External Sorting

Merge Sort, Replacement Selection

Overview

Structure:

1. What is “External Sorting”?
2. How does “Merge Sort” work?
   - Balanced n-way-merging
   - Improvements
3. What are the advantages of a “Selection Tree”?
4. What is “Replacement Selection”?
   - Snow-plow example
   - Improvements
5. Applicability and efficiency

1. Principle

Conventional sort algorithms:

e.g. *Quick Sort*, *Heap Sort*, *Selection Sort*,...

Very efficient but all data needs to fit completely into main memory.

External sorting:

performing sorting operations on amounts of data that are too large to fit into main memory.

External sorting can not be done in one step.

1. Principle

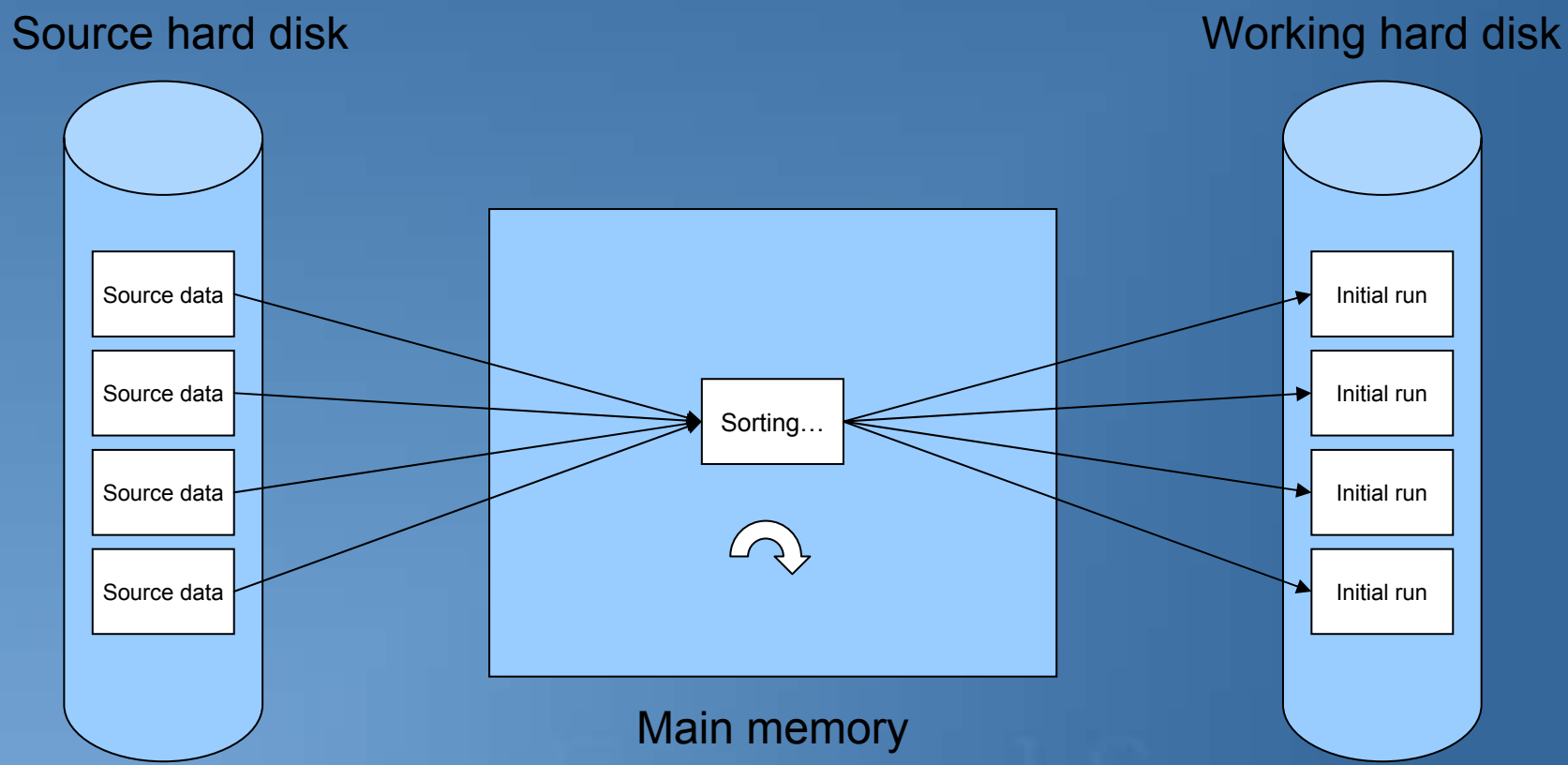
Multiple steps:

