Improving debug variable location coverage by using even more SSA

Variable location mappings

```
int ext();
extern int xyzzy;

int foo(int bar, int baz) {
  int qux = bar + baz;
  qux *= ext();
  qux -= xyzzy;
  return qux;
}
```

```
<foo>:
   push   %rbx
   mov    %edi,%ebx
   add    %esi,%ebx
   xor    %eax,%eax
   callq  c <foo+0xc>
   imul   %ebx,%eax
   sub    0x0(%rip),%eax
   pop    %rbx
   retq
```

bar	baz	qux
edi	esi	
edi	esi	
ebx	esi	
	esi	ebx
	esi	ebx
		ebx
		eax
		eax

Variable locations map between variable names and registers

dbg.value intrinsics

```
define dso_local i32 @foo(i32 %bar, i32 %baz) #0 !dbg !7 {
entry:
  call void @llvm.dbg.value(metadata i32 %bar, metadata !12, metadata !DIExpression()), !dbg !15
  call void @llvm.dbg.value(metadata i32 %baz, metadata !13, metadata !DIExpression()), !dbg !15
  %add = add nsw i32 %bar, %baz, !dbg !16
  call void @llvm.dbg.value(metadata i32 %add, metadata !14, metadata !DIExpression()), !dbg !15
  %call = call i32 (...) @ext(), !dbg !17
  %mul = mul nsw i32 %add, %call, !dbg !18
  call void @llvm.dbg.value(metadata i32 %mul, metadata !14, metadata !DIExpression()), !dbg !15
  %0 = load i32, i32* @xyzzy, align 4, !dbg !19, !tbaa !20
  %sub = sub nsw i32 %mul, %0, !dbg !24
  call void @llvm.dbg.value(metadata i32 %sub, metadata !14, metadata !DIExpression()), !dbg !15
  ret i32 %sub, !dbg !25
}
```

DBG_VALUE instructions

```
bb.0.entry:
  liveins: $edi, $esi
  %1:gr32 = COPY $esi
  DBG_VALUE %1:gr32, $noreg, !"baz", !DIExpression()
  %0:gr32 = COPY $edi
  DBG_VALUE %0:gr32, $noreg, !"bar", !DIExpression()
  %2:gr32 = nsw ADD32rr %1:gr32(tied-def 0), %0:gr32, implicit-def dead $eflags
  DBG_VALUE %2:gr32, $noreg, !"qux", !DIExpression()
  %3:gr32 = MOV32r0 implicit-def dead $eflags
  %4:gr8 = COPY %3.sub_8bit:gr32
  $al = COPY %4:gr8
  CALL64pcrel32 @ext, <regmask>, [Many args]
  %5:gr32 = COPY $eax
  %6:gr32 = nsw IMUL32rr %5:gr32(tied-def 0), killed %2:gr32, implicit-def dead $eflags
  DBG_VALUE %6:gr32, $noreg, !"qux", !DIExpression()
  %7:gr32 = nsw SUB32rm %6:gr32(tied-def 0), $rip, 1, $noreg, @xyzzy, [...]
  DBG_VALUE %7:gr32, $noreg, !"qux", !DIExpression()
  $eax = COPY %7:gr32
  RET 0, $eax
```

Register coalescing -- example

```
%0:gr32 = COPY killed $edi
%2:gr32 = COPY killed %0:gr32
%2:gr32 = nsw ADD32rr killed %2:gr32(tied-def 0), killed %1:gr32
DBG_VALUE %0:gr32, $noreg, !"baz", !DIExpression()
```

Register coalescing -- example

```
%2:gr32 = COPY killed $edi
%2:gr32 = COPY killed %0:gr32
%2:gr32 = nsw ADD32rr killed %2:gr32(tied-def 0), killed %1:gr32
DBG_VALUE $noreg, $noreg, !"baz", !DIExpression()
```

Live-range records

```
%0:gr32 = COPY killed $edi

%2:gr32 = COPY killed %0:gr32

%2:gr32 = nsw ADD32rr killed %2:gr32(tied-def 0), killed %1:gr32

DBG_VALUE %0:gr32, $noreg, !"baz", !DIExpression()
```

