Workshop on Essential Abstractions in GCC

Gray Box Probing of GCC

GCC Resource Center

(www.cse.iitb.ac.in/grc)

Department of Computer Science and Engineering,

Indian Institute of Technology, Bombay

29 June 2013

Outline

- Introduction to Graybox Probing of GCC
- Examining AST for C
- Examining GIMPLE Dumps for C
  - Translation of data accesses
  - Translation of intraprocedural control flow
  - Translation of interprocedural control flow
- Examining GIMPLE Dumps for C++
- Examining RTL Dumps

  (Will be covered later)
- Examining Assembly Dumps
- Examining GIMPLE Optimizations
- Conclusions

Part 1

Preliminaries

What is Gray Box Probing of GCC?

- Black Box probing:

  Examining only the input and output relationship of a system

- White Box probing:

  Examining internals of a system for a given set of inputs

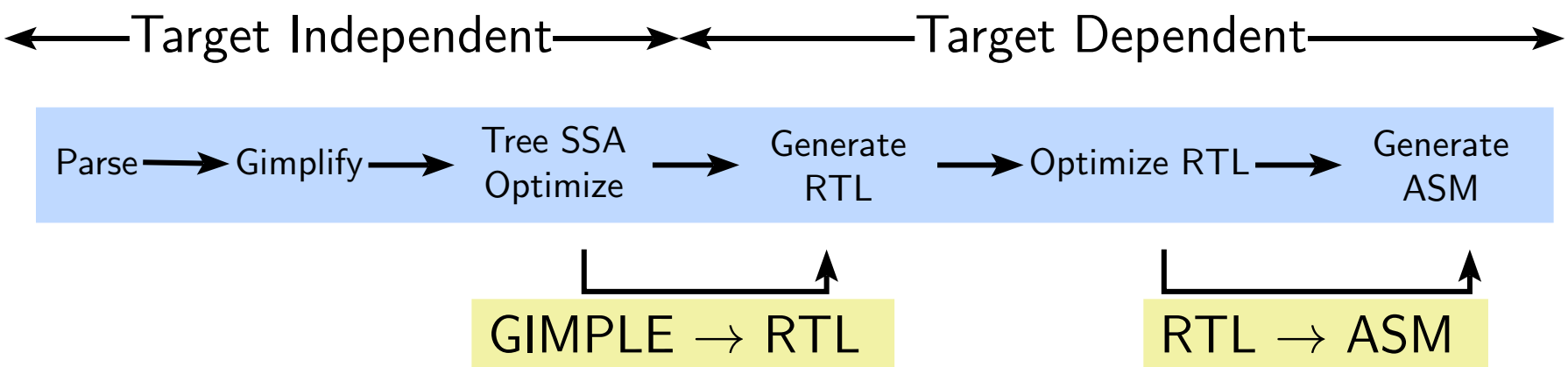
- Gray Box probing:

  Examining input and output of various components/modules

  - Overview of translation sequence in GCC
  - Overview of intermediate representations
  - Intermediate representations of programs across important phases

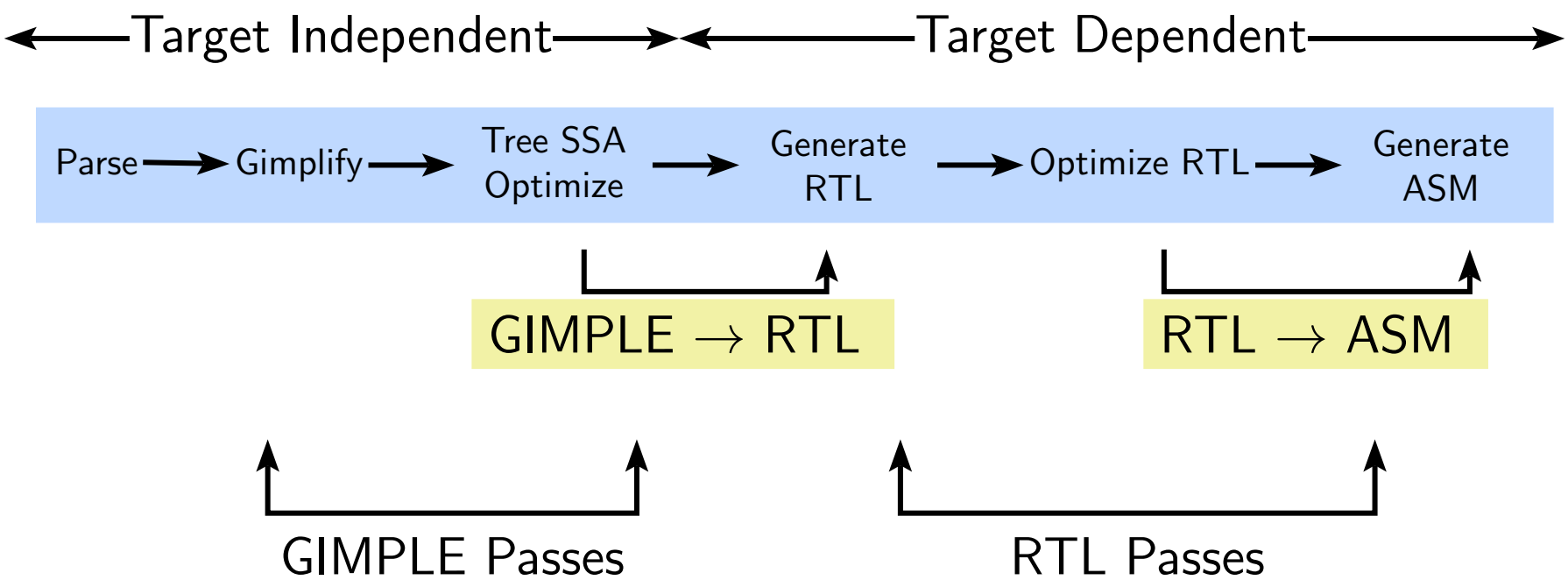
Basic Transformations in GCC

Tranformation from a language to a *different* language



Basic Transformations in GCC

Tranformation from a language to a *different* language



Transformation Passes in GCC 4.7.2

- A total of 215 unique pass names initialized in `${SOURCE}/gcc/passes.c`
  Total number of passes is 252.
  - Some passes are called multiple times in different contexts
    Conditional constant propagation is called thrice.
  - Some passes are enabled for specific architectures
  - Some passes have many variations

Pass Name	Optimization	Times
pass_cd_dce	Dead code elimination	2
pass_call_cdce	Dead call elimination	1
pass_dce	Dead code elimination	2
pass_dce_loop	Dead code elimination	3
pass_ud_rtl_dce	RTL dead code elimination	1
pass_fast_rtl_dce	RTL dead code elimination	1

- The pass sequence can be divided broadly in two parts
  - Passes on GIMPLE
  - Passes on RTL
- Some passes are organizational passes to group related passes

Passes On GIMPLE in GCC 4.7.2

Pass Group	Examples	Number of passes
Lowering	GIMPLE IR, CFG Construction	12
Simple Interprocedural Passes (Non-LTO)	Conditional Constant Propagation, Inlining, SSA Construction	40
Regular Interprocedural Passes (LTO)	Constant Propagation, Inlining	7
LTO generation passes		02
Late interprocedural passes (LTO)	Pointer Analysis	01
Other Intraprocedural Optimizations	Constant Propagation, Dead Code Elimination, PRE Value Range Propagation, Rename SSA	72
Loop Optimizations	Vectorization, Parallelization, Copy Propagation, Dead Code Elimination	28
Generating RTL		01
Total number of passes on GIMPLE		163

Passes On RTL in GCC 4.7.2

Pass Group	Examples	Number of passes
Intraprocedural Optimizations	CSE, Jump Optimization, Dead Code Elimination, Jump Optimization	27
Loop Optimizations	Loop Invariant Movement, Peeling, Unswitching	07
Machine Dependent Optimizations	Register Allocation, Instruction Scheduling, Peephole Optimizations	52
Assembly Emission and Finishing		03
Total number of passes on RTL		89

Finding Out List of Optimizations

Along with the associated flags

- A complete list of optimizations with a brief description

  ```
  gcc -c --help=optimizers
  ```

- Optimizations enabled at level 2 (other levels are 0, 1, 3, and s)

  ```
  gcc -c -O2 --help=optimizers -Q
  ```

Producing the Output of GCC Passes

- Use the option `-fdump-<ir>-<passname>`

  `<ir>` could be

  - `tree`: Intraprocedural passes on GIMPLE
  - `ipa`: Interprocedural passes on GIMPLE
  - `rtl`: Intraprocedural passes on RTL

- Use `all` in place of `<pass>` to see all dumps

  Example: `gcc -fdump-tree-all -fdump-rtl-all test.c`

- Dumping more details:

  Suffix `raw` for tree passes and `details` or `slim` for RTL passes

  Individual passes may have more verbosity options (e.g. `-fsched-verbose=5`)

- Use `-S` to stop the compilation with assembly generation

- Use `--verbose-asm` to see more detailed assembly dump

Total Number of Dumps

Dump Options: `-fdump-tree-all -fdump-ipa-all -fdump-rtl-all`

Optimization Level	Number of Dumps	Goals
Default	47	Fast compilation
O1	137	
O2	164	
O3	173	
Os	160	Optimize for space

Selected Dumps for Our Example Program

GIMPLE dumps (t)

001t.tu
003t.original
004t.gimple
006t.vcg
009t.omplower
010t.lower
013t.eh
014t.cfg
018t.ssa
020t.inline_param1
021t.einline
039t.release_ssa
040t.inline_param2
077t.cplxlower
137t.tailc
149t.optimized
232t.statistics

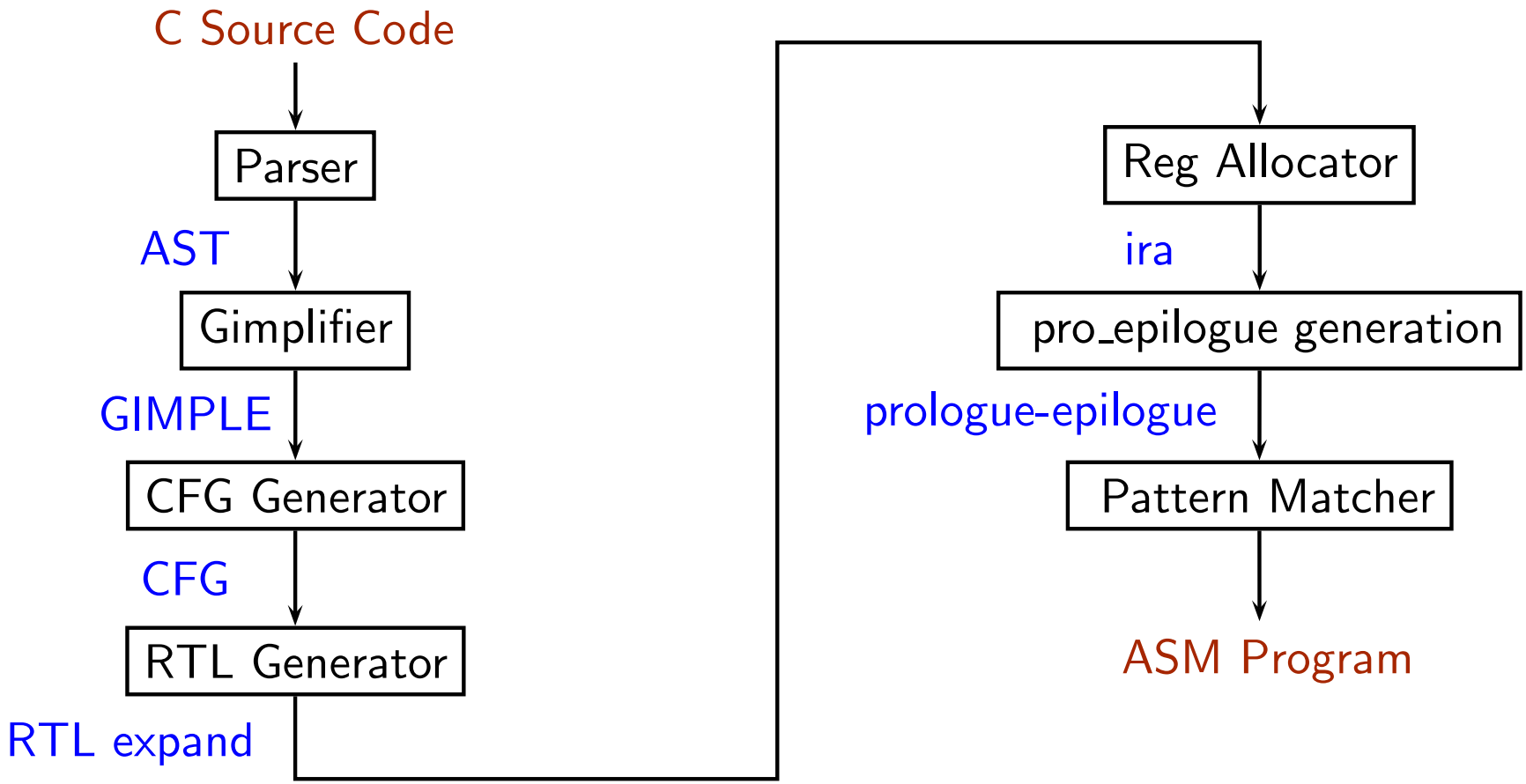
ipa dumps (i)

000i.cgraph
015i.visibility
016i.early_local_cleanups
047i.whole-program
048i.inline
054i.lto_gimple_out
055i.lto_decls_out

rtl dumps (r)

150r.expand
151r.sibling
153r.initvals
154r.unshare
155r.vregs
156r.into_cfglayout
157r.jump
158r.subreg1
159r.dfinit
163r.pre
169r.reginfo
189r.outof_cfglayout
190r.split1
193r.mode_sw
194r.asmcons
197r.ira
201r.split2
205r.pro_and_epilogue
218r.stack
219r.alignments
222r.mach
223r.barriers
227r.shorten
228r.nothrow
230r.final
231r.dfinish
assembly

Passes for First Level Graybox Probing of GCC



Lowering of abstraction!

Part 2

Examining AST Dump for C

Generating Abstract Syntax Tree

```
$ gcc -fdump-tree-original-raw test.c
```

Abstract Syntax Tree

test.c

```
int a;
int main()
{
    a = 55;
}
```

test.c.003t.original

```
;; Function main (null)
;; enabled by -tree-original

@1      bind_expr       type: @2        body: @3
@2      void_type       name: @4        algn: 8
@3      modify_expr     type: @5        op 0: @6        op 1: @7
@4      type_decl       name: @8        type: @2
@5      integer_type    name: @9        size: @10       algn: 32
                        prec: 32        sign: signed    min : @11
                        max : @12
@6      var_decl        name: @13       type: @5        srcp: t1.c:1
                        size: @10       algn: 32        used: 1
@7      integer_cst     type: @5        low : 55
@8      identifier_node strg: void      lngt: 4
@9      type_decl       name: @14       type: @5
@10     integer_cst     type: @15       low : 32
@11     integer_cst     type: @5        high: -1        low : -2147483648
@12     integer_cst     type: @5        low : 2147483647
@13     identifier_node strg: a         lngt: 1
@14     identifier_node strg: int       lngt: 3
@15     integer_type    name: @16       size: @17       algn: 64
                        prec: 64        sign: unsigned min : @18
                        max : @19
@16     identifier_node strg: bit_size_type             lngt: 13
@17     integer_cst     type: @15       low : 64
@18     integer_cst     type: @15       low : 0
@19     integer_cst     type: @15       low : -1
```

Abstract Syntax Tree

test.c

```
int a;
int main()
{
    a = 55;
}
```

test.c.003t.original

```
;; Function main (null)
;; enabled by -tree-original

@1      bind_expr       type: @2        body: @3
@2      void_type       name: @4        algn: 8
@3      modify_expr     type: @5        op 0: @6        op 1: @7
@4      type_decl       name: @8        type: @2
@5      integer_type    name: @9        size: @10       algn: 32
                        prec: 32        sign: signed    min : @11
                        max : @12
@6      var_decl        name: @13       type: @5        srcp: t1.c:1
                        size: @10       algn: 32        used: 1
@7      integer_cst     type: @5        low : 55
@8      identifier_node strg: void      lngt: 4
@9      type_decl       name: @14       type: @5
@10     integer_cst     type: @15       low : 32
@11     integer_cst     type: @5        high: -1        low : -2147483648
@12     integer_cst     type: @5        low : 2147483647
@13     identifier_node strg: a         lngt: 1
@14     identifier_node strg: int       lngt: 3
@15     integer_type    name: @16       size: @17       algn: 64
                        prec: 64        sign: unsigned min : @18
                        max : @19
@16     identifier_node strg: bit_size_type             lngt: 13
@17     integer_cst     type: @15       low : 64
@18     integer_cst     type: @15       low : 0
@19     integer_cst     type: @15       low : -1
```

Abstract Syntax Tree

test.c

```
int a;
int main()
{
    a = 55;
}
```

test.c.003t.original

```
;; Function main (null)
;; enabled by -tree-original

@1      bind_expr       type: @2       body: @3
@2      void_type       name: @4       algn: 8
@3      modify_expr     type: @5       op 0: @6       op 1: @7
@4      type_decl       name: @8       type: @2
@5      integer_type    name: @9       size: @10      algn: 32
                        prec: 32       sign: signed   min : @11
                        max : @12
@6      var_decl        name: @13      type: @5       srcp: t1.c:1
                        size: @10      algn: 32       used: 1
@7      integer_cst     type: @5       low : 55
@8      identifier_node strg: void     lngt: 4
@9      type_decl       name: @14      type: @5
@10     integer_cst     type: @15      low : 32
@11     integer_cst     type: @5       high: -1       low : -2147483648
@12     integer_cst     type: @5       low : 2147483647
@13     identifier_node strg: a        lngt: 1
@14     identifier_node strg: int      lngt: 3
@15     integer_type    name: @16      size: @17      algn: 64
                        prec: 64       sign: unsigned min : @18
                        max : @19
@16     identifier_node strg: bit_size_type           lngt: 13
@17     integer_cst     type: @15      low : 64
@18     integer_cst     type: @15      low : 0
@19     integer_cst     type: @15      low : -1
```

Abstract Syntax Tree

test.c

```
int a;
int main()
{
    a = 55;
}
```

test.c.003t.original

```
;; Function main (null)
;; enabled by -tree-original

@1      bind_expr       type: @2       body: @3
@2      void_type       name: @4       algn: 8
@3      modify_expr     type: @5       op 0: @6       op 1: @7
@4      type_decl       name: @8       type: @2
@5      integer_type    name: @9       size: @10      algn: 32
                        prec: 32       sign: signed   min : @11
                        max : @12
@6      var_decl        name: @13      type: @5       srcp: t1.c:1
                        size: @10      algn: 32       used: 1
@7      integer_cst     type: @5       low : 55
@8      identifier_node strg: void     lngt: 4
@9      type_decl       name: @14      type: @5
@10     integer_cst     type: @15      low : 32
@11     integer_cst     type: @5       high: -1       low : -2147483648
@12     integer_cst     type: @5       low : 2147483647
@13     identifier_node strg: a        lngt: 1
@14     identifier_node strg: int      lngt: 3
@15     integer_type    name: @16      size: @17      algn: 64
                        prec: 64       sign: unsigned min : @18
                        max : @19
@16     identifier_node strg: bit_size_type           lngt: 13
@17     integer_cst     type: @15      low : 64
@18     integer_cst     type: @15      low : 0
@19     integer_cst     type: @15      low : -1
```

Abstract Syntax Tree

test.c

```
int a;
int main()
{
    a = 55;
}
```

test.c.003t.original

```
;; Function main (null)
;; enabled by -tree-original

@1      bind_expr       type: @2       body: @3
@2      void_type       name: @4       algn: 8
@3      modify_expr     type: @5       op 0: @6       op 1: @7
@4      type_decl       name: @8       type: @2
@5      integer_type    name: @9       size: @10      algn: 32
                        prec: 32       sign: signed   min : @11
                        max : @12
@6      var_decl        name: @13      type: @5       srcp: t1.c:1
                        size: @10      algn: 32       used: 1
@7      integer_cst     type: @5       low : 55
@8      identifier_node strg: void     lngt: 4
@9      type_decl       name: @14      type: @5
@10     integer_cst     type: @15      low : 32
@11     integer_cst     type: @5       high: -1       low : -2147483648
@12     integer_cst     type: @5       low : 2147483647
@13     identifier_node strg: a        lngt: 1
@14     identifier_node strg: int      lngt: 3
@15     integer_type    name: @16      size: @17      algn: 64
                        prec: 64       sign: unsigned min : @18
                        max : @19
@16     identifier_node strg: bit_size_type           lngt: 13
@17     integer_cst     type: @15      low : 64
@18     integer_cst     type: @15      low : 0
@19     integer_cst     type: @15      low : -1
```

Abstract Syntax Tree

test.c

```
int a;
int main()
{
    a = 55;
}
```

test.c.003t.original

```
;; Function main (null)
;; enabled by -tree-original

@1      bind_expr       type: @2        body: @3
@2      void_type       name: @4        algn: 8
@3      modify_expr     type: @5        op 0: @6        op 1: @7
@4      type_decl       name: @8        type: @2
@5      integer_type    name: @9        size: @10       algn: 32
                        prec: 32        sign: signed    min : @11
                        max : @12
@6      var_decl        name: @13       type: @5        srcp: t1.c:1
                        size: @10       algn: 32        used: 1
@7      integer_cst     type: @5        low : 55
@8      identifier_node strg: void      lngt: 4
@9      type_decl       name: @14       type: @5
@10     integer_cst     type: @15       low : 32
@11     integer_cst     type: @5        high: -1        low : -2147483648
@12     integer_cst     type: @5        low : 2147483647
@13     identifier_node strg: a         lngt: 1
@14     identifier_node strg: int       lngt: 3
@15     integer_type    name: @16       size: @17       algn: 64
                        prec: 64        sign: unsigned min : @18
                        max : @19
@16     identifier_node strg: bit_size_type             lngt: 13
@17     integer_cst     type: @15       low : 64
@18     integer_cst     type: @15       low : 0
@19     integer_cst     type: @15       low : -1
```

Abstract Syntax Tree

test.c

```
int a;
int main()
{
    a = 55;
}
```

test.c.003t.original

```
;; Function main (null)
;; enabled by -tree-original

@1      bind_expr       type: @2        body: @3
@2      void_type       name: @4        algn: 8
@3      modify_expr     type: @5        op 0: @6        op 1: @7
@4      type_decl       name: @8        type: @2
@5      integer_type    name: @9        size: @10       algn: 32
                        prec: 32        sign: signed    min : @11
                        max : @12
@6      var_decl        name: @13       type: @5        srcp: t1.c:1
                        size: @10       algn: 32        used: 1
@7      integer_cst     type: @5        low : 55
@8      identifier_node strg: void      lngt: 4
@9      type_decl       name: @14       type: @5
@10     integer_cst     type: @15       low : 32
@11     integer_cst     type: @5        high: -1        low : -2147483648
@12     integer_cst     type: @5        low : 2147483647
@13     identifier_node strg: a         lngt: 1
@14     identifier_node strg: int       lngt: 3
@15     integer_type    name: @16       size: @17       algn: 64
                        prec: 64        sign: unsigned min : @18
                        max : @19
@16     identifier_node strg: bit_size_type             lngt: 13
@17     integer_cst     type: @15       low : 64
@18     integer_cst     type: @15       low : 0
@19     integer_cst     type: @15       low : -1
```

Part 3

Examining GIMPLE Dumps for C

Gimplifier

- About GIMPLE
  - Three-address representation derived from GENERIC
    Computation represented as a sequence of basic operations
    Temporaries introduced to hold intermediate values
  - Control construct are explicated into conditional jumps
- Examining GIMPLE Dumps
  - Examining translation of data accesses
  - Examining translation of control flow
  - Examining translation of function calls

GIMPLE: Composite Expressions Involving Scalar Variables

test.c

```
int a;

int main()
{
  int x = 10;
  int y = 5;

  x = a + x * y;
  y = y - a * x;

}
```

test.c.004t.gimple

```
  x = 10;
  y = 5;
  D.1954 = x * y;
  a.0 = a;
  x = D.1954 + a.0;
  a.1 = a;
  D.1957 = a.1 * x;
  y = y - D.1957;
```

Global variables are treated as "memory locations" and local variables are treated as "registers"

GIMPLE: Composite Expressions Involving Scalar Variables

test.c

```
int a;

int main()
{
  int x = 10;
  int y = 5;

  x = a + x * y;
  y = y - a * x;

}
```

test.c.004t.gimple

```
x = 10;
y = 5;
D.1954 = x * y;
a.0 = a;
x = D.1954 + a.0;
a.1 = a;
D.1957 = a.1 * x;
y = y - D.1957;
```

Global variables are treated as "memory locations" and local variables are treated as "registers"

GIMPLE: Composite Expressions Involving Scalar Variables

test.c

```
int a;

int main()
{
  int x = 10;
  int y = 5;

  x = a + x * y;
  y = y - a * x;

}
```

test.c.004t.gimple

```
x = 10;
y = 5;
D.1954 = x * y;
a.0 = a;
x = D.1954 + a.0;
a.1 = a;
D.1957 = a.1 * x;
y = y - D.1957;
```

Global variables are treated as "memory locations" and local variables are treated as "registers"

GIMPLE: Composite Expressions Involving Scalar Variables

test.c

```
int a;

int main()
{
  int x = 10;
  int y = 5;

  x = a + x * y;
  y = y - a * x;

}
```

test.c.004t.gimple

```
  x = 10;
  y = 5;
  D.1954 = x * y;
  a.0 = a;
  x = D.1954 + a.0;
  a.1 = a;
  D.1957 = a.1 * x;
  y = y - D.1957;
```

Global variables are treated as "memory locations" and local variables are treated as "registers"

GIMPLE: 1-D Array Accesses

test.c

```
int main()
{
  int a[3], x;
  a[1] = a[2] = 10;
  x = a[1] + a[2];
  a[0] = a[1] + a[1]*x;
}
```

test.c.004t.gimple

```
try {
    a[2] = 10;
    D.1952 = a[2];
    a[1] = D.1952;
    D.1953 = a[1];
    D.1954 = a[2];
    x = D.1953 + D.1954;
    D.1955 = x + 1;
    D.1956 = a[1];
    D.1957 = D.1955 * D.1956;
    a[0] = D.1957;
}
finally {
    a = {CLOBBER};
}
```

GIMPLE: 1-D Array Accesses

test.c

```
int main()
{
  int a[3], x;
  a[1] = a[2] = 10;
  x = a[1] + a[2];
  a[0] = a[1] + a[1]*x;
}
```

test.c.004t.gimple

```
try {
    a[2] = 10;
    D.1952 = a[2];
    a[1] = D.1952;
    D.1953 = a[1];
    D.1954 = a[2];
    x = D.1953 + D.1954;
    D.1955 = x + 1;
    D.1956 = a[1];
    D.1957 = D.1955 * D.1956;
    a[0] = D.1957;
}
finally {
    a = {CLOBBER};
}
```

GIMPLE: 1-D Array Accesses

test.c

```
int main()
{
  int a[3], x;
  a[1] = a[2] = 10;
  x = a[1] + a[2];
  a[0] = a[1] + a[1]*x;
}
```

test.c.004t.gimple

```
try {
    a[2] = 10;
    D.1952 = a[2];
    a[1] = D.1952;
    D.1953 = a[1];
    D.1954 = a[2];
    x = D.1953 + D.1954;
    D.1955 = x + 1;
    D.1956 = a[1];
    D.1957 = D.1955 * D.1956;
    a[0] = D.1957;
}
finally {
    a = {CLOBBER};
}
```

GIMPLE: 1-D Array Accesses

test.c

```
int main()
{
  int a[3], x;
  a[1] = a[2] = 10;
  x = a[1] + a[2];
  a[0] = a[1] + a[1]*x;
}
```

test.c.004t.gimple

```
try {
    a[2] = 10;
    D.1952 = a[2];
    a[1] = D.1952;
    D.1953 = a[1];
    D.1954 = a[2];
    x = D.1953 + D.1954;
    D.1955 = x + 1;
    D.1956 = a[1];
    D.1957 = D.1955 * D.1956;
    a[0] = D.1957;
}
finally {
    a = {CLOBBER};
}
```

GIMPLE: 1-D Array Accesses

test.c

```
int main()
{
  int a[3], x;
  a[1] = a[2] = 10;
  x = a[1] + a[2];
  a[0] = a[1] + a[1]*x;
}
```

test.c.004t.gimple

```
try {
    a[2] = 10;
    D.1952 = a[2];
    a[1] = D.1952;
    D.1953 = a[1];
    D.1954 = a[2];
    x = D.1953 + D.1954;
    D.1955 = x + 1;
    D.1956 = a[1];
    D.1957 = D.1955 * D.1956;
    a[0] = D.1957;
}
finally {
    a = {CLOBBER};
}
```

GIMPLE: 2-D Array Accesses

test.c

```
int main()
{
  int a[3][3], x, y;
  a[0][0] = 7;
  a[1][1] = 8;
  a[2][2] = 9;
  x = a[0][0] / a[1][1];
  y = a[1][1] % a[2][2];
}
```

test.c.004t.gimple

```
try {
      a[0][0] = 7;
      a[1][1] = 8;
      a[2][2] = 9;
      D.1953 = a[0][0];
      D.1954 = a[1][1];
      x = D.1953 / D.1954;
      D.1955 = a[1][1];
      D.1956 = a[2][2];
      y = D.1955 % D.1956;
}
finally {
      a = {CLOBBER};
}
```

GIMPLE: Use of Pointers

test.c

```
int main()
{
  int **a,*b,c;
  b = &c;
  a = &b;
  **a = 10;  /* c = 10 */
}
```

test.c.004t.gimple

```
main () {
    int * D.1953;
    int * * a;
    int * b;
    int c;
    try
    {
      b = &c;
      a = &b;
      D.1953 = *a;
      *D.1953 = 10;
    }
    finally {
        b = {CLOBBER};
        c = {CLOBBER};
    }
}
```

GIMPLE: Use of Pointers

test.c

```
int main()
{
  int **a,*b,c;
  b = &c;
  a = &b;
  **a = 10;  /* c = 10 */
}
```

test.c.004t.gimple

```
main () {
    int * D.1953;
    int * * a;
    int * b;
    int c;
    try
    {
      b = &c;
      a = &b;
      D.1953 = *a;
      *D.1953 = 10;
    }
    finally {
        b = {CLOBBER};
        c = {CLOBBER};
    }
}
```

Memory and Registers in GIMPLE

- Memory: Gobals, address taken variables, arrays
  - Scalar memory values must be explicitly loaded into registers

    ```
    a.0 = a;
    ```

  - No "addressable" memory within arrays
    - No base + offset modelling of arrays
    - Array reference is a single operation in GIMPLE
  - Since "memory" survives the lifetime of a given scope, locals are marked as clobbered at the end of the scope
- Registers: Locals, formals
  - Restricted visibility
  - Cannot be modified by function calls or a concurrent process
  - Can be freely rearranged

GIMPLE: Use of Structures

test.c

```
typedef struct address
{ char *name;
} ad;

typedef struct student
{ int roll;
  ad *ct;
} st;

int main()
{ st *s;
  s = malloc(sizeof(st));
  s->roll = 1;
  s->ct=malloc(sizeof(ad));
  s->ct->name = "Mumbai";
}
```

test.c.004t.gimple

```
main ()
{
  void * D.1957;
  struct ad * D.1958;
  struct st * s;
  extern void * malloc (unsigned int);

  s = malloc (8);
  s->roll = 1;
  D.1957 = malloc (4);
  s->ct = D.1957;
  D.1958 = s->ct;
  D.1958->name = "Mumbai";
}
```

GIMPLE: Use of Structures

test.c

```
typedef struct address
{ char *name;
} ad;

typedef struct student
{ int roll;
  ad *ct;
} st;

int main()
{ st *s;
  s = malloc(sizeof(st));
  s->roll = 1;
  s->ct=malloc(sizeof(ad));
  s->ct->name = "Mumbai";
}
```

test.c.004t.gimple

```
main ()
{
  void * D.1957;
  struct ad * D.1958;
  struct st * s;
  extern void * malloc (unsigned int);

  s = malloc (8);
  s->roll = 1;
  D.1957 = malloc (4);
  s->ct = D.1957;
  D.1958 = s->ct;
  D.1958->name = "Mumbai";
}
```

GIMPLE: Use of Structures

test.c

```
typedef struct address
{ char *name;
} ad;

typedef struct student
{ int roll;
  ad *ct;
} st;

int main()
{ st *s;
  s = malloc(sizeof(st));
  s->roll = 1;
  s->ct=malloc(sizeof(ad));
  s->ct->name = "Mumbai";
}
```

test.c.004t.gimple

```
main ()
{
  void * D.1957;
  struct ad * D.1958;
  struct st * s;
  extern void * malloc (unsigned int);

  s = malloc (8);
  s->roll = 1;
  D.1957 = malloc (4);
  s->ct = D.1957;
  D.1958 = s->ct;
  D.1958->name = "Mumbai";
}
```

GIMPLE: Use of Structures

test.c

```
typedef struct address
{ char *name;
} ad;

typedef struct student
{ int roll;
  ad *ct;
} st;

int main()
{ st *s;
  s = malloc(sizeof(st));
  s->roll = 1;
  s->ct=malloc(sizeof(ad));
  s->ct->name = "Mumbai";
}
```

test.c.004t.gimple

```
main ()
{
  void * D.1957;
  struct ad * D.1958;
  struct st * s;
  extern void * malloc (unsigned int);

  s = malloc (8);
  s->roll = 1;
  D.1957 = malloc (4);
  s->ct = D.1957;
  D.1958 = s->ct;
  D.1958->name = "Mumbai";
}
```

GIMPLE: Pointer to Array

test.c

```
int main()
{
  int *p_a, a[3];

  p_a = &a[0];

  *p_a = 10;
  *(p_a+1) = 20;
  *(p_a+2) = 30;
}
```

test.c.004t.gimple

```
main ()
{
  int * D.2048;
  int * D.2049;
  int * p_a;
  int a[3];
  try {
      p_a = &a[0];
      *p_a = 10;
      D.2048 = p_a + 4;
      *D.2048 = 20;
      D.2049 = p_a + 8;
      *D.2049 = 30;
  }
  finally {
      a = {CLOBBER};
  }
}
```

GIMPLE: Pointer to Array

test.c

```
int main()
{
  int *p_a, a[3];

  p_a = &a[0];

  *p_a = 10;
  *(p_a+1) = 20;
  *(p_a+2) = 30;
}
```

test.c.004t.gimple

```
main ()
{
  int * D.2048;
  int * D.2049;
  int * p_a;
  int a[3];
  try {
      p_a = &a[0];
      *p_a = 10;
      D.2048 = p_a + 4;
      *D.2048 = 20;
      D.2049 = p_a + 8;
      *D.2049 = 30;
  }
  finally {
      a = {CLOBBER};
  }
}
```

GIMPLE: Pointer to Array

test.c

```
int main()
{
  int *p_a, a[3];

  p_a = &a[0];

  *p_a = 10;
  *(p_a+1) = 20;
  *(p_a+2) = 30;
}
```

test.c.004t.gimple

```
main ()
{
  int * D.2048;
  int * D.2049;
  int * p_a;
  int a[3];
  try {
      p_a = &a[0];
      *p_a = 10;
      D.2048 = p_a + 4;
      *D.2048 = 20;
      D.2049 = p_a + 8;
      *D.2049 = 30;
  }
  finally {
      a = {CLOBBER};
  }
}
```

GIMPLE: Pointer to Array

test.c

```
int main()
{
  int *p_a, a[3];

  p_a = &a[0];

  *p_a = 10;
  *(p_a+1) = 20;
  *(p_a+2) = 30;
}
```

test.c.004t.gimple

```
main ()
{
  int * D.2048;
  int * D.2049;
  int * p_a;
  int a[3];
  try {
      p_a = &a[0];
      *p_a = 10;
      D.2048 = p_a + 4;
      *D.2048 = 20;
      D.2049 = p_a + 8;
      *D.2049 = 30;
  }
  finally {
      a = {CLOBBER};
  }
}
```

GIMPLE: Translation of Conditional Statements

test.c

```
int main()
{
    int a=2, b=3, c=4;
    while (a<=7)
    {
        a = a+1;
    }
     if (a<=12)
          a = a+b+c;

}
```

test.c.004t.gimple

```
if (a <= 12) goto <D.1200>;
else  goto <D.1201>;
<D.1200>:
D.1199 = a + b;
a = D.1199 + c;
<D.1201>:
```

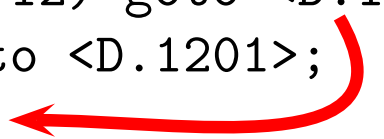
GIMPLE: Translation of Conditional Statements

test.c

```
int main()
{
    int a=2, b=3, c=4;
    while (a<=7)
    {
        a = a+1;
    }
     if (a<=12)
          a = a+b+c;

}
```

test.c.004t.gimple

```
if (a <= 12) goto <D.1200>;
else  goto <D.1201>;
<D.1200>:
D.1199 = a + b;
a = D.1199 + c;
<D.1201>:
```



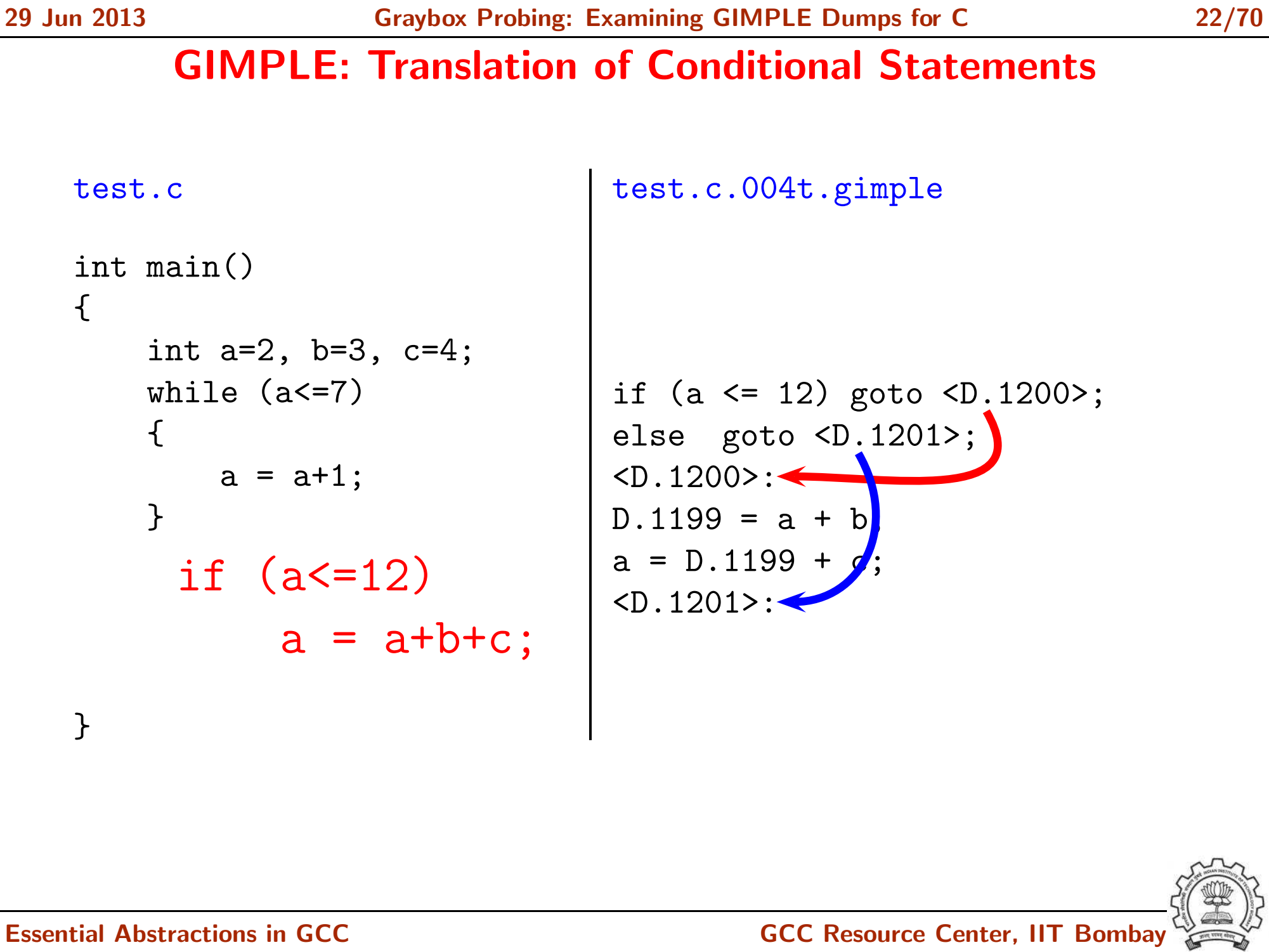
GIMPLE: Translation of Conditional Statements

test.c

```
int main()
{
    int a=2, b=3, c=4;
    while (a<=7)
    {
        a = a+1;
    }
     if (a<=12)
          a = a+b+c;

}
```

test.c.004t.gimple

```
if (a <= 12) goto <D.1200>;
else  goto <D.1201>;
<D.1200>:
D.1199 = a + b;
a = D.1199 + c;
<D.1201>:
```



GIMPLE: Translation of Loops

test.c

```
int main()
{
    int a=2, b=3, c=4;
     while (a<=7)
     {
          a = a+1;
     }
    if (a<=12)
        a = a+b+c;
}
```

test.c.004t.gimple

```
goto <D.1197>;
<D.1196>:
a = a + 1;
<D.1197>:
if (a <= 7) goto <D.1196>;
else goto <D.1198>;
<D.1198>:
```

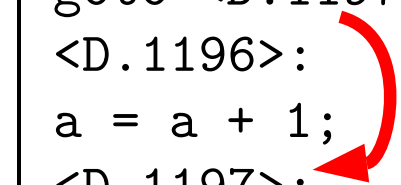
GIMPLE: Translation of Loops

test.c

```
int main()
{
    int a=2, b=3, c=4;
     while (a<=7)
     {
          a = a+1;
     }
    if (a<=12)
        a = a+b+c;
}
```

test.c.004t.gimple

```
goto <D.1197>;
<D.1196>:
a = a + 1;
<D.1197>:
if (a <= 7) goto <D.1196>;
else goto <D.1198>;
<D.1198>:
```



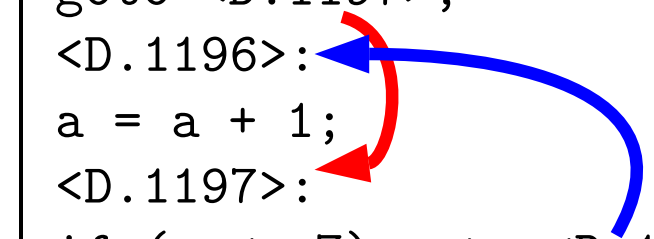
GIMPLE: Translation of Loops

test.c

```
int main()
{
    int a=2, b=3, c=4;
     while (a<=7)
     {
          a = a+1;
     }
    if (a<=12)
        a = a+b+c;
}
```

test.c.004t.gimple

```
goto <D.1197>;
<D.1196>:
a = a + 1;
<D.1197>:
if (a <= 7) goto <D.1196>;
else goto <D.1198>;
<D.1198>:
```



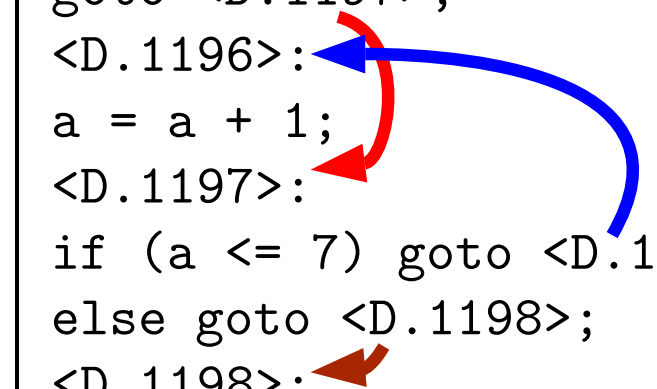
GIMPLE: Translation of Loops

test.c

```
int main()
{
    int a=2, b=3, c=4;
     while (a<=7)
     {
          a = a+1;
     }
    if (a<=12)
        a = a+b+c;
}
```

test.c.004t.gimple

```
goto <D.1197>;
<D.1196>:
a = a + 1;
<D.1197>:
if (a <= 7) goto <D.1196>;
else goto <D.1198>;
<D.1198>:
```



Control Flow Graph: Textual View

test.c.004t.gimple

```
if (a <= 12) goto <D.1200>;
else  goto <D.1201>;
<D.1200>:
D.1199 = a + b;
a = D.1199 + c;
<D.1201>:
```

test.c.014t.cfg

```
<bb 5>:
  if (a <= 12)
    goto <bb 6>;
  else
    goto <bb 7>;

<bb 6>:
  D.1199 = a + b;
  a = D.1199 + c;

<bb 7>:
  return;
```

Control Flow Graph: Textual View

test.c.004t.gimple

```
if (a <= 12) goto <D.1200>;
else  goto <D.1201>;
<D.1200>:
D.1199 = a + b;
a = D.1199 + c;
<D.1201>:
```

test.c.014t.cfg

```
<bb 5>:
  if (a <= 12)
    goto <bb 6>;
  else
    goto <bb 7>;

<bb 6>:
  D.1199 = a + b;
  a = D.1199 + c;

<bb 7>:
  return;
```

Control Flow Graph: Textual View

test.c.004t.gimple

```
if (a <= 12) goto <D.1200>;
else  goto <D.1201>;
<D.1200>:
D.1199 = a + b;
a = D.1199 + c;
<D.1201>:
```

test.c.014t.cfg

```
<bb 5>:
  if (a <= 12)
    goto <bb 6>;
  else
    goto <bb 7>;

<bb 6>:
  D.1199 = a + b;
  a = D.1199 + c;

<bb 7>:
  return;
```

Control Flow Graph: Textual View

test.c.004t.gimple

```
if (a <= 12) goto <D.1200>;
else  goto <D.1201>;
<D.1200>:
D.1199 = a + b;
a = D.1199 + c;
<D.1201>:
```

test.c.014t.cfg

```
<bb 5>:
  if (a <= 12)
    goto <bb 6>;
  else
    goto <bb 7>;

<bb 6>:
  D.1199 = a + b;
  a = D.1199 + c;

<bb 7>:
  return;
```

Control Flow Graph: Textual View

test.c.004t.gimple

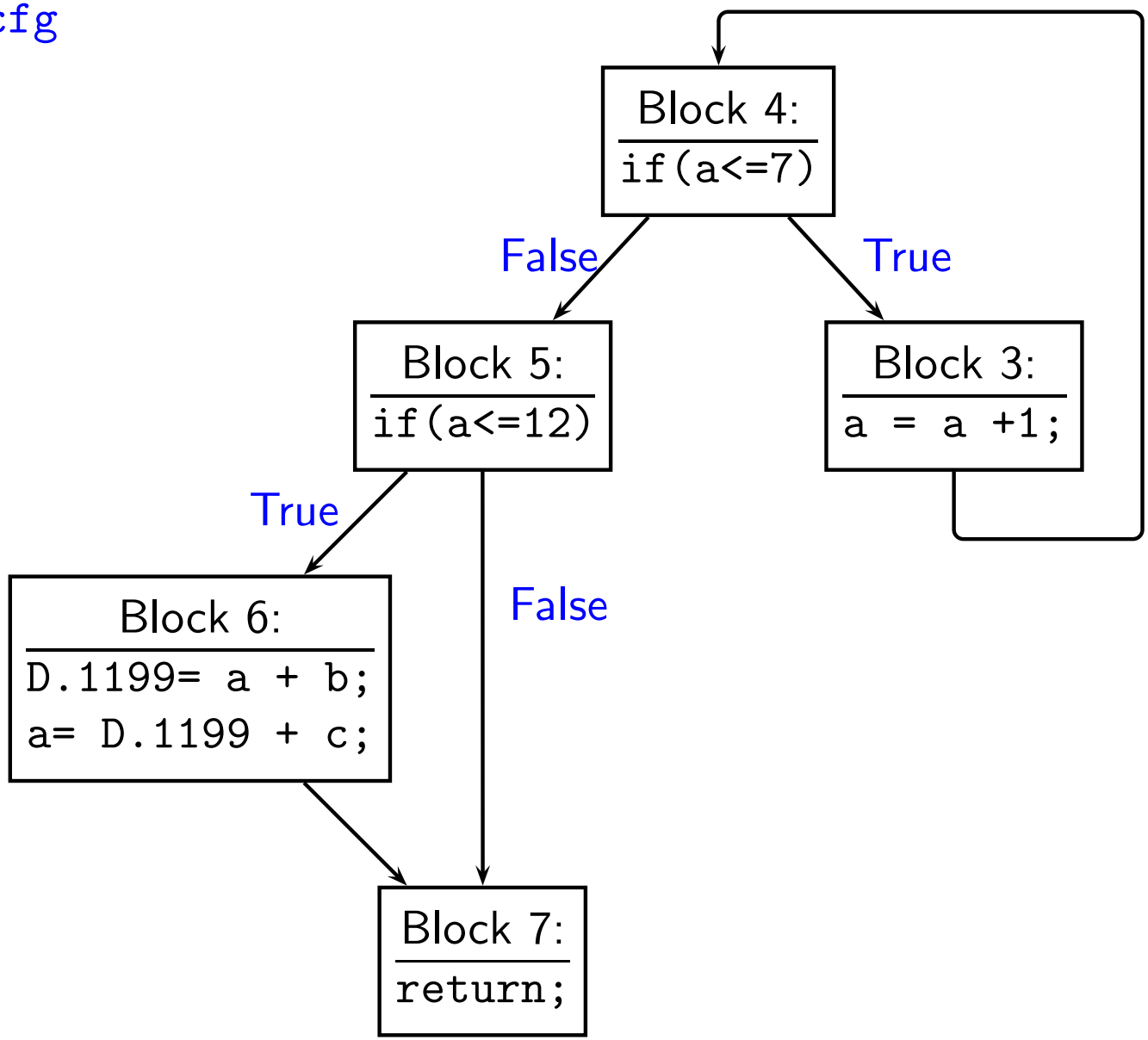
```
if (a <= 12) goto <D.1200>;
else  goto <D.1201>;
<D.1200>:
D.1199 = a + b;
a = D.1199 + c;
<D.1201>:
```

test.c.014t.cfg

```
<bb 5>:
  if (a <= 12)
    goto <bb 6>;
  else
    goto <bb 7>;

<bb 6>:
  D.1199 = a + b;
  a = D.1199 + c;

<bb 7>:
  return;
```

