

---

# **CMPEN 431**

## **Computer Architecture**

### **Fall 2017**

## **Understanding Program Dependencies**

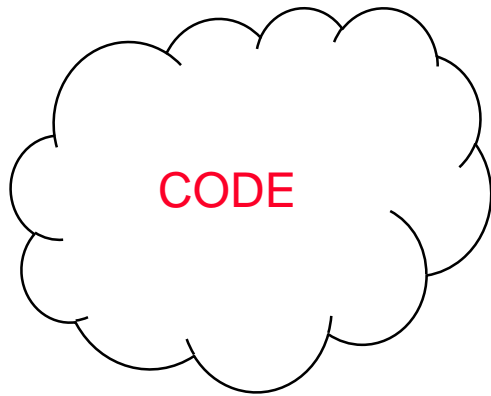
Mahmut Taylan Kandemir  
([www.cse.psu.edu/~kandemir](http://www.cse.psu.edu/~kandemir))

ACK= Jack Sampson

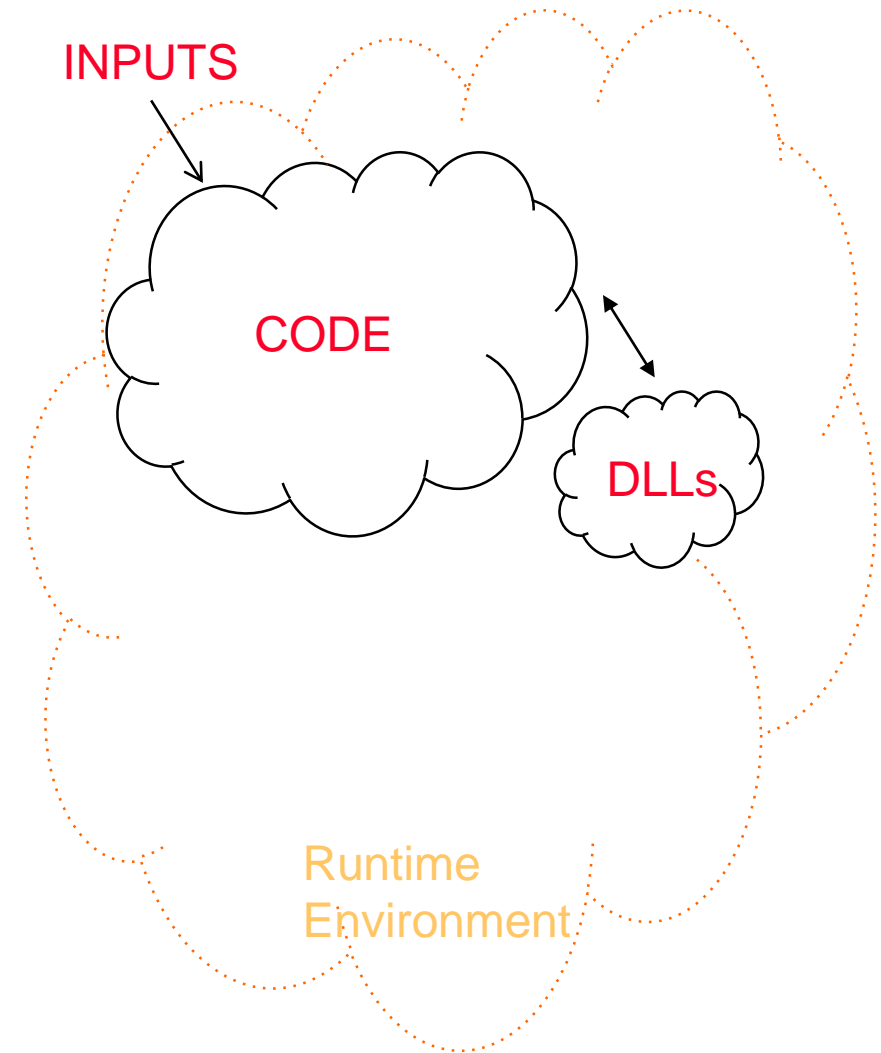
# Hierarchy of structures within a program

