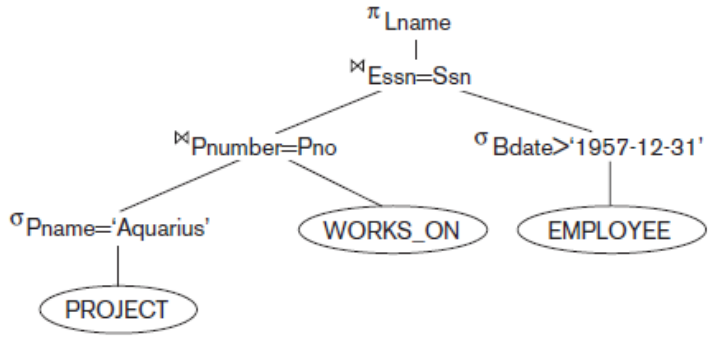
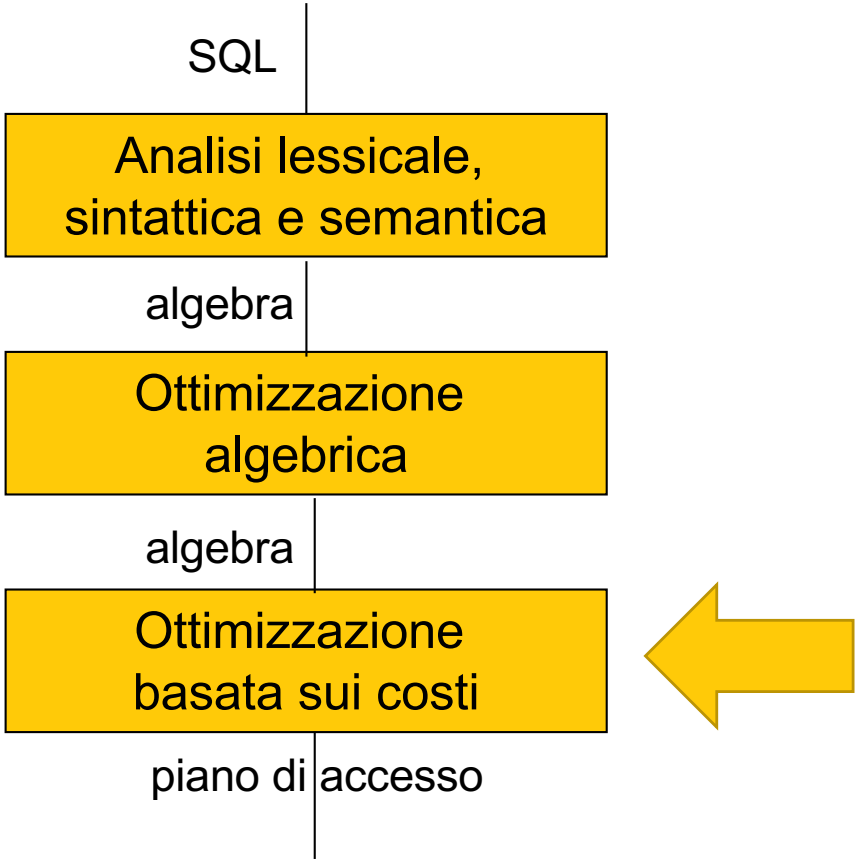
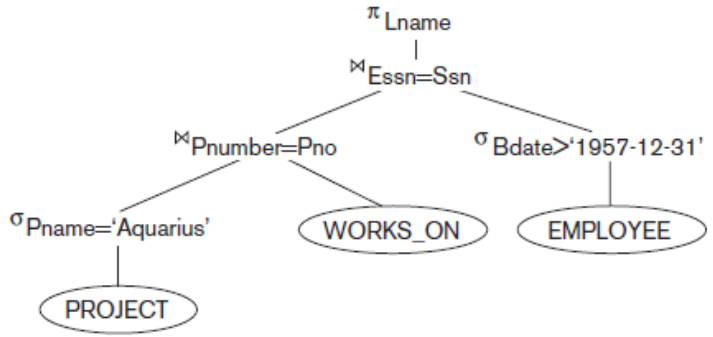
Esecuzione e ottimizzazione delle interrogazioni



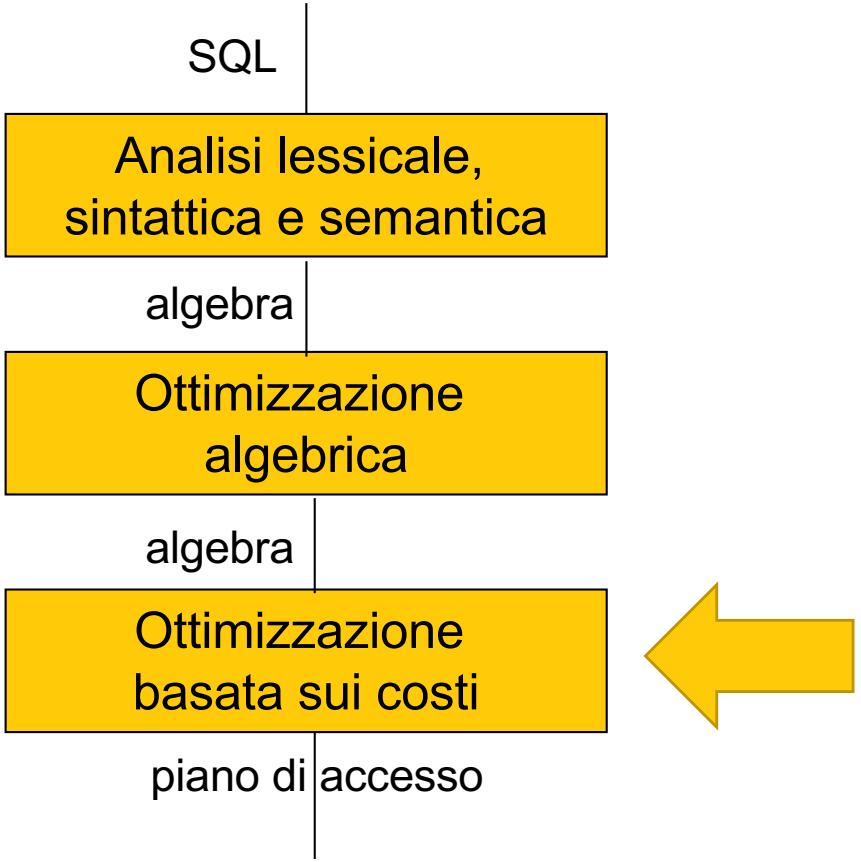
- Dato l'albero della interrogazione ottimizzato, il passo successivo è trovare una strategia per eseguire tutti i suoi operatori nell'ordine specificato



Esecuzione e ottimizzazione delle interrogazioni



- Il DBMS offre più strategie per eseguire gli operatori algebrici:
  - Algoritmi per la selezione
  - Algoritmi per il join
  - Algoritmi per la proiezione
- Molte strategie assumono i dati ordinati:
  - Algoritmo d'ordinamento



Algoritmo d'ordinamento esterno

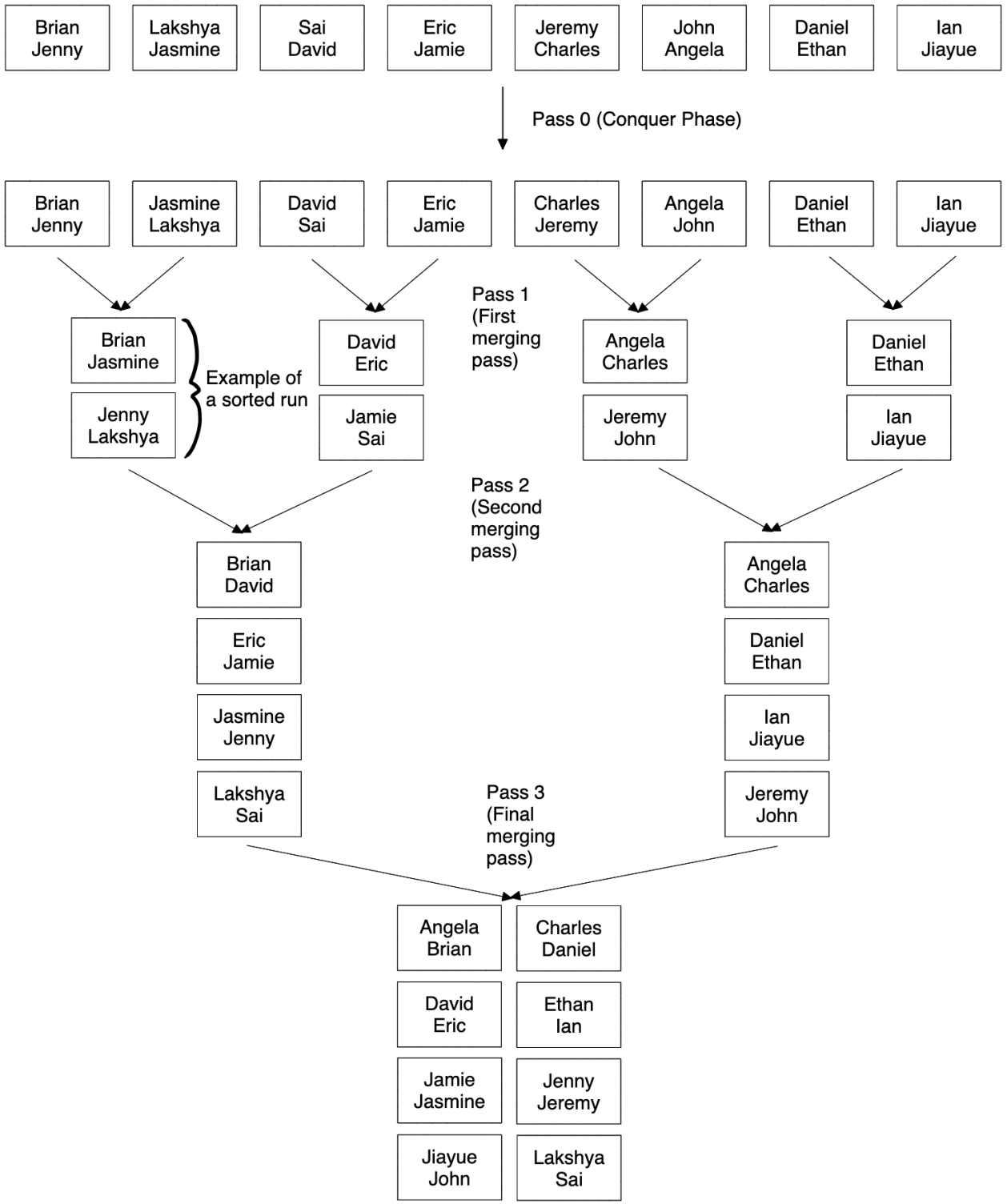
- Ordinamento interno: i dati da ordinare sono contenuti all'interno della memoria centrale in strutture in cui è possibile l'accesso diretto ai singoli elementi (array, etc)
- **Ordinamento esterno**: gli elementi da ordinare si trovano nella memoria secondaria. Non si accede direttamente ai singoli dati, ma a blocchi di dati che devono essere trasferiti nella memoria centrale per essere elaborati.

- L'algoritmo d'ordinamento esterno più utilizzato nei DBMS segue la strategia del **merge-sort (iterativo):**
  - **SORT:** si ordinano piccole parti del file principale, chiamate run
  - **MERGE:** si uniscono insieme, alcuni, run creando parti più grandi anch'esse ordinate
  - Si procede finché il file non è tutto ordinato

- Vediamo un primo esempio: 2-way merge sort

2-Way Merge Sort:

run = 1 blocco del file



Brian Jenny | Lakshya Jasmine | Sai David | Eric Jamie | Jeremy Charles | John Angela | Daniel Ethan | Ian Jiayue

2-Way Merge Sort:

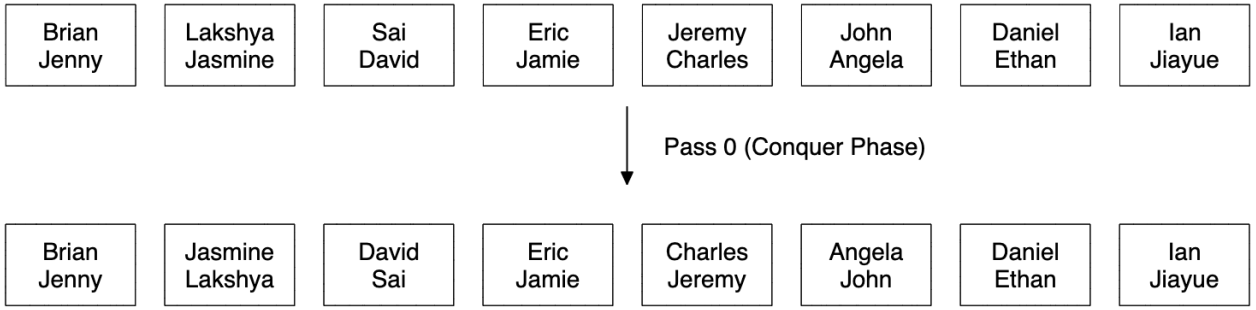
run = 1 blocco del file

- Passo 0: Ogni blocco del file (run) è caricato nel buffer, ordinato, e riscritto in memoria secondaria
  - Per questo passo è richiesto almeno un blocco nel buffer

2-Way Merge Sort:

run = 1 blocco del file

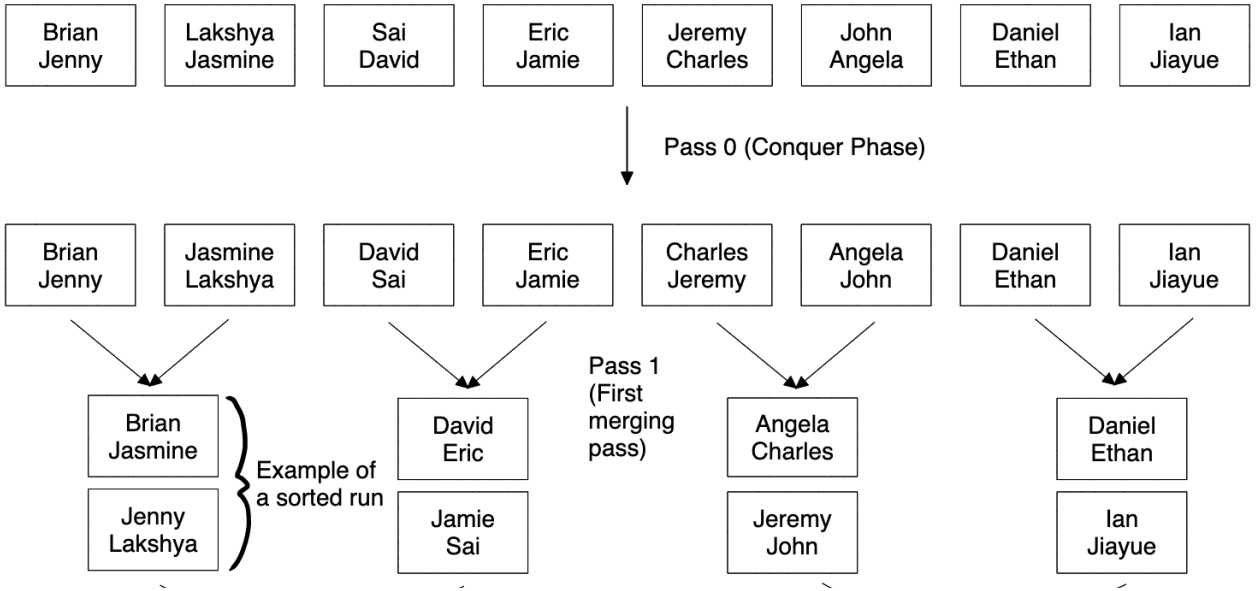
- Passo 0: Ogni blocco del file (run) è caricato nel buffer, ordinato, e riscritto in memoria secondaria
  - Per questo passo è richiesto almeno un blocco nel buffer



