

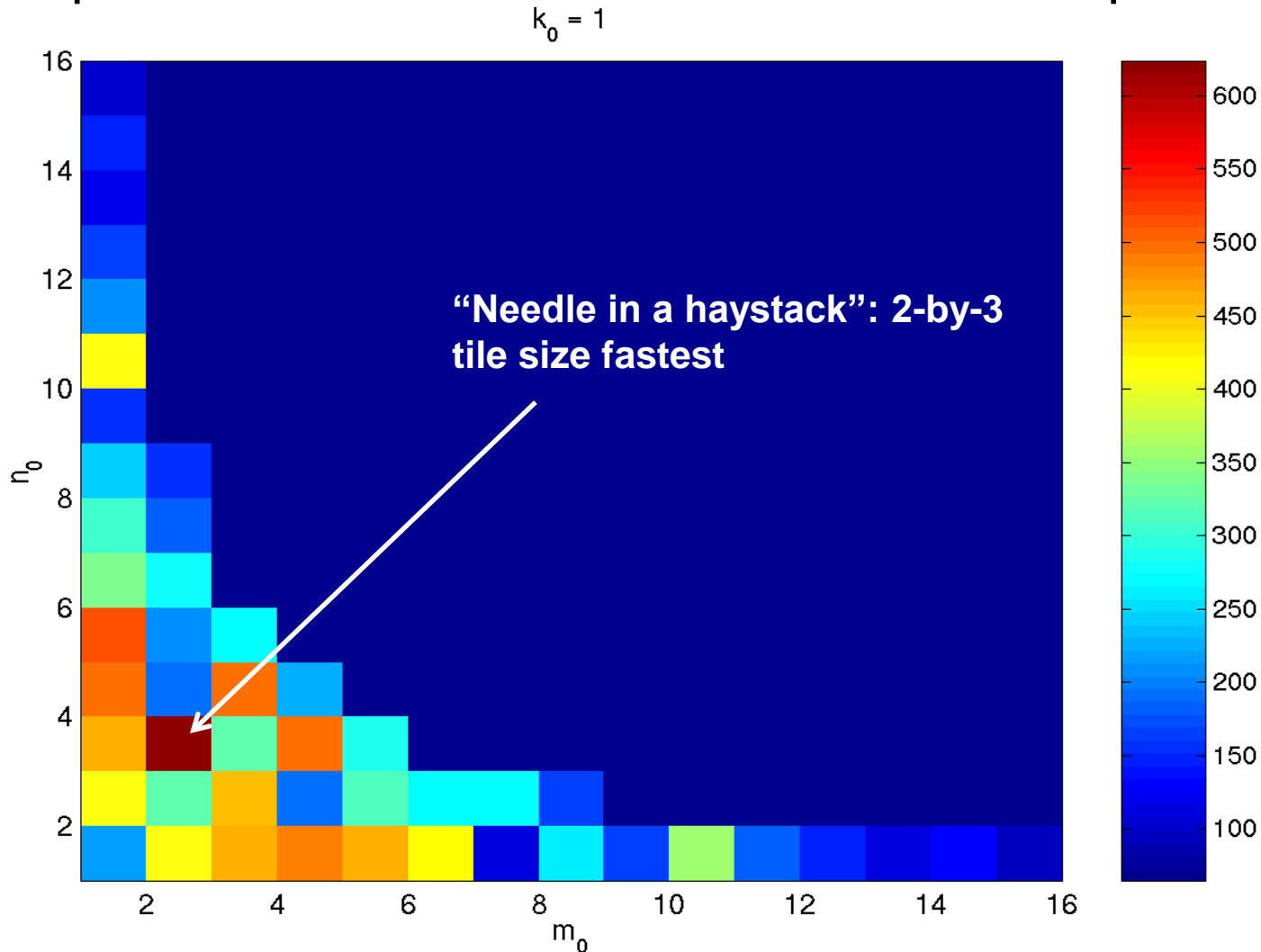
PySKI:

THE PYTHON SPARSE KERNEL INTERFACE

Erin Carson
Ben Carpenter
Armando Fox
James Demmel

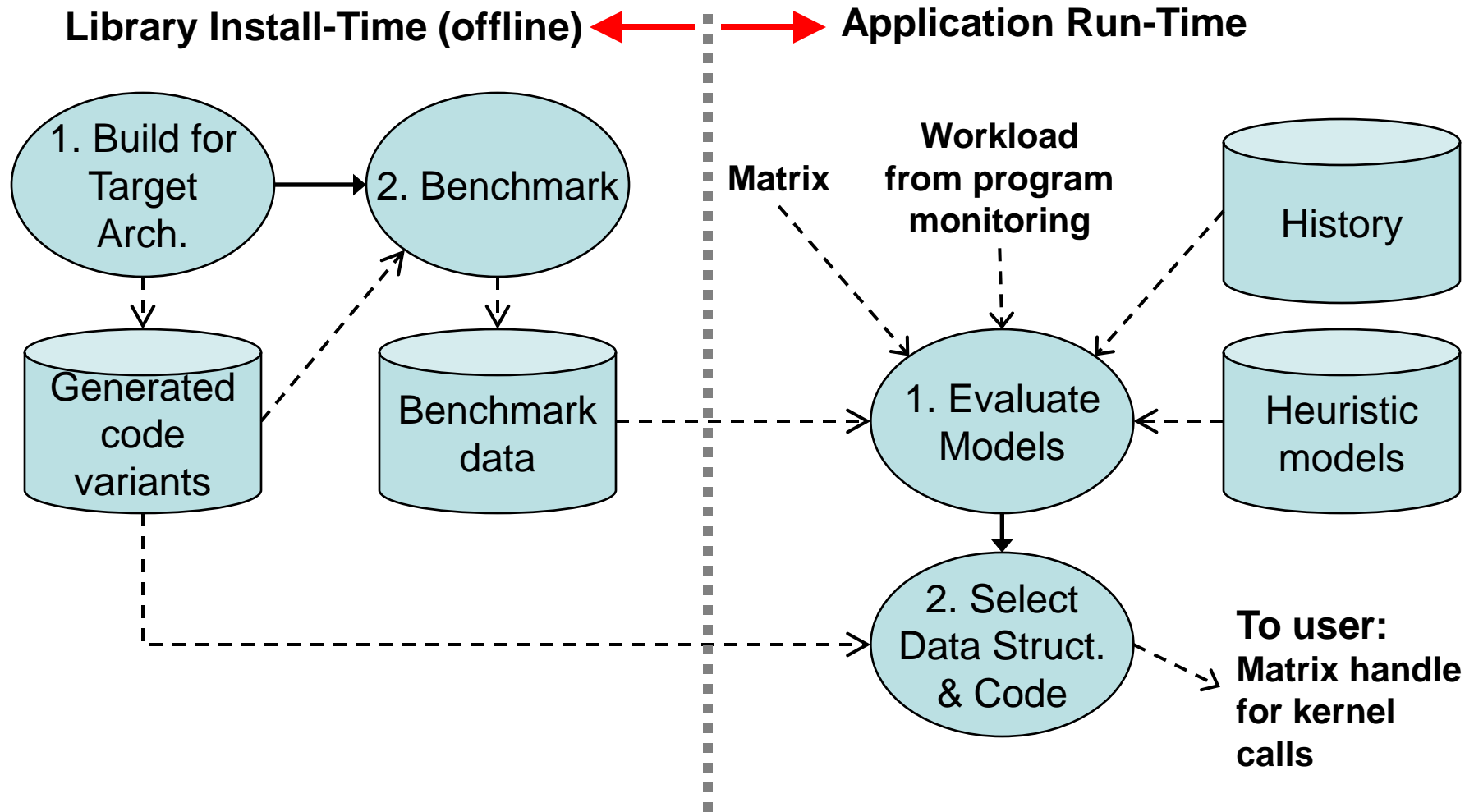
- ❖ Efficiency: Low-level Auto-tuning libraries, such as OSKI, enable better performance for scientific computations
 - Complex matrix tuning optimizations
 - C code enables near peak performance
 - Hard to write
- ❖ Productivity: Higher level languages, such as Python, enable faster/better code development
 - 2-5x faster development (P. Hudak and M. P. Jones, 1994)
 - Less efficiency
- ❖ Can we combine the benefits of both?

