# Streamlining Inter-Operation Memory Communication via Data Dependence Prediction

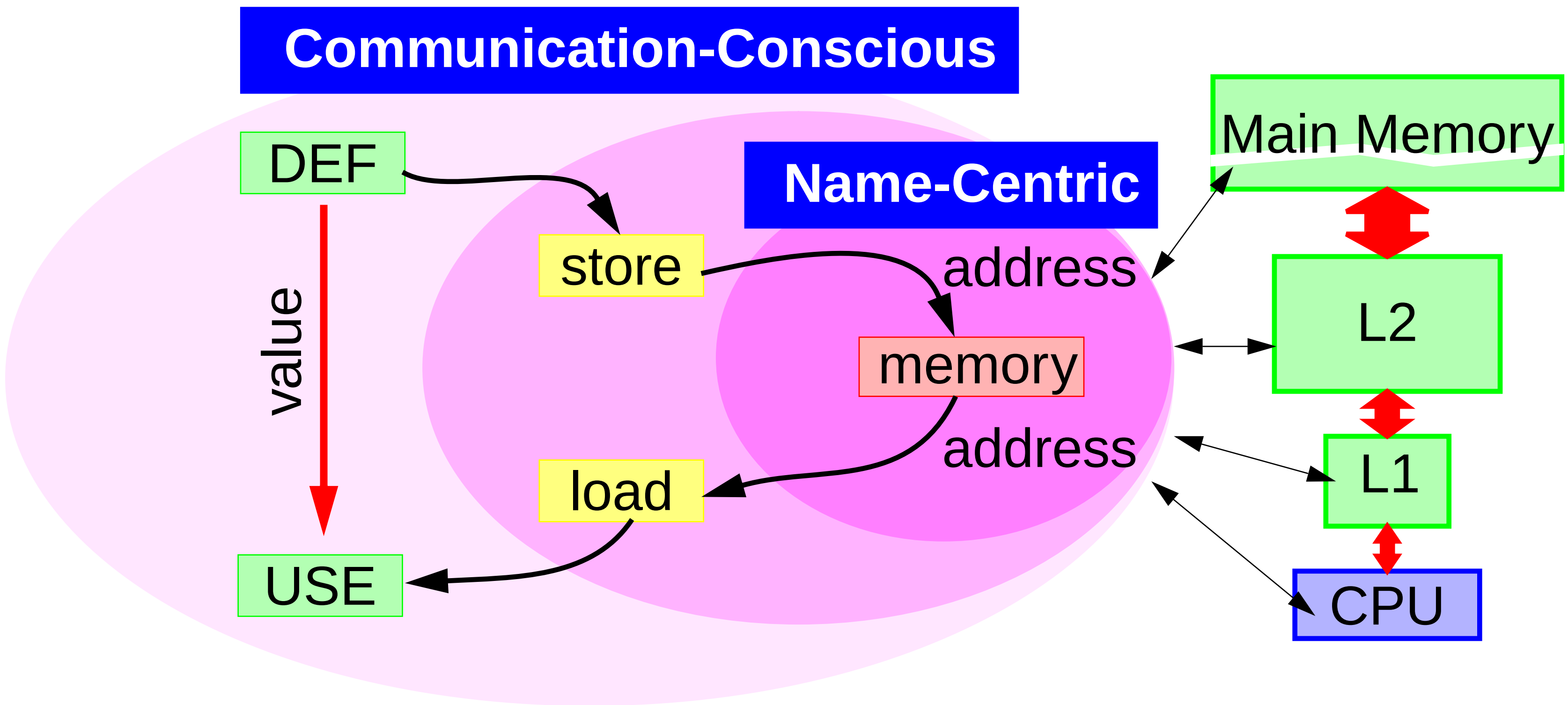
***Andreas Moshovos and Guri Sohi***

***{moshovos, sohi}@cs.wisc.edu***

*Computer Sciences Department*

*University of Wisconsin-Madison*

# Memory as a Communication Agent




**Communication-Conscious Approach**

***In Addition to*** *addresses*

**Expose the communication**

**Observe and exploit its behavior**

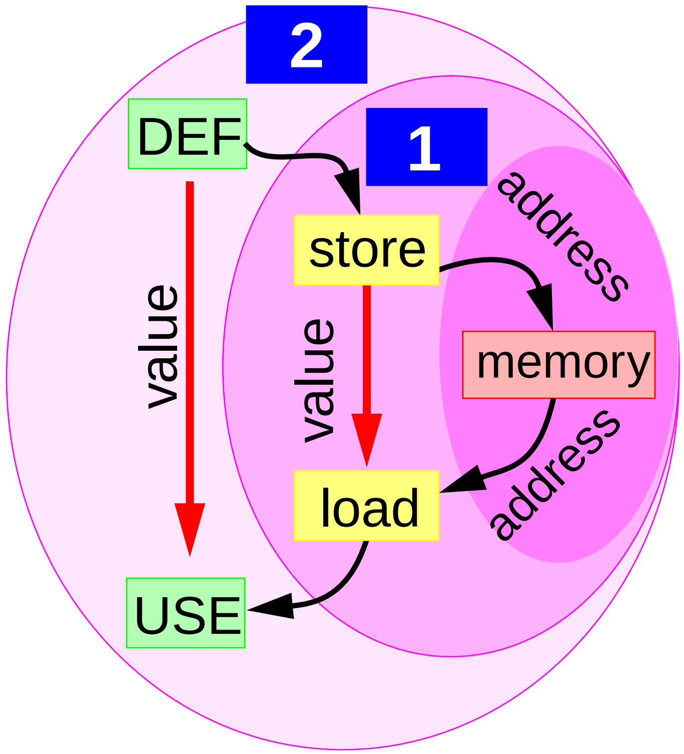
**This work:** How to use to: **1. ↓ Latency, 2. ↑ Bandwidth**

# Communication-Conscious Techniques

Communicating via addresses → inherent delay

**Observe:**

- Many loads get their value from a recent store
- These dependences are **predictable**




## 1. Speculative Memory Cloaking

- Prediction: link *load - store*
- pass value
- verify through memory

## 2. Speculative Memory Bypassing

- link ***DEF - USE***

**Communication Latency is Reduced**

# Communication-Conscious Techniques

DCache ports are becoming expensive

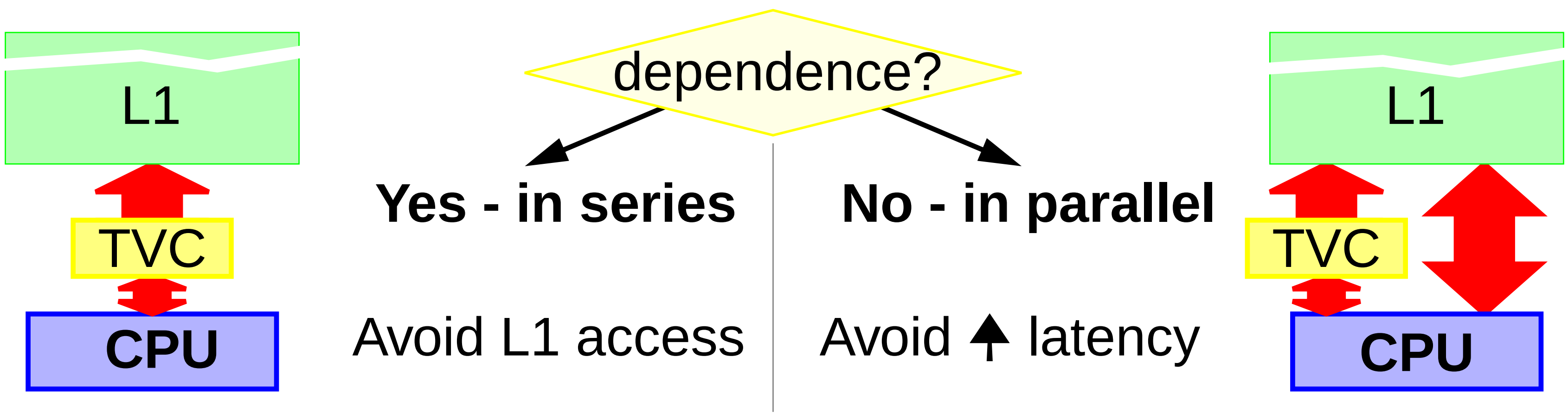
**Observe:**
- **A.** Recent stores feed many loads
- **B.** Many recent stores are killed

