WaveScalar: the Executive Summary

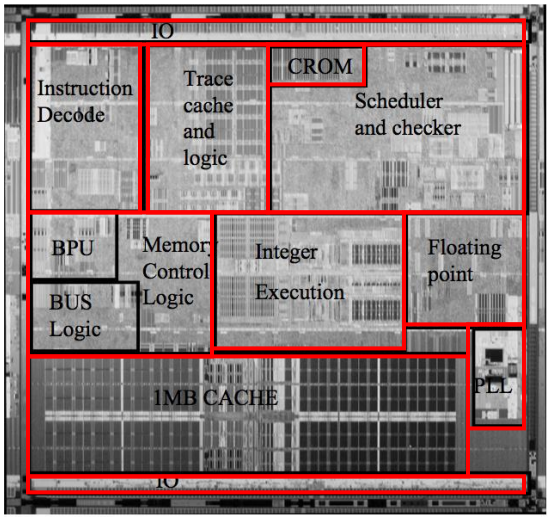
A modern dataflow computer
- solves the language compatibility & memory ordering issues
- solves the scalability issue

The executive summary:
- good at exploiting ILP (dataflow parallelism)
- also traditional coarser-grain parallelism
  - cheap thread management
- low operand latency because of a hierarchical PE-interconnect organization
- memory ordering enforced through **wave-ordered memory**
  - can execute imperative language programs
  - no special dataflow languages

WaveScalar

Motivation stems from shrinking feature sizes:
- increasing disparity between computation (fast transistors) & communication (long wires)
- increasing circuit complexity
- decreasing fabrication reliability

Monolithic von Nuemann Processors



A success a few years ago.
But in 2016?

☹ Performance
Centralized processing & control
Long wires
e.g., operand broadcast networks

☹ Complexity
40-75% of "design" time is design verification

☹ Defect tolerance
1 flaw -> tie pin, earrings, …

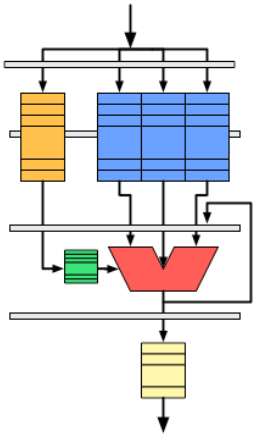
WaveScalar's Microarchitecture

Good performance via distributed microarchitecture ☺
- hundreds of PEs
- organized hierarchically for fast communication between neighboring PEs
- short point-to-point (producer to consumer) operand communication
- dataflow execution – no centralized control via a PC
- consequently scalable

Low design complexity through simple, identical PEs ☺
- design one & stamp out hundreds

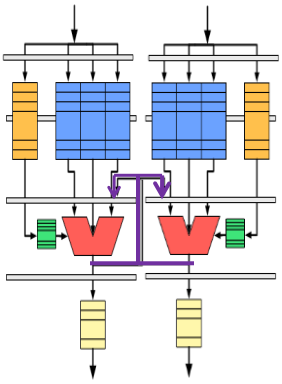
Defect tolerance ☺
- route around a bad PE

Processing Element



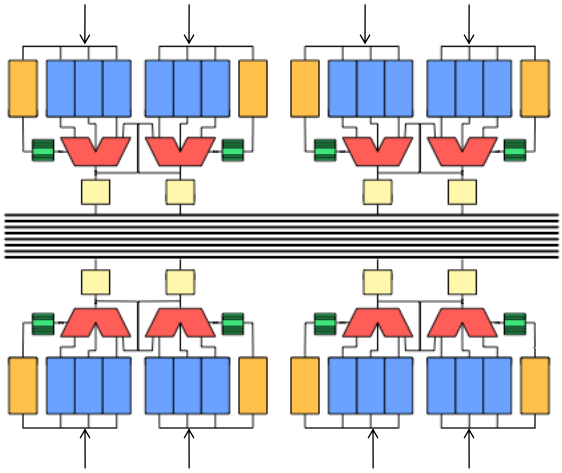
- Simple, small (.5M transistors)
- 5-stage pipeline (receive input operands, match tags, instruction issue, execute, send output)
- Holds 64 (decoded) instructions
- 128-entry token store
- 4-entry output buffer

