Instruction Scheduling in LLVM

Hsiangkai Wang
Hsiangkai@gmail.com
Andes Technology

Agenda

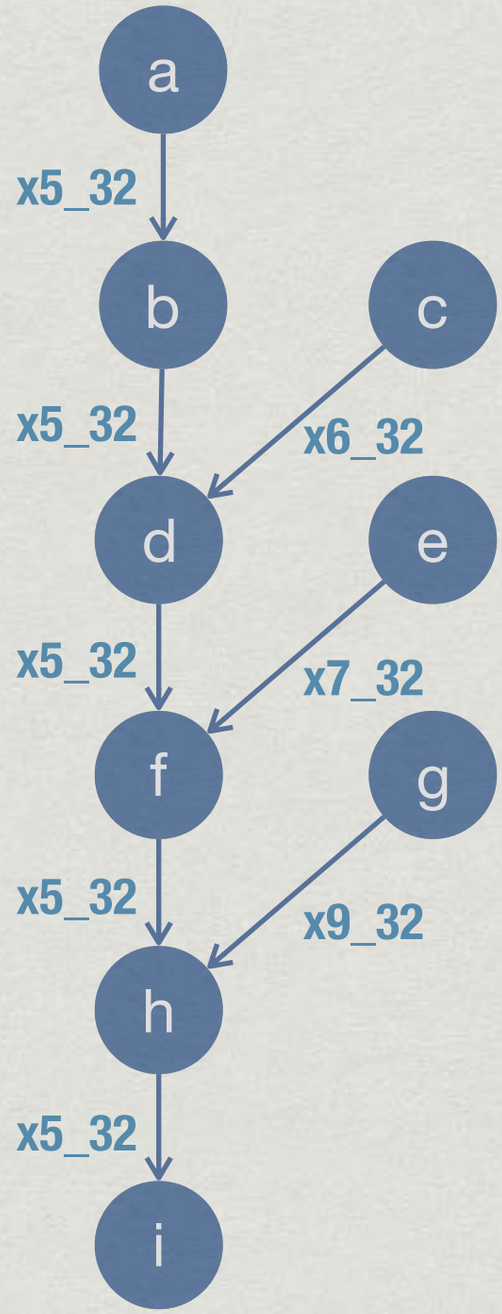
- Introduction to Instruction Scheduling
- Scheduler in LLVM
- Pipeline Modeling
- Scheduler Customization

Instruction Scheduling

- Different operations take different lengths of time.
- Instruction scheduling is the process reordering the operations in an attempt to decrease its running time.

