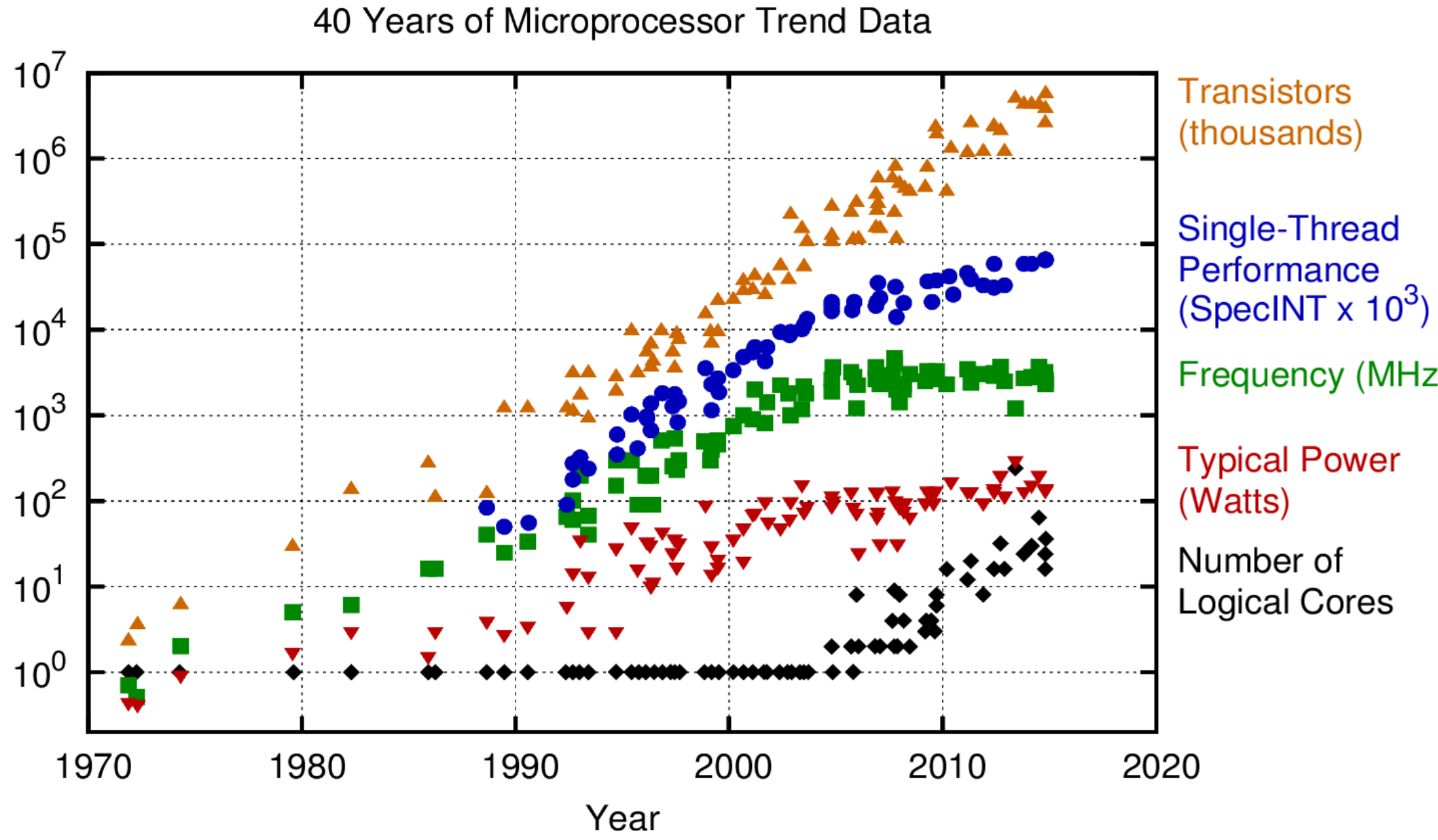
Outline

1)Course introduction, syllabus
2)Technology overview
3)Multiprocessor architecture



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2015 by K. Rupp

Cray 1

- “World's most expensive loveseat”
- Refrigeration in “seat”
- Only 4 types of ICs
- Vector computer

Illiac IV

- Early parallel processor
- 64 Processing Elements
- Only one built
- Custom ECL ICs

5th generation: Parallel Computing Earth simulator

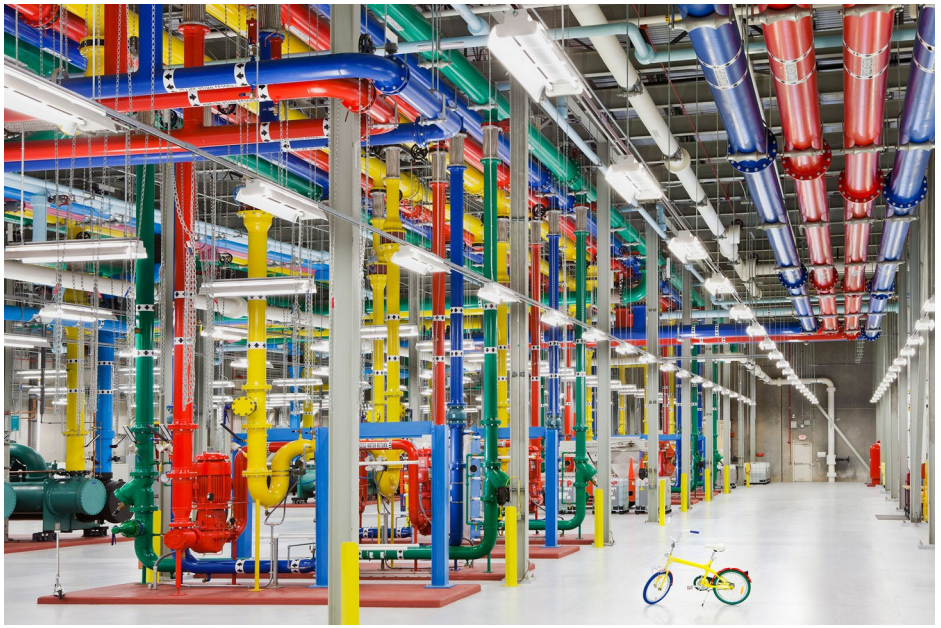
- 640 nodes
- Vector PE
- 1 node = 8 PE + 10 Memory modules
- World leader from 2002-2004

