Concetti di MapReduce

È un framework software brevettato e introdotto da Google per supportare la computazione distribuita su grandi quantità di dati in cluster di computer. Il framework è ispirato alle funzioni map e reduce usate nella programmazione funzionale, sebbene il loro scopo nel framework MapReduce non è lo stesso che nella forma originale [1].

Il framework MapReduce è composto da diverse funzioni per ogni step:

1. Input reader
2. Map Function
3. Partion funcion
4. Compare Function
5. Reduce Function
6. Output writer

[1] https://it.wikipedia.org/wiki/MapReduce

Concetti di MapReduce

L'Input Reader legge i dati dalla memoria di massa, li divide in sottogruppi generando una coppia chiave, valore => $<K^{in}, V^{in}>$ da associare ad ogni sottogruppo.

Si dividono le macchine in N cluster di cui 1 Master (M) e N-1 Slave (S), che ricevono i task assegnati M

Un generico S legge la coppia $<K^{in}, V^{in}>$ ed eseguirà la funzione di Map (MapJob-function) definita dall'utente che genera 0 o più coppie chiave, valore in uscita ($<K_{map}, V_{map}>$).

La partion function memorizza e partiziona le coppie $<K_{map}, V_{map}>$ in R Sezioni. Gli indirizzi delle sezioni partizione vengono poi passate a M che è responsabile di girare le locazioni agli S che processano funzioni di riduzione.

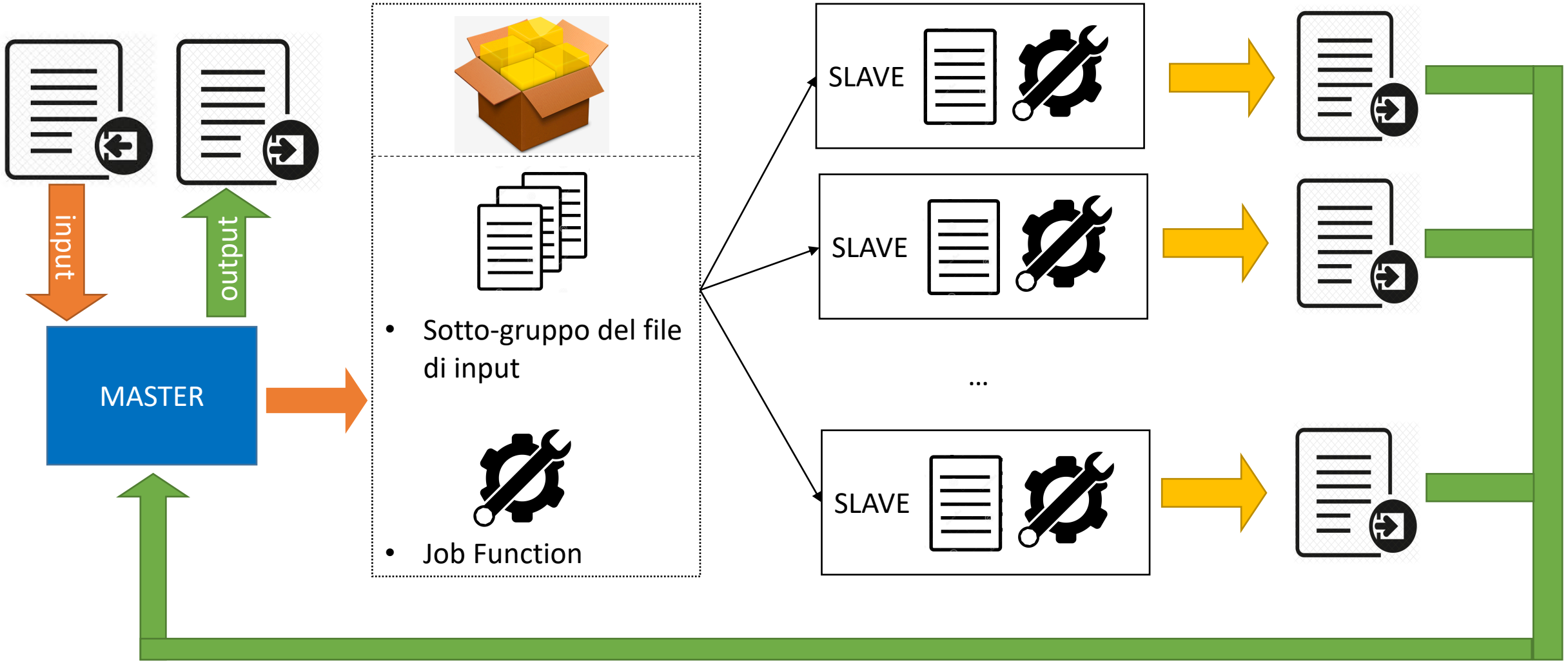
Tra S di Map e S di Reduce vengono riordinate tutte le coppie in modo da trovare quelle che puntano alla stessa chiave. Successivamente la funzione di comparazione effettua un'operazione di unione (merge).

