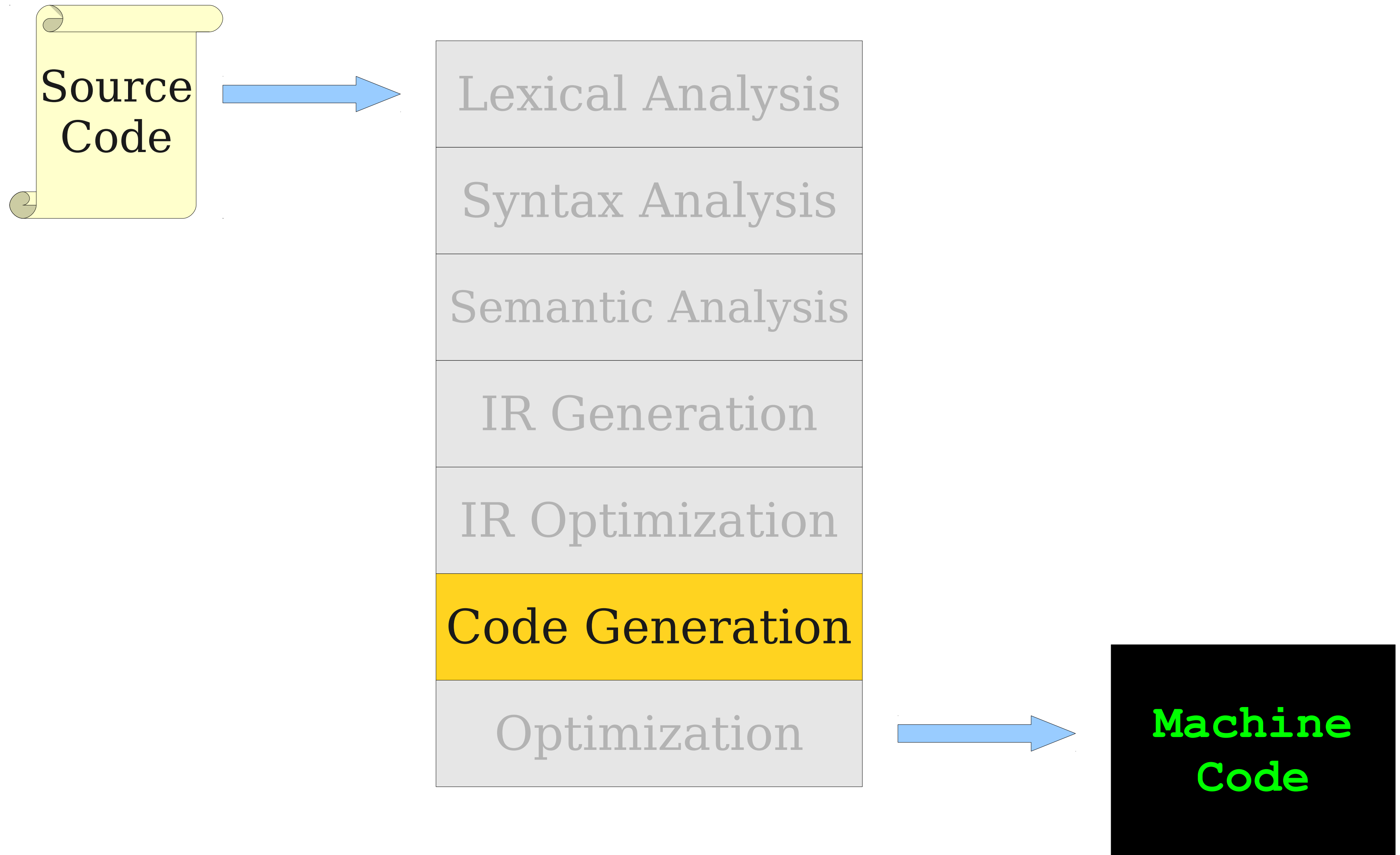
# Code Optimization

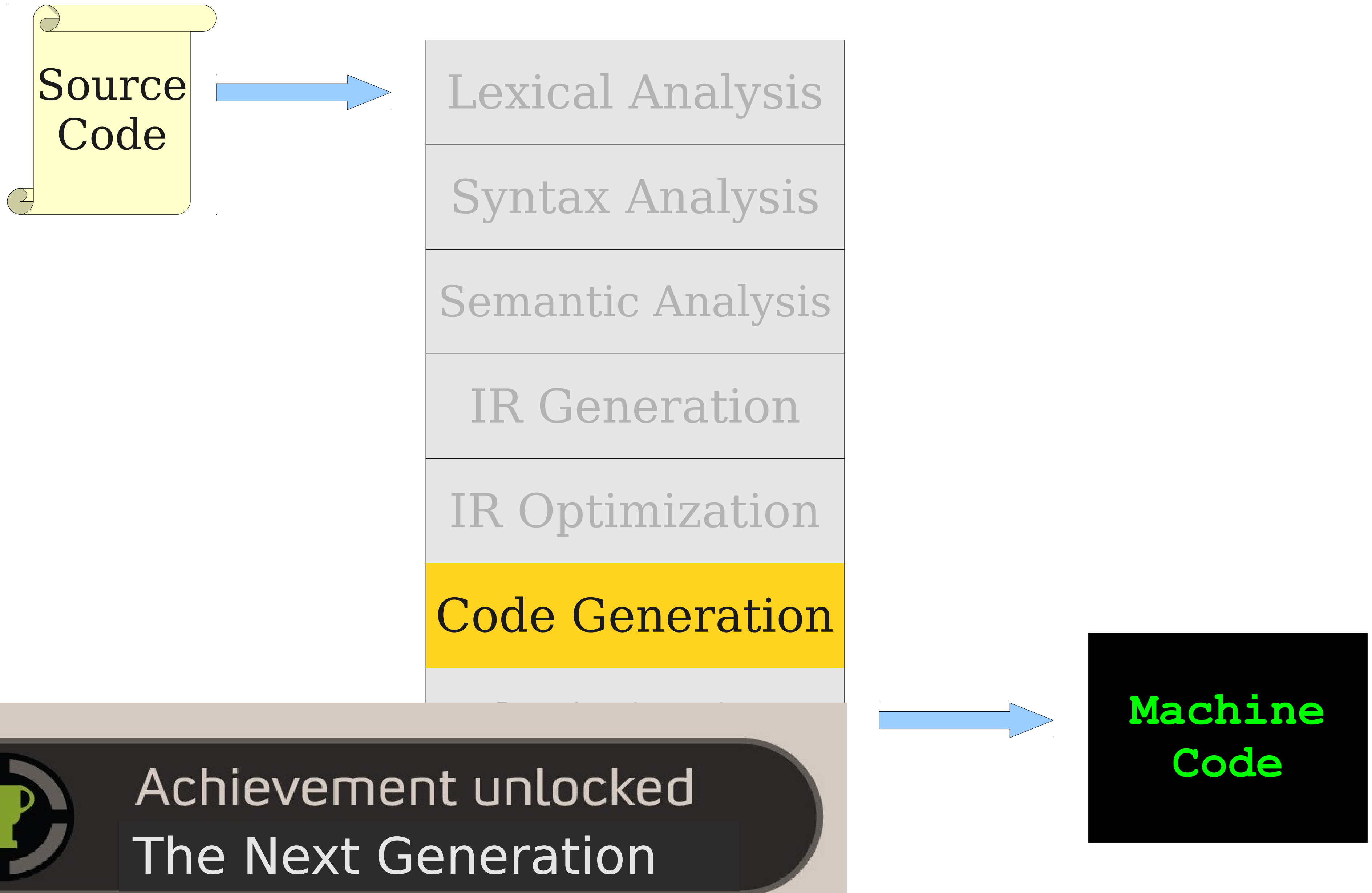
# Announcements

- Programming Project 4 due Saturday at 11:30AM.
  - Stop by office hours!
  - Ask questions via email!
  - Ask questions via Piazza!
- Keith has extra office hours this Wednesday and Friday from 2PM – 4PM.
- Online course evaluation available on Axess.
  - Please give feedback!

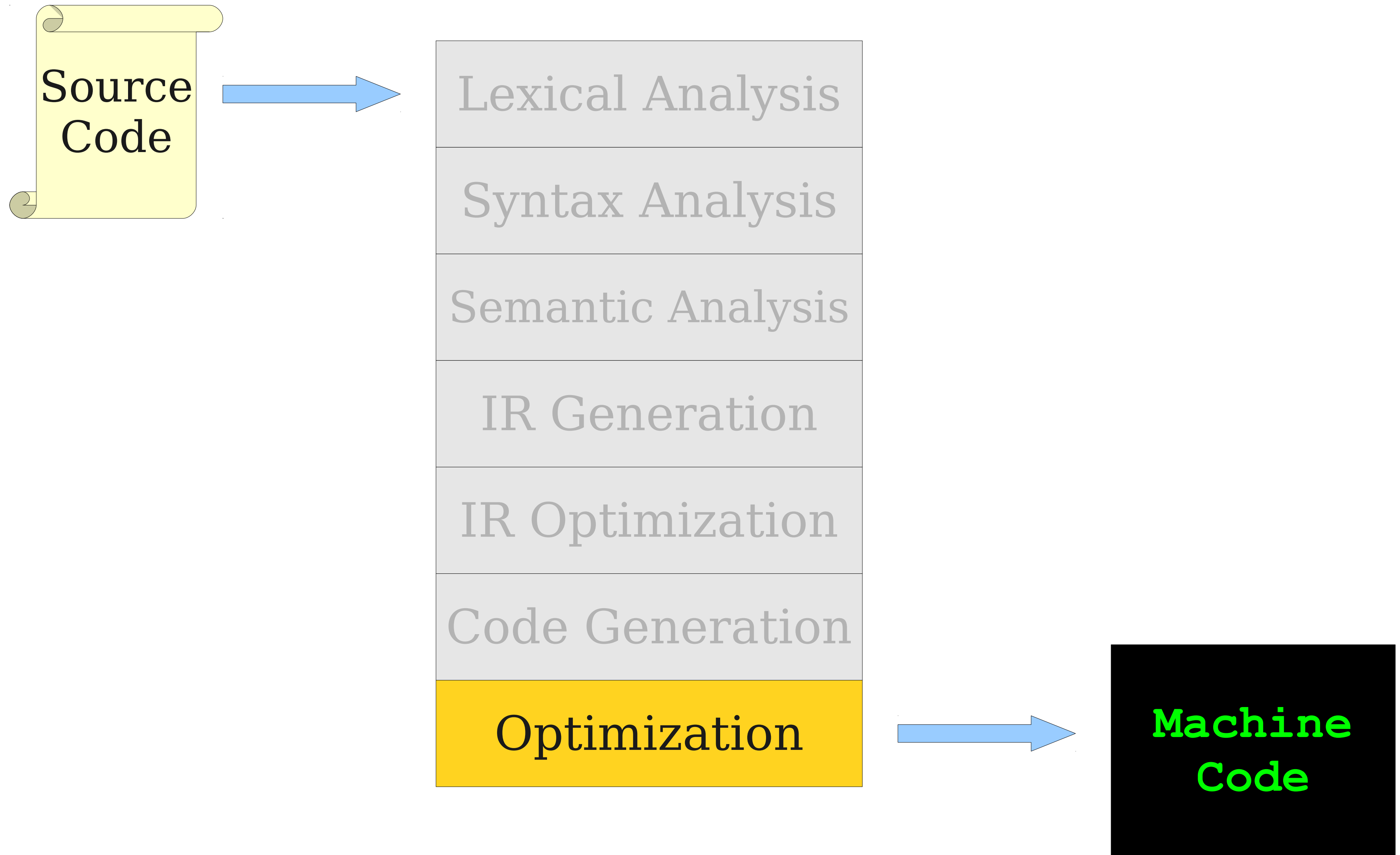
# Where We Are



# Where We Are



# Where We Are



# Final Code Optimization

- **Goal**: Optimize generated code by exploiting machine-dependent properties not visible at the IR level.
- Critical step in most compilers, but often very messy:
  - Techniques developed for one machine may be completely useless on another.
  - Techniques developed for one language may be completely useless with another.

# Outline

- Goals for Today:
  - Explore important properties of machines and how they impact performance.
  - Survey common optimization techniques and the theory behind them.
  - Motivate you to take CS243 to learn more!
- Non-Goals for Today:
  - Explore tricky details of the algorithms.

# Optimizations for Pipelining

# Processor Pipelines

# Processor Pipelines

```
add $t2, $t0, $t1  # $t2 = $t0 + $t1
add $t5, $t3, $t4  # $t5 = $t3 + $t4
add $t8, $t6, $t7  # $t8 = $t6 + $t7
```

# Processor Pipelines

**Instruction Decoder**

```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t5, $t3, $t4   # $t5 = $t3 + $t4
add $t8, $t6, $t7   # $t8 = $t6 + $t7
```

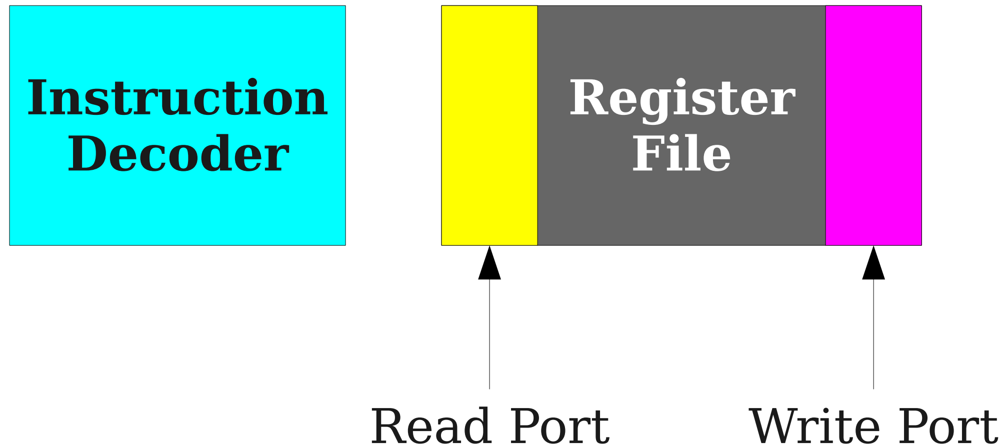
# Processor Pipelines

Instruction Decoder

Register File

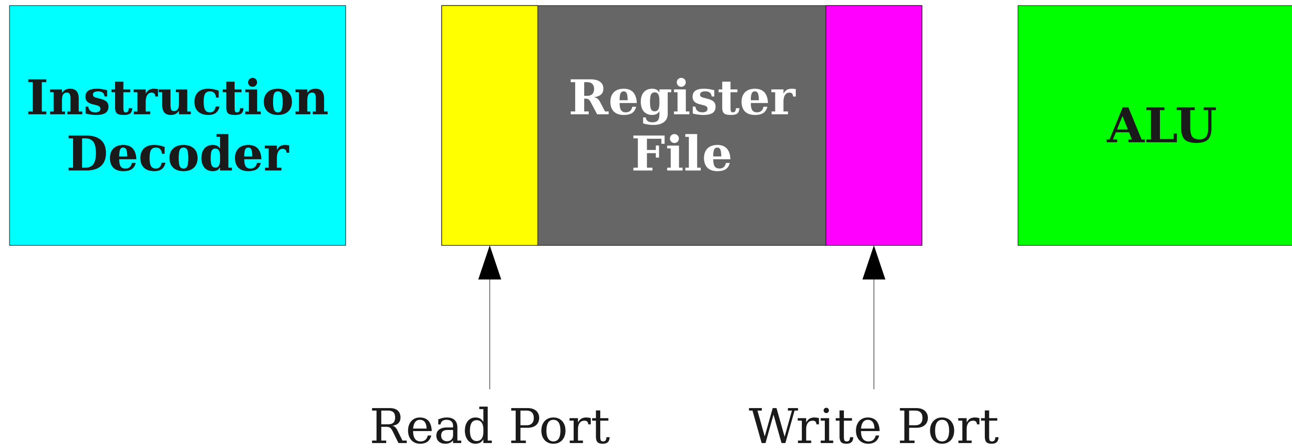
```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t5, $t3, $t4   # $t5 = $t3 + $t4
add $t8, $t6, $t7   # $t8 = $t6 + $t7
```

# Processor Pipelines



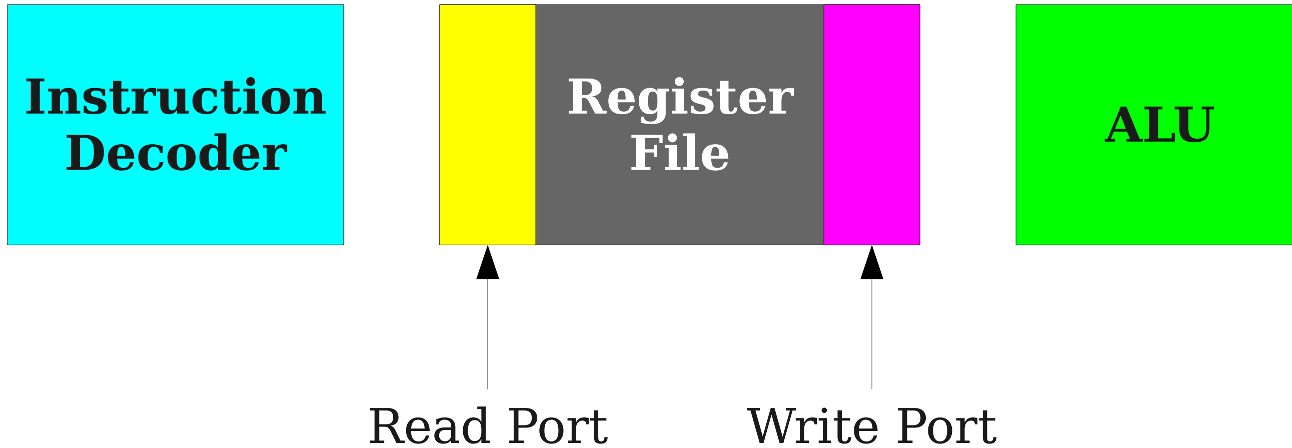

```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t5, $t3, $t4   # $t5 = $t3 + $t4
add $t8, $t6, $t7   # $t8 = $t6 + $t7
```

# Processor Pipelines




```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t5, $t3, $t4   # $t5 = $t3 + $t4
add $t8, $t6, $t7   # $t8 = $t6 + $t7
```

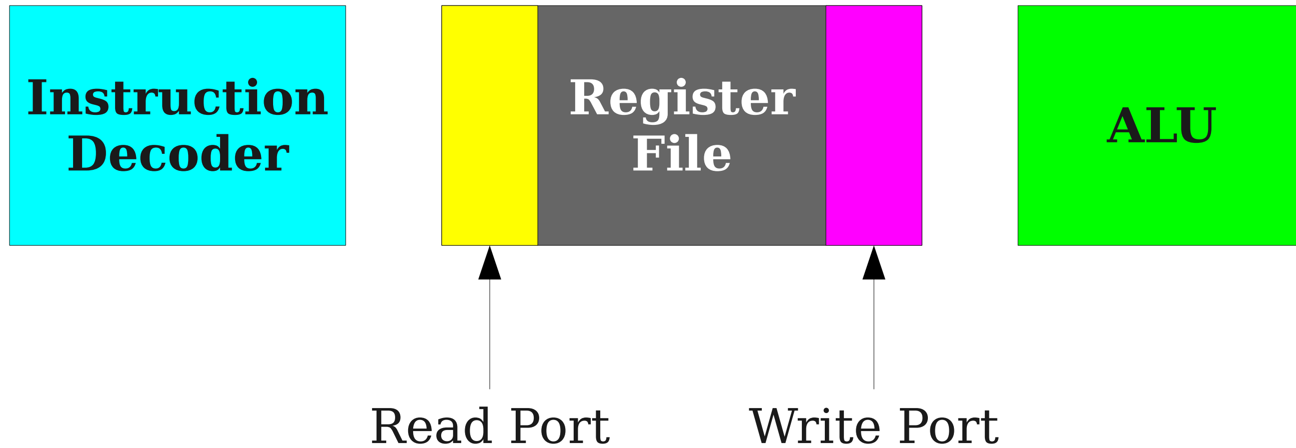
# Processor Pipelines




| ID | RR | ALU | RW |
| --- | --- | --- | --- |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t5, $t3, $t4   # $t5 = $t3 + $t4
add $t8, $t6, $t7   # $t8 = $t6 + $t7
```

# Processor Pipelines




```
add $t2, $t0, $t1    # $t2 = $t0 + $t1
add $t5, $t3, $t4    # $t5 = $t3 + $t4
add $t8, $t6, $t7    # $t8 = $t6 + $t7
```

| ID | RR | ALU | RW |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Processor Pipelines

Instruction Decoder

Register File

ALU

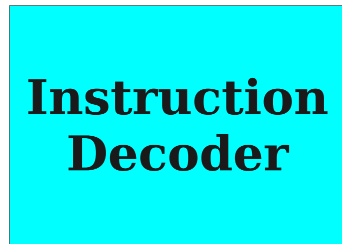
Read Port

Write Port

| ID | RR | ALU | RW |
|---|---|---|---|

```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t5, $t3, $t4   # $t5 = $t3 + $t4
add $t8, $t6, $t7   # $t8 = $t6 + $t7
```

# Processor Pipelines



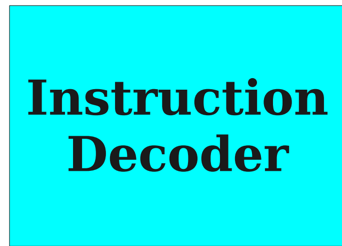
Read Port

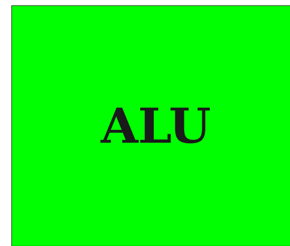
Write Port

**ID RR ALU RW**

```
add $t2, $t0, $t1    # $t2 = $t0 + $t1
add $t5, $t3, $t4    # $t5 = $t3 + $t4
add $t8, $t6, $t7    # $t8 = $t6 + $t7
```

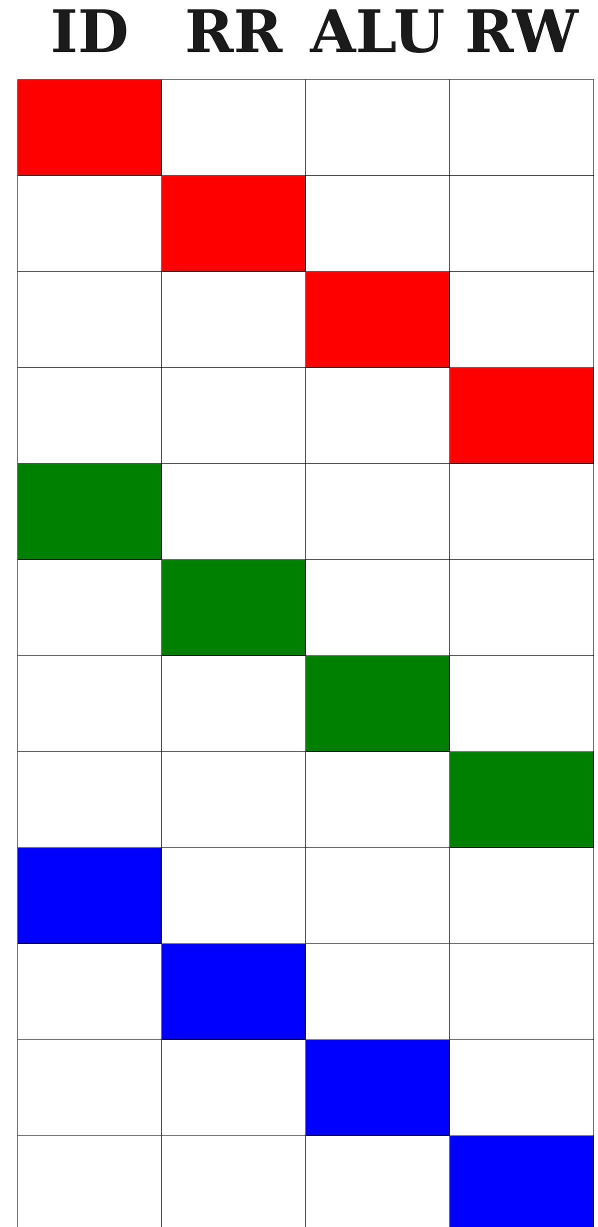
# Processor Pipelines






Read Port

Write Port




```
add $t2, $t0, $t1    # $t2 = $t0 + $t1
add $t5, $t3, $t4    # $t5 = $t3 + $t4
add $t8, $t6, $t7    # $t8 = $t6 + $t7
```

# Processor Pipelines

ID RR ALU RW

Instruction Decoder

Register File

ALU

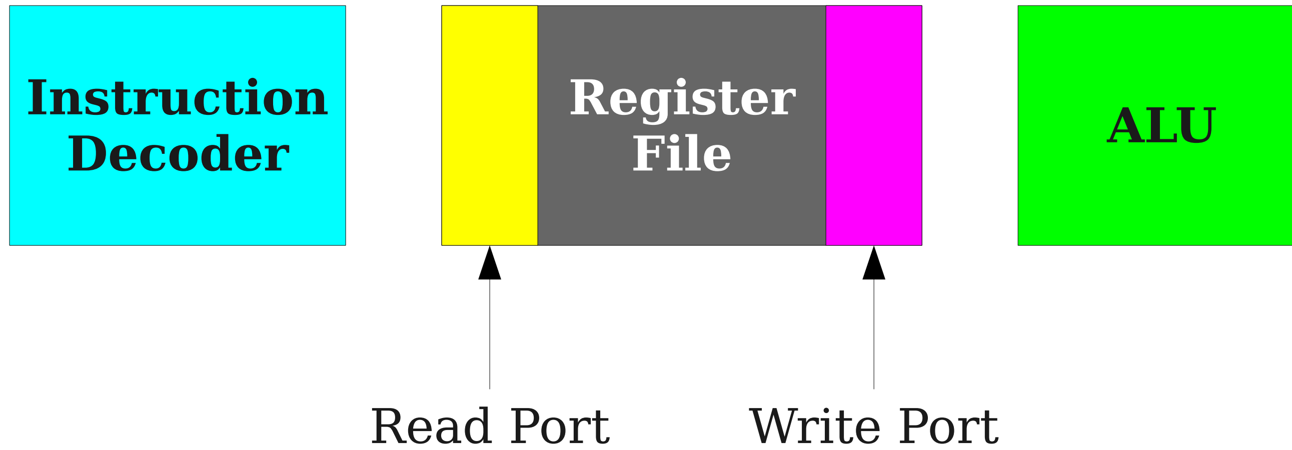
Read Port

Write Port

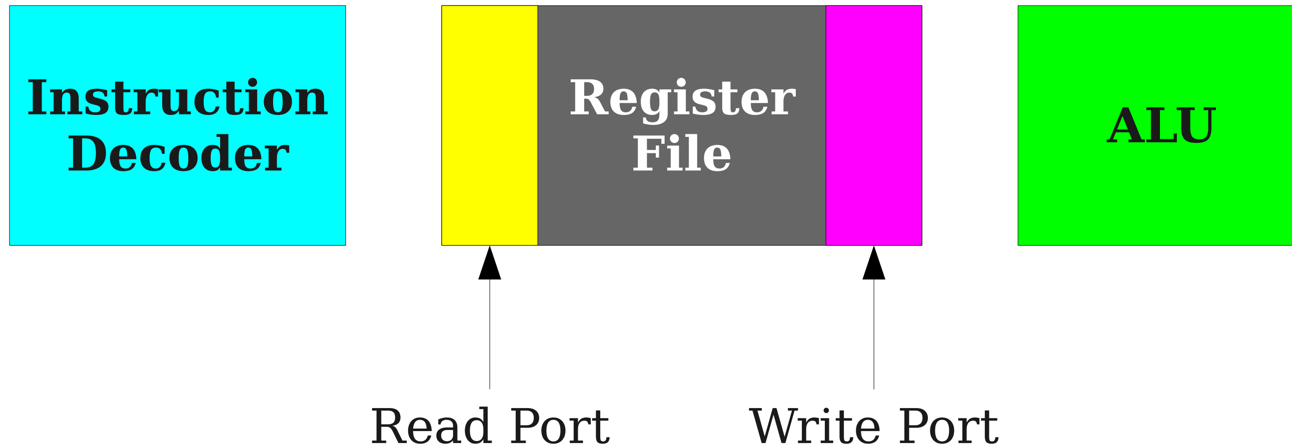
```
add $t2, $t0, $t1     # $t2 = $t0 + $t1
add $t5, $t3, $t4     # $t5 = $t3 + $t4
add $t8, $t6, $t7     # $t8 = $t6 + $t7
```

# Pipeline Hazards

# Pipeline Hazards

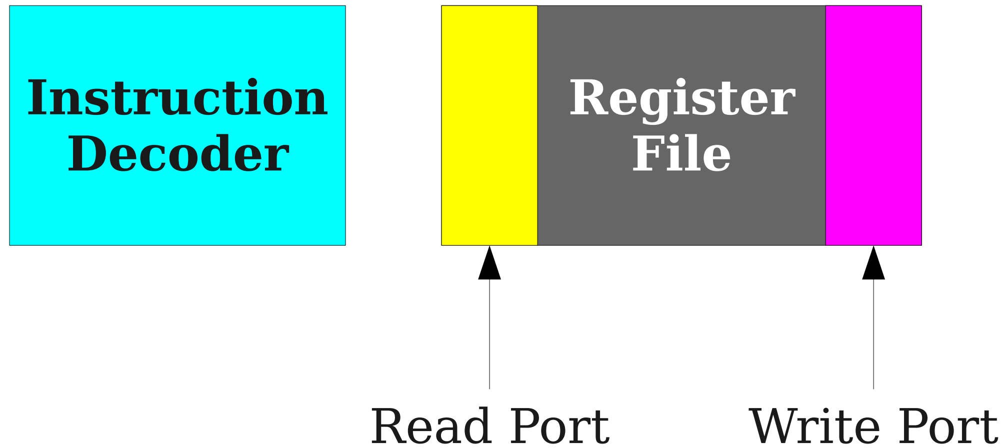


# Pipeline Hazards




```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t4, $t3, $t2   # $t5 = $t3 + $t2
add $t7, $t5, $t6   # $t7 = $t5 + $t6
add $t0, $t0, $t7   # $t0 = $t0 + $t7
```

# Pipeline Hazards




| ID | RR | ALU | RW |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

```
add $t2, $t0, $t1    # $t2 = $t0 + $t1
add $t4, $t3, $t2    # $t5 = $t3 + $t2
add $t7, $t5, $t6    # $t7 = $t5 + $t6
add $t0, $t0, $t7    # $t0 = $t0 + $t7
```

# Pipeline Hazards

| ID | RR | ALU | RW |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

**Instruction Decoder**

**Register File**

**ALU**

Read Port

Write Port

```
add $t2, $t0, $t1    # $t2 = $t0 + $t1
add $t4, $t3, $t2    # $t5 = $t3 + $t2
add $t7, $t5, $t6    # $t7 = $t5 + $t6
add $t0, $t0, $t7    # $t0 = $t0 + $t7
```

# Pipeline Hazards

ID RR ALU RW

**Instruction Decoder**

**Register File**

**ALU**

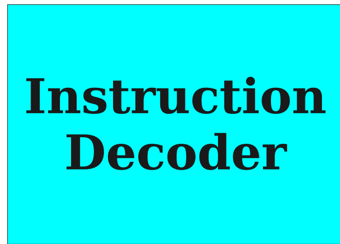
Read Port

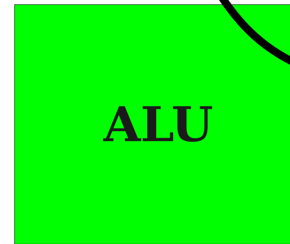
Write Port

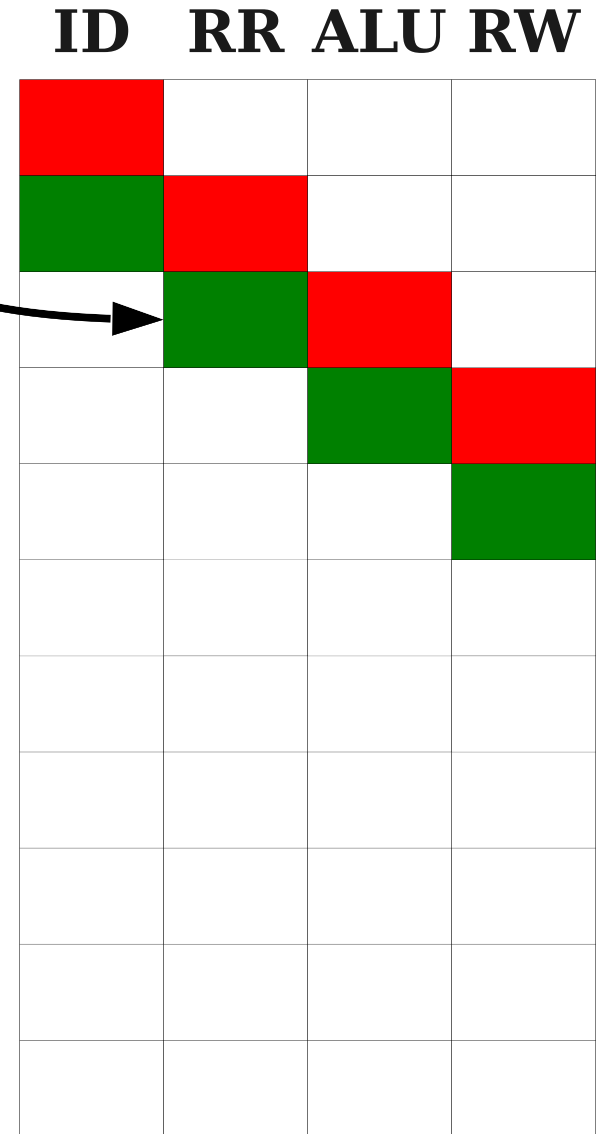
```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t4, $t3, $t2   # $t5 = $t3 + $t2
add $t7, $t5, $t6   # $t7 = $t5 + $t6
add $t0, $t0, $t7   # $t0 = $t0 + $t7
```

# Pipeline Hazards

This value isn't ready yet!






Read Port

Write Port

```
add $t2, $t0, $t1    # $t2 = $t0 + $t1
add $t4, $t3, $t2    # $t5 = $t3 + $t2
add $t7, $t5, $t6    # $t7 = $t5 + $t6
add $t0, $t0, $t7    # $t0 = $t0 + $t7
```

# Pipeline Hazards

Instruction Decoder

Register File

ALU

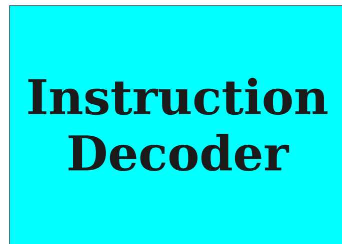
Read Port

Write Port

ID RR ALU RW