IBM Blue Gene

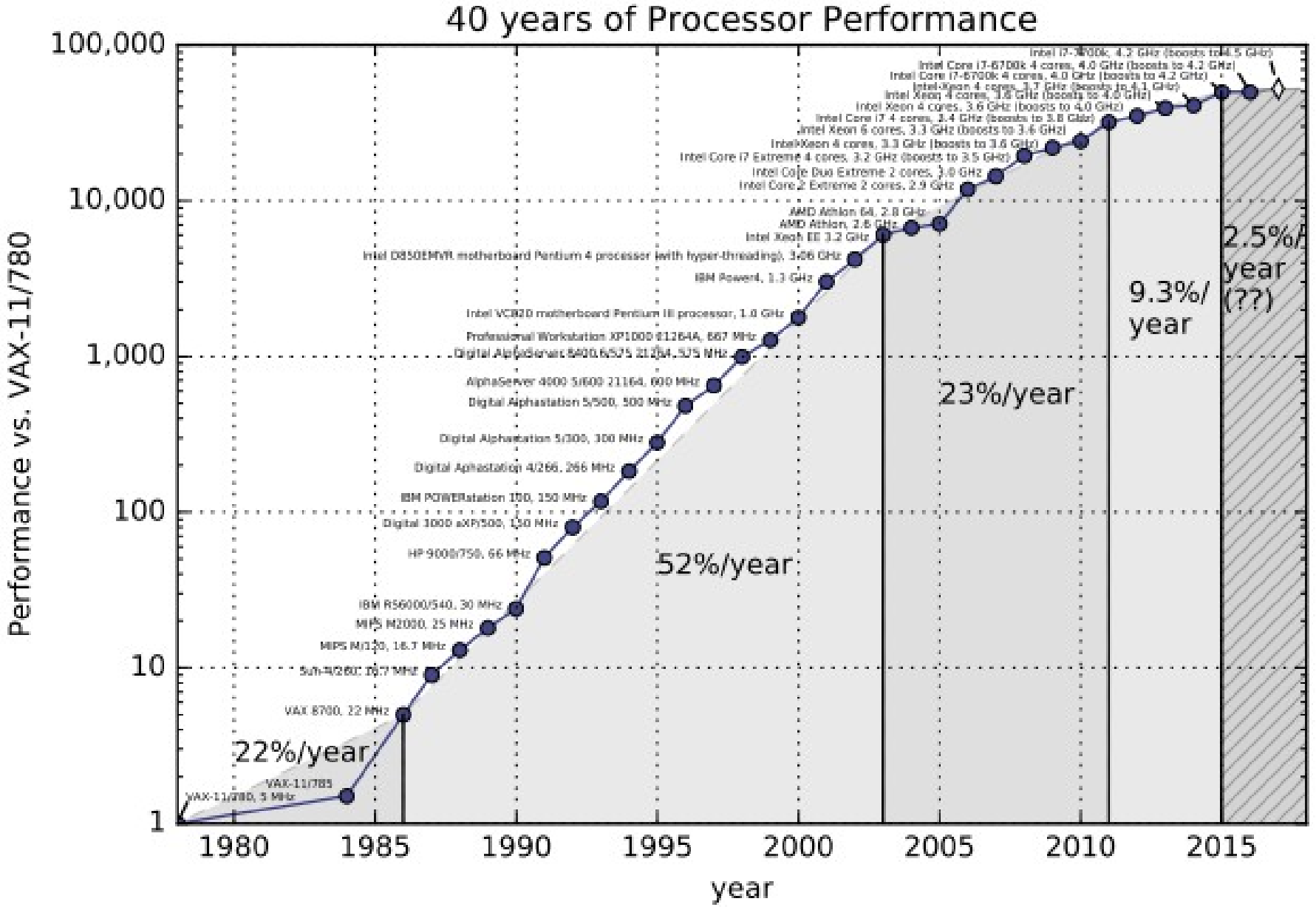
- Low power PowerPC cores + floating point
- Up to 65K nodes
- Torus interconnect

Nvidia Tesla (GPU Cluster)

Google Warehouse



Single-thread Processor Performance



Computer Design Upheaval

- Most of last 50 years, Moore's Law ruled
  - Technology scaling allowed continual performance/energy improvements without changing software model
- Last decade, technology scaling slowed/stopped
  - Dennard (voltage) scaling over (supply voltage ~fixed)
  - Moore's Law (cost/transistor) over?
  - No competitive replacement for CMOS anytime soon
  - Energy efficiency constrains everything
- No "free lunch" for software developers, must consider:
  - Parallel systems
  - Heterogeneous systems

Current Trends

- Mobile (smartphone/tablet)
  - >1 billion sold/year
  - Market dominated by ARM-ISA-compatible general-purpose processor in system-on-a-chip (SoC)
  - Plus sea of custom accelerators (radio, image, video, graphics, audio, motion, location, security, etc.)
- Warehouse-Scale Computers (WSCs)
  - 100,000's cores per warehouse
  - Market dominated by x86-compatible server chips
  - Dedicated apps, plus cloud hosting of virtual machines
  - Now seeing increasing use of GPUs, FPGAs, custom hardware to accelerate workloads
- Embedded computing
  - Wired/wireless network infrastructure, printers
  - Consumer TV/Music/Games/Automotive/Camera/MP3
  - Internet of Things!

Course outline

- Flynn’s Taxonomy
- Processor Parallelism such as pipelines, ILP, and superscalar
- Multiprocessor parallelism with threading
- Synchronization, communication and coordination.
- Clusters with message passing
- Interconnection networks
- Accelerator design
- Data parallelism

Flynn Taxonomy, 1966

		Data Streams	
		Single	Multiple
Instruction Streams	Single	SISD: Intel Pentium 4	SIMD: SSE instructions of x86
	Multiple	MISD: No examples today	MIMD: Intel Xeon e5345 (Clovertown)

