Chapter 8 Sorting Large Files

What happens if we have a file that is 10 times larger?

- The sort phase uses sequential I/O so this phase uses minimum time
- So we need to work on the *Merge Phase* to speed up our algorithm
- In fact we work on the *read step* of the merge phase
  - cut down on the number of random access reads during the merge phase

80,000,000 records

8,000 megabytes

800 way merge

800 runs * 800 seeks/run = 640,000 seeks

	Number of seeks	Amount transferred (megabytes)	Seek + Rotation time	Transfer Time (seconds)	Total Time (seconds)
Merge Reading	640,000	8,000	7,040	600	7,640
Merge Writing	40,000	8,000	440	600	1,040
Totals	680,000	16,000	7,480	1,200	8,680

In general for a K-way merge of K runs the buffer size is

$$\left(\frac{1}{k}\right) \times \text{size of memory} = \left(\frac{1}{k}\right) \times \text{size of each run}$$

There are *K* runs (i.e. *K* times through the entire file)

Each of the *K* runs requires *K* accesses.

So, sort merge is $O(K^2)$ seeks and since $K = rN$ for some r the merge is $O(N^2)$

So What Do We Do?

- Hardware Based Solutions
- Software Based Solutions
  - work on merge
  - work on length of runs
  - overlap I/O operations

HARDWARE BASED SOLUTIONS

- Increase Memory – fewer runs and fewer seeks/run
  - 80,000,000 record file with 10 megabytes of memory requires 2 hrs 6 min.
  - 40 megabytes 400,000 records/run means 200 runs
  - 4000 seeks resulting in 16 minutes 40 seconds for the sort.

- Increase the Number of Disk Drives
  - This reduces the movement of the disk between runs
  - 800 drives means 800 seeks as opposed to 640,000 seeks
  - 7040 seconds becomes 1 second

- Increase the number of I/O channels
  - give each drive its own I/O channel
  - Complete overlap of I/O

So, can we decrease the number of seeks???

Remember: when working with data in files we need to minimize the number of seeks not the number of comparisons.

If all we cared about was comparisons the K-way merge just looked is best because

1. Each record is read only once
2. If a selection tree is used for the comparisons performed in the merging then the number of comparisons required for the K-way merge of N records is a function of $N \times \log_2 K$
3. Since K is directly proportional to N this is $O(N * \log_2 N)$ comparisons

This is reasonable for a sort.

But it is the seeks that take time so we must give up time in the merge for seeks.

Use a Multiple Step Merge.

Don't try to merge all the runs at once.

More buffer space is available for each run.

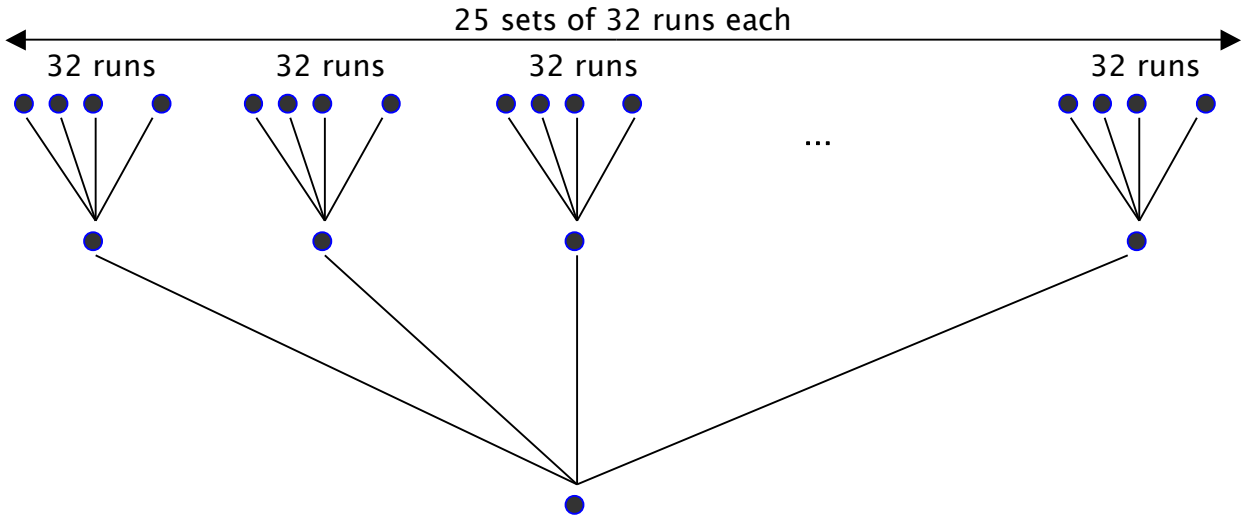
So, fewer seeks are required per run.

