Parallel, Concurrent & Distributed Programming in JAVA

www.pushpsoad.com

ASYNC
FORK

FINISH
JOIN

JAVA API

JAVA PARALLEL

Same thing

Task List

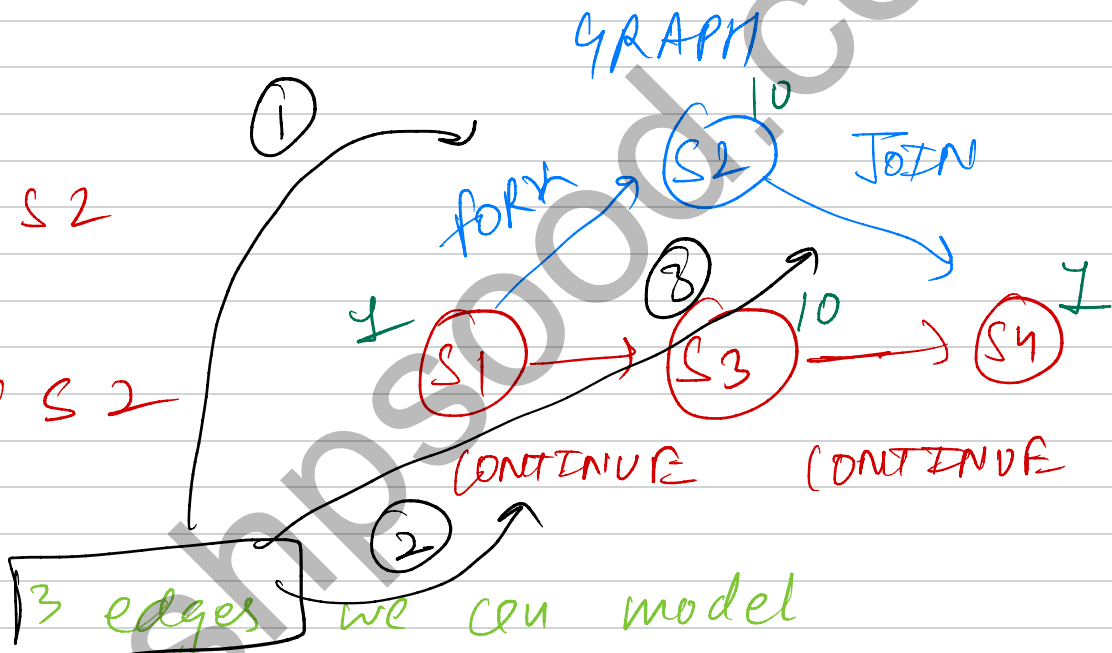
S1

FORK S2

S3

JOIN S2

S4



→ With these 3 edges we can model any parallel program

→ If there is a path directly b/w two nodes in the graph they cannot run in parallel for eg (S2) & (S3) can run in parallel, but (S2) & (S4) cannot

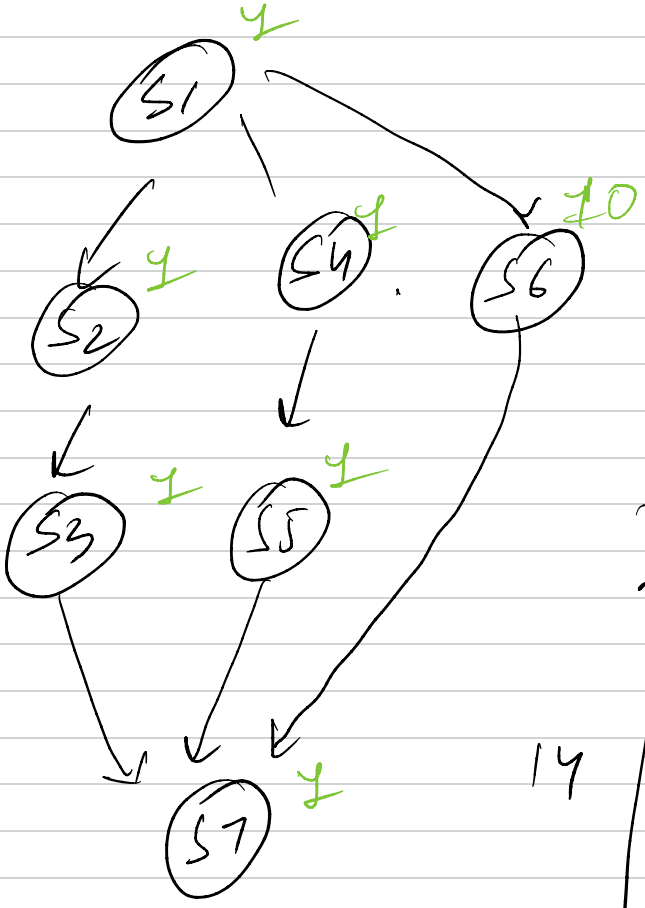
HERE → WORK = 22
(Total sum of all nodes)

→ SPAN = 12
(sum of longest path)

IDEAL PARALLISM =
$\frac{WORK}{SPAN} = \frac{22}{12}$

www.pushpsood.com

Multiprocessor scheduling, parallel speedup



T_p = Execution time on p processes

p is processor (2 in our case) done by H/W

P_0	P_1
S1	IDLE
S2	S4
S3	S5
S6	IDLE
S7	IDLE

14

SCHEDULE 1

Here, T_2 = 14

P_0	P_1
S4	IDLE
S6	S2
	S3
	S4
	S5
S7	

10 | 4

Here T_2 = 12

www.pushpsood.com

If $p = 1$, T_1 = WORK $T_1 = 16$

($p=1$ / 1 processor)

Here p is ∞ ← T_∞ = SPAN (length of longest path) = 12

$$T_\infty \leq T_p \leq T_1$$

$$\text{SPEEDUP} = \frac{T_1}{T_p}$$ (How much faster the parallel version was able to run)

BOUND ON SPEEDUP

$$\text{SPEEDUP} \leq p$$

$$\text{SPEEDUP} \leq \frac{\text{WORK}}{\text{SPAN}}$$ = IDEAL PARALLELISM

www.pushpsood.com

Amdahl's Law

(Interesting observation made by Amdahl 50 yrs back)

$$SPEEDUP \leq \frac{WORK}{SPAN}$$

q = FRACTION SEQUENTIAL

$$SPEEDUP \leq 1/q$$

$q = 0.5 \Rightarrow SPEEDUP \leq 2$

↓

No Need for large parallelism

$q = 0.1 \Rightarrow SPEEDUP \leq 10$

PROOF

$SPAN \geq q \times WORK$

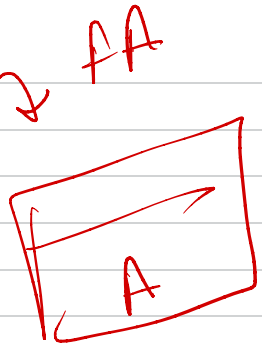
$q \times WORK$ must be included in the computational graph (Lower bound)

$$SPEEDUP \leq WORK/SPAN$$

$$SPEEDUP \leq \frac{WORK}{q \times WORK}$$

www.pushpsood.com

futures



$A = f(B)$

$C = g(A)$

$D = H(A)$ → only dependent on A.

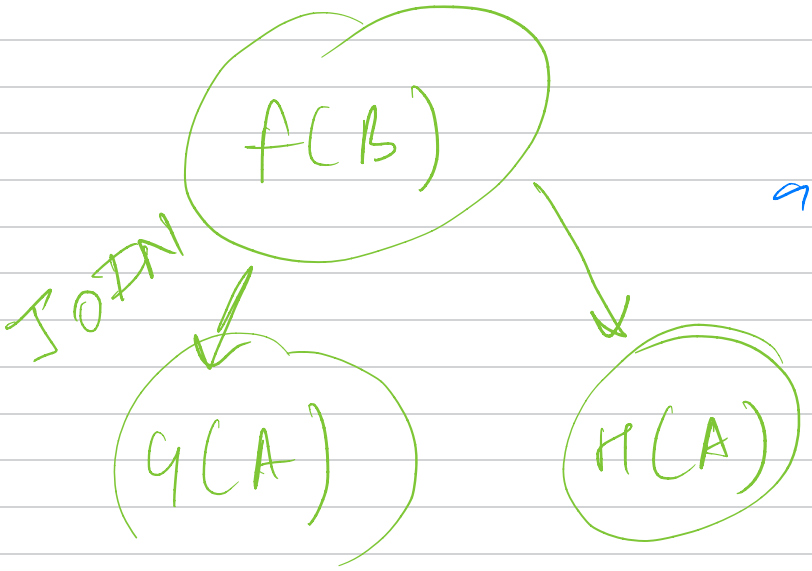
$FA = FUTURE(f(B))$

$FC = FUTURE(g(f(A).get()))$

$FD = FUTURE(H(f(A).get()))$

BLOCK / WAIT until A is Available

www.pushpsood.com



future provide a very natural way to convert functional program to parallel program

```
fSUM1 = FUTURE{ LOWER };

fSUM2 = FUTURE{ UPPER };

SUM = fSUM1.get() + fSUM2.get()
```

www.pushpsood.com

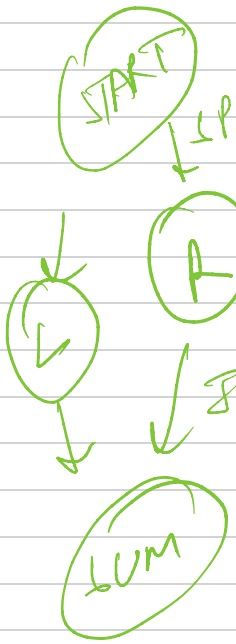
futures in JAVA FORK & JOIN

* NOT RECURSIVEACTION

CLASS ASUM EXTEND RECURSIVETASK

FIELDS : ARR, LO, HI

COMPUTE() {



if (low > HI) RETURN 0;

else if (low == HI) RETURN ARR[LO]

else {

MID = (low + HIGH) / 2

L = NEW ASUM (low, MID, ARR)

R = NEW ASUM (MID+1, HI, ARR)

R.FORK(); // FUTURE

RETURN L.COMPUTE() + R.JOIN();

}

Join for future implements future get operation

www.pushpsood.com

Memoization

y1 = g(x1)
INSERT(g, x1, y1)
y2 = g(x2)

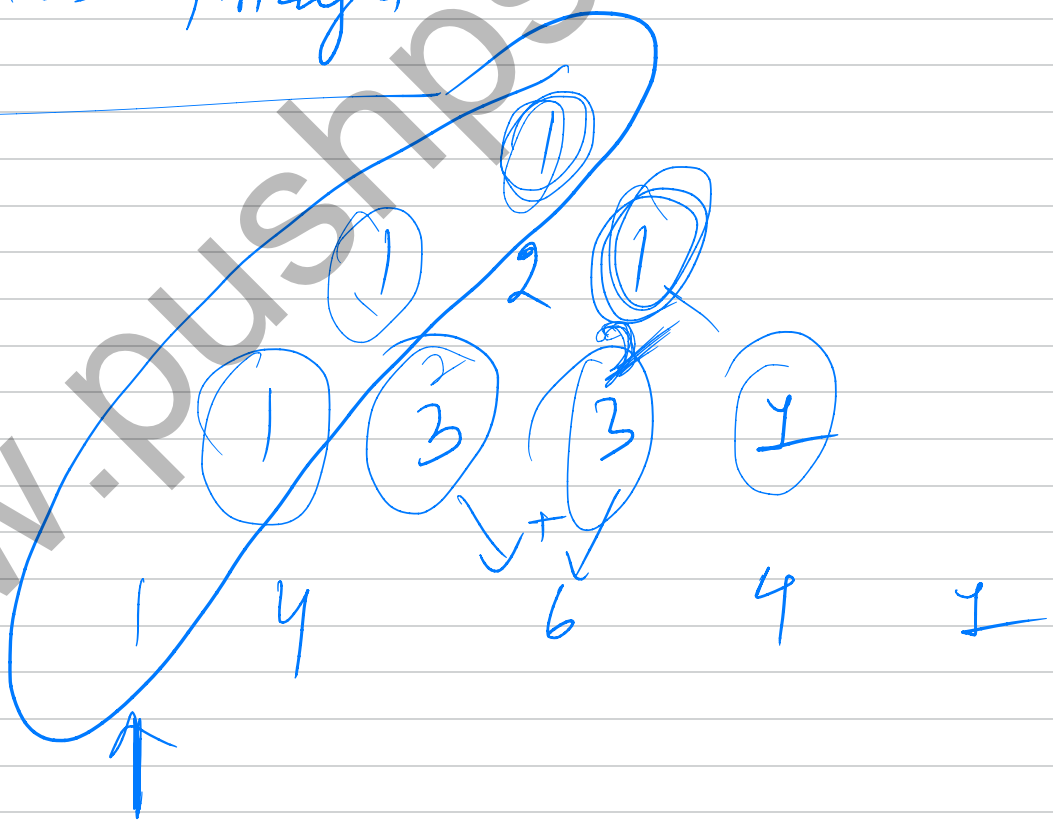
y3 = ~~g(x1)~~ →

LOOKUP(g, x1)

fyt = FUTURE{ g(x1) }
INSERT(g, x1, fyt)
y2 = g(x2)
wasted work (in sequential)

y3 = LOOKUP(g, x1).get()

PASCALS Triangle



www.pushpsood.com

Java Streams

1.

```
FOR ( S : STUDENTS )

    PRINT S

STUDENTS.STREAM().( S → PRINTS )
```

2.

```
FOR ( S : STUDENTS )

    if ( S is ACTIVE )
        ACTIVE.ADD( S )

FOR ( A : ACTIVE )

    AGESUM += A.AGE ;

AVG = AGESUM / ACTIV.SIZE()
```

www.pushpsood.com

USING STREAMS

```
STUDENTS.STREAM()
        .FILTER(S -> S.ISACTIVE())
        .MAP(S -> S.AGE())
        .AVERAGE();
```

TAKEAWAYS

- ITS EASY TO CONVERT SEQUENTIAL STREAM TO PARALLEL
- USE PARALLEL STREAM instead of STREAM
- AVG & FOREACH ARE TERMINAL OPERATIONS
- FILTER & MAP ARE INTERMEDIATE OPERATIONS

www.pushpsood.com

DATA RACES & 2 TYPES DETERMINISM

FUNCTIONAL DET

Same input will always lead to same output

We are interested in parallel programs that are both structurally & functionally deterministic but not always!

Benign Non Determinism (Next Page)

yes no data race

STRUCTURAL DETERMINISM

SAME INPUT lead to SAME COMPUTATIONAL GRAPH (Structure of parallel program)

FUTURE ASYNC

Read

Write

SUM1 = SUM of LOWER HALF

SUM2 = SUM of upper HALF

SUM = SUM1 + SUM2

BUG IN PARALLEL PROG (DATA RACE)

Read & Write OP in parallel

WRITE & WRITE in parallel.

get()

www.pushpsood.com

(vice versa if code is having DRF it is f2s Deterministic)

DRF → Data Race Freedom

vv

(can be achieved when we achieve functional & Struct determinism)

⇒ FUNCTIONAL + STRUCTURAL Determinism

(BENIGN NON DETERMINISM)

Cases when we get diff output but they are all accepted

eg Search → PARALLEL LOOP

~~FINISH~~ &

FOR (i : [0 : M-N])

FOUND = TRUE

ASYNC FOR (j : [0 : N-1]) {

IF (TEXT(i+j) != PAT[j])

FOUND = FALSE

}

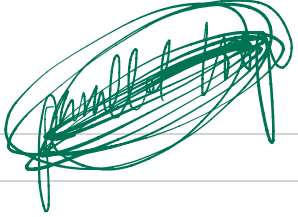
IF (FOUND = TRUE) index = i;

value of index is non deterministic but the answer is acceptable

If Multiple Cousins have same string thread any index is the right example;

www.pushpsood.com

PARALLEL LOOPS



When the task that is being performed inside the loop is independent of each other we can easily use parallelism to compute them independently

eg1

```
finish {
    for (p = head; p != null; p = p.Next)
    {
        async p.compute
    }
}
```

eg2

This will convert it to a parallel stream

```
forall (i : [0-n-1])
    a[i] = b[i] + c[i]
```

www.pushpsood.com

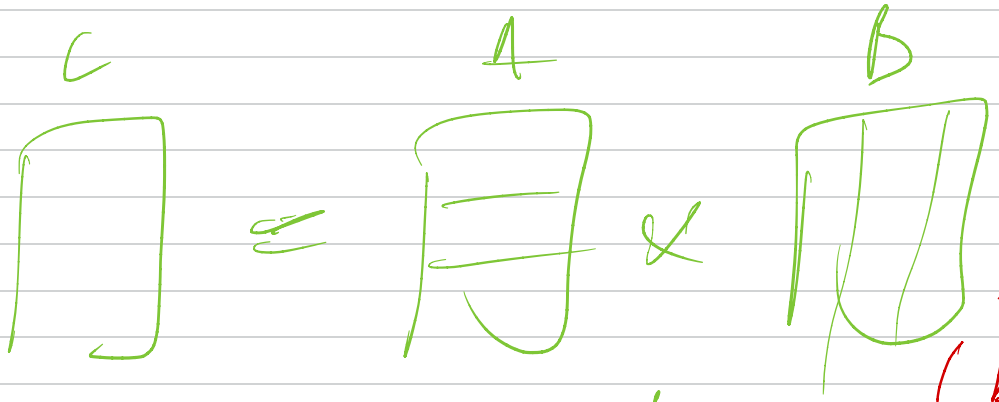
A better way is by using the JAVA API's parallel stream

```
a = IntStream.rangeClosed(0, n-1)
        .parallel()
        .toArray(i -> b[i] + c[i]);
```

www.pushpsood.com

Matrix Multiplication

This is used to measure speed of fastest computers in the world (benchmarking)



$$C[i][j] = \sum_{k=0}^{n-1} A[i][k] * B[k][j]$$

PORALL → Parallelism

~~FOR([i,j] : [0-N-1, 0 : N-1])~~

{

Can we run this in parallel

C[i][j] = 0;