PEs in a Pod

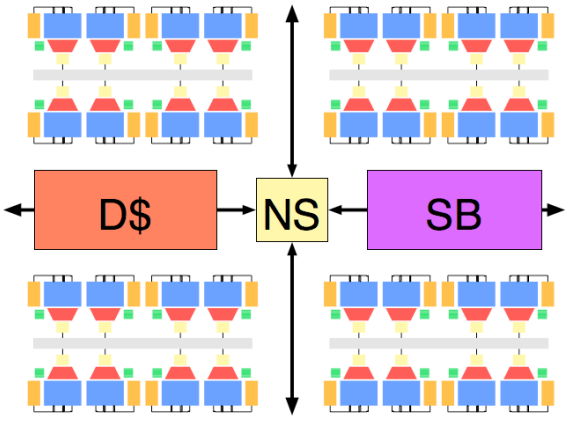


- Share operand bypass network
- Back-to-back producer-consumer execution across 2 PEs

Domain

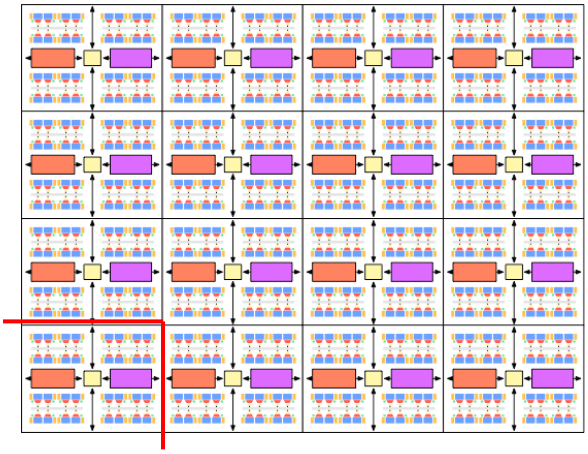


Cluster



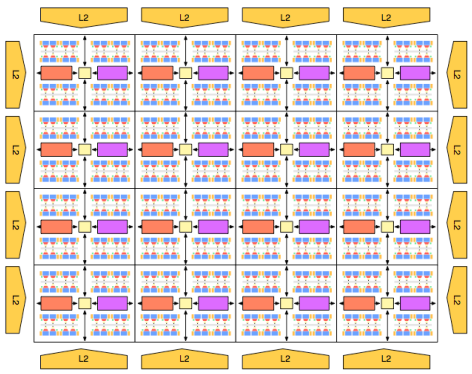
WaveScalar Processor

Long distance communication
- grid-based network
- 2-cycle hop/cluster
- dynamic routing



Whole Chip

- Can hold 32K instructions
- Normal memory hierarchy
- Traditional directory-based cache coherence
- ~400 mm^2 in 90 nm technology
- 1GHz.
- ~85 watts



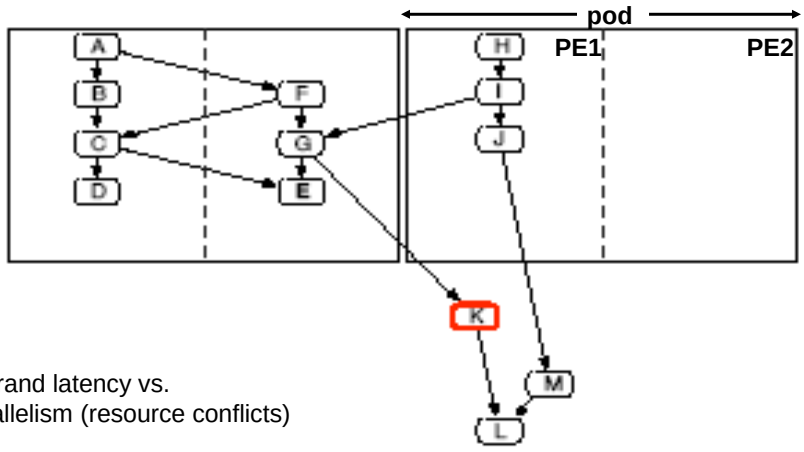
WaveScalar's Microarchitecture

Good performance via distributed microarchitecture ☺
- hundreds of PEs
- organized hierarchically for fast communication between neighboring PEs
- short point-to-point (producer to consumer) operand communication
- dataflow execution – no centralized control via a PC
- consequently scalable

