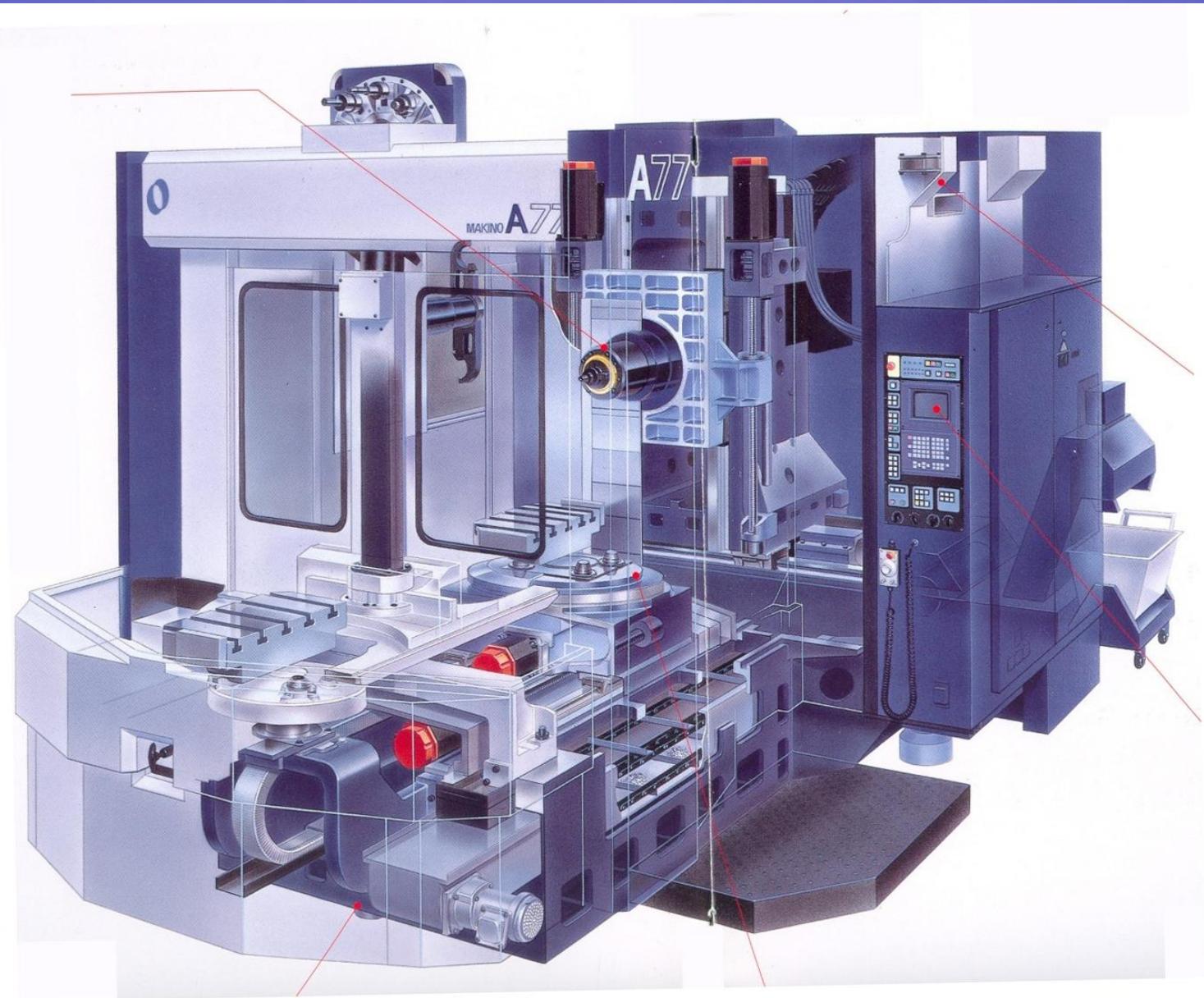


# Theory and Design of CNC Systems

*Chapter 1*  
**Introduction to NC Systems**











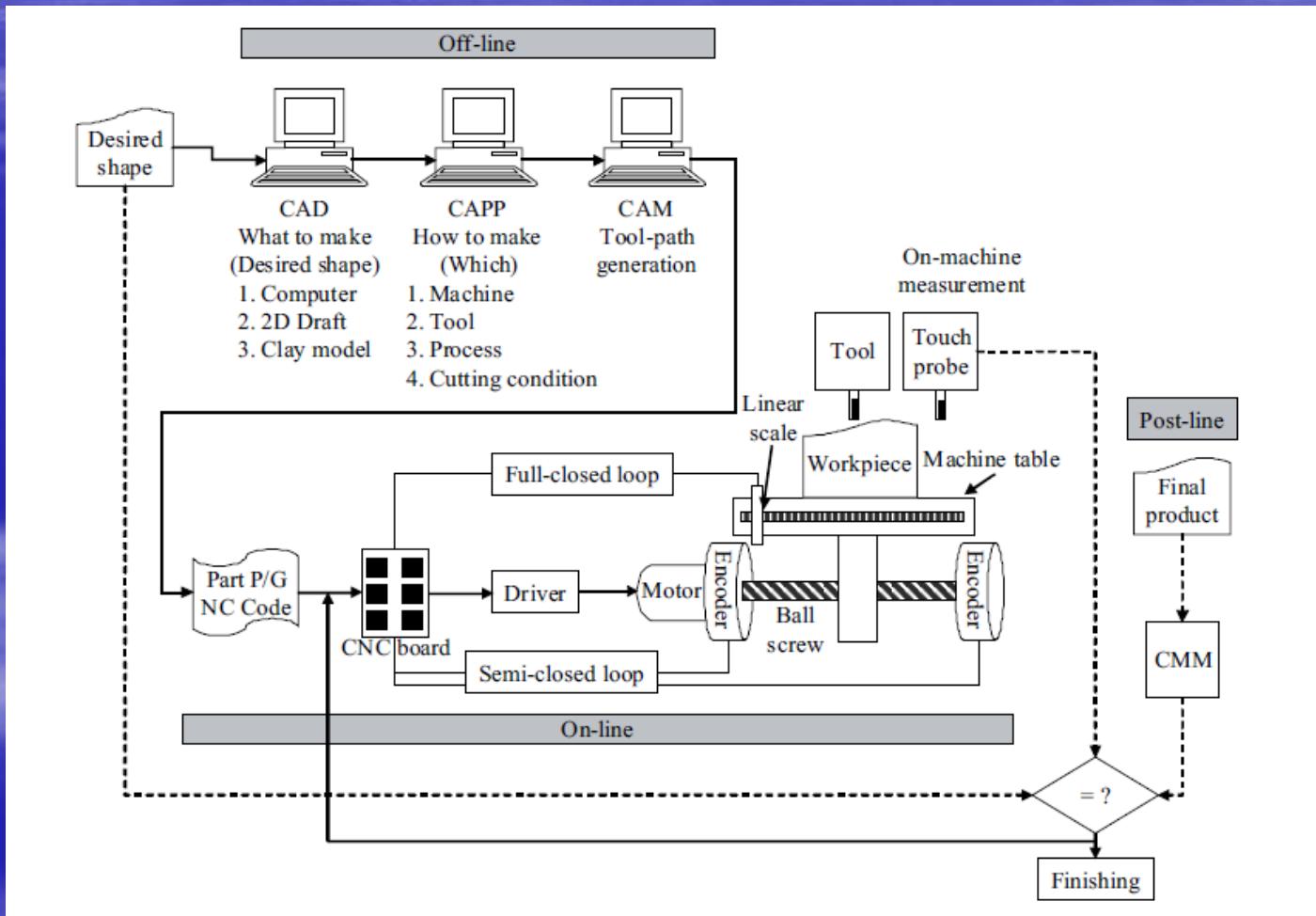
# What's CNC

- CNC (Computer Numerical Control) machine
  - mother machine: a machine that makes machines
  - mechanical components + NC system
- NC System
  - fundamental technology for factory automation
  - used not only for machine tools but also all machines that need motion controlled by servo systems

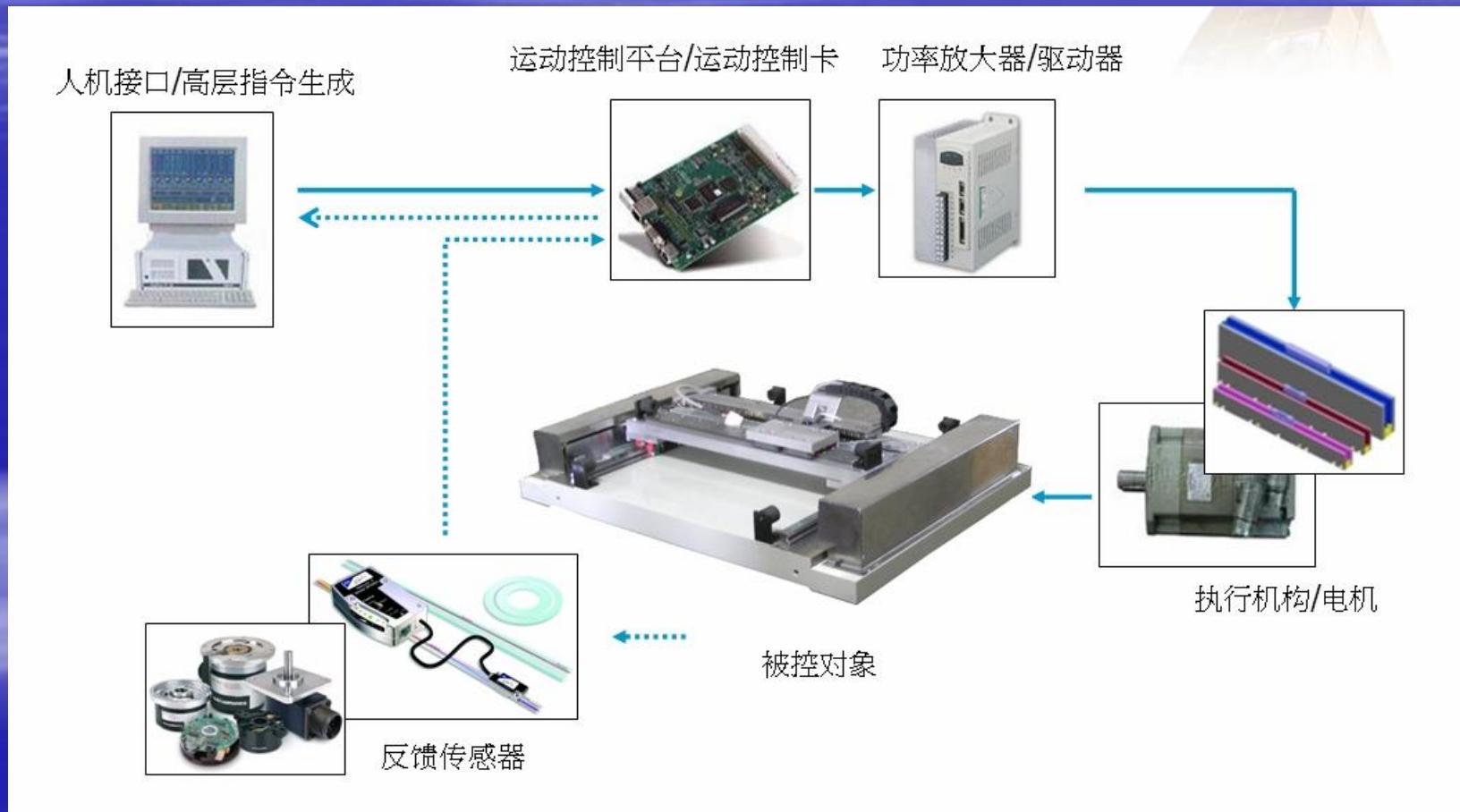
# History

- NC概念促發於1947年
  - 二次大戰過後
  - 用於飛機之製造
- 1952年, MIT servo laboratory發展出第一台3軸NC銑床

# Work Flow



# Architecture



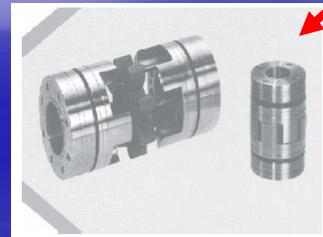
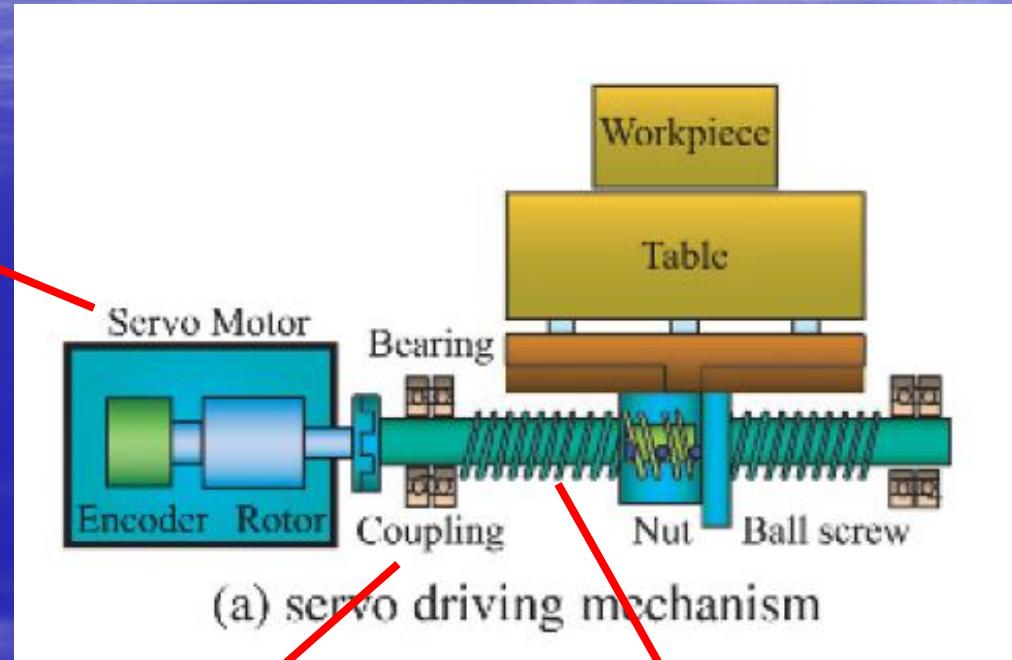
# Architecture

- 影響精度的因素
  - servo motor
  - table guide
  - ball screw
  - spindle
  - rigidity of machine construction
  - encoder and sensors
  - control mechanism
- The construction of the machine and the machine components should also be designed to be insensitive to **vibration** and **temperature**.

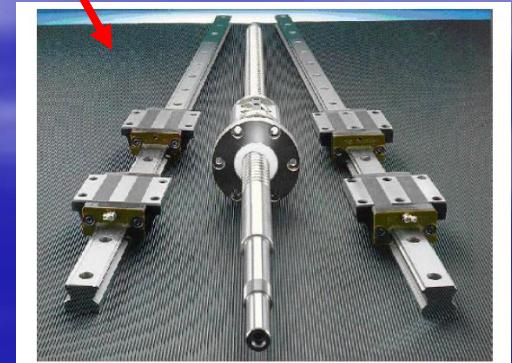
# 伺服單軸結構



精確定位  
動態響應  
寬轉速範圍



精度  
高速  
低摩擦



# 伺服單軸結構

- 對伺服馬達的要求
  - To be able to get adequate **output of power** according to work load
  - To be able to **respond quickly** to an instruction
  - To have good **acceleration and deceleration** properties
  - To have a **broad velocity range**

# 伺服單軸結構

## ■ 對伺服馬達的要求

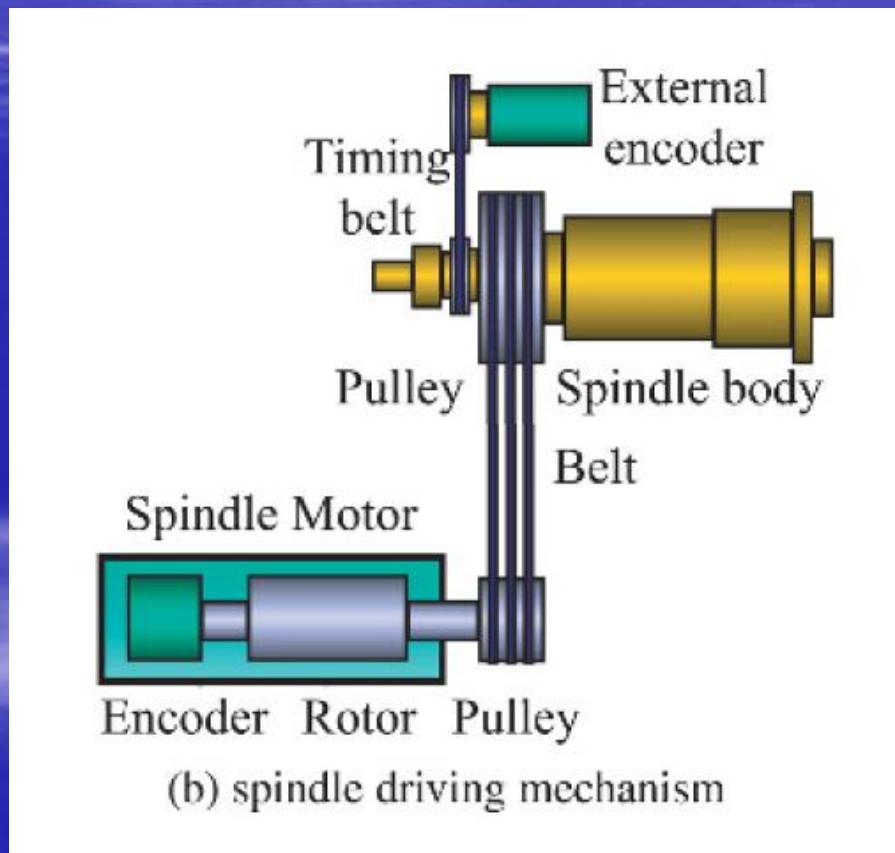
- To be able to control velocity safely in all velocity ranges.
- To be able to be continuously operated for a long time
- To be able to provide frequent acceleration and deceleration
- To have high resolution in order to generate adequate torque in the case of a small block

# 伺服單軸結構

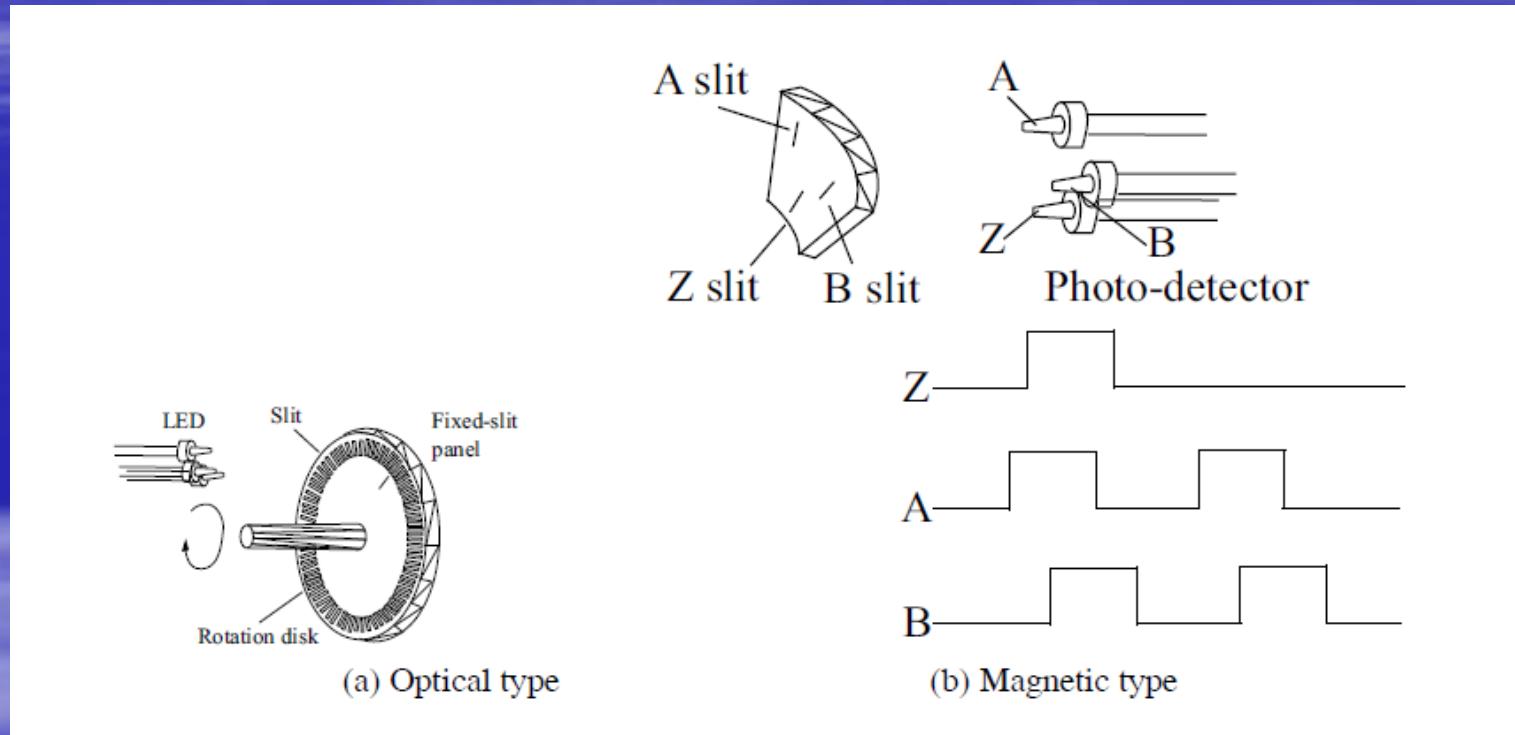
- 對伺服馬達的要求

- To be easy to rotate and have high rotation accuracy
- To generate adequate torque for stopping
- To have high reliability and long length of life
- To be easy to maintain

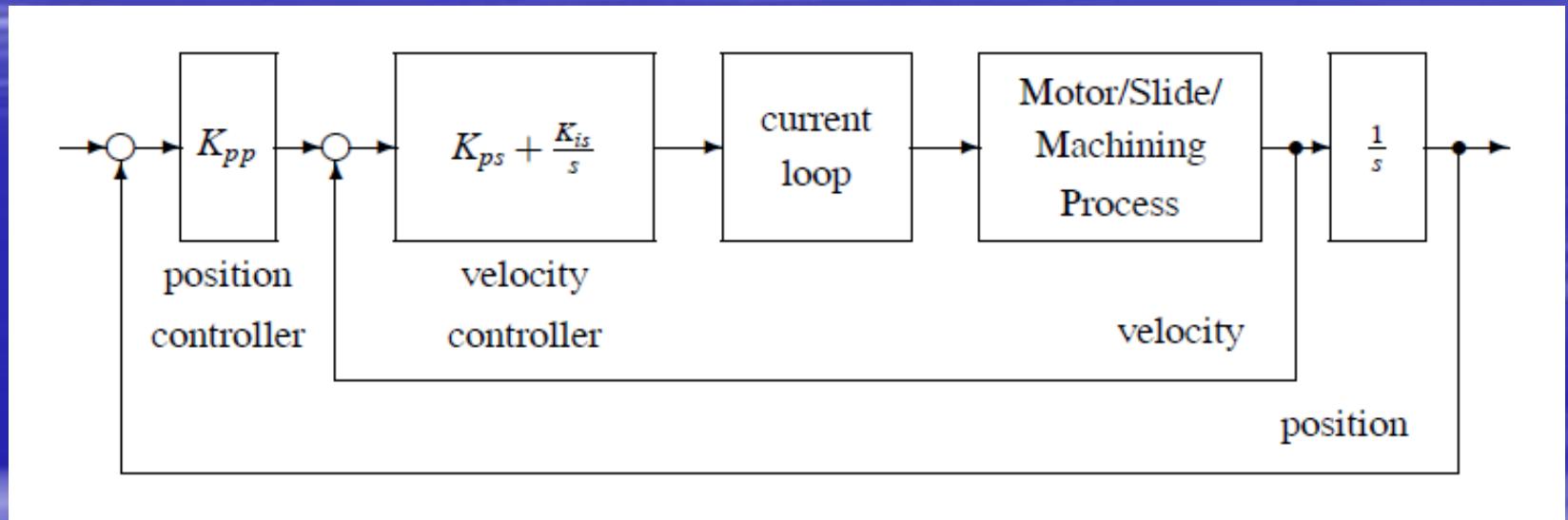
# 主軸結構



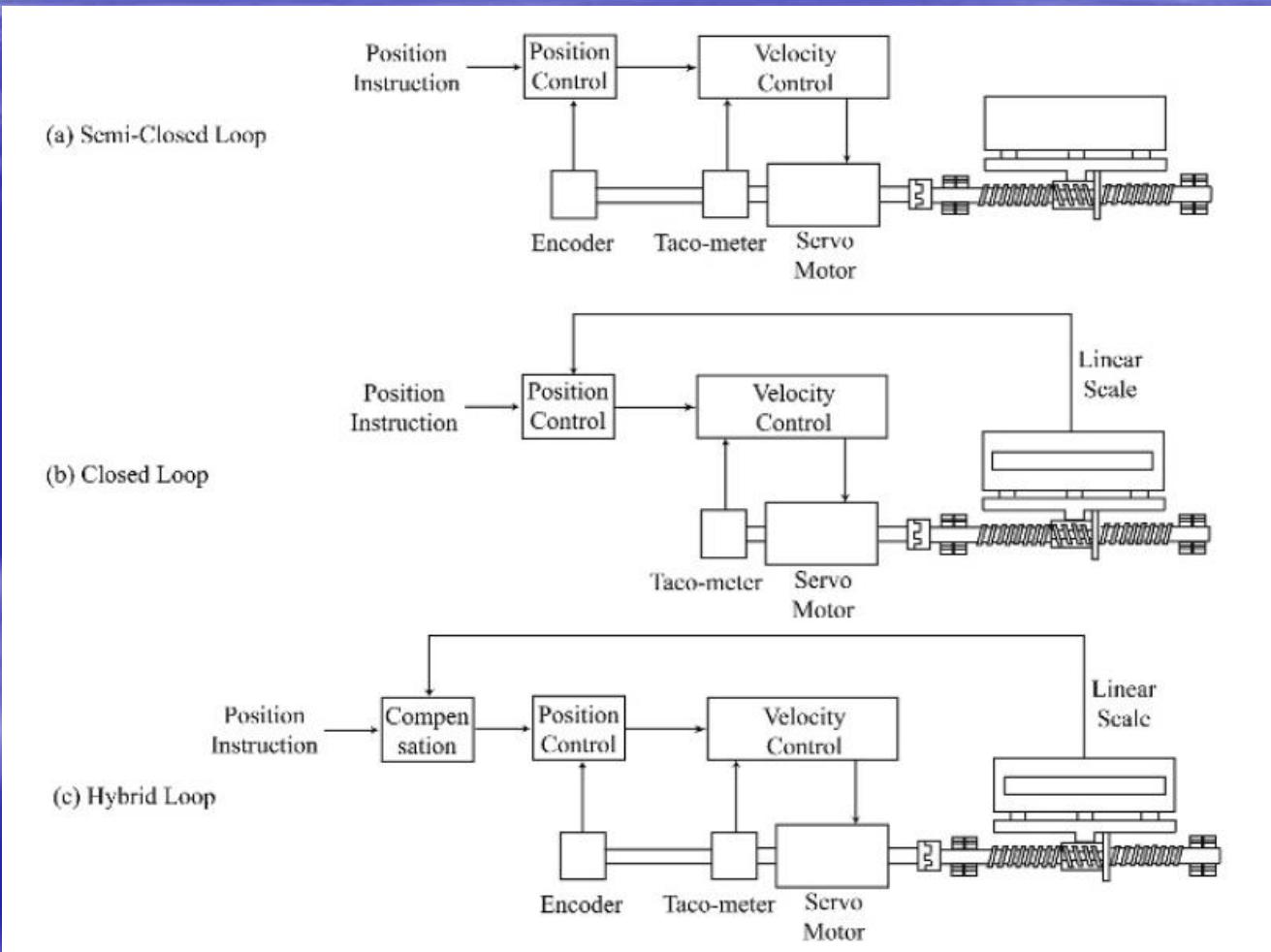
# 編碼器原理



# 伺服控制架構



# 伺服控制架構



# Semi-Closed Loop

- The most popular control mechanism
- Position accuracy is greatly influenced by the accuracy of the ball screw.
  - Due to the precision ball screw, the problem with accuracy has practically been overcome.
- **Pitch-error compensation and backlash compensation** can be used in order to increase the positional accuracy.

# Semi-Closed Loop

- Pitch-error compensation
  - The instructions to the servo are modified to remove the accumulation of positional error.
  - Accumulation pitch error is varied according to the **temperature**.
- Backlash compensation
  - whenever the moving direction is changed, additional pulses corresponding to the amount of backlash are sent to the servo.
  - Amount of backlash can be varied according to the **weight** and **location** of the workpiece.

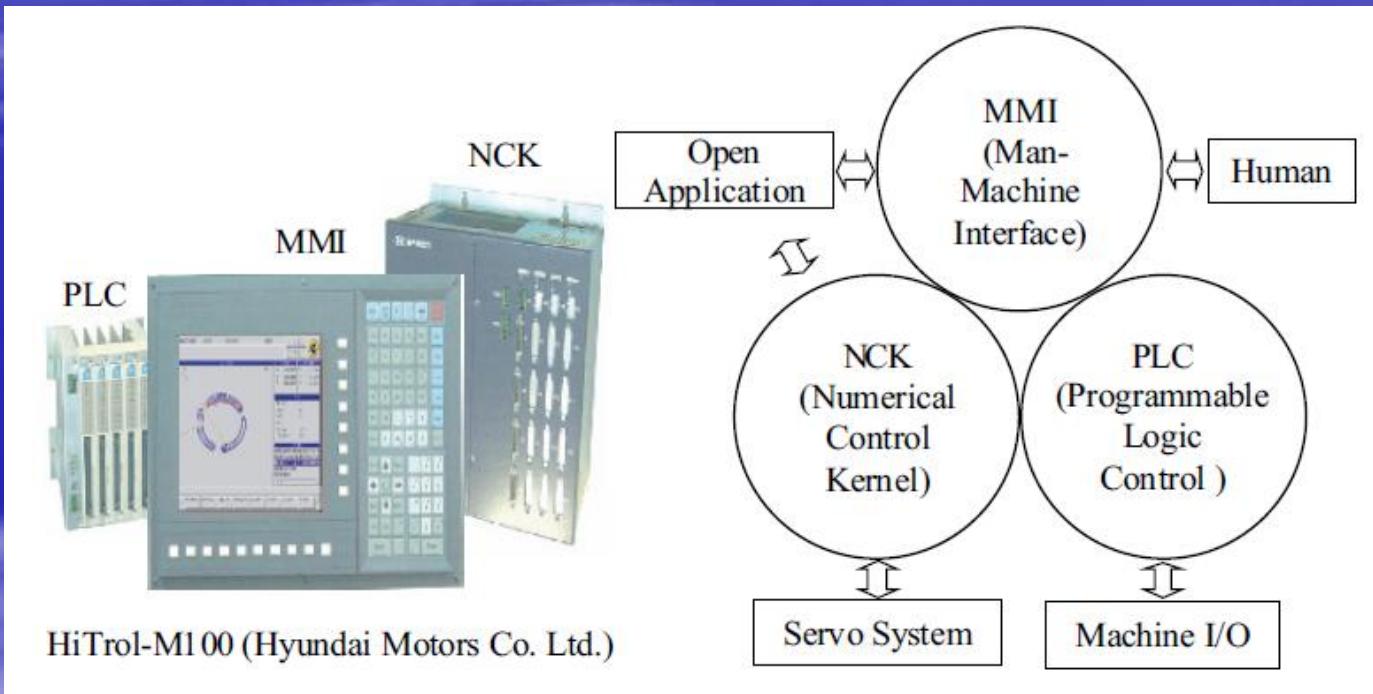
# Closed Loop

- Relatively insensitive to the accuracy of the ball screw
- But possibly unstable
  - it is necessary to increase the resonance frequency of the machine driving system by
    - increase the rigidity of the machine
    - decrease the friction coefficient of the perturbation surface
  - or it is necessary to lower the control gain
    - If the gain is very low, the performance becomes poor with respect to positioning time and accuracy

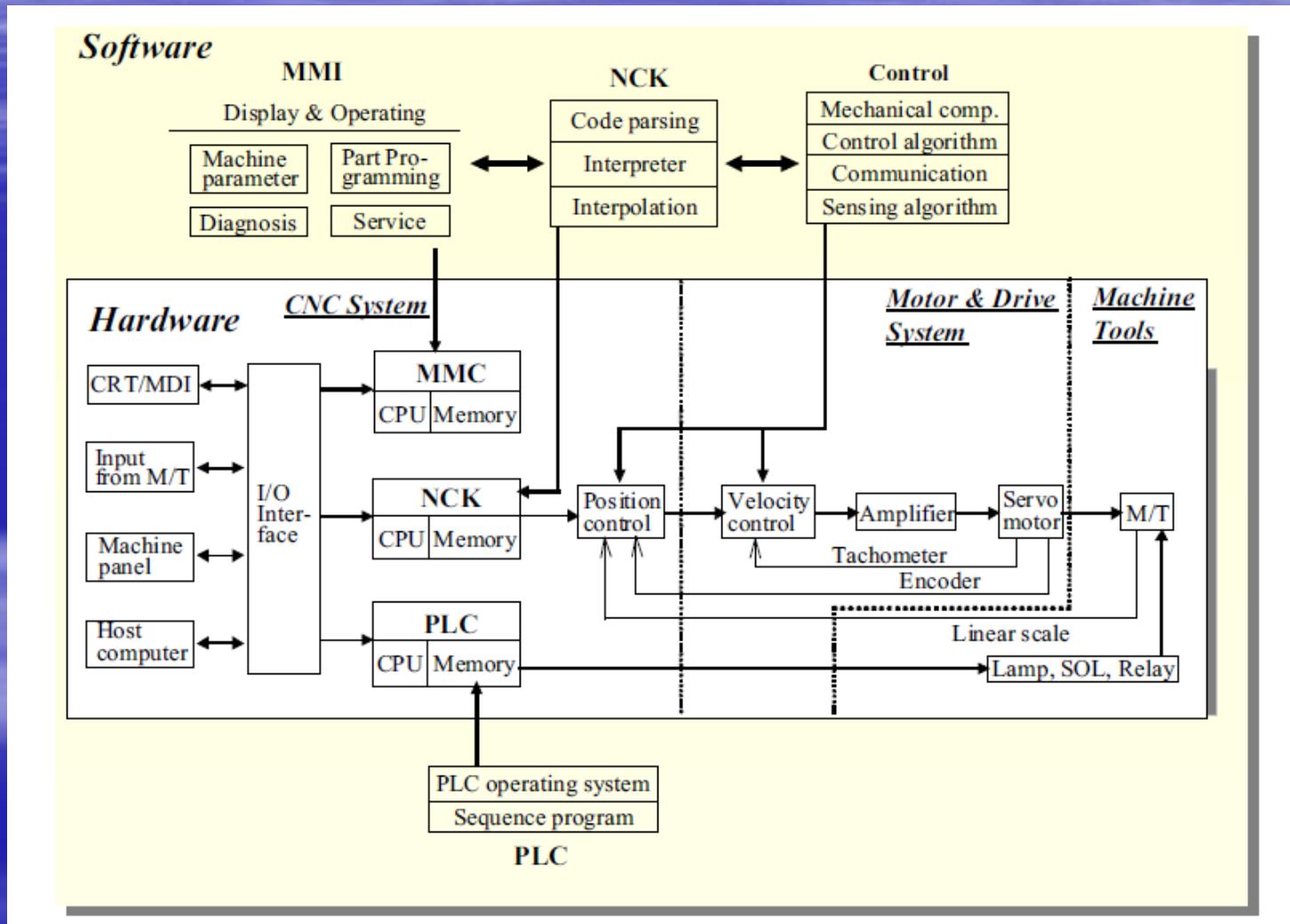
# Hybrid Loop

- Servo system is controlled by semi-closed loop method.
- The closed loop is used for compensating only positional error.
- Higher cost

# NC System 的組成



# NC System 的組成



# MMI (Man Machine Interface)

- Offer the interface between NC and the user
- Execute the machine operation command
- Display machine status
- Offer functions for editing the part program
- Communication
- More modern features: CAD, CAM, simulation, web interface ...

# NCK (Numerical Control Kernel)

- Being the core of the CNC system
- Interpret the part program
- Execute interpolation, position control, and error compensation
- Control the servo system

# NCK (Numerical Control Kernel)

- Advanced features:
  - tool-breakage detection
  - compensation of thermal deformation
  - adaptive control
  - compensation of tool deflection
  - monitoring of cutting force, heat, and electric current

# PLC (Programmable Logic Control)

- Play the role of controlling the machine's behavior except servo control
- Control tool change
- Control Spindle speed
- Control workpiece change
- I/O processing

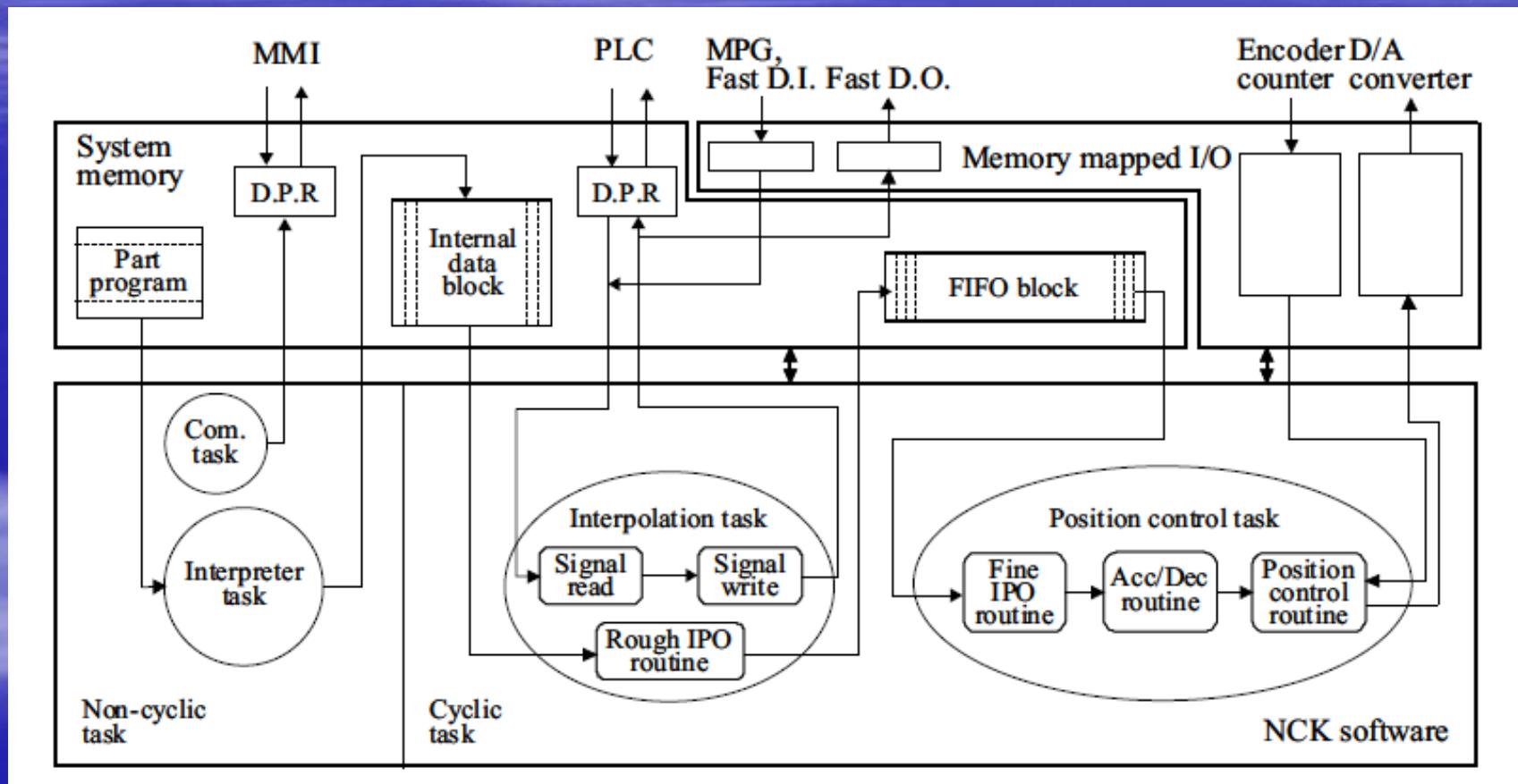
# Communication between NC and Servo Systems

- Traditionally, velocity commands in **analog** format are used for transmitting signals to the servo system.
  - But sensitive to noise → loss accuracy
- The trend is to employ **real-time** **digital communication** instead.
  - Insensitive to noise
  - Can exchange a variety of data, such as servo parameters and status

# Communication between NC and Servo Systems

- Fieldbus protocols
  - EtherCAT
  - Mechatrolink
  - RTEX
  - SSCNET
  - Profibus
  - DMCNET
- The communication mechanism has also been applied to I/O devices.

# Software Flow of NC System



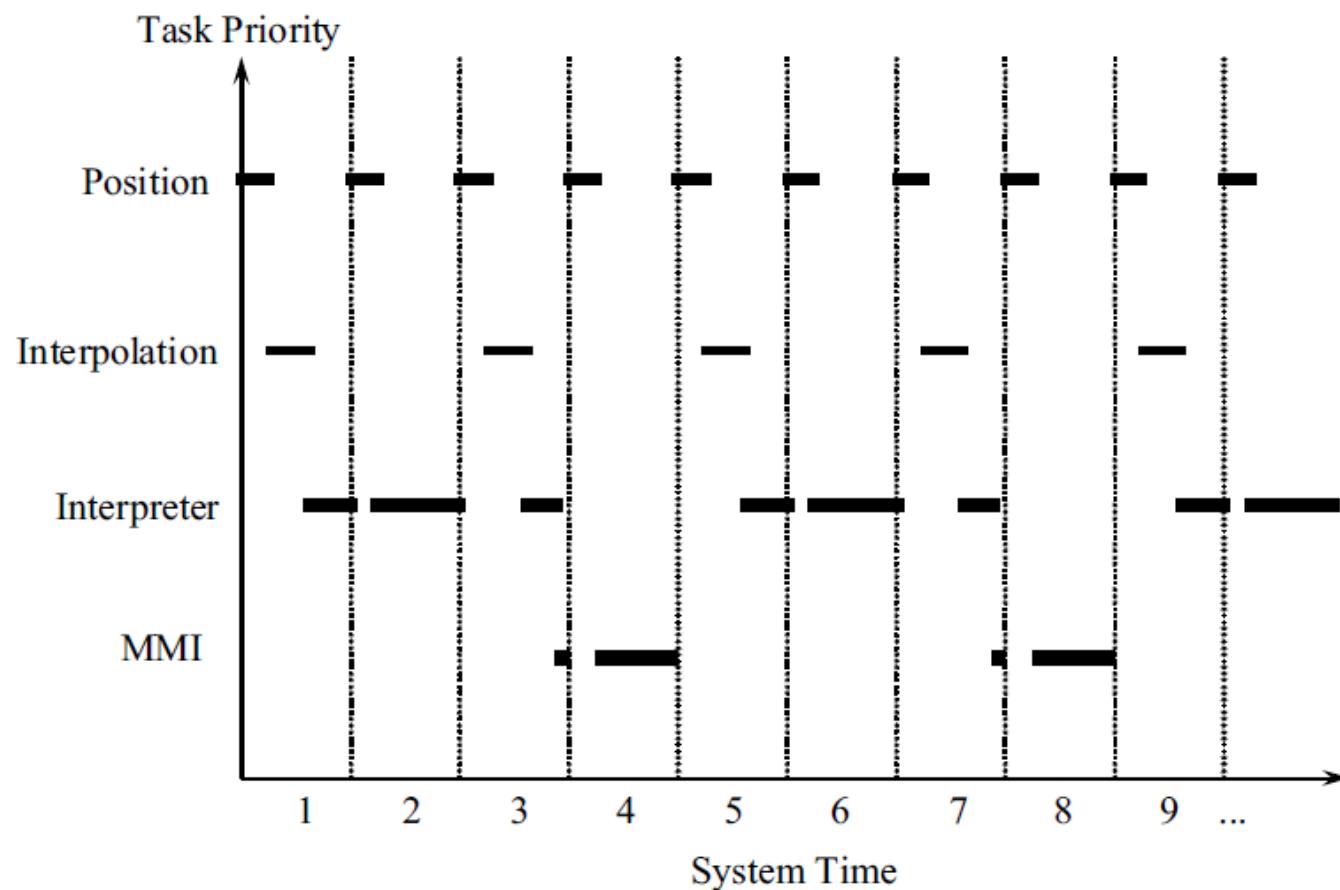
# Main Functions of NCK

- Interpreter
- Interpolator
- Acceleration/deceleration controller
- Position controller

# Real-Time Implementation

- In general, usage of individual processors for MMI, NCK, and PLC modules is typical.
- But using real-time OS, a single-processor implementation is possible by dividing the functions of these modules into cyclic tasks and non-cyclic tasks.

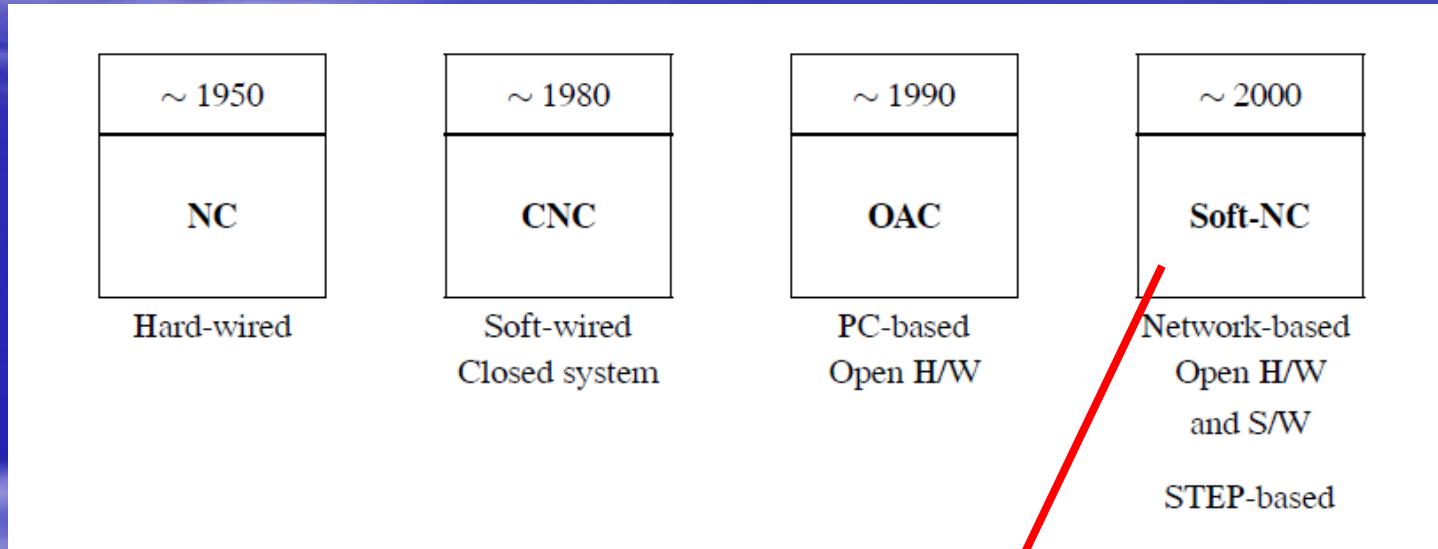
# Real-Time Implementation



# Real-Time Implementation

- In the example
  - the position control with the highest priority is activated every 1 msec
  - the interpolator with the 2nd priority every 2 msec
  - the interpreter is executed once every 4 msec
  - The MMI with the lowest priority is designed to use the surplus processor resource

# The Progress Direction of the CNC System



Examples:

EMC

EMC2 (LinuxCNC)

# Theory and Design of CNC Systems

*Chapter 2*  
**Interpreter**

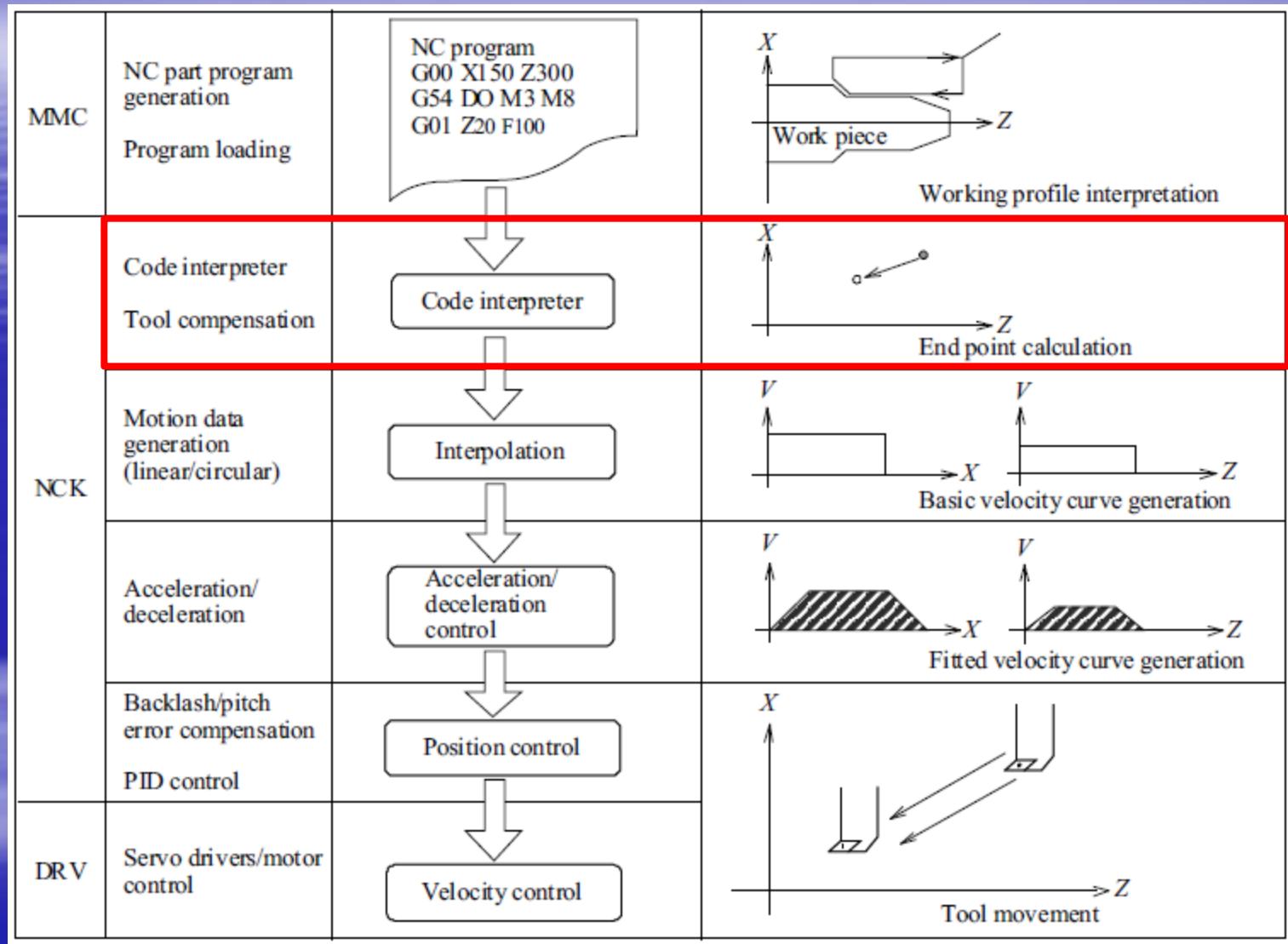
# What's interpreter

- A software module in NCK unit of CNC system.
- Translates **part program** into **internal commands** for moving tools and executing auxiliary function in the CNC system.

G/M code →

Interpreter

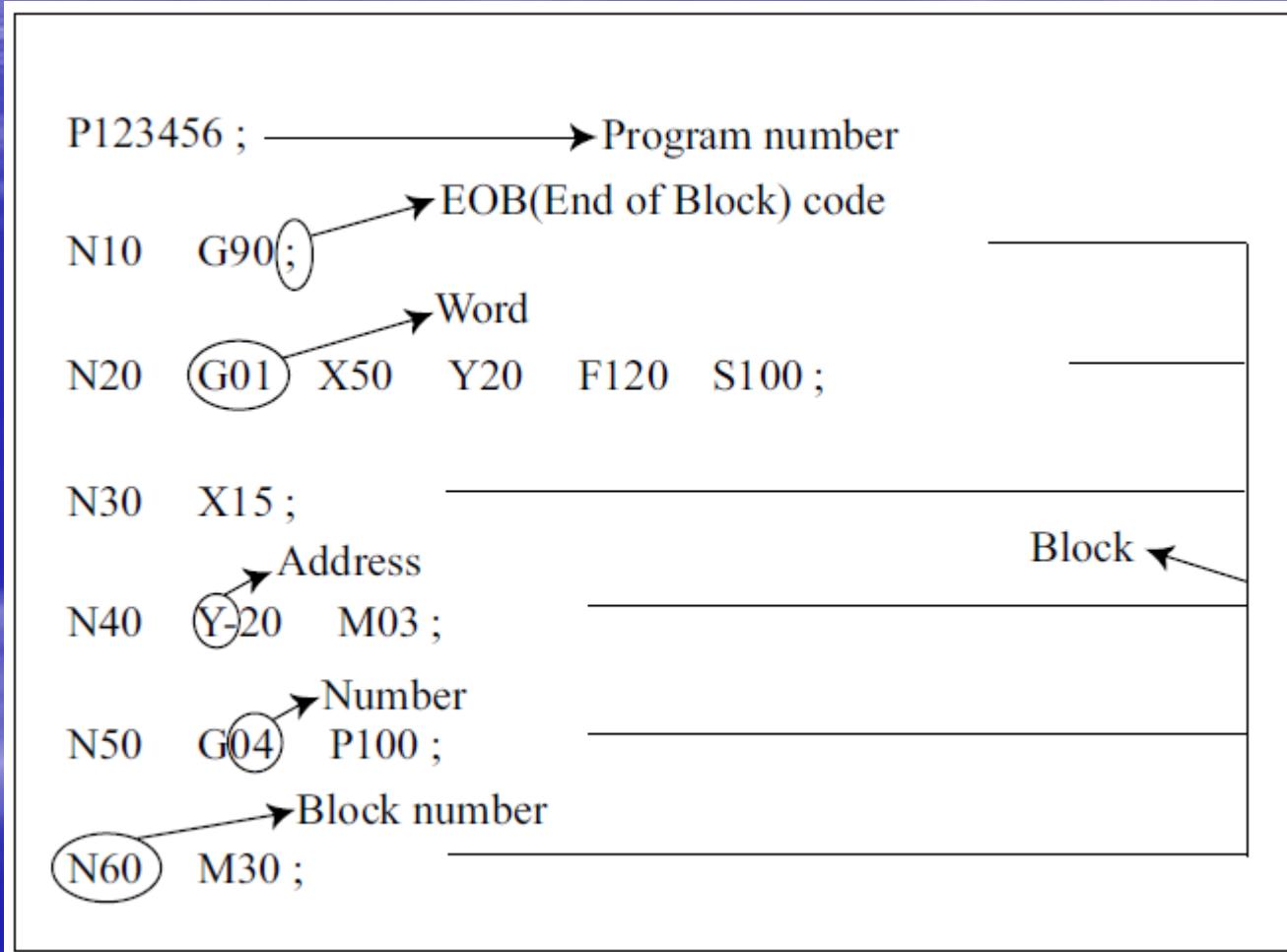
CNC  
→ understandable  
internal command



# Part Program

- English alphabet
  - G, M, S, F, N, P, ...
- Arabic numbers
  - 0, 1, 2, ..., 9
- Symbols
  - ;, -, .., ...

# Program Structure



# Program Structure

- Program Number

- a number for identifying the part program on CNC

- Block

- consists of one block number, at least one word, and the EOB

- Block Number

- a number for identifying the block

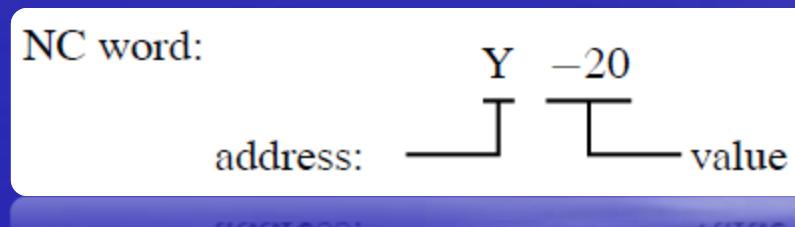
- EOB

- the end of block

# Program Structure

## ■ Word

- the minimum unit of part program.



## ■ Address

- constructed from one of the alphabetic characters

## ■ Number

- the data that is required to execute the behavior related with the address.

# Program Structure

- **Comment**

- has no influence on the execution of a part program.

N20 G01X0Y0 (MOVE TO ZERO POINT);



comment

- **End of a part program**

- M02 or M30 at the end of a part program.

# Address

Function	Address	Meaning (Example)	Unit
Program number	P	Program identity no. e.g. P123456	
Block number	N	NC seq.no. N100	
Preparatory function	G	Mode command G01	
Coordinate (command for translational axis)	X, Y, Z / U, V, W	Axis / dir. X100 W20	mm, inch
Coordinate (command for rotary axis)	A, B, C	Axis A30	deg
Feedrate	F	Feedrate per min. F200	mm/min, inch/min, deg/min
		Feedrate per rev. F1	mm/rev, inch/rev, deg/rev
Spindle speed	S	S3000	rpm

# Address

Function	Address	Meaning (Example)	Unit
Tool	T	Tool number T12	
Auxiliary function	M	Machine command M06	
Offset Number	H, D	Offset register no. H10	
Number of repetitions of subprogram	L	Iteration no. L5	
Radius of circle or arc	R	Arc rad., corner rad. R3	mm, inch
Chamfer	C	Chamfer amount C2	mm, inch
Center position of circle	I, J, K	Circle center coords.	mm, inch

# Address Group

- Preparatory function (G-code)
  - prepares the execution of the particular function.
    - Movement
    - Setting local coordinate system
    - Interpolation
    - ...

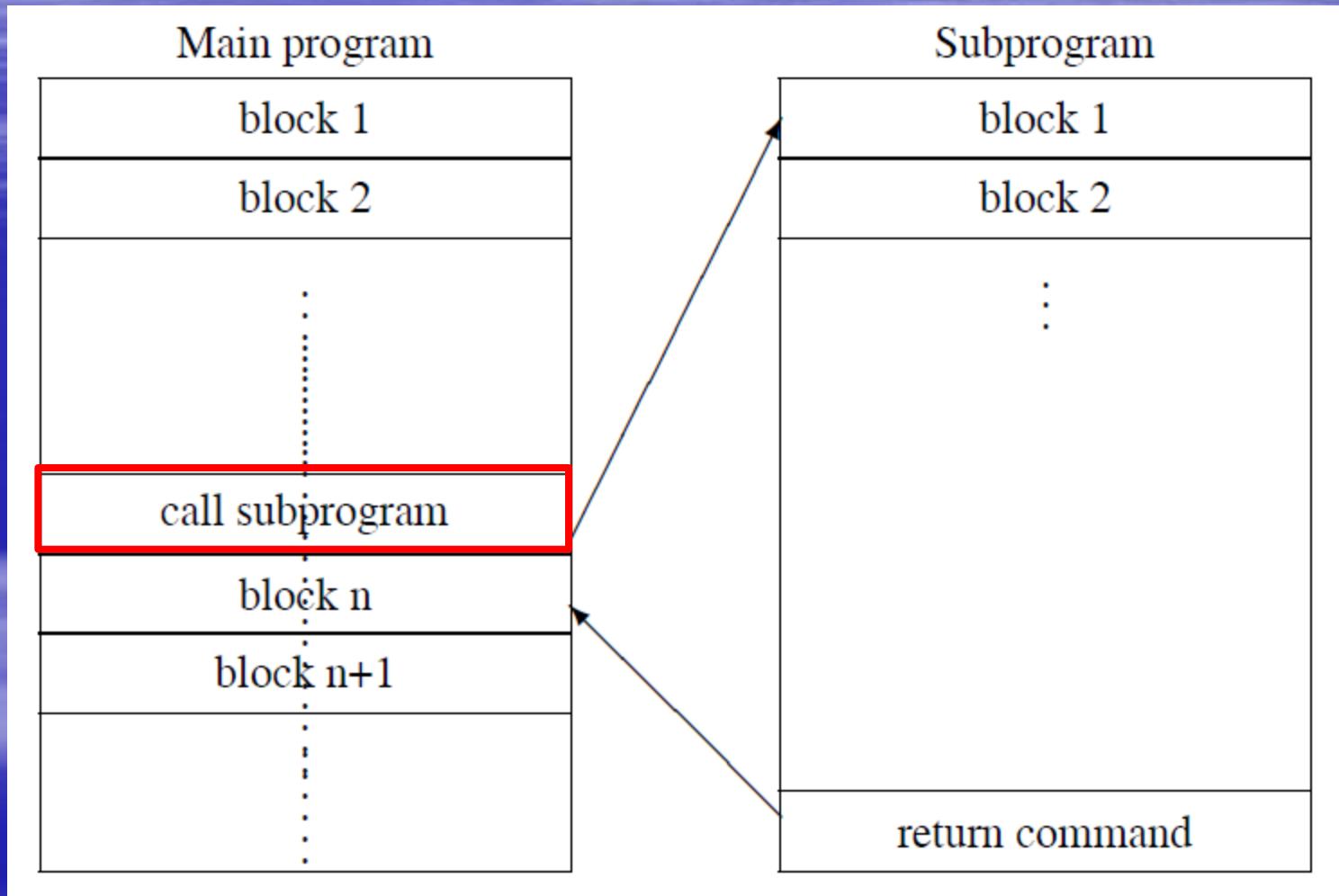
# Address Group

- Auxiliary function (M-code)
  - Function commands for the simple control.
    - Coolant
    - Start/End of a main and sub program
    - Spindle
      - CCW/CW, stop
    - ...

# Address Group

- Feed function (F-code)
  - Command the relative speed between tool and workpiece for interpolation command.
- Spindle function (S-code)
  - Command the spindle speed (RPM).
- Tool function (T-code)
  - Command tool change and specify the tool compensation.

# Main program and subprogram



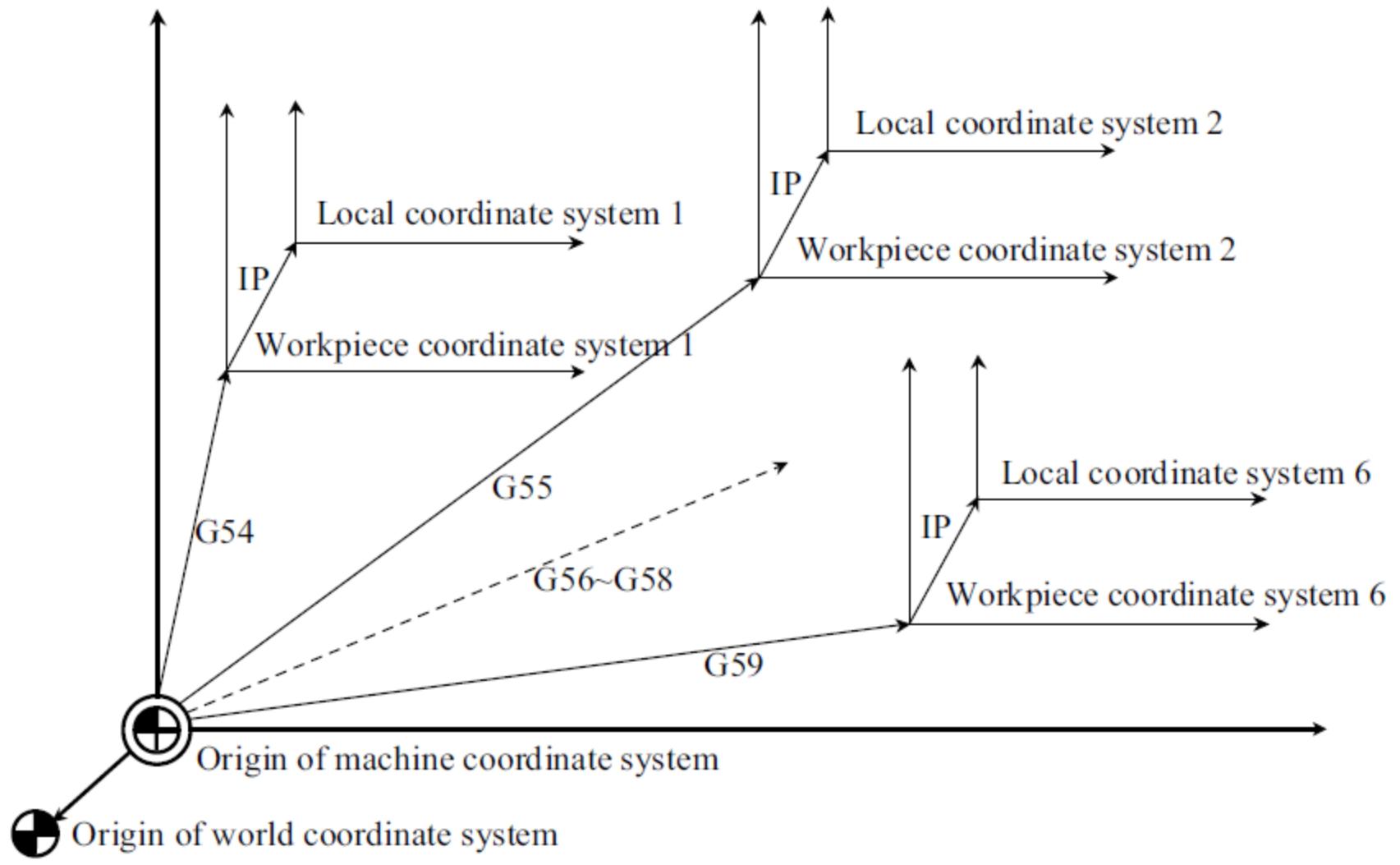
# Coordinate systems

- machine coordinate system
  - setting a particular point of the machine tool as the origin of a coordinate system.
  
- workpiece coordinate system
  - setting a particular point on the workpiece as the origin.
  - make editing a part program easier.

# Coordinate systems

- local coordinate system
  - define another coordinate system based on the workpiece coordinate system.

# Coordinate systems



# Coordinate systems

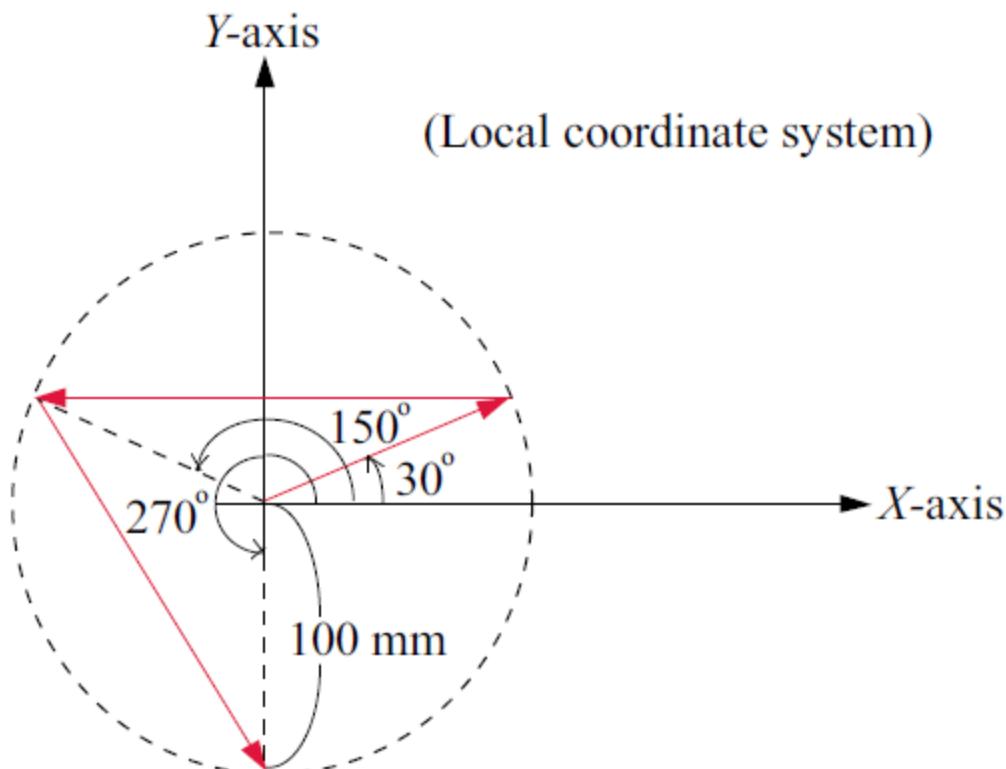
- G90
  - absolute programming mode
- G91
  - incremental programming mode
- G15
  - polar coordinate start
- G16
  - polar coordinates cancel

# Coordinate systems

- G17
  - XY-plane select
- G18
  - ZX-plane select
- G19
  - YZ-plane select

# Coordinate systems

N0100 G17 G90 **G15**; XY plane, absolute coordinate, **polar coordinate start**  
N0200 G00 X100 Y30 ; rad. 100mm, ang. 30deg  
N0201 X100 Y150 ; rad. 100mm, ang. 150deg  
N0202 X100 Y270 ; rad. 100mm, ang. 270deg  
N0203 G16 ; Polar coordinates cancel



# Coordinate systems

- G68  $\alpha$   $\beta$  R
  - Rotation function
  - $(\alpha, \beta)$  is center of rotation
  - R is degree of rotation

	G17	G18	G19
$\alpha$	X	Z	Y
$\beta$	Y	X	Z

- G69
  - Rotation cancel

# Coordinate systems

- G51 X\_Y\_Z\_P\_
  - Scaling function
  - X, Y, Z denotes the center position for scaling
  - P is used for the magnitude of the scaling
- G50
  - Scaling cancel
- G51.1
  - mirror image function.

# Coordinate systems

Main Program

P000001;

N10 G00 G90;

N30 G51.1 X50; → sym. X=50

N40 M98 P100001;

N50 G51.1 Y50; → sym. X=50, Y=50

N60 M98 P100001;

N70 G50.1 X0; → reset X

N80 M98 P100001;

N90 G50.1 Y0; → reset Y

N100 M30;

N20 M98 P100001;

Sub program

P100001;

N210 G00 G90 X60 Y60;

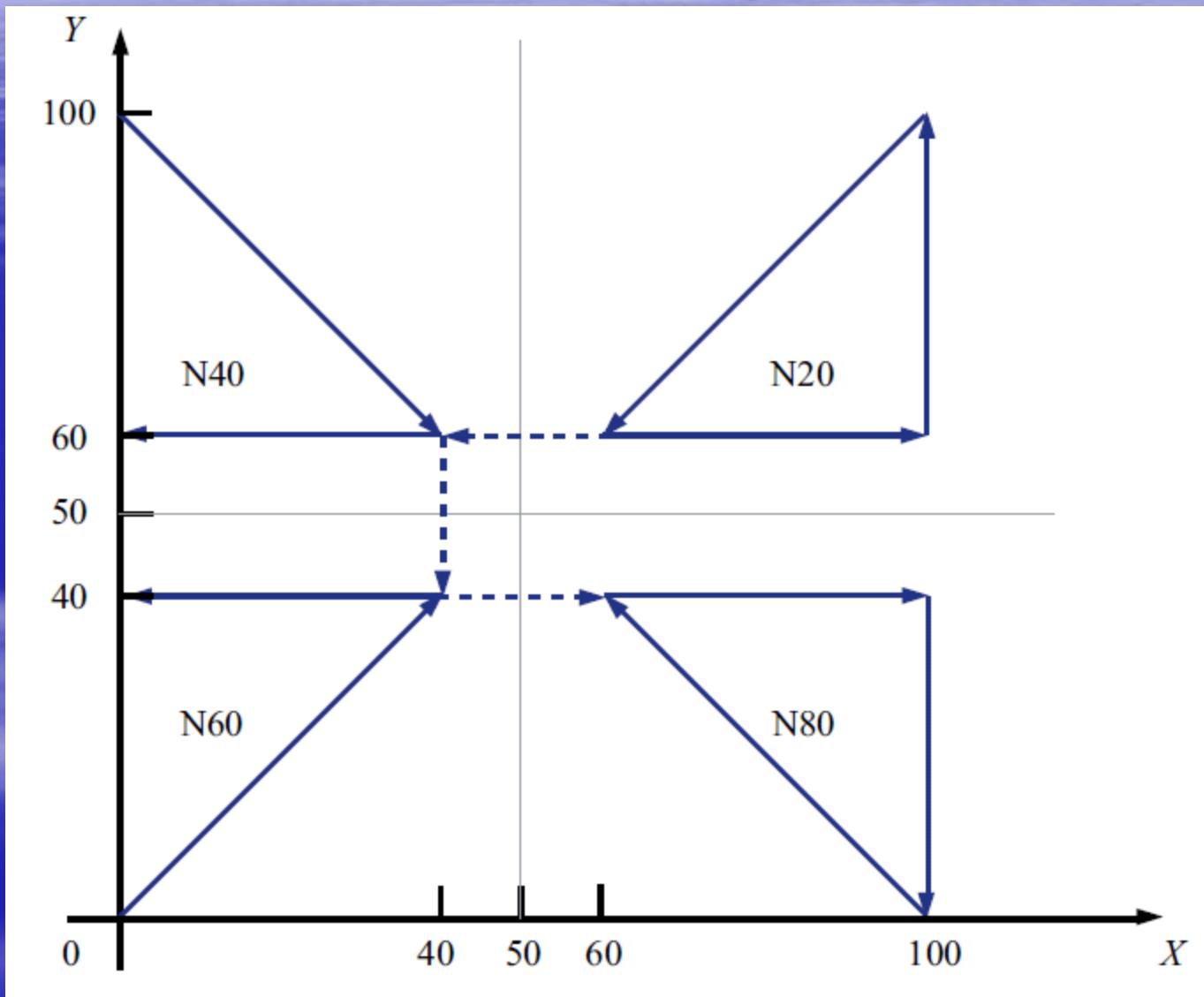
N230 Y100;

N240 X60 Y60;

N250 M99;

N220 X100;

# Coordinate systems

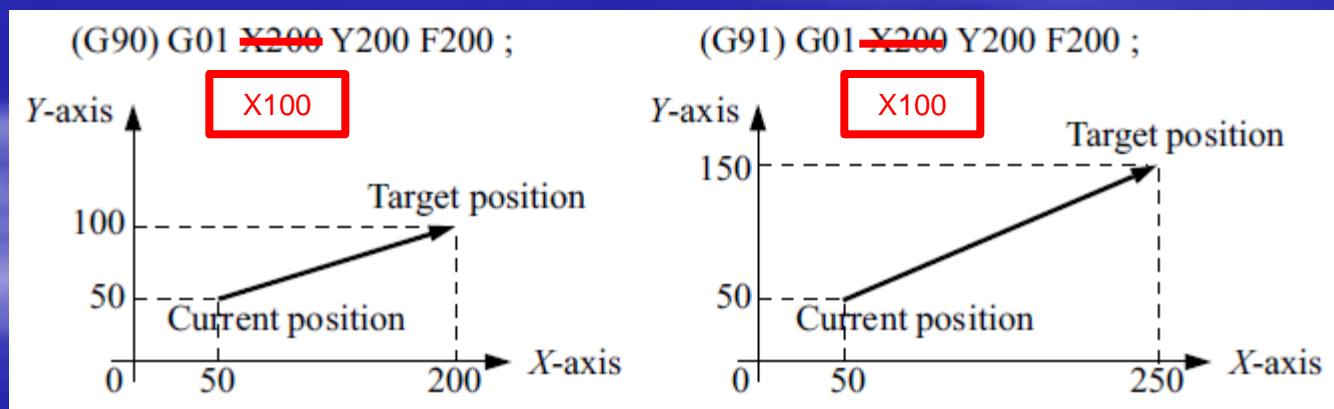


# Rapid move

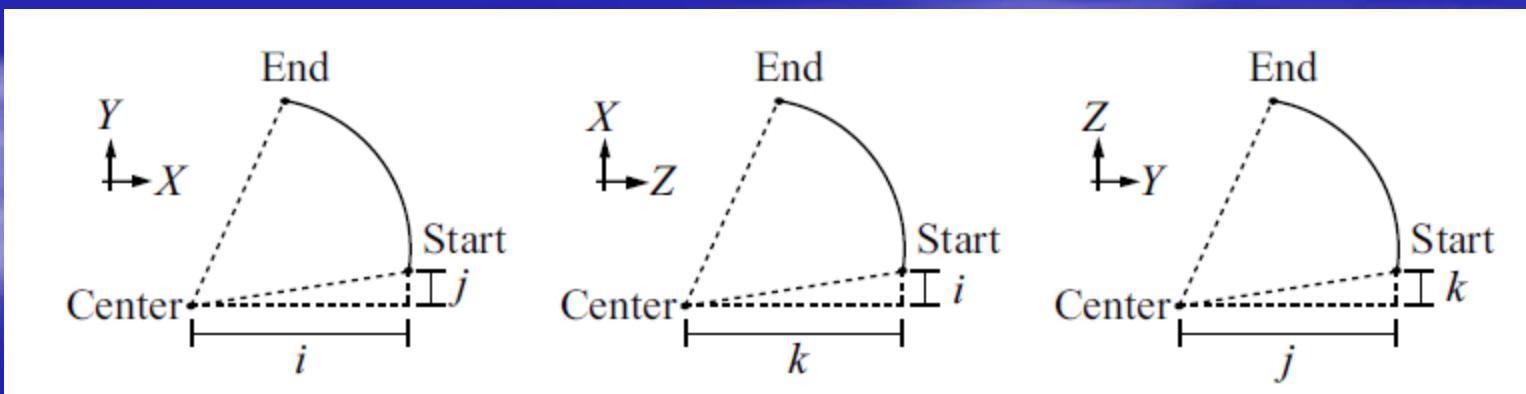
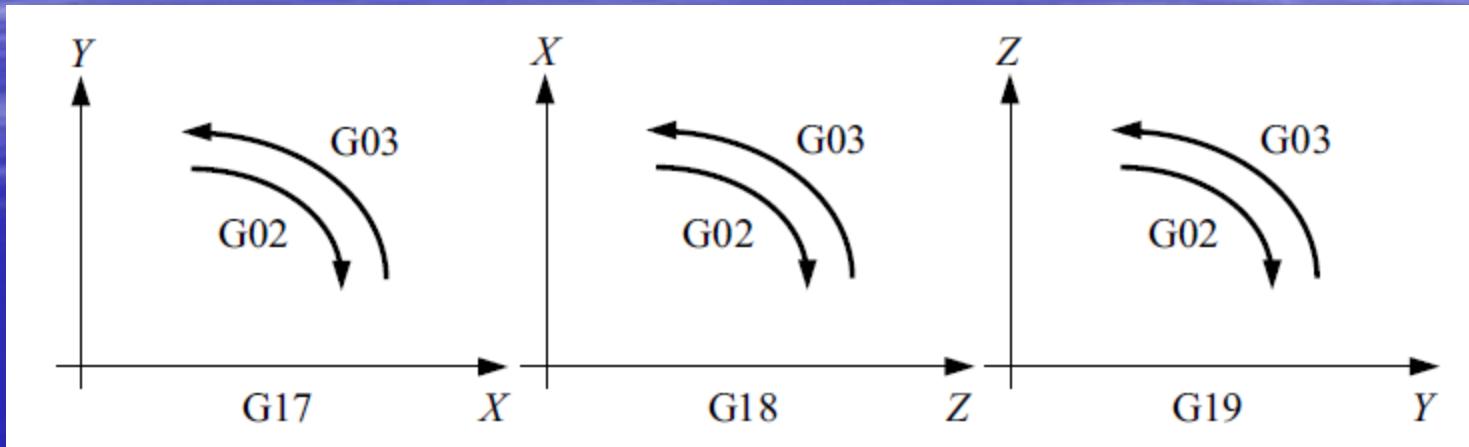
- G00 X\_ Y\_ Z\_
  - Rapid move
  - X, Y, Z is the end position

# Linear Interpolation

- G01 X\_ Y\_ Z\_ F\_
  - Linear interpolation
  - F is the feedrate
  - X, Y, Z is the end position



# Circular Interpolation

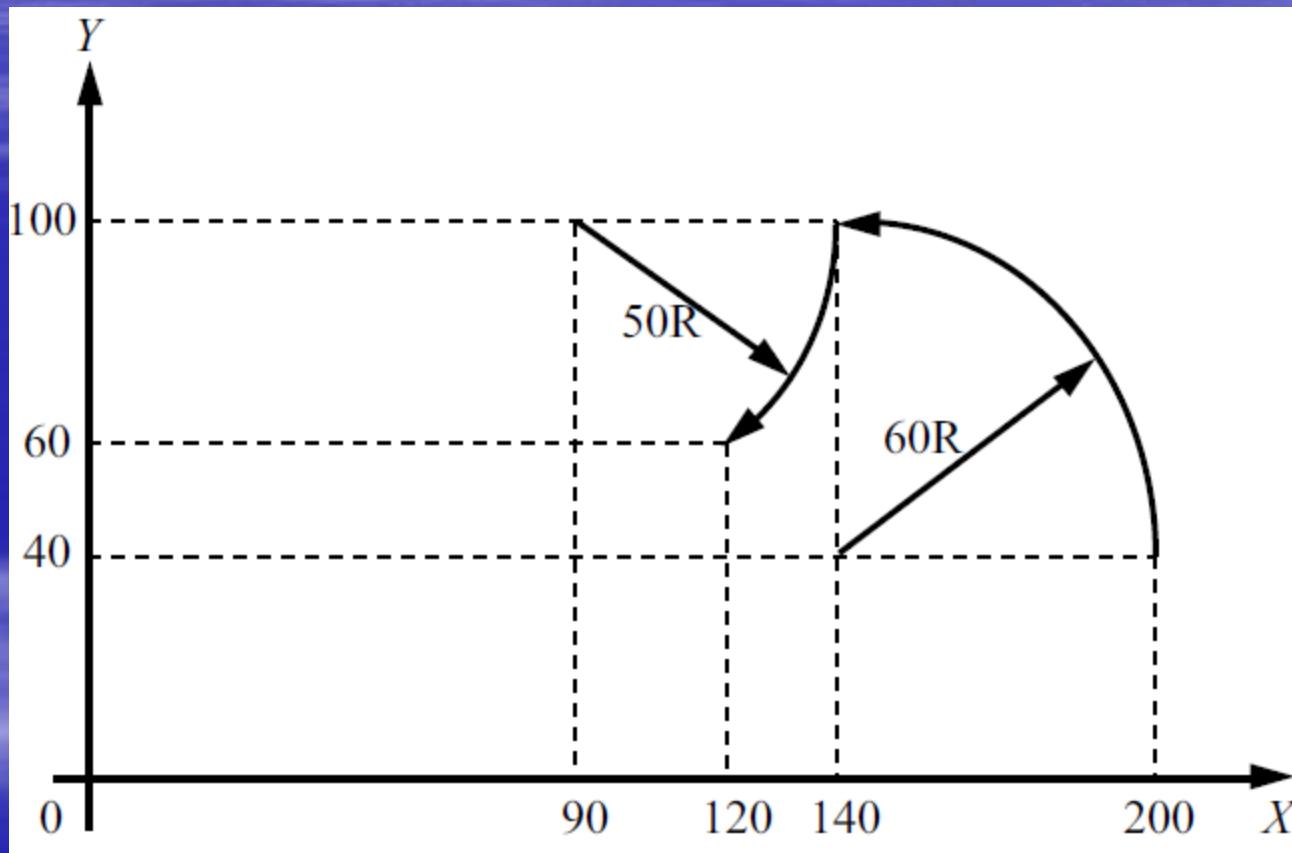


# Circular Interpolation

- G02/G03  $\alpha$   $\beta$  R F
- G02/G03  $\alpha$   $\beta$  m n F
  - G02 is circular interpolation in CW
  - G03 is circular interpolation in CCW
  - m, n is the distance from start to point and the arc center
  - R is radius of the arc
  - $\alpha$ ,  $\beta$  is the end position
  - F is the feedrate

	G17	G18	G19
$\alpha$	X	Z	Y
$\beta$	Y	X	Z
m	I	K	J
n	J	I	K

# Circular Interpolation



# Circular Interpolation

## i) Absolute programming mode

G92	X200	Y40	Z0	;	
G90	G03	X140	Y100	I-60	F300;
	G02	X120	Y60	I-50	;

or

G92		X200	Y40	Z0	;
G90	G03	X140	Y100	R60	F300;
	G02	X120	Y60	R50	;

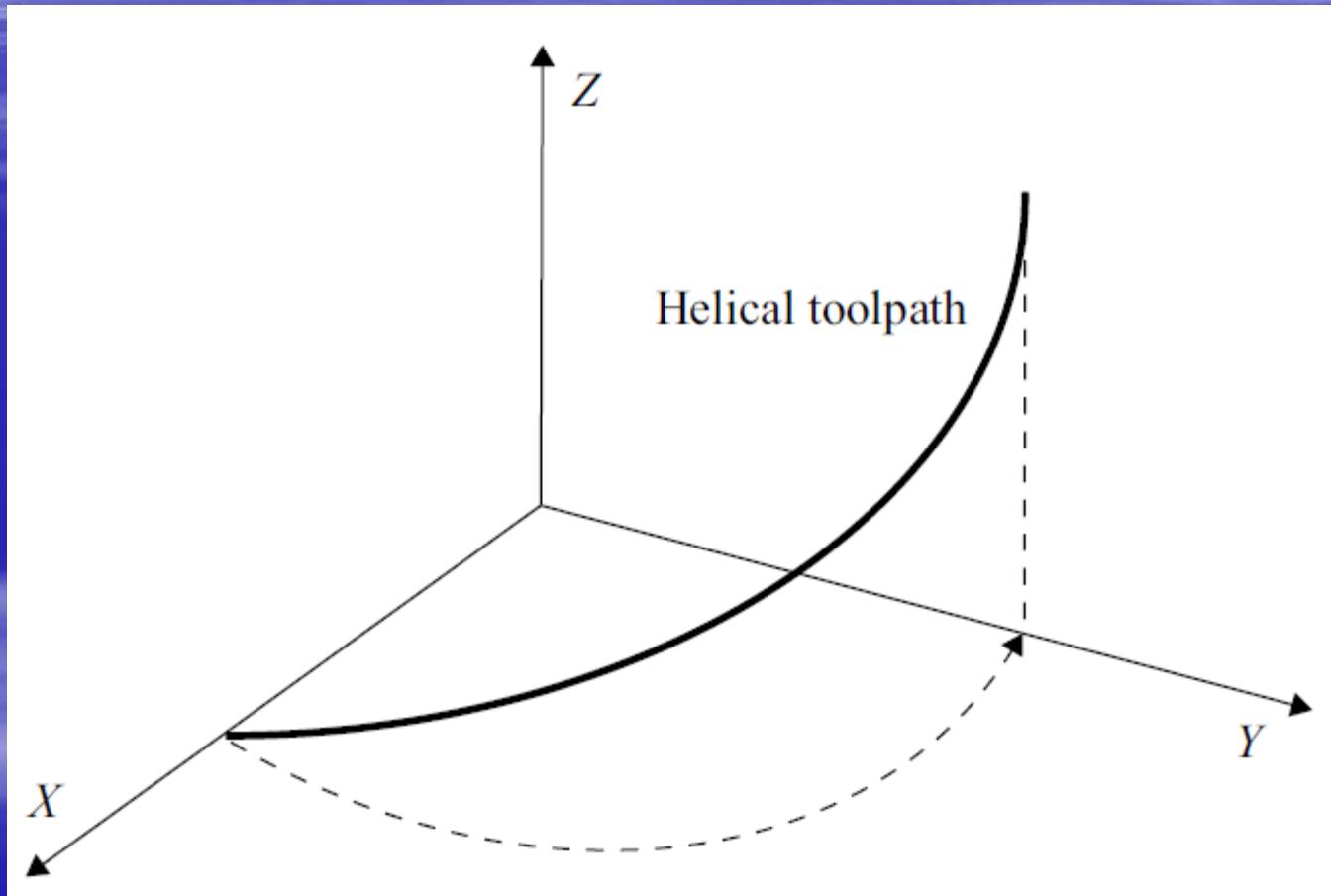
## ii) Incremental programming mode

G91	G03	X-60	Y60	I-60	F300;
	G02	X-20	Y-40	I-50	;

or

G91	G03	X-60	Y60	R60	F300;
	G02	X-20	Y-40	R50	;

# Helical Interpolation



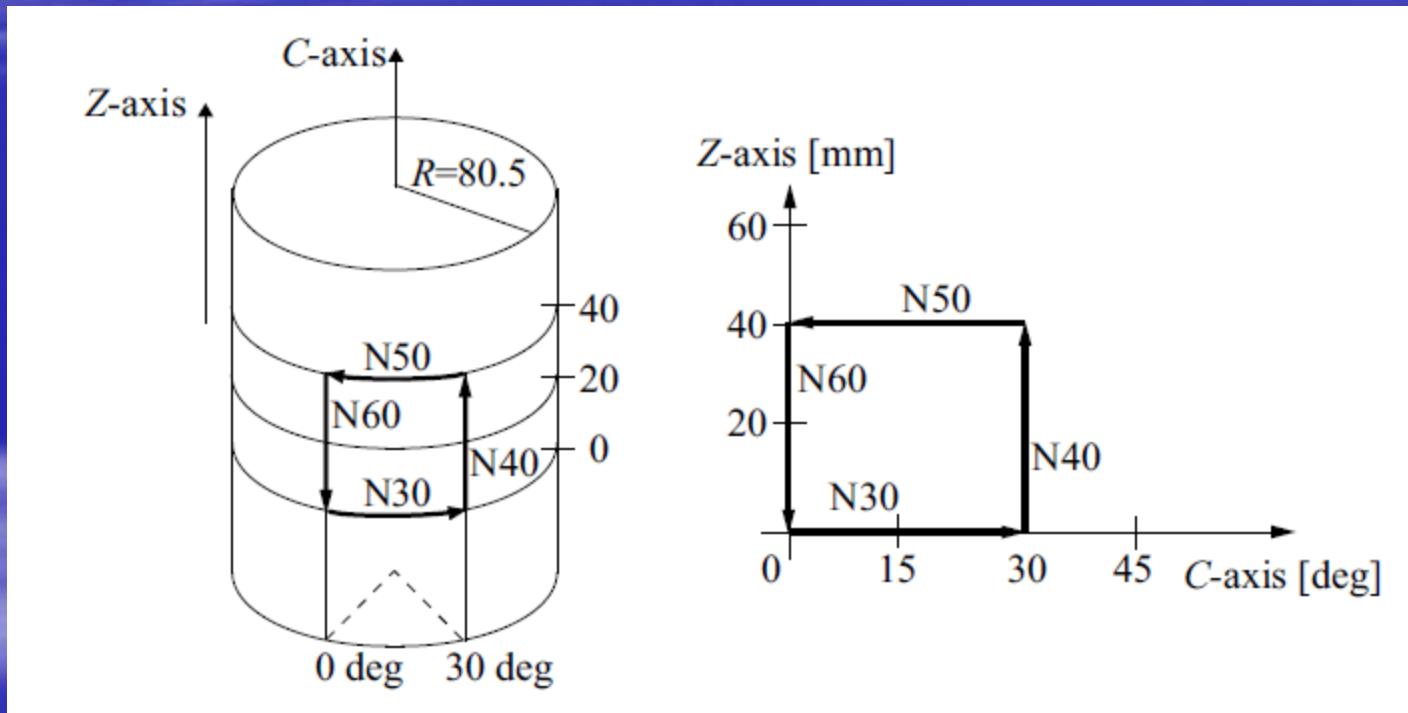
# Helical Interpolation

- G02/G03 α\_ β\_ R\_ γ\_ F\_

- Helical interpolation is enabled by specifying up to two other axes that move synchronously with the circular interpolation by circular commands.
- F is the tangential feedrate.
- γ is the end position.

	G17	G18	G19
α	X	Z	Y
β	Y	X	Z
γ	Z	Y	X

# Cylindrical Interpolation



# Cylindrical Interpolation

- G07.1 C\_

- Cylinder mode start
  - **C** is the radius of cylinder
  - When cylinder is started, you can use linear or circular interpolation on the surface of cylinder.

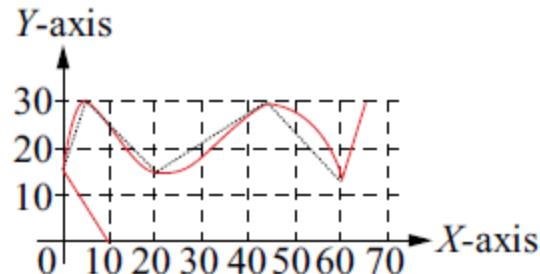
- G07.1 C0

- Cylinder mode cancel

# Spline Interpolation

## ■ G06.1

- machining free-form curves or surfaces
- passes through the specified points
- canceled by another G-Code (e.g. G00, G01, G02, G03)



```
N10 G17 G01 X10 Y0 F200 ;
N20 X0 Y15 ;
N30 G06.1 X5 Y30 ;
N50 X20 Y15 ;
N50 X45 Y30 ;
N60 X60 Y15 ;
N70 G01 X65 Y30 ; → Spline interpolation is canceled
N80 M30 ;
```

# Feed function

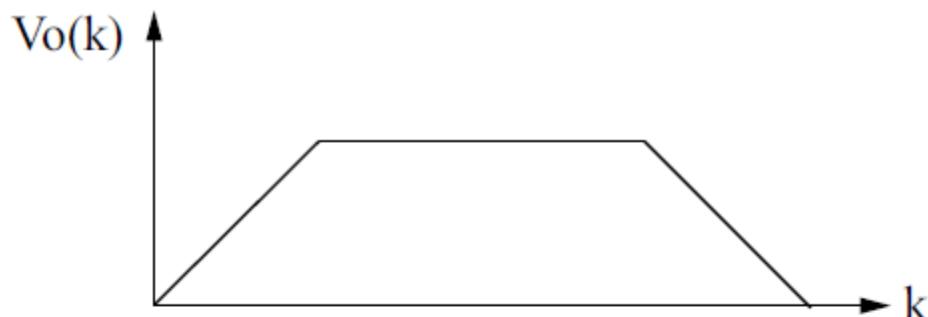
- Feedrate

- specified by the F-code
- mm/min
- inch/min
- mm/rev
- inch/rev
- Machining feedrate means the feedrate specified for linear interpolation or circular interpolation.

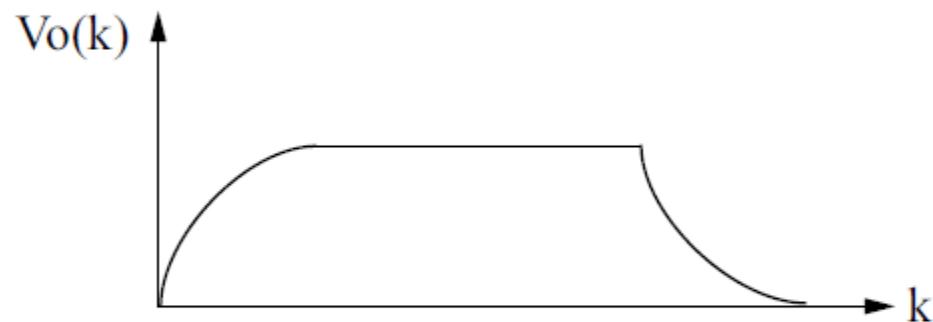
# Feed function

- acceleration/deceleration
  - when the tool starts and ends its movement.
  - prevent a mechanical shock
  - Linear-curve
  - Exponential-curve
  - S-curve
  - With servo delay

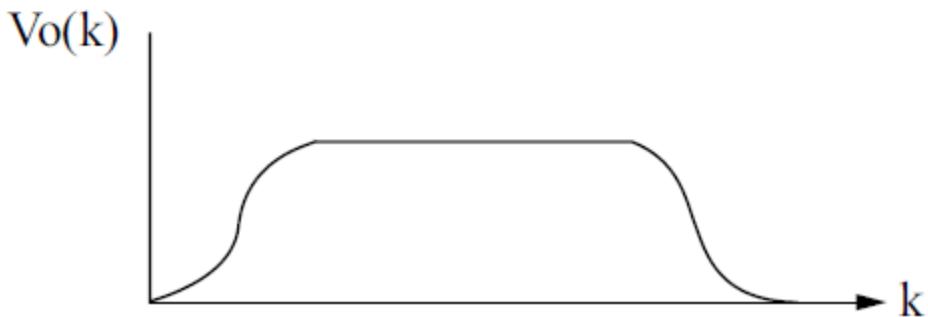
# Feed function



Linear



Exponential



S-shape

# Feed function

- G09

- Exact Stop
  - one-shot G-code
  - the tool is decelerated at the end point of the block, then an **in-position check** is made.

# Feed function

- G61

- Exact Stop Mode
- valid until G62, G63, or G64 is specified
- the tool is decelerated at the end point of the block, then an **in-position check** is made.

- G64

- Cutting Mode
- an **in-position check** is **not** made

# Feed function

- Look-Ahead function
  - under Cutting Mode
  - this function is useful for increasing the actual machining feedrate during execution of the part program which consists of small line segments.

# Feed function

- G04 X\_ (second)
- G04 U\_ (second)
- G04 P\_ (millisecond)
  - dwell function
  - Use for delaying the next block for specified time interval.

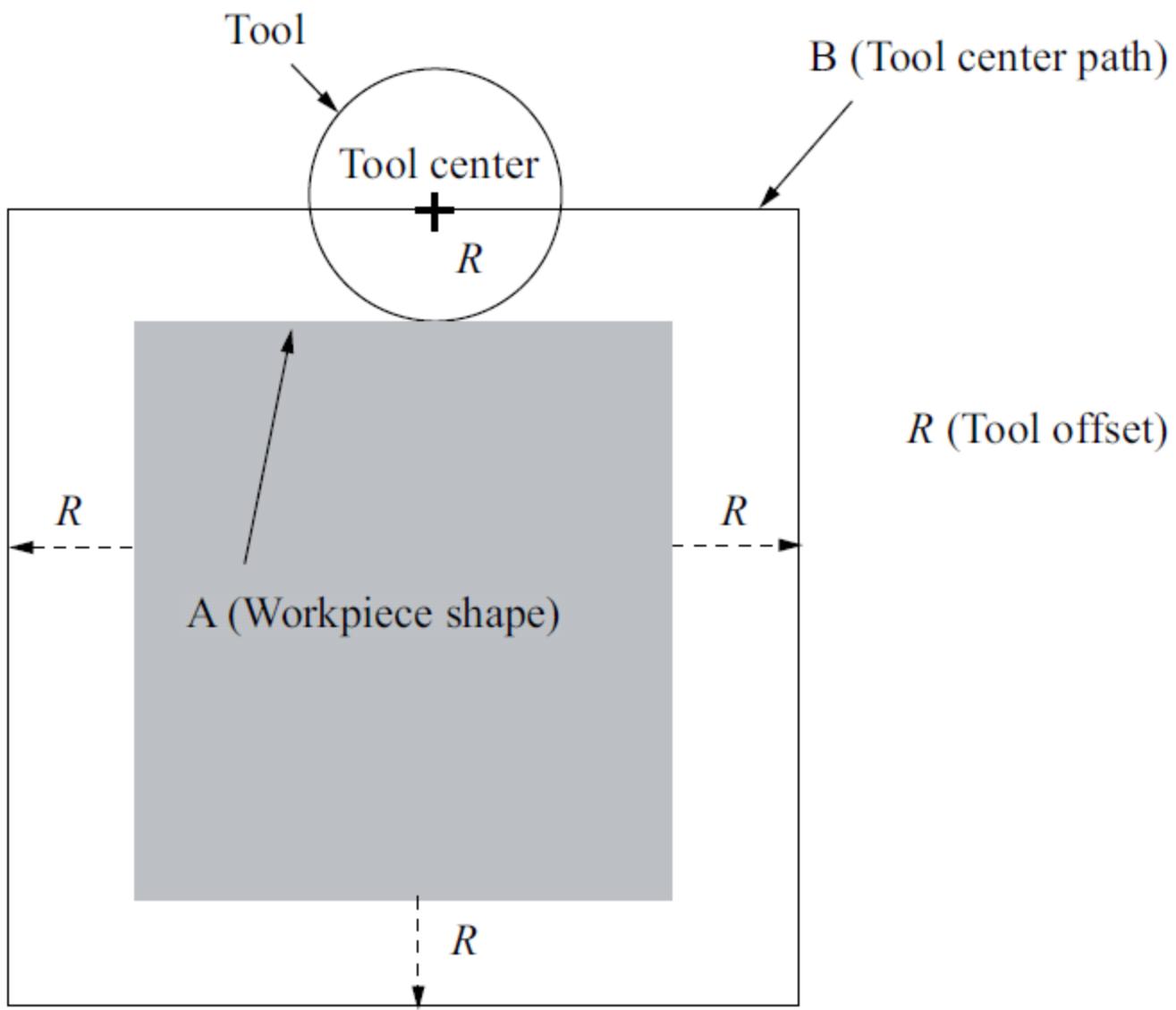
# Tool function

- T-code is used for selecting the machining tool with the specified tool number.

# Tool function

- G41 D\_
  - Tool radius compensation to the **left** of the tool movement direction
  - D is the tool number in tool compensation table
- G42 D\_
  - Tool radius compensation to the **right** of the tool movement direction
  - D is the tool number in tool compensation table
- G40
  - Tool radius compensation cancel

# Tool function



# Tool function

- G43 H\_
  - Tool length **increase**
  - H is the tool number in tool length table
- G44 H\_
  - Tool length **decrease**
  - H is the tool number in tool length table
- G49
  - Tool length compensation **cancel**

# Spindle function

- S-code is for specifying the spindle speed.
- spindle speed is restricted by the maximum spindle speed specified by user.
- The spindle speed specified by an S-code is canceled after power on, or when the system is reset or when M30 is commanded.

# Spindle function

- G96 S\_
  - Constant surface speed function
  - S is the surface speed. (m/min, inch/min)
  
- G97
  - Constant surface speed function cancel

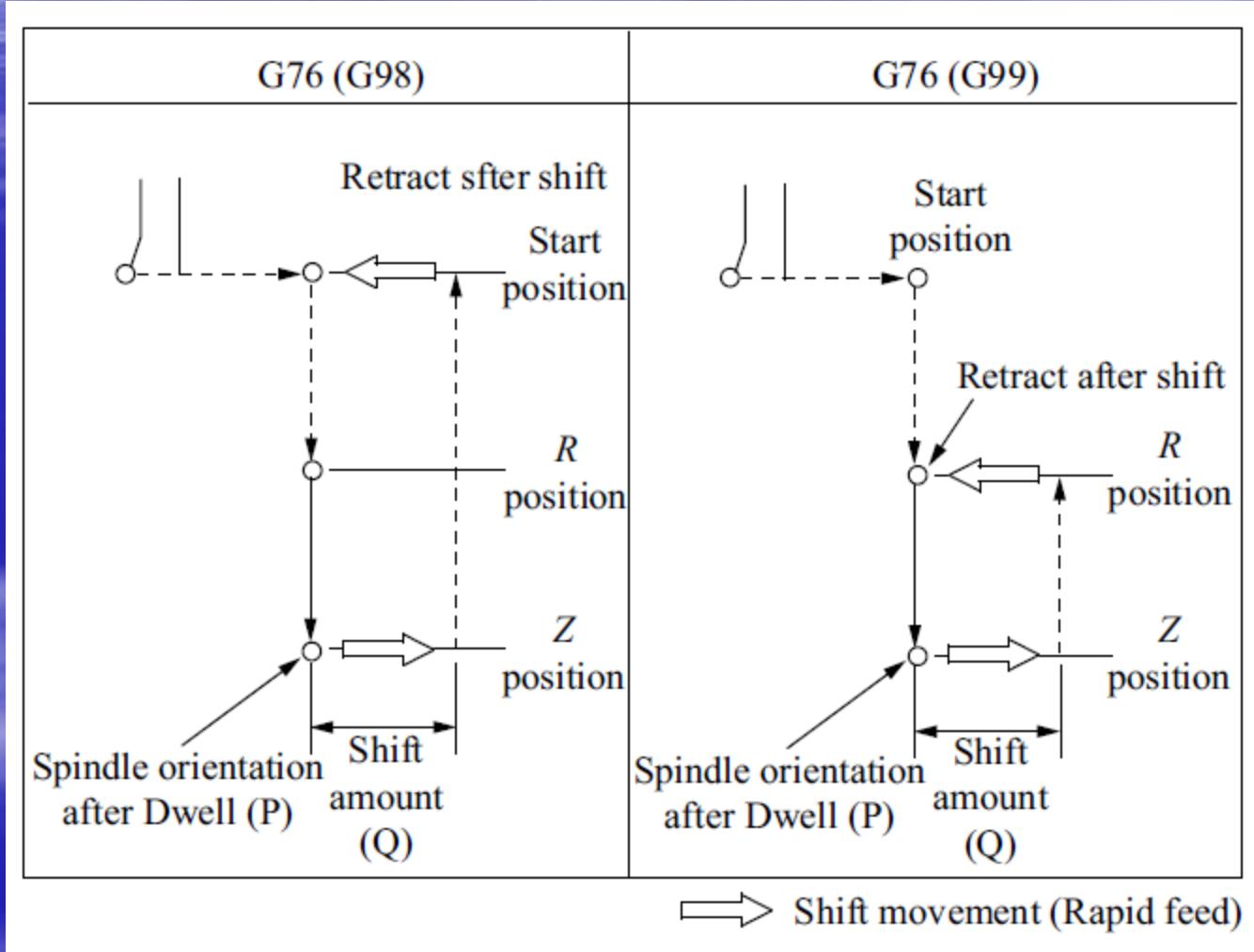
# Spindle function

- M19 S\_
  - Spindle orientation function
  - Spindle stop at S angle

# Fixed-cycle function

Operation	G-code	Operation	G-code		
Drilling	Peck Drilling	G73	Turning	Roughing	G90
	Reverse Tapping	G74		Threading	G92
	Fine Boring	G76		Face roughing	G94
	Cycle Cancel	G80		Finishing	G70
	Drilling Cycle, Spot Boring	G81		Roughing	G71
	Drilling Cycle, Count Boring	G82		Face roughing	G72
	Peck Drilling	G83		Copying	G73
	Tapping	G84		Grooving	G74
	Rigid Tapping	G84.2		Face grooving	G75
	Reverse Rigid Tapping	G84.3		Multiple threading	G76
	Boring	G85	Milling	Circular	
	Boring	G86		Elongated Holes	
	Back Boring	G87		Circular	
	Boring	G88		Circumferential Slot	
	Boring	G89		Facing	
				Circular Pocket	

# Fixed-cycle function



# Fixed-cycle function

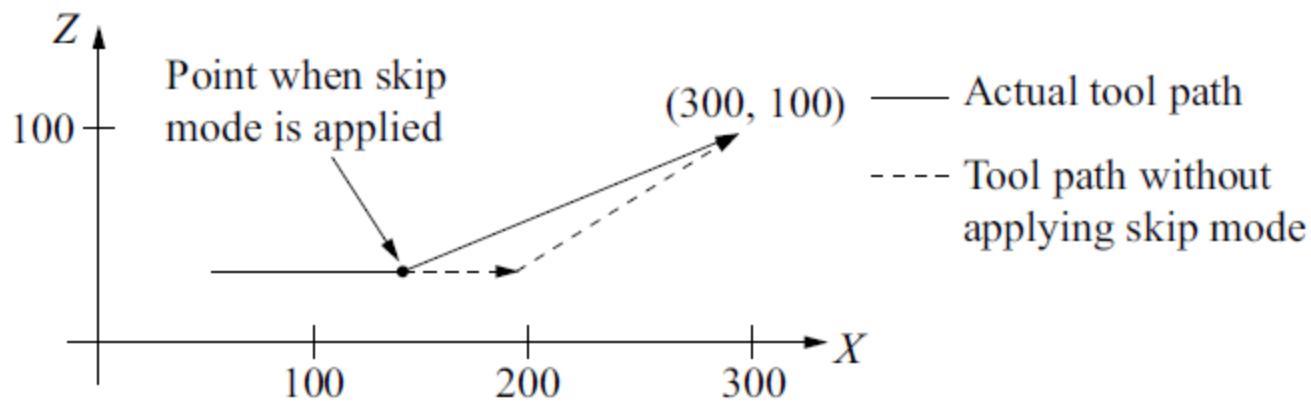
- G76 Z\_ R\_ P\_ Q\_ F\_
  - Fine boring cycle
- G98
  - Return to initial Z level in canned cycle
- G99
  - Return to R level in canned cycle

# Skip function

- G31 X\_ Y\_ Z\_ F\_
  - Same as the G01 function
  - a skip signal is detected, the is moved to the end point of the next block

G90 G31 X200.0 F100 ;

X300.0 Z100.0 ; → The tool is moved to the point of the next block.



# Program verification

- The part program error
  - grammatical errors
  - logic errors
  - numerical errors
  - incorrect computation of tool position
  - wrong tool-offset value
  - invalid feedrate
  - spindle speed

# Program verification

- Dry mode
  - the tool is moved at the feedrate specified by a parameter
  - the workpiece is removed from the table.

# Program verification

- Machine lock
  - display the change in position
  - without moving the tool
  - two types of machine lock
    - all-axis machine lock
    - specified-axis machine lock

# Program verification

- Block-by-block
  - can be used with the **dry run** function and **machine lock** function

# Advanced function

- Recently, CNC machine tools have become more **accurate** and **faster** and the functionality has become more complicated.
- The **highly functional** CNC systems
  - Look Ahead
  - Feedforward
  - NURBS Interpolation

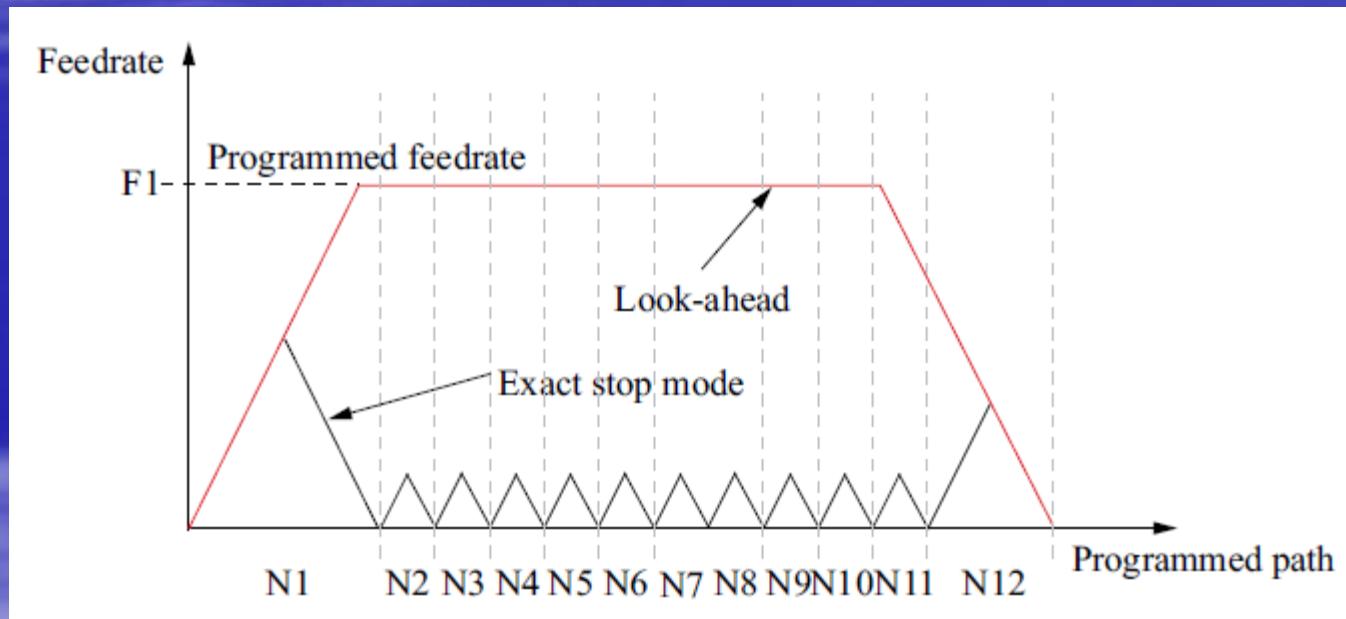
# Look Ahead

- When machine the free-form surfaces/contours
  - consists of **a sequential linear path** with short length
  - if each block is executed line by line
    - the machined surface is degraded due to **frequent acceleration/deceleration** and **the discontinuity of the feedrate**

# Look Ahead

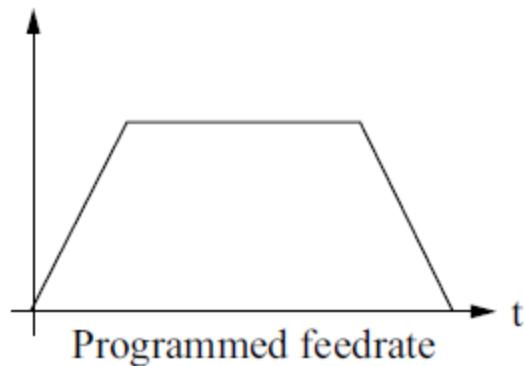
- When machine the free-form surfaces/contours
  - consists of **a sequential linear path** with short length
  - if use Look-Ahead function
    - Preview hundred/thousand blocks
    - calculates an **adequate feedrate** for each axis within the maximum allowable feedrate and **acceleration/deceleration**.

# Look Ahead

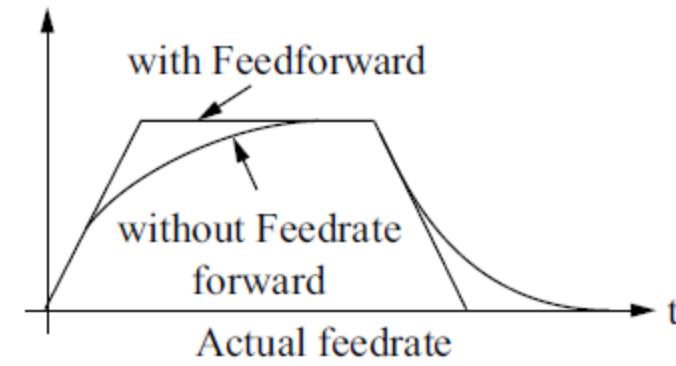


# Feedforward

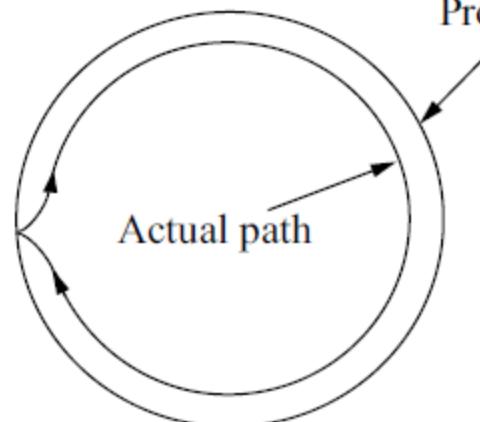
Feedrate



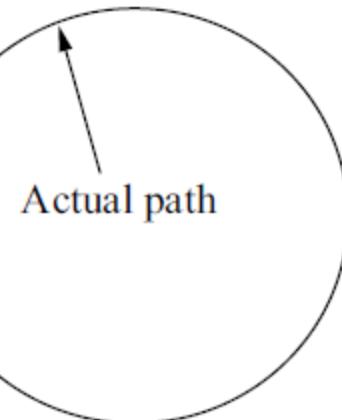
Feedrate



Programmed path



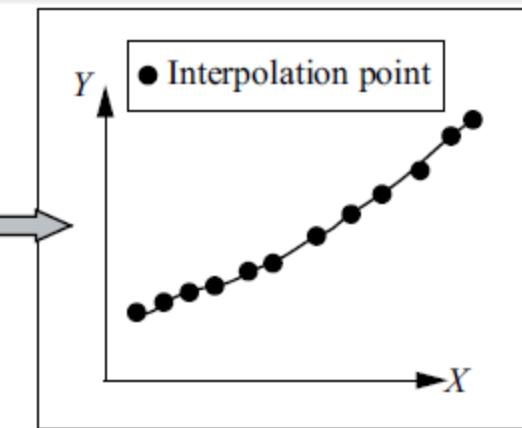
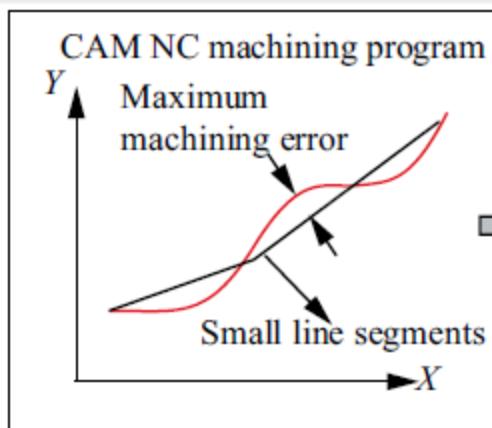
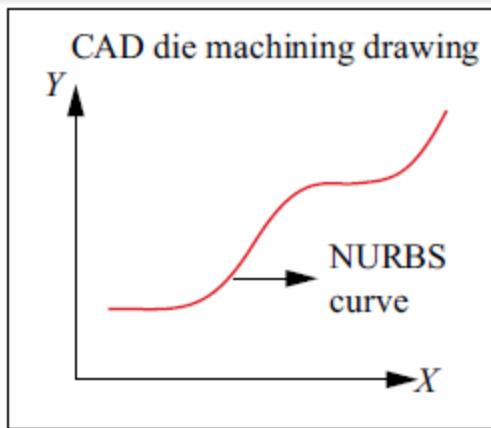
without Feedforward



with Feedforward

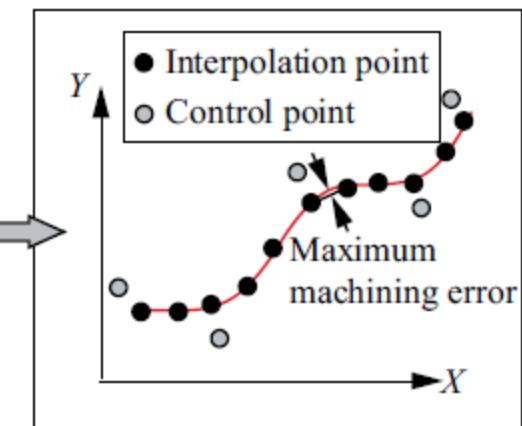
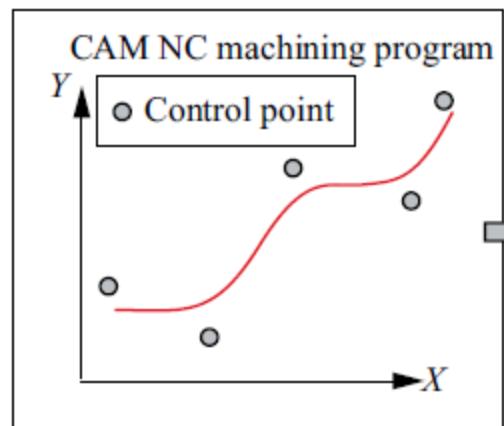
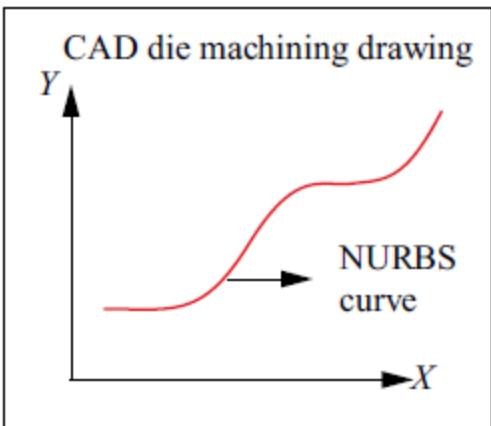
# NURBS Interpolation

- In conventional CNC systems, free curve is defined by **sequential small line segments or arcs.**
  - the size of the **internal memory** is limited
  - the **baud rate** of communication is restricted  
(raise the machining speed)
- To overcome this problem NURBS interpolation was developed.



Small line segment data  
(Large amount of data)

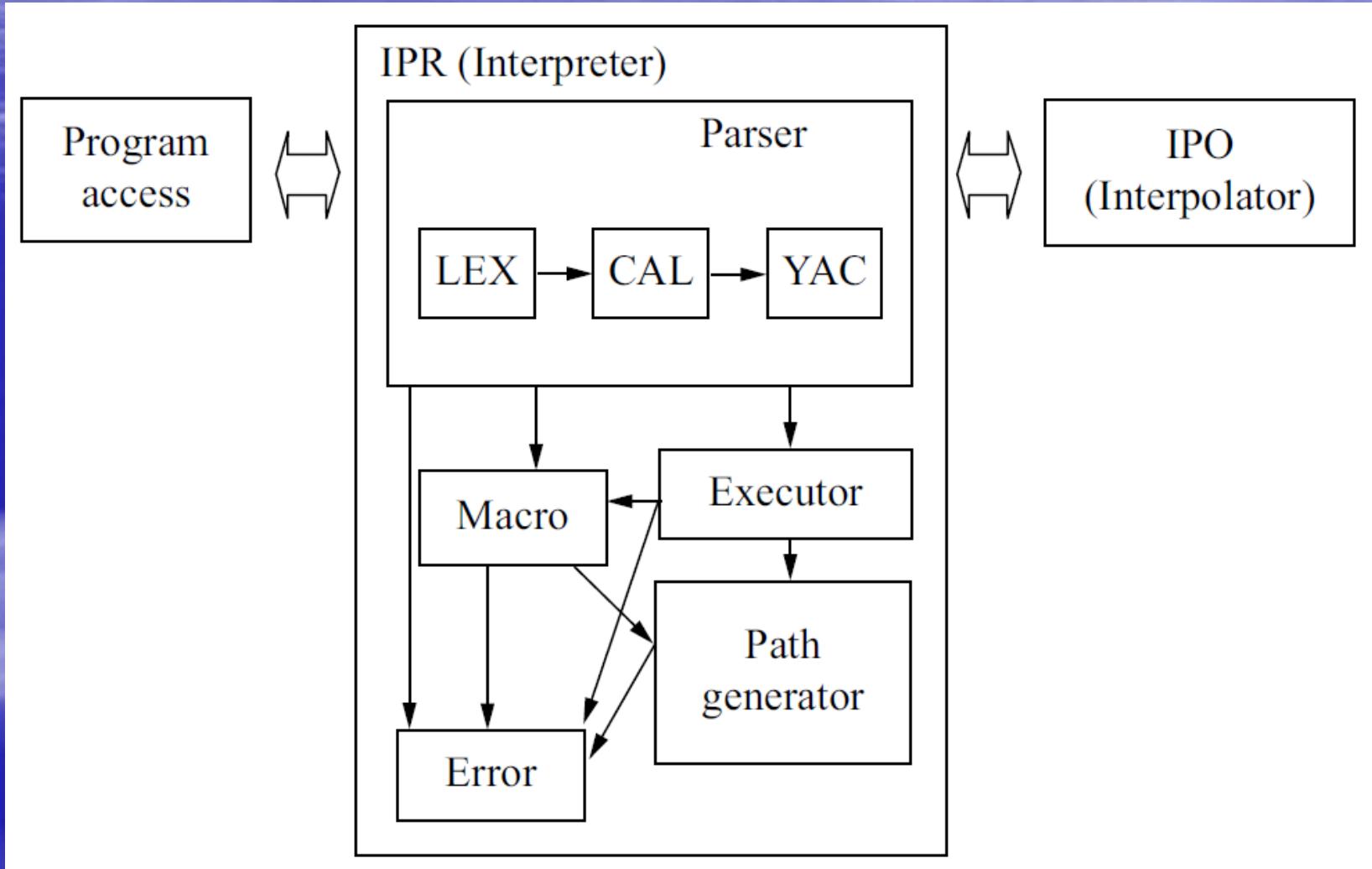
(a) Interoperation method based on small line segment approximation



Control point  
Knot  
Weight (Small amount of data)

(b) Interoperation method based on NURBS curve

# G&M-code Interpreter



# G&M-code Interpreter

- **Interpreter**

- Parser
- Executor
- Path generator
- Macro executor
- Error handler

# Parser

- Lexical interpreter
  - reads the block character by character and makes meaningful words from the characters
- Calculator
  - numerical operations within the part program
- Sentence interpreter
  - retrieves the command and the related data
    - such as G-code, M-code, S-code, T-code

# Executor

- executes the functions related with the interpreted sentence
- stores the execution result in the internal memory

# Path Generator

- generates the position data based on the programmed coordinates
- In this module, the computation for mapping from workpiece coordinates to machine coordinates, tool compensation, and the axis limit is carried out.

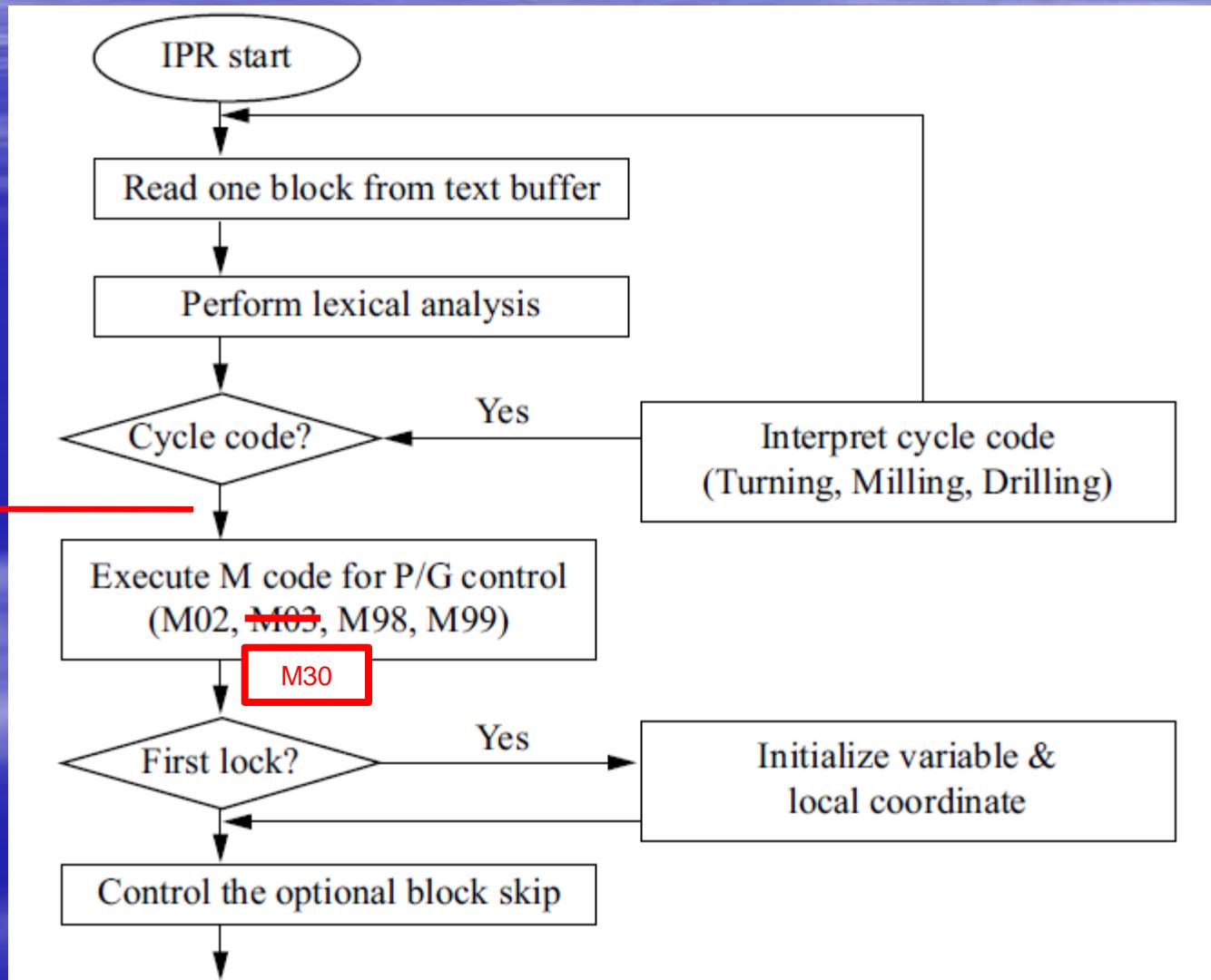
# Macro executor

- interprets and executes macro commands
- the macro is **user-defined** code
  - not provided by the CNC maker

# Error Handler

- if there is an error in a part program, the error should be noticed and the user notified.

# Interpreter Flowchart



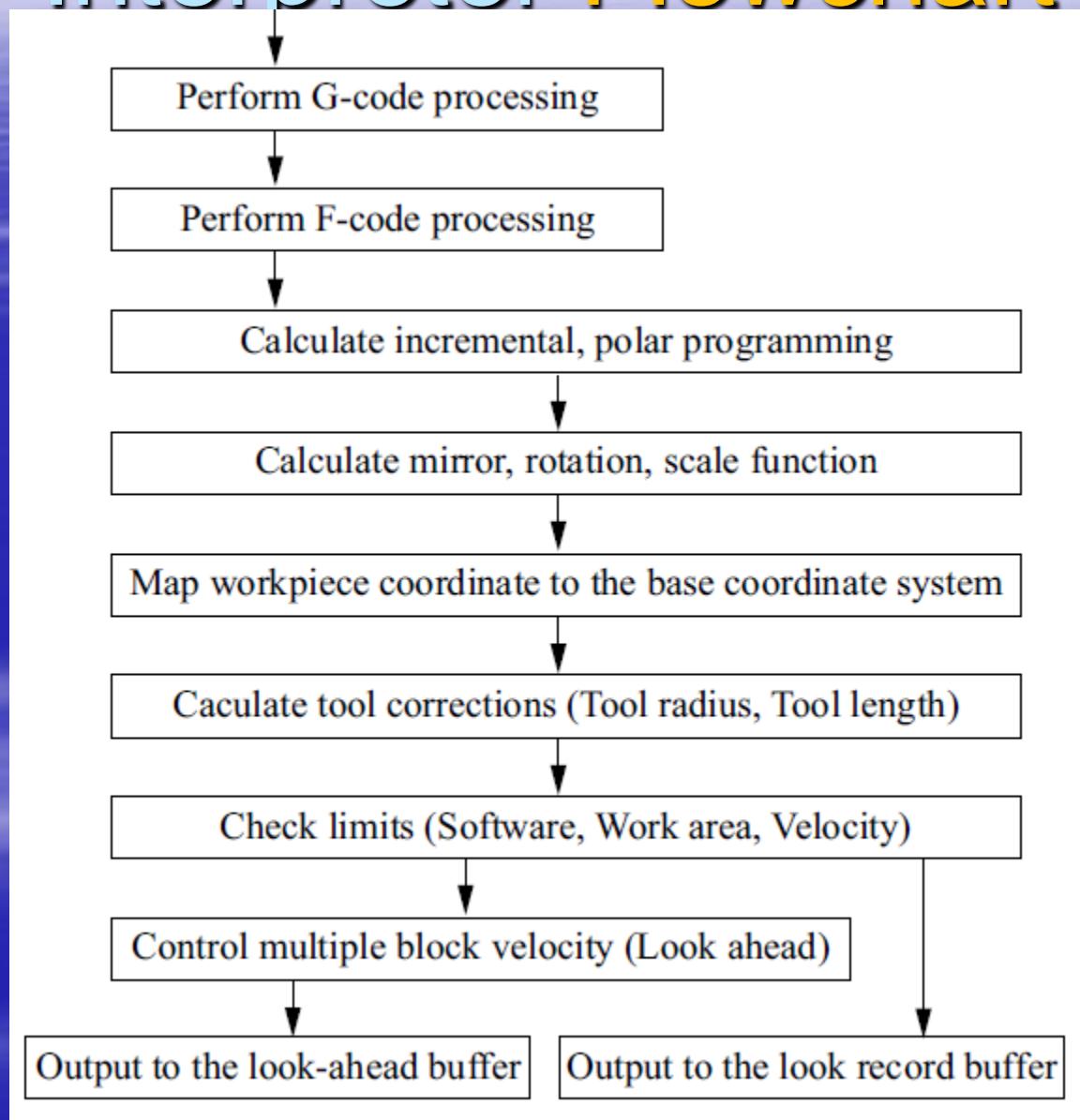
# Block Memory

Field	Type	Value	State	Type	Value
A	double		SLASH	unsigned	1
B	double		PERCENT	unsigned	
C	double		NUMBER	unsigned	
D	double		ID	unsigned	
E	double		CHANGE[M_ADD]	unsigned	
F	double	1000	GFLAG[MAX-G]	unsigned	
G	double	01	MFLAG[MAX-G]	unsigned	
H	double	G_MODAL	int		
I	double				
J	double				
K	double				
L	double				
M	double				
N	double	300			
O	double				
P	double				

# Block Memory

Q	double				
R	double				
S	double	5000			
T	double				
U	double				
V	double				
W	double				
X	double	150			
Y	double	20			
Z	double	30			
P9000 N100 G92 G96; N200 G00 X100; N300 M03; /N300 G01 X150 Y20 Z30 F1000 S5000; N400 G1 Z120;					

# Interpreter Flowchart



Block  
record  
memory

# Block record memory

Entity	Type	Comments
End_point	Real Array[8]	Block end point
Start_point	Real Array[8]	Block start point
Prog_F_value	Real	Feedrate
Prog_S_value	Real	Spindle speed
F_limit	Real	Feedrate limit
S_limit	Real	Spindle speed limit
Scaling_factor	Real	Scale
G_code_group	Integer Array[20]	G-code group
Exact_stop	Integer	Exact stop mark
Block_number	Integer	Block number
Sub_PG_index	Integer	Subprogram index

# Block record memory

Act_Number_PG_repeat	Integer	Number of program repetitions
Gear_box	Real	Parameter for gear box
Correction_active	Integer	Tool compensation start
Tool_correction	Real	Tool compensation
IPO_type	Integer	Interoperation type
G_Code	Integer Array[8]	G-code in block
Thread_flag	Integer	Thread start
Handwheel_flag	Integer	Handwheel start
Handwheel_direction	Integer	Handwheel rotation direction
Block_length	Real	Block length
Block_pointer	Pointer Array[4]	Block pointer

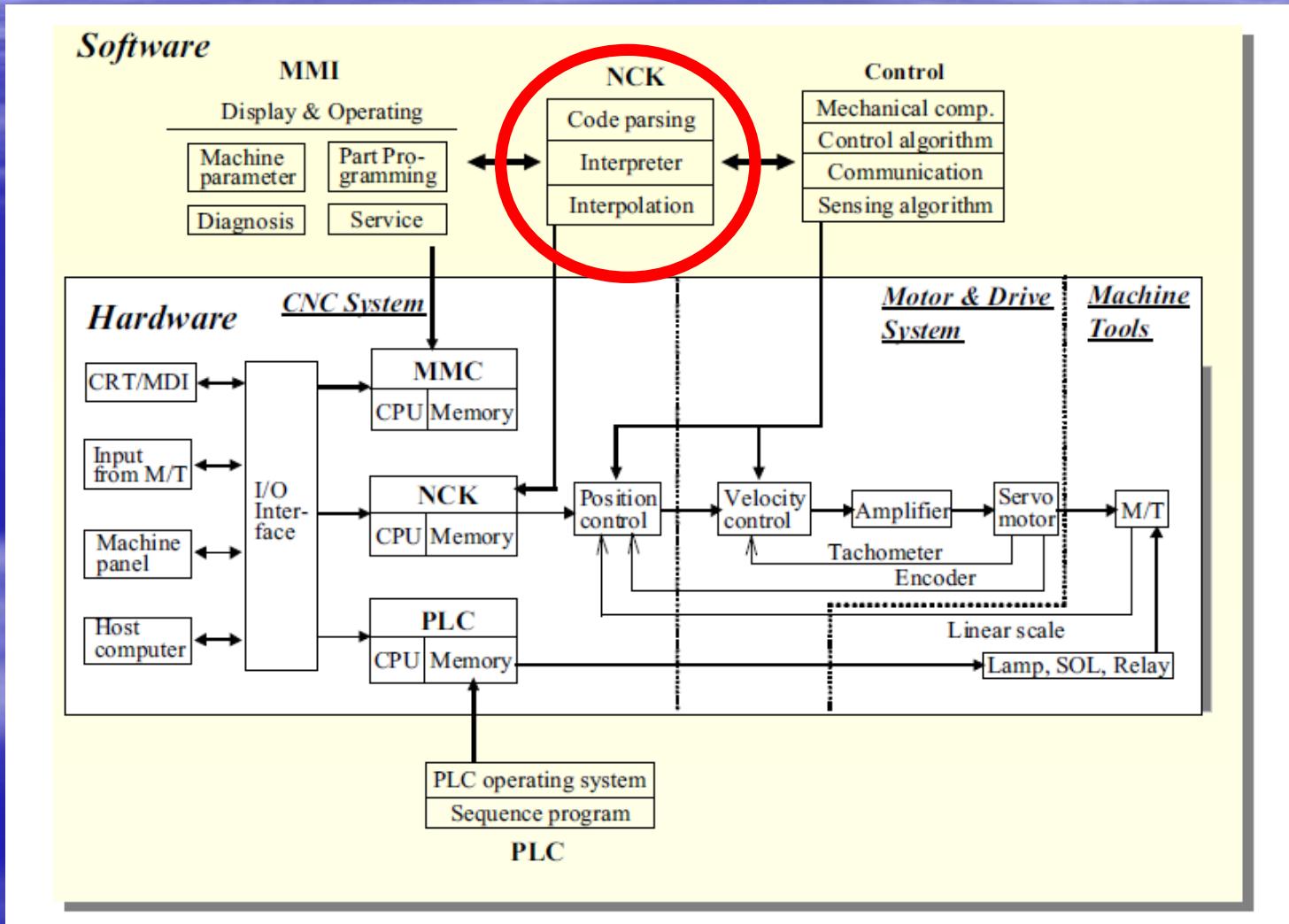
# Theory and Design of CNC Systems

*Chapter 3*  
**Interpolator**

# Out Line

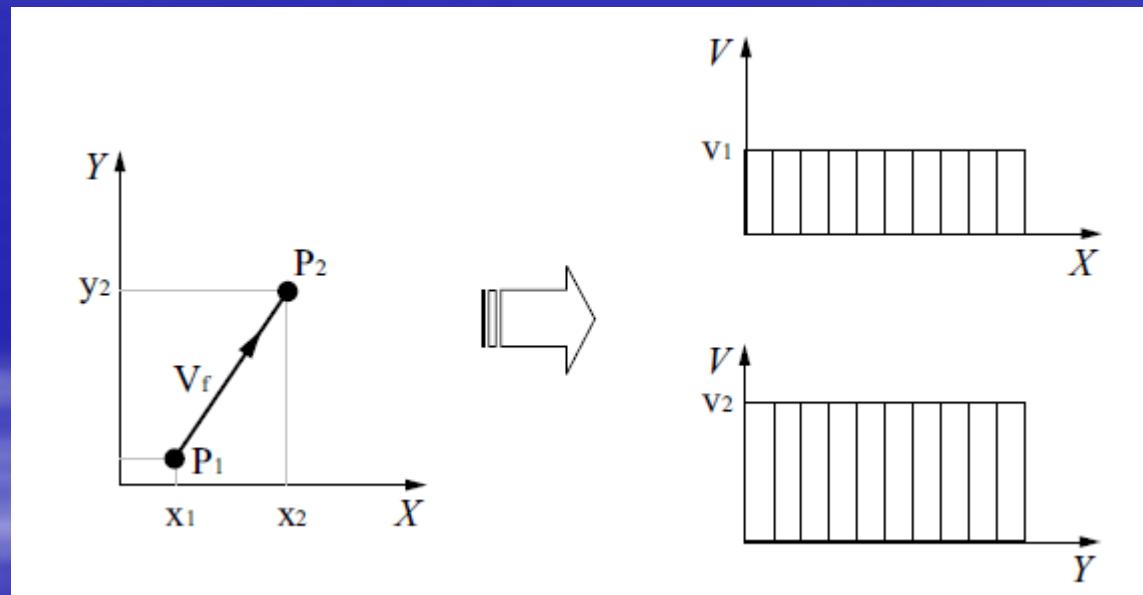
- Introduction
- Hardware Interpolator
- Software Interpolator
  - Reference pulse interpolator
  - reference word interpolator (Sampled-Data interpolator).
- Fine Interpolation
- NURBS Interpolation
- Summary

# Introduction



# Introduction

- Key components of CNC



# Introduction

- The interpolator requires the following characteristics
  - 1. Close to the actual part shape.
  - 2. Consider the limitation of speed .
  - 3. Accumulation error should be avoided.
- Two kinds of control can be carried out
  - 1. point-to-point control method
  - 2. contour control method

# Hardware Interpolator

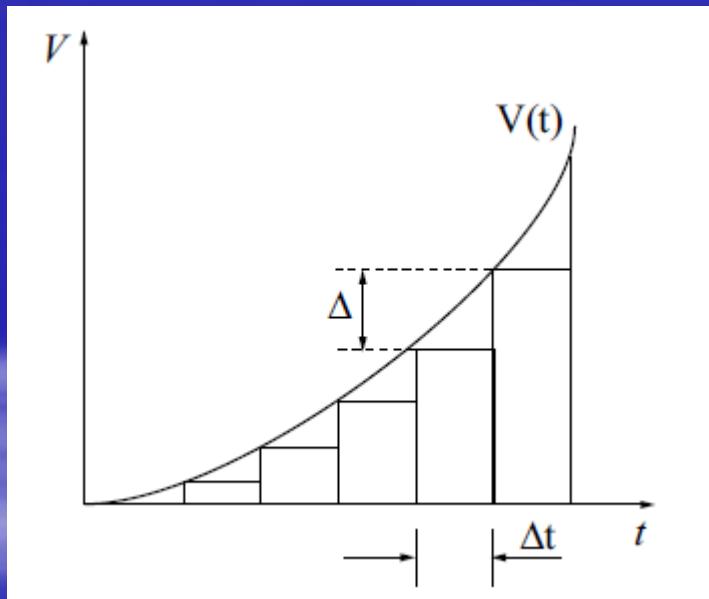
- Computation of interpolation
- Generates pulses (electric circuit)
- High-speed execution
- Be difficult to adapt new algorithms or modify algorithms
- EX : NC

# Hardware Interpolator

- DDA (Digital Differential Analyzer) integrator
- Based on the principle of a numerical integration.

# Hardware Interpolator

principle of interpolation



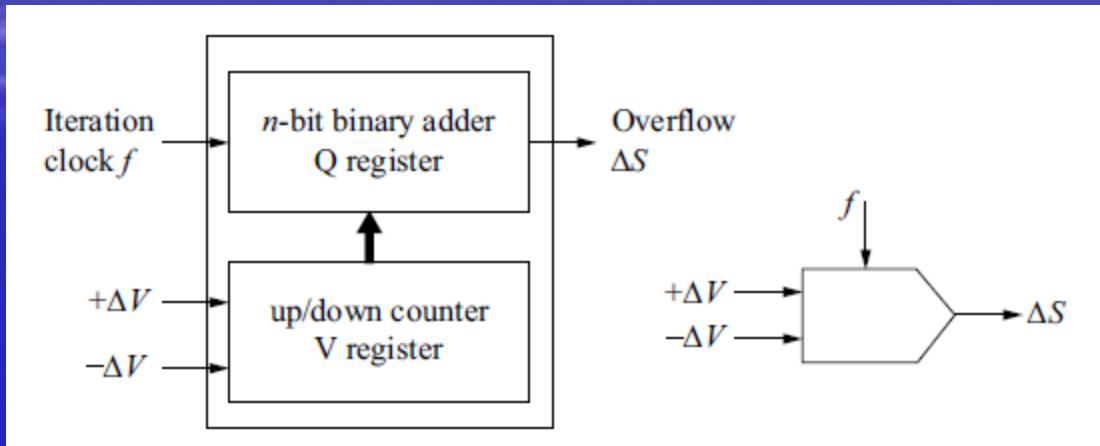
$$S(t) = \int_0^t V \cdot dt \cong \sum_{i=1}^k V_i \cdot \Delta t$$

$$S_k = \sum_{i=1}^{k-1} V_i \cdot \Delta t + V_k \cdot \Delta t$$

$$S_k = S_{k-1} + \Delta S_k$$

$$\Delta S_k = V_k \cdot \Delta t$$

# Hardware Interpolator



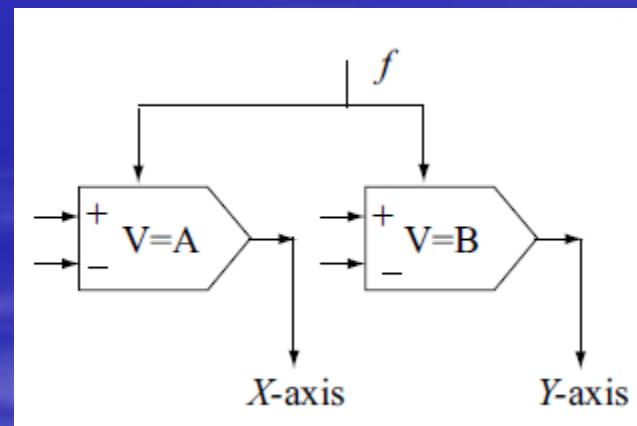
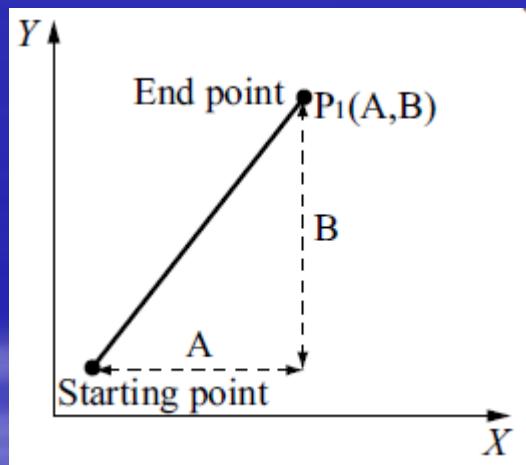
$$f = \frac{1}{\Delta t}$$

$$\Delta S_k = 2^{-n} \cdot V_k$$

$$\Delta S_k = 2^{-n} \cdot V_k \cdot (f \cdot \Delta t) = \frac{f}{2^n} \cdot V_k \cdot \Delta t$$

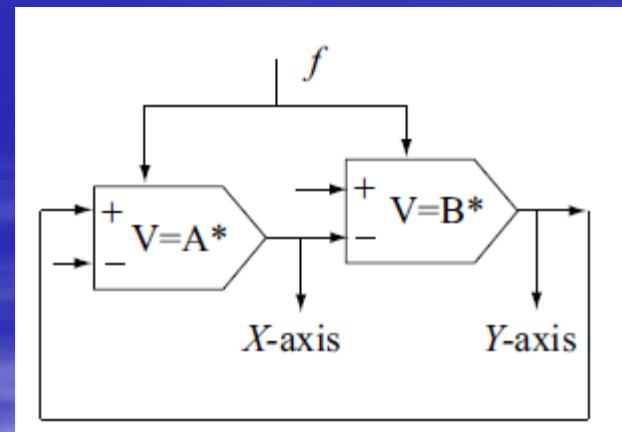
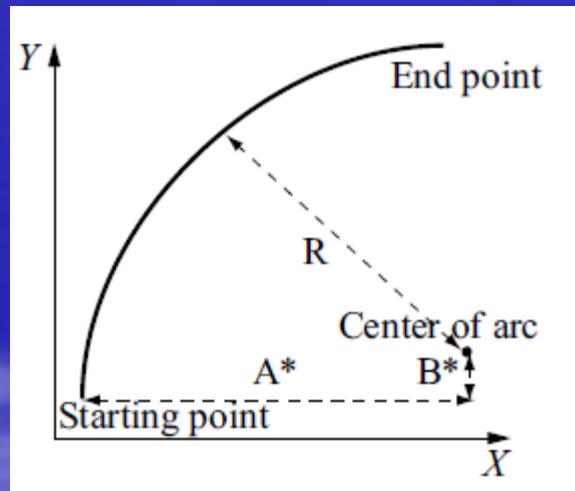
# Hardware Interpolator

- Linear interpolation



# Hardware Interpolator

- circular interpolator



# Hardware Interpolator

## ■ Example :

- radius ( $R$ ) of the circle is 15.
- length ( $n$ ) of the register is 4.
- start position of a circle is  $(R, 0)$ .
- initial value of the V registers of the DDA for the X- and Y-axes are 15 and 0.

