Hardware Success Stories

Chris Rossbach and Calvin Lin

Computer Architecture

The Instruction Set Architecture (ISA)

- The ISA is an example of a successful parallel abstraction

Computer Architecture

The Instruction Set Architecture (ISA)

- The ISA is an example of a successful parallel abstraction

- Today we'll look at microprocessor trends over the years

- What can we learn from this success story?

Parallelism in Hardware

Microprocessors are highly parall

- Consider a block diagram of Pentium processor

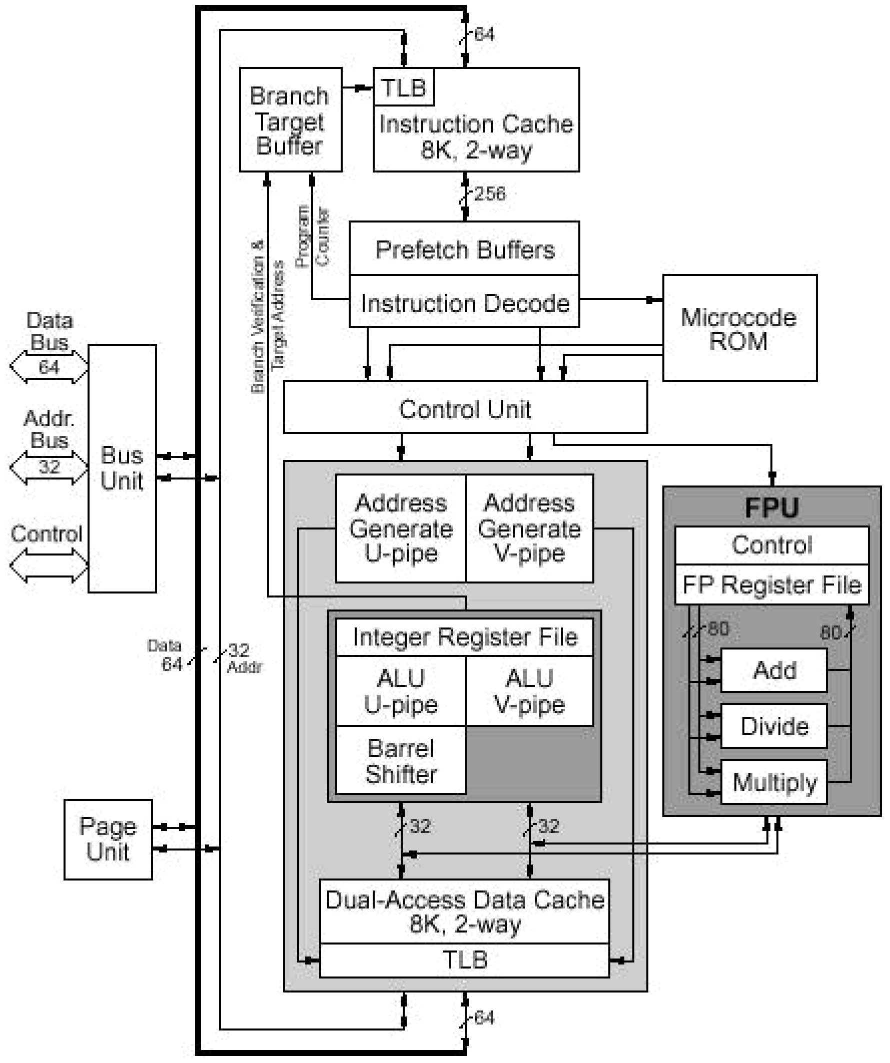
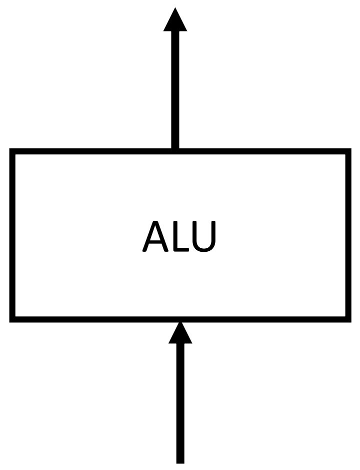


Figure 1. Pentium block diagram.

Hardware Has Long Exploited Parallelism

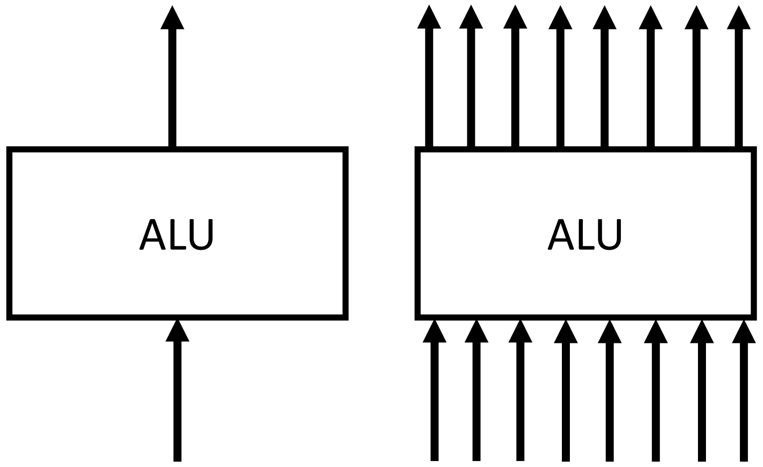
Bit-serial ALUs



Hardware Has Long Exploited Parallelism

Bit-serial ALUs

Bit-parallel ALUs

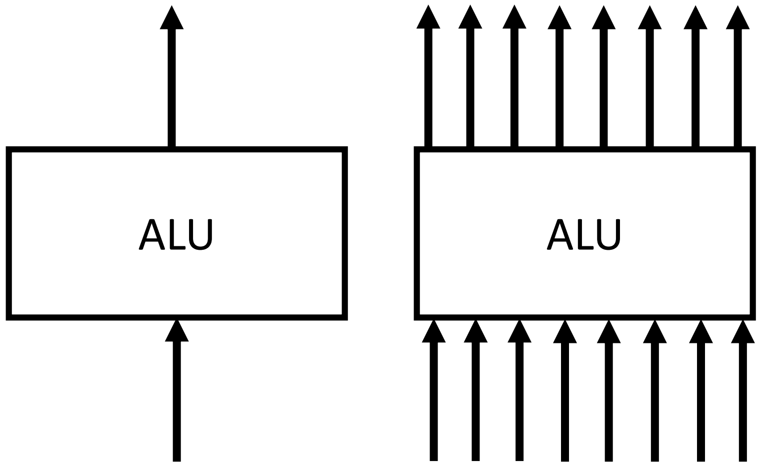


Hardware Has Long Exploited Parallelism

Bit-serial ALUs

Bit-parallel ALUs

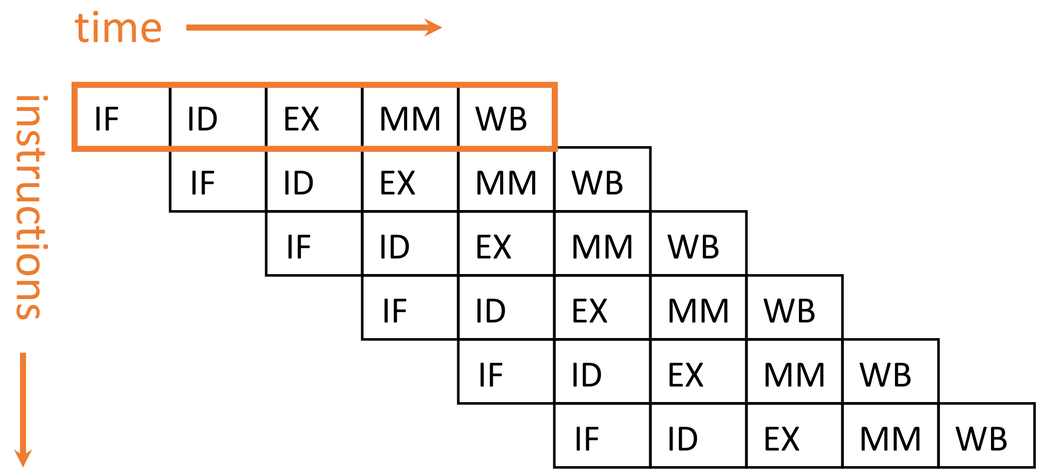
Pipelined microprocessors



Pipelined Microprocessors

Divide an instruction into stages

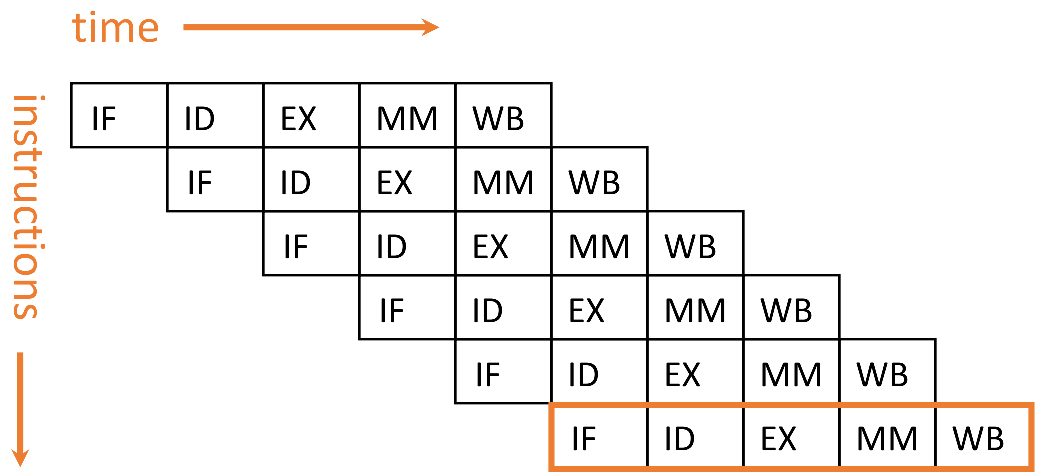
Stage 1		Stage 2		Stage 3		Stage 4		Stage 5
Instruction Fetch	⇒	Instruction Decode & Register Fetch	⇒	Execute	⇒	Memory Access	⇒	Register Write-back
IF	⇒	ID/RF	⇒	EX	⇒	MEM	⇒	WB