Control Flow Graph: Pictorial View

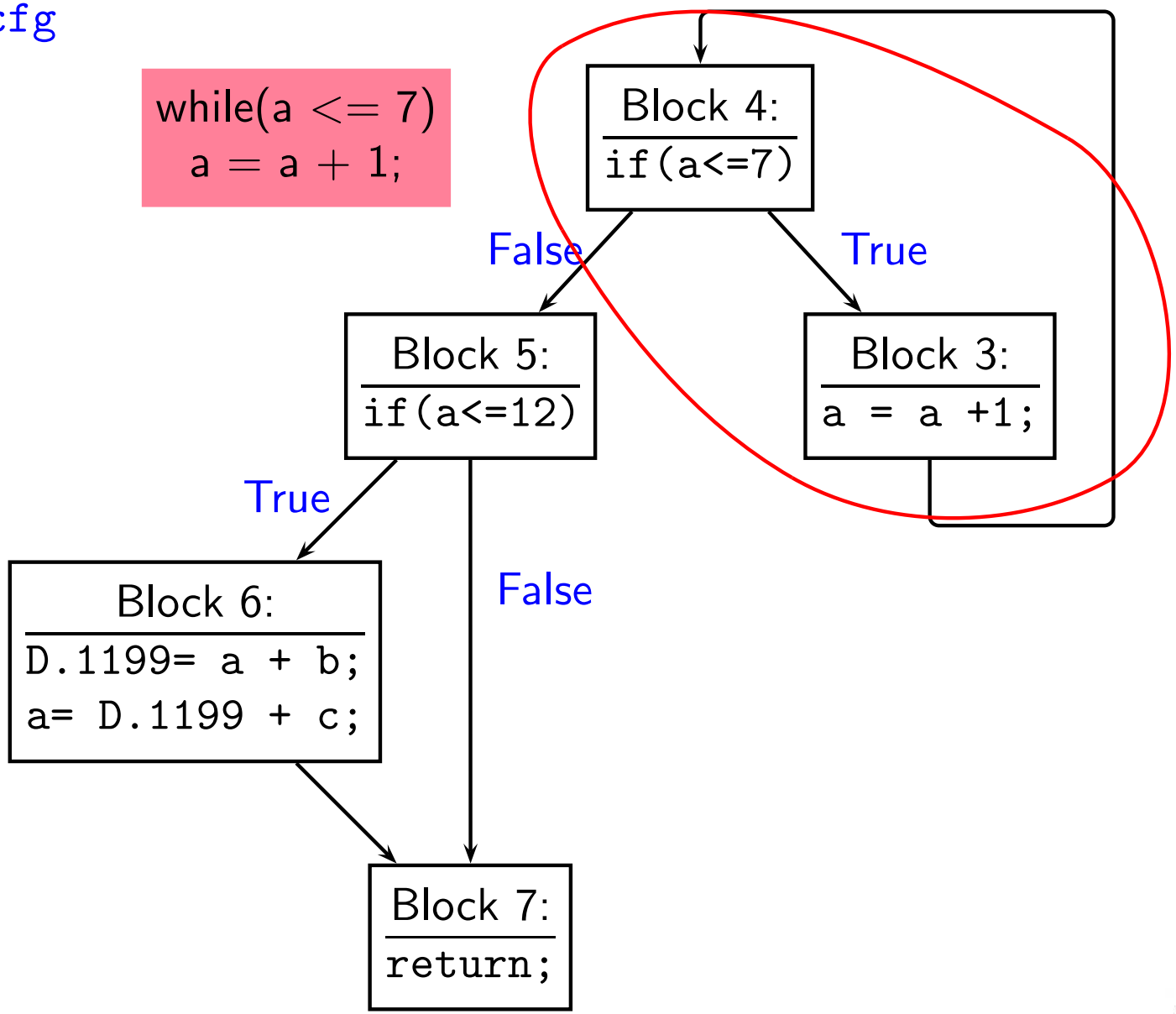
test.c.014t.cfg



Control Flow Graph: Pictorial View

test.c.014t.cfg

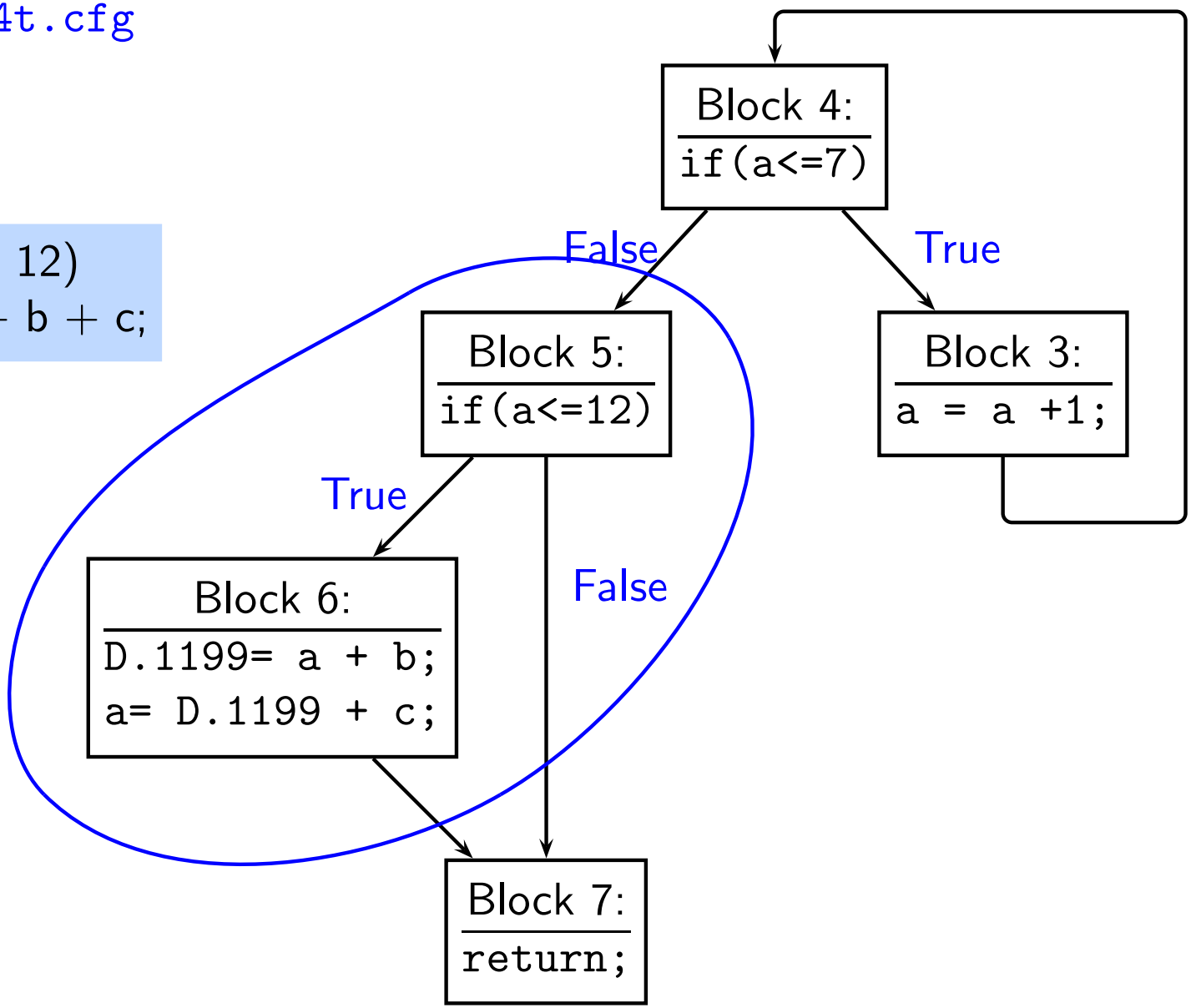
```
while(a <= 7)
  a = a + 1;
```



Control Flow Graph: Pictorial View

test.c.014t.cfg

```
if(a <= 12)
a = a + b + c;
```



GIMPLE: Function Calls and Call Graph

test.c

```
extern int divide(int, int);
int multiply(int a, int b)
{
   return a*b;
}

int main()
{ int x,y;
  x = divide(20,5);
  y = multiply(x,2);
  printf("%d\n", y);
}
```

test.c.000i.cgraph

```
printf/3  @0x7fd094bbba20 availabilit
  called by: main/1  (1.00 per call)
  calls:
divide/2  @0x7fd094bbb900 availabilit
  called by: main/1  (1.00 per call)
  calls:
main/1  @0x7fd094bbb7e0 (asm: main) a
  called by:
  calls: printf/3  (1.00 per call)
         multiply/0 (1.00 per call)
         divide/2 (1.00 per call)
multiply/0  @0x7fd094bbb6c0 (asm: mul
  called by: main/1  (1.00 per call)
  calls:
```

GIMPLE: Function Calls and Call Graph

test.c

```
extern int divide(int, int);
int multiply(int a, int b)
{
   return a*b;
}

int main()
{ int x,y;
  x = divide(20,5);
  y = multiply(x,2);
  printf("%d\n", y);
}
```

test.c.000i.cgraph

```
printf/3  @0x7fd094bbba20 availabilit
  called by: main/1  (1.00 per call)
  calls:
divide/2  @0x7fd094bbb900 availabilit
  called by: main/1  (1.00 per call)
  calls:
main/1  @0x7fd094bbb7e0 (asm: main) a
  called by:
  calls: printf/3  (1.00 per call)
         multiply/0 (1.00 per call)
         divide/2 (1.00 per call)
multiply/0  @0x7fd094bbb6c0 (asm: mul
  called by: main/1  (1.00 per call)
  calls:
```

GIMPLE: Function Calls and Call Graph

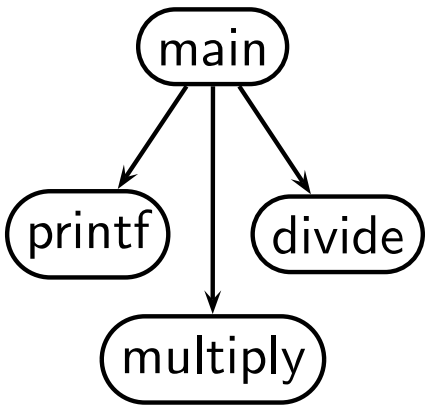
test.c

```
extern int divide(int, int);
int multiply(int a, int b)
{
   return a*b;
}

int main()
{ int x,y;
  x = divide(20,5);
  y = multiply(x,2);
  printf("%d\n", y);
}
```

test.c.000i.cgraph

```
printf/3
  called by: main/1
  calls:
divide/2
  called by: main/1
  calls:
main/1
  called by:
  calls: printf/3
         multiply/0
         divide/2
multiply/0
  called by: main/1
  calls:
```

call graph



GIMPLE: Function Calls and Call Graph

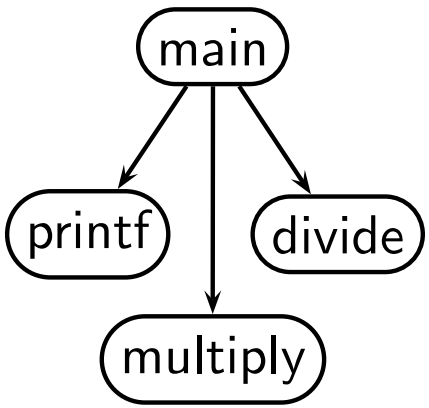
test.c

```
extern int divide(int, int);
int multiply(int a, int b)
{
   return a*b;
}

int main()
{ int x,y;
  x = divide(20,5);
  y = multiply(x,2);
  printf("%d\n", y);
}
```

test.c.000i.cgraph

```
printf/3
  called by: main/1
  calls:
divide/2
  called by: main/1
  calls:
main/1
  called by:
  calls: printf/3
         multiply/0
         divide/2
multiply/0
  called by: main/1
  calls:
```

call graph

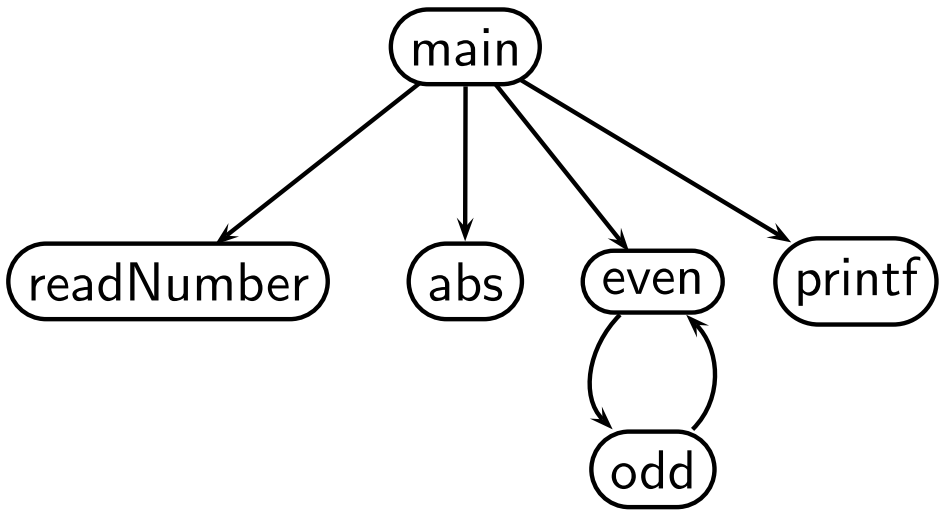


GIMPLE: Call Graphs for Recursive Functions

test.c

```
int even(int n)
{  if (n == 0) return 1;
   else return (!odd(n-1));
}

int odd(int n)
{  if (n == 1) return 1;
   else return (!even(n-1));
}

main()
{ int n;

  n = abs(readNumber());
  if (even(n))
    printf ("n is even\n");
  else printf ("n is odd\n");
}
```

call graph



Inspect GIMPLE When in Doubt (1)

```
int x=2, y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

Inspect GIMPLE When in Doubt (1)

```
int x=2, y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

$x = 10$, $y = 5$

Inspect GIMPLE When in Doubt (1)

```
int x=2, y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

$x = 10, y = 5$

```
x = 2;
y = 3;
x = x + 1;
D.1572 = y + x;
y = y + 1;
x = D.1572 + y;
y = y + 1;
```

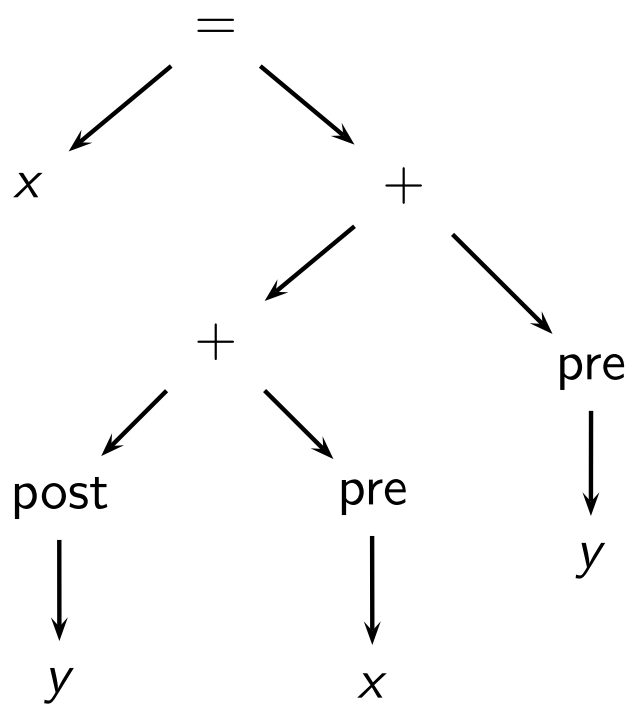
Inspect GIMPLE When in Doubt (1)

```
int x=2, y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

$x = 10, y = 5$

```
x = 2;
y = 3;
x = x + 1;
D.1572 = y + x;
y = y + 1;
x = D.1572 + y;
y = y + 1;
```



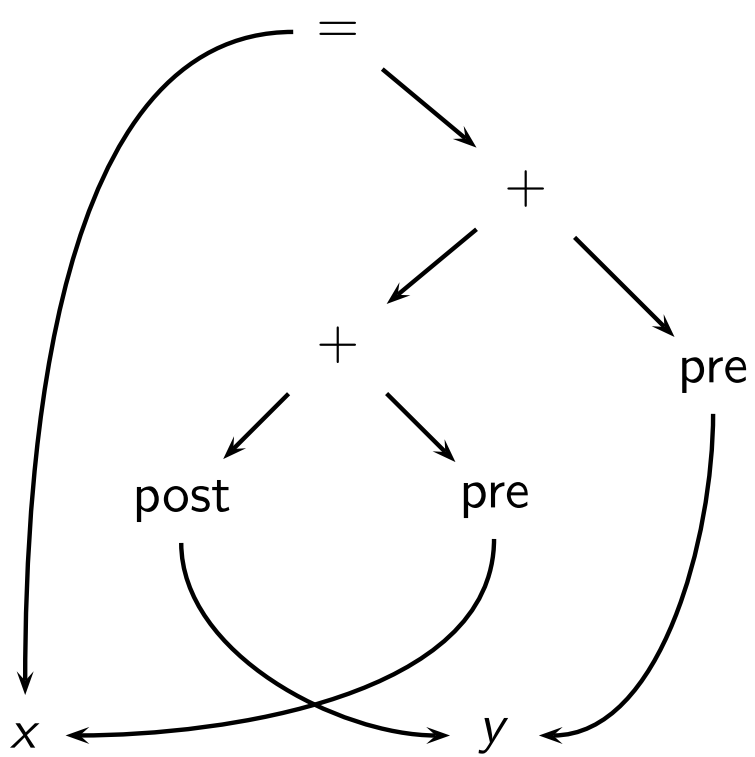
Inspect GIMPLE When in Doubt (1)

```
int x=2, y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

$x = 10, y = 5$

```
x = 2;
y = 3;
x = x + 1;
D.1572 = y + x;
y = y + 1;
x = D.1572 + y;
y = y + 1;
```



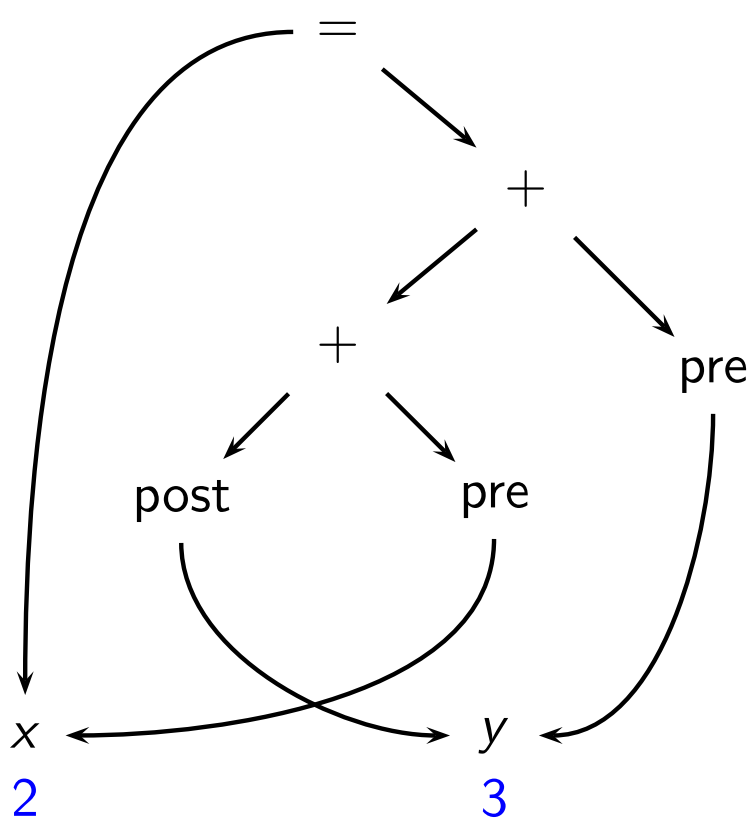
Inspect GIMPLE When in Doubt (1)

```
int x=2, y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

$x = 10, y = 5$

```
x = 2;
y = 3;
x = x + 1;
D.1572 = y + x;
y = y + 1;
x = D.1572 + y;
y = y + 1;
```



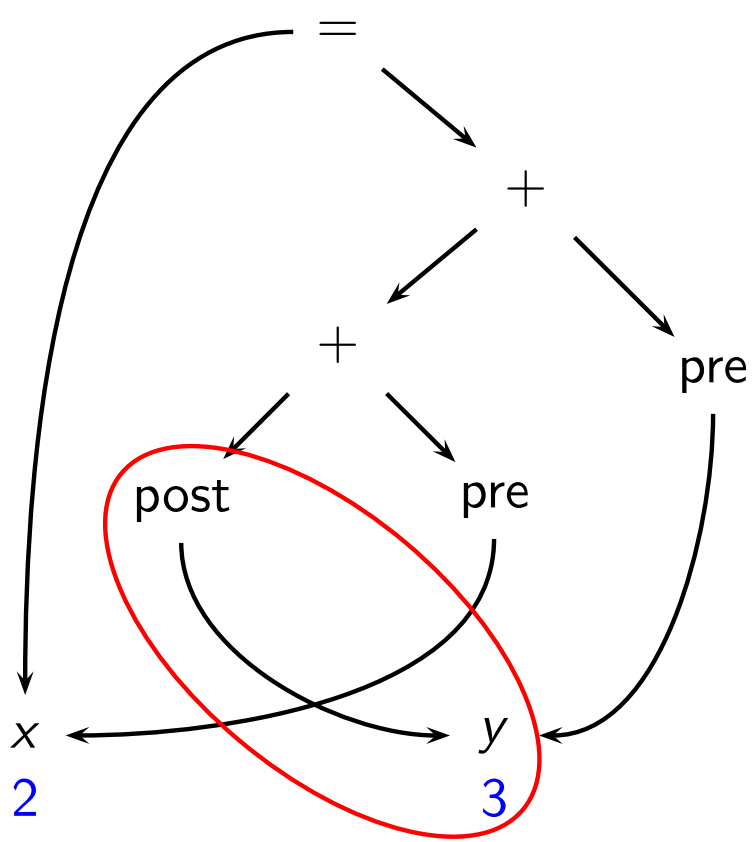
Inspect GIMPLE When in Doubt (1)

```
int x=2, y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

$x = 10, y = 5$

```
x = 2;
y = 3;
x = x + 1;
D.1572 = y + x;
y = y + 1;
x = D.1572 + y;
y = y + 1;
```



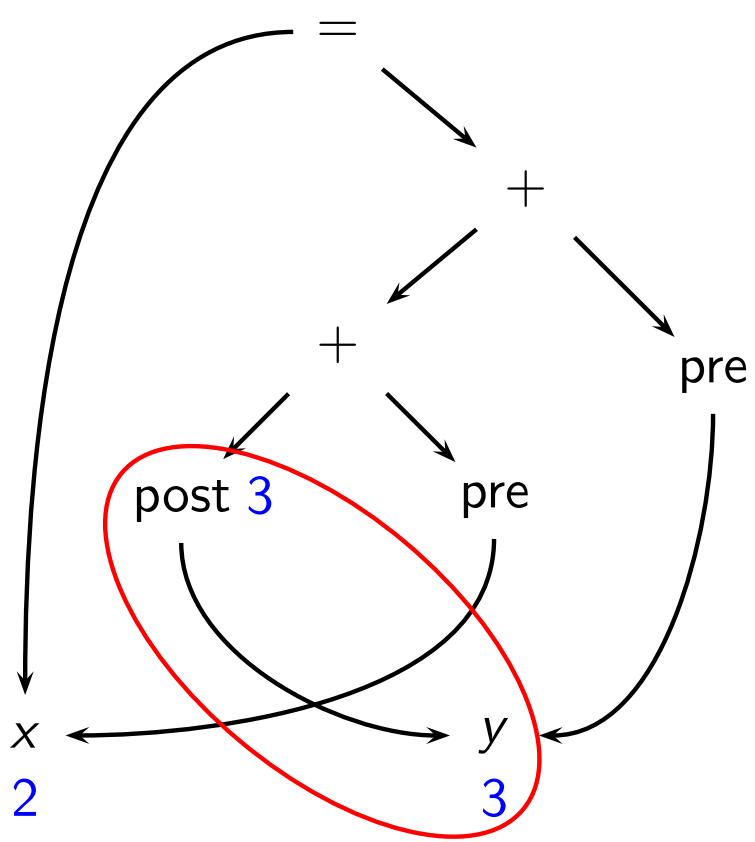
Inspect GIMPLE When in Doubt (1)

```
int x=2, y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

$x = 10, y = 5$

```
x = 2;
y = 3;
x = x + 1;
D.1572 = y + x;
y = y + 1;
x = D.1572 + y;
y = y + 1;
```



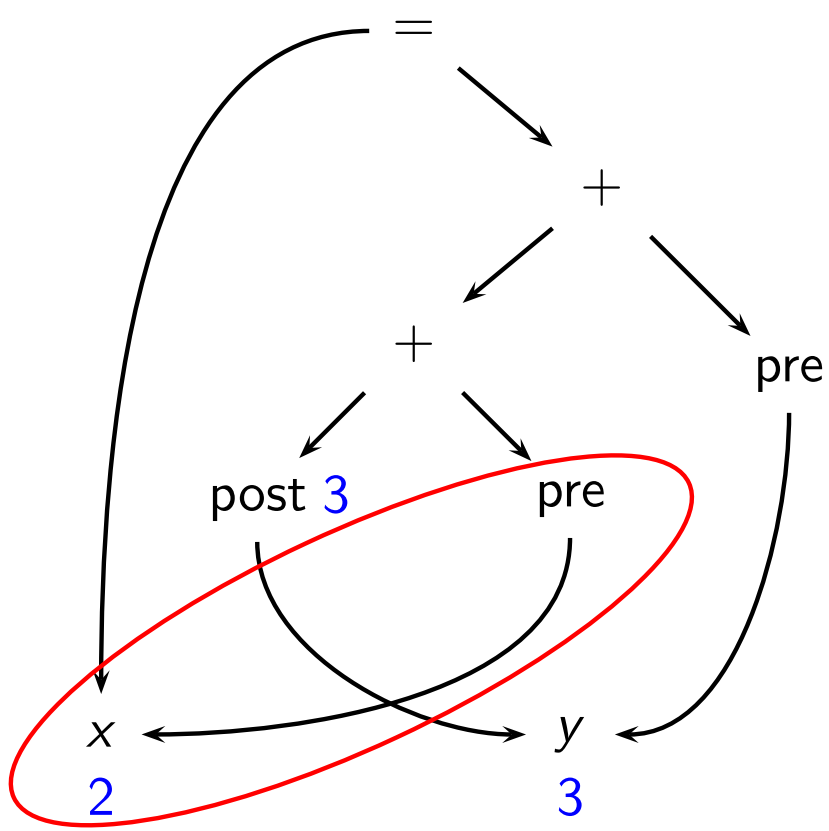
Inspect GIMPLE When in Doubt (1)

```
int x=2, y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

$x = 10, y = 5$

```
x = 2;
y = 3;
x = x + 1;
D.1572 = y + x;
y = y + 1;
x = D.1572 + y;
y = y + 1;
```



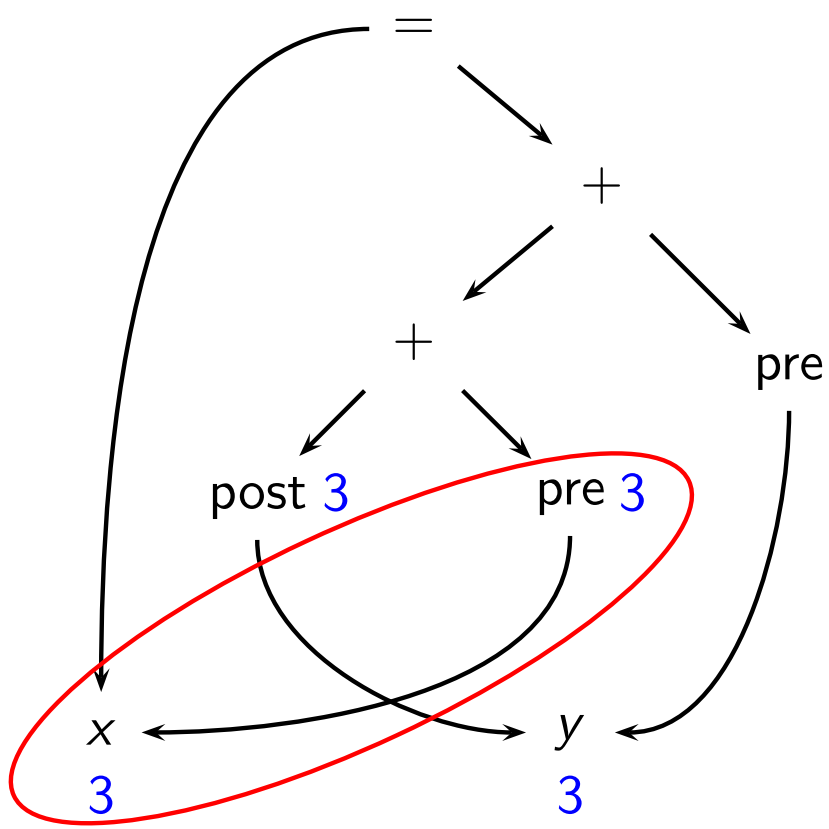
Inspect GIMPLE When in Doubt (1)

```
int x=2, y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

$x = 10, y = 5$

```
x = 2;
y = 3;
x = x + 1;
D.1572 = y + x;
y = y + 1;
x = D.1572 + y;
y = y + 1;
```



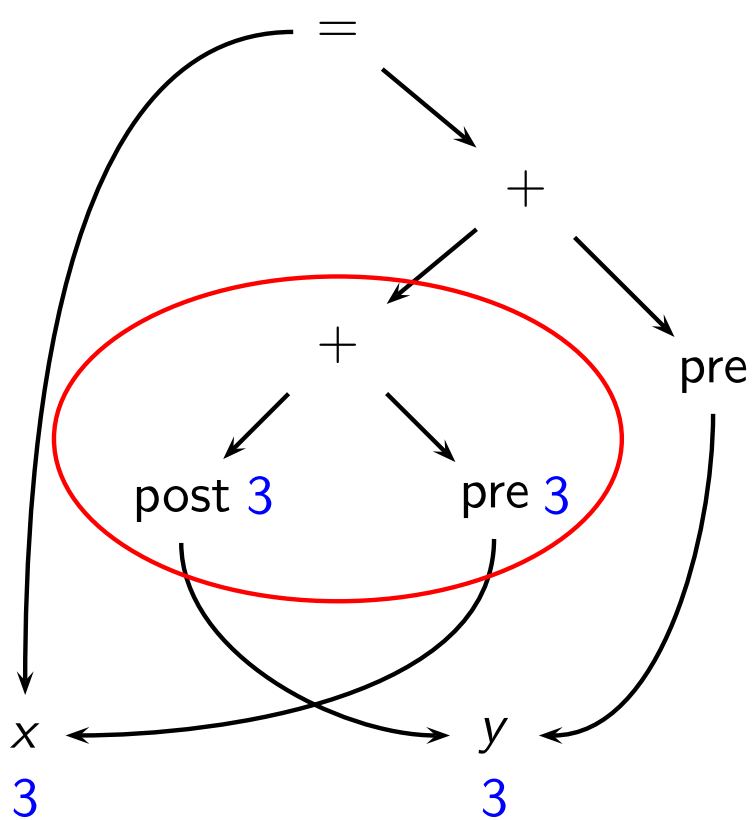
Inspect GIMPLE When in Doubt (1)

```
int x=2, y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

$x = 10, y = 5$

```
x = 2;
y = 3;
x = x + 1;
D.1572 = y + x;
y = y + 1;
x = D.1572 + y;
y = y + 1;
```



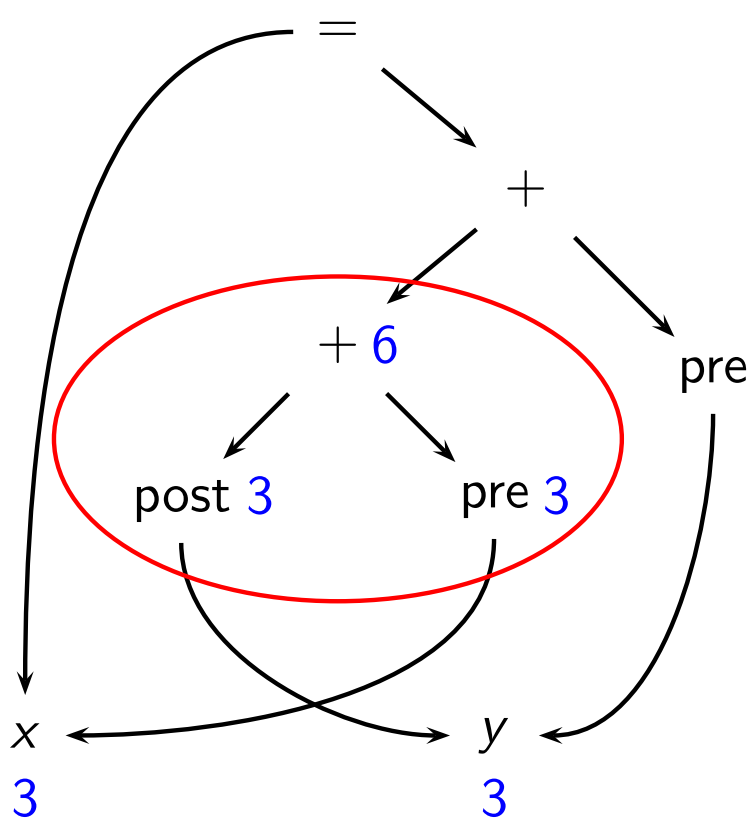
Inspect GIMPLE When in Doubt (1)

```
int x=2, y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

$x = 10, y = 5$

```
x = 2;
y = 3;
x = x + 1;
D.1572 = y + x;
y = y + 1;
x = D.1572 + y;
y = y + 1;
```



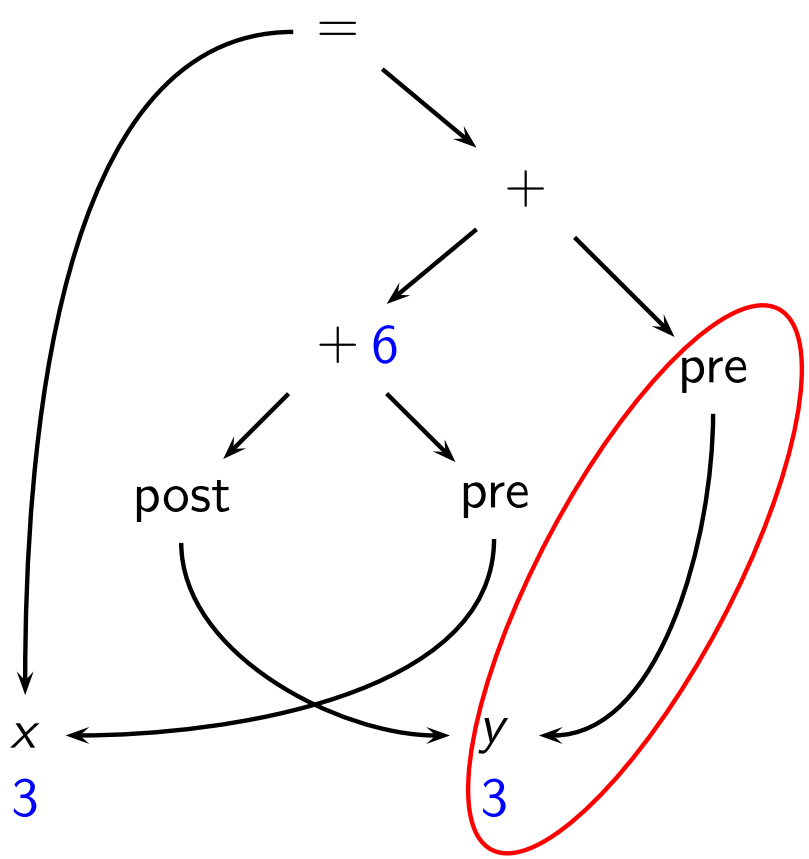
Inspect GIMPLE When in Doubt (1)

```
int x=2, y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

$x = 10, y = 5$

```
x = 2;
y = 3;
x = x + 1;
D.1572 = y + x;
y = y + 1;
x = D.1572 + y;
y = y + 1;
```



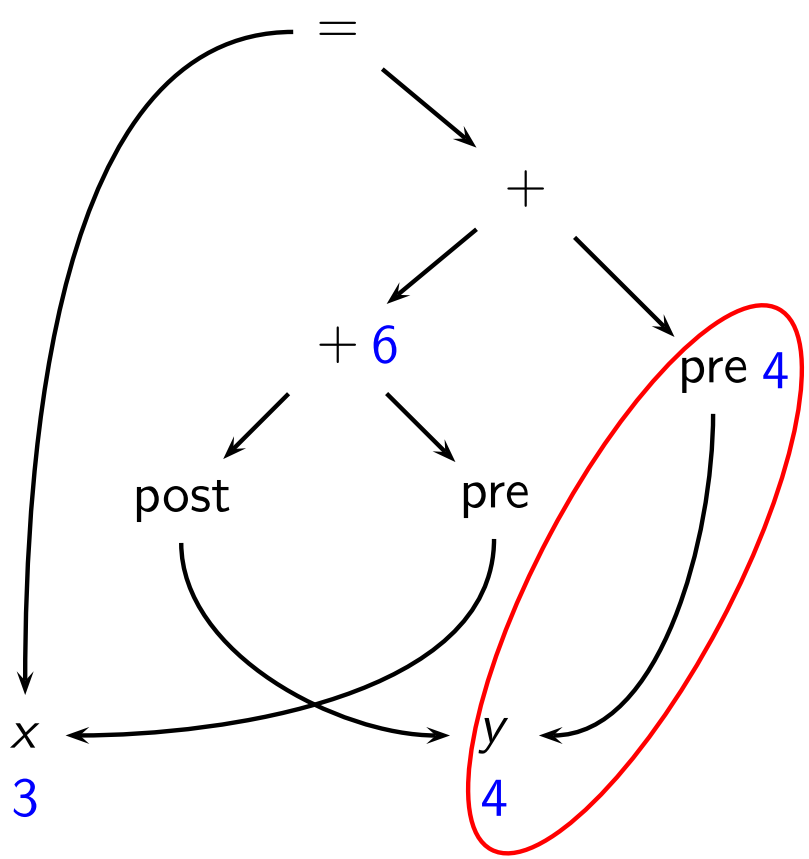
Inspect GIMPLE When in Doubt (1)

```
int x=2, y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

x = 10, y = 5

```
x = 2;
y = 3;
x = x + 1;
D.1572 = y + x;
y = y + 1;
x = D.1572 + y;
y = y + 1;
```



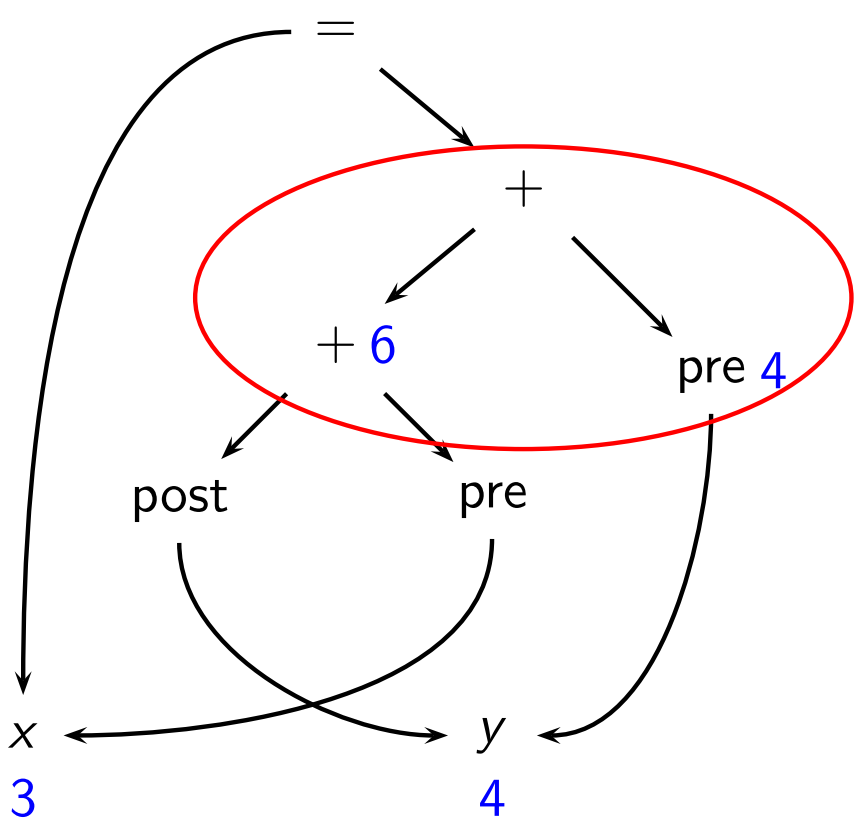
Inspect GIMPLE When in Doubt (1)

```
int x=2, y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

$x = 10, y = 5$

```
x = 2;
y = 3;
x = x + 1;
D.1572 = y + x;
y = y + 1;
x = D.1572 + y;
y = y + 1;
```



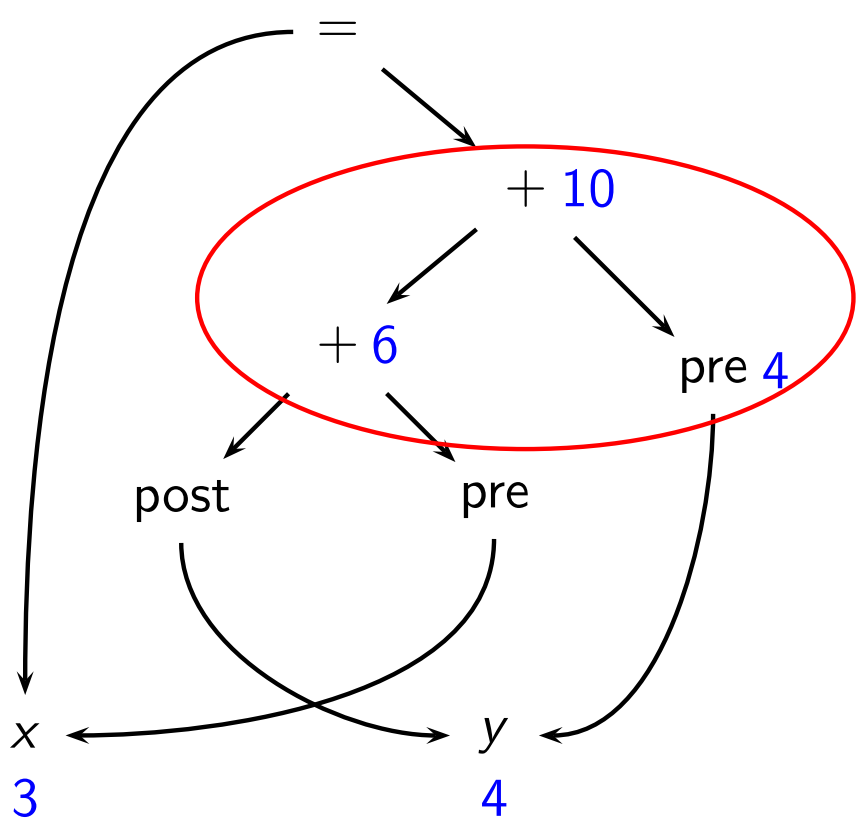
Inspect GIMPLE When in Doubt (1)

```
int x=2, y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

$x = 10, y = 5$

```
x = 2;
y = 3;
x = x + 1;
D.1572 = y + x;
y = y + 1;
x = D.1572 + y;
y = y + 1;
```



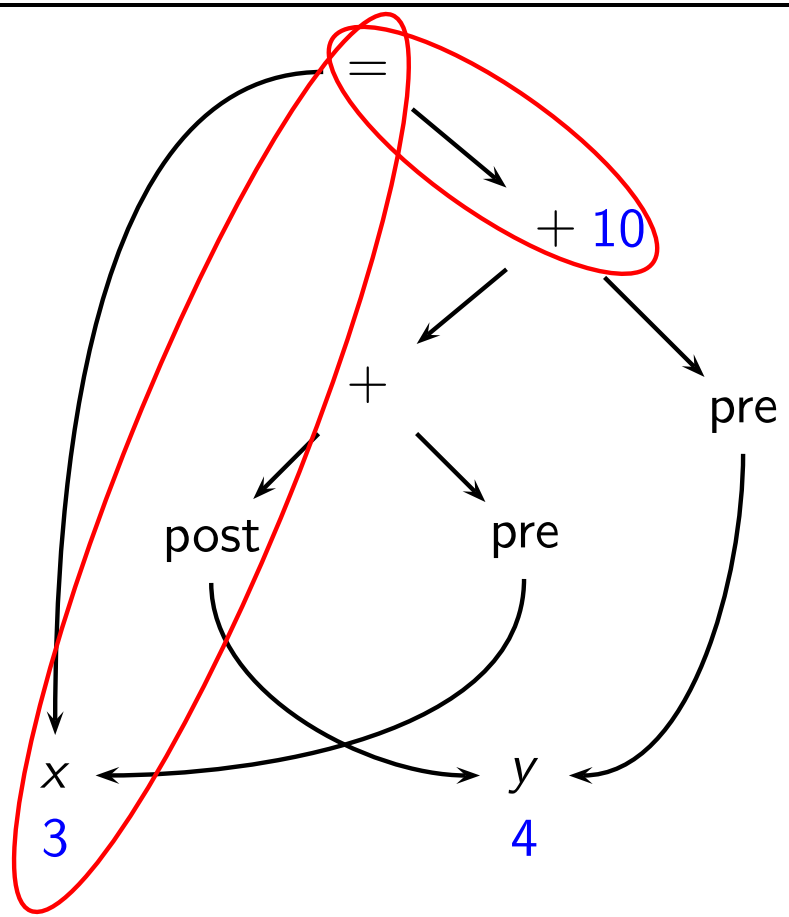
Inspect GIMPLE When in Doubt (1)

```
int x=2, y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

x = 10, y = 5

```
x = 2;
y = 3;
x = x + 1;
D.1572 = y + x;
y = y + 1;
x = D.1572 + y;
y = y + 1;
```



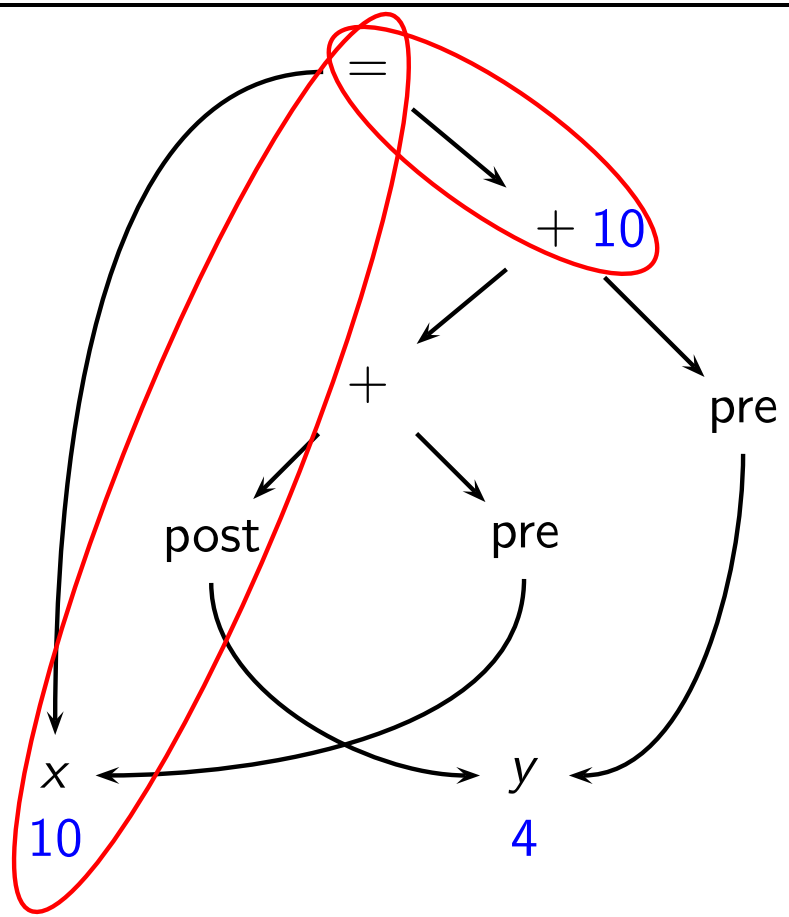
Inspect GIMPLE When in Doubt (1)

```
int x=2, y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

x = 10, y = 5

```
x = 2;
y = 3;
x = x + 1;
D.1572 = y + x;
y = y + 1;
x = D.1572 + y;
y = y + 1;
```



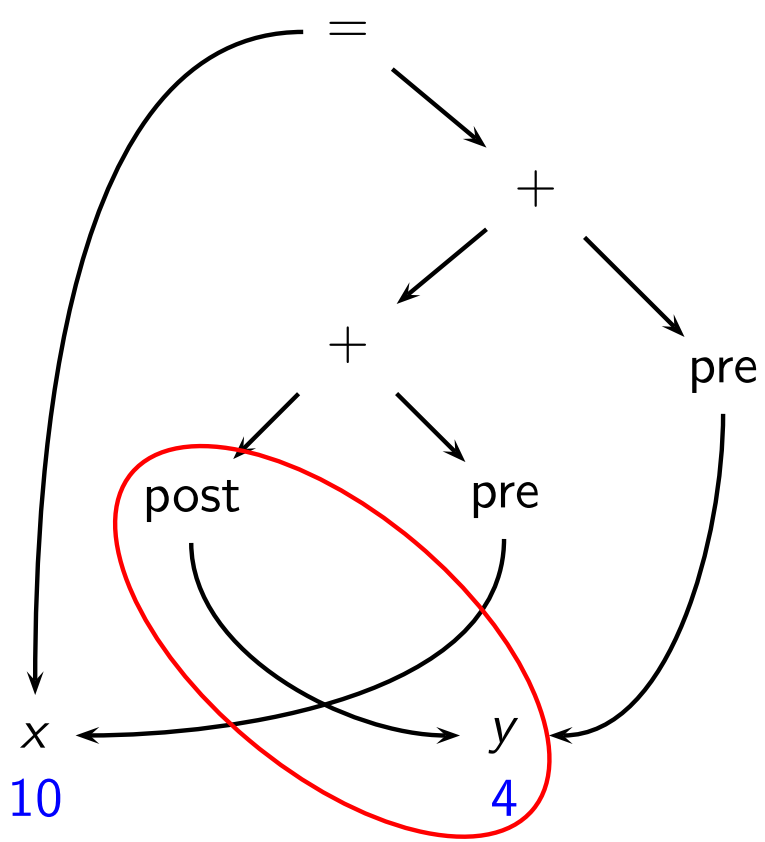
Inspect GIMPLE When in Doubt (1)

```
int x=2, y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

x = 10, y = 5

```
x = 2;
y = 3;
x = x + 1;
D.1572 = y + x;
y = y + 1;
x = D.1572 + y;
y = y + 1;
```



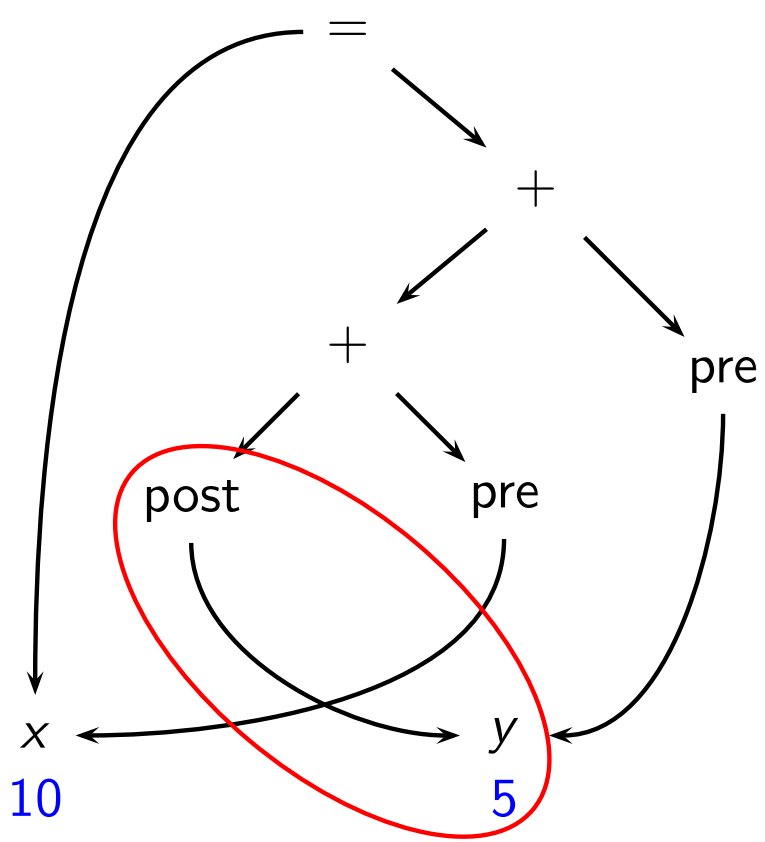
Inspect GIMPLE When in Doubt (1)

```
int x=2, y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

x = 10, y = 5

```
x = 2;
y = 3;
x = x + 1;
D.1572 = y + x;
y = y + 1;
x = D.1572 + y;
y = y + 1;
```



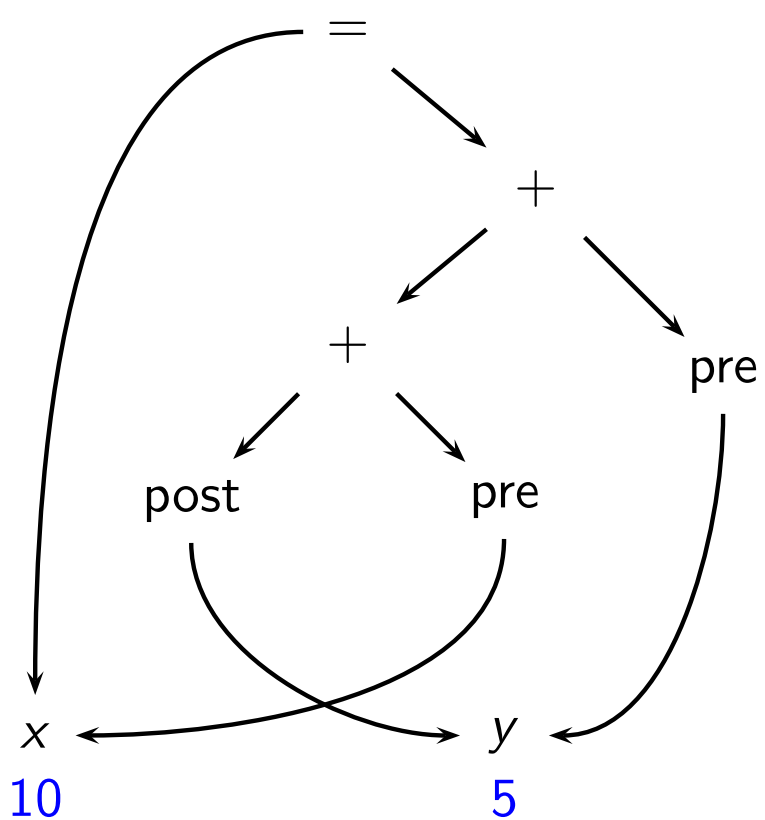
Inspect GIMPLE When in Doubt (1)

```
int x=2, y=3;
x = y++ + ++x + ++y;
```

What are the values of x and y?

x = 10, y = 5

```
x = 2;
y = 3;
x = x + 1;
D.1572 = y + x;
y = y + 1;
x = D.1572 + y;
y = y + 1;
```



Inspect GIMPLE When in Doubt (2)

- How is `a[i] = i++` handled?

  This is an undefined behaviour as per C standards.

- What is the order of parameter evaluation?

  For a call `f(getX(),getY())`, is the order left to right? arbitrary?

  Is the evaluation order in GCC consistent?