2-Way Merge Sort:

run = 1 blocco del file

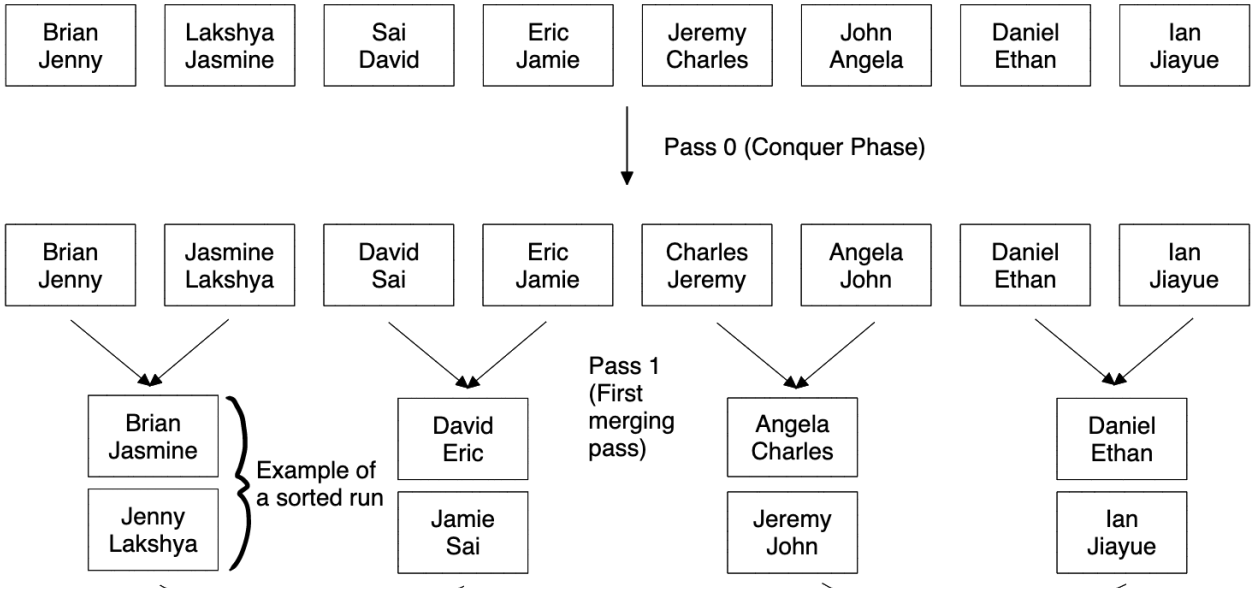
- Passo 0: Ogni blocco del file (run) è caricato nel buffer, ordinato, e riscritto in memoria secondaria
  - Per questo passo è richiesto almeno un blocco nel buffer
- Passo 1: Sono caricati due blocchi (run), riordinati e riscritti
  - Per questo passo sono richiesti almeno 3 blocchi nel buffer: 1 blocco per ogni run, e uno per memorizzare l'output ordinato

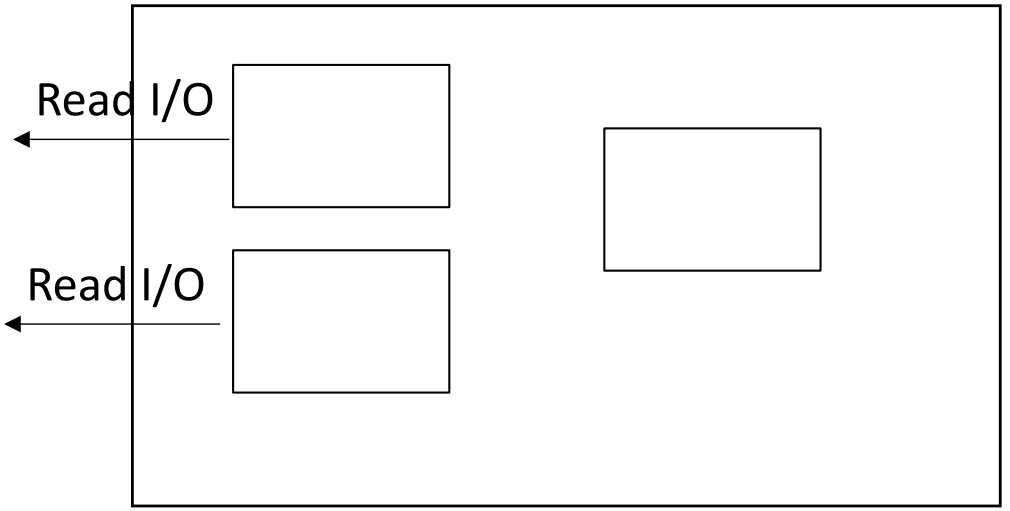


2-Way Merge Sort:

run = 1 blocco del file

- Passo 0: Ogni blocco del file (run) è caricato nel buffer, ordinato, e riscritto in memoria secondaria
  - Per questo passo è richiesto almeno un blocco nel buffer
- Passo 1: Sono caricati due blocchi (run), riordinati e riscritti
  - Per questo passo sono richiesti almeno 3 blocchi nel buffer: 1 blocco per ogni run, e uno per memorizzare l'output ordinato

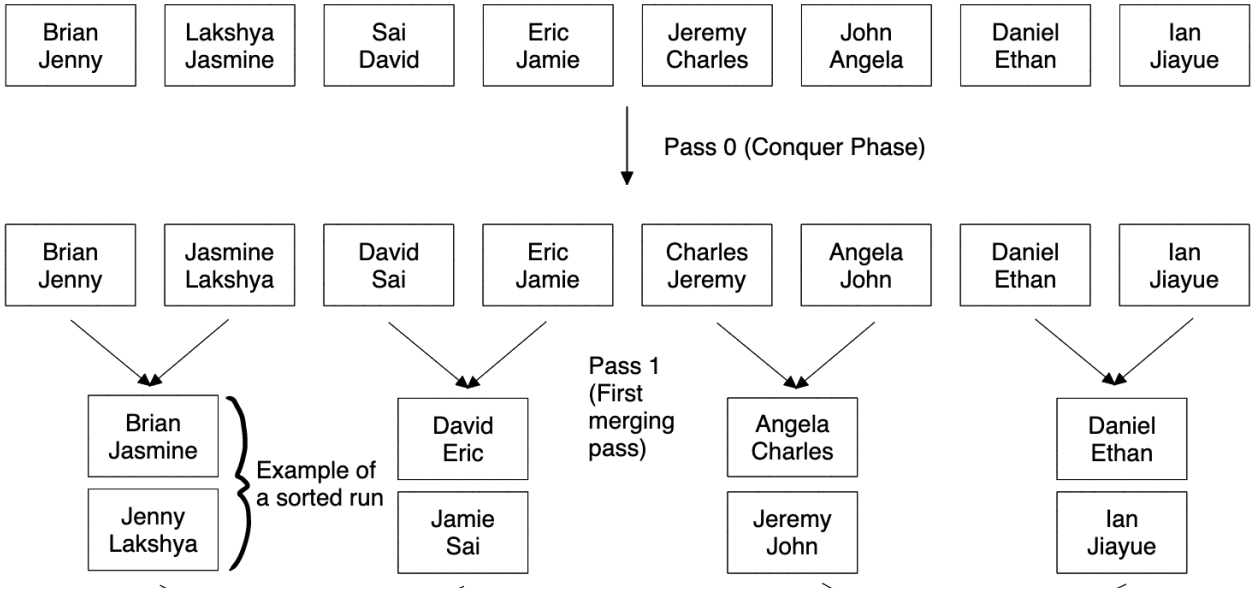


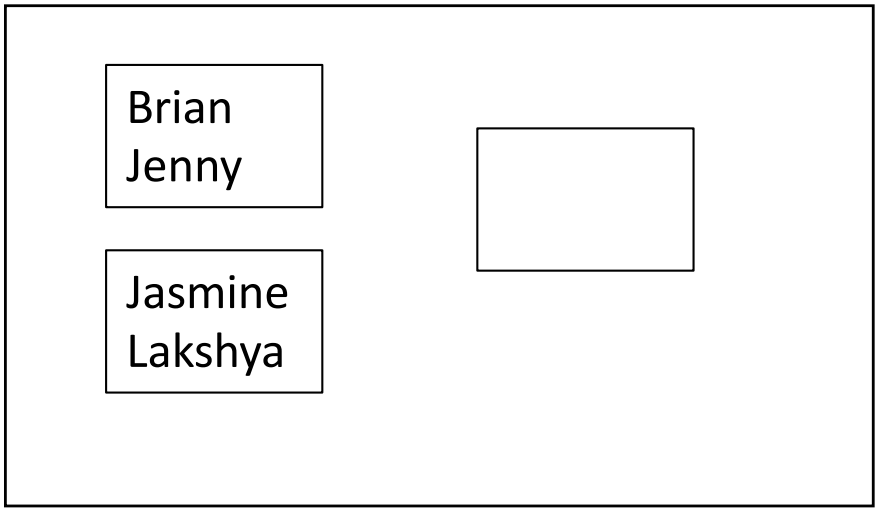


2-Way Merge Sort:

run = 1 blocco del file

- Passo 0: Ogni blocco del file (run) è caricato nel buffer, ordinato, e riscritto in memoria secondaria
  - Per questo passo è richiesto almeno un blocco nel buffer
- Passo 1: Sono caricati due blocchi (run), riordinati e riscritti
  - Per questo passo sono richiesti almeno 3 blocchi nel buffer: 1 blocco per ogni run, e uno per memorizzare l'output ordinato

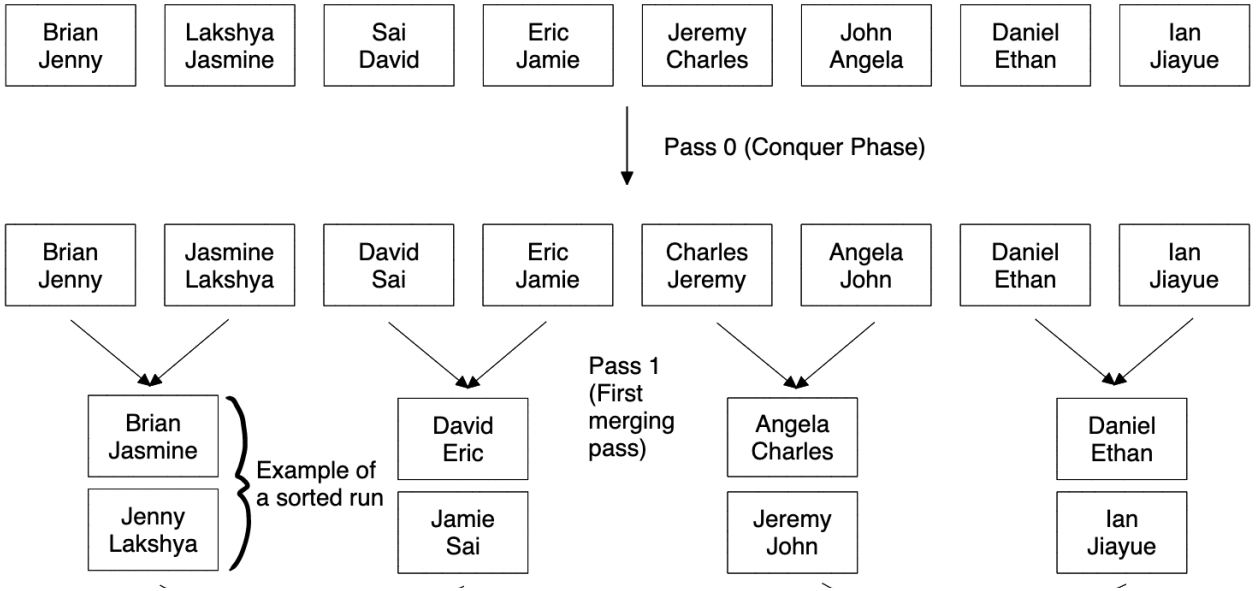


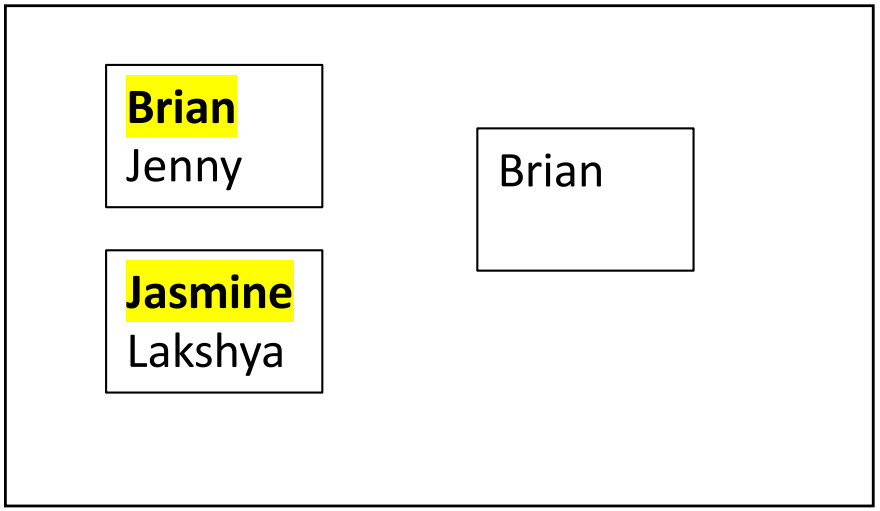


2-Way Merge Sort:

run = 1 blocco del file

- Passo 0: Ogni blocco del file (run) è caricato nel buffer, ordinato, e riscritto in memoria secondaria
  - Per questo passo è richiesto almeno un blocco nel buffer
- Passo 1: Sono caricati due blocchi (run), riordinati e riscritti
  - Per questo passo sono richiesti almeno 3 blocchi nel buffer: 1 blocco per ogni run, e uno per memorizzare l'output ordinato

