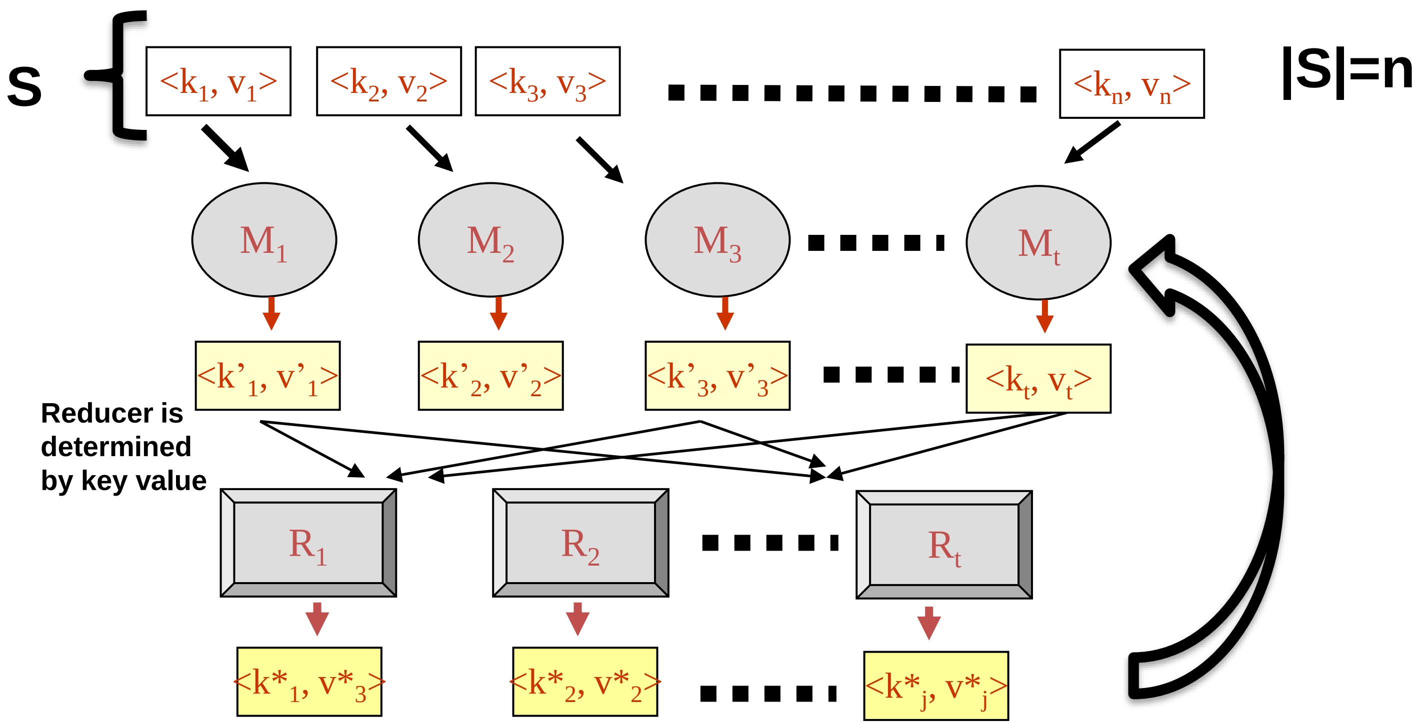
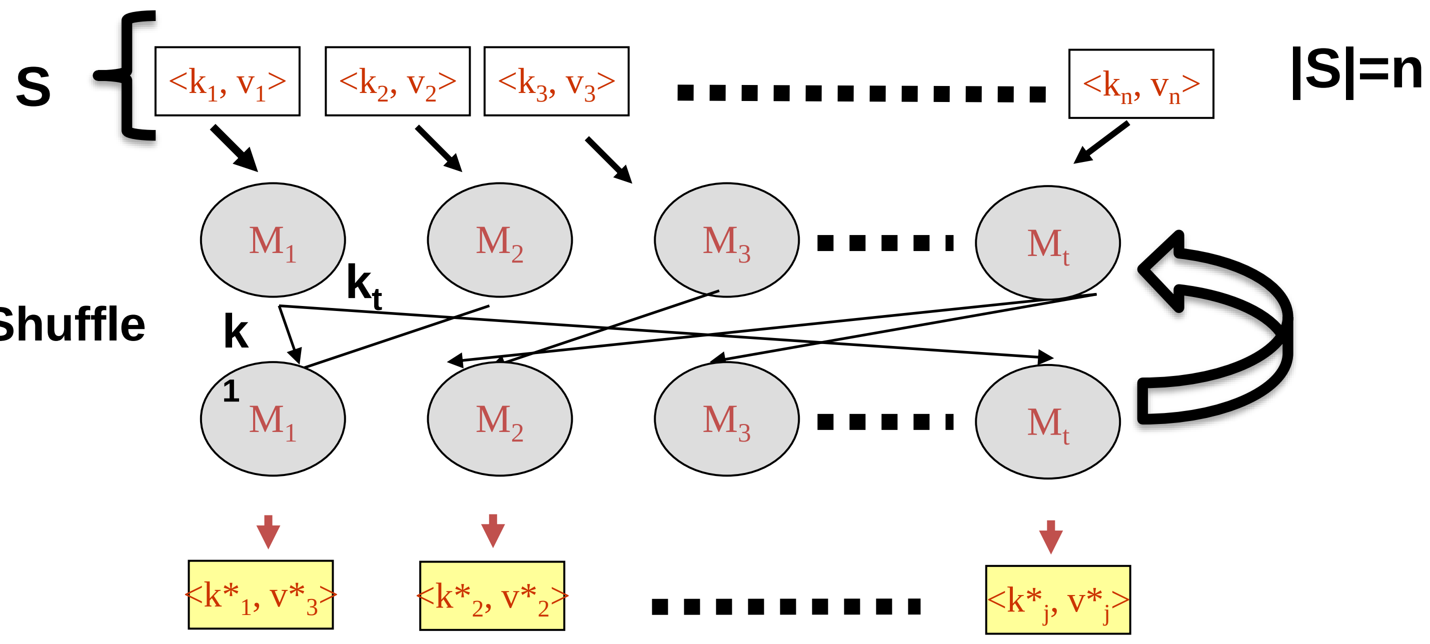
Minimal MapReduce Algorithms