%0 live range

%2 live range

Live-range with control flow

```
  %0:gr32 = COPY killed $edi


  %2:gr32 = COPY killed %0:gr32
  JCC %bb.2
bb.1:
  %2:gr32 = nsw ADD32rr killed %2:gr32(tied-def 0), killed %1:gr32
  JMP %bb.3
bb.2:
  DBG_VALUE %0:gr32, $noreg, !"baz", !DIExpression()
```


%0 live range

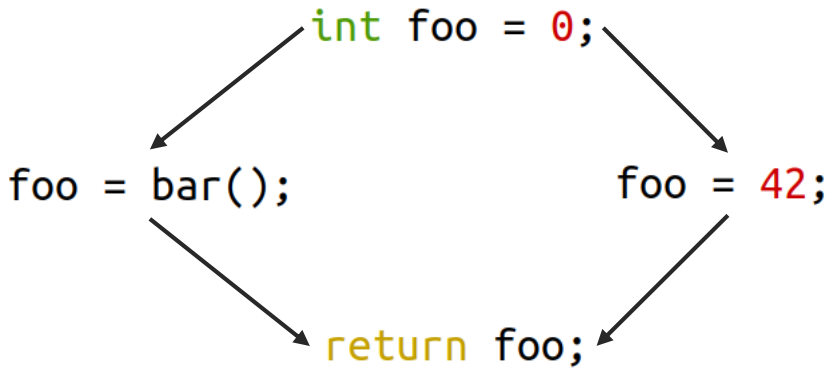
%2 live range

C SSA comparison

```
int ext();
extern int xyzzy;

int foo(int bar, int baz) {
  int qux = bar + baz;
  qux *= ext();
  qux -= xyzzy;
  return qux;
}
```

- `%0:gr32 = COPY killed $edi`
- `%2:gr32 = COPY killed %0:gr32`
- `%2:gr32 = nsw ADD32rr killed %2:gr32(tied-def 0), killed %1:gr32`
- `DBG_VALUE %0:gr32, $noreg, !"baz", !DIExpression()`

Instruction referencing




```
bb.0.entry:
  %0 = MOV32ri 0
  DBG_VALUE %0
  JCC_1 %bb.2, 4
bb.1.bb1:
  CALL64pcrel32 @bar
  %1 = COPY $rax
  DBG_VALUE %1
  JMP_1 %bb.3
bb.2.bb2:
  %2 = MOV32ri 42
  DBG_VALUE %2
bb.3.bb3:
  %3 = PHI [%1, %2]
  RETQ %3
```

```
bb.0.entry:
  %0 = MOV32ri 0, debug-instr-num 1
  DBG_INSTR_REF 1, 0
  JCC_1 %bb.2, 4
bb.1.bb1:
  CALL64pcrel32 @bar, debug-instr-num 2
  %1 = COPY $rax
  DBG_INSTR_REF 2, 4
  JMP_1 %bb.3
bb.2.bb2:
  %2 = MOV32ri 42, debug-instr-num 3
  DBG_INSTR_REF 3, 0
bb.3.bb3:
  %3 = PHI [%1, %2]
  RETQ %3
```

Optimisations must be recorded

```
bb.2.bb2:
  %2 = MOV32ri 0, debug-instr-num 3
  DBG_INSTR_REF 3, 0
```