Iteration step	DDA Integrator for X-axis			DDA Integrator for Y-axis		
	V	Q	$\Delta S$	V	Q	$\Delta S$
0	15	0		0	0	
1	15	15		0	0	
2	15	14	1	1	1	
3	15	13	1	2	3	
4	15	12	1	3	6	
5	15	11	1	4	10	
6	15	10	1	5	15	
7	15	9	1	6	5	1
8	14	7	1	7	12	
9	14	5	1	8	4	1
10	13	2	1	9	13	
11	13	15		9	6	1
12	12	11	1	10	0	1
13	11	6	1	11	11	
14	11	1	1	12	7	1
15	10	11		12	3	1
16	9	4	1	13	0	1
17	8	12		13	13	
18	8	4	1	14	11	1
19	7	11		14	9	1
20	6	1	1	15	8	1
21	5	6		15	7	1
22	4	10		15	6	1
23	3	13		15	5	1
24	2	15		15	4	1
25	1	0		15	3	1

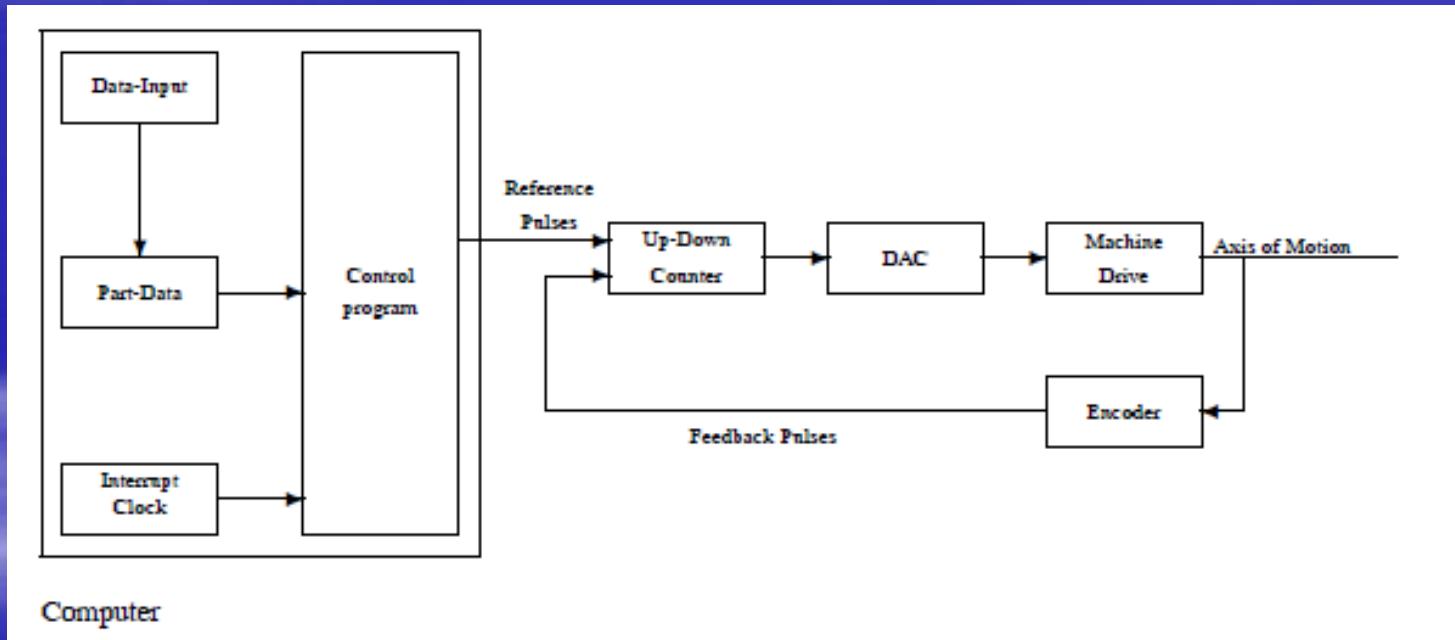
# Software Interpolator

- Reference pulse interpolator
- reference word interpolator (Sampled-Data interpolator)

Reference Pulse Method	Sampled Data Method
Software DDA Interpolator	Euler Interpolator
Stairs Approximation Interpolator	Improved Euler Interpolator
Direct Search Interpolator	Taylor Interpolator Tustin Interpolator Improved Tustin Interpolator

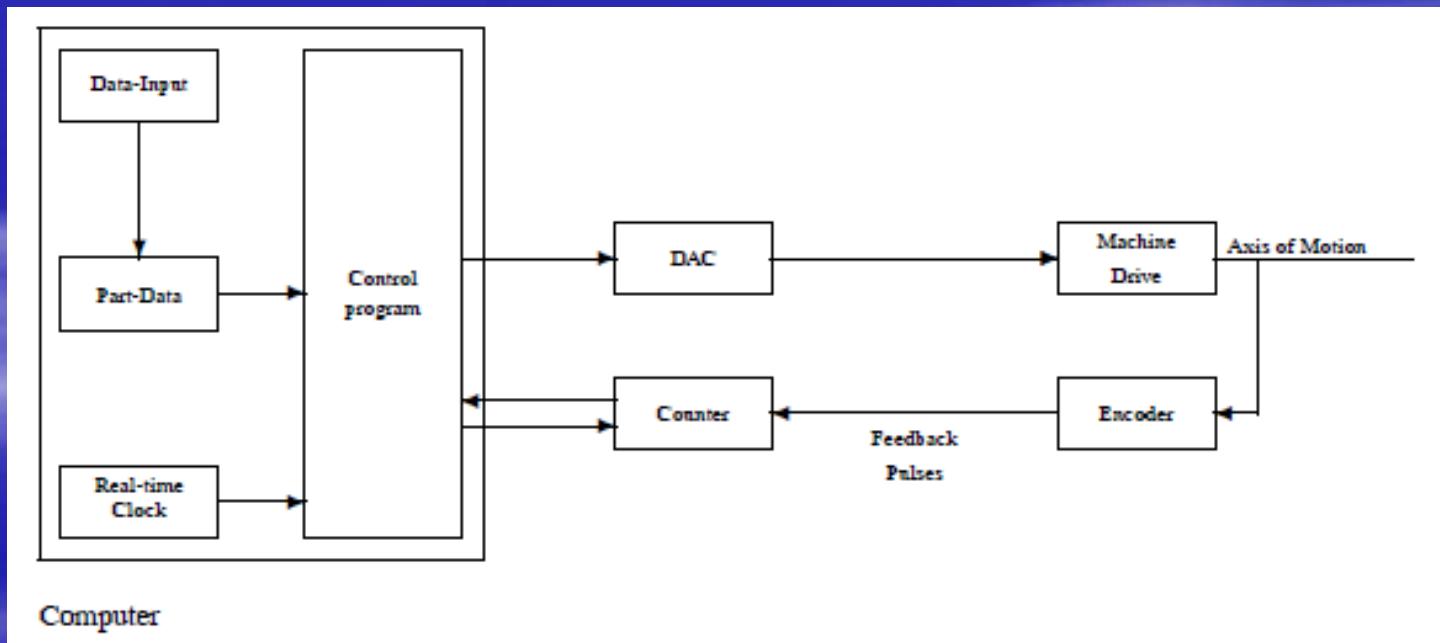
# Software Interpolator

- Reference pulse method



# Software Interpolator

- Sampled-Data interpolator
  - rough interpolation
  - fine interpolation



# Software Interpolator

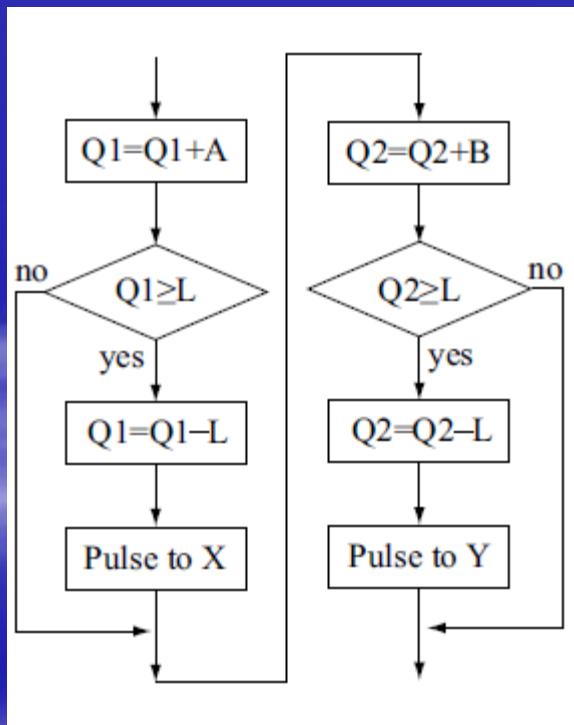
## ■ Comparison

	Reference Pulse Method	Sampled-Data Method
Description	The distance to go is computed with external reference pulses and the calculated pulses are fed to each axis.	The coordinate points to go per unit sampling time are computed and the calculated data is transmitted to each axis.
Strengths	Adequate for high-accuracy machining.	Adequate for high-speed machining.
Weakness	Inadequate for high-speed machining.	Inadequate for high-accuracy machining.
Remarks	Because the feedrate of each axis depends on the frequency of external interrupt signal, a high performance CPU is required.	

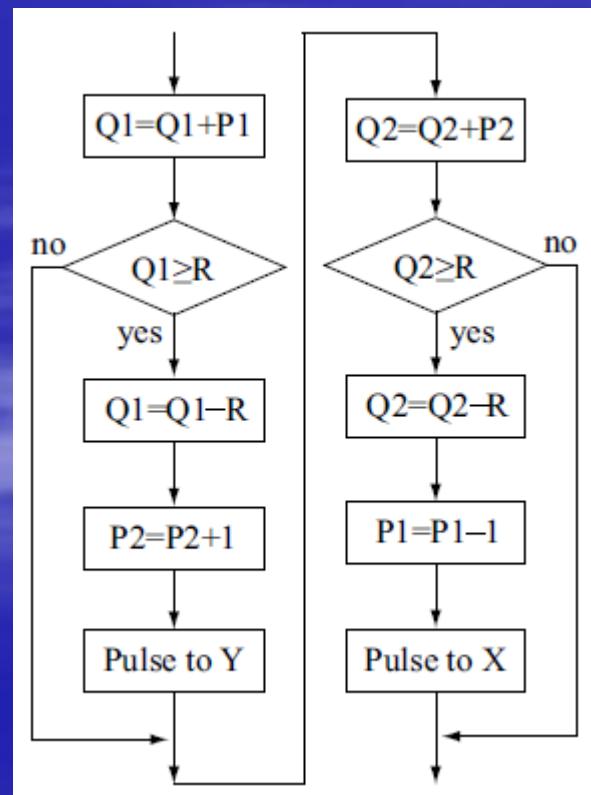
# Reference pulse method

- Software DDA Interpolator

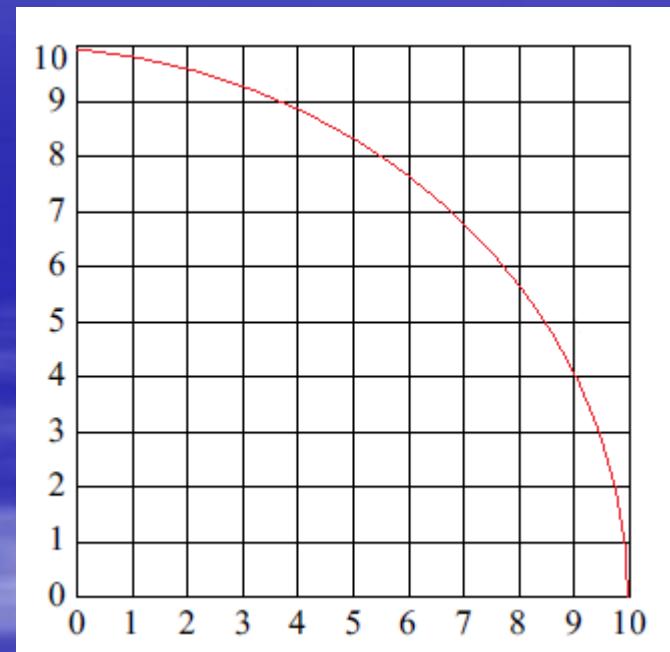
Linear



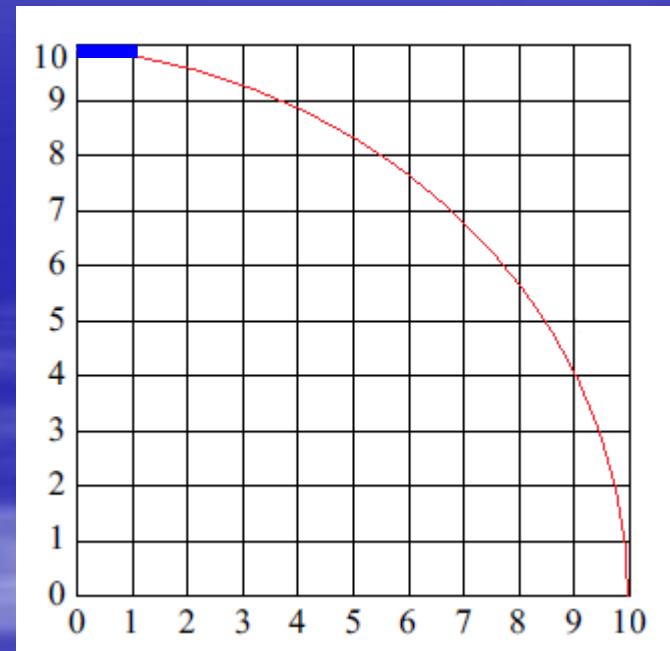
Circular



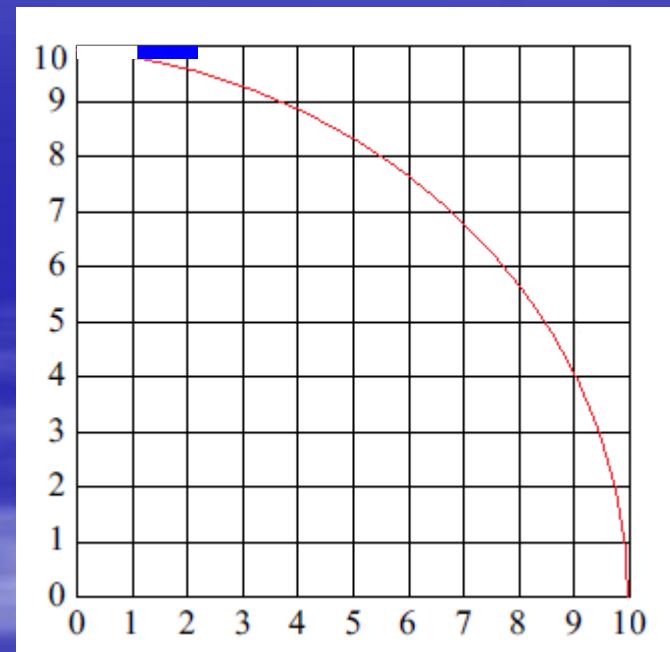
# Software DDA Interpolator



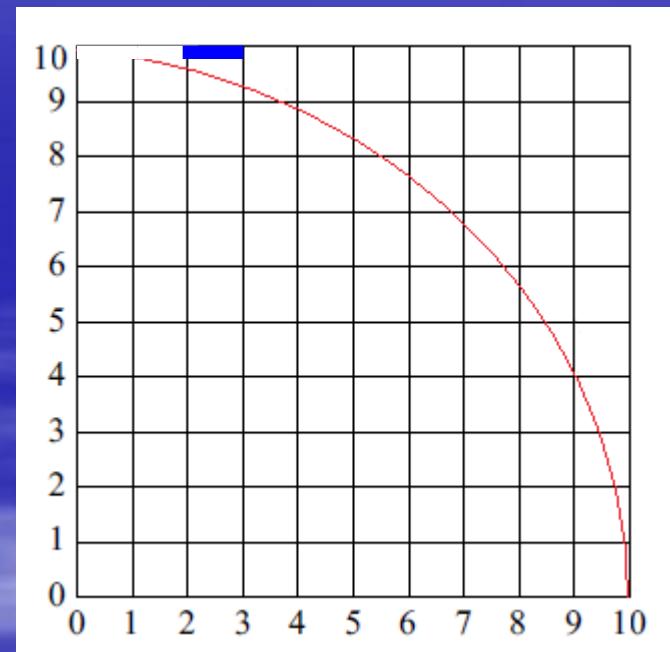
# Software DDA Interpolator



# Software DDA Interpolator

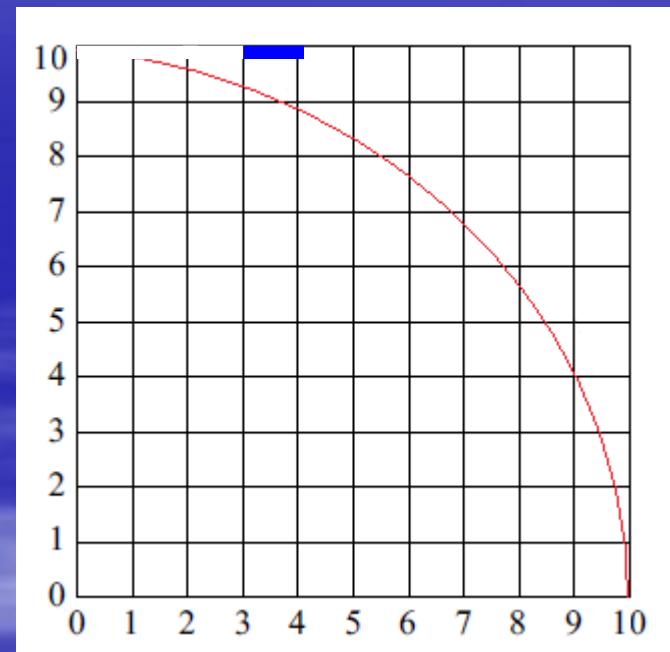


# Software DDA Interpolator



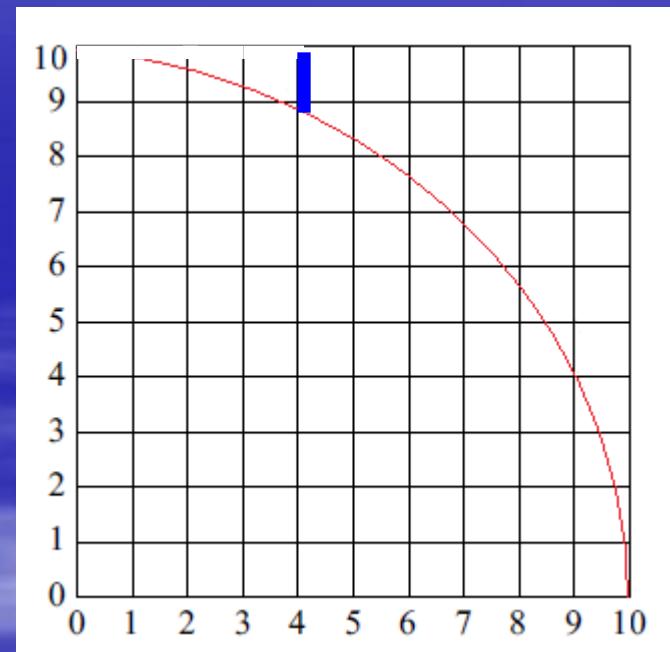
# Software DDA Interpolator

$P_1$	$Q_1$	$\Delta Y$	$P_2$	$Q_2$	$\Delta X$
0	0		10	0	
1	0	X	10	0	<input type="radio"/>
2	1	X	10	0	<input type="radio"/>
3	3	X	10	0	<input type="radio"/>
4	6	X	10	0	<input type="radio"/>



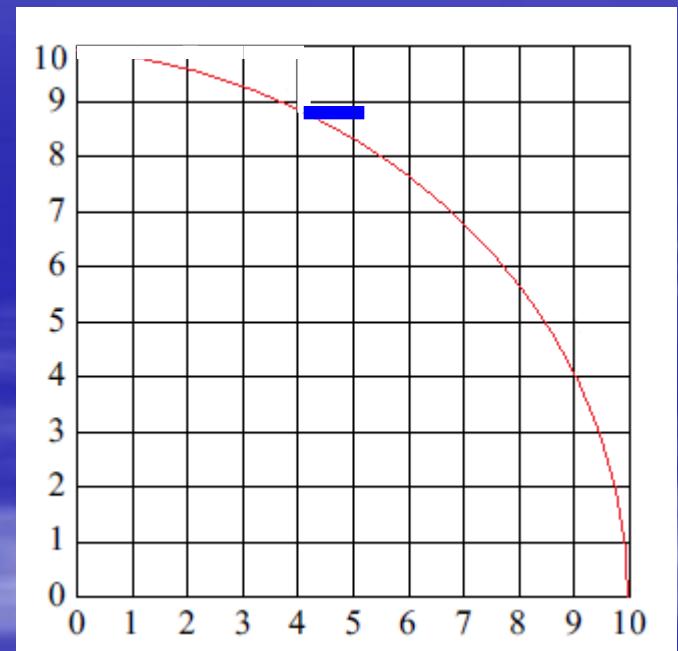
# Software DDA Interpolator

$P_1$	$Q_1$	$\Delta Y$	$P_2$	$Q_2$	$\Delta X$
0	0		10	0	
1	0	X	10	0	O
2	1	X	10	0	O
3	3	X	10	0	O
4	6	X	10	0	O
4	0	O	9	9	X



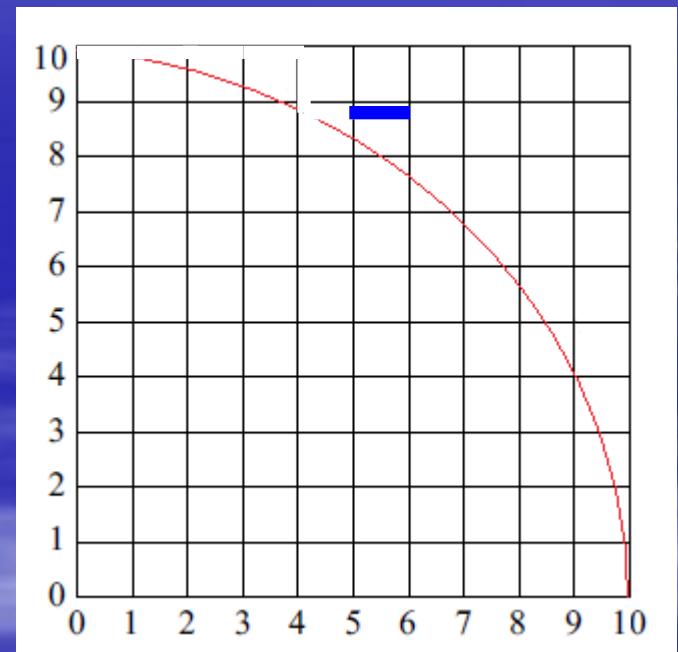
# Software DDA Interpolator

P <sub>1</sub>	Q <sub>1</sub>	ΔY	P <sub>2</sub>	Q <sub>2</sub>	ΔX
0	0		10	0	
1	0	X	10	0	○
2	1	X	10	0	○
3	3	X	10	0	○
4	6	X	10	0	○
4	0	○	9	9	X
5	4	X	9	8	○



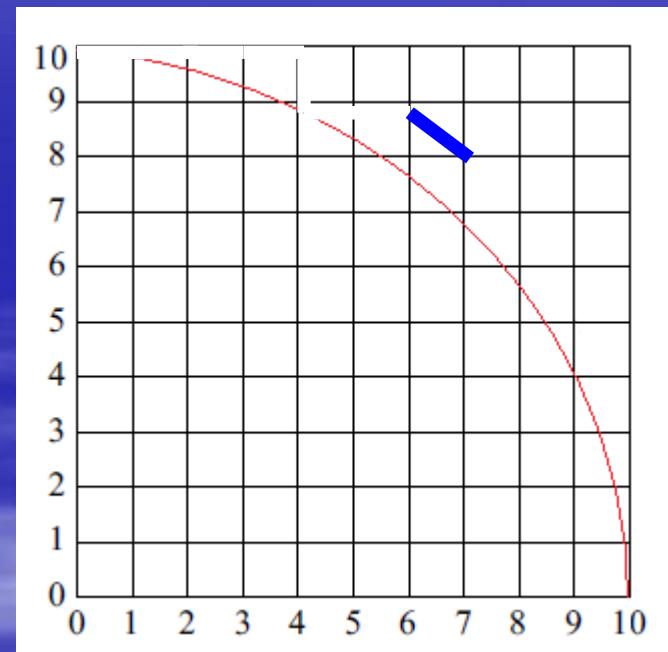
# Software DDA Interpolator

P <sub>1</sub>	Q <sub>1</sub>	ΔY	P <sub>2</sub>	Q <sub>2</sub>	ΔX
0	0		10	0	
1	0	X	10	0	○
2	1	X	10	0	○
3	3	X	10	0	○
4	6	X	10	0	○
4	0	○	9	9	X
5	4	X	9	8	○
6	9	X	9	7	○



# Software DDA Interpolator

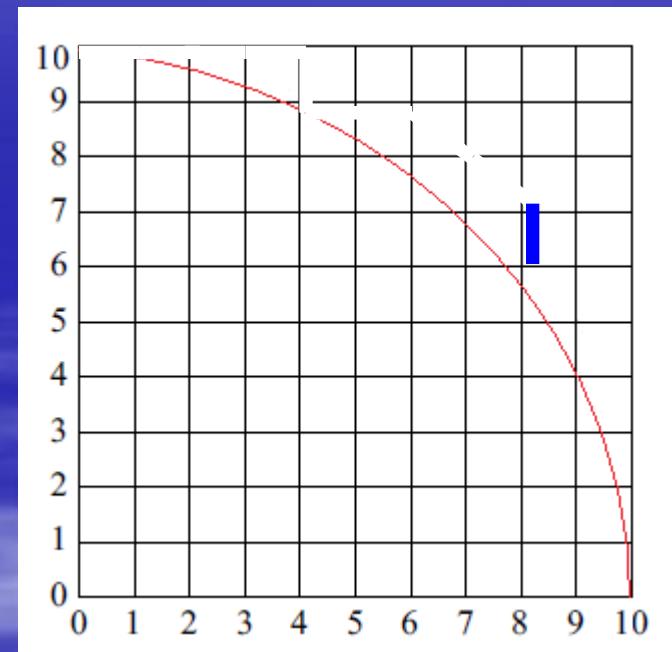
P <sub>1</sub>	Q <sub>1</sub>	ΔY	P <sub>2</sub>	Q <sub>2</sub>	ΔX
0	0		10	0	
1	0	X	10	0	○
2	1	X	10	0	○
3	3	X	10	0	○
4	6	X	10	0	○
4	0	○	9	9	X
5	4	X	9	8	○
6	9	X	9	7	○
7	5	○	8	5	○



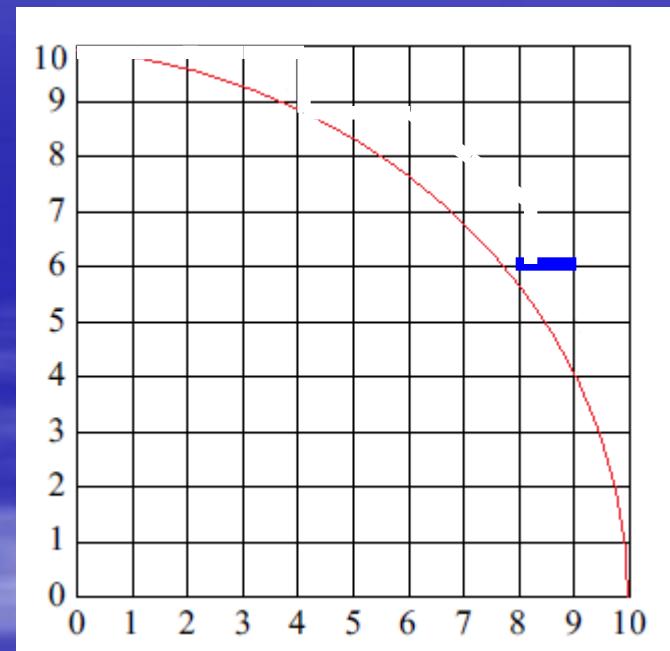
# Software DDA Interpolator



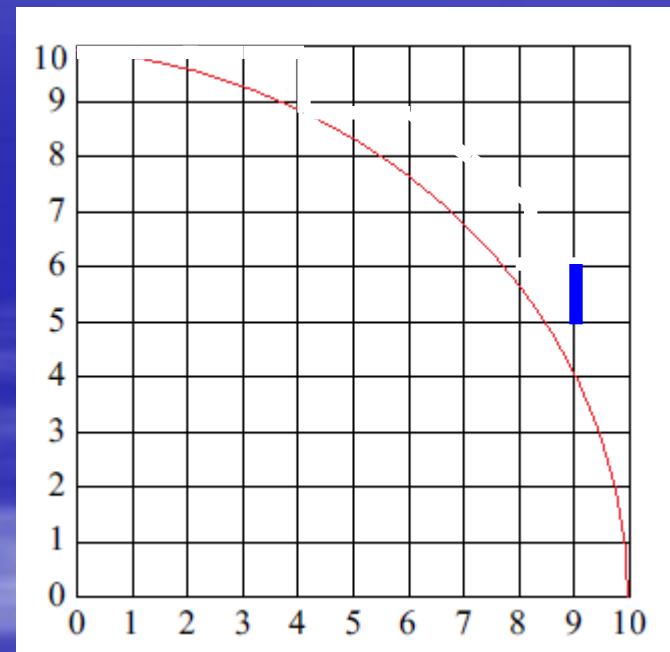
# Software DDA Interpolator



# Software DDA Interpolator

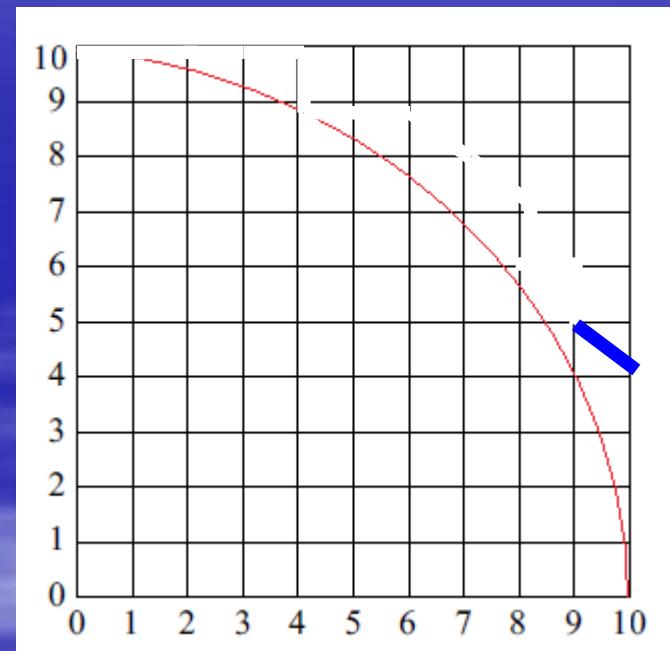


# Software DDA Interpolator



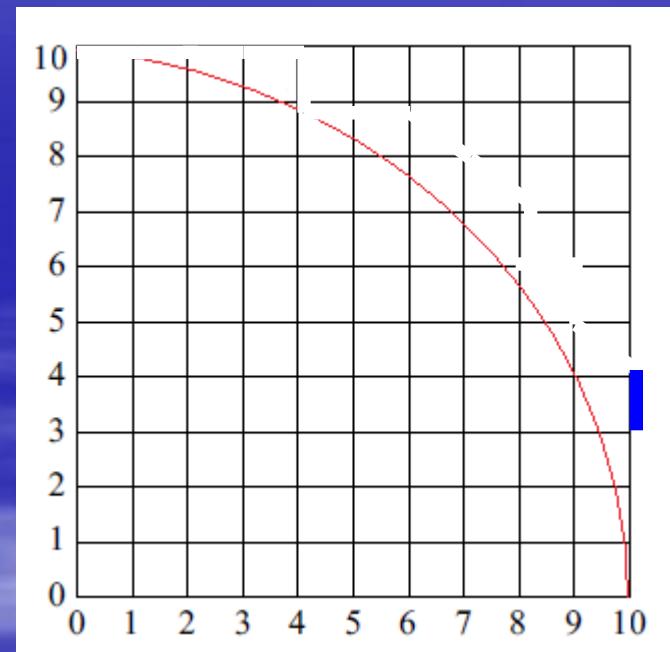
# Software DDA Interpolator

$P_1$	$Q_1$	$\Delta Y$	$P_2$	$Q_2$	$\Delta X$
8	2	○	7	2	○
8	0	○	6	8	X
9	8	X	6	4	○
9	7	○	5	9	X
10	6	○	4	3	○



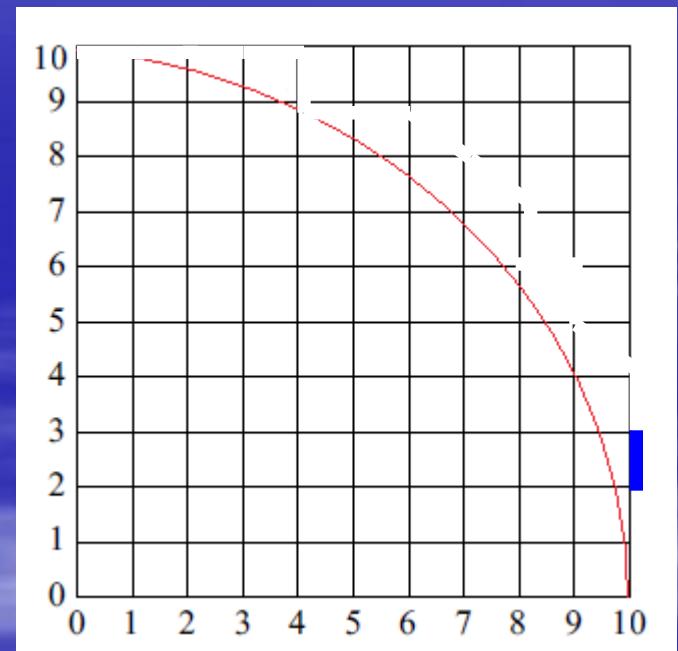
# Software DDA Interpolator

$P_1$	$Q_1$	$\Delta Y$	$P_2$	$Q_2$	$\Delta X$
8	2	○	7	2	○
8	0	○	6	8	X
9	8	X	6	4	○
9	7	○	5	9	X
10	6	○	4	3	○
10	6	○	3	6	X



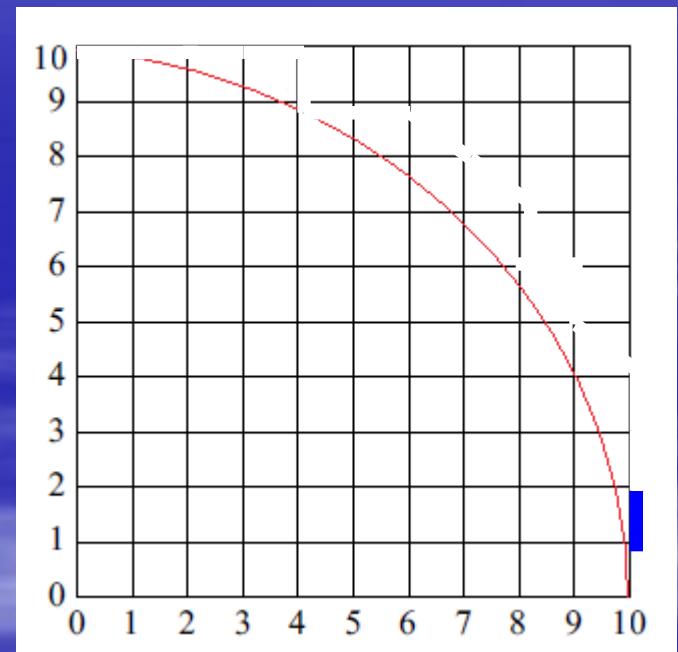
# Software DDA Interpolator

P <sub>1</sub>	Q <sub>1</sub>	ΔY	P <sub>2</sub>	Q <sub>2</sub>	ΔX
8	2	○	7	2	○
8	0	○	6	8	X
9	8	X	6	4	○
9	7	○	5	9	X
10	6	○	4	3	○
10	6	○	3	6	X
10	6	○	2	8	X



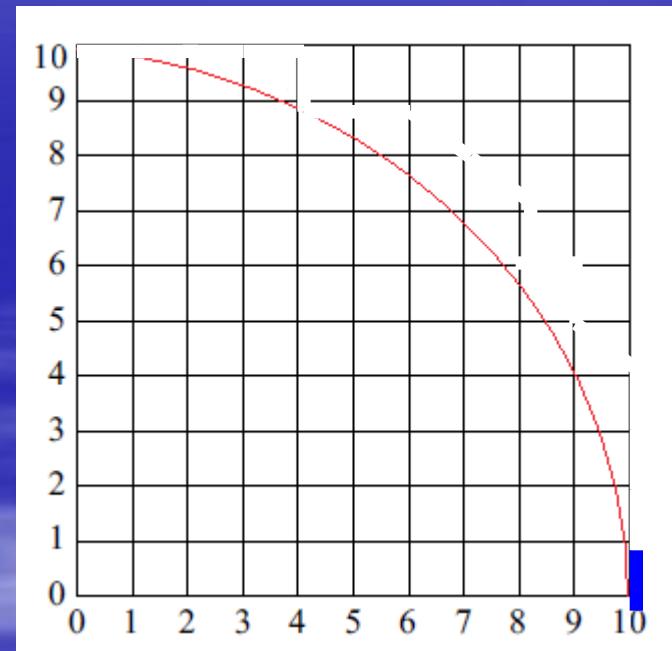
# Software DDA Interpolator

P <sub>1</sub>	Q <sub>1</sub>	ΔY	P <sub>2</sub>	Q <sub>2</sub>	ΔX
8	2	○	7	2	○
8	0	○	6	8	X
9	8	X	6	4	○
9	7	○	5	9	X
10	6	○	4	3	○
10	6	○	3	6	X
10	6	○	2	8	X
10	6	○	1	9	X



# Software DDA Interpolator

P <sub>1</sub>	Q <sub>1</sub>	ΔY	P <sub>2</sub>	Q <sub>2</sub>	ΔX
8	2	○	7	2	○
8	0	○	6	8	X
9	8	X	6	4	○
9	7	○	5	9	X
10	6	○	4	3	○
10	6	○	3	6	X
10	6	○	2	8	X
10	6	○	1	9	X
10	6	○	0	9	X

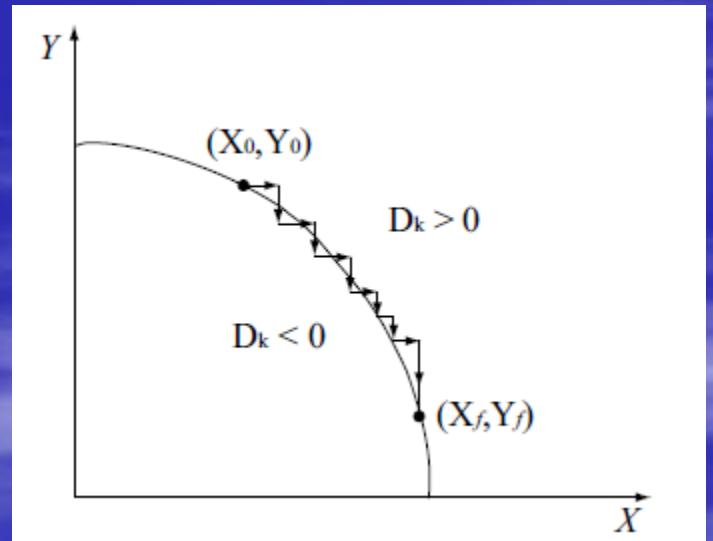


# Reference pulse method

- Stairs Approximation Interpolator (incremental interpolator)

$$D_k = X_k^2 + Y_k^2 - R^2$$

1.  $D_k < 0$  : position  $(X_k, Y_k)$  is located on the inside of a circle  $\rightarrow +X$
2.  $D_k > 0$  : position  $(X_k, Y_k)$  is located on the outside of a circle  $\rightarrow -Y$
3.  $D_k = 0$  : One of the above rules can be arbitrarily selected and applied



# Stairs Approximation Interpolator

- Eight stairs for arc path

No.	Quadrant	Direction	$D < 0$	$D > 0$
1	1	CW	+X	-Y
2	1	CCW	+Y	-X
3	2	CW	+Y	+X
4	2	CCW	-X	-Y
5	3	CW	-X	+Y
6	3	CCW	-Y	+X
7	4	CW	-Y	-X
8	4	CCW	+X	+Y

# Stairs Approximation Interpolator

- small amount of computation
- less memory space
- number of iteration steps

$$N = |X_0 - X_f| + |Y_0 - Y_f|$$

$(X_0, Y_0)$  : Start position

$(X_f, Y_f)$  : position command

# Stairs Approximation Interpolator

- Interpolating a quarter circle
  - R : radius
  - N : number of iterations (2R)
  - $V_l$  : velocity
  - $f_0$  : frequency

$$\frac{f_0}{V_l} = \frac{2R}{\pi R/2} = \frac{4}{\pi}$$

# Stairs Approximation Interpolator

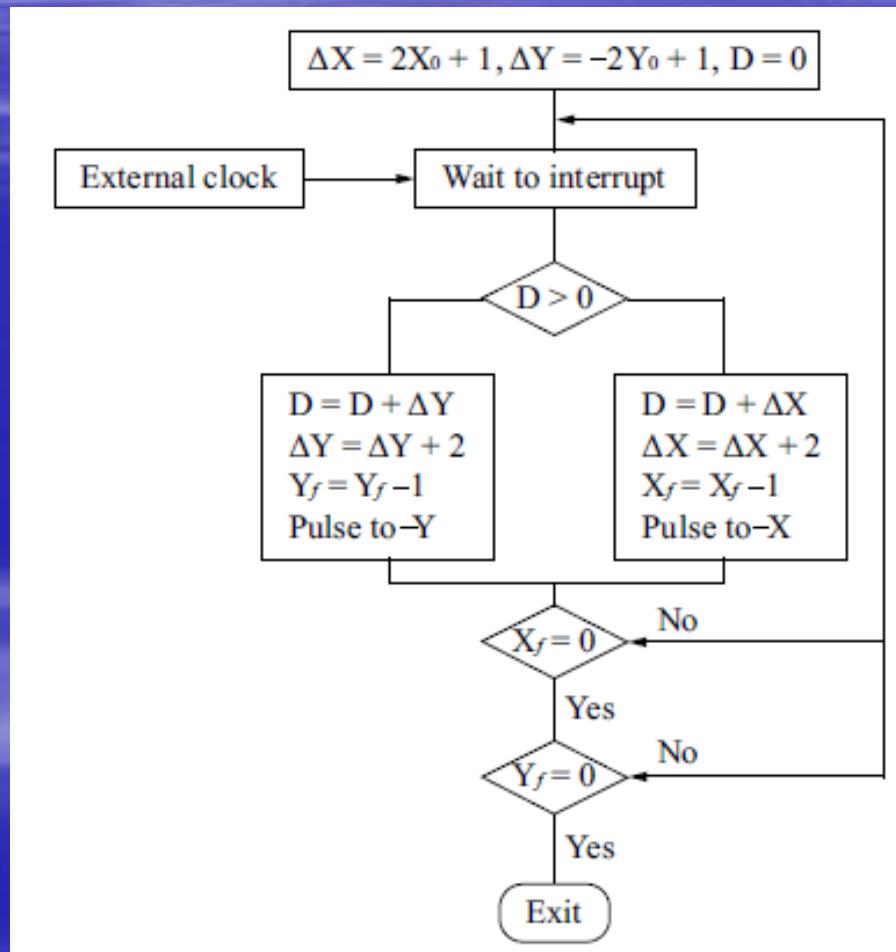
- Improved Stairs Approximation algorithm

$$D_{i,j} = X_i^2 + Y_j^2 - R^2$$

- When one step is added along the X-axis

$$\begin{aligned} D_{i+1,j} &= (X_i + 1)^2 + Y_j^2 - R^2 \\ &= D_{i,j} + 2X_i + 1 = D_{i,j} + \Delta X_i \\ \Delta X_{i+1} &= \Delta X_i + 2 \end{aligned}$$

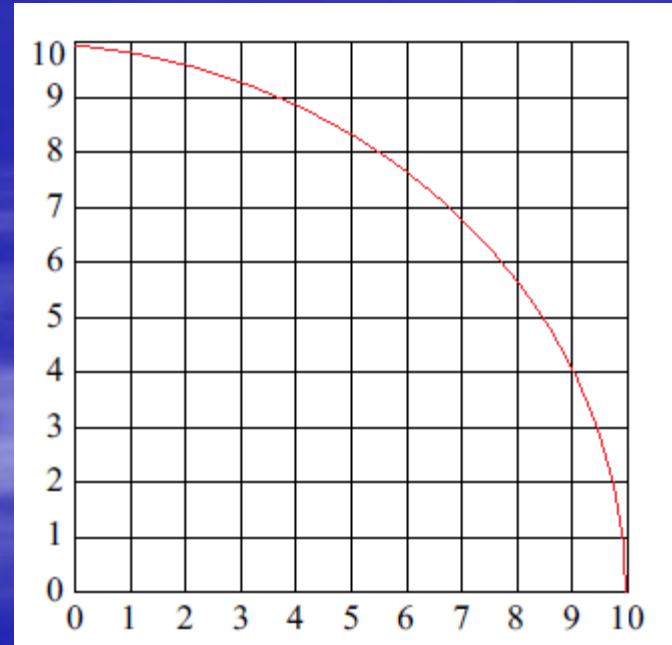
# Stairs Approximation Interpolator



# Stairs Approximation Interpolator

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

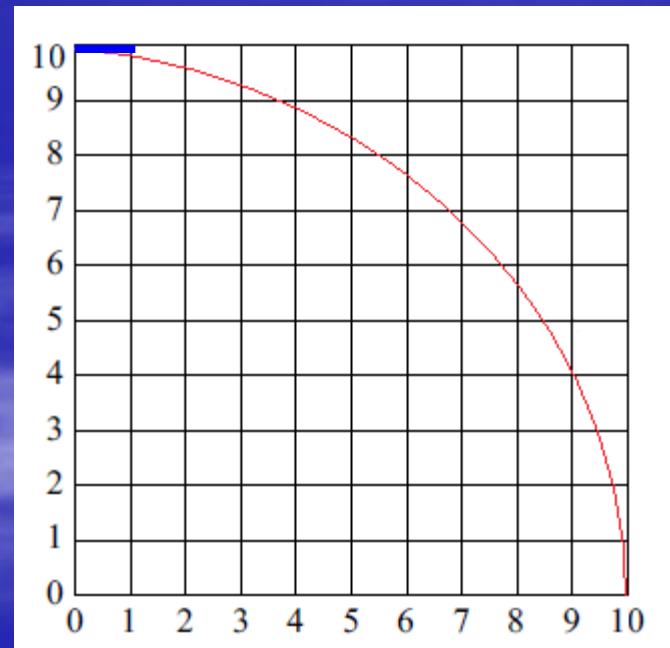
step	$D$	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	$X$	$Y$
0	0	1	-19	10	10	0	10
1	1	3	-19	9	10	1	10
2	-18	3	-17	9	9	1	9
3	-15	5	-17	8	9	2	9
4	-10	7	-17	7	9	3	9
5	-3	9	-17	6	9	4	9
6	6	1	-17	5	9	5	9
7	-11	11	-15	5	8	5	8
8	0	13	-15	4	8	6	8
9	13	15	-15	3	8	7	8
10	-2	15	-13	3	7	7	7
11	13	17	-13	2	7	8	7
12	0	17	-11	2	6	8	6
13	17	19	-11	1	6	9	6
14	6	19	-9	1	5	9	5
15	-3	19	-7	1	4	9	4
16	16	21	-7	0	4	10	4
17	9	21	-5	0	3	10	3
18	4	21	-3	0	2	10	2
19	1	21	-1	0	1	10	1
20	0	21	1	0	0	10	0



# Stairs Approximation Interpolator

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

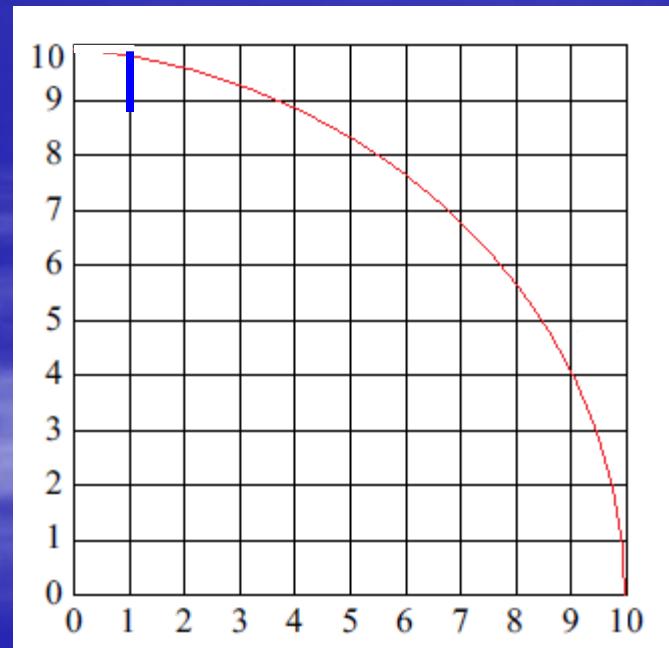
step	$D$	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	$X$	$Y$
0	0	1	-19	10	10	0	10
1	1	3	-19	9	10	1	10
2	-18	3	-17	9	9	1	9
3	-15	5	-17	8	9	2	9
4	-10	7	-17	7	9	3	9
5	-3	9	-17	6	9	4	9
6	6	1	-17	5	9	5	9
7	-11	11	-15	5	8	5	8
8	0	13	-15	4	8	6	8
9	13	15	-15	3	8	7	8
10	-2	15	-13	3	7	7	7
11	13	17	-13	2	7	8	7
12	0	17	-11	2	6	8	6
13	17	19	-11	1	6	9	6
14	6	19	-9	1	5	9	5
15	-3	19	-7	1	4	9	4
16	16	21	-7	0	4	10	4
17	9	21	-5	0	3	10	3
18	4	21	-3	0	2	10	2
19	1	21	-1	0	1	10	1
20	0	21	1	0	0	10	0



# Stairs Approximation Interpolator

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

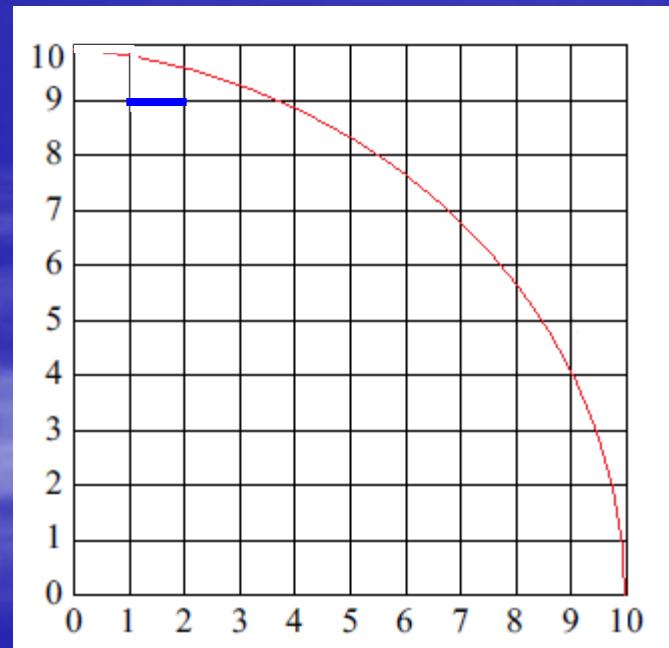
step	$D$	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	$X$	$Y$
0	0	1	-19	10	10	0	10
1	1	3	-19	9	10	1	10
2	-18	3	-17	9	9	1	9
3	-15	5	-17	8	9	2	9
4	-10	7	-17	7	9	3	9
5	-3	9	-17	6	9	4	9
6	6	1	-17	5	9	5	9
7	-11	11	-15	5	8	5	8
8	0	13	-15	4	8	6	8
9	13	15	-15	3	8	7	8
10	-2	15	-13	3	7	7	7
11	13	17	-13	2	7	8	7
12	0	17	-11	2	6	8	6
13	17	19	-11	1	6	9	6
14	6	19	-9	1	5	9	5
15	-3	19	-7	1	4	9	4
16	16	21	-7	0	4	10	4
17	9	21	-5	0	3	10	3
18	4	21	-3	0	2	10	2
19	1	21	-1	0	1	10	1
20	0	21	1	0	0	10	0



# Stairs Approximation Interpolator

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

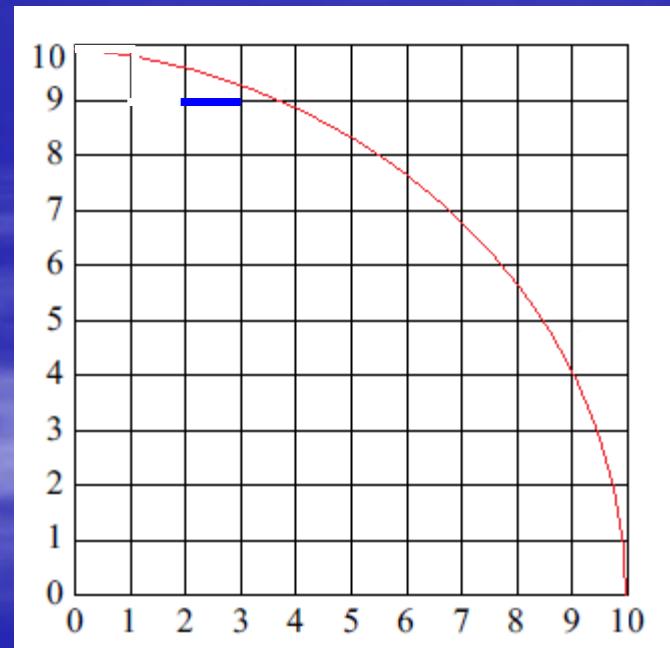
step	$D$	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	$X$	$Y$
0	0	1	-19	10	10	0	10
1	1	3	-19	9	10	1	10
2	-18	3	-17	9	9	1	9
3	-15	5	-17	8	9	2	9
4	-10	7	-17	7	9	3	9
5	-3	9	-17	6	9	4	9
6	6	1	-17	5	9	5	9
7	-11	11	-15	5	8	5	8
8	0	13	-15	4	8	6	8
9	13	15	-15	3	8	7	8
10	-2	15	-13	3	7	7	7
11	13	17	-13	2	7	8	7
12	0	17	-11	2	6	8	6
13	17	19	-11	1	6	9	6
14	6	19	-9	1	5	9	5
15	-3	19	-7	1	4	9	4
16	16	21	-7	0	4	10	4
17	9	21	-5	0	3	10	3
18	4	21	-3	0	2	10	2
19	1	21	-1	0	1	10	1
20	0	21	1	0	0	10	0



# Stairs Approximation Interpolator

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

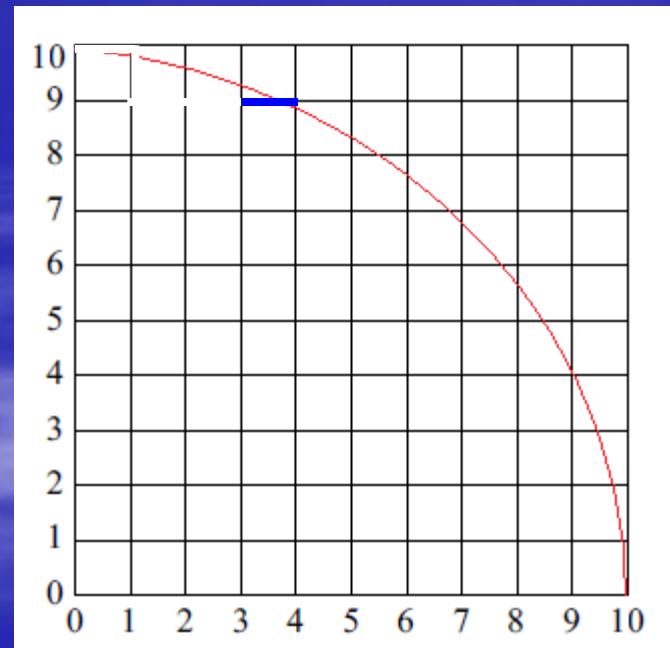
step	$D$	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	$X$	$Y$
0	0	1	-19	10	10	0	10
1	1	3	-19	9	10	1	10
2	-18	3	-17	9	9	1	9
3	-15	5	-17	8	9	2	9
4	-10	7	-17	7	9	3	9
5	-3	9	-17	6	9	4	9
6	6	1	-17	5	9	5	9
7	-11	11	-15	5	8	5	8
8	0	13	-15	4	8	6	8
9	13	15	-15	3	8	7	8
10	-2	15	-13	3	7	7	7
11	13	17	-13	2	7	8	7
12	0	17	-11	2	6	8	6
13	17	19	-11	1	6	9	6
14	6	19	-9	1	5	9	5
15	-3	19	-7	1	4	9	4
16	16	21	-7	0	4	10	4
17	9	21	-5	0	3	10	3
18	4	21	-3	0	2	10	2
19	1	21	-1	0	1	10	1
20	0	21	1	0	0	10	0



# Stairs Approximation Interpolator

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

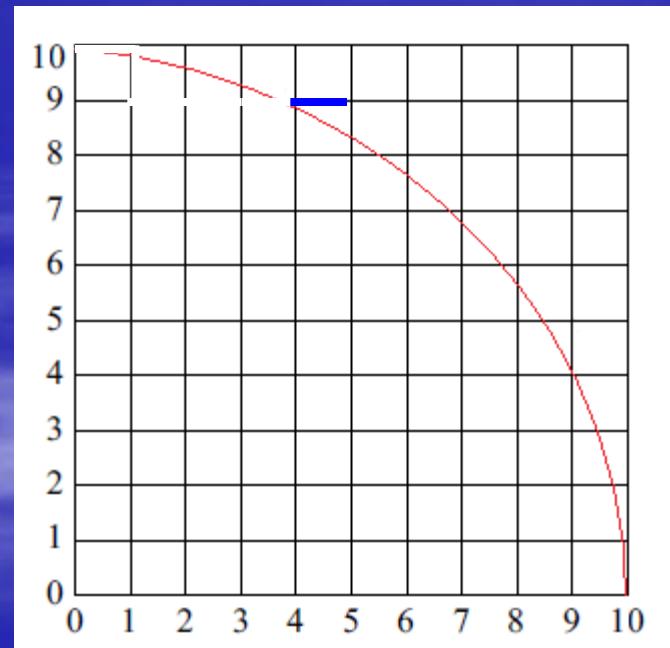
step	$D$	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	$X$	$Y$
0	0	1	-19	10	10	0	10
1	1	3	-19	9	10	1	10
2	-18	3	-17	9	9	1	9
3	-15	5	-17	8	9	2	9
4	-10	7	-17	7	9	3	9
5	-3	9	-17	6	9	4	9
6	6	1	-17	5	9	5	9
7	-11	11	-15	5	8	5	8
8	0	13	-15	4	8	6	8
9	13	15	-15	3	8	7	8
10	-2	15	-13	3	7	7	7
11	13	17	-13	2	7	8	7
12	0	17	-11	2	6	8	6
13	17	19	-11	1	6	9	6
14	6	19	-9	1	5	9	5
15	-3	19	-7	1	4	9	4
16	16	21	-7	0	4	10	4
17	9	21	-5	0	3	10	3
18	4	21	-3	0	2	10	2
19	1	21	-1	0	1	10	1
20	0	21	1	0	0	10	0



# Stairs Approximation Interpolator

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

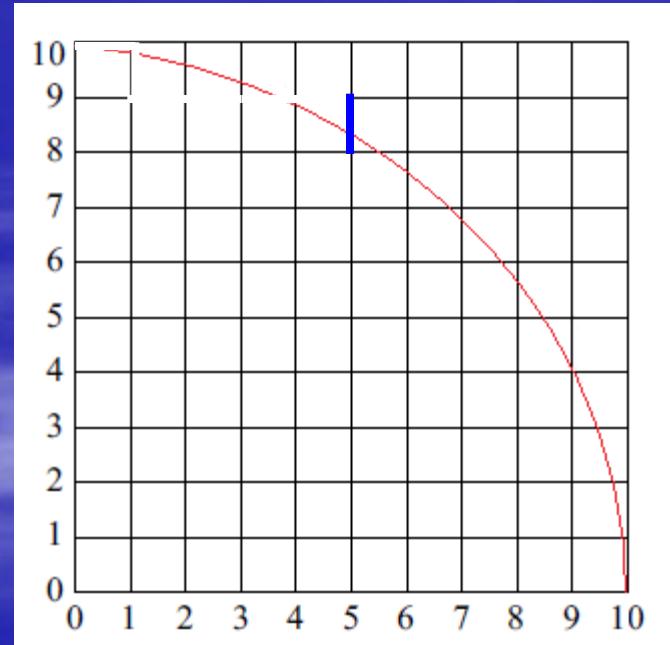
step	$D$	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	$X$	$Y$
0	0	1	-19	10	10	0	10
1	1	3	-19	9	10	1	10
2	-18	3	-17	9	9	1	9
3	-15	5	-17	8	9	2	9
4	-10	7	-17	7	9	3	9
5	-3	9	-17	6	9	4	9
6	6	1	-17	5	9	5	9
7	-11	11	-15	5	8	5	8
8	0	13	-15	4	8	6	8
9	13	15	-15	3	8	7	8
10	-2	15	-13	3	7	7	7
11	13	17	-13	2	7	8	7
12	0	17	-11	2	6	8	6
13	17	19	-11	1	6	9	6
14	6	19	-9	1	5	9	5
15	-3	19	-7	1	4	9	4
16	16	21	-7	0	4	10	4
17	9	21	-5	0	3	10	3
18	4	21	-3	0	2	10	2
19	1	21	-1	0	1	10	1
20	0	21	1	0	0	10	0



# Stairs Approximation Interpolator

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

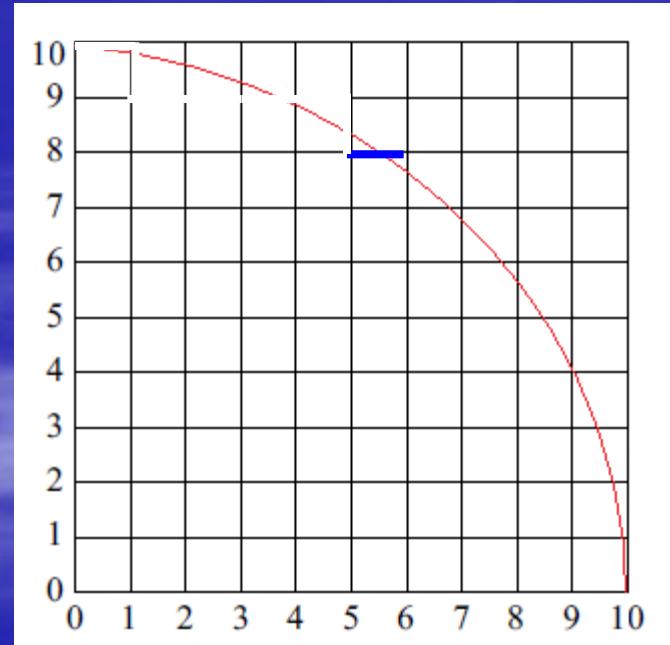
step	$D$	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	$X$	$Y$
0	0	1	-19	10	10	0	10
1	1	3	-19	9	10	1	10
2	-18	3	-17	9	9	1	9
3	-15	5	-17	8	9	2	9
4	-10	7	-17	7	9	3	9
5	-3	9	-17	6	9	4	9
6	6	1	-17	5	9	5	9
7	-11	11	-15	5	8	5	8
8	0	13	-15	4	8	6	8
9	13	15	-15	3	8	7	8
10	-2	15	-13	3	7	7	7
11	13	17	-13	2	7	8	7
12	0	17	-11	2	6	8	6
13	17	19	-11	1	6	9	6
14	6	19	-9	1	5	9	5
15	-3	19	-7	1	4	9	4
16	16	21	-7	0	4	10	4
17	9	21	-5	0	3	10	3
18	4	21	-3	0	2	10	2
19	1	21	-1	0	1	10	1
20	0	21	1	0	0	10	0



# Stairs Approximation Interpolator

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

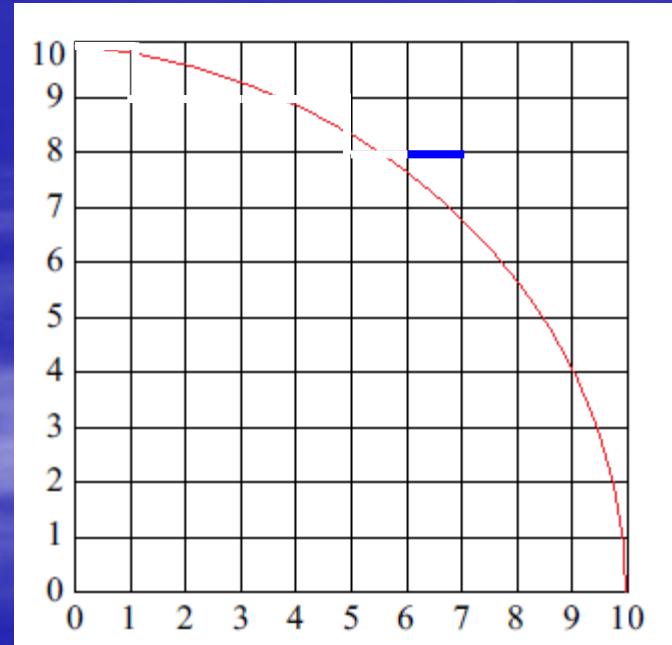
step	$D$	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	$X$	$Y$
0	0	1	-19	10	10	0	10
1	1	3	-19	9	10	1	10
2	-18	3	-17	9	9	1	9
3	-15	5	-17	8	9	2	9
4	-10	7	-17	7	9	3	9
5	-3	9	-17	6	9	4	9
6	6	1	-17	5	9	5	9
7	-11	11	-15	5	8	5	8
8	0	13	-15	4	8	6	8
9	13	15	-15	3	8	7	8
10	-2	15	-13	3	7	7	7
11	13	17	-13	2	7	8	7
12	0	17	-11	2	6	8	6
13	17	19	-11	1	6	9	6
14	6	19	-9	1	5	9	5
15	-3	19	-7	1	4	9	4
16	16	21	-7	0	4	10	4
17	9	21	-5	0	3	10	3
18	4	21	-3	0	2	10	2
19	1	21	-1	0	1	10	1
20	0	21	1	0	0	10	0



# Stairs Approximation Interpolator

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

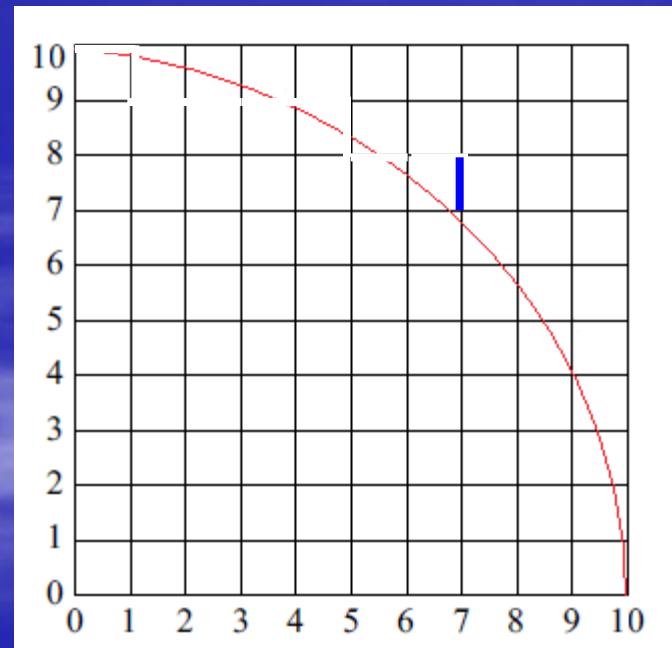
step	$D$	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	$X$	$Y$
0	0	1	-19	10	10	0	10
1	1	3	-19	9	10	1	10
2	-18	3	-17	9	9	1	9
3	-15	5	-17	8	9	2	9
4	-10	7	-17	7	9	3	9
5	-3	9	-17	6	9	4	9
6	6	1	-17	5	9	5	9
7	-11	11	-15	5	8	5	8
8	0	13	-15	4	8	6	8
9	13	15	-15	3	8	7	8
10	-2	15	-13	3	7	7	7
11	13	17	-13	2	7	8	7
12	0	17	-11	2	6	8	6
13	17	19	-11	1	6	9	6
14	6	19	-9	1	5	9	5
15	-3	19	-7	1	4	9	4
16	16	21	-7	0	4	10	4
17	9	21	-5	0	3	10	3
18	4	21	-3	0	2	10	2
19	1	21	-1	0	1	10	1
20	0	21	1	0	0	10	0



# Stairs Approximation Interpolator

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

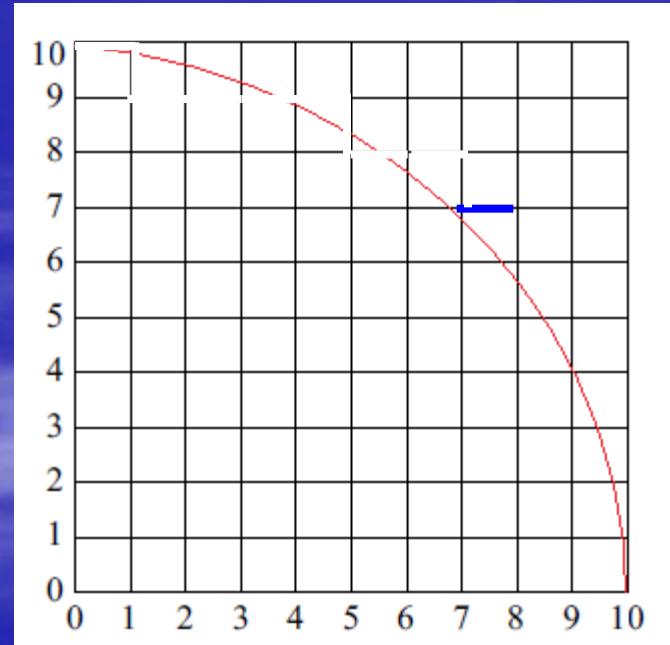
step	$D$	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	$X$	$Y$
0	0	1	-19	10	10	0	10
1	1	3	-19	9	10	1	10
2	-18	3	-17	9	9	1	9
3	-15	5	-17	8	9	2	9
4	-10	7	-17	7	9	3	9
5	-3	9	-17	6	9	4	9
6	6	1	-17	5	9	5	9
7	-11	11	-15	5	8	5	8
8	0	13	-15	4	8	6	8
9	13	15	-15	3	8	7	8
10	-2	15	-13	3	7	7	7
11	13	17	-13	2	7	8	7
12	0	17	-11	2	6	8	6
13	17	19	-11	1	6	9	6
14	6	19	-9	1	5	9	5
15	-3	19	-7	1	4	9	4
16	16	21	-7	0	4	10	4
17	9	21	-5	0	3	10	3
18	4	21	-3	0	2	10	2
19	1	21	-1	0	1	10	1
20	0	21	1	0	0	10	0



# Stairs Approximation Interpolator

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

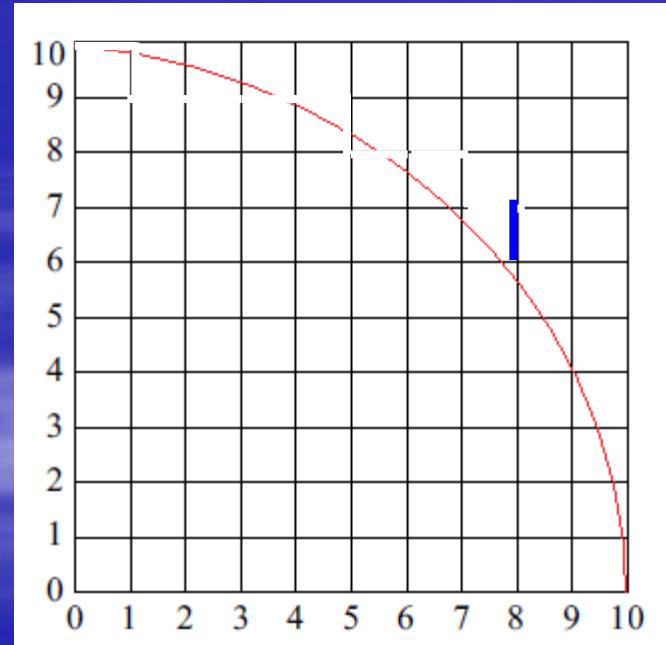
step	$D$	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	$X$	$Y$
0	0	1	-19	10	10	0	10
1	1	3	-19	9	10	1	10
2	-18	3	-17	9	9	1	9
3	-15	5	-17	8	9	2	9
4	-10	7	-17	7	9	3	9
5	-3	9	-17	6	9	4	9
6	6	1	-17	5	9	5	9
7	-11	11	-15	5	8	5	8
8	0	13	-15	4	8	6	8
9	13	15	-15	3	8	7	8
10	-2	15	-13	3	7	7	7
11	13	17	-13	2	7	8	7
12	0	17	-11	2	6	8	6
13	17	19	-11	1	6	9	6
14	6	19	-9	1	5	9	5
15	-3	19	-7	1	4	9	4
16	16	21	-7	0	4	10	4
17	9	21	-5	0	3	10	3
18	4	21	-3	0	2	10	2
19	1	21	-1	0	1	10	1
20	0	21	1	0	0	10	0



# Stairs Approximation Interpolator

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

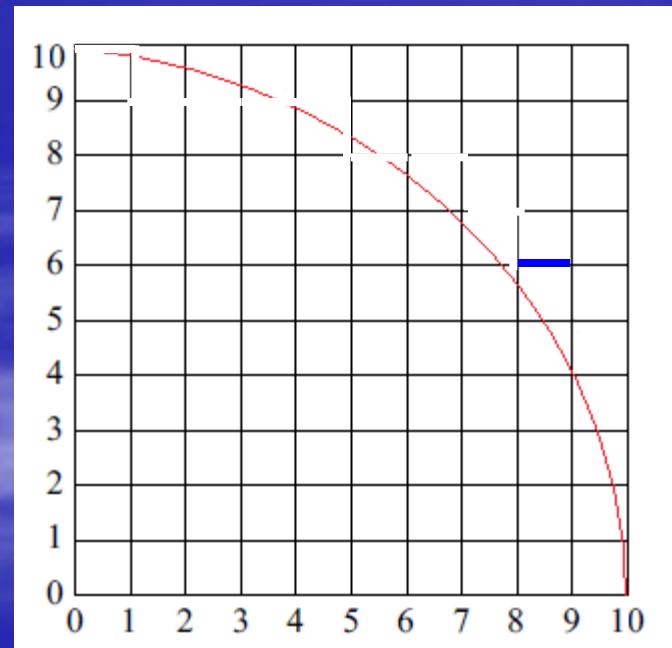
step	$D$	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	$X$	$Y$
0	0	1	-19	10	10	0	10
1	1	3	-19	9	10	1	10
2	-18	3	-17	9	9	1	9
3	-15	5	-17	8	9	2	9
4	-10	7	-17	7	9	3	9
5	-3	9	-17	6	9	4	9
6	6	1	-17	5	9	5	9
7	-11	11	-15	5	8	5	8
8	0	13	-15	4	8	6	8
9	13	15	-15	3	8	7	8
10	-2	15	-13	3	7	7	7
11	13	17	-13	2	7	8	7
12	0	17	-11	2	6	8	6
13	17	19	-11	1	6	9	6
14	6	19	-9	1	5	9	5
15	-3	19	-7	1	4	9	4
16	16	21	-7	0	4	10	4
17	9	21	-5	0	3	10	3
18	4	21	-3	0	2	10	2
19	1	21	-1	0	1	10	1
20	0	21	1	0	0	10	0



# Stairs Approximation Interpolator

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

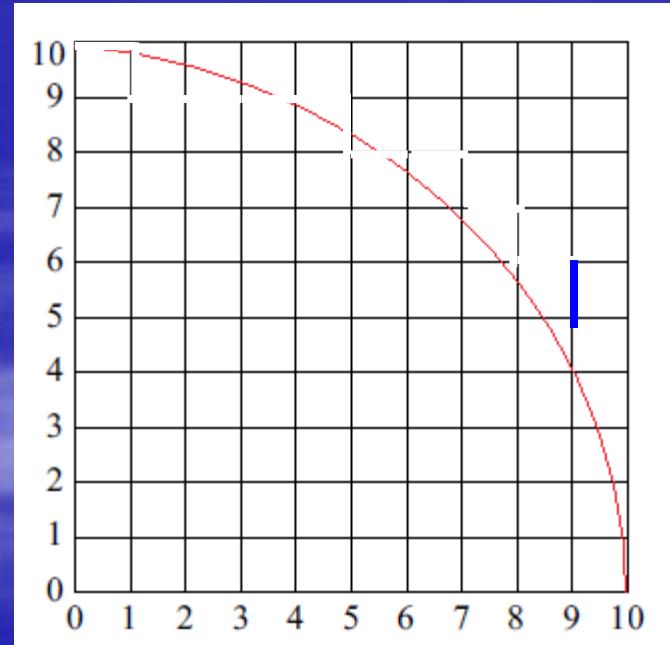
step	$D$	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	$X$	$Y$
0	0	1	-19	10	10	0	10
1	1	3	-19	9	10	1	10
2	-18	3	-17	9	9	1	9
3	-15	5	-17	8	9	2	9
4	-10	7	-17	7	9	3	9
5	-3	9	-17	6	9	4	9
6	6	1	-17	5	9	5	9
7	-11	11	-15	5	8	5	8
8	0	13	-15	4	8	6	8
9	13	15	-15	3	8	7	8
10	-2	15	-13	3	7	7	7
11	13	17	-13	2	7	8	7
12	0	17	-11	2	6	8	6
13	17	19	-11	1	6	9	6
14	6	19	-9	1	5	9	5
15	-3	19	-7	1	4	9	4
16	16	21	-7	0	4	10	4
17	9	21	-5	0	3	10	3
18	4	21	-3	0	2	10	2
19	1	21	-1	0	1	10	1
20	0	21	1	0	0	10	0



# Stairs Approximation Interpolator

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

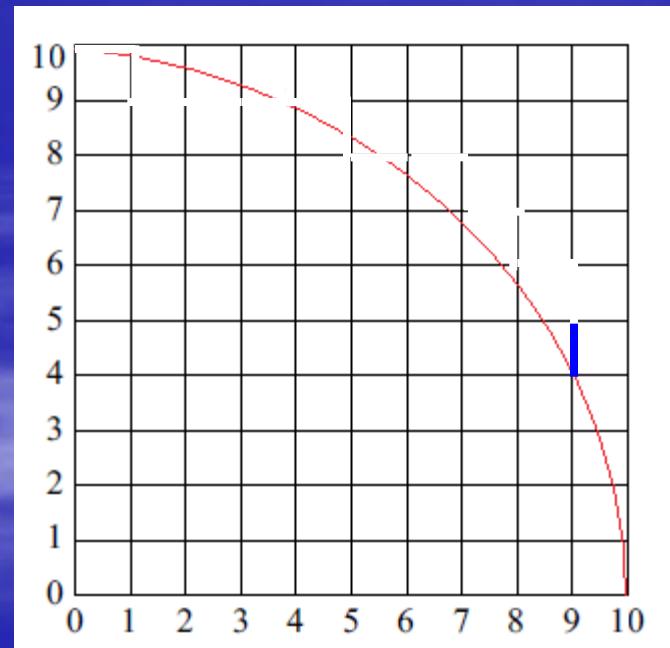
step	$D$	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	$X$	$Y$
0	0	1	-19	10	10	0	10
1	1	3	-19	9	10	1	10
2	-18	3	-17	9	9	1	9
3	-15	5	-17	8	9	2	9
4	-10	7	-17	7	9	3	9
5	-3	9	-17	6	9	4	9
6	6	1	-17	5	9	5	9
7	-11	11	-15	5	8	5	8
8	0	13	-15	4	8	6	8
9	13	15	-15	3	8	7	8
10	-2	15	-13	3	7	7	7
11	13	17	-13	2	7	8	7
12	0	17	-11	2	6	8	6
13	17	19	-11	1	6	9	6
14	6	19	-9	1	5	9	5
15	-3	19	-7	1	4	9	4
16	16	21	-7	0	4	10	4
17	9	21	-5	0	3	10	3
18	4	21	-3	0	2	10	2
19	1	21	-1	0	1	10	1
20	0	21	1	0	0	10	0



# Stairs Approximation Interpolator

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

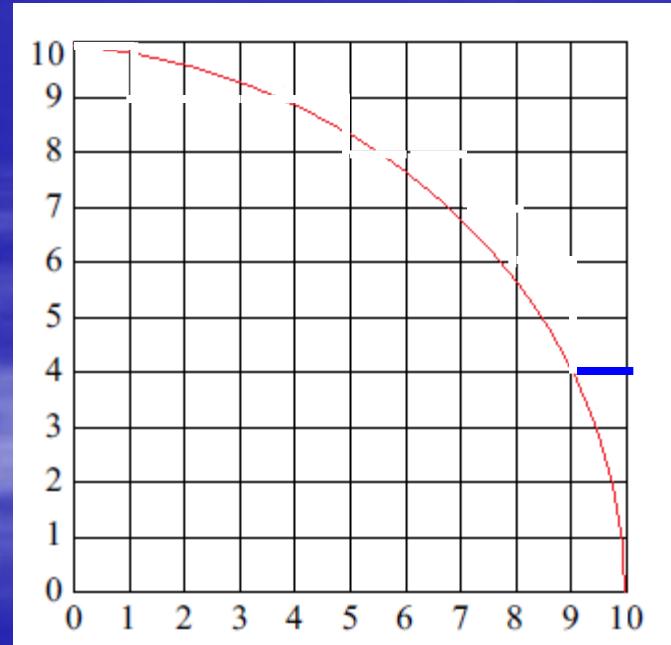
step	$D$	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	$X$	$Y$
0	0	1	-19	10	10	0	10
1	1	3	-19	9	10	1	10
2	-18	3	-17	9	9	1	9
3	-15	5	-17	8	9	2	9
4	-10	7	-17	7	9	3	9
5	-3	9	-17	6	9	4	9
6	6	1	-17	5	9	5	9
7	-11	11	-15	5	8	5	8
8	0	13	-15	4	8	6	8
9	13	15	-15	3	8	7	8
10	-2	15	-13	3	7	7	7
11	13	17	-13	2	7	8	7
12	0	17	-11	2	6	8	6
13	17	19	-11	1	6	9	6
14	6	19	-9	1	5	9	5
15	-3	19	-7	1	4	9	4
16	16	21	-7	0	4	10	4
17	9	21	-5	0	3	10	3
18	4	21	-3	0	2	10	2
19	1	21	-1	0	1	10	1
20	0	21	1	0	0	10	0



# Stairs Approximation Interpolator

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

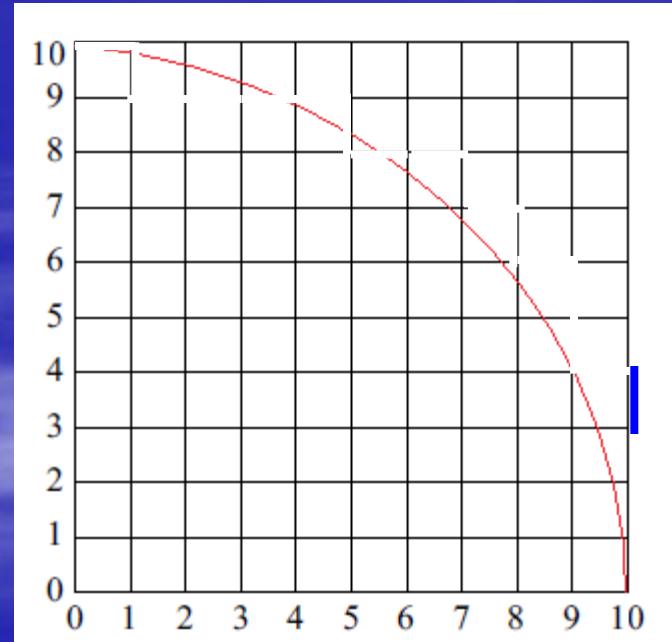
step	$D$	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	$X$	$Y$
0	0	1	-19	10	10	0	10
1	1	3	-19	9	10	1	10
2	-18	3	-17	9	9	1	9
3	-15	5	-17	8	9	2	9
4	-10	7	-17	7	9	3	9
5	-3	9	-17	6	9	4	9
6	6	1	-17	5	9	5	9
7	-11	11	-15	5	8	5	8
8	0	13	-15	4	8	6	8
9	13	15	-15	3	8	7	8
10	-2	15	-13	3	7	7	7
11	13	17	-13	2	7	8	7
12	0	17	-11	2	6	8	6
13	17	19	-11	1	6	9	6
14	6	19	-9	1	5	9	5
15	-3	19	-7	1	4	9	4
16	16	21	-7	0	4	10	4
17	9	21	-5	0	3	10	3
18	4	21	-3	0	2	10	2
19	1	21	-1	0	1	10	1
20	0	21	1	0	0	10	0



# Stairs Approximation Interpolator

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

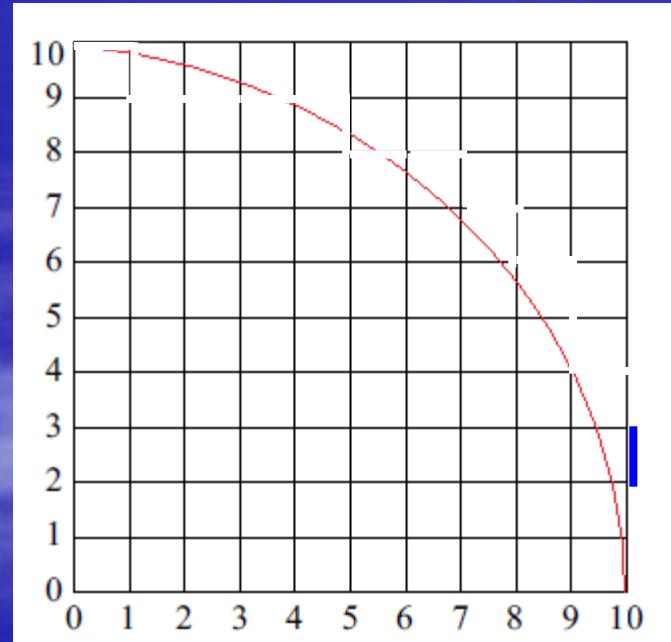
step	$D$	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	$X$	$Y$
0	0	1	-19	10	10	0	10
1	1	3	-19	9	10	1	10
2	-18	3	-17	9	9	1	9
3	-15	5	-17	8	9	2	9
4	-10	7	-17	7	9	3	9
5	-3	9	-17	6	9	4	9
6	6	1	-17	5	9	5	9
7	-11	11	-15	5	8	5	8
8	0	13	-15	4	8	6	8
9	13	15	-15	3	8	7	8
10	-2	15	-13	3	7	7	7
11	13	17	-13	2	7	8	7
12	0	17	-11	2	6	8	6
13	17	19	-11	1	6	9	6
14	6	19	-9	1	5	9	5
15	-3	19	-7	1	4	9	4
16	16	21	-7	0	4	10	4
17	9	21	-5	0	3	10	3
18	4	21	-3	0	2	10	2
19	1	21	-1	0	1	10	1
20	0	21	1	0	0	10	0



# Stairs Approximation Interpolator

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

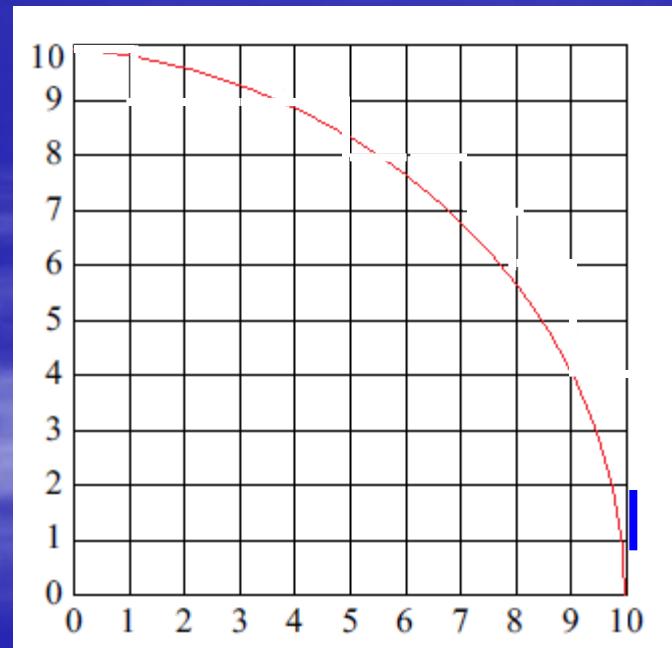
step	$D$	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	$X$	$Y$
0	0	1	-19	10	10	0	10
1	1	3	-19	9	10	1	10
2	-18	3	-17	9	9	1	9
3	-15	5	-17	8	9	2	9
4	-10	7	-17	7	9	3	9
5	-3	9	-17	6	9	4	9
6	6	1	-17	5	9	5	9
7	-11	11	-15	5	8	5	8
8	0	13	-15	4	8	6	8
9	13	15	-15	3	8	7	8
10	-2	15	-13	3	7	7	7
11	13	17	-13	2	7	8	7
12	0	17	-11	2	6	8	6
13	17	19	-11	1	6	9	6
14	6	19	-9	1	5	9	5
15	-3	19	-7	1	4	9	4
16	16	21	-7	0	4	10	4
17	9	21	-5	0	3	10	3
18	4	21	-3	0	2	10	2
19	1	21	-1	0	1	10	1
20	0	21	1	0	0	10	0



# Stairs Approximation Interpolator

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

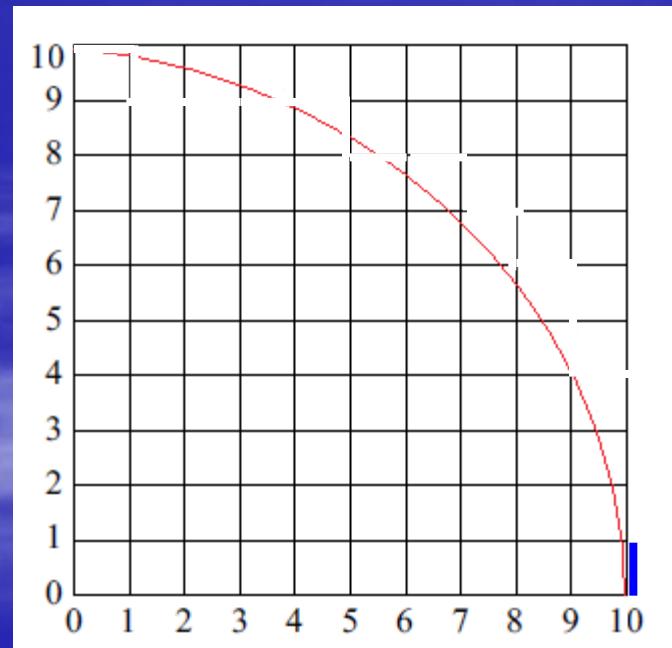
step	$D$	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	$X$	$Y$
0	0	1	-19	10	10	0	10
1	1	3	-19	9	10	1	10
2	-18	3	-17	9	9	1	9
3	-15	5	-17	8	9	2	9
4	-10	7	-17	7	9	3	9
5	-3	9	-17	6	9	4	9
6	6	1	-17	5	9	5	9
7	-11	11	-15	5	8	5	8
8	0	13	-15	4	8	6	8
9	13	15	-15	3	8	7	8
10	-2	15	-13	3	7	7	7
11	13	17	-13	2	7	8	7
12	0	17	-11	2	6	8	6
13	17	19	-11	1	6	9	6
14	6	19	-9	1	5	9	5
15	-3	19	-7	1	4	9	4
16	16	21	-7	0	4	10	4
17	9	21	-5	0	3	10	3
18	4	21	-3	0	2	10	2
19	1	21	-1	0	1	10	1
20	0	21	1	0	0	10	0



# Stairs Approximation Interpolator

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

step	$D$	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	$X$	$Y$
0	0	1	-19	10	10	0	10
1	1	3	-19	9	10	1	10
2	-18	3	-17	9	9	1	9
3	-15	5	-17	8	9	2	9
4	-10	7	-17	7	9	3	9
5	-3	9	-17	6	9	4	9
6	6	1	-17	5	9	5	9
7	-11	11	-15	5	8	5	8
8	0	13	-15	4	8	6	8
9	13	15	-15	3	8	7	8
10	-2	15	-13	3	7	7	7
11	13	17	-13	2	7	8	7
12	0	17	-11	2	6	8	6
13	17	19	-11	1	6	9	6
14	6	19	-9	1	5	9	5
15	-3	19	-7	1	4	9	4
16	16	21	-7	0	4	10	4
17	9	21	-5	0	3	10	3
18	4	21	-3	0	2	10	2
19	1	21	-1	0	1	10	1
20	0	21	1	0	0	10	0



# Reference pulse method

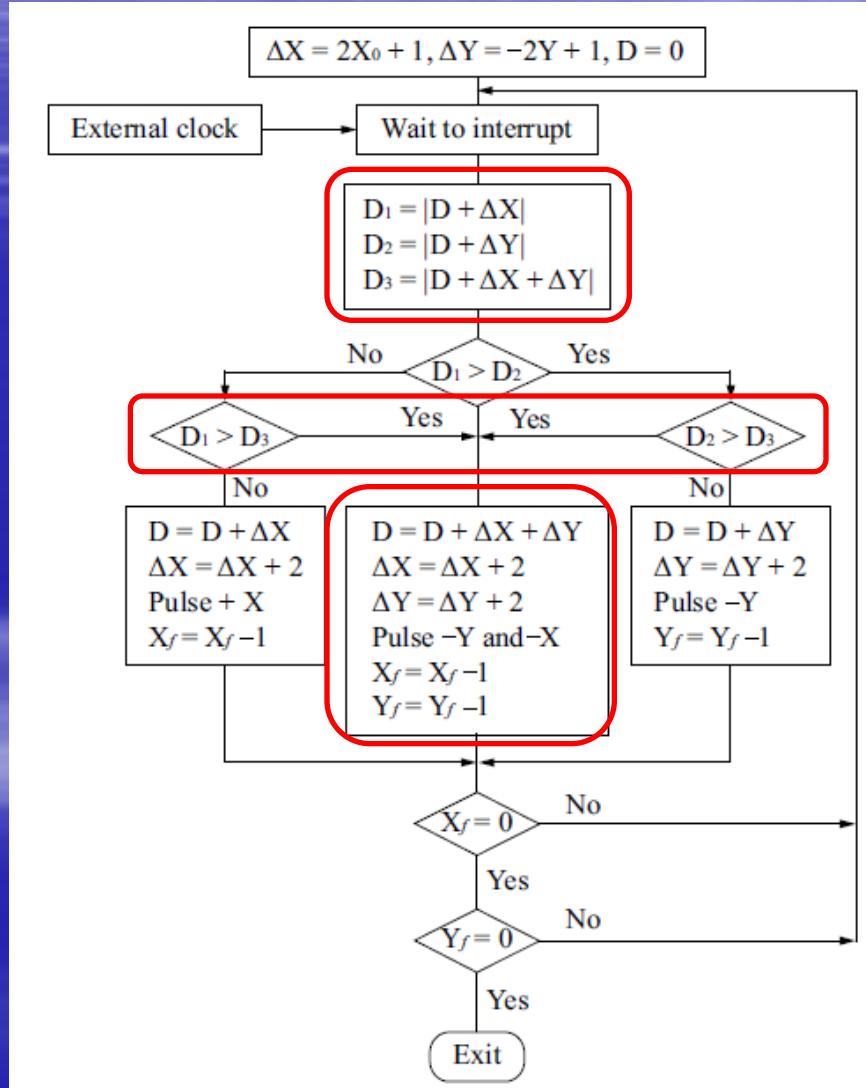
- Direct Search
  - simultaneous movement
  - optimal interpolation
    - searches through all possible directions
    - finds a direction with the minimum path error
- radial error  $E_{i,j}$

$$D_{i,j} \cong (2R)E_{i,j}$$

# Direct Search

- Finds a point having minimum error by estimating  $D_{i,j}$  at possible points.
- Three cases should be considered :
  1. Consider X-axes
  2. Consider Y-axes
  3. Consider X-axes & Y-axes

# Direct Search



# Direct Search

- Maximum error  $\frac{1}{2}$  BLU
- The number of iterations is 30% smaller than that of the Stairs Approximation algorithm and about 20% smaller than that of the DDA software algorithm.
- Frequency F

$$F = \frac{2\sqrt{2}}{\pi} V$$

# Direct Search

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

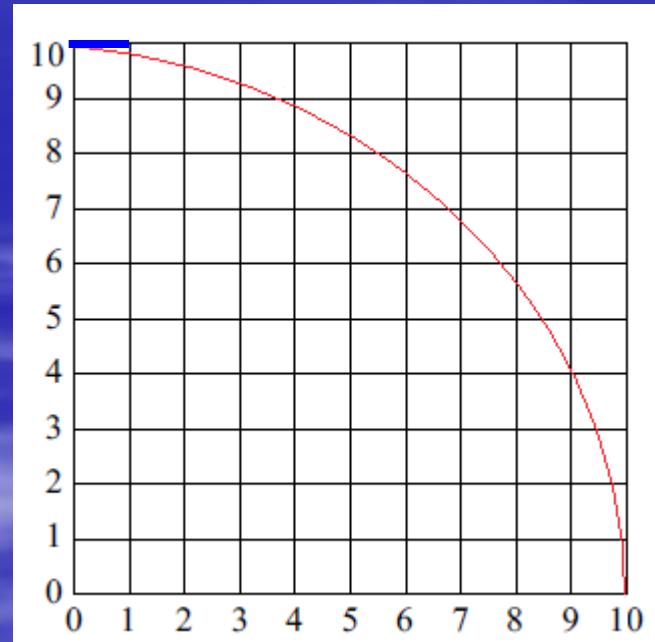
step	D	D1	D2	D3	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	X	Y
0	0				1	-19	10	10	0	10
1	1	1	-19	-18	3	-19	9	10	1	10
2	4	4	-18	-15	5	-19	8	10	2	10
3	9	9	-15	-10	7	-19	7	10	3	10
4	3	16	-10	-3	9	-17	6	9	4	9
5	6	6	-20	-11	11	-17	5	9	5	9
6	0	17	-11	0	13	-15	4	8	6	8
7	-2	13	-15	-2	15	-13	3	7	7	7
8	0	13	-15	0	17	-11	2	6	8	6
9	6	17	-11	6	19	-9	1	5	9	5
10	-3	25	-3	16	19	-7	1	4	9	4
11	9	16	-10	9	21	-5	0	3	10	3
12	4	30	4	25	21	-3	0	2	10	2
13	1	25	1	22	21	-1	0	1	10	1
14	0	22	0	21	21	1	0	0	10	0



# Direct Search

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

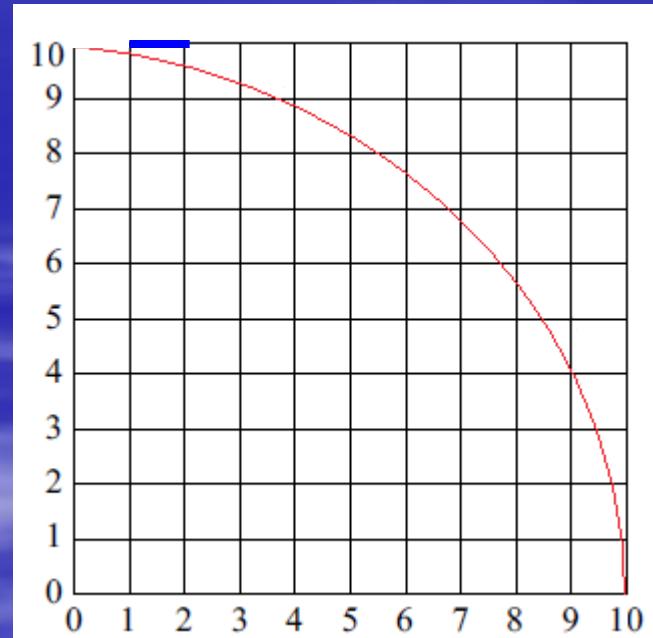
step	D	D1	D2	D3	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	X	Y
0	0				1	-19	10	10	0	10
1	1	1	-19	-18	3	-19	9	10	1	10
2	4	4	-18	-15	5	-19	8	10	2	10
3	9	9	-15	-10	7	-19	7	10	3	10
4	3	16	-10	-3	9	-17	6	9	4	9
5	6	6	-20	-11	11	-17	5	9	5	9
6	0	17	-11	0	13	-15	4	8	6	8
7	-2	13	-15	-2	15	-13	3	7	7	7
8	0	13	-15	0	17	-11	2	6	8	6
9	6	17	-11	6	19	-9	1	5	9	5
10	-3	25	-3	16	19	-7	1	4	9	4
11	9	16	-10	9	21	-5	0	3	10	3
12	4	30	4	25	21	-3	0	2	10	2
13	1	25	1	22	21	-1	0	1	10	1
14	0	22	0	21	21	1	0	0	10	0



# Direct Search

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

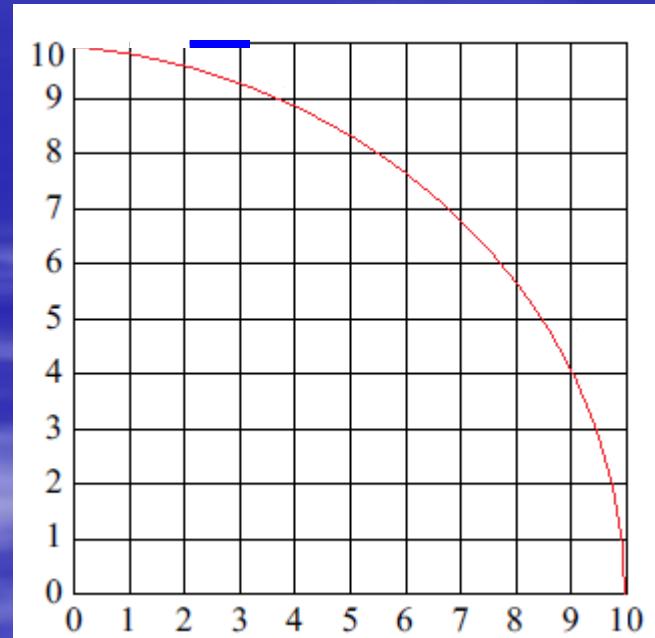
step	D	D1	D2	D3	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	X	Y
0	0				1	-19	10	10	0	10
1	1	1	-19	-18	3	-19	9	10	1	10
2	4	4	-18	-15	5	-19	8	10	2	10
3	9	9	-15	-10	7	-19	7	10	3	10
4	3	16	-10	-3	9	-17	6	9	4	9
5	6	6	-20	-11	11	-17	5	9	5	9
6	0	17	-11	0	13	-15	4	8	6	8
7	-2	13	-15	-2	15	-13	3	7	7	7
8	0	13	-15	0	17	-11	2	6	8	6
9	6	17	-11	6	19	-9	1	5	9	5
10	-3	25	-3	16	19	-7	1	4	9	4
11	9	16	-10	9	21	-5	0	3	10	3
12	4	30	4	25	21	-3	0	2	10	2
13	1	25	1	22	21	-1	0	1	10	1
14	0	22	0	21	21	1	0	0	10	0



# Direct Search

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

step	D	D1	D2	D3	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	X	Y
0	0				1	-19	10	10	0	10
1	1	1	-19	-18	3	-19	9	10	1	10
2	4	4	-18	-15	5	-19	8	10	2	10
3	9	9	-15	-10	7	-19	7	10	3	10
4	3	16	-10	-3	9	-17	6	9	4	9
5	6	6	-20	-11	11	-17	5	9	5	9
6	0	17	-11	0	13	-15	4	8	6	8
7	-2	13	-15	-2	15	-13	3	7	7	7
8	0	13	-15	0	17	-11	2	6	8	6
9	6	17	-11	6	19	-9	1	5	9	5
10	-3	25	-3	16	19	-7	1	4	9	4
11	9	16	-10	9	21	-5	0	3	10	3
12	4	30	4	25	21	-3	0	2	10	2
13	1	25	1	22	21	-1	0	1	10	1
14	0	22	0	21	21	1	0	0	10	0



# Direct Search

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

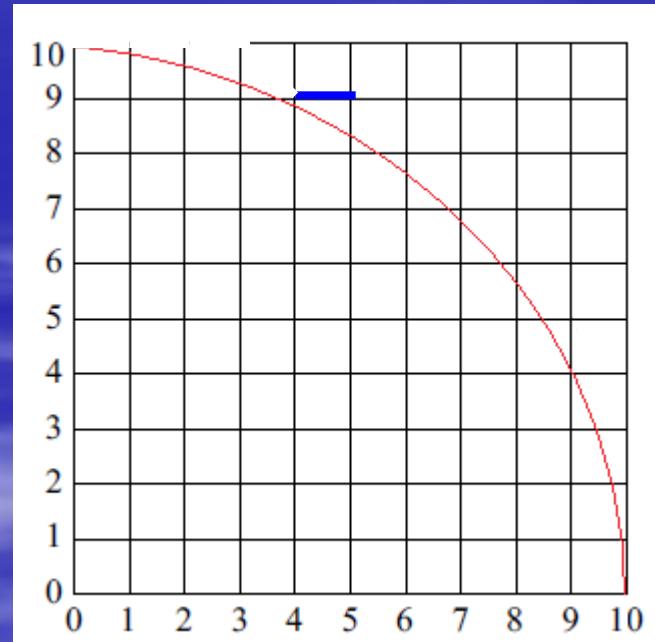
step	D	D1	D2	D3	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	X	Y
0	0				1	-19	10	10	0	10
1	1	1	-19	-18	3	-19	9	10	1	10
2	4	4	-18	-15	5	-19	8	10	2	10
3	9	9	-15	-10	7	-19	7	10	3	10
4	3	16	-10	-3	9	-17	6	9	4	9
5	6	6	-20	-11	11	-17	5	9	5	9
6	0	17	-11	0	13	-15	4	8	6	8
7	-2	13	-15	-2	15	-13	3	7	7	7
8	0	13	-15	0	17	-11	2	6	8	6
9	6	17	-11	6	19	-9	1	5	9	5
10	-3	25	-3	16	19	-7	1	4	9	4
11	9	16	-10	9	21	-5	0	3	10	3
12	4	30	4	25	21	-3	0	2	10	2
13	1	25	1	22	21	-1	0	1	10	1
14	0	22	0	21	21	1	0	0	10	0



# Direct Search

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

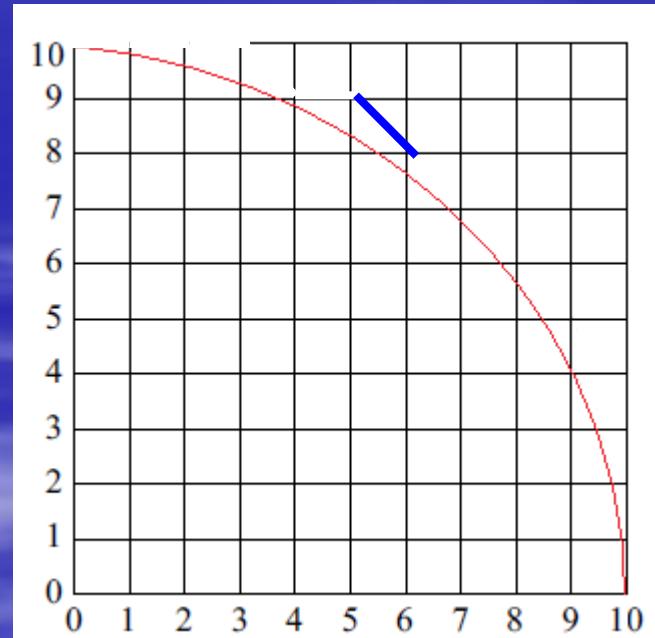
step	D	D1	D2	D3	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	X	Y
0	0				1	-19	10	10	0	10
1	1	1	-19	-18	3	-19	9	10	1	10
2	4	4	-18	-15	5	-19	8	10	2	10
3	9	9	-15	-10	7	-19	7	10	3	10
4	3	16	-10	-3	9	-17	6	9	4	9
5	6	6	-20	-11	11	-17	5	9	5	9
6	0	17	-11	0	13	-15	4	8	6	8
7	-2	13	-15	-2	15	-13	3	7	7	7
8	0	13	-15	0	17	-11	2	6	8	6
9	6	17	-11	6	19	-9	1	5	9	5
10	-3	25	-3	16	19	-7	1	4	9	4
11	9	16	-10	9	21	-5	0	3	10	3
12	4	30	4	25	21	-3	0	2	10	2
13	1	25	1	22	21	-1	0	1	10	1
14	0	22	0	21	21	1	0	0	10	0



# Direct Search

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

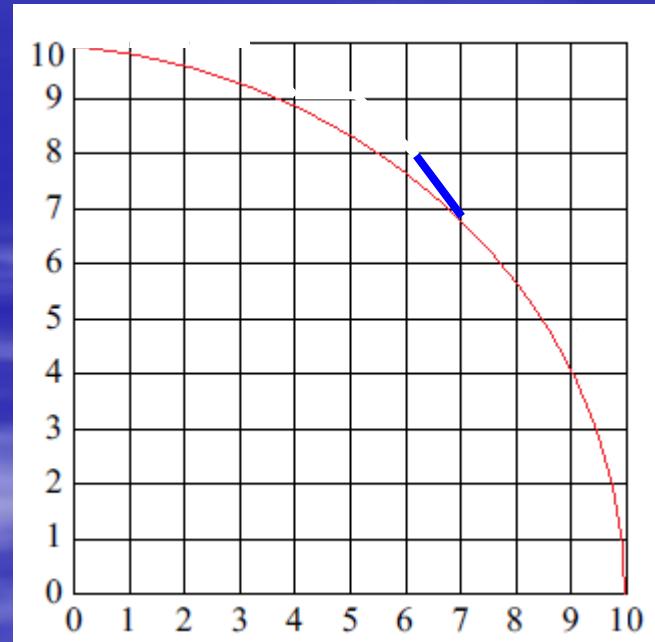
step	D	D1	D2	D3	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	X	Y
0	0				1	-19	10	10	0	10
1	1	1	-19	-18	3	-19	9	10	1	10
2	4	4	-18	-15	5	-19	8	10	2	10
3	9	9	-15	-10	7	-19	7	10	3	10
4	3	16	-10	-3	9	-17	6	9	4	9
5	6	6	-20	-11	11	-17	5	9	5	9
6	0	17	-11	0	13	-15	4	8	6	8
7	-2	13	-15	-2	15	-13	3	7	7	7
8	0	13	-15	0	17	-11	2	6	8	6
9	6	17	-11	6	19	-9	1	5	9	5
10	-3	25	-3	16	19	-7	1	4	9	4
11	9	16	-10	9	21	-5	0	3	10	3
12	4	30	4	25	21	-3	0	2	10	2
13	1	25	1	22	21	-1	0	1	10	1
14	0	22	0	21	21	1	0	0	10	0



# Direct Search

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

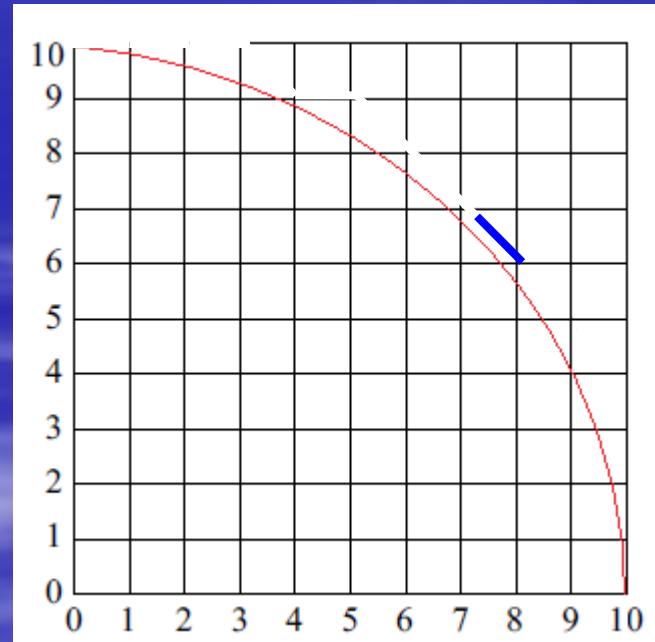
step	D	D1	D2	D3	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	X	Y
0	0				1	-19	10	10	0	10
1	1	1	-19	-18	3	-19	9	10	1	10
2	4	4	-18	-15	5	-19	8	10	2	10
3	9	9	-15	-10	7	-19	7	10	3	10
4	3	16	-10	-3	9	-17	6	9	4	9
5	6	6	-20	-11	11	-17	5	9	5	9
6	0	17	-11	0	13	-15	4	8	6	8
7	-2	13	-15	-2	15	-13	3	7	7	7
8	0	13	-15	0	17	-11	2	6	8	6
9	6	17	-11	6	19	-9	1	5	9	5
10	-3	25	-3	16	19	-7	1	4	9	4
11	9	16	-10	9	21	-5	0	3	10	3
12	4	30	4	25	21	-3	0	2	10	2
13	1	25	1	22	21	-1	0	1	10	1
14	0	22	0	21	21	1	0	0	10	0



# Direct Search

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

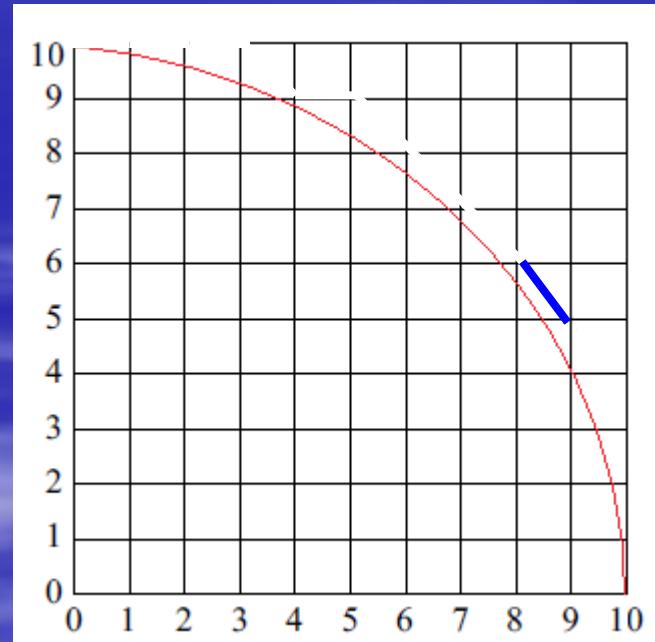
step	D	D1	D2	D3	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	X	Y
0	0				1	-19	10	10	0	10
1	1	1	-19	-18	3	-19	9	10	1	10
2	4	4	-18	-15	5	-19	8	10	2	10
3	9	9	-15	-10	7	-19	7	10	3	10
4	3	16	-10	-3	9	-17	6	9	4	9
5	6	6	-20	-11	11	-17	5	9	5	9
6	0	17	-11	0	13	-15	4	8	6	8
7	-2	13	-15	-2	15	-13	3	7	7	7
8	0	13	-15	0	17	-11	2	6	8	6
9	6	17	-11	6	19	-9	1	5	9	5
10	-3	25	-3	16	19	-7	1	4	9	4
11	9	16	-10	9	21	-5	0	3	10	3
12	4	30	4	25	21	-3	0	2	10	2
13	1	25	1	22	21	-1	0	1	10	1
14	0	22	0	21	21	1	0	0	10	0



# Direct Search

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

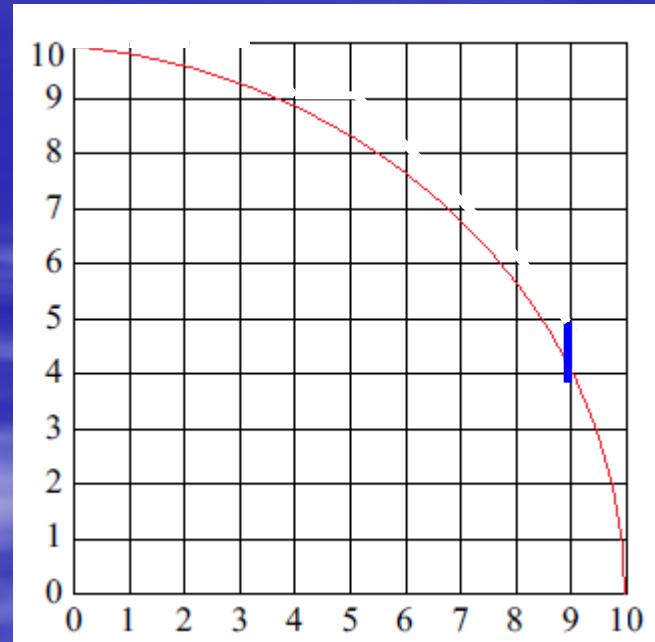
step	D	D1	D2	D3	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	X	Y
0	0				1	-19	10	10	0	10
1	1	1	-19	-18	3	-19	9	10	1	10
2	4	4	-18	-15	5	-19	8	10	2	10
3	9	9	-15	-10	7	-19	7	10	3	10
4	3	16	-10	-3	9	-17	6	9	4	9
5	6	6	-20	-11	11	-17	5	9	5	9
6	0	17	-11	0	13	-15	4	8	6	8
7	-2	13	-15	-2	15	-13	3	7	7	7
8	0	13	-15	0	17	-11	2	6	8	6
9	6	17	-11	6	19	-9	1	5	9	5
10	-3	25	-3	16	19	-7	1	4	9	4
11	9	16	-10	9	21	-5	0	3	10	3
12	4	30	4	25	21	-3	0	2	10	2
13	1	25	1	22	21	-1	0	1	10	1
14	0	22	0	21	21	1	0	0	10	0



# Direct Search

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

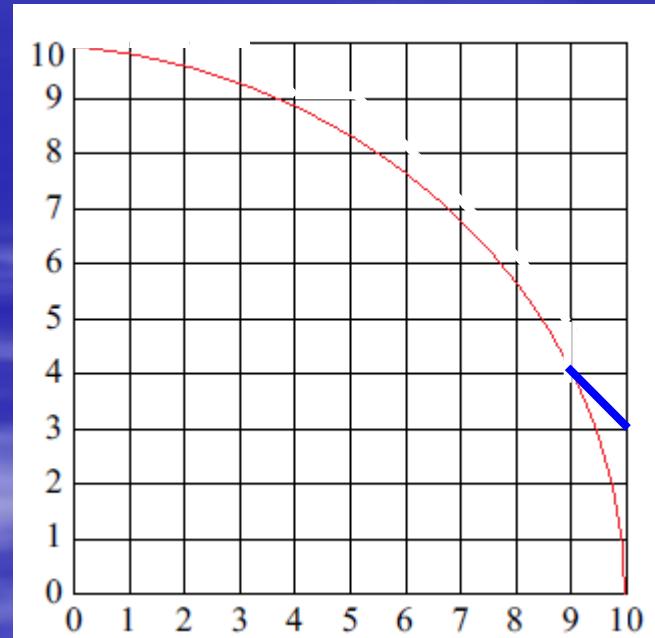
step	D	D1	D2	D3	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	X	Y
0	0				1	-19	10	10	0	10
1	1	1	-19	-18	3	-19	9	10	1	10
2	4	4	-18	-15	5	-19	8	10	2	10
3	9	9	-15	-10	7	-19	7	10	3	10
4	3	16	-10	-3	9	-17	6	9	4	9
5	6	6	-20	-11	11	-17	5	9	5	9
6	0	17	-11	0	13	-15	4	8	6	8
7	-2	13	-15	-2	15	-13	3	7	7	7
8	0	13	-15	0	17	-11	2	6	8	6
9	6	17	-11	6	19	-9	1	5	9	5
10	-3	25	-3	16	19	-7	1	4	9	4
11	9	16	-10	9	21	-5	0	3	10	3
12	4	30	4	25	21	-3	0	2	10	2
13	1	25	1	22	21	-1	0	1	10	1
14	0	22	0	21	21	1	0	0	10	0



# Direct Search

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

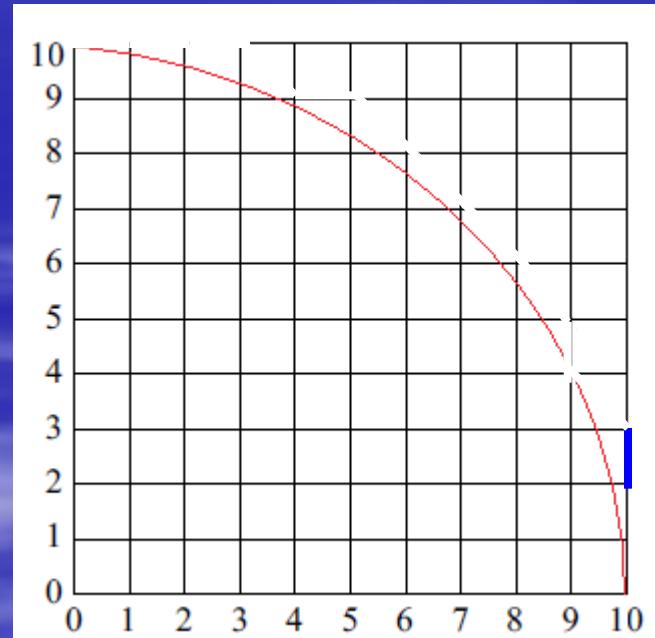
step	D	D1	D2	D3	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	X	Y
0	0				1	-19	10	10	0	10
1	1	1	-19	-18	3	-19	9	10	1	10
2	4	4	-18	-15	5	-19	8	10	2	10
3	9	9	-15	-10	7	-19	7	10	3	10
4	3	16	-10	-3	9	-17	6	9	4	9
5	6	6	-20	-11	11	-17	5	9	5	9
6	0	17	-11	0	13	-15	4	8	6	8
7	-2	13	-15	-2	15	-13	3	7	7	7
8	0	13	-15	0	17	-11	2	6	8	6
9	6	17	-11	6	19	-9	1	5	9	5
10	-3	25	-3	16	19	-7	1	4	9	4
11	9	16	-10	9	21	-5	0	3	10	3
12	4	30	4	25	21	-3	0	2	10	2
13	1	25	1	22	21	-1	0	1	10	1
14	0	22	0	21	21	1	0	0	10	0



# Direct Search

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

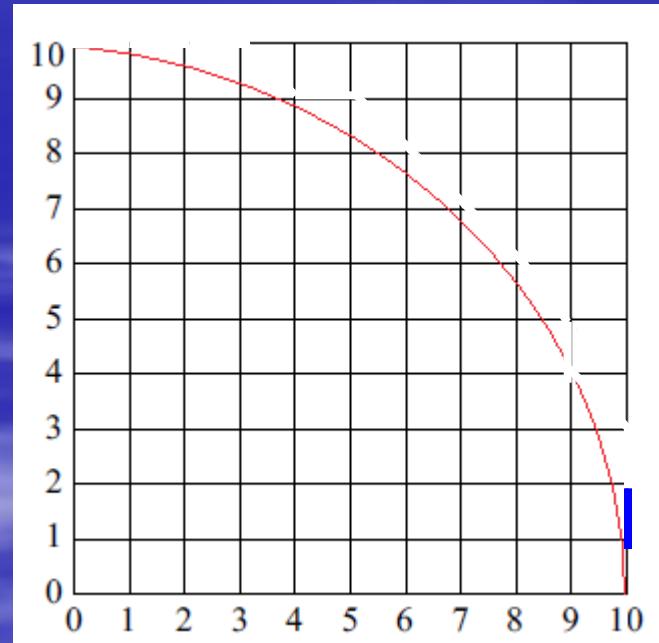
step	D	D1	D2	D3	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	X	Y
0	0				1	-19	10	10	0	10
1	1	1	-19	-18	3	-19	9	10	1	10
2	4	4	-18	-15	5	-19	8	10	2	10
3	9	9	-15	-10	7	-19	7	10	3	10
4	3	16	-10	-3	9	-17	6	9	4	9
5	6	6	-20	-11	11	-17	5	9	5	9
6	0	17	-11	0	13	-15	4	8	6	8
7	-2	13	-15	-2	15	-13	3	7	7	7
8	0	13	-15	0	17	-11	2	6	8	6
9	6	17	-11	6	19	-9	1	5	9	5
10	-3	25	-3	16	19	-7	1	4	9	4
11	9	16	-10	9	21	-5	0	3	10	3
12	4	30	4	25	21	-3	0	2	10	2
13	1	25	1	22	21	-1	0	1	10	1
14	0	22	0	21	21	1	0	0	10	0



# Direct Search

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

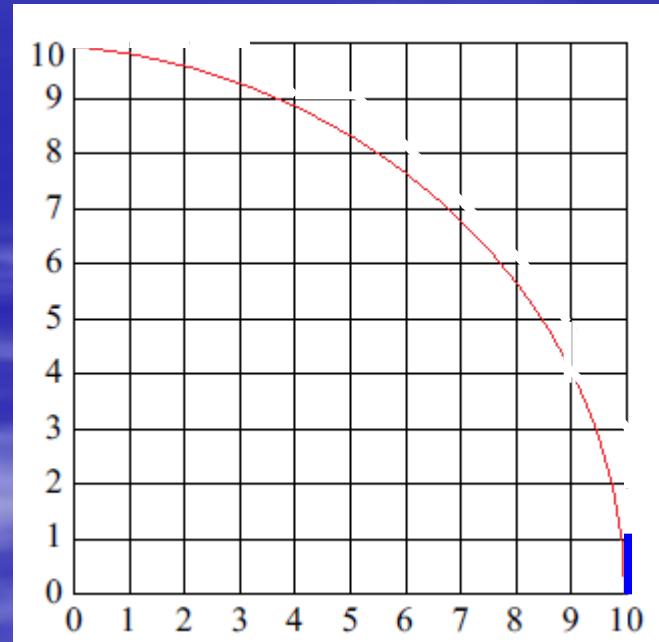
step	D	D1	D2	D3	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	X	Y
0	0				1	-19	10	10	0	10
1	1	1	-19	-18	3	-19	9	10	1	10
2	4	4	-18	-15	5	-19	8	10	2	10
3	9	9	-15	-10	7	-19	7	10	3	10
4	3	16	-10	-3	9	-17	6	9	4	9
5	6	6	-20	-11	11	-17	5	9	5	9
6	0	17	-11	0	13	-15	4	8	6	8
7	-2	13	-15	-2	15	-13	3	7	7	7
8	0	13	-15	0	17	-11	2	6	8	6
9	6	17	-11	6	19	-9	1	5	9	5
10	-3	25	-3	16	19	-7	1	4	9	4
11	9	16	-10	9	21	-5	0	3	10	3
12	4	30	4	25	21	-3	0	2	10	2
13	1	25	1	22	21	-1	0	1	10	1
14	0	22	0	21	21	1	0	0	10	0



# Direct Search

$$D_{i,j} = X_i^2 + Y_j^2 - 10^2, (X_0, Y_0) = (0,10), (X_f, Y_f) = (10,0)$$

step	D	D1	D2	D3	$\Delta x$	$\Delta y$	$\Delta X_f$	$\Delta Y_f$	X	Y
0	0				1	-19	10	10	0	10
1	1	1	-19	-18	3	-19	9	10	1	10
2	4	4	-18	-15	5	-19	8	10	2	10
3	9	9	-15	-10	7	-19	7	10	3	10
4	3	16	-10	-3	9	-17	6	9	4	9
5	6	6	-20	-11	11	-17	5	9	5	9
6	0	17	-11	0	13	-15	4	8	6	8
7	-2	13	-15	-2	15	-13	3	7	7	7
8	0	13	-15	0	17	-11	2	6	8	6
9	6	17	-11	6	19	-9	1	5	9	5
10	-3	25	-3	16	19	-7	1	4	9	4
11	9	16	-10	9	21	-5	0	3	10	3
12	4	30	4	25	21	-3	0	2	10	2
13	1	25	1	22	21	-1	0	1	10	1
14	0	22	0	21	21	1	0	0	10	0



# Reference Pulse Interpolator Algorithms

- Maximum allowable radius, consistency of feedrate, maximum allowable feedrate, and maximum error can be considered as performance indices for evaluating various reference-pulse interpolator algorithms

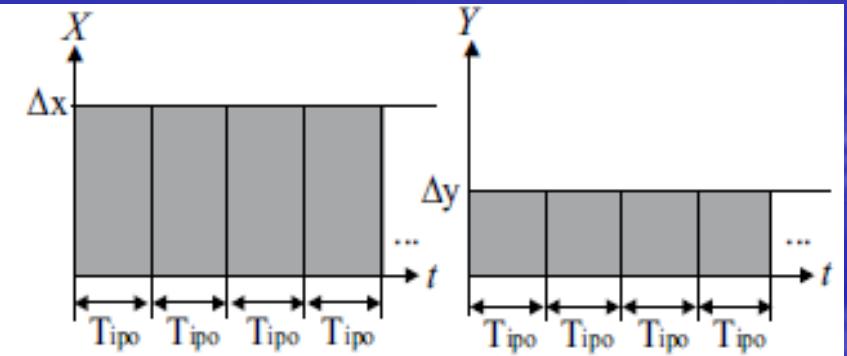
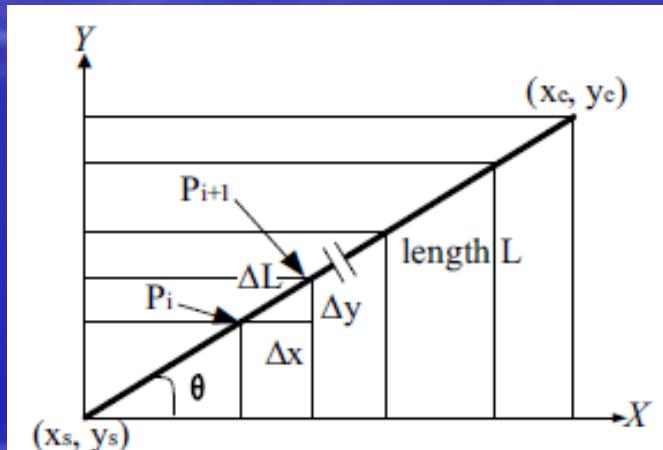
Method	$R_{max} + 1$	$E_{max}$	$N$	$V$	$V_{max}$	$V_{min}$
DDA	$2^{n-2}$	1	$(\pi/2)R$	$F$	$F$	$F$
Stairs	$2^{n-2}$	1	$2R$	$(\pi/4)F$	$F$	$(1/\sqrt{2})F$
DSM	$2^{n-2}$	$1/2$	$\sqrt{2}R$	$(\pi/2\sqrt{2})F$	$\sqrt{2}F$	$F$

# *Sampled-Data Interpolation*

- **Typical** for modern CNC
- Be repeated every **constant time** interval.
- Reference Word Interpolator for Lines
- Reference Word Interpolator for Circles
- Radial Error and Chord Height Error

# Reference Word Interpolator for Lines

- Fundamental idea : segmentation



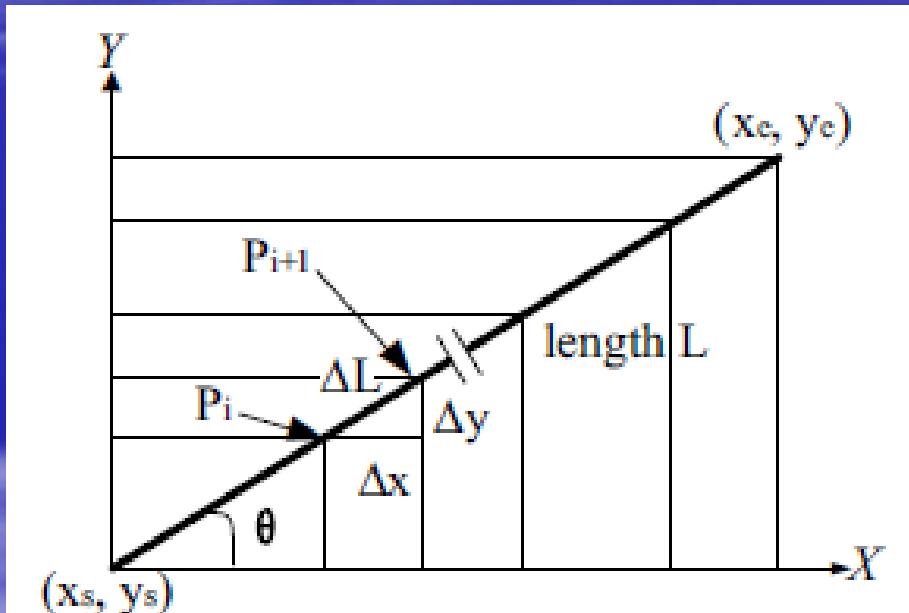
$$x_{i+1} = x_i + \Delta x$$

$$\Delta x = \Delta L \cdot \cos \theta = \Delta L \frac{x_e - x_s}{L}$$

$$y_{i+1} = y_i + \Delta y$$

$$\Delta y = \Delta L \cdot \sin \theta = \Delta L \frac{y_e - y_s}{L}$$

# Reference Word Interpolator for Lines

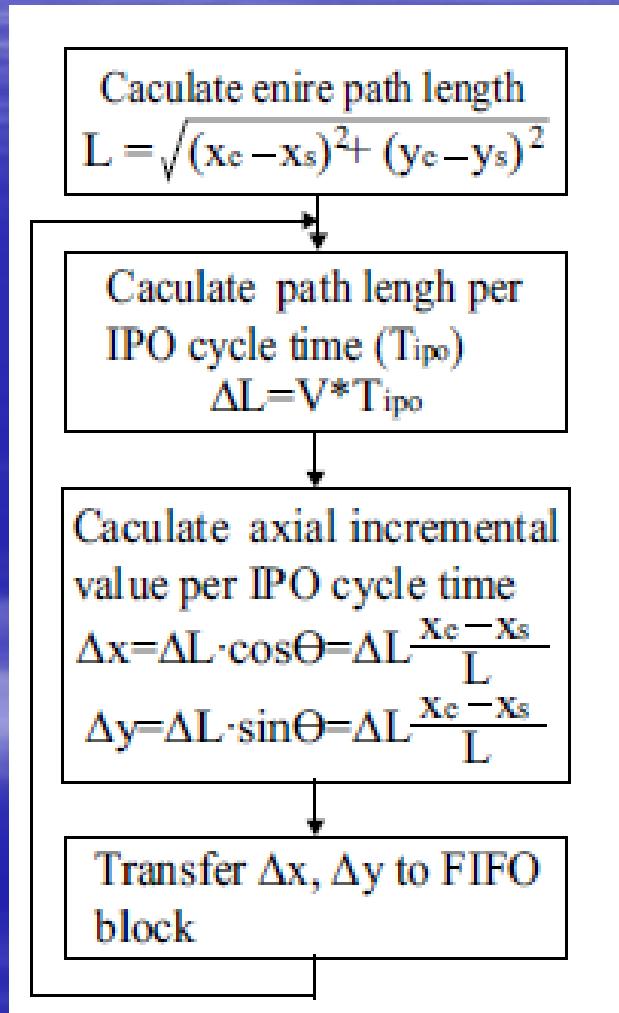


$$\Delta L = V \cdot T_{ipo}$$

$$L = \sqrt{(x_e - x_s)^2 + (y_e - y_s)^2}$$

$$V = V_0 \times \text{feed override}$$

# Reference Word Interpolator for Lines



# Reference Word Interpolator for Lines

- The total number of iterations for the interpolation

$$N = \text{int}\left(\frac{L}{\Delta L}\right)$$

The typical method for processing the residual length is to allocate the remainder evenly to every interpolation time.

## Reference Word Interpolator for Circles

- **Tangential velocity**,  $V$  should be held on the circular path.

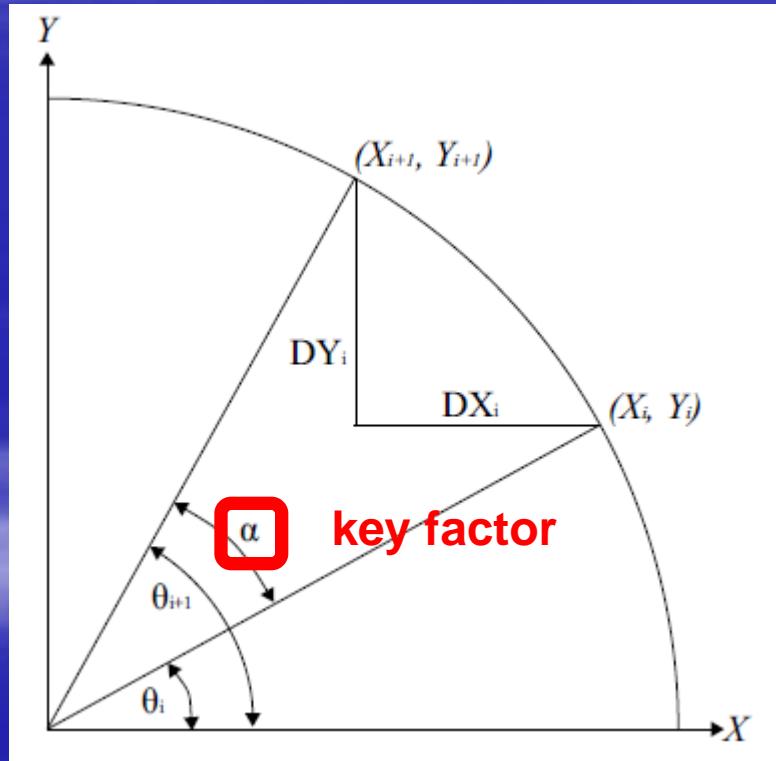
$$V_x(t) = V \sin \theta(t) \quad V_y(t) = V \cos \theta(t)$$

where       $\theta(t) = (Vt/R)$

- Circular path is approximated by **small line segments**.
- The **larger** the **number** of line segments, the **better** the **accuracy** of interpolation.

# Reference Word Interpolator for Circles

- Two successive interpolated points



# Reference Word Interpolator for Circles

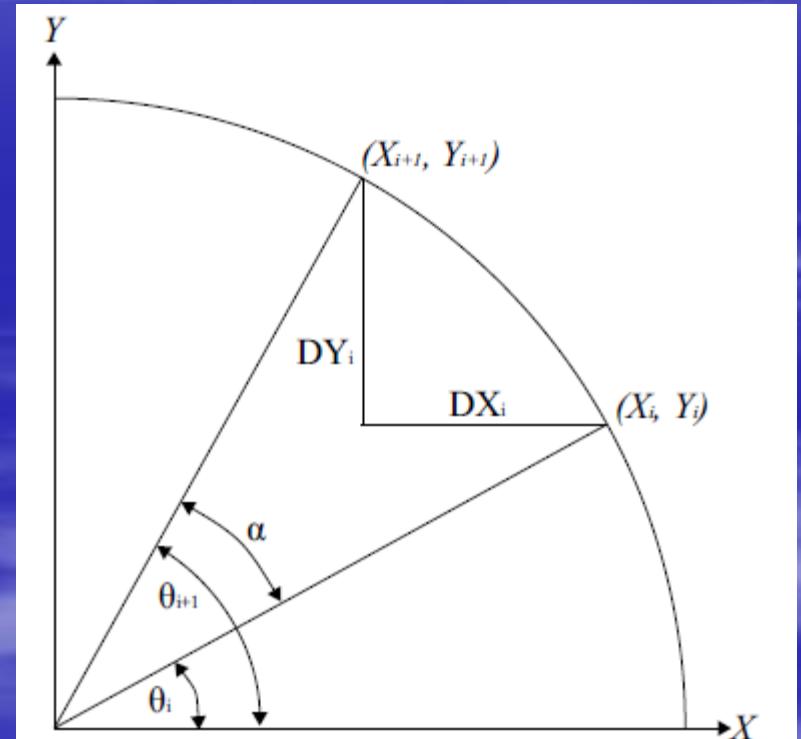
$$\begin{aligned}\cos \theta(i+1) &= A \cos \theta(i) - B \sin \theta(i) \\ \sin \theta(i+1) &= A \sin \theta(i) + B \cos \theta(i)\end{aligned}$$

$$A = \cos \alpha \quad B = \sin \alpha$$

$$\theta(i+1) = \theta(i) + \alpha$$

$$\sin(\alpha + \beta) = \sin \alpha \cos \beta + \cos \alpha \sin \beta$$

$$\cos(\alpha + \beta) = \cos \alpha \cos \beta - \sin \alpha \sin \beta$$



# Reference Word Interpolator for Circles

$$X(i) = R(i)\cos \theta(i)$$

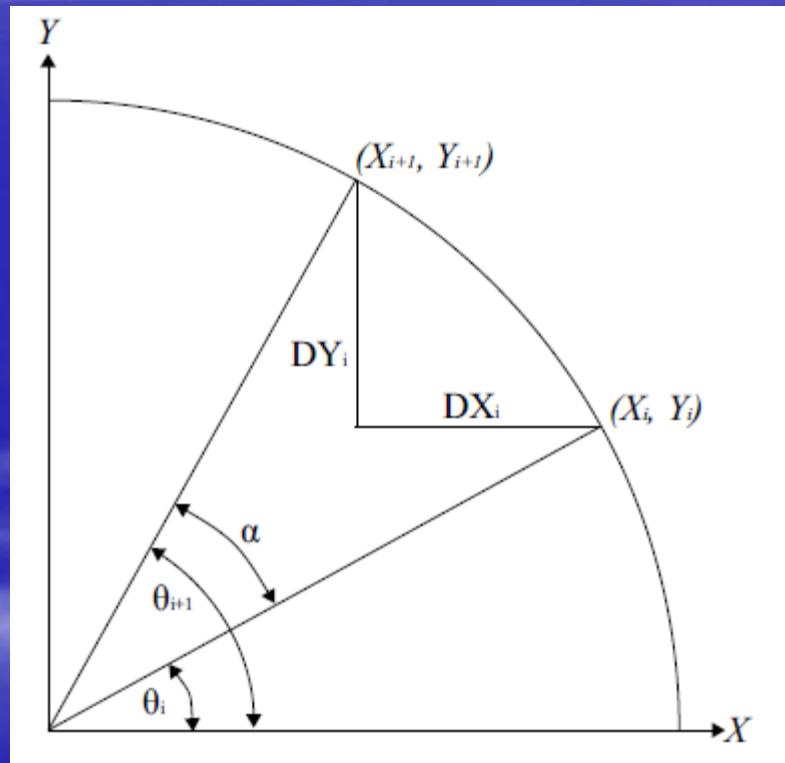
$$X(i+1) = R(i)\cos \theta(i+1)$$

$$X(i+1) = AX(i) - BY(i)$$

$$Y(i) = R(i)\sin \theta(i)$$

$$Y(i+1) = R(i)\sin \theta(i+1)$$

$$Y(i+1) = AY(i) + BX(i)$$



$$\cos \theta(i+1) = A \cos \theta(i) - B \sin \theta(i)$$

$$\sin \theta(i+1) = A \sin \theta(i) + B \cos \theta(i)$$

# Reference Word Interpolator for Circles

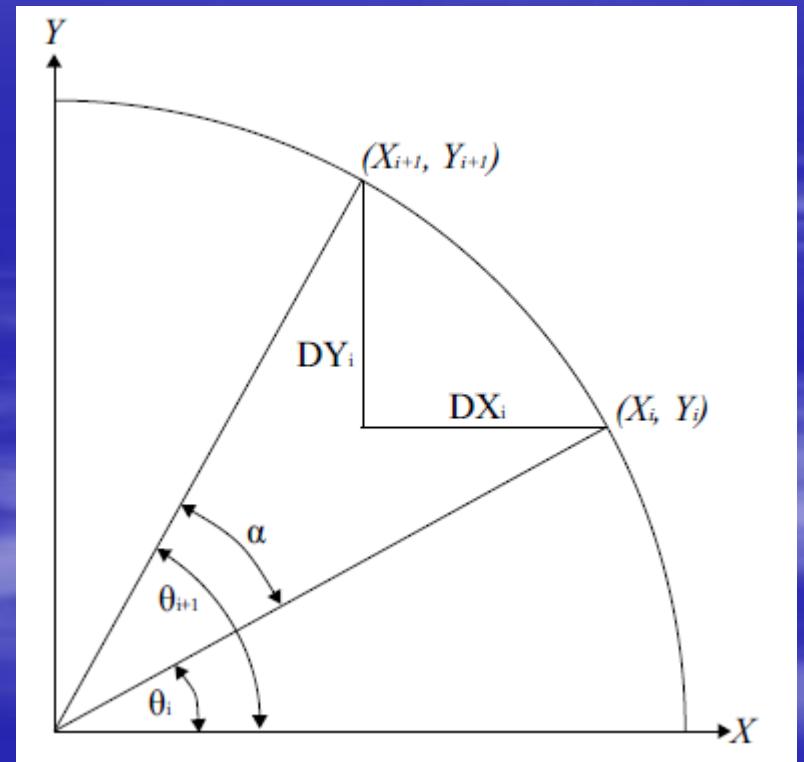
$$DX(i) = X(i+1) - X(i) = (A - 1)X(i) - BY(i)$$

$$DY(i) = Y(i+1) - Y(i) = (A - 1)Y(i) + BX(i)$$

$$Vx(i) = \frac{VDX(i)}{DS(i)}$$

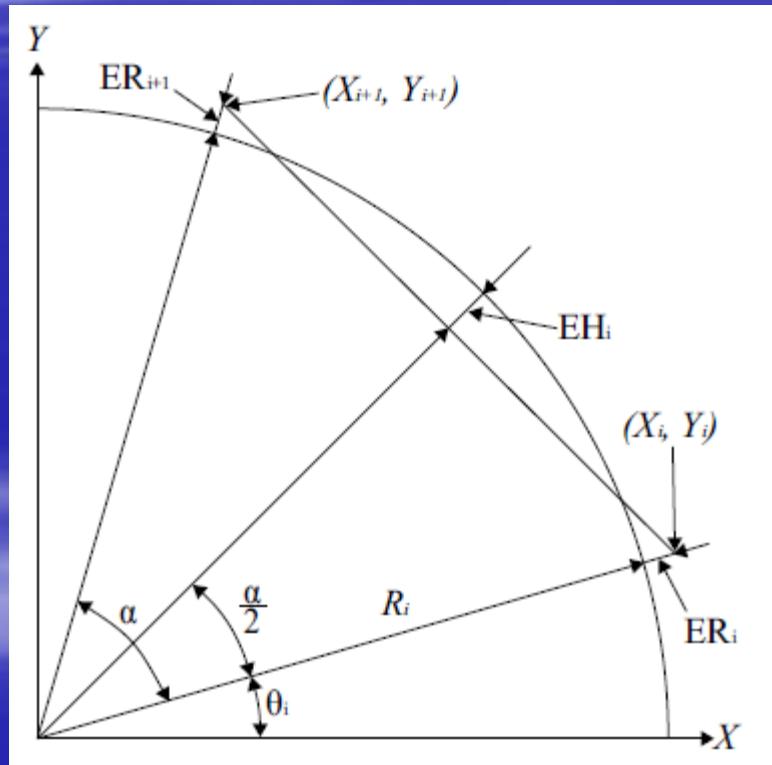
$$Vy(i) = \frac{VDY(i)}{DS(i)}$$

$$DS(i) = \sqrt{DX^2(i) + DY^2(i)}$$



# Radial Error and Chord Height Error

- Radial error ( $ER$ ) and chord height error ( $EH$ ).