```
add $t2, $t0, $t1    # $t2 = $t0 + $t1
add $t4, $t3, $t2    # $t5 = $t3 + $t2
add $t7, $t5, $t6    # $t7 = $t5 + $t6
add $t0, $t0, $t7    # $t0 = $t0 + $t7
```

# Pipeline Hazards



Read Port

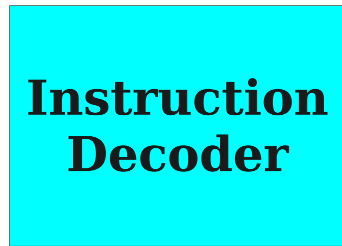
Write Port

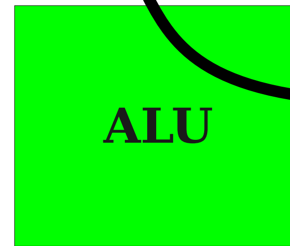
ID RR ALU RW

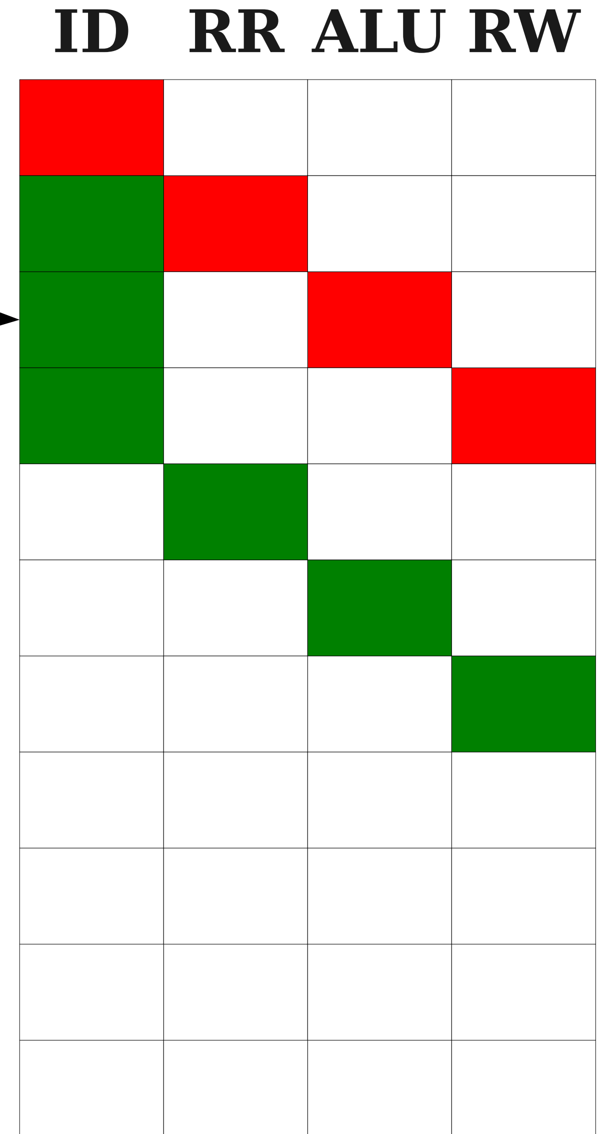
```
add $t2, $t0, $t1    # $t2 = $t0 + $t1
add $t4, $t3, $t2    # $t5 = $t3 + $t2
add $t7, $t5, $t6    # $t7 = $t5 + $t6
add $t0, $t0, $t7    # $t0 = $t0 + $t7
```

# Pipeline Hazards

Stall pipeline until value is ready





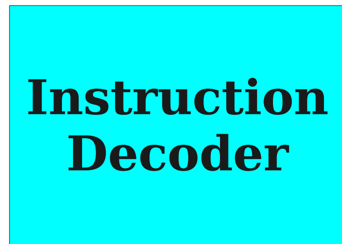




Read Port

Write Port

```
add $t2, $t0, $t1    # $t2 = $t0 + $t1
add $t4, $t3, $t2    # $t5 = $t3 + $t2
add $t7, $t5, $t6    # $t7 = $t5 + $t6
add $t0, $t0, $t7    # $t0 = $t0 + $t7
```

# Pipeline Hazards



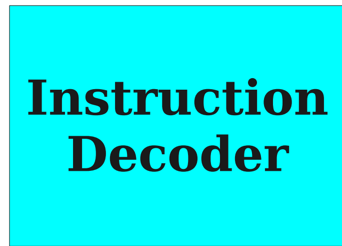
Read Port

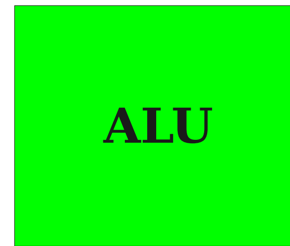
Write Port

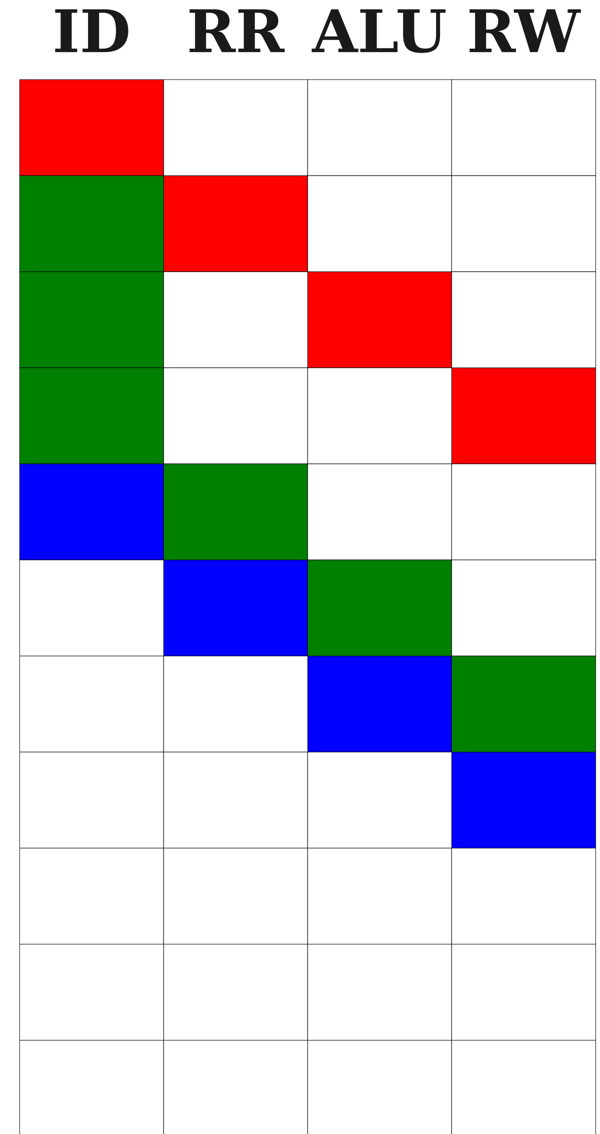
**ID RR ALU RW**

```
add $t2, $t0, $t1    # $t2 = $t0 + $t1
add $t4, $t3, $t2    # $t5 = $t3 + $t2
add $t7, $t5, $t6    # $t7 = $t5 + $t6
add $t0, $t0, $t7    # $t0 = $t0 + $t7
```

# Pipeline Hazards





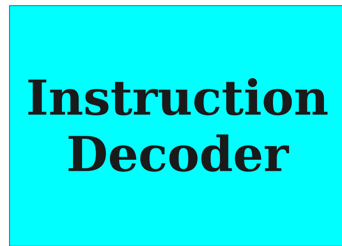



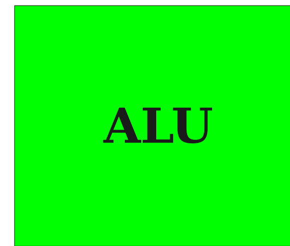

Read Port

Write Port

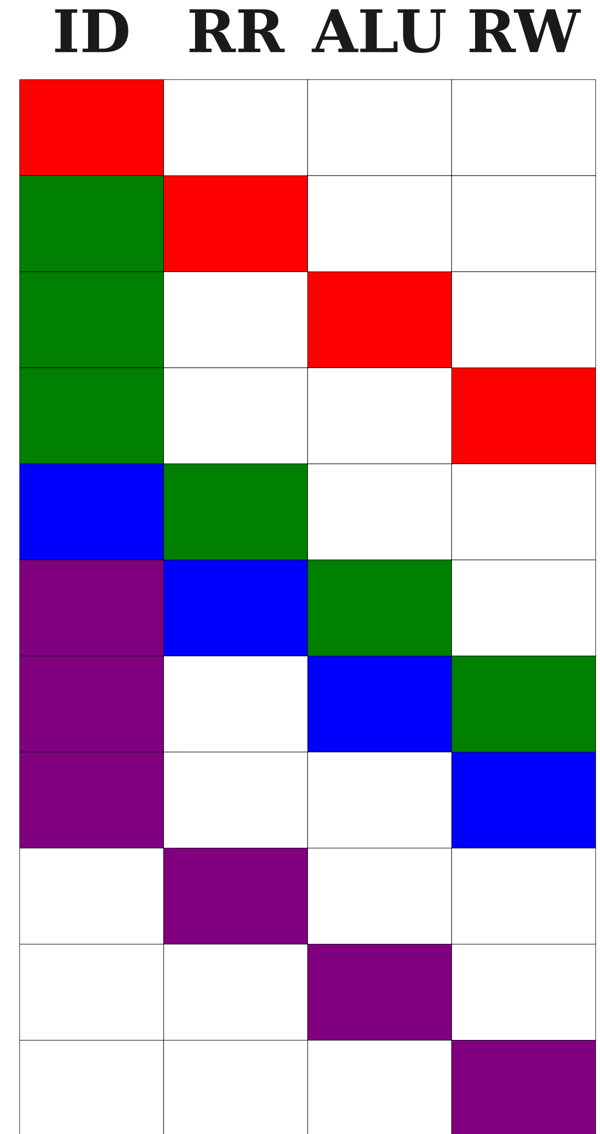
```
add $t2, $t0, $t1    # $t2 = $t0 + $t1
add $t4, $t3, $t2    # $t5 = $t3 + $t2
add $t7, $t5, $t6    # $t7 = $t5 + $t6
add $t0, $t0, $t7    # $t0 = $t0 + $t7
```

# Pipeline Hazards



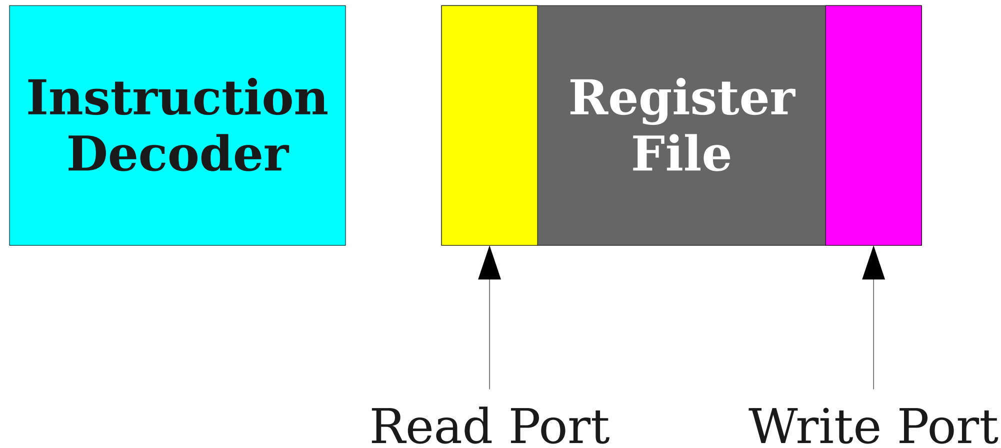



Read Port Write Port




```
add $t2, $t0, $t1    # $t2 = $t0 + $t1
add $t4, $t3, $t2    # $t5 = $t3 + $t2
add $t7, $t5, $t6    # $t7 = $t5 + $t6
add $t0, $t0, $t7    # $t0 = $t0 + $t7
```

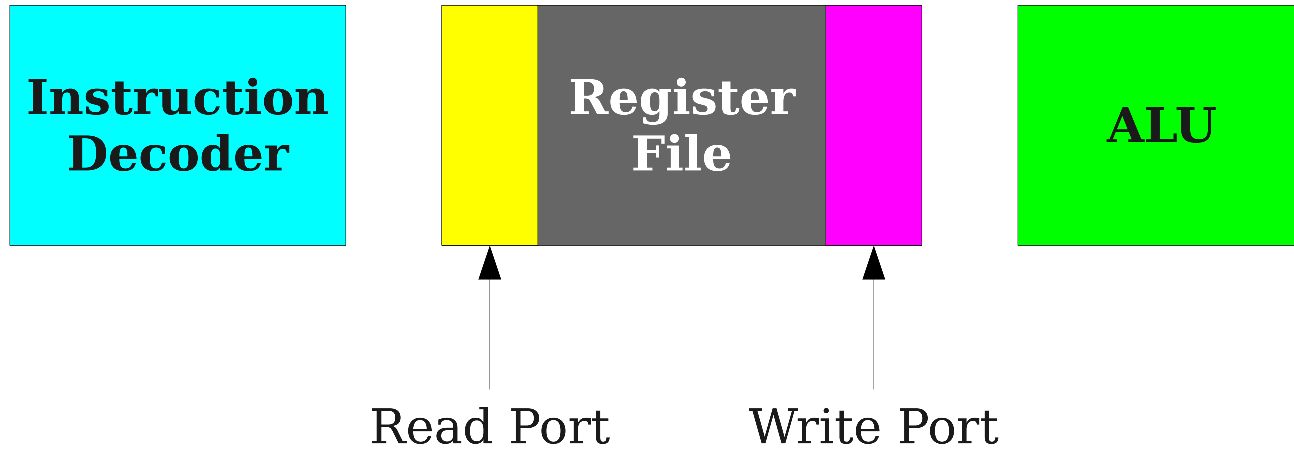
# Pipeline Hazards




| ID | RR | ALU | RW |
| --- | --- | --- | --- |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t4, $t3, $t2   # $t5 = $t3 + $t2
add $t7, $t5, $t6   # $t7 = $t5 + $t6
add $t0, $t0, $t7   # $t0 = $t0 + $t7
```

# Pipeline Hazards




| ID | RR | ALU | RW |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t7, $t5, $t6   # $t7 = $t5 + $t6
add $t4, $t3, $t2   # $t5 = $t3 + $t2
add $t0, $t0, $t7   # $t0 = $t0 + $t7
```

# Pipeline Hazards

Instruction Decoder

Register File

ALU

Read Port

Write Port

| ID | RR | ALU | RW |
|---|---|---|---|

```
add $t2, $t0, $t1    # $t2 = $t0 + $t1
add $t7, $t5, $t6    # $t7 = $t5 + $t6
add $t4, $t3, $t2    # $t5 = $t3 + $t2
add $t0, $t0, $t7    # $t0 = $t0 + $t7
```

# Pipeline Hazards

ID RR ALU RW

Instruction Decoder

Register File

ALU

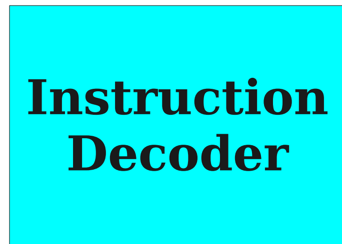
Read Port

Write Port

```
add $t2, $t0, $t1    # $t2 = $t0 + $t1
add $t7, $t5, $t6    # $t7 = $t5 + $t6
add $t4, $t3, $t2    # $t5 = $t3 + $t2
add $t0, $t0, $t7    # $t0 = $t0 + $t7
```

# Pipeline Hazards

ID RR ALU RW

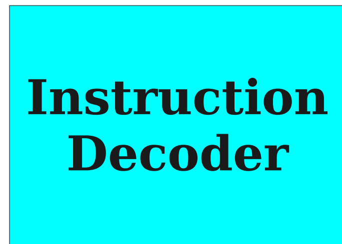


Read Port

Write Port

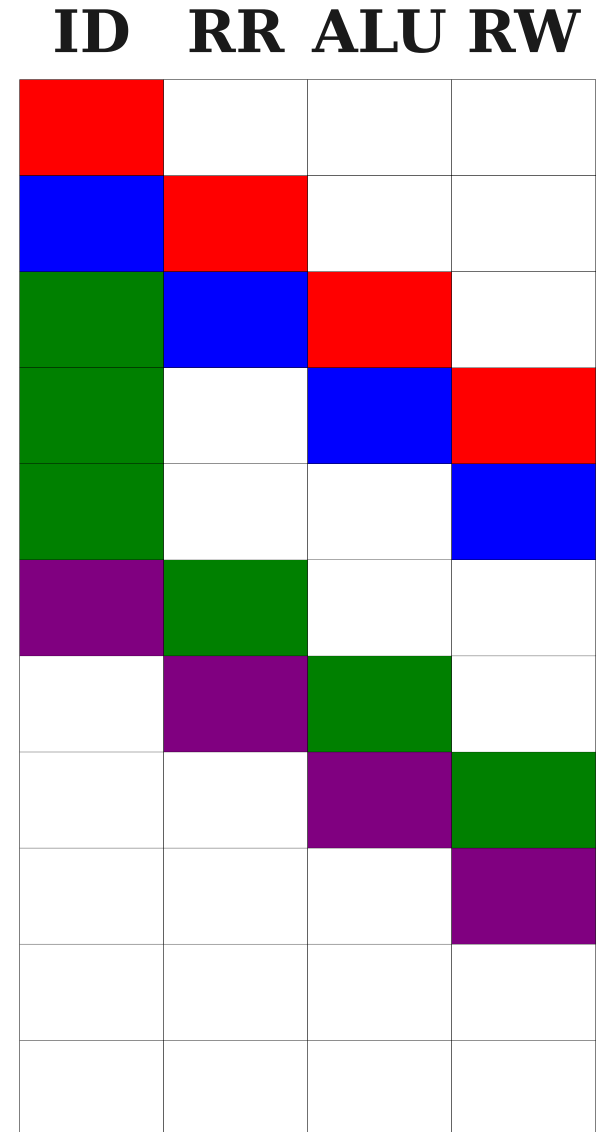
```
add $t2, $t0, $t1    # $t2 = $t0 + $t1
add $t7, $t5, $t6    # $t7 = $t5 + $t6
add $t4, $t3, $t2    # $t5 = $t3 + $t2
add $t0, $t0, $t7    # $t0 = $t0 + $t7
```

# Pipeline Hazards



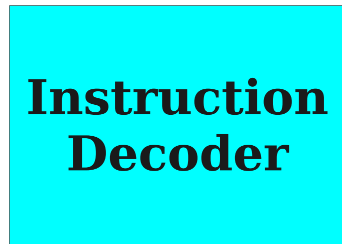
Read Port

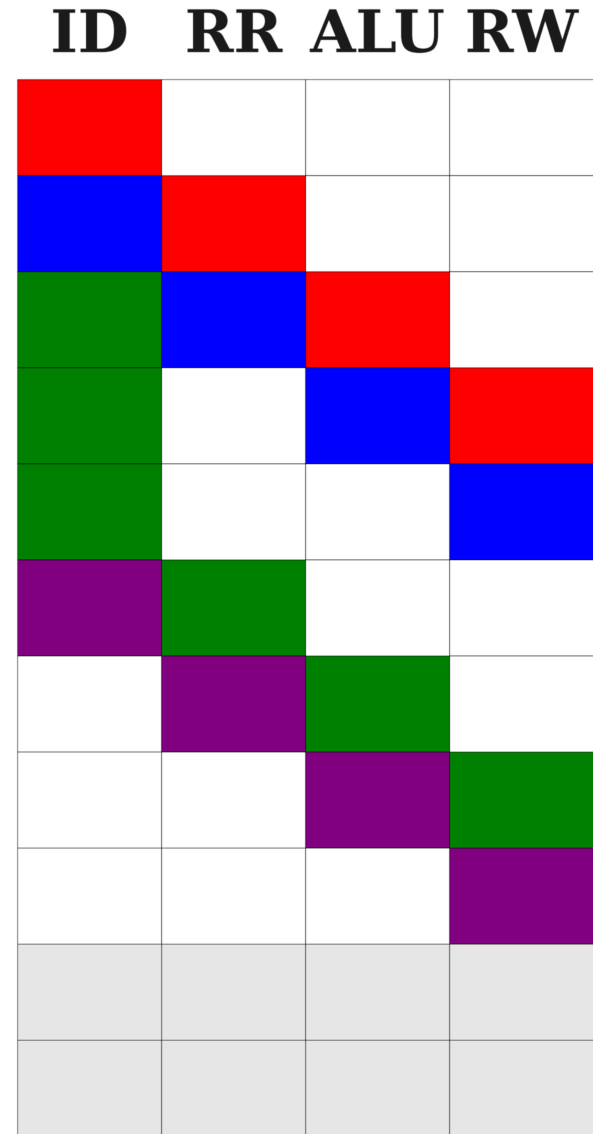
Write Port




```
add $t2, $t0, $t1    # $t2 = $t0 + $t1
add $t7, $t5, $t6    # $t7 = $t5 + $t6
add $t4, $t3, $t2    # $t5 = $t3 + $t2
add $t0, $t0, $t7    # $t0 = $t0 + $t7
```

# Pipeline Hazards






```
add $t2, $t0, $t1   # $t2 = $t0 + $t1
add $t7, $t5, $t6   # $t7 = $t5 + $t6
add $t4, $t3, $t2   # $t5 = $t3 + $t2
add $t0, $t0, $t7   # $t0 = $t0 + $t7
```

Two clock cycles faster!

# Instruction Scheduling

- Because of processor pipelining, the order in which instructions are executed can impact performance.
- **Instruction scheduling** is the reordering or insertion of machine instructions to increase performance.
- All good optimizing compilers have some sort of instruction scheduling support.

# Data Dependencies

- A **data dependency** in machine code is a set of instructions whose behavior depends on one another.
- Intuitively, a set of instructions that cannot be reordered around each other.
- Three types of data dependencies:

| **Read-after-Write (RAW)** | **Write-after-Read (WAR)** | **Write-after-Write (WAW)** |
|---|---|---|
| x = ...<br>... = x | ... = x<br>x = ... | x = ...<br>x = ... |

# Finding Data Dependencies

```
t0 = t1 + t2
t1 = t0 + t1
t3 = t2 + t4
t0 = t1 + t2
t5 = t3 + t4
t6 = t2 + t3
```

# Finding Data Dependencies

```
t0 = t1 + t2

t1 = t0 + t1

t3 = t2 + t4

t0 = t1 + t2

t5 = t3 + t4

t6 = t2 + t7
```

# Finding Data Dependencies

```
t0 = t1 + t2
```

```
t1 = t0 + t1
```

```
t3 = t2 + t4
```

```
t0 = t1 + t2
```

```
t5 = t3 + t4
```

```
t6 = t2 + t7
```

# Finding Data Dependencies

```
t0 = t1 + t2
```

```
t1 = t0 + t1
```

```
t3 = t2 + t4
```

```
t0 = t1 + t2
```

```
t5 = t3 + t4
```

```
t6 = t2 + t7
```

# Finding Data Dependencies

```
t0 = t1 + t2
```

```
t1 = t0 + t1
```

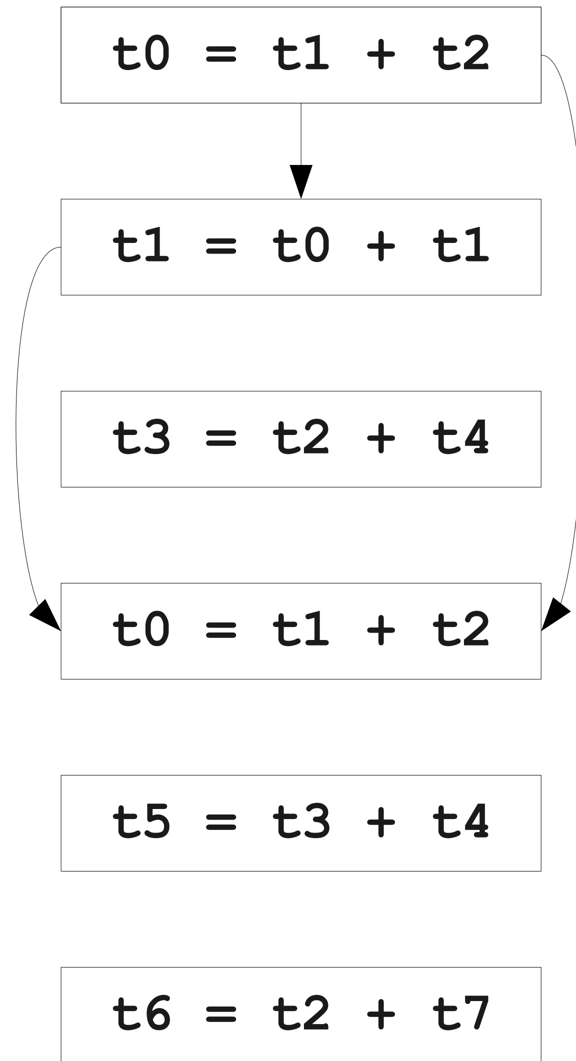
```
t3 = t2 + t4
```

```
t0 = t1 + t2
```

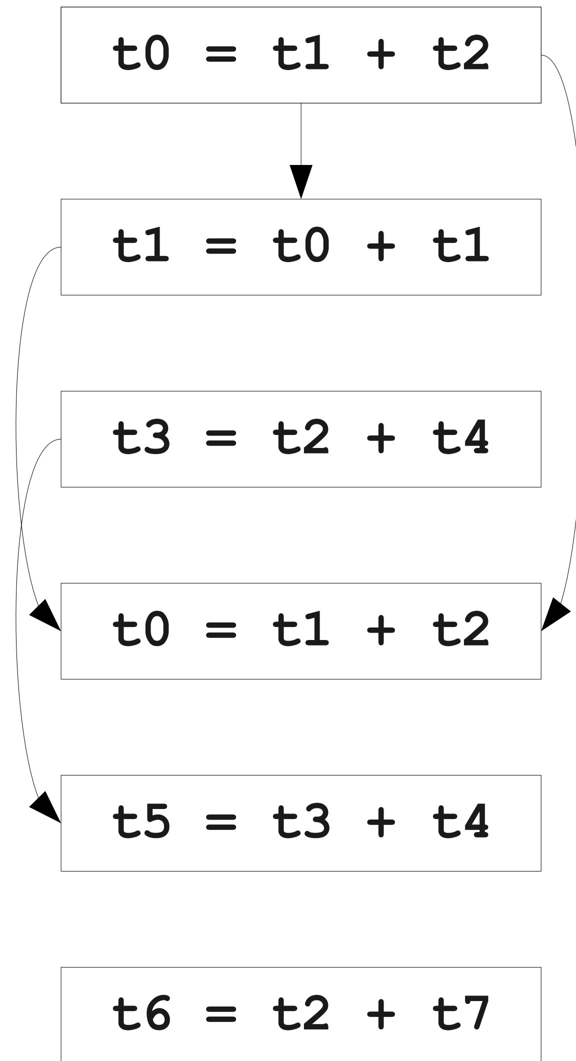
```
t5 = t3 + t4
```

```
t6 = t2 + t7
```

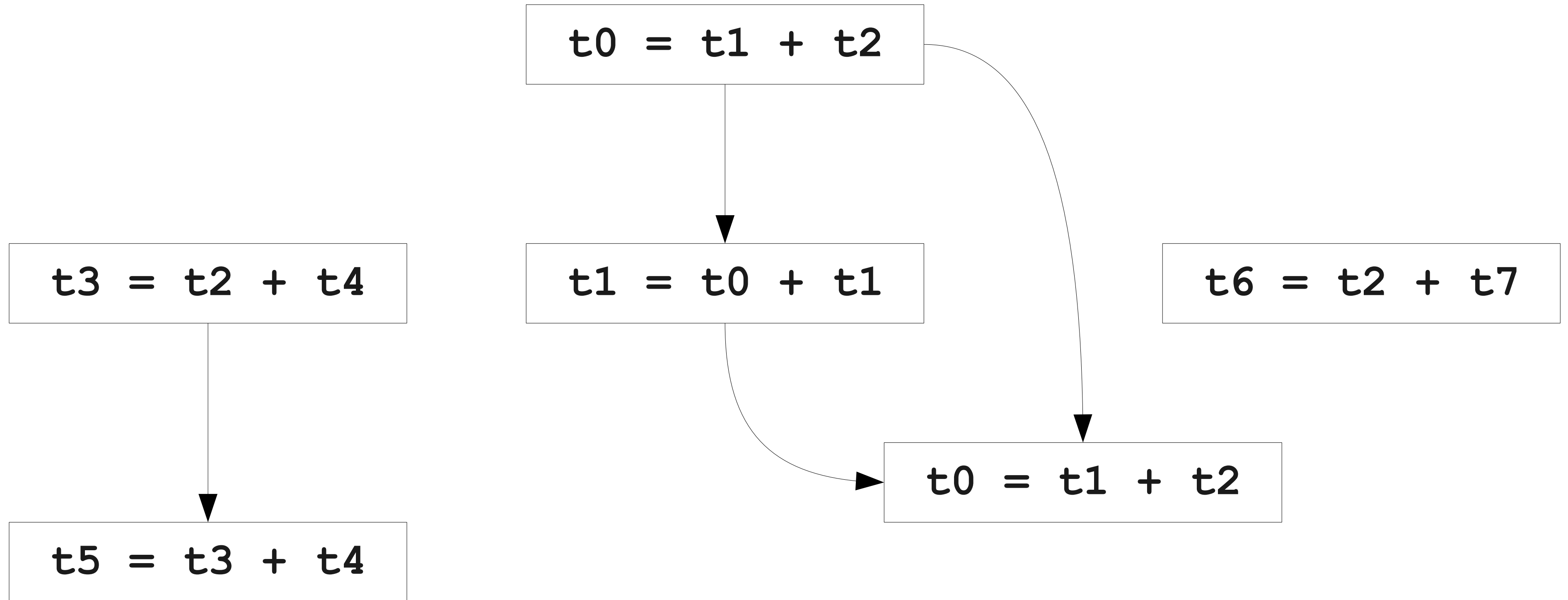
# Finding Data Dependencies



# Finding Data Dependencies



# Finding Data Dependencies

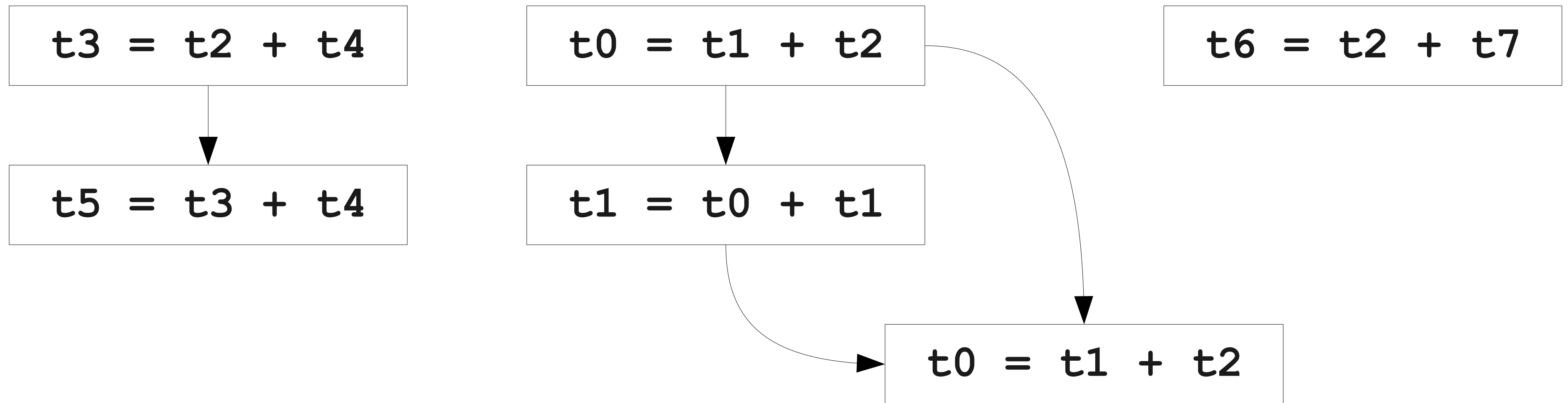


# Data Dependency Graphs

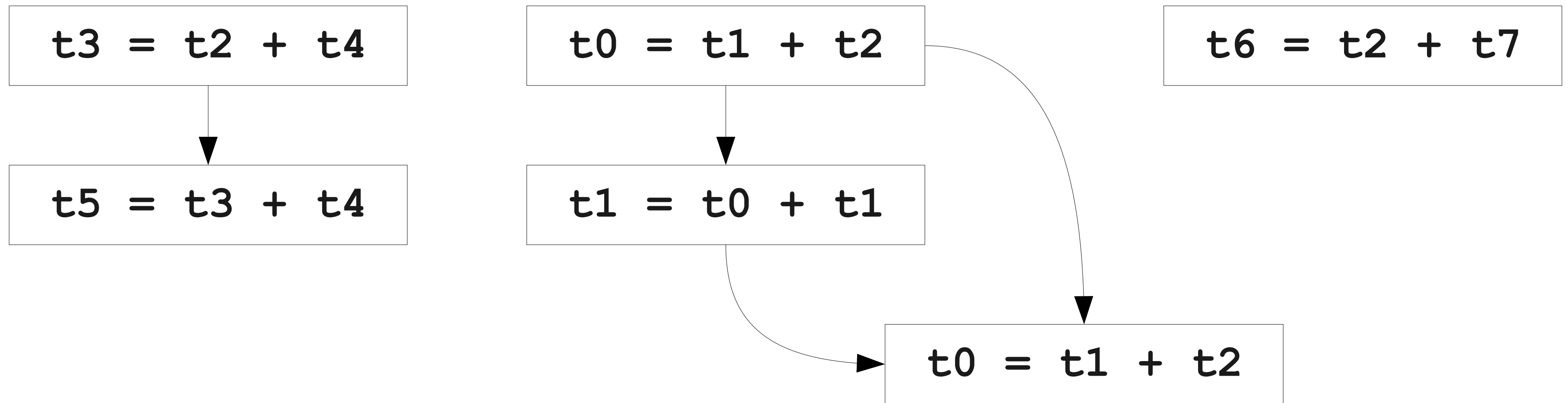
- The graph of the data dependencies in a basic block is called the **data dependency graph**.
- Always a **directed acyclic** graph:
  - **Directed**: One instruction depends on the other.
  - **Acyclic**: No circular dependencies allowed. *(Why?)*
- Can schedule instructions in a basic block in any order as long we never schedule a node before all its parents.
- **Idea**: Do a **topological sort** of the data dependency graph and output instructions in that order.

# Instruction Scheduling

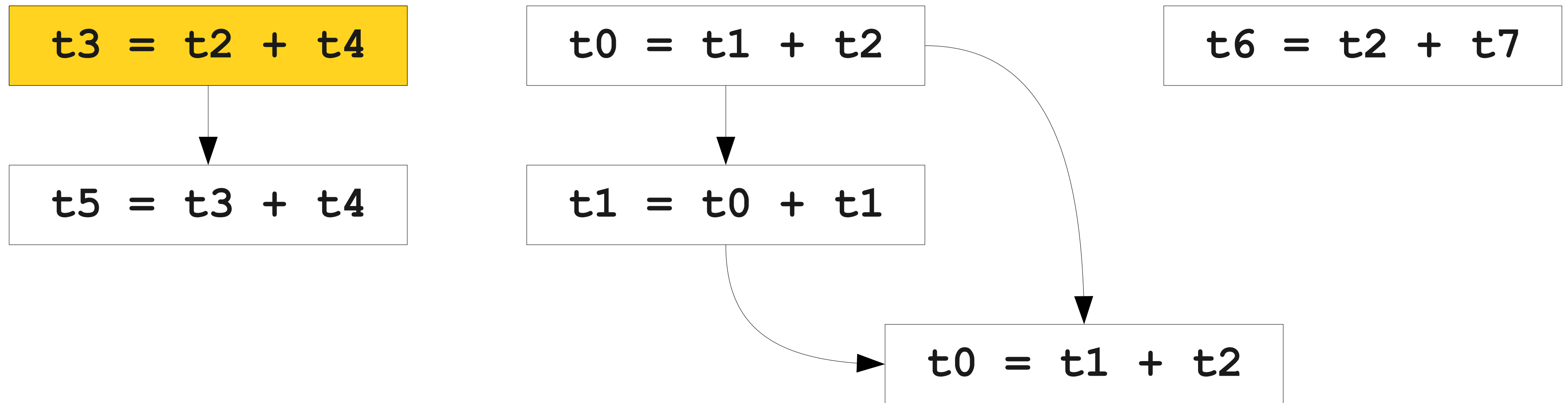
# Instruction Scheduling



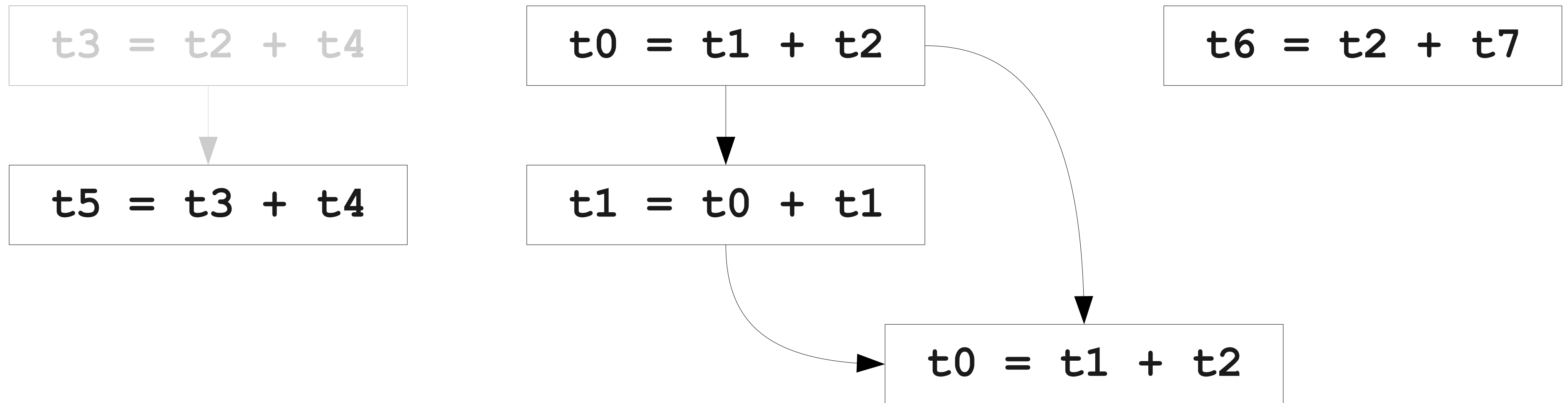
# Instruction Scheduling



# Instruction Scheduling

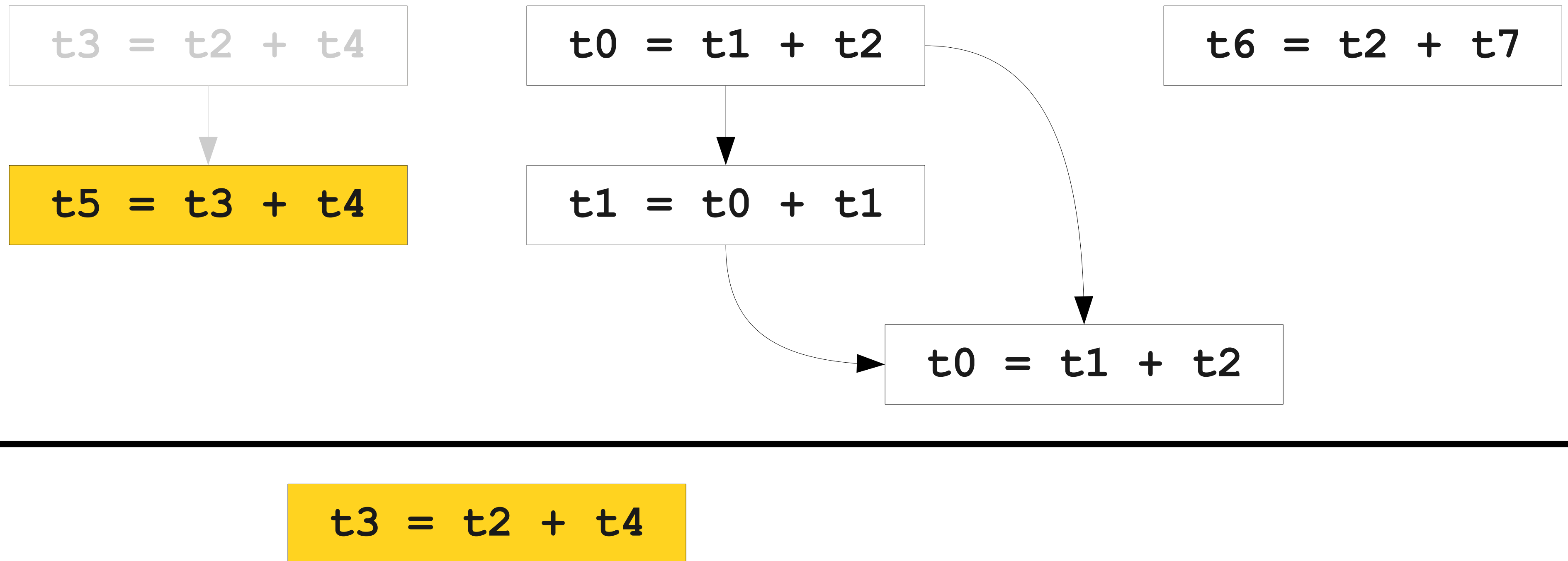


# Instruction Scheduling

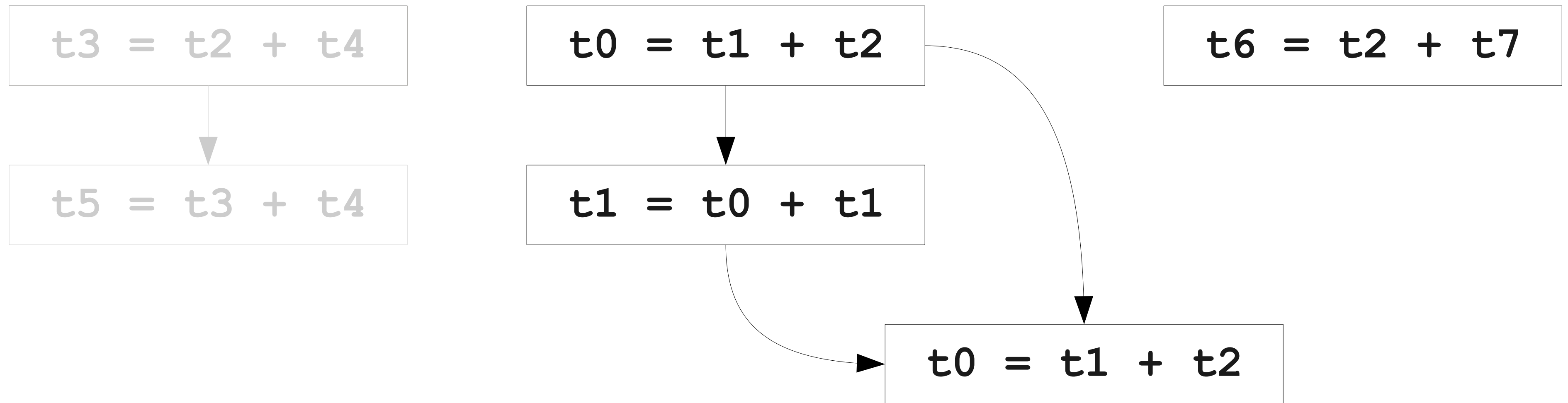



