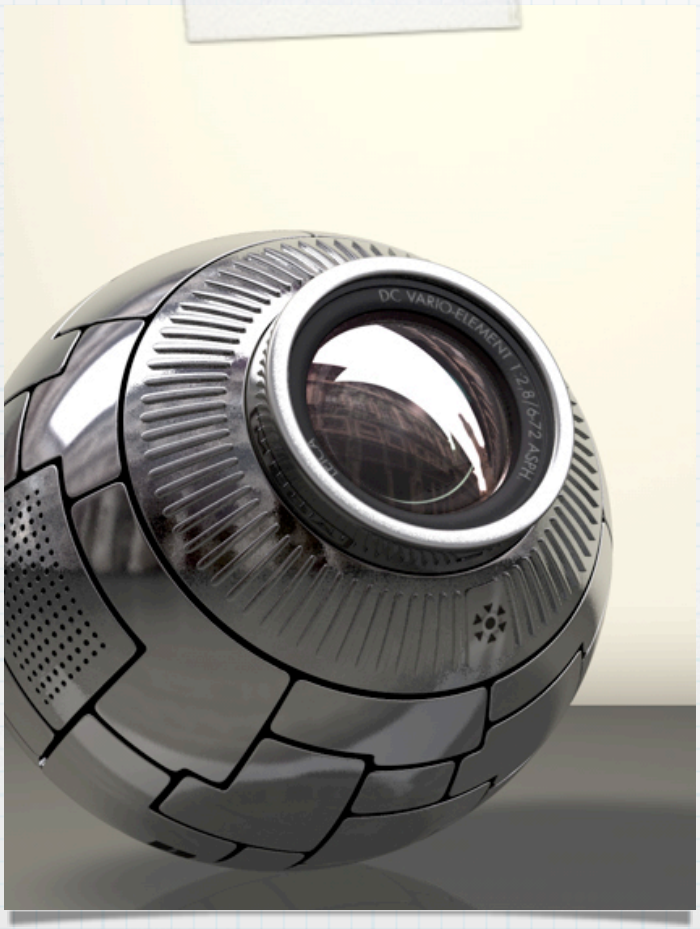
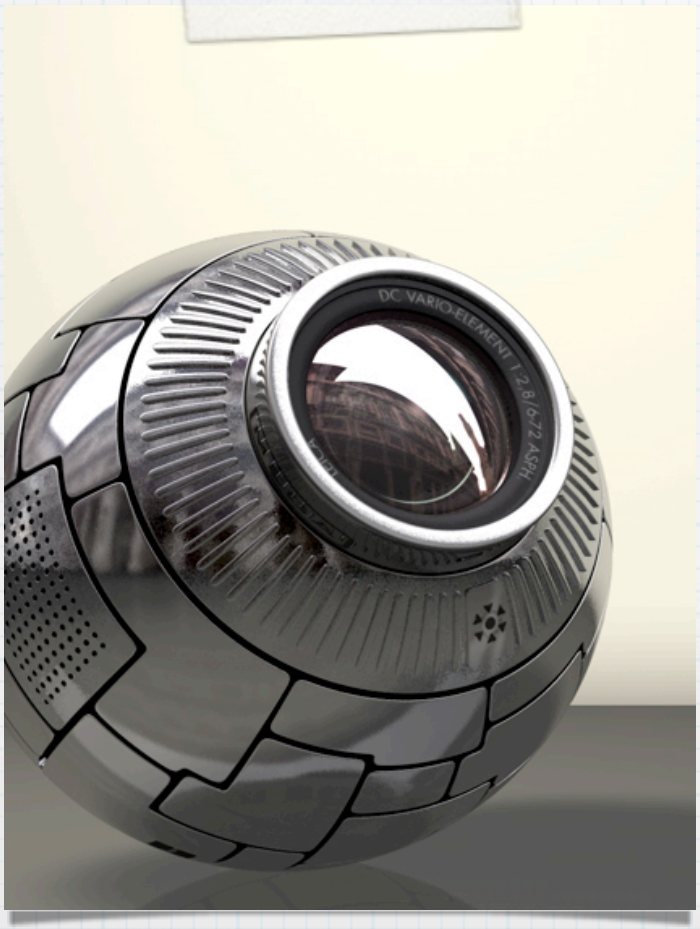
Generator (SASD)
Class B T10 → T8



T8

Presentor : 200810048 정재근
200811414 김연준
200811445 이성현
200812423 김준식

CFG Generator (SASD)



Contents

Structured Analysis

- Statement of Purpose
- Data Flow Diagram

Structured Design

- Structured Charts(Advanced)

Implementation & Reference

- Compare SA Process and Code

Statement of Purpose - CFG (1/2)

The program for auto making system for Control Flow Graph,
layout directivity Graph of analyze CFG for source

Execution must be using by gcc compile, Cygwin

Result printed on CUI, Can execute in Command Line instruction.
-If input instruction has been wrong, print help message

Entering the code written in C language.
If entering is successful the success message print
If that fails, a failure message is output

Code written in C language,
which includes the Main Function Line about 100 to 200,
User-defined header file, pointers are not allowed

Statement of Purpose - CFG (2/2)

When start converting, print "start" message.

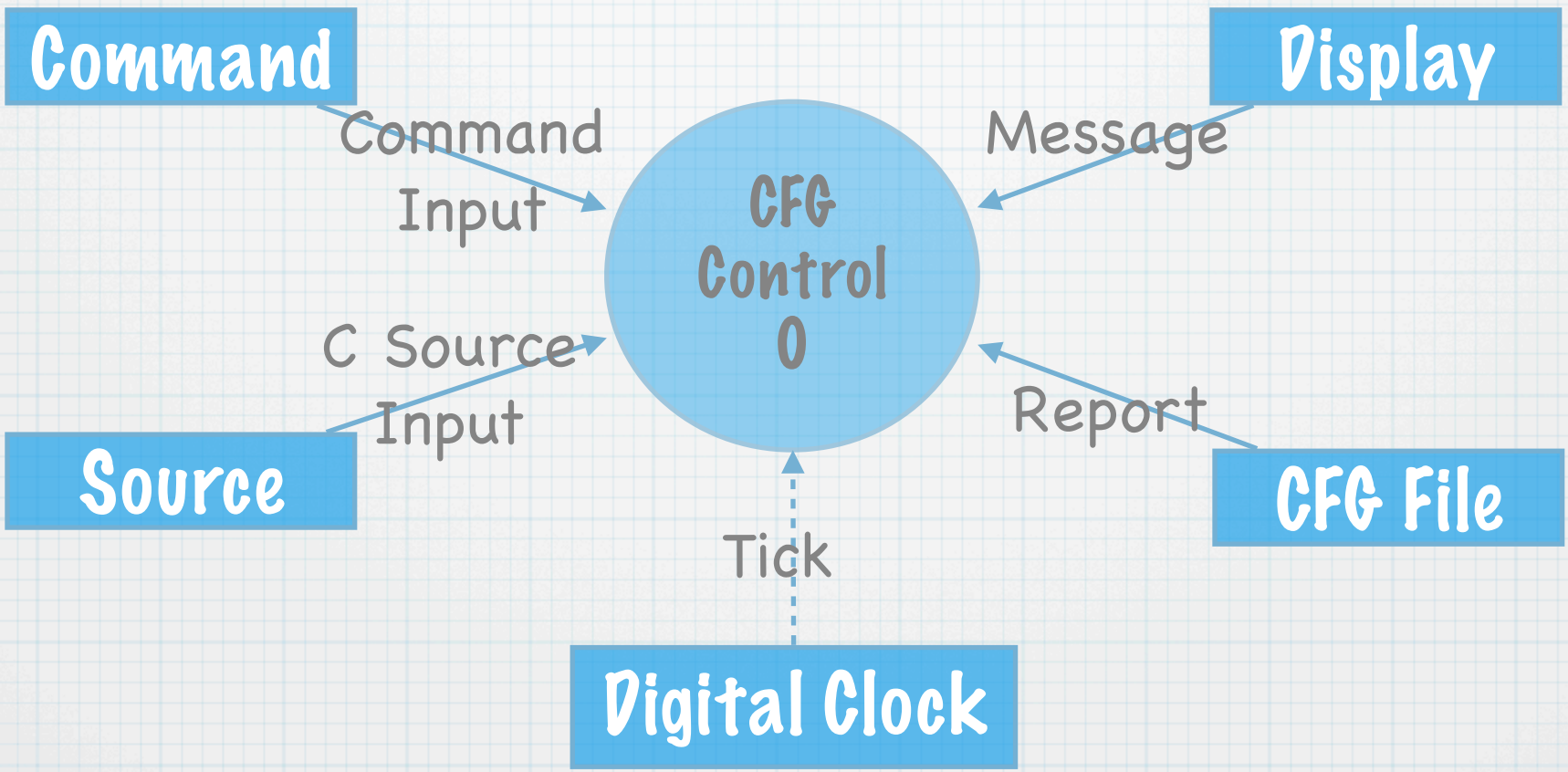
Block have "Start", "Finish" Nodes.

Branching node have lower nodes, Depending on condition,
distinguish output edge.

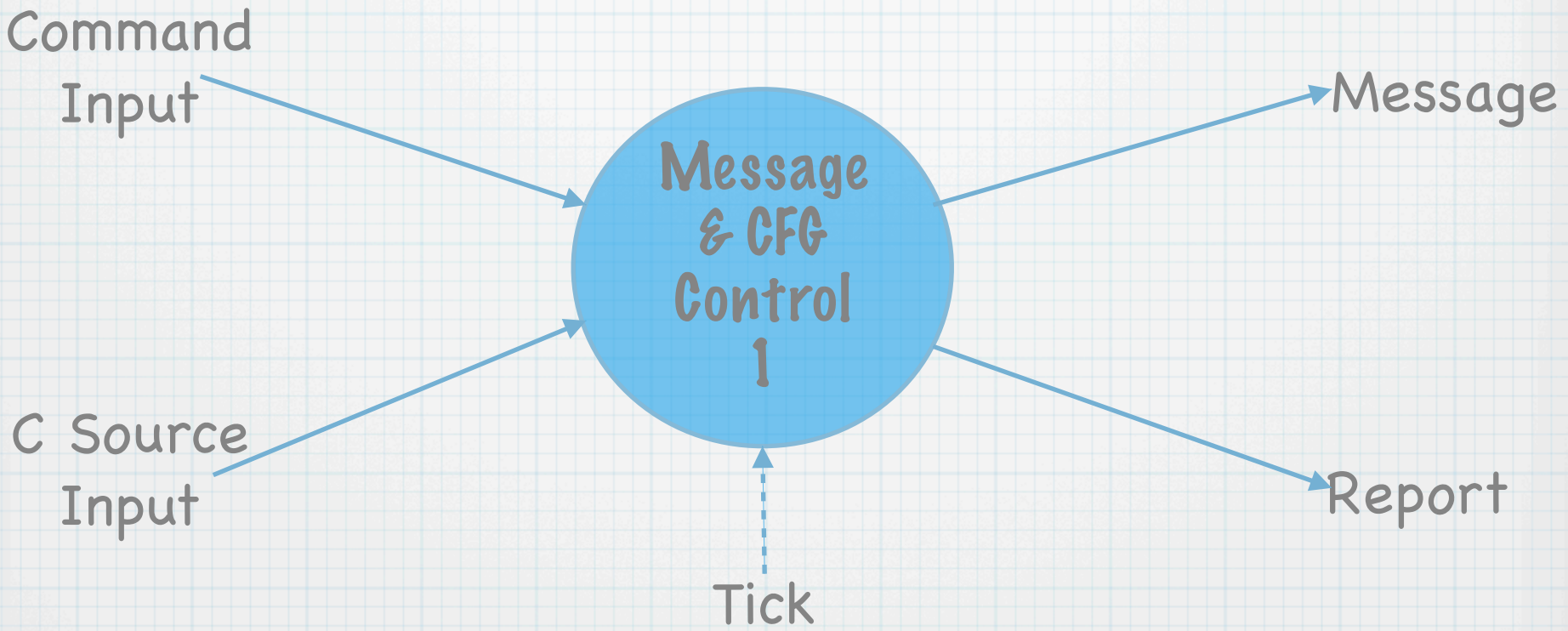
Print CFG Converting process in CUI format

