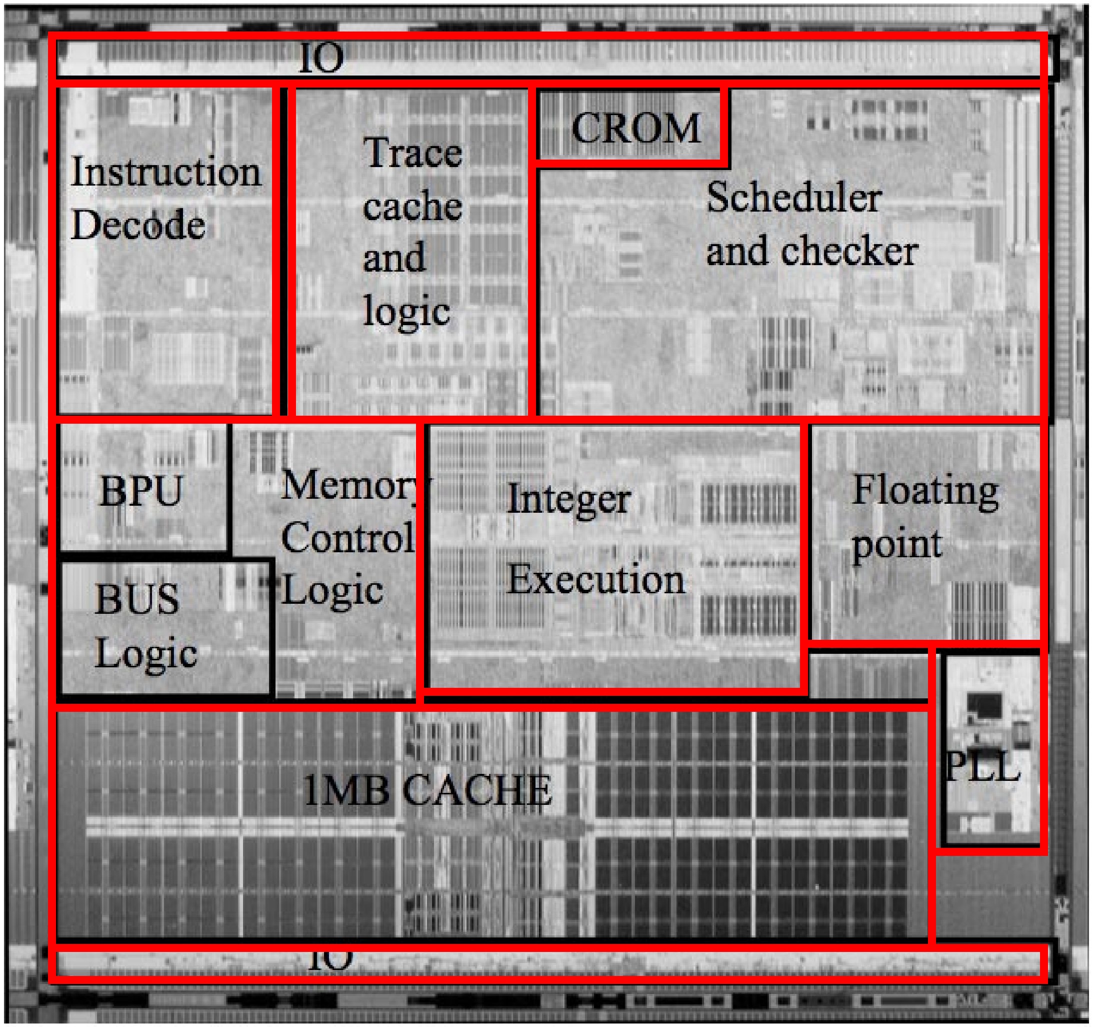
WaveScalar

Motivation:

- increasing disparity between computation (fast transistors) & communication (long wires)
- increasing circuit complexity
- decreasing fabrication reliability

Monolithic von Neumann Processors



A phenomenal success today.

But in 2016?

L Performance
Centralized processing & control,
e.g., operand broadcast networks
Deeper pipelines & wider issue
not possible

L Complexity
40-75% of “design” time is design
verification

L Defect tolerance
1 flaw -> paperweight

WaveScalar Executive Summary

Dataflow machine J

- good at exploiting ILP
- dataflow parallelism + traditional coarser-grain parallelism
- cheap thread synchronization
- memory ordering enforced through **wave-ordered memory**

