CSAIL

BERKELEY PAR LAB

Lithe: Enabling Efficient Composition of Parallel Libraries

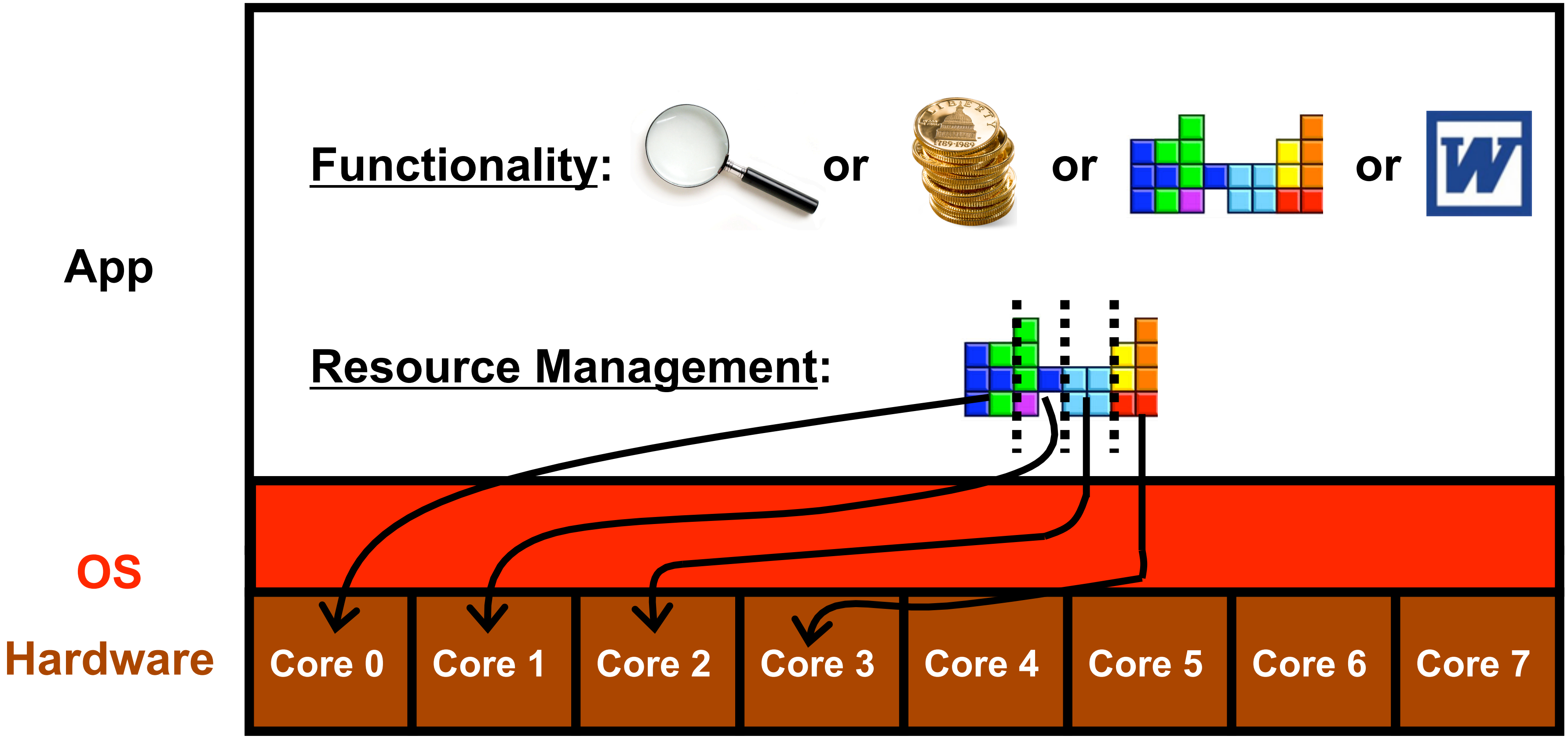
Heidi Pan, Benjamin Hindman, Krste Asanović

xoxo@mit.edu • {benh, krste}@eecs.berkeley.edu

Massachusetts Institute of Technology • UC Berkeley

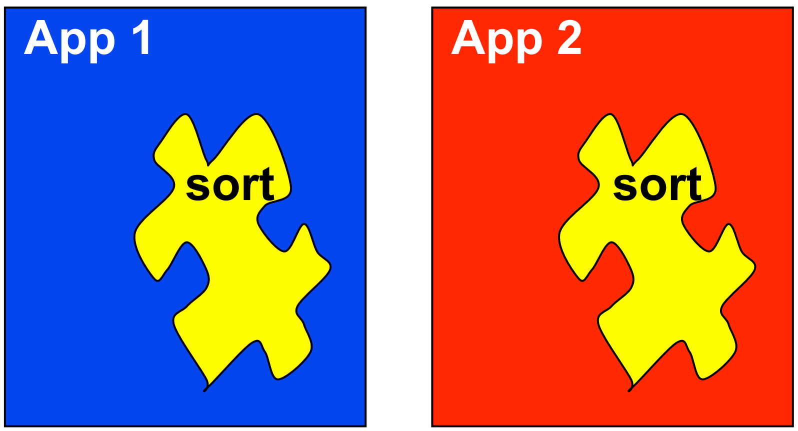
HotPar • Berkeley, CA • March 31, 2009

How to Build Parallel Apps?

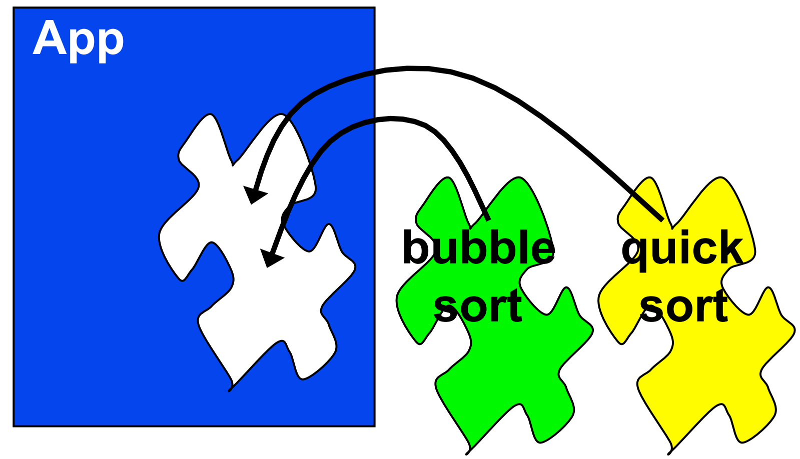


Need both programmer productivity and performance!

Composability is Key to Productivity



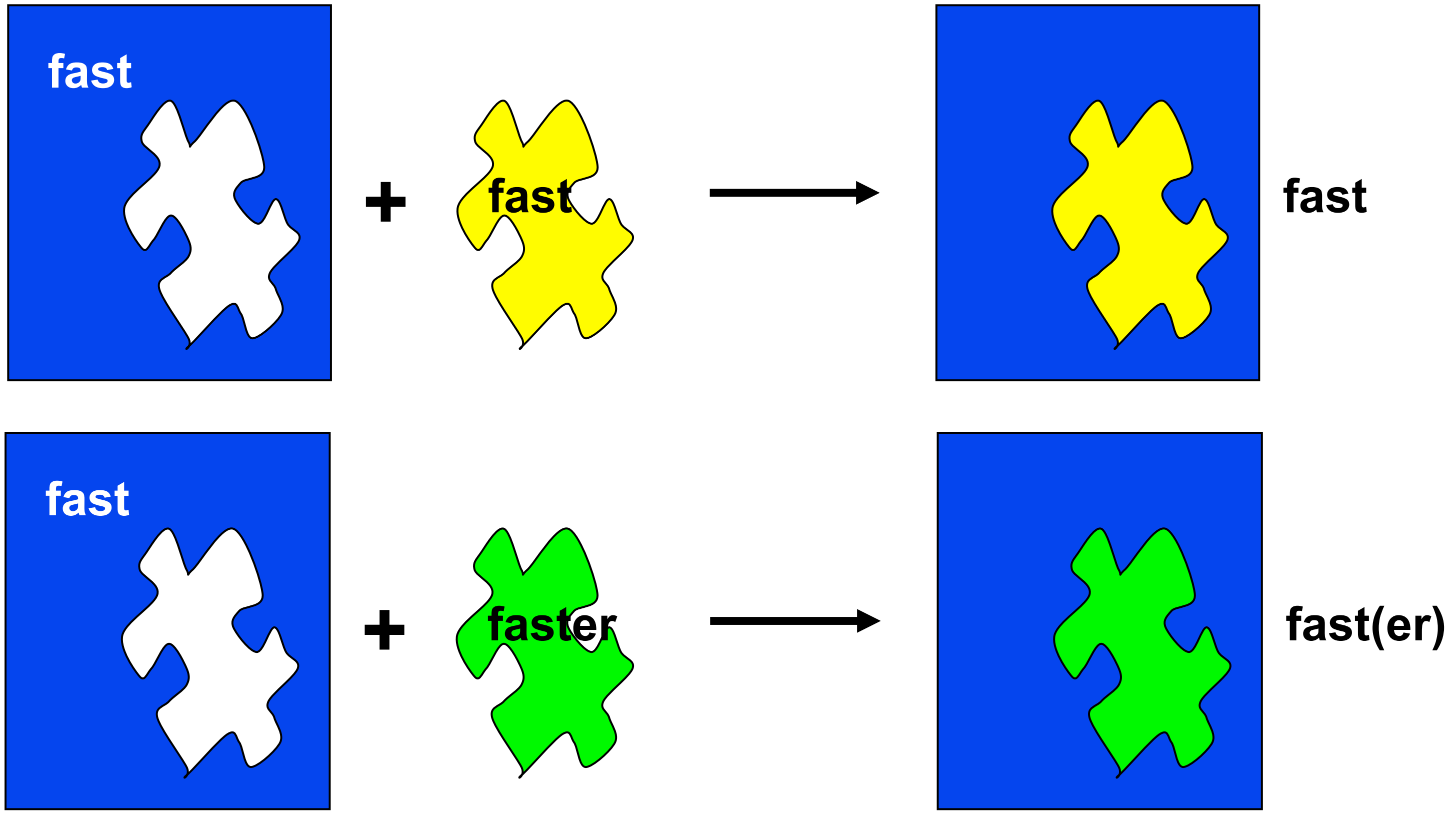
code reuse
same library implementation, different apps