No! This will lead to data race

It will remain SEQ

FOR(K : [0-N-1]) {

C[i][i] += A[i][k] * B[k][j];

}

}

www.pushpsood.com

Barriers in parallel loop

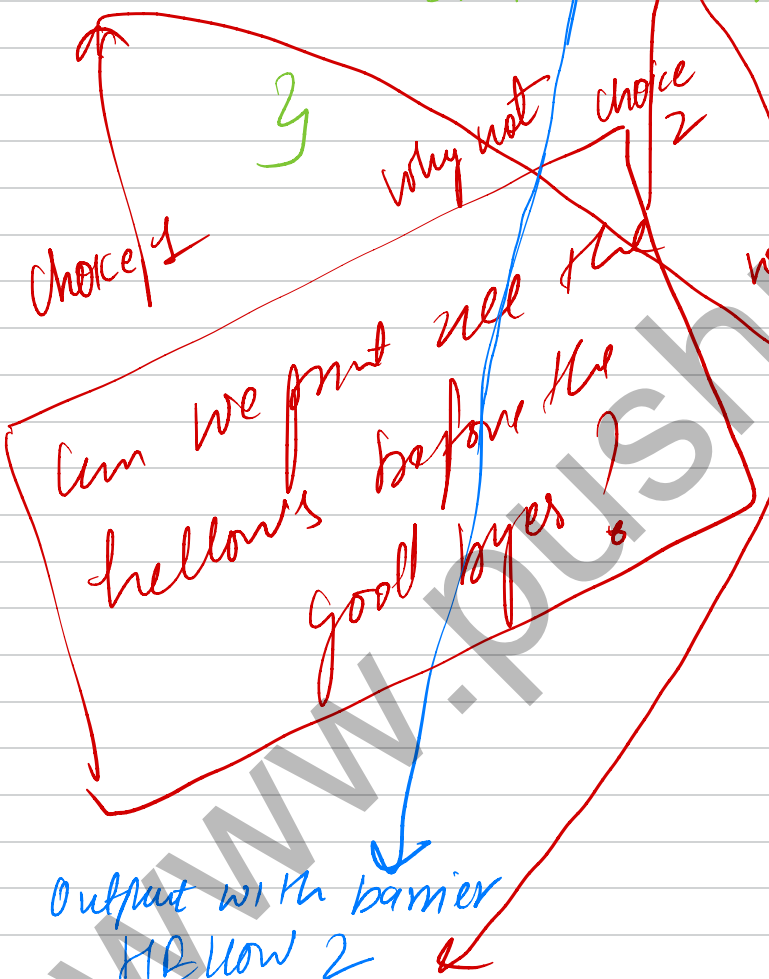
```
FORALL (I : [0 : N-1])
    S = LOOKUP (I);
    PRINT = "Hellow" + S
    BARRIER
    PRINT = "BYE" + S
```

OUTPUT

This order will be retained

HELLOW TWO
HELLOW ZERO
BYE TWO
HELLOW ON
⋮

Different Combination



Output with barrier

HELLOW 2
" 0
" 1
" 1
" 2
" 0

<BARRIER>

GOODBYE 1
" 2
" 0

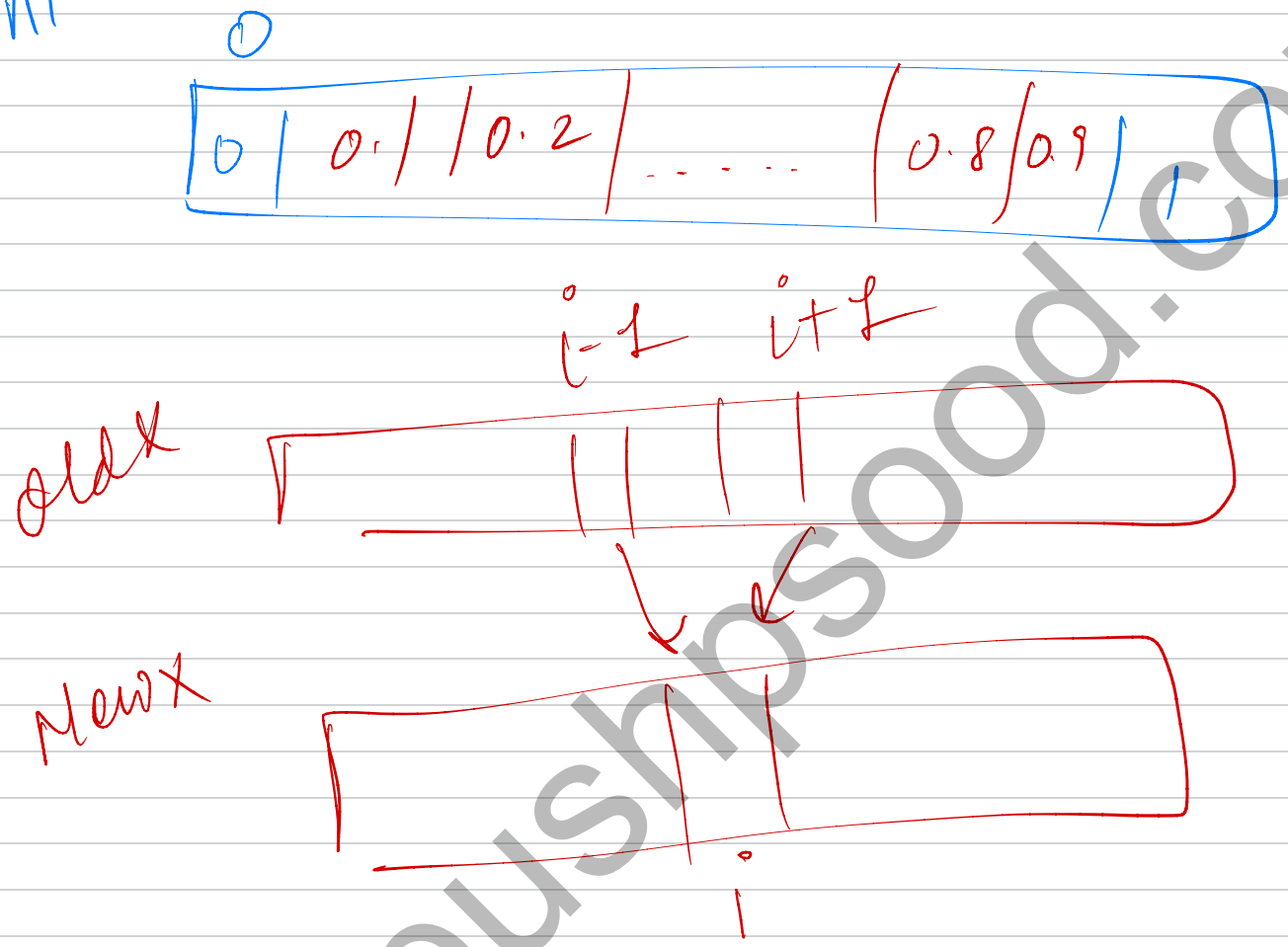
www.pushpsood.com

$x_0 = 0 \qquad x_N = 1$

$$x_i = (x_{i-1} + x_{i+1})/2$$

$0 < i < N$

$n = 10$



ONE DIMENTIONAL ITERATIVE AVERAGING APPROACH

www.pushpsood.com

sequential

FOR (ITER : [1 : NSTEPS]) {

↑ No of time we want to repeat

NSTEPS X (N-1 TASKS)

FORALL (I : [1 : N-2]) {

NEWX[i] = AVG (OLDX[I-1], OLDX[I+1])

}

SWAP POINTERS

}

Quadratic

---

using barriers

(N-1) TASKS

(N-1) X NSTEPS

FORALL (I : [1 : N-2]) { BARRIER

FOR (ITER : [1 : NSTEPS]) {

NEW x[I] = AVG (OLDX[i-1] OLDX[i+1])

BARRIER

SWAP POINTERS

}

}

www.pushpsood.com

FORALL (I : [0 : N-1])

A[I] = B[i] + C[i]

Too Many ! NTASKS

Solution

No of groups

Total task will be NG

FORALL (g : [0 : NG-1]

How to determine?

for [i : MyGroup (g, NG, [0 : N-1])

{

A[I] = B[i] + C[I]

}

Not always required to create these many tasks

fix to too many task is chunking or iteration grouping

CHUNKS of EQUAL SIZE

I = 0 1 2 3 4 5 ..

Block

G = 0 G = 1

NORMAL GROUPING : BLOCK

(1)

i 0 1 2 3 4 5

G1 G3 G2

Group = i MOD(ng)

CYCLIC

(2)

www.pushpsood.com

Split phase Barriers with Java PHASERS

~~CPL/SPAN = 200~~

```
forall (i : [1:N])
    print "HELLOW" + i;
    Do local work
    myid = lookup(i);
    ARRIVE
    AWAIT ADVANCE
    print "GOODBYE" + myid;
```

~100 unit of time

~100 unit of time

Barrier

NEXT

W/ PHASER CPL = 100

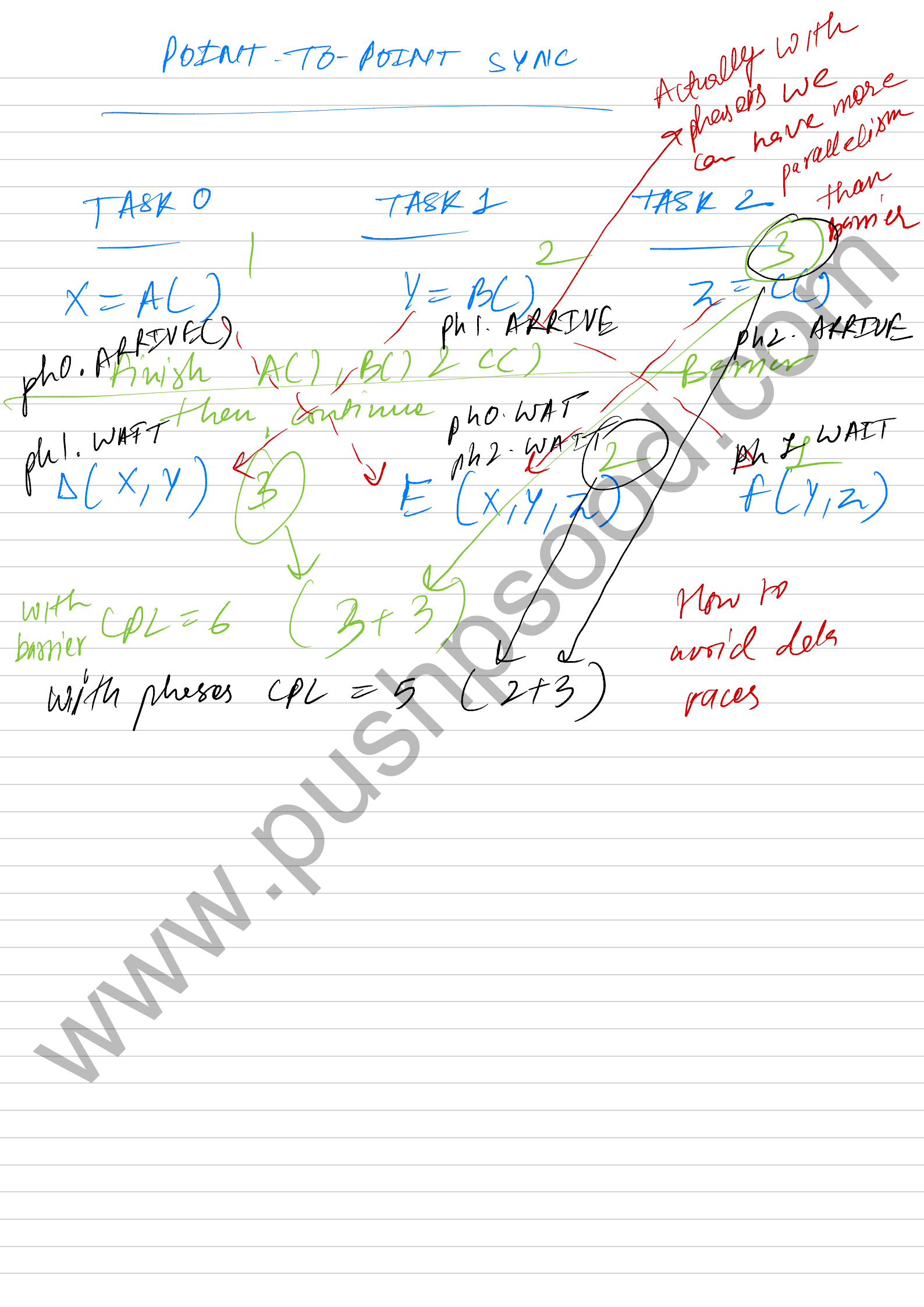
SPAN

- ARRIVE AND AWAIT ADVANCE
  - it is actually a Barrier

- ARRIVE
  - LOCAL WORK → Not associated can be executed
- AWAIT ADVANCE

www.pushpsood.com

POINT-TO-POINT SYNC



www.pushpsood.com

1-D Iterative Averaging with Phasers

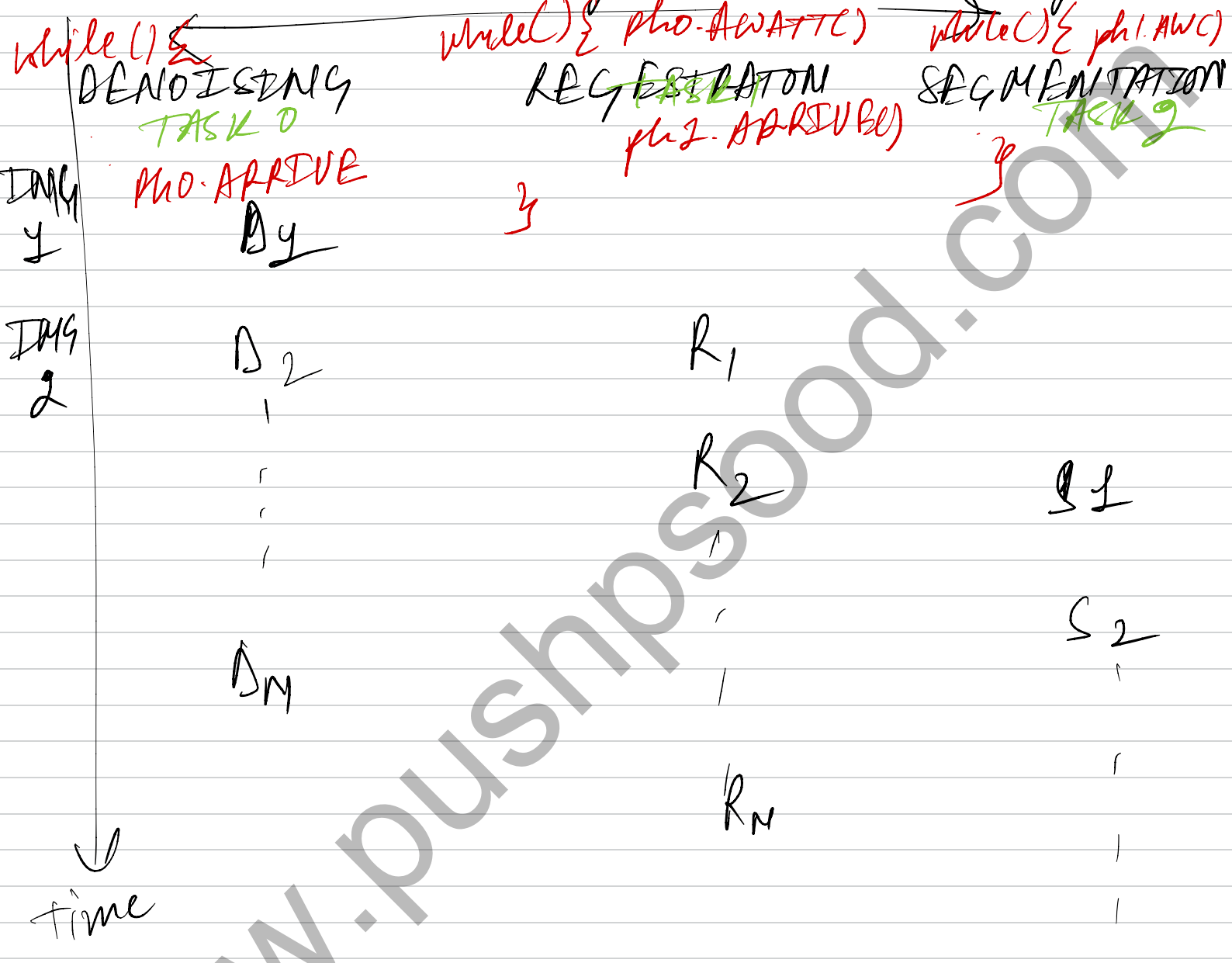
```
ph[] = new Phaser[N+2];
forall (i : [0 : N-1]) {
  for (iter : [0 : nsteps-1]) {
    newX[i] = avg(oldX[i-1], oldX[i], oldX[i+1]);
    ph[i].arrive();
    ph[i-1].wait();
    ph[i+1].wait();
  } // Barrier
  swap newX, oldX
} // used earlier
```

www.pushpsood.com

PIPELINING

MRI have 3 main stages

P pipelining stages



WORK = N * P

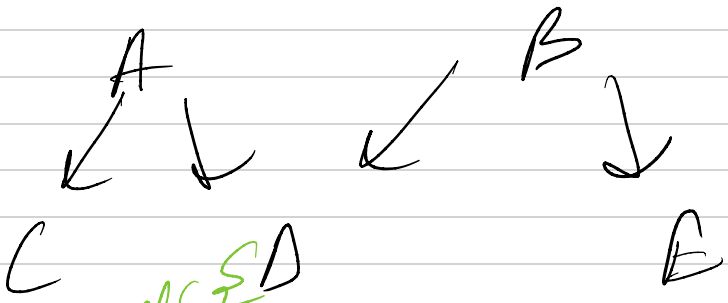
CL = N + P - 1

PARALLELISM = N+P / (N+P-1)

≈ P

www.pushpsood.com

DATA FLOW



ASYNC {
A.GET()
B.GET()
D.WORK
...
}

→ possibility 1

works but implicit dependency!

