CS 250
VLSI System Design

Lecture 6 – Accelerator Projects

2013-9-17

Professor Jonathan Bachrach

slides by John Lazzaro

TA: Colin Schmidt

www-inst.eecs.berkeley.edu/~cs250/

Today's lecture plan ...

- **Power and energy.** The techniques you'll be able to use in your project.
- **Pareto Optimality ...** and how it impacts your project definition.
- **Accelerator interface ...** and its limits.
- **Worked examples.** Several project ideas, sketched out in detail.

Break

- **Starting points.** Brief descriptions of projects ideas you may want to pursue.

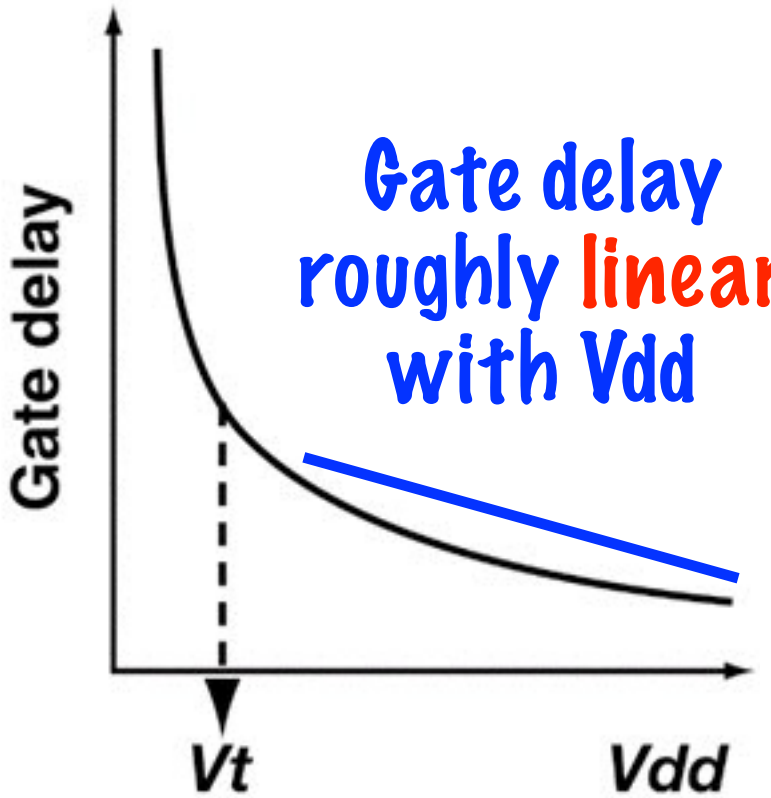
Power techniques available for project

- Parallelism and pipelining ✓
- ~~Power-down idle transistors~~
- Slow down non-critical paths ✓
- Clock gating ✓
- Data-dependent processing ✓
- ~~Thermal management~~

Cell libraries characterized at multiple V_{dd} values.

So, you can pick a different V_{dd} value for each of your implementations.

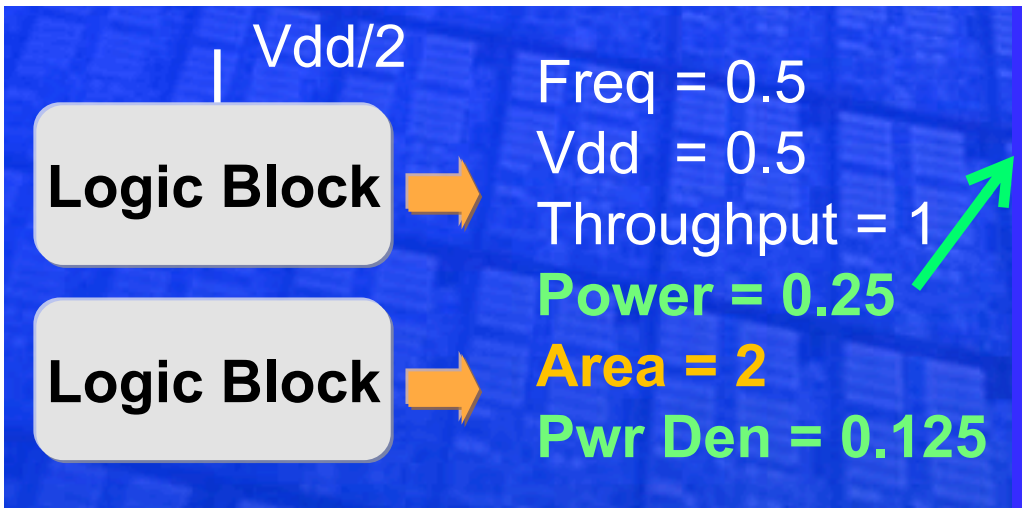
Several Vdd values in one implementation not supported.



Gate delay roughly linear with Vdd

The main trick:

Top block processes "left", bottom "right".



$P \sim \#blks \times F \times V_{dd}^2$

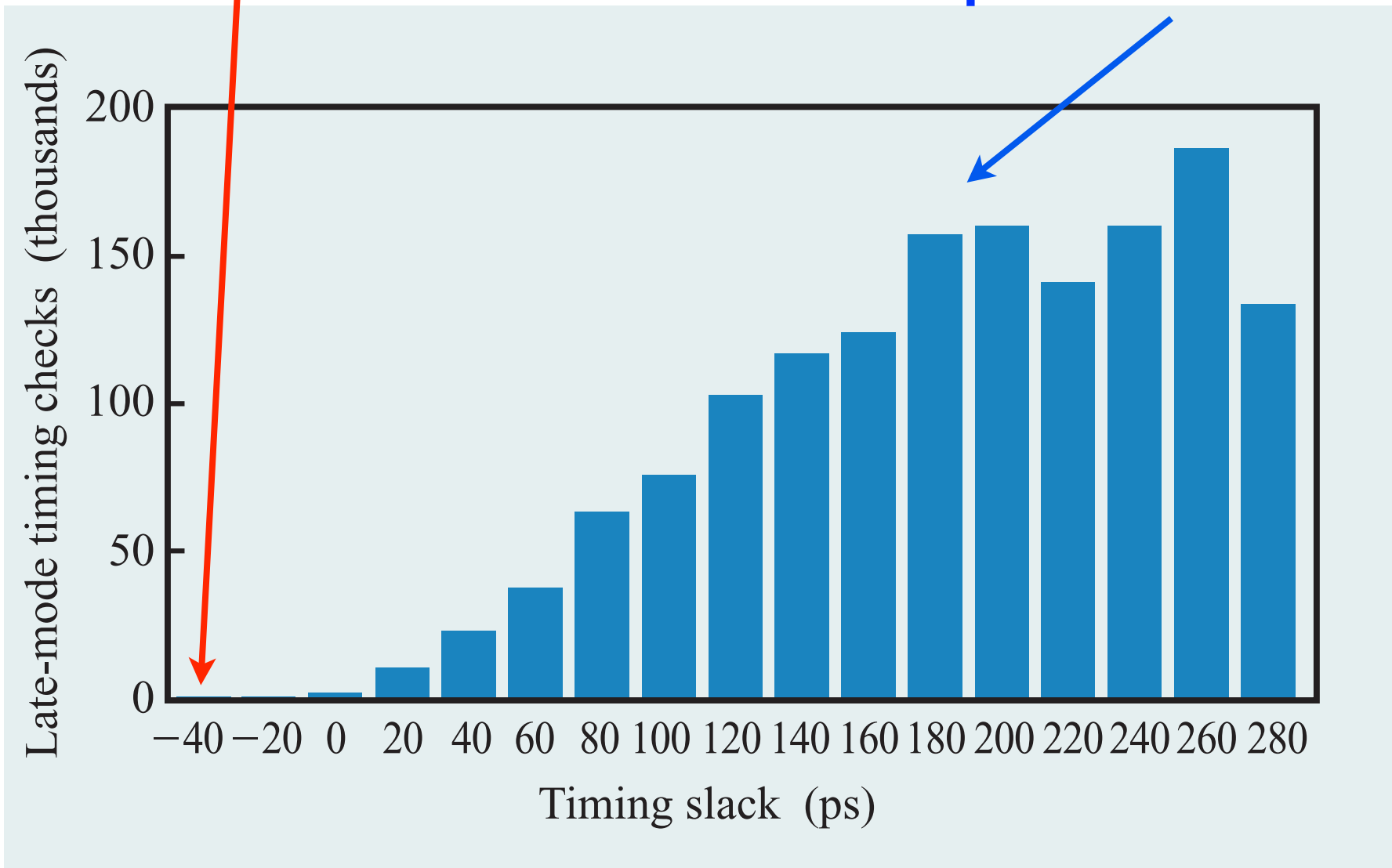
$P \sim 2 \times 1/2 \times 1/4 = 1/4$

CV^2 power only

Not by varying V_{dd}, but by cell choice

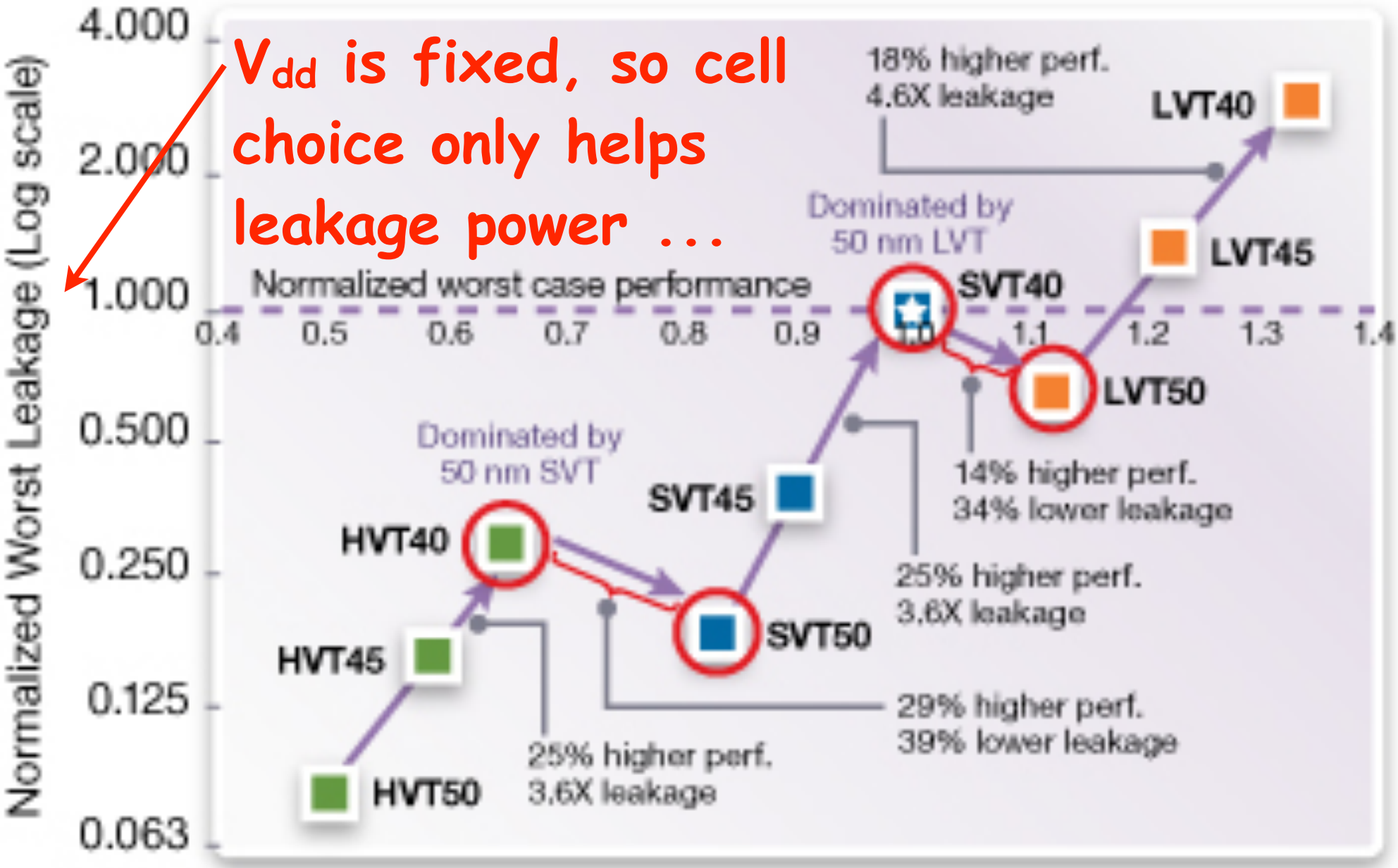
The critical path

Most wires have hundreds of picoseconds to spare.