modularity
same app, different library implementations

Functional Composability

Composability is Key to Productivity



Performance **Composability**

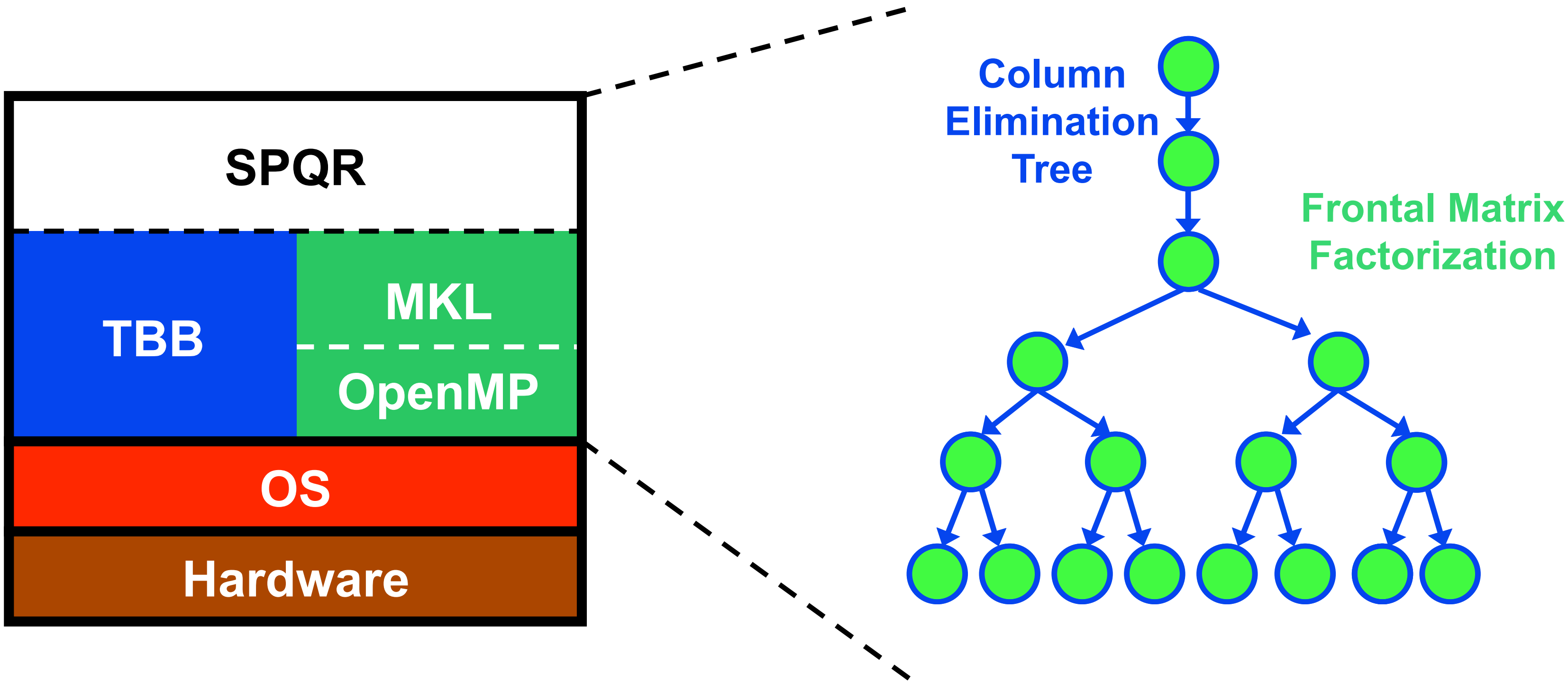
Talk Roadmap

- <u>**Problem**</u>**: *Efficient* parallel composability is hard!**
- <u>Solution</u>:
  - Harts
  - Lithe
- Evaluation

Motivational Example

Sparse QR Factorization

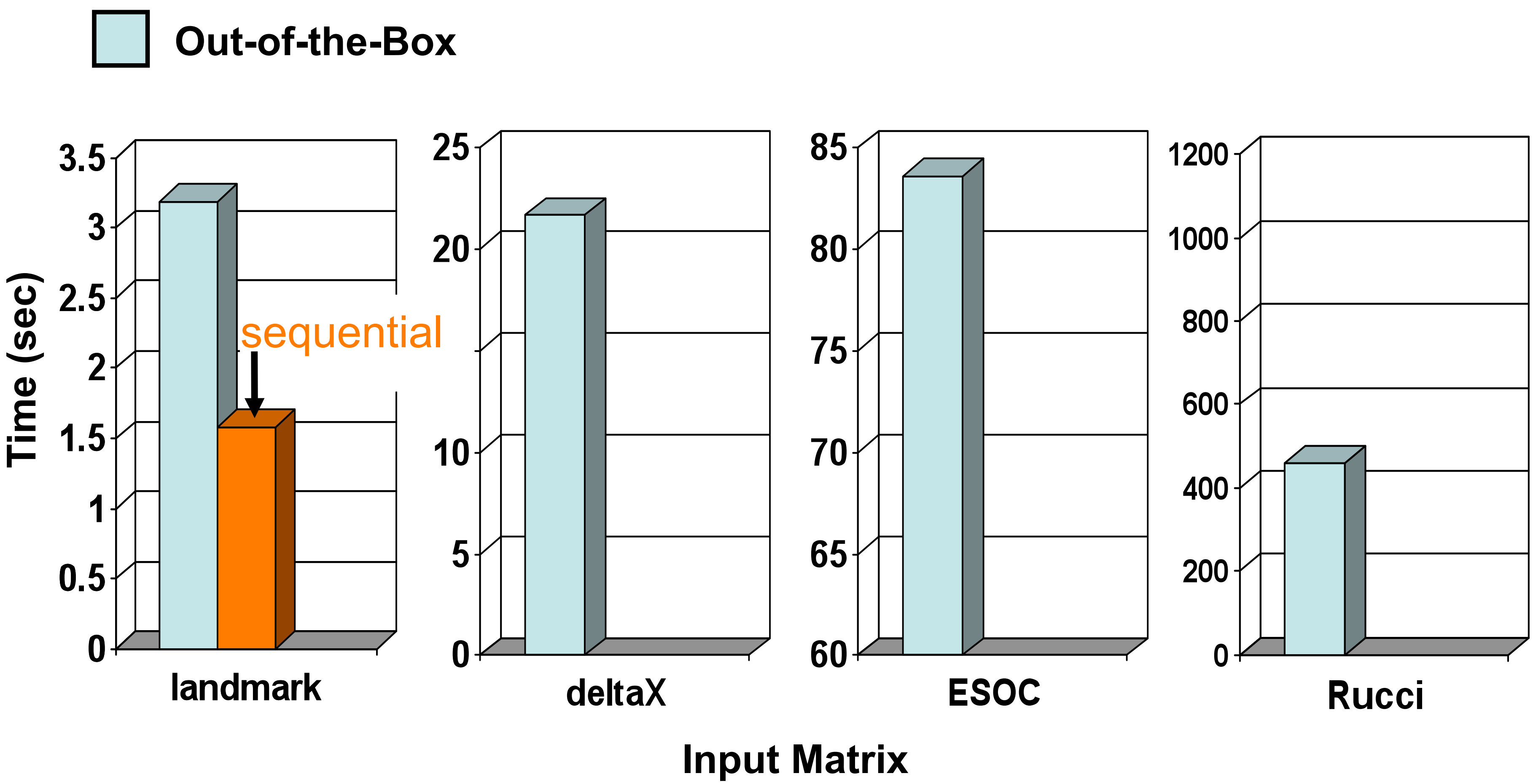
(Tim Davis, Univ of Florida)



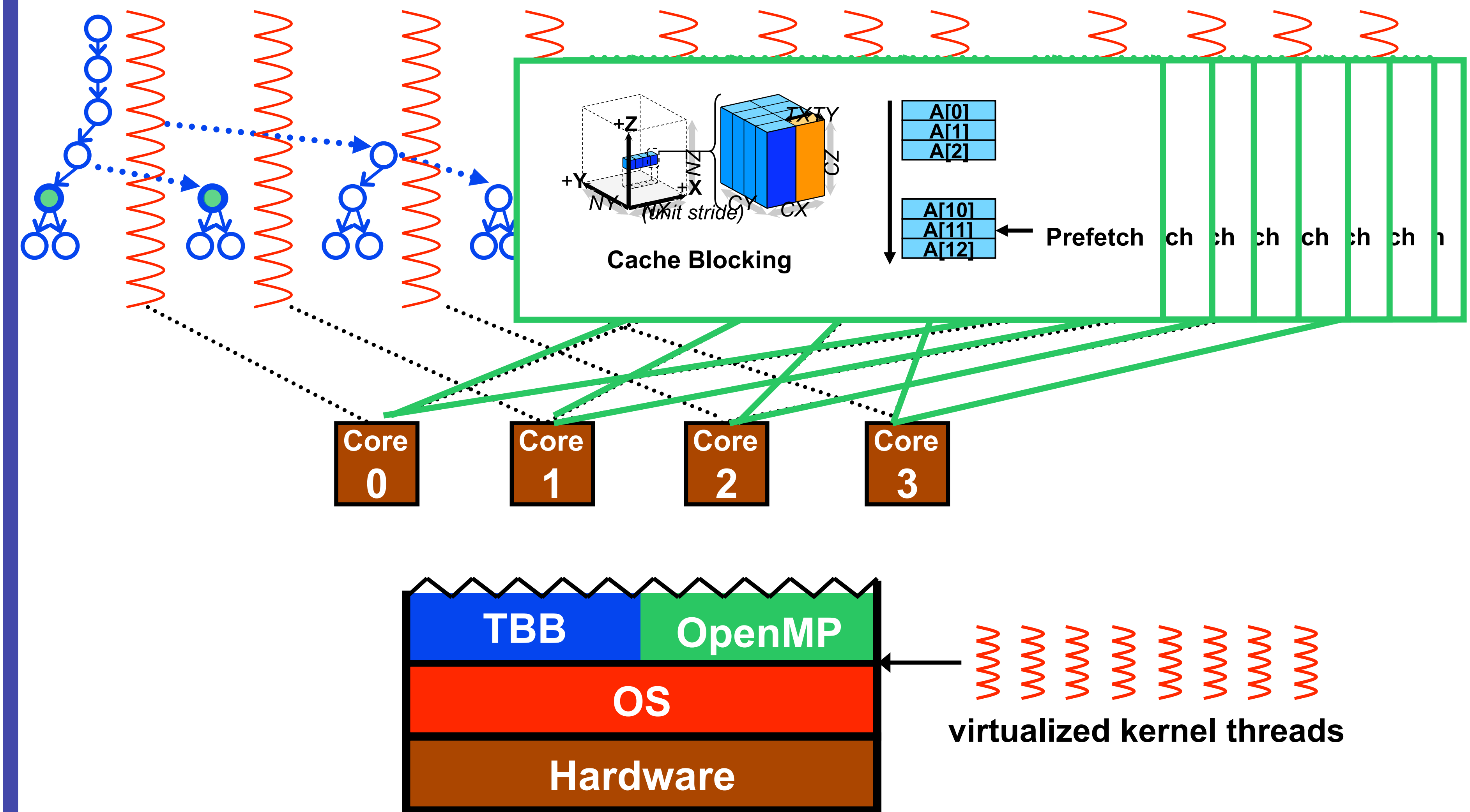
System Stack **Software Architecture**

Out-of-the-Box Performance

Performance of SPQR on 16-core Machine



Out-of-the-Box Libraries Oversubscribe the Resources



MKL Quick Fix

Using Intel MKL with Threaded Applications

http://www.intel.com/support/performancetools/libraries/mkl/sb/CS-017177.htm

Software Products

Intel® Math Kernel Library (Intel® MKL)
Using Intel® MKL with Threaded Applications