Per ogni chiave viene associato S di Reduce che prende i valori con la stessa chiave dalle coppie $<K_{map}, V_{map}>$ e li passa alla funzione di Reduce (ReduceJob-Function) definita dall'utente che genera o più coppie chiave, valore in uscita ($<K_{reduce}, V_{reduce}>$).

[1] https://it.wikipedia.org/wiki/MapReduce

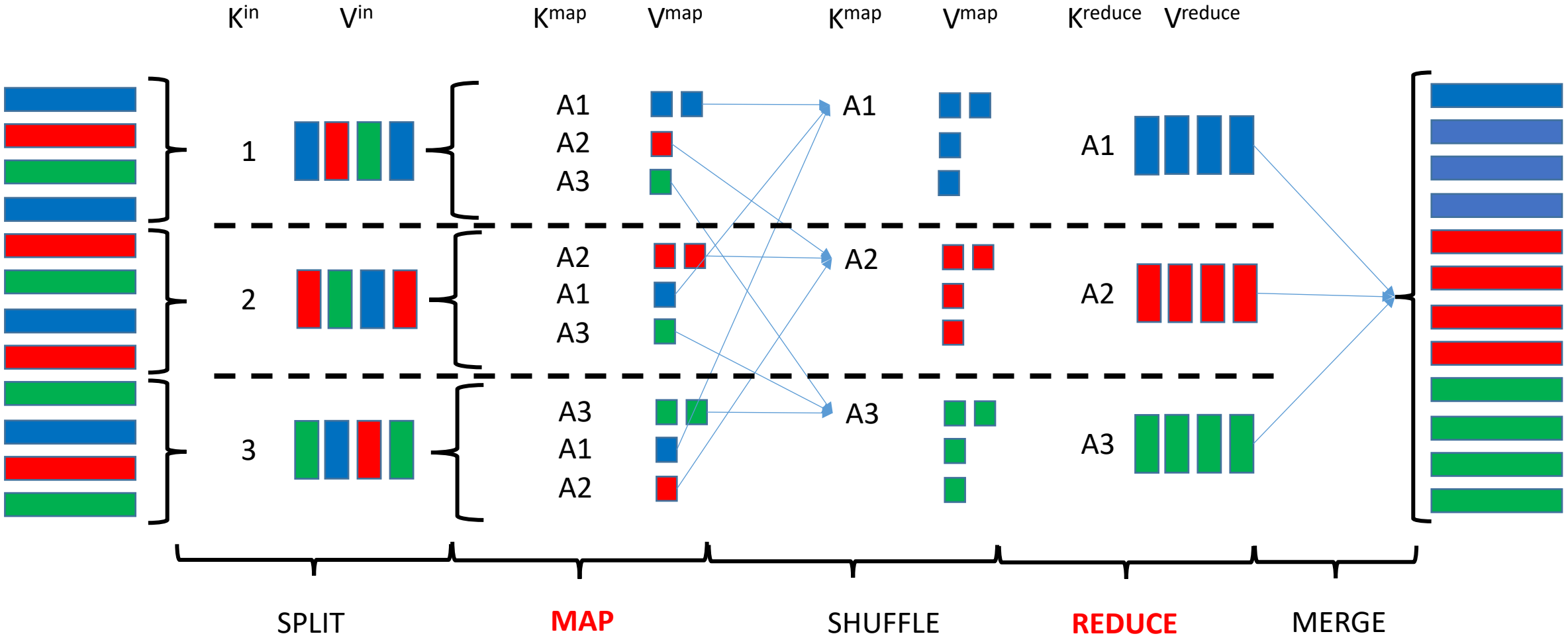
APPLICAZIONE DEL MAP REDUCE

Schema MapReduce



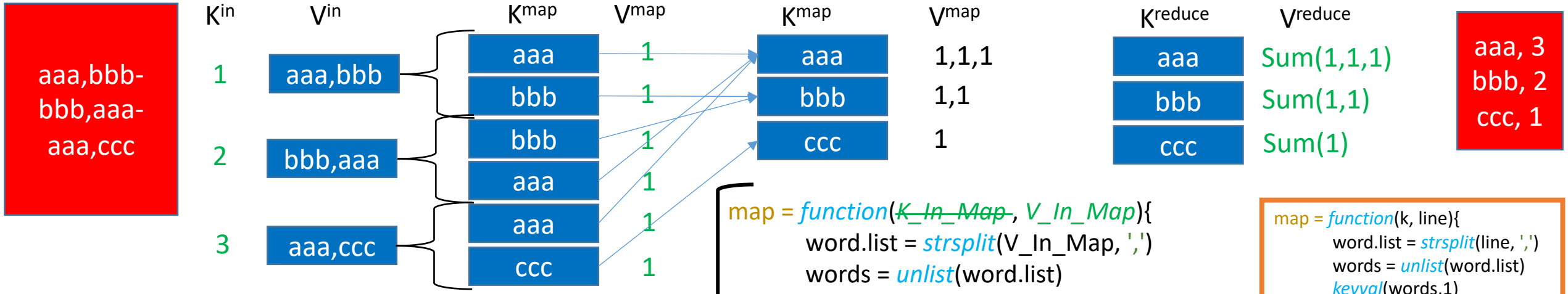
APPLICAZIONE DEL MAP REDUCE

Schema MapReduce



APPLICAZIONE DEL MAP REDUCE

Schema MapReduce Sequence count



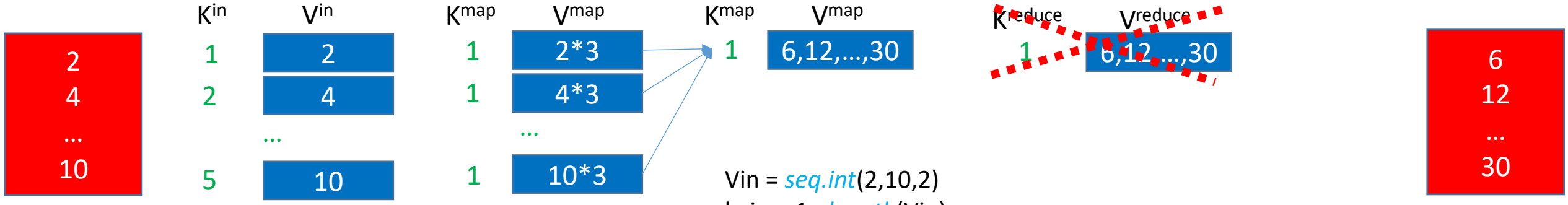

```
Input = "aaa,bbb-bbb,aaa-aaa,ccc";
lines = strsplit(Input, '-');
line = unlist(lines)
~~k = 1 : length(line);~~
Output = MapReduce(
    input = ( ~~K_In_Map = k~~, V_In_Map = line )
    input : V_In_Map = to.dfs(line)
    ...
)
print(Output)
Print:: {«aaa,3» , «bbb,2» , … , «ccc,1» }
```

```
map = function(~~K_In_Map~~, V_In_Map){
        word.list = strsplit(V_In_Map, ',')
        words = unlist(word.list)
        K_Out_Map = words
        V_Out_Map = 1
        keyval(K_Out_Map, V_Out_Map )
}
# reduce.input : K_Out_map , V_Out_map
reduce = function(K_In_Reduce, V_In_Reduce){
        K_Out_Reduce = K_In_Reduce
        V_Out_Reduce = sum(V_In_Reduce)
        keyval(K_Out_Reduce, V_Out_Reduce )
}
```

```
map = function(k, line){
        word.list = strsplit(line, ',')
        words = unlist(word.list)
        keyval(words,1)
}
# reduce.input : words, v
reduce = function(w,val){
        keyval(w, sum(val))
}
```

APPLICAZIONE DEL MAP REDUCE

Schema MapReduce Cicli Iterativi (1)




```
Vin =  [ 2 , 4 , … ,  10 ];
Vout = [];
For i = 1 to length(Vin){
    Vout[i] = Vin[i] * 3;
}
```

```
Vout[1] = Vin[1] * 3;
Vout[2] = Vin[2] * 3;
Vout[3] = Vin[3] * 3;
…
```

