

# **A Compiler Perspective on Architectural Evolutions**

**Nicholas Mitchell  
Larry Carter  
Jeanne Ferrante**

**University of California, San Diego**

# ***Introduction***

▶ **optimizing** compilers exist

naive code

simple (high-level) code

▶ compilers **fail** to optimize

complicated?

impossible?

**Certain architectural features**

**hinder or inhibit**

**optimizations.**

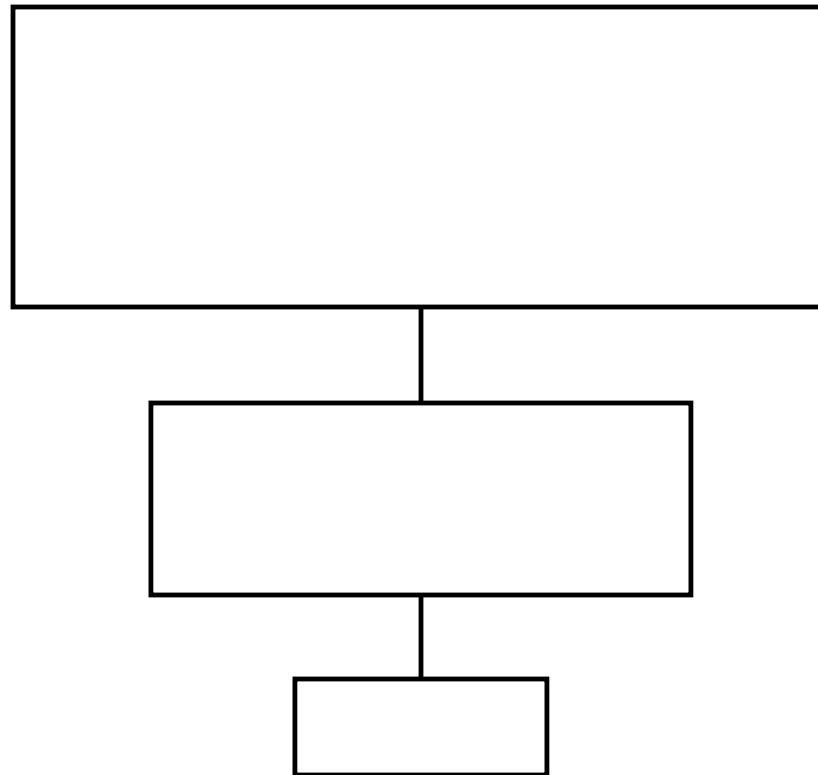
**We suggest three changes.**

- ▶ **Wide memory**
- ▶ **Explicit control**
- ▶ **Associativity**

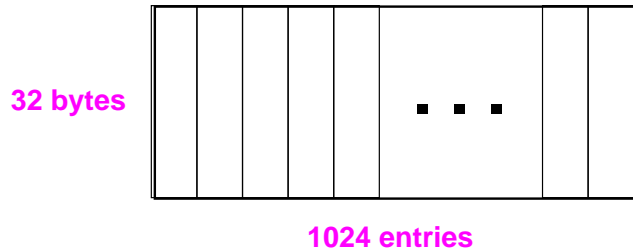
***Wide Memory***

# Wide memory principle

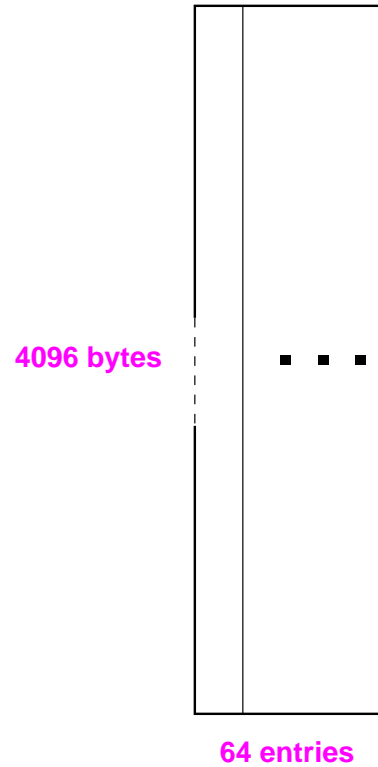
Larger capacity modules must be **wider** than smaller capacity modules.