Page Contents:

- Memory Allocation MKL: Memory appears to be allocated and not released when calling some Intel MKL routines (e.g. sgetrf).
- Using Threading with BLAS and LAPACK
- Setting the Numbe
- Changing the Num
- Can I use Intel M

If the user threads the program using OpenMP directives and uses the Intel® Compilers to compile the program, Intel MKL and the user program will both use the same threading library. Intel MKL tries to determine if it is in a parallel region in the program, and if it is, it does not spread its operations over multiple threads. But Intel MKL can be aware that it is in a parallel region only if the threaded program and Intel MKL are using the same threading library. If the user program is threaded by some other means, Intel MKL may operate in multithreaded mode and the computations may be corrupted. Here are several cases and our recommendations:

- User threads the program using OS threads (pthreads on Linux*, Win32* threads on Windows*). If more than one thread calls Intel MKL and the function being called is threaded, it is important that threading in Intel MKL be turned off. Set OMP_NUM_THREADS=1 in the environment.
- User threads the program using OpenMP directives and/or pragmas and compiles the program using a compiler other than a compiler from Intel. This is more problematic because setting OMP_NUM_THREADS in the environment affects both the compiler's threading library and the threading

library with Intel MKL. In this case, the safe approach is to set OMP_NUM_THREADS=1.

- Multiple programs are running on a multiple-CPU system. In cluster applications, the parallel program can run separate instances of the program on each processor. However, the threading software will see multiple processors on the system even though each processor has a separate process running on it. In this case OMP_NUM_THREADS should be set to 1.
- If the variable OMP_NUM_THREADS environment variable is not set, then the default number of threads will be assumed 1.

Setting the Number of Threads for OpenMP* (OMP)

If more than one thread calls Intel MKL and the function being called is threaded, it is important that threading in Intel MKL be turned off.
Set OMP_NUM_THREADS=1 in the environment.

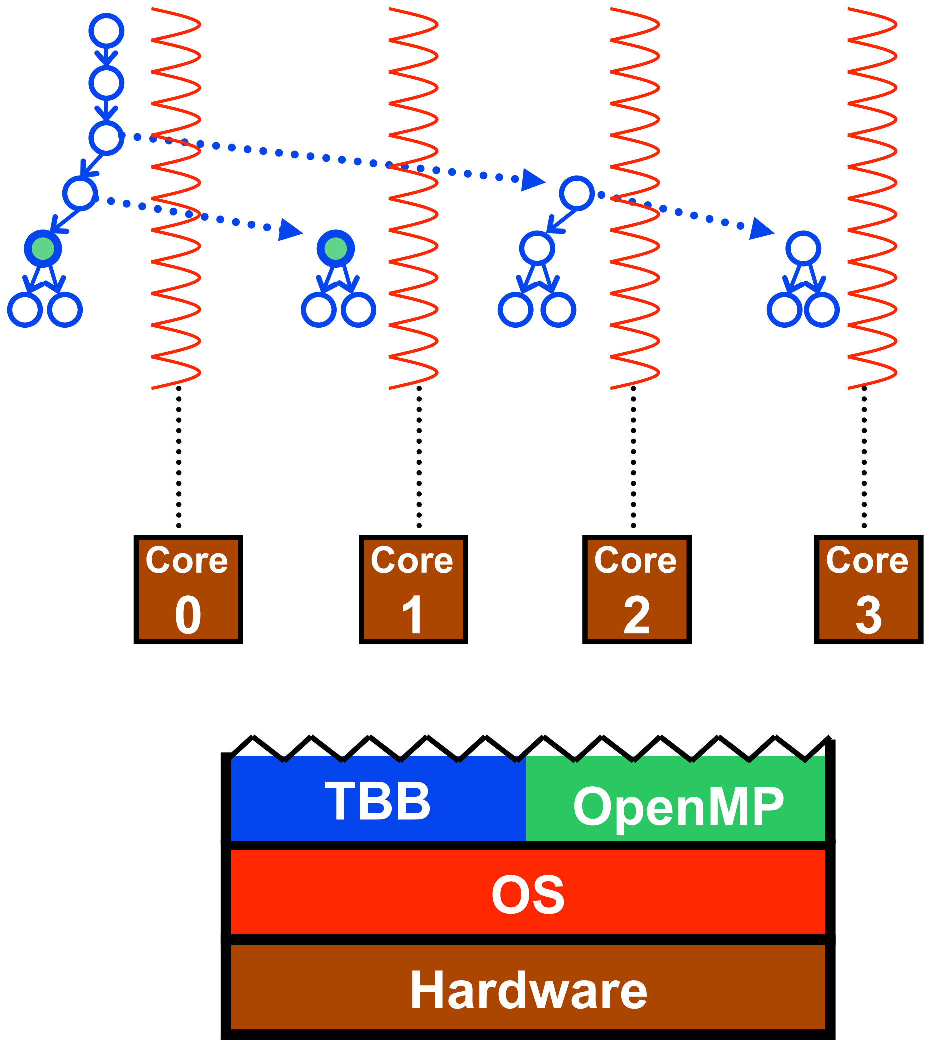
```
void main(int args, char *argv[]){

  double *a, *b, *c;
  a = new double [SIZE*SIZE];
  b = new double [SIZE*SIZE];
  c = new double [SIZE*SIZE];

  double alpha=1, beta=1;
  int m=SIZE, n=SIZE, k=SIZE, lda=SIZE, ldb=SIZE, ldc=SIZE, i=0, j=0;
  char transa='n', transb='n';

  for( i=0; i<SIZE; i++){
    for( j=0; j<SIZE; j++){
      a[i*SIZE+j]= (double)(i+j);
      b[i*SIZE+j]= (double)(i*j);
      c[i*SIZE+j]= (double)0;
    }
  }
  cblas_dgemm(CblasRowMajor, CblasNoTrans, CblasNoTrans,
        m, n, k, alpha, a, lda, b, ldb, beta, c, ldc);
```

```
printf("row\ta\tc\n");
for ( i=0;i<10;i++){
   printf("%d:\t%f\t%f\n", i, a[i*SIZE], c[i*SIZE]);
}

omp_set_num_threads(1);

for( i=0; i<SIZE; i++){
   for( j=0; j<SIZE; j++){
      a[i*SIZE+j]= (double)(i+j);
      b[i*SIZE+j]= (double)(i*j);
      c[i*SIZE+j]= (double)0;
   }
```

```
  printf("row\ta\tc\n");
  for ( i=0;i<10;i++){
     printf("%d:\t%f\t%f\n", i, a[i*SIZE],
c[i*SIZE]);
  }

  delete [] a;
  delete [] b;
  delete [] c;
}
```

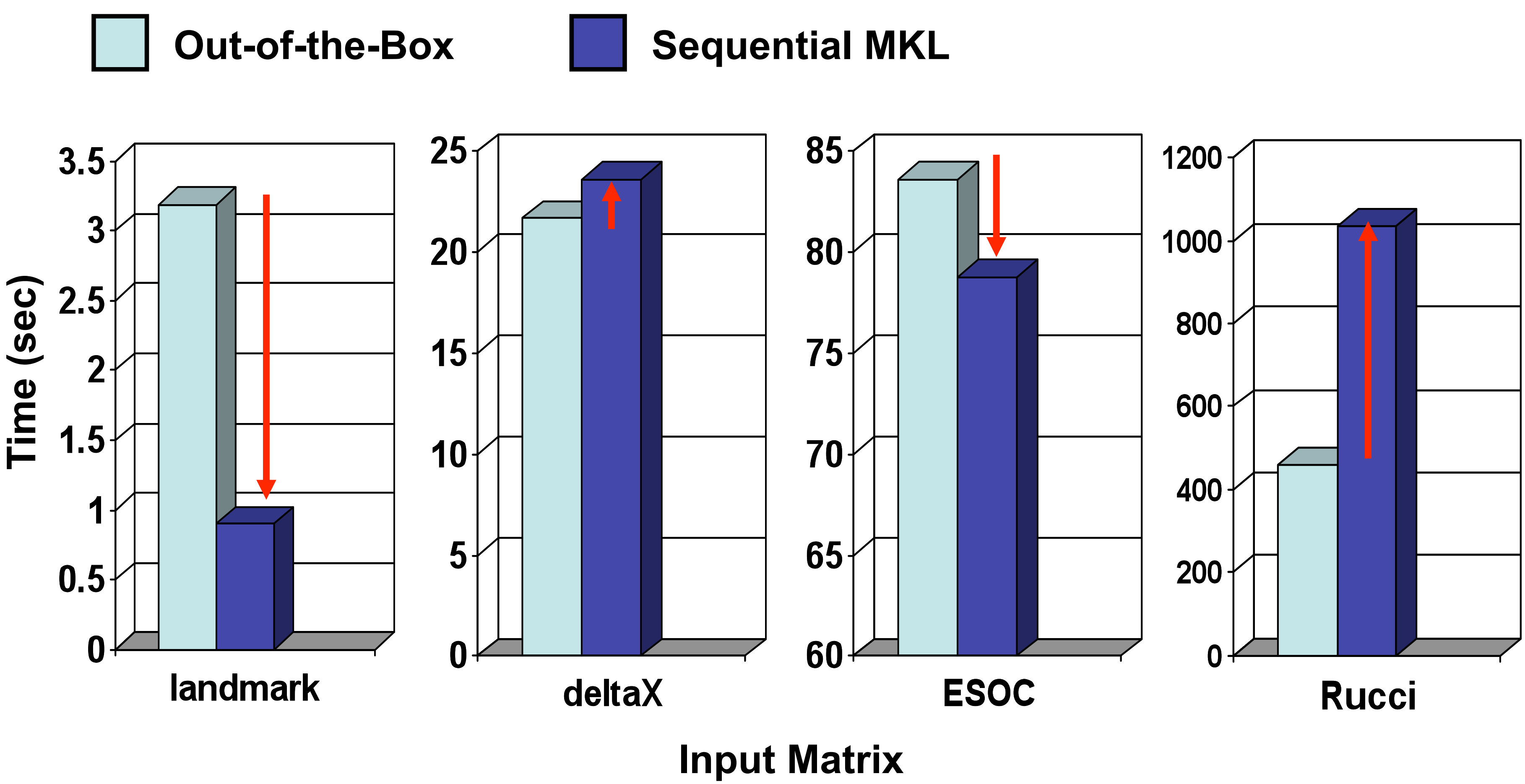
Can I use Intel MKL if I thread my application?
The Intel Math Kernel Library is designed and compiled for thread safety so it can be called from programs that are threaded. Calling Intel MKL routines that are threaded from multiple application threads can lead to conflict (including incorrect answers or program failures), if the calling library differs from the Intel MKL threading library.

