EE382V (17325): Principles in Computer Architecture
Parallelism and Locality
Fall 2007

Lecture 6 – Summary of HW Parallelism; SW Parallelism

Mattan Erez

The University of Texas at Austin

Outline

- Corrections/clarifications
- Pipelining
- Summary of parallel HW (multiple ALUs)
- Classification of ILP/DLP/TLP in software
- Patterns for parallel programming

Reminders

- This is not a microarchitecture class
  - We will be discussing microarch. of various stream processors though
    - Details deferred to later in the semester
- This class is not a replacement for Parallel Computer Architecture class
  - We only superficially cover many details of parallel architectures
  - Focus on parallelism and locality at the same time

Corrections/clarifications

- Intel μops vs. AMD R-ops
  - Intel μops are RISC-like LD/ST
    - μops can occasionally be fused to improve scheduling
  - AMD R-ops can have memory operands
    - Removed when actually issued to ALUs

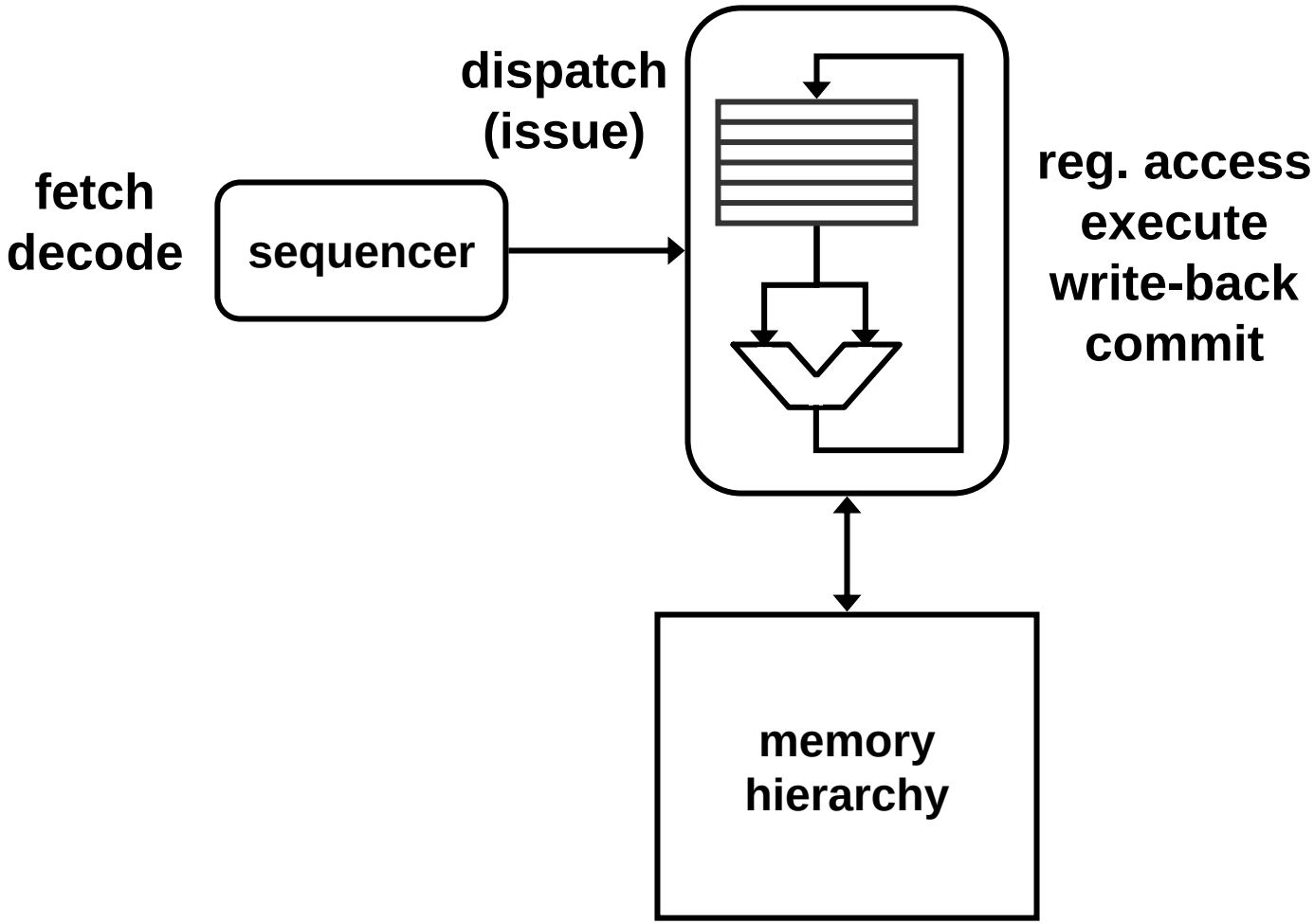
- SMT and TLS
  - TLS: convert ILP → TLP
  - SMT: convert TLP → ILP (in execution part of pipeline)

Evaluate DLP/ILP/TLP based on actual HW mechanisms (rather than names)

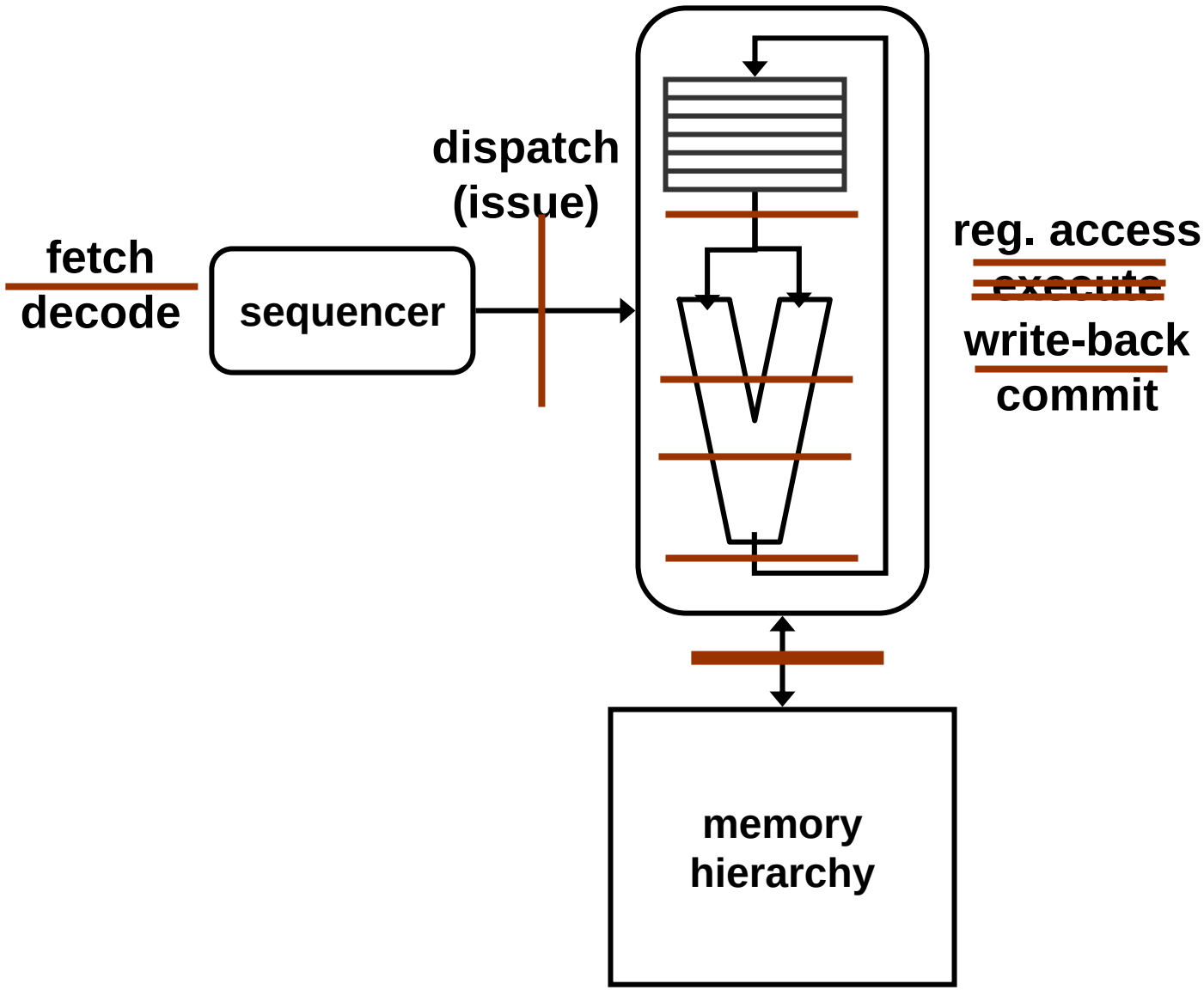
Outline

- Corrections/clarifications
- Pipelining
- Summary of parallel HW (multiple ALUs)
- Classification of ILP/DLP/TLP in software
- Patterns for parallel programming

