CS & IT ENGINEERING

Compiler Design

Intermediate code and code optimization

Lecture No. 3

By- DEVA Sir

TOPICS TO BE COVERED

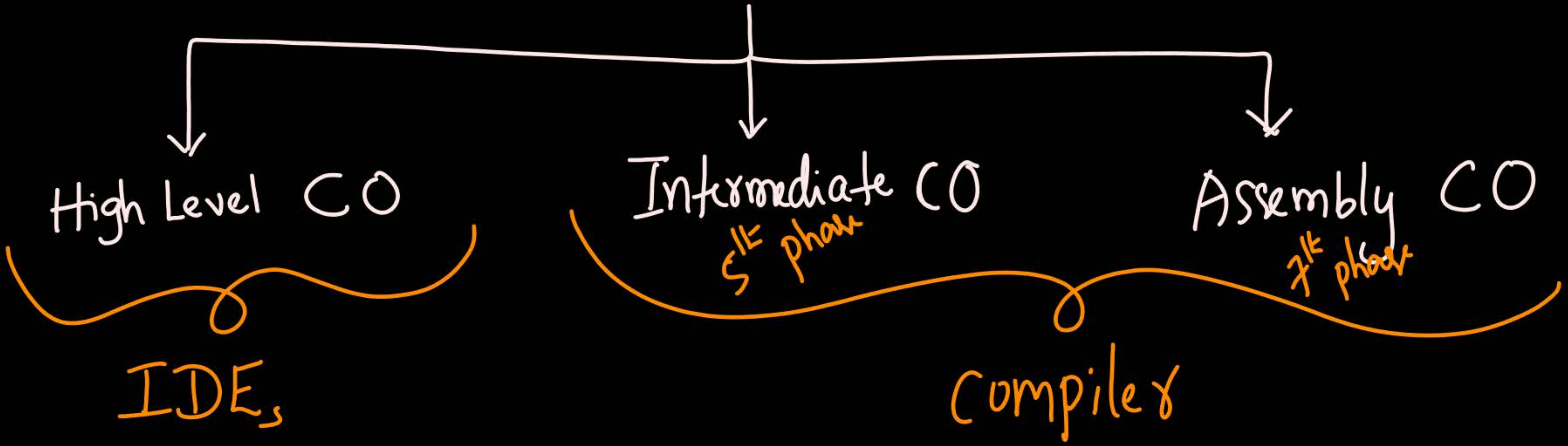
Code Optimization [CO]