# Radial Error and Chord Height Error

- $ER$  is an error from a truncation effect.  $ER$  can be approximated by coefficients  $A$  and  $B$  and this error is **accumulated** with iteration. At the  $i$ th iteration,  $ER$  can be computed approximately using

$$ER(i) = i(C - 1)R$$

where  $C = \sqrt{A^2 + B^2}$

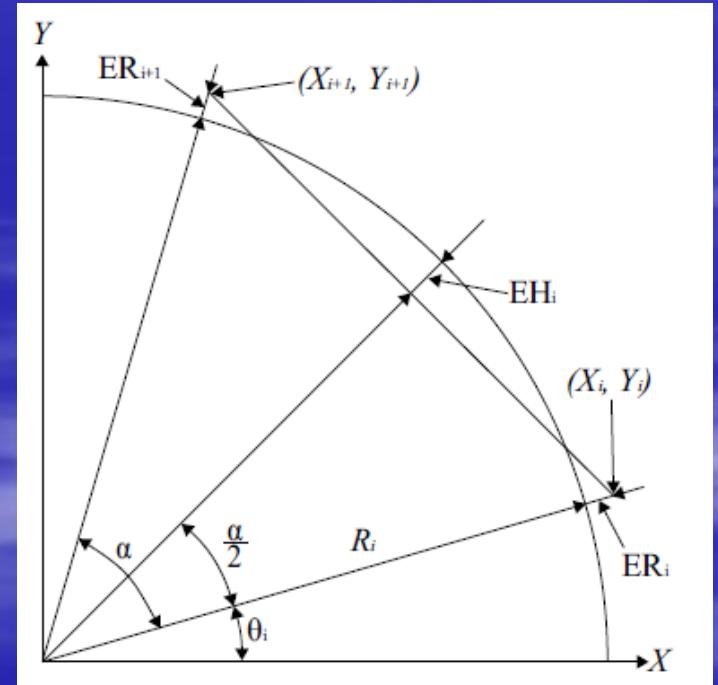
# Radial Error and Chord Height Error

- $EH$ , chord height error, is not accumulated

$$EH(i) = R - R(i)\cos \frac{\alpha}{2}$$

$$\cos \frac{\alpha}{2} = \sqrt{\frac{1 + \cos \alpha}{2}} = \sqrt{\frac{1 + A}{2}}$$

$$EH(i) = R - R(i)\sqrt{\frac{1 + A}{2}}$$



# Reference Word Interpolator for Circles

$$Vx(i) = \frac{V(A-1)X(i) - BY(i)}{\sqrt{((A-1)X(i) - BY(i))^2 + ((A-1)Y(i) + BX(i))^2}}$$

$$Vy(i) = \frac{V(A-1)Y(i) + BX(i)}{\sqrt{((A-1)X(i) - BY(i))^2 + ((A-1)Y(i) + BX(i))^2}}$$

$$ER(i) = i \left( \sqrt{A^2 + B^2} - 1 \right) R$$

$$EH(i) = R - R(i) \sqrt{\frac{1+A}{2}}$$

$$A = \cos \alpha \quad B = \sin \alpha$$

# Euler Algorithm

- Euler algorithm,  $\cos\alpha$  and  $\sin\alpha$  are approximated by first-order Taylor series expansion.

$$A = 1, B = \alpha$$

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} = x - \frac{x^3}{3!} + \frac{x^5}{5!} \dots$$

$$\cos x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} \dots$$

# Euler Algorithm

- Since the series expansion is truncated, a radial error  $ER$  influences the accuracy of the algorithm

$$ER_{\max} = (\pi/2\alpha) \left( \sqrt{1 + \alpha^2} - 1 \right) R$$

$$ER_{\max} \approx (\pi/4)\alpha R \quad \text{For small } \alpha$$

# Euler Algorithm

- Quarter circle
- Angle  $\alpha$  :

$$\alpha = 4/(\pi R)$$

- Number of iteration steps  $N$  :

$$N = \pi^2 R / 8$$

# Improved Euler Algorithm

$$X(i+1) = AX(i) - BY(i) = X(i) - \alpha Y(i)$$

$$Y(i+1) = AY(i) + BX(i+1) = (1 - \alpha^2)Y(i) + \alpha X(i)$$

- The radial error  $ER$  of the Improved Euler algorithm is maximized at

$$\theta = \pi/4$$

or

$$X(i) = (1 + \alpha)Y(i)$$

# Improved Euler Algorithm

- Quarter circle
- Angle  $\alpha$  :

$$\alpha = 4/R$$

- Number of iteration steps  $N$  :

$$N = \pi R / 8$$

# Taylor Algorithm

- In the Taylor algorithm, the coefficients  $A$  and  $B$  are approximated as a truncated series.

$$A = 1 - \frac{1}{2} \alpha^2, B = \alpha$$

$$\sin x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n+1)!} x^{2n+1} = x - \frac{x^3}{3!} + \frac{x^5}{5!} \dots$$

$$\cos x = \sum_{n=0}^{\infty} \frac{(-1)^n}{(2n)!} x^{2n} = 1 - \frac{x^2}{2!} + \frac{x^4}{4!} \dots$$

# Taylor Algorithm

- Maximum radial error

$$ER_{\max} = \frac{\pi}{2\alpha} \left( \sqrt{1 + \frac{1}{4}\alpha^4} - 1 \right) R \cong \pi R \alpha^3 / 16$$

- Chord height error  $EH$

$$EH = R - R(i) \sqrt{1 + \frac{1}{4}\alpha^4} = R - R(i) + (\alpha^2/8)R(i)$$

$$EH_{\max} = R\alpha^2/8 \quad \text{When } R(i) = R$$

$$ER_{\max} = \frac{\pi\alpha}{2} EH_{\max}$$

# Taylor Algorithm

- Quarter circle
- Angle  $\alpha$  :

$$\alpha = \sqrt{8/R}$$

- Number of iteration steps  $N$  :

$$N = \pi/8\alpha$$

# Tustin Algorithm

- The Tustin algorithm is based on an approximation relationship between differential operators and a discrete variable z

$$s = \frac{2}{T} \left( \frac{z - 1}{z + 1} \right)$$

- A and B :

$$A = \frac{1 - (\alpha/2)^2}{1 + (\alpha/2)^2} \quad B = \frac{\alpha}{1 + (\alpha/2)^2}$$

# Tustin Algorithm

$$DX(i) = -\frac{1}{1+(\alpha/2)^2} \left[ \frac{\alpha^2}{2} X(i) + \alpha Y(i) \right]$$

$$DY(i) = \frac{1}{1+(\alpha/2)^2} \left[ \frac{-\alpha^2}{2} Y(i) + \alpha X(i) \right]$$

$$V_x(i) = -\frac{V}{R[1+(\alpha/2)^2]} \left[ \frac{\alpha}{2} X(i) + Y(i) \right]$$

$$V_y(i) = \frac{V}{R[1+(\alpha/2)^2]} \left[ -\frac{\alpha}{2} Y(i) + X(i) \right]$$

# Tustin Algorithm

- ER :

$$ER(i) = 0$$

- EH:

$$EH(i) = R - \frac{R}{\sqrt{1 + (\alpha/2)^2}}$$

$$EH(i) = \frac{\alpha^2}{\alpha^2 + 8} R \quad \alpha \text{ is very small}$$

# Tustin Algorithm

- Quarter circle
- Angle  $\alpha$  :

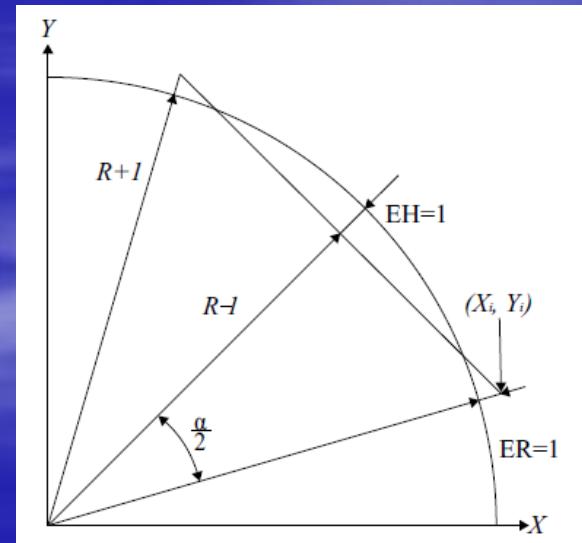
$$\alpha = \sqrt{\frac{8}{R-1}} \approx \sqrt{\frac{8}{R}}$$

- Number of iteration steps  $N$  :

$$N = \frac{\pi}{4} \sqrt{R/2}$$

# Improved Tustin Algorithm

- Idea is ER : 0 → 1
- Increase angle  $\alpha$  and the efficiency of the algorithm increases
- ER = 1, EH = 1

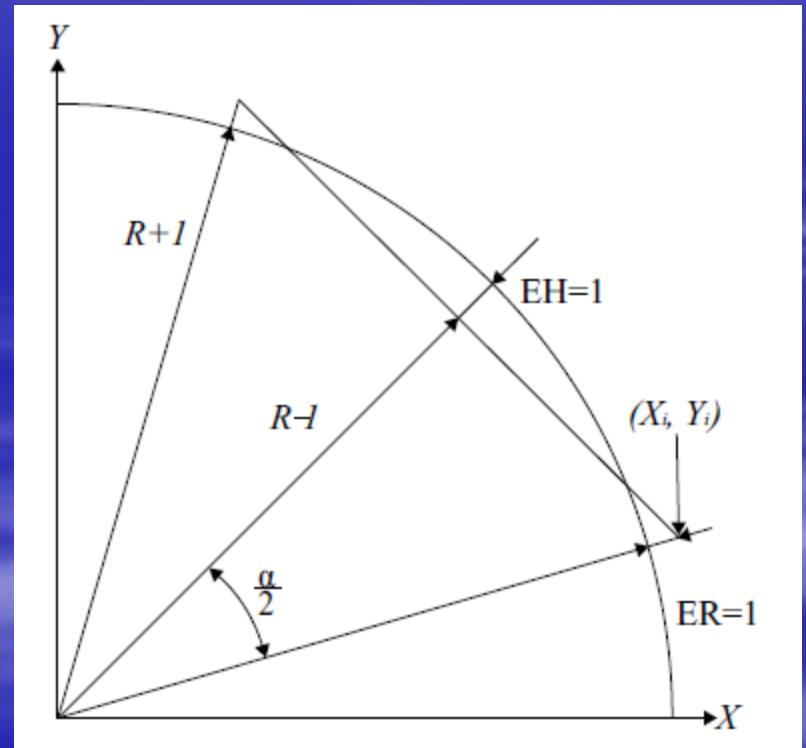


# Improved Tustin Algorithm

$$\cos \frac{\alpha}{2} = \frac{R-1}{R+1}$$

$$\frac{R+1}{R-1} = \sqrt{\frac{2}{1+A}} = \sqrt{1 + \left(\frac{\alpha}{2}\right)^2} \cong 1 + \frac{\alpha^2}{8}$$

$$\alpha = \sqrt{\frac{16}{R-1}} \cong \frac{4}{\sqrt{R}}$$



# Improved Tustin Algorithm

- Quarter circle
- Angle  $\alpha$  :

$$\alpha = \sqrt{\frac{16}{R-1}} \approx \frac{4}{\sqrt{R}}$$

- Number of iteration steps  $N$  :

$$N = \frac{\pi}{8} \sqrt{R}$$

# Improved Tustin Algorithm

- General case when  $ER$  and  $EH$  are set to  $\beta$

$$R_i = R + \beta$$

$$\cos \frac{\alpha}{2} = \frac{R - \beta}{R + \beta}$$

$$\alpha = 2 \cos^{-1} \left( \frac{R - \beta}{R + \beta} \right)$$

# Sampled-Data Interpolation

- characteristics

Method	$\alpha$	$N$
Euler	$4/\pi R$	$\pi^2 R/8$
IEM	$4/R$	$\frac{\pi}{4} \sqrt{R/2}$
Taylor	$\sqrt{8/R}$	$\frac{\pi}{4} \sqrt{R/2}$
Tustin	$\sqrt{8/R}$	$\frac{\pi}{4} \sqrt{R/2}$
ITM	$4/\sqrt{R}$	$\frac{\pi}{8} \sqrt{R}$

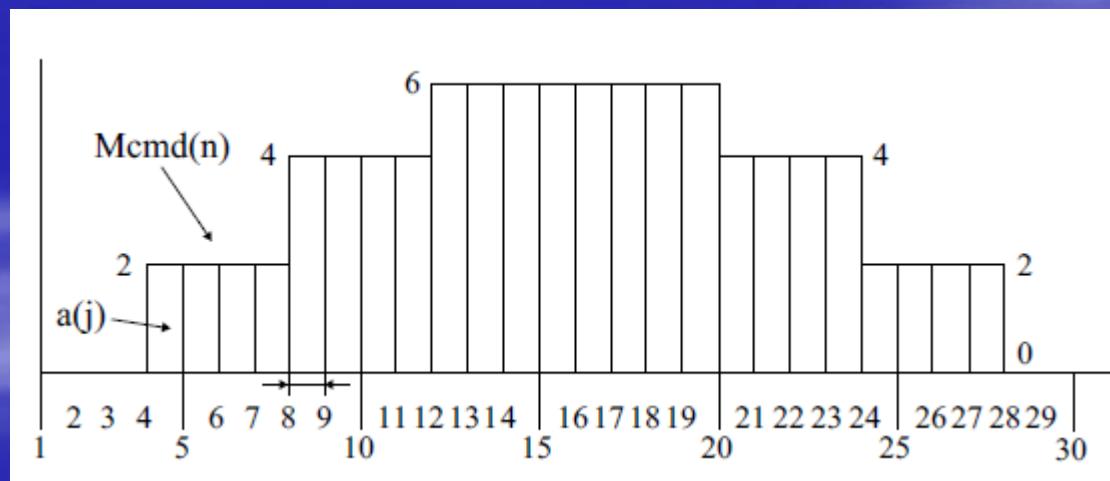
# Fine Interpolation

- When the **sampling interval** for rough interpolation and pulse train after acceleration/deceleration is **larger than** that of **position control**, fine interpolation is performed.
- EX:
  - sampling interval for rough interpolation : 4ms
  - sampling interval for position control : 1ms

# Fine Interpolation

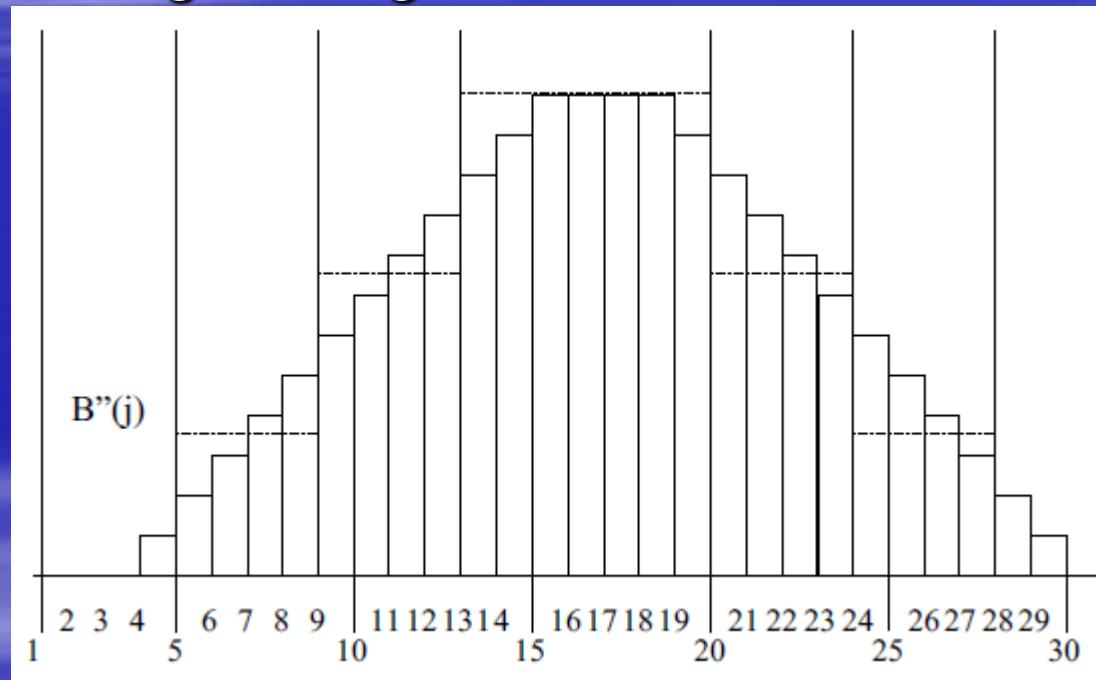
- Linear method

$$a(j) = \frac{p(i)}{N}, i \leq j < i + t_{ipo}$$



# Fine Interpolation

- Moving average method



$$b(j) = \frac{\sum_{k=-\frac{N}{2}+1}^{\frac{N}{2}} a(j-k)}{N}, b'(j) = \frac{\sum_{k=-\frac{N}{2}}^{\frac{N}{2}-1} a(j-k)}{N}, b''(j) = \frac{b(j)+b'(j)}{2}$$

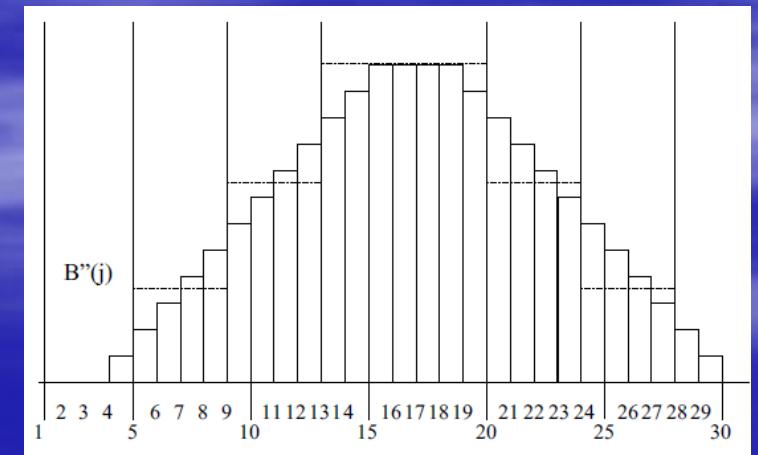
# Fine Interpolation

$n$	$j$	$a(j+4)$	$a(j+3)$	$a(j+2)$	$a(j+1)$	$a(j)$	$a(j-1)$	$a(j-2)$	$b(j)$	$b'(j)$	$b''(j)$
1	1	2	0	0	0	0	0	0	0	0	0
	2	2	2	0	0	0	0	0	0	0	0
	3	2	2	2	0	0	0	0	0	$\frac{1}{2}$	0.25
	4	2	2	2	2	0	0	0	$\frac{1}{2}$	1	0.75
2	5	4	2	2	2	2	0	0	1	$1\frac{1}{2}$	1.25
	6	4	4	2	2	2	2	0	$1\frac{1}{2}$	2	1.75
	7	4	4	4	2	2	2	2	2	$2\frac{1}{2}$	2.25
	8	4	4	4	4	2	2	2	$2\frac{1}{2}$	3	2.75
3	9	6	4	4	4	4	2	2	3	$3\frac{1}{2}$	3.25
	10	6	6	4	4	4	4	2	$3\frac{1}{2}$	4	3.75
	11	6	6	6	4	4	4	4	$4\frac{1}{2}$	$4\frac{1}{2}$	4.25
	12	6	6	6	6	4	4	4	$4\frac{1}{2}$	5	4.75
4	13	6	6	6	6	6	4	4	5	$5\frac{1}{2}$	5.25
	14	6	6	6	6	6	6	4	$5\frac{1}{2}$	6	5.75
	15	6	6	6	6	6	6	6	6	6	6
	16	6	6	6	6	6	6	6	6	6	6
5	17	4	6	6	6	6	6	6	6	6	6
	18	4	4	6	6	6	6	6	6	6	6
	19	4	4	4	6	6	6	6	6	$5\frac{1}{2}$	5.75
	20	4	4	4	4	6	6	6	$5\frac{1}{2}$	5	5.25
6	21	2	4	4	4	4	6	6	5	$4\frac{1}{2}$	4.75
	22	2	2	4	4	4	4	6	$4\frac{1}{2}$	4	4.25
	23	2	2	2	4	4	4	4	$4\frac{1}{2}$	$3\frac{1}{2}$	3.75
	24	2	2	2	2	4	4	4	$3\frac{1}{2}$	3	3.25
7	25	0	2	2	2	2	4	4	3	$2\frac{1}{2}$	2.75
	26	0	0	2	2	2	2	4	$2\frac{1}{2}$	2	2.25
	27	0	0	0	2	2	2	2	2	$1\frac{1}{2}$	1.75
	28	0	0	0	0	2	2	2	$1\frac{1}{2}$	1	1.25
8	29	0	0	0	0	0	2	2	1	$\frac{1}{2}$	0.75
	30	0	0	0	0	0	0	2	$\frac{1}{2}$	0	0.25
	31	0	0	0	0	0	0	0	0	0	0
	32	0	0	0	0	0	0	0	0	0	0

$$b(j) = \frac{a(j+1)a(j)a(j-1)a(j-2)}{4}$$

$$b'(j) = \frac{a(j+2)a(j+1)a(j)a(j-1)}{4}$$

$$b''(j) = \frac{b(j)+b'(j)}{2}$$



# NURBS Interpolation

- In CNC free-form curves can be approximated by a set of line segments or circle arcs
- Short segments result in **inconsistency of feedrate**
- **Reduces the surface quality**
- Many blocks are required to define these short path sand the size of the part program **increases dramatically**

# NURBS Interpolation

- CNC itself directly converts NURBS curve data from the part program into small line segments, using positions calculated from the NURBS curve data.
- Reduce the size of the part program
- Increase the machining speed

# NURBS Equation Form

- With NURBS geometry it is possible to define free-form curves with complex shapes by using **less data** and to **represent various geometric shapes by changing parameters.**
- NURBS geometry is generally used in CAD/CAM systems.

# NURBS Equation Form

- Mathematical form of a NURBS curve

$$p(u) = \frac{\sum_{i=0}^n N_{i,p}(u) w_i P_i}{\sum_{i=0}^n N_{i,p}(u) w_i} \quad a \leq u \leq b$$

- where  $N_{i,p}(u)$  is a B-spline basis function

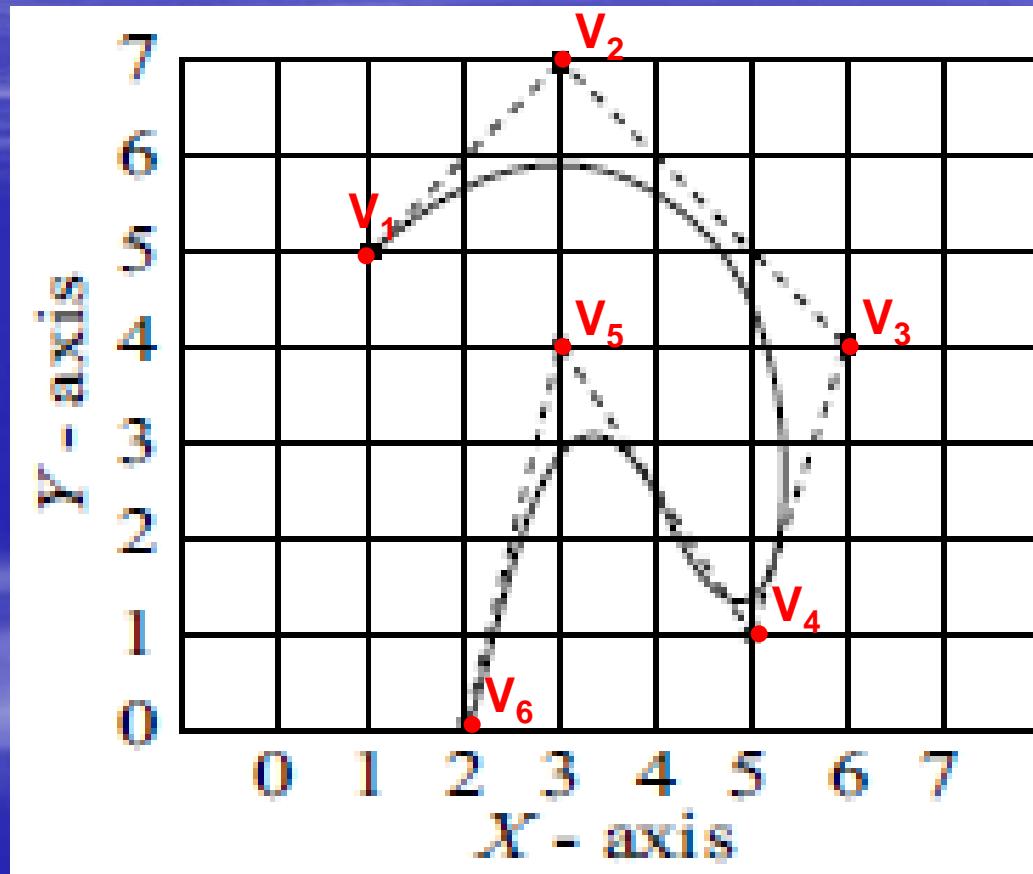
$$N_{i,0}(u) = \begin{cases} 1 & \text{if } u_i \leq u \leq u_{i+1} \\ 0 & \text{otherwise} \end{cases}$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u)$$

# NURBS Geometric Characteristics

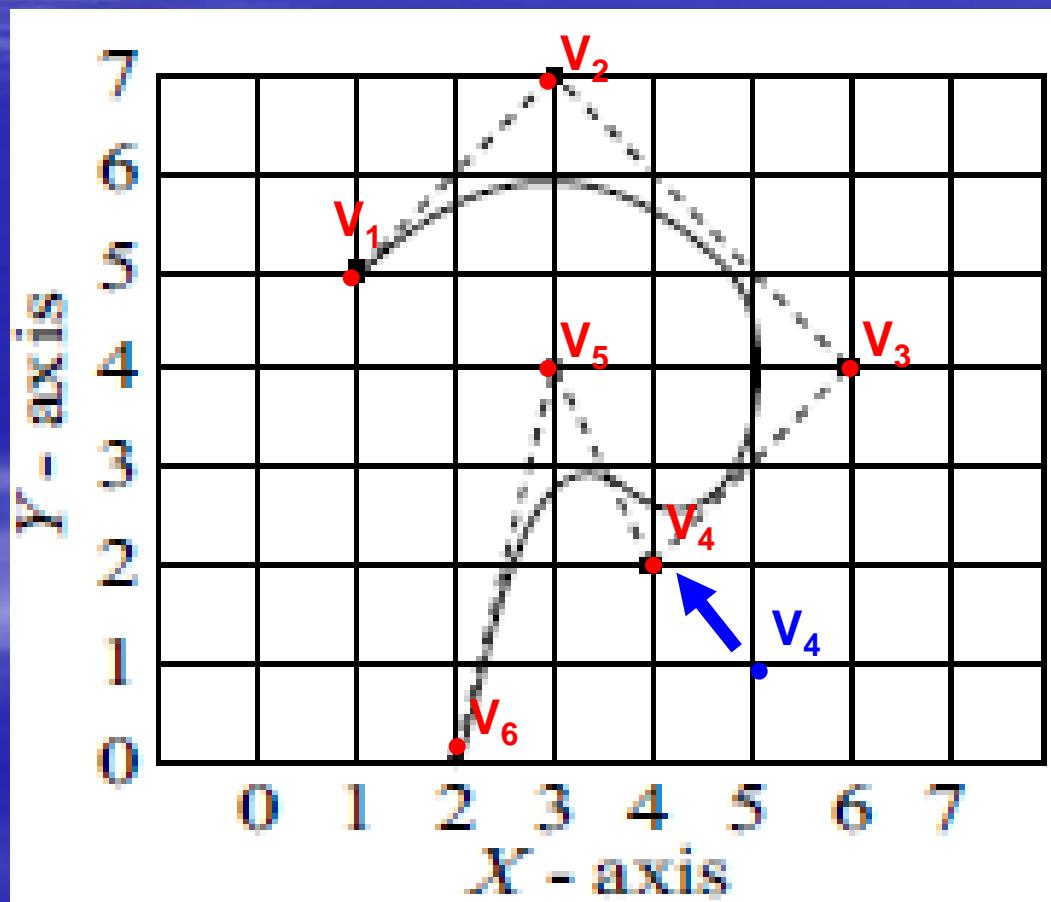
- The shape of a NURBS curve is defined based on **control points**, **knots**, and **weights**.
- Control points define the basic **position of the curve**.
- Weights decide the **importance** of individual **control points**.
- Knots decide the **tangents of curves**.

# NURBS Geometric Characteristics



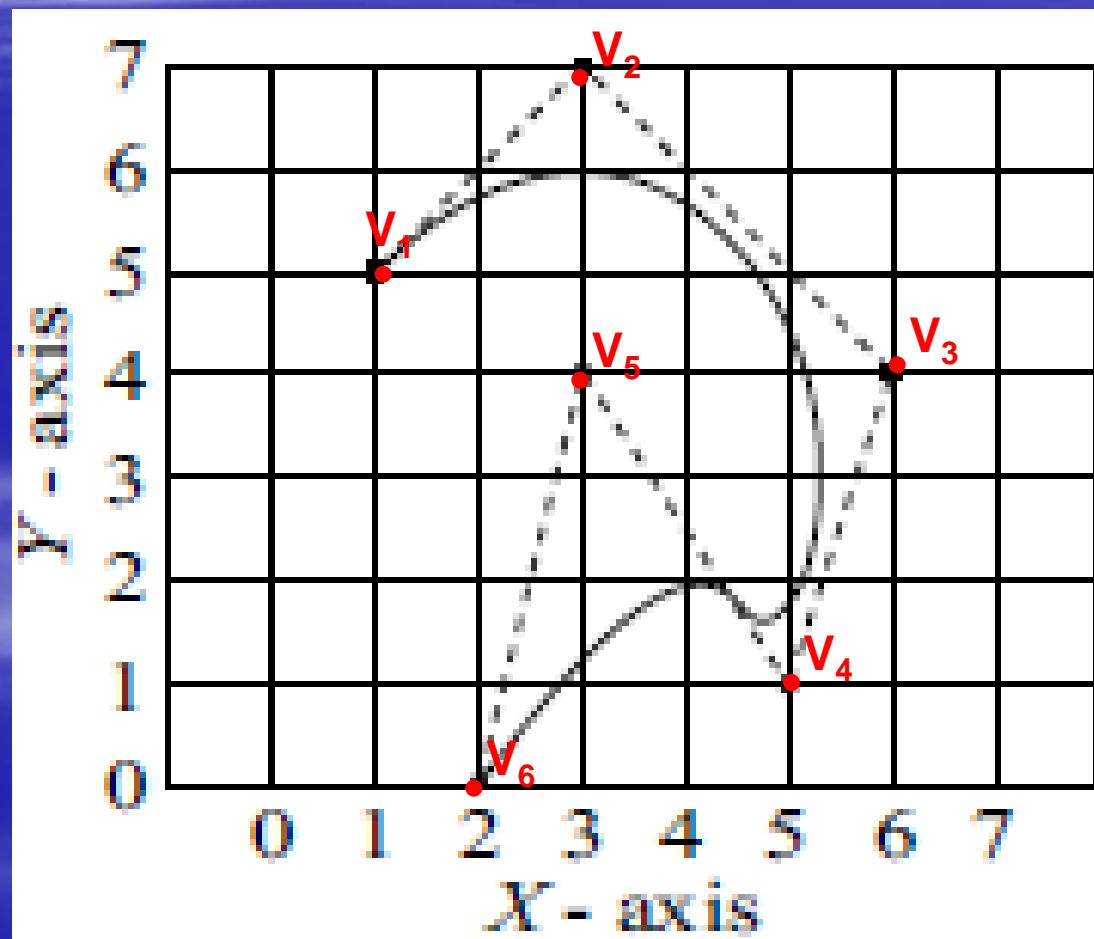
# NURBS Geometric Characteristics

move control point V4



# NURBS Geometric Characteristics

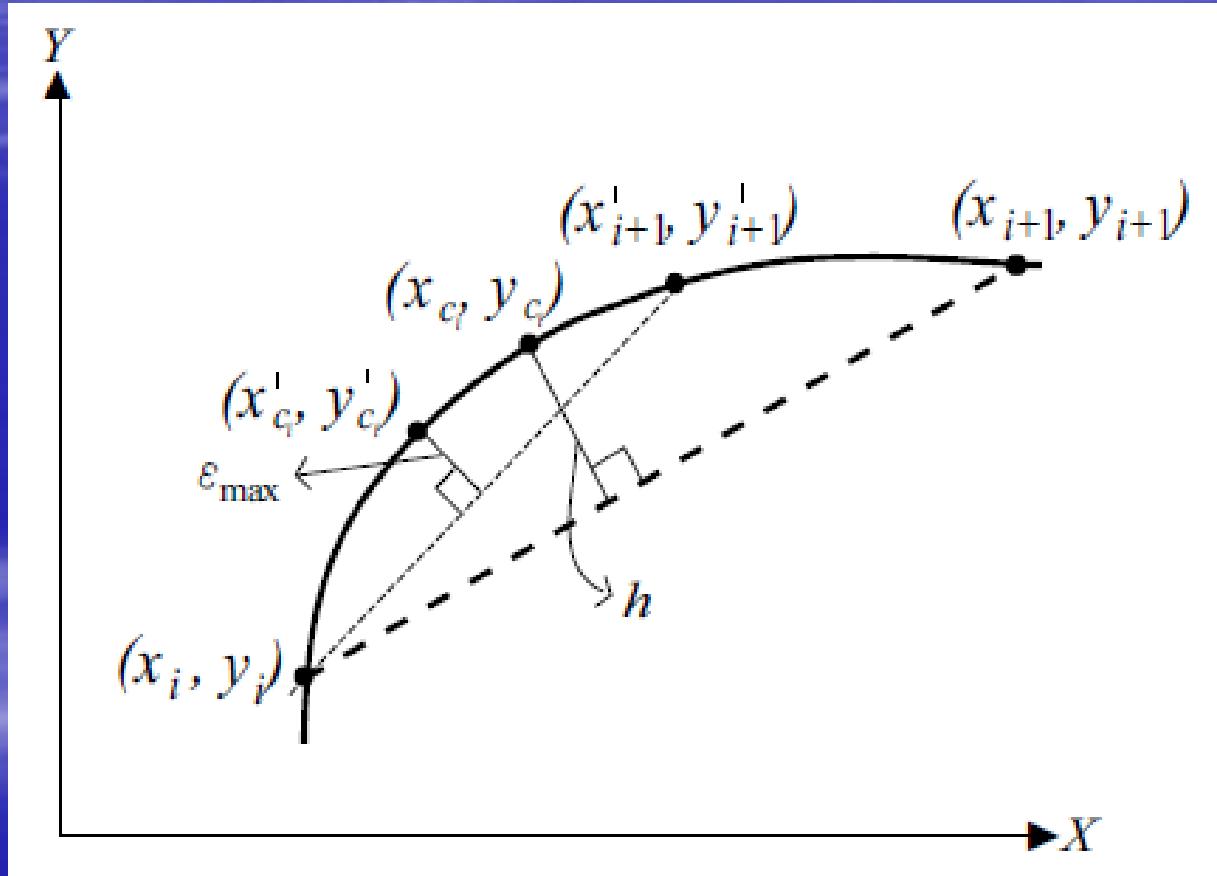
Change weight of control point V6



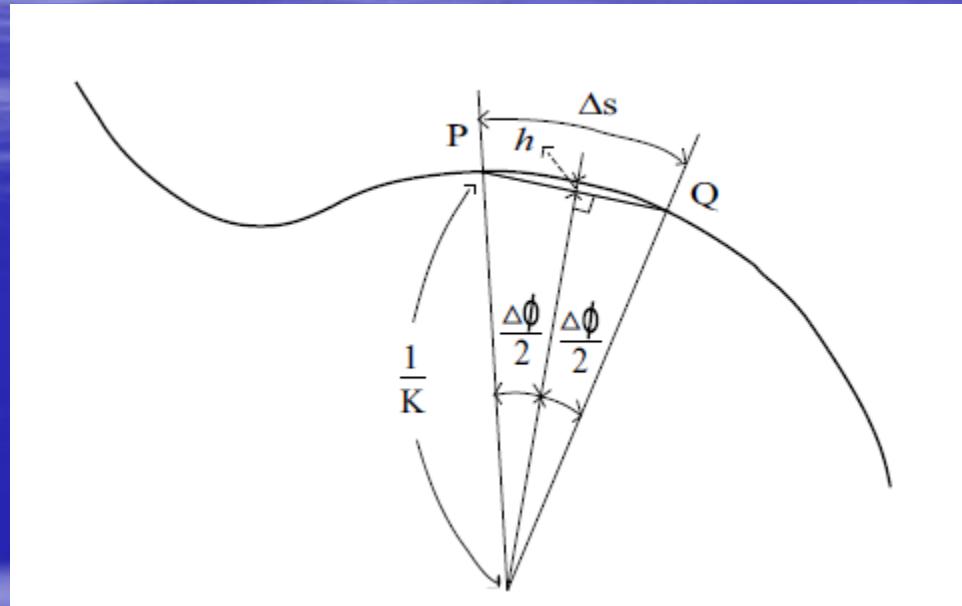
# NURBS Interpolation Algorithm

- Two stages
- successive interpolated points are obtained with a **maximum allowable interpolation error**.
- Check to determine whether it exceeds the **allowable acceleration**. If necessary, a new interpolated point is calculated that satisfies the allowable acceleration.

# NURBS Interpolation Errors



# NURBS Interpolation Errors



$$\kappa \equiv \lim_{\Delta s \rightarrow 0} \frac{\Delta\phi}{\Delta s},$$

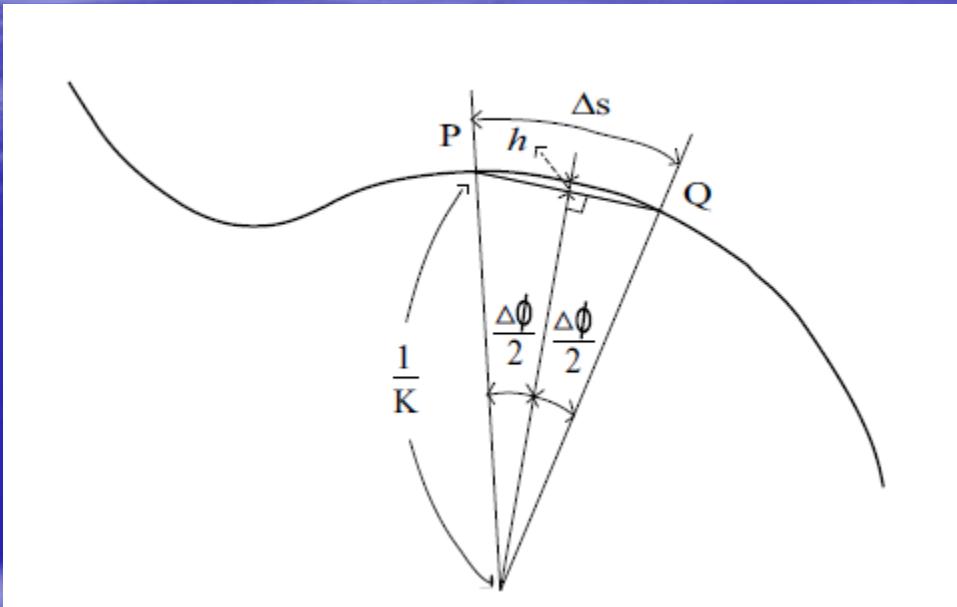
$$R \equiv \frac{1}{\kappa}$$

where  $\kappa$  : is the curvature  
 $R$  : is the radius of curvature  
 $\Delta\phi$  : angle at the circumference  
 $\Delta s$  : the length of the partial curve

$$h = \frac{1}{\kappa} - \frac{1}{\kappa} \times \cos\left(\frac{\Delta\phi}{2}\right) \approx \frac{1}{\kappa} - \frac{1}{\kappa} \times \left[1 - \frac{1}{2} \times \left(\frac{\Delta\phi}{2}\right)^2\right] = \frac{\Delta\phi^2}{8\kappa}$$

approximate by a second-order Taylor series expansion.

# NURBS Interpolation Errors



$$\kappa \equiv \lim_{\Delta s \rightarrow 0} \frac{\Delta\phi}{\Delta s},$$

$$R \equiv \frac{1}{\kappa}$$

where  $\kappa$  : is the curvature

$R$  : is the radius of curvature

$\Delta\phi$  : angle at the circumference

$\Delta s$  : the length of the partial curve

$$\Delta\phi = 2\sqrt{2\kappa h} \quad \kappa \cong \frac{\Delta\phi}{\Delta s} \cong \frac{\Delta\phi}{PQ} \cong \frac{\sqrt{8\kappa h}}{PQ} \quad K = \frac{8h}{PQ^2}$$

# NURBS Interpolation Errors

$$h \cong \kappa \times \frac{\overline{PQ}^2}{8} = \kappa \times \frac{(F \times \Delta T)^2}{8}$$

where  $F$  : is the feedrate

$\Delta T$  : is the interpolation iteration time

$\overline{PQ}$  :  $F \times \Delta T$

$$h = \varepsilon_{max}$$

$$F_\varepsilon = \frac{2}{\Delta T} \sqrt{\frac{2\varepsilon_{max}}{\kappa}}$$

where  $\varepsilon_{max}$  : maximum allowable interpolation error

$F_\varepsilon$  : allowable feedrate speed

$$\Delta L = F_\varepsilon(t) \times \Delta T = \sqrt{\frac{8 \times \varepsilon_{max}}{\kappa}}$$

## Acceleration Control keeping Axis-Velocity Limit

if  $\frac{\left| \frac{\Delta X}{\Delta T_i} - \frac{\Delta X}{\Delta T_{i+1}} \right|}{\Delta T} > A_{max}$

$$\left( \frac{\Delta X}{\Delta T_{i+1}} \right) = \frac{\Delta X}{\Delta T_i} + A_{max} \times \Delta T \quad (\text{when } \frac{\Delta X}{\Delta T_i} - \frac{\Delta X}{\Delta T_{i+1}} > 0)$$

$$\left( \frac{\Delta X}{\Delta T_{i+1}} \right) = \frac{\Delta X}{\Delta T_i} - A_{max} \times \Delta T \quad (\text{when } \frac{\Delta X}{\Delta T_i} - \frac{\Delta X}{\Delta T_{i+1}} < 0)$$

$$\Delta L_{new} = \sqrt{\left( \frac{\Delta X}{\Delta T_{i+1}} \right)^2 + \left( \frac{\Delta Y}{\Delta T_{i+1}} \right)^2}$$

# Summary

- **Hardware → NC systems**
  - DDA
- **Software → Modern CNC systems**
  - reference pulse → high-accuracy
  - sampled-data → high-speed
- **NURBS**
  - still under investigation
  - speed reduction, poor surface quality, poor machining accuracy  
→ been solved

# Theory and Design of CNC Systems

*Chapter 4*

## Acceleration and Deceleration

# OutLine

- Introduction
- Acc/Dec Control After Interpolation
  - Acc/Dec Control by Digital Filter
  - Acc/Dec Control by Digital Circuit
    - Linear-type Acc/Dec Control
    - S-shape-type Acc/Dec Control
    - Exponential-type Acc/Dec Control
  - *Acc/Dec Control Machining Errors*
    - Machining Error with Linear Type Acc/Dec Control
    - Machining Error with S-shape-type Acc/Dec Control
    - Machining Error with Exponential Type Acc/Dec Control
    - Machining Error Summary
  - *Block Overlap in ADCAI*

# OutLine

- Acc/Dec Control Before Interpolation
  - Speed-profile Generation
  - Block Overlap Control
    - Classification of Continuous Blocks
    - Normal Block/Normal Block, Identical Speed
    - Normal Block (High Speed)/Normal Block (Low Speed)
    - Normal Block (Low Speed)/Normal Block (High Speed)
    - Short Block/Normal Block with Identical Speed
    - Short Block/Normal Block with Different Speed
    - Normal Block/Short Block with Identical Speed
    - Normal Block/Short Block with Different Speed
    - Short Block/Short Block with Identical Speed
    - Short Block (High Speed)/Short Block (Low Speed)
    - Short Block (Low Speed)/Short Block (High Speed)
    - Overlap Between a Linear and a Circular Profile
  - Corner Speed of Two Blocks Connected by an Acute Angle
  - Corner Speed Considering Speed Difference of Each Axis
- Look Ahead
  - Look-Ahead Algorithm
    - Look Ahead with Respect to Length
    - Speed at a Corner
    - Look Ahead considering Length and Corner
    - Speed within Block
  - Simulation Results
- Summary

# Introduction(1)

- Processing order of acceleration and deceleration control
  - Acc/Dec control **before** interpolation (ADCBI)
  - Acc/Dec control **after** interpolation (ADCAI)

# Introduction(2)

## ■ ADCBI

- constructed **differently** according to the interpolation type such as linear-, exponential- and S-curve-type interpolation.
- needs to **hold** a lot of information, related to **all the interpolated points**.
- does **not** result in **machining error** because of the increased accuracy.
- requires more **computing power** and **larger memory**

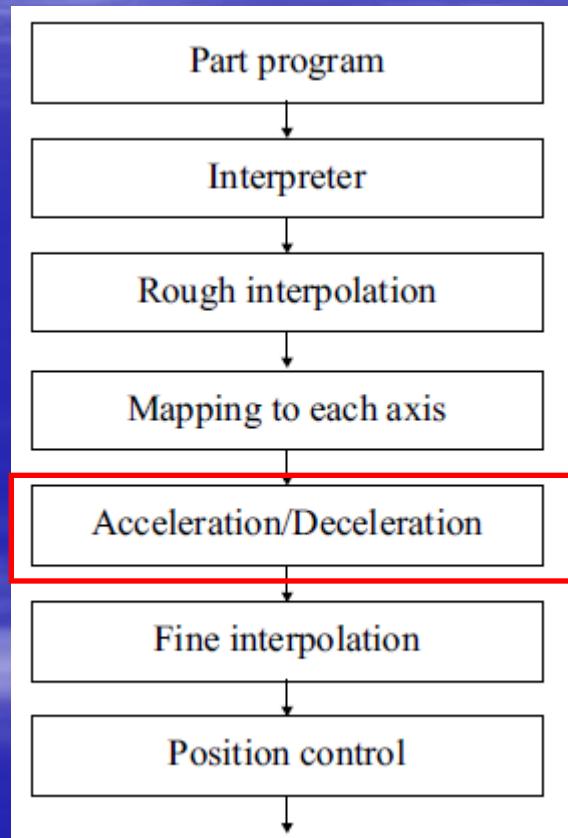
# Introduction(3)

## ■ ADCAI

- is applied in an **identical manner** for all interpolation methods.
- implementation is **simple** but **machining errors occur** because each **axis movement is determined separately**
- the interpolated points **deviate from the desired path.**

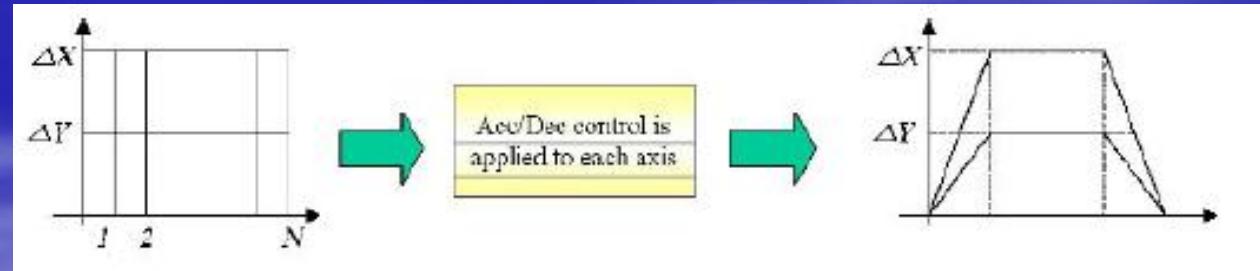
# Acc/Dec Control After Interpolation

## ■ ADCAI



# Acc/Dec Control After Interpolation

- has been widely used for NC and Motion Control systems in both hardware and software interpolation.



Change to pulse profile after Acc/Dec control

# Acc/Dec Control by Digital Filter

- Digital filter theory
  - if **input signal  $x[n]$**  is entered into the filter with **impulse response  $h[n]$** , the output signal  $y[n]$  is represented by the convolution of  $h[n]$  and  $x[n]$ .
- the general convolution of  $f_1[n]$  and  $f_2[n]$  for a discrete time system.

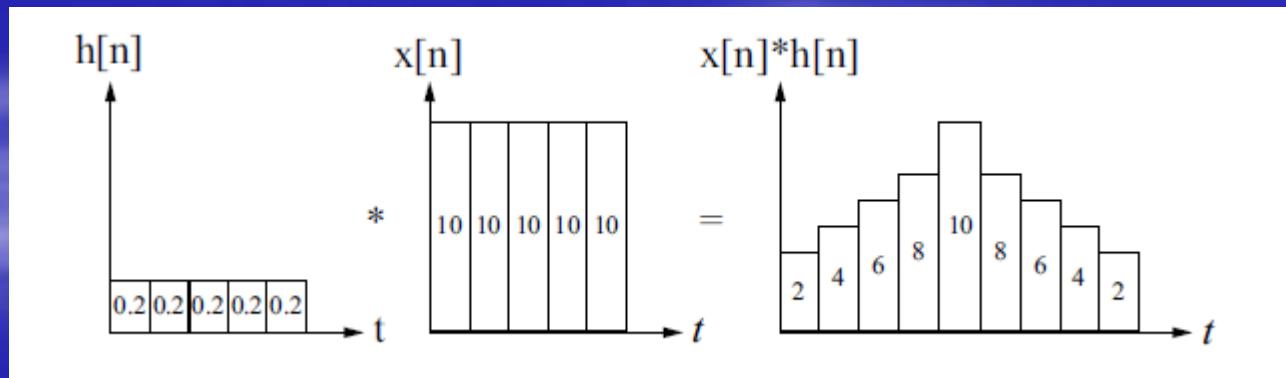
$$f[n] = f_1[n] * f_2[n]$$

$$= f_1[0]f_2[n] + \dots + f_1[k]f_2[n-k] + \dots + f_1[n]f_2[0]$$

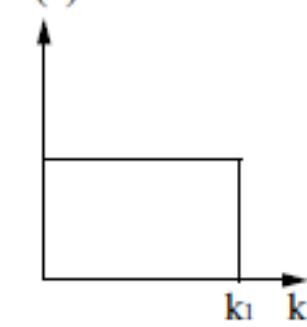
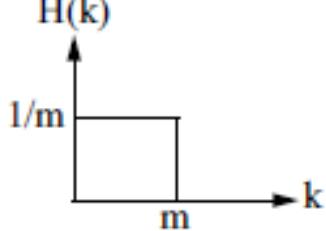
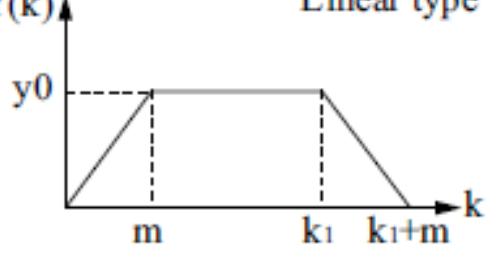
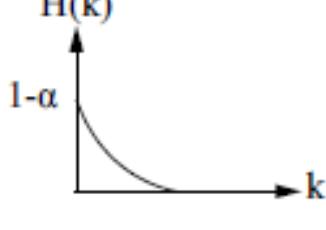
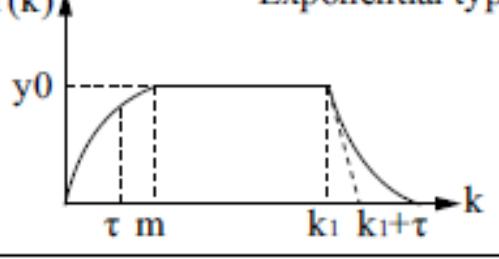
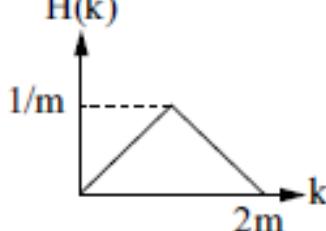
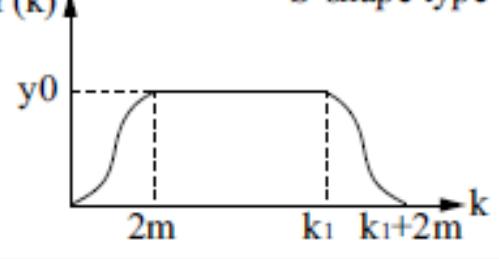
$$f[n] = f_1[n] * f_2[n] = \sum_{k=1}^n f_1[k]f_2[n-k]$$

# Acc/Dec Control by Digital Filter

- $x[n]$  is defined as the output of a rough interpolator
- $h[n]$  as the impulse response that has the normalized unit summation



# Acc/Dec Control by Digital Filter

Input pulse train	Impulse response	Output pulse train
$X(k)$ 	$H(k)$ 	$Y(k)$ Linear type 
	$H(k)$ 	$Y(k)$ Exponential type 
	$H(k)$ 	$Y(k)$ S-shape type 

# Acc/Dec Control by Digital Filter

- By using various digital filters different output profiles can be obtained even when identical input pulses are used.

$$H_L(z) = \frac{1}{m} \frac{1-z^{-m}}{1-z^{-1}}$$

Linear-type filter

$$H_E(z) = \frac{1-\alpha}{1-\alpha z^{-1}} \quad \text{where } \alpha = \exp^{-\frac{T}{\tau}}$$

Exponential-type filter

$$H_S(z) = H_L(z) * H_E(z) = \frac{1}{m} \frac{1-z^{-m}}{1-z^{-1}} * \frac{1}{m} \frac{1-z^{-m}}{1-z^{-1}}$$

S-shape-type filter

T : sampling time;  $\tau$  : time constant for Acc/Dec control

# Acc/Dec Control by Digital Filter

- the Acc/Dec pulse profile generated by passing the input signal  $V_i$  through the above-mentioned filters can be represented by a recursive equation.
- recursive equations :

$$V_{LO}(k) = \frac{1}{m} (V_i(k) - V_i(k-m)) + V_O(k-1)$$

Linear-type Acc/Dec pulse profile

$$V_{EO}(k) = (1-\alpha)(V_i(k) - V_O(k-1)) + V_O(k-1)$$

Exponential-type Acc/Dec pulse profile

$$V_{SO}(k) = \frac{1}{m} (V_i(k) - V_i(k-m)) + V_{Otemp}(k-1)$$

S-shape-type Acc/Dec pulse profile

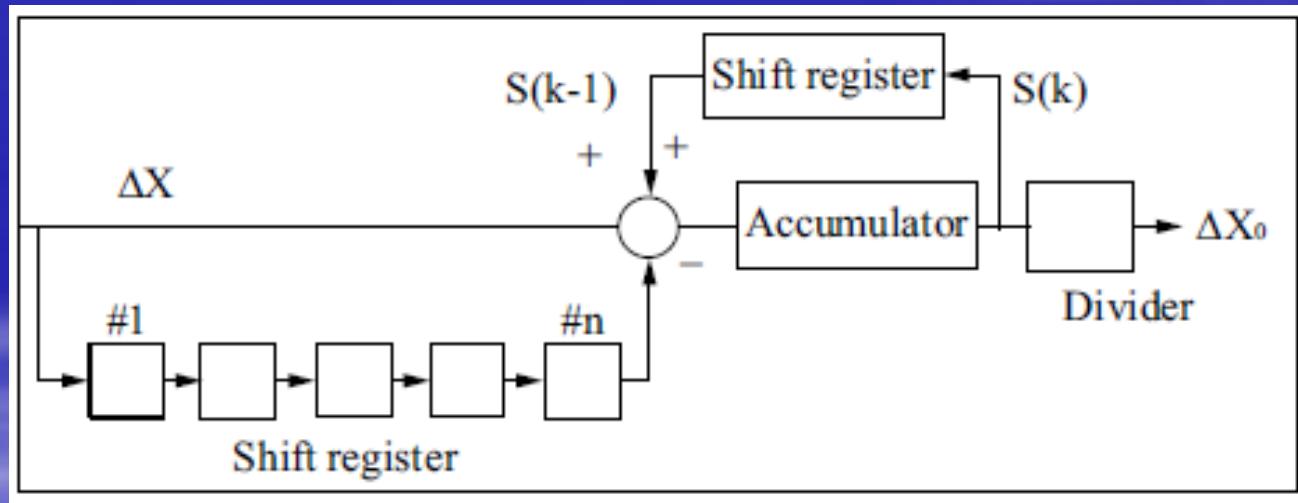
$$\text{where } V_{Otemp}(k) = \frac{1}{m} (V_{Otemp}(k) - V_{Otemp}(k-m)) + V_O(k-1)$$

# Acc/Dec Control by Digital Circuit

- Hardware : shift register, a divider and an accumulator
- Software : simple recursive equation & short calculation time.
- the pulse profile from rough interpolation is used as input of the Acc/Dec control circuit.
- The Acc/Dec control circuit plays the role of smoothing the change of pulse amount at the beginning and the end of a pulse profile.

# Linear-type Acc/Dec Control

- Hardware : Adder, Accumulator, SUM, Divider unit connections



$$S(k) = S(k-1) + \Delta X(k) - \Delta X(k-n)$$

$$\Delta X_0(k) = S(k)/n$$

$$V_{LO}(k) = \frac{1}{m} (V_i(k) - V_i(k-m)) + V_o(k-1)$$

# Linear-type Acc/Dec Control

- EX :  $n = 5$

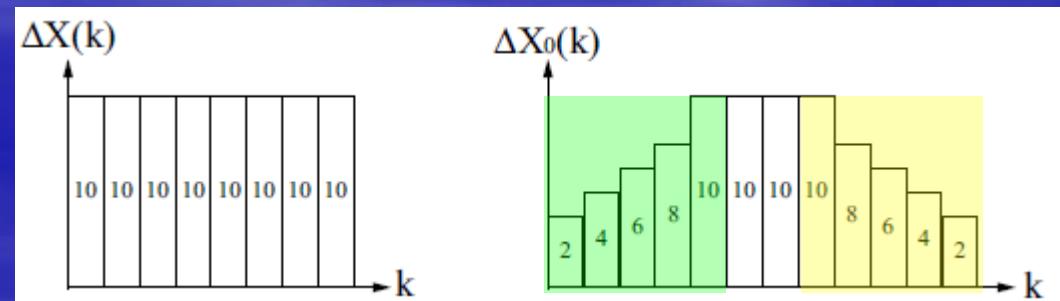
$$S(k) = S(k-1) + \Delta X(k) - \Delta X(k-n)$$

$$\Delta X_0(k) = S(k)/n$$

Acc/Dec time :

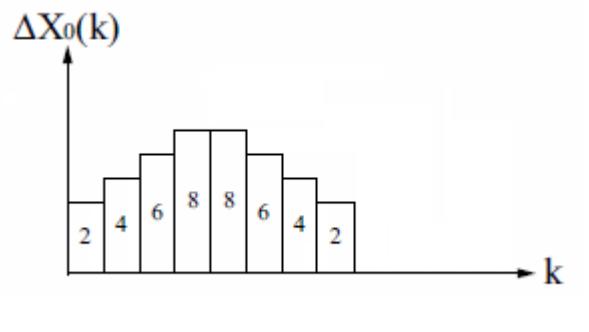
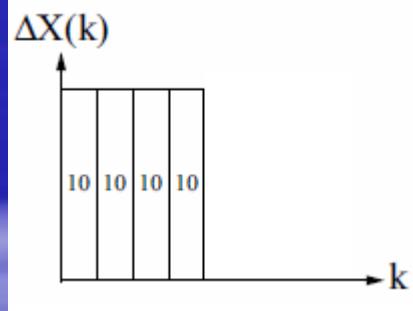
$$\tau = nT$$

Sampling time: $K$	Input pulse: $\Delta X(k)$	Output of buffer register: $\Delta X(k-5)$	Output of Adder: $S(k)$	Output Pulse: $\Delta X_0(k)$
1	10	0	10	2
2	10	0	20	4
3	10	0	30	6
4	10	0	40	8
5	10	0	50	10
6	10	10	50	10
7	10	10	50	10
8	10	10	50	10
9	0	10	40	8
10	0	10	30	6
11	0	10	20	4
12	0	10	10	2
13	0	10	0	0
$\Sigma$	80			80



# Linear-type Acc/Dec Control

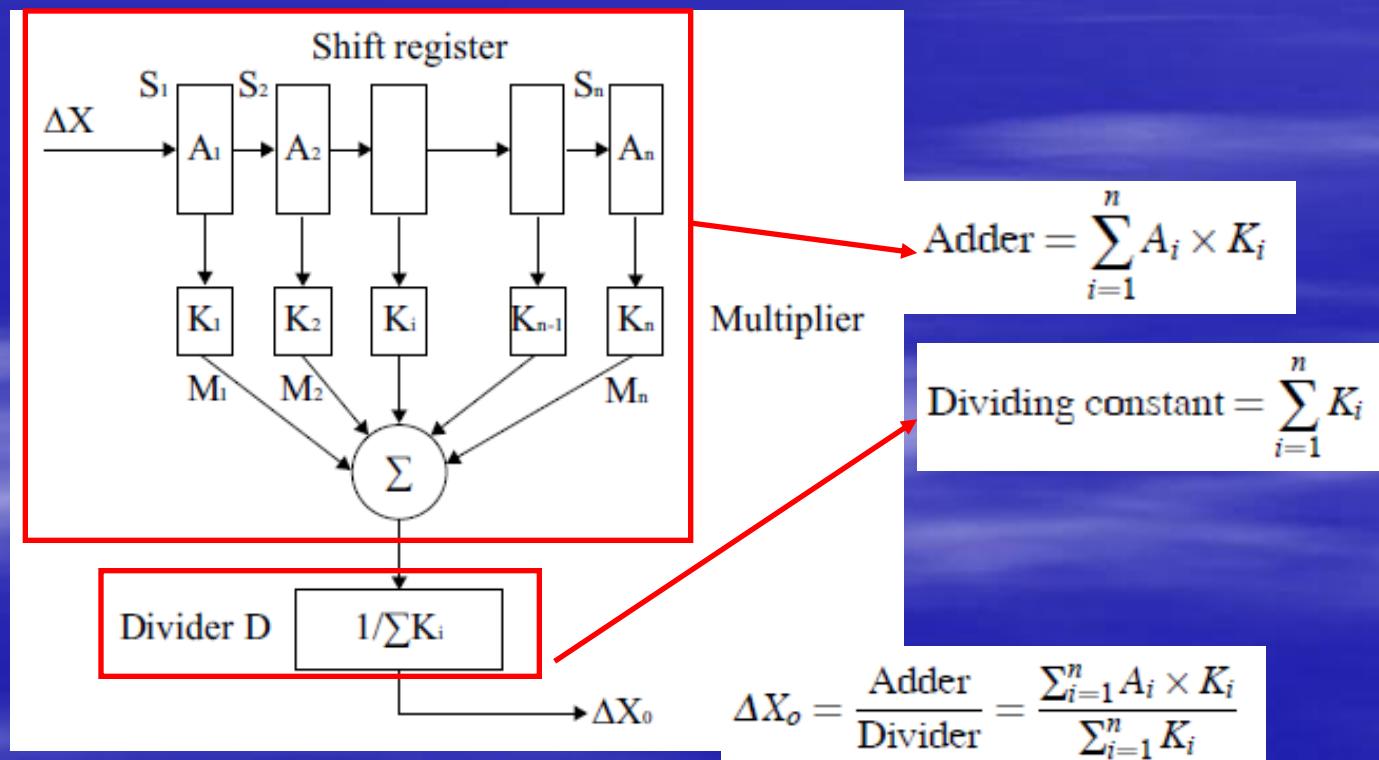
- the maximum number of output pulses is different from the number of input pulses



Sampling time: $K$	Input Pulse: $\Delta X(k)$	Output of Buffer Reg.: $\Delta X(k-5)$	Output of Adder: $S(k)$	Output Pulse: $\Delta X_o(k)$
1	10	0	10	2
2	10	0	20	4
3	10	0	30	6
4	10	0	40	8
5	0	0	40	8
6	0	10	30	6
7	0	10	20	4
8	0	10	10	2
9	0	10	0	0
$\Sigma$	40			40

# S-shape-type Acc/Dec Control

- The circuit consists of  $n$  buffer registers,  $n$  Multipliers, an Adder, and a Divider.



# S-shape-type Acc/Dec Control

- EX1 :
  - number of buffer registers is five
  - $K_1 = K_2 = K_3 = K_4 = K_5 = 1.$

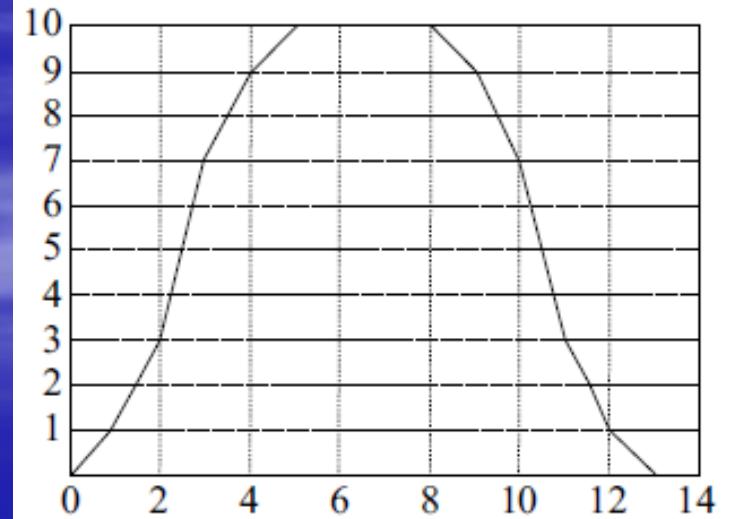
Sampling time: $K$	Input pulse: $\Delta X(k)$	Output of Adder:	Output Pulse: $\Delta X_o(k)$
1	10	10	2
2	10	20	4
3	10	30	6
4	10	40	8
5	10	50	10
6	10	50	10
7	10	50	10
8	10	50	10
9	0	40	8
10	0	30	6
11	0	20	4
12	0	10	2
$\Sigma$	80		80

# S-shape-type Acc/Dec Control

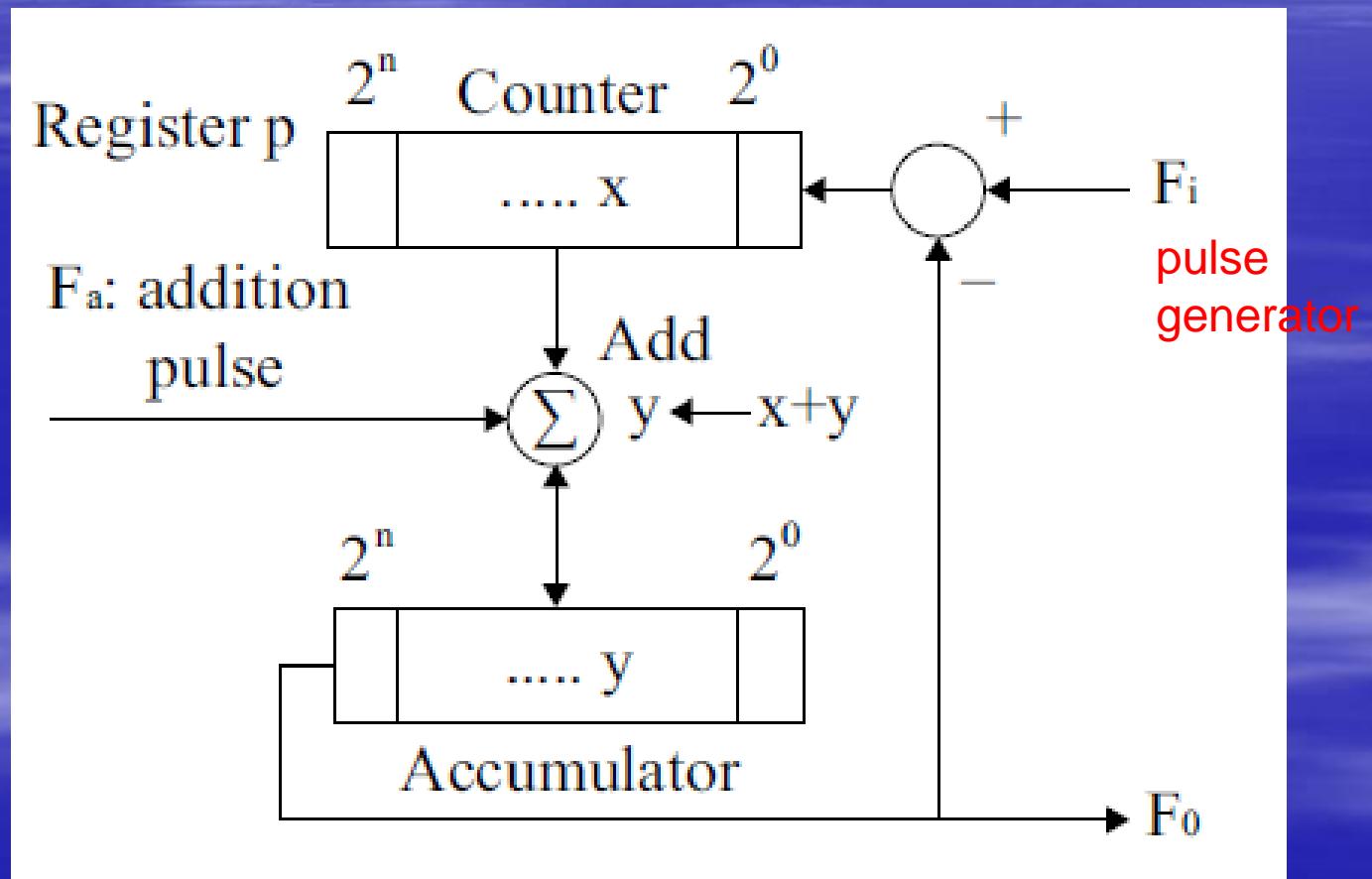
## EX2 :

- number of buffer registers is five
- $K_1 = 0.5, K_2 = 1, K_3 = 2, K_4 = 1, K_5 = 0.5$ .

Sampling time: K	Input Pulse: $\Delta X(k)$	Output of Adder:	Output Pulse: $\Delta X_o(k)$
1	10	5	1
2	10	15	3
3	10	35	7
4	10	45	9
5	10	50	10
6	10	50	10
7	10	50	10
8	10	50	10
9	0	45	9
10	0	35	7
11	0	15	3
12	0	5	1
$\Sigma$	80		80

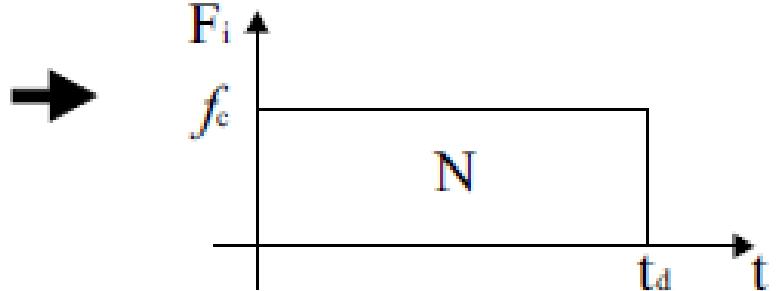
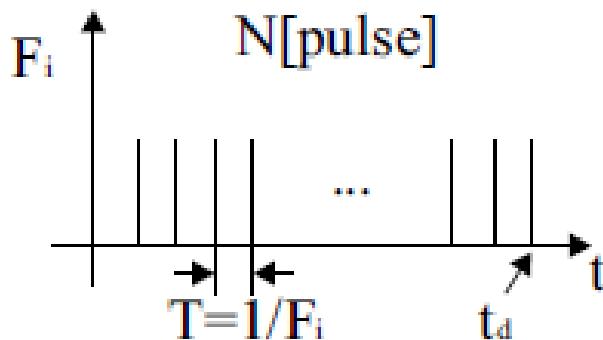


# Exponential-type Acc/Dec Control



# Exponential-type Acc/Dec Control

- Pulse generator



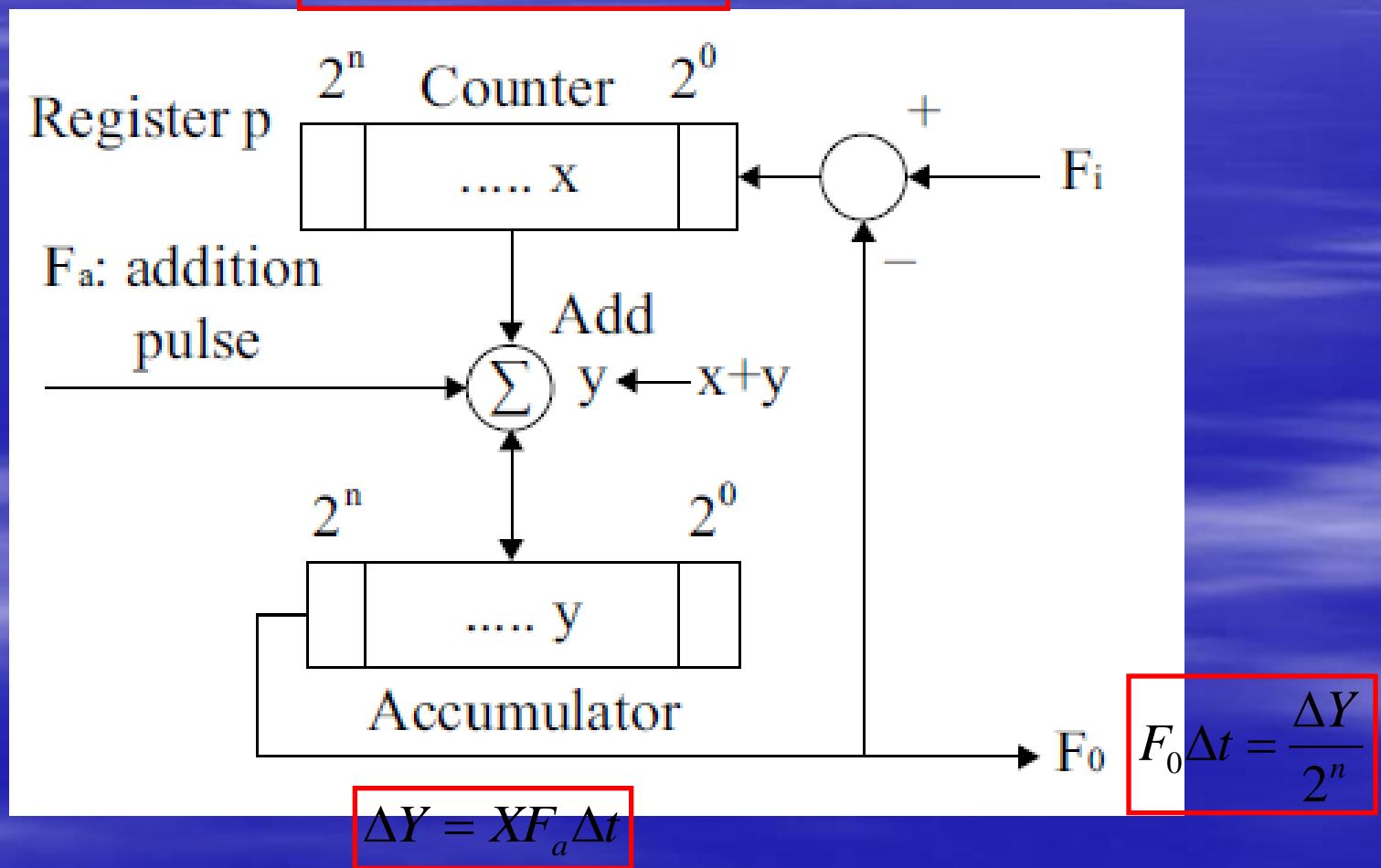
the frequency,  $F_i$ ,  
determines the speed of  
that axis

The number of generated  
pulses,  $N$ , determines the  
displacement of an axis

$$F_i = \begin{cases} f_c & (0 \leq t \leq t_d) \\ 0 & t > t_d \end{cases}$$

# Exponential-type Acc/Dec Control

$$\Delta X = (F_i - F_0)\Delta t$$



# Exponential-type Acc/Dec Control

we approximate  $\Delta t$ ,  $\Delta X$ , and  $\Delta Y$  with  $dt$ ,  $dx$ , and  $dy$

$$dx = (F_i - F_0)dt$$

$$dy = xF_a dt$$

$$F_0 dt = \frac{dy}{2^n}$$

$$X(0) = Y(0) = 0$$

# Exponential-type Acc/Dec Control

(i)  $0 \leq t \leq t_d$

ACC/Dec time;  $n \uparrow$ , time  $\uparrow$ ;  $n \downarrow$ , time  $\downarrow$

$$F_0(t) = F_i(1 - e^{-\frac{F_a}{2^n}t}) = F_i(1 - e^{-\frac{t}{X}}), X = \frac{2^n}{F_a}$$

(ii)  $t > t_d$

$$F_0(t) = F_i(1 - e^{-\frac{t_d}{X}})e^{-\frac{(t-t_d)}{X}} = F_0(t_d)e^{-\frac{(t-t_d)}{X}}$$

#pulses input

# Exponential-type Acc/Dec Control

$$P_i = \boxed{T} \times F_i$$

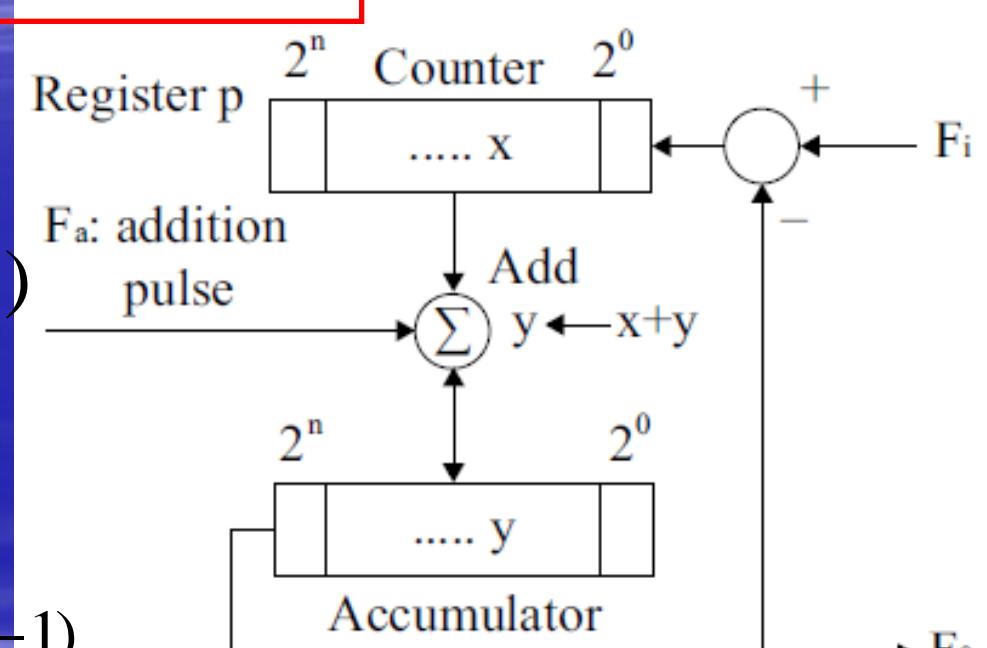
register p at  $k-1$

$$\hat{X}(k) = \boxed{x(k-1)} + P_i(k)$$

$$O(k) = \frac{y(k)}{2^n}$$

#overflow pulses

$$P_0(k) = O(k) - O(k-1)$$



$$x(k) = \hat{x}(k) - P_0(k)$$

$$y(k) = y(k-1) + \hat{x}(k)$$

# Exponential-type Acc/Dec Control

- By using the Z-transformation

$$\hat{X}(z) = \frac{1}{z} X(z) + P_i(z)$$

$$Y(z) = \frac{1}{z} Y(z) + \hat{X}(z)$$

$$O(z) = \frac{Y(z)}{2^n}$$

$$P_0(z) = O(z) - \frac{1}{z} O(z)$$

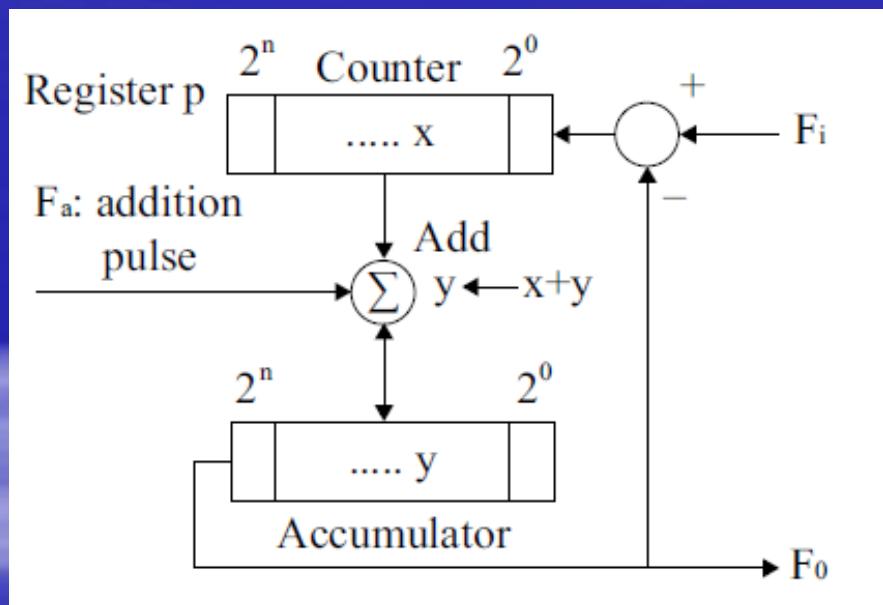
$$X(z) = \hat{X}(z) - P_0(z)$$

Time shifting

Time domain	Z-domain
$x[n-k]$	$z^{-k} x(z)$

# Exponential-type Acc/Dec Control

$$V_{EO}(k) = (1 - \alpha)(V_i(k) - V_O(k-1)) + V_O(k-1)$$



$$P_0(z) = \frac{1}{z} P_0(z) + \frac{P_i(z) - \frac{1}{2^n} P_0(z)}{z}$$

inverse Z-Transformation

$$P_0(k) = P_0(k-1) + \frac{P_i(k) - P_0(k-1)}{2^n}$$

Acc/Dec time constant

$$\tau = T \times 2^n$$

# Acc/Dec Control Machining Errors

- linear path on the XY plane, Acc/Dec control is constant, machining error does not occur.
- circular path on the XY plane, Acc/Dec control circuit is actually a sine wave or cosine wave form, machining error occur.
- the summation of input pulses and the summation of output pulses are identical
- with the distortion of the speed at the beginning and end of acceleration and deceleration, the speed ratio between the X-axis and Y-axis is changed
- a deviation between the programmed path and the path after Acc/Dec control

# Acc/Dec Control Machining Errors

- feedrate for a circular path is  $F(\text{mm/min})$ .
- the radius of the circular path is  $R(\text{mm})$ .

$$V_x(t) = -Rw \sin(wt)$$

$$V_y(t) = Rw \cos(wt)$$

$$w = \frac{F}{R}$$

By applying Laplace transformation

$$V_x(s) = -Rw \frac{w}{s^2 + w^2}$$

$$V_y(s) = Rw \frac{s}{s^2 + w^2}$$

Function	Time domain $f(t) = \mathcal{L}^{-1}\{F(s)\}$	Laplace s-domain $F(s) = \mathcal{L}\{f(t)\}$
sine	$\sin(\omega t) \cdot u(t)$	$\frac{\omega}{s^2 + \omega^2}$
cosine	$\cos(\omega t) \cdot u(t)$	$\frac{s}{s^2 + \omega^2}$

# Machining Error with Linear Type Acc/Dec Control

impulse response :  $H_l(s) = \frac{1}{\tau} \frac{1}{s} (1 - e^{-\tau s})$

where  $\tau$  represents the Acc/Dec time

Output : 
$$\begin{aligned} W_x(s) &= H_l(s)V_x(s) \\ &= -Rw \frac{1}{\tau} \frac{1}{s} \frac{1}{s^2 + w^2} [1 - e^{-\tau s}] \\ &= \frac{K}{w} \left[ \frac{1}{2} - \frac{s}{s^2 + w^2} \right] [1 - e^{-\tau s}] \end{aligned}$$

where  $K = -Rw \frac{1}{\tau}$

inverse Laplace transform : 
$$W_x(t) = \frac{2}{w\tau} \sin \frac{w\tau}{2} (-Rw) \sin w(t - \frac{\tau}{2})$$

# Machining Error with Linear Type Acc/Dec Control

radius of the circular path after applying the Linear-type Acc/Dec control:

$$r = R \frac{2}{w\tau} \sin \frac{w\tau}{2} \cos w(t - \frac{\tau}{2})$$

$$R' = \frac{2}{w\tau} \sin \frac{w\tau}{2} R \quad \text{after Acc/Dec time } (t > \tau)$$

machining error :

$$\Delta R = R - R' = R \left( 1 - \frac{2}{w\tau} \sin \frac{w\tau}{2} \right)$$

Taylor series

$$\Delta R = R \left\{ 1 - \frac{2}{w\tau} \left( \frac{w\tau}{2} - \frac{w^3\tau^3}{8 \cdot 3!} \dots \right) \right\}$$

$$\sin z = \sum_{n=0}^{\infty} (-1)^n \frac{z^{2n+1}}{(2n+1)!}$$

therefore  $\Delta R \approx \frac{R}{24} w^2 \tau^2 \approx \boxed{\frac{1}{24} \tau^2 \frac{F^2}{R}}$

# Machining Error with S-shape-type Acc/Dec Control

impulse response :  $H_s(s) = \frac{4}{\tau^2} \frac{1}{s^2} (1 - 2e^{-\frac{\tau}{2}s} + e^{-\tau s})$

where  $\tau$  represents the Acc/Dec time

Output : 
$$\begin{aligned} W_x(s) &= H_s(s)V_x(s) \\ &= -Rw \frac{4}{\tau^2} \frac{w}{s^2 + w^2} [1 - 2e^{-\frac{\tau}{2}s} + e^{-\tau s}] \\ &= \frac{K}{w} \left[ \frac{1}{s^2} \frac{1}{s^2 + w^2} \right] [1 - 2e^{-\frac{\tau}{2}s} + e^{-\tau s}] \end{aligned}$$

where  $K = -Rw \frac{4}{\tau^2}$

inverse Laplace transform :

$$W_x(t) = \frac{K}{w} \left[ \left\{ t - \frac{1}{w} \sin wt \right\} - 2 \left\{ t - \frac{\tau}{2} \sin w(t - \frac{\tau}{2}) \right\} + \left\{ t - \tau - \frac{1}{w} \sin w(t - \tau) \right\} \right]$$

# Machining Error with S-shape-type Acc/Dec Control

radius of the circular path after applying the S-shape-type Acc/Dec control:

$$r = R \frac{8}{w^2 \tau^2} \left(1 - \cos \frac{w\tau}{2}\right) \sin w(t - \frac{\tau}{2})$$

$$R' = \frac{8}{w^2 \tau^2} \left(1 - \cos \frac{w\tau}{2}\right) R \quad \text{after Acc/Dec}$$

machining error :

$$\Delta R = R - R' = R \left[1 - \frac{8}{w^2 \tau^2} \left(1 - \cos \frac{w\tau}{2}\right)\right] \quad \text{Taylor series}$$

$$\Delta R = R \left\{1 - \frac{8}{w^2 \tau^2} \left(\frac{w^2 \tau^2}{4 \cdot 2!} + \frac{w^4 \tau^4}{2^4 \cdot 4!} \dots\right)\right\} \quad \cos z = \sum_{n=0}^{\infty} (-1)^n \frac{z^{2n}}{(2n)!}$$

$$so \quad \Delta R \approx \frac{R}{48} w^2 \tau^2 \approx \frac{1}{48} \tau^2 \frac{F^2}{R}$$

# Machining Error with Exponential Type Acc/Dec Control

impulse response : 
$$H_e(s) = \frac{1}{s + \frac{1}{\tau}}$$

where  $\tau$  represents the Acc/Dec time

Output : 
$$W_x(s) = H_e(s)V_x(s)$$

$$\begin{aligned} &= -Rw \frac{1}{\tau} \frac{1}{s + a} \frac{w}{s^2 + w^2} \\ &= \frac{K}{s + a} \frac{w}{s^2 + w^2} \end{aligned}$$

where  $K = -Rw \frac{1}{\tau}$ ,  $a = \frac{1}{\tau}$

inverse Laplace transform : 
$$W_x(t) = Ae^{-at} + B \cos wt + \frac{C}{w} \sin wt$$

where  $A = \frac{wK}{w^2 + a^2}$ ,  $B = \frac{-wK}{w^2 + a^2}$ ,  $C = \frac{awK}{w^2 + a^2}$

# Machining Error with Exponential Type Acc/Dec Control

radius of the circular path after applying the Exponential-type Acc/Dec control:

$$r = -\frac{A}{a} e^{-at} + \frac{1}{w} \sqrt{B^2 + \frac{C^2}{w^2}} \sin(wt - \theta) \text{ where } \theta = \cos^{-1}\left(\sqrt{B^2 + \frac{C^2}{w^2}}\right)$$

after Acc/Dec

$$\begin{aligned} R' &= \sqrt{B^2 + \frac{C^2}{w^2}} = B \sqrt{1 + \frac{a^2}{w^2}} = \frac{-wK}{w^2 + a^2} \sqrt{1 + \frac{a^2}{w^2}} \\ &= \frac{-K}{a} \frac{1}{\sqrt{1 + \frac{w^2}{a^2}}} = -R \frac{1}{\sqrt{1 + \frac{w^2}{a^2}}} \end{aligned}$$

# Machining Error with Exponential Type Acc/Dec Control

machining error :

$$\Delta R = R - R' = R \left( 1 - \frac{1}{\sqrt{1 + \frac{w^2}{a^2}}} \right)$$

$$\Delta R = R \left[ 1 - \left\{ 1 - \frac{1}{2} \left( \frac{w}{a} \right)^2 + \frac{3}{8} \left( \frac{w}{a} \right)^4 \dots \right\} \right]$$

$$so \quad \Delta R \approx \frac{R}{2} \left( \frac{w}{a} \right)^2 \approx \boxed{\frac{1}{2} \tau^2 \frac{F^2}{R}}$$

binomial series       $\frac{1}{(1+z)^m} = \sum_{n=0}^{\infty} \binom{-m}{n} z^n$

$$= 1 - mz + \frac{m(m+1)}{2!} z^2 - \frac{m(m+1)(m+2)}{3!} z^3 + \dots$$

# Machining Error Summary

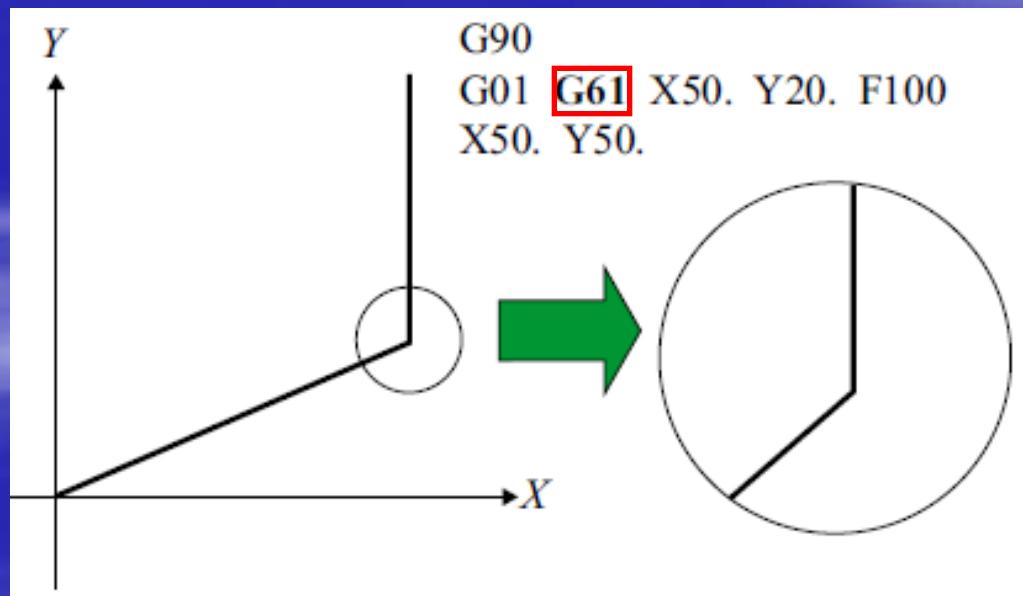
Control type	Machining Error	Remarks
Linear	$\Delta R = \frac{F^2 \tau^2}{24R}$	F: Feedrate
Exponential	$\Delta R = \frac{F^2 \tau^2}{2R}$	$\tau$ : Time constant
S-shape	$\Delta R = \frac{F^2 \tau^2}{48R}$	R: Radius of circle

# Block Overlap in ADCAI

- G-code system provides various instructions for controlling axes
- Setting the block control mode is one of the G-code functions
- EX : FANUC controller
  - exact stop mode (G61)
  - continuous mode (G64)

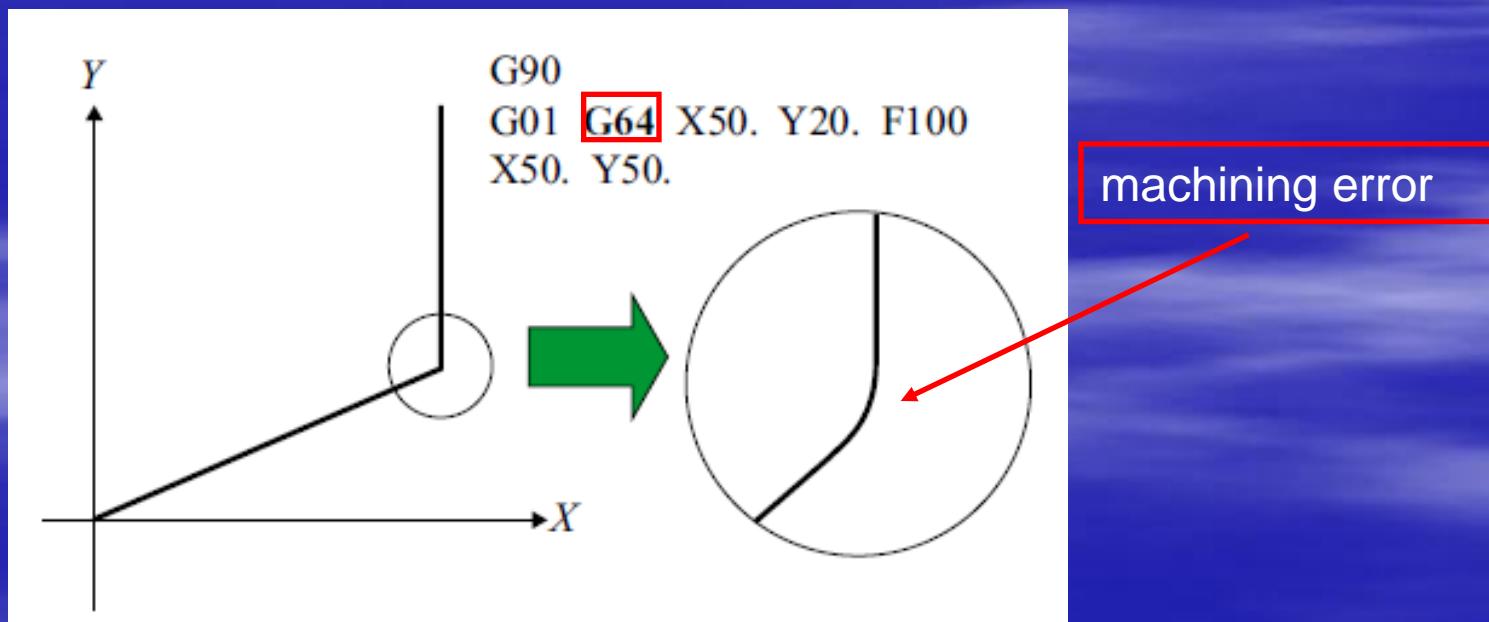
# Block Overlap in ADCAI

- Exact stop mode generally results in **reduction** of machined surface **quality** due to the stoppage of axis movement and **increases machining time** due to acceleration and deceleration for all blocks.



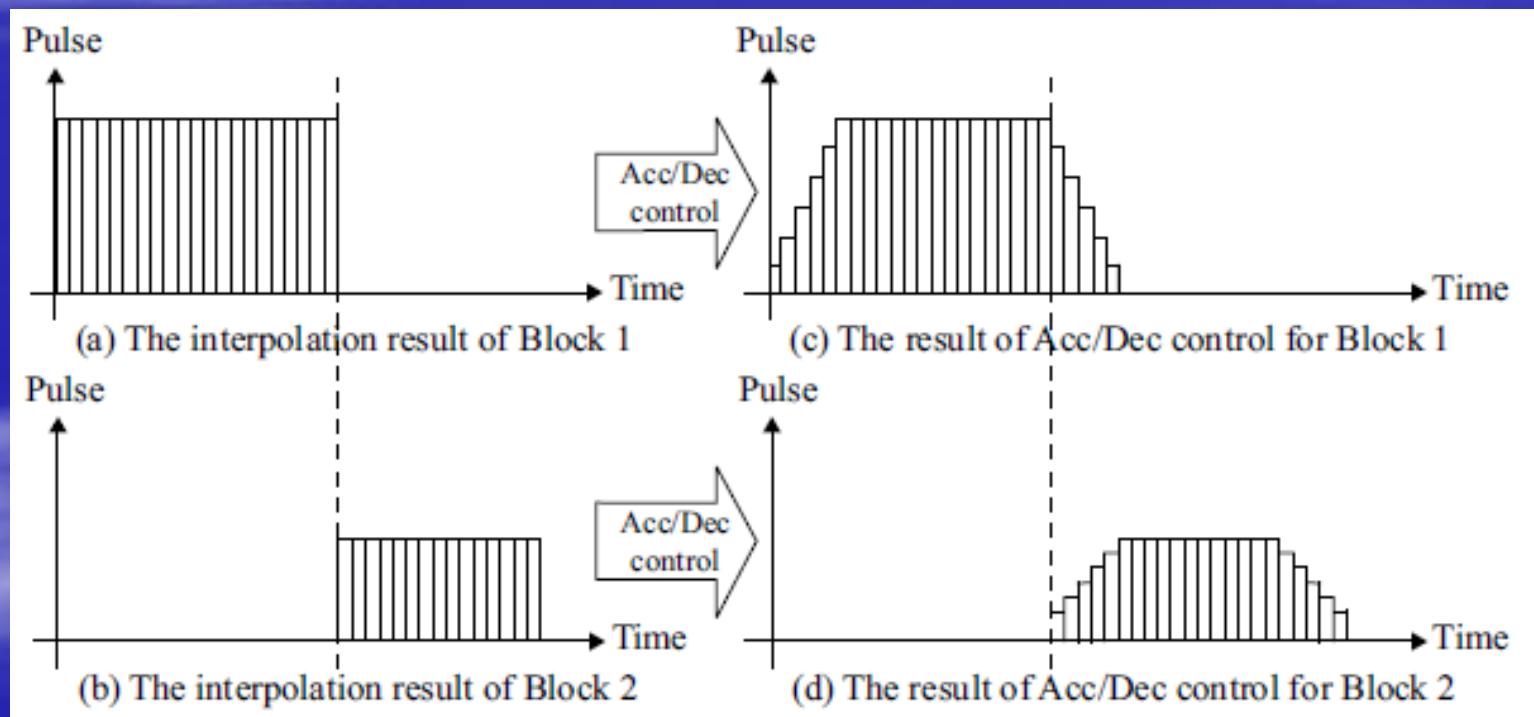
# Block Overlap in ADCAI

- In continuous mode, the tool begins the movement to the successive block before the tool reaches the end of the block. Unlike exact stop mode, this mode **does not** result in **reduction of the surface quality** and **increase in machining time**



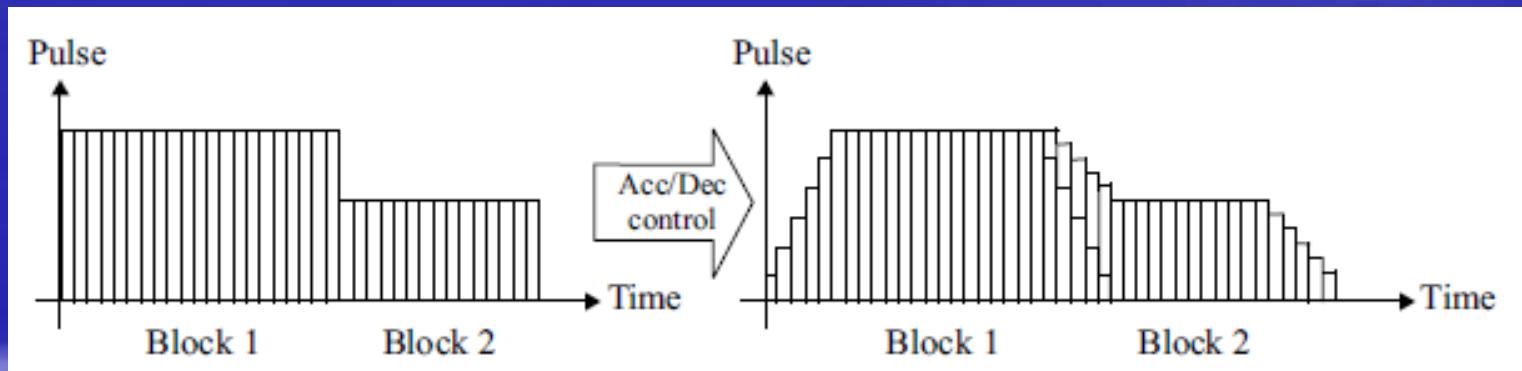
# Block Overlap in ADCAI

- X-axis interpolation and Acc/Dec control



# Block Overlap in ADCAI

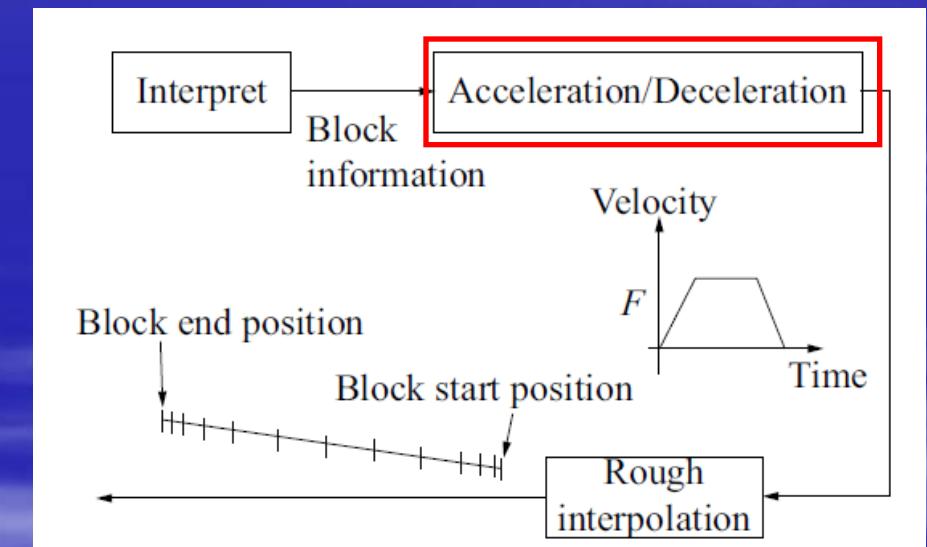
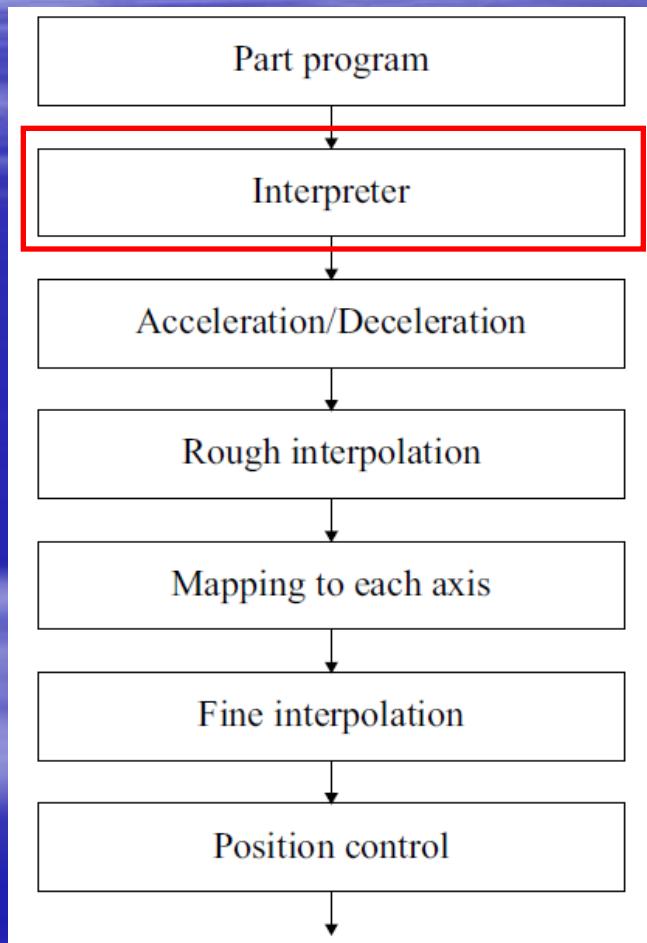
- Time-pulse graph for two successive blocks



# Acc/Dec Control Before Interpolation

- Acc/Dec control is carried out separately for individual axes, ADCBI-type NCK carries out the Acc/Dec control for the programmed path itself
- ADCBI-type NCK does not result in machining error
- latest machine tools provide ADCBI as a basic function

# Acc/Dec Control Before Interpolation



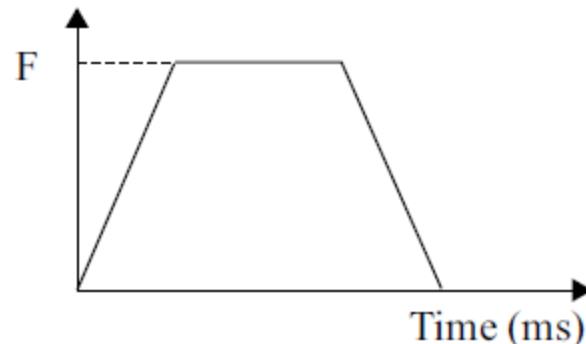
# Speed-profile Generation

- the path length
  - $L(\text{mm})$
- the allowable acceleration and deceleration
  - $A(\text{mm/s}^2)$  &  $D(\text{mm/s}^2)$
- the iteration time for rough interpolation
  - $\tau(\text{s})$
- the commanded feedrate
  - $F(\text{mm/s})$

# Speed-profile Generation

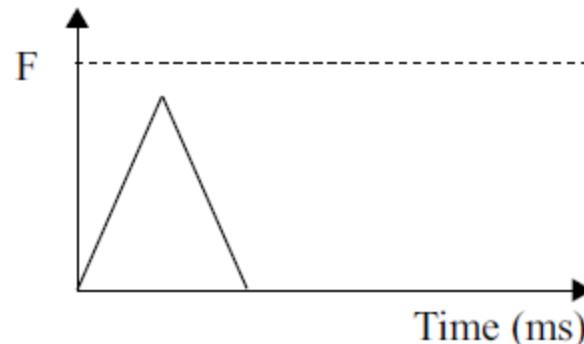
## Normal block & Short block

Velocity (mm/s<sup>2</sup>)



(a) Normal block

Velocity (mm/s<sup>2</sup>)



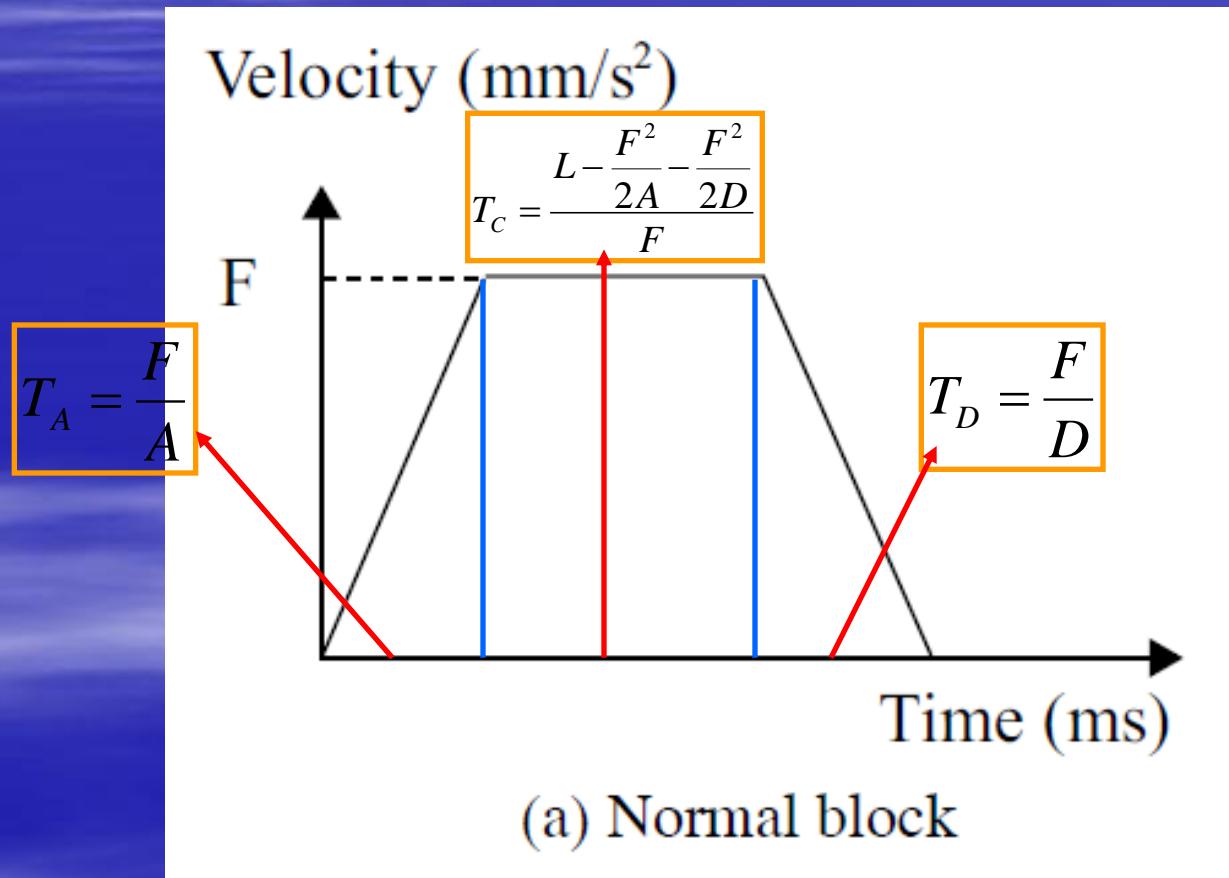
(b) Short block

$$\frac{F^2}{2A} + \frac{F^2}{2D} \geq L$$

$$\frac{F^2}{2A} + \frac{F^2}{2D} < L$$

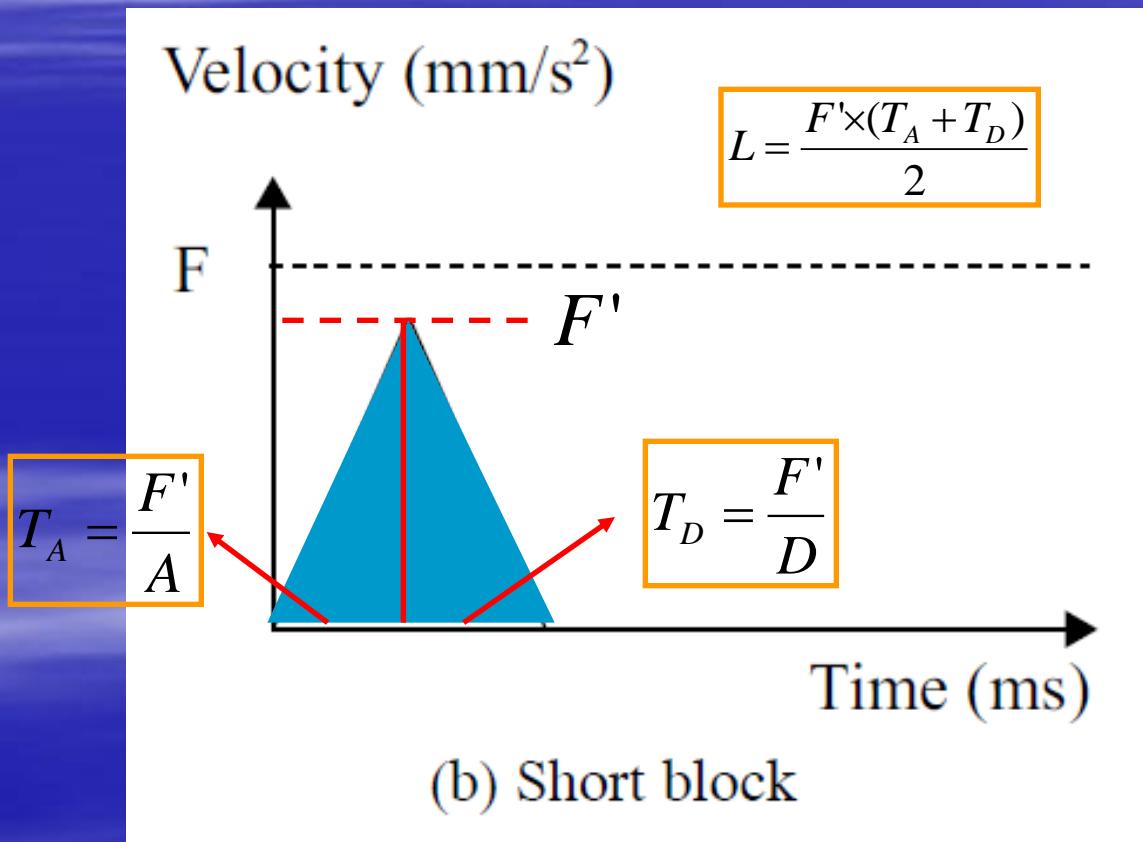
# Speed-profile Generation

## Normal block



# Speed-profile Generation

## Short block



# Speed-profile Generation

- In the acceleration range (interpolation)

$$V_{i+1} = V_i + \tau \cdot A \quad (i = 0, 1, 2, \dots, N_A)$$

$$L_i = \frac{V_{i+1}^2 - V_i^2}{2A}$$

where,  $V_i$  is the velocity of the  $i$ th interval and  $V_0 = 0$

$L_i$  is the displacement for the  $i$ th sample time.

$$N_A = \frac{T_A}{\tau}$$

- In the constant speed range (interpolation)

$$L_i = \tau \times F$$

# Speed-profile Generation

- In the deceleration range (interpolation)

$$V_{i+1} = V_i - \tau \cdot D \quad (i = 0, 1, 2, \dots, N_A)$$

$$L_i = \frac{V_{i+1}^2 - V_i^2}{2D}$$

where,  $V_i$  is the velocity of the  $i$ th interval and  $V_0 = F$

$L_i$  is the displacement for the  $i$ th sample time.

$$N_D = \frac{T_D}{\tau}$$

# Block Overlap Control

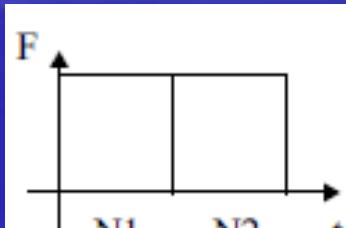
- Multiple linear blocks and circular blocks.
- The successive blocks should be considered.
- All possible cases for connection relationships will be addressed.

# Classification of Continuous Blocks

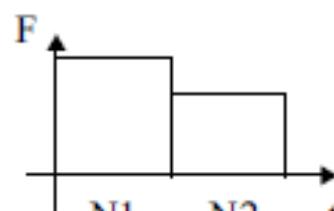
- Two successive blocks
  - Normal block & short block
  - Type of block & difference of commanded feedrate
- Twelve types
- The direction of two successive blocks is identical

# Classification of Continuous Blocks

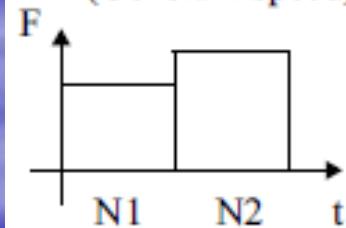
- Classified into eight types



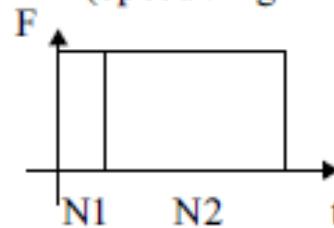
(a) Normal block  
→ Normal block  
(Constant speed)



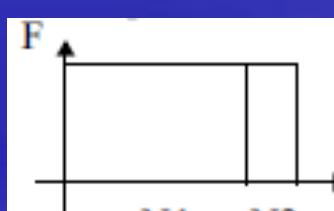
(b) Normal block  
→ Normal block  
(Speed : high → low)



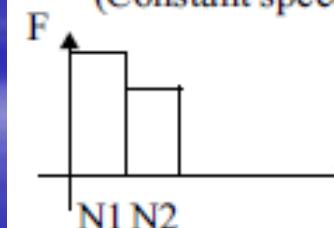
(c) Normal block  
→ Normal block  
(Speed : low → high)



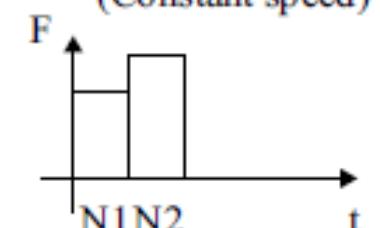
(d) Short block  
→ Normal block  
(Constant speed)



(e) Normal block  
→ Short block  
(Constant speed)



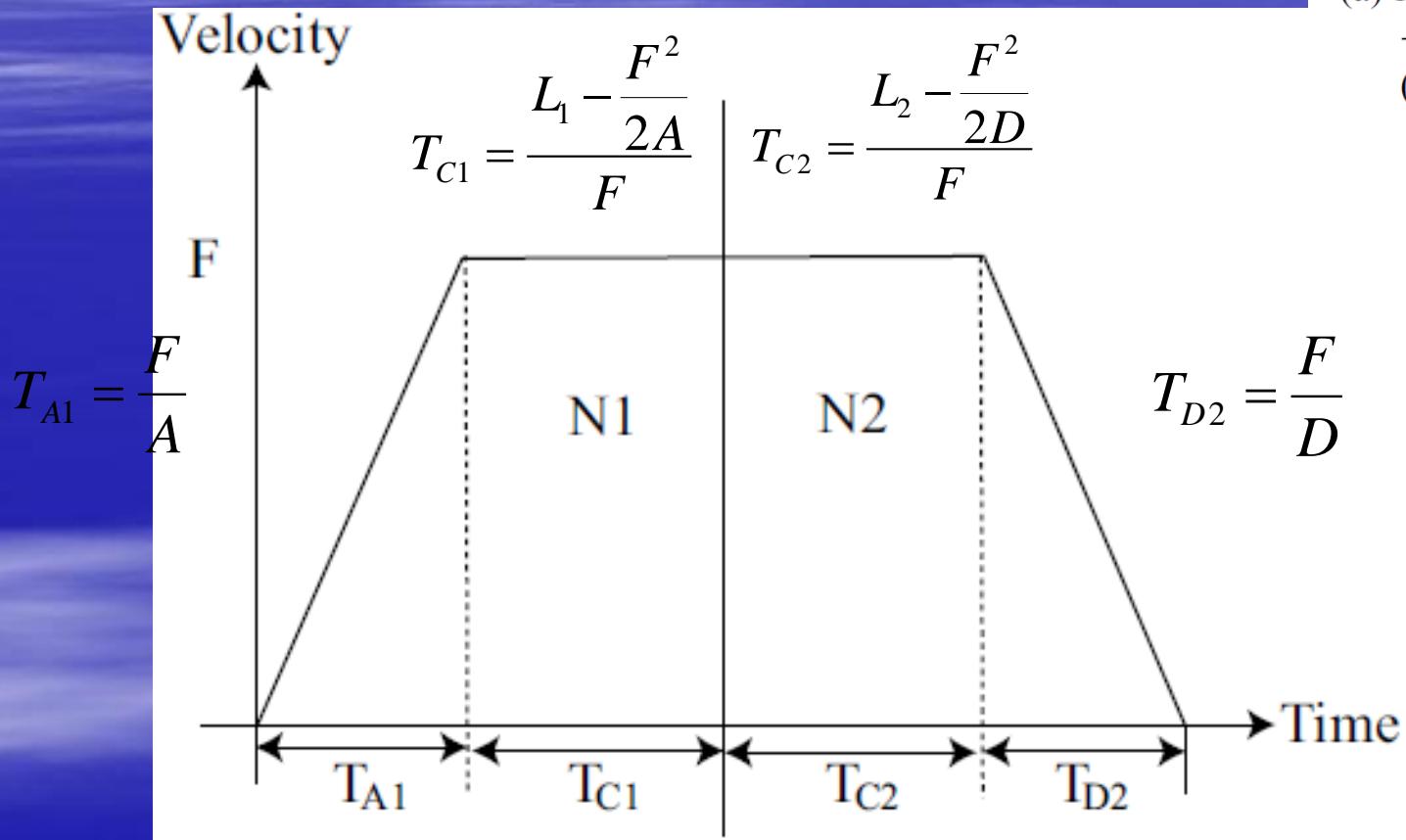
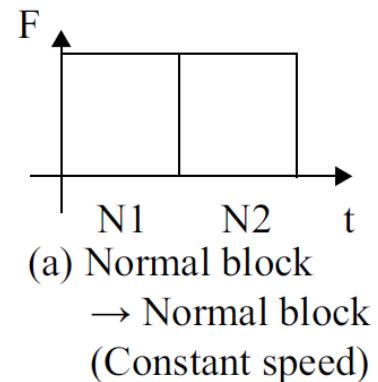
(f) Short block  
→ Short block  
(Constant speed)



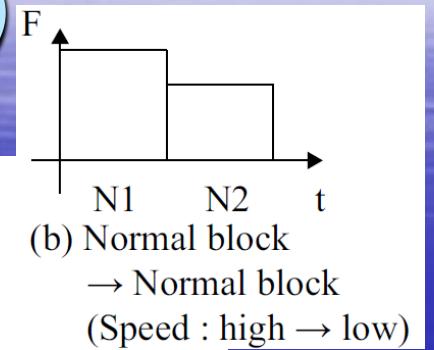
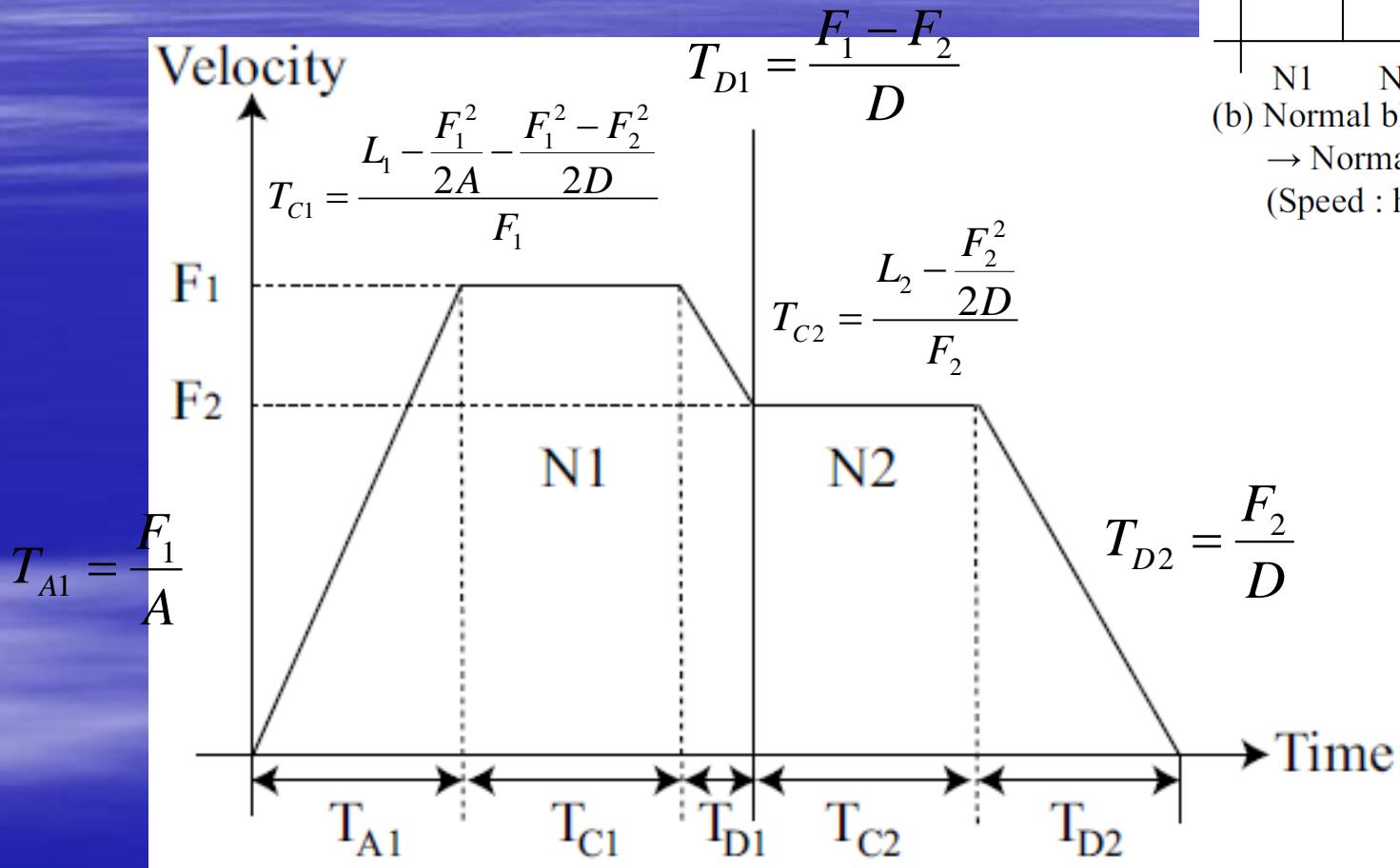
(g) Short block  
→ Short block  
(Speed : high → low)

(h) Short block  
→ Short block  
(Speed : low → high)

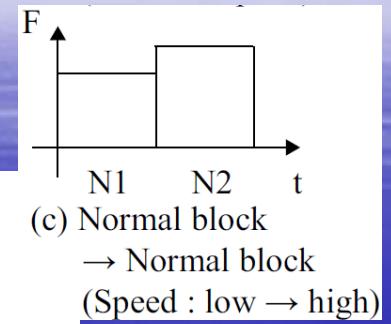
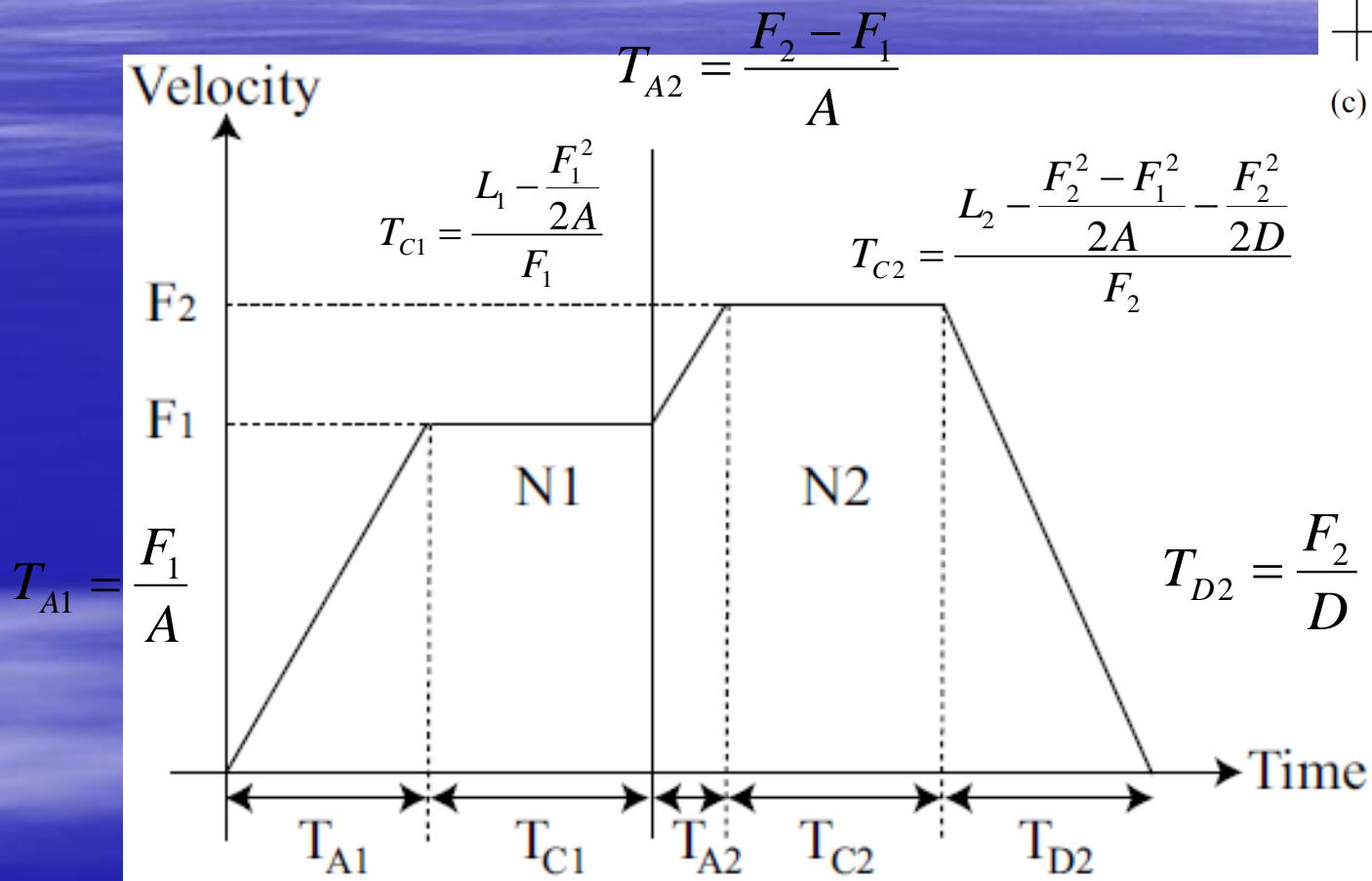
# Normal Block/Normal Block Identical Speed



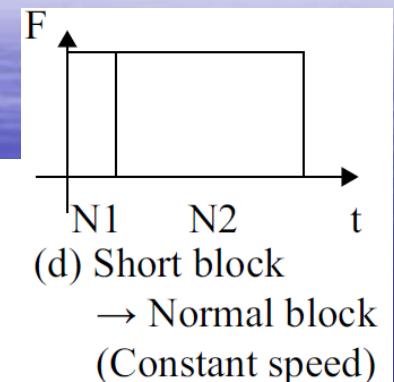
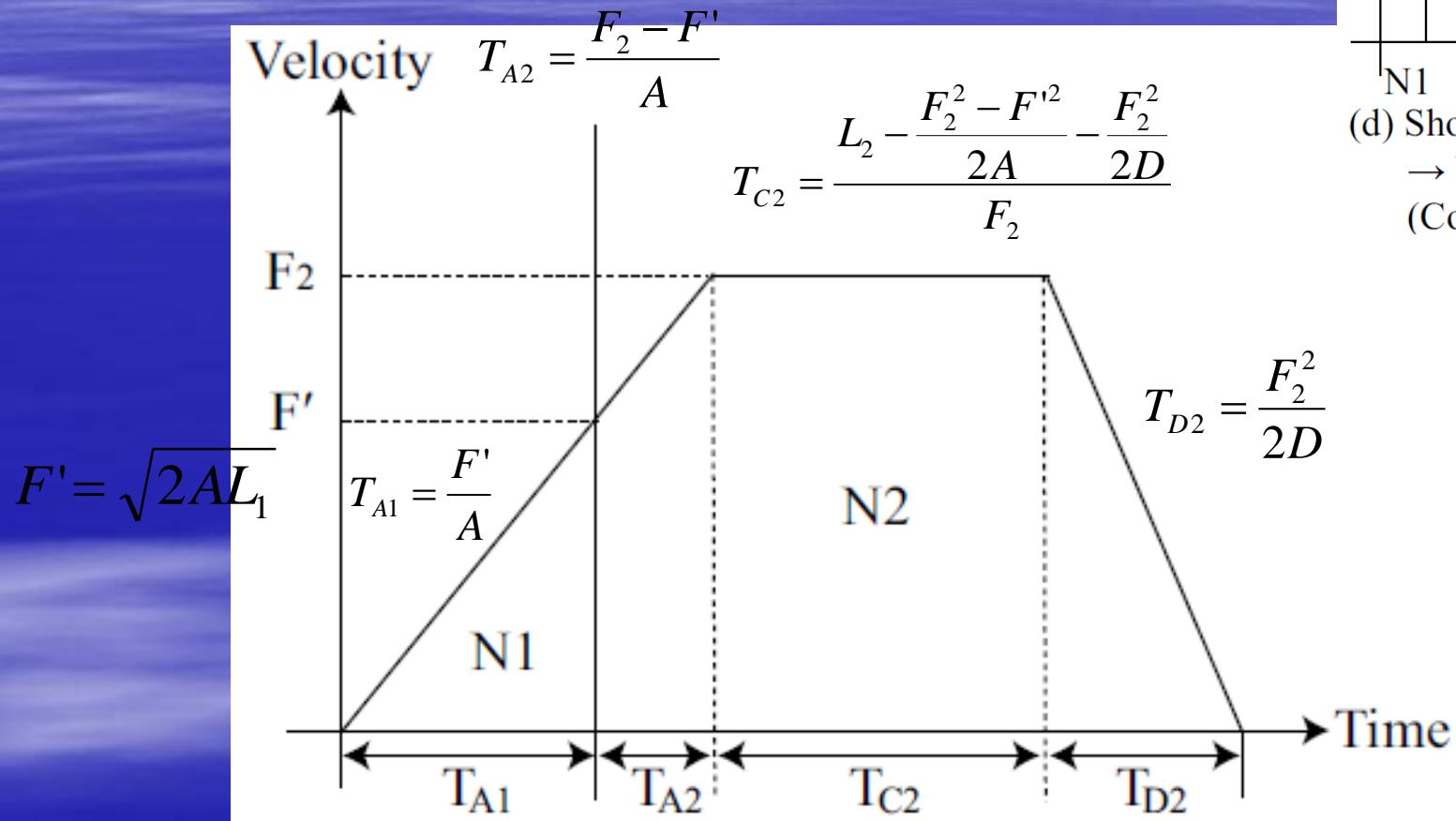
# Normal Block (High Speed)/Normal Block (Low Speed)



# Normal Block (Low Speed)/Normal Block (High Speed)

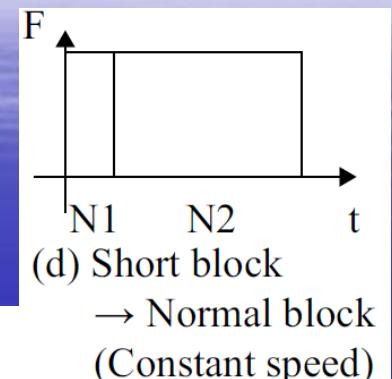
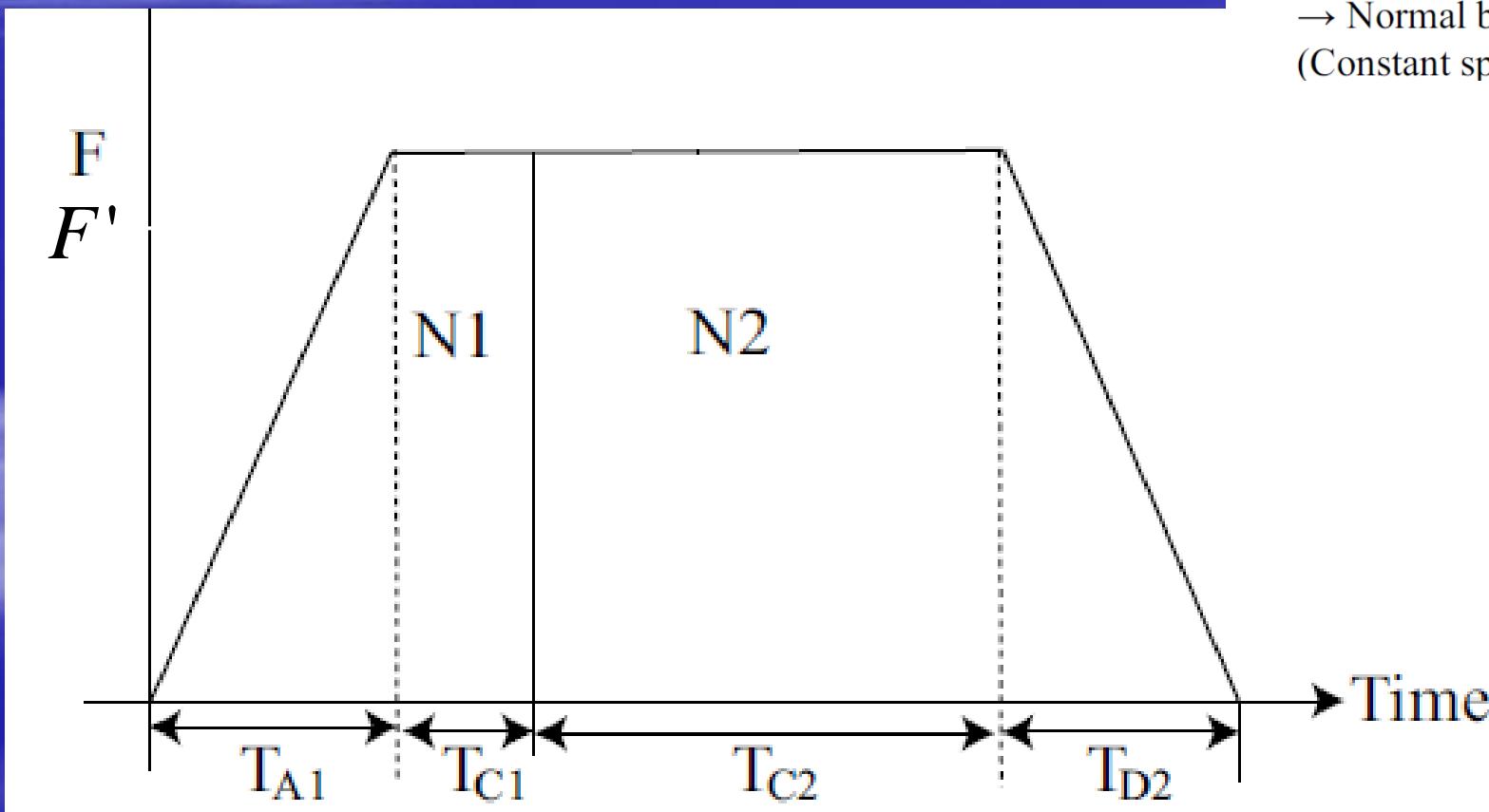


# Short Block/Normal Block with Identical Speed



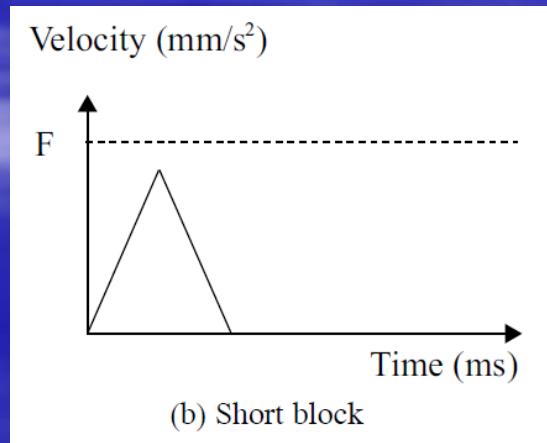
# Short Block/Normal Block with Identical Speed

- 這種情況也可能？

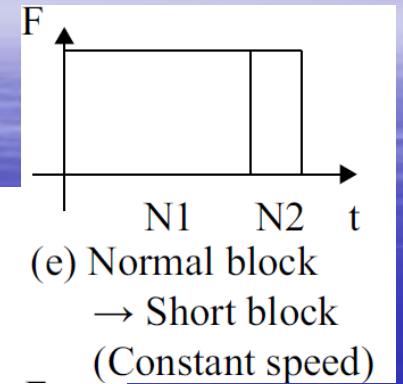
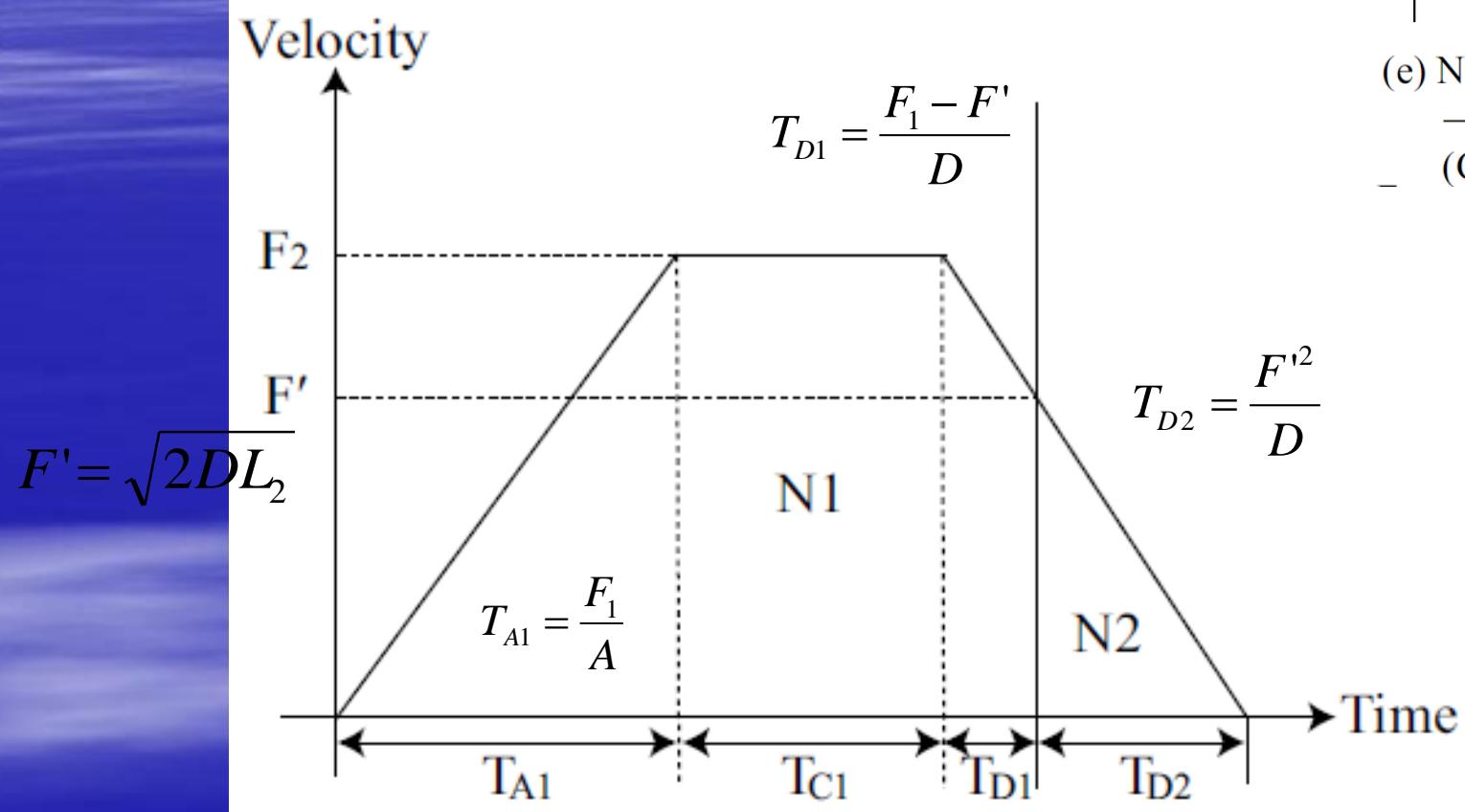


# Short Block/Normal Block with Different Speed

- same method as Identical Speed
- Ps:似乎認定在short block的情況，把所有路程都加速也無法達到命令的速度，但實際上卻可能出現加速過頭的情況。



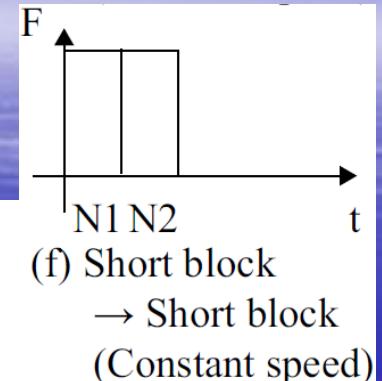
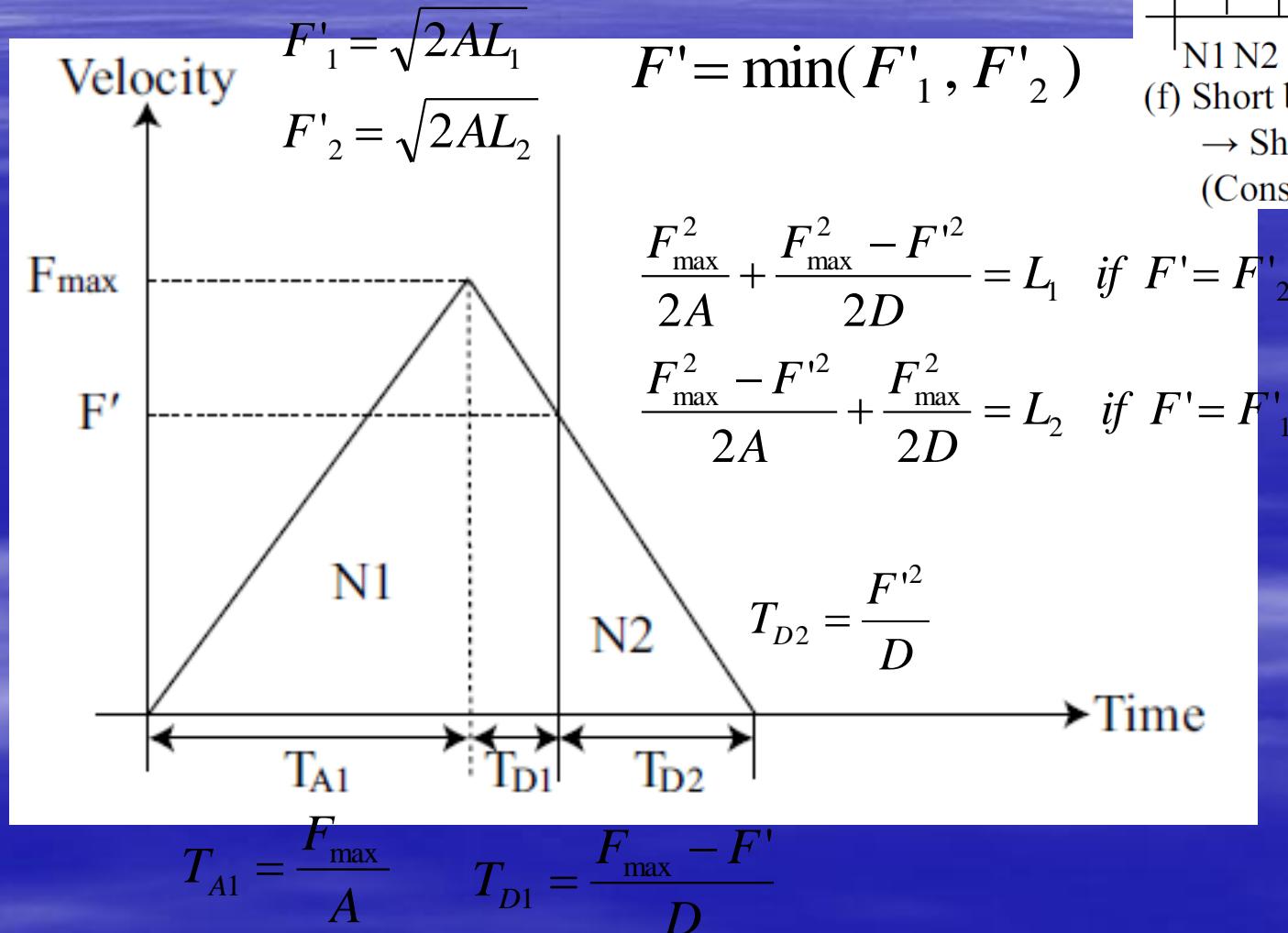
# Normal Block/Short Block with Identical Speed



# Normal Block/Short Block with Different Speed

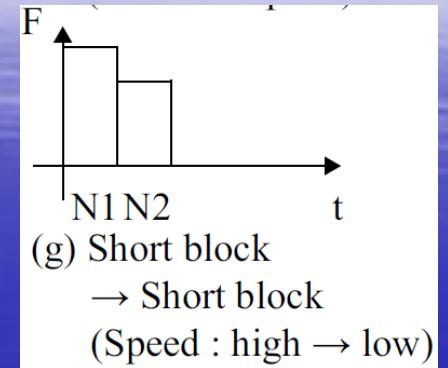
- same method as Identical Speed
- Orz.....