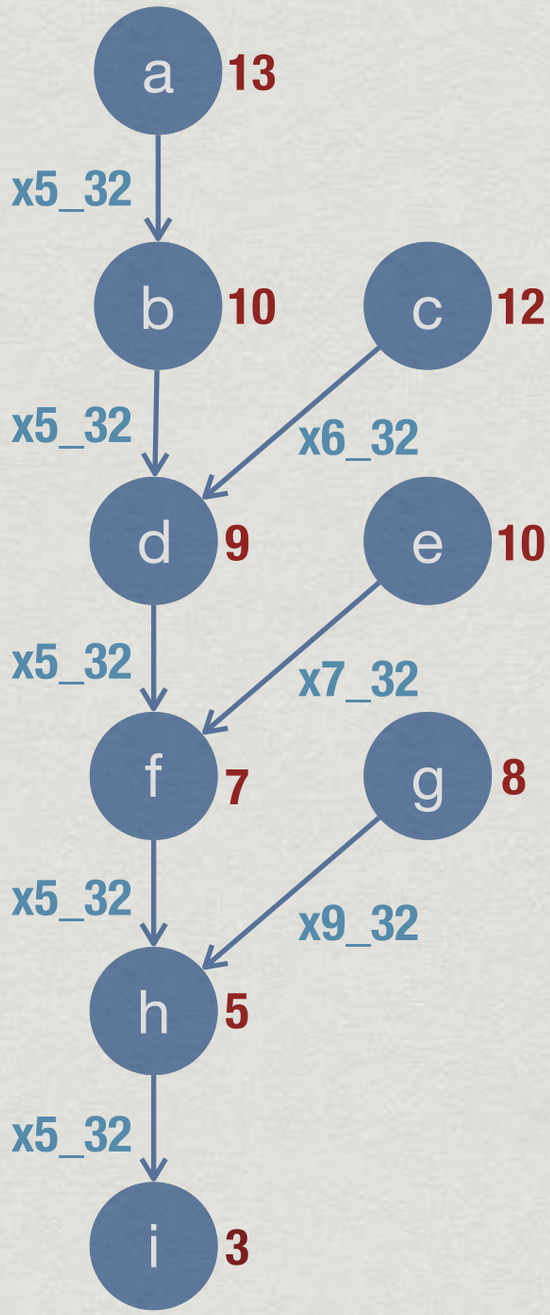
```
a( 1): load $x5_32, $x8_32, @a
b( 4): add $x5_32, $x5_32, $x5_32
c( 5): load $x6_32, $x8_32, @b
d( 8): mul $x5_32, $x5_32, $x6_32
e(10): load $x7_32, $x8_32, @c
f(13): mul $x5_32, $x5_32, $x7_32
g(15): load $x9_32, $x8_32, @d
h(18): mul $x5_32, $x5_32, $x9_32
i(20): store $x5_32, $x8_32, @a
```

```
load, store: 3
add: 1
mul: 2
```



```
a( 1): load $x5_32, $x8_32, @a
b( 4): add $x5_32, $x5_32, $x5_32
c( 5): load $x6_32, $x8_32, @b
d( 8): mul $x5_32, $x5_32, $x6_32
e(10): load $x7_32, $x8_32, @c
f(13): mul $x5_32, $x5_32, $x7_32
g(15): load $x9_32, $x8_32, @d
h(18): mul $x5_32, $x5_32, $x9_32
i(20): store $x5_32, $x8_32, @a
```

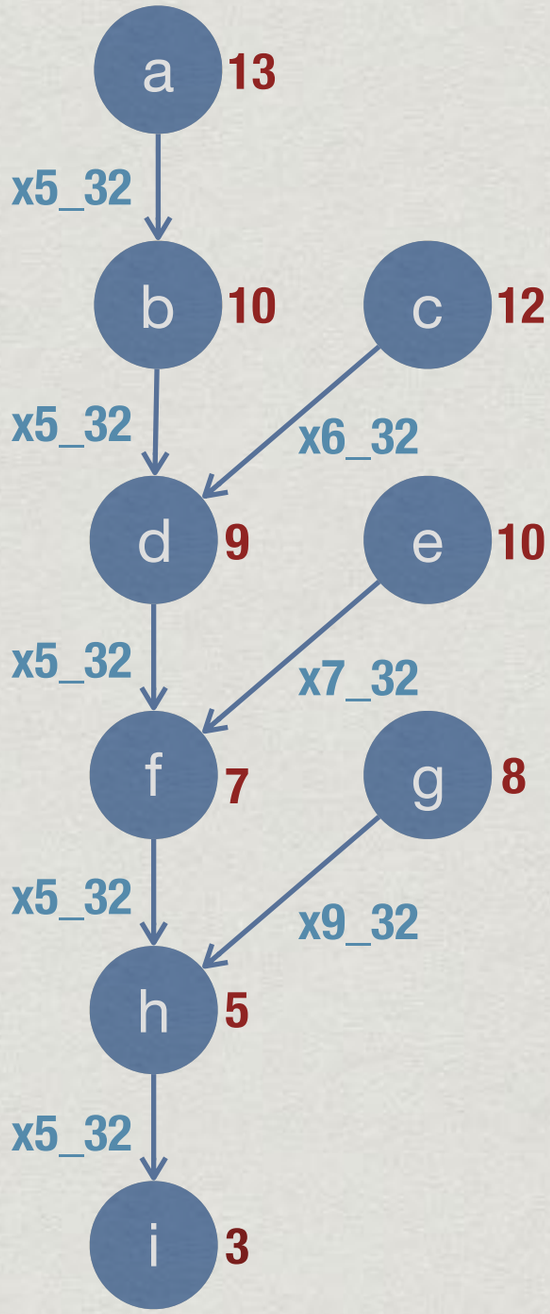
```
load, store: 3
add: 1
mul: 2
```

```
a( 1): load $x5_32, $x8_32, @a
b( 4): add $x5_32, $x5_32, $x5_32
c( 5): load $x6_32, $x8_32, @b
d( 8): mul $x5_32, $x5_32, $x6_32
e(10): load $x7_32, $x8_32, @c
f(13): mul $x5_32, $x5_32, $x7_32
g(15): load $x9_32, $x8_32, @d
h(18): mul $x5_32, $x5_32, $x9_32
i(20): store $x5_32, $x8_32, @a
```