Take code written in C language, generate report file converted to CFG

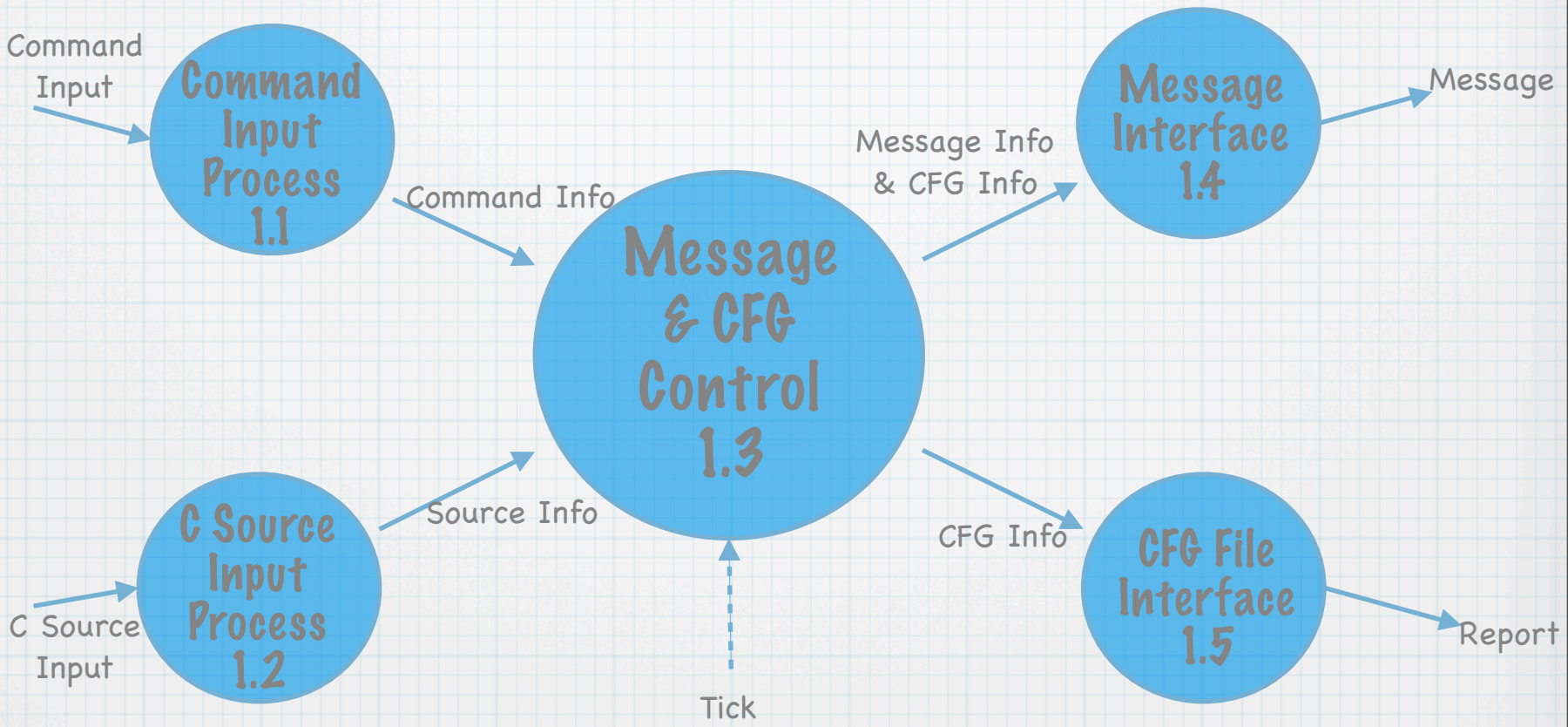
Data Flow Diagram Level 0



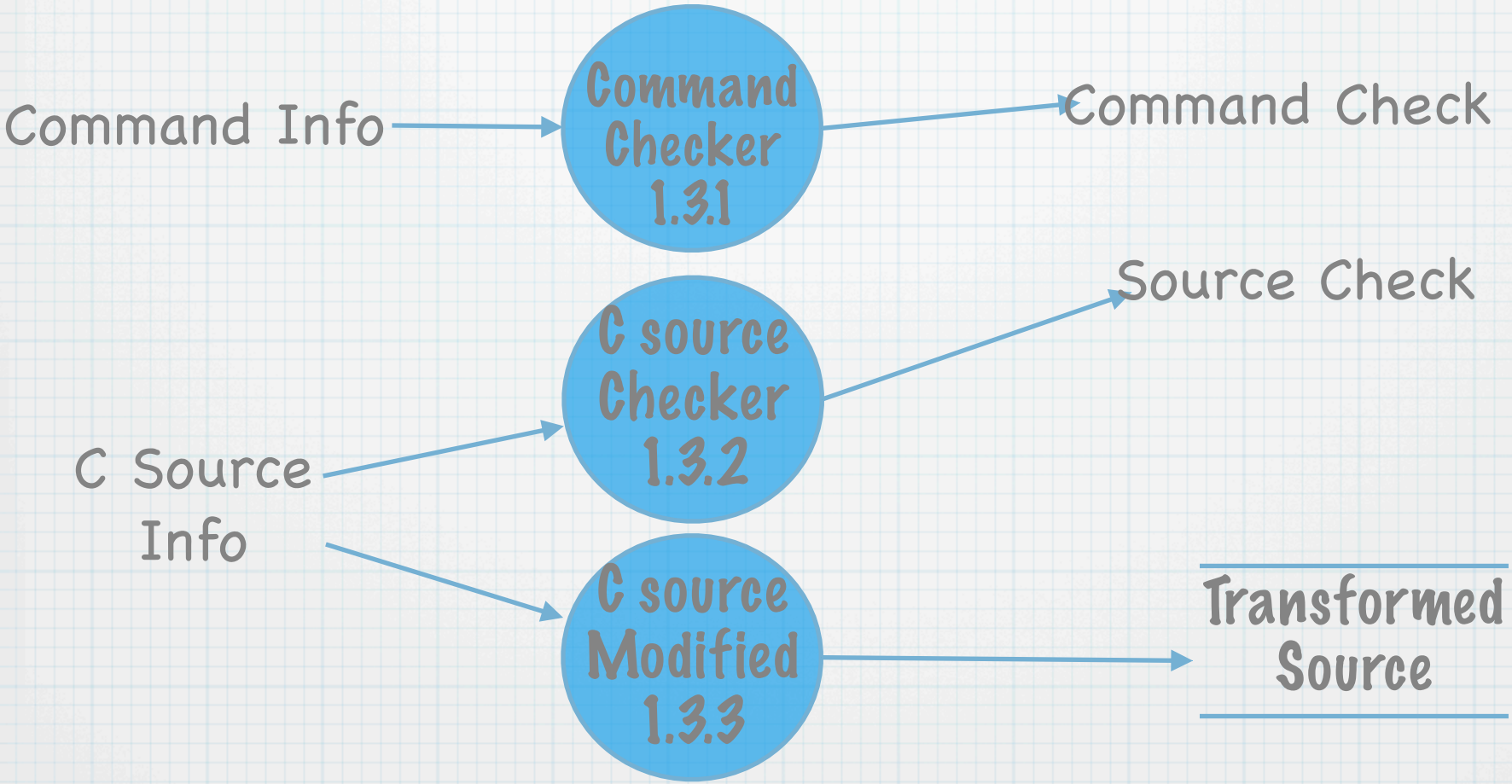
DFD Level 1


Command
Input
C Source
Input
Message
& CFG
Control
1
Message
Report
Tick

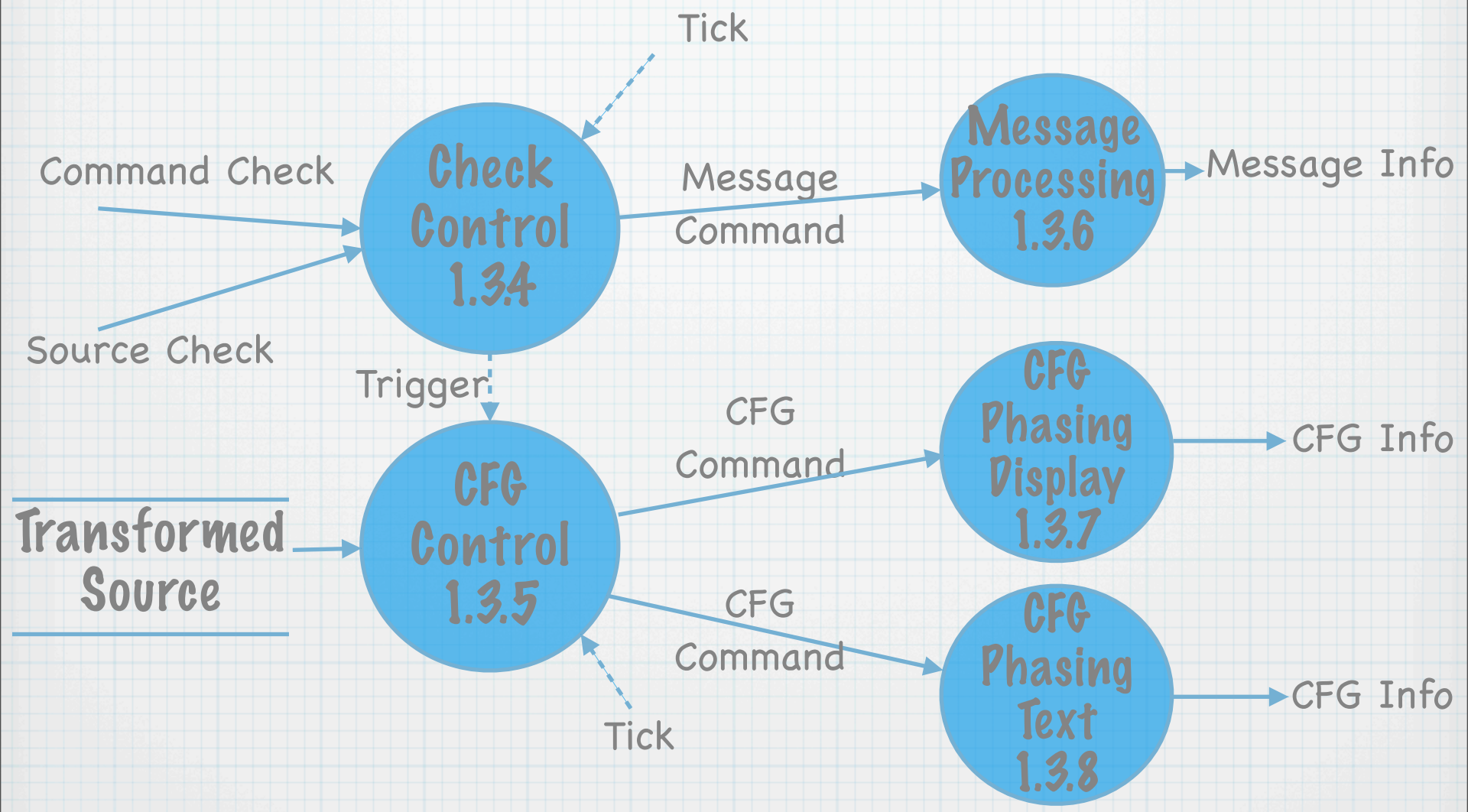
DFD Level 2


Command Input
Command Input Process 1.1
Command Info
Message & CFG Control 1.3
Message Info & CFG Info
Message Interface 1.4
Message
C Source Input
C Source Input Process 1.2
Source Info
Tick
CFG Info
CFG File Interface 1.5
Report

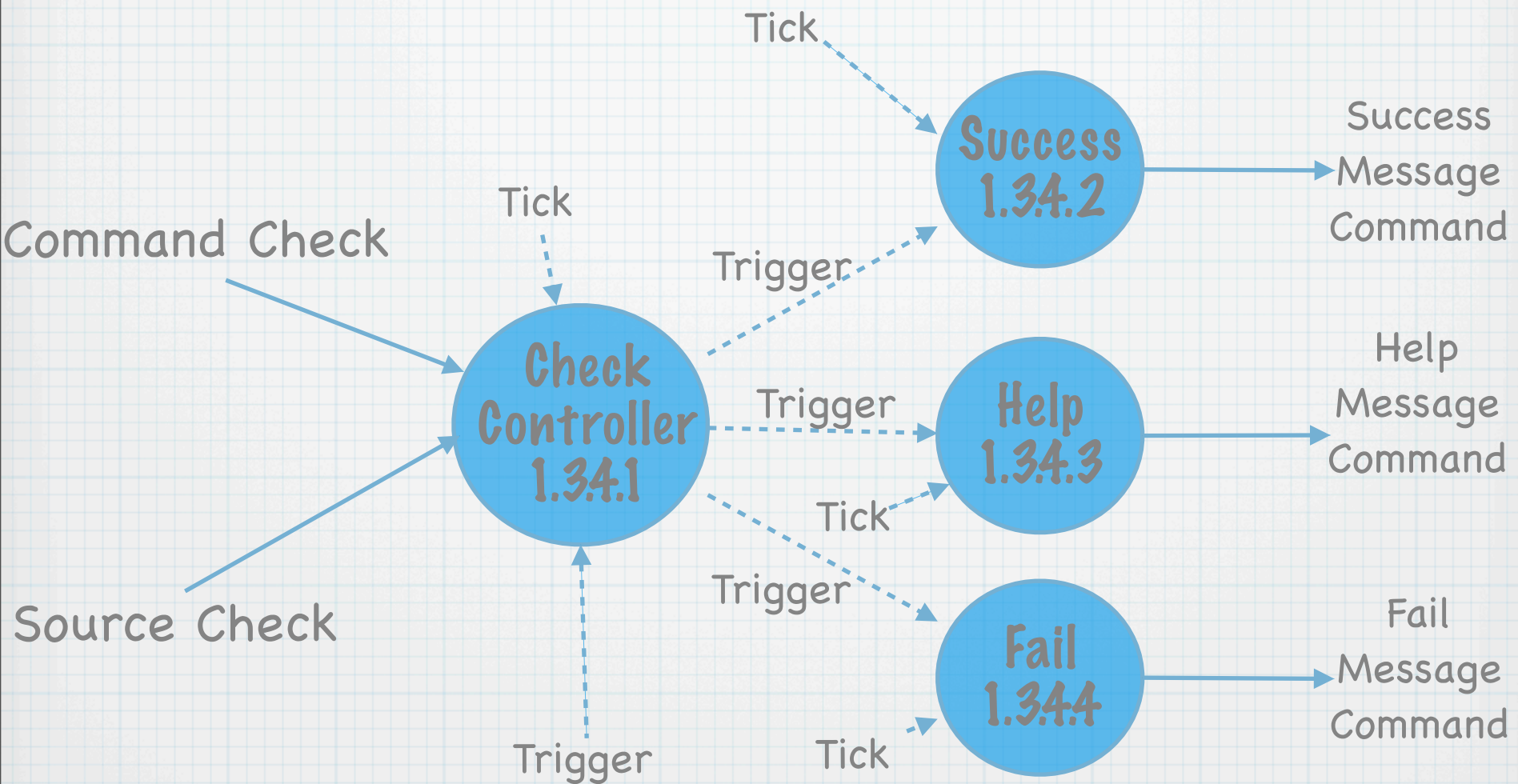
DFD Level 3


Command Info
Command Checker 1.3.1
Command Check
Source Check
C source Checker 1.3.2
C Source Info
C source Modified 1.3.3
Transformed Source

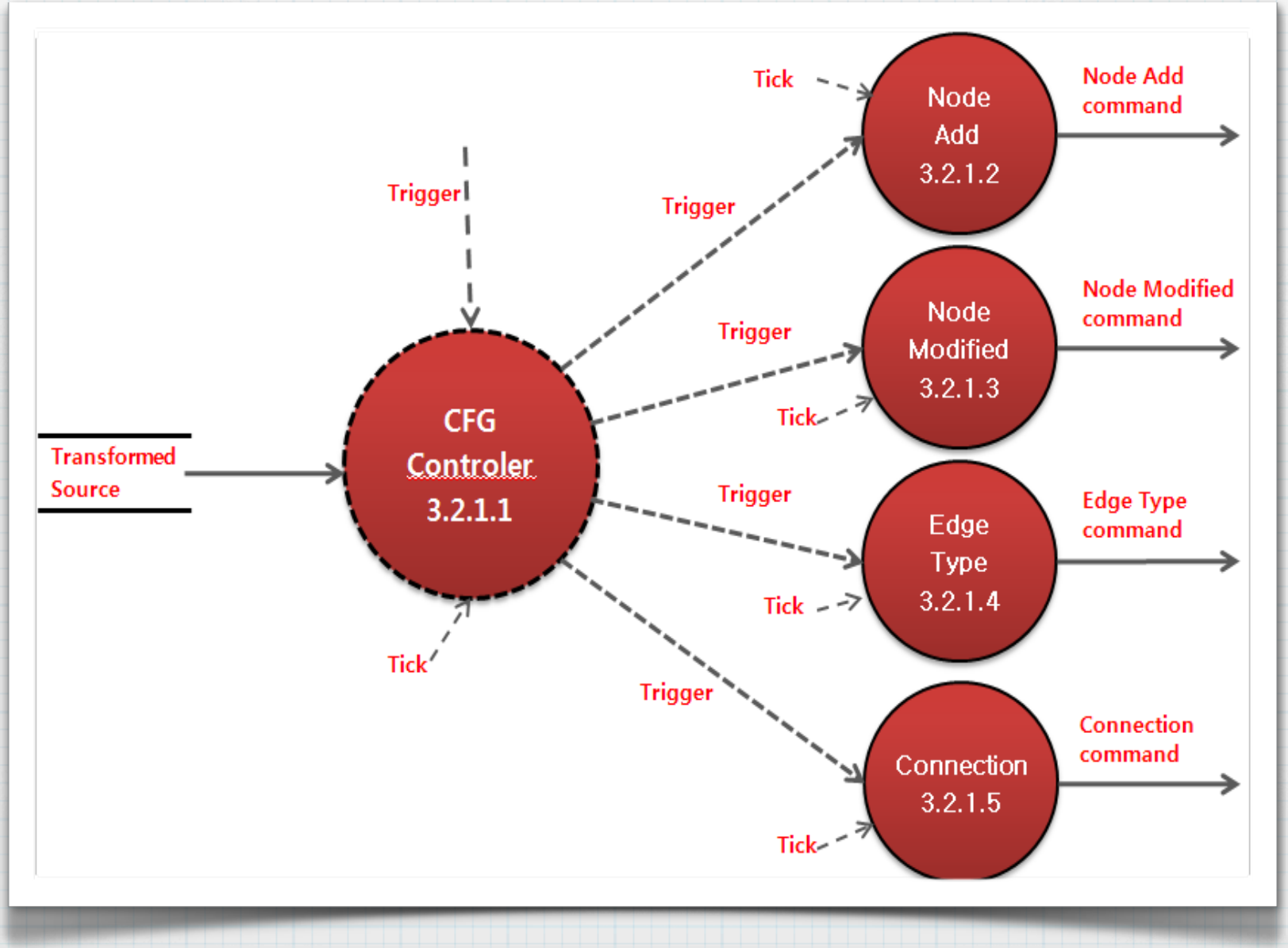
DFD Level 3


Tick
Command Check
Check Control 1.3.4
Message Command
Message Processing 1.3.6
Message Info
Source Check
Trigger
CFG Command
CFG Phasing Display 1.3.7
CFG Info
Transformed Source
CFG Control 1.3.5
CFG Command
CFG Phasing Text 1.3.8
CFG Info
Tick