1. Split the data into pieces that fit into main memory
2. Sort the pieces with conventional sorting algorithms
3. Merge those so called *runs* and build the completely sorted data-set

Internal Sorting and **External Merging**

2. Merge Sort

Principle: Internal Sorting and External Merging



2.1 Balanced n-way-merge

Step 1:

Unsorted data-set:

535	288	351	354	412	198	451	852	291	448	898	165	217	366	756	665

In this example four elements each fit into main memory.

Creation of initial runs:

288	351	354	535

RUN1

198	412	451	852

RUN2

165	291	448	898

RUN3

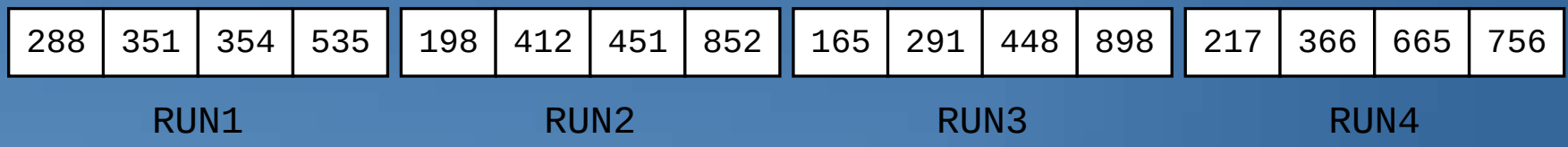
217	366	665	756

RUN4

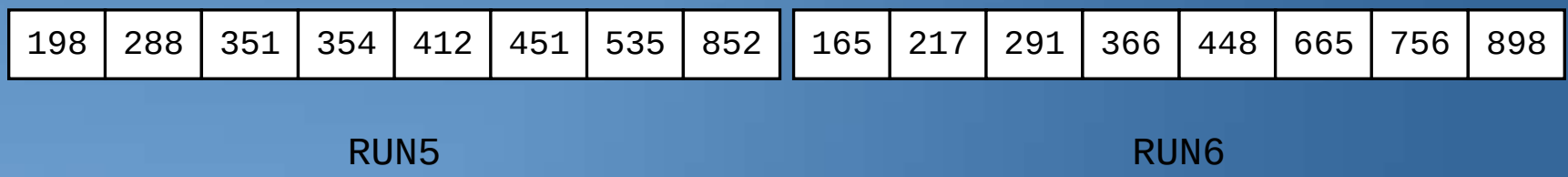
2.1 Balanced n-way-merge

Step 2:

Initial runs:



Merging of initial runs:



2.1 Balanced n-way-merge

Step 3:

Merged runs:

198	288	351	354	412	451	535	852

RUN5

165	217	291	366	448	665	756	898

RUN6

Re-merging:

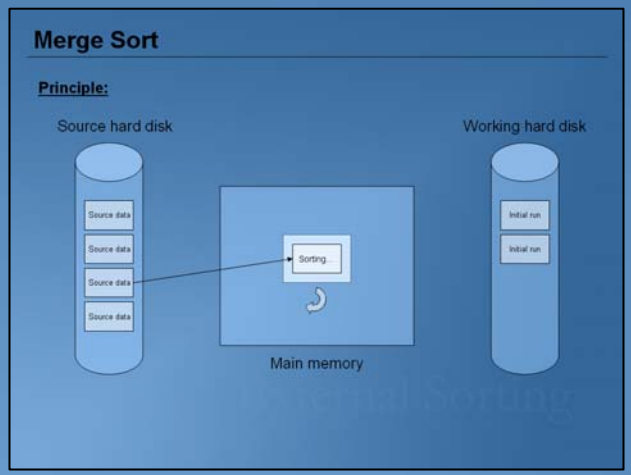
165	198	351	288	291	351	354	366	412	448	451	535	665	756	852	898

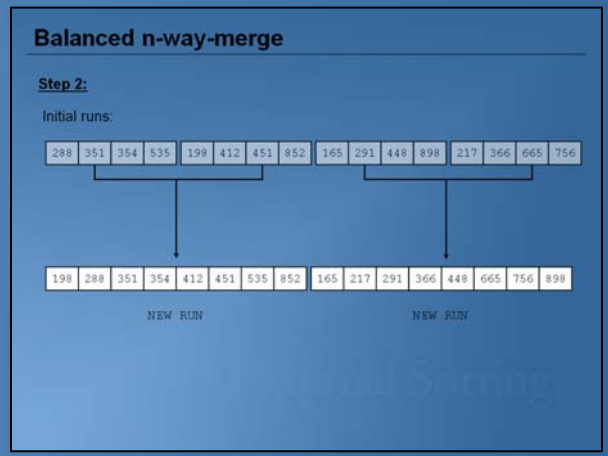
Result:
After two merge-procedures our formerly unsorted set is in **perfect order** and merge sort is complete.