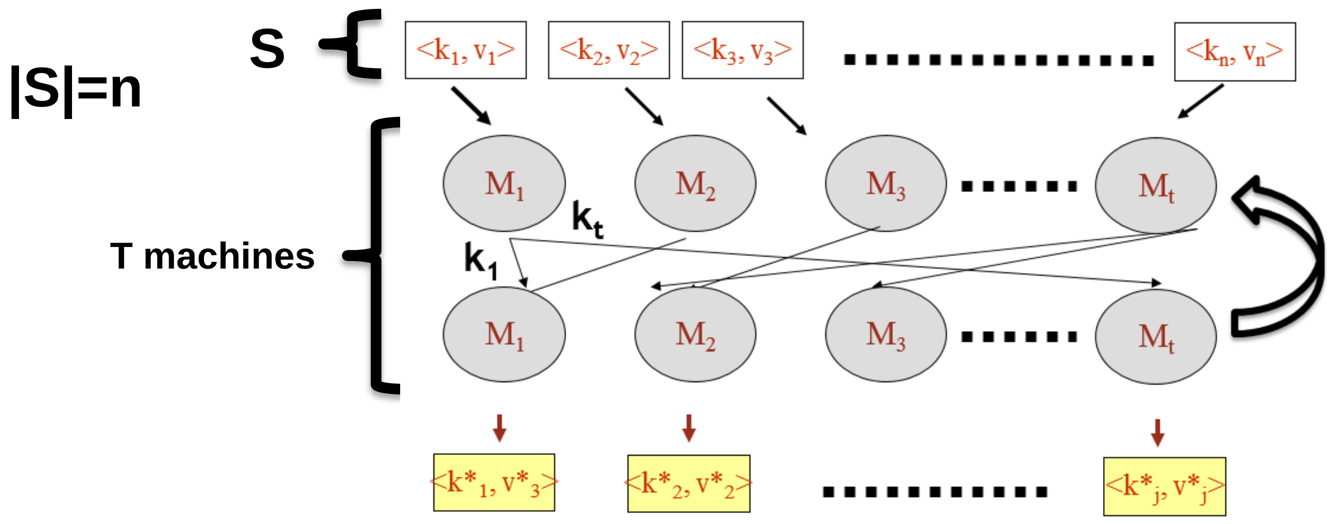
MapReduce Simplified view



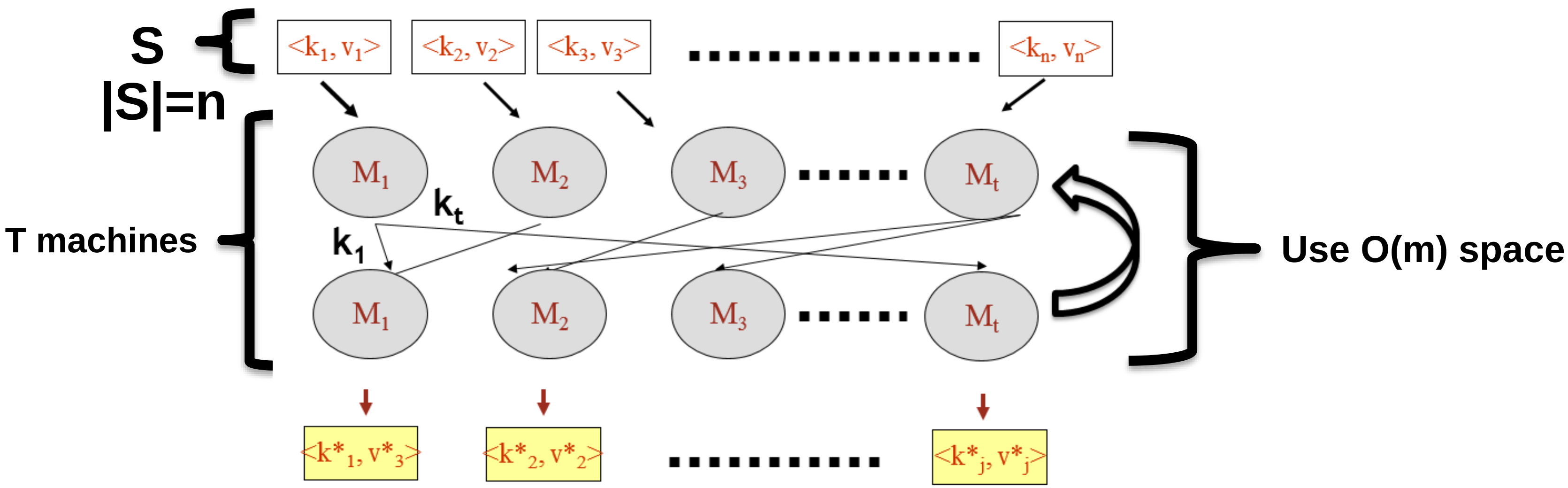
MapReduce Simplified view



Minimal MapReduce



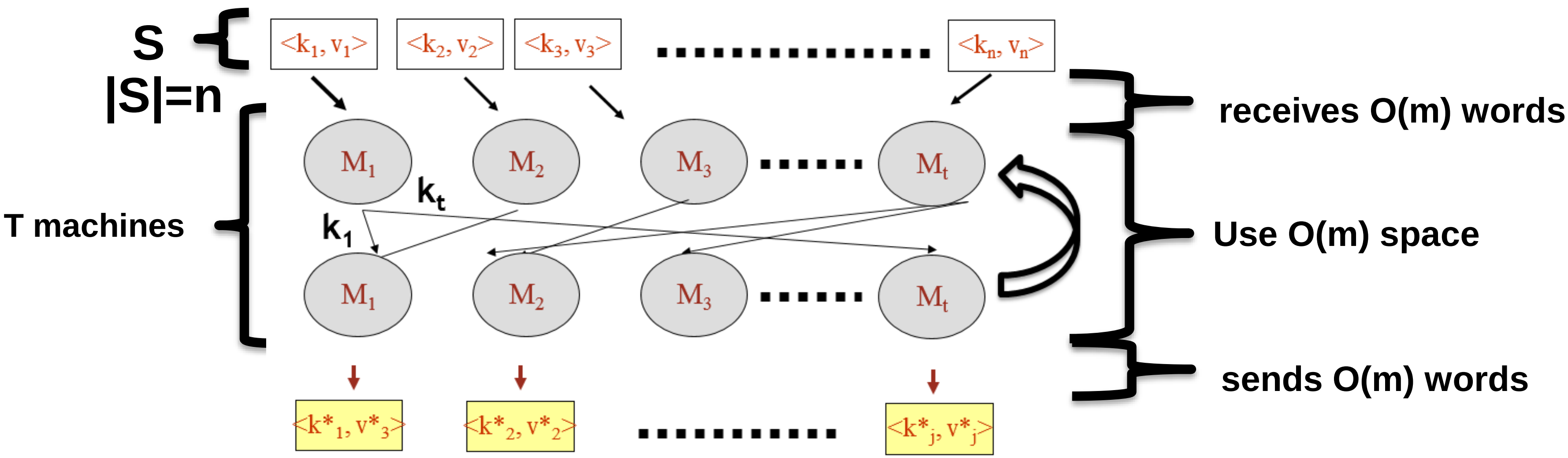
Minimal MapReduce



Denote m=n/t - the number of objects per machine when s is evenly distributed

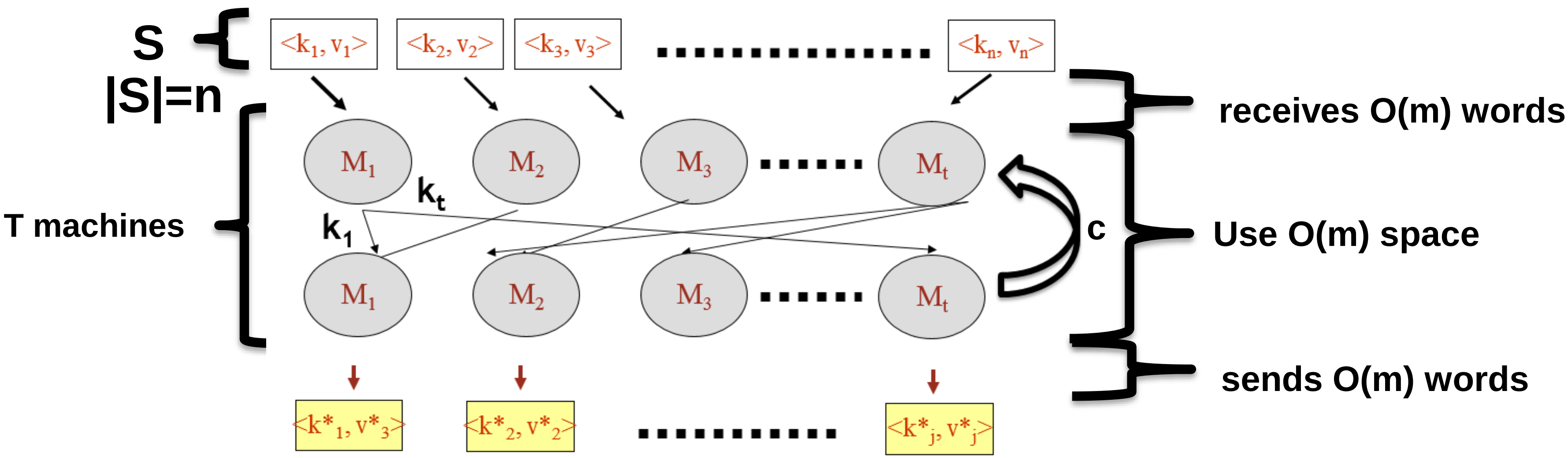
1. Minimum footprint: at all times – each machine uses only O(m) space

Minimal MapReduce



Denote **m=n/t** - the number of objects per machine when s is evenly distributed
1. Minimum footprint: at all times – each machine uses only o(m) space
2. Bounded net-traffic: in each round every machine sends and receives at most O(m) words