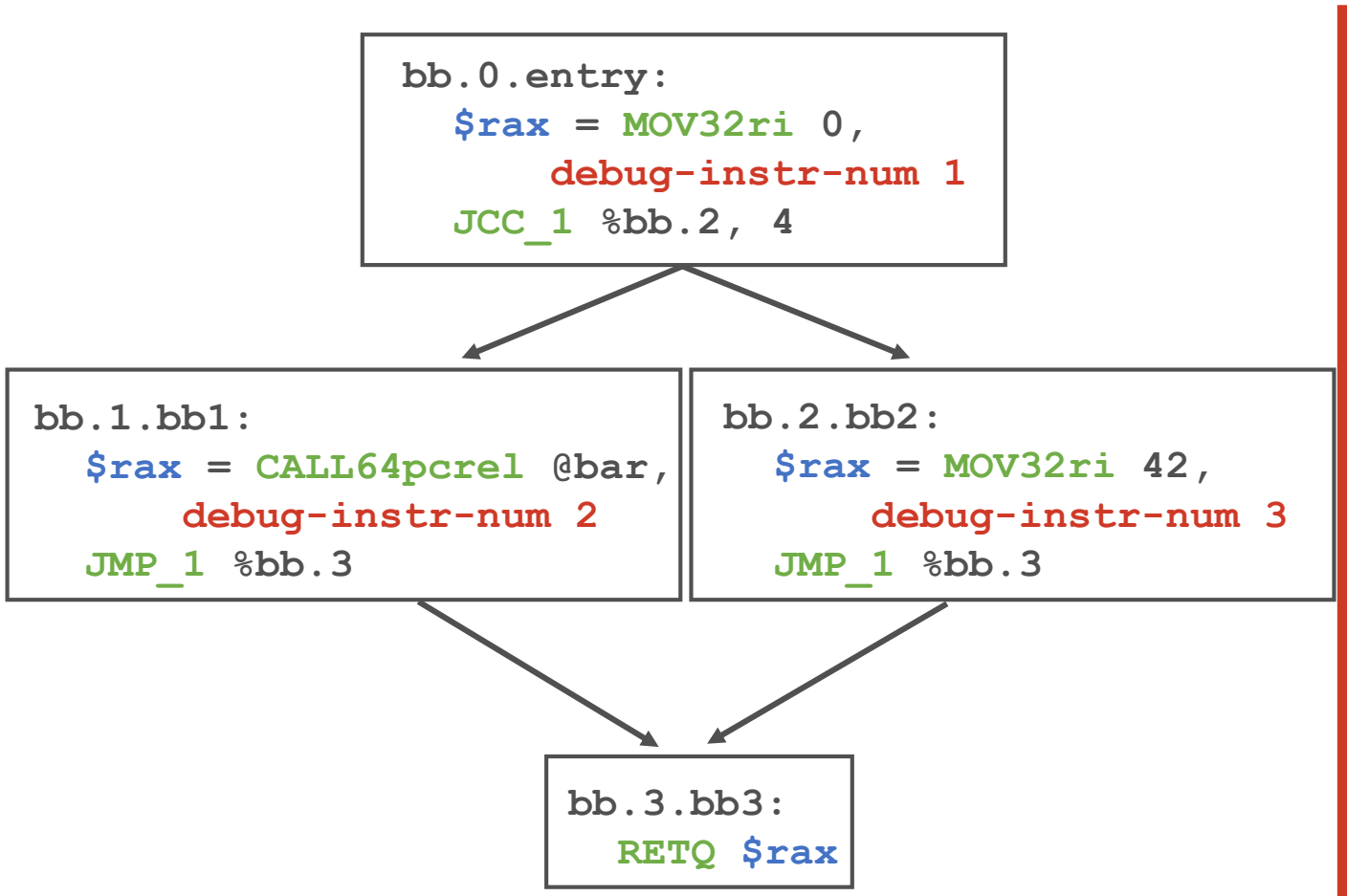
```
bb.2.bb2:
  $rax, %2 = MOV32ri 42, 0, debug-instr-num 4
  DBG_INSTR_REF 3, 0

debugValueSubstitutions:
 - { srcinst: 3, srcop: 0, dstinst: 4, dstop: 1, subreg: 0}
```