```
t3 = t2 + t4
```

# Instruction Scheduling



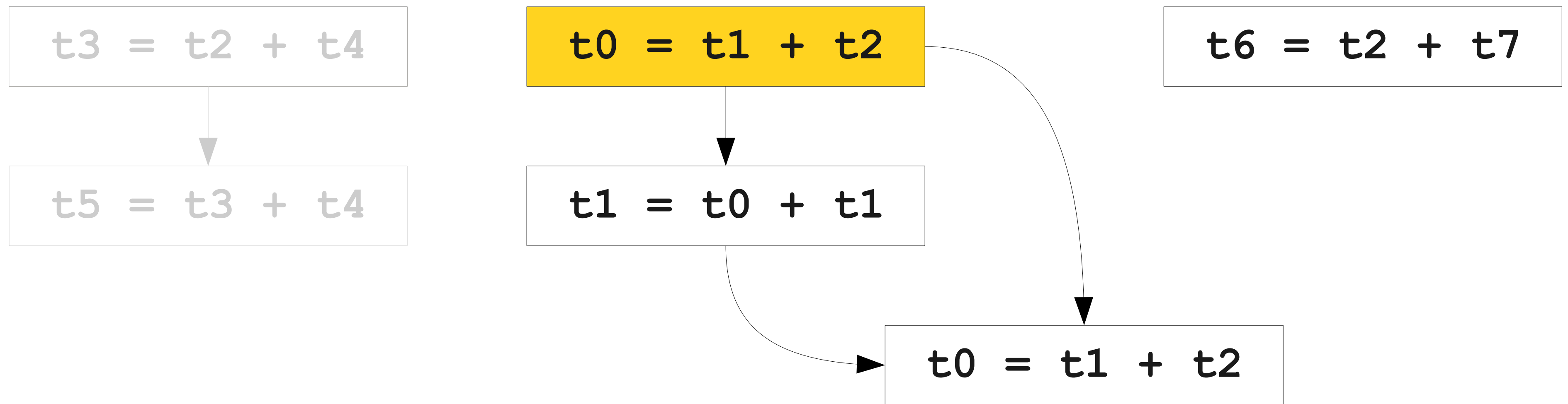
# Instruction Scheduling

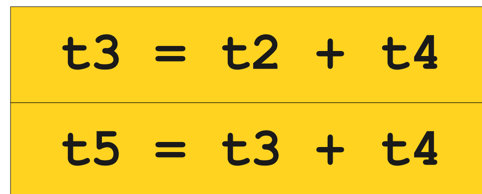



---

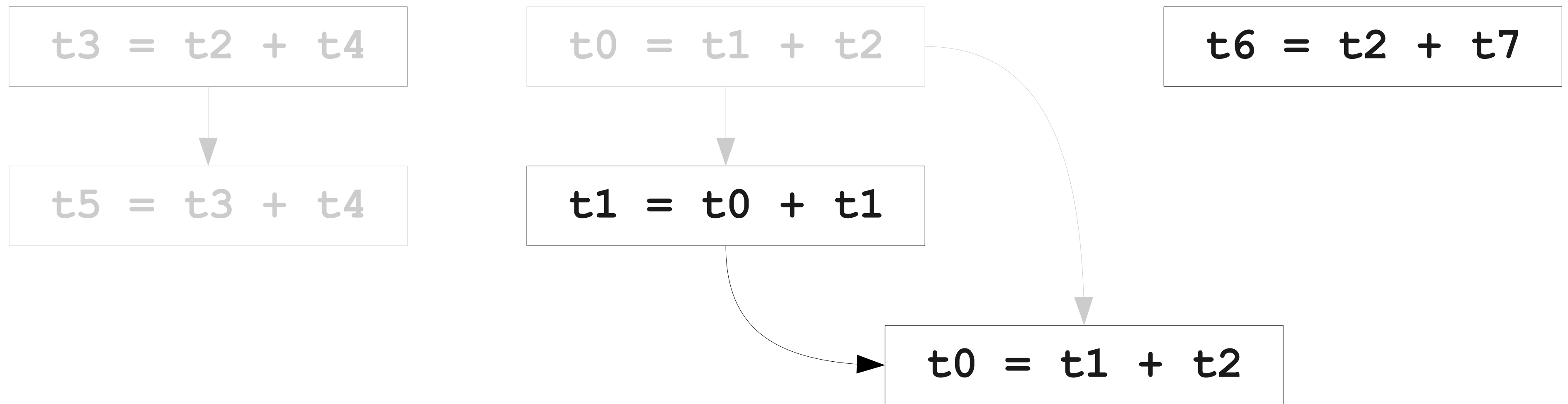
```
t3 = t2 + t4
t5 = t3 + t4
```

# Instruction Scheduling



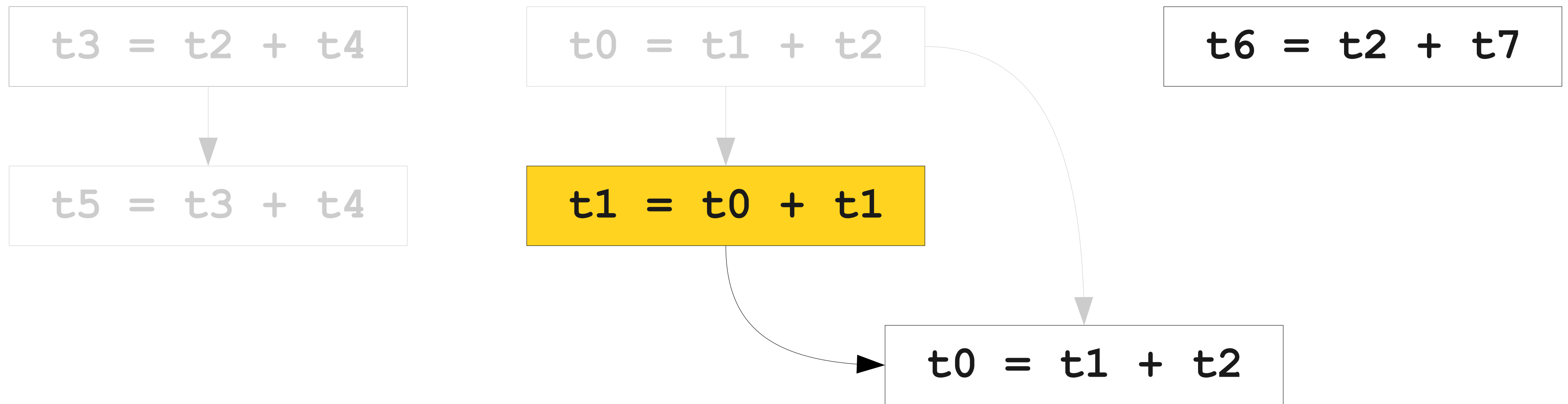
# Instruction Scheduling




---

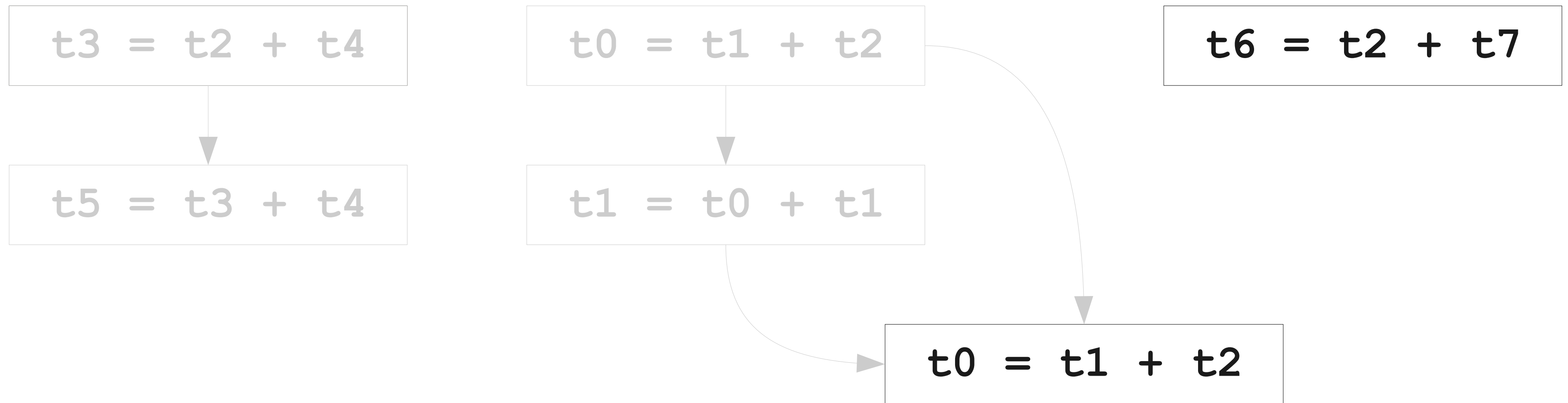
```
t3 = t2 + t4
t5 = t3 + t4
t0 = t1 + t2
```

# Instruction Scheduling



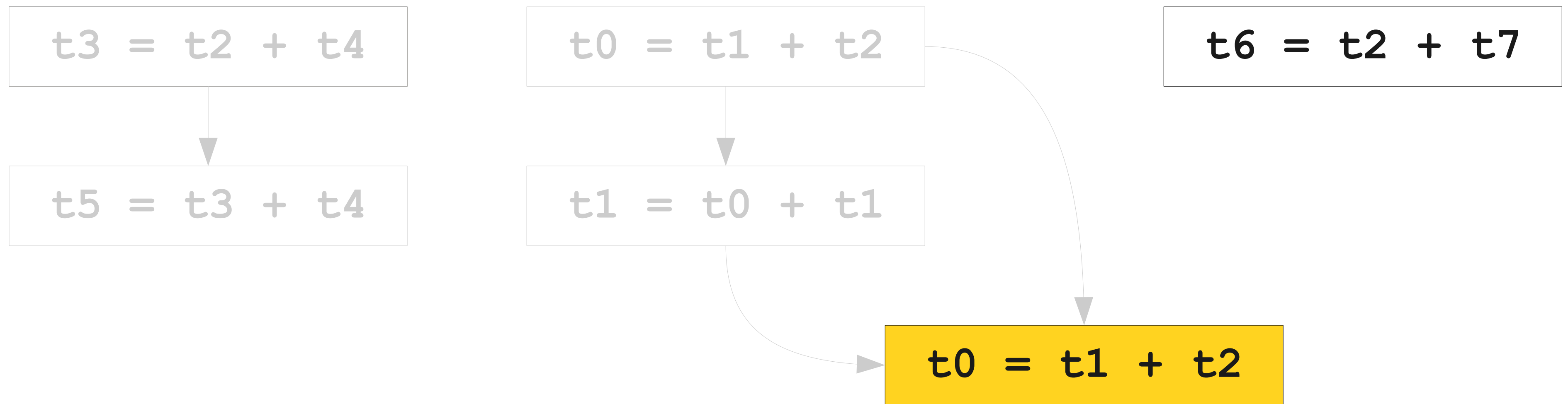

```
t3 = t2 + t4
t5 = t3 + t4
t0 = t1 + t2
```

# Instruction Scheduling



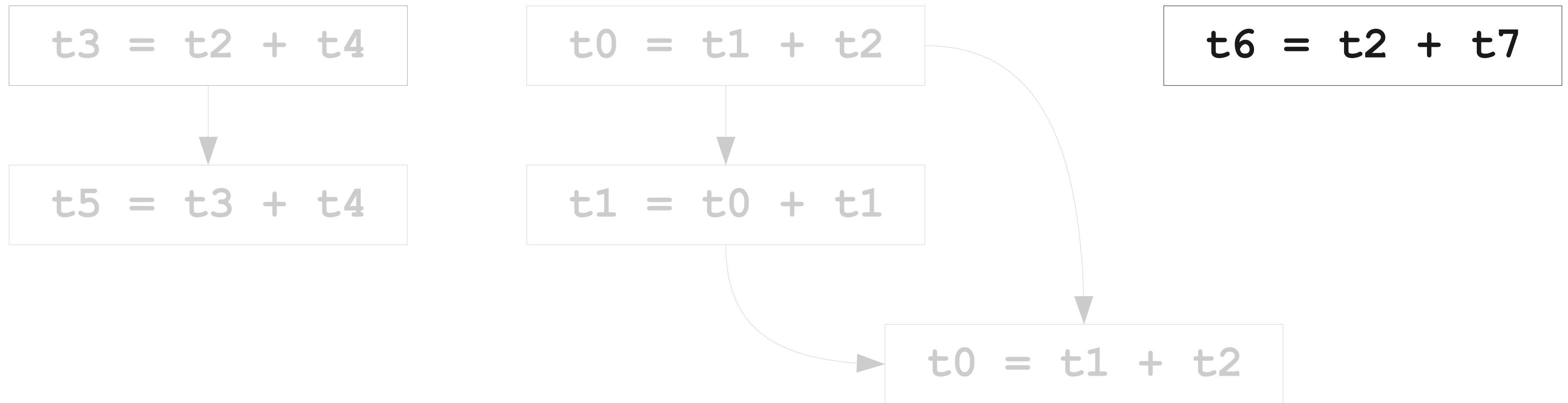

```
t3 = t2 + t4
t5 = t3 + t4
t0 = t1 + t2
t1 = t0 + t1
```

# Instruction Scheduling



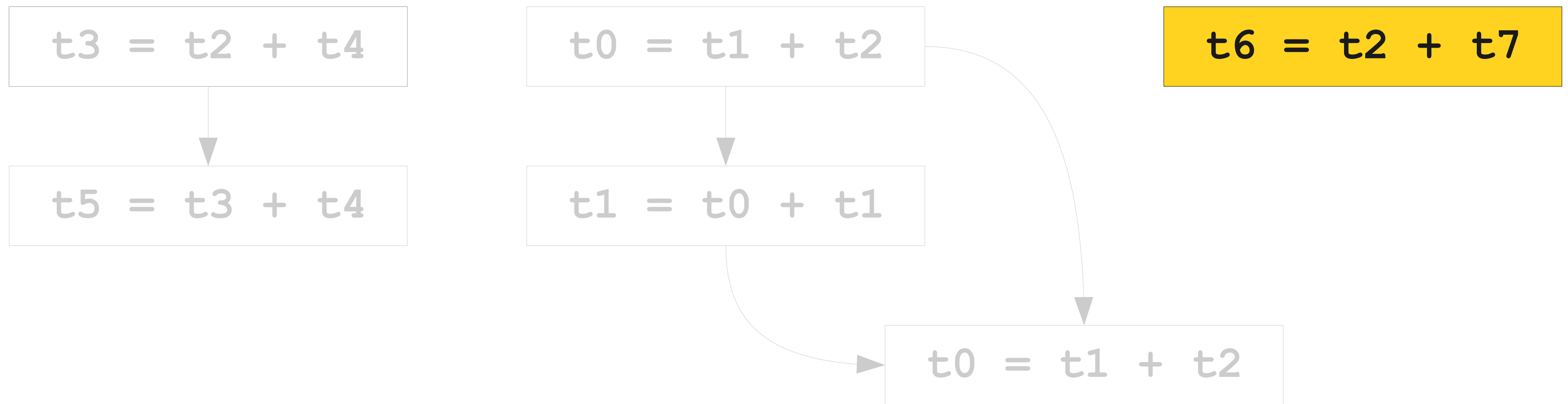

```
t3 = t2 + t4
t5 = t3 + t4
t0 = t1 + t2
t1 = t0 + t1
```

# Instruction Scheduling



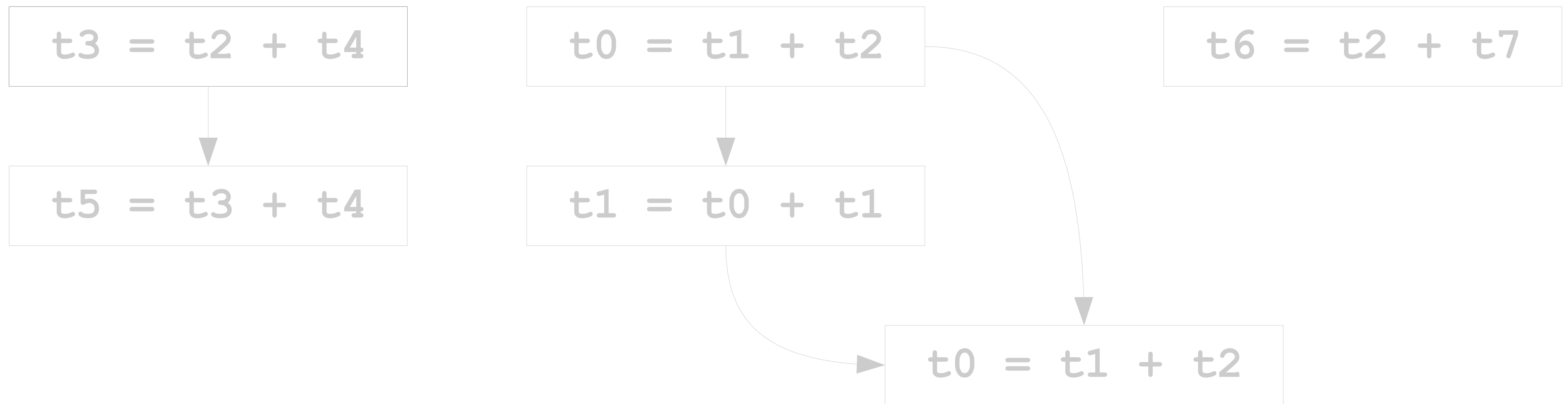

| t3 = t2 + t4 |
|---|
| t5 = t3 + t4 |
| t0 = t1 + t2 |
| t1 = t0 + t1 |
| t0 = t1 + t2 |

# Instruction Scheduling




| t3 = t2 + t4 |
| --- |
| t5 = t3 + t4 |
| t0 = t1 + t2 |
| t1 = t0 + t1 |
| t0 = t1 + t2 |

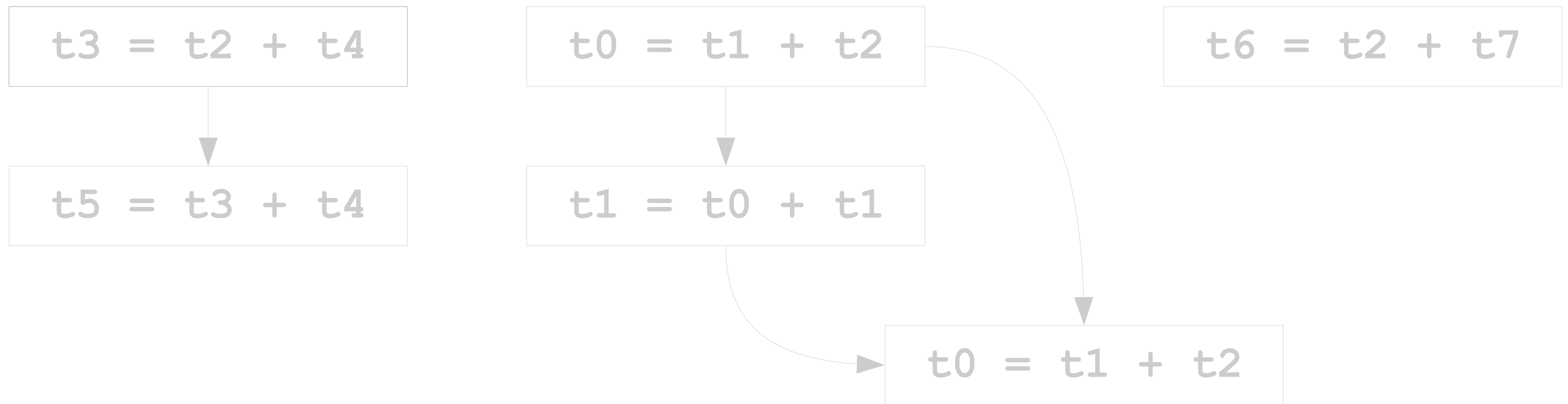
# Instruction Scheduling




---

| t3 = t2 + t4 |
| --- |
| t5 = t3 + t4 |
| t0 = t1 + t2 |
| t1 = t0 + t1 |
| t0 = t1 + t2 |
| t6 = t2 + t7 |

# Instruction Scheduling

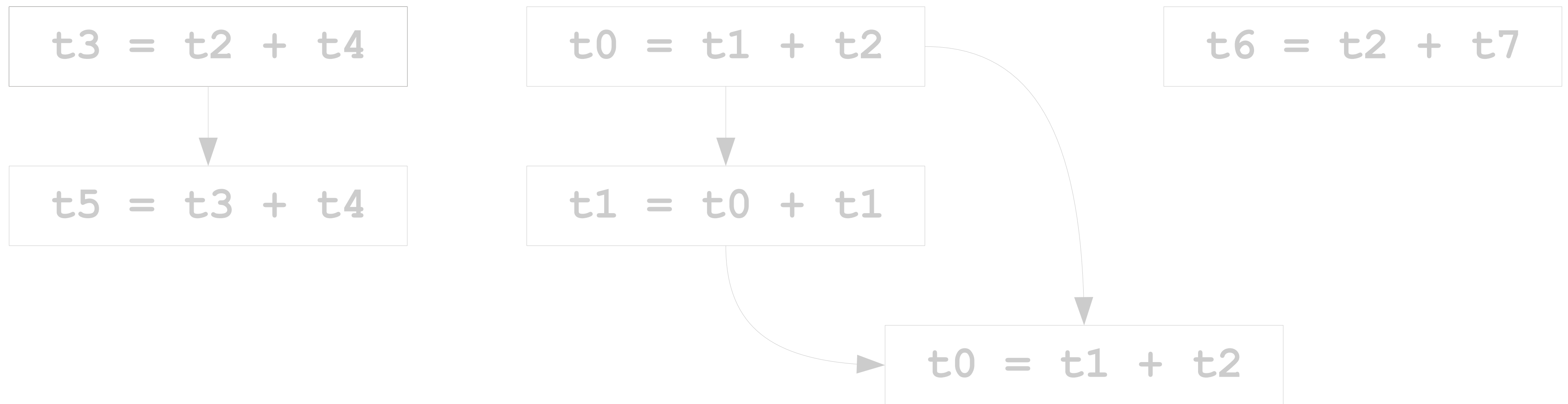



---

```
t3 = t2 + t4
t5 = t3 + t4
t0 = t1 + t2
t1 = t0 + t1
t0 = t1 + t2
t6 = t2 + t7
```

Is this a legal schedule?

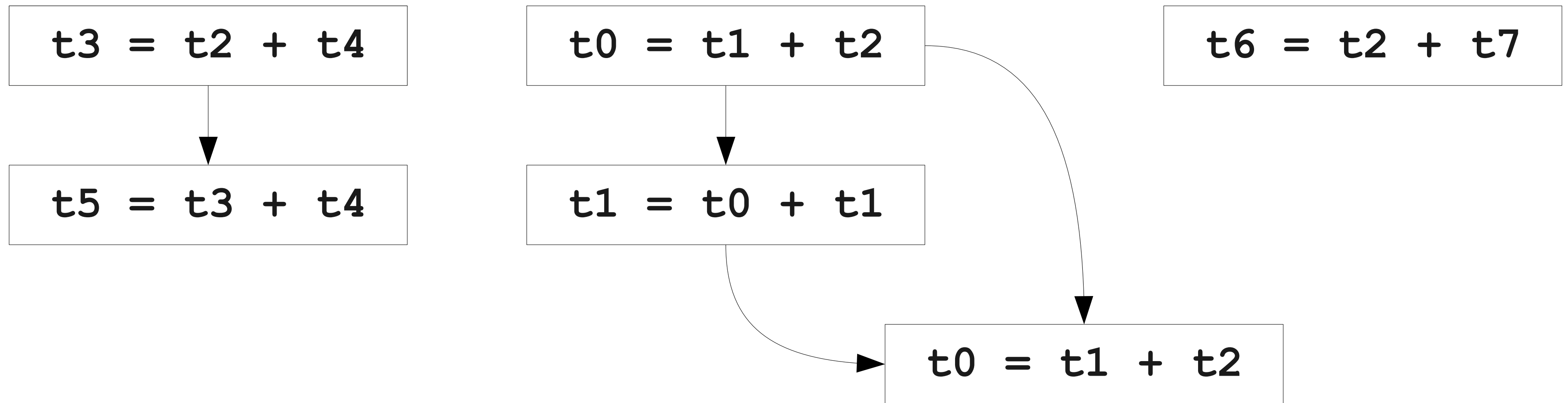
# Instruction Scheduling




```
t3 = t2 + t4
t5 = t3 + t4
t0 = t1 + t2
t1 = t0 + t1
t0 = t1 + t2
t6 = t2 + t7
```

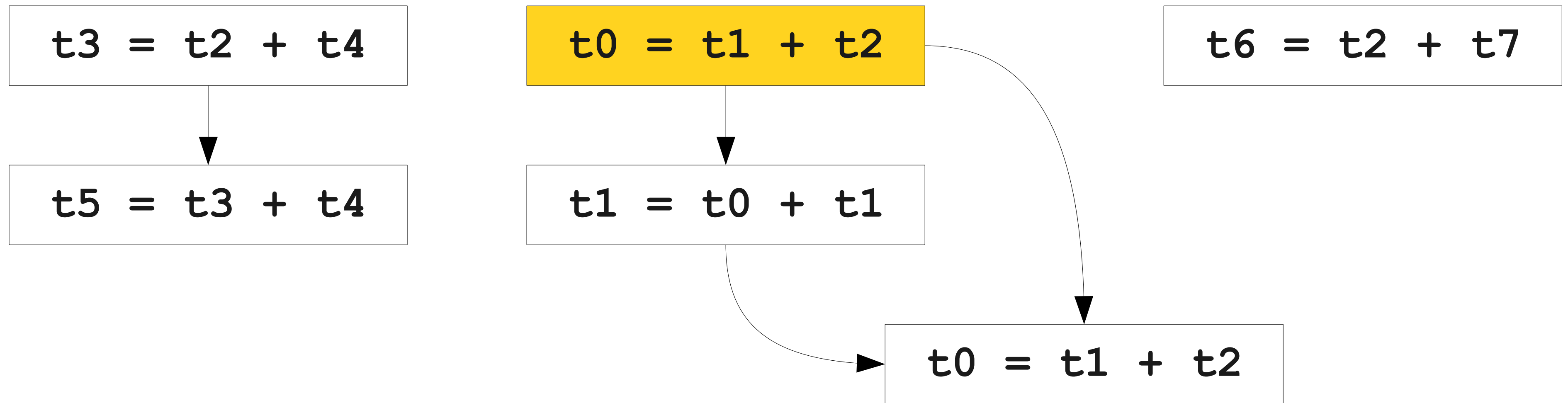
Is this a good schedule?

# Instruction Scheduling



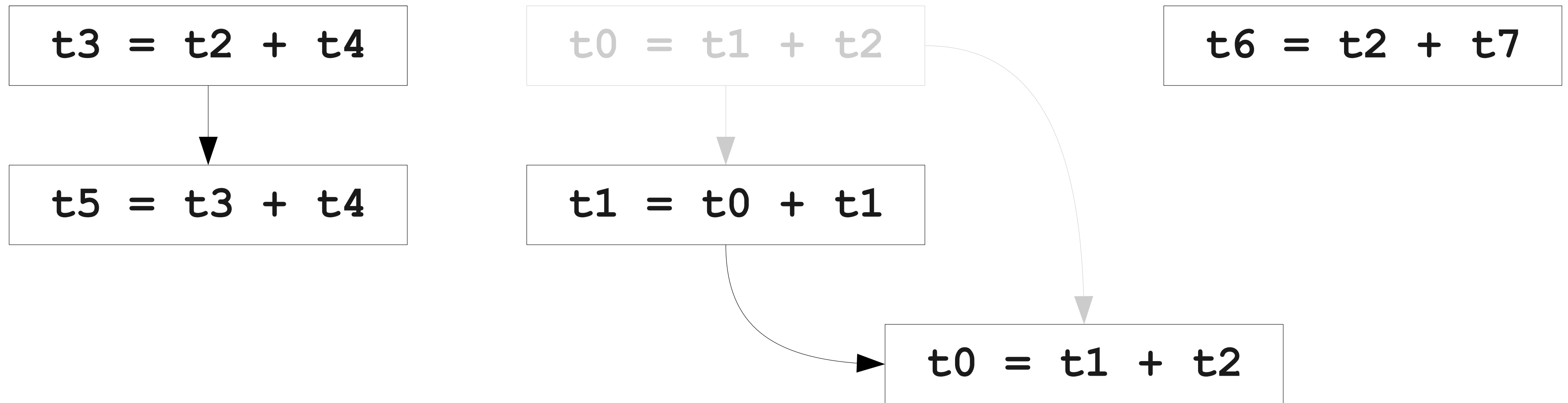

```
t3 = t2 + t4
t5 = t3 + t4
t0 = t1 + t2
t1 = t0 + t1
t0 = t1 + t2
t6 = t2 + t7
```

# Instruction Scheduling




