CS & IT ENGINEERING

Compiler Design

Intermediate code and code optimization

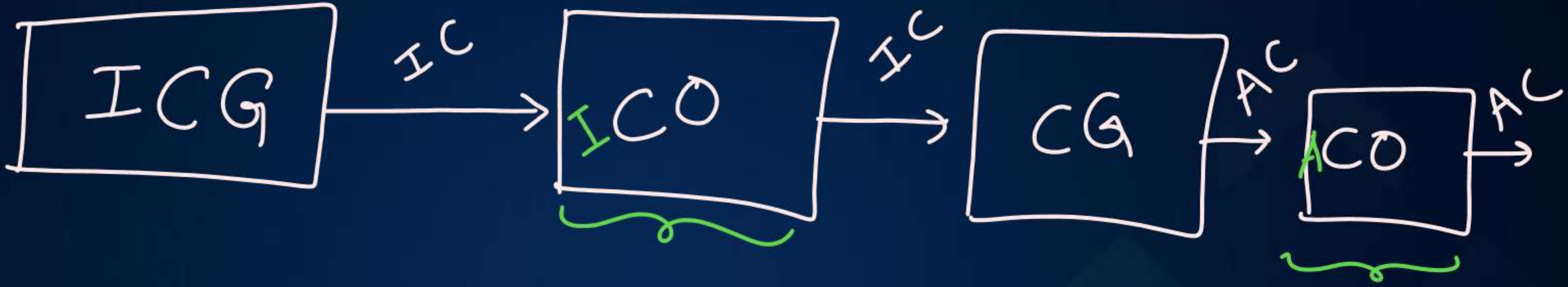
Lecture No. 3

By- DEVA Sir

TOPICS TO BE COVERED

Code Optimization [CO]

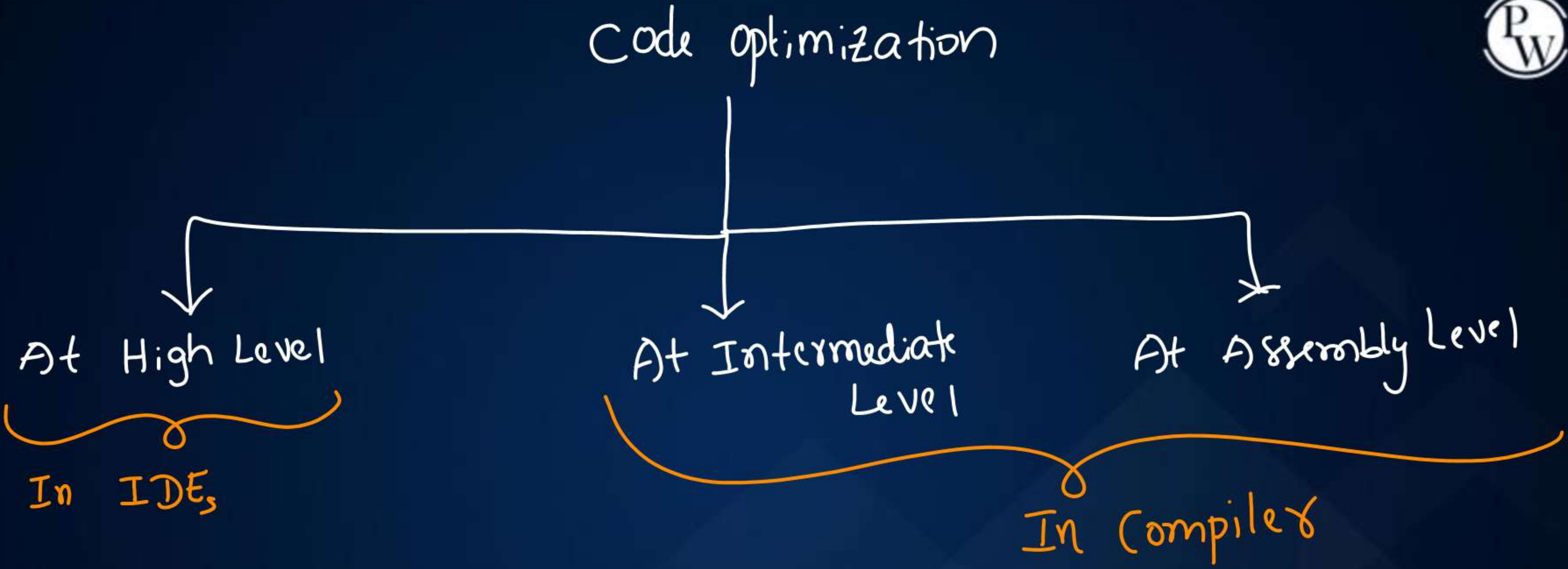
- What is CO ?
- CO Techniques
- Data Flow Analysis
  - Live Variable Analysis
  - Reaching Definition Analysis


ICG
IC
ICO
IC
CG
AC
ACO
AC

Code optimization ?