# Short Block/Short Block with Identical Speed



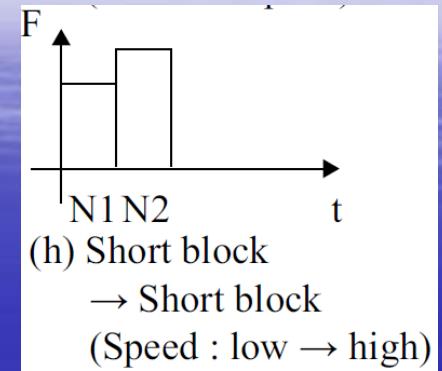
# Short Block (High Speed)/Short Block (Low Speed)

- the speed profile can be identically obtained by the method of Short Block/Short Block with Identical Speed.



# Short Block (Low Speed)/Short Block (High Speed)

- the speed profile can be identically obtained by the method of Short Block/Short Block with Identical Speed.



# Overlap Between a Linear and a Circular Profile

- circular-path
  - The change of the axis speed results in mechanical shock
  - The mechanical shock is proportional to the acceleration.
  - necessary to restrict the maximum allowable acceleration for a circular path

# Overlap Between a Linear and a Circular Profile

- the speed & acceleration of each axis

$$V_x = F \cos \omega t \quad V_y = F \sin \omega t$$

where,  $\omega = \frac{F}{R}$

$$A_x = -F\omega \sin \omega t \quad A_y = F\omega \cos \omega t$$

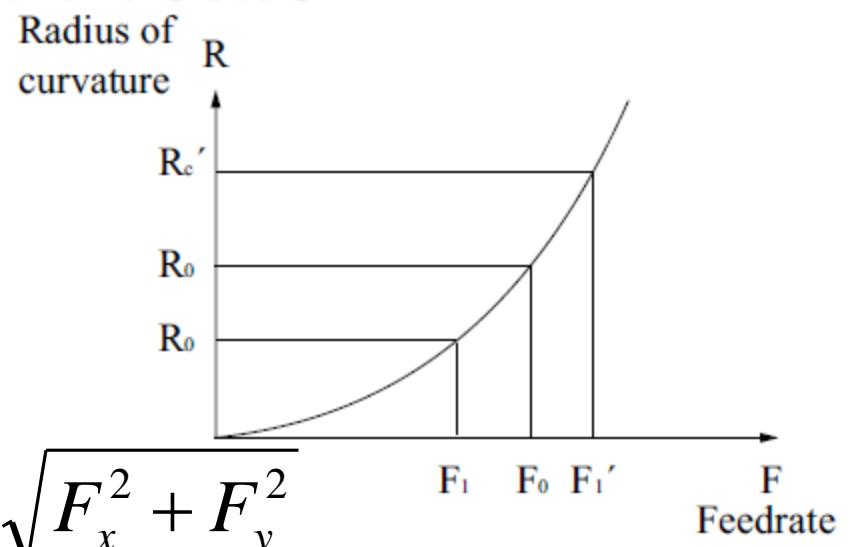
# Overlap Between a Linear and a Circular Profile

- radius :  $R_o$

$$F_x = \sqrt{A_x R_o}$$

$$F_y = \sqrt{A_y R_o}$$

$$F_o = \sqrt{F_x^2 + F_y^2}$$



- If radius :  $R_c$

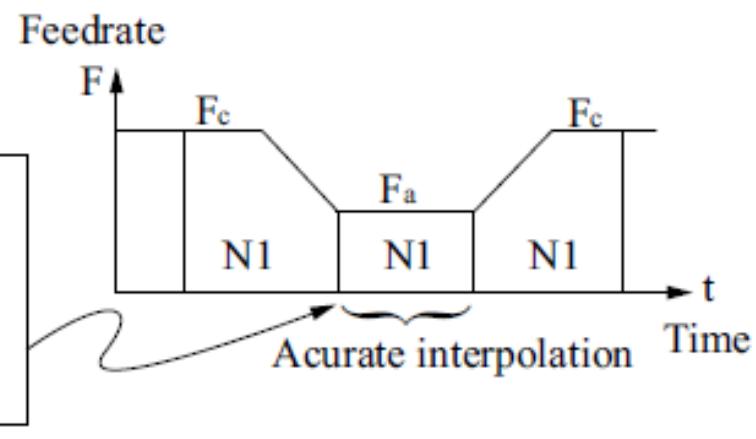
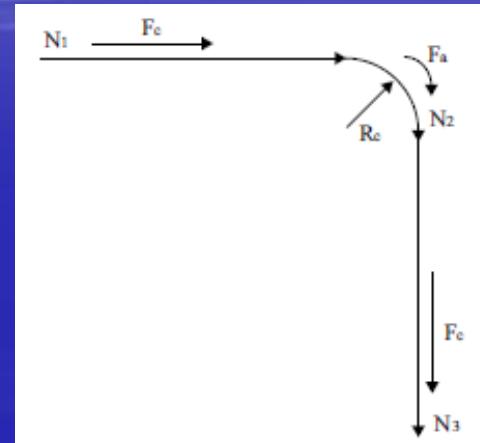
$$\frac{F_{\perp}^2}{R_c} = \frac{F_o^2}{R_o},$$

$$F_{\perp} = F_o \sqrt{\frac{R_c}{R_o}}$$

# Overlap Between a Linear and a Circular Profile

EX :

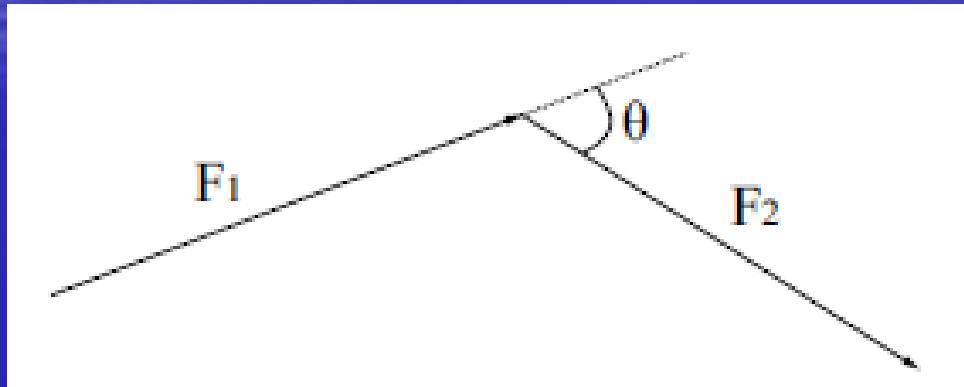
N1 G91 G01 X100. F10000;  
N2 G02 X50. Y-50. R50;  
N3 G01 Y-100;  
N4 M06



The actual feedrate is set to an allowable feedrate if the commanded feedrate of the circular path is larger than the allowable feedrate.

# Corner Speed of Two Blocks Connected by an Acute Angle

- two successive blocks with different directions



- acceleration at the corner

$$A_C = \frac{F_1 - F_2 \cos \theta}{T_{pos}}$$

where,  $T_{pos}$  is the sampling time for position control

# Corner Speed of Two Blocks Connected by an Acute Angle

- If  $A_c > A_{max}$ , a mechanical shock or vibration can occur.

$$F_C = \frac{AT_{pos}}{1 - \cos \theta}$$

where,     $A$  is the maximum allowable acceleration

- commanded feedrate, the length of blocks, the allowable acceleration and the corner speed , a speed profile can be generated.

# Corner Speed Considering Speed Difference of Each Axis

- another method for deciding the corner speed
- individual servo motors & allowable acceleration value for each axis

$$V_{X1} = F_1 \cdot \frac{X_{E1} - X_{S1}}{L_1}, V_{Y1} = F_1 \cdot \frac{Y_{E1} - Y_{S1}}{L_1}, V_{Z1} = F_1 \cdot \frac{Z_{E1} - Z_{S1}}{L_1},$$

$$V_{X2} = F_2 \cdot \frac{X_{E2} - X_{S2}}{L_2}, V_{Y2} = F_2 \cdot \frac{Y_{E2} - Y_{S2}}{L_2}, V_{Z2} = F_2 \cdot \frac{Z_{E2} - Z_{S2}}{L_2},$$

where  $V_{Ai}$  is the A – axis component of velocity of block  $N_i$

$L_i$  is the length of block  $N_i$

# Corner Speed Considering Speed Difference of Each Axis

- The difference in speed along the directions of each axis

$$\Delta V_x = (V_{x2} - V_{x1}), \Delta V_y = (V_{y2} - V_{y1}), \Delta V_z = (V_{z2} - V_{z1})$$

- the smallest of the speed change ratios ( $Q$ )

$$Q = \min \left\{ \frac{\Delta V_{mx}}{\Delta V_x}, \frac{\Delta V_{my}}{\Delta V_y}, \frac{\Delta V_{mz}}{\Delta V_z} \right\}$$

the maximum allowable change of speed along each axis  
as  $\Delta V_{mx}$ ,  $\Delta V_{my}$ ,  $\Delta V_{mz}$

# Corner Speed Considering Speed Difference of Each Axis

- If  $Q < 1$

$$F_{E1} = Q \cdot F_1, F_{S2} = Q \cdot F_2$$

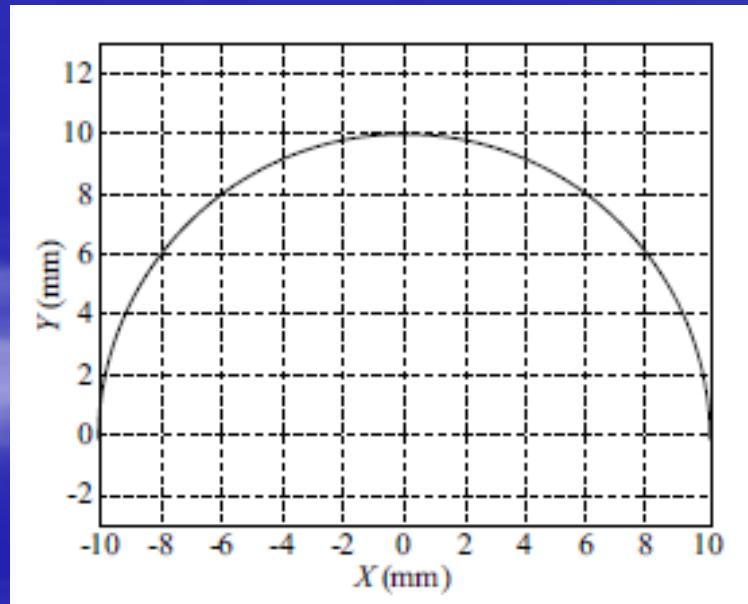
- Although discontinuity of speed occurs, this does not result in any problem because the speed change is enough small for a servo motor to follow the changed speed.

# Look Ahead

- Speed & accuracy
- In the ADCBI type of NCK, the accuracy of machining is very high (theoretically the error is zero) and sudden change of feedrate is a major factor of machining error.
- necessary to smooth down change of feedrate and limit the axis speed to an allowable value.
- two short blocks are connected, the length of two blocks is too short to reach the commanded feedrate and the resulting speed profile shows a special shape similar to a saw tooth

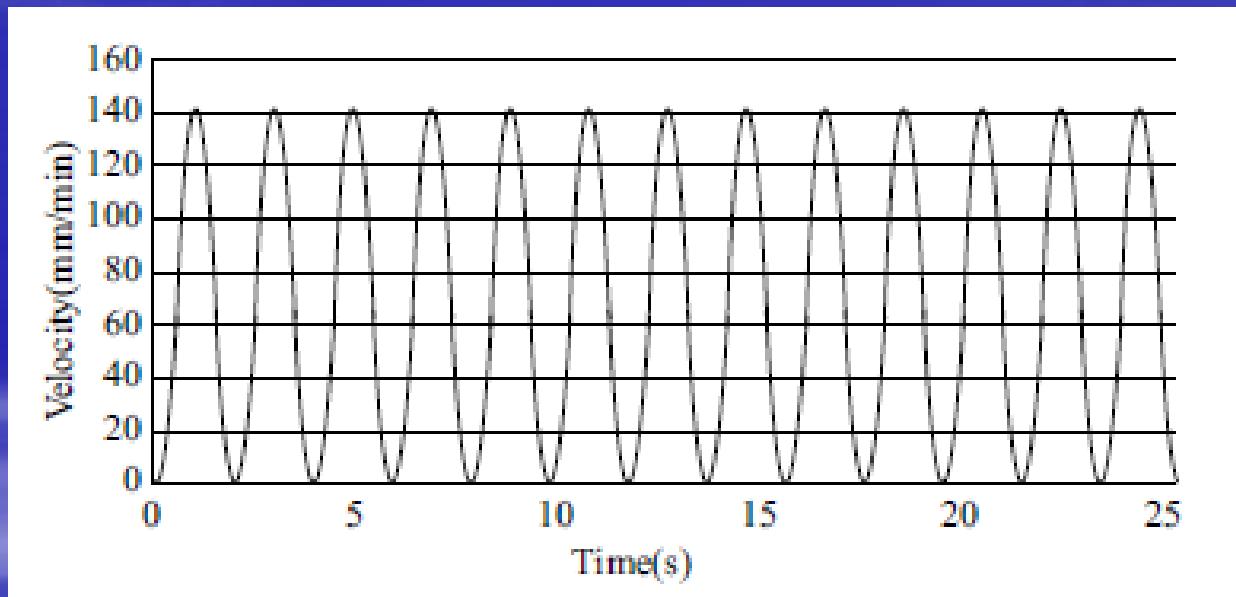
# Look Ahead

- EX :
  - 15 line segments, the radius of the half-circle is 10 mm, the commanded feedrate is 400 mm/min, and the allowable acceleration is 9600 mm/min.



# Look Ahead

- The maximum reachable feedrate is 141.78mm/min and acceleration and deceleration were repeated.



Speed profile for circular profile

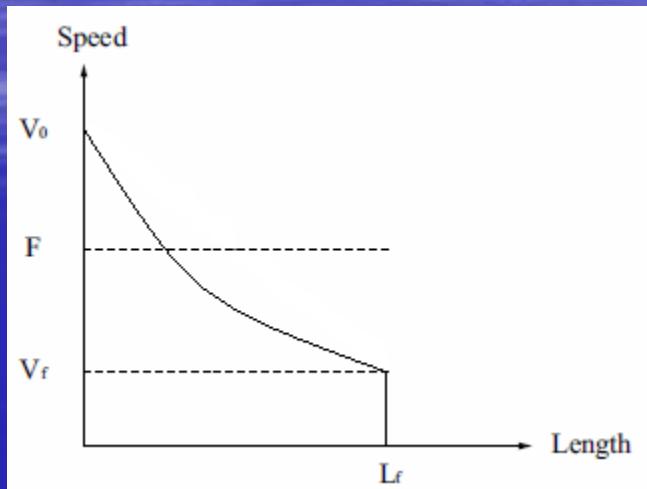
# Look Ahead

- The Look Ahead algorithm enables minimization of the decrease of feedrate by calculating the maximum allowable feedrate and the end feedrate for a current block investigating not only the current block but also successive blocks.
- The latest FANUC controller is able to calculate the end speed of a current block by pre-interpreting about 1000 blocks

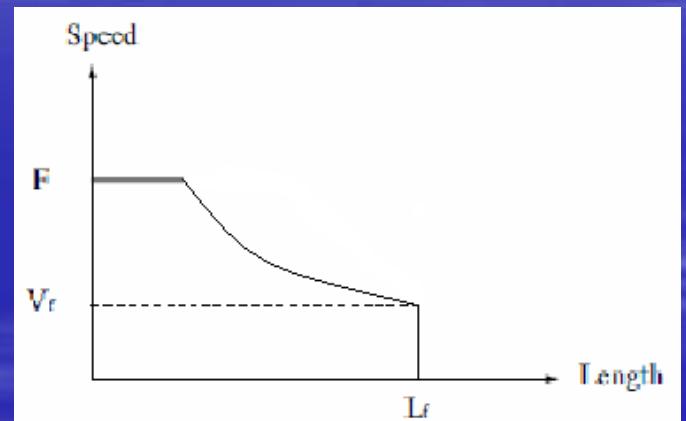
# Look Ahead Algorithm

- A Look Ahead algorithm calculates the start speed and the end speed of each block based on the remaining length of the successive blocks and the maximum allowable acceleration.

# Look Ahead with Respect to Length



After Look  
Ahead



$$V_0 = \sqrt{V_f^2 + 2 \cdot A \cdot L}$$

$$V_f = \begin{cases} V, & V < F \\ F, & V > F \end{cases}$$

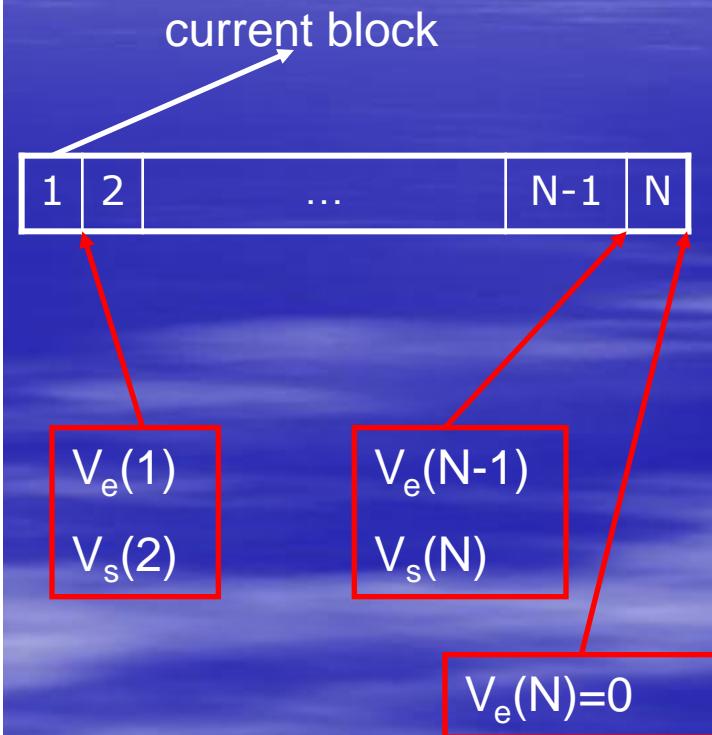
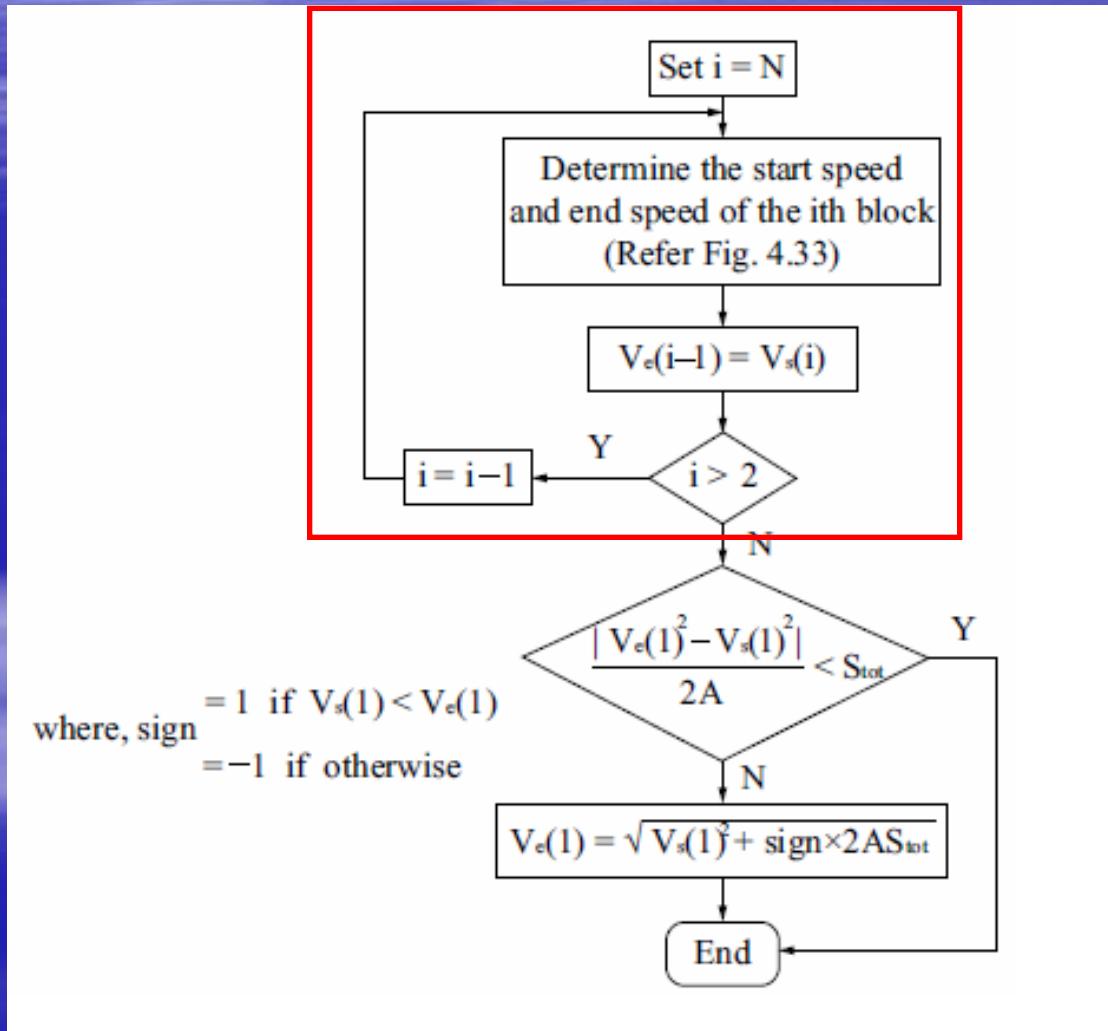
## Look Ahead with Respect to Length

- it is assumed that the end speed of the last block among the look-ahead blocks is zero.

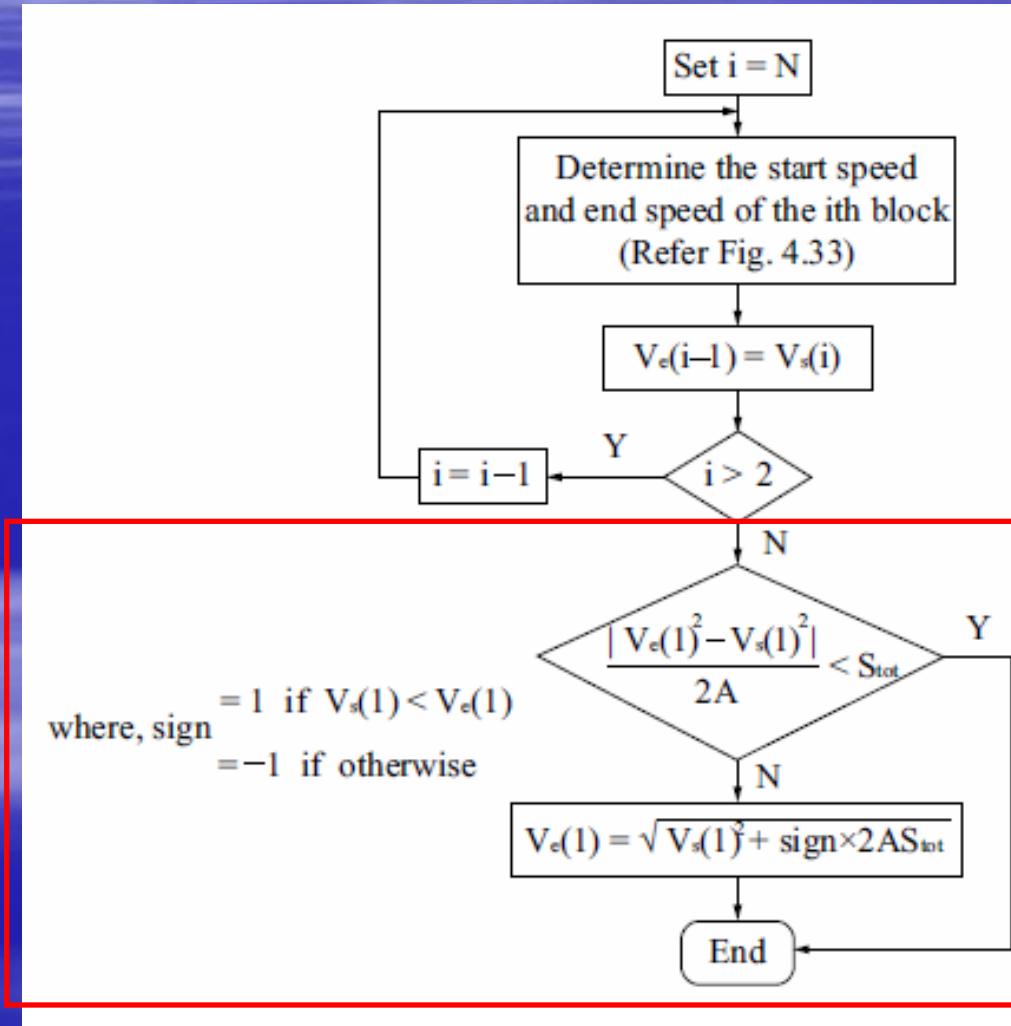
# Speed at a Corner

- Two methods :
  - Corner Speed of Two Blocks Connected by an Acute Angle
  - Corner Speed Considering Speed Difference of Each Axis

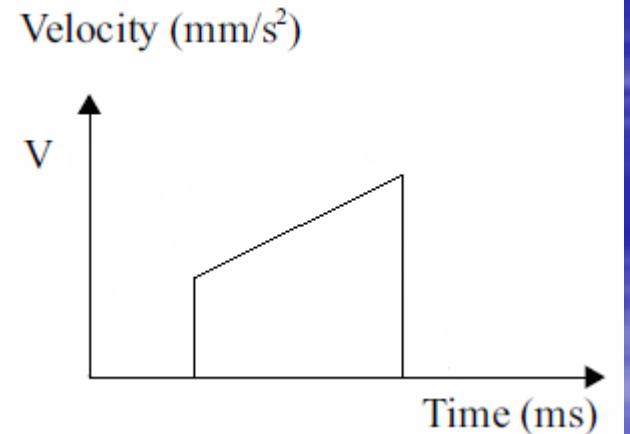
# Look Ahead considering Length and Corner



# Look Ahead considering Length and Corner



$$\frac{|V_e(1)^2 - V_s(1)^2|}{2A} < S_{tot}$$

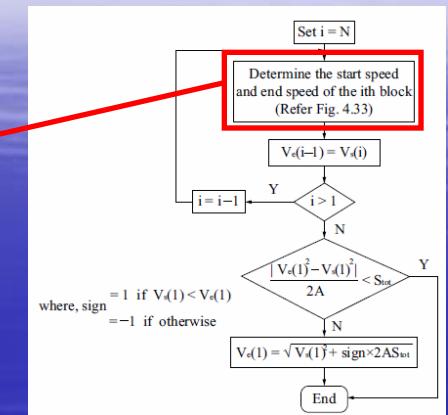
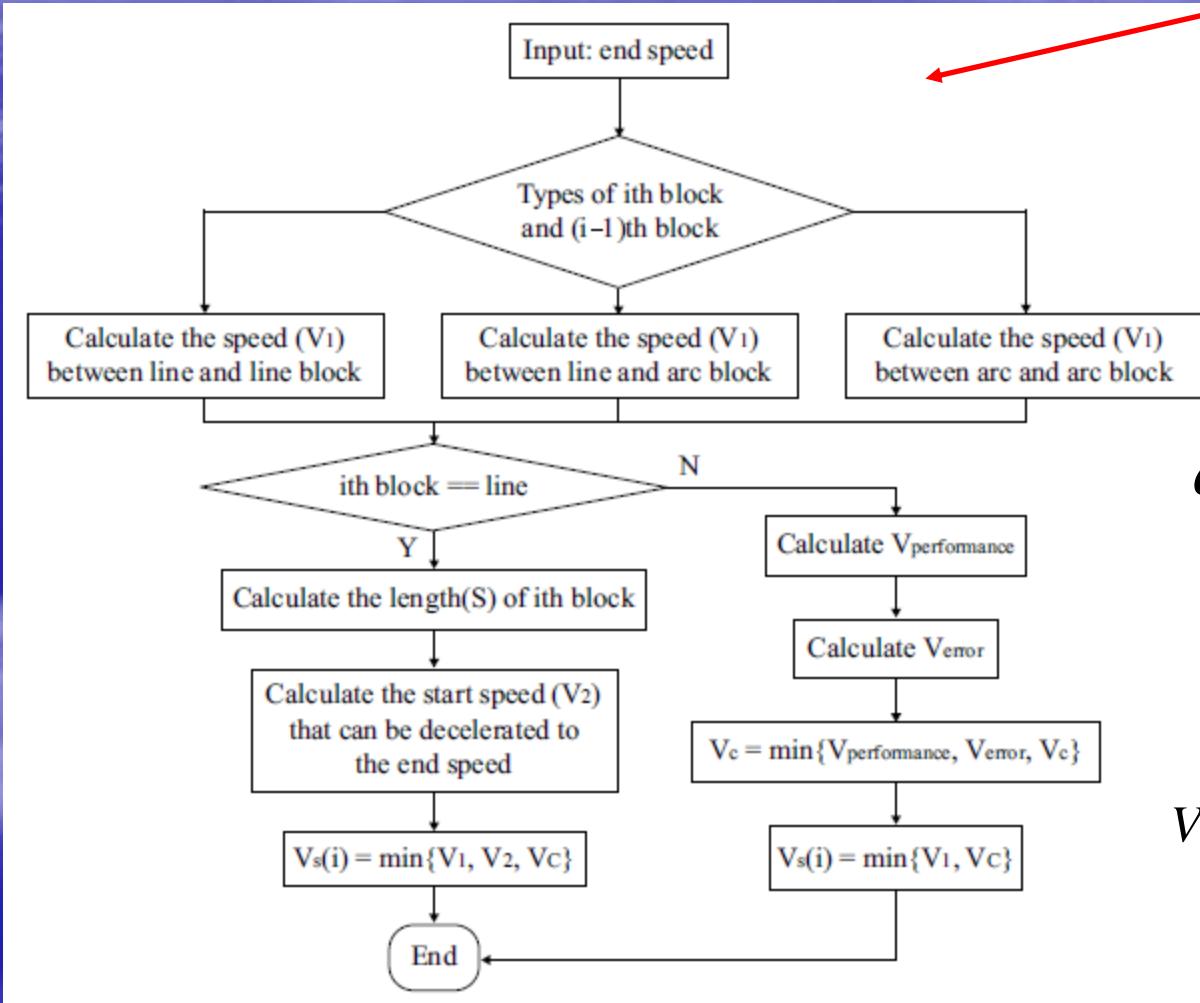


$$V_e(1) = \sqrt{V_s(1)^2 + sign \times 2AS_{tot}}$$

where,

$sign$	$= 1$ if $V_s(1) < V_e(1)$
	$= -1$ otherwise

# Look Ahead considering Length and Corner



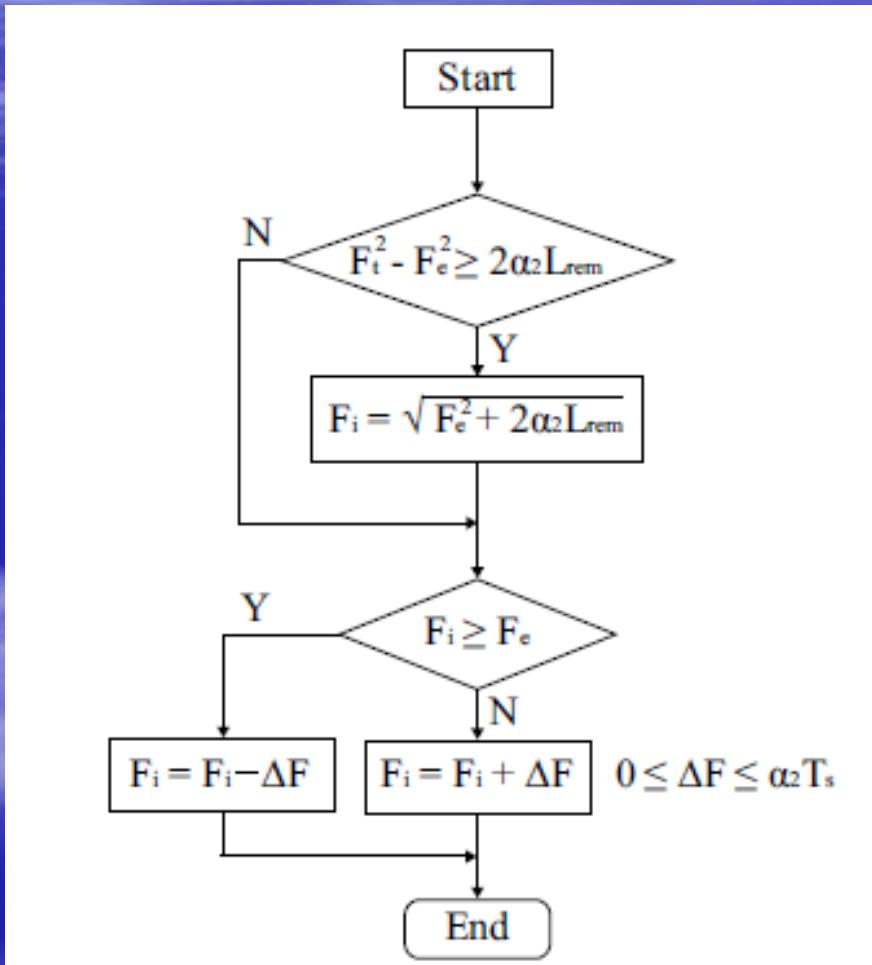
$$\alpha = \frac{(V_{performance})^2}{R}$$

$\alpha$  : performance index

$$V_{error} = \frac{2R \times \cos^{-1} \frac{R_E}{R+E}}{T_S}$$

$E$  : denotes the chordal error

# Speed within Block



$$F_i^2 - F_e^2 \geq 2\alpha_2 L_{rem}$$

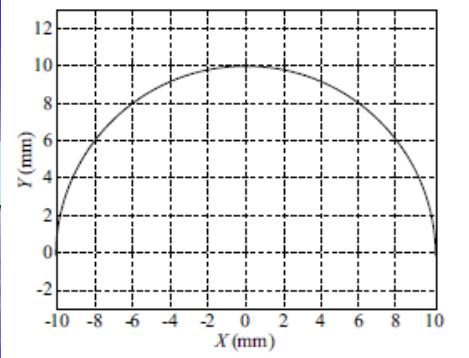
$$F_i = \sqrt{F_e^2 + 2\alpha_2 L_{rem}}$$

$$0 \leq \Delta F \leq \alpha_2 T_s$$

# Simulation Results

$$V_s = \sqrt{V_e^2 + 2AS}$$

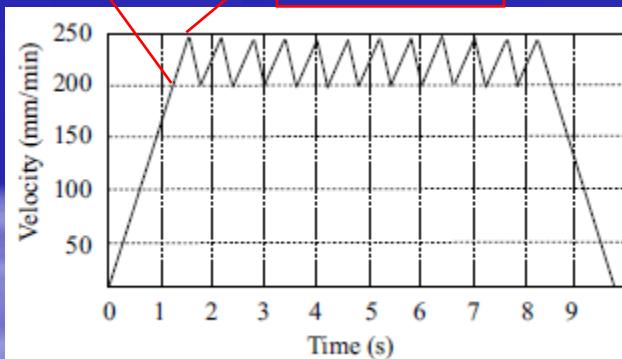
S=2.094



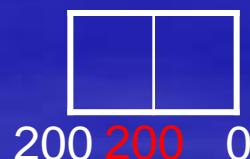
15 line segments, the radius of the half-circle is 10 mm, the commanded feedrate is 400 mm/min, and the allowable acceleration is 9600 mm/min.

$$V_{e(2)} = \sqrt{0^2 + 2 \times 2.094 \times 9600} = 200$$

245.17

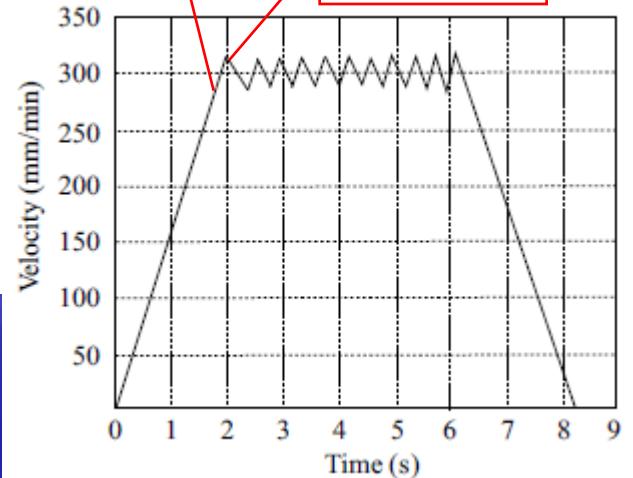


Look-ahead buffer size 2

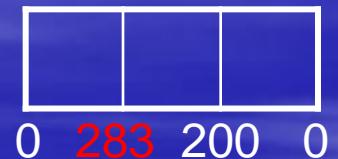


$$V_{e(3)} = \sqrt{200^2 + 2 \times 2.094 \times 9600} = 283$$

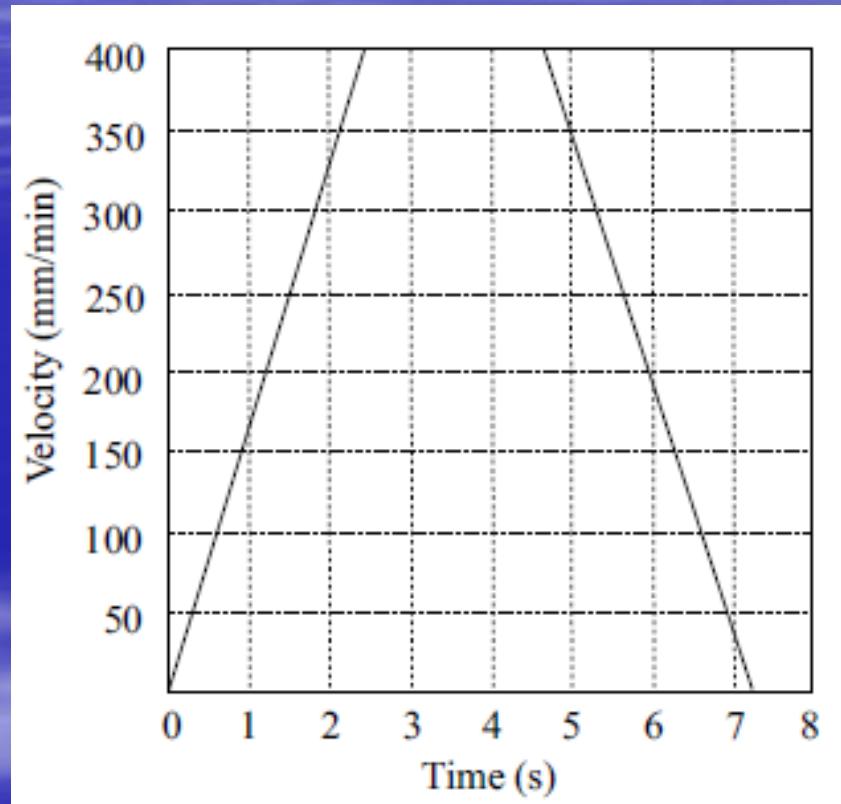
316.53



Look-ahead buffer size 3



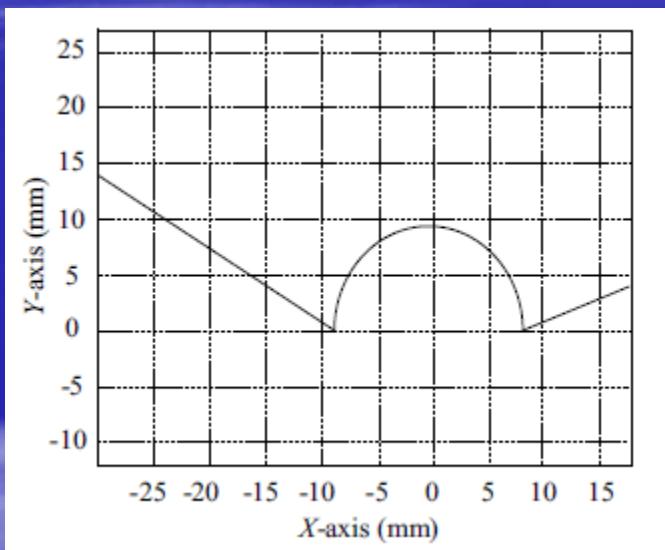
# Simulation Results



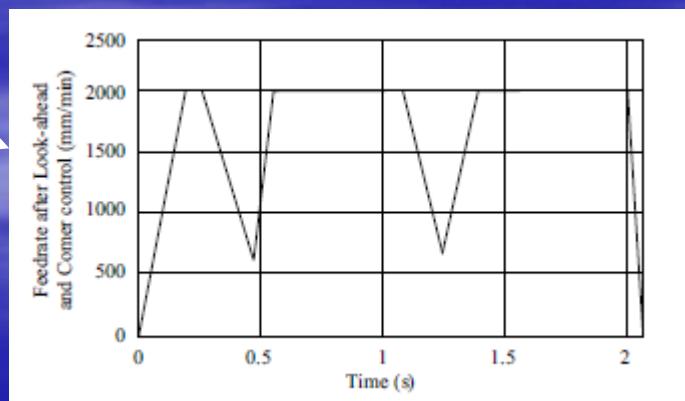
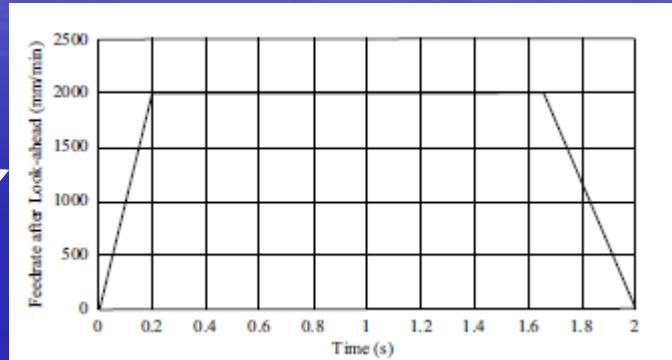
Look-ahead buffer size 6

# Simulation Results

EX :



feedrate of the paths is 2000 mm/min, the acceleration time is 200 msec, and the maximum allowable acceleration is 200000 mm/min<sup>2</sup>



# Summary

- ADCBI
  - the speed profile for two successive blocks is generated by considering the type of the current block
  - an arc path generates acceleration and deceleration due to the change of velocity and the acceleration and deceleration generate a mechanical shock
- mechanical shock  $\propto A$
- the method of determining the corner speed, two methods were introduced
- ADCAI
  - Acc/Dec control is applied to each axis separately and this leads to machining error in the case of machining an arc
  - ADCAI + Look Ahead algorithm → reduce the machining error within a specified amount
- ADCBI-type NCK is widely used for high-speed machining and ADCAI-type NCK is used for machining where high accuracy is not important, such as roughing machining.

# Theory and Design of CNC Systems

*Chapter 5*  
**PID Control System**

# Introduction

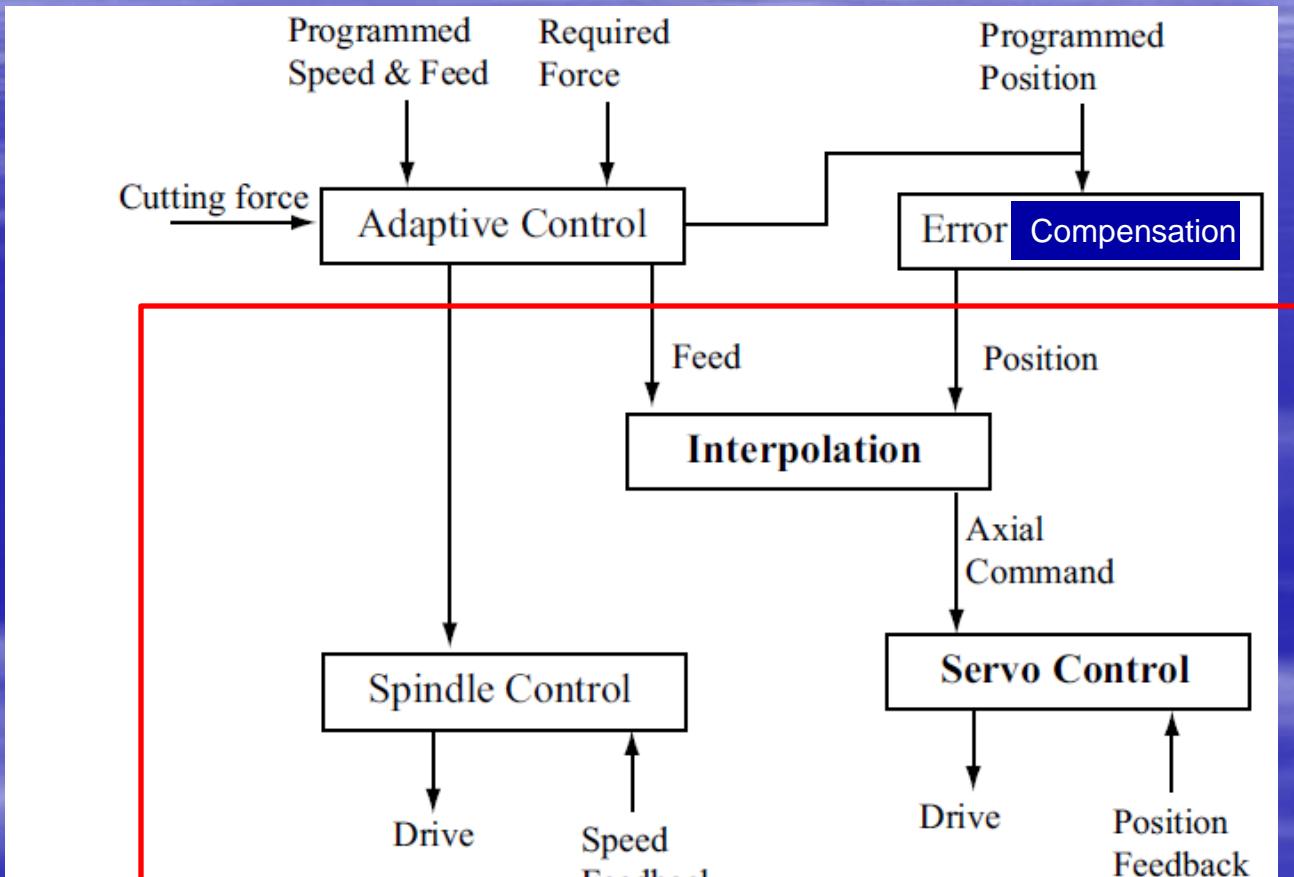


Fig. 5.1 Three-tier CNC system architecture

The majority of CNC system.

# Introduction

## ■ Adaptive Control

The adaptive control module in the upper layer generates optimized cutting conditions such as spindle speed and feedrate based on the programmed spindle speed, the programmed feedrate, measured actual cutting force, and the cutting capacity of machine tools. Accordingly, the adaptive control module plays the role of increasing Material Removal Rate (MRR) and decreasing machining time to increase productivity.

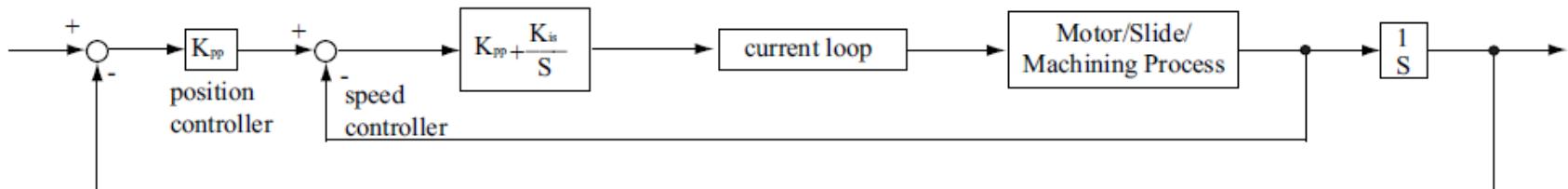
# Introduction

## ■ Error Compensation

Another module in the upper layer, the error compensation module, carries out the compensation of error factors that cause deviations from the programmed path. This module handles various kinds of error including the heat from moving elements, the volumetric error of machine tools, tool wear, and tool deflection.

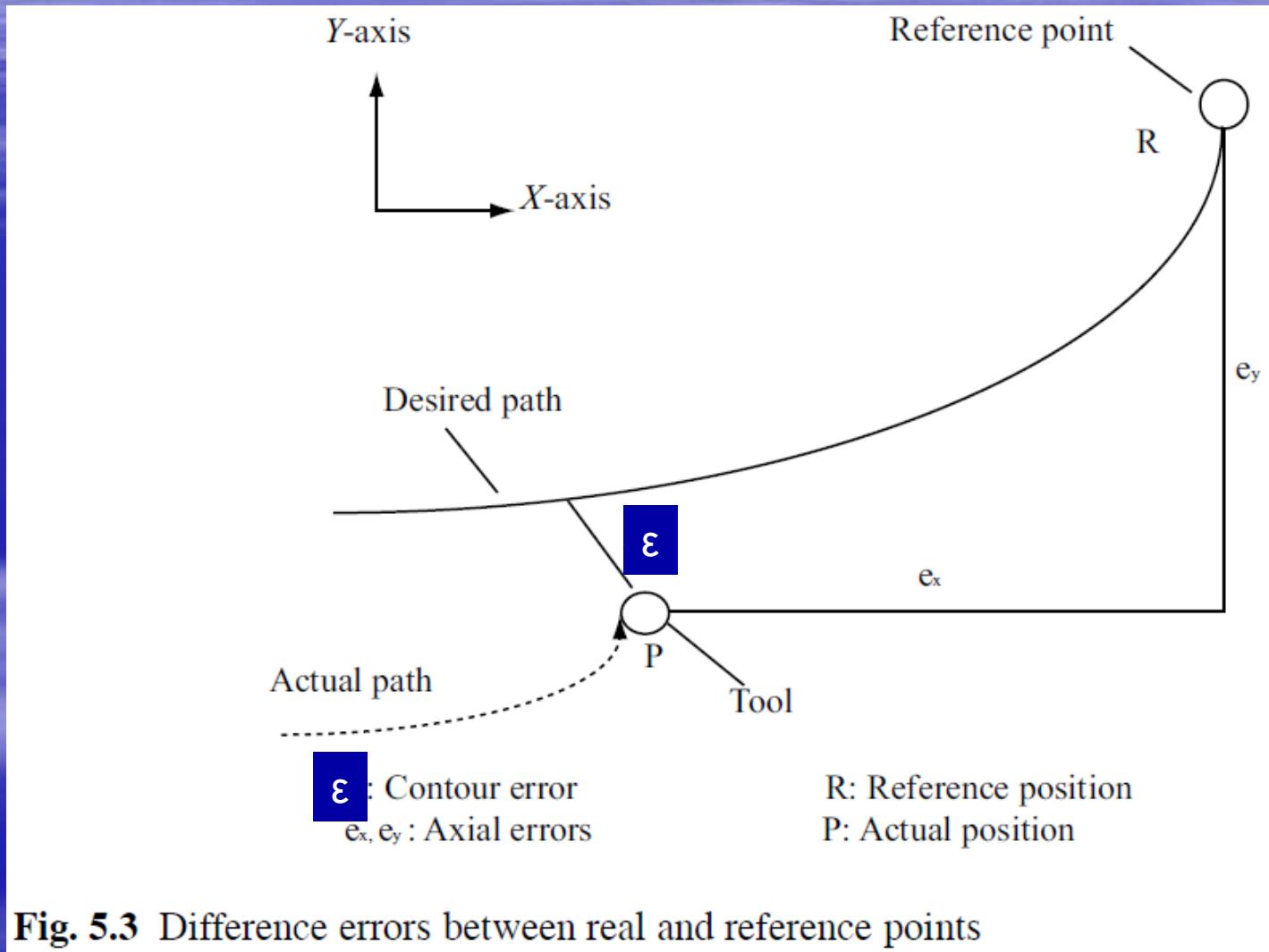
# The Servo Controller

In a CNC system, generally the NCK task is the outermost loop and performs only the position control with slowest response; while the speed control and current (torque) control are executed in a servo drive system. However, recently the conventional control configurational has no longer been preserved in the CNC system industry. All control loops can be implemented in the drive system, or the position and speed control loop are performed within the NCK task, while only the current loop for the fastest response is performed within the drive system. This can be chosen on the basis of the hardware capacity of the NCK and driver system, and the application purpose of the CNC system.



**Fig. 5.2** Cascade structure for closed-loop control

# Servo Control for Positioning



# Servo Control for Positioning

The trajectory and contour error appearing in the multi-axis machine tools are shown in Fig. 5.3.  $P$  is the current position, and  $R$  denotes the reference point to go to. The purpose of tracking control is to minimize the position errors of each axis,  $e_x$ ,  $e_y$ , which are the deviation of the current position from the reference point. On the other hand, contour control is implemented to minimize the contour error,  $\varepsilon$ , which is the error of the current position to the desired contour. Therefore, the position error and contour error are used as the indices for evaluating the accuracy of the controlled path.

The position error is the linear distance between the actual tool position and the reference point, and is represented like Eq. 5.1 by using the position error of each axis.

$$e = \sqrt{(e_x)^2 + (e_y)^2} \quad (5.1)$$

where,  $e$  denotes the position error and  $e_x$  and  $e_y$  denote the position error of  $X$ -axis and  $Y$ -axis respectively. The contour error,  $\varepsilon$ , is the minimum distance between the actual tool position and the desired path, as shown in Fig. 5.3.

# Servo Control for Positioning

Compared with the P controller, feedback controllers such as the PID controller and the Fuzzy controller decrease the position errors of each axis. The feed forward controller is useful for reducing the multi-axis error by reducing the axial error of each axis, and finally contributes to reduction of the contour error. The cross coupling controller performs accurate control in real time by generating control commands to reduce contour error based on a contour error model. In this textbook, the causes of various errors and algorithms for reducing the position error will be addressed.

# PID Controller

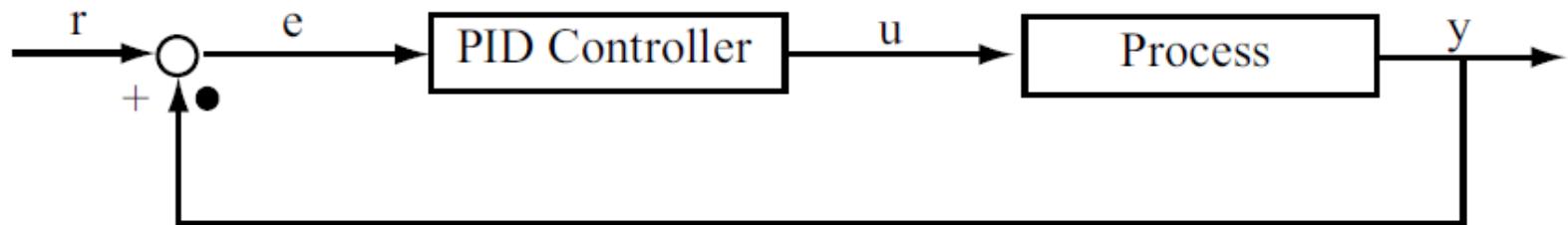


Fig. 5.4 Block diagram of position control in PID

$$G_c(s) = K_p + \frac{K_i}{s} + K_d s \quad (5.2)$$



$$G_c(s) = K_p \left( 1 + \frac{1}{T_i s} + T_d s \right) \quad (5.3)$$

$$T_i = K_p / K_i \quad (5.4a)$$

$$T_d = K_d / K_p \quad (5.4b)$$

# PID Controller

The transfer functions for the proportional controller, the integral controller, and the derivative controller defined in a continuous time domain can be approximated by the transfer functions for the discrete time domain for digital control as in Eqs. 5.5, 5.6 and 5.7.

$$G(s) = K_p \Leftrightarrow G(z) = K_p \quad (5.5)$$

$$G(s) = \frac{K_i}{s} \Leftrightarrow G(z) = \frac{K_i T}{1 - z^{-1}} \quad (5.6)$$

$$G(s) = K_d s \Leftrightarrow G(z) = \frac{K_d (1 - z^{-1})}{T} \quad (5.7)$$

# PID Controller

By combining the above three approximated equations, the transfer function for the PID controller for discrete time domains can be approximated as in Eq. 5.8.

$$G_c(z) = \frac{k_0 + k_1 z^{-1} + k_2 z^{-2}}{1 - z^{-1}} \quad (5.8)$$

where,  $k_0 = k_p + k_i T + \frac{k_d}{T}$ ,  $k_1 = -k_p - \frac{2k_d}{T}$ , and  $k_2 = \frac{k_d}{T}$  and  $T$  denotes the iteration time for position control. Accordingly, the output of the digital PID controller with proportional, integral, and derivative control can be represented as the difference equation, as in Eq. 5.9.

$$\frac{u}{e} = \frac{k_0 + k_1 z^{-1} + k_2 z^{-2}}{1 - z^{-1}} \quad (5.9a)$$

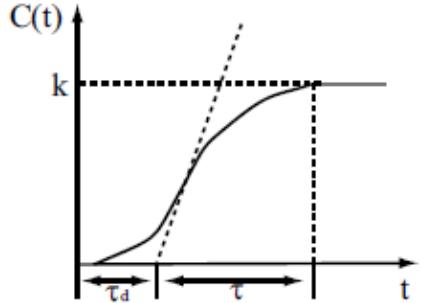
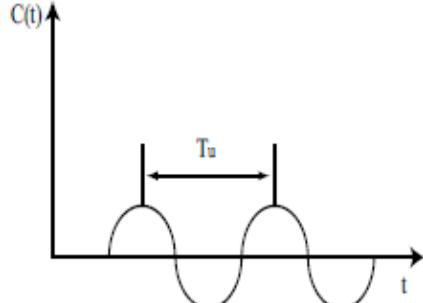
$$\text{or, } u(n) - u(n-1) = k_0 e(n) + k_1 e(n-1) + k_2 e(n-2) \quad (5.9b)$$

Equation 5.9b can be rewritten as Eq. 5.10.

$$u(n) = u(n-1) + k_0 e(n) + k_1 e(n-1) + k_2 e(n-2) \quad (5.10)$$

# PID Gain Tuning

Table 5.1 Ziegler–Nichols Method

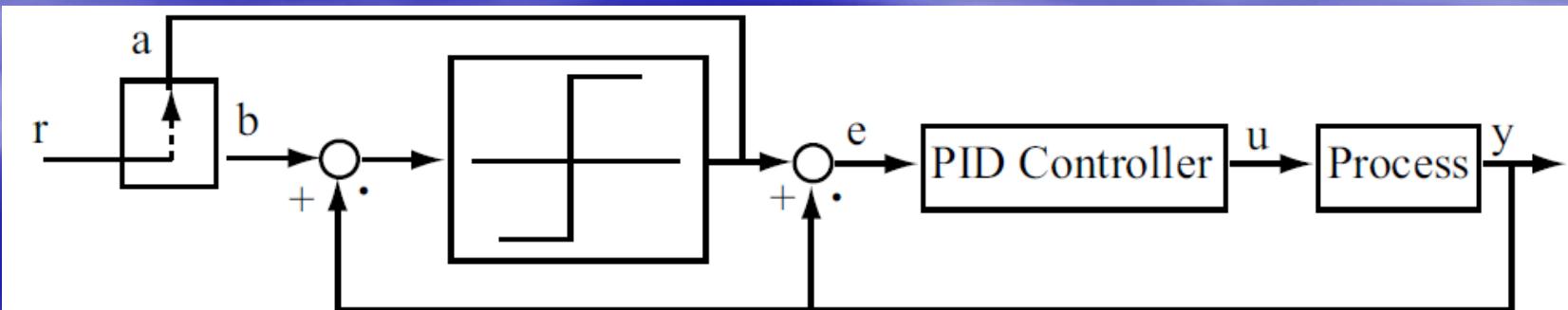
Control	Step Response	Ultimate Sensitivity Method
Condition	Step response has S-shape	As proportional gain increases, output oscillates
Procedure	1. By inputting step pulse to target, generate S-shaped response. 2. Draw tangent line on point of inflection of output. From intersection points where tangent line meets time axis and response K, calculate response delay time $\tau_d$ and time constant $\tau$	1. By setting $T_i = \infty$ and $T_d = 0$ in closed loop system, activate only proportional control loop. 2. Until output $c(t)$ keeps oscillating, regulate $K_p$ 3. Proportional gain at moment of oscillation is kept to critical gain $K_u$ . Set critical frequency $T_u$ to frequency at this moment.
Output response		
Ctrl. eqn.	P	$K_p = \tau / \tau_d$
	PI	$K_p = 0.9\tau / \tau_d, T_i = 3.3\tau_d$
	PID	$K_p = 1.2\tau / \tau_d, T_i = 2\tau_d, T_d = 0.5\tau_d$
		$K_p = 0.5K_u, T_i = 0.8T_u$
		$K_p = 0.45K_u, T_i = 0.8T_u$
		$K_p = 0.6K_u, T_i = 0.5T_u$
		$T_d = 0.125T_u$

# PID Gain Tuning

## ■ Relay Gain Tuning

The Ziegler–Nichols method mentioned above has the limitation that it cannot be applied to a system that is not critically oscillated with only proportional gain control. To overcome the non-oscillating problem, a relay gain tuning method was introduced. In this method, a relay is utilized for system oscillation by force. When oscillation occurs, the limits of frequency and amplitude are extracted from the system response.

The circuit for the relay-based automatic gain tuning method consists of the conventional PID controller, a relay, and a switch for switching reference input. The relay and the switch are located in front of the PID controller, as shown in Fig. 5.6.



**Fig. 5.6** Circuit for relay-based automatic gain tuning

# PID Gain Tuning

## ■ Relay Gain Tuning

The automatic gain tuning stage can be done as shown in Fig. 5.7a. The relay is modeled as a system with gain  $K_R$ . Furthermore,  $G_C(s)$  and  $G_P(s)$  mean the transfer transition function of the PID controller and the process, respectively. The error can be expressed as Eq. 5.13.

$$e = K_R r - y - K_R y = K_R r - (1 + K_R)y \quad (5.13)$$

If Fig. 5.7a is arranged as the form of the rightmost equation of Eq. 5.13, it can be denoted as Fig. 5.7b, and the loop transfer function is  $(1 + K_R)G_C(s)G_P(s)$ .

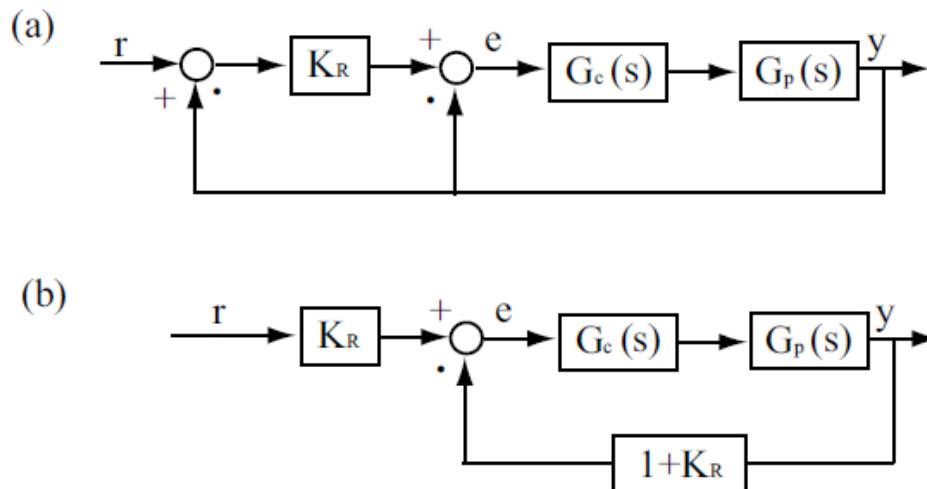


Fig. 5.7 Block diagrams of gain setting

# PID Gain Tuning

## ■ Relay Gain Tuning

While the gain of a relay is expressed using the describing function of non-linear system theory as Eq. 5.14.

$$K_R = \frac{4d}{\pi a} \quad (5.14)$$

where,  $a$  is the amplitude of the process output and  $d$  denotes the amplitude of the relay.

At this stage, the process transfer function is assumed to be as in Eq. 5.15 which is used in the Ziegler–Nichols method and briefly represents the characteristics of the process.

$$G_p(s) = \frac{e^{-\tau s}}{T_p s} \quad (5.15)$$

where,  $T_p$  is the time constant and  $\tau$  stands for the dead time.

# PID Gain Tuning

## ■ Relay Gain Tuning

On the other hand, the transition function of the PID controller can be represented as in Eq. 5.16.

$$G_C(s) = K_C \left( 1 + \frac{1}{T_i s} \right) \left( \frac{T_d s + 1}{\alpha T_d s + 1} \right) \quad (5.16)$$

where,  $T_i$  and  $T_d$ , respectively, denote the integration time and derivative time and  $\alpha$  stands for derivative gain.

# PID Gain Tuning

## ■ Relay Gain Tuning

If oscillation is continuously induced at the moment that the relay is connected with the PID controller and the automatic gain tuning stage begins, the frequency and amplitude of the induced oscillation are derived as in the following equations,

$$\arg [(1 + K_R)G_P(j\omega_0)G_C(j\omega_0)] = \quad (5.17)$$

$$\arg \left[ (1 + K_R) \frac{e^{-\tau s}}{T_p s} \left( \frac{K_C(T_i s + 1)}{T_i s} \right) \left( \frac{T_d s + 1}{\alpha T_d s + 1} \right) \right]_{s=j\omega_0} = -\pi$$

$$|(1 + K_R)G_P(j\omega_0)G_C(j\omega_0)| = \quad (5.18)$$

$$\left| (1 + K_R) \frac{e^{-\tau s}}{T_p s} \left( \frac{K_C(T_i s + 1)}{T_i s} \right) \left( \frac{T_d s + 1}{\alpha T_d s + 1} \right) \right|_{s=j\omega_0} = 1$$

# PID Gain Tuning

## ■ Relay Gain Tuning

where  $\omega_0$  is the oscillation frequency. The dead time  $\tau$  and time constant  $T_p$  which show the characteristic of the process can be found using Eqs. 5.19 and 5.20.

$$\tau = \frac{\tan^{-1}(T_i\omega_0) + \tan^{-1}(T_d\omega_0) - \tan^{-1}(\alpha T_d\omega_0)}{\omega_0} \quad (5.19)$$

$$T_p = \frac{(1 + K_R)K_C \sqrt{T_i^2\omega_0^2 + 1} \sqrt{T_d^2\omega_0^2 + 1}}{T_i\omega_0^2 \sqrt{\alpha^2 T_d^2\omega_0^2 + 1}} \quad (5.20)$$

# PID Gain Tuning

## ■ Relay Gain Tuning

In order to compute PID gains by the estimation of the dead time and time constant from Eqs. 5.19 and 5.20, an ultimate gain and frequency are necessary to use the equation of the response curve of the Ziegler–Nichols method mentioned in Table 5.1. In the case that the Ziegler–Nichols gain tuning method is used, the relationships between those parameters, the frequency and the amplitude are given by Eqs. 5.21 and 5.22.

$$\arg [G_P(j\omega_u)G_C(j\omega_u)] = \arg \left[ K_u \frac{e^{-\tau s}}{T_{PS}} \right]_{s=j\omega_u} = -\pi \quad (5.21)$$

$$|G_P(j\omega_u)G_C(j\omega_u)| = \left| K_u \frac{e^{-\tau s}}{T_{PS}} \right|_{s=j\omega_u} = 1 \quad (5.22)$$

# PID Gain Tuning

## ■ Relay Gain Tuning

where,  $\omega_u$  denotes an ultimate frequency. From  $T_u = 2\pi/\omega_u$ , the ultimate frequency can be obtained from Eq. 5.23.

$$T_u = 4\tau \quad (5.23)$$

Also, the ultimate gain is given by Eq. 5.24.

$$K_u = \frac{2\pi T_P}{T_u} \quad (5.24)$$

Finally, by applying Eq. 5.23 and Eq. 5.24 to the ultimate sensitivity method, adequate PID gains can be obtained.

# PID Gain Tuning

## ■ Relay Gain Tuning Step 1 to 5

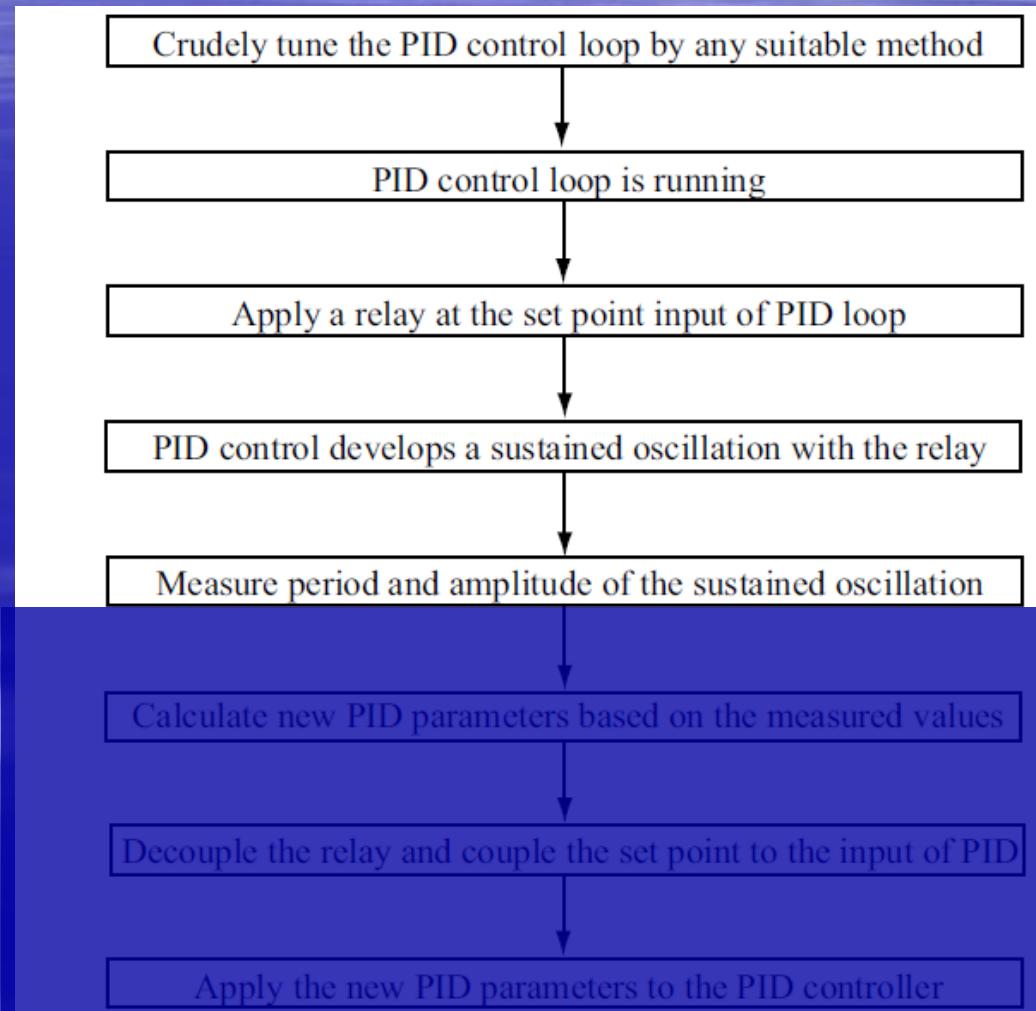


Fig. 5.8 Automatic gain tuning method

# PID Gain Tuning

## ■ Relay Gain Tuning

### Step 6

$$\tau = \frac{\tan^{-1}(T_i\omega_0) + \tan^{-1}(T_d\omega_0) - \tan^{-1}(\alpha T_d\omega_0)}{\omega_0} \quad (5.19)$$

$$T_p = \frac{(1 + K_R)K_C \sqrt{T_i^2\omega_0^2 + 1} \sqrt{T_d^2\omega_0^2 + 1}}{T_i\omega_0^2 \sqrt{\alpha^2 T_d^2\omega_0^2 + 1}} \quad (5.20)$$

Crudely tune the PID control loop by any suitable method

PID control loop is running

Set point input of PID loop

Observe sustained oscillation with the relay

Measure period and amplitude of the sustained oscillation

Calculate new PID parameters based on the measured values

$$T_u = 4\tau \quad (5.23)$$

Decouple the relay and couple the set point to the input of PID

$$K_u = \frac{2\pi T_P}{T_u} \quad (5.24)$$

Apply the new PID parameters to the PID controller

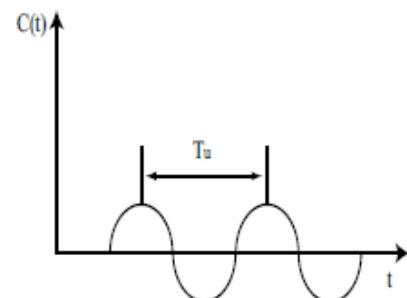
Fig. 5.8 Automatic gain tuning method

# PID Gain Tuning

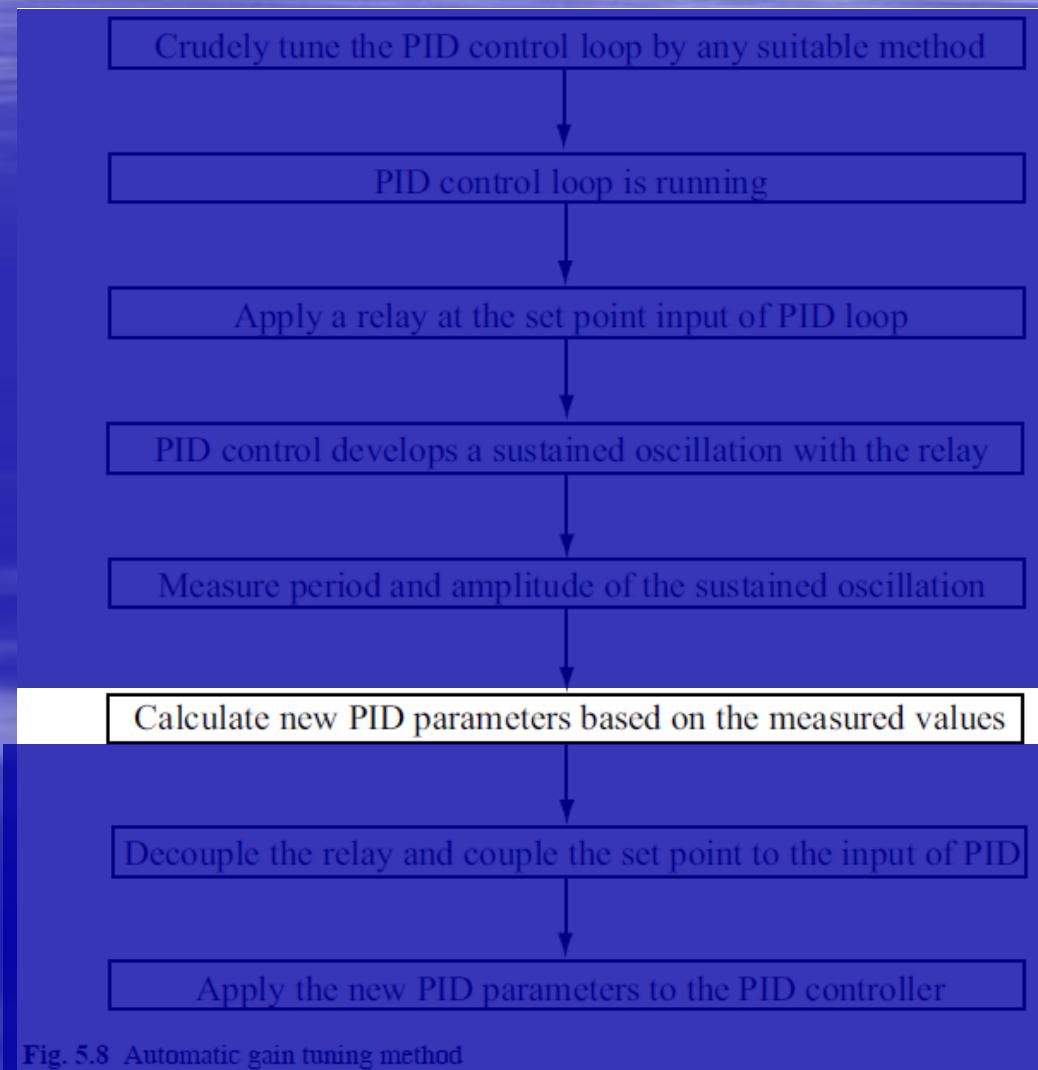
## ■ Relay Gain Tuning

### Step 6

Ultimate Sensitivity Method
As proportional gain increases, output oscillates
1. By setting $T_i = \infty$ and $T_d = 0$ in closed loop system, activate only proportional control loop.
2. Until output $c(t)$ keeps oscillating, regulate $K_p$
3. Proportional gain at moment of oscillation is kept to critical gain $K_u$ . Set critical frequency $T_u$ to frequency at this moment.



$$\begin{aligned}K_p &= 0.5K_u \\K_p &= 0.45K_u, T_i = 0.8T_u \\K_p &= 0.6K_u, T_i = 0.5T_u \\T_d &= 0.125T_u\end{aligned}$$



# PID Gain Tuning

## ■ Relay Gain Tuning

### Step 7 to 8

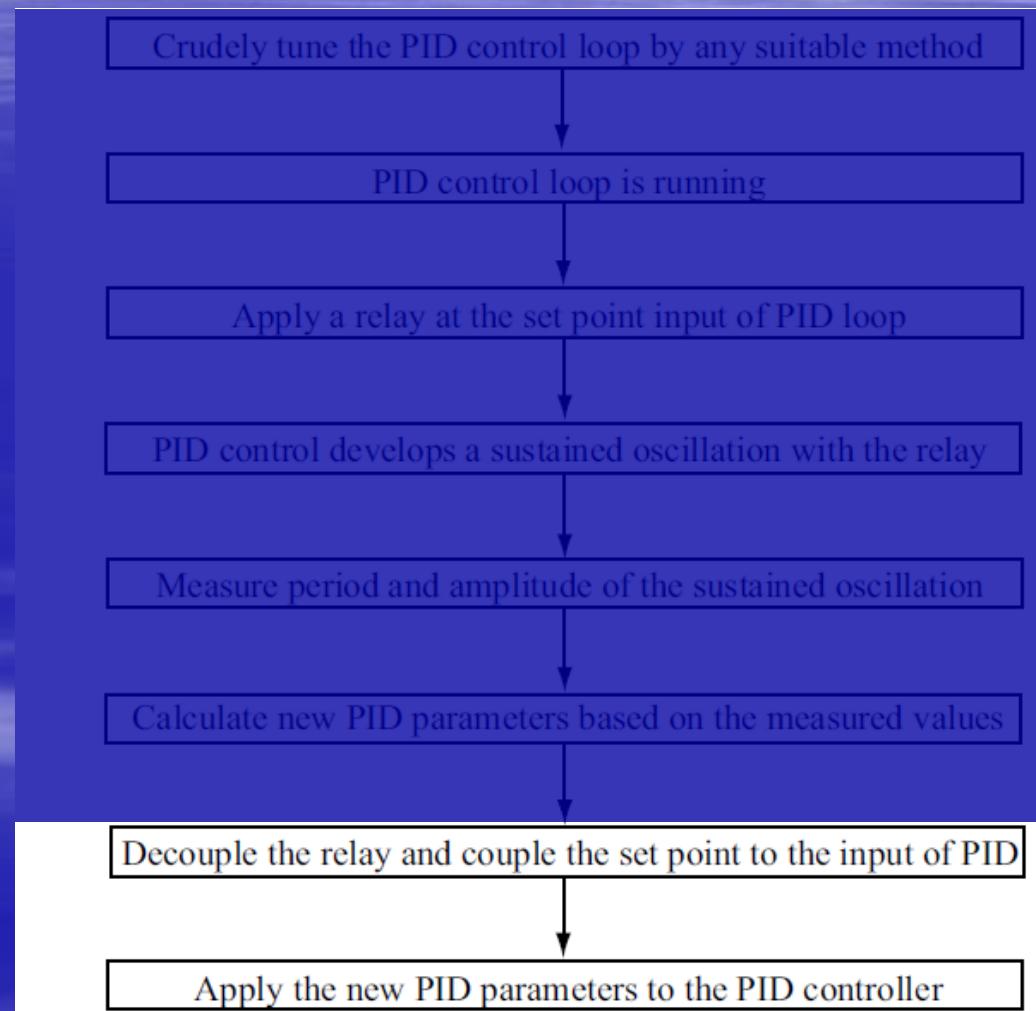


Fig. 5.8 Automatic gain tuning method

# Feedforward Control

The feedforward control method can be classified into two types from the point of view of the loop structure.

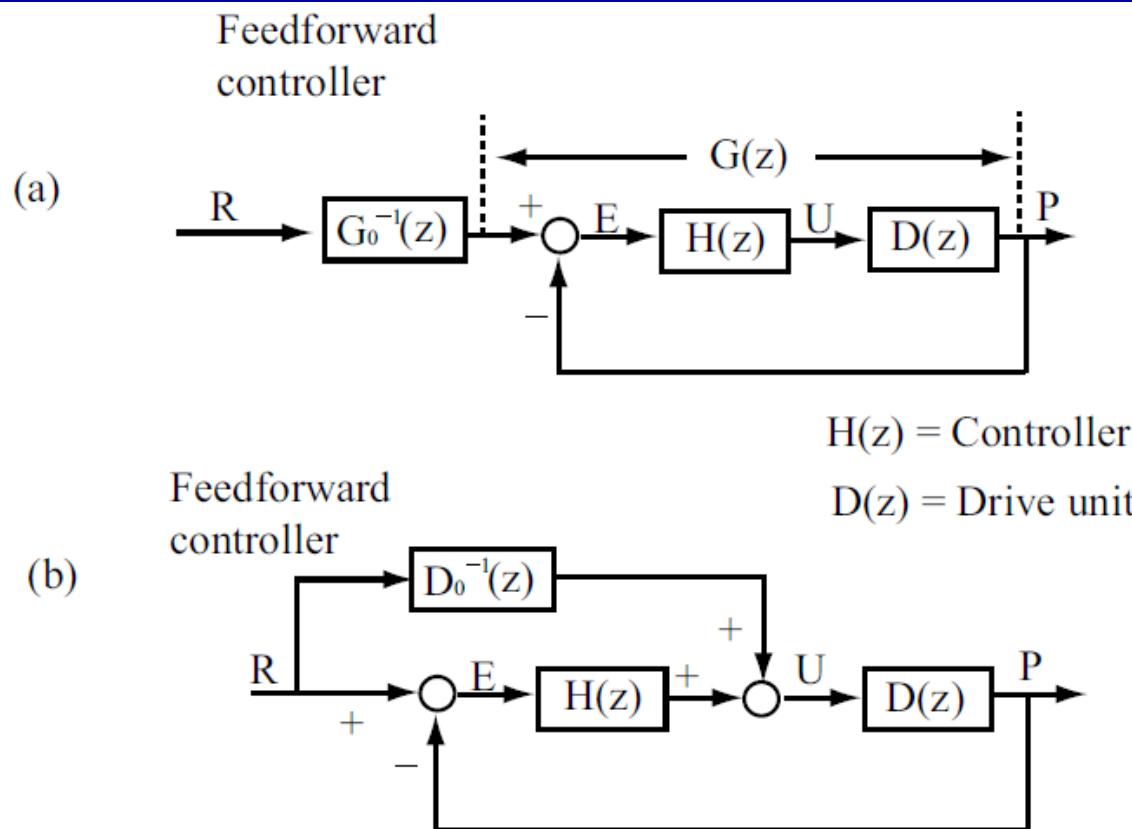


Fig. 5.10 Feedforward control structure

# Feedforward Control

In the first type,  $G_0^{-1}(z)$ , the inverse of the transfer function of the complete system including the feedback loop,  $G(z)$ , is set as the transfer function of the feedforward control loop. Since the total transfer function of the complete system is set as  $G_0^{-1}(z)G(z) = 1$ , the actual controlled position and the desired position can become equal. In the second type, the transfer function of the feedforward controller is set to be the inverse of the transfer function of the drive unit,  $D_0^{-1}(z)$ . Therefore, the transfer function of the entire control loop is written as Eq. 5.25.

$$G_C(z) = \frac{D_0^{-1}(z)D(z) + H(z)D(z)}{1 + H(z)D(z)} \quad (5.25)$$

# Feedforward Control

Consequently, if  $D_0^{-1}(z)D(z) = 1$  is satisfied, the actual position and the desired position have come to be the same. In the first type, the inverse transfer function is very complex because the transfer functions of the feedback controller and the drive module are included in the inverse transfer function. However, when the inverse transfer function ( $G_0^{-1}$  or  $D_0^{-1}$ ) has unstable poles, modification of the feedback controller is necessary. For modification of the feedback controller, the first type is more suitable than the second.

# Feedforward Control

## ■ ZPETC (Zero Phase Error Tracking Control)

In ZPETC, the numerator term of the transfer function of a closed loop can be divided into terms including only the stable zeroes  $B_c^s(z^{-1})$  and terms including only the unstable zeroes  $B_c^u(z^{-1})$ , as shown in Eq. 5.26.

$$B_c(z^{-1}) = B_c^s(z^{-1})B_c^u(z^{-1}) \quad (5.26)$$

Now, the ZPETC feedforward controller can be represented as follows by utilizing the special numerator term.

$$G^{-1}(z) = \frac{z^d A(z^{-1}) B_c^u(z)}{B_c^s(z^{-1})(B_c^u(1))^2} \quad (5.27)$$

In consequence, the entire transfer function can be summarized as follows,

$$G_{zpetc}(z) = \frac{B_c^u(z) B_c^u(z^{-1})}{(B_c^u(1))^2} \quad (5.28)$$

# Feedforward Control

## ■ ZPETC (Zero Phase Error Tracking Control)

We can understand that the phase difference of the transfer function from the desired path to output is zero because the transfer function results in the multiplication of two conjugating complex numbers. This means that the output traces the input with no time delay, the gain in the steady state is one and the error in the steady state becomes zero. When the transfer function of a closed loop has zeroes located on the left half of the plane, the gain increases. This means that ZPETC can have the gain error in the high-frequency range instead of making the phase difference zero. On the other hand, ZPETC is a good algorithm for building good characteristic by using small information in the low frequency range. The ZPETC requires an accurate system model, which is somewhat difficult, and shows poor tracing performance when the desired trajectory includes a fast transient shape such as at sharp corners.

# Feedforward Control

## ■ IKF (Inverse Compensation Filter)

The IKF (Inverse Compensation Filter) was introduced by Weck to solve the corner-tracing problem of ZPETC. As shown in Fig. 5.11, a low-pass filter is added at the front of the feedforward controller to improve traceability. Consequently the smooth trace becomes possible by removing the high-frequency range from the input signal.

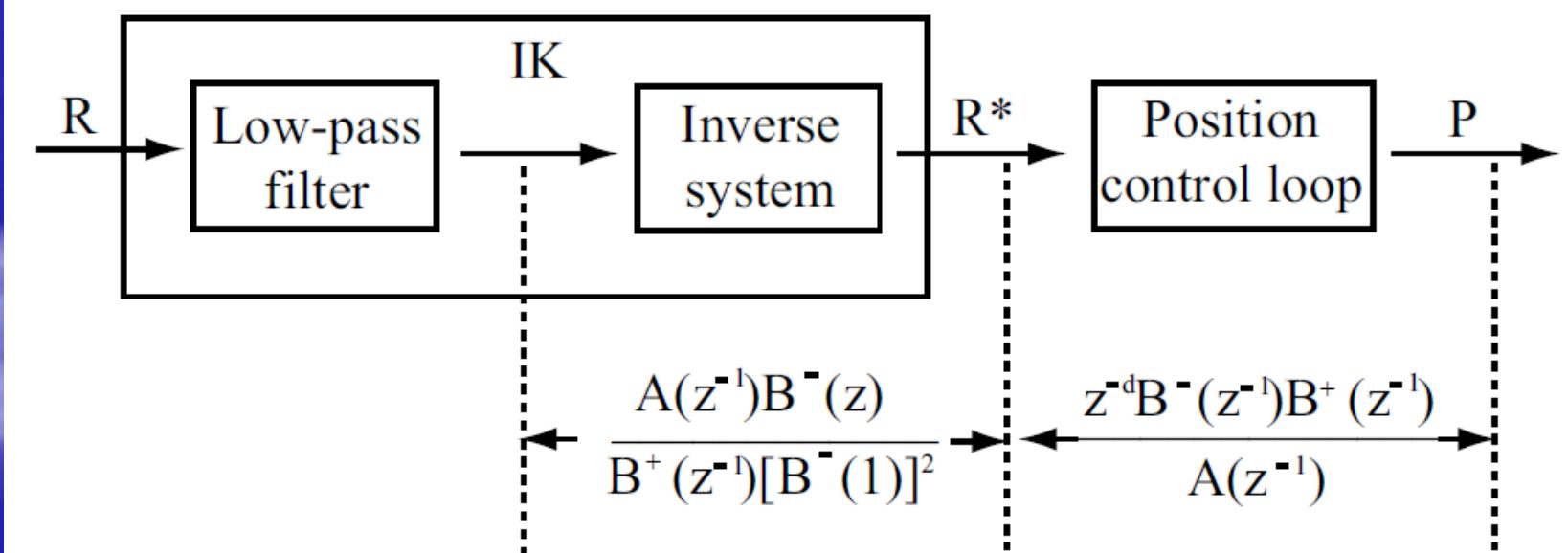


Fig. 5.11 IKF filter structure

# Feedforward Control

## ■ Causal/Non-causal FIR (Finite Impulse Response)

The controllers mentioned in the previous sections require an accurate system model and the transfer function is formulated using a complicated process. However, it is difficult to apply these controllers to a real CNC system. Therefore, a practical technique is typically used in industry, where the position command is forwarded to the feedback controller together with velocity and acceleration/deceleration commands, as shown in Fig. 5.12.

$$G(z) = a_0 z^{-2} + a_1 z^{-1} + a_2 \quad (5.29)$$

$$\text{where, } a_0 = \frac{V_3}{T^2}, \quad a_1 = -\frac{V_2}{T} - \frac{2V_3}{T^2}, \quad a_2 = 1 + \frac{V_2}{T} + \frac{V_3}{T^2}$$

# Feedforward Control

## ■ Causal/Non-causal FIR (Finite Impulse Response)

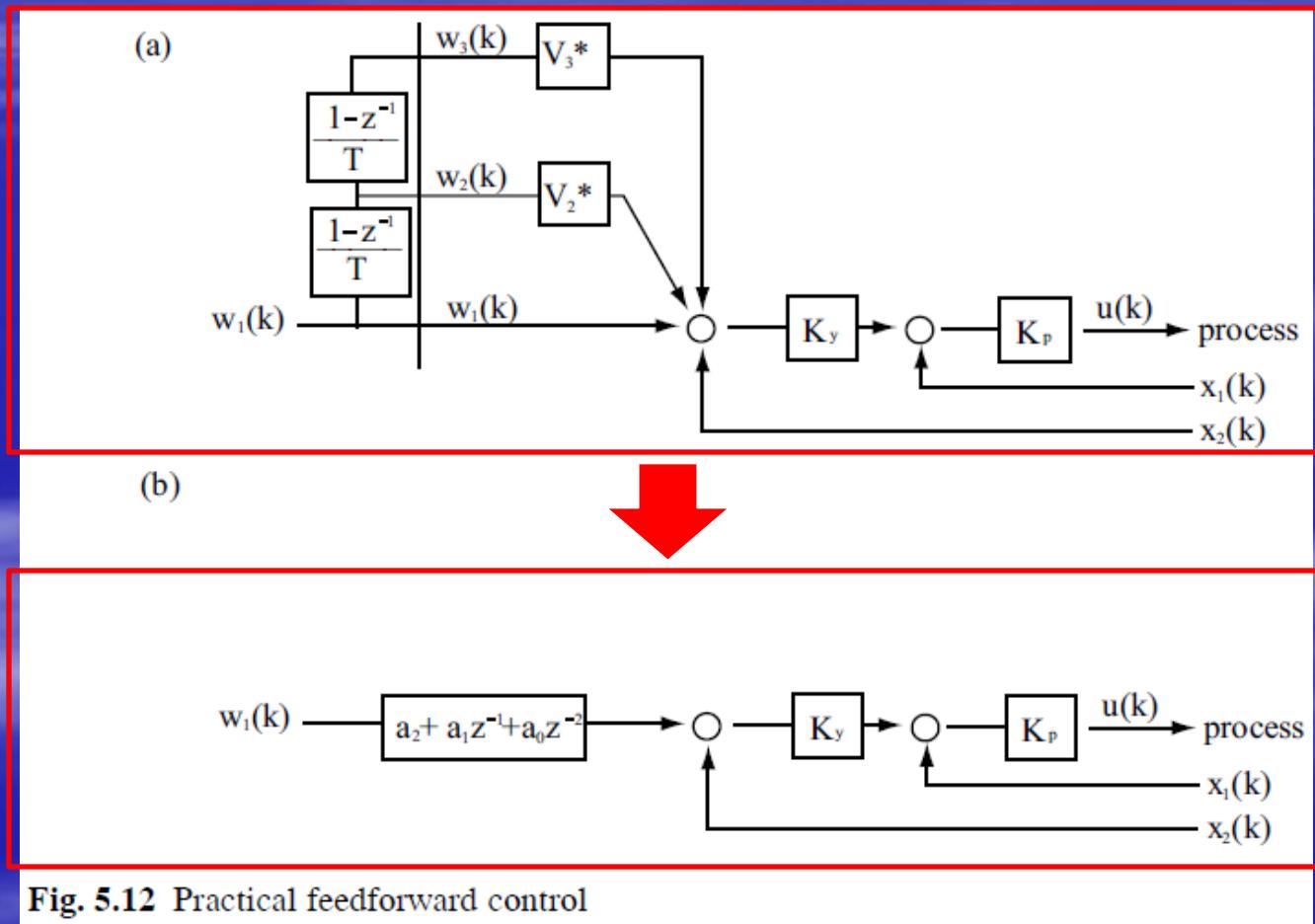


Fig. 5.12 Practical feedforward control

# Feedforward Control

## ■ Causal/Non-causal FIR (Finite Impulse Response)

The above causal FIR filter uses the current position data and the position data from one and two steps before. It can also be represented in the non-causal FIR filter form that requests future position data.

$$G(z) = a_0 + a_1z + a_2z^2 \quad (5.30)$$

The architecture of the controller can be as shown in Fig. 5.13.

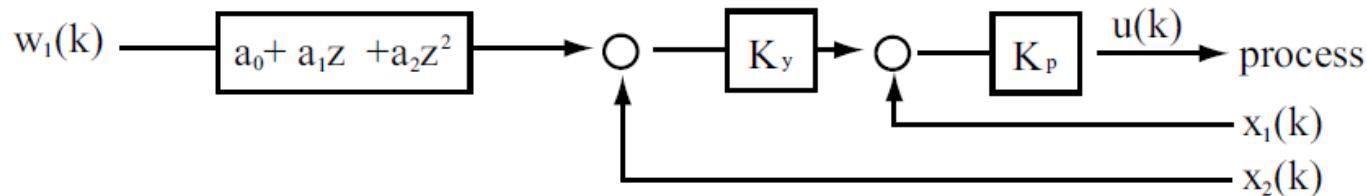
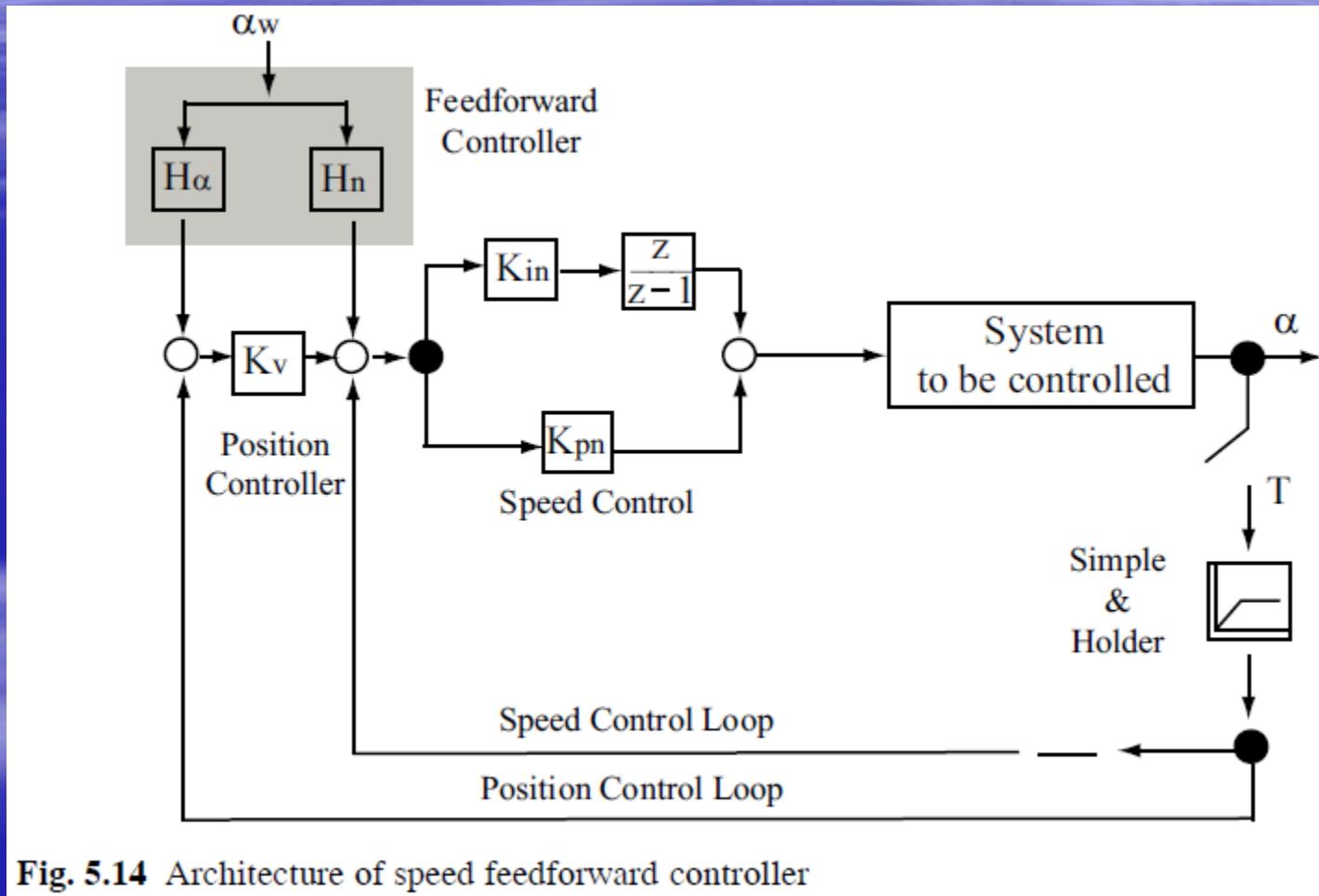


Fig. 5.13 Non-causal FIR

# Speed Feedforward Controller



# Speed Feedforward Controller

Figure 5.14 shows the architecture of a speed feedforward controller where the position command,  $\alpha_w$ , is filtered and the filtered command is input directly to the speed control loop in a drive system in order to improve the response of the position control loop.  $K_v$  in Fig. 5.14 denotes the proportional gain of the position controller and only proportional control is used for the position control of a CNC system.  $K_{pn}$  and  $K_{in}$  in Fig. 5.14 denote a propositional gain and an integral gain of the speed controller, respectively, and it is typical that a PI control is used for the speed control of a CNC system.  $\alpha_w$  and  $\alpha$  denote the desired position and the actual position respectively, and  $T$  is the sampling time of the position control loop.

# Torque Feedforward Controller

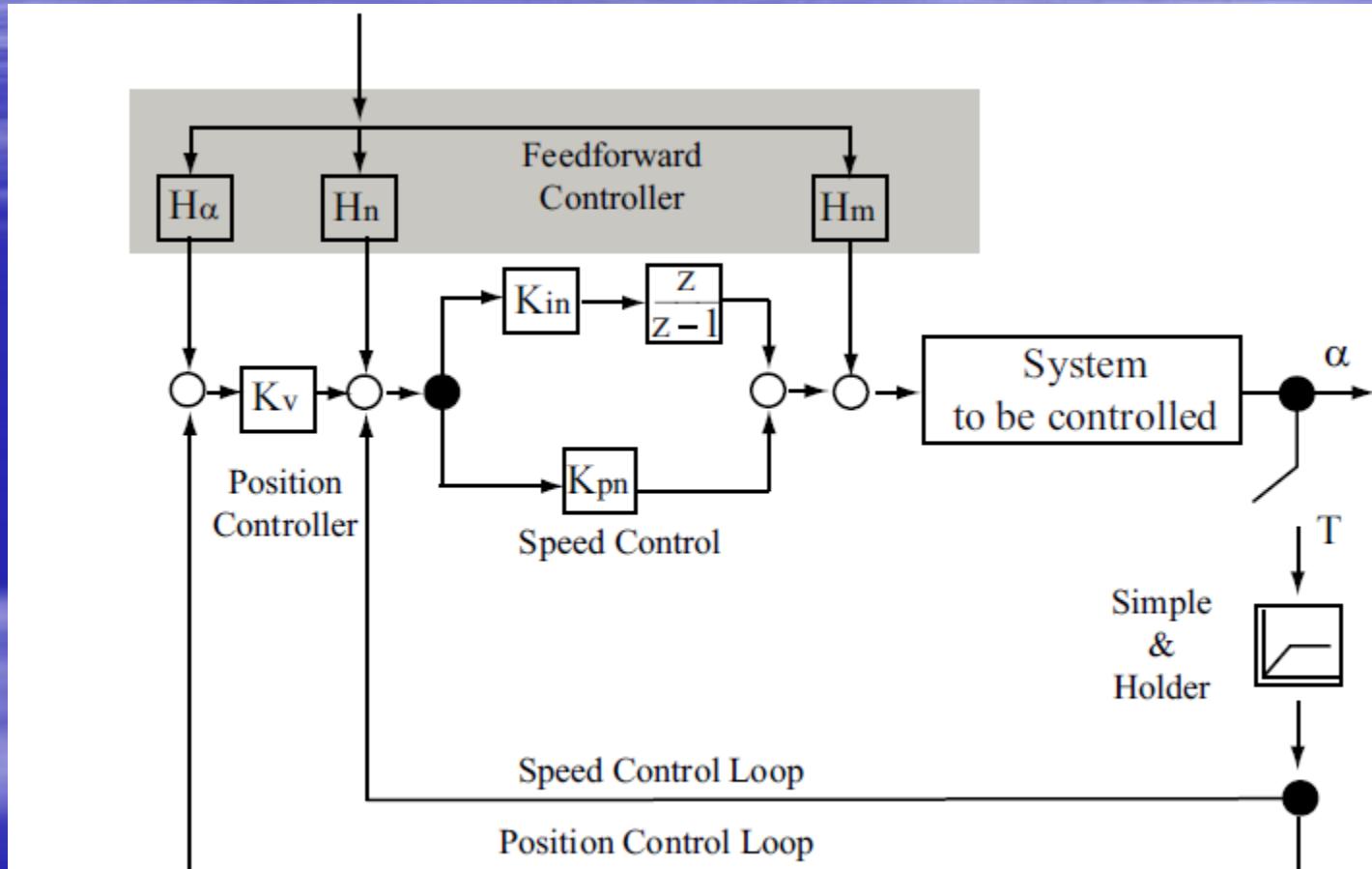


Fig. 5.15 Architecture of a torque feedforward controller

# Torque Feedforward Controller

Figure 5.15 shows the architecture of a torque feedforward controller. In the torque feedforward controller, a position command is directly input to the current loop (*e.g.* system to be controlled) that shows the fastest system response. Therefore, in the torque feedforward controller, the speed control loop including a current loop inside and position control loop should fulfill the input balance. Since the torque feedforward controller generates the position command that is directly input to the current loop, the speed command from an interpolator is used as an input to the feedforward controller for stable control. Because the torque feedforward controller actuates the motors by using the current loop with the fastest response, it generates the minimum following error under serial control architecture.

# The Following Error

## Feedback Controller

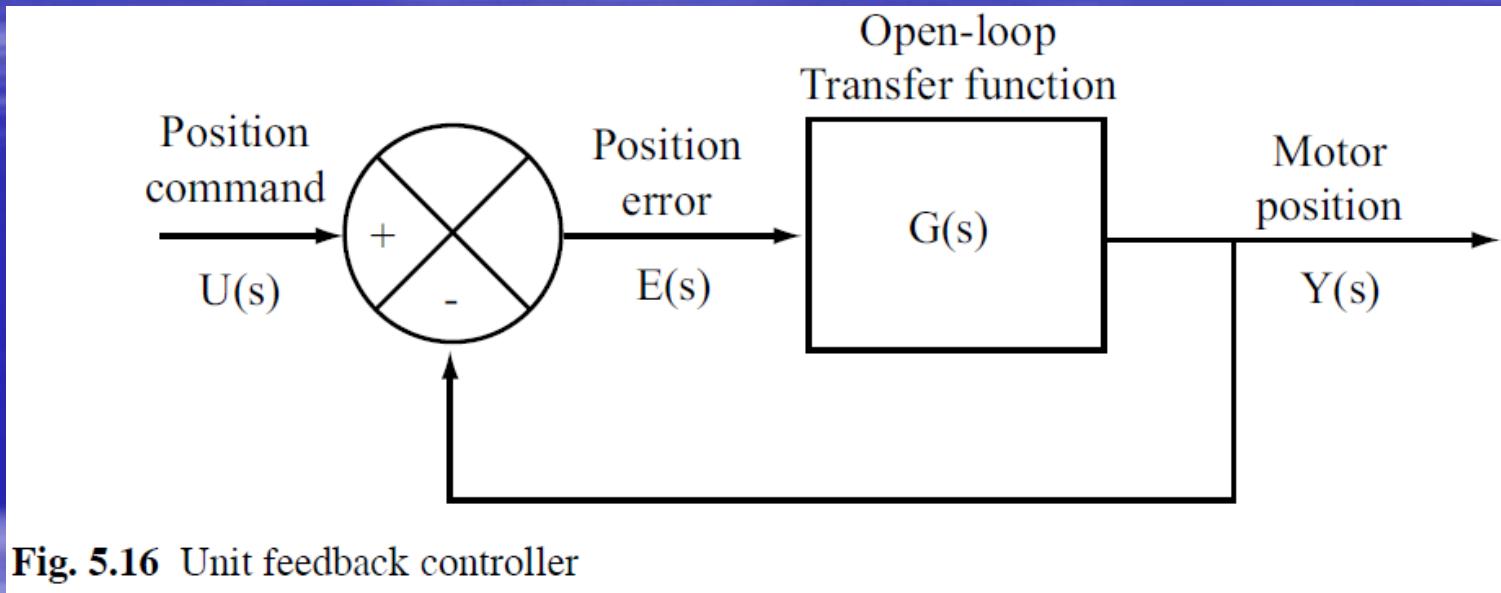


Fig. 5.16 Unit feedback controller

The transfer function of an open loop,  $G(s)$ , is generally represented as follows.

$$G(s) = \frac{K}{s^l} \cdot \frac{(1 + sT'_1)(1 + sT'_2) \dots (1 + sT'_m)}{(1 + sT_1)(1 + sT_2) \dots (1 + sT_n)} \quad (5.31)$$

where  $l + n \geq m$  and  $K$  denotes the gain of the controller.

Proper transfer function

# The Following Error

## ■ Feedback Controller

In this case, the position error is defined as the difference between the desired position and the output position.

$$E(s) = U(s) - Y(s) \quad (5.32)$$

The transfer function of a closed loop,  $W(s)$ , can be written as follows,

$$W(s) = \frac{Y(s)}{U(s)} = \frac{G(s)}{1 + G(s)} \quad (5.33)$$

From Eq. 5.32 and Eq. 5.33,  $E(s)$  is summarized as follows,

$$E(s) = \frac{1}{1 + G(s)} U(s) \quad (5.34)$$

where the steady state error is denoted by  $e(\infty)$ , being  $e(t)$  as  $t \rightarrow \infty$ , and it can be calculated by using the Final Value Theorem for Laplace transformations,

$$e(\infty) = \lim_{t \rightarrow \infty} e(t) = \lim_{s \rightarrow 0} sE(s) \quad (5.35)$$

# The Following Error

## ■ Feedback Controller

The steady state error  $e_p$  for the step input  $u(t) = A$  or  $U(s) = \frac{A}{s}$  is summarized as follows by using Eqs. 5.31, 5.34, and 5.35.

$$\begin{aligned} e_p &= \lim_{s \rightarrow 0} sE(s) \\ &= \lim_{s \rightarrow 0} s \cdot \frac{1}{1 + \frac{K}{s^l} \cdot \frac{(1+sT'_1)(1+sT'_2)\dots(1+sT'_m)}{(1+sT_1)(1+sT_2)\dots(1+sT_n)}} \cdot \frac{A}{s} \\ &= \lim_{s \rightarrow 0} s \cdot \frac{1}{1 + \frac{K}{s^l}} \cdot \frac{A}{s} \\ &= \lim_{s \rightarrow 0} \frac{A}{1 + \frac{K}{s^l}} \end{aligned} \tag{5.36}$$

Consequently, the steady state error is obtained from Eq. 5.36 as follows,

$$\begin{aligned} \text{if } l = 0 \text{ then } e_p &= \frac{A}{1+K} \\ \text{if } l \geq 1, \text{ then } e_p &= 0 \end{aligned}$$

wherein,  $l = 1$  means that  $G(s)$  or the controlled system includes the integral element  $\frac{1}{s}$ . If the number of the integral elements is more than one,  $e_p$  is equal to zero then the steady-state error does not occur.

# The Following Error

## ■ Feedback Controller

Now find the normal error,  $e(\infty)$ , if the input is a Ramp input.

In the case of  $u(t) = At$  or  $U(s) = \frac{A}{s^2}$ , the steady-state error  $e_p$  can be expressed as follows,

$$\begin{aligned} &= \lim_{s \rightarrow 0} s \cdot \frac{1}{1 + \frac{K}{s^l} \cdot \frac{(1+sT'_1)(1+sT'_2)\dots(1+sT'_m)}{(1+sT_1)(1+sT_2)\dots(1+sT_n)}} \cdot \frac{A}{s^2} \\ &= \lim_{s \rightarrow 0} s \cdot \frac{1}{1 + \frac{K}{s^l}} \cdot \frac{A}{s^2} \\ &= \lim_{s \rightarrow 0} \frac{A}{1 + \frac{K}{s^{l-1}}} \end{aligned} \tag{5.37}$$

Finally,

if  $l = 0$  then  $e_p = \infty$ .

if  $l = 1$  then  $e_p = \frac{A}{K}$ .

if  $l \geq 2$  then  $e_p = 0$ .

# The Following Error

## ■ Feedback Controller

By utilizing the same method, if the steady state error  $e_p$  due to the acceleration input  $u(t) = \frac{A}{2}t^2$  is considered the result is expressed as:

If  $l = 0, 1$ , then  $e_p = \infty$ .

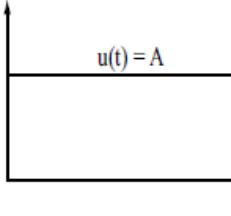
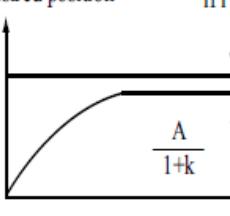
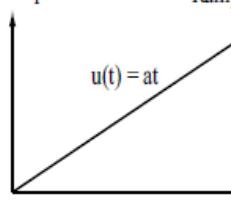
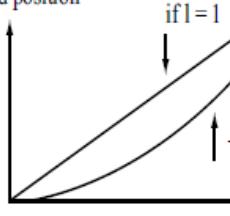
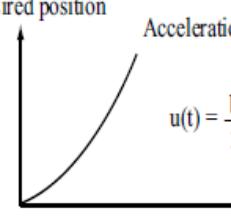
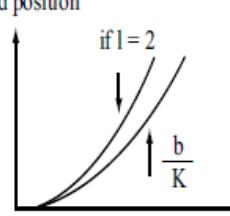
If  $l = 2$ , then  $e_p = \frac{A}{K}$ .

If  $l \geq 3$ , then  $e_p = 0$ .

# The Following Error

## ■ Feedback Controller

Table 5.2 Normal error summary

Input $u(t)$	Steady state error	Example of steady state error
Desired position  $u(t) = A$	$e_p = \begin{cases} l=0: \frac{A}{1+k} \\ l>1: 0 \end{cases}$	Desired position  $\text{if } l=0$
Desired position  $u(t) = at$	$e_p = \begin{cases} l=0: \infty \\ l=1: \frac{a}{K} \\ l>2: 0 \end{cases}$	Desired position  $\text{if } l=1$
Desired position  $u(t) = \frac{b}{2} t^2$	$e_p = \begin{cases} l=0, 1: \infty \\ l=2: \frac{b}{K} \\ l>3: 0 \end{cases}$	Desired position  $\text{if } l=2$

# The Following Error

## ■ Feedback Controller

Consequently, if we assume that the input command is a function of time,  $u(t) = Vt \times t$  (where,  $Vt$  is the commanded feedrate in the CNC system), the following error is typically as in Eq. 5.38.

$$e_{fbc} = \frac{V_t}{K_v} \quad (5.38)$$

Therefore, we conclude that the following error is inversely proportional to the gain of the position controller and is proportional to the feedrate when only feedback control is used. In consequence, it is necessary to increase the gain of the position controller or decrease the feedrate in order to decrease the following error.

# The Following Error

## ■ Feedforward Controller

If we summarize the following error of the speed feedforward without the complicated derivation process, it can be expressed in short form as Eq. 5.39.

$$e_{ffc}^v = V_t T_{en} \quad (5.39)$$

The following error of the torque feedforward controller can written as Eq. 5.40.

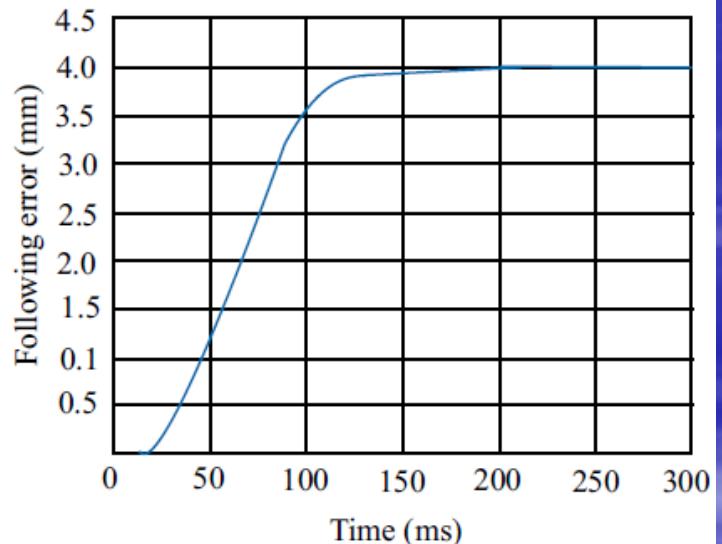
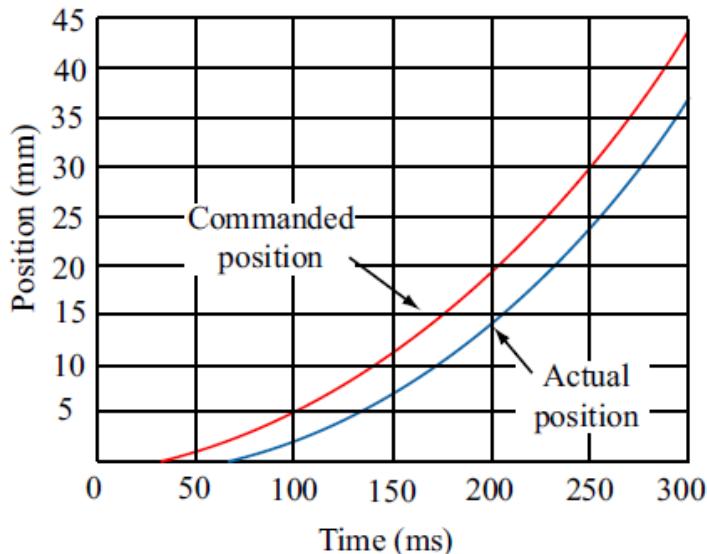
$$e_{ffc}^t = V_t (T_{ei} + T/2) \quad (5.40)$$

where  $V_t$ ,  $T_{en}$ ,  $T_{ei}$ , and  $T$  stand for the feedrate, the time constant of the speed control loop, the time constant of the current loop, and the sampling time, respectively.

# The Following Error

## ■ Comparison of Following Error

Without Feedforward



**Fig. 5.17** Comparison of following errors

# The Following Error

## Comparison of Following Error

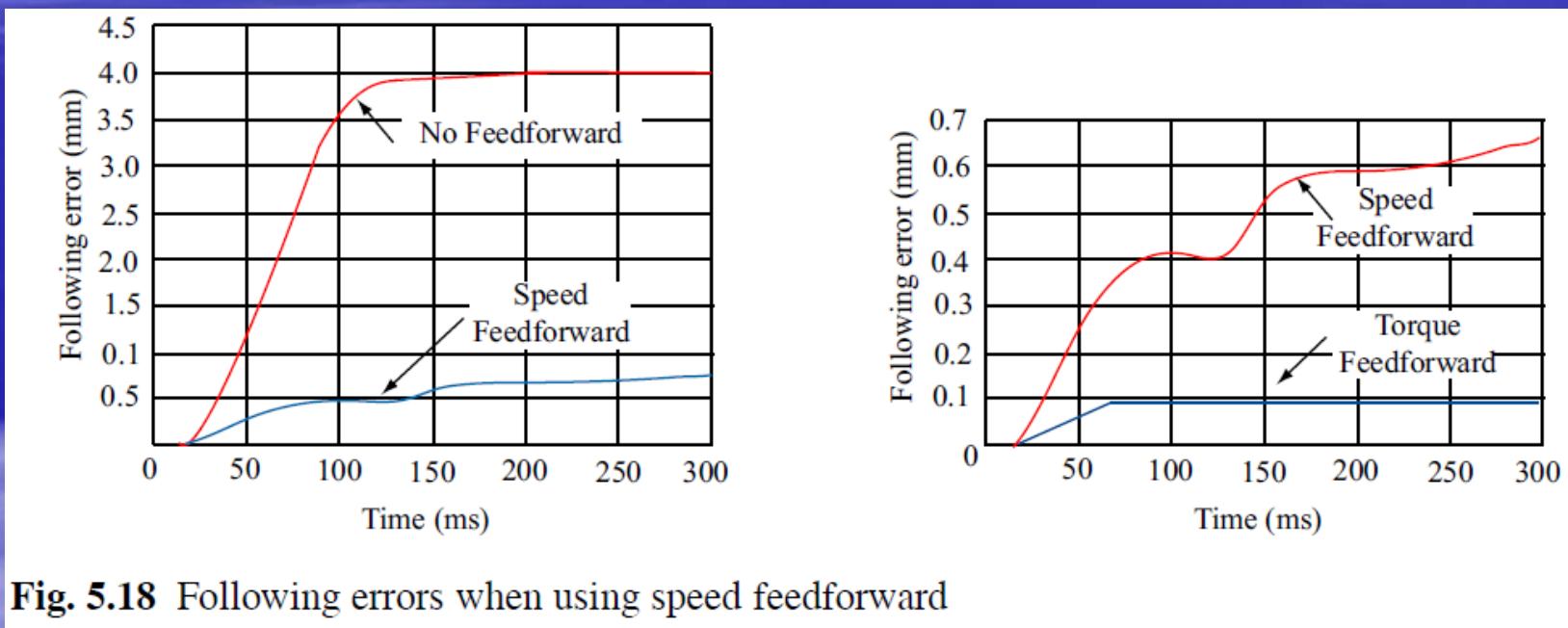


Fig. 5.18 Following errors when using speed feedforward

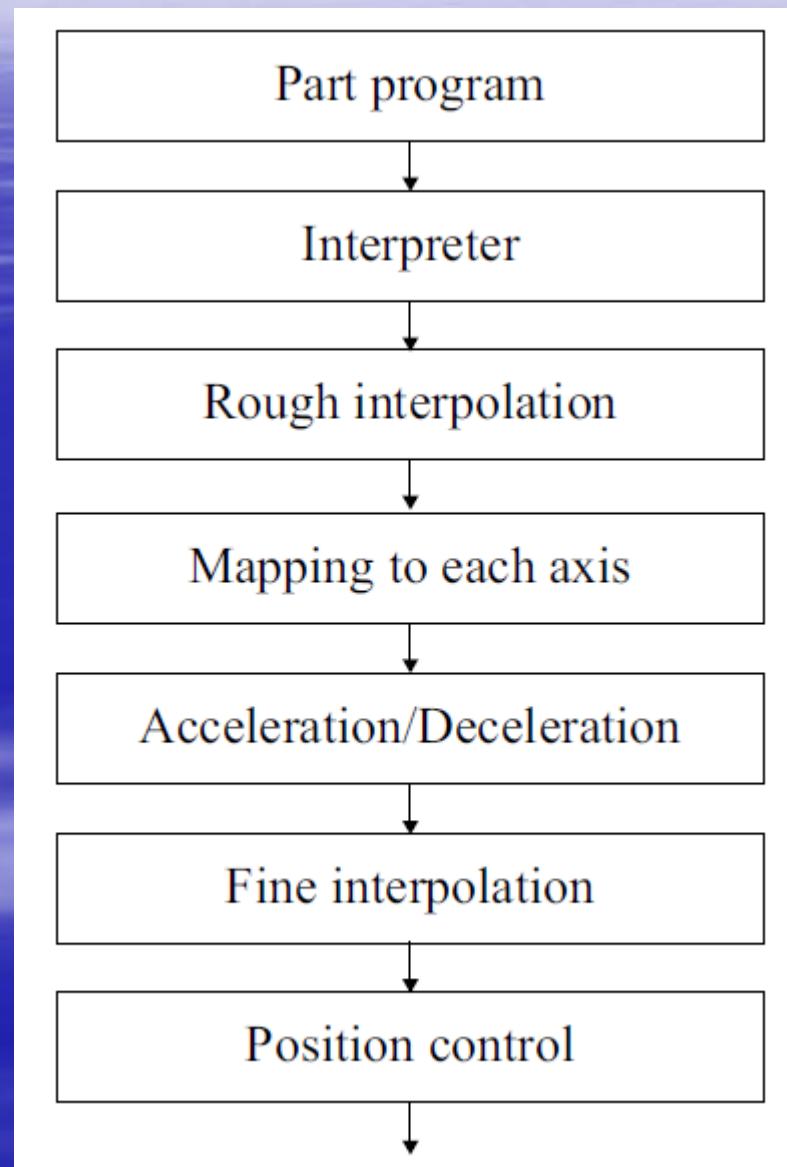
# Theory and Design of CNC Systems

## *Chapter 6* Numerical Control Kernel

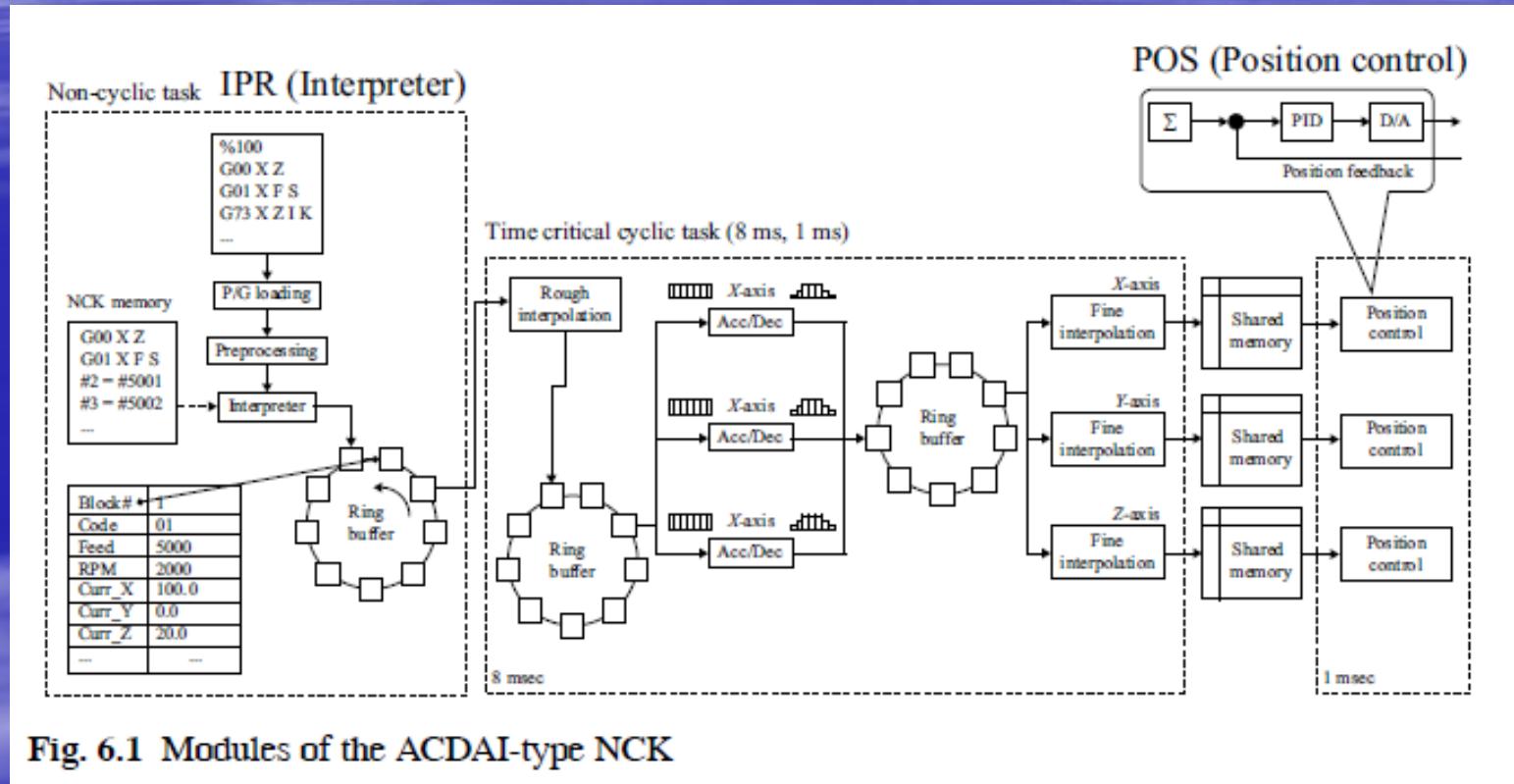
# Introduction

- Architecture of ACDAI-type NCK
- Architecture of an ADCBI-type NCK

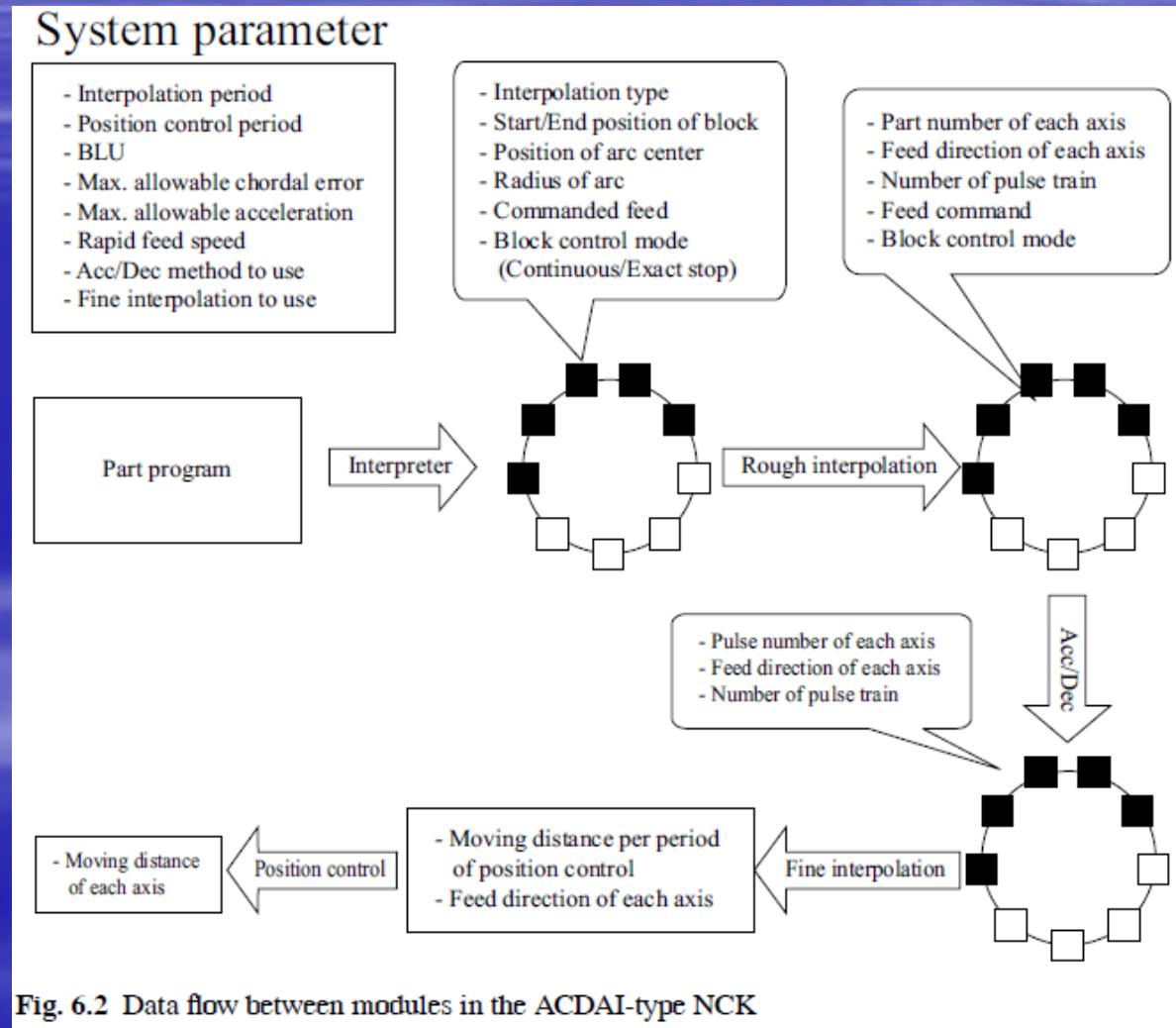
# ACDAI

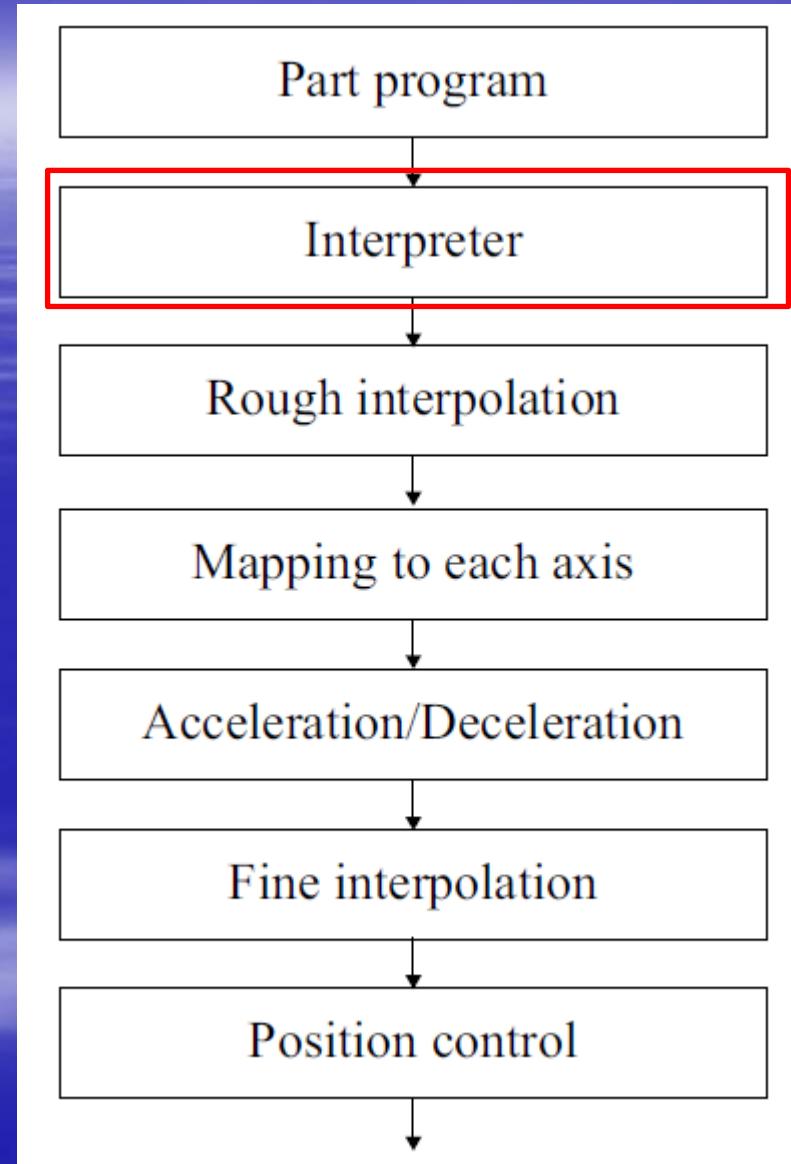


# Architecture of ACDAI-type NCK



# Architecture of ACDAI-type NCK

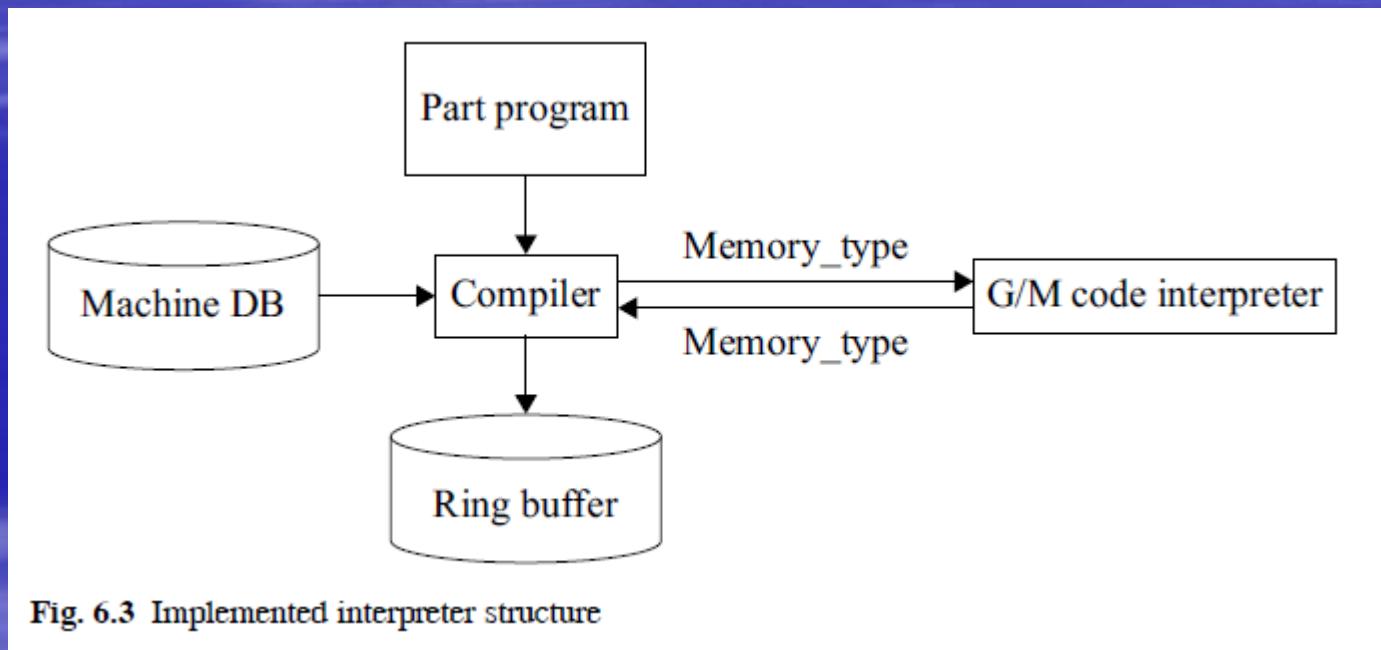




# Implementation of the Interpreter

As mentioned in Chapter 2, the input of the NCK is the part program. G-code, which is currently used for defining the part program, supports various functions such as tool compensation, coordinate transformation, cycle code, user-defined G-code, and sub program calls for convenience of editing part programs. To execute these functions and calculate the actual toolpath accurately, complicated computational tasks such as offsetting and coordinate transformations are required. The Interpreter has the task of computing the actual toolpath from the part program specified by G-codes or Macros.

# The Structure of the Interpreter



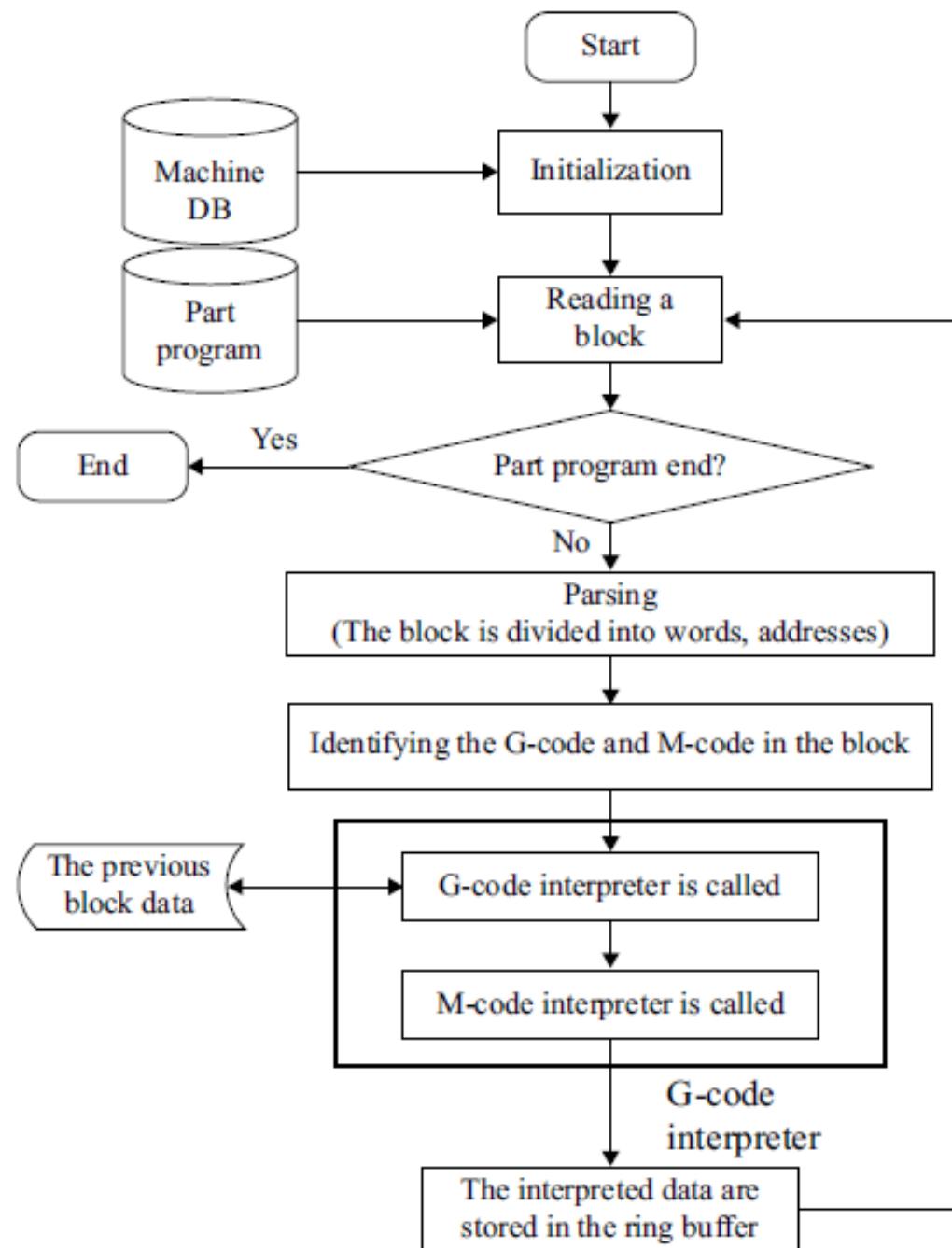
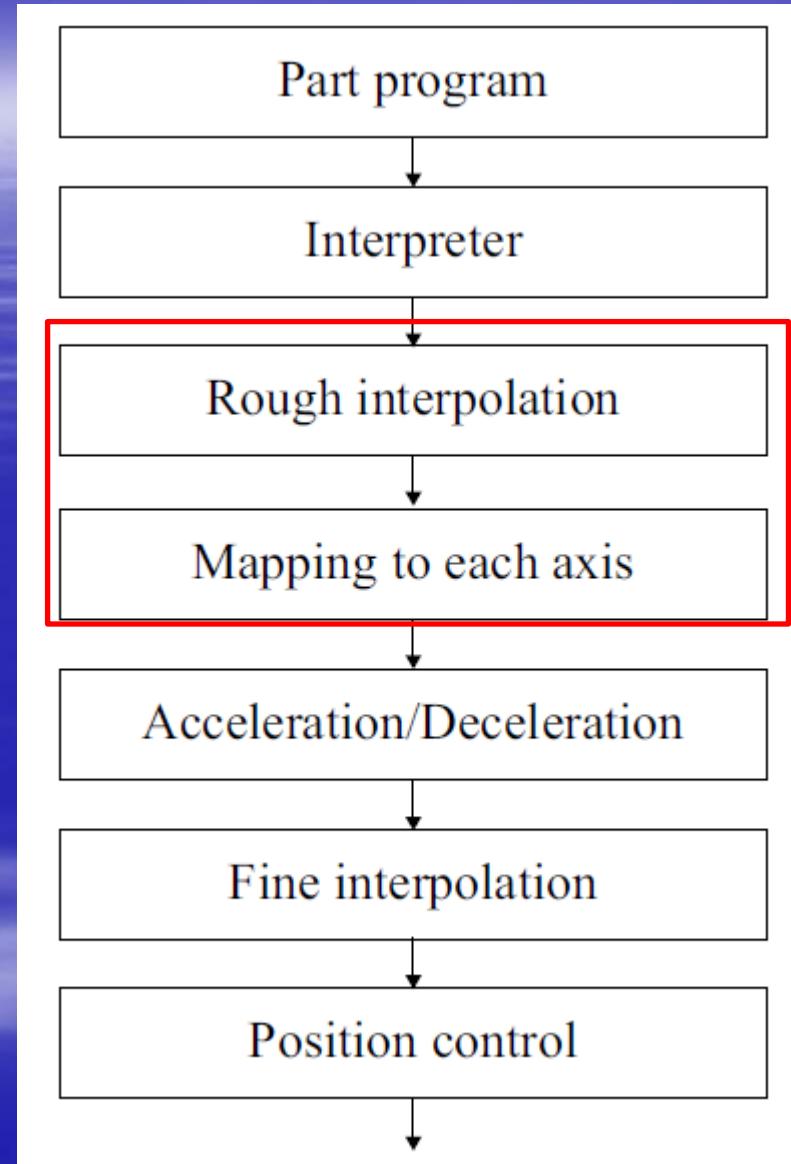


Fig. 6.4 Execution procedure for NCK

## Output from the interpreter

The interpreter finally generates and saves the actual start and end positions of the tool for each block in the workpiece coordinate system. If the block specifies an arc movement, the interpreter generates and saves the center position and radius of the arc. The interpolation type and control mode that are required for execution of the Interpolator and Acc/Dec Controller are stored. The output of the Interpolator is used as input to the Rough Interpolator in the case of ADCAI-type NCK and as input to the Look-Ahead algorithm. The following is an implementation example of the output of the Interpreter.

```
class CRingIR : public CObject { public :  
    int nGCode;          // G-code type (0: G00, 1: G01, 2: G02)  
    CVector Start;      // Start position of block (mm)  
    CVector End;        // End position of block (mm)  
    CVector Cen;        // Center point of arc (mm)  
    double dRadius;     // Radius of arc (mm)  
    double dFeed;        // Feedrate (mm/min)  
    int nStatus;         // Block status (0: start, 1: end)  
    int nStopMode;       // Path control mode. (1: Exact Stop,  
                        // 0: otherwise)  
    int nBlockNumber;    // Block number.  
};
```



# Implementation of the Rough Interpolator

The Rough Interpolator carries out linear interpolation and circular interpolation depending on the G-code type from the interpolator. In Chapter 3, the algorithms for linear interpolation and circular interpolation were described in detail and we implemented the circular interpolator based on the Improved Tustin algorithm, which is better than other algorithms in terms of speed and accuracy.