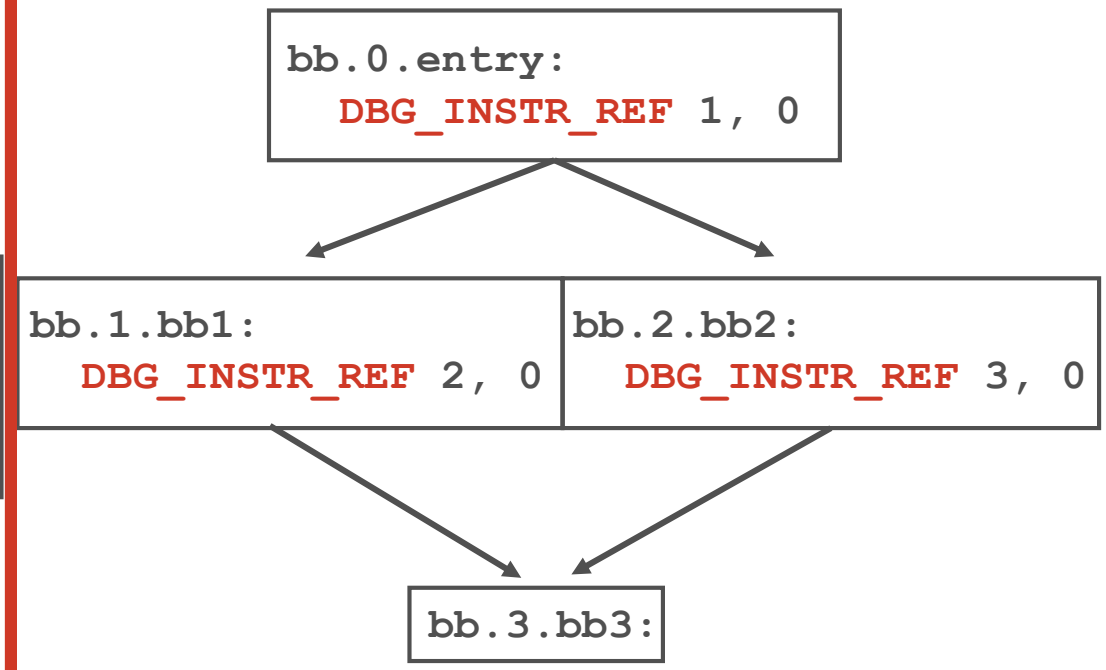
PHIs are and aren't instructions

```
bb.3.bb3:
  %3 = PHI [%1, %2], debug-instr-num 4
  DBG_INSTR_REF 4, 0
  RETQ %3
```

