GPU Microarchitecture Note Set 6—Warps and Branch Divergence

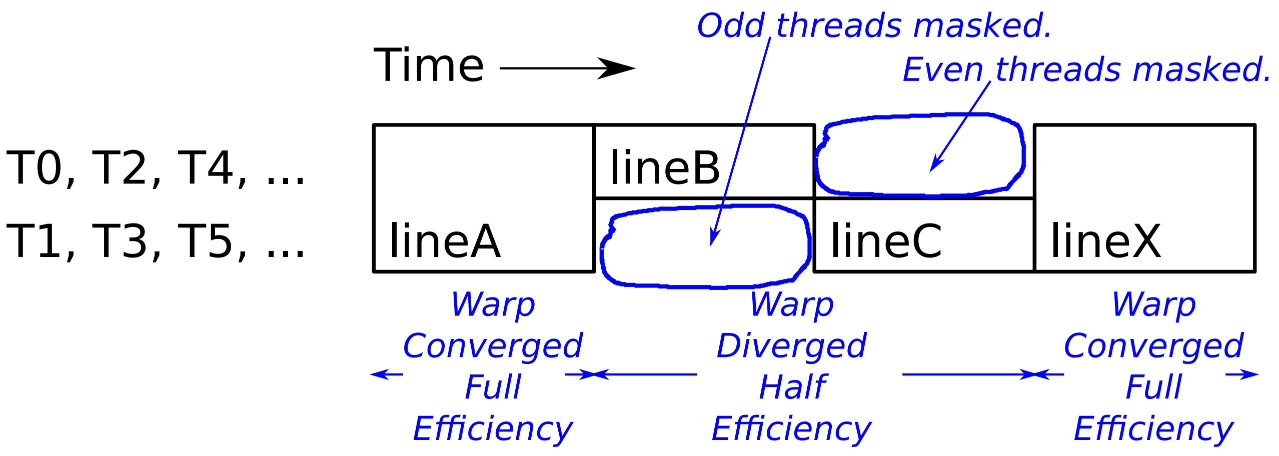
Definition

Branch Divergence:

Effect of execution of a branch where for some threads in a warp the branch is taken, and for other(s) it is not taken.

Can slow down execution by a factor of 32 (for a warp size of 32).

```
lineA: float elt = array[idx];     // All threads execute this insn.
       if ( threadIdx.x & 0x1 )
lineB:   x = elt + 10;             // Executed if threadIdx.x is odd.
       else
lineC:   x = elt - 10;             // Executed if threadIdx.x is even.
lineX: moreStuff;                  // All threads execute this.
```



Background

Warp Review

Group of threads scheduled together as a unit.

A single instruction is fetched for the entire warp.

One set of fetch and decode hardware used for entire warp (32 threads for now).

This reduces hardware cost and energy consumption.

What about branches (and other control transfers)?

Hardware

Path [for a warp]:

A PC and a bit vector. The bit vector indicates which threads are part of the path and the PC is the address of the next instruction to fetch on the path.

Each warp has an *active path* . . .

. . . and zero or more inactive paths.

The inactive paths are kept in a *reconvergence stack*.

The warp scheduler operates on warps' the active paths.

Instructions that Affect Active Path and Reconvergence Stack

Summary

- `SSY`: Pushes item on to reconvergence stack.
- `BRA`: May push item on to reconvergence stack.
- Instructions to pop stack:
  - `SYNC`
  - `BRK`, `PBRK`
  - `foo.S` (`foo` is an ordinary instruction such as FADD, followed by a .S).