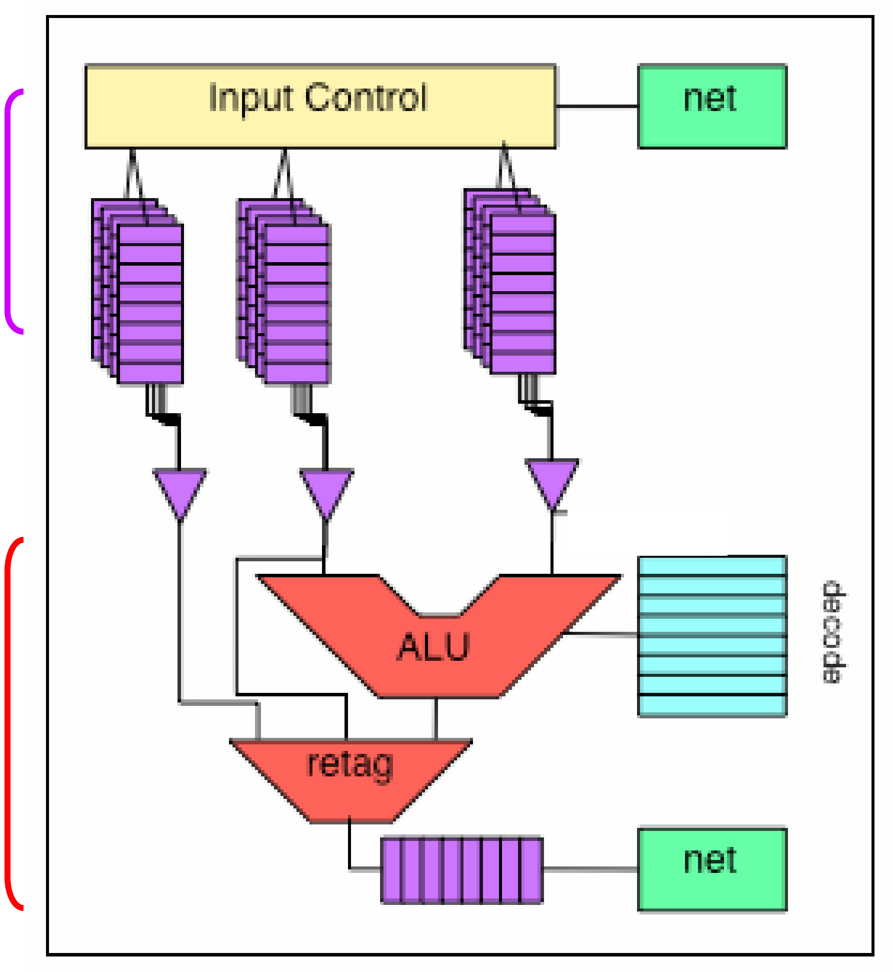
Distributed microarchitecture J

- hundreds of PEs
- short point-to-point communication for neighbor PEs
- organized hierarchically for fast communication between any PEs
- dataflow execution – no centralized control

