Dynamic Speculation-Synchronization of Data Dependences

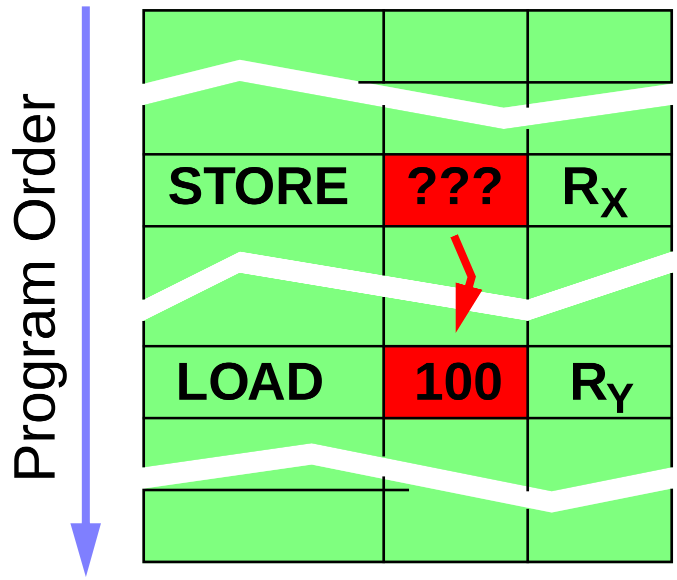
Guri Sohi

work with :

Andreas Moshovos, Scott Breach, T. N. Vijaykumar

Computer Sciences Department
University of Wisconsin — Madison
URL: http://www.cs.wisc.edu/~sohi

Overview



Dependence ?

YES	LOAD has to wait
NO	LOAD may execute immediately

opportunity for higher ILP

Unfortunately don't know in advance!

Solution: Speculate whether a dependence exists

Overview

So far two policies:

Speculate Never

Safe but loss of opportunity

Speculate Always (blind)

Penalty when wrong, but it pays (today)

Argue:

As the window size increases

Net penalty of mis-speculation becomes significant

Room for significant improvement over both policies

Our Solution:

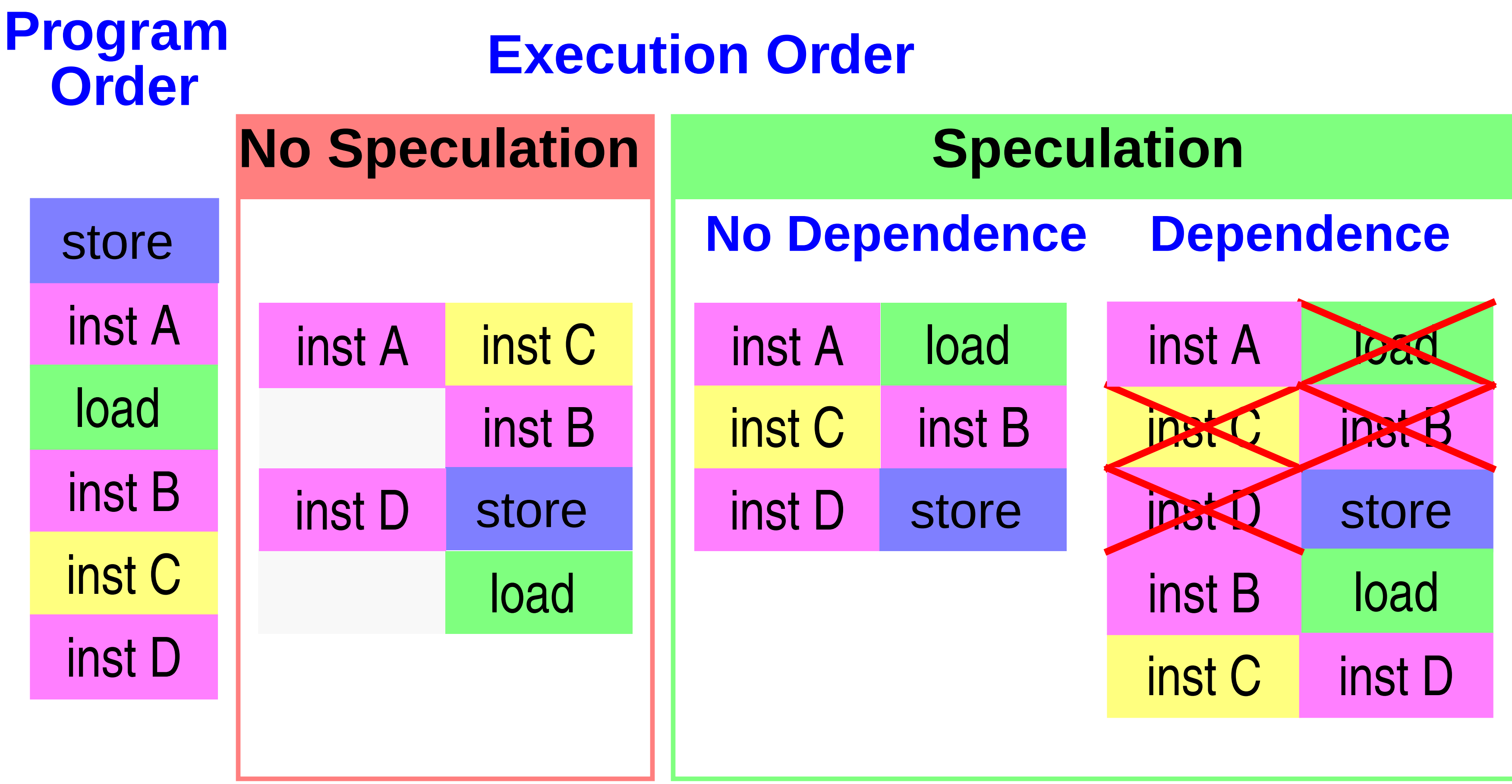
(1). Predict Dependences

(2). Force Synchronization

Roadmap

- ~~Overview~~
- **The Problem and Our Solution** (14 slides)
  - Dependence Speculation and Performance
  - Impact of Window Size
  - Ideal Solution - Alternatives
  - Our Solution
- Evaluation (7 slides)
- Other uses - Ongoing work