Sequential MKL in SPQR



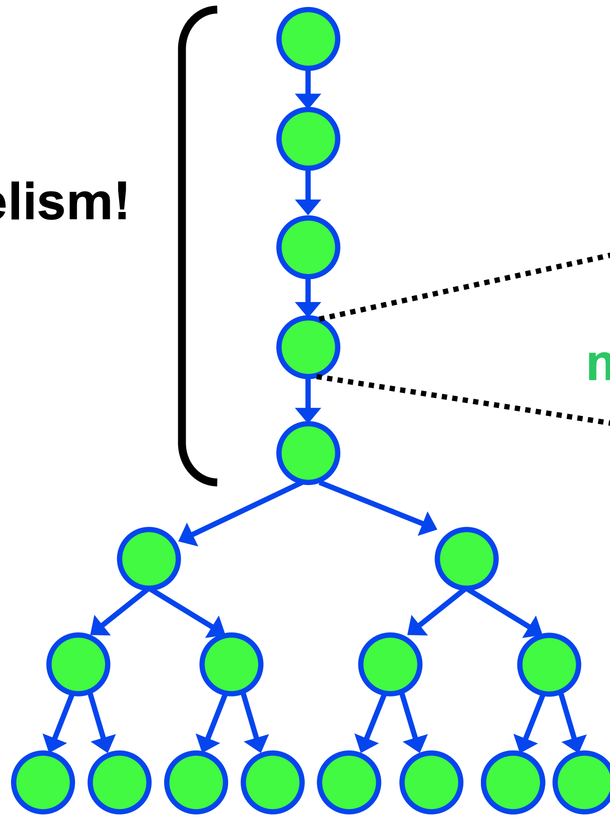
Sequential MKL Performance

Performance of SPQR on 16-core Machine



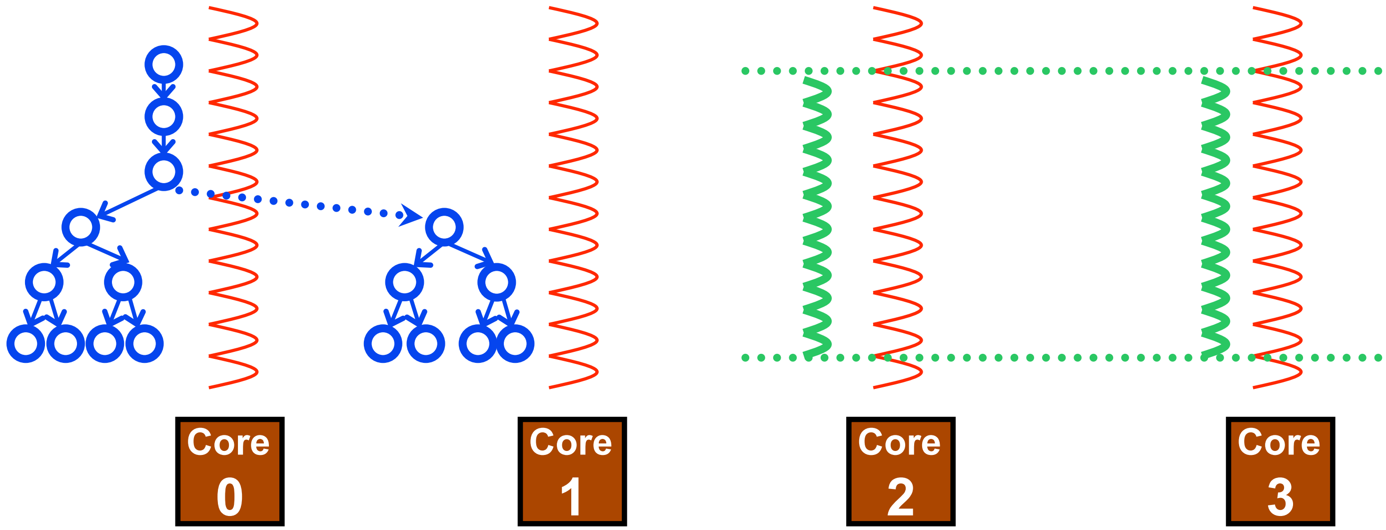
SPQR Wants to Use Parallel MKL

No task-level parallelism!

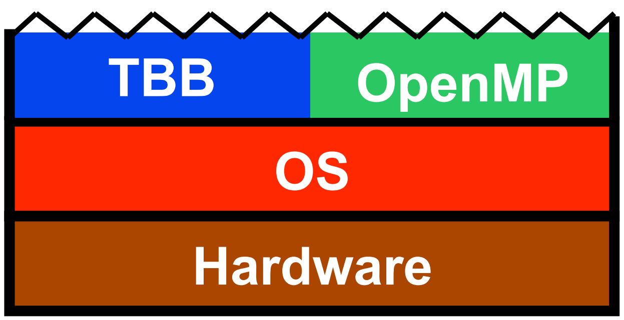


Want to exploit matrix-level parallelism.

Share Resources Cooperatively



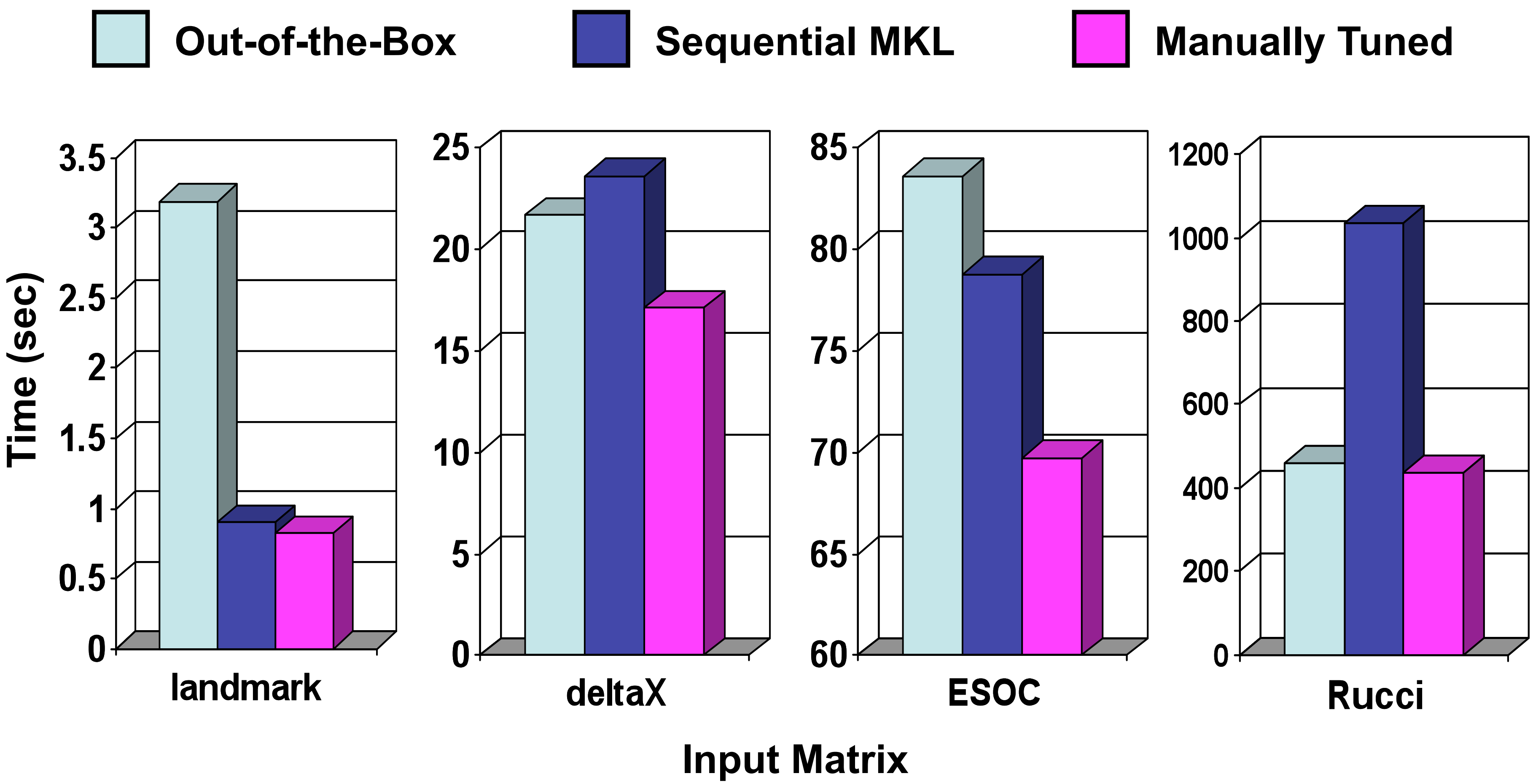
TBB_NUM_THREADS = 2 OMP_NUM_THREADS = 2



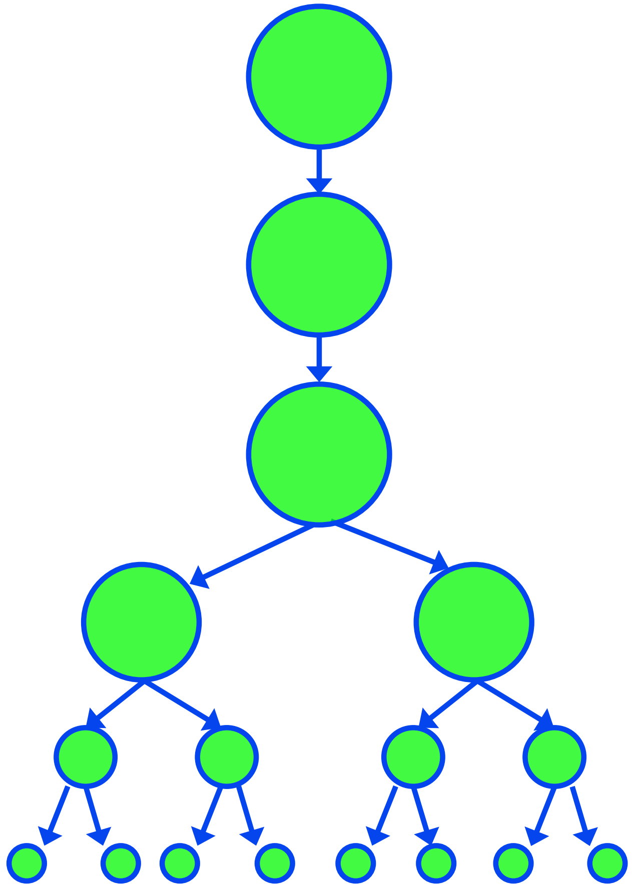
Tim Davis manually tunes libraries to effectively partition the resources.