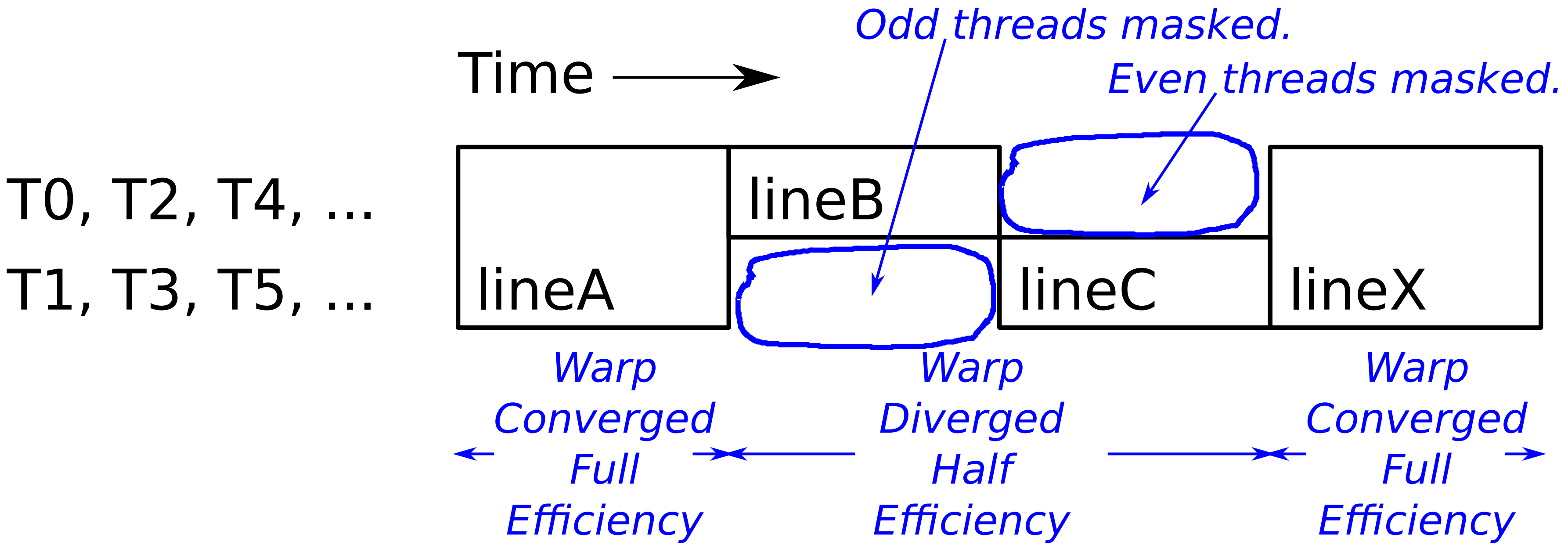
Simple Example

```
 if ( threadIdx.x & 1 ) { r4 = r1 + 10; } else { r4 = r1 - 10 };

     ISETP.NE.AND P1, PT,  R3, RZ  // Set predicate P1.
     SSY ‘(RECONV)                 // Push (RECONV, all thds) on stack.
 @P1 BRA EVEN                      // Push (EVEN, even thds) on stack.
     FADD R4, R1, 10
     SYNC                          // Pop stack, setting PC = EVEN.
EVEN:
     FADD R4, R1, -10
     SYNC                          // Pop stack, setting PC = RECONV.
RECONV:
```

(Note: In code this simple predication would be used.)

Execution Timing Under Divergence



Goal is to minimize time that warps are diverged.