Low design complexity through simple, identical PEs ☺
- design one & stamp out hundreds

Defect tolerance ☺
- route around a bad PE

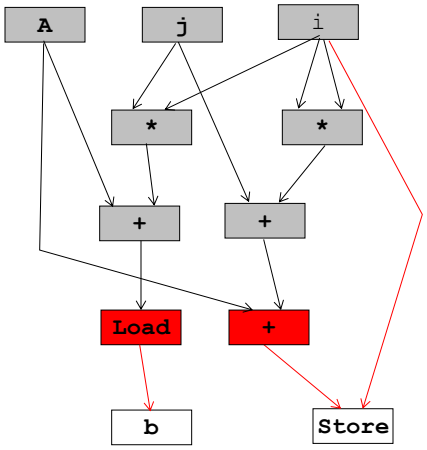
WaveScalar Instruction Placement



operand latency vs.
parallelism (resource conflicts)

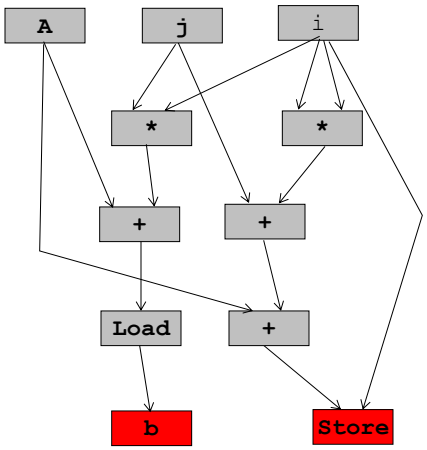
Revisit Example to Illustrate the Memory Ordering Problem

```
A[j + i*i] = i;

b = A[i*j];
```



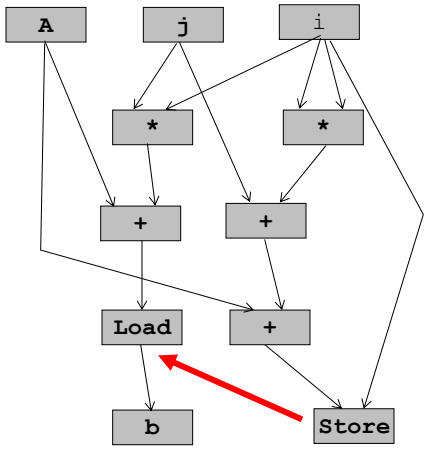
Revisit Example to Illustrate the Memory Ordering Problem

```
A[j + i*i] = i;

b = A[i*j];
```



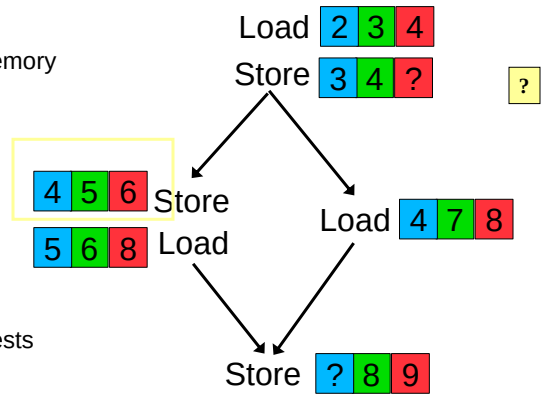
Revisit Example to Illustrate the Memory Ordering Problem

```
A[j + i*i] = i;

b = A[i*j];
```