Use multiple passes to merge all the runs.

How does this save time? Remember the 80 million record sort

800 runs of 100,000 records each

Merge 25 sets of 32 runs followed by a 25-way merge of the intermediate runs



But can you save any time?

	Number of Seeks	Amount transferred (megabytes)	Seek + rotation time (seconds)	Transfer time (seconds)	Total time (seconds)
1st Merge Reading	25,600	8,000	282	600	882
1st Merge Writing	40,000	8,000	440	600	1040
2nd Merge Reading	20,000	8,000	220	600	820
2nd Merge Writing	40,000	8,000	440	600	1040
Totals	125,600	32,000	1,382	2,400	3,782

This is not 27 minutes. The one step merge was 2 hours 6 minutes.

1st Merge Step 25 sets of 32 runs

- each buffer can hold 1/32 of a run so we make 32 * 32 = 1024 seeks
- so for all 25, 32-way merges we make 25 * 1024 = 25,600 seeks
- each of the 25, 32-way mergers results in a run of 32 * 100,000 = 3,200,000 records

2nd Merger Step 25 runs

- each buffer can use 1/25 of the available space or 1/800 of a run
- so we have 25 * 800 = 20,000 seeks

Each Merge Step requires that all records be written back to the file which is still 40,000 seeks per merge.

The decrease in time is attributed to the decrease in seeks.

So, if we can increase the size of the initial runs we can make it go even faster.

If the initial runs are 200,000 records each, we have a 400-way merge

We have 400 buffers each holding 1/800 of the run.

Seeks is 800 seeks/run * 400 runs = 320,000 seeks

How can we do this without buying more memory?

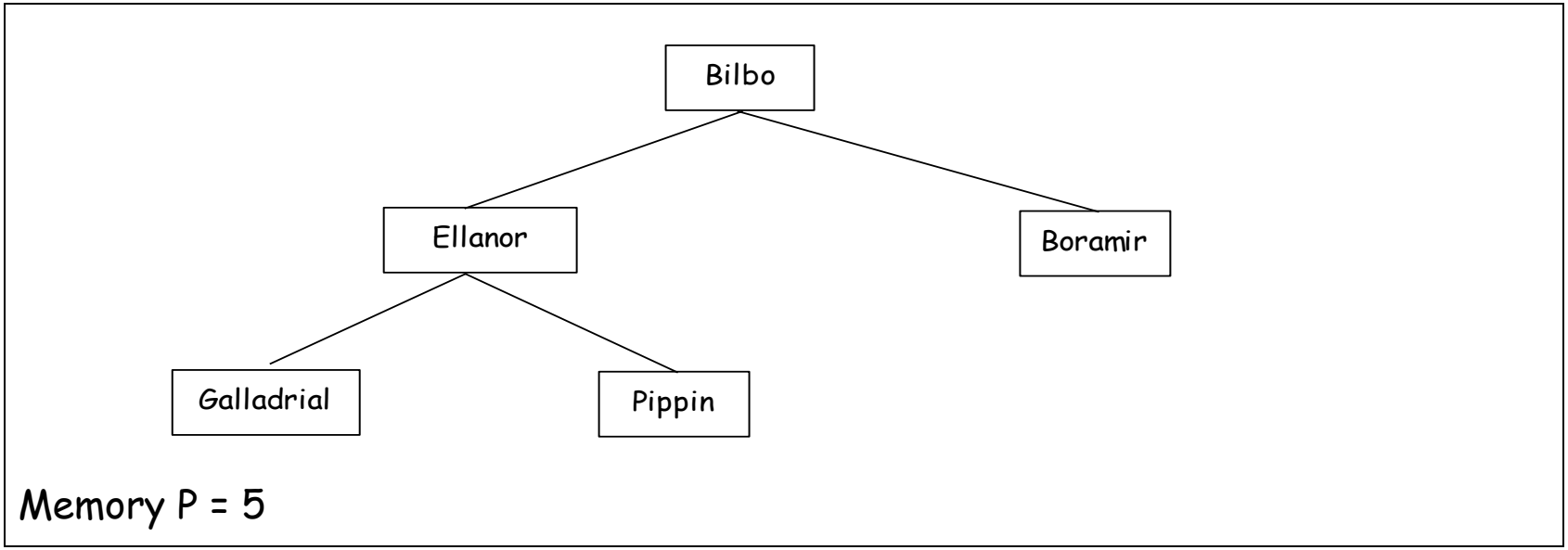
Trade time in memory for seeks.