Pipelined Microprocessors

Divide an instruction into stages

Stage 1		Stage 2		Stage 3		Stage 4		Stage 5
Instruction Fetch	⇒	Instruction Decode & Register Fetch	⇒	Execute	⇒	Memory Access	⇒	Register Write-back
IF	⇒	ID/RF	⇒	EX	⇒	MEM	⇒	WB



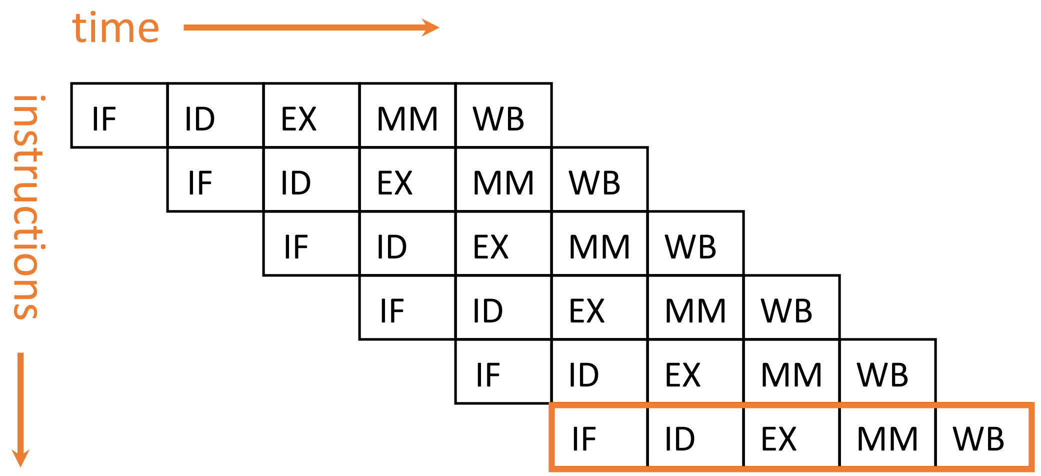
Pipelined Microprocessors

Divide an instruction into stages

Stage 1		Stage 2		Stage 3		Stage 4		Stage 5
Instruction Fetch	⇒	Instruction Decode & Register Fetch	⇒	Execute	⇒	Memory Access	⇒	Register Write-back
IF	⇒	ID/RF	⇒	EX	⇒	MEM	⇒	WB

A form of task parallelism

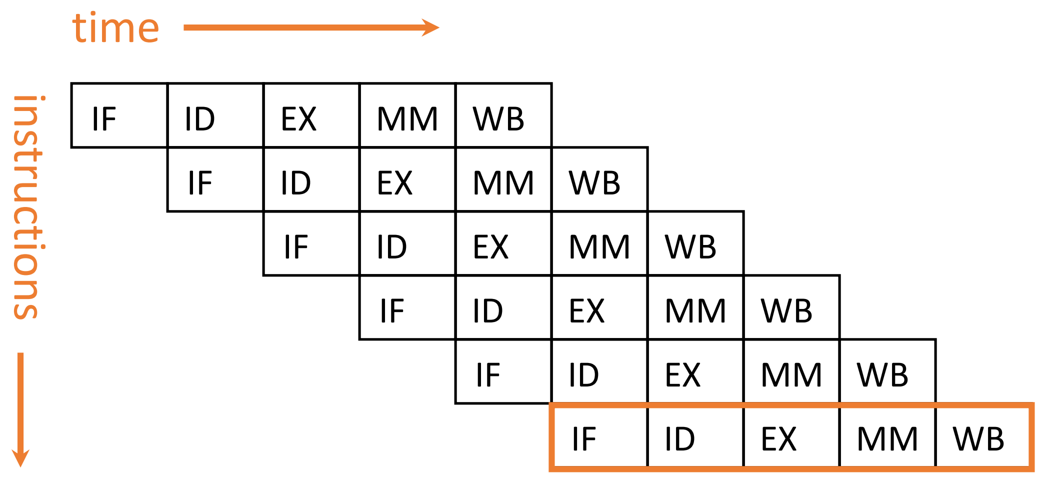
Increases latency of a single instruction



Pipelined Microprocessors

Divide an instruction into stages

Stage 1		Stage 2		Stage 3		Stage 4		Stage 5
Instruction Fetch	⇒	Instruction Decode & Register Fetch	⇒	Execute	⇒	Memory Access	⇒	Register Write-back
IF	⇒	ID/RF	⇒	EX	⇒	MEM	⇒	WB



