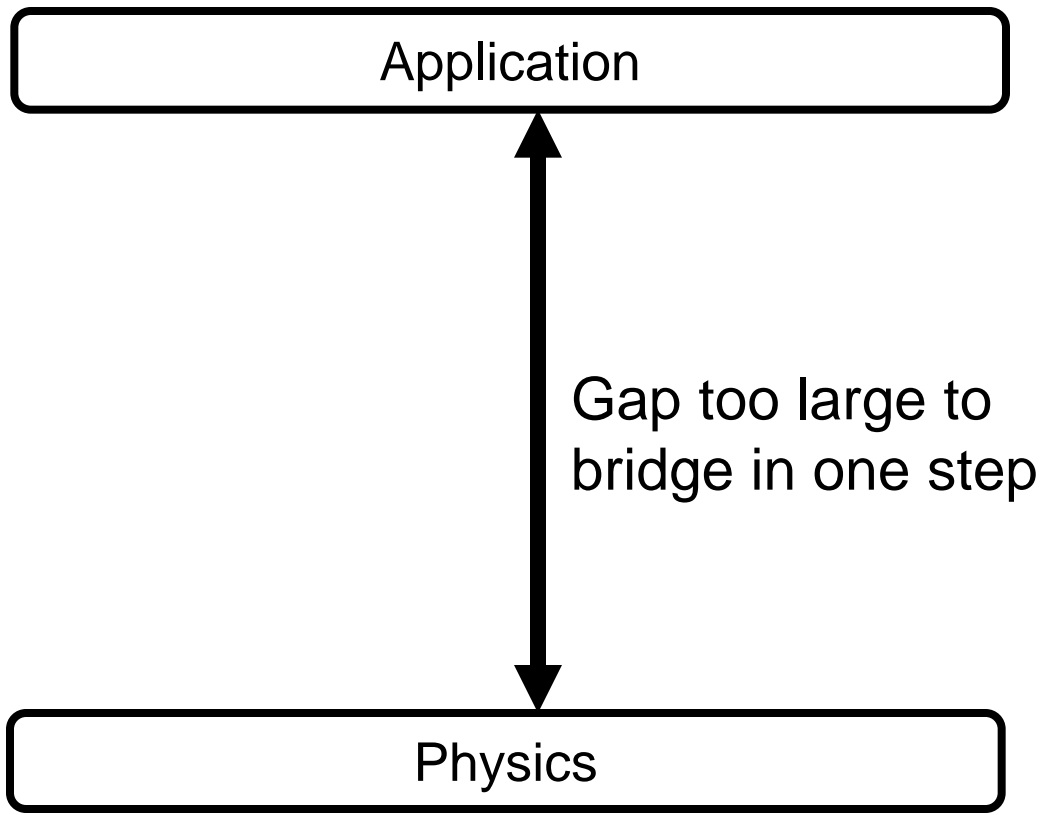


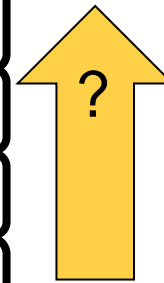
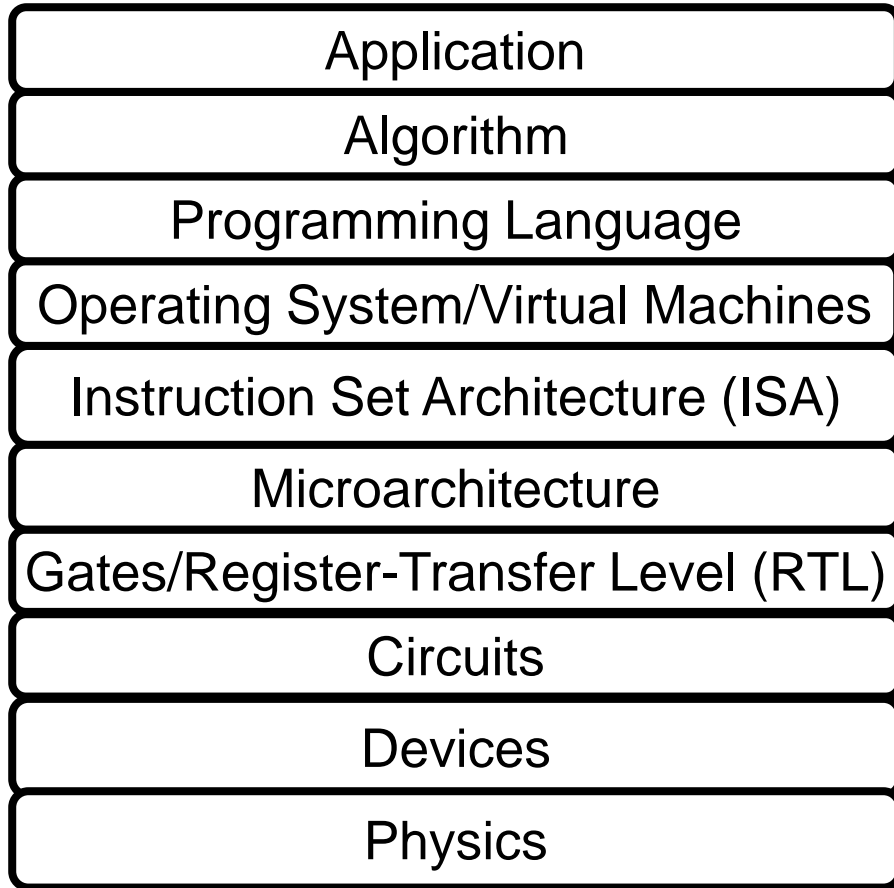
Architectures Beyond Moore