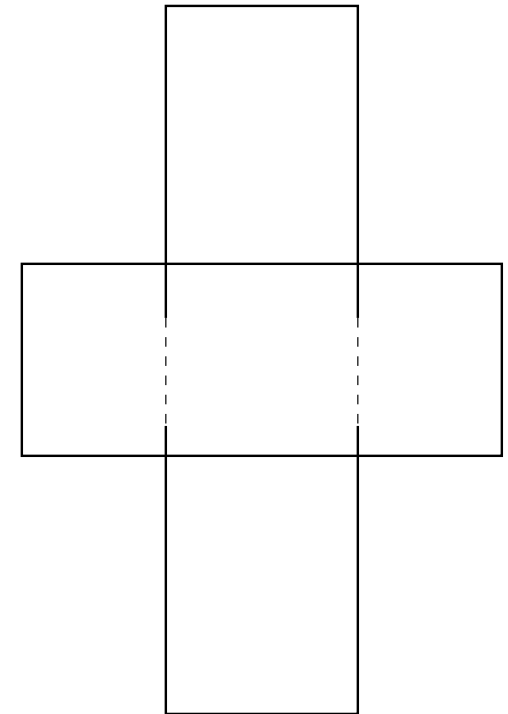
# Memory shapes



most caches are squat



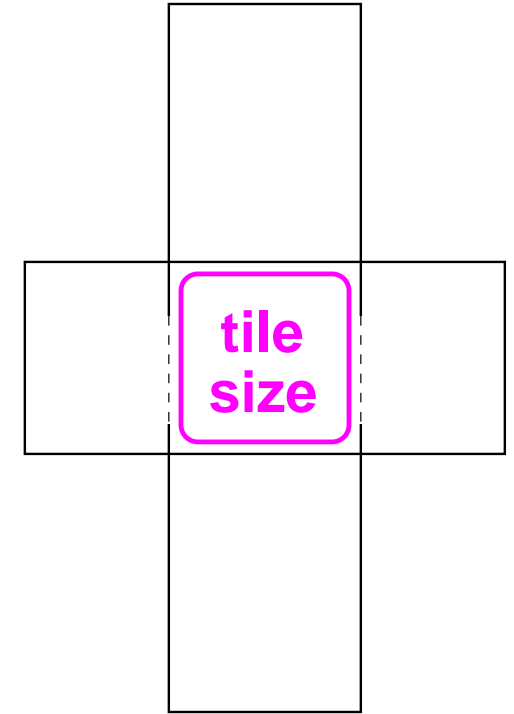
TLBs are skinny



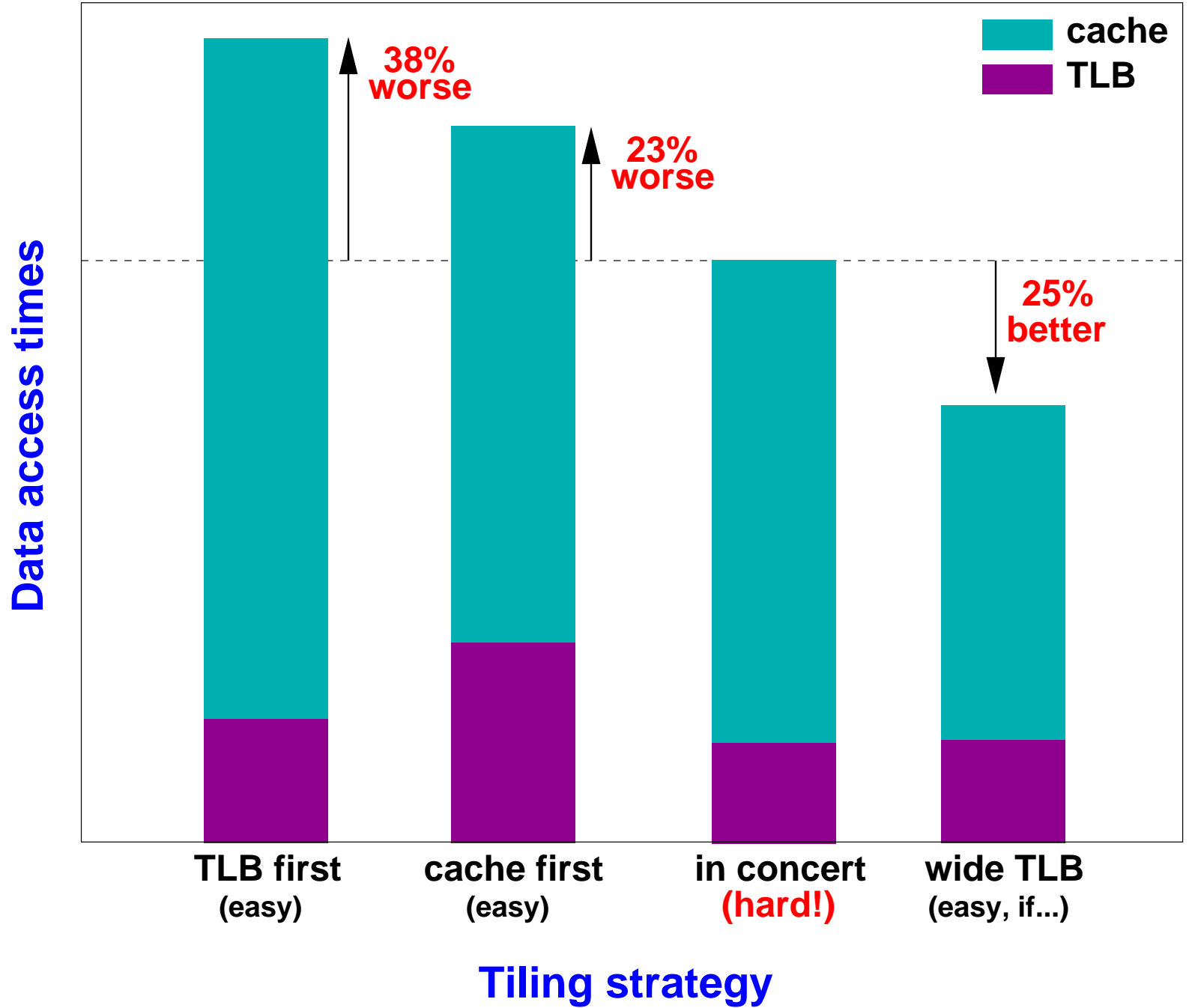