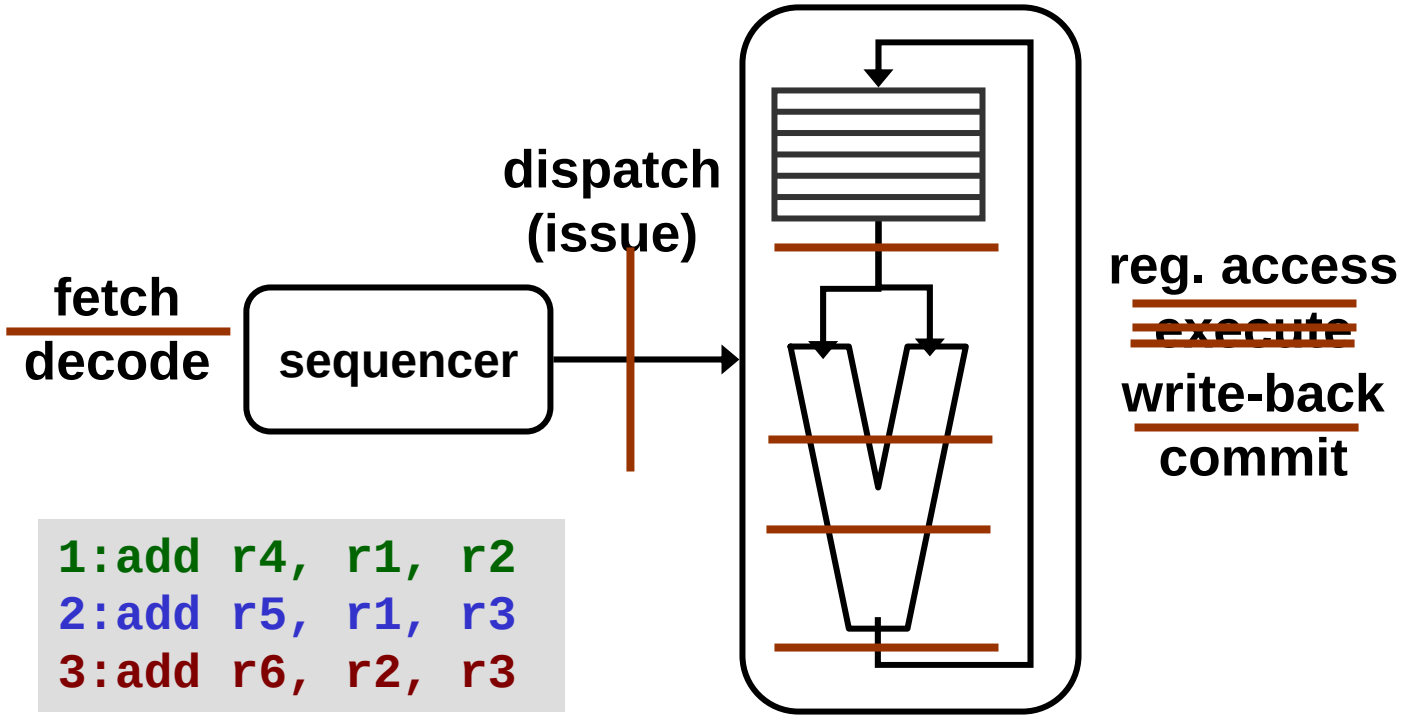
Simplified view of a processor



Simplified view of a pipelined processor



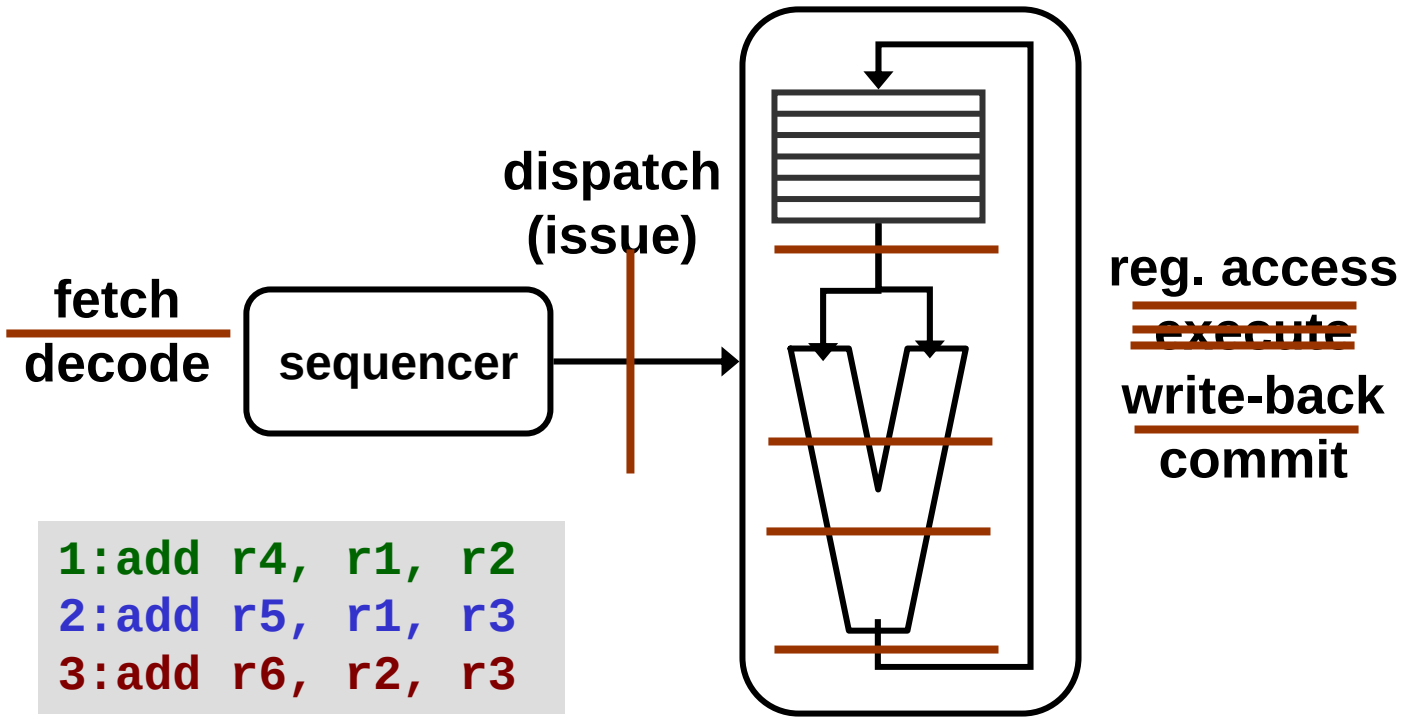
Simplified view of a pipelined processor




```
1:add r4, r1, r2
2:add r5, r1, r3
3:add r6, r2, r3
```

1	F	D	I	R	E	E	E	W	C

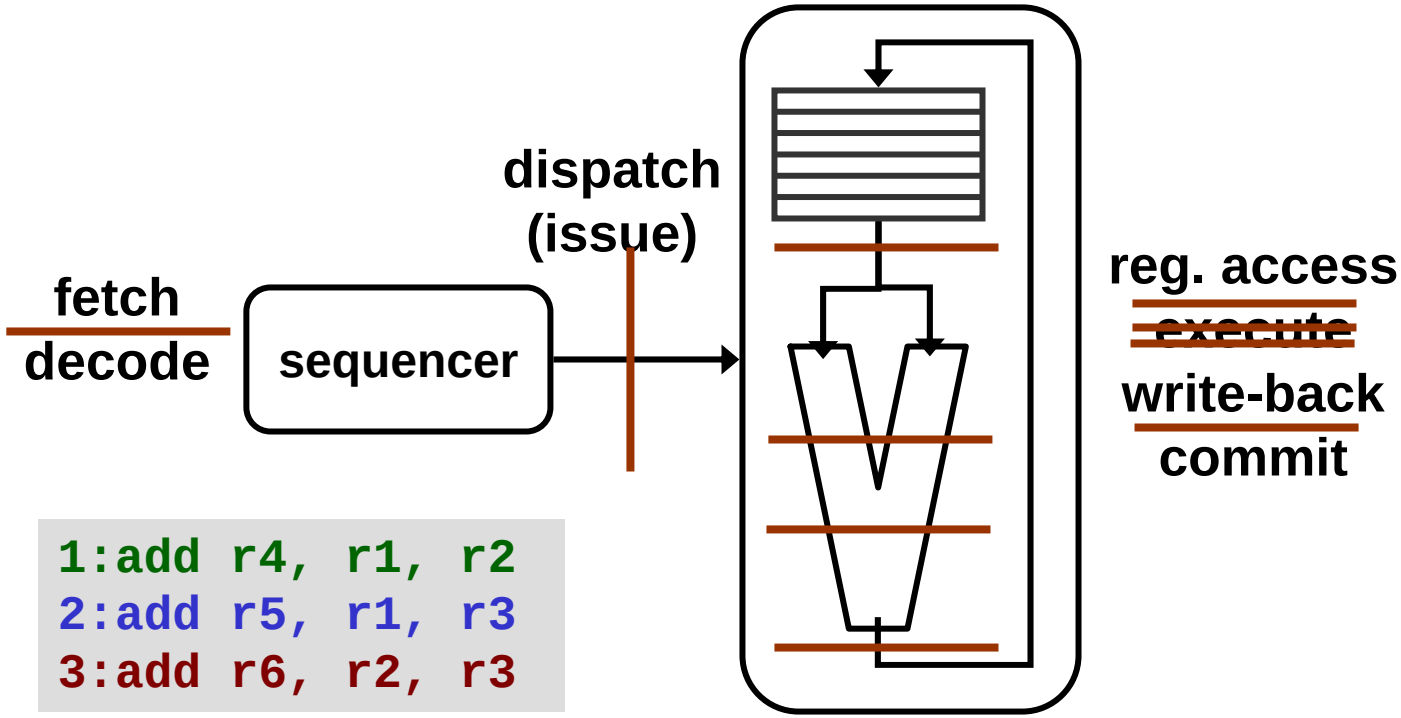
Simplified view of a pipelined processor




```
1:add r4, r1, r2
2:add r5, r1, r3
3:add r6, r2, r3
```

1	F	D	I	R	E	E	E	W	C	F	D	I	R	E	E	E	W	C	F	D	I	R	E	E

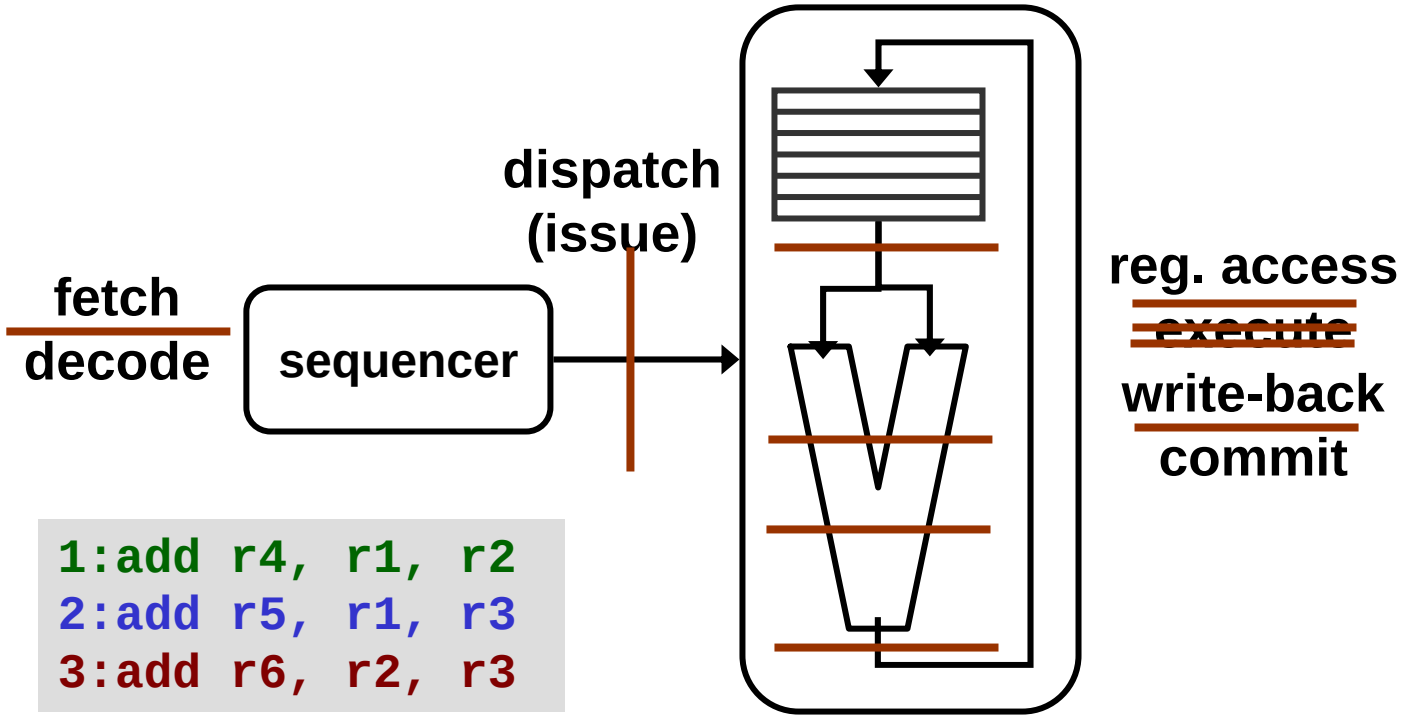
Simplified view of a pipelined processor




```
1:add r4, r1, r2
2:add r5, r1, r3
3:add r6, r2, r3
```

1	F	D	I	R	E	E	E	W	C													

Simplified view of a pipelined processor

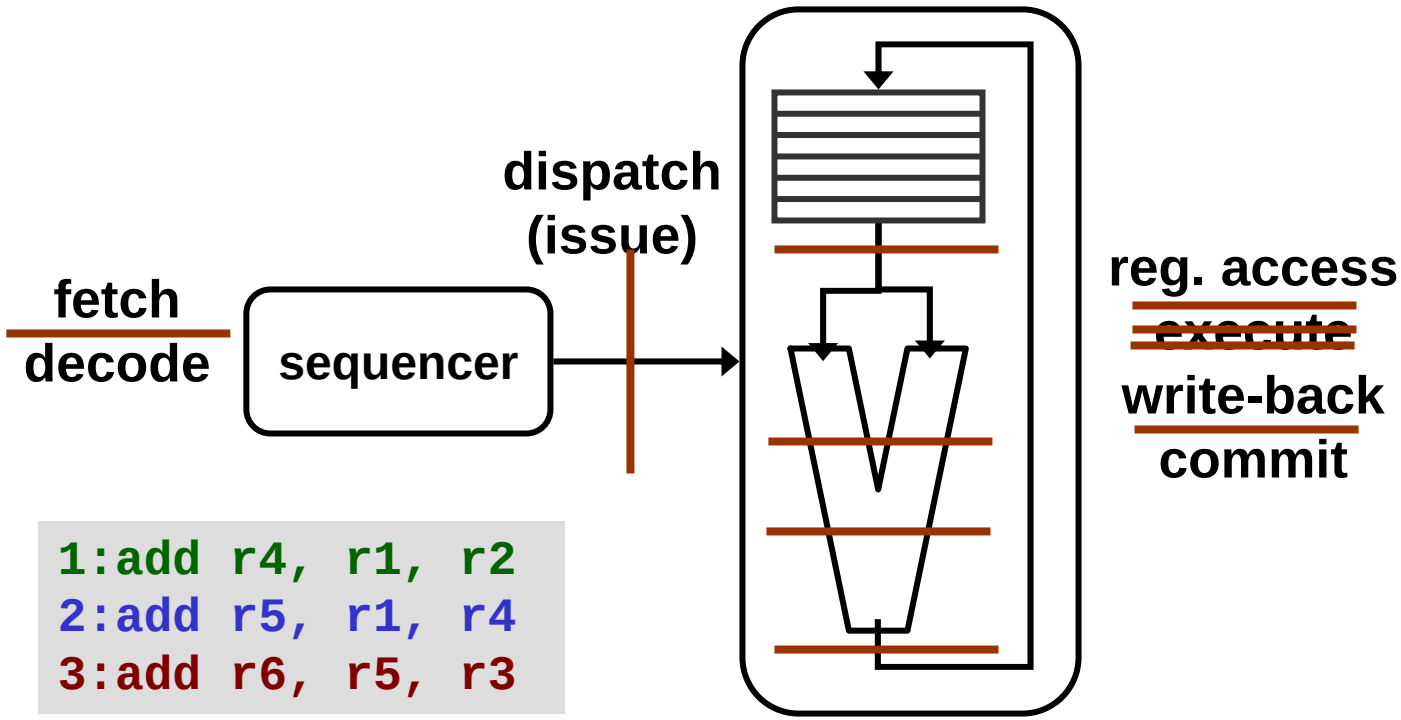



```
1:add r4, r1, r2
2:add r5, r1, r3
3:add r6, r2, r3
```

1	F	D	I	R	E	E	E	W	C														
2		F	D	I	R	E	E	E	W	C														
3			F	D	I	R	E	E	E	W	C													

What are the parallel resources?