Replacement Selection

1. Read a collection of records and sort them using heapsort creating the **primary heap**

Input Stream: Ellanor is the front, Rose is the end

Ellanor	Boramir	Bilbo	Galladrial	Pippin	Thor	Babs	Dirk	Arthur	Merry	Rose



Memory P = 5

2. Instead of writing the entire primary heap in sorted order, write only the record whose key has the lowest value.

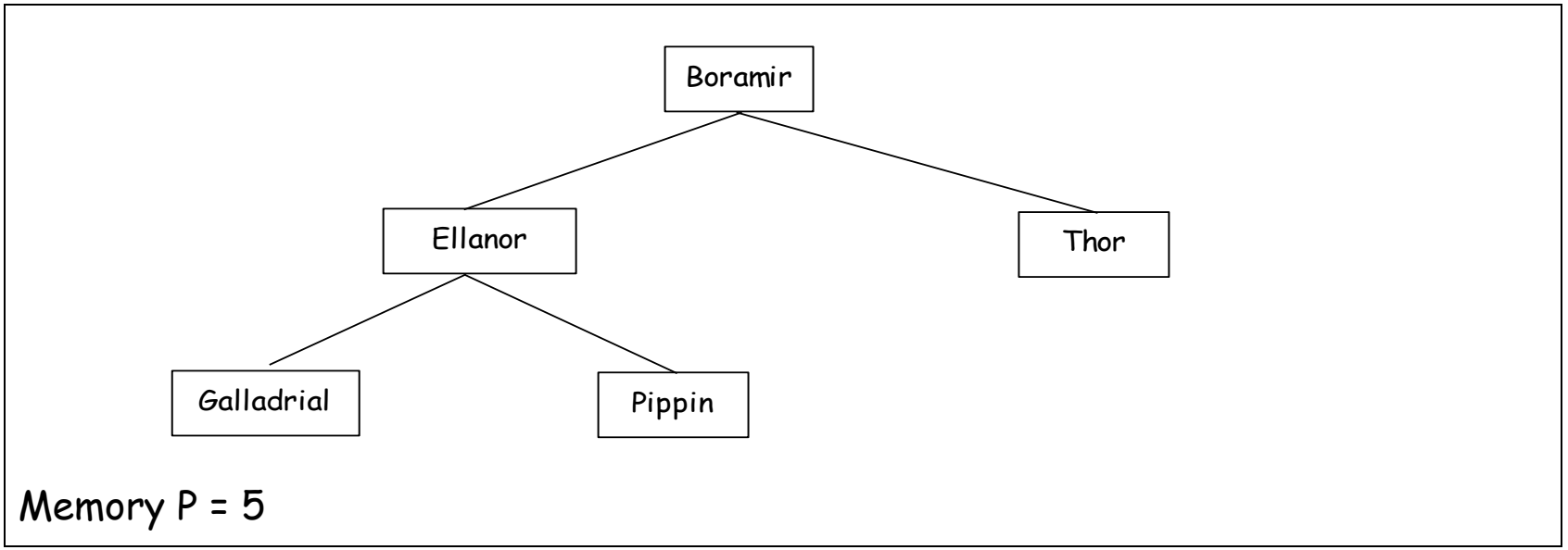
OUTPUT RUN: Bilbo

3. Bring in a new record, then compare the value of its key with that of the key that has just been output.
   a. If the new key value is higher, insert the new record into its proper place in the primary heap along with the other records that are being selected for output.
   b. If the new record's key value is lower, place the record in *secondary heap* of records with key values lower than those already written.

4. Repeat step 3 as long as there are records left in the primary heap, and there are records to be read. When the primary heap is empty, make the secondary heap into the primary heap, then repeat steps 2 and 3.