```
t3 = t2 + t4
t5 = t3 + t4
t0 = t1 + t2
t1 = t0 + t1
t0 = t1 + t2
t6 = t2 + t7
```

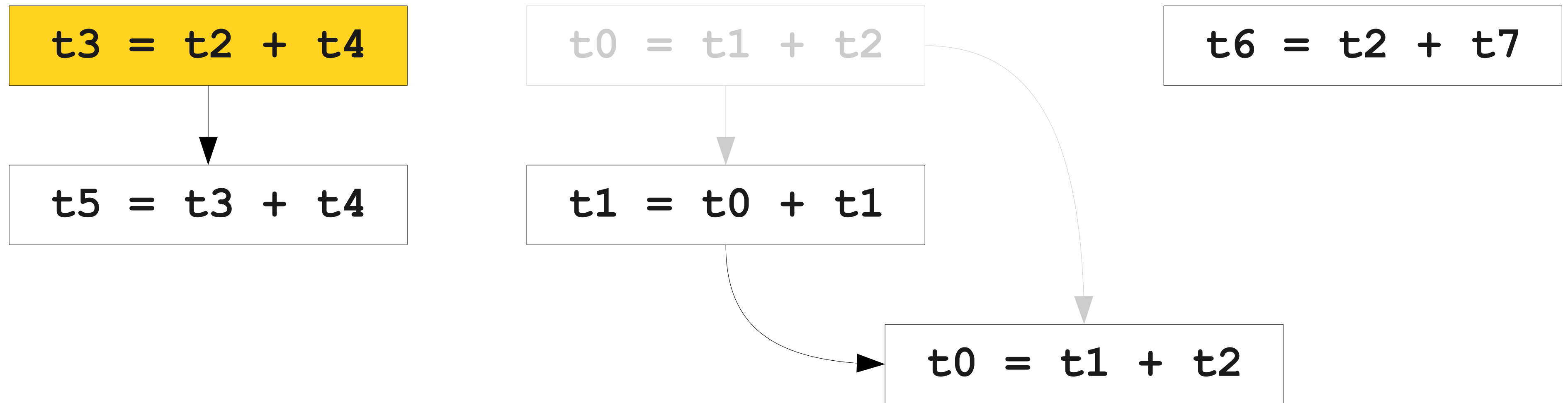
# Instruction Scheduling




```
t3 = t2 + t4
t5 = t3 + t4
t0 = t1 + t2
t1 = t0 + t1
t0 = t1 + t2
t6 = t2 + t7
```

```
t0 = t1 + t2
```

# Instruction Scheduling

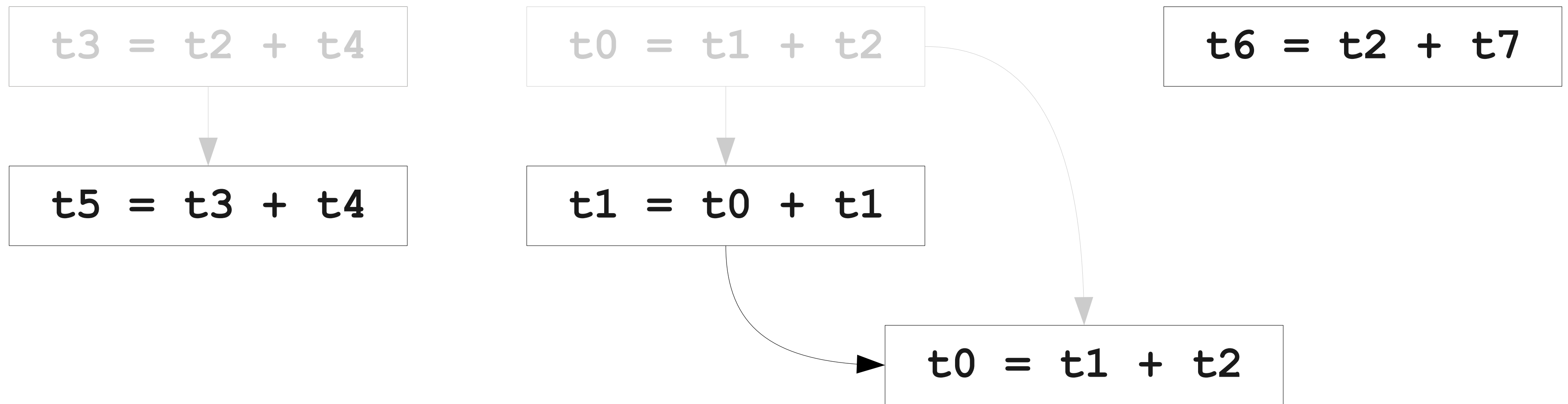



---

```
t3 = t2 + t4
t5 = t3 + t4
t0 = t1 + t2
t1 = t0 + t1
t0 = t1 + t2
t6 = t2 + t7
```

```
t0 = t1 + t2
```

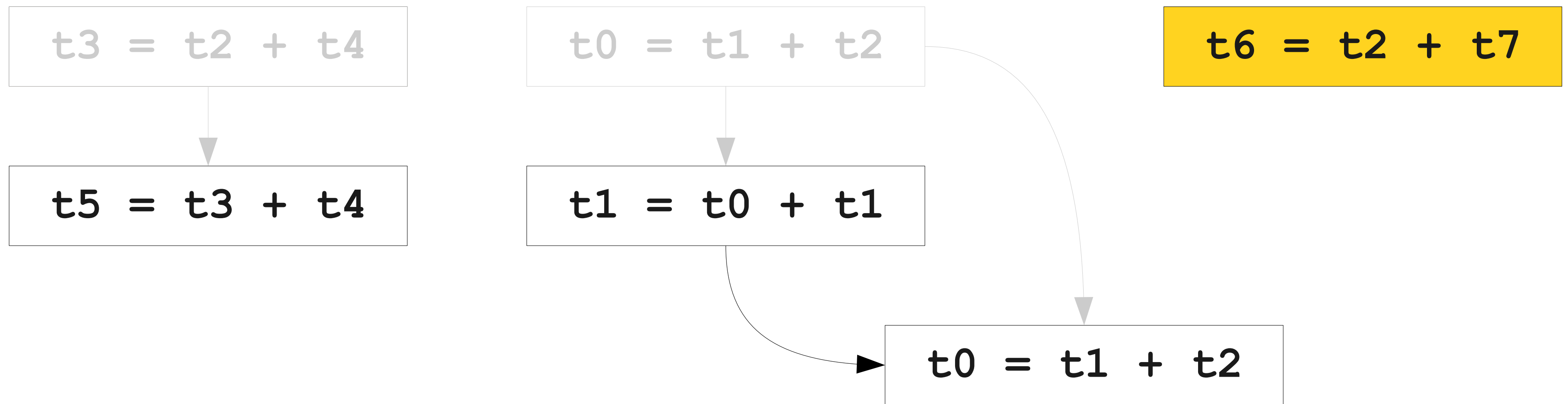
# Instruction Scheduling




```
t3 = t2 + t4
t5 = t3 + t4
t0 = t1 + t2
t1 = t0 + t1
t0 = t1 + t2
t6 = t2 + t7
```

```
t0 = t1 + t2
t3 = t2 + t4
```

# Instruction Scheduling

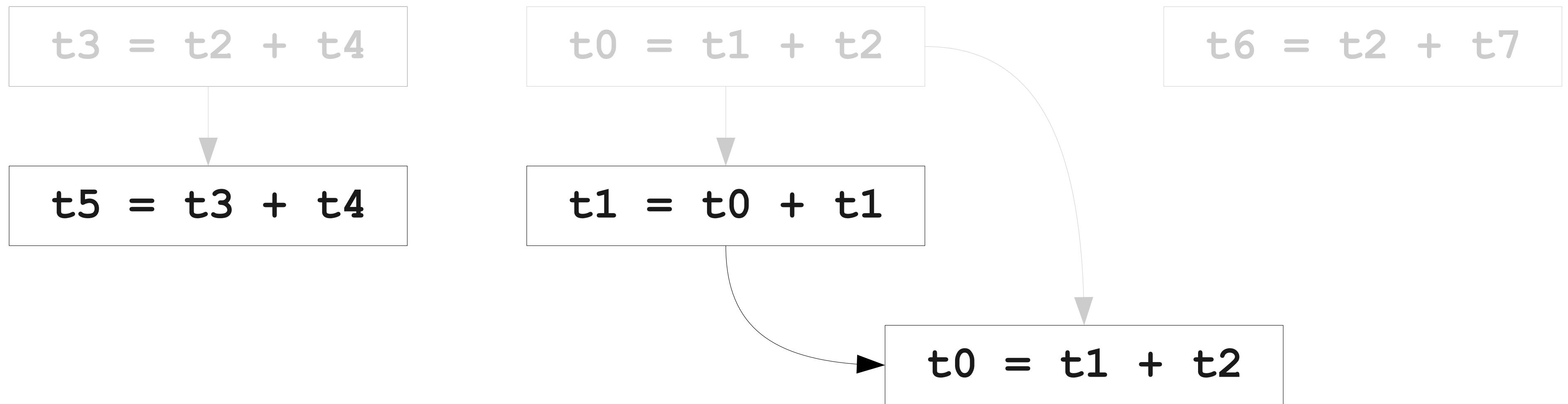



---

```
t3 = t2 + t4
t5 = t3 + t4
t0 = t1 + t2
t1 = t0 + t1
t0 = t1 + t2
t6 = t2 + t7
```

```
t0 = t1 + t2
t3 = t2 + t4
```

# Instruction Scheduling

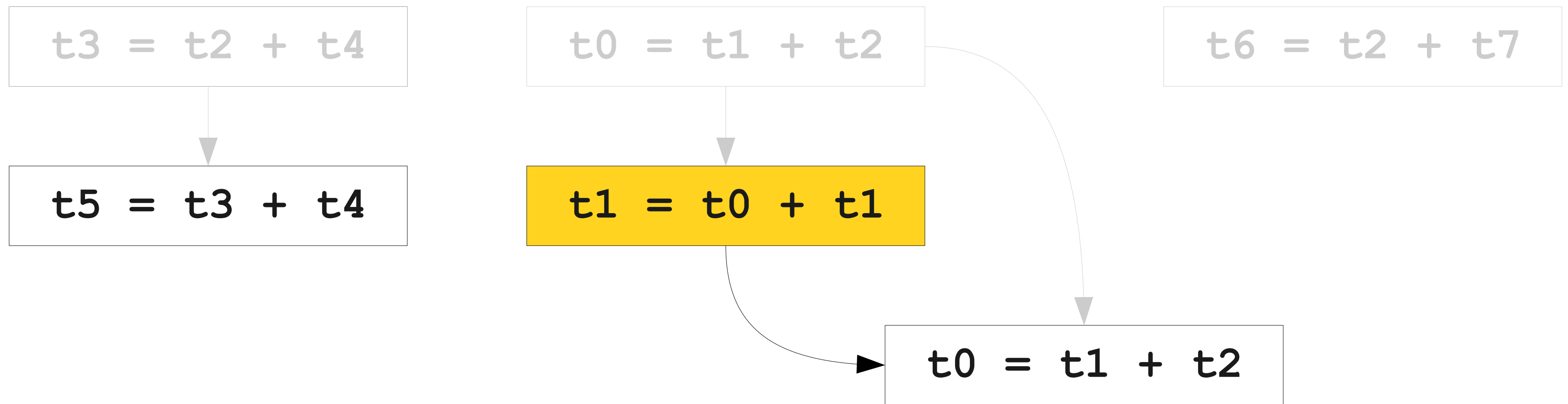



---

```
t3 = t2 + t4
t5 = t3 + t4
t0 = t1 + t2
t1 = t0 + t1
t0 = t1 + t2
t6 = t2 + t7
```

```
t0 = t1 + t2
t3 = t2 + t4
t6 = t2 + t7
```

# Instruction Scheduling

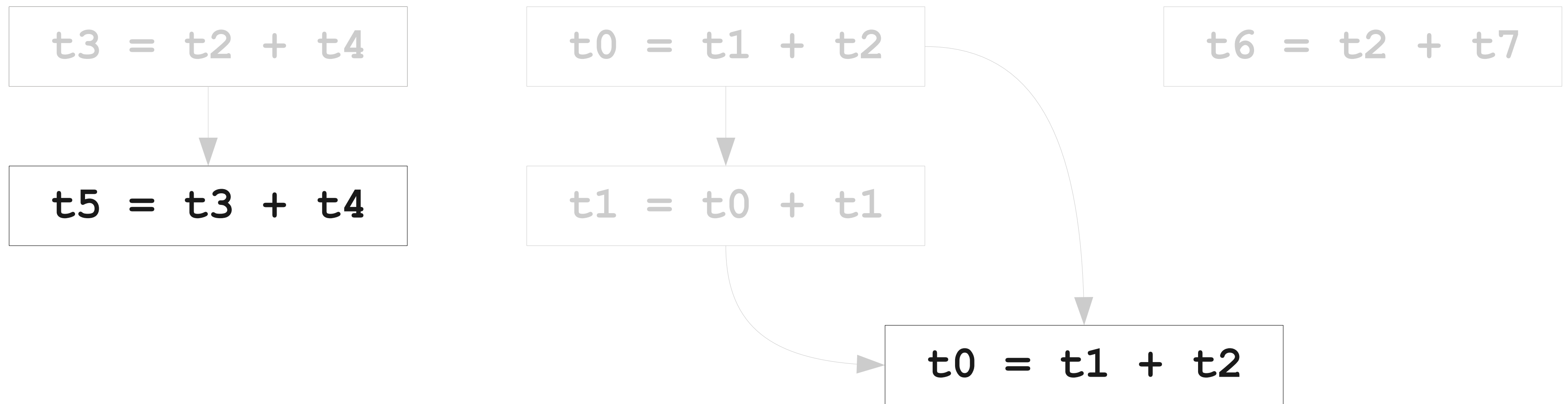



---

```
t3 = t2 + t4
t5 = t3 + t4
t0 = t1 + t2
t1 = t0 + t1
t0 = t1 + t2
t6 = t2 + t7
```

```
t0 = t1 + t2
t3 = t2 + t4
t6 = t2 + t7
```

# Instruction Scheduling

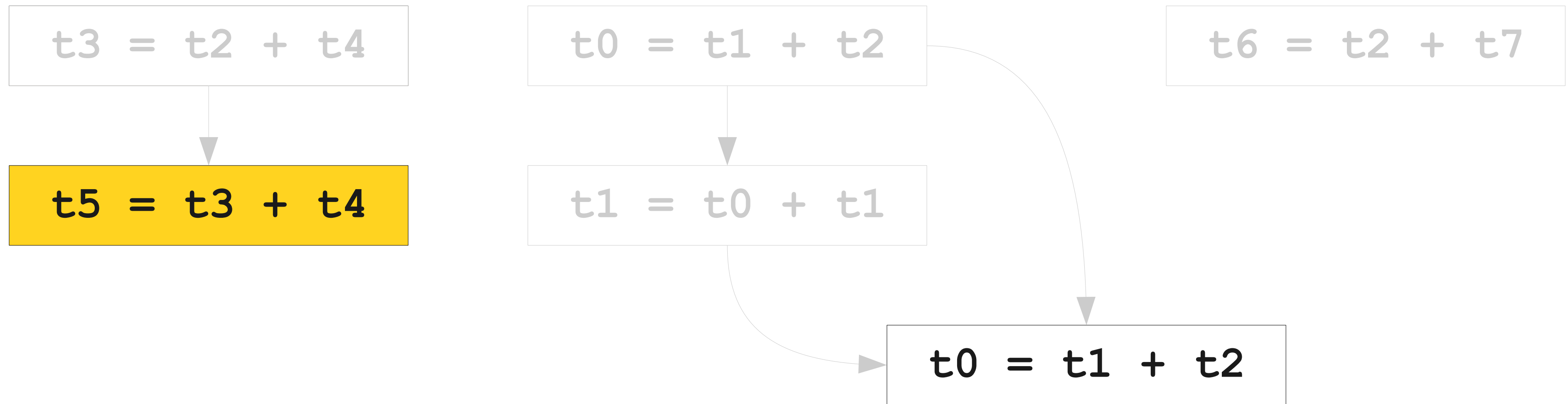



---

```
t3 = t2 + t4
t5 = t3 + t4
t0 = t1 + t2
t1 = t0 + t1
t0 = t1 + t2
t6 = t2 + t7
```

```
t0 = t1 + t2
t3 = t2 + t4
t6 = t2 + t7
t1 = t0 + t1
```

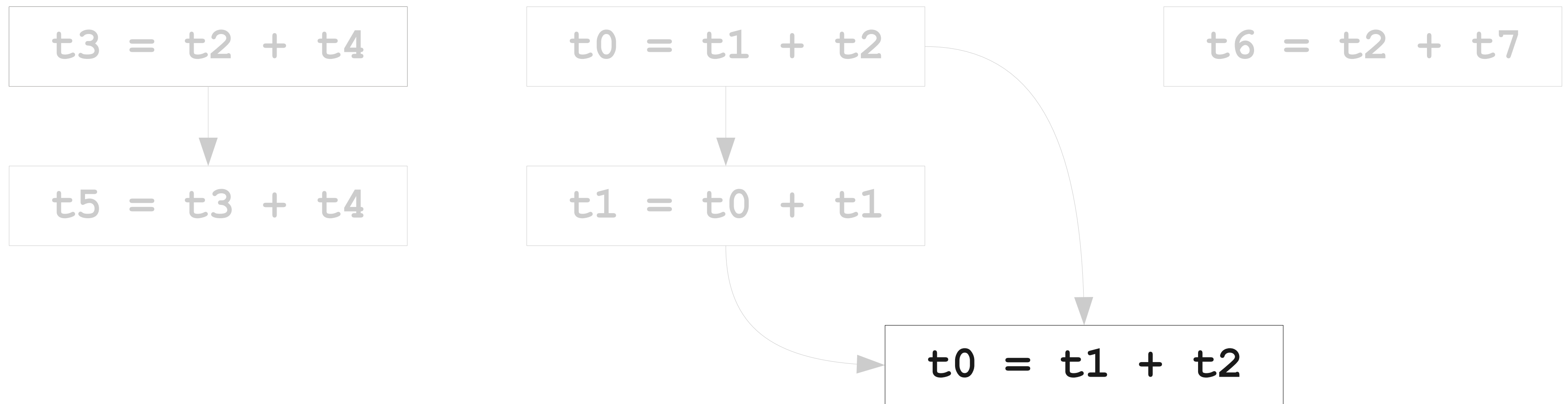
# Instruction Scheduling




```
t3 = t2 + t4
t5 = t3 + t4
t0 = t1 + t2
t1 = t0 + t1
t0 = t1 + t2
t6 = t2 + t7
```

```
t0 = t1 + t2
t3 = t2 + t4
t6 = t2 + t7
t1 = t0 + t1
```

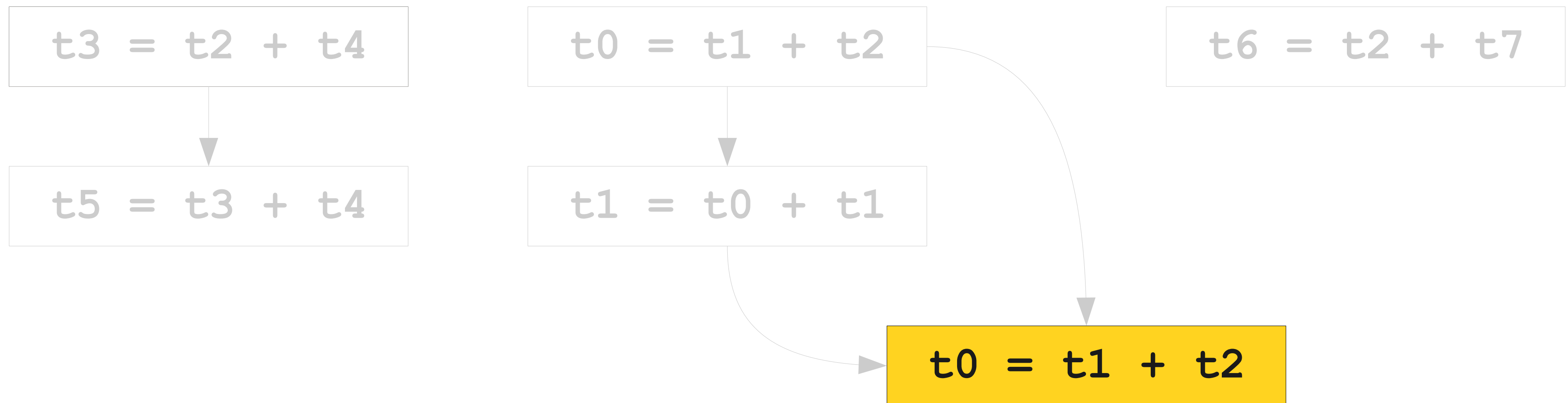
# Instruction Scheduling




```
t3 = t2 + t4
t5 = t3 + t4
t0 = t1 + t2
t1 = t0 + t1
t0 = t1 + t2
t6 = t2 + t7
```

```
t0 = t1 + t2
t3 = t2 + t4
t6 = t2 + t7
t1 = t0 + t1
t5 = t3 + t4
```

# Instruction Scheduling

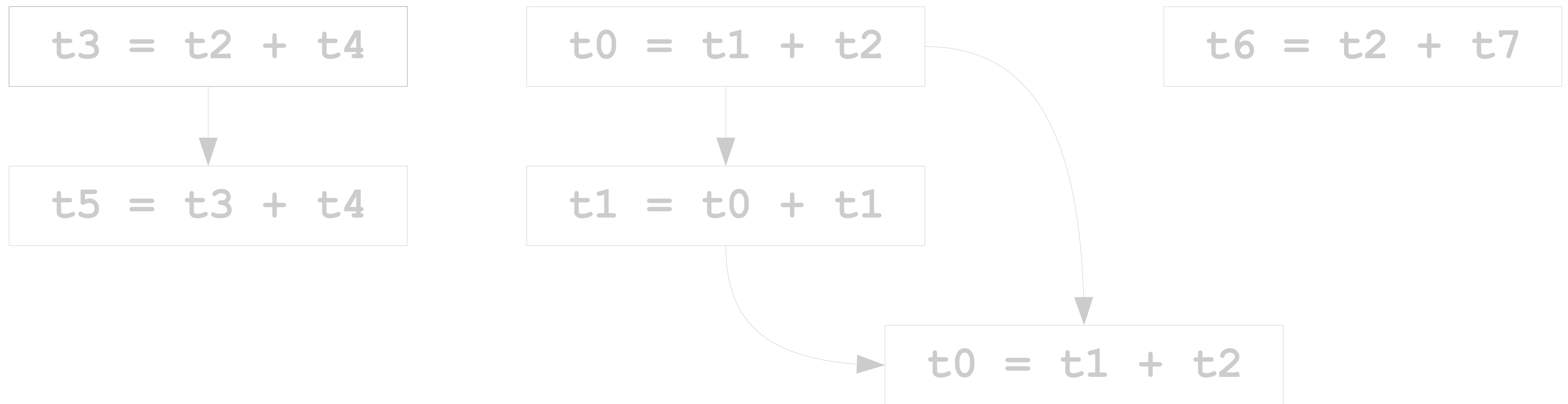



---

```
t3 = t2 + t4
t5 = t3 + t4
t0 = t1 + t2
t1 = t0 + t1
t0 = t1 + t2
t6 = t2 + t7
```

```
t0 = t1 + t2
t3 = t2 + t4
t6 = t2 + t7
t1 = t0 + t1
t5 = t3 + t4
```

# Instruction Scheduling



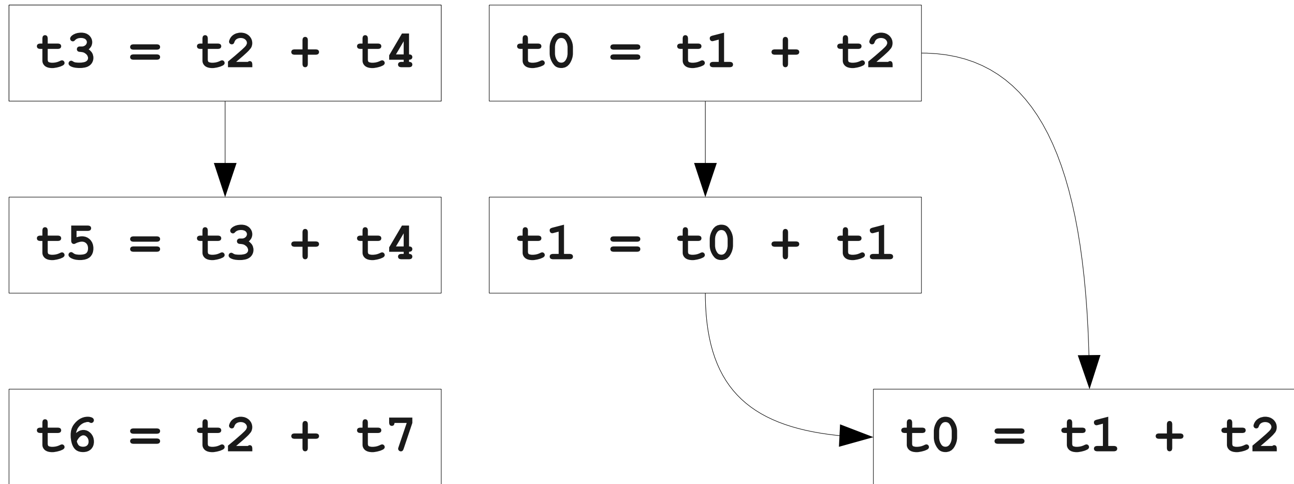

```
t3 = t2 + t4
t5 = t3 + t4
t0 = t1 + t2
t1 = t0 + t1
t0 = t1 + t2
t6 = t2 + t7
```

```
t0 = t1 + t2
t3 = t2 + t4
t6 = t2 + t7
t1 = t0 + t1
t5 = t3 + t4
t0 = t1 + t2
```

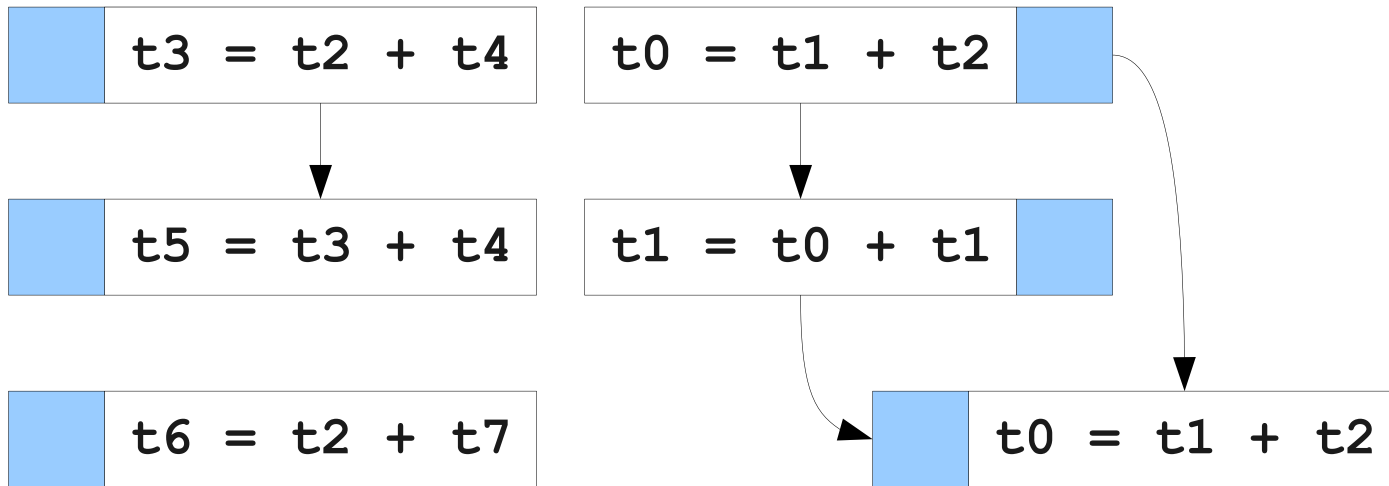
# One Small Problem

- There can be many valid topological orderings of a data dependency graph.
- How do we pick one that works well with the pipeline?
- In general, finding the fastest instruction schedule is known to be **NP-hard**.
  - Don't expect a polynomial-time algorithm anytime soon!
- Heuristics are used in practice:
  - Schedule instructions that can run to completion without interference before instructions that cause interference.
  - Schedule instructions with more dependents before instructions with fewer dependents.

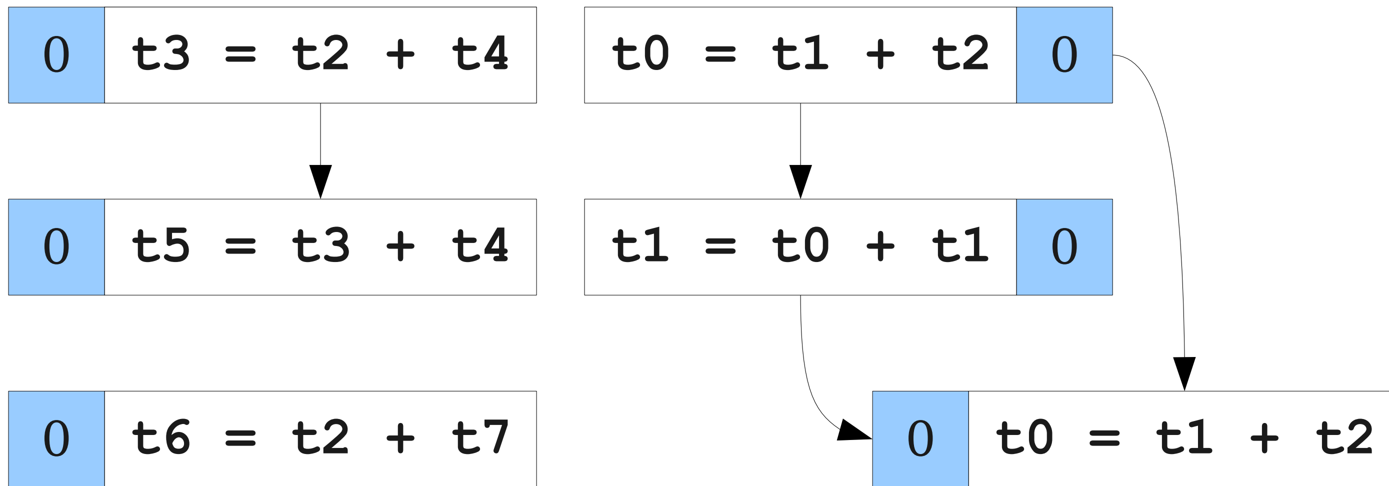
# Instruction Scheduling



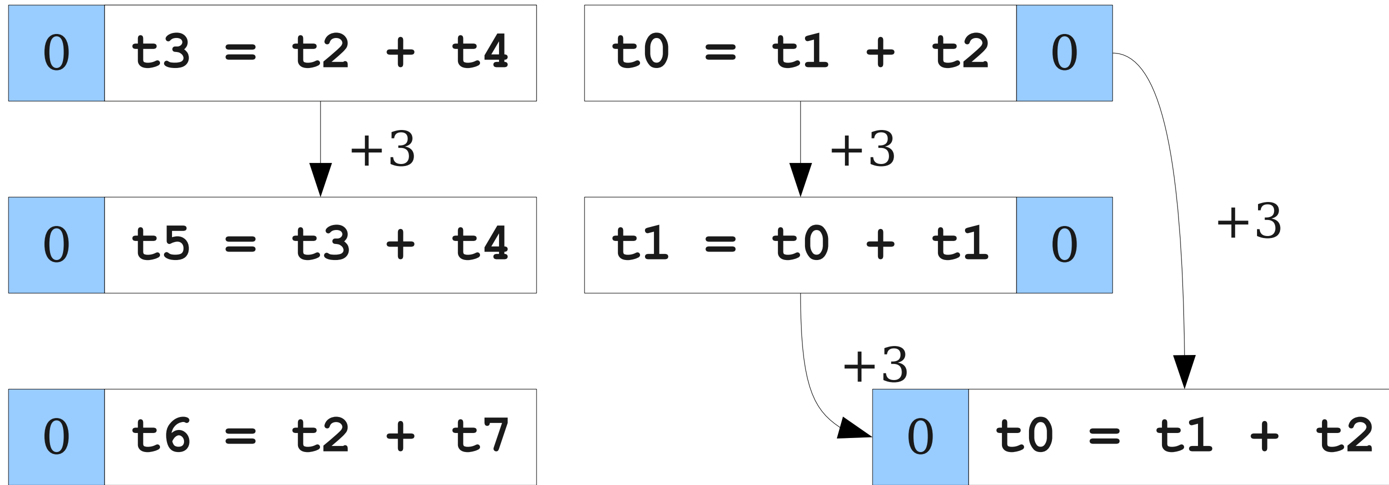
# Instruction Scheduling



# Instruction Scheduling



# Instruction Scheduling



# Instruction Scheduling

0 `t3 = t2 + t4`

+3

0 `t5 = t3 + t4`

0 `t6 = t2 + t7`

`t0 = t1 + t2` 0

+3

`t1 = t0 + t1` 0

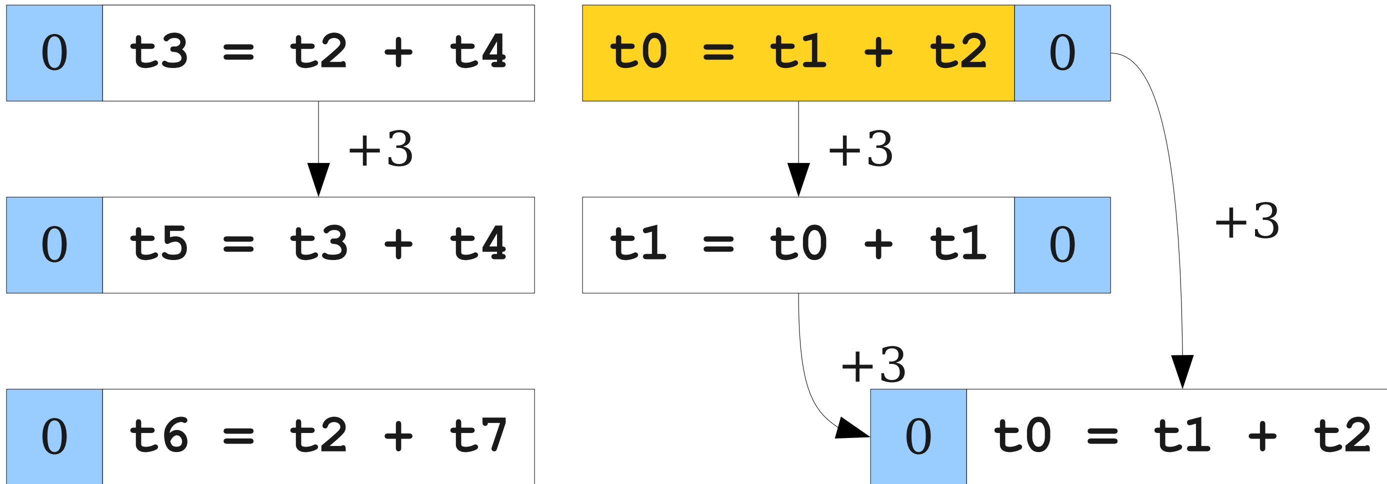
+3

+3

0 `t0 = t1 + t2`

| ID | RR | ALU | RW |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Instruction Scheduling




```
t0 = t1 + t2
```

| ID | RR | ALU | RW |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Instruction Scheduling

| 0 | t3 = t2 + t4 |
|---|---|

+3

| 0 | t5 = t3 + t4 |
|---|---|

| 0 | t6 = t2 + t7 |
|---|---|

| t0 = t1 + t2 | 0 |
|---|---|

+3

| t1 = t0 + t1 | 3 |
|---|---|

+3

+3

| 3 | t0 = t1 + t2 |
|---|---|

---