Reconvergence Point

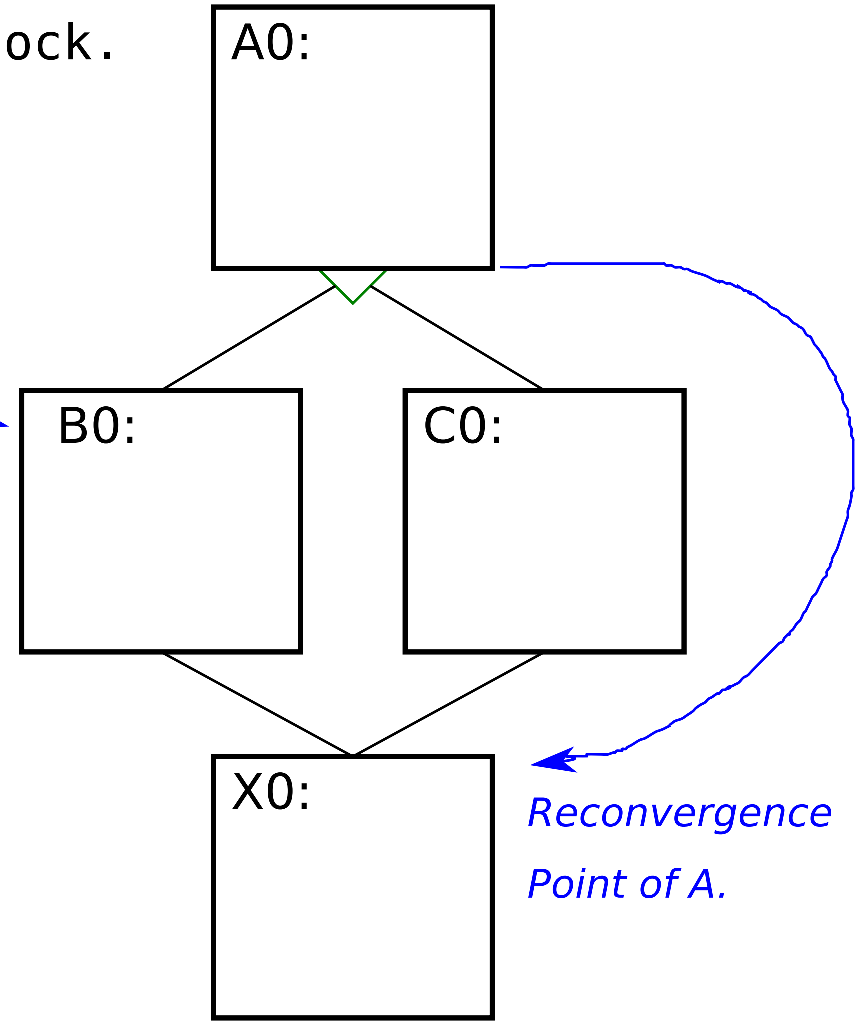
Reconvergence Point [of a branch]:
The closest instruction on all paths starting at the branch. Also known as the *post-dominator* of the branch.

For the prior example `lineX` is the reconvergence point.