Dependence Speculation



Speculation may affect performance either way

Dependence Speculation and Performance

Performance

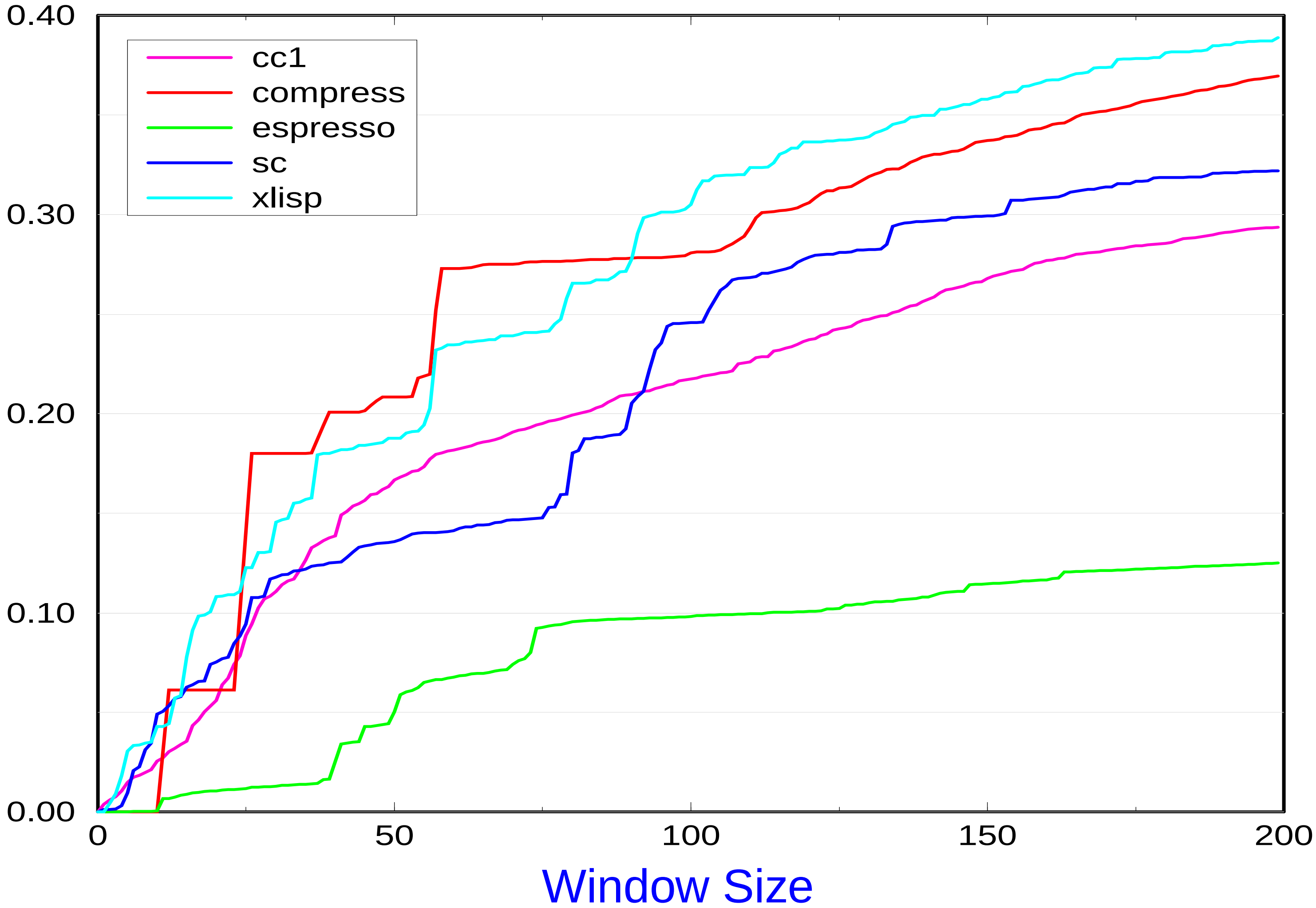
+ Gain × (100% - Mis-speculation%)

− Penalty × Mis-speculation%

↑	Gain	↓
↑	Penalty	↓
↑	Mis-speculation%	↓

- Balance between Gain and Penalty

Dependences vs. Window Size



Frequency of loads with Dependences within the Window

Small Instruction Windows and Speculation

Small Instruction Window:

- Loads are speculated past few instructions
- Dependences are infrequent

Blind Speculation a good choice:

- Mis-speculations are infrequent
- Low probality of other, independent work
- Low mis-speculation penalty

Not Speculating at times is acceptable.

Wider Instruction Windows

As the Window size increases:

- Loads are speculated past many more instructions
- Dependences become more frequent

Overall:

- Mis-speculations are more frequent
- Higher probability of other, independent work
- Higher mis-speculation penalty

Blind Speculation is still a viable approach

Not Speculating is not

HOWEVER! Net penalty of mis-speculation becomes significant

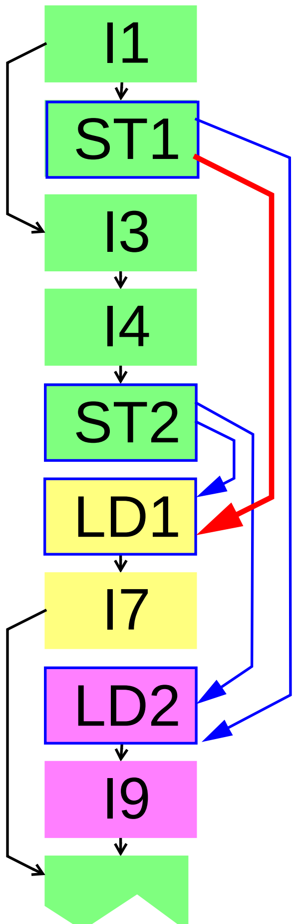
Potential for performance improvement

Reducing the Net Mis-speculation Penalty

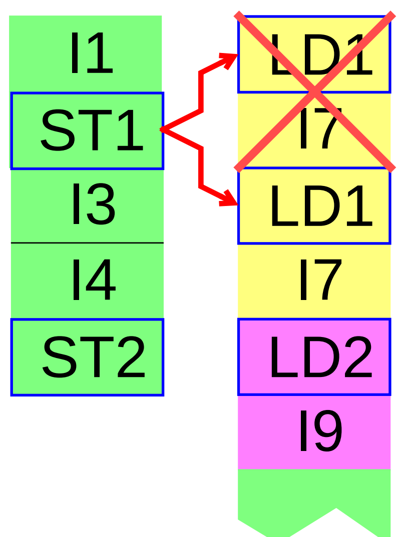
Ideally:

- Dependent load/store pairs are synchronized
- Other loads execute as early as possible

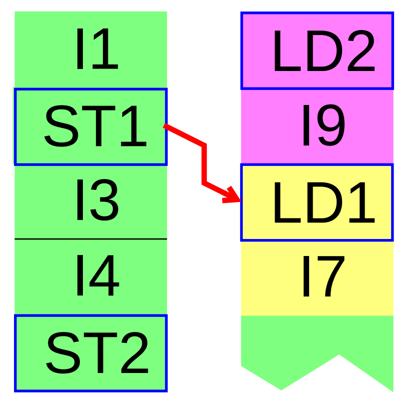
Code



Blind Speculation



Ideal Speculation



Dependence Speculation/Synchronization

To mimic the ideal we need:

(1). Identify the loads that have dependences

(2). Identify the relevant stores

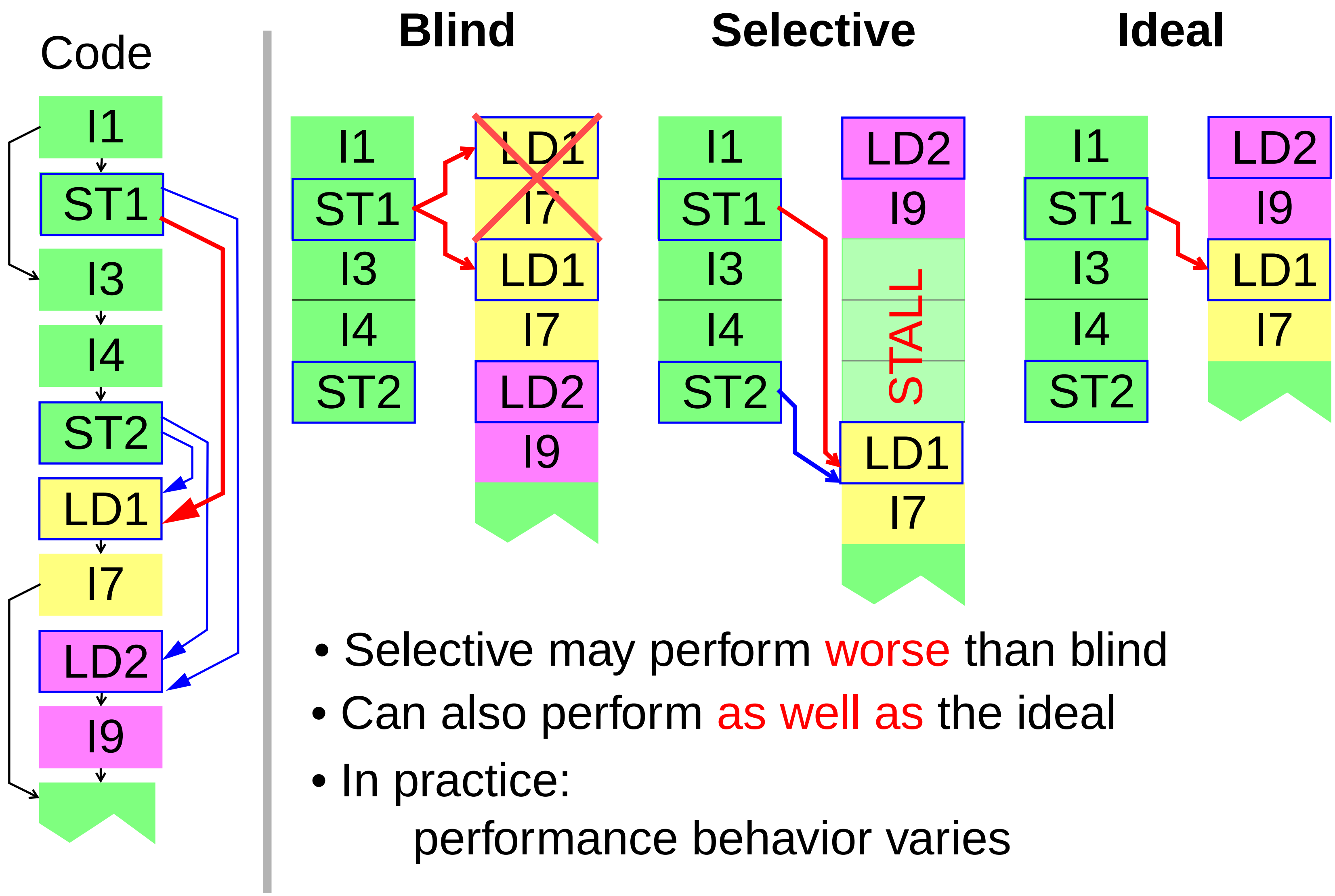
(3). Enforce synchronization

Can we do without synchronization?

How about selective speculation:

- Identify the loads that have dependences
- Do not speculate them

Selective Dependence Speculation



- Selective may perform worse than blind
- Can also perform as well as the ideal
- In practice:
  performance behavior varies

Our approach

Attempt to mimic the Ideal:

- To identify the dependent load/store pairs:

  Predict!

  Based on the history of mis-speculations

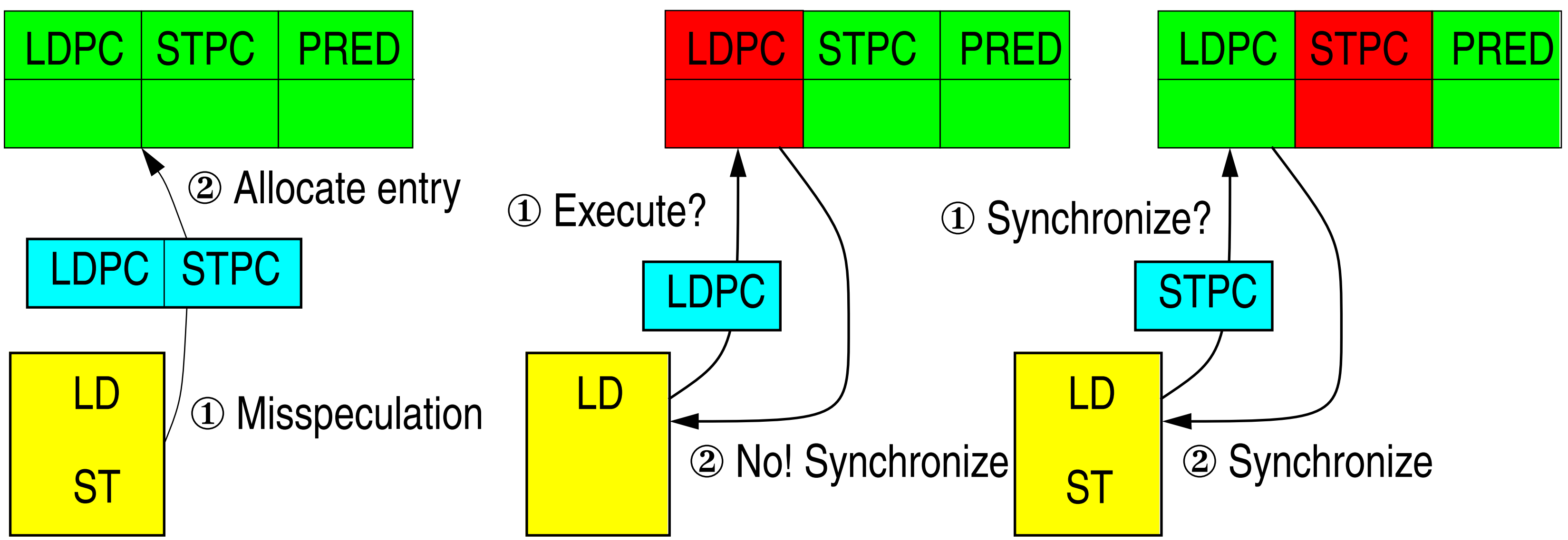
- To synchronize:

  Use **dynamically assigned** synchronization variables

Predicting Dependences

- Dependence: (Load PC, Store PC)
- Temporal locality - Small Working Set.
- Use a small table to: (1). track recent mis-speculations
  (2). Predict dependences