Mflops/s for Various Block Sizes in MatMul Operation



- ❖ C Library used in solver libraries
- ❖ BLAS-style interface
 - SpMV, SpTS, etc.
- ❖ Automatically tuned computational kernels on sparse matrices
 - Optimal tuning choices are often non-obvious
- ❖ 3 Types of Tuning
 - Install-time tuning (based on system)
 - Implicit run-time tuning (performance monitoring)
 - Explicit run-time tuning (workload hints)

How OSKI Tunes (Overview)



```
oski_matrix_t A_tunable = oski_CreateMatCSR( ... );
```

```
/* Tell OSKI we will call SpMV 500 times (explicit workload hint) */  
oski_SetHintMatMult(A_tunable, OP_NORMAL,  $\alpha$ , x_view,  $\beta$ , y_view, 500);  
  
/* Tell OSKI we think the matrix has 8x8 blocks (structural hint) */  
oski_SetHint(A_tunable, HINT_SINGLE_BLOCKSIZE, 8, 8);  
  
/* Ask OSKI to tune */  
oski_TuneMat(A_tunable);
```

```
for( i = 0; i < 500; i++ )  
    oski_MatMult(A_tunable, OP_NORMAL,  $\alpha$ , x_view,  $\beta$ , y_view);
```

- ❖ Can we enable users to both write code productively and achieve speedups from auto-tuning?
- ❖ Currently: C/OSKI requires the user to mix tuning and computation code – Not productive
 - When to change representation of a matrix?
 - When to do expensive "unmarshal" of a representation?
 - When to tune and re-tune?
 - Setting explicit tuning hints

- ❖ Provide Python bindings for OSKI via `scipy.sparse`
 - A python sparse matrix package with some overlap with OSKI
 - OSKI maintains data structures plus "shadow" data structures for tuning
 - Abstract datatypes wrap pointers to these structures
- ❖ Expose higher-level abstract datatypes & methods to productivity programmer
 - low-level OSKI objects become transparent to mainline computation
- ❖ Idea: separate tuning hints from main source code
 - changes to policy don't contaminate source
 - *policy experimentation can proceed in parallel*
 - Enables performance portability

USER PROGRAM: PYTHON

```
import scipy.sparse

A = csr_matrix()
b = array()

C = A*b
```

SCIPY SOURCE CODE

```
@check_OSKI
def _mul_(*args)

    perform matmul
```

DECORATOR CODE

```
def check_OSKI(*args)
    if OSKI is installed:
        if check_for_hint():
            set_hints()
            tune_mats()

            call OSKI SpMV
            gather profiling data

    else:
        fall through to
        scipy matmul code
```

- ❖ Need to know when and where to associate tuning hints
- ❖ Questions
 - How much (if any) information should the user specify?
 - How can we keep track of this information?

Matrix A1, A2
Vector v

GMRES(A1,v)

GMRES(A2,v)

Change nonzero entries of A1

GMRES(A1,v)

**How does PySKI
know which tuned
SPMV and TSQR to
use? What if co-
tuning is required?**

```
@tune_function
def GMRES(Matrix A, Vector v):
```