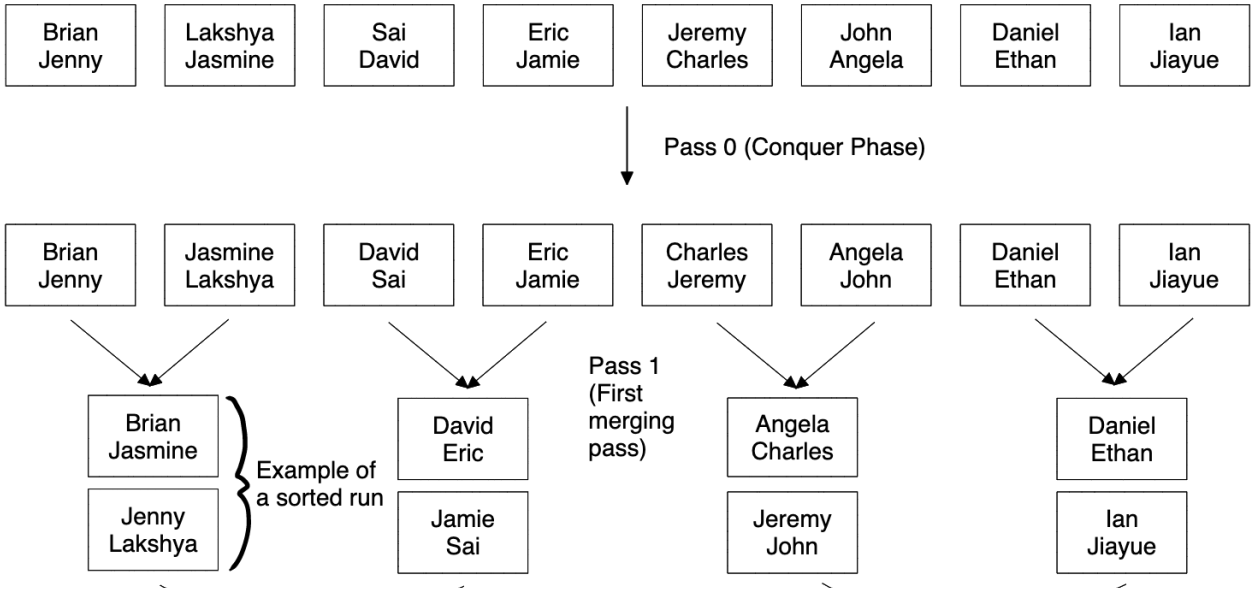


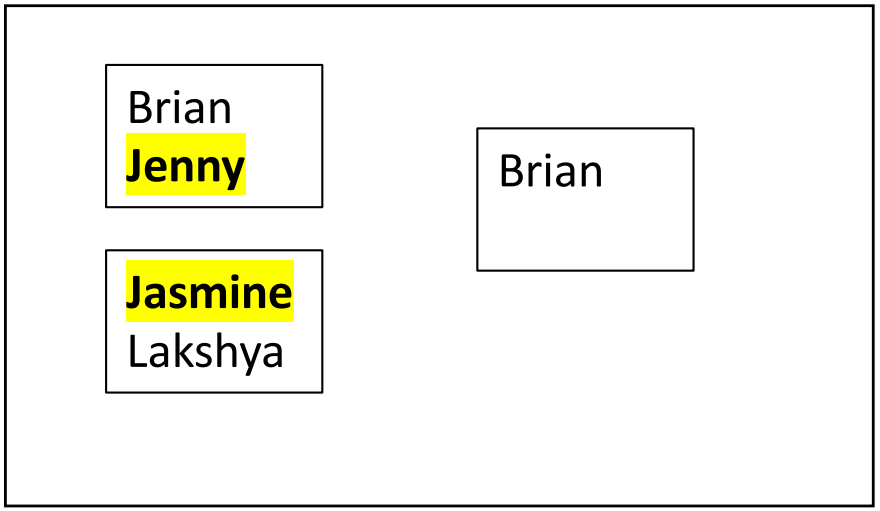


2-Way Merge Sort:

run = 1 blocco del file

- Passo 0: Ogni blocco del file (run) è caricato nel buffer, ordinato, e riscritto in memoria secondaria
  - Per questo passo è richiesto almeno un blocco nel buffer
- Passo 1: Sono caricati due blocchi (run), riordinati e riscritti
  - Per questo passo sono richiesti almeno 3 blocchi nel buffer: 1 blocco per ogni run, e uno per memorizzare l'output ordinato

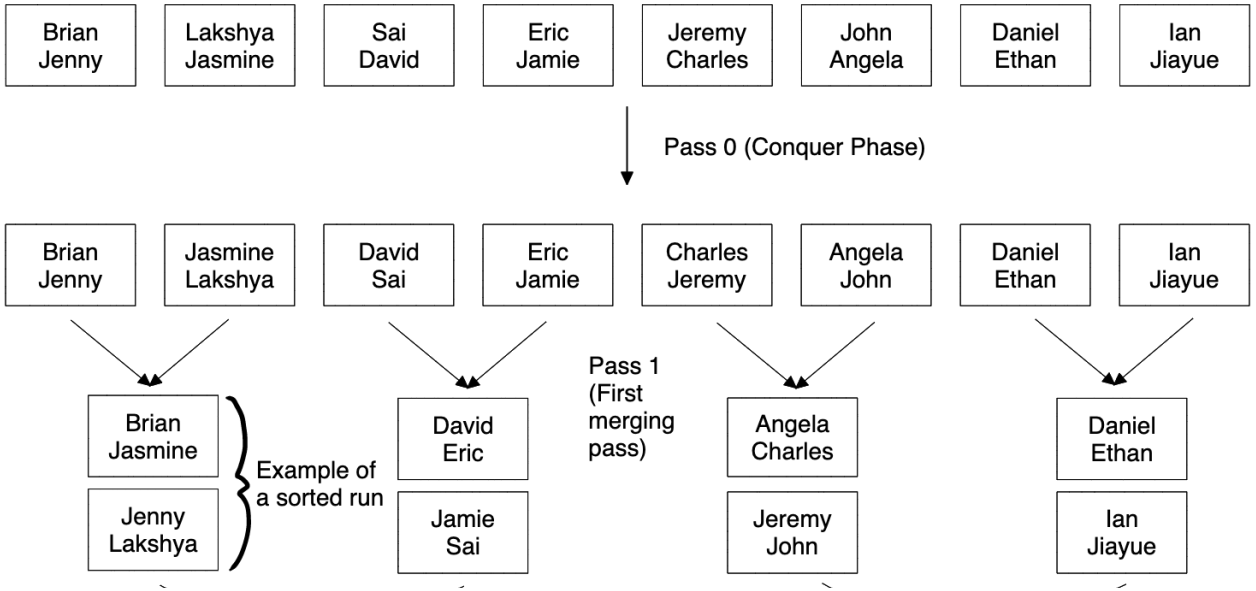


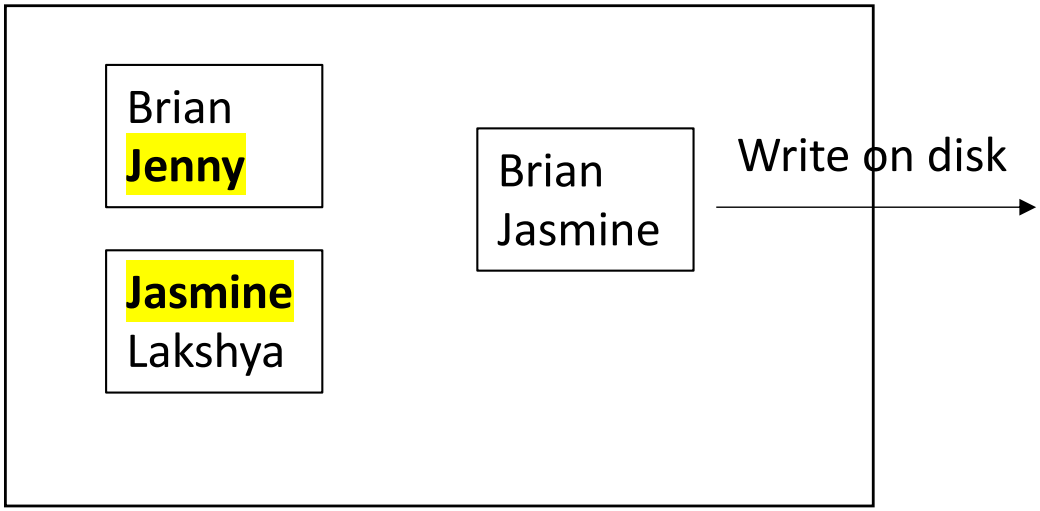


2-Way Merge Sort:

run = 1 blocco del file

- Passo 0: Ogni blocco del file (run) è caricato nel buffer, ordinato, e riscritto in memoria secondaria
  - Per questo passo è richiesto almeno un blocco nel buffer
- Passo 1: Sono caricati due blocchi (run), riordinati e riscritti
  - Per questo passo sono richiesti almeno 3 blocchi nel buffer: 1 blocco per ogni run, e uno per memorizzare l'output ordinato

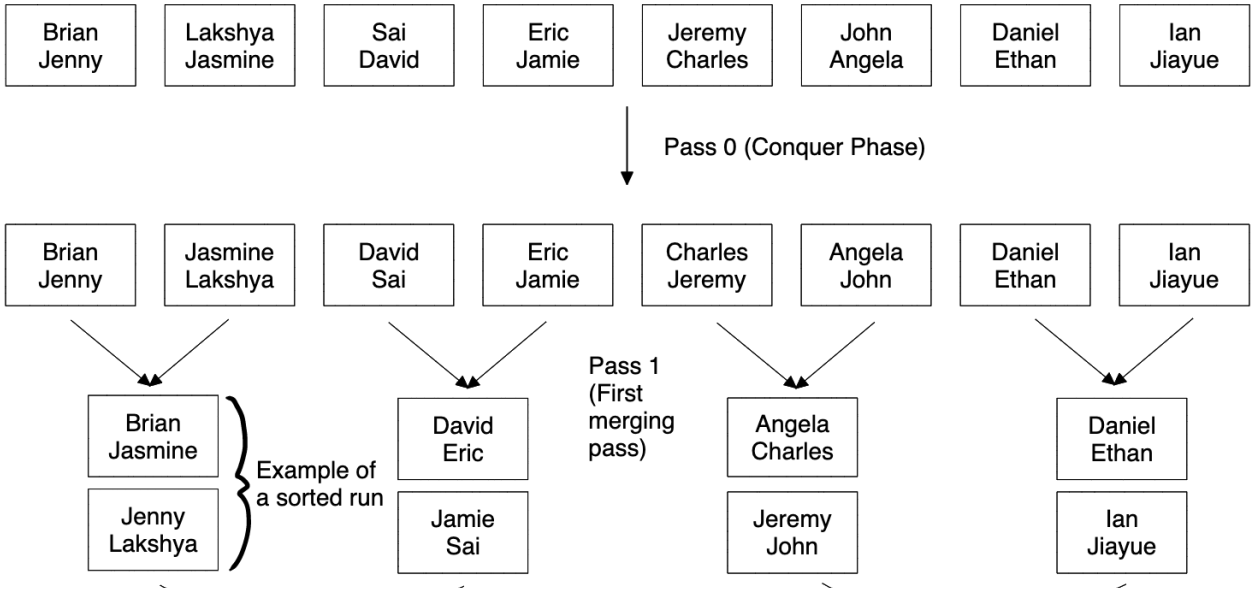


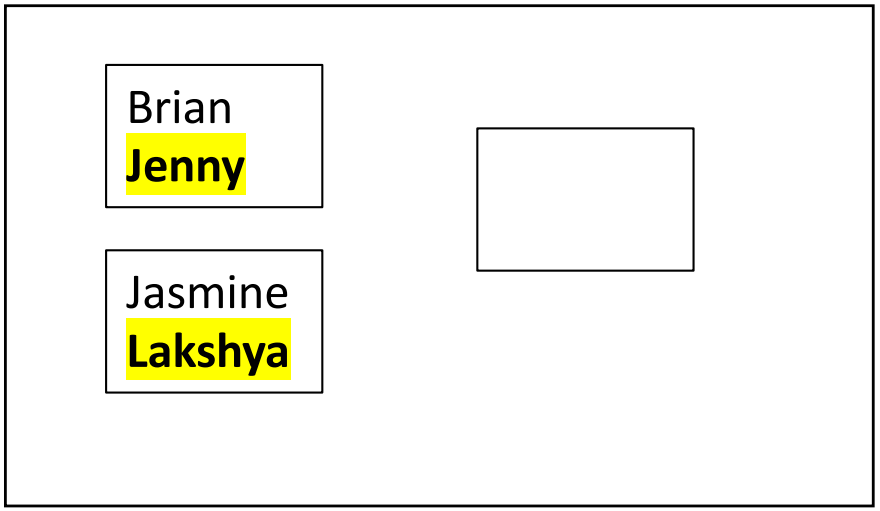


2-Way Merge Sort:

run = 1 blocco del file

- Passo 0: Ogni blocco del file (run) è caricato nel buffer, ordinato, e riscritto in memoria secondaria
  - Per questo passo è richiesto almeno un blocco nel buffer
- Passo 1: Sono caricati due blocchi (run), riordinati e riscritti
  - Per questo passo sono richiesti almeno 3 blocchi nel buffer: 1 blocco per ogni run, e uno per memorizzare l'output ordinato