Krste Asanovic

The Parallel Computing Laboratory
EECS Dept., UC Berkeley

Beyond Moore's Law Workshop
SRI, Menlo Park, CA
February 10, 2011





If devices change, how much else needs to change?
(& do we need all these layers?)

A Seductive Example

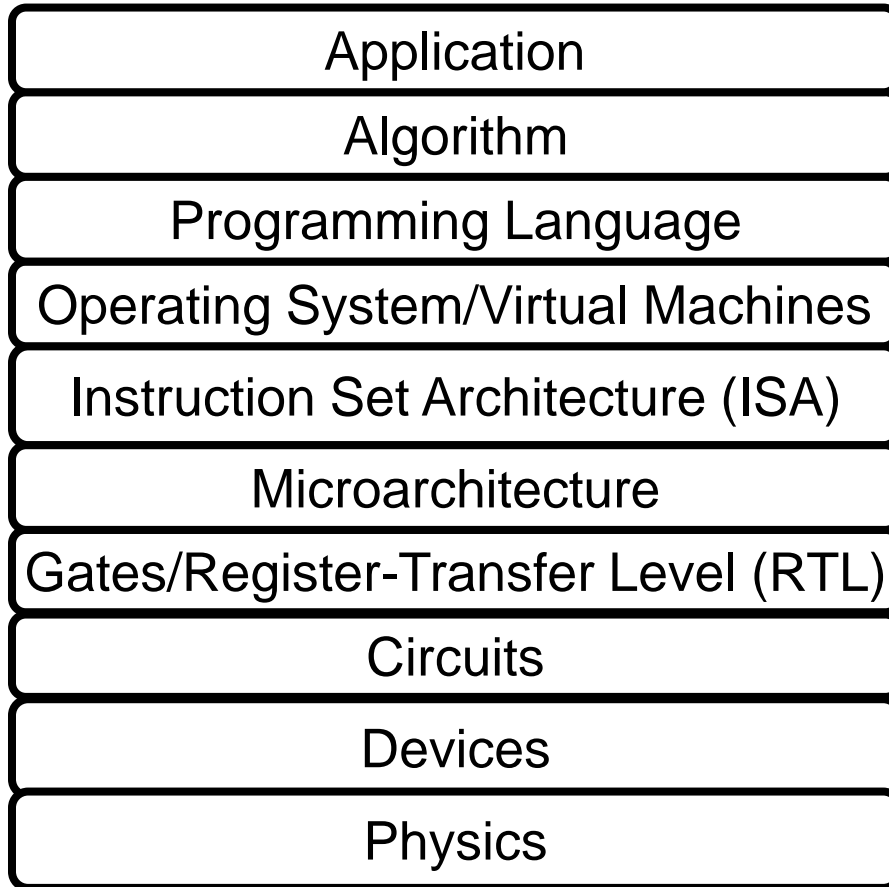
Application



Magnetic compass



Physics



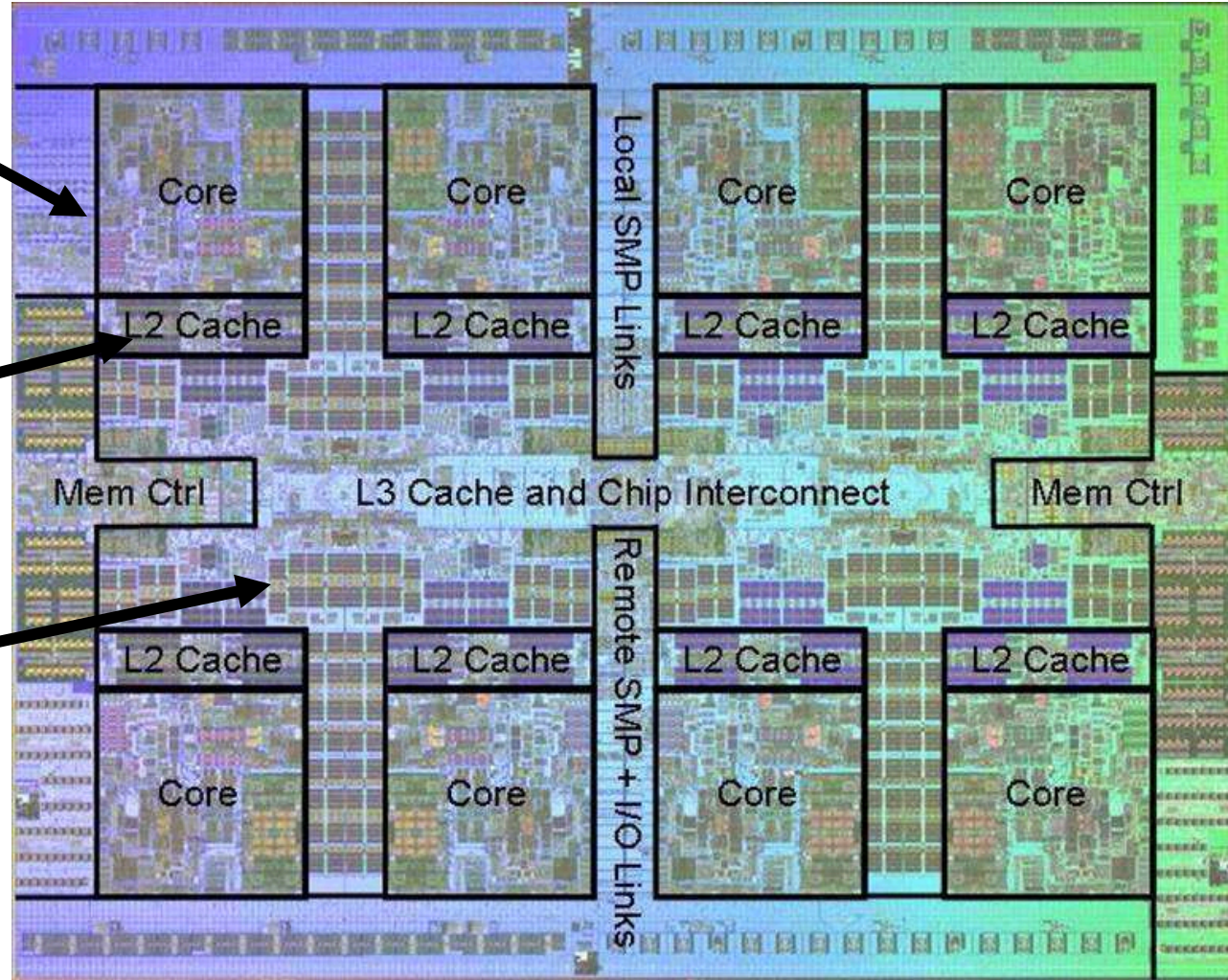
It's not this complicated because we want a compass.
It's this complicated because we want augmented reality applications.

Power 7 [IBM 2009]