ASYNC {A.WORK; A.PUT()}

ASYNC {B.WORK; B.PUT()} explicit

ASYNC AWAIT (A) { → precondition

C.WORK }

ASYNC AWAIT (A, B) { D.WORK }

ASYNC AWAIT (B) { E.WORK }

→ possibility 2

www.pushpsood.com

Threads

→ Thread.
* Main program is itself a thread

OS will schedule the threads for processing on processor

P0 P1

Here, P is processor

MAIN THREAD : T0

CREATE : t_1 = new Thread (S_1) → x = 0 (S_1 assigns the var x)

t_2 = new Thread (S_i)

START = t_1.START()

t_2.START()

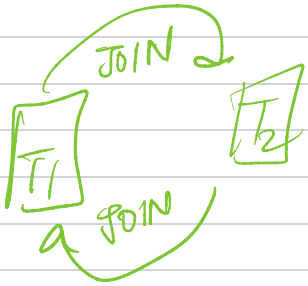
S_2;

JOIN/WAIT = t_1.Join(); ← because of this

(S_3 uses the var x)

S_3 // Requires x

if S_1 is performing t_2.JOIN() and if S_i is performing t_1.JOIN() → leads to Deadlock



we should use threads cautiously and avoid such deadlock conditions

www.pushpsood.com

STRUCTURED LOCKS

Only one resource can Acquire lock at a time.

Q1

```
INCR(A) {
SYNCRONISED(A) { // ACQUIRE A
A.COUNT = A.COUNT - 1
} // RELEASE A
}
```

① ② ③

CAN LEAD TO DATA RACE

INC(A) { INC(A) }

TO // Both the threads are running INCR(A) function at the same time!

(i.e. Exception will remove the lock too) Implicit

Executed by one thread at a time, which will ensure MUTUAL EXCLUSION

Q2

X { BUF, IN, OUT }

```
INSERT(ITEM) {
SYNC(X) { // WAIT IF X IS FULL
BUF[IN] = ITEM;
IN++;
} // NOTIFY
}

REMOVE() {
SYNC(X) { // WAIT IF X IS EMPTY
ITEM = BUF[OUT]
OUT++;
} // NOTIFY
RETURN ITEM;
}
```

WAIT is lifted after notify.

This code requires synchronization.

www.pushpsood.com

SUCCESS

Unstructured Locks

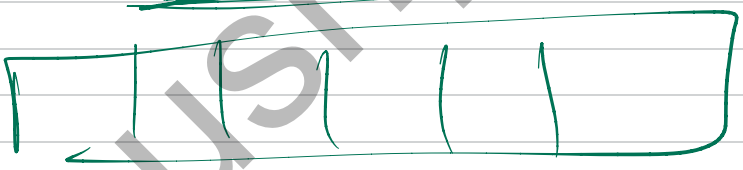
HAND OVER HAND UNLOCKING

```
(
TRY LOCK (L1)
If (SUCCESS) {
LOCK (L1)
WORK with (N1, N2)
UNLOCK (L1), LOCK (L3)
WORK with (N2, N3)
UNLOCK (L2), LOCK (L4)
WORK with (N3, N4)
UNLOCK (L3)
:
}
ELSE { ... } → if unable to get lock in TRY LOCK (L1)
```

NOT NESTED
EXPLICIT CALL TO LOCK & UNLOCK

We can do other work

READ WRITE LOCK



Read write lock on the array L

ARRAY

② Multiple threads can acquire read lock

```
SEARCH(x) { READLOCK(L)
A[i] == x; → only READ operation is being performed
UNLOCK(L)
}
```

① READ operation is being performed

② only one thread can acquire write lock

```
UPDATE(i, y) {
WRITELOCK(L)
A[i] = y;
UNLOCK(L)
}
```

① WRITE operation is being performed

www.pushpsood.com

LIVENESS → fundamental property of concurrent programming

correctness

DEADLOCK → LIVENESS COMPROMISED

① DEADLOCK

T1

SYNC(A){

SYNC(B)

}

}

T2

SYNC(B){

SYNC(A)

}

here the not blocked but they are not making any progress

② LIVELOCK

MUTABLE int

T1

DO {

SYNC(X) X.INC()

R = X.GET()

} WHILE(R < 2)

Even adding synchronise won't fix the issue and lead to livelock (infinite loop)

diagram

LIVE LOCK

T1

X = 0

X = 1

T2

X = 0

X = 1

X = 0

DO {

SYNC(X) {

X.DEC()

R = X.GET()

}

} WHILE(R > 2)

www.pushpsood.com

(3) STARVATION

S_1 ..... S_{100} → We have 100 sockets with us

NO LIVELOCK & DEADLOCK but STARVATION because of limited processors

T_1

do {

IN1 = S1.READLINE()

PRINT IN1

} while (...)

T_2

:

T_{100} → STARVES AS THE OTHER THREADS ARE TAKING UP ALL THE THREADS

do {

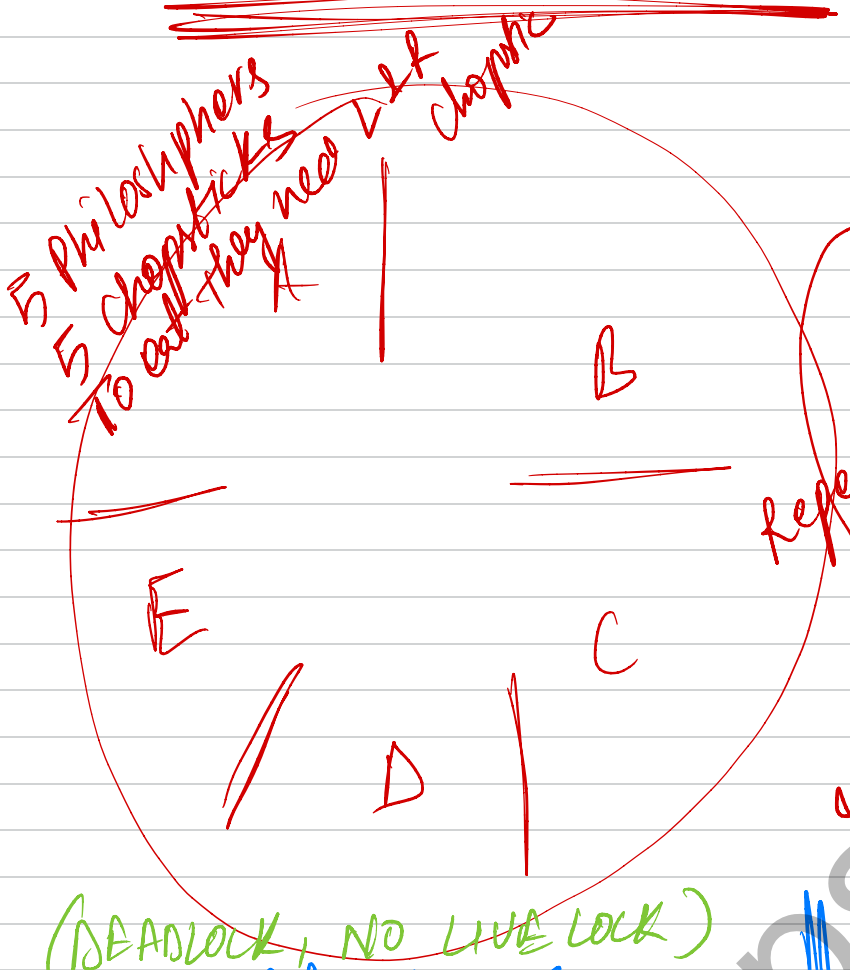
IN100 = S100.READLINE()

PRINT IN100

} while (...)

www.pushpsood.com

DINING PHILOSOPHERS (LIVELOCK & DEADLOCK CLASSICAL EXAMPLE)



repeat
- THINK
- PICKUP CHOPSTICK
- EAT
- PUTDOWN CHOPSTICK

Code

Never able to eat

(DEADLOCK, NO LIVE LOCK)
STRUCTURED LOCKS

```
WHILE(...) {
  THINK
  SYNC(LEFT) {
    SYNC(RIGHT) {
      EAT
    }
  }
}
```

(NO deadlock, but Livelock)
UNSTRUCTURED LOCKS

```
WHILE(...) {
  THINK
  S1 = TRYLOCK(LEFT)
  if(!S1) CONTINUE;
  S2 = TRYLOCK(RIGHT)
  if(!S2) {
    UNLOCK(LEFT)
    CONTINUE
  }
}
```

www.pushpsood.com

POSSIBLE DEADLOCK IN STRUCTURED LOCK

All Philosopher done thinking and pick the chopstick on their LEFT and keep waiting for the right one, which could never happen

DEADLOCK

UNSTRUCTURE LOCKS (SAME EXAMPLE IN PREVIOUS PAGE)

```
while (...) {
    THINK
    s1 = TRYLOCK(LEFT);
    if (!s1) CONTINUE;
    s2 = TRYLOCK(RIGHT);
    if (!s2) {
        UNLOCK(LEFT);
        CONTINUE;
    }
    EAT
}
```

→ GAURANTEES NO DEADLOCK

If all of them fail to grab the right one they would be able to eat which is LIVELOCK

MODIFICATION

Ephilosphr → PICK RIGHT
PICK LEFT

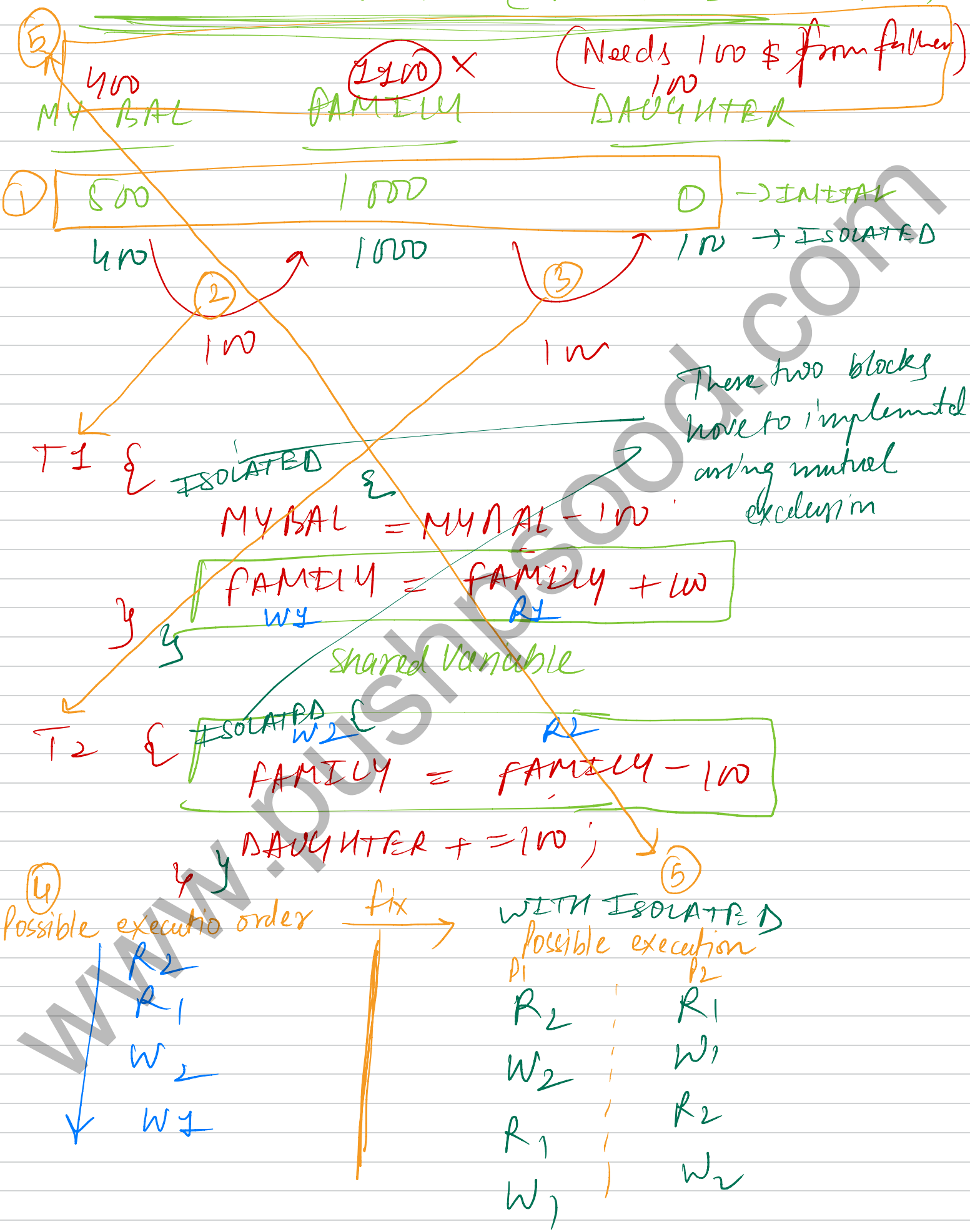
} Change order of one philosopher

CAN LEAD TO STARVATION which is known issue, but this can be fixed using SEMAPHORES

www.pushpsood.com

CRITICAL SECTION

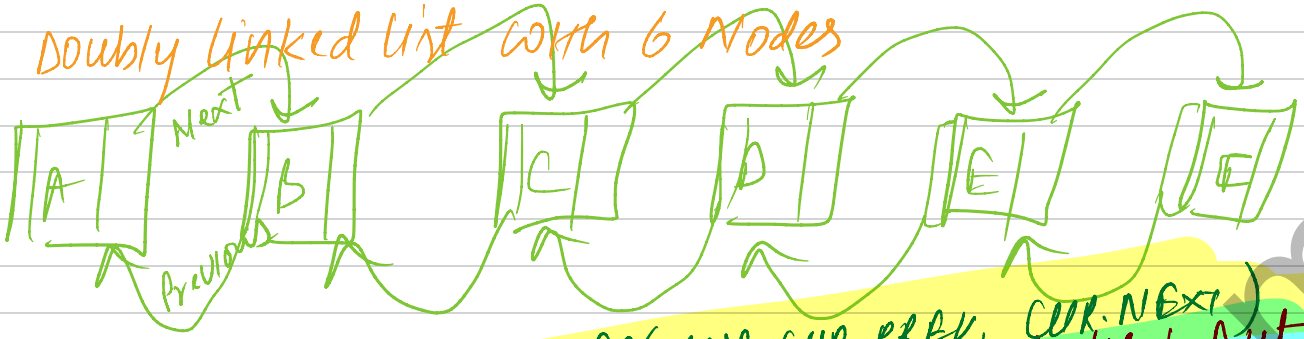
Also known as (ISOLATE CONSTRUCT)



www.pushpsood.com

Object based Isolation (MONITORS)

Doubly Linked list with 6 Nodes



DELETE (CUR)
{ ISOLATED(CUR, CUR.PREV, CUR.NEXT) → WORKS! But this becomes sequential
ISOLATED(CUR) → OVERKILL
CUR.PREV.NEXT = CUR.NEXT;
CUR.NEXT.PREV = CUR.PREV;
}

can lead to wrong answer

Execution of Delete function without ISOLATED

T_1 : DELETE(B) → (A, B, C)

T_2 : D(C) → (B, C, D)

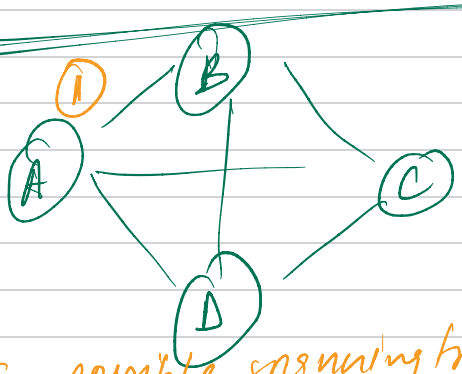
T_3 : D(E) → (D, E, F)

can be ran parallelly

No objects in common so they will run parallel

www.pushpsood.com

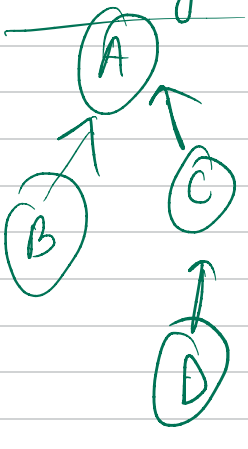
SPANNING TREE OF AN UNDIRECTED GRAPH



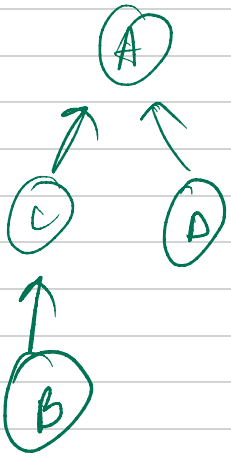
(1)

(2) Two possible spanning tree

Possibility 1



Possibility 2



(3)

```
COMPUTE(V) {                          → boolean successFlag
  FOR EACH NEIGHBOR C OF V
    CAN LEAD TO Data Race
    S ← MAKE PARENT(V, C)
    IF (S)                            → can be computed in a new thread.
      COMPUTE(C)                        i.e. parallel
  }

MAKE PARENT(V, C) {
  ISOLATED {
    IF (C.PARENT == NULL) {
      C.PARENT = V;
      S = TRUE
    }
    else {
      S = FALSE
    }
  }
  RETURN S;
}
```

This is a famous pattern compare & set

Here multiple threads can be trying to make parent

www.pushpsood.com

Atomic Var & References are higher level than lower level locks

ATOMIC VARIABLES

T0 T1 T2 ... Tk-1

X0 [0 : N-1]

GLOBAL
CUR = 0

Very standard problem

DO {

ISOLATED (CUR) {
J = CUR;
CUR = CUR + 1;
}

IF (J >= N)
BREAK
PROCESS THEM (X[J])

