Chapter 7

Multicores, Multiprocessors, and Clusters

There are finer fish in the sea than

have ever been caught

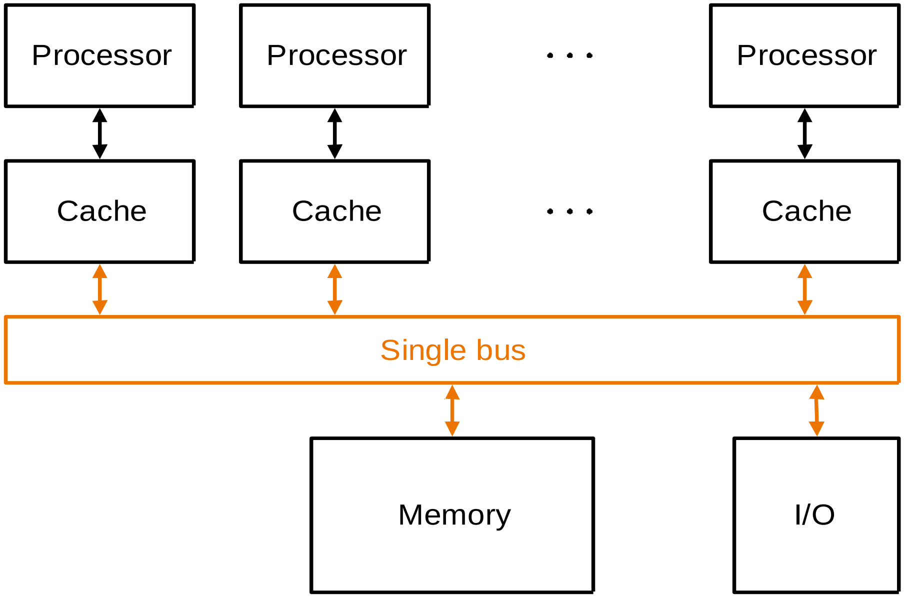
Irish proverb

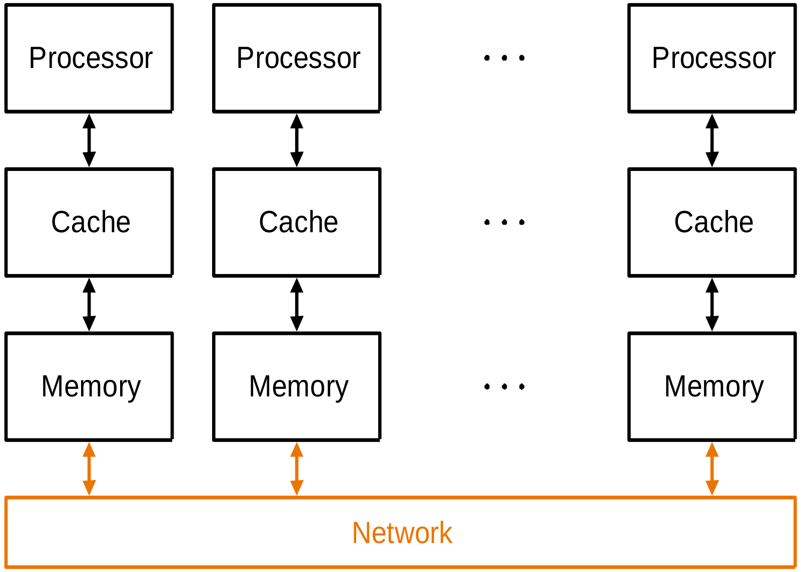
Multiprocessors

- **Idea: create powerful computers by connecting many smaller ones**

good news: works for timesharing (better than supercomputer)
vector processing may be coming back

bad news: its really hard to write good concurrent programs
many commercial failures



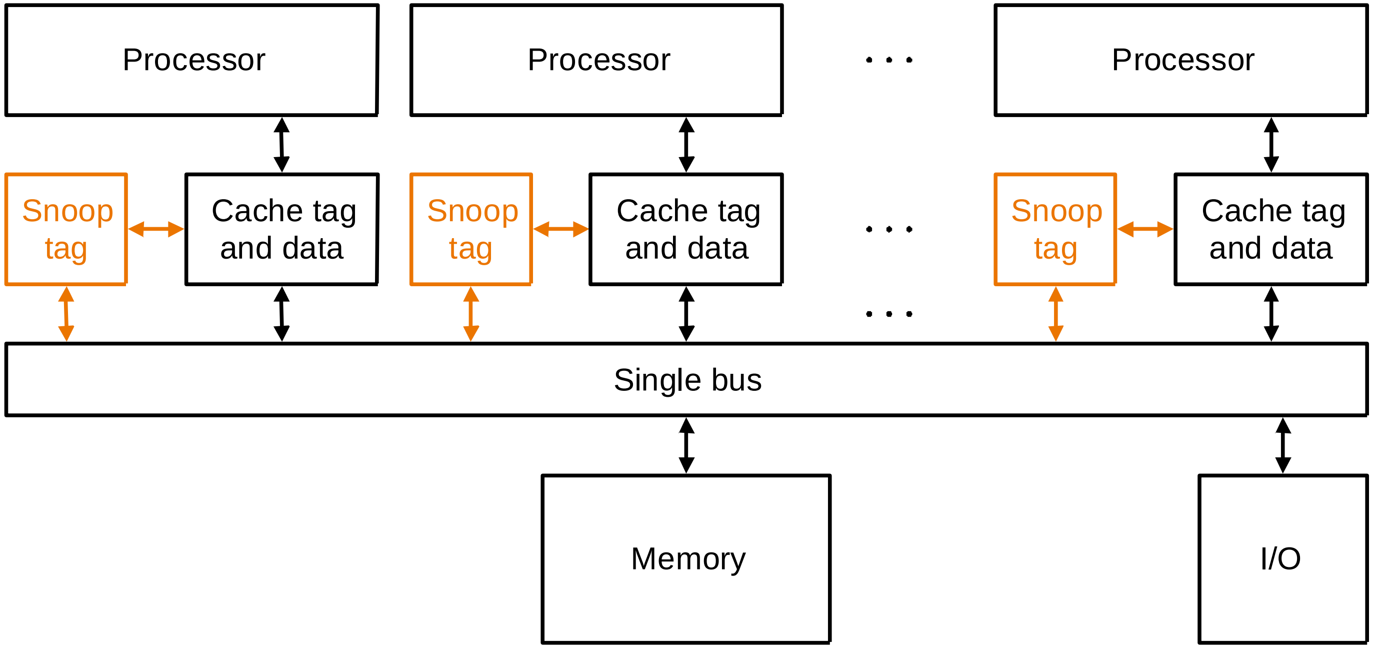


Questions

- How do parallel processors share data?
  - — single address space (SMP vs. NUMA)
  - — message passing
- How do parallel processors coordinate?
  - — synchronization (locks, semaphores)
  - — built into send / receive primitives
  - — operating system protocols
- How are they implemented?
  - — connected by a single bus
  - — connected by a network