GET: Thor

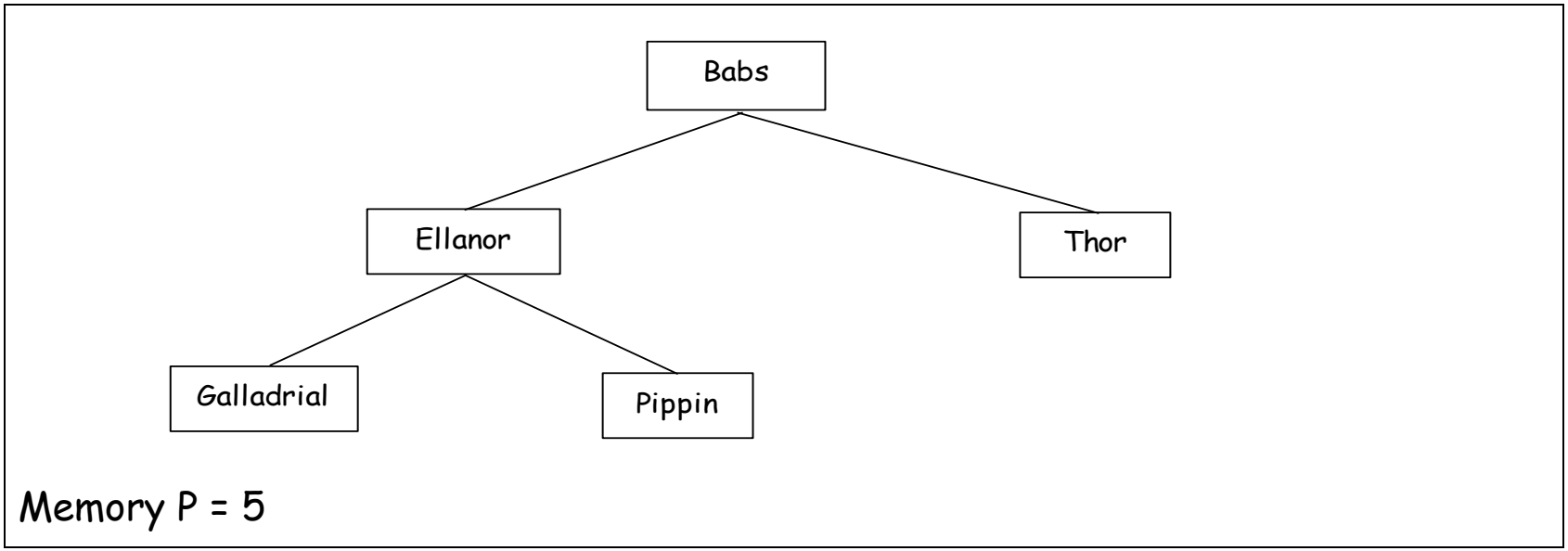
Ellanor	Boramir	Bilbo	Galladrial	Pippin	Thor	Babs	Dirk	Arthur	Merry	Rose