- Understanding complicated declarations in C can be difficult

  What does the following declaration mean :

  ```
  int * (* (*MYVAR) (int) ) [10];
  ```

  Hint: Use `-fdump-tree-original-raw-verbose` option. The dump to see is `003t.original`

Part 4

Examining RTL Dumps

Will be Covered Before Machine Descriptions

Part 5

Examining Assembly Dumps

i386 Assembly

Dump file: `test.s`

```
        jmp   .L2
.L3:
        addl  $1, -4(%rbp)
.L2:
        cmpl  $7, -4(%rbp)
        jle   .L3
        cmpl  $12, -4(%rbp)
        jg    .L5
        movl  -8(%rbp), %eax
        movl  -4(%rbp), %edx
        addl  %eax, %edx
        movl  -12(%rbp), %eax
        addl  %edx, %eax
        movl  %eax, -4(%rbp)
.L5:
```

```
while (a <= 7)
{
    a = a+1;
}
if (a <= 12)
{
    a = a+b+c;
}
```

i386 Assembly

Dump file: `test.s`

```
        jmp   .L2
.L3:
        addl  $1, -4(%rbp)
.L2:
        cmpl  $7, -4(%rbp)
        jle   .L3
        cmpl  $12, -4(%rbp)
        jg    .L5
        movl  -8(%rbp), %eax
        movl  -4(%rbp), %edx
        addl  %eax, %edx
        movl  -12(%rbp), %eax
        addl  %edx, %eax
        movl  %eax, -4(%rbp)
.L5:
```

```
while (a <= 7)
{
    a = a+1;
}
if (a <= 12)
{
    a = a+b+c;
}
```

i386 Assembly

Dump file: `test.s`

```
        jmp   .L2
.L3:
        addl  $1, -4(%rbp)
.L2:
        cmpl  $7, -4(%rbp)
        jle   .L3
        cmpl  $12, -4(%rbp)
        jg    .L5
        movl  -8(%rbp), %eax
        movl  -4(%rbp), %edx
        addl  %eax, %edx
        movl  -12(%rbp), %eax
        addl  %edx, %eax
        movl  %eax, -4(%rbp)
.L5:
```

```
while (a <= 7)
{
    a = a+1;
}
if (a <= 12)
{
    a = a+b+c;
}
```

i386 Assembly

Dump file: `test.s`

```
        jmp   .L2
.L3:
        addl  $1, -4(%rbp)
.L2:
        cmpl  $7, -4(%rbp)
        jle   .L3
        cmpl  $12, -4(%rbp)
        jg    .L5
        movl  -8(%rbp), %eax
        movl  -4(%rbp), %edx
        addl  %eax, %edx
        movl  -12(%rbp), %eax
        addl  %edx, %eax
        movl  %eax, -4(%rbp)
.L5:
```

```
while (a <= 7)
{
    a = a+1;
}
if (a <= 12)
{
    a = a+b+c;
}
```


Part 6

Understanding C++ Translation

Outline

- Internal representation of a class
- Encoding function prototypes for type checking and resolving overloading
- Handling templates
- Inheritance of data
- Inheritance of functions

Internal Representation of a Class (1)

```
class A
{
  public:
    int y,z;
    void f1(int i)
      { x = f2(i)*2;}
  private:
    int x;
    int f2(int i)
      { return i+1;}
};
```

Data Memory

Code Memory

Internal Representation of a Class (1)

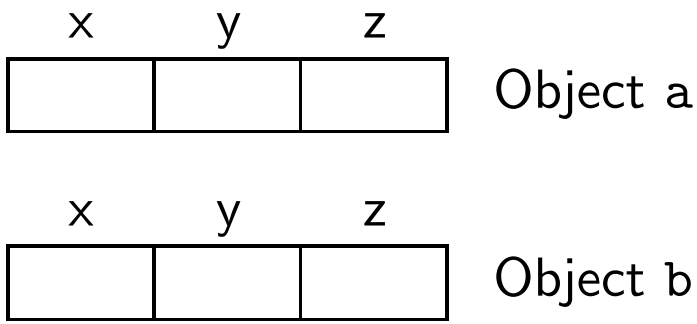
```
class A
{
  public:
    int y,z;
    void f1(int i)
      { x = f2(i)*2;}
  private:
    int x;
    int f2(int i)
      { return i+1;}
};

A a;
A b;
```

Data Memory

Code Memory

Internal Representation of a Class (1)

```
class A
{
  public:
    int y,z;
    void f1(int i)
      { x = f2(i)*2;}
  private:
    int x;
    int f2(int i)
      { return i+1;}
};

A a;
A b;
```



There is no distinction between the public and private data in memory

Internal Representation of a Class (1)

```
class A
{
  public:
    int y,z;
    void f1(int i)
      { x = f2(i)*2;}
  private:
    int x;
    int f2(int i)
      { return i+1;}
};

A a;
A b;
```

Every function with n parameters is converted to a function of $n+1$ parameter with the first parameter being the address of the object

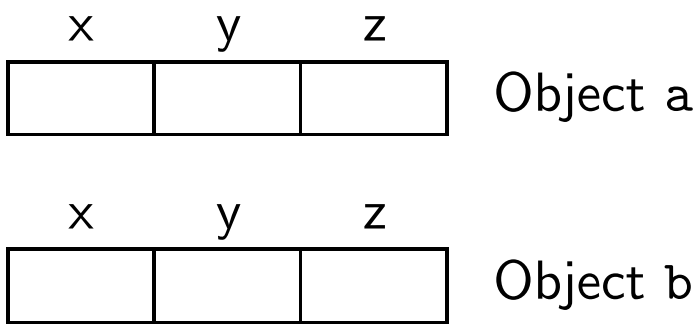
Data Memory

Code Memory

```
void A::f1(struct A * const this , int i)
{ this-> x  = A::f2(this, i)*2;}

int A::f2(struct A * const this, int i)
{ return i+1;}
```

Internal Representation of a Class (1)

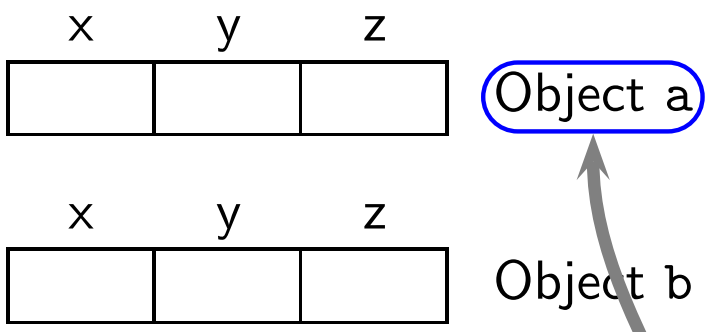
```
class A
{
  public:
    int y,z;
    void f1(int i)
      { x = f2(i)*2;}
  private:
    int x;
    int f2(int i)
      { return i+1;}
};

A a;
A b;

a.f1(5);
b.f1(10);
```




```
void A::f1(struct A * const this , int i)
{ this-> x  = A::f2(this, i)*2;}

int A::f2(struct A * const this, int i)
{ return i+1;}
```

Internal Representation of a Class (1)

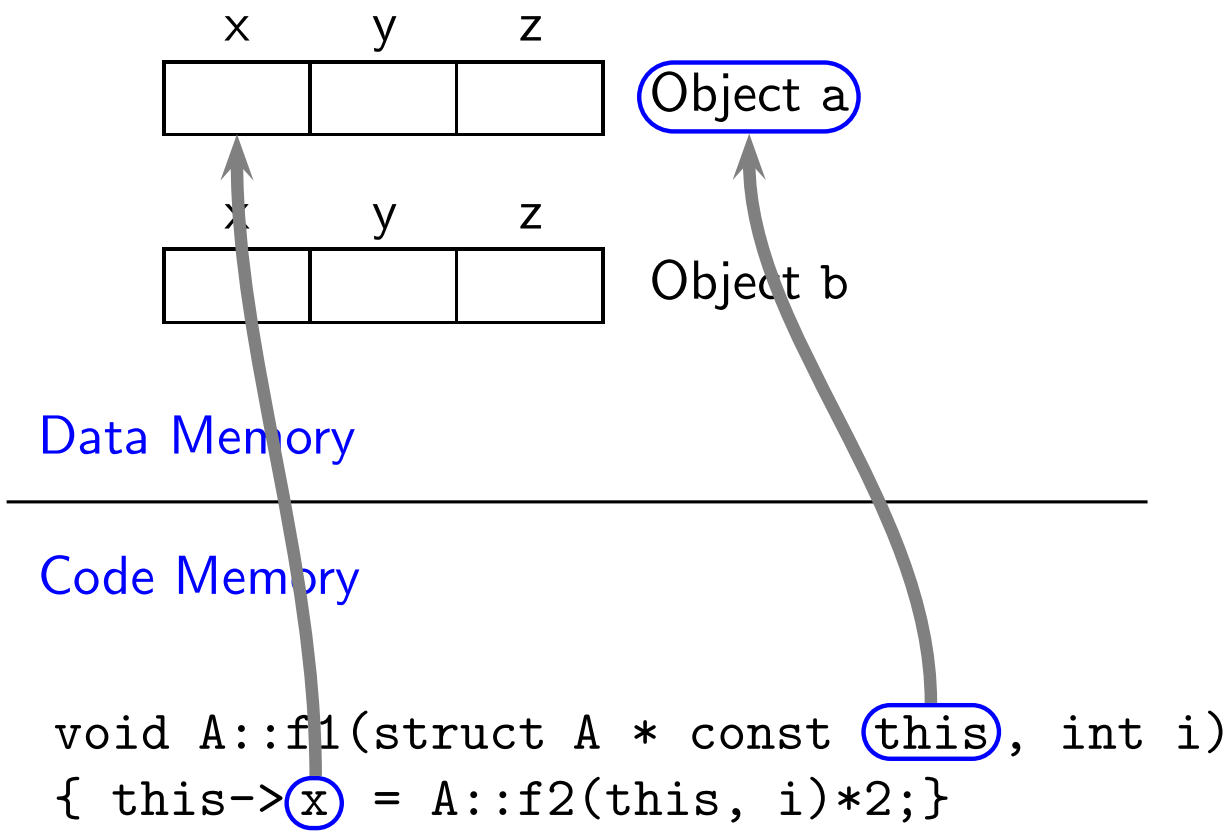
```
class A
{
  public:
    int y,z;
    void f1(int i)
      { x = f2(i)*2;}
  private:
    int x;
    int f2(int i)
      { return i+1;}
};

A a;
A b;

a.f1(5);
b.f1(10);
```




```
void A::f1(struct A * const this, int i)
{ this-> x  = A::f2(this, i)*2;}

int A::f2(struct A * const this, int i)
{ return i+1;}
```

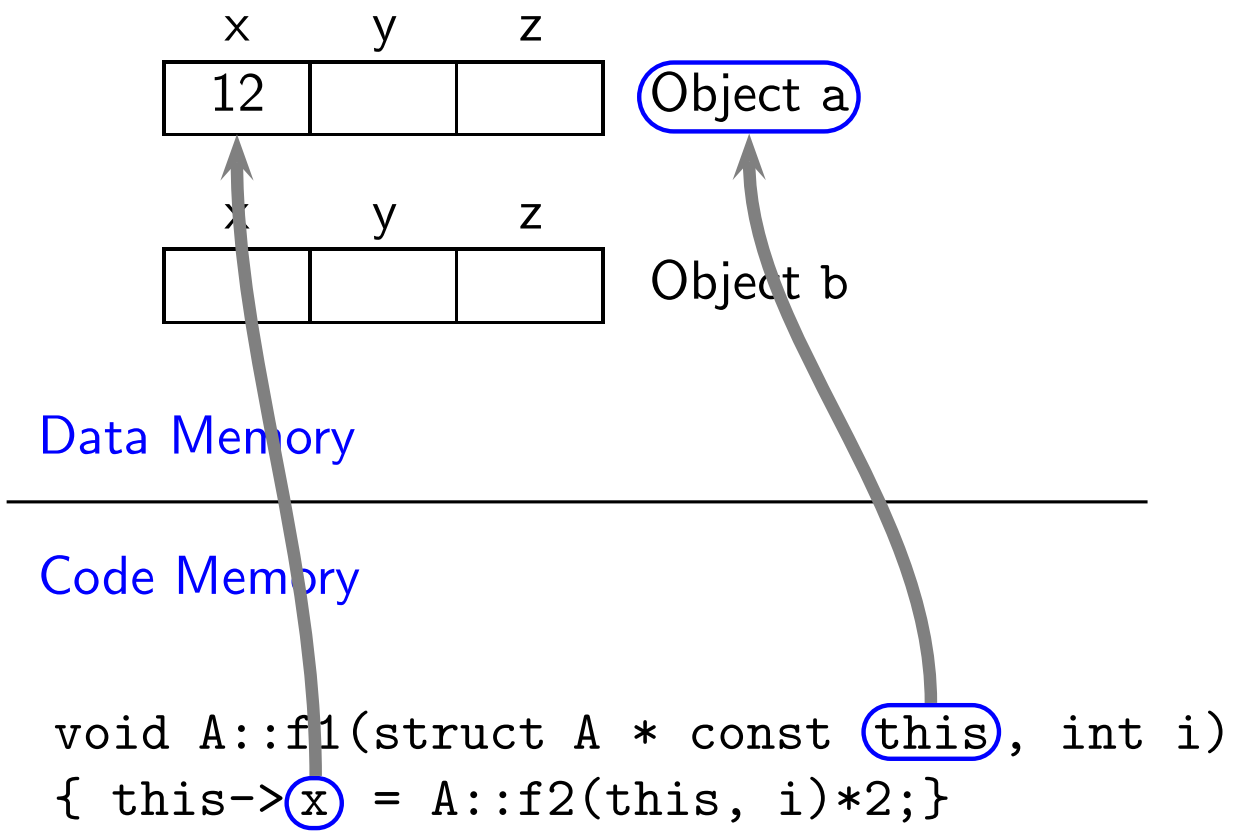
Internal Representation of a Class (1)

```
class A
{
  public:
    int y,z;
    void f1(int i)
      { x = f2(i)*2;}
  private:
    int x;
    int f2(int i)
      { return i+1;}
};

A a;
A b;

a.f1(5);
b.f1(10);
```



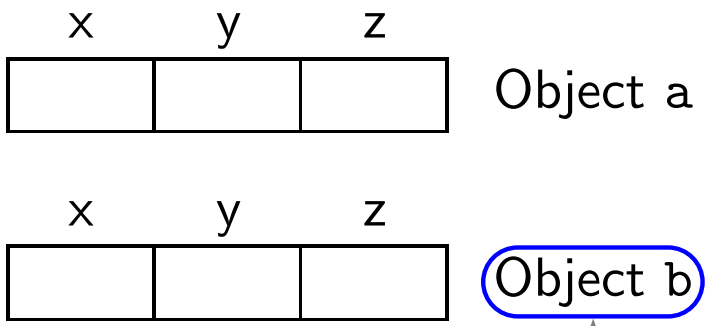
Internal Representation of a Class (1)

```
class A
{
  public:
    int y,z;
    void f1(int i)
      { x = f2(i)*2;}
  private:
    int x;
    int f2(int i)
      { return i+1;}
};

A a;
A b;

a.f1(5);
b.f1(10);
```



Internal Representation of a Class (1)

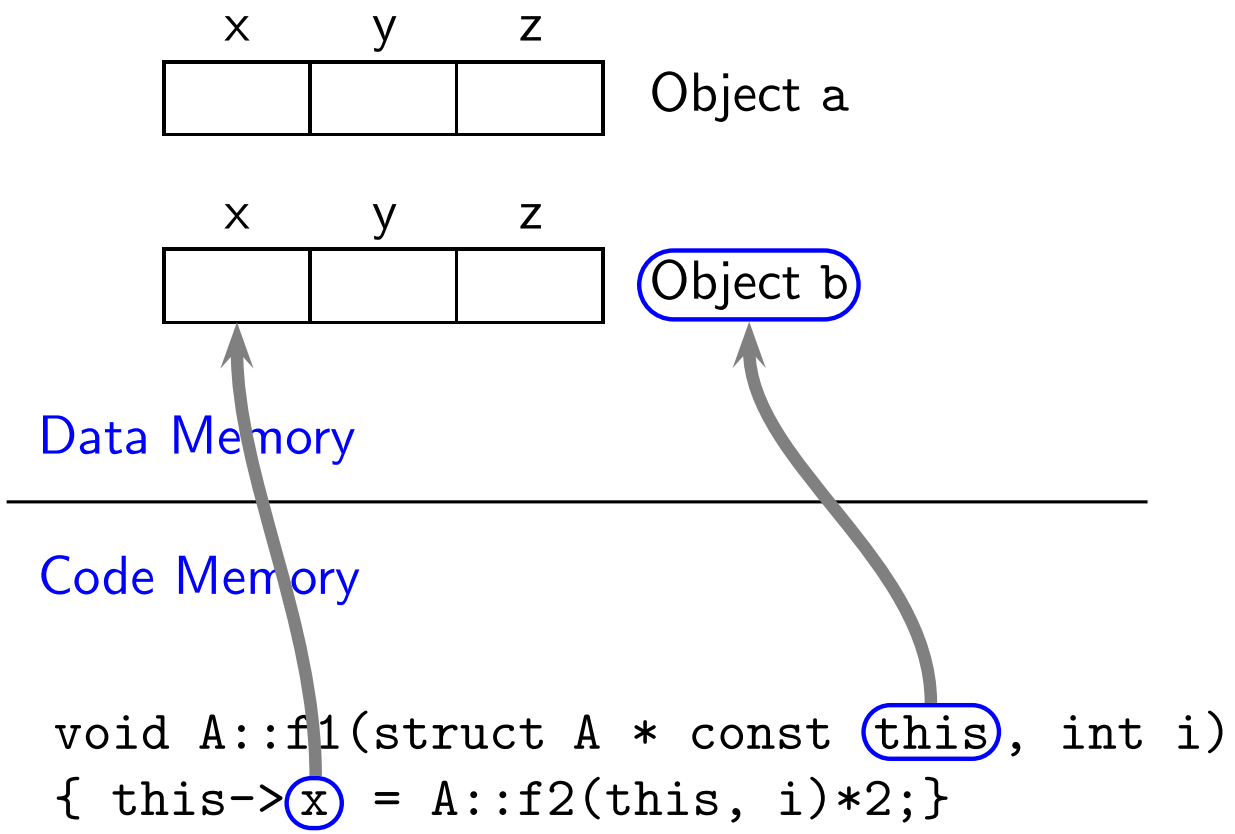
```
class A
{
  public:
    int y,z;
    void f1(int i)
      { x = f2(i)*2;}
  private:
    int x;
    int f2(int i)
      { return i+1;}
};

A a;
A b;

 a.f1(5);
a.f1(10);
```




```
void A::f1(struct A * const this, int i)
{ this-> x  = A::f2(this, i)*2;}

int A::f2(struct A * const this, int i)
{ return i+1;}
```

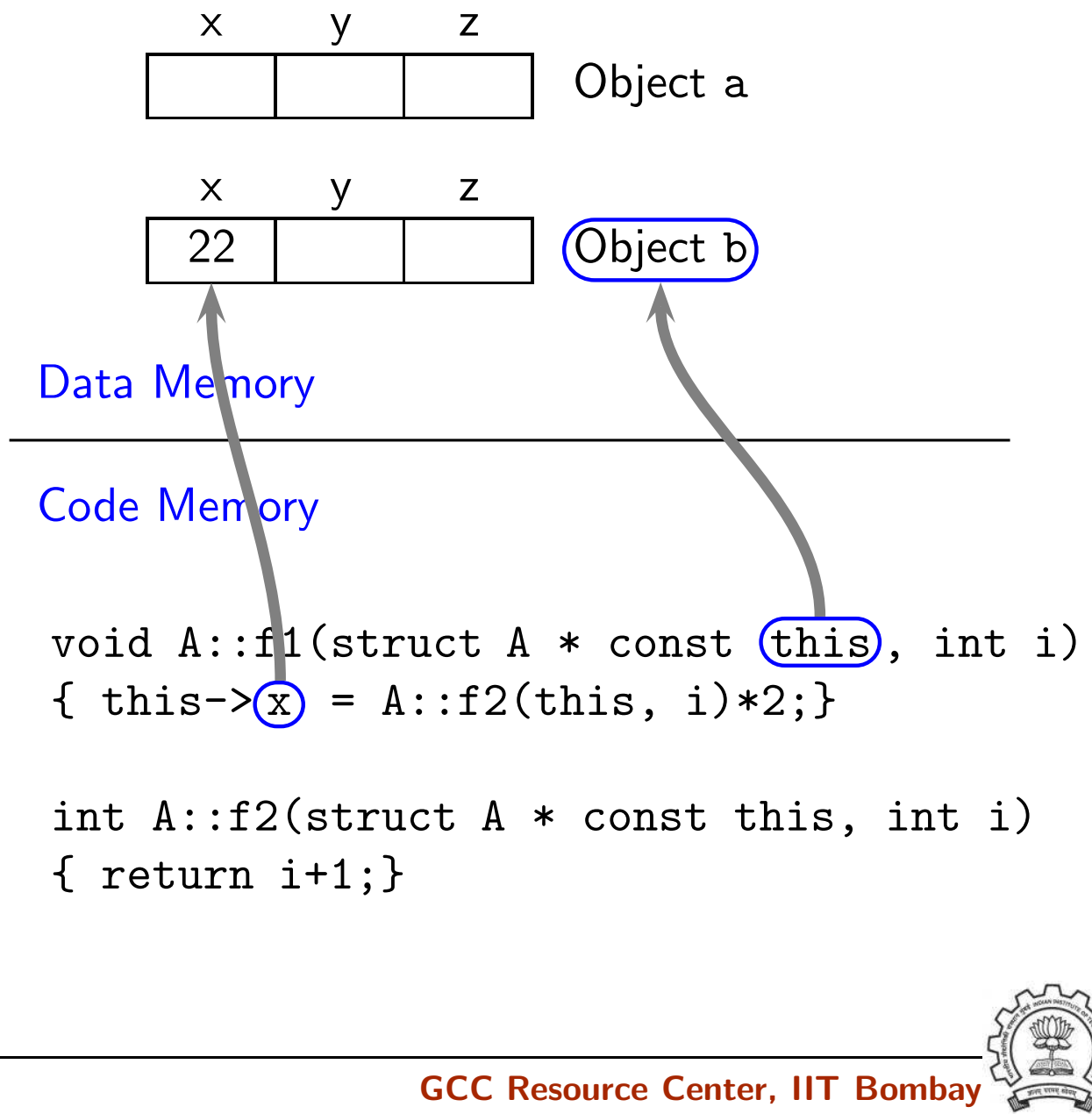
Internal Representation of a Class (1)

```
class A
{
  public:
    int y,z;
    void f1(int i)
      { x = f2(i)*2;}
  private:
    int x;
    int f2(int i)
      { return i+1;}
};

A a;
A b;

a.f1(5);
a.f1(10);
```



Internal Representation of a Class (1)

```
class A
{
  public:
    int y,z;
    void f1(int i)
      { x = f2(i)*2;}
  private:
    int x;
    int f2(int i)
      { return i+1;}
};

A a;
A b;

 a.f1(5);
a.f1(10);
```



Internal Representation of a Class (2)

dump file: test.cpp.014t.cfg

```
int main() ()
{
  struct A b;
  struct A a;
  int D.1727;

<bb 2>:
  A::f1 (&a, 5);
  A::f1 (&b, 10);
  D.1727 = 0;
  return D.1727;

}
```

```
void A::f1(int) (struct A * const this, int i)
{ int D.1730;
  int D.1729;
<bb 2>:
  D.1729 = A::f2 (this, i);
  D.1730 = D.1729 * 2;
  this->x = D.1730;
  return;
}
int A::f2(int) (struct A * const this, int i)
{ int D.1731;
<bb 2>:
  D.1731 = i + 1;
  return D.1731;
}
```

Storing Function Prototypes

Function Prototype

```
int A::add(int, int)

float A::add(int, float)

float A::add(float, int)

double A::add(float, float)
```

Storing Function Prototypes

Function Prototype	Encoding
`int A::add(int, int)`	_ZN1A3addEii
`float A::add(int, float)`	_ZN1A3addEif
`float A::add(float, int)`	_ZN1A3addEfi
`double A::add(float, float)`	_ZN1A3addEff

Storing Function Prototypes

Function Prototype	Encoding	Deconstruction
`int A::add(int, int)`	_ZN1A3addEii	_ZN 1 A 3 add E ii
`float A::add(int, float)`	_ZN1A3addEif	_ZN 1 A 3 add E if
`float A::add(float, int)`	_ZN1A3addEfi	_ZN 1 A 3 add E fi
`double A::add(float, float)`	_ZN1A3addEff	_ZN 1 A 3 add E ff

Storing Function Prototypes

Function Prototype	Encoding	Deconstruction
`int A::add(int, int)`	_ZN1A3addEii	_ZN 1 A 3 add E ii
`float A::add(int, float)`	_ZN1A3addEif	_ZN 1 A 3 add E if
`float A::add(float, int)`	_ZN1A3addEfi	_ZN 1 A 3 add E fi
`double A::add(float, float)`	_ZN1A3addEff	_ZN 1 A 3 add E ff

Fixed prefix

Storing Function Prototypes

Function Prototype	Encoding	Deconstruction
`int A::add(int, int)`	_ZN1A3addEii	_ZN 1 A 3 add E ii
`float A::add(int, float)`	_ZN1A3addEif	_ZN 1 A 3 add E if
`float A::add(float, int)`	_ZN1A3addEfi	_ZN 1 A 3 add E fi
`double A::add(float, float)`	_ZN1A3addEff	_ZN 1 A 3 add E ff

No. of characters in the class name

Storing Function Prototypes

Function Prototype	Encoding	Deconstruction
`int A::add(int, int)`	_ZN1A3addEii	_ZN 1 A 3 add E ii
`float A::add(int, float)`	_ZN1A3addEif	_ZN 1 A 3 add E if
`float A::add(float, int)`	_ZN1A3addEfi	_ZN 1 A 3 add E fi
`double A::add(float, float)`	_ZN1A3addEff	_ZN 1 A 3 add E ff

Class name

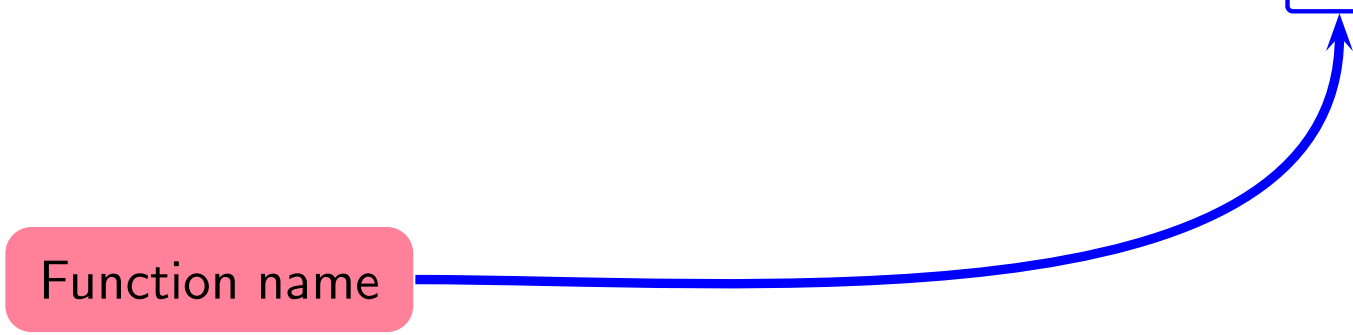
Storing Function Prototypes

Function Prototype	Encoding	Deconstruction
`int A::add(int, int)`	_ZN1A3addEii	_ZN 1 A 3 add E ii
`float A::add(int, float)`	_ZN1A3addEif	_ZN 1 A 3 add E if
`float A::add(float, int)`	_ZN1A3addEfi	_ZN 1 A 3 add E fi
`double A::add(float, float)`	_ZN1A3addEff	_ZN 1 A 3 add E ff

No. of characters in the function name

Storing Function Prototypes

Function Prototype	Encoding	Deconstruction
`int A::add(int, int)`	_ZN1A3addEii	_ZN 1 A 3 add E ii
`float A::add(int, float)`	_ZN1A3addEif	_ZN 1 A 3 add E if
`float A::add(float, int)`	_ZN1A3addEfi	_ZN 1 A 3 add E fi
`double A::add(float, float)`	_ZN1A3addEff	_ZN 1 A 3 add E ff



Storing Function Prototypes

Function Prototype	Encoding	Deconstruction
`int A::add(int, int)`	_ZN1A3addEii	_ZN 1 A 3 add E ii
`float A::add(int, float)`	_ZN1A3addEif	_ZN 1 A 3 add E if
`float A::add(float, int)`	_ZN1A3addEfi	_ZN 1 A 3 add E fi
`double A::add(float, float)`	_ZN1A3addEff	_ZN 1 A 3 add E ff

End of function name

Storing Function Prototypes

Function Prototype	Encoding	Deconstruction
`int A::add(int, int)`	_ZN1A3addEii	_ZN 1 A 3 add E ii
`float A::add(int, float)`	_ZN1A3addEif	_ZN 1 A 3 add E if
`float A::add(float, int)`	_ZN1A3addEfi	_ZN 1 A 3 add E fi
`double A::add(float, float)`	_ZN1A3addEff	_ZN 1 A 3 add E ff

Types of the arguments

Representing Classes with Templates

```
template <class T>
T GetMax (T a, T b)
{
  T result;
  result = (a>b)?a:b;
  return result;
}
```

Definition

Representing Classes with Templates

```
template <class T>
T GetMax (T a, T b)
{
  T result;
  result = (a>b)?a:b;
  return result;
}
```

Definition

```
k = GetMax<int> (i, j);
n = GetMax<long int> (l, m);
```

Instantiation

Representing Classes with Templates

```
template <class T>
T GetMax (T a, T b)
{
  T result;
  result = (a>b)?a:b;
  return result;
}
```

Definition

```
k = GetMax<int> (i, j);
n = GetMax<long int> (l, m);
```

Instantiation

Internal representation for each instantiation

```
;; Function T GetMax(T,T) [with T = long int] (_Z6GetMaxIlET_S0_S0_)
T GetMax(T, T) [with T = long int] (long int a, long int b)

;; Function T GetMax(T,T) [with T = int] (_Z6GetMaxIiET_S0_S0_)
T GetMax(T, T) [with T = int] (int a, int b)
```

Representing Classes with Templates

```
template <class T>
T GetMax (T a, T b)
{
  T result;
  result = (a>b)?a:b;
  return result;
}
```

Definition

```
k = GetMax<int> (i, j);
n = GetMax<long int> (l, m);
```

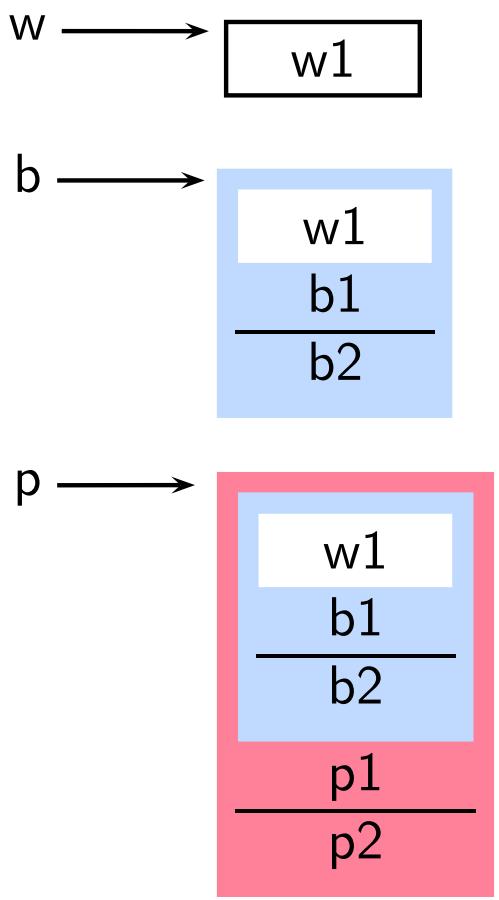
Instantiation

Internal representation for each instantiation

```
;; Function T GetMax(T,T) [with T = long int] (_Z6GetMaxIlET_S0_S0_)
T GetMax(T, T) [with T = long int] (long int a, long int b)

;; Function T GetMax(T,T) [with T = int] (_Z6GetMaxIiET_S0_S0_)
T GetMax(T, T) [with T = int] (int a, int b)
```

Representing Inheritance of Data

```
class White
{ public:
    int  w1;
};
class Blue : public White
{ public:
     int b1;
     int b2;
};
class Pink : public Blue
{ public:
     int p1;
     int p2;
};
White w;
Blue b;
Pink p;
```



Representing Inheritance of Functions

- Non-virtual functions are inherited much like data members

  Type of the object pointer does not matter

- Virtual functions create interesting possibilities based on the object to which a pointer points to

  A pointer to a base class object may point to an object of any derived class in the class hierarchy

An Example with Virtual Functions

```
class A
{  public:
     virtual void f()
     virtual void f(string i)
     virtual void g()
};

class B : public A
{  public:
     virtual void g()
              void f()
};

class C : public B
{  public:
              void f()
};
```

An Example with Virtual Functions

```
class A
{  public:
     virtual void f() {cout << "\tA:f" << endl;}
     virtual void f(string i) {cout << "\tA:f." << i << endl;}
     virtual void g() {cout << "\tA:g" << endl;}
};

class B : public A
{  public:
     virtual void g() {cout << "\tB:g" << endl;}
              void f() {cout << "\tB:f" << endl;}
};

class C : public B
{  public:
              void f() {cout<< "\tC:f" << endl;}
};
```

An Example with Virtual Functions

```
class A
{  public:
     virtual void f() {cout << "\tA:f" << endl;}
     virtual void f(string i) {cout << "\tA:f." << i << endl;}
     virtual void g() {cout << "\tA:g" << endl;}
};

class B : public A
{  public:
     virtual void g() {cout << "\tB:g" << endl;}
              void f() {cout << "\tB:f" << endl;}
};

class C : public B
{  public:
              void f() {cout<< "\tC:f" << endl;}
};
```

Examining the Behaviour of Virtual Functions

Program	Output

```
A a, *array[3];
B b;
C c;

array[0]=&a;
array[1]=&b;
array[2]=&c;

for (int i=0;i<3;i++)
{
  cout << i ;
  array[i]->f();
  array[i]->f("x");
  array[i]->g();
}
```

```
0       A:f
        A:f.x
        A:g
1       B:f
        A:f.x
        B:g
2       C:f
        A:f.x
        B:g
```

Virtual Function Resolution

- Partially static and partially dynamic activity
- At compile time, a compiler creates a virtual function table for each class
- At run time, a pointer may point to an object of any derived class
- Compiler generates code the pick up the appropriate function by indexing into the virtual table to each class

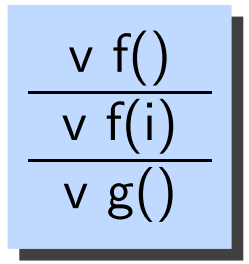
  (the exact virtual table depends on the pointee object)

Virtual Function Resolution Requires Dynamic Information

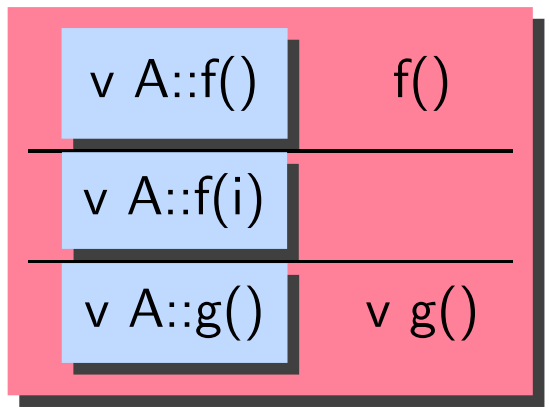
```
A *p;

p = ...
p->f();
```

```
B *q;

q = ...
q->f();
```

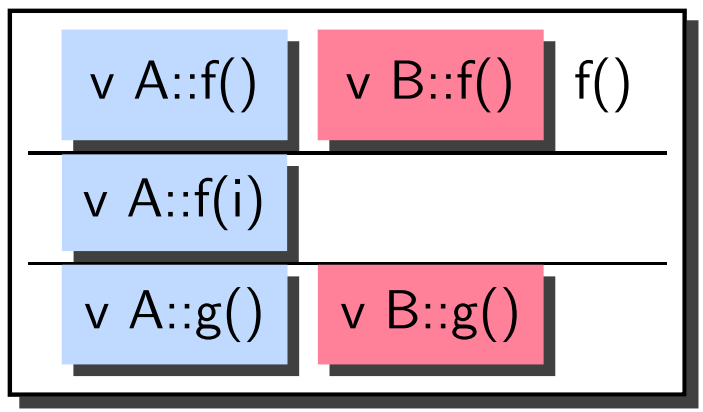
```
C *r;

r = ...
r->f();
```



A



B



C

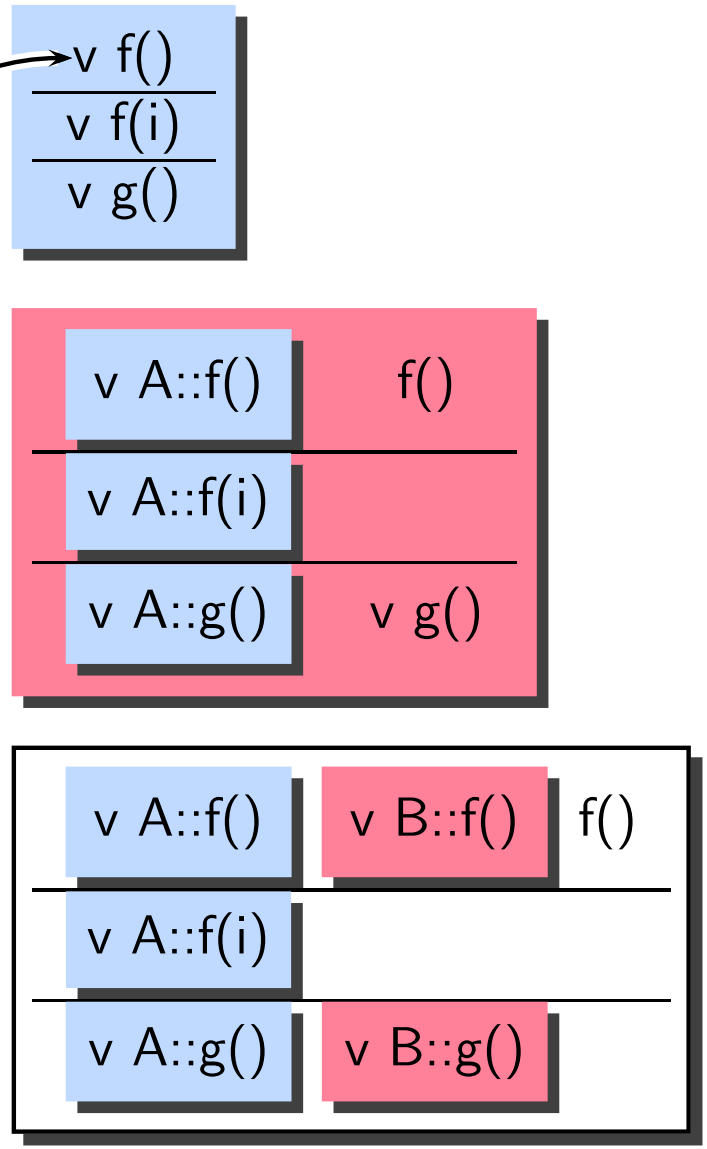
Virtual Function Resolution Requires Dynamic Information

```
A *p;

p = ...
p->f();
```

```
B *q;

q = ...
```

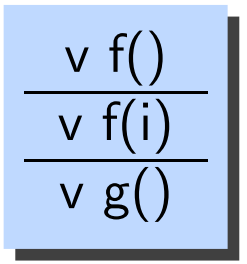
In general, at compile time we do not know the class of the pointee of p

```
r->f();
```

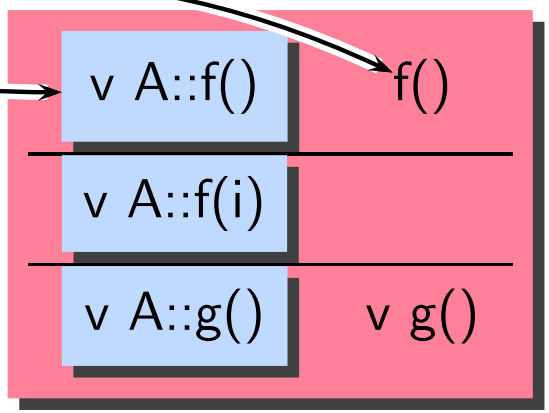


Virtual Function Resolution Requires Dynamic Information

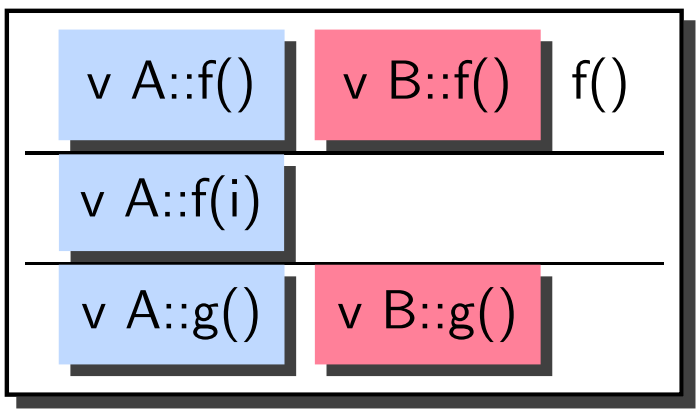
```
A *p;

p = ...
p->f();

B *q;

q = ...

r->f();
```



A



B

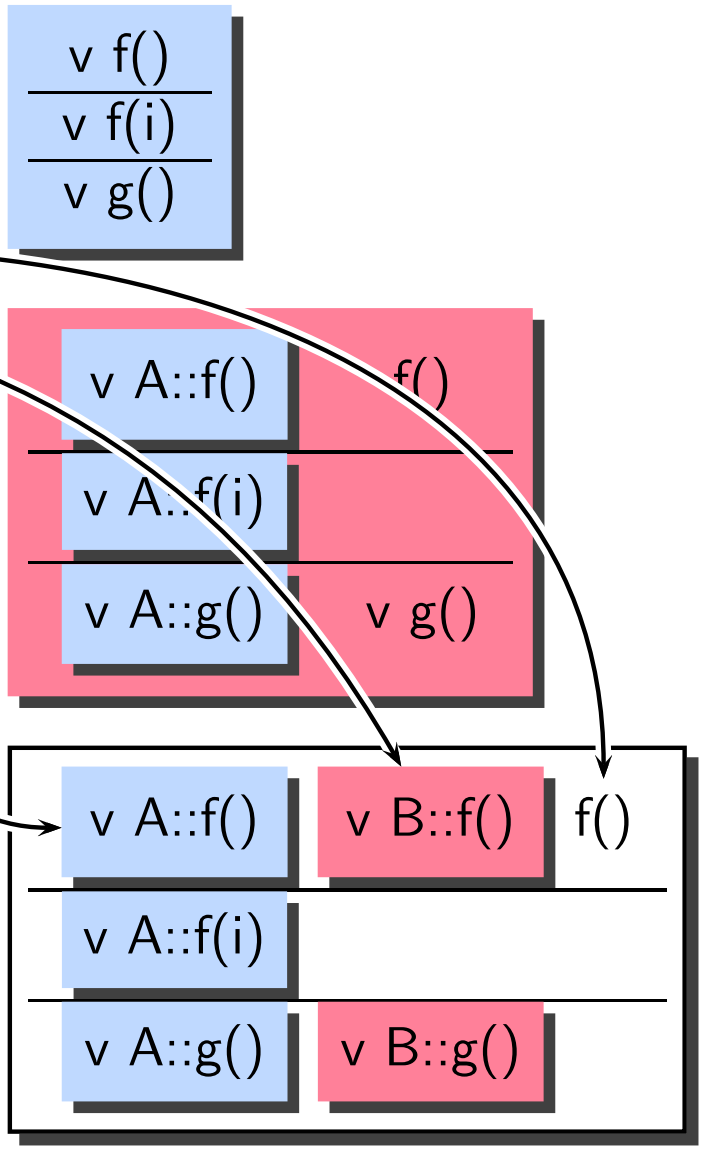


C

In general, at compile time we do not know the class of the pointee of p

Virtual Function Resolution Requires Dynamic Information

```
A *p;

p = ...
p->f();

B *q;

q = ...

r->f();
```



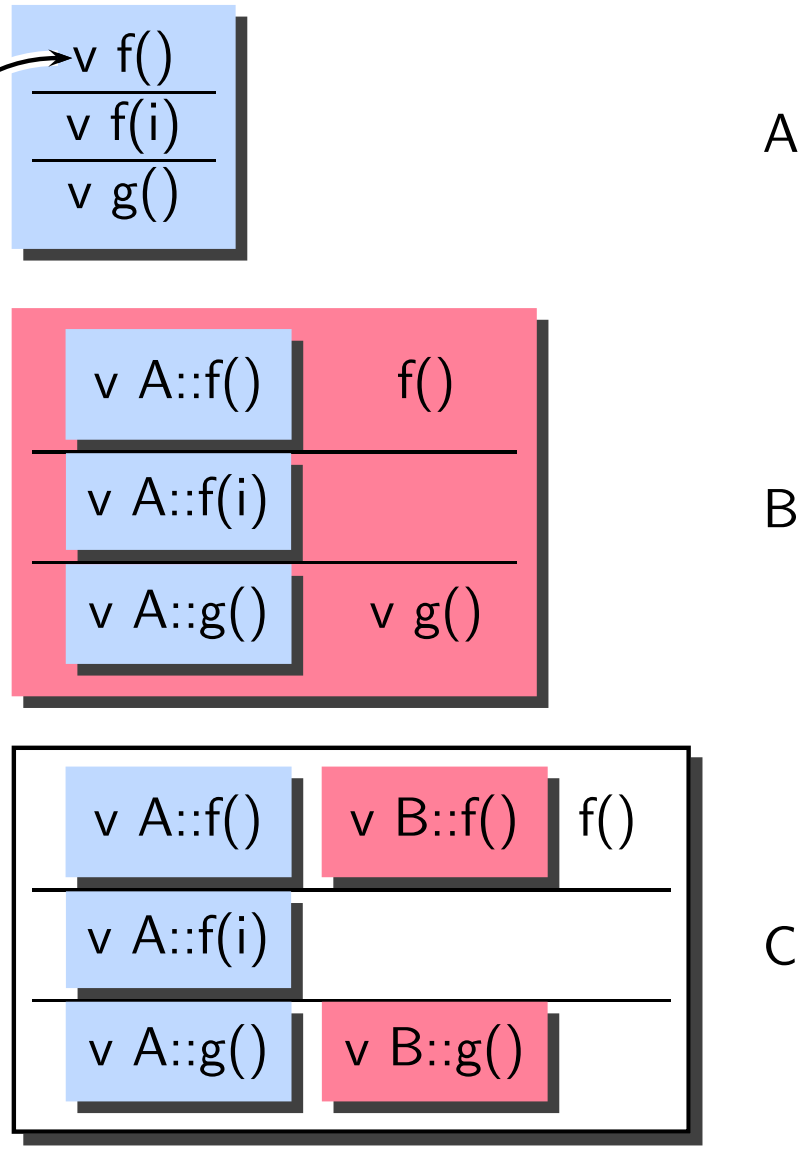
In general, at compile time we do not know the class of the pointee of p

Virtual Function Resolution Requires Dynamic Information

```
A *p;

p = ...
p->f();
```

```
B *q;

q = ...
q->f();
```

```
C *r;

r = ...
r->f();
```

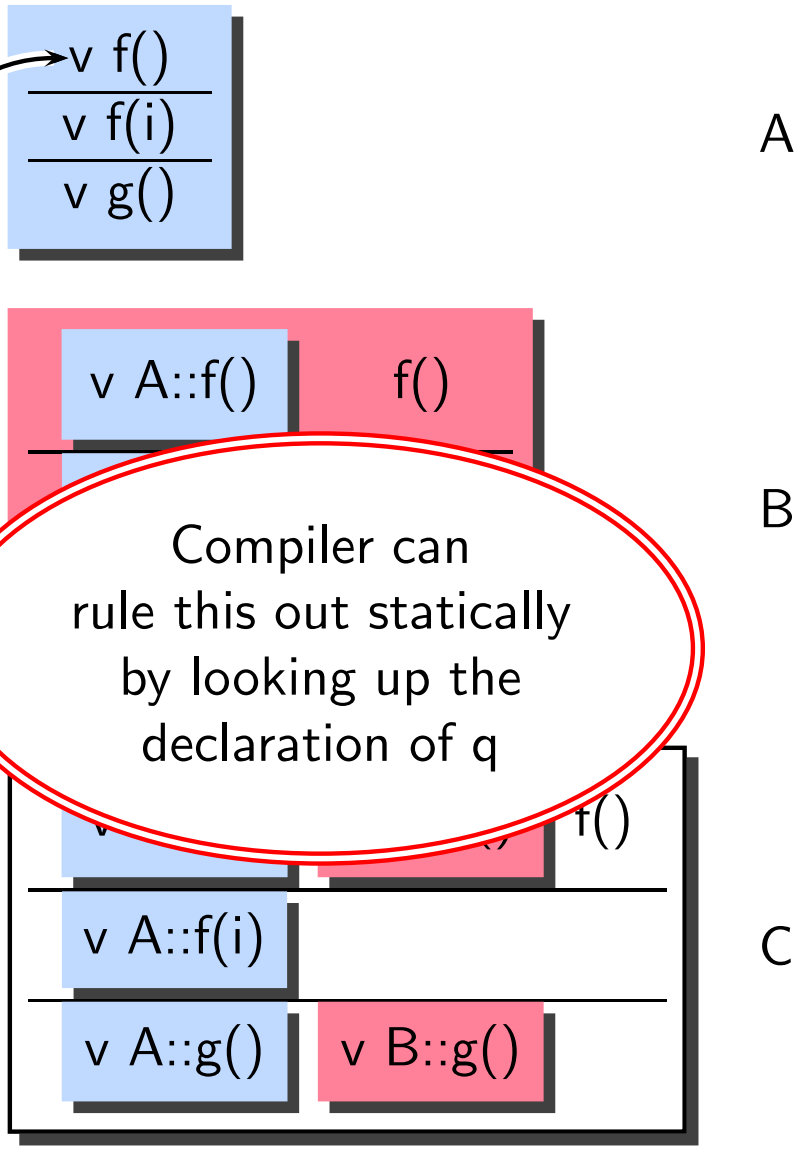


Virtual Function Resolution Requires Dynamic Information

```
A *p;

p = ...
p->f();
```

```
B *q;

q = ...
q->f();
```

```
C *r;

r = ...
r->f();
```

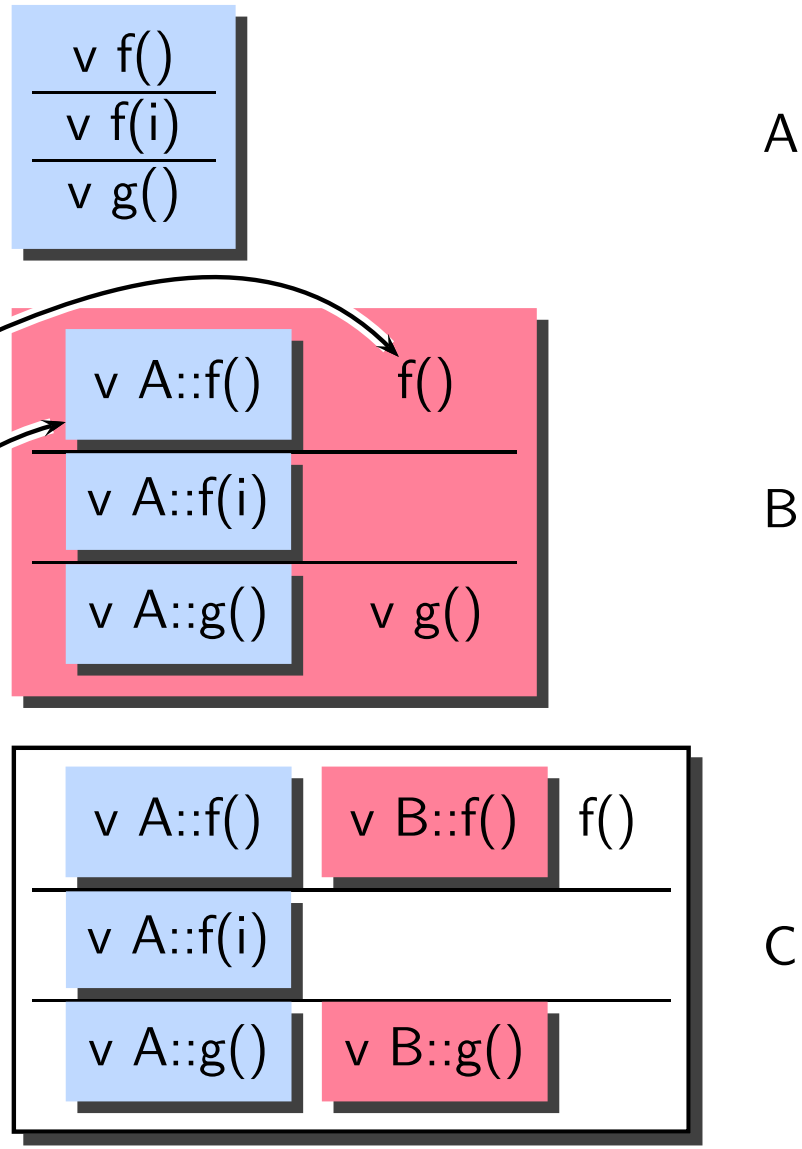


Virtual Function Resolution Requires Dynamic Information

```
A *p;

p = ...
p->f();
```

```
B *q;

q = ..
q->f();
```

```
C *r;

r = ...
r->f();
```