From "The circuit and physical design of the POWER4 microprocessor", IBM J Res and Dev, 46:1, Jan 2002, J.D. Warnock et al.

(H,S,L) == High Vt, Standard Vt, Low Vt

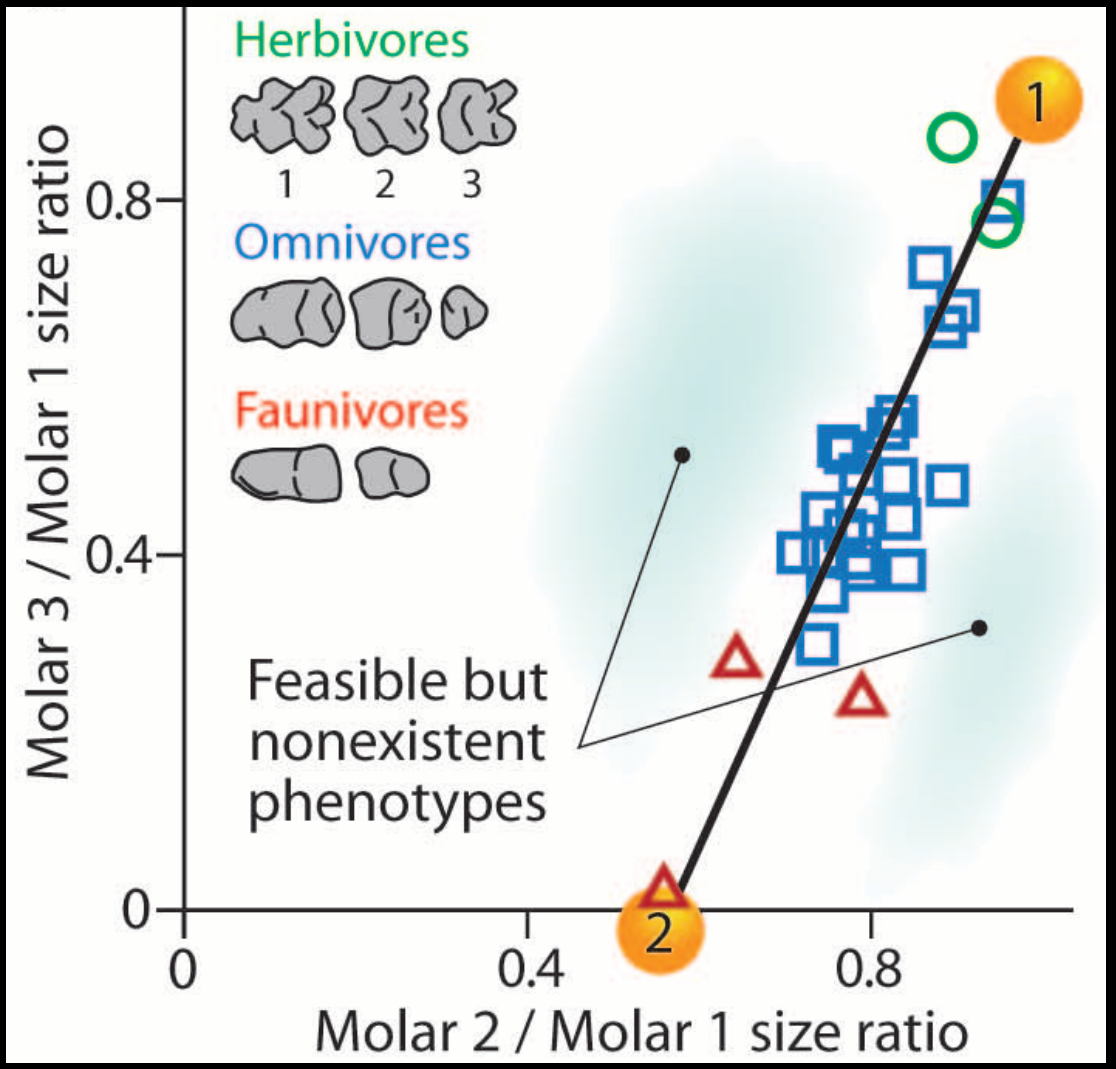


(40, 45, 50) are channel lengths (in nm)

*Source: Synopsys (Virage Logic)

Pareto Optimality

Neanderthal teeth



Evolutionary Trade-Offs, Pareto Optimality, and the Geometry of Phenotype Space

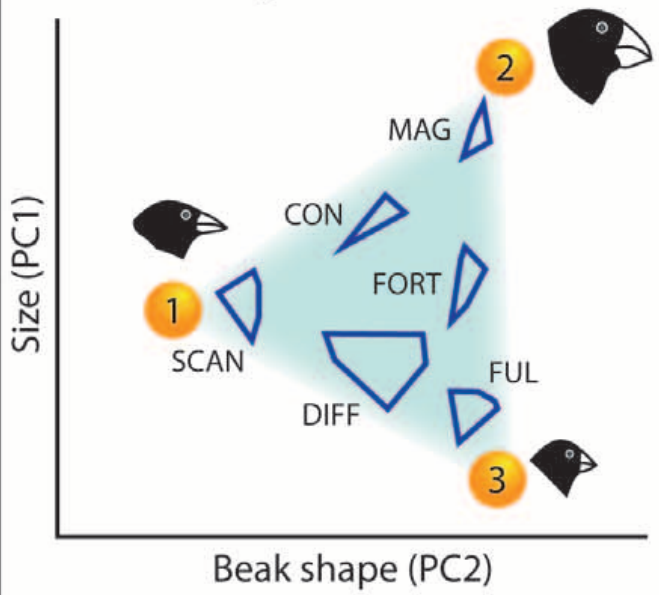
O. Shoval,[1] H. Sheftel,[1] G. Shinar,[1] Y. Hart,[1] O. Ramote,[1] A. Mayo,[1] E. Dekel,[1] K. Kavanagh,[2] U. Alon[1]*

Pareto Optimality

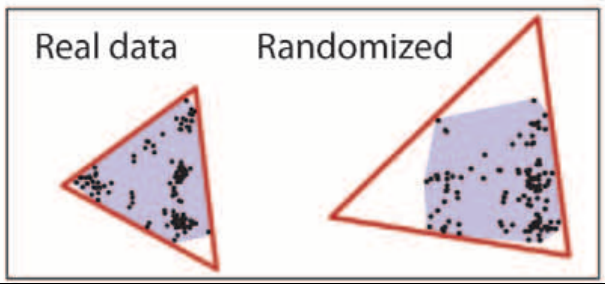
Evolutionary Trade-Offs, Pareto Optimality, and the Geometry of Phenotype Space

O. Shoval,[1] H. Sheftel,[1] G. Shinar,[1] Y. Hart,[1] O. Ramote,[1] A. Mayo,[1] E. Dekel,[1] K. Kavanagh,[2] U. Alon[1]*

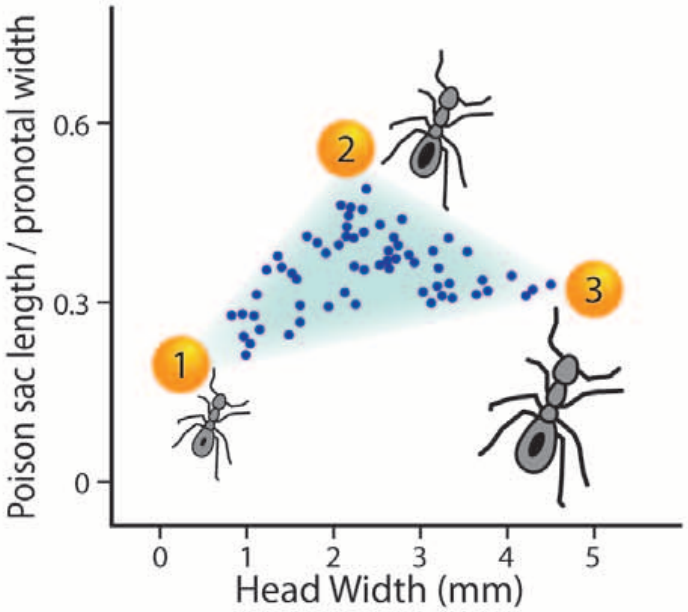
A Darwin's ground finches



	Archetypes	Task (Diet)
1	Long beak, medium body	Insects, nectar
2	Large thick beak, large body	Large, hard seeds
3	Small thick beak, small body	Small, soft seeds

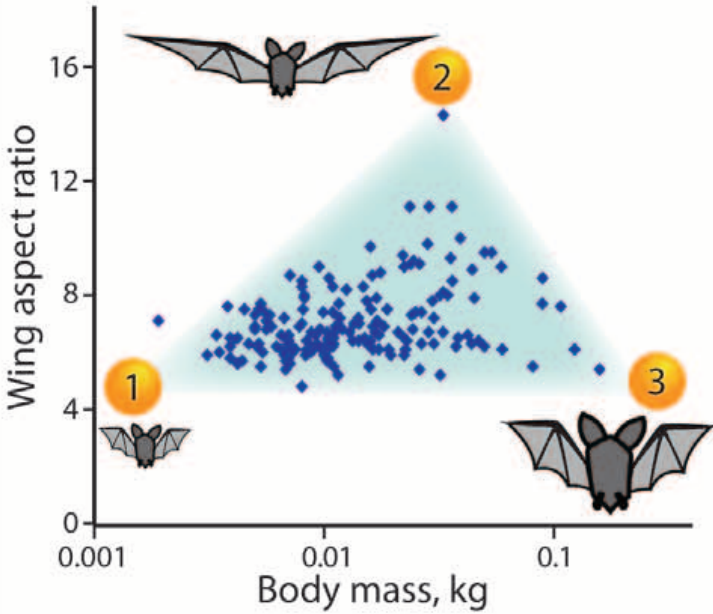


B Leaf-cutter ants (A. sexdens)



	Archetypes	Task
1	Small head width (HW), small poison sac / pronotal width (PS/PW)	Gardening / nursing
2	Medium HW, large PS/PW	Foraging
3	Large HW, small PS/PW	Soldiering

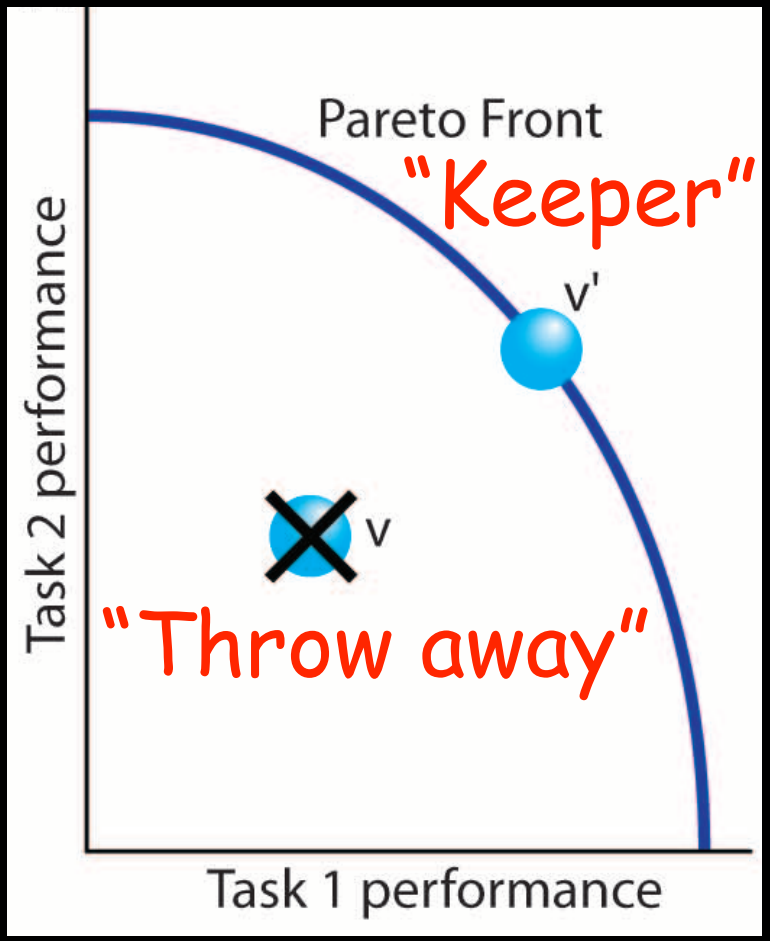
C Bats (Microchiroptera)