Some Interesting Problems

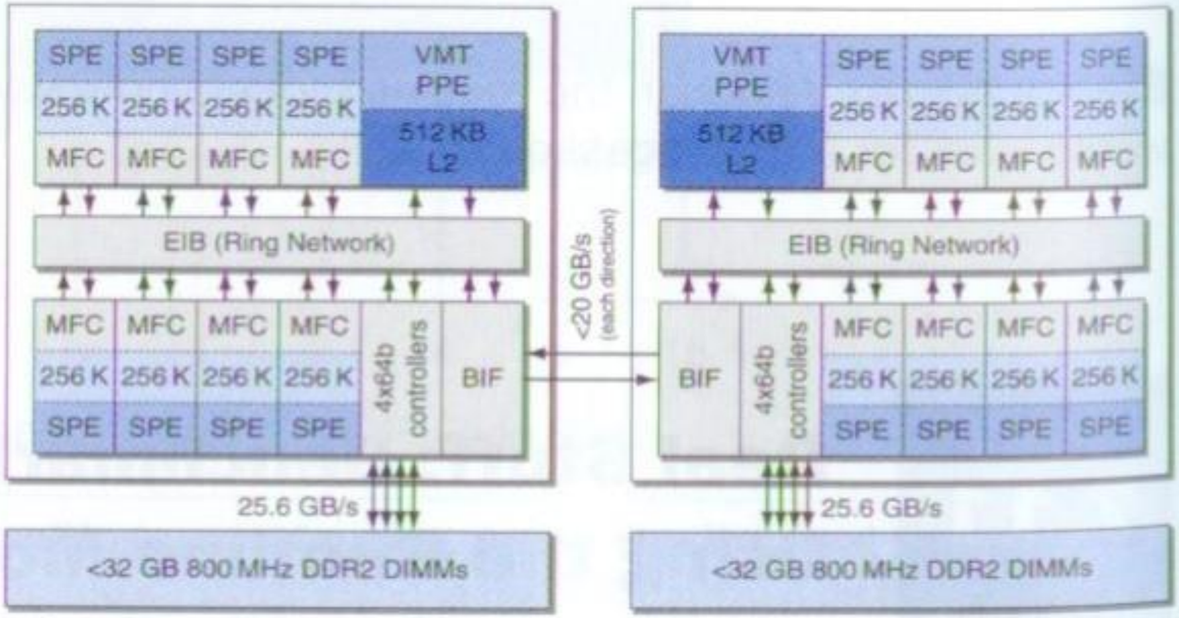
- **Cache Coherency**



- **Synchronization**
  — provide special atomic instructions (test-and-set, swap, etc.)
- **Network Topology**

Terminology

- **Multi-core (MIMD)**
- **Multi-threading**
- **Simultaneous multithreading (SMT); hyper-threading**



(d) IBM Cell QS20

Simultaneous Multi-threading ...

One thread, 8 units

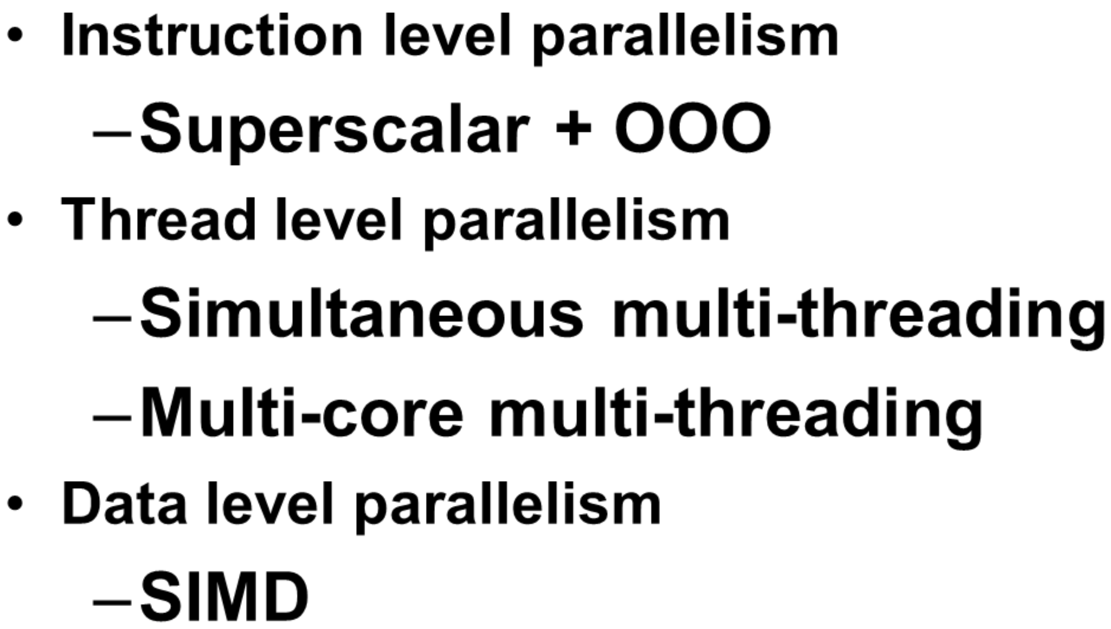
Cycle	M	M	FX	FX	FP	FP	BR	CC
1	🟨							🟨
2	🟨	🟨					🟨	
3				🟨	🟨			
4								
5								
6								
7	🟨			🟨		🟨		
8		🟨			🟨			
9				🟨				

Two threads, 8 units

Cycle	M	M	FX	FX	FP	FP	BR	CC
1	🟨	🟦	🟦					🟨
2	🟨	🟨	🟦			🟦	🟨	
3	🟦			🟨	🟨			
4	🟦	🟦				🟦		
5		🟦						🟦
6								
7	🟨		🟦	🟨	🟦	🟨		
8		🟨		🟦	🟨	🟦		
9	🟦	🟦		🟨		🟦		