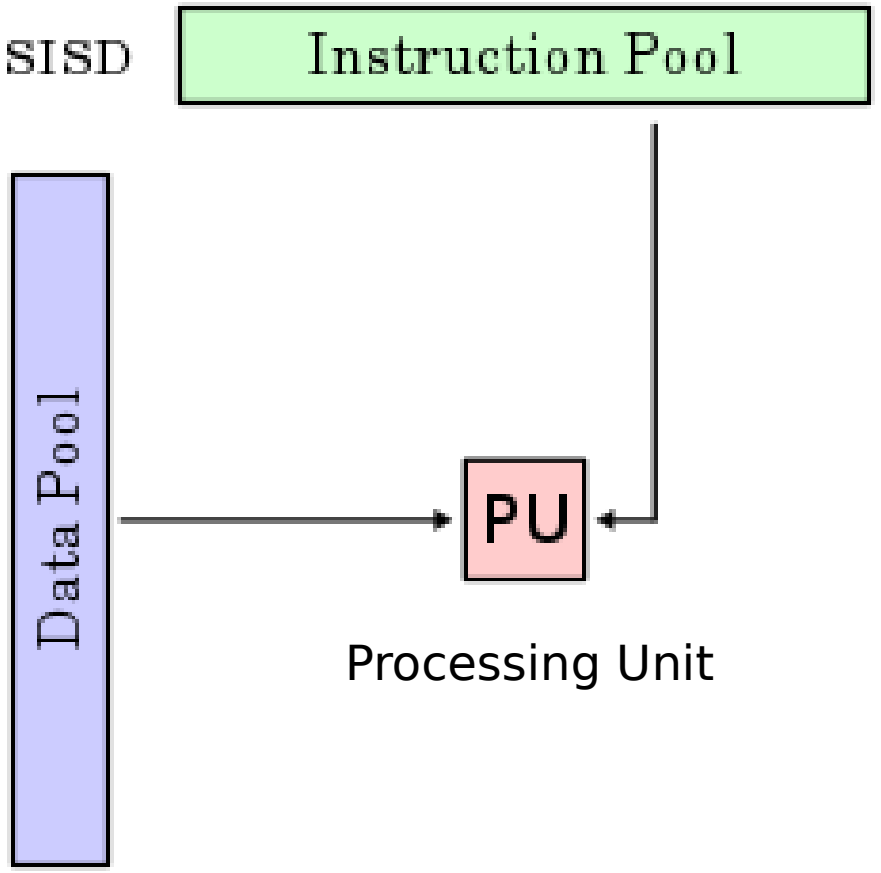
- SIMD and MIMD are currently the most common parallelism in architectures – usually both in same system!
- Most common parallel processing programming style: Single Program Multiple Data (“SPMD”)
  - Single program that runs on all processors of a MIMD
  - Cross-processor execution coordination using synchronization primitives
  - Found in GPUs

Flynn's Taxonomy

Mike Flynn, "Very High-Speed Computing Systems," Proc. of IEEE, 1966

- **S**ingle **I**nstruction **S**ingle **D**ata
- **S**ingle **I**nstruction **M**ultiple **D**ata
  - Array Processor
  - Vector Processor
  - GPU
- **M**ultiple **I**nstruction **S**ingle **D**ata
  - Closest form: systolic array processor, streaming processor
- **M**ultiple **I**nstruction **M**ultiple **D**ata
  - Multiprocessor
  - Multithreaded processor

Single-Instruction/Single-Data Stream (SISD)

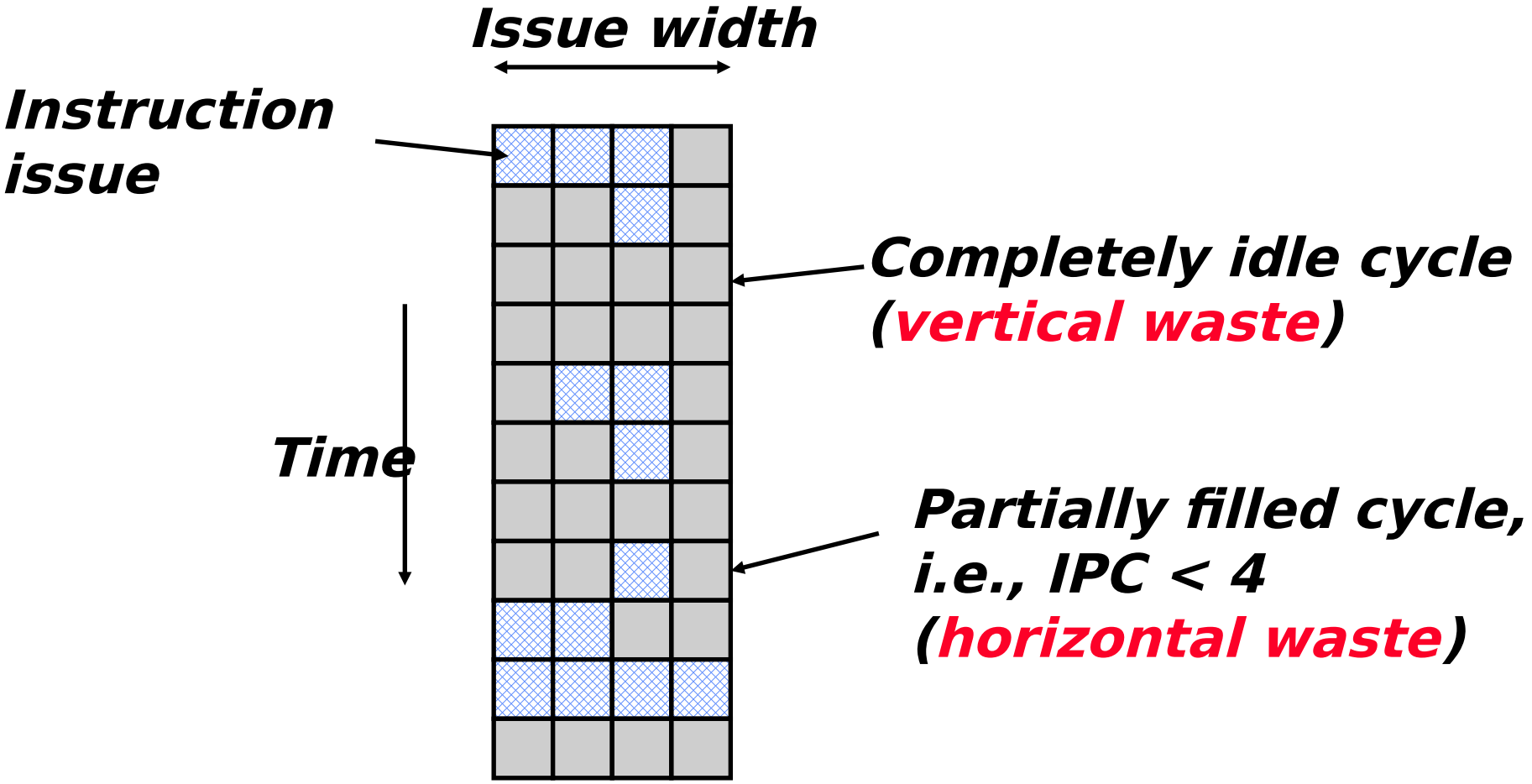


- Sequential computer that exploits no parallelism in either the instruction or data streams. Examples of SISD architecture are traditional uniprocessor machines
  - E.g. our trusted RISC-V pipeline

