

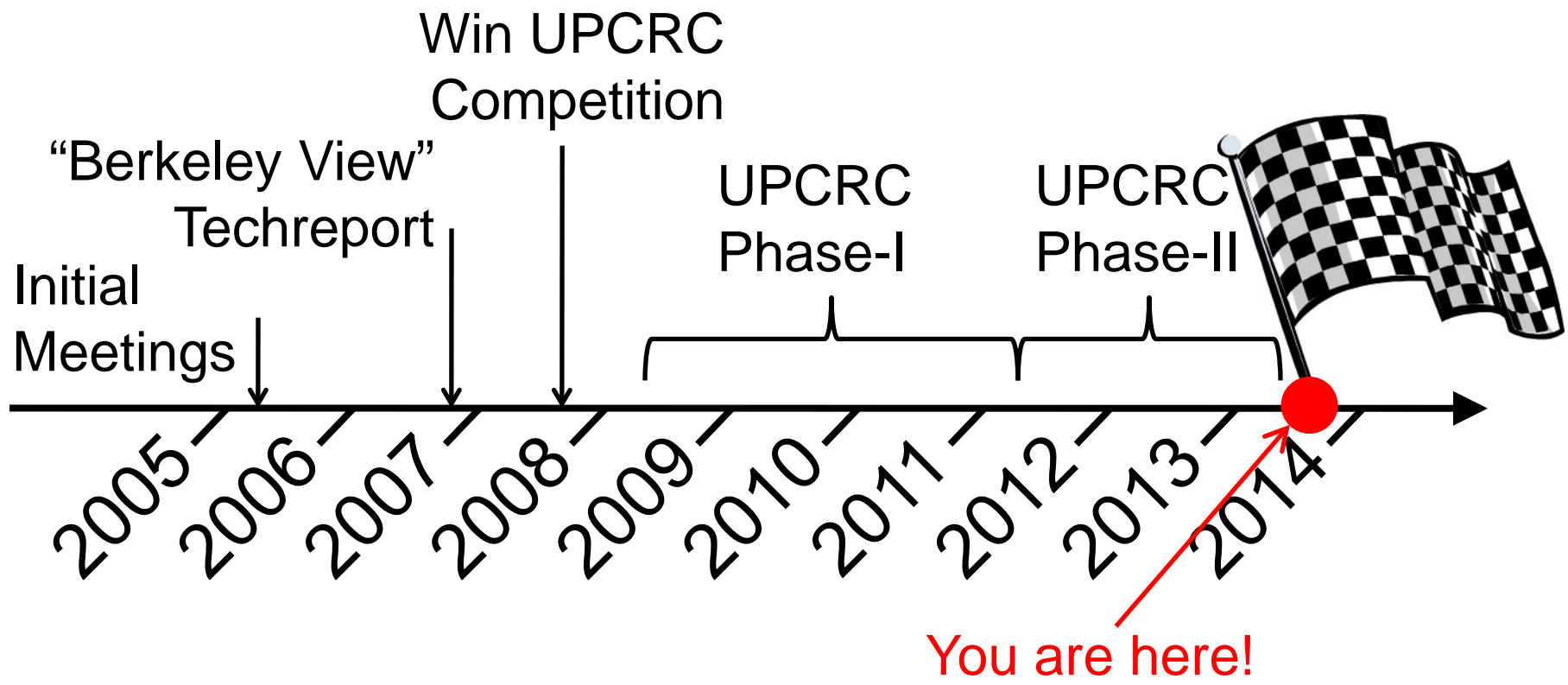
# Par Lab: Where we ended up

Krste Asanovic, Ras Bodik,  
Jim Demmel, Armando Fox, Tony Keaveny,  
Kurt Keutzer, John Kubiawicz,  
Nelson Morgan, Dave Patterson, Koushik Sen,  
David Wessel, and Kathy Yelick  
UC Berkeley

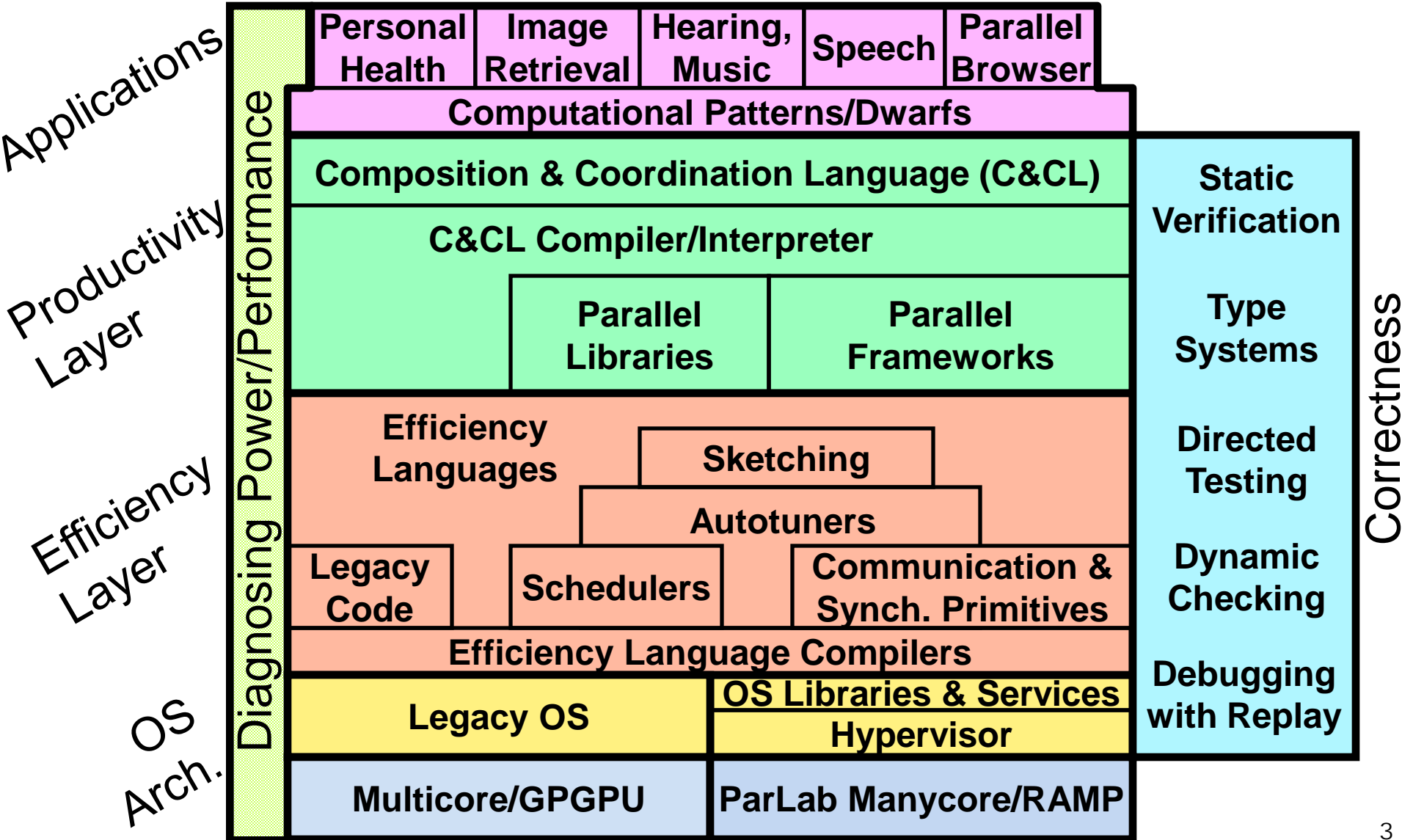
Par Lab End-of-Project Party

May 30, 2013

# Par Lab Timeline



*Easy to write correct programs that run efficiently on manycore*





Controversial in 2005,  
"Obvious" in 2013

- ❖ Laptop/Handheld ("Mobile Client")
  - Par Lab focuses on mobile clients
- ❖ Data Center or Cloud ("Cloud")
  - RAD Lab/AMPLab focuses on Cloud
- ❖ Both together ("Client+Cloud")
  - ParLab-AMPLab collaborations

- ❖ Original predictions, 2x cores every 2 years
  - 256 cores by 2013
- ❖ Reality was <2+ cores every 2 years
  - 8-16 cores in 2013
- ❖ But real growth was in SIMD performance
  - Wider, more capable SIMD units on multicore
  - GP-GPUs

Recent GPUs have up to 2,048 vector lanes!

- ❖ Many of the parallel patterns are amenable to data-parallel execution
- ❖ Despite limited memory capacity and cumbersome programming model, GPUs were surprisingly effective on wide range of apps
  - Easier to get higher speedups than multicore
  - Apps developers voted with their feet
- ❖ **Prediction:** Better CPU SIMD extensions and integrated GPUs having to use same memory system will blur/narrow CPU/GPU difference

- ❖ We architected our bare minimum requirements for accurate performance/energy counters
- ❖ **Bad news:** In 2013, commercial processors still don't meet our bare minimum
- ❖ **Good news:** Energy counters have started appearing
- ❖ **Prediction:** Counters should be given higher priority but will continue to be “unloved” parts of future architectures



- ❖ Rapid accurate simulation of manycore architectural ideas using FPGAs
- ❖ Initial version models 64 cores of SPARC v8 with shared memory system on \$750 board
- ❖ Hardware FPU, MMU, boots our OS and Par Lab stack!



	Cost	Performance (MIPS)	Time per 64 core simulation
Software Simulator	\$2,000	0.1 - 1	250 hours
RAMP Gold	\$2,000 + \$750	50 - 100	1 hour