```
t0 = t1 + t2
```

| ID | RR | ALU | RW |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Instruction Scheduling



| ID | RR | ALU | RW |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Instruction Scheduling

| 0 | t3 = t2 + t4 |
|---|---|

+3

| 0 | t5 = t3 + t4 |
|---|---|

| 0 | t6 = t2 + t7 |
|---|---|

t0 = t1 + t2

| t1 = t0 + t1 | 3 |
|---|---|

+3

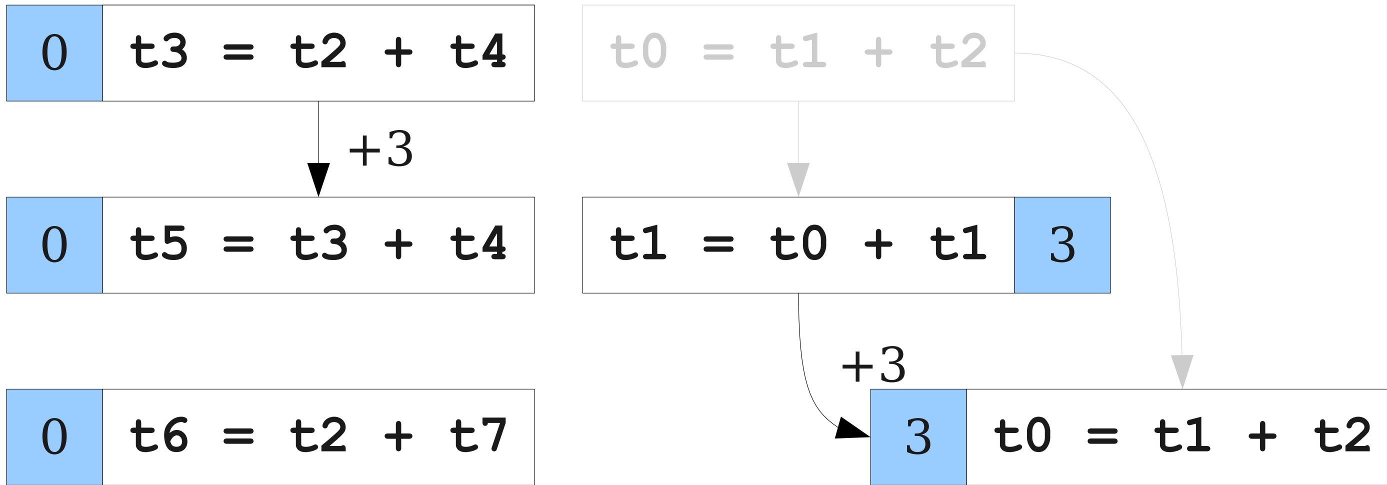
| 3 | t0 = t1 + t2 |
|---|---|

---

t0 = t1 + t2

| ID | RR | ALU | RW |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Instruction Scheduling




`t0 = t1 + t2`

| ID | RR | ALU | RW |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Instruction Scheduling

| | |
|---|---|
| 0 | `t3 = t2 + t4` |

+3

| | |
|---|---|
| 0 | `t5 = t3 + t4` |

| | |
|---|---|
| 0 | `t6 = t2 + t7` |

`t0 = t1 + t2`

| | |
|---|---|
| `t1 = t0 + t1` | 3 |

+3

| | |
|---|---|
| 3 | `t0 = t1 + t2` |

---

```
t0 = t1 + t2
t3 = t2 + t4
```

| ID | RR | ALU | RW |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Instruction Scheduling

0 `t3 = t2 + t4`

+3

4 `t5 = t3 + t4`

0 `t6 = t2 + t7`

`t0 = t1 + t2`

`t1 = t0 + t1` 3

+3

3 `t0 = t1 + t2`

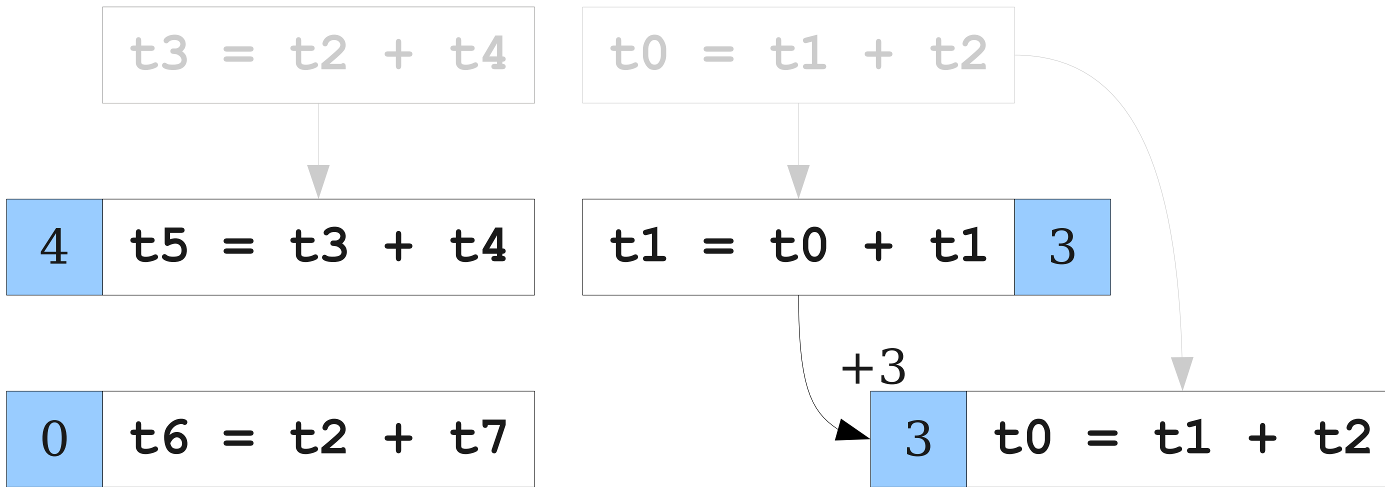
---

`t0 = t1 + t2`

`t3 = t2 + t4`

| ID | RR | ALU | RW |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Instruction Scheduling




```
t0 = t1 + t2
t3 = t2 + t4
```

| ID | RR | ALU | RW |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Instruction Scheduling

t3 = t2 + t4

t0 = t1 + t2

4 t5 = t3 + t4

t1 = t0 + t1 3

+3

0 t6 = t2 + t7

3 t0 = t1 + t2

t0 = t1 + t2

t3 = t2 + t4

| ID | RR | ALU | RW |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Instruction Scheduling

t3 = t2 + t4

t0 = t1 + t2

4 t5 = t3 + t4

t1 = t0 + t1 3

+3

0 t6 = t2 + t7

3 t0 = t1 + t2

t0 = t1 + t2

t3 = t2 + t4

| ID | RR | ALU | RW |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Instruction Scheduling

t3 = t2 + t4

t0 = t1 + t2

4 | t5 = t3 + t4

t1 = t0 + t1 | 3

+3

0 | t6 = t2 + t7

3 | t0 = t1 + t2

---

t0 = t1 + t2

t3 = t2 + t4

t6 = t2 + t7

| ID | RR | ALU | RW |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Instruction Scheduling

# Instruction Scheduling

```
t3 = t2 + t4        t0 = t1 + t2

4 | t5 = t3 + t4    t1 = t0 + t1 | 3

                         +3
t6 = t2 + t7             3 | t0 = t1 + t2
```

```
t0 = t1 + t2
t3 = t2 + t4
t6 = t2 + t7
```

| ID | RR | ALU | RW |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Instruction Scheduling

t3 = t2 + t4

t0 = t1 + t2

4 | t5 = t3 + t4

t1 = t0 + t1 | 3

+3

3 | t0 = t1 + t2

t6 = t2 + t7

---

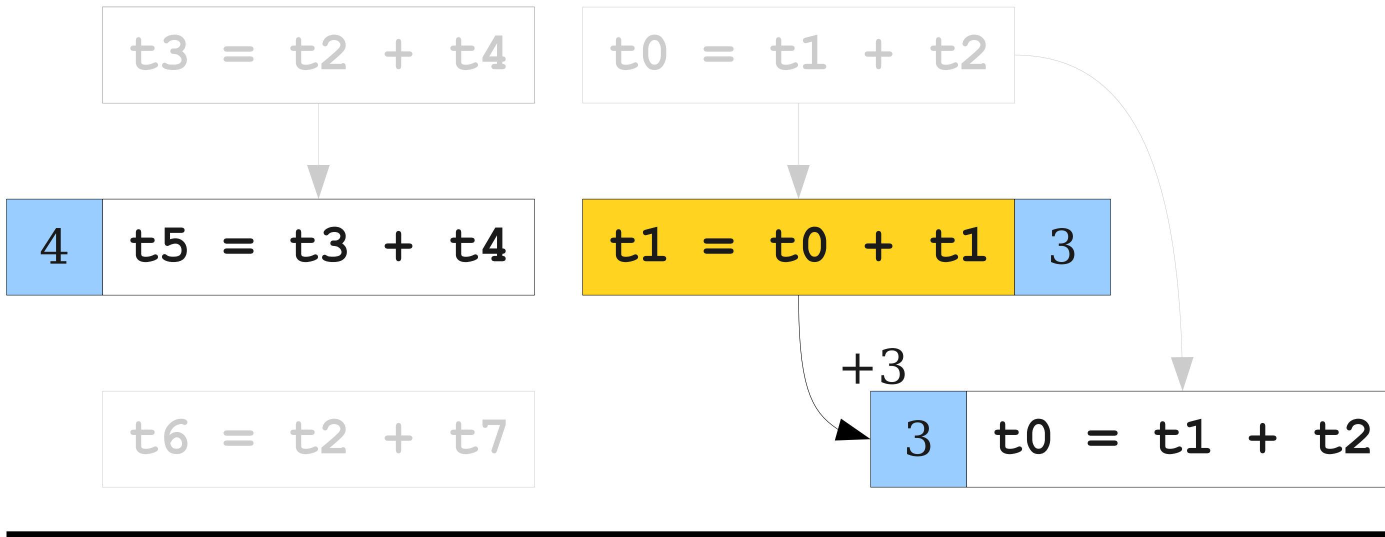
t0 = t1 + t2

t3 = t2 + t4

t6 = t2 + t7

| ID | RR | ALU | RW |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Instruction Scheduling




```
t0 = t1 + t2
t3 = t2 + t4
t6 = t2 + t7
t1 = t0 + t1
```

| ID | RR | ALU | RW |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Instruction Scheduling

```
t3 = t2 + t4        t0 = t1 + t2

4 | t5 = t3 + t4    t1 = t0 + t1 | 3

                        +3
t6 = t2 + t7        6 | t0 = t1 + t2
```

---

```
t0 = t1 + t2
t3 = t2 + t4
t6 = t2 + t7
t1 = t0 + t1
```

| ID | RR | ALU | RW |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Instruction Scheduling

t3 = t2 + t4

t0 = t1 + t2

4 t5 = t3 + t4

t1 = t0 + t1

t6 = t2 + t7

6 t0 = t1 + t2

---

| t0 = t1 + t2 |
|---|
| t3 = t2 + t4 |
| t6 = t2 + t7 |
| t1 = t0 + t1 |

| ID | RR | ALU | RW |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Instruction Scheduling

t3 = t2 + t4

t0 = t1 + t2

4 | t5 = t3 + t4

t1 = t0 + t1

t6 = t2 + t7

6 | t0 = t1 + t2

---

t0 = t1 + t2

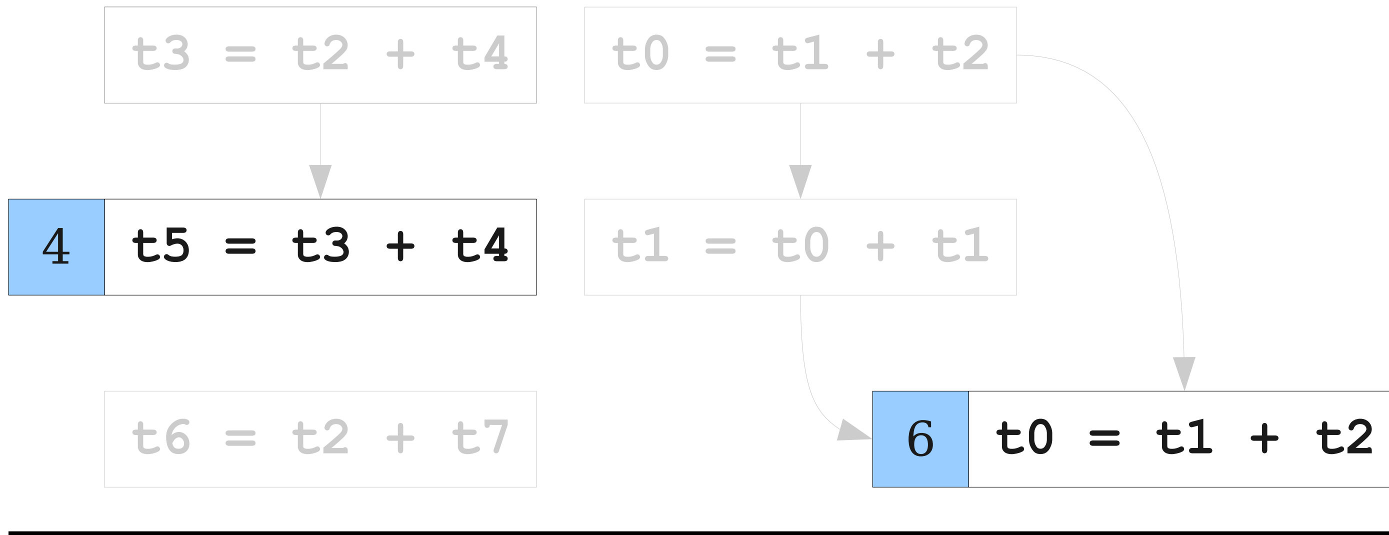
t3 = t2 + t4

t6 = t2 + t7

t1 = t0 + t1

| ID | RR | ALU | RW |
| --- | --- | --- | --- |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Instruction Scheduling




t0 = t1 + t2

t3 = t2 + t4

t6 = t2 + t7

t1 = t0 + t1

**ID RR ALU RW**

# Instruction Scheduling

t3 = t2 + t4

t0 = t1 + t2

4 t5 = t3 + t4

t1 = t0 + t1

t6 = t2 + t7

6 t0 = t1 + t2

---

t0 = t1 + t2
t3 = t2 + t4
t6 = t2 + t7
t1 = t0 + t1
t5 = t3 + t4

| ID | RR | ALU | RW |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Instruction Scheduling

```
t3 = t2 + t4        t0 = t1 + t2
t5 = t3 + t4        t1 = t0 + t1
t6 = t2 + t7
```

6 `t0 = t1 + t2`

| ID | RR | ALU | RW |
|---|---|---|---|
| | | | |

```
t0 = t1 + t2
t3 = t2 + t4
t6 = t2 + t7
t1 = t0 + t1
t5 = t3 + t4
```

# Instruction Scheduling

```
t3 = t2 + t4        t0 = t1 + t2

t5 = t3 + t4        t1 = t0 + t1

t6 = t2 + t7              6  t0 = t1 + t2
```

---

```
t0 = t1 + t2
t3 = t2 + t4
t6 = t2 + t7
t1 = t0 + t1
t5 = t3 + t4
```

| ID | RR | ALU | RW |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Instruction Scheduling

t3 = t2 + t4

t0 = t1 + t2

t5 = t3 + t4

t1 = t0 + t1

t6 = t2 + t7

6 t0 = t1 + t2

---

t0 = t1 + t2

t3 = t2 + t4

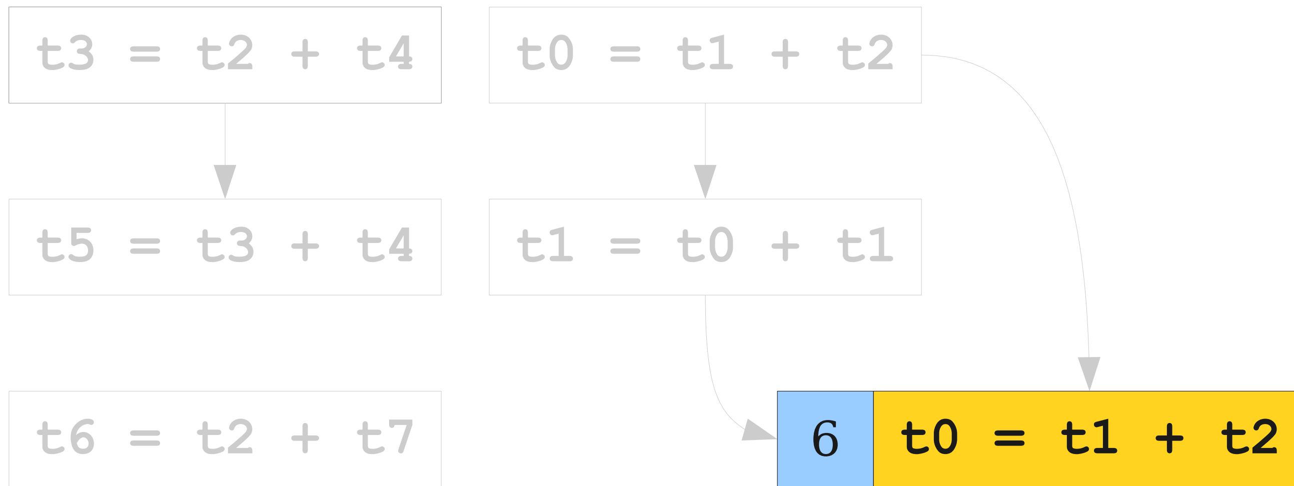
t6 = t2 + t7

t1 = t0 + t1

t5 = t3 + t4

| ID | RR | ALU | RW |
| --- | --- | --- | --- |
| t0 = t1 + t2 | | | |
| t3 = t2 + t4 | t0 = t1 + t2 | | |
| t6 = t2 + t7 | t3 = t2 + t4 | t0 = t1 + t2 | |
| t1 = t0 + t1 | t6 = t2 + t7 | t3 = t2 + t4 | t0 = t1 + t2 |
| t5 = t3 + t4 | t1 = t0 + t1 | t6 = t2 + t7 | t3 = t2 + t4 |
| | t5 = t3 + t4 | t1 = t0 + t1 | t6 = t2 + t7 |
| | | t5 = t3 + t4 | t1 = t0 + t1 |
| | | | t5 = t3 + t4 |
| | | | |
| | | | |

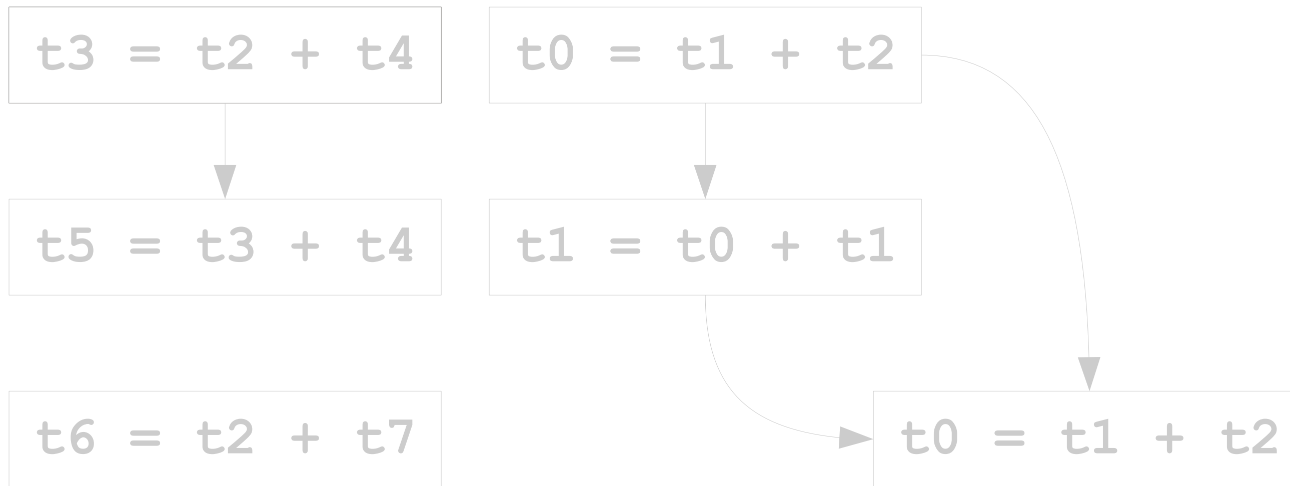
# Instruction Scheduling




```
t0 = t1 + t2
t3 = t2 + t4
t6 = t2 + t7
t1 = t0 + t1
t5 = t3 + t4
t0 = t1 + t2
```

| ID | RR | ALU | RW |
|---|---|---|---|
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

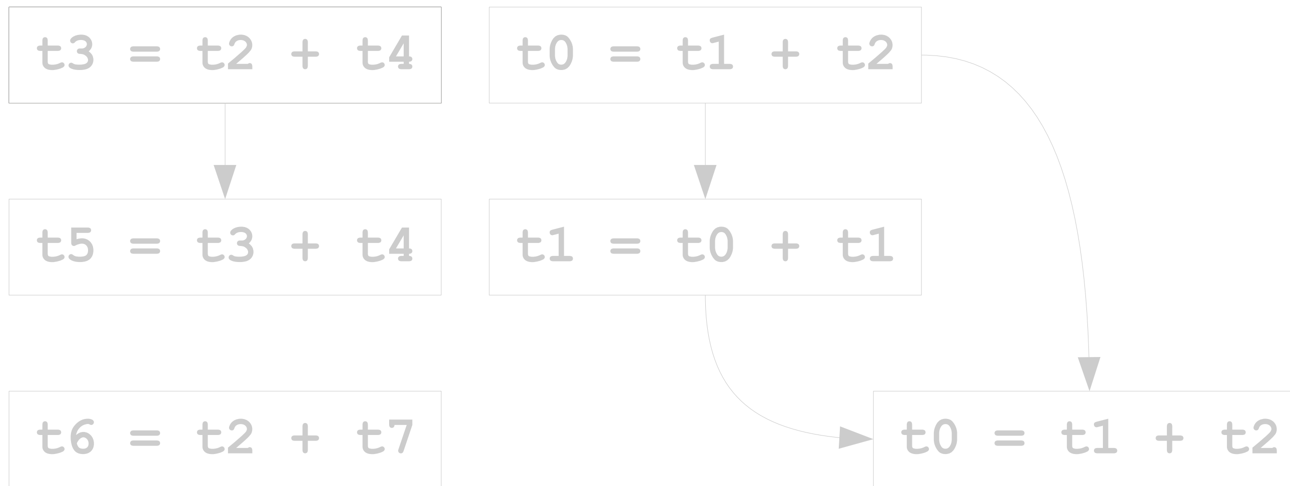
# Instruction Scheduling




```
t0 = t1 + t2
t3 = t2 + t4
t6 = t2 + t7
t1 = t0 + t1
t5 = t3 + t4
t0 = t1 + t2
```

| ID | RR | ALU | RW |
| --- | --- | --- | --- |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |
| | | | |

# Instruction Scheduling




```
t0 = t1 + t2
t3 = t2 + t4
t6 = t2 + t7
t1 = t0 + t1
t5 = t3 + t4
t0 = t1 + t2
```

| ID | RR | ALU | RW |
| --- | --- | --- | --- |

# Instruction Scheduling

```
t3 = t2 + t4        t0 = t1 + t2

t5 = t3 + t4        t1 = t0 + t1

t6 = t2 + t7                           t0 = t1 + t2
```

---

```
t0 = t1 + t2
t3 = t2 + t4
t6 = t2 + t7
t1 = t0 + t1
t5 = t3 + t4
t0 = t1 + t2
```

| ID | RR | ALU | RW |
|---|---|---|---|
| t0 = t1 + t2 | | | |
| t3 = t2 + t4 | t0 = t1 + t2 | | |
| t6 = t2 + t7 | t3 = t2 + t4 | t0 = t1 + t2 | |
| t1 = t0 + t1 | t6 = t2 + t7 | t3 = t2 + t4 | t0 = t1 + t2 |
| t5 = t3 + t4 | t1 = t0 + t1 | t6 = t2 + t7 | t3 = t2 + t4 |
| t0 = t1 + t2 | t5 = t3 + t4 | t1 = t0 + t1 | t6 = t2 + t7 |
| t0 = t1 + t2 | | t5 = t3 + t4 | t1 = t0 + t1 |
| | t0 = t1 + t2 | | t5 = t3 + t4 |
| | | t0 = t1 + t2 | |
| | | | t0 = t1 + t2 |

# For Comparison

```
t0 = t1 + t2
t3 = t2 + t4
t6 = t2 + t7
t1 = t0 + t1
t5 = t3 + t4
t0 = t1 + t2
```

ID RR ALU RW

```
t0 = t1 + t2
t1 = t0 + t1
t3 = t2 + t4
t0 = t1 + t2
t5 = t3 + t4
t6 = t2 + t7
```

ID RR ALU RW

# More Advanced Scheduling

- Modern optimizing compilers can do far more aggressive scheduling to obtain impressive performance gains.
- One powerful technique: **Loop unrolling**
  - Expand out several loop iterations at once.
  - Use previous algorithm to schedule instructions more intelligently.
  - Can find pipelining-level parallelism across loop iterations.
- Even more powerful technique: **Software pipelining**
  - Loop unrolling on steroids; can convert loops using tens of cycles into loops averaging two or three cycles.

# Optimizations for Locality

# Memory Caches

- Because computers use different types of memory, there are a variety of memory caches in the machine.
- Caches are designed to anticipate common use patterns.
- Compilers often have to rewrite code to take maximal advantage of these designs.

# Locality

- Empirically, many programs exhibit **temporal locality** and **spatial locality**.
- Temporal locality: Memory read recently is likely to be read again.
- Spatial locality: Memory read recently will likely have nearby objects read as well.
- Most memory caches are designed to exploit temporal and spatial locality by
  - Holding recently-used memory addresses in cache.
  - Loading nearby memory addresses into cache.

# Memory Caches

# Memory Caches

```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

# Memory Caches

```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

| | |
|---|---|
| arr[0] | 0 |
| arr[1] | 0 |
| arr[2] | 0 |
| arr[3] | 0 |
| arr[4] | 0 |
| arr[5] | 0 |
| arr[6] | 0 |
| arr[7] | 0 |
| arr[8] | 0 |
| arr[9] | 0 |
| arr[10] | 0 |
| arr[11] | 0 |

# Memory Caches

```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

| | |
|---|---|
| arr[0] | 0 |
| arr[1] | 0 |
| arr[2] | 0 |
| arr[3] | 0 |
| arr[4] | 0 |
| arr[5] | 0 |
| arr[6] | 0 |
| arr[7] | 0 |
| arr[8] | 0 |
| arr[9] | 0 |
| arr[10] | 0 |
| arr[11] | 0 |

Memory Cache

# Memory Caches

```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

| | |
|---|---|
| arr[0] | 0 |
| arr[1] | 0 |
| arr[2] | 0 |
| arr[3] | 0 |
| arr[4] | 0 |
| arr[5] | 0 |
| arr[6] | 0 |
| arr[7] | 0 |
| arr[8] | 0 |
| arr[9] | 0 |
| arr[10] | 0 |
| arr[11] | 0 |

Memory Cache

# Memory Caches

```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

| | |
|---|---|
| arr[0] | 0 |
| arr[1] | 0 |
| arr[2] | 0 |
| arr[3] | 0 |
| arr[4] | 0 |
| arr[5] | 0 |
| arr[6] | 0 |
| arr[7] | 0 |
| arr[8] | 0 |
| arr[9] | 0 |
| arr[10] | 0 |
| arr[11] | 0 |

Memory Cache

| | |
|---|---|
| arr[0] | 0 |
| arr[1] | 0 |
| arr[2] | 0 |
| arr[3] | 0 |

# Memory Caches

```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

| | |
|---|---|
| arr[0] | 0 |
| arr[1] | 0 |
| arr[2] | 0 |
| arr[3] | 0 |
| arr[4] | 0 |
| arr[5] | 0 |
| arr[6] | 0 |
| arr[7] | 0 |
| arr[8] | 0 |
| arr[9] | 0 |
| arr[10] | 0 |
| arr[11] | 0 |

Memory Cache

| | |
|---|---|
| arr[0] | 5 |
| arr[1] | 0 |
| arr[2] | 0 |
| arr[3] | 0 |

# Memory Caches

```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

| | |
|---|---|
| arr[0] | 0 |
| arr[1] | 0 |
| arr[2] | 0 |
| arr[3] | 0 |
| arr[4] | 0 |
| arr[5] | 0 |
| arr[6] | 0 |
| arr[7] | 0 |
| arr[8] | 0 |
| arr[9] | 0 |
| arr[10] | 0 |
| arr[11] | 0 |

Memory Cache

| | |
|---|---|
| arr[0] | 5 |
| arr[1] | 0 |
| arr[2] | 0 |
| arr[3] | 0 |

# Memory Caches

```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

| arr[0] | 0 |
|---|---|
| arr[1] | 0 |
| arr[2] | 0 |
| arr[3] | 0 |
| arr[4] | 0 |
| arr[5] | 0 |
| arr[6] | 0 |
| arr[7] | 0 |
| arr[8] | 0 |
| arr[9] | 0 |
| arr[10] | 0 |
| arr[11] | 0 |

Already in cache!

Memory Cache

| arr[0] | 5 |
|---|---|
| arr[1] | 0 |
| arr[2] | 0 |
| arr[3] | 0 |

# Memory Caches

```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

| | |
|---|---|
| arr[0] | 0 |
| arr[1] | 0 |
| arr[2] | 0 |
| arr[3] | 0 |
| arr[4] | 0 |
| arr[5] | 0 |
| arr[6] | 0 |
| arr[7] | 0 |
| arr[8] | 0 |
| arr[9] | 0 |
| arr[10] | 0 |
| arr[11] | 0 |

Memory Cache

| | |
|---|---|
| arr[0] | 5 |
| arr[1] | 0 |
| arr[2] | 0 |
| arr[3] | 0 |

# Memory Caches

```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

| | |
|---|---|
| arr[0] | 0 |
| arr[1] | 0 |
| arr[2] | 0 |
| arr[3] | 0 |
| arr[4] | 0 |
| arr[5] | 0 |
| arr[6] | 0 |
| arr[7] | 0 |
| arr[8] | 0 |
| arr[9] | 0 |
| arr[10] | 0 |
| arr[11] | 0 |

Memory Cache

| | |
|---|---|
| arr[0] | 5 |
| arr[1] | 0 |
| arr[2] | 6 |
| arr[3] | 0 |

# Memory Caches

```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

| | |
|---|---|
| arr[0] | 0 |
| arr[1] | 0 |
| arr[2] | 0 |
| arr[3] | 0 |
| arr[4] | 0 |
| arr[5] | 0 |
| arr[6] | 0 |
| arr[7] | 0 |
| arr[8] | 0 |
| arr[9] | 0 |
| arr[10] | 0 |
| arr[11] | 0 |

Memory Cache

| | |
|---|---|
| arr[0] | 5 |
| arr[1] | 0 |
| arr[2] | 6 |
| arr[3] | 0 |

# Memory Caches

```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

| | |
|---|---|
| arr[0] | 5 |
| arr[1] | 0 |
| arr[2] | 6 |
| arr[3] | 0 |
| arr[4] | 0 |
| arr[5] | 0 |
| arr[6] | 0 |
| arr[7] | 0 |
| arr[8] | 0 |
| arr[9] | 0 |
| arr[10] | 0 |
| arr[11] | 0 |

Memory Cache

# Memory Caches

```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

| | |
|---|---|
| arr[0] | 5 |
| arr[1] | 0 |
| arr[2] | 6 |
| arr[3] | 0 |
| arr[4] | 0 |
| arr[5] | 0 |
| arr[6] | 0 |
| arr[7] | 0 |
| arr[8] | 0 |
| arr[9] | 0 |
| arr[10] | 0 |
| arr[11] | 0 |

Memory Cache

| | |
|---|---|
| arr[8] | 0 |
| arr[9] | 0 |
| arr[10] | 0 |
| arr[11] | 0 |

# Memory Caches

```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

| | |
|---|---|
| arr[0] | 5 |
| arr[1] | 0 |
| arr[2] | 6 |
| arr[3] | 0 |
| arr[4] | 0 |
| arr[5] | 0 |
| arr[6] | 0 |
| arr[7] | 0 |
| arr[8] | 0 |
| arr[9] | 0 |
| arr[10] | 0 |
| arr[11] | 0 |

Memory Cache

| | |
|---|---|
| arr[8] | 0 |
| arr[9] | 0 |
| arr[10] | 13 |
| arr[11] | 0 |

# Memory Caches

```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

| | |
|---|---|
| arr[0] | 5 |
| arr[1] | 0 |
| arr[2] | 6 |
| arr[3] | 0 |
| arr[4] | 0 |
| arr[5] | 0 |
| arr[6] | 0 |
| arr[7] | 0 |
| arr[8] | 0 |
| arr[9] | 0 |
| arr[10] | 0 |
| arr[11] | 0 |

Memory Cache

| | |
|---|---|
| arr[8] | 0 |
| arr[9] | 0 |
| arr[10] | 13 |
| arr[11] | 0 |

# Memory Caches

```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

| | |
|---|---|
| arr[0] | 5 |
| arr[1] | 0 |
| arr[2] | 6 |
| arr[3] | 0 |
| arr[4] | 0 |
| arr[5] | 0 |
| arr[6] | 0 |
| arr[7] | 0 |
| arr[8] | 0 |
| arr[9] | 0 |
| arr[10] | 13 |
| arr[11] | 0 |

Memory Cache

# Memory Caches

```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

| | |
|---|---|
| arr[0] | 5 |
| arr[1] | 0 |
| arr[2] | 6 |
| arr[3] | 0 |
| arr[4] | 0 |
| arr[5] | 0 |
| arr[6] | 0 |
| arr[7] | 0 |
| arr[8] | 0 |
| arr[9] | 0 |
| arr[10] | 13 |
| arr[11] | 0 |

Memory Cache

| | |
|---|---|
| arr[0] | 5 |
| arr[1] | 0 |
| arr[2] | 6 |
| arr[3] | 0 |

# Memory Caches

```
arr[0]  = 5;
arr[2]  = 6;
arr[10] = 13;
arr[1]  = 4;
```

| | |
|---|---|
| arr[0] | 5 |
| arr[1] | 0 |
| arr[2] | 6 |
| arr[3] | 0 |
| arr[4] | 0 |
| arr[5] | 0 |
| arr[6] | 0 |
| arr[7] | 0 |
| arr[8] | 0 |
| arr[9] | 0 |
| arr[10] | 13 |
| arr[11] | 0 |

Memory Cache

| | |
|---|---|
| arr[0] | 5 |
| arr[1] | 4 |
| arr[2] | 6 |
| arr[3] | 0 |

# The Problem with Caches

# The Problem with Caches

```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```

# The Problem with Caches

```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```

| 00 | 01 | 02 | 03 |
|---|---|---|---|
| 10 | 11 | 12 | 13 |
| 20 | 21 | 22 | 23 |
| 30 | 31 | 32 | 33 |

# The Problem with Caches

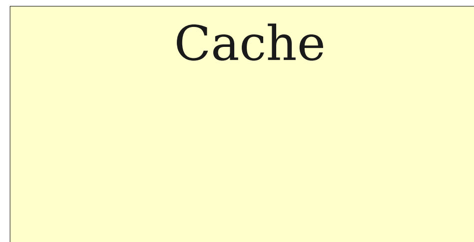
```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```

| 00 | 01 | 02 | 03 | 10 | 11 | 12 | 13 | 20 | 21 | 22 | 23 | 30 | 31 | 32 | 33 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|

# The Problem with Caches

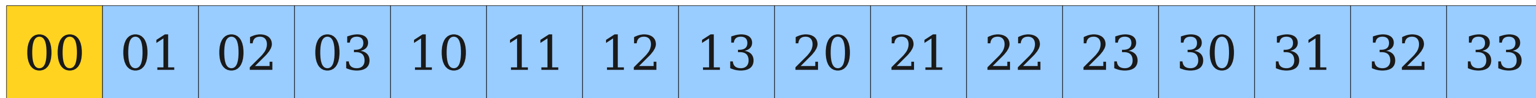
```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```

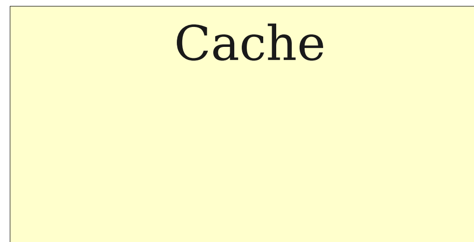
| 00 | 01 | 02 | 03 | 10 | 11 | 12 | 13 | 20 | 21 | 22 | 23 | 30 | 31 | 32 | 33 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|