32KB L1 I\$/core
32KB L1 D\$/core
3-cycle latency

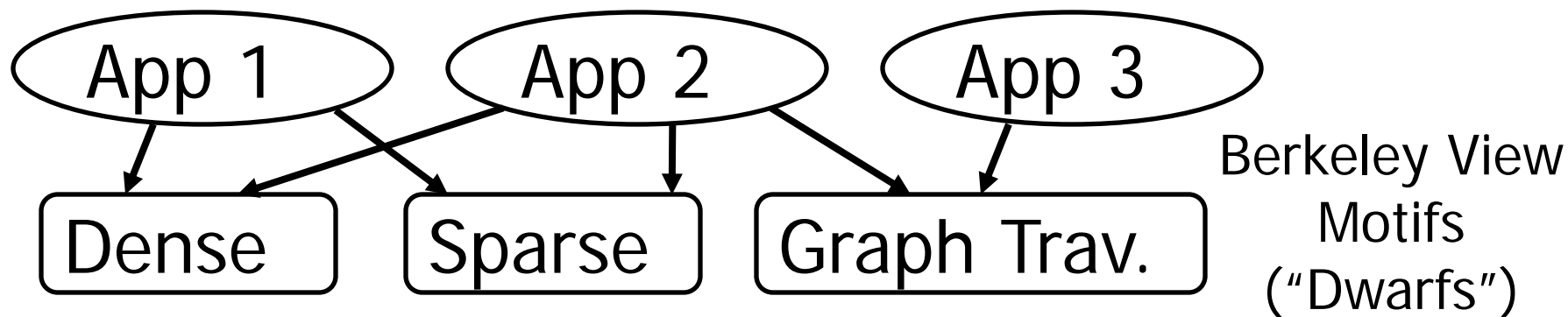
256KB Unified L2\$/core
8-cycle latency

32MB Unified Shared L3\$
Embedded DRAM
25-cycle latency to local
slice




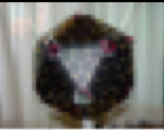



- ❖ Programmable, reliable, digital computation that supports a large software base
- ❖ “Von-Neumann” architectures, aka instruction-stream processors, are the best way we know how to do this
 - Time-multiplexing essential to efficiency, and instruction streams are how to manage complexity of controlling time-multiplexing
- ❖ Not to say that there aren’t many interesting fixed-function information processing systems
 - Sensors + analog processing for real-world I/O
 - Hardwired digital accelerators
 - more complex ones *are* instruction-stream processors

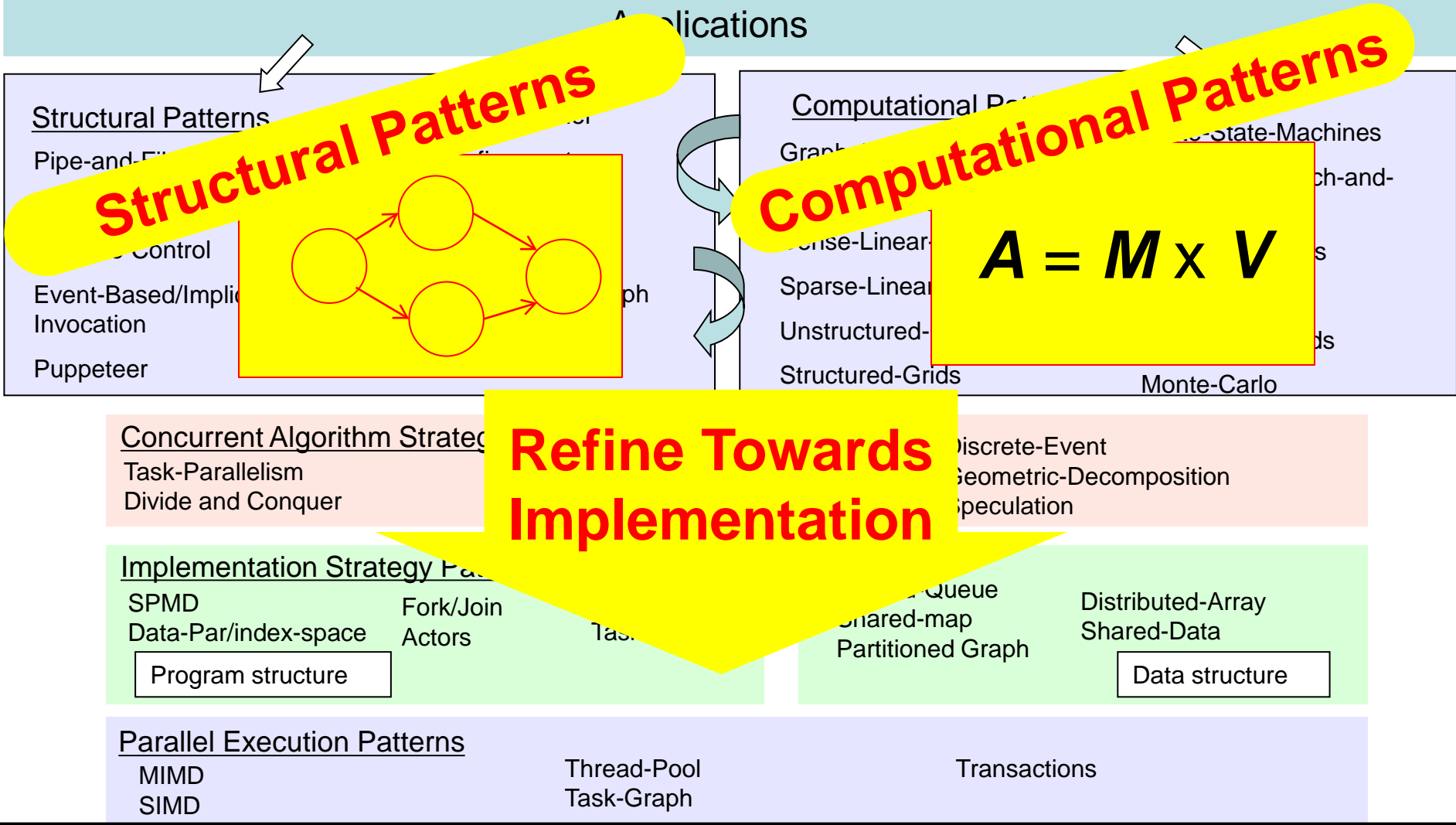
What do applications need?