	Archetypes	Task
1	Low aspect ratio, small body	Prey: small insects, near vegetation
2	High aspect ratio, medium body	Prey: high flying large insects
3	Low aspect ratio, large body	Prey: animals, near vegetation

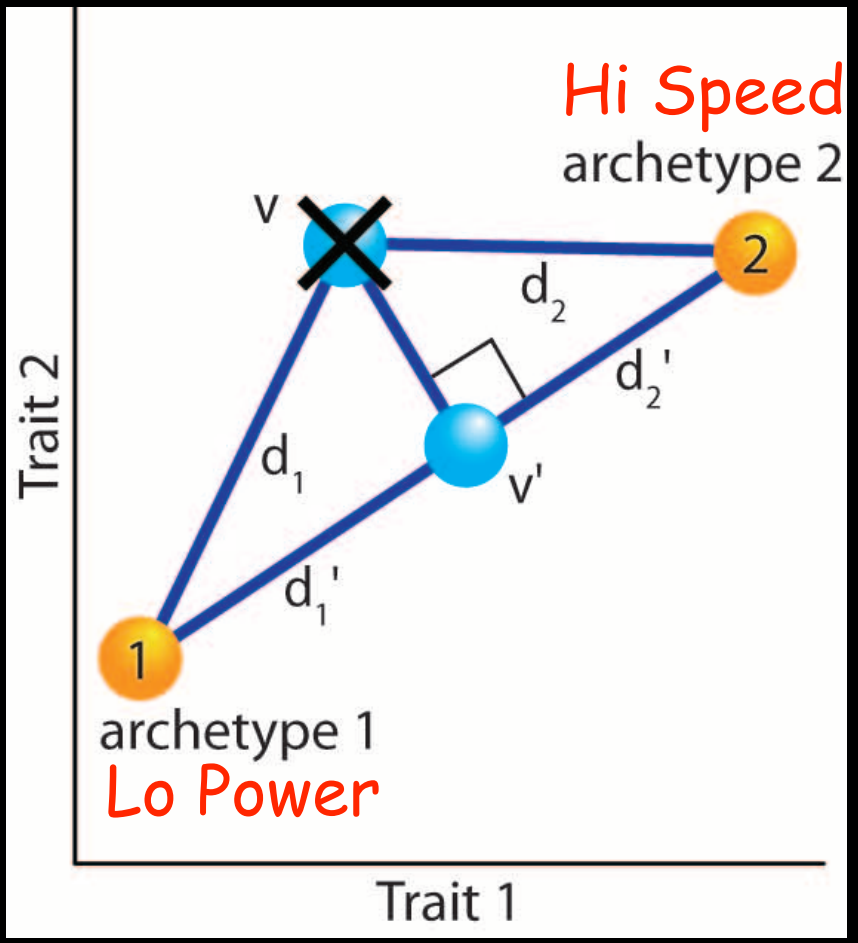
Map to our project

Energy



Clock Frequency

V_{dd}

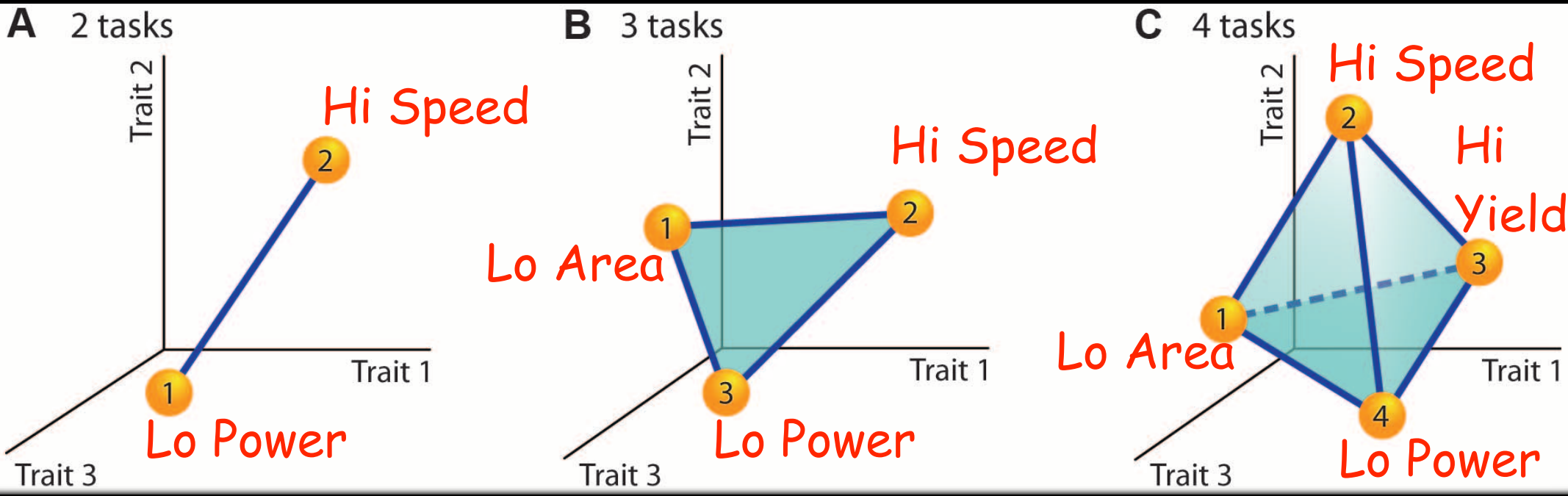


% of Low-Vt Cells

Trajectories in trait space are linear in biology ... but in chip design? Open question.