# The Problem with Caches

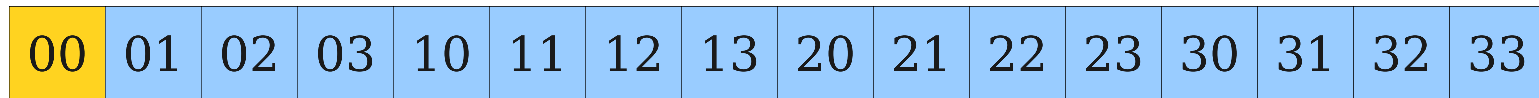
```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```

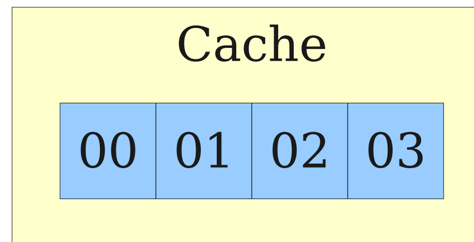


# The Problem with Caches

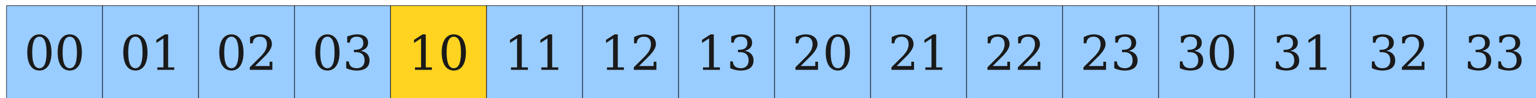
```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```

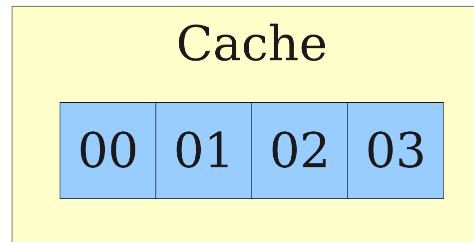


# The Problem with Caches

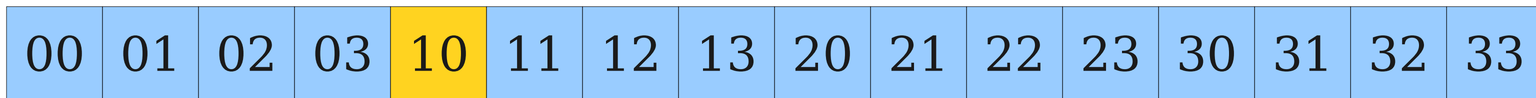
```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```

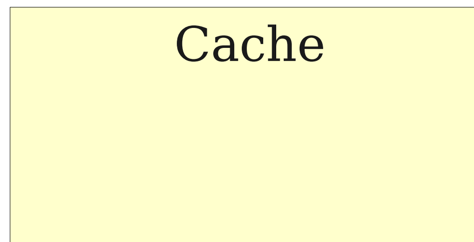


# The Problem with Caches

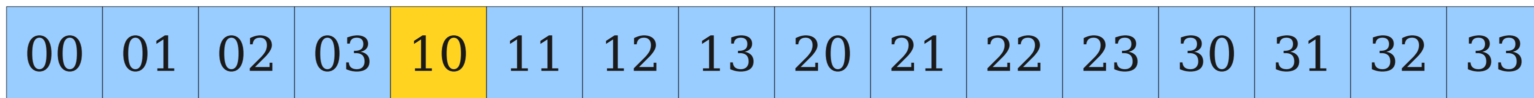
```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```

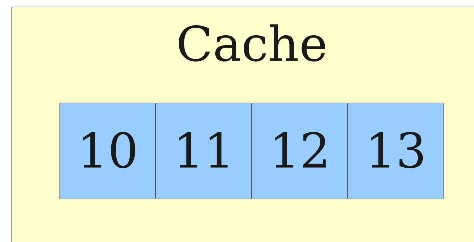


# The Problem with Caches

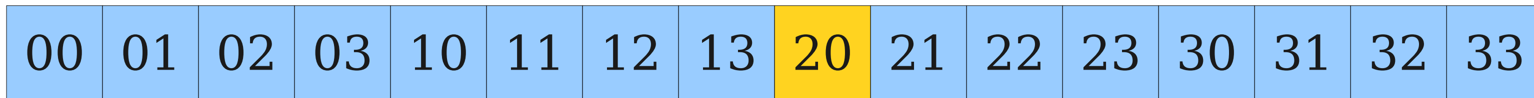
```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```

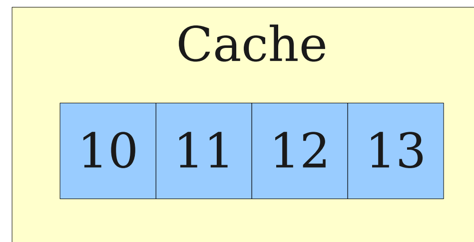


# The Problem with Caches

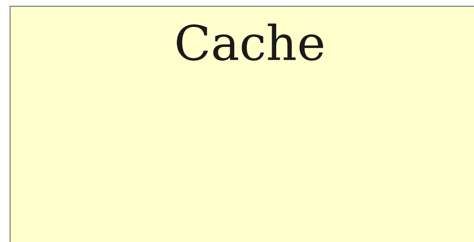
```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```



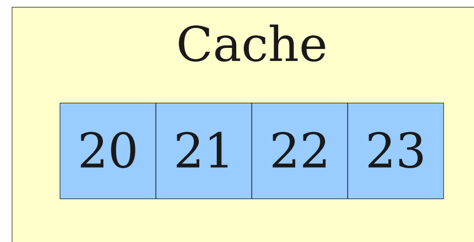
# The Problem with Caches

```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```



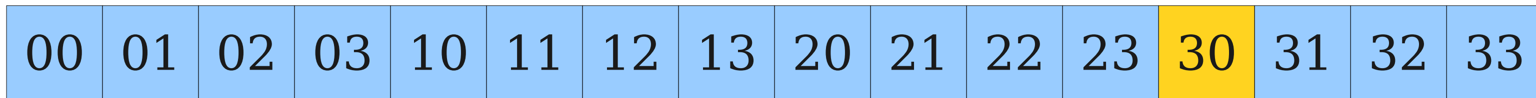
# The Problem with Caches

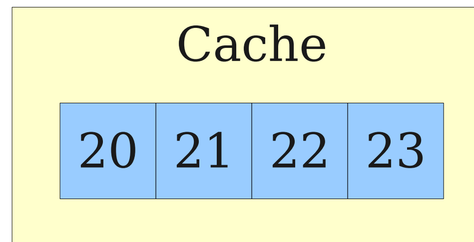
```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```



# The Problem with Caches

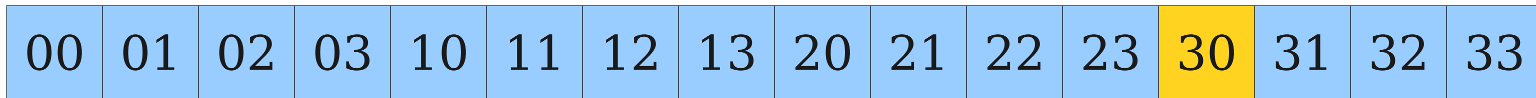
```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```

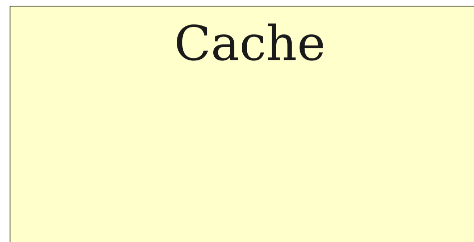


# The Problem with Caches

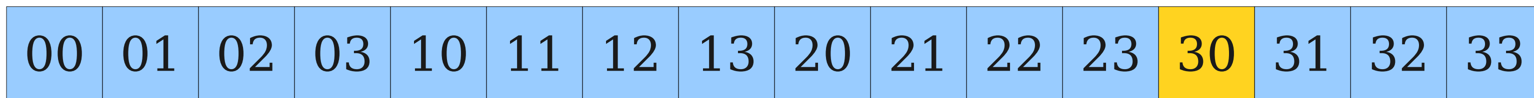
```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```

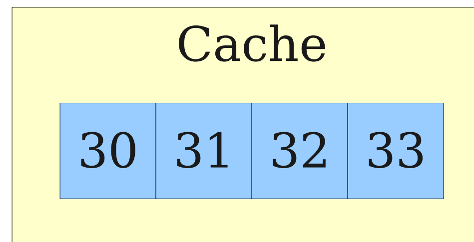


# The Problem with Caches

```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```



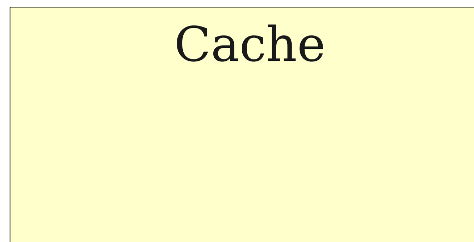
# Improving Locality

- Programmers frequently write code without understanding the locality implications.
  - Languages don't expose low-level memory details.
- Some compilers are capable of rewriting code to take advantage of locality.
- Cool optimizations:
  - Loop reordering.
  - Structure peeling.

# Loop Reordering

```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```

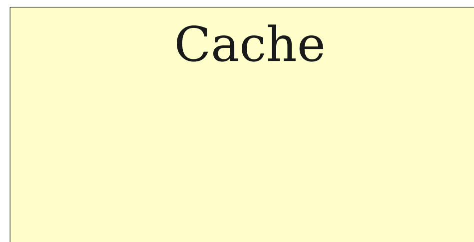
| 00 | 01 | 02 | 03 | 10 | 11 | 12 | 13 | 20 | 21 | 22 | 23 | 30 | 31 | 32 | 33 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|



# Loop Reordering

```
int[][] array;
for (j = 0; j < 4; j = j + 1)
    for (i = 0; i < 4; i = i + 1)
        array[i][j] = 0;
```

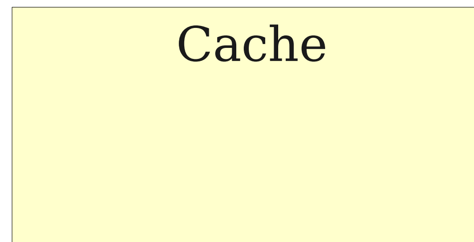
| 00 | 01 | 02 | 03 | 10 | 11 | 12 | 13 | 20 | 21 | 22 | 23 | 30 | 31 | 32 | 33 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|



# Loop Reordering

```
int[][] array;
for (i = 0; i < 4; i = i + 1)
    for (j = 0; j < 4; j = j + 1)
        array[i][j] = 0;
```

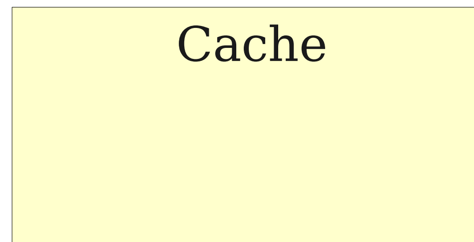
| 00 | 01 | 02 | 03 | 10 | 11 | 12 | 13 | 20 | 21 | 22 | 23 | 30 | 31 | 32 | 33 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|



# Loop Reordering

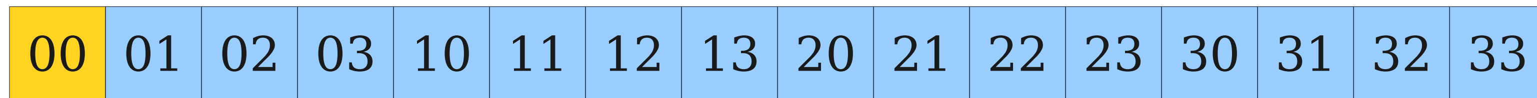
```
int[][] array;
for (i = 0; i < 4; i = i + 1)
    for (j = 0; j < 4; j = j + 1)
        array[i][j] = 0;
```

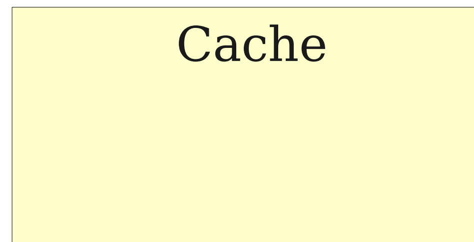
| 00 | 01 | 02 | 03 | 10 | 11 | 12 | 13 | 20 | 21 | 22 | 23 | 30 | 31 | 32 | 33 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|



# Loop Reordering

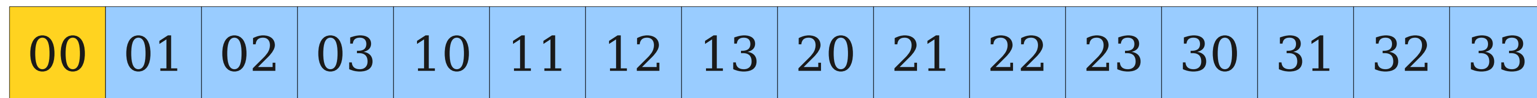
```
int[][] array;
for (i = 0; i < 4; i = i + 1)
    for (j = 0; j < 4; j = j + 1)
        array[i][j] = 0;
```

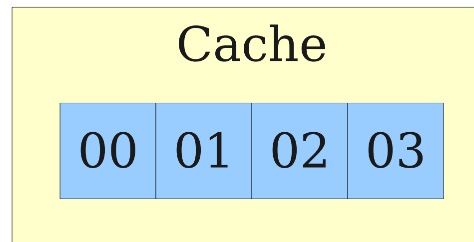


# Loop Reordering

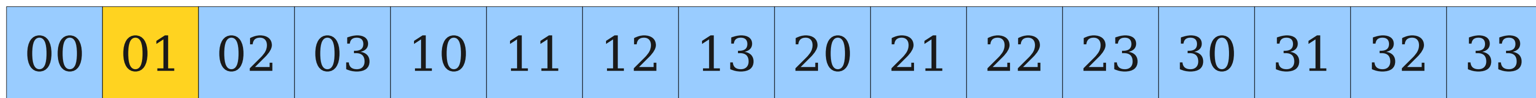
```
int[][] array;
for (i = 0; i < 4; i = i + 1)
    for (j = 0; j < 4; j = j + 1)
        array[i][j] = 0;
```

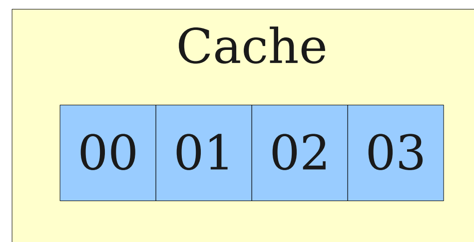


# Loop Reordering

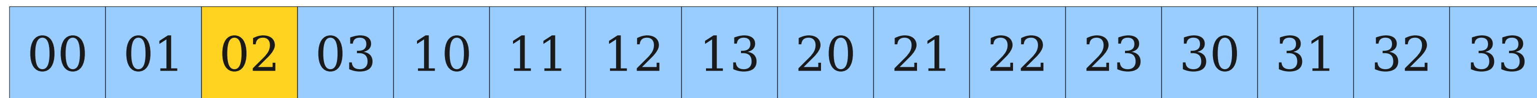
```
int[][] array;
for (i = 0; i < 4; i = i + 1)
    for (j = 0; j < 4; j = j + 1)
        array[i][j] = 0;
```

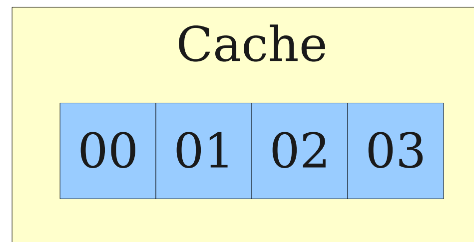


# Loop Reordering

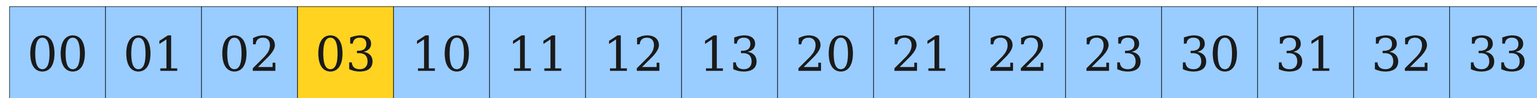
```
int[][] array;
for (i = 0; i < 4; i = i + 1)
    for (j = 0; j < 4; j = j + 1)
        array[i][j] = 0;
```

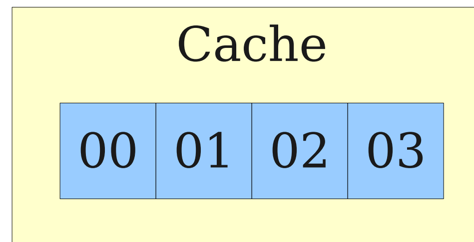


# Loop Reordering

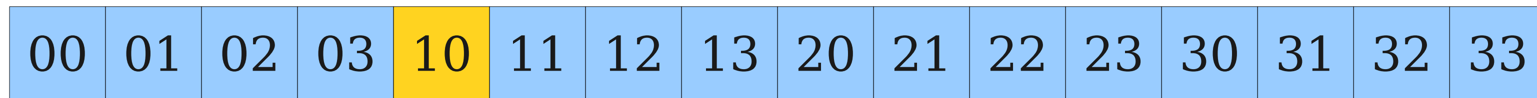
```
int[][] array;
for (i = 0; i < 4; i = i + 1)
    for (j = 0; j < 4; j = j + 1)
        array[i][j] = 0;
```

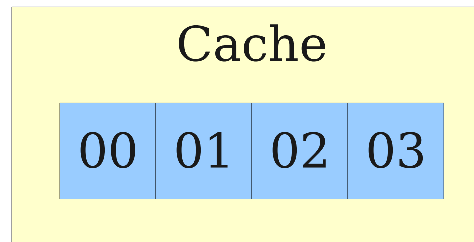


# Loop Reordering

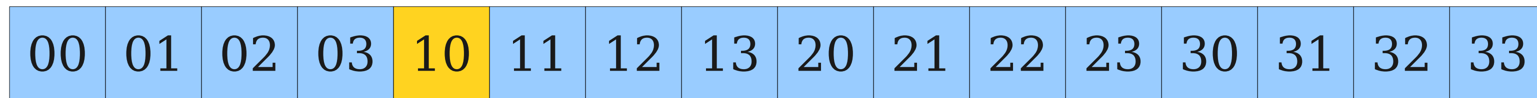
```
int[][] array;
for (i = 0; i < 4; i = i + 1)
    for (j = 0; j < 4; j = j + 1)
        array[i][j] = 0;
```

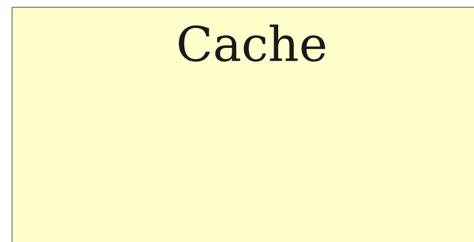


# Loop Reordering

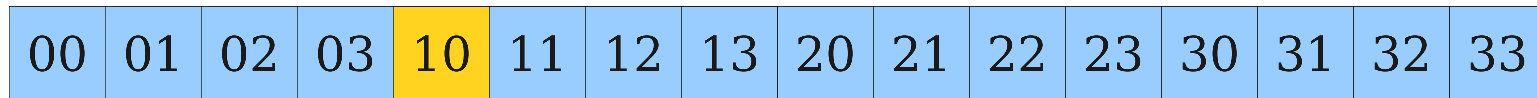
```
int[][] array;
for (i = 0; i < 4; i = i + 1)
    for (j = 0; j < 4; j = j + 1)
        array[i][j] = 0;
```

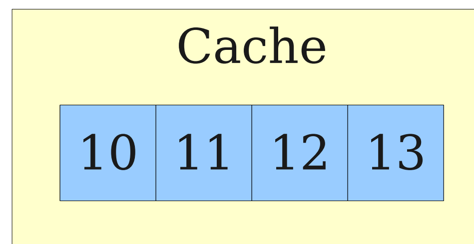


# Loop Reordering

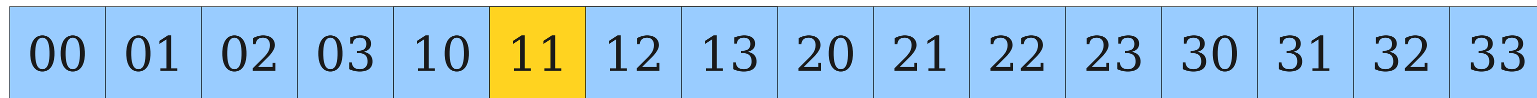
```
int[][] array;
for (i = 0; i < 4; i = i + 1)
    for (j = 0; j < 4; j = j + 1)
        array[i][j] = 0;
```

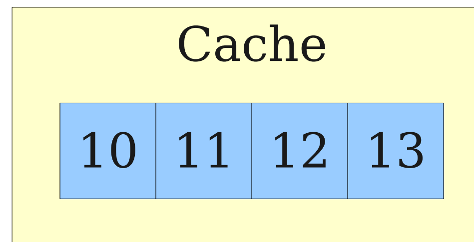


# Loop Reordering

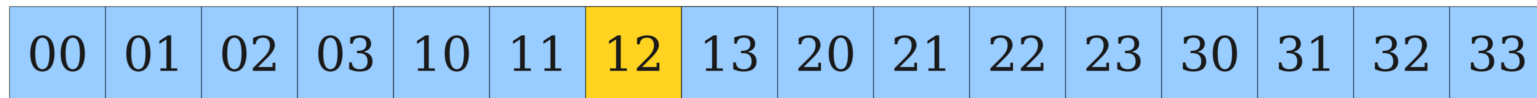
```
int[][] array;
for (i = 0; i < 4; i = i + 1)
    for (j = 0; j < 4; j = j + 1)
        array[i][j] = 0;
```

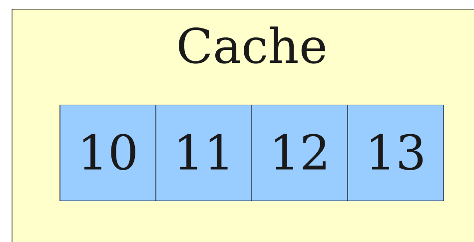


# Loop Reordering

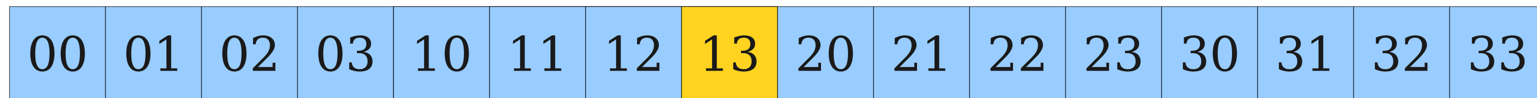
```
int[][] array;
for (i = 0; i < 4; i = i + 1)
    for (j = 0; j < 4; j = j + 1)
        array[i][j] = 0;
```

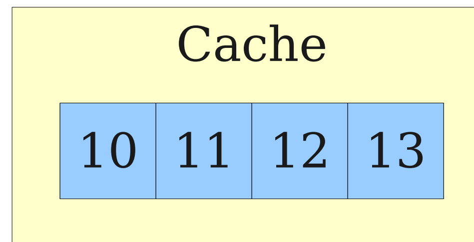


# Loop Reordering

```
int[][] array;
for (i = 0; i < 4; i = i + 1)
    for (j = 0; j < 4; j = j + 1)
        array[i][j] = 0;
```



# Structure Peeling

# Structure Peeling

```
class Point2D {
    int x;
    int y;
}
```

# Structure Peeling

```
class Point2D {
    int x;
    int y;
}

void MyFunction() {
    Point2D[] pts = new Point2D[1024];

    /* … initialize the points … */

    int maxX = 0, maxY = 0;
    for (int i = 0; i < 512; ++i)
        maxX = max(pts[i].x, maxX);
    for (int i = 512; i < 1024; ++i)
        maxY = max(pts[i].y, maxY);
}
```

# Structure Peeling

```
class Point2D {
    int x;
    int y;
}

void MyFunction() {
    Point2D[] pts = new Point2D[1024];

    /* … initialize the points … */

    int maxX = 0, maxY = 0;
    for (int i = 0; i < 512; ++i)
        maxX = max(pts[i].x, maxX);
    for (int i = 512; i < 1024; ++i)
        maxY = max(pts[i].y, maxY);
}
```

| |
|---|
| **pts[0].x** |
| **pts[0].y** |
| **pts[1].x** |
| **pts[1].y** |
| **pts[2].x** |
| **pts[2].y** |
| **pts[3].x** |
| **pts[3].y** |
| **pts[4].x** |
| **pts[4].y** |
| **pts[5].x** |
| **pts[5].y** |

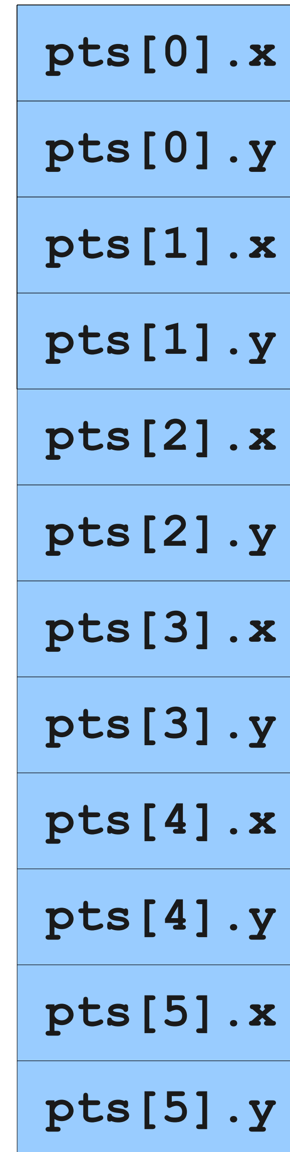
...

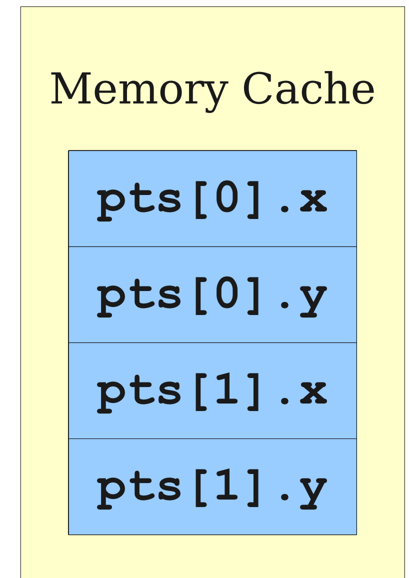
# Structure Peeling

```
class Point2D {
    int x;
    int y;
}

void MyFunction() {
    Point2D[] pts = new Point2D[1024];

    /* … initialize the points … */

    int maxX = 0, maxY = 0;
    for (int i = 0; i < 512; ++i)
        maxX = max(pts[i].x, maxX);
    for (int i = 512; i < 1024; ++i)
        maxY = max(pts[i].y, maxY);
}
```

# Structure Peeling

```
class Point2D {
    int x;
    int y;
}

void MyFunction() {
    Point2D[] pts = new Point2D[1024];

    /* … initialize the points … */

    int maxX = 0, maxY = 0;
    for (int i = 0; i < 512; ++i)
        maxX = max(pts[i].x, maxX);
    for (int i = 512; i < 1024; ++i)
        maxY = max(pts[i].y, maxY);
}
```



# Structure Peeling

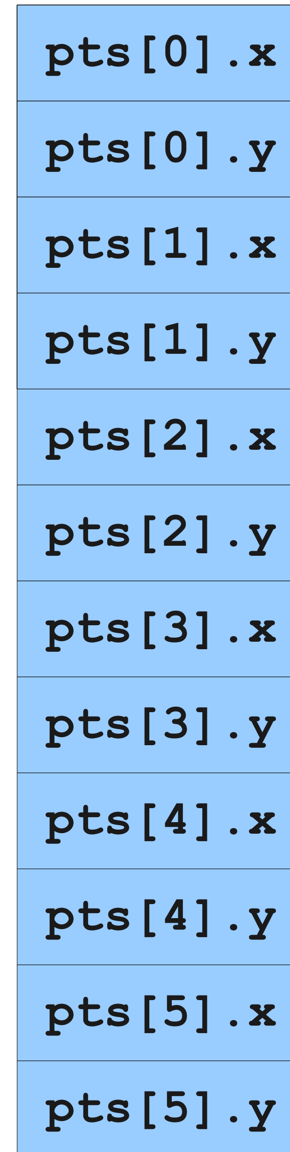
```
class Point2D {
    int x;
    int y;
}

void MyFunction() {
    Point2D[] pts = new Point2D[1024];

    /* … initialize the points … */

    int maxX = 0, maxY = 0;
    for (int i = 0; i < 512; ++i)
        maxX = max(pts[i].x, maxX);
    for (int i = 512; i < 1024; ++i)
        maxY = max(pts[i].y, maxY);
}
```

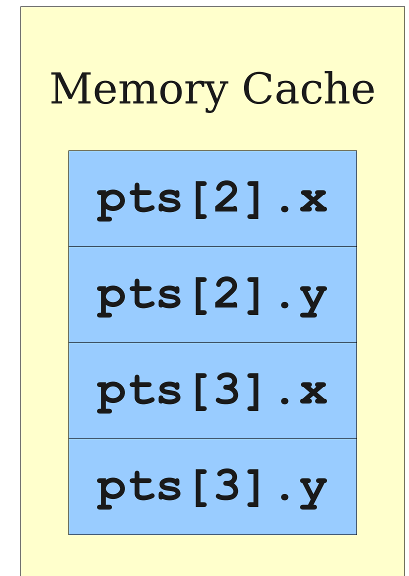
**pts[0].x**
**pts[0].y**
**pts[1].x**
**pts[1].y**
**pts[2].x**
**pts[2].y**
**pts[3].x**
**pts[3].y**
**pts[4].x**
**pts[4].y**
**pts[5].x**
**pts[5].y**
...

Memory Cache

# Structure Peeling

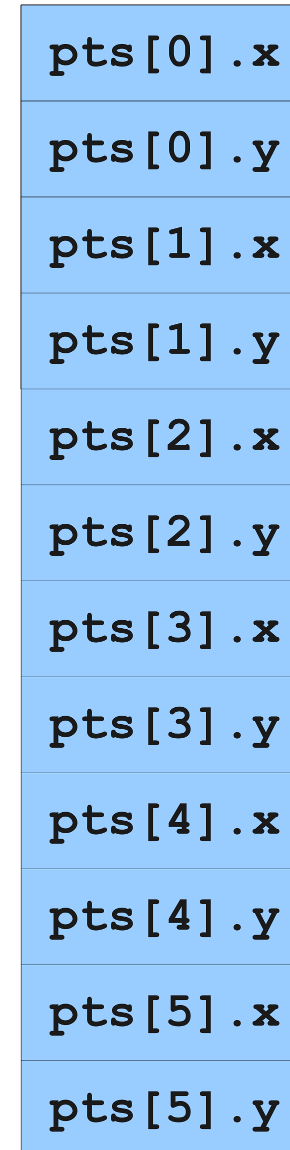
```
class Point2D {
    int x;
    int y;
}

void MyFunction() {
    Point2D[] pts = new Point2D[1024];

    /* … initialize the points … */

    int maxX = 0, maxY = 0;
    for (int i = 0; i < 512; ++i)
        maxX = max(pts[i].x, maxX);
    for (int i = 512; i < 1024; ++i)
        maxY = max(pts[i].y, maxY);
}
```



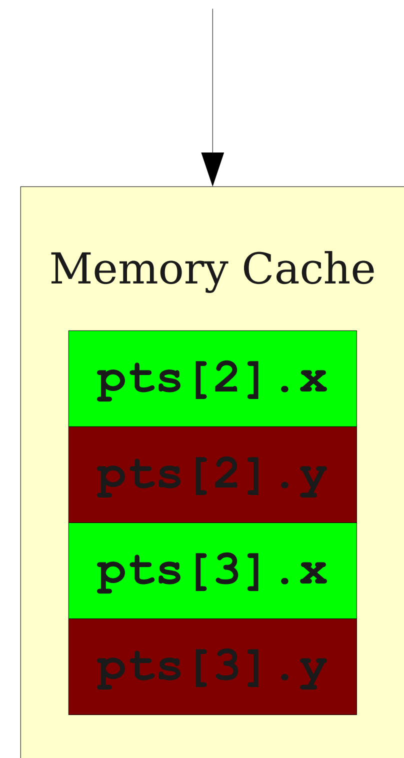



# Structure Peeling

```
class Point2D {
    int x;
    int y;
}

void MyFunction() {
    Point2D[] pts = new Point2D[1024];

    /* … initialize the points … */

    int maxX = 0, maxY = 0;
    for (int i = 0; i < 512; ++i)
        maxX = max(pts[i].x, maxX);
    for (int i = 512; i < 1024; ++i)
        maxY = max(pts[i].y, maxY);
}
```

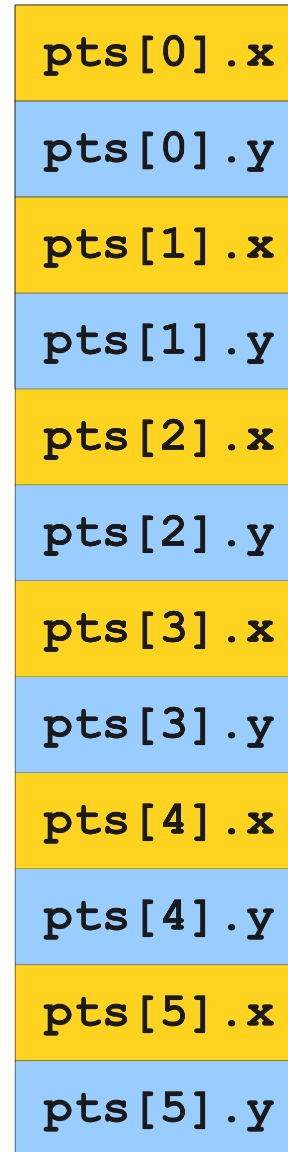



Only half the cache is useful!



