Tour of common optimizations
Simple example

```plaintext
foo(z) {
    x := 3 + 6;
    y := x - 5
    return z * y
}
```
Simple example

```c
foo(z) {
    x := 3 + 6;
    y := x - 5
    return z * y
}
```

- **Constant Propagation (CP)**
  - `x := 3 + 6;` → `g`
  - `y := x - 5` → `4 (CF)`

- **Arith simp**
  - `z <= 2` → `4 (CP)`

- **Strength Reduction**
  - `z * y` → `4 (CF)`
Another example

\[
x := a + b;
\]

\[
\ldots
\]

\[
y := a + b;
\]
Another example

\[ x := a + b; \]
\[ \ldots \]
\[ y := a + b; \]

\(\{\text{only if } x, a, b \text{ not modified!}\}\)
Another example

```plaintext
if (...) {
    a := read();
    x := a + b;
    print(x);
}

...

y := a + b;
```
Another example

```plaintext
if (...) {
    a := read();
    x := a + b;
    print(x);
} else { t := a + b }

...

y := a + b; t
```

Partial Redundancy
Elimination (PRE)
Another example

\[
x := y \\
... \\
z := z + x
\]
Another example

\[
x := y \\
\ldots \\
z := z + x \quad \text{\textit{\text{\color{red}\{ x, y not modified \}}}} \\
\text{\textit{\text{\color{red}copy prop}}}
\]
Another example

\[ x := y \]
\[ \ldots \]
\[ z := z + y \]

What if we run CSE now?
Another example

\[ x := y \]
\[ \ldots \]
\[ z := z + y \times \]

What if we run CSE now?
Another example

\[ x := y^{**z} \]

\[ \ldots \]

\[ x := \ldots \]
Another example

\[ x := y^*z \]

\[ \ldots \]

\[ \{ \text{if } x \text{ is not used} \]

\[ \text{dead assignment elim} \]

\[ \text{(unused assignment elim)} \]

• Often used as a clean-up pass

\[ x := y \]

\[ z := z + x \]

\[ \text{Copy prop} \]

\[ x := y \]

\[ z := z + y \]

\[ \text{DAE} \]

\[ x := y \]

\[ z := z + y \]
Another example

if (false) {

    ...

}
Another example

if (false) {
    ...
}

dead code elim
(unreachable code elim)

Another common clean up opt
Another example

- In Java:

```java
a = new int [10];
for (index = 0; index < 10; index ++) {
    a[index] = 100;
}
```
Another example

- In “lowered” Java:

```java
a = new int [10];
for (index = 0; index < 10; index ++) {
    if (index < 0 || index >= a.length()) {
        throw OutOfBoundsException;
    }
    a[index] = 0;
}
```
Another example

- In “lowered” Java:

```java
int[] a = new int[10];
for (int index = 0; index < 10; index++) {
    if (index < 0 || index >= a.length()) {
        throw new OutOfBoundsException;
    }
    a[index] = 0;
}
```
Another example

```plaintext
p := &x;
*p := 5
y := x + 1;
```
Another example

\textbf{pointer/alias analysis}

\begin{align*}
p & := \&x; \\
\textbf{x} & := 5; \\
\ast p & := 5 \\
x & := 5; \\
\ast p & := 3 \\
y & := x + 1; \quad ???
\end{align*}
Another example

for j := 1 to N
for i := 1 to M
a[i] := a[i] + b[j]
Another example

for j := 1 to N
  for i := 1 to M
    a[i] := a[i] + b[j]

Loop invariant
Code motion
Another example

\[
\text{area}(h,w) \{ \text{ return } h \times w \}
\]

\[
h := \ldots;
\]
\[
w := 4;
\]
\[
a := \text{area}(h,w)
\]
Another example

\[ \text{area}(h, w) \{ \text{return } h \times w \} \]

\[
\begin{align*}
    h &:= \ldots; \\
    w &:= 4; \\
    a &:= \text{area}(h, w) \\
    h &\times w \\
    h &\times 4 \\
    h &\ll 2
\end{align*}
\]

Many "silly" opts became important after inlining
Optimization themes

• Don’t compute if you don’t have to
  – unused assignment elimination

• Compute at compile-time if possible
  – constant folding, loop unrolling, inlining

• Compute it as few times as possible
  – CSE, PRE, PDE, loop invariant code motion

• Compute it as cheaply as possible
  – strength reduction

• Enable other optimizations
  – constant and copy prop, pointer analysis

• Compute it with as little code space as possible
  – unreachable code elimination