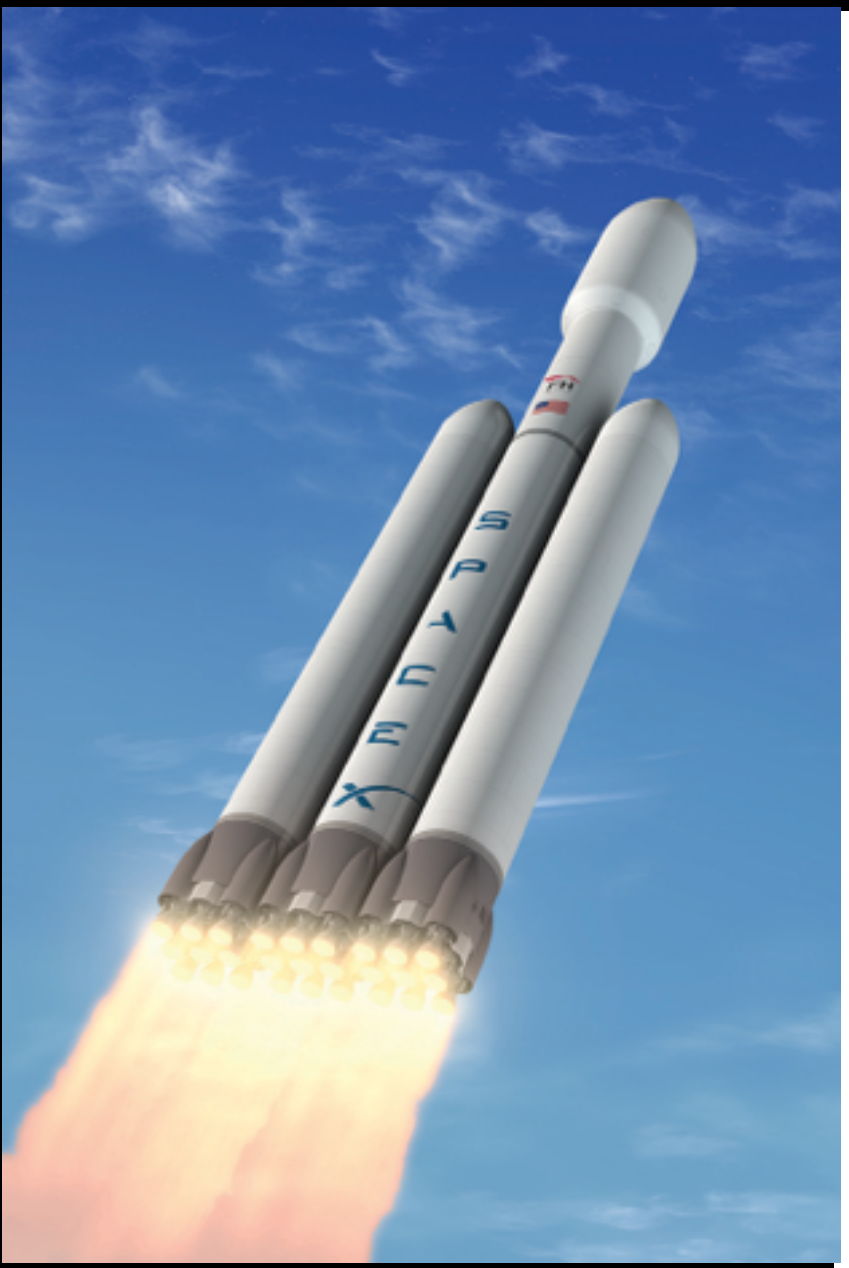
In multiple dimensions

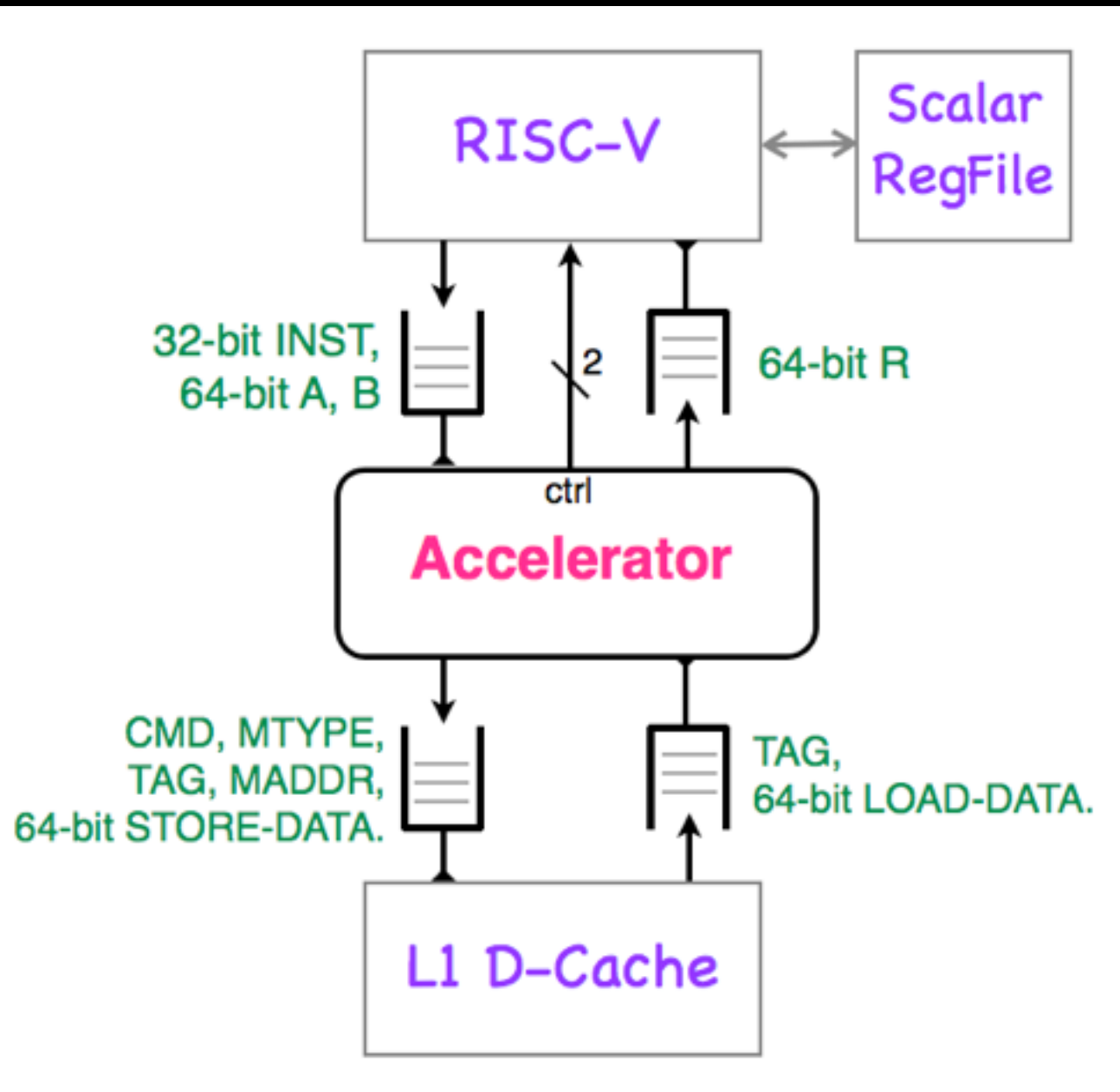
"Tasks": Power, Speed, Area, Yield

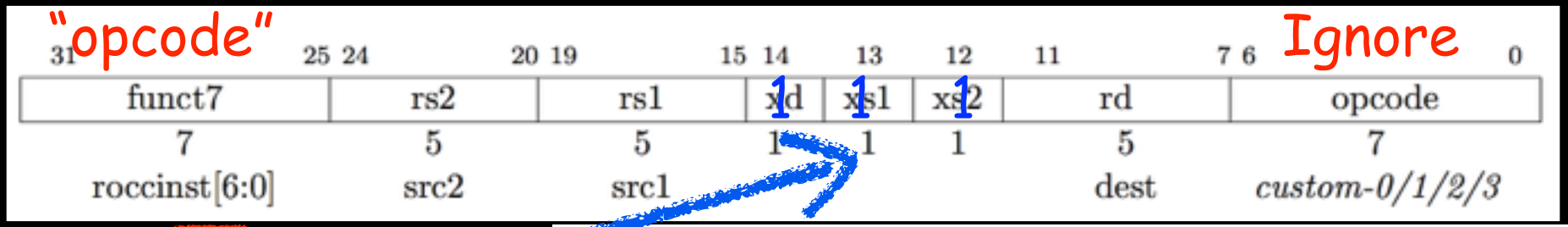


"Traits": V_{dd} choice, cell V_t choice,
choice of SRAM vs. synthesized state, etc.

Rocket Facts

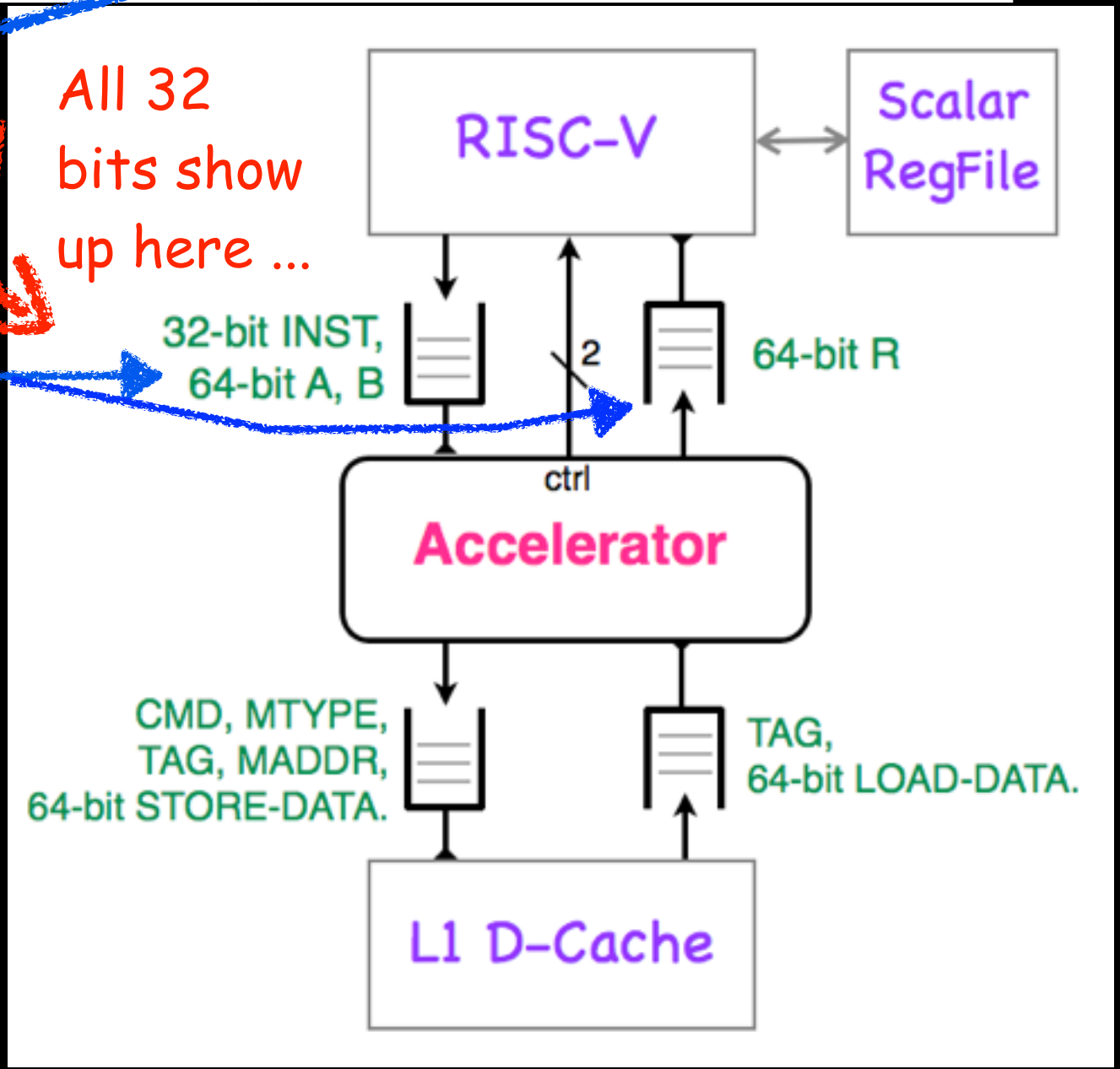




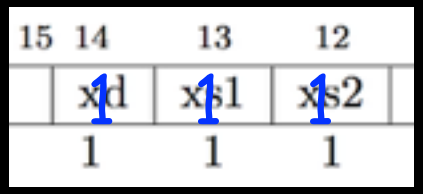


Blue "1" bits yield register queuing shown on diagram ... the 7-bit funct7 field is accelerator's 128 opcodes.

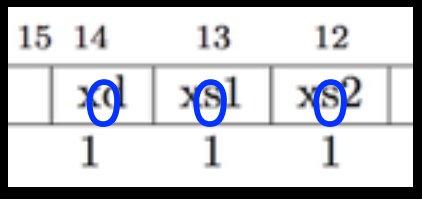
All 32 bits show up here ...



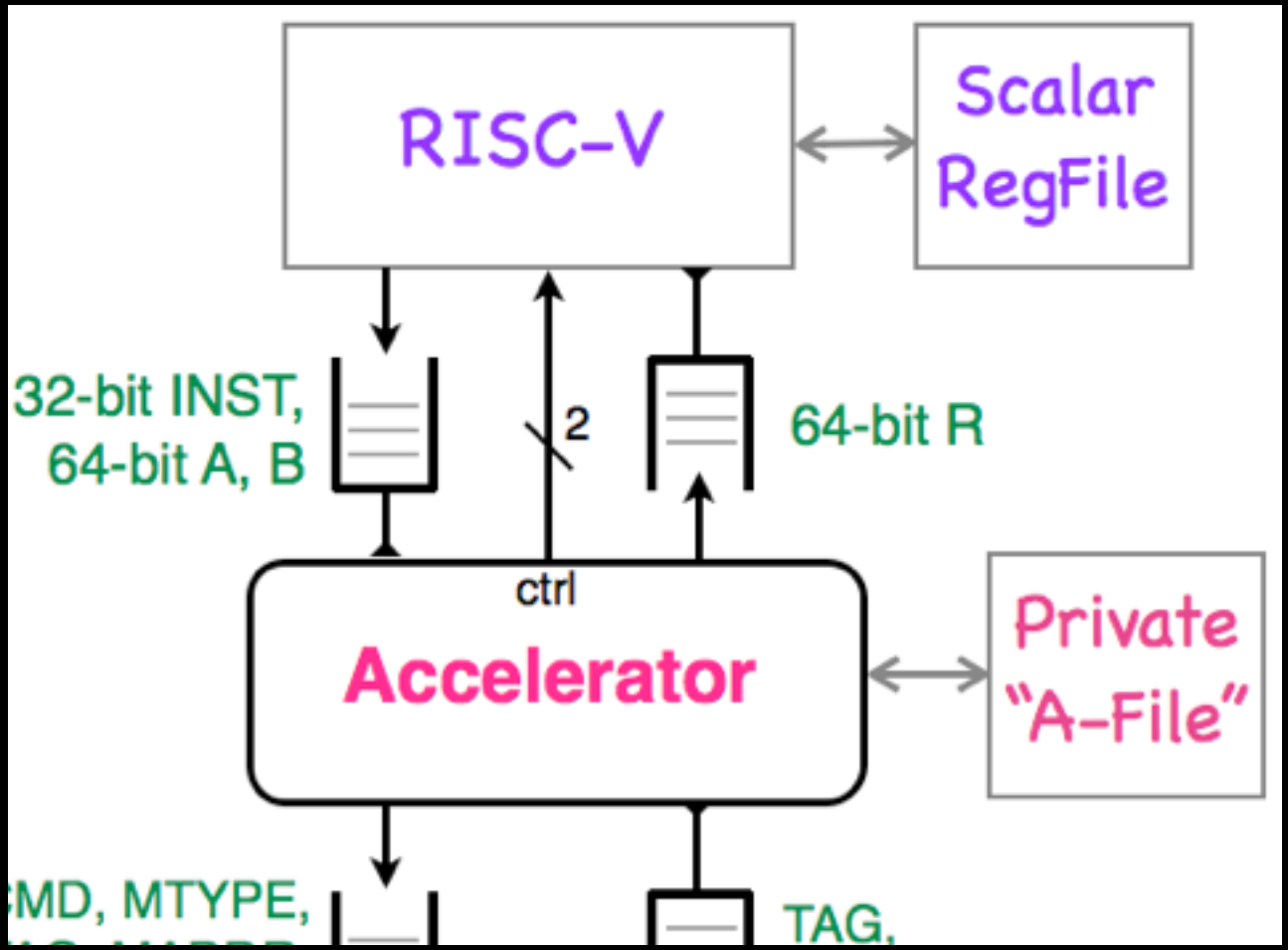
RISC-V Regfile Ops



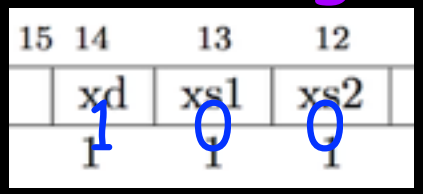
Private "A-File" Ops



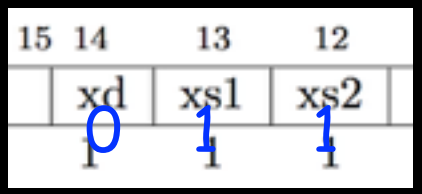
How to manage private 32-entry "A-File" register bank without using up opcode bits ...