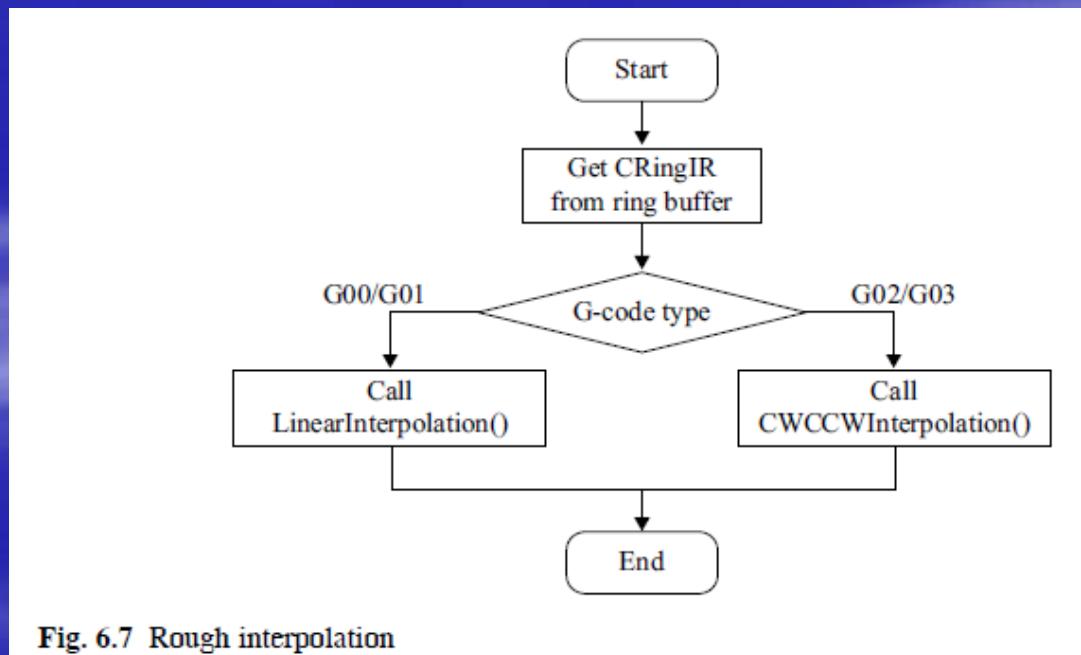


Fig. 6.7 Rough interpolation

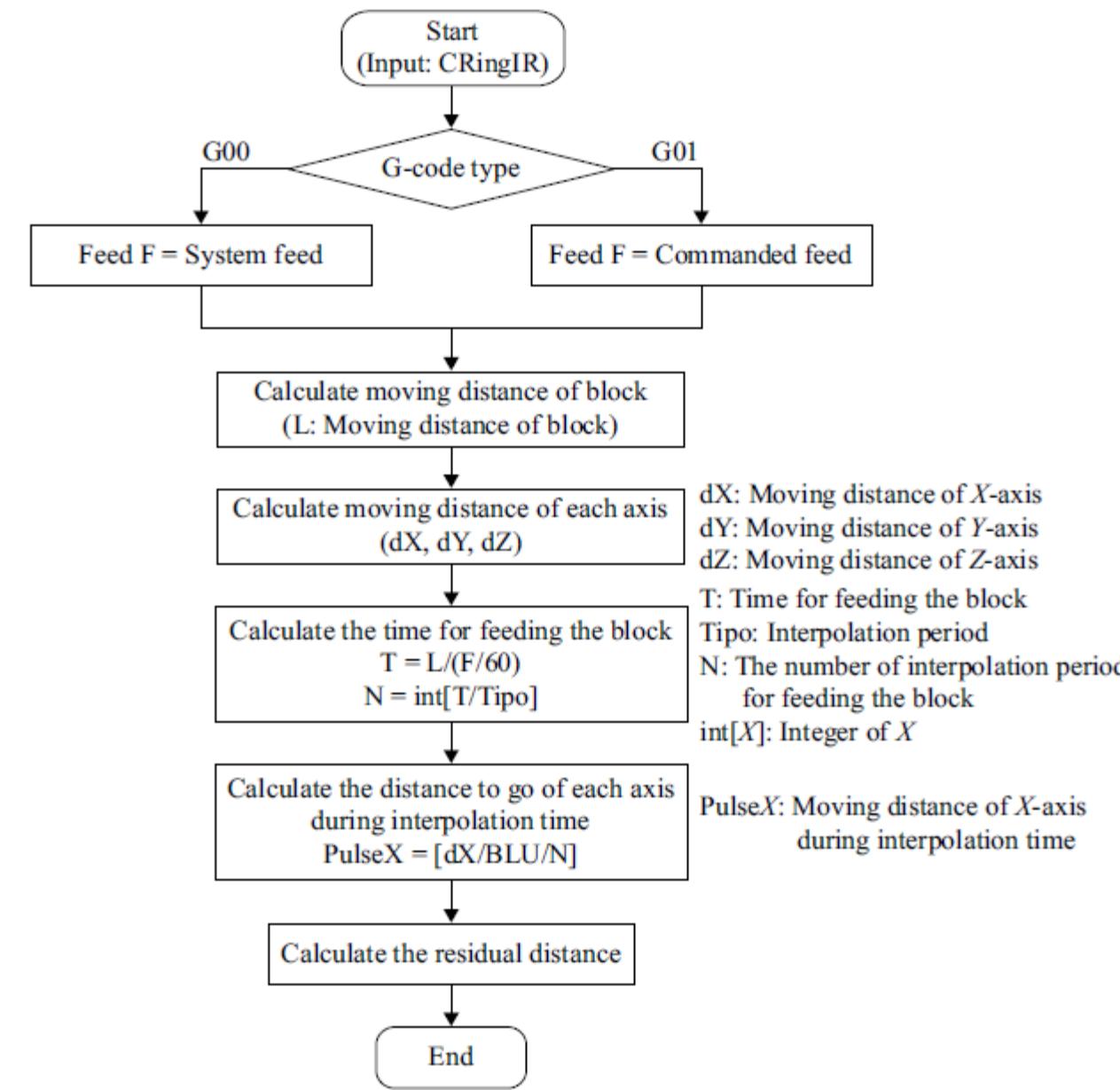


Fig. 6.5 Procedure for executing linear interpolation for X-axis

# Linear Interpolator

- For practical implementation of the interpolator, however, there is something to be considered besides the above-mentioned procedure. In general, the **moving length of an axis is not an exact multiple of pulses**. In this case, numerical error can be accumulated because of the significant figures of numerical computation on a computer and this accumulated numerical error causes reduction of the accuracy of the interpolator.

# Linear Interpolator

- The first method is to move the axis with the remaining pulses during an **additional iteration interpolation time**.
- The second method is to **allocate the remaining pulses to each iteration** interpolation time.
- There is no one correct method to handle the remaining pulses and the developer has to decide on an adequate method depending on the application environment.

## Circular Interpolator

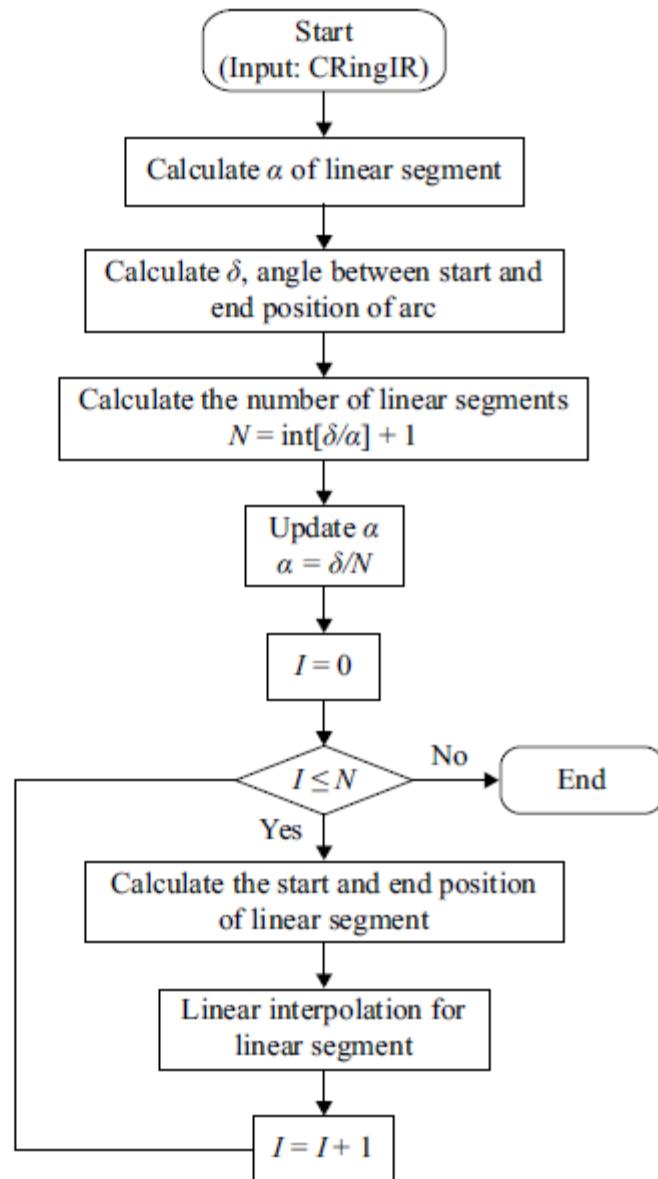
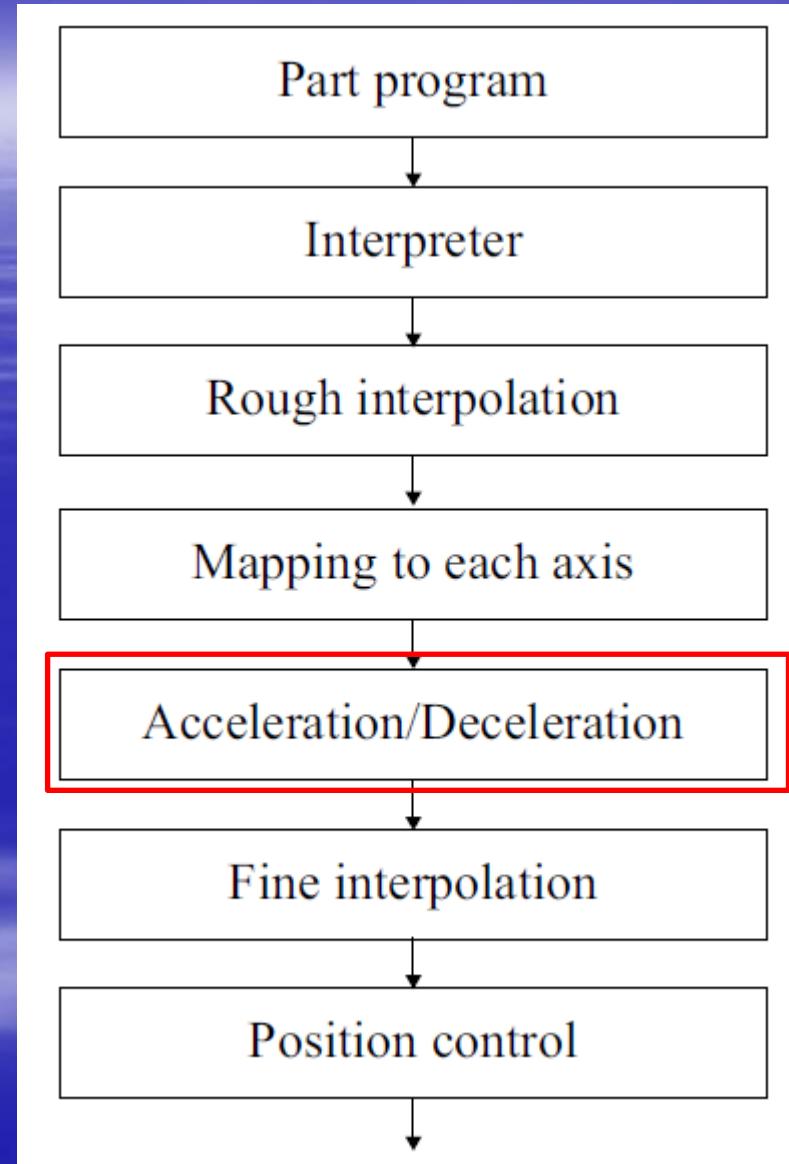


Fig. 6.6 Flowchart of the implemented circular interpolator

## Rough Output

The output from the interpolator consists of the displacement of each axis within the interpolation iteration time (in pulse units) and the number of interpolation sampling times that are required to carry out the block. In addition, the feedrate that is applied for Acc/Dec control and the path control mode are stored as the output. The following is the implemented data structure to store the output from the Rough Interpolator.

```
class CRingRA : public CObject { public :  
    double dFeed; // Feedrate (mm/min)  
    int nStopMode; // Path control mode (1: exact stop,  
                  // 0: otherwise)  
    CVector* P; // Distance to move per interpolation  
                 // cycle in terms of number of pulses.  
    int nStatus; // Block status (0: start, 1: end)  
    int N; // Number of repetitions for interpolation  
           // in executing block.  
};
```



# Implementation of an Acc/Dec Controller

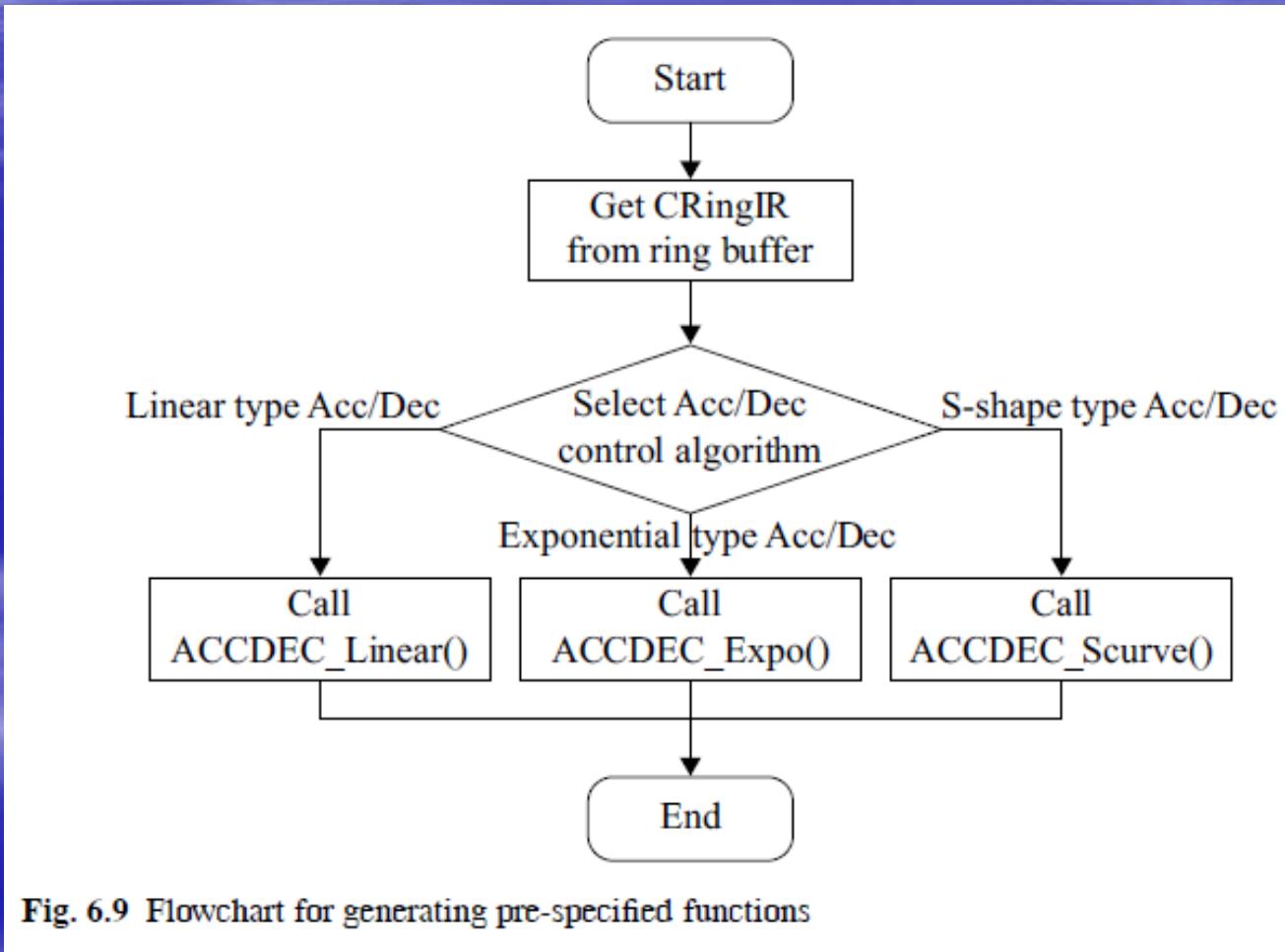


Fig. 6.9 Flowchart for generating pre-specified functions

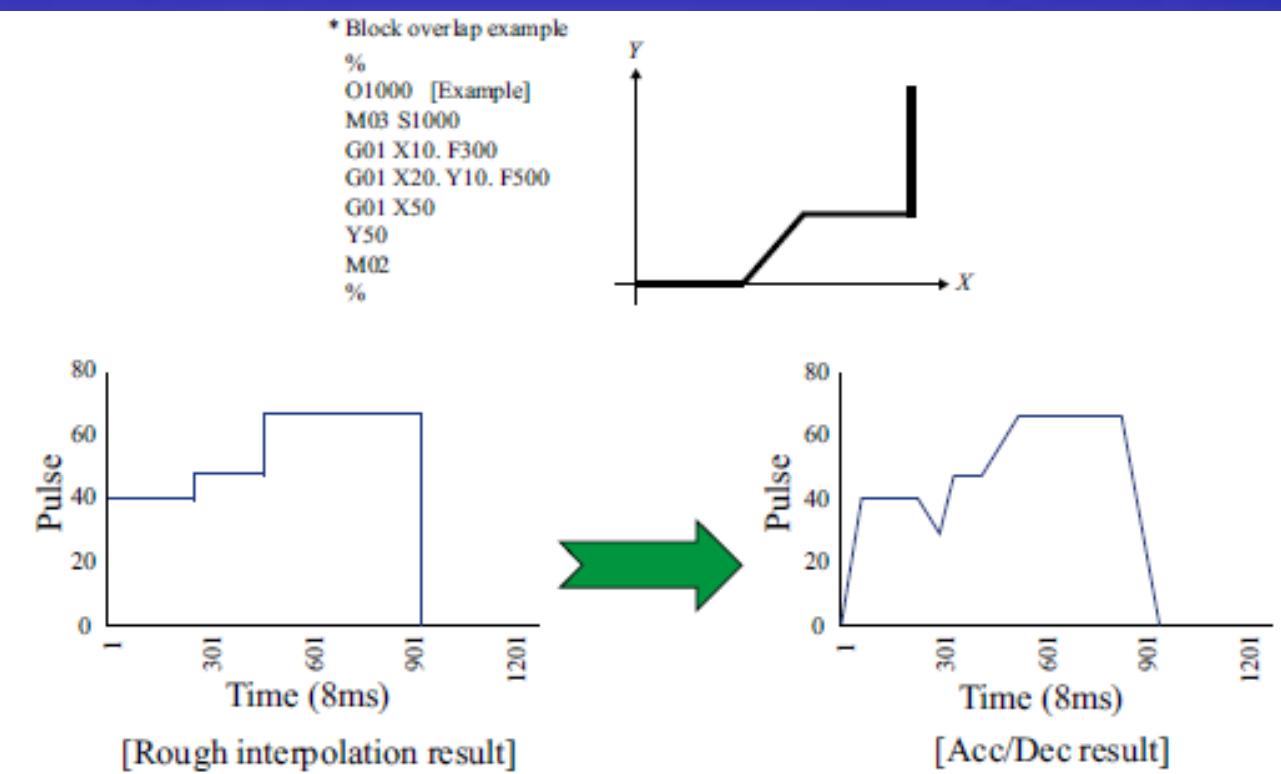


Fig. 6.10 Continuous cutting mode

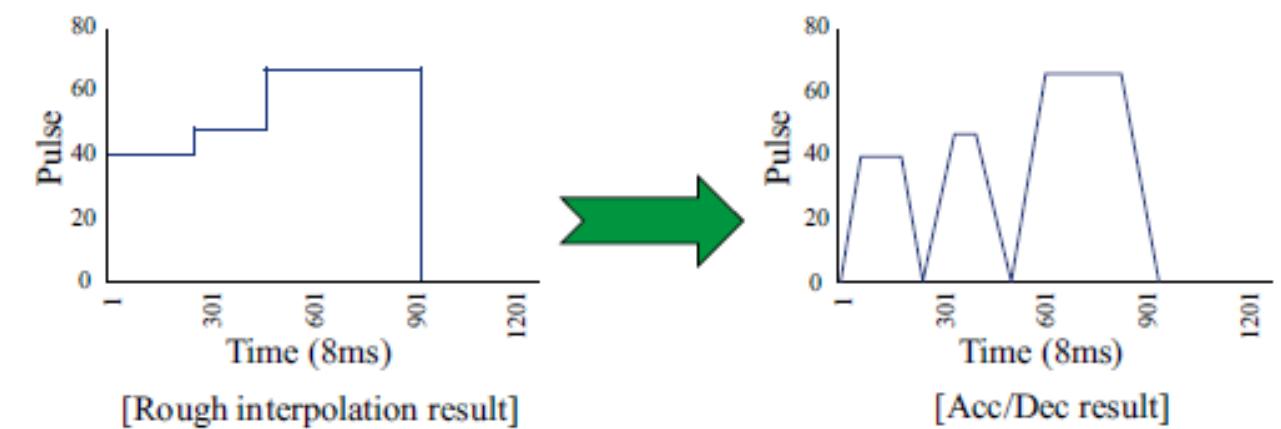


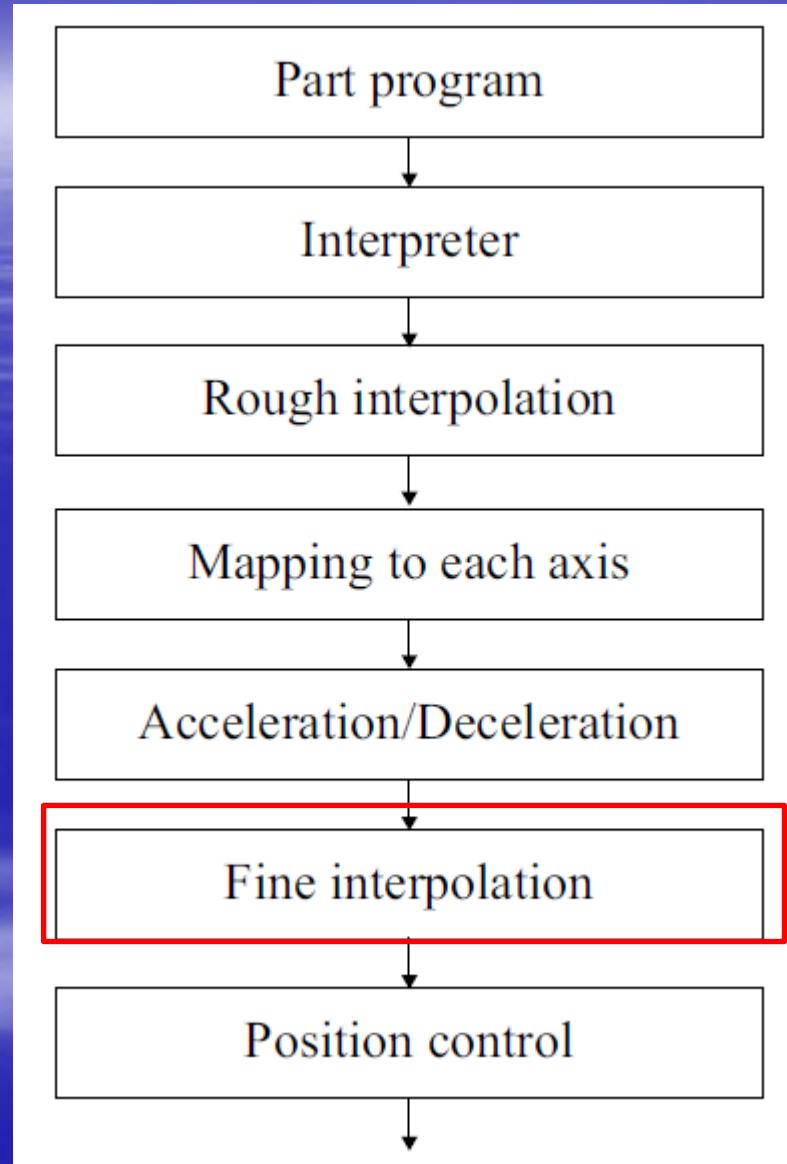
Fig. 6.11 Exact Stop mode

# Implementation of an Acc/Dec Controller

## Output

The Acc/Dec Controller outputs the displacement of each axis every interpolation iteration time (pulse profile) and the interpolation iteration time for each block.

```
class CRingAF : public CObject { public:  
    CVector* P; // Distance to move per interpolation  
                // cycle in terms of number of pulses.  
    int N;     // Number of repetitions for interpolation  
                // in executing block.  
}
```



# Fine Interpolator

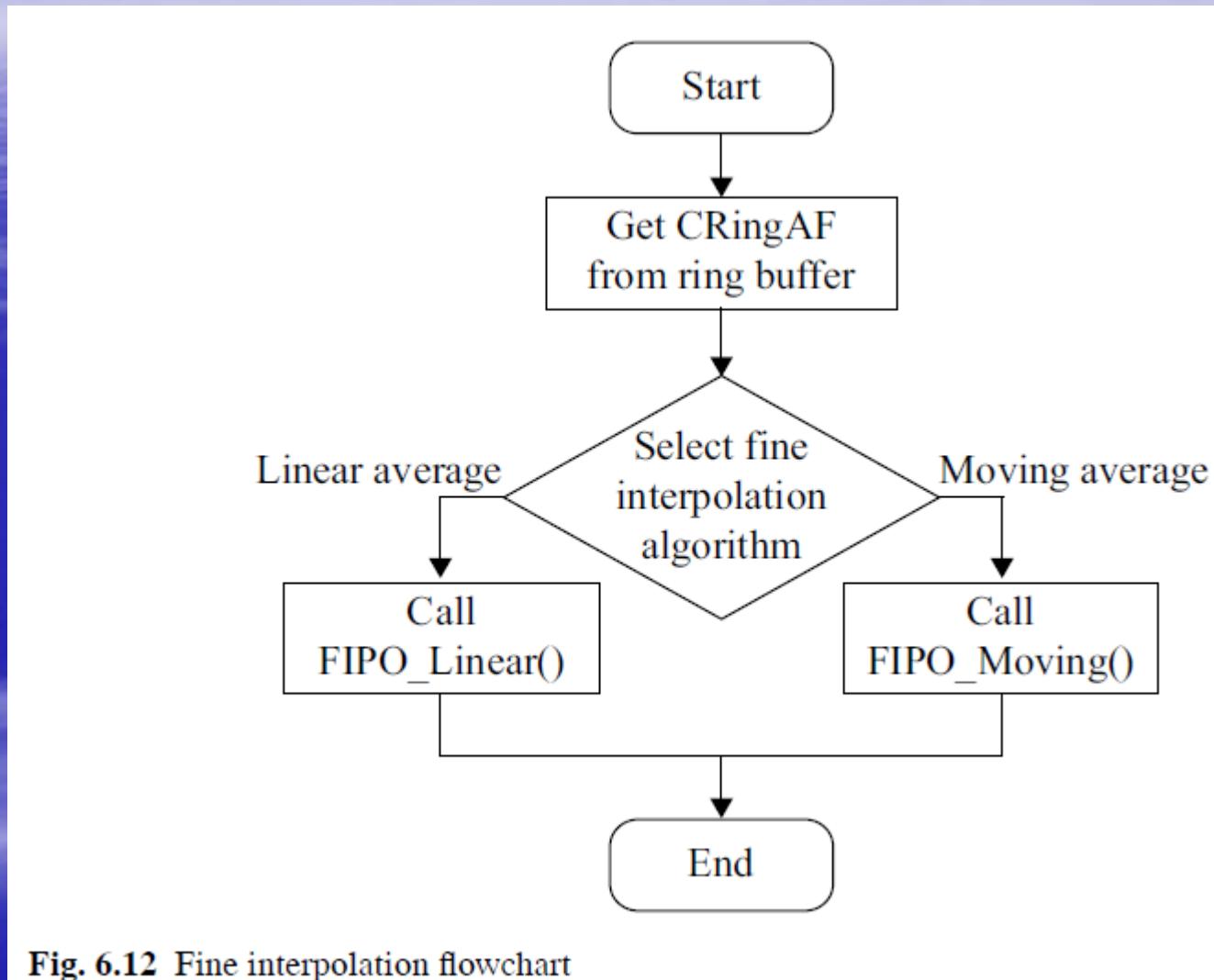


Fig. 6.12 Fine interpolation flowchart

# FIPO Linear()

- The linear method means distributing the displacement of each axis **uniformly** within the interpolation iteration time over the iteration times for position control.

## FIPO Linear()

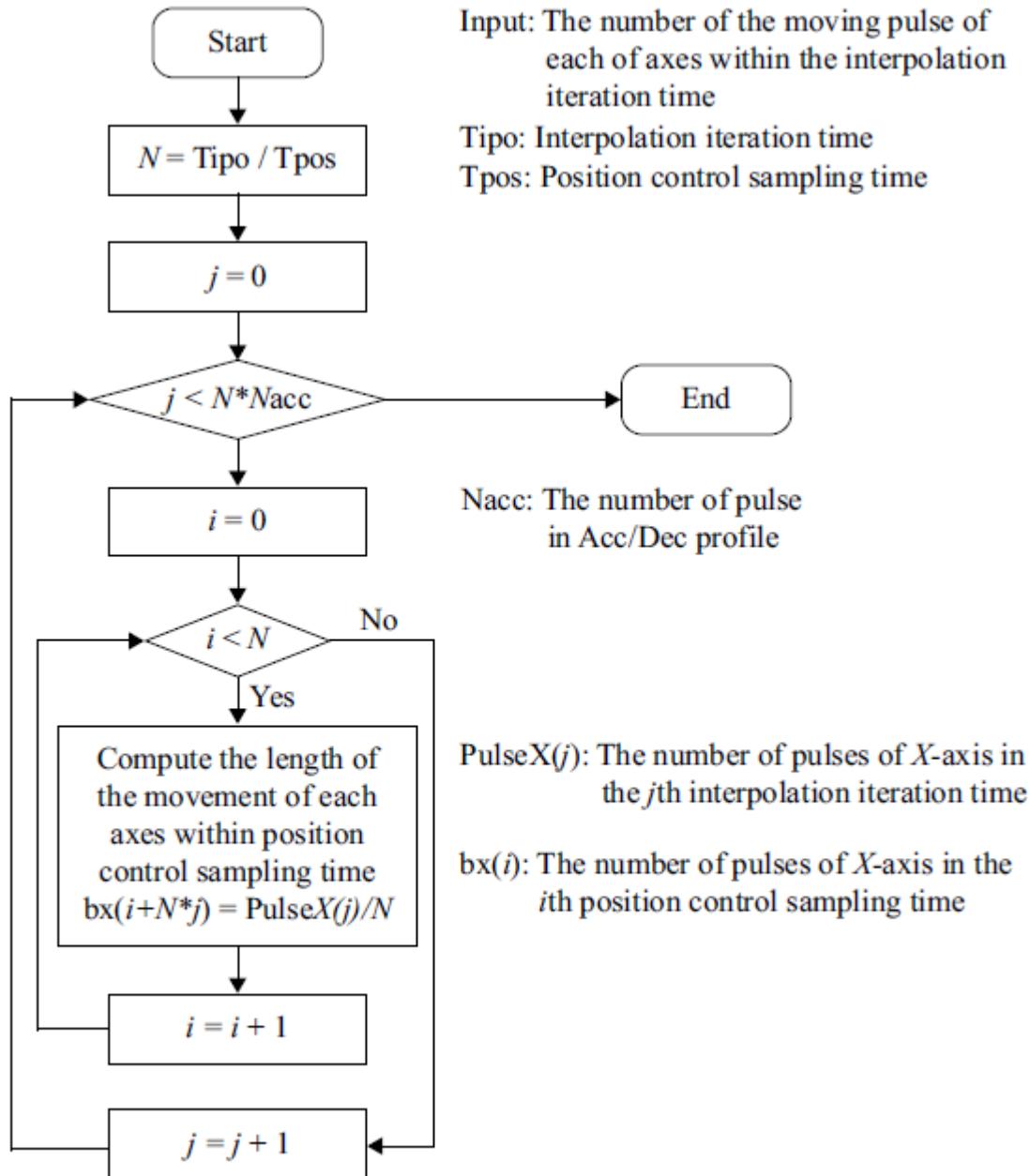
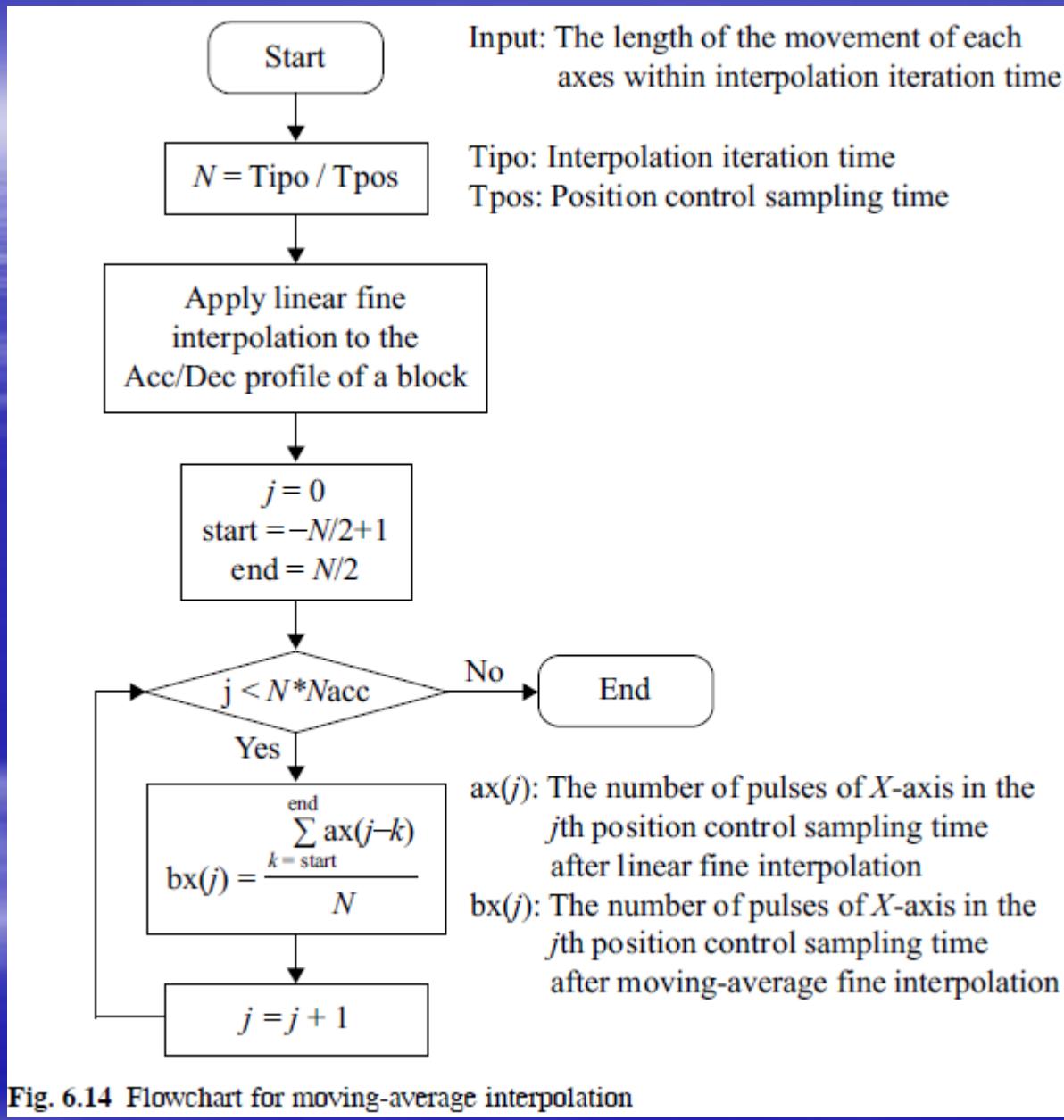


Fig. 6.13 Flowchart for linear fine interpolation

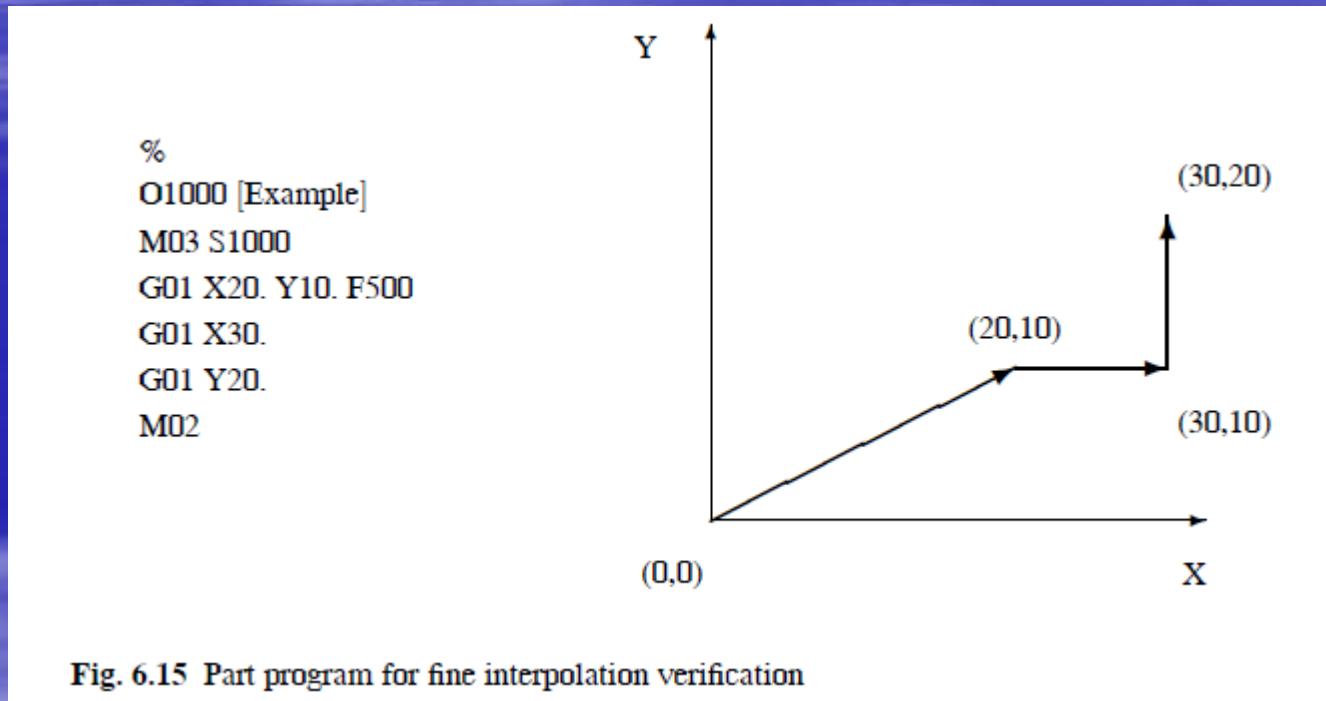
# FIPO Moving()

- In the moving-average method, firstly the number of pulses for every iteration time of position control is calculated by applying the linear fine interpolation to a block. Secondly, the moving average is applied to the **pulse profile**.

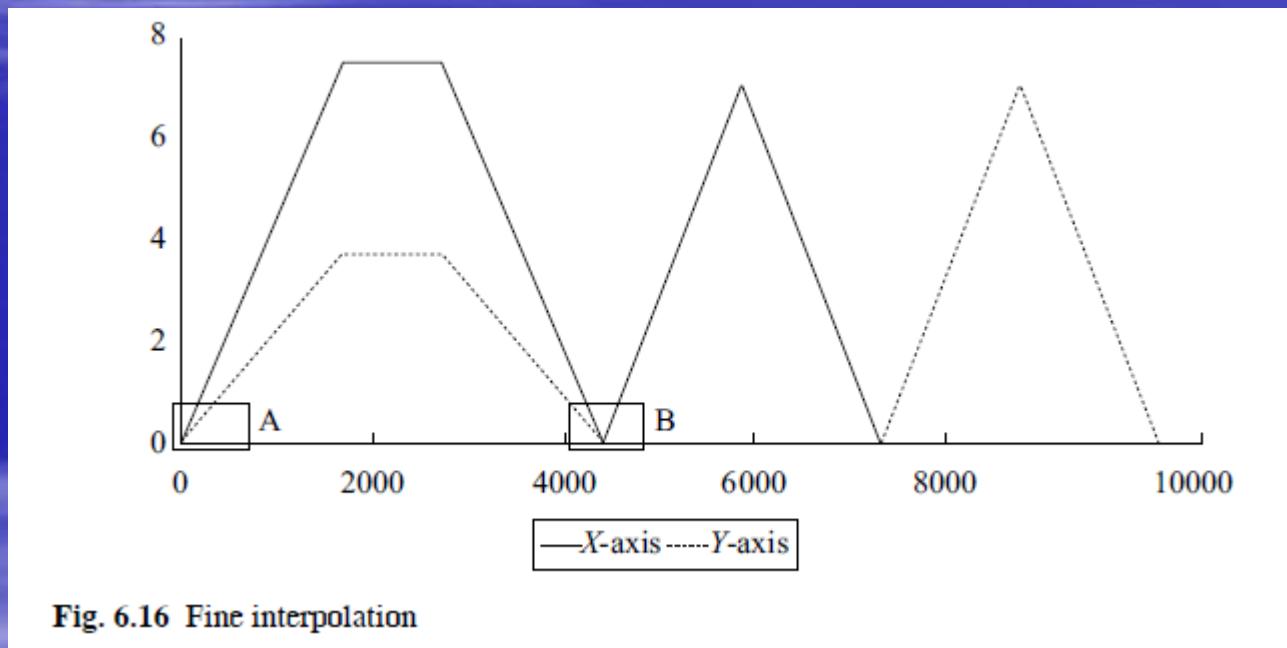
## FIFO Moving()



# Example



# Example



# Example

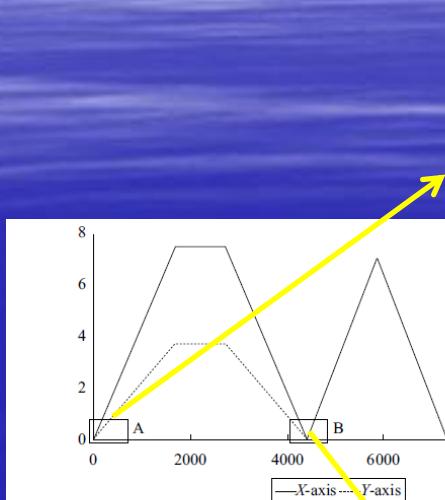


Fig. 6.16 Fine interpolation

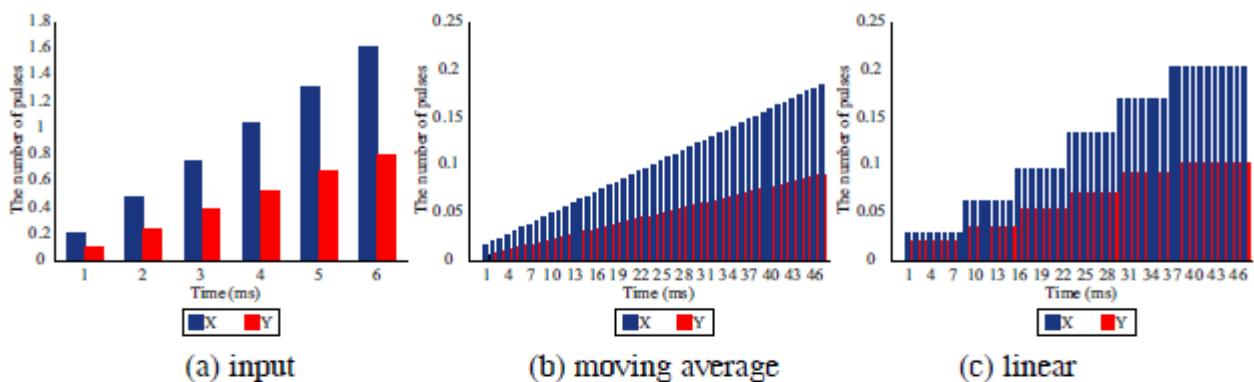


Fig. 6.17 Enlargement of box A from Fig. 6.16

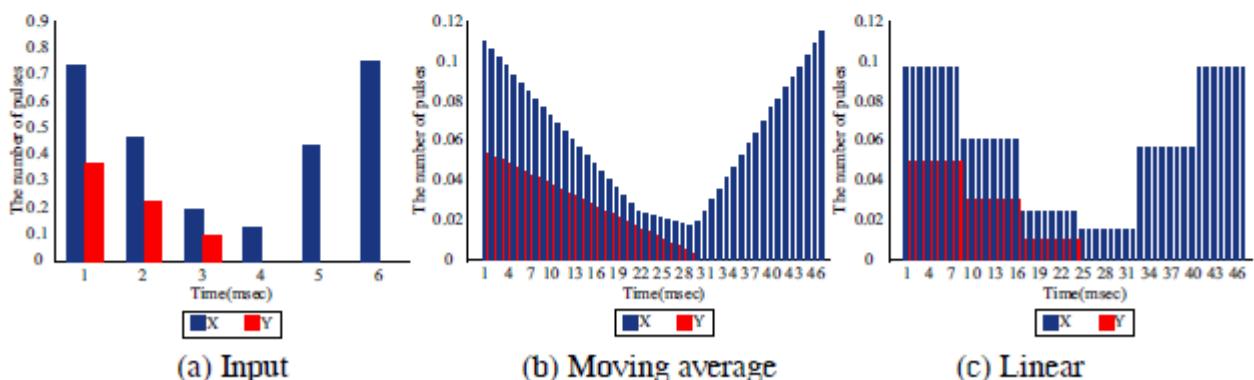


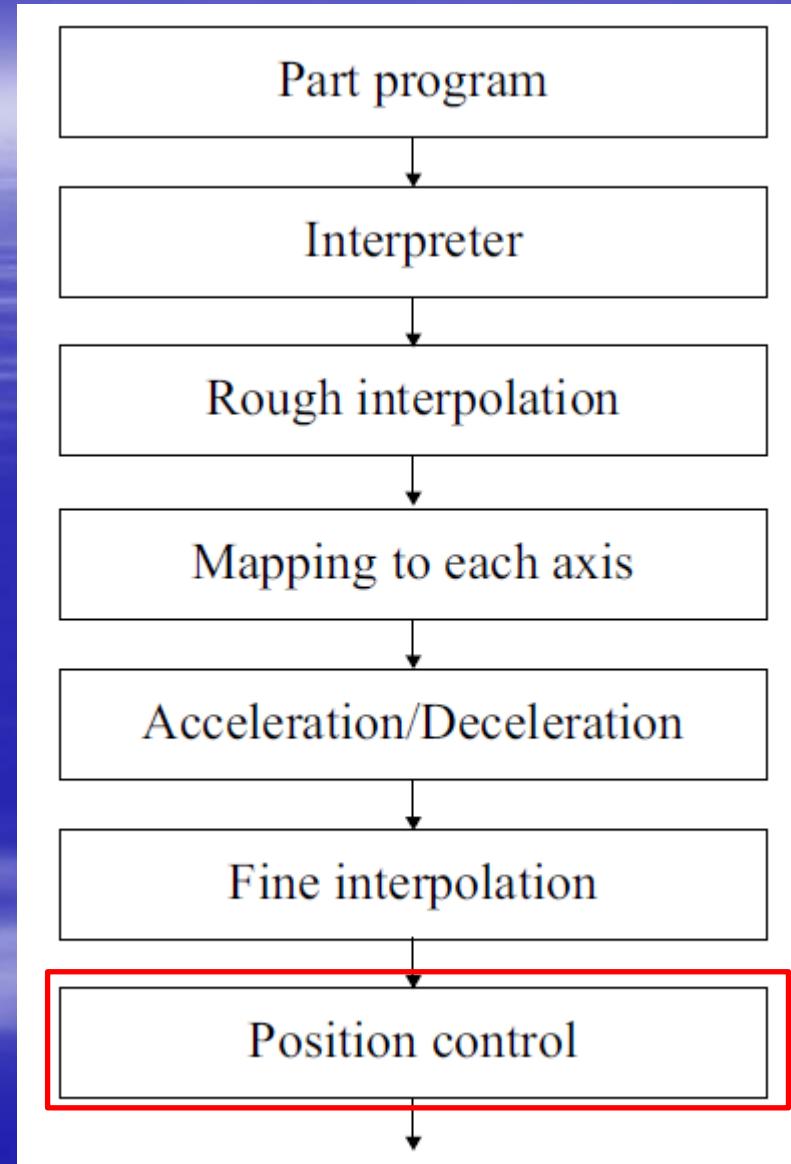
Fig. 6.18 Enlargement of box B from Fig. 6.16

# Fine Interpolator

## Output

The output of the Fine Interpolator is the length through which each axis moves every position control sampling time. Instead of a ring buffer, a global variable, shown below, is used to transmit the data between the Fine Interpolator and the Position Controller. The transmitted data consists of the displacement and direction of the axis movement.

```
CVector P[16]; // Distance to move per interpolation cycle in terms  
// of number of pulses.
```



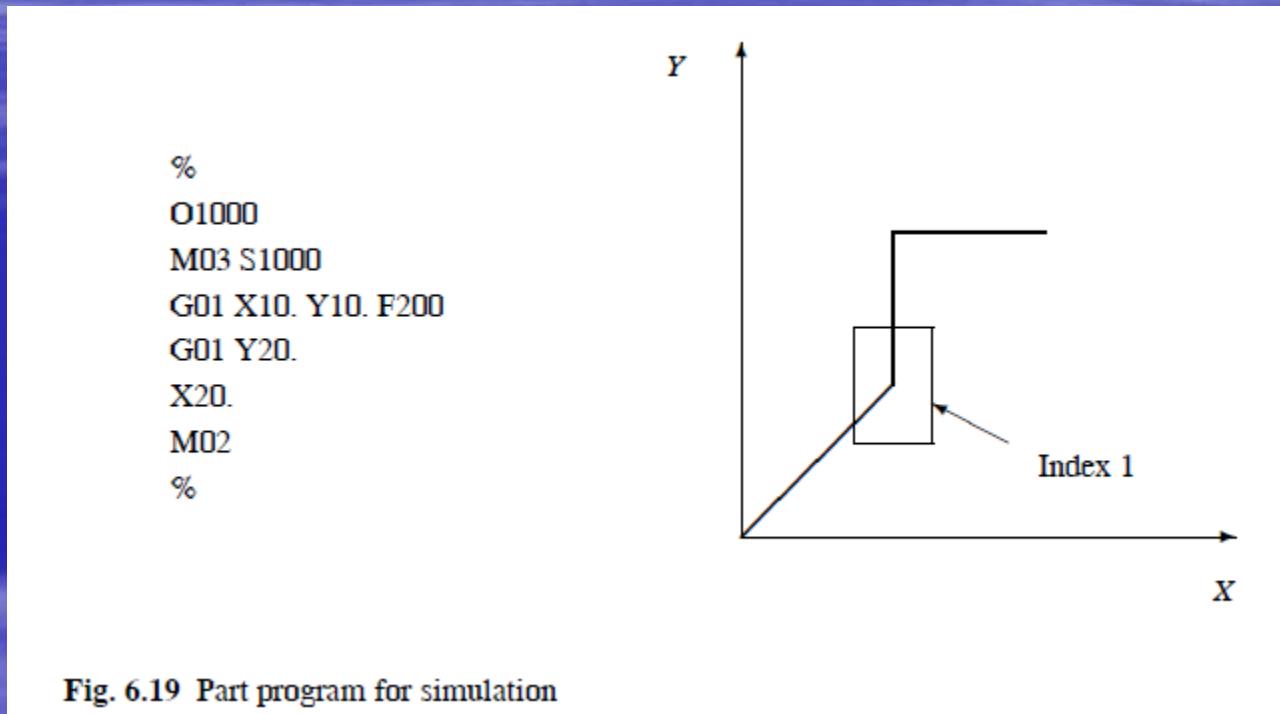
# Position Controller

- The Position Controller is the module that compares the actual position of each axis obtained from an encoder with the commanded position and generates an analog signal (voltage signal) to be sent to each servo drive based on the comparison result. It is iteratively executed every interpolation time of position control

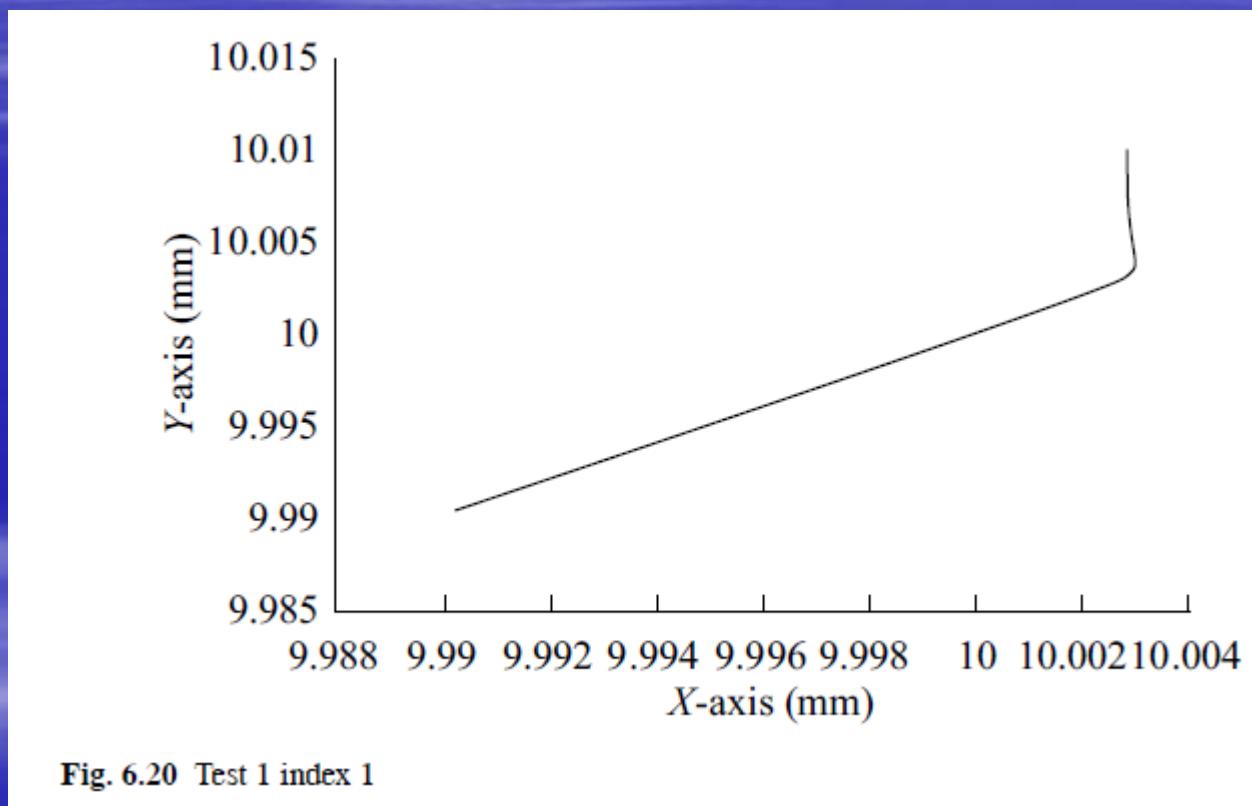
# POS()

- In this function, first the actual position data is taken from encoders and the position error is computed by comparing the actual position from the commanded position. The computed position error is used as the input to the PID control algorithm together with PID gains. The implemented PID algorithm generates the control signal, meaning the velocity of the axes during the iteration time of the position control and this control signal is then converted into a voltage signal that is transmitted to the servo drive.

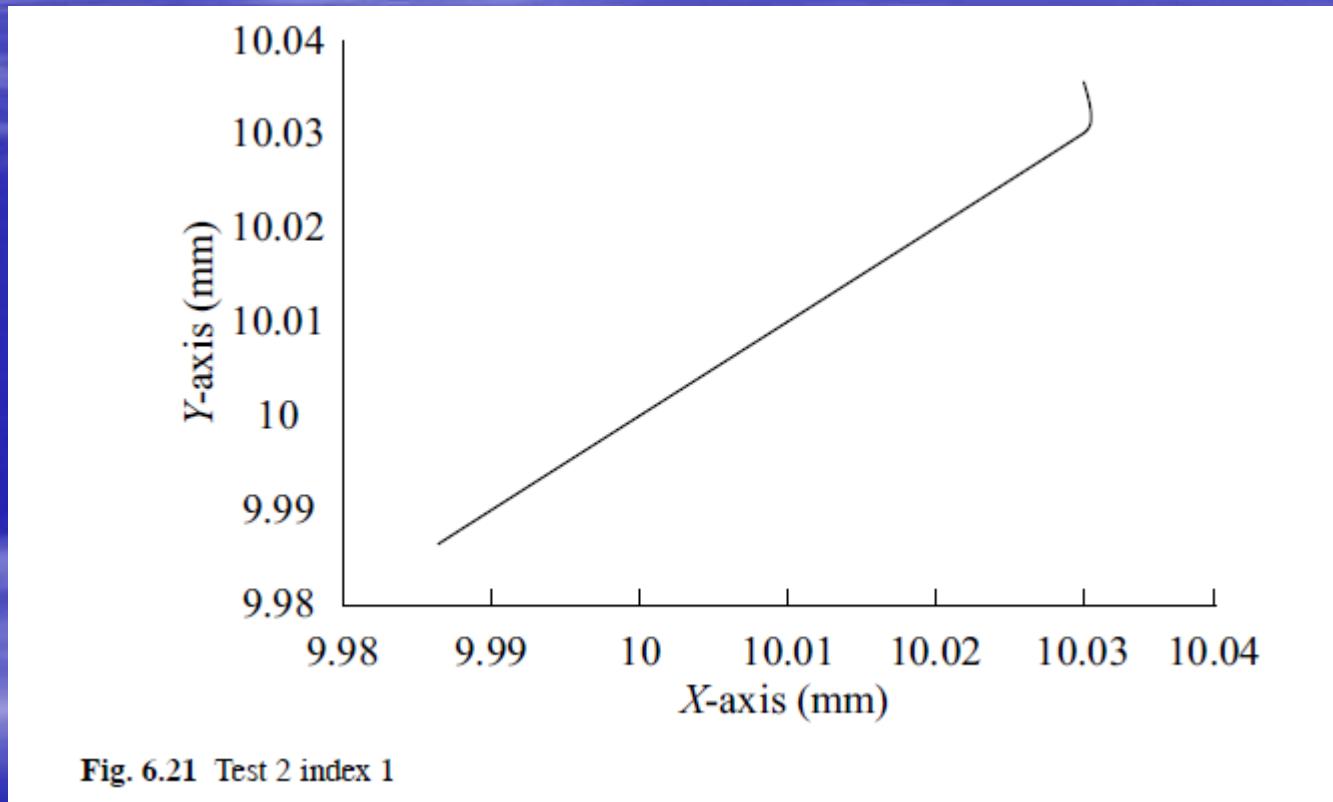
# Example



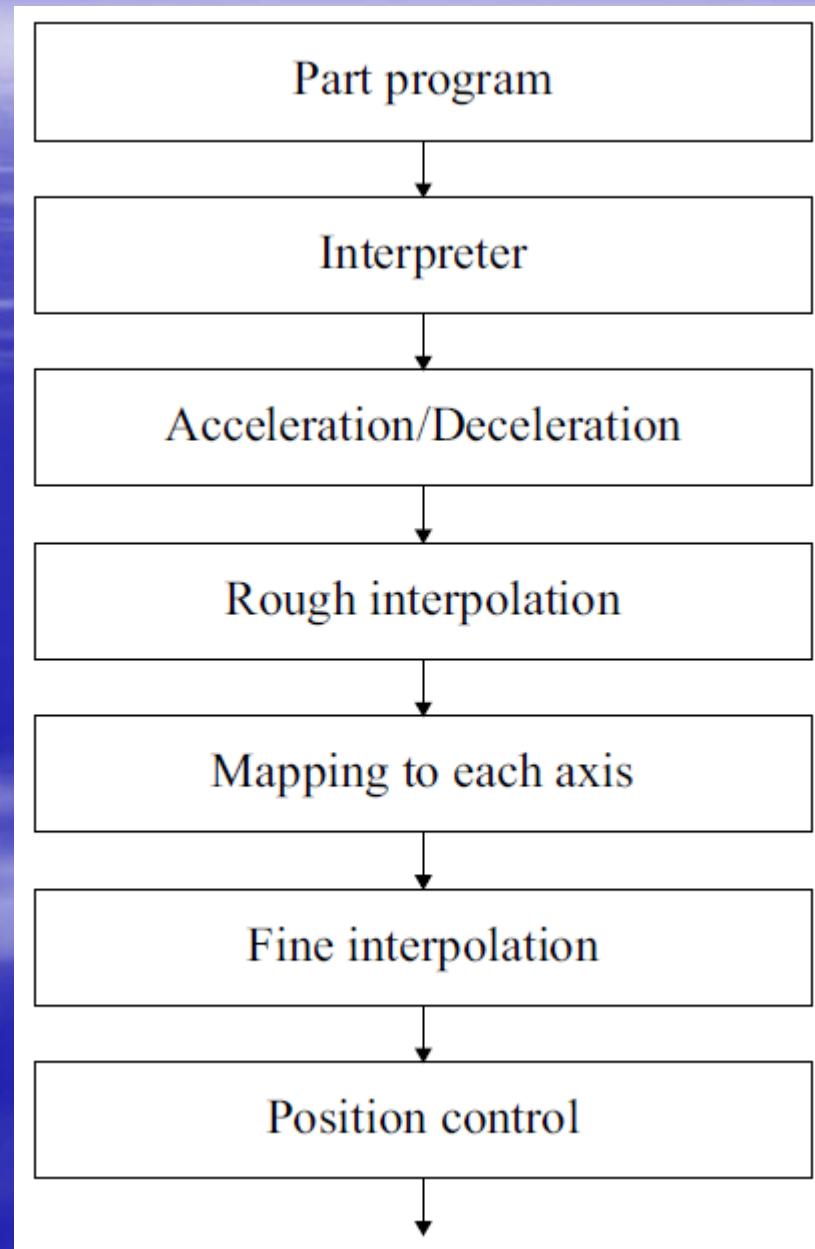
# Example

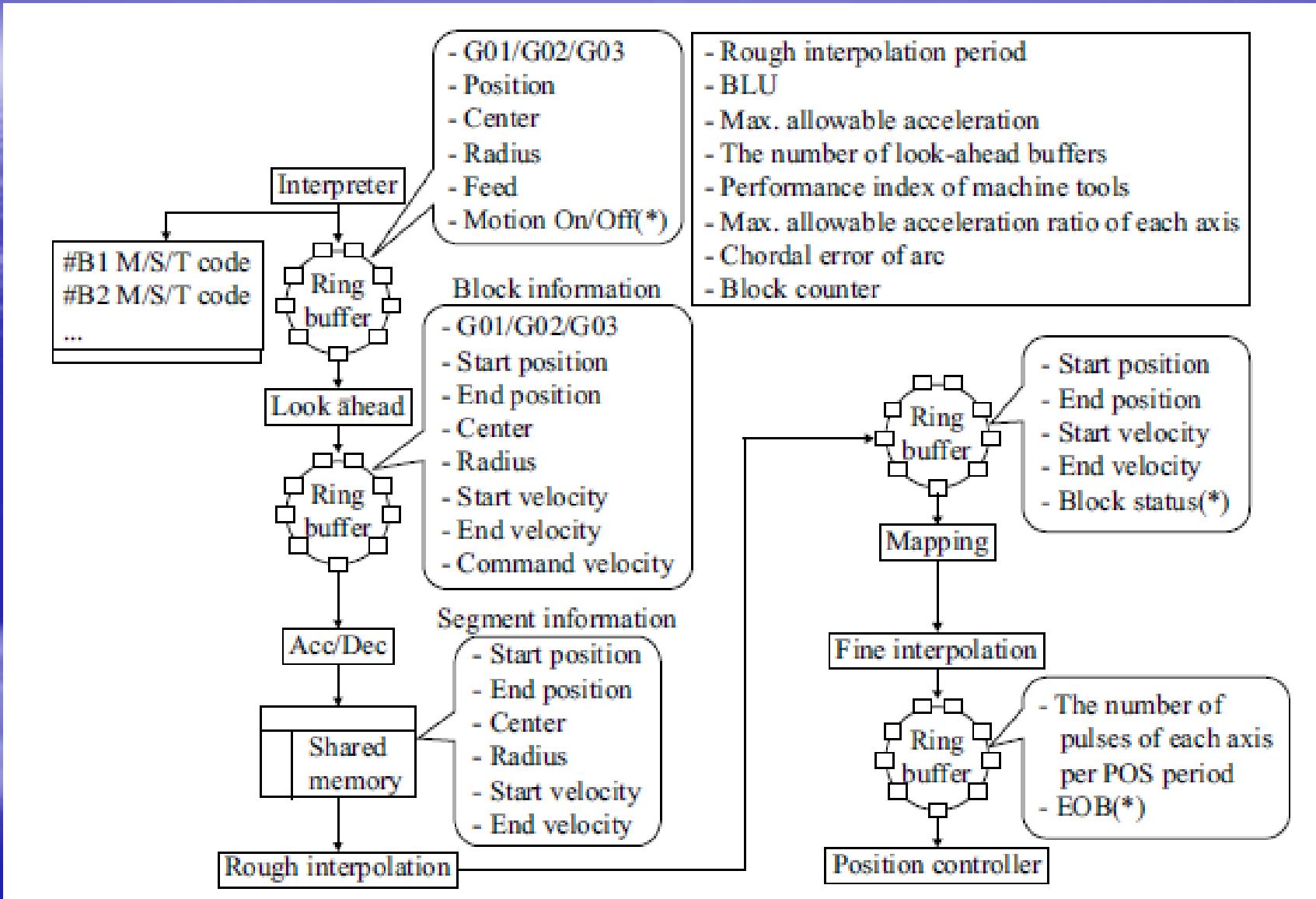


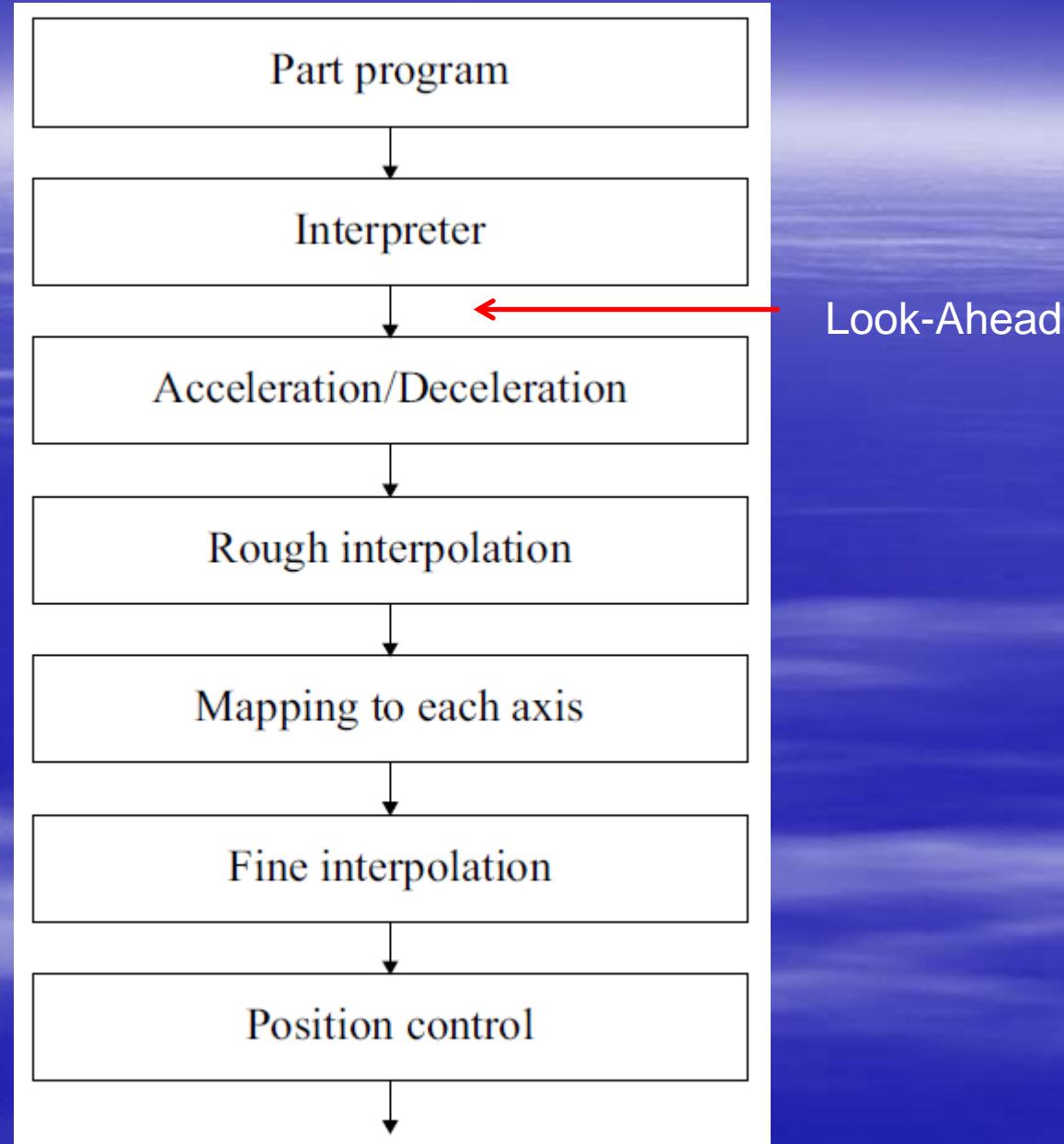
# Example



# ACDBI







# Look-Ahead

- The Look-ahead module calculates the feasible speeds at the beginning and end of the current block by looking ahead at successive blocks

# Look-Ahead

## Input

The output of the Interpreter is input to the Look-ahead module. The following is the data structure that is implemented to store the input of the Look-ahead module. It is used as an element of the ring buffer between the Interpreter and Look-ahead modules.

```
class CRingIR : public CObject { public :  
    int nGCode;          // G-code type, (0: G00, 1: G01, 2: G02)  
    CVector Start;      // Start position of a block (mm)  
    CVector End;        // End position of a block (mm)  
    CVector Cen;        // Center point of arc (mm)  
    double dRadius;     // Radius of arc (mm)  
    double dFeed;        // Feedrate (mm/min)  
    int nStatus;         // Block status (0: start, 1: end)  
    int nStopMode;       // Path control mode. (1: Exact Stop,  
                        // 0: otherwise)  
    int nBlockNumber;    // Block number.  
};
```

# Look-Ahead

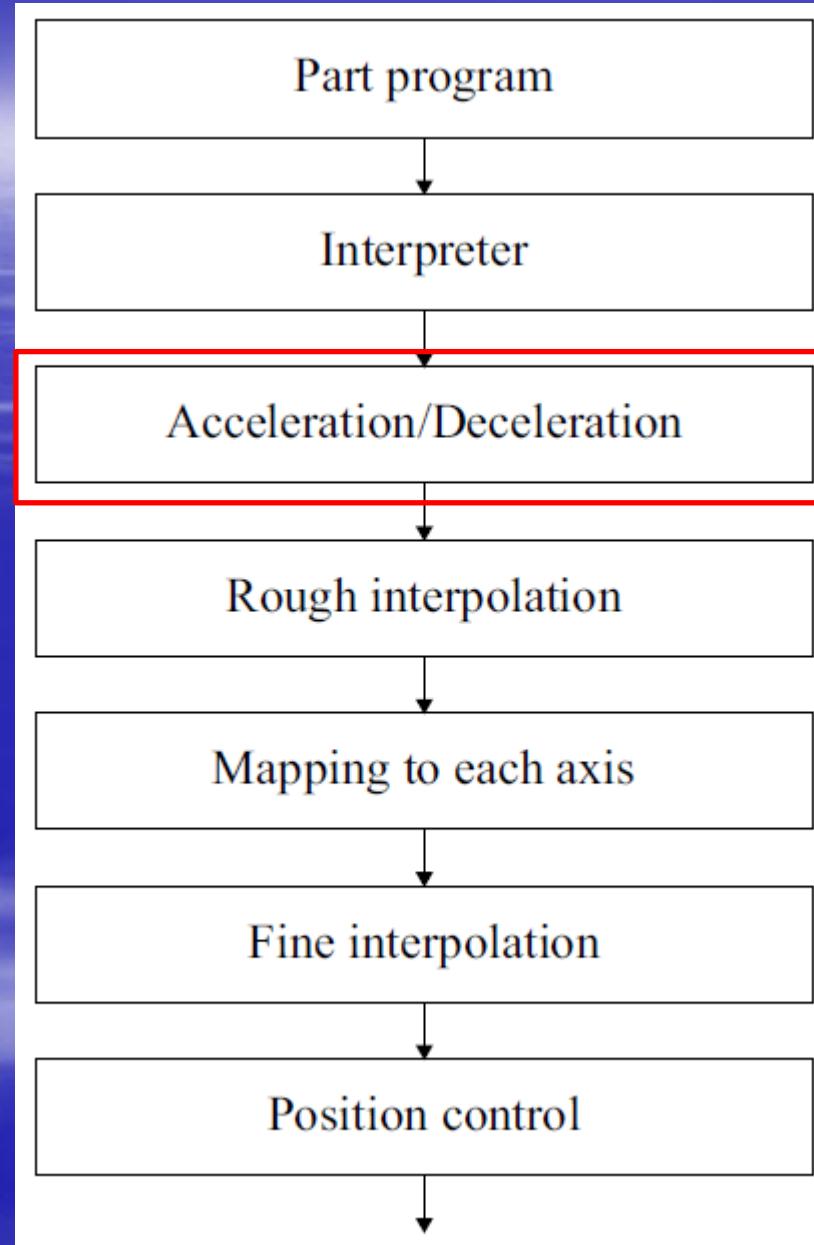
- LookAhead()
- DetermineBlockVelocity()
- DetermineVelocityBetweenLL()
- DetermineVelocityBetweenLC()
- DetermineVelocityBetweenCL()
- DetermineVelocityBetweenCC()

# Look-Ahead

## Output

The Look-ahead module generates the speeds at the beginning and the end of a block as an output. The following is the data structure used to store the output of the Look-ahead module and is used as the element of the ring buffer between the Look-ahead module and the Acc/Dec Controller.

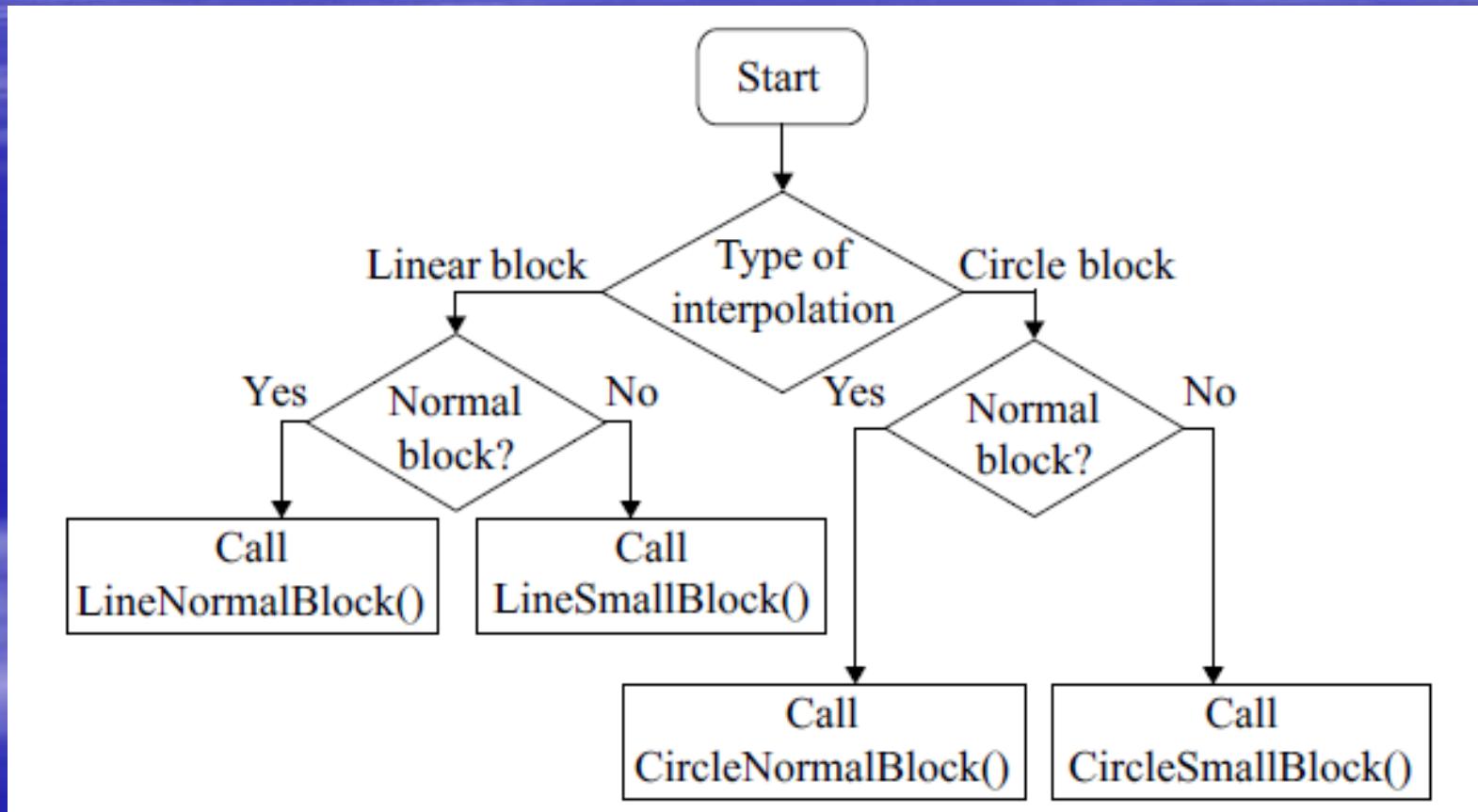
```
class CRBLookaheadBlock : public CObject { public:  
    int nGCode;      // G-code type (0: G00, 1: G01, 2: G02)  
    Cvector vPStart; // Start position of a block (mm)  
    Cvector vPEnd;  // End position of a block (mm)  
    double dVStart; // Speed at the start position (mm/sec)  
    double dVEnd;   // Speed at the end position (mm/sec)  
    double dVComm;  // Speed modified by Look-ahead  
                    // algorithm (mm/sec)  
    CVector vPCenter; // Center position of arc (mm)  
    double dRadius;  // Radius of arc (mm)  
};
```



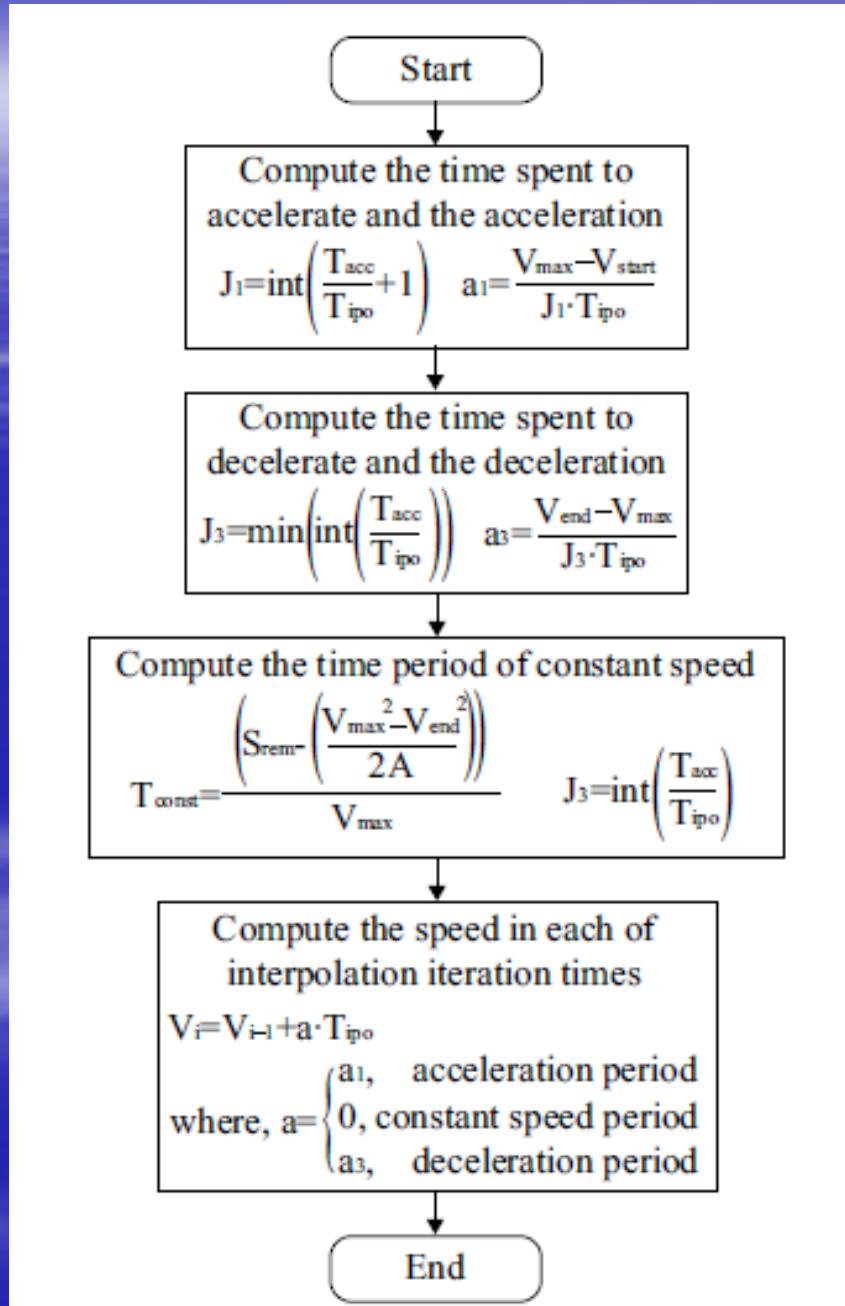
# Acc/Dec Controller

- In an Acc/Dec Controller, whether a block is a normal block or a small block and whether a block is a linear block or an arc block is checked first of all. According to the type of block, a block can be divided into four kinds; a linear normal block, a linear small block, an arc normal block, and an arc small block. The Acc/Dec Controller then calls the appropriate sub-function based on the block type.

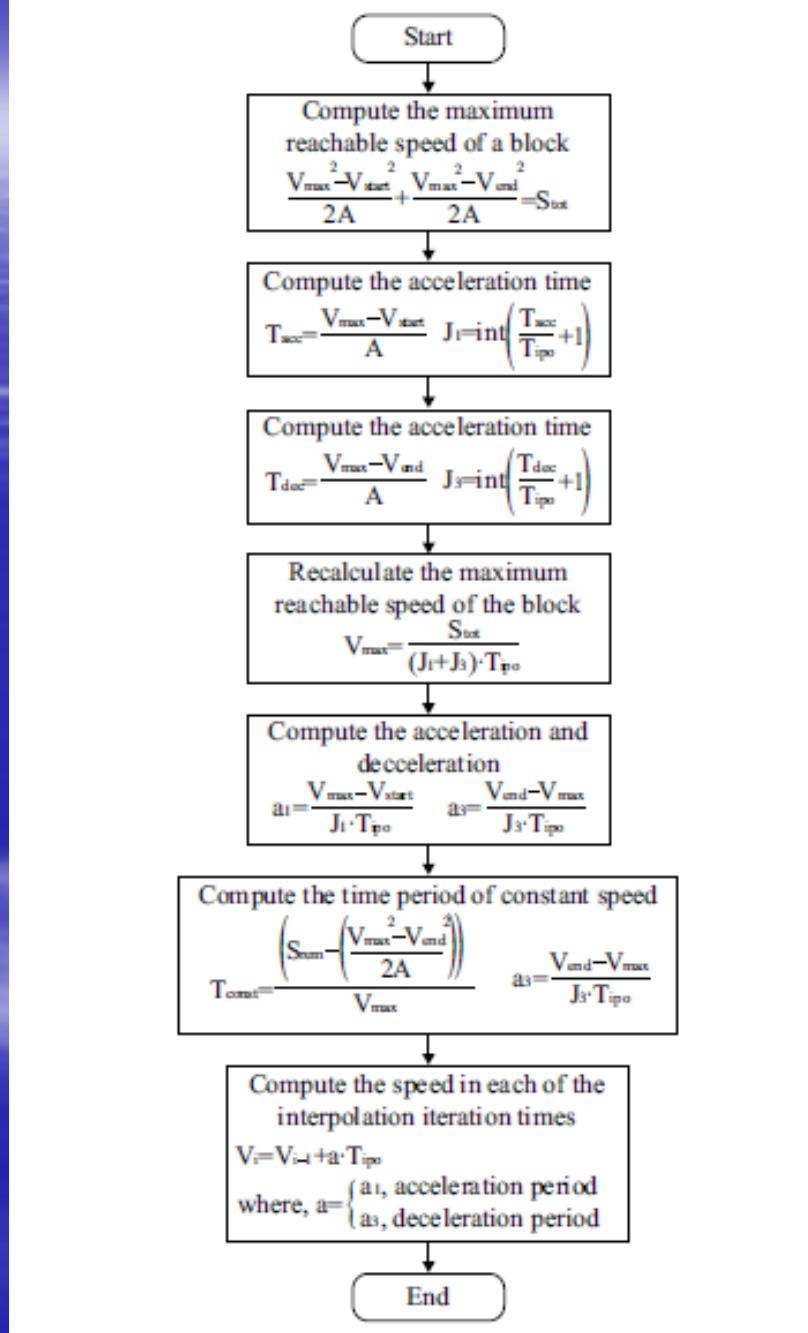
# Acc/Dec Controller



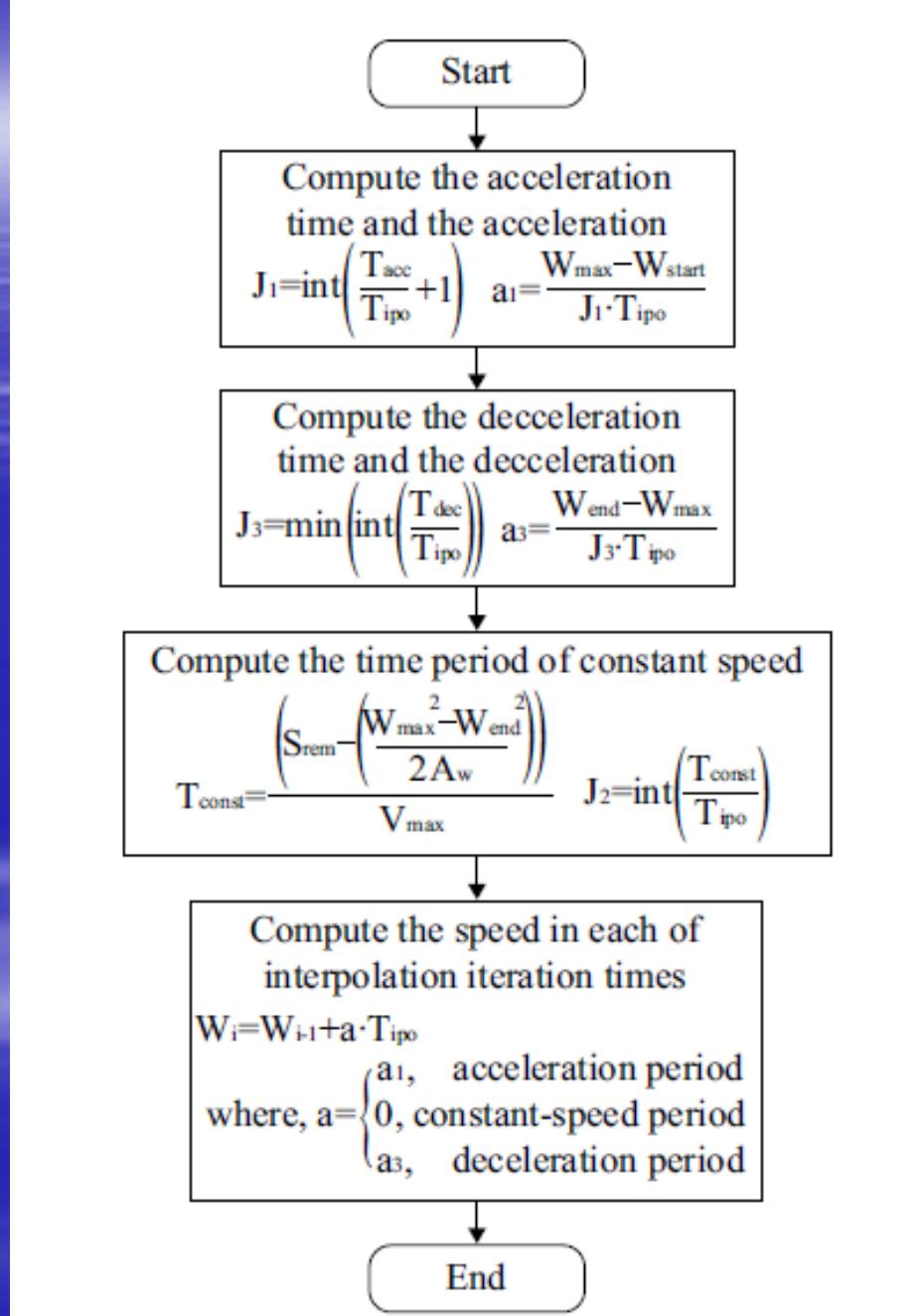
## Flowchart for LineNormalBlock function



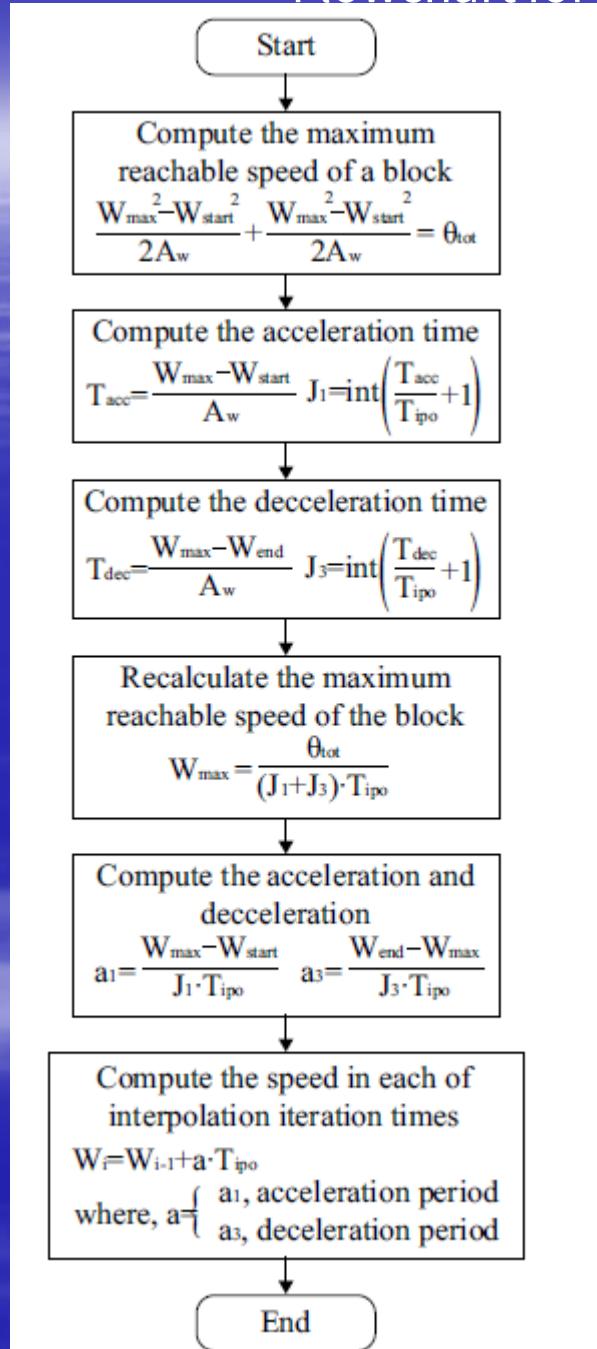
# Flowchart for LineSmallBlock function



# Flowchart for CircleNormalBlock function



# Flowchart for CircleSmallBlock function

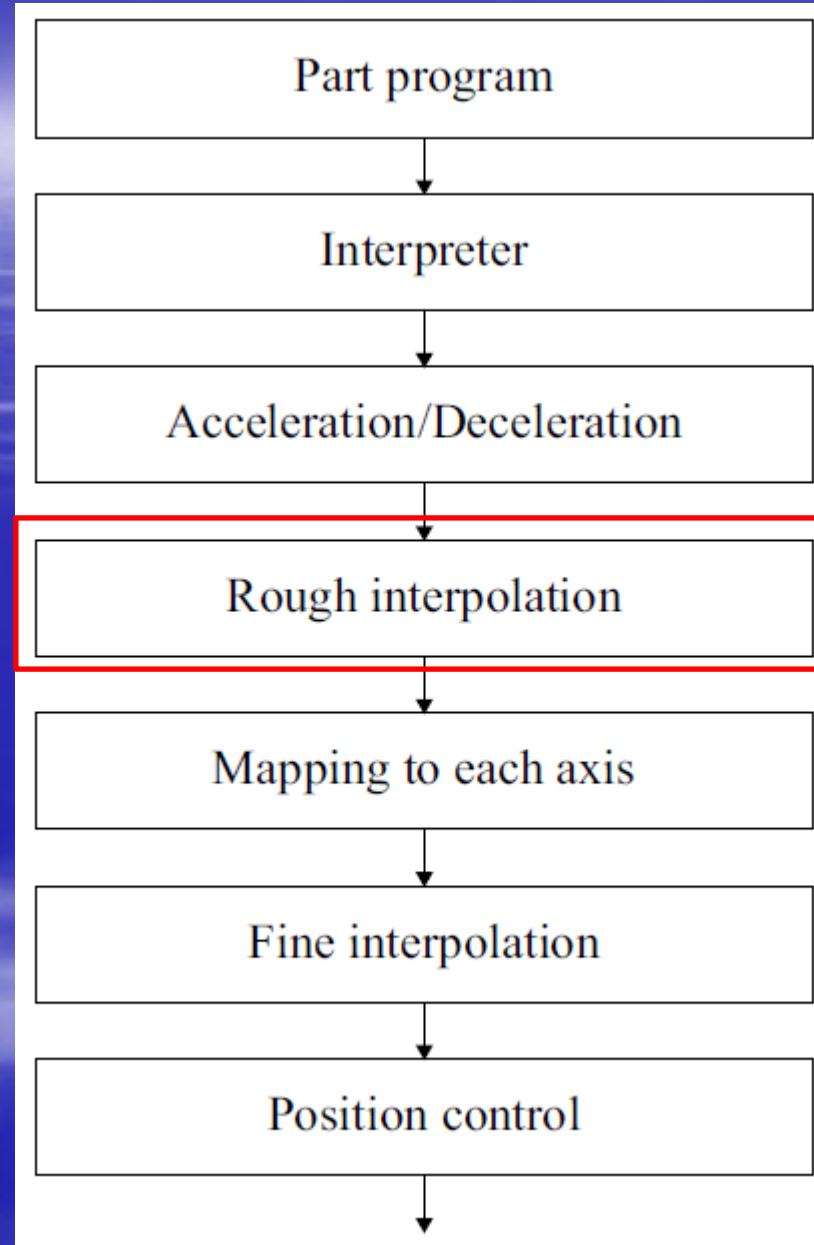


# Acc/Dec Controller

## Output

The Acc/Dec Controller generates the speed for every interpolation iteration time in the case of a linear block and generates the angular speed every interpolation iteration time in the case of an arc block. In addition, it outputs the G-code type, the start position, the end position, and the center position of a block for Rough Interpolation.

```
class CRBAccDecBlock : public CObject { public:  
    int nGCode;      // G-Code type (0: G00, 1: G01, 2: G02)  
    CVector vPStart; // Start position of a block (mm)  
    CVector vPEnd;  // End position of a block (mm)  
    double* dVi;    // Speed during iteration time of inter-  
                    // polation in the case of a linear block.  
    int nBlockN;     // Iteration number of interpolation.  
    double* dAi;    // Angular speed during iteration time of  
                    // interpolation in the case of an arc  
                    // block.  
    double dError;   // Remaining distance.  
    double dRadius;  // Radius of an arc (mm).  
    double dTheta;   // Remaining angle of arc.  
    CVector vPCenter; // Center position of an arc (mm).  
};
```



# Rough Interpolator

- The Rough Interpolator of an ADCBI-type NCK calculates the position at which the tool should arrive for every interpolation iteration time based on the velocity profile from the Acc/Dec Controller

# Rough Interpolator

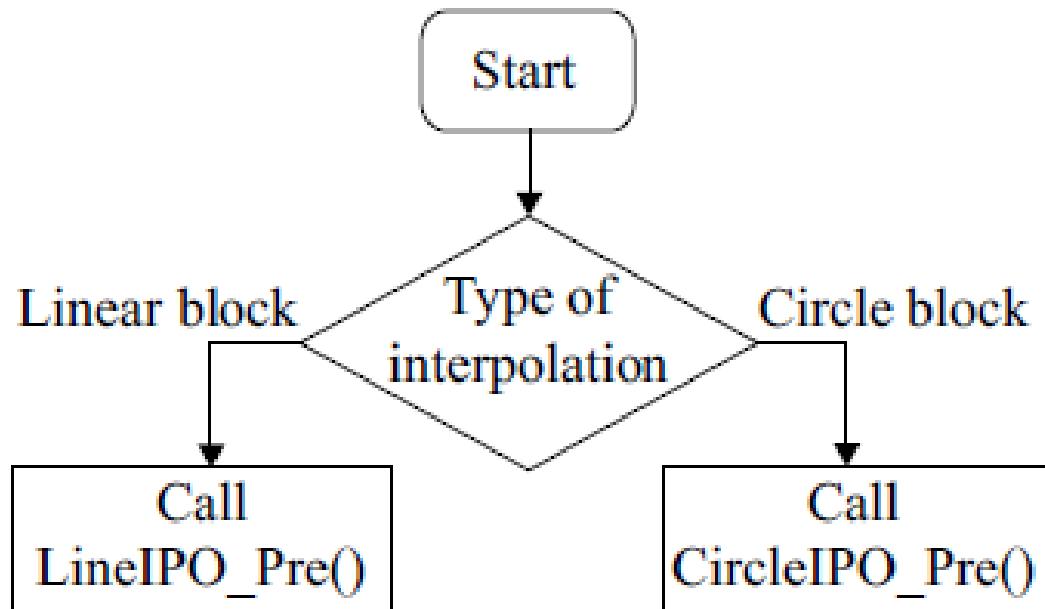
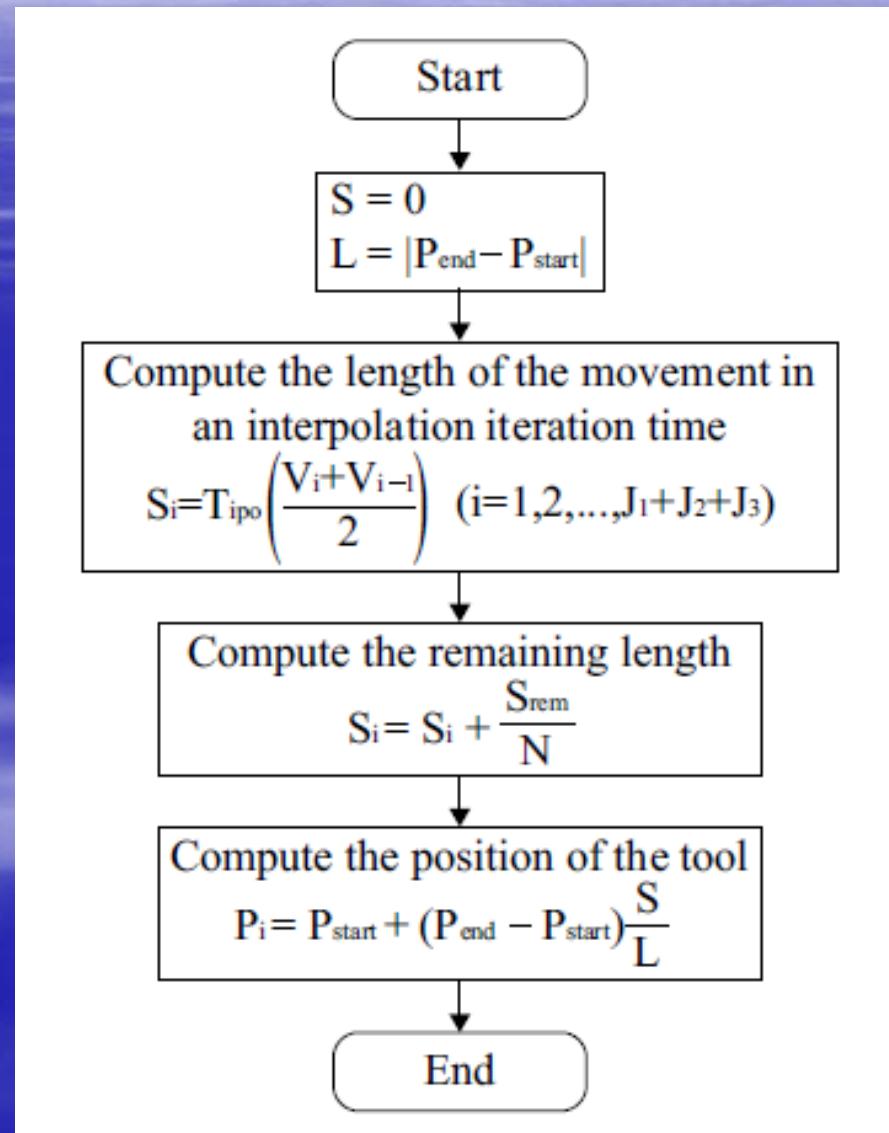
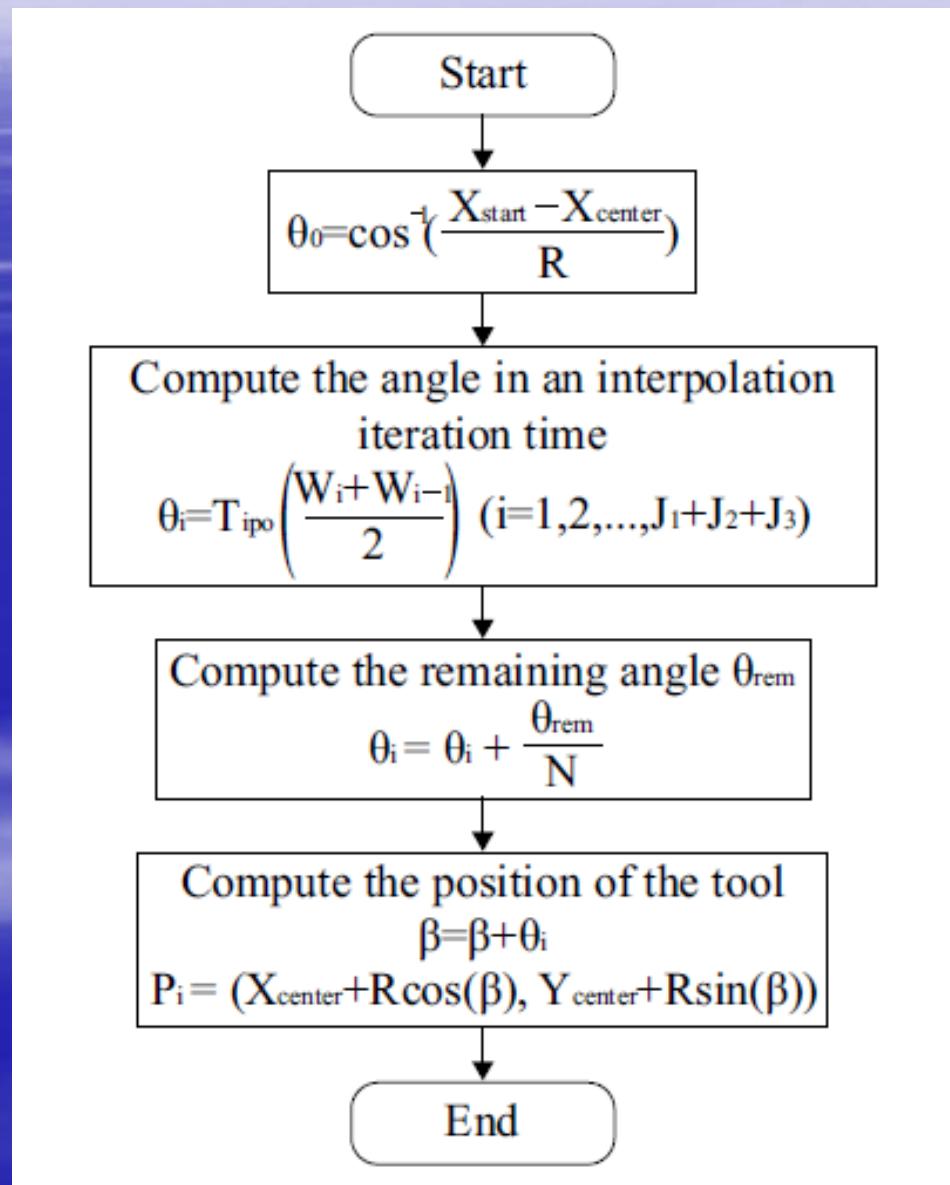


Fig. 6.29 Flowchart for the RoughInterpolation function

## Flowchart for LinearIPO Pre function



## Flowchart for CircularIPO Pre function

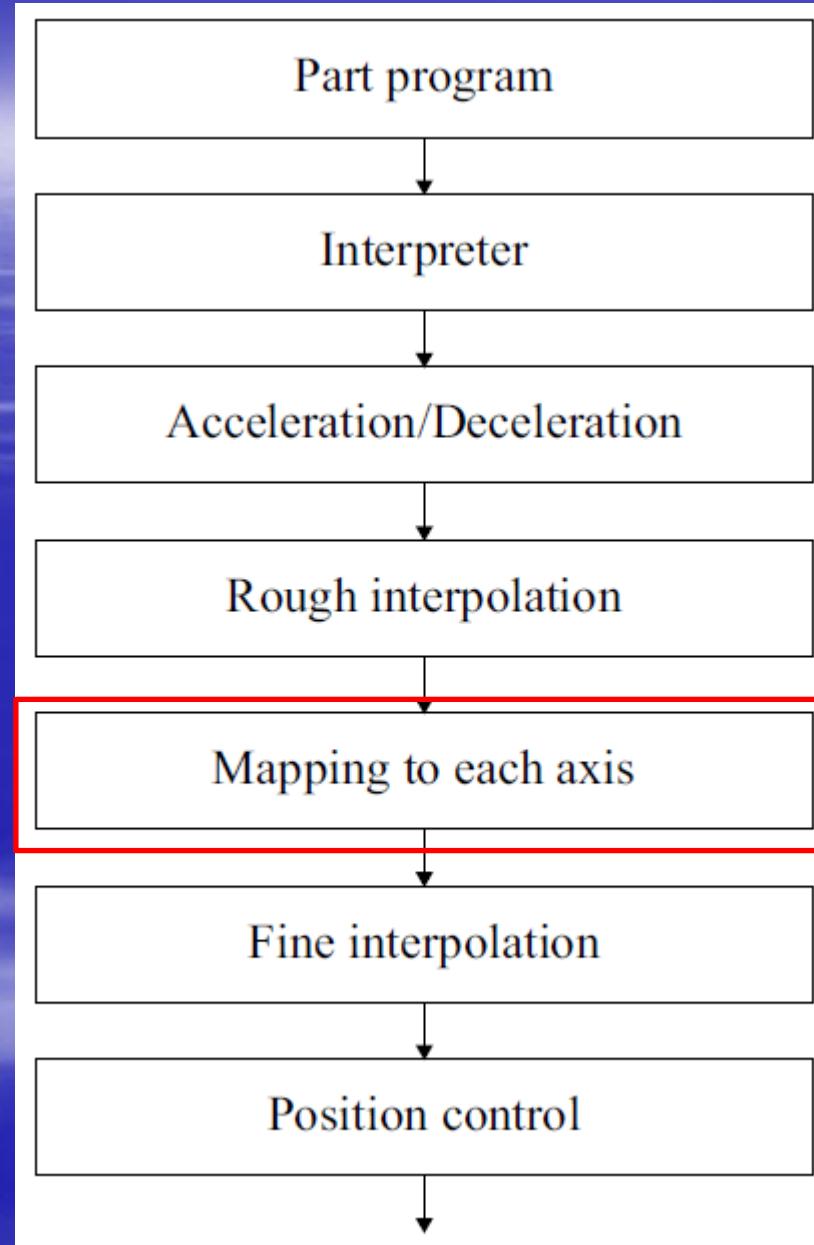


# Rough Interpolator

## Output

The Rough Interpolator generates and stores the position at which a tool should arrive for every interpolation iteration time. The following is the implemented data structure for storing the output from the Rough Interpolator.

```
class CRBRoughIPOBlock : public CObject { public:  
    CVector* vPi; // Position at which a tool should arrive  
                  // each iteration time of interpolation.  
    int nBlockN; // Iteration number of interpolation.  
};
```



# Mapping Module

- The Mapping module divides the tool movement over the interpolation iteration time into displacements of each axis, being the input to the Fine Interpolator. In the case of ADCBI-type NCK, because Acc/Dec control and rough interpolation are applied to the tangential direction of tool movement, unlike ADCAI-type NCK, it is necessary to divide the interpolated point into displacements for each axis. In general, the Mapping function can be included in the Rough Interpolator. However, in this textbook, the Mapping function is implemented as a separate module in order to show the execution procedure of NCK.

# Mapping Module

## Mapping output

The Mapping module generates and stores the displacement of each axis every interpolation iteration time. The stored output is used as input to the Fine Interpolator. The following is the implemented data structure to save the output of the Mapping module.

```
class CRingAF : public CObject { public:  
    Cvector* P; // Distance to move per interpolation  
                // cycle in terms of number of pulses.  
    int N;     // Number of repetitions for interpolation  
                // in executing block.  
};
```

# Mapping Module

- An ACDAI-type NCK consists of an Interpreter that converts a programmed block into the actual toolpath, a Rough Interpolator that divides the toolpath into linear segments, an Acc/Dec Controller that smoothes drastic changes in axis speed, a Fine Interpolator that divides the linear segments over one interpolation iteration time into linear segments over iteration times for position control, and a Position Controller that computes the displacement of each axis based on the position error. These modules are executed sequentially and the data between the modules is transferred via ring buffers.

# Summary

- The difference between an ADCBI-type NCK and an ADCAI-type NCK is the execution sequence of the Acc/Dec Controller and the Rough Interpolator.
- In ADCBI-type NCK, the speed profile along the tangential direction of tool movement is generated before rough interpolation. Therefore, the functions and algorithms for rough interpolation and Acc/Dec control are different from those of an ADCAI-type NCK.

# Theory and Design of CNC Systems

*Chapter 7*  
**Programmable Logic Control**

# Introduction

A software-based PLC system has the following advantages compared with a hardware-based sequence control system.

- Flexibility: the modification of control logic is possible by changing a PLC program.
- Scalability: the extension of a system is easily possible by adding logic and then changing PLC programs.
- Economic efficiency: the cost decreases through reduction of design, high reliability, and convenience of maintenance.
- Miniaturization: the physical volume required for installation is small compared with hardware control systems.
- Reliability: the probability on breaking down due to poor contact decreases because the system consists of a semiconductor having a non-contact switch.
- Performance: advanced control functions such as data handling and arithmetic calculation are enabled.

# Introduction

Table 7.1 Comparison between hardwired sequence controller and PLC

Type	Sequence control	PLC
Logic type	Hard Logic	Soft Logic
Supported functions	Relay, Timer, Preset Counter	Relay (AND, OR, NOT), Up/down Counter, Shift register Arithmetic calculation, logic calculation
Control type	Contact type (limited life, slow control)	Non-contact type (long life, fast, high reliability)
How to change the logic	By changing wiring between hardware elements	By changing PLC program
Installation time	Building, inspection and test run take a long time.	The time for inspection and test run decreased
System characteristics	Stand-alone control equipment	It is easy to extend system. It is possible to connect to a computer.
Maintainability	For maintenance, long time is needed.	Due to high reliability and long life, the need for maintenance is small.
Volume	Miniaturization is difficult.	Miniaturization is possible.

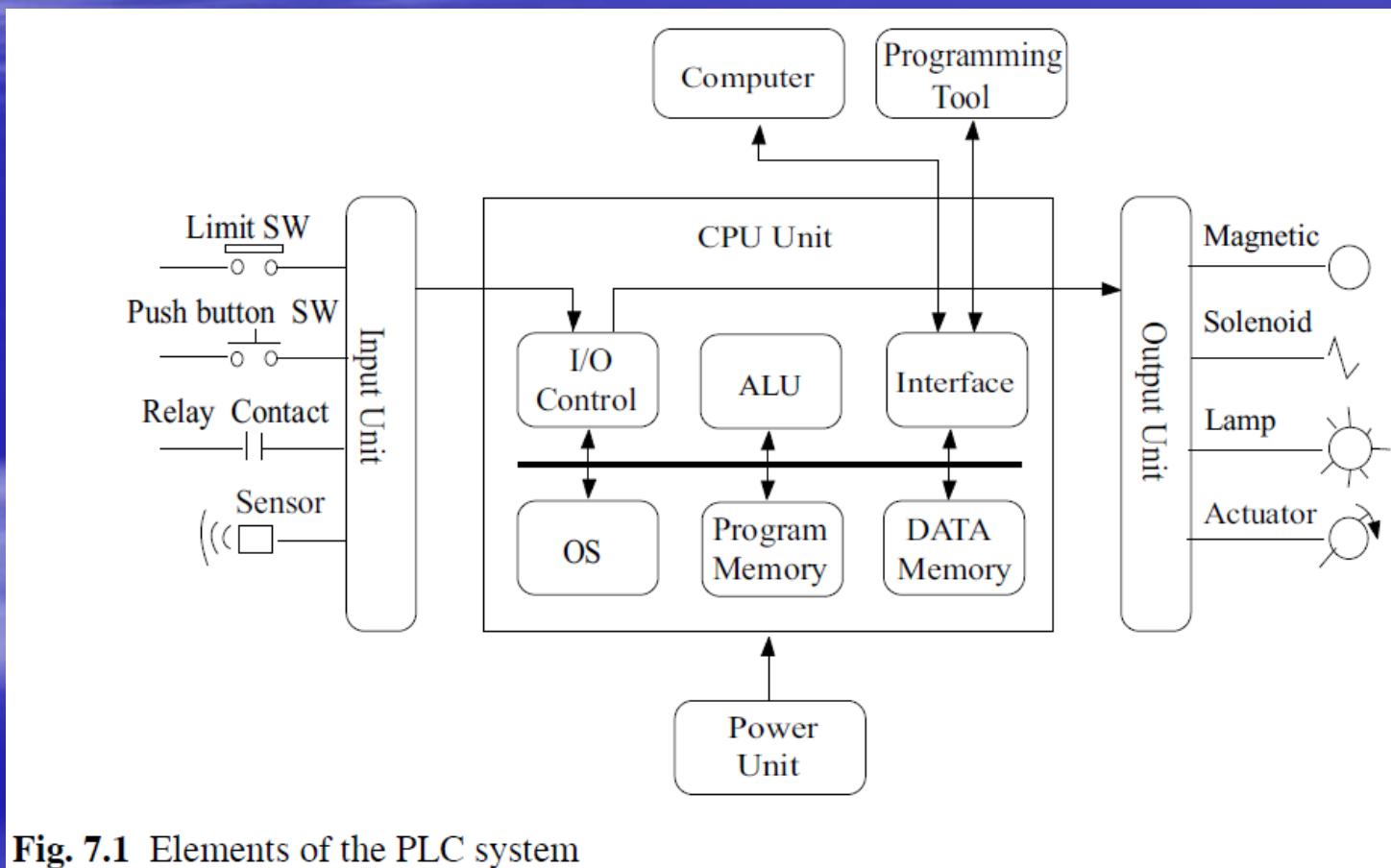
# PLC Elements

## The Elements of a PLC system.

1. Programming Tool – used for editing the PLC program and loading the program to the CPU unit,
2. Input Unit – receives on/off signals from a variety of switches, sensors, and timer and converts the received signals into CPU-interpretable signals,
3. Output Unit – outputs on/off signal to motor, relay, and display,
4. Program Memory – stores the user program,
5. DATA Memory – stores executable program such as OS, and
6. CPU Unit – interfaces various auxiliary equipment and executes the logic calculation.

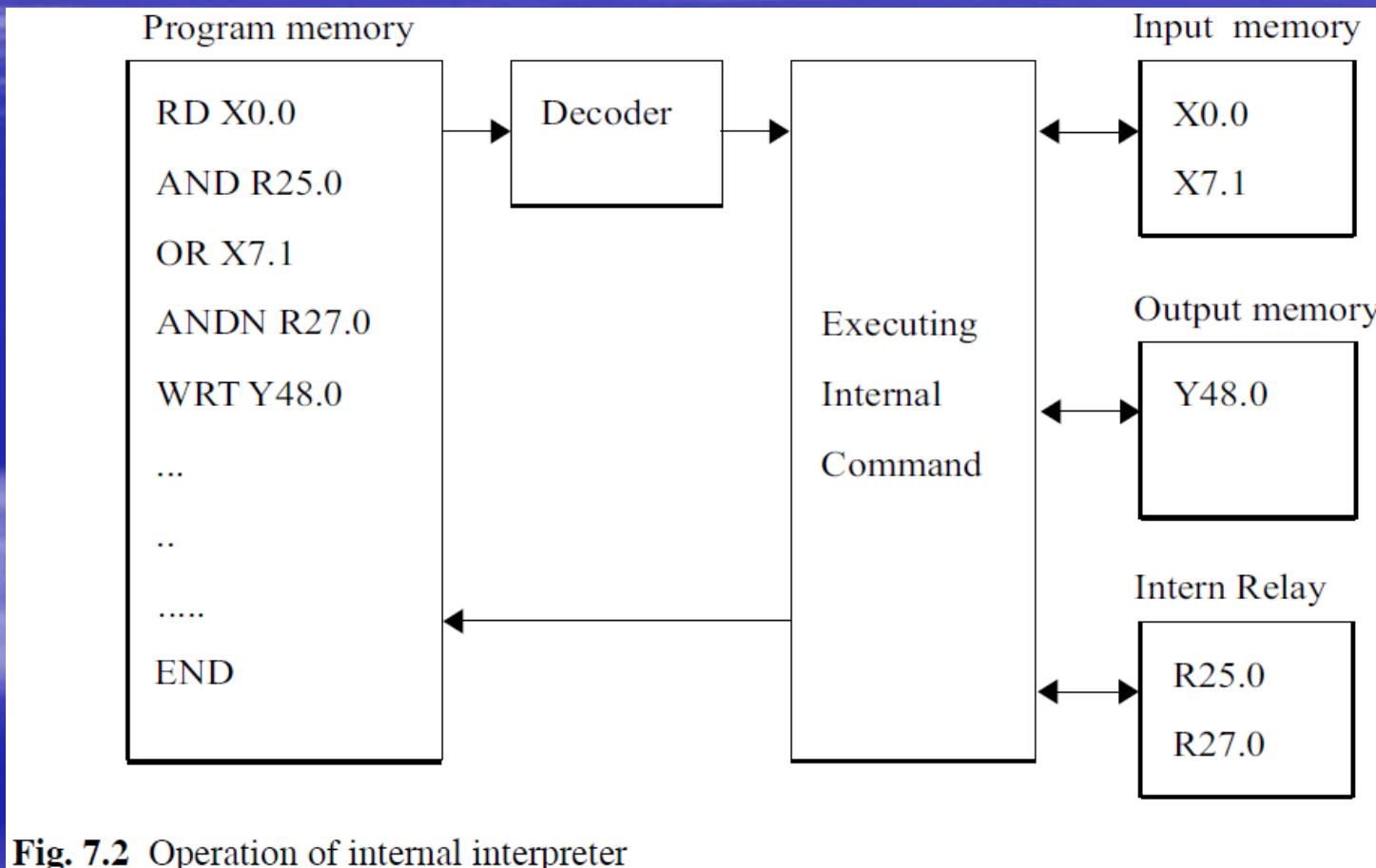
# PLC Elements

The Elements of a PLC system.



# PLC Elements

## The interpreter-type PLC system.



# PLC Elements

## The interpreter-type PLC system.

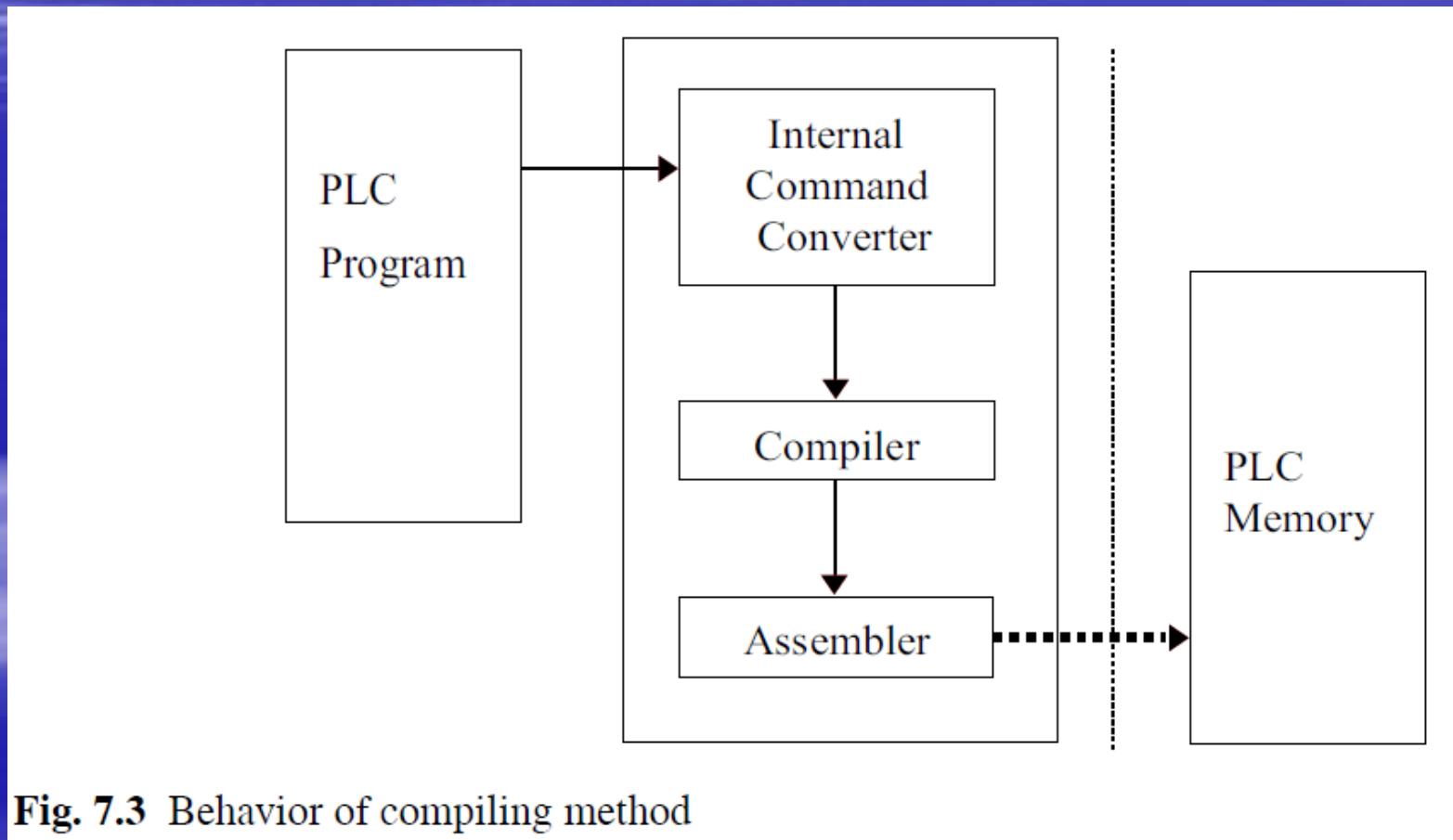
As shown in Fig. 7.2, an internal interpreter takes one command from the part program in Program Memory and interprets the command. The interpreted command is executed by calling the appropriate built-in function. In the process of interpretation, bits of an address in the PLC program are read, and the corresponding address bits are set to ON or OFF.

The above-mentioned method is an interpretative method whereby the interpretation and execution of steps is repeated one by one while logic control is performed. Because an interpreter-type PLC system reads and interprets the individual native code of a program sequence and performs the pre-specified macro routine related with the native code, reduction of execution speed cannot be avoided. Furthermore, because various subroutines for handling internal commands are included in an interpreter-type PLC system, calling subroutines and returning results occur frequently during the execution of a sequence program.

In general, the elapsed time for one scan is very important for the performance of PLC system. Therefore, in order to overcome the slow speed and inefficiency of the interpretative method, a more efficient and faster method is required.

# PLC Elements

The compiler-type PLC system.



# PLC Elements

## The compiler-type PLC system.

The workflow of the compiling method can be summarized as follows:

1. The PLC programmer edits a sequence program using a programming tool, which can exist outside or inside the PLC system.
2. The sequence program is converted to a native program with internal commands by the Internal Command Converter.
3. The compiler replaces the internal commands with the appropriate pre-defined assembly codes.
4. The assembler converts the assembly code into the machine language (binary code) which can be executed by the CPU. Because the majority of commands in a PLC program are independent from each other it is possible to save memory and increase interpretation speed if commands are handled one by one in the assembler after the labels and variables relevant to multiple steps (blocks) have been handled.
5. Finally, the native code is transmitted to the internal memory when the PLC is idle. It is then executed sequentially.

# PLC Programming

## The graphical language and textual language.

There are a variety of the programming languages to represent logic sequences and the IEC (International Electrical Committee) classifies the programming language into the statement list representation and the graphical representation.

As the graphical language, there is the ladder logic that is a method of drawing electrical logic schematics. As the statement list (textual) language, there are mnemonic language, Boolean language, and machine language. In practice, the ladder logic, which can be easily mapped with a sequence logic drawing, has been widely used. Figure 7.4 shows a ladder diagram and a mnemonic program that has same meaning as the ladder diagram.

# PLC Programming

The graphical language and textual language.

Ladder diagram		Mnemonic langeuage	
X0.1	Y2.1	1	RD X0.1
		2	ANDN Y2.1
Y0.1	X1.1	3	RDS Y0.1
		4	AND X1.1
F0.1	X7.7	5	ORS
		6	WR Y4.3
END	G0.1	7	RD F0.1
		8	ANDN X7.7
		9	WRN G0.1
		10	END

**Fig. 7.4** Ladder diagram and mnemonic program

# PLC Programming

The Command sets for Yasnac's PLC programming.

**Table 7.2** Basic commands and functional commands for PLC programming

Type	Instruction	Meaning
Basic command	LD	Regular contact used in beginning of step.
	LD-NOT	'Not' contact used in beginning of step.
	AND	Regular contact that represents the serial connection in Ladder diagram.
	AND-NOT	'Not' contact that represents the serial connection.
	OR	Regular contact that represents the parallel connection.
	OR-NOT	'Not' contact that represents the parallel connection.
	XOR	Exclusive OR.
	XNR	Exclusive AND.
	STR	After storing the operation result on stack, perform LD command.
	STR-NOT	After storing the operation result on stack, perform LD-NOT command.
	AND-STR	Operation result AND the value on stack.
	OR-STR	Operation result OR the value on stack.
	OUT	Ouput the operation result.

# PLC Programming

The Command sets for Yasnac's PLC programming.

Timer	TIM	Fixed timer.
	TMR	Variable timer.
Control command	NOP	No action.
	MCR	If input condition is ON, the program is performed until END.
	END	Represents the end of MCR command.
	RET	Represents the end of PLC program.
	RTI	If input condition is ON, perform RET command.
	SET	Set ON.
	RTH	Rep. end of high-speed PLC program.
	JMP	Jump to the number specified by ADR.
	ADR	Specify the number to which is jumped by the JMP command.
	INR	Increase the value in register by one.
Register command	DCR	Decrease the value in register by one.
	CLR	Reset the register.
	CMR	Reverse the register
	ADI	Add value of register to the specified value.

# Machine Tool PLC Programming

The PLC system of a CNC machine tool executes not only M-, T- and S-codes specified in a part program but also activates or inactivates external switches, executing the PLC program together with input signals from the sensors in machine tools.

Therefore, when we create a PLC program for a CNC machine tool, special considerations are necessary compared with the general PLC system. However, from a functional point of view, the two types of PLC system are not different.

# Machine Tool PLC Programming

The role and characteristics of a PLC program in a machine tool are summarized as follows:

1. The PLC program sends the status of the operation panel to the NCK and shows the status of the NCK to the operator via the operation panel.
  - i. The operator changes the machine operation mode (*e.g.* Auto mode, MDI mode, and Zero Return mode) by turning on or off switches on the operation panel. The change of machine operation mode is sent to the NCK by a PLC program.
  - ii. The operator controls the axis' movement, such as JOG, cycle start, or emergency stop by turning on or off switches on the operation panel.
  - iii. By turning on or off the LEDs and lamps on the operation panel, the PLC program displays the execution status of a part program.

# Machine Tool PLC Programming

2. Through interaction with the NCK, a PLC program helps the execution of a part program.
  - i. A PLC program prevents execution of the next block until the execution of an M-code is completed.
  - ii. PLC program prevents execution of the next block until the spindle speed reaches the value specified by an S-code.
  - iii. The PLC program prevents execution of the next block until the tool specified by a T-code has been attached to the spindle.
3. A PLC program provides various interlock functions to prevent the operator and the workpiece being damaged.
  - i. It prohibits rotation of the spindle in the case of the chuck being unclamped.
  - ii. It stops axis movement as soon as the spindle is stopped.
  - iii. It changes operation mode to single block mode when the coolant's motor is overheated.

# Machine Tool PLC Programming

The general procedure for editing a PLC program is follows:

1. Assign addresses to the input and output ports,
2. Assign addresses to the internal relays and counters,
3. Design the sequence circuit to enable the intended logical operation based on the assigned addresses,
4. Select the appropriate programming language and edit the PLC program in the selected language,
5. Load the PLC program to the CPU module and carry out debugging.

# Machine Tool PLC Programming

Add.	Signal Direction	Example Notation
X	Input from M/C to PLC	64 points (X0.0 ~ X7.7)
Y	Output from PLC to M/C	64 points (Y0.0 ~ Y7.7))
G	Output from PLC to CNC	256 points (G0.0 ~ G31.7)
F	Input from CNC to PLC	256 points (F0.0 ~ F31.7)
R	Internal Relay	512 points (R0.0 ~ R73.7)
T	Internal Timer	512 byte (32bit timer * 16)
C	Internal Counter	512 byte (32bit counter * 16)
D	Internal Memory	2048 byte (32bit data * 64)

Signal Address and Directions

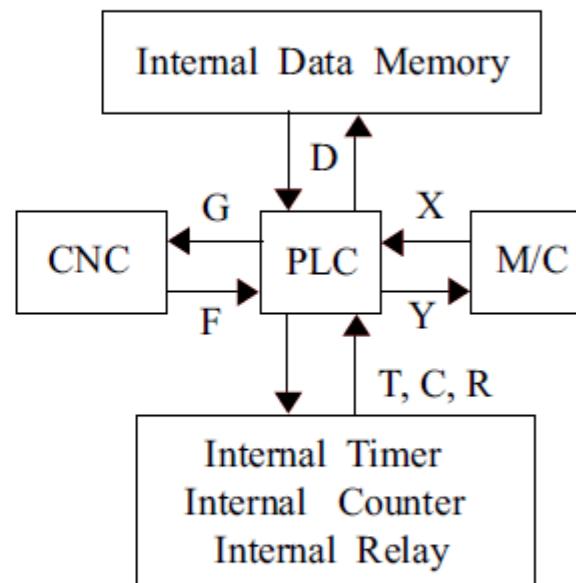


Fig. 7.5 Type, transmitting direction, and reserved address for PLC programming

# Machine Tool PLC Programming

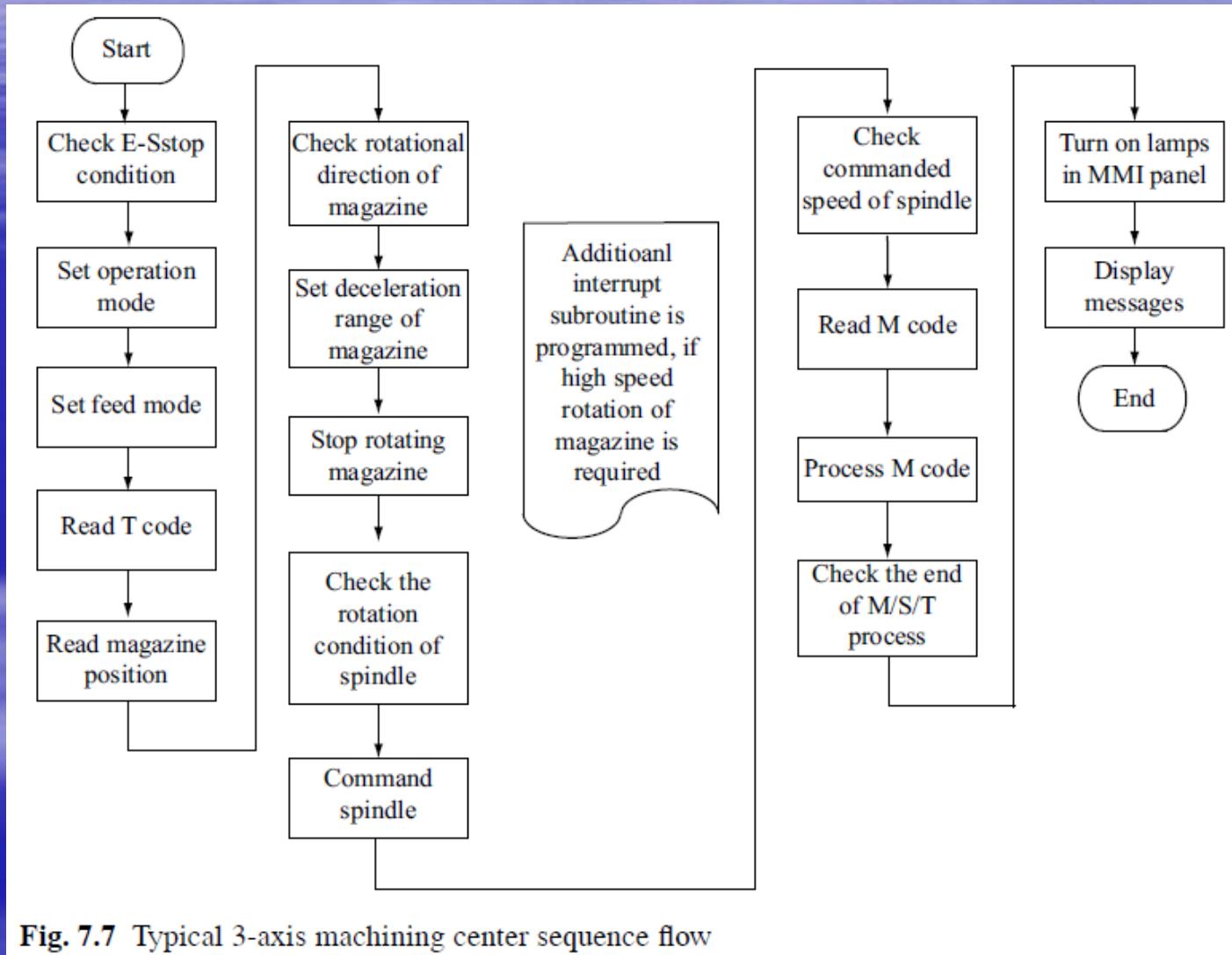
X00.0	MPG Selection	Y00.0	Spindle CW	F05.0	CNC Ready
X00.1	+X axis Jog	Y00.1	Spindle CCW	F05.1	CNC Mode Auto
X00.2	- X axis Jog	Y00.2	Chuck Clamp	F05.2	CNC Mode Manual
X00.3	+Y axis Jog	Y00.3	Chuck Unlamp	F05.3	CNC Mode MDI
X00.4	- Y axis Jog	Y00.4	Servo(on/off)	F05.4	Reset
X00.5	+Z axis Jog	Y00.5	Hydraulic Motor	F05.5	CNC Alarm
X00.6	- Z axis Jog	Y00.6	Lubrication Motor	F05.6	Ref retrun-X axis
X00.7	Rapid	Y00.7	Work Light	F05.7	Ref return-Y axis
X01.0	Emergency Stop	Y01.0	Program End	F07.0	Ref return-Z axis
X01.1	+X axis Limit	Y01.1	+X axis Lamp	F07.1	Dry Run
X01.2	- X axis Limit	Y01.2	- X axis Lamp	F07.2	Machine Lock
X01.3	+Y axis Limit	Y01.3	+Y axis Lamp	F07.3	Optional Stop
X01.4	- Y axis Limit	Y01.4	- Y axis Lamp	F07.4	Optional Block Skip
X01.5	+Z axis Jog	Y01.5	+Z axis Lamp	F07.5v	Spindle Orient
X01.6	- Z axis Jog	Y01.6	- Z axis Lamp	G03.0	PLC Ready
X01.7	Over-travel Cancel	Y01.7	Cycle Start Lamp	G03.1	Emergency Stop

# Machine Tool PLC Programming

X02.0	Edit Lock	Y02.0	Feed Hold Lamp	G03.2	MPG Selection
X02.1	Cycle Start	Y02.1	Machine Lock	G03.3	PLC Alarm
X02.2	Feed Hold	Y02.2	Coolant Auto Lamp	G03.4	Rapid
X02.3	Chuck Switch	Y02.3	Coolant Run Lamp	G03.5	Overtravel Cancel
X02.4	Auto Door	Y02.4	Coolnat Stop Lamp	G03.6	Cycle Start
X02.5	Bar Feeder FW	Y02.5	Run Lamp	G03.7	Free Hold
X02.6	Bar Feeder BW	Y02.6	Alarm Lamp	G04.0	Spindle CW
X02.7	Door Interlock	Y02.7	Spindle CW Lamp	G04.1	Spindle CW
*					
*					
*					

Fig. 7.6 PLC programming signal definition (partial)

# Machine Tool PLC Programming



# PLC System Functions

However, the PLC system is generally a closed system and depends highly on the maker's own technology. This means that the user can use only the functions provided by the maker and the user's own technology and functions cannot be applied. Because of this, whenever the PLC system is changed, the user should be re-trained and the PLC program should be re-programmed. To solve the above-mentioned problem, compatibility and openness of PLC system are necessary. For this, the PLC system has advanced to become an open PLC system that can meet the following requirements:

1. Portability: The PLC program can be operated and is reusable regardless of PLC system and maker.
2. Connectivity: Communication (data transmission) between PLC systems whose makers are different should be guaranteed.
3. Standardization: The user interface and programming language are unified regardless of system and maker.

# PLC System Functions

IEC61131添加6, 作為國際標準的編號

To overcome these problems, the activity for standardizing programming environments for industrial automation equipment was started and the IEC, (International Electrotechnical Commission), established IEC1131-3 in 1993. The standard IEC1131, is the international standard for PLC, consisting of five parts and IEC1131-3 is one of the parts of which IEC1131 is composed.

1. IEC1131-1: PLC General information.
2. IEC1131-2: Equipment and test requirements
3. IEC1131-3: PLC programming language
4. IEC1131-4: User guidelines
5. IEC1131-5: Communications

IEC1131-3 is the international standard for programmable controller programming languages. It specifies the syntax, semantics and display for the following suite of PLC programming languages: 1) Ladder diagram (LD), 2) Sequential Function Charts (SFC), 3) Function Block Diagram (FBD), 4) Structured Text (ST), and 5) Instruction List (IL).

# PLC System Functions

If we use IEC1131-3 to edit a PLC program, it is possible to obtain the following advantages:

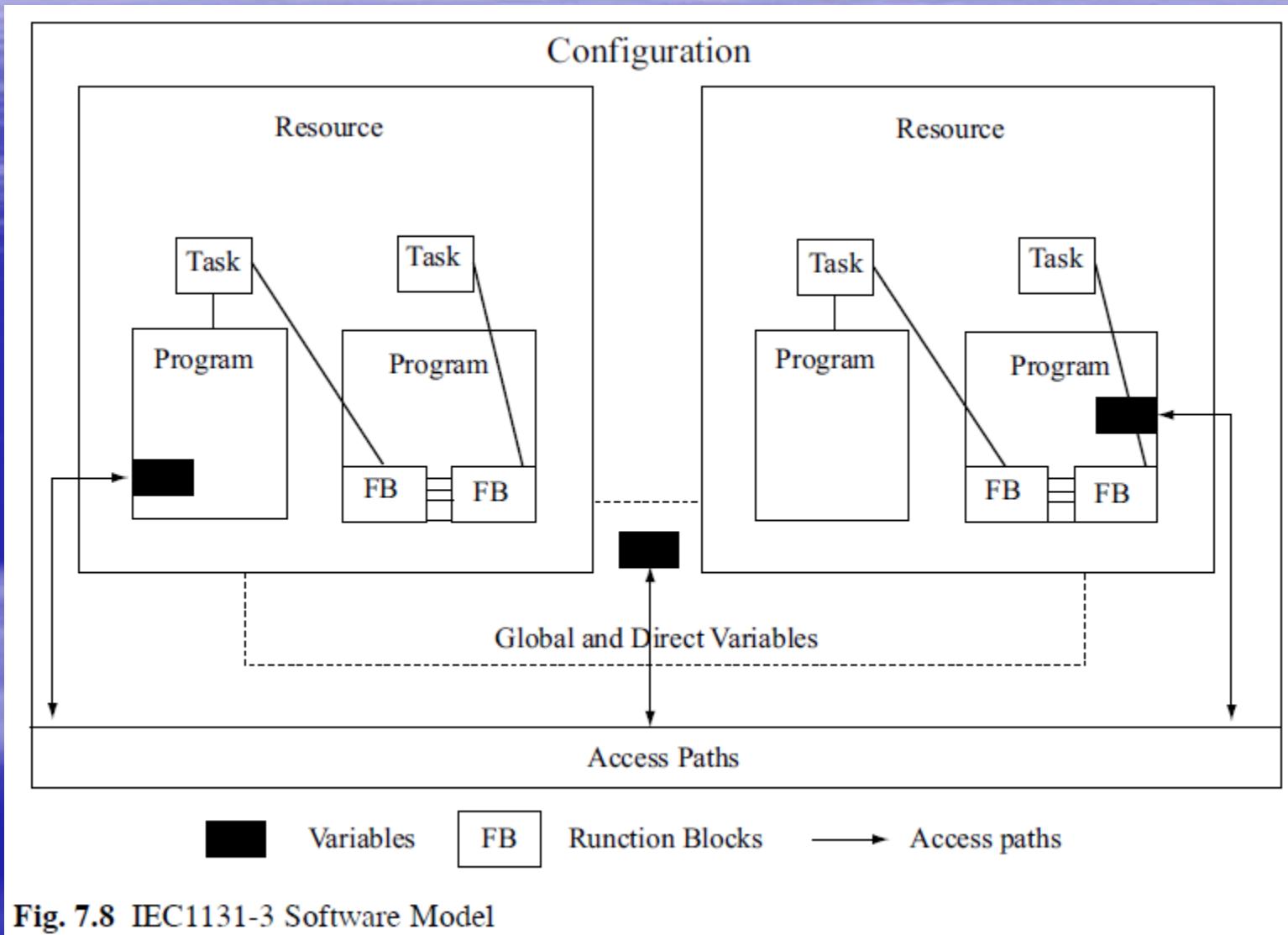
1. Because syntax and semantics are unified, it is possible to generate a program that can be operated on all makers' systems and the program can be executed regardless of maker.
2. It is easy to maintain the program.
3. Because the standard supports the structured programming method, any complex program can be edited in easily understandable and structured format and can easily be maintained.
4. Due to the rigorous syntax and semantics it is possible to reduce program error.
5. The standard makes modularization of a program easy and it is possible to increase the efficiency of programming using program modules.