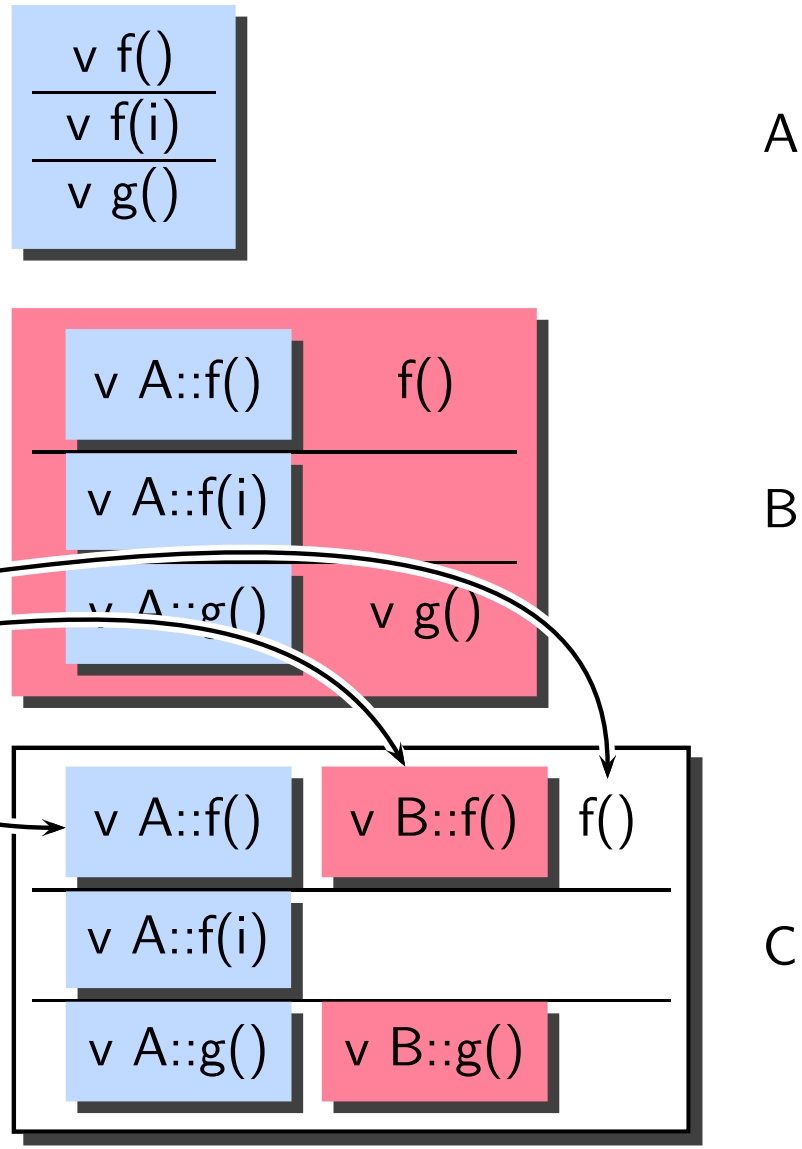


Virtual Function Resolution Requires Dynamic Information

```
A *p;

p = ...
p->f();
```

```
B *q;

q = ...
q->f();
```

```
C *r;

r = ...
r->f();
```

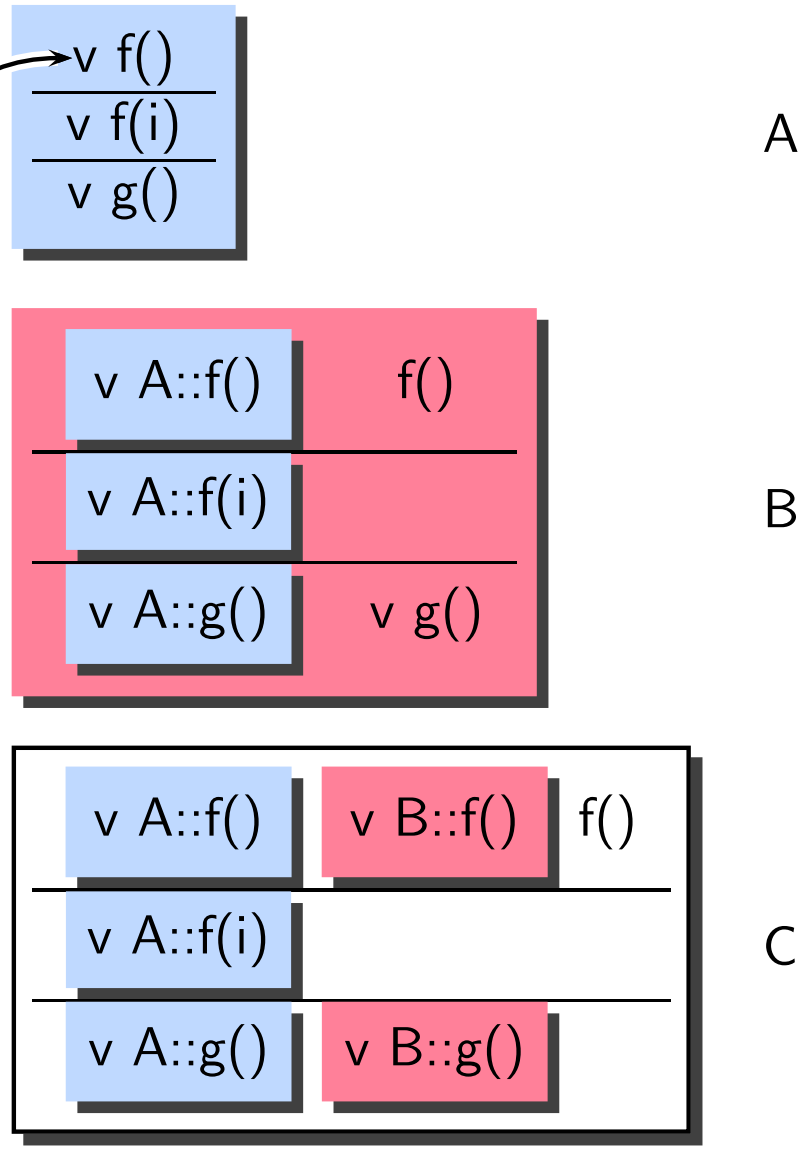


Virtual Function Resolution Requires Dynamic Information

```
A *p;

p = ...
p->f();
```

```
B *q;

q = ...
q->f();
```

```
C *r;

r = ...
r->f();
```

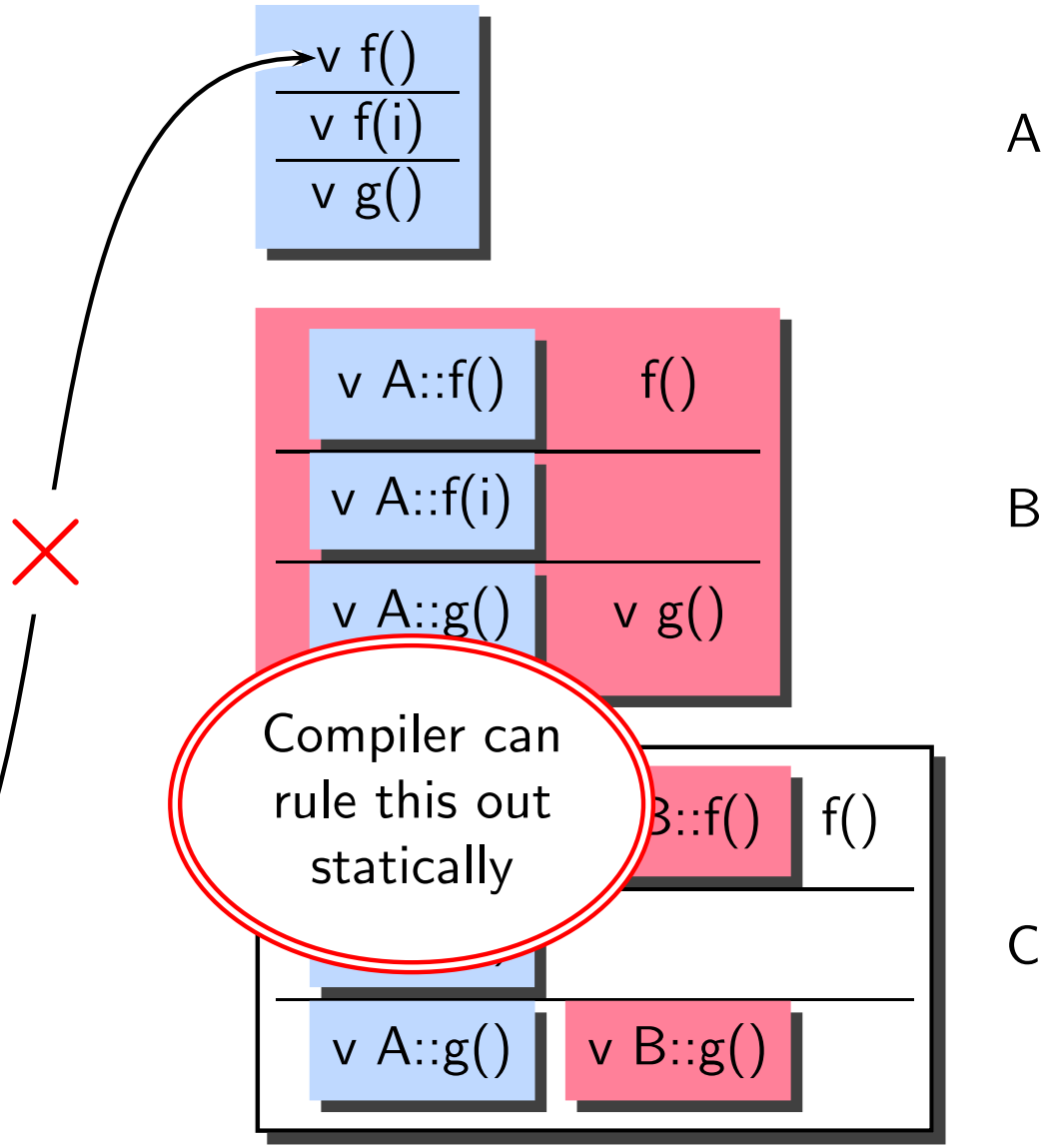


Virtual Function Resolution Requires Dynamic Information

```
A *p;

p = ...
p->f();
```

```
B *q;

q = ...
q->f();
```

```
C *r;

r = ...
r->f();
```

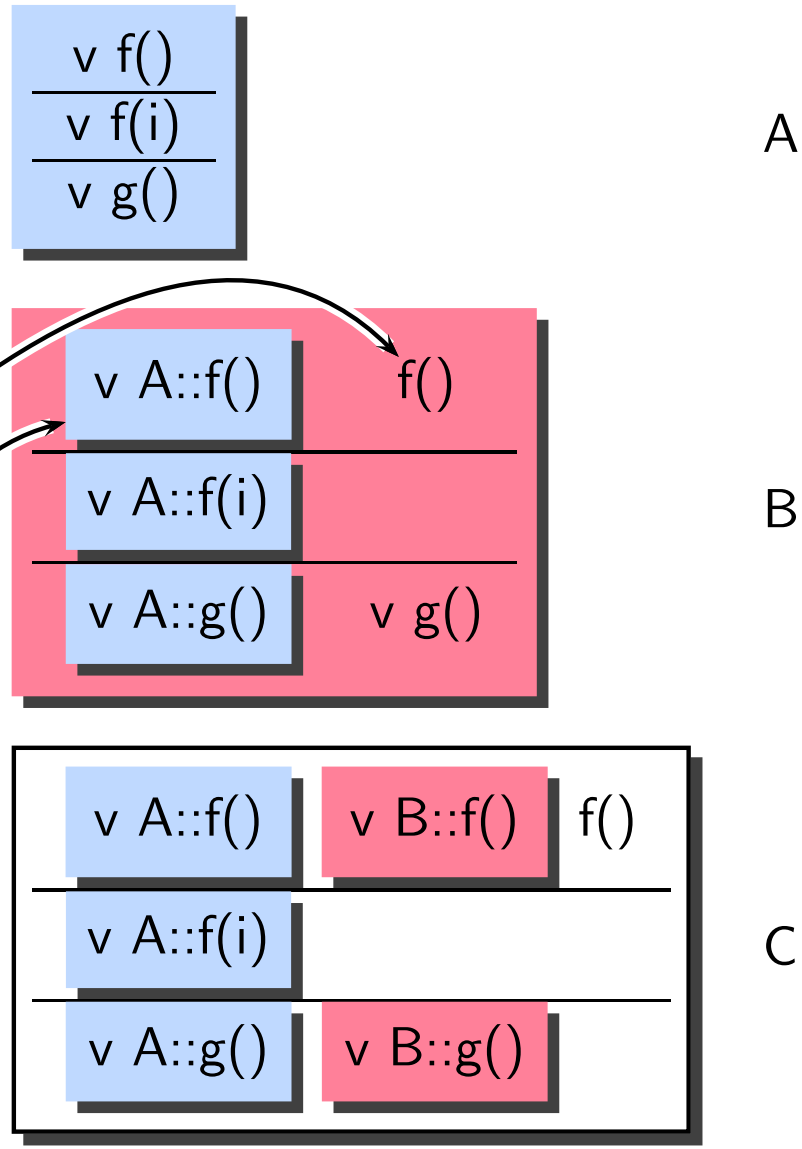


Virtual Function Resolution Requires Dynamic Information

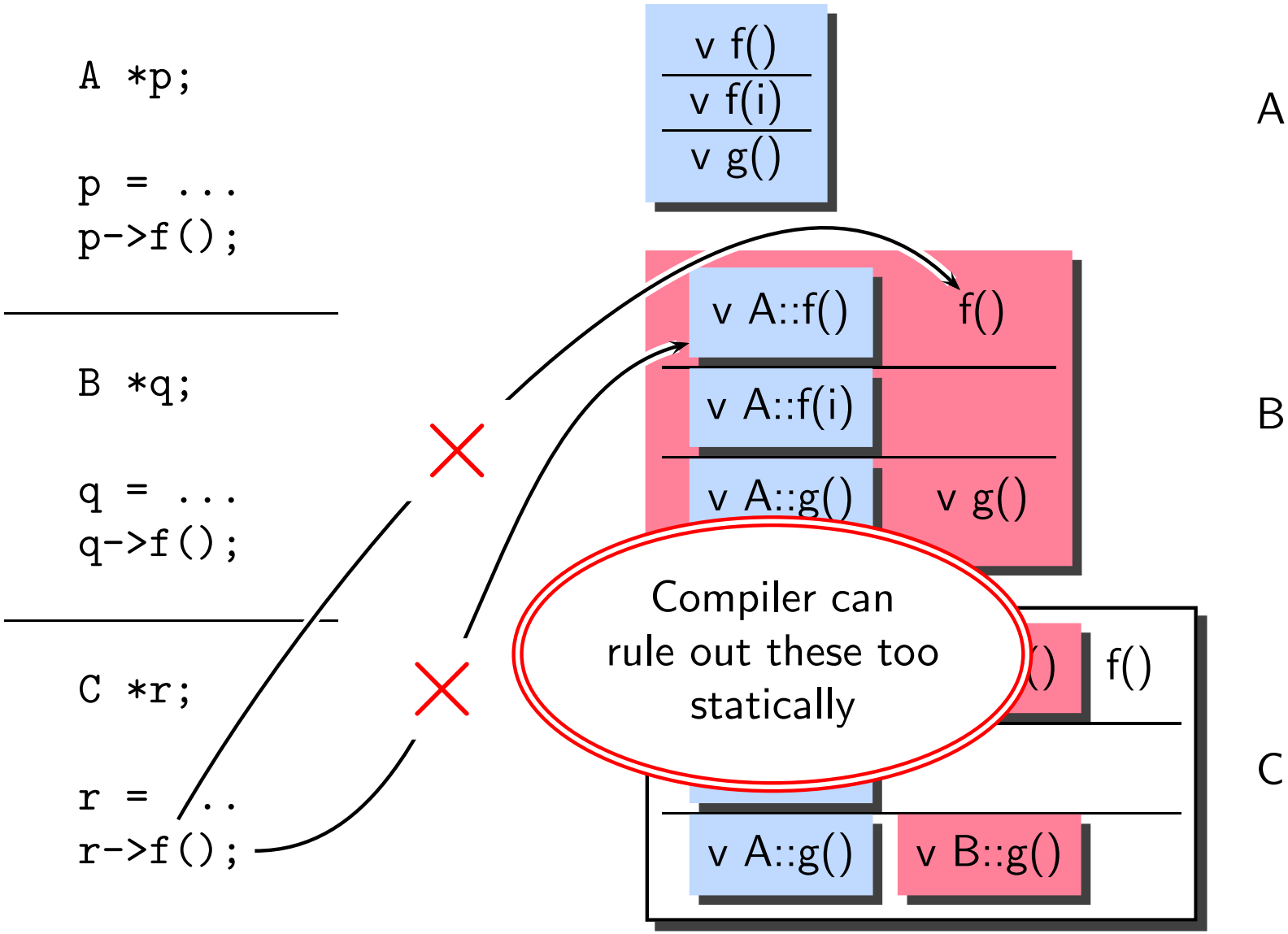
```
A *p;

p = ...
p->f();
```

```
B *q;

q = ...
q->f();
```

```
C *r;

r = ..
r->f();
```



Virtual Function Resolution Requires Dynamic Information

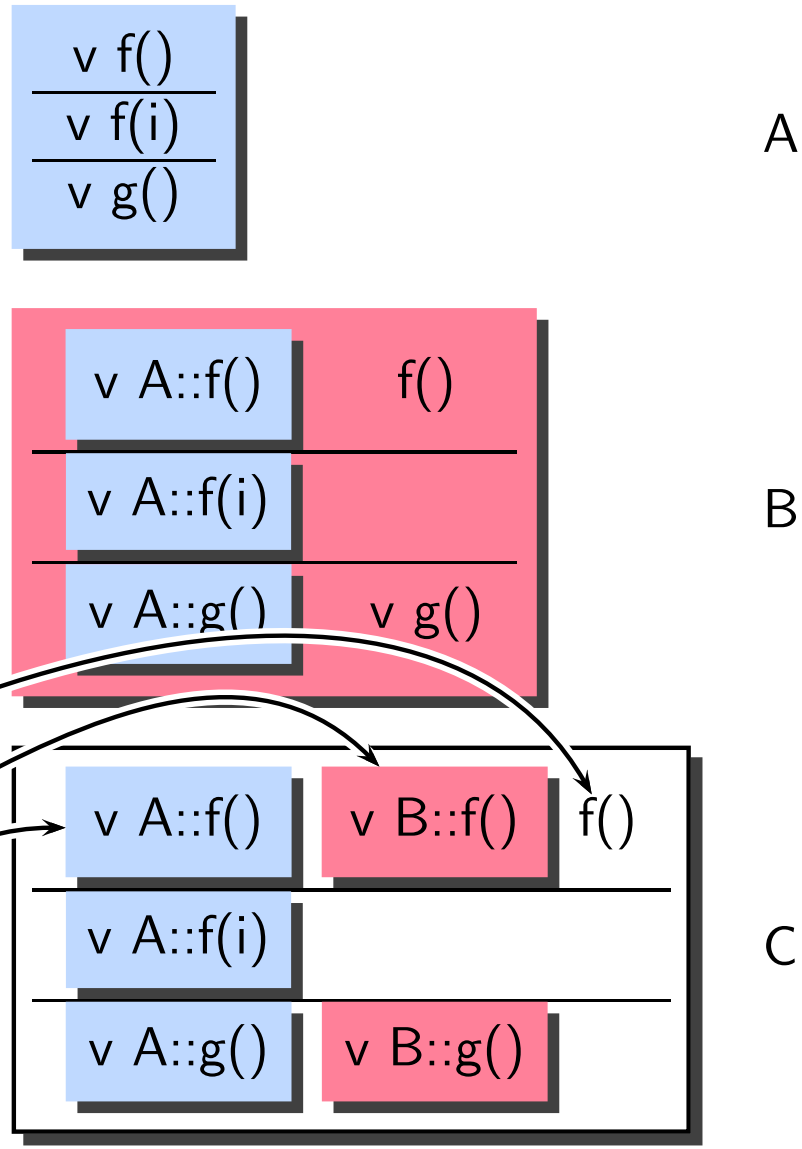


Virtual Function Resolution Requires Dynamic Information

```
A *p;

p = ...
p->f();
```

```
B *q;

q = ...
q->f();
```

```
C *r;

r = ...
r->f();
```



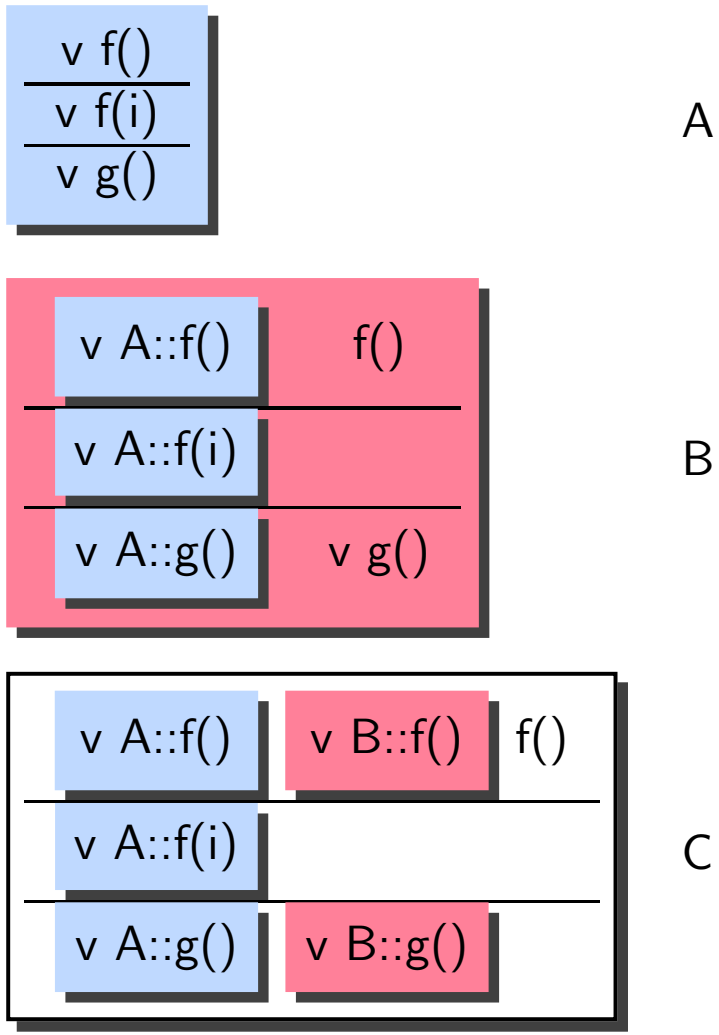
Non-Virtual Functions Do Not Require Dynamic Information

Non-virtual function = a function which is not virtual in *any* class in a hierarchy

```
A *p;
p = ...
p->f();
```

```
B *q;
q = ...
q->f();
```

```
C *r;
r = ...
r->f();
```



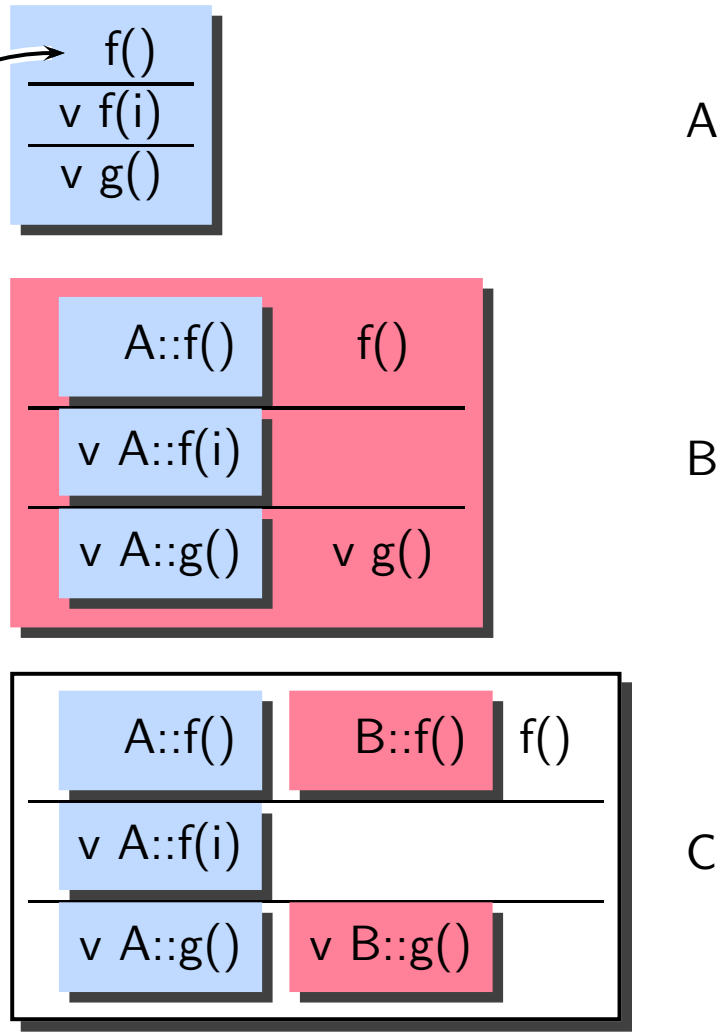
Non-Virtual Functions Do Not Require Dynamic Information

Non-virtual function = a function which is not virtual in *any* class in a hierarchy

```
A *p;
p = ...
p->f();
```

```
B *q;
q = ...
q->f()
```

In general, at compile time we do not know the class of the pointee of p



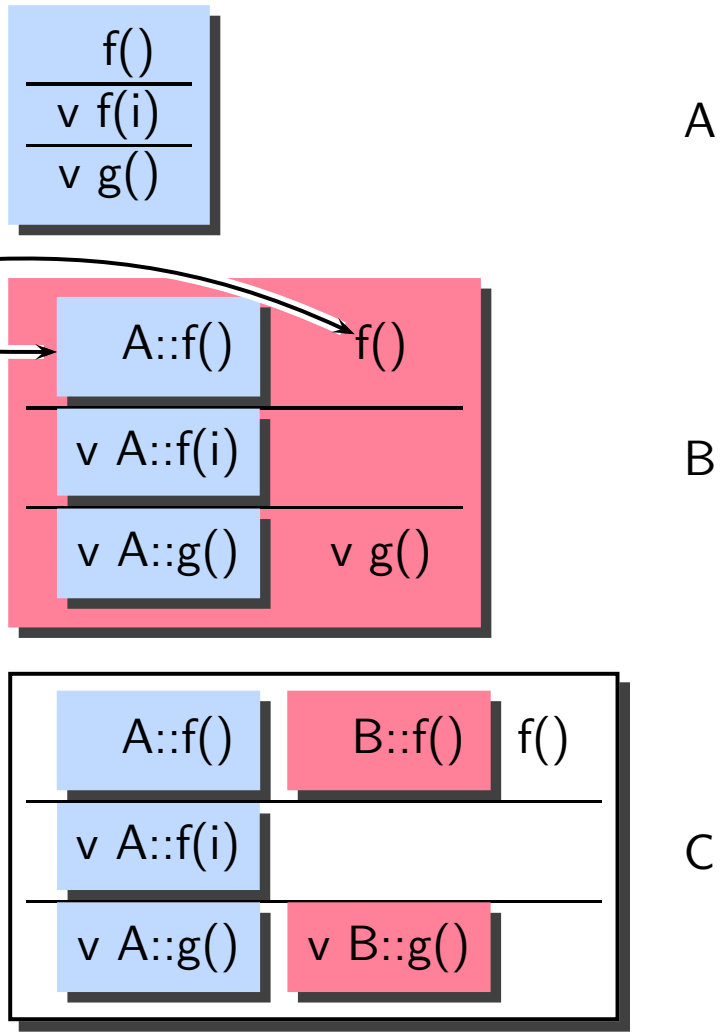
Non-Virtual Functions Do Not Require Dynamic Information

Non-virtual function = a function which is not virtual in *any* class in a hierarchy

```
A *p;
p = ...
p->f();
```

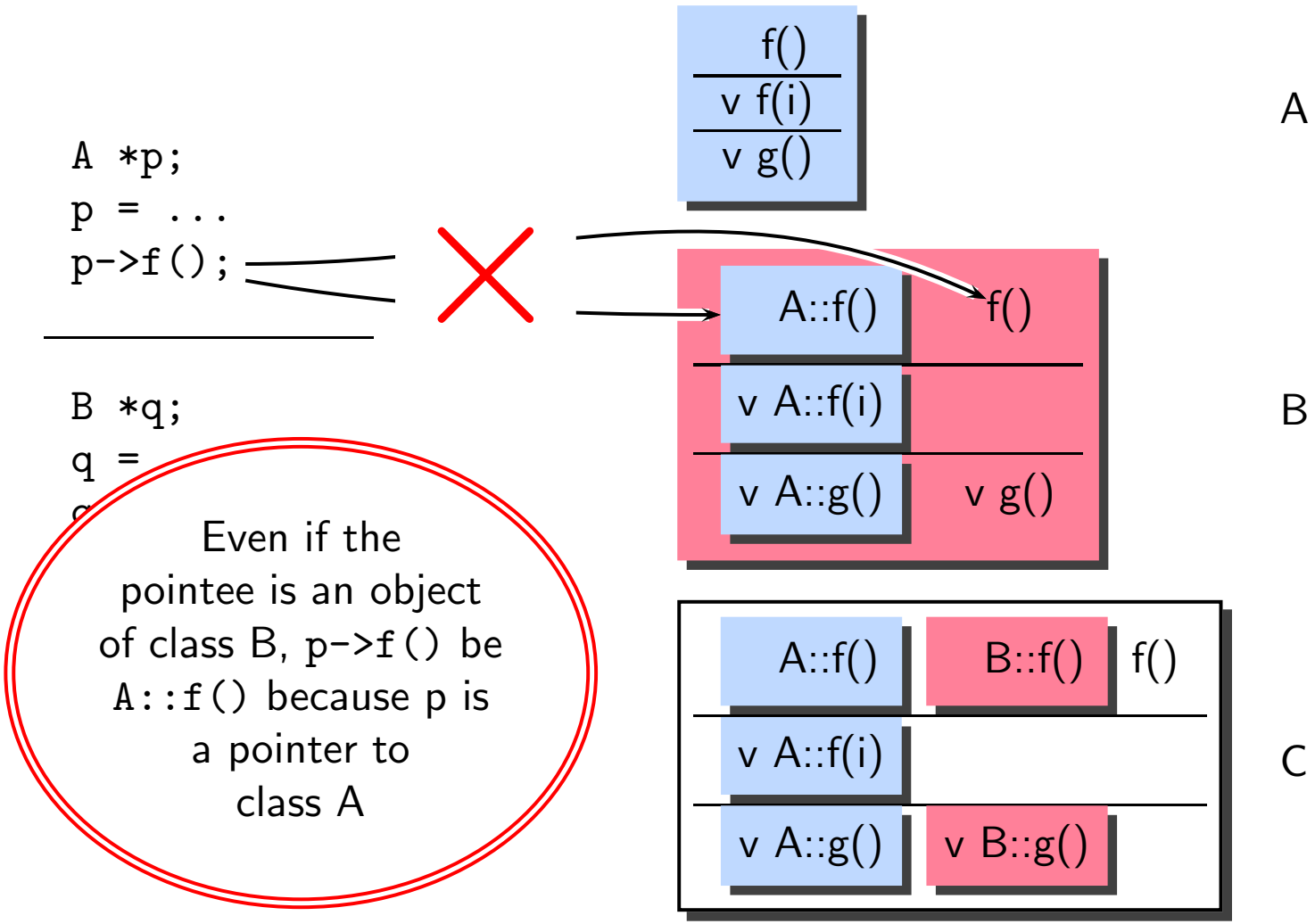
```
B *q;
q = ...
q->f();
```

```
C *r;
r = ...
r->f();
```



Non-Virtual Functions Do Not Require Dynamic Information

Non-virtual function = a function which is not virtual in *any* class in a hierarchy



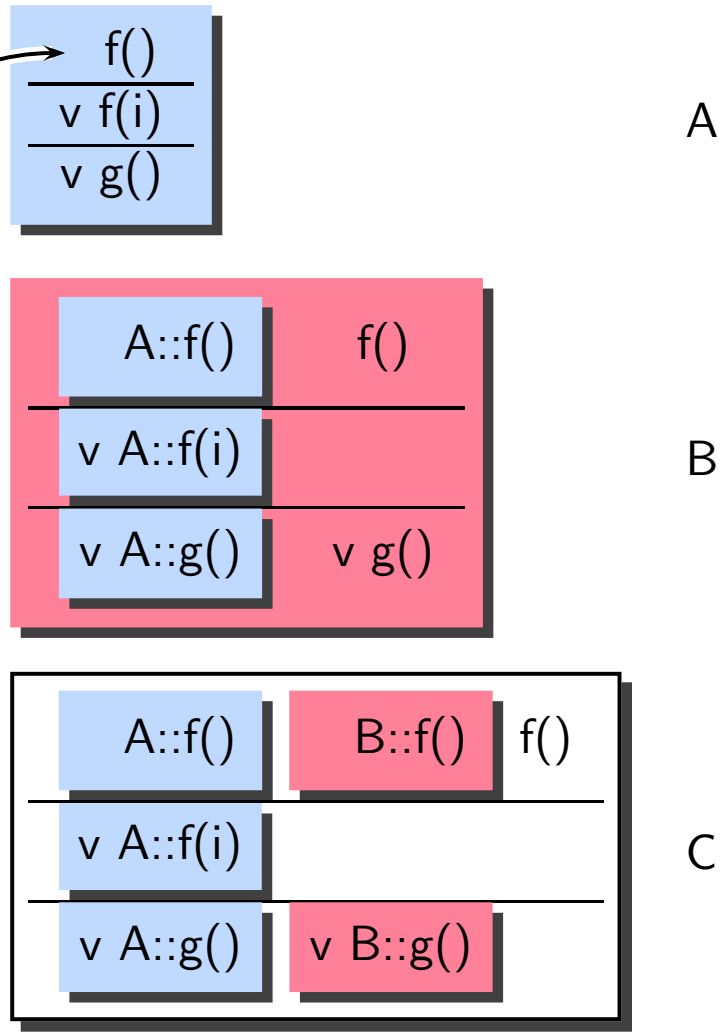
Non-Virtual Functions Do Not Require Dynamic Information

Non-virtual function = a function which is not virtual in *any* class in a hierarchy

```
A *p;
p = ...
p->f();
```

```
B *q;
q =
```

Even if the pointee is an object of class B, `p->f()` be `A::f()` because p is a pointer to class A



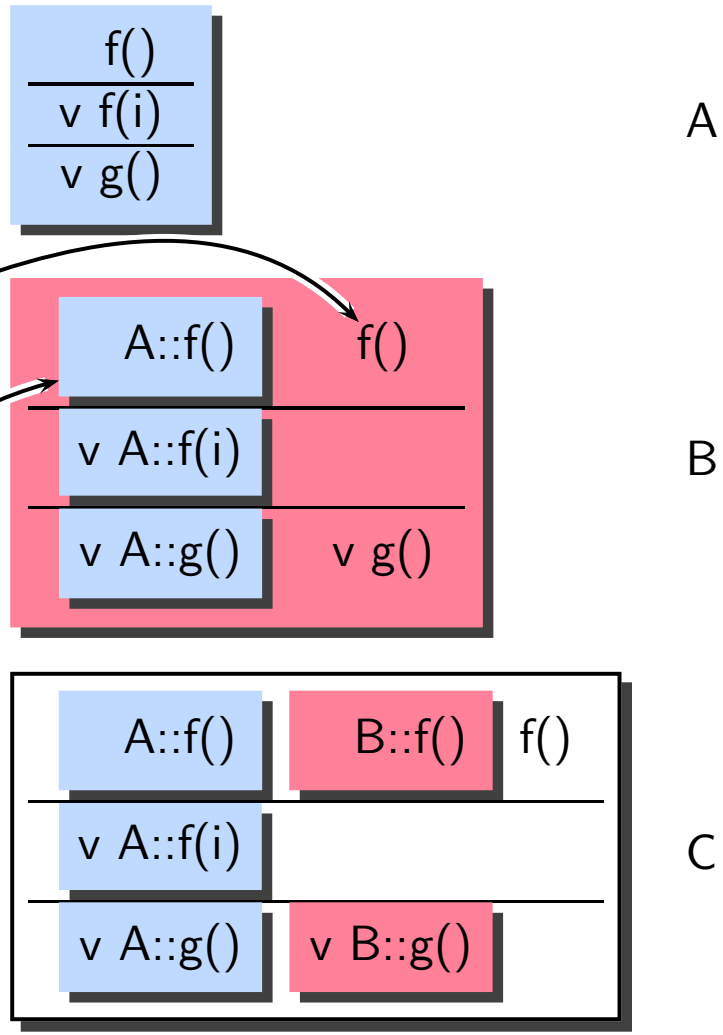
Non-Virtual Functions Do Not Require Dynamic Information

Non-virtual function = a function which is not virtual in *any* class in a hierarchy

```
A *p;
p = ...
p->f();
```

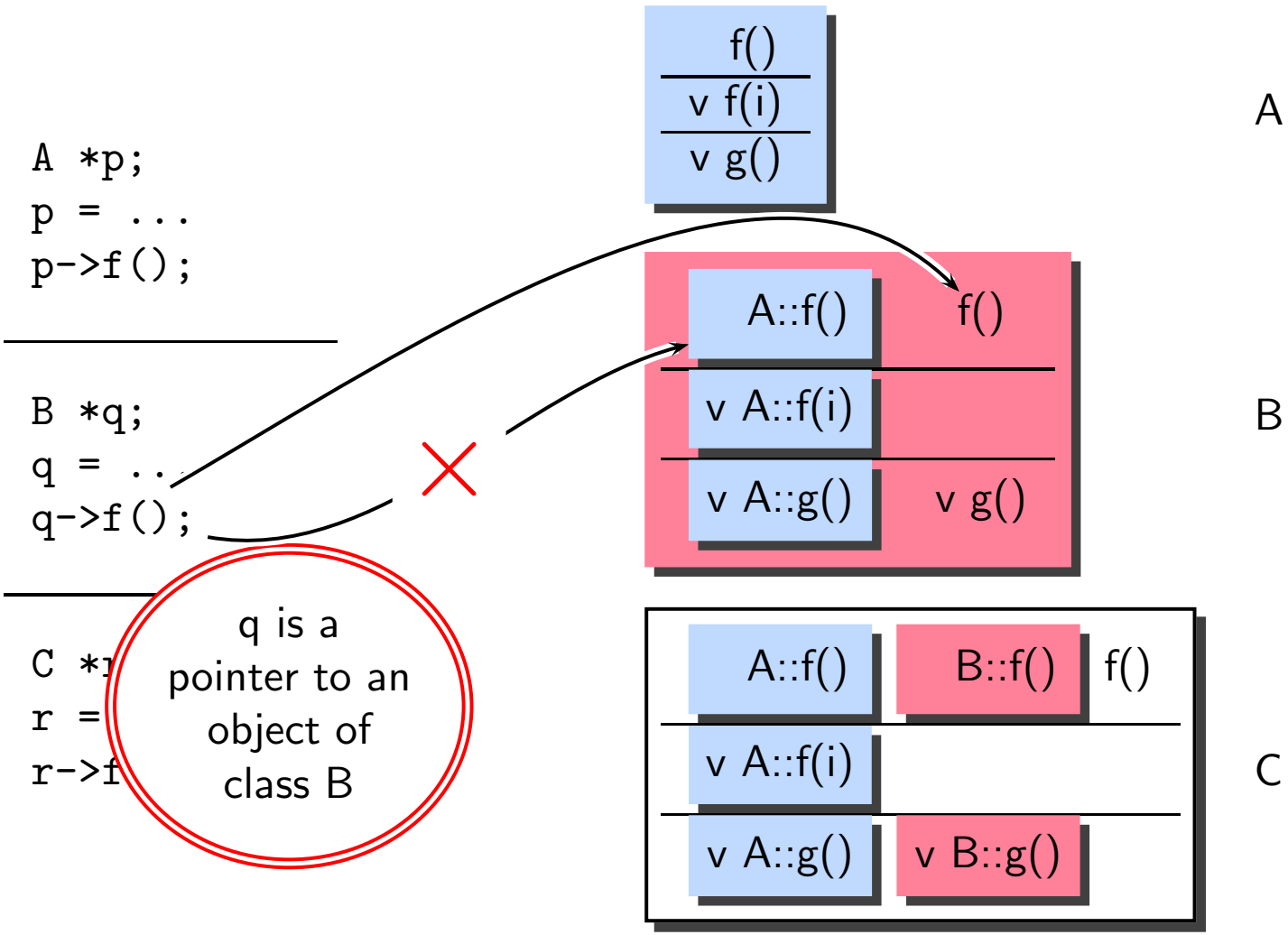
```
B *q;
q = ...
q->f();
```

```
C *r;
r = ...
r->f();
```



Non-Virtual Functions Do Not Require Dynamic Information

Non-virtual function = a function which is not virtual in *any* class in a hierarchy



Examining the Behaviour of Virtual Functions

Program	With virtual functions	No virtual function

`class A`
`{ public:`
`  virtual void f()`
`  virtual void f(string i)`
`  virtual void g()`
`};`
`class B : public A`
`{ public:`
`  virtual void g()`
`  void f()`
`};`
`class C : public B`
`{ public:`
`  void f()`
`};`
	`0 A:f`
	`A:f.x`
	`A:g`
	`1 B:f`
	`A:f.x`
	`B:g`
	`2 C:f`
	`A:f.x`
	`B:g`
		`0 A:f`
		`A:f.x`
		`A:g`
		`1 A:f`
		`A:f.x`
		`A:g`
		`2 A:f`
		`A:f.x`
		`A:g`

Examining the Behaviour of Virtual Functions

Program	With virtual functions	No virtual function

`class A`
`{ public:`
`  virtual void f()`
`  virtual void f(string i)`
`  virtual void g()`
`};`
`class B : public A`
`{ public:`
`  virtual void g()`
`  void f()`
`};`
`class C : public B`
`{ public:`
`  void f()`
`};`
	0 A:f
	A:f.x
	A:g
	1 B:f
	A:f.x
	B:g
	2 C:f
	A:f.x
	B:g
		0 A:f
		A:f.x
		A:g
		1 A:f
		A:f.x
		A:g
		2 A:f
		A:f.x
		A:g

A Summary of Function Call Resolution

- Resolution of virtual functions depends on the class of the pointee object
  $\Rightarrow$ Needs dynamic information
- Resolution of non-virtual functions depends on the class of the pointer
  $\Rightarrow$ Compile time information is sufficient
- In either case, a pointee cannot belong to a "higher" class in the hierarchy
  ("higher" class = a class from which the class of the pointer is derived)

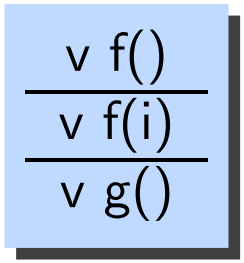
Constructing Virtual Function Table (1)

```
class A
{ public:
    virtual void f()
    virtual void f(string i)
    virtual void g()
};

class B : public A
{ public:
    virtual void g()
             void f()
};

class C : public B
{ public:
             void f()
};
```

Constructing Virtual Function Table (1)

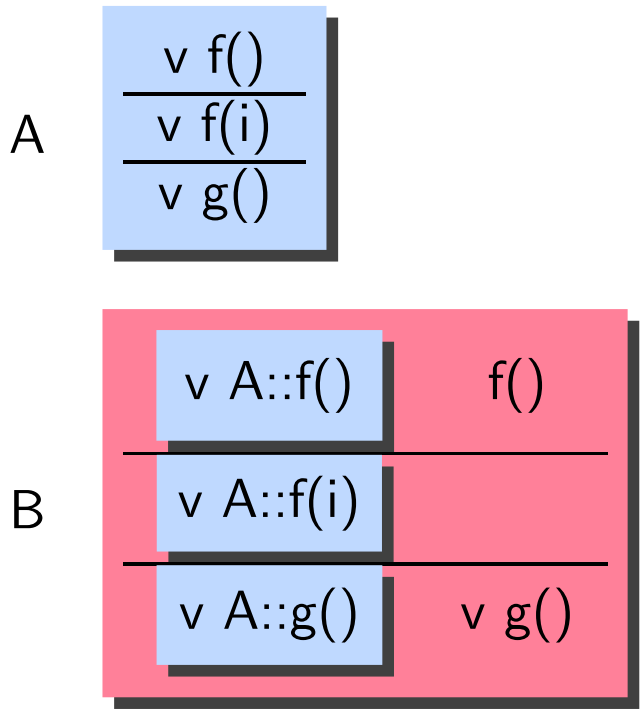
```
class A
{ public:
    virtual void f()
    virtual void f(string i)
    virtual void g()
};

class B : public A
{ public:
    virtual void g()
             void f()
};

class C : public B
{ public:
             void f()
};
```

A



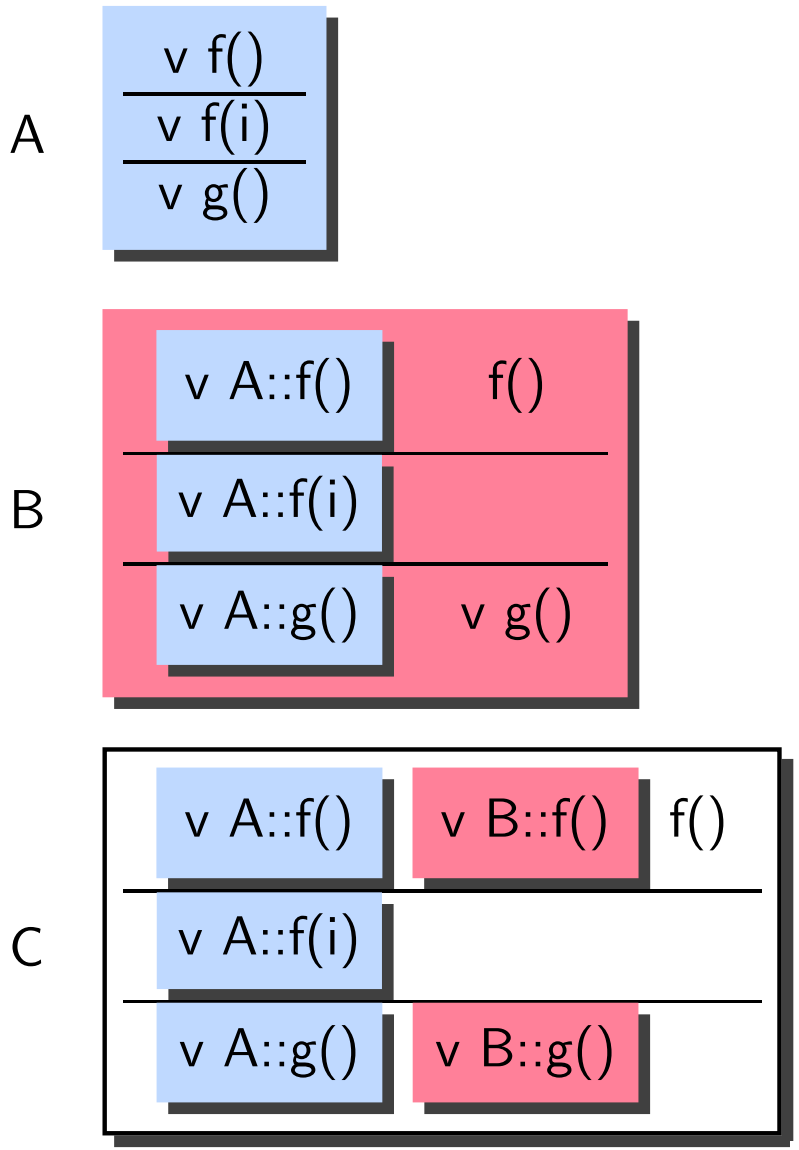
Constructing Virtual Function Table (1)

```
class A
{ public:
    virtual void f()
    virtual void f(string i)
    virtual void g()
};

class B : public A
{ public:
    virtual void g()
             void f()
};

class C : public B
{ public:
             void f()
};
```

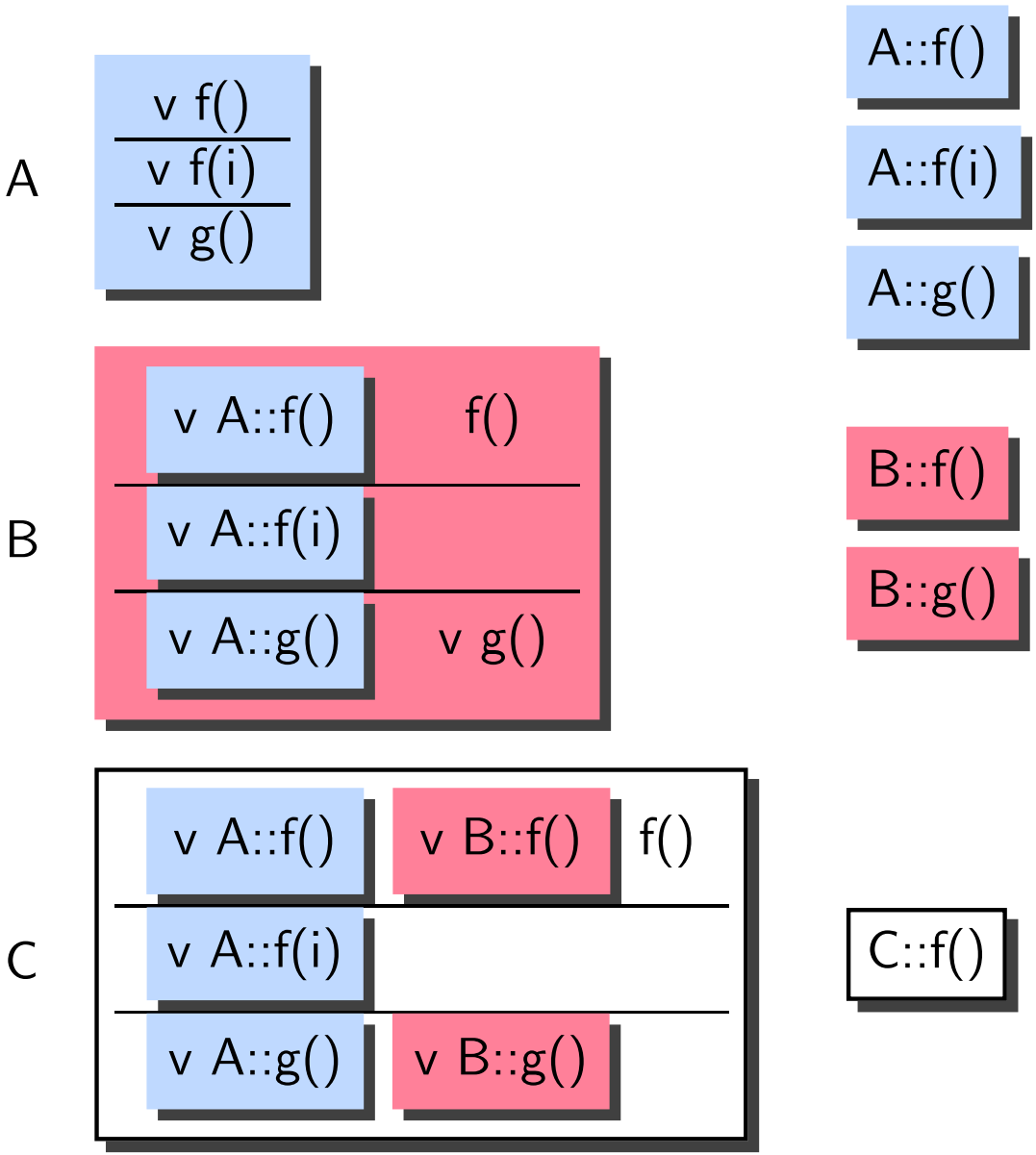


Constructing Virtual Function Table (1)