**bad:** neither one contains the other

# Wide memory principle: Why?

- ▶ **better performance**  
no "needless" subtiling



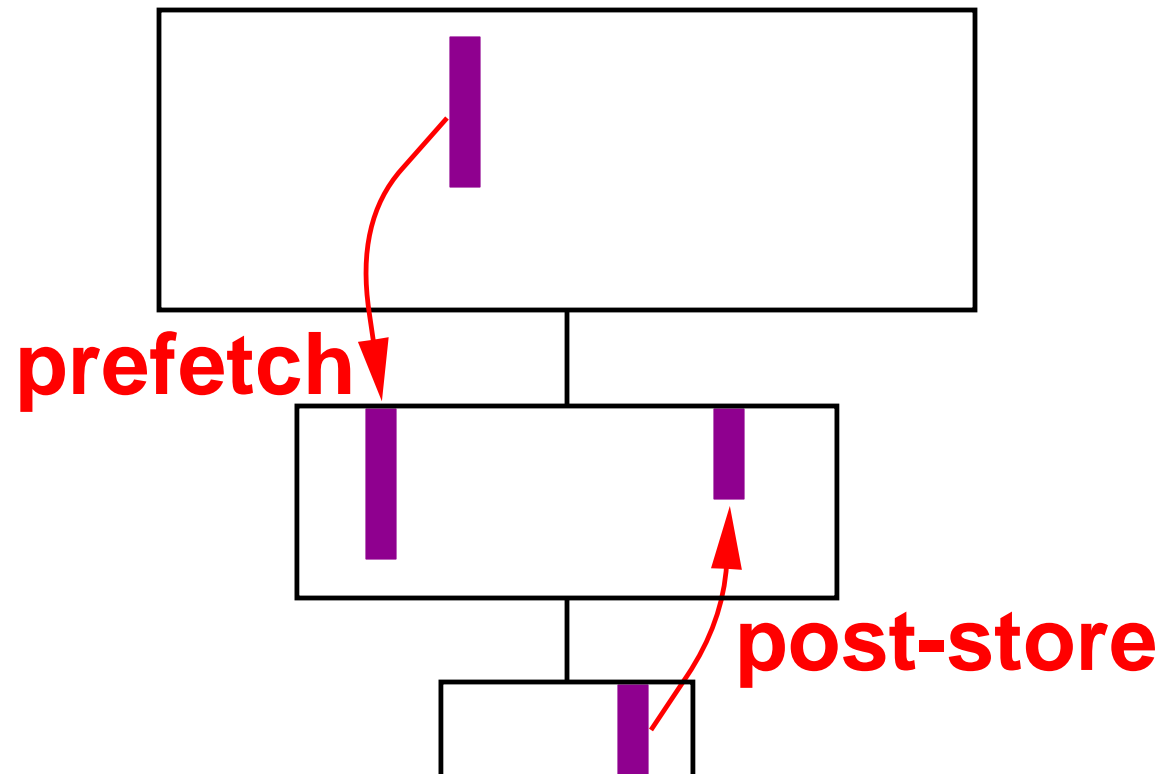
- ▶ **simpler optimization**  
recursively improve locality