M = Load/Store, FX = Fixed Point, FP = Floating Point, B nch, CC = Co ion Codes

SIMD and Vector Processing



- Instruction level parallelism
  - Superscalar + OOO
- Thread level parallelism
  - Simultaneous multi-threading
  - Multi-core multi-threading
- Data level parallelism
  - SIMD

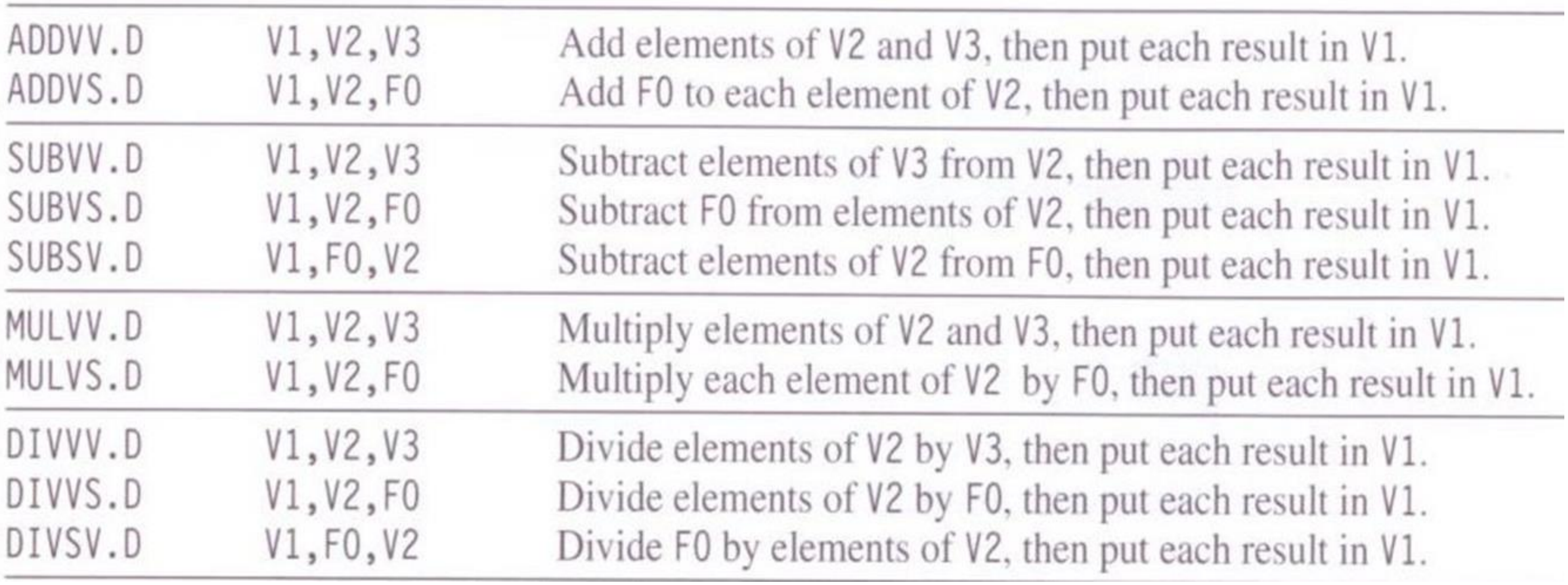
Example SIMD Code, 256-bit reg

4D: operate on 4 double-precision operands

- Example DXPY:

```
    L.D         F0,a             ;load scalar a
    MOV         F1, F0           ;copy a into F1 for SIMD MUL
    MOV         F2, F0           ;copy a into F2 for SIMD MUL
    MOV         F3, F0           ;copy a into F3 for SIMD MUL
    DADDIU      R4,Rx,#512       ;last address to load
Loop:           L.4D F4,0[Rx]    ;load X[i], X[i+1], X[i+2], X[i+3]
    MUL.4D      F4,F4,F0         ;a×X[i],a×X[i+1],a×X[i+2],a×X[i+3]
    L.4D        F8,0[Ry]         ;load Y[i], Y[i+1], Y[i+2], Y[i+3]
    ADD.4D      F8,F8,F4         ;a×X[i]+Y[i], ..., a×X[i+3]+Y[i+3]
    S.4D        0[Ry],F8         ;store into Y[i], Y[i+1], Y[i+2], Y[i+3]
    DADDIU      Rx,Rx,#32        ;increment index to X
    DADDIU      Ry,Ry,#32        ;increment index to Y
    DSUBU       R20,R4,Rx        ;compute bound
    BNEZ        R20,Loop         ;check if done
```