Instruction Level Parallelism (ILP)

- Improve processor performance by having multiple processor components or functional units simultaneously executing instructions.
- Pipelining - functional units are arranged in *stages*.
- Multiple issue - multiple instructions *might* be simultaneously initiated with resource duplication.
  - VLIW – Very long instruction word - functional units are scheduled at compile time (static scheduling).
  - Superscalar - functional units are scheduled at run-time (dynamic scheduling)

Superscalar Machine Efficiency



Threads

- Provides a means to continue doing useful work when the currently executing task has stalled (ex. wait for long memory latency).
- Lighter weight than multi-tasking because context switching is usually more costly than thread-switching.
- Software-based multi-threading (Posix Threads)
  - Hardware still traps on long-latency processes
  - Software handles thread context switch
  - Issues with overhead and “multi-level” control
- Hardware-based multi-threading
  - User defines threads (or kernel)
  - Hardware “helps” in context switch
  - Ex: IBM Power5, Pentium-4, Sun

Multithreading

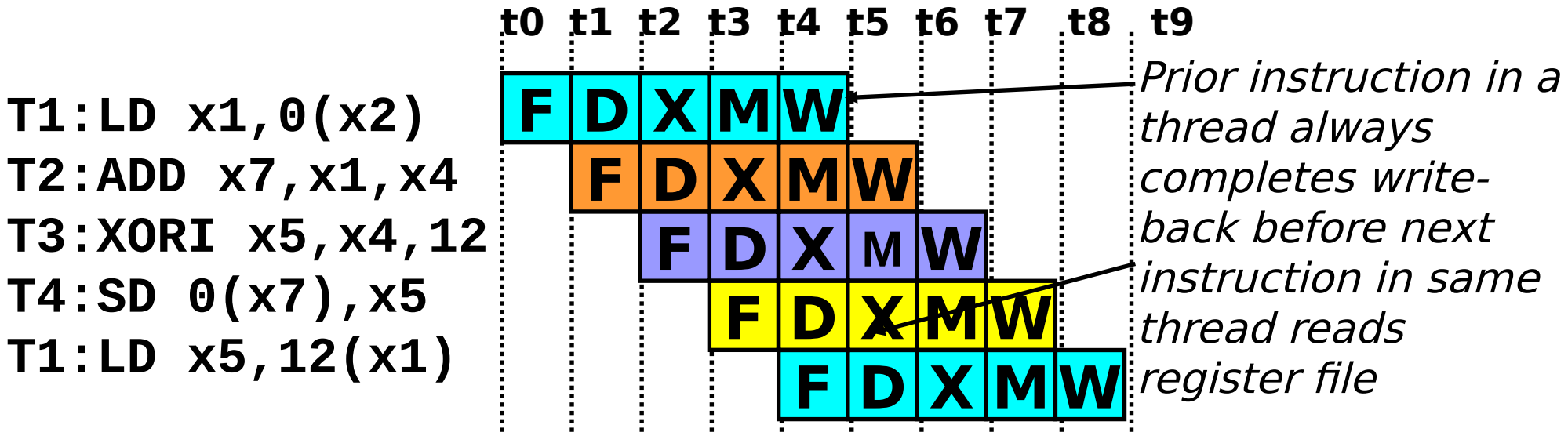
- Difficult to continue to extract instruction-level parallelism (ILP) from a single sequential thread of control
- Many workloads can make use of thread-level parallelism (TLP)
  - TLP from multiprogramming (run independent sequential jobs)
  - TLP from multithreaded applications (run one job faster using parallel threads)
- Multithreading uses TLP to improve utilization of a single processor

Multithreading

How can we guarantee no dependencies between instructions in a pipeline?

One way is to interleave execution of instructions from different program threads on same pipeline

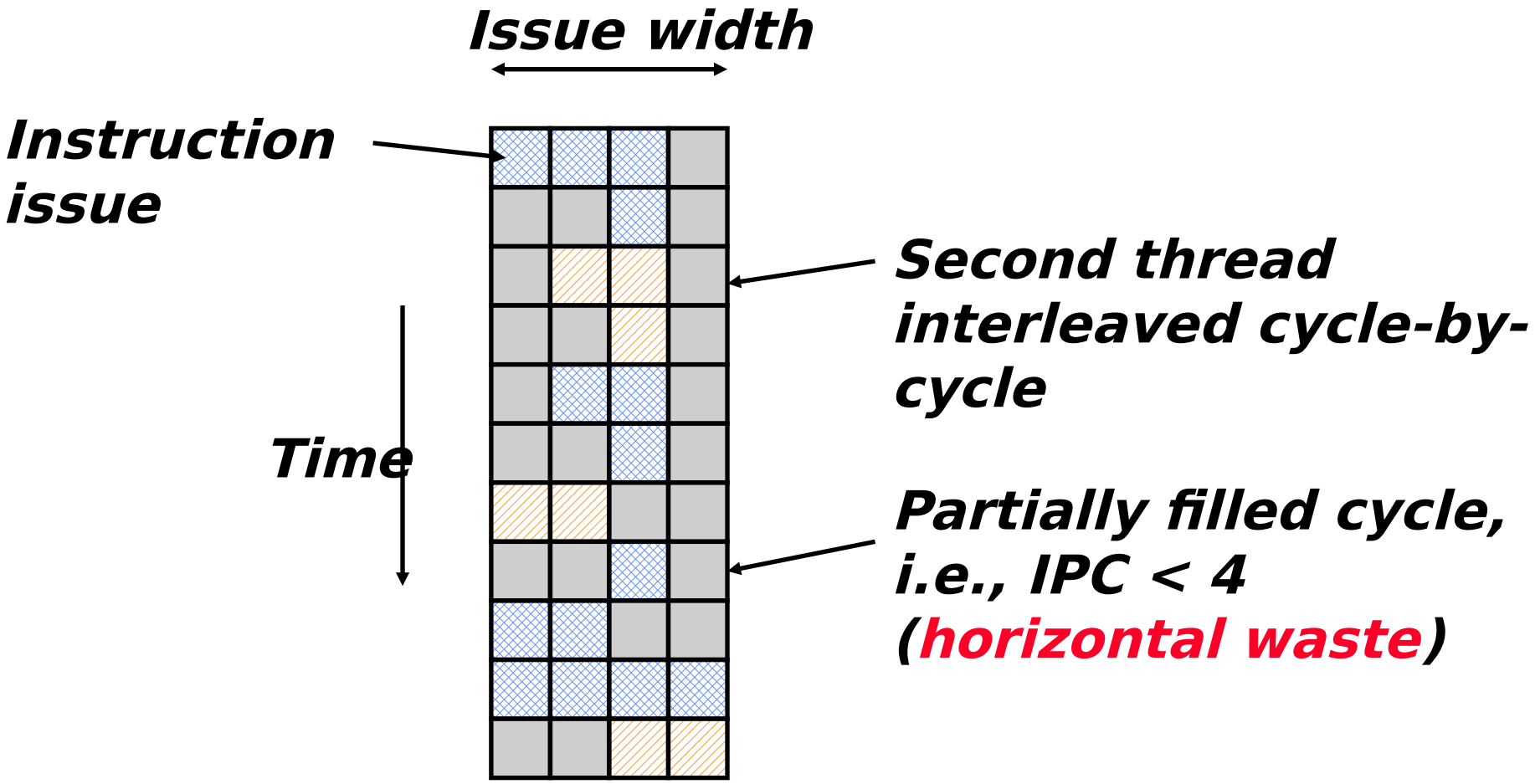
*Interleave 4 threads, T1-T4, on **non-bypassed** 5-stage pipe*