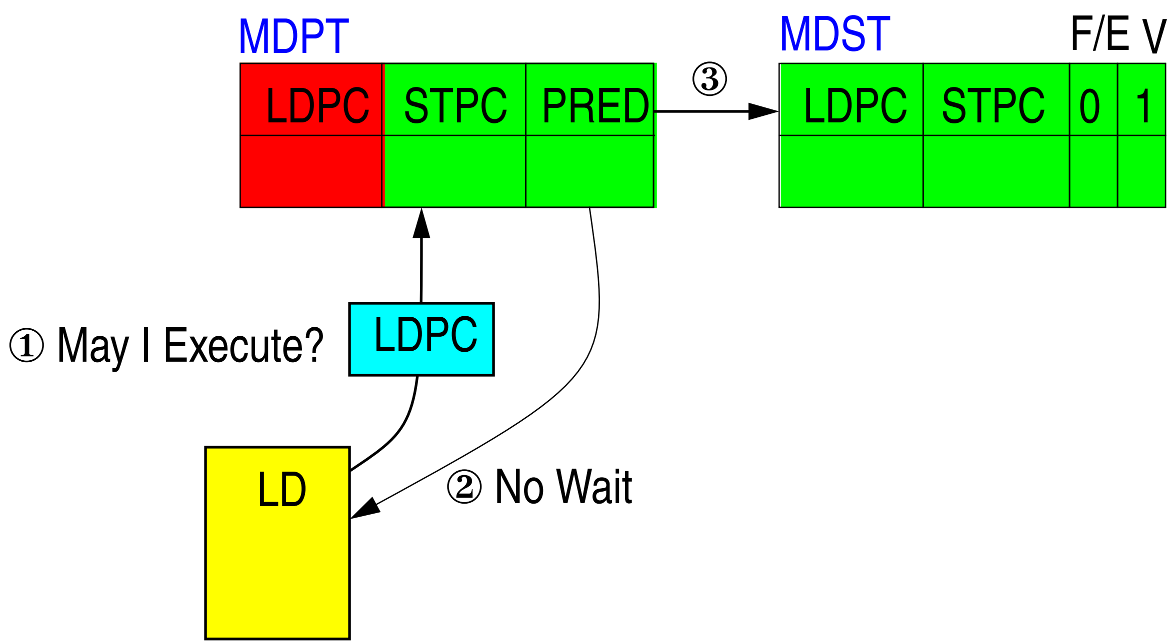
Memory Dependence Prediction Table



Synchronization - Load Waits

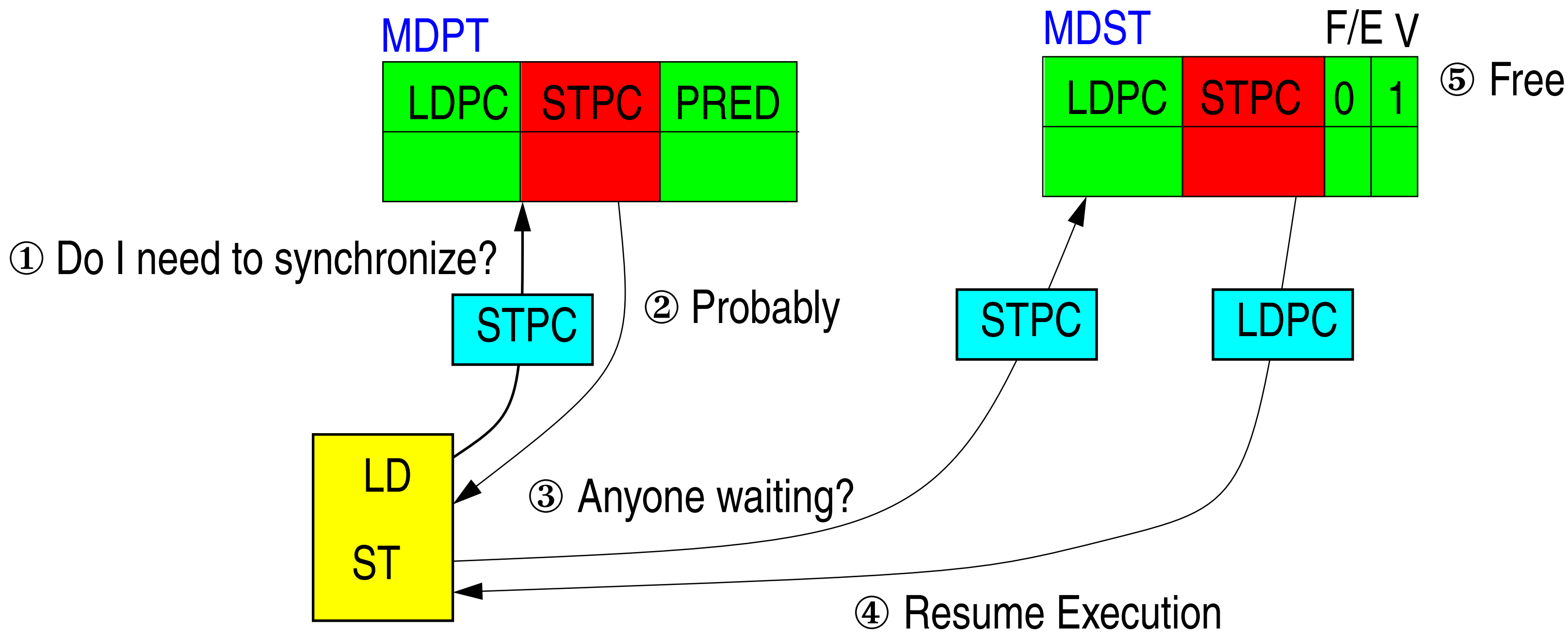
- Provide a small pool of full/empty bits
- Use *(LD PC, ST PC)* to associate entries w/ dependences

Memory Dependence Synchronization Table

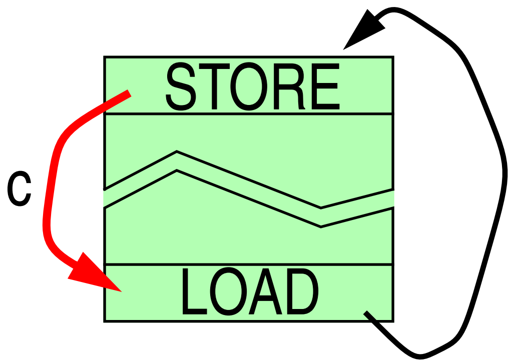


Synchronization - Load Resumes

Memory Dependence Synchronization Table



Multiple Instances of the Same Dependence



0	STORE
1	STORE
c	LOAD
c+1	LOAD

- Can't just use (Load PC, Store PC) for synchronization

In addition to (Load PC, Store PC) use:

- The data address accessed, OR
- Attempt to determine the dependence distance

  Analogous to simple static linear recurrence analysis

  - In Multiscalar we use the distance in stages
  - In a superscalar we may count the # of stores

Dependence Speculation/Synchronization

- Other alternatives exist for both prediction and synchronization.
- Simplifications may be possible.