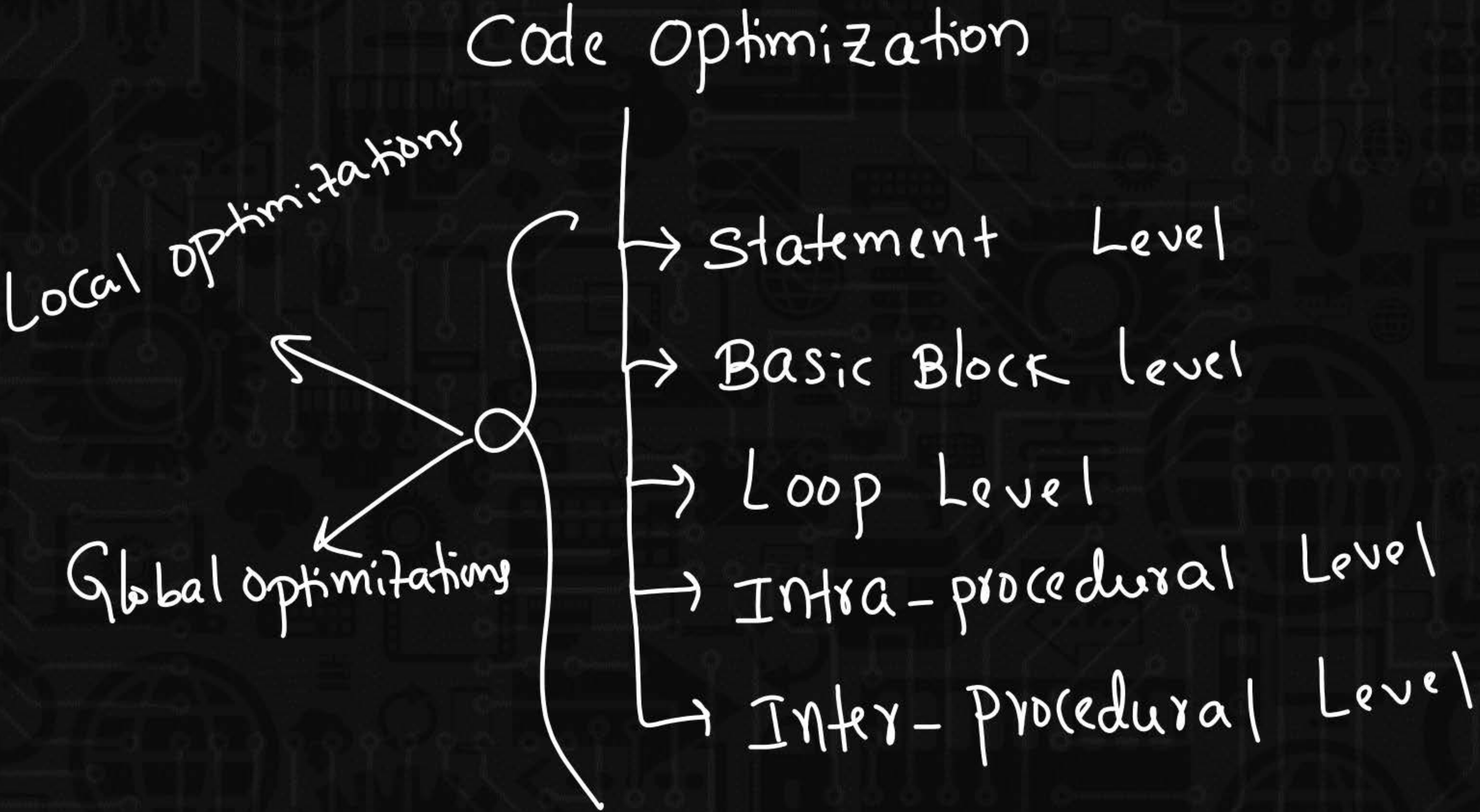
- What is CO ?
- CO Techniques
- Data Flow Analysis
  - Live variable Analysis
  - Reaching Definitim Analysis

Code Optimization

└→ It may save space / Time

Code Optimization





Code Optimization

- Statement Level
- Basic Block level
- Loop Level
- Intra-procedural Level
- Inter-procedural Level

Local optimizations

Global optimizations

Code optimization Techniques :

1. Algebraic Simplifications
   - Constant Folding
   - Identity simplifications
   - Strenglt Reduction
   - Cancellations
2. Copy propogation
   - Constant
   - variable
3. Common Sub expressions elimination [Using DAG]
4. Dead code elimination
5. Loop Optimizations
   - Code Motion
   - Induction variables elimination
   - loop merging
   - loop Unroling

① Constant Folding

$$x = \underbrace{2*3}_{\text{folding}}+y$$

$$\Downarrow$$

$$x = 6+y$$

$$x = 2+3*y$$

$$\Downarrow$$

Constant Folding not possible

② Identity Simplification :

$$x = y + 0 + Z$$

$$\Downarrow \quad \underbrace{y + \overset{\to \text{Identity for } +}{0} = y}_{\text{Identity Law}}$$

$$x = y + Z$$

$$x = \overbrace{y * 1} + Z$$

$$\Downarrow \quad y * \overset{\to \text{Identity for } *}{1} = y$$

$$x = y + Z$$

*** ③ Strength Reduction

↳ "costlier" Instruction can be replaced with "Cheaper"

$$x = \underbrace{a * 2}_{\text{multiplication is costlier}} \quad (\text{costlier})$$

⇓

$$x = \underbrace{a + a}_{\text{Addition}}$$

⇘

$$x = \underbrace{a \ll 1}_{\text{Left Shift}}$$

$x = a * 8$

$\Downarrow$

$x = a \ll 3$

$x = a / 8$

$\Downarrow$

$x = a \gg 3$

④ Cancellation

$$x = \cancel{a} + b*c - \cancel{a}$$

$$\Downarrow$$

$$x = b*c$$

***⑤ Copy propogation

↳ we need DATA Flow Analysis (live variable Analysis)