2.1 Balanced n-way-merge

Explanation:

This procedure is called **Balanced 2-way-merging**





Balanced:

As well source- as working space is required

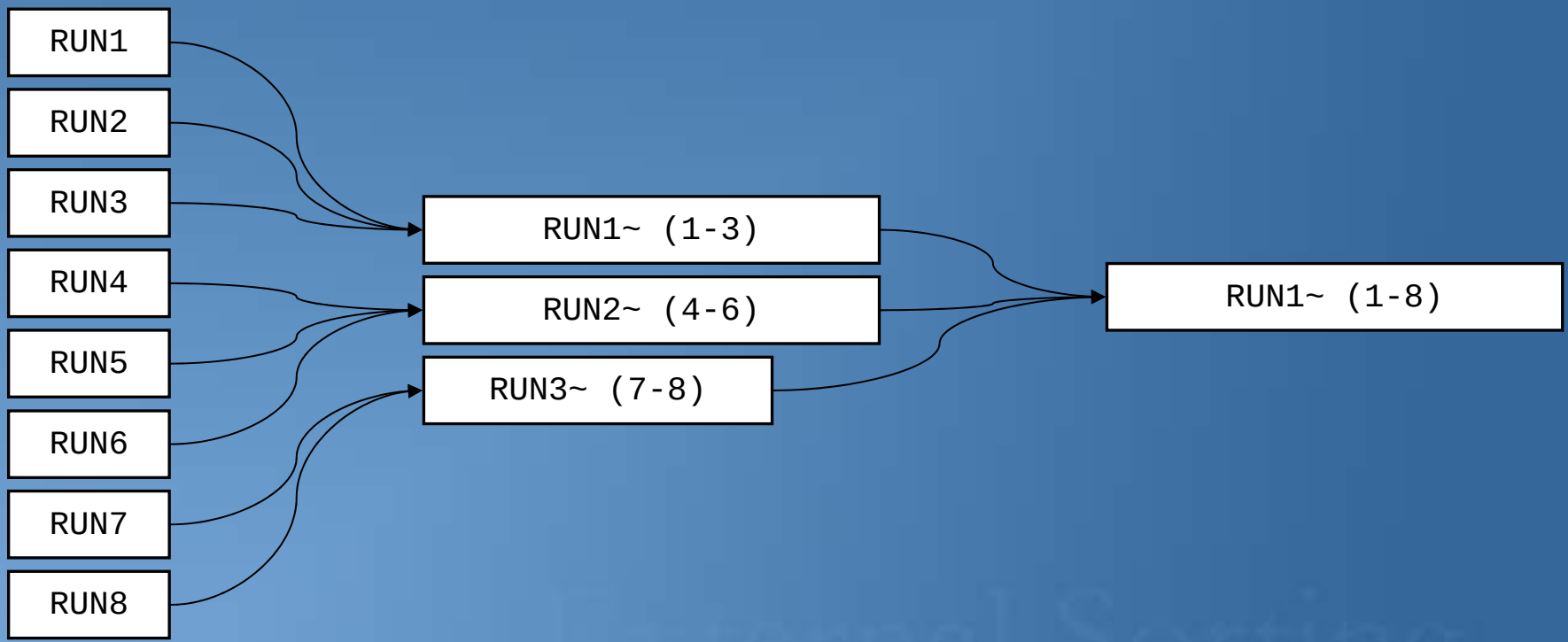
2-way:

Out of 2 merged runs one new run is formed

2.1 Balanced n-way-merge

Example:

The merging-procedure can be certainly applied to more than two runs at each time. Then, it is termed *n-way-merge* or *multiway merge*. A balanced *3-way merge* would be implemented as follows:



2.2 Sophisticated n-way-merge

Optimizations: Algorithms like *Polyphase merge*, *cascade merge*

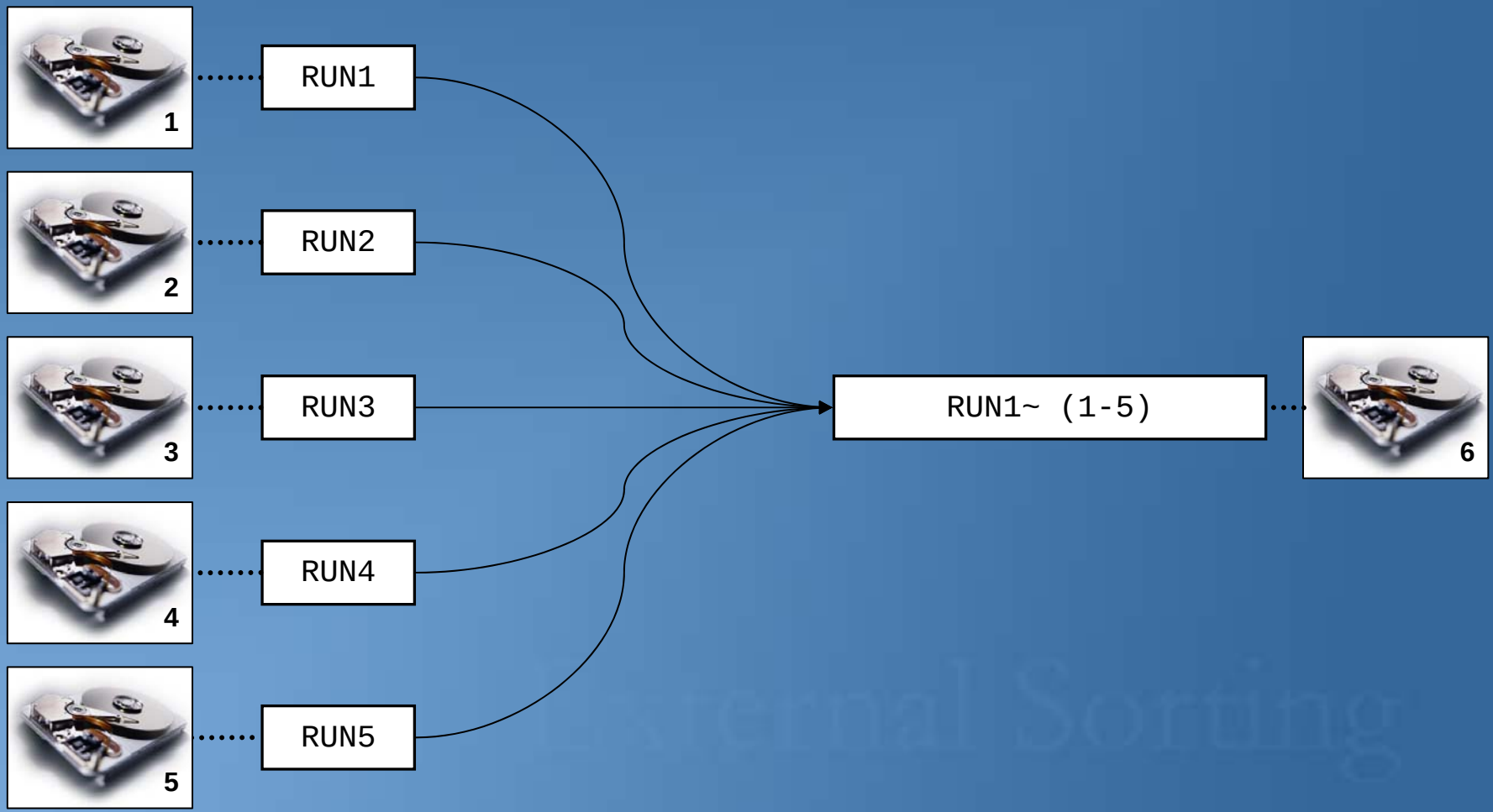
- Reducing the number of intermediate steps by implementing *n*-way-merging with great values of *n*.
- Maximizing speed by increasing the number of drives for storage disposals for minimal access time.
- Saving time by doing a perfect spreading of the runs on the storage media.