---

❑ Static

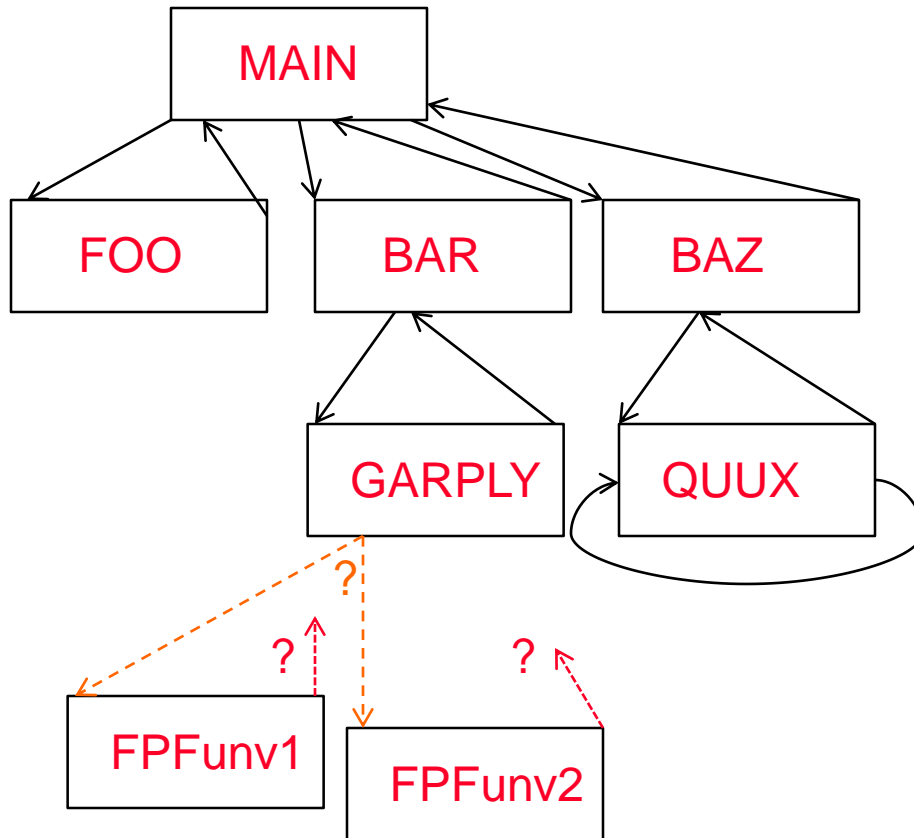


❑ Dynamic

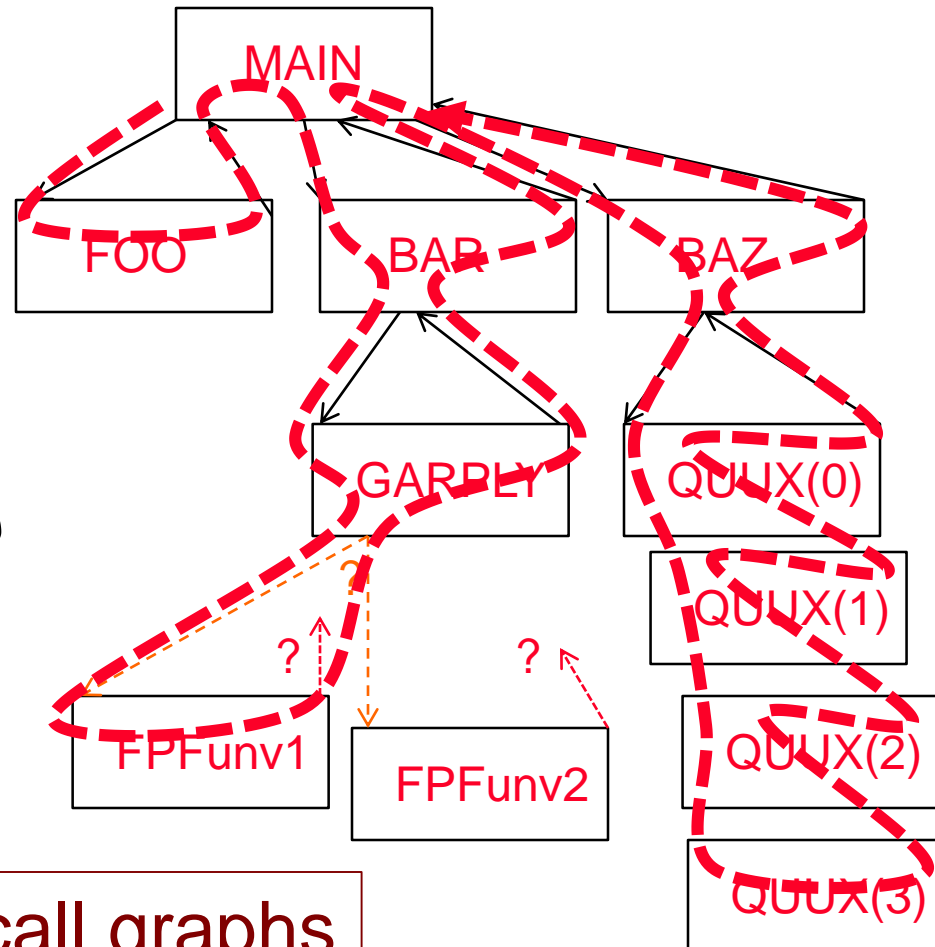


# Hierarchy of structures within a program

❑ Static



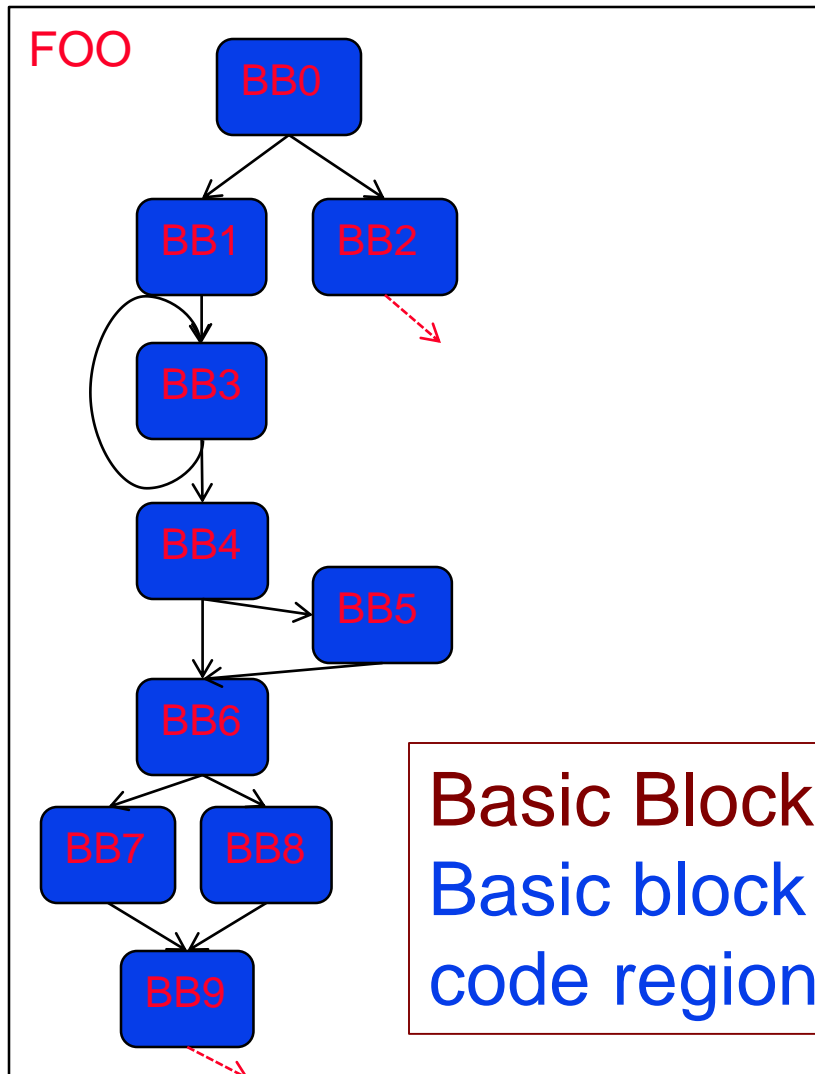
❑ Dynamic



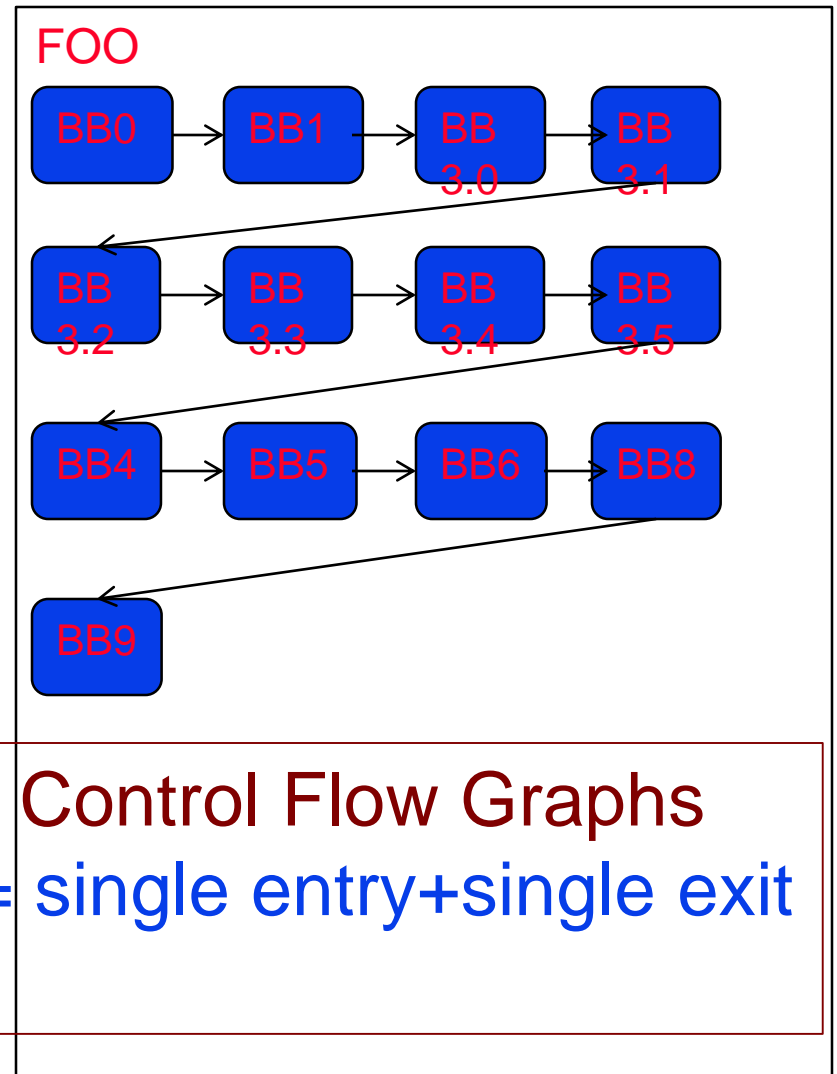
Function call graphs