Defect tolerance & low design complexity through simple, identical PEs J

- design one & stamp out thousands

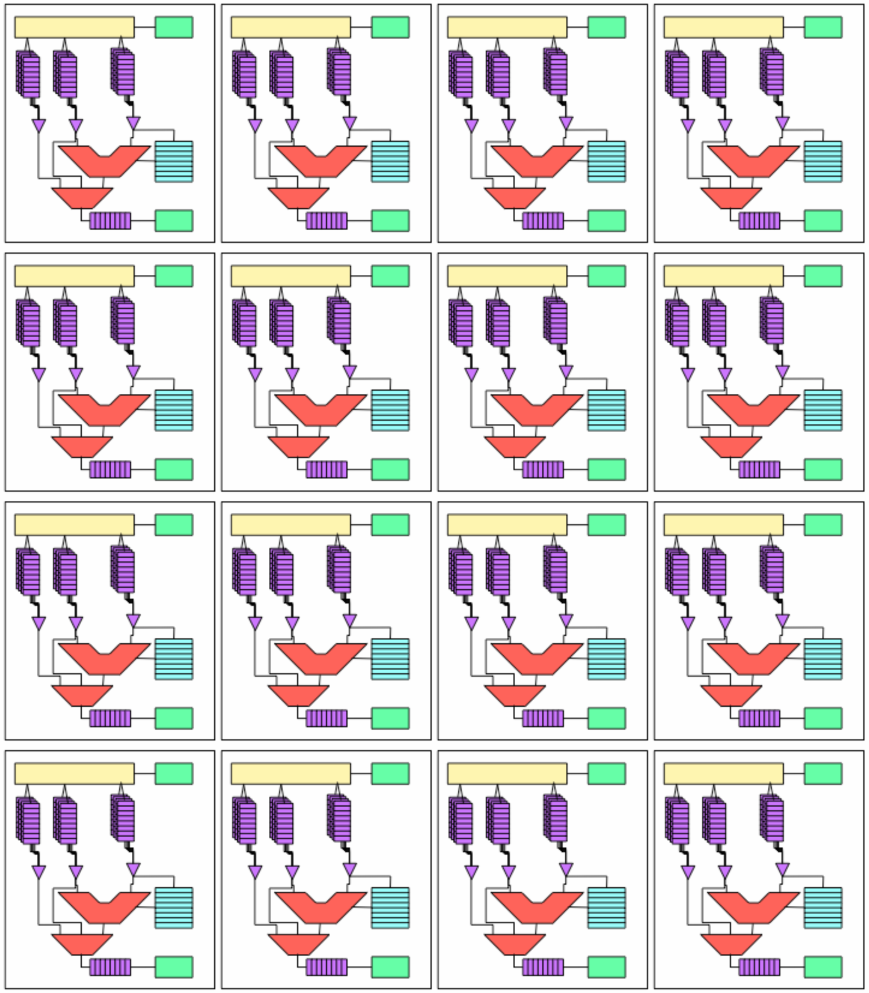
Processing Element



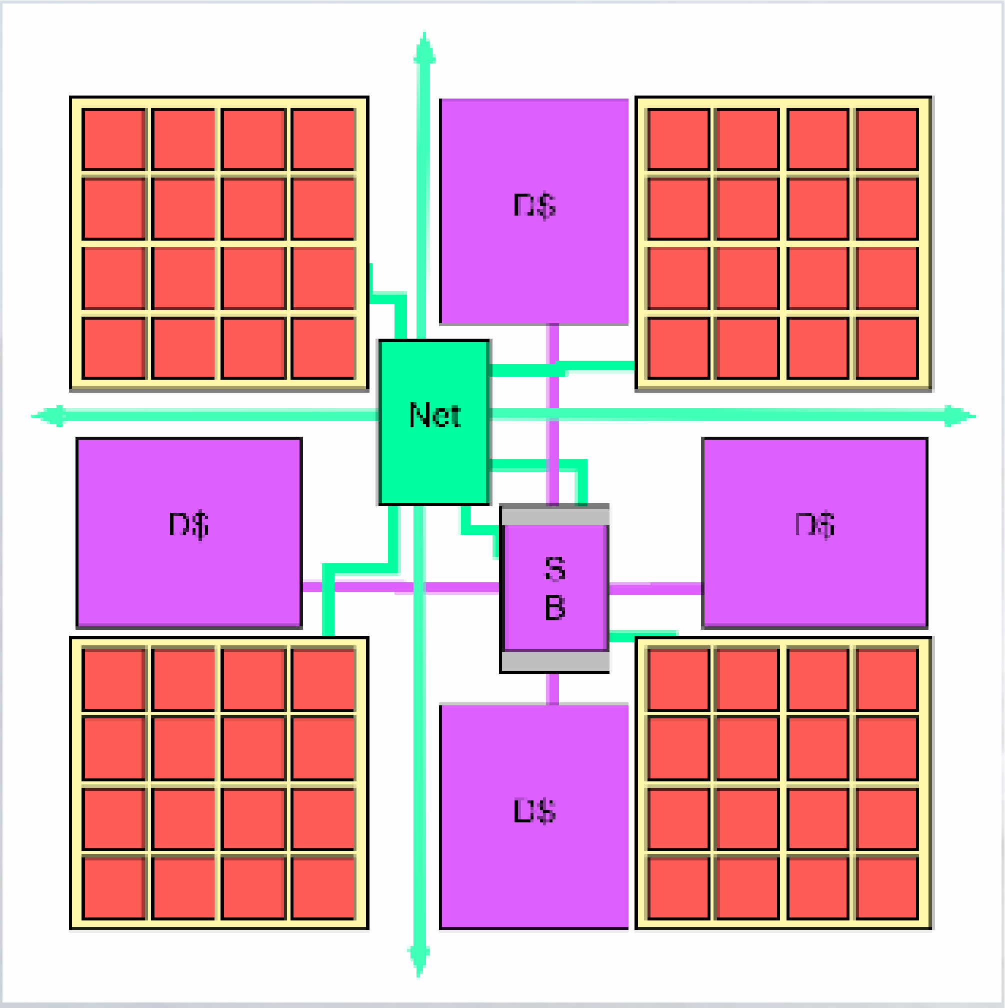
distributed
tag matching

2 PEs
in a pod

Domain

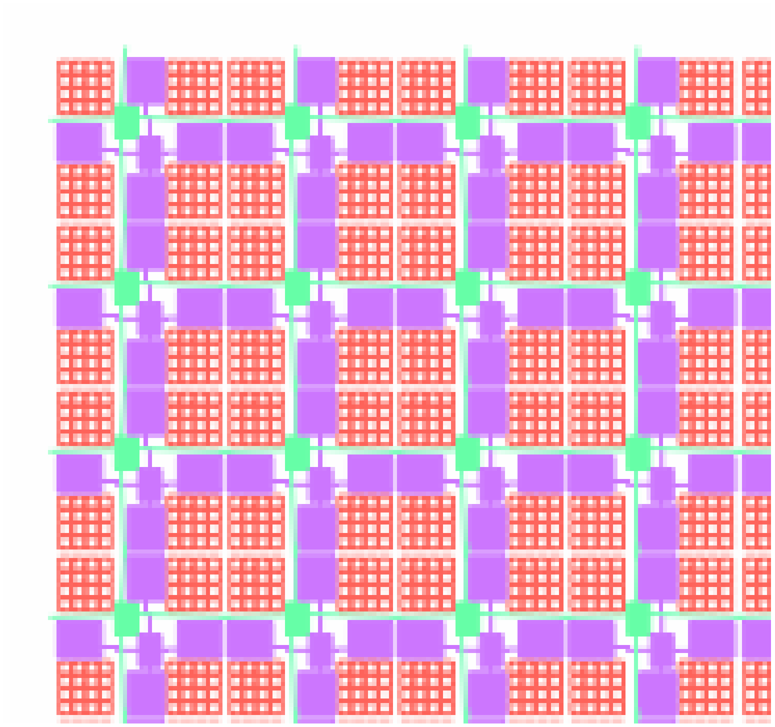


Cluster



The WaveCache

- Can hold 32K instructions
- Long distance communication
  - Dynamic routing
  - Grid-based network
  - 2-cycle hop/cluster
- Normal memory hierarchy
- Traditional directory-based cache coherence



WaveScalar Execution Model

Place instructions in PEs to maximize locality & parallelism.