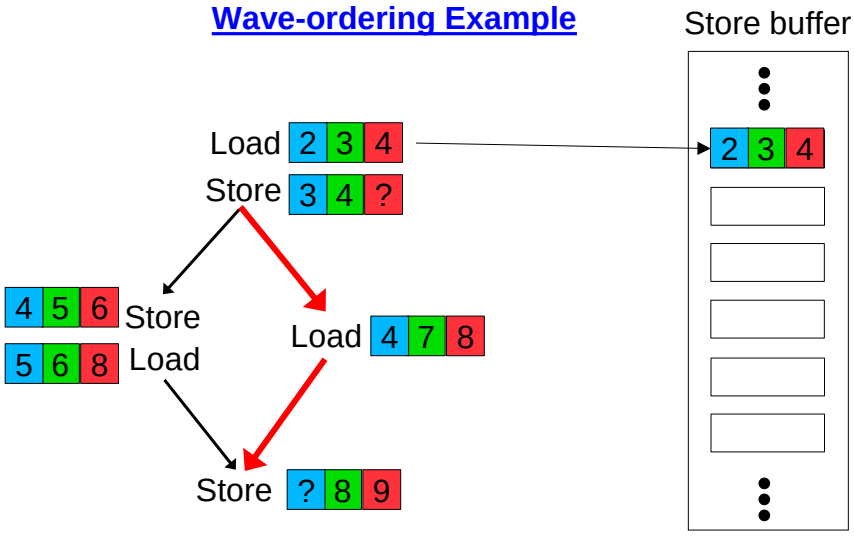


Wave-ordered Memory

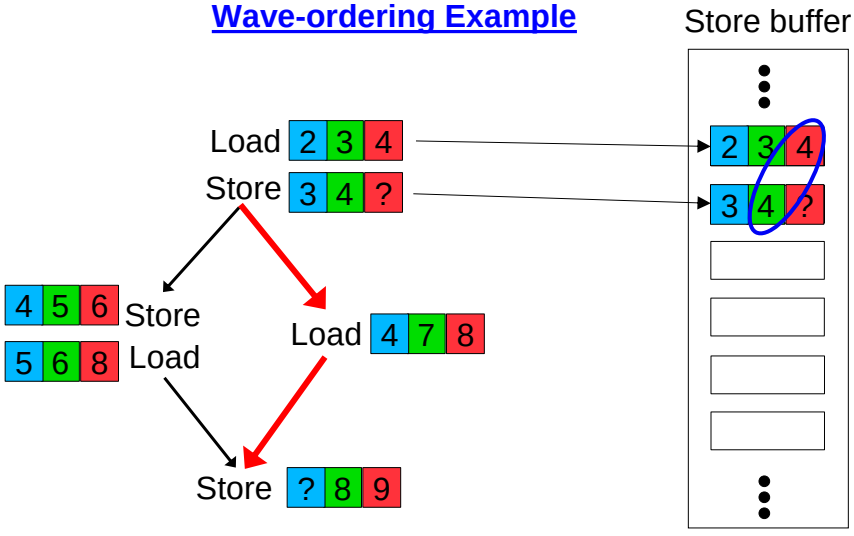
- Compiler annotates memory operations
  - Sequence #
  - Successor
  - Predecessor
- Execute memory requests in any order
- Store buffer hardware reconstructs the correct order