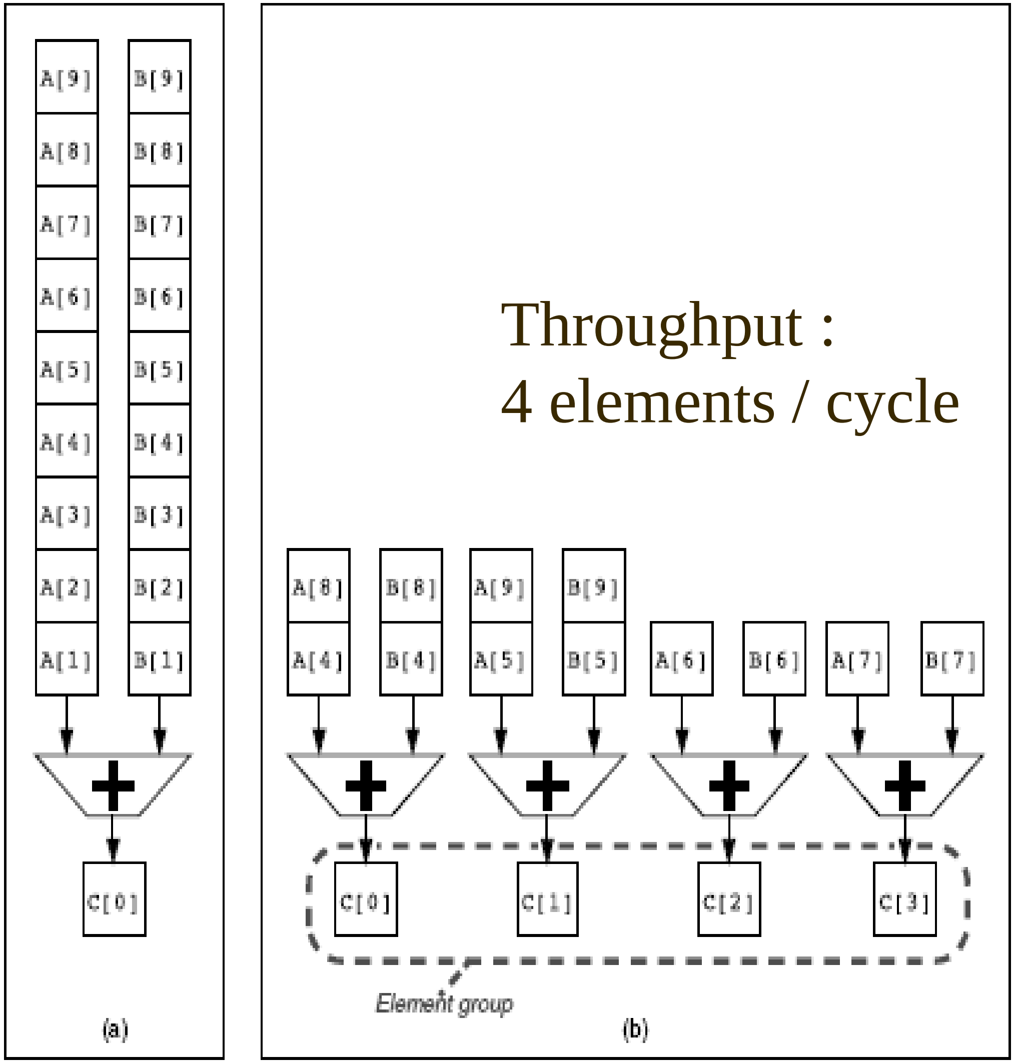
Vector Instructions; some examples



ADDVV.D	V1,V2,V3	Add elements of V2 and V3, then put each result in V1.
ADDVS.D	V1,V2,F0	Add F0 to each element of V2, then put each result in V1.
SUBVV.D	V1,V2,V3	Subtract elements of V3 from V2, then put each result in V1.
SUBVS.D	V1,V2,F0	Subtract F0 from elements of V2, then put each result in V1.
SUBSV.D	V1,F0,V2	Subtract elements of V2 from F0, then put each result in V1.
MULVV.D	V1,V2,V3	Multiply elements of V2 and V3, then put each result in V1.
MULVS.D	V1,V2,F0	Multiply each element of V2 by F0, then put each result in V1.
DIVVV.D	V1,V2,V3	Divide elements of V2 by V3, then put each result in V1.
DIVVS.D	V1,V2,F0	Divide elements of V2 by F0, then put each result in V1.
DIVSV.D	V1,F0,V2	Divide F0 by elements of V2, then put each result in V1.

Multiple pipelines for a vector ADD

- C = A+ B
- (a) single pipeline
- (b) four pipelines
- Element *n* of vector register *A* is “hardwired” to element *n* of vector register *B*
  - Allows for multiple hardware lanes
  - Elements of A and B are interleaved across the four pipelines



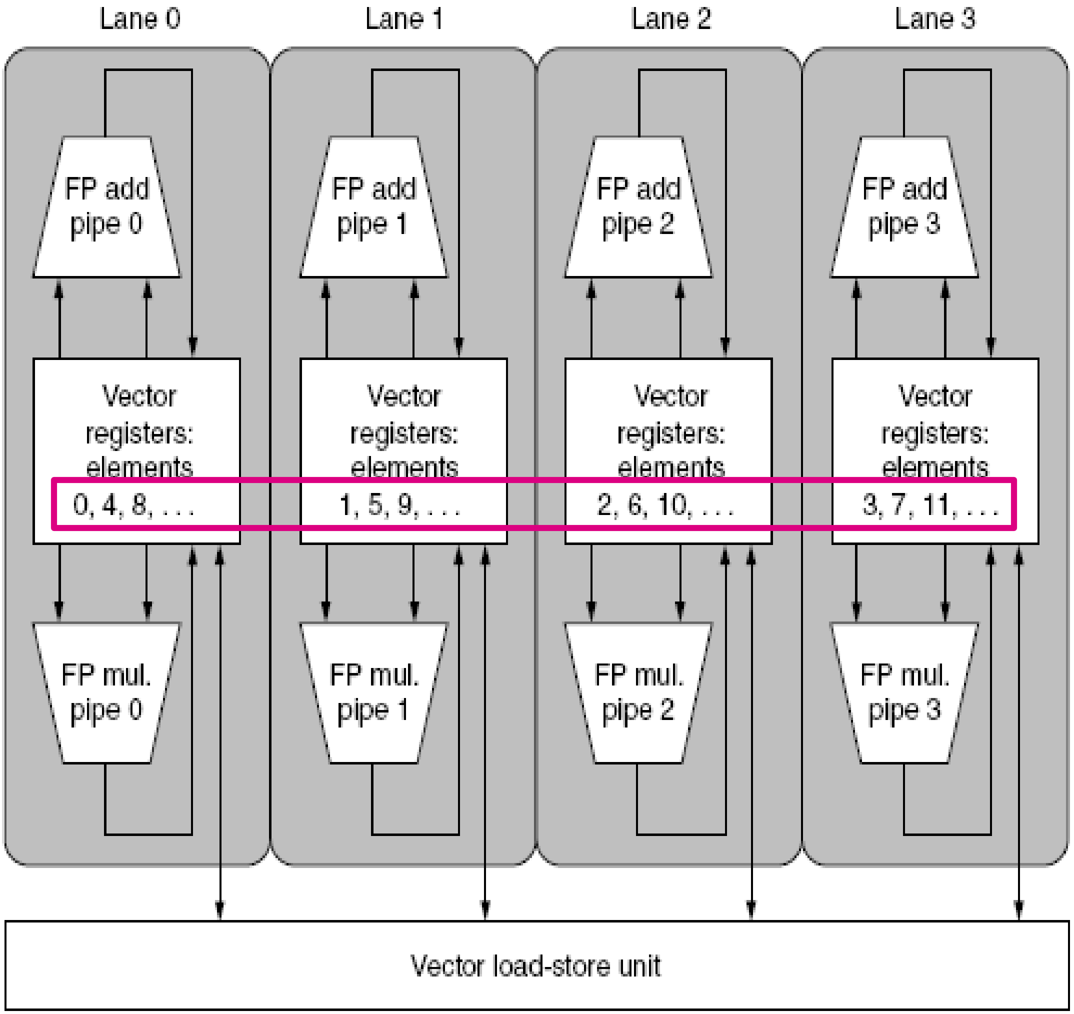
Throughput :
4 elements / cycle

)