# Hierarchy of structures within a program

## ❑ Static



## ❑ Dynamic



Basic Block  
Basic block =  
code region

Control Flow Graphs  
single entry+single exit

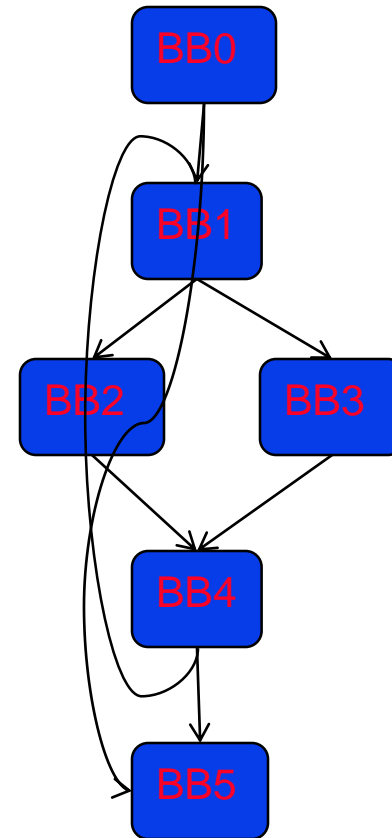
# From Code to Structure

---

## ❑ Code

```
for (i=0;i<4;++i){  
    if(a[i]<b[i])  
        ++c1;  