How do compelling apps relate to 12 motifs?

| | Embed | SPEC | DB | Games | ML | CAD | HPC |  Health |  Image |  Speech |  Music |  Browser |
|----------------------|------------|------------|-------------|------------|-------------|------------|------------|--|---|--|---|---|
| 1 Finite State Mach. | Red | Red | Red | Yellow | Yellow | Yellow | Light Blue | Light Blue | Light Blue | Light Blue | Light Blue | Light Blue |
| 2 Circuits | Red | Light Blue | Light Green | Light Blue | Light Green | Light Blue | Light Blue | Light Blue | Light Blue | Light Blue | Light Blue | Light Blue |
| 3 Graph Algorithms | Red | Yellow | Yellow | Yellow | Red | Red | Light Blue | Red | Light Blue | Red | Light Green | Light Green |
| 4 Structured Grid | Red | Red | Light Blue | Yellow | Light Blue | Light Blue | Red | Light Blue | Red | Light Blue | Light Blue | Light Blue |
| 5 Dense Matrix | Red | Red | Yellow | Red | Red | Red | Light Blue | Red | Red | Red | Red | Light Blue |
| 6 Sparse Matrix | Yellow | Yellow | Light Blue | Red | Red | Red | Light Blue | Red | Light Blue | Light Blue | Red | Light Blue |
| 7 Spectral (FFT) | Yellow | Light Blue | Light Blue | Yellow | Yellow | Yellow | Red | Light Blue | Light Green | Red | Red | Red |
| 8 Dynamic Prog | Yellow | Light Blue | Red | Light Blue | Red | Red | Light Blue | Light Blue | Light Blue | Yellow | Light Blue | Red |
| 9 Particle Methods | Light Blue | Yellow | Light Blue | Yellow | Light Blue | Light Blue | Red | Light Blue | Light Blue | Light Blue | Light Blue | Light Blue |
| 10 Backtrack/ B&B | Light Blue | Light Blue | Yellow | Light Blue | Red | Red | Light Blue | Light Blue | Light Blue | Light Blue | Yellow | Light Blue |
| 11 Graphical Models | Light Blue | Light Blue | Yellow | Light Blue | Red | Light Blue | Light Blue | Light Blue | Light Blue | Light Blue | Red | Light Blue |
| 12 Unstructured Grid | Light Blue | Light Blue | Light Blue | Yellow | Yellow | Yellow | Red | Red | Light Blue | Light Blue | Red | Light Blue |

"Our" Pattern Language (OPL-2010) (Kurt Keutzer, Tim Mattson)



Concurrency Foundation constructs (not expressed as patterns)

- Thread creation/destruction
- Process creation/destruction
- Message-Passing
- Collective-Comm.
- Point-To-Point-Sync. (mutual exclusion)
- collective sync. (barrier)

- ❖ Multicore, multiple processors per chip
 - +*Manylane* – wide vector units common
- ❖ Heterogeneous cores, specialized for subset of workload
 - Data-parallel versus thread-parallel versus instruction-parallel
 - High-performance versus low-energy
 - App-specific: Video codec, crypto
- ❖ Heterogeneous memory hierarchies/interconnect
 - Hardware-managed versus software-managed
- ❖ Integration, System-on-a-Chip/Package
 - Smartphone SoC, Server SoC
- ❖ Drive for energy-proportionality (servers), low standby power (handheld)

How will computer architects benchmark technologies?