Disadvantages: Additional costs and expenditure

2.2 Sophisticated n-way-merge

Example:

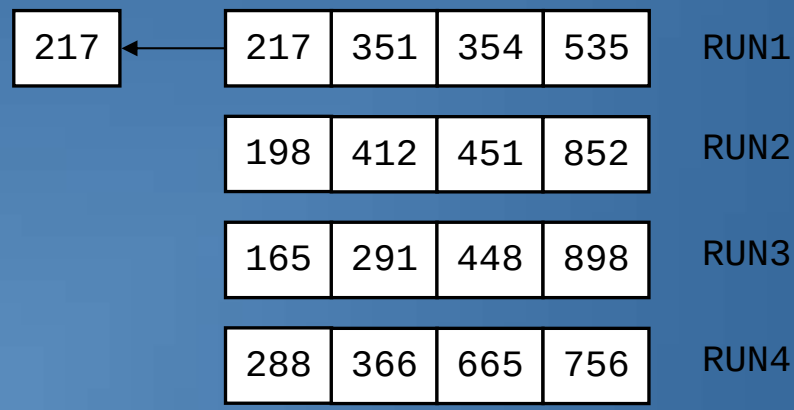
Significant speed increase by storing all runs on different drives for minimal access time:



3. Selection Tree

Problem: Selecting the smallest element is very time-consuming.

It requires $(n / p) - 1$ comparisons when using a non-advanced algorithm.



first element is compared subsequently with all remaining *p-1* elements

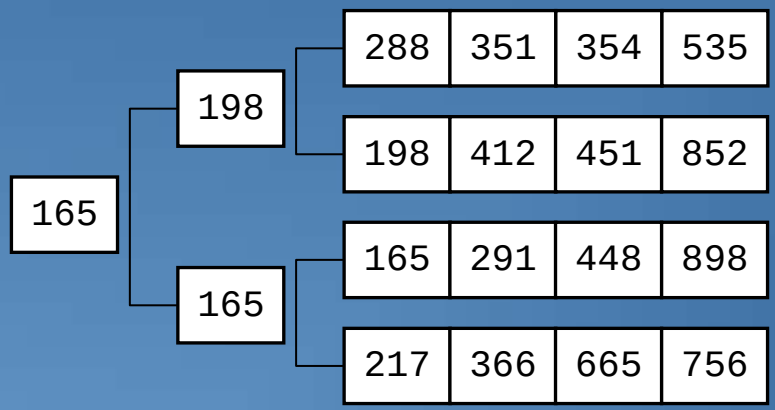
Solution:

Building a *selection tree* saves lots of comparisons and speeds up the selection process:

Then, just $log_2\ p$ comparisons are necessary.

3. Selection Tree

Start: Building a selection tree:



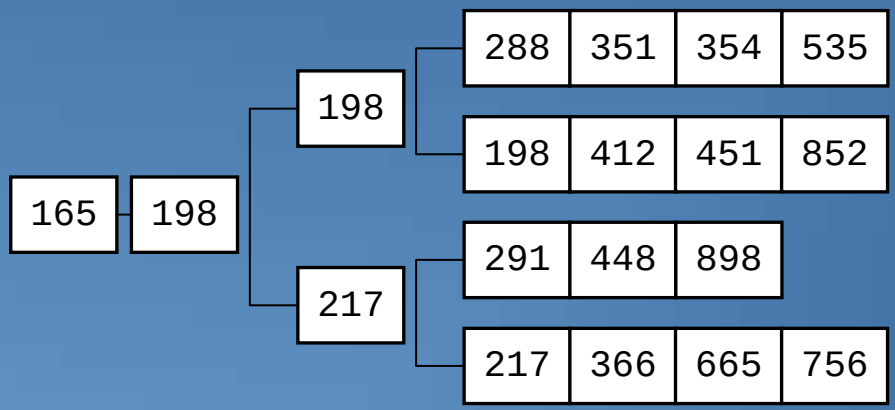
Always the smallest element is taken out of the top of the tree