1 element / cycle

Simply spread the elements of a vector register across the lanes

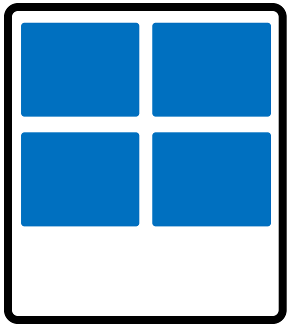
- **A four lane**
- **First lane holds element 0 for all vector registers**
- **A 64-cycle Chime -> 16 cycles**


Lane 0
Lane 1
Lane 2
Lane 3
FP add pipe 0
FP add pipe 1
FP add pipe 2
FP add pipe 3
Vector registers: elements 0, 4, 8, . . .
Vector registers: elements 1, 5, 9, . . .
Vector registers: elements 2, 6, 10, . . .
Vector registers: elements 3, 7, 11, . . .
FP mul. pipe 0
FP mul. pipe 1
FP mul. pipe 2
FP mul. pipe 3
Vector load-store unit

Heterogeneous System

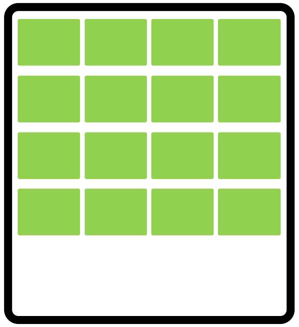
CPU & GPU

CPU



- CPU wants GPU’s capability (SIMD)
- Sequential thread with limited data parallelism
- 8 ~ 16 cores

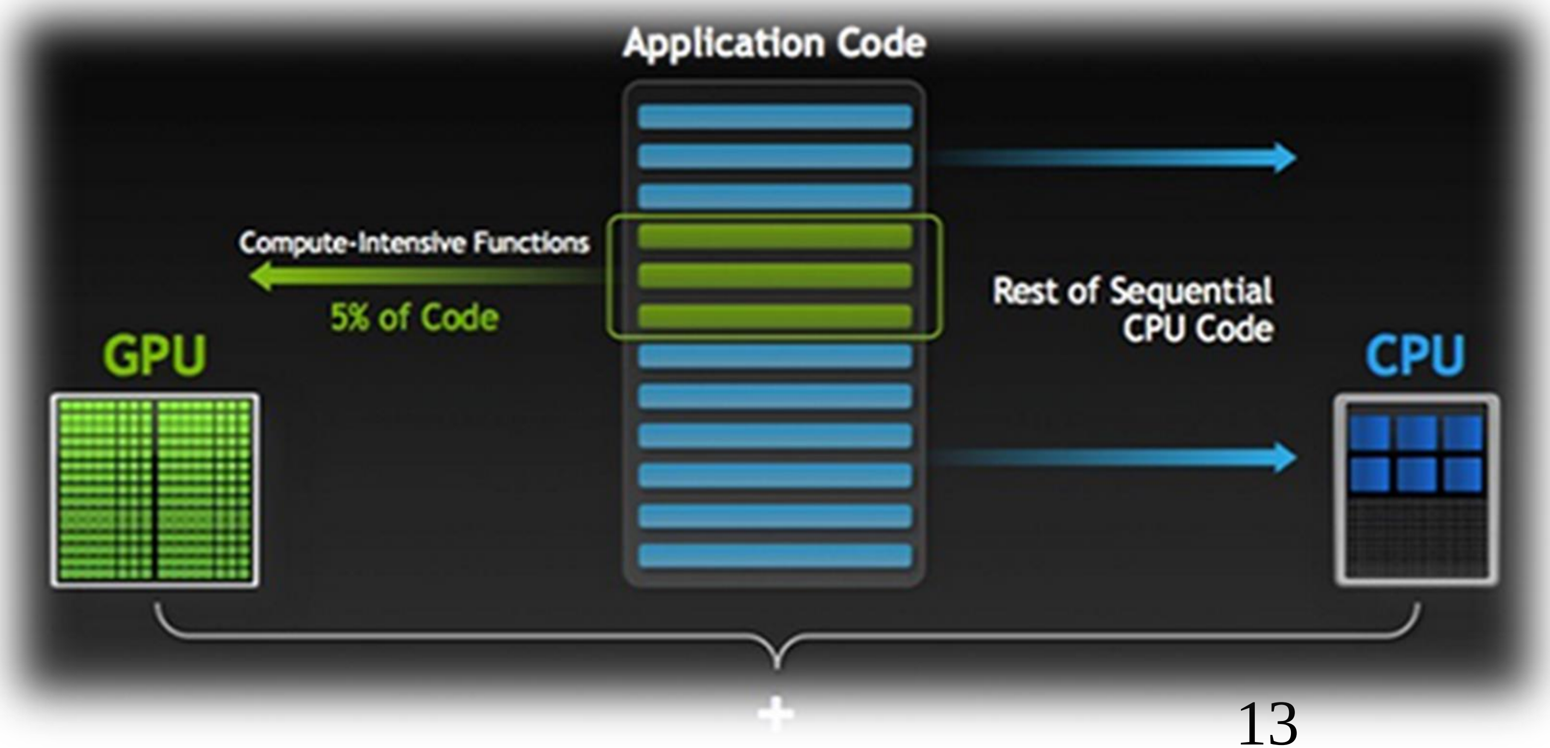
GPU



- GPU wants to have CPU’s features (GPGPU)
- Data parallel processing
- 192 ~ cores (NV Kepler)

Heterogeneous Computing

Acceleration based on data parallelism

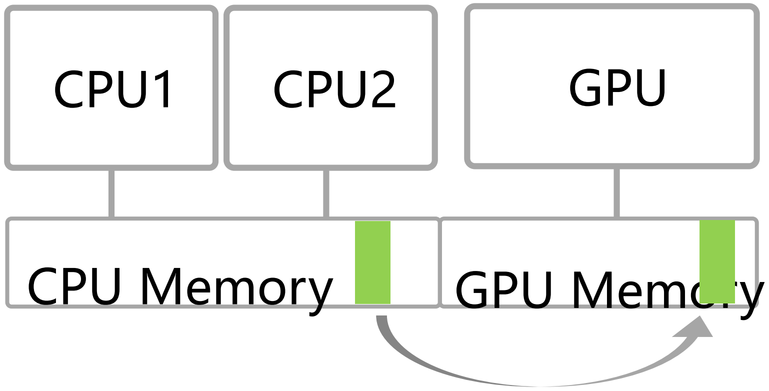


Shared Memory or not

Challenge of Traditional Heterogeneous Systems

Support only dedicated address space

- Require cumbersome copy operations
- Prevent the use of pointer
- DMA copy



Pointer addressable: load/store access

Heterogeneous Systems Architecture

- Unified Memory Address Space (hMMU)

