Scheduling Example

```
Loop: lw   $t0, 0($s1)      # $t0=array element
      addu $t0, $t0, $s2    # add scalar in $s2
      sw   $t0, 0($s1)      # store result
      addi $s1, $s1,-4      # decrement pointer
      bne  $s1, $zero, Loop # branch $s1!=0
```

	ALU/branch	Load/store	cycle
Loop:	nop	lw $t0, 0($s1)	1
	addi $s1, $s1,-4	nop	2
	addu $t0, $t0, $s2	nop	3
	bne $s1, $zero, Loop	sw $t0, 4($s1)	4

- IPC = 5/4 = 1.25 (c.f. peak IPC = 2)

Loop Unrolling

- Replicate loop body to expose more parallelism
  - Reduces loop-control overhead
- Use different registers per replication
  - Called “register renaming”
  - Avoid loop-carried “anti-dependencies”
    - Store followed by a load of the same register
    - Aka “name dependence”
      - Reuse of a register name

Loop Unrolling

```
Loop: lw   $t0, 0($s1)
      addu $t0, $t0, $s2
      sw   $t0, 0($s1)
      addi $s1, $s1,-4
      lw   $t0, 0($s1)
      addu $t0, $t0, $s2
      sw   $t0, 0($s1)
      addi $s1, $s1,-4
      lw   $t0, 0($s1)
      addu $t0, $t0, $s2
      sw   $t0, 0($s1)
      addi $s1, $s1,-4
      lw   $t0, 0($s1)
      addu $t0, $t0, $s2
      sw   $t0, 0($s1)
      addi $s1, $s1,-4
      bne  $s1, $zero, Loop
```

```
Loop: addi $s1, $s1,-16
      lw   $t0, 16($s1)
      addu $t0, $t0, $s2
      sw   $t0, 16($s1)
      lw   $t1, 12($s1)
      addu $t1, $t1, $s2
      sw   $t1, 12($s1)
      lw   $t2, 8($s1)
      addu $t2, $t2, $s2
      sw   $t2, 8($s1)
      lw   $t3, 4($s1)
      addu $t3, $t3, $s2
      sw   $t3, 4($s1)
      bne  $s1, $zero, Loop
```

Loop Unrolling Example

	ALU/branch	Load/store	cycle
Loop:	addi $s1, $s1,-16	lw $t0, 0($s1)	1
	nop	lw $t1, 12($s1)	2
	addu $t0, $t0, $s2	lw $t2, 8($s1)	3
	addu $t1, $t1, $s2	lw $t3, 4($s1)	4
	addu $t2, $t2, $s2	sw $t0, 16($s1)	5
	addu $t3, $t4, $s2	sw $t1, 12($s1)	6
	nop	sw $t2, 8($s1)	7
	bne $s1, $zero, Loop	sw $t3, 4($s1)	8

(a 0 displacement & the orignal value in $s1 are used in the first lw instruction)

- IPC = 14/8 = 1.75
  - Closer to 2, but at cost of registers and code size

Dynamic Multiple Issue

- "Superscalar" processors
- CPU selects instructions to execute each cycle
  - Avoiding structural and data hazards
- Avoids the need for compiler scheduling
  - Though compiler may still help
  - Code semantics ensured by the CPU