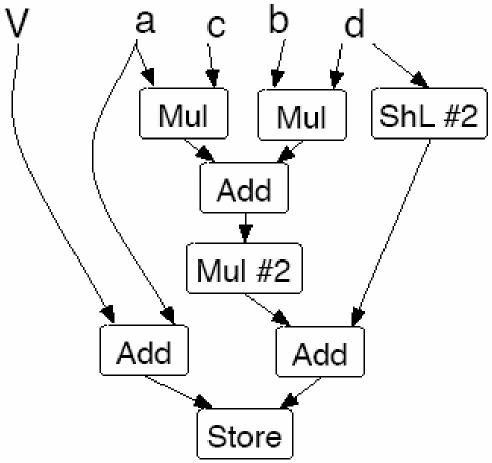
- Instruction placement algorithm based on a performance model
  - cost (operand latency of closely placed instructions)
  - benefit (increased parallelism of dispersed instructions)

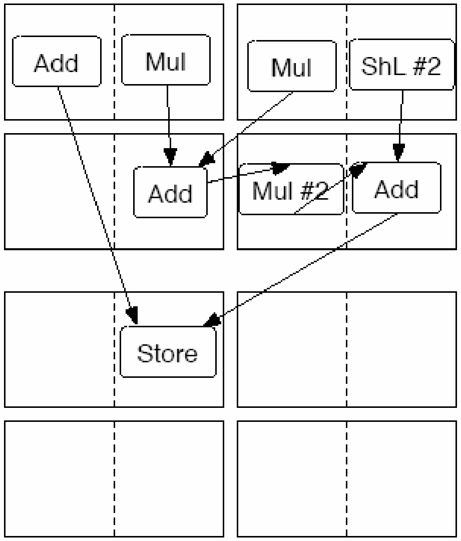
Instructions communicate values directly (point-to-point).

No processor core, no PC.

WaveScalar Instruction Placement

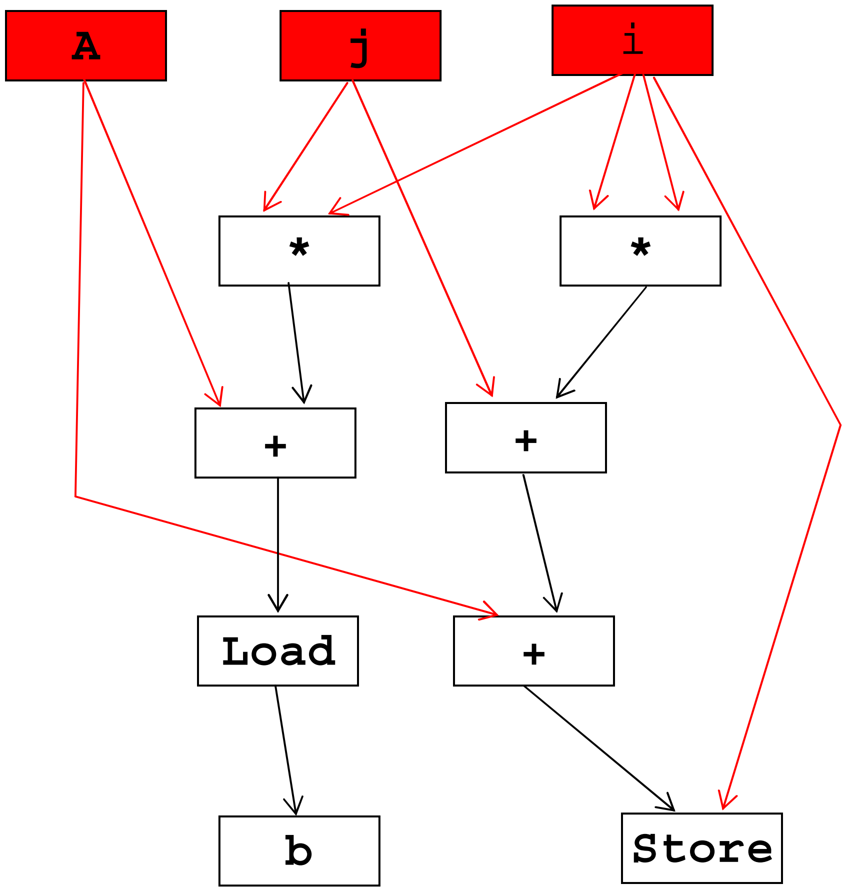
```
int *V;
int a, b;
int c, d, r;

r = a*c + b*d;
V[a] = 2*r + d << 2;
```