# PLC System Functions

However, the following disadvantages have been identified:

1. Compared with the sequence programming method, the programming procedure is complex due to computer programming.
2. Much effort is needed to understand and know the grammar of the programming language.
3. Due to the rigorous grammar, the flexibility of programming is restricted.
4. Because IEC1131-3 is heavy, it is not appropriate for application in small-sized PLC systems.

# Software Model



# Software Model

In the software model,

1. Configuration is the top-most concept that represents the PLC system and includes all the software that is contained in one PLC system.
2. Resource is the element that makes up the configuration and means the functions that a processor board provides. It consists of the software that is needed to execute a PLC program.
3. Program means the logical management unit of a user program and is edited by one of the languages specified by IEC1131-3.
4. Function block is a key concept of IEC1131-3 and makes a program structured and modularized. It is the logical management unit for data transmission and consists of the data for defining input/output parameters and the algorithms for performing specific functions.
5. Task represents how a program or function block works. It begins iteratively or by a specific trigger.

# Software Model

6. Function is one of the elements of which a program is composed. It is different from the function block, and it denotes the software that generates a single output from a specific input.
7. Access path does not exist in a single resource system. In multiple resource systems it manages the data of elements and the communication between elements. There are various ways to transmit data in a PLC system. In a program, internal variables are used. To input and output the data to the program, function, and function block, input variables and output variables are used. To share resource or configuration information between programs, external variables that specify them are used. In addition, data transmission between configurations is done by the communication object defined by an access path. Using the access path, it is possible to exchange data between the functions and the programs that are located in different resources or configurations.

# Software Model

Comparing PLC systems with the software model in IEC-1131-3, we can regard the controlled system as configuration. Configuration exchanges data or information with other configurations via Access paths (only specific variables can be transmitted via access paths and extended communication functions, as specified in IEC1131-5).

# Programming Model

In IEC1131-3, five programming languages (actually, four languages and one common element) are specified; LD (Ladder Diagram), IL (Instruction List), ST (Structured Text), FBD (Function Block Diagram), and SFC (Sequential Function Chart). The user can select the appropriate language depending on the characteristics of the program.

# Programming Model

Actually, SFC is not a programming language for implementing the control program, but the representation tool to depict all sequences of a control program. As shown in Fig. 7.9, SFC classifies continuous tasks into Steps as well as Transitions, which are the conditions for shifting between steps, and Actions, being the job to be performed at a step.

In other words, SFC is composed of multiple steps, the module of a program, an action block associated with a particular step, and a transition to represent the condition for shifting between steps. This graphical representation method is based on Petri-nets or IEC848. SFC can be used not only by itself but also with other program languages specified in IEC1131-3. Therefore, SFC is used as a common element in IEC1131-3. SFC supports not only conditional sequencing but also parallel sequencing where one sequence monitors or executes a background task simultaneously with another performing the main control.

# Programming Model



Fig. 7.9 SFC continuous task classification

# Programming Model

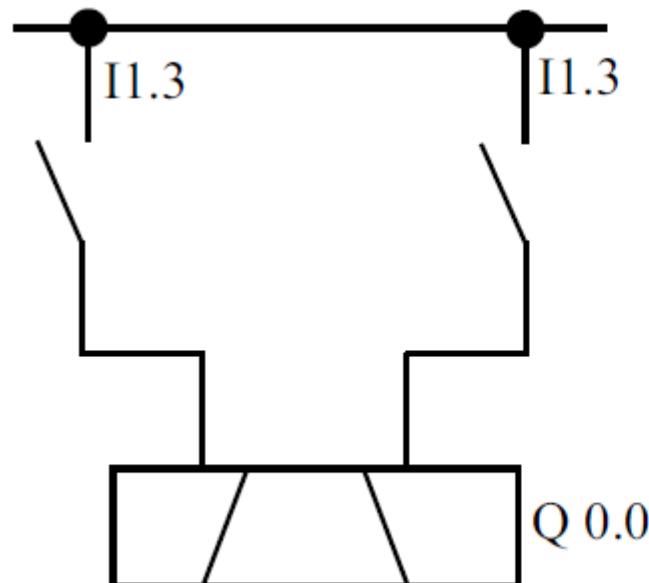
Among the four languages, IL (Instruction List) and ST (Structured Text) are textual languages, while the LD (Ladder Diagram) and FBD (Function Block Diagram) are graphical languages.

1. IL, which is widely used in Europe, is a low-level language like assembler. The advantage of IL is that it is adequate for small-sized programs thanks to simple sequences and logic. It can be used for describing transitions in SFC with function, function block, and program sequences.
2. As ST is a high-level language, it is related to Ada, Pascal, and C. By using ST, it is possible to allocate values to variables and describe tasks by wiring functions, function blocks, and program blocks. Conditional branching and iteration loops are possible too. It can also be used to describe steps, action blocks, and transitions. In addition, it is adequate for numerical calculation and defining complex function blocks.

# Programming Model

3. LD is a well-known language. A program is composed of a combination of input and output contacts. LD is used not only for editing the control program but also monitoring the output/input contacts while a program is being executed. In addition, with functions, function blocks, and programs, it can be used for describing the results of transitions in SFC.
4. FBD is a graphic-based language. It is largely used for programming signal flow between control blocks because of its easy understanding of the flow of a program. FBD is similar to electric circuits including the signal flow of process control. It can be used for describing the behavior by wiring functions, function blocks, and program blocks and describing the step, action block, and transition in SFC.

# Programming Model



Objective:

If input port I 1.3 becomes ON, the output port Q0.0 is Set

If input port I 3.1 becomes ON, the output port Q0.0 is Reset

# Programming Model

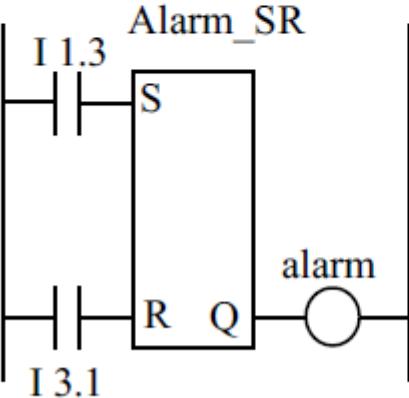
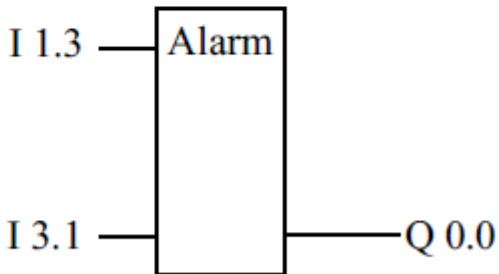
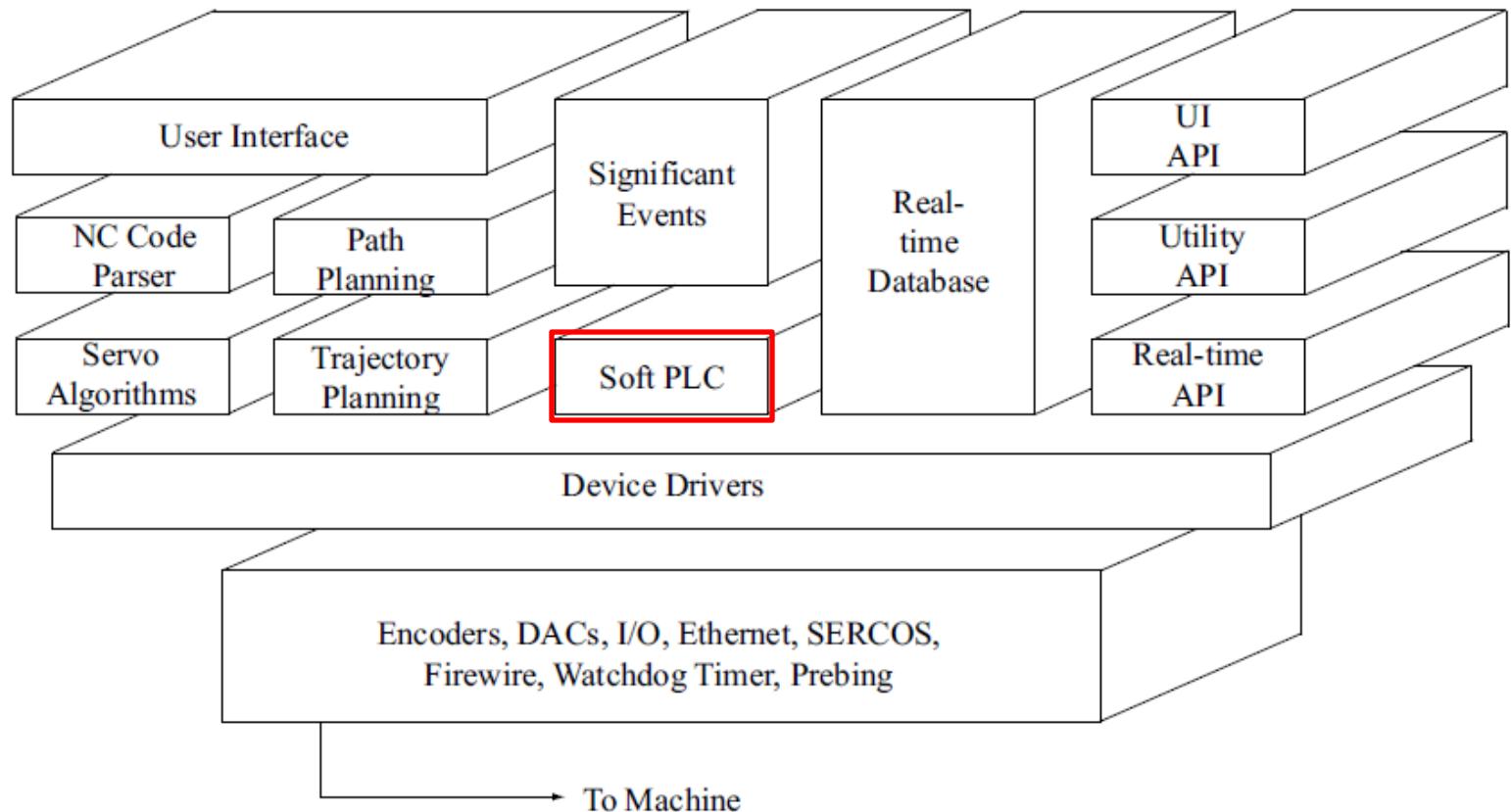
IL(instruction list)	ST(structured text)
LD I 1.3 OR Q 0.0 ANDN I 3.1 OUT Q 0.0	IF I 1.3:= ON THEN Q 0.0:=SET; END-IF IF I 3.1:=ON THEN Q 0.0:=RESET; END-IF
<b>LD(Ladder Diagram)</b>	<b>FBD(Function Block Diagram)</b>
	

Fig. 7.10 The PLC languages specified in IEC1131-3

# Soft PLC

To develop a soft PLC system, real-time operation and reliability of response, which are key requirements for industrial PLC systems, should be guaranteed. Basically, though, PC operating systems cannot satisfy these. However, the non real-time property of DOS or Windows OS can be overcome by various methods and the method of designing a soft PLC system will be described together with design of Soft-NC in the later in this textbook. In Soft-NC, a PC is used as the hardware platform and all CNC functions including PLC functions are implemented in software. In this point, Soft PLC is very similar to Soft-NC. Furthermore, Soft NC includes more functions than Soft PLC, used for NCK control and MMI. Therefore, if NCK functions and MMI functions are omitted or simplified from Soft NC, Soft NC and Soft PLC can be regarded as the same system. Soft PLC which is made by a user interface and the PLC kernel based on the IEC1131-3 can be regarded as partial systems of Soft-NC.

# Soft PLC



**Fig. 7.12** Open CNC system of MDSI

# PLC System Functions

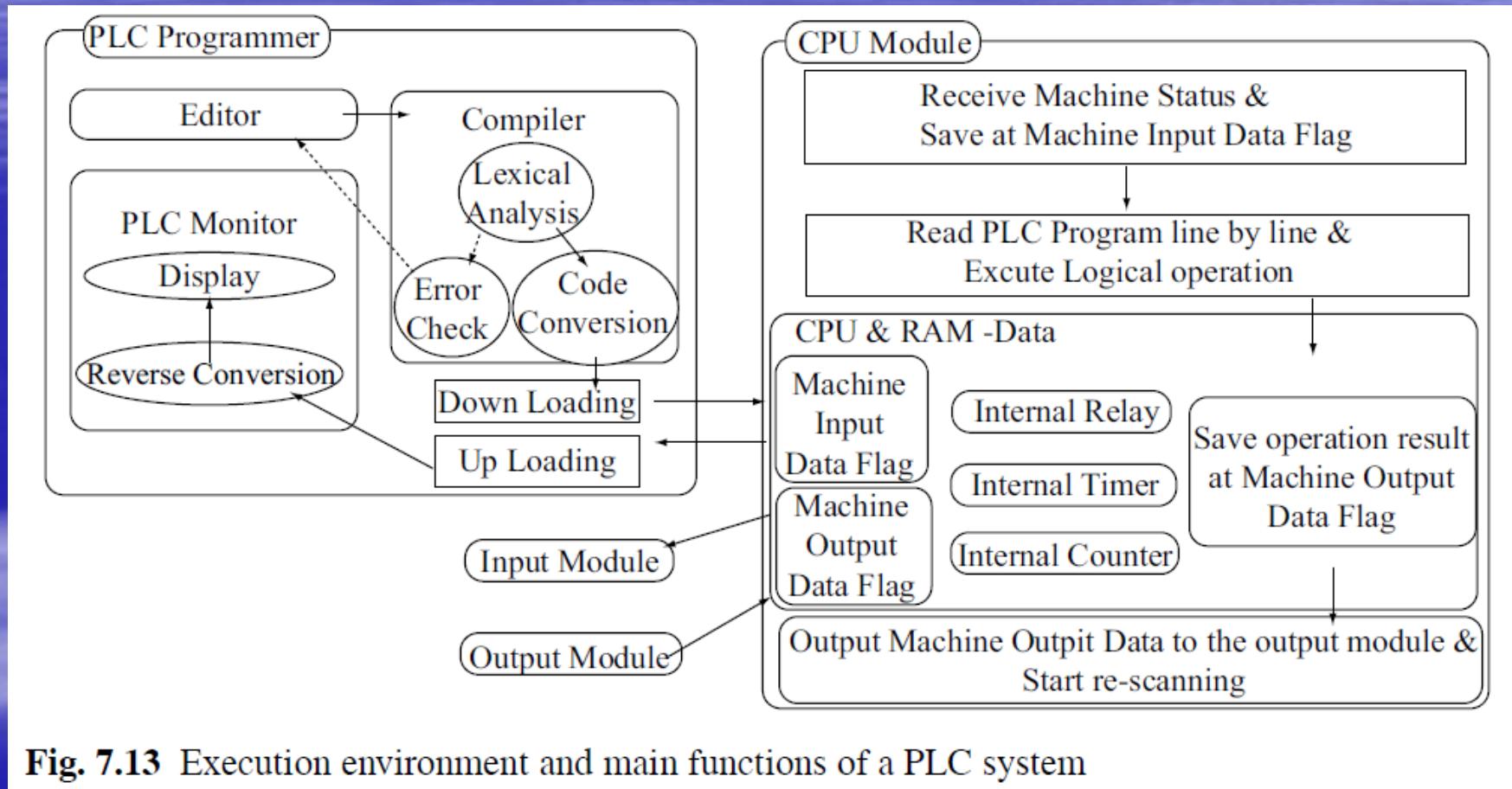


Fig. 7.13 Execution environment and main functions of a PLC system

# PLC System Functions

The main task of the CPU module is to read and execute a binary program from the PLC programmer. We call this module the ‘executor’ and how the CPU module works is as follows. Firstly, the CPU module receives the machine status from the input port and stores it to the machine input data flag. In the next step, it reads the PLC program (binary code) line by line and carries out the logical operations. At this moment, the internal relay, timer, counter, and input data flag are referenced. The result from one line execution is saved to the machine output data flag and, finally, the values from the machine output data flag are sent to the output port for actuating the relay, solenoid, and user interface.

If one scan like that mentioned above has been completed, the executor repeats the scan by reading the input ports (or address). Therefore, for designing the PLC system, it is necessary to understand the behavior of this executor.

# PLC Commands and Functions

**Table 7.3** PLC programming commands and functions

Command	Mnemonic	Function
Read	RD X1.0	Read the specific address.
Read Not	RND X2.1	Read the specific address and reverse the value.
Write	WR Y1.0	Write the operation result to the specified address.
Write Not	WRN Y2.0	Reverse the operation result and write the reversed result to the specified address.
And	AND X3.2	Perform logical product between the value of the specified address and the value on the stack register and store the result on the stack.
And Not	ANDN Y0.2	Reverse the value of the specific address, perform logical product between the reversed value and the value on the stack register, and store the result on the stack.
Or	OR Y2.7	Perform logical sum between the value of the specified address and the value on stack register and store the result in stack.

# PLC Commands and Functions

Or Not	ORN G2.1	Reverse the value of the specified address, perform logical sum between the reversed value and the value on the stack register, and store the result on the stack.
Read Stack	RDS Y3.1	Shift the values of the stack to the left and store the value of the specified address in bit 0 of stack.
Read Not Stack	RDNS R1.7	Shift the values of the stack to the left and store the reversed value of the specific address in bit 0 of stack.
And Stack	ANDS	Perform logical product between the values of the lower two bits, save the result in the first bit of stack, SR1, and shift the values of stack to the right.
Or Stack	ORS	Perform logical sum between the values of lower two bits, save the result in the first bit of stack, SR1, and shift the values of stack to the right.

# PLC Commands and Functions

End of P/G	END	Terminate the program.
Timer	TMR #5, 4000	When the input signal is ON, the timer with specified number is executed during the specified time.
Counter	CTR #1, 100	Whenever the input signal is changed to ON from OFF, the counter with the specified number is actuated. When the count reaches the specified number, the counter output maintains ON status before receiving the reset signal.
Move	MOV 7, X1.2	Copy the left operand to the right operand.
And Move Y2.3	ANDM 1, 3,	When the input signal is ON, performs logical product between the values of left operand and middle operand and finally saves the result in the right operand.
Or Move X1.1	ORM 3, 7,	When the input signal is ON, performs a logical sum between the values of the left operand and the middle operand and finally saves the result in the right operand.

# PLC Commands and Functions

Call	CALL Rosa	When the input signal is ON, call the subroutine with the specified name.
Subroutine	SBRT Steph	Start the sub routine.
Return	RET	Terminate the sub routine.
Jump	JMP Khang	When the input signal is ON, jump to the program part starting with the specified label.
Equal	EQU 5, X1.0	When the input signal is ON, if the value of the left operand and the value of the right operand are equal, set the specified bit as ON. Otherwise, set the specified bit as OFF.
Greater Than	GT Y1.1, 10	When the input signal is ON, if the value of the left operand is greater than that of the right operand, set the specified bit as ON. Otherwise, set the specified bit as OFF.
Less Than	LT 10, X4.0	When the input signal is ON, if the value of the left operand is less than that of the right operand, set the specified bit as ON. Otherwise, set the specified bit as OFF.

# PLC Commands and Functions

Shift Right	SFTR X1.0,5	When the input signal is ON, shift the value of the left operand to the right as many bits as given by the right operand.
Shift Left	SFTL Y1.1, 3	When the input signal is ON, shift the value of the left operand to the left as many bits as given by the right operand.
Addition	ADD 1, 2, X1.7	When the input signal is ON, add the values of the left operand and the middle operand and store the result to the right operand.
Subtraction	SUB 3, 1, Y3.1	When the input signal is ON, subtract the values of the left operand and the middle operand and store the result to the right operand.

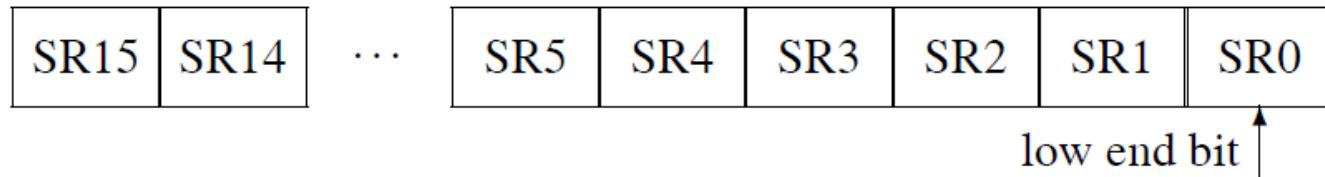
# PLC Commands and Functions

Multiply	MUL 2, 5, X1.3	When the input signal is ON, multiply the values of the left operand and the middle operand and store the result to the right operand.
Division	DIV 4, 2, Y2.2	When the input signal is ON, divide the value of the left operand by the value of the middle operand and store the result to the right operand.
Inverse	INV X1.1, 5	When the input signal is ON, reverse the bit of the address specified by the left operand as specified by the value of the right operand.

# Executor Implementation Example

## ■ Stack Register

The stack register saves the temporary operation result. It is assumed to be composed of 16 bits as follows:



# Executor Implementation Example

## ■ Class Definition

```
class CPLCStack
{
public:
    CPLCStack();
    virtual ~CPLCStack();

private:
    bool m_stack[STACKSIZE];

public:
    void RD(char simbol, int upperadd, int loweradd);
    void RDN(char symbol, int AddNum, int BitNum);
    void WR(char symbol, int AddNum, int BitNum);
    void WRN(char symbol, int AddNum, int BitNum);
    void AND(char symbol, int AddNum, int BitNum);
    void ANDN(char symbol, int AddNum, int BitNum);
    void OR(char symbol, int AddNum, int BitNum);
    void ORN(char symbol, int AddNum, int BitNum);
    void RDS(char symbol, int AddNum, int BitNum);
```

# Executor Implementation Example

## ■ Class Definition

```
void RDNS(char symbol, int AddNum, int BitNum);
void ANDS(char symbol, int AddNum, int BitNum);
void ORS(char symbol, int AddNum, int BitNum);

public:
    void LShift(int i);
    void RShift(int i);
};

void CPLCStack::LShift(int i)
{
    for(int j = STACKSIZE - 1;j >= i;j --)
        m_stack[j] = m_stack[j-i];
}

void CPLCStack::RShift(int i)
{
    for(int j = 0;j < STACKSIZE - i;j++)
        m_stack[j] = m_stack[j+i];
}
```

# Executor Implementation Example

## a) RD (READ)

- This command is to read the value of the specified address and save the value read in SR0.
- It is used for A-type switch.
- Program structure

### - Ladder Diagram



### - Coding sheet and operation result

Number	Command	Address	Comment		SR3	SR2	SR1	SR0
1	RD	X01.0						①
2	WR	Y00.0	W1 Output					①

# Executor Implementation Example

```
void CPLCStack::RD(char symbol, int AddNum, int BitNum)
{
    CSoftPLCDoc* pDoc = GetDoc();
    int index;
    bool value;
    switch (symbol) {
        case 'X':
            // Get the logical status of the particular address.
            index = AddNum*8 + BitNum;
            value = pDoc->XAddress[index];
            break;
        case 'Y':
            // Get the logical status of the particular address.
            index = AddNum*8 + BitNum;
            value = pDoc->YAddress[index];
            break;
        default:
            break;
    }
    m_stack[0] = value; // Save the status of SR0
}
```

# Executor Implementation Example

## b) RDN (READ NOT)

- This command is to read the value of the specified address, reverse the value, and save the reversed value to SR0.
- This is used for B-type switch.
- Program structure

- Ladder diagram



- Coding sheet and operation result

Number	Command	Address	Comment		SR3	SR2	SR1	SR0
1	RDN	X7.2						(1)
2	WR	Y4.1	W1 Output					(1)

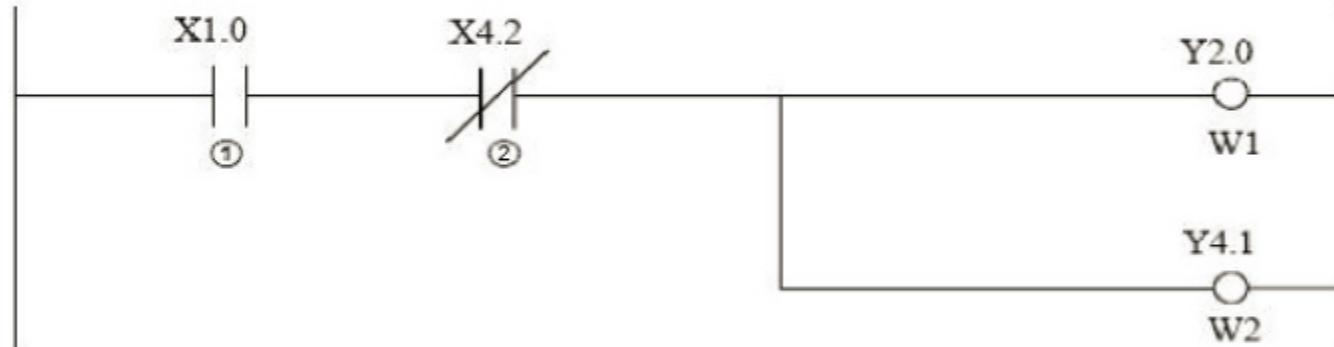
# Executor Implementation Example

```
void CPLCStack::RDN(char symbol, int AddNum, int BitNum)
{
    CSoftPLCDoc* pDoc = GetDoc();
    int index;
    bool value;
    switch (symbol) {
        case 'X':
            // Get the logical status of the particular address.
            index = AddNum*8 + BitNum;
            value = pDoc->XAddress[index];
            break;
        case 'Y':
            // Get the logical status of the particular address.
            index = AddNum*8 + BitNum;
            value = pDoc->YAddress[index];
            break;
        default:
            break;
    }
    m_stack[0] = !value; // Save the reversed status of SR0
}
```

# Executor Implementation Example

## c) WR (WRITE)

- After completing logic operation, it commands to output the value of SR0 to the specific address.
- The logic operation result can be output to more than one address.
- Program structure
  - Ladder diagram



## - Coding sheet and operation result

Number	Command	Address	Comment		SR3	SR2	SR1	SR0
1	RD	X1.0						①
2	ANDN	Y4.2						① * ②
3	WR	Y2.0	W1 Output					① * ②
4	WR	Y4.1	W2 Output					① * ②

# Executor Implementation Example

```
void CPLCStack::WR(char symbol, int AddNum, int BitNum)
{
    CSoftPLCDoc* pDoc = GetDoc();
    int index;
    switch (symbol) {
        case 'X':
            // Output the logical status to the specific address.
            index = AddNum*8 + BitNum;
            pDoc->XAddress[index] = m_stack[0];
            break;
        case 'Y':
            // Output the logical status to the specific address.
            index = AddNum*8 + BitNum;
            pDoc->YAddress[index] = m_stack[0];
            break;
        default:
            break;
    }
}
```

# Executor Implementation Example

## d) WRN (WRITE NOT)

- After completing logical operation, this commands reversal of the value of SR0 and output of the reversed value to the specified address.
- Program structure
  - Ladder diagram



## - Coding sheet and operation result

Number	Command	Address	Comment		SR3	SR2	SR1	SR0
1	RD	X1.1						①
2	ANDN	X7.7						① * ②
3	WRN	G3.1	W1 Output					!(① * ②)

# Executor Implementation Example

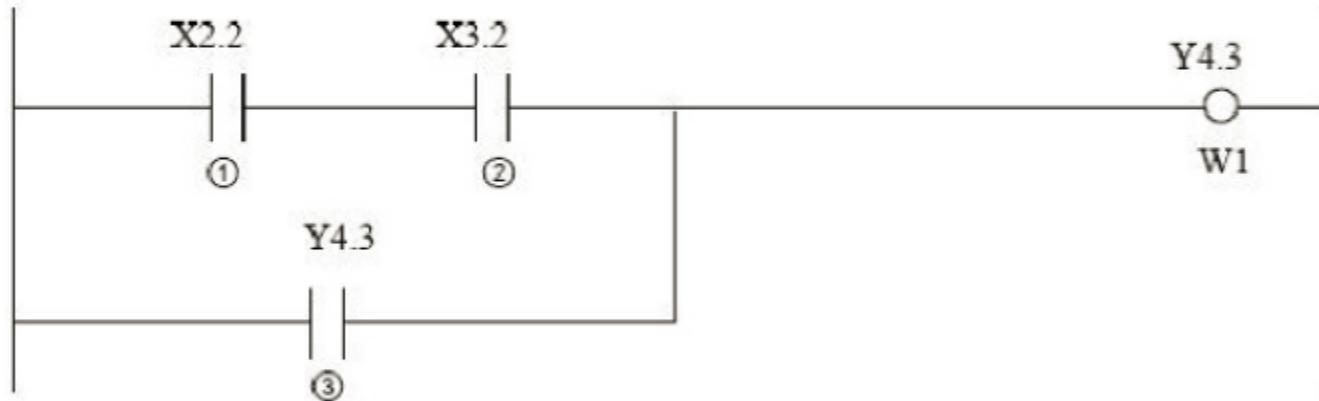
```
void CPLCStack::WRN(char symbol, int AddNum, int BitNum)
{
    CSoftPLCDoc* pDoc = GetDoc();
    int index;
    switch (symbol) {
        case 'X':
            // Output the reversed logical status to the specific address.
            index = AddNum*8 + BitNum;
            pDoc->XAddress[index] = !m_stack[0];
            break;
        case 'Y':
            // Output the reversed logical status to the specific address.
            index = AddNum*8 + BitNum;
            pDoc->YAddress[index] = !m_stack[0];
            break;
        default:
            break;
    }
}
```

# Executor Implementation Example

## e) AND (AND)

- It commands to perform the AND operation (logical product) between the values of the specified address and SR0 and save the operation result to SR0.
- Program structure

- Ladder diagram



- Coding sheet and operation result

Number	Command	Address	Comment		SR3	SR2	SR1	SR0
1	RD	X2.2						①
2	AND	X3.2						① * ②
3	OR	Y4.3						① * ② + ③
4	WR	Y4.3	W1 Output					① * ② + ③

# Executor Implementation Example

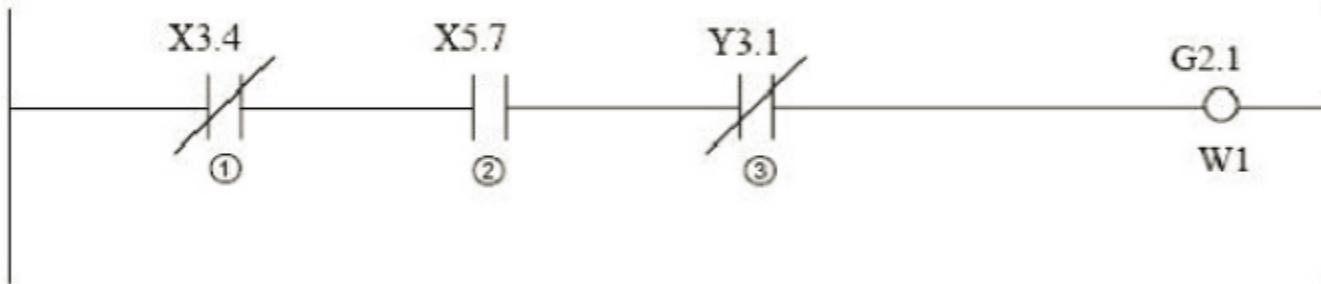
```
void CPLCStack::AND(char symbol, int AddNum, int BitNum)
{
    CSoftPLCDoc* pDoc = GetDoc();
    bool state;
    int index = AddNum*8 + BitNum;
    switch (symbol) {
        case 'X':
            // Get the logical status of the particular address.
            state = pDoc->XAddress[index];
            break;
        case 'Y':
            // Get the logical status of the particular address.
            state = pDoc->YAddress[index];
            break;
        default:
            break;
    }
    if(m_stack[0] && state)
        m_stack[0] = true;
    else
        m_stack[0] = false;
}
```

# Executor Implementation Example

## f) ANDN (AND NOT)

- This command reversal of the value of the specific address, execute the logic product with the value of SR0, and save the result on SR0.
- Program structure.

- Ladder diagram



- Coding sheet and operation result

Number	Command	Address	Comment		SR3	SR2	SR1	SR0
1	RDN	X3.4						①
2	AND	X5.7						① * ②
3	ANDN	Y3.1						① * ② * ③
4	WR	G2.1	W1 Output					① * ② * ③

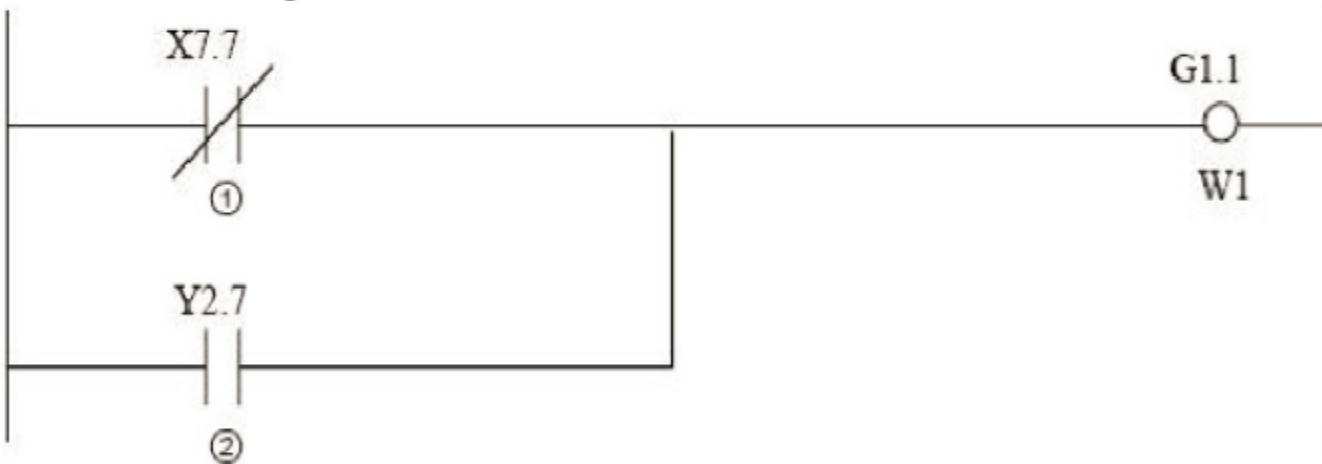
# Executor Implementation Example

```
void CPLCStack::ANDN(char symbol, int AddNum, int BitNum) {  
    CSoftPLCDoc* pDoc = GetDoc();  
    bool state;  
    int index = AddNum*8 + BitNum;  
    switch (symbol) {  
        case 'X':  
            // Get the logical status of the particular address.  
            state = pDoc->XAddress[index];  
            break;  
        case 'Y':  
            // Get the logical status of the particular address.  
            index = AddNum*8 + BitNum;  
            state = pDoc->YAddress[index];  
            break;  
        default:  
            break;  
    }  
    if(m_stack[0] && !state)  
        m_stack[0] = true;  
    else  
        m_stack[0] = false;  
}
```

# Executor Implementation Example

## g) OR (OR)

- It commands to execute OR operation (logical sum) between the values of the specific address and SR0 and save the result on SR0.
- Program structure
  - Ladder diagram



## - Coding sheet and operation result

Number	Command	Address	Comment		SR3	SR2	SR1	SR0
1	RDN	X7.7						①
2	OR	Y2.7						①+②
3	WR	G1.1	W1 Output					①+②

# Executor Implementation Example

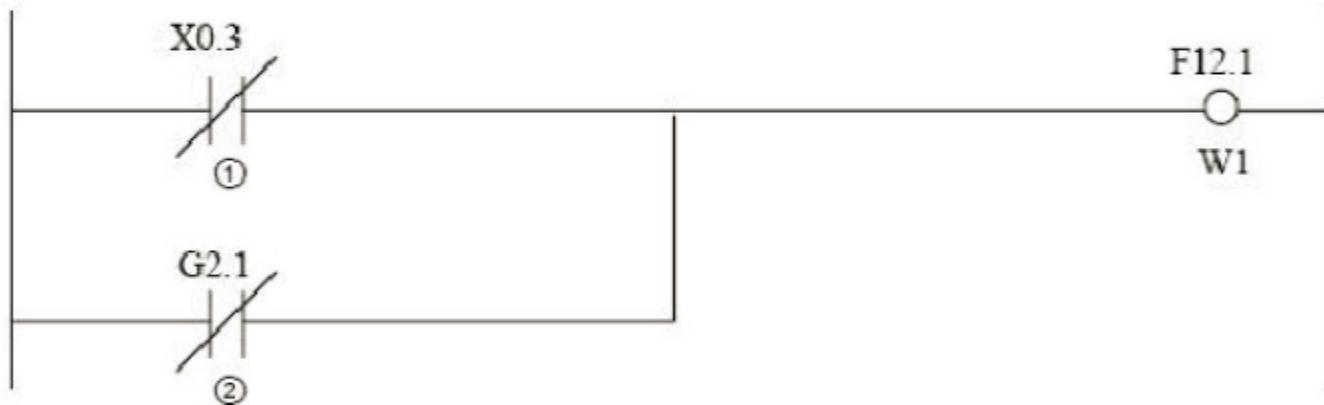
```
void CPLCStack::OR(char symbol, int AddNum, int BitNum)
{
    CSoftPLCDoc* pDoc = GetDoc();
    bool state;
    int index = AddNum*8 + BitNum;
    switch (symbol) {
        // Get the logical status of the address.
        case 'X':
            state = pDoc->XAddress[index];
            break;
        case 'Y':
            state = pDoc->YAddress[index];
            break;
        default:
            break;
    }
    if(m_stack[0] || state)
        m_stack[0] = true;
    else
        m_stack[0] = false;
}
```

# Executor Implementation Example

## h) ORN (OR NOT)

- This command is to reverse the value of the specific address, perform OR operation (logical sum) with the value of SR0, and save the result on SR0.
- Program structure.

- Ladder diagram



- Coding sheet and operation result

Number	Command	Address	Comment		SR3	SR2	SR1	SR0
1	RDN	X0.3						①
2	ORN	G2.1						① + ②
3	WR	F12.1	W1 Output					① + ②

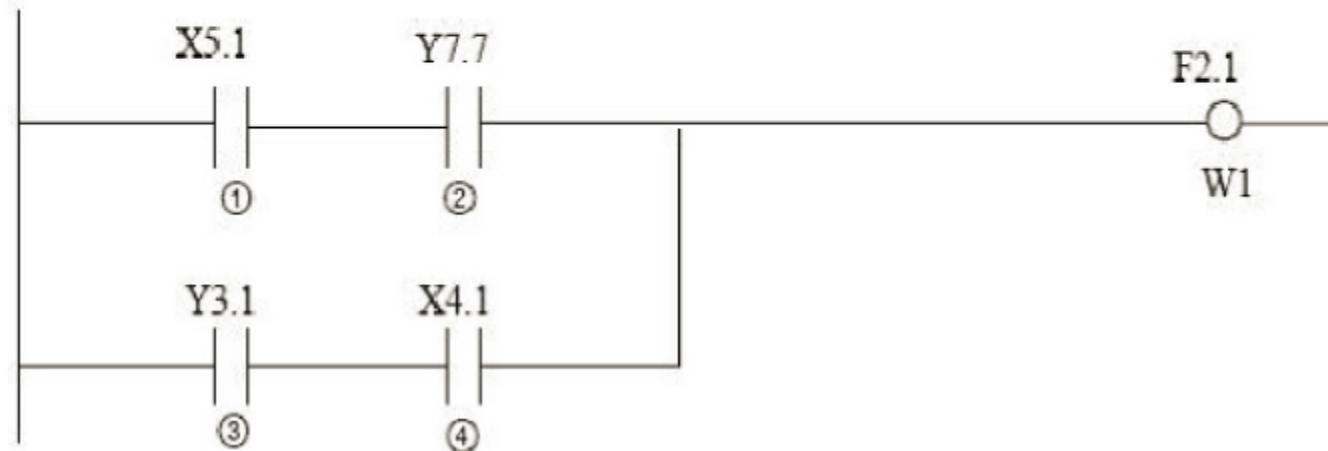
# Executor Implementation Example

```
void CPLCStack::ORN(char symbol, int AddNum, int BitNum)
{
    CSoftPLCDoc* pDoc = GetDoc();
    bool state;
    int index = AddNum*8 + BitNum;
    switch (symbol) {
        // Get the logical status of the address.
        case 'X':
            state = pDoc->XAddress[index];
            break;
        case 'Y':
            state = pDoc->YAddress[index];
            break;
        default:
            break;
    }
    if(m_stack[0] || !state)
        m_stack[0] = true;
    else
        m_stack[0] = false;
}
```

# Executor Implementation Example

## i) RDS (READ STACK)

- This command is to shift the values of stack register to left bit and set the value of the specific address in SR0.
- Program structure
  - Ladder diagram



# Executor Implementation Example

- Coding sheet and operation result

Number	Command	Address	Comment		SR2	SR1	SR0
1	RD	X51					①
2	AND	Y7.7					① * ②
3	RDS	Y3.1			① * ②		③
4	AND	X4.1			① * ②		③ * ④
5	ORS						① * ② + ③ * ④
7	WR	F2.1	W1 Output				① * ② + ③ * ④

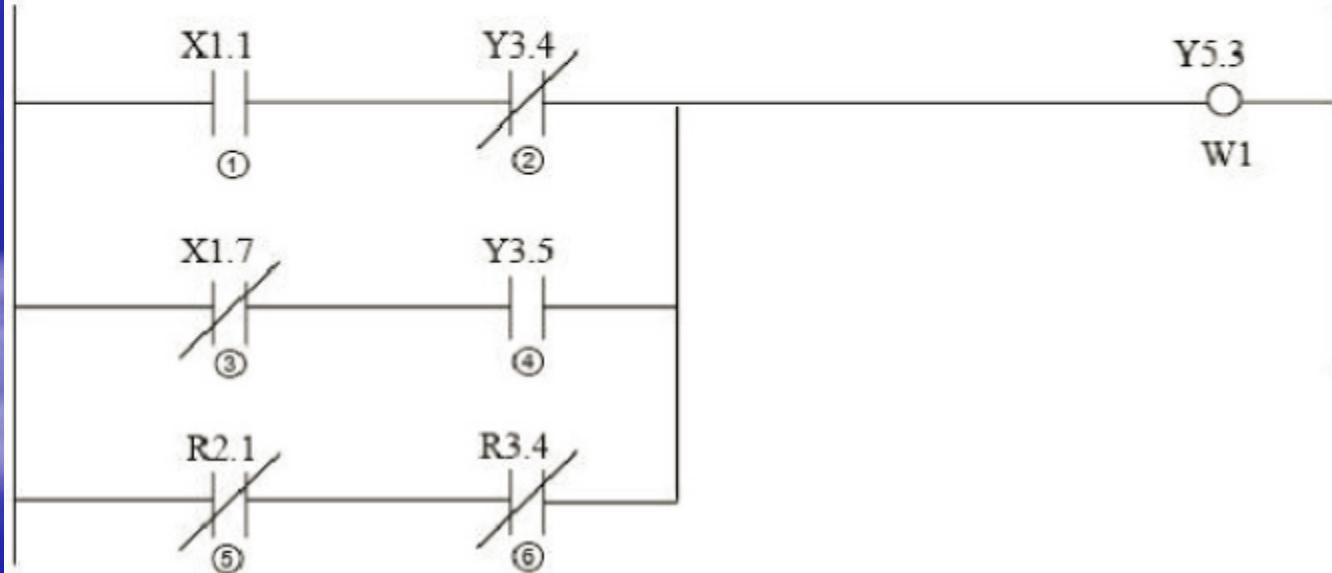
# Executor Implementation Example

```
void CPLCStack::RDS(char symbol, int AddNum, int BitNum)
{
    CSoftPLCDoc* pDoc = GetDoc();
    bool state;
    int index = AddNum*8 + BitNum;
    switch (symbol) {
        // Get the logical status of the address.
        case 'X':
            state = pDoc->XAddress[index];
            break;
        case 'Y':
            state = pDoc->YAddress[index];
            break;
        default:
            break;
    }
    LShift(1);
    m_stack[0] = state;
}
```

# Executor Implementation Example

## j) RDNS (READ NOT STACK)

- This command is to shift the values of stack register to the left bit, reverse the value of the specific address, and set the result on SR0.
- Program structure
  - Ladder diagram



# Executor Implementation Example

- Coding sheet and operation result

Number	Command	Address	Comment	SR1	SR0
1	RD	X1.1			①
2	ANDN	Y3.4			① * ②
3	RDNS	X1.7		① * ②	③
4	AND	Y3.5		① * ②	③ * ④
5	ORS				① * ② + ③ * ④
7	RDNS	R2.1		① * ② + ③ * ④	⑤
7	ANDS	R3.4		① * ② + ③ * ④	⑤ * ⑥
8	ORS				① * ② + ③ * ④ + ⑤ * ⑥
9	WR	Y5.3	W1 Output.		① * ② + ③ * ④ + ⑤ * ⑥

# Executor Implementation Example

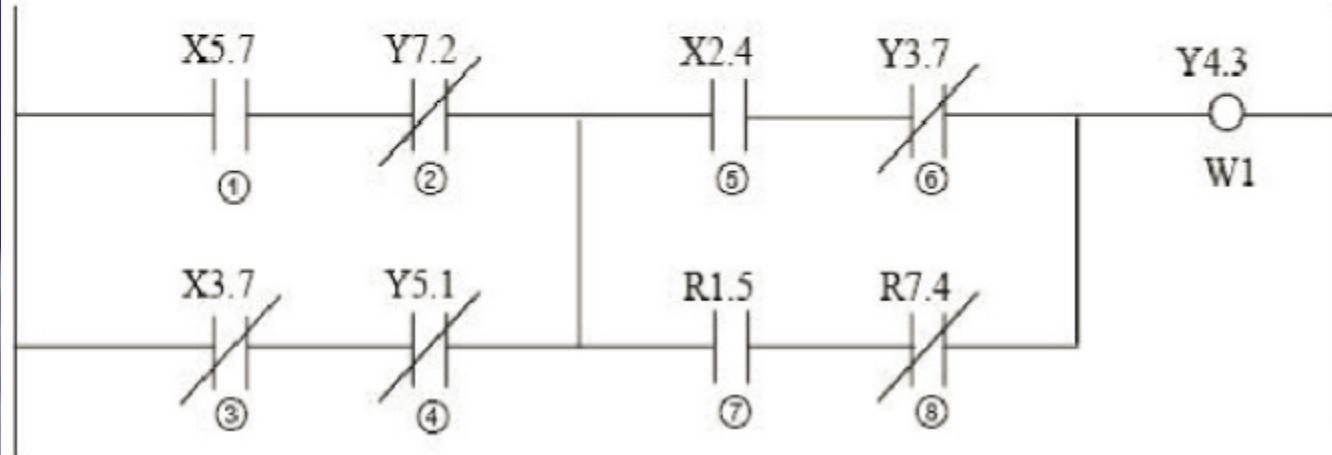
```
void CPLCStack::RDNS(char symbol, int AddNum, int BitNum)
{
    CSoftPLCDoc* pDoc = GetDoc();
    bool state;
    int index = AddNum*8 + BitNum;
    switch (symbol) {
        // Get the logical status of the address.
        case 'X':
            state = pDoc->XAddress[index];
            break;
        case 'Y':
            state = pDoc->YAddress[index];
            break;
        default:
            break;
    }
    LShift(1);
    m_stack[0] = !state;
}
```

# Executor Implementation Example

## k) ANDS (AND STACK)

- This command is to execute the AND operation (logical product) between the values of SR0 and SR1 and shift all the values of the stack register to the right.
- Program structure

- Ladder diagram



# Executor Implementation Example

- Coding sheet and operation result

Number	Command	Address	Comment	SR2	SR1	SR0
1	RD	X5.7				①
2	ANDN	Y7.2				⑤ * ②
3	RDNS	X3.7			① * ②	③
4	ANDN	Y5.1			① * ②	③ * ④
5	ORS					① * ② + ③ * ④
7	RDS	X2.4			① * ② + ③ * ④	⑤
7	ANDN	Y3.7			① * ② + ③ * ④	⑤ * ⑥
8	RDS	R1.5		① * ② + ③ * ④	⑤ * ⑥	⑦
9	ANDN	R7.4		① * ② + ③ * ④	⑤ * ⑥	⑦ * ⑧
10	ORS			① * ② + ③ * ④	⑤ * ⑥ + ⑦ * ⑧	
11	ANDS					(① * ② + ③ * ④) *
12	WR	Y4.3	W1 Output			(⑤ * ⑥ + ⑦ * ⑧)

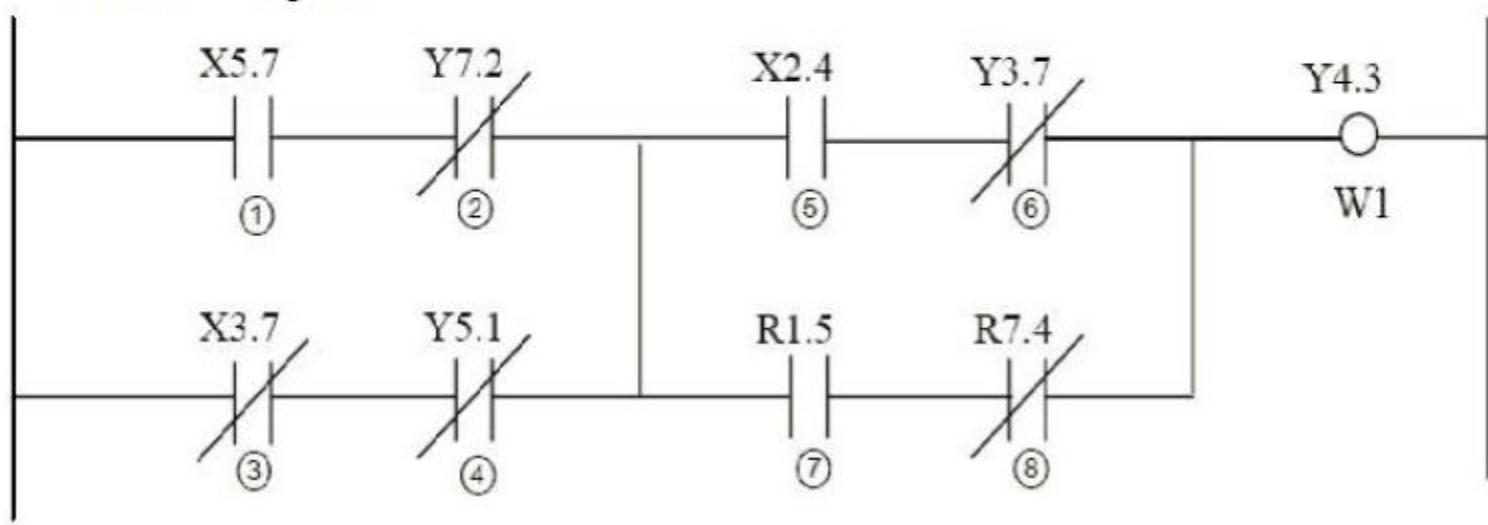
# Executor Implementation Example

```
void CPLCStack::ANDS(char symbol, int AddNum, int BitNum)
{
    if(m_stack[1] && m_stack[0])
        m_stack[1] = true;
    else
        m_stack[1] = false;
    RShift(1);
}
```

# Executor Implementation Example

## 1) ORS (OR STACK)

- This command carries out the logical summation of SR0 and SR1 and sets the result to SR1. It also shifts the value stack register one place to the right.
- Program structure
  - Ladder diagram



# Executor Implementation Example

- Coding sheet and operation result

Number	Command	Address	Comment	SR2	SR1	SR0
1	RD	X5.7				①
2	ANDN	Y7.2				① * ②
3	RDNS	X3.7			① * ②	③
4	ANDN	Y5.1			① * ②	③ * ④
5	ORS					① * ② + ③ * ④
7	RDS	X2.4			① * ② + ③ * ④	⑤
7	ANDN	Y3.7			① * ② + ③ * ④	⑤ * ⑥
8	RDS	R1.5		① * ② + ③ * ④	⑤ * ⑥	⑦
9	ANDN	R7.4		① * ② + ③ * ④	⑤ * ⑥	⑦ * ⑧
10	ORS				① * ② + ③ * ④	⑤ * ⑥ + ⑦ * ⑧
11	ANDS					(① * ② + ③ * ④) * (⑤ * ⑥ + ⑦ * ⑧)
12	WR	Y4.3	W1 Output			

# Executor Implementation Example

```
void CPLCStack::ORS(char symbol, int AddNum, int BitNum)
{
    if(m_stack[1] || m_stack[0])
        m_stack[1] = true;
    else
        m_stack[1] = false;
    RShift(1);
}
```

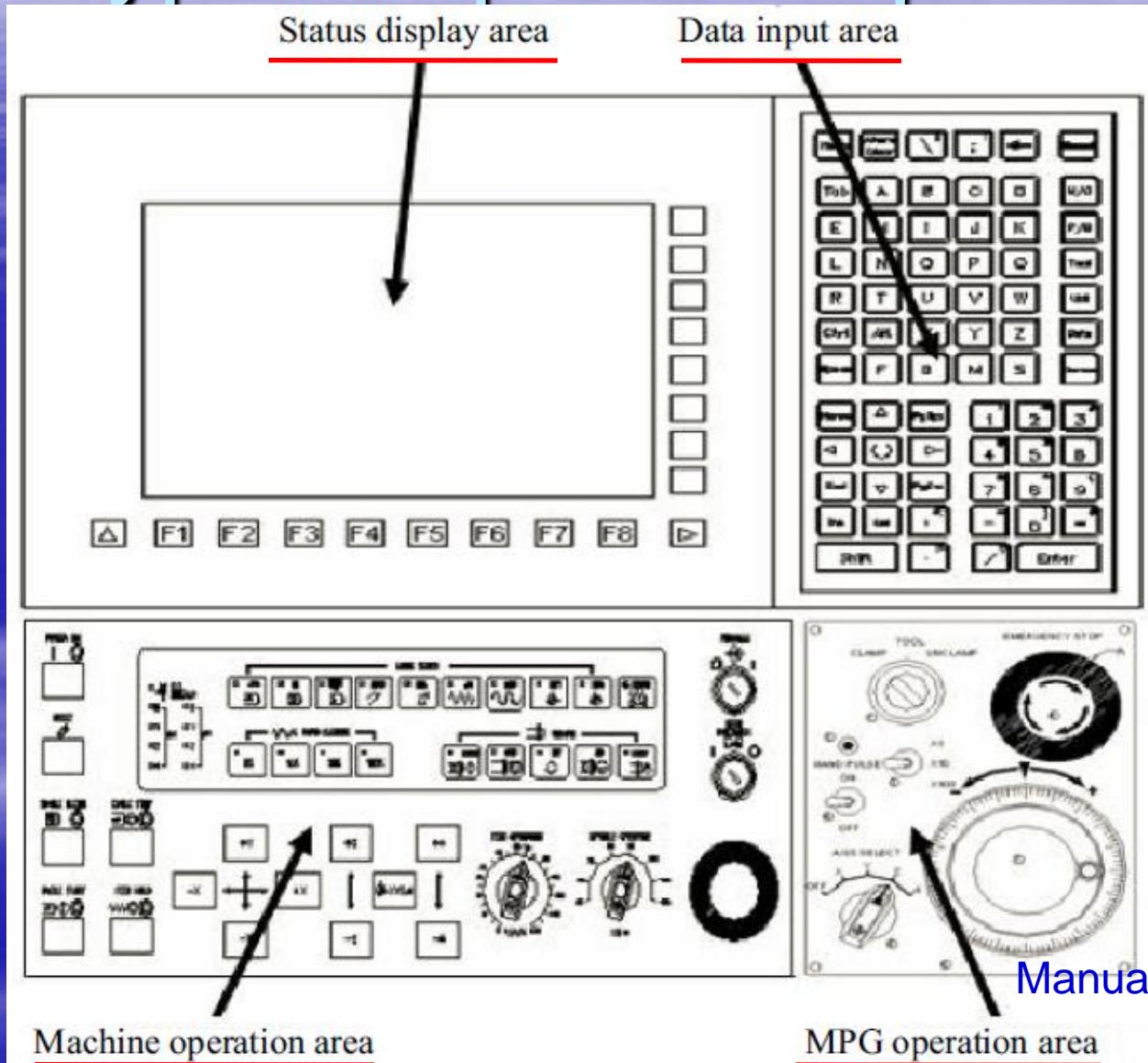
# Theory and Design of CNC Systems

*Chapter 8*  
**Man-Machine Interface**

# Introduction

- The functions of the Man-Machine Interface (MMI):
  - Operate a machine tool.
  - Edit a part program.
  - Perform the part program.
  - Set the parameters.
  - Transmit data.

# Typical operation panel



# Area for Status Display

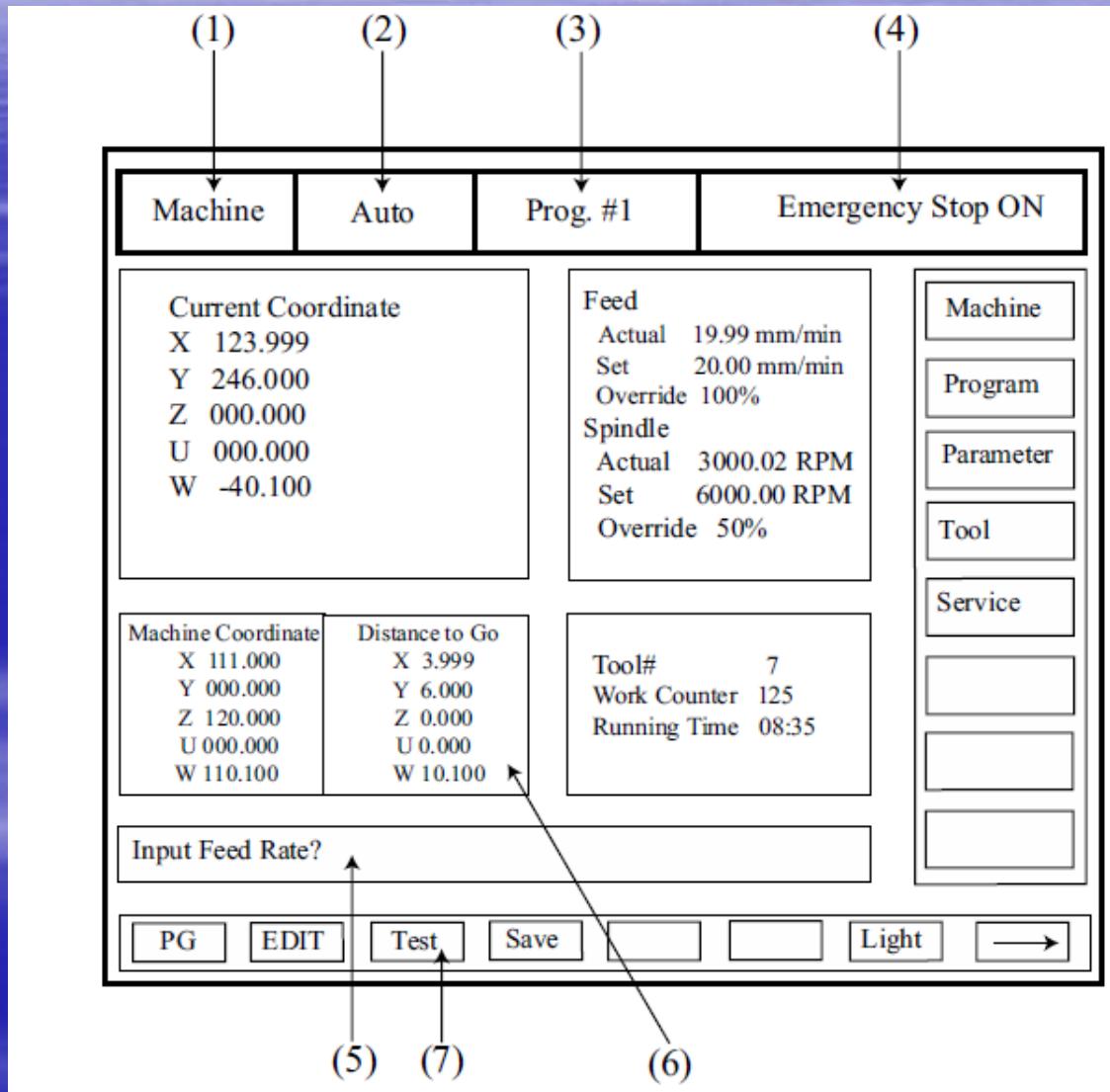


Fig. 8.2 Typical machine status and NC parameters display area

# Area for Status Display

1. Machining information: Displaying information related to the current machine status including the coordinates of machine tools, current part program, cutting tools and machine parameters.
2. Operation Mode: Displaying the operation modes of machine tools, such as zero position return mode, JOG mode, Automatic mode and MDI mode.
3. Program name: Displaying the name of the program that is currently loaded in the memory for machining.
4. Alarm window: Displaying the warning and alarm messages.
5. Key input window: Displaying the strings that are typed by a user.

# Area for Status Display

## 6. Window for displaying user interface relevant to operation mode and function:

- *Machining status (POS)*: operation status such as axis position, spindle speed, feedrate, modal G-codes, and tool number is displayed by this function.
- *Program (PROG)*: the GUI for editing a part program, managing the program folders, graphical simulation, and CAPS is provided by this function.
- *Tool management*: the GUI for managing tool compensation, tool life, and tool offset is provided by this function.
- *Parameter and system*: the GUI for managing the NC parameters, system parameters for servo and spindle is provided.
- *Auxiliary application*: the GUI for monitoring PLC, displaying alarms, performing DNC, and compensating pitch error is provided.

## 7. Function keys: these keys are horizontally placed in the bottom or vertically on the right-hand side of the display and are mapped to the particular functions.

# Area for Data Input

As this area is the keyboard for inputting user data to the CNC system, it consists of alphanumerical input buttons and hot keys for executing the functions of CNC.

# Area for MPG Handling

This area consists of the MPG (Manual Pulse Generator), the MPG handle ON/OFF switch and the feed ratio selection key that are used for the user to move each servo axis manually. In addition, the Chuck CLAMP/UNCLAMP key for manually loading and unloading tools to the spindle and the emergency stop button are located in this area.

# Area for Machine Operation

1. Mode selection switch: for selecting Auto mode, MDI mode, Teach-In mode, Return mode, JOG mode, Handle mode, Incremental Moving mode, and Rapid Moving mode.
2. Rapid Override button: by using this button, rapid feed can be adjusted in scale to 10%, 50%, and 100%.
3. Feed override switch: by using this switch, the commanded feedrate can be adjusted from 10% to 150%.
4. Spindle speed override switch: using this switch, the commanded spindle speed can be adjusted from 50% to 150%.
5. Spindle handling buttons: these buttons consist of the spindle start button, the spindle stop button, rotation direction selection button, and the spindle orientation button, inverse. These buttons are used in MDI mode.
6. Cycle Start button: This button is used for starting the auto-execution or resuming the execution of a part program during feed hold state.
7. Feed Hold button: This button is used for temporarily stopping the axis movement in automatic machining. When the button is pushed, the spindle continues to rotate. If any axis of the machine tool is moving, that axis is stopped after deceleration.

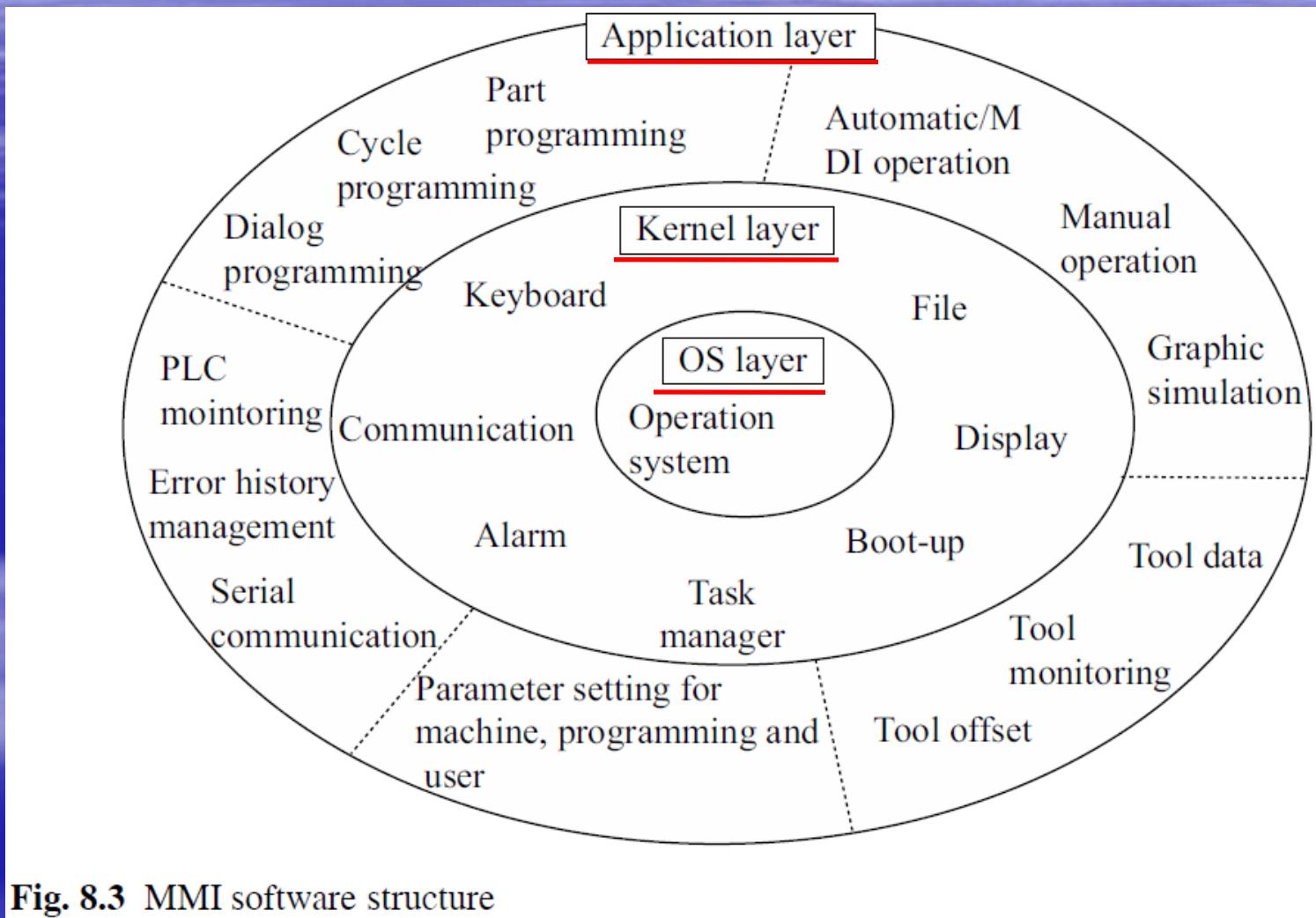
# Area for Machine Operation

8. *Single Block Button:* Single block execution means that in auto mode or MDI mode, the execution of a part program is stopped after the execution of one block has been completed and the next block begins only after the Cycle Start button has been pushed. The single block button turns on or off single block execution mode. If this button is ON during the execution of a part program, the CNC system goes into the idle state after completing the executed block. If this button is OFF, the remaining blocks are executed.
9. *Zero return button:* This button is used for making each axis return to the zero position. All axes can be returned to the zero position simultaneously. Feed override is validated during zero return.
10. *Emergency Stop button:* This button is used for stopping the machine in an abnormal state as soon as possible.

# Area for Machine Operation

11. Part program modification Lock/Unlock key: This key is used for preventing an unauthorized user from modifying, editing, or deleting part programs or preventing unintended modification of a part program due to incorrect operation by a user.
12. Door Interlock key: In the case that this key is ON, if a door is opened while the spindle is rotating, the emergency stop is invoked.
13. In addition, there is an OT (Over Travel) cancel button that temporarily cancels safety mode when an axis moves beyond its set limit, a power switch, and a reset button that initializes the CNC system.

# Structure of the MMI System



# Application Layer

1. Machine Manager: This program monitors the machine status and displays the real-time tool path during machining in Auto mode or MDI operation mode.
2. Parameter Manager: The user can edit NC parameters and system parameters using this program.
3. Program Manager: This program provides the functions for editing G-code programs and managing part programs such as saving and deleting.
4. Tool Manager: This program is used for editing and managing the tool information, such as tool offset, tool life, and tool geometry.
5. Utility: Service functions of the CNC system such as alarm history management, PLC monitoring, DNC, and communication with external systems are provided.

# Kernel Layer

1. *System boot-up*: This function initializes the variables of the operating system and system boot manager for setting the language type of MS Windows, machine parameters, etc.
2. *Communications interface*: This carries out communication and data exchange with the NCK and PLC. It manages the services for sending the data required by the user to the MMI for display.
3. *File management*: This provides the services for managing folders and files, such as copying, saving, deleting, and changing part programs and PLC programs.
4. *Alarm*: This displays alarm and error messages from the machine, PLC, and MMC in the alarm window. It manages the history and displays the help information.
5. *Key input*: This transmits the key input from soft keys, keyboard, and dialog boxes to the applications and the CNC system.
6. *Screen Display*: This handles the horizontal or vertical function key window that is shared by all applications and connects the function keys with particular applications. In addition, it provides the interface for handling MMI soft keys.

# Kernel Layer

7. Task manager: This executes the programs registered in the application layer and provides the function for calling and switching them. It registers the applications as a program list in a text file format and executes the applications sequentially when the task manager begins. When the task manager is terminated, it terminates the applications in reverse order. The basic functions can be summarized as follows.
  - Registering/terminating applications
  - Defining the execution sequence for applications and initializing them while booting up.
  - Switching applications while they are executing.
  - Monitoring system resources.

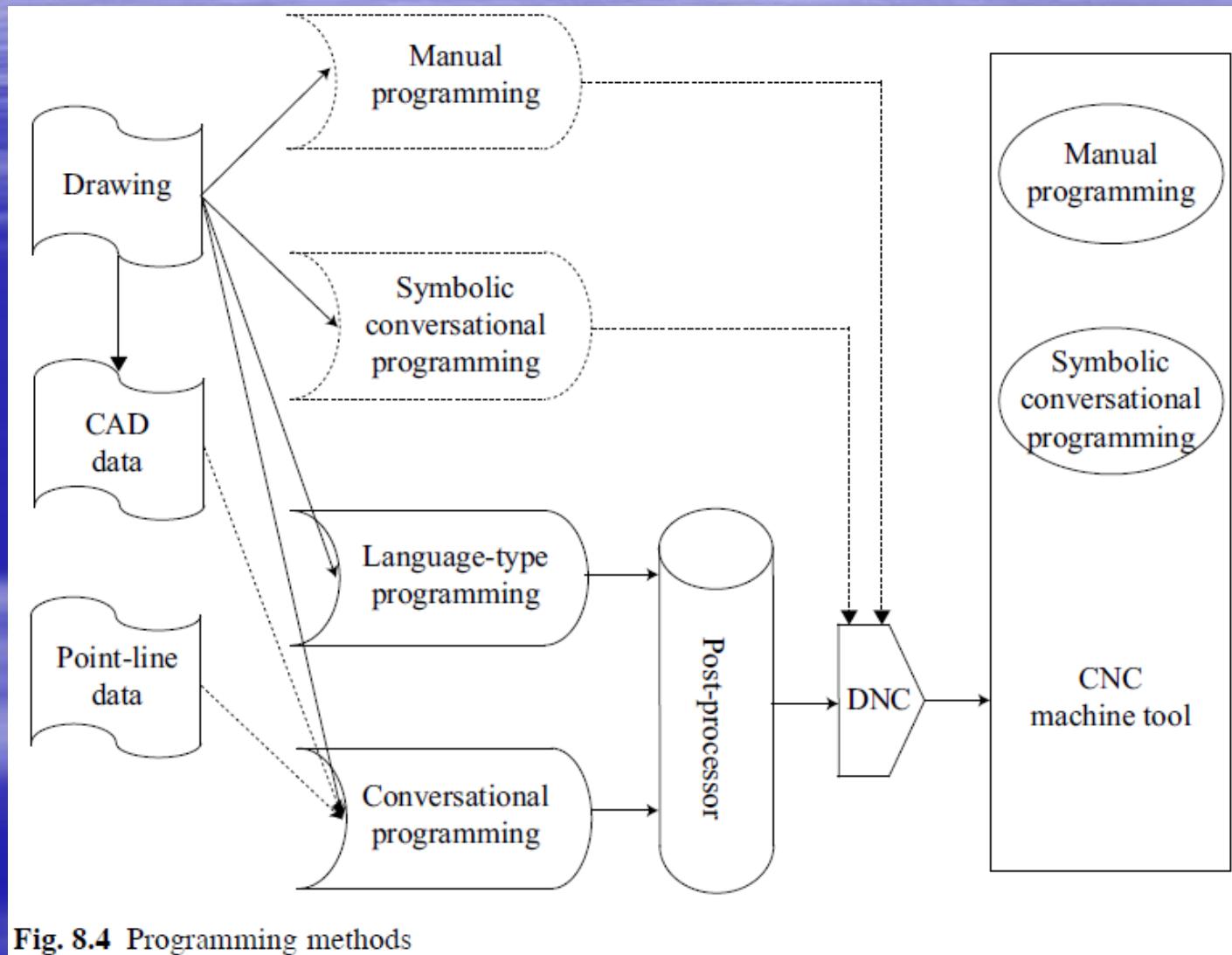
# CNC Programming

In order to machine the part in a drawing by using CNC machine tools, it is necessary to generate a series of instructions for activating those CNC machine tools. This task is called CNC programming.

# The Sequence of Part Programming

1. To analyze the part drawing.
2. To decide on the removal volume and to select the machine.
3. To decide on the jig and chuck.
4. To decide on the setups, machining sequences, cut start points, cut depths for roughing and finishing allowance.
5. To select tools and tool holders and to decide on the tool position.
6. To decide on the technology data such as spindle speed, feedrate, and coolant on/off.
7. To generate the part program (including post-processing).
8. To verify the part program.
9. To machine.

# Programming Methods



# Manual Part Programming

Due to the differences in terms of function and design concept between CNC makers, each CNC system has a slightly different programming instruction set compared with other CNC systems, although the EIA/ISO standard for programming instructions exists. This makes it difficult for a programmer to use a variety of CNC systems. Also, for the manual programming method, the efficiency and productivity of the part program depends on the programmer's ability. Therefore, knowledge about process planning, machining theory, G-code, and complex computations for tool-path generation are necessary for a good programmer and a long training time and much effort are also required. Further, because of the lack of compatibility between programming instructions (G-code), a programmer has to learn new programming instructions if the CNC system is changed. In addition, it is almost impossible to create a part program for machining 2.5D or 3D shape using the manual programming method. However, in the case of simple machining and repeated machining tasks, the manual programming method makes quick programming possible. It also makes it possible to generate a part program quickly by modifying an existing program and using macro programming. Moreover, depending on the programmer's ability, it is possible to generate a part program for unusual and specific shapes.

# Automatic Part Programming

The automatic programming method can be classified into the language-type programming method and the conversational programming method. In the language-type programming method, the machining sequence, part shape, and tools are defined in a language that can be understood by humans. The human-understandable language is then converted into a series of CNC-understandable instructions. In the conversational programming method, the programmer inputs the data for the part shape interactively using a GUI (Graphical User Interface), selects machining sequences, and inputs the technology data for the machining operation. Finally, the CNC system generates the part program based on the programmer's input. Typically conversational programming can be carried out by an external CAM system and a symbolic conversational system that is located either inside the CNC system or in the external computer. In this book, the implementation of symbolic conversational programming systems embedded in the CNC will be addressed in detail.

# Language-type Programming

Language-type programming is the method in which a programmer edits a part program using language-type instructions that the user can easily understand. As the manual programming method is similar to assembly language programming, so the language-type programming is similar to programming in BASIC or FORTRAN. For language-type programming, APT, EXAPT, FAPT, KAPT, and COMPACT II have been widely used.

# Language-type Programming

- *APT (Automatically Programmed Tool)*

APT, which was developed in the USA in the 1960s, is the most famous system for the language-type programming tool and has the greatest number of functions. APT allows representation of various geometries, such as line, circle, ellipse, sphere, cylinder, cone, tabulated cylinder, and general two-dimensional surfaces. By using APT, it is possible to generate programs for 3-axis, 4-axis, and 5-axis machining, including rotation control for spindles and machining tables. Figure 8.5 shows the structure of a part program in APT. The part program consists basically of four parts; 1) the shape definition part where the shape for the machined part is specified, 2) the motion definition part where the tool paths are specified, 3) the post processor part where cutting conditions and the characteristics of the CNC system are specified, and 4) the Auxiliary part where auxiliary data such as tool size, workpiece number, and so on is specified.

# Language-type Programming

CLPRNT

LI82 =LINE/6.25,-1.0,2.0,0.25,-1.0,2.0  
LI83 =LINE/0.25,-1.0,2.0,2.0,3.5,2.0  
LI84 =LINE/2.0,3.5,2.0,6.7525,1.1319,2.0  
CI58 =CIRCLE/6.2507,0.125,2.0,1.125  
LI85 =LINE/6.2507,-1.0,2.0,6.25,-1.0,2.0

CUTTER/0.25,0.05,0.075,0.05,0.0,0.0,4.0 → 選擇刀具  
COOLNT/ON  
SPINDL/1200  
FEDRAT/1.0  
OUTTOL/0.005  
TLAXIS/0.0,0.0,1.0  
FROM/0.0,0.0,5.0  
RAPID  
GOTO/-0.1228,-1.255,3.0  
THICK/0.0,0.13  
DNTCUT  
GOTO/-0.1228,-1.255,1.0  
GO/ON,LI82,TO,(PLANE/0.0,0.0,1.0,1.0),TO,LI83

# Language-type Programming

CUT

INDIRV/0.3624454,0.932005,0.0

TLLFT, GOFWD/LI83, PAST, LI84

GORGT/LI84, TANTO, CI58

GOFWD/CI58, TANTO, LI85

THICK/0.0,0.13,0.0

GOFWD/LI85, ON, (LINE/(POINT/6.25,-1.255,1.0), PERPTO, (LINE/\$  
6.2507,-1.255,1.0,6.25,-1.255,1.0))

THICK/0.0,0.13

GOFWD/LI82, PAST, LI83

GORGT/LI83, PAST, LI84

GORGT/LI84, TANTO, CI58

GOFWD/CI58, TANTO, LI85

THICK/0.0,0.13,0.0

GOFWD/LI85, ON, (LINE/(POINT/6.25,-1.255,1.0), PERPTO, (LINE/\$  
6.2507,-1.255,1.0,6.25,-1.255,1.0))

THICK/0.0,0.13

GOFWD/LI82, PAST, LI83

TLON, GORGT/(LINE/-0.1228,-1.255,1.0,2.0,1.0,1.0), ON, (LINE/\$  
POINT/2.0,1.0,1.0), PERPTO, (LINE/-0.1228,-1.255,1.0,2.0,1.0,1.0))

FINI

# Language-type Programming

- *EXAPT*

EXAPT was developed in Germany. There are three kinds of EXAPT; EXAPT I for position control and linear machining, EXAPT II for turning, and EXAPT III for milling such as two-dimensional contour machining and one-Dimensional linear machining. EXAPT is very similar to APT but without workshop technology. EXAPT decides automatically how many tools are needed by considering the material of the workpiece, required surface roughness, and the shape of the hole specified by the programmer. It calculates automatically the spindle speed and feedrate. In EXAPT II, with user specification of the shape of the blank material and machined part, all machining operations including the machining allowances are generated automatically. On the other hand, it is necessary to register the pre-specified data because appropriate spindle speed, feedrate, and cutting depth can be varied according to the machine and tools. Because EXAPT generates automatically not only the tool path but also machining operations and cutting conditions, it is easier to use than APT. However, the kinds of machineable part shape that can be handled are more limited than with APT.

# Language-type Programming

- *FAPT*

FAPT was developed by FANUC and is similar to APT. FAPT can be used in carry-on exclusive programming equipment. By using particular programming software such as FAPT Turn, FAPT Mill, and FAPT DIE-II, part programs for turning, milling, and die and mold machining can be generated easily. The FAPT Turn/Mill system has the following characteristics.

# Conversational Programming

In order to carry out manual programming or language-type programming, a programmer must know the program instructions, and this makes the generation of part programs difficult. To overcome this problem, creation of part programs without knowledge of detailed program instructions needs to be possible. Due to this requirement, conversational automatic programming systems were developed that enable a programmer to generate tool paths by selecting machining features and operations as well as inputting data and following the system's instructions. In general, the conversational programming system category includes systems executed outside the CNC system in order to generate part programs for two-dimensional contours and three-dimensional free-form surfaces, so-called CAM (Computer-Aided Manufacturing). There are various examples of this type of system, such as CATIA, MasterCAM, EdgeCAM, so on.

# Conversational Programming

As the above-mentioned conversational programming system is an offline system, a part program is generated on an external computer rather than on the CNC system. Because of this, the part program has to be transferred to the CNC system via a DNC system. Therefore, the creator of the part program and the operator of the part program can be different and so, in practice, it can be difficult to apply data optimization to the part program. In addition, in the case of simple machining, the usage of a CAM system reduces productivity. Accordingly, with the improvement in CPU and graphic performance, the symbolic conversational programming method, which enables programmers (including novices) to generate part programs quickly and accurately on the shop floor in order to overcome the disadvantages of CAM systems, has been widely used.

In general, the symbolic conversational programming method is called WOP (Workshop Oriented Programming) or SOP (Shopfloor Oriented Programming). As shown in Table 8.1, this has different characteristics compared with other programming methods. It has been widely used on the shop floor and has a good effect on productivity. In this text book, the design and development of Shopfloor Oriented Programming systems embedded in CNC systems and used on the shopfloor will be addressed in detail.

# Conversational Programming

**Table 8.1** Comparison between programming methods

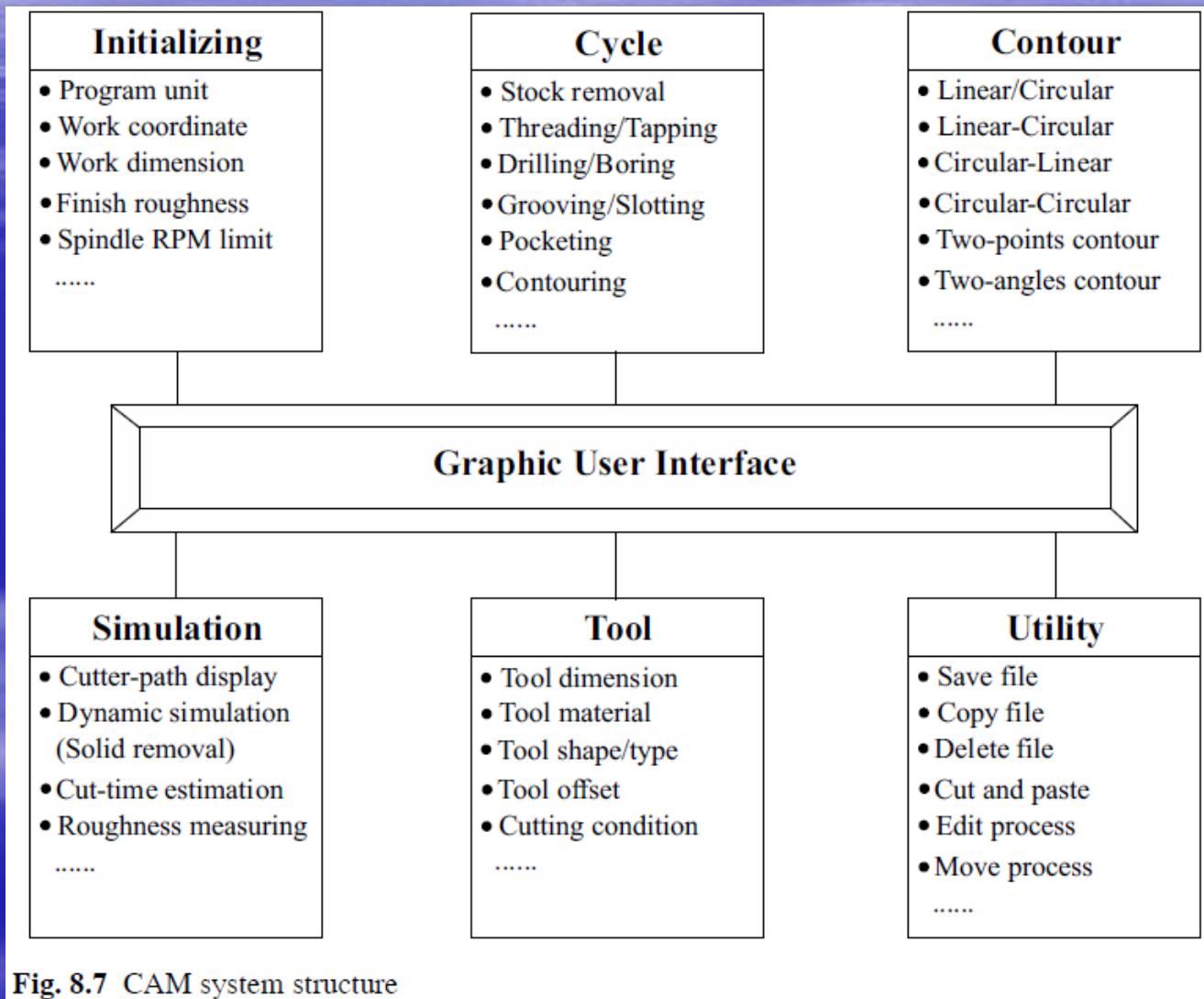
	Advantage	Disadvantage
EIA/ ISO	Easy to apply to simple operations such as tapping, drilling Basic function of CNC equipment.	Full knowledge of G-code required. Knowledge of geometry/mathematics needed for calculating toolpaths.
CAM	Possible to specify complicated shape. Possible to generate programs for various machines with one package.	Very expensive and requires expert. Impossible to feed back programs optimized on shopfloor.
Symbolic	Experienced person can use easily. Easy to create part program. Possible to feed back program optimized on shopfloor.	Program can be used only on a particular machine. In order to apply program to different machine, re-programming required. Programming for complicated parts is restricted.

# CAM Systems & Shopfloor Programming

Recently, with the use of PCs as **MMI** hardware, attempts have been made to embed **PC-based CAM systems in the MMI** and to replace shopfloor programming systems with online CAM systems. Because the ultimate goal is to edit the part program easily, they play similar roles. Each system consists of a **graphical user interface**, **initialization module**, **contour module** for specifying part profiles, **machining cycle module** for specifying machining operations and generating toolpaths, **tool module** for managing tools, **simulation module** for verifying the toolpath, and **utility module** for managing the part programs, as shown in Fig. 8.7.

The CAM system and Shopfloor programming system have slight differences in terms of function. The target machine of a shopfloor programming system is restricted to one machine or to machines of a similar type, but CAM systems can be applied to a variety of machines by providing a post-processing function. Therefore, in the case of a CAM system, a variety of machining conditions have to be considered. However, because only machine-specific functions are considered in the case of the shopfloor programming system, the function and architecture of the shopfloor programming system can be simpler than those of the CAM system.

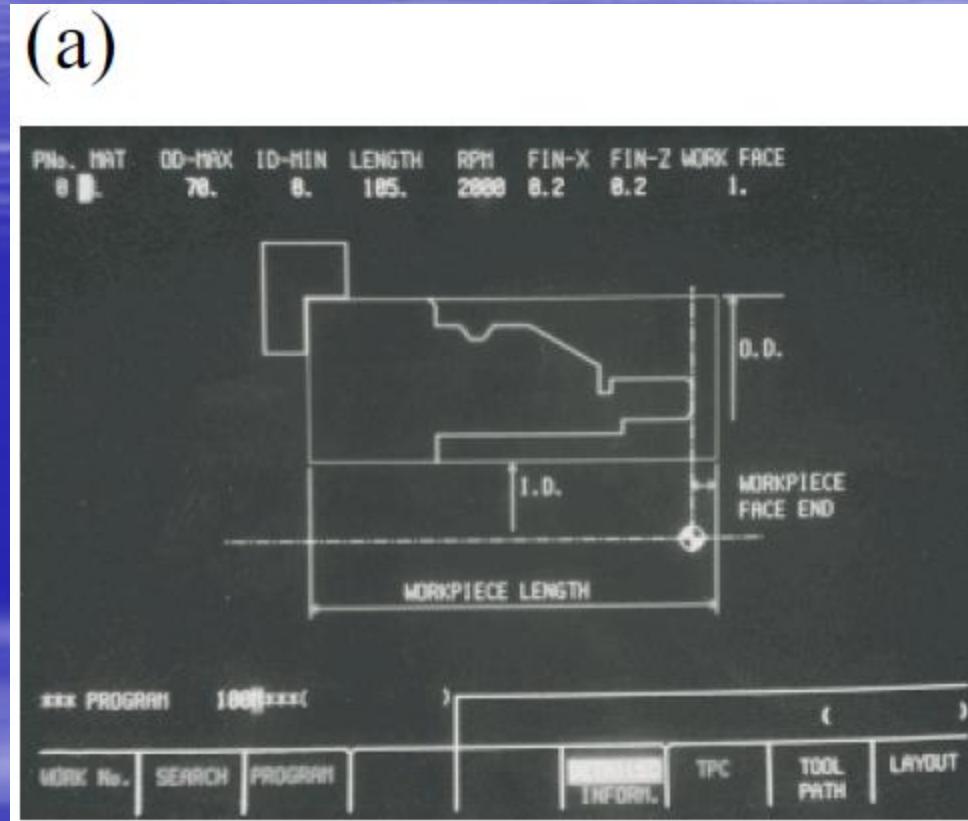
# CAM Systems



# Shopfloor Programming

1. Initialization module: In this module, the global variables, coordinate system (absolute/incremental), programming unit (inch/metric or diameter/radius), spindle data, feed unit (mm/rev or m/min), tool retract position, tool-retraction method, workpiece material, and machine data are specified, (see Fig. 8.8a).

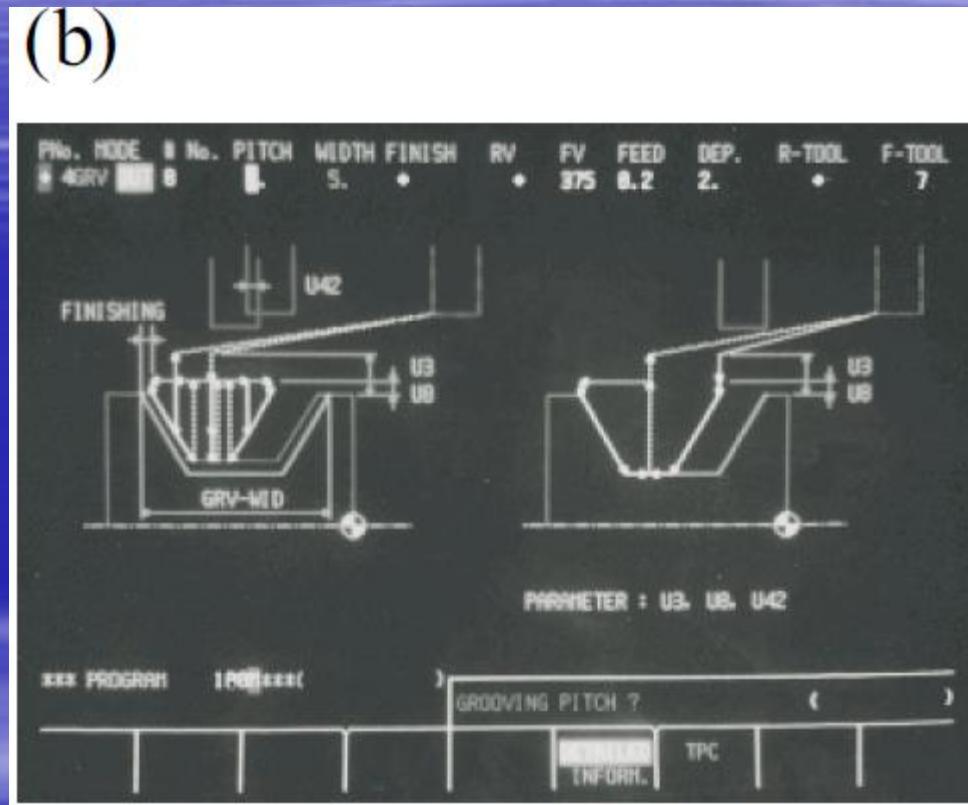
# Shopfloor Programming



# Shopfloor Programming

2. Machining Cycle Module: The specific machining scheme for milling, turning, and drilling is defined as one block. This block makes programming simple and efficient. The block is called a module. The machining operations such as roughing, finishing, drilling, slotting, and pocketing provide a variety of machining strategies. It is important to minimize the data that a programmer should input and select via the GUI during programming. This module is the core module of the conversational programming system (see Fig. 8.8b).

# Shopfloor Programming



# Shopfloor Programming

3. Module for defining the part profile: This module is used for defining the part shape. For this module, a different GUI is provided compared to that of a CAD system. This module provides the conversational contour programming GUI that consists of various graphic menus including line and arc geometries. In particular, in the case of finishing, individual surface finishes can be specified for each profile and feedrate can be computed automatically for each profile based on the surface finish. Of course, in the case of threading, slotting, and drilling, except for contour machining (profile machining), feature definition is carried out together with specification of machining cycles. The important thing for contour programming is that the dimensional data can be input easily without additional calculations during specification of the part profile. In addition, chamfer and round should be easily specified (see Fig. 8.8c).

# Shopfloor Programming

(c)

PIN.	MATERIAL	OD-MAX	ID-MIN	LENGTH	RPM	FIN-X	FIN-Z	WORK FACE					
B		78.	8.	185.	2800	8.2	8.2	1.					
PIN.	MODE	#1	#2	#3	#4	#5	#6	#7	#8	#9	#10	#11	#12
		1	1	1	1	1	1	1	1	1	1	1	1
PIN.	MODE												
2EDG FCE													
SEQ		SPT-X		SPT-Z		FPT-X		FPT-Z					
1		78.		1.		8.		8.					
PIN.	MODE	#	CPT-X	CPT-Z		RV	FV	R-FEED	R-DEP.	R-TOOL	F-TOOL		
3BMR OUT	1	78.	8.			357	700	8.4	5.	1	1		
SEQ	SHP S-CNR		SPT-X		SPT-Z		FPT-X	FPT-Z	F-CNR/S	RADIUS/0	ROUGH		
1LIN C 3.		*	*			28.	25.		*			*** 3	
2TPR		23.923	25.			48.	55.	R 2.	15.			*** 5	
3LIN C 1.		*	*			52.	69.	R 2.5	*			*** 3	
4LIN C 2.		*	*			78.	85.		*			*** 3	

# Shopfloor Programming

4. Tool module: The tool module actually consists of two modules; the first is used for attaching the tool to the turret or tool magazine and the second is used for selecting the tool from the turret or tool magazine. One provides the GUI for specifying tool position, tool type, and tool geometry and the other provides the GUI for selecting the appropriate tool from the turret or tool magazine. Cutting conditions and spindle speed are automatically recommended by the system based on the tool, workpiece material and tool geometry. When the tool has been selected, a variety of data required for machining are automatically set using predefined values. If modification is needed, the programmer can modify these individually (see Fig. 8.8d).

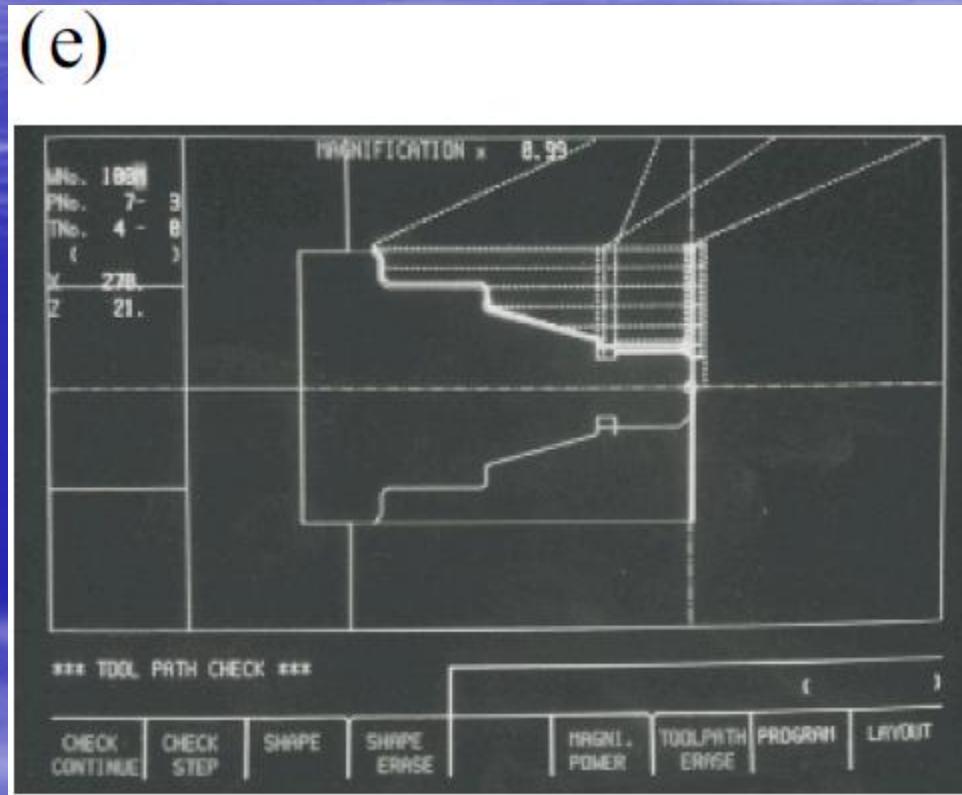
# Shopfloor Programming

(d)

# Shopfloor Programming

5. Toolpath verification module: This module provides the functions for graphically simulating the toolpath of the program that was generated based on the programmer's input. By using this module, a programmer can verify the process from blank material to the final shape. Moreover, because this module displays the machining time (cutting time and non-cutting time) it can be used for optimization of the toolpath, (see Fig. 8.8e).

# Shopfloor Programming



# Shopfloor Programming

6. Utility module: this module provides the functions for copying, deleting, saving, and moving part programs, tool data files, and tool path files. It provides a text editor for modifying the file and moving, deleting, and editing operations for the generated programs, (see Fig. 8.8f)

# Shopfloor Programming

(f)

# Mazatrol Conversational System

The machining cycles in terms of the machining mode and cutting mode, the key characteristic of the Mazatrol Turning Conversational Programming System, are summarized as follows.

1. *Feature Mode*: This denotes the machining cycles that are provided in conversational programming system. In this system, twelve machining cycles are provided as machining cycles, as shown in Fig. 8.9. As can be seen from the figure, the twelve cycles are as follows:

- BAR: This denotes the operation for machining a cylindrical part by turning. This cycle is used for rough machining of an arbitrary part.
- CPY: This is used for finish machining of a specified part with finishing allowance.
- CNR: After finish and rough machining, an undercut area can be left due to the tool's shape. This cycle is used for machining the undercut area.
- EDG: This cycle is used for machining the end face of the cylindrical part.
- THR: This cycle is used for threading.
- GRV: This cycle is used for machining a groove with arbitrary shape.

# Mazatrol Conversational System

- MTR: This cycle is used for cutting in the part.
- DRL: This cycle is used for drilling a hole.
- TAP: This cycle is used for tapping.
- MNP: This is used for generating a part program in manual mode in order to machine special features that are not included specifically in this list.
- MES: This is used for measuring the machined part on the machine after machining has been completed.
- M: This is used for setting M-codes for controlling the machine behavior other than the servo motors.

# Mazatrol Conversational System

1. BAR	2. CPY	3. CNR	4. EDG	5. THR	6. CRV	7. MTR	8. DRV
9. TAP	10. MNP	11. MES	12. M				
	(Manual)	(Measurement)	(Auxiliary)				

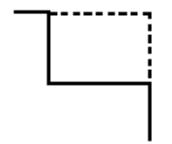
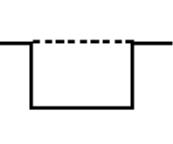
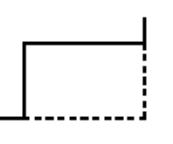
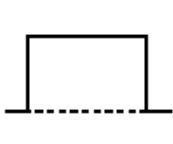
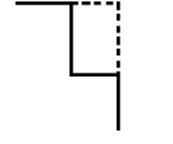
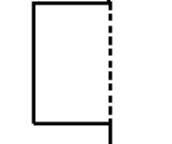
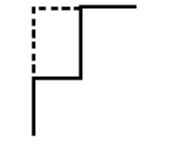
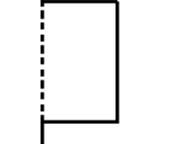
Fig. 8.9 Machining cycles

# Mazatrol Conversational System

2. *Cutting Feature:* After selecting the feature mode, the cutting method should be decided. For example, the rough machining mode feature (*i.e.* BAR) should be followed by inner contouring, outer contouring, facing, and back facing depending on the machined region. Therefore, the Cutting Feature is restricted by the type of Feature Mode. The relationship between Feature Mode and Cutting Feature is shown in Fig. 8.10. When BAR, CPY, CNR, EDG, THR, or GRV are selected, eight kinds of Mode Feature can be selected. In the case of MTR, only OUT (outer contouring) and IN (inner contouring) can be selected. In addition, because DRL and TAP can be applied in the face, selection of Mode Feature is not needed.

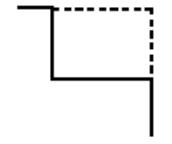
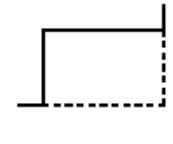
# Mazatrol Conversational System

[1]. Mode Feature: BAR, CPY, CNR, EDG, THR, GRV

1. OUT	2. [OUT]	3. IN	4. [IN]	5. FCE	6. [FCE]	7. BAK	8. [BAK]
							

[3]. Mode Featuring: DRL, TAP-Set as “FCE”

[2]. Mode Feature: MTR

1. OUT	2. IN
	

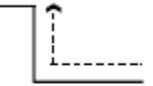
**Fig. 8.10** Relationship between Feature Mode and Cutting Feature

# Mazatrol Conversational System

3. *Machining Strategy:* In order to execute the operation selected from Mode Feature, it is necessary to decide on the machining strategy. The machining strategies that can be applied according to the Feature Mode are shown in Fig. 8.11. For BAR and CNR, the tool retraction method has to be selected. When THR is selected, six kinds of machining strategy can be selected. In the case of GRV, various groove shapes can be selected. Since the conversational programming system guides the choice of appropriate strategies depending on the feature and operation, even non-expert programmers can select the appropriate machining strategy.

# Mazatrol Conversational System

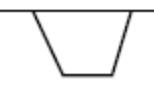
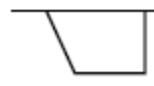
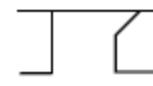
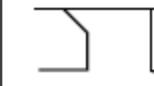
[1] Mode: BAR, CNR

1. #0	2. #1	3. #2
		

[2] Mode: THR

1. #0	2. #1	3. #2	4. [#0]	5. [#1]	6. [#2]
Standard	Constant depth	Constant width	Standard	Constant length	Constant area

[3] Mode: GVR

1. #0	2. #1	3. #2	4. #3	5. #4	6. #5
					

# Mazatrol Conversational System

[4] Mode: DRL

1. #0	2. #1	3. #2	4. #3	5. #0	2. #1	3. #2
Stationary hole drilling	Deep hole drilling	High- speed drilling	Stationary hole reaming	Through hole drilling	Deep hole drilling	High- speed drilling

[5] Mode: TAP

1. #0	2. #1	3. #2	4. #3	5. #4	2. #5
M	UM	PT	PF	PS	Special

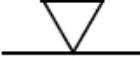
**Fig. 8.11** Machining strategies to be applied according to Feature Mode

# Mazatrol Conversational System

4. *Tool and cutting condition:* After Feature Mode, Cutting Feature, and machining strategy have been specified, it is necessary to select the appropriate tool and decide on the cutting conditions. The tool is selected from the tool database that is pre-specified based on the tools loaded onto the machine. The cutting conditions can be recommended automatically by the system according to the tool and workpiece material or can be input directly by the programmer.

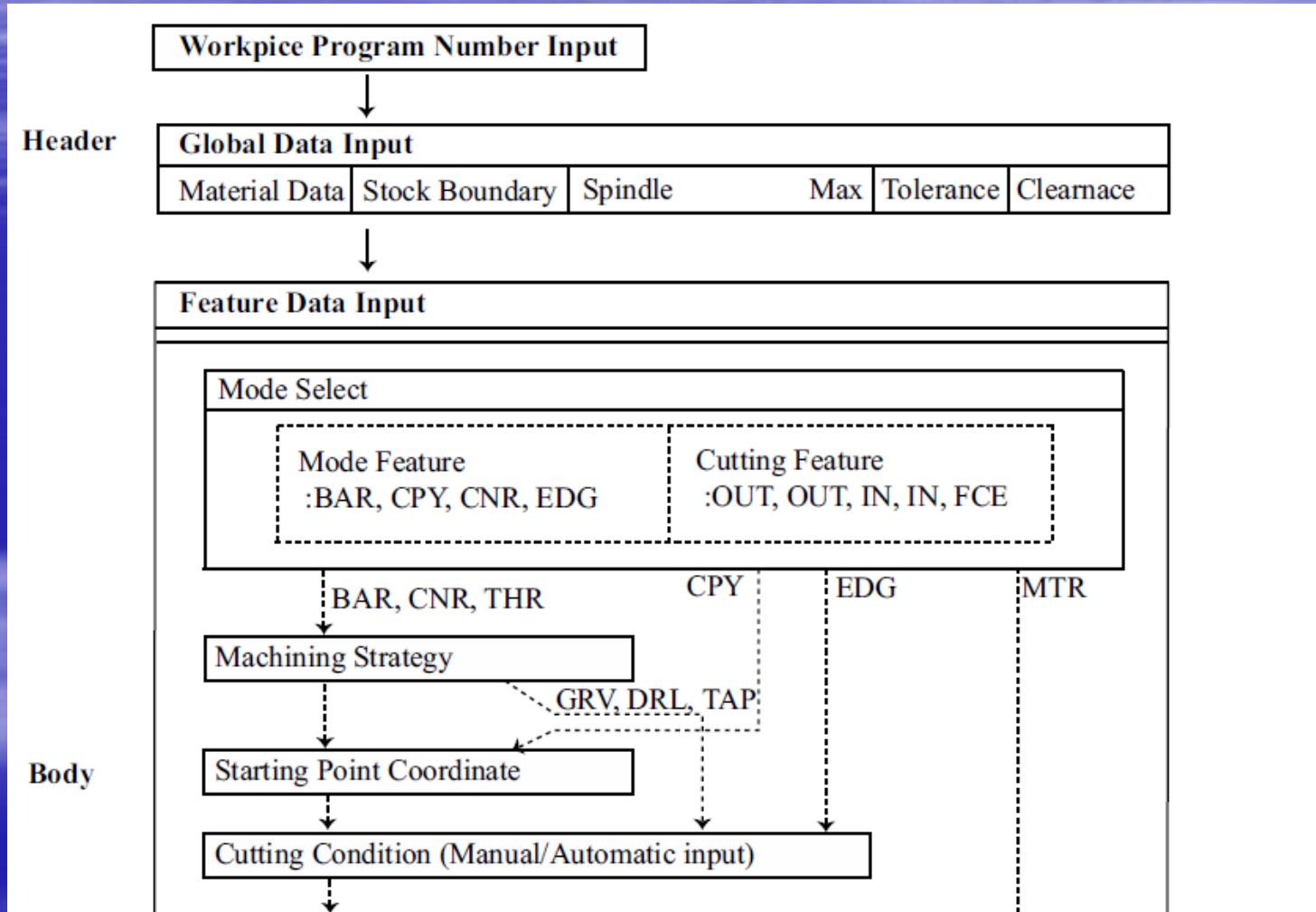
# Mazatrol Conversational System

5. *Machining Geometry*: The last step to input the feature data is to specify the machining geometry. The geometry of the feature can be made up from lines, slanted lines, convex arcs, concave arcs, and circle centers, see Fig. 8.12. The programmer selects the geometric elements that compose the feature and inputs their positions to define them fully. For each segment, surface roughness can be specified. If the programmer does not specify this, a default value, defined by a global variable, is set.

1. LIN	2. TPR	3.	4.	5.
				CTR (Center)

**Fig. 8.12** Feature geometric elements

# Mazatrol Programming Procedure



# Mazatrol Programming Procedure

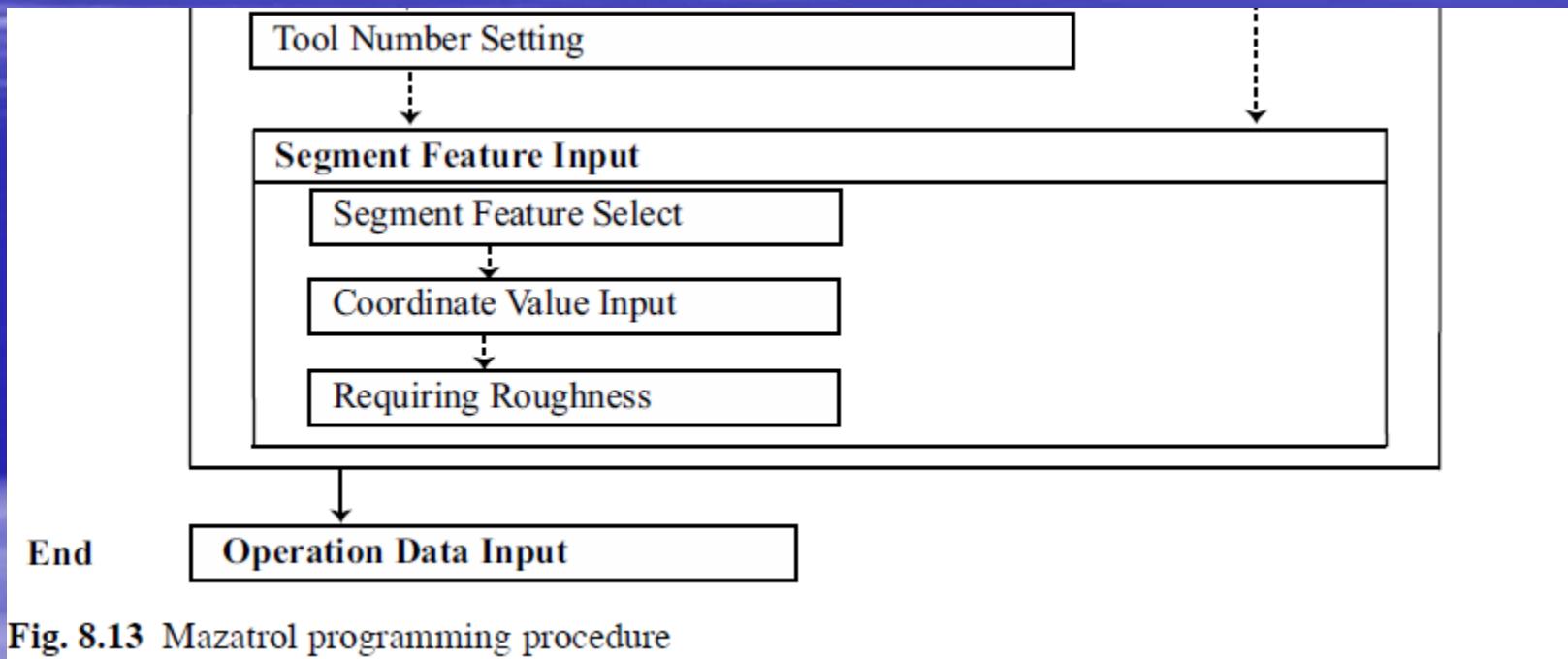


Fig. 8.13 Mazatrol programming procedure

# Conversational Programming System Design

A conversational programming system is designed as a system that:

1. can be used even by an inexperienced operator,
2. can generate a part program quickly with minimal key input,
3. can verify the generated part program in a short time,
4. can introduce the modified information easily into the generated part program,  
and
5. can be operated on the CNC system on the shopfloor.

# Conversational Programming System Design

## ■ Main Sequence for Design

1. Start the conversational programming system by selecting the programming key in the MMI.
2. Set the initial data (global data) following the screen indications generated by the Conversational Programming System.
3. Select the particular operation and input the data via the GUI (graphical user interface) relevant to the selected operation. The shape of a part, machining strategy, tool, and cutting condition are given as input data.
4. After specifying all operations, generate the part program in standard G-Code or the manufacturer's own code by selecting the program generation key in the MMI.
5. Check the tool path or the finished part via the simulator.
6. If the verification result is not satisfactory, select the modification key and modify the data of the unsatisfactory operation.

# Conversational Programming System Design

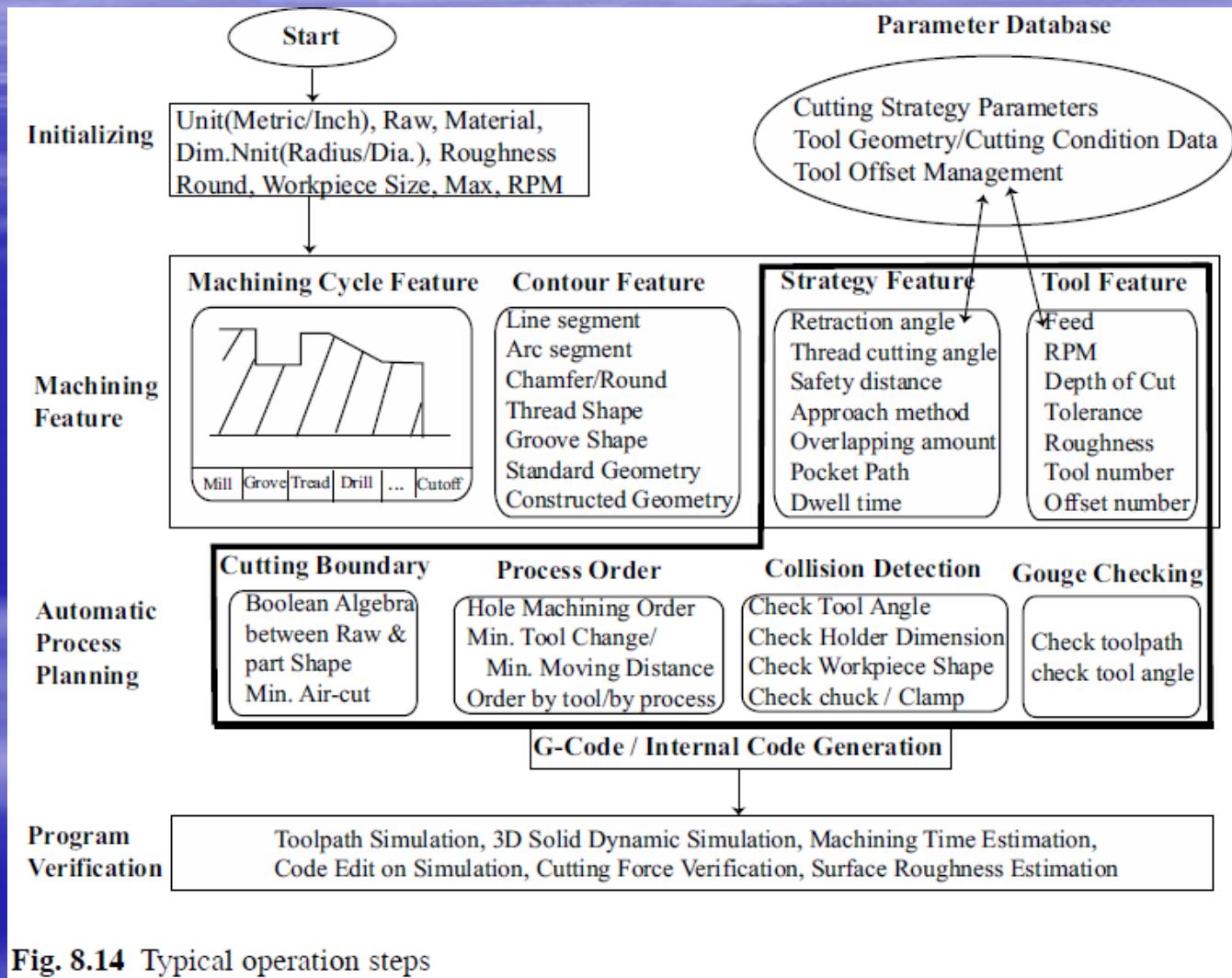


Fig. 8.14 Typical operation steps

# Conversational Programming System Design

## ■ The key points for designing a conversational programming system with an easy-to-use user interface:

1. For adequate design of the machining cycle feature that meets the machining characteristics of the machine, key functions are needed for minimizing user input by automatic recommendation of machining operations, helping non-experts to edit a program by automatic recommendation of cutting conditions, generation of the toolpath without tool interference, overcut, undercut, and aircut, and determining the operation sequence that minimizes the cutting time and tool change.
2. A unique method for specifying the part shape is needed. In order for an operator to generate a part program quickly at the machine, a simple and easy way of specifying the part shape is needed instead of an offline CAD system.
3. In addition, for realistic simulation, 3D graphics functions are needed. However, because this subject is outside the scope of this book, details of this are omitted.

# Conversational Programming System Design

## ■ Initial Setup

In the initial setup module, not only global data that is used by the CNC system but also the coordinate system (absolute/incremental), programming units (inch/metric, diameter/radius), spindle data, feed unit (mm/rev, m/min), tool retract point, tool retract strategy, workpiece material and machine specification data are defined in this module. As shown in Fig. 8.15, workpiece geometry, start Z-point, Z safety plane, work coordinates, and clamp design are defined as well. The workpiece material is used together with a tool database for calculating the cutting conditions automatically. The workpiece geometry is used for displaying the initial material on the solid simulator. The working coordinates are used as the reference coordinates during real machining. Clamp design is used by the simulator for detecting tool collisions. Therefore, in order to use the advanced functions of the conversational programming system efficiently, it is essential to carry out an initial setup before specifying the machining operation cycle.

# Conversational Programming System Design

## ■ Initial Setup

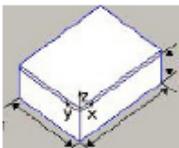
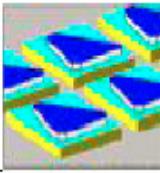
Categories	Contents
File type	Main Program / subprogram
Work material	Workpiece material is classified in terms of the hardness of material.
Work Size	Unit, Length, Width, Height, Face-off 
Initial point - Z	The Z position where tool and workpiece do not collide in the case of rapid traverse (G00).
Multi-mode	Pitch-X Pitch-Y or arbitrary points Xn, Yn  Number : N × M
Work-Zero	Work Zero offset (G54, G55, G56, ....)

Fig. 8.15 Fundamental data for machining

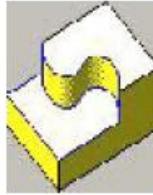
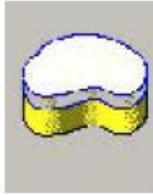
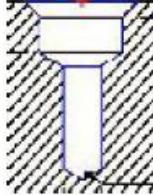
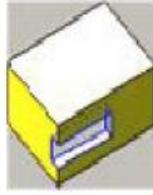
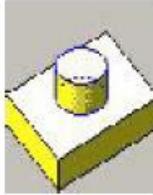
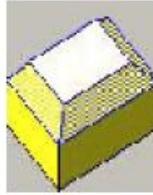
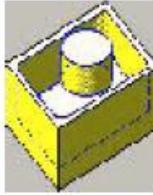
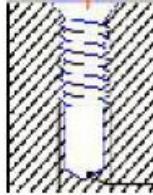
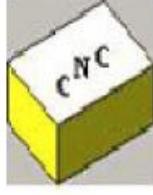
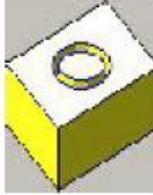
# Conversational Programming System Design

## ■ Machining Operation Cycle

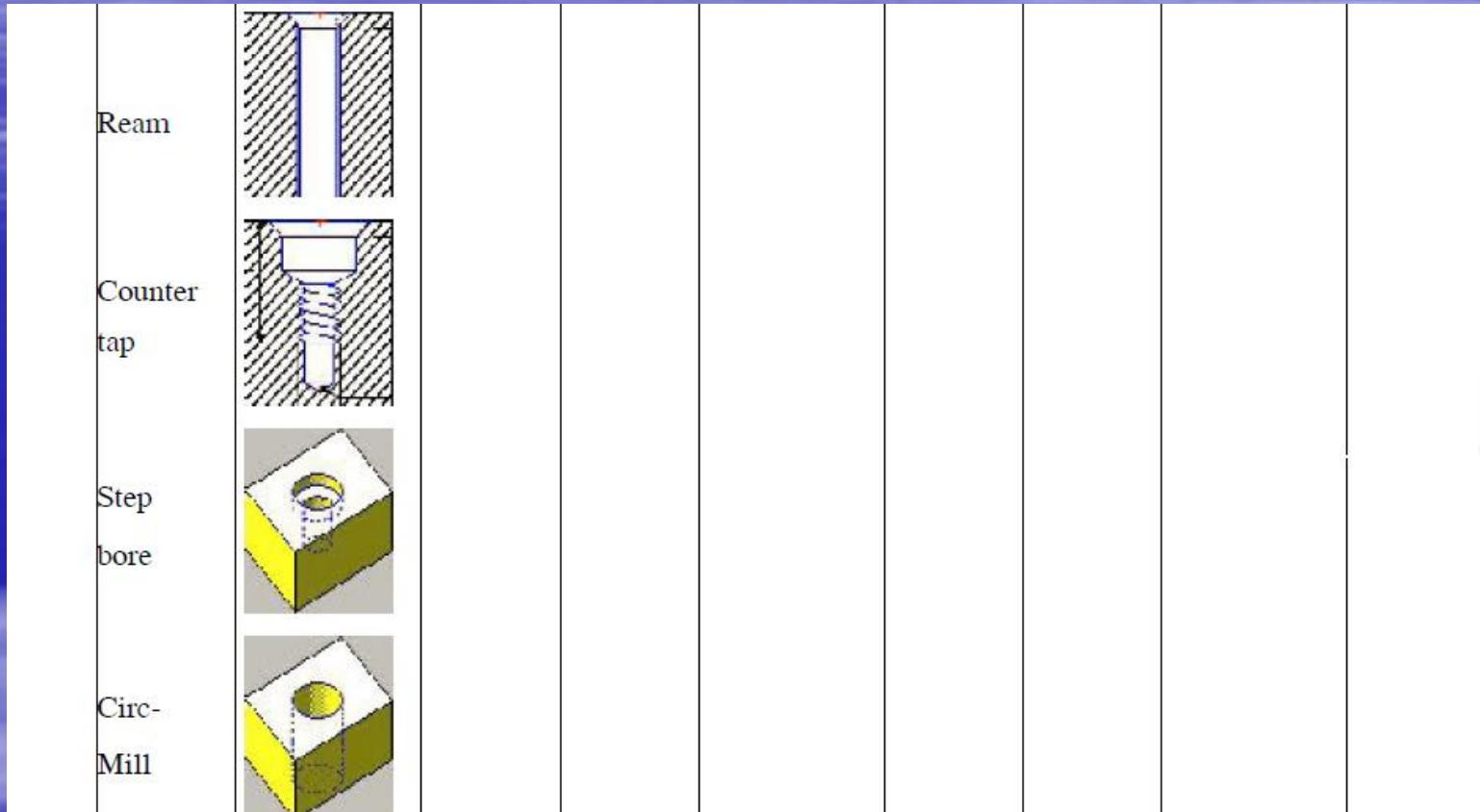
Initially, how a tool approaches a workpiece, how it is retracted from the workpiece, and how an operation is terminated are specified regardless of the types of machining operation (*e.g.*, milling, turning, and drilling). The region to be machined can be arbitrarily divided and machined.

The machining operation cycle is the code block to command roughing, finishing, drilling, or grooving. It is used for making programming simple and efficient. The machining operation cycle supports various machining strategies and is designed for minimizing user input via interactive user interface. The majority of CAM systems have CAD functions as methods to specify the part shape and features. However, with regard to shopfloor programming systems, CAD functions are inconvenient for an operator. So, for a shopfloor programming system, a different part specification method is used where machining features and operations are specified simultaneously is used.

# Conversational Programming System Design

HOLE		PROFILE			MILL		
DRILL		Center deep hole	Line		R/L/C IN/OUT	Face	
Counter bore		F/B- Cbore	Side		R/L IN/OUT	Boss	
Bore		F/B- bore	Chamfer		R/L IN/OUT	Pocket	
Tap			Engrave			Slot	

# Conversational Programming System Design



**Fig. 8.16** Classification of milling operation cycles

# Conversational Programming System Design

## ■ Input Data List for Machining Operation Cycles

**Table 8.2** Comparison between programming methods

1. Hole machining

Operation	type	Z-height	Diameter 1	Depth 1	Chamfer 1	Diameter (d2)	Depth (d2)	Pitch	Cutting Condition	
drill	center drill dp.-hl. break	O	hole dia.	hole dep.	hole cham.	x	x	x	Ts,Td, Tch: Speed, Feed	
count. bore	face back	O	c-bore Dia.	c-bore Dep.	c-bore Cham.	Drill Dia.	Drill Dep.	Bott. Cham.	x	Ts, Td Tch,Tm: S,F
Bore	face back	O	bore Dia.	bore Dep.	bore Cham.	Drill Dia.	Drill Dep.	x	x	Ts, Td, Tch ,Tb: S,F
Tap		O	Hoj.- ky.	Tap Dep.	Cham.	Drill Dia.	Drill Dep.	x	Pit.	Ts, Td, Tch ,Tt: S,F

# Conversational Programming System Design

## ■ Input Data List for Machining Operation Cycles

Ream.		O	ream. Dia.	ream. Dep.	Cham.	Drill Dia.	Drill Dep.	x	x	Ts, Td, Tch ,Tr: S,F
Count. tap		O	Hoj.- ky.	Tap Dep.	Cham.	Cbore Dia.	Cbore Dep.	Bott. Cham.	Pit.	Ts, Td, Tch,Tm, Tt: S,F
Step bore		O	Bore Dia.	Bore Dep.	Cham.	Bore Dia.	Bore Dep.	Bott. Cham.	x	Ts, Td, Tch,Tb: S,F
Cir- Mill		O	Circ. Dia.	Circ. Dep.	Cham.	Drill Dia.	Drill Dep.	Bott. Cham.	x	Ts, Td, Tch ,Tm: S,F

\* Ts: spot drill, Td: drill, Tch: chamfer, Tm: endmill, Tb: bore, Tt: thread

# Conversational Programming System Design

## ■ Input Data List for Machining Operation Cycles

Table 8.3 Comparison between programming methods

### 2. PROFILE

Operation	M-type	Start-Z	Cut-D	Cut-R	Chamfer	Fin-D	Fin-R	Cutting Condition
Line	R/L/C IN/OUT	O	O	O	O	O	O	Tr, Tf, Tch: S, F
Chamf.	R/L IN/OUT	O	kans. -D	kans. -R	X	X	X	Tch: S, F
Side Grv	R/L IN/OUT	O	O	O	O	X	O	Tr, Tf: S, F
Engr.	Font	O	Width	Height	X	Col. Span	Row Span	Tf: S, F

# Conversational Programming System Design

## ■ Input Data List for Machining Operation Cycles

**Table 8.4** Comparison between programming methods

3. Mill

Operation	M-type	Cut-R	Chamfer	Fin-D	Fin-R	Cutting Condition	
FACE		O	O	x	O	O	Tr, Tf, Tch: S,F,Dd,Dr
Boss		O	O	O	O	O	Tr, Tf, Tch: S,F,Dd,Dr
Pocket	island valley	O	O	x	O	O	Tr, Tf, Tch: S,F,Dd,Dr
Slot		O	O	Slot width	O	O	Tr, Tf, Tch: S,F,Dd,

# Conversational Programming System Design

## ■ Machining Geometry Definition

In order to specify the shape and profile of a part, another user interface, different from and easier than that of a conventional CAD system, must be provided. For this, three kinds of method for part shape specification are provided, as shown in Fig. 8.17. The first is a method to design primitive geometric elements, rectangles, polygons, and ellipses, by specifying basic parameters. This is called the “standard geometry method”. The second is a method to design the contour profile by adding lines and/or arc profiles sequentially via a graphic menu (called the oriented geometry method). The third method, which is similar to that of a 2D CAD system, is a method to design profiles by cutting, connecting and copying points, lines and arcs (called the constructed geometry method).

# Conversational Programming System Design

## Machining Geometry Definition

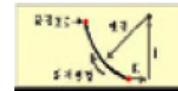
Method	Geometry				
Standard Geometry	Rectangle, Circle, Ellipse, Polygon				
Oriented Geometry		Line		Arc	
		Line-Arc		Arc-Line	
		Arc-Arc		2 Points	
		2 Angles			
Constructed Geometry	Point (Cartesian/polar, circular, matrix, line) Line (parallel, perpendicular, tangent...) Circle (center + R, 3 line,...)				

Fig. 8.17 Part-shape specification methods

# Conversational Programming System Design

## ■ Tool/Technology Data

The Tool/Technology database recommends the appropriate cutting conditions in terms of the tool and workpiece material and the tool shape. Four kinds of database are supported for the tool used and tool sequence for hole machining.

1. Tool database: As shown in Fig. 8.18, this manages the data about tool shape and material. It provides the data for generating toolpaths and deciding cutting conditions.
2. Cutting condition database: this manages the data about cutting speed and feed according to tool type, material, and workpiece material.
3. Tool sequence database: This manages the machining sequence for efficient hole machining.
4. Tool offset database: This manages the data about tool offset.

# Conversational Programming System Design

## ■ Tool/Technology Data

Tool	Feature	Tool data	Operation to which tool is applied
Center		Material, diameter, length, point, angle	Hole
Chamfer		Material, diameter, length, point, angle	Profile-chamfer
Drill		Material, diameter, length, point, angle	Hole
Bore		Material, diameter, length	Hole
Tap		Material, diameter, length	Hole

# Conversational Programming System Design

## ■ Tool/Technology Data

Reamer		Material, diameter, length	Hole
Face mill		Material, diameter, length, cutting teeth number	Mill-face
End mill		Material, diameter, len., flute num., tool type (flat, ball), ball radius	Hole, Profile, Mill
Side mill		Material, diameter, len., cutting teeth num., cutter length	Profile-side

Fig. 8.18 Milling tool database

# Conversational Programming System Design

## ■ Machining Strategy Data

To generate toolpaths using an operation cycle, it is necessary to decide the machining strategy once the machining features and tools have been decided. In Table 8.5, the data about the machining strategy that should be considered for milling operations are summarized. The usage of these for each operation cycle (hole/profile/milling or ‘H’, ‘P’, ‘M’ in Table 8.5) is checked.

# Conversational Programming System Design

**Table 8.5** Machining strategy data

Parameters	Description	H	P	M	Remark
Return type	Tool retraction method (XY/Z, XYZ)	O	O	O	
CLD	Allowance amount when tool approaches	O	O	O	
RTD	Tool retraction distance	O	O	O	
Chamfer all. -R	Chamfer allowance along radial direction	O	O	O	
Chamfer all. -D	Chamfer allowance along depth direction	O	O	O	
Spot Depth	Depth of Spot Drill	O	X	X	
Through Hole	Selection of stop hole or through hole	O	X	X	
Through all.	Backside allowance of through hole	O	X	X	
Dwell	Dwell Time at bottom of hole	O	X	X	
Relief	Return relief amount in case of drilling and boring	O	X	X	Drill/Bore

# Conversational Programming System Design

Feed factor	Retract feed factor	O	X	X	Bore/Reamer
Mill type	Exact arc processing method	O	X	X	C bore/Cir-Mill
Tool end allowance	Tool end allowance in Boring	O	X	X	Bore
Tool end allowance 2	Tool end allowance in Back-Boring	O	X	X	Bore
Finishing allowance	Allowance of bottom in boring	O	X	X	Back C-Bore/ Bore
Stop Hole allowance	Drilling depth allowance for threading, boring or reaming	O	X	X	Bore
Incomplete thread num.	Number of incomplete threads in tapping	O	X	X	
Interference	Checking the interference before machining (the number of blocks ahead that are examined)	X	O	O	
Corner	Machining method at edge. (Round/sharp/square/trang/fanuc)	X	O	O	

# Conversational Programming System Design

Cycle path	Straight/Zigzag with or without retract	X	X	O	Face
Pock path	para-contour/para-axial	X	X	O	
Cut type	Down/upward machining	X	X	X	
Direction	machining direction: any/ CW/CCW	X	X	X	
Run-in	Approach: no/Arc/ Parallel/Perpendicular	X	O	X	

# Conversational Programming System Design

**Table 8.5** (continued)

Run-out	Approach: no/Arc / Parallel/Perpendicular	X	O	X	
Face R feed factor	Feed factor in radius direction.	X	X	O	Face
Face R allowance	Facing allowance in radius direction.	X	X	O	Face
Face allowance	Removal rate in radius direction	X	X	O	Face
Pock R feed factor	Feed factor in radius direction	X	X	O	Pocket
Boss outer Ffac	Feed factor for machining outside of Boss	X	X	O	Boss
Axial D-Ffac	Feed factor for machining in axial direction	X	X	O	Boss, Pocket
Pock R-Fac	Feed factor for full slot cutting in the case of pocketing.	X	X	O	Pocket
Overlap amount for closed shape	Overlap amount when approaching and retracting in closed shape	X	O	O	

# Conversational Programming System Design

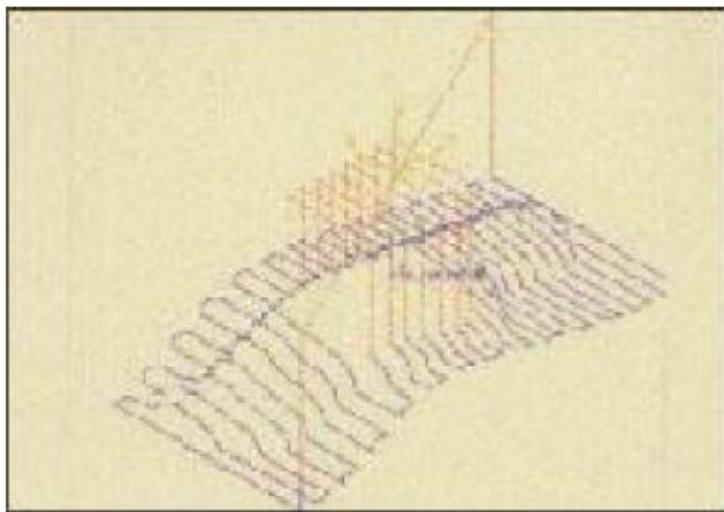
## ■ Graphic Simulation for Verification

Graphic simulation is carried out by a path simulator for verifying tool paths and a solid simulator for verifying the machined shape. A path simulator displays toolpaths as a sequence of lines or arcs and is used for visual verification of the toolpath of a part program (see Fig. 8.19a). It provides the functions for checking for collisions between tools and clamps and editing a part program for correcting incorrect tool paths.

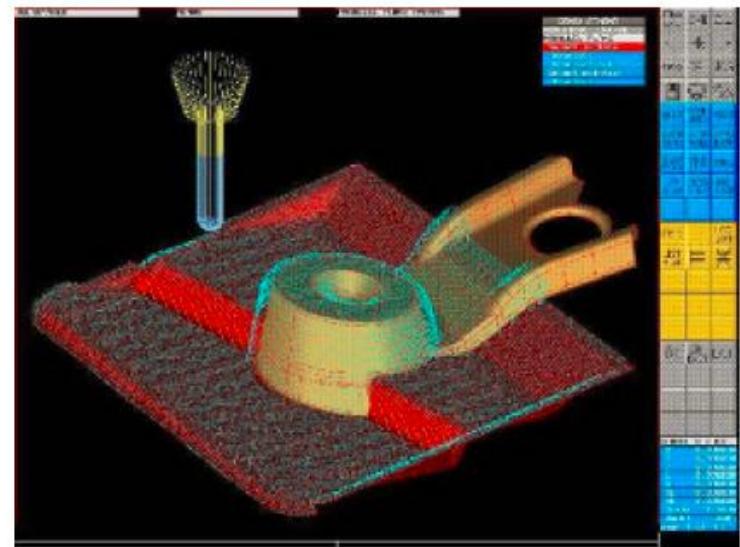
A solid simulator shows the change of part shape of a 3D solid model during machining. Also by using a solid simulator, it is possible to verify tool paths and analyze realistically the machined part (see Fig. 8.19b).

# Conversational Programming System Design

## ■ Graphic Simulation for Verification



(a)



(b)

**Fig. 8.19** Graphical simulation

# Conversational Programming System Design

## ■ Operation Sequence Control

This module shows the specified operation cycles and enables an operator to modify and delete them while editing the operation cycle. It also enables addition of new operation cycles and operation sequence changes. It enables operators who are unfamiliar with process planning to generate consistent and efficient programs. Moreover, it is possible to store the generated program on memory or disk and use it whenever it is needed.

# Development of the Machining Cycle

## ■ Turning Fixed Cycle

From the programmer's point of view, it is necessary that frequently used machining operations are defined in fixed format and used like subprograms when a part program is edited. A series of machining operations that are used repeatedly in NC machining are defined as one block that is called a “fixed cycle”. The fixed cycle for turning can be classified into two types as shown in Table 8.6. Figure 8.20 shows G92, which is the simple fixed G-code for threading, and G76, which is the complex fixed G-code for threading. Compared with the tool path of the simple fixed cycle, that of the complex fixed cycle is complicated. However, it is relatively simple to generate the toolpath from the input data.

# Development of the Machining Cycle

## ■ Turning Fixed Cycle

**Table 8.6** Machining strategy data

Type	Code	Description	type	Code	Description
Single Fixed Cycle	G90	Turning (Cutting Cycle A)	Complex Fixed Cycle	G70	Finishing
	G92	Thread cutting		G71	Outer turning
	G94	Facing (Cutting Cycle B)		G72	Facing rough
				G73	Pattern repet.
				G74	Peck drilling in Z-axis
				G75	Grooving in X-axis
				G76	Thread cutting

# Development of the Machining Cycle

## ■ Turning Fixed Cycle

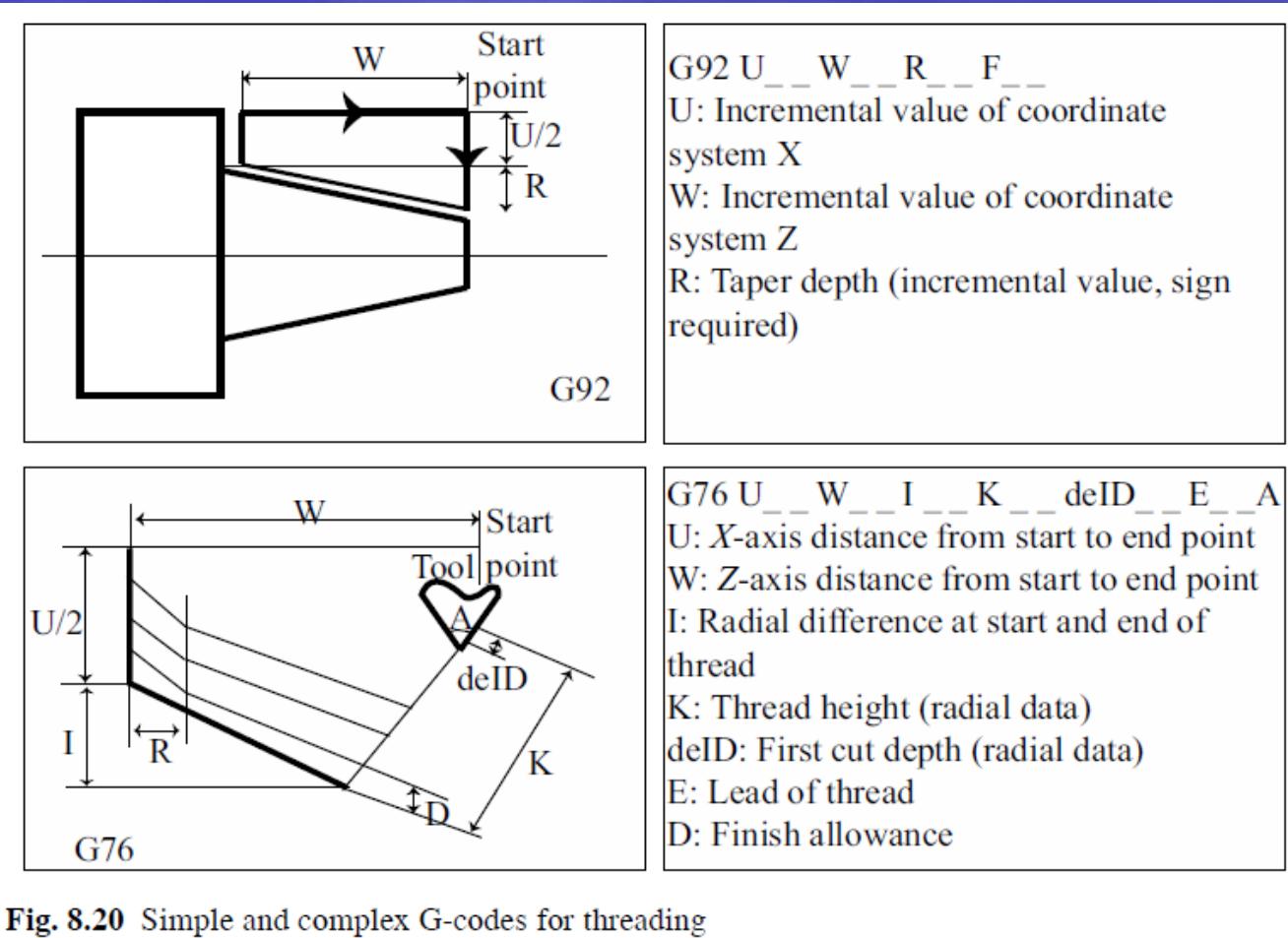


Fig. 8.20 Simple and complex G-codes for threading

# Development of the Machining Cycle

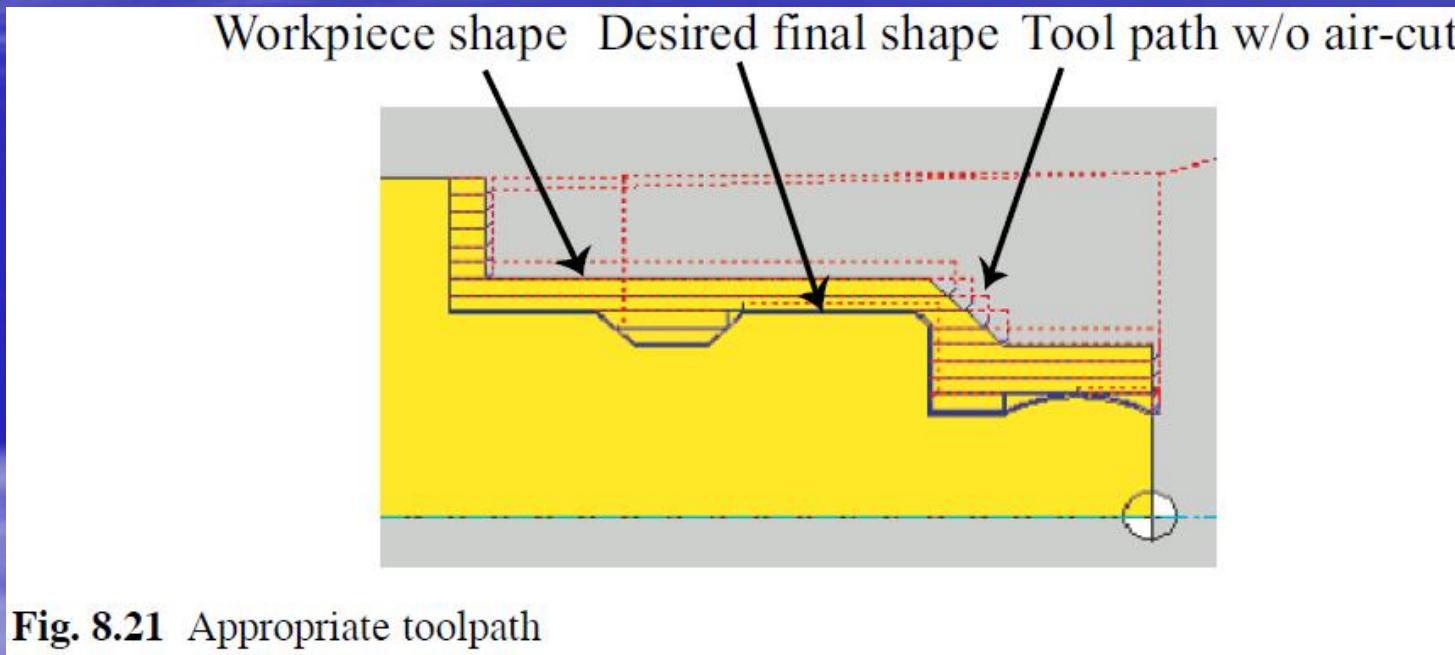
## ■ Turning Cycle for Arbitrary Shape

The G-code cycles mentioned in the above section are used for generating tool paths for a cylindrical part. In order to apply them successfully it should be assumed that the radius of a part increases or decreases consistently and that tool interference does not occur. However, a forged part or cast part typically has arbitrary shape and, in this section, the roughing cycle for these will be addressed. The roughing cycle generates the toolpath without tool interference by considering the geometry of the tool. It does not generate toolpaths for regions where material is absent in order to prevent cutting air. If there is a region where tool interference cannot be avoided, the region is not cut and remains to be cut in a subsequent operation.

For example, as shown in Fig. 8.21, the dotted line that represents the toolpath without air-cut is an appropriate tool path for obtaining the finished part from a cast workpiece.