A form of task parallelism

Increases latency of a single instruction

Increases throughput– ideally completes one instruction per cycle

Hardware Has Long Exploited Parallelism

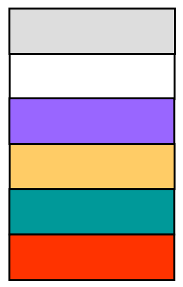
Bit-serial ALUs

Bit-parallel ALUs

Pipelined microprocessors

Out-of-order execution

Program order



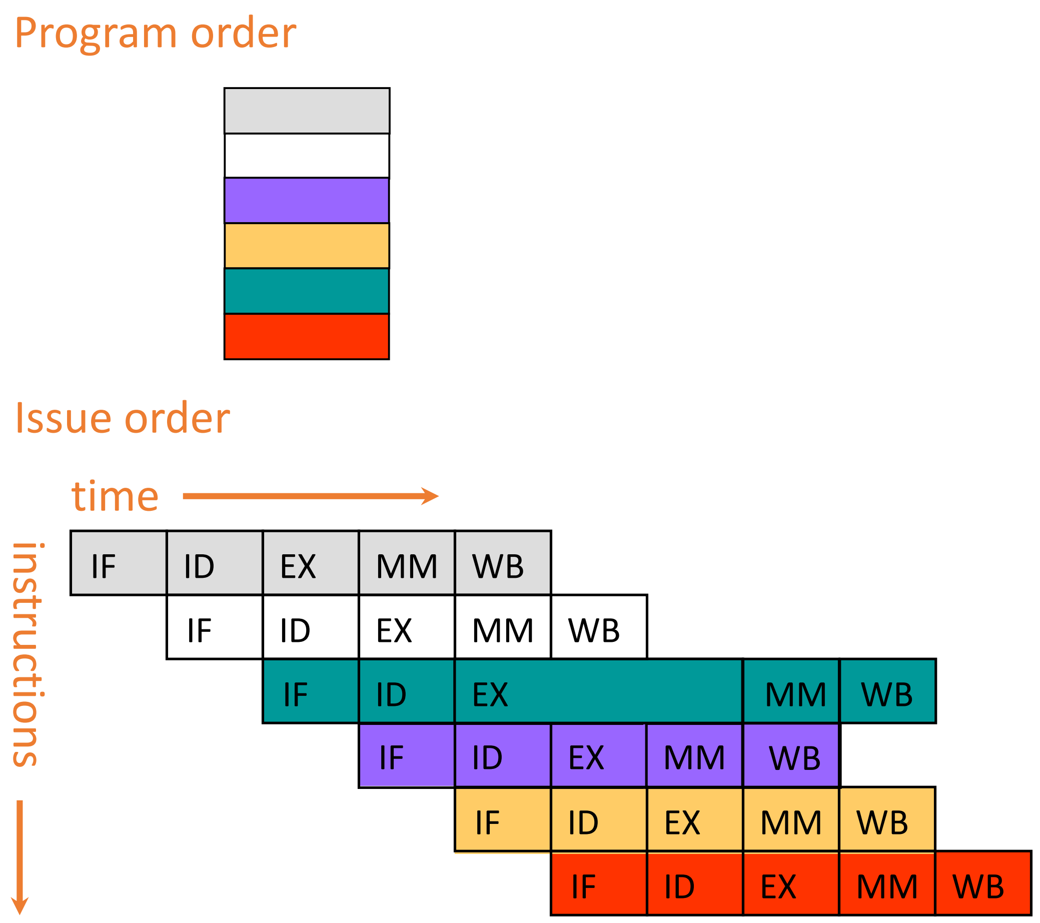
Hardware Has Long Exploited Parallelism

Bit-serial ALUs

Bit-parallel ALUs

Pipelined microprocessors

Out-of-order execution



Hardware Has Long Exploited Parallelism

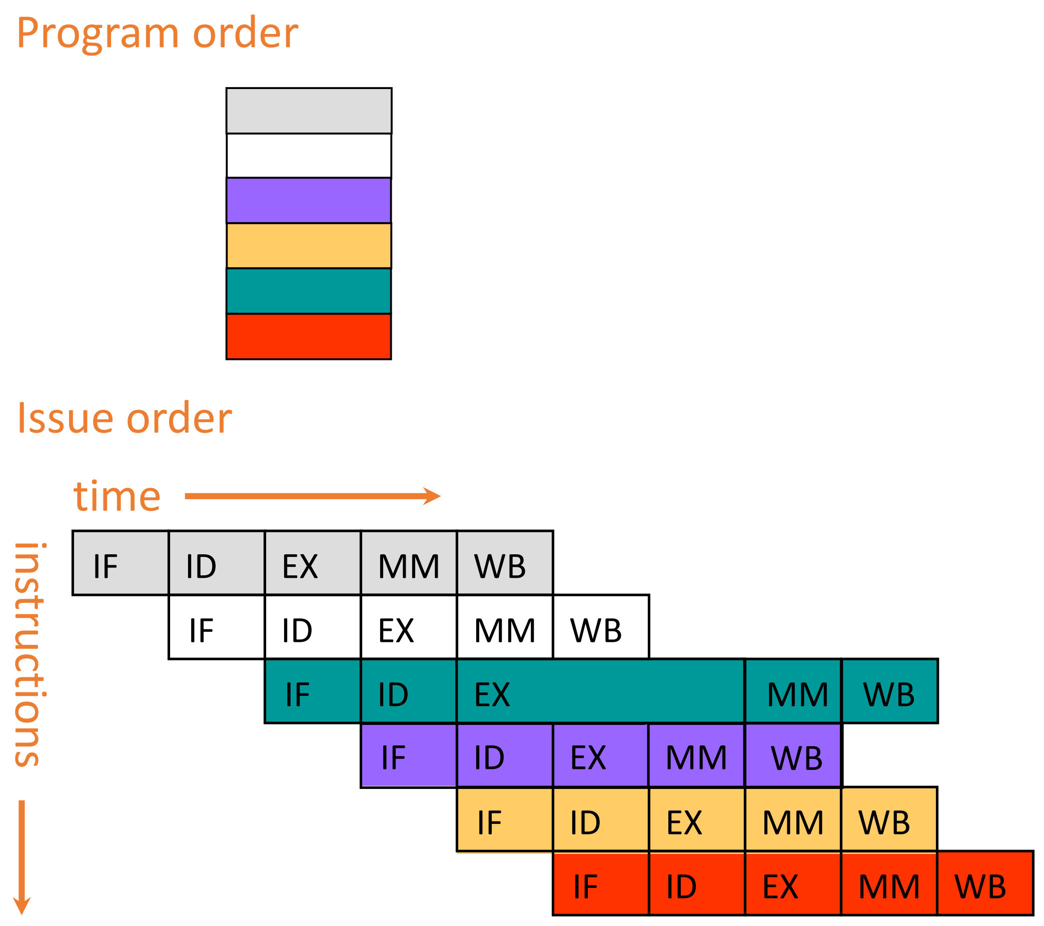
Bit-serial ALUs

Bit-parallel ALUs

Pipelined microprocessors

Out-of-order execution

How does out-of-order execution help?