**+** Small cache can service these

**-** Latency for other loads will increase

**C.** A & B / **Dependence Status** is **predictable**

## 3. Transient Value Cache



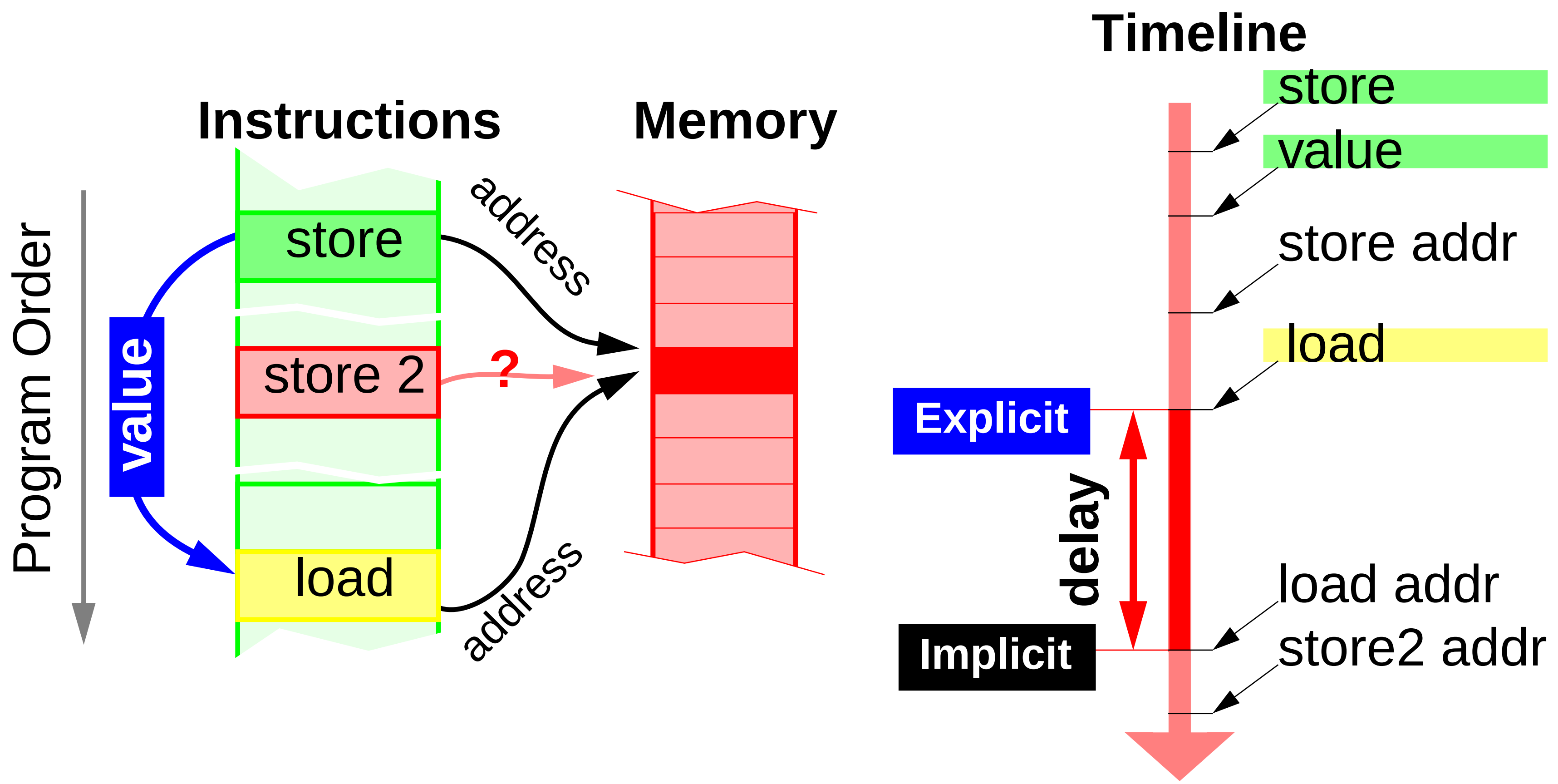

**L1 DCache Bandwidth/Port Requirements are Reduced**

# Roadmap

- ~~Introduction~~
- **Traditional Memory Communication Specification- Limitations**
- **Speculative Memory Cloaking**
- **Speculative Memory Bypassing**
- **Transient Value Cache**
- Evaluation
- Summary

# Memory Communication Specification - Limitations

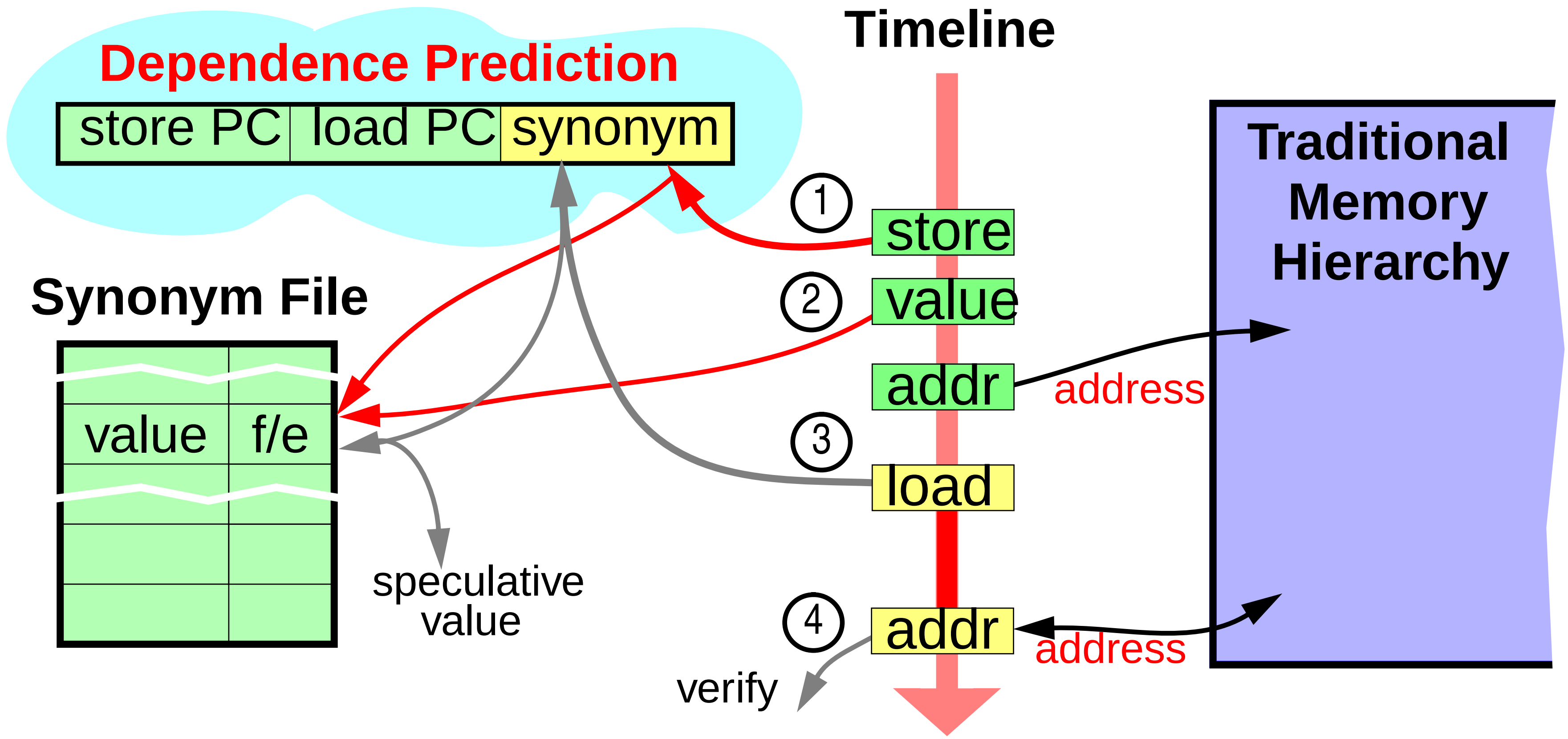
| **Implicit** | **Explicit** |
|---|---|
| **Delays:** 1. Calculate address<br>2. Establish Dependence | Store - Load: Direct Link<br>**No Delays** |



# Speculative Memory Cloaking

**Dynamically & Transparently convert implicit into explicit**

- **Dependence prediction** ➔ direct load-store link: **synonym**
- Speculative and has to be **eventually** verified



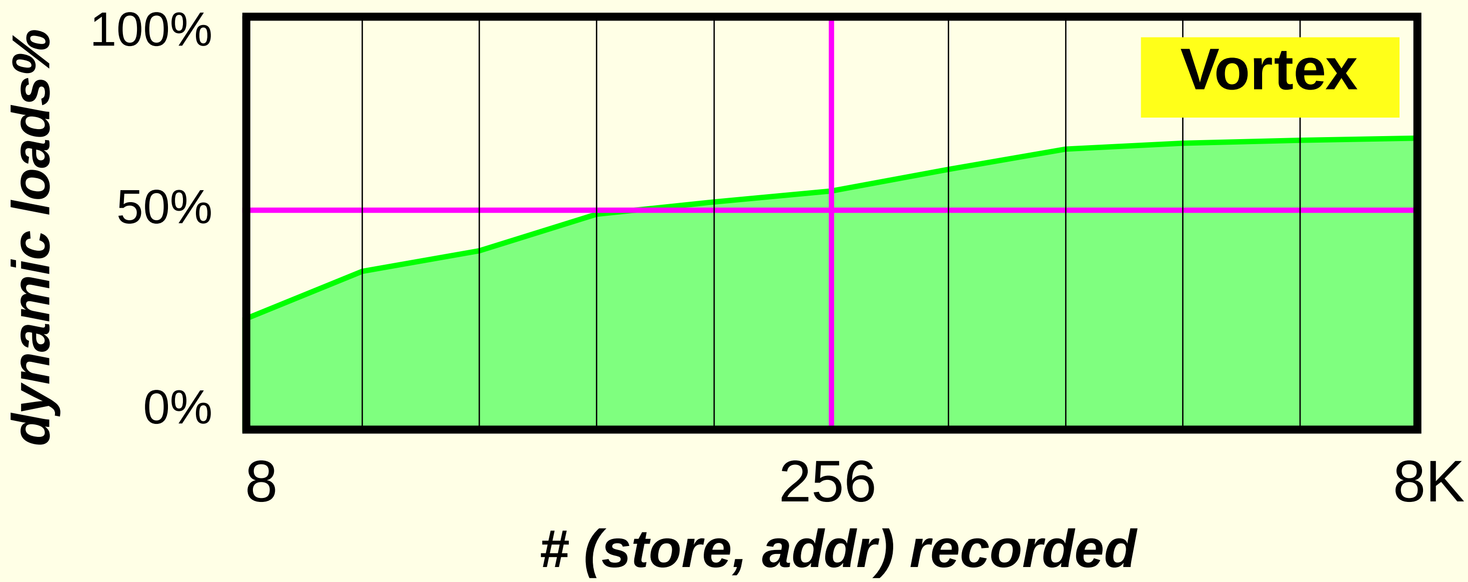
# Predicting True Dependences

## 1. Detect Dependences / Build History

**Record:** (*store PC*, *address*)

**Loads:** (*load PC*, *address*)

→ (*store PC, load PC*)