(Following slides from earlier presentation for ISAT
Nanometer Computing Study, at AAAS,
Cambridge, MA, July 2002)

Metrics for Nanometer Technology

Krste Asanovic

krste@mit.edu

Computer Architecture Group

MIT Laboratory for Computer Science

ISAT Nanometer Computing Study, Third Workshop

American Academy of Arts and Sciences,

Cambridge, 19 July 2002

Metrics

- **Delay**
- **Energy per Operation**
 - separate switching power from leakage power
- **Cost per Function**
 - for CMOS, depends on area, yield, power dissipation,
...
- **Physical Size**
 - volume, weight
 - affects possible applications

Constraints

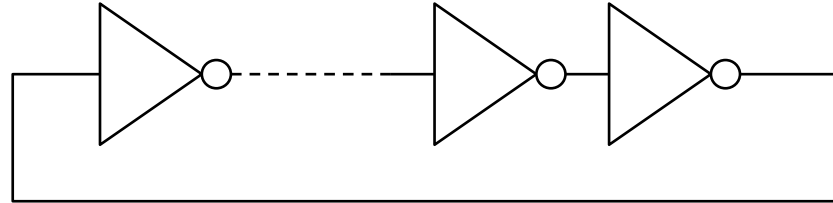
Must compare circuits optimized for chosen metric keeping others constant

- **E.G., delay of CMOS gate varies as a function of:**
 - **inputs' slew rate**
 - **output load**
 - **switching energy/operation**
 - **leakage current**
 - **noise margin**
 - **temperature (function of package cost)**
 - **MTBF (overdriven gate voltage)**
 - **speed binning (use fastest out of batch of 1,000,000?)**
 - **designer skill**

Nanocircuit Benchmarks

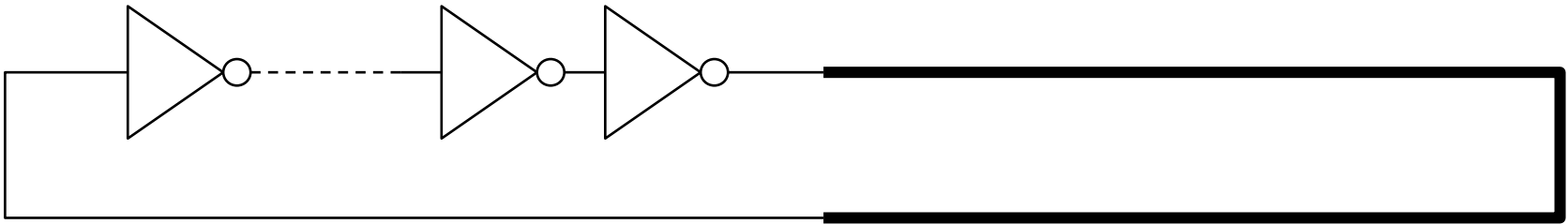
- **Ring Oscillator**
 - tests simple gates
- **Wire Oscillator**
 - tests wire latency
- **Accumulator**
 - tests clocked circuit and complex logic
- **Memory Pointer Chaser**
 - tests memory for read latency
- **Memory Streamer**
 - tests memory at peak read and write bandwidth

Ring Oscillator



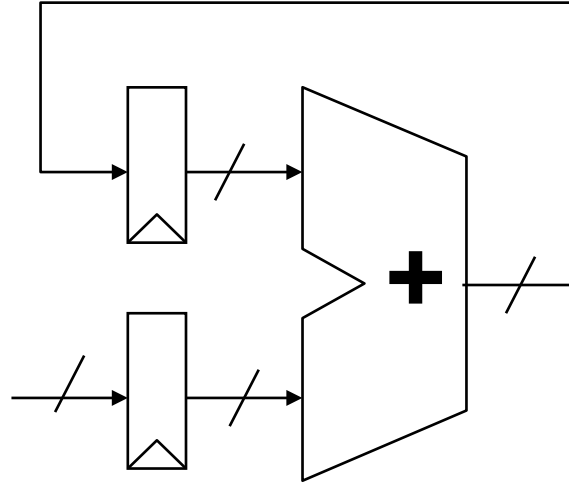
- **Self-loading**
- **Fanout of 1 (plus local wiring)**
- **Measures basic device speed and energy**
 - **result is plot of delay versus energy**