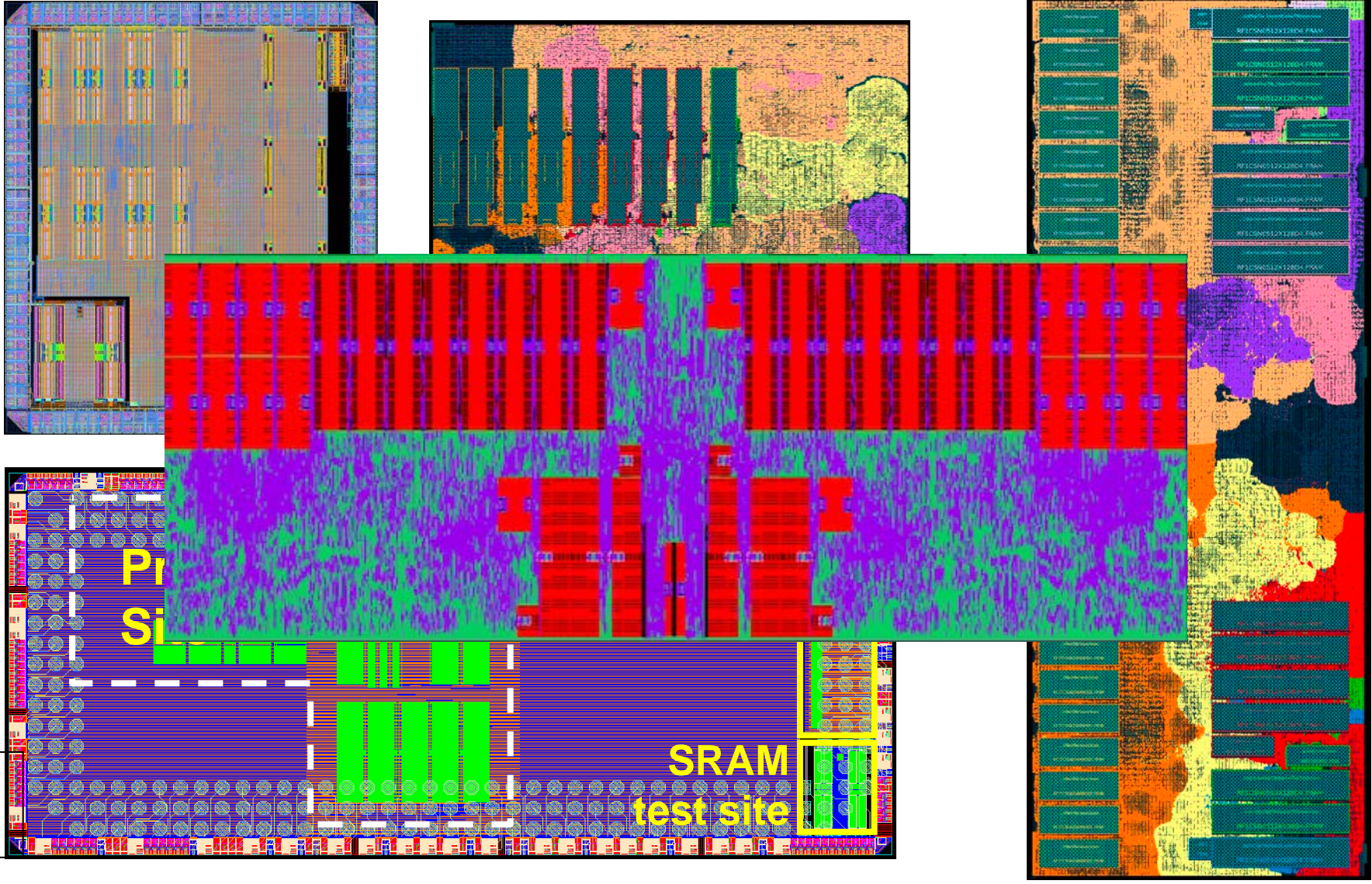
Download at: <https://sites.google.com/site/rampgold/>



- ❖ RAMP Gold design forms core of DIABLO  
“Datacenter-In-A-Box at LOw cost”
  - Execution-driven model of entire 2,000-node datacenter including network switches
- ❖ Now, generate FPGA emulations (FAME-0) of own RISC-V processors from Chisel code
- ❖ Future, developing techniques for automatic generation of efficient FPGA models from RTL
  - Chisel automatically generating higher FAME
  - New DREAMER emulation architecture

- ❖ A new clean-slate open-source ISA to support research and education
- ❖ Ports of Tessellation, Akaros, Linux OS, gcc, LLVM,..
- ❖ Multiple implementations including “Rocket” in-order research core, plus “Sodor” family of educational processors
- ❖ New vector-thread architecture designs
- ❖ FPGA emulations + tapeouts of real chips
- ❖ To be released soon at:  
**<http://www.riscv.org>**

- ❖ Embed a hardware-description language in Scala, using Scala's extension facilities
- ❖ A hardware module is just a data structure in Scala
- ❖ Different output routines can generate different types of output (C, FPGA-Verilog, ASIC-Verilog) from same hardware representation
- ❖ Full power of Scala for writing hardware generators
  - Object-Oriented: Factory objects, traits, overloading etc
  - Functional: Higher-order funcs, anonymous funcs, currying
  - Compiles to JVM: Good performance, Java interoperability
- ❖ Download from <http://chisel.eecs.berkeley.edu>

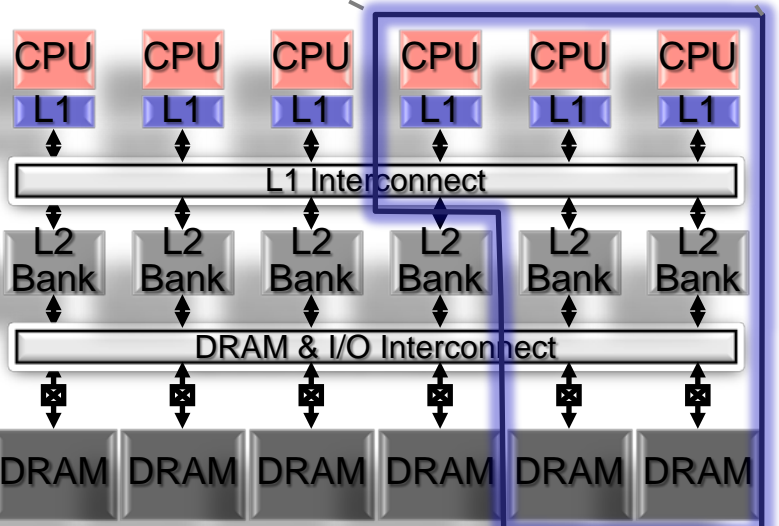
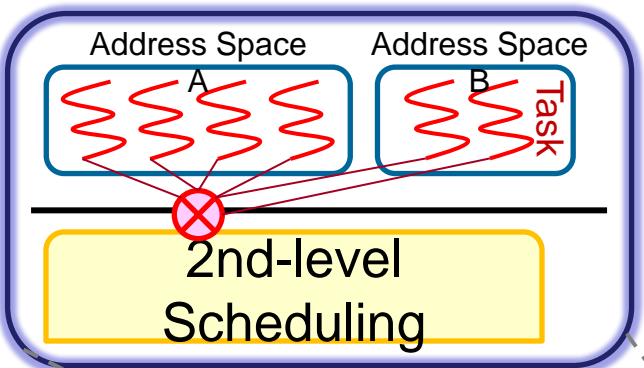
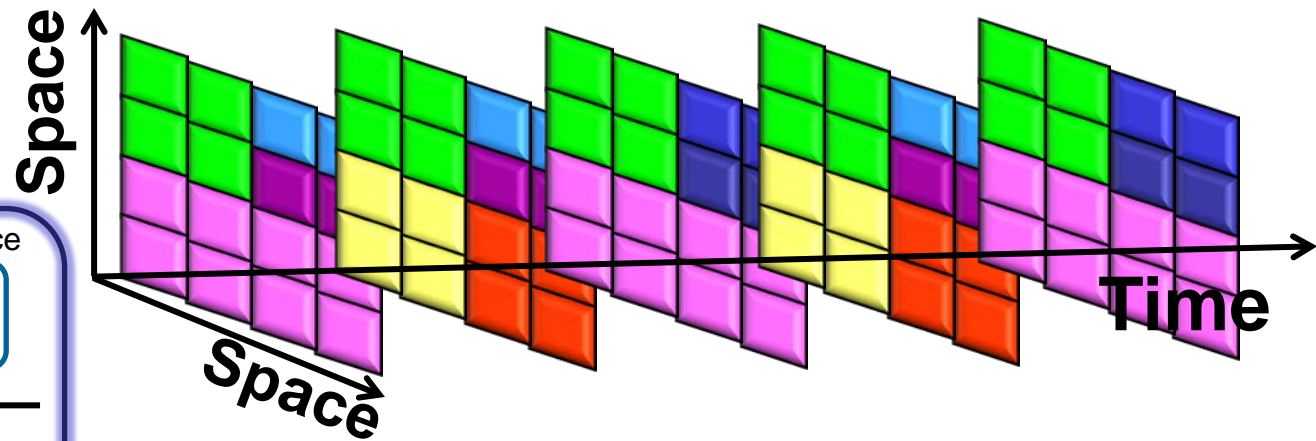


Multiple 28nm and 45nm GHz-class processor tapeouts



- ❖ Pursued our original approach of very thin hypervisor layer managing partitions
- ❖ Many ideas swirling in early days of project, concrete implementations on real x86 hardware and RAMP Gold helped provide focus
- ❖ OS group split into cloud team that moved to AMPLab (Akaros) and client team in Par Lab (Tessellation)

# Tessellation OS: Space-Time Partitioning + 2-Level Scheduling



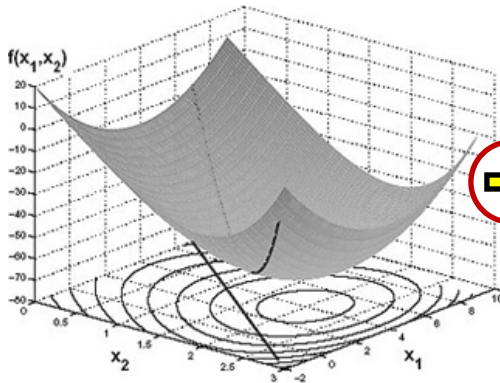
**1<sup>st</sup> level:** OS determines coarse-grain allocation of resources to jobs over space and time

**2<sup>nd</sup> level:** Application schedules component tasks onto available “harts” (hardware thread contexts) using Lithe



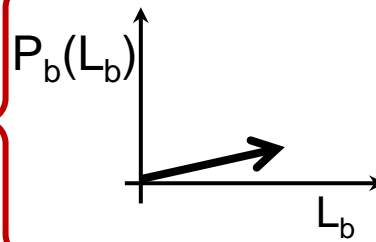
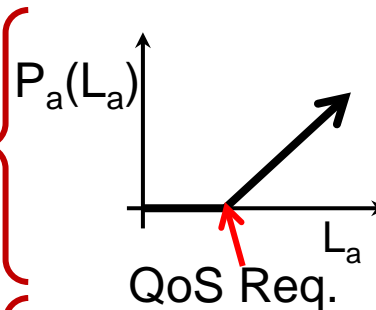
- ❖ Each process receives a **vector of basic resources** dedicated to it
  - e.g., fractions of cores, cache slices, memory pages, bandwidth
- ❖ Allocate minimum for QoS requirements
- ❖ Allocate remaining to meet some system-level objective
  - e.g., best performance, lowest energy, best user experience

Continuously  
Minimize  
(subject to restrictions  
on the total amount of  
resources)

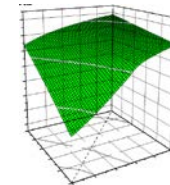


Convex Surface

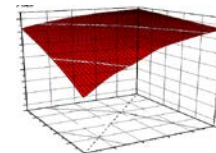
**Penalty Function**  
Reflects the app's  
importance