Hardware Has Long Exploited Parallelism

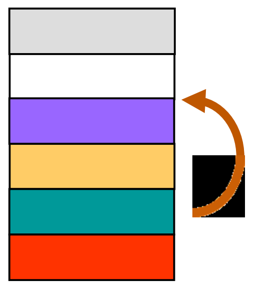
Bit-serial ALUs

Bit-parallel ALUs

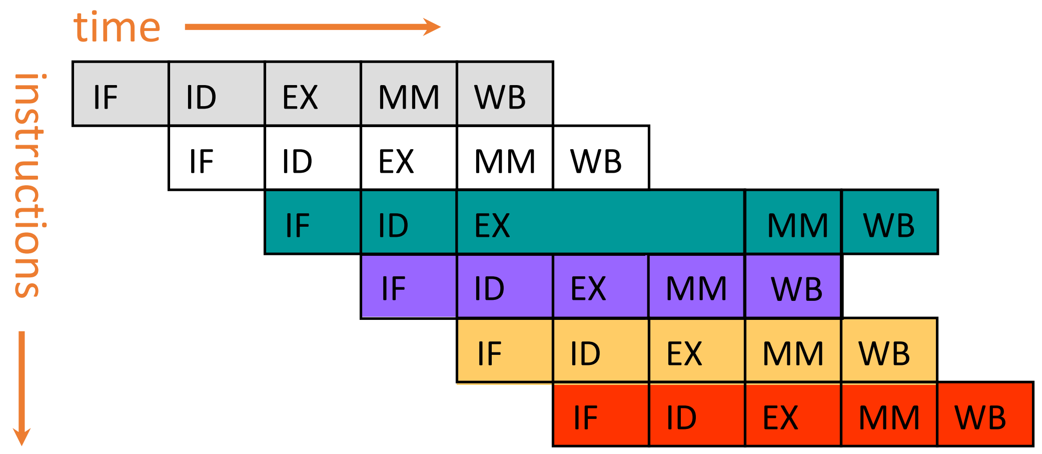
Pipelined microprocessors

Out-of-order execution

Program order



Issue order



How does out-of-order execution help?
Initiate a slow instruction early

Hardware Has Long Exploited Parallelism

Bit-serial ALUs

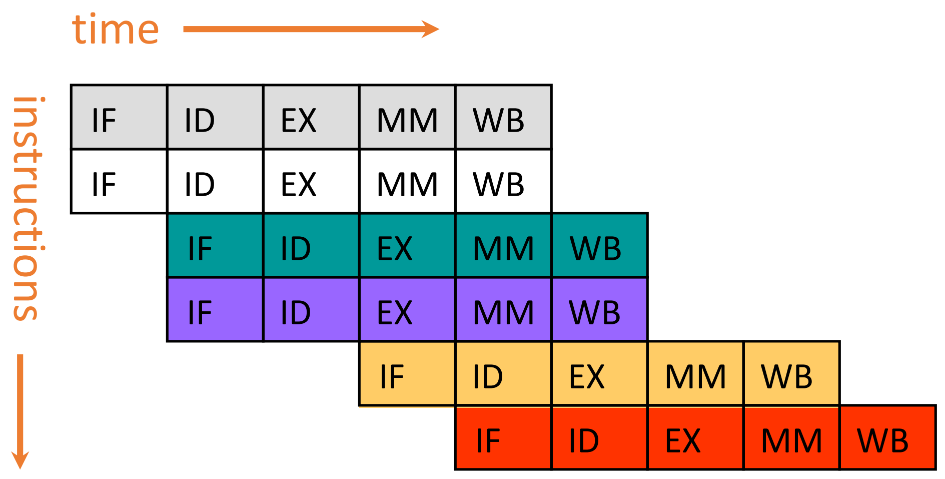
Bit-parallel ALUs

Pipelined microprocessors

Out-of-order execution

Superscalar execution

Issue multiple instructions per cycle



Hardware Has Long Exploited Parallelism

Bit-serial ALUs

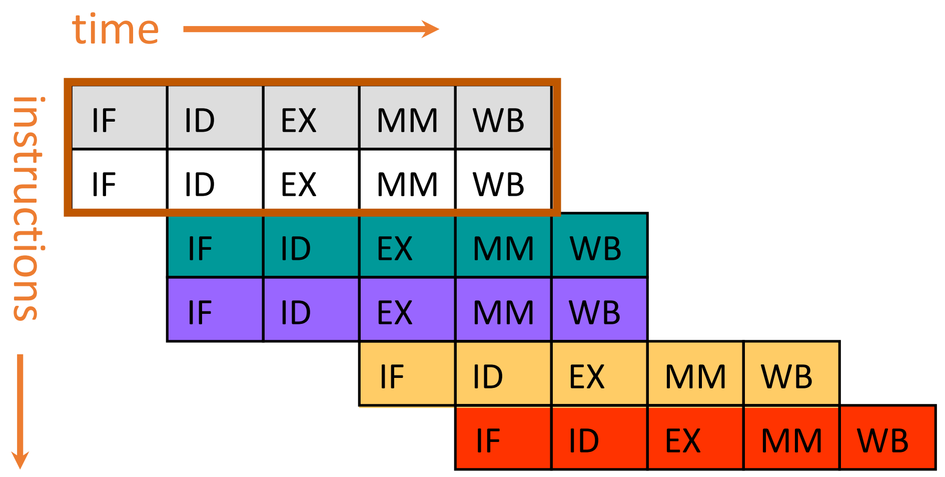
Bit-parallel ALUs

Pipelined microprocessors

Out-of-order execution

Superscalar execution

Issue multiple instructions per cycle



Hardware Has Long Exploited Parallelism

Bit-serial ALUs

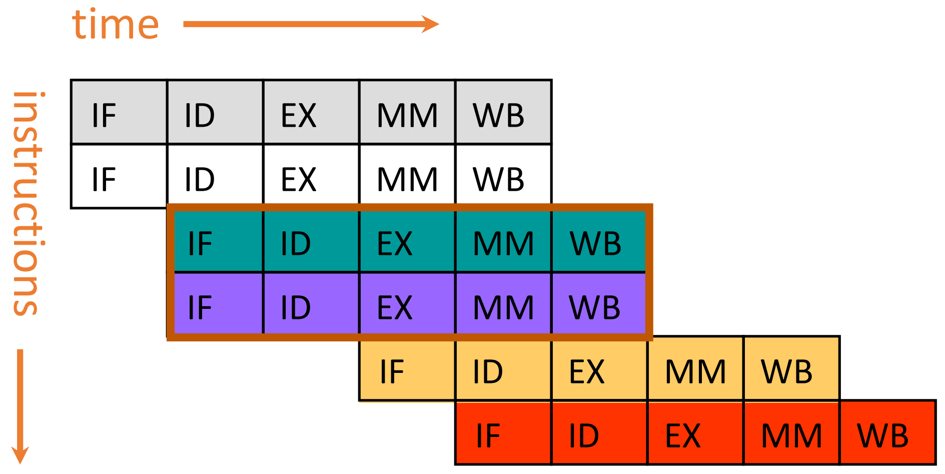
Bit-parallel ALUs

Pipelined microprocessors

Out-of-order execution

Superscalar execution

Issue multiple instructions per cycle



Hardware Has Long Exploited Parallelism

Bit-serial ALUs

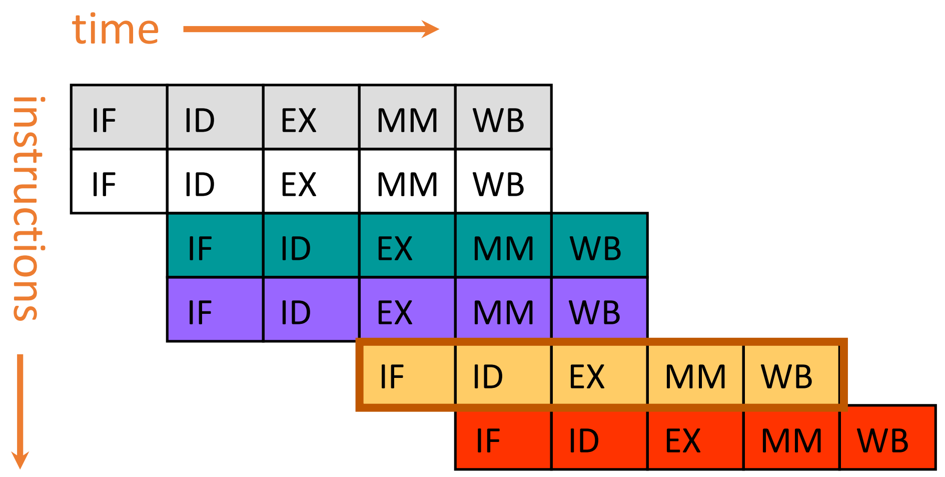
Bit-parallel ALUs

Pipelined microprocessors

Out-of-order execution

Superscalar execution

Issue multiple instructions per cycle



Hardware Has Long Exploited Parallelism

Bit-serial ALUs

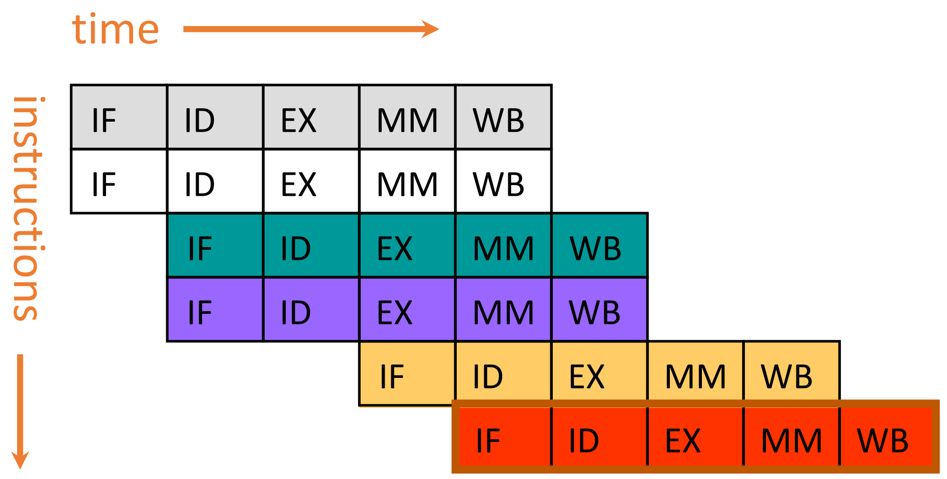