OUTPUT RUN: Bilbo Boramir

GET: Babs

Ellanor	Boramir	Bilbo	Galladrial	Pippin	Thor	Babs	Dirk	Arthur	Merry	Rose

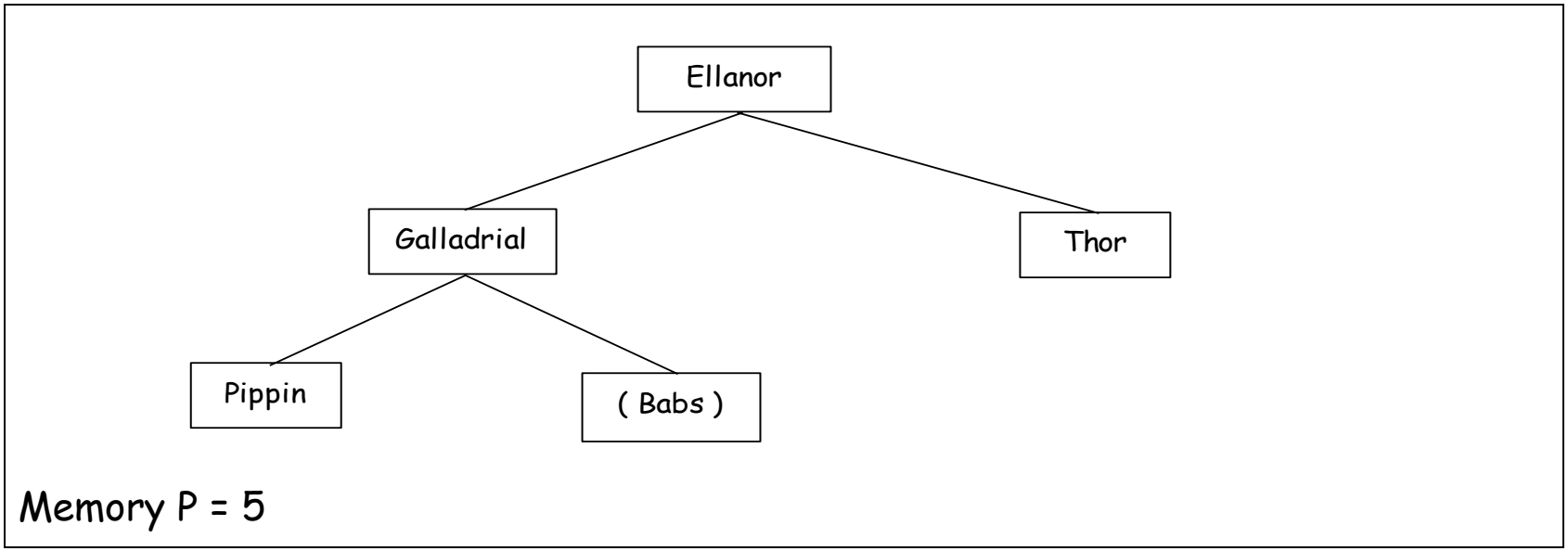


OUTPUT RUN: Bilbo Boramir

At this point it is too late to make Babs part of the output run so we put it away at the end of the heap for later use.

Now we can output Ellanor to the run

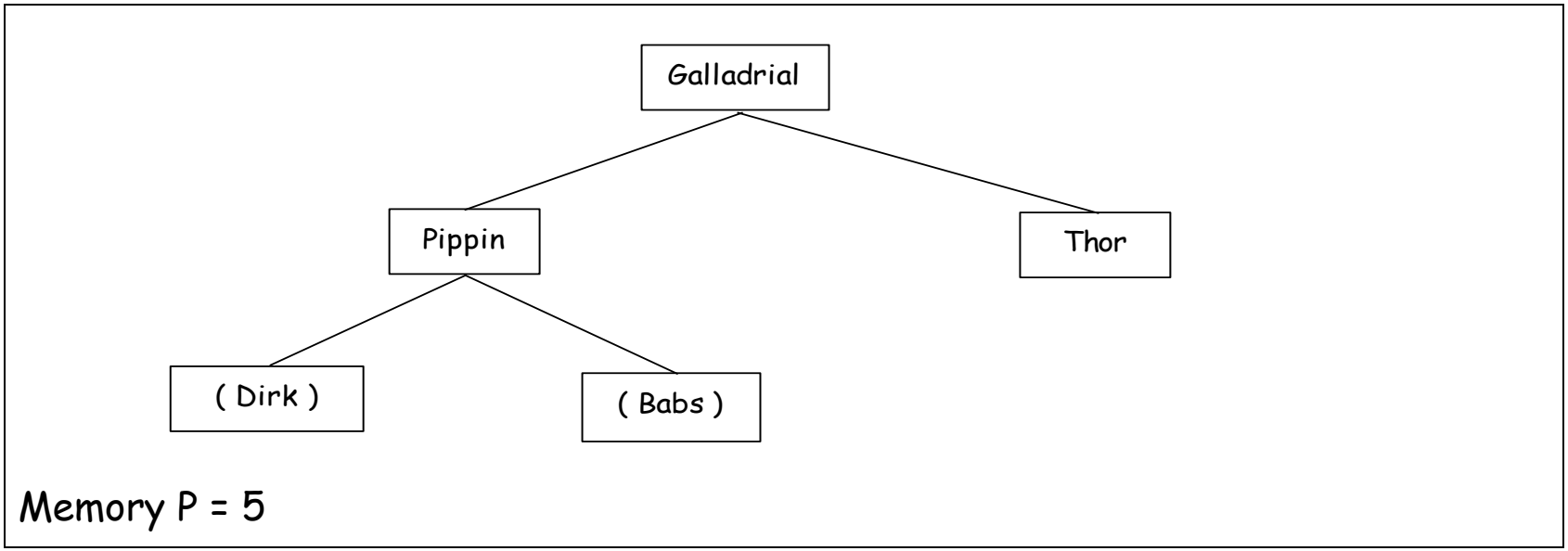
Ellanor	Boramir	Bilbo	Galladrial	Pippin	Thor	Babs	Dirk	Arthur	Merry	Rose