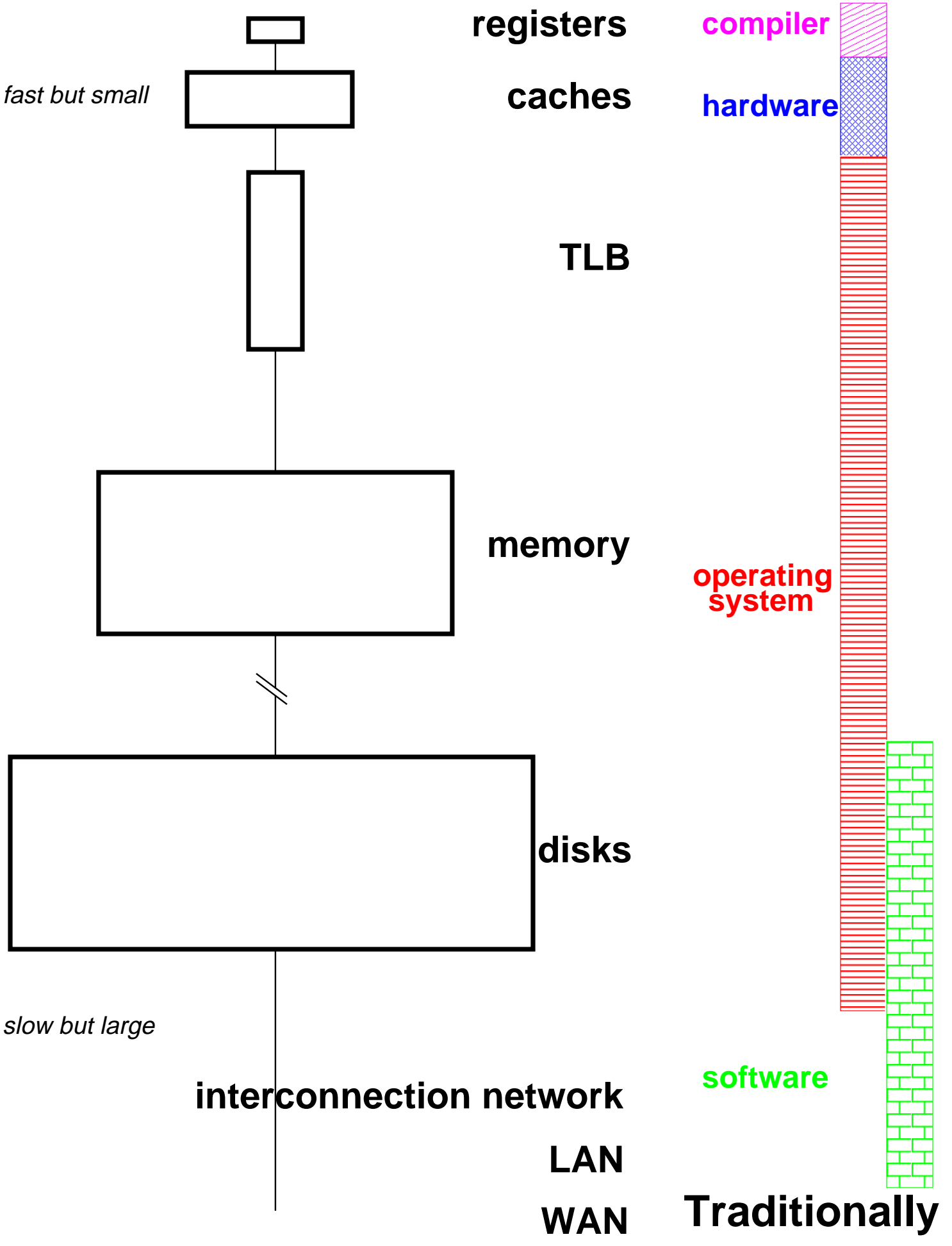
# *Explicitly Controlled Memory Hierarchy*