Wire Oscillator



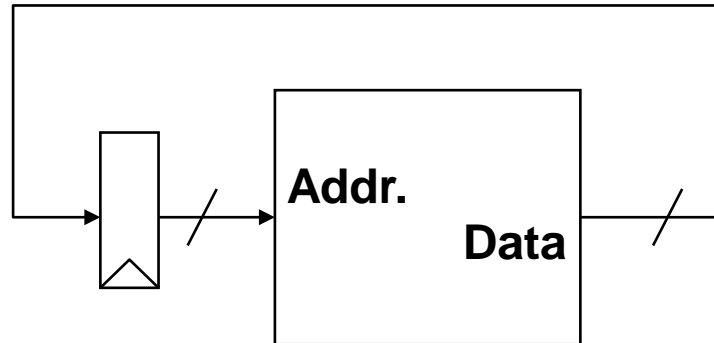
- **Self-loading**
- **Can use repeaters in wire**
- **Vary wire length in units of gate size**
 - $\text{gate width} = \text{gate area}^{1/2}$
- **Measures wire speed and energy**
 - **result is plot of delay versus length for different energies**

Accumulator



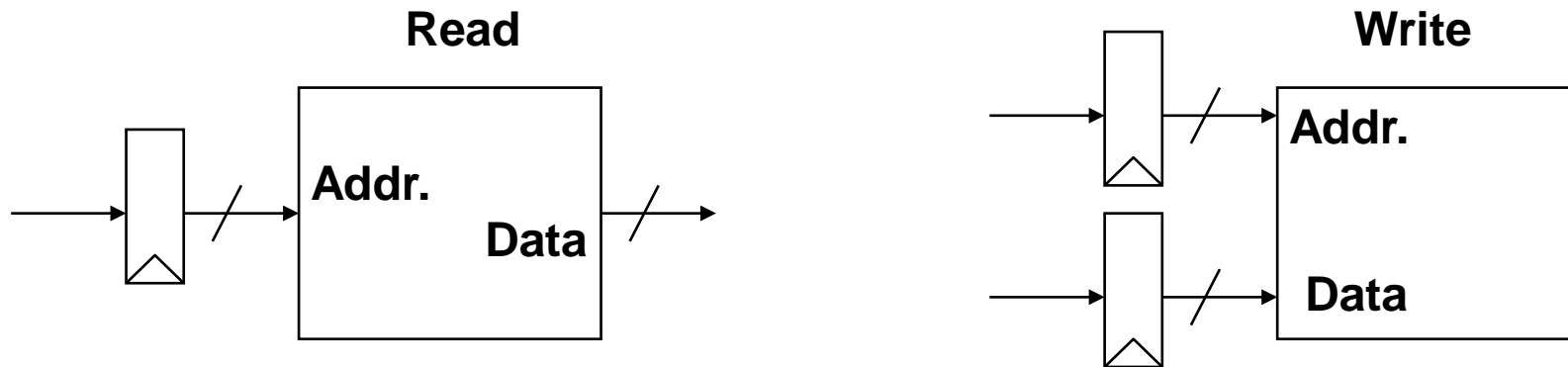
- **N-bit wraparound accumulator**
- **Self-loading**
- **Can vary bit width N**
- **Tests complex synchronous logic with fanout**
 - result is plot of delay versus energy for various N
- **Possible alternative or additional benchmark**
 - Checksum circuit with XOR/shift

Memory Read Latency



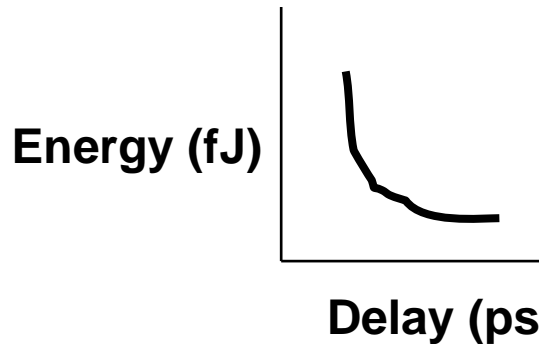
- Follow chain of random addresses preloaded into memory
- Self-loading
- Can vary RAM size and bit width
 - Data port can be wider than address input
- Measures “pointer-chasing” latency
 - result is plot of delay versus access energy for different RAM sizes