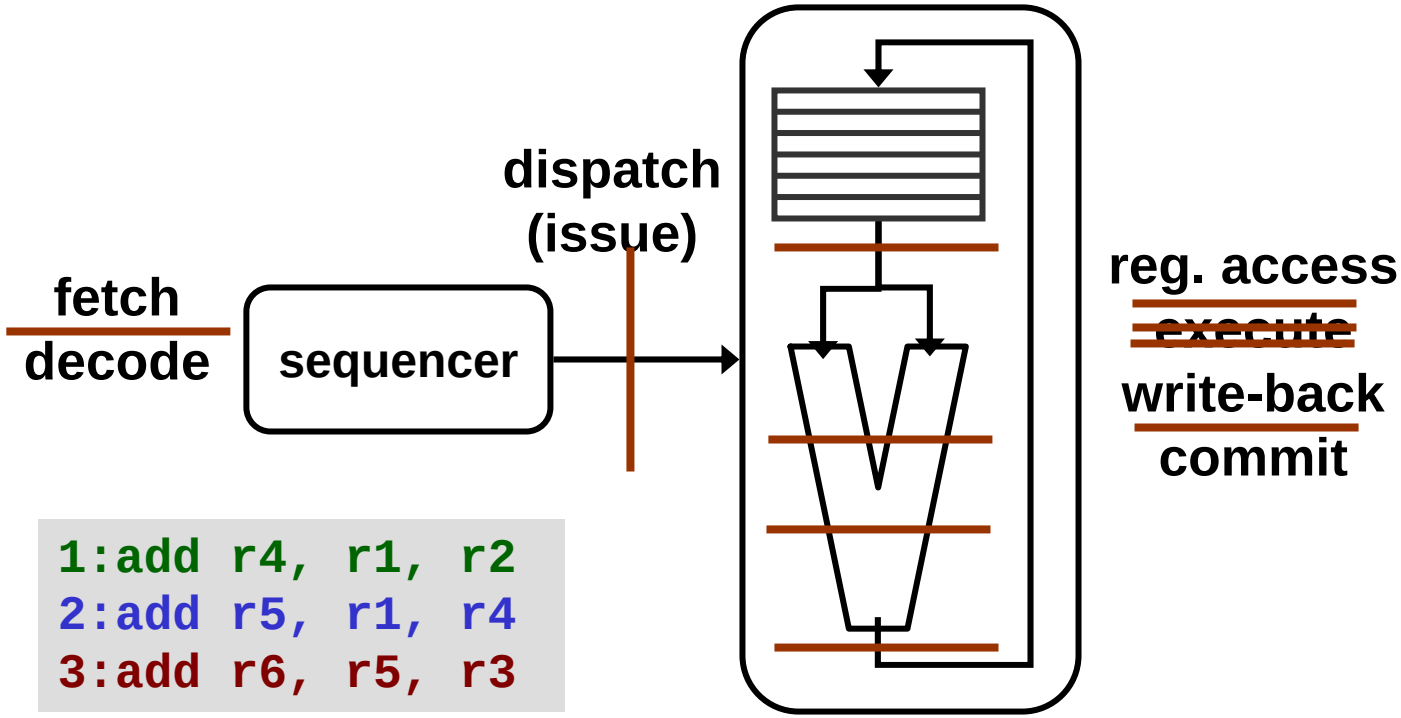
Simplified view of a pipelined processor




```
1:add r4, r1, r2
2:add r5, r1, r4
3:add r6, r5, r3
```

1	F	D	I	R	E	E	E	W	C													

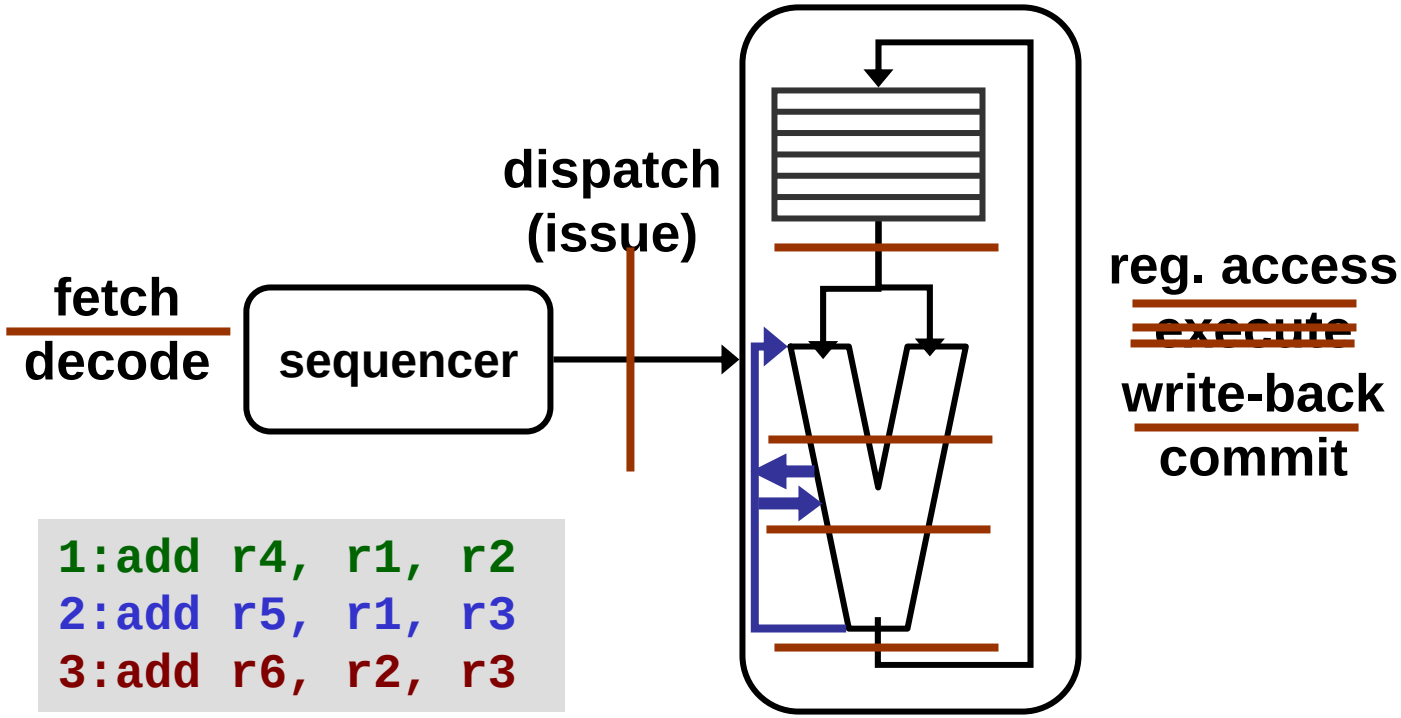
Simplified view of a pipelined processor




```
1:add r4, r1, r2
2:add r5, r1, r4
3:add r6, r5, r3
```

1	F	D	I	R	E	E	E	W	C														
2		F	D	I					R	E	E	E	W	C									
3			F	D	I									R	E	E	E	W	C				

Simplified view of a pipelined processor

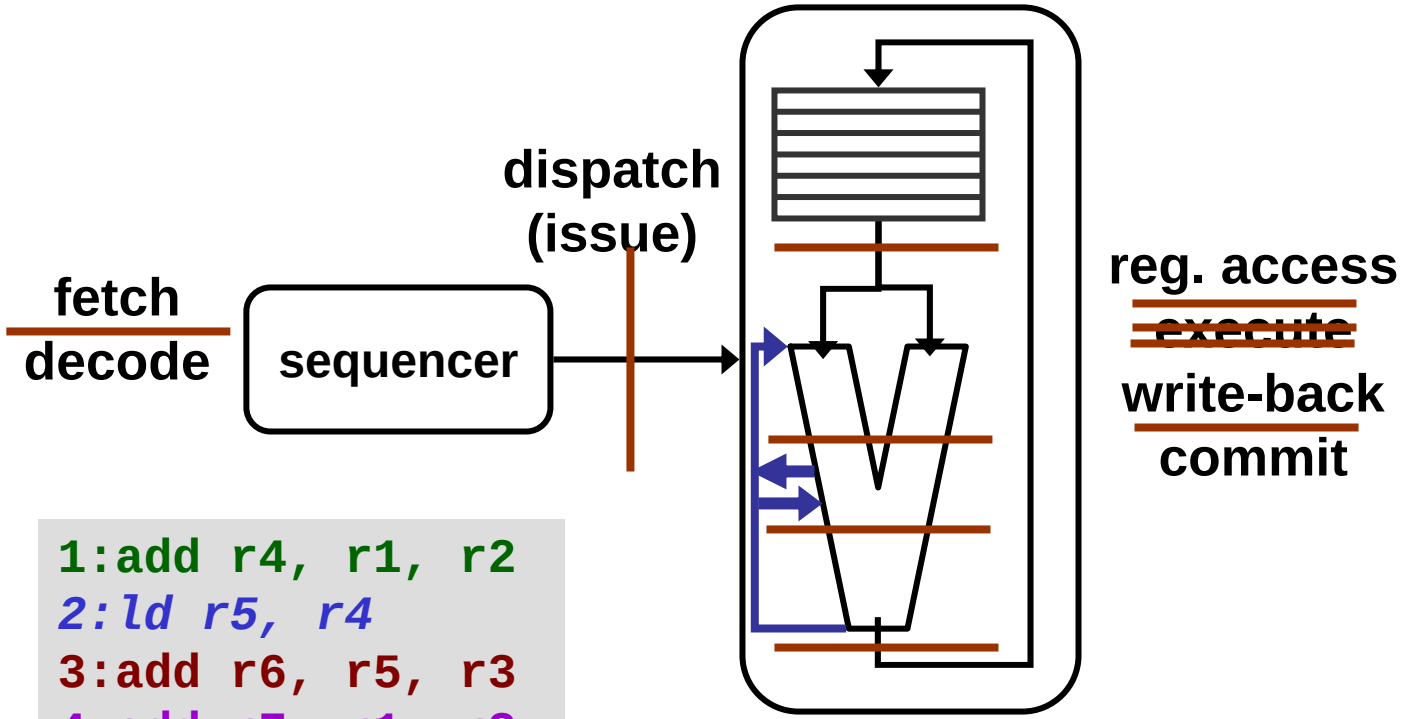



```
1:add r4, r1, r2
2:add r5, r1, r3
3:add r6, r2, r3
```

1	F	D	I	R	E	E	E	W	C														
2		F	D	I	R	E	E	E	W	C														
3			F	D	I	R	E	E	E	W	C													

Communication and synchronization mechanisms?

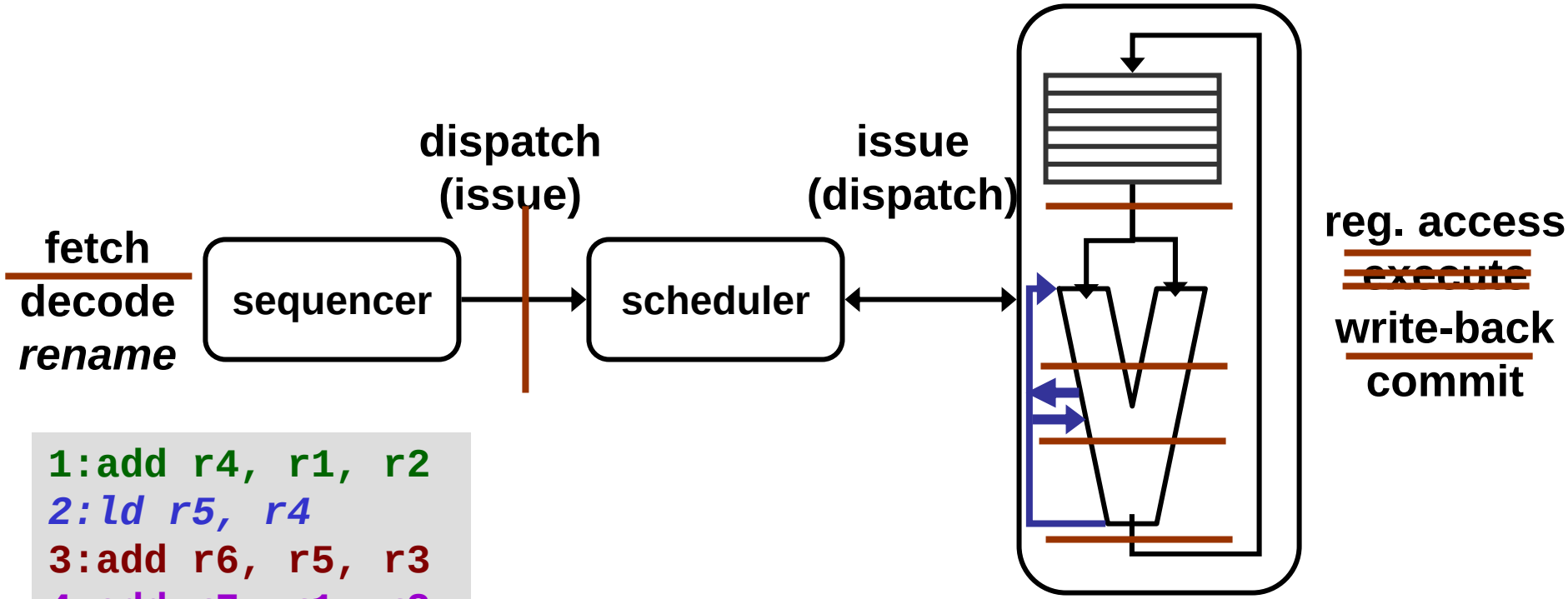
Simplified view of a pipelined processor




```
1:add r4, r1, r2
2:ld r5, r4
3:add r6, r5, r3
4:add r7, r1, r3
```

1	F	D	I	R	E	E	E	W	C														
2		F	D	I	R	L	L	L	L	L	L	L	L	L	W	C							
3			F	D	I											R	E	E	E	W	C		
4				F	D											I	R	E	E	E	W	C	

Simplified view of a OOO pipelined processor




```
1:add r4, r1, r2
2:ld r5, r4
3:add r6, r5, r3
4:add r7, r1, r3
```

1	F	D	I	d	R	E	E	E	W	C													
2		F	D	I	d	R	L	L	L	L	L	L	L	L	L	W	C							
3			F	D	I	d	R	E	E	E	W	C												

Communication and synchronization mechanisms?