Multithreading Costs

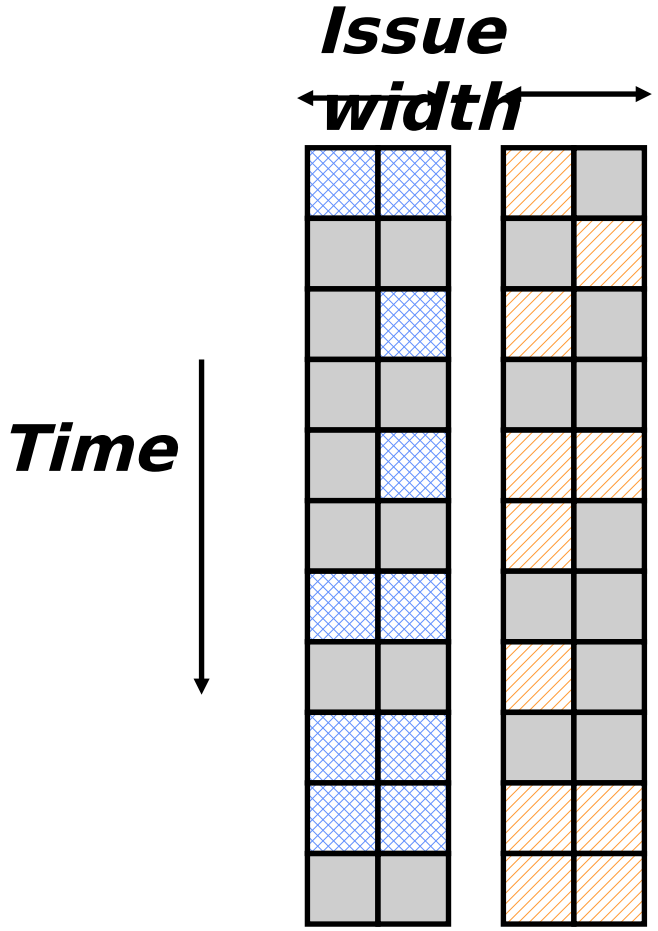
- Each thread requires its own user state
  - PC
  - GPRs
- Also, needs its own system state
  - Virtual-memory page-table-base register
  - Exception-handling registers
- *Other overheads:*
  - Additional cache/TLB conflicts from competing threads (or add larger cache/TLB capacity)
  - More OS overhead to schedule more threads (where do all these threads come from?)

Vertical Multithreading



- Cycle-by-cycle interleaving removes vertical waste, but leaves some horizontal waste

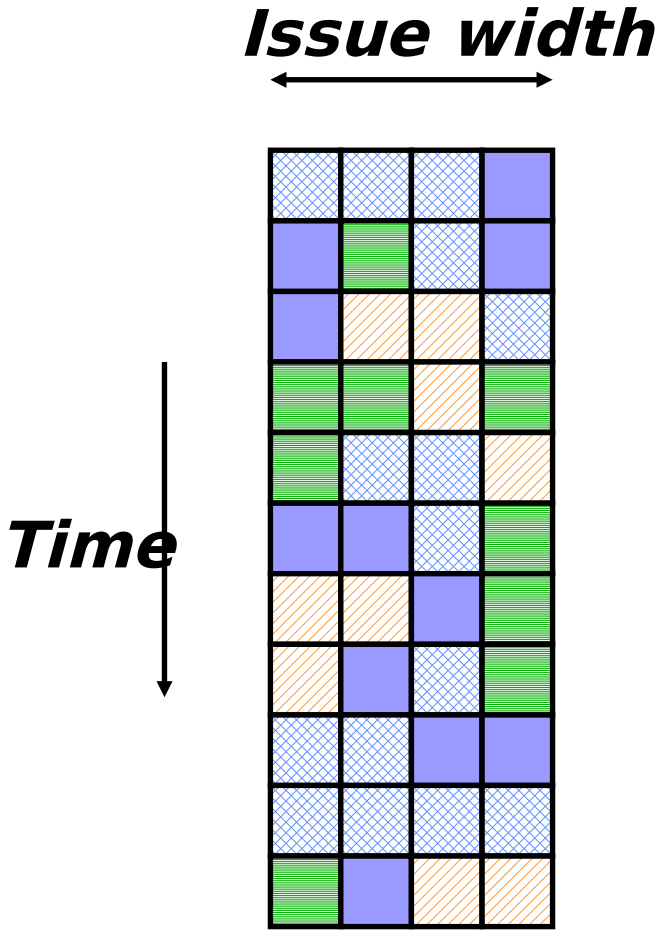
Chip Multiprocessing (CMP)



- What is the effect of splitting into multiple processors?
  - reduces horizontal waste,
  - leaves some vertical waste, and
  - puts upper limit on peak throughput of each thread.