For example:

- Use PC to identify only loads
- Use the data address to indirectly identify the stores and to synchronize

Roadmap

- ~~Overview~~
- ~~The Problem and Our Solution~~
- **Evaluation** (7 slides)
    - Comparison of speculation policies
    - Accuracy of prediction
    - Reduction in Mis-speculation rate
    - Speedup
- Other uses - Ongoing work

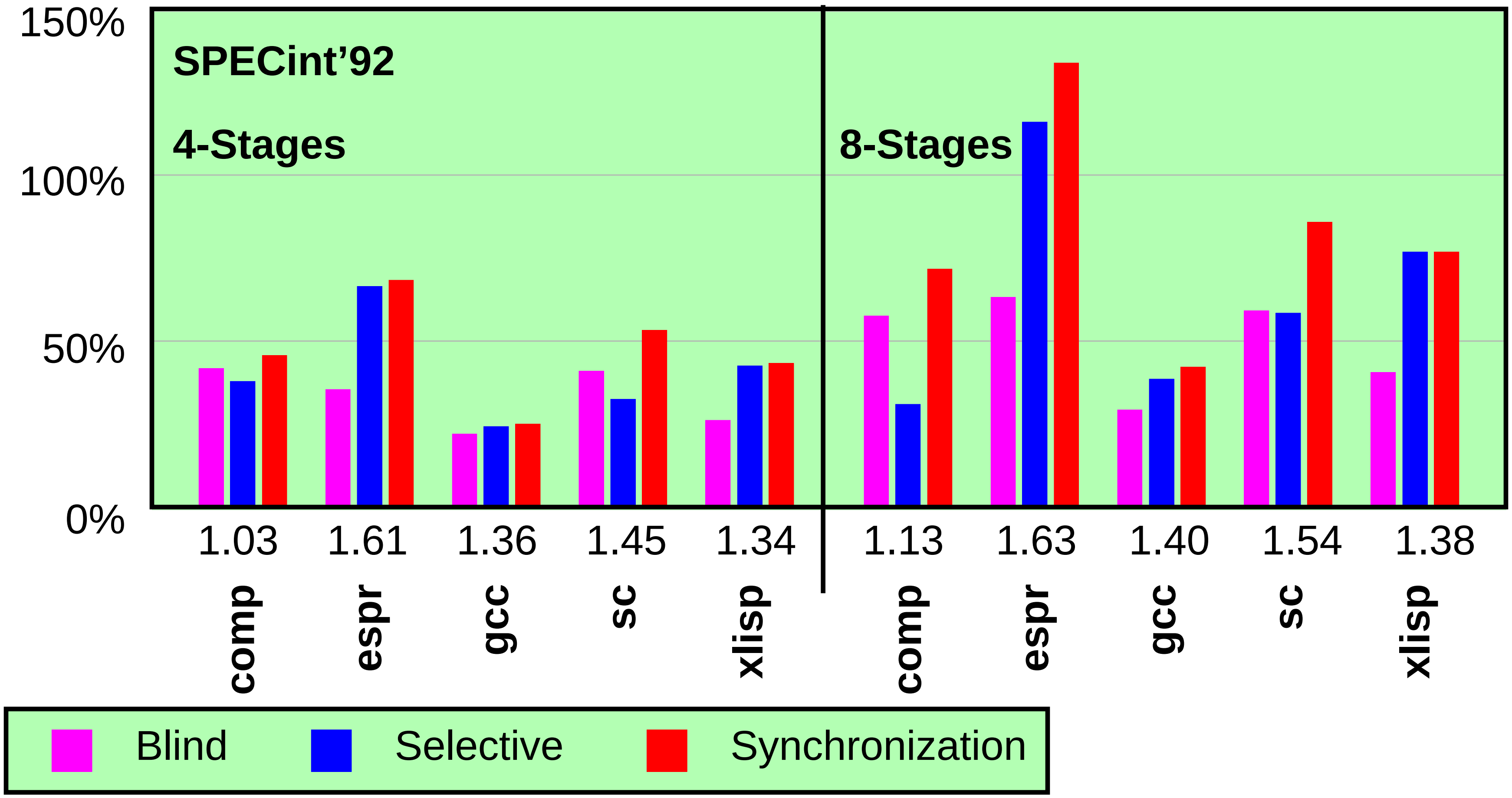
Evaluation - Methodology

- Use the Multiscalar architecture
  - is able to establish wide windows
  - uses memory data speculation aggressively
  - penalty of mis-speculation is significant
- Parameters
  - 8 stages, 2-way OoO
  - 32k 2-way Icache, 128k direct mapped data cache
  - 512 entry ARB
  - Non-blocking loads/stores
- Benchmarks
  - SPEC'95 INT and FP (compiled w/ gcc 2.7.2)
  - train/test inputs up to 2 Billion instructions
  - SPEC'92 for some experiments
- Instruction driven timing simulation

Evaluation - Parameters

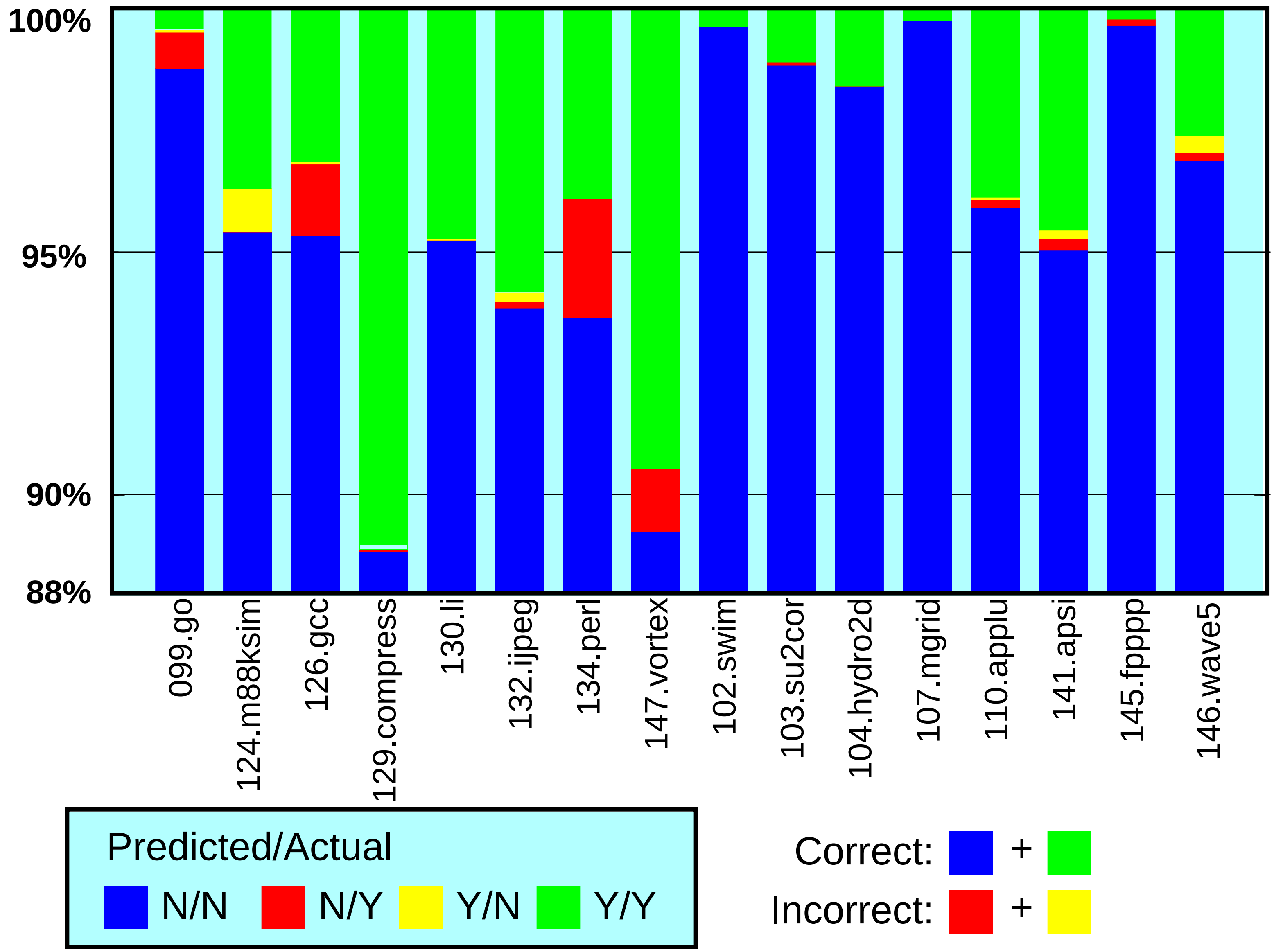
- Centralized scheme
- 64 entry combined MDPT/MDST
- Fully associative
- Multiple dependences: wait for all
- Multiple instances: use dependence distance
- Predictor:
    - 3-bit counter based (threshold of 3)
    - Also maintains minimal control path information:
    - Records the PC of the task that issued the store