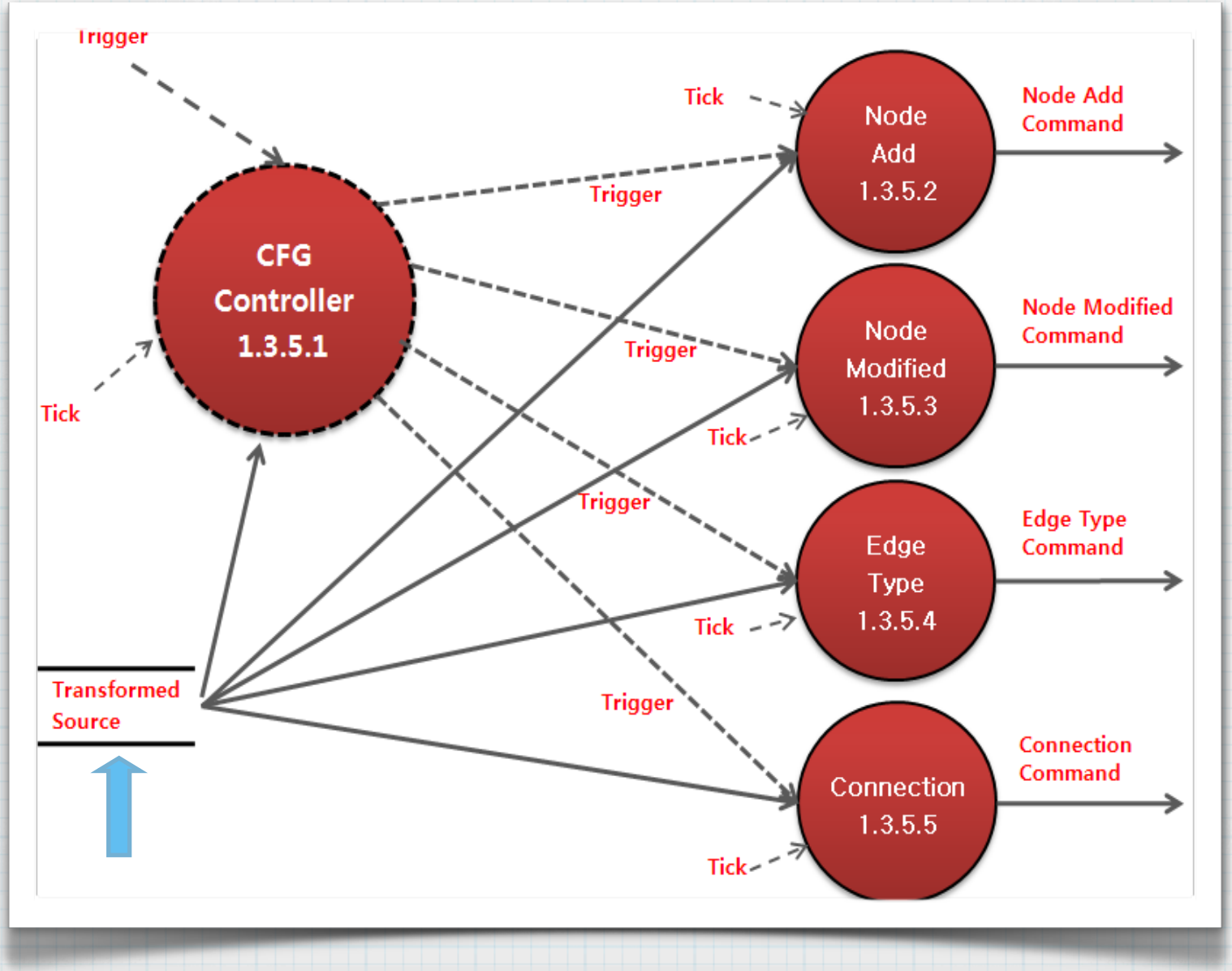
DFD Level 4


Tick
Success
1.34.2
Success
Message
Command
Tick
Command Check
Trigger
Check
Controller
1.34.1
Trigger
Help
1.34.3
Help
Message
Command
Tick
Trigger
Fail
1.34.4
Fail
Message
Command
Source Check
Trigger
Tick

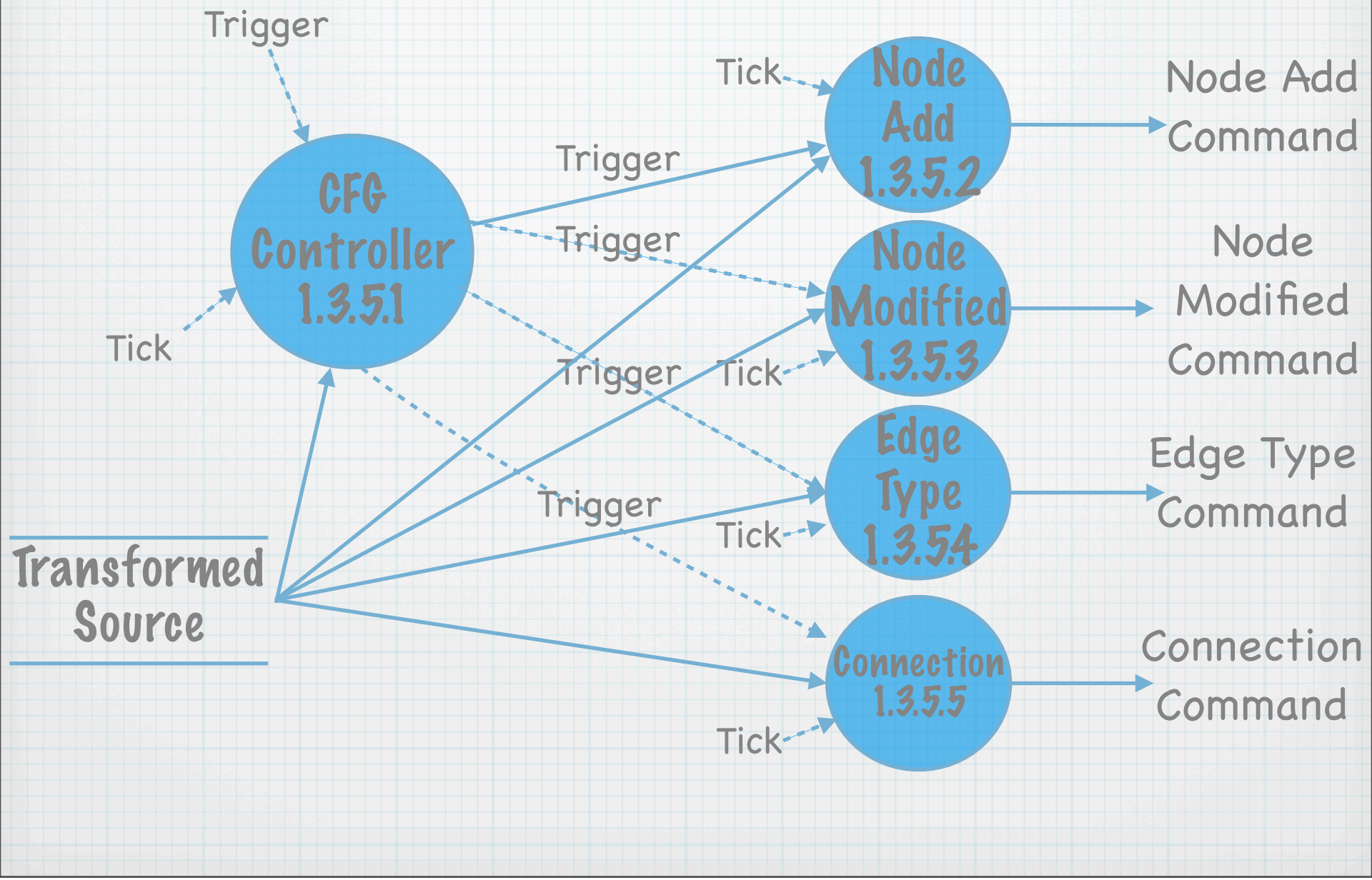
DFD Level 4


Tick
Node Add command
Node Add 3.2.1.2
Trigger
Trigger
Node Modified command
Node Modified 3.2.1.3
Trigger
Tick
Transformed Source
CFG Controler 3.2.1.1
Trigger
Edge Type command
Edge Type 3.2.1.4
Tick
Tick
Trigger
Connection command
Connection 3.2.1.5
Tick

DFD Level 4


Trigger
Tick
Node Add Command
Node Add 1.3.5.2
Trigger
CFG Controller 1.3.5.1
Trigger
Node Modified Command
Node Modified 1.3.5.3
Tick
Tick
Trigger
Edge Type Command
Edge Type 1.3.5.4
Tick
Transformed Source
Trigger
Connection Command
Connection 1.3.5.5
Tick

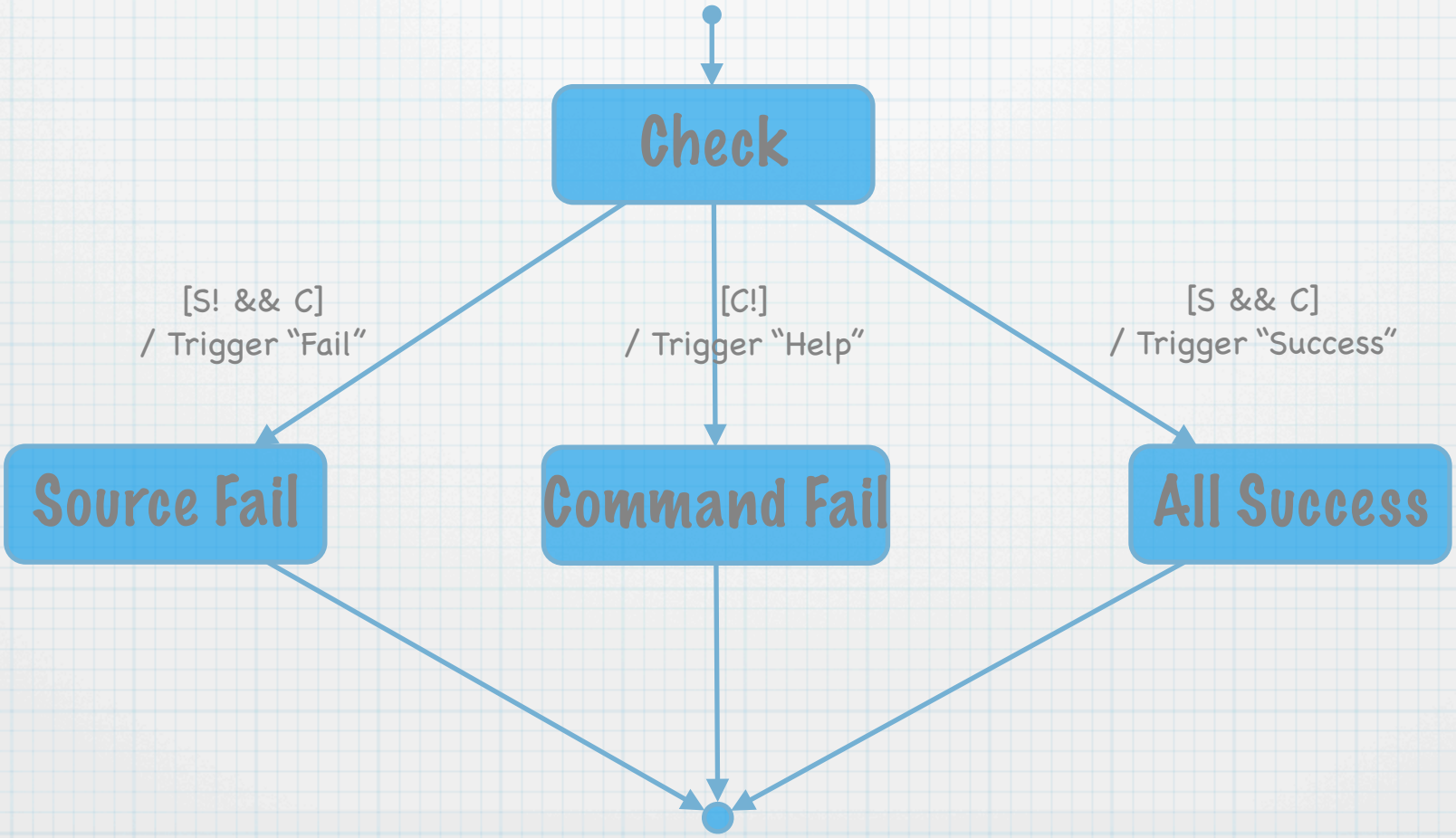
DFD Level 4


Trigger
Tick
Node Add 1.3.5.2
Node Add Command
Trigger
CFG Controller 1.3.5.1
Trigger
Node Modified 1.3.5.3
Node Modified Command
Tick
Trigger
Tick
Edge Type 1.3.5.4
Edge Type Command
Trigger
Tick
Transformed Source
Connection 1.3.5.5
Connection Command
Tick