- not in critical path
- #stores << #instructions

**Recording last 256 stores captures ~50% of all load deps.**

# Cloaking - Issues & Implementation

***See paper for:***

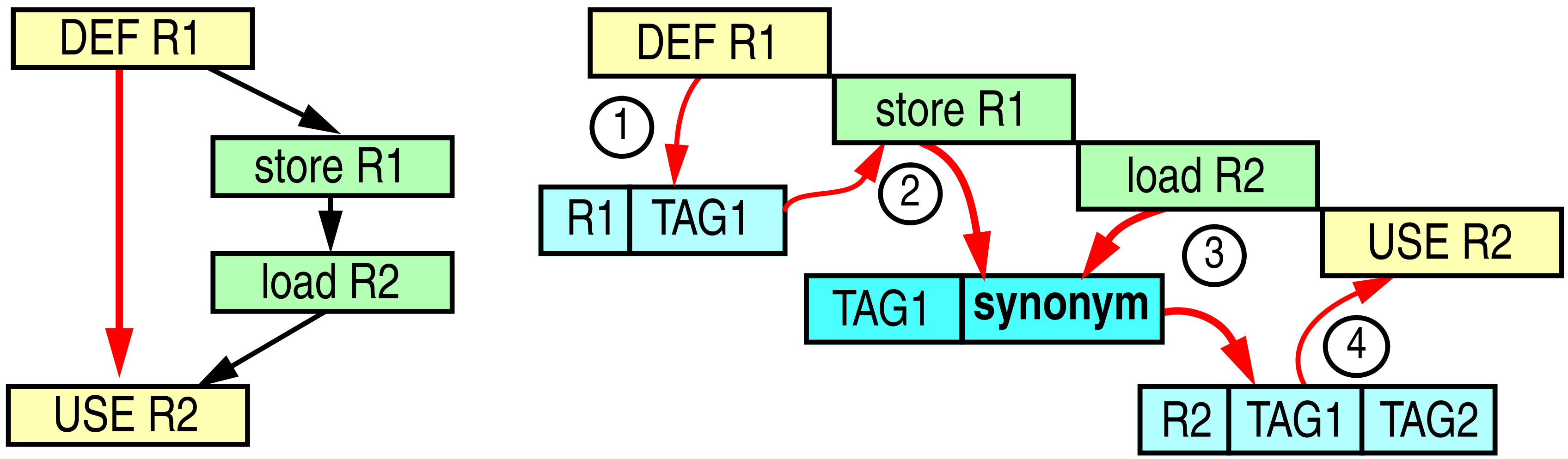
- **Predicting Dependences / Synonym Generation**
    - 1-to-1 and n-to-m dependences
    - Dependences w/ distance > 1
    - Multiple synonym instances
- **Data Types / Sign-extension**
- **Sample Implementation**

    Simple, table-like structures:

    - Detection
    - Prediction
    - Synonym File

# Speculative Memory Bypassing

**Observe:** Store and Load are used to just pass values




- Extents over multiple store-load dependences
- DEF and USE **must co-exist** in the instruction window

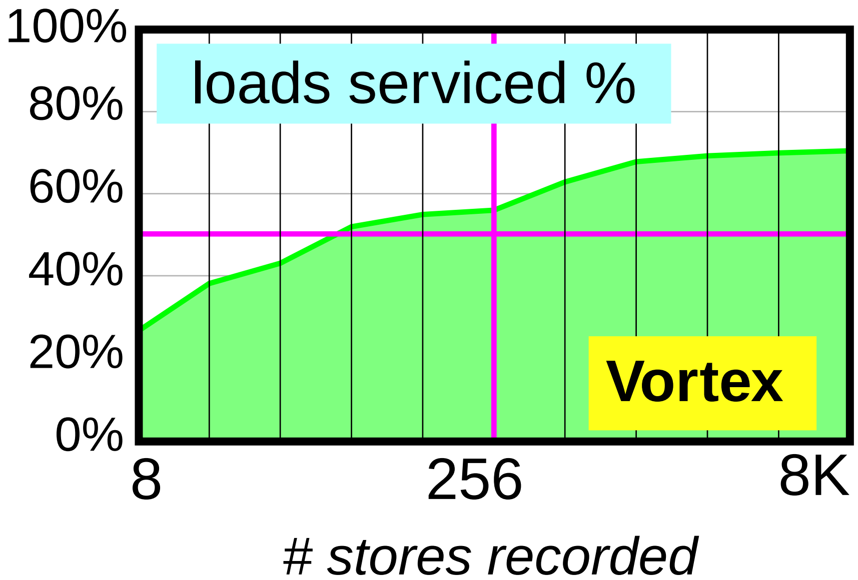
**Takes load-store off the communication path**