Bit-parallel ALUs

Pipelined microprocessors

Out-of-order execution

Superscalar execution

Issue multiple instructions per cycle



Hardware Has Long Exploited Parallelism

Bit-serial ALUs

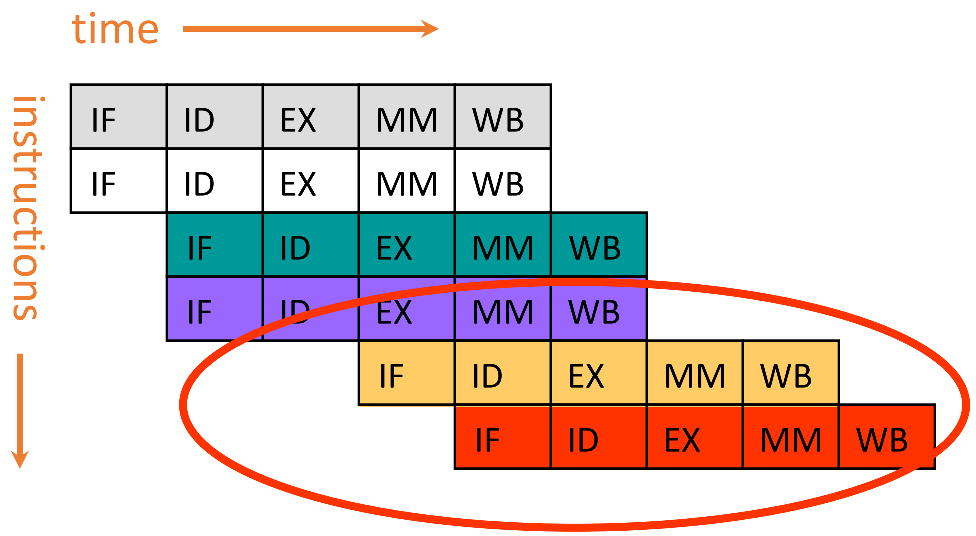
Bit-parallel ALUs

Pipelined microprocessors

Out-of-order execution

Superscalar execution

Issue multiple instructions per cycle



Can’t always exploit full issue width

Hardware Has Long Exploited Parallelism

Bit-serial ALUs

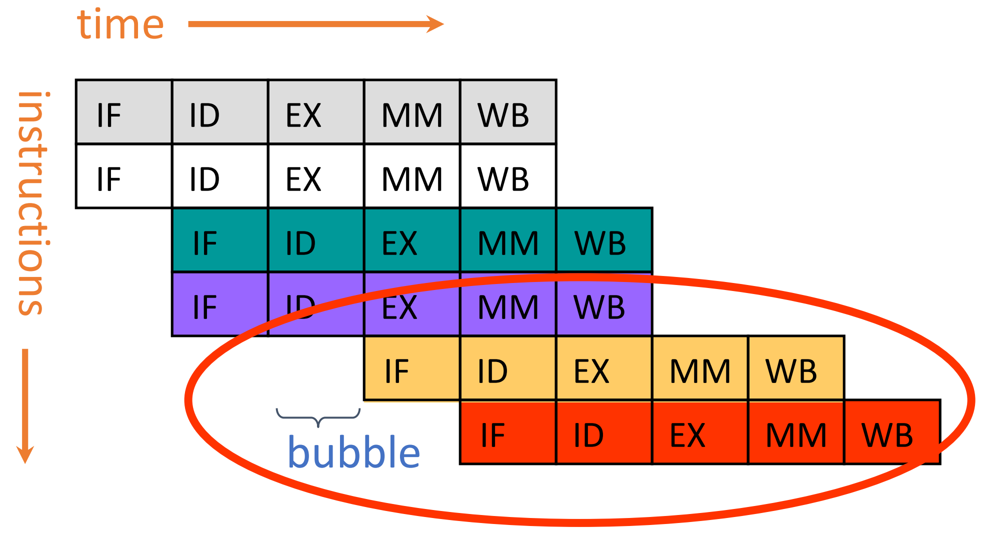
Bit-parallel ALUs

Pipelined microprocessors

Out-of-order execution

Superscalar execution

Issue multiple instructions per cycle



Can’t always exploit full issue width

Hardware Has Long Exploited Parallelism

Bit-serial ALUs

Bit-parallel ALUs

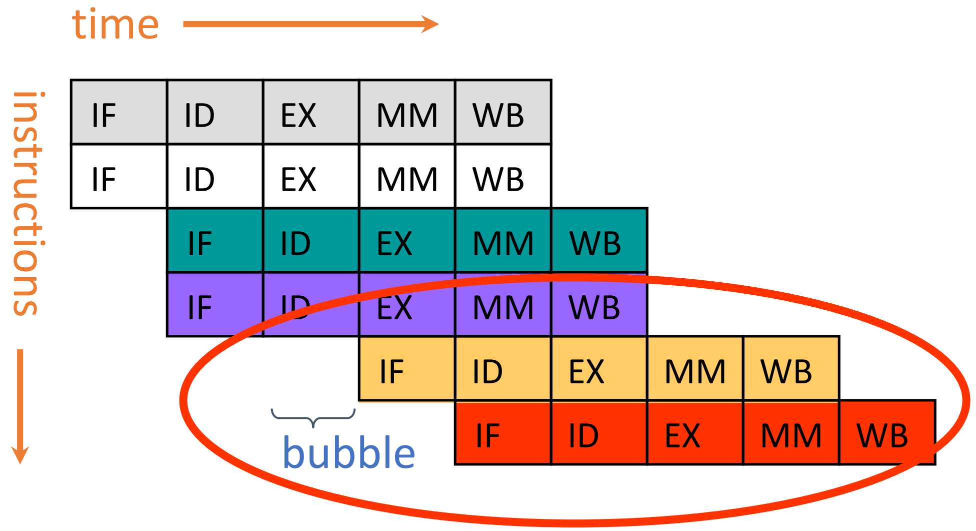
Pipelined microprocessors

Out-of-order execution

Superscalar execution

Multithreading

Issue multiple instructions per cycle

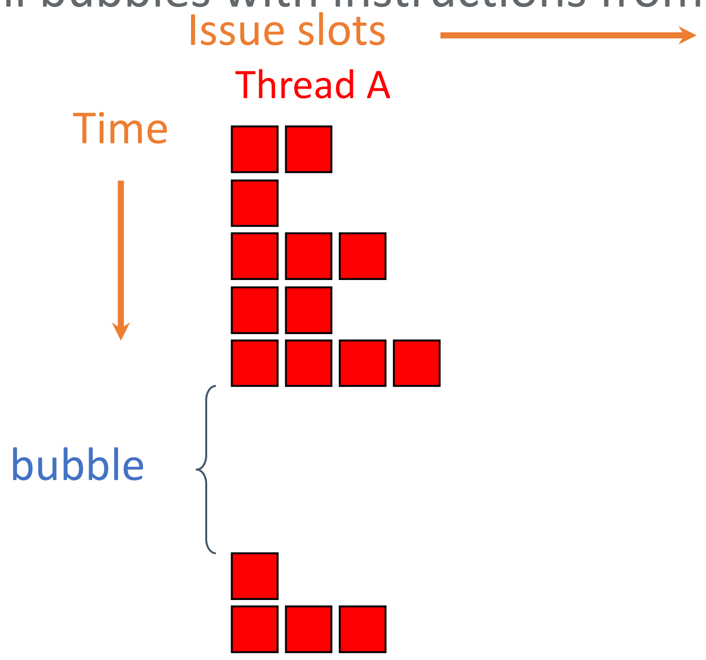


Can’t always exploit full issue width

Multithreading

Bubbles in the pipeline represent performance loss

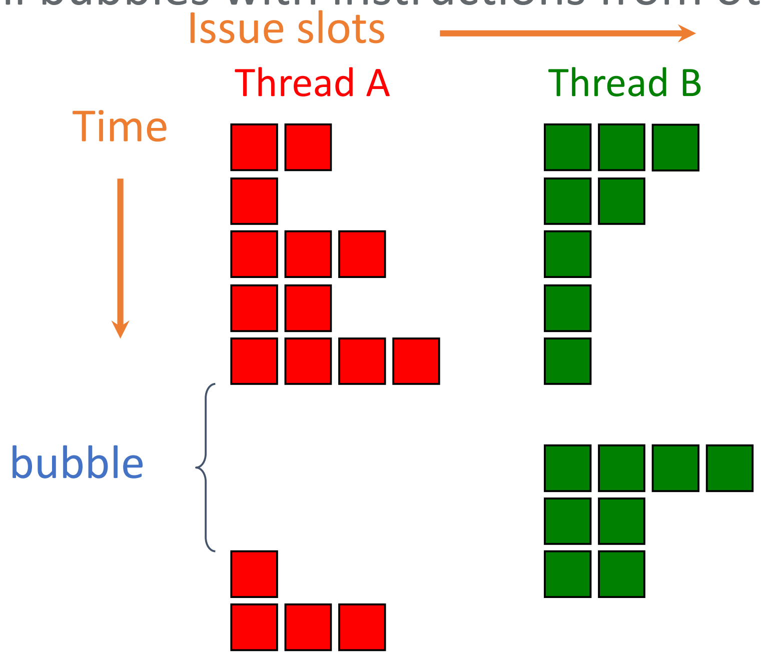
Fill bubbles with instructions from other threads



Multithreading

Bubbles in the pipeline represent performance loss

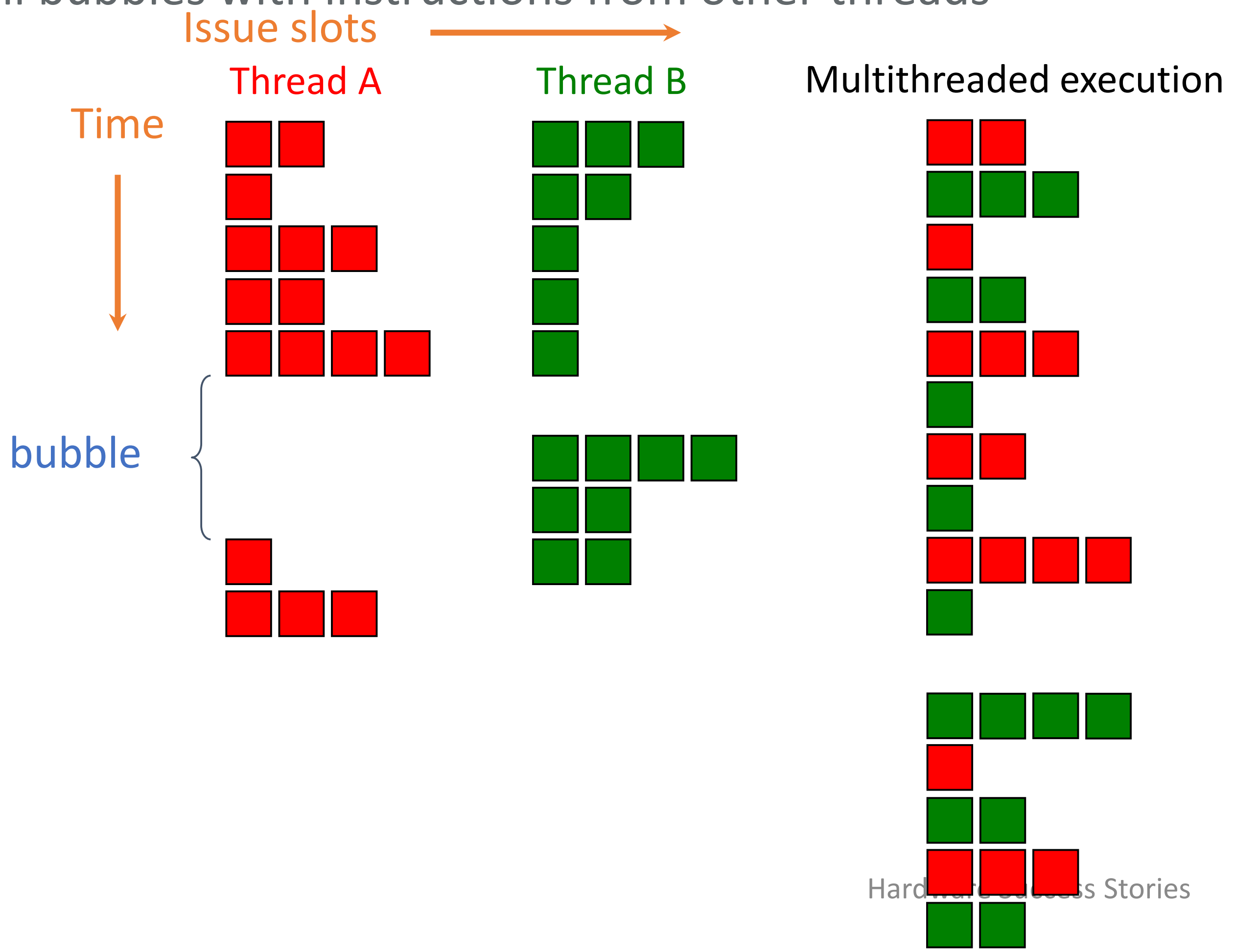
Fill bubbles with instructions from other threads