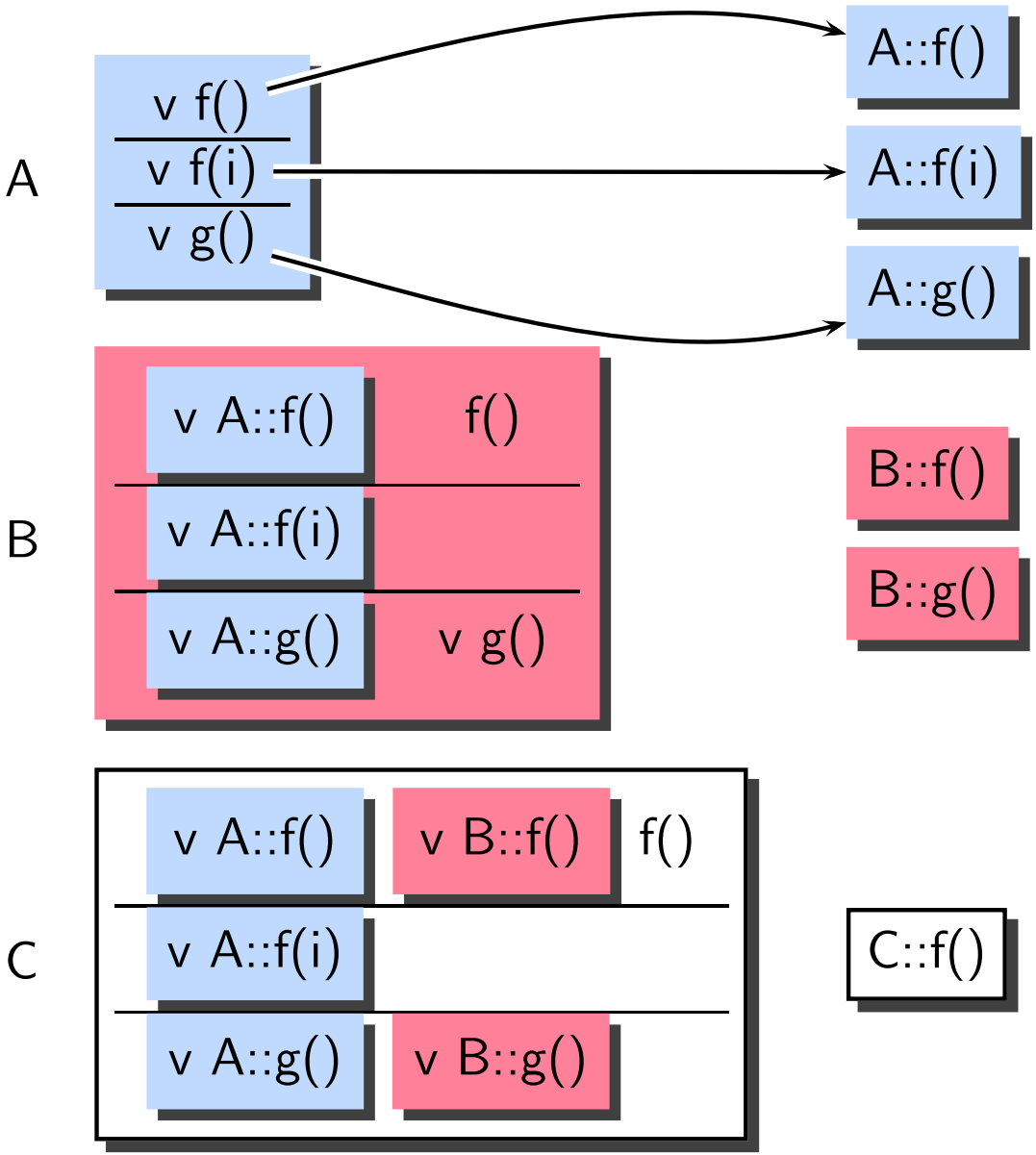
```
class A
{ public:
    virtual void f()
    virtual void f(string i)
    virtual void g()
};

class B : public A
{ public:
    virtual void g()
             void f()
};

class C : public B
{ public:
             void f()
};
```



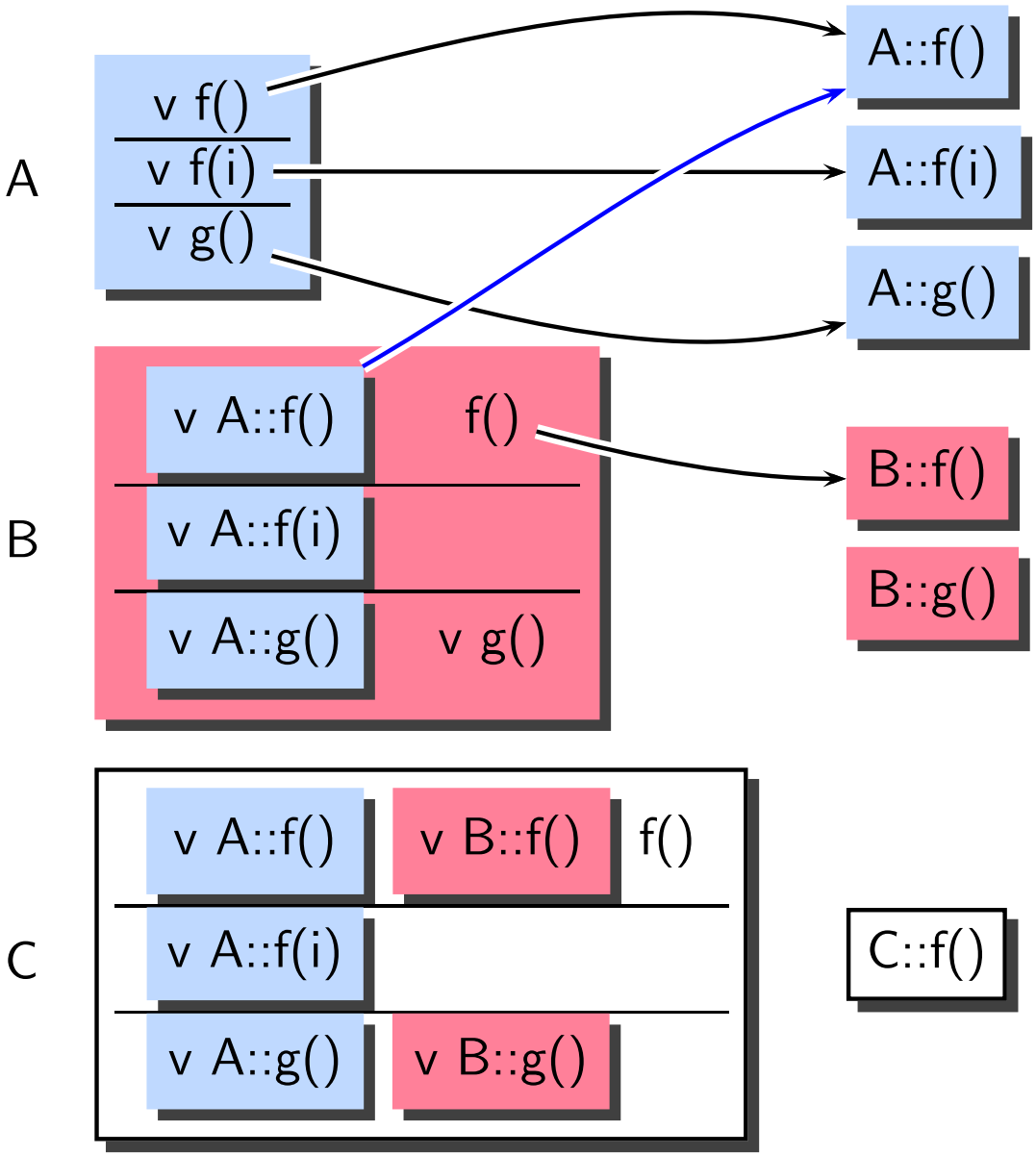
Constructing Virtual Function Table (2)



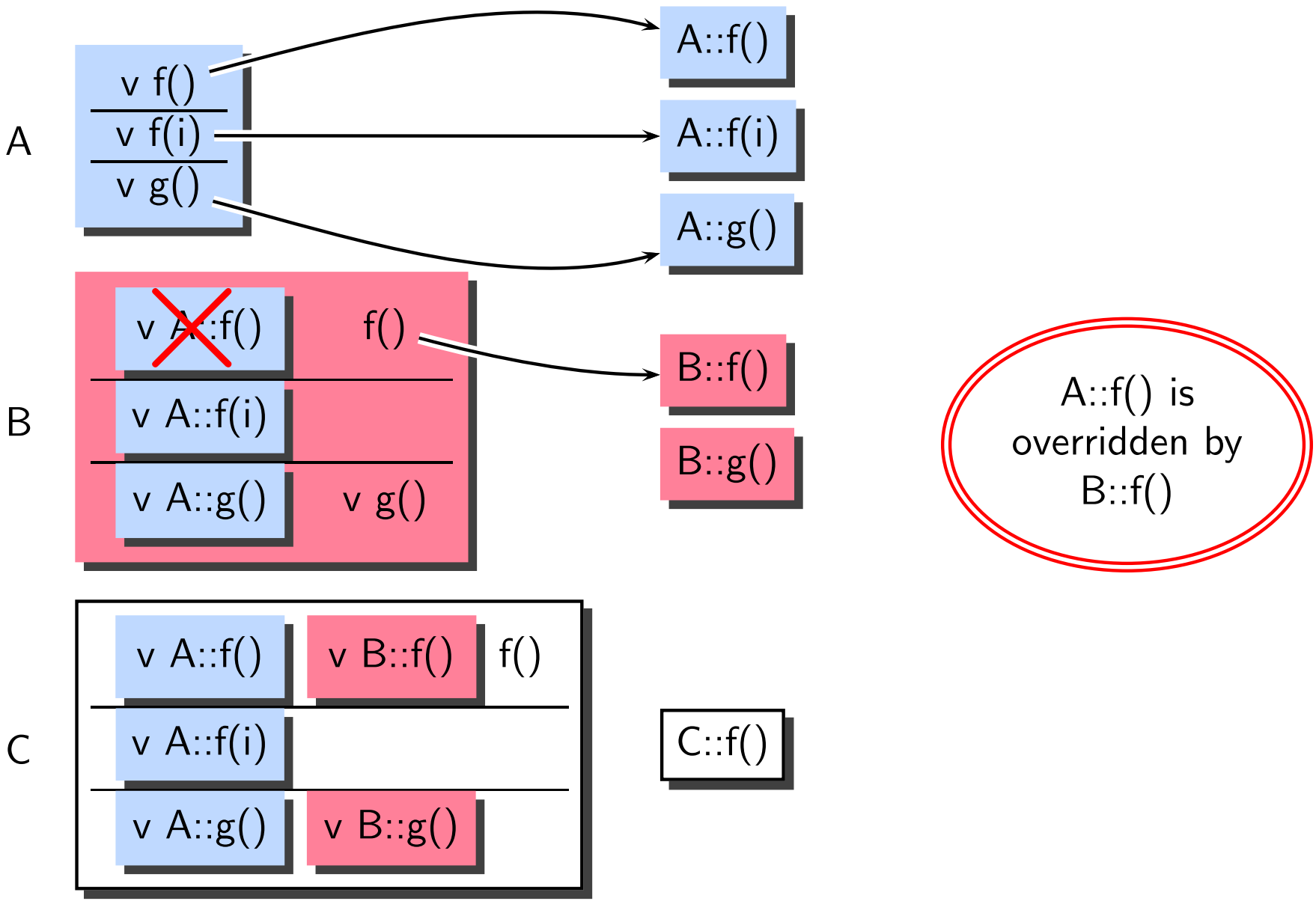
Constructing Virtual Function Table (2)



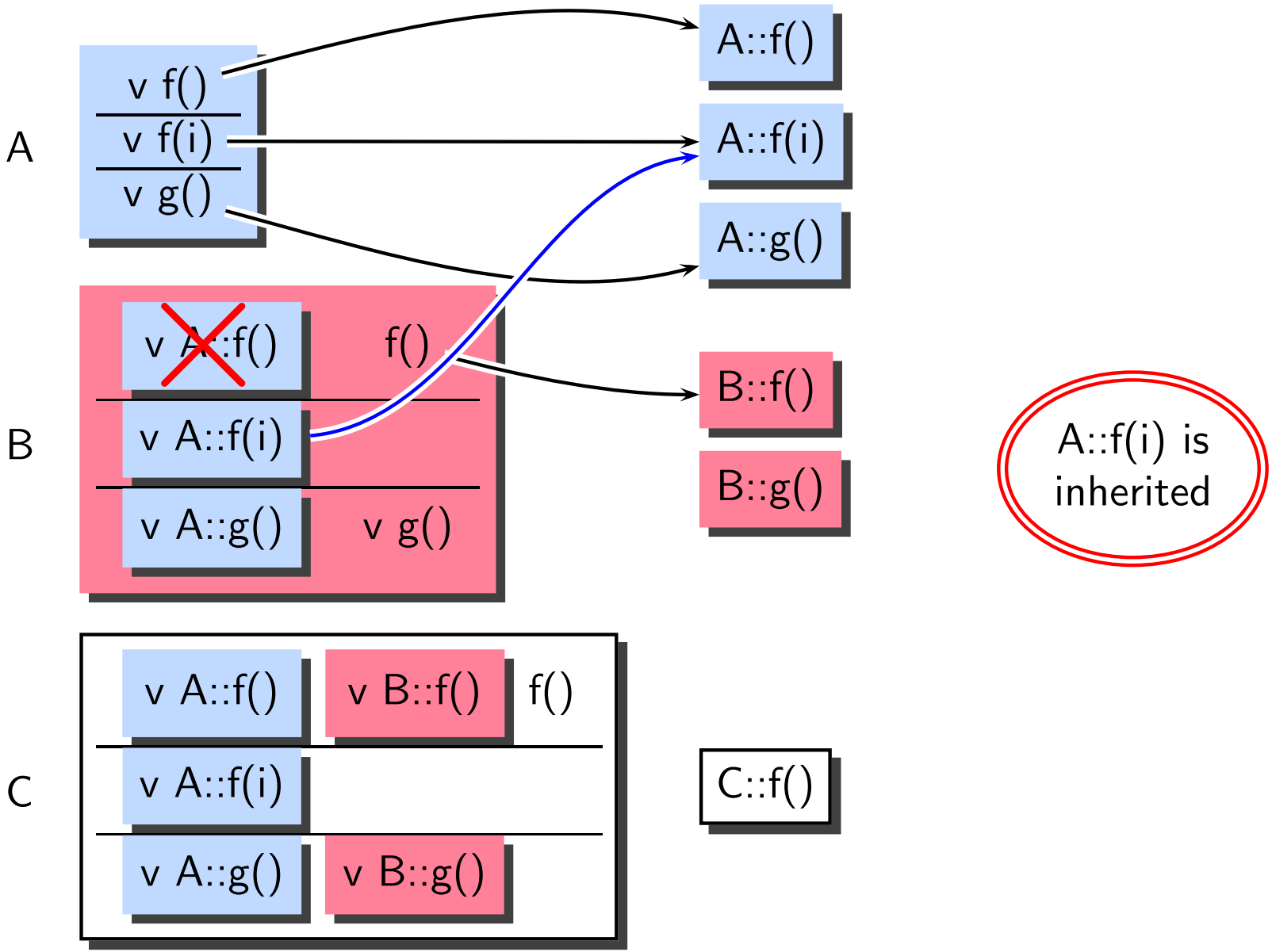
Constructing Virtual Function Table (2)



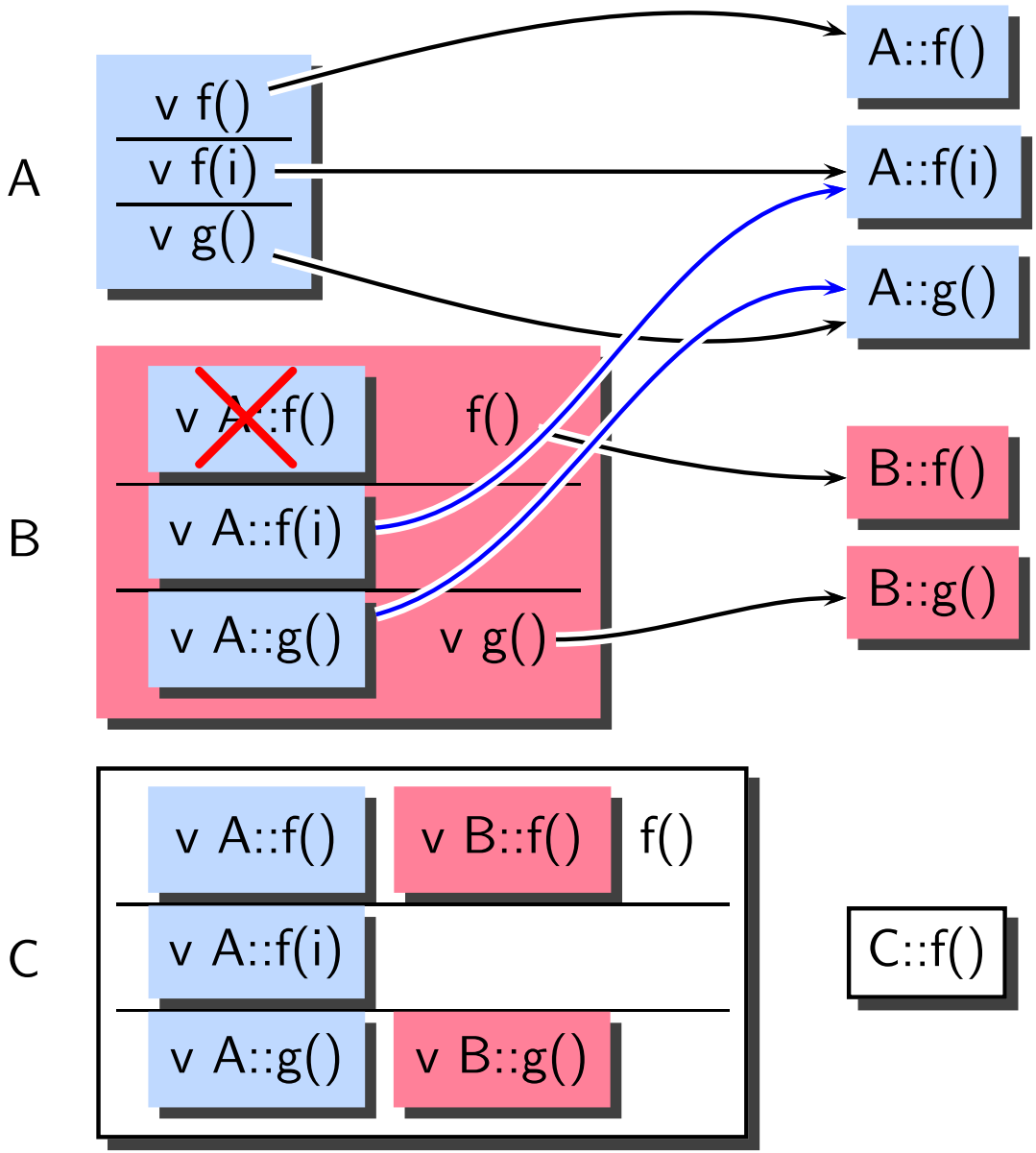
Constructing Virtual Function Table (2)



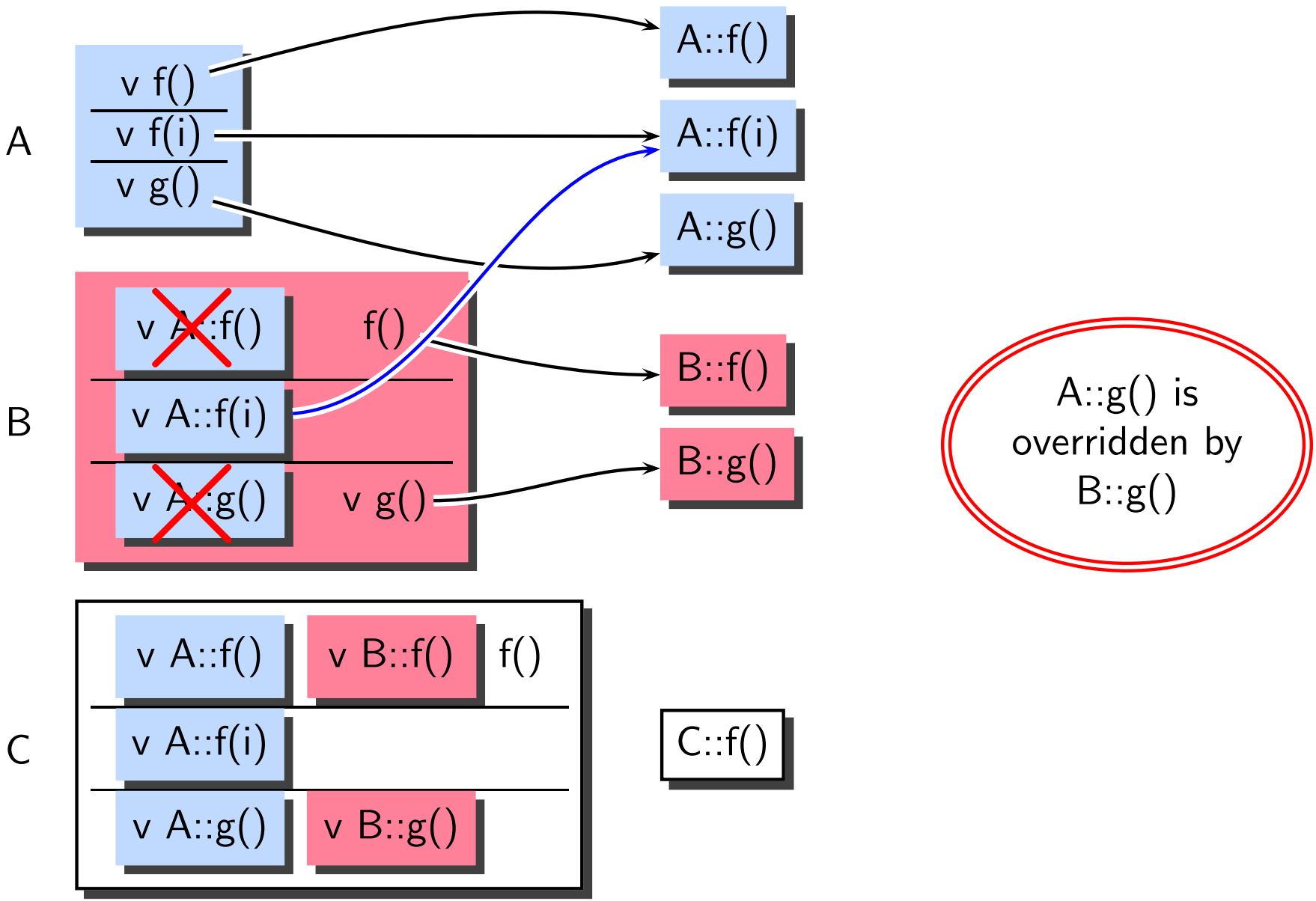
Constructing Virtual Function Table (2)



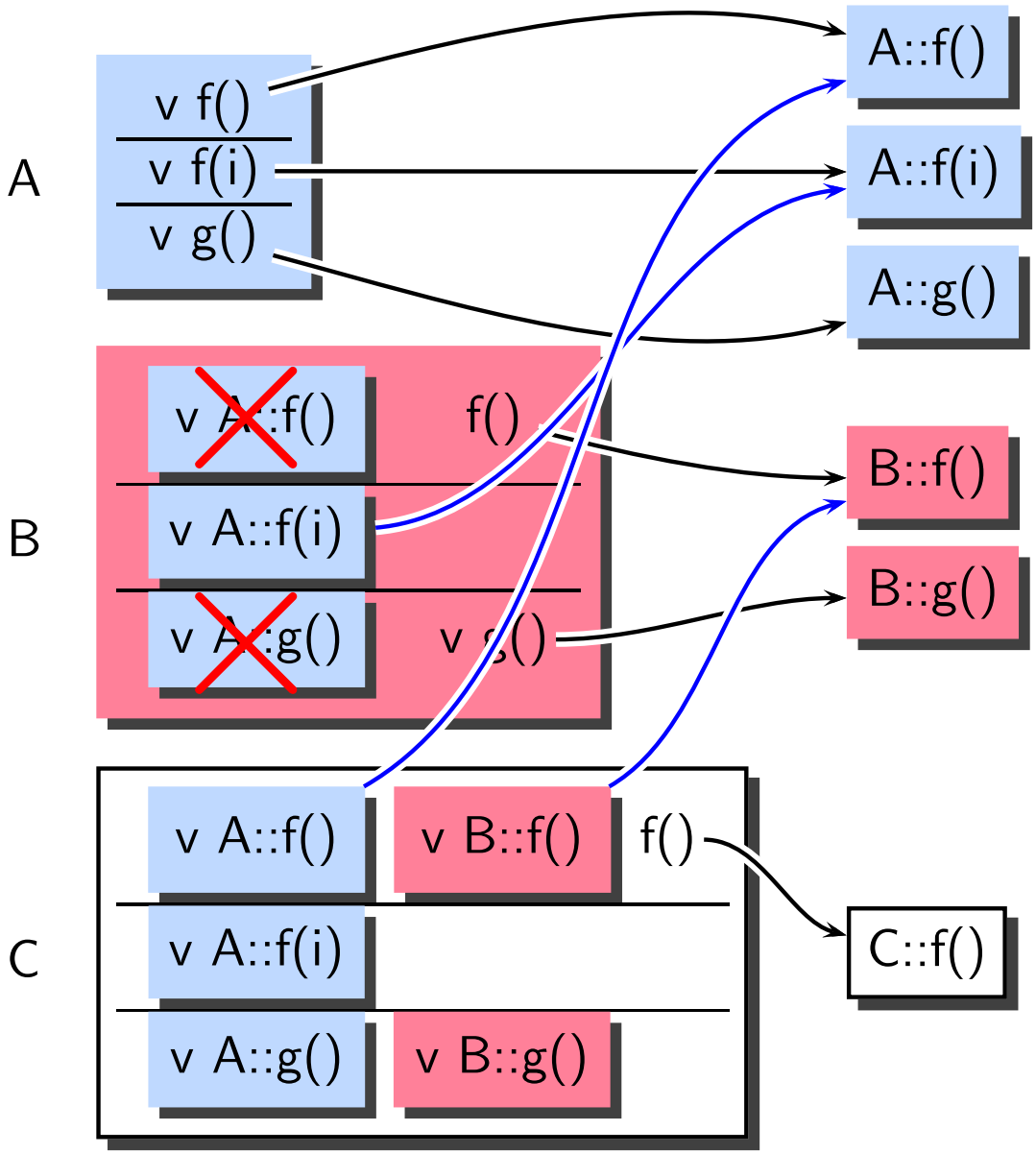
Constructing Virtual Function Table (2)



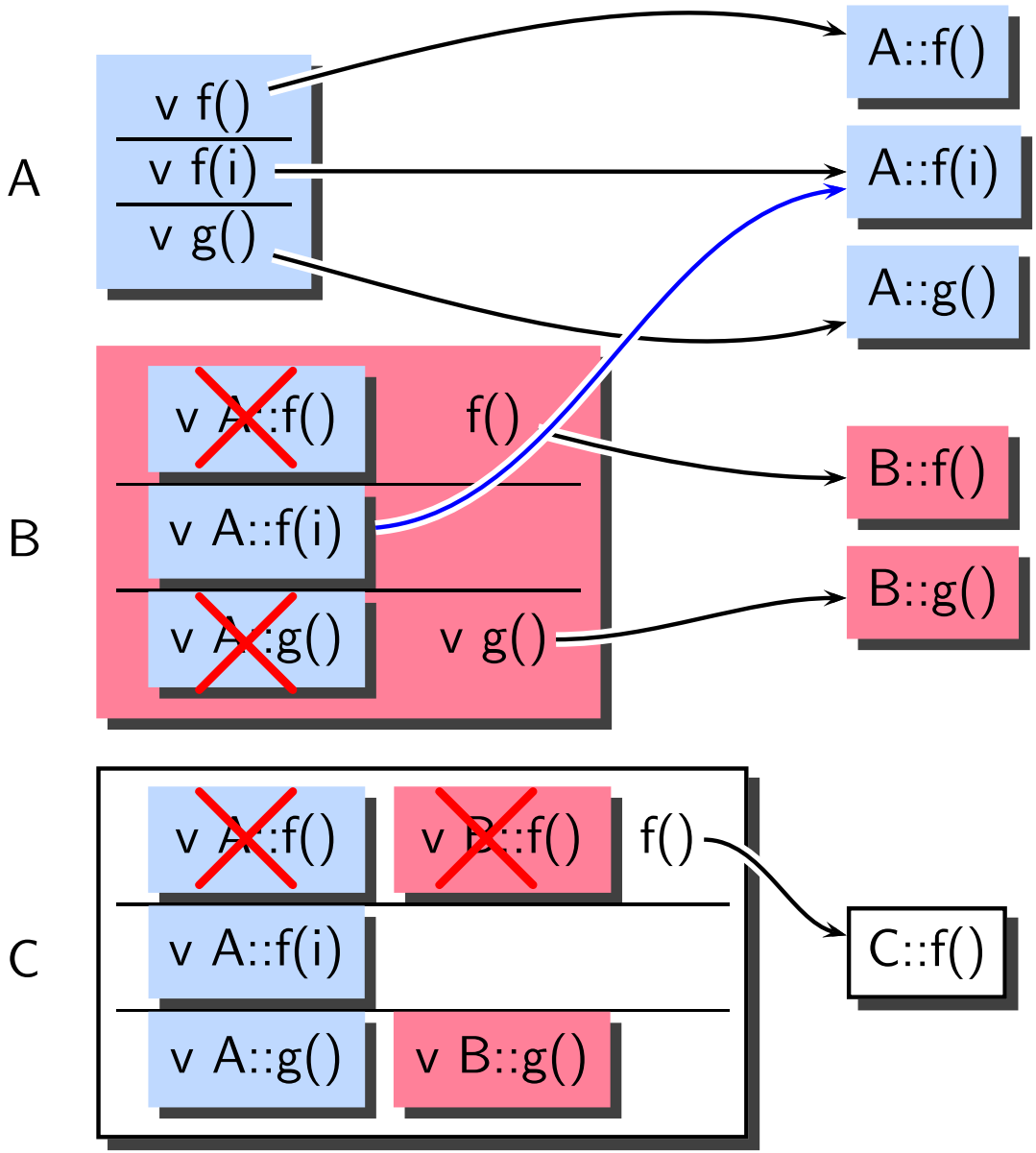
Constructing Virtual Function Table (2)



Constructing Virtual Function Table (2)

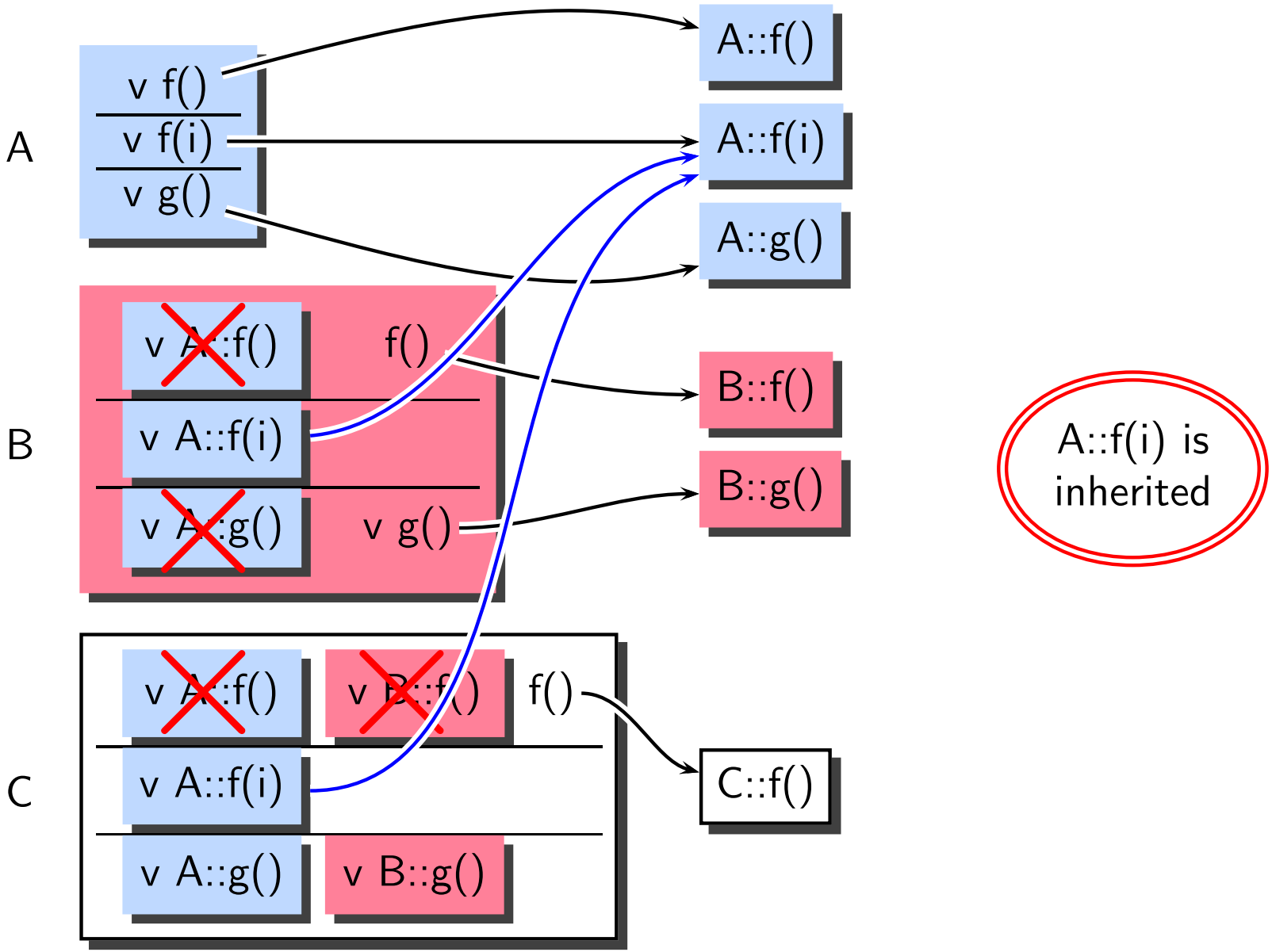


Constructing Virtual Function Table (2)

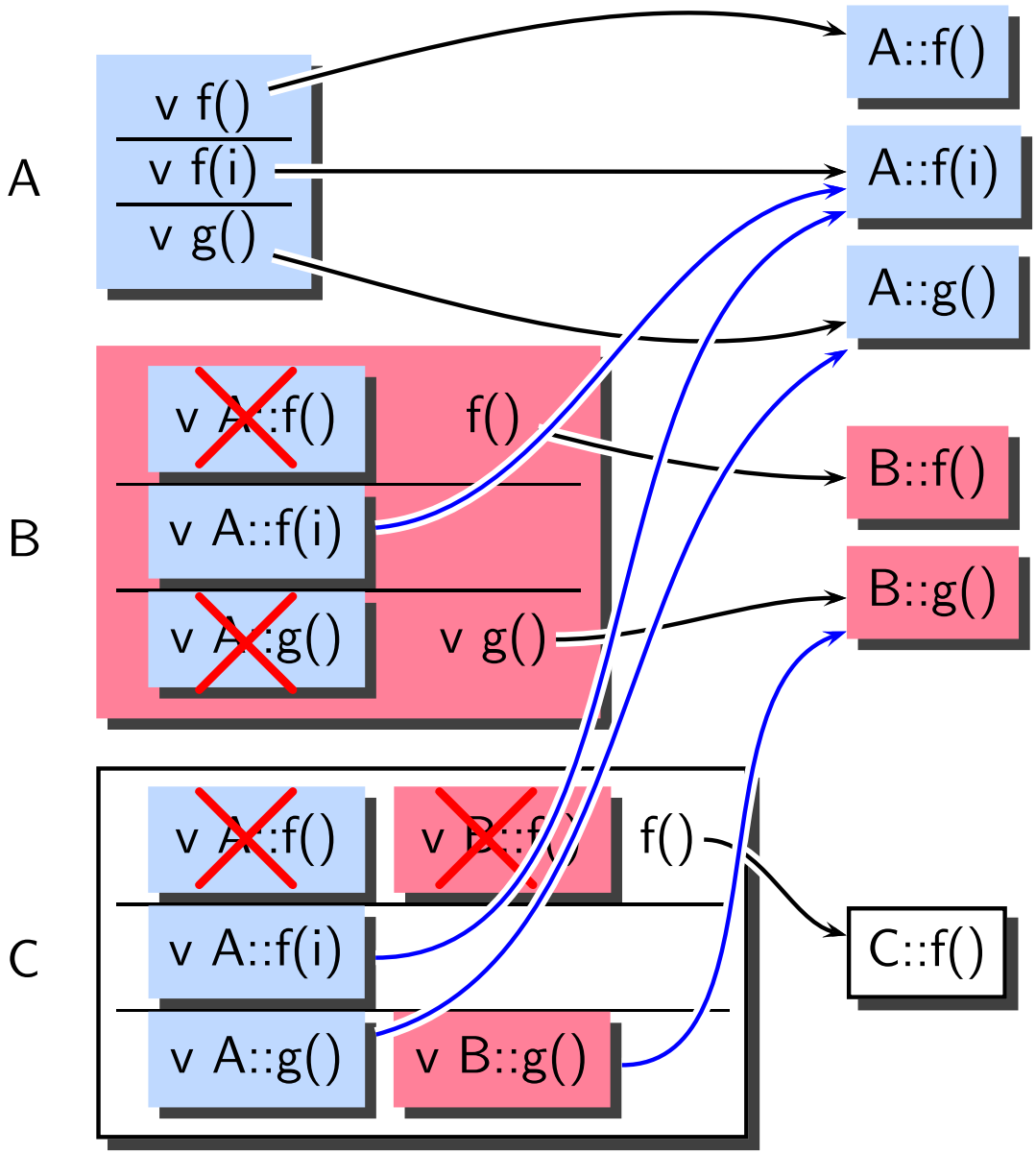


Both A::f() and B::f() are overridden by C::f()

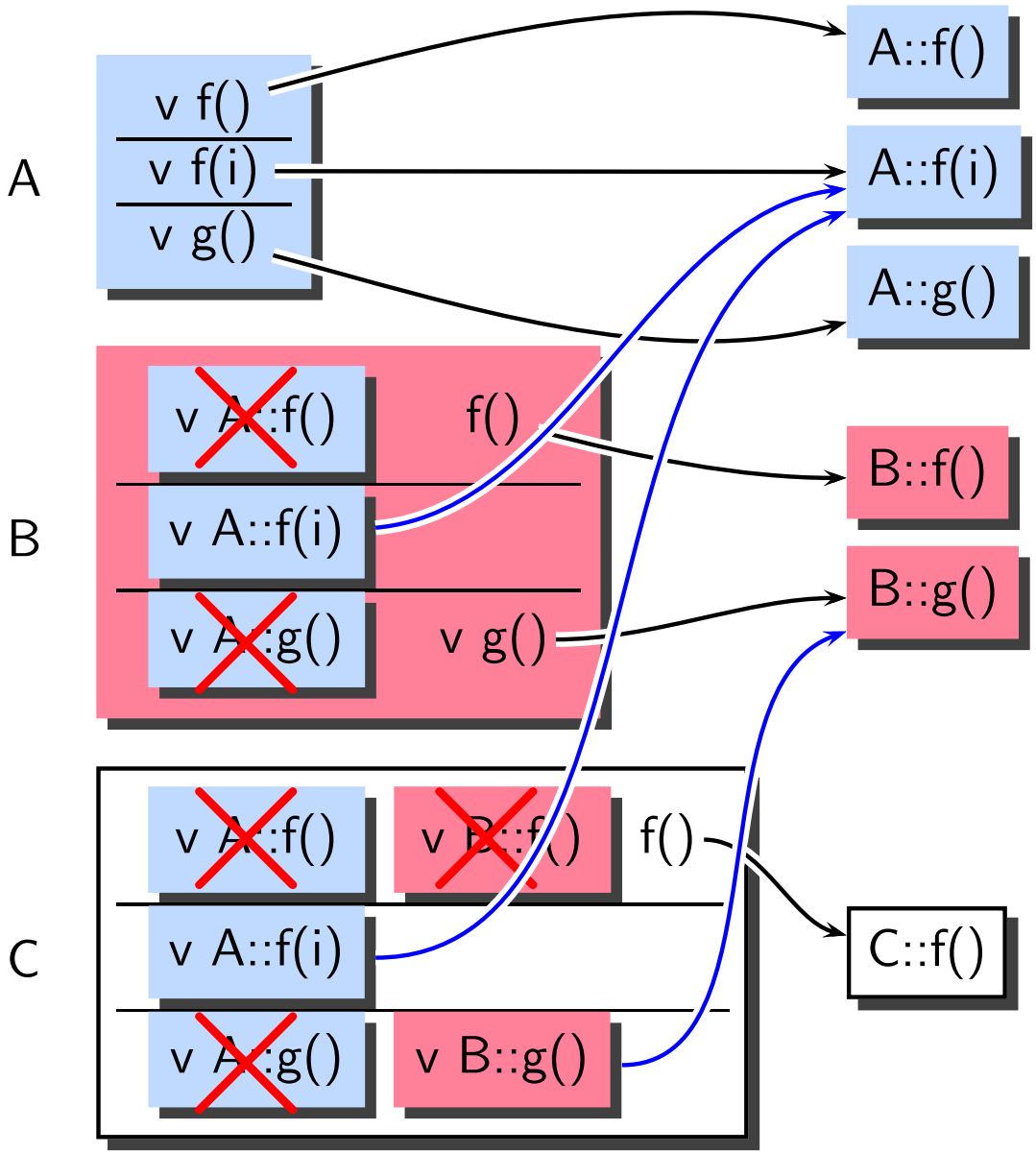
Constructing Virtual Function Table (2)



Constructing Virtual Function Table (2)

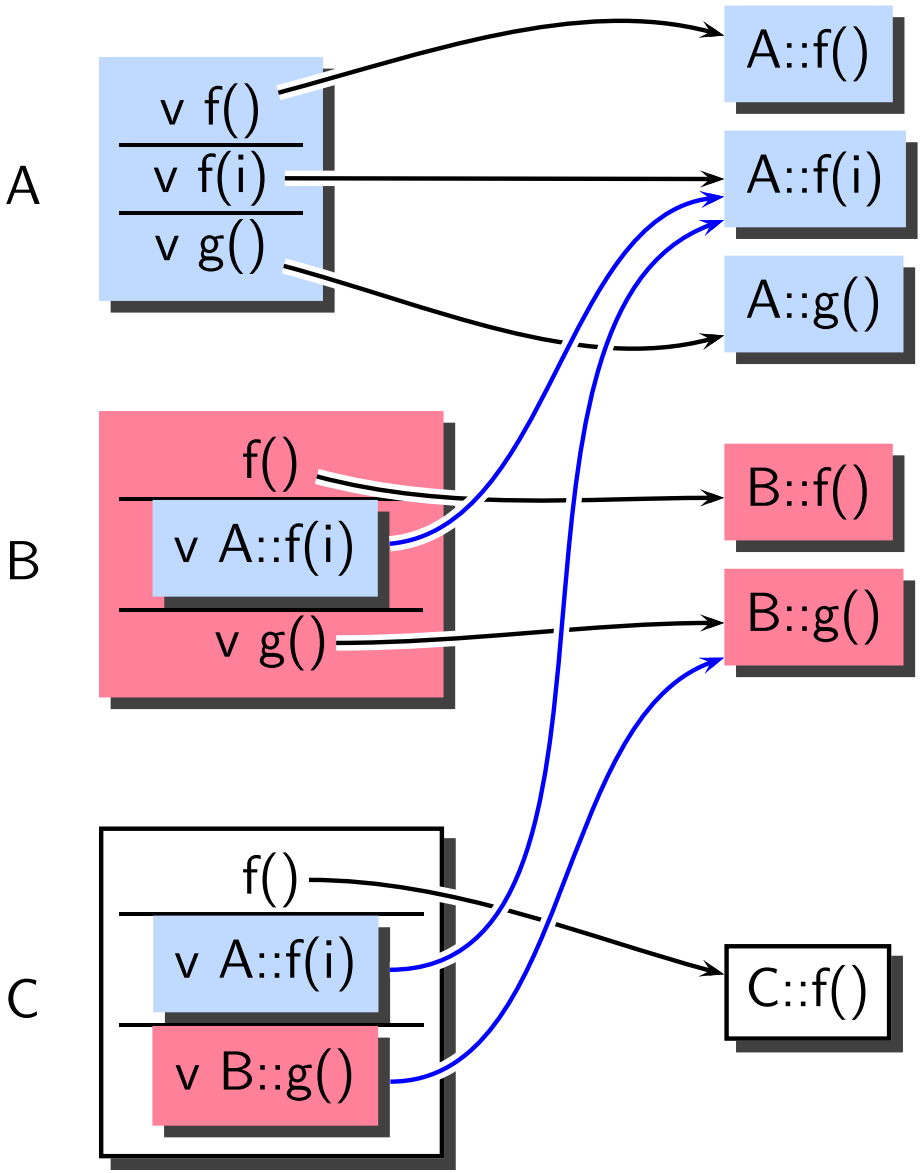


Constructing Virtual Function Table (2)

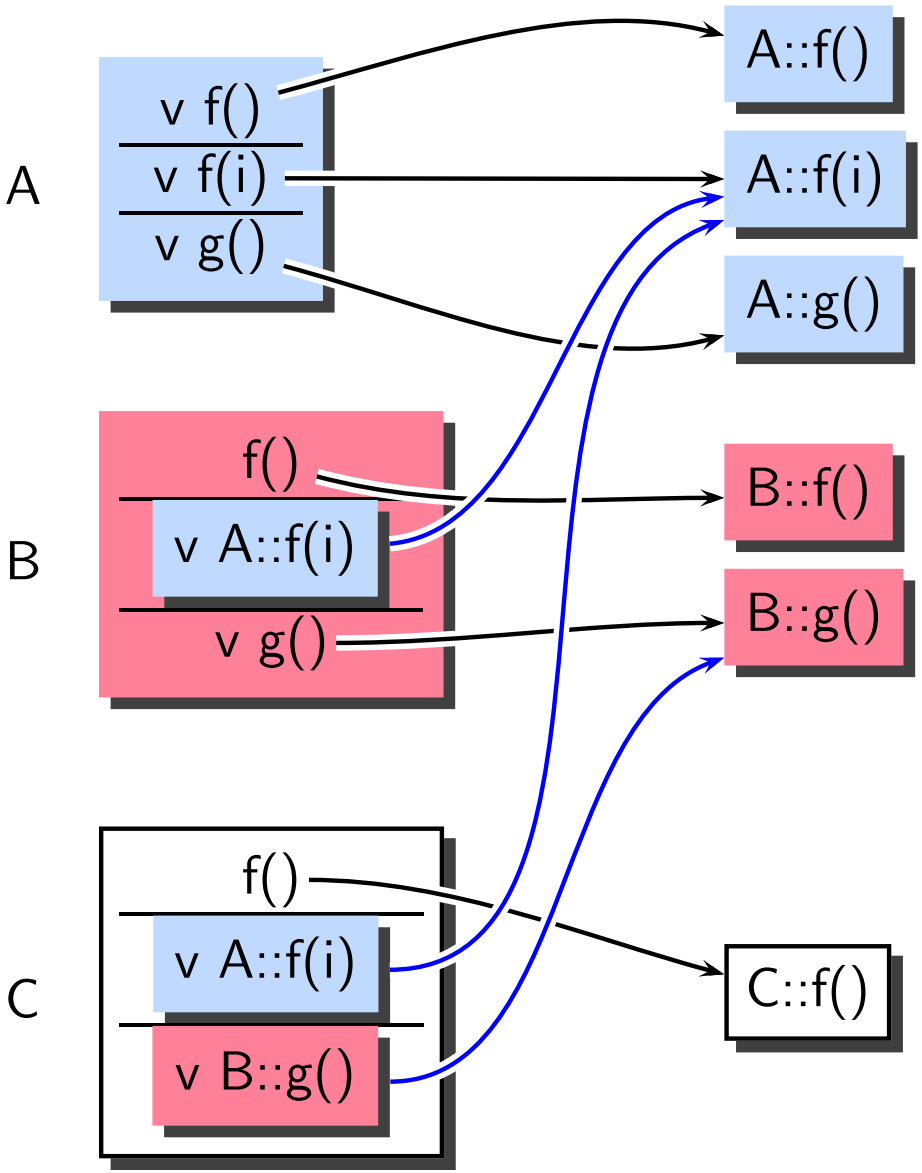


A::g() is overridden by B::g() which is inherited

Constructing Virtual Function Table (2)

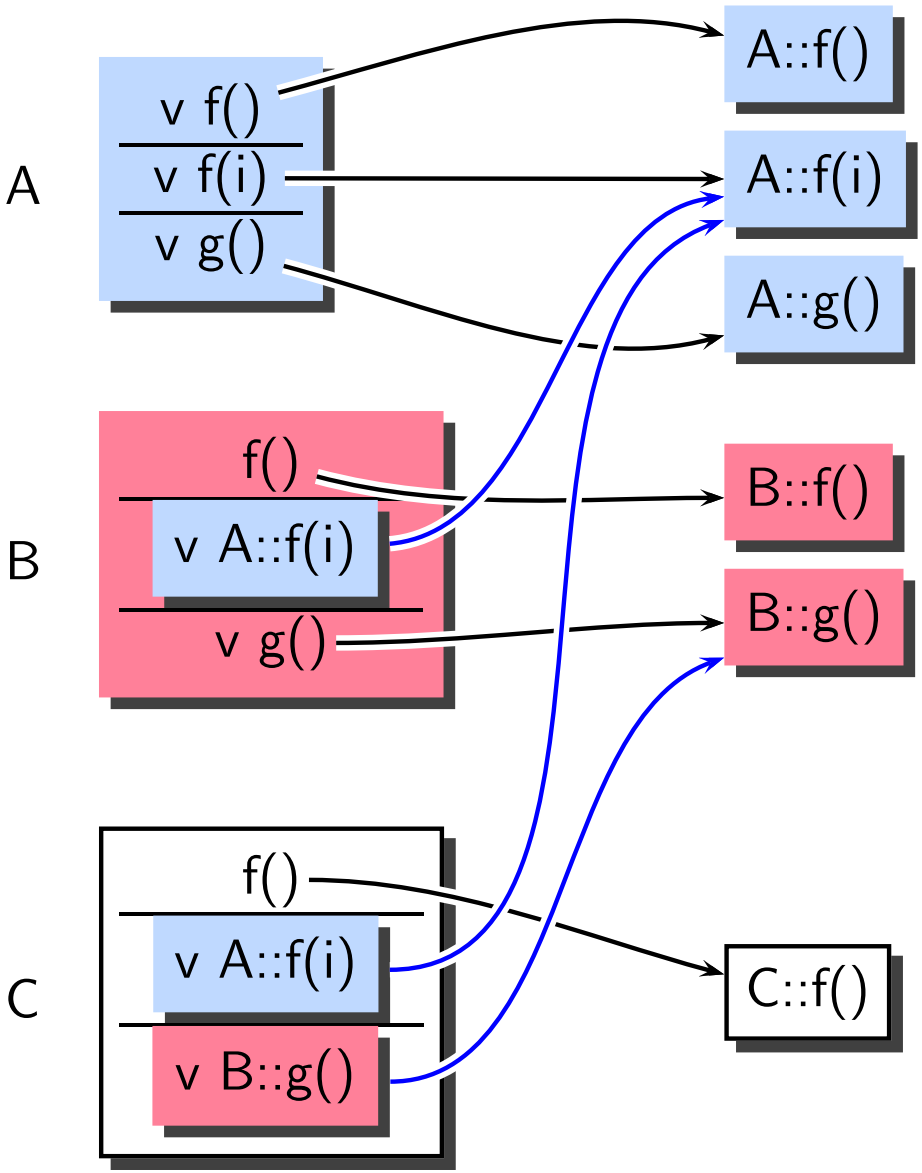


Constructing Virtual Function Table (2)




```
Vtable for A
A::_ZTV1A: 5u entries
0 (int (*)(...))0
8 (int (*)(...))(& _ZTI1A)
16 A::f
24 A::f
32 A::g

Vtable for B
B::_ZTV1B: 5u entries
0 (int (*)(...))0
8 (int (*)(...))(& _ZTI1B)
16 B::f
24 A::f
32 B::g

Vtable for C
C::_ZTV1C: 5u entries
0 (int (*)(...))0
8 (int (*)(...))(& _ZTI1C)
16 C::f
24 A::f
32 B::g
```

Constructing Virtual Function Table (2)




```
Vtable for A
A::_ZTV1A: 5u entries
0 (int (*)(...))0
8 (int (*)(...))(& _ZTI1A)
16 A::f
24 A::f
32 A::g

Vtable for B
B::_ZTV1B: 5u entries
0 (int (*)(...))0
8 (int (*)(...))(& _ZTI1B)
16 B::f
24 A::f
32 B::g

Vtable for C
C::_ZTV1C: 5u entries
0 (int (*)(...))0
8 (int (*)(...))(& _ZTI1C)
16 C::f
24 A::f
32 B::g
```

Virtual Table and Class Information

Dump file: test.cpp.002t.class

Virtual Table	Class Information
Vtable for A A::_ZTV1A: 5u entries 0 (int (*)(...))0 8 (int (*)(...))(& _ZTI1A) 16 A::f 24 A::f 32 A::g	Class A size=8 align=8 base size=8 base align=8 A (0x7fa1d08bfcc0) 0 nearly-empty vptr=((& A::_ZTV1A) + 16u)
Vtable for B B::_ZTV1B: 5u entries 0 (int (*)(...))0 8 (int (*)(...))(& _ZTI1B) 16 B::f 24 A::f 32 B::g	Class B size=8 align=8 base size=8 base align=8 B (0x7fa1d07082d8) 0 nearly-empty vptr=((& B::_ZTV1B) + 16u) A (0x7fa1d08bfd20) 0 nearly-empty primary-for B (0x7fa1d07082d8)

Virtual Table and Class Information

Dump file: test.cpp.002t.class

Virtual Table	Class Information
Vtable for A A::_ZTV1A: 5u entries 0 (int (*)(...))0 8 (int (*)(...))(& _ZTI1A) 16 A::f 24 A::f 32 A::g	Class A size=8 align=8 base size=8 base align=8 A (0x7fa1d08bfcc0) 0 nearly-empty vptr=((& A::_ZTV1A) + 16u)
Vtable for B B::_ZTV1B: 5u entries 0 (int (*)(...))0 8 (int (*)(...))(& _ZTI1B) 16 B::f 24 A::f 32 B::g	Class B size=8 align=8 base size=8 base align=8 B (0x7fa1d07082d8) 0 nearly-empty vptr=((& B::_ZTV1B) + 16u) A (0x7fa1d08bfd20) 0 nearly-empty primary-for B (0x7fa1d07082d8)