} WHILE (TRUE)

these atomic variables & references will execute code much efficiently

Replace with atomic integer

CUR is ATOMIC INTEGER Now

special H/W can handle this really well

CUR.GETANDADD(1);

Other use cases:

ATOMIC REFERENCES

COMPAREANDSET (expectedVal, NewVal) {

ISOLATED (THIS) {

IF (THIS.VAL == expectedVal) {

THIS.VAL = newVal
RETURN TRUE ;
ELSE { RETURN FALSE }

} }

Special H/W can handle this really well

www.pushpsood.com

Refinement of object based isolation

↳ READ-WRITE ISOLATION

① Collection C

(option 1) Isolated

$V_1 \leftarrow C.get(K_1)$ ISOLATED (READ(C))

$C.PUT(K_2, V_2)$ Isolated

Fixed the issue but gets can run concurrently

(option 2)

Concurrency

ISOLATE (WRITE(C))

② All K's are distinct keys

T_1	T_2	T_3
① $C.GET(K_1)$	③ $C.GET(K_2)$	⑤ $C.PUT(K_0, V_0)$
② $C.GET(K_2)$	④ (K_4)	
$C.GET(K_3)$	(K_6)	

$T_1, 2, 3, 4$ are parallel

5 will make sure it's isolated

And then other gets will get executed

Can further Refine using R-W Isolation

Write Mode

Access to CUR is in read mode

```
DELETE(CUR) {
    ISOLATED(CUR.NEXT, CUR.PREV, CUR) {
        CUR.PRV.NEXT = CUR.NEXT;
        CUR.NEX.PREV = CUR.PRV;
    }
}
```

www.pushpsood.com

ACTORS (Even higher level of concurrency)

Implementation of atomic int getAndAdd to show issue with CS & Isolation!

```
GETANDADD(DELTA) {
    I = CUR
    CUR += DELTA
    RETURN I;
}
```

isolated → ensures concurrency

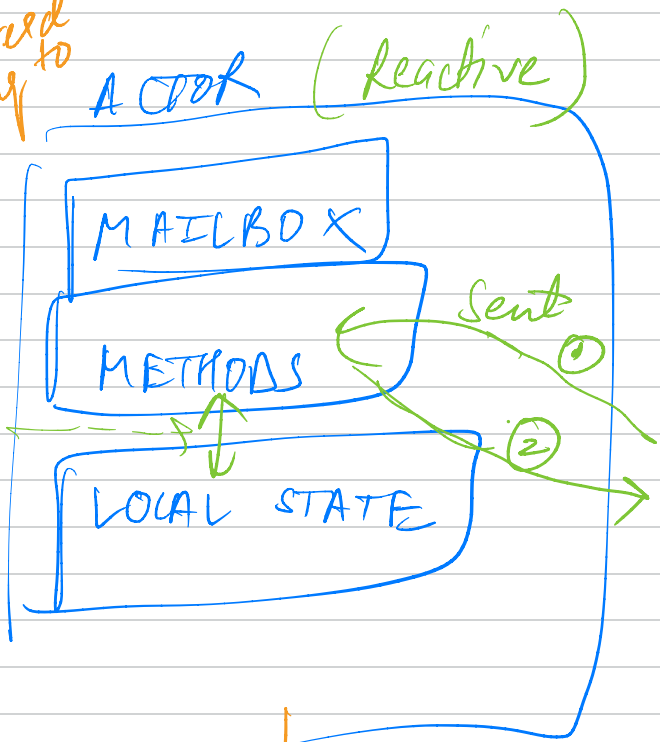
if we forget to put isolation on CUR in foo function, leads to bug!

```
foo() {
    CUR = ....
}
```

→ if isolation is missed this can put bugs, how to put this at a global level

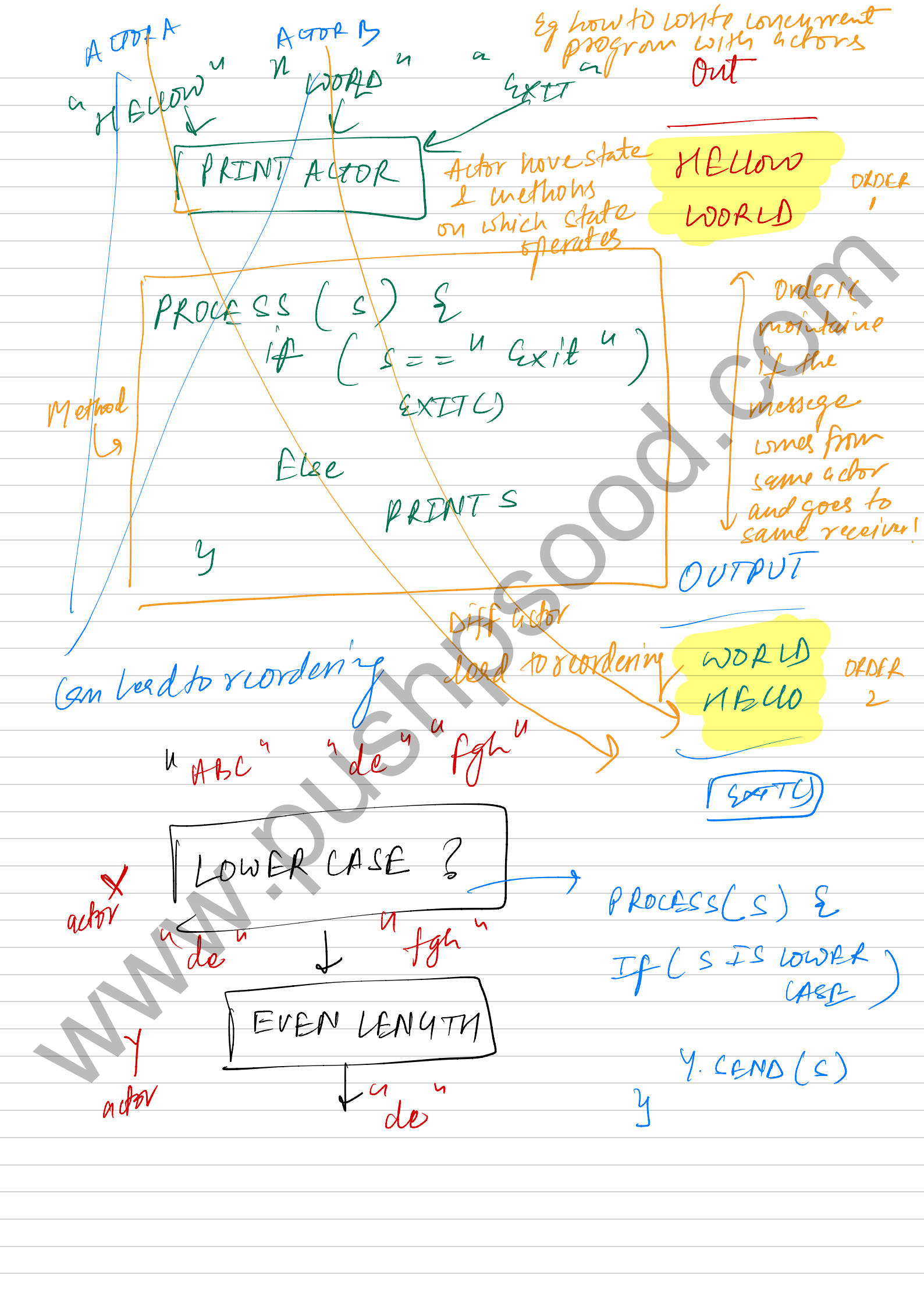
GETANADD()

No way any other thread can access the val directly, it can interact with owner of CUR value through sending messages



This combination is called actor

www.pushpsood.com

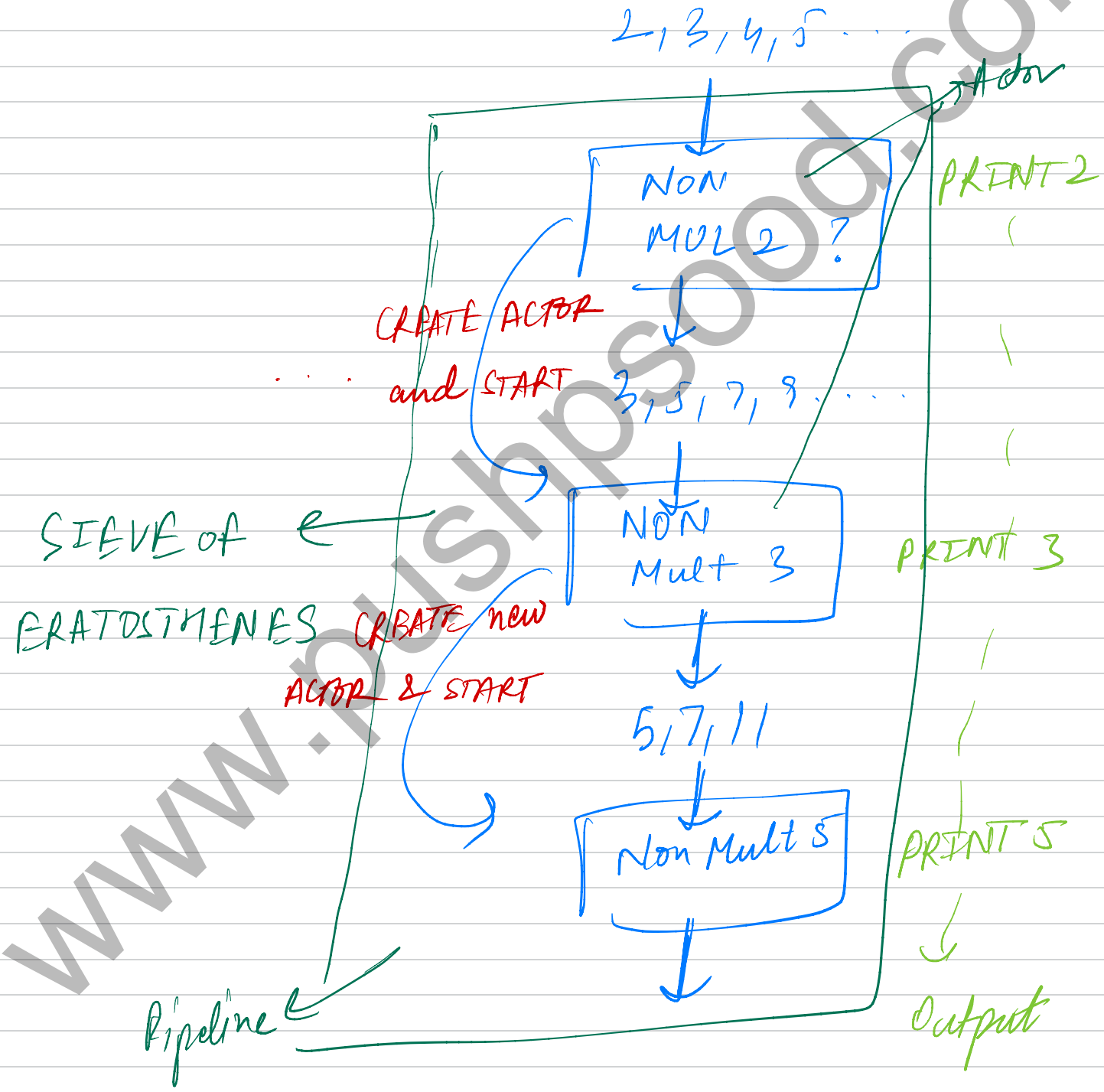


www.pushpsood.com

Generating prime numbers <= n using Actors

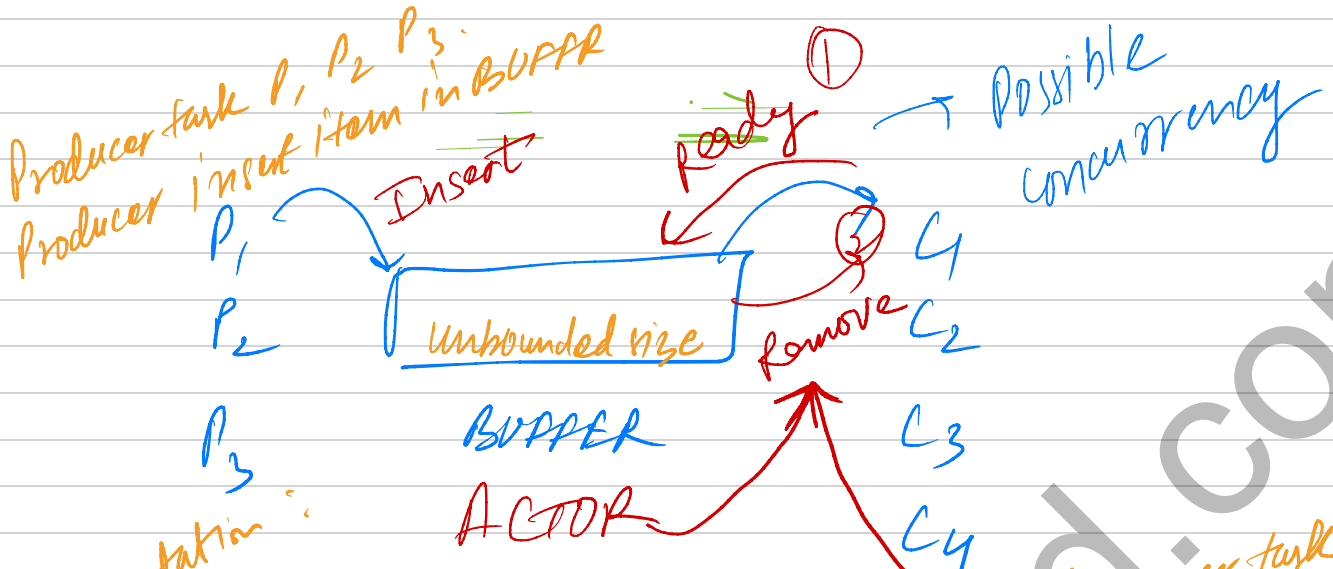
Normal Version: 2, 3, 4, 5, ... N → [Actor] → 2, 3, 5, ...

Optimized Greek algorithm : SIEVE OF ERATOSTHENES



www.pushpsood.com

PRODUCER CONSUMER PROBLEM (classic concurrent programming problem)



Implementation:

CONSUMER THREAD

```
while ( BUFFER EMPTY )
{ } // Wait
PROCESS ITEM REMOVED
FROM BUFFER
```

Consumer task C1, C2...
Consumer returns item

CONCURRENCY AT LOWER LEVEL

V.S.

CONCURRENCY AT UPPER LEVEL

CONSUMER ACTOR

```
PROCESS( S ) {
PROCESS ITEMS
BUFFER.SEND ( "READY" )
}
```

NEAT!

Will need a lot of low level logic like isolation, wait & synchronize etc

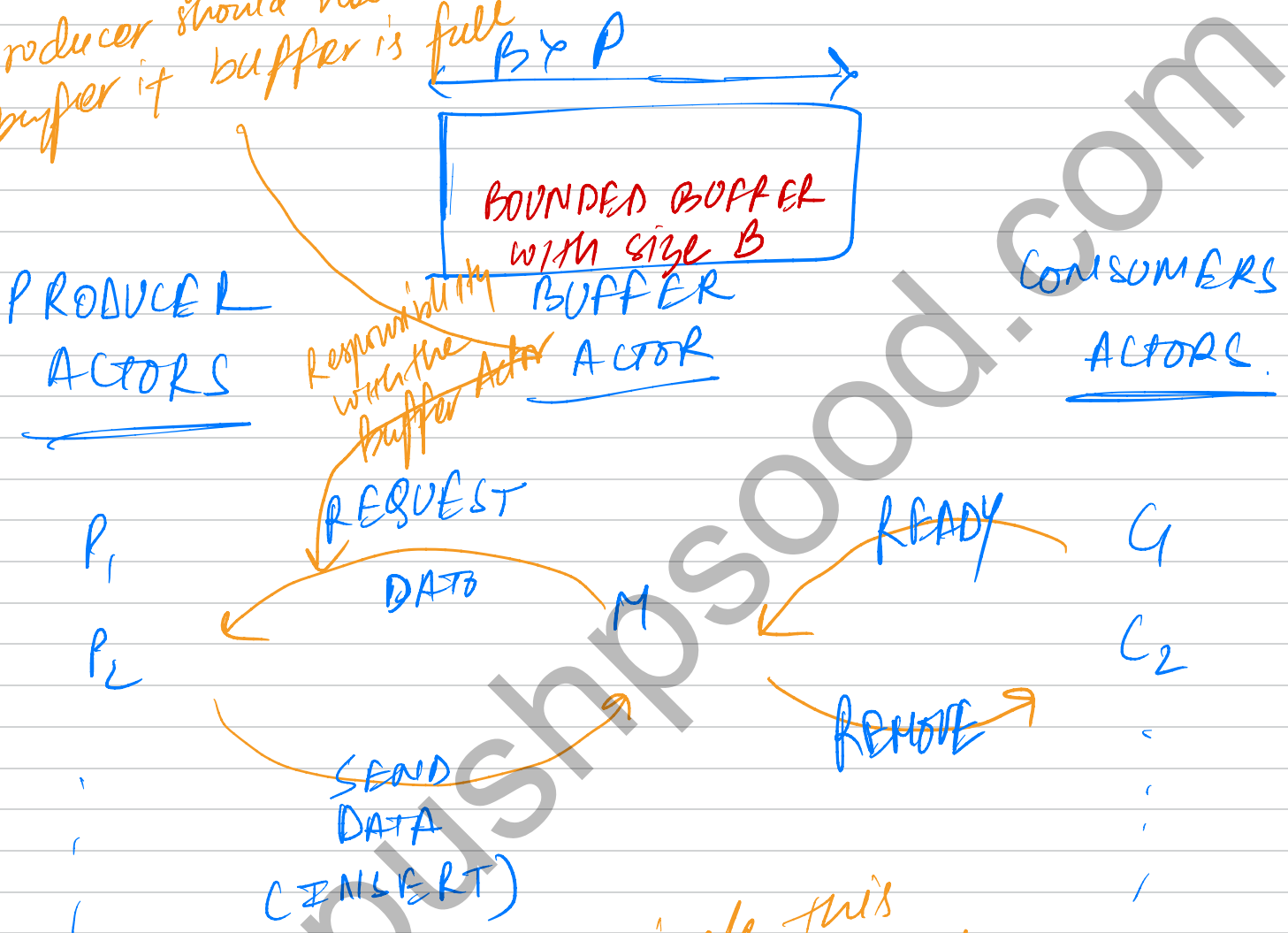
Everything participating in concurrent interaction should be an actor shown in RED

www.pushpsood.com

BOUNDED BUFFER PRODUCER CONSUMER

(Realistic version of producer consumer example)

Main challenge with bounde buffer is producer should not add item to buffer if buffer is full



Though this looks so simple this problem has been studied for decades in advance OS classes and there we use primitives such as semaphores

www.pushpsood.com

OPTIMISTIC CONCURRENCY

Advance concept to create concurrent ds by experts

Normal Implementation

```
INTEGER {
    get()
    set()
    getANDADD(delta) {
        cur = this.get();
        NEXT = cur + delta
        this.set(NEXT);
        return cur;
    }
}
```

Multiple threads calling get and add will lead to data race

Expert Implementation

```
ATOMIC INTEGER {
  GET()
ATOMIC INTEGER {
  SET()
  GET()
  COMPAREANDSET()
    GETANDADD(DELTA) {
COMPAREANDSET {
GETANDADD(DELTA) {
  WHILE(TRUE) {
    cur = this.Get();
    WHILE(TRUE) {
      cur = this.Get();
      Next = cur + DELTA
      IF (this.COMPAREANDSET(cur, NEXT);) {
      Next = cur + DELTA
      IF (this.COMPAREANDSET(cur, NEXT);) {
      }
      RETURN cur;
    }
  }
}
```

① If multiple threads were updating the value another thread can update the integer value in between then the cur value will be stale

② Returns a boolean value to indicate success

③ Each thread will repitedly run this loop till successful

④ The code ensures no DEADLOCK and LIVELOCK

www.pushpsood.com

CONCURRENT QUEUE

(Popular concurrent DS)

- Flow black first
- IN BETWEEN

① Implementation:

② In concurrent prog every shared var is potential hazard

{{{{ Multiple set can be enqueue

Queue {

retry loop helps in concurrency

ENQ(X) {

TAIL.NEXT = X;

TAIL = X;

}

④

DEQUEUE() {

if (EMPTY) ....