MOV: A-file to Regfile



MOV: Regfile to A-File



D-Cache Facts

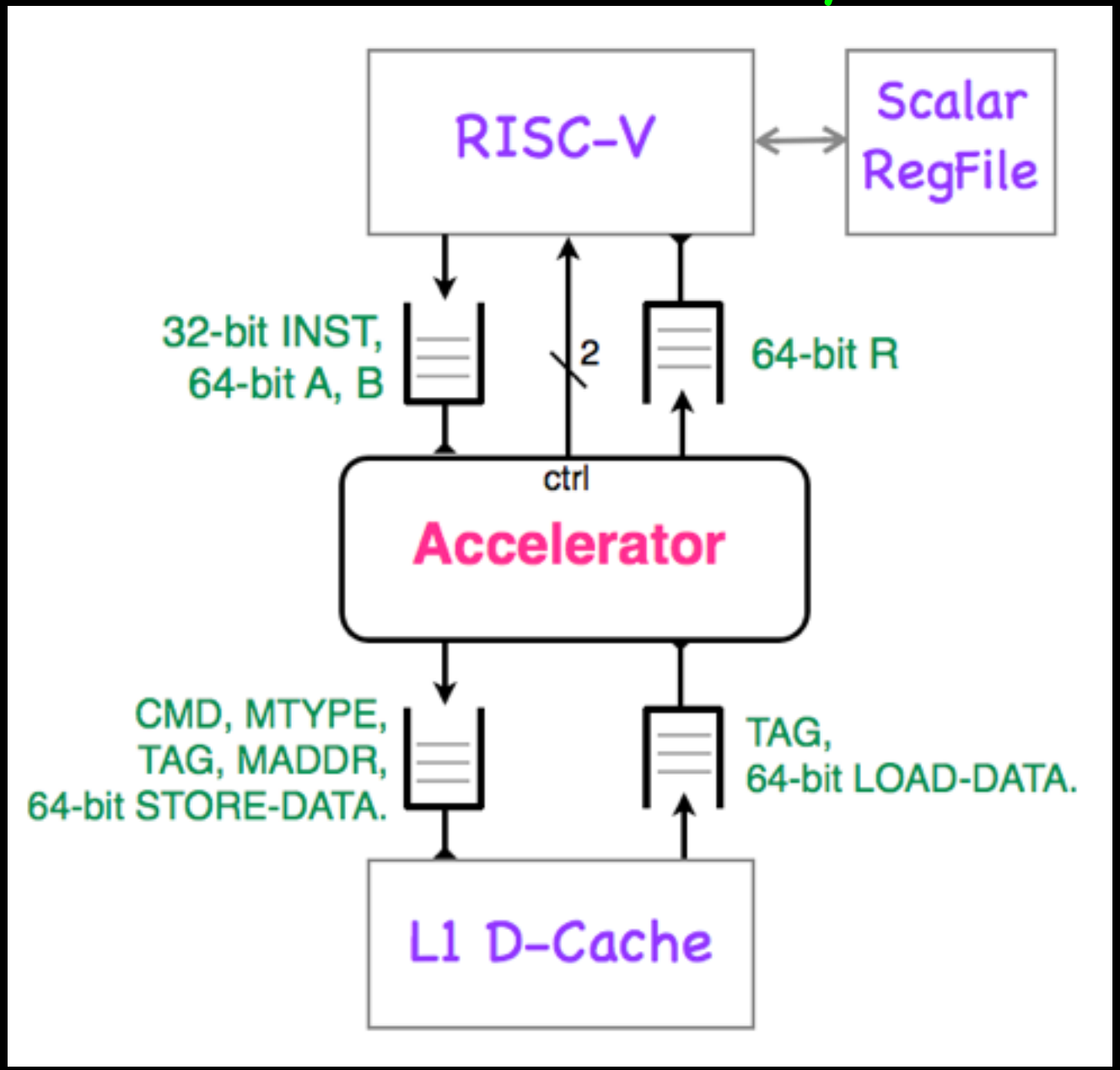
Size: 64KB L1 with 64 byte lines.

CMD: Load, Store

MTYPE:
8, 16, 32, 64 bits.

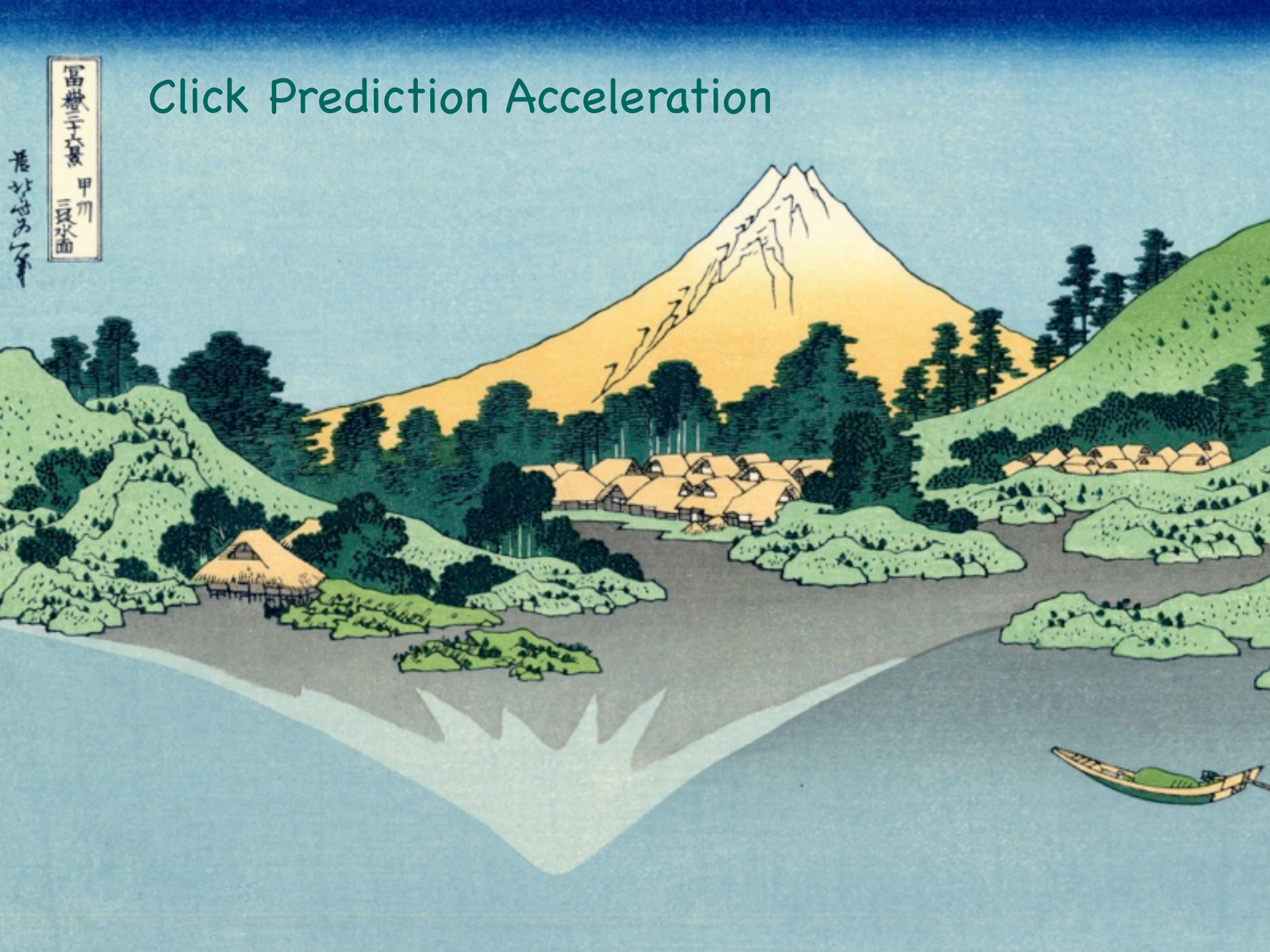
MADDR:
Align to MTYPE

TAG: 9-bits. Lets loads be OoO, up to 4 missed loads.



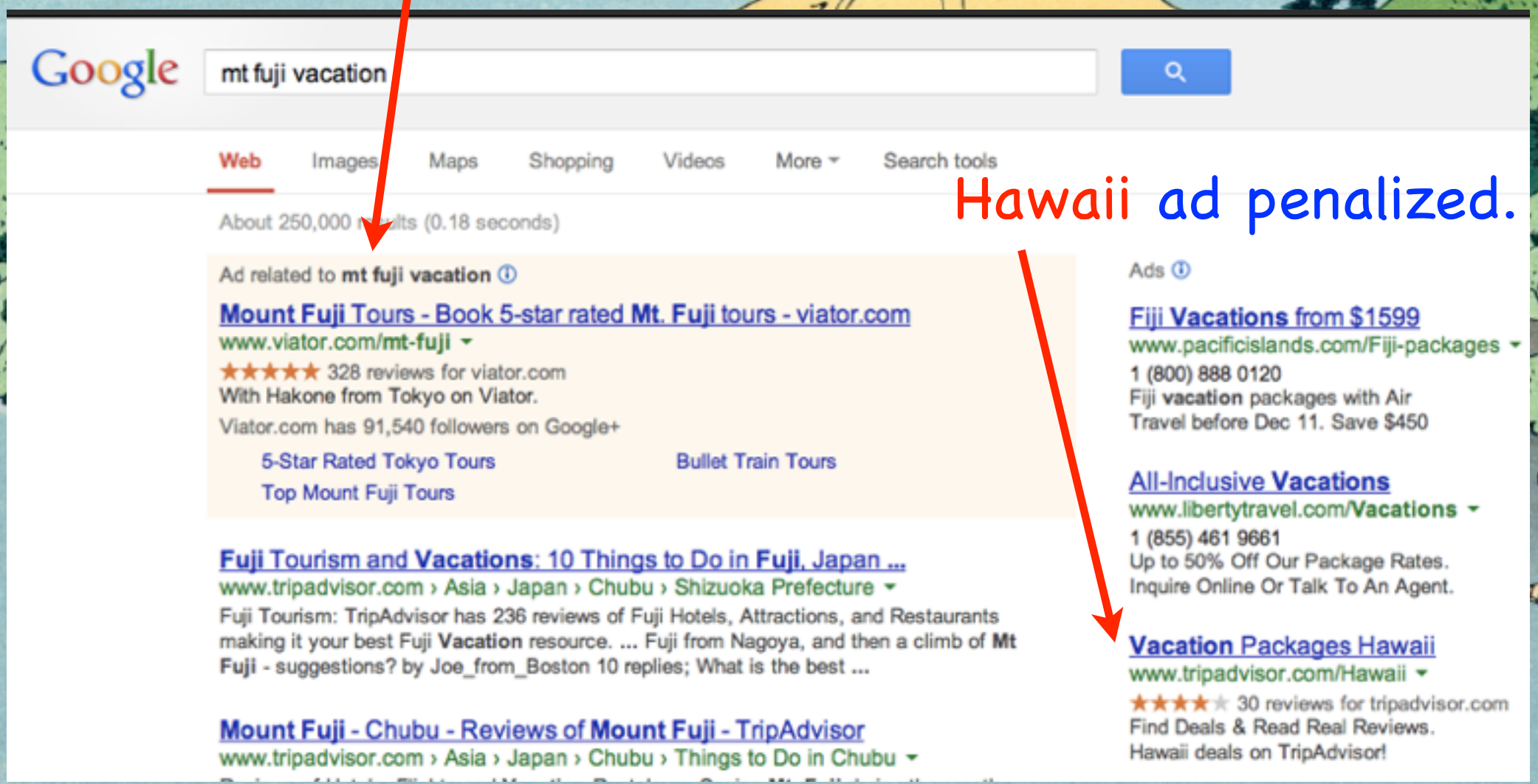
Performance: 4 cycle latency on a cache hit, 40-60 on a miss. No prefetching built in ...

Click Prediction Acceleration



Advertisers pay Google $1.47, on average, if Google Search displays their ad in response to the search term mt fuji vacation.

Since Google is only paid if the user clicks, they predict, in real time, which of the bidding ads is most likely to yield a click.



Hawaii ad penalized.

Ad Click Prediction: a View from the Trenches

H. Brendan McMahan, Gary Holt, D. Sculley, Michael Young,
Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov,
Daniel Golovin, Sharat Chikkerur, Dan Liu, Martin Wattenberg,
Arnar Mar Hrafnkelsson, Tom Boulos, Jeremy Kubica
Google, Inc.

Basic idea: Billions of "features" are developed to predict, given an ad and a search, how likely it is that the searcher will click on the ad.

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

a vector: feature values for the search.
b vector: feature values for the ad.

Example 50 features: Does geo info indicate that the searcher is in the state of (1) Alabama? (2) Alaska .... (50) Wyoming. Binary, sparse features.