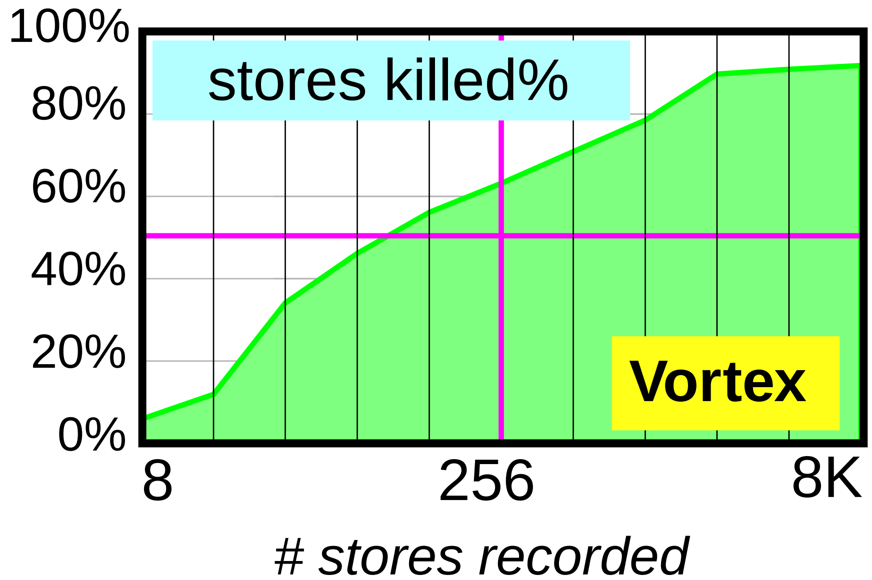
# Transient Value Cache

**Observations:**

Many loads get their value from a recent store

Many stored values are quickly killed




A 256-FA word cache can service > 50% of loads, 60% of stores

- **+** **Hit:** No need to consume L1 ports
- **-** **Miss:** Latency increases

# Transient Value Cache

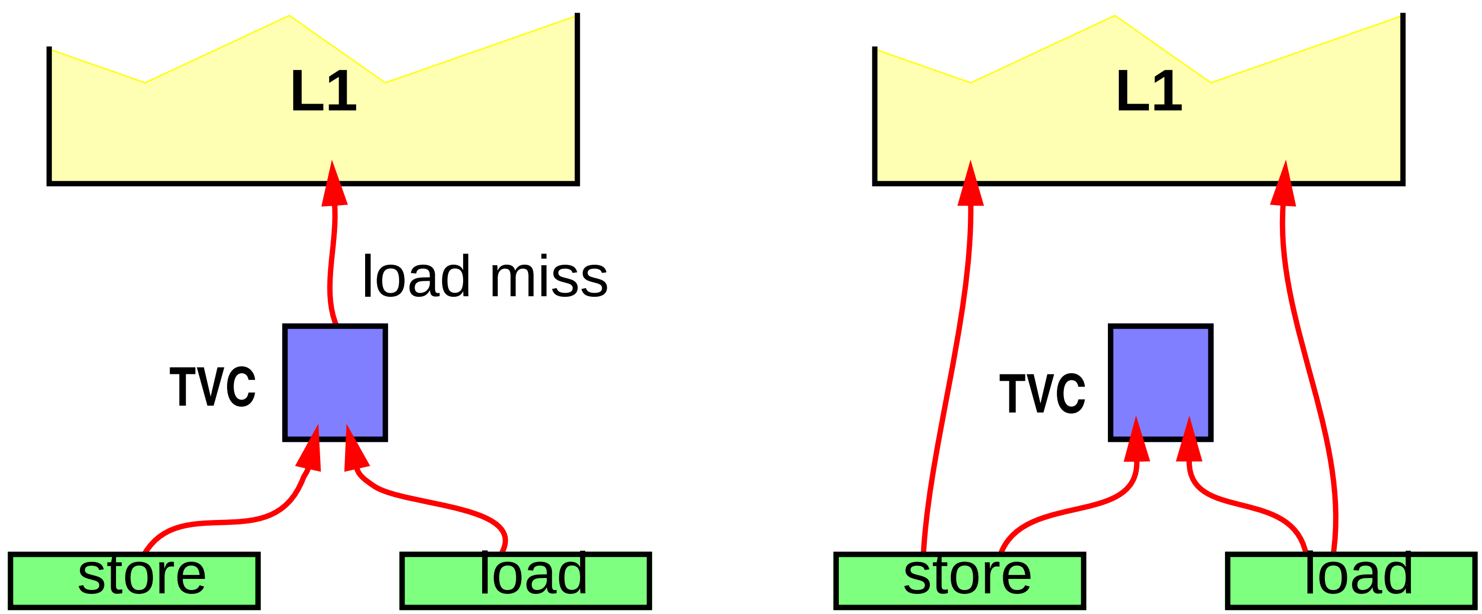
**The best of both worlds?**

**Data Dependence Status Prediction to steer**

**load:** reads a value from a recent store? *true dep.*

**store:** will be killed by a closeby store? *output dep.*

**Dependence: In Series** | **No Dependence: In Parallel**



# Evaluation

- **True Dependence Status prediction**

**Why:** • 1st step in cloaking & bypassing

• loads hidden by TVC

- **Cloaking Accuracy**

**Why:** • Loads % that benefit from cloaking

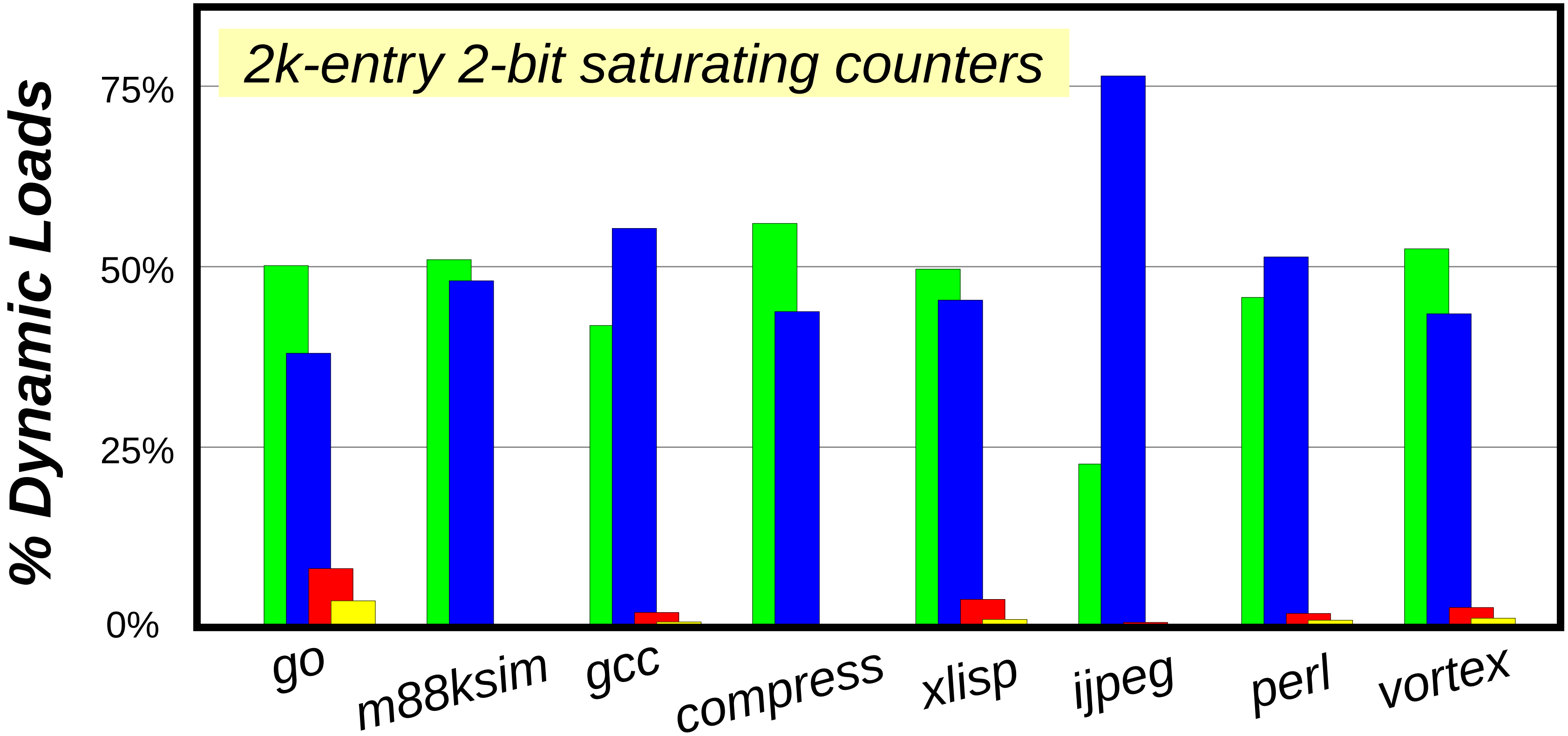
• Mis-speculated

- **TVC low bound on reduction in accesses**

***See Paper for:***

- **Cloaking:** Accuracy vs. Prediction table size, History kept
- **Output Dependence Status Prediction**
- **Impact on Performance**

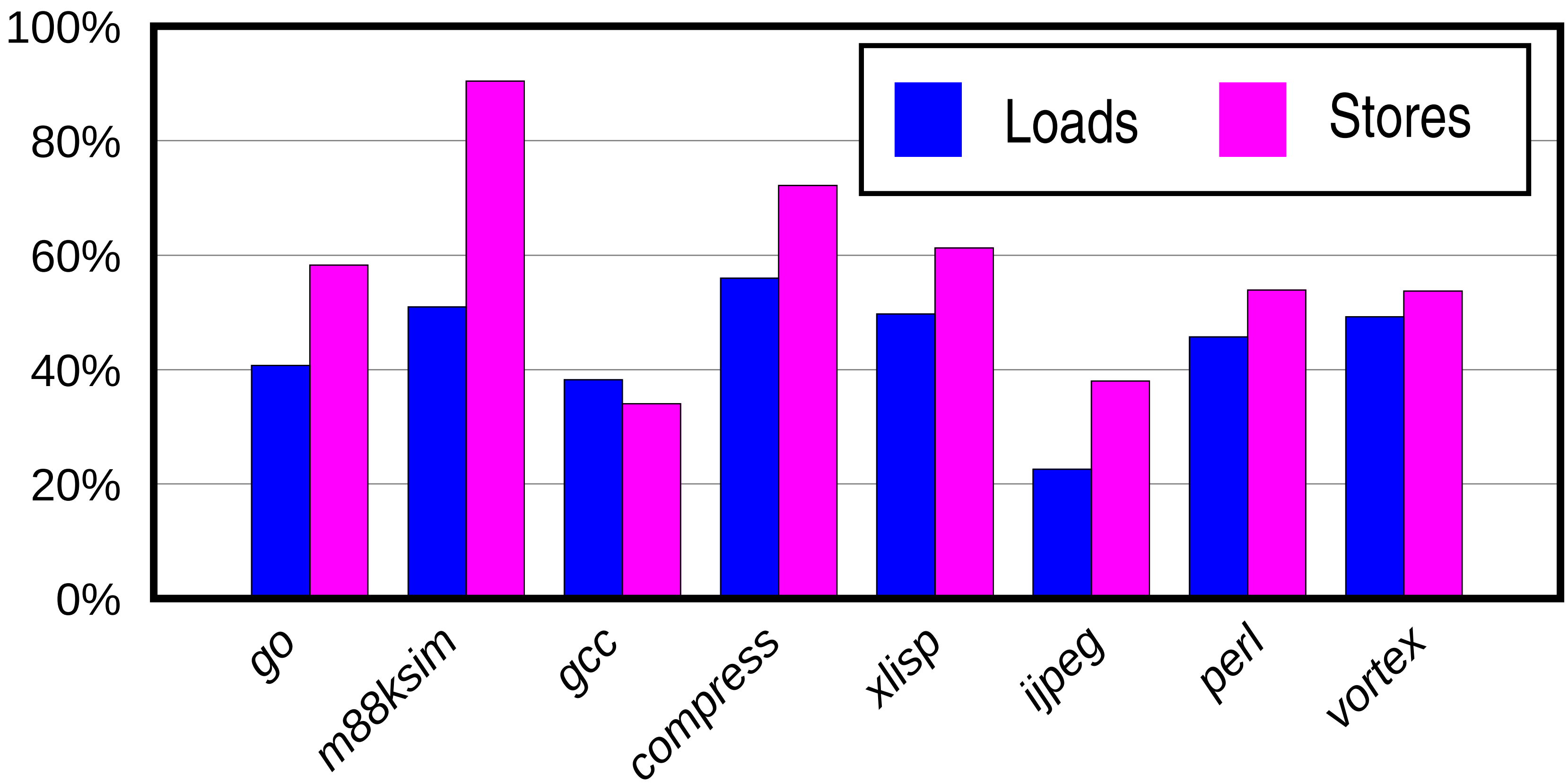
# True Dependence Status Prediction Accuracy