DFD Level 5

S	Source Input of Success
C	Command Input of Success

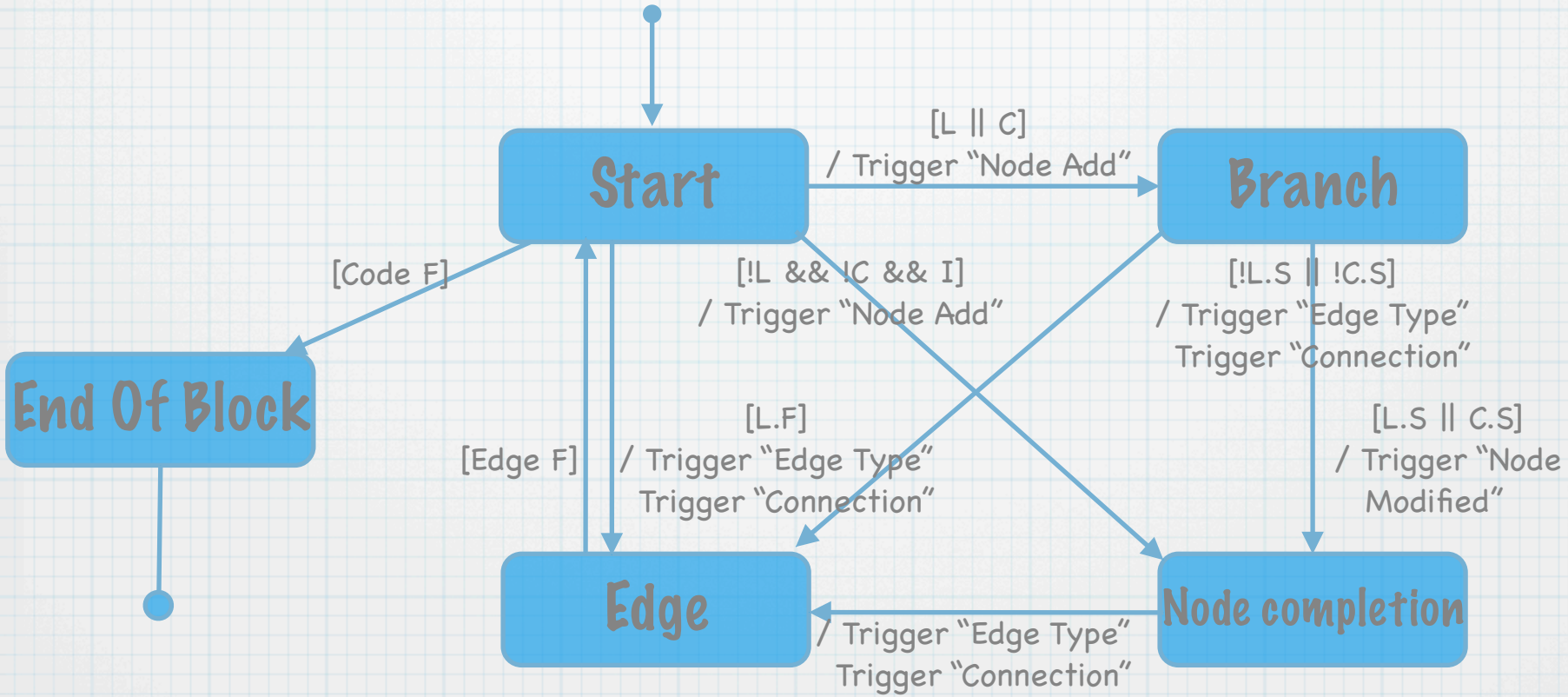
State Transition Diagram for
Controller 1.3.4.1 (Check Controller)



DFD Level 5

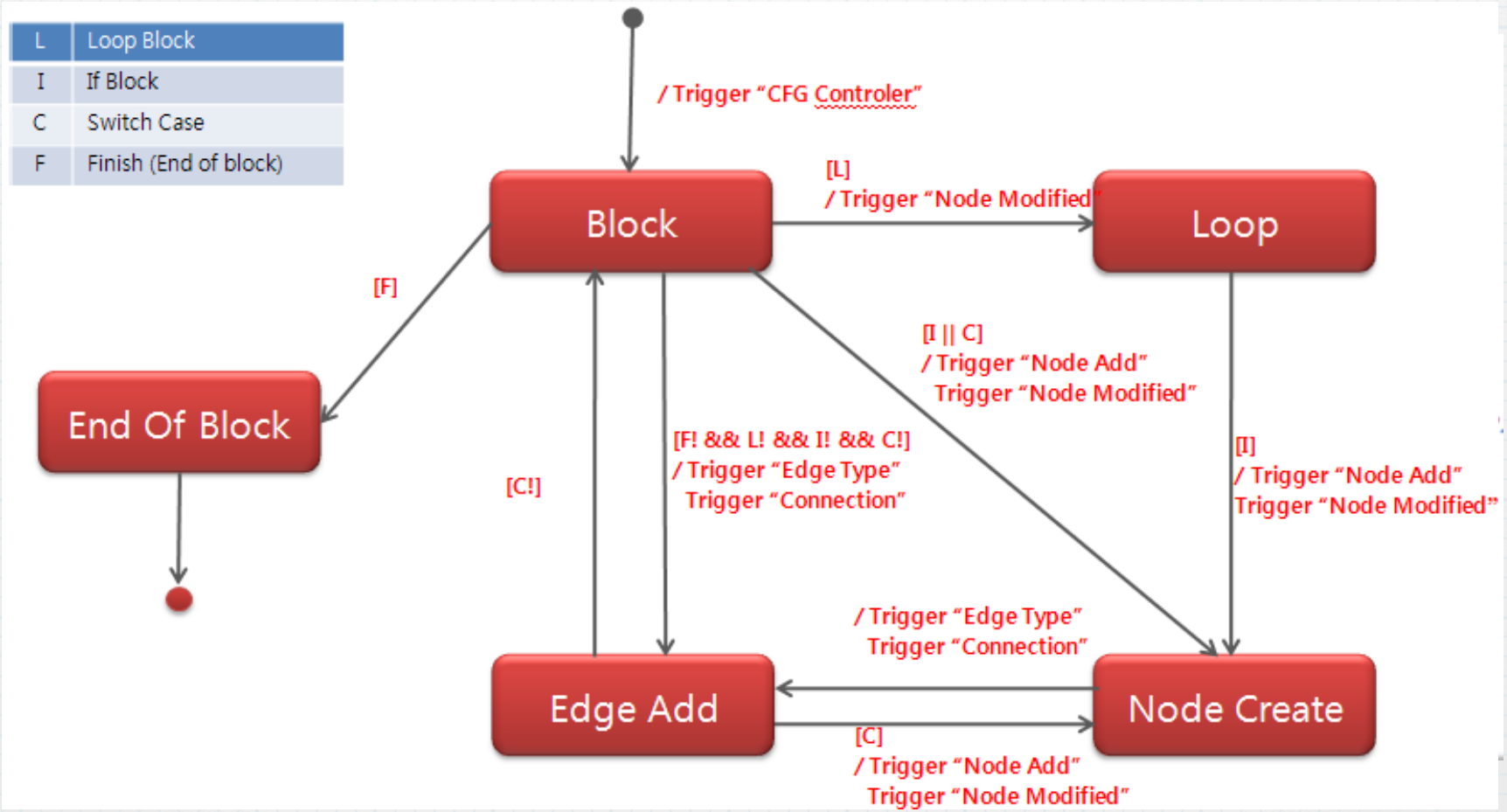
L	In Loop
I	If Just Block
C	In Condition
F	Finish (Line)
S	Start Line

State Transition Diagram for Controller 1.3.5.1 (CFG Controller)



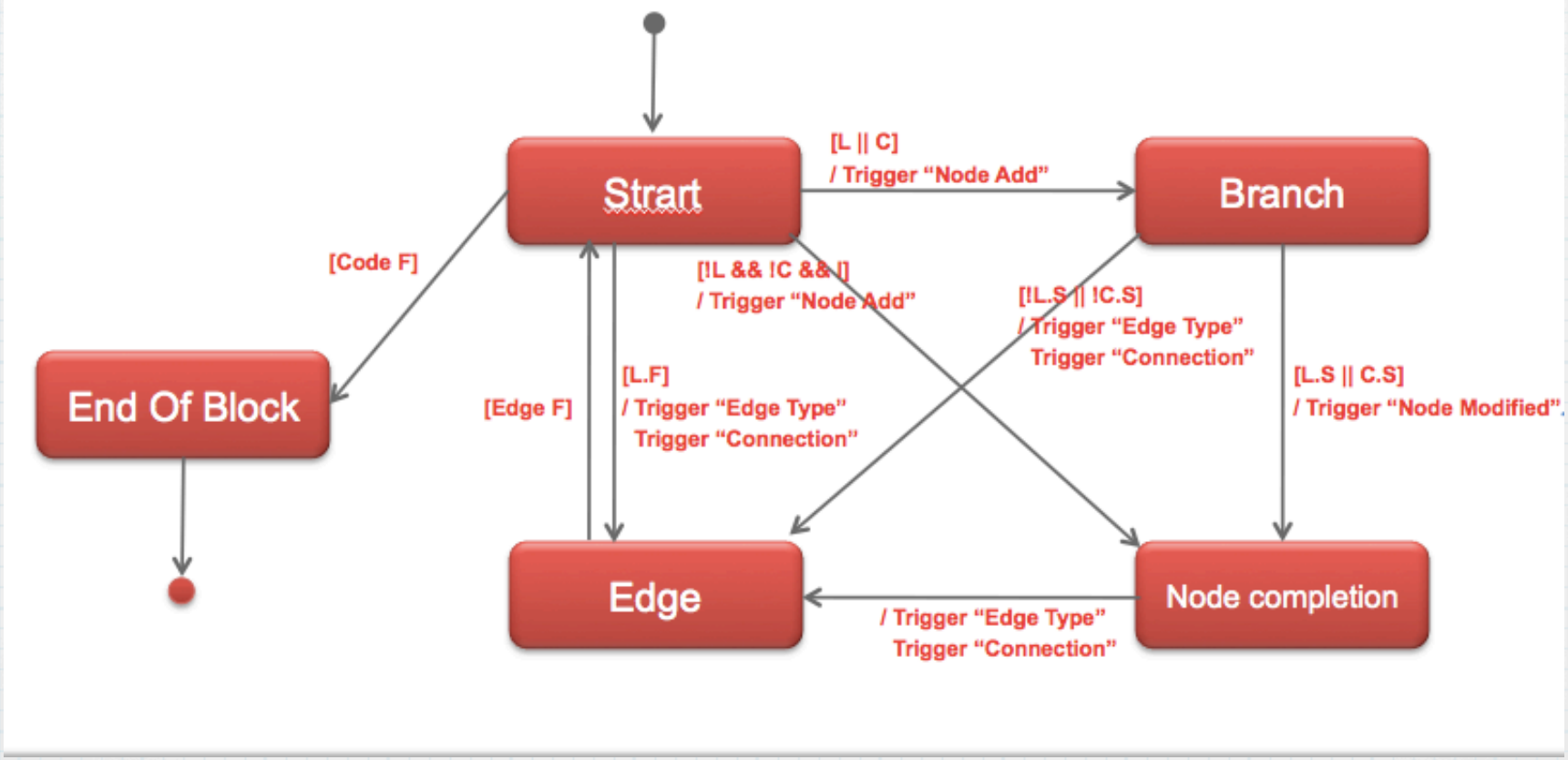
DFD Level 5

State Transition Diagram for Controller 1.3.5.1 (CFG Controller)

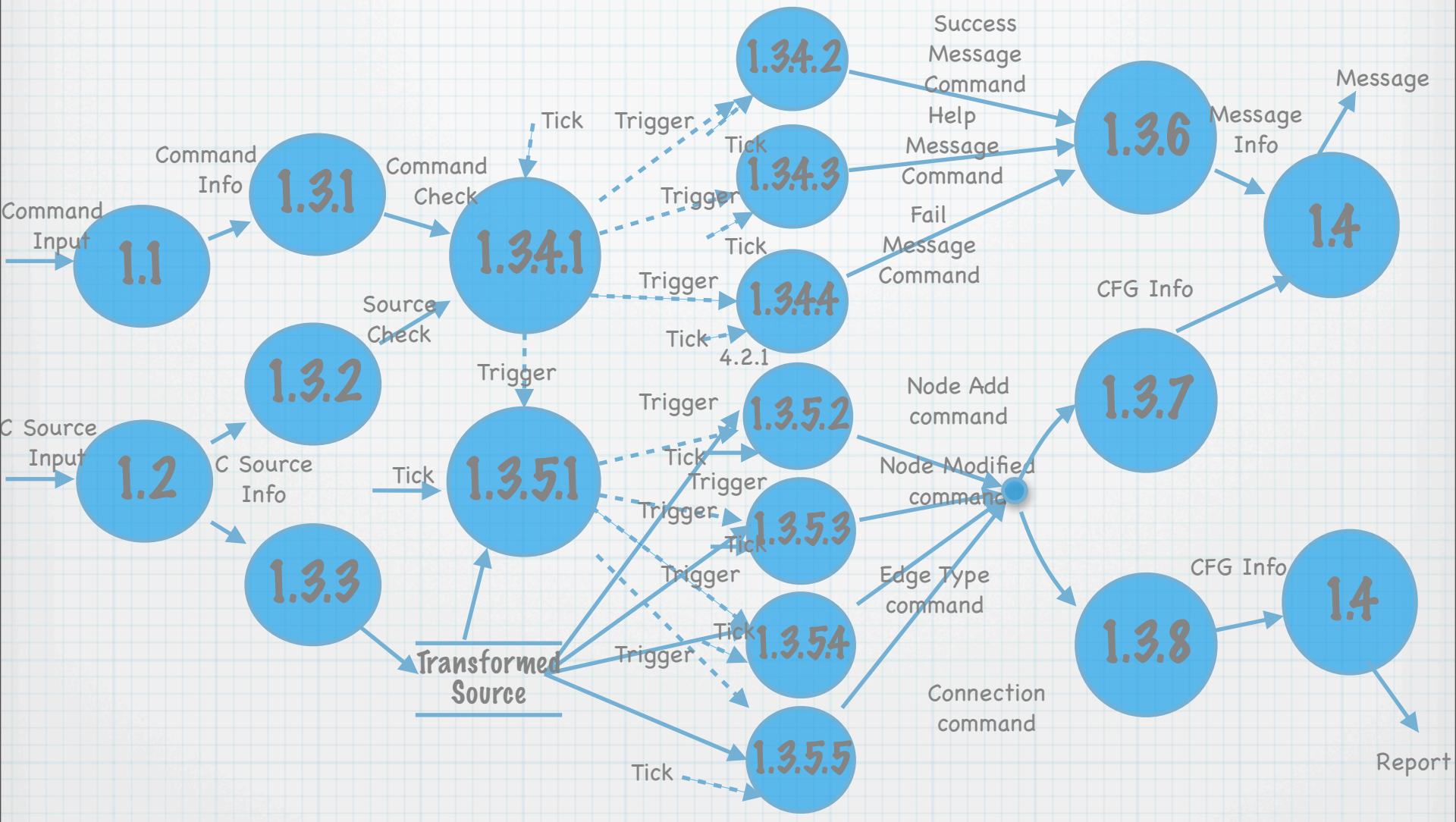


DFD Level 5

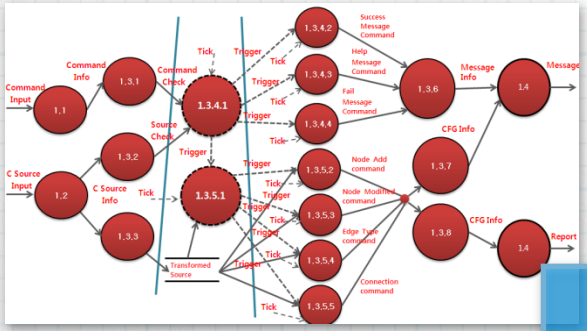
State Transition Diagram for Controller 1.3.5.1 (CFG Controller)