Minimal MapReduce



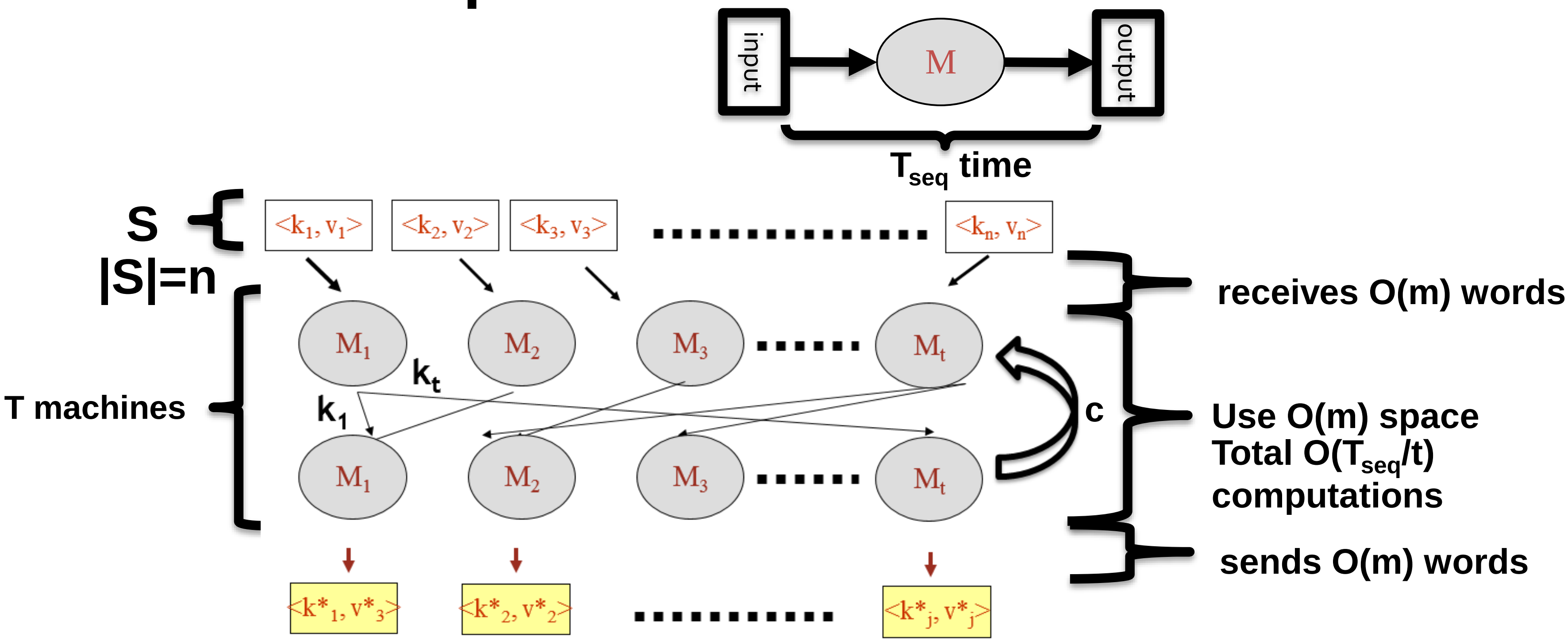
Denote **m=n/t** - the number of objects per machine when s is evenly distributed

1. Minimum footprint: at all times – each machine uses only o(m) space

2. Bounded net-traffic: in each round every machine send and receives at most O(m) words

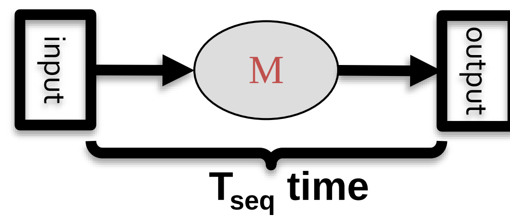
3. constant round: the algorithm must terminate after constant number of rounds

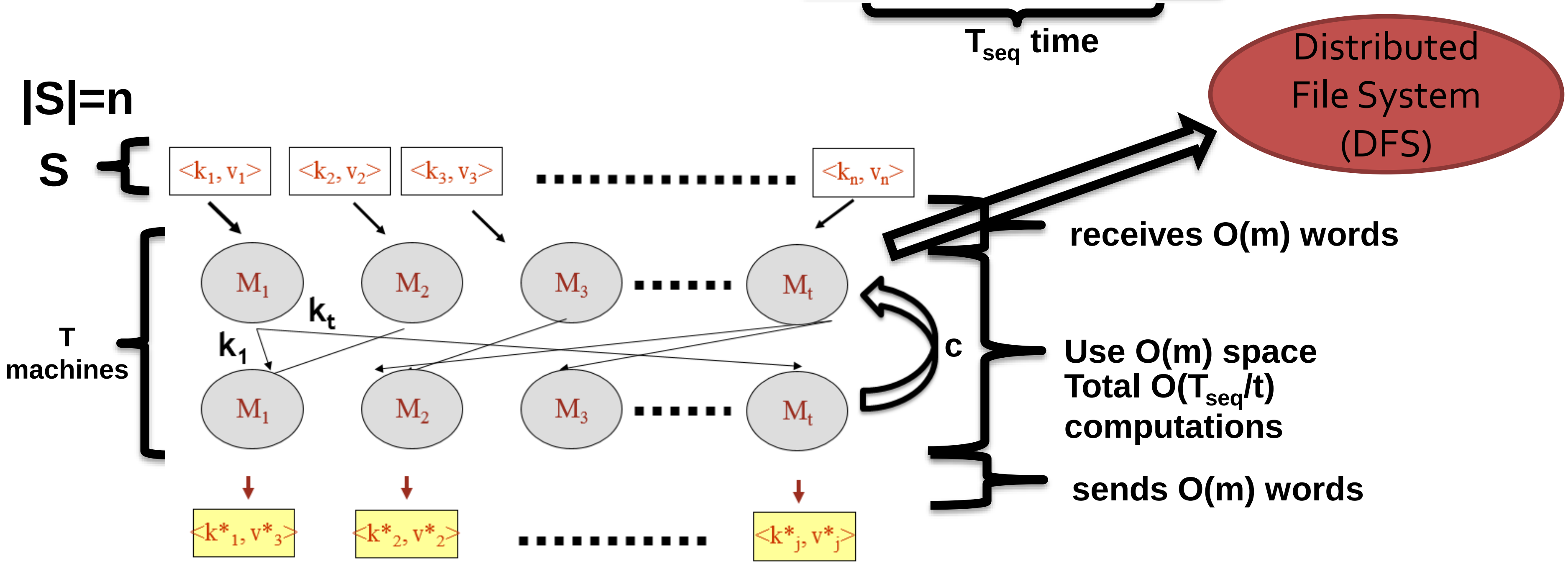
Minimal MapReduce



Denote **m=n/t** - the number of objects per machine when s is evenly distributed
1. Minimum footprint: at all times – each machine uses only o(m) space
2. Bounded net-traffic: in each round every machine send and receives at most O(m) words
3. constant round: the algorithm must terminate after constant number of rounds
4. Optimal computation: every machine performs only O(T_{seq}/t) amount of computation total when
T_{seq} = time to solve the same problem on single sequential computer
means the algorithm would get a speedup of t by using t machines in parallel

Minimal MapReduce



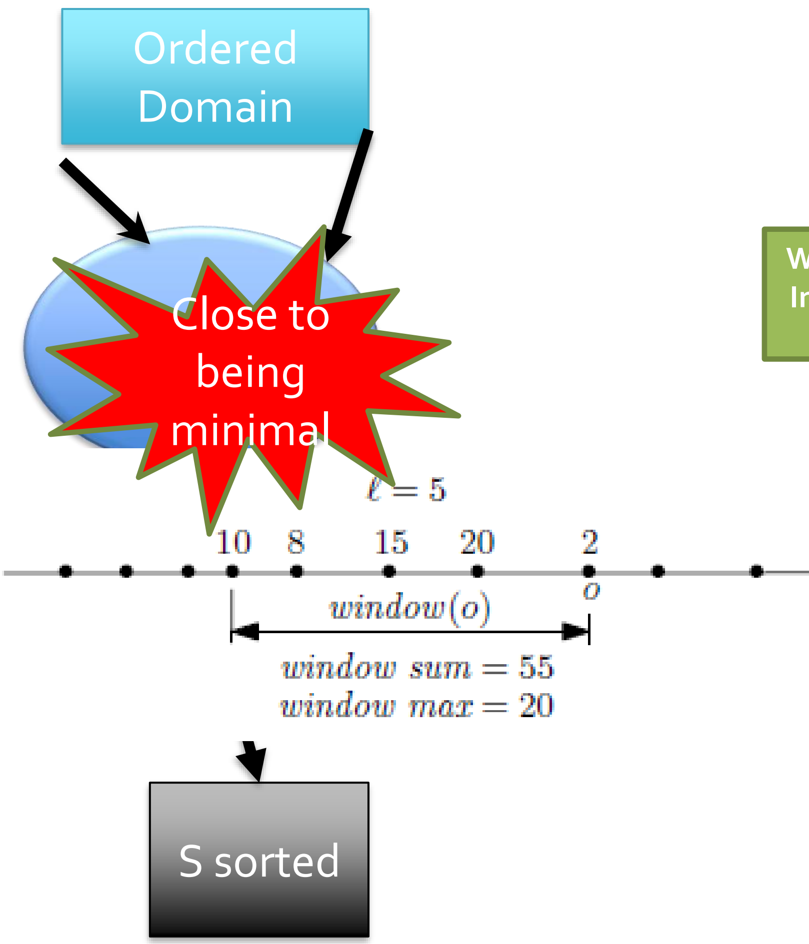


Minimal MapReduce advantages:

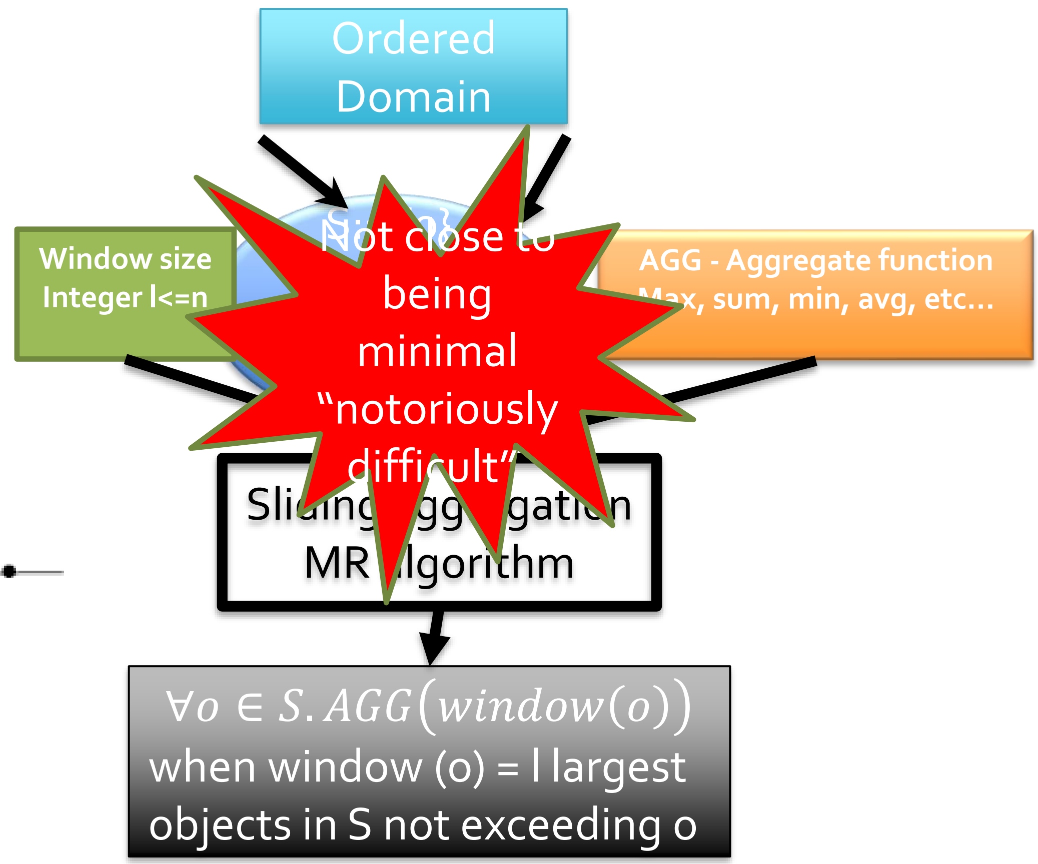
1. Each machine using O(m) in each phase = O(1/t) of S – prevent partition skew
2. Bounded net traffic – O(m) words ensures that each shuffle phase transfer at most o(n) words
3. Due to parallelization:
   The duration of each phase is ≈ the time for a machine to send and receive O(m) words.
   Good for building stateless algorithm - improve the system's robustness
4. Constant rounds = O(n) word traffic overall
5. Optimal computation – the very original motivation of MapReduce – do things faster

Not minimal MapReduce algorithms:

Terasort for sorting

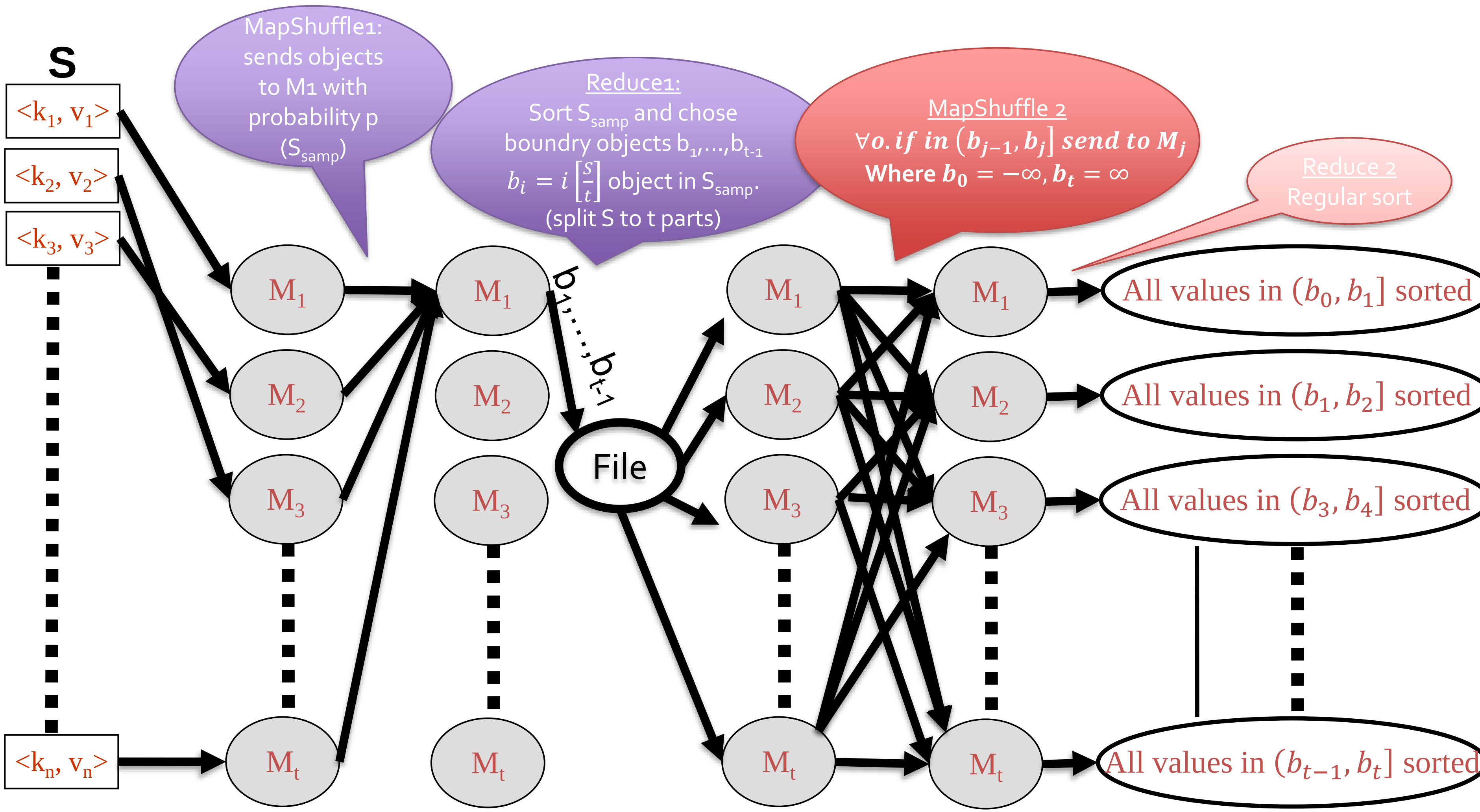


- Sliding aggregation



TeraSort ➔ minimal TeraSort

TeraSort(p)


S
<k_1, v_1>
<k_2, v_2>
<k_3, v_3>
<k_n, v_n>
MapShuffle1: sends objects to M1 with probability p (S_{samp})
Reduce1: Sort S_{samp} and chose boundry objects $b_1,\dots,b_{t-1}$ $b_i = i\left\lceil \frac{S}{t} \right\rceil$ object in S_{samp}. (split S to t parts)
MapShuffle 2 $\forall o.\, if\ in\ (b_{j-1}, b_j]\ send\ to\ M_j$ Where $b_0 = -\infty, b_t = \infty$
Reduce 2 Regular sort
M_1
M_2
M_3
M_t
$b_1,\dots,b_{t-1}$
File
All values in $(b_0, b_1]$ sorted
All values in $(b_1, b_2]$ sorted
All values in $(b_3, b_4]$ sorted
All values in $(b_{t-1}, b_t]$ sorted

How to choose to right p?