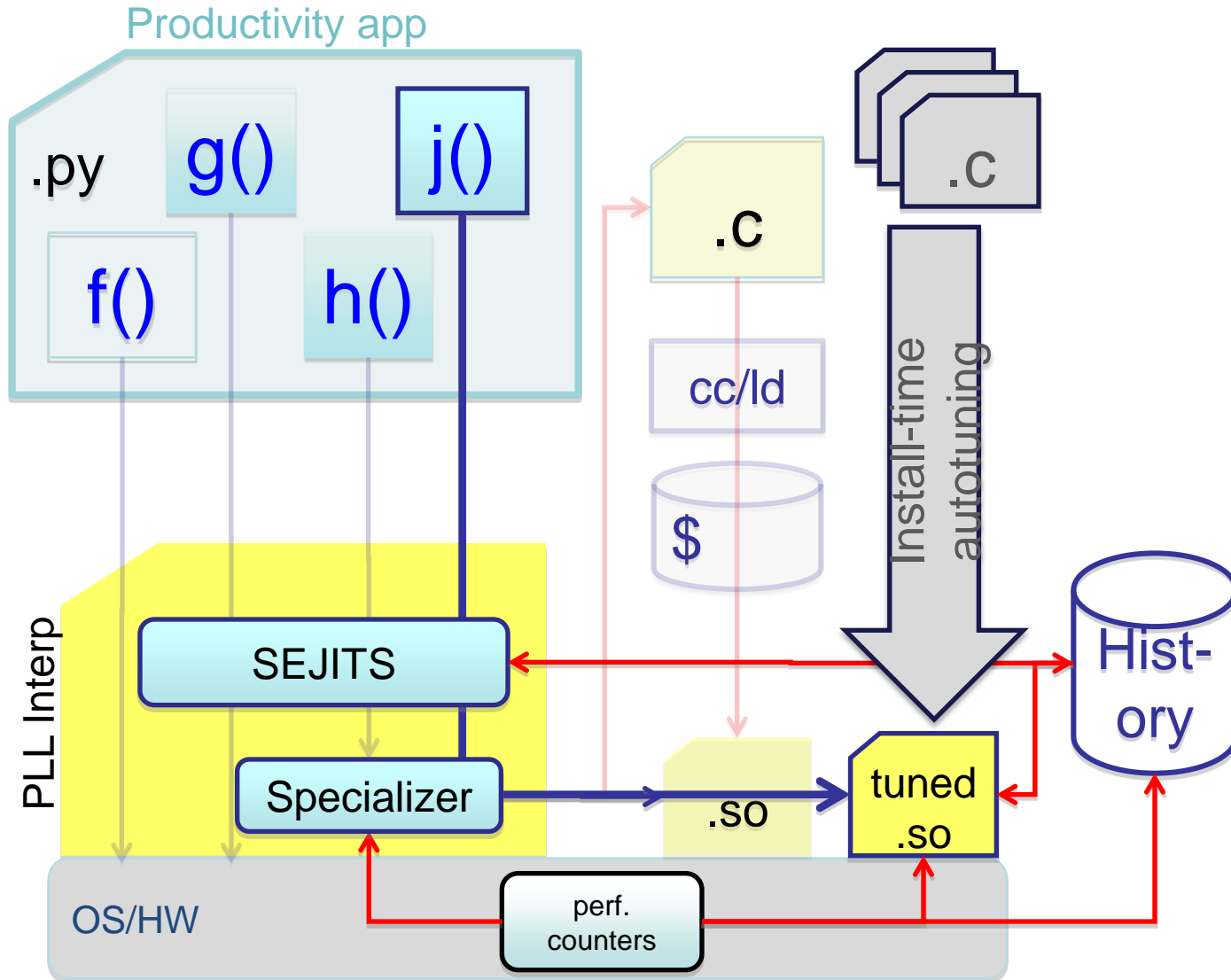
```
    SPMV(A,v)
    TSQR(v)
```

- ❖ History, or profiling data, can be useful in future tuning operations

- ❖ How much history should we keep?
 - From this execution?
 - Currently in OSKI, along with load/save transformation methods
 - Across multiple runs?

- ❖ Future: maintain tuning databases

The Big Picture



BACKUP SLIDES

Summary of Performance Optimizations

- ❖ Optimizations for SpMV
 - **Register blocking (RB)**: up to **4x** over CSR
 - **Variable block splitting**: **2.1x** over CSR, **1.8x** over RB
 - **Diagonals**: **2x** over CSR
 - **Reordering** to create dense structure + **splitting**: **2x** over CSR
 - **Symmetry**: **2.8x** over CSR, **2.6x** over RB
 - **Cache blocking**: **2.8x** over CSR
 - **Multiple vectors (SpMM)**: **7x** over CSR
 - And combinations...
- ❖ Sparse triangular solve
 - Hybrid sparse/dense data structure: **1.8x** over CSR
- ❖ Higher-level kernels
 - **$A \cdot A^T \cdot x$, $A^T \cdot A \cdot x$** : **4x** over CSR, **1.8x** over RB
 - **$A^2 \cdot x$** : **2x** over CSR, **1.5x** over RB
 - **$[A \cdot x, A^2 \cdot x, A^3 \cdot x, \dots, A^k \cdot x]$**

- ❖ Preliminary results: **2x** speedup over Python for $\sim 1000 \times 1000$ matrices
 - Need to test larger sizes, where matrix does not fit in cache

- ❖ P. Hudak and M. P. Jones. Haskell vs. Ada vs. C++ vs. Awk vs...an experiment in software prototyping productivity. Technical Report YALEU/DCS/RR-1049, Yale University Department of Computer Science, New Haven, CT, 1994.
- ❖ 80 implementations of same set of requirements were attempted by 74 different programmers. task was to see if a given phone number spells anything interesting, given access to a dictionary of legal words. programmers self-reported their development time. PLL programmers (Perl, Tcl, Python, Rexx) took anywhere from 2x-5x quicker to develop than ELL programmers (C, C++, Java). roughly, the "number of LOC per hour" is stable across all languages, except that for C/C++ the ratio is superlinear (ie, a C/C++ program that is twice as many LOC takes more than twice as long to produce), yet scripting languages do more work per LOC.

