# CS614: Advanced Compilers

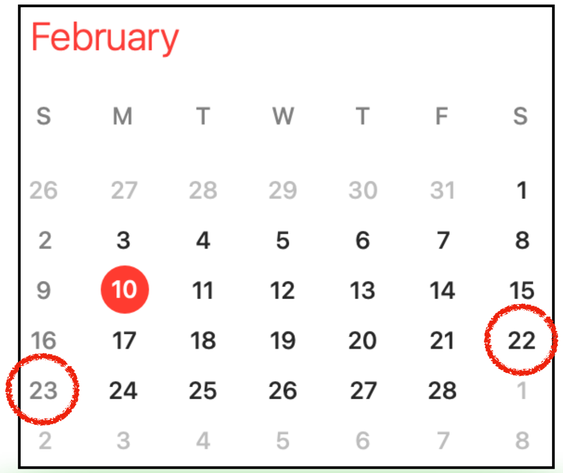
*Optimizations based on SSA*

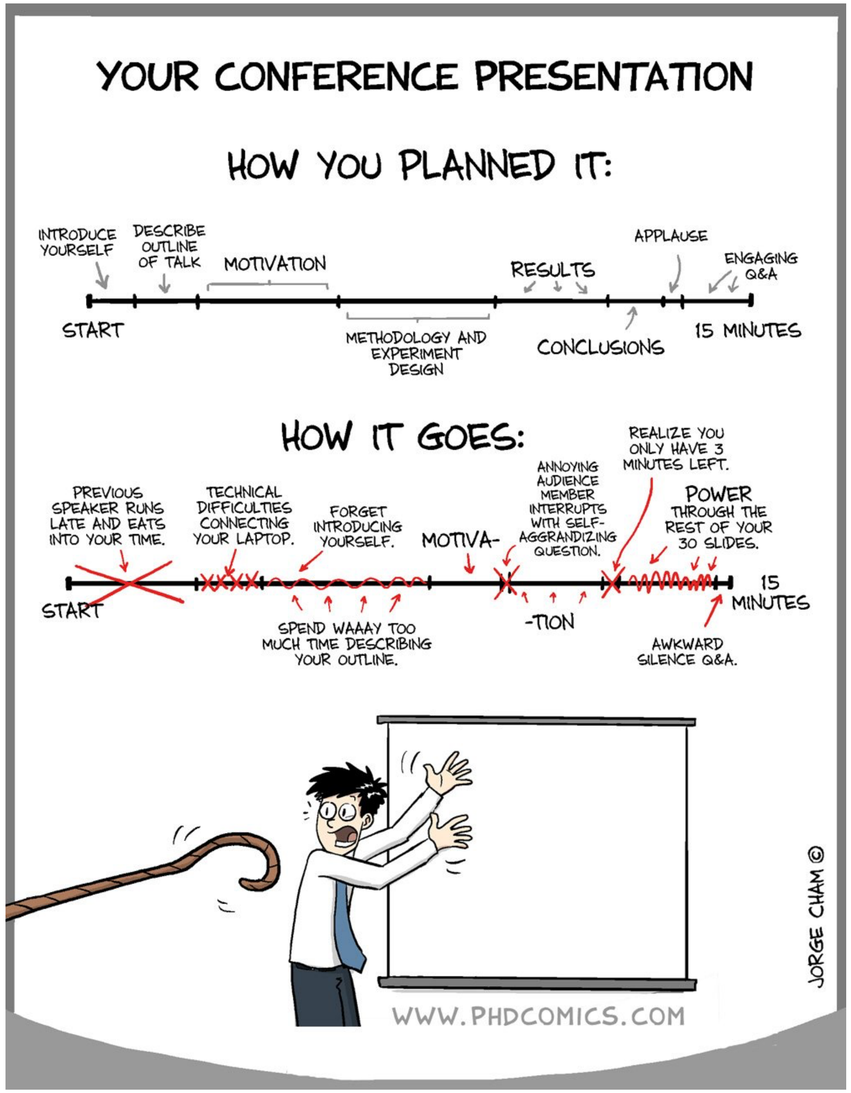
**Manas Thakur**
CSE, IIT Bombay


Spring 2025

# Outline of the recent weeks

- Intermediate code for OO languages
- CFGs and IDFAs
- SSA form: construction, optimizations
- SCP, SSCP, CCP, SCCP
- CSE, GVN, PRE
- Today:
  - PRE Practice
  - SSA Demo
  - SSA Goodbye



# Clarification: Inserting computations to introduce full redundancy

- Among the partially available expressions at $n$, we can place those at its entry that
  - are either anticipated locally, or can be placed at its exit (`PpOut`) and don't get killed; and
  - for every predecessor $p$, are either available at $p$'s exit or can be placed at $p$'s exit.