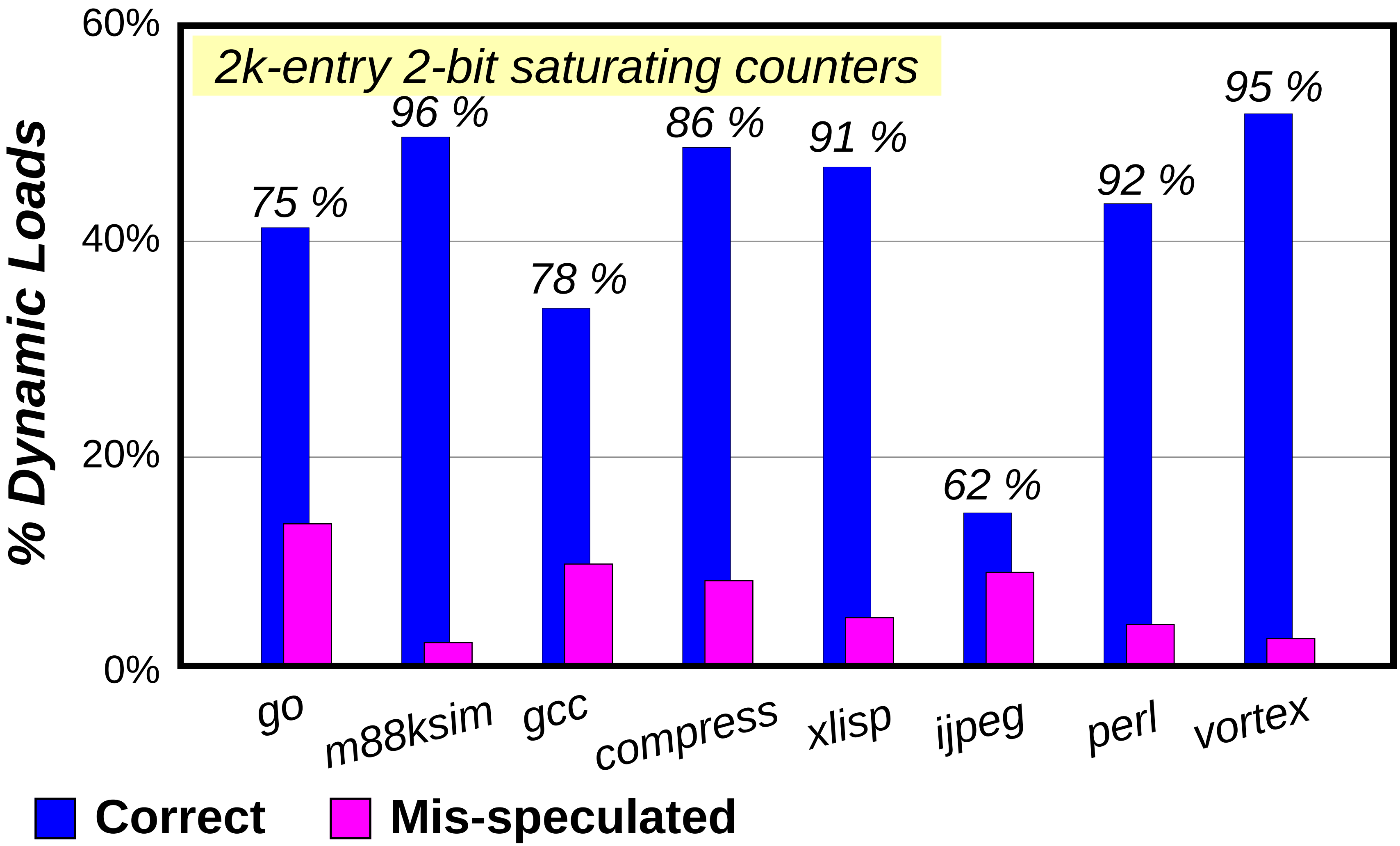

***Pred/Actual***

| ***Correct*** | | ***Wrong*** | |
|---|---|---|---|
| **Y/Y** | *can benefit* | **N/Y** | *missed opportunity* |
| **N/N** | *can't benefit* | **Y/N** | *harm* |

# TVC - Reduction in Accesses



# Cloaking - Dynamic Loads Serviced




**First-cut implementation**

# Summary

**1. Many loads get their value from a recent store**

**2. Many stored values are quickly killed**

**Predictable**

- **Speculative Memory Cloaking - *Latency***

  Convert **implicit** into **explicit**

  Pass values using just the PC

- **Speculative Memory Bypassing - *Latency***

  Take load-store off the communication path

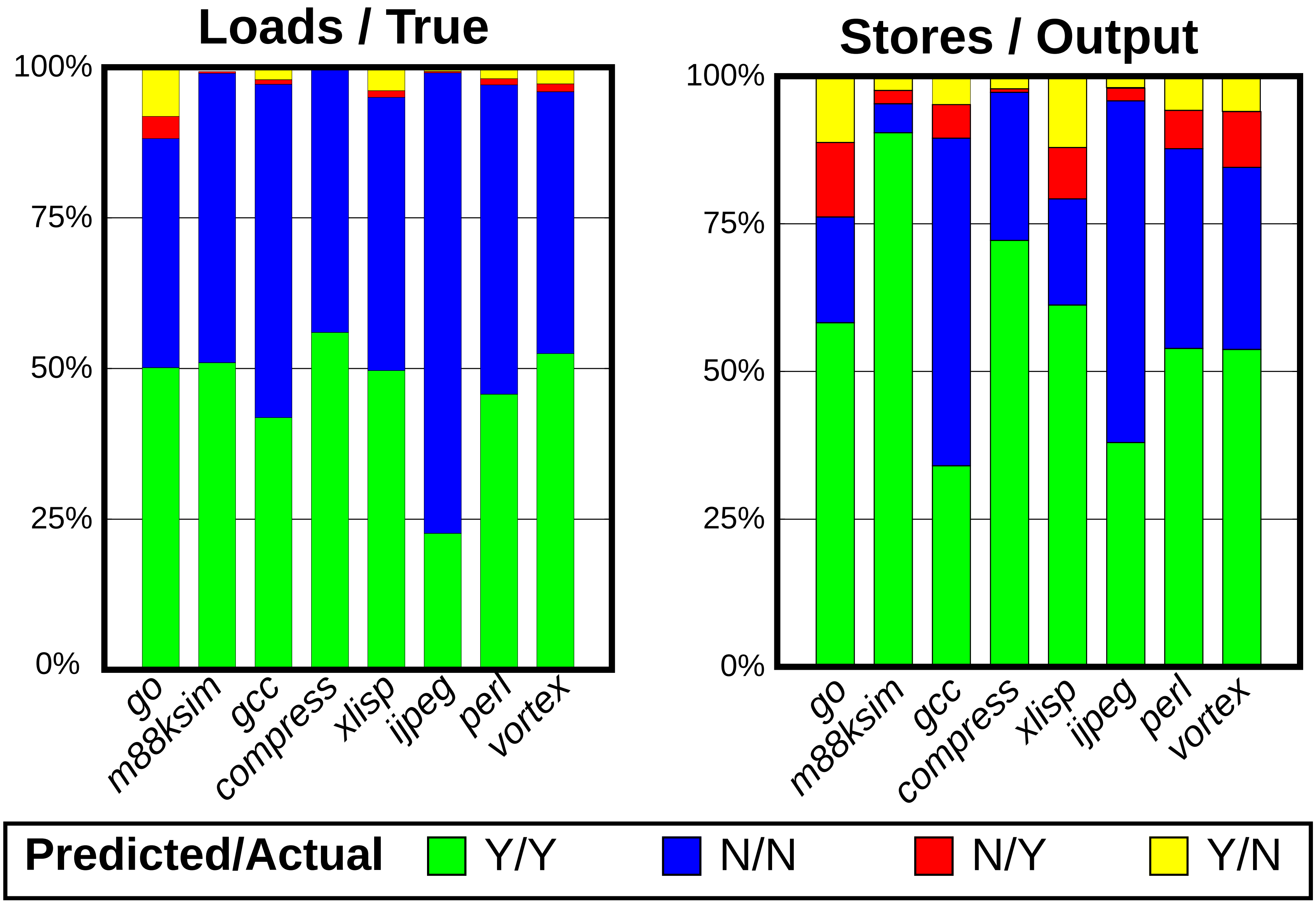
- **Transient Value Cache - *Bandwidth***