Ad Click Prediction: a View from the Trenches

H. Brendan McMahan, Gary Holt, D. Sculley, Michael Young,
Dietmar Ebner, Julian Grady, Lan Nie, Todd Phillips, Eugene Davydov,
Daniel Golovin, Sharat Chikkerur, Dan Liu, Martin Wattenberg,
Arnar Mar Hrafnkelsson, Tom Boulos, Jeremy Kubica
Google, Inc.

To rank each ad: Take the dot product of a and b for each ad, give the highest-valued ads placement.

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

"n" here is in the billions, but non-zero "a" and "b" values are in the thousands. This real-time system needs to exploit the sparsity to perform well.

A good candidate problem for an accelerator.

Assumptions

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

- "a" and "b" are binary. The click prediction system uses fixed-point "b" and integer "a". Binary vectors let us focus on deeper issues.
- Sorted 32-bit index list, zero-terminated.
  - a: 1, 13, 1827, 2000, 1938475, 0 (24 bytes)
  - b: 12, 13, 2000, 1602938, 0 (20 bytes)
  - Dot product is 2: (13 and 2000 match)

One Instruction

SBDT dest_reg, a_reg, b_reg

a_reg: Holds 64-bit memory address pointing to the first byte of "a" list.

b_reg: Holds 64-bit memory address pointing to the first byte of "b" list.

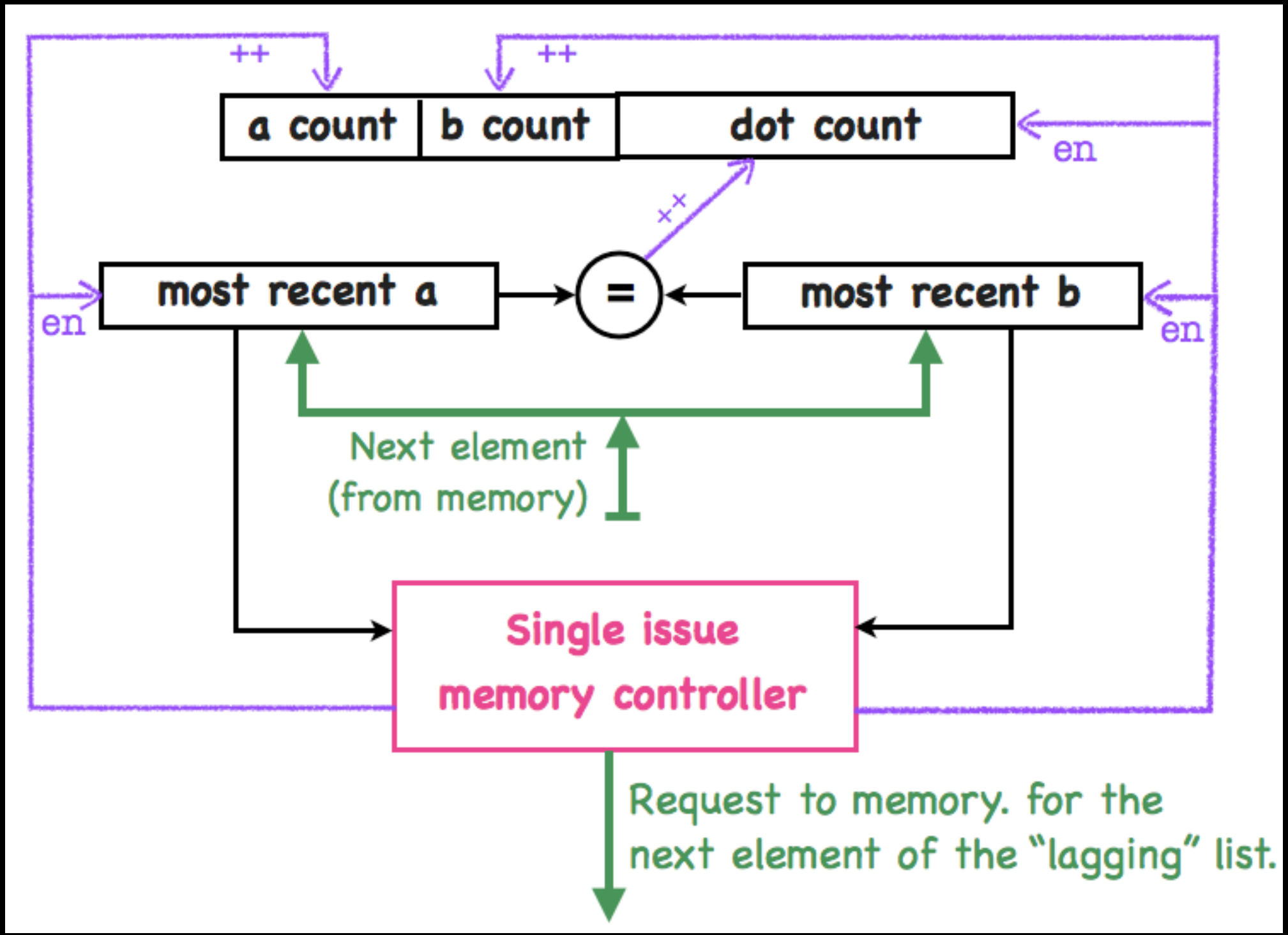
dest_reg:

# of list elements. saturating 16-bit unsigned ints.

32-bit unsigned int

$$\mathbf{a} \cdot \mathbf{b} = \sum_{i=1}^{n} a_i b_i = a_1 b_1 + a_2 b_2 + \cdots + a_n b_n$$

Simple implementation



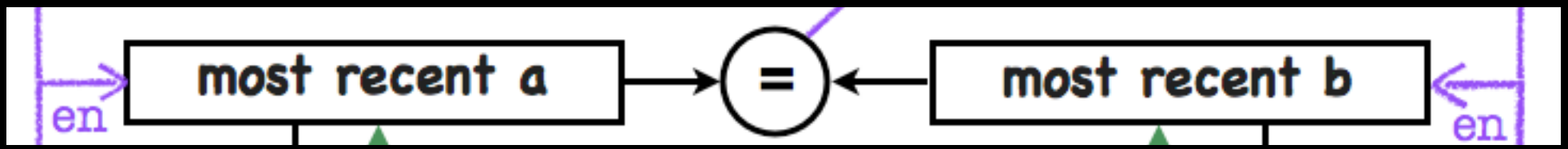
Improving performance

Do more compares/clock.

Tactics:

Memory controller should strive to reach one 64-bit load per clock (max possible).

Replace most-recent registers by list structures. Comparator? Parallelize.



Beyond one 64-bit load per clock

Delta data compression

Turn a: 200, 300, 301, 400, 402, 0

into a: 200, 100, 1, 99, 2, 0

Store with MIDI compression scheme:

```
Coding 0-127:

  Encoded form: 0ddddddd
  Decoded form: 00000000 00000000 00000000 0ddddddd

Coding 128-16383:

    Encoded form: 1ccccccc 0ddddddd
    Decoded form: 00000000 00000000 00ccccccc cddddddd
```

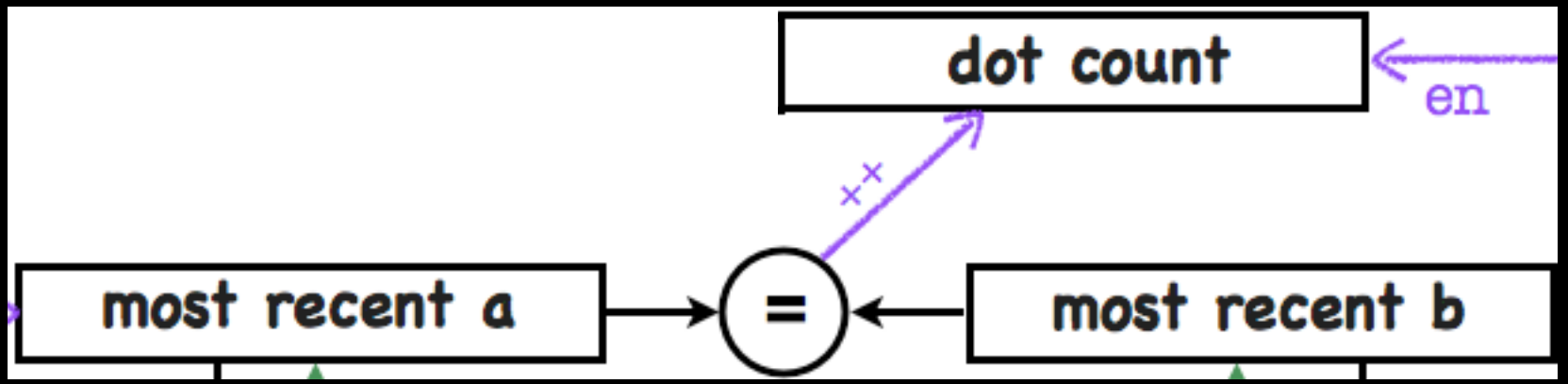
For "a" list above, 3.4x memory bandwidth gain.

Improving performance

Do fewer compares/instruction.

Observations:

To compute the correct dot product, we only need to load list elements that match.



Because lists are sorted, a speculative memory controller can "skip" many loads.

Works best if RISC-V passes in list lengths.

Project variants

- Let "b" vector be sparse fixed-point.
- Unsorted 32-bit index list, zero-terminated.
  Changes many aspects of the problem.
- Change sparse vector representation to be binary bit streams stored in memory:

  a: 00000000100000000100000

  Accelerator defines a compression method that differs from "indices list".

Break

Play:

Gabriel Cramer 1704-1752

Cramer's Rule

Solve this matrix equation:

$$\begin{pmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}$$

$$A \qquad x \qquad b$$

With determinants:

To solve for x_i: Substitute b for column i in the numerator ...

$$x_i = \frac{\det(A_i)}{\det(A)} \qquad i = 1, \ldots, n$$

$$x_1 = \frac{\begin{vmatrix} b_1 & a_{12} \\ b_2 & a_{22} \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}} = \frac{b_1 a_{22} - b_2 a_{12}}{a_{11} a_{22} - a_{12} a_{21}}$$

$$x_2 = \frac{\begin{vmatrix} a_{11} & b_1 \\ a_{21} & b_2 \end{vmatrix}}{\begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix}} = \frac{b_2 a_{11} - b_1 a_{21}}{a_{11} a_{22} - a_{12} a_{21}}$$

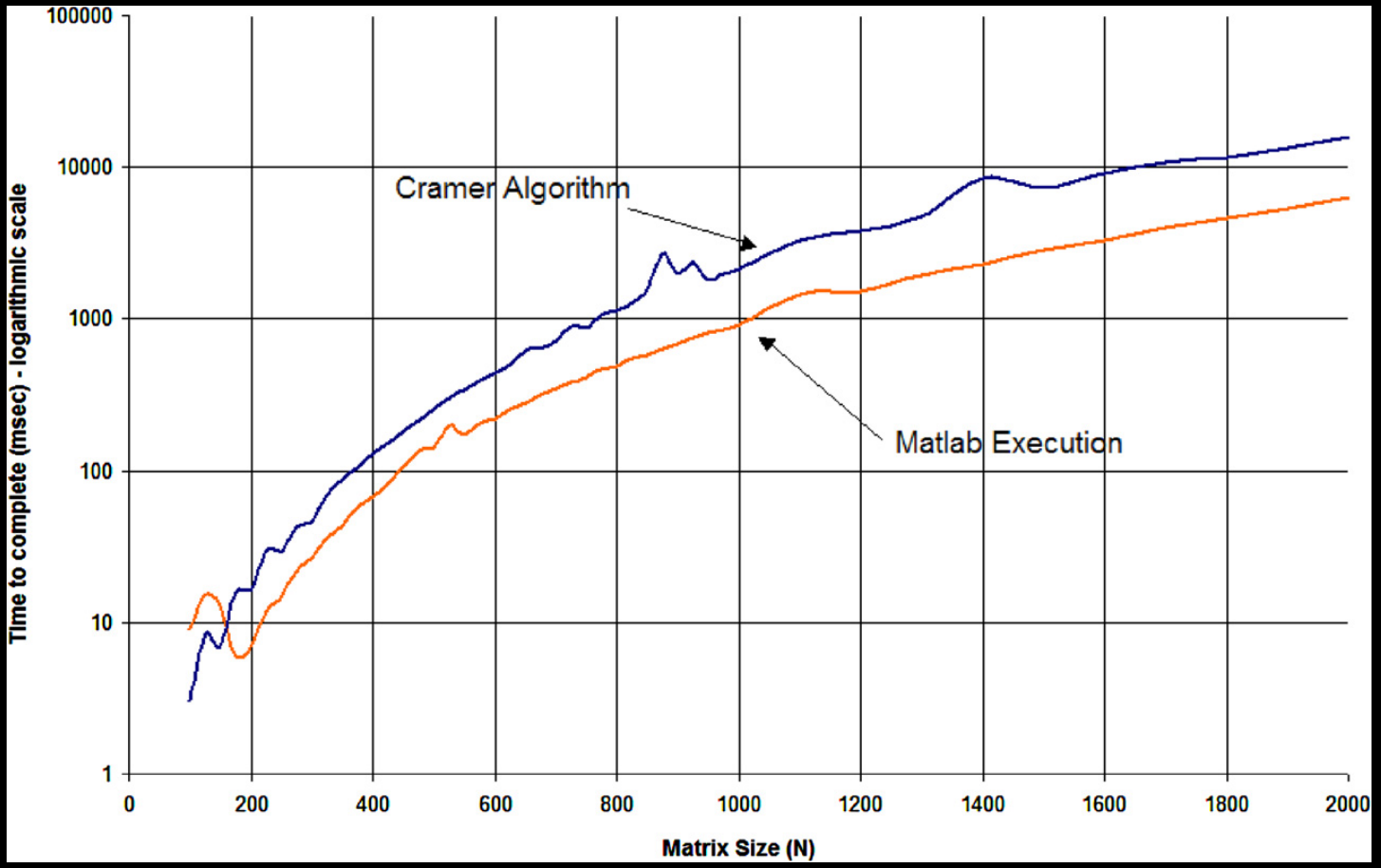
Useful?