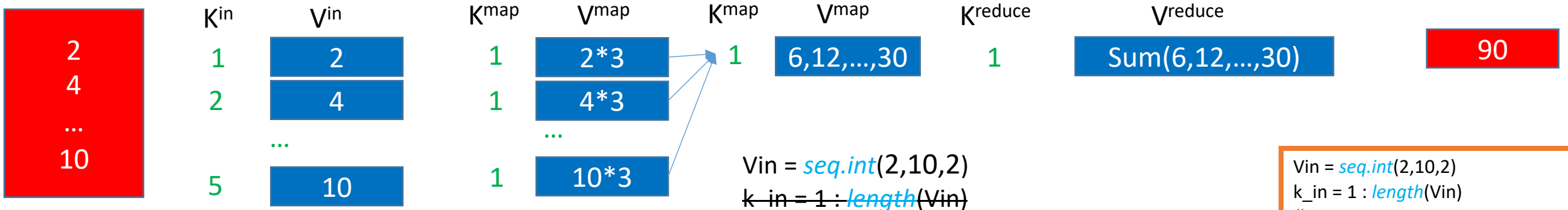
	K	V	TASK
Input			
Map			
Reduce			

```
Vin = seq.int(2,10,2)
k_in = 1 : length(Vin)
# V_In_Map = to.dfs(Vin)
input = ( K_In_Map = Vin ,  V_In_Map = k_in )
map = function(K_In_Map , V_In_Map){
        K_Out_Map = 1
        V_Out_Map = V_In_Map * 3
        keyval(K_Out_Map, V_Out_Map )
}
#reduce.input : K_Out_Map , V_Out_Map
reduce = function(K_in_reduce,V_in_reduce){
        K_out_reduce = K_in_reduce
        V_out_reduce = V_in_reduce
        keyval(K_out_map, V_out_map )
}
```

```
Vin = seq.int(2,10,2)
k_in = 1 : length(Vin)
#----------------------------------------
map = function(k_in, Vin){
        keyval(1, Vin * 3)
}
# reduce.input : K_Map, V_Map
reduce = function(k,v){
        keyval(k,v)
}
```

APPLICAZIONE DEL MAP REDUCE

Schema MapReduce Cicli Iterativi (2)




```
Vin =  { 1, 2, … ,  10 };
Somma = 0;
for i = 1 to length(Vin){
    Somma = Somma  + Vin[i] * 3;
}
```

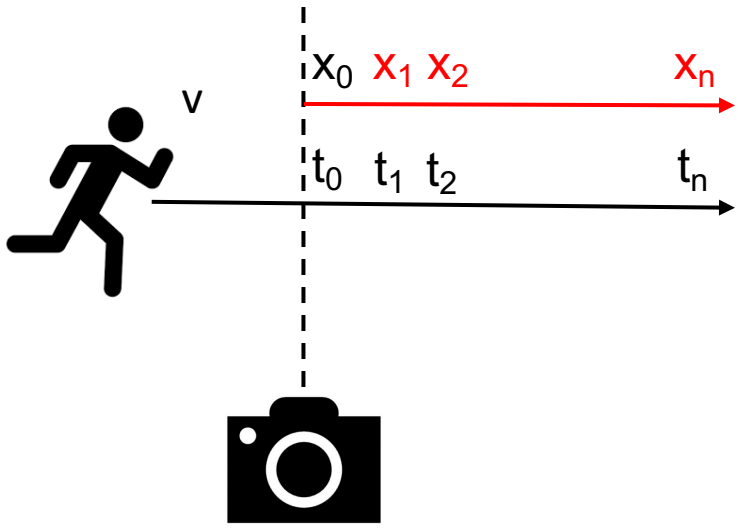
	K	V	TASK
Input			
Map			
Reduce			

```
Vin = seq.int(2,10,2)
k_in = 1 : length(Vin)
# V_In_Map = to.dfs(Vin)
input = ( K_In_Map = k_in,  V_In_Map = Vin)
map = function(K_In_Map , V_In_Map){
        K_Out_Map = 1
        V_Out_Map = V_In_Map * 3
        keyval(K_Out_Map, V_Out_Map )
}
# reduce.input : K_Out_Map , V_Out_Map
reduce = function(K_In_Reduce,V_In_Reduce){
        K_Out_Reduce = K_In_Reduce
        V_Out_Reduce = sum(V_In_Reduce)
        keyval(K_Out_Map, V_Out_Map )
}
```

```
Vin = seq.int(2,10,2)
k_in = 1 : length(Vin)
#---------------------------------------
map = function(k_in, Vin){
        keyval(1, Vin * 3)
}
# reduce.input : K_Map, V_Map
reduce = function(k,v){
        keyval(k, sum(v))
}
```