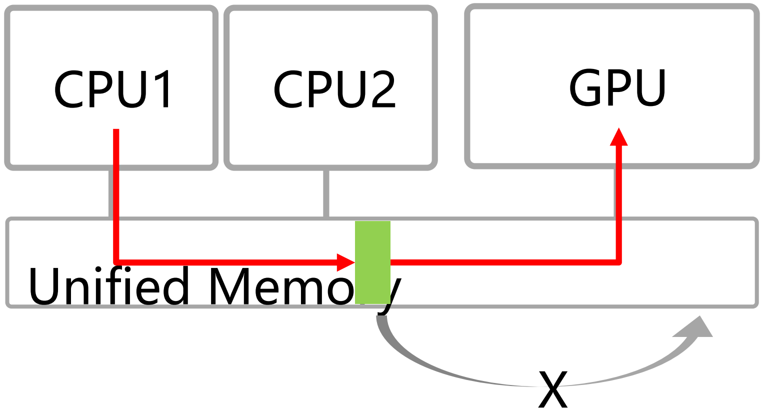
Both CPU and GPU and access the memory directly.

- HSA Intermediate Language

Remove the overhead due to memory copy.

1.Enable Unified address space access

2. Provide a unified intermediate language for high-level languages and different hardware ISA



15

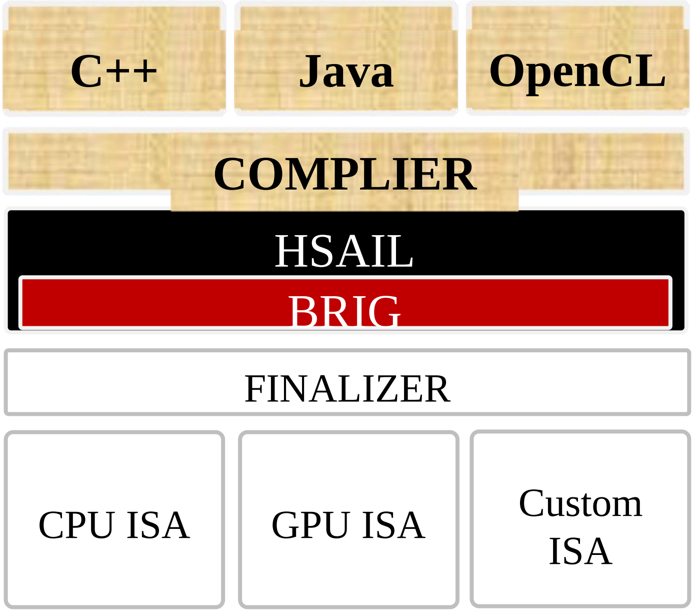
HSA INTERMEDIATE LANGUAGE

HSAIL Overview

- Introduced by HSA Foundation
- A virtual ISA with many operations.
- A textual representation of instructions and a binary format called BRIG.
- ISA, programming model, memory model, machine model, profile…..

HSAIL: Virtual ISA Abstraction

HSAIL: A virtual ISA abstraction for popular programming languages.



Programmers can program in languages they already know, and with the features and tools they expect. Complier that generates HSAIL can be assured that the resulting code will be able to run on different target platforms

The conversion from HSAIL to machine ISA is more of a translation than a complex complier optimization.

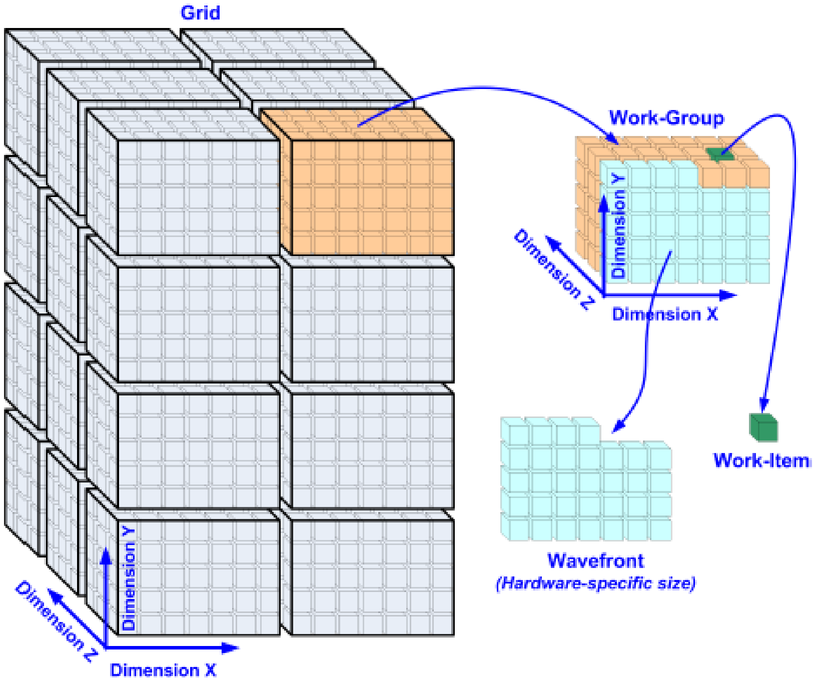
HSAIL: Programming Model

Sequential program expanded in three dimensional parallel execution model.

Grids are divided into one or more work-groups.

Work-groups are composed of work-items.

Work-items in the same work-group can efficiently communicate and synchronize with each other through the "group" memory.