**Resource Utility Function**  
Performance as function of  
resources



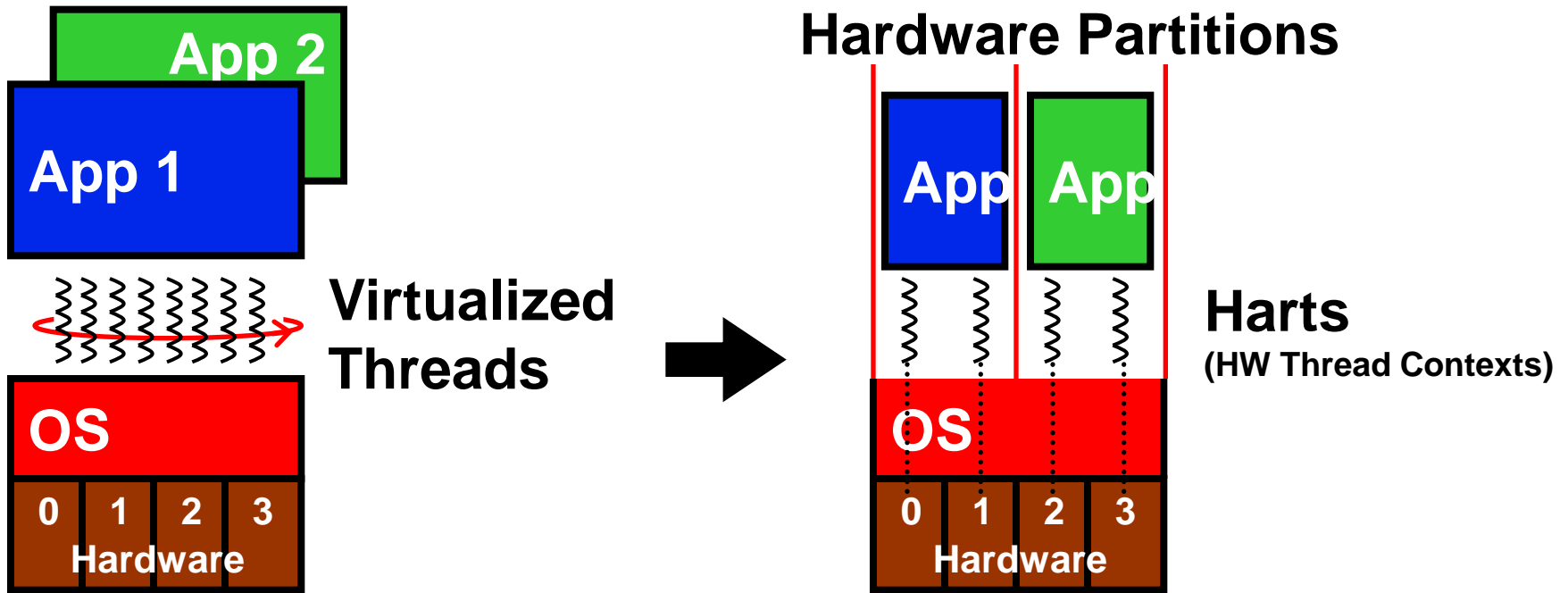
$$L_a = RU_a(r_{(0,a)}, r_{(1,a)}, \dots, r_{(n-1,a)})$$



$$L_b = RU_b(r_{(0,b)}, r_{(1,b)}, \dots, r_{(n-1,b)})$$

⋮  
**Performance Metric (L)**, e.g., latency

# “Harts”: **Hardware Threads** A Better Resource Abstraction



- *Merged* resource and computation abstraction.

- More accurate resource abstraction.
- Let apps provide own computation abstractions

# Lithe: “Liquid Thread Environment”

- ❖ Lithe is an ABI to allow application components to co-operatively share hardware threads.
- ❖ Each component is free to map computational to hardware threads in any way they see fit
  - No mandatory thread or task abstractions
- ❖ Components request but cannot demand harts, and must yield harts when blocked or finished with task

# Types of Programming (or “types of programmer”)

Example Languages

Example Activities

**Domain-Level  
(No formal CS)** Max/MSP, SQL,  
CSS/Flash/Silverlight,  
Matlab, Excel

Builds app with DSL  
and/or by customizing  
app framework

**Productivity-Level  
(Software Engineer)** Python/Ruby/Lua  
Scala

Uses application  
frameworks (or apps)

**Efficiency-Level  
(MS in CS)** Java/C#  
C/C++/FORTRAN  
assembler

Uses hardware/OS  
primitives, builds  
programming  
frameworks (or apps)

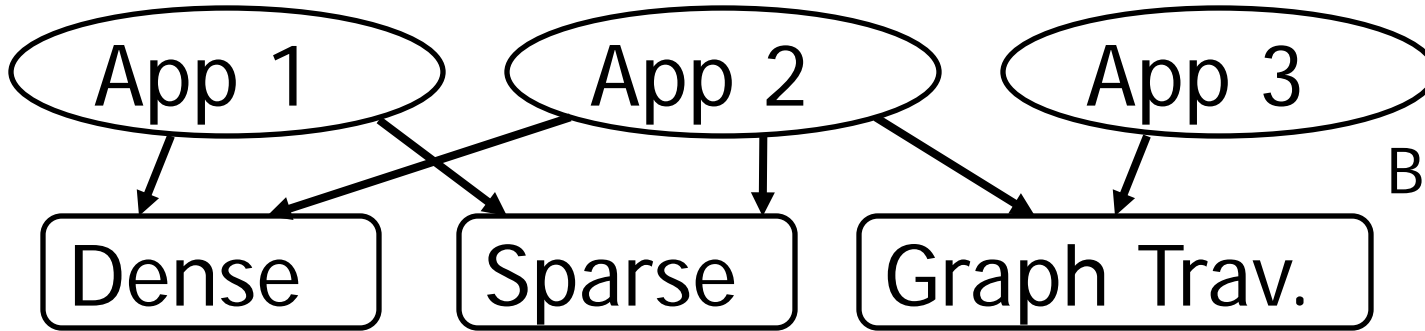
**Hardware/OS**

Provides hardware  
primitives and OS services

**Where & how to make parallelism  
visible?**

# How to make parallelism visible?

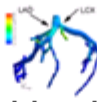




- ❖ In a new general-purpose parallel language?
  - An oxymoron?
  - Won't get adopted
  - Most big applications written in >1 language
- ❖ **Par Lab bet on Computational and Structural Patterns at all levels of programming (Domain thru Efficiency)**
  - Patterns provide a good vocabulary for domain experts
  - Also comprehensible to efficiency-level experts or hardware architects
  - *Lingua franca* between the different levels in Par Lab

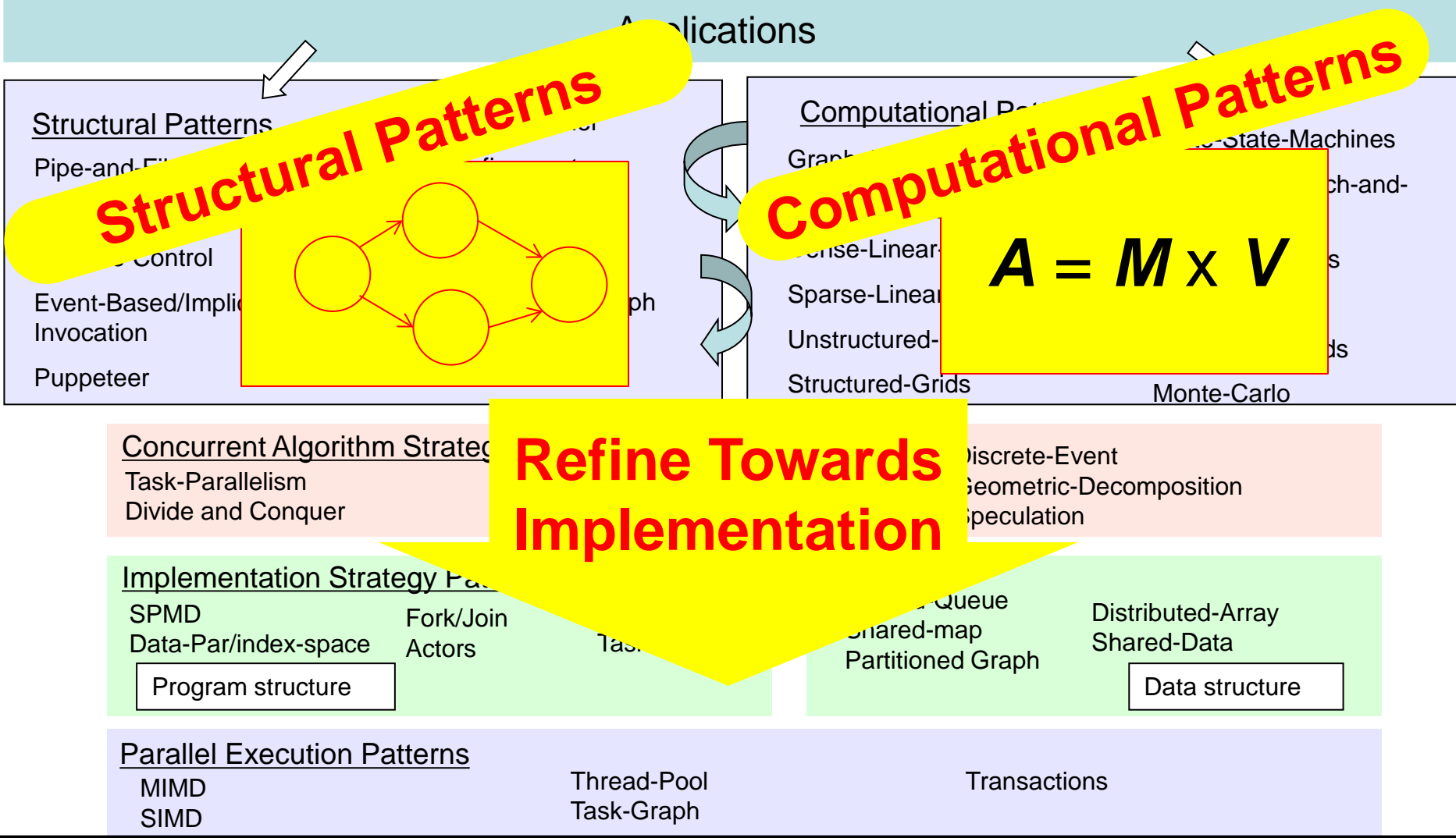


Berkeley View  
"Dwarfs"



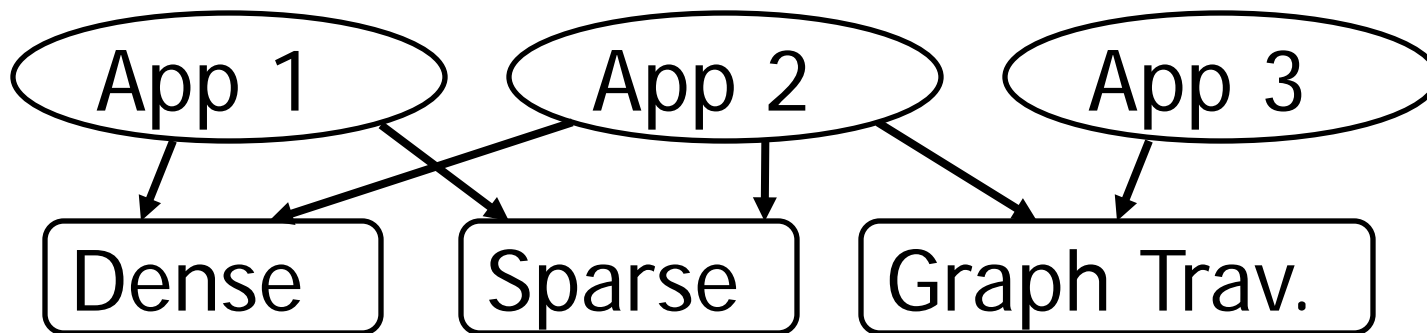
## How do compelling apps relate to 13 dwarfs?