Execution of Simple If / Else

Control Flow Diagram

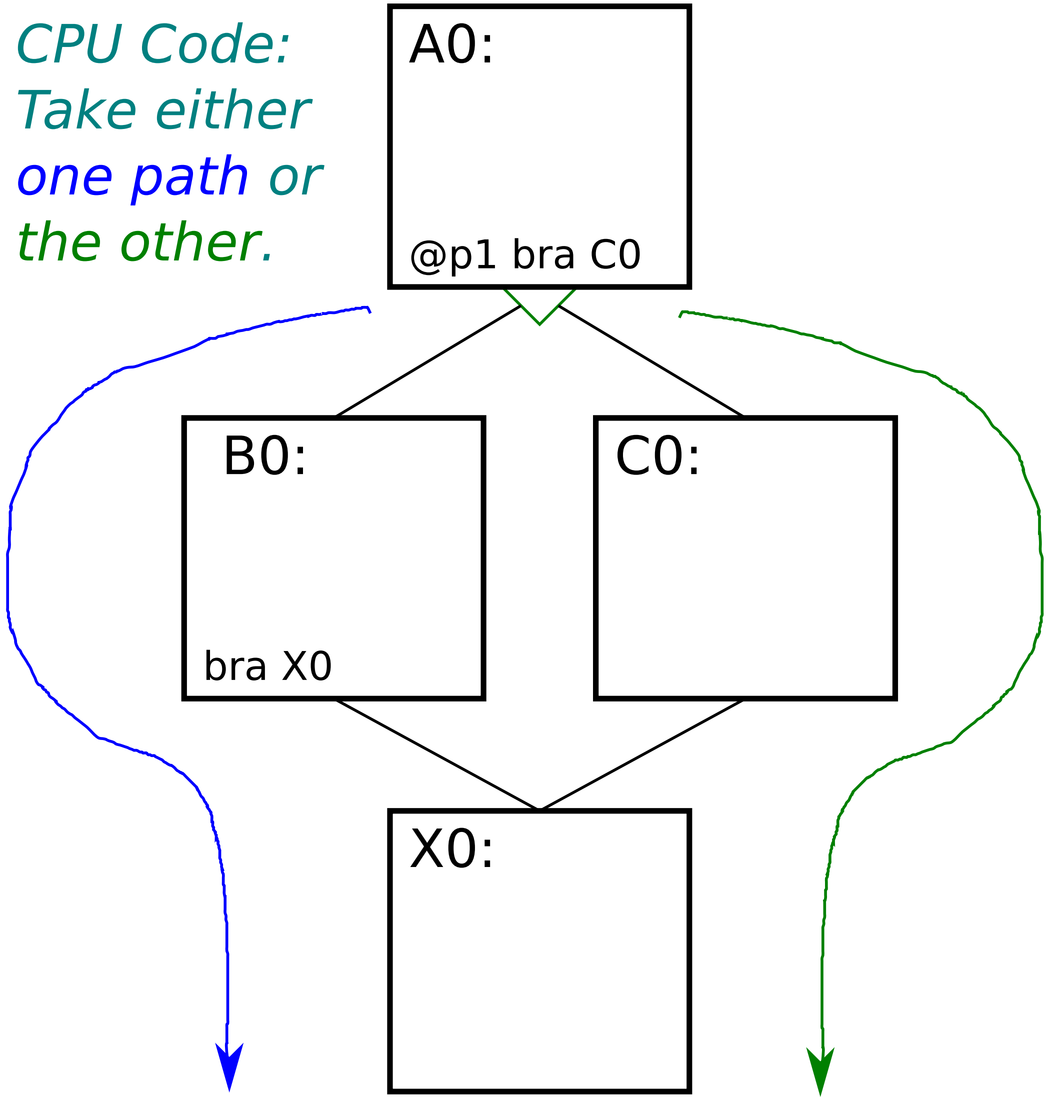
```
A0: // Start of A block.
...
p1 = w > z;
if ( p1 )
{
   B0:
   ...
} else {
  C0:
   ...
}
X0:
...
```



Execution of Simple If / Else

Execution on CPU

```
A0:
...
p1 = w > z;
if ( p1 )
{
   B0:
   ...
} else {
  C0:
   ...
}
X0:
...
```