Virtual Table and Class Information

Dump file: test.cpp.002t.class

Virtual Table	Class Information
Vtable for A A::_ZTV1A: 5u entries 0 (int (*)(...))0 8 (int (*)(...))(& _ZTI1A) 16 A::f 24 A::f 32 A::g	Class A size=8 align=8 base size=8 base align=8 A (0x7fa1d08bfcc0) 0 nearly-empty vptr=((& A::_ZTV1A) + 16u)
Vtable for B B::_ZTV1B: 5u entries 0 (int (*)(...))0 8 (int (*)(...))(& _ZTI1B) 16 B::f 24 A::f 32 B::g	Class B size=8 align=8 base size=8 base align=8 B (0x7fa1d07082d8) 0 nearly-empty vptr=((& B::_ZTV1B) + 16u) A (0x7fa1d08bfd20) 0 nearly-empty primary-for B (0x7fa1d07082d8)

Class Information With and Without Virtual Functions

Dump file: test.cpp.002t.class

With virtual qaualifier	Without virtual qaualifier
Class A size=8 align=8 base size=8 base align=8 A (0x7fa1d08bfcc0) 0 nearly-empty vptr=((& A::_ZTV1A) + 16u)	Class A size=1 align=1 base size=0 base align=1 A (0x7fa1d08bfde0) 0 empty
Class B size=8 align=8 base size=8 base align=8 B (0x7fa1d07082d8) 0 nearly-empty vptr=((& B::_ZTV1B) + 16u) A (0x7fa1d08bfd20) 0 nearly-empty primary-for B (0x7fa1d07082d8)	Class B size=1 align=1 base size=1 base align=1 B (0x7fa1d07084e0) 0 empty A (0x7fa1d08bfe40) 0 empty

Class Information With and Without Virtual Functions

Dump file: test.cpp.002t.class

With virtual qaualifier	Without virtual qaualifier
Class A size=8 align=8 base size=8 base align=8 A (0x7fa1d08bfcc0) 0 nearly-empty vptr=((& A::_ZTV1A) + 16u)	Class A size=1 align=1 base size=0 base align=1 A (0x7fa1d08bfde0) 0 empty
Class B size=8 align=8 base size=8 base align=8 B (0x7fa1d07082d8) 0 nearly-empty vptr=((& B::_ZTV1B) + 16u) A (0x7fa1d08bfd20) 0 nearly-empty primary-for B (0x7fa1d07082d8)	Class B size=1 align=1 base size=1 base align=1 B (0x7fa1d07084e0) 0 empty A (0x7fa1d08bfe40) 0 empty

Class Information With and Without Virtual Functions

Dump file: test.cpp.002t.class

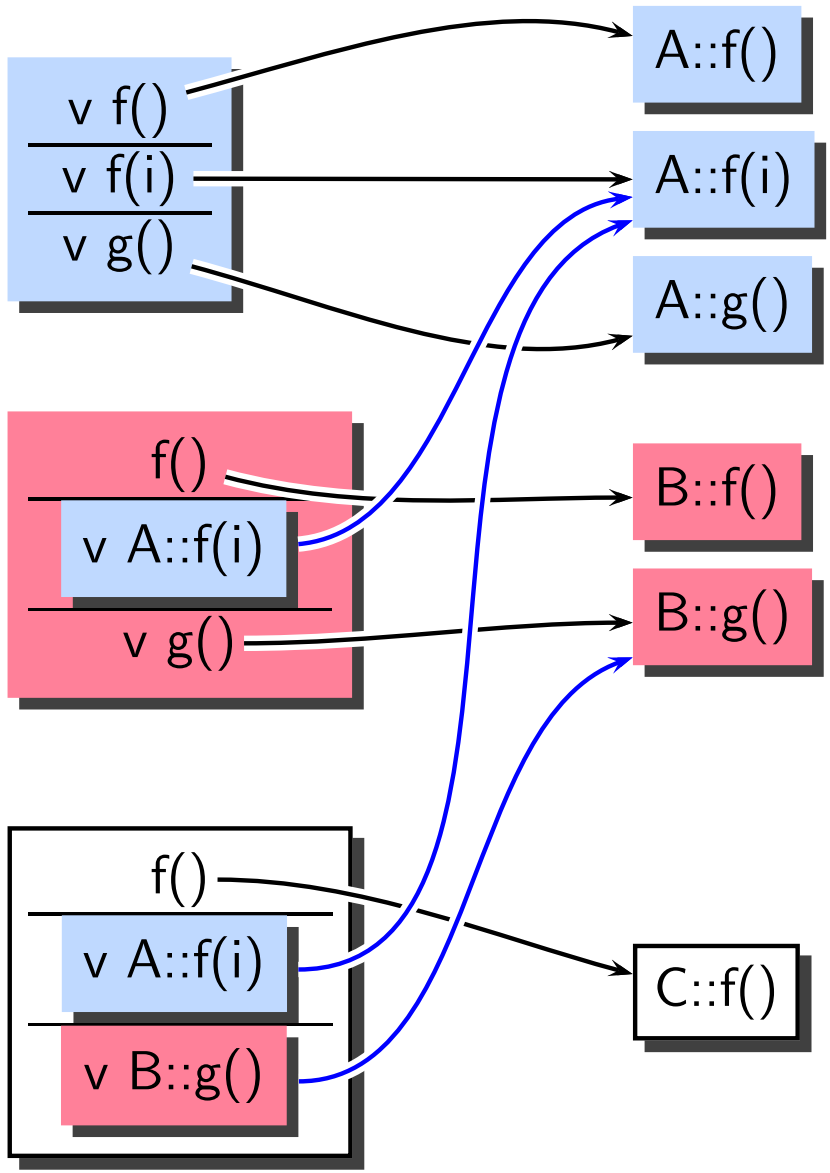
With virtual qaualifier	Without virtual qaualifier
Class A size=8 align=8 base size=8 base align=8 A (0x7fa1d08bfcc0) 0 nearly-empty vptr=((& A::_ZTV1A) + 16u)	Class A size=1 align=1 base size=0 base align=1 A (0x7fa1d08bfde0) 0 empty
Class B size=8 align=8 base size=8 base align=8 B (0x7fa1d07082d8) 0 nearly-empty vptr=((& B::_ZTV1B) + 16u) A (0x7fa1d08bfd20) 0 nearly-empty primary-for B (0x7fa1d07082d8)	Class B size=1 align=1 base size=1 base align=1 B (0x7fa1d07084e0) 0 empty A (0x7fa1d08bfe40) 0 empty

Using the Constructed Virtual Function Table

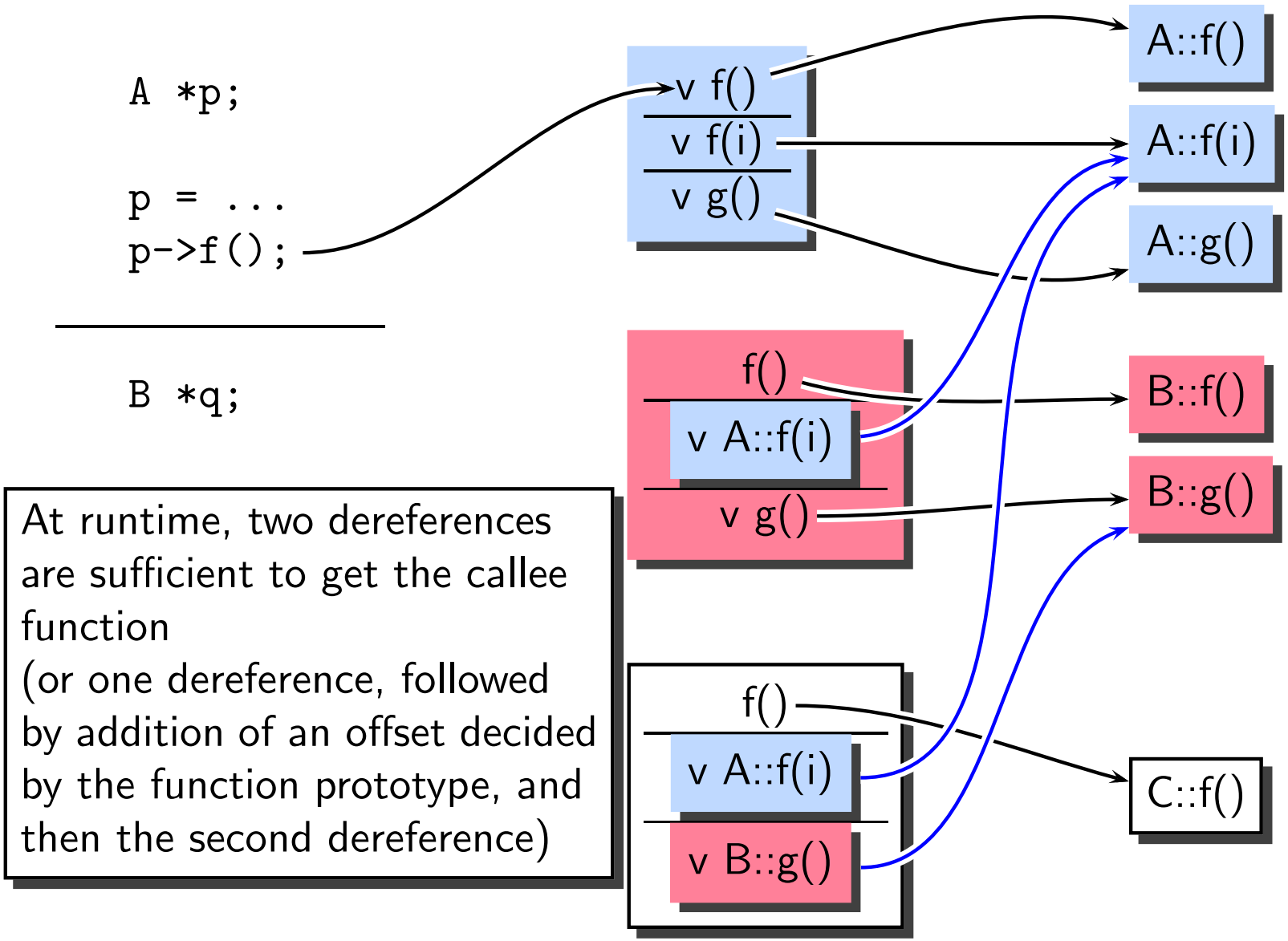
```
A *p;

p = ...
p->f();
```

```
B *q;

q = ...
q->f();
```

```
C *r;

r = ...
r->f();
```

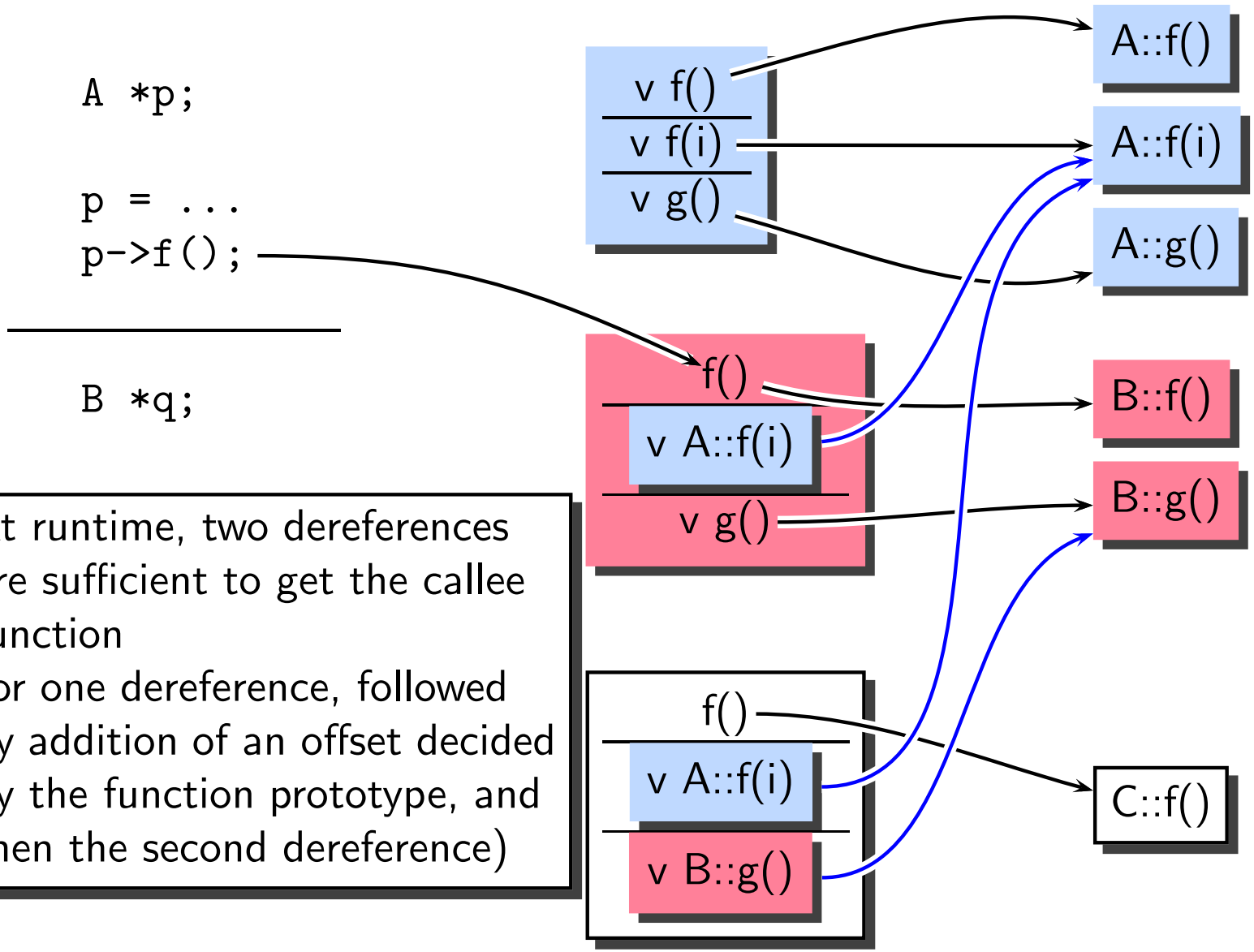


Using the Constructed Virtual Function Table



At runtime, two dereferences are sufficient to get the callee function
(or one dereference, followed by addition of an offset decided by the function prototype, and then the second dereference)

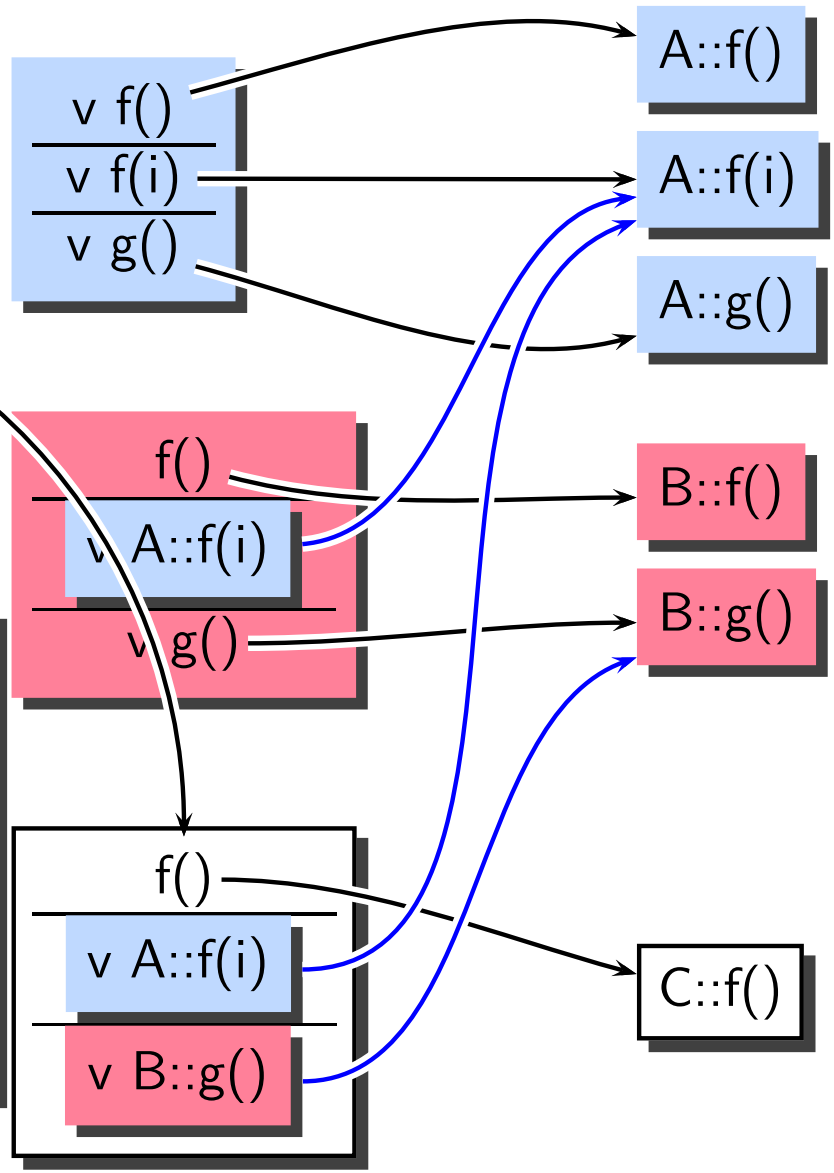
Using the Constructed Virtual Function Table



At runtime, two dereferences are sufficient to get the callee function
(or one dereference, followed by addition of an offset decided by the function prototype, and then the second dereference)

Using the Constructed Virtual Function Table

```
A *p;

p = ...
p->f();

---

B *q;
```



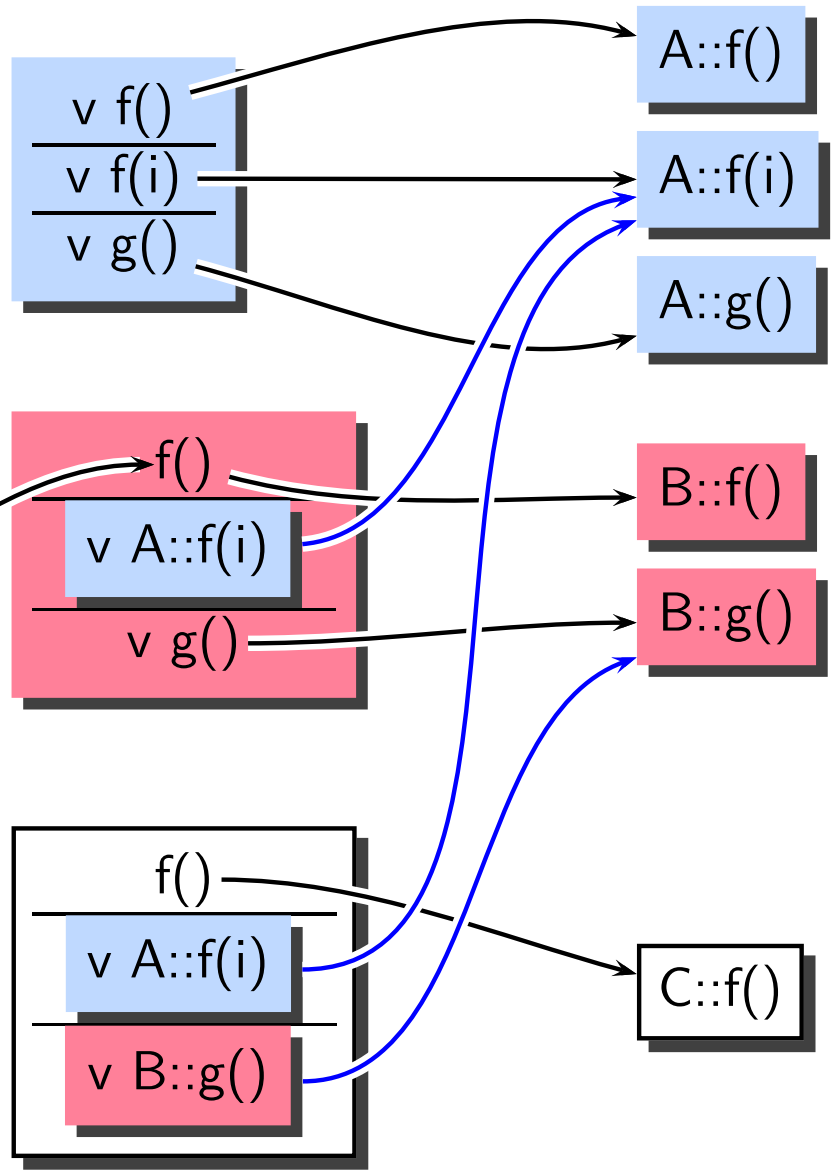
At runtime, two dereferences are sufficient to get the callee function
(or one dereference, followed by addition of an offset decided by the function prototype, and then the second dereference)

Using the Constructed Virtual Function Table

```
A *p;

p = ...
p->f();
```

```
B *q;

q = ...
q->f();
```

```
C *r;

r = ...
r->f();
```

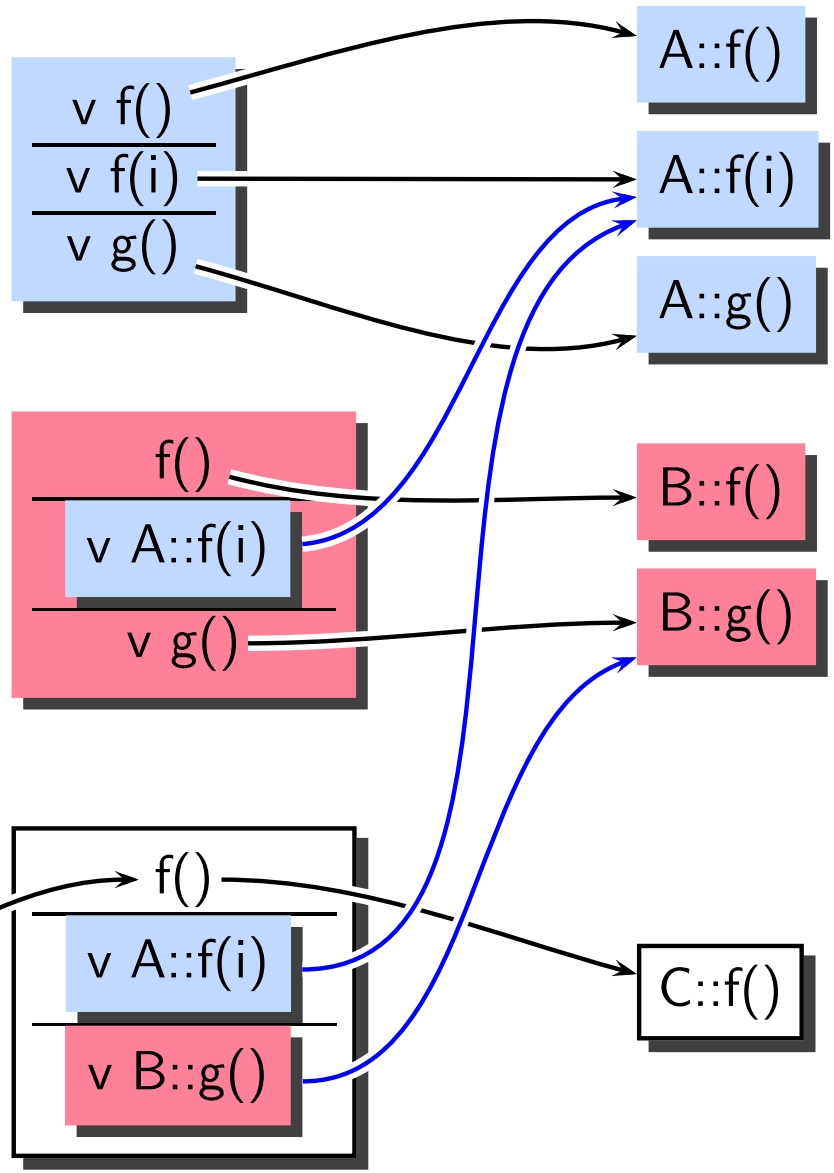


Using the Constructed Virtual Function Table

```
A *p;

p = ...
p->f();
```

```
B *q;

q = ...
q->f();
```

```
C *r;

r = ...
r->f();
```

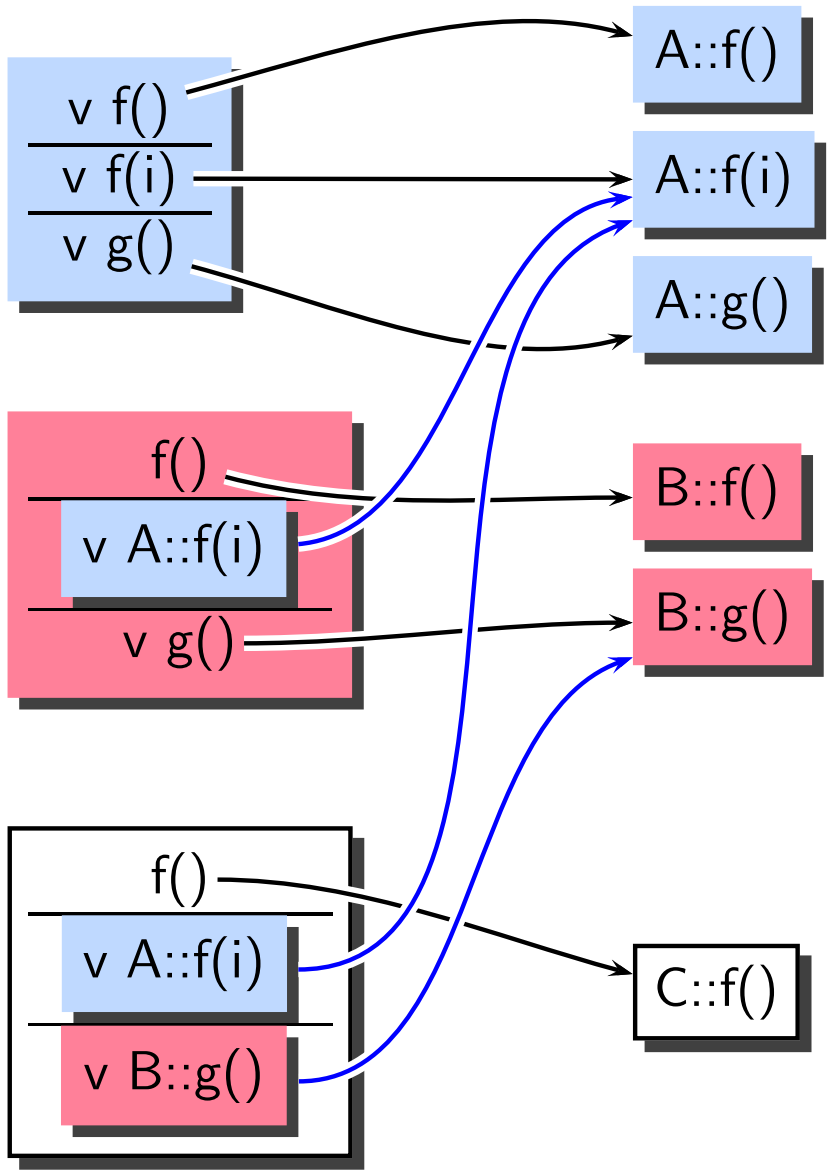


Using the Constructed Virtual Function Table

```
A *p;

p = ...
p->f();
```

This is possible because the size of the virtual function table is made same for all classes in the hierarchy (there can be only one function with a given name and permutation of types of the arguments)

```
r = ...
r->f();
```



A Summary of Virtual Function Implementation

- Compile time activity
  - Collect all virtual functions across a class hierarchy
  - Ignore non-virtual functions
  - Analyse the class hierarchy to locate the appropriate function with a given permutation of argument types

A Summary of Virtual Function Implementation

- Compile time activity
  - Collect all virtual functions across a class hierarchy
  - Ignore non-virtual functions
  - Analyse the class hierarchy to locate the appropriate function with a given permutation of argument types
- Execution time activity
  - Dereference object pointer to access the virtual function table
  - Add offset to the base of the table to access the function pointer
  - Dereference the function pointer to make the call

A Summary of Virtual Function Implementation

- Compile time activity
  - Collect all virtual functions across a class hierarchy
  - Ignore non-virtual functions
  - Analyse the class hierarchy to locate the appropriate function with a given permutation of argument types
- Execution time activity
  - Dereference object pointer to access the virtual function table
  - Add offset to the base of the table to access the function pointer
  - Dereference the function pointer to make the call
- Study time activity
  - Study the virtual table in the `.class` dump
  - Observe the code for handling virtual calls in `.cfg` and assembly dumps
  - Use the options `-fdump-tree-all -S -fverbose-asm`

Part 7

Examining GIMPLE Optimization

Example Program for Observing Optimizations

```
int main()
{ int a, b, c, n;

  a = 1;
  b = 2;
  c = 3;
  n = c*2;
  while (a <= n)
  {
    a = a+1;
  }
  if (a < 12)
    a = a+b+c;
  return a;
}
```

- What does this program return?

Example Program for Observing Optimizations

```
int main()
{ int a, b, c, n;

  a = 1;
  b = 2;
  c = 3;
  n = c*2;
  while (a <= n)
  {
    a = a+1;
  }
  if (a < 12)
    a = a+b+c;
  return a;
}
```

- What does this program return?
- 12

Example Program for Observing Optimizations

```
int main()
{ int a, b, c, n;

  a = 1;
  b = 2;
  c = 3;
  n = c*2;
  while (a <= n)
  {
    a = a+1;
  }
  if (a < 12)
    a = a+b+c;
  return a;
}
```

- What does this program return?
- 12
- We use this program to illustrate various shades of the following optimizations:

  Constant propagation, Copy propagation, Loop unrolling, Dead code elimination

Compilation Command

```
$gcc -fdump-tree-all  -O2 ccp.c
```

Example Program 1

Program ccp.c

```
int main()
{ int a, b, c, n;

  a = 1;
  b = 2;
  c = 3;
  n = c*2;
  while (a <= n)
  {
    a = a+1;
  }
  if (a < 12)
    a = a+b+c;
  return a;
}
```

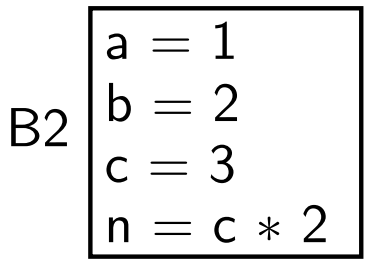
Control flow graph

Example Program 1

Program ccp.c

```
int main()
{ int a, b, c, n;

  a = 1;
  b = 2;
  c = 3;
  n = c*2;
  while (a <= n)
  {
    a = a+1;
  }
  if (a < 12)
    a = a+b+c;
  return a;
}
```

Control flow graph

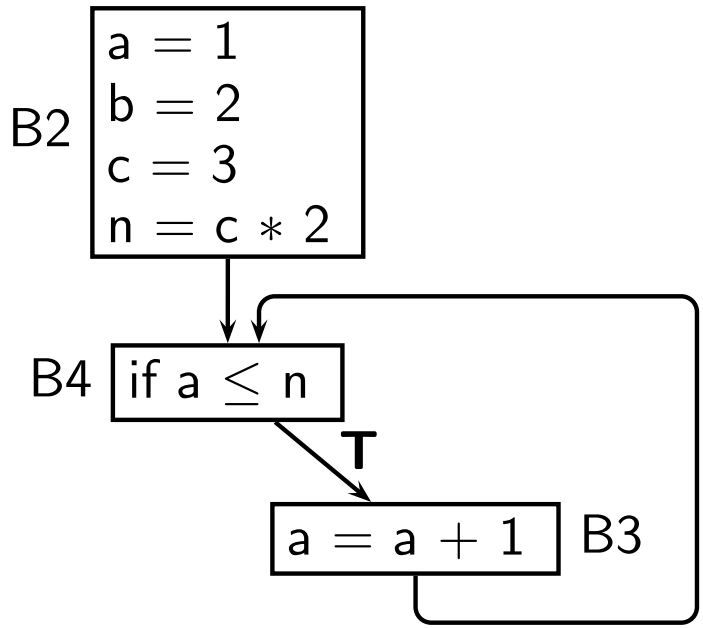


Example Program 1

Program ccp.c

```
int main()
{ int a, b, c, n;

  a = 1;
  b = 2;
  c = 3;
  n = c*2;
  while (a <= n)
  {
    a = a+1;
  }
  if (a < 12)
    a = a+b+c;
  return a;
}
```

Control flow graph

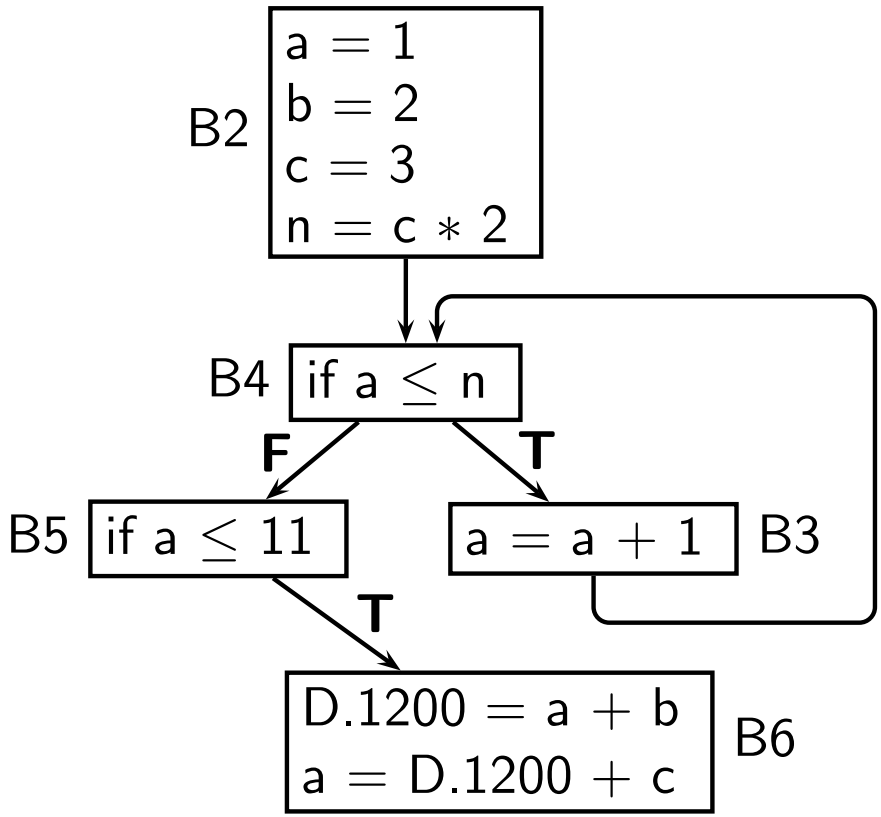


Example Program 1

Program ccp.c

```
int main()
{ int a, b, c, n;

  a = 1;
  b = 2;
  c = 3;
  n = c*2;
  while (a <= n)
  {
    a = a+1;
  }
  if (a < 12)
    a = a+b+c;
  return a;
}
```

Control flow graph

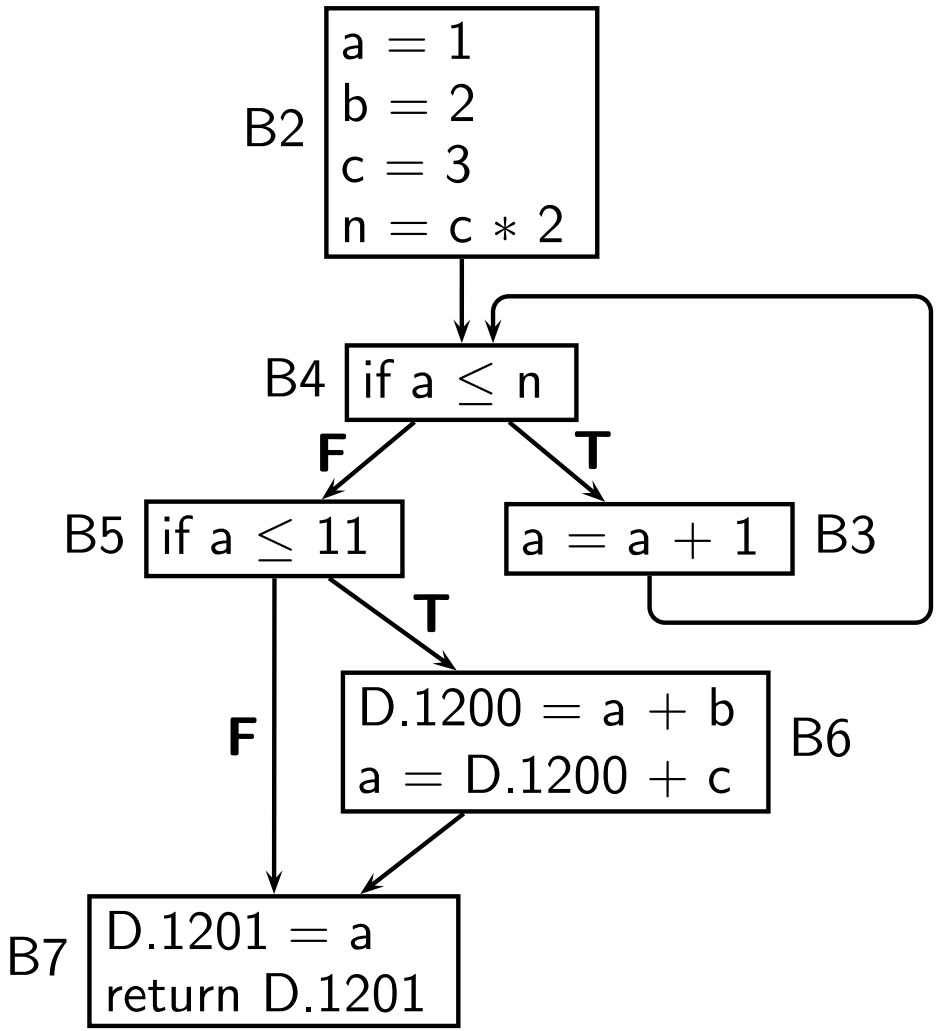


Example Program 1

Program ccp.c

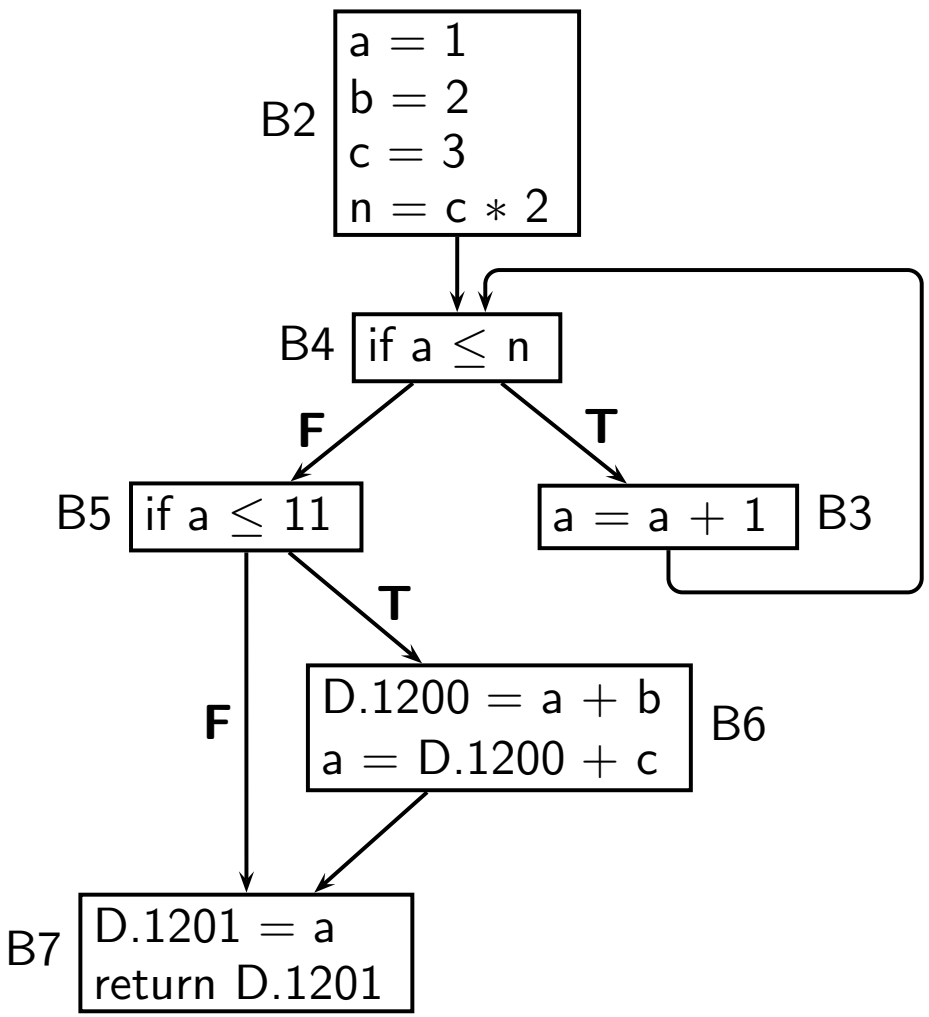
```
int main()
{ int a, b, c, n;

  a = 1;
  b = 2;
  c = 3;
  n = c*2;
  while (a <= n)
  {
    a = a+1;
  }
  if (a < 12)
    a = a+b+c;
  return a;
}
```

Control flow graph



Control Flow Graph: Pictorial and Textual View

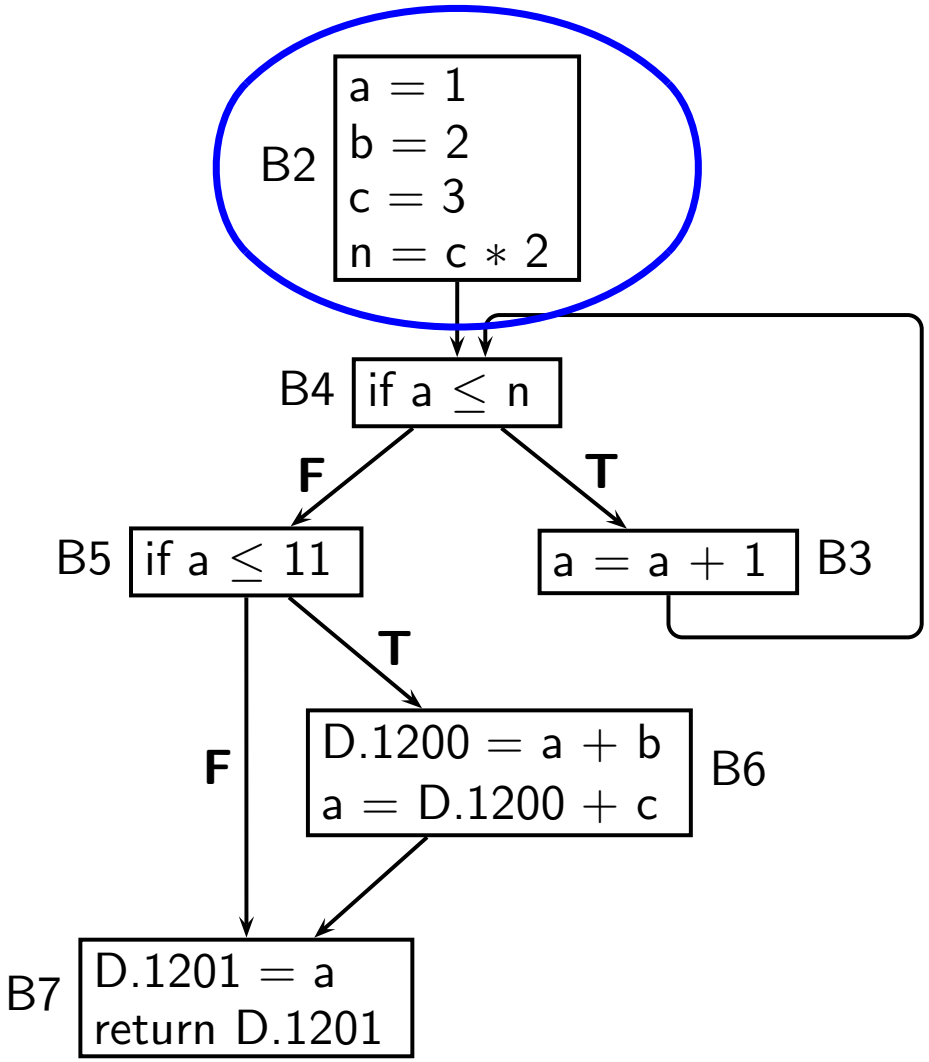
Control flow graph



Dump file ccp.c.014t.cfg

Control Flow Graph: Pictorial and Textual View

Control flow graph

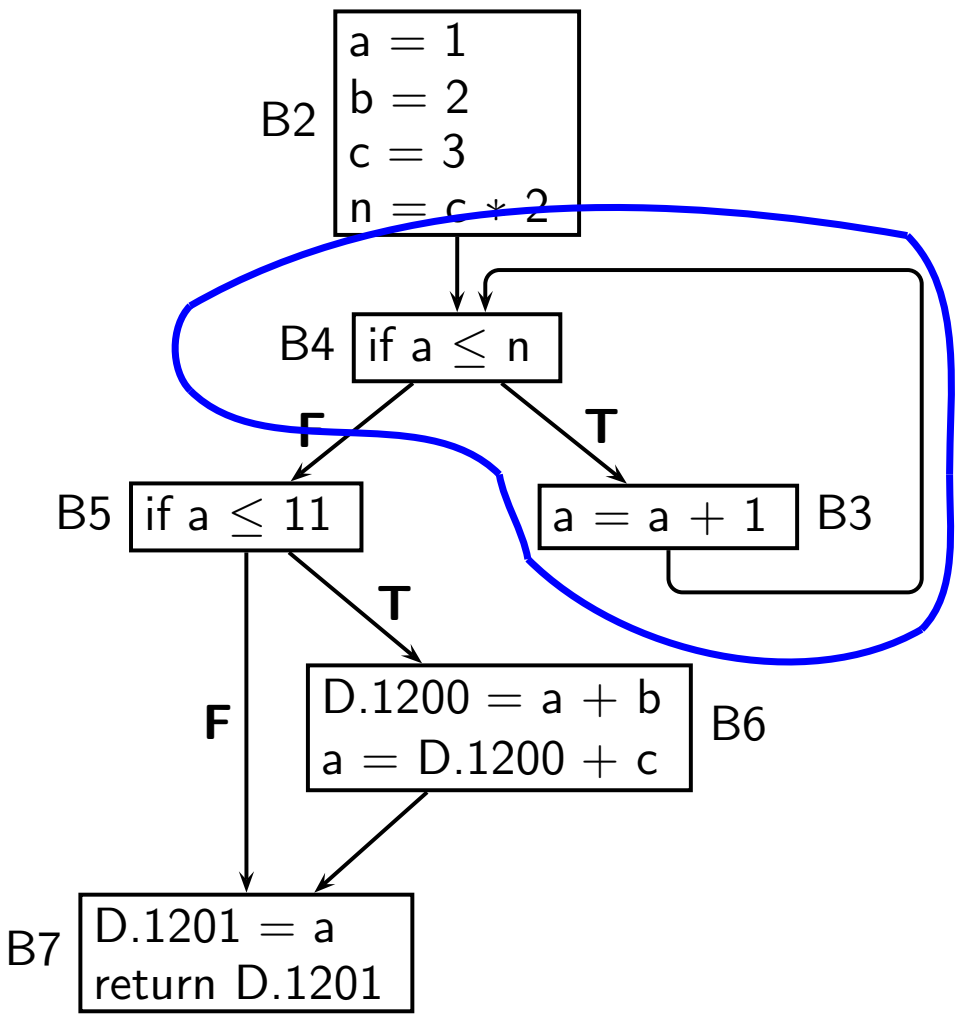


Dump file ccp.c.014t.cfg

```
<bb 2>:
a = 1;
b = 2;
c = 3;
n = c * 2;
goto <bb 4>;
```

Control Flow Graph: Pictorial and Textual View

Control flow graph

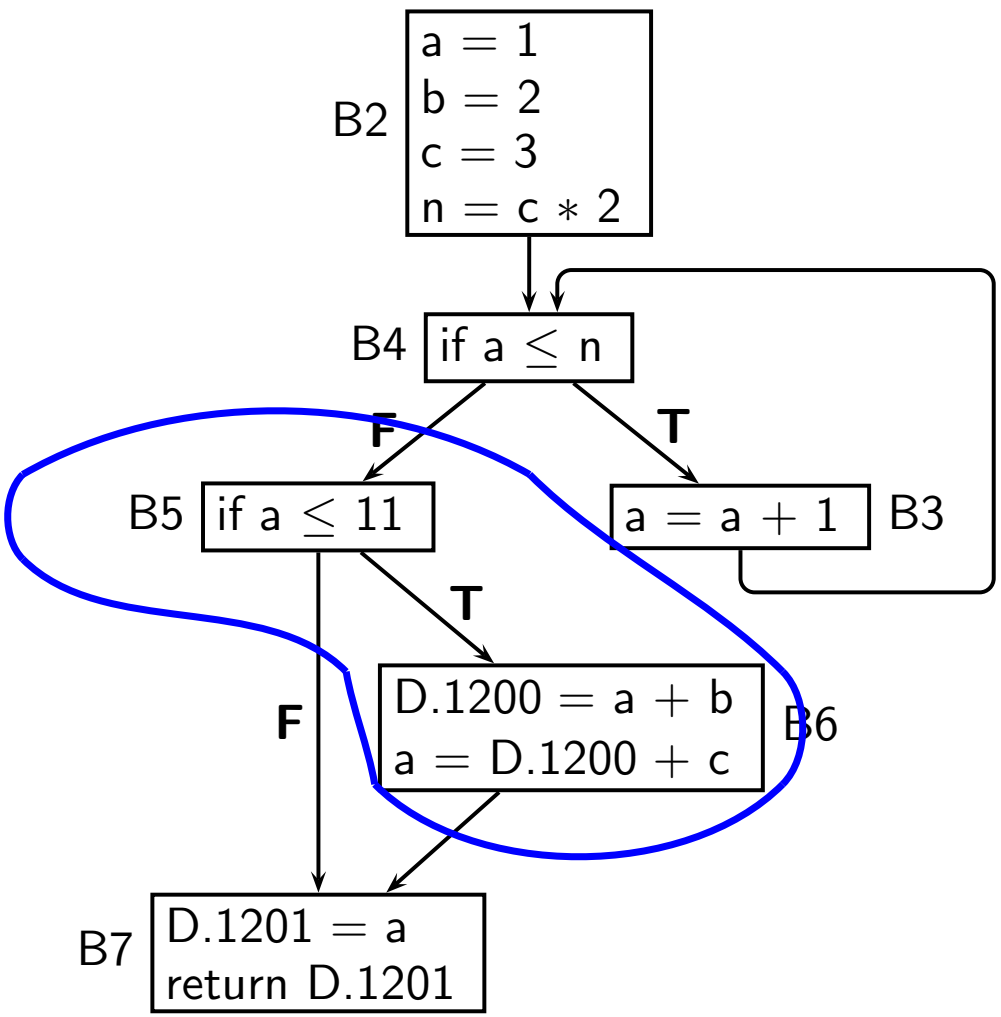


Dump file ccp.c.014t.cfg

```
<bb 3>:
a = a + 1;

<bb 4>:
if (a <= n)
   goto <bb 3>;
else
   goto <bb 5>;
```

Control Flow Graph: Pictorial and Textual View

Control flow graph

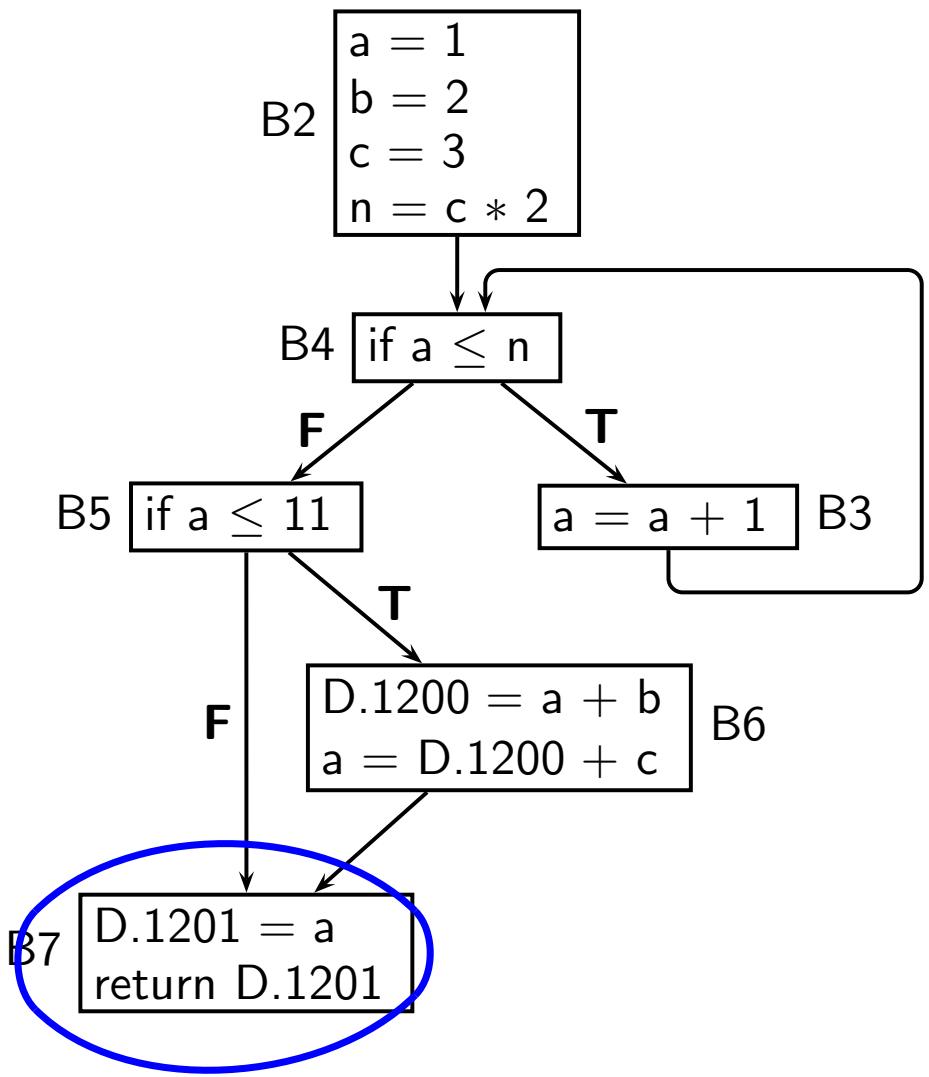


Dump file ccp.c.014t.cfg

```
<bb 5>:
if (a <= 11)
    goto <bb 6>;
else
    goto <bb 7>;

<bb 6>:
D.1200 = a + b;
a = D.1200 + c;
```