Pipelining Summary

- Pipelining is using parallelism to hide latency
  - Do useful work while waiting for other work to finish
- Multiple parallel components, not multiple instances of same component
- Examples:
  - Execution pipeline
  - Memory pipelines
    - Issue multiple requests to memory without waiting for previous requests to complete
  - Software pipelines
    - Overlap different software blocks to hide latency: **computation/communication**

Outline

- Corrections/clarifications
- Pipelining
- Summary of parallel HW (multiple ALUs)
  - Analyze by shared resources
  - Analyze by synch/comm mechanisms
  - ILP, DLP, and TLP organizations
- Classification of ILP/DLP/TLP in software
- Patterns for parallel programming

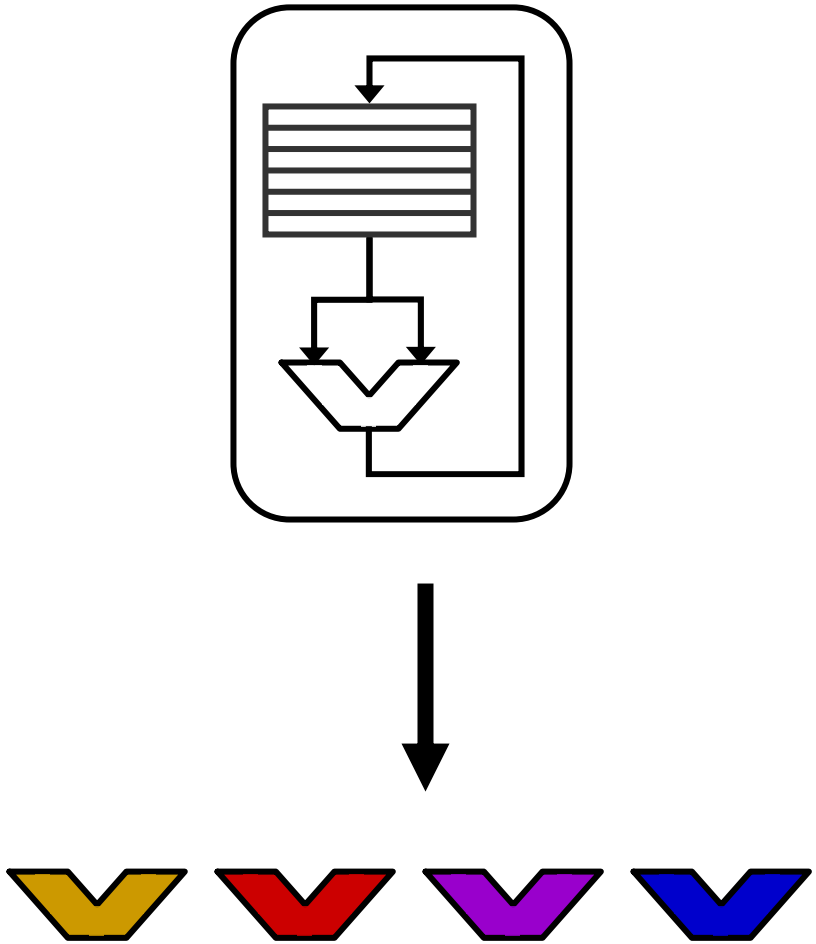
Resources in a parallel processor/system

- Execution
  - ALUs
  - Cores/processors
- Control
  - Sequencers
  - Instructions
  - OOO schedulers
- State
  - Registers
  - Memories
- Networks

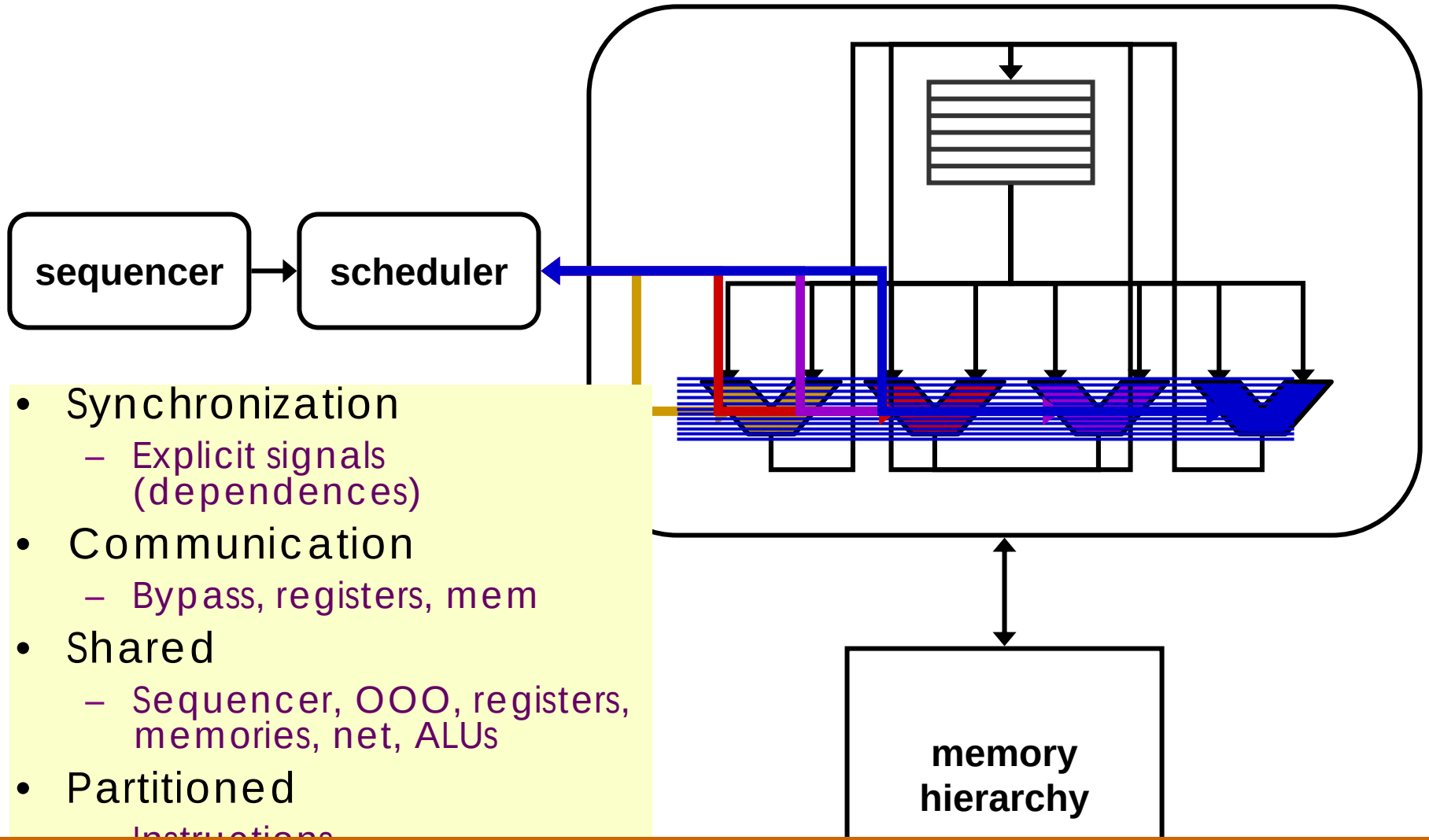
Communication and synchronization

- Synchronization
  - Clock – explicit compiler order
  - Explicit signals (e.g., dependences)
  - Implicit signals (e.g., flush/stall)
    - More for pipelining than multiple ALUs
- Communication
  - Bypass networks
  - Registers
  - Memory
  - Explicit (over some network)

Organizations for ILP (for multiple ALUs)



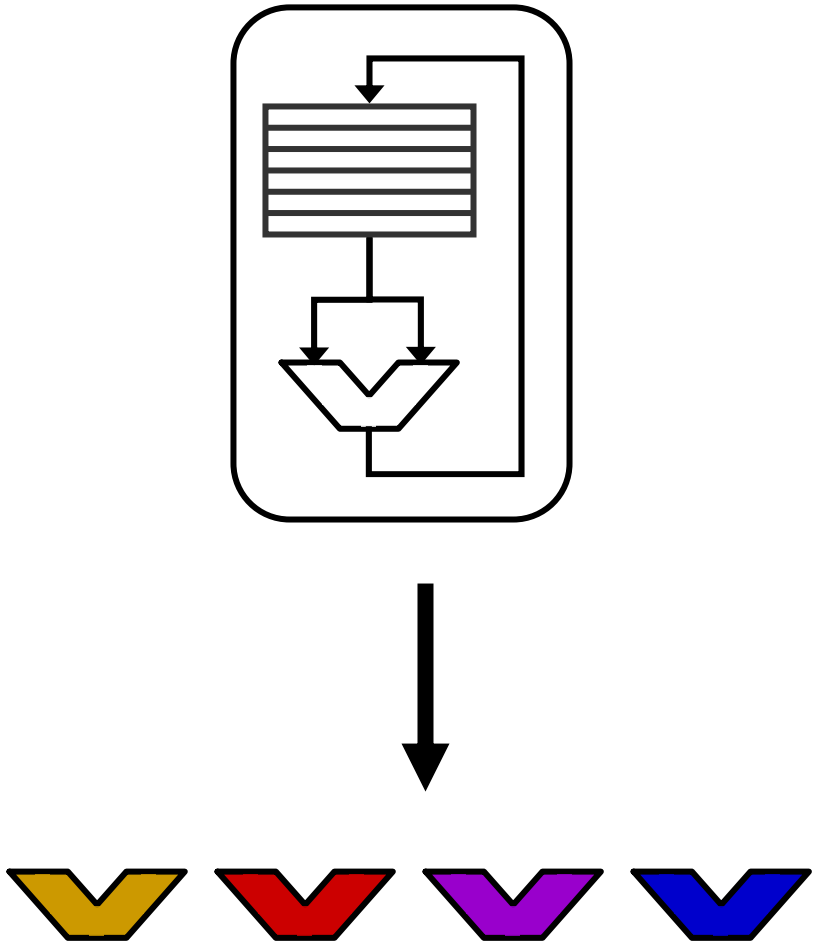
Superscalar (ILP for multiple ALUs)



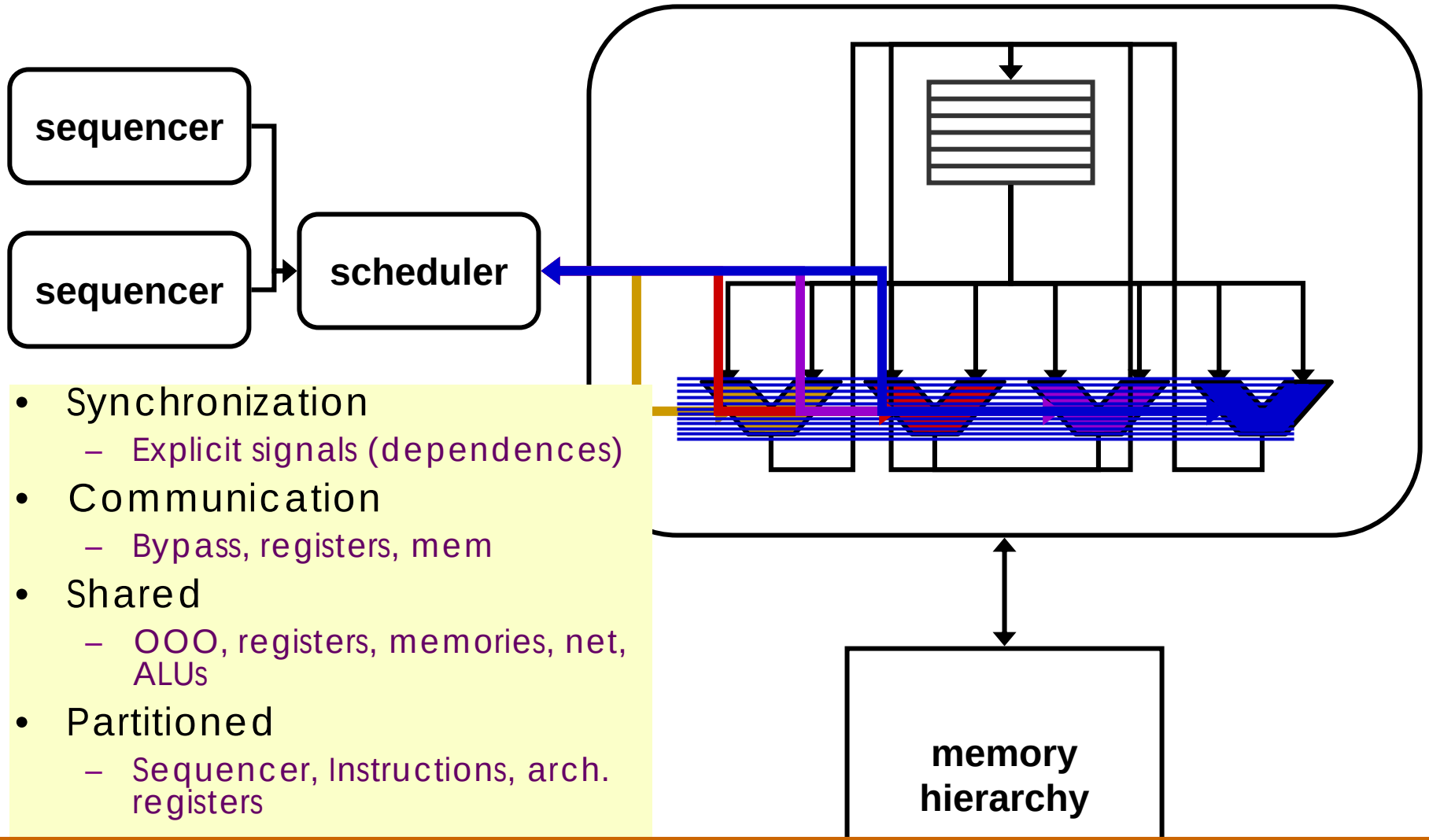
- Synchronization
  - Explicit signals (dependences)
- Communication
  - Bypass, registers, mem
- Shared
  - Sequencer, OOO, registers, memories, net, ALUs
- Partitioned
  - Instructions

How many ALUs?