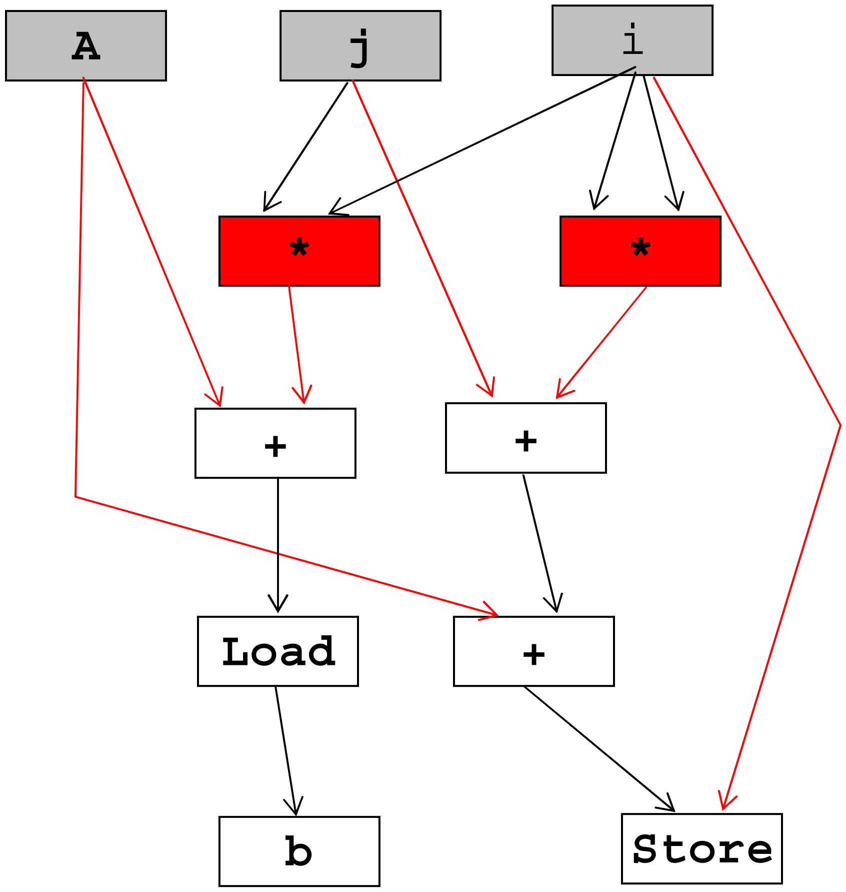
WaveScalar Example

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

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



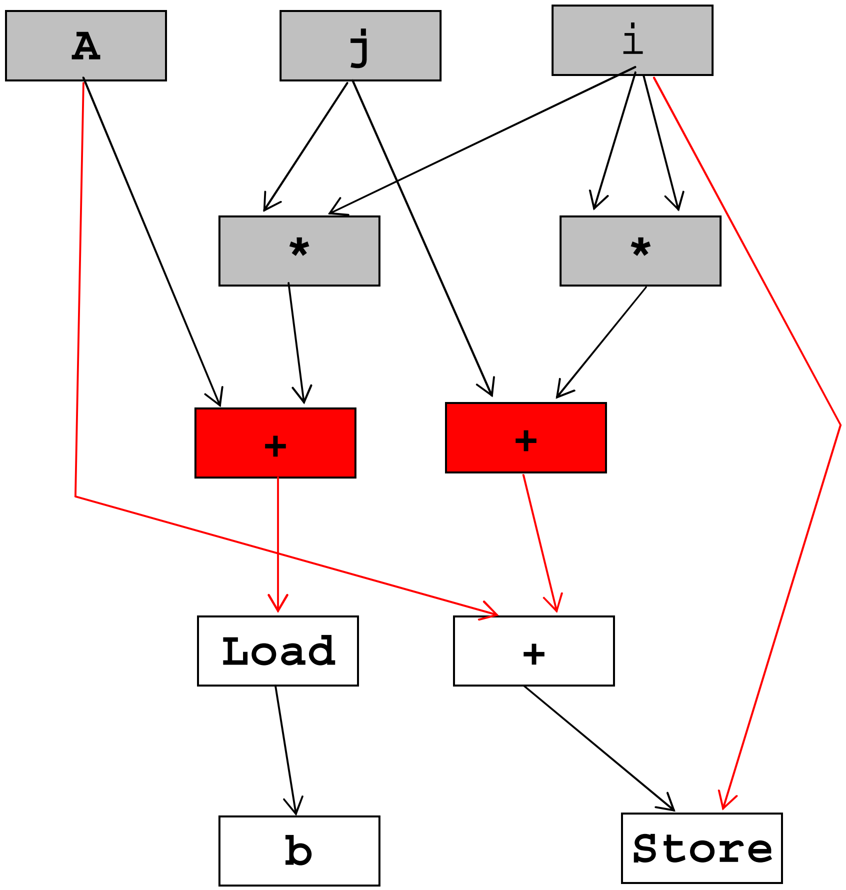
WaveScalar Example

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

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



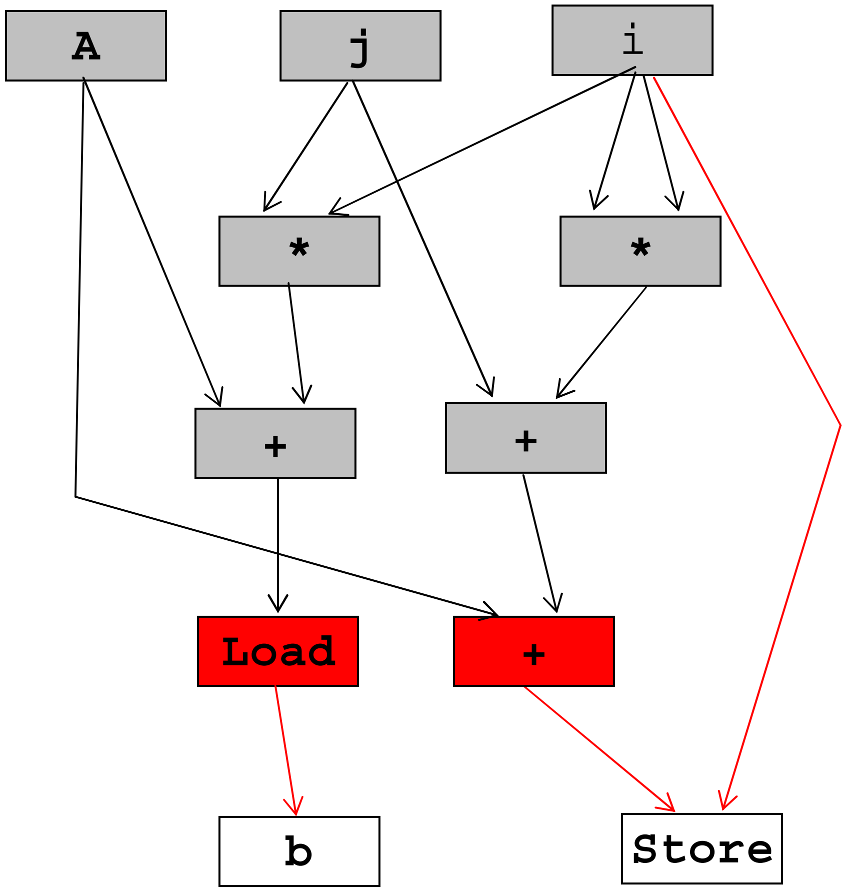
WaveScalar Example

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

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



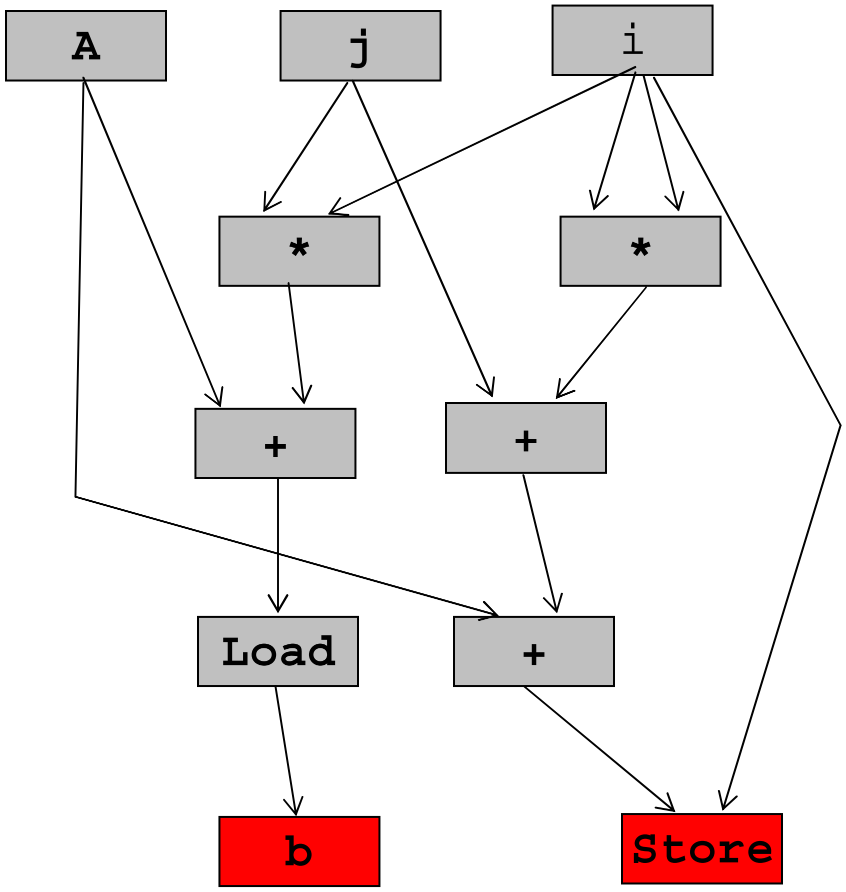
WaveScalar Example

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

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



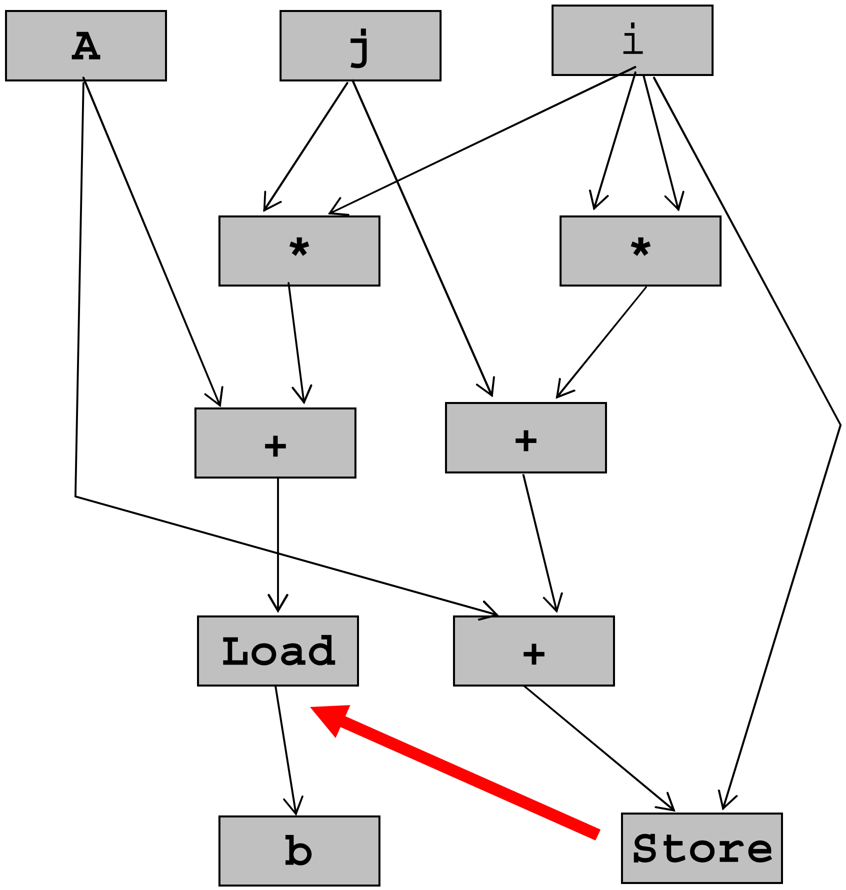