Multithreading

Bubbles in the pipeline represent performance loss

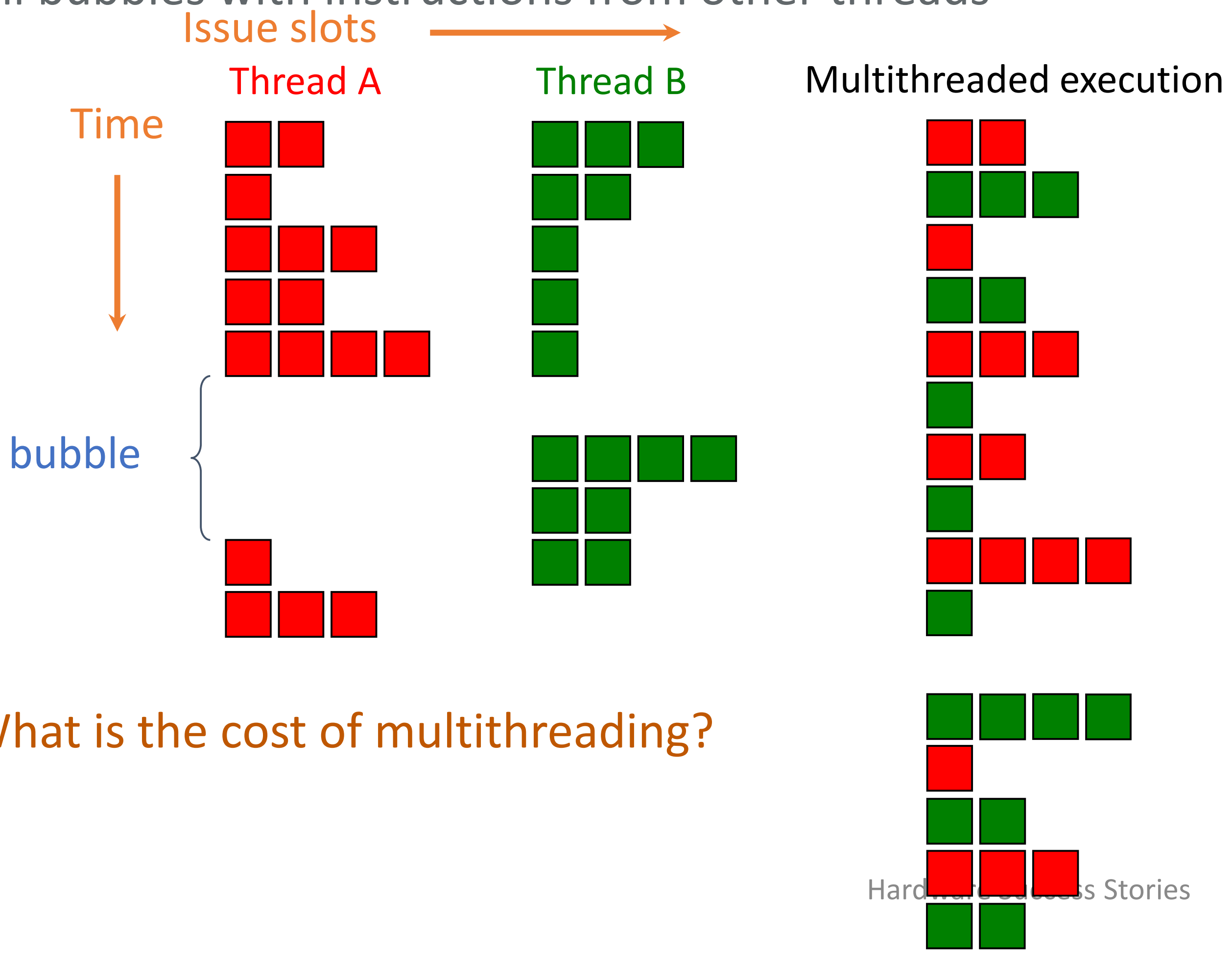
Fill bubbles with instructions from other threads



Multithreading

Bubbles in the pipeline represent performance loss

Fill bubbles with instructions from other threads

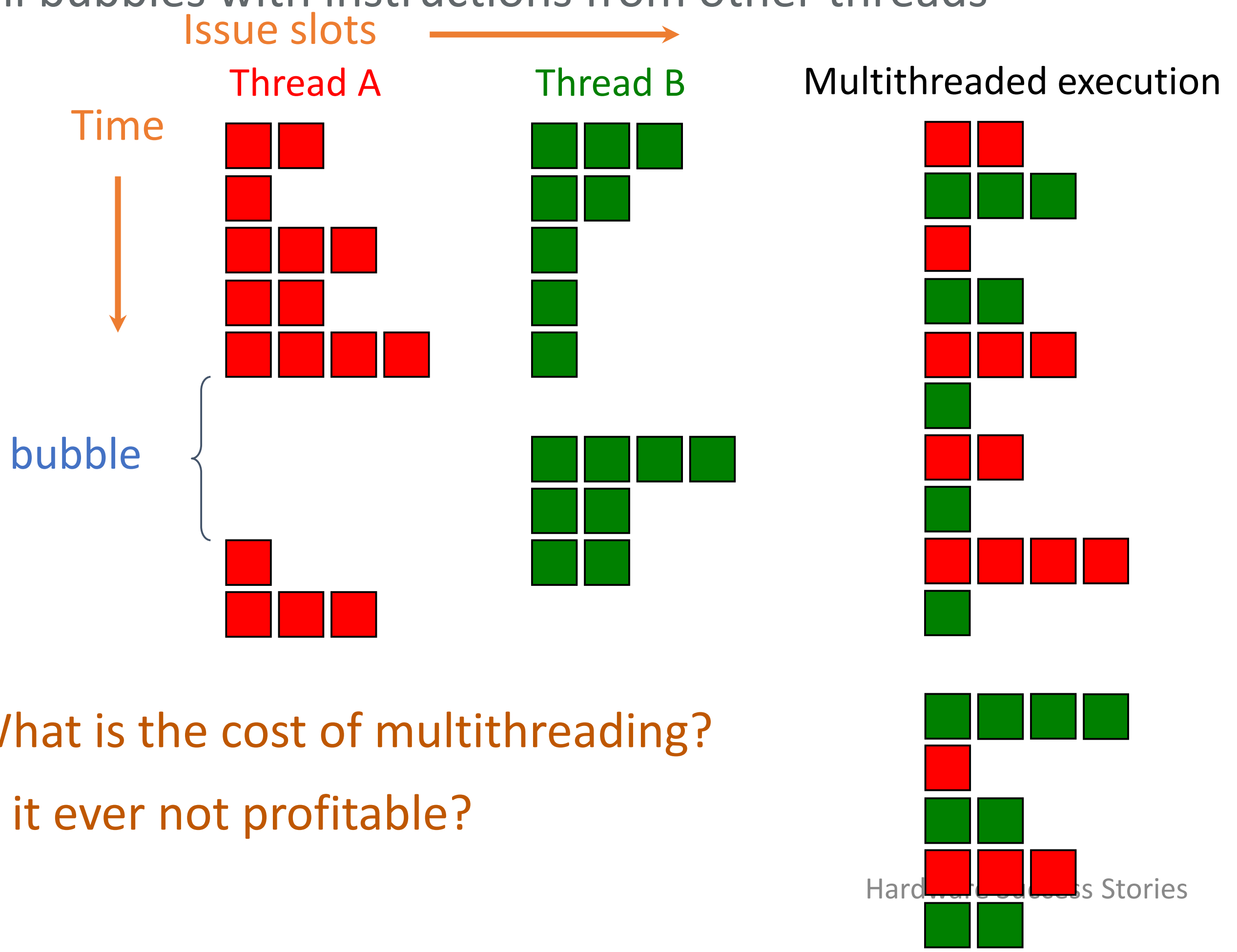


What is the cost of multithreading?