Apps Computation	Embed	SPEC	DB	Games	ML	HPC	CAD	 Health	 Image	 Speech	 Music	 Browser
Graph Algorithms	Red	Yellow	Yellow	Yellow	Red	Light Blue	Red	Red	Green	Red	Green	Green
Graphical Models	Light Blue	Light Blue	Yellow	Green	Red	Light Blue	Light Blue	Light Blue	Green	Red	Red	Light Blue
Backtrack / B&B	Light Blue	Light Blue	Yellow	Green	Red	Light Blue	Red	Light Blue	Light Blue	Light Blue	Yellow	Light Blue
Finite State Mach.	Red	Red	Red	Yellow	Light Blue	Light Blue	Yellow	Light Blue	Light Blue	Light Blue	Light Blue	Red
Circuits	Red	Light Blue	Green	Light Blue	Green	Light Blue	Light Blue	Light Blue	Light Blue	Light Blue	Light Blue	Red
Dynamic Prog.	Yellow	Light Blue	Red	Light Blue	Red	Light Blue	Yellow	Light Blue	Light Blue	Yellow	Light Blue	Red
Unstructured Grid	Light Blue	Light Blue	Light Blue	Yellow	Yellow	Red	Light Blue	Red	Light Blue	Light Blue	Red	Light Blue
Structured Grid	Red	Red	Light Blue	Yellow	Light Blue	Red	Light Blue	Light Blue	Red	Light Blue	Light Blue	Light Blue
Dense Matrix	Red	Red	Yellow	Red	Red	Red	Yellow	Light Blue	Red	Red	Red	Light Blue
Sparse Matrix	Yellow	Yellow	Light Blue	Red	Red	Red	Yellow	Red	Light Blue	Light Blue	Red	Light Blue
Spectral (FFT)	Yellow	Light Blue	Light Blue	Yellow	Yellow	Red	Light Blue	Light Blue	Green	Red	Red	Red
Monte Carlo	Light Blue	Light Blue	Light Blue	Yellow	Light Blue	Red	Light Blue	Yellow	Light Blue	Light Blue	Light Blue	Light Blue
N-Body	Light Blue	Yellow	Light Blue	Yellow	Light Blue	Red	Light Blue	Green	Light Blue	Light Blue	Light Blue	Light Blue



Concurrency Foundation constructs (not expressed as patterns)

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



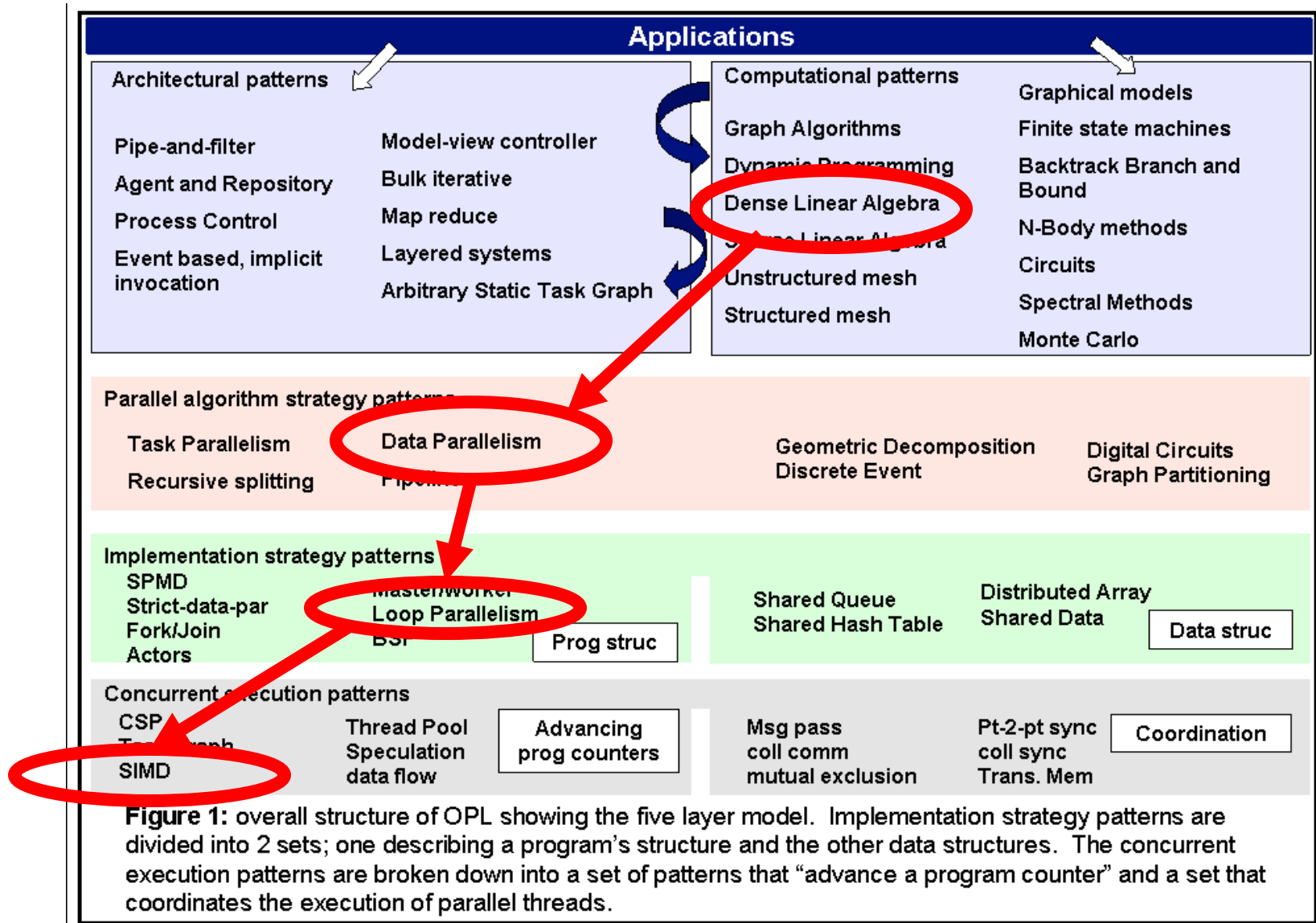
Only a few types of hardware platform

Multicore

GPU

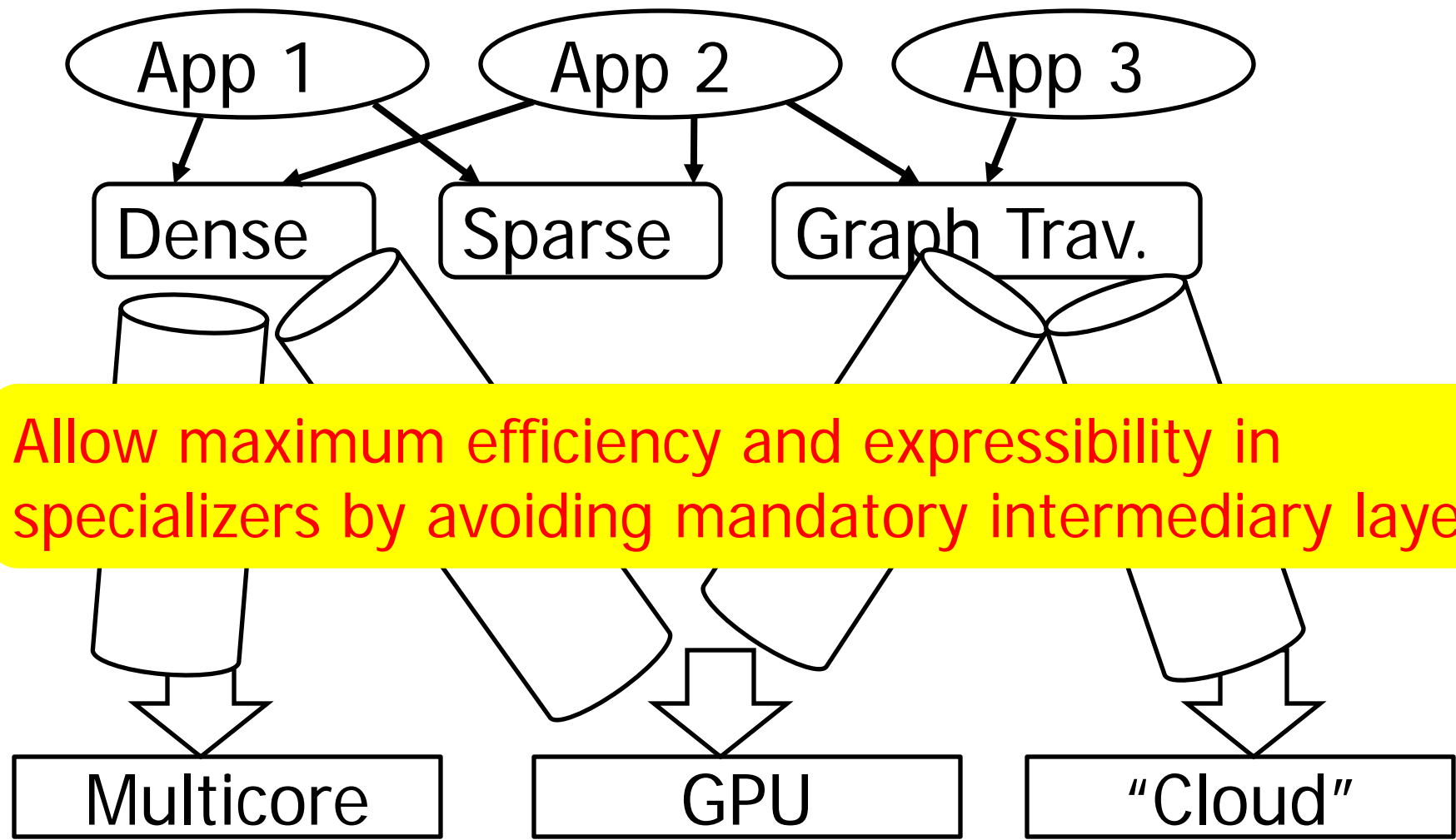
“Cloud”