# Structure Peeling

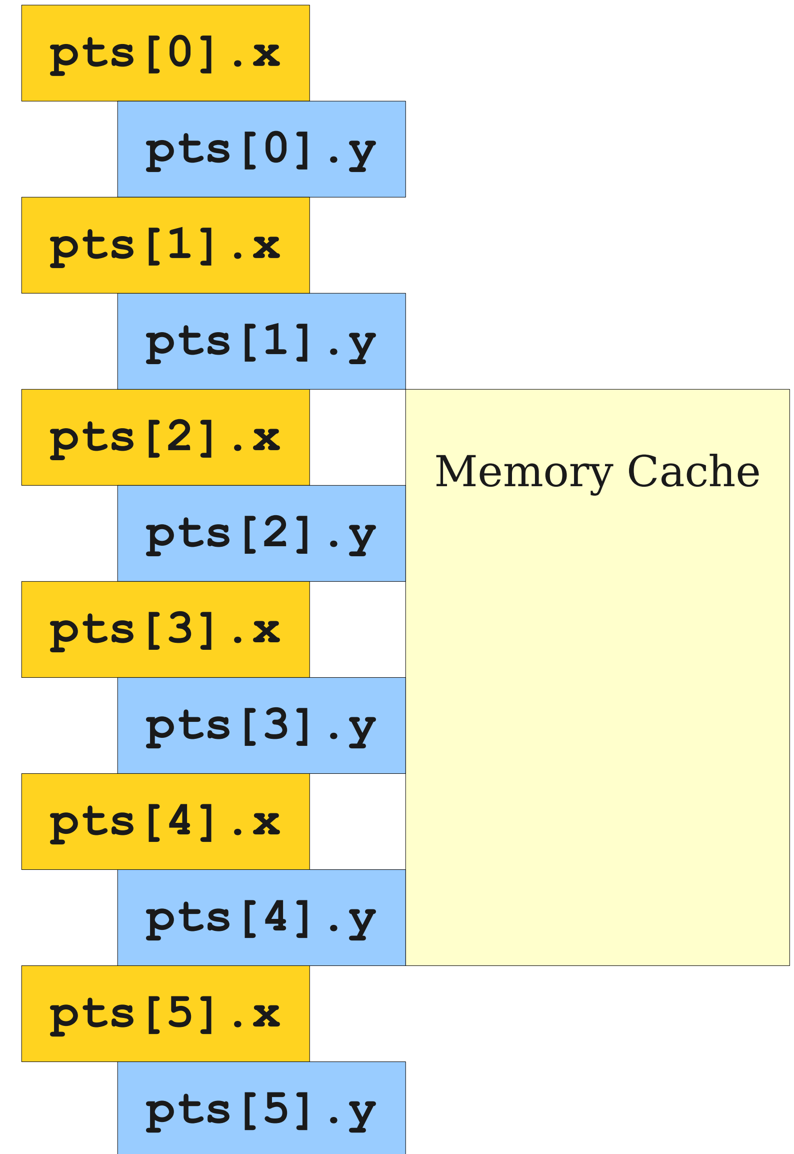
```
class Point2D {
    int x;
    int y;
}

void MyFunction() {
    Point2D[] pts = new Point2D[1024];

    /* … initialize the points … */

    int maxX = 0, maxY = 0;
    for (int i = 0; i < 512; ++i)
        maxX = max(pts[i].x, maxX);
    for (int i = 512; i < 1024; ++i)
        maxY = max(pts[i].y, maxY);
}
```




Memory Cache

# Structure Peeling

```
class Point2D {
    int x;
    int y;
}

void MyFunction() {
    Point2D[] pts = new Point2D[1024];

    /* … initialize the points … */

    int maxX = 0, maxY = 0;
    for (int i = 0; i < 512; ++i)
        maxX = max(pts[i].x, maxX);
    for (int i = 512; i < 1024; ++i)
        maxY = max(pts[i].y, maxY);
}
```



# Structure Peeling

```
class Point2D {
    int x;
    int y;
}

void MyFunction() {
    Point2D[] pts = new Point2D[1024];

    /* … initialize the points … */

    int maxX = 0, maxY = 0;
    for (int i = 0; i < 512; ++i)
        maxX = max(pts[i].x, maxX);
    for (int i = 512; i < 1024; ++i)
        maxY = max(pts[i].y, maxY);
}
```

# Structure Peeling

```
class Point2D {
    int x;
    int y;
}

void MyFunction() {
    Point2D[] pts = new Point2D[1024];

    /* … initialize the points … */

    int maxX = 0, maxY = 0;
    for (int i = 0; i < 512; ++i)
        maxX = max(pts[i].x, maxX);
    for (int i = 512; i < 1024; ++i)
        maxY = max(pts[i].y, maxY);
}
```

**pts[0].x**
**pts[1].x**
**pts[2].x**
**pts[3].x**
**pts[4].x**
...
**pts[0].y**
**pts[1].y**
**pts[2].y**
**pts[3].y**
**pts[4].y**
...

Memory Cache

# Structure Peeling

```
class Point2D {
    int x;
    int y;
}

void MyFunction() {
    Point2D[] pts = new Point2D[1024];

    /* … initialize the points … */

    int maxX = 0, maxY = 0;
    for (int i = 0; i < 512; ++i)
        maxX = max(pts[i].x, maxX);
    for (int i = 512; i < 1024; ++i)
        maxY = max(pts[i].y, maxY);
}
```

**pts[0].x**

**pts[1].x**

**pts[2].x**

**pts[3].x**

**pts[4].x**

...

**pts[0].y**

**pts[1].y**

**pts[2].y**

**pts[3].y**

**pts[4].y**

...

Memory Cache

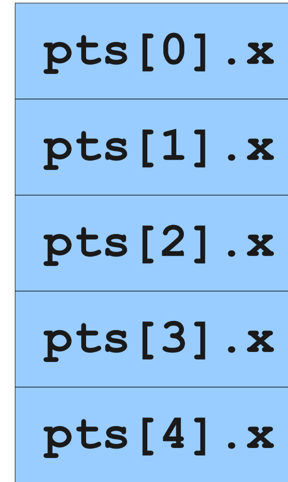
Array of classes to class of arrays!

# Structure Peeling

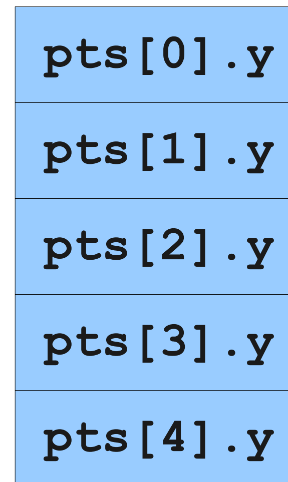
```
class Point2D {
    int x;
    int y;
}

void MyFunction() {
    Point2D[] pts = new Point2D[1024];

    /* … initialize the points … */

    int maxX = 0, maxY = 0;
    for (int i = 0; i < 512; ++i)
        maxX = max(pts[i].x, maxX);
    for (int i = 512; i < 1024; ++i)
        maxY = max(pts[i].y, maxY);
}
```

# Structure Peeling

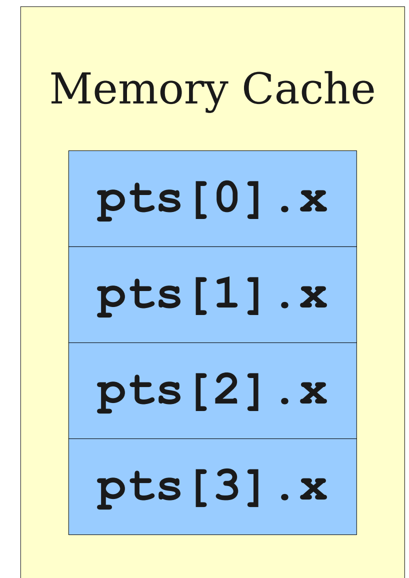
```
class Point2D {
    int x;
    int y;
}

void MyFunction() {
    Point2D[] pts = new Point2D[1024];

    /* … initialize the points … */

    int maxX = 0, maxY = 0;
    for (int i = 0; i < 512; ++i)
        maxX = max(pts[i].x, maxX);
    for (int i = 512; i < 1024; ++i)
        maxY = max(pts[i].y, maxY);
}
```




...

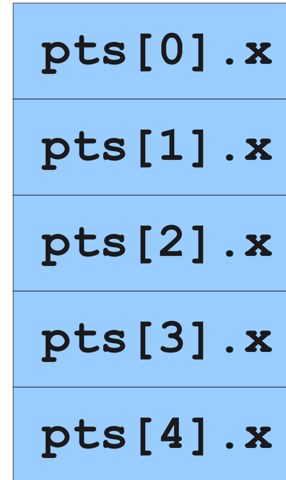



...

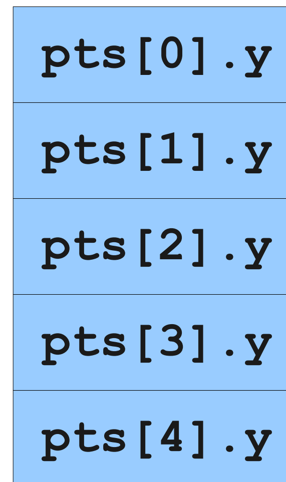


# Structure Peeling

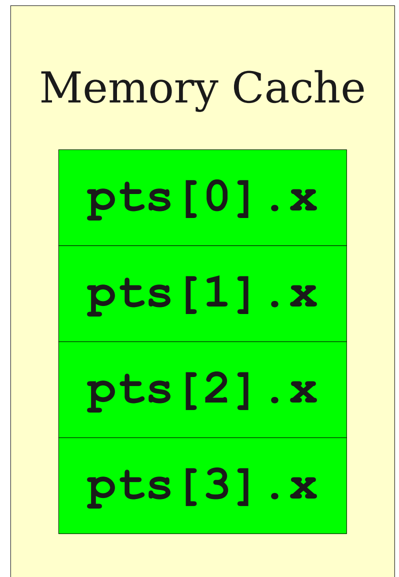
```
class Point2D {
    int x;
    int y;
}

void MyFunction() {
    Point2D[] pts = new Point2D[1024];

    /* … initialize the points … */

    int maxX = 0, maxY = 0;
    for (int i = 0; i < 512; ++i)
        maxX = max(pts[i].x, maxX);
    for (int i = 512; i < 1024; ++i)
        maxY = max(pts[i].y, maxY);
}
```




...




...



# Optimizations for Parallelism

# Everything is Parallel Now

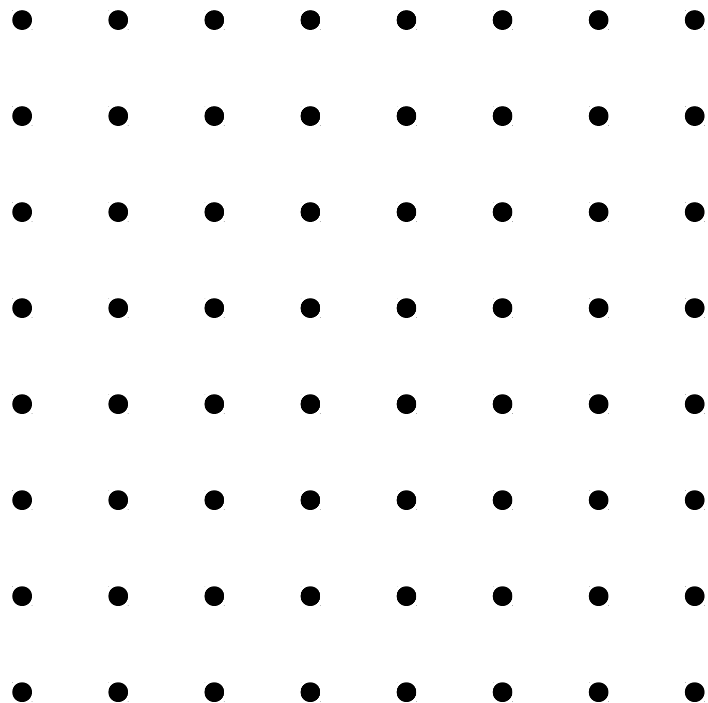
- Virtually all new computers have multiple processors.
  - Even some phones have multiple cores!
- High-end machines now usually have at least eight cores.
- How do we optimize code to work well in parallel?

# Loop Parallelization

- Many numeric computations work by iterating over large arrays of elements.
  - Especially true in FORTRAN code.
- Optimize loops over arrays by identifying inherent parallelism.
- Three-step process:
  - Identify which array values depend on one another.
  - Split each group of dependent values into its own task.
  - Map each task onto one processor.
- Beautiful (but tricky!) framework for doing this automatically; I'm not even going to try to go into mathematical details.

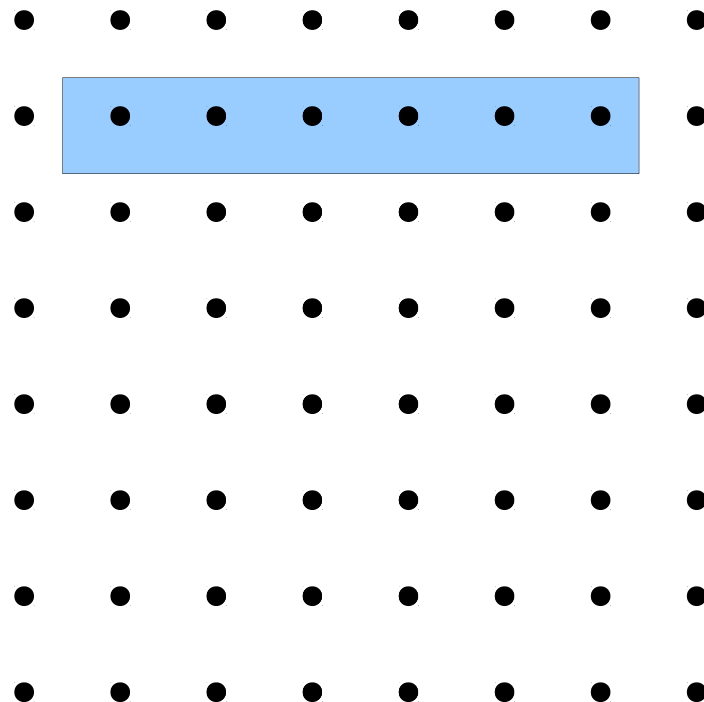
# Iteration Spaces

- The **iteration space** of a set of loops is the set of valid indices taken by the loop counters.



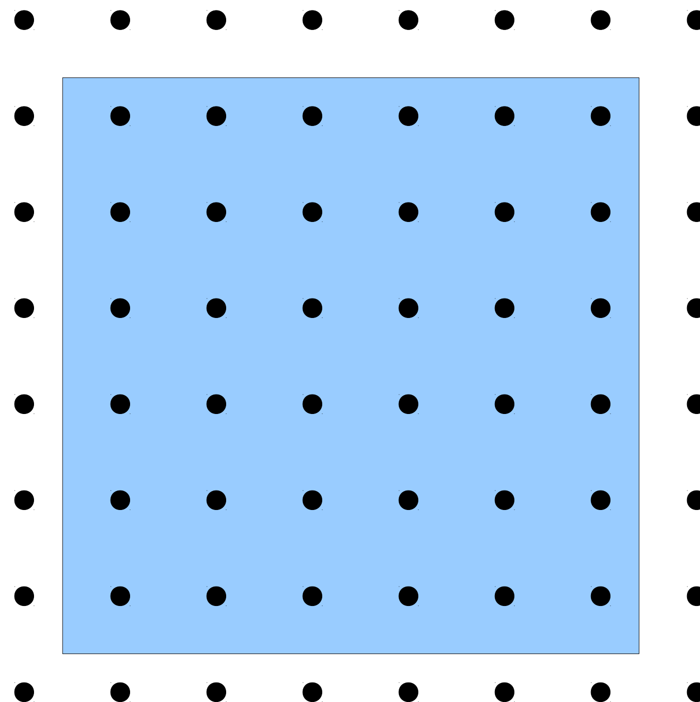
# Iteration Spaces

- The **iteration space** of a set of loops is the set of valid indices taken by the loop counters.



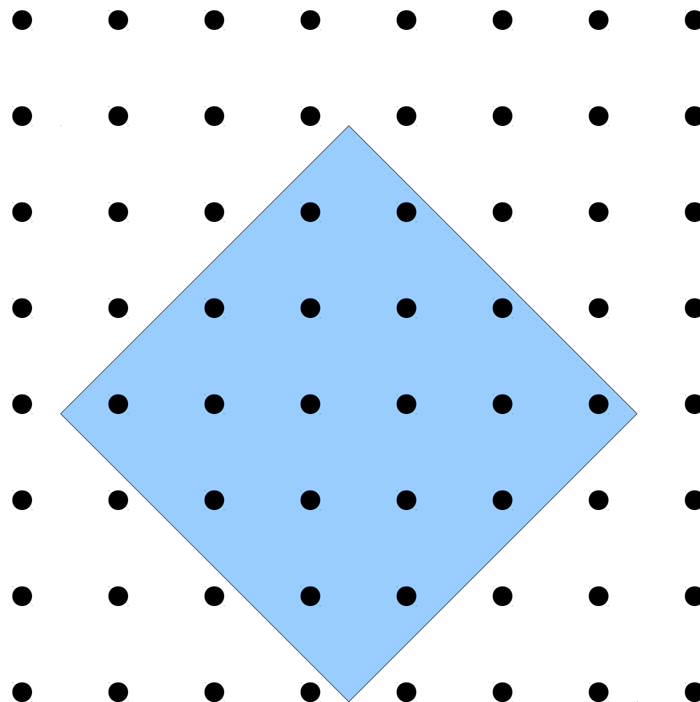
# Iteration Spaces

- The **iteration space** of a set of loops is the set of valid indices taken by the loop counters.



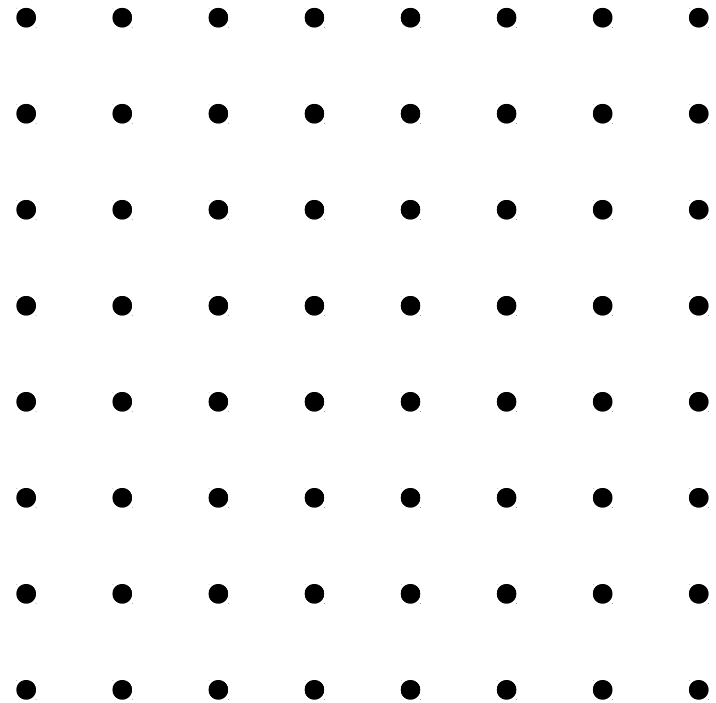
# Iteration Spaces

- The **iteration space** of a set of loops is the set of valid indices taken by the loop counters.



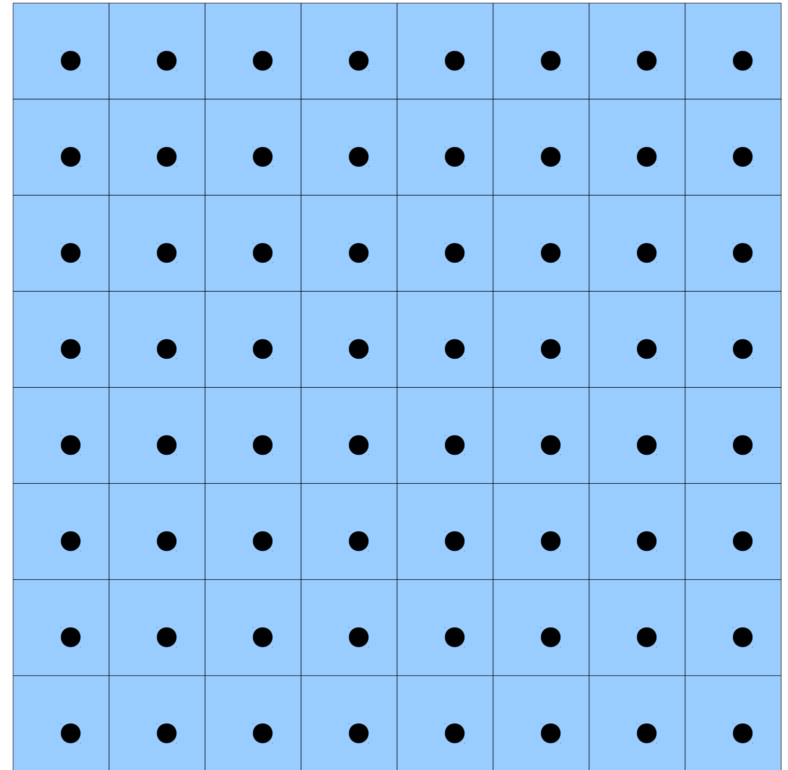
# Identifying Dependent Accesses

```
for (int i = 0; i < 8; ++i)
    for (int j = 0; j < 8; ++j)
        arr[i][j] = 0;
```



# Identifying Dependent Accesses

```
for (int i = 0; i < 8; ++i)
    for (int j = 0; j < 8; ++j)
        arr[i][j] = 0;
```



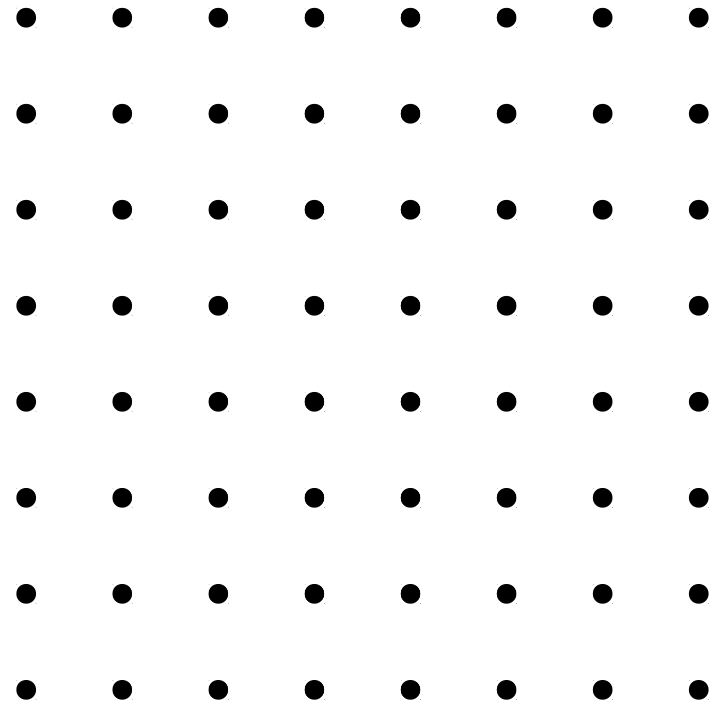
# Identifying Dependent Accesses

# Identifying Dependent Accesses

```
for (int i = 0; i < 8; ++i)
    for (int j = 1; j < 8; ++j)
        arr[i][j] = arr[i][j-1];
```

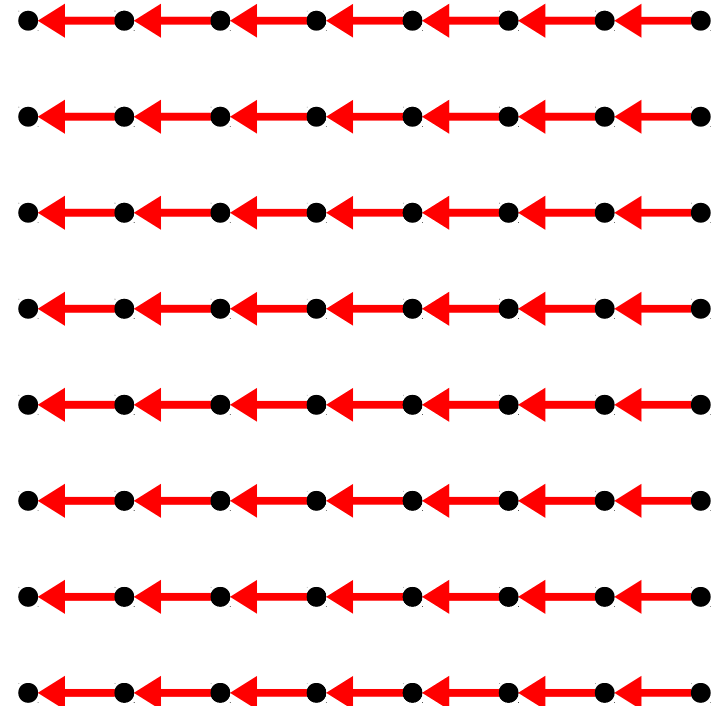
# Identifying Dependent Accesses

```
for (int i = 0; i < 8; ++i)
    for (int j = 1; j < 8; ++j)
        arr[i][j] = arr[i][j-1];
```



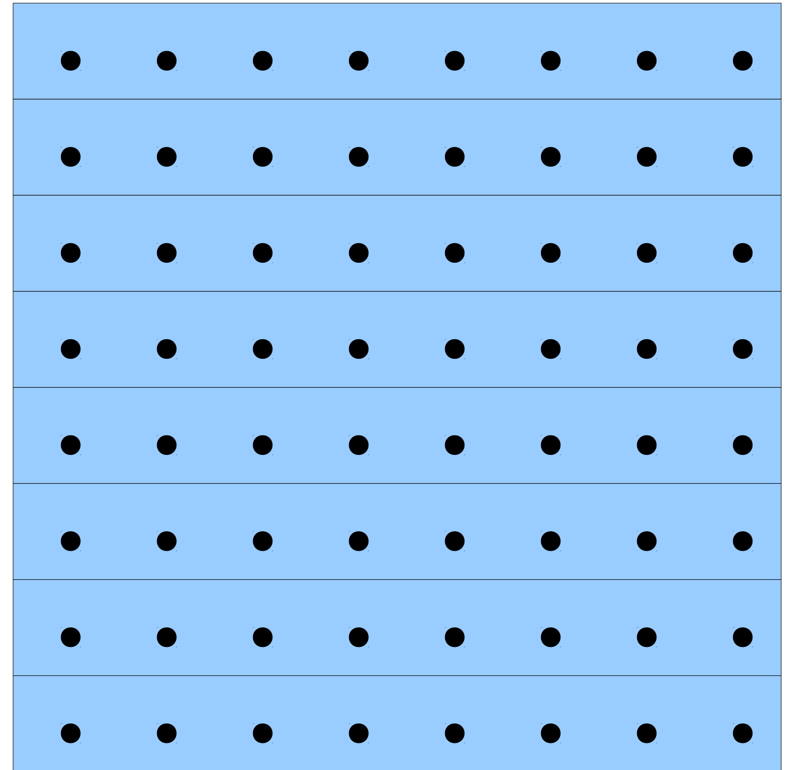
# Identifying Dependent Accesses

```
for (int i = 0; i < 8; ++i)
    for (int j = 1; j < 8; ++j)
        arr[i][j] = arr[i][j-1];
```



# Identifying Dependent Accesses

```
for (int i = 0; i < 8; ++i)
    for (int j = 1; j < 8; ++j)
        arr[i][j] = arr[i][j-1];
```



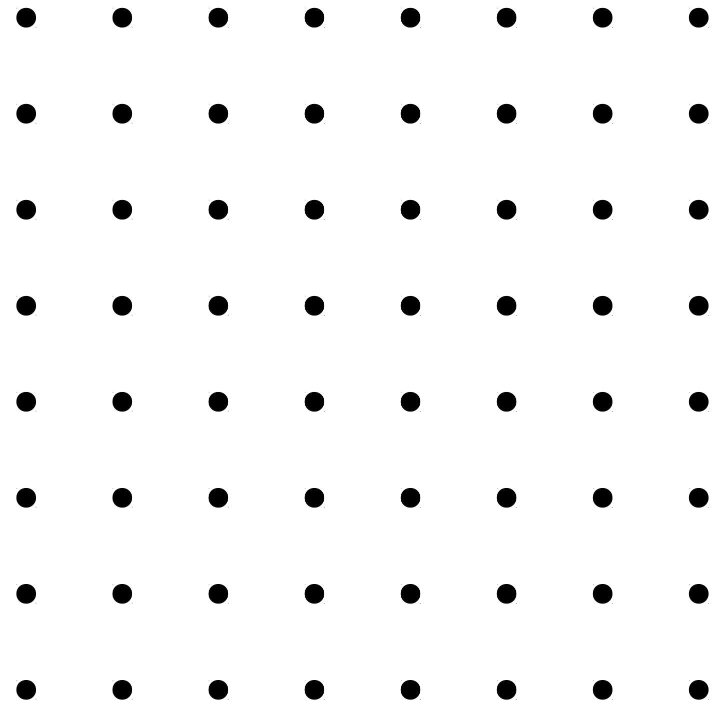
# Identifying Dependent Accesses

# Identifying Dependent Accesses

```
for (int i = 1; i < 8; ++i)
    for (int j = 1; j < 8; ++j)
        arr[i][i] = arr[i][j-1] +
                    arr[i-1][j] +
                    arr[i-1][j-1];
```

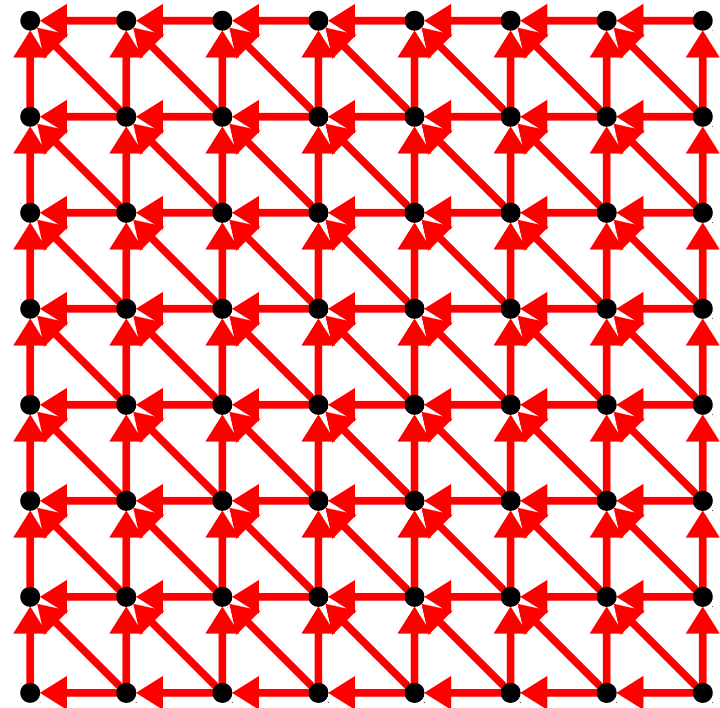
# Identifying Dependent Accesses

```
for (int i = 1; i < 8; ++i)
    for (int j = 1; j < 8; ++j)
        arr[i][i] = arr[i][j-1] +
                    arr[i-1][j] +
                    arr[i-1][j-1];
```



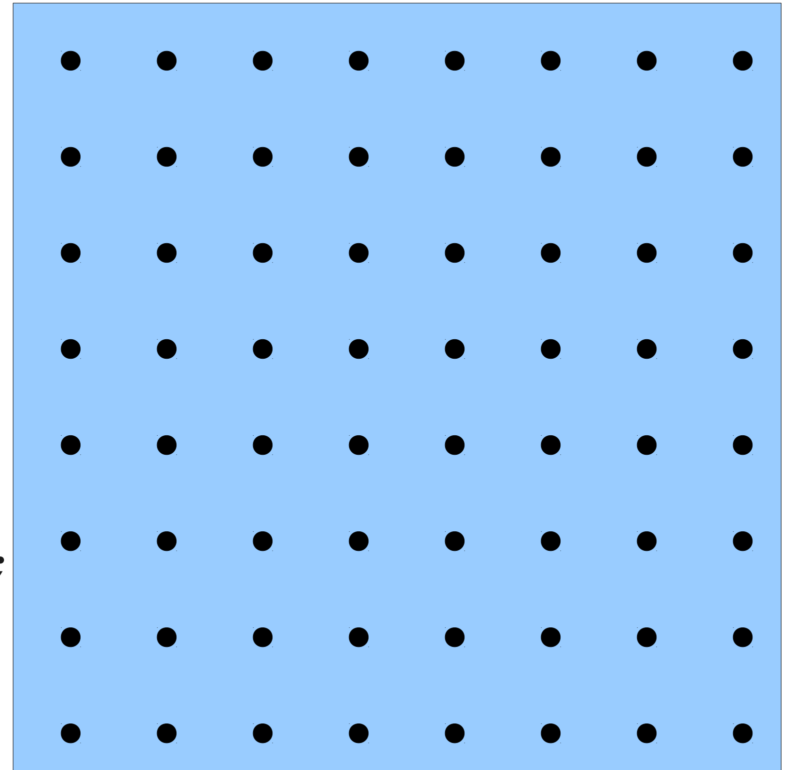
# Identifying Dependent Accesses

```
for (int i = 1; i < 8; ++i)
    for (int j = 1; j < 8; ++j)
        arr[i][i] = arr[i][j-1] +
                    arr[i-1][j] +
                    arr[i-1][j-1];
```



# Identifying Dependent Accesses

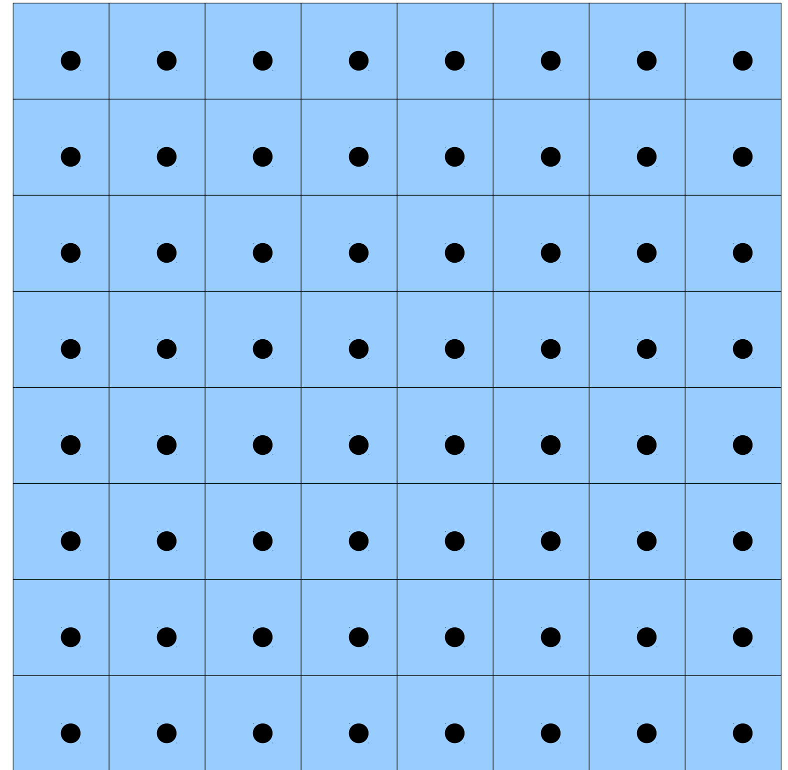
```
for (int i = 1; i < 8; ++i)
    for (int j = 1; j < 8; ++j)
        arr[i][i] = arr[i][j-1] +
                    arr[i-1][j] +
                    arr[i-1][j-1];
```



# Assigning Groups to Processors

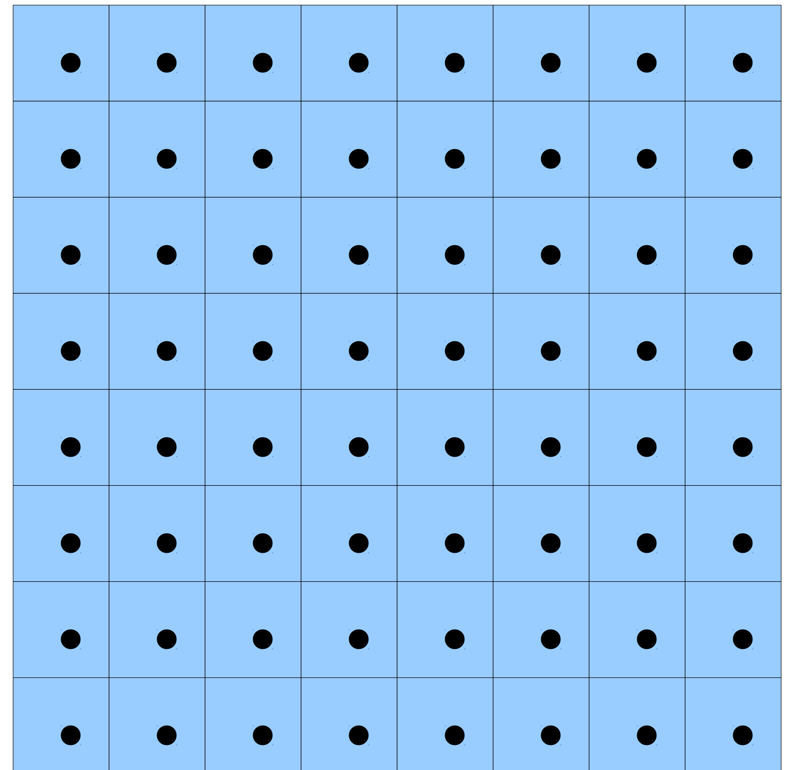
# Assigning Groups to Processors

```
for (int i = 0; i < 8; ++i)
    for (int j = 0; j < 8; ++j)
        arr[i][j] = 0;
```

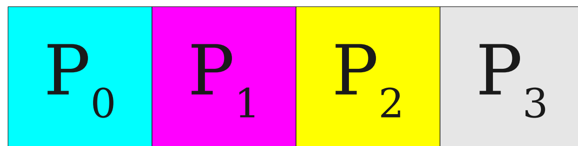


# Assigning Groups to Processors

```
for (int i = 0; i < 8; ++i)
    for (int j = 0; j < 8; ++j)
        arr[i][j] = 0;
```



Processors



# Assigning Groups to Processors

```
for (int i = 0; i < 8; ++i)
    for (int j = 0; j < 8; ++j)
        arr[i][j] = 0;
```

Processors

| $P_0$ | $P_1$ | $P_2$ | $P_3$ |
|---|---|---|---|

# Assigning Groups to Processors

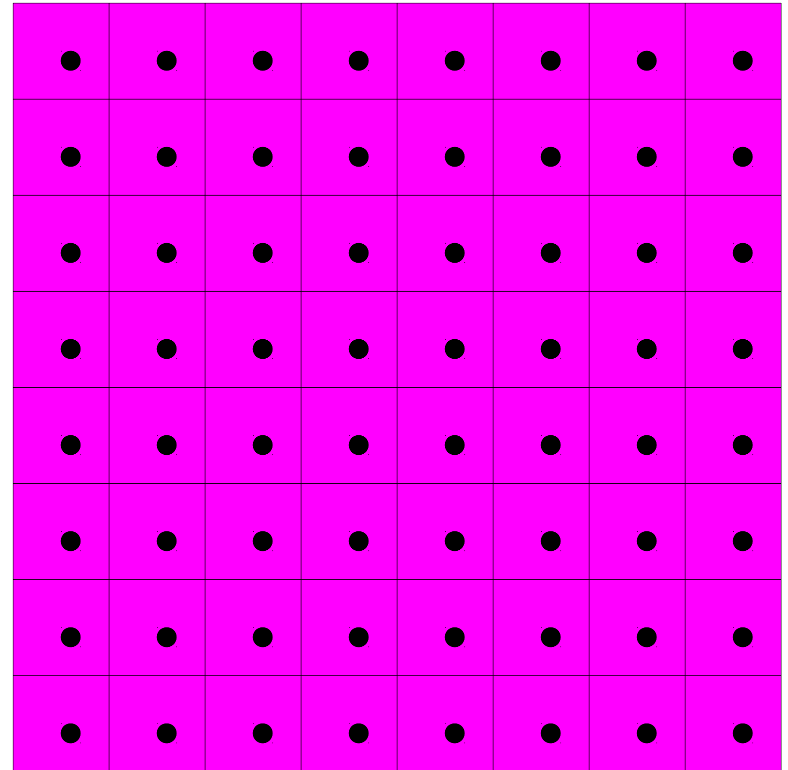
```
for (int i = 0; i < 8; ++i)
    for (int j = 0; j < 8; ++j)
        arr[i][j] = 0;
```

Processors

| $P_0$ | $P_1$ | $P_2$ | $P_3$ |
|---|---|---|---|

# Assigning Groups to Processors

```
for (int i = 0; i < 8; ++i)
    for (int j = 0; j < 8; ++j)
        arr[i][j] = 0;
```



Processors

# Assigning Groups to Processors

```
for (int i = 0; i < 8; ++i)
    for (int j = 0; j < 8; ++j)
        arr[i][j] = 0;
```

Processors

| $P_0$ | $P_1$ | $P_2$ | $P_3$ |
|---|---|---|---|

# Assignment Considerations

- When assigning iterations to processors:
  - Pick an assignment that is good for locality.
  - Pick an assignment that maximizes the degree of parallelism.
- In many cases, can be determined automatically!
  - See Ch. 11 of the Dragon Book.