A condensation-based application of Cramer's rule for solving large-scale linear systems

Ken Habgood *, Itamar Arel

Department of Electrical Engineering and Computer Science, The University of Tennessee, Knoxville, TN, USA

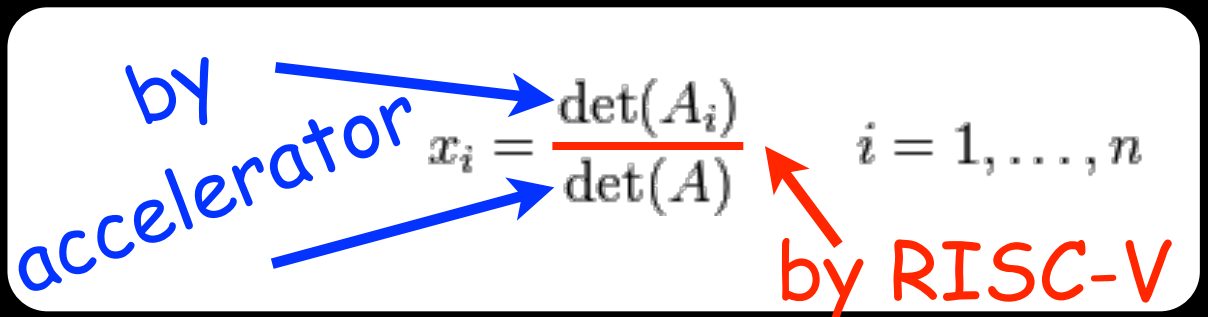
Recent work shows how to make Cramer's rule scale and be stable for large systems:



So, let's make an accelerator based on it ...

Observations

Determinant works on a matrix, but returns a scalar. We use accelerator instructions to compute determinants, and let RISC-V compute the x vector by doing the divides.



Determinants of matrices with integer coefficients can be computed exactly, with only integer multiplies and adds.
So, we restrict our accelerator accordingly.

Chió's Trick

For n = 3, computes determinant of a 3 x 3 matrix by computing the 2 x 2 determinant of four 2 x 2 determinant results.

$$\det(A) = \frac{1}{a_{11}^{n-2}} \begin{vmatrix} \begin{vmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{13} \\ a_{21} & a_{23} \end{vmatrix} & \cdots & \begin{vmatrix} a_{11} & a_{1n} \\ a_{21} & a_{2n} \end{vmatrix} \\ \begin{vmatrix} a_{11} & a_{12} \\ a_{31} & a_{32} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{13} \\ a_{31} & a_{33} \end{vmatrix} & \cdots & \begin{vmatrix} a_{11} & a_{1n} \\ a_{31} & a_{3n} \end{vmatrix} \\ \vdots & \vdots & \ddots & \vdots \\ \begin{vmatrix} a_{11} & a_{12} \\ a_{n1} & a_{n2} \end{vmatrix} & \begin{vmatrix} a_{11} & a_{13} \\ a_{n1} & a_{n3} \end{vmatrix} & \cdots & \begin{vmatrix} a_{11} & a_{1n} \\ a_{n1} & a_{nn} \end{vmatrix} \end{vmatrix}$$

Can be ignored (cancels out of Ax=b)

$$x_i = \frac{\det(A_i)}{\det(A)} \qquad i = 1, \ldots, n$$

We can reuse the "leaf" 2x2 determinants when we compute the full set of det(Ai) and det(A).

Reuse

Implicit architected state for the accelerator can be used to store and reuse partial results ...

$$\begin{pmatrix} a_{11} & a_{12} & a_{13} \\ a_{12} & a_{22} & a_{23} \\ a_{21} & a_{32} & a_{33} \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix}$$

Color coding shows reuse.

$$\det A = \begin{vmatrix} (a_{11}\,a_{22} - a_{12}\,a_{21}) & (a_{11}\,a_{23} - a_{13}\,a_{21}) \\ (a_{11}\,a_{32} - a_{12}\,a_{31}) & (a_{11}\,a_{33} - a_{13}\,a_{31}) \end{vmatrix}$$

$$\det A_2 = \begin{vmatrix} (a_{11}\,b_2 - b_1\,a_{21}) & (a_{11}\,a_{23} - a_{13}\,a_{21}) \\ (a_{11}\,b_3 - b_1\,a_{31}) & (a_{11}\,a_{33} - a_{13}\,a_{31}) \end{vmatrix}$$

$$\det A_3 = \begin{vmatrix} (a_{11}\,a_{22} - a_{12}\,a_{21}) & (a_{11}\,b_2 - b_1\,a_{21}) \\ (a_{11}\,a_{32} - a_{12}\,a_{31}) & (a_{11}\,b_3 - b_1\,a_{31}) \end{vmatrix}$$

Two instructions

First, it clears all implicit state

- DETA dest_reg, a_reg, len_reg
  - a_reg: 64-bit memory address of *A* matrix.
  - len_reg: Holds the n of the n x n *A* matrix.
  - dest_reg: Return register for det(*A*).
    Retains n and partial results for *A*.
- DETAI dest_reg, b_reg, col_reg
  - b_reg: 64-bit memory address of b vector.
  - col_reg: The "i" (column) for det(A_i)
  - dest_reg: Return register for det(A_i).
    Adds to partial results (for A_i).

SIMD Instructions

First Pentium with SIMD instructions. Released in 1996.
Commemorative keychain, using a die that failed test.

The "right" complexity for your project ...

PCMPESTRM — Packed Compare Explicit Length Strings, Return Mask

Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
66 0F 3A 60 /r imm8 PCMPESTRM *xmm1, xmm2/m128, imm8*	RMI	V/V	SSE4_2	Perform a packed comparison of string data with explicit lengths, generating a mask, and storing the result in *XMM0*
VEX.128.66.0F3A.WIG 60 /r ib VPCMPESTRM *xmm1, xmm2/m128, imm8*	RMI	V/V	AVX	Perform a packed comparison of string data with explicit lengths, generating a mask, and storing the result in *XMM0*.

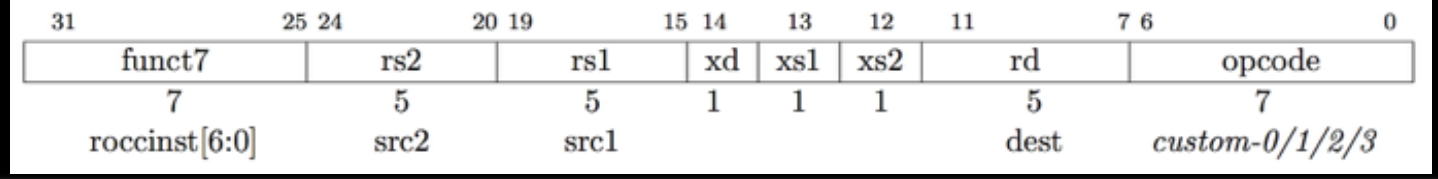
PCMPISTRI — Packed Compare Implicit Length Strings, Return Index

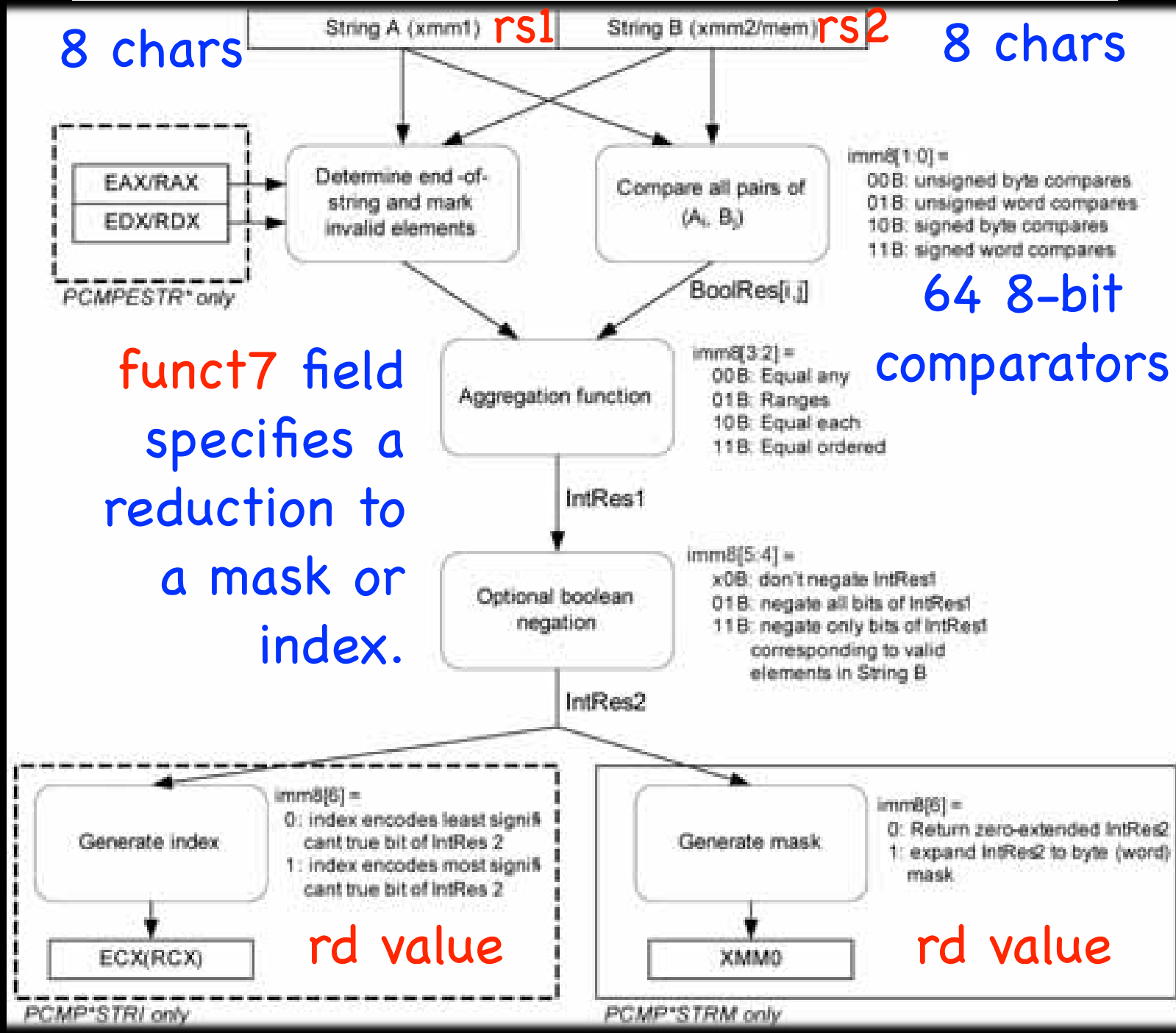
Opcode/ Instruction	Op/ En	64/32 bit Mode Support	CPUID Feature Flag	Description
66 0F 3A 63 */r imm8* PCMPISTRI *xmm1, xmm2/m128, imm8*	RM	V/V	SSE4_2	Perform a packed comparison of string data with implicit lengths, generating an index, and storing the result in ECX.
VEX.128.66.0F3A.WIG 63 /r ib VPCMPISTRI *xmm1, xmm2/m128, imm8*	RM	V/V	AVX	Perform a packed comparison of string data with implicit lengths, generating an index, and storing the result in ECX.