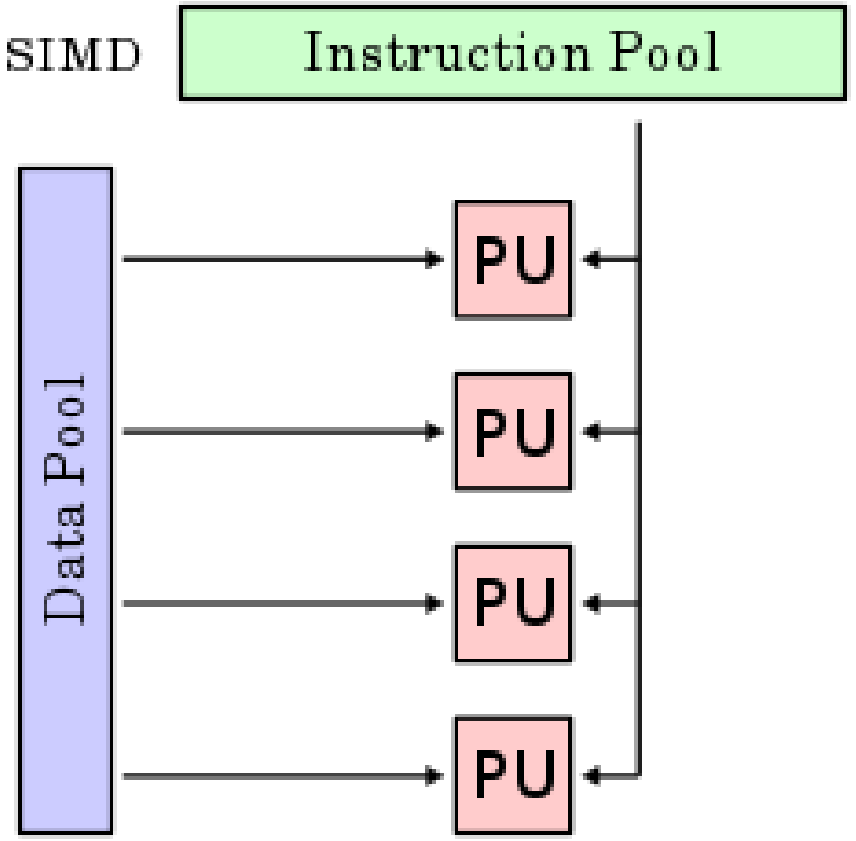
Ideal Superscalar Multithreading

[Tullsen, Eggers, Levy, UW, 1995]



- Interleave multiple threads to multiple issue slots with no restrictions

Single-Instruction/Multiple-Data Stream (SIMD or “sim-dee”)



- SIMD computer exploits multiple data streams against a single instruction stream to operations that may be naturally parallelized, e.g., Intel SIMD instruction extensions or NVIDIA Graphics Processing Unit (GPU)

Lucasfilm Chap

- SIMD graphics in 1984!
- 64 MIPS
- Microcoded control
- 16 bit arithmetic
- Tessellated access to memory via Crossbar (pixel, component, broadcast, indexed)
- “run flag” condition code testing

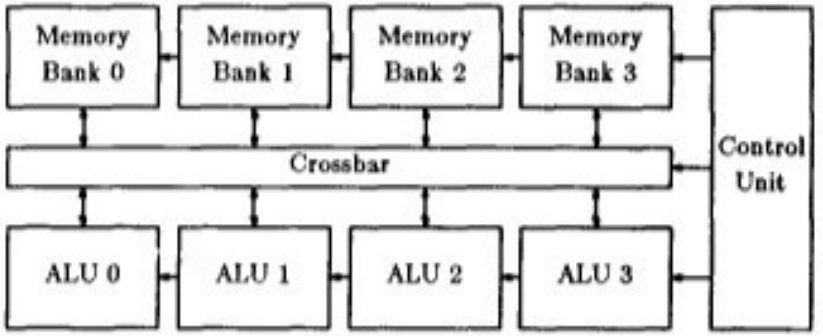


Figure 2: Chap Block Diagram

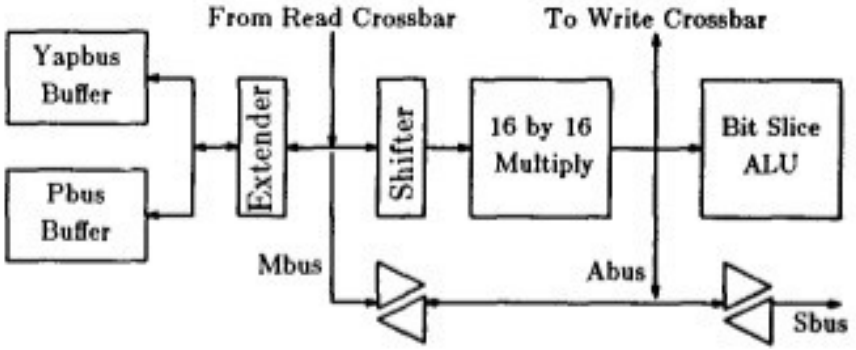


Figure 3: Chap Processing Element

SIMD drawbacks

- All ALUs are required to execute the same instruction, or remain idle.
- In classic design, they must also operate synchronously.
- The ALUs have no instruction storage.
- Efficient for large data parallel problems, but not other types of more complex parallel problems

Vector machines (SIMD)

- Operate on arrays or vectors of data while conventional CPU's operate on individual data elements or scalars.
  - Also known as *array processors*
- Vector registers.
  - Capable of storing a vector of operands and operating simultaneously on their contents.
- Vectorized and pipelined functional units.
  - The same operation is applied to each element in the vector (or pairs of elements).