  Selective redirection of loads/stores

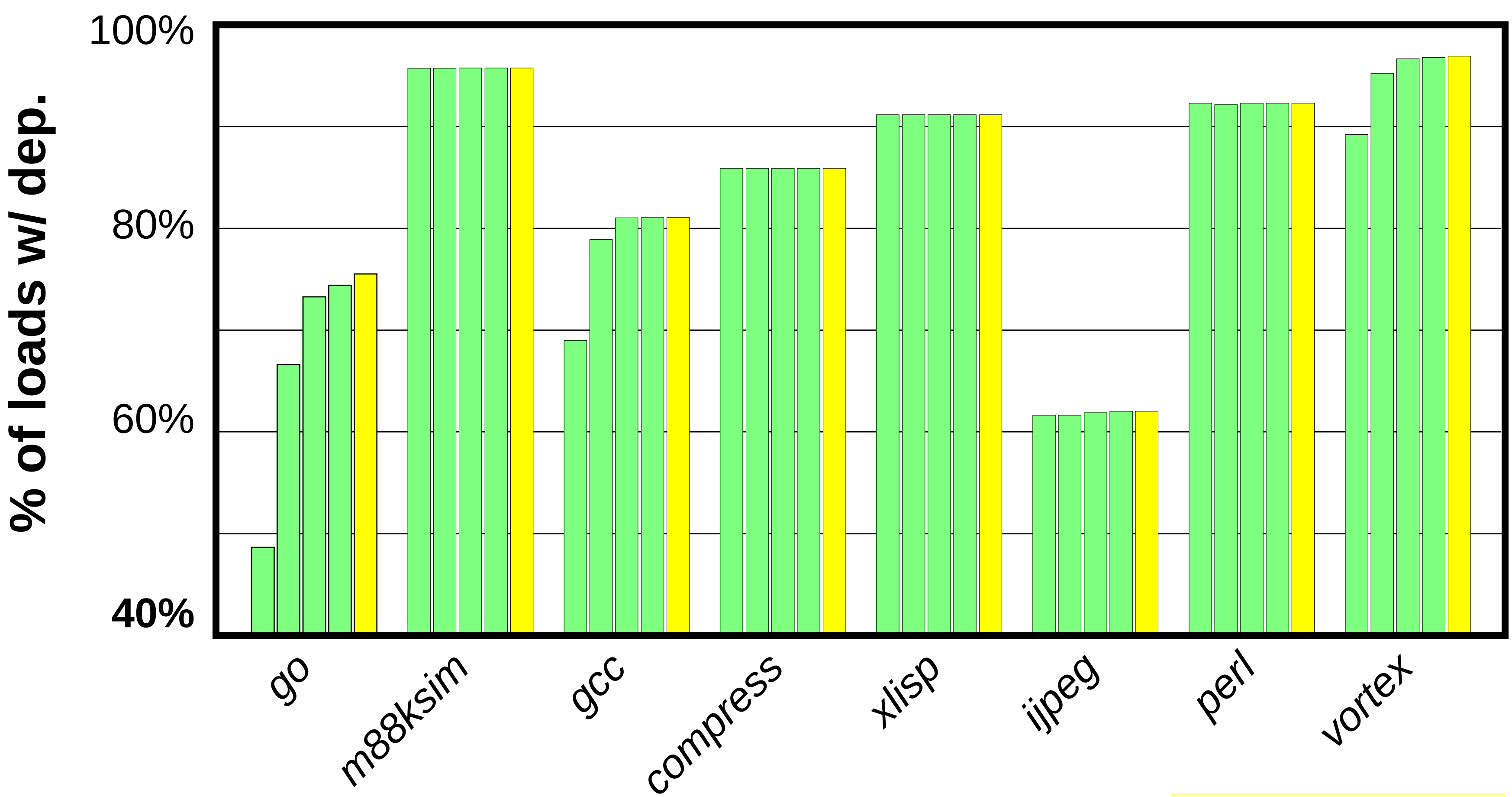
  Reduces bandwidth (port) requirements

Pssst... I'm hoping to graduate by Fall '98

# True/Output Dependence Status Prediction

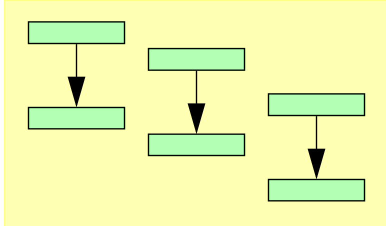


# Cloaking Accuracy

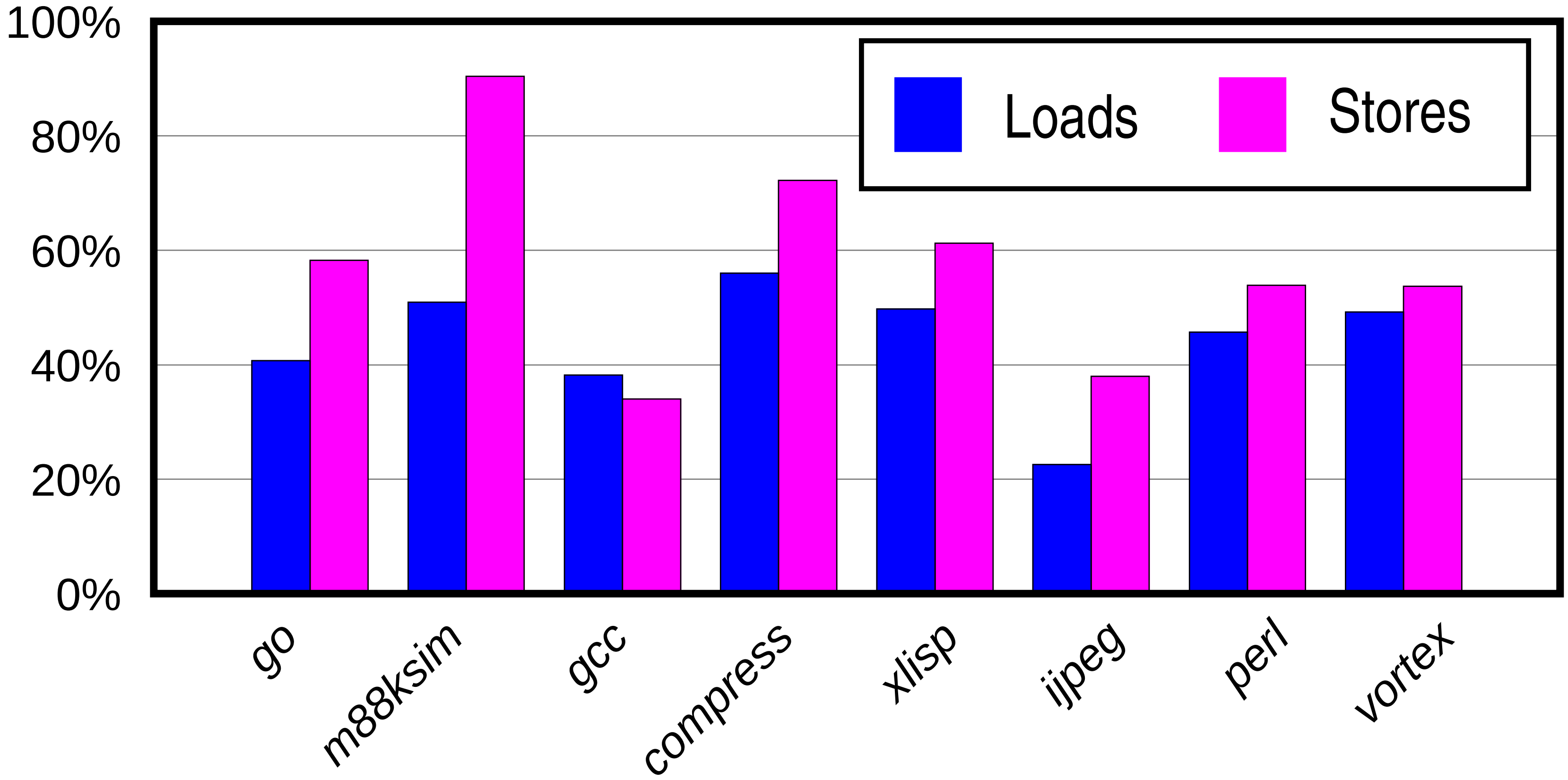



512, 1K, 2K, 4K

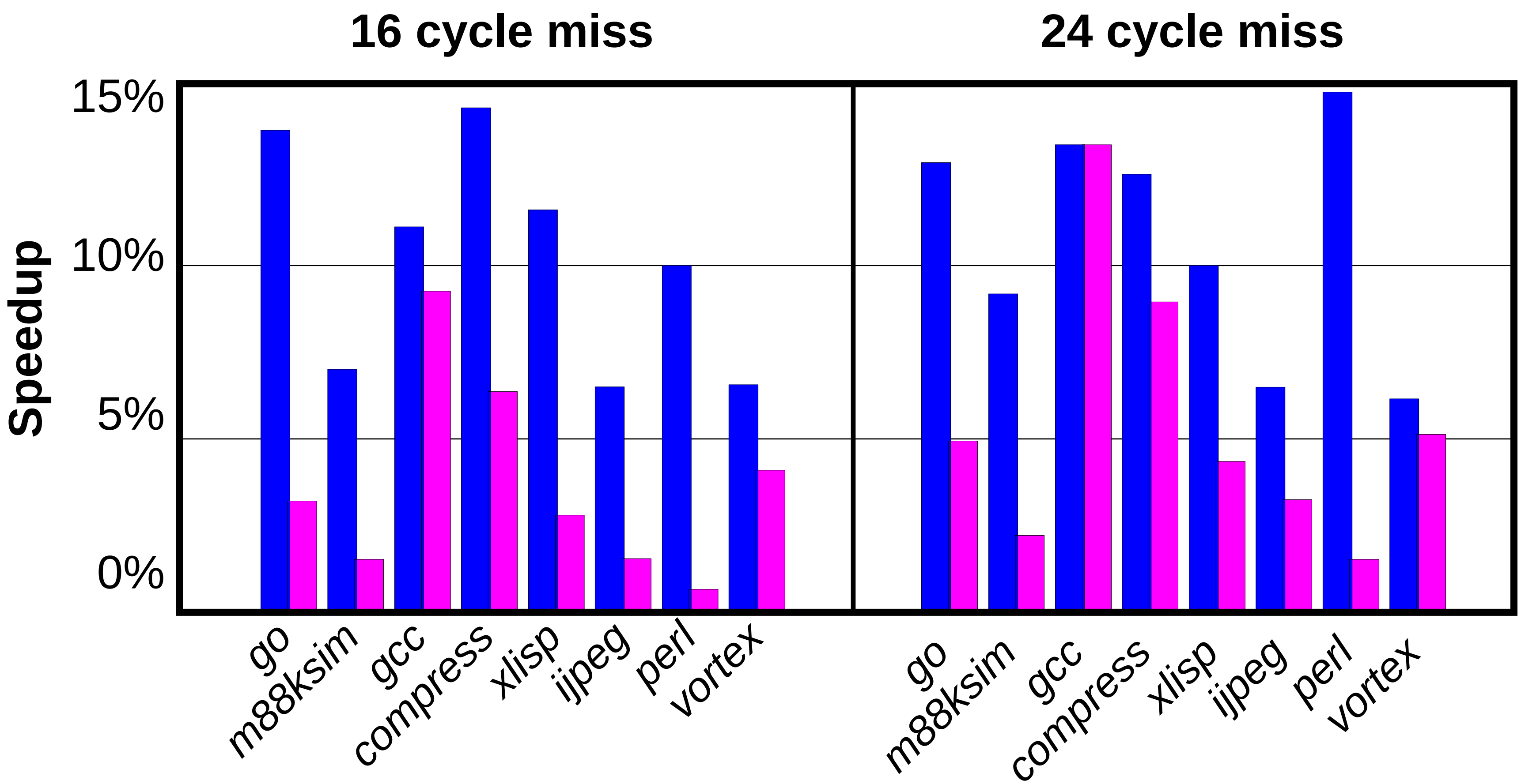
Infinite



# TVC - Reduction in Accesses




**Pessimistic model: last 256 stores - not last 256 addresses**


16 cycle miss
24 cycle miss
Speedup
15%
10%
5%
0%
go
m88ksim
gcc
compress
xlisp
ijpeg
perl
vortex
go
m88ksim
gcc
compress
xlisp
ijpeg
perl
vortex

# Why “Cloaking”?

**cloak** *n \’klok*

2 : *to alter so as to hide the character of*

3 : *something that conceals*

**“Speculative Memory Renaming”?**

- Already in use in the same context: ARB, LSQ (w/o Speculative)
- **Re-name:** *change the name*
  - *associate address with a new name*
  - *Legacy of “Register Renaming”:*
  - *can go from address to new name*

