap flag, L

Swap the pivot to L, partition is completed.