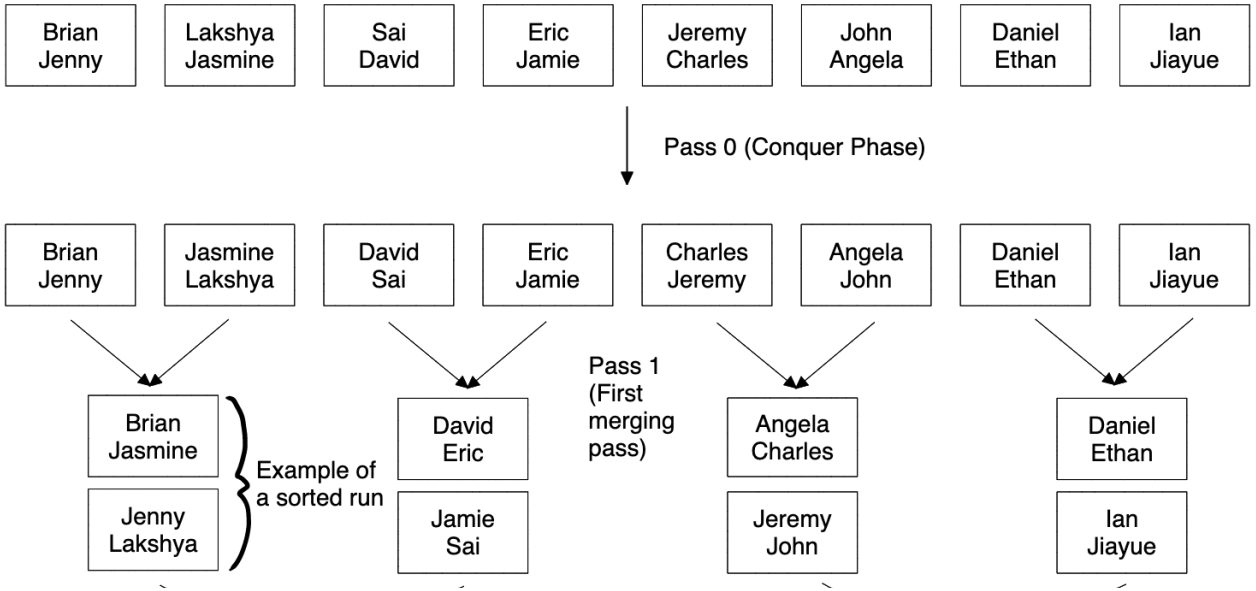


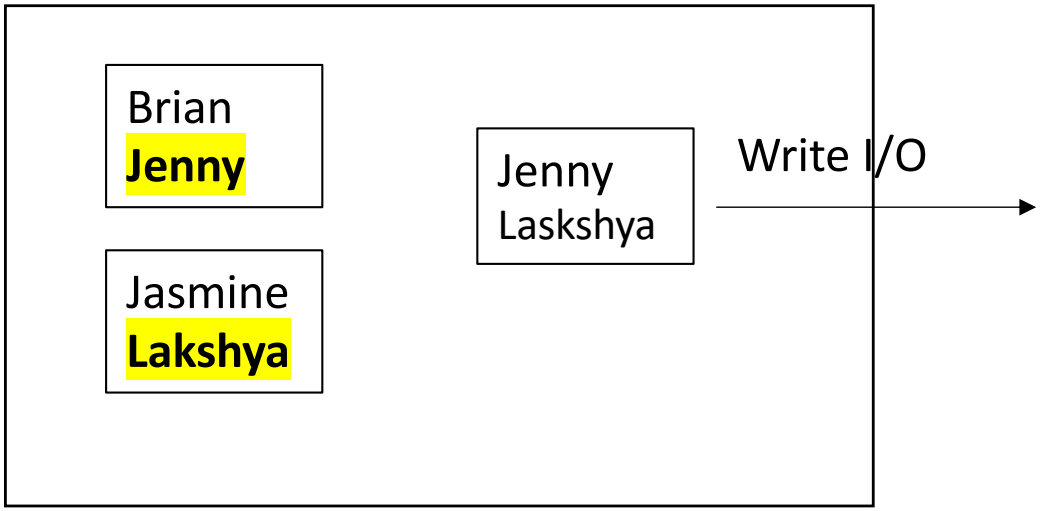


2-Way Merge Sort:

run = 1 blocco del file

- Passo 0: Ogni blocco del file (run) è caricato nel buffer, ordinato, e riscritto in memoria secondaria
  - Per questo passo è richiesto almeno un blocco nel buffer
- Passo 1: Sono caricati due blocchi (run), riordinati e riscritti
  - Per questo passo sono richiesti almeno 3 blocchi nel buffer: 1 blocco per ogni run, e uno per memorizzare l'output ordinato

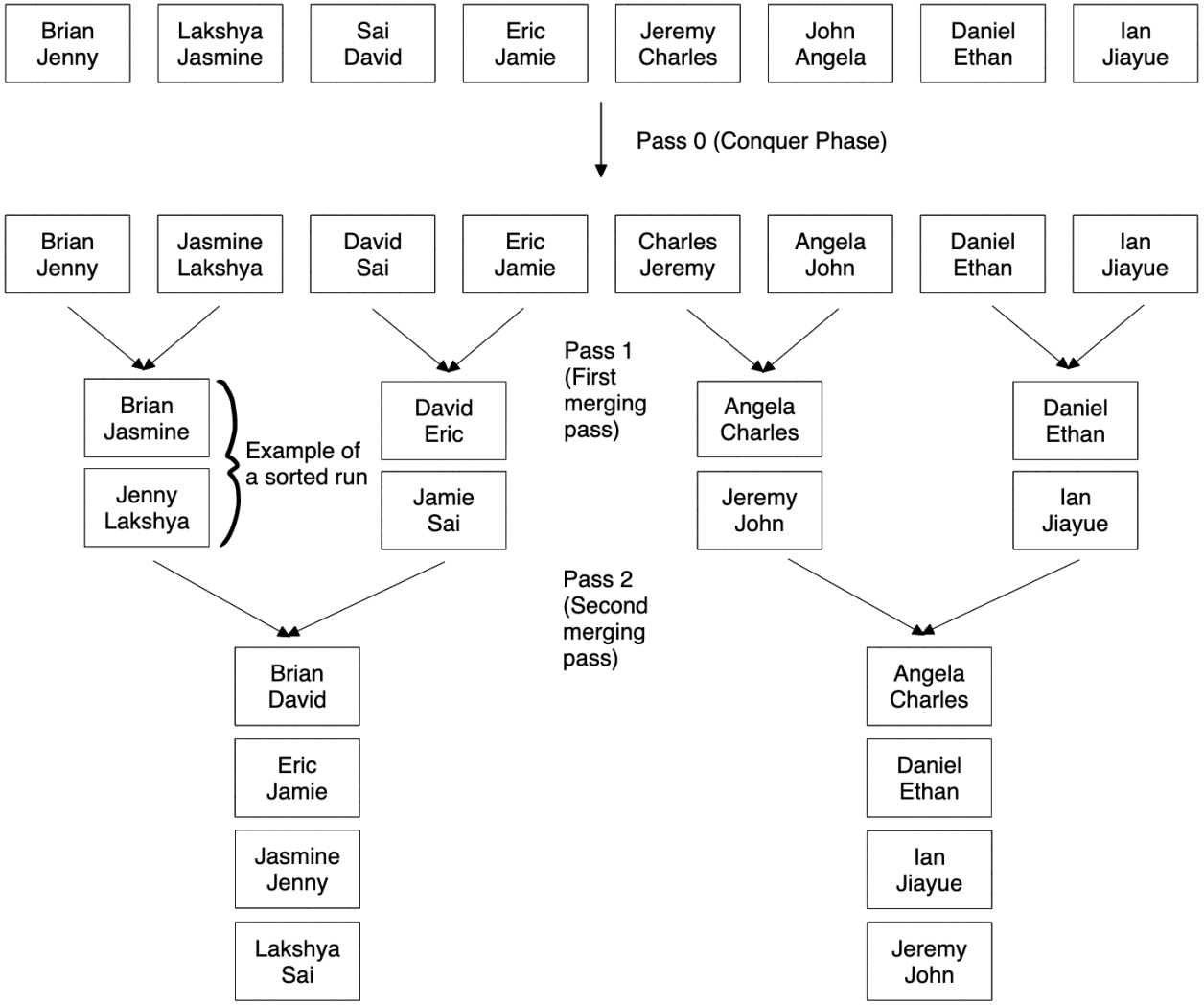




2-Way Merge Sort:

run = 1 blocco del file

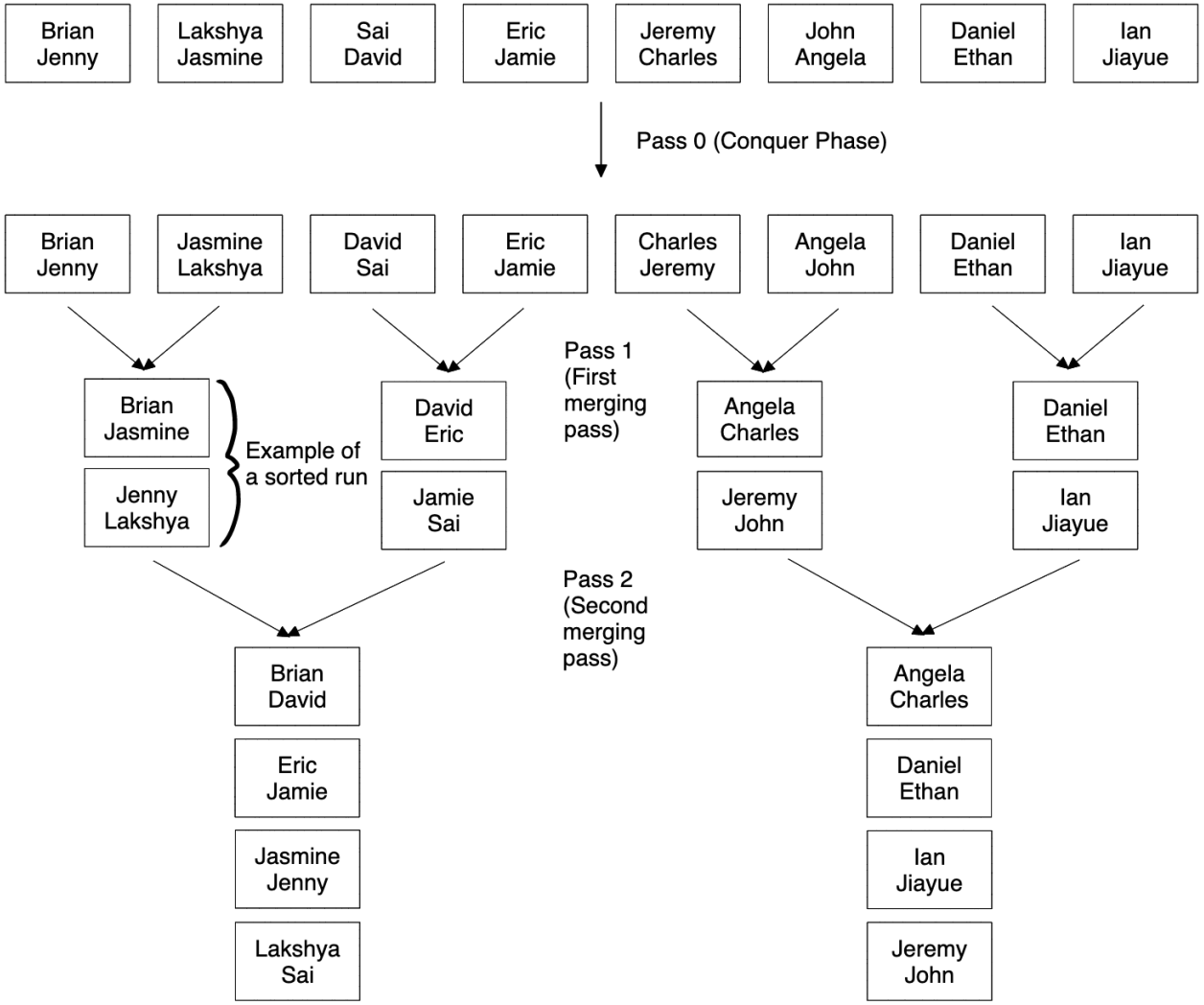
- Passo 0: Ogni blocco del file (run) è caricato nel buffer, ordinato, e riscritto in memoria secondaria
  - Per questo passo è richiesto almeno un blocco nel buffer
- Passo 1: Sono caricati due blocchi (run), riordinati e riscritti
  - Per questo passo sono richiesti almeno 3 blocchi nel buffer: 1 blocco per ogni run, e uno per memorizzare l'output ordinato
- Passo n: i run hanno n. blocchi >2.

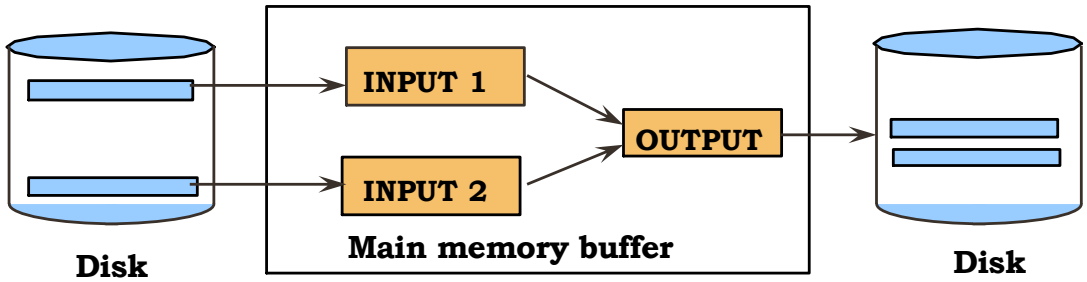


2-Way Merge Sort:

run = 1 blocco del file

- Passo 0: Ogni blocco del file (run) è caricato nel buffer, ordinato, e riscritto in memoria secondaria
  - Per questo passo è richiesto almeno un blocco nel buffer
- Passo 1: Sono caricati due blocchi (run), riordinati e riscritti
  - Per questo passo sono richiesti almeno 3 blocchi nel buffer: 1 blocco per ogni run, e uno per memorizzare l'output ordinato
- Passo n: i run hanno n. blocchi >2.