Manually Tuned Performance

Performance of SPQR on 16-core Machine



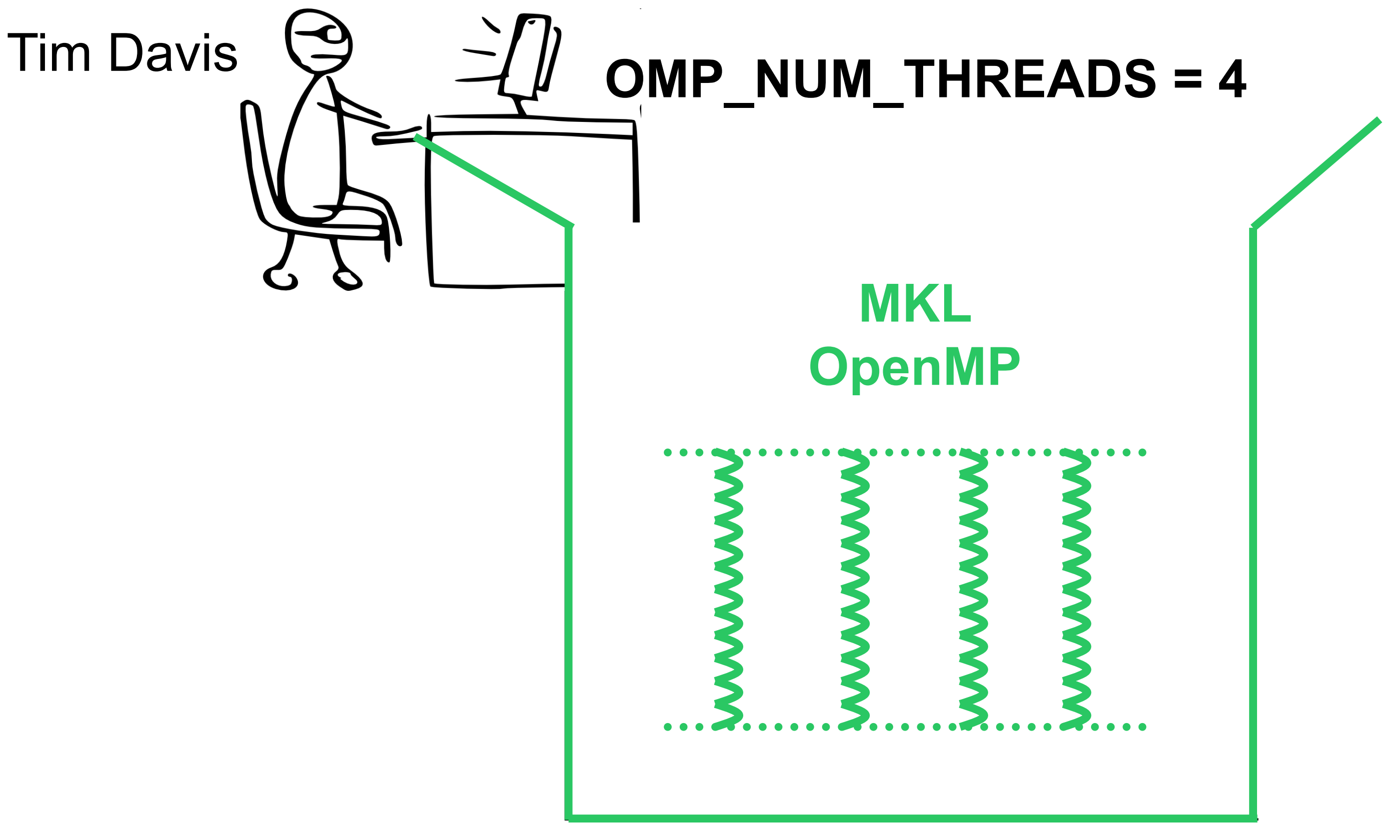
Manual Tuning Cannot Share Resources Effectively