18

HSAIL: Single instruction Multiple Data

Sequential programming run in parallel

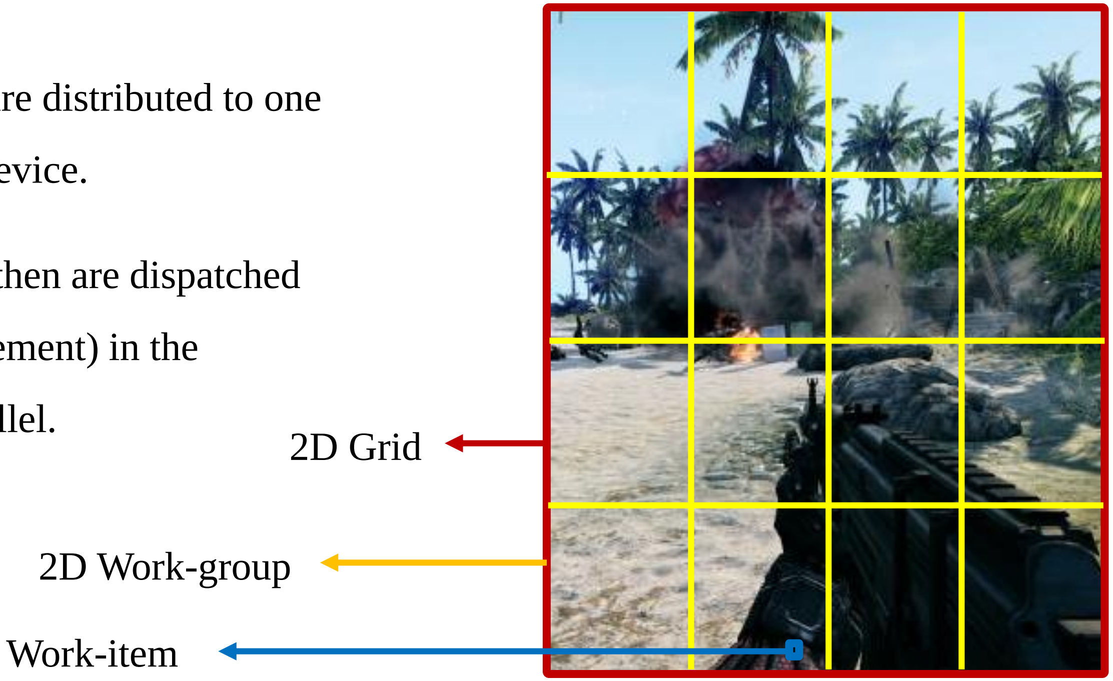
- When the parallel task is dispatched, the dispatch command specifies the number of work-items that should be executed.
- Each work-item has a unique identifier specified with x, y, z coordinate.
- HSAIL contains instructions so that each work-item can determine its unique coordinate and operate on certain part of the data.

HSAIL: Example

Parallel workgroups, parallel work-items

When a grid executes, work-groups are distributed to one or more compute units in the target device.

Work-items in the same work-group then are dispatched to each execution unit (processing element) in the computing unit, and executed in parallel.



2D Grid

2D Work-group

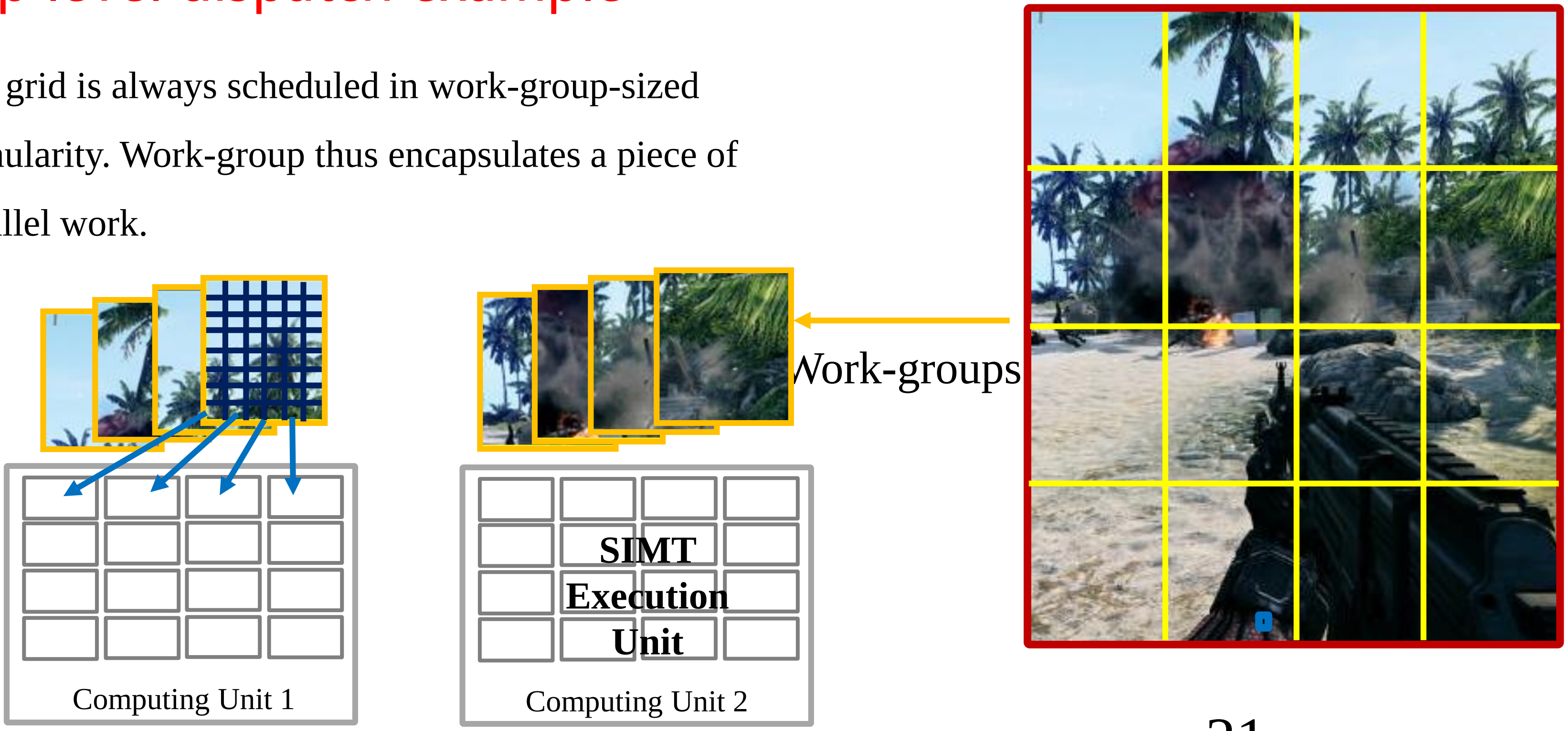
Work-item

http://www.gamesaktuell.de/Crysis-Xbox360-237049/News/Crysis-3D-Support-und-Parallax-Occlusion-Mapping-fuer-die-Konsolen-Version-844757/