Insertion happens only at the exit of a block;
hence `PPIn` has expressions that can be pushed to the exit of predecessors.

```
PpIn[n] = PavIn[n] ∩ (AntLoc[n] ∪ (PpOut[n] - Kill[n]))
                   ∩ ∀p∈pred[n] (AvOut[p] ∪ PpOut[p])
```

# Where do we insert new computations then?!

- We don't want to insert an expression at node $n$ if it can be placed at a predecessor of $n$.
- Insert an expression $e$ at the exit of a node $n$ if
  - Exit of $n$ is a possible placement point for $e$;
  - $e$ is not already available at $n$; and
  - moving $e$ further up does not work because either e cannot be placed at $n$'s entry or because $n$ kills $e$.

```
Insert[n] = PpOut[n] ∩ !AvOut[n] ∩ (!PpIn[n] ∪ Kill[n])
```

# Finally, removing existing computations

- We have identified partial redundancies and added expressions to convert them to full redundancies.
- Now we can remove full redundancies!

- From a node $n$, we can remove the computation of expressions that are anticipated locally and can be placed at $n$'s beginning:

```
Remove[n] = AntLoc[n] ∩ PpIn[n]
```

# PRE Practice

$$AvIn[n] = \forall p \in pred[n] \cap AvOut[p]$$

$$AvOut[n] = Gen[n] \cup (AvIn[n] - Kill[n])$$

$$PavIn[n] = \forall p \in pred[n] \cup PavOut[p]$$

$$PavOut[n] = Gen[n] \cup (PavIn[n] - Kill[n])$$

$$AntOut[n] = \forall s \in succ[n] \cap AntIn[s]$$

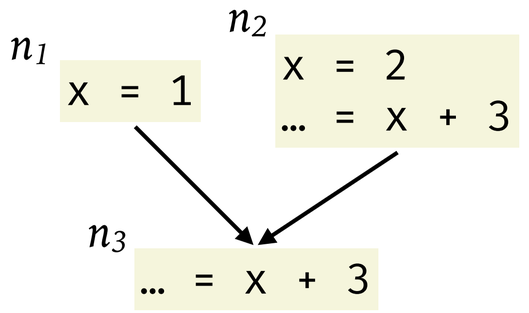
$$AntIn[n] = AntLoc[n] \cup (AntOut[n] - Kill[n])$$

$$PPOut[n] = \forall s \in succ[n] \cap PPIn[s]$$

$$PpIn[n] = PavIn[n] \cap (AntLoc[n] \cup (PpOut[n] - Kill[n])) \cap \forall p \in pred[n] (AvOut[p] \cup PpOut[p])$$

$$Insert[n] = PpOut[n] \cap !AvOut[n] \cap (!PpIn[n] \cup Kill[n])$$

$$Remove[n] = AntLoc[n] \cap PpIn[n]$$



# PRE Notes

- Our PRE algorithm (1979) is bidirectional:

$$PPOut[n] = \forall s \in succ[n] \cap PPIn[s]$$

$$PpIn[n] = PavIn[n] \cap (AntLoc[n] \cup (PpOut[n] - Kill[n])) \cap \forall p \in pred[n] (AvOut[p] \cup PpOut[p])$$

  - This makes it complex as well as expensive.
- Researchers have later proposed other formulations of PRE that are combinations of multiple unidirectional analyses.
- A paper as recently as 2024 on reducing the number of required *unidirectional* analyses!
- Think how one could utilize SSA form to improve PRE.
  - Called the SSAPRE algorithm.