    else  
        ++c2;  
    c3+=a[i];  
}  
c4=c3+c1-c2;
```

## ❑ Control Flow Graph

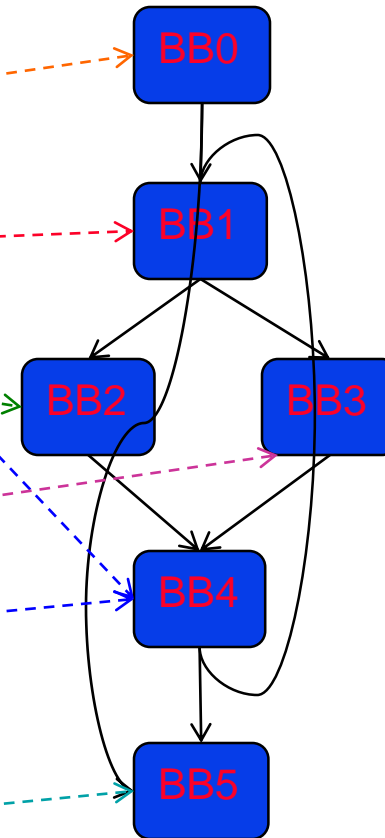


# From Code to Structure

❑ Code

```
for (i=0; i<4; ++i){  
    if(a[i]<b[i])  
        ++c1;  
    else  
        ++c2;  
    c3+=a[i];  
}  
c4=c3+c1-c2;
```