SMT/TLS (ILP for multiple ALUs)



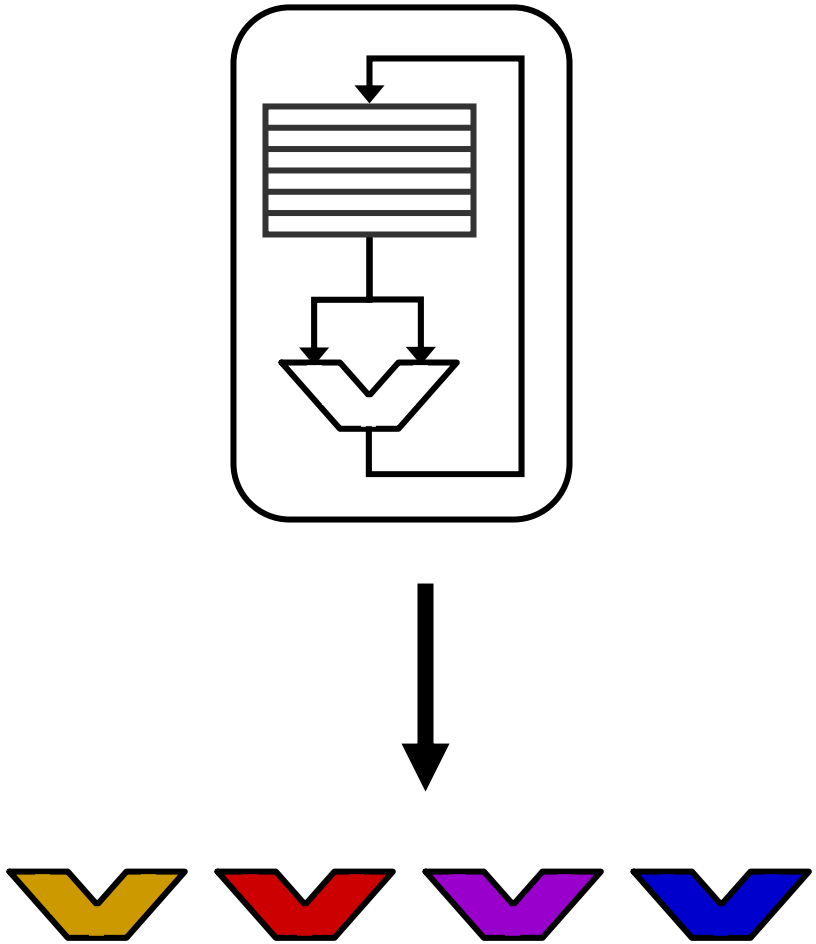
SMT/TLS (ILP for multiple ALUs)



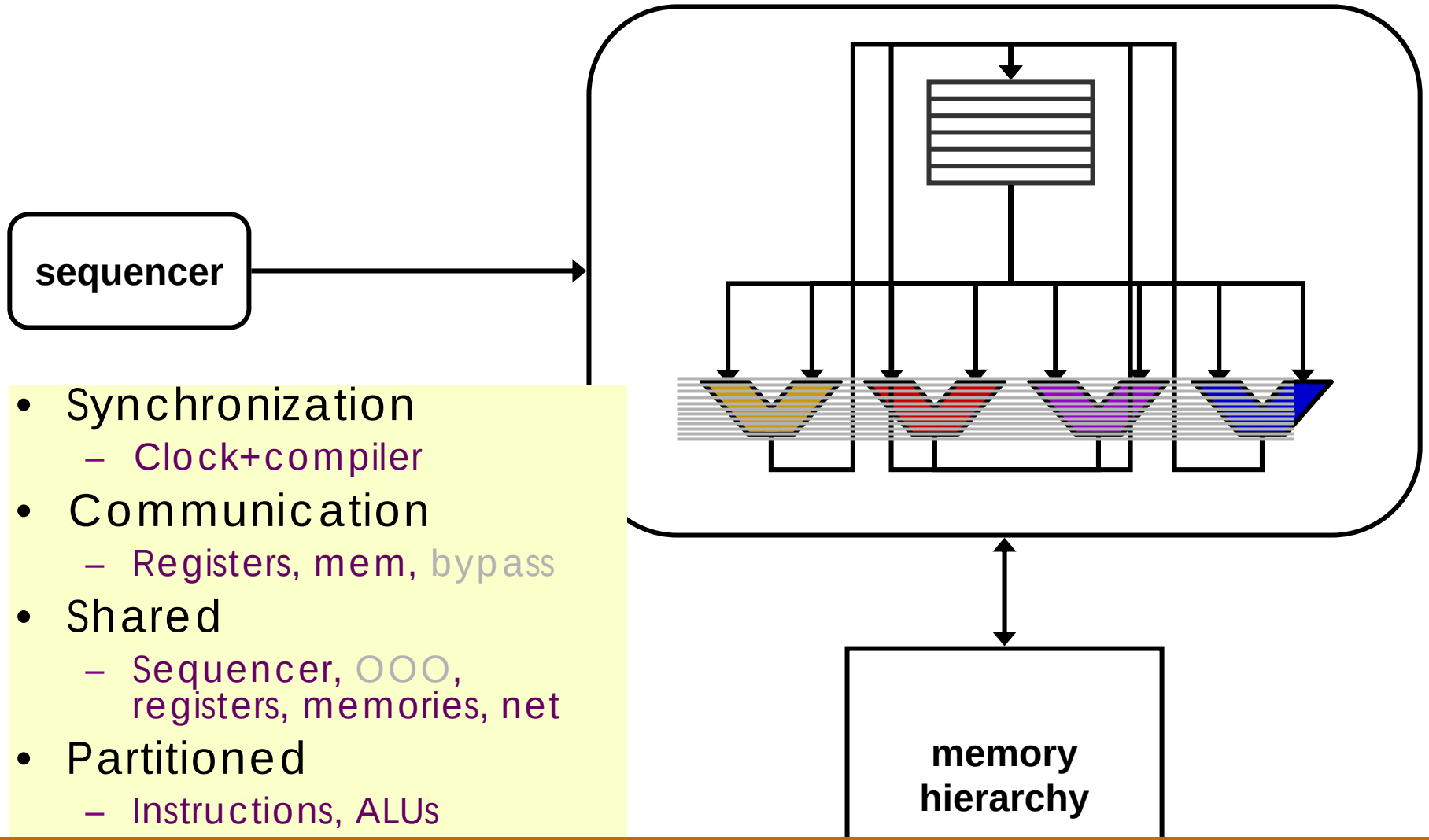
- Synchronization
  - Explicit signals (dependences)
- Communication
  - Bypass, registers, mem
- Shared
  - OOO, registers, memories, net, ALUs
- Partitioned
  - Sequencer, Instructions, arch. registers

Why is this ILP? How many threads?

VLIW (ILP for multiple ALUs



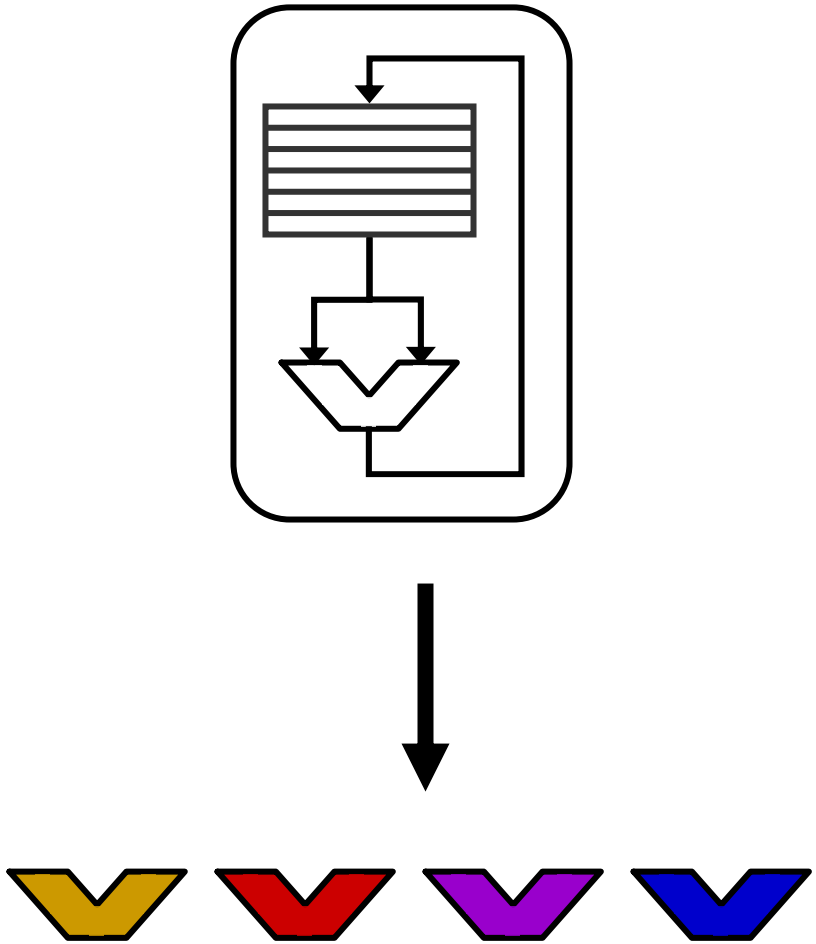
VLIW (ILP for multiple ALUs)