WaveScalar Example

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

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



WaveScalar Example

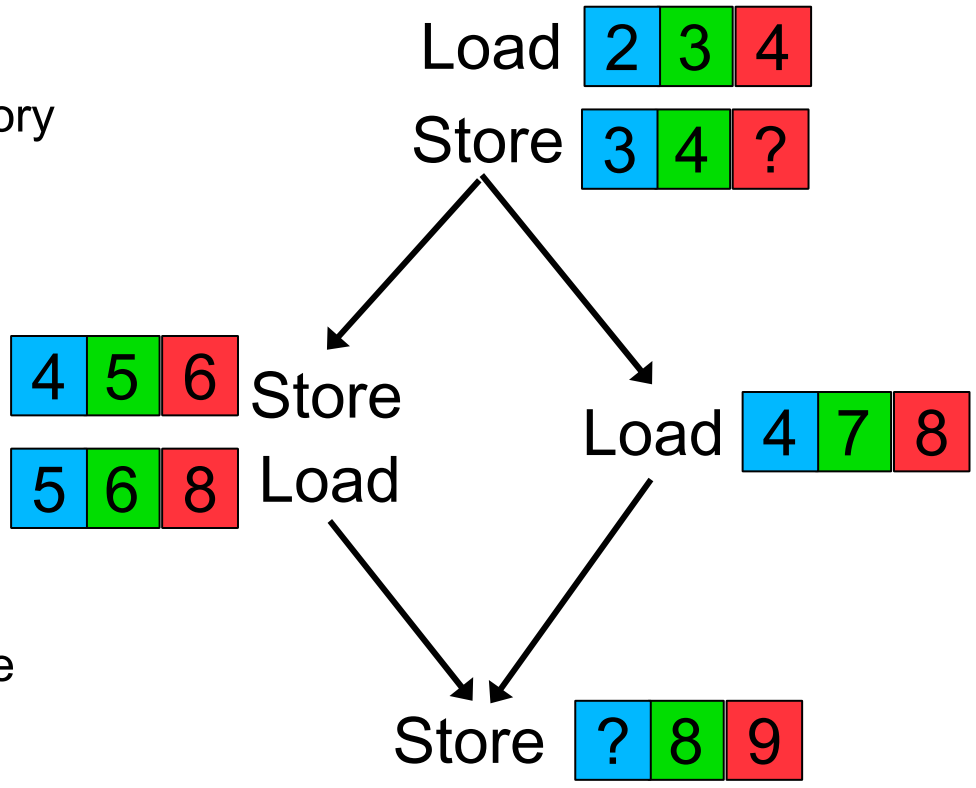
```
A[j + i*i] = i;

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



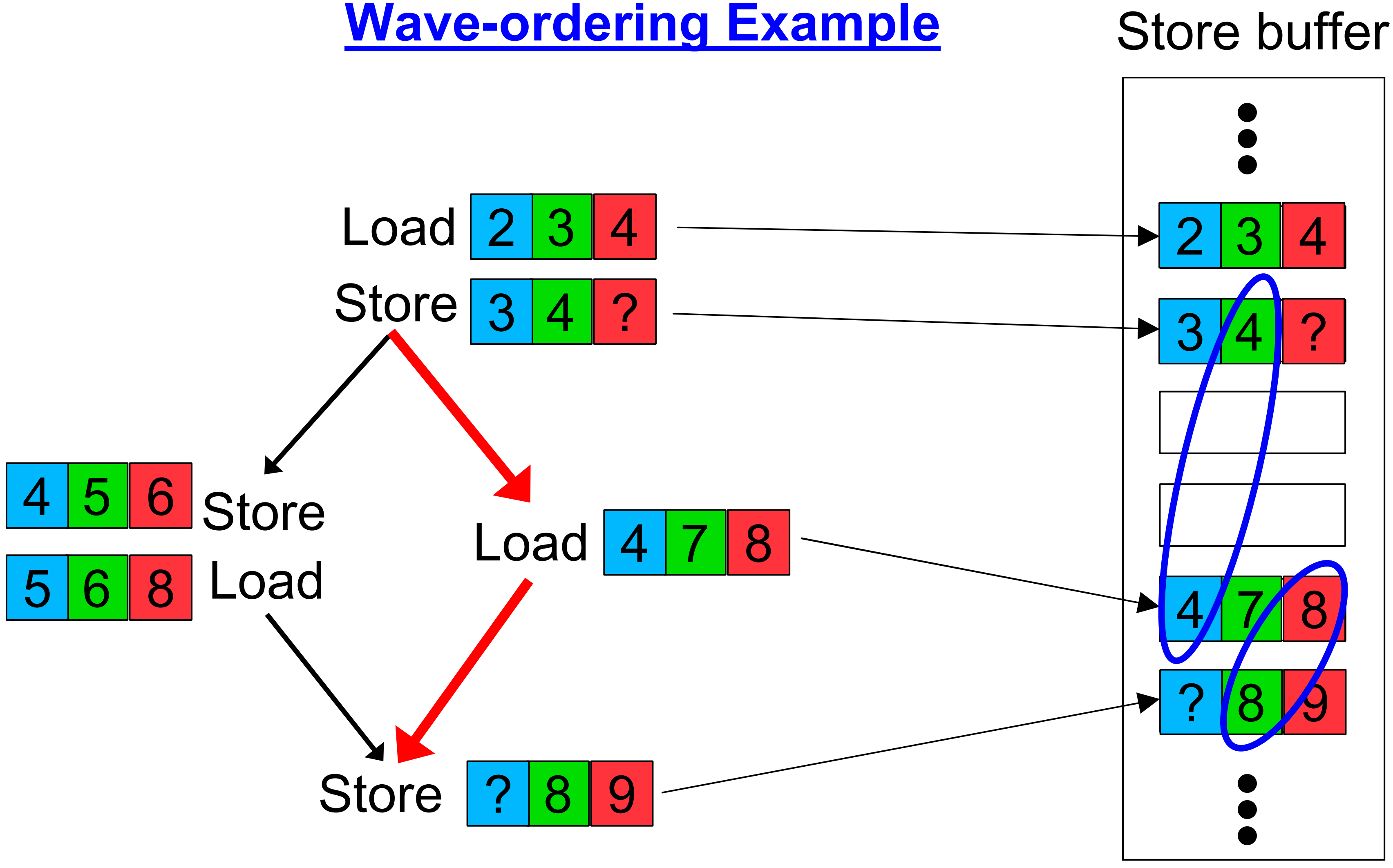
Global load-store ordering issue

Wave-ordered Memory

- Compiler annotates memory operations
    - n Sequence #
    - n Successor
    - n Predecessor
- Send memory requests in any order
- Hardware reconstructs the correct order



Wave-ordering Example



Wave-ordered Memory

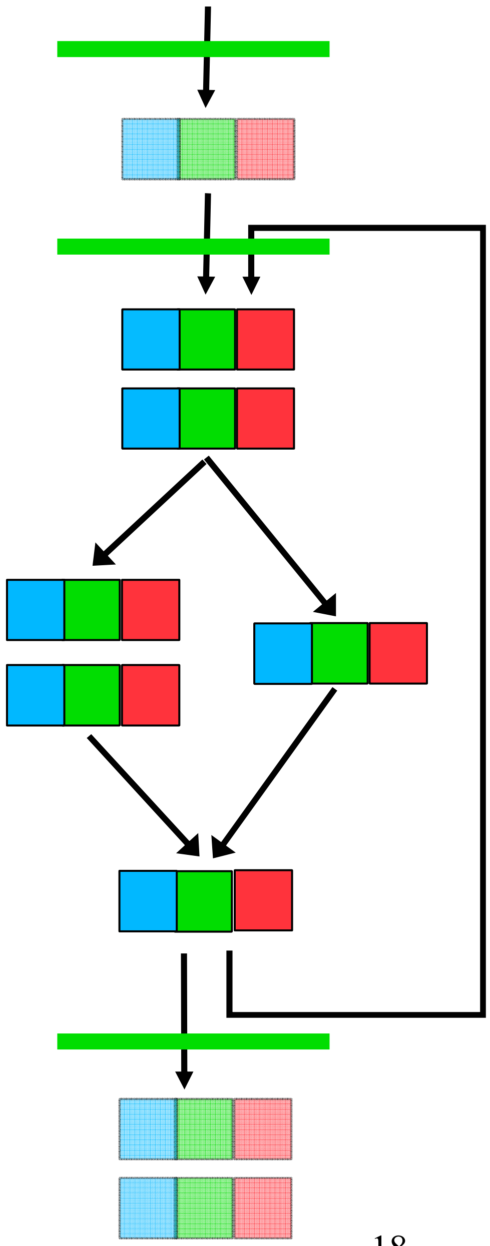
Waves are loop-free sections of the dataflow graph