# Assigning Groups to Processors

# Assigning Groups to Processors

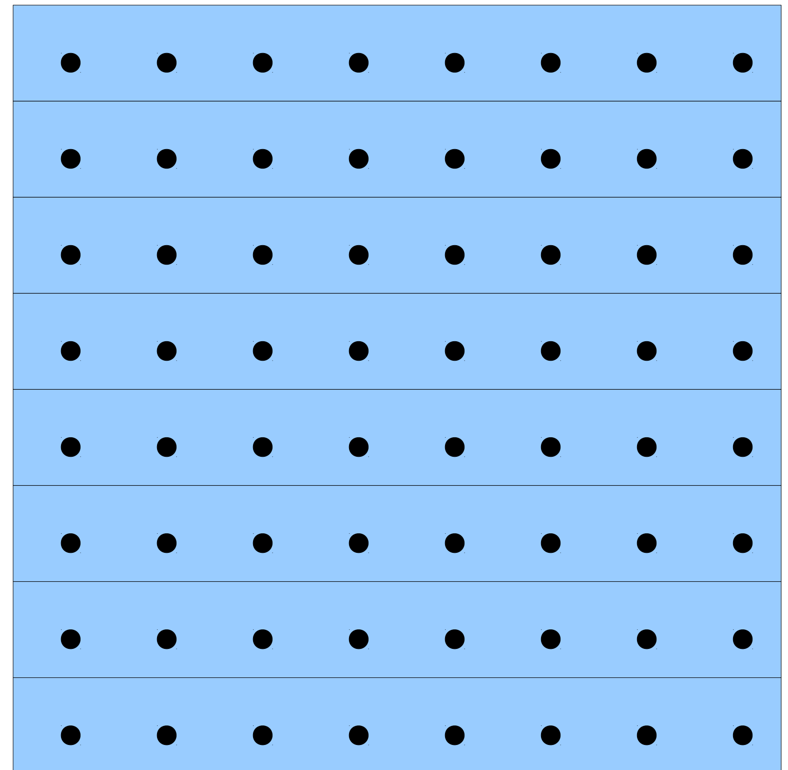
```
for (int i = 0; i < 8; ++i)
    for (int j = 1; j < 8; ++j)
        arr[i][j] = arr[i][j-1];
```

# Assigning Groups to Processors

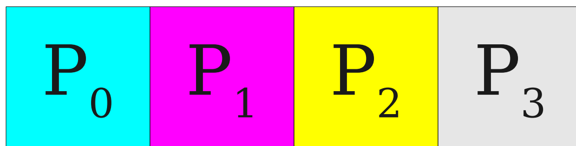
```
for (int i = 0; i < 8; ++i)
    for (int j = 1; j < 8; ++j)
        arr[i][j] = arr[i][j-1];
```

# Assigning Groups to Processors

```
for (int i = 0; i < 8; ++i)
    for (int j = 1; j < 8; ++j)
        arr[i][j] = arr[i][j-1];
```

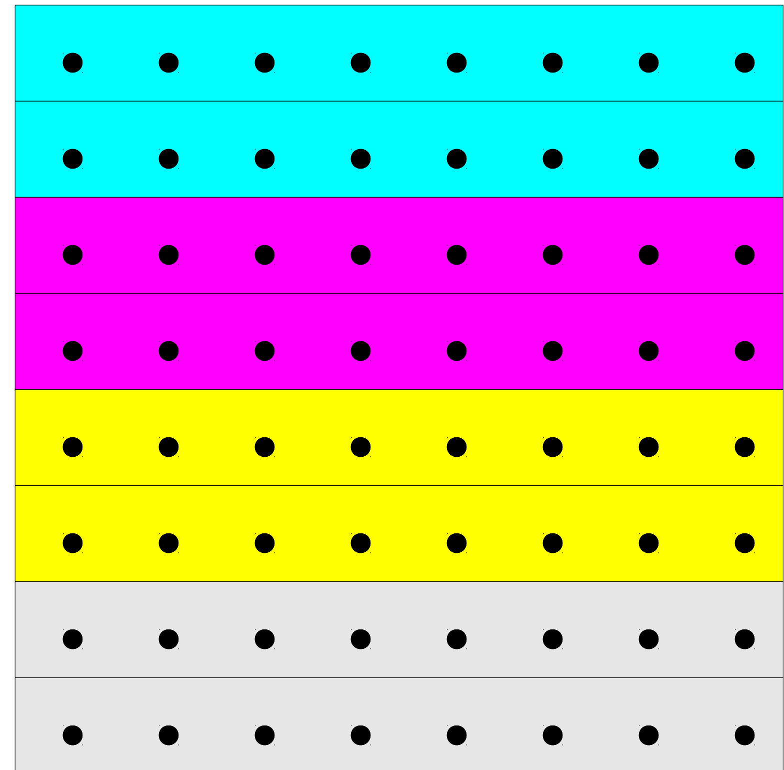


Processors

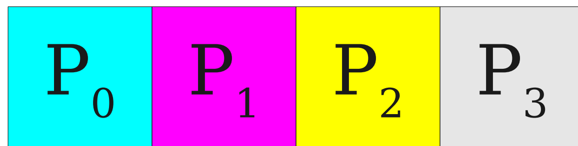


# Assigning Groups to Processors

```
for (int i = 0; i < 8; ++i)
    for (int j = 1; j < 8; ++j)
        arr[i][j] = arr[i][j-1];
```



Processors



# Assigning Groups to Processors

```
for (int i = 0; i < 8; ++i)
    for (int j = 1; j < 8; ++j)
        arr[i][j] = arr[i][j-1];
```

Processors

| $P_0$ | $P_1$ | $P_2$ | $P_3$ |
|---|---|---|---|

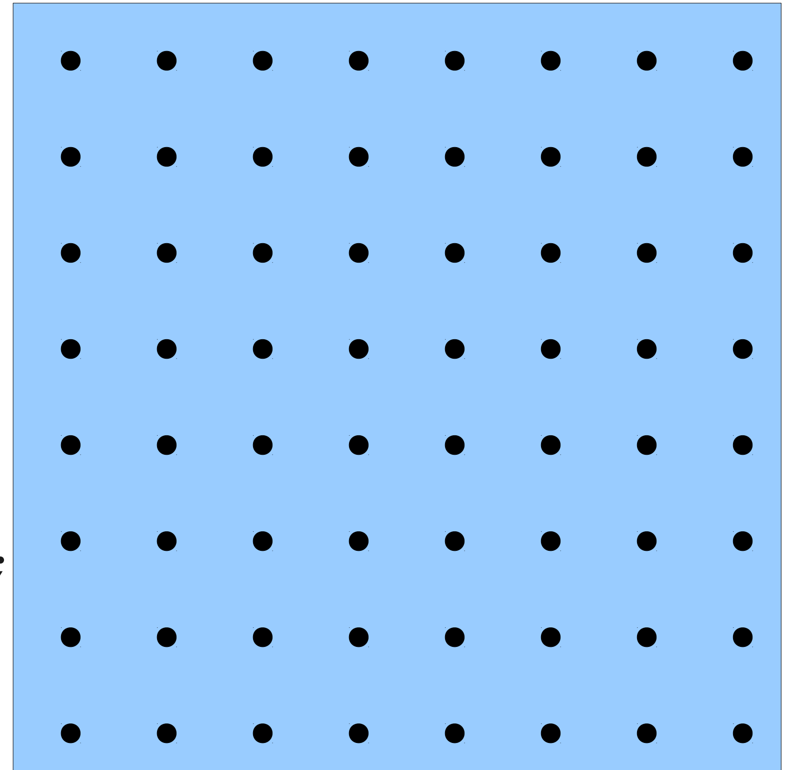
# Assigning Groups to Processors

# Assigning Groups to Processors

```
for (int i = 1; i < 8; ++i)
    for (int j = 1; j < 8; ++j)
        arr[i][i] = arr[i][j-1] +
                    arr[i-1][j] +
                    arr[i-1][j-1];
```

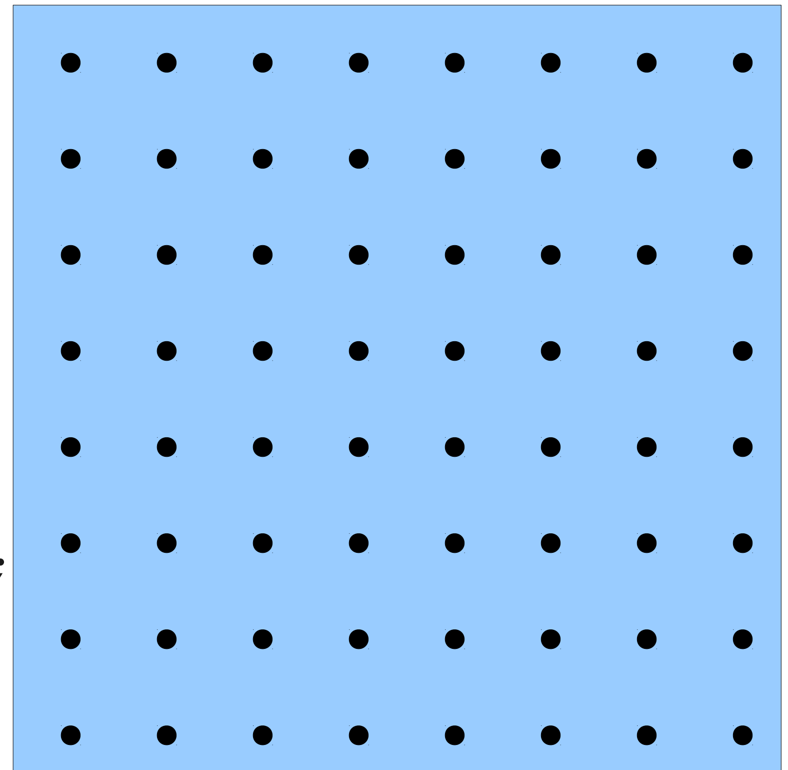
# Assigning Groups to Processors

```
for (int i = 1; i < 8; ++i)
    for (int j = 1; j < 8; ++j)
        arr[i][i] = arr[i][j-1] +
                    arr[i-1][j] +
                    arr[i-1][j-1];
```

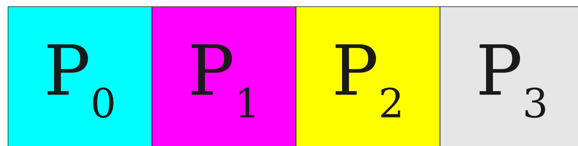


# Assigning Groups to Processors

```
for (int i = 1; i < 8; ++i)
    for (int j = 1; j < 8; ++j)
        arr[i][i] = arr[i][j-1] +
                    arr[i-1][j] +
                    arr[i-1][j-1];
```

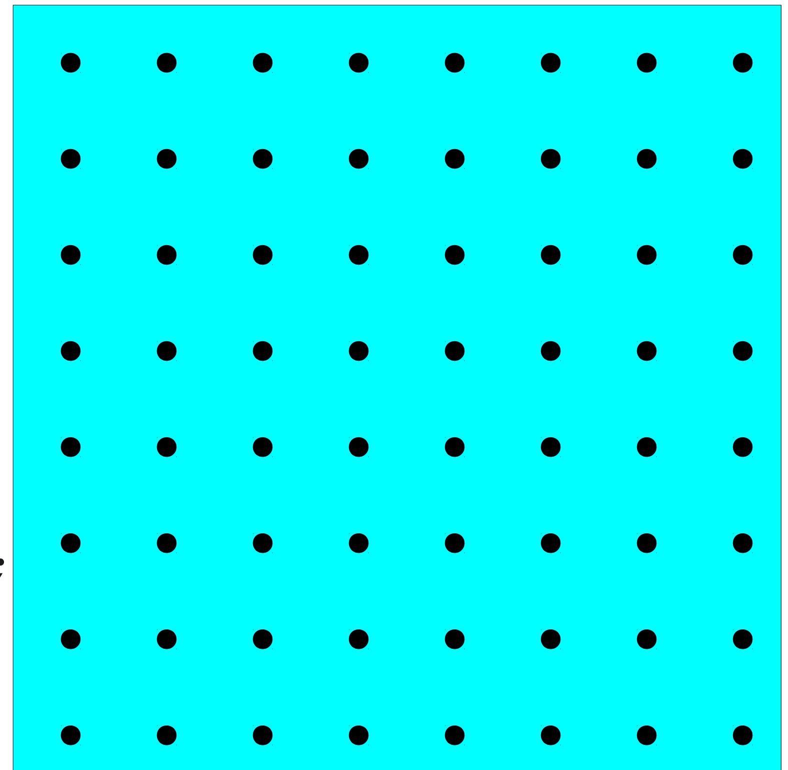


Processors

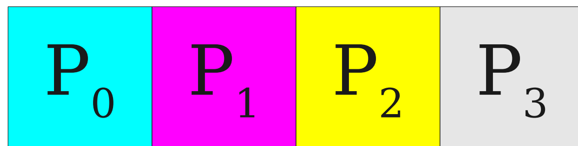


# Assigning Groups to Processors

```
for (int i = 1; i < 8; ++i)
    for (int j = 1; j < 8; ++j)
        arr[i][i] = arr[i][j-1] +
                    arr[i-1][j] +
                    arr[i-1][j-1];
```



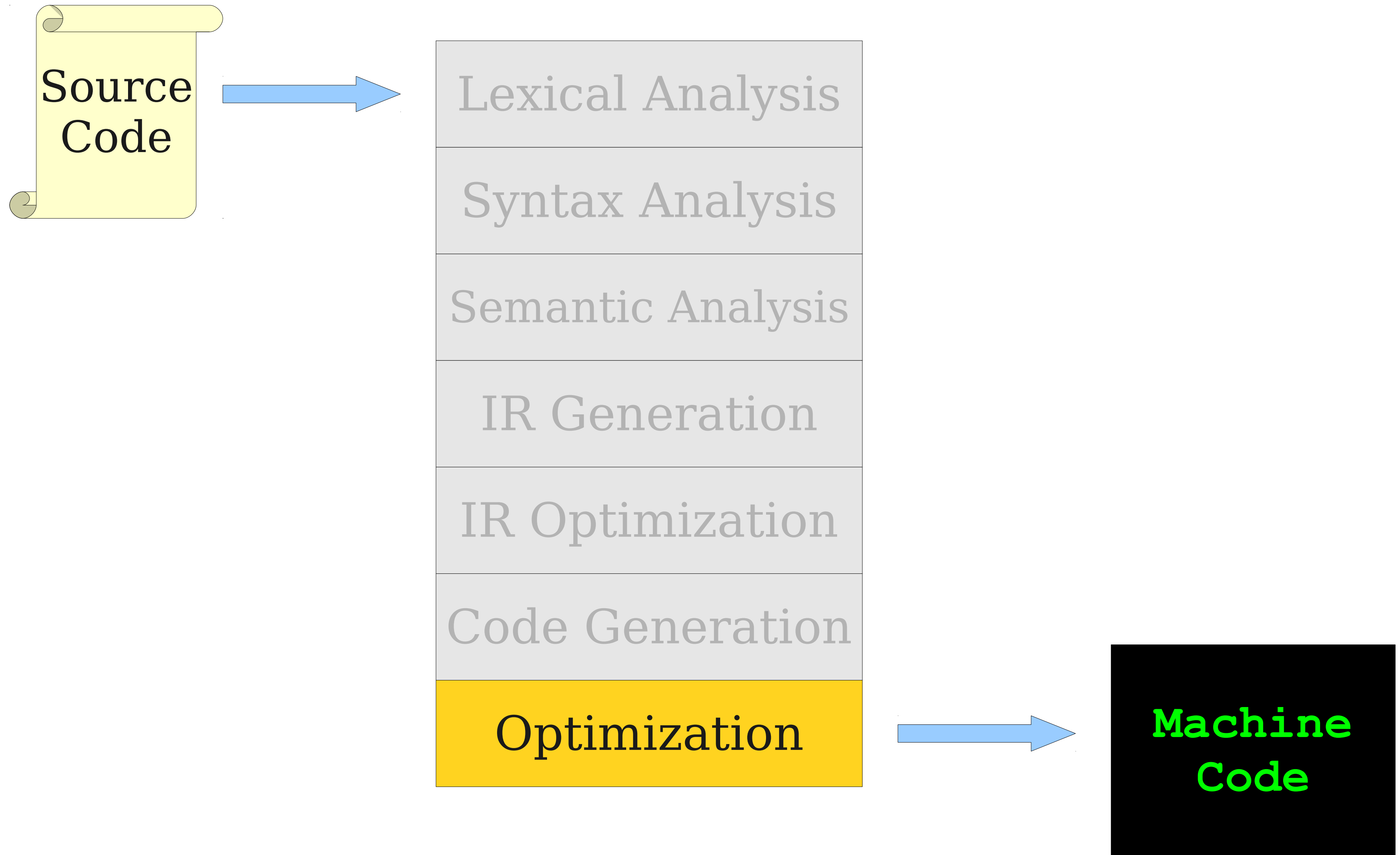
Processors



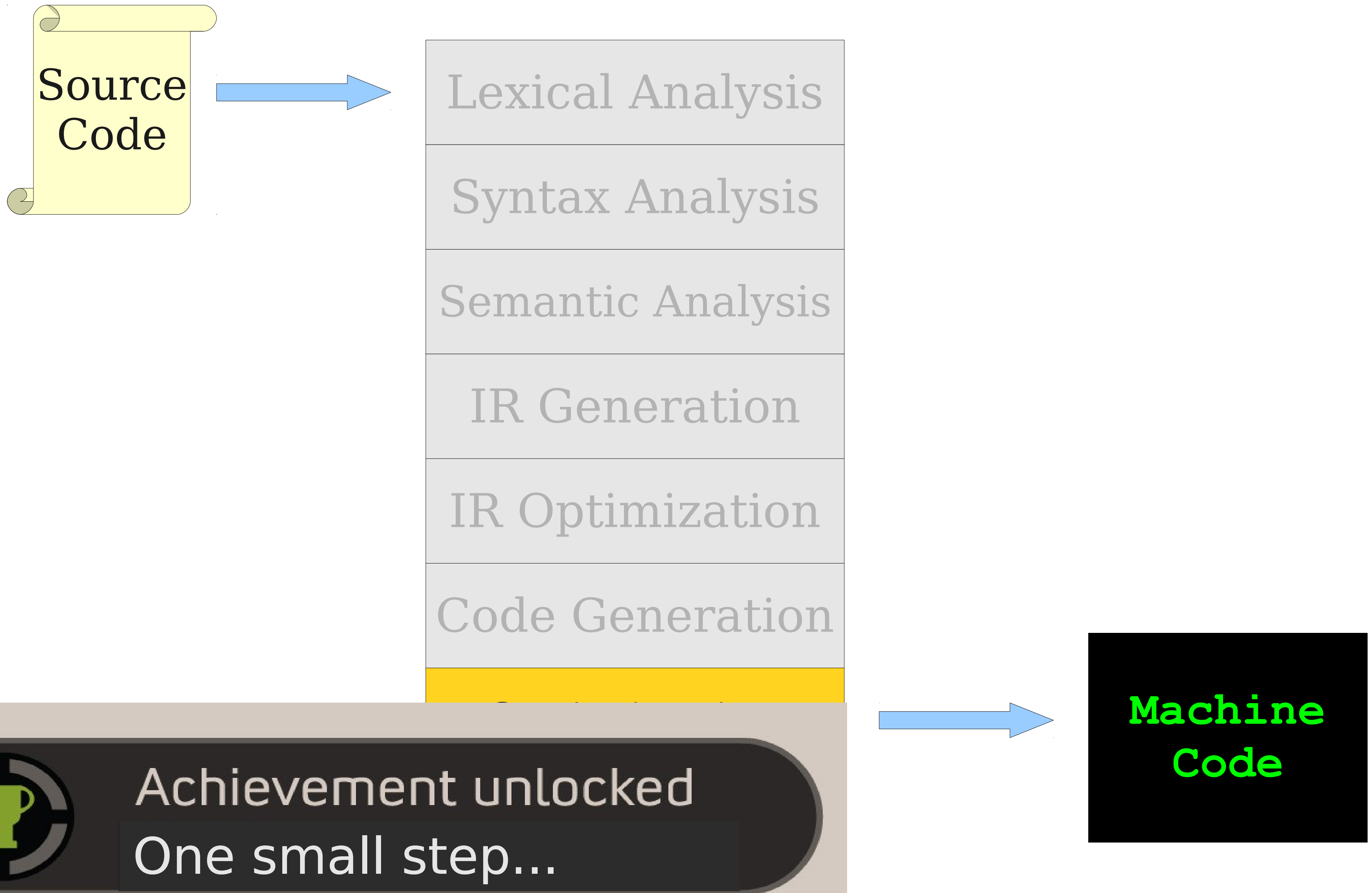
# Summary

- **Instruction scheduling** optimizations try to take advantage of the processor pipeline.
- **Locality** optimizations try to take advantage of cache behavior.
- **Parallelism** optimizations try to take advantage of multicore machines.
- There are *many more* optimizations out there!

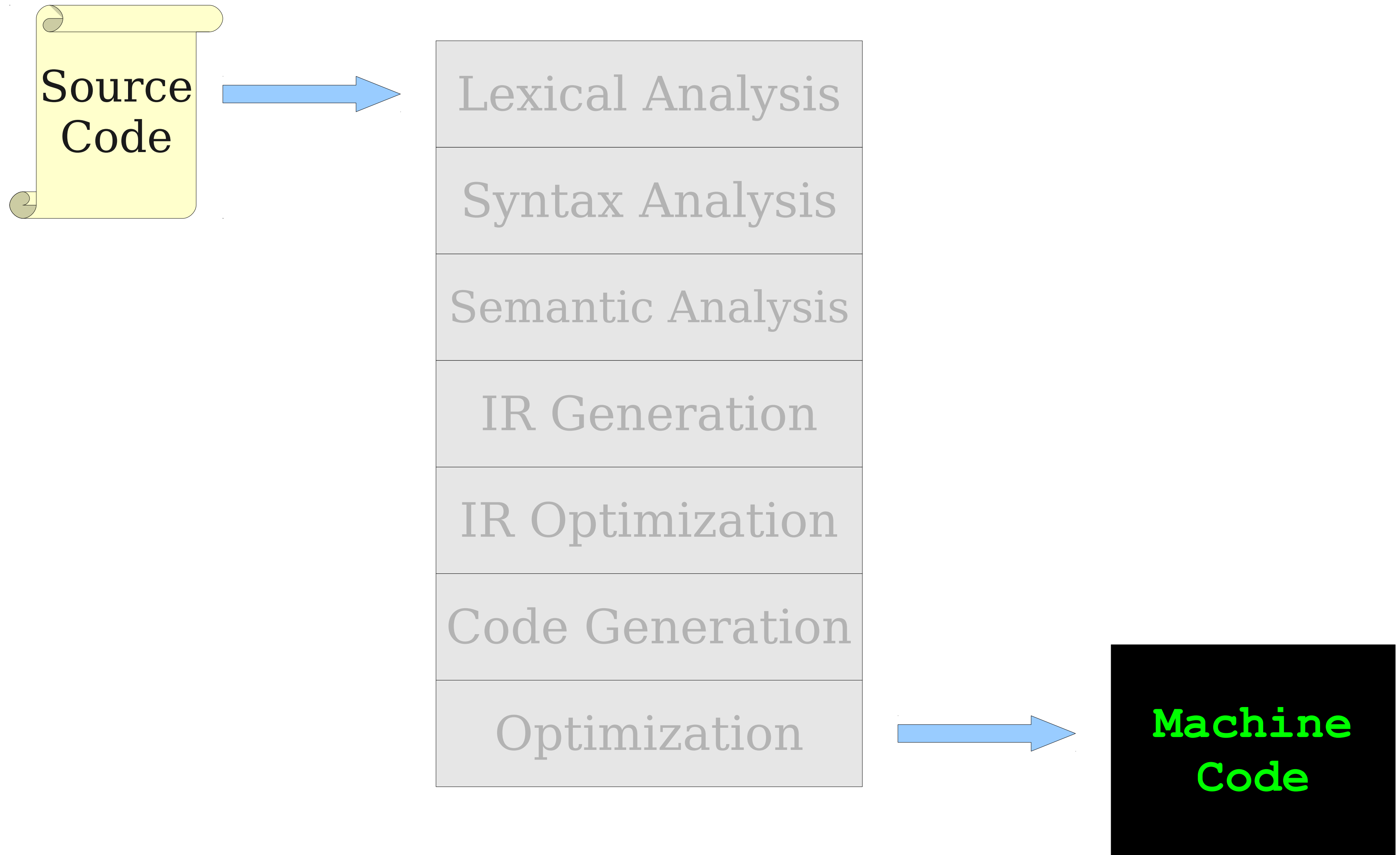
# Where We Are



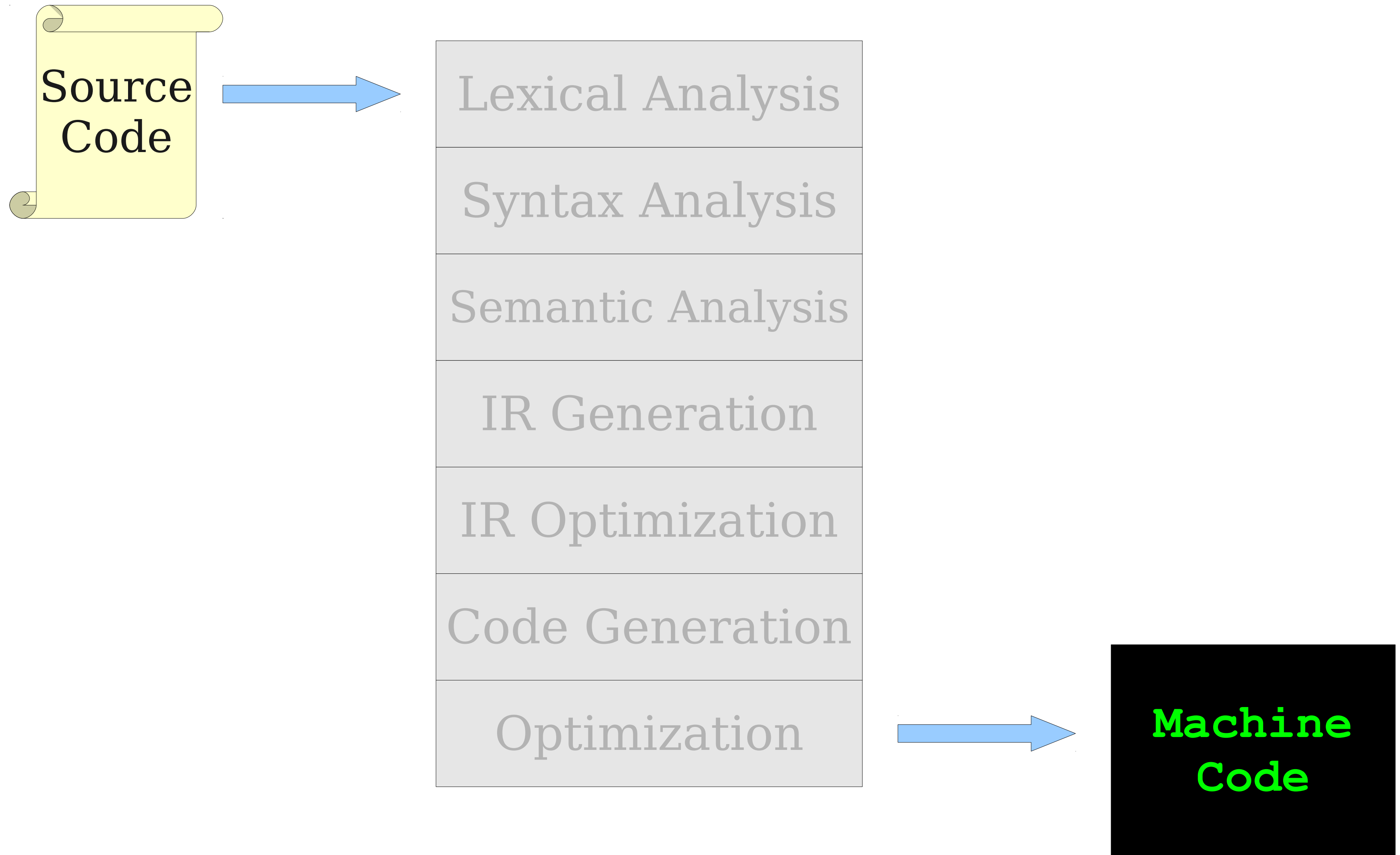
# Where We Are



# Where We Are



# Where We've Been



# Where We've Been

- Lexical Analysis
- Syntax Analysis
- Semantic Analysis
- IR Generation
- IR Optimization
- Code Generation
- Optimization

- Regular Expressions
- Finite Automata
- Maximal-Munch
- Subset Construction
- **`flex`**

# Where We've Been

- Lexical Analysis
- **Syntax Analysis**
- Semantic Analysis
- IR Generation
- IR Optimization
- Code Generation
- Optimization

- Context-Free Grammars
- Parse Trees
- ASTs
- Leftmost DFS
- LL(1)
- Handles
- LR(0)
- SLR(1)
- LR(1)
- LALR(1)
- Earley
- Earley-on-DFA
- **`bison`**

# Where We've Been

- Lexical Analysis
- Syntax Analysis
- **Semantic Analysis**
- IR Generation
- IR Optimization
- Code Generation
- Optimization

- Scope-Checking
- Spaghetti Stacks
- Function Overloading
- Type Systems
- Well-Formedness
- Null and Error Types
- Covariance

# Where We've Been

- Lexical Analysis
- Syntax Analysis
- Semantic Analysis
- **IR Generation**
- IR Optimization
- Code Generation
- Optimization

- Runtime Environments
- Function Stacks
- Closures
- Coroutines
- Parameter Passing
- Object Layouts
- Vtables
- Inline Caching
- TAC

# Where We've Been

- Lexical Analysis
- Syntax Analysis
- Semantic Analysis
- IR Generation
- **IR Optimization**
- Code Generation
- Optimization

- Basic Blocks
- Control-Flow Graphs
- Common Subexpression Elimination
- Copy Propagation
- Dead Code Elimination
- Arithmetic Simplification
- Constant Folding
- Meet Semilattices
- Transfer Functions
- Global Constant Propagation
- Partial Redundancy Elimination

# Where We've Been

- Lexical Analysis
- Syntax Analysis
- Semantic Analysis
- IR Generation
- IR Optimization
- **Code Generation**
- Optimization

- Memory Hierarchies
- Live Ranges
- Live Intervals
- Linear-Scan Register Allocation
- Register Interference Graphs
- Chaitin's Graph-Coloring Algorithm
- Reference Counting
- Mark-and-Sweep Collectors
- Baker's Algorithm
- Stop-and-Copy Collectors
- Generational Collectors

# Where We've Been

- Lexical Analysis
- Syntax Analysis
- Semantic Analysis
- IR Generation
- IR Optimization
- Code Generation
- **Optimization**

- Instruction Scheduling
- Loop Reordering
- Structure Peeling
- Automatic Parallelization

# Why Study Compilers? (Recap)

- Build a **large, ambitious software system**.
- See theory **come to life**.
- Learn how to **build programming languages**.
- Learn **how programming languages work**.
- Learn **tradeoffs in language design**.

# Where to Go from Here

# CS243: Program Analysis and Optimization

- In-depth treatment of optimization topics:
  - Dataflow framework.
  - Register allocation.
  - Garbage collection.
  - Instruction scheduling.
  - Locality optimizations.
  - Parallelization.
  - Interprocedural analysis.

# CS242: Programming Languages

- Survey of programming languages, their innovations, and their implementations:
  - Functional programming.
  - Object-oriented languages through history.
  - Runtime environments and optimizations.
  - Language-level security.
  - Modern language features.

# CS258: Intro to Programming Language Theory

- Mathematical exploration of programs and their properties:
  - Operational semantics.
  - Fixed-point operators and recursion.
  - Axiomatic semantics.
  - Formal algebras
  - Denotational semantics.

# CS343: Advanced Topics in Compilers

- Research-level topics in compilers and interpreters:
    - Building fast JITs.
    - Binary translation.
    - Dynamic and static analysis.
    - Sandboxing.
    - Superoptimizers.

My Email Address

**htiek@cs.stanford.edu**

# Final Thoughts