HSAIL: Scheduling Granularity (1)

Top level dispatch example

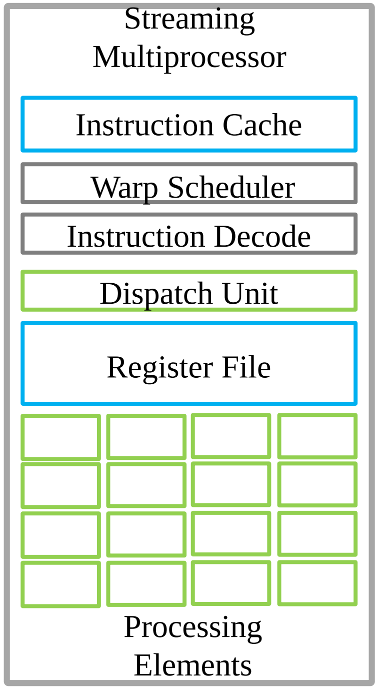
The grid is always scheduled in work-group-sized granularity. Work-group thus encapsulates a piece of parallel work.



HSAIL: Scheduling Granularity (2)

Warp or Wavefront: An independent instruction stream which works on multiple data.

- The wavefront is a hardware concept indicating the number of work-items that are scheduled together.
- Wavefront width is implementation dependent.
- HSAIL also provides cross-lane operations that combine results from several work-items in the work-group. (cross lane transfer of data…)

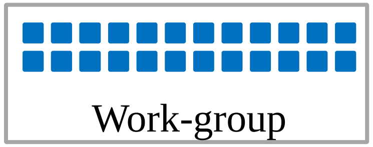


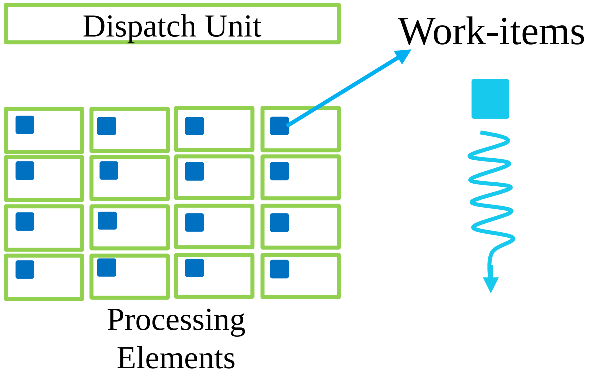
22

HSAIL: A RISC-like ISA, inherent for SIMT Operations

Effective Parallel Processing Comes from Machine.

- Each work-item in the HSA execution model represents a single thread of execution.
- HSAIL thus looks like a sequential program.
- Parallelism is expressed by the grid and the work-groups, which specify how many work-items to run, rather than expressed in the HSAIL code itself.

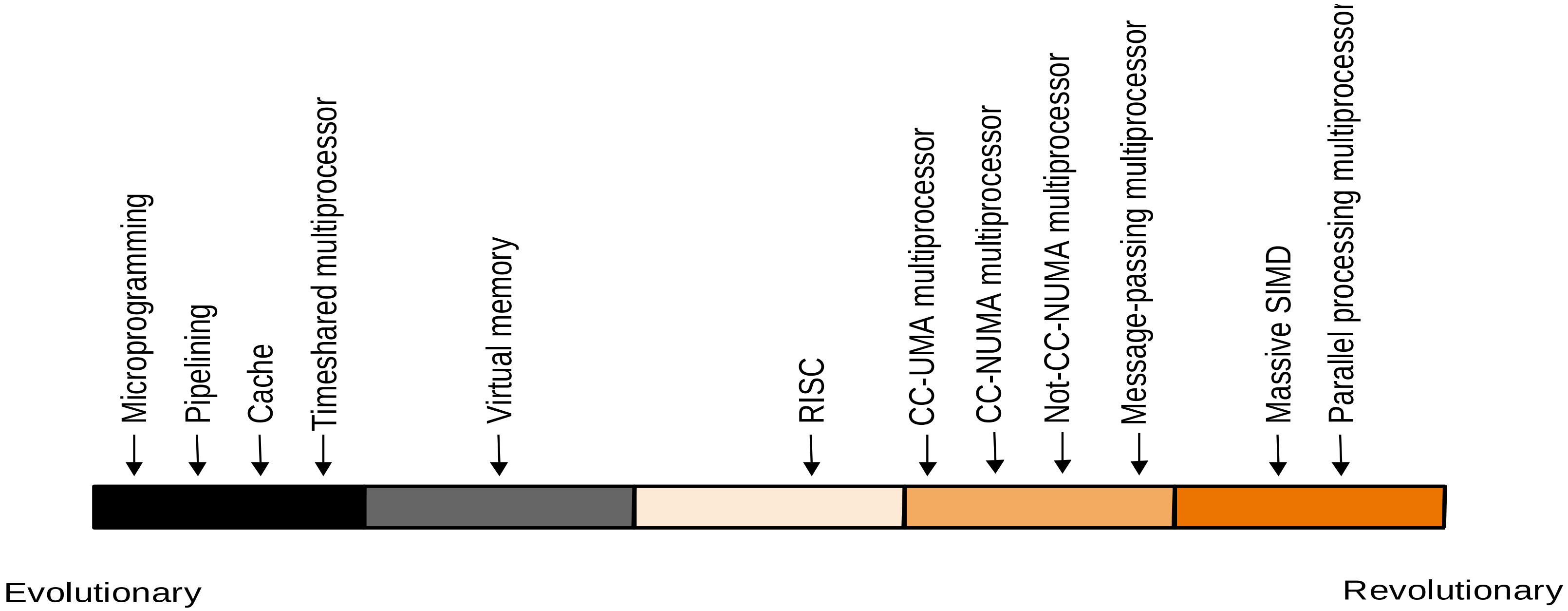




Concluding Remarks

- **Evolution vs. Revolution**

“More often the expense of innovation comes from being too disruptive to computer users”



“Acceptance of hardware ideas requires acceptance by software people; therefore hardware people should learn about software. And if software people want good machines, they must learn more about hardware to be able to communicate with and thereby influence hardware engineers.”