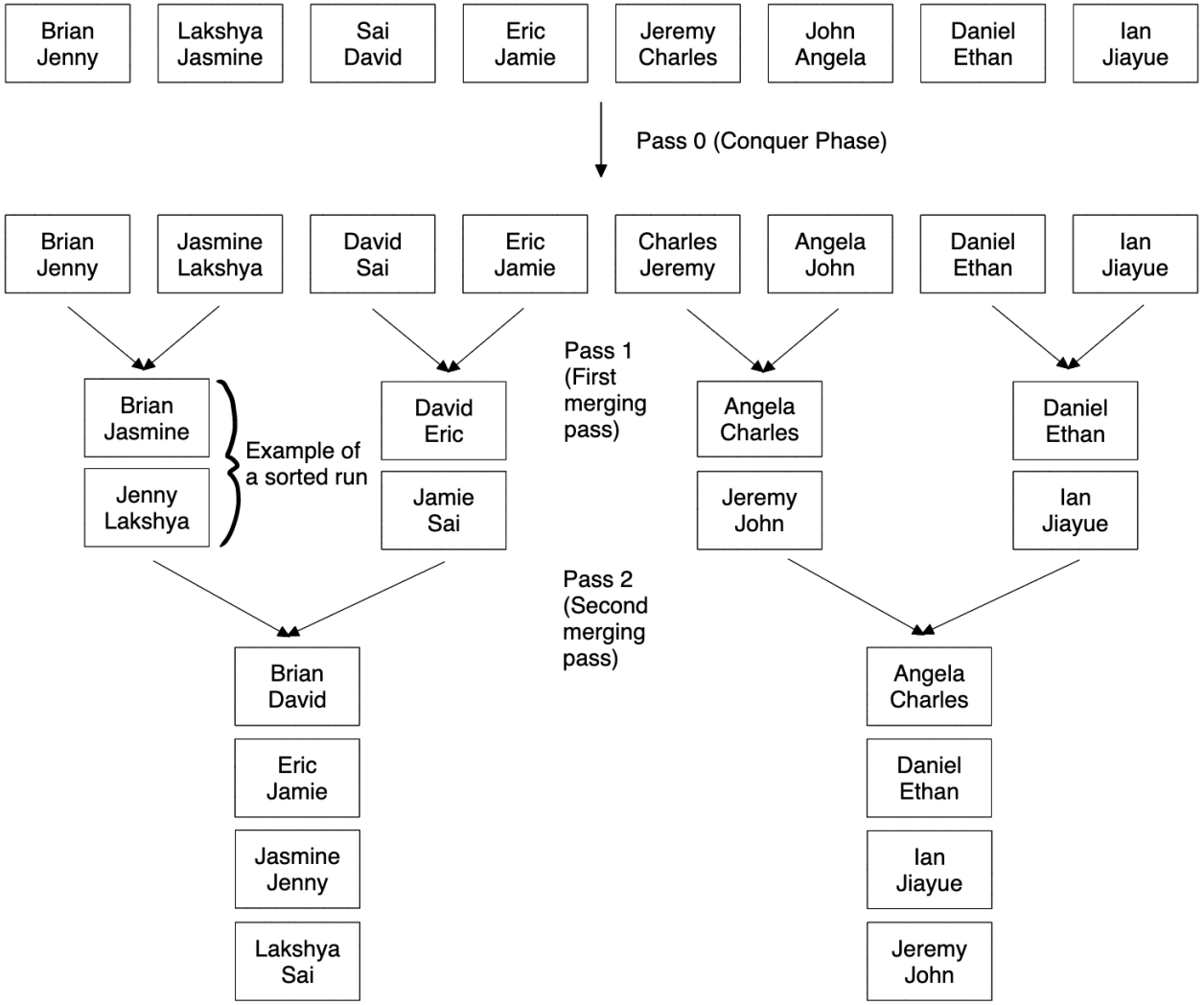


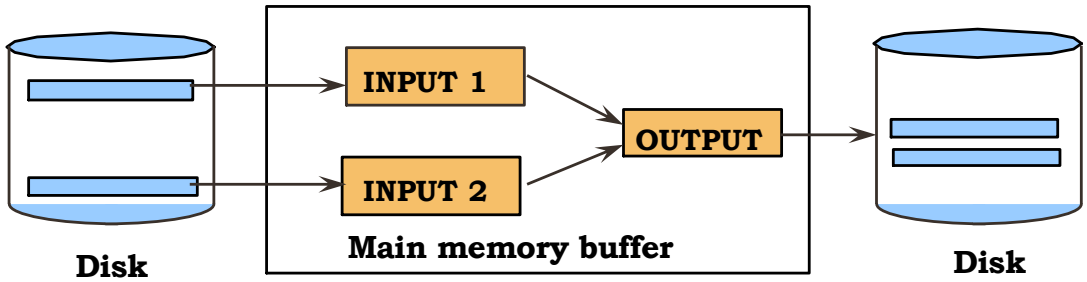


2-Way Merge Sort:

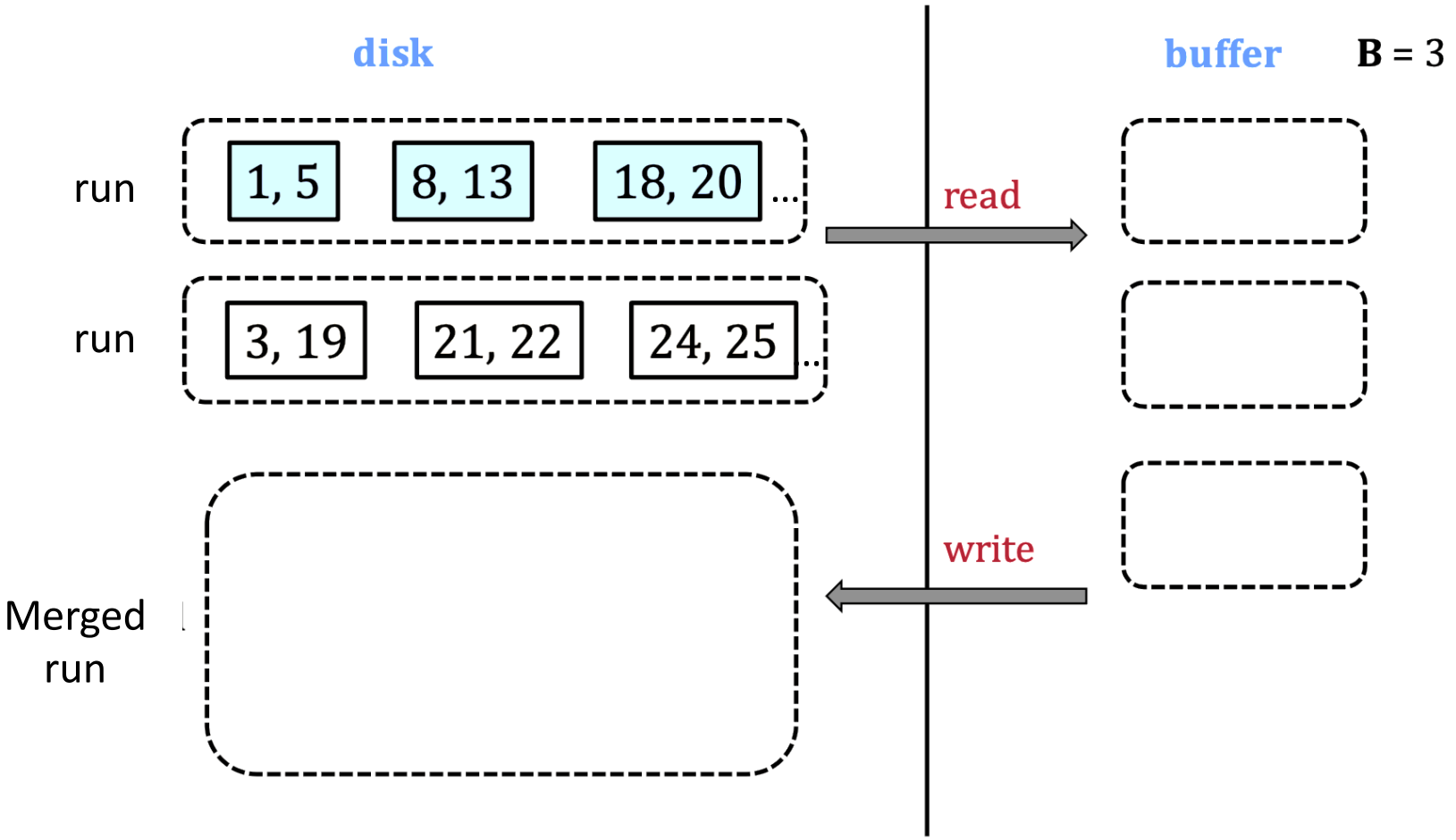
run = 1 blocco del file

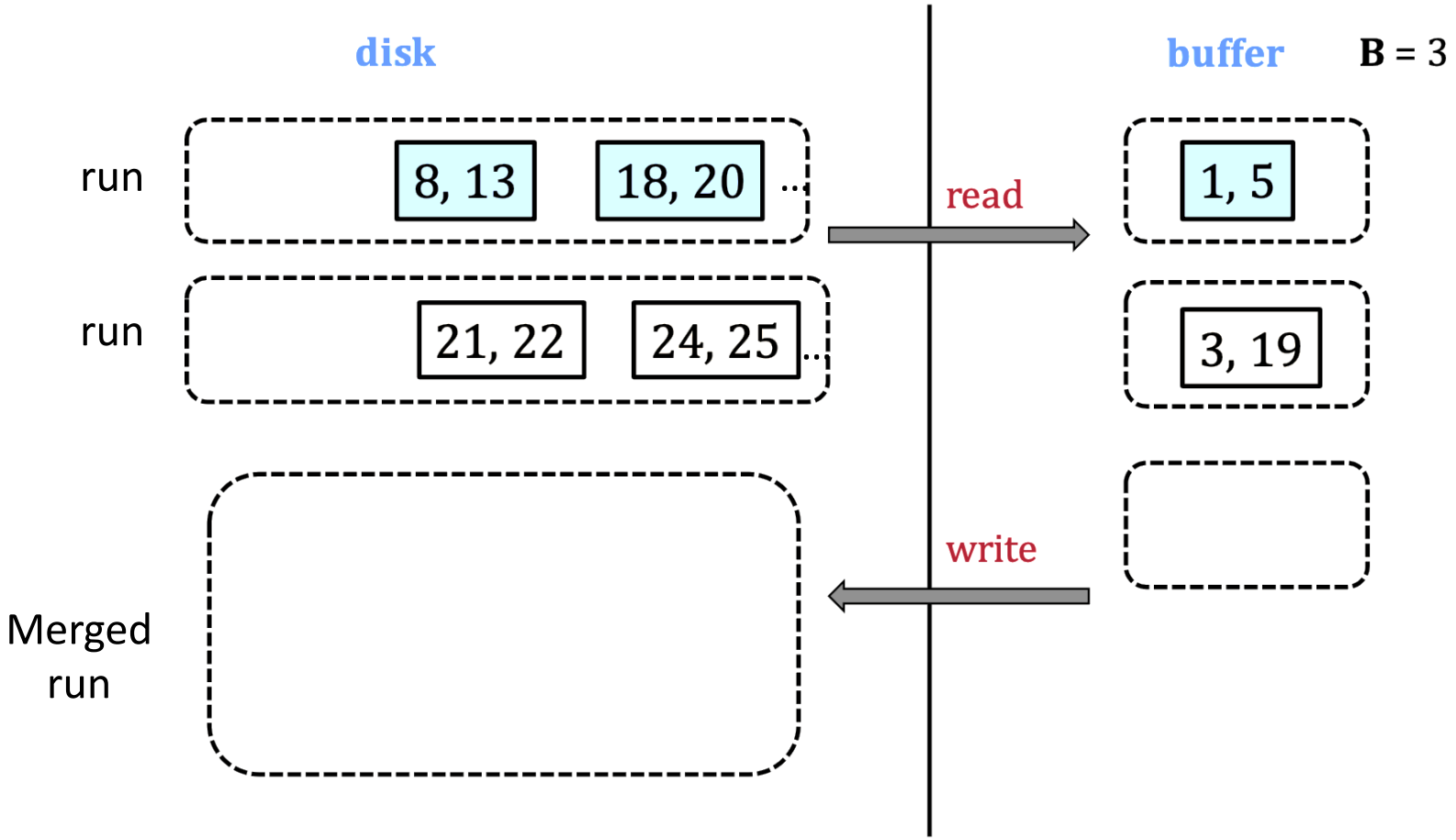
- Passo 0: Ogni blocco del file (run) è caricato nel buffer, ordinato, e riscritto in memoria secondaria
  - Per questo passo è richiesto almeno un blocco nel buffer
- Passo 1: Sono caricati due blocchi (run), riordinati e riscritti
  - Per questo passo sono richiesti almeno 3 blocchi nel buffer: 1 blocco per ogni run, e uno per memorizzare l'output ordinato
- Passo n: i run hanno n. blocchi >2. In memoria è caricato solo il primo blocco dei due run, i record vengono confrontati e memorizzati nel buffer d'output.