Memory Bandwidth

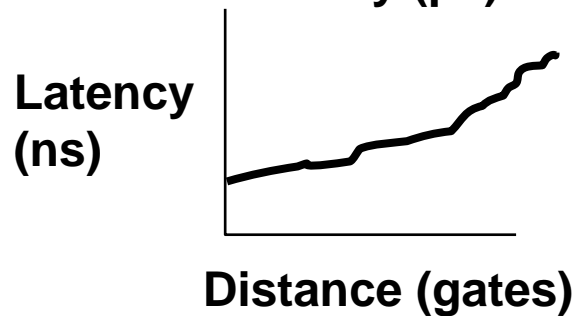


- Stream data into/out of sequential addresses
- NOT self-loading (room to cheat)
- Can vary RAM size and bit width
 - Data port can be wider than address input
- Measures peak memory bandwidth
 - results is plot of bandwidth versus energy for various capacities

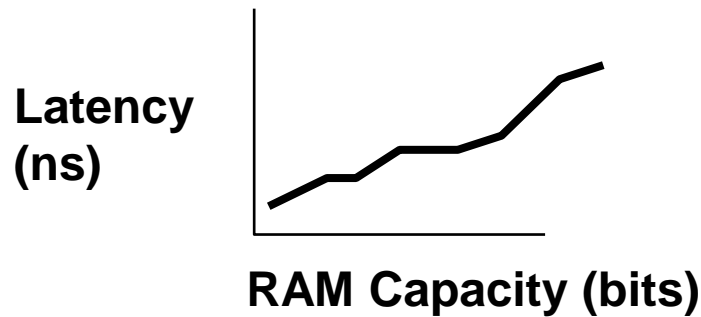
Example Result Plots



Energy-delay curves for one stage of ring oscillator.



Latency versus distance for wires.



Latency/Capacity curves for memory structures. (Also, access energy versus delay)

Handling Constraints

- **Inputs' slew rate and output load**
 - use self-loading circuits
- **Switching energy/operation and leakage current**
 - show energy-delay curves
- **Noise margin and MTBF**
 - 10 years MTBF? 1 day MTBF? (Cray-1 had 100 hour MTBF)
- **Temperature**
 - report temperature in results
- **Process variation**
 - show speed distribution
- **Designer skill**
 - use simple benchmark circuits

- ❖ “It’s lower energy/op, but 100x slower – just add parallelism”
 - Need to interconnect 100x components (size?)
 - Need to find/control 100x parallelism
 - Amdahl’s Law
- ❖ “It only works with local interconnect – make all computations local”
 - Some computations provably need global communication
- ❖ “Need both 100x parallelism and purely local communication”
 - Highly unlikely to get both in any piece of computation

- Artificial neural nets, cellular automata, ...
 - These are all usable with current technologies, i.e., supposed benefit should have shown up already.
 - Too specialized, i.e., even if Turing-Complete, really bad at many common computational patterns
- Quantum Computing
 - This does warrant a change in abstraction layers up through algorithms
 - Very, very specialized (one app?)
- The “Brain”
 - We don’t actually know how it works.
 - Great at some things we’d like to do.
 - Lousy at most (“invert this 1000x1000 matrix”)
 - Nature would probably have built something different if had to use a modern fab.

- ❖ Microcoding evolved, and was a good idea, in the era when
 - Logic was expensive (tubes)
 - ROM was cheap/fast (diode matrix)
 - RAM was expensive/slow (core memory)
- ❖ CISC->RISC was mostly about SRAM being now built out of same stuff as ROM/logic

- ❖ Photonic interconnects
 - Cut energy cost of chip-chip communication
 - Massive improvement in bandwidth density, relaxation of packaging constraints
- ❖ Fast, non-volatile memories
 - Reduce power to hold large amounts of state

- ❖ Communication-Avoiding Algorithms (Demmel, Yelick, UCB)
 - Integer factor improvements on dense matrix multiply
- ❖ Better software stacks
 - More optimized, customizable
 - 2-10x?
- ❖ Better architectures
 - Parallel, heterogeneous
 - Possible 2-10x improvements for certain apps
- ❖ Helps any future technology (CMOS or not)

- ❖ General-purpose computing architectures are quite unforgiving of new technologies' limitations
- ❖ Not because of legacy or poor design, but because of intrinsic application needs and programmer productivity costs