# High-level pattern constrains space of reasonable low-level mappings



# Specializers: Pattern-specific and platform-specific compilers

*aka. "Stovepipes"*

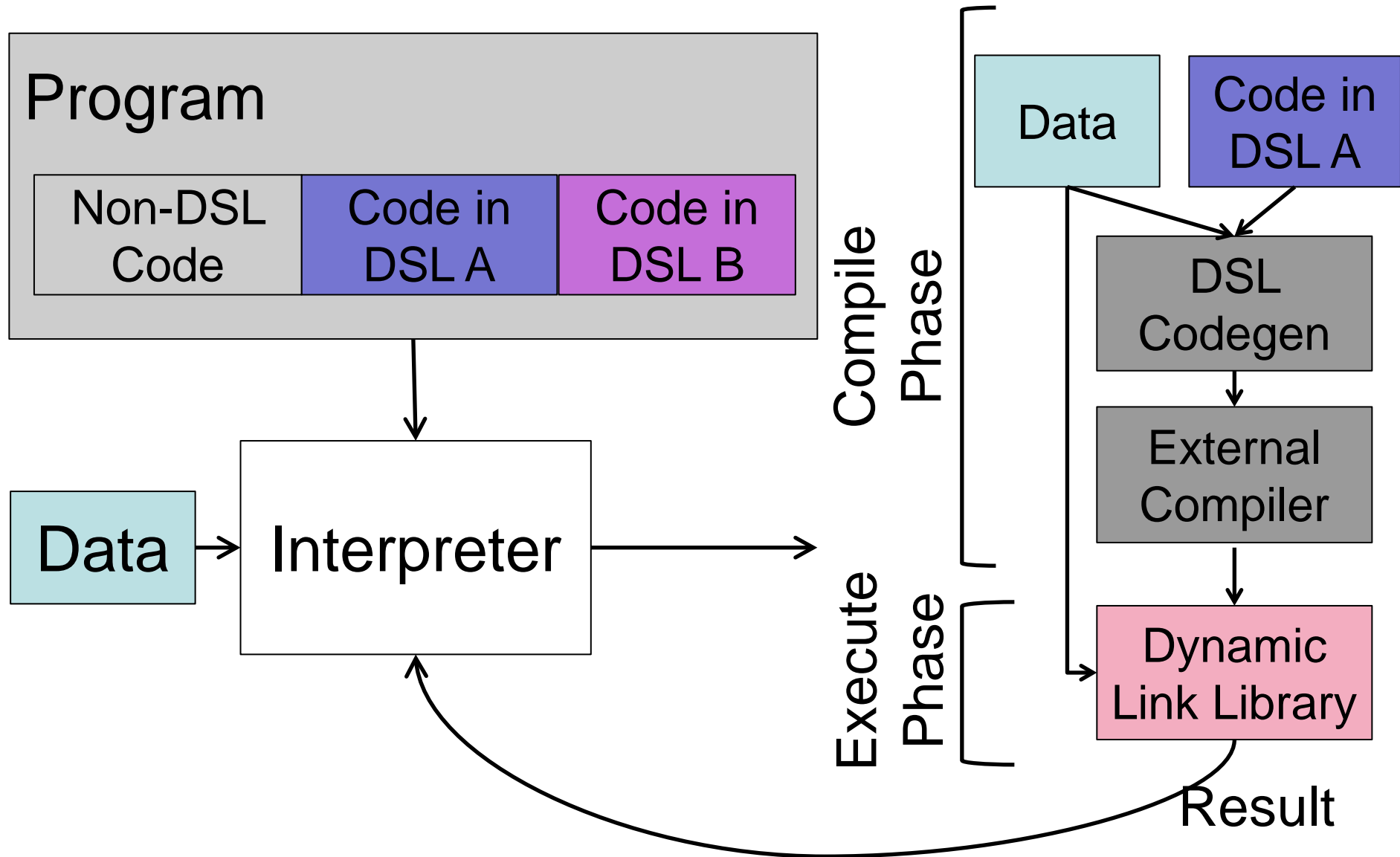


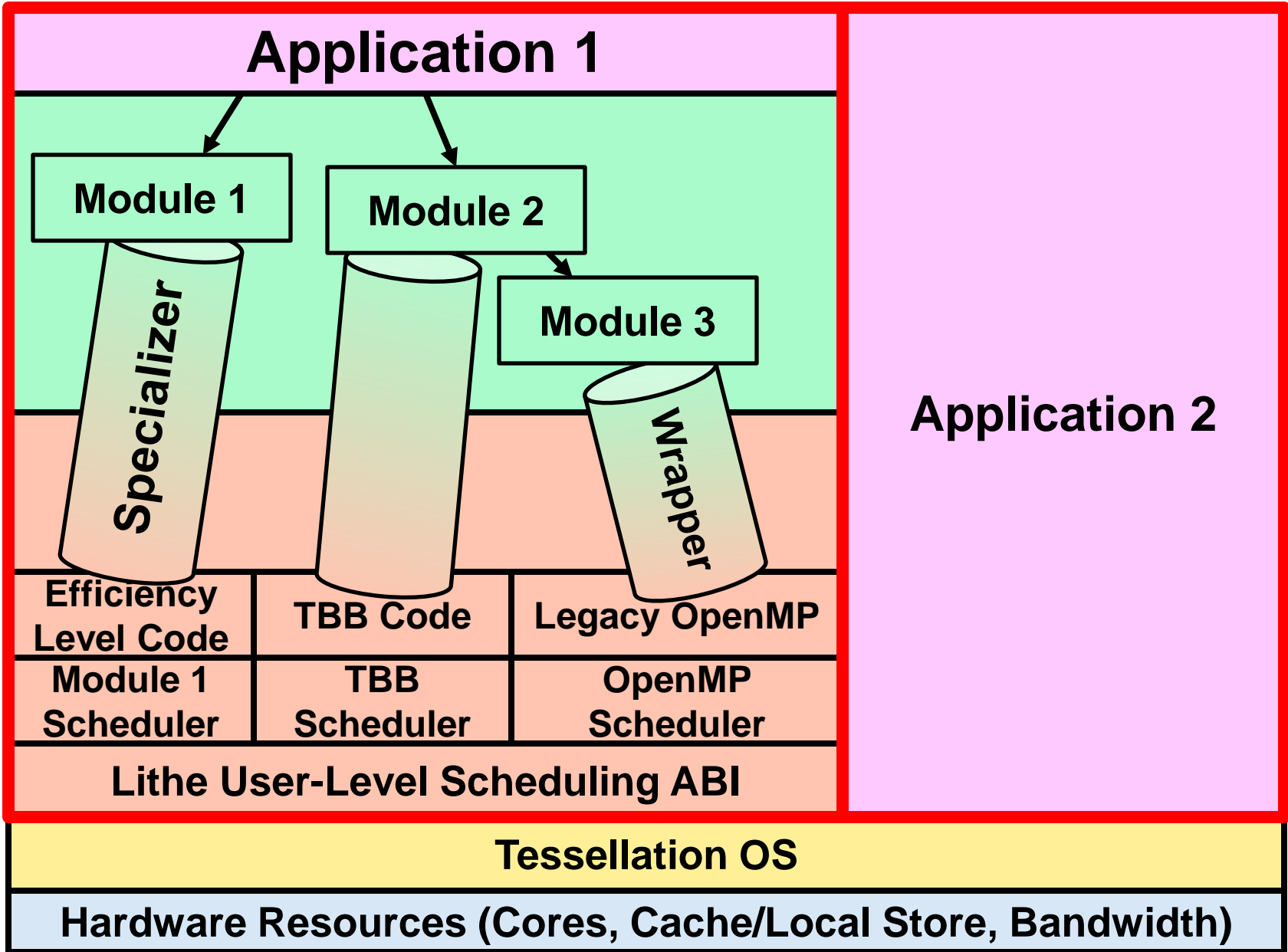
- ❖ SEJITS bridges productivity and efficiency layers through specializers embedded in modern high-level productivity language (Python, Ruby)
  - Embedded “specializers” use language facilities to map high-level pattern to efficient low-level code (at run time, install time, or development time)
  - Specializers can incorporate or package autotuners

## Two ParLab SEJITS projects:

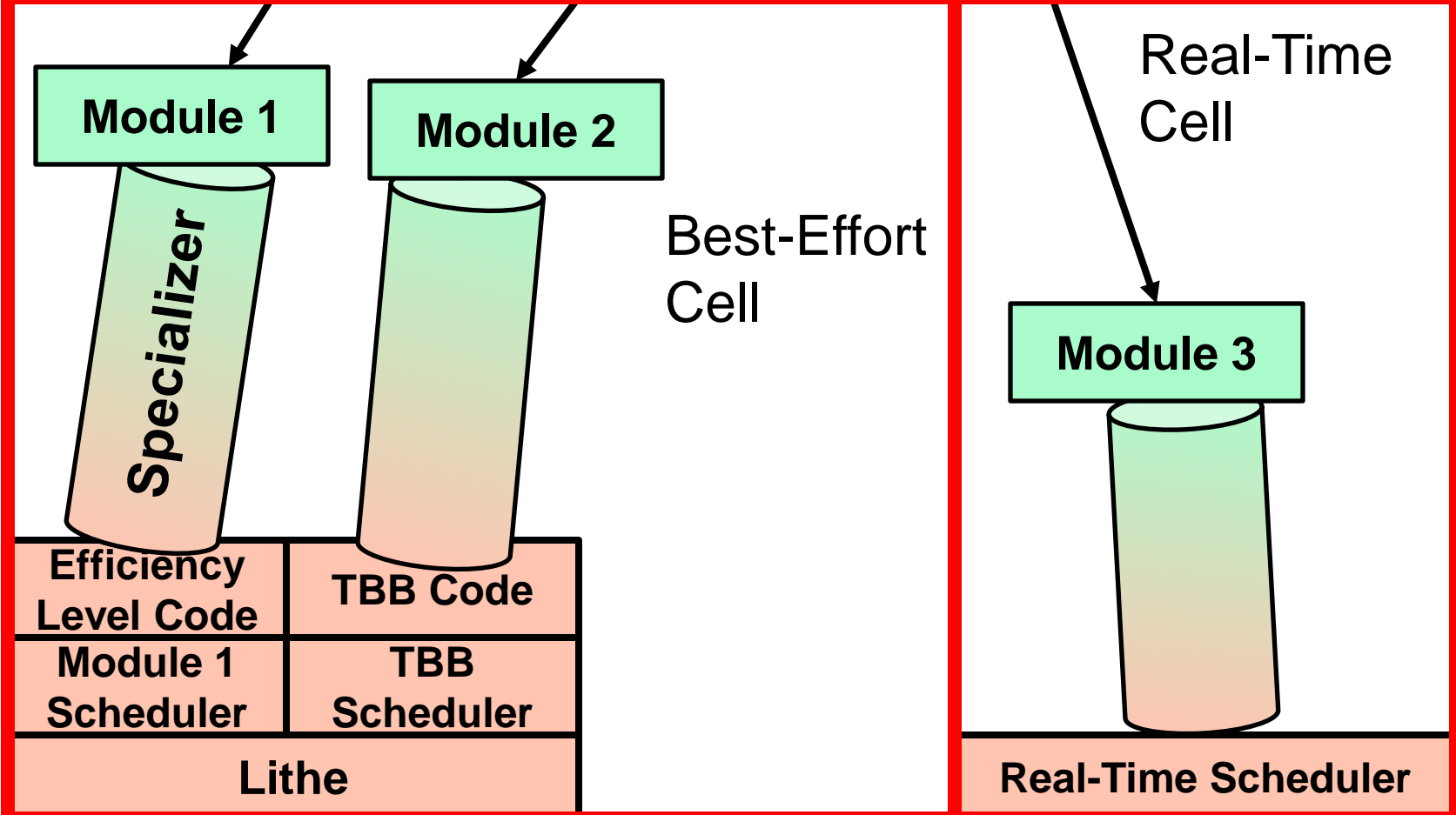
- ❖ **Copperhead**: Data-parallel subset of Python, development continuing at NVIDIA
- ❖ **Asp**: “Asp is Sejits in Python” general specializer framework
  - Provide functionality common across different specializers







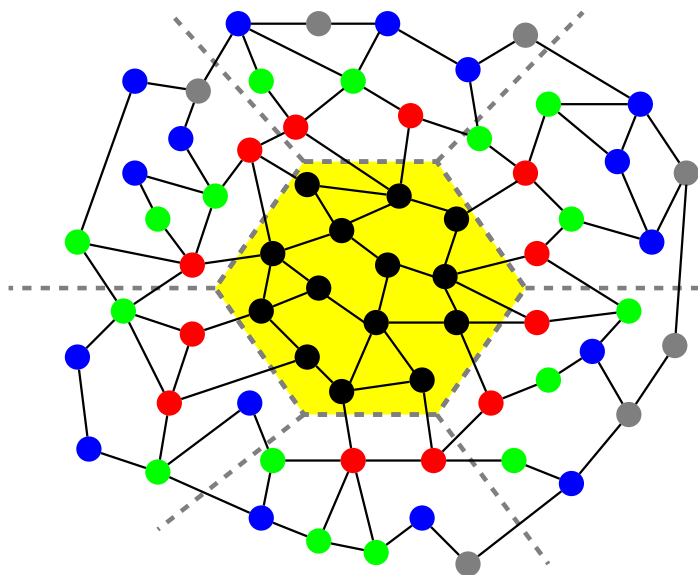
## Application



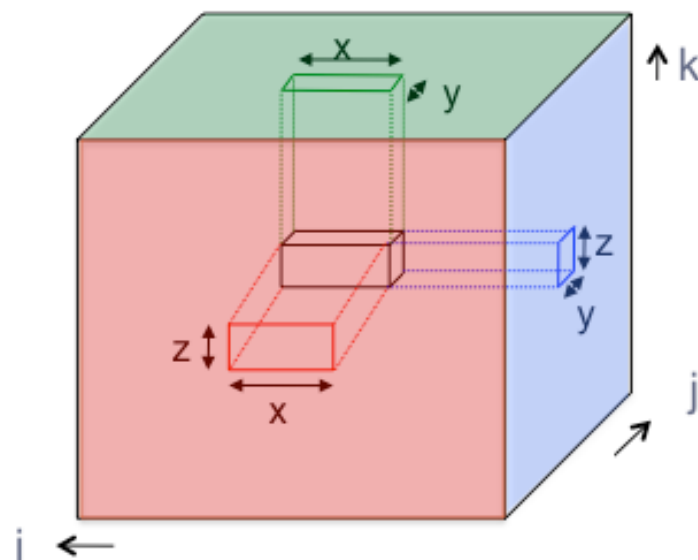
## Tessellation OS

Hardware Resources (Cores, Cache/Local Store, Bandwidth)

- ❖ Past algorithms: FLOPs expensive, Moves cheap
- ❖ New theory: proves lower bounds on data movement; both serial (memory hierarchy) and parallel data movement
- ❖ New practice: codes achieve lower bound and speedups
- ❖ Widely applicable: all linear algebra, Health app...