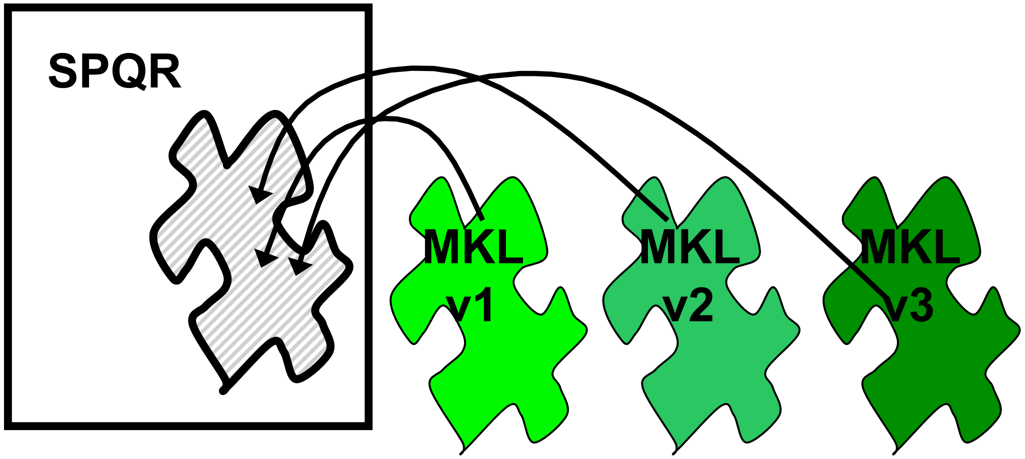
Give resources to OpenMP

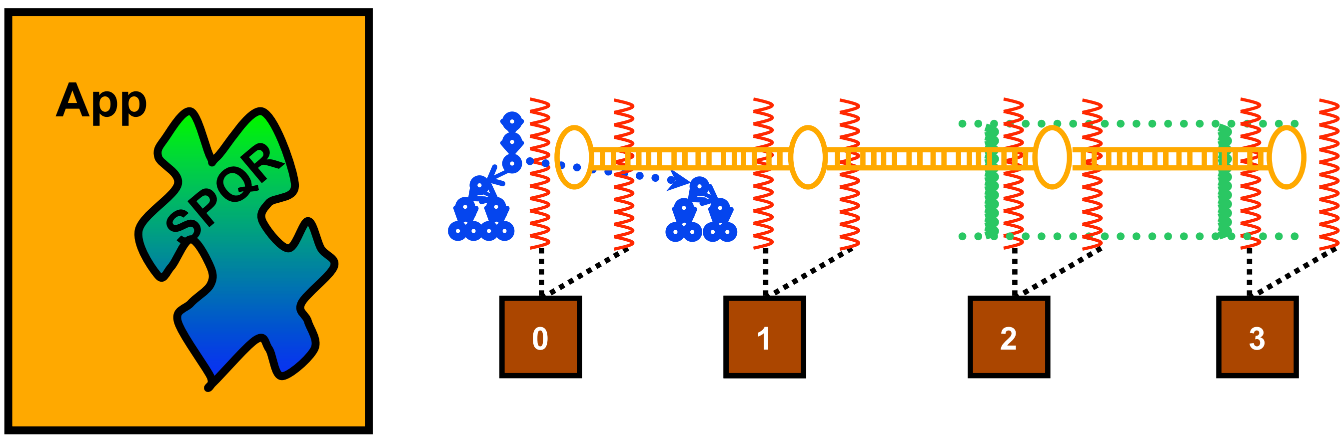
Give resources to TBB

Manual Tuning Destroys Functional Composability



Manual Tuning Destroys Performance Composability

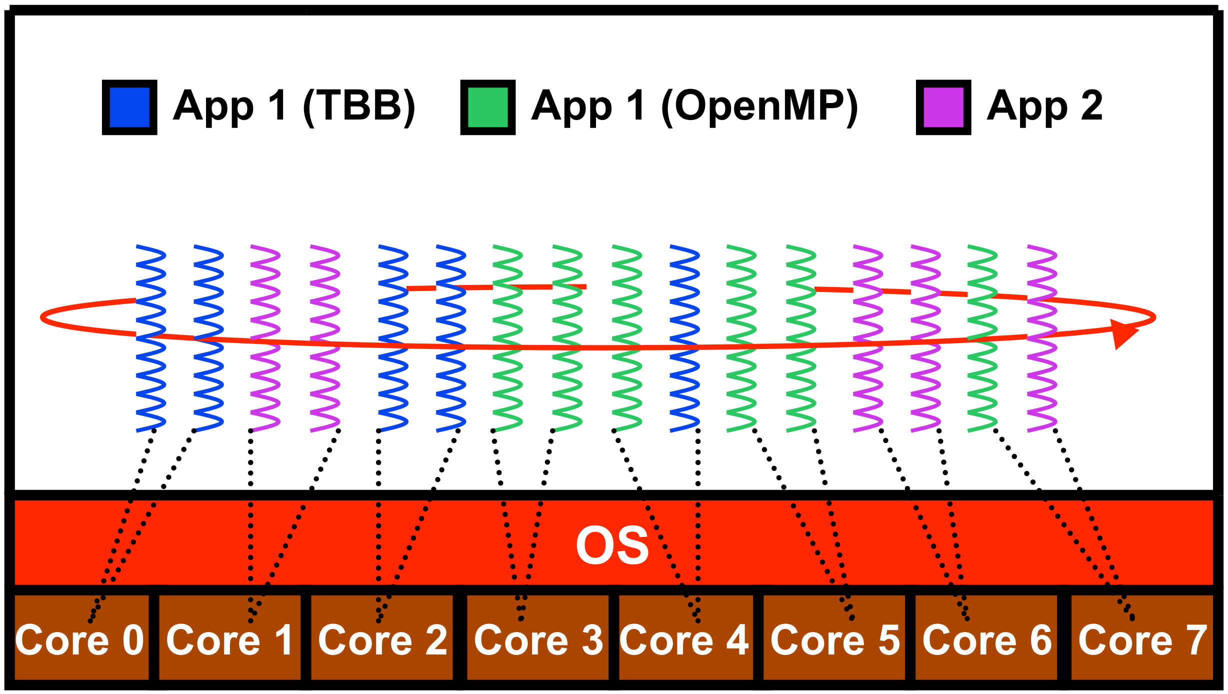




Talk Roadmap

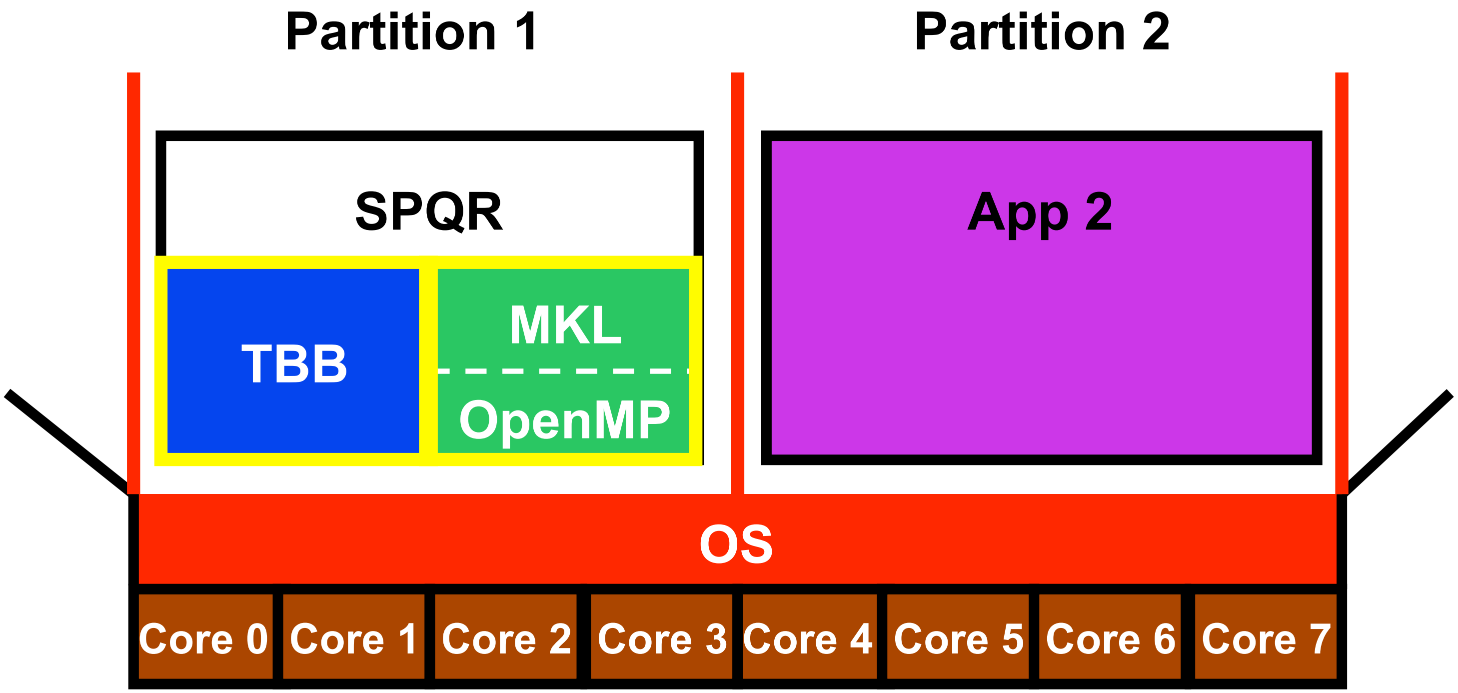
- Problem: *Efficient* parallel composability is hard!
- **Solution:**
  - **Harts: better resource abstraction**
  - **Lithe: framework for sharing resources**
- Evaluation

Virtualized Threads are Bad



Different codes compete unproductively for resources.

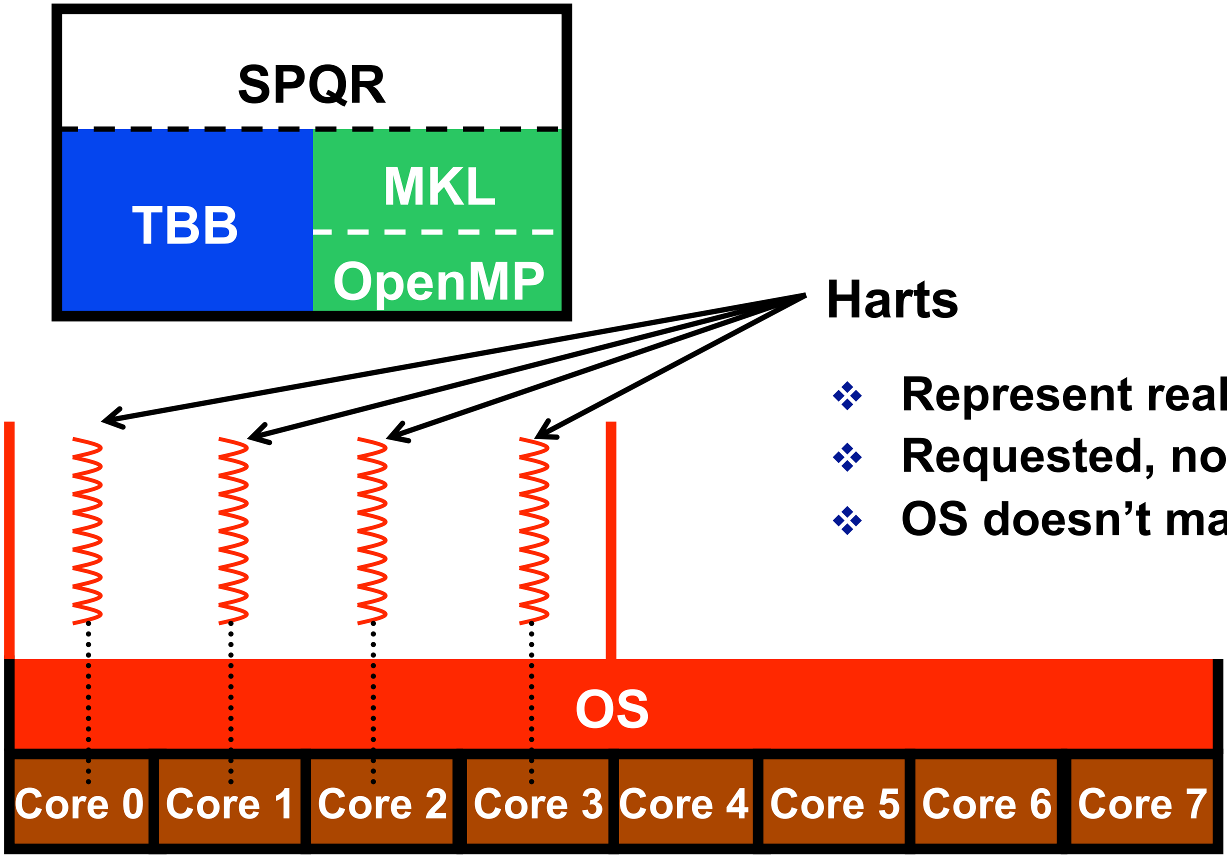
Space-Time Partitions aren't Enough



Space-time partitions isolate diff apps.

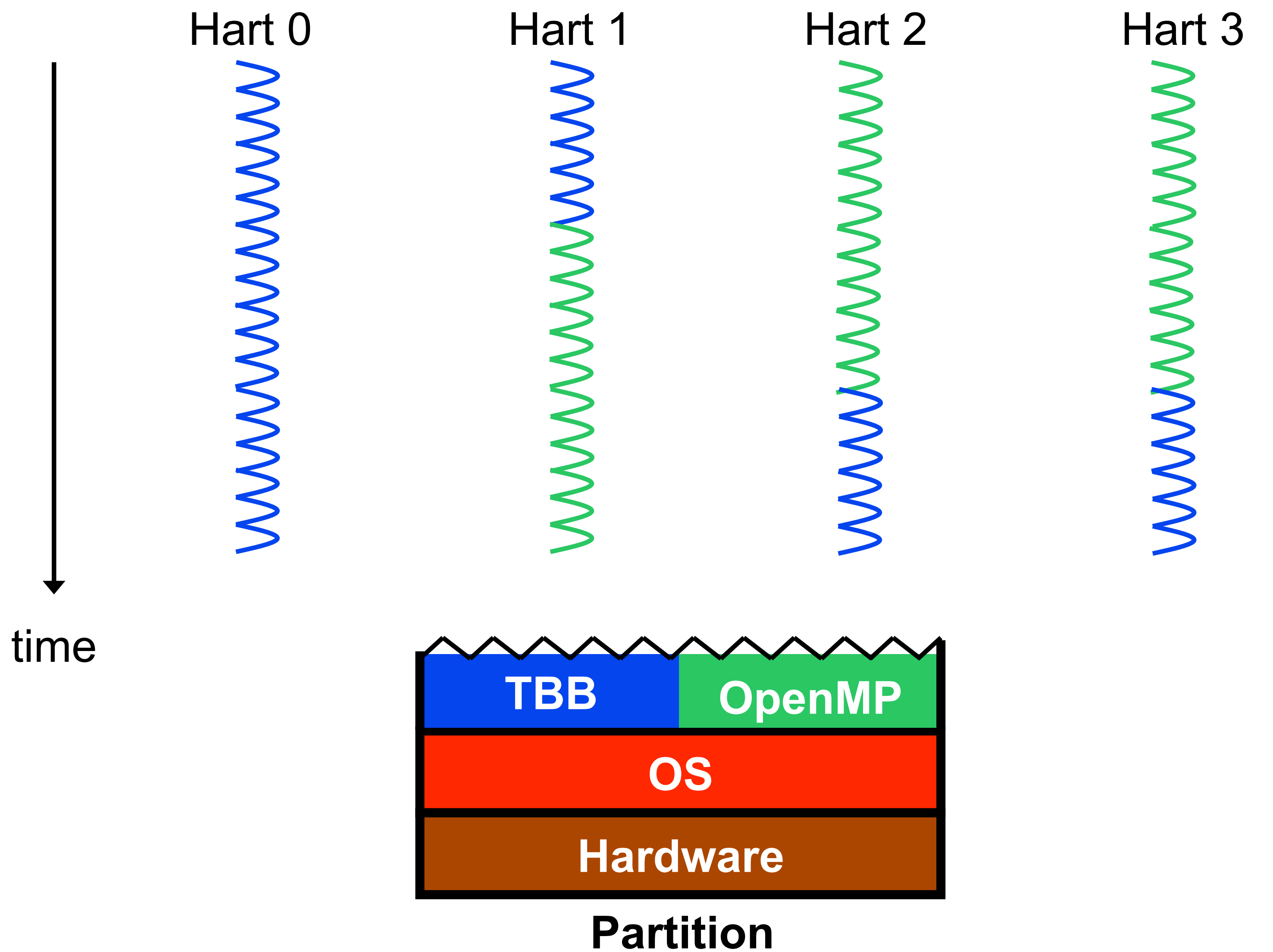
What to do within an app?