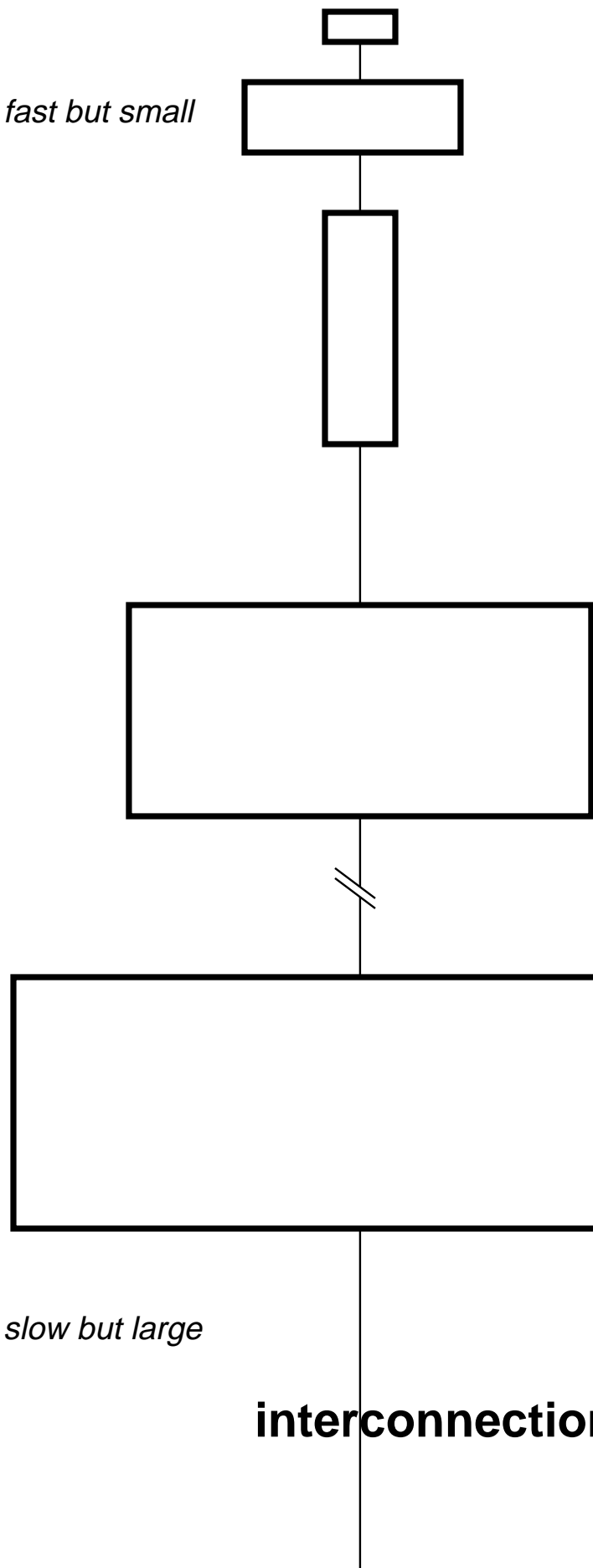
# Explicitly controlled hierarchy

Hardware must allow compilers to **choreograph data motion** throughout the memory hierarchy.