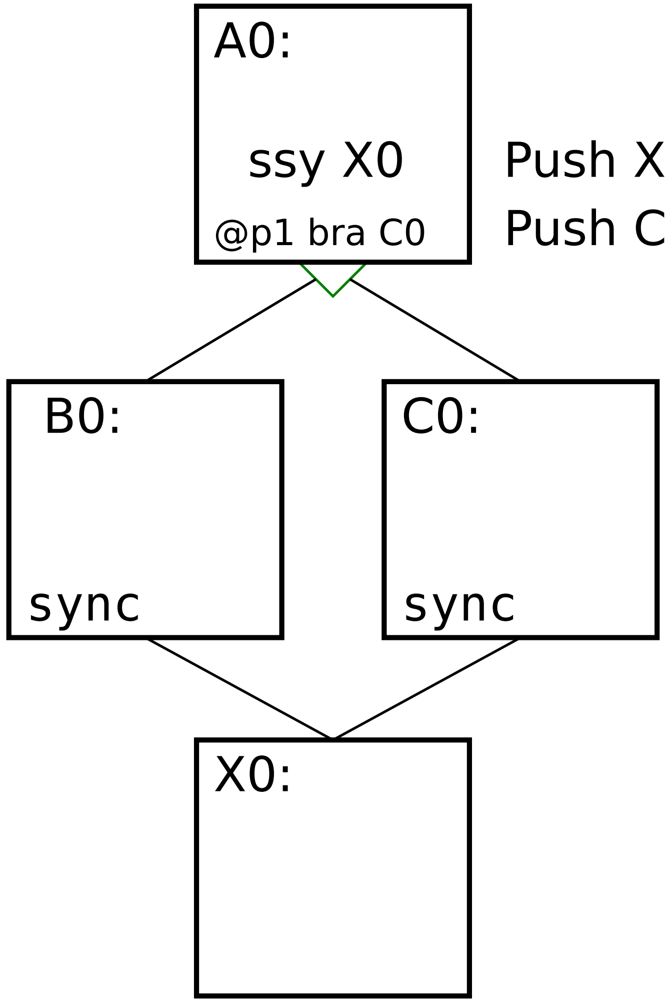


CPU Code: Take either one path or the other.

Execution of Simple If / Else

Execution on GPU

```
A0:
...
p1 = w > z;
if ( p1 )
{
   B0:
   ...
} else {
  C0:
   ...
}
X0:
...
```



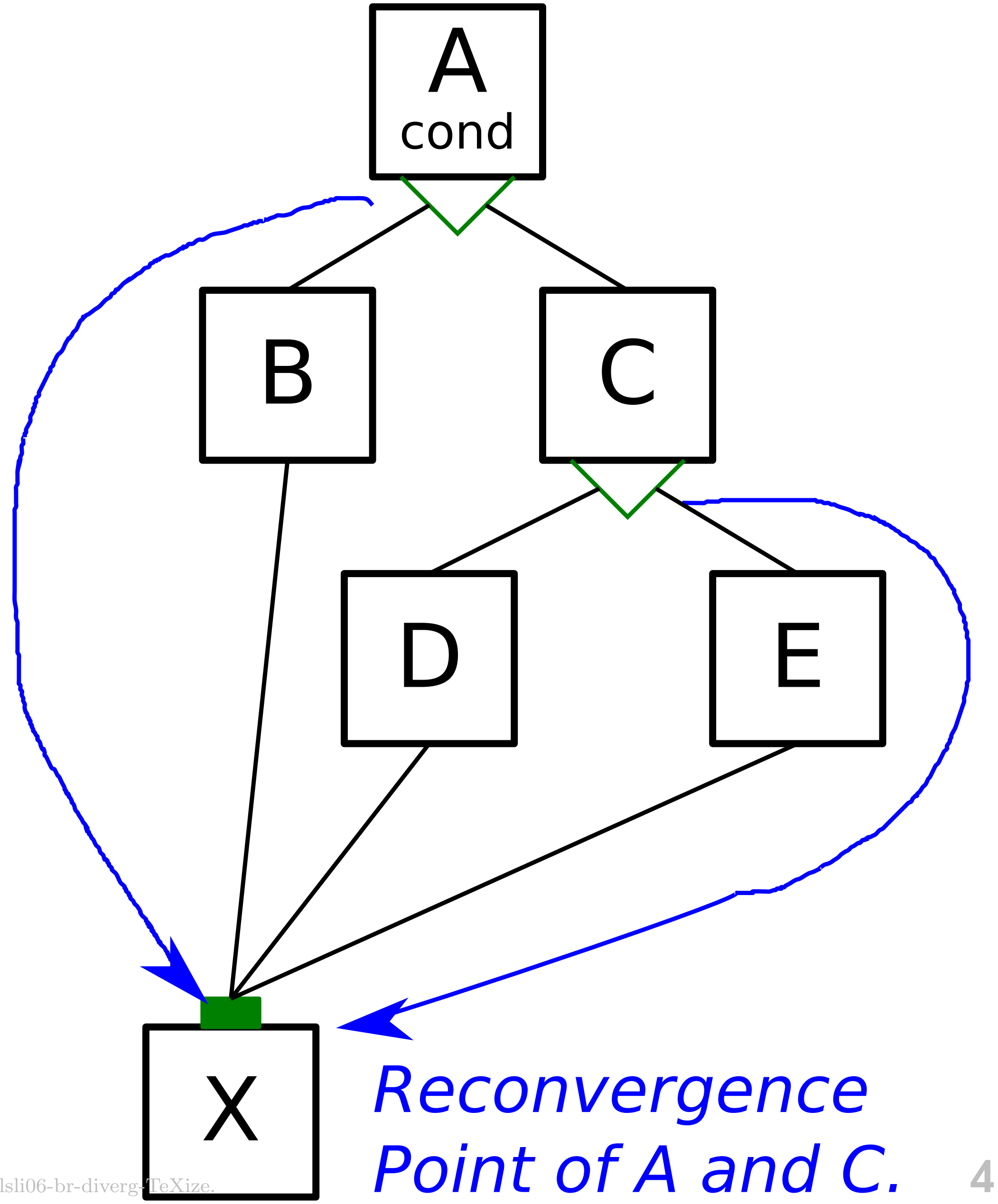
Push X0 on stack.