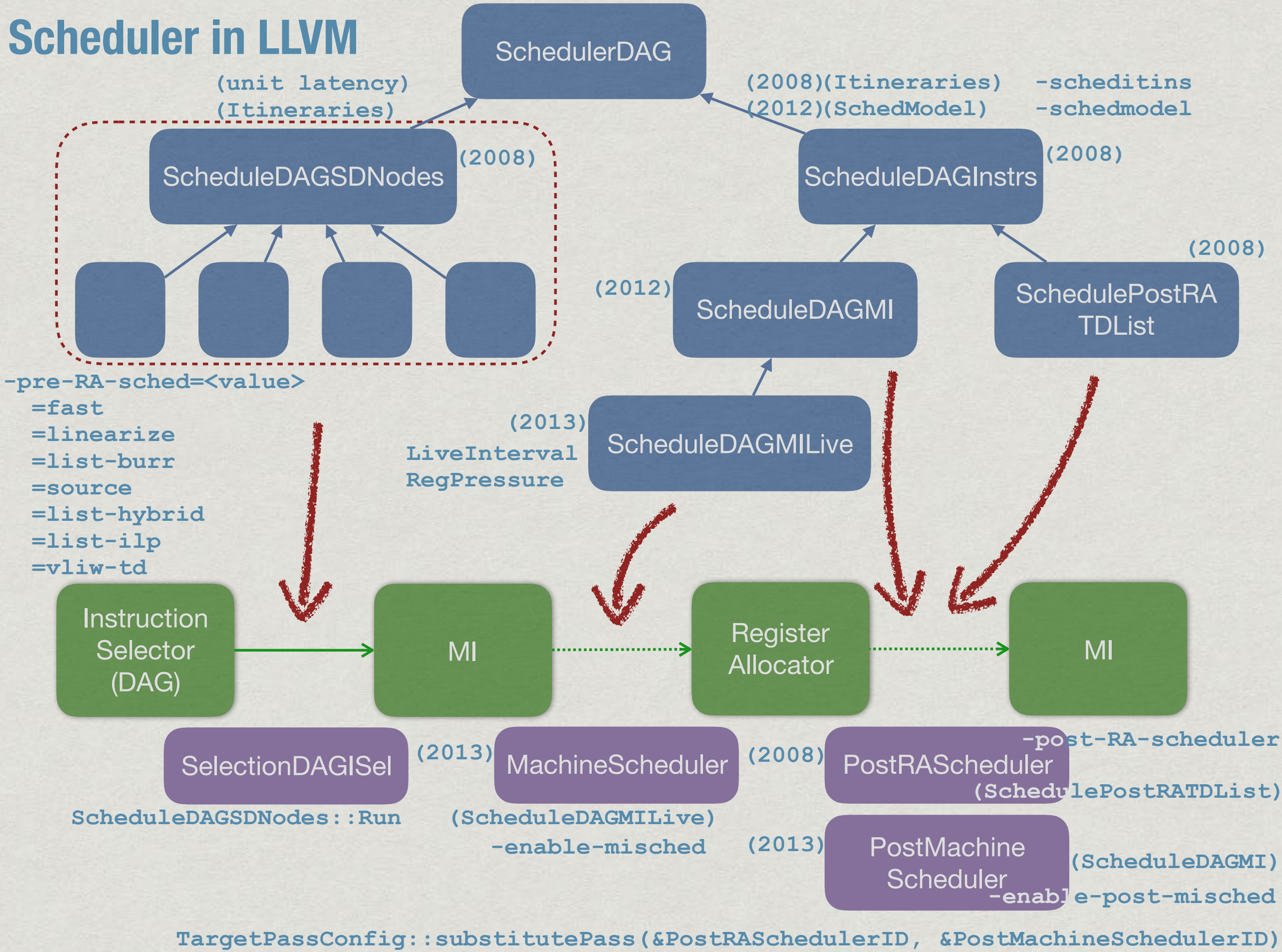
```
load, store: 3
add: 1
mul: 2
```

a c e b d g f h i



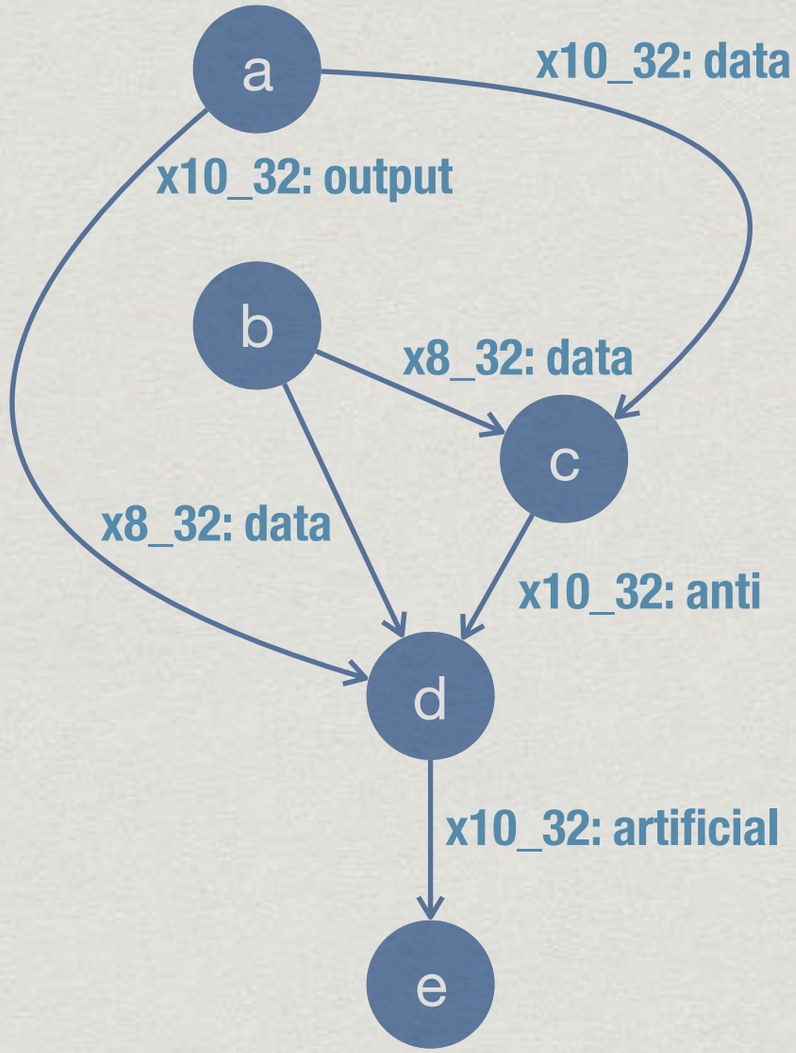

```
a( 1): load $x5_32, $x8_32, @a
c( 2): load $x6_32, $x8_32, @b
e( 3): load $x7_32, $x8_32, @c
b( 4): add $x5_32, $x5_32, $x5_32
d( 5): mul $x5_32, $x5_32, $x6_32
g( 6): load $x9_32, $x8_32, @d
f( 7): mul $x5_32, $x5_32, $x7_32
h( 9): mul $x5_32, $x5_32, $x9_32
i(11): store $x5_32, $x8_32, @a
```