Harts: Hardware Thread Contexts

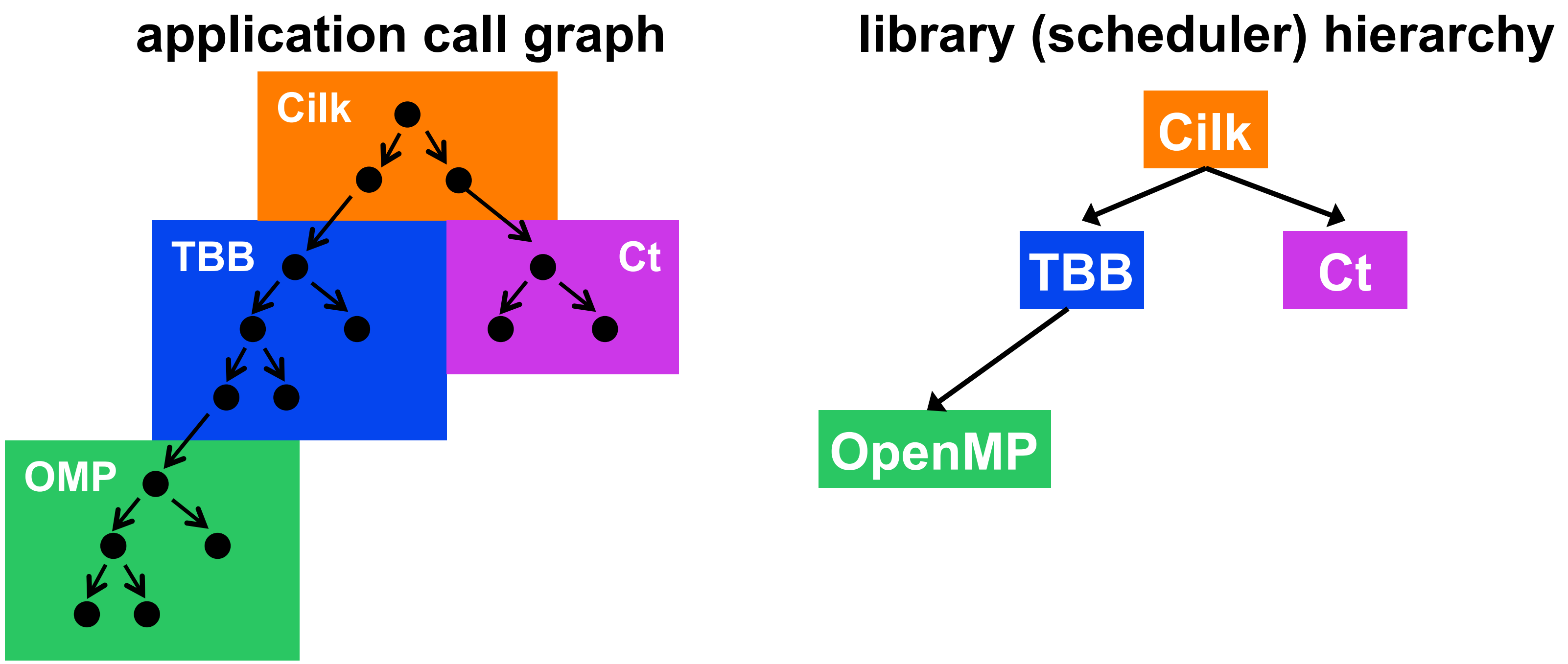


- Represent real hw resources.
- Requested, not created.
- OS doesn't manage harts for app.