Esempio MapReduce

Un corpo si muove ad una velocità di 10 m/s una macchina fotografica posizionata ad un punto fisso scatta delle foto al corpo in movimento con una frequenza pari a = 0,5 Hz. Supponendo che la fotocamera inizia a scattare le foto quando il corpo passa in linea ad essa per una durata di 30s Calcolare le posizioni del corpo ad ogni scatto.

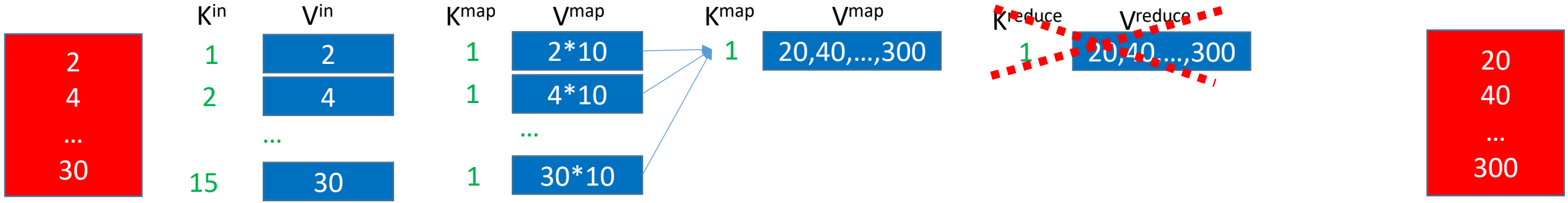


Soluzione:
la formula della velocità è: $v = (x_1-x_0)/(t_1-t_0)$
da cui si ricava la legge oraria del moto rettilineo uniforme, il quale mette in relazione la posizione con la velocità: $x_1 = v*(t_1 - t_0) + x_0$. Il problema ci indica che a $t_0=0$ ci troviamo a posizione iniziale nulla: **$x_1 = v * t_1$** (perché $t_0=0$, $x_0=0$). Una frequenza di campionamento è l' inverso del tempo di campionamento ovvero l'intervallo temporale tra un campione e l' altro per cui ci saranno 30 * 0,5 = 15 campioni, quindi mi creo un vettore temporale di 15 elementi partendo da 2 fino a 30 inclusi $t_1 = \{2,4,\ldots,30\}$;
Infine per calcolare la posizione moltiplico la velocità v = 10 per ogni elemento del vettore t_n

Applico un paradigma MapReduce per calcolarmi il vettore delle posizioni x_n

APPLICAZIONE DEL MAP REDUCE

Esempio MapReduce




```
Fc = 0.5
tn= seq.int(2,30,1/Fc)
k_in = 1 : length(t1)
v = 10
# V_in_map = to.dfs(tn)
input = ( K_In_Map = k_in ,  V_In_Map = tn)
map = function(K_In_Map , V_In_Map){
        K_Out_Map = 1
        V_Out_Map = V_In_Map * v
        keyval(K_Out_Map, V_Out_Map )
}
```

```
Fc = 0.5
tn = seq.int(2,30,1/Fc)
k_in = 1 : length(tn)
v = 10
#V_in_map = to.dfs(tn)
map = function(k_in, tn){
        keyval(1, tn * v)
}
# reduce.input : K_Map, V_Map
reduce = function(k,v){
        keyval(k,v)
}
```

	K	V	TASK
Input	1, 2, 3, 4, 5, 6, 7, 8, 9, …, 15	2, 4, 6, 8, 10, 12, 14, …, 30	Fc = 0.5 tn = seq.int(2,30,1/Fc) k_in = 1 : length(tn) v = 10
Map	1, 1, …, 1	20, 40, …, 300	keyval(1, tn * v)
Reduce	XXX	XXX	XXX