***Domains of control  
in the memory hierarchy***





**registers**

**caches**

*cache prefetch*  
(R10000, PlayDoh)

**TLB**

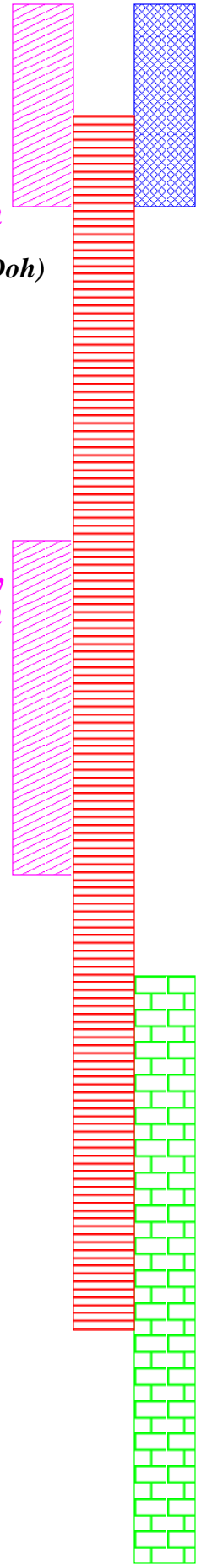
**memory**

*memory prefetch*  
(PlayDoh)