OUTPUT RUN: Bilbo Boramir Ellanor

GET: Dirk

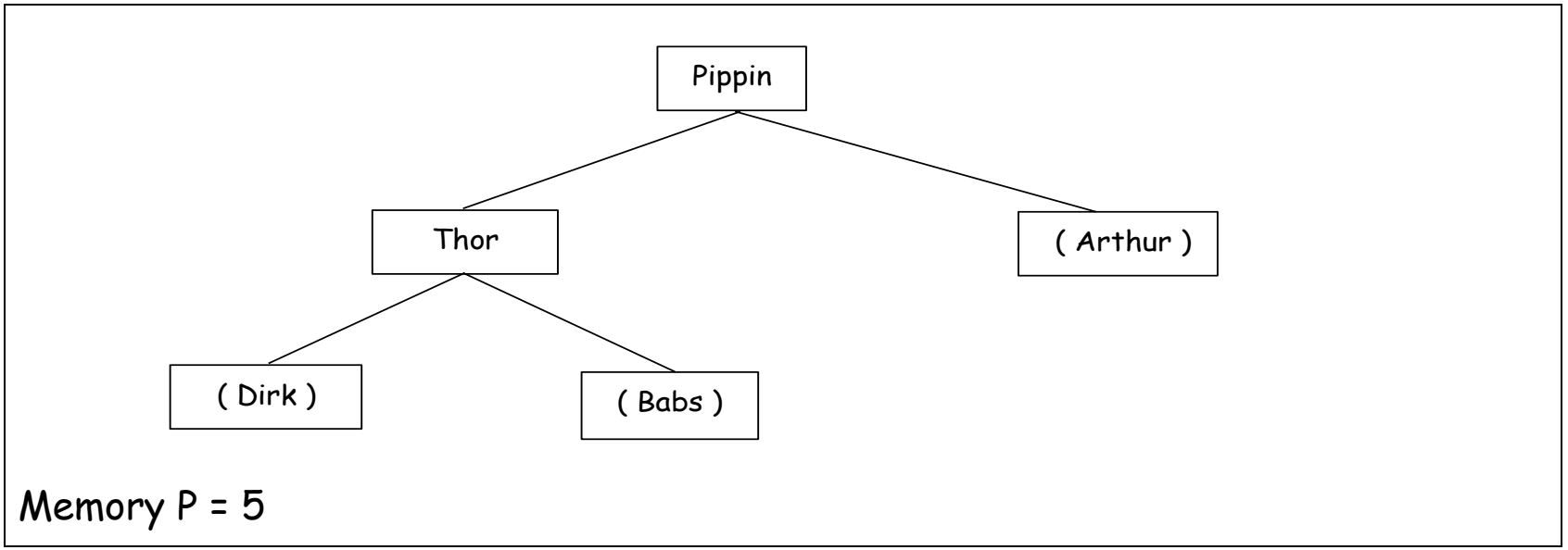
Ellanor	Boramir	Bilbo	Galladrial	Pippin	Thor	Babs	Dirk	Arthur	Merry	Rose



OUTPUT RUN: Bilbo Boramir Ellanor Galladrial

GET: Arthur

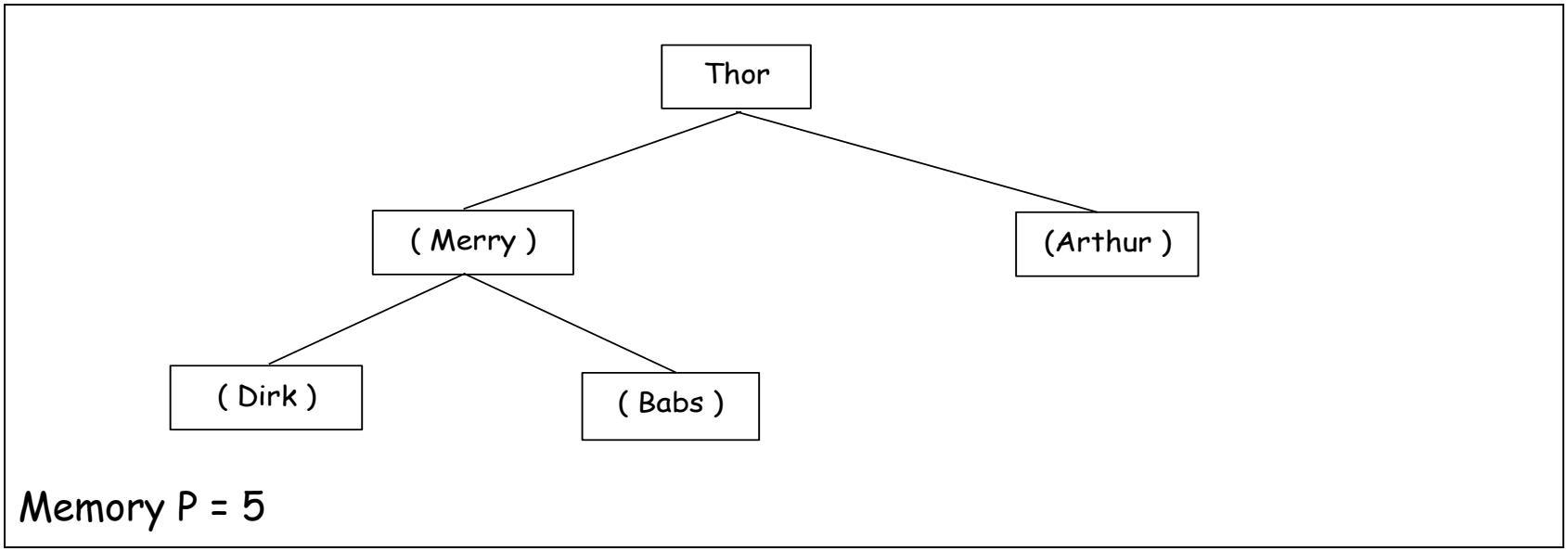
Ellanor	Boramir	Bilbo	Galladrial	Pippin	Thor	Babs	Dirk	Arthur	Merry	Rose