❑ Control Flow Graph

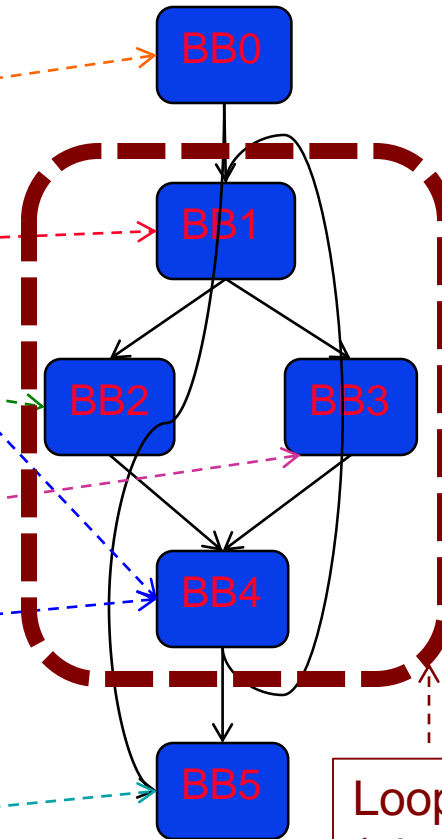


# From Code to Structure

## Code

```
for (i=0; i<4; ++i){  
    if(a[i]<b[i])  
        ++c1;  
    else  
        ++c2;  
    c3+=a[i];  
}  
c4=c3+c1-c2;
```

## Control Flow Graph



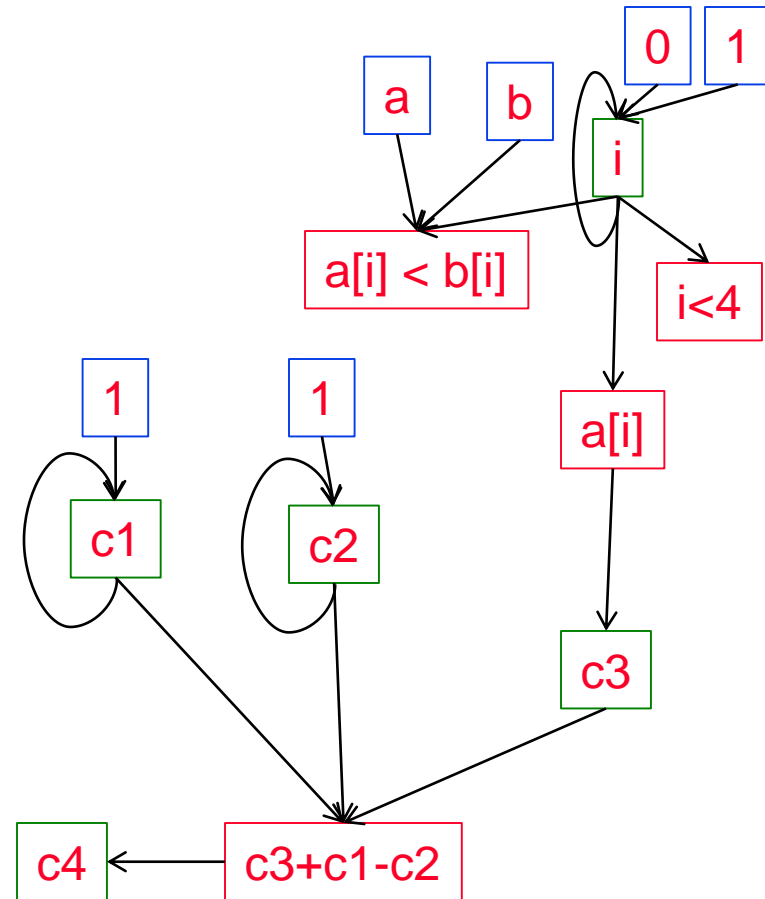
Loop body  
(also happens to be  
a “hyperblock”: a  
single entry multiple  
exit forward flow set of  
BBs)

# From Code to Structure

## ❑ Code

```
for (i=0;i<4;++i){  
    if(a[i]<b[i])  
        ++c1;  
    else  
        ++c2;  
    c3+=a[i];  
}  
c4=c3+c1-c2;
```

## ❑ Data Dependence Graph





# From Code to Structure

---

## ❑ Code

```
for (i=0;i<4;++i){  
    if(a[i]<b[i])  
        ++c1;  
    else  
        ++c2;  
    c3+=a[i];  
}  
c4=c3+c1-c2;
```

## ❑ Program Dependence

mov i, 0 // check statically elided

LOOP:

getelementptr tmp1, a, i

getelementptr tmp2, b, i

lw tmp1, tmp1

lw tmp2, tmp2

slt tmp2, tmp1, tmp2

bnez tmp2, ELSE

add c1, c1, 1

j JOIN

ELSE: add c2, c2, 1

JOIN: add c3, c3, tmp1

add i, i, 1

slt tmp1, i, 4

bnez tmp1, LOOP

END: sub tmp1, c1, c2

add c4, tmp1, c3

# From Code to Structure

## ❑ Code

```
for (i=0;i<4;++i){  
    if(a[i]<b[i])  
        ++c1;  
    else  
        ++c2;  
    c3+=a[i];  
}  
c4=c3+c1-c2;
```