Define S_i = the group that arrives to machine M_i at the second phase

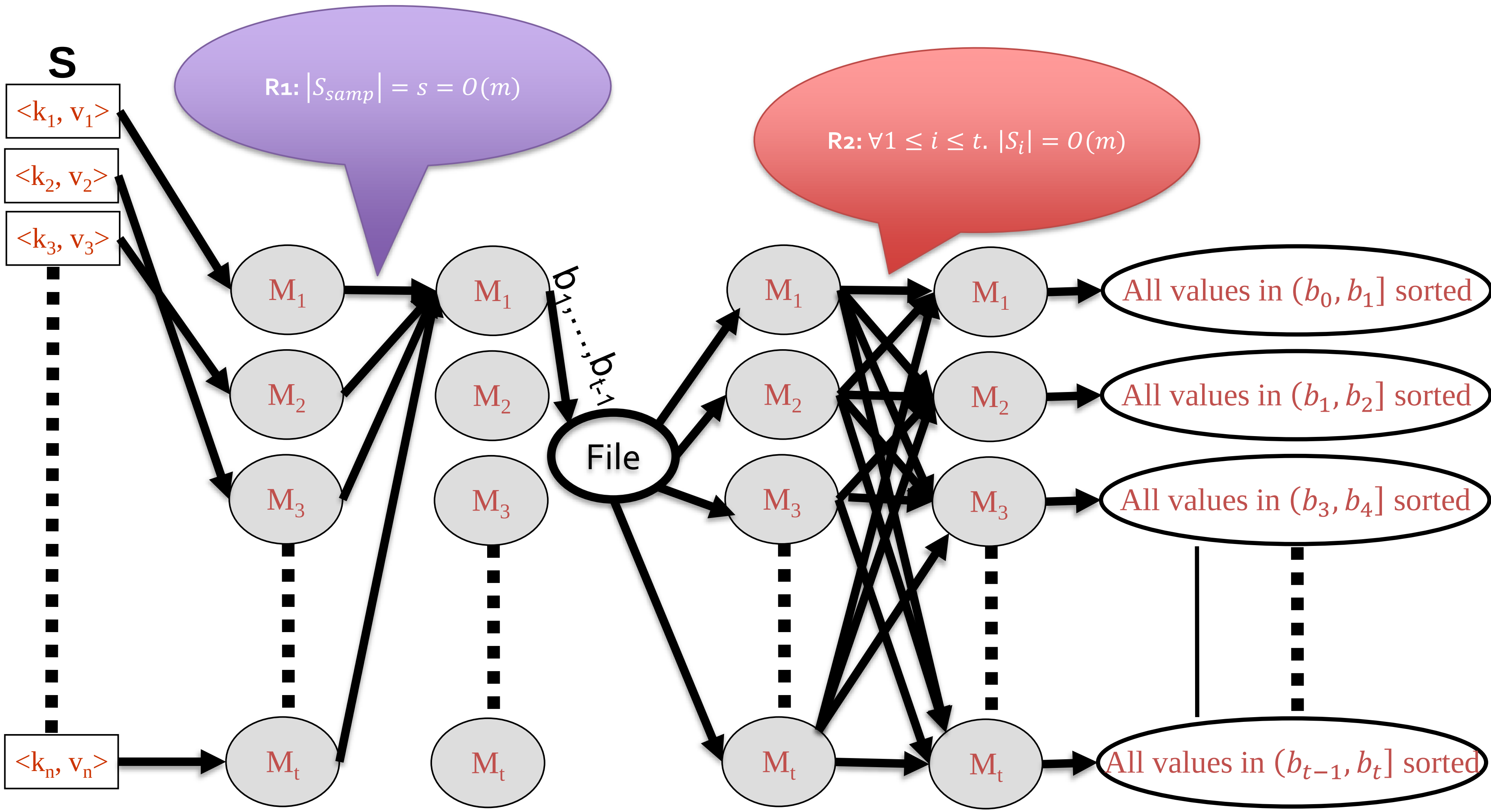
Means $S_i = S \cap (b_{i-1}, b_i]$ for $1 \leq i \leq t$

For Terasort to be minimal we need 2 restrictions:

R1: $\left|S_{samp}\right| = s = O(m)$ Round 1

R2: $\forall 1 \leq i \leq t.\ |S_i| = O(m)$ Round 2

Minimal TeraSort Restrictions



How to choose to right p?

Theorem 1: when $m = \frac{n}{t} > tln(nt)$ both R1 & R2 holds with probability $\geq 1 - O\left(\frac{1}{n}\right)$, when setting $p = \frac{1}{m}\ln(nt)$

When t<9, then $m = \Omega(n)$ and therefore R1 and R2 holds Trivially

Proof (when $t \geq 9$):

» E[s]= $p \cdot n = \left[m = \frac{n}{t}\right] = mpt = t \cdot \ln(nt)$

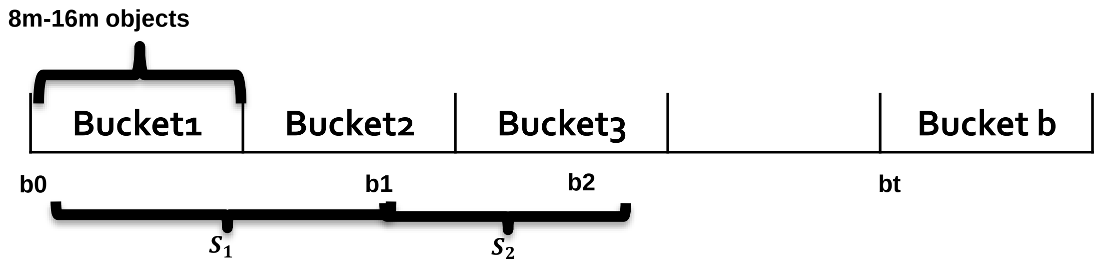
» Chernoff: $\Pr[s \geq a] \leq \frac{E\left[e^{t \cdot s}\right]}{e^{t \cdot a}}$

» $\Pr[R1\ fails] \leq \Pr[s > 1.6 \cdot t \cdot \ln(nt)] \leq \frac{e^{t \cdot \ln(nt)}}{e^{1.6 \cdot t \cdot \ln(nt)}}$

$$= e^{t \cdot \ln(nt) \cdot (-0.6)} = \frac{1}{nt} \cdot e^{0.6t} \overbrace{\leq}^{when\ t \geq 9} \frac{1}{n}$$

This means that when p=$\frac{1}{m}\ln(nt)$ R1 fails with probability $\leq \frac{1}{n}$

What about R2



If we try to divide S (our input) to b=$\lfloor t/8 \rfloor$ buckets as evenly as possible:
if β is a bucket

best case: $\quad$ if $\mathrm{t}\%8 = 0$ and $\mathrm{n}\%\mathrm{b} = 0$ then each bucket has $\frac{n}{\lfloor t/8 \rfloor} = \frac{8n}{t}$ objects

worst case: $\quad$ if $\mathrm{t}\%8 = 7$ and $\mathrm{n}\%\mathrm{b} \neq 0$ then β can hold at most $\frac{n}{t/16} = \frac{16n}{t}$ objects

$$8\mathrm{m} = \frac{8n}{t} \leq |\beta| \leq \frac{16n}{t} \leq 16m$$

Theorem: if after choosing our boundary objects: $\mathrm{b}_0 = -\infty, b_1, \dots, b_{t-1}, \mathrm{b}_\mathrm{t} = \infty$ every bucket has at least one b_i ➔ R2 will hold

because then each S_i can contain maximum 2 buckets $\leq 2 \cdot 16m = 32m = O(m)$

What about R2

If we are in the case that R1 holds – s $\leq 1.6 \cdot t \cdot \ln(nt)$ as before:

» If $|\beta| \geq \frac{s}{t} = 1.6\ln(nt)$ samples (objects from S_{samp}) than

» What is the probability that $|\beta| < 1.6\ln(nt)$?

$$\forall 1 \leq i \leq |\beta|.\, x_i = \begin{cases} 1 & x_i \in S_{samp} \\ 0 & x_i \notin S_{samp} \end{cases} \Rightarrow X = \sum_{j=1}^{|\beta|} x_j = |\beta \cap S_{samp}|$$

$$|\beta| \geq 8m \Rightarrow E[X] \geq 8mp = 8\ln(nt)$$

$$\Pr[\beta\ doesn't\ contain\ any\ b_i] \leq$$

$$\begin{aligned} \mathbf{Pr}[X \leq 1.6\ln(nt)] &= \mathbf{Pr}[X \leq (1-4/5)8\ln(nt)] \\ &\leq \mathbf{Pr}[X \leq (1-4/5)\mathbf{E}[X]] \\ \text{(by Chernoff)} &\leq \exp\left(-\frac{16}{25}\frac{\mathbf{E}[X]}{3}\right) \\ &\leq \exp\left(-\frac{16}{25}\cdot\frac{8\ln(nt)}{3}\right) \\ &\leq \exp(-\ln(nt)) \\ &\leq 1/(nt). \end{aligned}$$

The probability that one bucket fails

We have t/8 buckets the probability that 1+ fails is $\leq \frac{1}{8n}$

So in total…

Theorem 1: when $m = \frac{n}{t} > tln(nt)$ both R1 & R2 holds with probability $\geq 1 - O\left(\frac{1}{n}\right)$, when setting $p = \frac{1}{m}\ln(nt)$

Proof so far:

When $m = \frac{n}{t} > tln(nt)$ we saw that:

- R1 fails with probability $\leq \frac{1}{n}$
- When R1 doesn't fail ($s \leq 1.6 \cdot t \cdot \ln(nt)$)
  R2 fails with probability $\leq \frac{1}{8n}$

$$\text{P[R2 fails | R1 holds]} \leq \frac{1}{8n} \cdot \frac{n-1}{n} \leq \frac{9}{8n}$$

- P[R1 ∨ R2 fails] $=$ P[R2 fails | R1 holds] $+$ P[R2 fails | R1 fails] $\leq$ P[R2 fails | R1 holds] $+$ P[R1 fails] $\leq \frac{9}{8n} + \frac{1}{n} = \frac{17}{8n}$

  P[R1 ∧ R2 hold] $\geq 1 - \frac{17}{8n}$

  ➔ P[R1 ∧ R2 hold] = $1 - O\left(\frac{1}{n}\right)$

When R1 & R2 hold - Minimality?