DFD Total


Command Input
1.1
Command Info
1.3.1
Command Check
1.3.4.1
Tick
Trigger
Source Check
1.3.2
C Source Input
1.2
C Source Info
1.3.3
Tick
1.3.5.1
Trigger
Transformed Source
1.3.4.2
Tick
1.3.4.3
Trigger
Tick
1.3.4.4
4.2.1
Trigger
Tick
1.3.5.2
Trigger
Tick
1.3.5.3
Trigger
Tick
1.3.5.4
Trigger
Tick
1.3.5.5
Success Message Command
Help Message Command
Fail Message Command
1.3.6
Message Info
Message
1.4
CFG Info
1.3.7
Node Add command
Node Modified command
Edge Type command
Connection command
1.3.8
CFG Info
1.4
Report

Structured Charts- CFG Generator(Advanced)



Main

Check Controller

CFG Controller

Trigger

Command Check

Source Check

Transformed Source

Transformed Source

Trigger

Trigger

Trigger

Trigger

Trigger

Trigger

Trigger

Transformed Source

Success

Help

Fail

Node Add

Node Modified

Edge Type

Connection

Transformed Source

Message Command

CFG Command

Command Checker

C source Checker

C source Modified

Message Processing

CFG Phasing Display

CFG Phasing Text

Command Info

Source Info

Source Info

Success Message

CFG Phase

CFG Phase

Command Input Process

C Source Input Process

Message Interface

Proccess Interface

Struct Block Information

```
struct Block
{
        char* info;
        int block_num;
        int block_type;
        int edge_type;
        int visit;
        int condition;
        struct Block* next[5];
        struct Block* prev;
};
```

Block Content
Block Number
Block Type(■ or ◆)
Edge Type(Forward or Back)

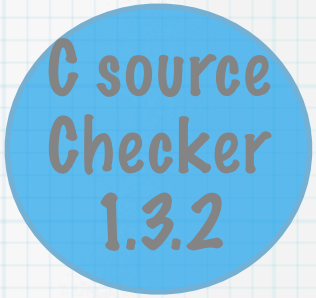
Next Block Address
Previous Block Address

1.3.1 Command Checker


```
int Command_Checker(int argc, const char *argv[])
{
        if(argc < 3)
        {
                return 0;
        }
        else if(argc > 3)
        {
                return 0;
        }
        else if(strcmp(argv[1],"inputcode.txt") != 0 || strcmp(argv[2],"result.txt") != 0)
        {
                return 0;
        }
        return 1;
}
```

Reference No.	1.3.1
Name	Command Checker
Input	Command Input
Output	Command Check
Process Description	Processing input data in Command Line Type, input success & failure(True, False) is passed to Check Control

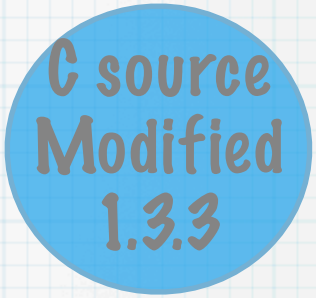
1.3.2 C source Checker




```
int C_Source_Checker(int argc, char *argv[])
{
        if(argc = 3){
                FILE *fin;
                fin=fopen(argv[1],"r");

                if(fin == NULL)
                {
                        return 0;
                }
                fclose(fin);
        }
        return 1;
}
```

Reference No.	1.3.2
Name	C Source Checker
Input	C Source Input
Output	Source Check
Process Description	C Code input success & failure(True, False) is passed to Check Control

1.3.3 C source Modified




```
int C_Source_Modifed(char *list[], char *argv[])
{
        int i = 0;
        int line = 0;
        char buffer[100];
        FILE *fin;
        fin=fopen(argv[1], "r");

        while (fgets(buffer,100,fin)!= NULL)
        {
                list[i]= (char*)malloc(sizeof(char)*100);

                if(strlen(buffer)!=2)
                {
                        strcpy(list[i],buffer);
                        i++;
                        line++;
                }
        }
        fclose(fin);

        return line;
}
```

Reference No.	1.3.3
Name	C Source Modified
Input	C Source Input
Output	Transformed Source
Process Description	CFG Control to handle the C Code inputs, converted source information (Block List) is stored in Transformed Source (Data Store)

1.3.4.1 Check Controller


```
ccheck = Command_Checker(argc, argv);
scheck = C_Source_Checker(argc, argv);

if(ccheck == 0)
{
        Help();
        return;
}
else if(scheck == 0)
{
        Fail();
        return;
}
else if(scheck == 1 && ccheck == 1)
{
        line = C_Source_Modifed(list, argv);
        Success();
}
```

Reference No.	1.3.4.1
Name	Check Controller
Input	Command Check, Source Check
Output	Trigger
Process Description	Control Process that confirms the Input C code and commands success & failure(True, False) and send the suitable Trigger and Process.

1.3.4.2 Success


```
void Success()
{
        printf(" Input of Command & C Source Code is successful. \n **** CFG Converting Start. ****\n\n");
}
```

Reference No.	1.3.4.2
Name	Success
Input	Trigger, Tick
Output	Success Message Command
Process Description	If the Input of C code and commands is successful, send the suitable Message in String type.

1.3.4.3 Help


```
void Help()
{
        printf(" Wrong Input of Command. \n Must be entered like './CG inputcode.c result.txt'. \n");
}
```

Reference No.	1.3.4.3
Name	Help
Input	Trigger, Tick
Output	Help Message Command
Process Description	If command Input is unvalid, send the suitable help manual in String type.

1.3.4.4 Fail


```
void Fail()
{
        printf(" Failure of reading C Source Code. \n");
}
```

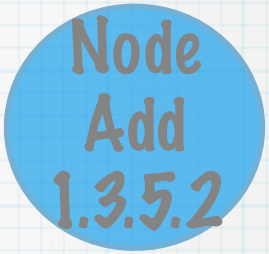
Reference No.	1.3.4.4
Name	Fail
Input	Trigger, Tick
Output	Fail Message Command
Process Description	If C code Input is unvalid, send the suitable Message in String type.

1.3.5.1 CFG Controller


```
fnum = Find_Function(list, line, p, function_name);

p= Node_Add(list,line,p, function_name, fnum);
Node_Modified(p,line);
Edge_Type(p);
Connection(p);
CFG_Phasing_Display(p, fnum, function_name);
CFG_Phasing_Text(p, fnum, function_name);
```

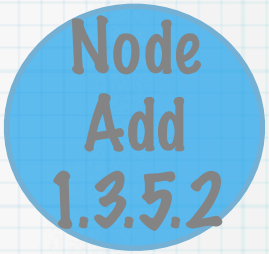
Reference No.	1.3.5.1
Name	CFG Controller
Input	Trigger, Transformed Source
Output	Trigger
Process Description	Decide Depending on Entering the Block list that make Node, modify Node, define Edge Type and Connection status and then send the suitable Trigger and Process.

1.3.5.2 Node Add



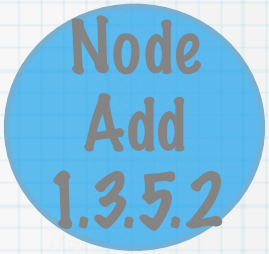
Reference No.	1.3.5.2
Name	Node Add
Input	Trigger, Tick, Transformed Source
Output	Node Add Command
Process Description	Make the new Node and define Node Type and the contents.

1.3.5.2 Node Add




```
block* Node_Add(char* list[], int line, block *p, char* function_name[], int fnum)
{
        block *head, *q;
        int i,n,j=0, k=0, flag=0;
        int number=1;
        for(i=0; i<line; i++)
        {
                k=0;
                j=0;

                if(strstr(list[i],"main") != 0)
                {
                        flag++;
                }
                if(strstr(list[i],"#include")!=0 )
                {
```

1.3.5.2 Node Add

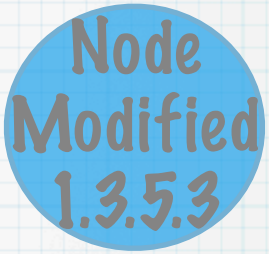



```
if(strstr(list[i], ");") != 0)
```

```
else if(strstr(list[i],"{")!=0 || strstr(list[i],"}")!=0 || strstr(list[i],"return")!=0
        || strstr(list[i],"main")!=0 || strstr(list[i], "else")!=0)
```

```
            }
            p->prev=q;
            q=p;
}
return head;
```

1.3.5.3 Node Modified



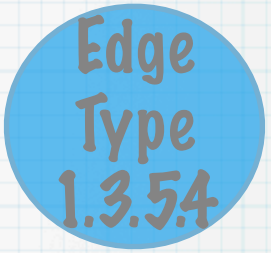

```
void Node_Modified(block *p, int line)
{
        while(p->next[0]!=NULL)
        {

                if(strstr(p->info,"for")!=0 )
                {
                        p->block_type=1;
                        p->edge_type=0;
                        p= p->next[0];
                }
                else if(strstr(p->info,"if")!=0 )
                {
                        p->block_type=1;
                        p->edge_type=0;
                        p= p->next[0];
                }
                else if(strstr(p->info,"while")!=0 )
                {
                        p->block_type=1;
                        p->edge_type=0;
                        p= p->next[0];
                }
                else if(strstr(p->info,"switch")!=0 )
                {
                        p->block_type=1;
                        p->edge_type=0;
                        p= p->next[0];
                }
                else
                        p= p->next[0];
        }
}
```

Reference No.	1.3.5.3
Name	Node Modified
Input	Trigger, Tick, Transformed Source
Output	Node Modified Command
Process Description	Change information in Node

1.3.5.4 Edge Type

Reference No.	1.3.5.4
Name	Edge Type
Input	Trigger, Tick, Transformed Source
Output	Edge Type Command
Process Description	Define Type for will be created Edge

1.3.5.4 Edge Type




```
void Edge_Type(block *p)
{
        int flag1=0,flag2=0,flag3=0;
        int bind=0;
        int back=0;
        while(p->next[0]!=NULL)
        {
                if((strstr(p->info,"for")!=0)||(strstr(p->info,"while")!=0))
                        flag1=1;
                else if((strstr(p->info,"else")!=0)&&(strlen(p->info)==6))
                        flag2=1;
                else if((strstr(p->info,"switch")!=0))
                        flag3=1;

                if(flag1==1)
                {
                        if(strstr(p->info,"{")!=0)
                                bind++;
                        else if(strstr(p->info,"}")!=0)
                        {
                                bind--;
                                back=1;
                        }
                        else if(strstr(p->info,"for")!=0)
                                bind++;
                }

                if(flag2==1)
                {
                        if(strstr(p->info,"}")!=0)
                        {
                                p->prev->condition=1;
```

```
        int flag1=0,flag2=0,flag3=0;
        int bind=0;
        int back=0;
        while(p->next[0]!=NULL)
        {
                if((strstr(p->info,"for")!=0)||(strstr(p->info,"while")!=0))
                        flag1=1;
                else if((strstr(p->info,"else")!=0)&&(strlen(p->info)==6))
                        flag2=1;
                else if((strstr(p->info,"switch")!=0))
                        flag3=1;

                if(flag1==1)
                {
                        if(strstr(p->info,"{")!=0)
                                bind++;
                        else if(strstr(p->info,"}")!=0)
                        {
                                bind--;
                                back=1;
                        }
                        else if(strstr(p->info,"for")!=0)
                                bind++;
                }

                if(flag2==1)
                {
                        if(strstr(p->info,"}")!=0)
                        {
                                p->prev->condition=1;
                                flag2=0;
                        }
                        else if(strstr(p->info,";")!=0)
                        {
                                p->next[0]->condition=1;
                                flag2=0;
                        }
                }

                if(flag3==1)
                {
                        if(strstr(p->info,"}")!=0)
                        {
                                p->prev->condition=1;
                                flag3=0;
```

```
                        bind++;
                else if(strstr(p->info,"}")!=0)
                {
                        bind--;
                        back=1;
                }
                else if(strstr(p->info,"for")!=0)
                        bind++;
        }

        if(flag2==1)
        {
                if(strstr(p->info,"}")!=0)
                {
                        p->prev->condition=1;
                        flag2=0;
                }
                else if(strstr(p->info,";")!=0)
                {
                        p->next[0]->condition=1;
                        flag2=0;
                }
        }

        if(flag3==1)
        {
                if(strstr(p->info,"}")!=0)
                {
                        p->prev->condition=1;
                        flag3=0;
                }
        }

        if((back==1)&&(bind==1))
        {
                p->prev->edge_type=1;
                flag1=0;
                back=0;
        }
        p= p->next[0];
    }
```

1.3.5.5 Connection

Reference No.	1.3.5.5
Name	Connection
Input	Trigger, Tick, Transformed Source
Output	Connection Command
Process Description	Connect between one Node and the another Node in Type Edge defined

1.3.5.5 Connection


```
void Connection(block *p)
{
if(strstr(p->info,"{")!=0|| strstr(p->info,"}")!=0)
if((strstr(p->info,"for")!=0)||(strstr(p->info,"while")!=0))
    if((strncmp(p->info,"if",2)==0))
      if((strstr(p->info,"switch")!=0))
        if((strstr(p->info,"else if")!=0))
        if((strstr(p->info,"else")!=0)&&(strlen(p->info)==6))
          if(strstr(p->info,"case")!=0)
            if(strstr(p->info,"break")!=0)
                p->visit++;
                p= p->next[flag];
```