Each dynamic wave has a **wave number**

Wave number is incremented between waves

Ordering memory:

- wave-numbers
- sequence number within a wave



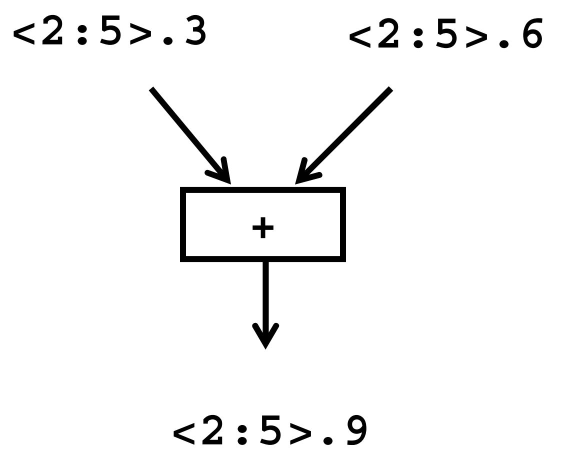
WaveScalar Tag-matching

WaveScalar tag

- thread identifier
- wave number

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

`<ThreadID:Wave#>` . `value`



Multithreading the WaveCache

Architectural-support for WaveScalar threads

- instructions to start & stop memory orderings, i.e., threads
- memory-free synchronization (acts like a hardware queue lock)
- fence instruction to allow relaxed consistency

Combine to build threads with multiple granularities

- coarse-grain threads: 25-168X over a single thread; 10-20X over CMP, 6-18X over SMT
- fine-grain, dataflow-style threads: 18-242X over single thread
- combine the two in the same application: 1.6X or 7.9X -> 9X

Building the WaveCache

RTL-level implementation

- some didn't believe it could be built in a normal-sized chip
- some didn't believe it could achieve a decent cycle time and load-use latencies
- Verilog & Synopsis CAD tools

Different WaveCache's for different applications

- 1 cluster: low-cost, single-thread or embedded
  - 18 mm^2 in 90 nm process technology
- 16 clusters: multiple threads, higher performance: 236 mm^2
  - 87% PEs, 4% L1 D$, 7% store buffers, 2% all networks
- 5-stage pipeline: shortest stage is tag matching, longest is wave-ordered memory