## ❑ Program Dependence

```
mov i, 0 // check statically elided  
LOOP:  
getelementptr tmp1, a, i  
getelementptr tmp2, b, i  
lw tmp1, tmp1  
lw tmp2, tmp2  
slt tmp2, tmp1, tmp2  
bnez tmp2, ELSE  
add c1, c1, 1  
j JOIN  
ELSE: add c2, c2, 1  
JOIN: add c3, c3, tmp1  
add i, i, 1  
slt tmp1, i, 4  
bnez tmp1, LOOP  
END: sub tmp1, c1, c2  
add c4, tmp1, c3
```

```
graph TD  
    Start(( )) --> MOV[mov i, 0] --> LOOP[LOOP:]  
    LOOP --> GET1[getelementptr tmp1, a, i] --> GET2[getelementptr tmp2, b, i]  
    GET1 --> LW1[lw tmp1, tmp1] --> LW2[lw tmp2, tmp2]  
    GET2 --> LW1  
    GET2 --> LW2  
    LW1 --> SLT[slt tmp2, tmp1, tmp2]  
    LW2 --> SLT  
    SLT --> BNEZ[bnez tmp2, ELSE] --> ADDC1[add c1, c1, 1]  
    SLT --> ADDC1  
    ADDC1 --> JJOIN[j JOIN]  
    BNEZ --> ELSE[ELSE: add c2, c2, 1]  
    JJOIN --> JOIN[JOIN: add c3, c3, tmp1]  
    ELSE --> JOIN  
    JOIN --> ADDI[add i, i, 1] --> SLTI[slt tmp1, i, 4]  
    SLTI --> BNEZ_LOOP[bnez tmp1, LOOP] --> LOOP  
    SLTI --> END[END: sub tmp1, c1, c2]  
    BNEZ_LOOP --> LOOP  
    END --> ADDC4[add c4, tmp1, c3]
```

RAW

# From Code to Structure

## ❑ Code

```
for (i=0;i<4;++i){  
    if(a[i]<b[i])  
        ++c1;  
    else  
        ++c2;  
    c3+=a[i];  
}  
c4=c3+c1-c2;
```

## ❑ Program Dependence

```
mov i, 0 // check statically elided  
LOOP:  
    getelementptr tmp1, a, i  
    getelementptr tmp2, b, i  
    lw tmp1, tmp1  
    lw tmp2, tmp2  
    slt tmp2, tmp1, tmp2  
    bnez tmp2, ELSE  
    add c1, c1, 1  
    j JOIN  
ELSE: add c2, c2, 1  
JOIN: add c3, c3, tmp1  
    add i, i, 1  
    slt tmp1, i, 4  
    bnez tmp1, LOOP  
END: sub tmp1, c1, c2  
    add c4, tmp1, c3
```

RAW  
WAW

# From Code to Structure

## ❑ Code

```
for (i=0;i<4;++i){  
    if(a[i]<b[i])  
        ++c1;  
    else  
        ++c2;  
    c3+=a[i];  
}  
c4=c3+c1-c2;
```

## ❑ Program Dependence

```
mov i, 0 // check statically elided  
LOOP:  
    getelementptr tmp1, a, i  
    getelementptr tmp2, b, i  
    lw tmp1, tmp1  
    lw tmp2, tmp2  
    slt tmp2, tmp1, tmp2  
    bnez tmp2, ELSE  
    add c1, c1, 1  
    j JOIN  
ELSE: add c2, c2, 1  
JOIN: add c3, c3, tmp1  
    add i, i, 1  
    slt tmp1, i, 4  
    bnez tmp1, LOOP  
END: sub tmp1, c1, c2  
    add c4, tmp1, c3
```

RAW  
WAW  
WAR

# From Code to Structure

---

## ❑ Code

```
for (i=0;i<4;++i){  
    if(a[i]<b[i])  
        ++c1;  
    else  
        ++c2;  
    c3+=a[i];  
}  
c4=c3+c1-c2;
```

## ❑ Program Dependence

mov i, 0 // check statically elided

LOOP: **phi i1, i, i1, BB0, BB4**

getelementptr tmp1, a, i1

getelementptr tmp2, b, i1

lw tmp3, tmp1

lw tmp4, tmp2

slt tmp5, tmp3, tmp4

bnez tmp5, ELSE

add c1, c1, 1

j JOIN

ELSE: add c2, c2, 1

JOIN: add c3, c3, tmp3

add i1, i1, 1

slt tmp6, i1, 4

bnez tmp6, LOOP

END: sub tmp7, c1, c2

add c4, tmp7, c3

# From Code to Structure

## □ Code

```
for (i=0;i<4;++i){  
    if(a[i]<b[i])  
        ++c1;  
    else  
        ++c2;  
    c3+=a[i];  
}  
c4=c3+c1-c2;
```