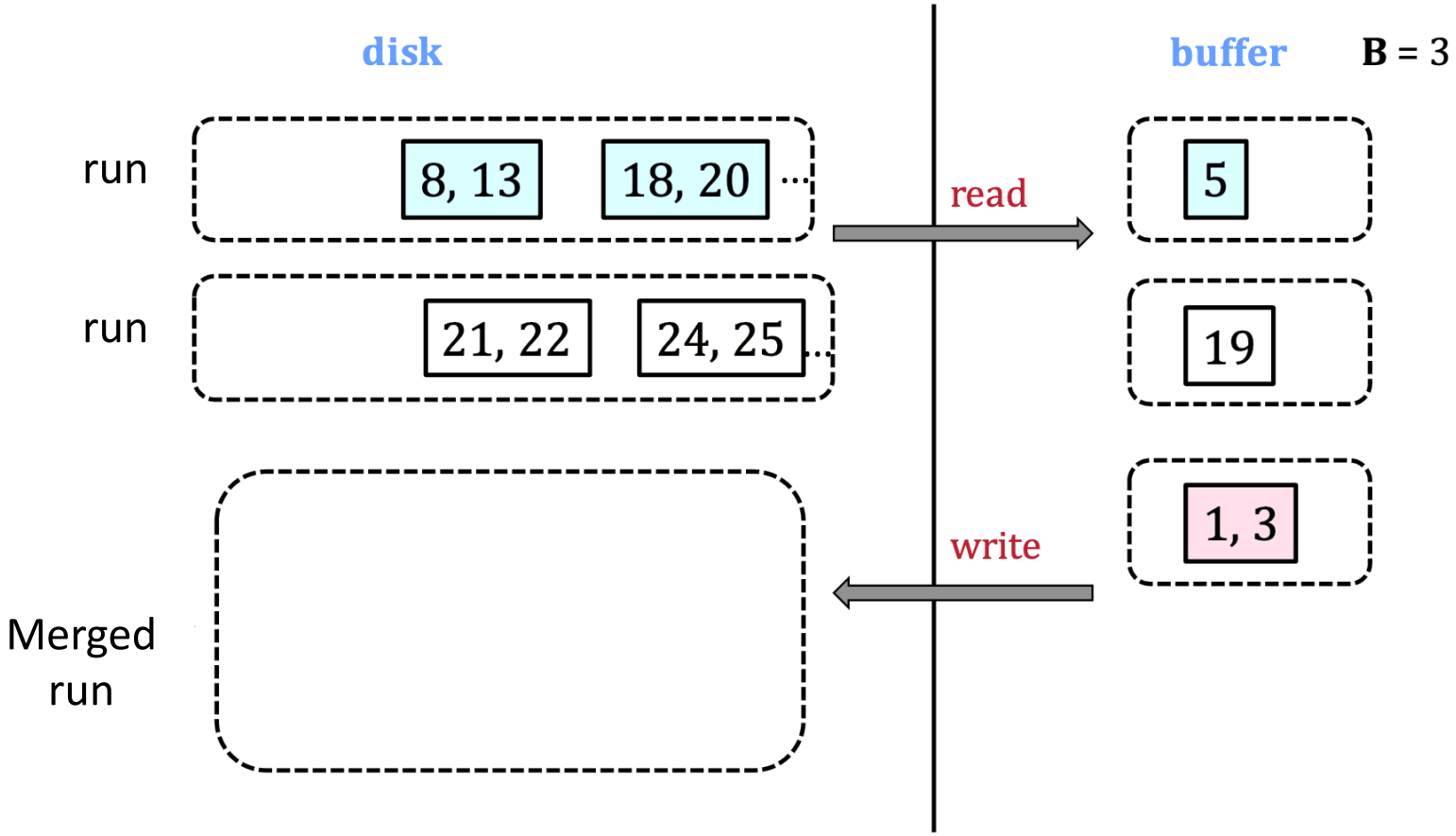


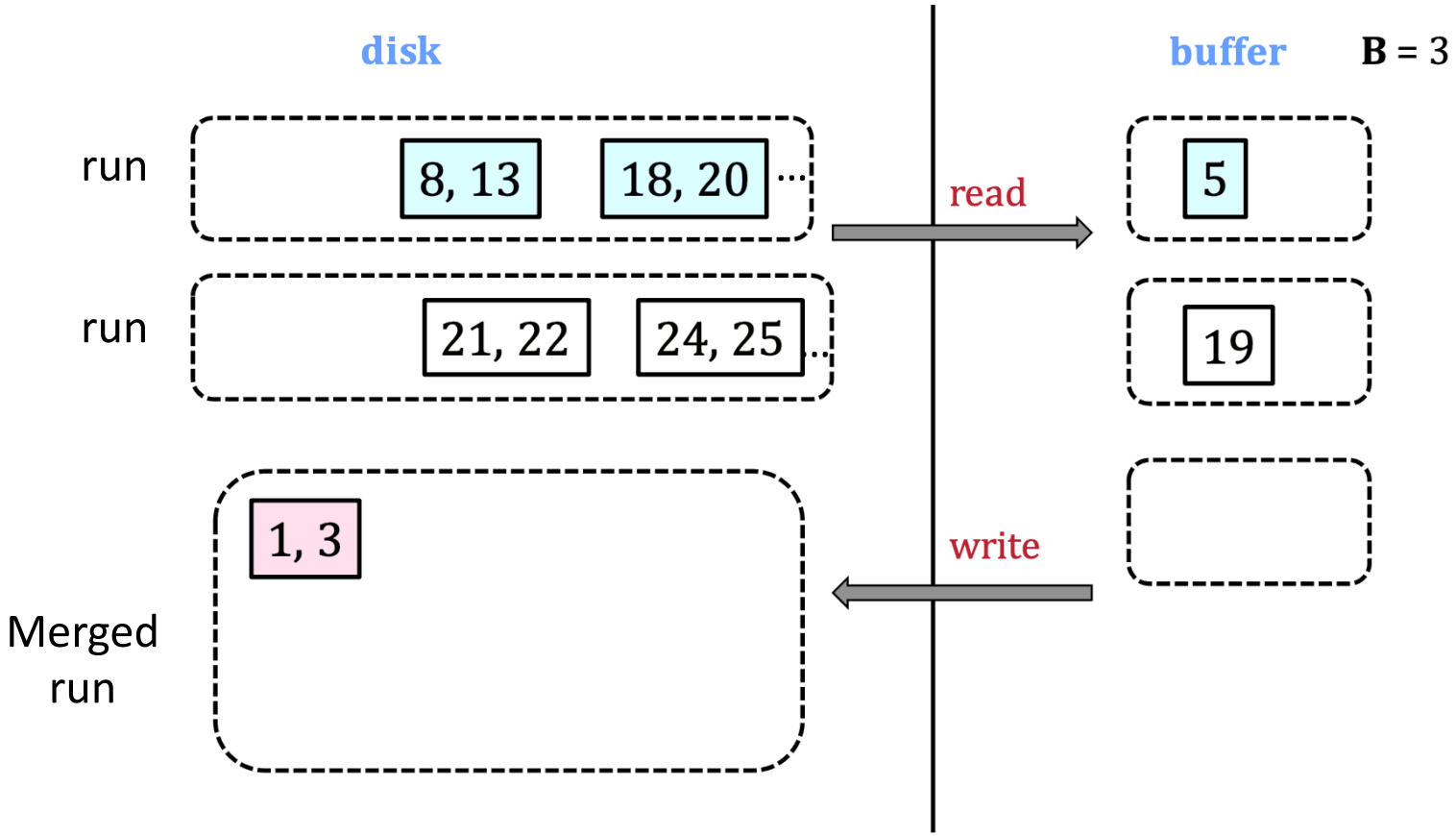


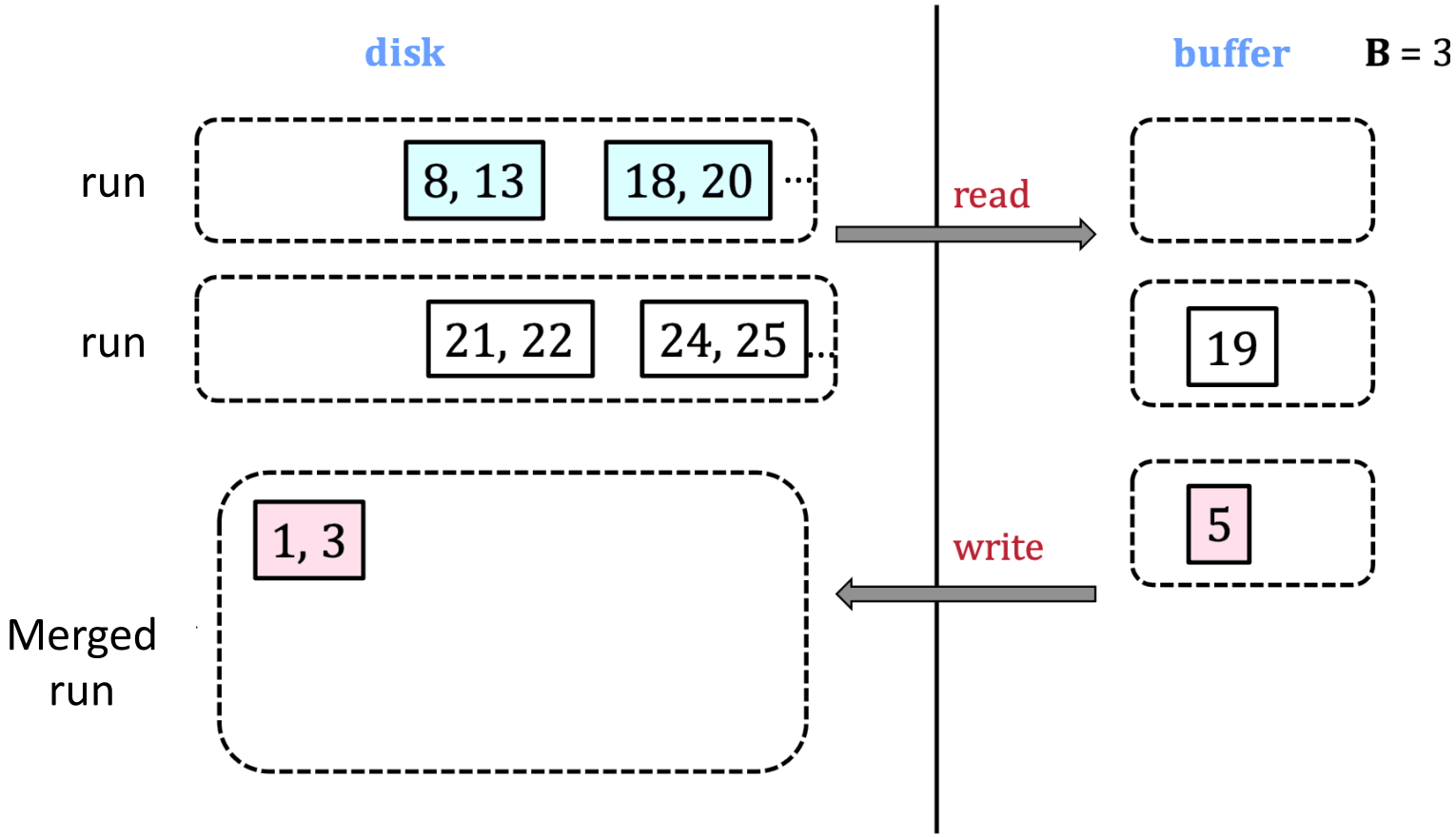
Esempio

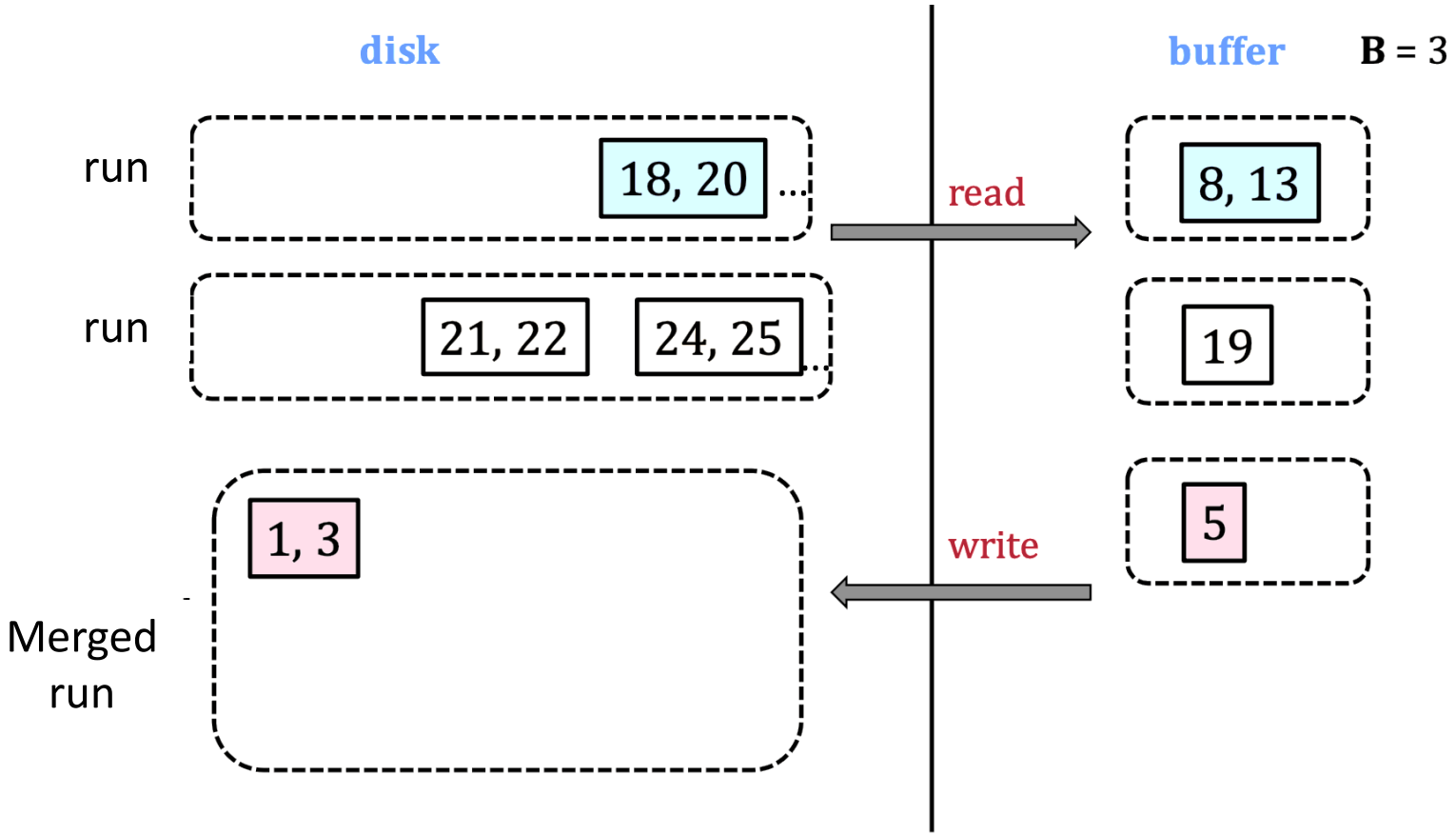

disk
buffer
B = 3
run
1, 5
8, 13
18, 20
...
run
3, 19
21, 22
24, 25
...
read
write
Merged
run

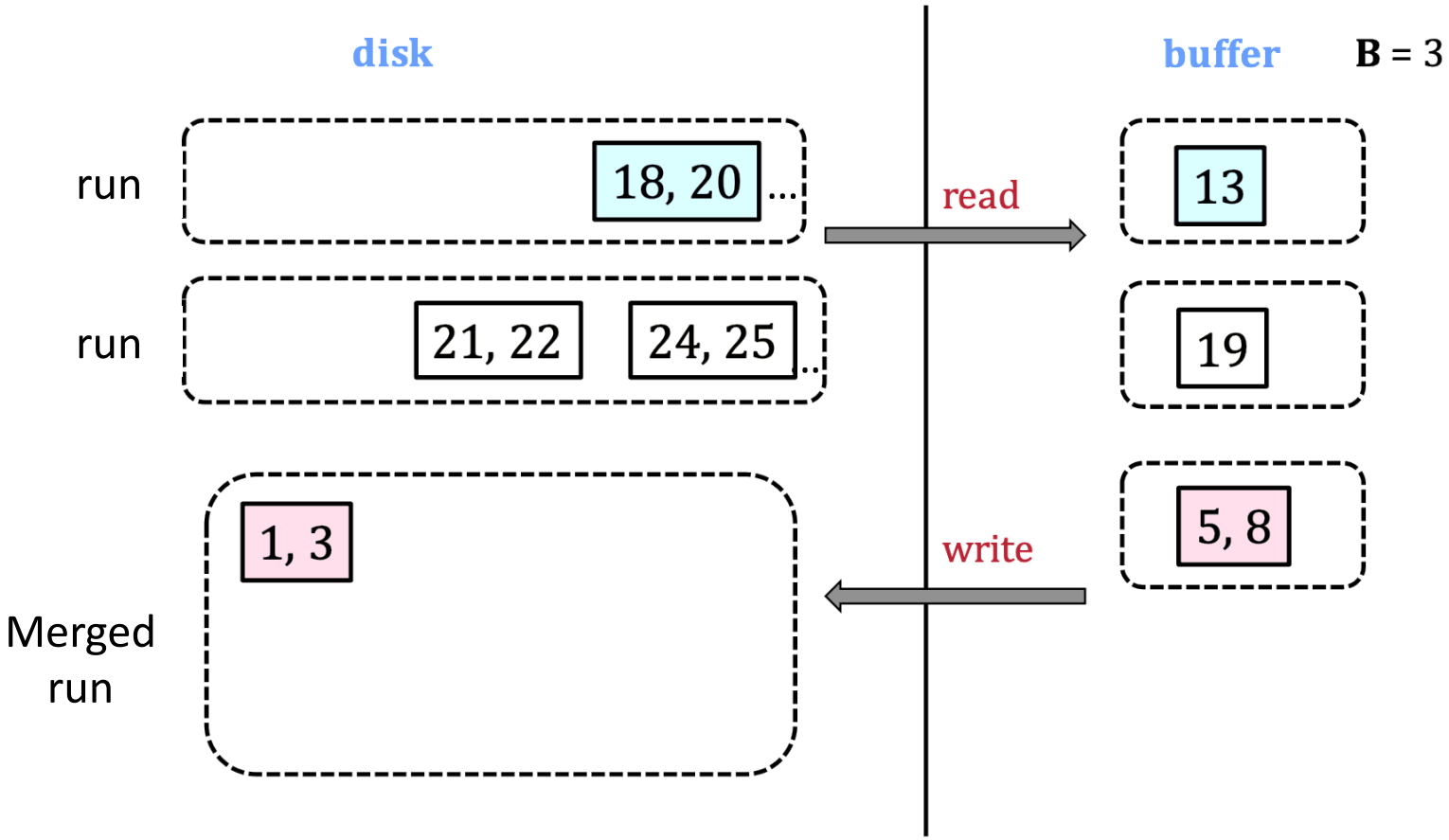

disk
buffer
B = 3
run
8, 13
18, 20
..
read
1, 5
run
21, 22
24, 25
..
3, 19
write
Merged
run