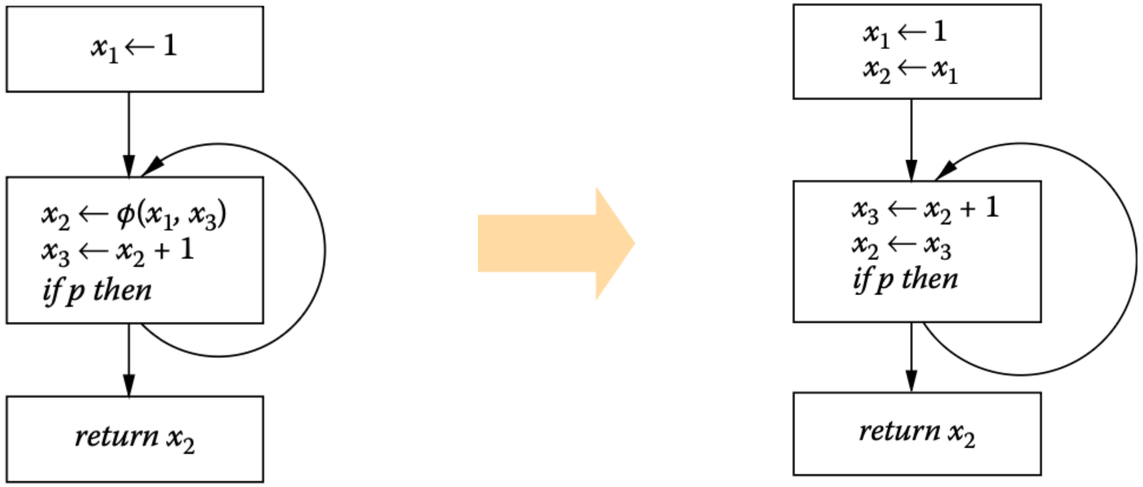
# SSA Destruction

# Removal of Φ Functions

- Φ functions have no equivalence in hardware.
- They need to be removed after performing enabled optimizations.
- Naive removal:
  - Insert a copy statement at the end of each predecessor node, corresponding to each argument:


  - Not so straightforward in multiple scenarios though.

# The Lost Copy Problem



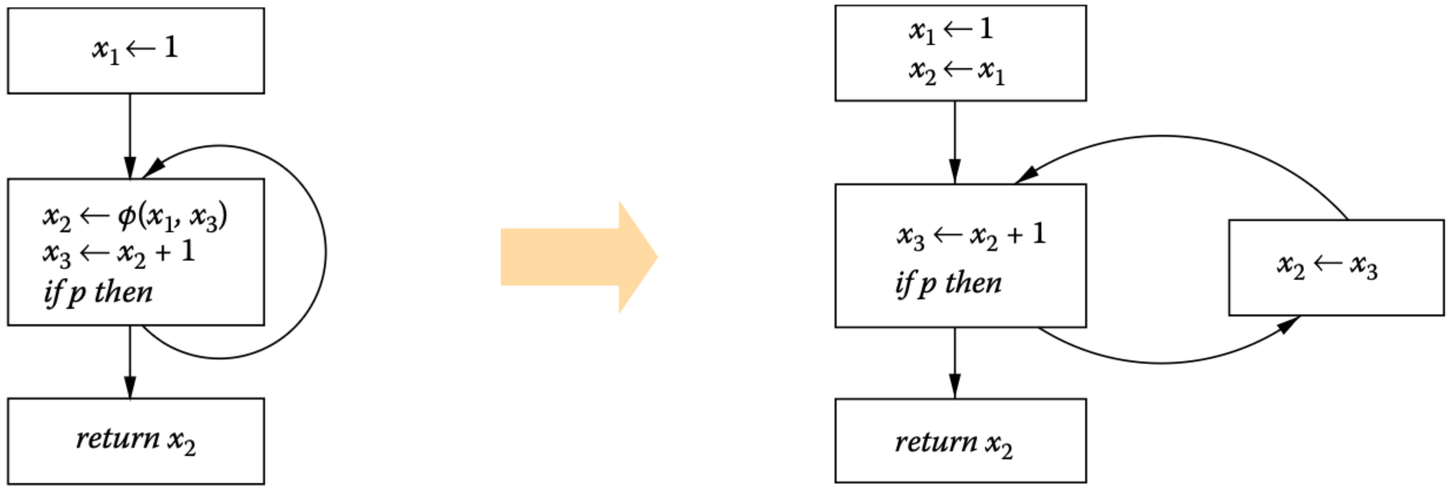

- Wrong value of $x2$ will be returned by the enclosing function.