R = HEAD; {{{

{{{ HEAD = HEAD.NEXT;

Return R

}

③

Multiple thread can access

⑤ Implementation of Enqueue

ENQ(X) {

WHILE(true) {

IF(!TAIL.NEXT.COMPARE AND SET(NULL, X))

{ CONTINUE; }

}

}

usual way of dealing with concurrency is locks & isolation but optimistic concurrency ensures more concurrency

Only succeed when it has correct val of TAIL.NEXT

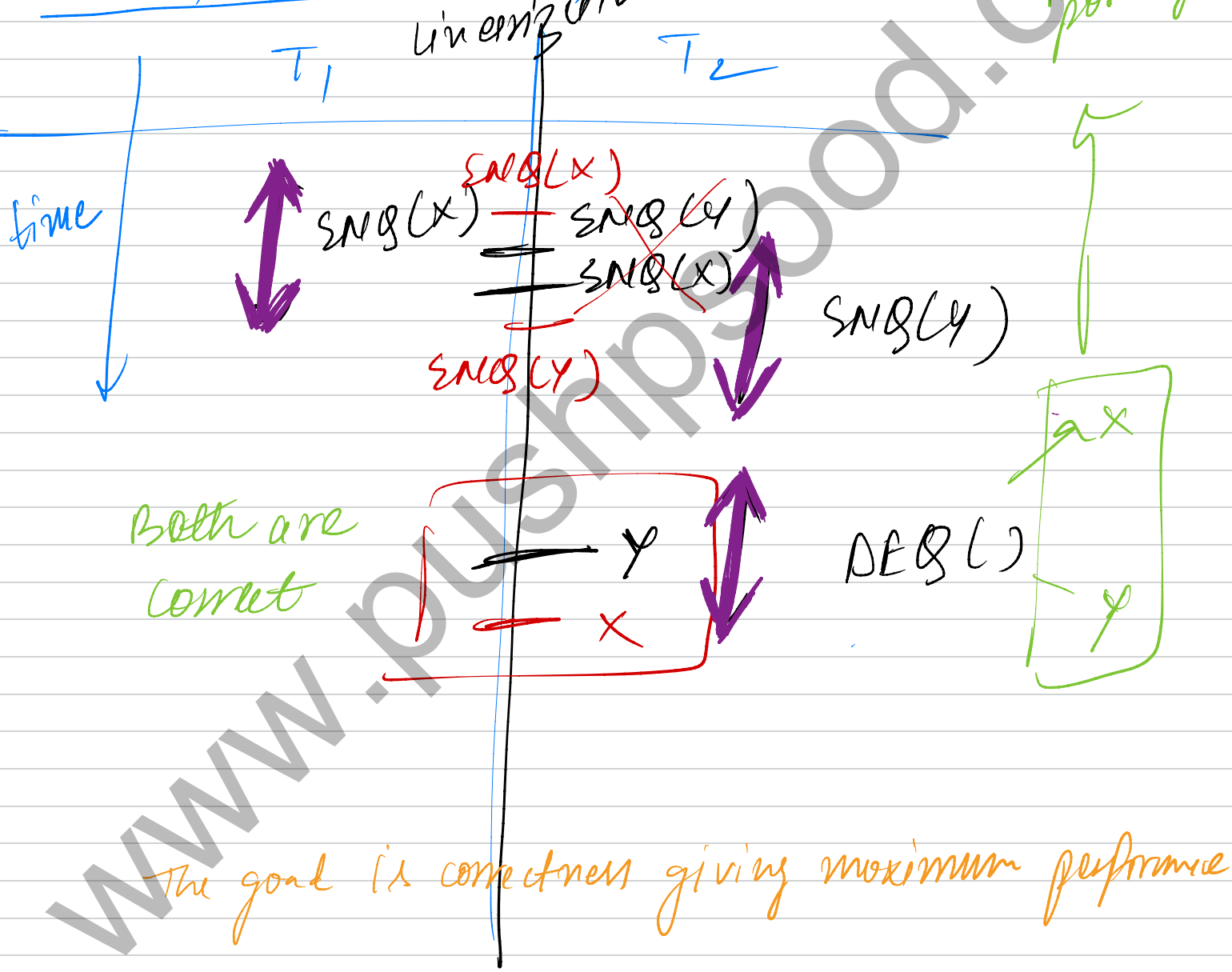
www.pushpsood.com

LINEARIZABILITY (Important property of concurrent object)

Concurrent object is an object which is made "thread safe", i.e. safe for multiple threads to operate on it eg Atomic integer, concurrent queue, concurrent hashmap etc. It has some sets of of [OP1, OP2 OP3

continue

[OP1, OP2, OP3, ...



The goal is correctness giving maximum performance

www.pushpsood.com

CONCURRENT HASHMAP (one of many implementation of Java concurrent DS)

- CON HASHMAP
- CON LINKED LIST
- CON SKIP LIST SET

Implements operations of normal map with some additional functions()

extending to concurrent world

Linearizable

- GET (KEY)
- PUT (KEY, value)
- PUTIFABSENT (K, V)

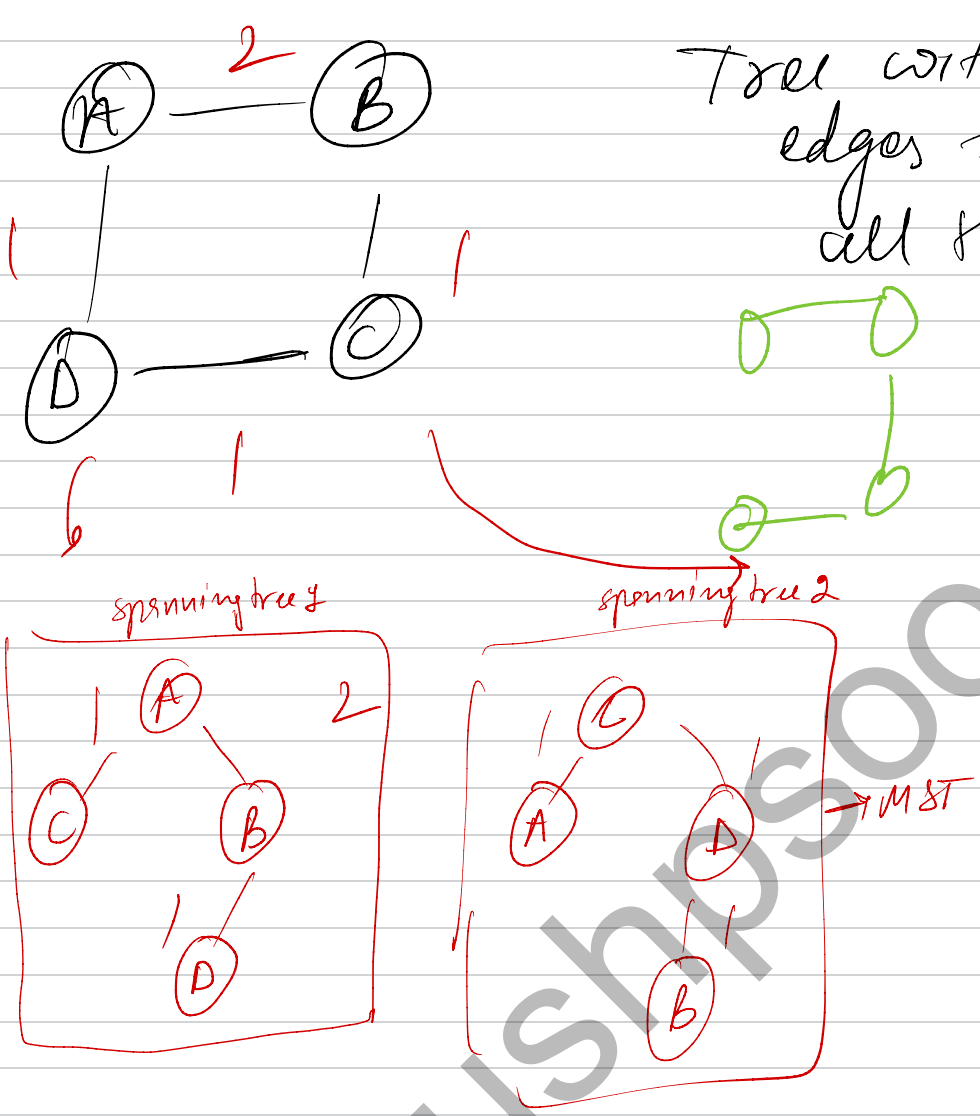
- Clear()
- PutAll()

Not linearizable

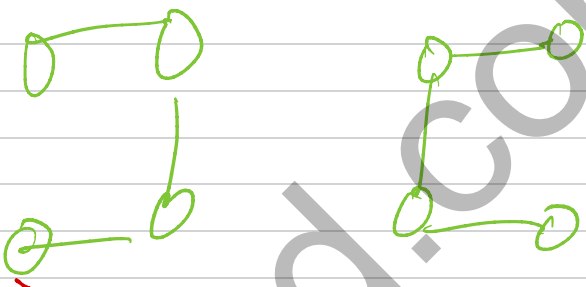
Don't call clear() in parallel with a put()

www.pushpsood.com

Minimum Spanning Tree



Tree with subset of edges that connects all the nodes



minimum spanning tree Algo

L ← Vertices → Concurrent Linked Queue

```
WHILE (SIZE(L) > 1) {
    N1 ← REMOVE(L)
    if(! TRYLOCK(N1)) continue
    E ← GETMINEDGE(N1)
    if(! trylock(N2)) { fixup continue; }
    N2 ← GETNEIGHBOR(N1, E)
    N3 ← MERGE(N1, N2)
    REMOVE(N2), INSERT(N3)
}
```

www.bushpsood.com

Distributed Programming in Java

www.pushpsood.com

Map Reduce

Big Data

HOW BIG DATA WORKS

Abstract the data into Key Value Pairs (K, V)

MAP

for (KEY, VALUE) pair (K_A, V_A)

Map f

Output

Key/Value pair 1: (K_A, V_A), (K_{A_1}, V_{A_1}), (K_{A_2}, V_{A_2})

KVP2: K_B, V_B & K_{B_1}, V_{B_1}

grouping goes through all the (K, V) pairs from map fun and puts together all the same keys

GROUP

Group with same key

if $K_{A_1} = K_{B_1} = K$

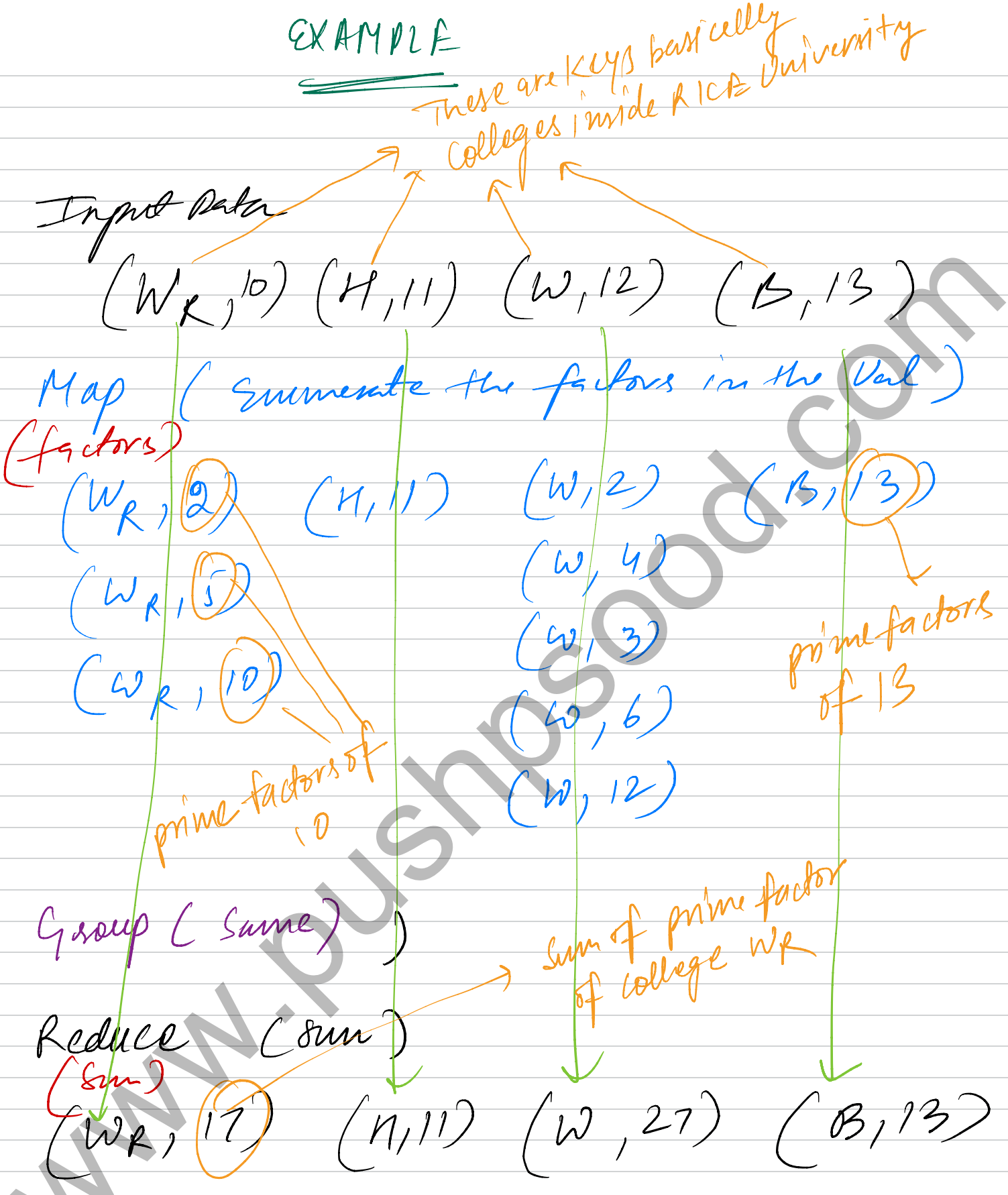
$(K, (V_{A_1}, V_{B_1}))$

REDUCE

$(K, g(V_{A_1}, V_{B_1}))$

www.pushpsood.com

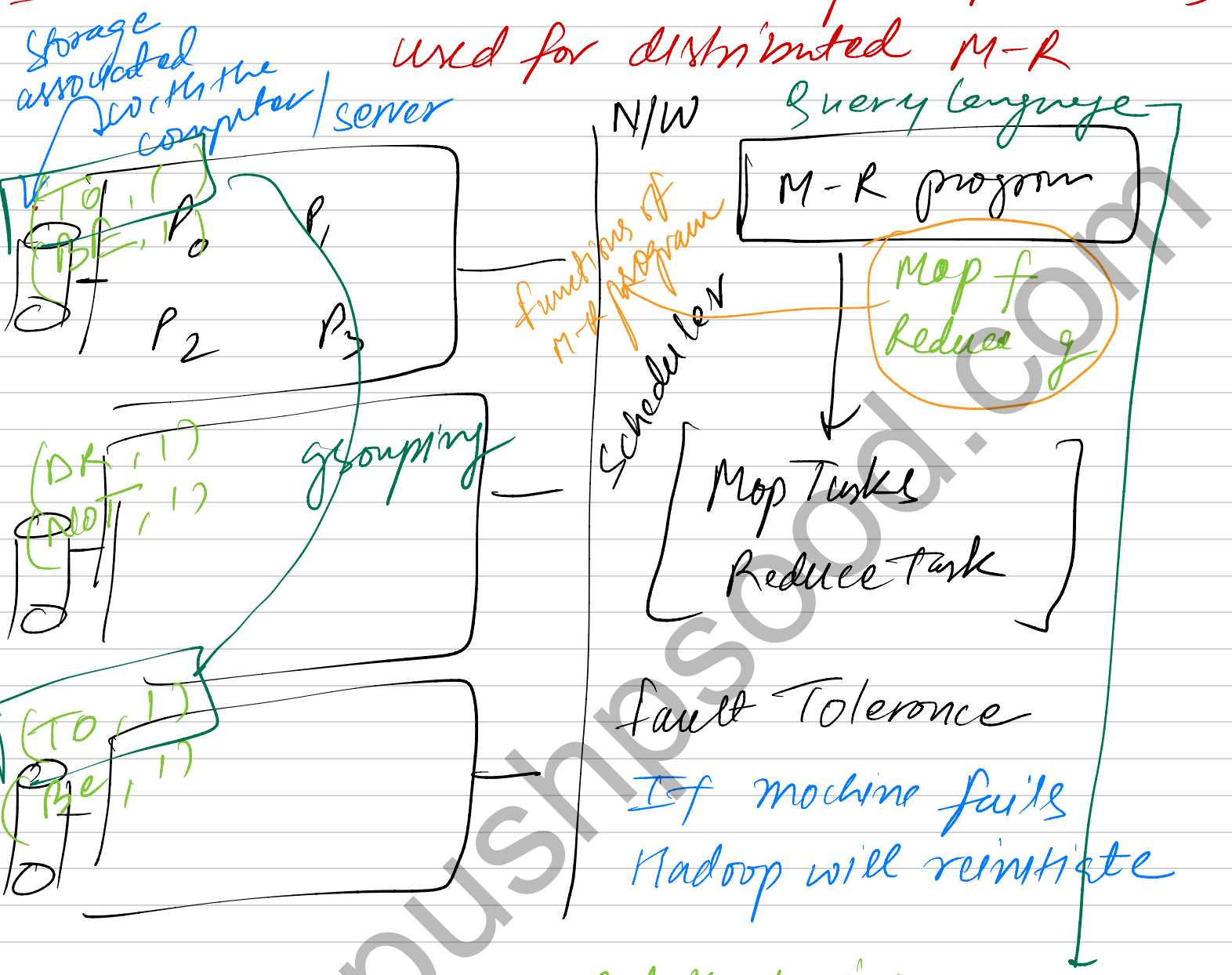
EXAMPLE



www.pushpsood.com

Apache Hadoop framework (Implementation of Map & Reduce)

used for distributed M-R



Word Count

(To, 2)
(Be, 2)
(OR, 2)
(Not, 2)