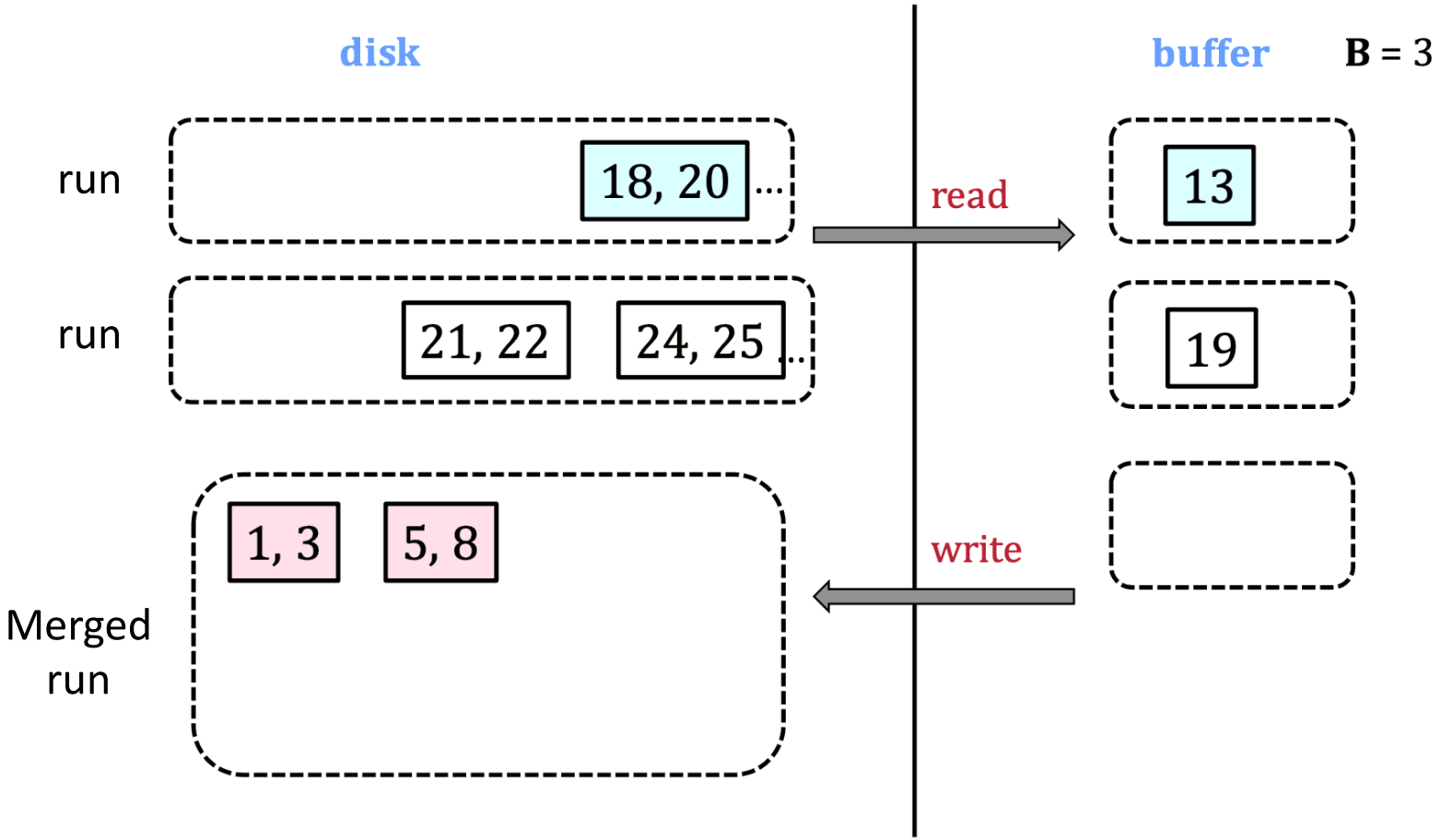

disk
buffer
B = 3
run
8, 13
18, 20
...
read
5
run
21, 22
24, 25
...
19
1, 3
write
Merged
run

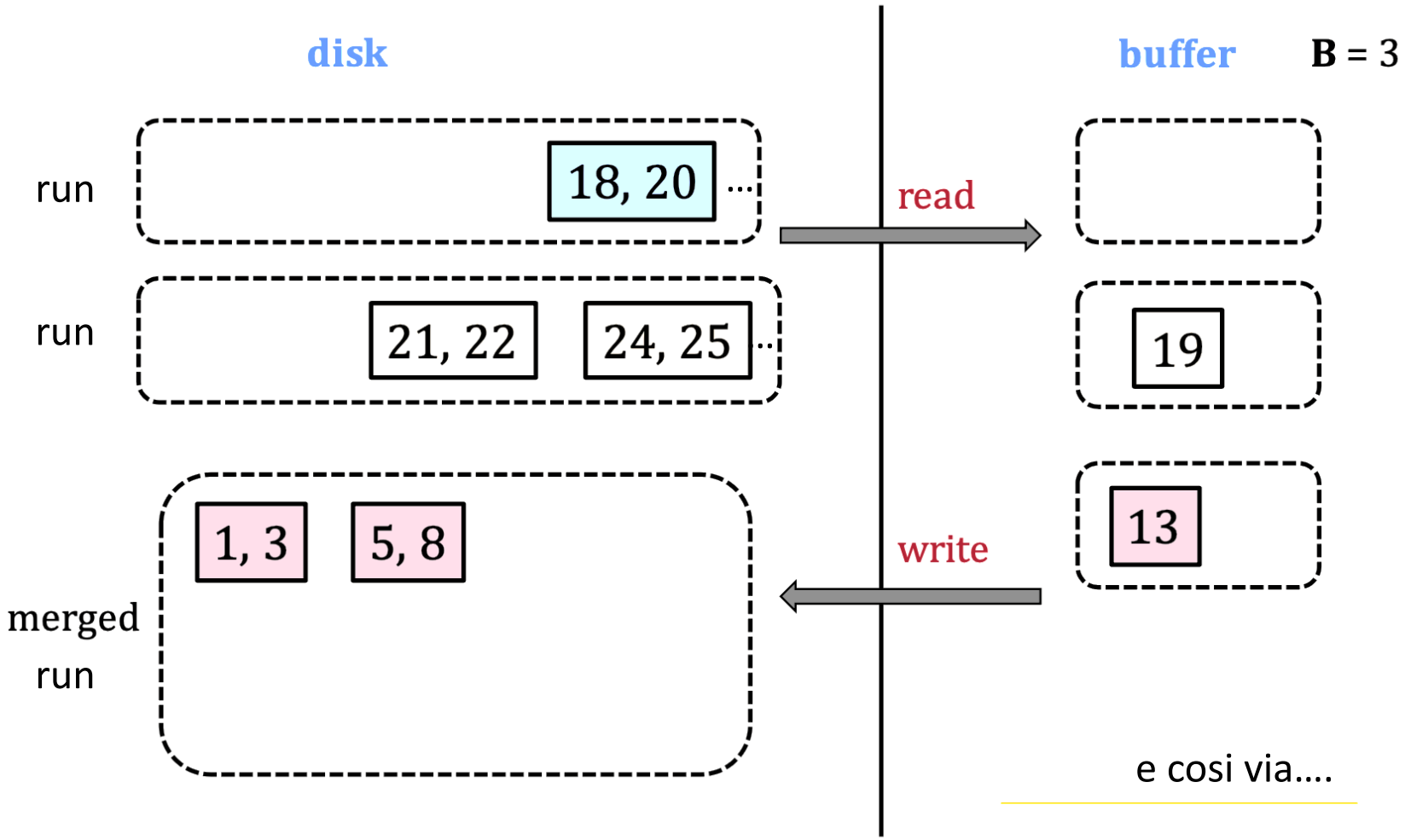

disk
buffer
B = 3
run
8, 13
18, 20
...
read
5
run
21, 22
24, 25
...
19
1, 3
write
Merged
run


disk
buffer
B = 3
run
8, 13
18, 20
read
run
21, 22
24, 25
19
5
write
1, 3
Merged run


disk
buffer
B = 3
run
18, 20
...
read
8, 13
run
21, 22
24, 25
...
19
1, 3
write
5
Merged
run


disk
buffer
B = 3
run
18, 20 ...
run
21, 22
24, 25 ...
read
13
19
write
5, 8
1, 3
Merged run

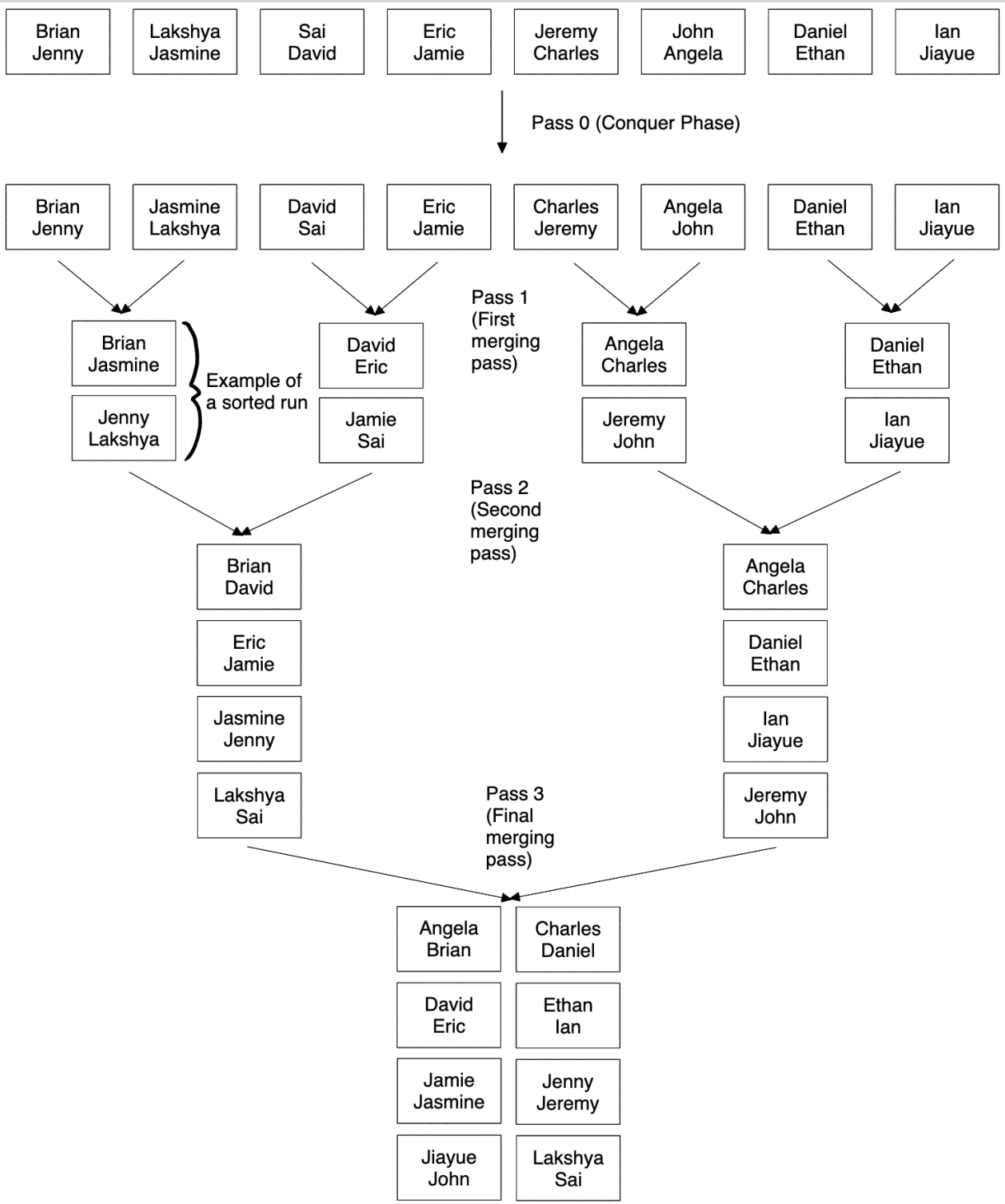

disk
buffer
B = 3
run
18, 20 ...
read
13
run
21, 22
24, 25 ...
19
1, 3
5, 8
write
Merged run


disk
buffer
B = 3
run
18, 20
...
read
run
21, 22
24, 25
..
19
1, 3
5, 8
13
write
merged
run
e cosi via....

2-Way Merge Sort:

run = 1 blocco del file

- Passo 0: Ogni blocco del file (run) è caricato nel buffer, ordinato, e riscritto in memoria secondaria
  - Per questo passo è richiesto almeno un blocco nel buffer
- Passo 1: Sono caricati due blocchi (run), riordinati e riscritti
  - Per questo passo sono richiesti almeno 3 blocchi nel buffer: 1 blocco per ogni run, e uno per memorizzare l'output ordinato
- Passo n: i run hanno n. blocchi >2. In memoria è caricato solo il primo blocco dei due run, i record vengono confrontati e memorizzati nel buffer d'output.
- Si ripete Passo n finchè tutto il file è ordinato