# Development of the Machining Cycle

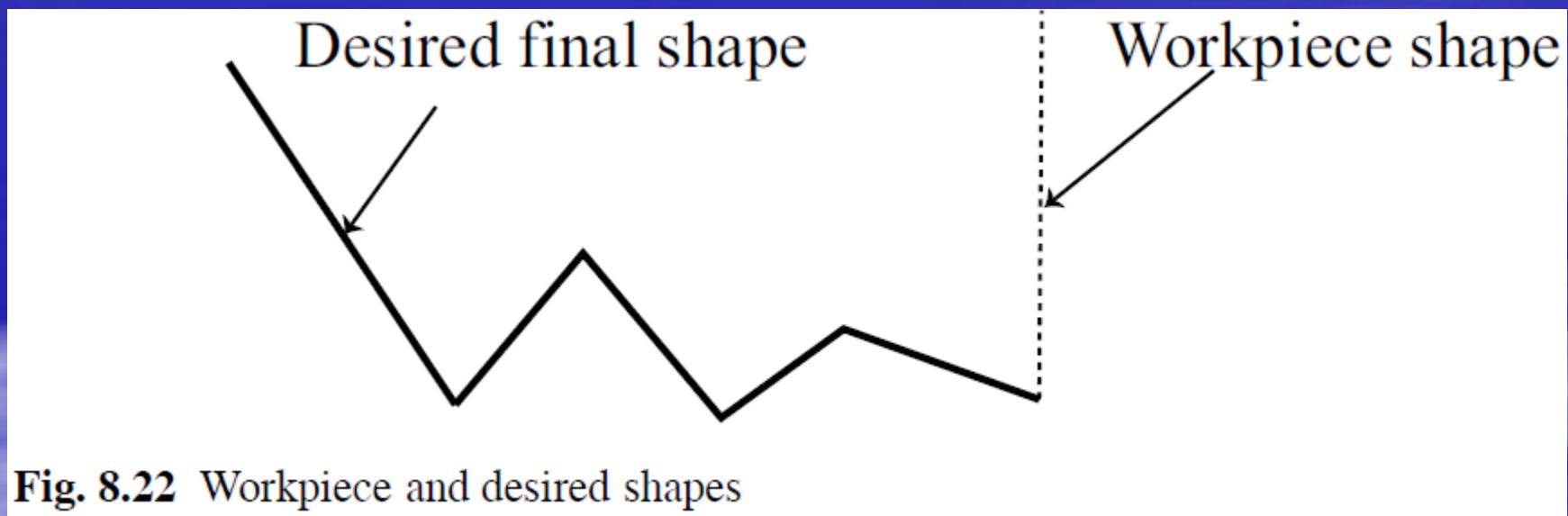
## ■ Turning Cycle for Arbitrary Shape



# Development of the Machining Cycle

## ■ Toolpath Algorithm

In the first step, the workpiece shape and desired shape are specified (Fig. 8.22).



# Development of the Machining Cycle

## ■ Toolpath Algorithm

In the second step, the collision-free region (machineable region) is calculated based on the cutting edge angle (side cutting edge angle and end cutting edge angle) of the tool, cutting angle, tool imaginary nose, tool type, tool holder's shape and workpiece shape, see Fig. 8.23.

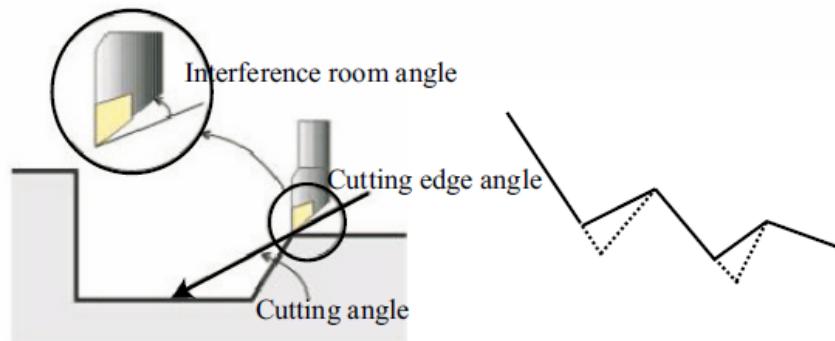


Fig. 8.23 Collision-free region calculation

The cutting angle is calculated as follows, based on the cutting edge angle and interference room angle.

$$\text{Cutting angle} = \text{cutting edge angle} - \text{interference space angle} \text{ (in general, 3 or 5 degrees)}$$

# Development of the Machining Cycle

## ■ Toolpath Algorithm

In the third step, the offset profile is generated by offsetting the machineable region from the second step within the finish allowance and with the tool nose radius.

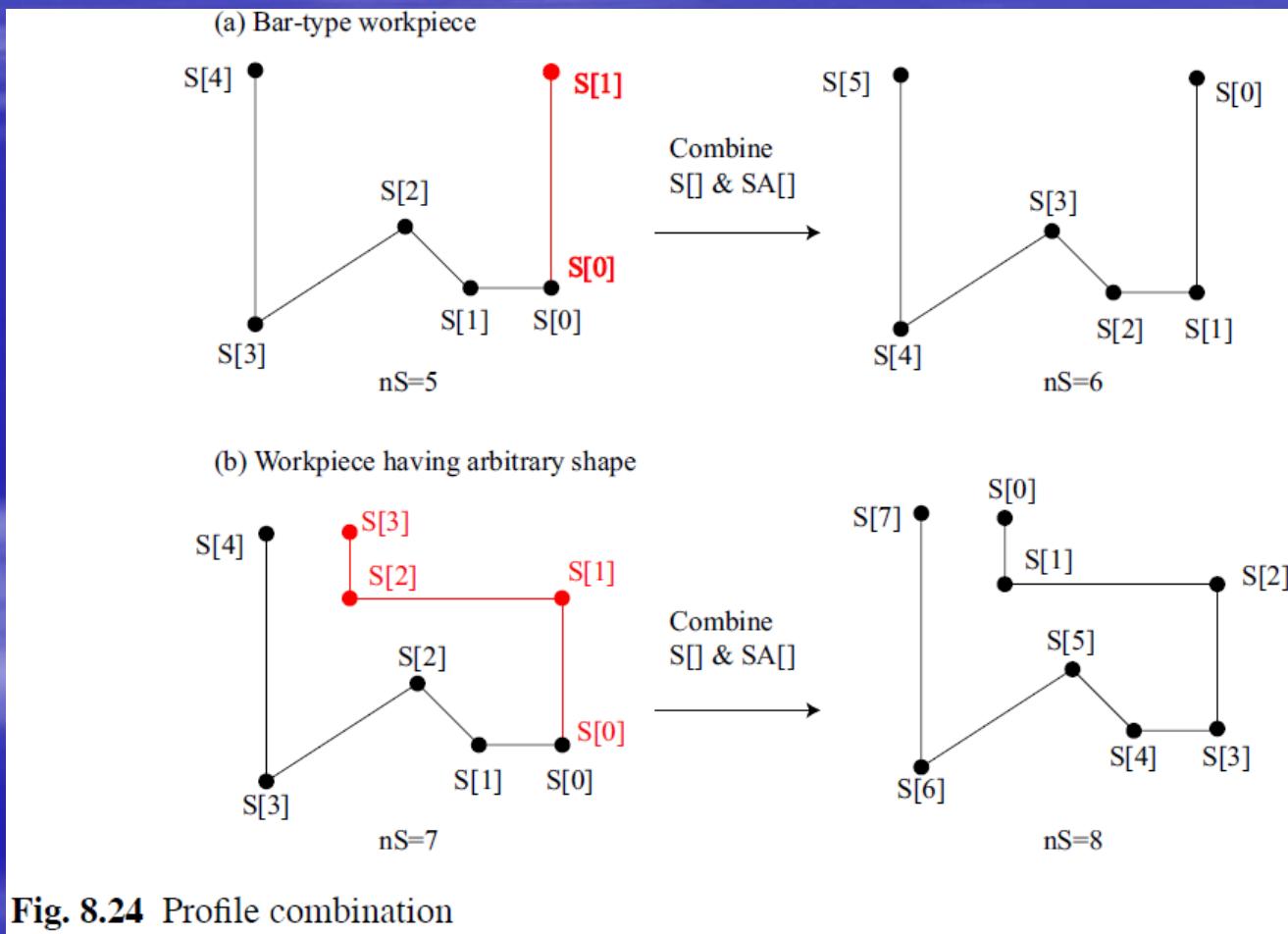
# Development of the Machining Cycle

## ■ Toolpath Algorithm

In the fourth step, by combining the offset profile with the original profile of the part, a new profile is generated. In the case where the blank material is a cylinder, a new profile is generated by adding the linear profile of the cylinder,  $SA[]$ , to the offset profile,  $S[]$ , as shown in Fig. 8.24a. In the case where the blank material has an arbitrary shape, as with a cast part, the profile to be machined is created by combining the profile of the part,  $SA[]$ , with the offset profile,  $S[]$ , as shown in Fig. 8.24b.

# Development of the Machining Cycle

## Toolpath Algorithm



# Development of the Machining Cycle

## ■ Toolpath Algorithm

In the fifth step, peak points and valley points are sought from the S[] obtained from the fourth step, and the total number of peak points is counted.

Peak point ( $P_i$ ):  $\{ P_i | X_i \geq X_{i-1} \text{ and } X_i > X_{i+1}, \forall i \}$

where,  $i$  is the index of a point on the profile.

If  $X_i = X_{i-1}$  and  $X_i < X_{i-2}$ ,  $P_i$  is not peak point.

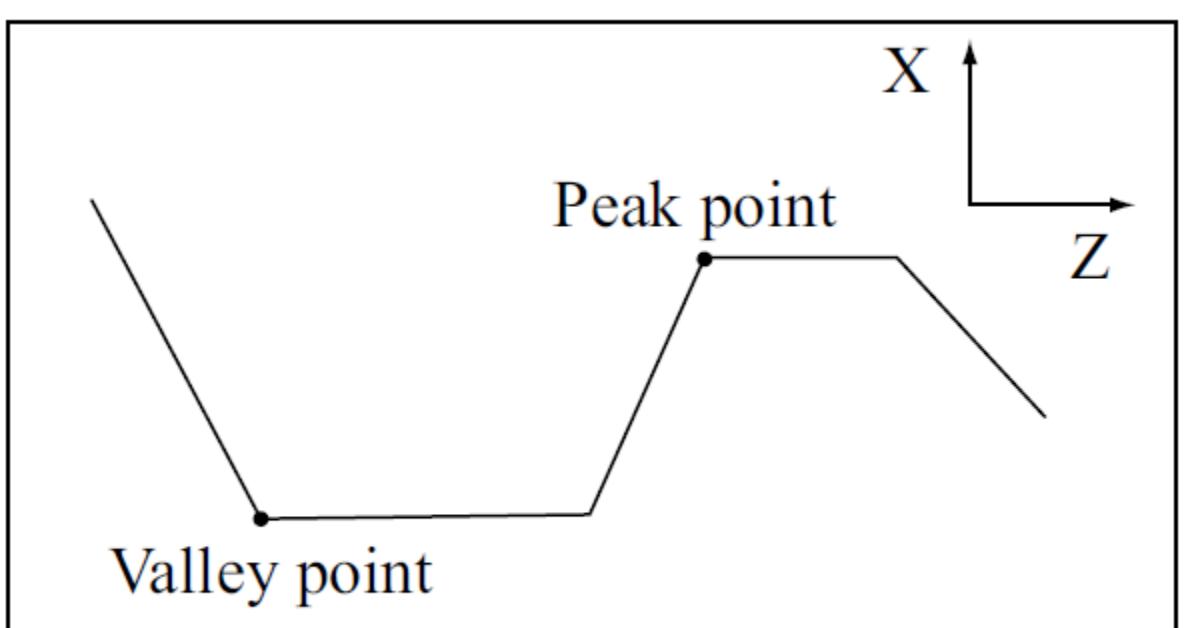
Valley point ( $V_i$ ):  $\{ V_i | X_i \leq X_{i-1} \& X_i < X_{i+1}, \forall i \}$

where,  $i$  is the index of a point of the profile.

In Fig. 8.24a,  $S[3]$  is a peak point and  $S[1]$  is a valley point. In Fig. 8.24b,  $S[5]$  is a peak point and  $S[4]$  is a valley point.

# Development of the Machining Cycle

## ■ Toolpath Algorithm



**Fig. 8.25** Peak and valley points

# Development of the Machining Cycle

## ■ Toolpath Algorithm

In the sixth step, the profile from the fourth step is divided into multiple profiles at the valley points and the divided profiles are stored in a buffer. In the case of the profile shown in Fig. 8.24a, the profile is divided at valley point S[1] as shown in Fig. 8.26.

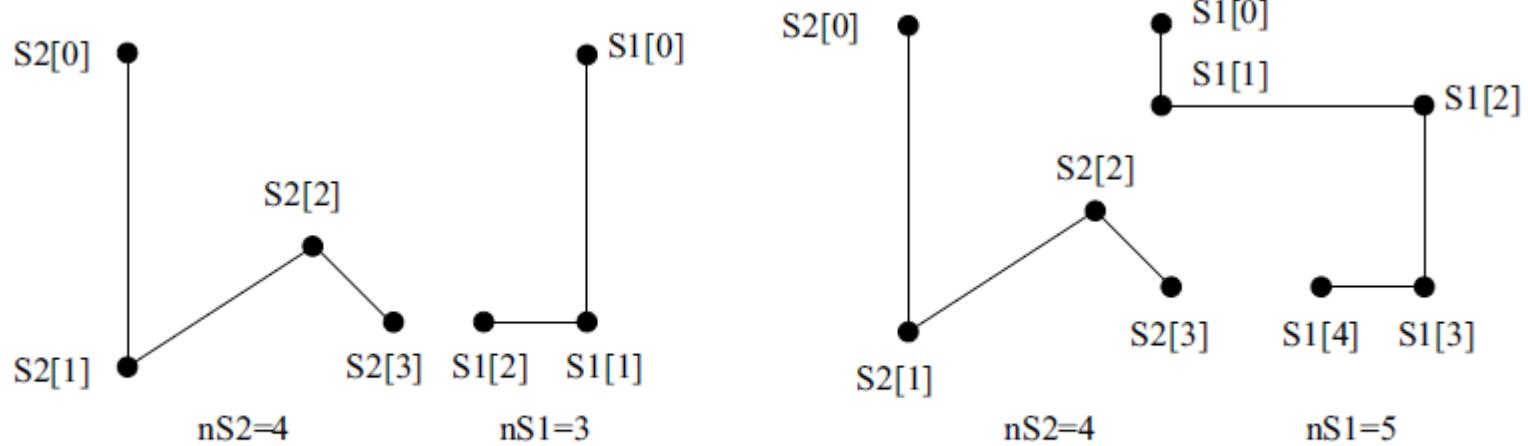


Fig. 8.26 Divided profiles

# Development of the Machining Cycle

## ■ Toolpath Algorithm

In the seventh step, the toolpath is generated based on the divided profiles and the specified cutting depth, as shown in Fig. 8.27a.

```
Stock_removal_path () {
```

- The peak points are sorted in terms of the X position and stored in peak\_array[].
- The number of cutting layers is calculated ( $num = (X_e - X_s)/feed + 2$ ).
- From S1[] (first layer) and cutting depth are calculated the intersection point, cross\_S1[] and cross[].

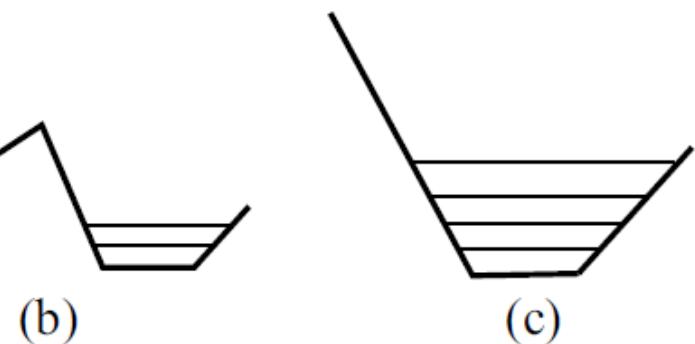
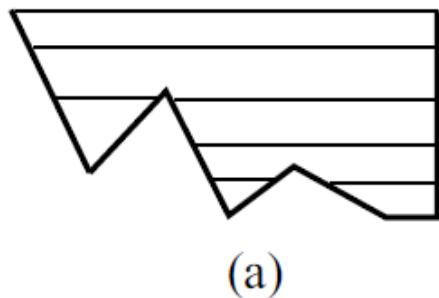
Where, cross[] is the path whose X position is constant and cross\_S1[] is the intersection point between cross[] and S1[].

- From S2[] and cross[], cross\_S2[] is computed.
- The tool path is generated based on cross\_S1[], cross\_S2[], S1[], and S2[].

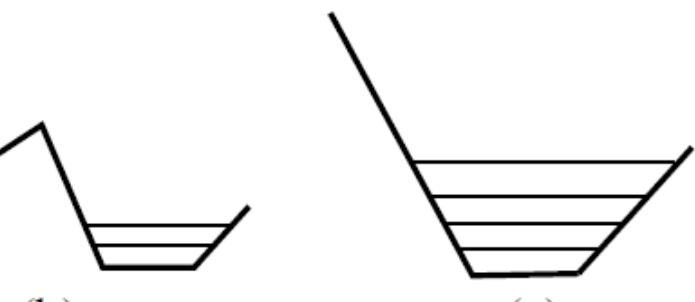
```
}
```

# Development of the Machining Cycle

## ■ Toolpath Algorithm



(b)



(c)

**Fig. 8.27** Cutting toolpaths for turning

# Development of the Machining Cycle

## ■ Toolpath Algorithm

The eighth step, after the above steps have been completed, checks whether the current machined profile is the last peak profile. If yes, the cycle is terminated and, if not, the region to be machined is recalculated as shown in Figs. 8.27b and 8.27c.

# Development of the Machining Cycle

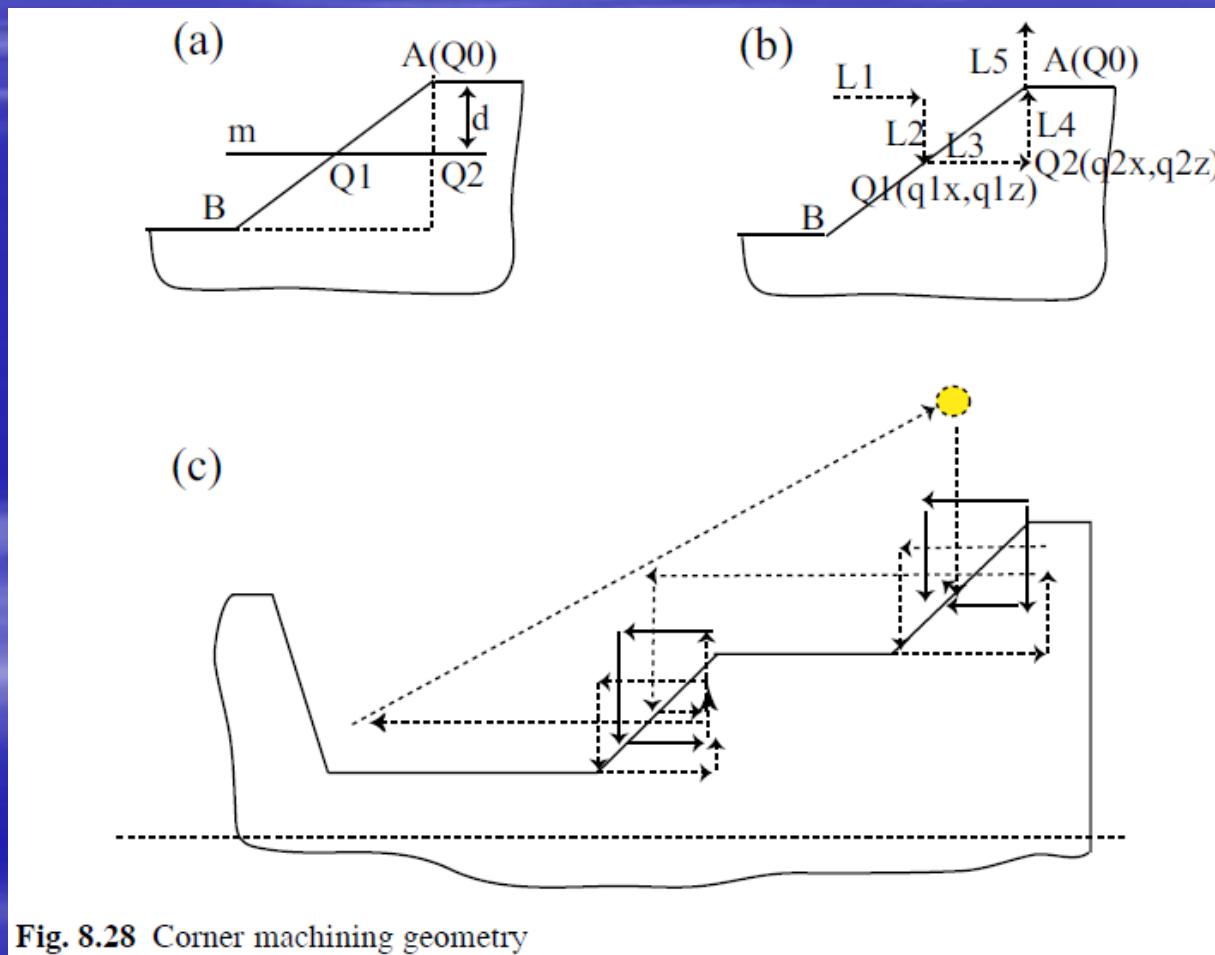
## ■ Corner Machining Cycle

As mentioned in the previous section, an uncut area due to the tool shape can remain after completing the machining by the specified tool in the case of turning. This is why a toolpath is not generated in a region in which tool interference occurs. Therefore, in order to machine the uncut region after roughing, partial machining is executed after selecting a different tool. In the case of turning this is called “corner machining”.

The toolpath for corner machining is generated based on the intersection points (A and B) between the uncut area and the finished shape, cutting depth  $d$ , and finishing allowance  $k$ . The details of the algorithm can be summarized, with Fig. 8.28, as follows:

# Development of the Machining Cycle

## ■ Corner Machining Cycle



# Development of the Machining Cycle

## ■ Corner Machining Cycle

This algorithm can be summarized as the procedure chart in Fig. 8.29.

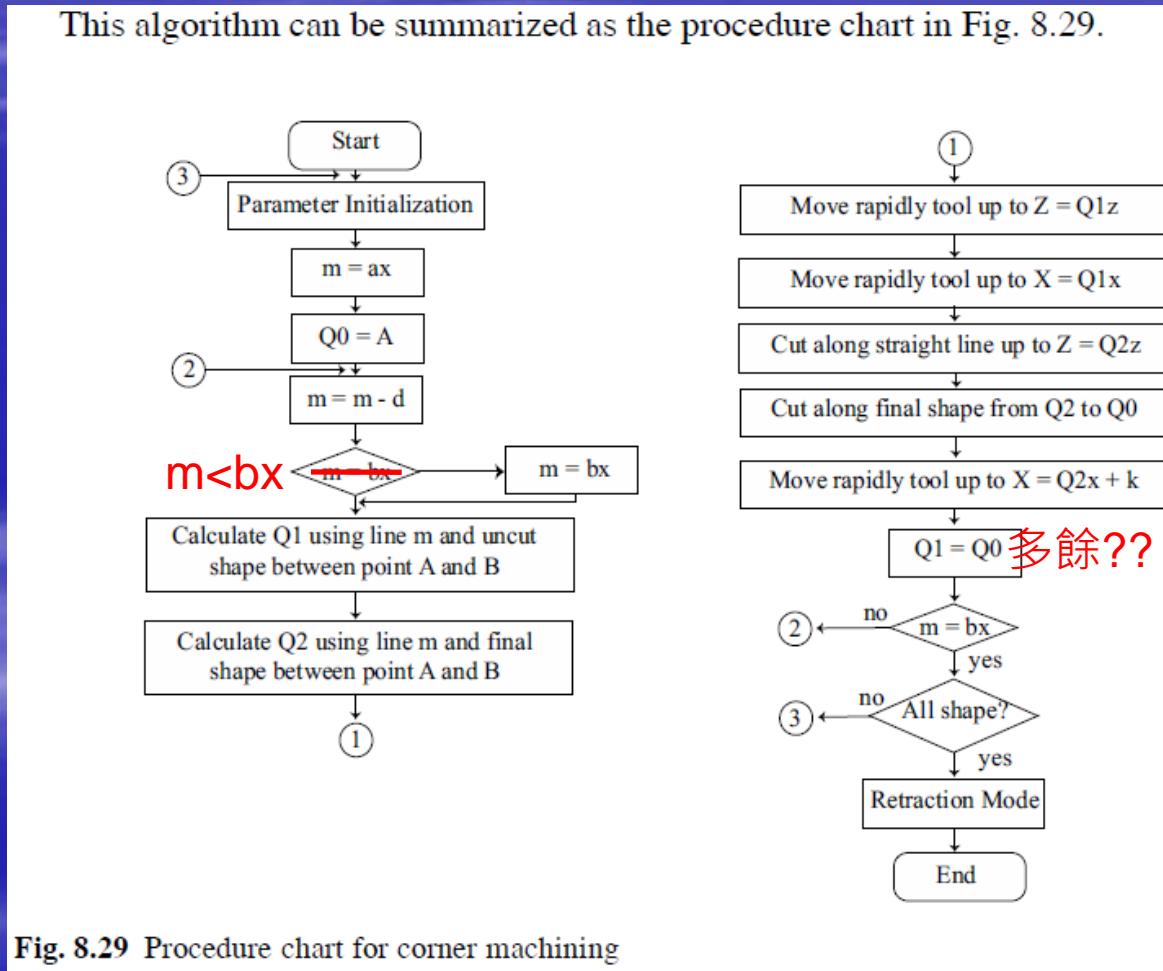


Fig. 8.29 Procedure chart for corner machining

# Development of the Machining Cycle

## ■ Drilling Sequence

Typically, when the NC program for drilling multiple holes is generated, the programmer first classifies the holes into the groups depending on the hole shape and generates a program where the holes belonging to the same group are machined in a row. After completing machining of the holes in the group, holes belonging to another group are machined. In order to machine a hole, it is typical to use more than one tool. For example, in the case of tapping, center drilling, drilling, boring, and tapping should be executed one after the other. Therefore, if we consider the usage of tools related to drilling, the sequence of tools is considered as follows:

Group 1:  $T_{11}, T_{12}, \dots, T_{1a}$

Group 2:  $T_{21}, T_{22}, \dots, T_{2b}$

Group M:  $T_{m1}, T_{m2}, \dots, T_{mm}$

where tools used in one particular group may be used in another group. Therefore, if the usage sequence of tools is well determined, it is possible to decrease the number of tool changes and hence the machining time.

# Development of the Machining Cycle

## ■ Drilling Sequence

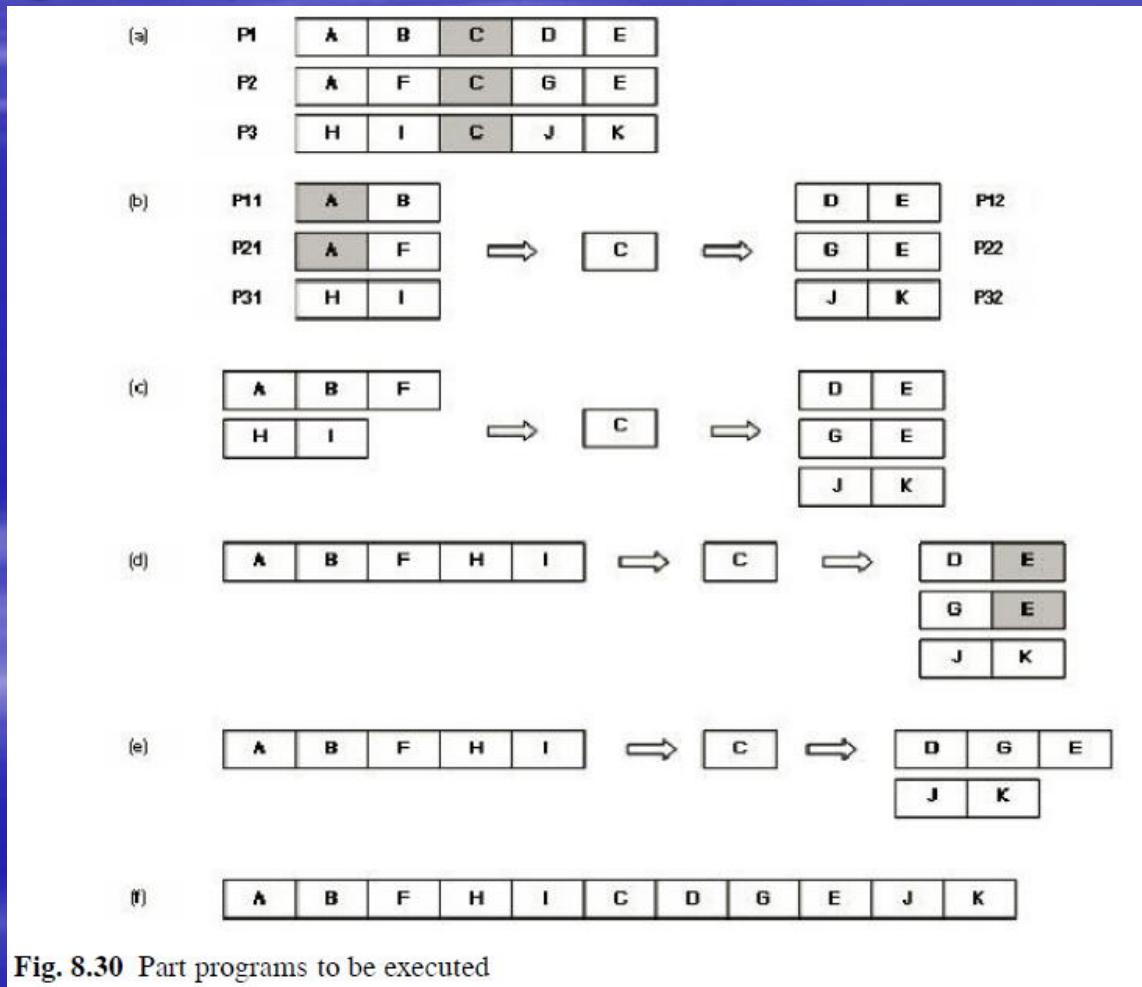


Fig. 8.30 Part programs to be executed

# Summary

The part that provides the user interface in the CNC system is the MMI unit. For the design of an MMI that allows a user to edit a part program, operate the machine, and monitor the machine status, it is necessary to consider ergonomics, design, and the user's aptitude as well as technological aspects.

In order to develop an MMI system, not only the user applications but also the design of the kernel layer for connecting to the NCK and the applications for monitoring the machine status, operating the NC, editing the program, and managing parameters are required.

# Summary

The program-editing function, which is one of the key MMI functions, has advanced in a variety of versions developed by many CNC makers. Recently, conversational programming systems that can be used on the shopfloor have become a basic programming tool. The conversational programming system makes it possible for an unskilled user to generate a part program by allowing users to input data in an interactive way.

The core of the conversational programming system provides functions for generating interference-free toolpaths whose machining time is minimized and in which the number of tool changes is minimized, verifying tool paths and the finished part, and estimating machining time based on the minimum user input.

# Theory and Design of CNC Systems

*Chapter 9*  
**CNC Architecture Design**

# Introduction

1. Real-time operating system (the kind of system call and usable hardware)
2. The architecture of hardware (the number of CPUs and the way of connecting between modules)
3. The architecture of the software (the design of task modules and the system kernel)

# Operating Systems

- An operating system is a software program for the purpose of using the hardware efficiently.
- A **multi-programming** system increases the throughput of the processor by loading several user applications simultaneously into main memory, executing different user applications in turn using one processor
- As a **time-sharing** system is the extension of the multi-programming concept, several tasks are carried out by one processor in turn by giving them specified time slices

# Operating Systems

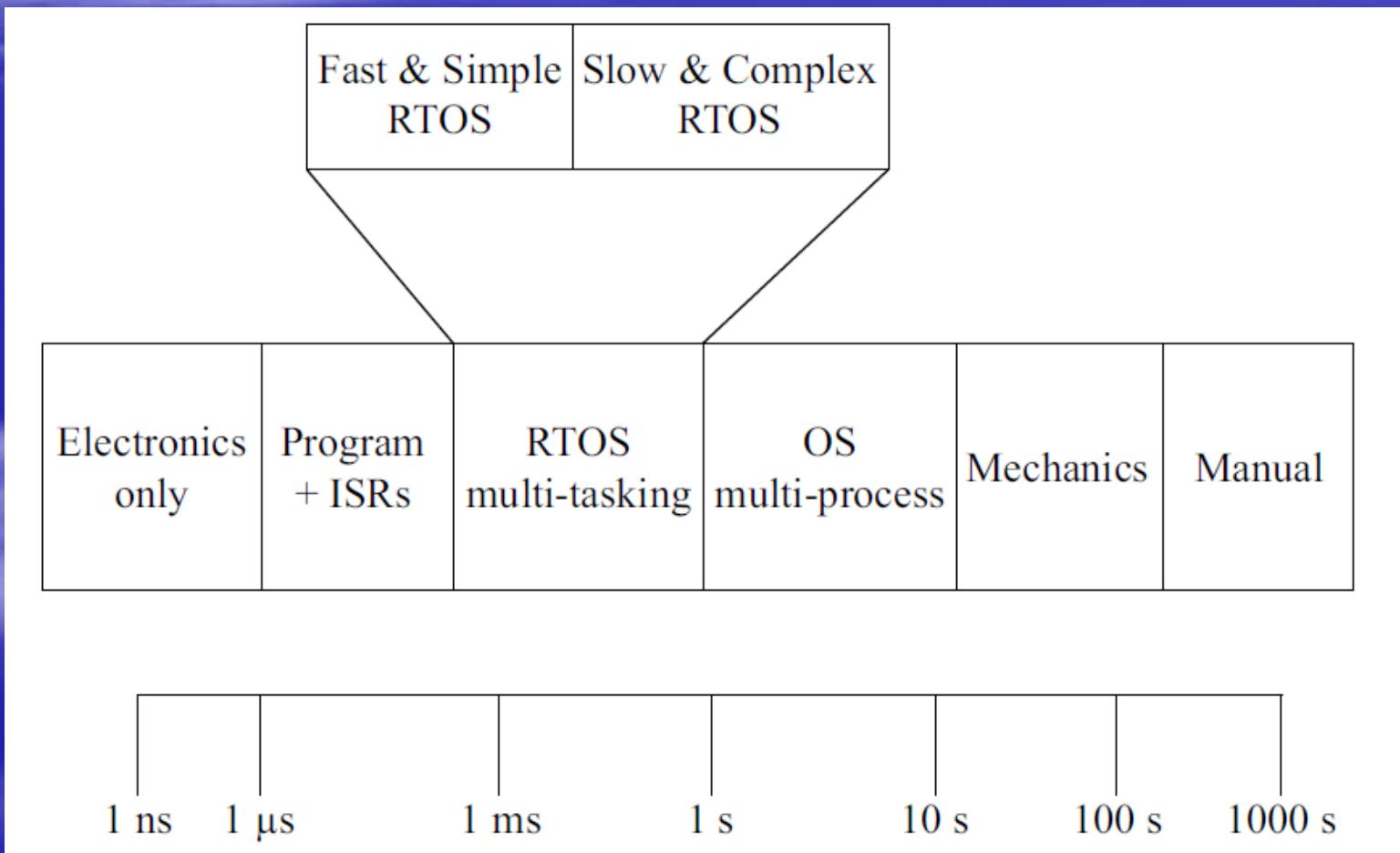
- A **multi-processing** system is used to overcome the drawback of a single processor system, it is a system where multiple processors carry out tasks collaboratively through communication and memory sharing in order to improve the system performance and reliability
- The term “**distributed system**” denotes a system that carries out a task by using simultaneously multiple executors such as node, site, and computer in order to share the resources of multiple computers, increase the computing power by means of parallel and distributed execution, and increase the reliability of a system. In this type of system, data communication between computers is done via external communication cables

# Operating Systems

- “**real-time system**” denotes a system that can begin the desired tasks within a specific time and complete it within a given time period, it is necessary that the response time (operational deadline) from event to system response is smaller than a specific time period. The ultimate goal of a real-time system is handling the data regularly occurring in real time or promptly handling occasional events

# Operating Systems

- Time spectrum of a real-time system



# Operating Systems

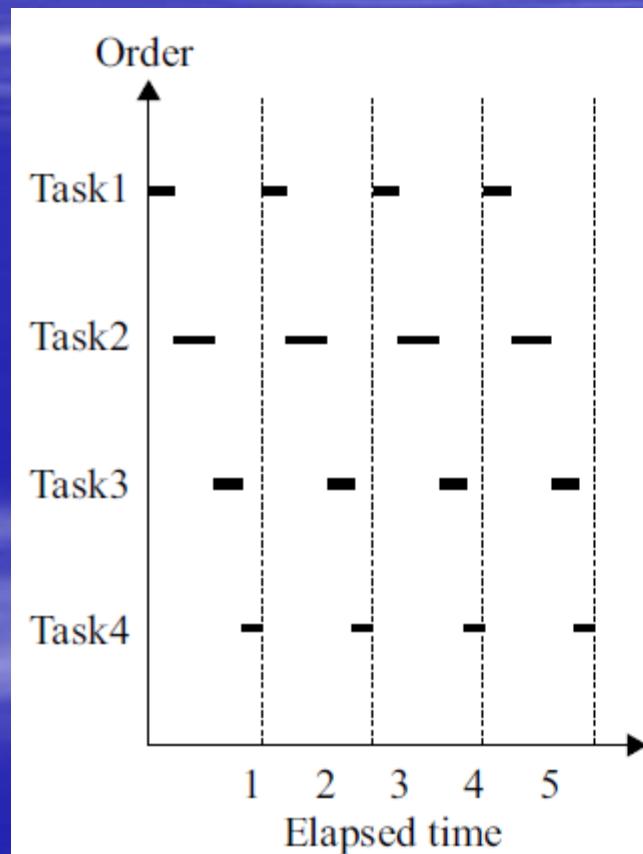
- Real-time systems can be divided into two types: **hard real-time** systems and **soft real-time** systems
- a hard real-time system executes iteratively within **several milliseconds** or **several tens of milliseconds** and never miss a deadline
- a soft real-time system execute iteratively within **several tens** or **several hundreds of milliseconds** and does not miss a deadline

# Operating Systems

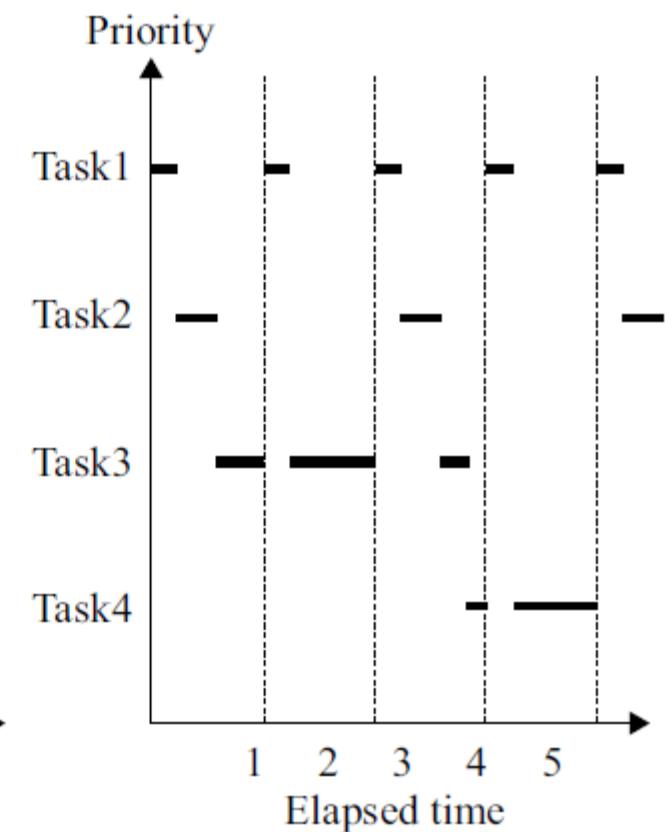
- **CNC system** can be defined as a **hard real-time** system. Generally, position control of a CNC system is repeated every few milliseconds or every few hundreds of nanoseconds and delay in position control stops the machine and results in a difference between the programmed path and the actual path. Therefore, the OS for a CNC system must be a real-time OS that guarantees the hard real-time property on a single processor or multiple processors.

# Real-time Programming

Sequential programming



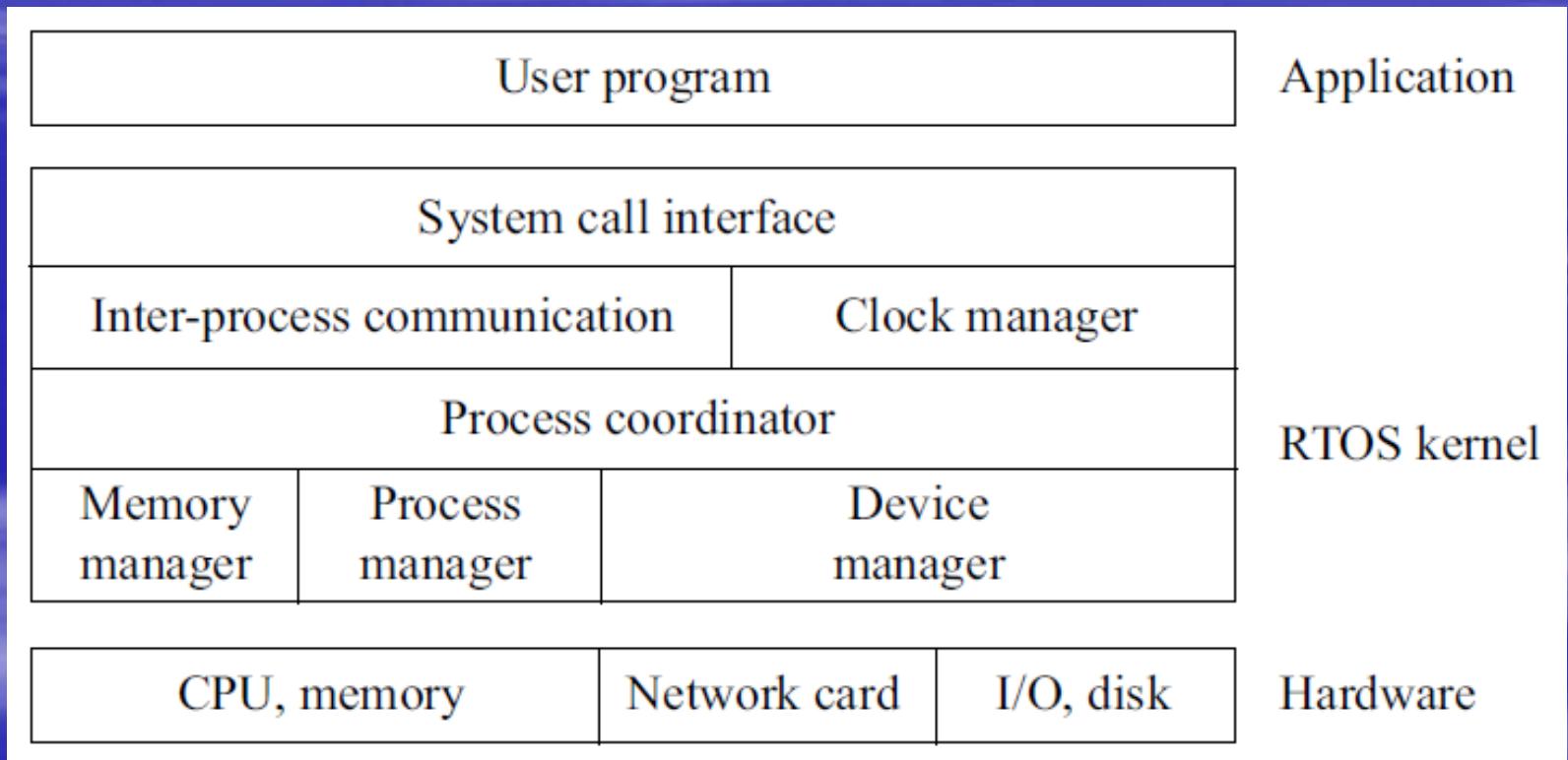
parallel (real-time) programming



# Real-time Programming

1. **Multi-tasking function** that enables execution of more than one task by a single processor or multiple processors.
2. **Synchronization function** that enables adjustment of the execution of tasks.
3. **Inter-communication function** that enables exchange of data between tasks.
4. **Preemption function** that enables stopping the task being executed in order to carry out a high-priority task and scheduling functions that enable management of the execution order of tasks.
5. **Timer function** that enables beginning a task at the right time.
6. **Interrupt function and exception handling function** that enables transmission of an asynchronous event to a processor.

# Structure of a Real-time OS



# Structure of a Real-time OS

- A real-time OS (RTOS) makes it easy to build real-time systems and its kernel provides the following functions:
  1. Multi-tasking
  2. Priority-based, pre-emptive scheduling
  3. Synchronization and inter-process communication
  4. Interrupt service

# Structure of a Real-time OS

- *Process Manager*
  - This module is the key function of a real-time OS and makes multi-tasking possible. It makes it possible to switch context depending on the priority and condition of a process or task

# Structure of a Real-time OS

- *Memory Manager*
  - in the case of real-time programming, a **memory-handling** method to specify directly the size and address of the allocated memory is used. This is for preventing the memory of each task from coming into conflict. Because particular tasks are always executed in real-time programs, this module provides a simple and fast memory management function by using a flat model instead of a paging model.

# Structure of a Real-time OS

- *Process coordinator*

- In a real-time OS, a process exists in one of a number of states, such as executing state, suspended state, and ready state. A process coordinator performs state transition according to the priority and schedule of a task. It plays the role of controlling the status of a task, such as creating a task, deleting, suspending until a specified condition is satisfied, and resuming if the specified condition is satisfied.

# Structure of a Real-time OS

- *Inter-process Communication (IPC)*
  - a process must be able to exchange data with other processes or the interrupt service routine (ISR) in order to be selectively synchronized with other processes. a **semaphore** mechanism is used for synchronization of processes and controlling critical regions and a **message queue** and **mail box** are used for exchanging data between processes

# Structure of a Real-time OS

- *Clock Manager*

- Basically, this module plays the role of real-time timer for multiprocessing. It is used for invoking the sleep function for delaying the execution of a task and the wake-up function for synchronizing the execution of tasks

# Structure of a Real-time OS

- *Device Manager*
  - This module plays the role of managing input/output devices (e.g., *RS232C*, *Ethernet*, *I/O*, *Printer*, and *Servo*, etc.) connected to the CNC system via the device driver.

# Process Management

## ■ *Process Creation and Termination*

1. **Creation:** Several processes can be created with a system call for process creation. The process that creates a process is called the ‘parent process’ and the generated process is called the ‘child process’. The parent process can share resources with the child processes.
2. **Termination:** After completing the last instruction, it is possible to request the OS to terminate a process or delete a particular process by a system call from another process. Typically, the system does not permit the existence of a child process after destruction of its parent process. Therefore, all child processes should be terminated when a parent process is terminated.

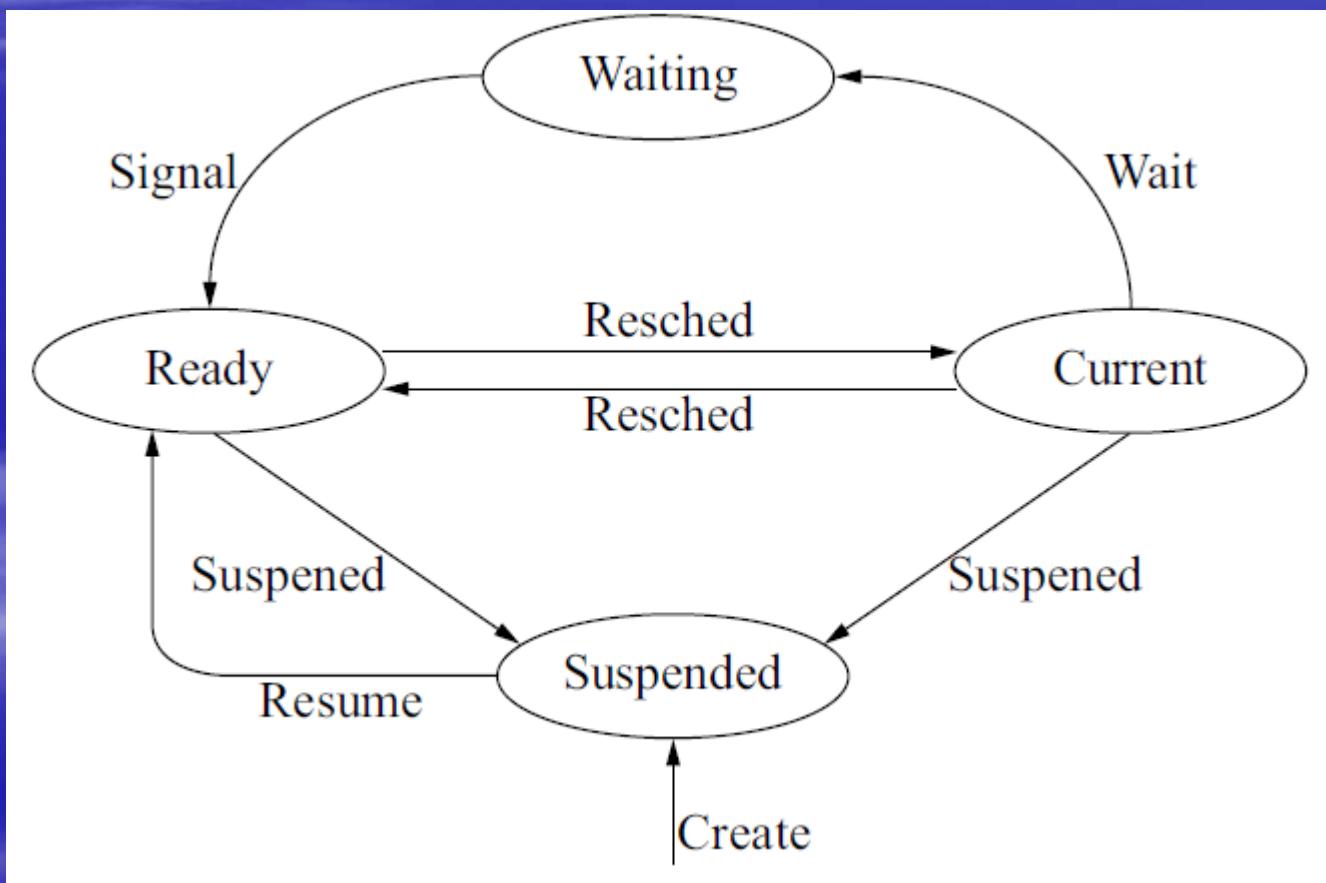
# Process Management

- *Process State Transition*

1. *Current state:* This means the case that a process occupies a processor and is being executed (running). In the case of a single processor, only one process can be in the current state.
2. *Ready state:* This means the case that a process does not currently occupy a processor but can be executed at any time.
3. *Receiving state:* This means the case that a process is awaiting a message or a mail from another process.
4. *Sleeping state:* This means the case that a process is ‘sleeping’ during a specified time.
5. *Suspended state:* This means the case that a process has stopped execution. When a process is created it is always in this state.
6. *Waiting state:* This means the case that a process is waiting for an external event or semaphore.

# Process Management

- Diagram of Process State Transitions



# Process management program example

```
/* RTOS task management          */
/* - Create/Suspend/Resume/Delete a task */

#include<conf.h>
#include<kernel.h>
#include<errno.h>

void main()
{
    int    err, tid;
    void  task1();
    struct timespec time;

    printf("[1. Create a task by the name of task1.]\n");
    tid = rt_create(task1, 1, INITPRIO+1, &err);
    if (err != RET_OK)
        printf("**** Error: can't create a task.\n");
    time.seconds = 5;
    rt_delay(time, &err);

    printf("\n[2. Suspend task1.]");
    rt_suspend(tid, 0, &err);
    time.seconds = 2;
    rt_delay(time, &err);

    printf("\n[3. Resume task1.]");
    rt_resume(tid, 0, &err);
    time.seconds = 3;
    rt_delay(time, &err);

    printf("\n[4. Delete task1.]");
    rt_delete(tid, 0, &err);

    printf("\n[----- End of test -----]\n");
}
```

# Process Management

## ■ *Process Scheduling*

- The majority of real-time operating systems use a scheduling algorithm for managing several real-time tasks that require real-time execution in a **preemption multitasking** environment

# Process Management

- *Process Scheduling*

- First-Come, First-Served Scheduling

- As First-Come, First-Served scheduling is the simplest scheduling algorithm, it allocates a resource according to the queue of requests. When the task is inserted into a ready queue the control block of the task is connected to the end of the queue. When the current task ends, the resource is allocated to the task at the head of the queue and the allocated task is deleted from the queue. Consequently, system resources are allocated by the sequence of the queue.

# Process Management

- *Process Scheduling*

- Time Slice

- In the time-slice scheduling algorithm, time is split into intervals of the **same length** and a task is allowed to operate during a certain amount of a time slice
    - The execution sequence of tasks is typically determined by a **round-robin** method
    - This scheduling algorithm **is generally used for soft real-time systems** and is appropriate for background scheduling of regular tasks having long response times.

# Process Management

- Priority

- As a more complicated scheduling method, a method based on task priority can be used. Priority is allocated to each task and the scheduler allocates the processor to the highest priority task. If tasks have the same priority, they are executed by a First-Come First-Served scheduling method. The priority specified by a programmer can be changed while the task is being carried out

# Process Management

- Priority
  - The **fixed-priority scheduling** minimizes the execution burden of a real-time system and the **Rate Monotonic** (RM) algorithm is the most typical fixed-priority scheduling algorithm. In this algorithm, the priority is static and tasks with **shorter periods are given higher priorities**. The task with the highest priority that can be run immediately pre-empts all other tasks. In the RM algorithm, each task has its own static priority and the instance of each task is not given a new priority. Because the static-priority scheduling consumes less computing power and implementation is easier compared to dynamic-priority scheduling, it is widely used in real-time systems that require a deterministic guarantee with regard to response time

# Process Management

## ■ Priority

- In **static-priority** scheduling, only the task with highest priority may be executed. To overcome this problem, the priority of the task being executed is linearly decreased by the scheduler when its current time slice is gone. Therefore, the executing task comes to have lower priority than a waiting task. By using this method, it is certain that all tasks come to be executed. In consequence, dynamic priority assignment is done at the end of each time slice.

# Process Management

## ■ Priority

- As another **dynamic-priority assignment** method, the aging method is used. In the aging method, the priority of a task becomes higher after each time slice. This method prevents the task with low priority from waiting endlessly and allows the task with lowest priority to be executed. In conclusion, because of the different initial priority, a task with high priority is executed more frequently than a task with low priority. Therefore, a task which has to be called frequently or promptly has high priority and a task in which long response time is permitted has low priority.

# Process Management

## ■ Fixed Sample Time

- In fixed-sample-time scheduling, **time is not divided into fixed slices** but is sliced depending on the property of a task. This means that, in the case that the same time slice is assigned to all the tasks, a task that has not completed in the fixed time may be terminated without any result. To solve this problem, at the stage of defining a task, an **adequate time period** is specified for each task and the task is scheduled by using the individual software timer corresponding to the sample time

# Process Management

- Event-driven
  - The majority of scheduling methods assume periodic task service. However, event driven scheduling is used for irregular tasks. This method is appropriate in the case when some task is fired by an event or data from a sensor.

# Programming example of task scheduling

```
/* RTOS task scheduling          */
/* - Lock/unlock scheduling    */
/* - Change priority of a task */

#include<conf.h>
#include<kernel.h>
#include<errno.h>

void main()
{
    int    err, tid1, tid2;
    int    flag;
    void  task1(), task2();
    struct timespec time;

    printf("[1. Create two tasks(task1, task2) which have the same priority.]\\n");
    tid1 = rt_create(task1, 1, INITPRIO+2, &err);
    if (err != RET_OK)
        printf("**** Error: can't create a task\\n");
    tid2 = rt_create(task2, 2, INITPRIO+2, &err);
    if (err != RET_OK)
        printf("**** Error: can't create a task\\n");

    time.seconds = 5;
    rt_delay(&time, &err);

    printf("\\n[2. Lock scheduling.]\\n");
    flag = rt_lock();
    time.seconds = 3;
    rt_delay(&time, &err);

    printf("\\n[3. Unlock scheduling.]\\n");
    rt_unlock(flag);
    time.seconds = 2;
    rt_delay(&time, &err);

    printf("\\n[4. Let task1 have higher priority than task2.]\\n");
    rt_priority(tid1, INITPRIO+1, &err);

    printf("\\n[5. Delete task1.]\\n");
    rt_delete(tid1, 0, &err);

    printf("\\n[6. Delete task2.]\\n");
    rt_delete(tid2, 0, &err);

    printf("\\n[----- End of test -----]\\n");
}
```

# Process Synchronization

- In a system based on multi-processing OS, all tasks can possibly be carried out simultaneously. Therefore, in order to guarantee the right execution sequence of tasks it is necessary for the OS to provide a synchronization mechanism between tasks

# Process Synchronization

- **Semaphores**

- The behavior of the semaphore can be summarized as follows.
  - The P action decreases the semaphore variable by one and is performed by calling WAIT(semaphore variable). By a P action, whether a particular process can access a shared resource is checked. If the semaphore variable is greater than one, the process that connects to the semaphore variable can access the resource and the P action decreases the semaphore variable by one before access.
  - The V action increases the semaphore variable by one and is performed by calling SIGNAL(semaphore variable). It gives the access right to the next process. After increasing the semaphore value by one, a process connecting to the semaphore can access the resource during its scheduled execution.

```

/* Coordinated by scheduler for displaying 'A', 'B', 'C' */

#include<conf.h>
#include<kernell.h>

void main()
{
    int proc1(), proc2(), proc3();

    printf("\n Display 'A', 'B', 'C'\n");
    printf(" Output...\n\n");
    rt_resume(rt_create(proc1, INITSTK, INITPRIO, "proc1", 0, 0));
    rt_resume(rt_create(proc2, INITSTK, INITPRIO, "proc2", 0, 0));
    rt_resume(rt_create(proc3, INITSTK, INITPRIO, "proc3", 0, 0));
}

proc1()
{
    int i;
    for (i = 0; i < 1000; i++) {
        printf("A");
    }
}

proc2()
{
    int i;
    for (i = 0; i < 1000; i++) {
        putc(CONSOLE, 'B');
    }
}

proc3()
{
    int i;
    for (i = 0; i < 1000; i++) {
        putc(CONSOLE, 'C');
    }
}

```

### Display 'A' 'B' 'C'

```
Display 'A', 'B', 'C'  
Output...  
AAAAAAAAAAAAAAA.AAAA.AAAA.AAAA.AAAABBBBBBBBBBBBBBBBBB  
BBBBBBBBBBBBBCCCCCCCCCCCCCCCCCCCCCCCCCCCCAA.AA.AAA.AA  
AAAAAAA AAA BBBB BBBB BBBB BBBB BBBB BBBB CCCCCCCCCCCCCC
```

## Programming example without the synchronization mechanism

# Programming example of task synchronization by using semaphores

# Process Synchronization

## ▪ *Events and Signals*

- The event method uses an event flag and is an appropriate mechanism for realizing synchronization when multiple events occur. The event flag that corresponds to a particular event is located in the event memory. So, if a particular event occurs the corresponding event flag is turned on. As soon as the event flag is turned on, the task that is waiting for that event moves into the ready state. The event flag plays the role of passing control and causes the OS to activate the appropriate event handler.
- The signal method is slightly different from the event method and works like an interrupt. If a particular signal is fired, the task currently running is stopped and the task corresponding to the signal is called. This is very similar to the way that the interrupt service routine (ISR) is activated by an interrupt.

# Resources

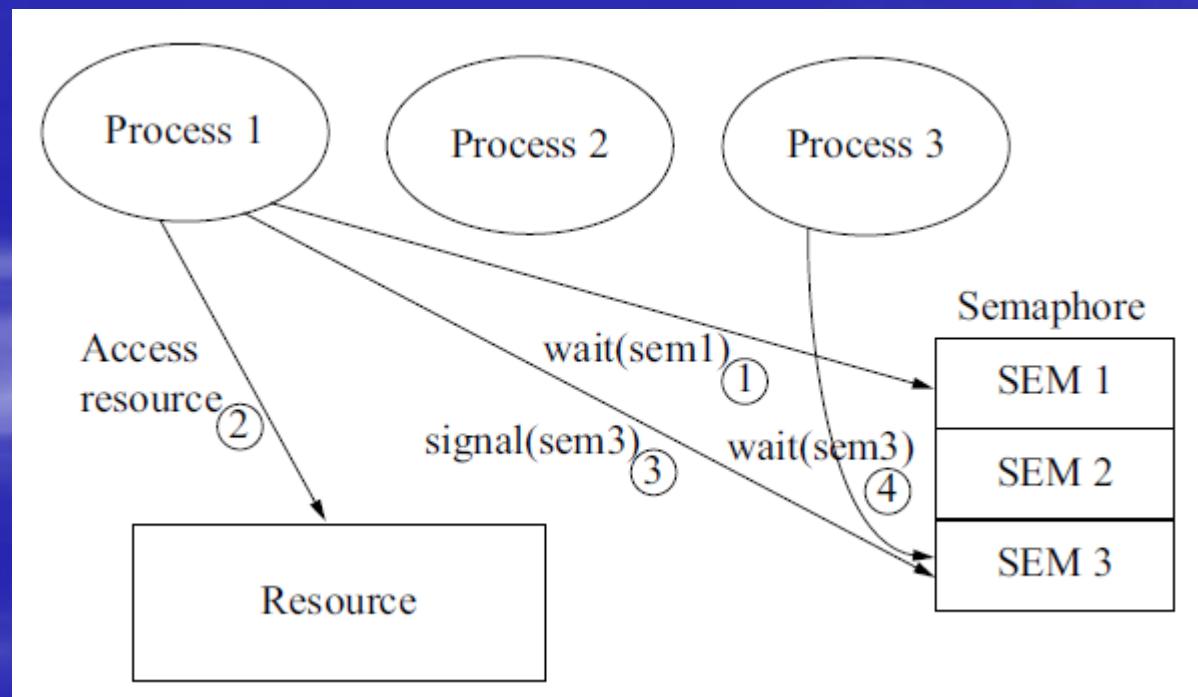
## ■ *System Resources*

- A resource means not only hardware such as a printer or a disk but also things that the task being executed accesses such as variables in main memory. In multiprogramming, competition between tasks for the use of resources sometimes occurs. If this competition is not effectively managed, a system may work abnormally or it may be terminated. Therefore, concern about resource protection is one of the key issues in multi-programming theory.
- “critical section”

# Resources

## ■ Mutual Exclusion

- Allowing only one task to have access to a common variable among the tasks that want to access it is called the “mutual exclusion mechanism”.



# Resources

## ■ *Deadlock*

– the necessary and sufficient condition of the deadlock occurrence can be summarized as follows:

- Mutual exclusion
- Hold and Wait
- Non-Pre-empted allocation
- Circular wait

# Inter-process Communication

- As methods to realize the inter-process communication, **shared memory** and **message passing** can be used. These complement each other and they can be simultaneously used in one OS.

# Inter-process Communication

- *Shared Memory*

- For inter-process communication via shared memory, global variables where processes can read and write can be considered. However, because usage only of global variables may cause data clashes when more than one process accesses the global variables simultaneously, it is essential to use **critical sections** with this method

# Inter-process Communication

## ■ Message System

- **Direct Communication:** The process that wants to send or receive a message specifies the name of the receiver or sender. For this, it is necessary to know the name of the corresponding process.
  - SEND (P, message): send message to P.
  - RECEIVE(Q, message): receive message from Q.
- **Indirect Communication:** the message is transmitted via a mail box. The mail box has a unique identification number and when two processes share the same mail box communication is possible.
  - SEND (A, message): Send message to mail box A.
  - RECEIVE(A, message): Receive message from mail box A.

```
/* Display 'A' and 'B' one by one utilizing message passing method */

#include<conf.h>
#include<kemel.h>

void main()
{
    int proc1(), proc2();
    printf("\n Display A and B one by one by utilizing message passing method \n");
    printf(" Output...\n\n");
    pid1 = create(proc1, INITSTK, INITPRIO, "proc1", 0, 0); /* create process 1 */
    resume(pid1); /* make process 1 READY */
    pid2 = create(proc2, INITSTK, INITPRIO, "proc2", 0, 0);
    resume(pid2);
}

proc1()
{
    int i;
    int msg1, msg2;

    for (i = 0; i < 10; i++) {
        /* receive message, if message is not received, process1 turns to wait state */
        msg1 = receive();
        if (msg1 != 0) printf("A");
        msg2 = 1; /* assign nonzero value for sending to process 2 */
        send(pid2, msg2);
    }
}

proc2()
{
    int i;
    int msg2 = 0, msg1 = 5;
    for (i = 0; i < 10; i++) {
        msg2 = receive(); /* receive message from process1 */
        if (msg2 != 0) printf("B");
        send(pid1, msg1);
    }
}
```

Programming example utilizing a message for inter-task communication

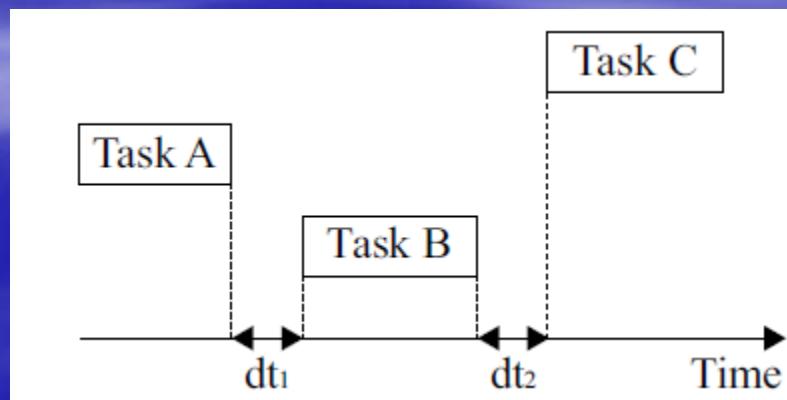
# Key Performance Indices

- A real-time OS supports pre-emption multi-tasking. The multi-tasking ability enables effective resource management as well as parallel execution of processes or tasks. For efficient multi-tasking, it is essential to increase the response characteristics of the OS by reducing the **context switching** time. Furthermore, a real-time OS can predict the required time for running tasks in order to realize the real-time scheduling and synchronization mechanism. In addition, it is necessary to know the characteristics of the interrupter, such as **interrupt latency**, which is the time spent to **resume a task after an interrupt has been fired**, the maximum elapsed time of the **system call**, etc.

# Key Performance Indices

## ■ *Task Switching Time*

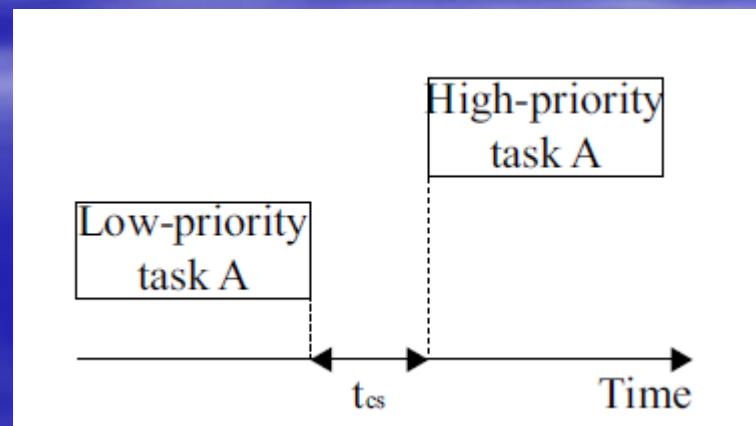
- Task switching time means the average time spent to switch between two tasks with the **same priority**. Task switching is done when real-time software uses the time-sharing algorithm to carry out tasks with the same priority. Task switching time is used for **storing and restoring context**. And it depends on the efficiency of control data structure, processor architecture, and instruction set



# Key Performance Indices

## ■ *Context Switching Time*

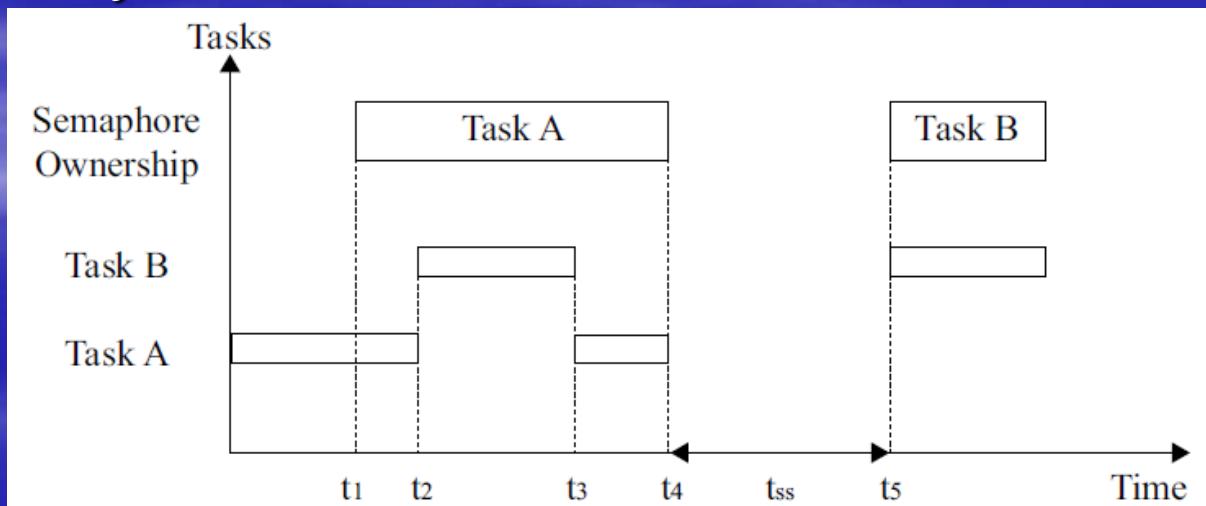
- The context switching time denotes the time spent to start task B when task A with low priority is being executed. For the context switching time, the context of the pre-empted task is stored, the context of the new task is loaded, and the new task is scheduled. Note that the task switching time denotes the time spent to switch tasks with the same priority, and that the context switching time is different from the task switching time.



# Key Performance Indices

## ■ *Semaphore Shuffling Time*

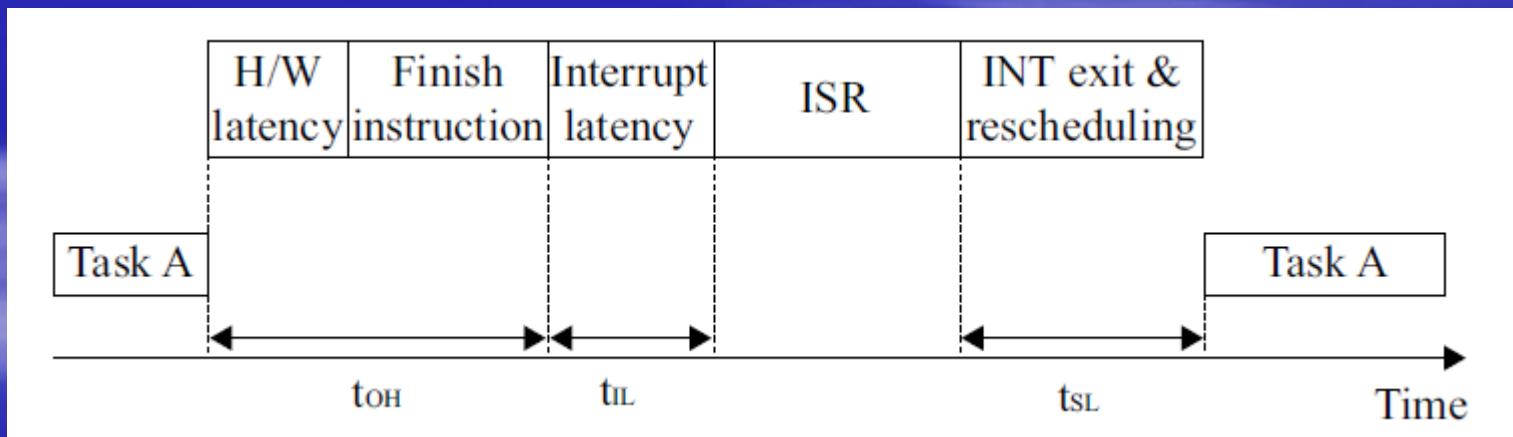
- The semaphore shuffling time denotes the time delay from when some task frees the semaphore to when the task waiting for the semaphore is activated. Because the semaphore shuffling time itself represents a computing burden related with the mutual exclusion, the semaphore shuffling time is one of the key performance indices of real-time systems.



# Key Performance Indices

## ■ *Task Dispatch Latency Time*

- The task dispatch latency time denotes the time spent to start a task from the interrupt service request. The task dispatch latency time is highly related to the **interrupt latency** time and the **context switching** time



# Key Performance Indices

the worst case example of the task dispatch latency time executing in LynxOS (one of the widely used RTOS).

Interrupt dispatch time	12
Interrupt routine	10
Other interrupt	60
Pre-emption disabled	75
Scheduling	28
Context switch	13
Return from system call	12

# Key Performance Indices

the comparison of the performance index of three kinds of a real-time OS; Hyperkernel 4.3 (Imagination), INTIME 1.2 (Radisys), and RTX4.1 (VentureCom). They were tested on a PC with a Pentium 200MHz CPU.

Performance Index		System A	System B	System C
Task Switching Latency	Avg.	5.47	4.68	2.64
	Max.	23.13	10.68	5.73
ISR Latency	Avg.	11.26	8.78	7.66
	Max.	19.23	14.52	25.68
Scheduling Latency	Avg.	25.95	4.73	6.36
	Max.	39.0	10.14	32.25

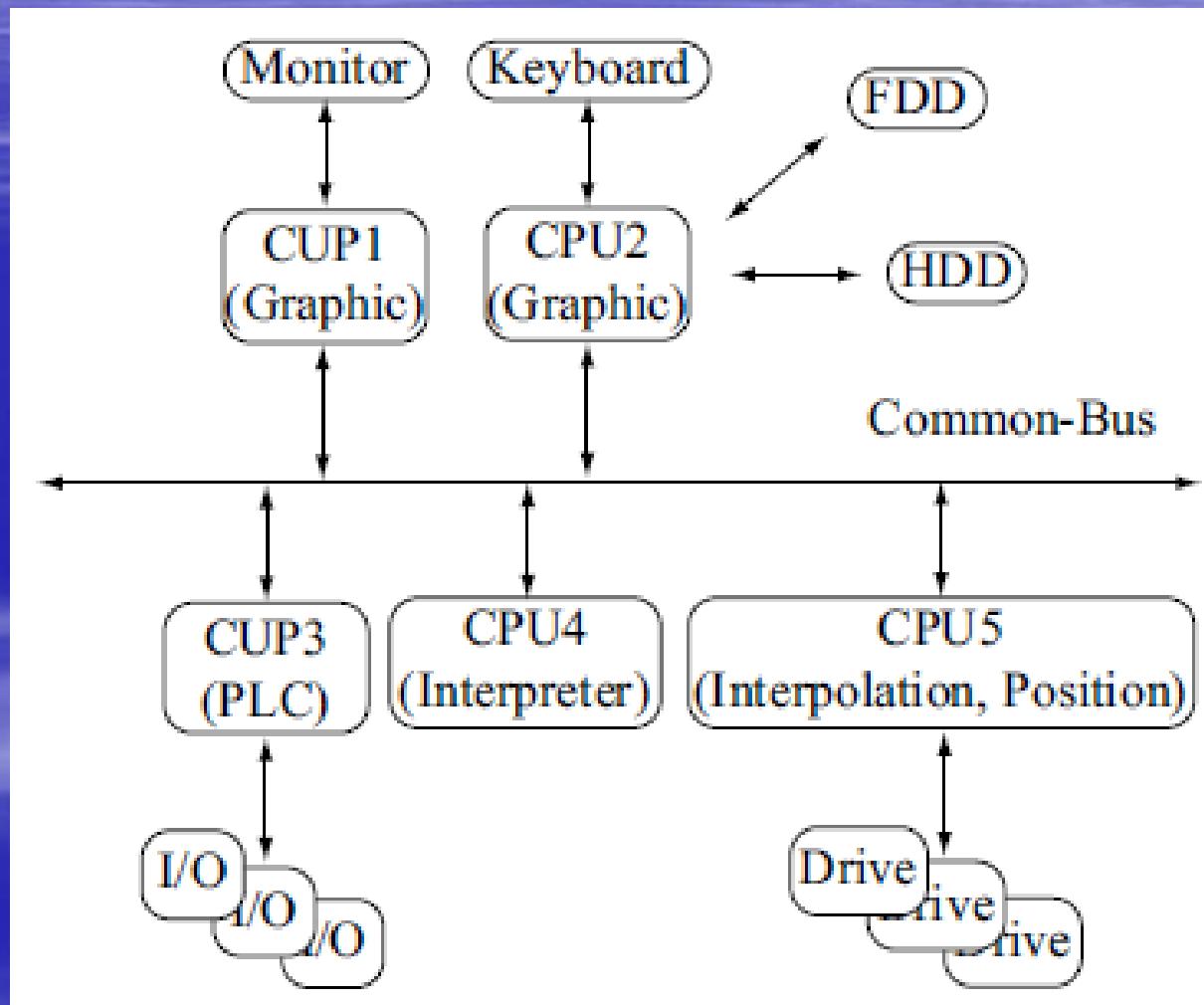
# Hardware and Operating Systems

- *Architecture of Multi-processing Hardware*
  - As the hardware architecture for multi-processing systems, a bus structure has been widely used. Bus structures are classified into two types; one is the **common bus type**, where one bus is shared by multiple processors, and the other is the **standard bus type**, where a computer unit with a heterogeneous local bus is connected to a standard bus (e.g., *Multi-bus* and *VME bus*).

# Hardware and Operating Systems

- *Architecture of Multi-processing Hardware*
  - For the sake of explanation, let us suppose that each bus type is applied to the CNC system. The **common bus type** is a structure where multiple processors, carrying out tasks such as PLC, MMI, interpreter, and interpolator, **are connected via one bus**, using the common memory and having their own individual input/output interfaces. It can be called a “closed” architecture, where system scaling is very difficult.

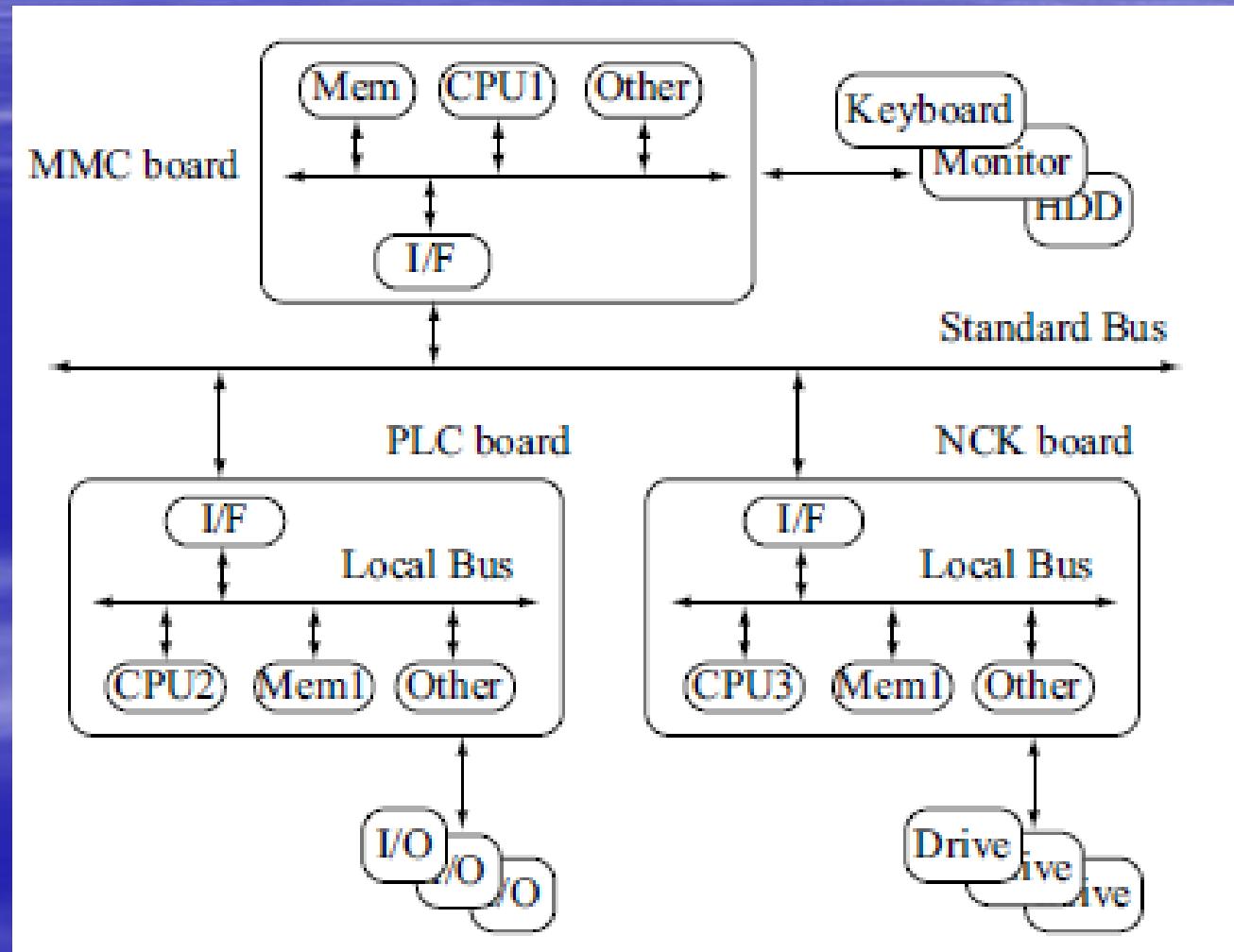
# Common bus type



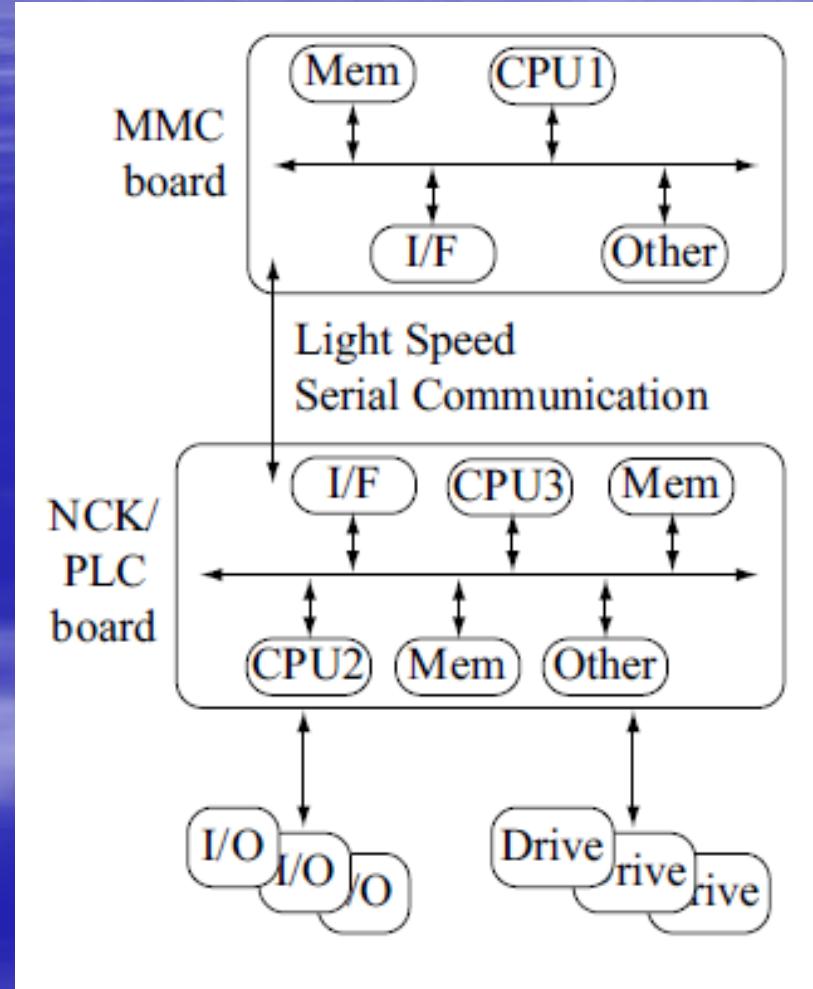
# **Hardware and Operating Systems**

- *Architecture of Multi-processing Hardware*
  - The architecture implements several processor units executing NCK, PLC, and MMC and all units are connected via **a standard bus**. Accordingly, expansion of such a system is easily possible only by adding process units for particular tasks to the standard bus. In this architecture, each processor unit communicates with the other units via a common memory module.

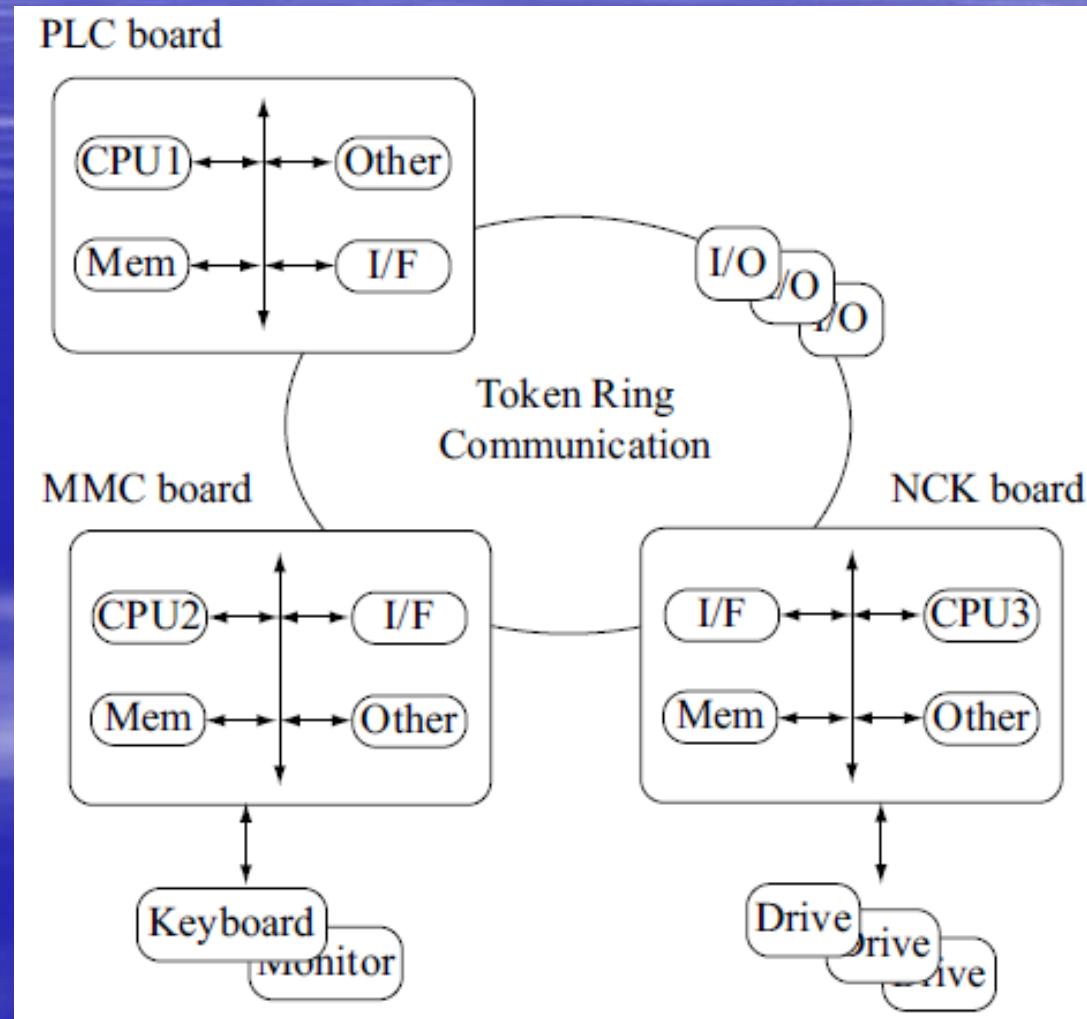
# Standard bus type



# Dedicated Communication



# Distributed Communication



# Hardware and Operating Systems

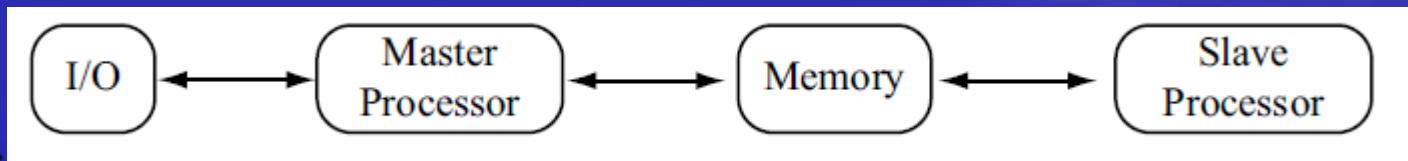
- *Operating System Configuration*

- Not only integration of hardware for multi-processing but also configuration of operating systems must be considered. The configuration method for allocating and managing resources, protecting resources, preventing deadlocks, terminating abnormal execution, balancing input/output load, and balancing process load should be defined together with the configuration method of the hardware. As configuration methods for multi-processing operating systems, there are the **master/slave method**, **separate executive method**, and **symmetrical method**.

# Hardware and Operating Systems

- *Operating System Configuration*

- Master-Slave

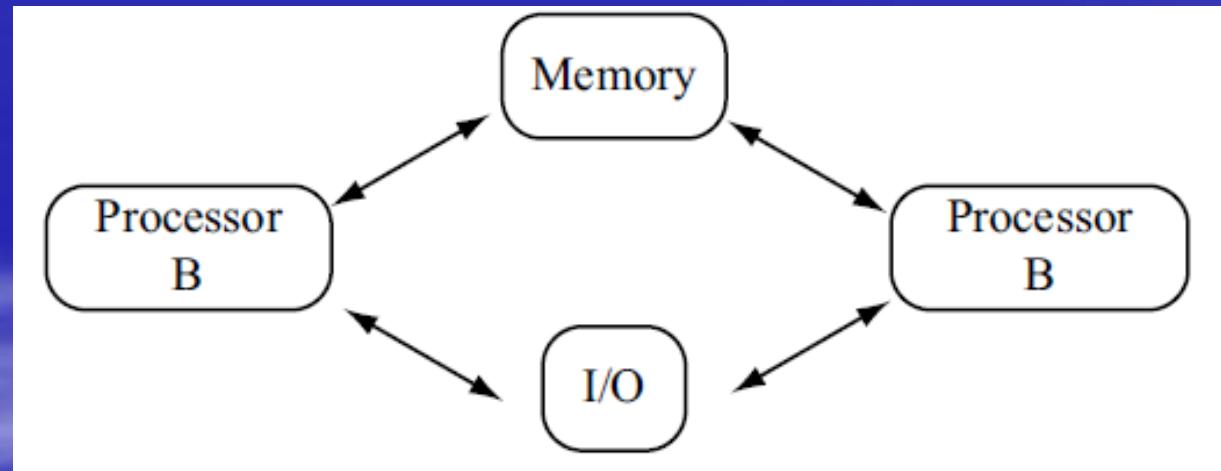


- Separate executive



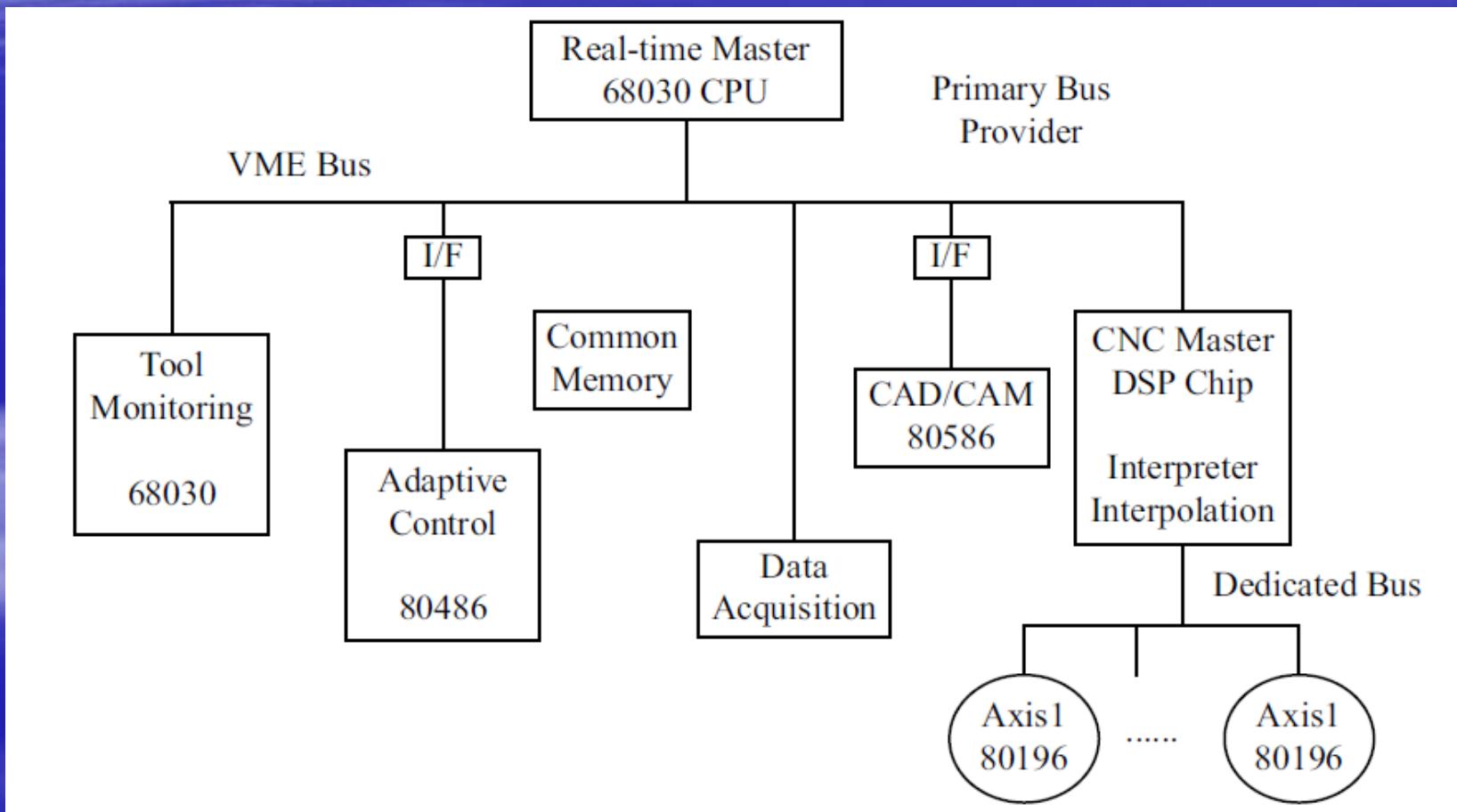
# Hardware and Operating Systems

- *Operating System Configuration*
  - symmetrical



# Hardware and Operating Systems

## ■ CNC System Architecture



# Hardware and Operating Systems

- Commercial CNC system having a **bus system**



FANUC 0 series



Siemens 840C

# Hardware and Operating Systems

- Dedicated & Distributed Communication



Siemens 840D

FANUC 150i

# Summary

- A CNC system is a real-time system where NCK, MMI, and PLC should be executed within a specified time. In order to design a CNC system that guarantees hard real-time property, a harmonized relationship between the functionality of **real-time OS** and the **architecture of hardware and software** has to be considered.

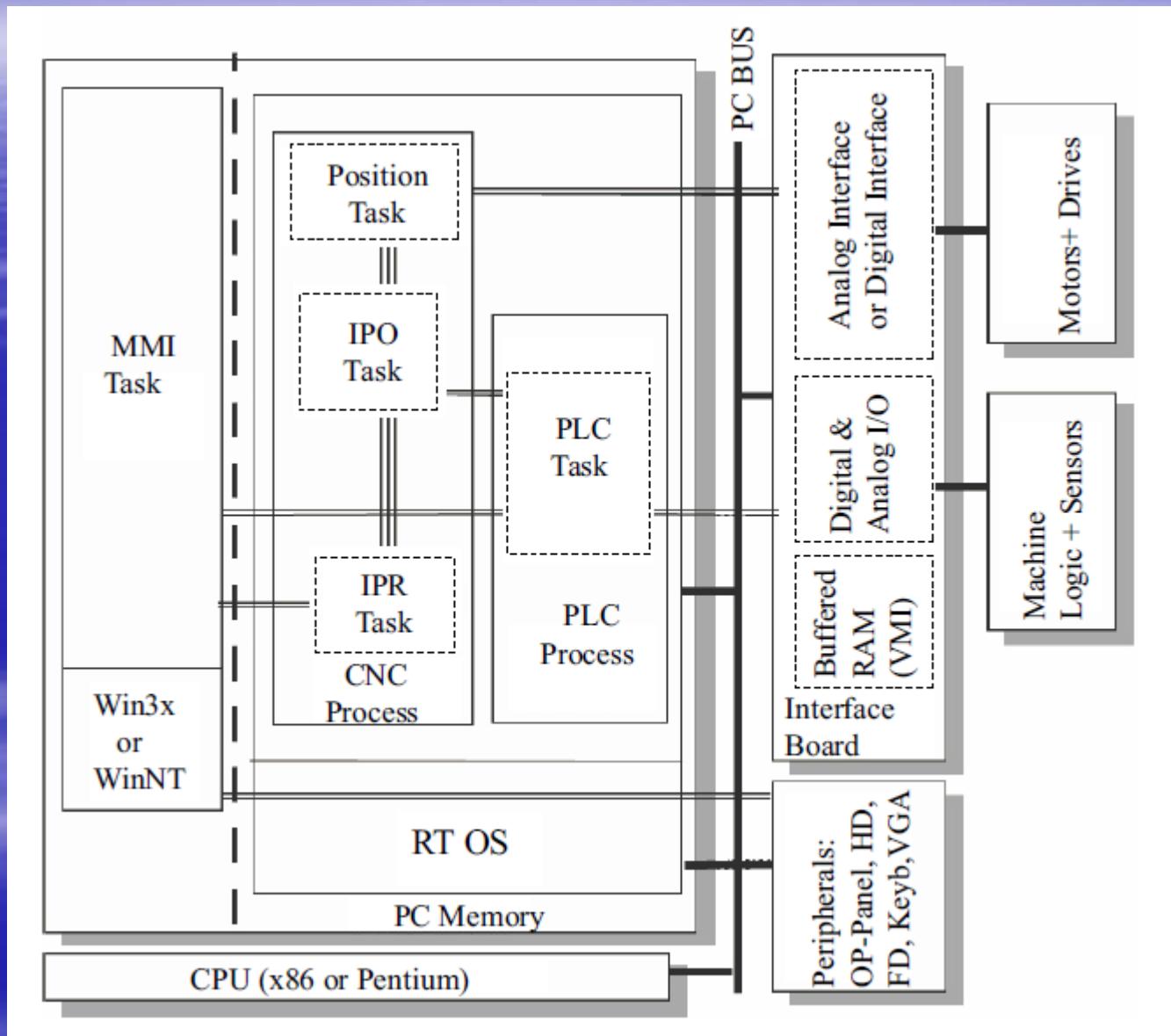
# Theory and Design of CNC Systems

*Chapter 10*  
**Design of PC-NC and Open CNC**

# Introduction

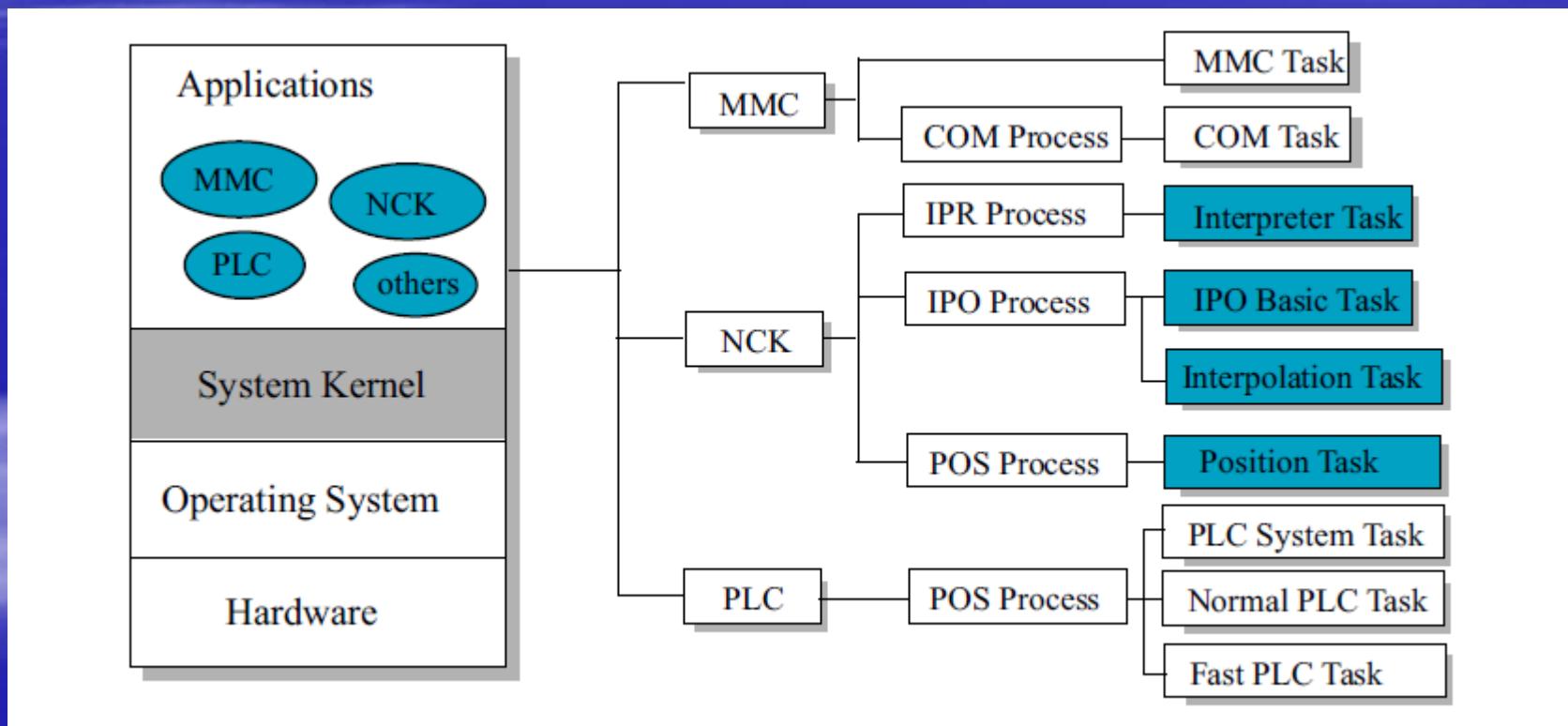
1. Embedded motion controller, which carries out the NCK/PLC function with its own processor, is attached to the extended slot of PC. The MMI is operated on the PC.
2. Two PCs are used and are connected via high-speed communication. One PC is used for MMI and the other is used for NCK/PLC.
3. One PC with single CPU executes MMI, NCK and PLC in a multi-threading environment with real-time OS.

## Example of Soft-NC architecture using both RTOS and Windows



# Design of Software Architecture

Software model and application tasks



# Design of Software Architecture

If we classify the variety of tasks in terms of function in order to make the model, they can be divided into two types; one is the non-cyclic task and the other is the cyclic task. A non-cyclic task means a task in which the real-time property is not required and the MMC, interpreter and communication function are typical examples of the non-cyclic task. A cyclic task is a task for which the real-time property is required and the interpolator and position control tasks are typical examples of cyclic tasks.

# Design of Software Architecture

- Non-cyclic Task
- The **interpreter** reads ASCII blocks from a part program and interprets them. Then, for the next task, it saves the interpreted data in internal memory. The interpreter steps performed are as follows:
  1. Read ASCII block and check the schema and grammar.
  2. Transform the G-code block read into an internal data structure.
  3. Save modal or one-shot data in an internal data structure.
  4. Perform operations related with variables in part program.
  5. Perform control flow operations such as jumps and subroutine calls in the part program.
  6. Interpret macros in the part program.
  7. Synchronize with the interpolator.

# Design of Software Architecture

- Non-cyclic Task
- The **intermediate handler** generates data concerning tools, spindle, and coordination data based on the information in the data from the interpreter and stores it in an internal buffer. The details of the intermediate handler are as follows:
  1. Set spindle function.
  2. Transform program coordination system into the local coordinate system specified in the CNC system.
  3. Do tool compensation.
  4. Check speed limits and prohibited machining areas.
  5. Get the data ready for interpolation.
  6. Perform look-ahead functions

# Design of Software Architecture

- Non-cyclic Task
- The **external communication** task is to provide an interface between NCK and external components and provides the following functions:
  1. Send a part program to NCK.
  2. Read and write the parameters and user-specified variables such as tool compensation to and from NCK.

# Design of Software Architecture

- **Cyclic Task – Low Priority**
- The **interpolator** belongs to the category of cyclic tasks with low priority. The interpolator reads the interpreted data from the internal buffer and then interpolates the movement of all axes. The interpolated data is then sent to the position control loop. The interpolator is executed as follows:
  1. Read the data from the internal buffer.
  2. Read the actual position from the position controller.
  3. Interpolate spindle rotation in position control mode
  4. Interpolate axis movement along the path.
  5. Transform the coordinate system and check prohibited machining areas.
  6. Send interpolated data to the position controller and check software limits of axes.

# Design of Software Architecture

- **Cyclic Task – High Priority**
- The **position control** task belongs to the category of cyclic tasks with high priority. In the position control task, interpolated data is transformed to motor rotation speed and, if necessary, the torque of the drive is computed. The limits of the control loop and drive are checked.
  1. Actual position is fed back from a drive.
  2. Perform position control algorithm and compute commands for drives.
  3. Communicate with the interpolator.
  4. Prepare the instruction at the next position control sampling time.

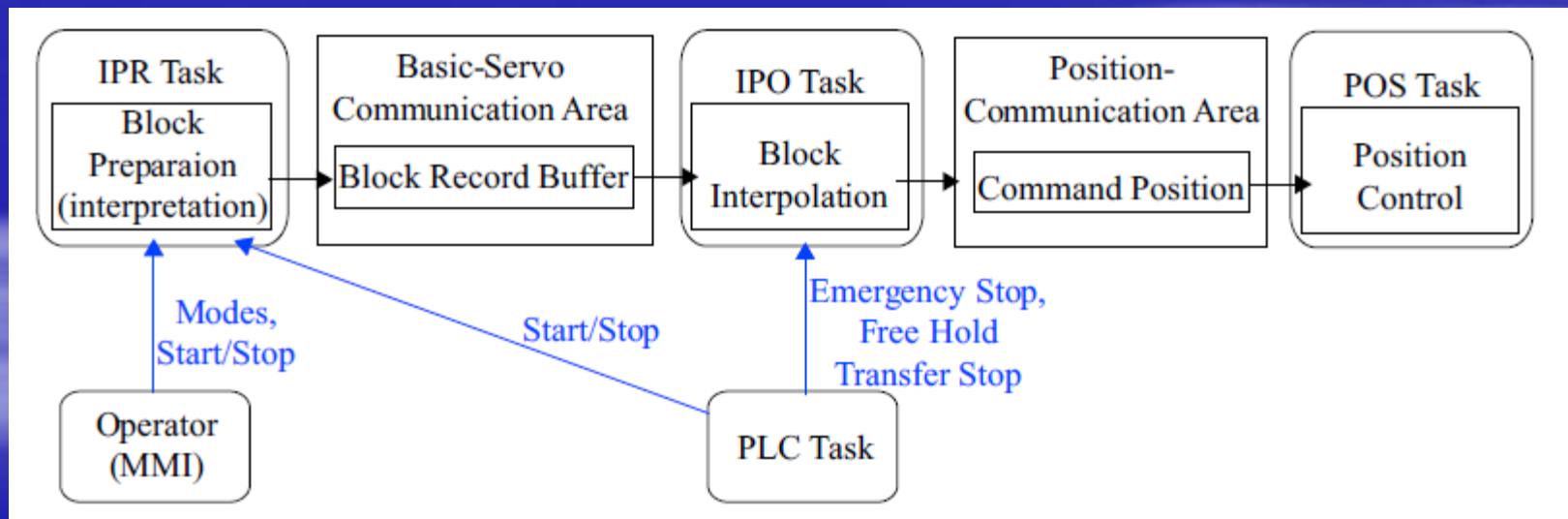
# Design of Software Architecture

- In conclusion, the key to the design of a Soft-NC is:
  1. to realize the system kernel programming to make the tasks **execute perfectly** in the kernel, and
  2. to develop each task in software form as **modeled**, under the determined hardware architecture and operating system.

# Design of Soft-NC System

## ■ Design of Task Module

- The task modules of the CNC system can be designed in unlimited combinations according to the specification of the CNC and the designer's concept.



Design example of the task modules

## Functions of each task module

Module	Task	Main function
NCK	Interpreter Task (IPR)	<p>Control the operating mode of NC.</p> <p>Send position data and NC status to MMI.</p> <p>Communicate with the Interpolation Task.</p> <p>Communicate with the PLC task.</p> <p>Interpret NC blocks.</p> <p>Store the interpreted data in the block record buffer.</p> <p>Compensate for tool offset and tool length.</p> <p>Check software limits.</p>
	Interpolation Task (IPO)	<p>Read the block record buffer.</p> <p>Communicate with the Position Task.</p> <p>Read/write CNC-PLC interface.</p> <p>Perform Linear/Circular/Spline/Polar interpolation.</p> <p>Perform real-time transformation.</p> <p>Compensate for backlash and pitch error.</p> <p>Carry out spindle processing.</p>
	Position Task (POS)	<p>Read the encoder.</p> <p>Calculate position error.</p> <p>Perform the position control algorithm.</p> <p>Monitor servo faults.</p> <p>Output velocity instructions to each drive.</p>

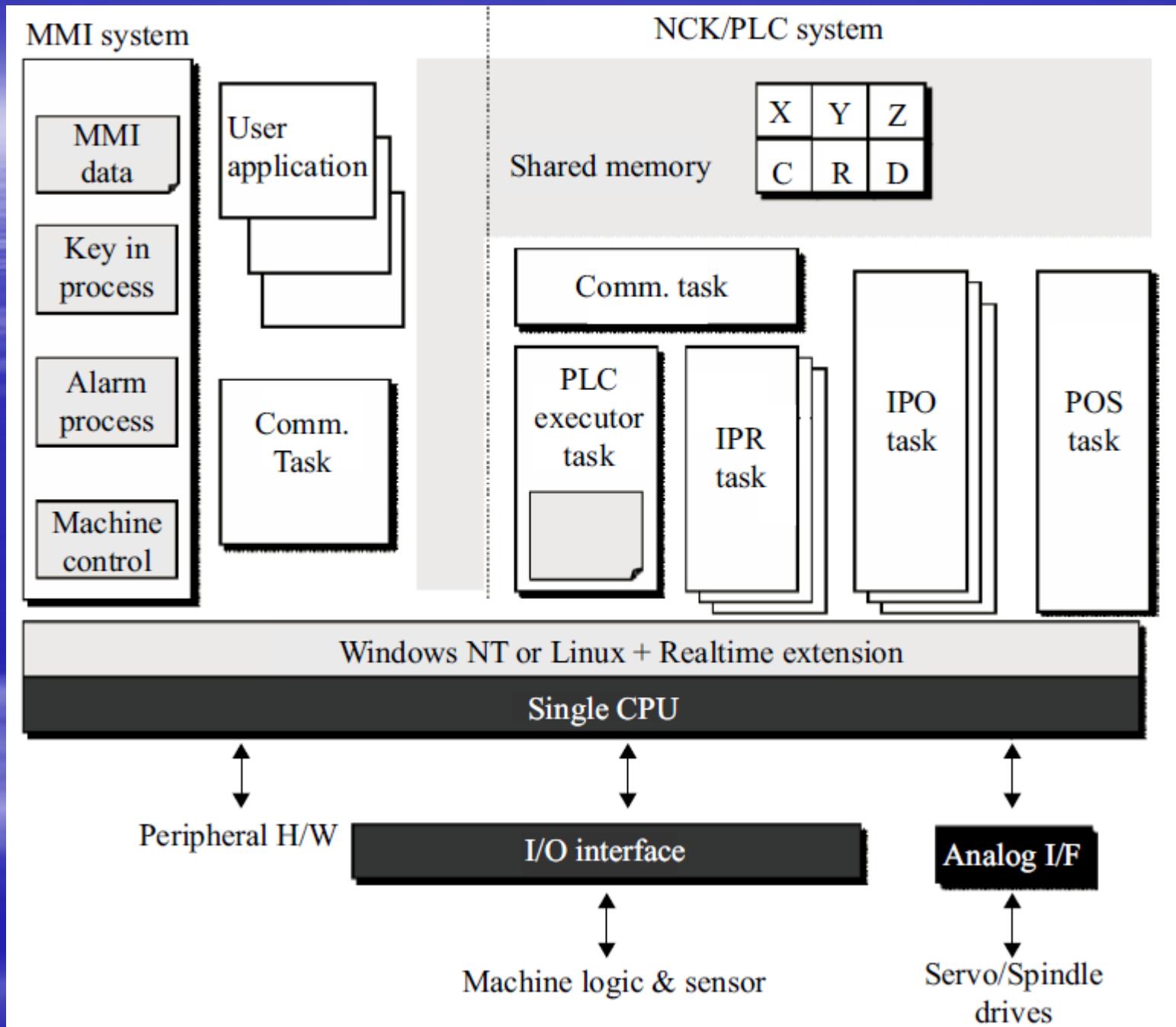
## Functions of each task module

Module	Task	Main function
PLC	Fast PLC Task (FPLC)	Handle interrupt signals. Handle inspection signals. Handle Fast I/O signals.
	Normal PLC Task (NPLC)	Handle Normal I/O signals. Handle M, S, and T codes.
Module	Task	Main function
MMI	Machine	Display auto-mode, MDI mode, and tool path.
	Program	Execute G-code editor, folder manager, and conversational programming system.
	Parameter	Manage parameters related to system, programming, and tool parameters.
	Tools	Manage tool offset, tool life, and tool shape.
	Utility	Manage DNC, PLC monitoring, alarm, and data communication
	Kernel	Manage screen display, key input, file management, application module handling, system boot up and external communication.

# Design of Soft-NC System

- *Design of the System Kernel*
  - the system kernel should be designed based on a **pre-emptive multi-tasking OS** in which each task should be regularly executed within a specified time. In particular, the NCK/PLC system that must guarantee hard real-time property is executed regularly and the MMI system with soft real-time property is executed non-periodically.

## Modular structure of Soft-NC



# Design of Soft-NC System

- *Design of the System Kernel*

- Therefore, a variety of tasks are executed sequentially every specified sampling time. The position control task having the highest priority is executed every shortest sampling time, the interpolation task having the next highest priority is performed every multiple times of the sampling time of the position control task. The interpreter task having low priority is regularly performed during the spare time of the processor. Finally, the MMI system for user interface process, having the lowest priority, is designed as a non-cyclic process. Therefore, the MMI task is executed using the remaining computing power of the processor after all other tasks are finished. The scheduling method, above, is somewhat simple and is called “**monotonic scheduling**”. It is suitable for a CNC system because the functions, or tasks, of a CNC system can easily be modularized and the behavior of each task working within the module depends on the sequential result of preceding tasks.

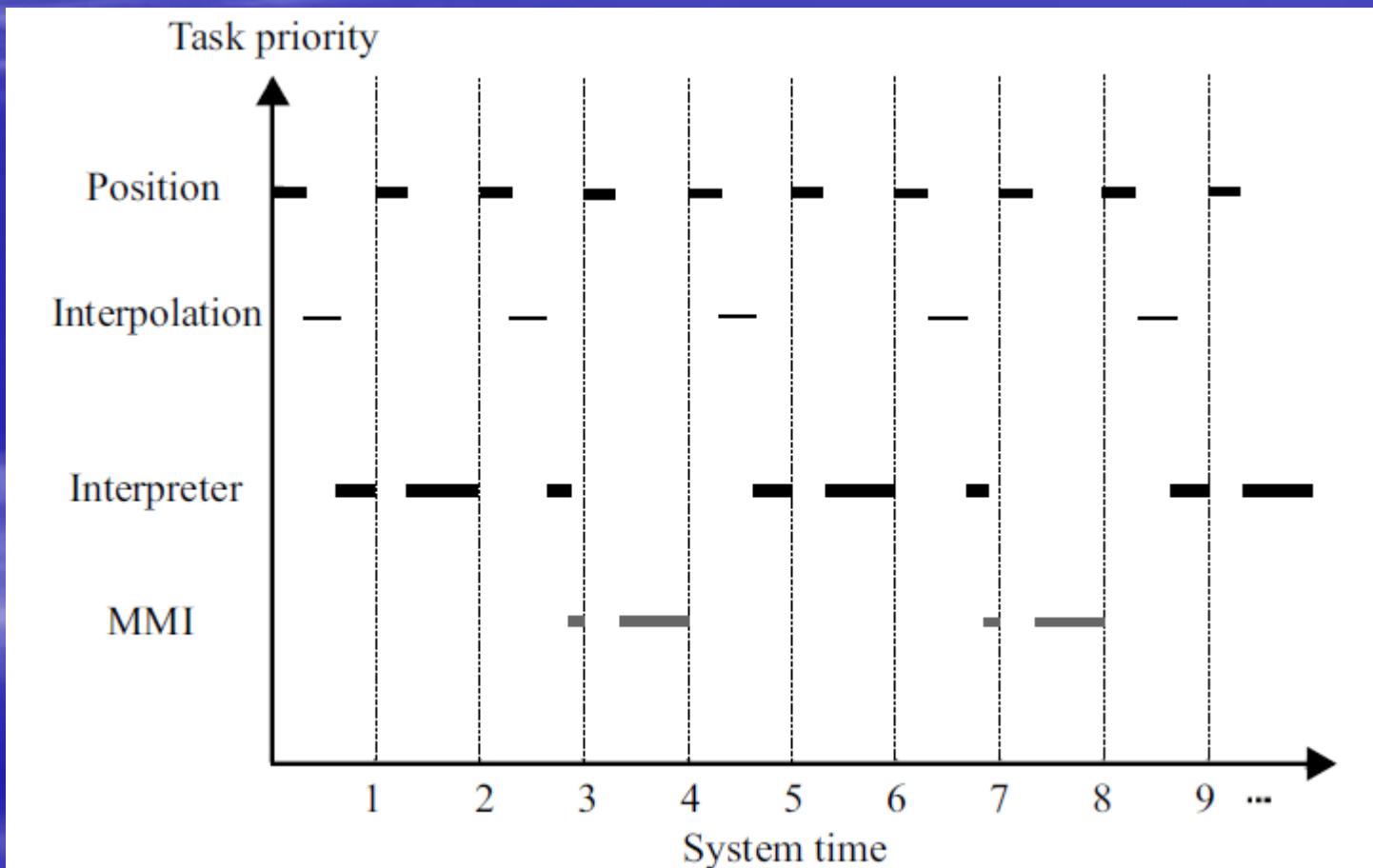
# Design of Soft-NC System

CPU	Position Time (ms)	Position Transit Time ( $\mu$ s)	IPO Time (ms)	IPO Transit Time ( $\mu$ s)	IPR Time (ms)	IPR Transit Time ( $\mu$ s)
Pentium-75	2	350	4	700	8	2600
Pentium-100	1	210	2	470	4	1800

Sampling time and execution time of each task

# Design of Soft-NC System

Task scheduling of Soft-NC

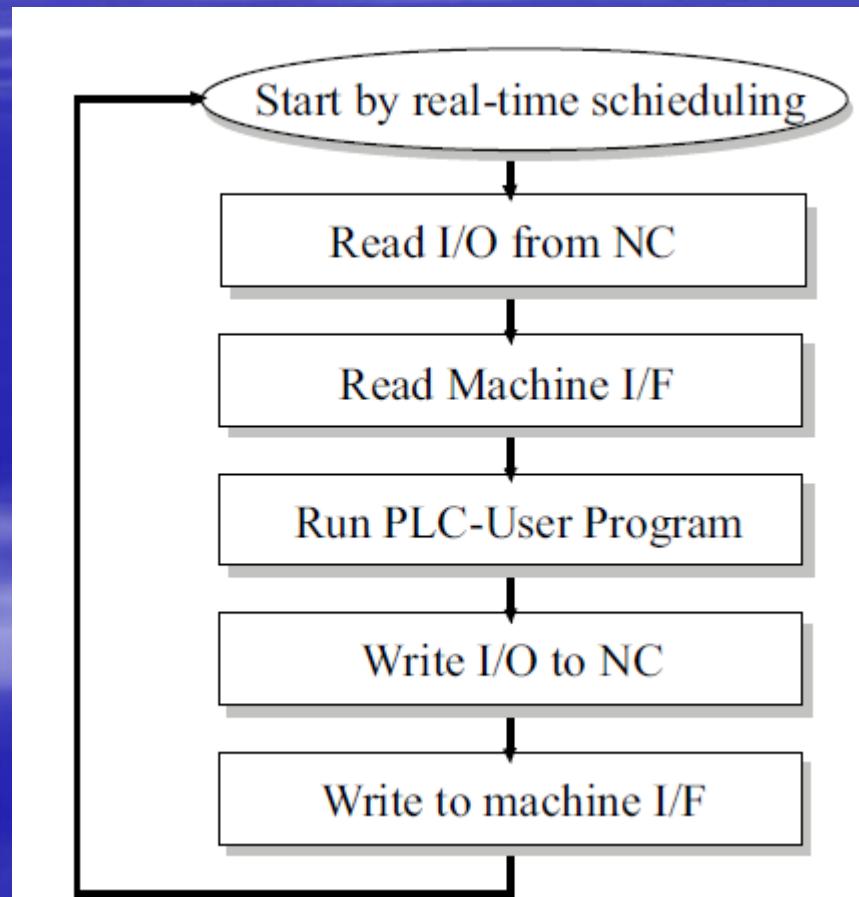


# Design of Soft-NC System

- *PLC Program Scanning and Scheduling*
- The PLC program scan cycle consists mainly of three steps:
  - Step 1: Checking input ports: first, PLC checks whether an input port is on or off status and then it stores these statuses in memory.
  - Step 2: Executing a program: PLC executes a program sequentially code by code. It calculates the status of output ports based on the status of the input ports obtained in Step 1. The calculated results are stored for use at the next step.
  - Step 3: Changing output: Finally, PLC changes the output port status based on the results from Step 1 and Step 2. After finishing Step 3, PLC returns to, and repeats Step 1.

# Design of Soft-NC System

Scan cycle of PLC program executor



# Design of Soft-NC System

## Priorities of PLC tasks

Interrupt Program	<ol style="list-style-type: none"><li>1. Irregularly executed from the input signal of a sensor.</li><li>2. The program with the highest priority among PLC tasks.</li><li>3. Used for handling signals requiring quick response.</li><li>4. Processing within hundreds of microseconds.</li><li>5. Feedback control routine performed by PLC system.</li></ol>
Fast Processing Program (Fast Task)	<ol style="list-style-type: none"><li>1. Repeated within every several milliseconds.</li><li>2. Highest priority among cyclic tasks.</li><li>3. Used for handling signals requiring quick response.</li><li>4. Processing under 1 ms.</li><li>5. Routine for counting position of turret and ATC.</li></ol>
Main Processing Program (Normal Task)	<ol style="list-style-type: none"><li>1. Repeated every tenth of a millisecond.</li><li>2. Program edited by user.</li><li>3. Program length should be controlled for executing every 10–20 ms.</li><li>4. PLC program edited in a variety of languages such as Ladder diagram.</li></ol>
Custom Program	<ol style="list-style-type: none"><li>1. Executing custom program after completing main program.</li><li>2. Program having the lowest priority.</li><li>3. Programs such as screen display and serial interface.</li></ol>

# Design of Soft-NC System

Priority assignment for various tasks

Task	Priority
CNC Position task	2
CNC Interpolation task	5
Fast PLC task	12
CNC Interpreter task	17
Normal PLC task	22
Comm task	36
MMI task	64

# Design of Soft-NC System

- ***Task Synchronization Mechanism***
- To synchronize all tasks of the CNC system an OS provides various mechanisms, such as **global variables**, **semaphores**, and **mailboxes**. In this section, we use the semaphore to design Soft-NC. The system kernel procedure for synchronization is as follows:
  1. Create the semaphores for the position control task and the interpretation task.
  2. Create the position control task and the interpretation task and then set them into idle status.
  3. Set an interrupt enabled by the timer for the position control task.
  4. On receiving the semaphore from the position control task, the interpretation task continues to execute its own routine.

# Design of Soft-NC System

- Task synchronization by semaphore

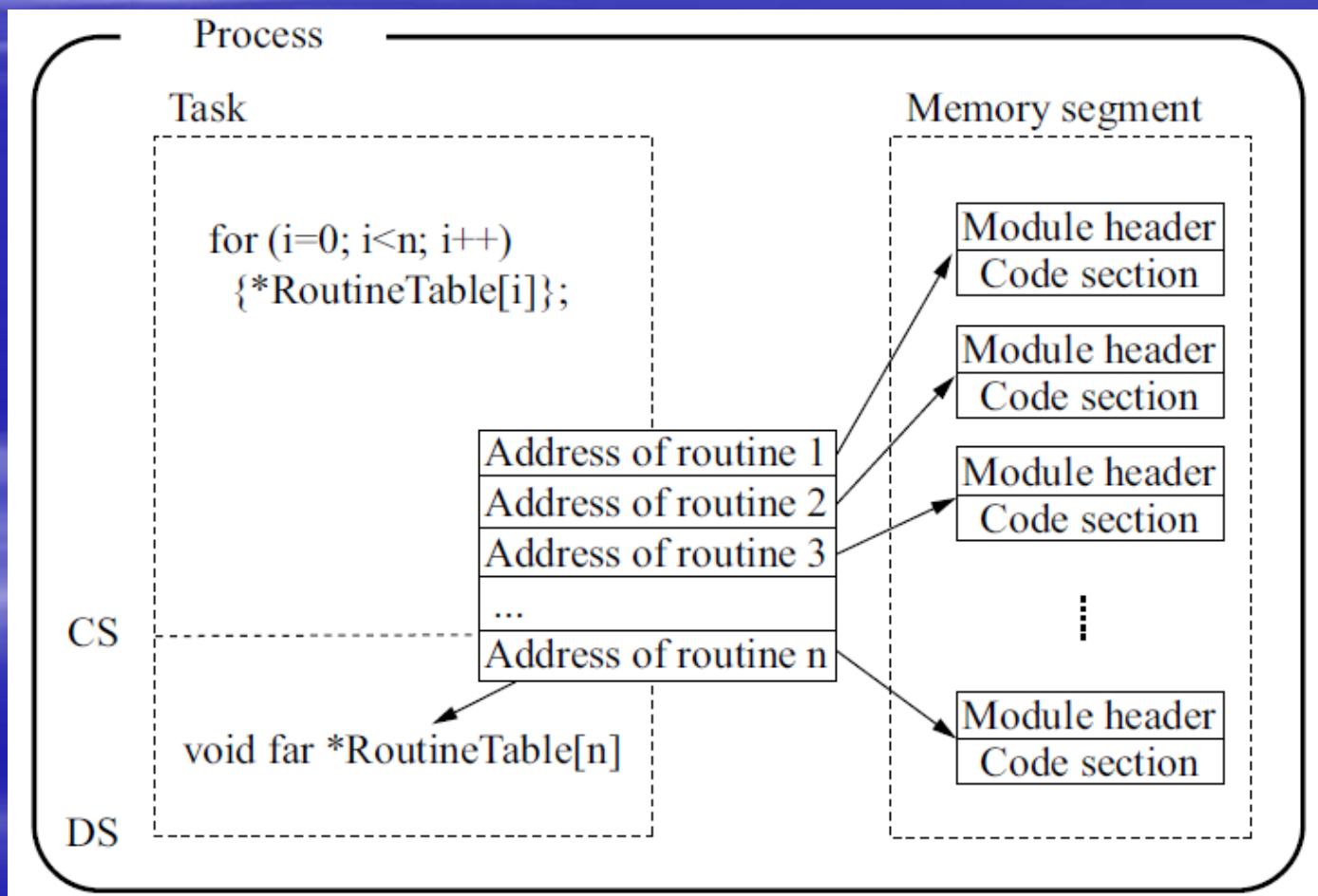
```
.....
while(True)
{
    IPR_status = idle;
    /* wait for semaphore from POS task */
    receive_unit(IPR_semaphore, ....);
    IPR_statue = busy;

    /* execute IPR routines */
    for (i=0; i<number of routine; i++)
        { routine table [i]; }

    IPR_Act_CNT++;
}
```

# Design of Soft-NC System

- Execution method of routines in a task



# Design of Soft-NC System

- *Task Synchronization Mechanism*

- ◆ **Task status:**

- this is the flag variable to denote the status of a task. If the task has completed before the interrupt, it has ‘idle’ value. If the task was interrupted during execution it has ‘busy’ value.

- ◆ **Task Set CNT:**

- This is computed by dividing Task Loop Time by Position Sampling Time. It defines the execution frequency of the task.

- ◆ **Task Act CNT:**

- means how many times the tasks are executed. So, whenever the task completes its own routine, this variable is increased by one.

# Design of Soft-NC System

```
PositionTask()
{
    .....
    IPO_Act_CNT = IPO_Act_CNT % IPO_Set_CNT;
    If(IPO_Act_CNT == 0)
    {
        if (IPO_statur = busy)
            return busy;
        else
            /* Send semahpore to activate IPO task */
            send_unit(IPO_semaphore);
    }

    IPR_Act_CNT = IPR_Act_CNT % IPO_Set_CNT;
    If (IPR_Act_CNT ==0)
    {
        if (IPR_status = busy)
            return busy;
        else
            /* Send semaphore to activate IPR task */
            send_unit (IPR_semaphore);
    }
    .....
}
```

# Design of Soft-NC System

## ■ Inter-Task Communication

Classification	Items	Comment
Data	Files	NC program, PLC program, NC parameter
	Command	Request service
	Variable	Various data set
Direction	NCK/PLC → MMI	Data for display
	MMI → NCK/PLC	Transfer information for op.
	PLC ↔ NCK	Bidirectional data
Frequency	Event	Data transfer if needed
	Cyclic	Data transfer periodically
Method	Shared memory	Dual port memory
	Comm task	Communication Task

Communication data classification of CNC system

# Inter-module communication data classification

Direction	Classification	Item	Comments
NCK/PLC → MMI	CNC Status	Actual Position/Velocity Distance to go Actual Lag, Velocity % Final Position for NC block Active G code Active Block number Selected Axis Tool information/ Correction	Display purpose Cyclic
	Part Program	Program number/name Subprogram number/ name Program repeat/ Program active	
	Parameter	Control Gain PLC Tool Programming	Display purpose Request/Answer
	System State	Emergency stop Cycle stop	Cyclic
	Spindle/Feed	Actual RPM/feed Programmed RPM/feed Maximum RPM/feed Minimum RPM/feed Override % Active Spindle number	Display purpose Cyclic
	PLC Information	Input/Output Byte information Timer information	Display purpose Request/Answer
	Error Handling	Error Level - Key/command delete - Control reset - Cold/warm start Error Type - Editor - Block processing - Servo - Hardware - PLC - Communication	Event
	Message Handling	Send Message	Event

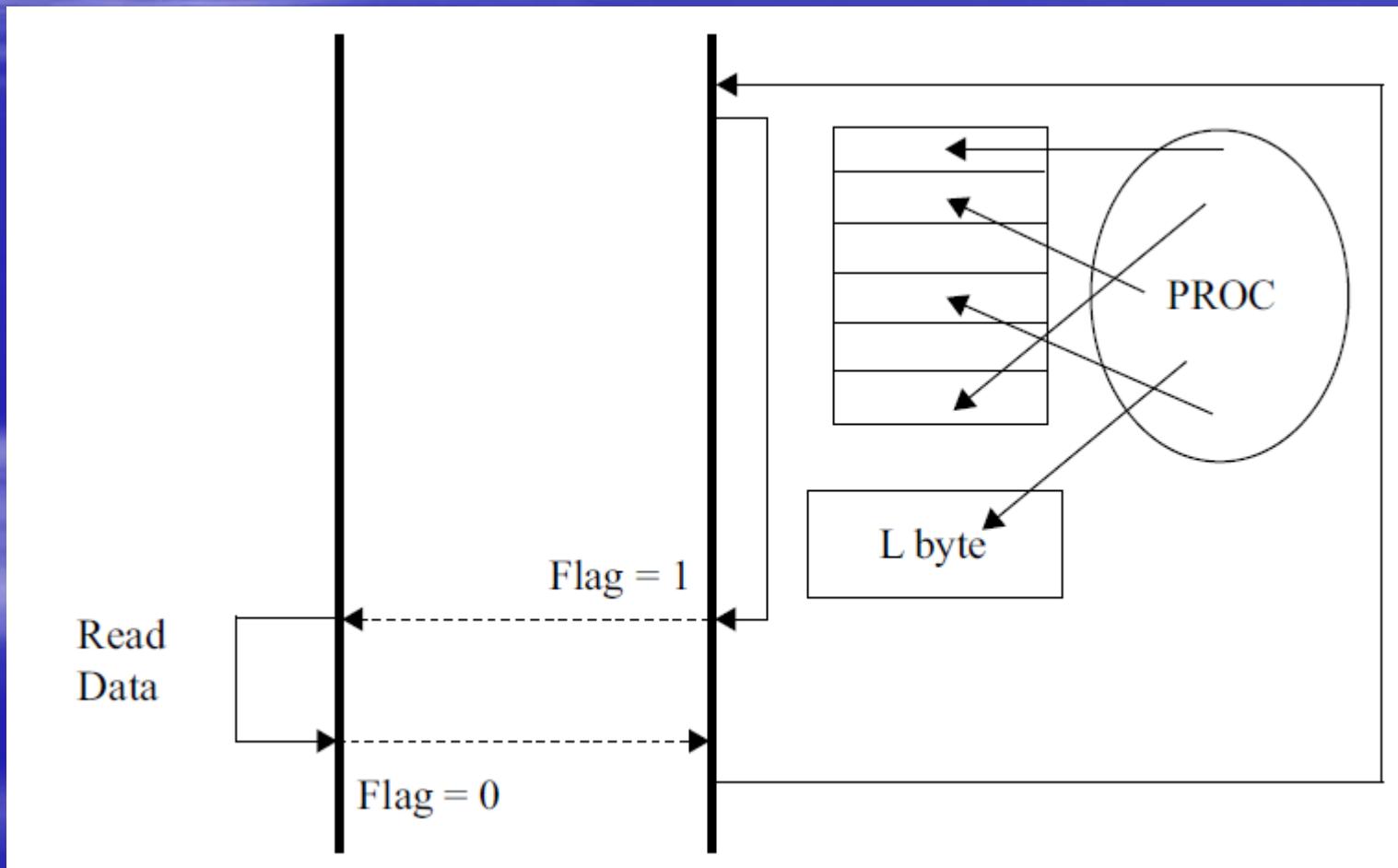
# Design of Soft-NC System

- Inter-module communication data classification

PLC ↔ NCK	Interface	Signal Input / Output A/D, D/A information Spindle Value/ Axis Position Handwheel values NC parameter NC correction	Cyclic
MMI → NCK/PLC	Operation Mode	Cycle Start/stop Auto/Manual/MDI Single block/Homing Warm start Jog+/Jog- Time(sec/min/hour)	Cyclic
MMI ↔ NCK	File Management	Program file Transfer System parameter PLC program loading Copy/Delete file	Request/Answer

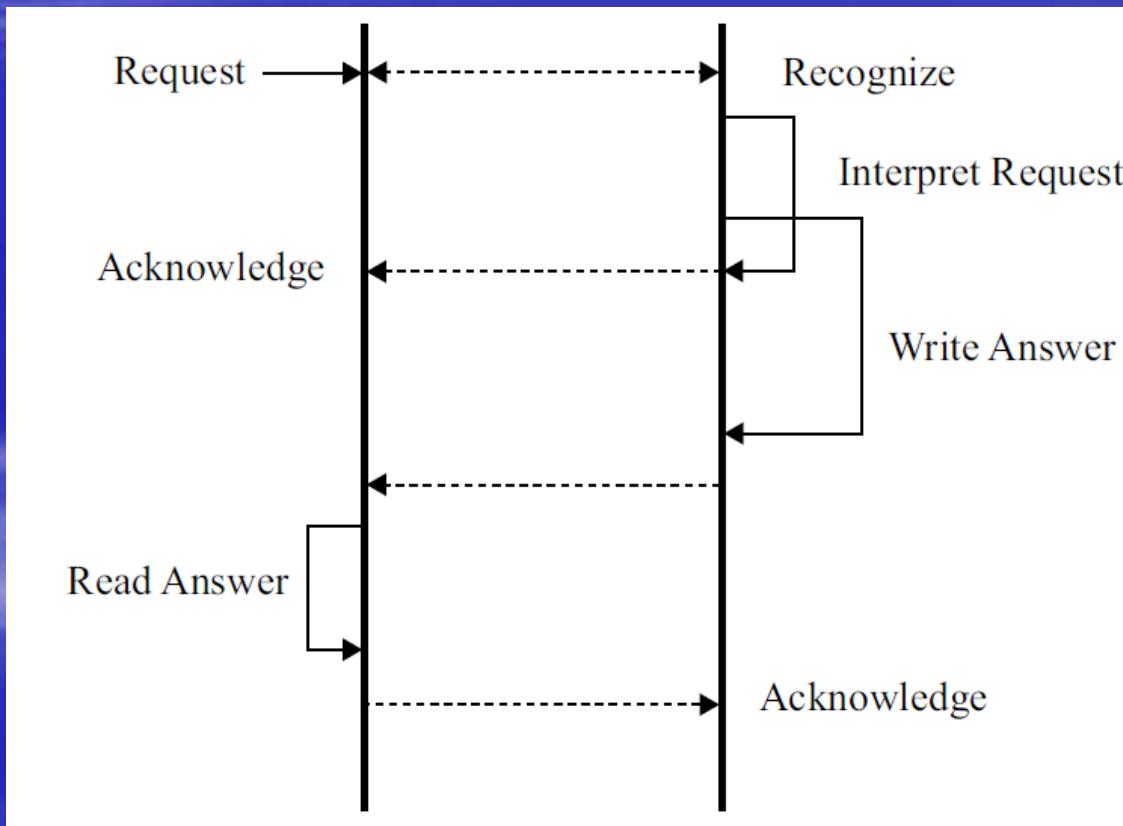
# Design of Soft-NC System

- Direct access method



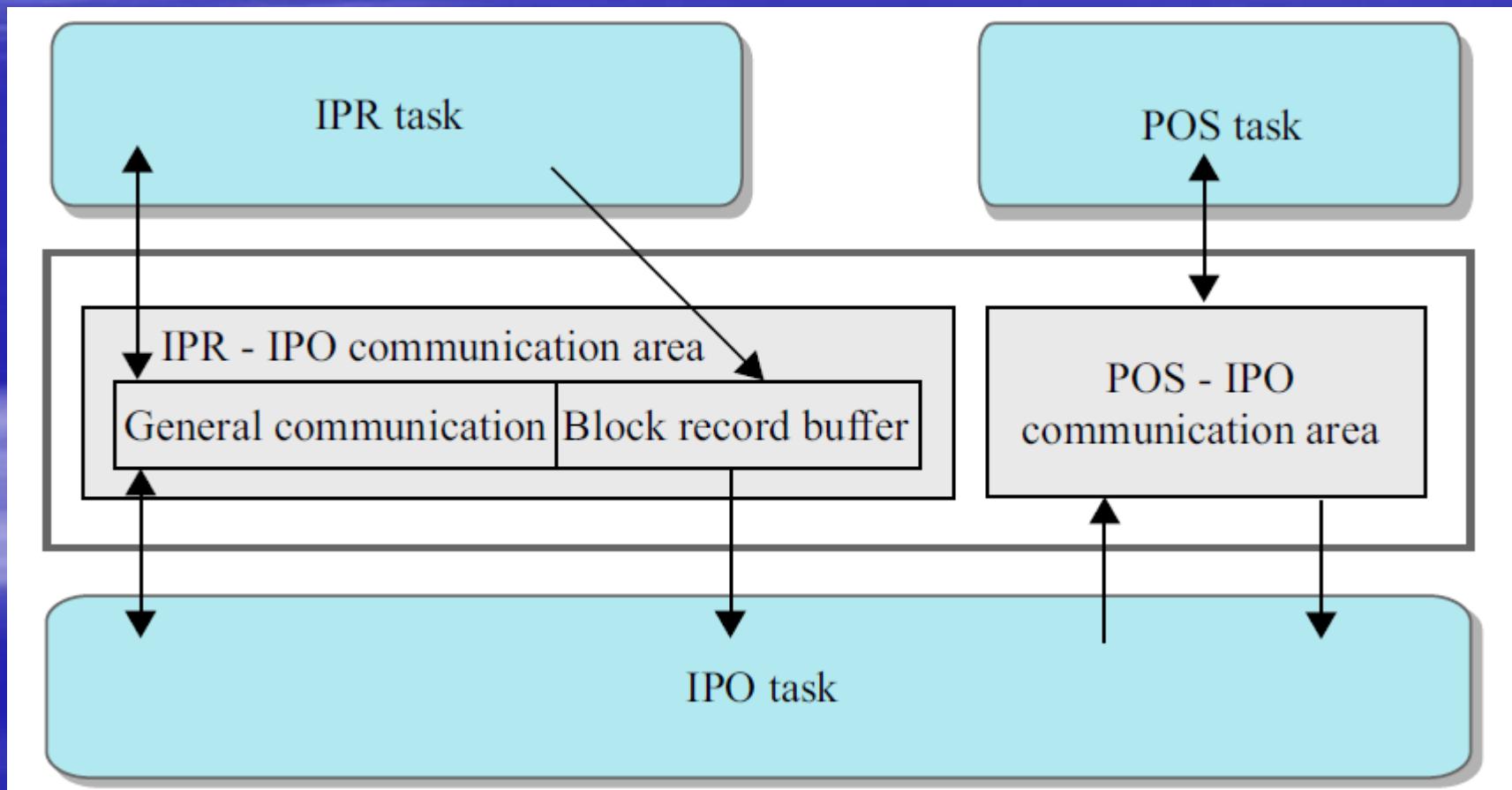
# Design of Soft-NC System

- Request/Answer method



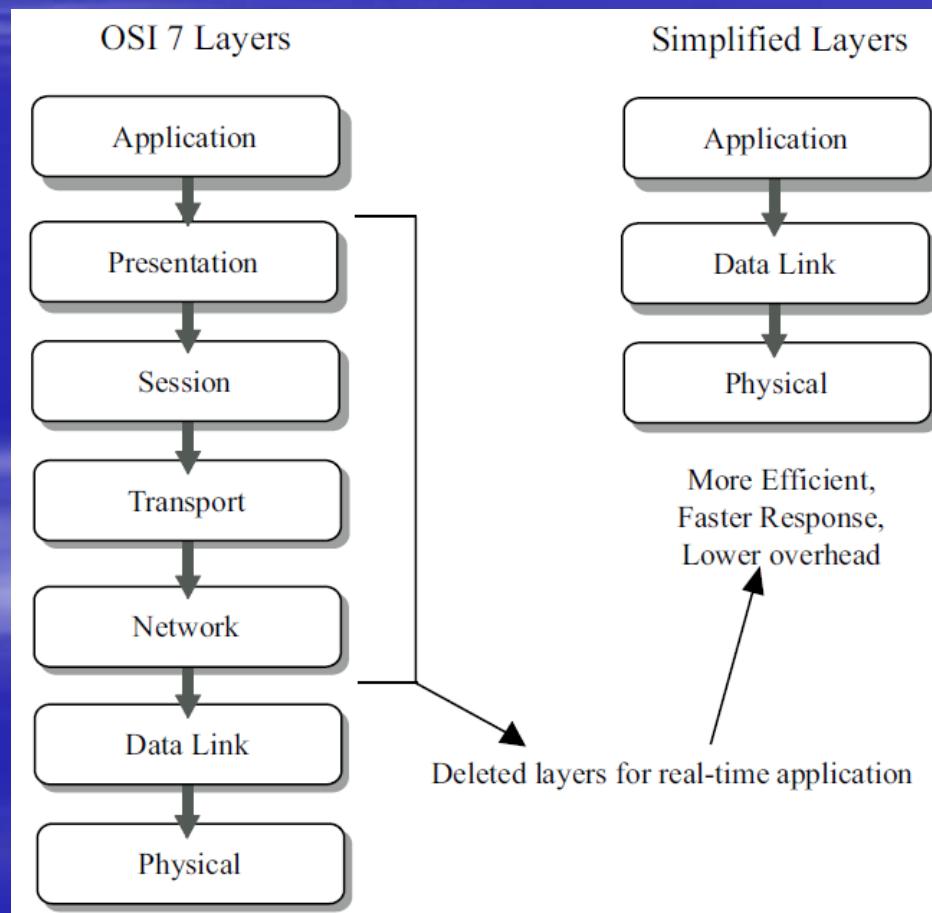
# Design of Soft-NC System

## ■ Inter-task Communication in NCK



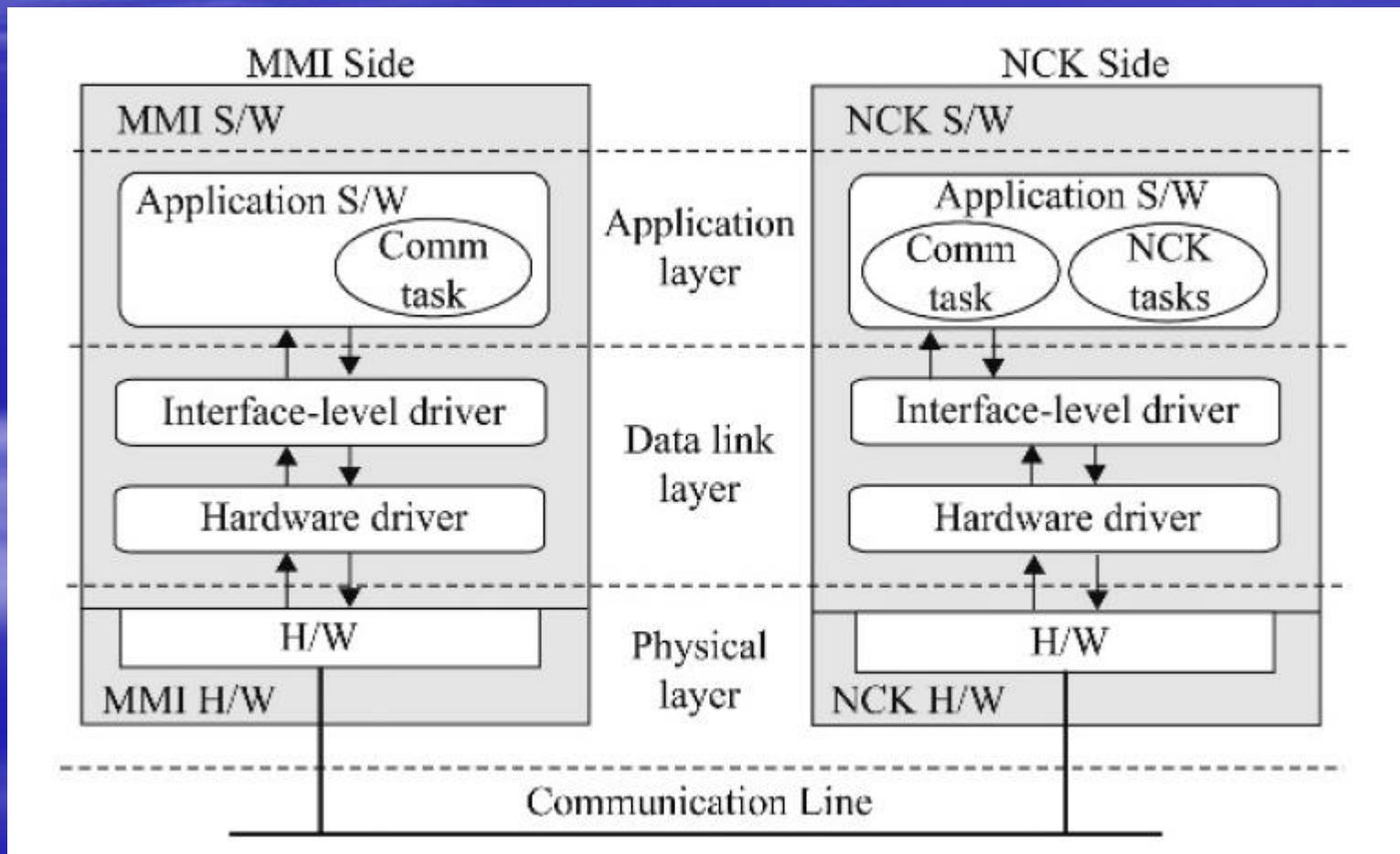
# Design of Soft-NC System

## ■ Communication Between NCK and MMI



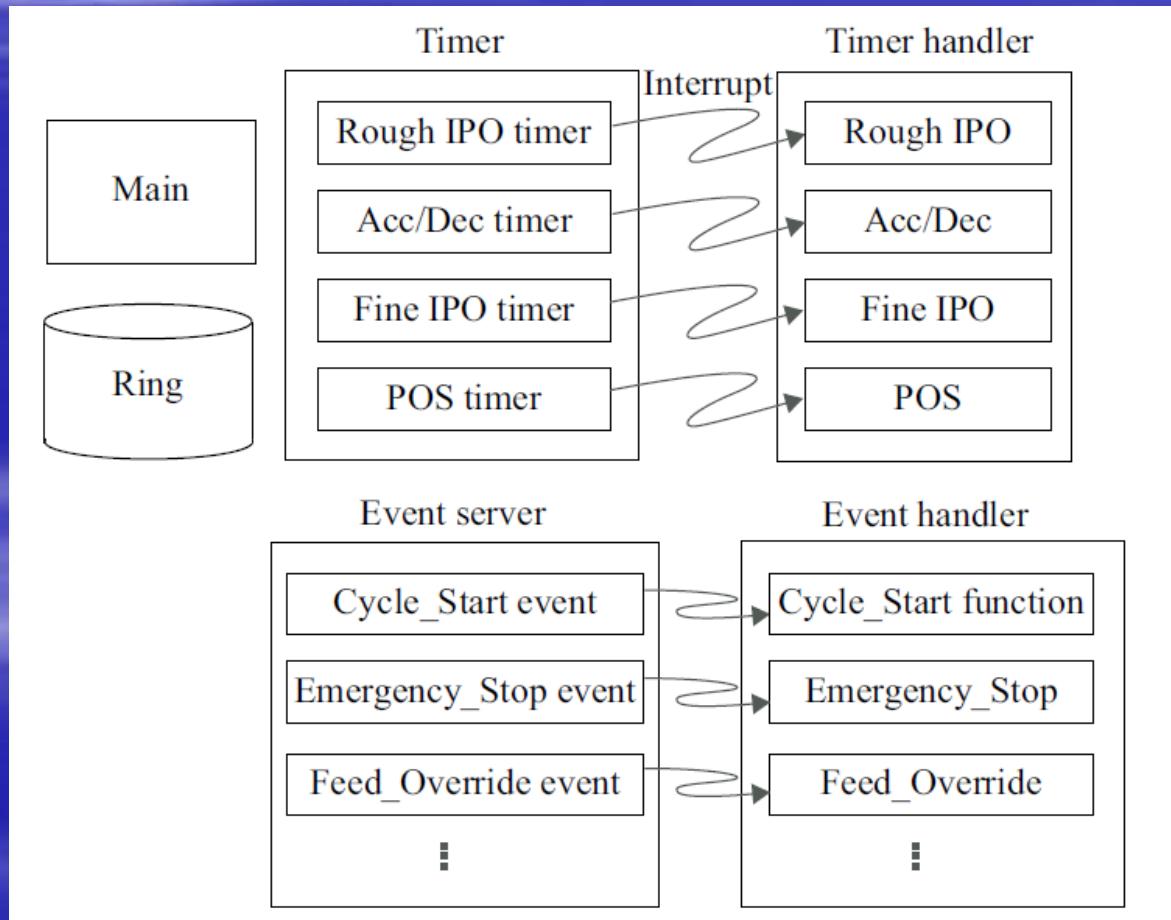
# Design of Soft-NC System

## ■ Communication Between NCK and MMI



# Design of Soft-NC System

- Motion Control System Programming Example



# Design of Soft-NC System

- *Design of System Architecture*
- The main function is the function that is called first when the NCK boots up and performs the following steps:
  1. To initialize internal variables required for execution of NCK.
  2. To create the timer and timer's handler for each task.
  3. To create the event server for handling the instruction from MMI.
  4. To create ring buffers for transmitting data between tasks.