Push C0 on stack.

Three way if/else.

```
A;
               // -> SSY X:  Push ( X, ACT )
if ( cond )    // -> BRA C:  Push (C, ACT & cond)
{
  B;           // SYNC;
} else {
  C;
  if ( cond2 ) // -> BRA E;  Push (E, ACT & cond2)
    D;         // SYNC;
  else
    E;         // SYNC;
}
X;
```

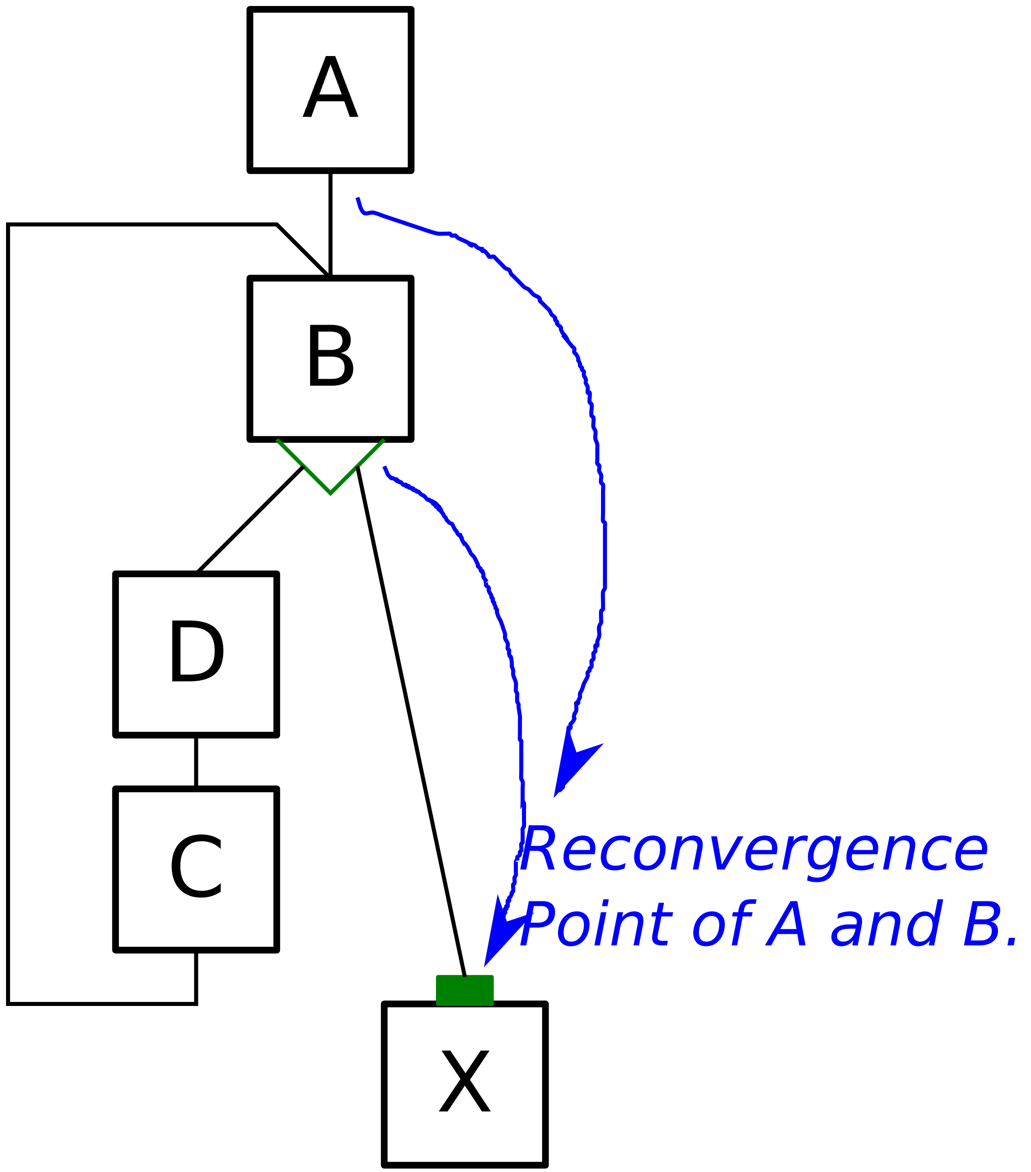
Diagram for case where D and E jump to X.