↳ It may save time/space

$$\underbrace{x = y * 2}_{\text{Costlier}} \Longrightarrow \underbrace{x = y + y}_{\text{cheaper}}$$



Code optimization

At High Level

In IDEs

At Intermediate Level

At Assembly Level

In Compiler

Code Optimization

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

- Local optimization
- Global optimization

Code optimization Techniques :

① Constant Folding

$$x = \underbrace{2*3}_{\text{folding}} + y \Longrightarrow x = 6 + y$$

② Cancellations

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

③ Identity

$$x = \underbrace{y+0} - z \Longrightarrow x = y - z$$

$$a = \underbrace{b*1} + e \Longrightarrow x = b + e$$

④ Strength Reduction

$$\underbrace{x = y * 2}_{\text{Costlier}} \implies \underbrace{x = y << 1 \quad \text{OR} \quad x = y + y}_{\text{Cheaper}}$$

n-bit data

$$\underbrace{y * 2}_{n \text{ registers}}$$

$$\underbrace{x = y + y}_{2 \text{ registers}}$$

(5) Common Sub-expression elimination

↳ We can use DAG

$$x = \boxed{(a*b)} + \boxed{(a*b)} - c$$

⇓

$t_1 = a*b$

$x = t_1 + t_1 - c$

⑥ Copy propogation
[constant/variable]

$$\boxed{\begin{aligned} x &= \overset{\text{constant}}{20} \\ y &= x + a \end{aligned}} \xrightarrow{\text{constant propogation}} \boxed{y = 20 + a}$$

$$\boxed{\begin{aligned} x &= b \\ y &= x + a \end{aligned}} \xrightarrow{\text{Variable propogation}} \boxed{y = b + a}$$

⑦ Dead Code Elimination :

$x = a + b$

$\boxed{y = a * b}$ Dead code

$z = x + c$

$\text{print}(z)$

$\Longrightarrow$

$x = a + b$

$z = x + c$

$\text{print}(z)$

(8) Loop optimizations

- Code Motion
- Induction variables Elimination
- Loop Merging
- Loop unrolling.

Code Motion:

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

loop invariant code

⇓

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

Induction variables Elimination:

Variables:

i
j
n
x
y
a

```
j=0;
for(i=0; i<n; i++)
 {
     x=x+i;
     y=y*j;

     z=x-a;

     j++;
 }
```

⇒

Induction variables

i
j
x
y
z

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

}
```

Loop Merge / Loop Combine / Loop fusion :

6n+4

3n+2

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

3n+2

```
for (j=0 ; j<n ; j++)
{
    B[j]= j*3 ;
}
```

⇒

```
for(i=0; i<n; i++)
{
    A[i]=i+1;
    B[i]=i*3;
}
```

4n+2

Loop unrolling:

Half

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

Double

⇒

$12n+2$

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

⇒

$8n+2$

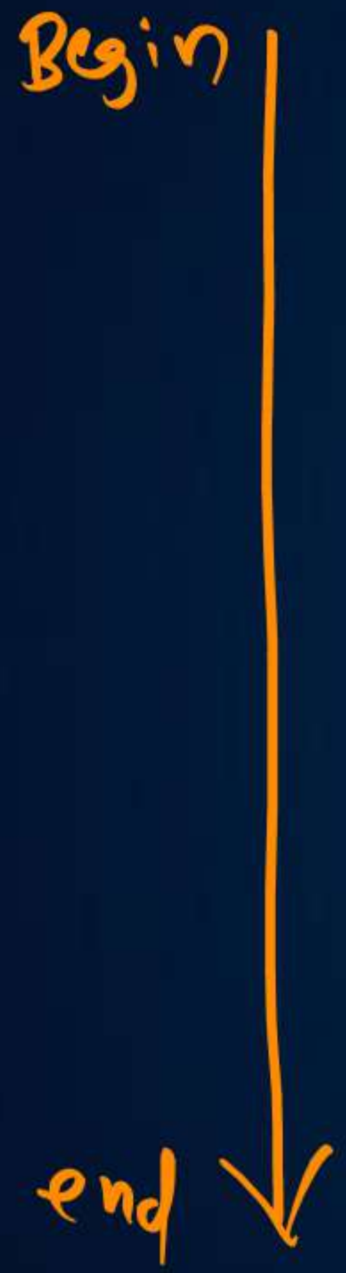
```
for(i=1;i<=n ;i++)
{
    printf("a");
    printf("a");
    printf("a");
    printf("a");
}
```

$6n+2$

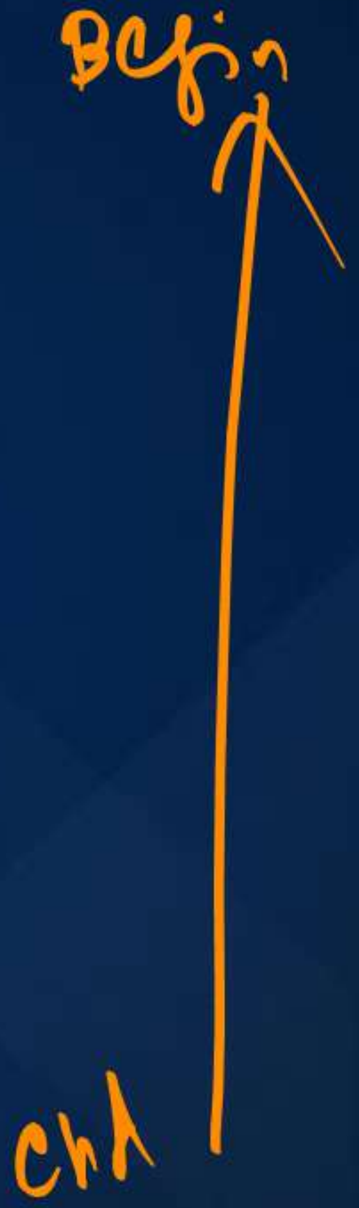
Data Flow Analysis:

- 1) Forward Analysis
  - Reaching Definitions
  - Available Expressions
- 2) Backward Analysis
  - Live variable Analysis

Forward Analysis



Backward Analysis



Summary

↳ Code optimization techniques

Next : Data Flow Analysis

THANK YOU
GW
SOLDIERS !