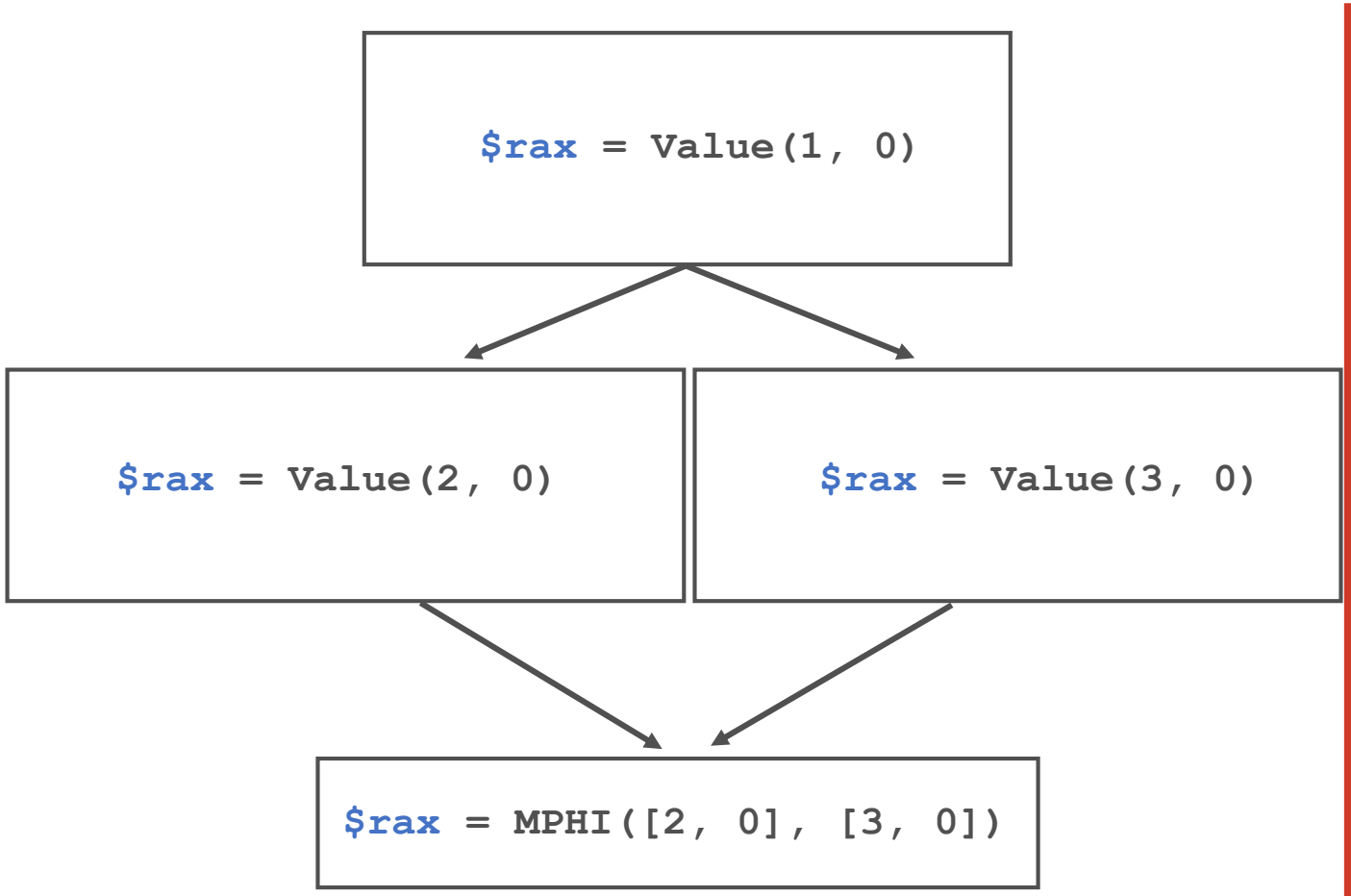
```
bb.3.bb3:
  DBG_PHI $eax, 4
  DBG_INSTR_REF 4, 0
  RETQ $eax
```