Reconvergence Point of A and C.

Simple Loop

```
for ( A; B; C ) { D; }  X;
```



EE 7722 Lecture Transparency. Formatted 12:08, 13 April 2020 from lsli06-br-diverg-TeXize.

```
for ( A; B; C ) { if ( cond ) D; else E; }  X
```

```
for ( A; B; C ) { D; if ( cond ) E; else F; }  X
```

```
for ( A; B; C ) {
  D;
  if ( cond1 )
   { E; }
  else
   {
     F;
     if ( cond2 ) { G; break; }
   }
 }
 X;
```

Favorable Cases: No `goto`s, `break`s, `return`s.

Handling Warp Divergence

Method used based on nature of code.

Predication

Predication and *undiverged branch* instruction.

Branches and synchronization points.

Implementation of if/else.

Use predication when warp diverged, use branches when warp converged.

```
        /*0090*/                   FSETP.LT.AND P0, PT, R7, 0.5, PT;
        /*0098*/              @!P0 BRA.U ‘(.L_4);
        /*00a0*/               @P0 LD.E R10, [R2+0x4];
        /*00a8*/               @P0 FFMA R12, R7, c[0x3][0x4], RZ;
        /*00b0*/               @P0 LD.E R11, [R2+0x8];
        /*00b8*/               @P0 LD.E R7, [R2+0xc];
        /*00c8*/               @P0 FFMA R10, R10, c[0x3][0x8], R12;
        /*00d0*/               @P0 FFMA R10, R11, c[0x3][0xc], R10;
        /*00d8*/               @P0 FFMA R7, R7, c[0x3][0x10], R10;
        /*00e0*/               @P0 BRA.U ‘(.L_5);
.L_4:
        /*00e8*/              @!P0 LD.E R11, [R2+0x4];
        /*00f0*/              @!P0 FADD R12, R7, c[0x3][0x4];
        /*00f8*/              @!P0 LD.E R10, [R2+0x8];
        /*0108*/              @!P0 F2F.F32.F32 R12, R12;
        /*0110*/              @!P0 LD.E R7, [R2+0xc];
        /*0118*/              @!P0 FADD R11, R11, c[0x3][0x8];
        /*0120*/              @!P0 FADD R13, R10, c[0x3][0xc];
        /*0128*/              @!P0 FADD R10, R12, R11;
        /*0130*/              @!P0 FADD R11, R7, c[0x3][0x10];
        /*0138*/              @!P0 FADD R7, R10, R13;
        /*0148*/              @!P0 FADD R7, R7, R11;
.L_5:
        /*0150*/                   IADD R10.CC, R8, c[0x0][0x140];
        /*0158*/                   IADD.X R11, R9, c[0x0][0x144];
```