New elements are pulled forward in the current branch

Repeats until all branches of the selection tree are empty

3. Selection Tree

Step 1: Pulling smallest elements forward



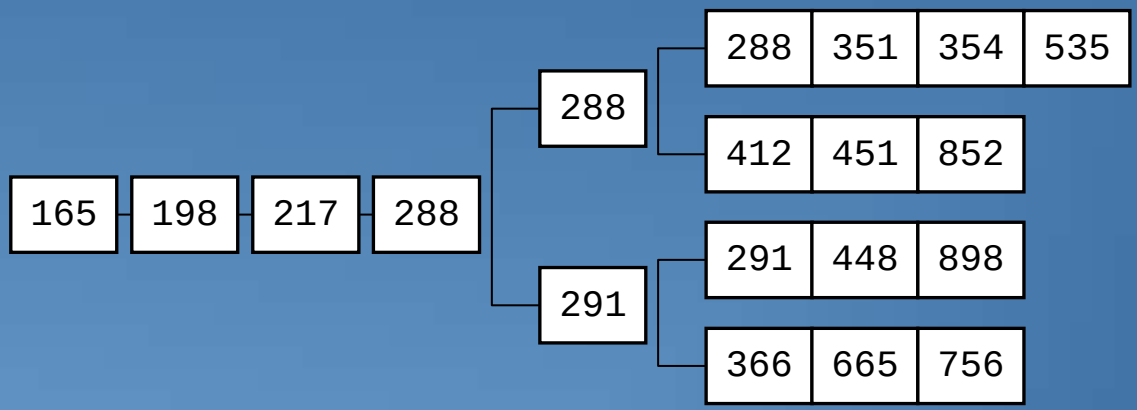
Always the smallest element is taken out of the top of the tree

New elements are pulled forward in the current branch

Repeats until all branches of the selection tree are empty

3. Selection Tree

Step 3: Pulling smallest elements forward



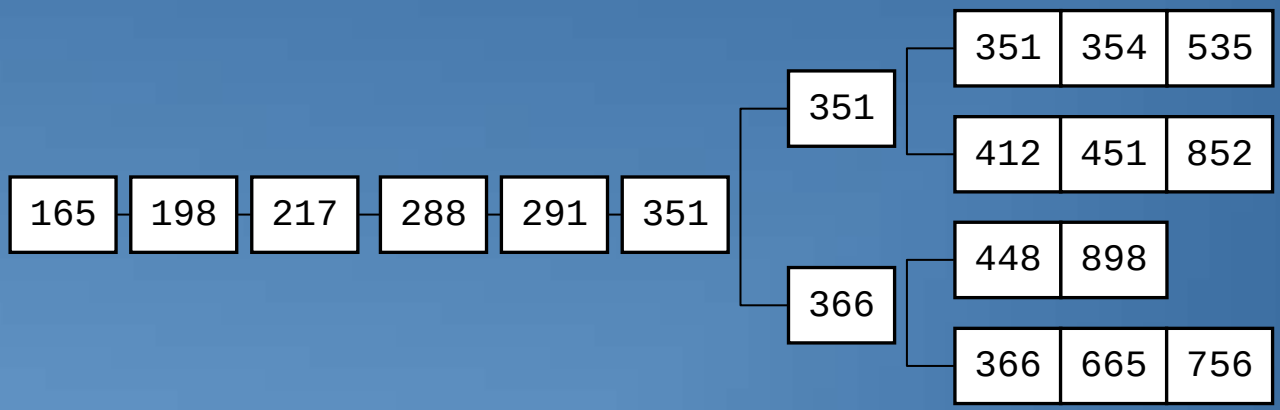
Always the smallest element is taken out of the top of the tree

New elements are pulled forward in the current branch

Repeats until all branches of the selection tree are empty

3. Selection Tree

Step 5: Pulling smallest elements forward



Always the smallest element is taken out of the top of the tree

New elements are pulled forward in the current branch

Repeats until all branches of the selection tree are empty

4.1 Replacement selection

Most efficient is to keep the number of initial runs very low

→ The length of runs has to be as great as possible

Conventional run-creation:

Maximum size of a run is limited by available size of main memory

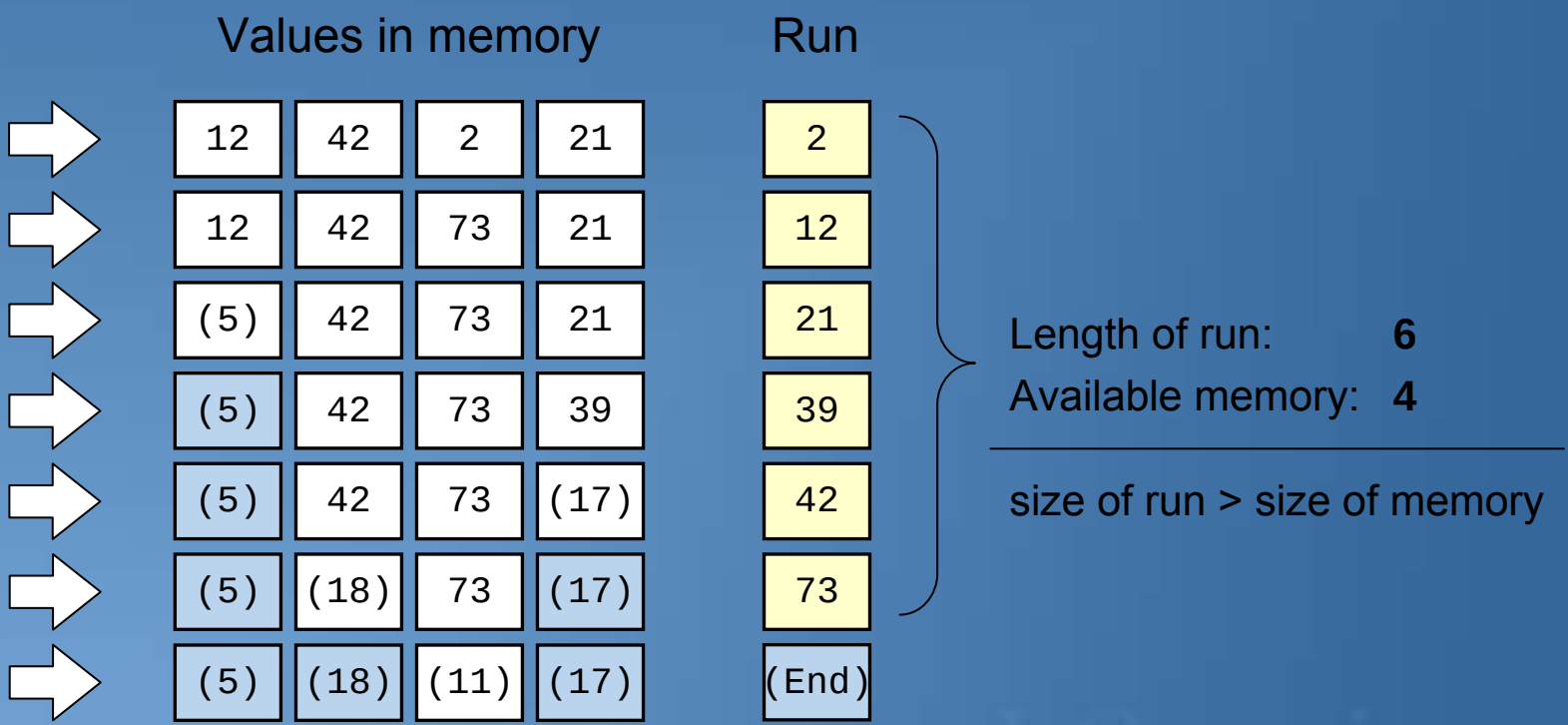
Modification:

Records are replaced in memory to form even longer runs than memory is available. This technique is called replacement selection.

4.1 Replacement selection

Example of a replacement selection sequence:

Four elements each fit into main memory



4.1 Replacement selection

What happened:

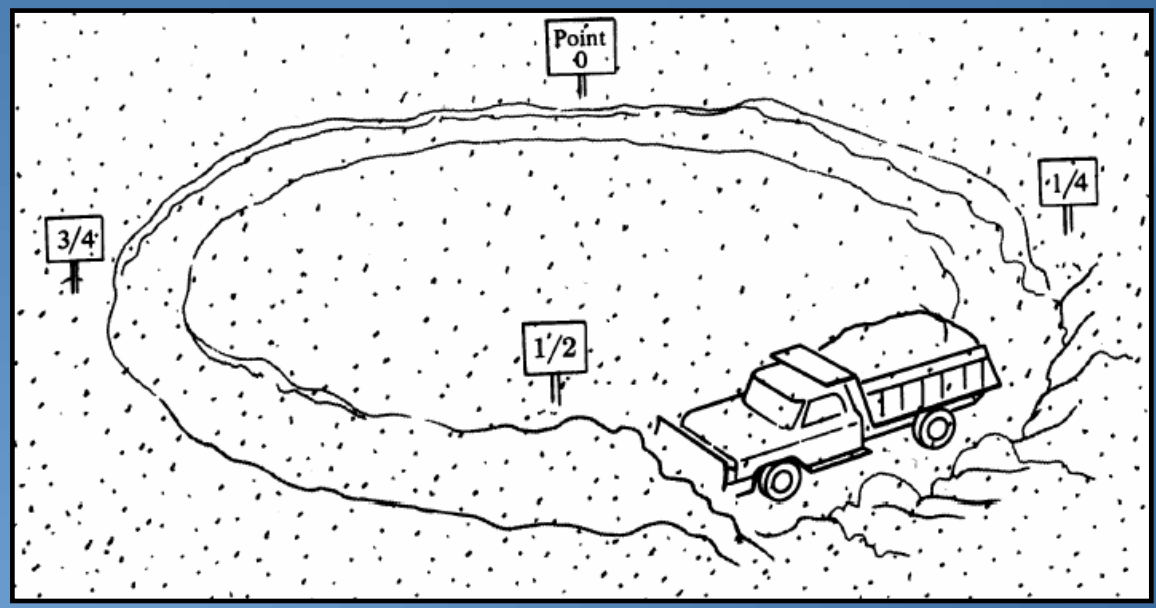
1. The smallest record in memory is stored to the run
2. Right after that, a new record is loaded at its position in memory
3. If this new record is smaller than our last element of the current run, it is tagged, because we can't use it now
4. Records are replaced in memory to form even longer runs than memory is available