Multithreading

Bubbles in the pipeline represent performance loss

Fill bubbles with instructions from other threads



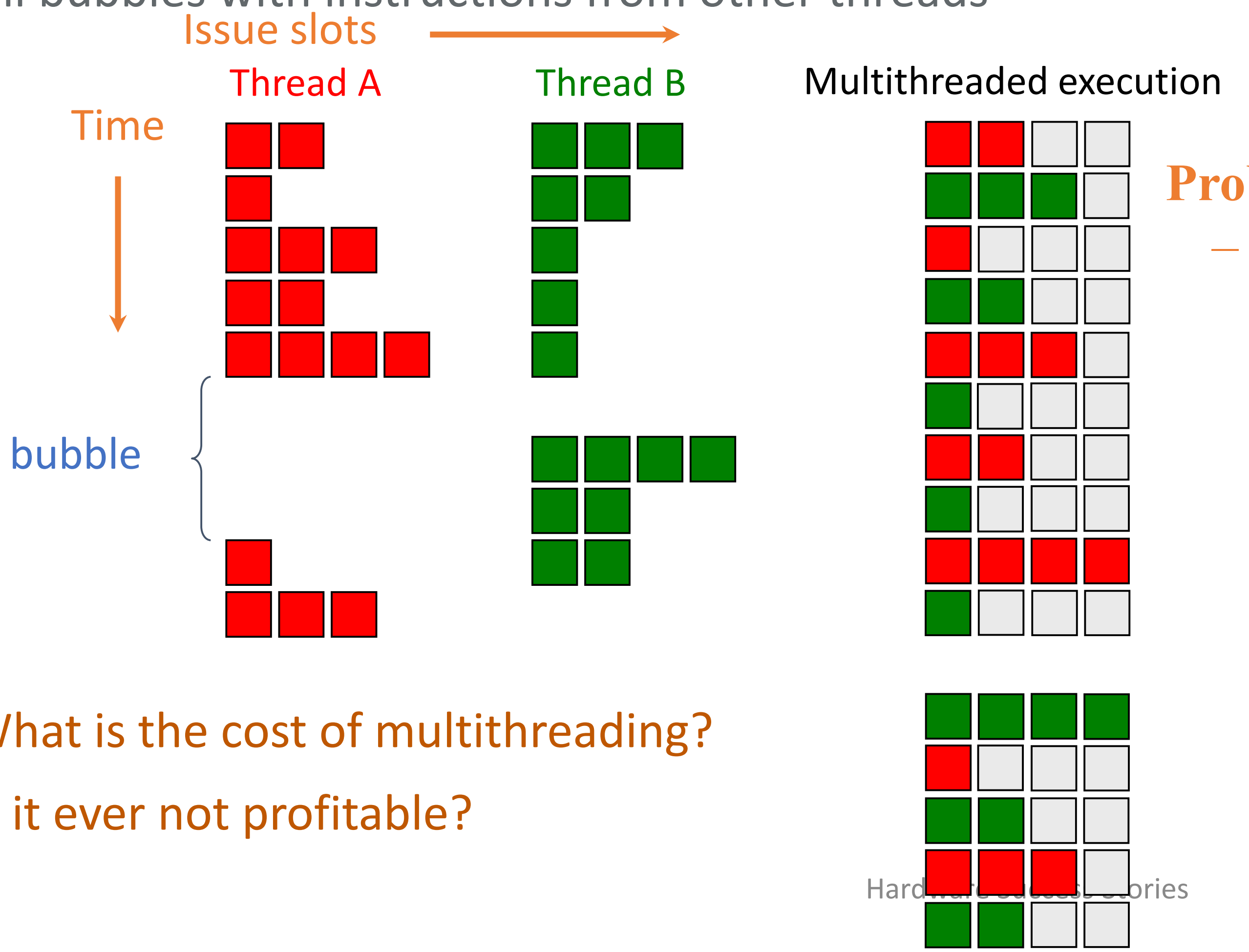
What is the cost of multithreading?

Is it ever not profitable?

Multithreading

Bubbles in the pipeline represent performance loss

Fill bubbles with instructions from other threads



Problem

- Still have lots of unfilled issue slots

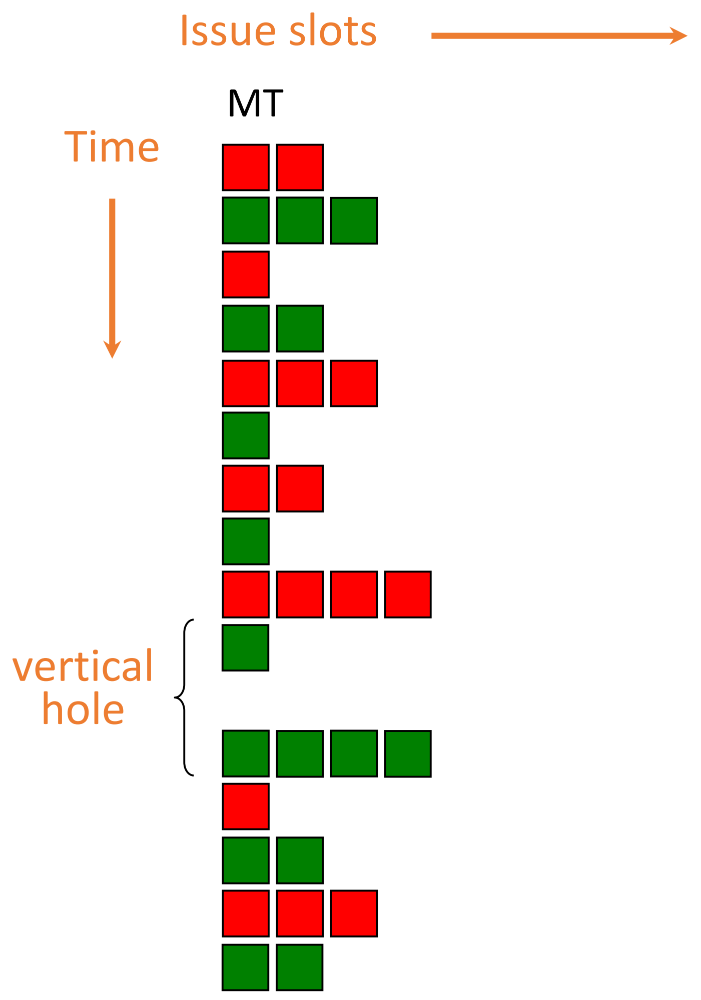
What is the cost of multithreading?

Is it ever not profitable?

Improving Hardware Utilization

Limitation of Multithreading

- At each cycle, issue instructions from any one thread



Improving Hardware Utilization

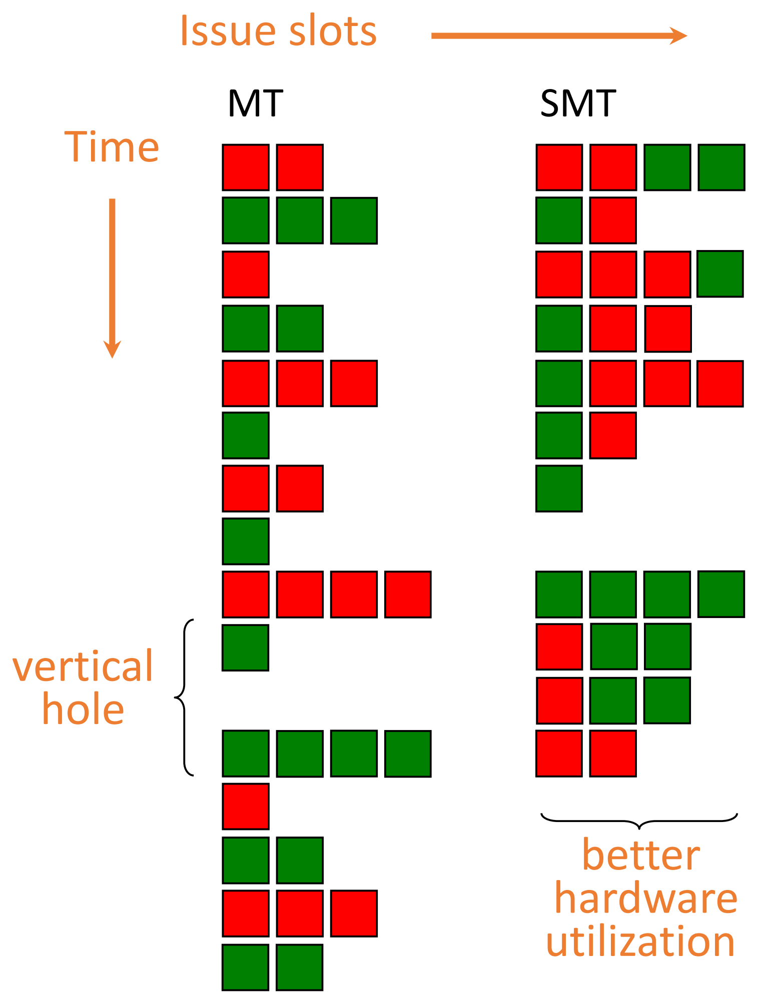
Limitation of Multithreading

At each cycle, issue instructions from any one thread

Simultaneous Multithreading (SMT)

[Tulsen, Eggers, Levy, ISCA95]