OUTPUT RUN: Bilbo Boramir Ellanor Galladrial Pippin

GET: Merry

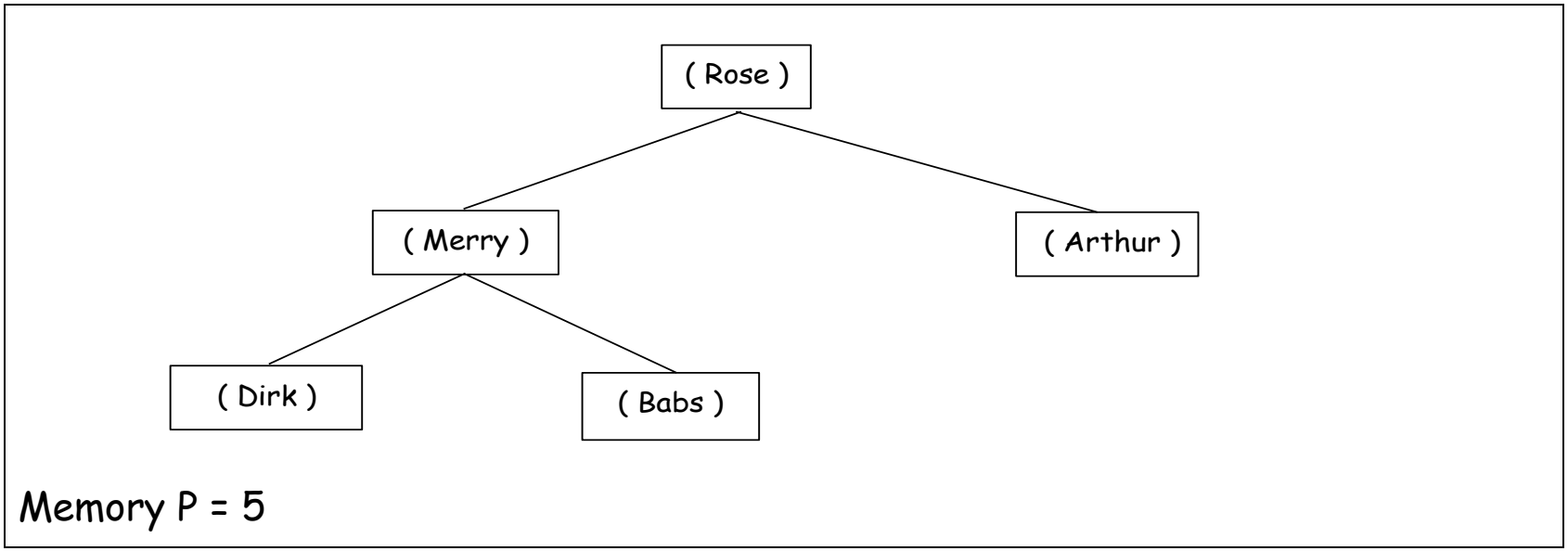
Ellanor	Boramir	Bilbo	Galladrial	Pippin	Thor	Babs	Dirk	Arthur	Merry	Rose