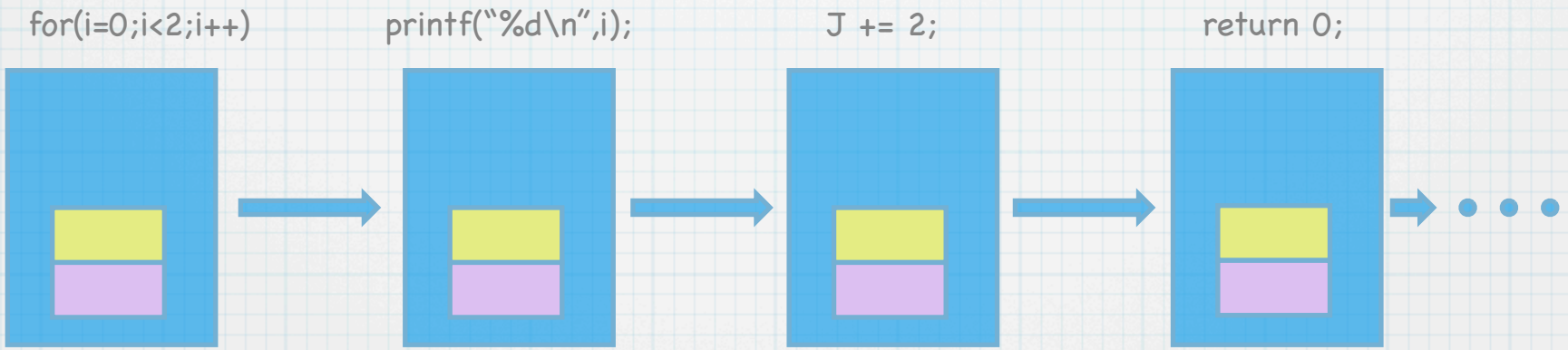
1.3.5.5 Connection Instruction

```
Code
for(i=0;i<2;i++)
{
printf(%d\n",i);
j += 2;
}
return 0;
```

In case of (For, While)

Where the address points to Next block

Where the address points to Another block



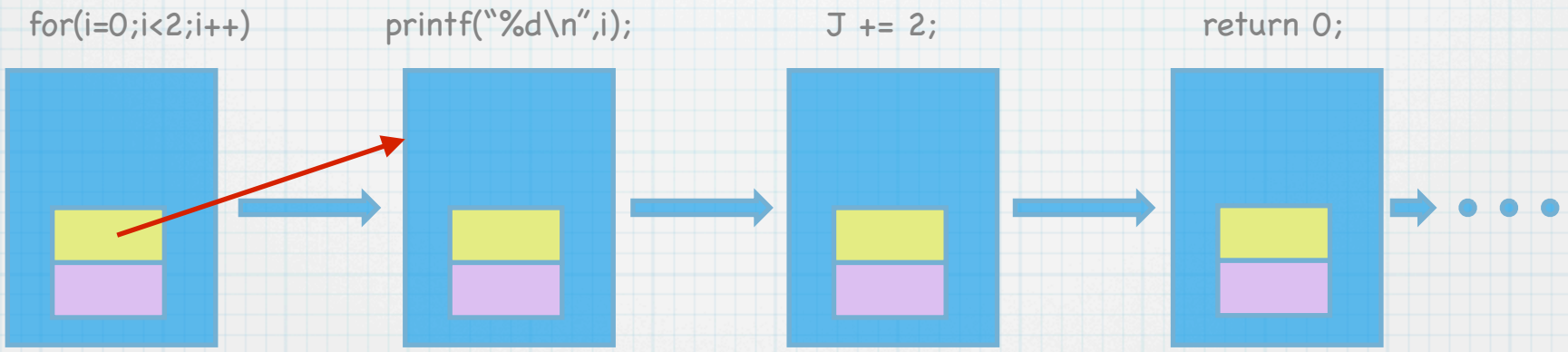
1.3.5.5 Connection Instruction

```
Code
for(i=0;i<2;i++)
{
printf(%d\n",i);
j += 2;
}
return 0;
```

In case of (For, While)

Where the address points to Next block

Where the address points to Another block



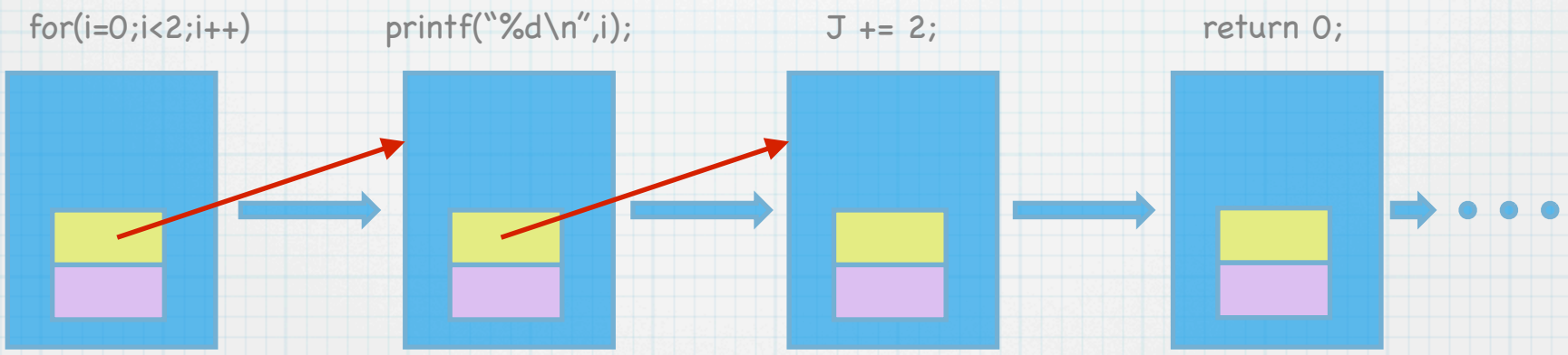
1.3.5.5 Connection Instruction

```
Code
for(i=0;i<2;i++)
{
printf(%d\n",i);
j += 2;
}
return 0;
```

In case of (For, While)

Where the address points to Next block

Where the address points to Another block



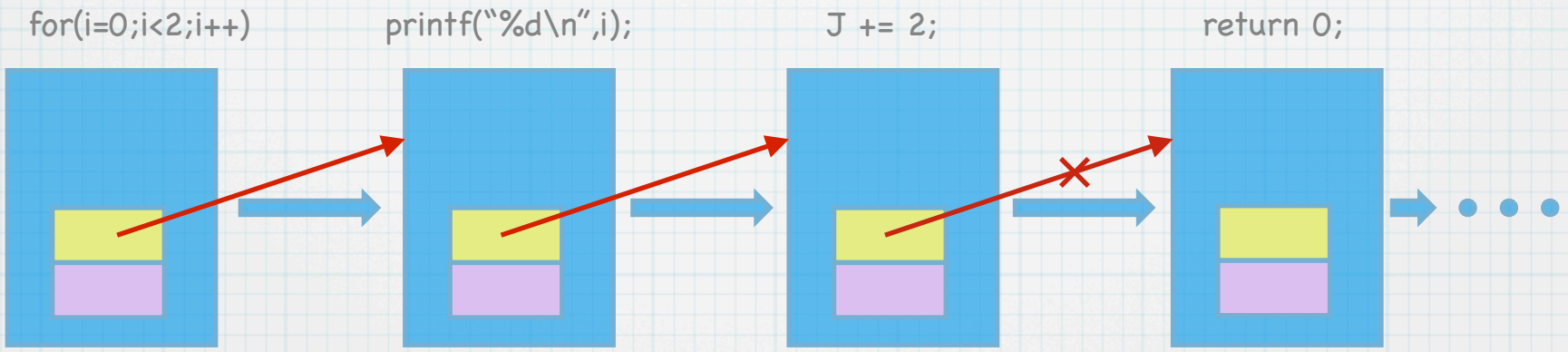
1.3.5.5 Connection Instruction

```
Code
for(i=0;i<2;i++)
{
printf(%d\n",i);
j += 2;
}
return 0;
```

In case of (For, While)

Where the address points to Next block

Where the address points to Another block



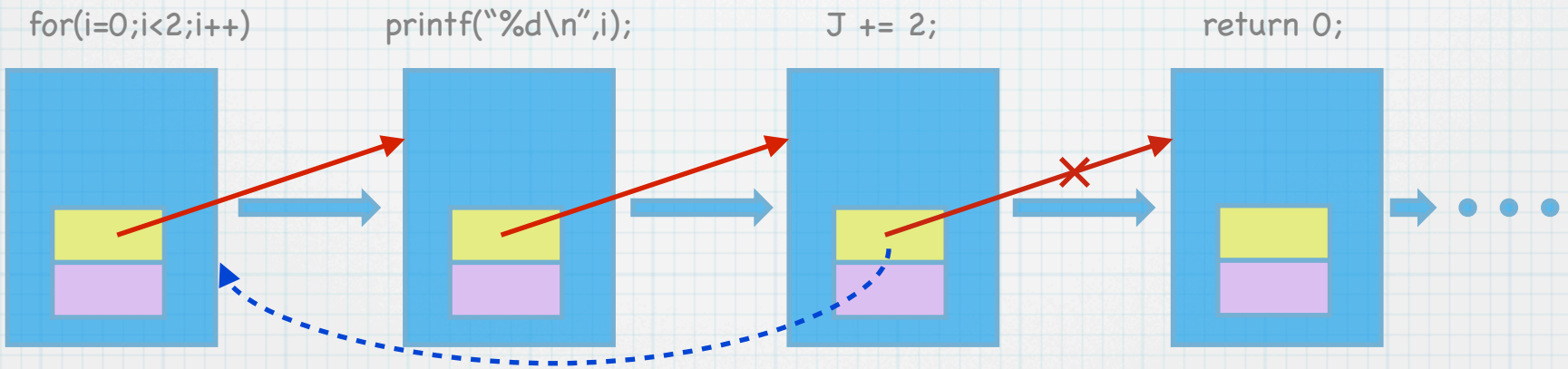
1.3.5.5 Connection Instruction

```
Code
for(i=0;i<2;i++)
{
printf(%d\n",i);
j += 2;
}
return 0;
```

In case of (For, While)

Where the address points to Next block

Where the address points to Another block



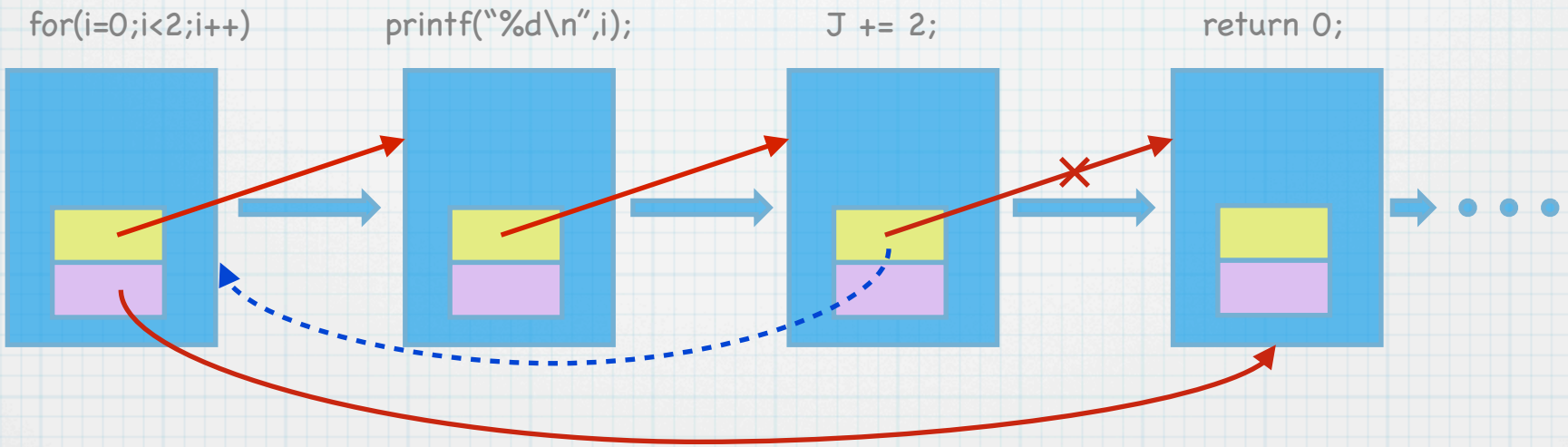
1.3.5.5 Connection Instruction

```
Code
for(i=0;i<2;i++)
{
printf(%d\n",i);
j += 2;
}
return 0;
```

In case of (For, While)

Where the address points to Next block

Where the address points to Another block



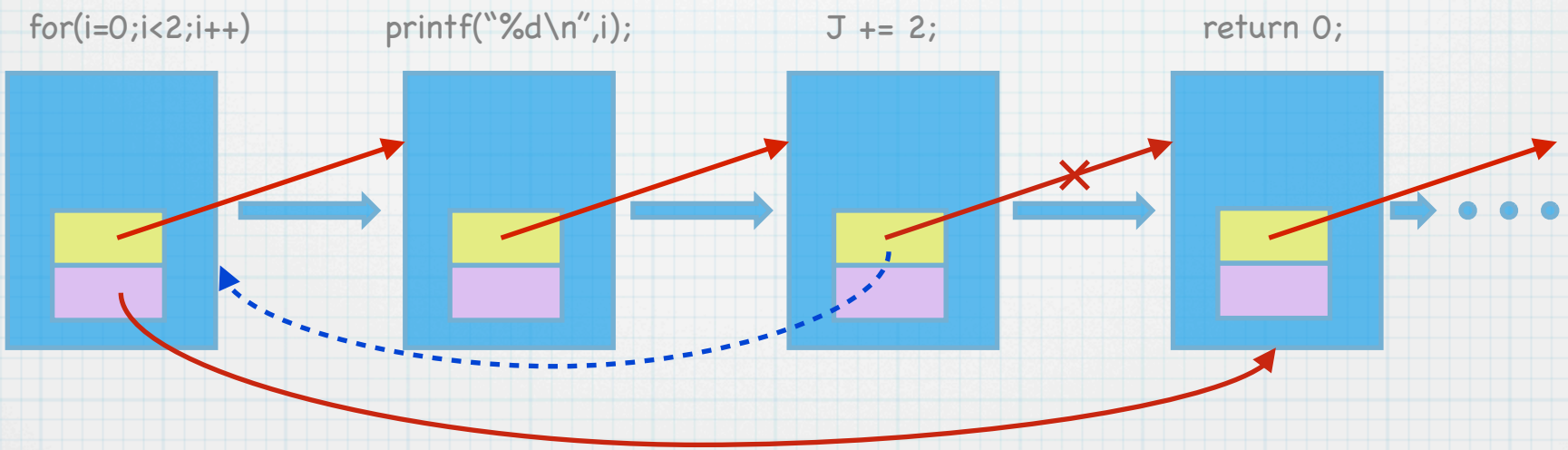
1.3.5.5 Connection Instruction

```
Code
for(i=0;i<2;i++)
{
printf(%d\n",i);
j += 2;
}
return 0;
```

In case of (For, While)