Constant

$x = \boxed{5}$

$y = x * a$

$z = x + y$

⇓ constant propogation

$$\boxed{\begin{array}{l} y = 5 * a \\ z = 5 + y \end{array}}$$

variable

$x = \boxed{b}$

$y = x * a$

$z = x + y$

⇓ variable propogation

$$\boxed{\begin{array}{l} y = b * a \\ z = b + y \end{array}}$$

*** ⑥ Common Sub-expression Elimination

- → we can use DAG to eliminate common sub-exps
- → We can use Available Expression Analysis (Data Flow Analysis)

$$x = (a+b) * (a+b)$$

$$\Downarrow$$

$$\boxed{\begin{array}{l} t_1 = a+b \\ x = t_1 * t_1 \end{array}}$$

*** 7 Dead Code Elimination

↳ We can do live variable analysis
(Data Flow Analysis)

```
x = a+b
y = a*c
z = x+c
print(z)
```

Dead code

⟹

```
x=a+b
z=x+c
print(z)
```

8 Loop optimizations

i) Code Motion

↳ Find loop invariant code and move outside loop

$n=0 \Rightarrow 2$ statements

$n=1 \Rightarrow 6$ "

$n=2 \Rightarrow 10$ "

⋮

$\boxed{4n+2 \text{ statements}}$

```
for ( i=0; i<n; i++)
{
    x = y;        loop Invariant Code
    a = a+i;
}
```

⇓

```
if(n>0)
x = y;
for(i=0;i<n;i++) {a=a+i;}
```

3n+3 Statements ⇐

ii) Induction Variables Elimination

```
for(i=0;i<n;i++)
{
    x= i+a;
    y = i*b;
    z= i-c;
}
```

j and k are deleted

⇐

```
      k=0;
for (i=0, j=0; i<n; i++)
   {
      x = i+a;
      y = j*b;
      z = k-c;
      k++;
      j++;
   }
```

Variables	Induction Variables
i	i
j	j
n	k
k	x
x	y
y	z
z	
a	
b	
c	

iii) Loop Merge/Loop Combine/Loop fusion

```
for (i=0; i<n; i++)
{
      A[i] = i+1;
}
for (j=0; j<n; j++)
{
      B[j] = j*5;
}
```

⇒

```
for (i=0; i<n; i++)
{
      A[i] = i+1;
      B[i] = i*5;
}
```

iv) Loop Unrolling

```
for ( i= 1;  i< =4n; i++)
{
    printf ("gate");
}
```

Half

Double

⇒

```
for (i=1 ; i<=2n; i++)
{
    printf("gate");
    printf("gate");
}
```

one more time loop unroll

```
for(i=1; i<=n ; i++)
{
    printf("gate");
    printf("gate");
    printf("gate");
    printf("gate");
}
```

Example:

common

i<<1

```
i = i*2

for (x=0; x < i*2 ; x++)
{
        A[x] = 2+3;
        y = a*b
}
```

constant foldin

code move

✓ A) Code Motion

✓ B) Strength Reduction

✓ C) Constant Folding

D) Copy propogation

✓ E) common Sub exp elimination

Example:

After propogation, this code is dead code

fold

10*2

```
i = 10 ;
for (x=0; x < i*2 ; x++)
{
        A[x] = a ;
        y = a*b
}
```

code move

[✓ A) Code Motion]

B) Strength Reduction

[✓ C) Constant Folding]