OUTPUT RUN: Bilbo Boramir Ellanor Galladrial Pippin Thor

GET: Rose

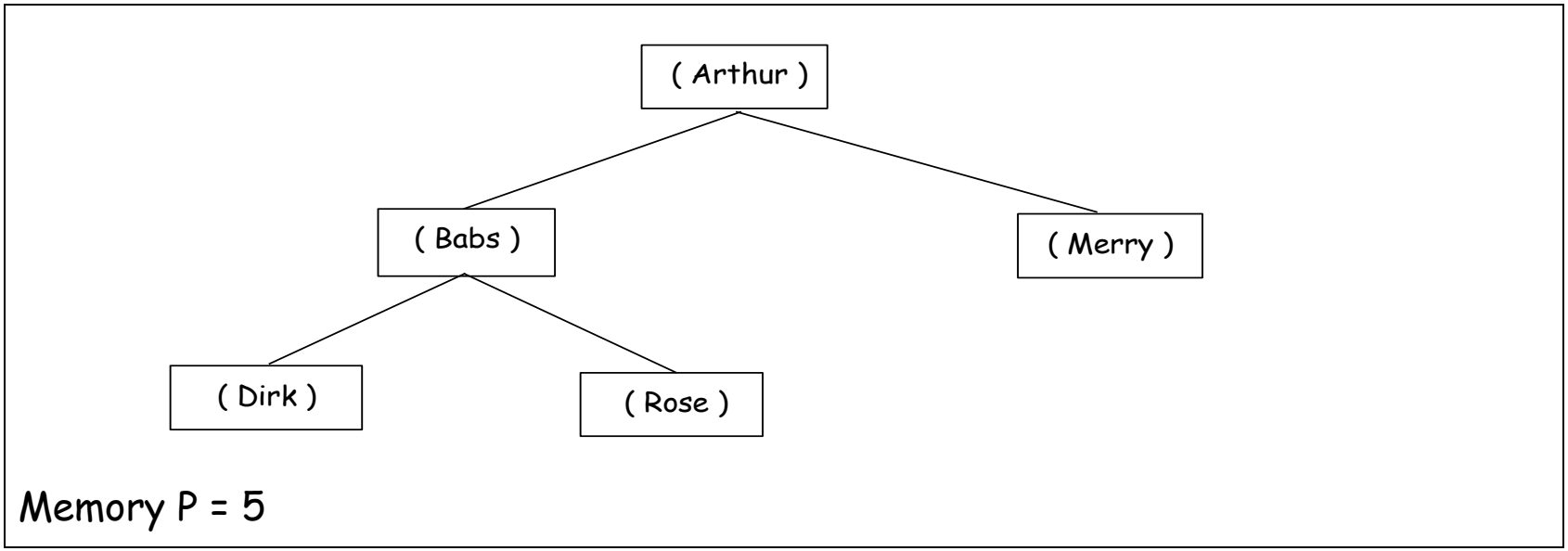
Ellanor	Boramir	Bilbo	Galladrial	Pippin	Thor	Babs	Dirk	Arthur	Merry	Rose



OUTPUT RUN: Bilbo Boramir Ellanor Galladrial Pippin Thor

At this point we heapify the secondary heap and start the process over.

Ellanor	Boramir	Bilbo	Galladrial	Pippin	Thor	Babs	Dirk	Arthur	Merry	Rose



OUTPUT RUN 1: Bilbo Boramir Ellanor Galladrial Pippin Thor

OUTPUT RUN 2: Arthur Babs Dirk Merry Rose

Now we can perform the merge on these two runs.

It turns out that the average run length for replacement selection is twice the length of the buffer we are using for sorting.

If the records are already partially in order the runs will be much longer.

If we add multi-step merging we get even more savings.

If you have 4 disk drives you can us a Fibonacci distribution to place the runs on the drives and make the sort even faster.

Lets look at the table from the text.

80 million records using both memory sort and replacement selection.

Merge order is equal to the number of runs formed

Approach	Number of records per seek to form runs	Size of runs formed	Number of runs formed	Number of seeks required to form runs	Merge order used	Total number of seeks	Total seek and rotational delay time
800 memory sorts followed by an 800-way merge	100,000	100,000	800	1600	800	681,600	2hr 5min
Replacement selection followed by 534-way merge (records in random order)	25,000	150,000	534	6,400	534	521,134	1hr 36min
Replacement selection followed by 200-way merge (records partially ordered)	25,000	400,000	200	6,400	200	206,400	38min

80 million records using both memory sort and replacement selection.

Each is followed by a two step merge

Approach	Number of records per seek to form runs	Size of runs formed	Number of runs formed	Merge pattern used	Number of seeks in merge phase	Total number of seeks	Total seek and rotational delay time
800 memory sorts	100,000	100,000	800	25 * 32 way then 25-way	25,600 20,000	127,200	24min
Replacement selection (records in random order)	25,000	150,000	534	19 * 28 way then 19-way	22,876 15,162	124,438	23min
Replacement selection (records partially ordered)	25,000	400,000	200	20 * 10 way then 20-way	8,000 16,000	110,400	20min

Read about sorting with tapes yourselves