Where the address points to Next block

Where the address points to Another block



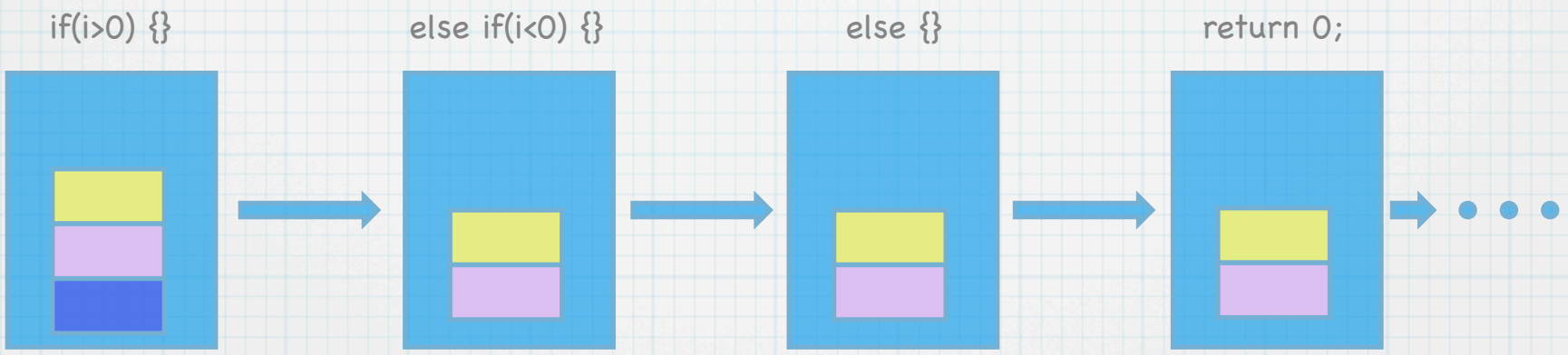
1.3.5.5 Connection Instruction

```
Code
if(i>0) {}
else if(i<0) {}
else {}
return 0;
```

In case of (If, Switch)

Where the address points to Next block

Where the address points to Another block



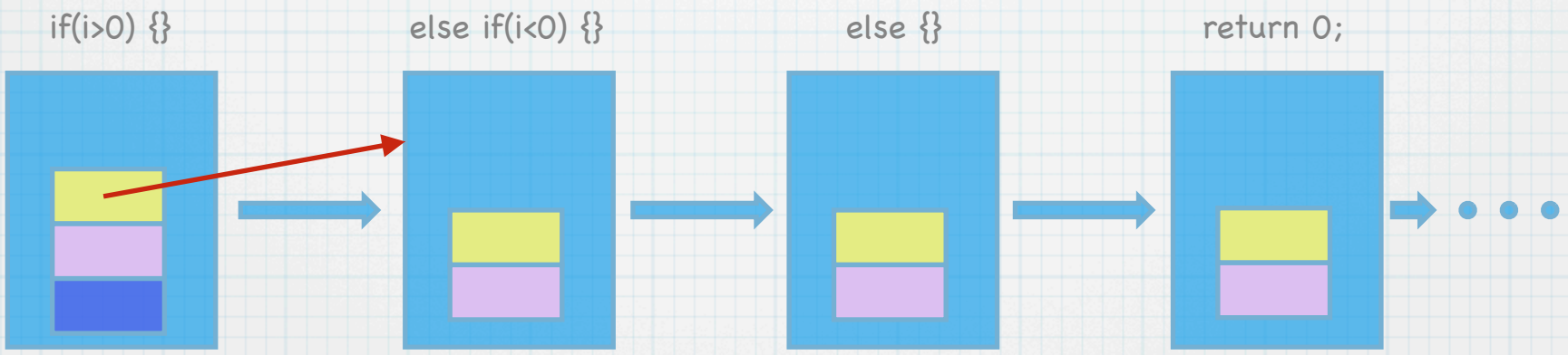
1.3.5.5 Connection Instruction

```
Code
if(i>0) {}
else if(i<0) {}
else {}
return 0;
```

In case of (If, Switch)

Where the address points to Next block

Where the address points to Another block



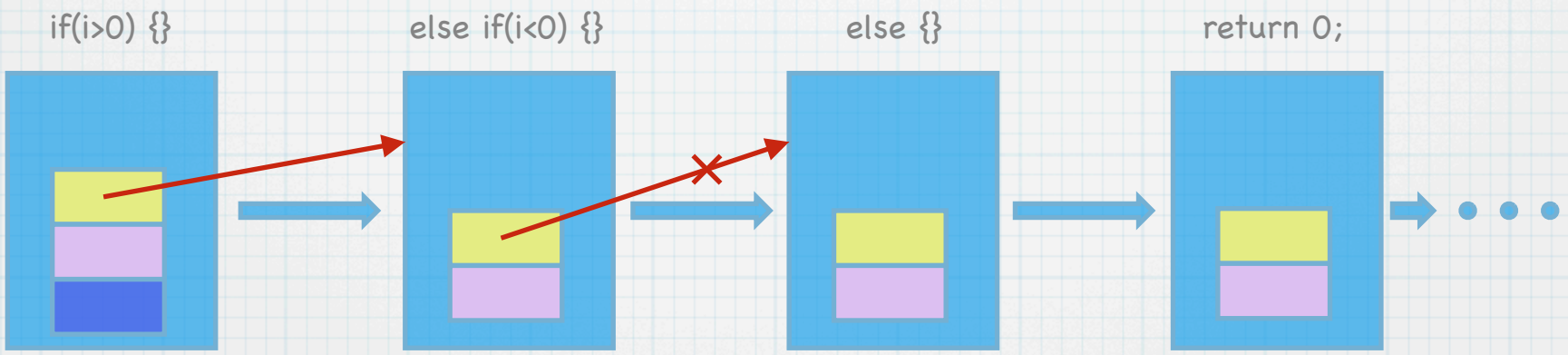
1.3.5.5 Connection Instruction

```
Code
if(i>0) {}
else if(i<0) {}
else {}
return 0;
```

In case of (If, Switch)

Where the address points to Next block

Where the address points to Another block



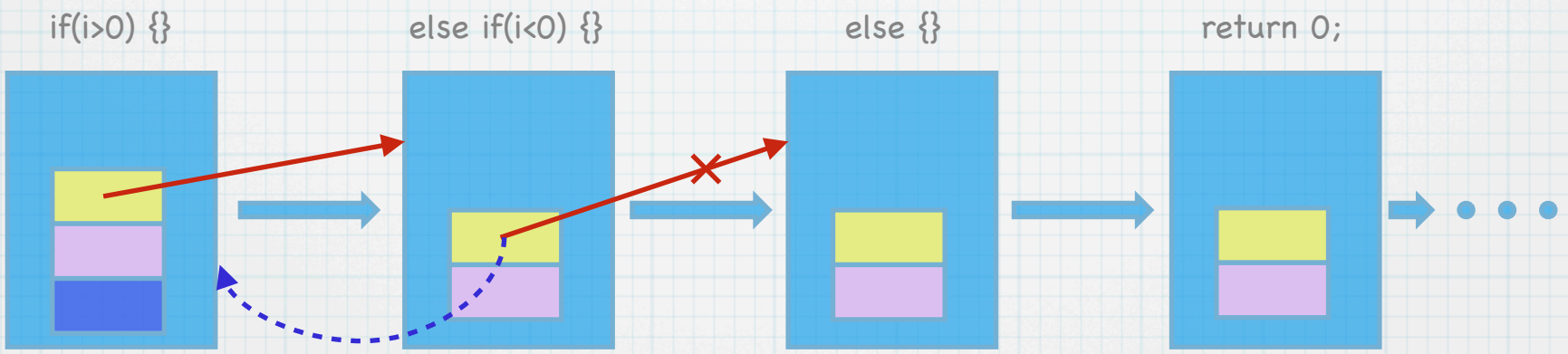
1.3.5.5 Connection Instruction

```
Code
if(i>0) {}
else if(i<0) {}
else {}
return 0;
```

In case of (If, Switch)

Where the address points to Next block

Where the address points to Another block



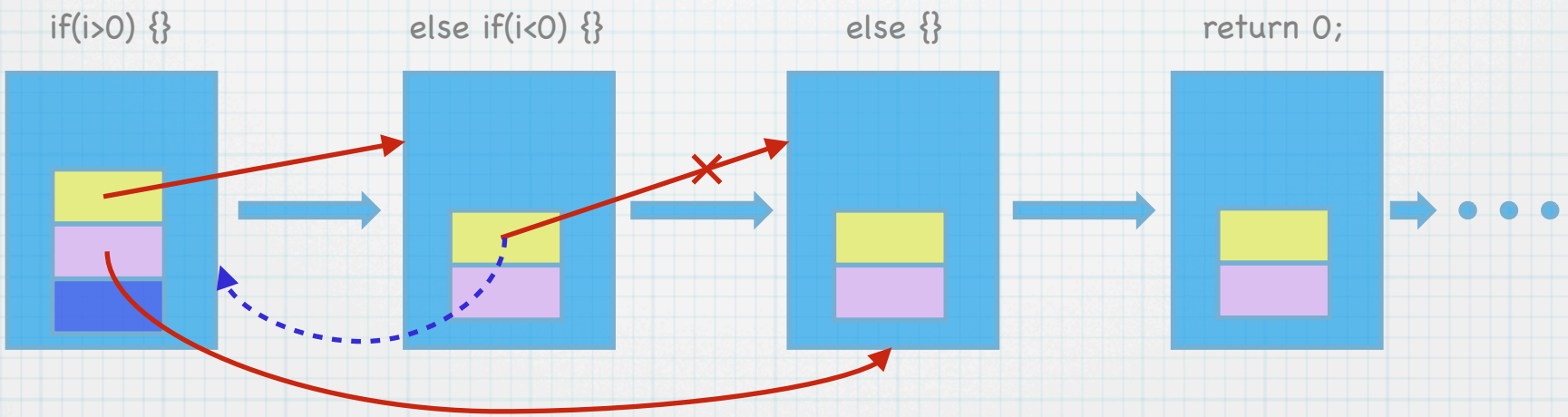
1.3.5.5 Connection Instruction

```
Code
if(i>0) {}
else if(i<0) {}
else {}
return 0;
```

In case of (If, Switch)

Where the address points to Next block

Where the address points to Another block



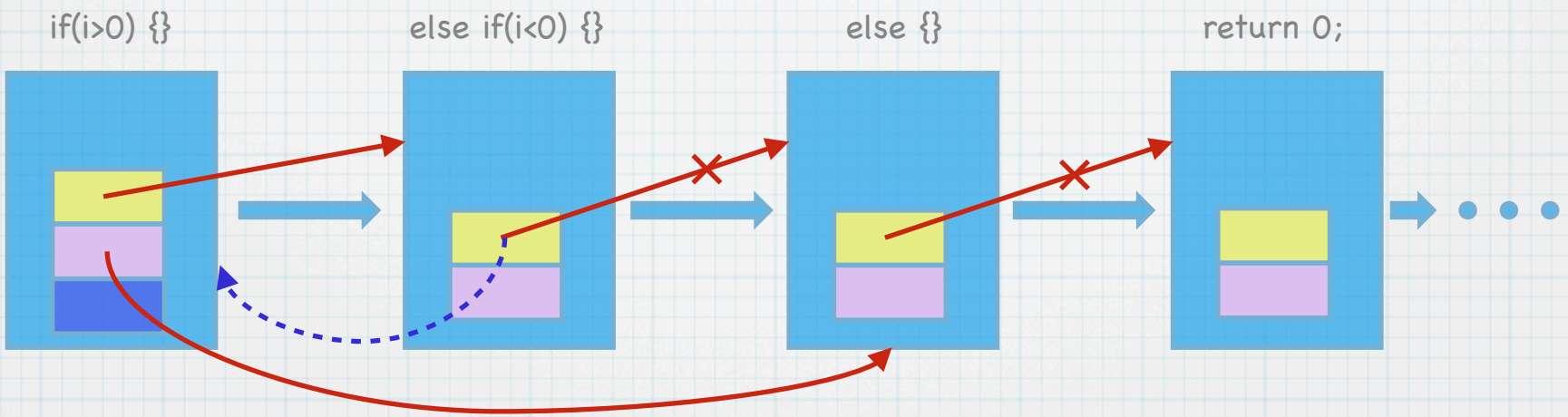
1.3.5.5 Connection Instruction

```
Code
if(i>0) {}
else if(i<0) {}
else {}
return 0;
```

In case of (If, Switch)

Where the address points to Next block

Where the address points to Another block



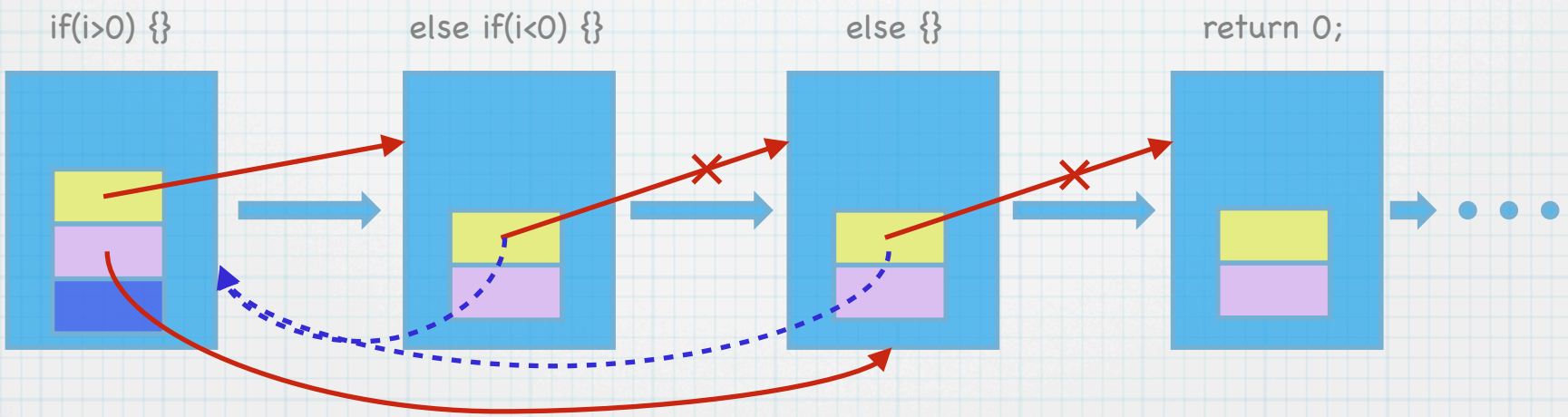
1.3.5.5 Connection Instruction

```
Code
if(i>0) {}
else if(i<0) {}
else {}
return 0;
```

In case of (If, Switch)