Comparison of Speculation Policies

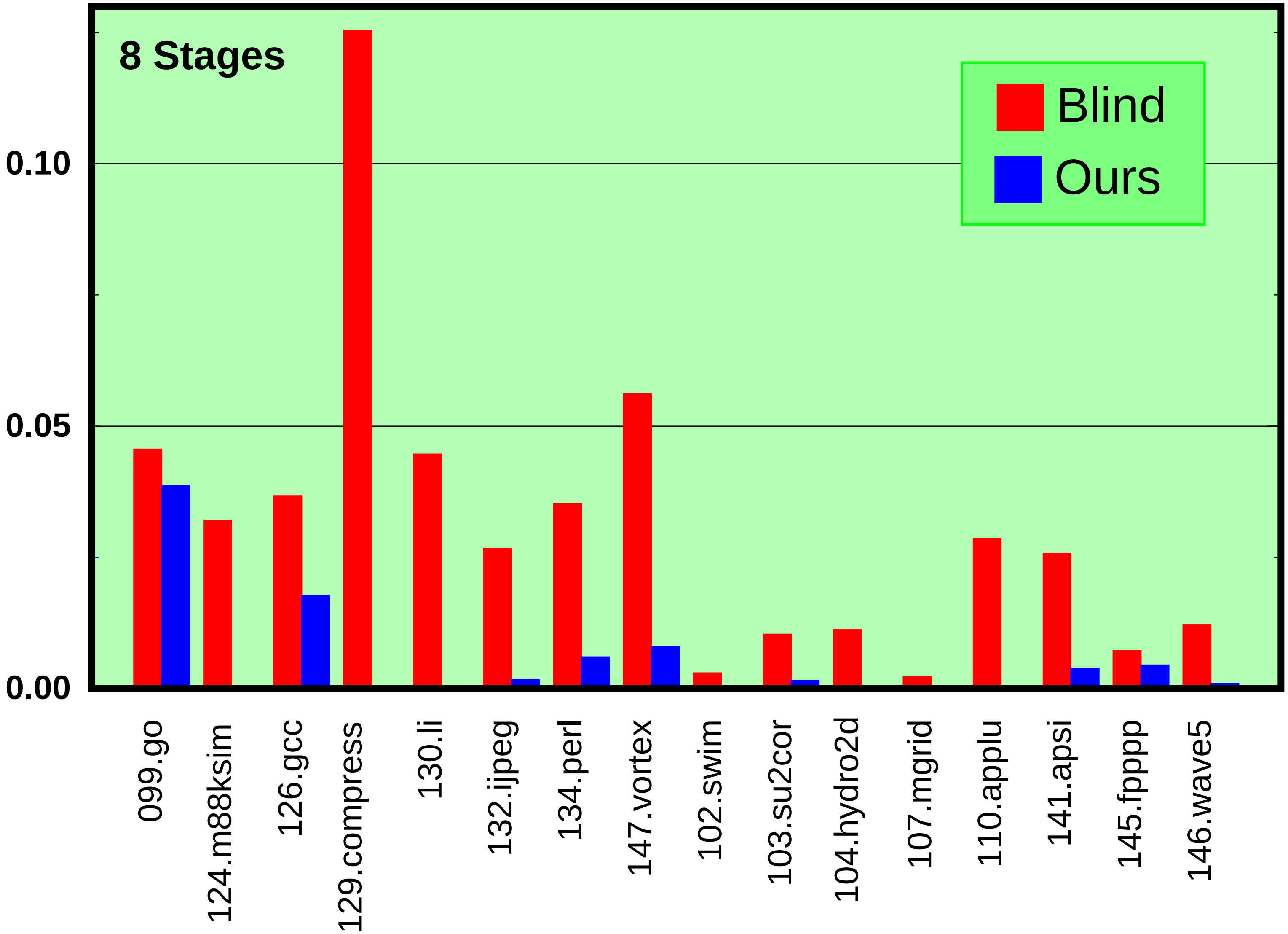


- Speedups are relative to no speculation (IPC along X axis)
- Perfect dependence prediction is used