1. Minimum footprint: at all times – each machine uses only o(m) space ✓

2. Bounded net-traffic: in each round every machine send and receives at most O(m) words
R1 ensures M1 receives and sends only O(m) in round 1
R2 ensures all M's receive and send only O(m) in round 2 ✓

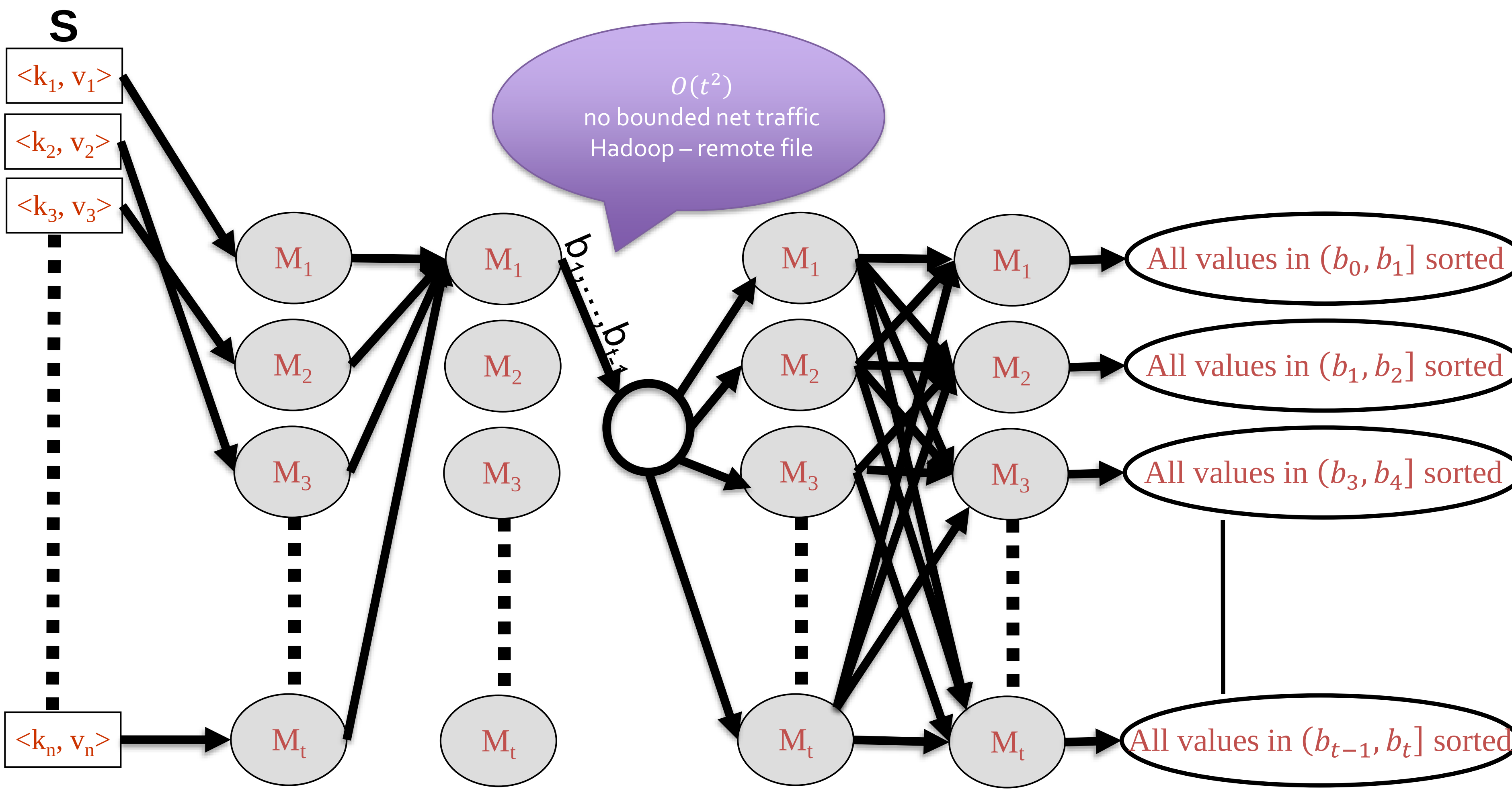
3. constant round: 2 rounds ✓

4. Optimal computation: every machine performs only O(T_{seq}/t) amout of computation total
when T_{seq} = time to solve the same problem on single sequential computer ✓
means the algorithm would get a speedup of t by using t machines in parallel

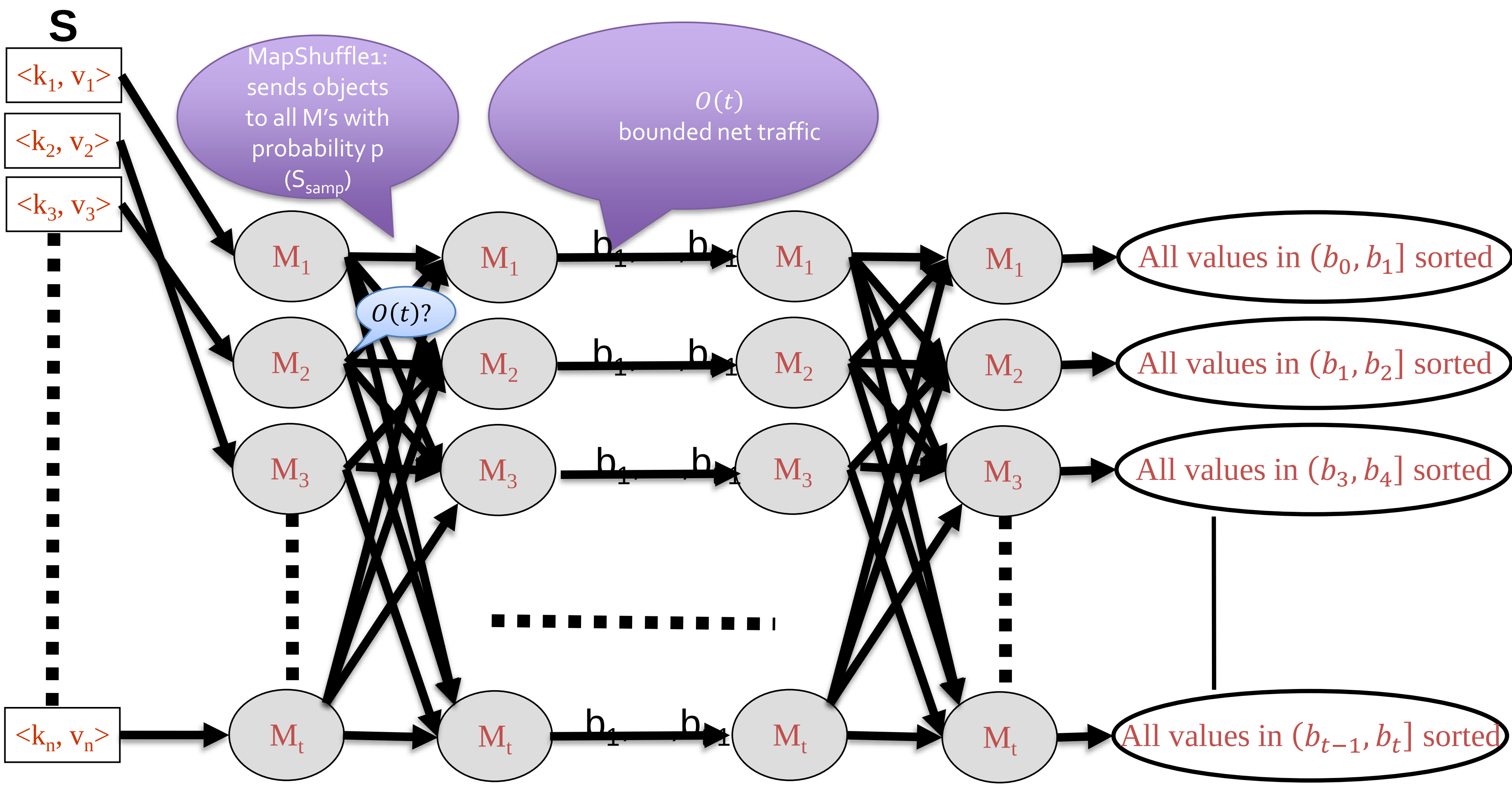
Sorting S_i in round 2: $O(mlogm) = \mathbf{O}\left(\frac{\mathbf{n}}{\mathbf{t}}\mathbf{logn}\right) = \frac{\mathbf{1}}{\mathbf{t}}\mathbf{O}(\mathbf{nlogn})$