Control Flow Graph: Pictorial and Textual View

Control flow graph



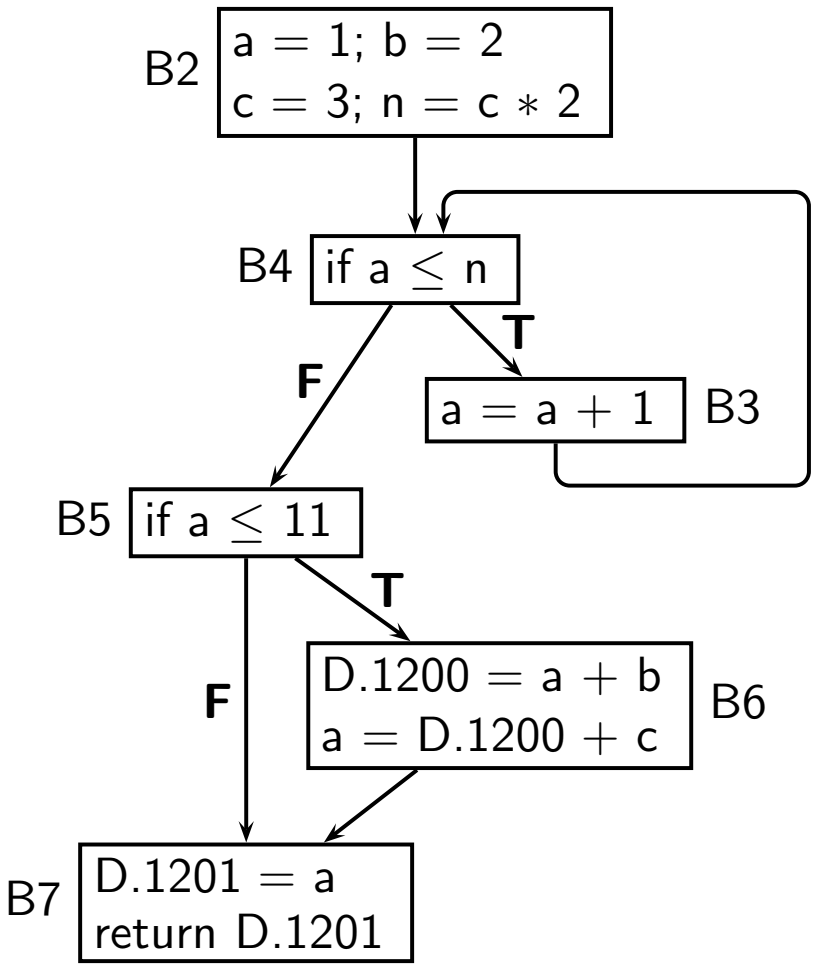
Dump file ccp.c.014t.cfg

```
<bb 7>:
D.1201 = a;
return D.1201;
```

Single Static Assignment (SSA) Form

Control flow graph

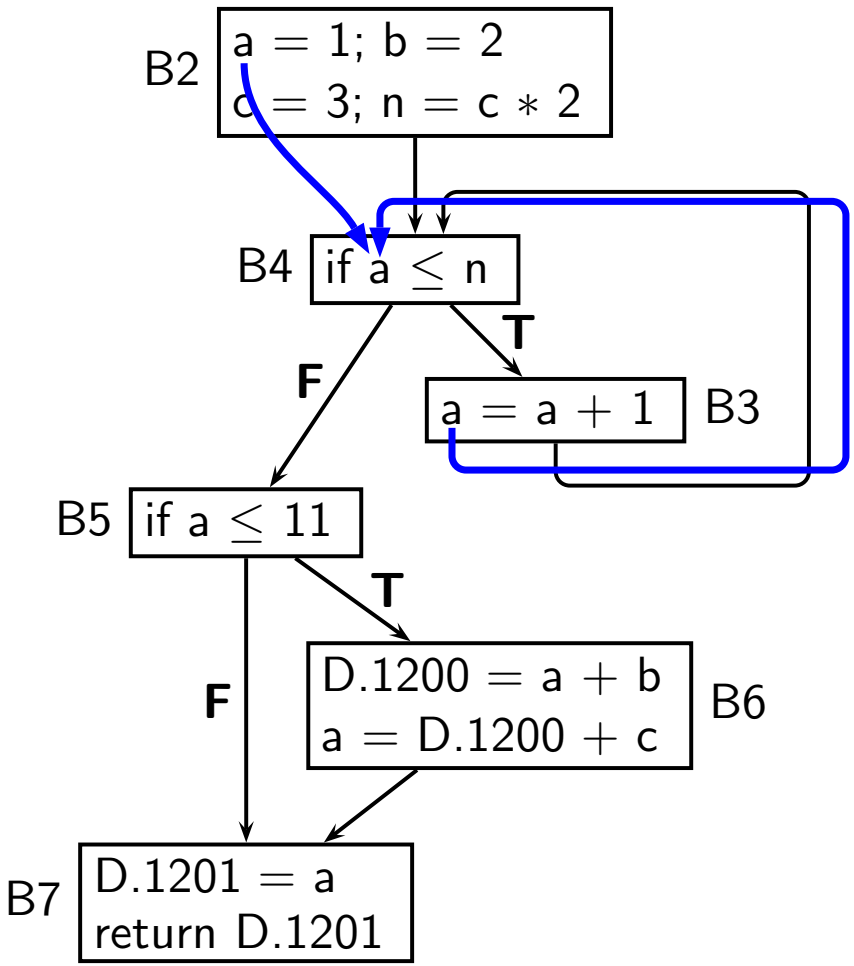
SSA Form



Single Static Assignment (SSA) Form

Control flow graph

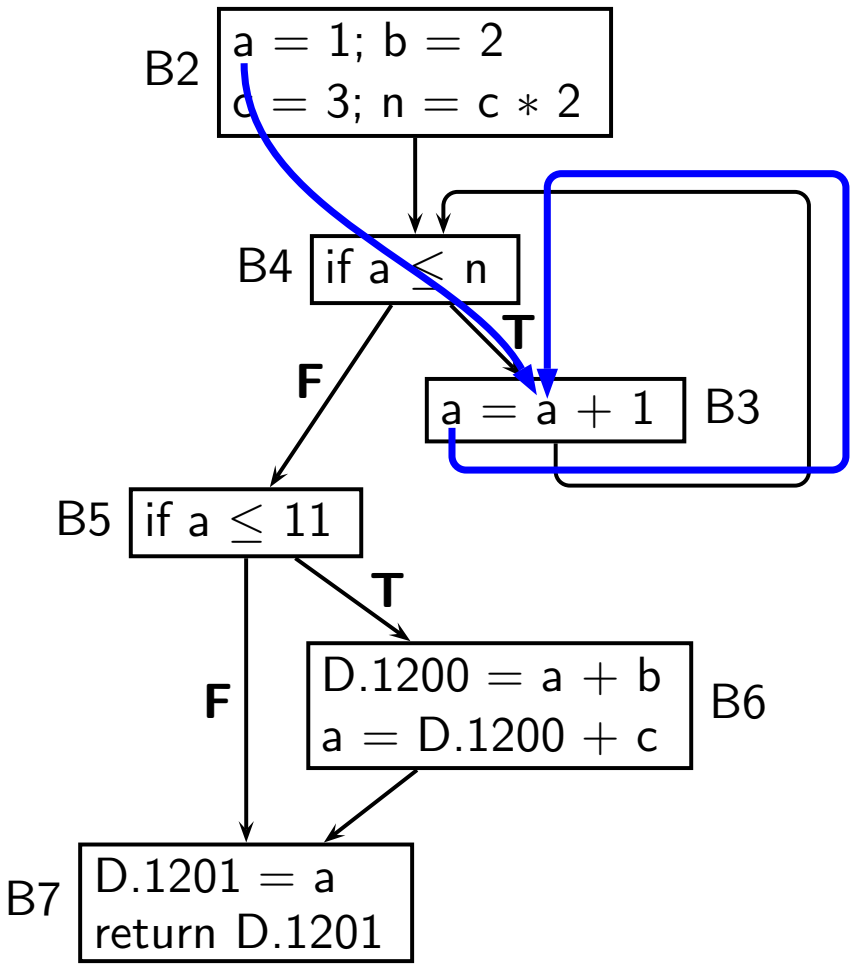
SSA Form



Single Static Assignment (SSA) Form

Control flow graph

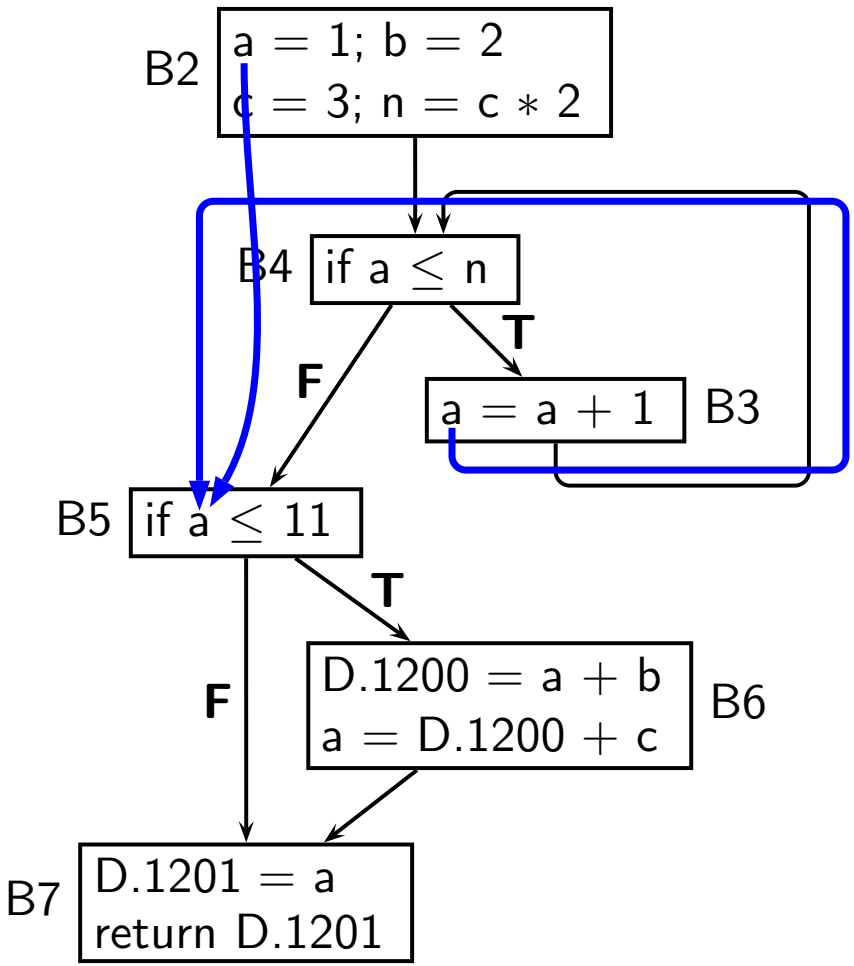
SSA Form



Single Static Assignment (SSA) Form

Control flow graph

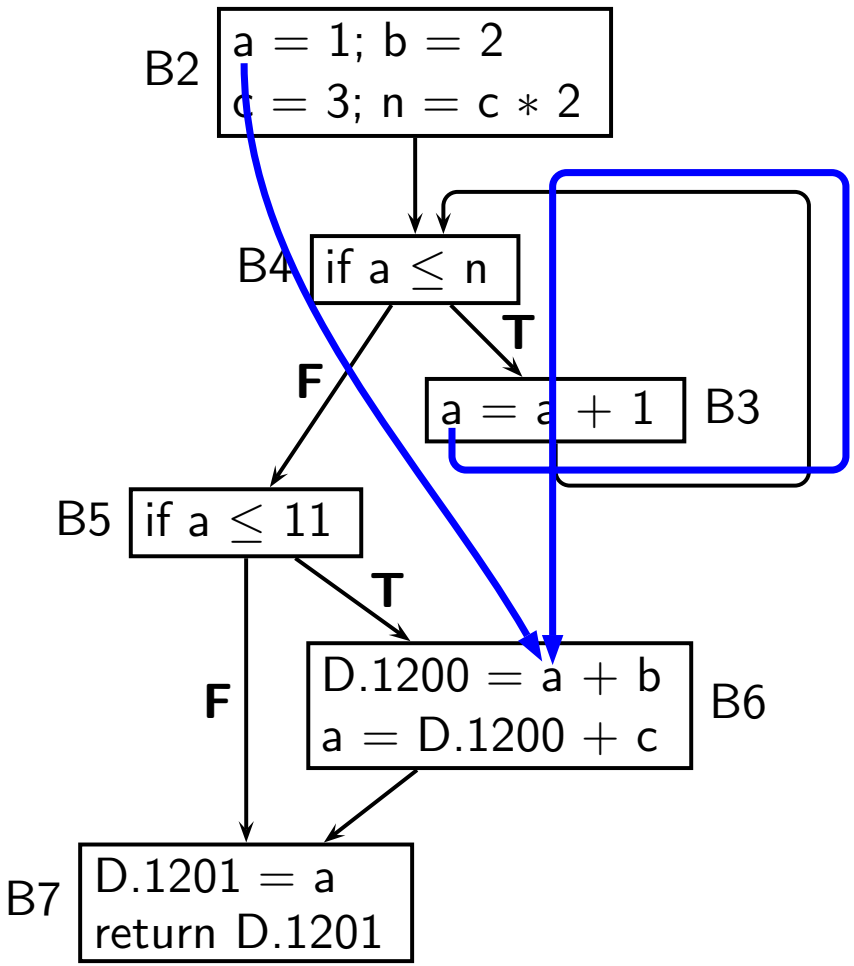
SSA Form



Single Static Assignment (SSA) Form

Control flow graph

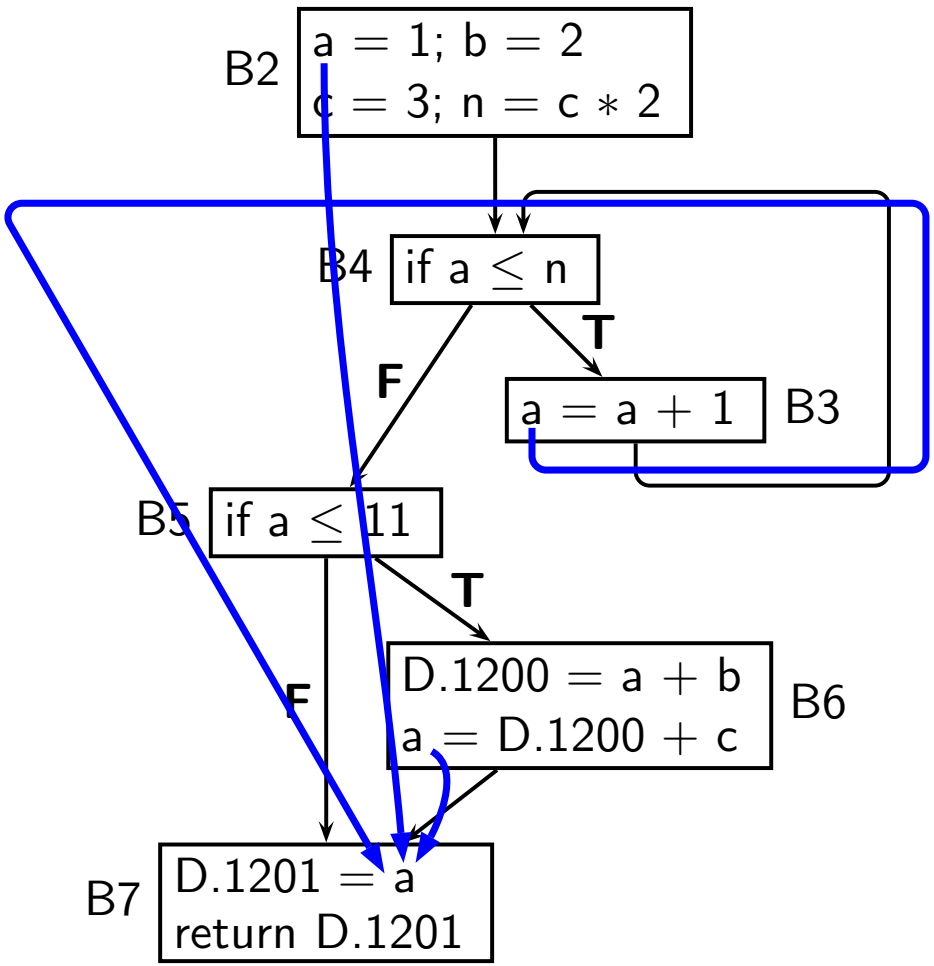
SSA Form



Single Static Assignment (SSA) Form

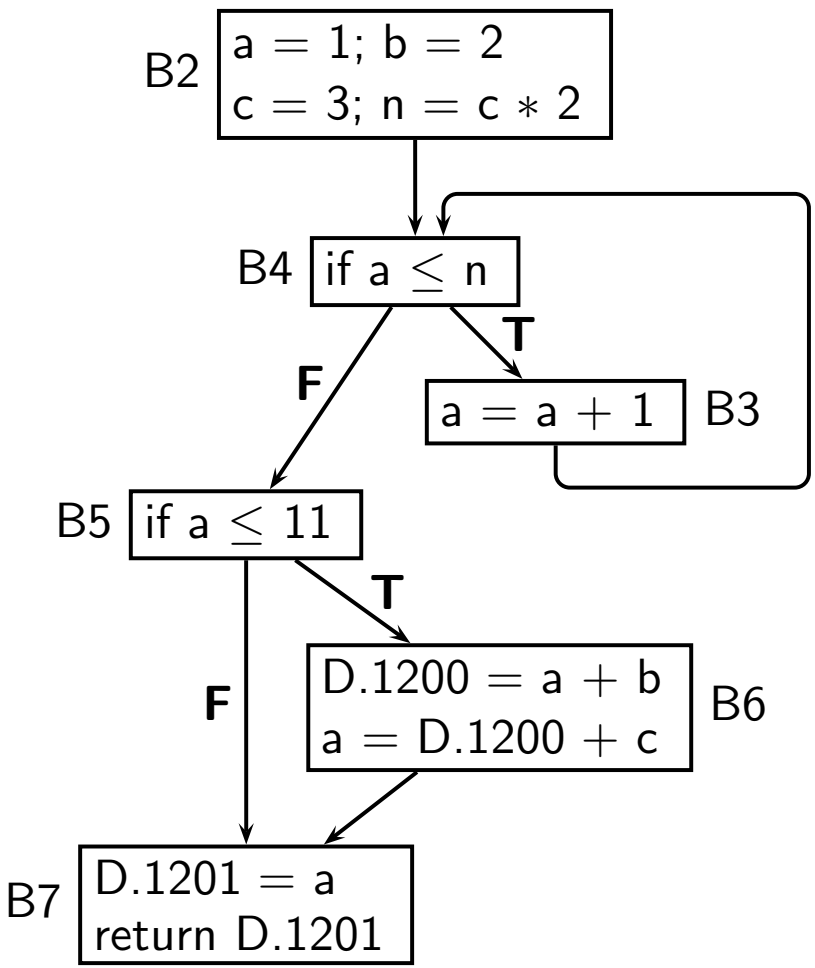
Control flow graph

SSA Form

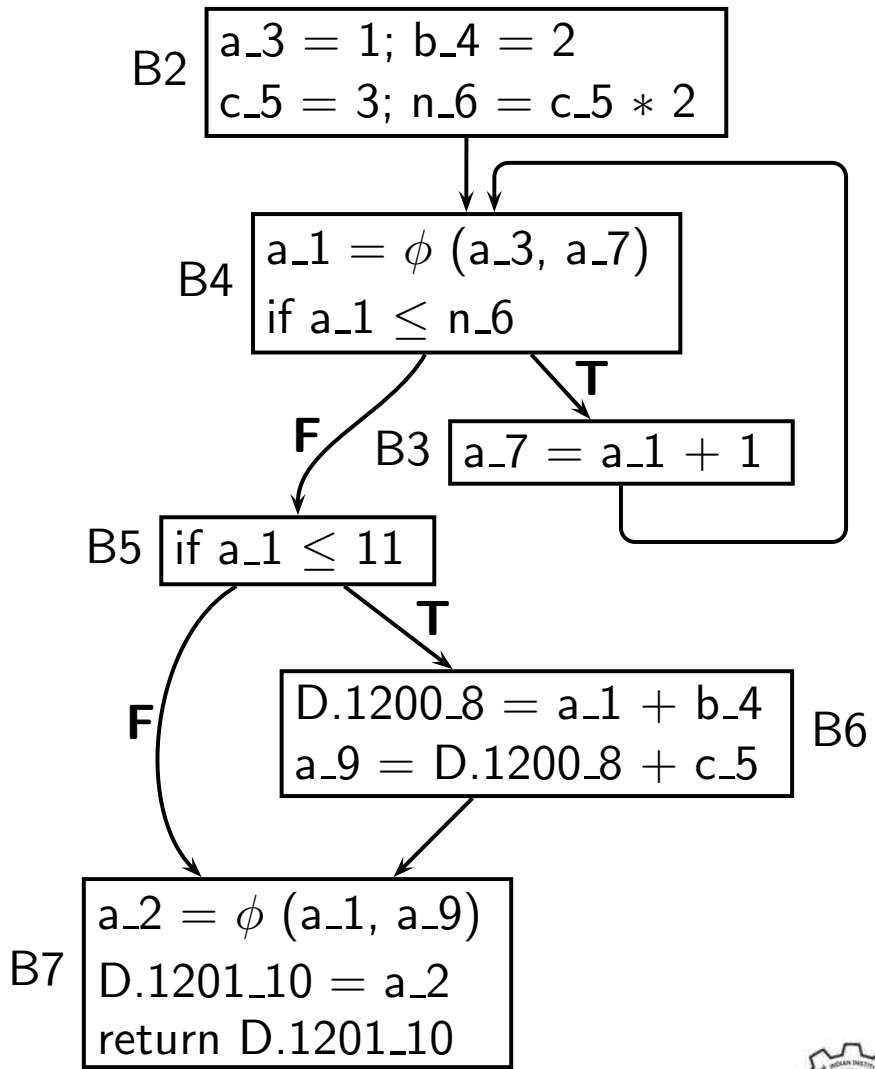


Single Static Assignment (SSA) Form

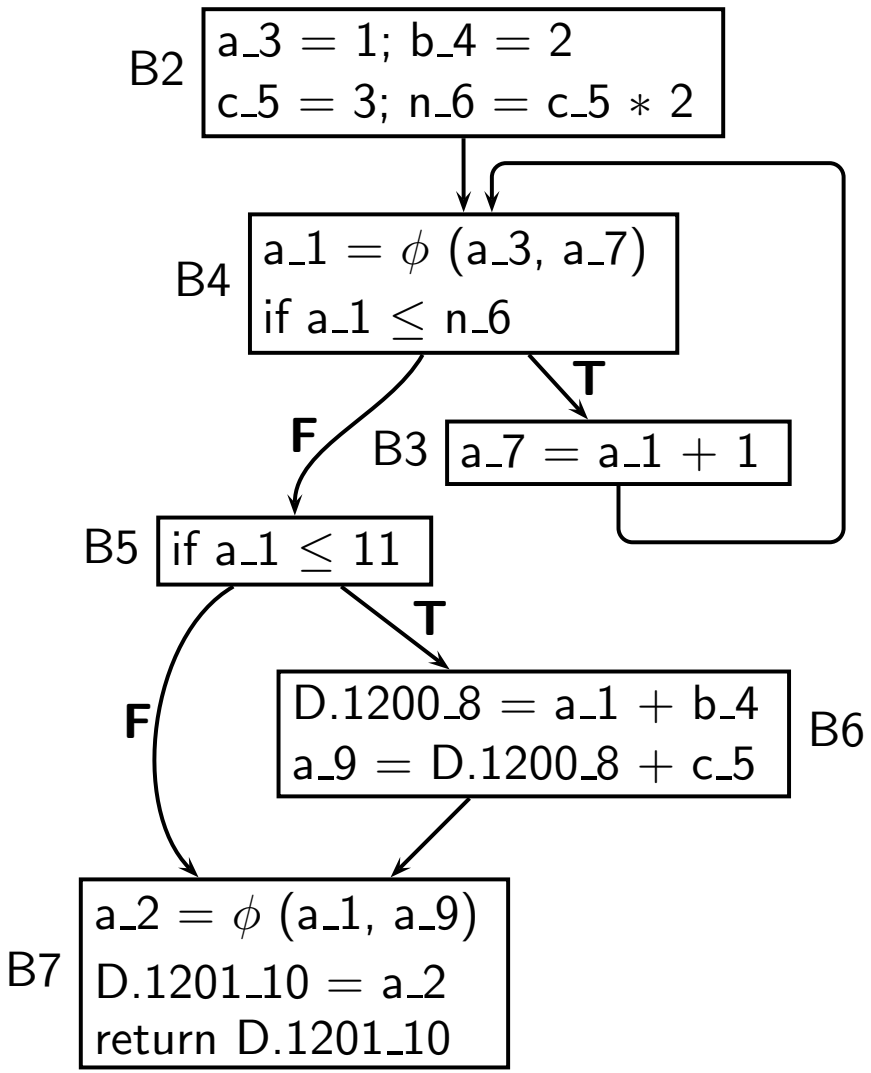
Control flow graph



SSA Form



Properties of SSA Form

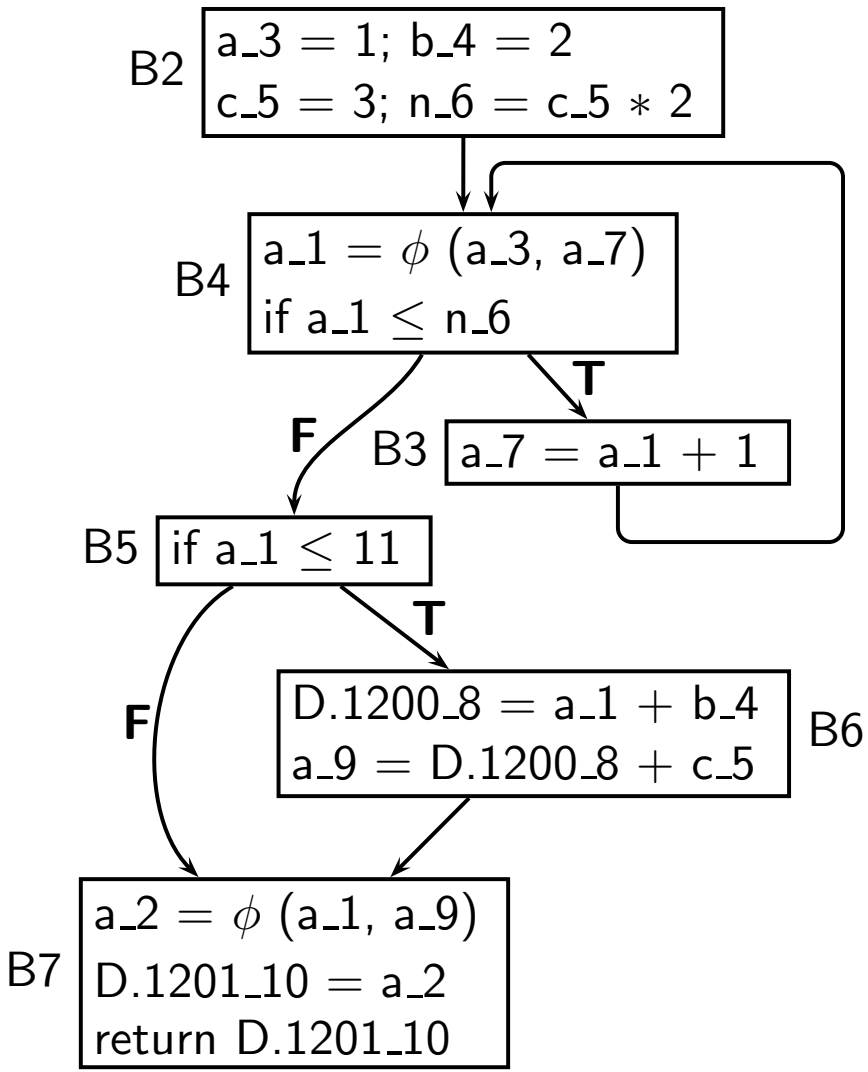


- A ϕ function is a multiplexer or a selection function
- Every use of a variable corresponds to a unique definition of the variable
- For every use, the definition is guaranteed to appear on every path leading to the use

SSA construction algorithm is expected to insert as few ϕ functions as possible to ensure the above properties

SSA Form: Pictorial and Textual View

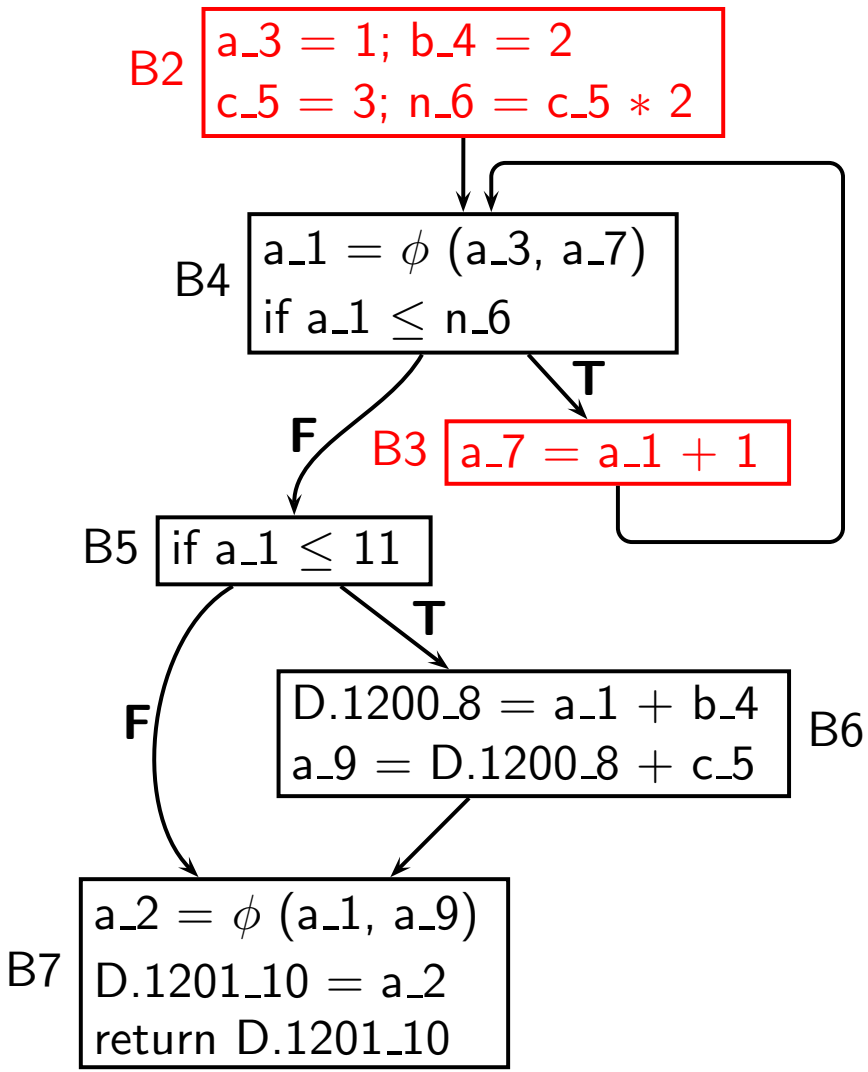
CFG in SSA form



Dump file ccp.c.018t.ssa

SSA Form: Pictorial and Textual View

CFG in SSA form

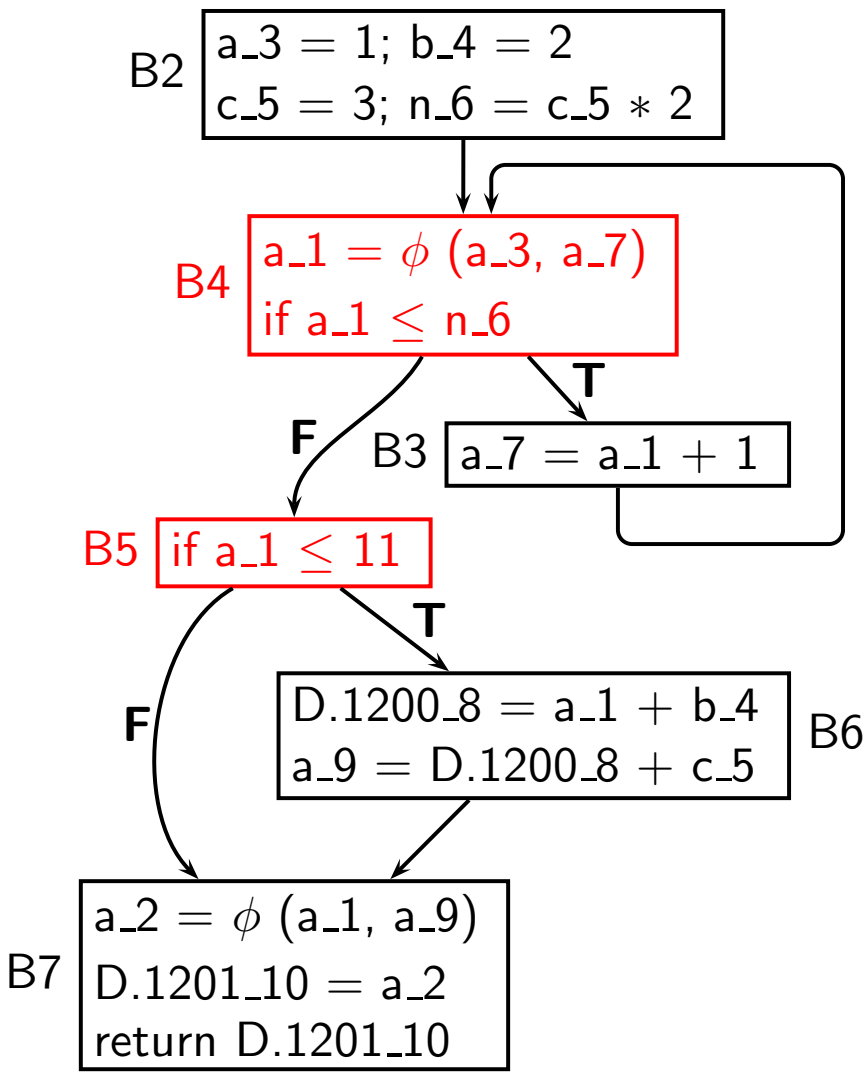


Dump file ccp.c.018t.ssa

```
  <bb 2>:
  a_3 = 1;
  b_4 = 2;
  c_5 = 3;
  n_6 = c_5 * 2;
  goto <bb 4>;

<bb 3>:
  a_7 = a_1 + 1;
```

SSA Form: Pictorial and Textual View

CFG in SSA form

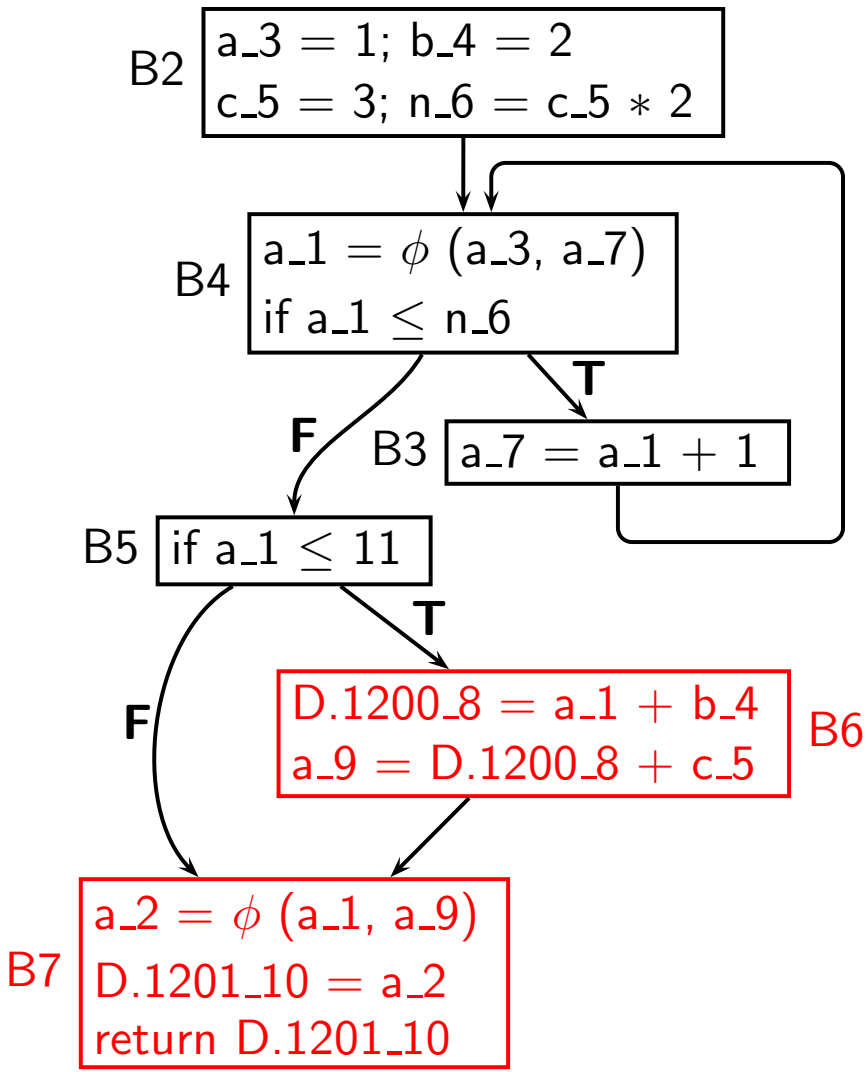


Dump file ccp.c.018t.ssa

```
<bb 4>:
  # a_1 = PHI <a_3(2), a_7(3)>
  if (a_1 <= n_6)
    goto <bb 3>;
  else
    goto <bb 5>;

<bb 5>:
  if (a_1 <= 11)
    goto <bb 6>;
  else
    goto <bb 7>;
```


SSA Form: Pictorial and Textual View

CFG in SSA form



Dump file ccp.c.018t.ssa

```
<bb 6>:
  D.1200_8 = a_1 + b_4;
  a_9 = D.1200_8 + c_5;

<bb 7>:
  # a_2 = PHI <a_1(5), a_9(6)>
  D.1201_10 = a_2;
  return D.1201_10;
```


A Comparison of CFG and SSA Dumps

Dump file ccp.c.014t.cfg

Dump file ccp.c.018t.ssa

A Comparison of CFG and SSA Dumps

Dump file ccp.c.014t.cfg

```
<bb 2>:
  a = 1;
  b = 2;
  c = 3;
  n = c * 2;
  goto <bb 4>;

<bb 3>:
  a = a + 1;
```

Dump file ccp.c.018t.ssa

```
<bb 2>:
  a_3 = 1;
  b_4 = 2;
  c_5 = 3;
  n_6 = c_5 * 2;
  goto <bb 4>;

<bb 3>:
  a_7 = a_1 + 1;
```

A Comparison of CFG and SSA Dumps

Dump file ccp.c.014t.cfg

```
<bb 4>:
  if (a <= n)
     goto <bb 3>;
  else
     goto <bb 5>;

<bb 5>:
  if (a <= 11)
      goto <bb 6>;
  else
      goto <bb 7>;
```

Dump file ccp.c.018t.ssa

```
<bb 4>:
  # a_1 = PHI <a_3(2), a_7(3)>
  if (a_1 <= n_6)
    goto <bb 3>;
  else
    goto <bb 5>;

<bb 5>:
  if (a_1 <= 11)
    goto <bb 6>;
  else
    goto <bb 7>;
```

A Comparison of CFG and SSA Dumps

Dump file ccp.c.014t.cfg

```
<bb 6>:
D.1200 = a + b;
a = D.1200 + c;

<bb 7>:
D.1201 = a;
return D.1201;
```

Dump file ccp.c.018t.ssa

```
<bb 6>:
  D.1200_8 = a_1 + b_4;
  a_9 = D.1200_8 + c_5;

<bb 7>:
  # a_2 = PHI <a_1(5), a_9(6)>
  D.1201_10 = a_2;
  return D.1201_10;
```

Copy Renamimg

Input dump: ccp.c.018t.ssa

```
<bb 7>:
  # a_2 = PHI <a_1(5), a_9(6)>
  D.1201_10 = a_2;
  return D.1201_10;
```

Output dump: ccp.c.023t.copyrename1

```
<bb 7>:
  # a_2 = PHI <a_1(5), a_9(6)>
  a_10 = a_2;
  return a_10;
```

First Level Constant and Copy Propagation

Input dump: ccp.c.023t.copyrename1

```
<bb 2>:
  a_3 = 1;
  b_4 = 2;
  c_5 = 3;
  n_6 = c_5 * 2;
  goto <bb 4>;

<bb 3>:
  a_7 = a_1 + 1;

<bb 4>:
  # a_1 = PHI < a_3(2), a_7(3)>
  if (a_1 <= n_6)
    goto <bb 3>;
  else
    goto <bb 5>;
```

Output dump: ccp.c.024t.ccp1

```
<bb 2>:
  a_3 = 1;
  b_4 = 2;
  c_5 = 3;
  n_6 = 6;
  goto <bb 4>;

<bb 3>:
  a_7 = a_1 + 1;

<bb 4>:
  # a_1 = PHI < 1(2), a_7(3)>
  if (a_1 <= 6)
    goto <bb 3>;
  else
    goto <bb 5>;
```

First Level Constant and Copy Propagation

Input dump: ccp.c.023t.copyrename1

```
<bb 2>:
  a_3 = 1;
  b_4 = 2;
  c_5 = 3;
  n_6 = 6;
  goto <bb 4>;

...

<bb 6>:
  D.1200_8 = a_1 + b_4;
  a_9 = D.1200_8 + c_5;
```

Output dump: ccp.c.024t.ccp1

```
<bb 2>:
  a_3 = 1;
  b_4 = 2;
  c_5 = 3;
  n_6 = 6;
  goto <bb 4>;

 ...

<bb 6>:
  D.1200_8 = a_1 + 2;
  a_9 = D.1200_8 + 3;
```

Second Level Copy Propagation

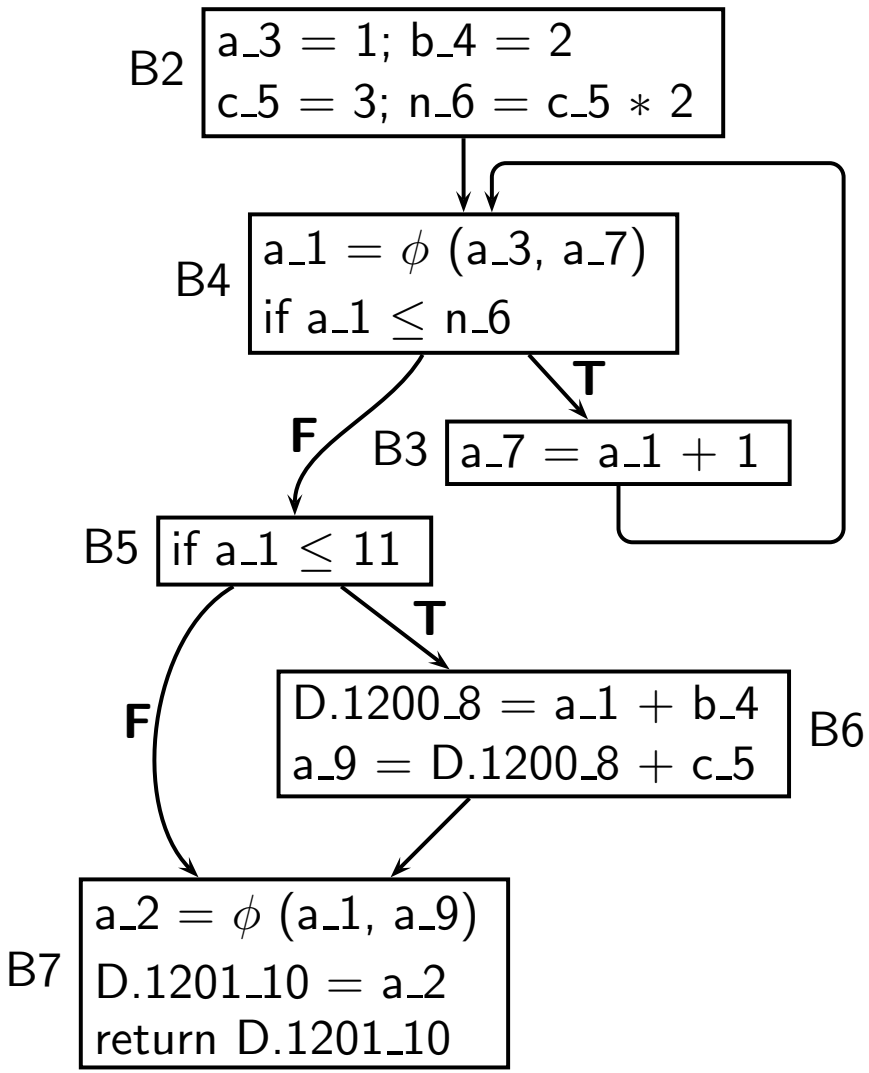
Input dump: ccp.c.024t.ccp1

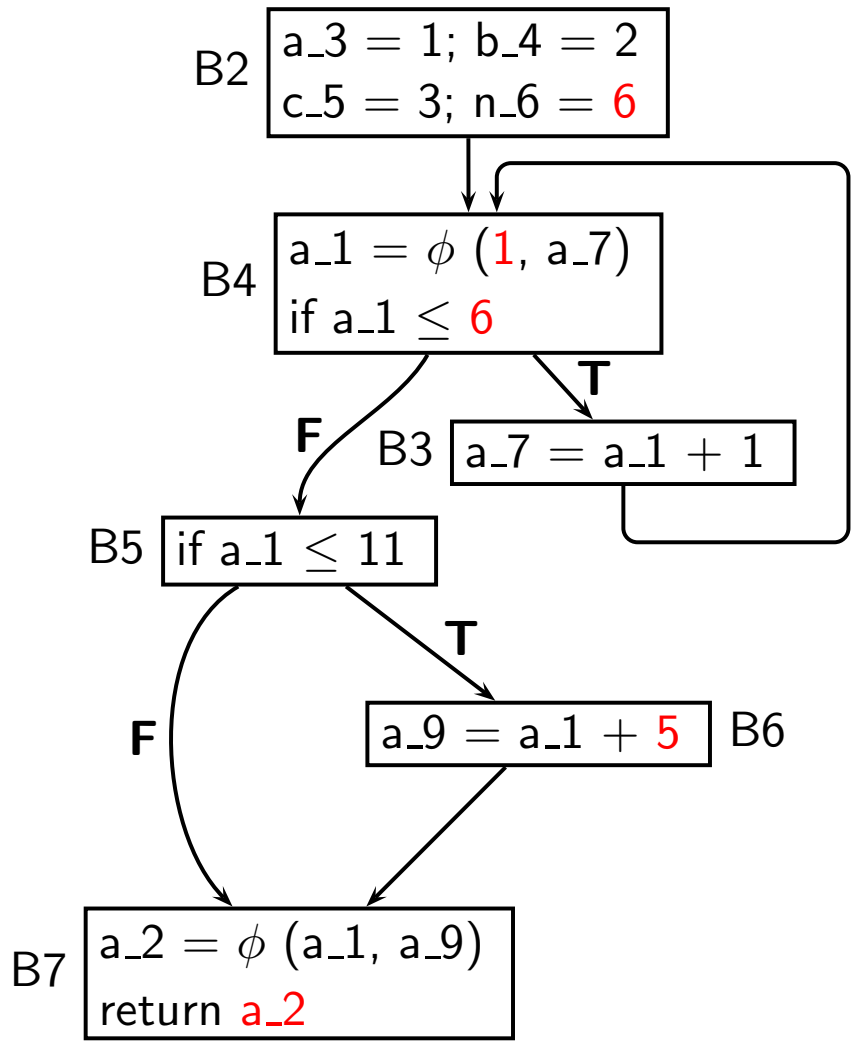
```
<bb 6>:
  D.1200_8 = a_1 + 2;
  a_9 = D.1200_8 + 3;

<bb 7>:
  # a_2 = PHI <a_1(5), a_9(6)>
  a_10 = a_2;
  return a_10;
```

Output dump: ccp.c.031t.copyprop1

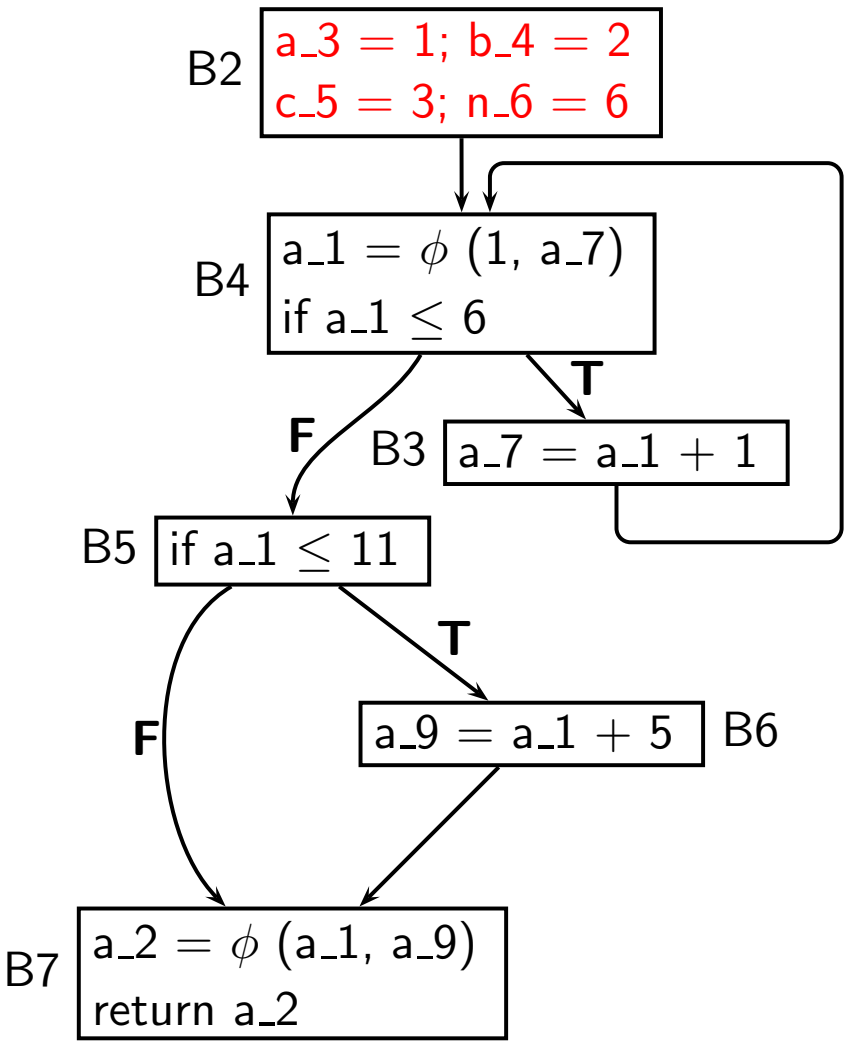
```
<bb 6>:
  a_9 = a_1 + 5;

<bb 7>:
  # a_2 = PHI <a_1(5), a_9(6)>
  return a_2;
```

The Result of Copy Propagation and Renaming



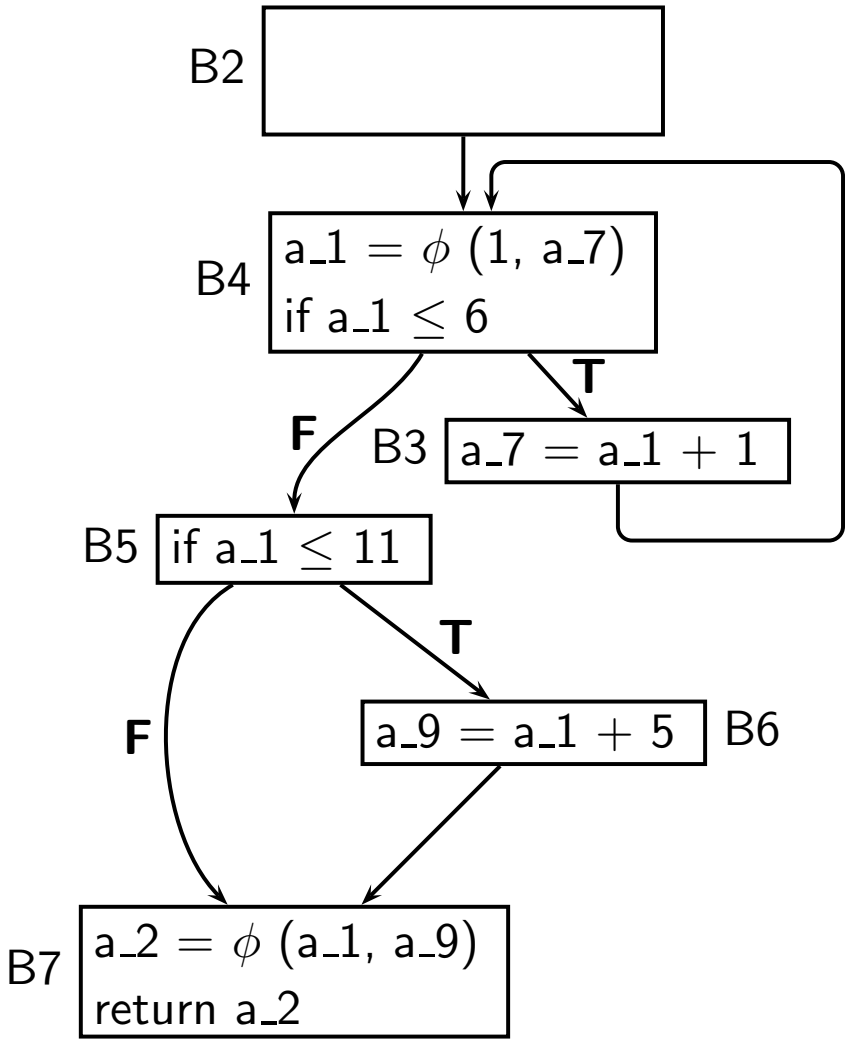


The Result of Copy Propagation and Renaming



- No uses for variables a_3, b_4, c_5, and n_6
- Assignments to these variables can be deleted