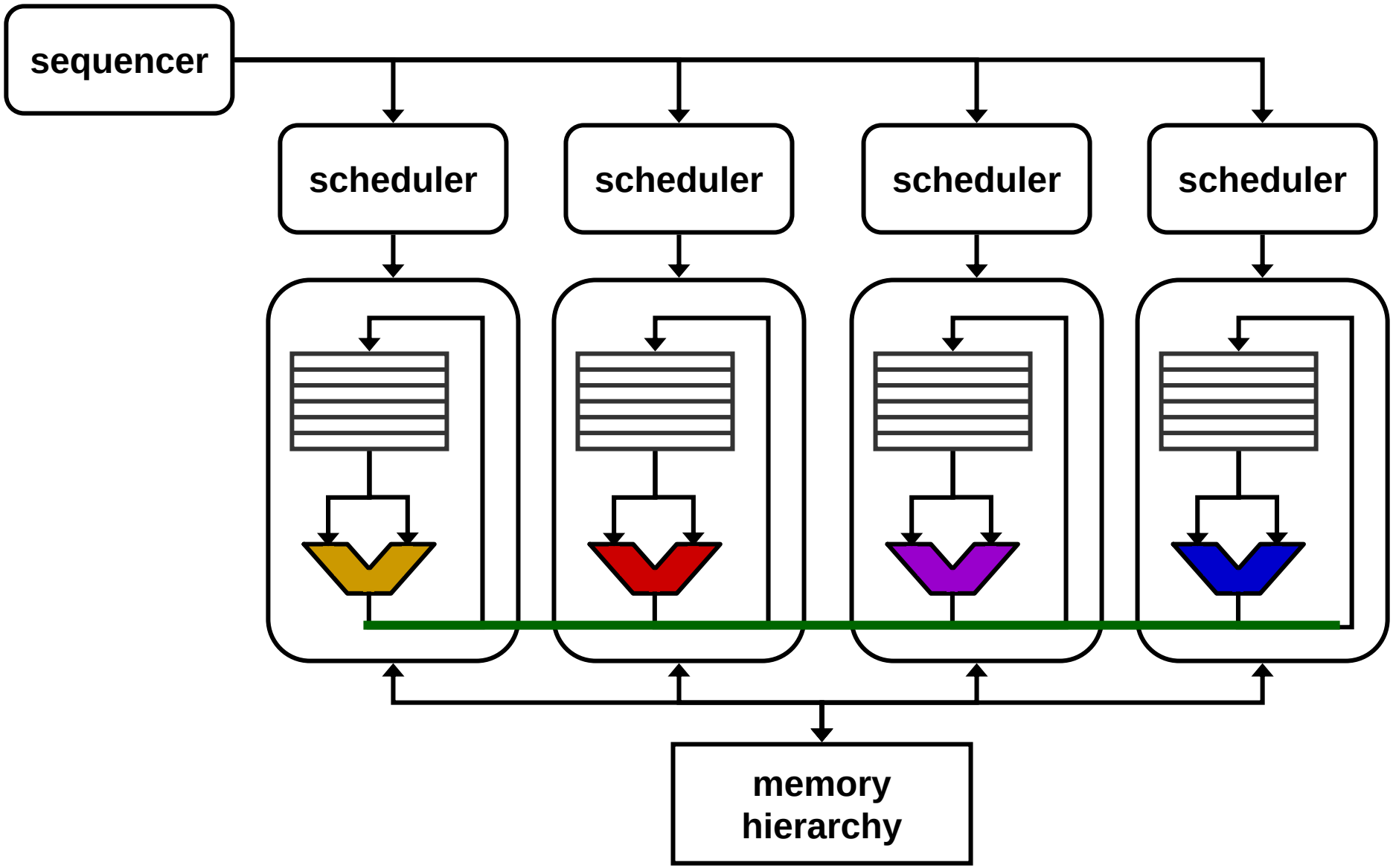
- Synchronization
  - Clock+compiler
- Communication
  - Registers, mem, bypass
- Shared
  - Sequencer, OOO, registers, memories, net
- Partitioned
  - Instructions, ALUs

How many ALUs?

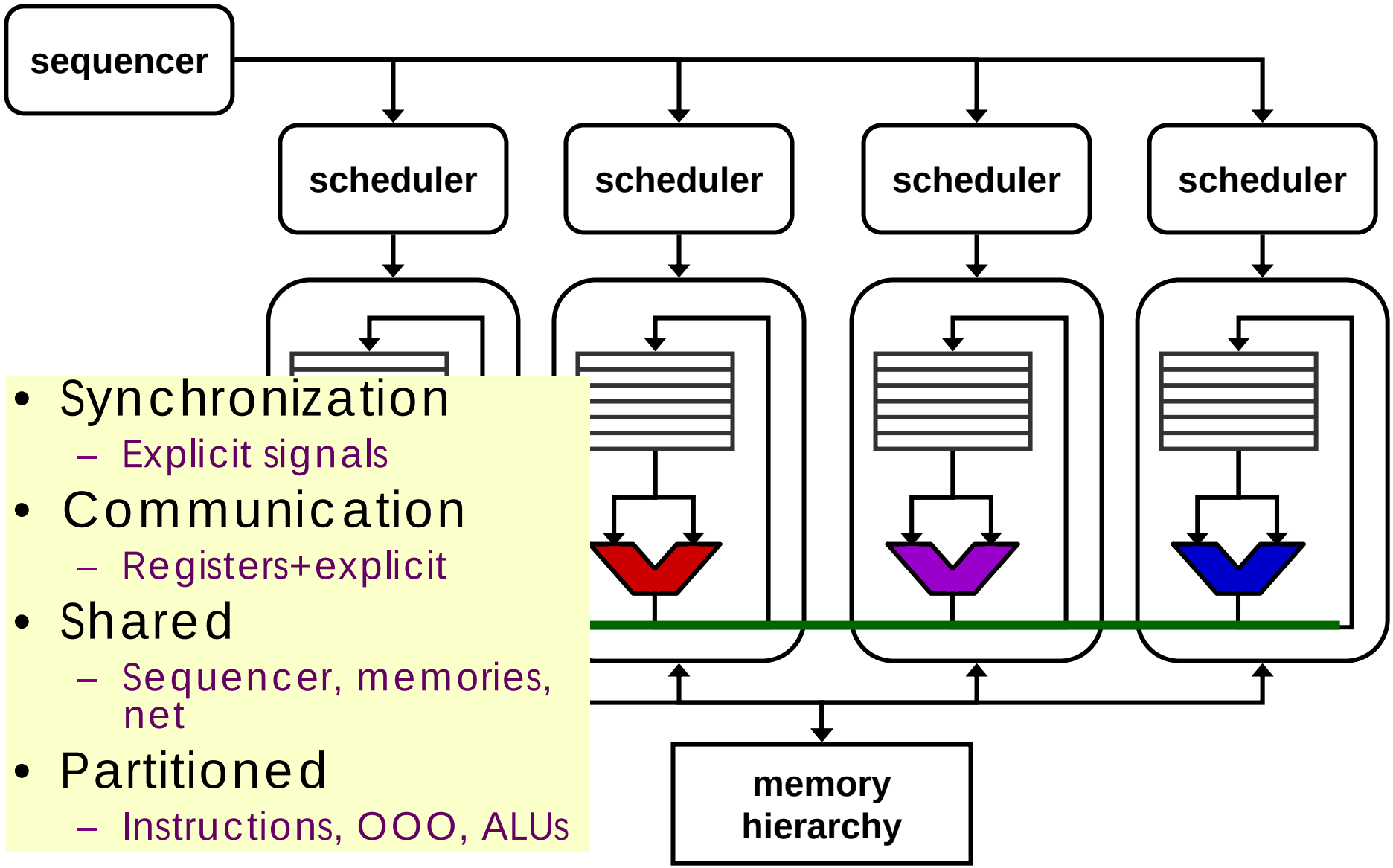
Explicit Dataflow (ILP for multiple ALUs)



Explicit Dataflow (ILP for multiple ALUs)

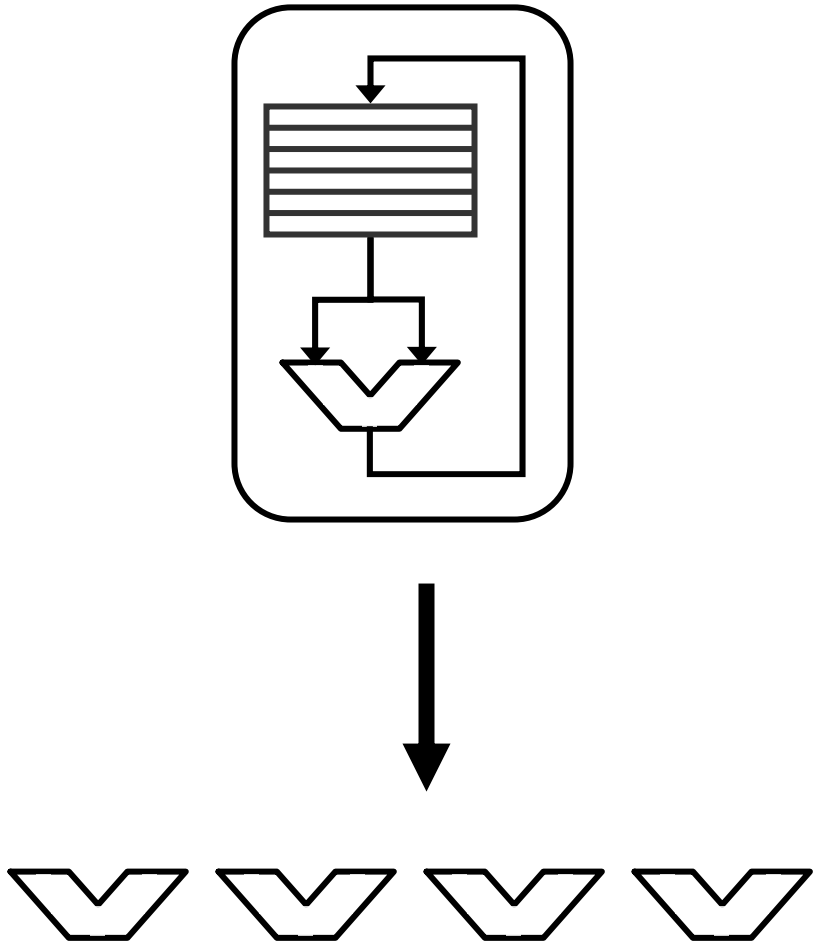


Explicit Dataflow (ILP for multiple ALUs)



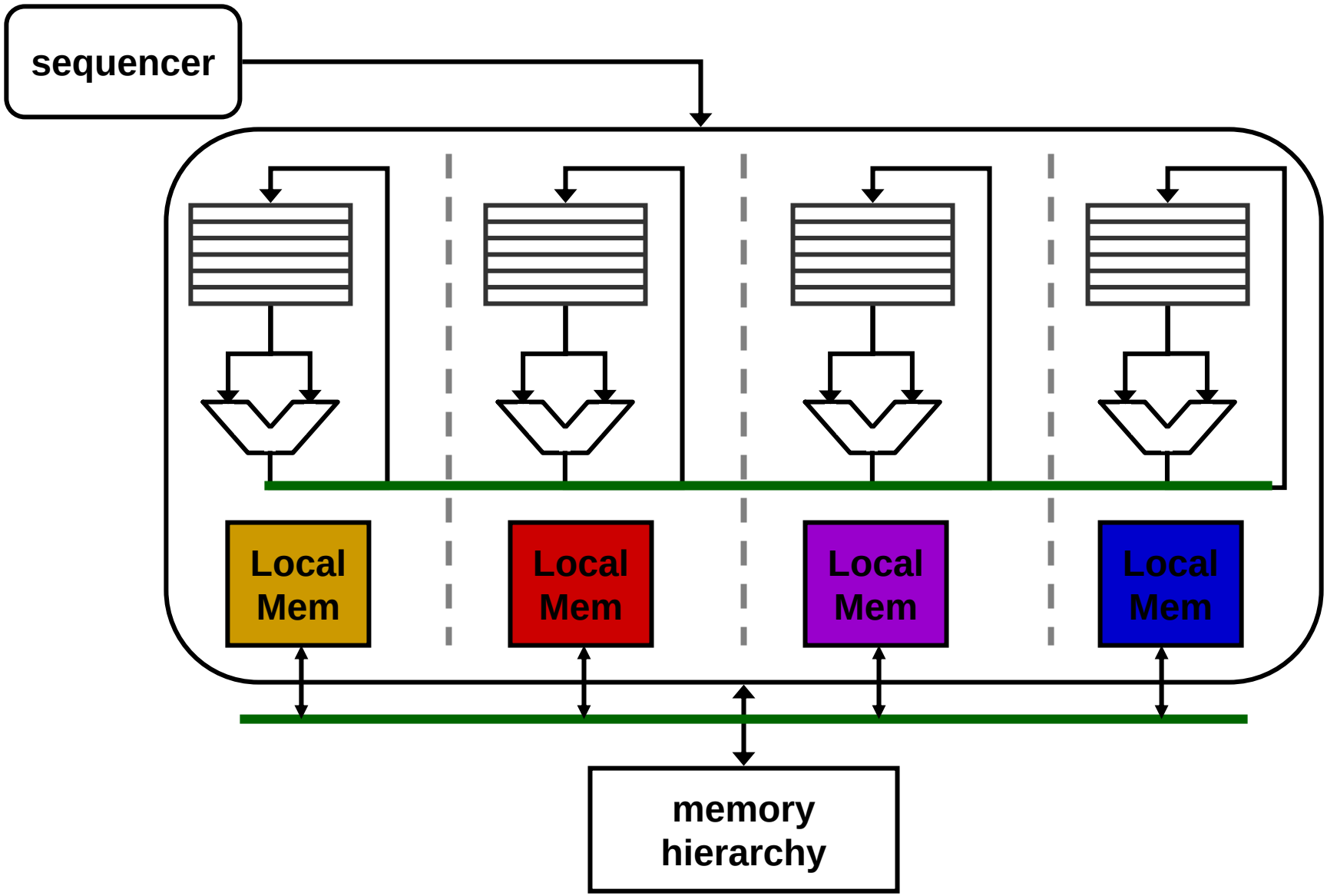
- Synchronization
  - Explicit signals
- Communication
  - Registers+explicit
- Shared
  - Sequencer, memories, net
- Partitioned
  - Instructions, OOO, ALUs