## □ Program Dependence

mov i, 0 // check statically elided

LOOP: **phi i1, i, i1, BB0, BB4**

getelementptr tmp1, a, i1

getelementptr tmp2, b, i1

lw tmp3, tmp1

lw tmp4, tmp2

slt tmp5, tmp3, tmp4

bnez tmp5, ELSE

add c1, c1, 1

j JOIN

ELSE: add c2, c2, 1

JOIN: add c3, c3, tmp3

add i1, i1, 1

slt tmp6, i1, 4

bnez tmp6, LOOP

END: sub tmp7, c1, c2

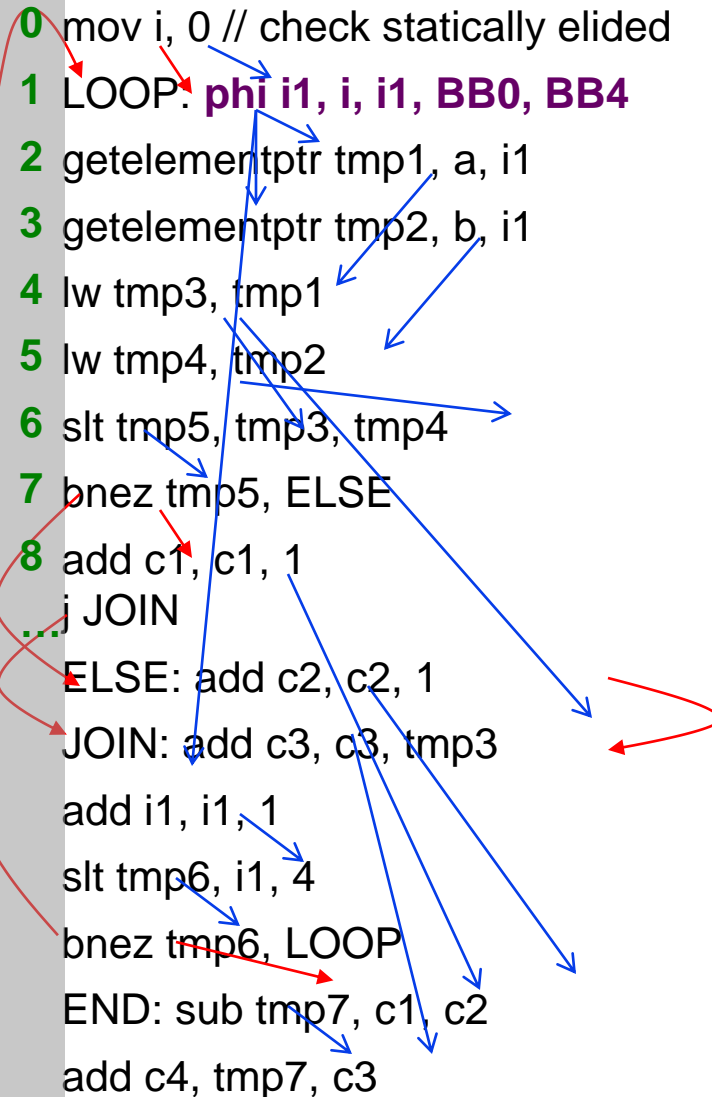
add c4, tmp7, c3

# From Code to Structure

## □ Code

```
for (i=0;i<4;++i){  
    if(a[i]<b[i])  
        ++c1;  
    else  
        ++c2;  
    c3+=a[i];  
}  
c4=c3+c1-c2;
```