Scheduler in LLVM



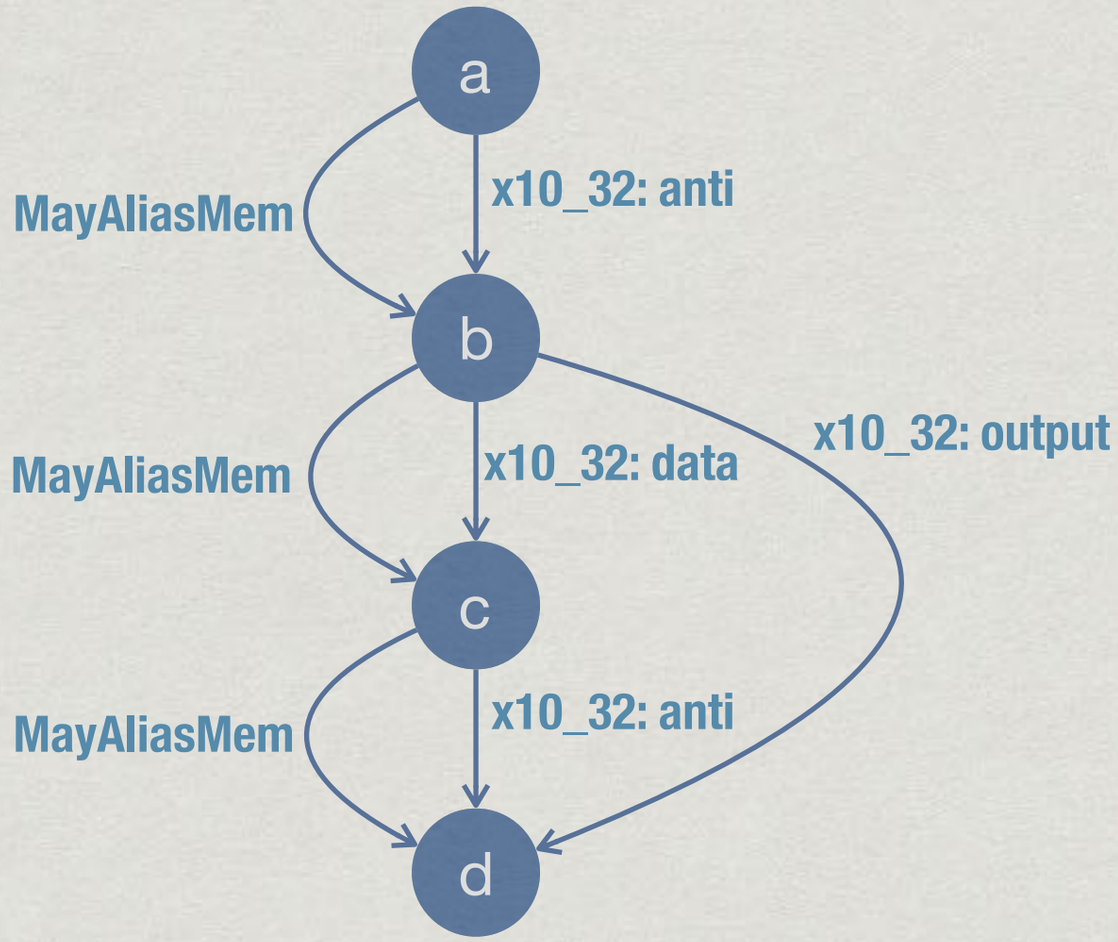
Data Dependency Graph

```
a: $x10_32 = LUI @Arr
b: $x8_32 = CLI 10
c: SW $x8_32, $x10_32, @Arr
d: $x10_32 = ADDI $x8_32, 0
e: Call @foo, implicit $x10_32 (ExitSU)
```



Data Dependency Graph

```
a: SW $x10_32, $x27_32, 12
b: $x10_32 = LW $x9_32, 0
c: SW $x10_32, $x27_32, 0
d: $x10_32 = LW $x8_32, @Glob
...
```



Pipeline Modeling for Target

- Use target description to describe the pipeline model.
- For architecture
  - Create scheduling categories for operands.
    - <Target>Schedule.td
  - Associate scheduling categories to instructions.
    - <Target>InstrInfo.td
- For processor
  - Associate pipeline information to scheduling categories.
    - <Target>Schedule<Processor>.td

Associate per-operand SchedReadWrite types with Instructions by modifying the Instruction definition to inherit from Sched.