DLP for multiple ALUs

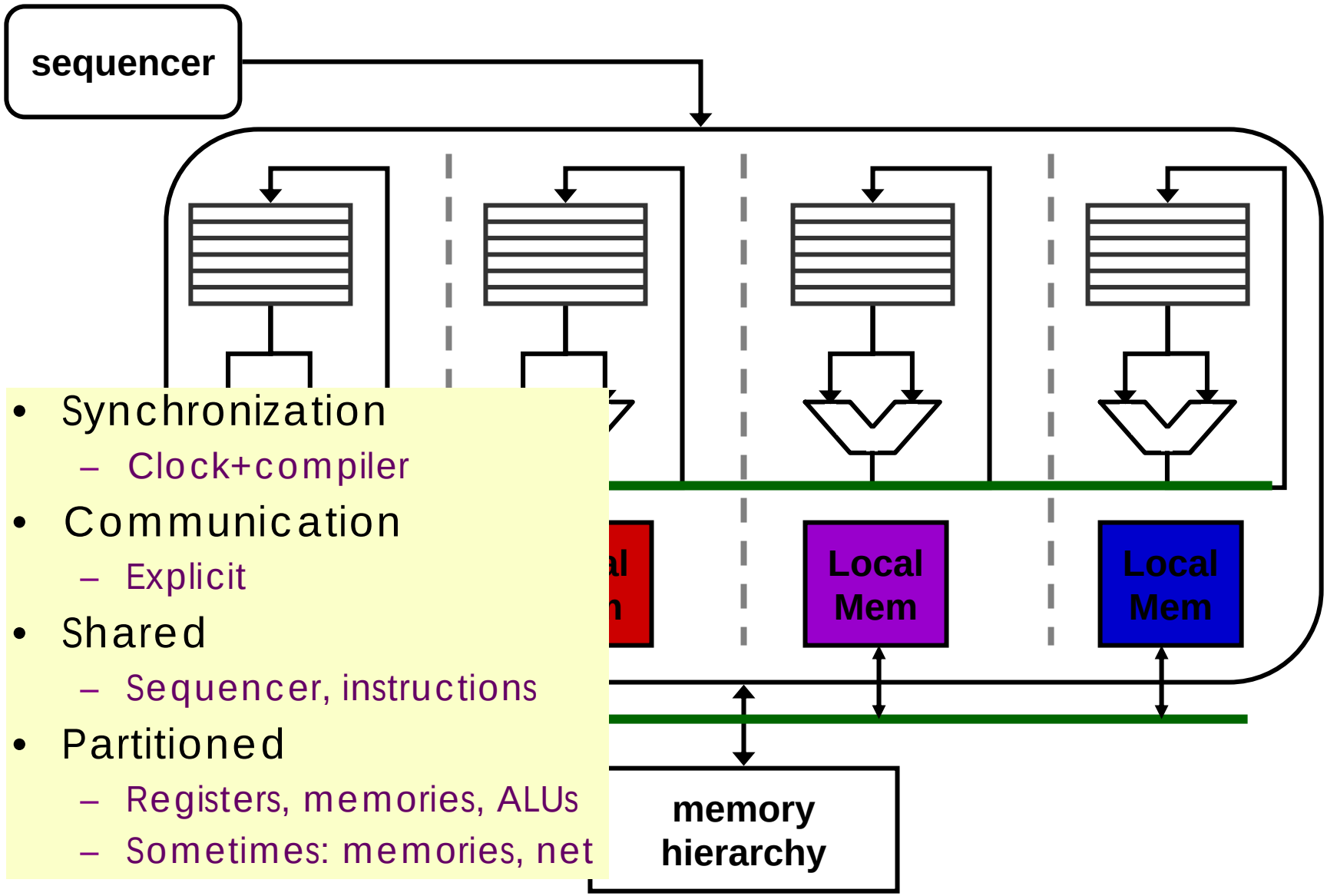


From HW – this is SIMD

SIMD (DLP for multiple ALUs)

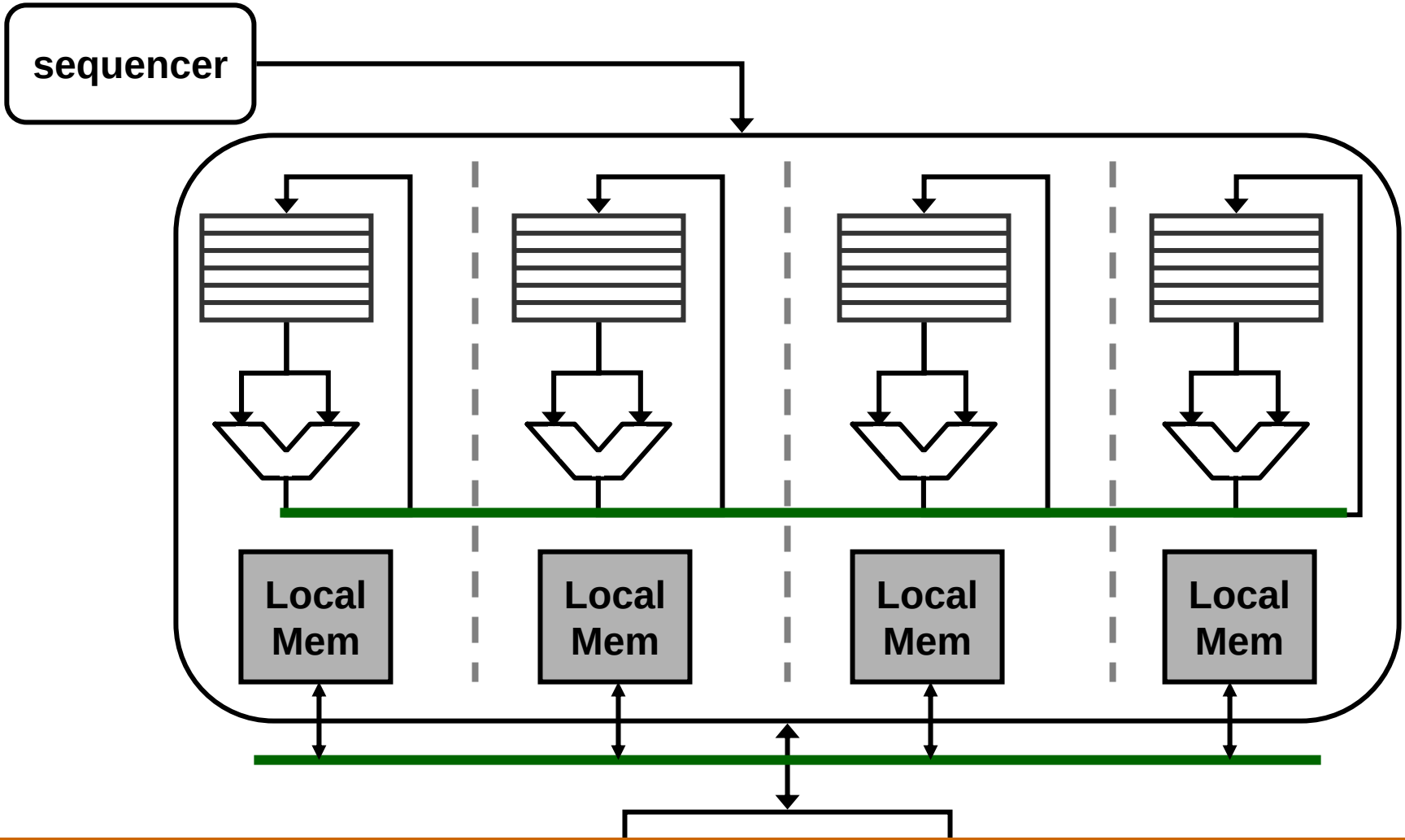


SIMD (DLP for multiple ALUs)



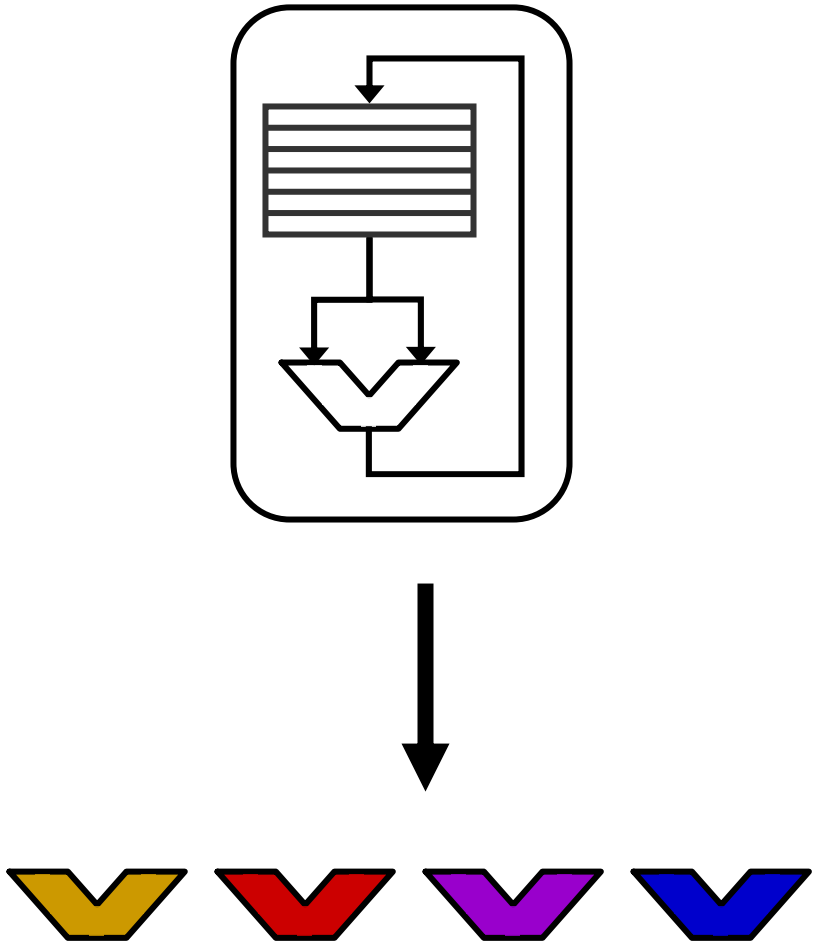
- Synchronization
  - Clock+compiler
- Communication
  - Explicit
- Shared
  - Sequencer, instructions
- Partitioned
  - Registers, memories, ALUs
  - Sometimes: memories, net

Vectors (DLP for multiple ALUs)

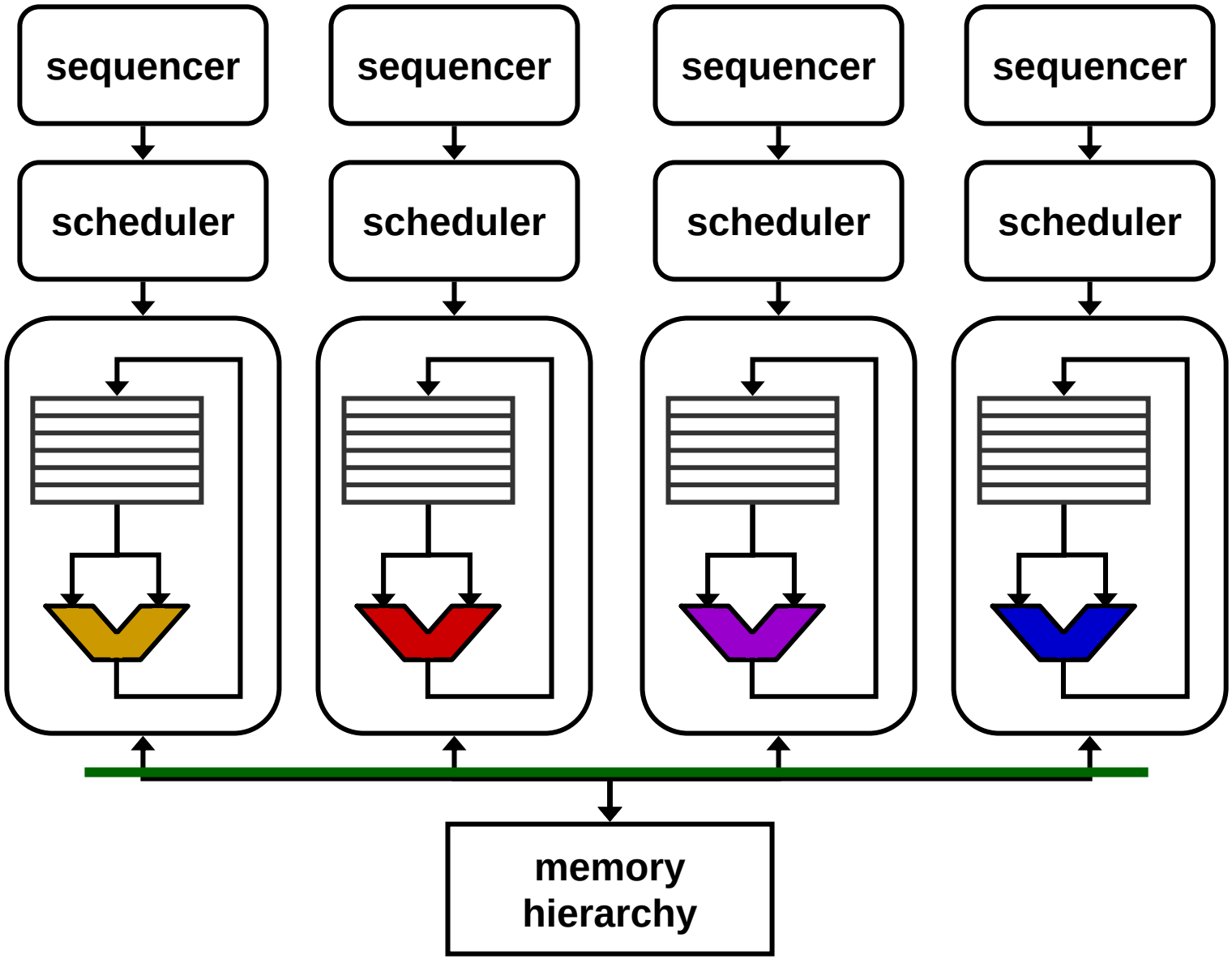


Vectors: memory addresses are part of single-instruction and not part of multiple-data