Idea #1: read a piece of a sparse matrix (= graph) into fast memory and take multiple steps of higher-level algorithm



Idea #2: replicate data (including left-hand side arrays, as in  $C$  in  $C=A*B$ ) and compute partial results, reduce later

## ❖ Matrix multiplication

- Up to **12x** on IBM 64K-core BG/P for  $n=8K$ ; **95% less communication**

## ❖ QR decomposition (used in least squares, data mining, ...)

- Up to **8x** on 8-core dual-socket Intel Clovertown, for  $10M \times 10$
- Up to **6.7x** on 16-proc. Pentium III cluster, for  $100K \times 200$
- Up to **13x** on Tesla C2050 / Fermi, for  $110k \times 100$
- “infinite speedup” for out-of-core on PowerPC laptop
  - LAPACK thrashed virtual memory, didn’t finish

## ❖ Eigenvalues of band symmetric matrices

- Up to **17x** on Intel Gainestown, 8 core, vs MKL 10.0

## ❖ Iterative sparse linear equations solvers (GMRES)

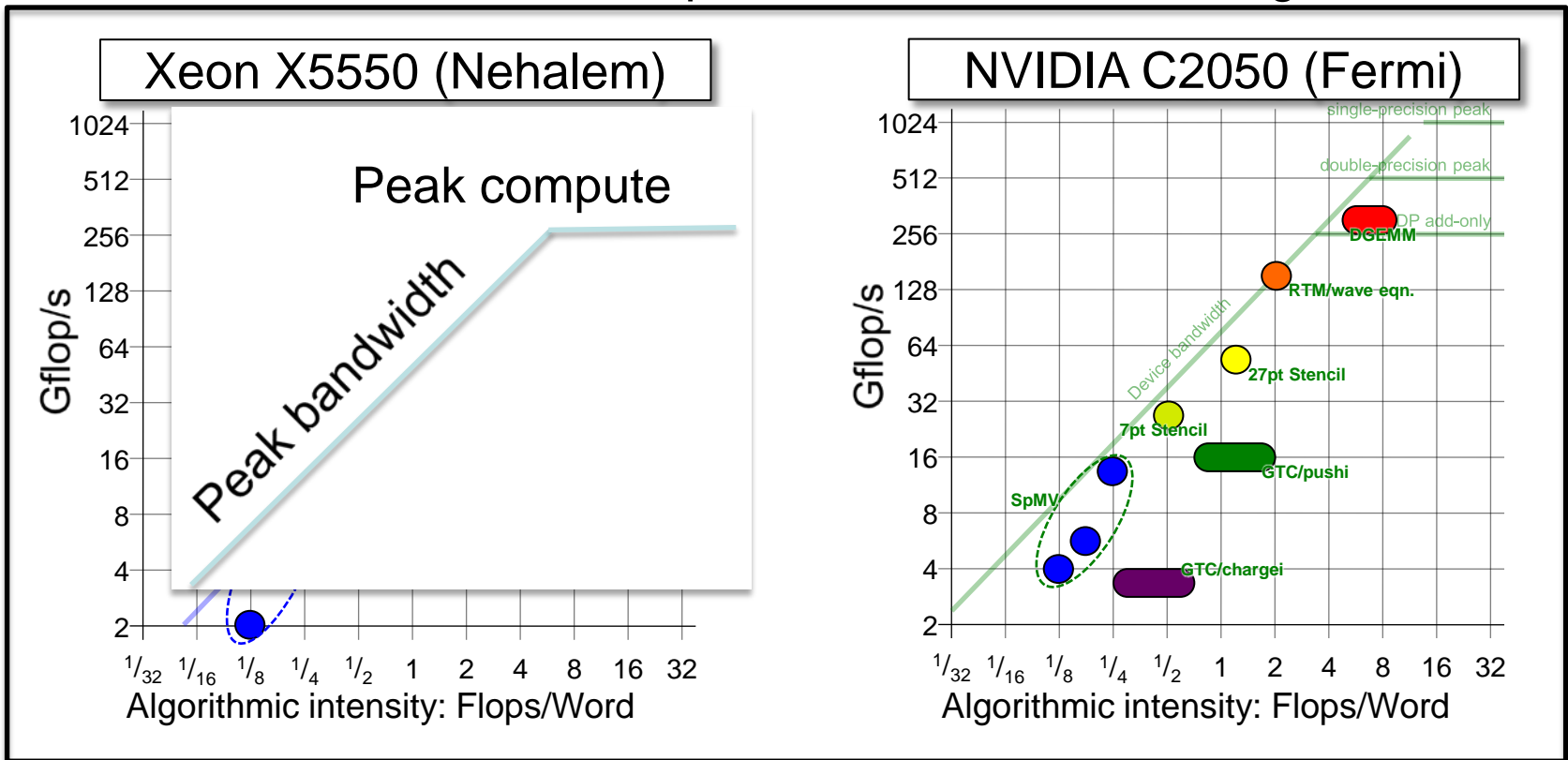
- Up to **4.3x** on Intel Clovertown, 8 core

## ❖ N-body (direct particle interactions with cutoff distance)

- Up to **10x** on Cray XT-4 (Hopper), 24K particles on 6K procs.

**Next: automatically xform code; new “HBL” theory just out!**

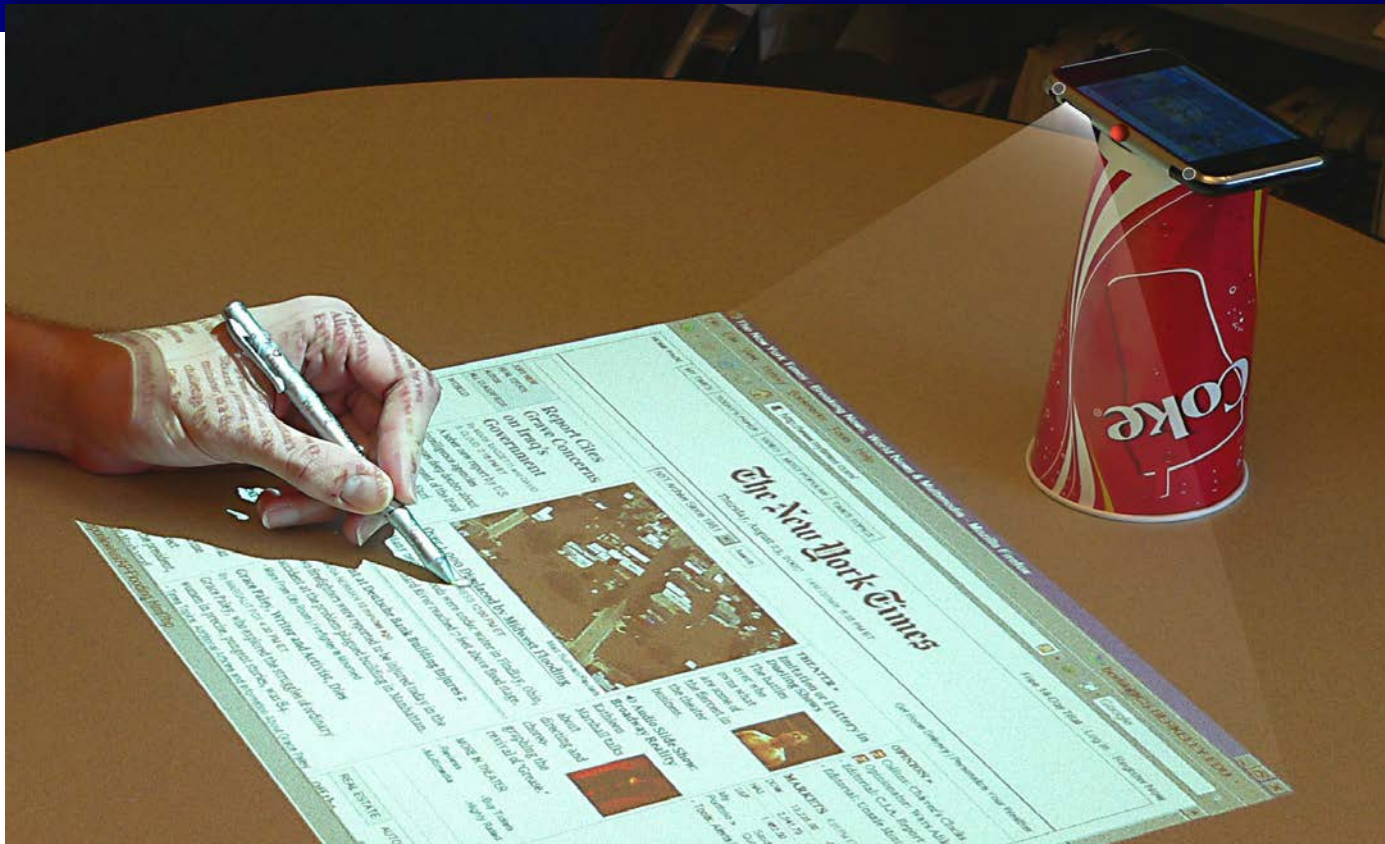
- ❖ Autotuners are code generators plus search
- ❖ Avoids two unsolved compiler problems: dependence analysis and accurate performance models
- ❖ New: particles, stencils, graphs,... and manylane/core optimizations
- ❖ New roofline model to aid in performance understanding



Work by Williams, Olike, Shalf, Madduri, Kamil, Im, Ethier,...

## ❖ Some Results

- Active Testing: for finding non-deterministic bugs such as data races, deadlocks, atomicity violations
  - Open-source [BSD-licensed] CalFuzzer for **Java**
  - Thrille for C/Pthreads and UPC
- Specification and assertion framework for parallelism correctness
  - Introduced Bridge Predicates
  - Nondeterministic Sequential Specification to separate parallel correctness from functional correctness
- ConcurrIt: A testing framework for concurrent programs
  - JUnit or xUnit for concurrent programs
  - Applied to Chrome and Firefox browsers
- Concolic testing: for automated test input generation
  - Java
  - Javascript (ongoing)



- ❖ **2007 Vision:** desktop-quality browsing on mobiles.
- ❖ **Now:** yes, but only 200 web pages / battery charge.
- ❖ **2007 Vision:** browser as an app platform.
- ❖ **Now:** Google Glass is a browser app.