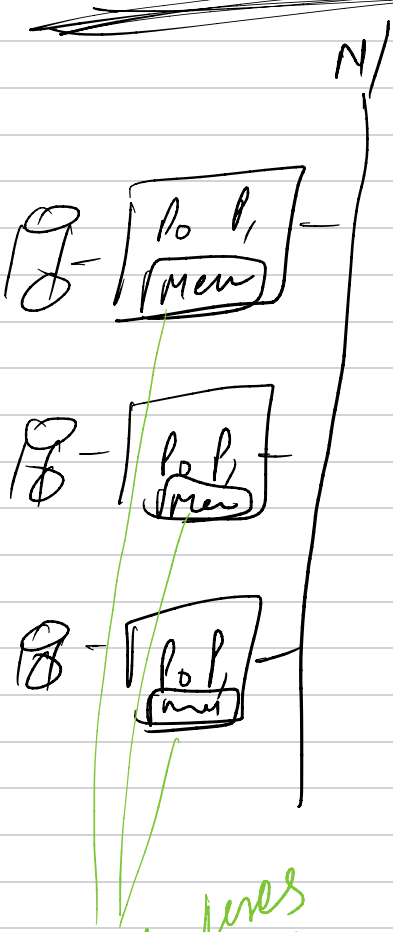
Input K-V pairs
↓
Map, f
↓
Intermidiate (K, V) pairs
↓
grouping
↓
Reduce, g
↓
output (K, V' pairs)

(Hive, Pig etc)

SQL type to automatically generate Map Reduce queries

www.pushpsood.com

Apache Spark Project (Modern version of Hadoop)



Changes while moving from HADOOP → SPARK

① K-V pairs | Resilient Distributed Dataset (RDD)

↳ Lets us store data in other forms including but not limited to K-V store

② main op is Map-Reduce | More general O/P similar to Java stream

Intermediate O/P — transform, map, filter, join

terminal O/P — Actions (Reduce, collect ....)

Spark uses in memory for the O/P & keep intermediate that we keep compute memory value inside the i.e. we are able to reuse the memory

Key innovation in Spark are

- Lazy Evaluation
- Caching (memory)

Caching gives significant improvement but we should run the application on computers which have enough memory to hold the data & in cloud nodes with large memory are expensive than nodes with low memory

www.pushpsood.com

EXAMPLE OF SPARK M/R FRAMEWORK IN JAVA

WORD COUNT (using spark API in Java)

① SparkContext (sc) = new JavaSparkContext();

② Input = SC.TextFile("....")

③ Words = Input.flatMap(line → line.split(" "));

④ Pairs = words.mapToPair(
(x) → New+tuple2(x, 1))

⑤ counts = Pairs.ReducebyKey
((x,y) → x+y)

www.pushpsood.com

Eg of Hadoop M/R Homework

Document 1, 2 .. n

(find the most similar doc across large no of doc)

TF - IDF Weights for info retrival and document

Document D_0

$D_0 = \{ TERMS \}$

$D_1, D_2 \ldots D_n$

NO of time term i occurs in Document 2

DF

No of documents the term$_x$ occurs in

Term 1	Tf_{11}	Tf_{12}	...
Term 2	Tf_{21}	Tf_{22}	...
⋮	⋮	⋮	

Words like "the" appears a lot of time

IDF = Inverse Document freq = N/Df

(N/Df for commonly occuring word 'the' will be 1)

$$Weight(Term_i, Doc_j) = Tf_{(i,j)} \times \log\left(\frac{N}{Df_i}\right)$$

eg

The idea is to use this as weights to give more prominence to word that occur infrequently.

www.pushpsood.com

TF(i,j) → computing total frequency

Input : (DOC, {TERMS})

↓ Map

((DOC, TERM), 1)

↓ Reduce

((DOC, TERM), TF)

→ Total frequency the term appears in the doc

↓ Map

(TERM, 1) → for every occurence of term in the document when TF > 0

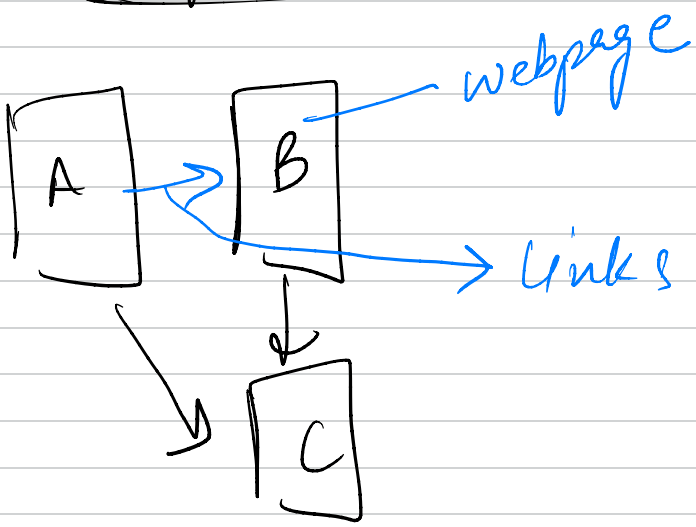
inv → when TF > 0

↓ Reduce

(TERM, DF)

www.pushpsood.com

Page Rank Example



$$Rank(B) = \sum_{A \in SRC(B)} \frac{RANK(A)}{DEST\_COUNT(A)}$$

FOR (ITER . ...) {

1) ~~FOR~~ EACH PAGE A → B

CONTRIBS(B) += $\frac{RANK(A)}{DEST\_COUNT(A)}$

weighting factor

2) ~~FOR EACH~~ PAGE B

RANK(B) = 0.15 + 0.85 * CONTRIBS(B)

BETTER SUITED FOR SPARK?

Spark is good for iterative programming because of cache!

www.pushpsood.com

CODE

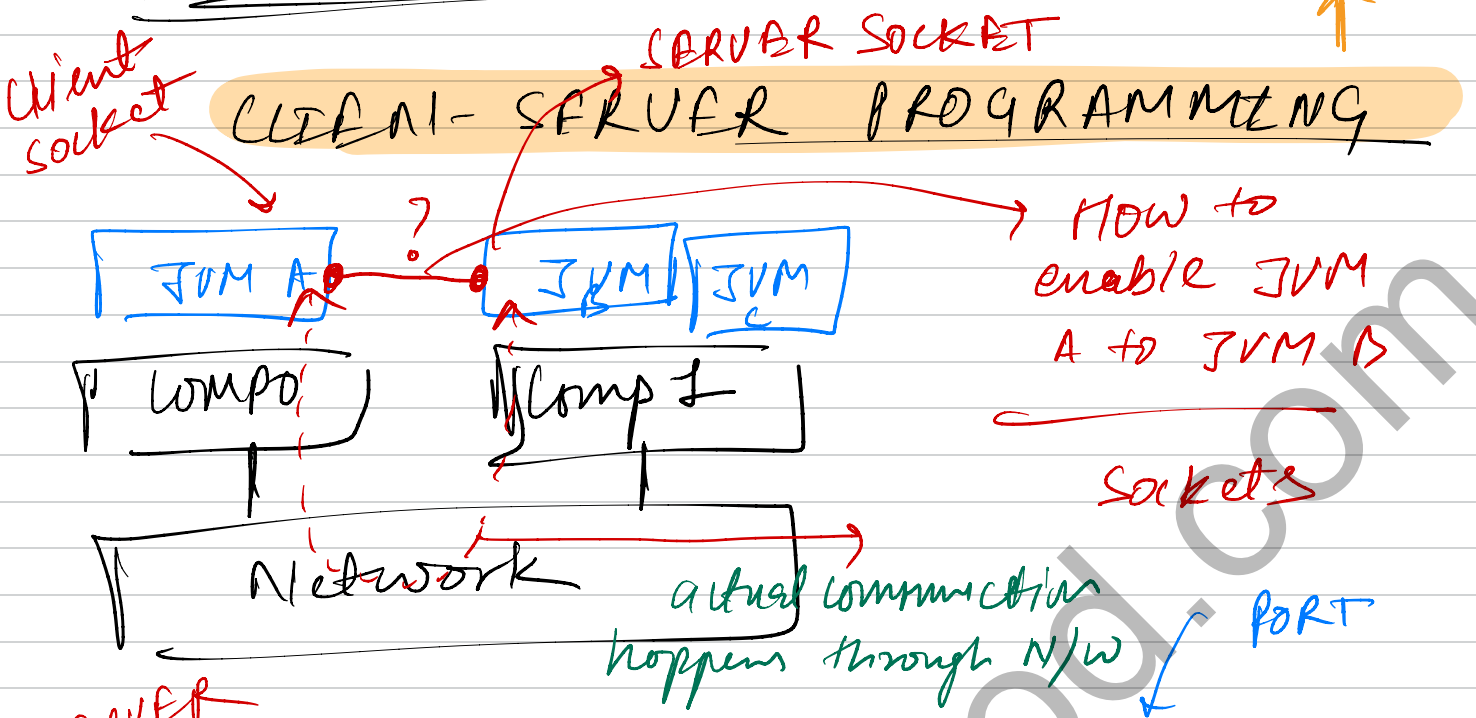
```
FOR (ITER ...) {
  FOR EACH PAGE A → B {
    CONTRIBS += LINKS
                .JOIN(RANKS)
                .VALUE
                .FLATMAPTOPAIR(
                  DEST → (DEST,
                    RANK(SRC)/DEST-COUNT));
  }

  FOREACH PAGE B {
    RANKS = CONTRIBS.
            REDUCEBYKEY (NEW SUM())
            MAPVALUES (
              SUM → 0.15 + 0.85 * SUM);
  }
}
```

www.pushpsood.com

INTRODUCTION TO SOCKETS

lower level concept than distributed comp

CLIENT-SERVER PROGRAMMING



bSocket = new ServerSocket();

- CREATE
- BIND
- PORT

a = bSocket.accept()

a.getInputStream() // Reads

a.getOutputStream() // Writes

CLIENT

aSocket = new Socket()

aSocket.getInputStream() // Read

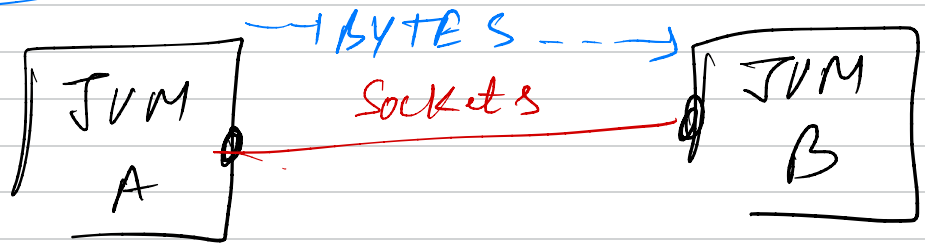
aSocket.getOutputStream() // Write

www.pushpsood.com

SERIALIZATION / DE SERIALIZATION

Objects
A_0, A_1, A_2

B_0, B_1, B_2

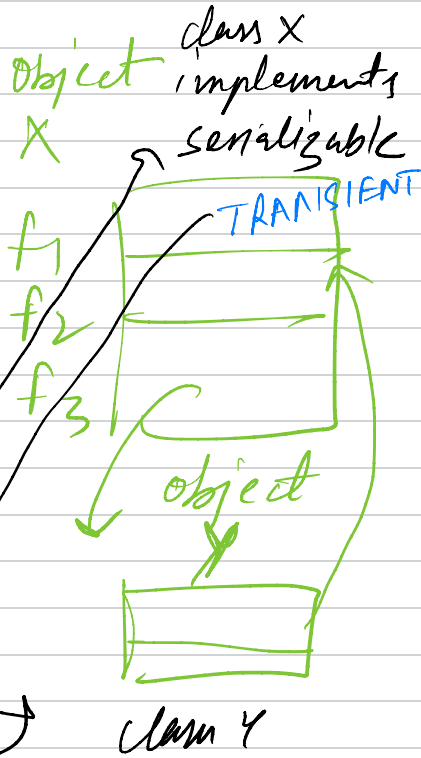


Objects to Bytes
Serialization

Bytes to Objects
Deserialization

Choices for DE | Serialization

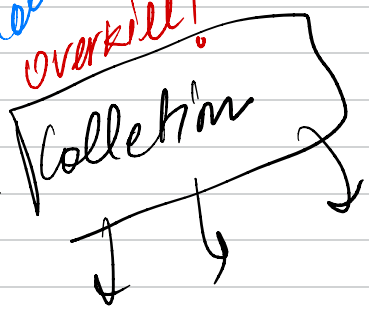
1. Custom SER | DESR (Complicated! when nested objects)
2. Use XML (Heavy Weight)
3. Using Java Serialization & Deserialization (It is Robust)



4. Interface Definition Language (IDL)
   Eg. PROTOCOL BUFFER
   (C++, JAVA etc)
   Language Agnostic
   PROTO FILE needs to be created!

if f_3 is transient its fields will not be copied

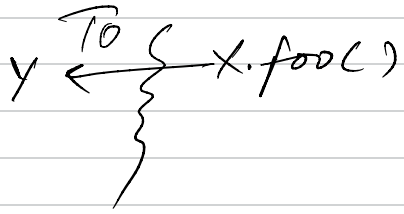
overkill!