In practice:
typically m >> t
the memory size of a machine is significantly greater the number of machines
m= $O(10^6)$ bytes = O(MB)
t = 10^4 machines or lower
therefore $m \geq tln(nt)$ is a very reasonable assumption = excellent efficiency in practice

TeraSort – broadcast assumption



Pure TeraSort



Pure Terasort - Sending words in round 1

Each sample word is being sent to t machines

Lemma : p[every machine sends O(tln(nt)) words] $\geq 1 - \frac{1}{n}$

Proof: X – random variable = the number of object sampled from machine M

$$E[X] = mp = \ln(nt)$$

Chernoff: $\Pr[X \geq 6\ln(nt)] \leq 2^{-6\ln(nt)} \leq \frac{1}{nt}$

P[M samples and sends more than $O\big(tln(nt)\big)$ in round 1] $\leq \frac{1}{nt}$

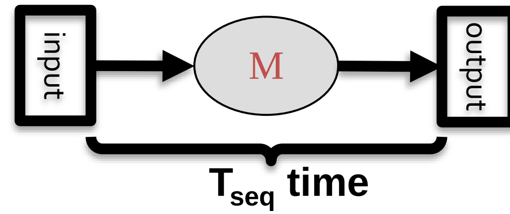
For t machines ➔ 1 machine or more sends more than $O\big(tln(nt)\big)$ in round 1

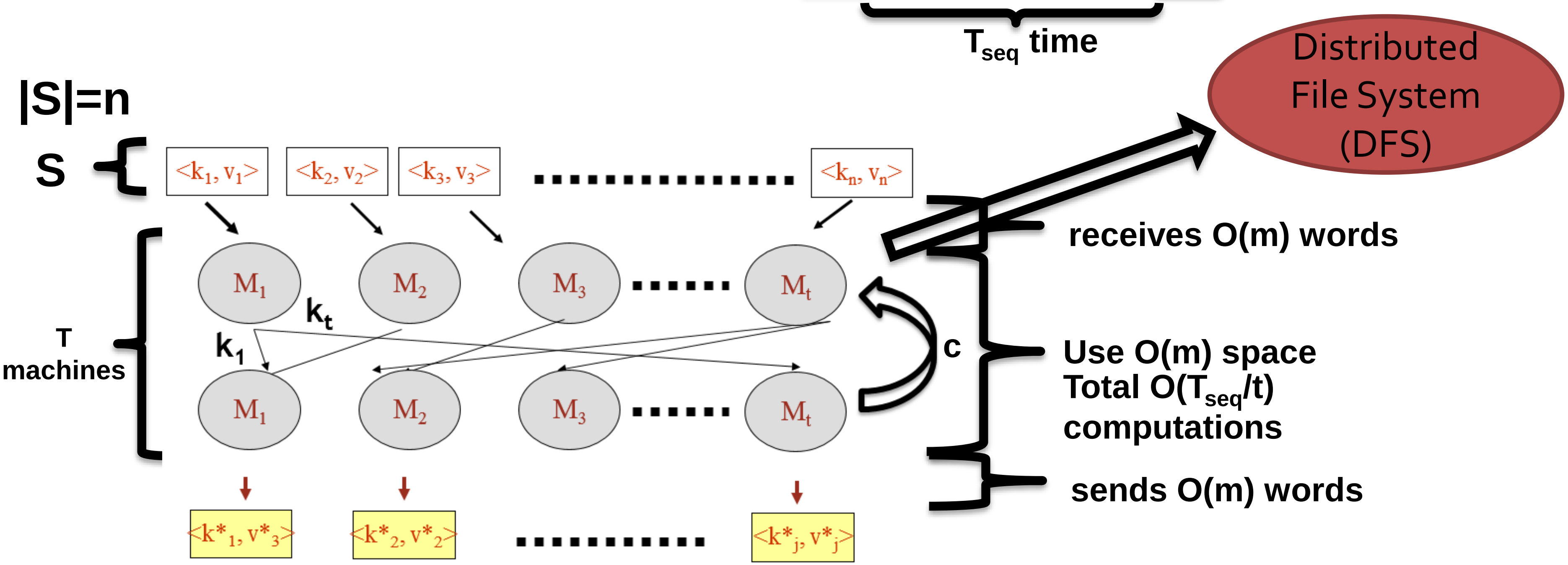
with probability $\leq \underbrace{\frac{1}{nt} + \cdots + \frac{1}{nt}}_{t} = \frac{1}{n}$

p[every machine sends O(tln(nt)) words] $= 1 - O\left(\frac{1}{n}\right)$

Combining it with the previous calculations – pure TeraSort is minimal with probability $\geq 1 - O\left(\frac{1}{n}\right)$ $when\ m \geq tln(nt)$

Minimal MapReduce





Minimal MapReduce advantages:

1. Each machine using O(m) in each phase = O(1/t) of S – prevent partition skew
2. Bounded net traffic – O(m) words ensures that each shuffle phase transfer at most o(n) words
3. Due to parallelization:
   The duration of each phase is ≈ the time for a machine to send and receive O(m) words.
   Good for building stateless algorithm - improve the system's robustness
4. Constant rounds = O(n) word traffic overall
5. Optimal computation – the very original motivation of MapReduce – do things faster

Using minimal TeraSort to make DB algorithms minimal

Terasort based algorithms in databases

Database algorithms:
- » Ranking
- » Group-by
- » Semi-join
- » 2D-skyline
- » Etc…

All are O(nlogn) on sequential machine

All include TeraSort + extra round (MapShuffle + Reduce) where:
- » Each machine sends O(t) words of network traffic

From now on we assume $m \geq tln(nt)$ for TeraSort to be minimal

Prefix sum

S – objects with weight function w

Prefix(o, S) = $\sum_{o'<o} w(o')$

O(nlogn) time in sequential machine

S_i the numbers on machine M_i after sorting

MapShuffle:

» M_i $computes$ $W_i = \sum_{o \in S_i} w(o)$
» M_i sends W_i to $M_{i+1}, \dots, M_t$

Reduce:

» M_i computes $V_i = \sum_{j<i} W_j$
» M_i computes locally $prefix(o, S_i)$
» M_i computes for each o∈ S_i: $prefix(o, S) = V_i + prefix(o, S_i)$

In MapShuffle M_i sends t-i words, and receives in reduce i-1 words = t-1 = O(t) and because $t \leq m$ o(m)

S

Obj	O5	O3	O4	O6	O2	O1
w	8	2	3	1	4	7

S sorted

Obj	O1	O2	O3	O4	O5	O6
w	7	4	2	3	8	1
PS	0	7	11	13	16	24

Prefix Min

S – objects with weight function w

Prefix(o, S) = $\cancel{\sum_{o'<o} w(o')}$

$$\min\{w(o')|o' < o, \quad o' \in S\}$$

S

Obj	O5	O3	O4	O6	O2	O1
w	8	2	3	1	4	7

S sorted

Obj	O1	O2	O3	O4	O5	O6
w	7	4	2	3	8	1
PM	∞	7	4	2	2	2

O(nlogn) time in sequential machine

S_i the numbers on machine M_i after sorting

MapShuffle:

» M_i *computes* $W_i = \cancel{\sum_{o \in S_i} w(o)}$ $\min\{w(o)|o \in S_i\}$

» M_i sends W_i to $M_{i+1}, \dots, M_t$

Reduce:

» M_i computes $V_i = \cancel{\sum_{j<i} W_j}$ $\min\{W_j | j < i\}$

» M_i computes locally $prefix(o, S_i)$

» M_i computes for each o∈ S_i: $prefix(o, S) = \cancel{V_i + prefix(o}, S_i)$ $\min(V_i, prefix(o, S_i))$

In MapShuffle M_i sends t-i words, and receives in reduce i-1 words = t-1 = O(t) and because $t \le m$ o(m)