Parallelism improves responsiveness, energy use.

- 2007: parallel browser controversial
- 2013: Mozilla Servo & Google Blink browsers

Some of our results:

- First scalable parser (in Qualcomm browser)
- Synthesizer of parallel layout engines
- Parallel layout retrofitted to Safari via WebCL
- Collaboration with Mozilla on parallel Servo

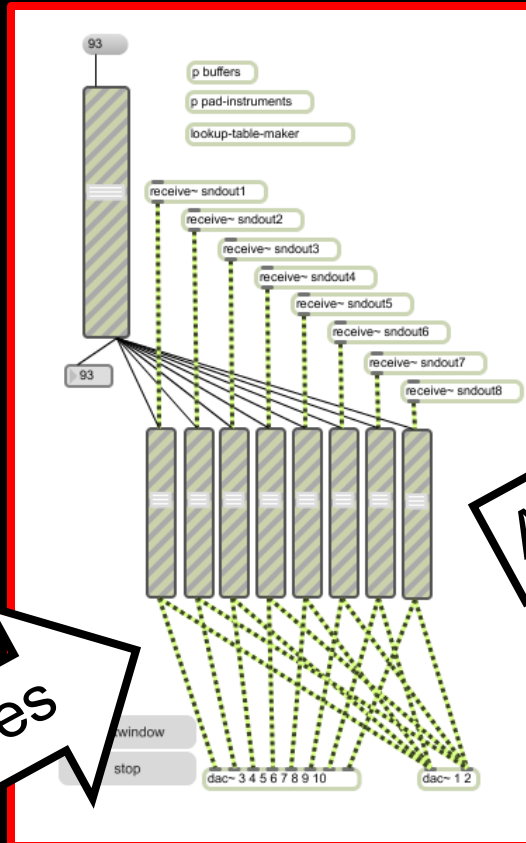
Synthesis: search a huge space for a program that is semantically correct and high-performance

- alternative to classical AST-rewrite compilers
- search implemented as constraint solving

Some of our synthesizers:

- FTL: parallel layout engines
- Programming for ULP spatial manycore
- SQL query programming by demonstration

New user interfaces  
with pressure-sensitive  
multi-touch gestural  
interfaces



120-channel  
speaker array

Gestures

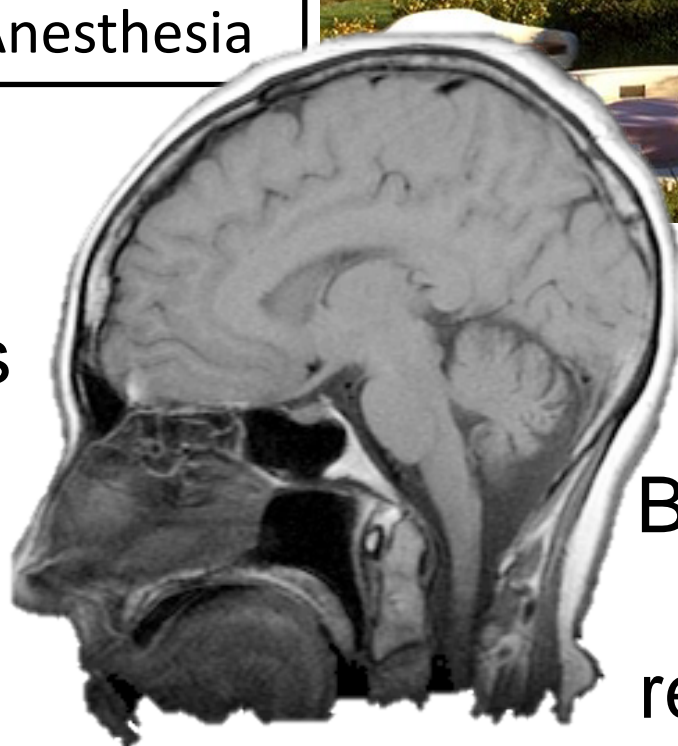
Programmable virtual instrument  
and audio processing

- ❖ 5 Original Apps: Parallel Browser (Ras Bodik), Music (David Wessel), Speech (Nelson Morgan), Health (Tony Keavney), Image Retrieval (Kurt Keutzer)
- ❖ New external application collaborators:
  - Pediatric MRI (Michael Lustig, Shreyas Vassanwala @Stanford)
  - Multimedia and Speech (Dorothea Kolossa @TU Berlin)
  - Computer Vision (Jitendra Malik, Thomas Brox)
  - Computational Finance (Matthew Dixon @UCD)
  - Natural Language Translation (Dan Klein)
  - Programming multitouch interfaces (Maneesh Agrawala)
  - Protein Docking (Henry Gabb, Intel)

Typical exam ~ 1 hour  
Motion blurs the images  
Scanner is a small loud tunnel  
  
Difficult for children to stay still!  
  
Traditional Solution: Anesthesia



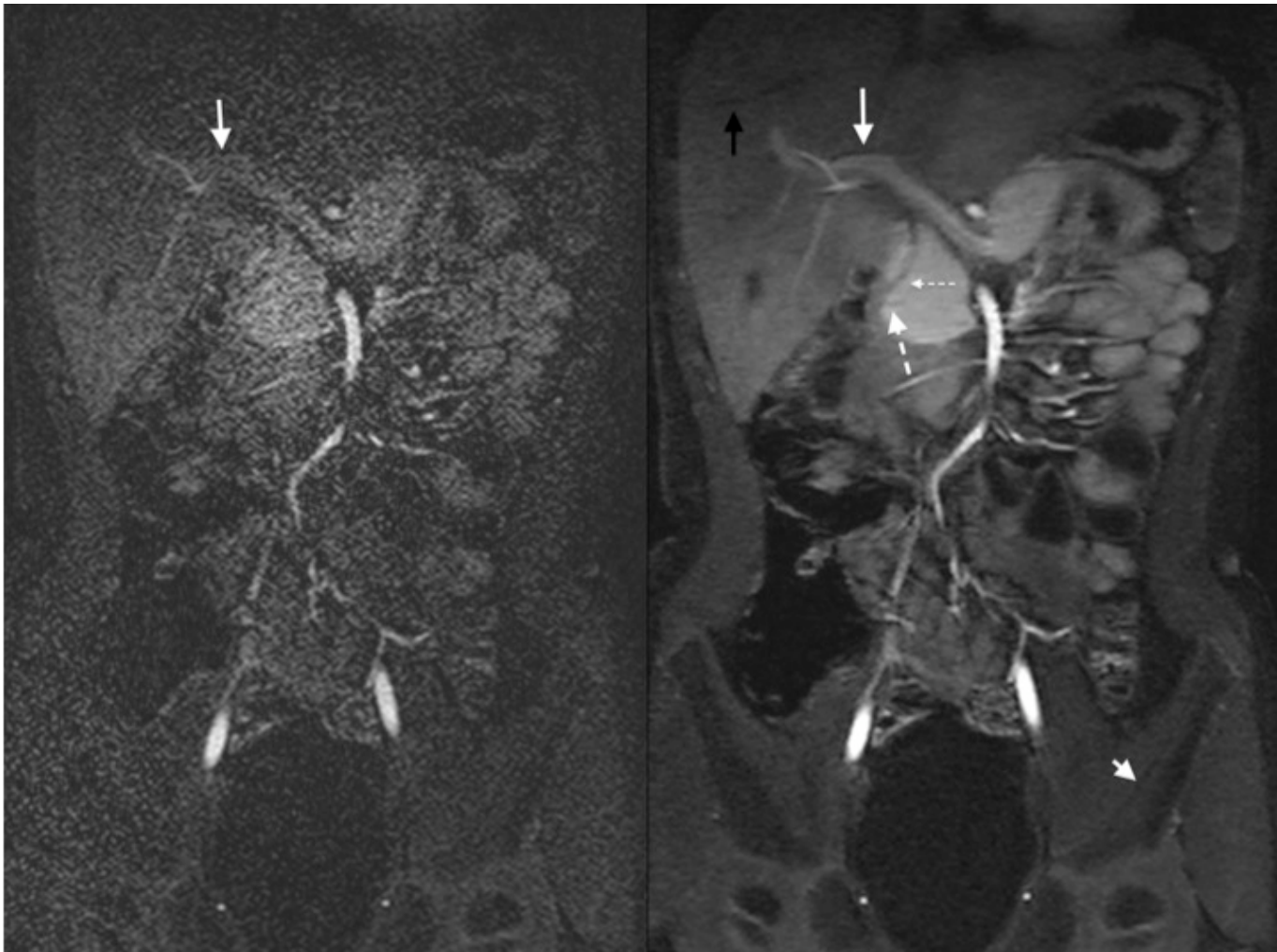
Compressed  
Sensing reduces  
each scan to 15  
seconds



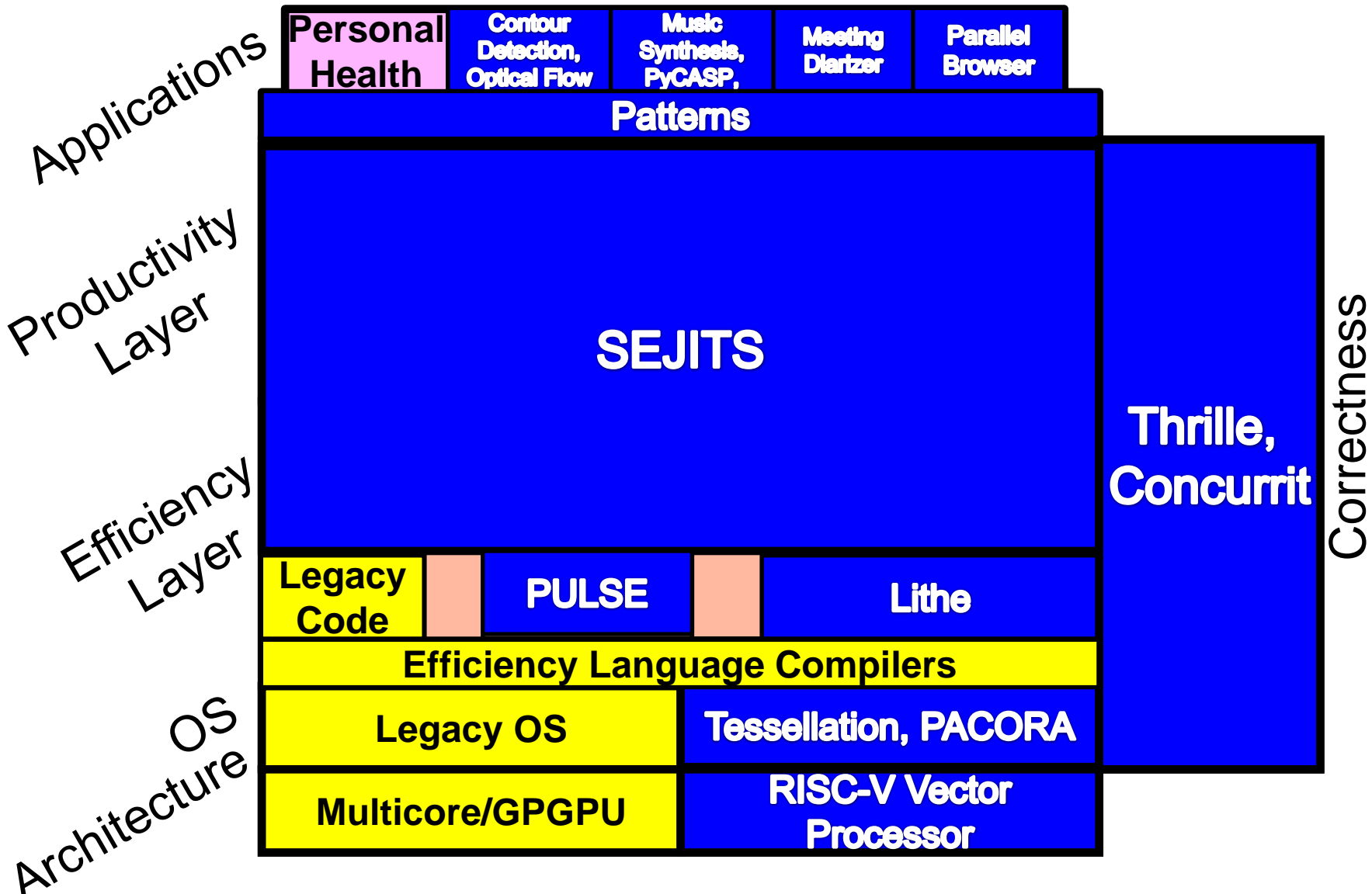
But takes too long  
(hours) to  
reconstruct image