Use m64 instead of m128, to match RISC-V register size.

RISC-V'd

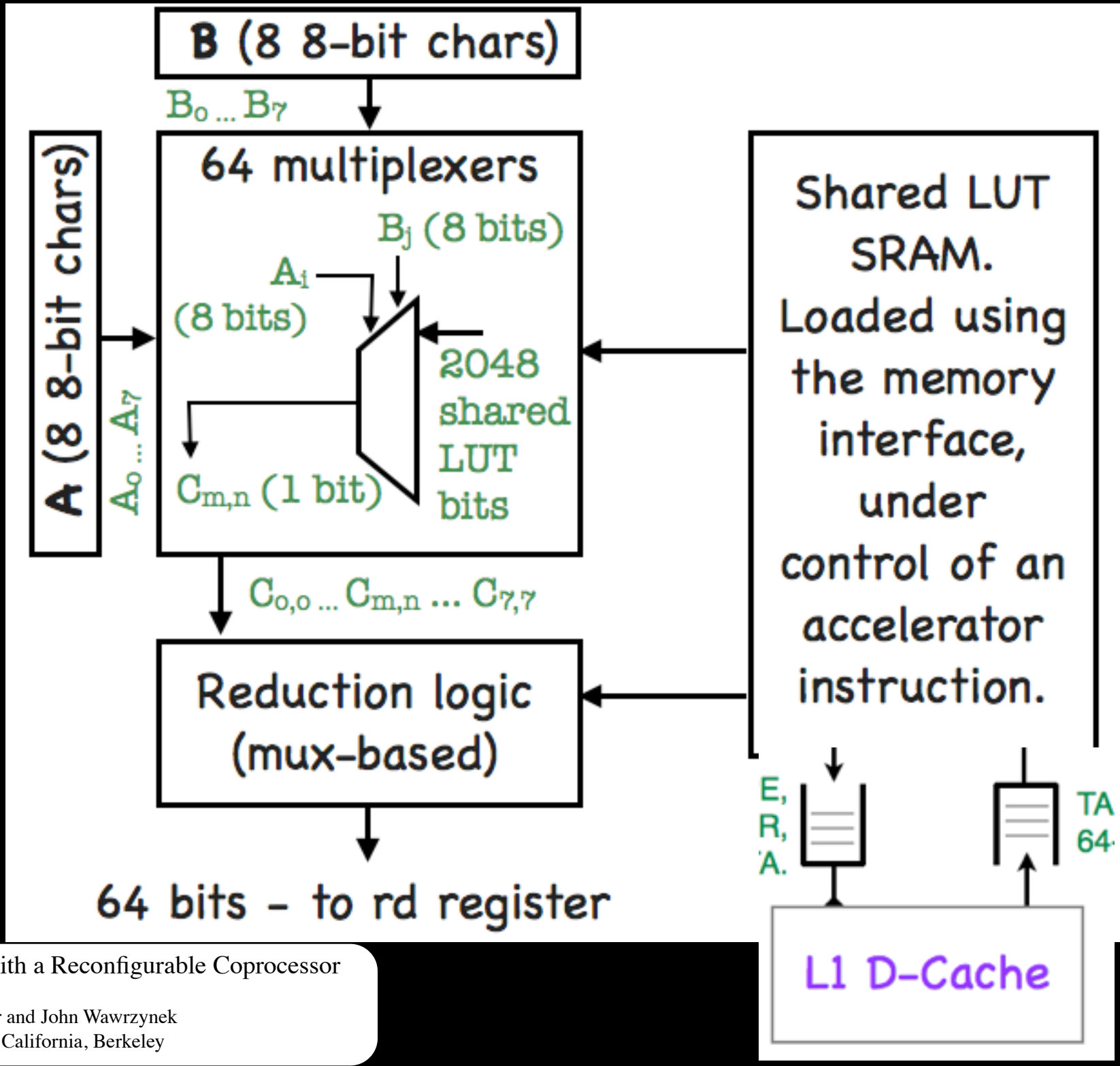
By using an existing ISA spec, you can focus on Pareto tradeoffs and micro-architecture





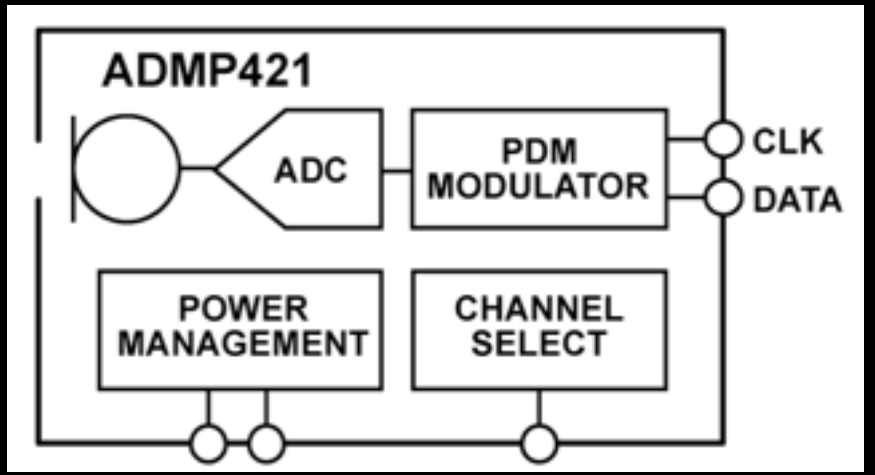
GARP'd

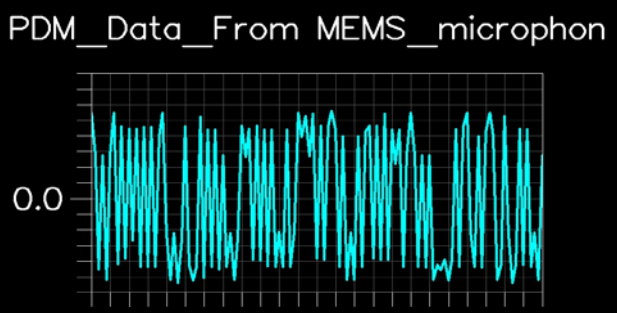
You could also take Intel idea as a starting point for a high level redesign ...

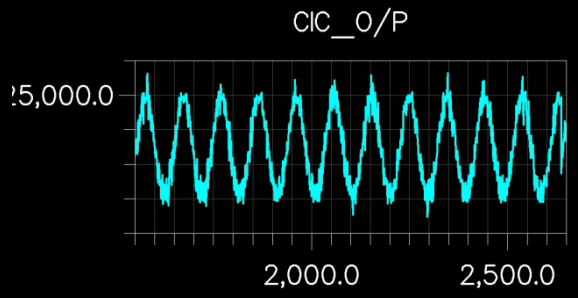


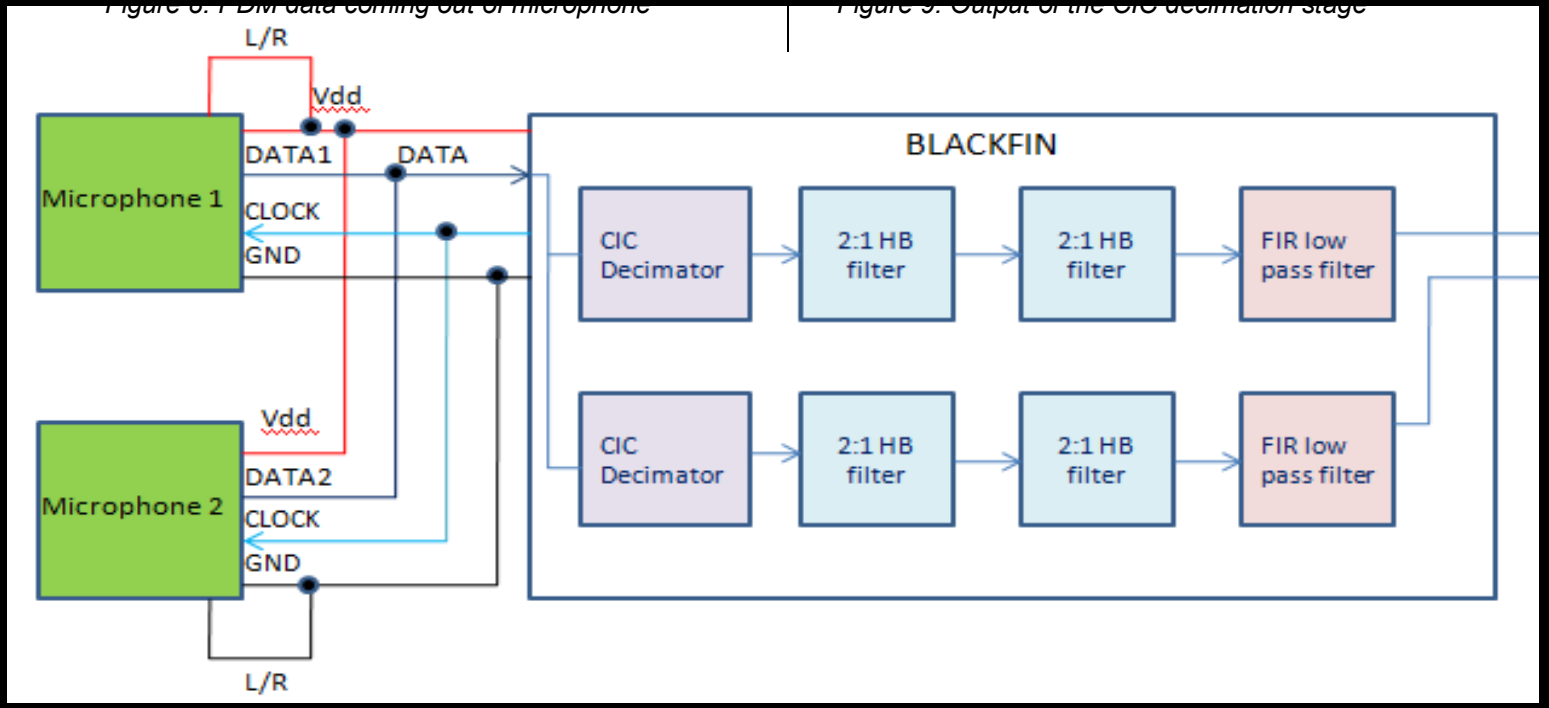
Quick Idea #1

MEMS microphone post-processing accelerator









Quick Idea #2-6

A 280 mV-to-1.1 V 256b Reconfigurable SIMD Vector Permutation Engine With 2-Dimensional Shuffle in 22 nm Tri-Gate CMOS

Steven K. Hsu, *Member, IEEE*, Amit Agarwal, *Member, IEEE*, Mark A. Anders, *Member, IEEE*,

Thin Servers with Smart Pipes: Designing SoC Accelerators for Memcached

Kevin Lim
HP Labs

David Meisner
Facebook

Ali G. Saidi
ARM R&D

Systolic Sorting on a Mesh-Connected Network

HANS-WERNER LANG, MANFRED SCHIMMLER,
HARTMUT SCHMECK, AND HEIKO SCHRÖDER

FINITE AUTOMATA BASED COMPRESSION OF BI-LEVEL AND SIMPLE COLOR IMAGES

KAREL CULIK II† and VLADIMIR VALENTA

Department of Computer Science, University of South Carolina, Columbia, SC 29208, U.S.A.
e-mail: culik@cs.scarolina.edu

Convolution Engine: Balancing Efficiency & Flexibility in Specialized Computing

Wajahat Qadeer, Rehan Hameed, Ofer Shacham,
Preethi Venkatesan, Christos Kozyrakis, Mark A. Horowitz