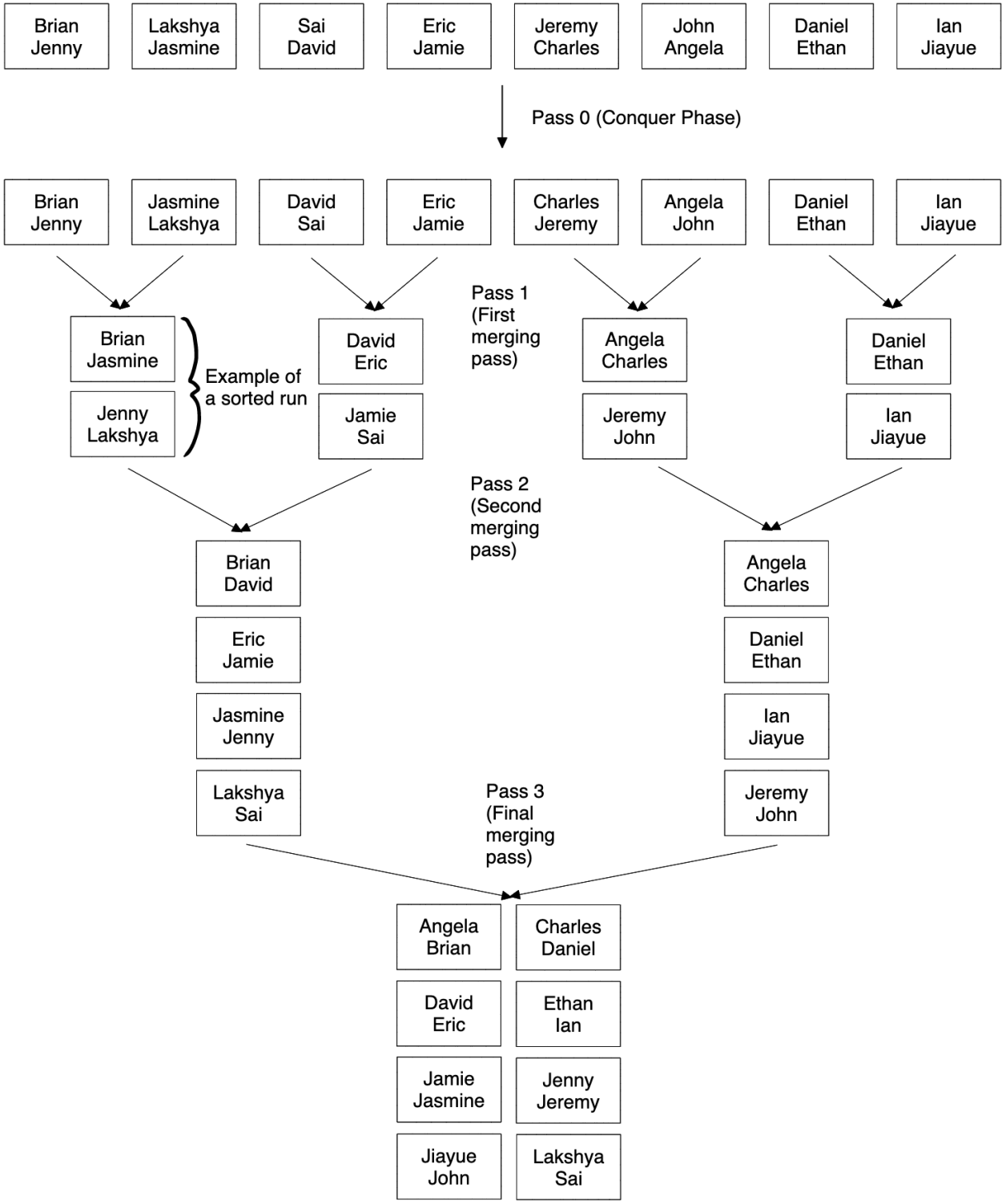
2-Way Merge Sort: costo I/O

- Dato un file di N blocchi ogni passo richiede:
  - N letture ed N scritture: 2N I/O
- Costo I/O= 2N * #passi
- Con N blocchi, 2-way Mergesort richiede
  - 1 passo per il Sort iniziale (passo 0)
  - $\lceil \log_2 N \rceil$ passi di merge
- Costo totale
  - 2N $*$ $(\lceil \log_2 N \rceil + 1)$ accessi

- Es: File di 1.000 blocchi richiede:

  $2*1000(\lceil \log_2 1000 \rceil + 1) =$

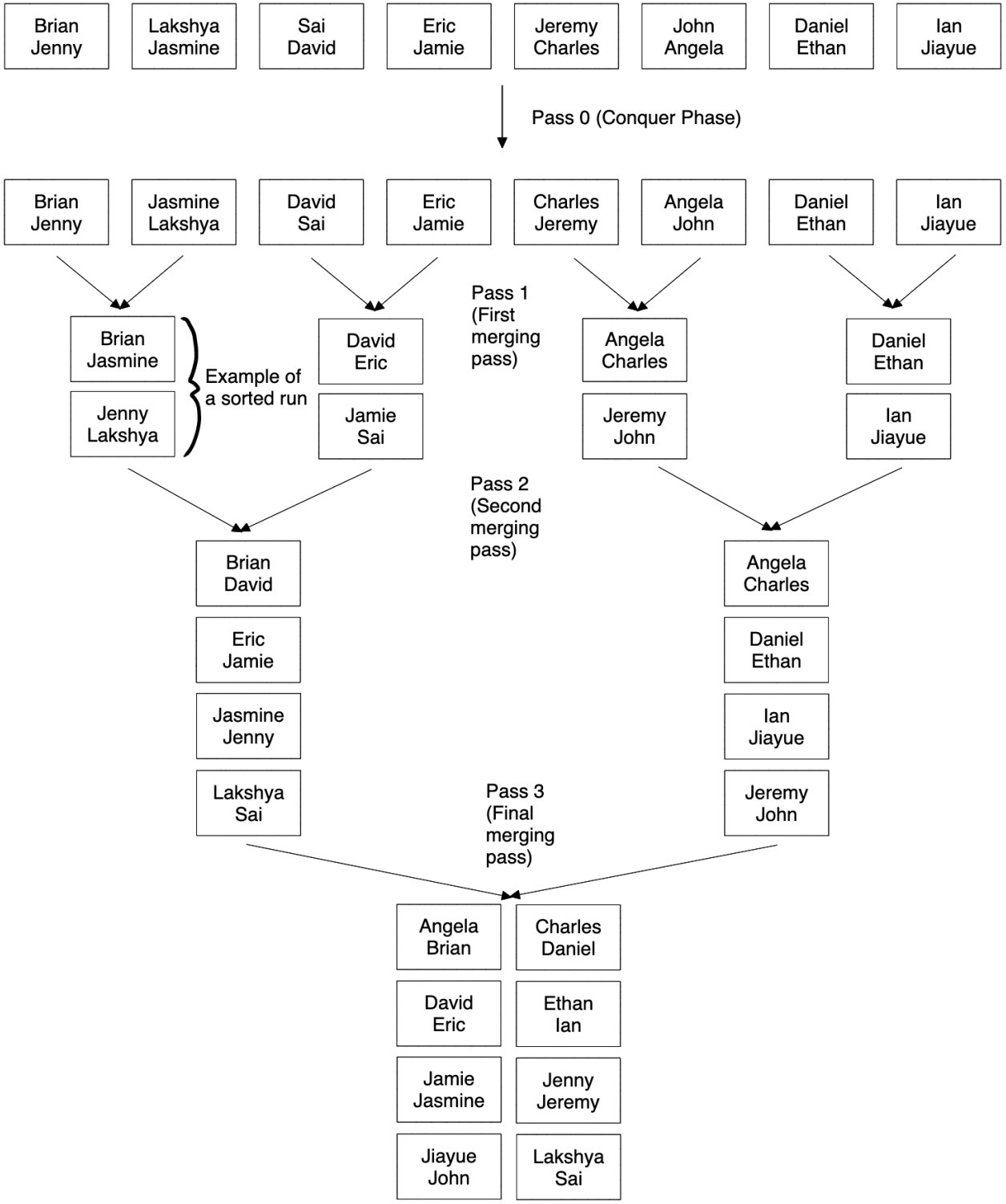
  2.000 *(10+1)= 22.100



2-Way Merge Sort: costo I/O

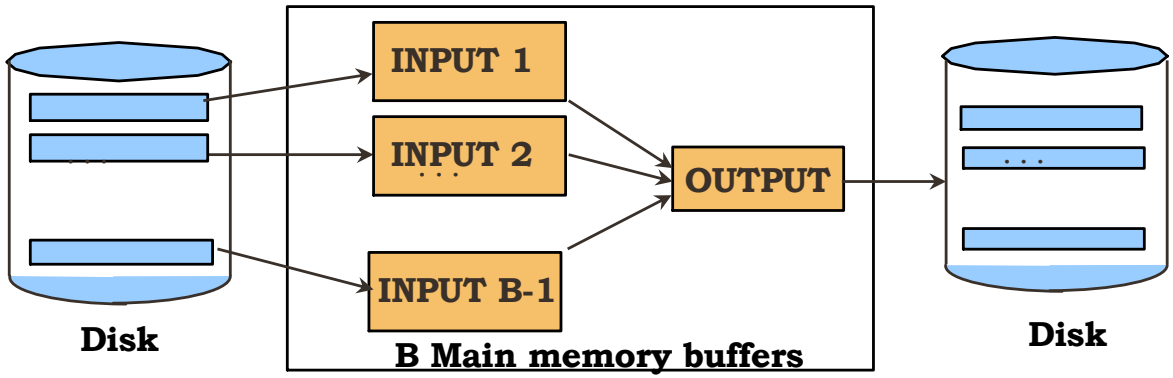
- Dato un file di N blocchi ogni passo richiede:
  - N letture ed N scritture: 2N I/O
- Costo I/O= 2N * #passi
- Con N blocchi, 2-way Mergesort richiede
  - 1 passo per il Sort iniziale (passo 0)
  - $\lceil \log_2 N \rceil$ passi di merge
- Costo totale
  - $2N * (\lceil \log_2 N \rceil + 1)$ accessi

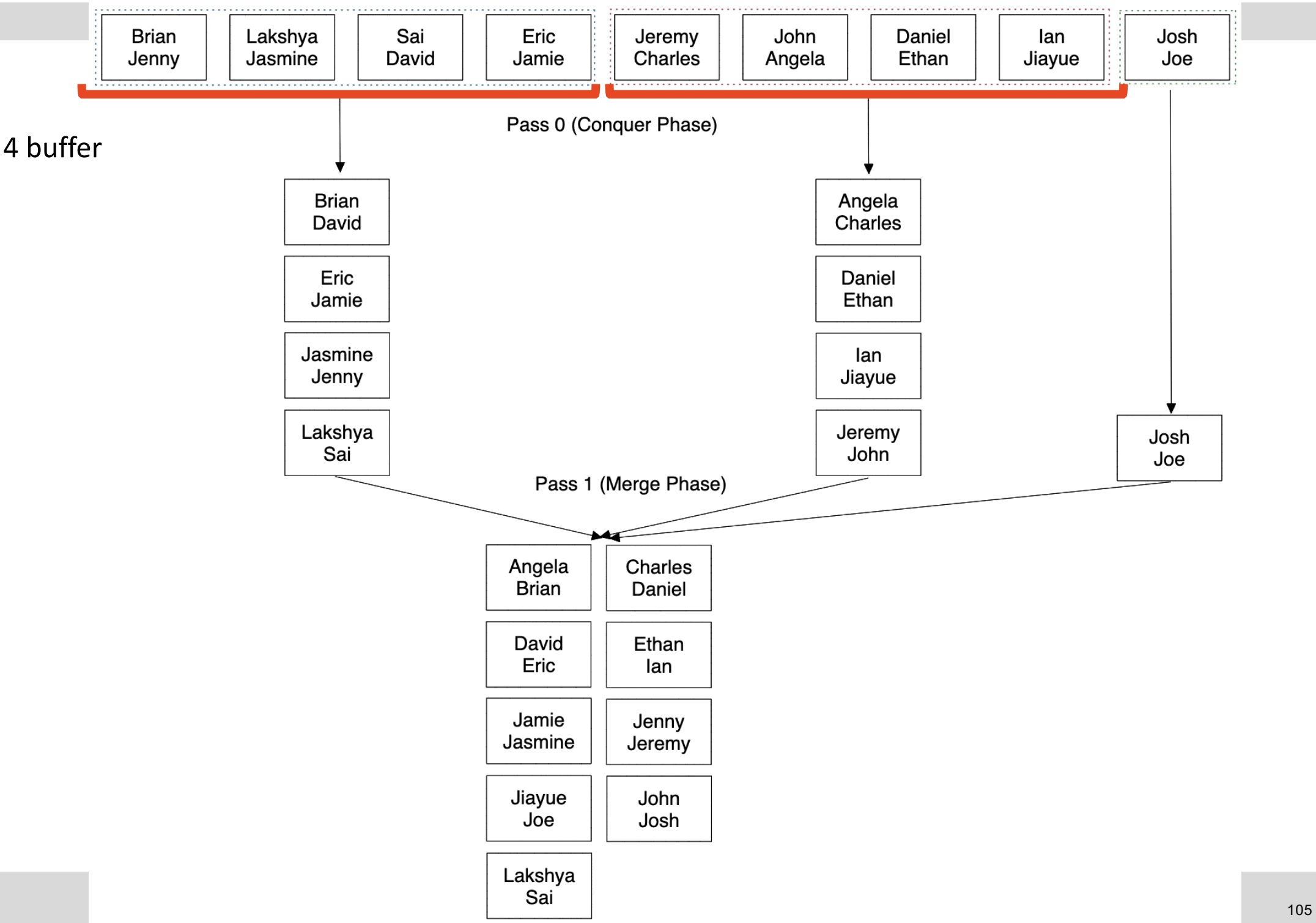
- Idea: si può ridurre il numero di passi (e accessi I/0) aumentando i numero di blocchi del buffer?



Merge Sort esterno generico:

- Supponiamo di avere a disposizione B>3 blocchi nel Buffer e un file di N blocchi da riordinare
  - Passo 0: posso utilizzare tutti i B blocchi del buffer per riordinare run più grandi:
    - run di B blocchi
    - ottengo $\lceil N/B \rceil$ run, ognuno di B blocchi (l'ultimo può avere dimensione minore)
  - Passo n: utilizzo *B-1* buffer come input per i primi blocchi dei run ed 1 buffer come output




Brian
Jenny
Lakshya
Jasmine
Sai
David
Eric
Jamie
Jeremy
Charles
John
Angela
Daniel
Ethan
Ian
Jiayue
Josh
Joe
4 buffer
Pass 0 (Conquer Phase)
Brian
David
Eric
Jamie
Jasmine
Jenny
Lakshya
Sai
Angela
Charles
Daniel
Ethan
Ian
Jiayue
Jeremy
John
Josh
Joe
Pass 1 (Merge Phase)
Angela
Brian
Charles
Daniel
David
Eric
Ethan
Ian
Jamie
Jasmine
Jenny
Jeremy
Jiayue
Joe
John
Josh
Lakshya
Sai

Merge Sort esterno generico: costo

- Numero di passi: $1 + \lceil \log_{B-1} \lceil N / B \rceil \rceil$
- Costo = 2N * $1 + \lceil \log_{B-1} \lceil N / B \rceil \rceil$

- Esempio: quanti e quali passi devo eseguire per riordinare un file di 108 blocchi con external merge sort con 5 buffer?

Merge Sort esterno generico: costo

- Numero di passi: $1 + \lceil \log_{B-1} \lceil N / B \rceil \rceil$
- Costo = 2N * $1 + \lceil \log_{B-1} \lceil N / B \rceil \rceil$

- Esempio: quanti e quali passi devo eseguire per riordinare un file di 108 blocchi con external merge sort con 5 buffer?
  - Pass 0: $\lceil 108 / 5 \rceil$ = 22 run ordinati, 21 di 5 blocchi ognuno, e l'ultimo di solo 3 blocchi
  - Pass 1: $\lceil 22 / 4 \rceil$ = 6 run ordinati di 20 blocchi ognuno (l'ultimo solo 8 blocchi)
  - Pass 2: 2 run ordinati, uno di 80 blocchi ed uno di 28 blocchi
  - Pass 3: 108 blocchi ordinati

Numero di passi

N	B=3	B=5	B=9	B=17	B=129	B=257
100	7	4	3	2	1	1
1,000	10	5	4	3	2	2
10,000	13	7	5	4	2	2
100,000	17	9	6	5	3	3
1,000,000	20	10	7	5	3	3
10,000,000	23	12	8	6	4	3
100,000,000	26	14	9	7	4	4
1,000,000,000	30	15	10	8	5	4