# Design of Soft-NC System

## ■ *Creating Tasks*

An individual timer for each task is created. This timer activates iteratively particular handler functions every specified time. The following shows how to use the timer for creating the Acc/Dec control task using RTX API.

```
// ****
// Description: Create Timer, set start time and period //
// ****
hAccDec = RtCreateTimer(
    NULL,           // Security - NULL is none
    0,              // stack size - 0 is use default.
    DoAccDec,       // Timer handler
    NULL,           // NULL context
    P_ACCDEC,       // priority
    CLOCK_2);      // RTX HAL Timer
RtSetTimer(hAccDec, &Start_time4, &RTPeriod);
```

# Design of Soft-NC System

- *Task Synchronization*
- 執行順序

$$TS_{RIFO} < TS_{ACCDEC} < TS_{FIPO} < TS_{POS}$$

- continuous mode

$$TS_{RIFO} + 2 * P_{RIFO} < TS_{ACCDEC}$$

- the relationship between the sampling time and the start times of tasks

$$TS_{FIPO} < TS_{POS} < TS_{FIPO} + P_{POS}$$

# Design of Soft-NC System

## ■ Task Synchronization

```
// *****
// Description: Set start time and period for each timer //
// *****

// Set reference time.
RtGetClockTime(CLOCK_2, &Start_time);
// Start rough IPO after 20 milliseconds.
Start_time1.QuadPart = Start_time.QuadPart + 200000;
// Start ACC/DEC after 54 milliseconds.
Start_time4.QuadPart = Start_time.QuadPart + 200000 + 160000 +
160000 + 20000;
// Start fine IPO after 62 milliseconds.
Start_time3.QuadPart = Start_time.QuadPart + 540000 + 80000;
// Start POS after 79 milliseconds.
Start_time2.QuadPart = Start_time.QuadPart + 630000 + 160000;
// Set repeat for each timer.
RTPeriod.QuadPart = 160000;      // 16 msec
POSPeriod.QuadPart = 20000;      // 2 msec
// Create rough timer and start.
hRIPO = RtCreateTimer(
    NULL, // Security - NULL is none
    0, // stack size - 0 is use default.
    DoRIPO, // Timer handler
    NULL, // NULL context
    P_RIPO, // priority
    CLOCK_2); // RTX HAL Timer
RtSetTimer(hRIPO, &Start_time1, &RTPeriod);
```

# Design of Soft-NC System

- *Task Priority*

$$PR_{MMI} < PR_{RIPO} \leq PR_{ACCDEC} < PR_{FIPO} < PR_{POS} < PR_{ES}$$

# Design of Soft-NC System

## ■ *Inter-Task Communication*

```
// ****//  
// Description: Ring buffer structure and handling function //  
// ****//  
// Ring buffer structure for communication between IPR and rough IPO  
typedef struct CRingIRTag {  
    int nGCode;                      // IPO type 0 : G00, 1 : G01, 2 : G02  
    Vector Start;                    // Start position (mm)  
    Vector End;                     // End position (mm)  
    Vector Cen;                     // Center of circular IPO (mm)  
    float dRadius;                  // Radius of circular IPO (mm)  
    float dFeed;                    // Feedrate (mm/min)  
    float dSpindle;                 // Spindle speed (RPM)  
    short int nSpindleDir;          // CW: 1, CCW: 2, Stop: 0
```

# Design of Soft-NC System

## ■ *Inter-Task Communication*

```
int nStatus;           // block status 0: start, 1: end
int nControlMode;     // EXACTSTOPMODE
                      // /BLOCKOVERRAPMODE
int nBlockNumber;     // Index number of block
int nWorkingstepID;   // Working step ID
BOOL IsProgramEnd;    // TRUE: end of block, FALSE: under machining
struct CRingIRTag* next;
} CRingIR;
// Head and tail of buffer
typedef struct CIRListTag
{
    CRingIR* head;
    CRingIR* tail;
}CIRList;
```

# Design of Soft-NC System

## ■ *Inter-Task Communication*

```
// ****//  
// Description: Add item at tail of buffer //  
// ****//  
void CIRList_AddTail(CIRList* list, CRingIR* item)  
{  
    item->next = NULL;  
    if(list->tail == NULL) {  
        list->head = item;  
        list->tail = item;  
    }  
    else {  
        list->tail->next = item;  
        list->tail = item;  
    }  
}
```

# Design of Soft-NC System

## ■ *Inter-Task Communication*

```
// ****//  
// Description: Delete item structure at head of ring buffer //  
// ****//  
BOOL CIRList_RemoveHead(CIRList* list)  
{  
    CRingIR* temp;  
    if(list->head == NULL)  
        return TRUE;  
    if(list->head == list->tail) {  
        free(list->head);  
        list->head = list->tail = NULL;  
        return TRUE;  
    }  
    else {  
        temp = list->head;  
        list->head = list->head->next;  
        free(temp);  
        return FALSE;  
    }  
}
```

# Design of Soft-NC System

## ■ *Inter-Task Communication*

```
// **** //  
// Description: Delete all item structures in ring buffer. //  
// **** //  
void CIRList_DeleteAll(CIRList* list)  
{  
    BOOL bRtn;  
    while(1) {  
        bRtn = CIRList_RemoveHead(list);  
        if(bRtn)  
            break;  
    }  
}
```

# Design of Soft-NC System

## ■ *Inter-Task Communication*

```
// ****//  
// Description: Create a shared memory. //  
// ****//  
HANDLE hShm;  
DWORD dwMaximumSizeHigh = 0;  
PVOID locate;  
hShm = RtCreateSharedMemory(PAGE_READWRITE,  
                           dwMaximumSizeHigh, sizeof(struct NC_Status_DB),  
                           "StatusDBName", &locate);  
RtUnmapSharedMemory(locate); // Unmapping of created memory  
RtCloseHandle(hShm); // Close handle of shared memory
```

# Design of Soft-NC System

## ■ *Inter-Task Communication*

```
// **** //  
// Description: Access the previous shared memory. //  
// **** //  
hShm = RtOpenSharedMemory(SHM_MAP_WRITE, FALSE,  
                           "StatusDBName", &locate);  
  
pStatus = (NC_Status_DB*)locate;  
pStatus->CommandX = CurPos_X.dCommand;  
pStatus->CommandY = CurPos_Y.dCommand;  
pStatus->CommandZ = CurPos_Z.dCommand;  
pStatus->CurrentX = CurPos_X.dGmOut;  
pStatus->CurrentY = CurPos_Y.dGmOut;  
pStatus->CurrentZ = CurPos_Z.dGmOut;  
pStatus->CurrentFeed = ActualFeed;  
RtUnmapSharedMemory(locate);  
RtCloseHandle(hShm);
```

# Design of Soft-NC System

## ■ *Create Event Service*

```
// ****//  
// Description: Create Event //  
// ****//  
// Create event object.  
hStartEvent = RtCreateEvent(  
    NULL, //security attribute  
    FALSE, //manual reset  
    FALSE, //initial state  
    "NCKSTARTEVENT" //the event name  
);  
// Create thread to handle the event.  
hStartHandler = RtCreateThread(0, 0, StartHandler, NULL,  
                            CREATE_SUSPENDED, 0 );  
// Priority assignment of thread.  
if (RtSetThreadPriority(hStartHandler, STARTPRIORITY) == FALSE)  
{  
    RtPrintf("RtSetThreadPriority error = %d  
n",GetLastError());  
    ExitProcess(1);  
}  
// Start thread.  
dwSuspendCount = RtResumeThread(hStartHandler);
```

# Design of Soft-NC System

## ■ *Create Event Service*

```
// *****
// Description: Handling "Cycle start" thread. //
// *****

// Function to handle 'cycle start' event.
ULONG RTFCNDCL StartHandler(void * nContext)
{
    DWORD dwEventReturn;
    while(1) {
        // Wait until Cycle start button is pushed.
        dwEventReturn = RtWaitForSingleObject(hStartEvent, INFINITE);
        Sim_total_count = 0;
        // Start NCK.
        StartNCK();
    }
    return(0);
}
```

# Design of Soft-NC System

- *Create Event Service*

```
// ****//  
// Description: Open the created Event object. //  
// ****//  
hStartEvent = RtOpenEvent(EVENT_MODIFY_STATE, TRUE,  
                         "NCKSTARTEVENT");  
  
RtSetEvent(hStartEvent);  
RtCloseHandle(hStartEvent);
```

# Design of Soft-NC System

- After development by MIT in the early 1950s, CNC systems have advanced with the appearance and advancement of the microprocessor. With the introduction of automation systems in the 1970s, the function of CNC systems has made rapid progress. However, due to the complexity of NC technology, which requires not only fundamental control function but also various auxiliary technologies such as machining technology, process planning technology, and manufacturing technology, the market for NC systems has been dominated by a few market leaders in **Japan and Germany**. The advanced manufacturers evolved CNC systems into **closed systems** in order to prevent their own technology from leaking out and keeping their market share.

# Design of Soft-NC System

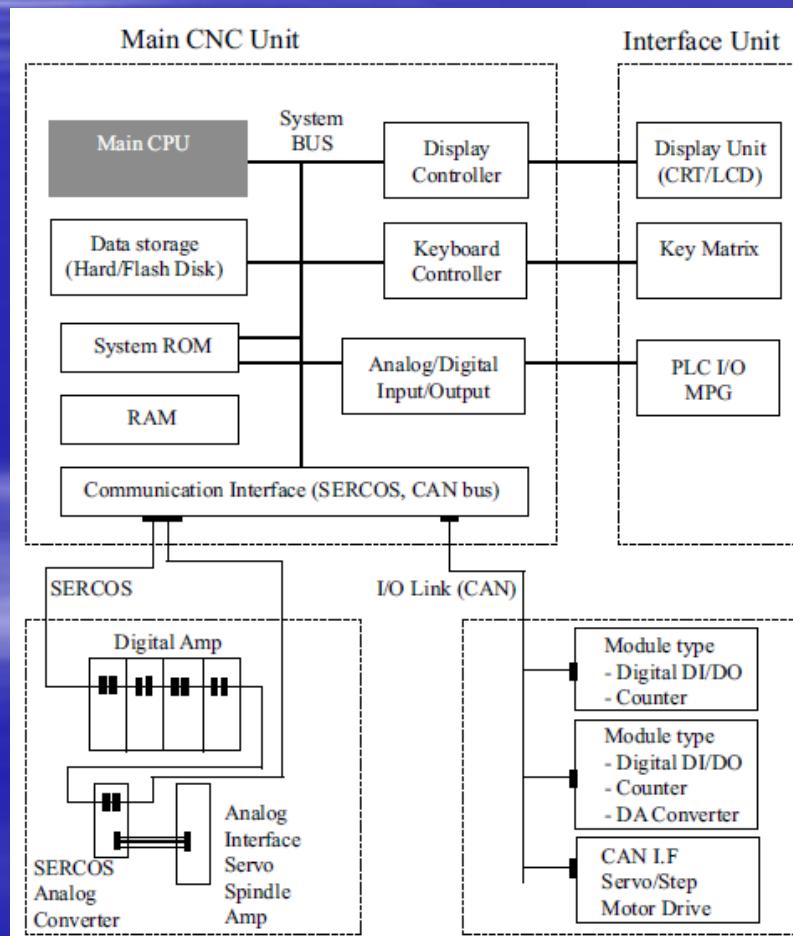
- However, after the middle of the 1980s, a new manufacturing paradigm, where computer network and optimization techniques were applied to manufacturing systems with the progress of computer technology, has appeared together with the requirement for advanced control functions for **high-speed and high-accuracy** machining. Closed CNC systems were not adequate for realizing the new manufacturing paradigm. The architecture of closed CNC systems could not meet the user's requirements and improvement of CNC systems was possible, not by MTB (Machine Tool Builders) but CNC makers. The limited resources of CNC makers made it impossible to meet the new paradigm.

# Design of Soft-NC System

- Therefore, various efforts to develop open CNC systems have been tried. As a typical result of these attempts, PC-NC that was introduced in the early 1990s. Like IBM PC technology, which appeared in the early 1980s, has progressed by third-party developments based on openness, CNC systems have progressed to PC-NC based on the openness of PC technology. **However, now, despite low price, openness, and many developers of PC-NC, the lack of reliability and openness to application S/W has made it impossible to implement perfectly open systems.**

# Design of Soft-NC System

## ■ Closed-type CNC Systems



# Design of Soft-NC System

- A CNC system has analog/digital input/output devices for communication with the machine and communication interface. In the past, in CNC systems, communication between NC equipment and motors and drives was done by analog signals and the communication interface was very simple. Because of **the problem of noise**, digital communication has now typically come to be used and SERCOS is a typical digital communication method. With the usage of fiber optical cable, digital communication makes the exchange of various data and the removal of noise possible. Therefore, it is becoming possible that the CNC system adjusts the parameters of servo drives and motors directly and that the CNC system monitors the status of the servo system in real time. Improvement of machining accuracy has become possible due to the removal of noise. In addition to communication with the servo system, digital communication has been applied to communication with input/output devices. In order to enable communication with a variety of sensors and machine components via a single communication line, a standardized communication method is required. For this, various kinds of field bus, such as Profibus, CAN Bus and InterBus-S, have been introduced, but **one standard method has not yet been established.**

Item	Requirement
Reconfigurability	<p>In the case of machining an engine cylinder block of a car, about 80% of machining does not require high precision machining and not only hole drilling and plane milling are performed. The CNC system used in this machining does not require a variety of functions in the user interface but functions about automation. Therefore, the functionality of the CNC system can be <u>added to or subtracted from according to the user's requirements.</u></p>
Extensibility	<p>The hardware and software, such as the number of controllable axes, cycle program, and program storage are <u>independent</u> in terms of functionality. They can be <u>reconfigured</u> if needed.</p>
Program	<p>Part programming and macro programming based on EIA are very complicated and each CNC maker provides their own special functions. In the case of using CAD/CAM, there are <u>many problems with exchanging data between software programs and devices</u>. To solve these problems, a new CNC programming language is required.</p>

Advanced function	<p>In the case of milling machining for mold and die, surface interpolation functions for machining free-form surfaces are needed in order to avoid grinding operations for post-milling operations. Sensor-based feedback control for high-precision machining is also needed. Therefore, when it is necessary to apply new technology, <u>the addition of new functions should be possible.</u></p>
Intelligence	<p>The cutting conditions for machining should be determined depending on the diameter of the tool and the materials of the workpiece and tool. As the selection of cutting conditions requires much know-how, automation of the selection in the CNC system is insufficient and <u>an intelligent CNC system is required for optimal process planning and optimal toolpath generation.</u></p>
Standardization	<p>Despite the fact that a variety of machines are used together in the field, <u>they cannot communicate with each other unless their CNC equipment is the same.</u> A CNC system has limitations on communication with PC and FA controllers. In addition, the options provided by CNC makers depend on the CNC makers and are expensive. To solve these problems, standardization and openness are required.</p>

# Design of Soft-NC System

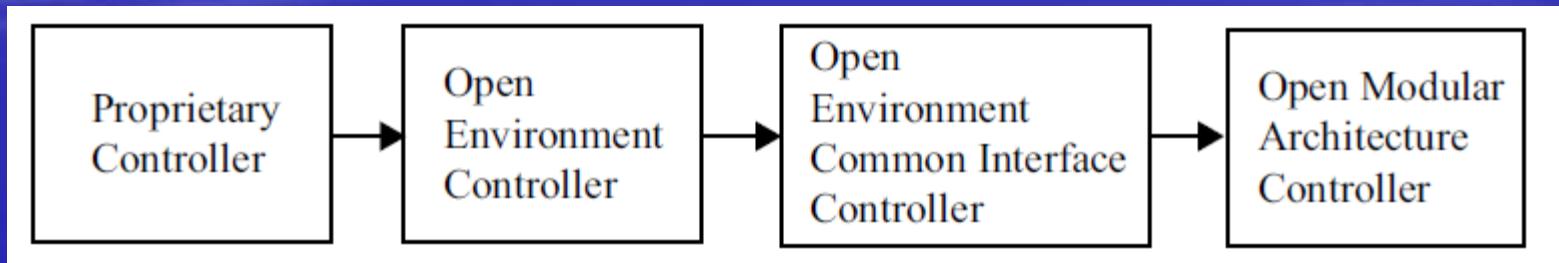
- *Open CNC Systems*
- An open system is defined as a system that satisfies the following
  1. **Interoperability:** This means the ability that the components that compose the system cooperate to perform the specified task. For this ability, the standard specifications of the data representation language, behavior model, physical interface, communication mechanism, and interaction mechanism are needed. **A bus-based system design is most important.**
  2. **Portability:** This means the ability for a component to be executed on the CNC system with different hardware or different software. Portability is very important from the commercial point of view. Since this means that a **hardware device or a software module can be used on various platforms**, it contributes to increasing the efficiency of a platform.

# Design of Soft-NC System

- *Open CNC Systems*
  3. **Scalability:** This means the ability **to make extensions to or reductions of the system's functionality possible without large cost.** Adding memory or a board to a PC is a typical example.
  4. **Interchangeability:** This means the ability **to replace the existing component with a new component.** Instead of replacing the whole system, replacing an existing motion board with a motion board with a new algorithm is a typical example.

# Design of Soft-NC System

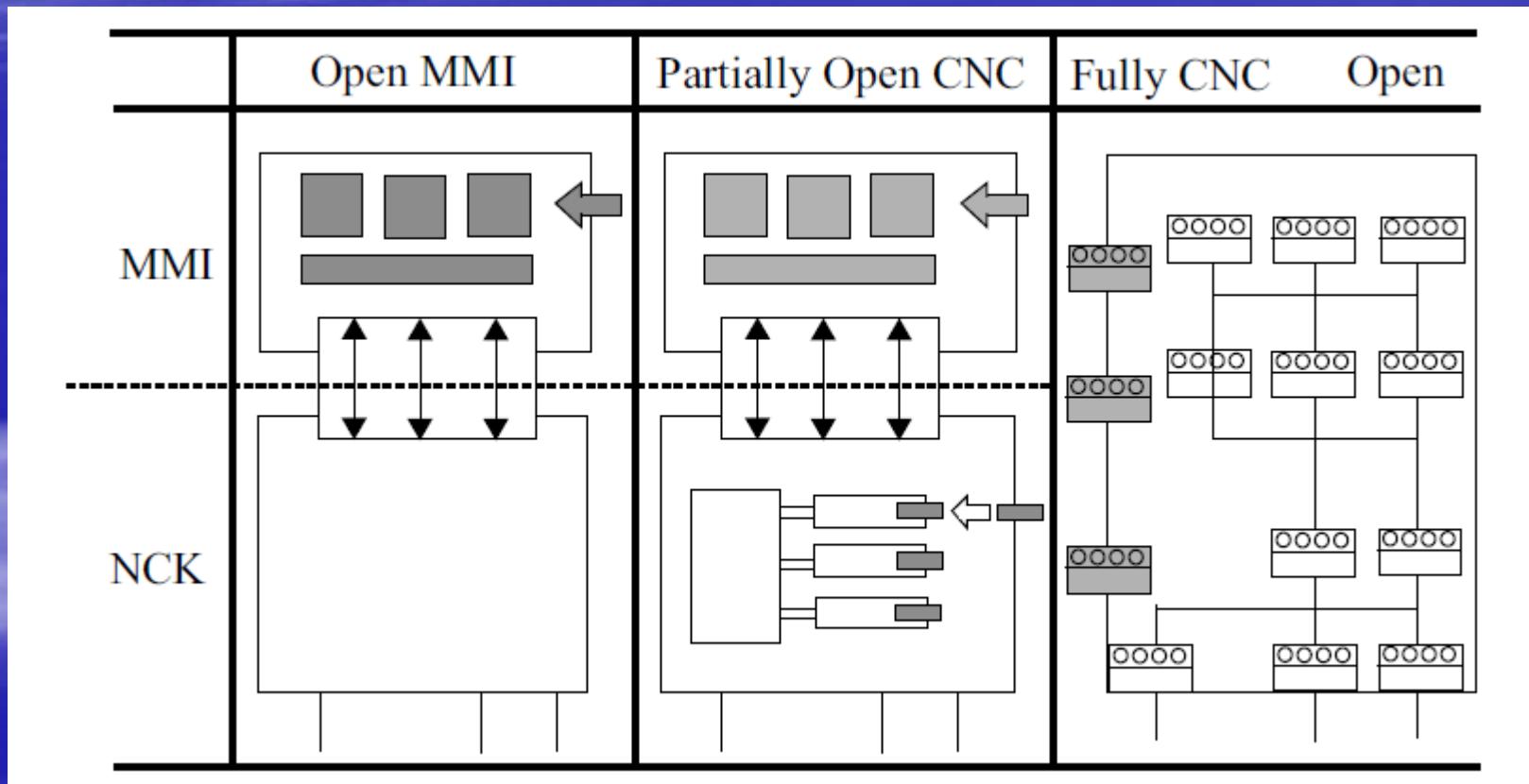
- Classification of an Open System
- Progress of Open System development



- A system with open environment controller, with each piece of equipment connected to another via an **open communication network**. As the second step, a system with **open environment common interface controller**, portability, interchangeability, and scalability can be realized. As the last step, a system with **open modular architecture controller**, a variety of **applications based on distributed network technology and component technology** can be implemented.

# Design of Soft-NC System

- open CNC system can be classified into three types



# Design of Soft-NC System

- **Summary**
- PC-NC, being the alternative for replacing a closed and expensive hardware-based NC, is distinguished from the traditional hardware-based NC in terms of hardware architecture, software model, and communication mechanism. The configuration of PC-NC is classified into the following three types:
  1. PC is used for MMI and the closed-type motion control boards for NCK and PLC are inserted into the PC.
  2. Two PCs are used for MMI and NCK/PLC, respectively. The two PCs are connected via high-speed communication.
  3. MMI, NCK, and PLC are implemented as software tasks operating in a multiprocessing environment using a single CPU.

# Design of Soft-NC System

- **Summary**
- In terms of Soft-NC design, a variety of modules are regarded as tasks and they can be divided into non-cyclic tasks and cyclic tasks. A **non-cyclic task** is a task that does not require tight response time, such as MMC, interpreter, and external communication manager. A **cyclic task** is a task that requires the hard real-time property. For developing Soft-NC, it is necessary to design a **scheduler**, **synchronization** between tasks, and **communication** between tasks. Moreover, this design is realized by real-time programming techniques.

# Theory and Design of CNC Systems

*Chapter 11*

## **STEP-NC System**

# STEP-NC System

With the rapid advancement of information technology associated with NC technology, the manufacturing environment has changed significantly since the last decade. However, the low-level standard, G&M codes, have for over 50 years been used as the interface between CAM and CNC, and are now considered as an obstacle for global, collaborative and intelligent manufacturing. A new model of data transfer between CAD/CAM systems and CNC machines, known as STEP-NC is being developed worldwide to replace G&M codes. In this chapter, we will give an overview of STEP-NC and its related technology, including data models for STEP-NC, CNC systems based on STEP-NC, namely STEP-compliant CNC systems, together with worldwide research status and future prospects.

**STEP:** STandard for the Exchange of Product model data

# Introduction

## ■ STEP-NC

### – A narrow sense (in this book)

- A new interface language between CAM and CNC.

### – A broad sense

- A new interface language between CAM and CNC.
- The technologies to implement CAD/CAM and CNC software.
- The products based on the new interface.

## ■ ISO 14649

- An international standard specification defines the data model for STEP-NC.
- It specifies **information contents and semantics** (ICS) for various CNC manufacturing processes and resources including cutting tools and machine tools.

# Introduction

- **STEP-NC data model**
  - The contents of ISO 14649.
  - It is same as the narrow meaning of STEP-NC.
- **STEP compliant CNC**
  - A kind of new CNC controller implementing STEP-NC.
  - Depending on how the STEP-NC is interfaced and used, STEP-compliant CNC is classified into 3 types.
    - 1) Conventional.
    - 2) Basic.
    - 3) Intelligent.

# Introduction

- **STEP-CNC**

- The abbreviation of STEP-compliant CNC.

- **STEP-NC technology**

- Various technologies required for implementing software and products based on the STEP-NC interface.

# Background of STEP-NC

## ■ Problems with G&M Codes

### 1) Information loss

- A G&M-code part program is defined by simple alphabetical or numerical codes such as G, T, M, F, S indicating the movement of a machine and an axis to the controller.
- Since this delivers only limited information to the CNC, it makes the CNC simply an executing mechanism, completely unaware of the motions being executed.

### 2) Difficult traceability

- As a G&M-code part program is made up of a coded set of numbers for axis movements, it is not easy for machine operators to understand the operational flow, machining condition and specification of tools only by reading the low-level part program.
- It makes it more difficult, not only in finding which part happens to cause problems, but also modifying the program for solving these problems.

# Background of STEP-NC

## ■ Problems with G&M Codes

### 3) Lack of interoperability

- The G&M code schema is dependent on the machine tool builder or controller maker.
- The part program for a certain targeted controller cannot be applied to another heterogeneous controller.

### 4) Non-compatibility with higher level systems

- The rich information environment has almost perished at the CNC on the shop floor level.
- Also, there is little information feedback from the CNC, which makes the shop floor status obscure to the upper systems.

# Comparison of G-code and STEP-NC

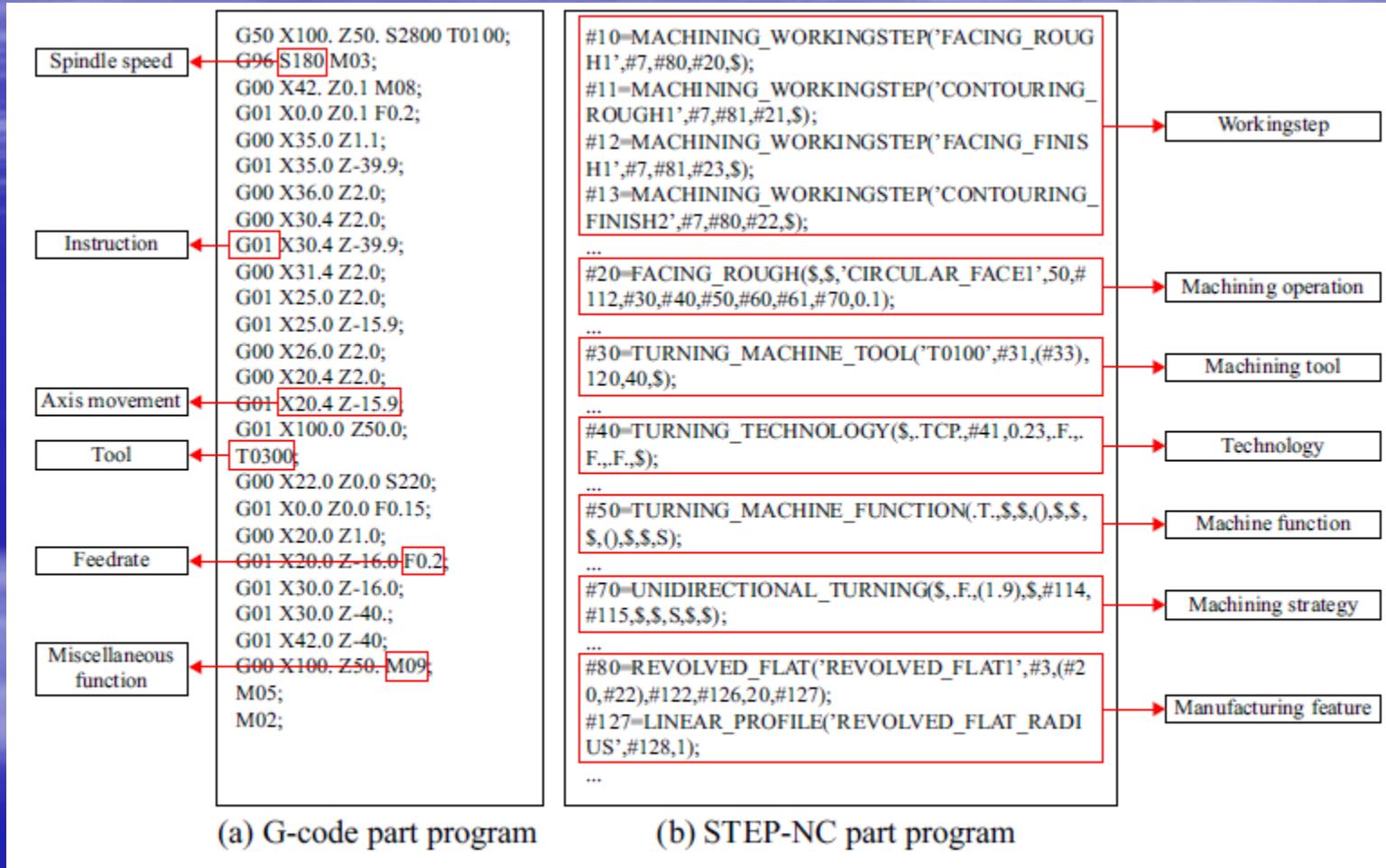


Fig. 11.2 Comparison of G-code part program and STEP-NC part program

# Comparison of G-code and STEP-NC

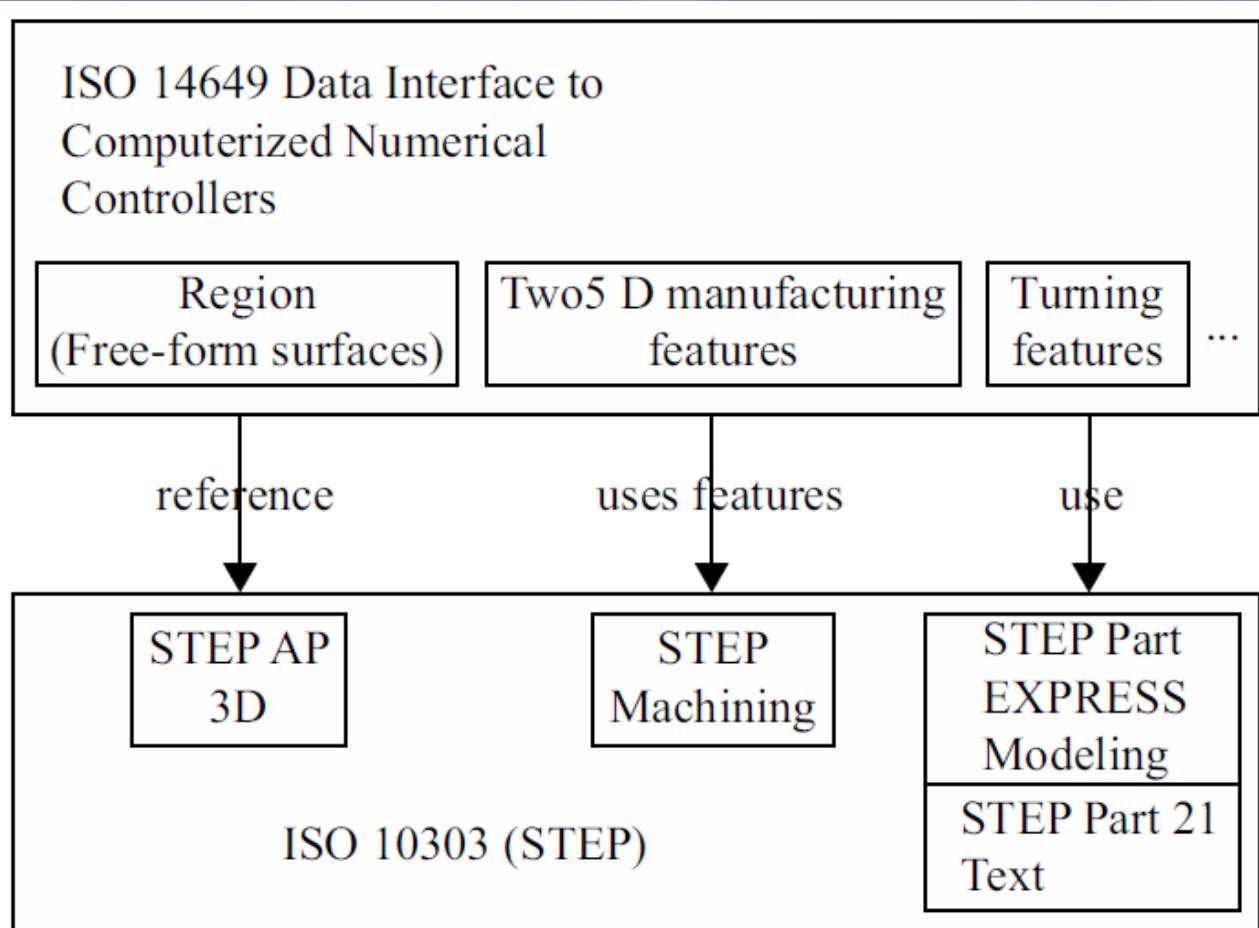
## ■ G-code part program

- It is very difficult for machine operators to understand the operational flow, machining conditions and specification of tools only by reading a part program.
- It is impossible for the CNC controller to execute an autonomous and intelligent control and to cope with emergency cases with this limited information.

## ■ STEP-NC part program

- It contains workingstep, machining feature, machining operation, machining tool, machining strategy, machine function and workpiece.
- It includes a much richer information set including ‘what-to-make’ (geometry) and ‘how-to-make’ (process plan).

# STEP and STEP-NC



**Fig. 11.3** The relationship between ISO 14649 and ISO 10303

# STEP-NC Data Model

**Table 11.1** ISO Current status of Parts in the ISO 14649 (December, 2007)

ISO 14649	Title of documents	Edition	Status
Part 1	Overview & fundamental principles	1	IS
Part 10	General process data	1	IS
Part 11	Process data for milling	1	IS
Part 12	Process data for turning	1	IS
Part 13	Process data for wire-EDM	2	CD
Part 14	Process data for sink-EDM	2	CD
Part 15	Contour cutting	2	WD
Part 16	Process data for inspection	2	WD
Part 17	Process Data for rapid prototyping	2	WD
Part 110	Machine tools for general process	2	NWIP
Part 111	Tools for milling machines	1	FDIS
Part 121	Tools for turning machines	1	IS

IS : International Standard, CD : Committee Draft,

WD : Working Draft, NWIP : New Work Item Proposal,

FDIS: Final Draft International Standard

# STEP-NC Data Model

- ISO 14649 Part 1: Overview and Fundamental Principles
  - It describes the fact that the ISO 14649 data model is composed of three levels.
    - Level A deals with the modeling of the manufacturing technologies.
    - Level B deals with integration and compatibility with ISO 10303.
    - Level C deals with adoption software, which is the implementation of Level A or B in controllers.
  - The most important feature of the ISO 14649 data model is to remedy the shortcoming of ISO6983 by specifying machining process rather than machine tool motion, using the object-oriented concept of the workingstep.

# STEP-NC Data Model

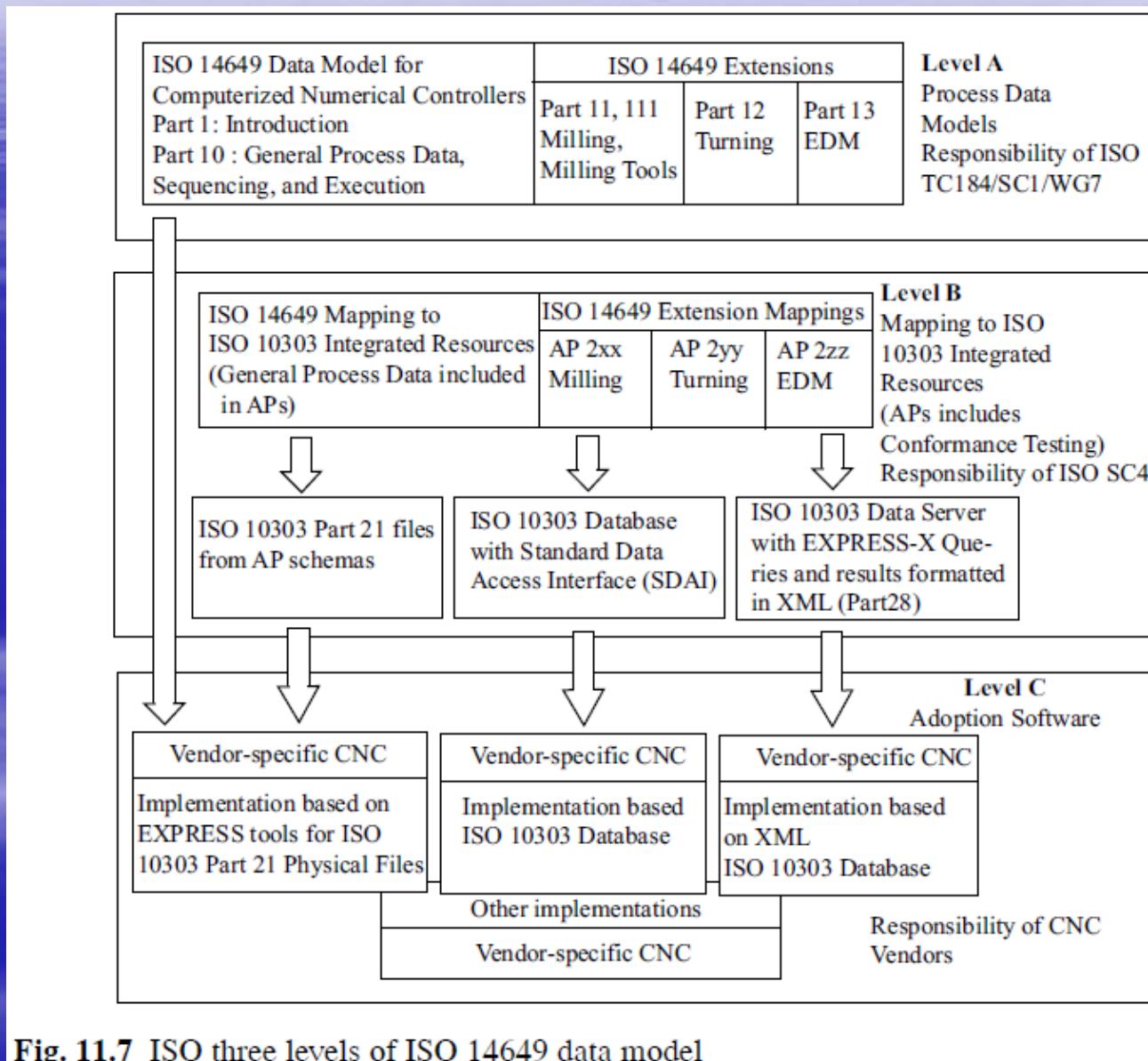


Fig. 11.7 ISO three levels of ISO 14649 data model

# STEP-NC Data Model

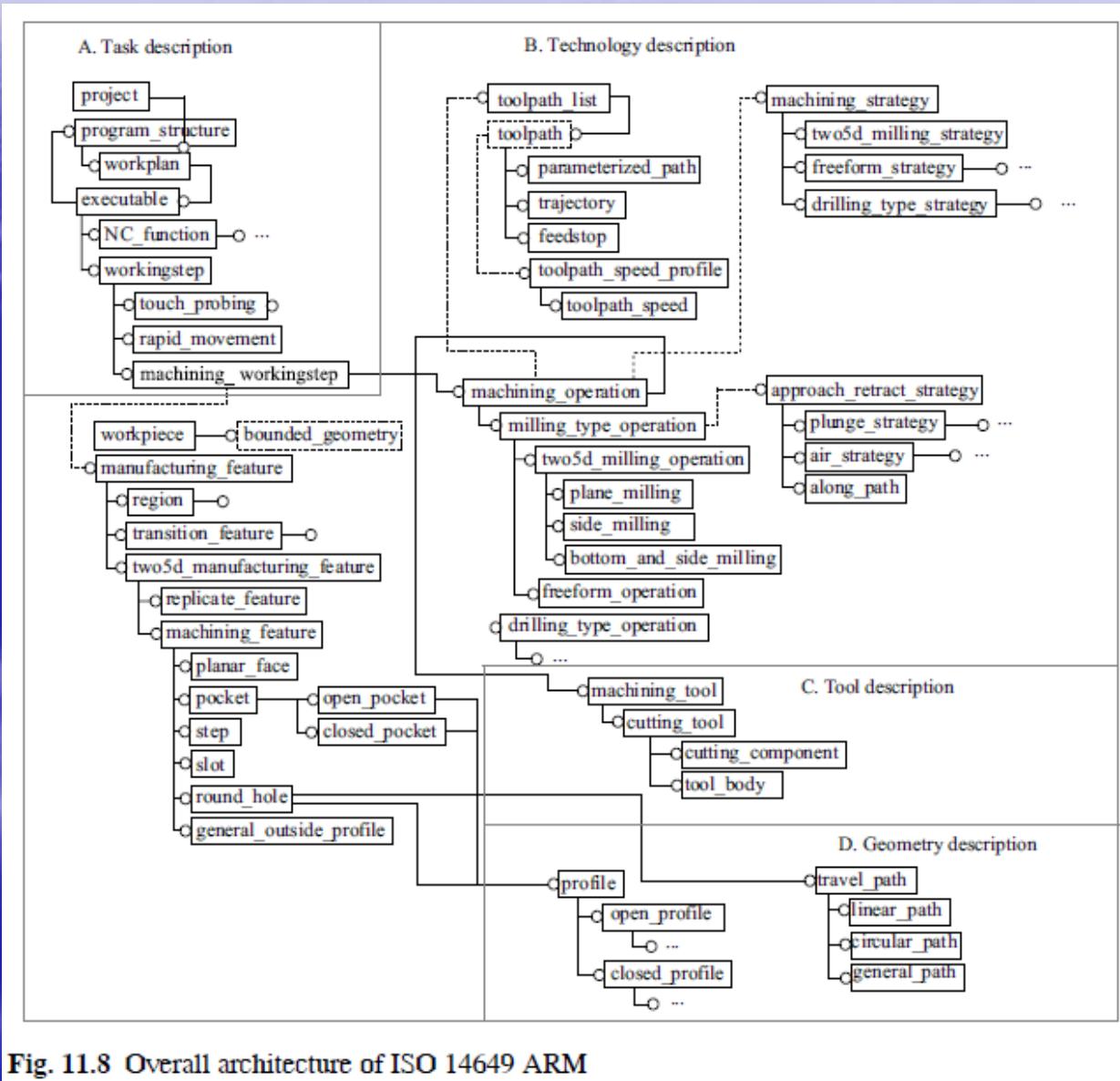
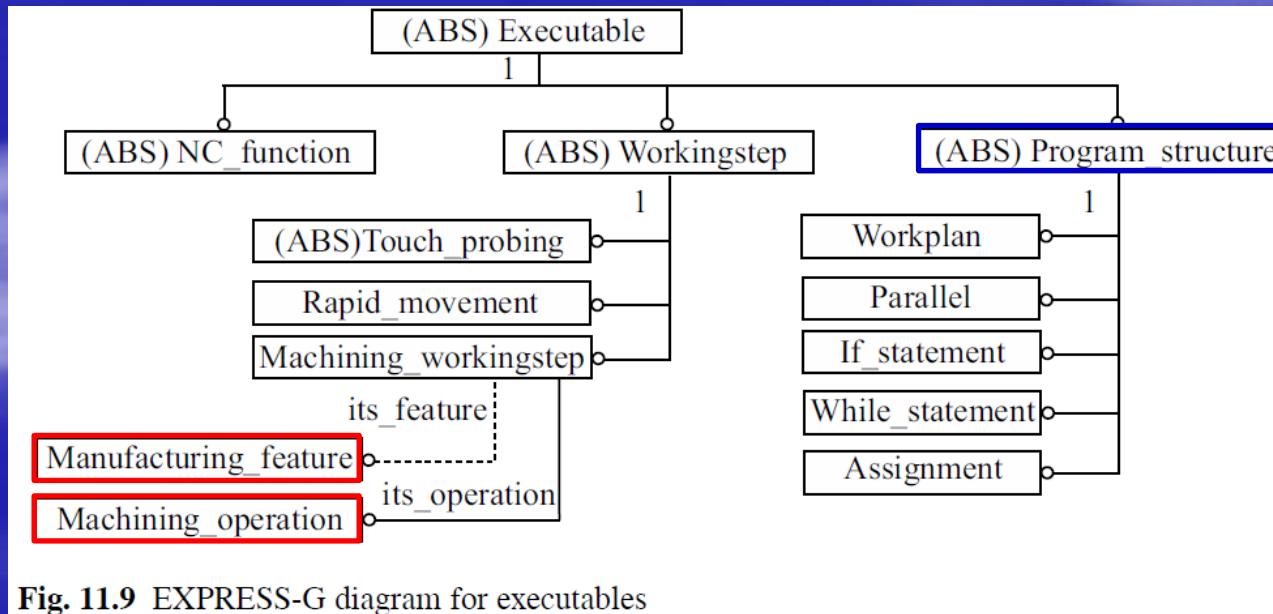


Fig. 11.8 Overall architecture of ISO 14649 ARM

# STEP-NC Data Model

- ISO 14649 Part 10: General Process Data
  - The execution sequence of the part program.
    - `program_structure`
  - The general process data for NC-programming.
    - `manufacturing_feature`
    - `machining_operation`



**EXPRESS-G** is a standard graphical notation for information models. (ISO 10303-11)

Fig. 11.9 EXPRESS-G diagram for executables

# STEP-NC Data Model

- ISO 14649 Part 10: General Process Data
  - NC-function
    - It describes manufacturing or handling operations that do not involve interpolation of axes.
  - Workingsteps
    - They describe manufacturing or handling operations that involve interpolating axes.
    - rapid\_movements
    - probing\_operations
    - machining\_workingsteps
  - Program\_structure
    - They are used to build logical blocks for structured programming of the manufacturing operation.

# STEP-NC Data Model

- ISO 14649 Part 10: General Process Data
  - **manufacturing\_feature** defines 2.5D and 3D machining features.
  - **machining\_operation** defines the types of the operations.
  - **machining\_strategy** defines various strategies, technology that defines feedrate and spindle.
  - **machine\_functions** define coolant, chip removal.
  - **tool\_path** defines pre-defined tool paths.

# STEP-NC Data Model

## ■ ISO 14649 Part 10: General Process Data

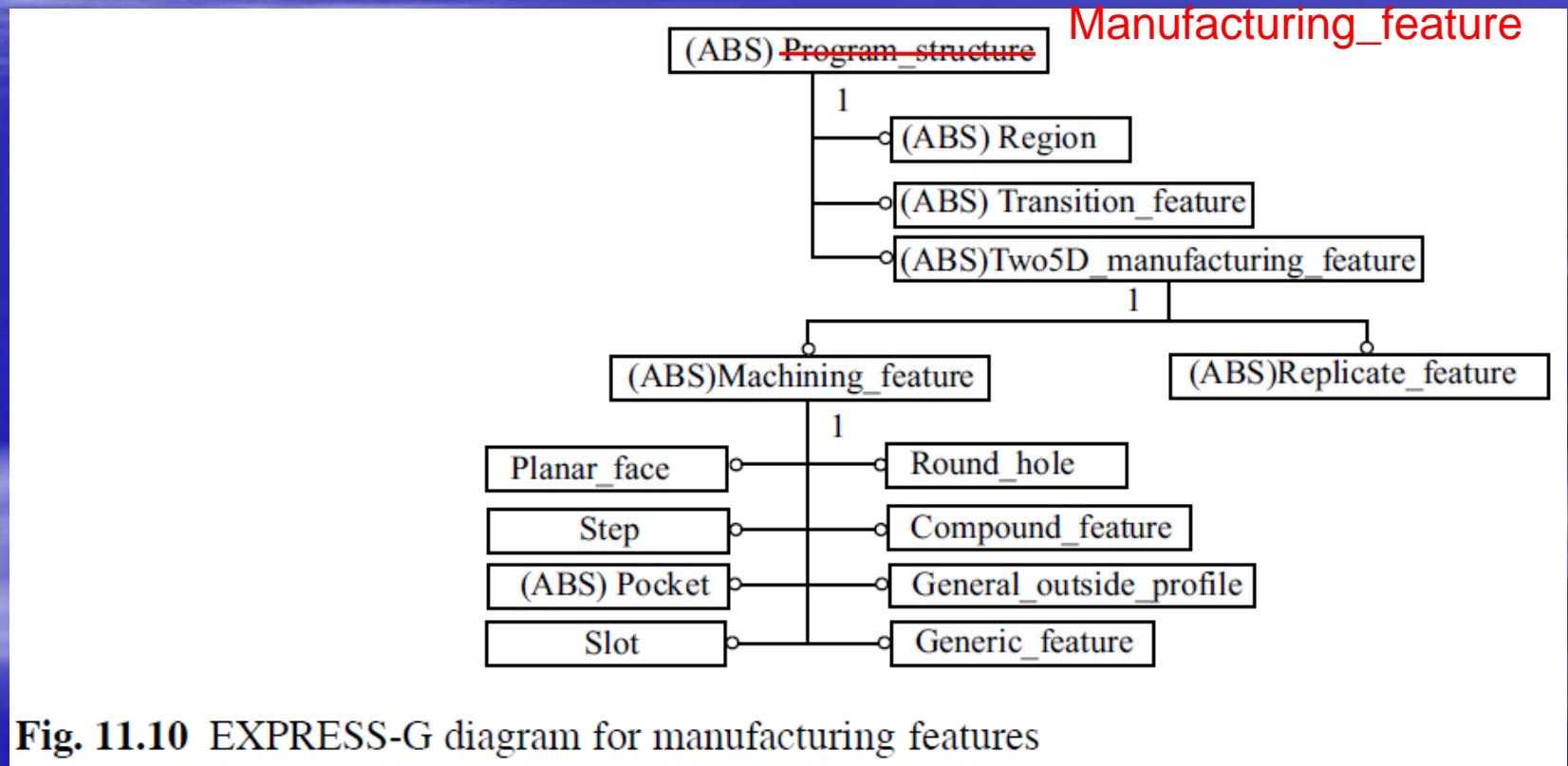


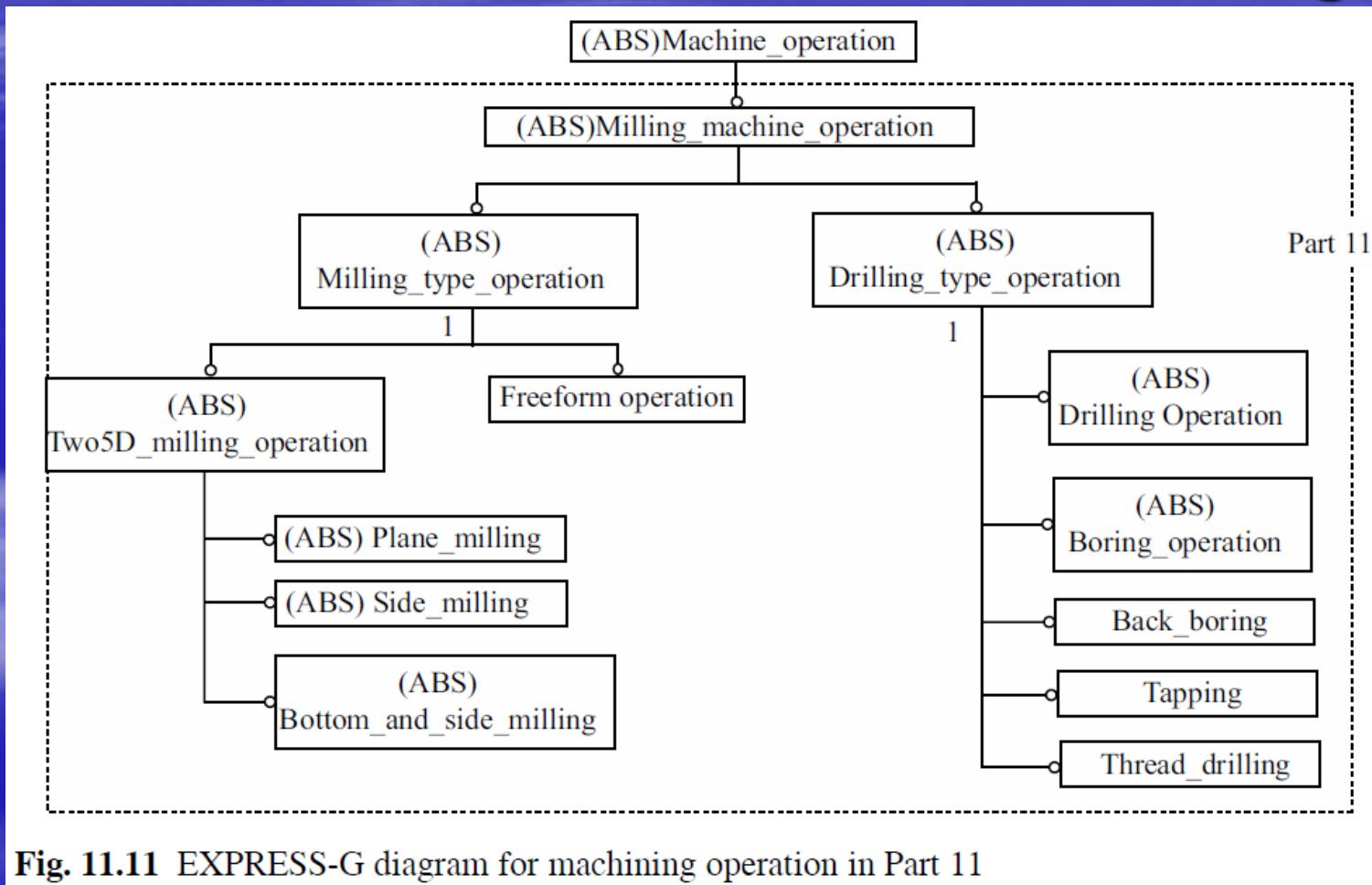
Fig. 11.10 EXPRESS-G diagram for manufacturing features

# STEP-NC Data Model

- ISO 14649 Part 11: Process Data for Milling
  - It specifies the technology-specific **data elements** needed as process data for **milling**.
  - It describes the technology-specific **data types** representing the machining process for **milling** and **drilling**.

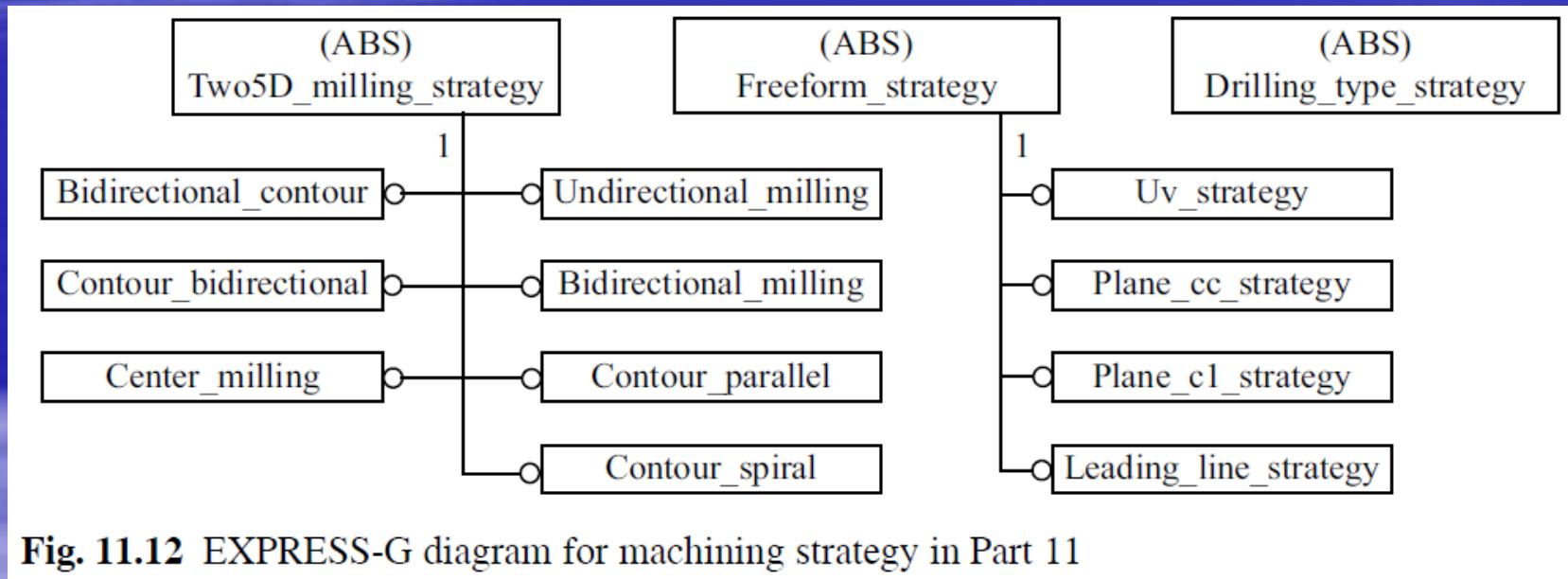
# STEP-NC Data Model

## ■ ISO 14649 Part 11: Process Data for Milling



# STEP-NC Data Model

## ISO 14649 Part 11: Process Data for Milling

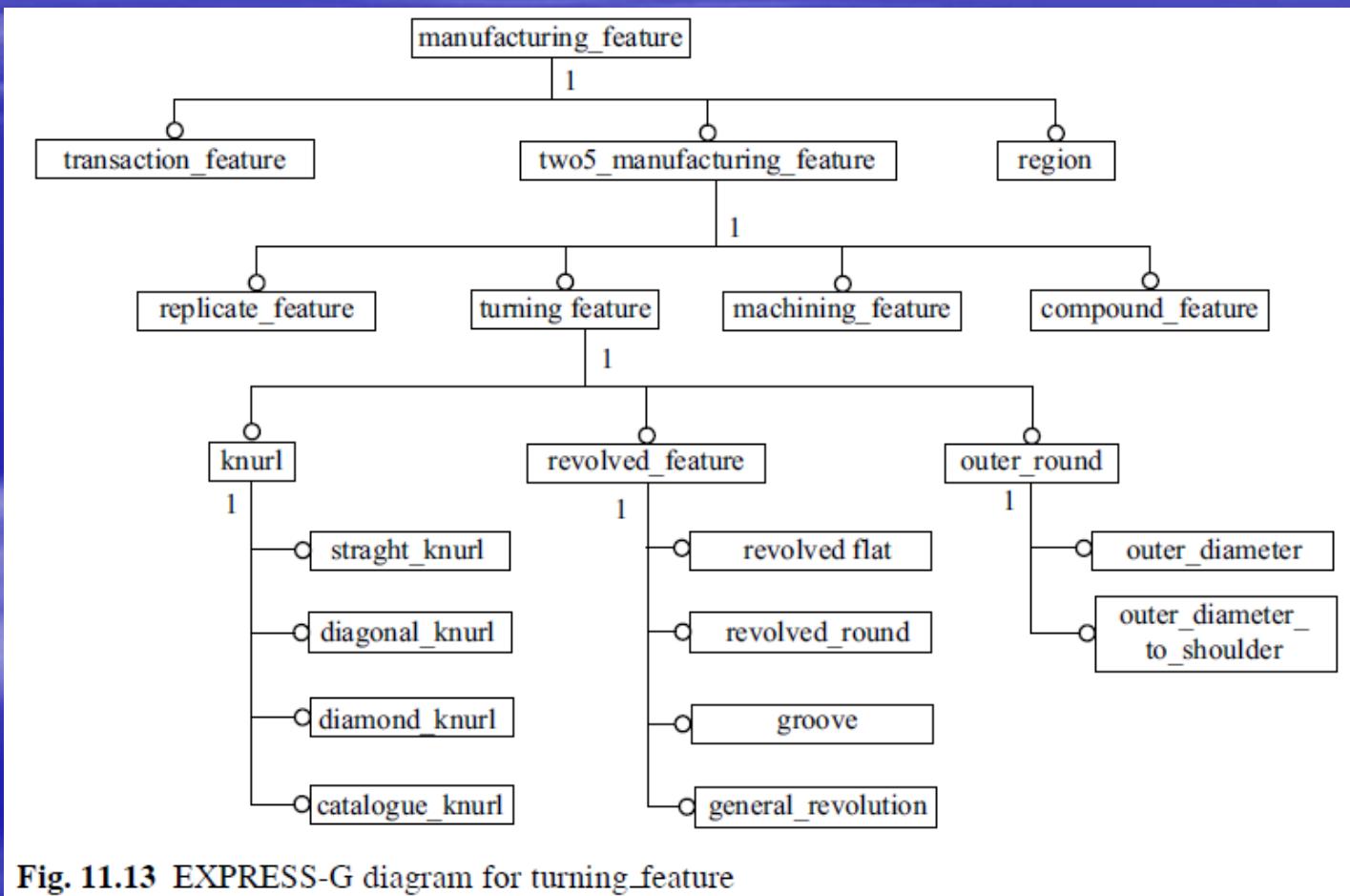


# STEP-NC Data Model

- ISO 14649 Part 12: Process Data for Turning
  - Part 12 defines the technology-specific **data elements** needed as process data for **turning**.

# STEP-NC Data Model

## ISO 14649 Part 12: Process Data for Turning



# STEP-NC Data Model

## ISO 14649 Part 12: Process Data for Turning

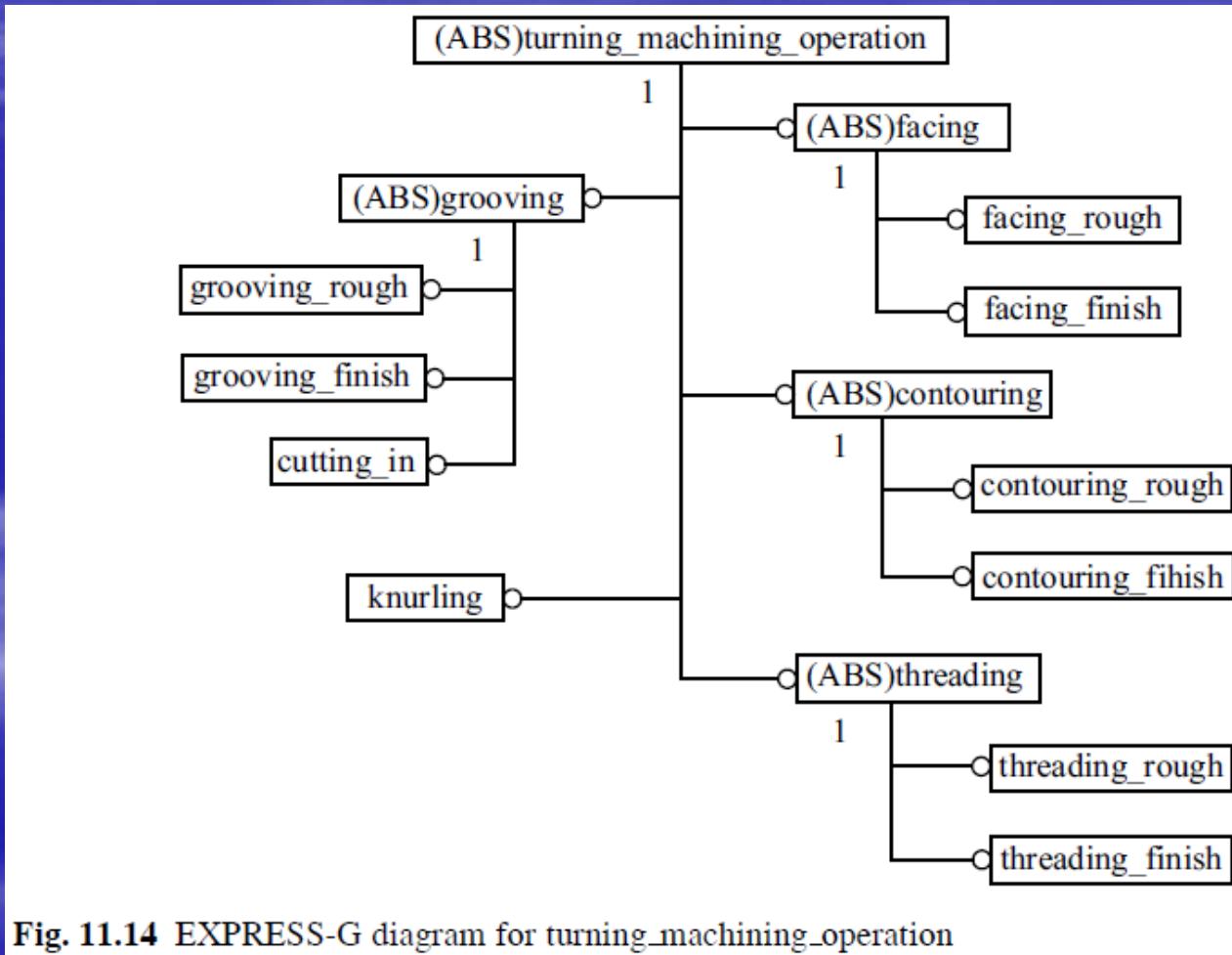


Fig. 11.14 EXPRESS-G diagram for turning\_machining\_operation

# STEP-NC Data Model

- ISO 14649 Part 111: Tools for milling machines
  - Part 111 defines data elements describing cutting tool data for milling machine tools

# STEP-NC Data Model

## ■ ISO 14649 Part 111: Tools for milling machines

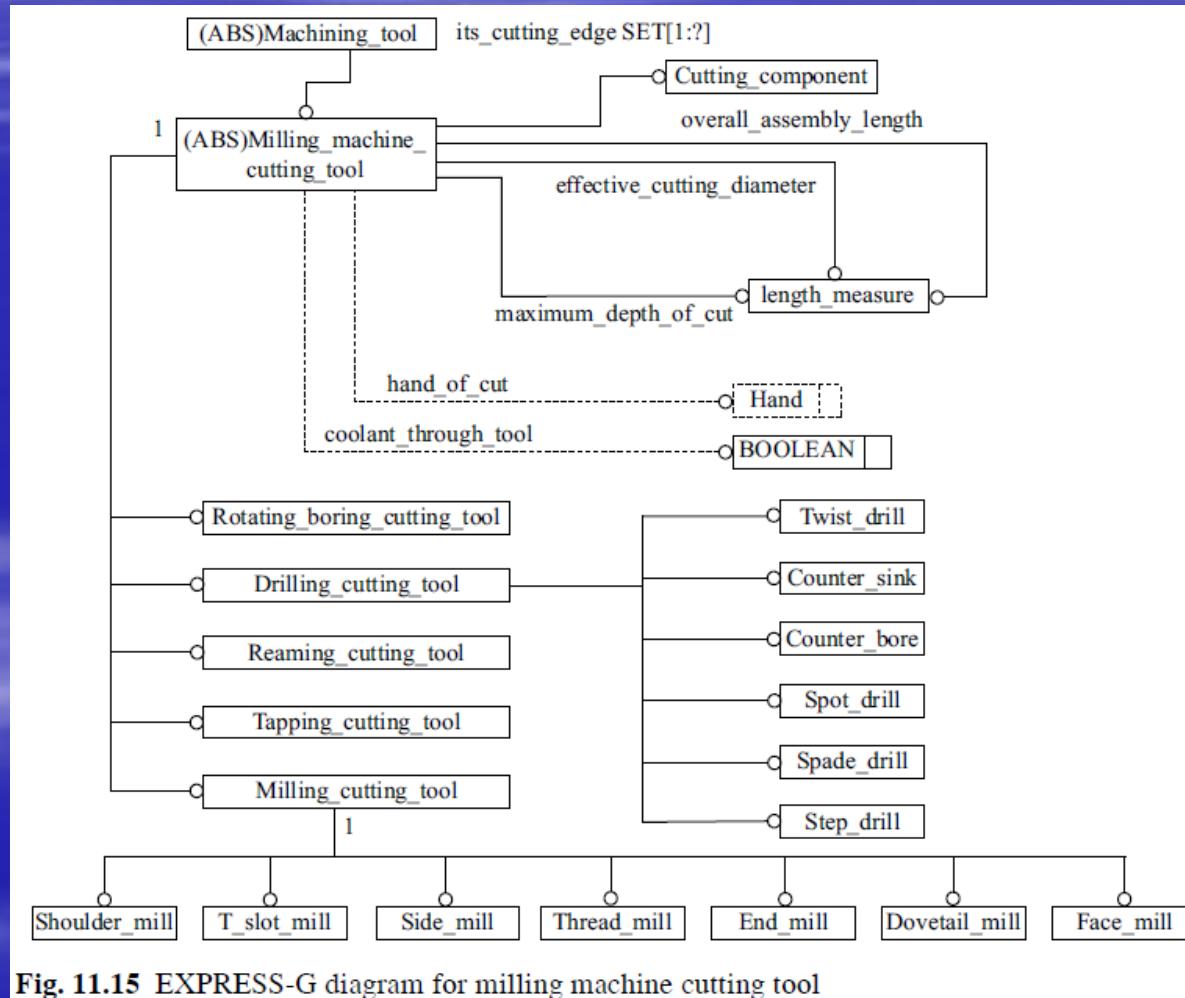


Fig. 11.15 EXPRESS-G diagram for milling machine cutting tool

# STEP-NC Data Model

- ISO 14649 Part 121: Tools for turning machines
  - Part 111 defines data elements describing cutting tool data for turning machine tools

# STEP-NC Data Model

## ■ ISO 14649 Part 121: Tools for turning machines

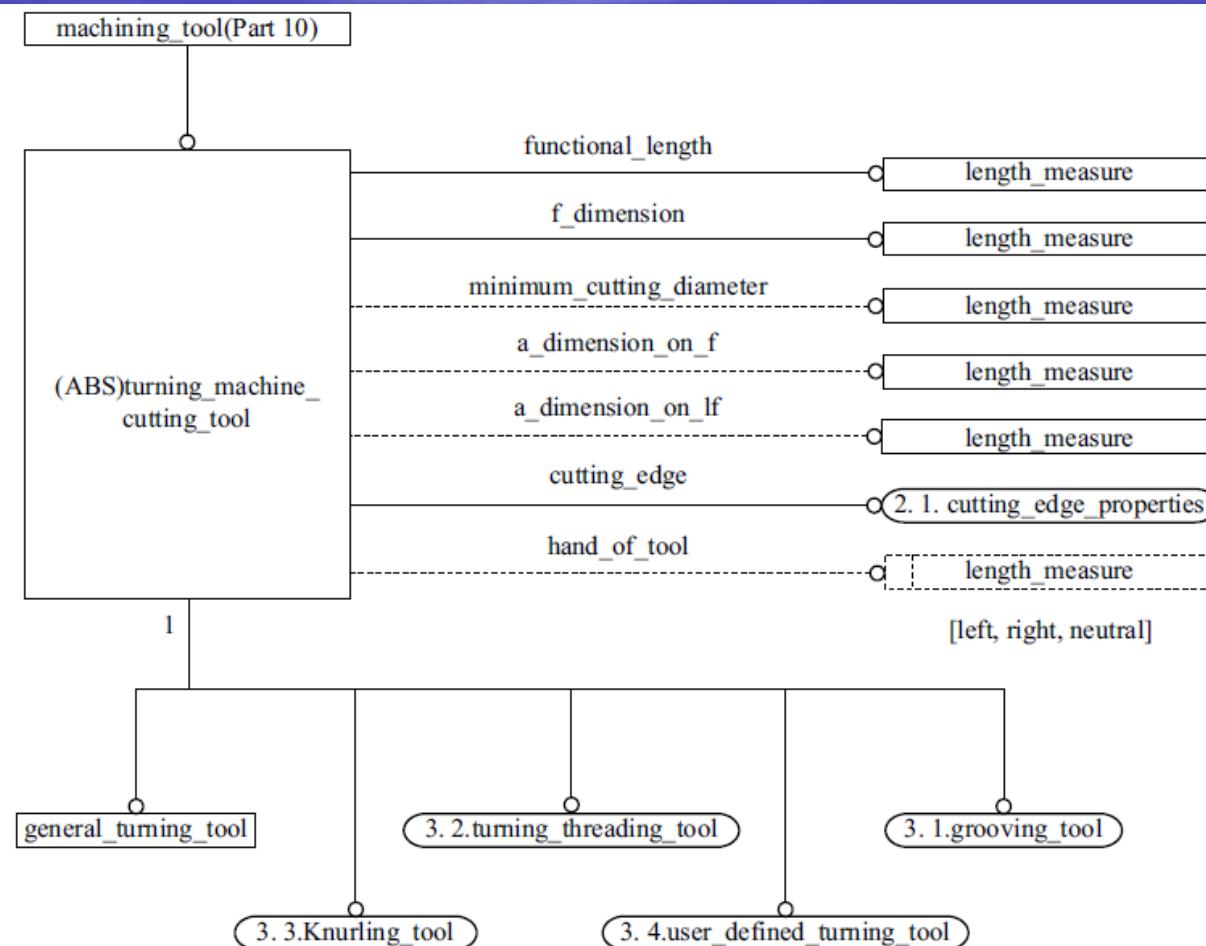


Fig. 11.16 EXPRESS-G diagram for turning machine cutting tool

# Part Programming

- **STEP-NC part program**
  - According to ISO 10303 Part 21: *Clear Text Encoding Rule*.
  - It is divided into the **header section** and the **data section**.
- **Header Section**
  - Author information.
  - Schema information.
  - Version .

# Part Programming

- **Data Section (about the manufacturing)**
  - Process sequence.
  - Manufacturing feature.
  - Operation type.
  - Machining strategy.
  - Machining technology.
  - Machine function.
  - Workpiece.
  - Geometry.

# Part Program for the Milling Operation

- The example for milling

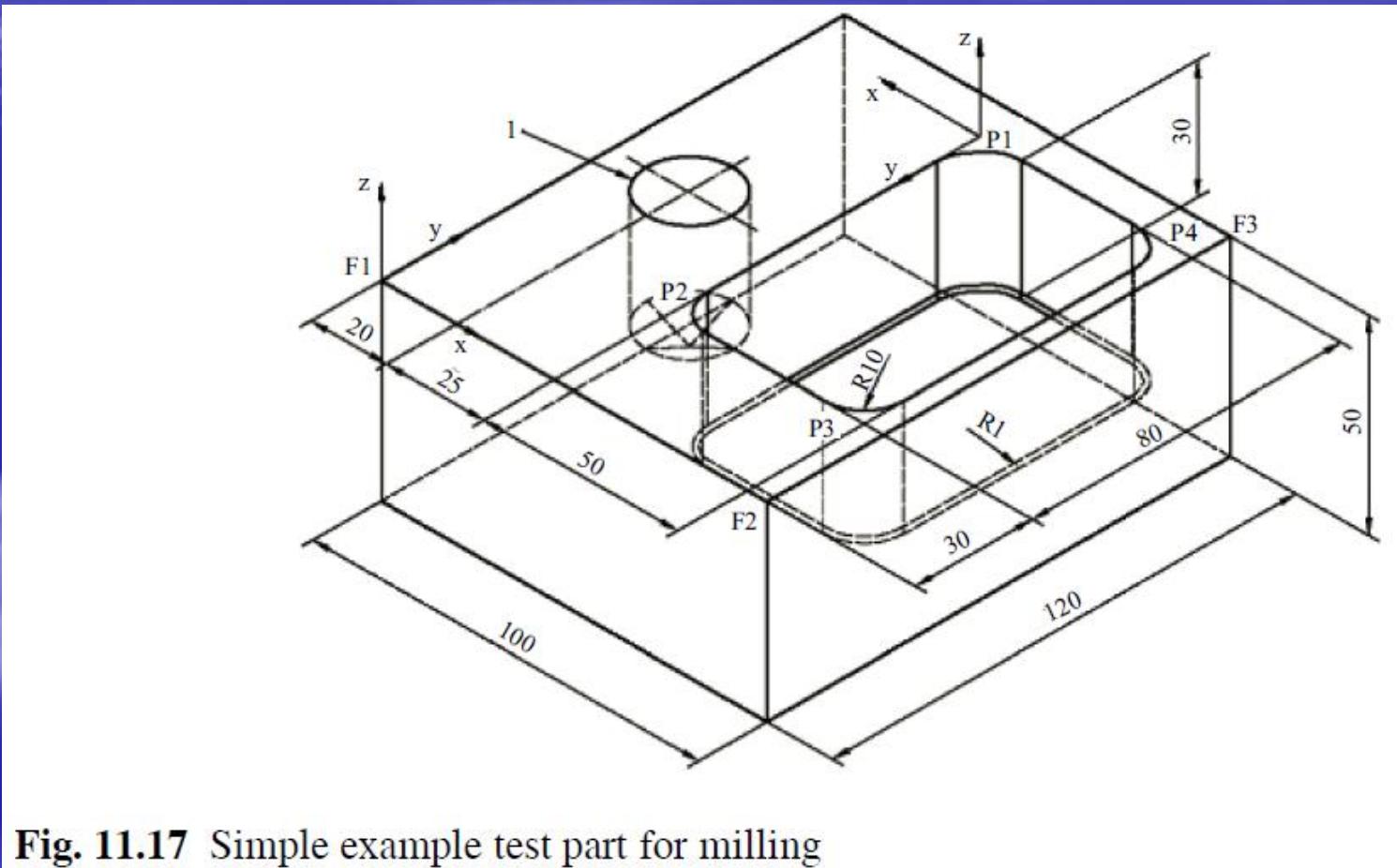


Fig. 11.17 Simple example test part for milling

# Part Program for the Milling Operation

- The part program of the example in Figure 11.17

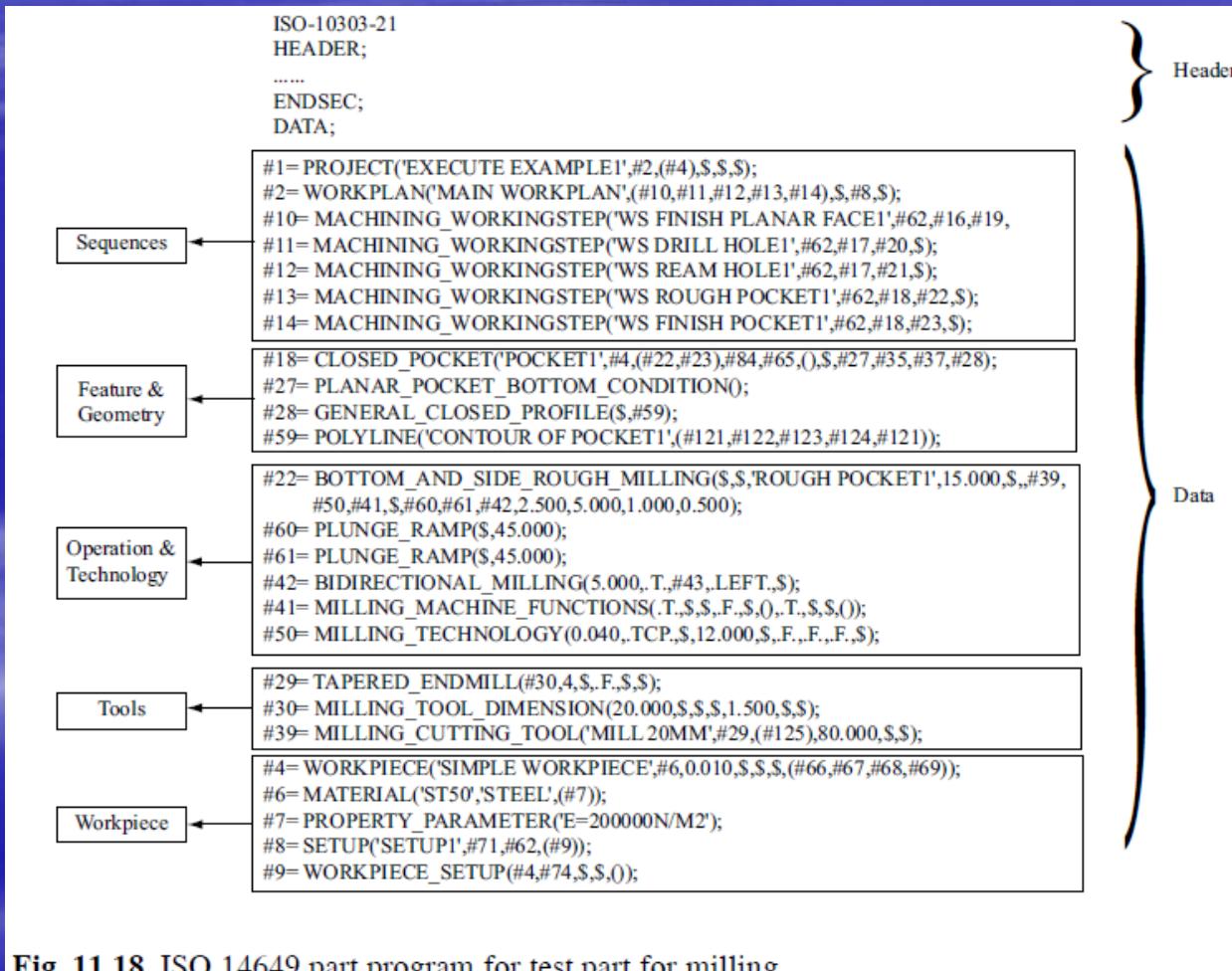


Fig. 11.18 ISO 14649 part program for test part for milling

# Part Program for the Milling Operation

## ■ Sequences

- The information about the machining sequence.
- #1 = PROJECT(... #2, (#4), ...)
  - Every STEP-NC part program **starts** with the PROJECT entity.
- #2 = WORKPLAN(..., (#10, #11, #12, #13, #14), ..., #8, ...)
  - It defines the **sequence** of machining processes.
- #10 = MACHINING\_WORKINGSTEP(..., #62, #16, #19, ...)
  - The **finishing** operation for the **planar\_face** at the top.
- #11 = MACHINING\_WORKINGSTEP (... , #62, #17, #20, ...)
  - The **drilling** operation.
- #12 = MACHINING\_WORKINGSTEP (... , #62, #17, #21, ...)
  - The **reaming** operation.

# Part Program for the Milling Operation

## ■ Sequences

- #13 = MACHINING\_WORKINGSTEP (... , #62, #18, #22, ...)
  - Roughing operation for closed\_pocket.
- #14 = MACHINING\_WORKINGSTEP (... , #62, #18, #23, ...)
  - Finishing operation for closed\_pocket.

# Part Program for the Milling Operation

## ■ Feature and geometry

- The feature information in the STEP-NC part program.
- #18 = CLOSED\_POCKET(..., #4, (#22, #23),  
#84, #65, ..., #27, #35, #37, #28)
- #27 =  
PLANAR\_POCKET\_BOTTOM\_CONDITION()
- #28 = GENERAL\_CLOSED\_PROFILE(..., #59)
- #59 = POLYLINE(..., (#121, #122, #123 , #124 ,  
#121))

# Part Program for the Milling Operation

## ■ Operation & Technology

- It defines the **method** to execute the given machining operation.
- #22 = BOTTOM\_AND\_SIDE\_ROUGH\_MILLING(..., #39, #50, #41, ..., #60, #61, #42, ...)
- #60 = PLUNGE\_RAMP(...)
- #61 = PLUNGE\_RAMP(...)
- #42 = BIDIRECTIONAL\_MILLING(..., #43, ...)
- #41 = MILLING\_MACHINE\_FUNCTIONS(...)
- #50 = MILLING\_TECHNOLOGY(...)

# Part Program for the Milling Operation

## ■ Tools

- It defines the **diameter**, **edge radius**, **overall length**, and **number of cutting teeth**.
- **#29 = TAPERED\_ENDMILL(#30, ...)**
- **#30 = MILLING\_TOOL\_DIMENSION(...)**
- **#39 = MILLING\_CUTTING\_TOOL(..., #29, (#125), ...)**

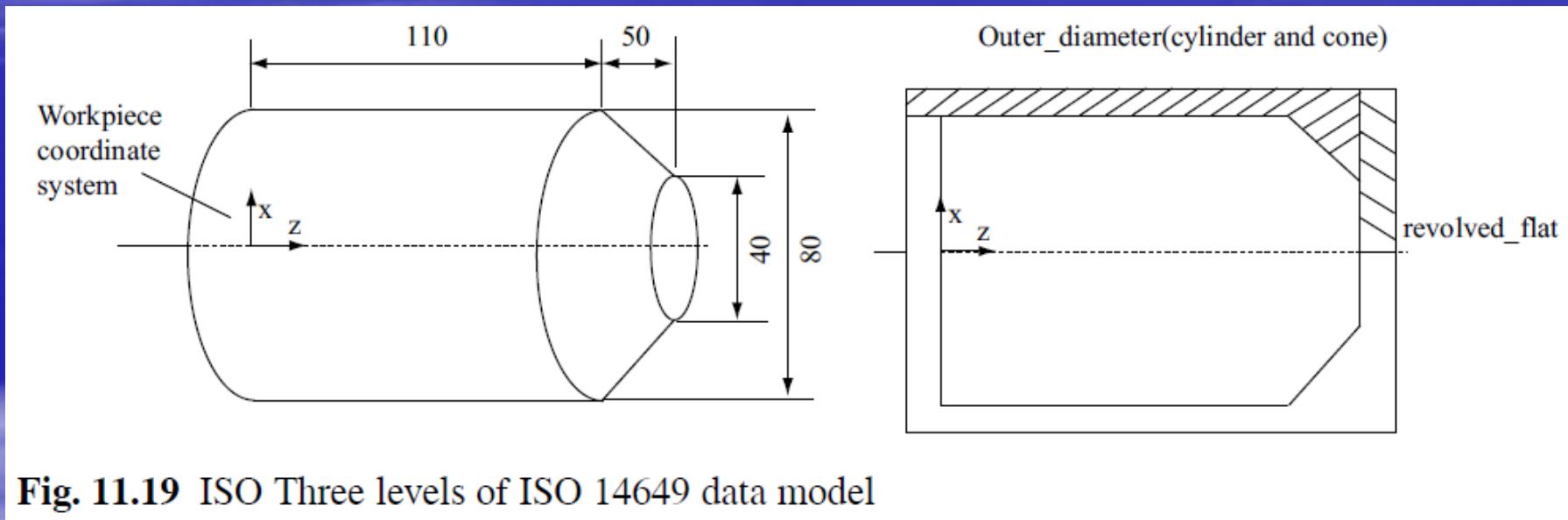
# Part Program for the Milling Operation

## ■ Workpiece

- It supports the **initial and final shape** of the raw workpiece, **material** of the workpiece, **chucking position** of the workpiece and so on.
- **#4 = WORKPIECE(..., #6, ..., (#66, #67, #68, #69))**
- **#6 = MATERIAL(..., (#7))**
- **#7 = PROPERTY\_PARAMETER(...)**
- **#8 = SETUP(..., #71, #62, (#9))**
- **#9 = WORKPIECE\_SETUP(#4, #74, ...)**

# Part Program for the Turning Operation

- The example for turning



# Part Program for the Turning Operation

- The part program of the example in Figure 11.19

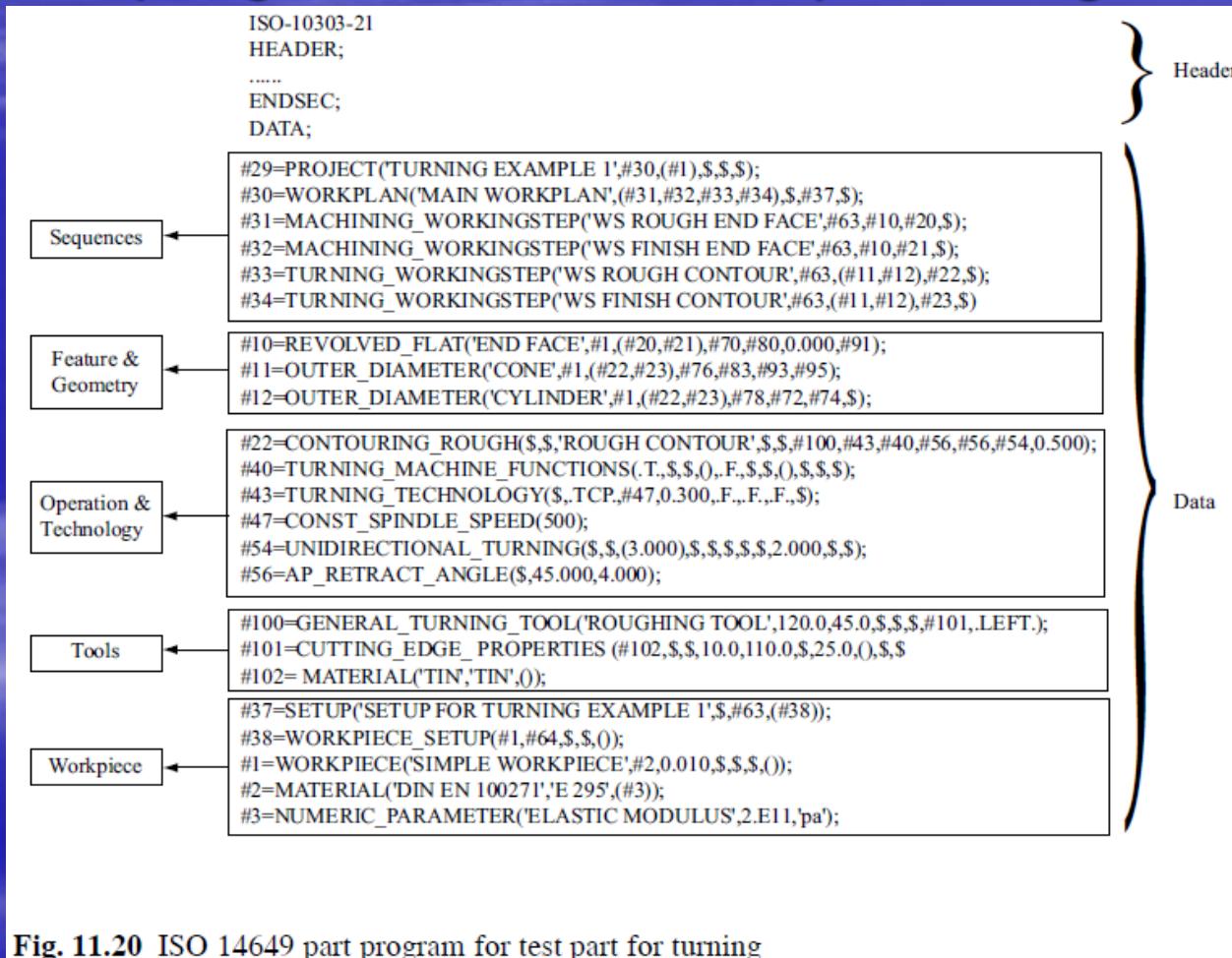
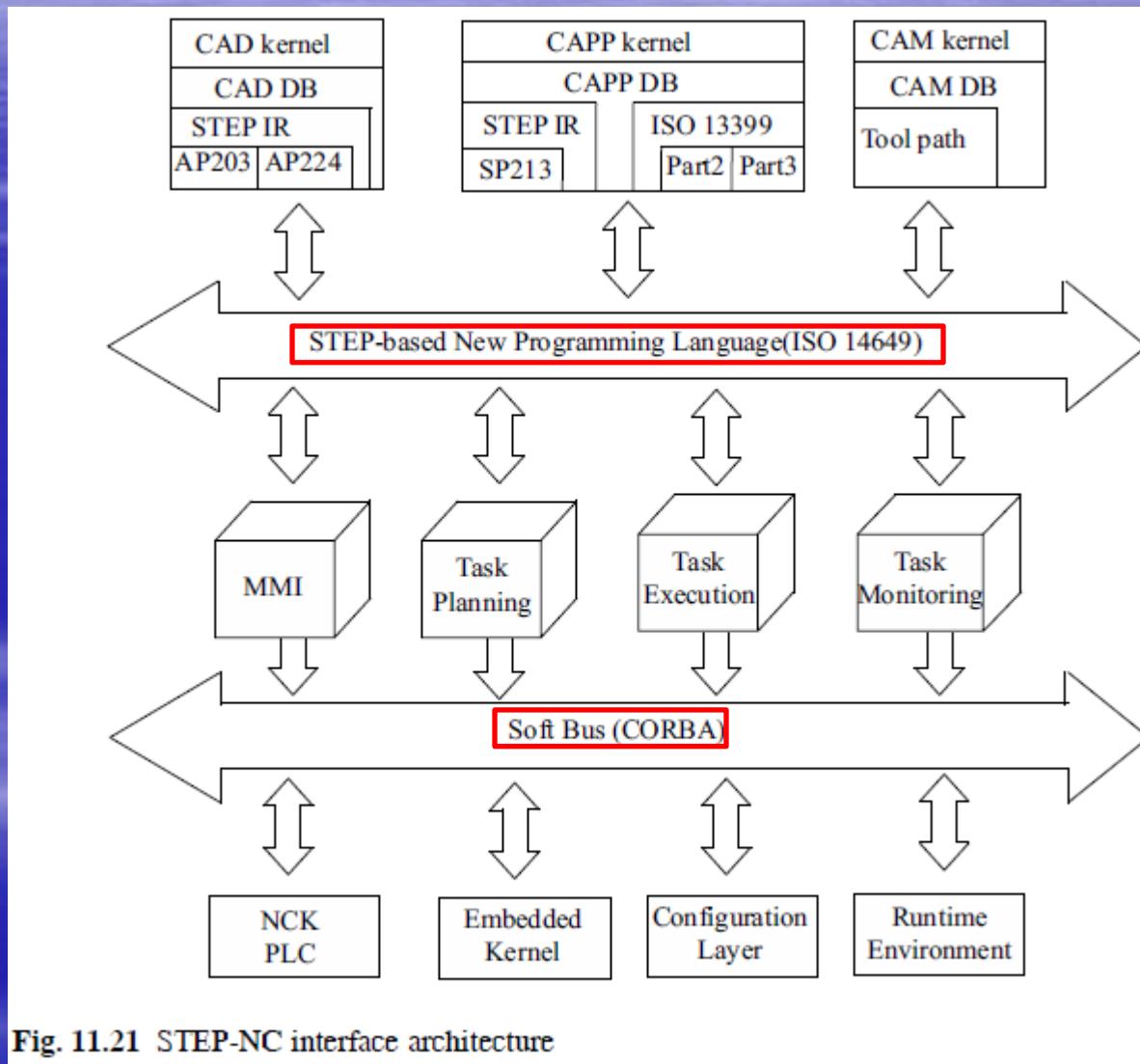


Fig. 11.20 ISO 14649 part program for test part for turning

# STEP-CNC System

- STEP-CNC has two types of interface bus.
  - **External Bus** (STEP based New Programming Language, **ISO 14649**)
    - It connects CNC and the CAD/CAPP/CAM system.
  - **Internal Bus** (**Soft Bus, CORBA**)
    - It makes it possible for the various intelligent modules on the inside of the CNC controller to communicate with each other.

# STEP-CNC System



# STEP-CNC System

- STEP-NC requires various technologies.

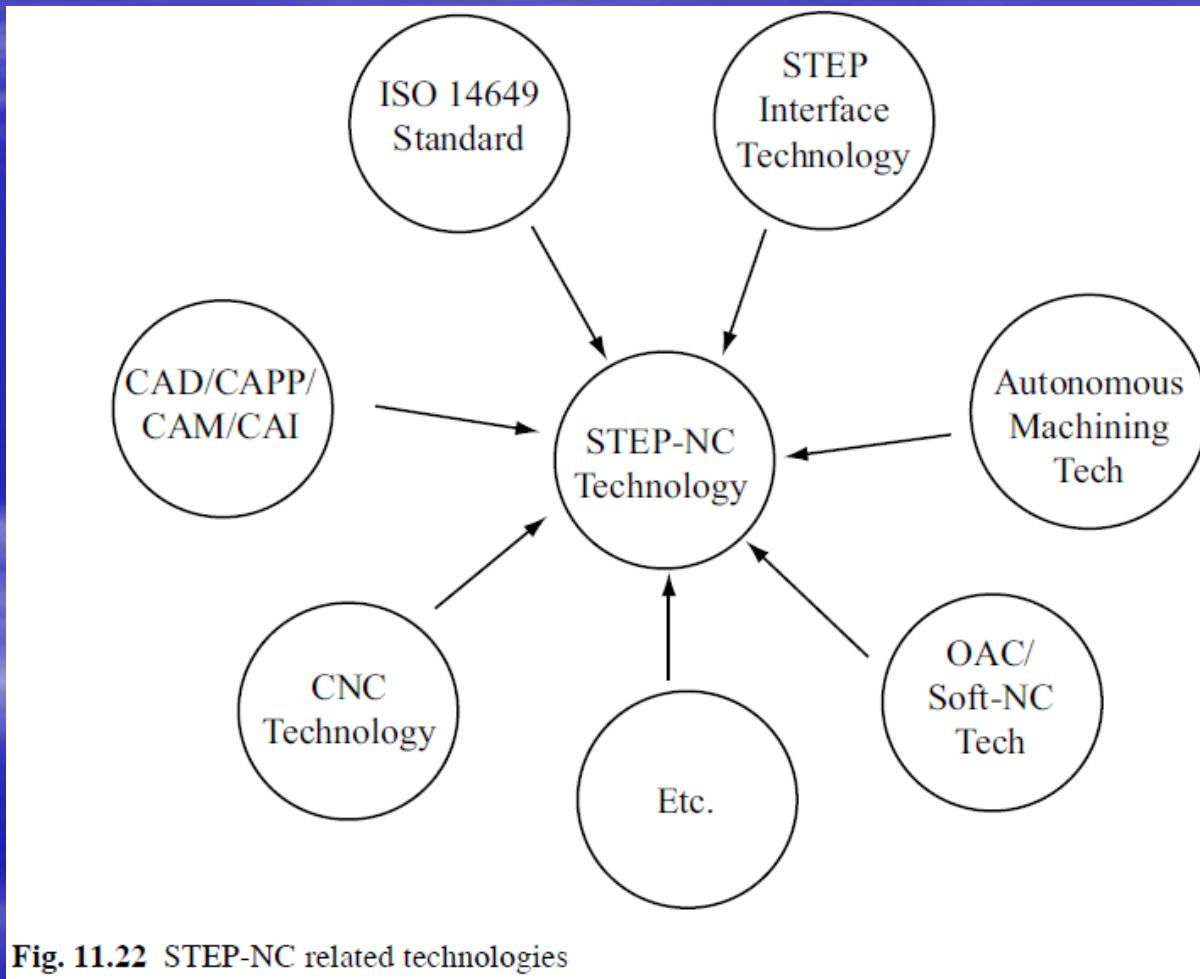


Fig. 11.22 STEP-NC related technologies

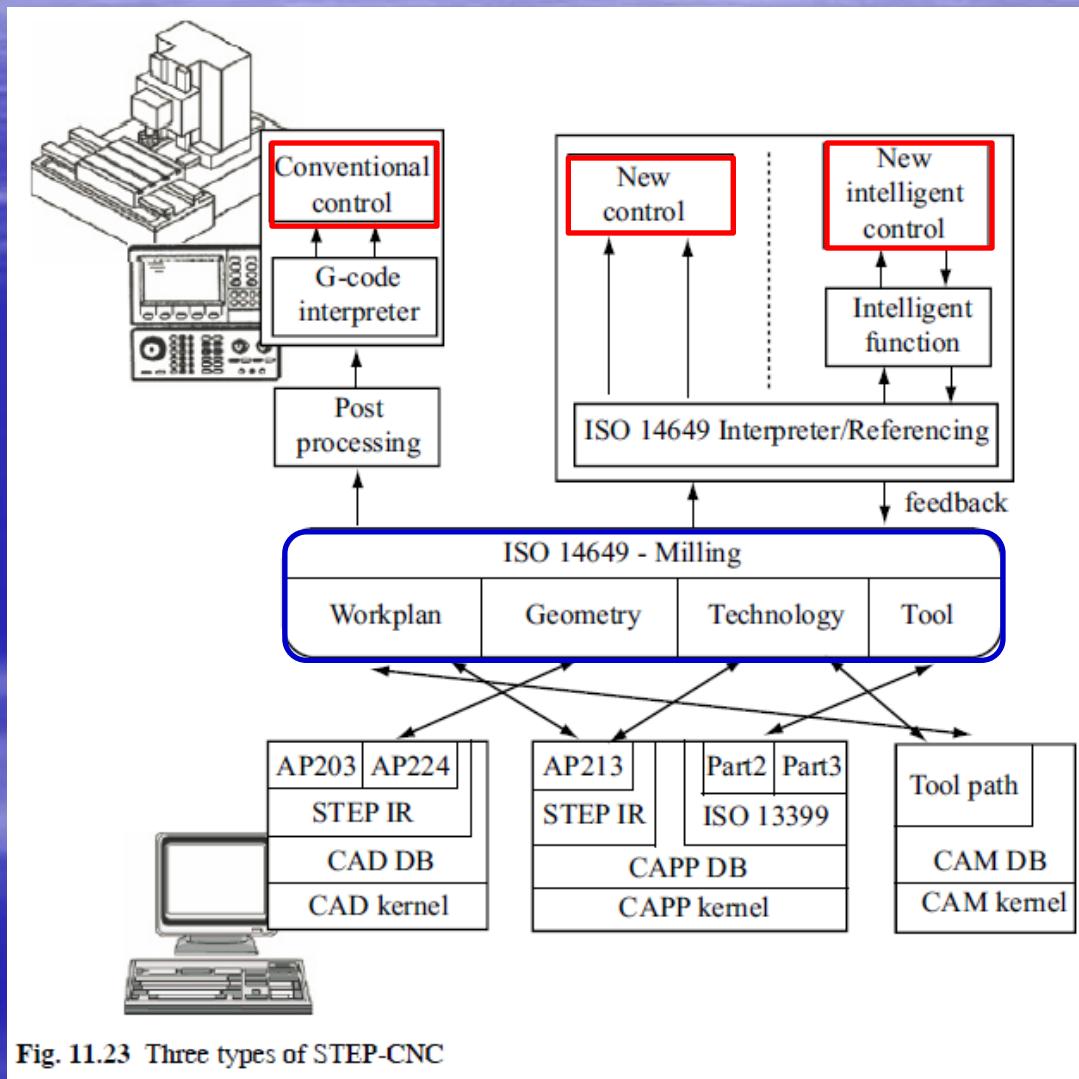
# Types of STEP-CNC

- STEP-compliant CNC is classified into 3 types.
  - Conventional control.
    - It **simply incorporates ISO 14649** in a conventional controller via post-processing.
    - Conventional CNC can be used **without modification**.
    - Strictly speaking, this cannot be considered as a STEP-compliant CNC.
    - It should at least be able to read ISO 14649 code.
  - New control.
    - It has a **STEP-NC interpreter**.
    - CNC kernel with built-in **toolpath generation capability**.
    - The motion is executed based on the **machining strategy and sequence** as specified by the ISO 14649 part program.

# Types of STEP-CNC

- STEP-compliant CNC is classified into 3 types.
  - New intelligent control.
    - It much **more promising** than the predecessors.
    - CNC is able to perform machining task '**intelligently**' and '**autonomously**'.
    - Automatic feature **recognition**.
    - Automatic **collision-free** toolpath generation.
    - Automatic **tool** selection.
    - Automatic **cutting condition** selection.
    - Status **monitoring**.
    - Automatic **recovery**.
    - Machining status and result **feedback**.

# Types of STEP-CNC



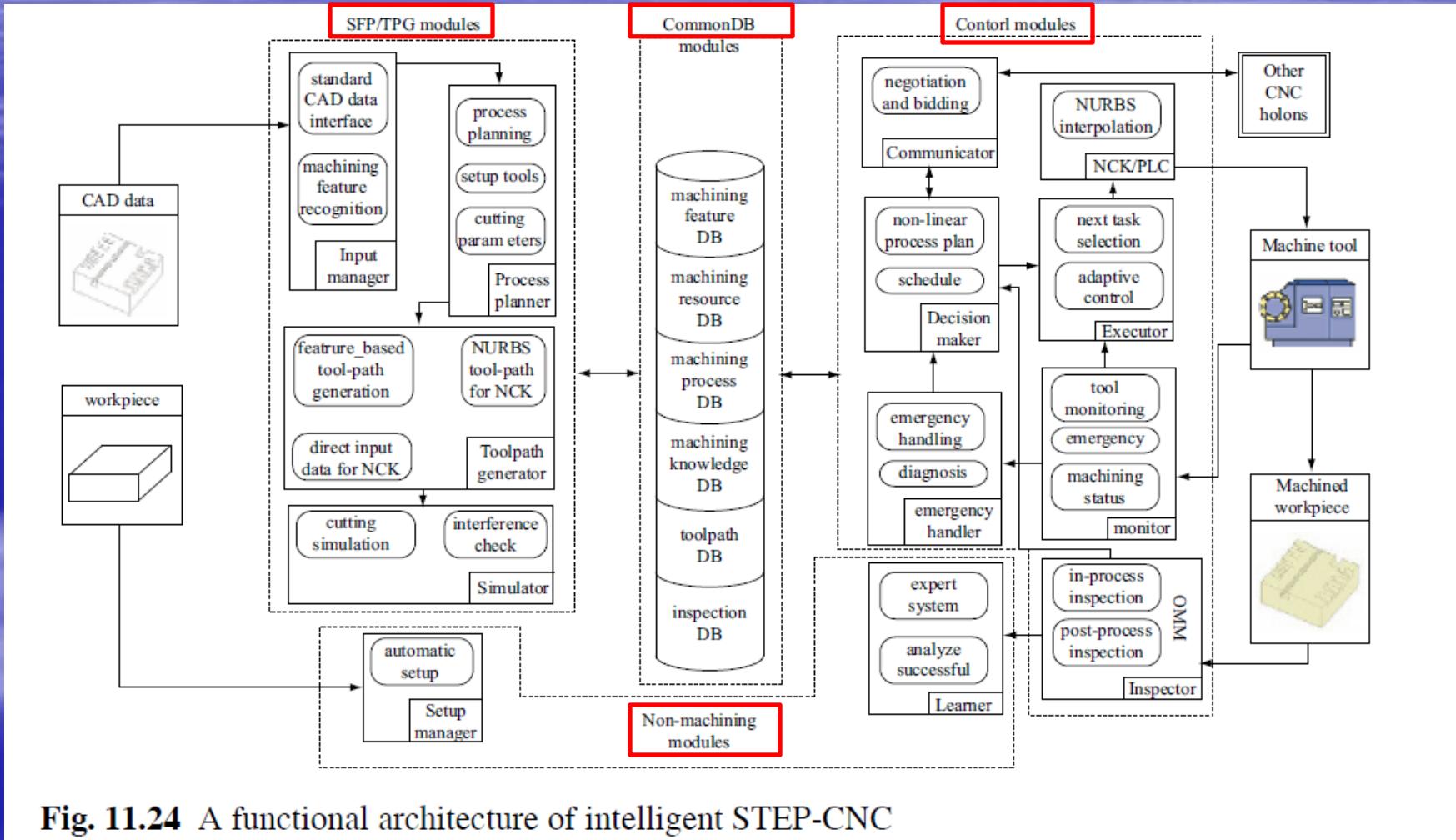
# Intelligent STEP-CNC Systems

- The requirements for the next-generation CNC.
  - The **data-level** point of view.
    - CAD data interface with a standard schema.
    - Internet interface.
    - Seamless information exchange.
  - The **functional-level** point of view.
    - Autonomy.
    - Multi-functionality.
    - Change/failure recovery.
    - High speed machining.
    - Learning.

# Intelligent STEP-CNC Systems

- The requirements for the next-generation CNC.
  - The **implementation-level** point of view.
    - Software-based CNC.
    - Open and modular architecture.
    - User configurable structure.

# Intelligent STEP-CNC Systems



# Intelligent STEP-CNC Systems (Control Modules)

- Decision Maker
  - This **schedules** the task, selecting the next task from various alternative from a non-linear process plan.
  - One of the critical decisions is to assign the **priorities** between the scheduled task and the newly invoked task by the emergency handler and inspector.
- Executor
  - This **converts** the task into commands and **passes** them to NCK/PLC.
  - If the task is machining operation, it retrieves the corresponding toolpath from Tool-Path DB and passes it to NCK/PLC.
  - If the task is a tool change, it finds the tool in the tool magazine and passes it to NCK/PLC.
  - Executor keeps **track of the commands executed by NCK/PLC** for adaptive control.

# Intelligent STEP-CNC Systems

(Control Modules)

- NCK/PLC
  - NCK interprets the **toolpath commands** and **executes them** by activating the servo mechanism.
  - PLC executes machinery commands, such as **tool change** and **workpiece loading/unloading**.
  - For free-form surface machining, NCK is capable of **NURBS interpolation** in which accurate and high-speed machining can be carried out with reduced data.
- Monitor
  - The entire machining status is **continuously monitored** by capturing information from sensor signals.
  - **Tool monitoring** and **emergency detection** are crucial.
  - The results are sent to the emergency handler and/or the decision maker accordingly.

# Intelligent STEP-CNC Systems (Control Modules)

- Emergency Handler
  - In case of an emergency, which is monitored and reported by the monitor, the emergency handler makes a **diagnosis** and **decides what to do** about it.
  - The result is **sent to the decision maker** for the final decision and scheduling.
  - For example, in the case of tool breakage, the emergency handler retracts the tool, and checks if an **alternative** tool is available in the tool magazine (through Machine Resource DB).
  - If one is available the operation is resumed with the alternative tool, otherwise it report to the decision maker and waits for a final decision.

# Intelligent STEP-CNC Systems (Control Modules)

- Inspector
  - In-process and post-process inspections are carried out automatically by the inspector.
  - In either case, inspector generates the toolpath for the touch probe and stores the data into the Inspection DB.
  - Any geometrical errors between the designed part and the machined part are found by comparing the data of the inspection DB with that of the Machining Feature DB.

# Intelligent STEP-CNC Systems (Control Modules)

- Communicator
  - The communicator is responsible for the **interactions** with external unit, such as **CAD/CAM system**, **shopfloor control system**, and **human operator**.

# Intelligent STEP-CNC Systems (SFP/TPG Modules)

- Input Manager
  - The roles of the input manager are **CAD data interface handling** and **machining feature recognition**.
  - It **translates** standard CAD data (STEP, AP203) into built-in geometric modeling kernel data, **recognizes** the machining features, and **extract** the feature attributes required for machining.
  - Output is stored in the **Machining Feature DB**.

# Intelligent STEP-CNC Systems (SFP/TPG Modules)

- Process Planner

- This determines the **processing sequence, operations, fixtures, setups** and **cutting tools** required to machine the feature.
  - The processing sequence is represented by a **non-linear process plan** so that the **decision maker** can select an appropriate plan at the time of execution.
  - **Optimal cutting parameters, machining strategies** and **tools for operations** are determined using the **Machining Knowledge DB**.
  - For this, a **knowledge-based** process planning system is required.
  - Output is stored in the **Machining Process DB**.

# Intelligent STEP-CNC Systems (SFP/TPG Modules)

- Tool-Path Generator
  - This generates toolpaths both for **machining** and **measurement**.
  - It can generate a complete path include **approach**, **departure**, and **connection** path between the machining or measurement paths.
  - The generated toolpaths are stored in the **Tool-Path DB**, which is accessed by NCK/PLC.
  - As NCK/PLC is able to **interpret NURBUS** curve directly, the toolpath generator does not segment the tool path of a freedom curve into line/arc.

# Intelligent STEP-CNC Systems (SFP/TPG Modules)

- Simulator
  - Prior to actual machining, it is necessary to perform a cutting simulation to **verify** the given tool path and to detect any possible errors.
  - This simulator finds undercut or gouging and tool interference by cutting simulation.
  - In addition to error detection in the **toolpath**, **optimal feedrate** is calculated by using the required material removal rate during the solid cutting simulation.
  - Output is stored in **the Tool-Path DB** and the **Machining Process DB**.

# Intelligent STEP-CNC Systems

(Non-Machining Modules)

- Setup Manager
  - This supports the **part setup** operation.
  - Once the part is loaded onto the machine, it finds the datum position **by moving a touch probe** using the workpiece and fixture **geometry information**.
- Learner
  - Information captured during machining is analyzed by an expert algorithm, and stored in the **Machining Knowledge information**.

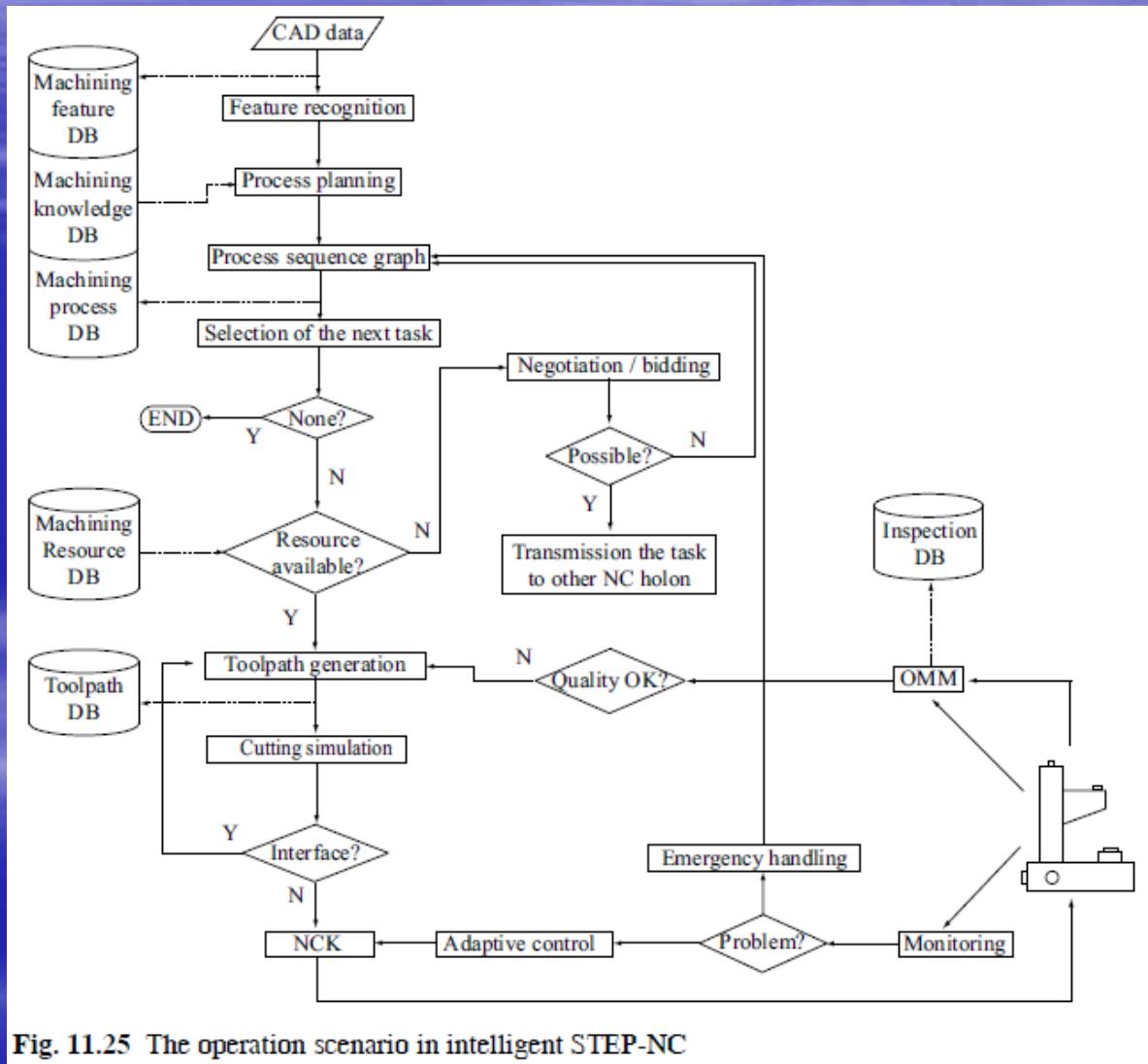
# Intelligent STEP-CNC Systems

(Common DB Modules)

- Common DB modules

- These DB modules are the **repositories of data** that are generated, updated, and retrieved by **control modules** and **SFP/TPG modules**.
  - **Short-Term Database**
    - Machining Feature DB
    - Machining Process DB
    - Toolpath DB
    - Inspection DB
  - **Long-Term Database**
    - Machine Resource DB
    - Machining Knowledge DB
  - On **completion** of the part machining, the **short-term database is cleared**.

# Intelligent STEP-CNC Systems



# Worldwide Research and Development

亞琛工業大學 (德國)  
工具機暨製造工程研究實驗室

## ■ WZL-Aachen University (Germany)

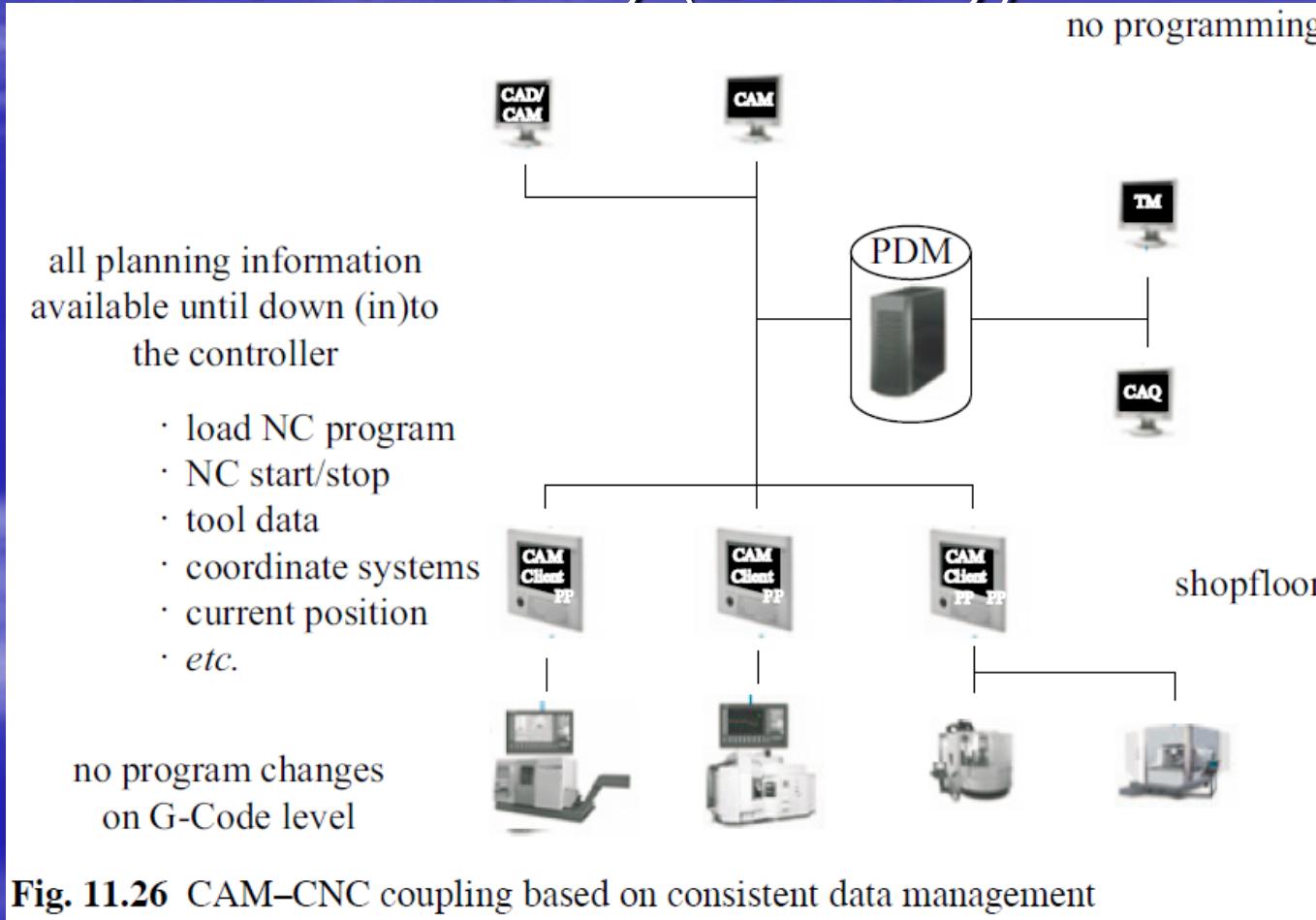
- Research at WZL has focused on optimizing manufacturing planning by **close coupling** of a **CAM System** and **CNC Controller**.
- Since the main requirement is to **assure the usability of existing machine tools and controllers**, a **post-processor** is still necessary to translate the process information into the data format of the specific CNC.
- Such a CAM client system might take the form of an integration framework that allows **integrating software solutions of different providers**.

# Worldwide Research and Development

亞琛工業大學 (德國)

工具機暨製造工程研究實驗室

## ■ WZL-Aachen University (Germany)



# Worldwide Research and Development

斯圖加特大學 (德國)

機床與製造單元控制工程研究所

- ISW-University of Stuttgart (Germany)
  - The work focuses on **methodologies**, **data models** and **software tools** to utilize bidirectional information exchange between **CNC** and a **unified manufacturing process planning database** capturing STEP-NC information.
  - ISW developed a **STEP-NC data model for turning**.
  - To **verify** the turning data model, ISW developed a Computer-Aided Planning demonstrator for turning, "**STEPturn**", and a software module to convert STEP-NC data into the **Siemens ShopTurn CNC data format**.
  - Relating this information about executed machining workingsteps to the corresponding manufacturing features and machining operations as well as additional context information, like the executing machine tool, **helps to build a comprehensive manufacturing knowledge database**.

# Worldwide Research and Development

斯圖加特大學 (德國)

機床與製造單元控制工程研究所

- ISW-University of Stuttgart (Germany)

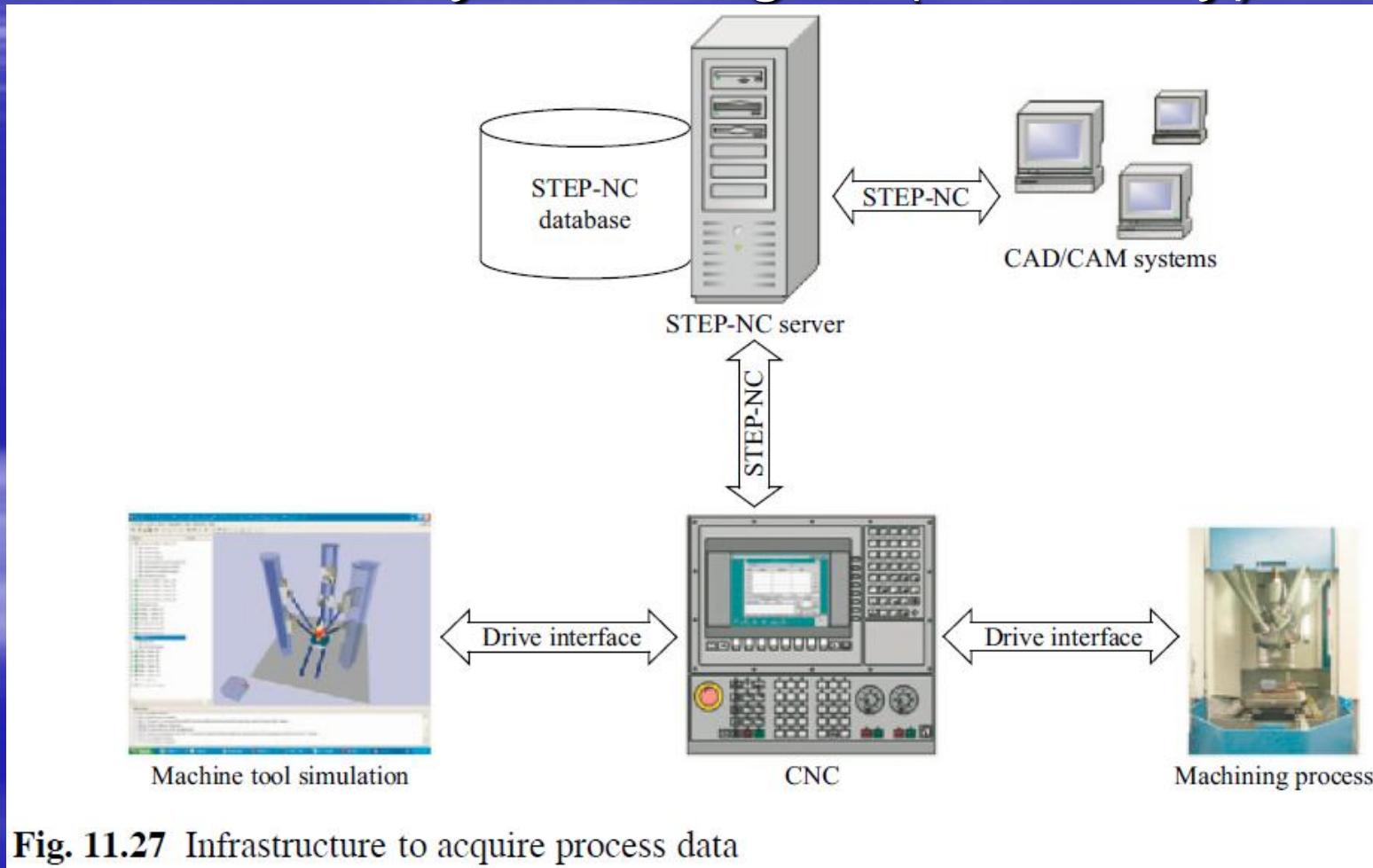


Fig. 11.27 Infrastructure to acquire process data

# Worldwide Research and Development

浦項工科大學 (南韓)

## ■ POSTECH (South Korea)

- The National Research Laboratory for STEP-NC Technology at POSTECH has made the following achievements related to STEP-NC technology:
  - Development of Korea STEP-NC: **STEP-CNC system for milling**
  - Development of TurnSTEP: **STEP-CNC system for turning**
  - Development of the **data model for turning** (ISO 14649 Part 12 and 121) with ISW-University of Stuttgart
  - Suggestion and reflection on **revision the ISO 14649 data model for milling**
  - Promotion of international and domestic **seminars for STEP-NC**

# Worldwide Research and Development

- POSTECH (South Korea) 浦項工科大學 (南韓)
    - The following issues have been **considered** in designing the architecture of STEP-CNC and are also **technical contributions** for implementation of STEP-NC.
      - Full compliance with ISO14649 and STEP APs
      - Suite of STEP-manufacturing
      - Distributed architecture for e-manufacturing
      - Extension to intelligent/autonomous CNC execution
      - Feature recognition/mapping capability
      - Tolerance handling capability
      - Optimization of the machining sequence for the CNC controller
      - Internet interfacing
      - XML support
      - Accommodation of conventional CNC
      - Automated/interactive generation capability

# Worldwide Research and Development

浦項工科大學 (南韓)

## ■ POSTECH (South Korea)

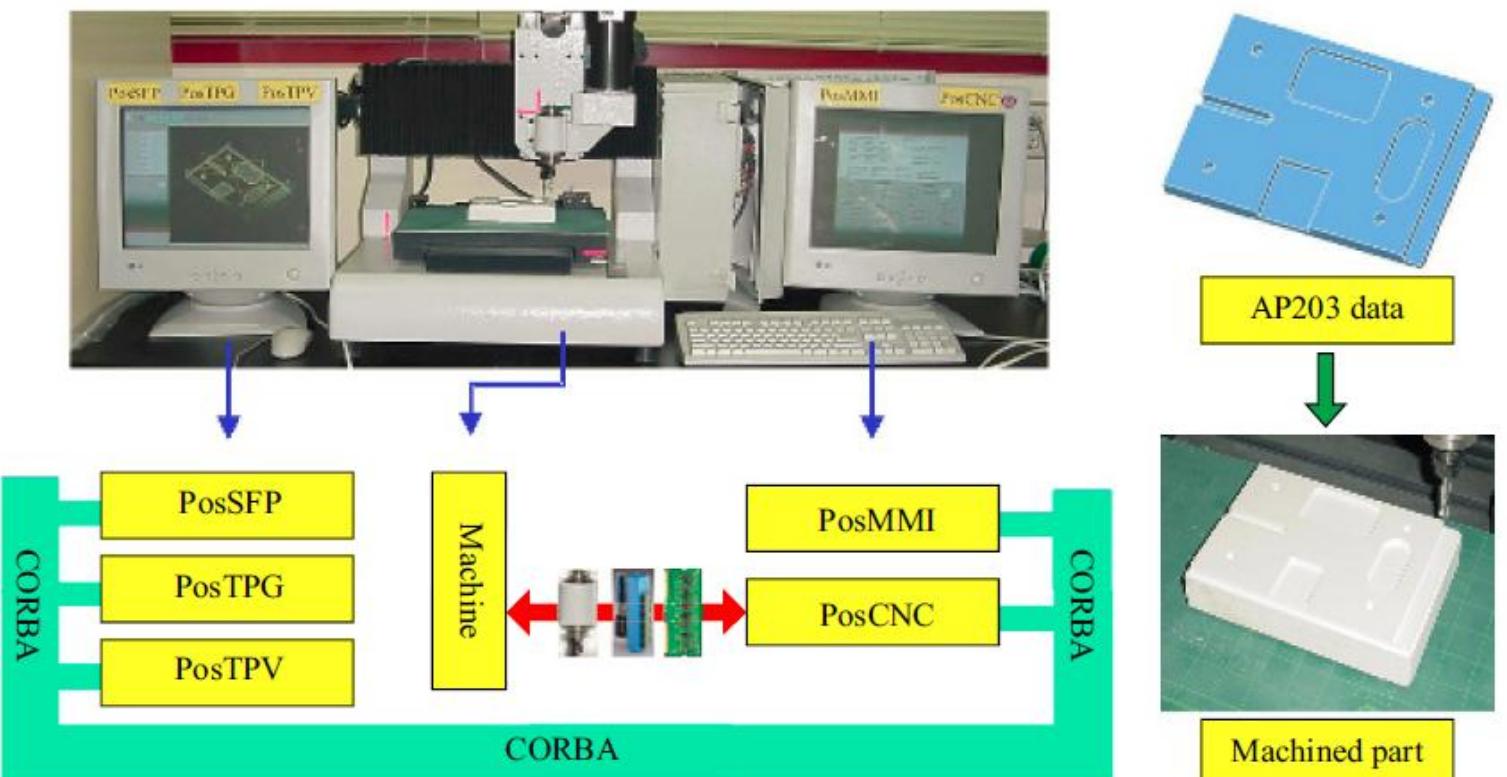
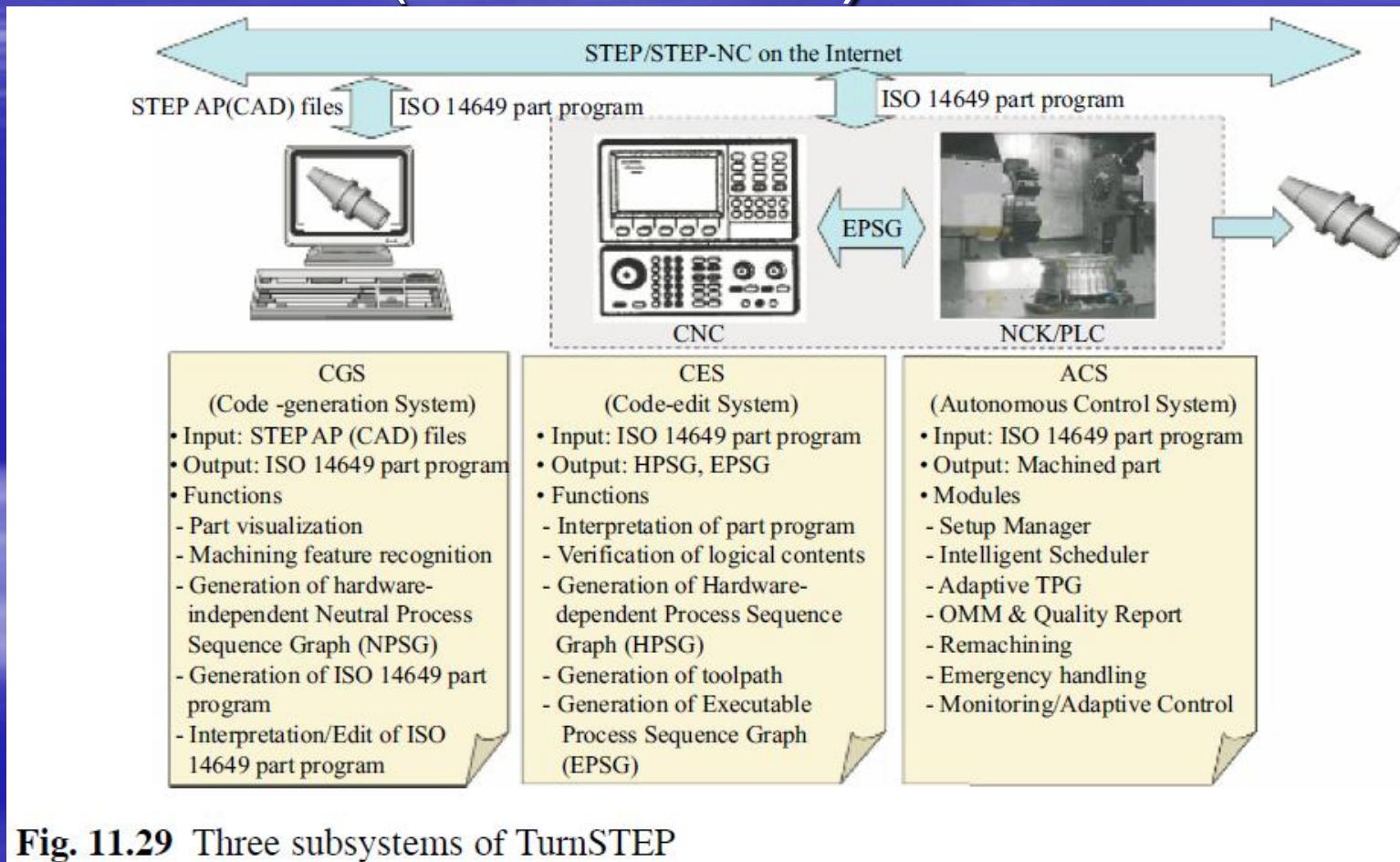


Fig. 11.28 The prototype Korea STEP-NC and the machined part

# Worldwide Research and Development

## ■ POSTECH (South Korea)

浦項工科大學 (南韓)



# Worldwide Research and Development

洛桑聯邦理工學院 (瑞士)

- Ecole Polytechnic Fédérale Lausanne (Switzerland )
  - STEP-NC work at the EPFL concentrated on **EDM** with other Swiss partners.
  - As well as control based on STEP-NC, design feature, feature-based process planning and optimization methods are being developed.
  - For manufacturing, possible feature information from CAD **may or may not be useful** for manufacturing, depending on the reasons for which they were introduced.
  - Current work is on Malcolm Sabin's back-building process planning method, involving recognizing and selecting sets of features and removing them successively until the desired stock is reached.
  - The features removed are recorded for organization into a 'micro' process plan for machining using STEP-NC.

# Worldwide Research and Development

巴斯大學 (英國)

- University of Bath (UK)
  - Research at the University of Bath is developing a novel universal manufacturing platform that utilizes the STEP-NC data models and accentuates it with **the functionality of mobile agents** and **manufacturing knowledge-bases**.
  - In addition to CAD, CAM, CAPP and CNC interfaces, **business applications** such as ERP, scheduling and costing can also exchange information with the various system.

# Worldwide Research and Development

巴斯大學 (英國)

## ■ University of Bath (UK)

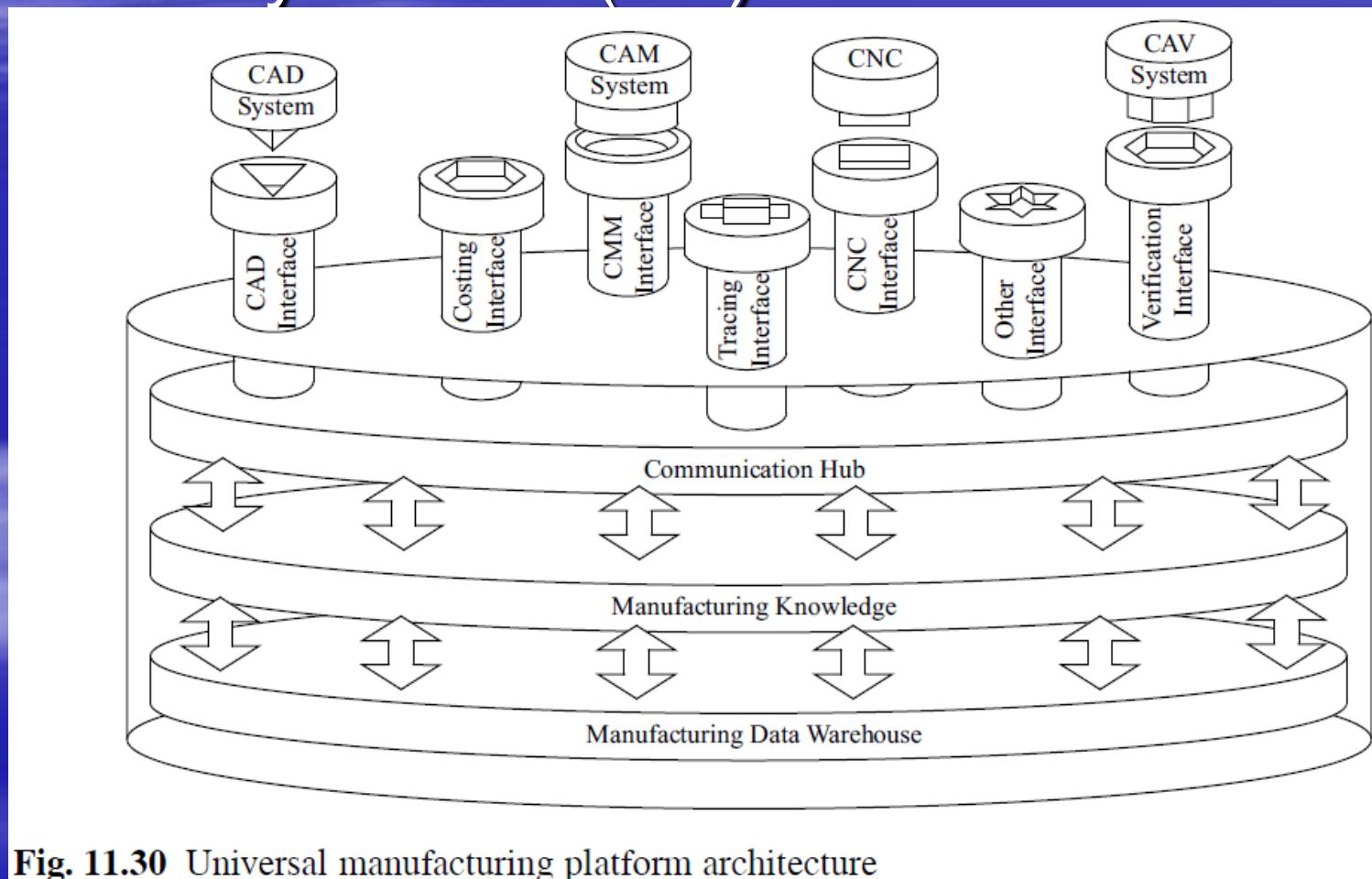


Fig. 11.30 Universal manufacturing platform architecture

# Worldwide Research and Development

- NIST (USA) 國家標準技術研究院 (美國)
  - MEL's Smart Machining Systems program is focusing on issues relevant to **CNC interoperability**.
  - The program considers 'smart data' to be vital to achieving smart machines.
  - The NIST Advanced Technology Program funded '**Super Model**' to validate the use of STEP-NC in manufacturing applications.
  - NIST has developed a **dynamic optimizer** that uses **physics-based models** of machining, coupled with measurement of machine tool performance and tool characteristics, to generate optimal speed and feed settings that reduce cycle times.
  - The Matlab-based optimizer was recently **coupled with a STEP-NC front end** that takes a process plan for turning, extracts relevant information for optimization, runs the optimizer, and merges the optimized parameters back into the original STEP-NC file.

# Future Prospects

- Research and development on **STEP-Manufacturing** has been actively pursued and it has been **demonstrated to work** in practice both internationally and locally.
- At present, an effort has been made to apply the technique to real industrial areas.
- However, truly, it is **hard** to realize full STEP-Manufacturing in one step due to the **time, cost** and **technological difficulties**.
- For this reason, the authors suggested the STEP-Manufacturing Roadmap more focused on the **STEP-NC** domain.

# Future Prospects

**Table 11.3** STEP-Manufacturing roadmap

		1 step (the beginning period)	2 step (the employment period)	3 step (the completion period)
Objective/ Benefit		STEP-Mfg infra introduction through the minimum investment	Merit acquisition of STEP-Mfg by STEP-Mfg settlement	e-Mfg paradigm implementation based on STEP-Mfg
Time frame		2 year (TBA)	3~4 year (TBA)	After 5 year (TBA)
Infra range		Intranet (in company)	Internet (in local area)	Internet (international)
Information exchange level		Hybrid (STEP, STEP-NC, G-code)	Partial STEP-Mfg (STEP AP203, ISO14649)	Full STEP-Mfg (STEP APs, ISO14649, ...)
Implementation level	CNC	Type 1 (conventional control) via post-processing	Type 2 (new control) via new & w/STEP-NC interpreter (Siemens)	Type 3 (intelligent control) via new & intelligent controller (TurnSTEP-ACS)
	CAD/CAM	Legacy software with STEP-NC interface (ST-Plan, ST-Machine)	STEP & STEP-NC based CAPP/CAM (PosSFP, TurnSTEP-CGS)	CAPP/CAM for intelligent STEP-Mfg (TurnSTEP-CES)
Required technology	STEP-x interface	STEP-NC interpreter, Post-processor for Type 1 (STEP-NC → G-code)	STEP & STEP-NC interpreter, STEP-NC converter (G-code → STEP-NC)	STEP, STEP-NC interpreter
	Web service	Web-service build-up in server side (settlement of web-service range)	STEP-Mfg application build-up in client side	Client-Server harmonization and improvement
	DB build-up	Local DB	Global DB (STEP-Mfg repository)	Global DB (STEP-Mfg repository)
Role division	Company	Intranet infra in company, STEP-Mfg introduction	STEP-Mfg infra employment	e-Mfg infra employment
	R&D center	STEP-Mfg component technology research and spread	Component technology development, Conformance verification	Verification of reliability, conformance, interoperability
	Government	STEP-Mfg introduction support, local IT infra build-up business	Infra technology employment business, Local IT infra build-up business	Commercial use business, certification business, IT infra enlargement (nation)

End