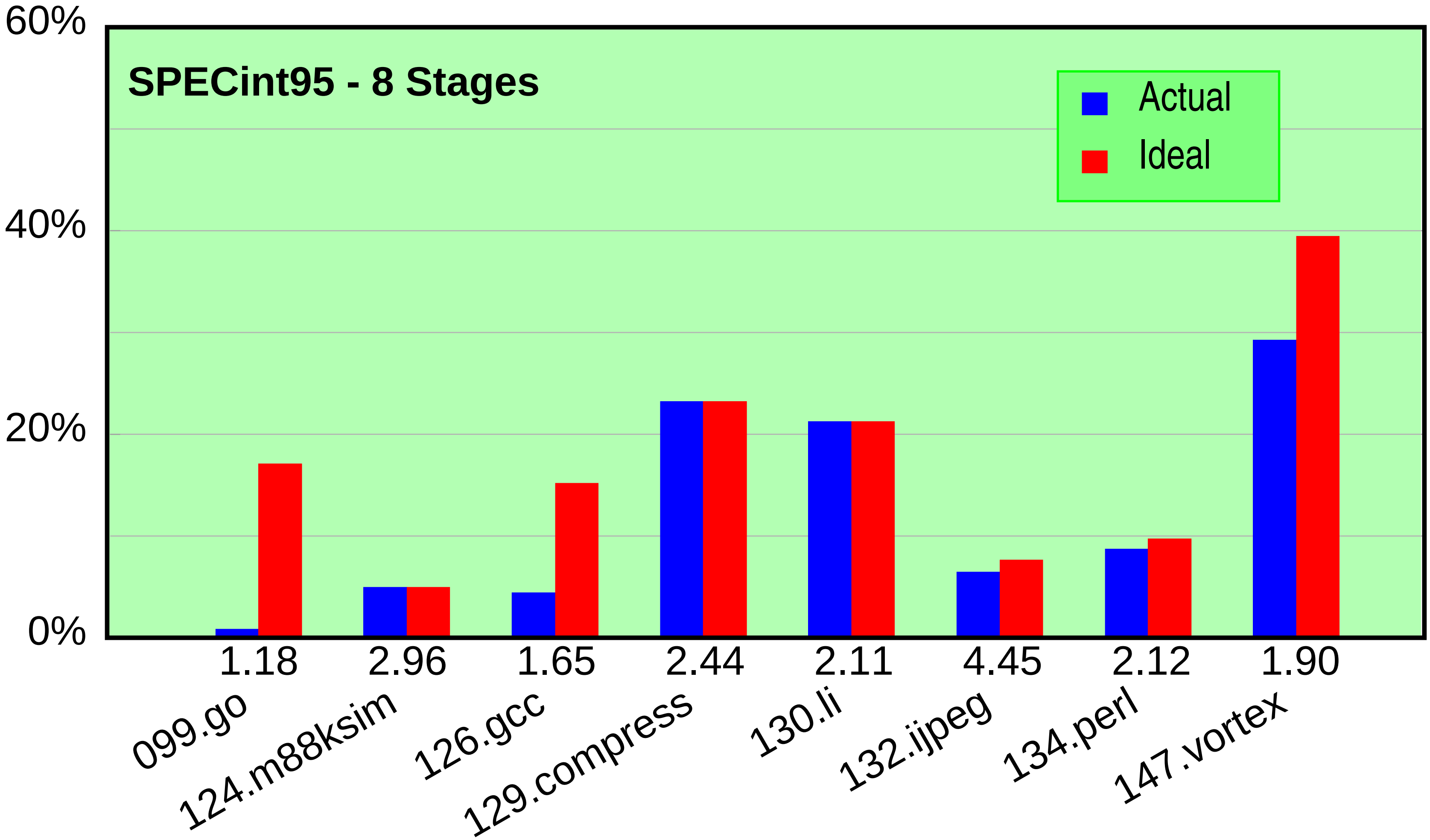
Dependence Prediction Accuracy



Mis-speculation Rates

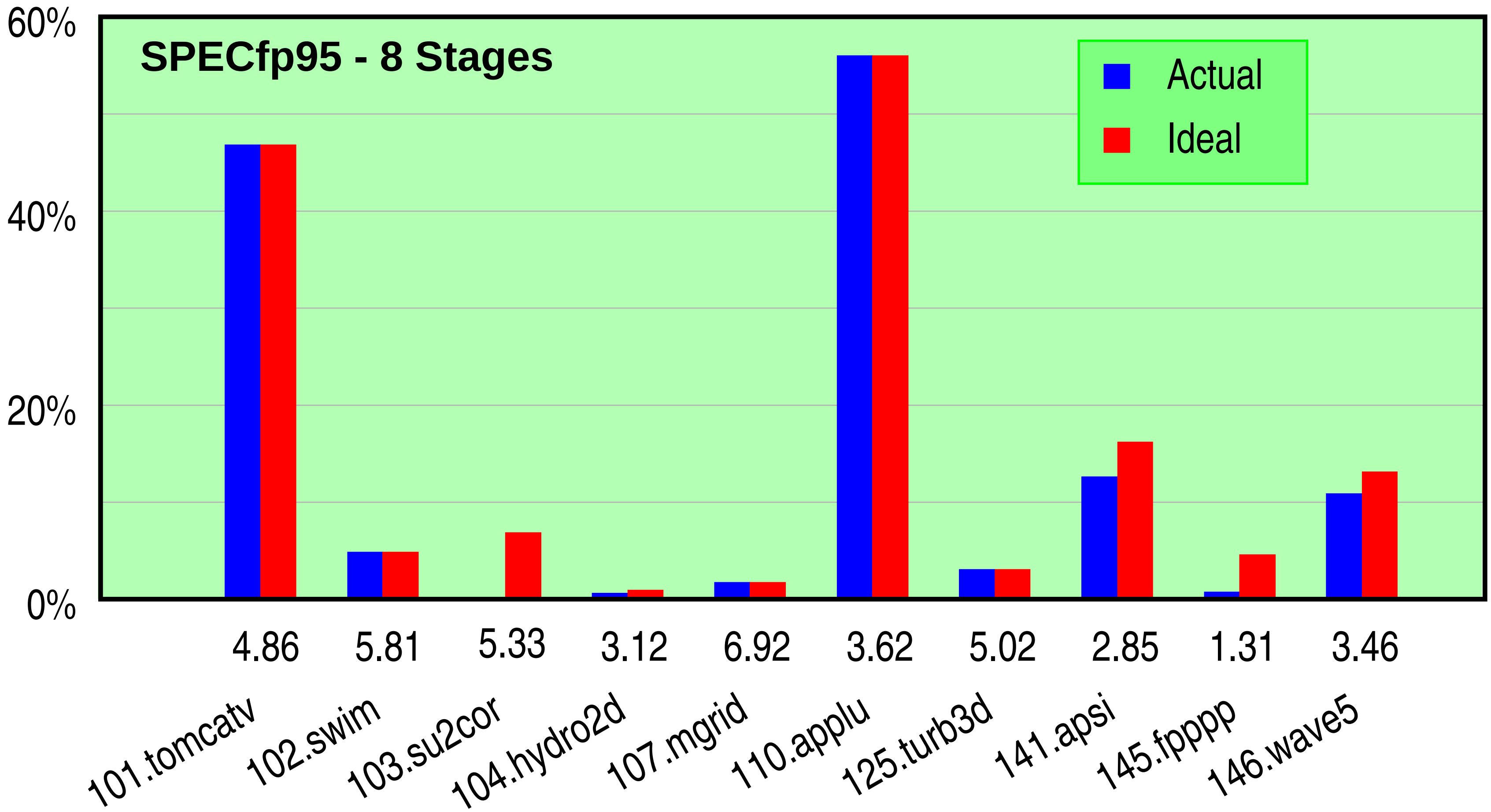


Speedup - SPECint95



- Speedups are relative to blind speculation
- IPC w/ our mechanism

Speedup - SPECfp95



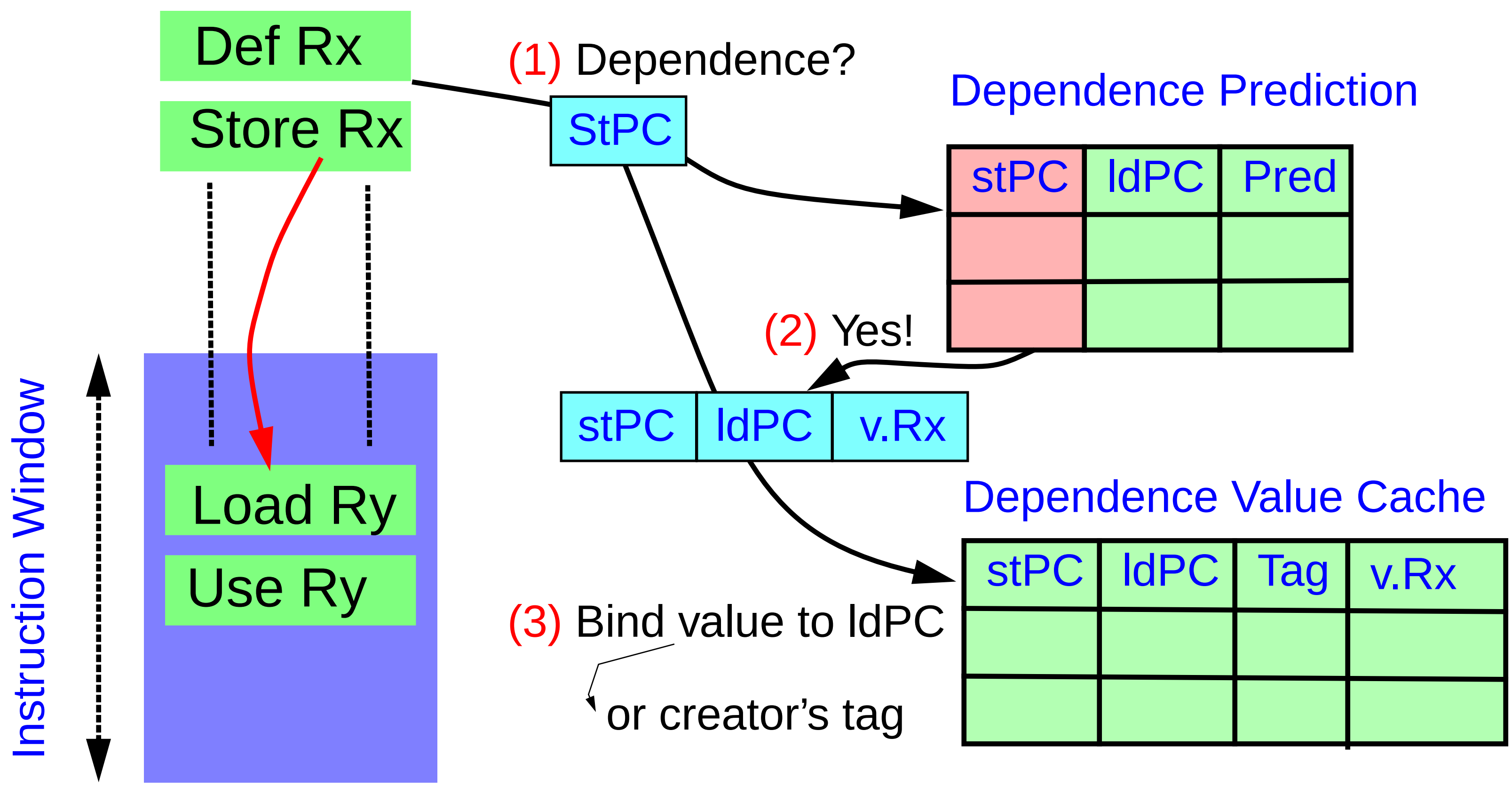
- Speedups are relative to blind speculation
- IPC w/ our mechanism

Roadmap

- ~~Overview~~
- ~~The Problem and Our Solution~~
- ~~Evaluation~~
- **Other uses - Ongoing work**
    - Speeding up the communication of data values
    - Other

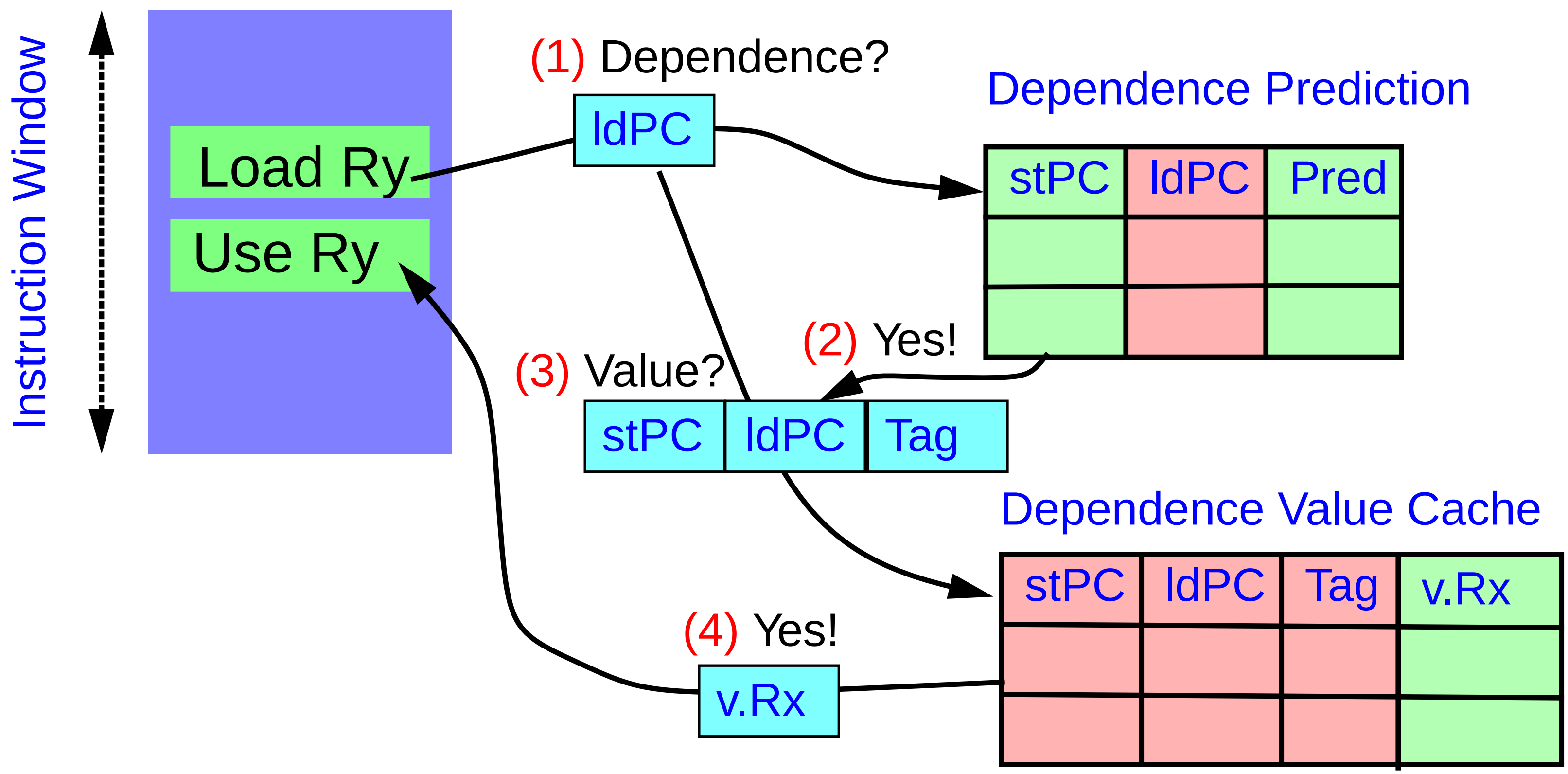
Other Uses of Dependence Prediction

- Speedup the communication of memory data values
  - (1). When store is decoded, predict the dependent load
  - (2). Associate the value or creator with the PC of the load



Other Uses of Dependence Speculation

- When load is decoded, use its PC to determine the value or creator
- Dependent Instructions may use the speculative value
- Load is also executed, to verify the value speculation



Ongoing Work

- Alternative prediction/synchronization methods
- Superscalar environment
- Distributed organization
- Integration with the memory disambiguation mechanism
- Sensitivity analysis
- Exposing to the compiler