## □ Program Dependence



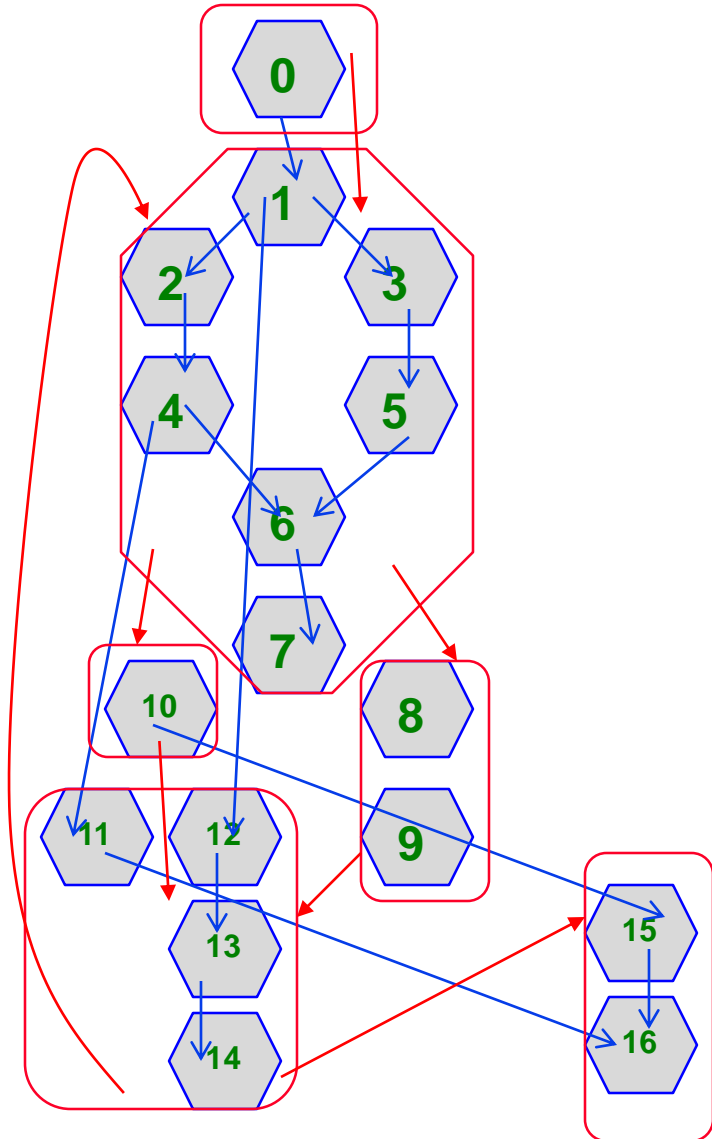
The control flow graph (CFG) for the code block is as follows:

- 0** `mov i, 0 // check statically elided`
- 1** `LOOP: phi i1, i, i1, BB0, BB4`
- 2** `getelementptr tmp1, a, i1`
- 3** `getelementptr tmp2, b, i1`
- 4** `lw tmp3, tmp1`
- 5** `lw tmp4, tmp2`
- 6** `slt tmp5, tmp3, tmp4`
- 7** `bnez tmp5, ELSE`
- 8** `add c1, c1, 1`
- ...** `JOIN`
- ELSE:** `add c2, c2, 1`
- JOIN:** `add c3, c3, tmp3`
- `add i1, i1, 1`
- `slt tmp6, i1, 4`
- `bnez tmp6, LOOP`
- END:** `sub tmp7, c1, c2`
- `add c4, tmp7, c3`

Control flow edges are indicated by blue arrows: from 0 to 1, 1 to 2, 2 to 3, 3 to 4, 4 to 5, 5 to 6, 6 to 7, 7 to 8, 8 to JOIN, JOIN to ELSE, ELSE to JOIN, JOIN to add i1, add i1 to slt tmp6, slt tmp6 to bnez tmp6, bnez tmp6 to LOOP, LOOP to 1, and from JOIN to END. A red arrow indicates a loop from LOOP back to 1. A red arrow also points from the JOIN block to the END block.

# From Code to Structure

❑ Program Dependence Graph    ❑ Program Dependence



```
0 mov i, 0 // check statically elided
```

1 LOOP: phi i1, i, i1, BB0, BB4

```
2 getelementptr tmp1, a, i1
```

```
3 getelementptr tmp2, b, i1
```

```
4 lw tmp3, tmp1
```

```
5 lw tmp4, tmp2
```

```
6  slt tmp5, tmp3, tmp4
```

7 bnez tmp5, ELSE

8 add c1, c1, 1

9 JOIN

10 ELSE: add c2, c2, 1

11 JOIN: add c3, c3, tmp3

```
12 add i1, i1, 1
```

```
13 slt tmp6, i1, 4
```

14 bnez tmp6, LOOP

15 END: sub tmp7, c1, c2

16 add c4, tmp7, c3



# So What?

---

- ❑ View seen by compiler
  - Helps understand what optimizations are/are not possible and when
  - Helps understand what information is lost during code generation
  
- ❑ Limitations of PDGs are targets for optimizations
  - Control dependence → branch prediction, dynamic unrolling, predication
  - WAR/WAW → hardware renaming
  - Memory dependence ambiguity → speculative loads
  - RAW → Scheduling, latency hiding, value prediction
  - Limited Von Neumann parallelism / parallelism information → dynamic scheduling, dynamic unrolling, dynamic PDG reconstruction
  
- ❑ Important to understand static vs. dynamic view of execution
  - Possible relationships / always relationships vs. this time relationships
  - Different analysis / optimizations possible