    *synonym and address are NEVER associated*

    *can’t determine synonym from address*

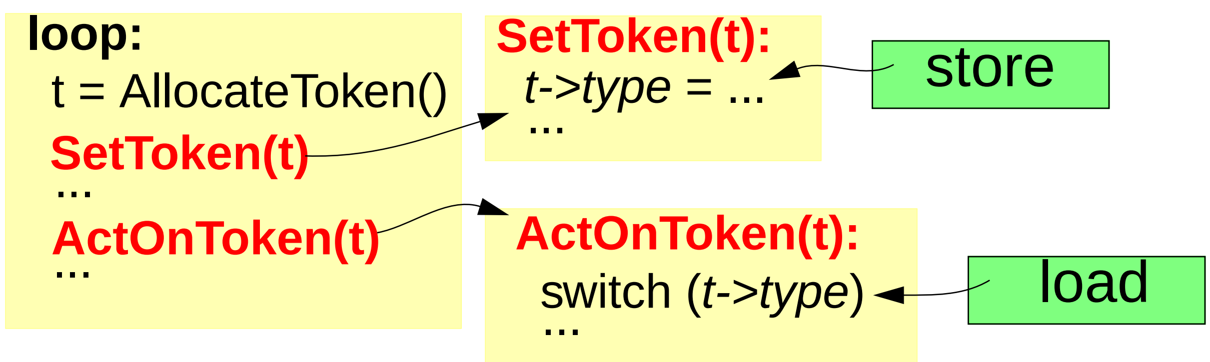
    *other accesses to the same address can’t locate synonym*

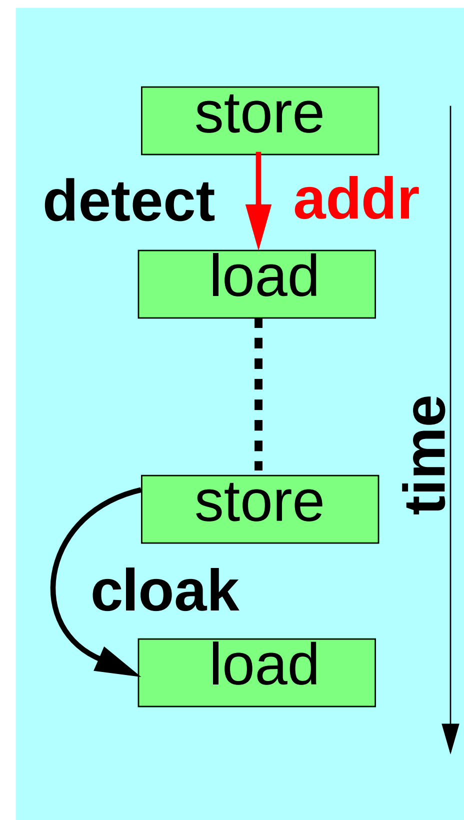
# An Implementation

Support Structures:

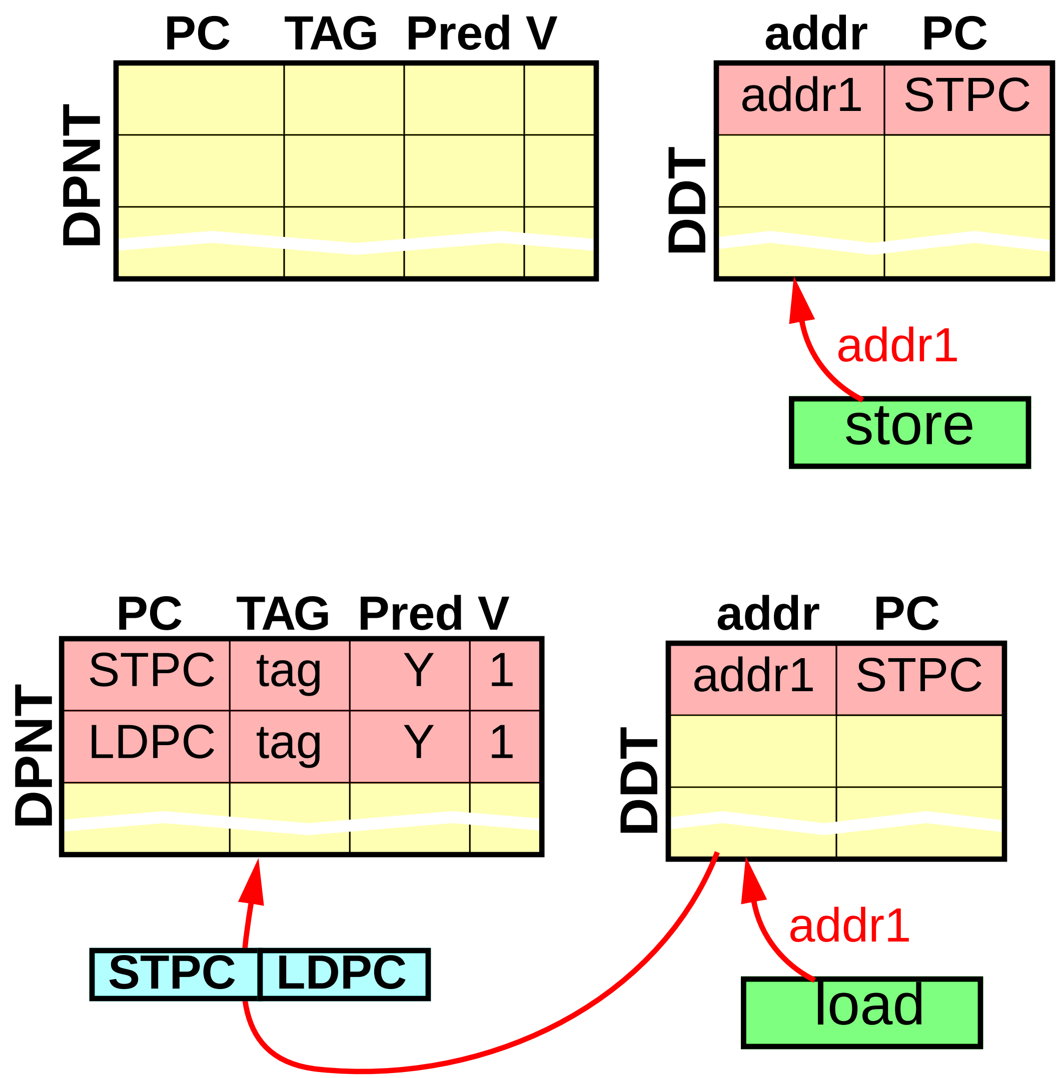
1. **Dependence Detection Table DDT**
2. **Dependence Prediction and Naming Table DPNT**
3. **Synonym File SF**

Example:

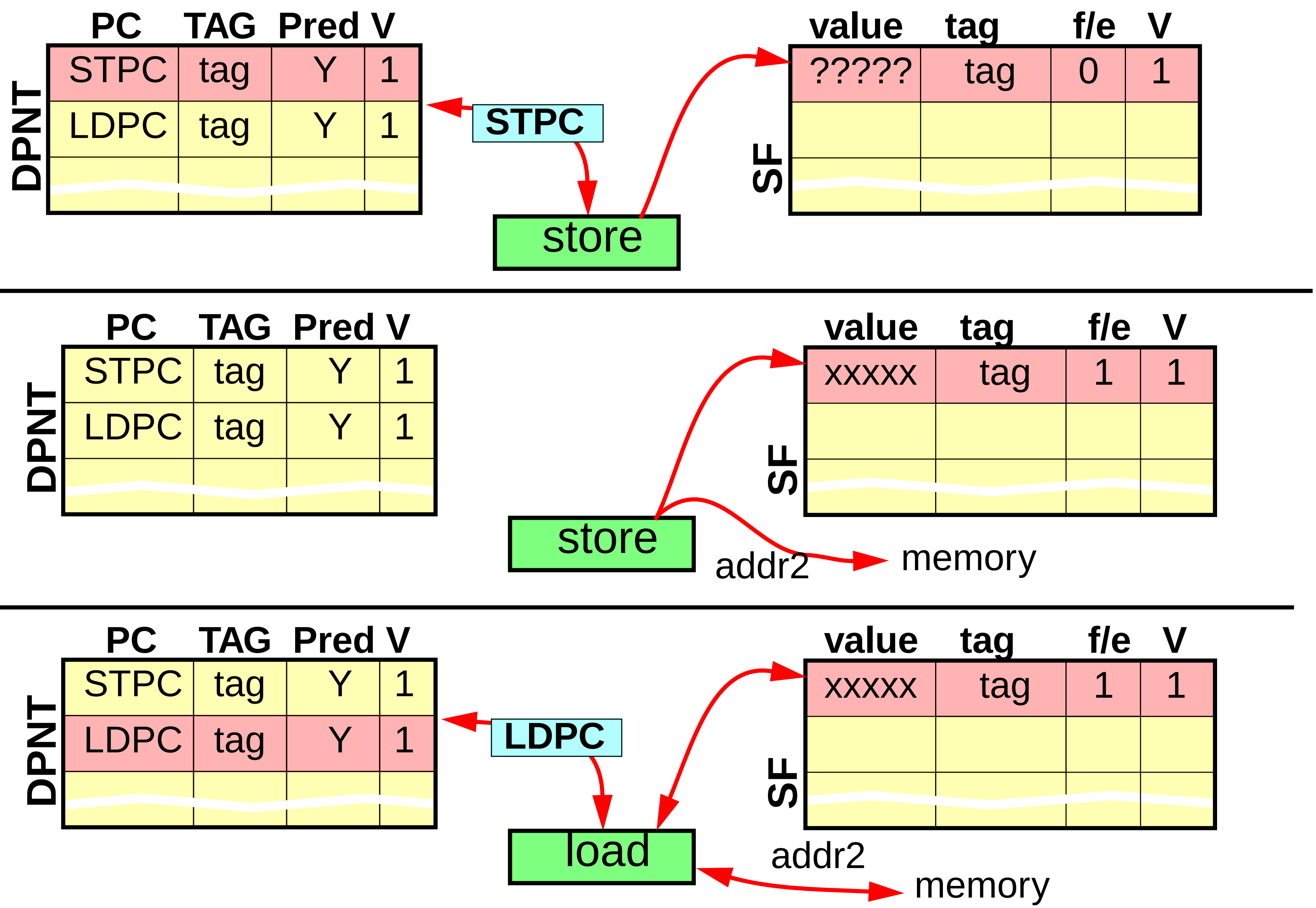


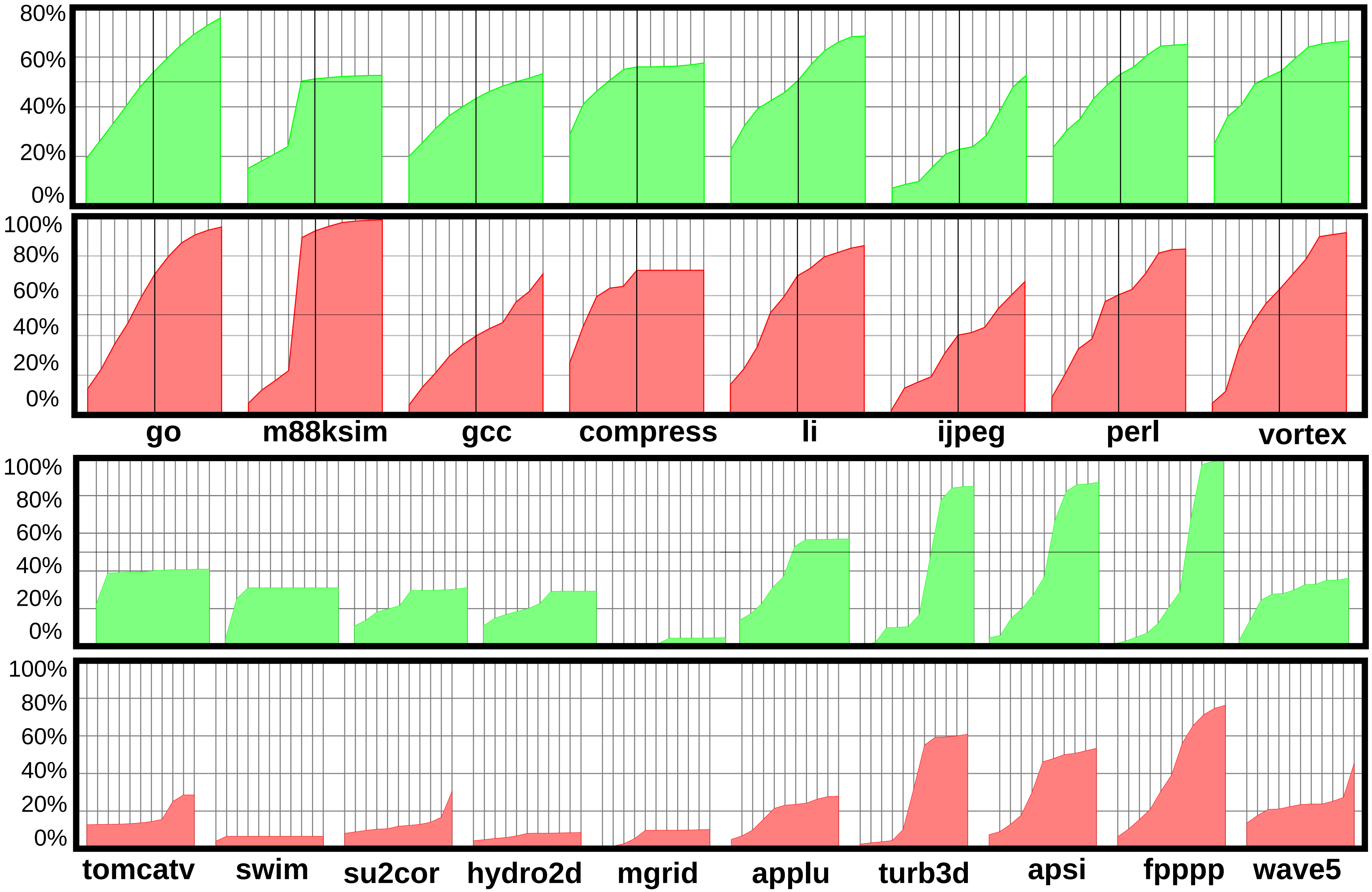



# An implementation - Example



# An implementation - Example

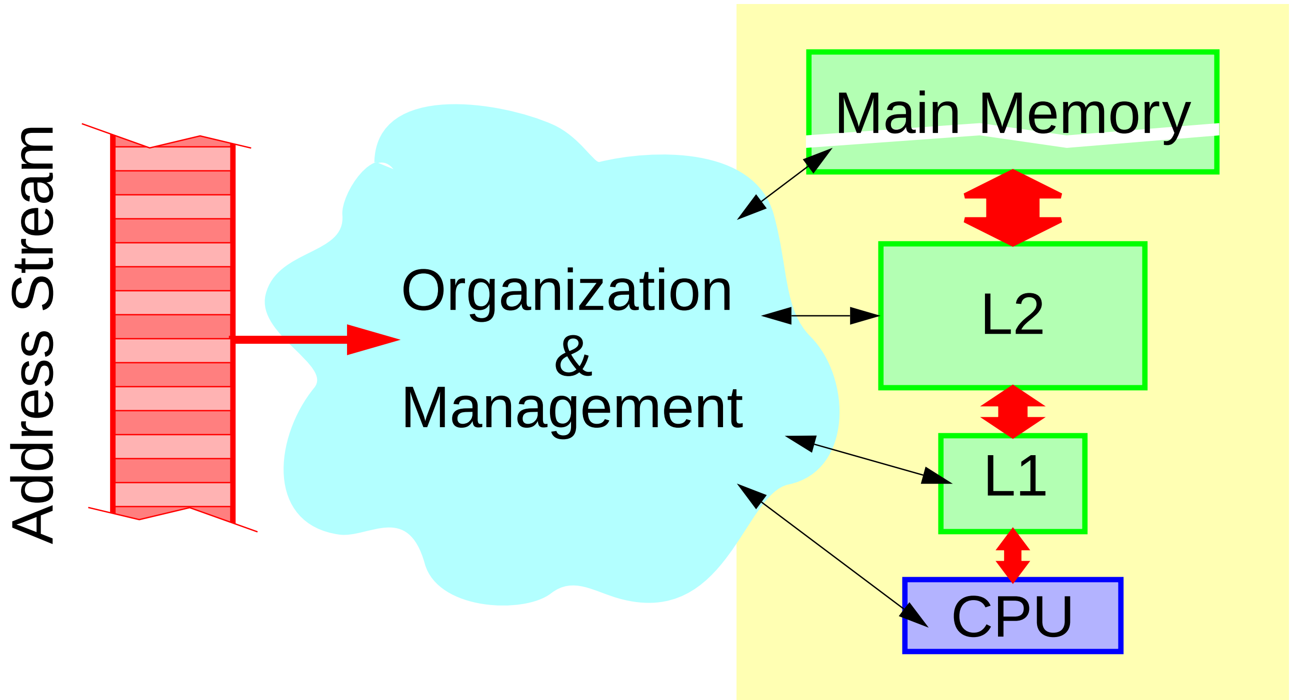



80%
60%
40%
20%
0%
100%
80%
60%
40%
20%
0%
go
m88ksim
gcc
compress
li
ijpeg
perl
vortex
100%
80%
60%
40%
20%
0%
100%
80%
60%
40%
20%
0%
tomcatv
swim
su2cor
hydro2d
mgrid
applu
turb3d
apsi
fpppp
wave5

# The “Memory Problem”

**Store & Retrieve Values with:** **1. Low Latency** **2. High Bandwidth**

Not all storage can be built this way → Intelligent Mechanisms

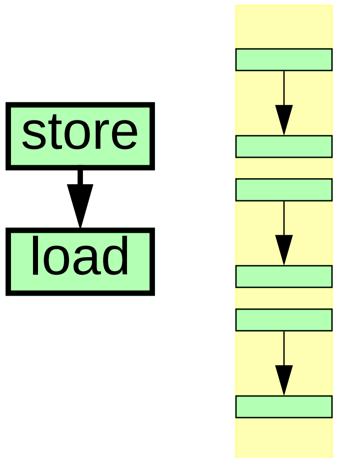



**Name-Centric Approach**

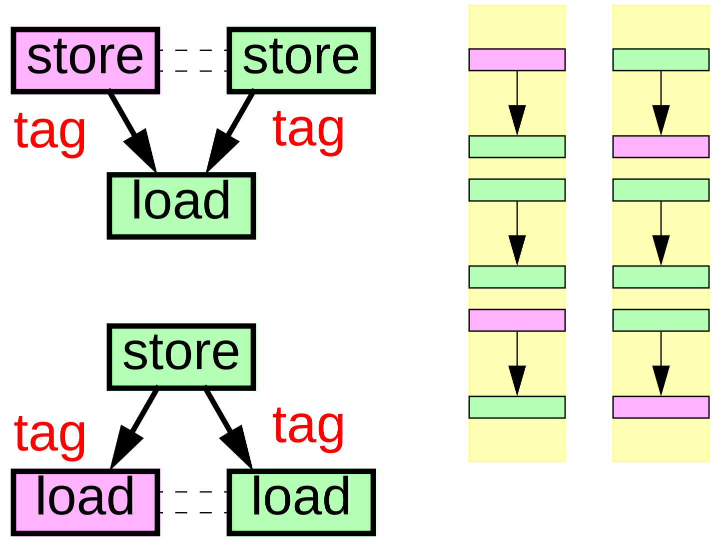
**observe and exploit address stream behavior**

# Predicting Dependences - Synonym Generation

**1-on-1 straightforward**




**N-to-N is common**




Break into steps:

**1. Predict dependence status (existence)**

**2. Figure out with who / synonym**

dependences w/ common parties same synonym

execution path determines which is the right one