Vector Supercomputers

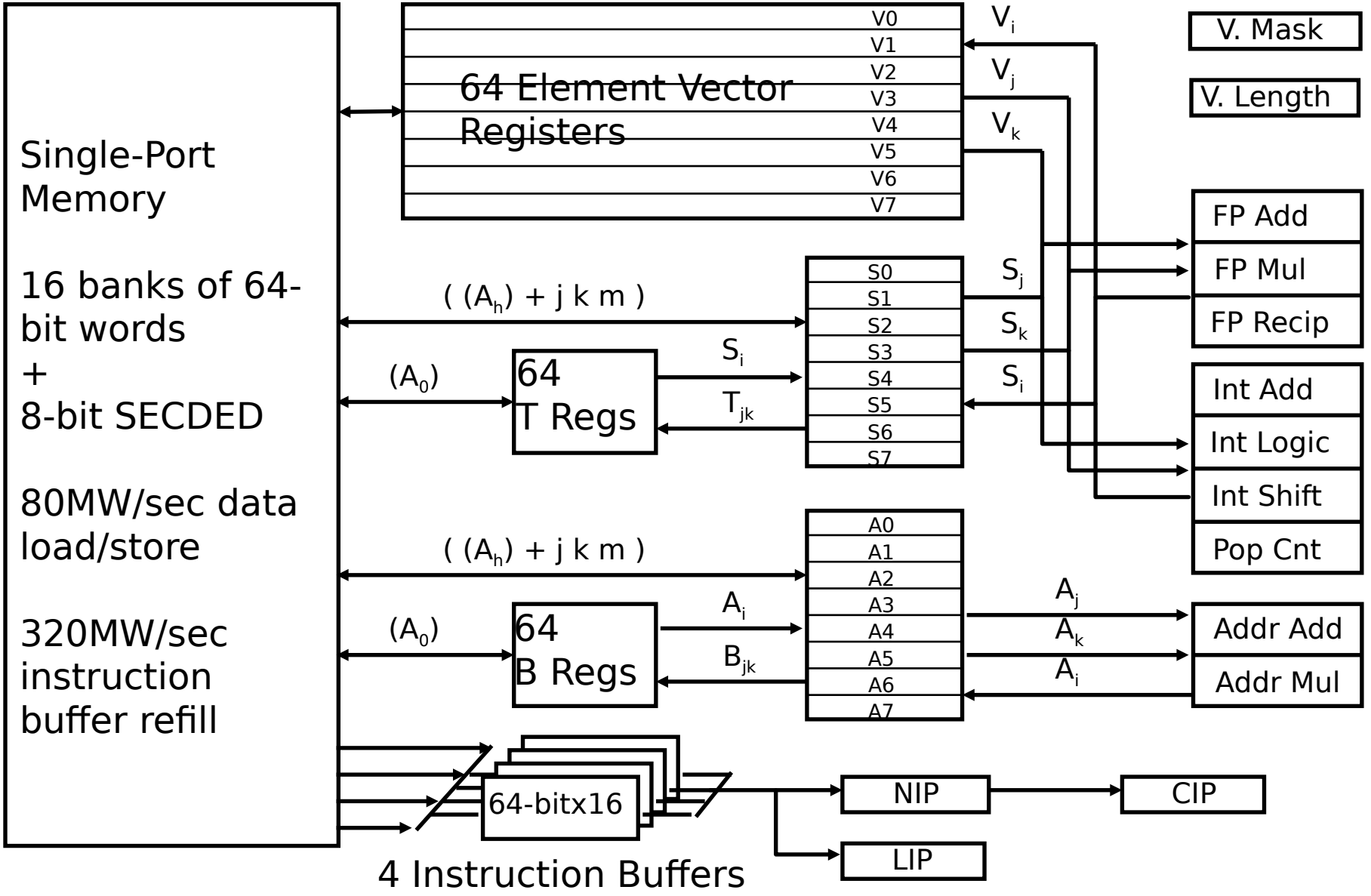

[©Cray Research, 1976]

- Epitomized by Cray-1, 1976:
- Scalar Unit
  - Load/Store Architecture
- Vector Extension
  - Vector Registers
  - Vector Instructions
- Implementation
  - Hardwired Control
  - Highly Pipelined Functional Units
  - Interleaved Memory System
  - No Data Caches
  - No Virtual Memory

A few Cray-1 facts

- 5 tons and 115 kilowatts
- 1662 modules, 3 different chips
- 1969 Gore invents GoreTex for cable insulation for Cray, *not* sportswear
- Case of Leinenkugel (Chippewa Falls brewery) included in every order
- Cray thought in boolean equations, not schematics

Cray-1 (1976)

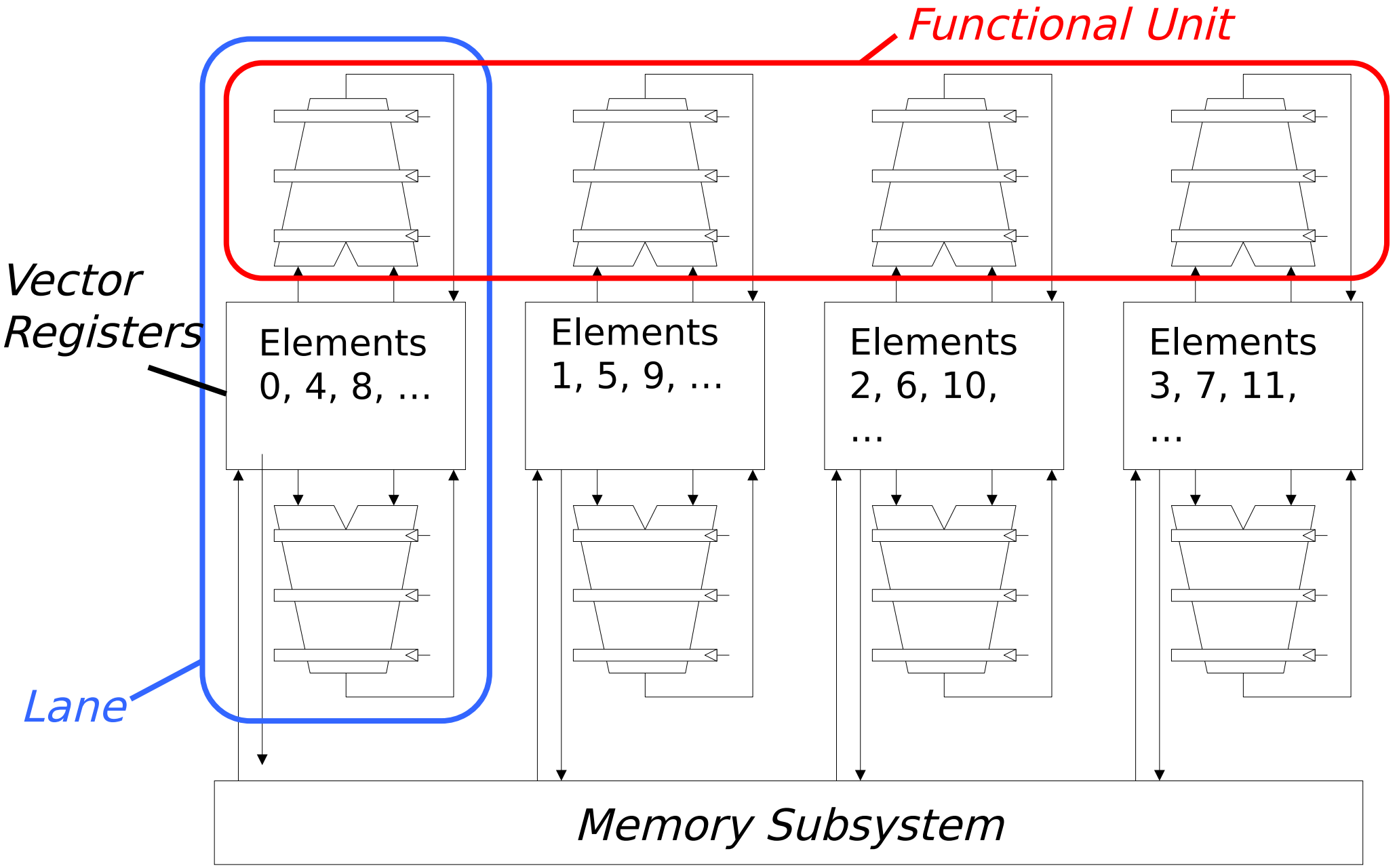


memory bank cycle 50 ns *processor cycle* 12.5 ns (80MHz)