Ranking =
Prefix sum with w=1

S

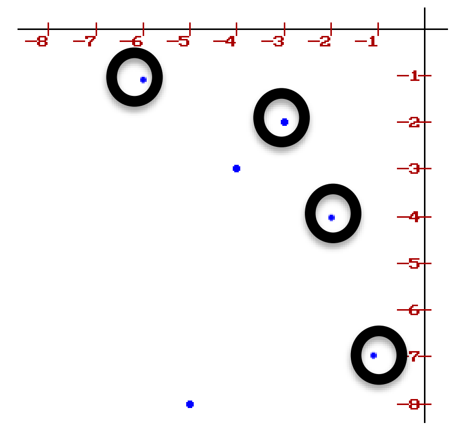
Obj	O5	O3	O4	O6	O2	O1
w	1	1	1	1	1	1

S sorted

Obj	O1	O2	O3	O4	O5	O6
w	1	1	1	1	1	1
R	0	1	2	3	4	5

Skyline

(-5,-8) , (-3,-2) , (-4,-3) , (-6,-1) , (-2,-4) , (-1,-7)



S

Obj	O5	O3	O4	O6	O2	O1
w	8	2	3	1	4	7

S sorted

Obj	O1	O2	O3	O4	O5	O6
w	7	4	2	3	8	1
PM	∞	7	4	2	2	2

S = {(x,y)}

Skylines(S)

=

$\{p=(x_p,y_p) \mid \forall p' \neq p \in S.\left(x_p > x_{p'}\right) \vee \left(y_p > y_{p'}\right)\}$

Prefix Min in Disguise:

» S = {-x_p}, w=-y_p, skylines: all the points such that $w(p) < prefixMin(p)$

GroupBy

S where for each o∈S – k(o) - key, w(o) – weight, AGG – distributive aggregate function

We want to get $\{AGG(G_k)|G_k = \{o|k(o) = k\}\}$

First: TeraSort while sorting objects by keys, breaking ties by id

The main problem:
if there is a key k s.t. $|G_k|$>>m and can't fit in one machine

Lets look on one machine M:

» denote $k_{min}(M), k_{max}(M)$ – smallest and largest keys on M
» Every k s.t. $k_{min}(M) < k < k_{max}(M)$ can be stored and locally computed on M
» So at most 2 "problematic" keys per machine ➔ at most 2t overall

Solution: send the problem to M1

» MapShuffle: compute $\left(k_{min}(M_i), AGG(\{o \in M_i|k(o) = k_{min}(M_i)\})\right)$ and $\left(k_{max}(M_i), AGG(\{o \in M_i|k(o) = k_{max}(M_i)\})\right)$. send both to M1 (one if $k_{min}(M_i)$=$k_{max}(M_i)$)
» Reduce on M1: compute $AGG\left(\left\{w_j\middle|k_j = k\right\}\right)$

$$\mathrm{t} \leq \mathrm{m} = \frac{\mathrm{n}}{\mathrm{t}} \Rightarrow O\big(t\log(t)\big) = O\left(\frac{n}{t}\log(n)\right)$$

Can help solve "word count" problem we saw earlier

Semi-Join

R, T two sets from the same domain. Each object has a key.
SemiJoin(R,T)=$\{o \in R | \exists o' \in T. k(o) = k(o')\}$

Almost like GroupBy

Sort $S = R \cup T$ with terasort while saving the source group

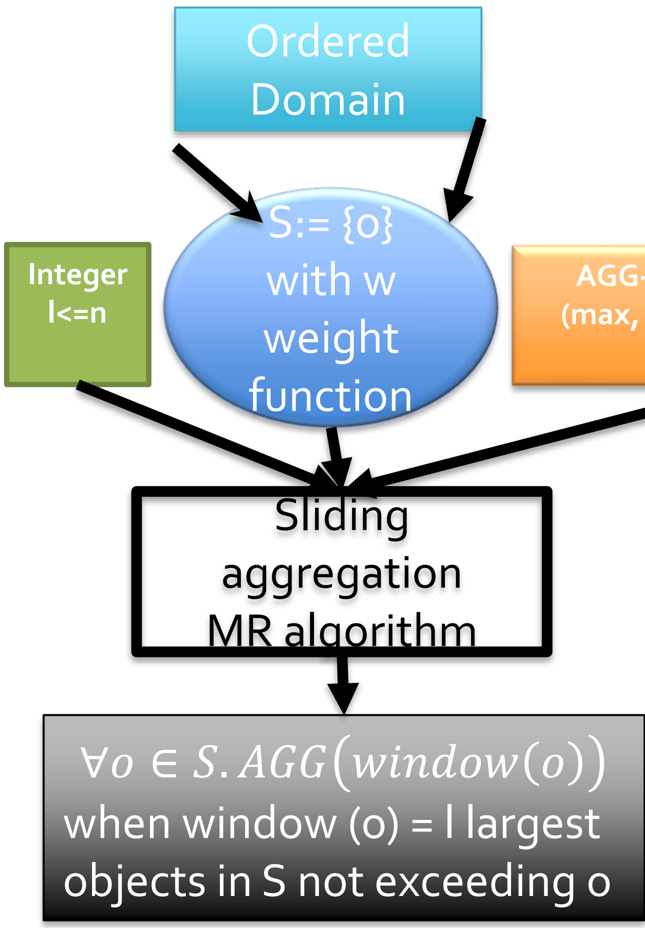
MapShuffle - each M_i sends $k_{min}(T, M_i)$, $k_{max}(T, M_i)$ to all machines

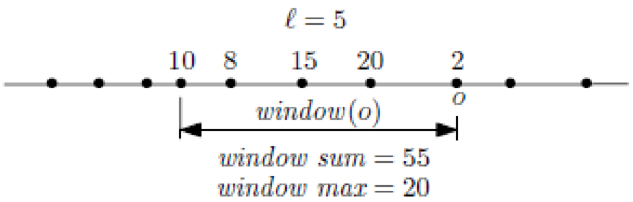
Denote :
$K(T, M_i)$ -the set of keys of T objects stored in M_i
K_{border} the set from the previous MapShuffle
Reduce: for every R-object o in M_i output if $k(o) \in K(T, M_i) \cup K_{border}$

MapShuffle – 2t pairs are sent to all the machine
R objects in M are already sorted - Reduce – O(t+mlogm)=$O\left(\frac{n}{t} log(n)\right)$

Sliding aggregation

Sliding aggregation remainder





Before getting started: Sorting with perfect balance

Input : set S where |S|=n.

We assume n%t = 0 so m is an integer – if not pad S with dummy zeros.

What we want in the end:
In every machine M_i $(1 \leq i \leq t-1)$ there will be exactly m objects
their rank range is: $[(i-1)m+1, im]$

Sounds very familiar to the ranking problem.

Solved by adding another round after the ranking algorithm:
previous reduce : Each machine M_i computed rank(o) for each $o \in S_i$
MapShuffle: fore each $o \in S_i$ compute $j = \left\lceil \frac{rank(o)}{m} \right\rceil$ and send it to M_j
Reduce: do nothing.

Minimal

Back to Sliding Aggregation

Solved with just one more round after sorting with perfect balance

We got: In every machine M_i $(1 \leq i \leq t-1)$ there will be exactly m objects
their rank range is: $[(i-1)m+1, im]$

$$Window(o) = [rank(o) - l + 1, rank(o)]$$

if rank(o) = 6, l=3, window(o)={4,5,6}

Therefore objects in the window of o are in machines $M_\alpha, M_{\alpha+1}, \dots, M_\beta$
where $\alpha = \left\lceil \frac{rank(o)-l+1}{m} \right\rceil, \beta = \left\lceil \frac{rank(o)}{m} \right\rceil$

» M_β is where o is currently is - the calculation will be committed there
» if $\alpha = \beta$ AGG(window(o)) can be computed locally on M_β – not a problem.

So from now on we focus on $\alpha < \beta$

If $\alpha < \beta - 1$ then $Window(o)$ includes all objects in machine $\alpha + 1, \dots, \beta - 1$
So if $W_i = AGG(\{o' \in M_i\})$ we will ensure that every machine knows $W_1, \dots, W_t$

Now to calculate $AGG\big(window(o)\big)$ M_β only needs is $AGG(\{o' \in M_\alpha | o' \in Window(o)\})$
We call those objects in M_α "remotely relevant to M_β"
If M_α stores at least one remotely relevant object to M_β, M_α is "pertinent" to M_β

Back to Sliding Aggregation

Lemma: every object is remotely relevant to at most 2 machines

Objects in machine M_i can be remotely relevant only to :

– *machine* $i + 1$, *if* $\ell \leq m$

– *machines* $i + \lfloor(\ell - 1)/m\rfloor$ *and* $i + 1 + \lfloor(\ell - 1)/m\rfloor$, *otherwise.*

(if the machine id >m ignore it)

New MapShuffle (Machine M_i):

» Compute and send W_i to all machines

» Send all objects in M_i to the machines to 1 / 2 machines