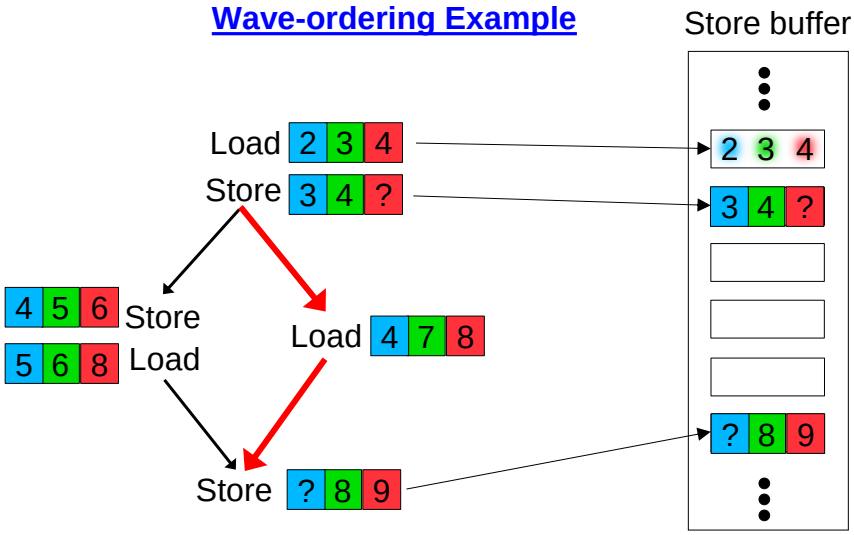
Wave-ordering Example



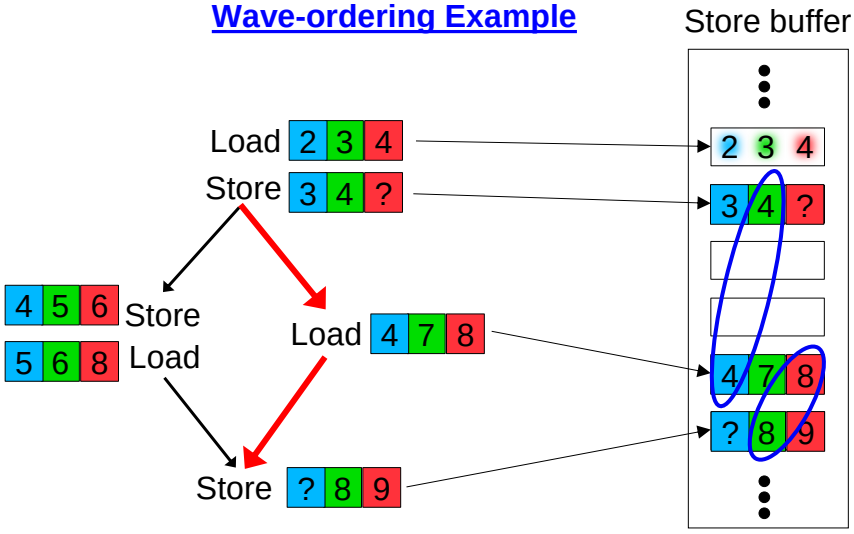
Wave-ordering Example



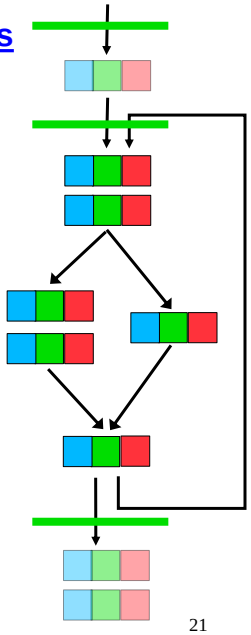
Wave-ordering Example



Wave-ordering Example



Wave-ordered Memory across Waves



Waves are loop-free sections of the dataflow graph

Each *dynamic* wave has a **wave number**

Wave number is incremented between waves

Ordering memory in a whole program:

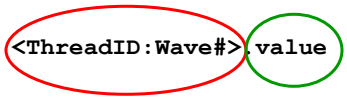
- wave-numbers
- sequence numbers within a wave
- can execute imperative language programs

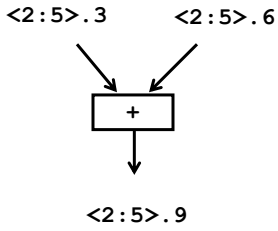
WaveScalar Tag-matching

WaveScalar tag

- thread identifier
- wave number

Token: **tag** & **value**



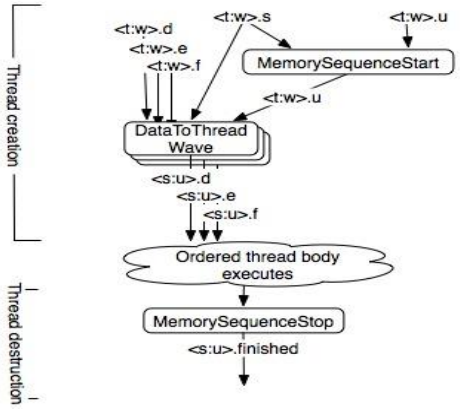


Multithreading the WaveCache

Architectural-support for WaveScalar threads

- instructions to start & stop memory orderings, i.e., threads
- memory-free synchronization to allow exclusive access to data
- "barrier" or "fence" instruction to force all previous memory operations to fully execute (to allow other threads to see the results of this one's memory operations)

Creating & Terminating a Thread



Multithreading the WaveCache

Combine to build threads with multiple granularities

- coarse-grain threads: 25-168X over a single thread; 2-16X over CMP, 5-11X over SMT
- fine-grain, dataflow-style threads: 18-242X over single thread
- a demonstration that one can combine the two in the same application (`equake`): 1.6X or 7.9X -> 9X

Important Issues

Original dataflow machines

- comparison to von Neumann architectures
- dataflow firing rule
- token
- handling branches
- problems

Modern dataflow machines, aka Wavescalar

- comparison to von Neumann microarchitecture
- how solve past dataflow problems
- hierarchical structure
- wave-ordered memory
- thread management