Sched
+SchedRW

SchedReadWrite

`SchedRW` lists the per-operand types that map to the machine model of an instruction.

Associate with instructions

Define a scheduler resource associated with a use operand.

Define a scheduler resource associated with a def operand.

SchedRead

SchedWrite

Associate with target

Associate with subtargets

ProcReadAdvance
+Cycles
+ValidWrites

ProcWriteResources
+ProcResources
+ResourceCycles
+Latency

For use with InstRW or ItinRW.

For use with InstRW or ItinRW.

SchedReadAdvance

SchedWriteRes

ReadAdvance
+ReadType

WriteRes
+WriteType

InstRW: Map a set of opcodes to a list of SchedReadWrite types.
This allow the sub target to easily override specific operations.

ItinRW: Map a set of itinerary classes to SchedReadWrite resources.

Define WriteRes and ReadAdvance to associate processor resources and latency with each SchedReadWrite type.

Create scheduling categories

```
def ALUOut : SchedWrite;  // For define operands of ALU op
def ALUIn : SchedRead;    // For use operands of ALU op
def MULOut : SchedWrite;  // For define operands of MUL op
def MULIn : SchedRead;    // For use operands of MUL op
```

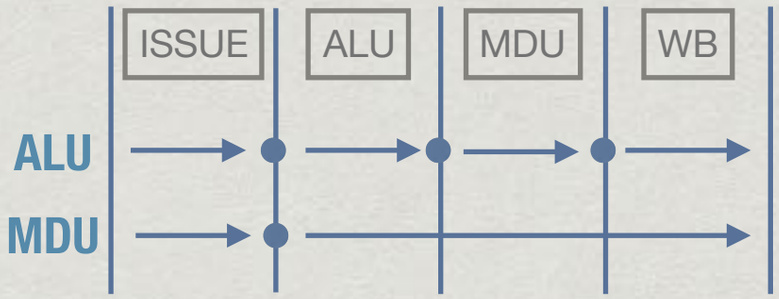
Associate instructions with scheduling categories

```
class ALU_ri<bits<3> funct3, string opcodestr>
    : RVInstI<funct3, OPC_OP_IMM,
              (outs GPR:$rd),
              (ins GPR:$rs1, simm12:$imm12),
             opcodestr, "$rd, $rs1, $imm12”>,
      Sched<[ALUOut, ALUIn]>;
```

Associate pipeline information to scheduling categories

```
def UnitALU : ProcResource<1> { let BufferSize = 0; }
def UnitMDU : ProcResource<1> { let BufferSize = 0; }
```

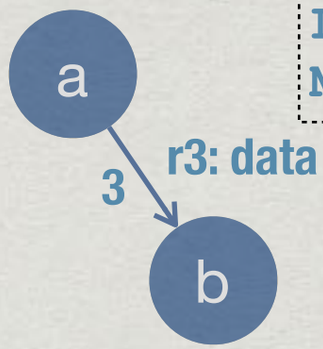
```
def : WriteRes<ALUOut, [UnitALU]> { let Latency = 2; }
def : WriteRes<MULOut, [UnitMDU]> { let Latency = 4; }
def : ReadAdvance<ALUIn, 1>;
def : ReadAdvance<MULIn, 1>;
```