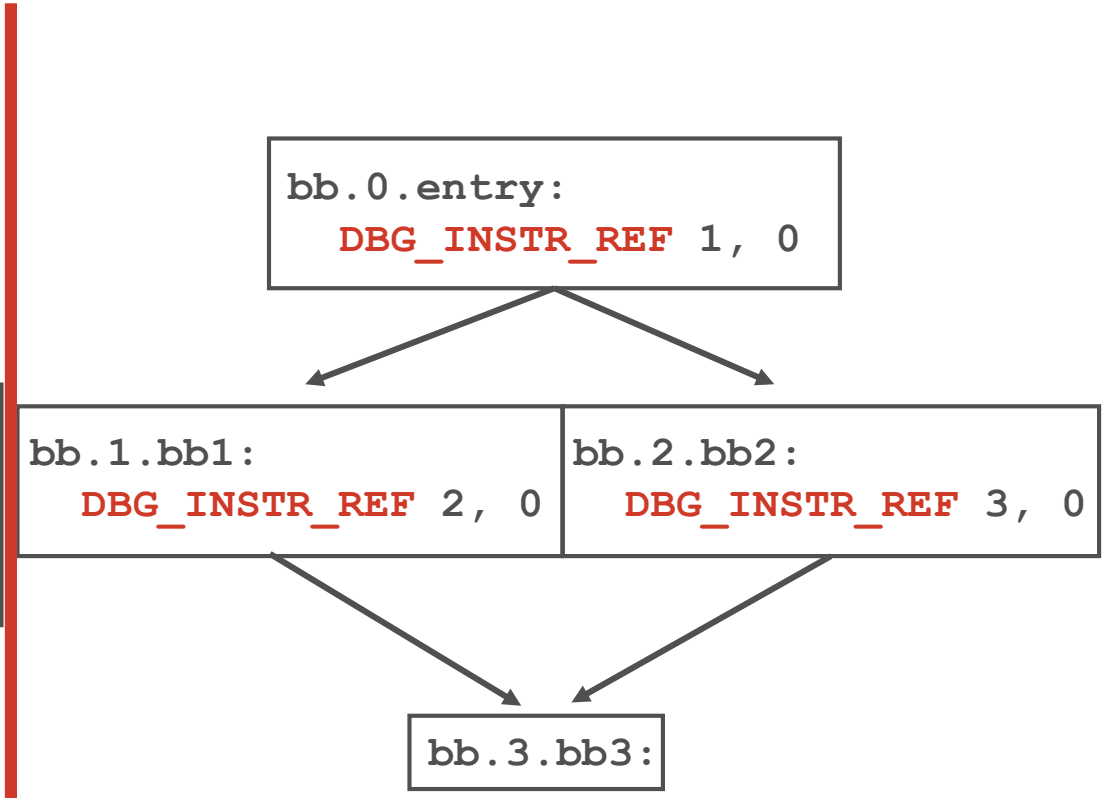
Worked example for SSA construction



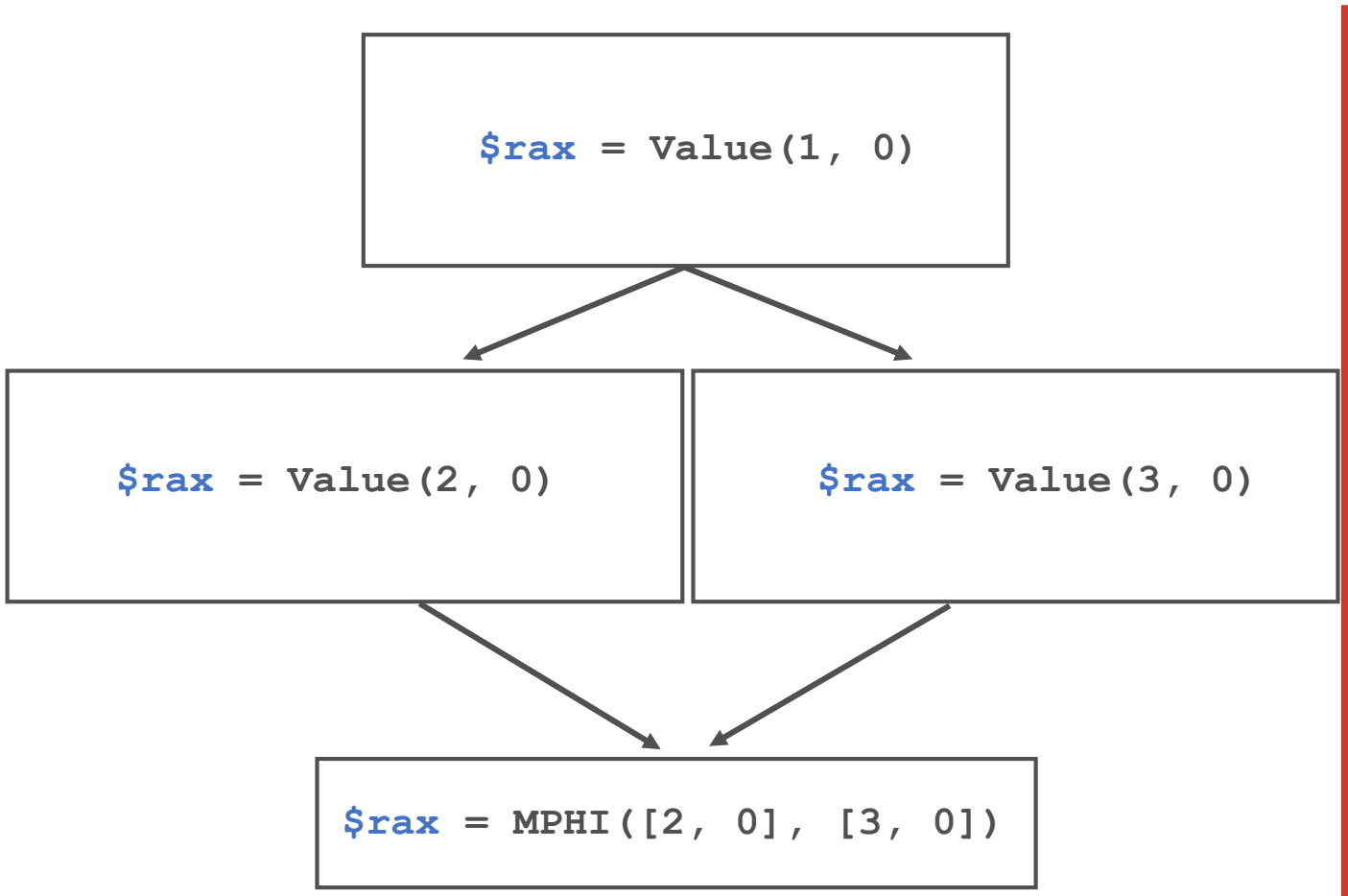


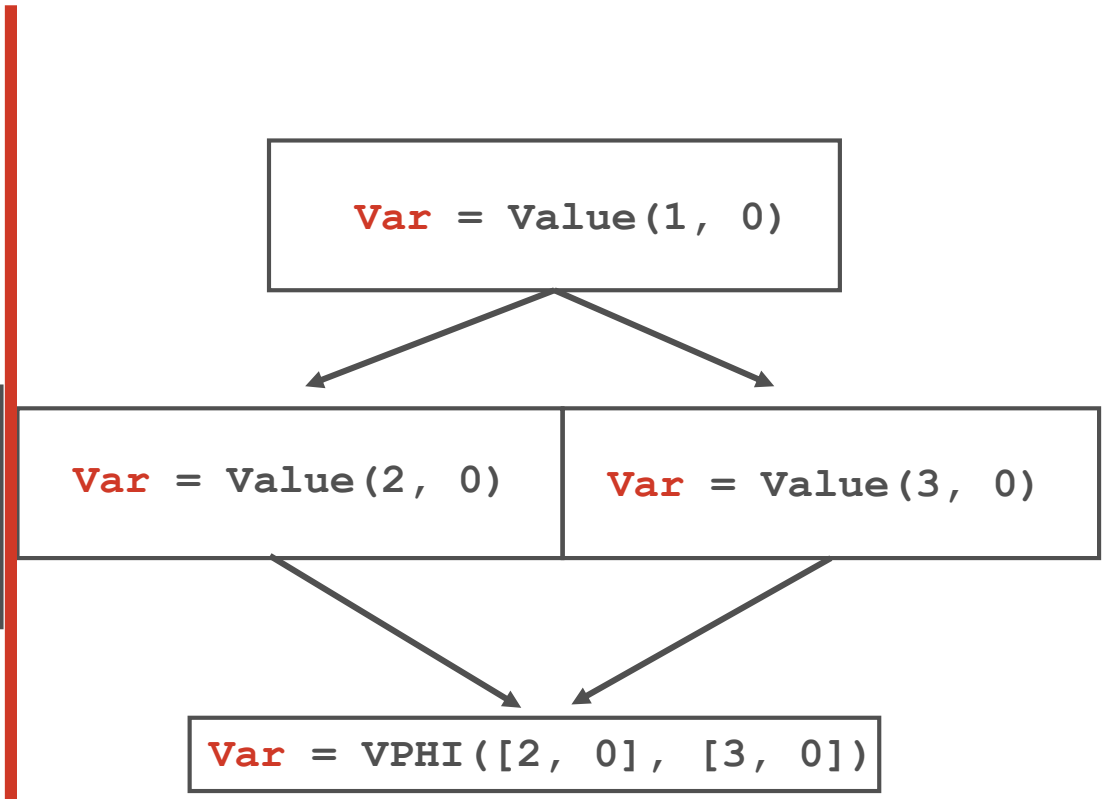
Worked example for SSA construction





Worked example for SSA construction





Results

	Gain in total variables	Average coverage Normal locations	Average coverage Instr-ref locations
Game A	+3.3%	78%	81%
Game B	+6.4%	75%	79%
Clang	+1.3%	89%	90%

Gain: increase in absolute number of variables with DW_AT_location attributes
Average coverage: average of % bytes covered for all variables