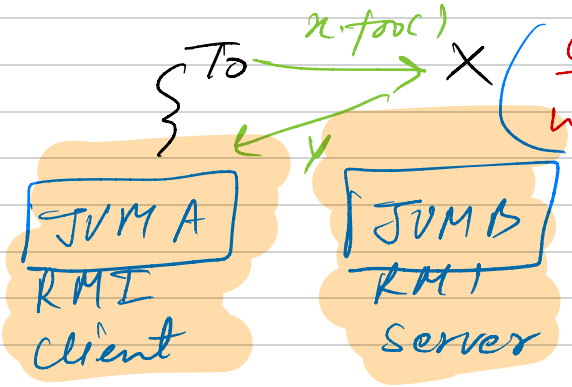
www.pushpsood.com

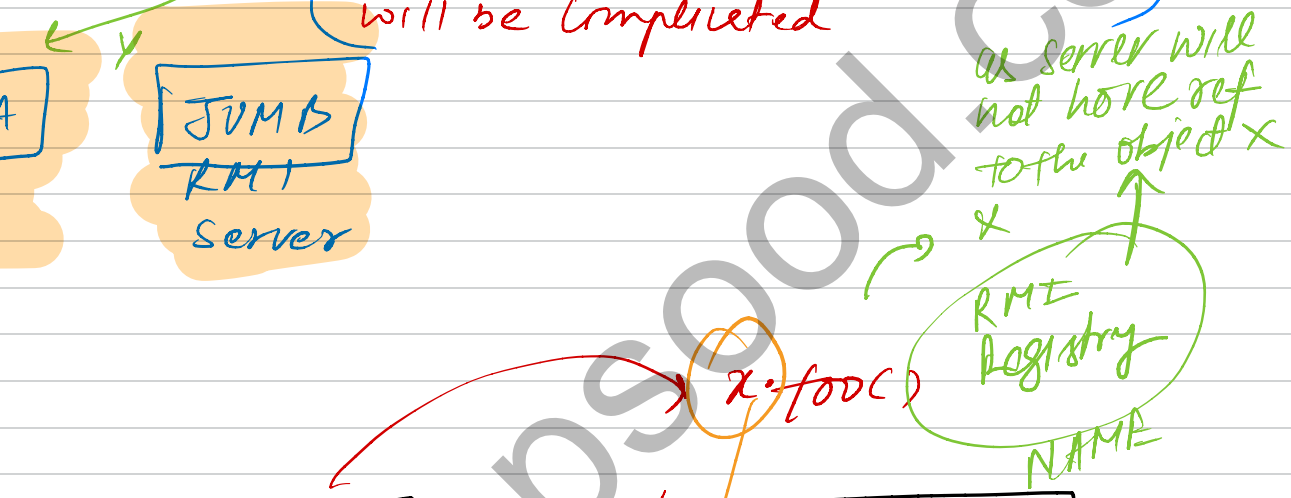
REMOTE METHOD INVOCATION (RMI)

(Higher level concept build on Sockets and Serialization)



what if x.foo() is in a different JVM ? That is when we use RMI





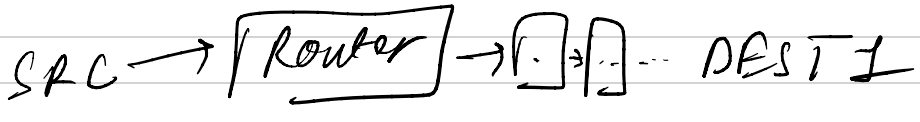
Conditions : X & Y need to be serializable

www.bushpsood.com

MULTICAST SOCKETS (since JDK 1.1)

Usage of multicast: newsfeeds, video conferenc, gaming etc

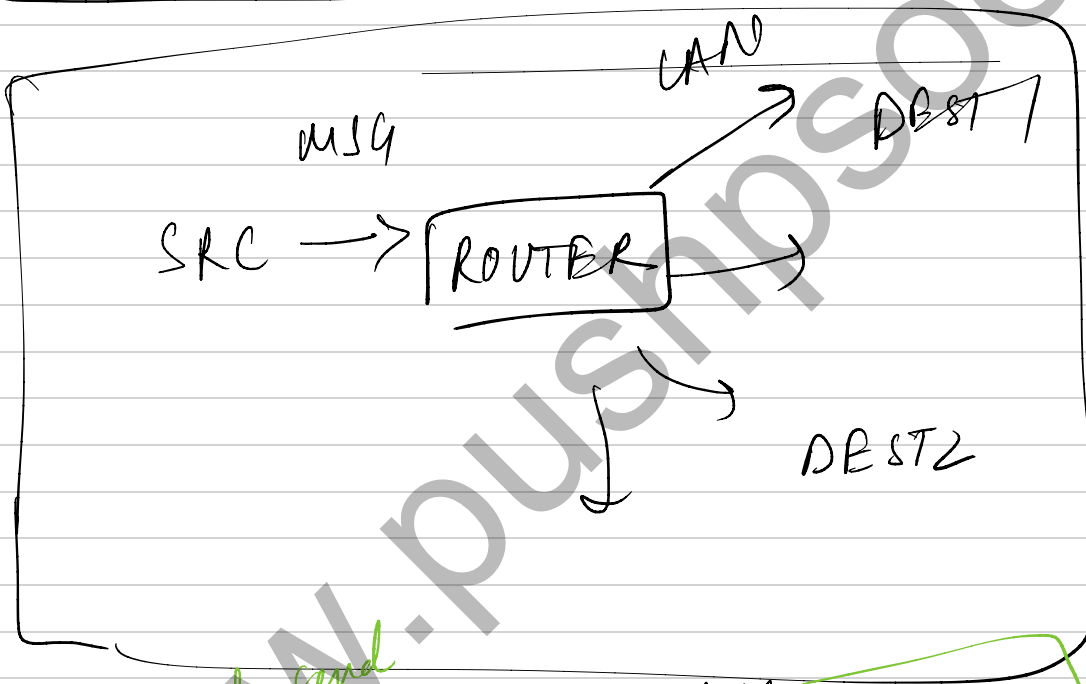
Earlier we studied UNICAST



DEST2

this is unicast Point to Point

Unicast
(one message goes to one destination)



BROADCAST
(one message goes to all destination but local area)

only send to 1,2

MULTI CAST
(one message goes to selected destinations)

www.pushpsood.com

Multicast Socket Application

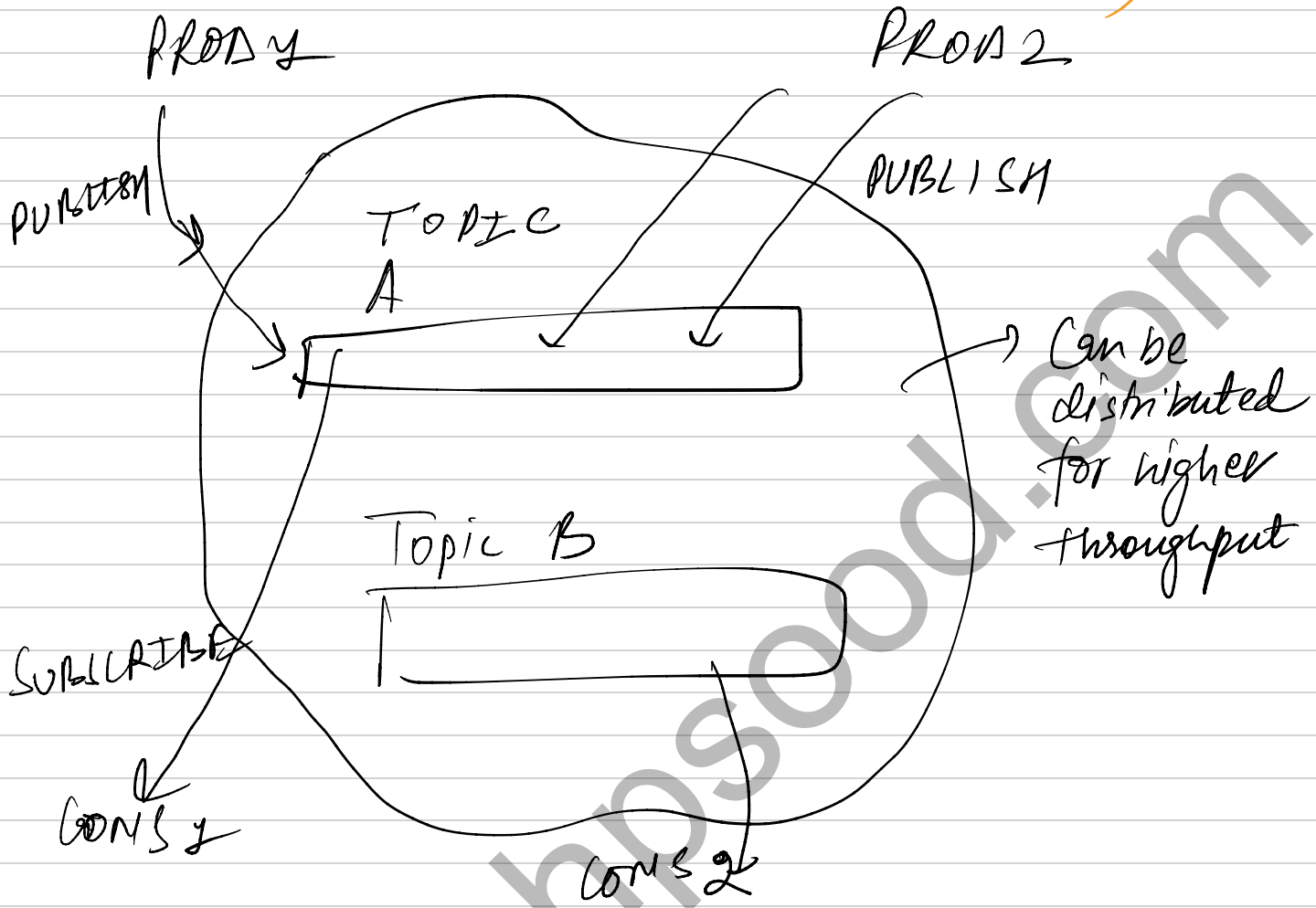
```
S = new multicast socket ( PORT )
S. JOINGROUP ( G );
MSG = New DATAGRAM PACKET (...);
S. SEND( MSG );
MSG2 = NEW DATAGRAM PACKET (...);
S. RECEIVE ( MSG2 )
S. LEAVEGROUP( G );
```

64KB limit

www.pushpsood.com

PUBLISH-SUBSCRIBE

(Higher level pattern to Group communication than MULTICAST)



PRODUCER

```
P = new KafkaProducer(...)
P.SEND(new ProducerRecord
    (topic, key, value))
P.close()
```

CONSUMER

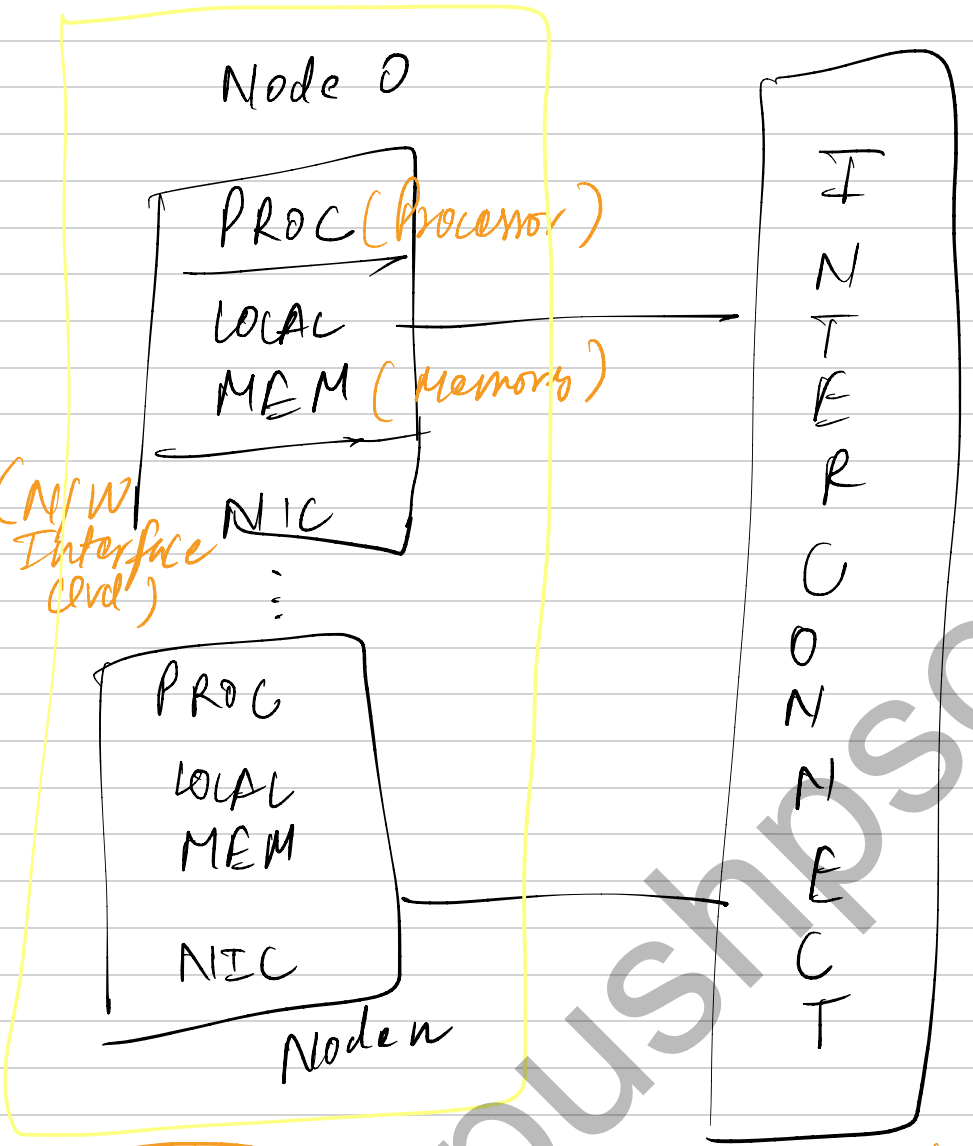
```
C = new KafkaConsumer
    (...)
C.SUBSCRIBE(
    TOPIC1, ...)
RECORD = POLL
    Timeout)
for(R : Records)
R.offset, key, value
```

www.pushpsood.com

How to use collection of computer as single parallel computer

↳ DISTRIBUTED PARALLELISM

CLUSTER (collection of Nodes)



DATA DISTRIBUTION

GLOBAL VIEW: X_{global}

A	B	C	D
0	1	2	3

LOCAL VIEOR

Node 0 : X_{local}

X_L = | A | B |
|---|---|
| 0 | 1 |

Node 1 : X_{local}

X_L = | C | D |
|---|---|

Program for implementation

```
Main()
MPI_INIT()
FOR(I...) {
XL[I] = XL.LENGTH * RANK() + I
}
```

OUTPUT:

NODE 0: XL[0] = 0
XL[1] = 1

NODE 1: XL[0] = 2
XL[1] = 3

Single Program Multiple Data Abstraction (SPMD)

Interface: MPI

www.pushpsood.com

POINT TO POINT COMMUNICATION

(Basic primitive in distributed parallelism)

Core primitives ← (Send & Receive)

INTERCONNECT

RANK 0

S = "ABCD"

SEND S TO RANK 1

Using SPMD Program (Diff things using same program using conditionals)

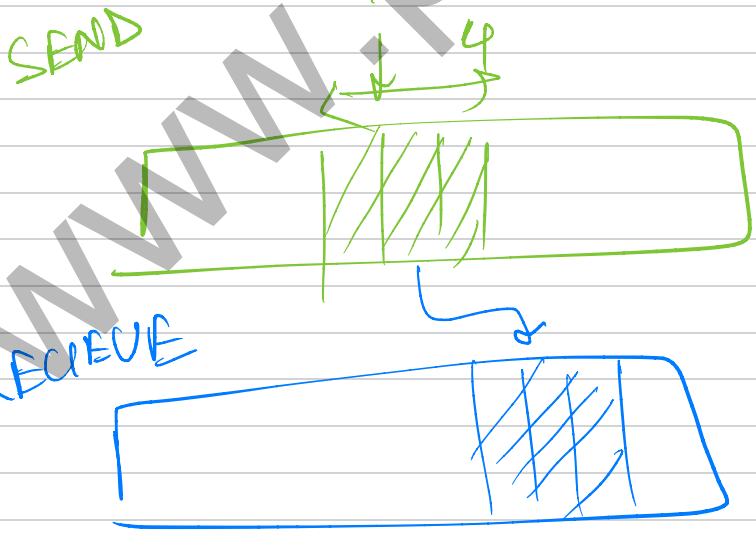
RANK 1

ALLOCATE BUF

RECEIVE INTO BUF FROM RANK 0

PRINT BUF;

```
MAIN() {
MPI.INIT();
if(RANK()==0) {
    S="ABCD"
    MPI.send(S,0,
        S.LENGTH,
        MPI.CHAR, 1, 99)
        DEST      TAG
}
else {
BUF = new CHAR[4];
MPI.receive(
    BUF, 0, BUF.LENGTH,
    MPI.CHAR, 0, 99)
        Sender    TAG
    PRINT BUF
}
MPI.FINALIZE();
}
```



We can serialization & deserialization for objects

www.pushpsood.com

Message Ordering & Deadlock

Time

R0 R1 R2 R3

SEND A to R0

SEND B to R3

RECV A FROM R1

RECV B FROM R2

SAME ORDERING FOR TWO MSGS

- SAME Sender
- SAME Receiver
- SAME Type
- SAME TAG

MPI is blocking operation

R0 R1

BLOCKED send x to R1

BLOCKED send y to R0

Receive y from R1

Receive x from R0

Leads to deadlock!