# The Lost Copy Problem

- Cause: $x2$ was live beyond the scope of the $\Phi$ function that defined it.
- Solution 1: Introduce a temporary variable to save the original value of $x2$:

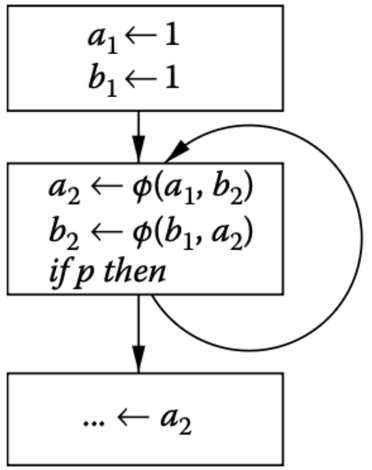
# The Lost Copy Problem

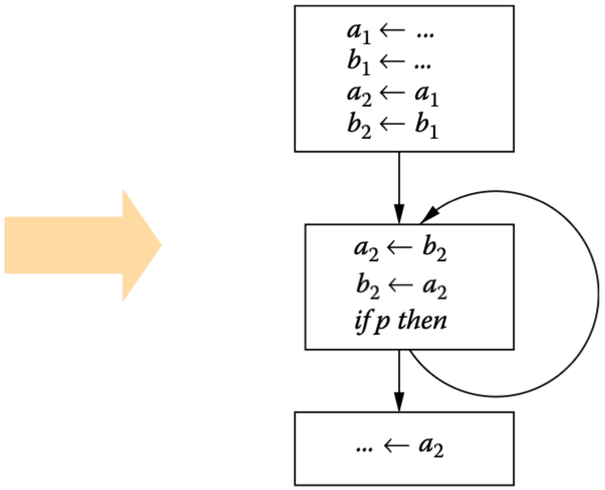
- Cause: *x2* was live beyond the scope of the Φ function that defined it.
- Solution 2: Split the *critical* edge and insert the overwrite in a new basic block:

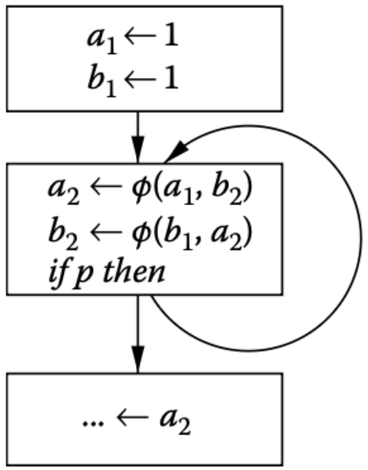


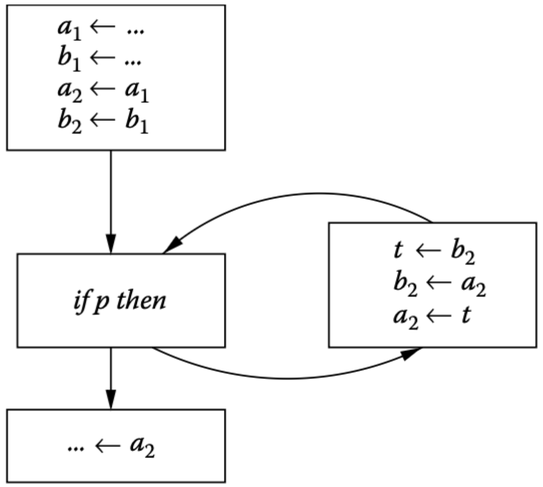
# The Swap Problem

- *a2* gets assigned the value of *b2* instead of *a1*, in the first iteration.
- Can be again solved with edge splitting:




Insight: The Φ functions at a node have no inherent order.

# SSA Demo

Bhavya Hirani
NIT Surat
Intern, CompL

# A2 Discussion