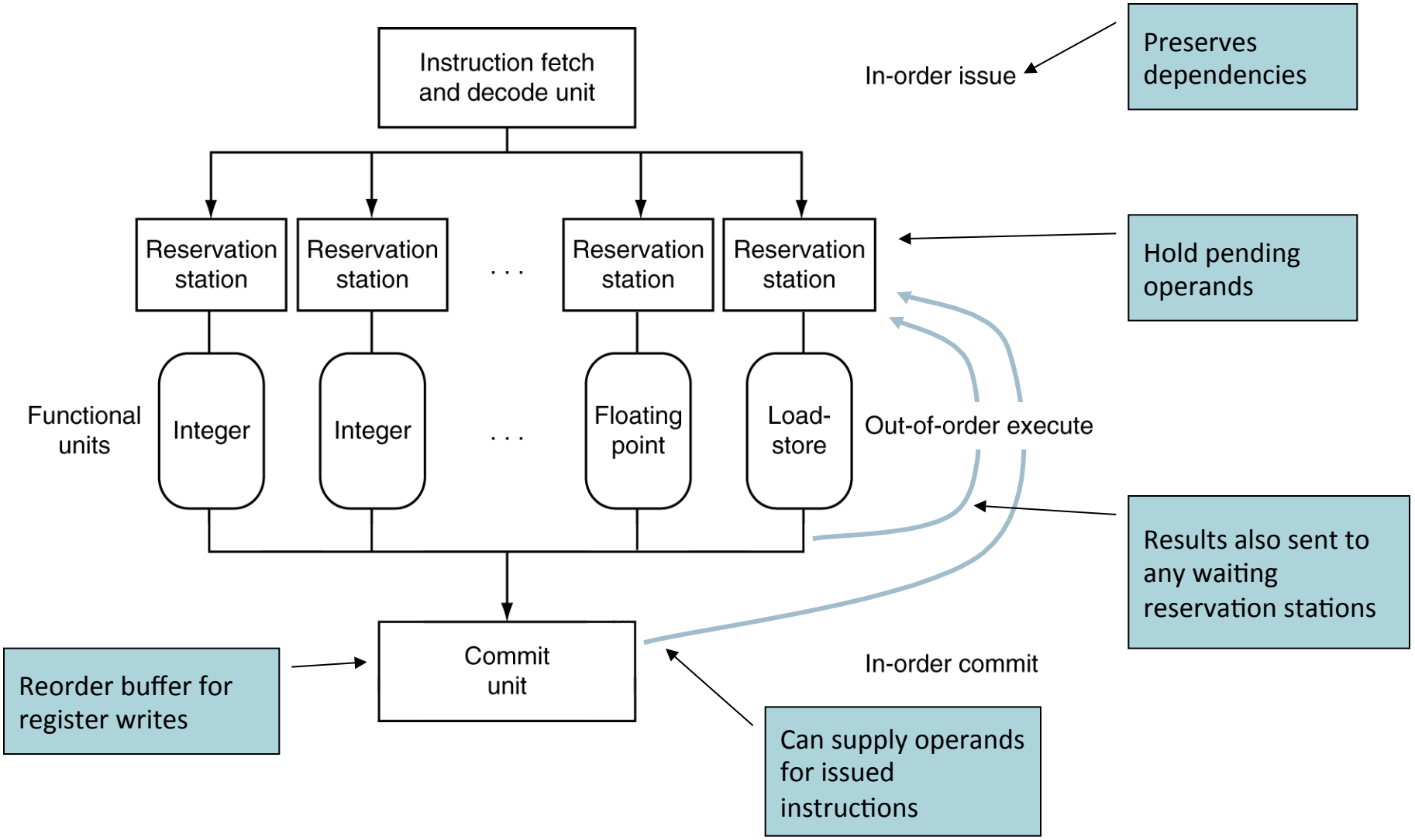
Dynamic Scheduling

- Allow the CPU to execute instructions out of order to avoid stalls
  - But commit result to registers in order
- Example

```
lw    $t0, 20($s2)
addu  $t1, $t0, $t2
sub   $s4, $s4, $t3
slti  $t5, $s4, 20
```

  - Can start sub while addu is waiting for lw

Dynamic Scheduling



Register renaming

- Register renaming via reservation stations (RS) & ROB
  - ROB is reorder buffer
- On instruction issue to reservation station
  - copy available operand to reservation station
  - from register file or from ROB
    - Once done, register can be overwritten
  - When available, operands are provided to RS
    - Come from the function unit producing the result
    - Register update may not be required