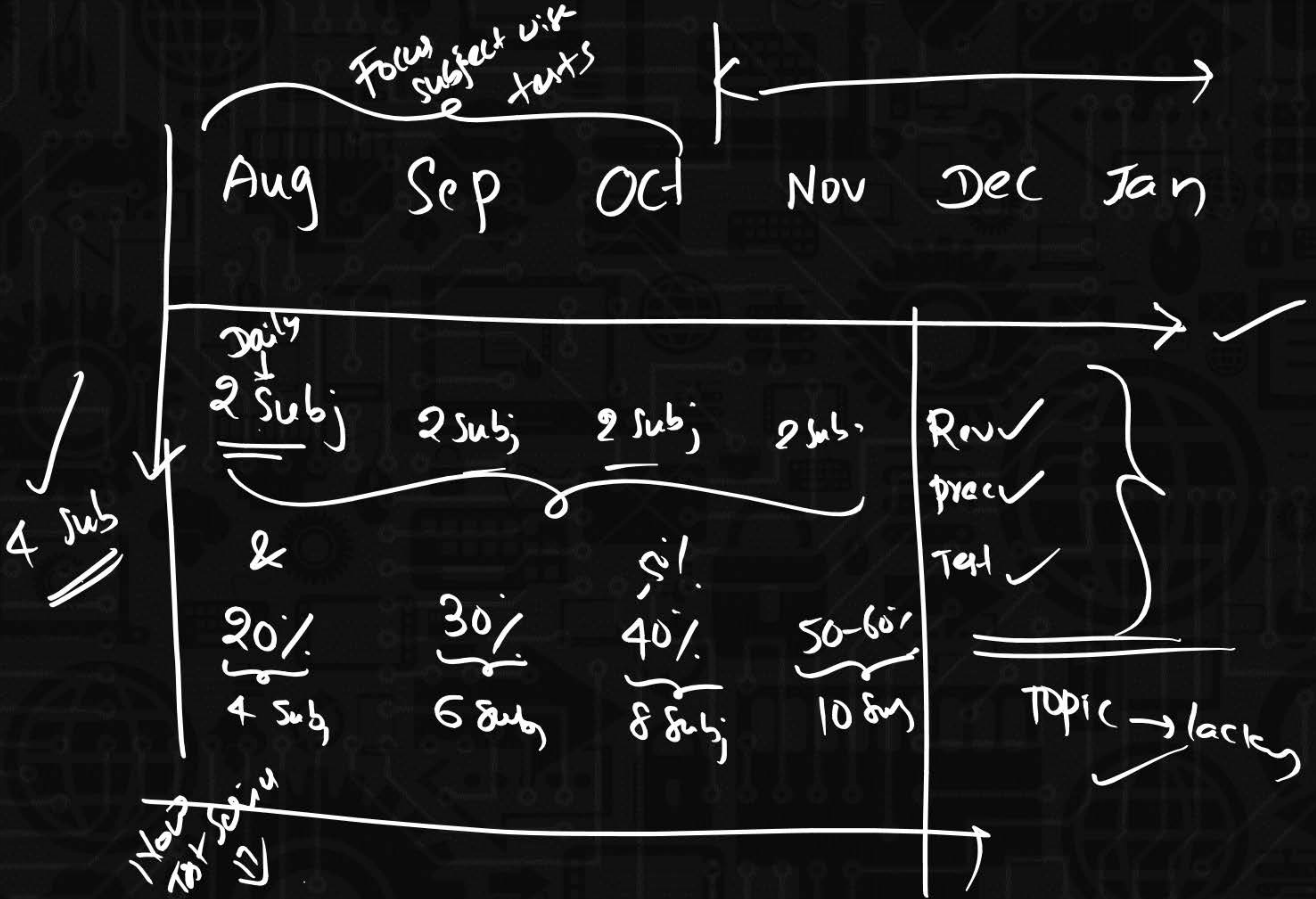
[✓ D) Copy propogation]

E) common Sub exp elimination

[✓ F) Dead code Elimination]

Data Flow Analysis

- 1) Forward Analysis
  - Reaching definitions Analysis
  - Available Expressions Analysis
- 2) Backward Analysis
  - *** live variable Analysis



Focus subject wise tests

Aug	Sep	Oct	Nov	Dec	Jan
Daily ↓ 2 Subj	2 Subj	2 Subj	2 Sub.	Rev ✓ Prac ✓ Test ✓	
&		syl.			
20%	30%	40%	50-60%		
4 Sub	6 Sub	8 Subj	10 Sub	TOPIC → lacking	

4 Sub ✓

Now Test Series ✓

Summary

↳ Code optimization techniques

Next : Data Flow Analysis

THANK YOU
GW
SOLDIERS !