Result:

- Long length of runs, especially when data is presorted
- Statistically, length of runs levels off at *2 * size of memory*
- Practically, runs tend to contain even more records, because in almost every commercial application data is presorted

4.2 Replacement selection

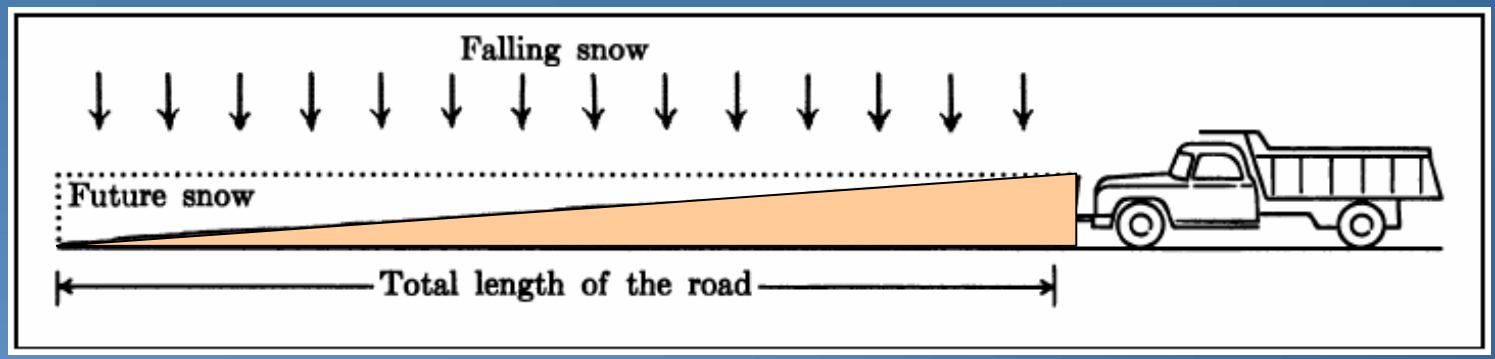
Demonstration: There's a well-known way to proof why initial runs of a length of *2 * q* can be expected when *q* is the size of main memory.



A snowplow is clearing a road with snow randomly distributed all over.

4.2 Replacement selection

Because snow is falling at constant speed, this stable situation will never change:



- Rectangle is cut in half by the line representing the actual snow level
- Level of existing snow represents records in main memory
- At the end of the road, there is no snow from the previous turn left
- All records from the last run are tagged with the marker, so a new run has to be created.
- The volume of snow removed in one circle (namely the length of a run) is twice the amount that is present on the track at any time.

5. Applicability and efficiency

Most popular algorithms:

1. *Internal sorting:*

   creates short runs with a constant maximum length equal to the size of main memory.

2. *Replacement selection:*

   mostly used, creates runs of big size.

As well as

3. *Delayed Reconstitution of the Runs*

4. *Replacement Selection with natural selection*

Today, speed and efficiency of external sorting is less concerned with the algorithm than with the thereby used hardware.

6. Conclusion

Speed:

Can’t compete by far with speed of internal sort algorithms

Intention:

Minimize accesses to slow external media

Provide suitable and affordable solution

Advantage:

In practice, data records are often presorted in some way.

In this case, replacement selection can produce extremely long runs

Development:

Increase of speed because of more sophisticated algorithms

Increase of speed because of much faster external hardware