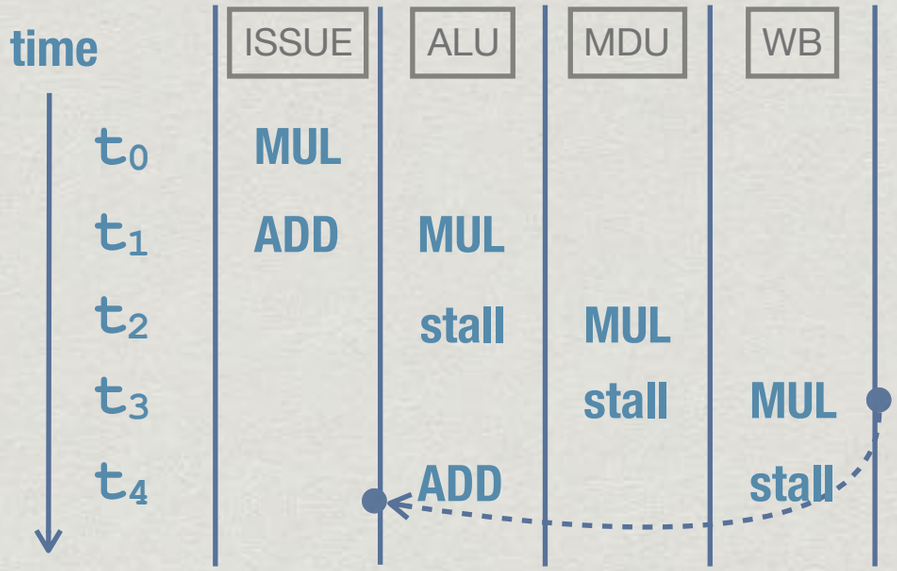


```
def : WriteRes<ALUOut, [UnitALU]> { let Latency = 2; }
def : WriteRes<MULOut, [UnitMDU]> { let Latency = 4; }
def : ReadAdvance<ALUIn, 1>;
def : ReadAdvance<MULIn, 1>;
```

Latency =
MUL's Latency - ADD's Advance

```
a: MUL r3, r3, r2
b: ADD r4, r3, r2
```





GenericScheduler::tryCandidate

- Physical register copies
- Register pressure (Excess, CriticalMax)
- Acyclic Latency
- Cluster
- Weak edges
- Register pressure (CurrentMax)
- Resource
- Latency
- Source order

Customize Scheduler for Target

- Define your scheduling policy.
- Define your scheduling strategy.
- Add DAG mutations.

Implement overrideSchedPolicy

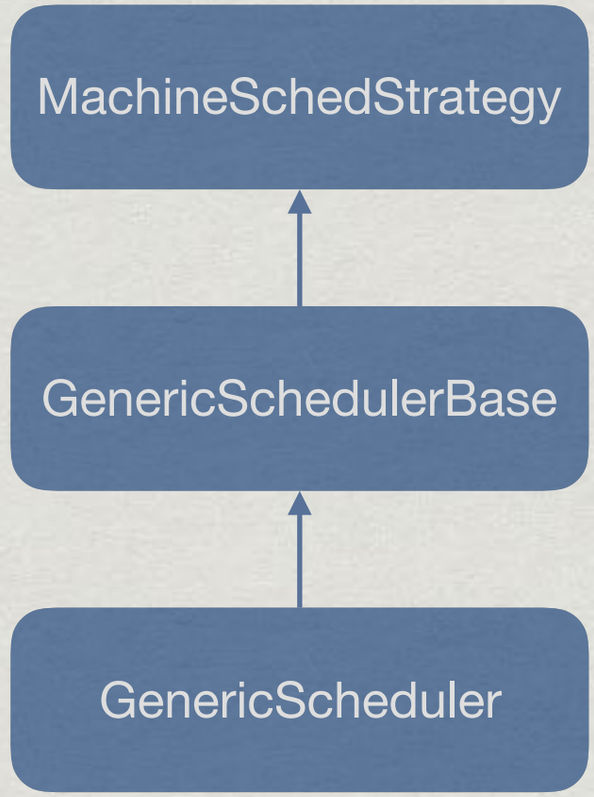
```
struct MachineSchedPolicy {
  bool ShouldTrackPressure = false;
  bool ShouldTrackLaneMasks = false;
  bool OnlyTopDown = false;
  bool OnlyBottomUp = false;
  bool DisableLatencyHeuristic = false;
};


void
<Target>Subtarget::overrideSchedPolicy(MachineSchedPolicy &Policy,
                                       unsigned NumRegionInstrs) const {
  Policy.OnlyTopDown = false;
  Policy.OnlyBottomUp = false;
  Policy.ShouldTrackPressure = true;
}
```