**disks**

**LAN**

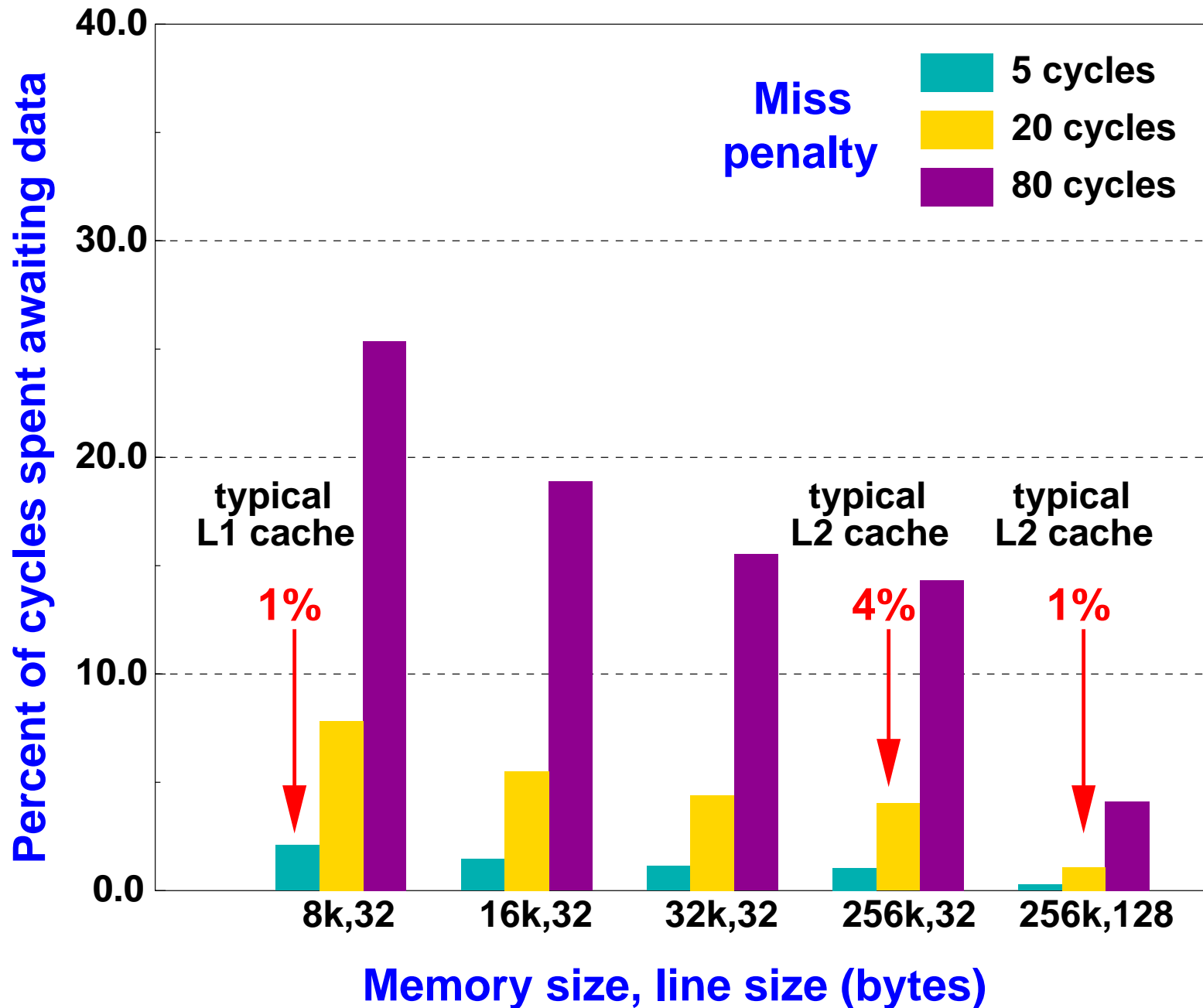
**WAN**



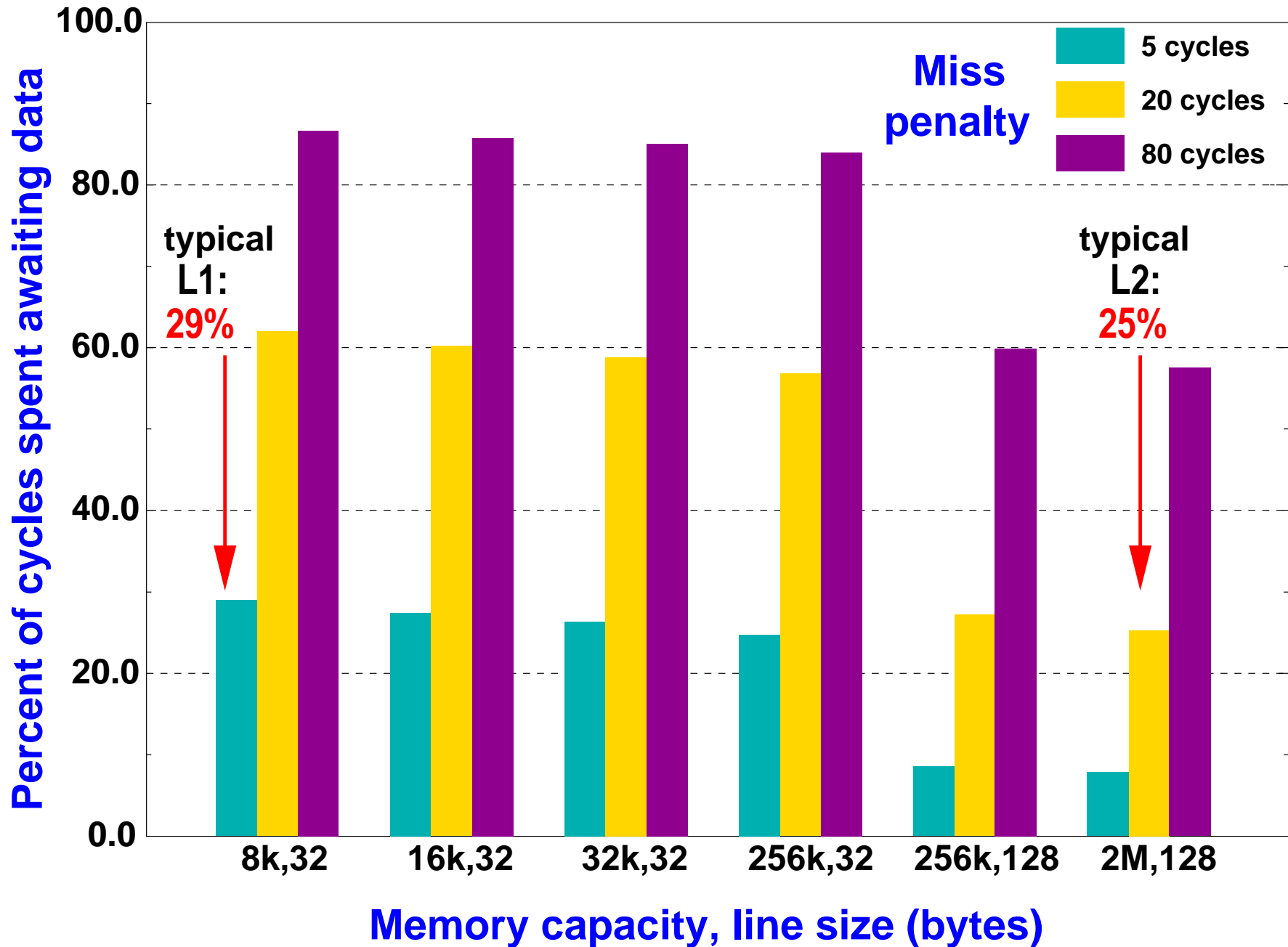
*dynamic dataflow*  
(Pentium Pro)  
**SRAM main memory**