- Image reconstruction from 1-2 hours down to  $< 1$  min
- In use in clinical trials



# Today's Demos



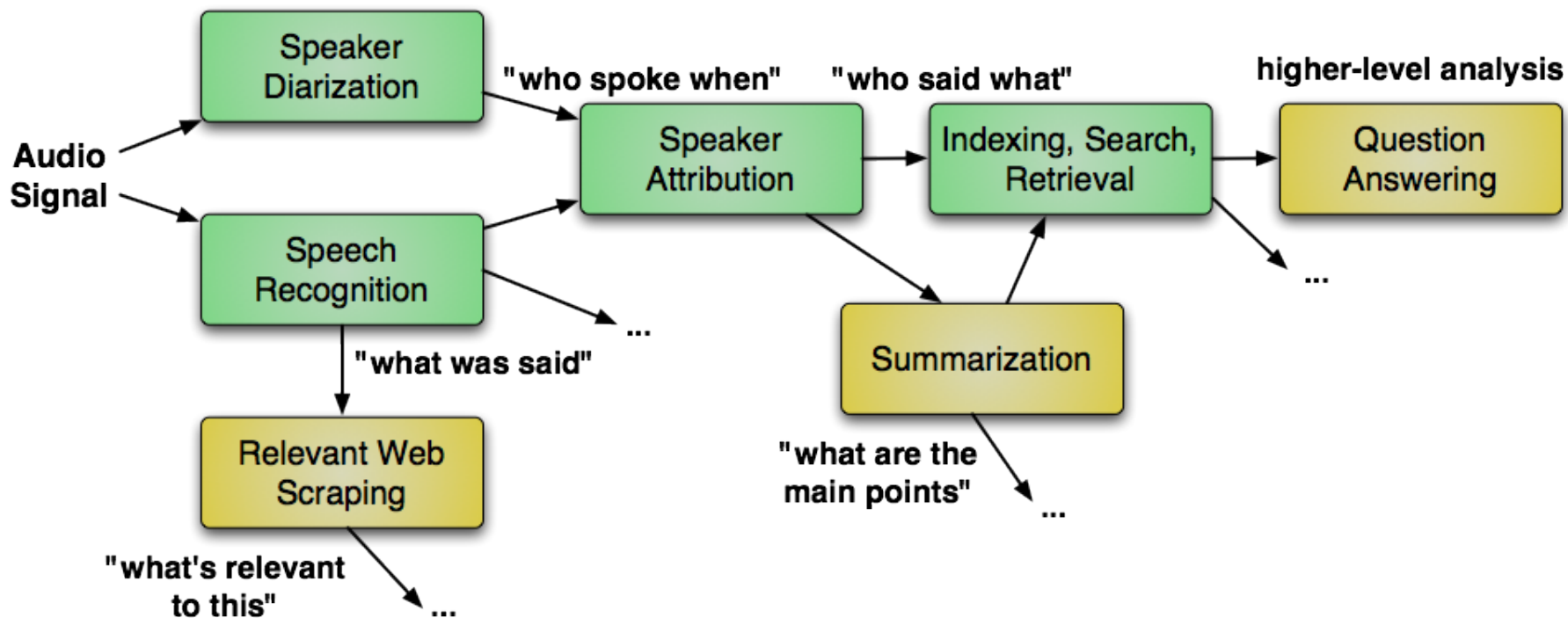


## Gerald Friedland International Computer Science Institute (ICSI)

**Work together with  
primarily:**  
**Katya Gonina**  
**David Sheffield**  
**Adam Janin**  
**Brian Hill**  
**Jike Chong**  
**Nelson Morgan**  
**Kurt Keutzer**  
**Ganapati S.**  
**Mishali N.**



Applet started.



Created a fully integrated, very fast meeting diarist SEJITized that is currently tech-transferred to Intel.

	Before Par Lab	After Par Lab
“who spoke when”	~10k LOC 0.3 x RT	~100 LOC 250 x RT
“what was said”	~100k LOC 0.1 x RT	~ 5k LOC 1 x RT

Created a fully integrated, very fast meeting diarist using SEJITs:

- ❖ Online=Offline processing for “who spoke when”
- ❖ Diarization used for BIG DATA video processing
- ❖ Tech-transfer to Intel

- ❖ 193 Conference Papers
  - ❖ 28 Journal Papers
  - ❖ 94 Technical Reports
  - ❖ 12+ Best Paper Awards
- 
- ❖ Berkeley View TR, >1,300 citations
  - ❖ Par Lab papers, >8,000 citations

- ❖ Par Lab summer bootcamps
  - 2009-2012: >1300 attendees including >300 from industry
- ❖ CS61C Reworked intro architecture class with parallelism
  - 480 students in Fall 2012 semester
  - Revised 5<sup>th</sup> edition of undergrad text (used at 400 universities)
- ❖ CS152 Undergrad Architecture using Chisel processors
- ❖ CS164 Students design and own DSLs, implement a browser
- ❖ CS194 Undergrad parallel patterns class
  - 3<sup>rd</sup> offering, co-taught with Tim Mattson, Intel
  - 3 posters here from successful undergrad projects
- ❖ CS250 Graduate VLSI Design using Chisel/Agile Hardware
- ❖ CS267 Graduate parallel computing class
  - Added material on dwarfs, patterns, autotuning, apps
  - Homeworks ported to .net with Microsoft
  - NSF-funded MOOC XSEDE launched spring 2013

## PhD students

- ❖ 91 total PhD participants,
- ❖ 23 of which graduated by 2013

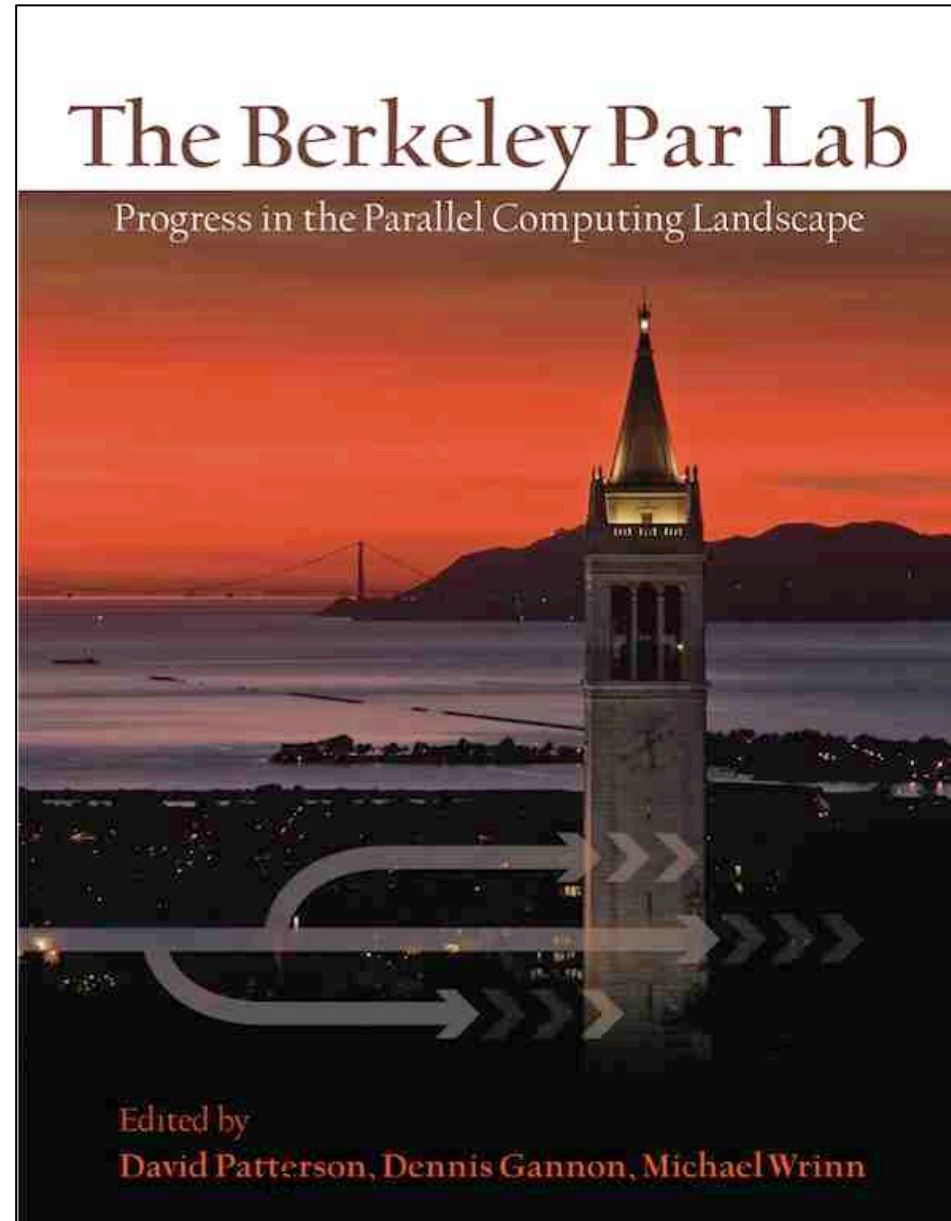
## MS Students

- ❖ 35 total MS participants,
- ❖ 13 of which graduated by 2013

## Post-Docs

- ❖ 5 current

- ❖ 18 chapters
  - Overview + 1-2 research papers
- ❖ ≈ 600 pages
- ❖ Expected by June 30
  - Amazon Ebook \$0.99
- ❖ Print book signup page





- ❖ OS and Music work continuing in new SWARM Lab, programming the “swarm” of environmental devices  
<http://swarmlab.eecs.berkeley.edu>
- ❖ Software synthesis, Correctness -> Chaperone, ExCape NSF Expedition
- ❖ ASPIRE: patterns, communication-avoiding algorithms, SEJITS, RISC-V, specialized architectures, Chisel, Agile Hardware  
<http://aspire.eecs.berkeley.edu>

- ❖ Research supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227).
- ❖ Additional support comes from Par Lab affiliates National Instruments, NEC, Nokia, NVIDIA, Samsung, and Oracle/Sun.