At each cycle, issue instructions from multiple threads



Improving Hardware Utilization

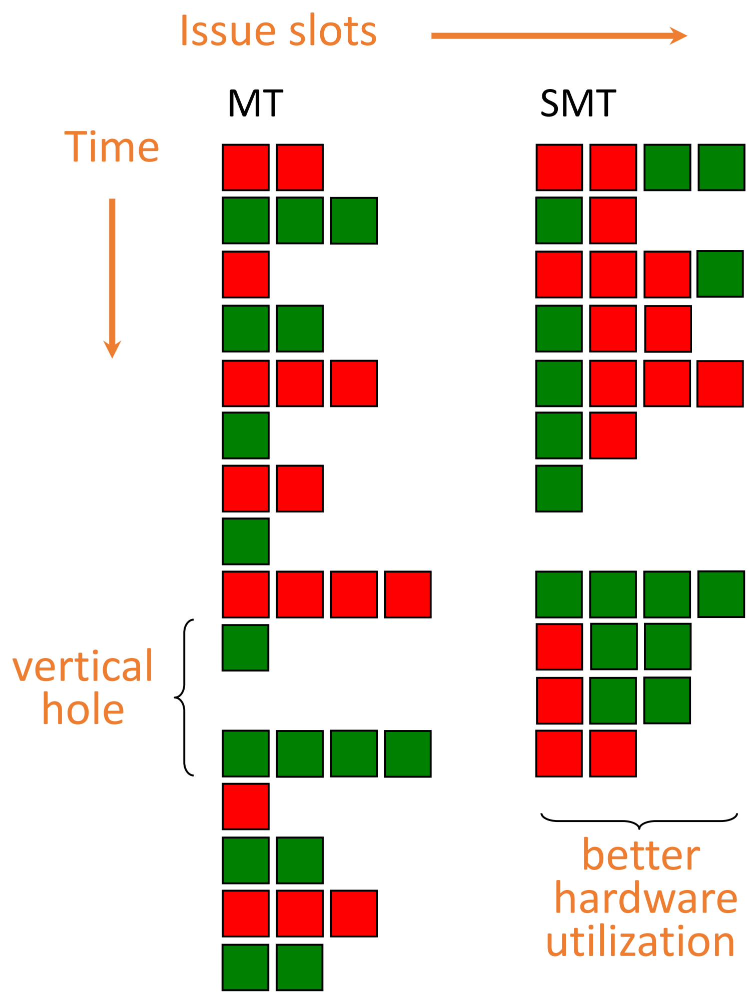
Limitation of Multithreading

- At each cycle, issue instructions from any one thread

Simultaneous Multithreading (SMT)

- [Tulsen, Eggers, Levy, ISCA95]
- At each cycle, issue instructions from multiple threads

What is the cost of SMT?



Hardware Has Long Exploited Parallelism

Bit-serial ALUs

Bit-parallel ALUs

Pipelined microprocessors

Out-of-order execution

Superscalar execution

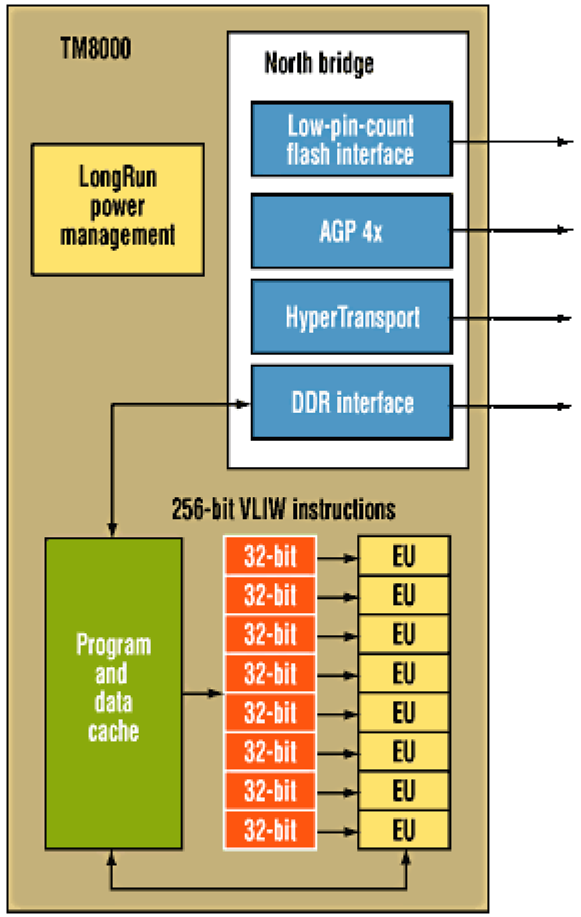
Multithreading

SMT

VLIW

VLIW

- Very Long Instruction Word
  - Wide instructions for controlling multiple functional units
  - ISA explicitly encodes parallelism
  - Statically scheduled
    - Reduces power by devoting less hardware to control



Transmeta TM8000 Core

Vector Processors

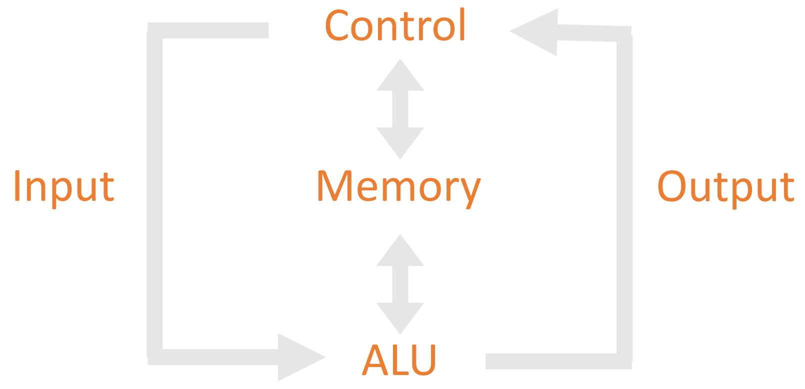
Goal:

- Reduce instruction fetch bandwidth
- Apply instructions to all elements of vectors simultaneously
  - eg. **`VectorAdd`**

Vector Processors

Goal:

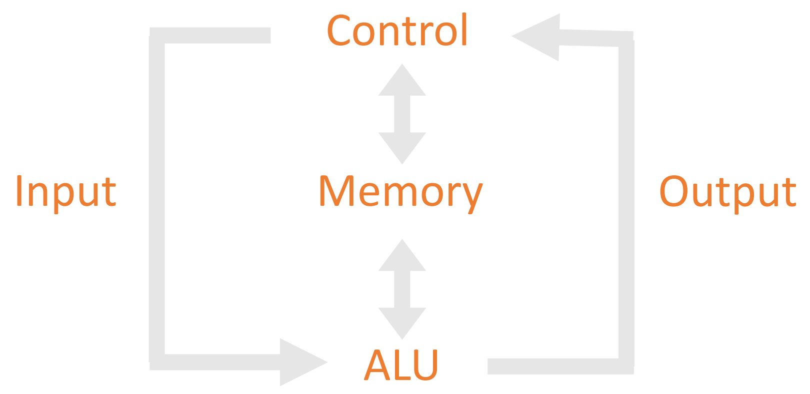
- Reduce instruction fetch bandwidth
- Apply instructions to all elements of vectors simultaneously
  - eg. `VectorAdd`
- Alleviates the von Neumann Bottleneck:
  - Instruction fetch and data fetch contend for memory



Vector Processors

Goal:

- Reduce instruction fetch bandwidth
- Apply instructions to all elements of vectors simultaneously
  - eg. `VectorAdd`
- Alleviates the von Neumann Bottleneck:
  - Instruction fetch and data fetch contend for memory



- Have compilers convert existing programs to vector programs

Vector Processors: A Success Story?

Vectorizing compilers have been quite successful

Vector Processors: A Success Story?

Vectorizing compilers have been quite successful

But initially, many programs could not be vectorized

Vector Processors: A Success Story?

Vectorizing compilers have been quite successful

But initially, many programs could not be vectorized

Did compilers get better?

Vector Processors: A Success Story?

Vectorizing compilers have been quite successful

But initially, many programs could not be vectorized

Did compilers get better?

No. Through compiler feedback and training, programmers learned how to write code that could be vectorized

Vector Processors: A Success Story?

Vectorizing compilers have been quite successful

But initially, many programs could not be vectorized

Did compilers get better?

No. Through compiler feedback and training, programmers learned how to write code that could be vectorized

Can we follow a similar path for parallelizing compilers?

Vector Processors: A Success Story?

Vectorizing compilers have been quite successful

But initially, many programs could not be vectorized

Did compilers get better?

No. Through compiler feedback and training, programmers

learned how to write code that could be vectorized

Can we follow a similar path for parallelizing compilers?

Parallelization is more difficult.

The question is not just “is a loop parallelizable?” There are many more tradeoffs to consider.

The Big Picture

Bit-serial ALUs

Bit-parallel ALUs

Pipelined microprocessors

Out-of-order execution

Superscalar execution

Multithreading

SMT

VLIW

Vector processors

Multi-cores

Increasing granularity of parallelism

The Big Picture

Bit-serial ALUs

Bit-parallel ALUs

Pipelined microprocessors

Out-of-order execution

Superscalar execution

Multithreading

SMT

VLIW

Vector processors

Multi-cores

Increasing granularity of parallelism

Which of these expose parallelism to through the ISA?

The Big Picture

Bit-serial ALUs

Bit-parallel ALUs

Pipelined microprocessors

Out-of-order execution

Superscalar execution

Multithreading

SMT

VLIW

Vector processors

Multi-cores

Increasing granularity of parallelism

Which of these expose parallelism to through the ISA?

The Big Picture

Bit-serial ALUs

Bit-parallel ALUs

Pipelined microprocessors

Out-of-order execution

Superscalar execution

Multithreading

SMT

VLIW

Vector processors

Multi-cores

Increasing granularity of parallelism

Which of these expose parallelism to through the ISA?

Conclusions

For years, the ISA was able to hide hardware parallelism, so programmers could just write sequential code

As the granularity of parallelism has increased, it’s become difficult to hide the parallelism

Why is the granularity increasing?

Conclusions

For years, the ISA was able to hide hardware parallelism, so programmers could just write sequential code

As the granularity of parallelism has increased, it’s become difficult to hide the parallelism

Why is the granularity increasing?

Is hidden complexity always good?