**Trends**

# Prefetch, tiled matrix multiply



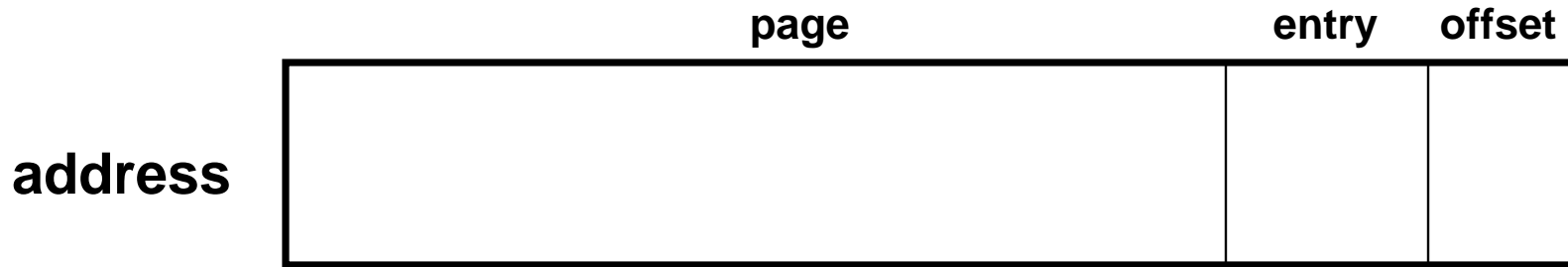
# Prefetch, matrix-vector product



# *Problems with Associativity*

# Associativity problem

**Associativity leads to unpredictable performance when caches are physically addressed.**

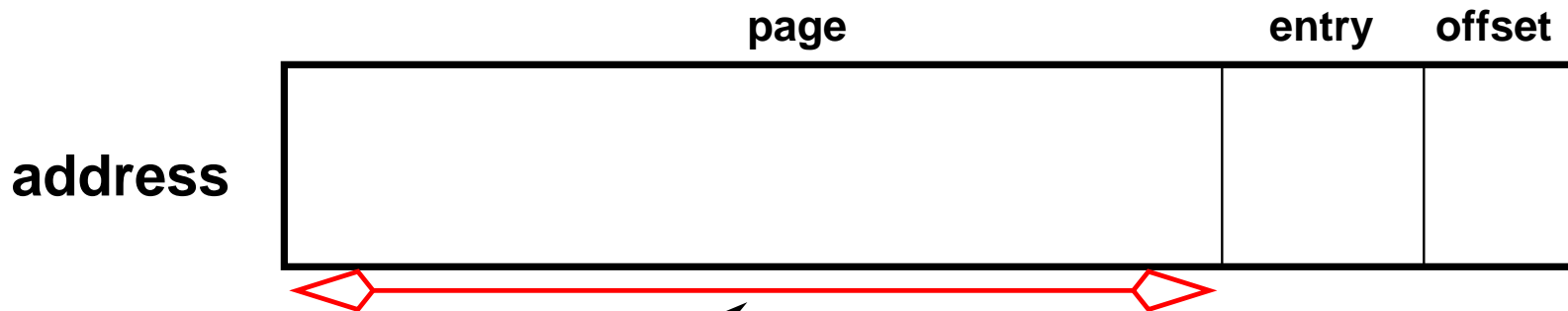


low order bits of page number in part determine **associativity class**

**virtual** address, **deterministic** class

**Problem:**

**physical** address, **unknowable** class



hash page number to determine **associativity class**

even with physical addresses,  
**Poisson model** predicts  
distribution to classes

# Conclusions

code optimization

versus

hardware optimization

- ▶ different objectives
- ▶ lack of communication

## Problem

architectural features  
**complicate** or **prevent**  
code optimization

## Solution

interactions!  
work together to optimize