Vector Instruction Set Advantages

- Compact
  - one short instruction encodes N operations
- Expressive, tells hardware that these N operations:
  - are independent
  - use the same functional unit
  - access disjoint registers
  - access registers in same pattern as previous instructions
  - access a contiguous block of memory (unit-stride load/store)
  - access memory in a known pattern (strided load/store)
- Scalable
  - can run same code on more parallel pipelines (lanes)

Vector Unit Structure



Vector machines: Pros and Cons

- Pros
  - Fast and Easy to use.
  - Vectorizing compilers are good at identifying code to exploit.
  - Compilers also can provide information about code that cannot be vectorized.
  - High memory bandwidth.
  - Uses every item in a cache line.
- Cons
  - They don’t handle irregular data structures as well as other parallel architectures.
  - A very finite limit to their ability to handle ever larger problems. (scalability)

VLIW Advantages & Disadvantages (versus Superscalar)

Advantages

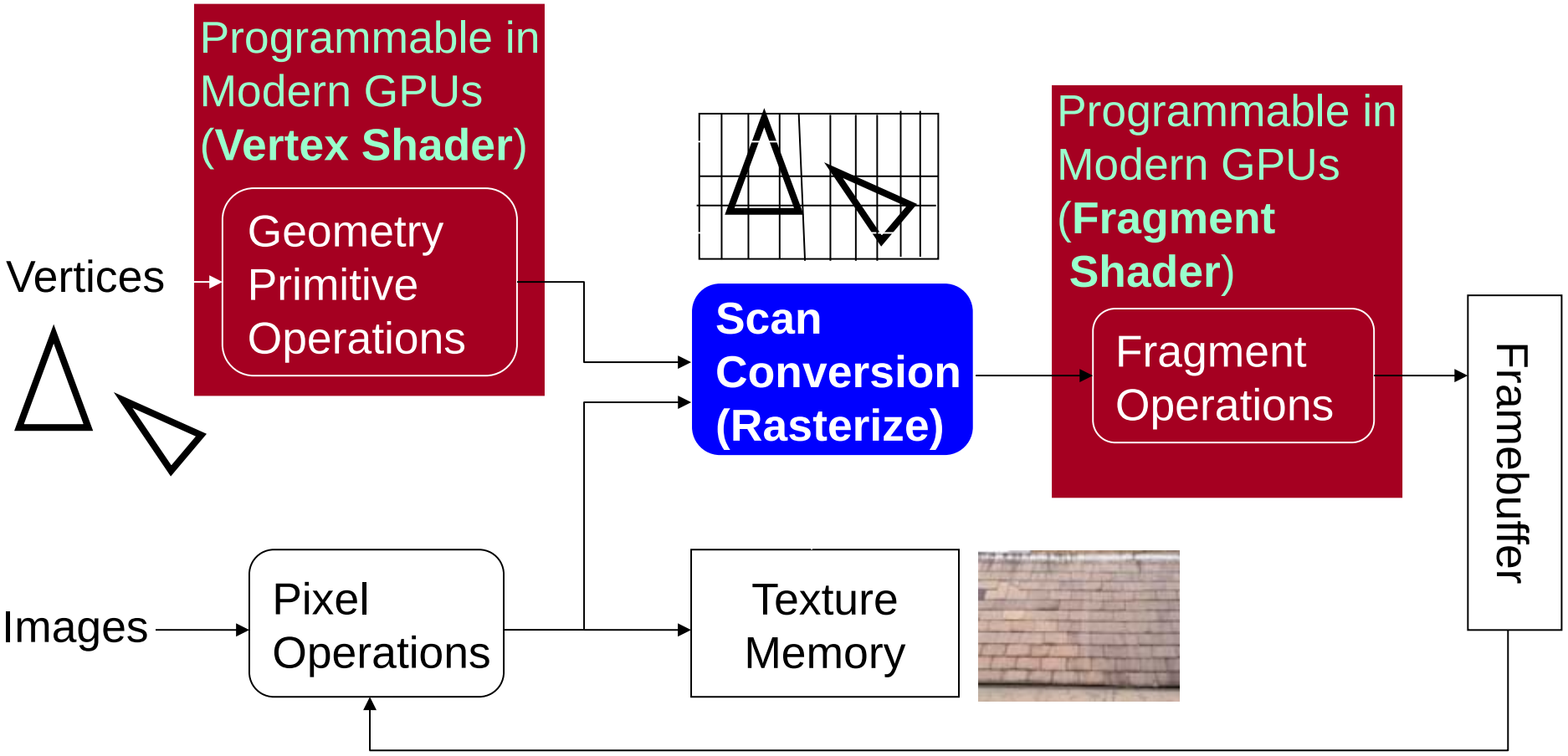
- Simpler hardware (potentially less power hungry)
- E.g. Transmeta Crusoe processor was a VLIW that converted x86 code to VLIW code using dynamic binary translation
- Many processors aimed specifically at digital signal processing (DSP) are VLIW
  - DSP applications usually contain lots of instruction level parallelism in loops that execute many times
- Potentially more scalable
  - Allow more instructions per VLIW bundle and add more functional units

VLIW Advantages & Disadvantages (versus Superscalar)

Disadvantages

- Programmer/compiler complexity and longer compilation times
  - Deep pipelines and long latencies can be confusing (making peak performance elusive)
- Lock step operation, i.e., on hazard all future issues stall until hazard is resolved (hence need for predication)
- Object (binary) code incompatibility
- Needs lots of program memory bandwidth
- Code bloat
  - Nops are a waste of program memory space
  - Speculative code needs additional fix-up code to deal with cases where speculation has gone wrong
  - Register renaming needs free registers, so VLIWs usually have more programmer-visible registers than other architectures
  - Loop unrolling (and software pipelining) to expose more ILP uses more program memory space

GPU Programmable Shaders



Traditional Approach: Fixed function pipeline (state machine)

New Development (2003-): Programmable pipeline