Derive MachineSchedStrategy

```
class YourStrategy : public GenericScheduler {
...
SUnit *pickNode(bool &IsTopNode) override {
  // ...
  // Your heuristic algorithm.
  //

  return GenericScheduler::pickNode(IsTopNode);
}
};
```

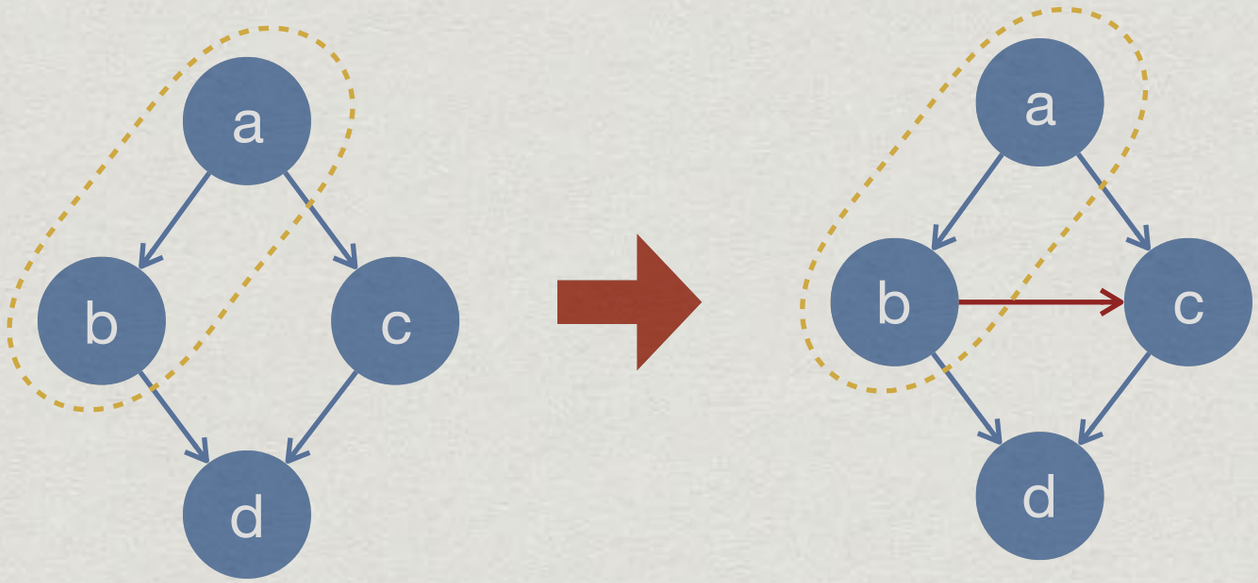


DAG mutations

```
// Implement your mutation.
class CustomMutation : public ScheduleDAGMutation {
public:
  void apply(ScheduleDAGInstrs *DAGInstrs) override;
};

std::unique_ptr<SchuduleDAGMutation>
llvm::createCustomMutation(const <Target>Subtarget *STI) {
  return llvm::make_unique<CustomMutation>(STI);
}

// Install
ScheduleDAGInstr *
createMachineScheduler(MachineSchedContext *C) const override {
  const <Target>Subtarget &STI = C->MF->getSubtarget<<Target>Subtarget>();
  ScheduleDAGMILive *DAG = createGenericSchedLive(C);
  DAG->addMutation(createCustomMutation(STI));
  return DAG;
}
```



Reference

- Engineering a Compiler, 2nd Edition
- 2017 LLVM Developers' Meeting: "Writing Great Machine Schedulers"