DEAD LOCK fix

APPROACH 1

SEND → R1	RECV ← R0
REC ← R1	SEND → R0

APPROACH 2

PAIR is method allows 2 way exchange

SENDRECV()	SEND RECV()

MPI have an API that clubs send & receive and takes care of addressing deadlock

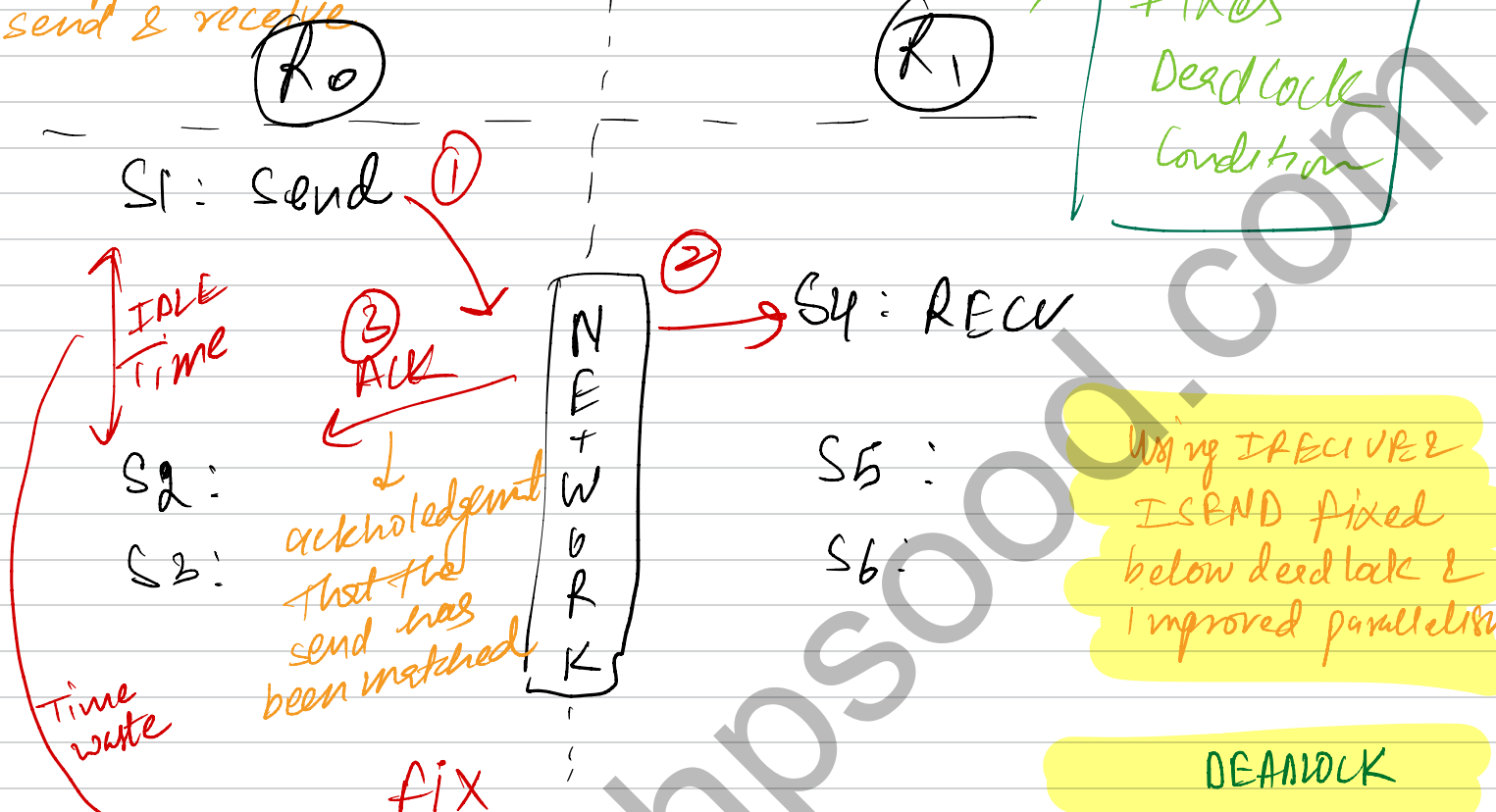
www.pushpsood.com

(Extention to the message passing paradigm)

Non-Blocking Communication → Immediate

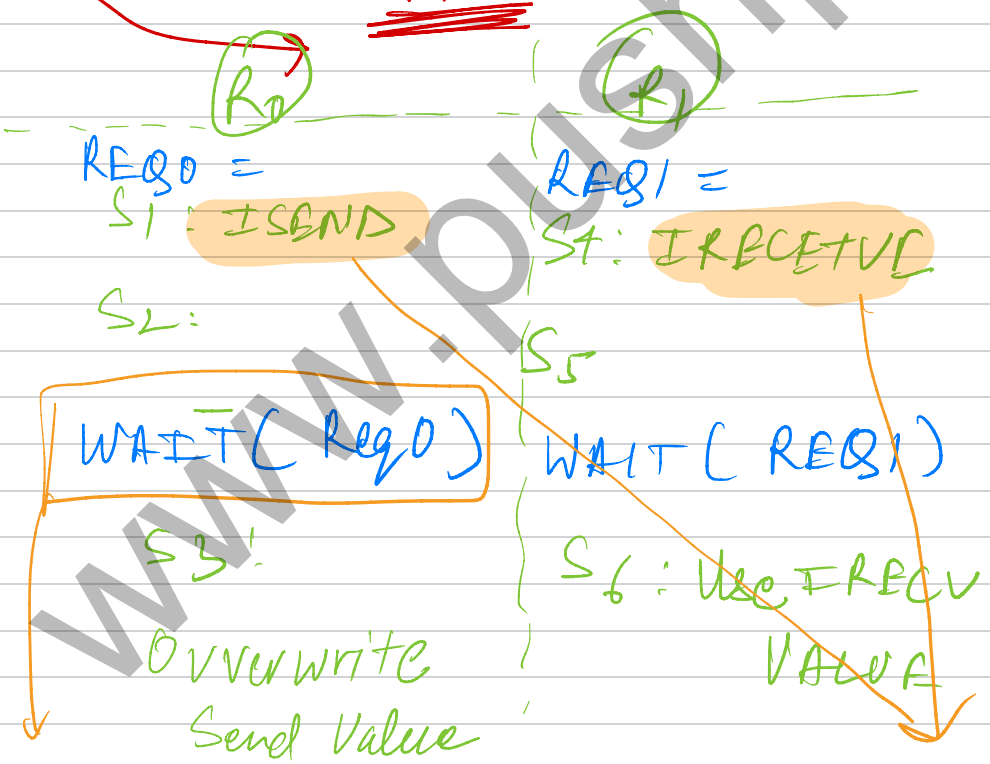
(ISEND, IRECEIVE) ← operations as opposed to send & receive

fixes Deadlock Condition



Using IRECIVE2 ISEND fixed below deadlock & improved parallelism

fix



DEADLOCK

R_0	R_1
ISEND X to R_1	ISEND Y to R_0
REQ 0 = IRECEIVE Y FROM R_1	REQ 1 = IRECEIVE X from R_1
⋮	⋮
WAIT (REQ0)	WAIT (REQ1)
USE Y	USE X

API waiting for multiple or one

WaitAll ( )
Wait Any( )

Non blocking Communication

www.pushpsood.com

Collective Communication

(includes multiple ranks as opposed to P2P communication)

(Multiple Ranks)

→ Can be done using loops (not efficient) fix: collective communication

R_0 : Send X to All Ranks

R_1 :

R_2 : Sum of all y's into z

R_3 :

Implementation:

```
main() {  MPI.INIT();
          INT x[] = ... ;
          if(RANK == 0)  x[0] = 99;
          BCAST(x, 0, 1, MPI.INT, 0)
          PRINT(RANK, x[0]);
          REDUCE[Y, 0, z, 0, 1,
                 MPI.INT, 2)
          If(RANK == 2) {
                 PRINT z[0];
          }
```

Uses full parallelism using competency of the CPU

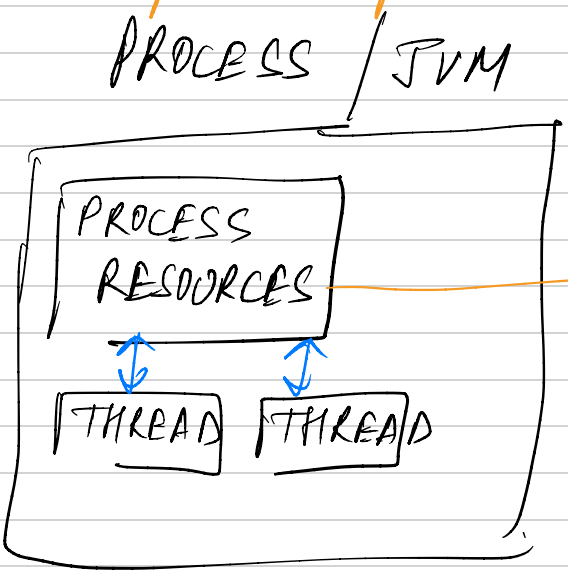
Root

www.pushpsood.com

FUNDAMENTAL BUILDING BLOCK OF DISTRIBUTED COMPUTING

PROCESSES & THREAD

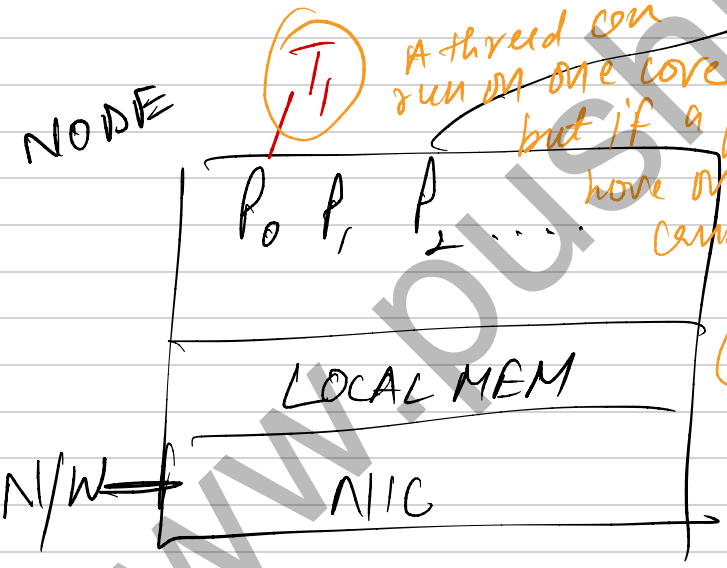
In conter of JAVA, procen is JVM



THREAD CAN RUN ON ONE CORE

within a procen we can create multiple threads

Threads share the procen resources



CORES (16 cores, 4 core, etc)

A thread can run on one core but if a procen only have one thread it cannot using more cores in compute node

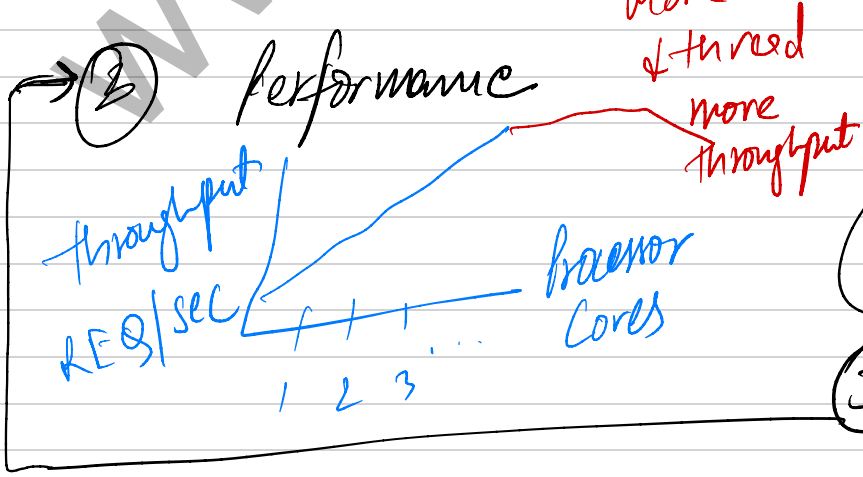
Multiple threads in a procen

① Memory / Resource efficiency due to sharing.

② Responsiveness (specially in N/W delay)

③

② Performance



www.pushpsood.com

Adv Multiple Processes in a Node

① Responsiveness (Jvm Delays)
↓
example is Garbage Collection

② Scalability
Even though we can use multiple threads to increase throughput there is a limitation on throughput to flatten out or even decrease after a certain point

③ Availability / Resilience

www.pushpsood.com

Using process & thread to build something usefull

↳ Multithreaded Servers

PORT, ... etc

Implementing file server:

1) LISTENER = new SERVERSOCKET( )

2) WHILE(TRUE) {

2A) S = LISTENER.ACCEPT

2B) READ FILE REQUEST INFO FROM S.GETINPUTSTREAM();

2C) ACCESS THE FILE

2D) SEND FILE TO S.GETOUTPUT STREAM( )

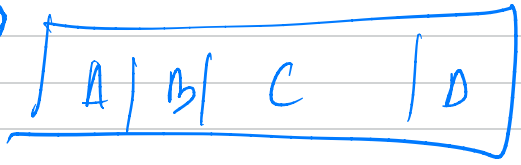
T = new Thread

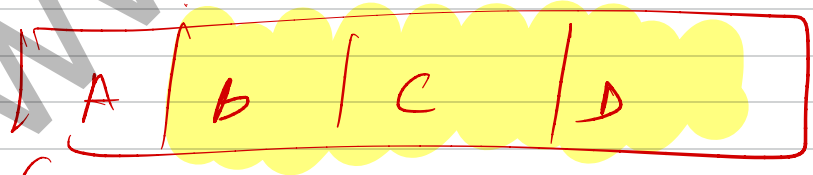
T.start();

We should use threadpool or task (to avoid overhead)

delay because of waiting!

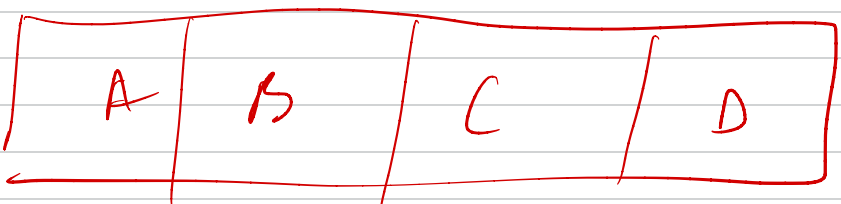
Second request





1st req

No delay.



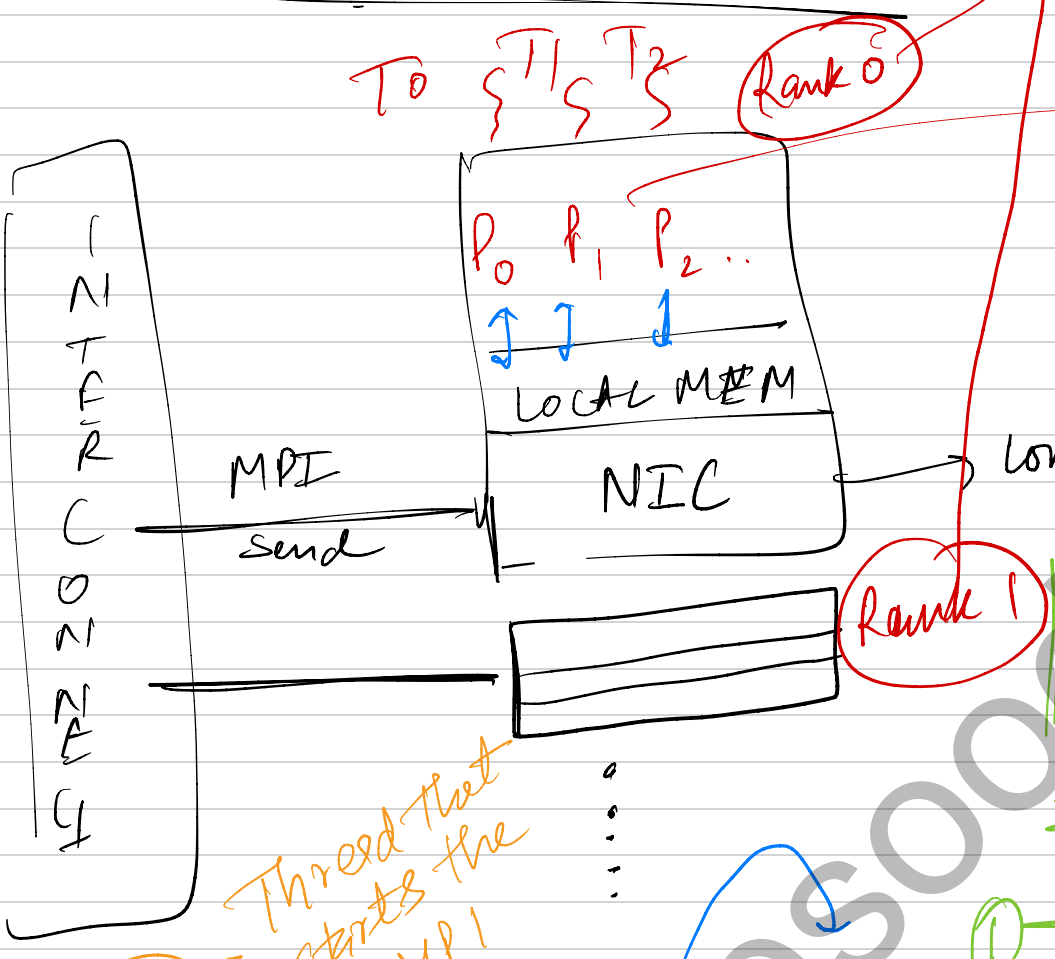
New req

Creating a thread everytime will be an overhead

So, we should be using a thread pool

www.pushpsood.com

MPI + THREADDING



1 MPI process in each Node

Multiple Processor Core

THREADING MODES IN MPI

1 — FUNNELED

All MPI calls are performed by one thread

2 — SERIALIZED

one MPI call at a time

3 — MULTIPLE

Multiple MPI calls at the same time

Can't have 2 threads waiting on same MPI call

Thread that starts the MPI

T0 : MASTER THREAD

T1 T0 T2 ...

Threads for new workers

MPI.INIT()

CREATE_WORKERS_THREADS

MPI.SEND

MPI.RECEIVE

MPI.REDUCE

MPI.FINALIZE()

Possible tasks we can parallelize within the RANK

www.pushpsood.com

DISTRIBUTED ACTORS

ACTOR

Object that has a

- MAILBOX
- LOCAL STATE
- METHODS

(2, 3, 4, 5, ...)

Non Multiples of 2

(3, 5 ...)

Non Multiples of 3

(5 ...)

5

(...)

SIEVE OF ERATOSTHNESS

Node 0

1. CREATE CONFIG FILE ACTOR-HOST
2. CREATE REMOTE ACTOR
3. LOOKUP REMOTE ACTOR BY LOGICAL NAME
4. MESSAGES NEED TO BE SERIALISED

Node 1

Non Multiples of 7

(but extra work has to be done in case of multiple nodes)

www.pushpsood.com

REACTIVE PROGRAMMING

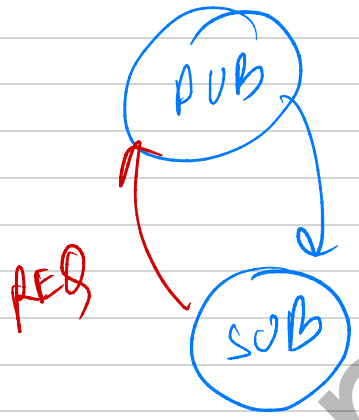
(Implementing distributed service oriented Architecture)

Micro Services

REACTIVE STREAM SPECIFICATION

ASYNCRONOUS EVENTS

FLOW. PUBLISHER
FLOW. SUBSCRIBER
PROCESS SUBSCRIPTION




```
pub = new SubmissionPublisher();
pub.subscribe(S1);
pub.subscribe(S2);
pub.submit(...)
....
```

SUBSCRIBER

```
onSubscribe(SN) {
    SN.REQUEST(100);
}
onNext(ITEM) {
    PROCESS ITEM.
    if(...) SN.REQ(100);
}
```

Req 100 items at a time

www.pushpsood.com