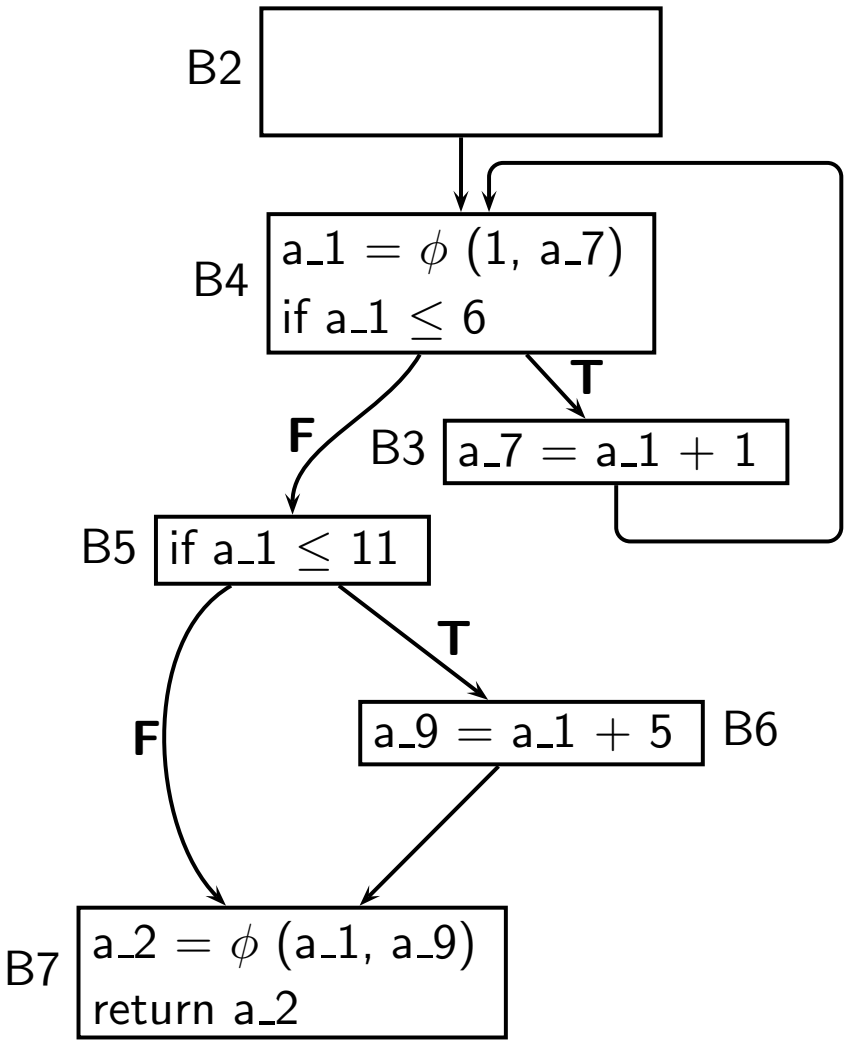
Dead Code Elimination Using Control Dependence



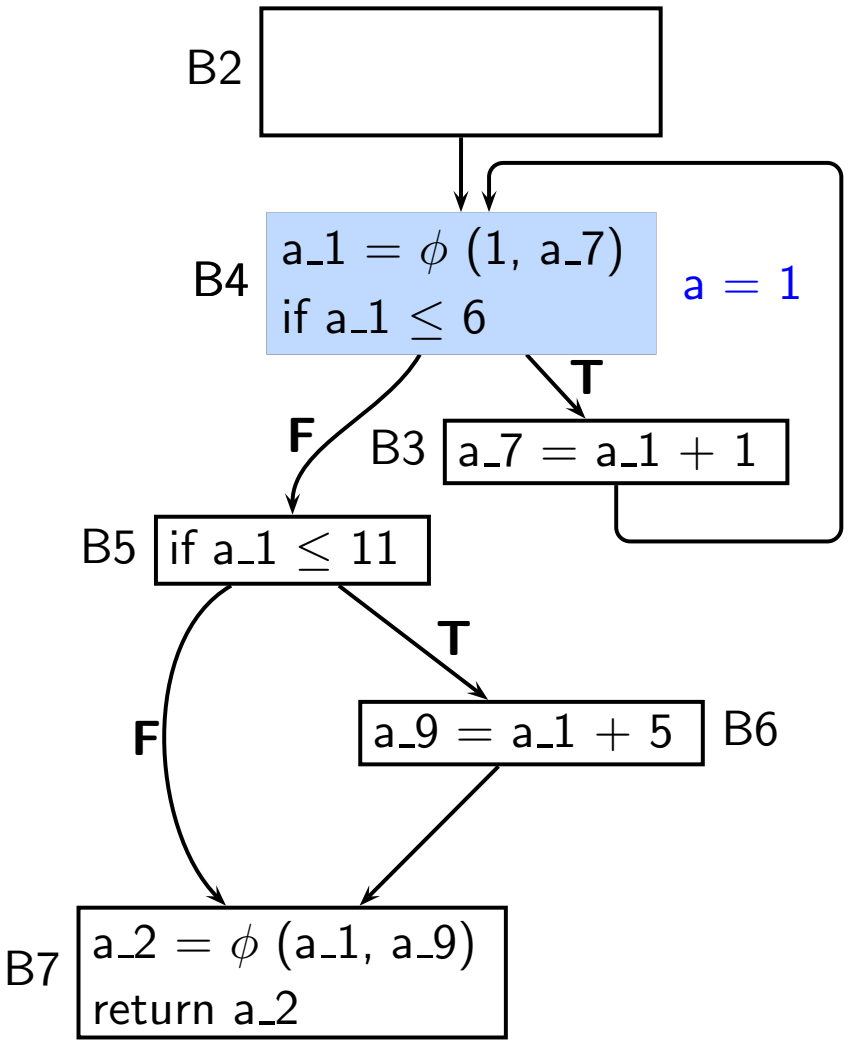
Dump file ccp.c.031t.cddce1

```
<bb 2>:
  goto <bb 4>;
<bb 3>:
  a_7 = a_1 + 1;
<bb 4>:
  # a_1 = PHI <1(2), a_7(3)>
  if (a_1 <= 6) goto <bb 3>;
  else goto <bb 5>;
<bb 5>:
  if (a_1 <= 11) goto <bb 6>;
  else goto <bb 7>;
<bb 6>:
  a_9 = a_1 + 5;
<bb 7>:
  # a_2 = PHI <a_1(5), a_9(6)>
  return a_2;
```

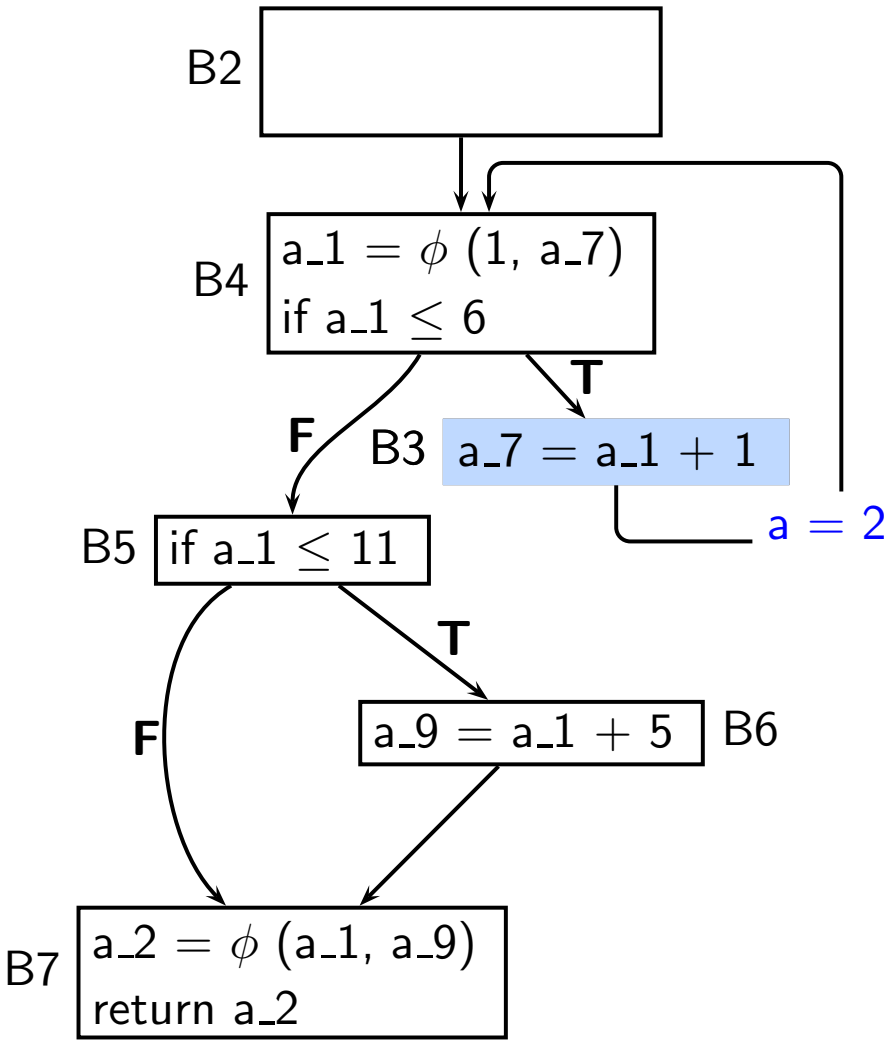
Loop Unrolling



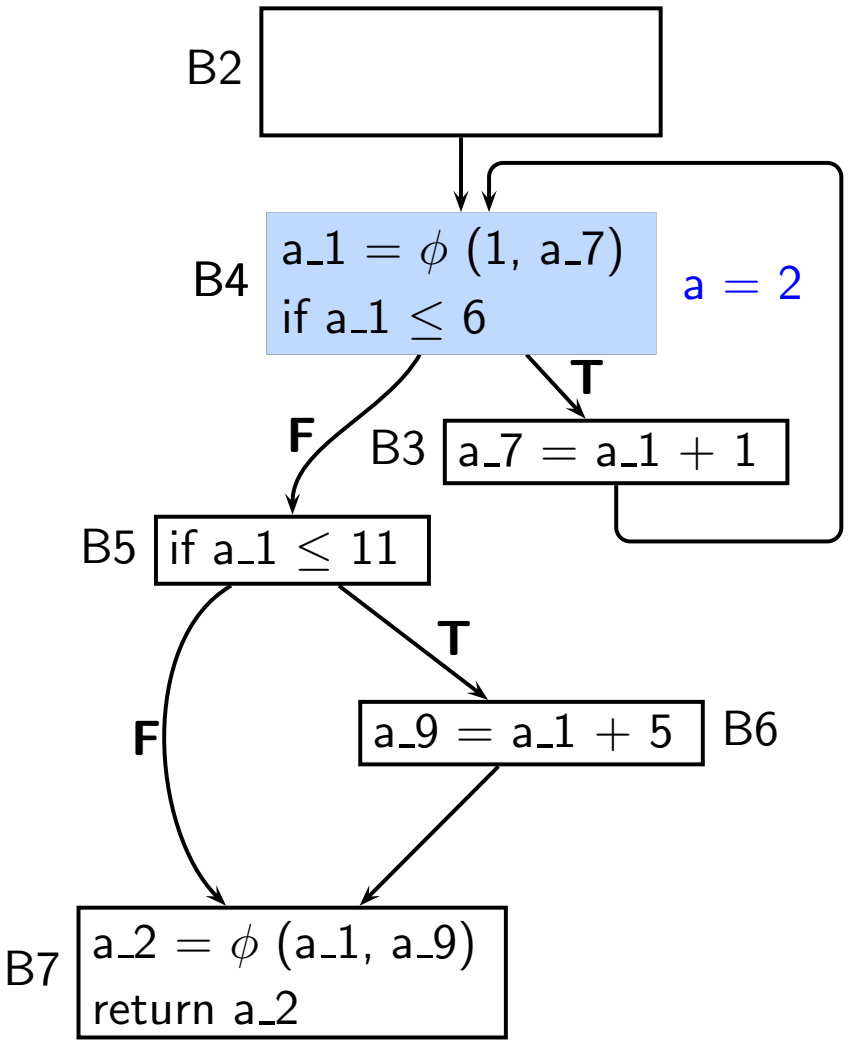
Loop Unrolling



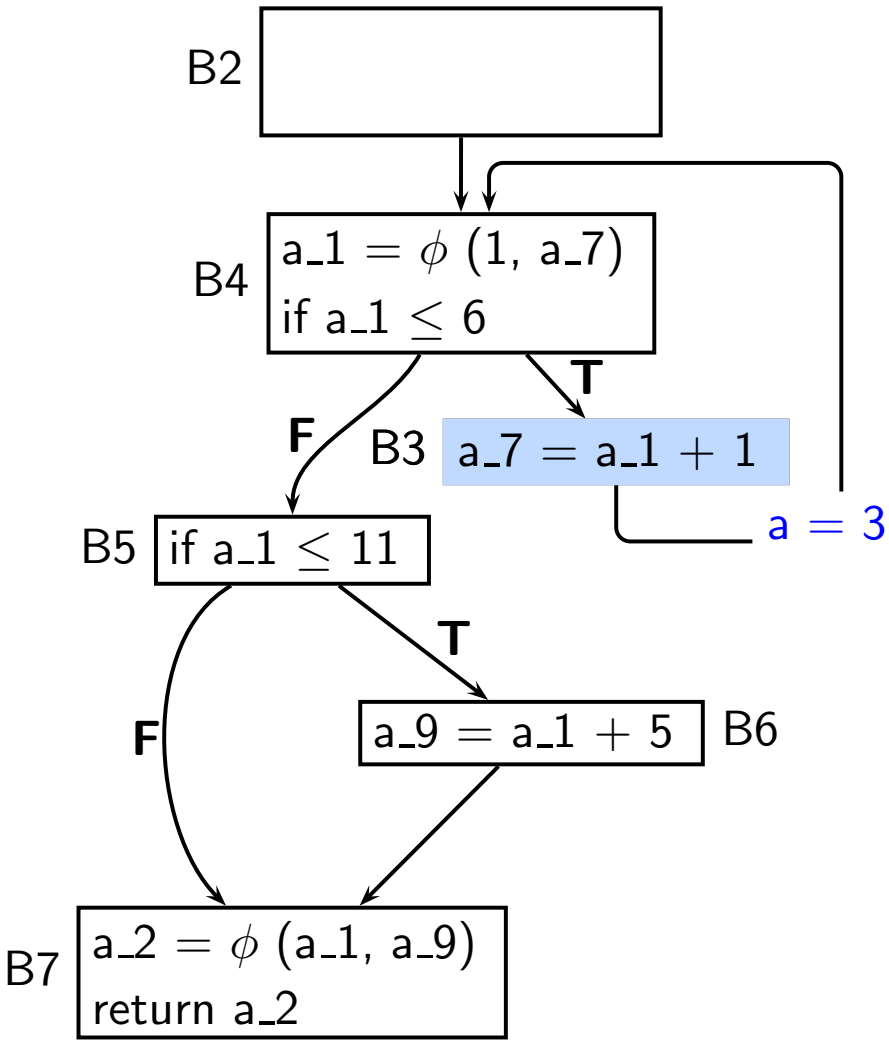
Loop Unrolling



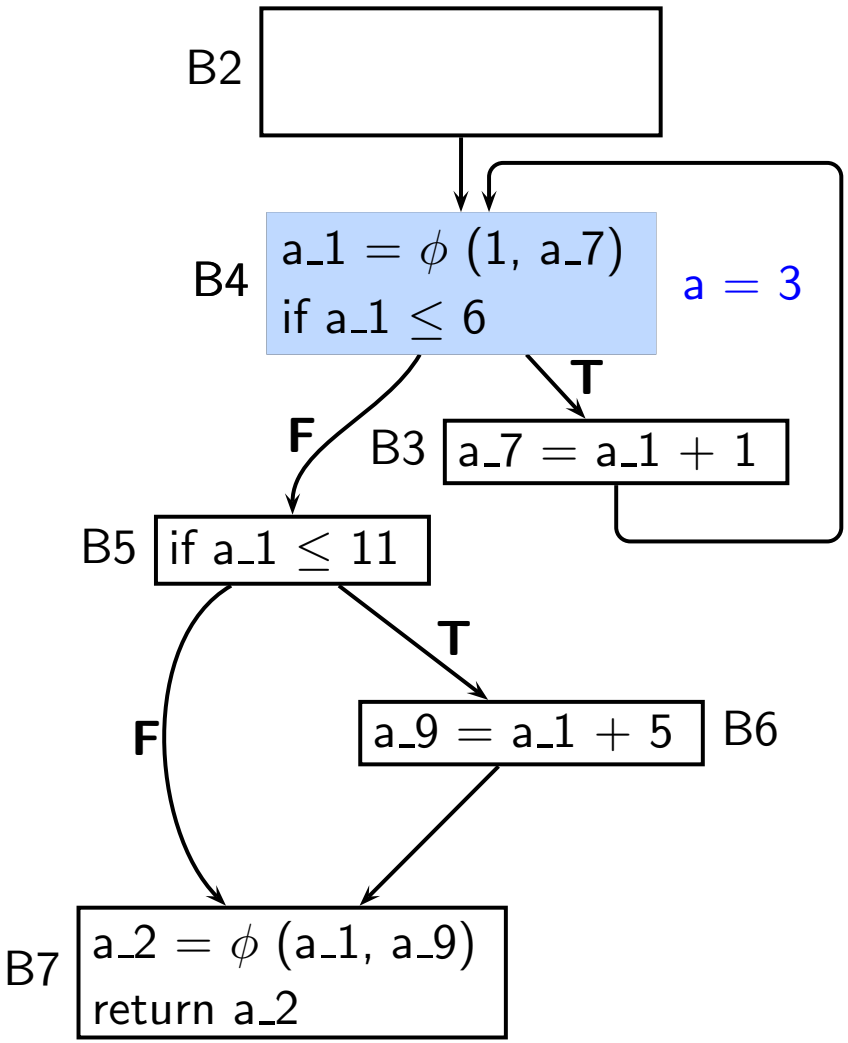
Loop Unrolling



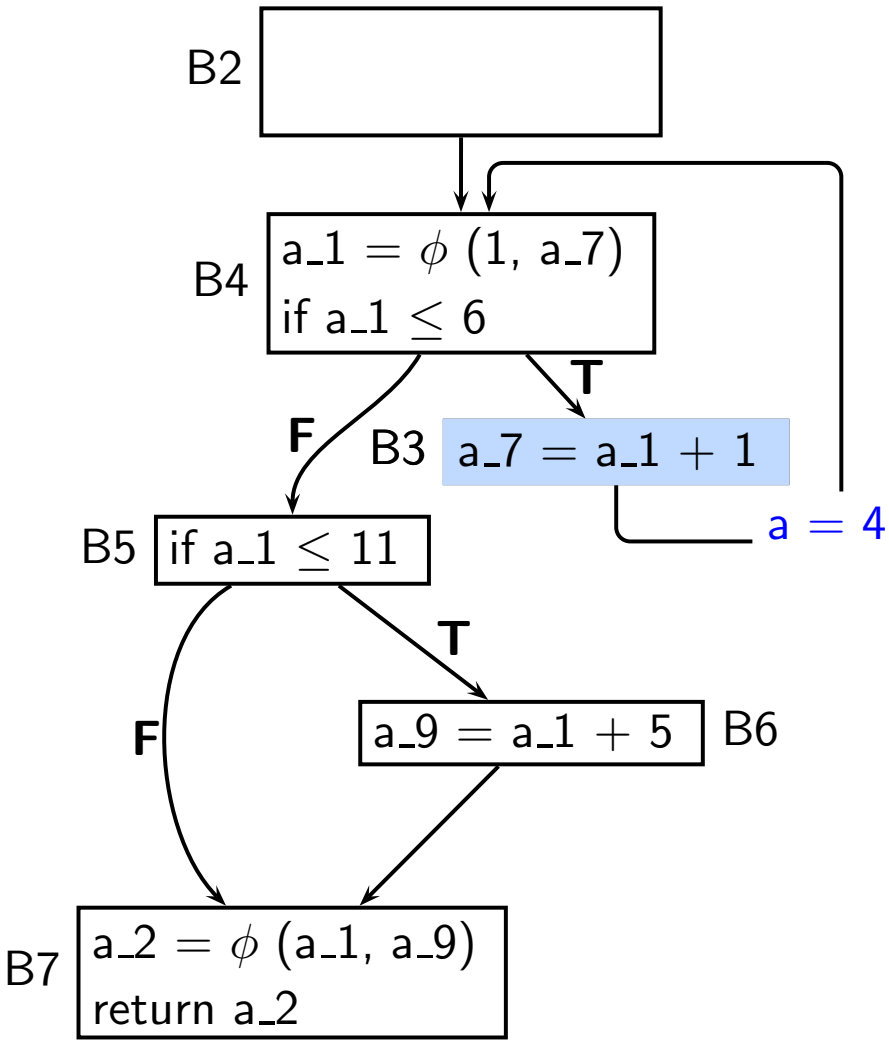
Loop Unrolling



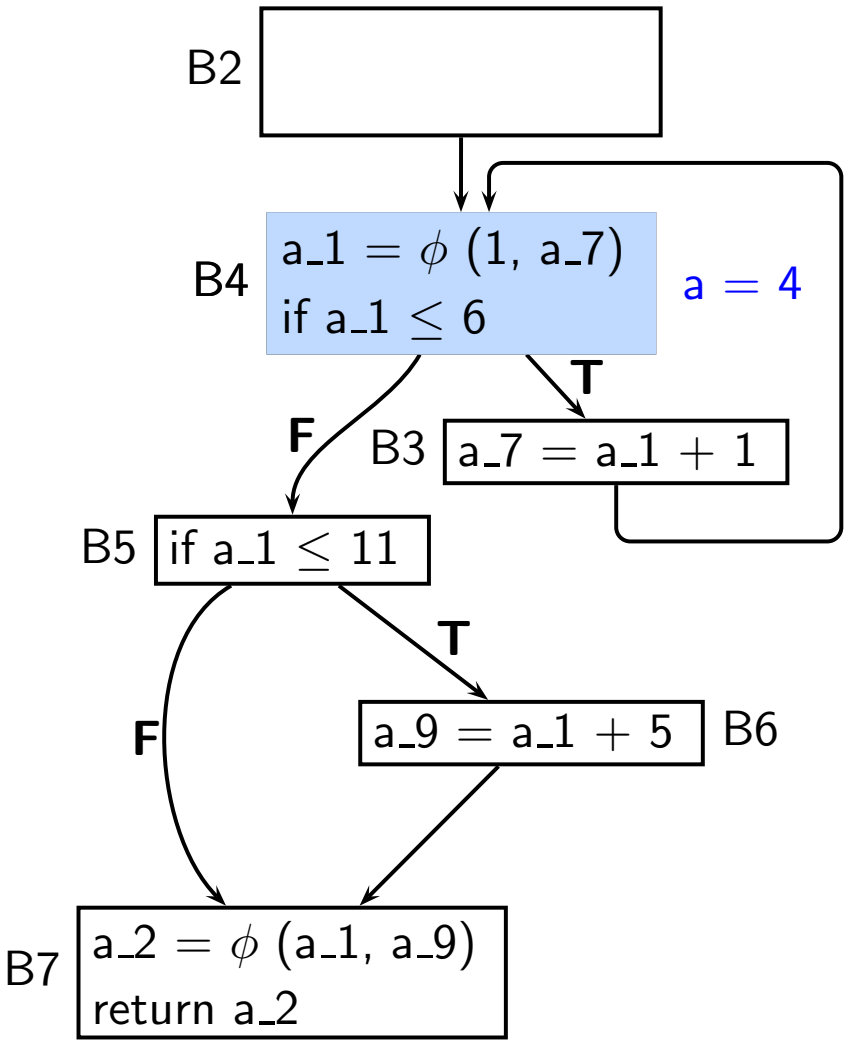
Loop Unrolling



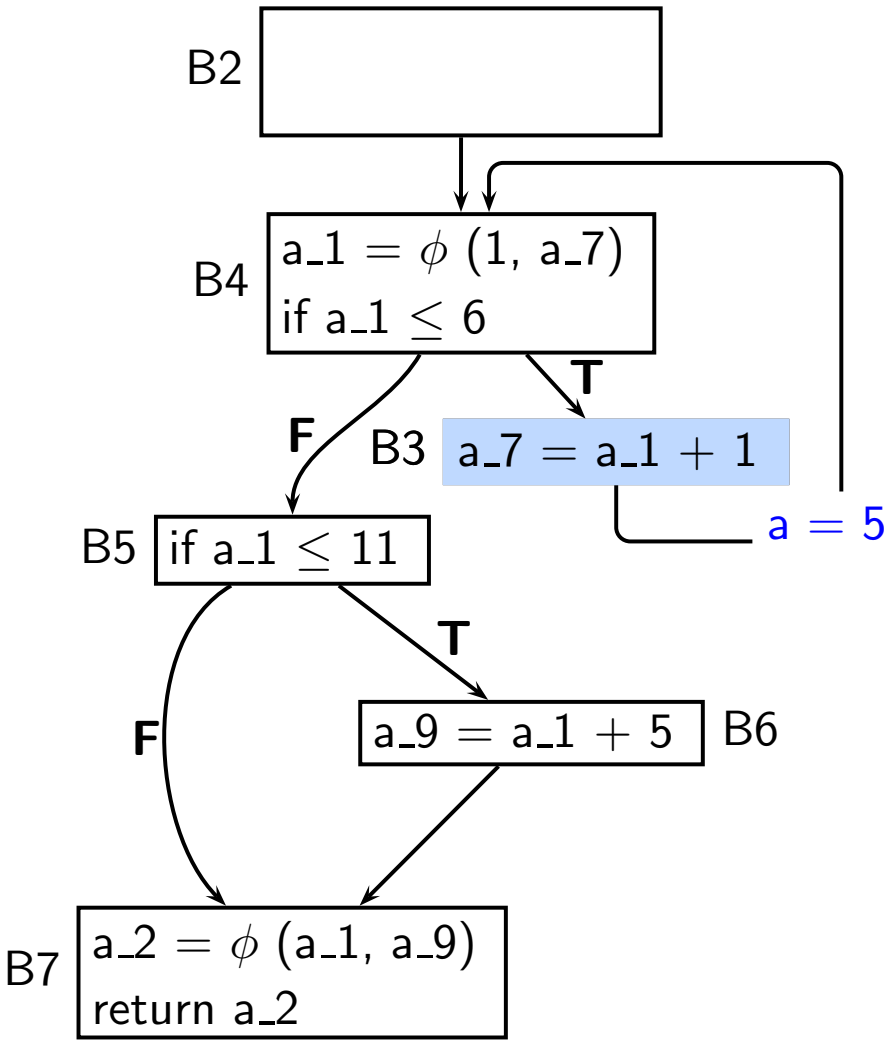
Loop Unrolling



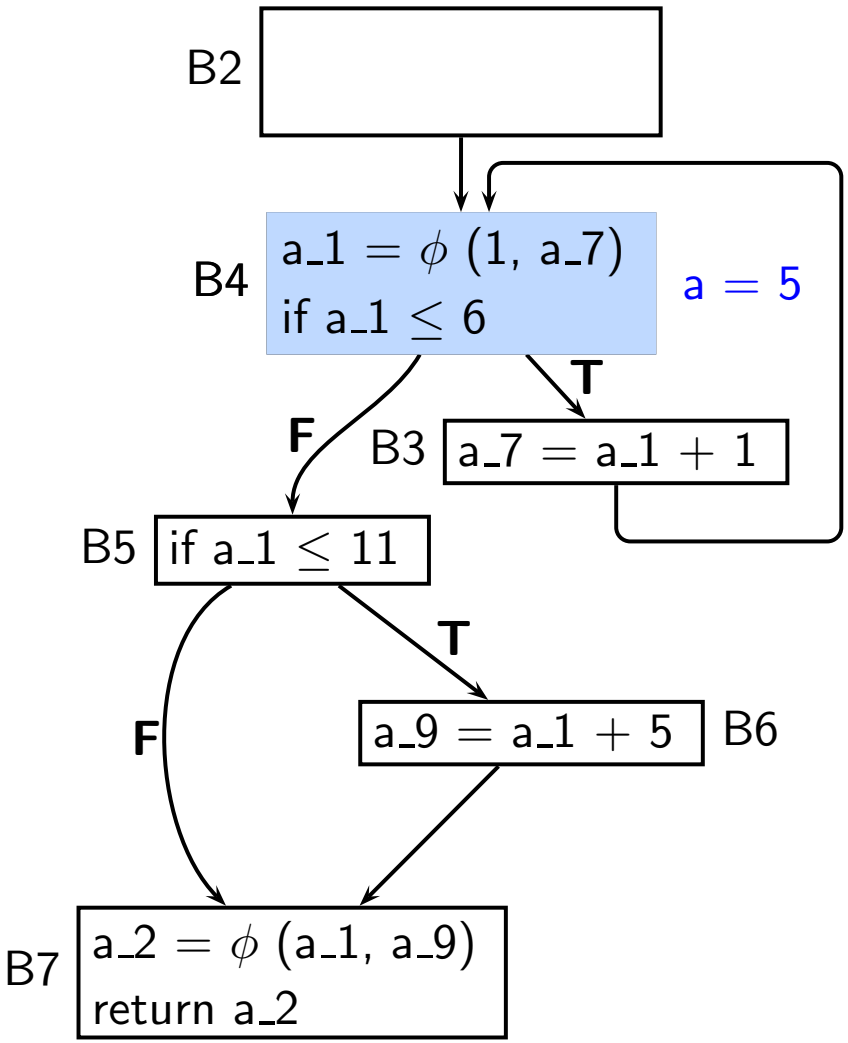
Loop Unrolling



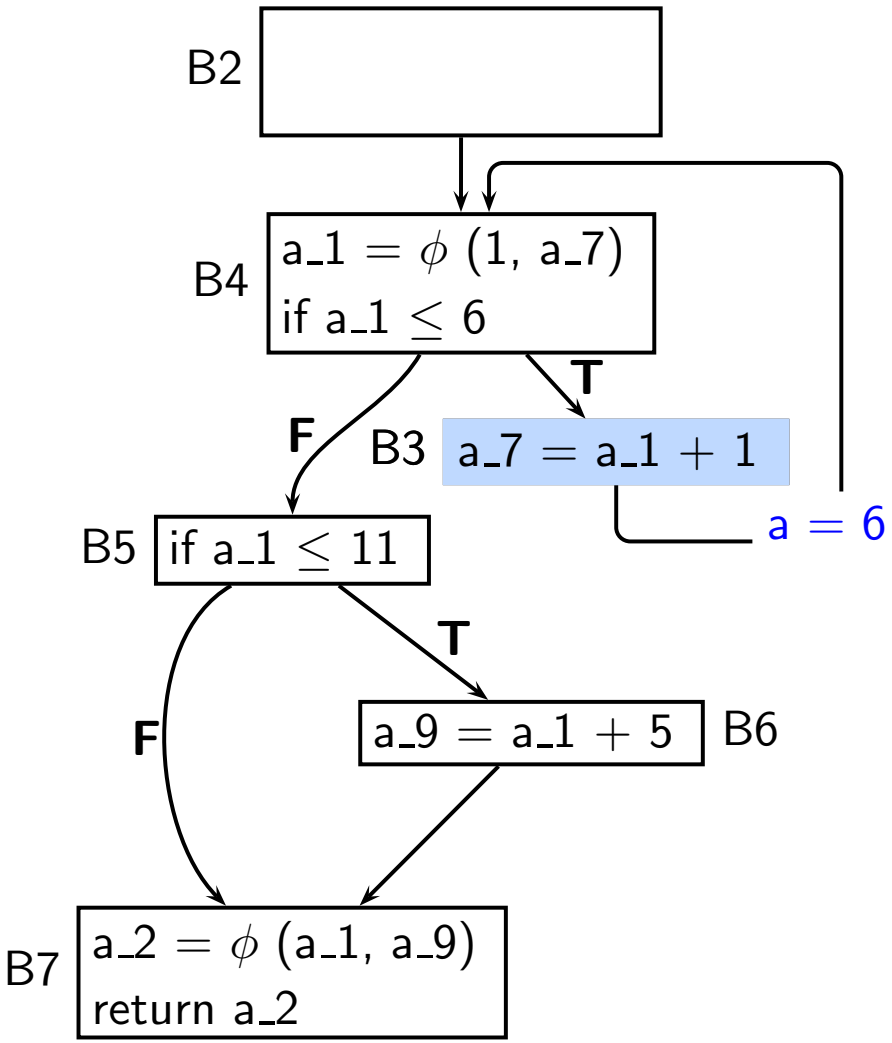
Loop Unrolling



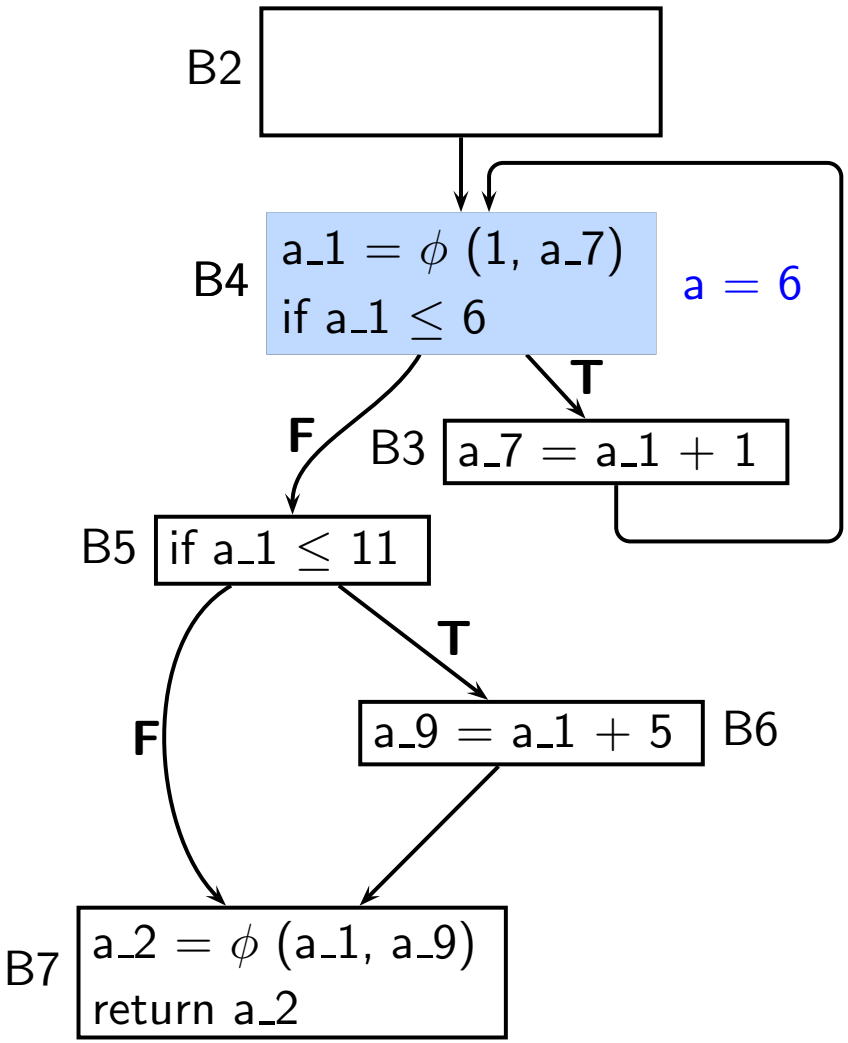
Loop Unrolling



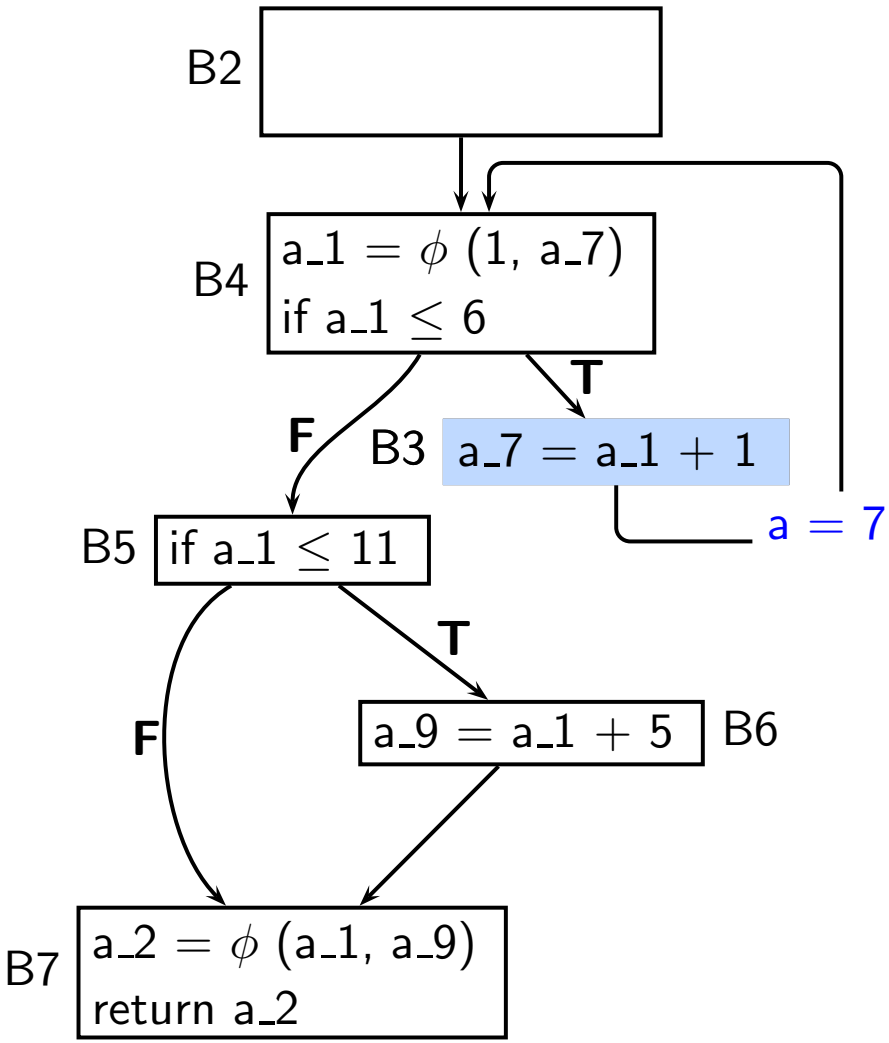
Loop Unrolling



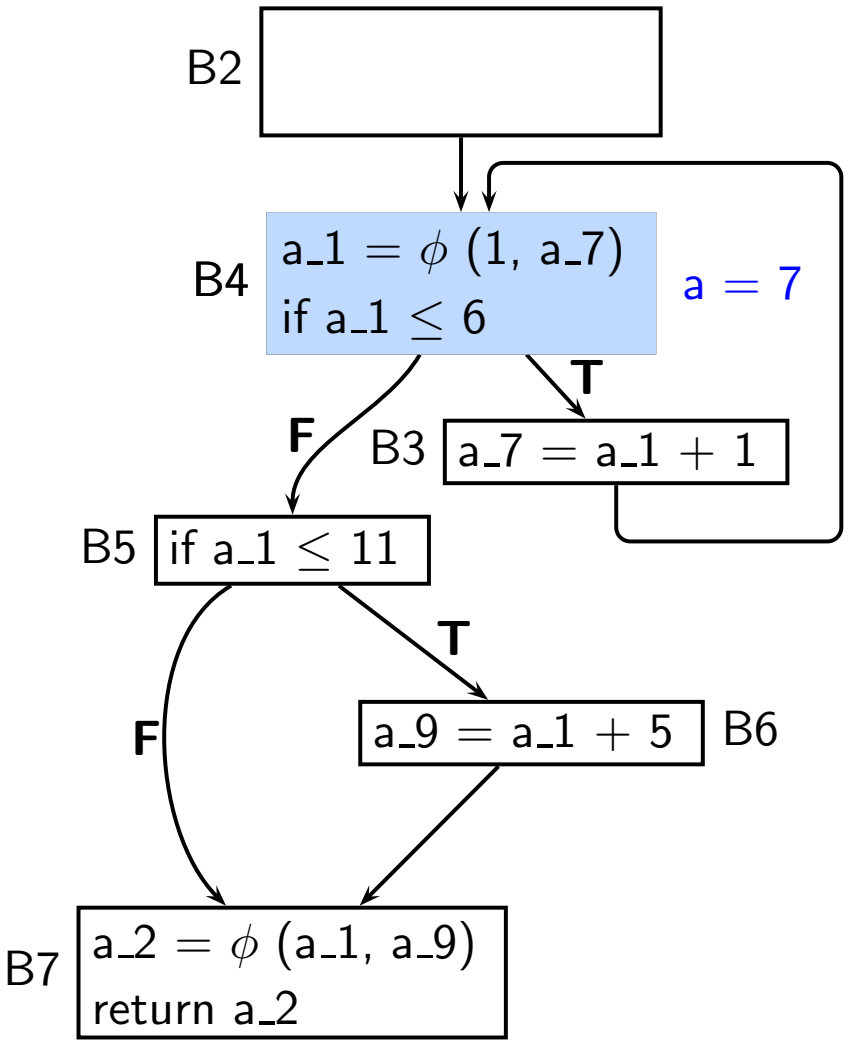
Loop Unrolling



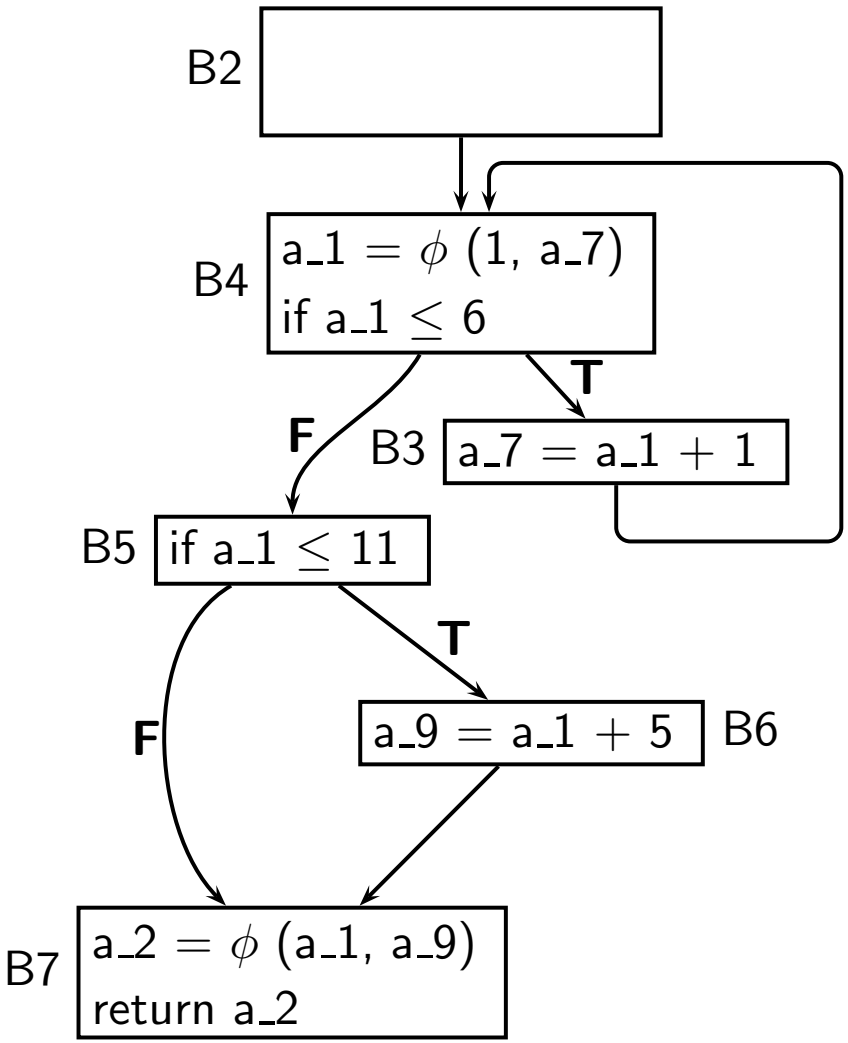
Loop Unrolling

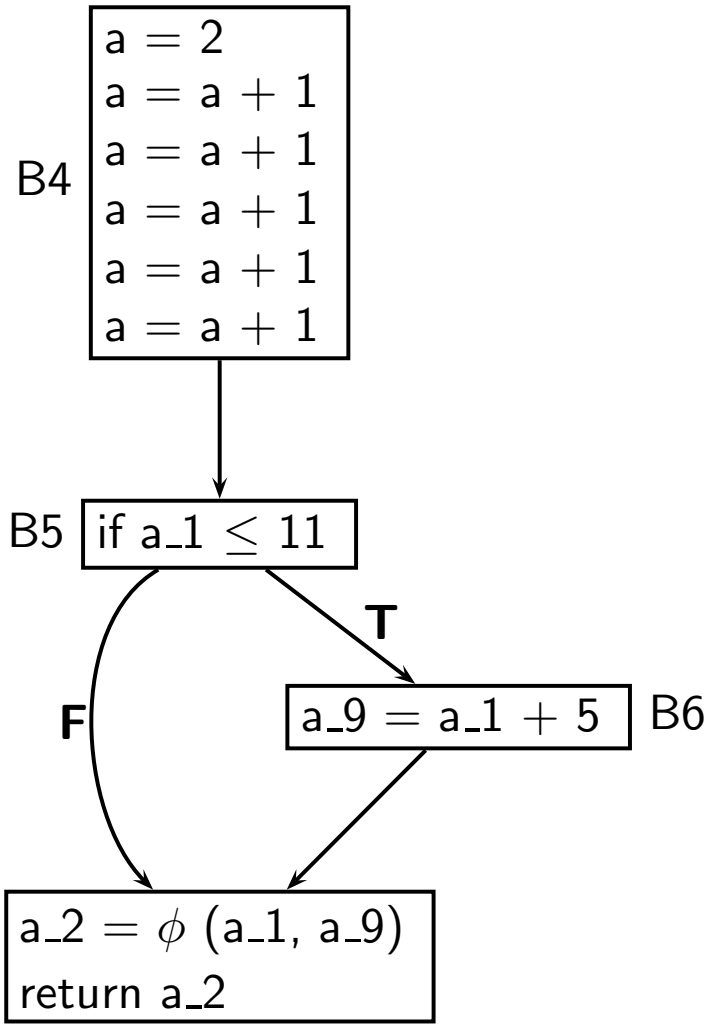


Loop Unrolling



Loop Unrolling

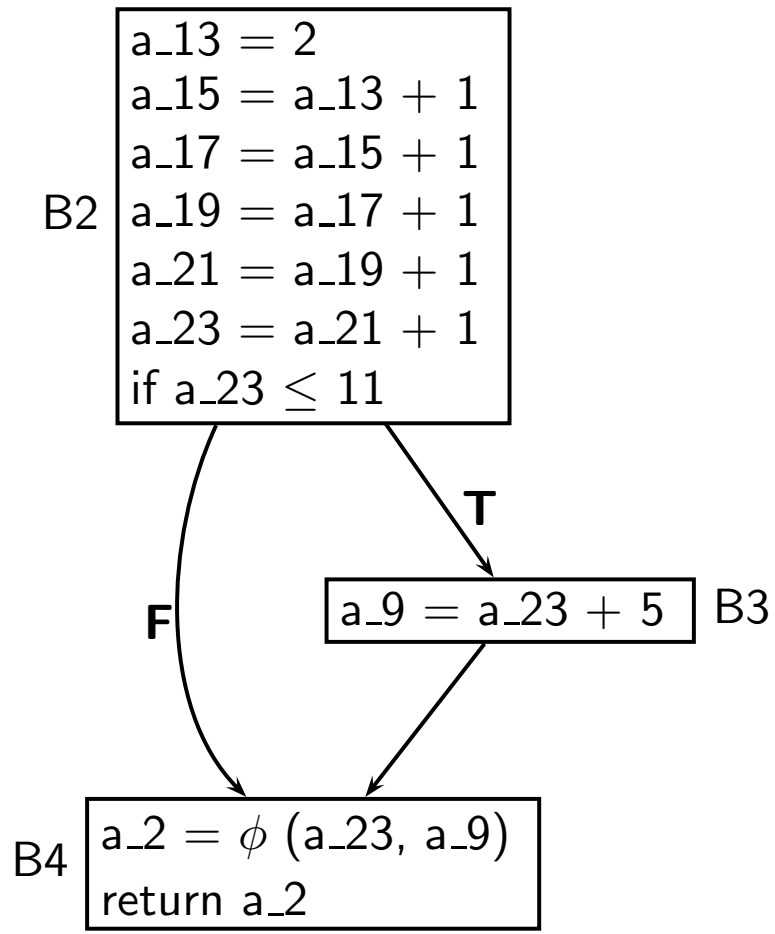




Complete Unrolling of Inner Loops

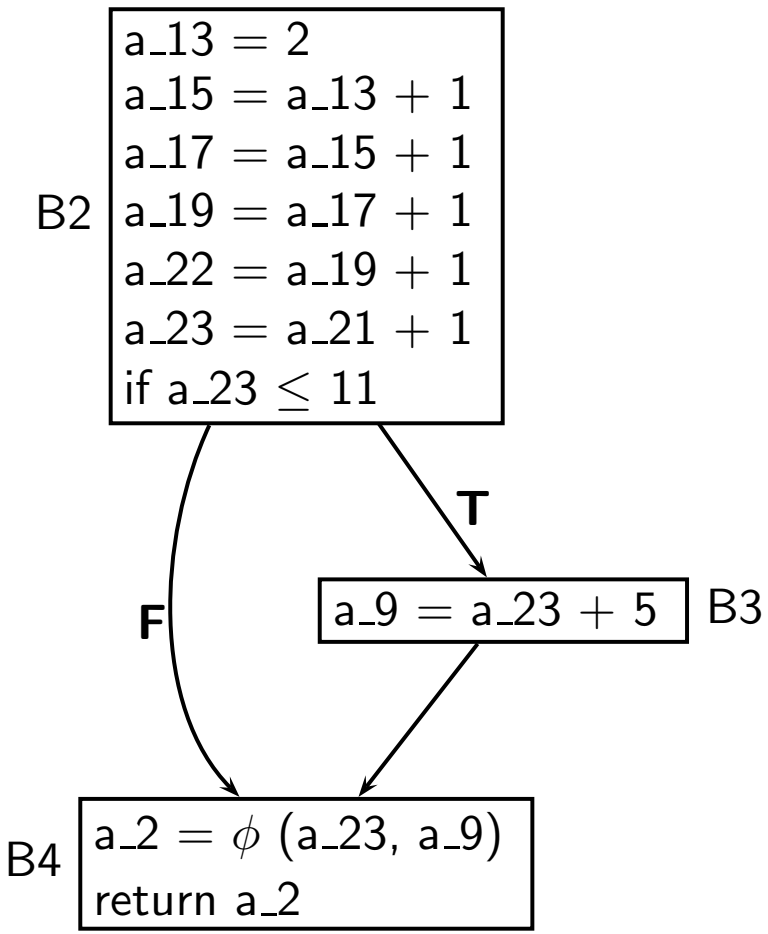
Dump file: ccp.c.059t.cunrolli

```
<bb 2>:
  a_13 = 2;
  a_15 = a_13 + 1;
  a_17 = a_15 + 1;
  a_19 = a_17 + 1;
  a_21 = a_19 + 1;
  a_23 = a_21 + 1;
  if (a_23 <= 11) goto <bb 3>;
  else goto <bb 4>;

<bb 3>:
  a_9 = a_23 + 5;

<bb 4>:
  # a_2 = PHI <a_23(2), a_9(3)>
  return a_2;
```



Another Round of Constant Propagation

Input



Dump file: ccp.c.060t.ccp2

```
main ()
{
  <bb 2>:
    return 12;

}
```

Part 8

Conclusions

Gray Box Probing of GCC: Conclusions

- Source code is transformed into assembly by lowering the abstraction level step by step to bring it close to the machine
- This transformation can be understood to a large extent by observing inputs and output of the different steps in the transformation
- It is easy to prepare interesting test cases and observe the effect of transformations
- One optimization often leads to another

  Hence GCC performs many optimizations repeatedly

  (eg. copy propagation, dead code elimination)