Where the address points to Next block

Where the address points to Another block



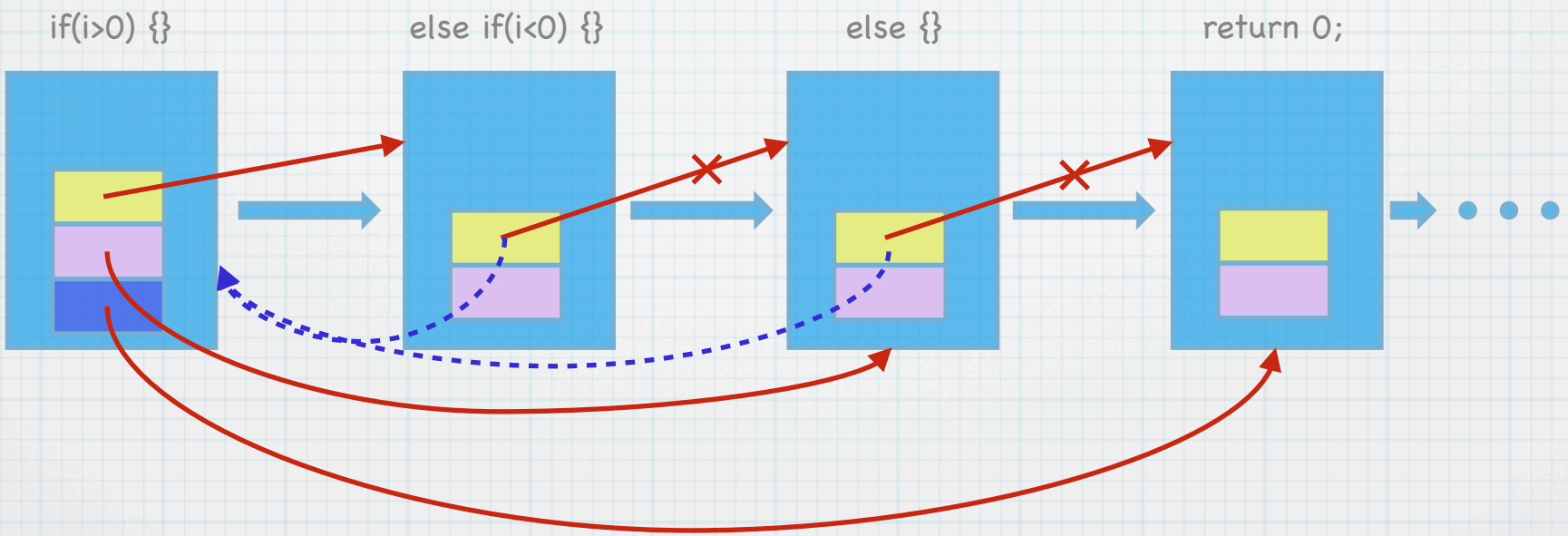
1.3.5.5 Connection Instruction

```
Code
if(i>0) {}
else if(i<0) {}
else {}
return 0;
```

In case of (If, Switch)

Where the address points to Next block

Where the address points to Another block



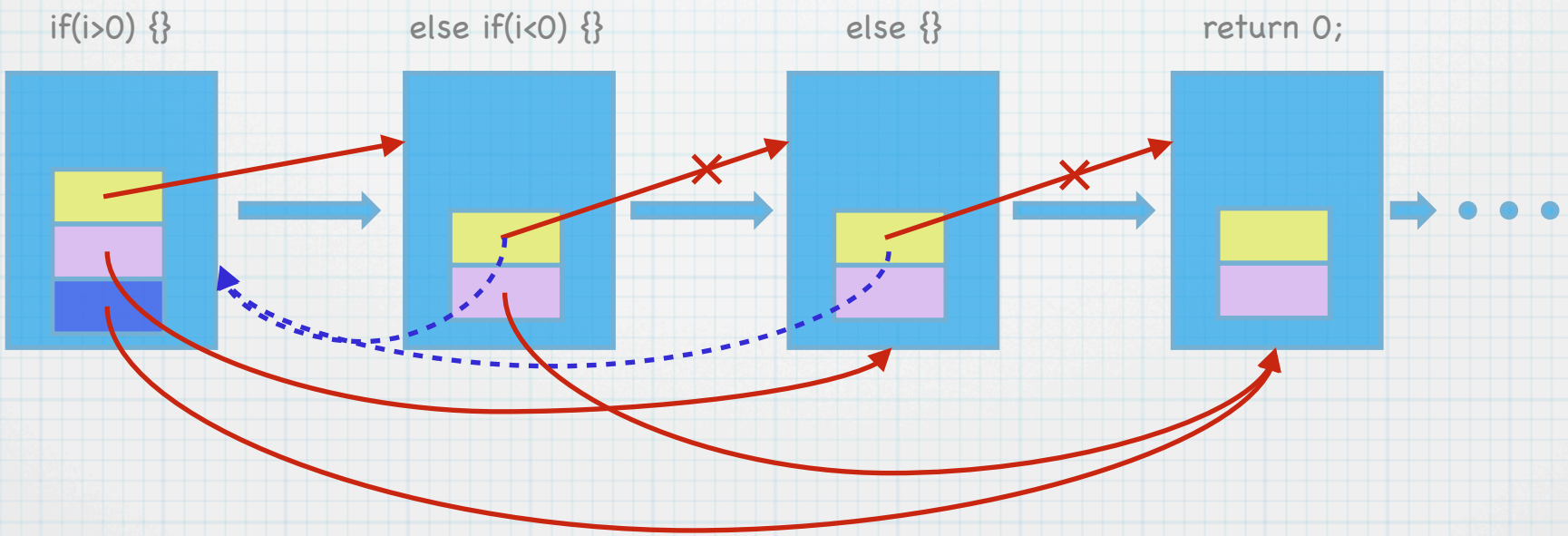
1.3.5.5 Connection Instruction

```
Code
if(i>0) {}
else if(i<0) {}
else {}
return 0;
```

In case of (If, Switch)

Where the address points to Next block

Where the address points to Another block



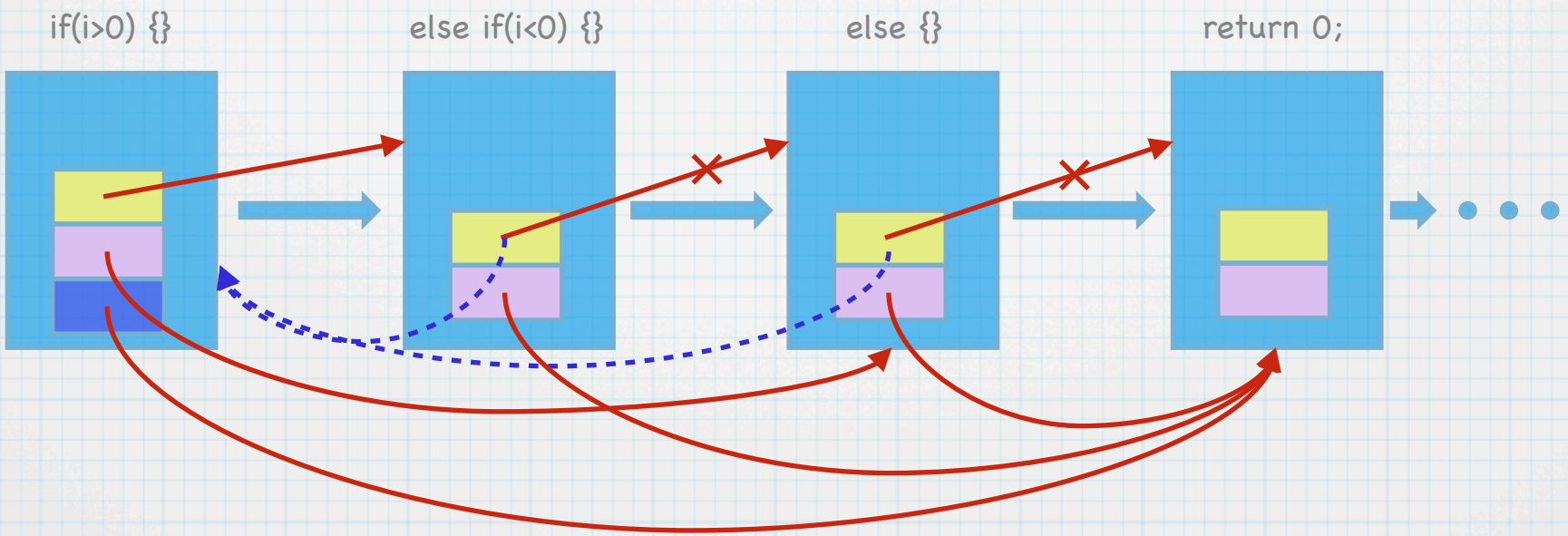
1.3.5.5 Connection Instruction

```
Code
if(i>0) {}
else if(i<0) {}
else {}
return 0;
```

In case of (If, Switch)

Where the address points to Next block

Where the address points to Another block



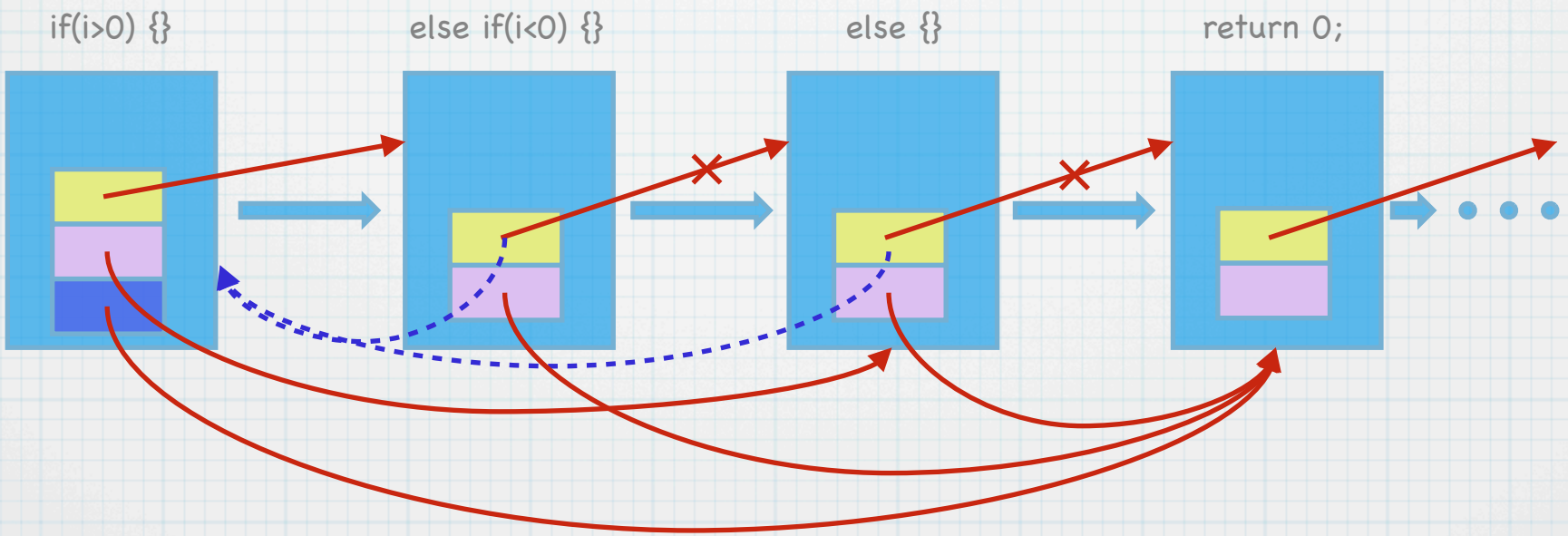
1.3.5.5 Connection Instruction

```
Code
if(i>0) {}
else if(i<0) {}
else {}
return 0;
```

In case of (If, Switch)

Where the address points to Next block

Where the address points to Another block



Find Function

Name	Find Function
Input	Trigger, Tick, Transformed Source
Output	Find Function Command
Process Description	After Function Block is found, it's address is sent to CFG Info

```
int Find_Function(char* list[], int line, block *p , char* function_name[])
{
        int flag = 0;
        int a = 0;
        int i = 0;
        int j, k;
        char buffer[100] = {' '};
        for(i = 0; i<line; i++)
        {
                k = 0;
                j = 0;
                if(flag == 0)
                {
                        if(strstr(list[i], ");") != 0)
                        {
                                function_name[a]= (char*)malloc(sizeof(char)*100);

                                while(*(*(list+i)+j) == ' ')
                                        j++;
                                while(*(*(list+i)+j) != ' ')
                                        j++;
                                j++;
                                while(*(*(list+i)+j)!='(')
                                {
                                        buffer[k] = *(*(list+i)+j);
                                        j++;
                                        k++;
                                }

                                strcpy(function_name[a], buffer);

                                a++;
                        }
                }
                if(strstr(list[i],"main") != 0)
                {
                        flag++;
                }
        }
        return a--;
```

1.3.7 CFG Phasing Display

Reference No.	1.3.7
Name	CFG Phasing Display
Input	CFG Command
Output	CFG Phase
Process Description	Send CFG converting process in String Type to Message Interface.

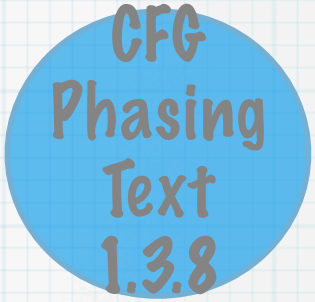
1.3.7 CFG Phasing Display


```
void CFG_Phasing_Display(block *p, int fnum, char* function_name[])
{
        block *temp,*q;
        int i,n=1,k=0;
        int flag1=0, flag2=0;
        int check=1,check_1=0;
while(strstr(p->info,"return 0")==0)
{

        if(p->block_num > 0)
        {
                if((strstr(p->info,"for")!=0)||(strstr(p->info,"while")!=0))
                {
else if((strstr(p->next[0]->info,"else if")!=0)||(strstr(p->next[0]->info,"else")!=0))
{
else if((strstr(p->info,"switch")!=0))
```

1.3.7 CFG Phasing Display

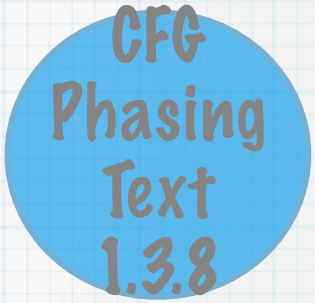

```
printf("Block Infomation: %s",p->info);
for(i = 0;i<fnum;i++)
{
        if(strstr(p->info, function_name[i]))
                printf("                    <Function>\n");
}
if(p->block_type==0)
{       printf("\nBlock Type:   ___\n");
        printf("              |   |\n");
        printf("              |___|\n");


}
else{
        printf("\nBlock Type:    /\\ \n");
        printf("              /  \\ \n ");
        printf("             \\   / \n");
        printf("              \\/  \n");
}

if(p->edge_type==0)
        printf("Edge Type: forward_edge\n");
else
        printf("Edge Type: back_edge\n");
printf("\n");
```

1.3.8 CFG Phasing Text



Reference No.	1.3.8
Name	CFG Phasing Text
Input	CFG Command
Output	CFG Phase
Process Description	Send CFG converting process in String Type to CFG File Interface.

1.3.8 CFG Phasing Text




```
void CFG_Phasing_Text(block *p, int fnum, char* function_name[])
{
        block *temp,*q;
        int i,n=1,k=0;
        int flag1=0, flag2=0;
        int check=1,check_1=0;

        FILE *fout;
                fout=fopen("Result.txt","w");

    fprintf(fout,"****** Main Fuction ******\n\n");

        while(p->next[0]!=NULL)
        {
        if((strstr(p->info,"return 0")!=0))
          fprintf(fout,"****** User Fuction ******\n\n");
if(p->block_type==0)
        fprintf(fout,"Block Type: ?n");
else
        fprintf(fout,"Block Type: 옜\n");
```

THE END