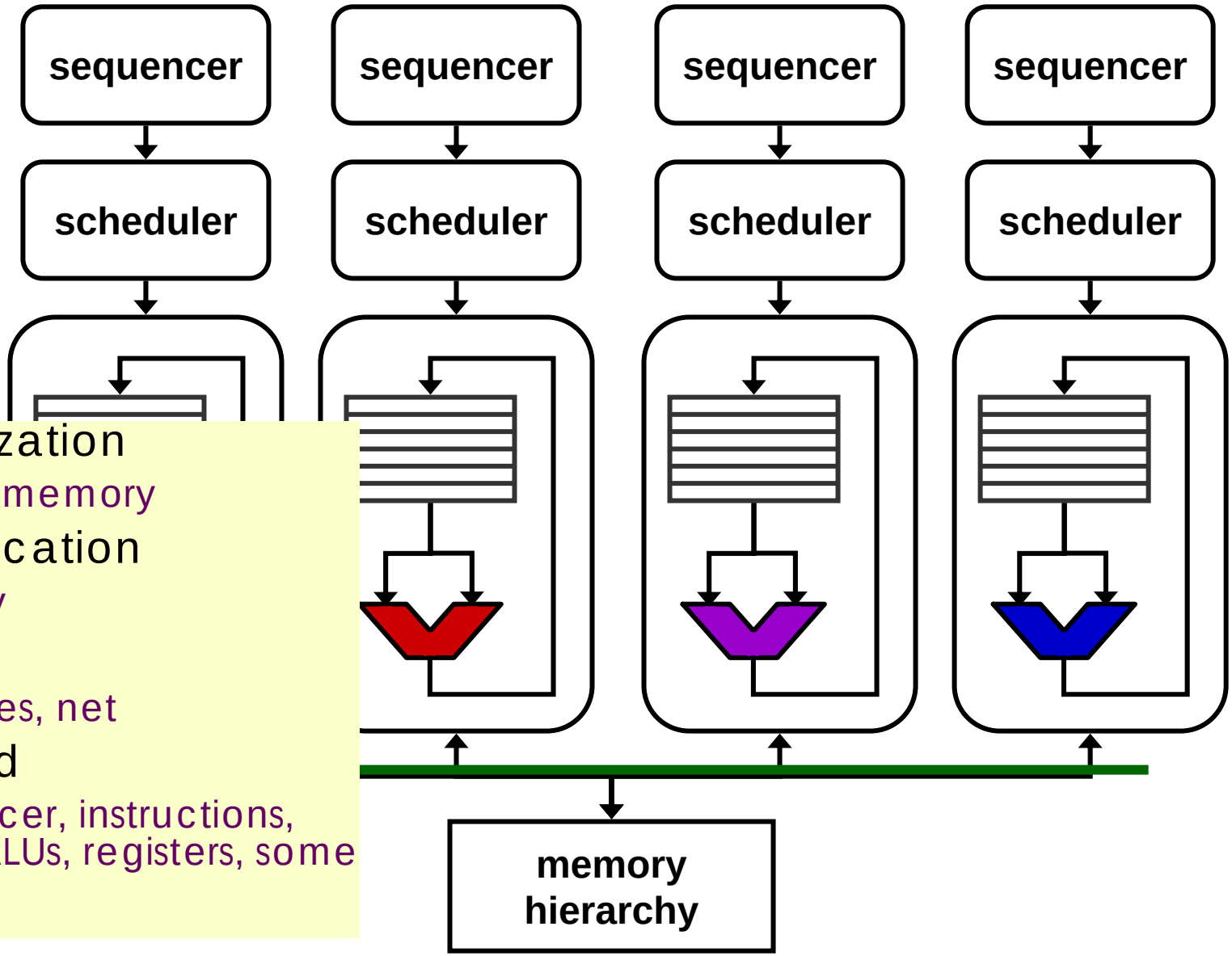
TLP for multiple ALUs



MIMD – shared memory (TLP for multiple ALUs)

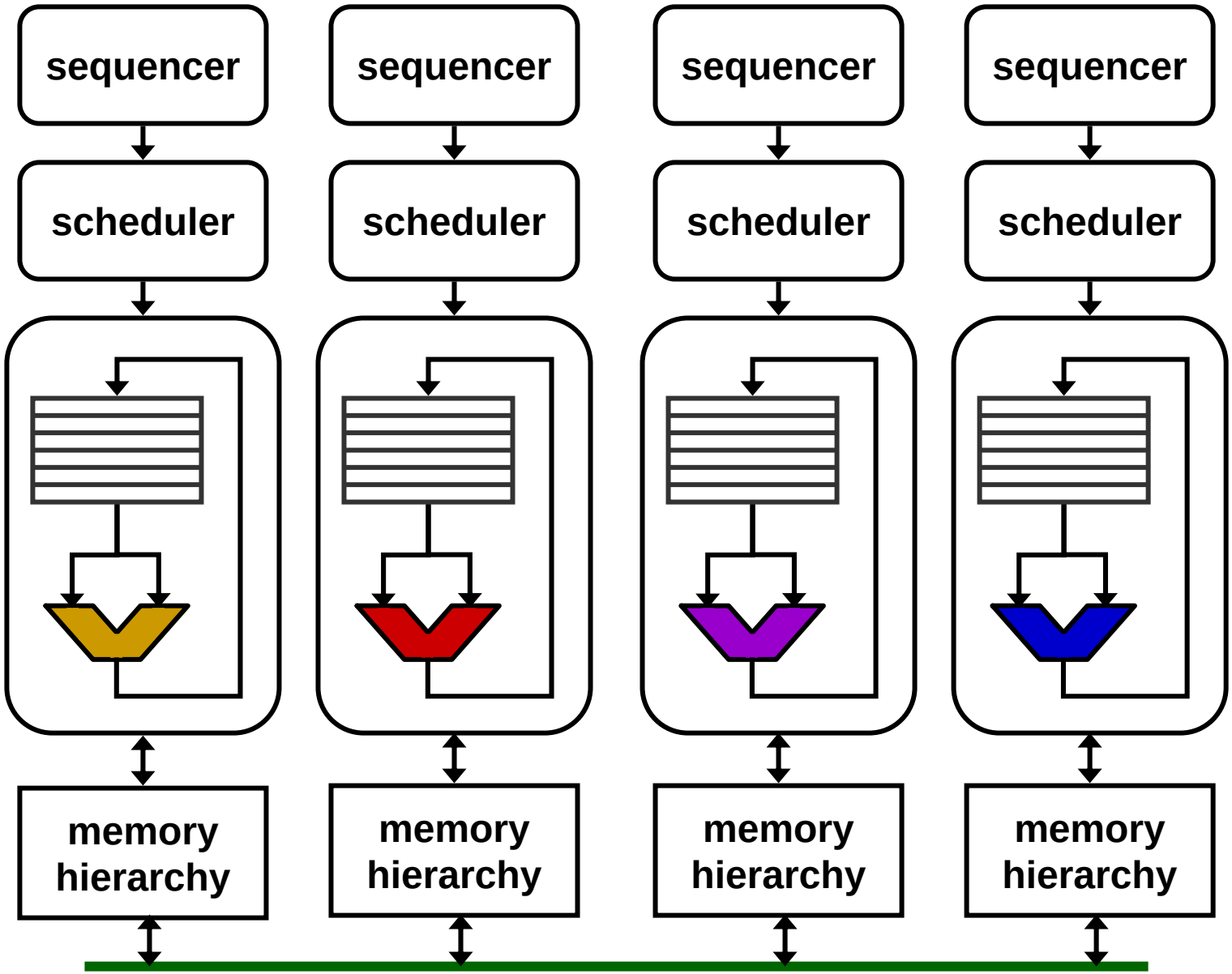


MIMD – shared memory (TLP for multiple ALUs)

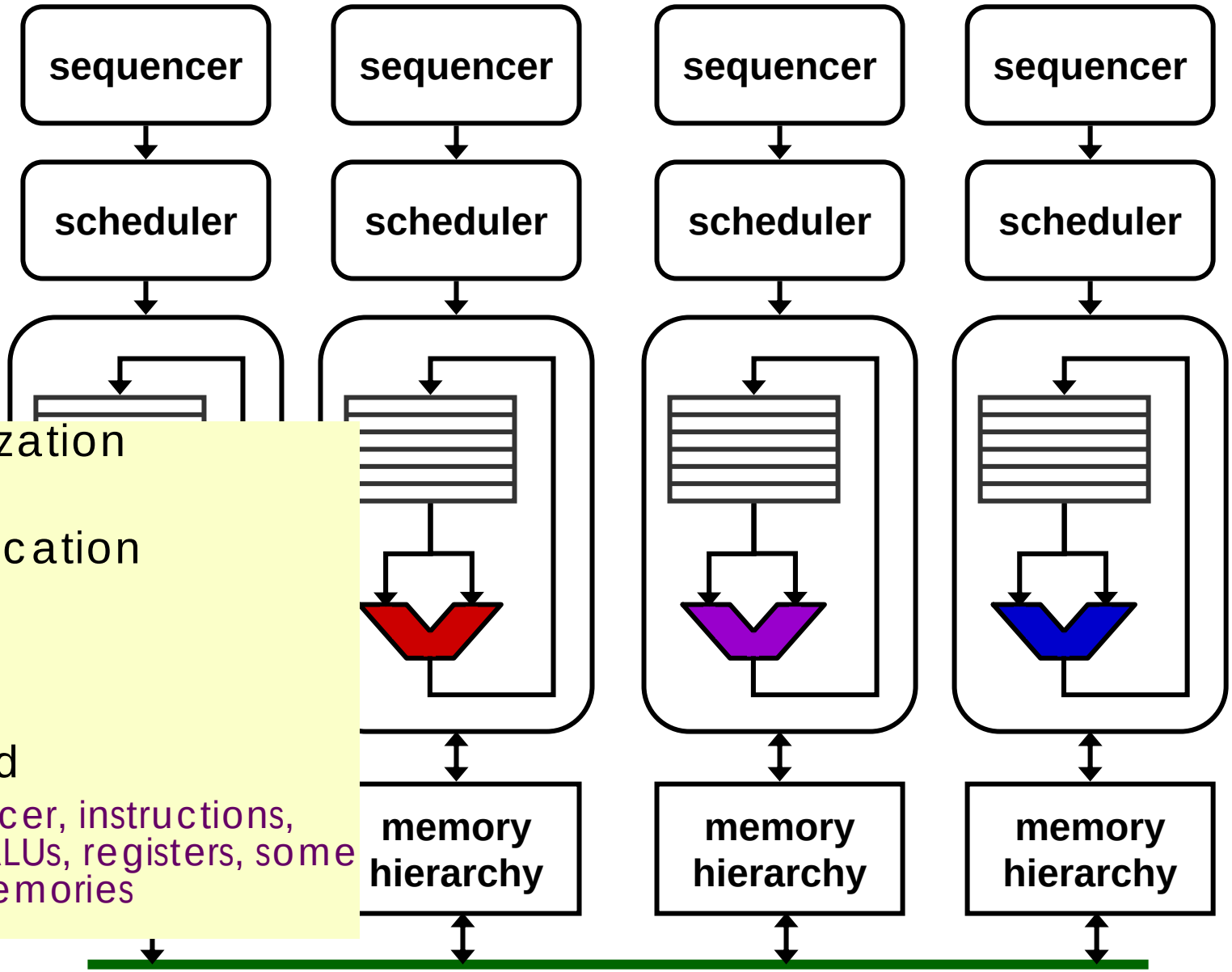


- Synchronization
  - Explicit, memory
- Communication
  - Memory
- Shared
  - Memories, net
- Partitioned
  - Sequencer, instructions, OOO, ALUs, registers, some nets

MIMD – distributed memory



MIMD – distributed memory



- Synchronization
  - Explicit
- Communication
  - Explicit
- Shared
  - Net
- Partitioned
  - Sequencer, instructions, OOO, ALUs, registers, some nets, memories

Summary of communication and synchronization

Style	Synchronization	Communication
Superscalar	explicit signals (RS)	registers+bypass
VLIW	clock+compiler	registers (bypass?)
Dataflow	explicit signals	registers+explicit
SIMD	clock+compiler	explicit
MIMD	explicit signals	memory+explicit

Summary of sharing in ILP HW

Style	Seq	Inst	OOO	Regs	Mem	ALUs	Net
Superscalar	S	P	S	S	S	S	S
SMT/TLS	P	P	S	S	S	S	S
VLIW	S	P	N/A	S	S	**P**	S
Dataflow	B	P	P	B	S	**P**	S

Summary of sharing in DLP and TLP

Style	Seq	Inst	OOO	Regs	Mem	ALUs	Net
Vector	S	S	N/A	P	**s**	P	B
SIMD	S	S	N/A	P	**p**	P	B
MIMD	P	P	P	P	S/P	P	B

ILP/DLP/TLP in Software?

- Back to the board.