Sharing Harts

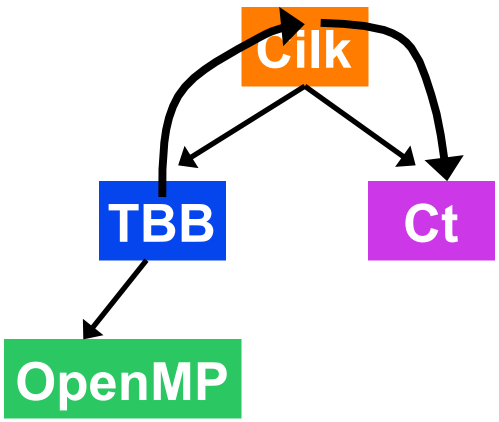


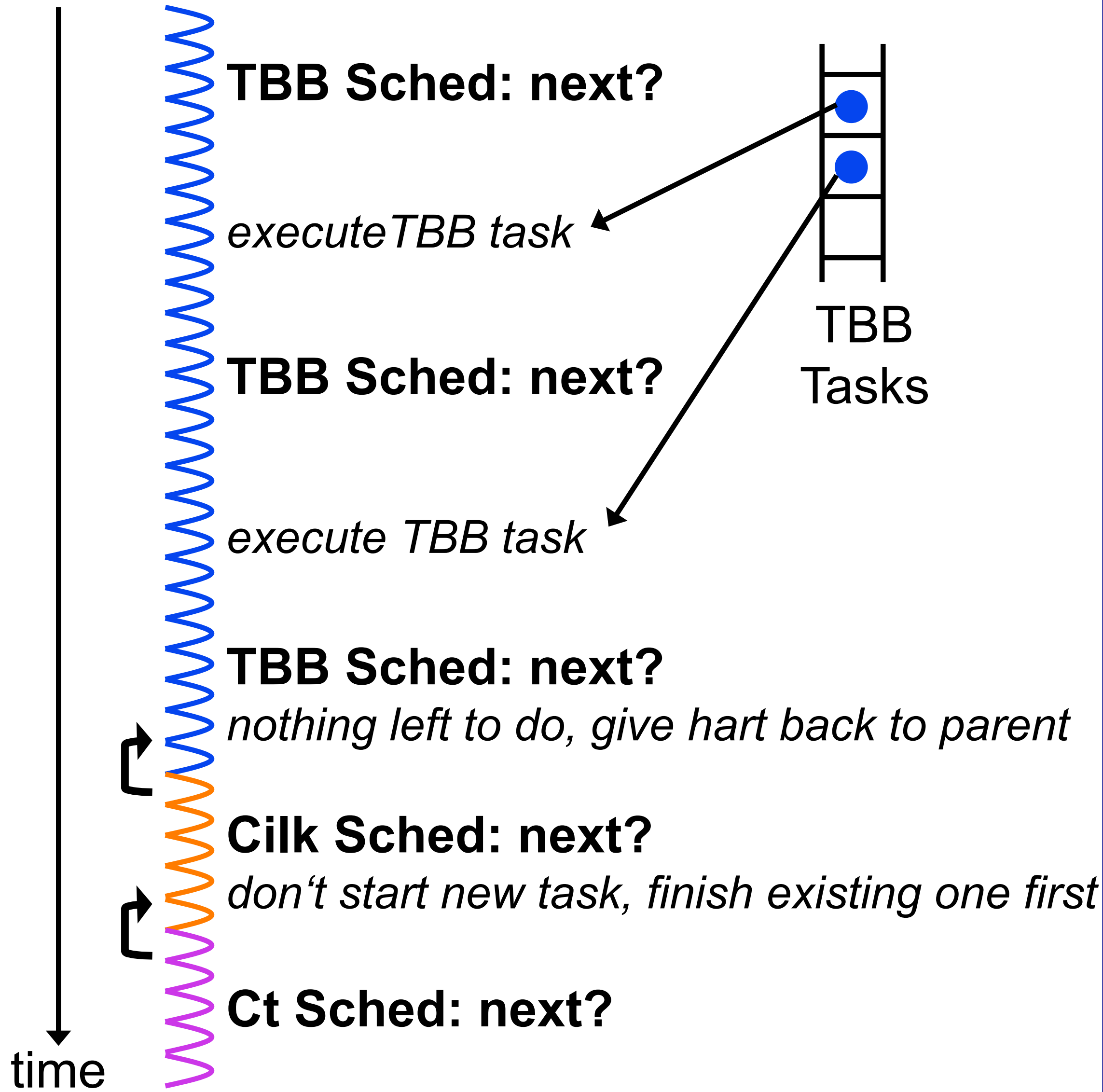
Cooperative Hierarchical Schedulers



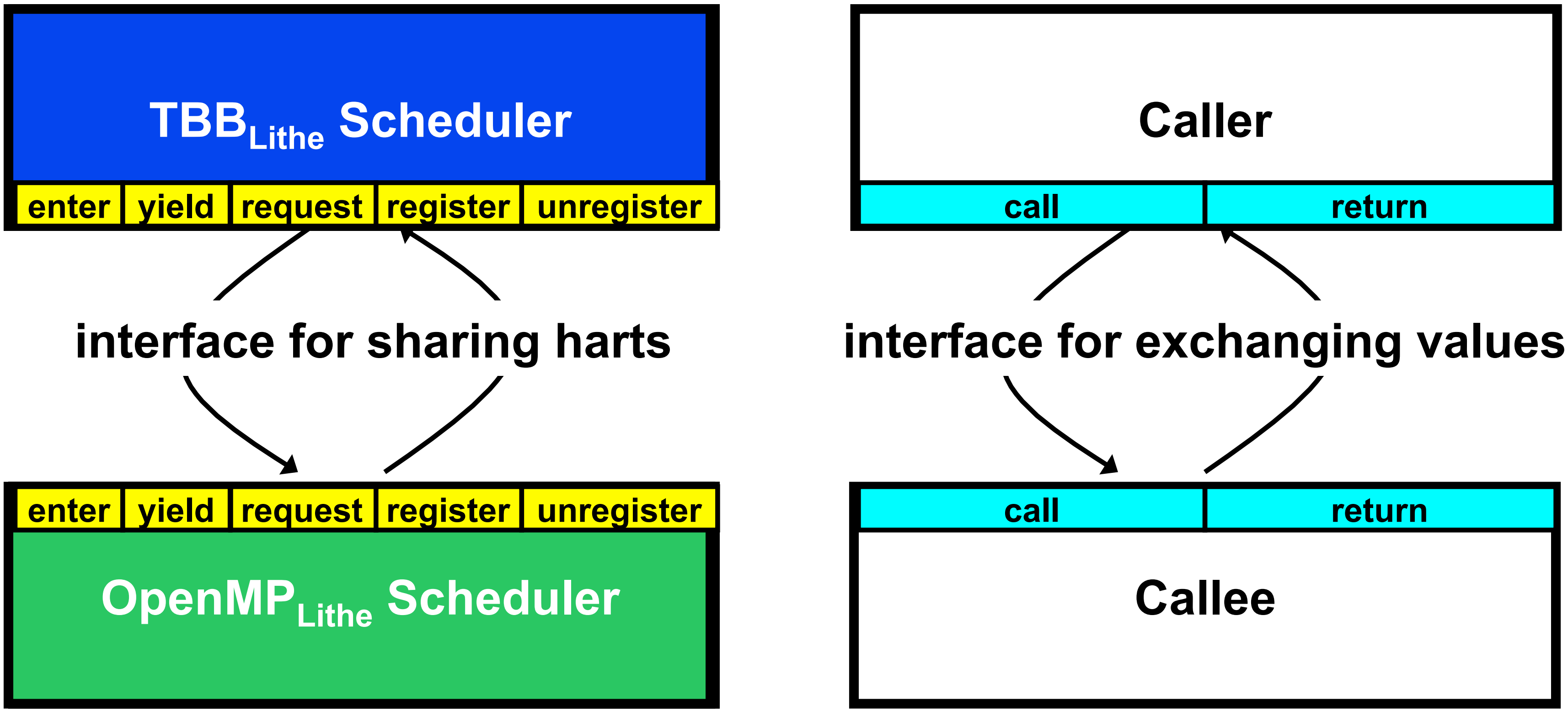
- **Modular:** Each piece of the app scheduled independently.
- **Hierarchical:** Caller gives resources to callee to execute on its behalf.
- **Cooperative:** Callee gives resources back to caller when done.

A Day in the Life of a Hart



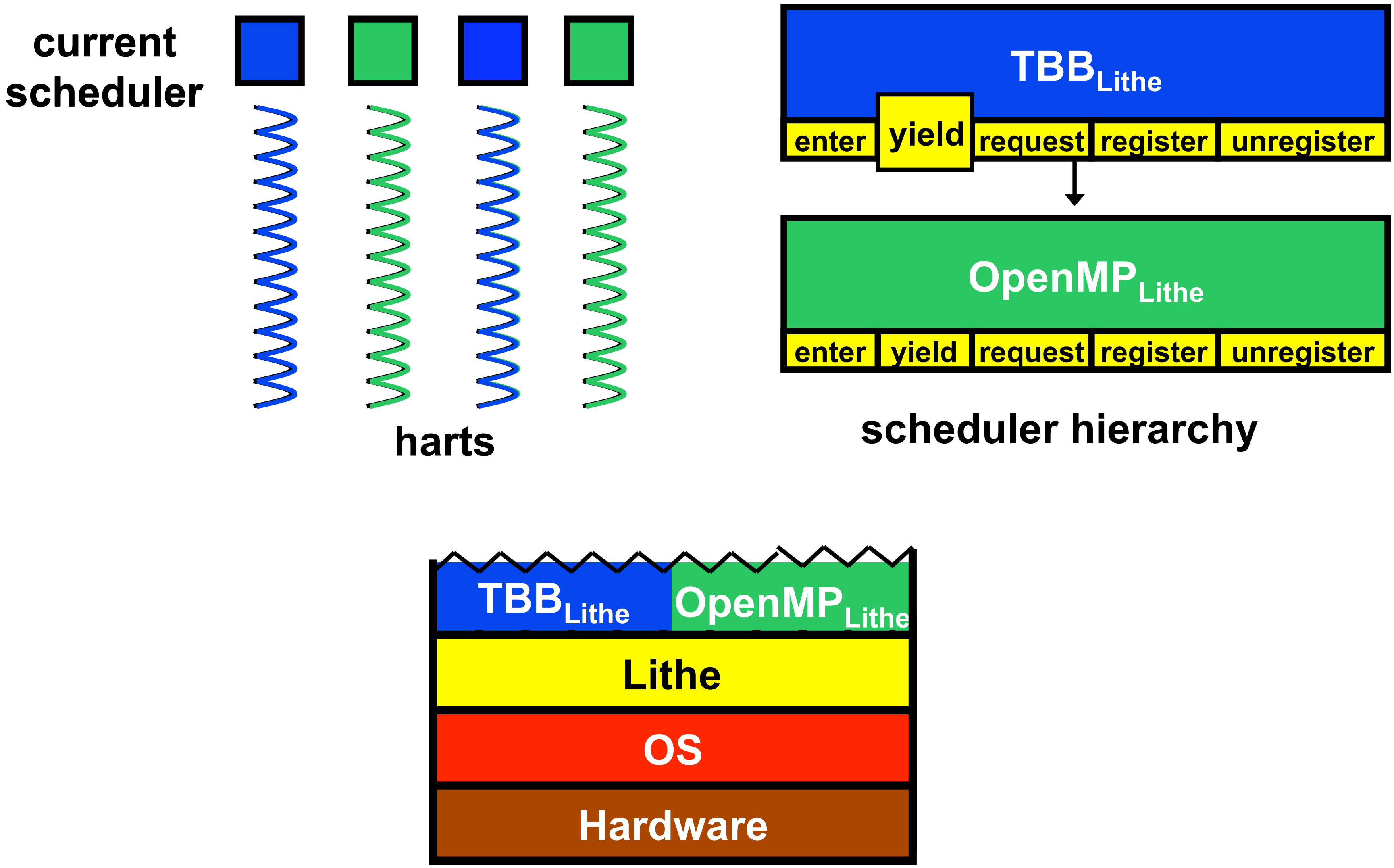


Standard Lithe ABI

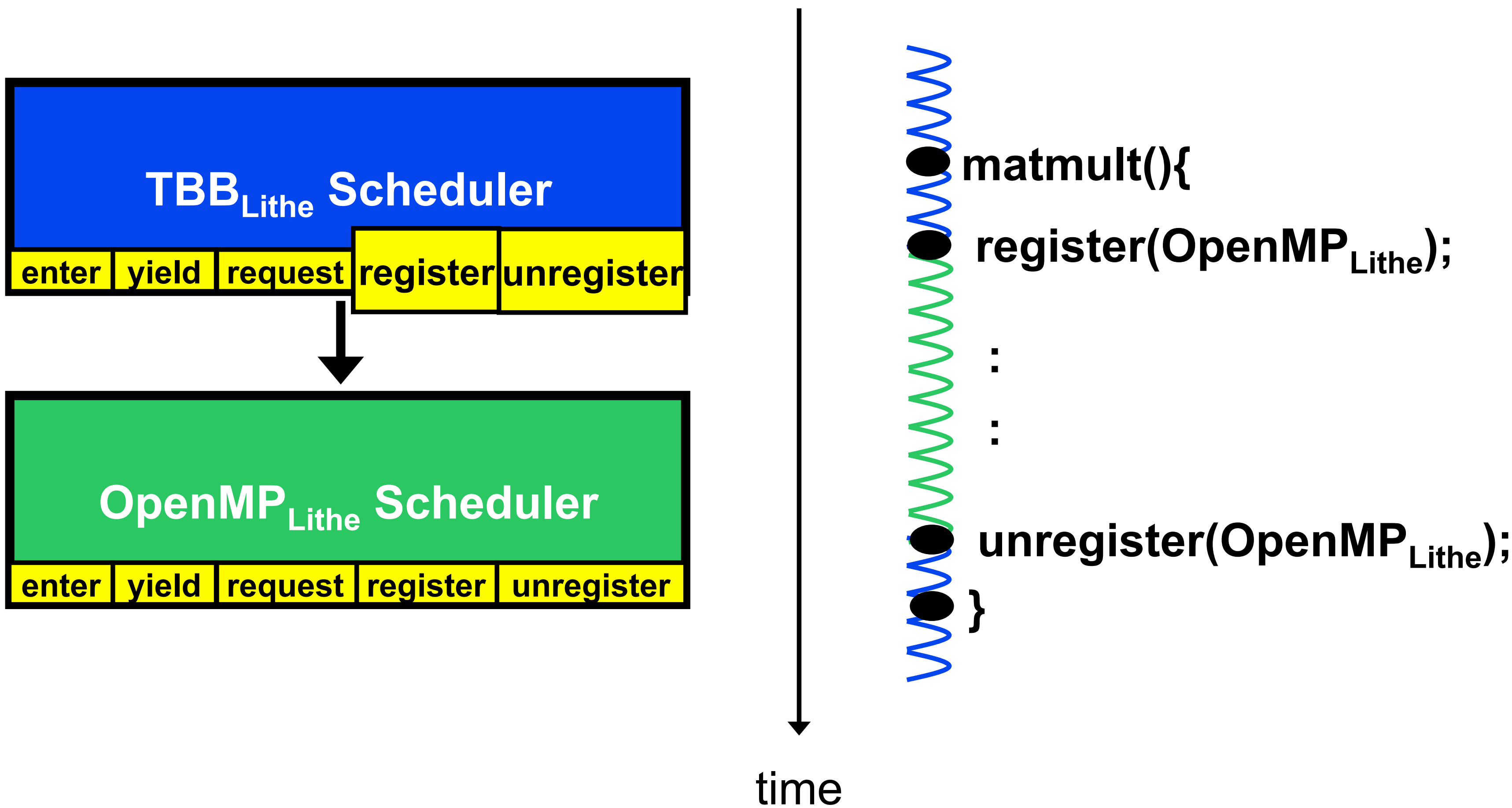


- **Analogous to function call ABI for enabling interoperable codes.**
- **Mechanism for sharing harts, *not* policy.**

Lithe Runtime

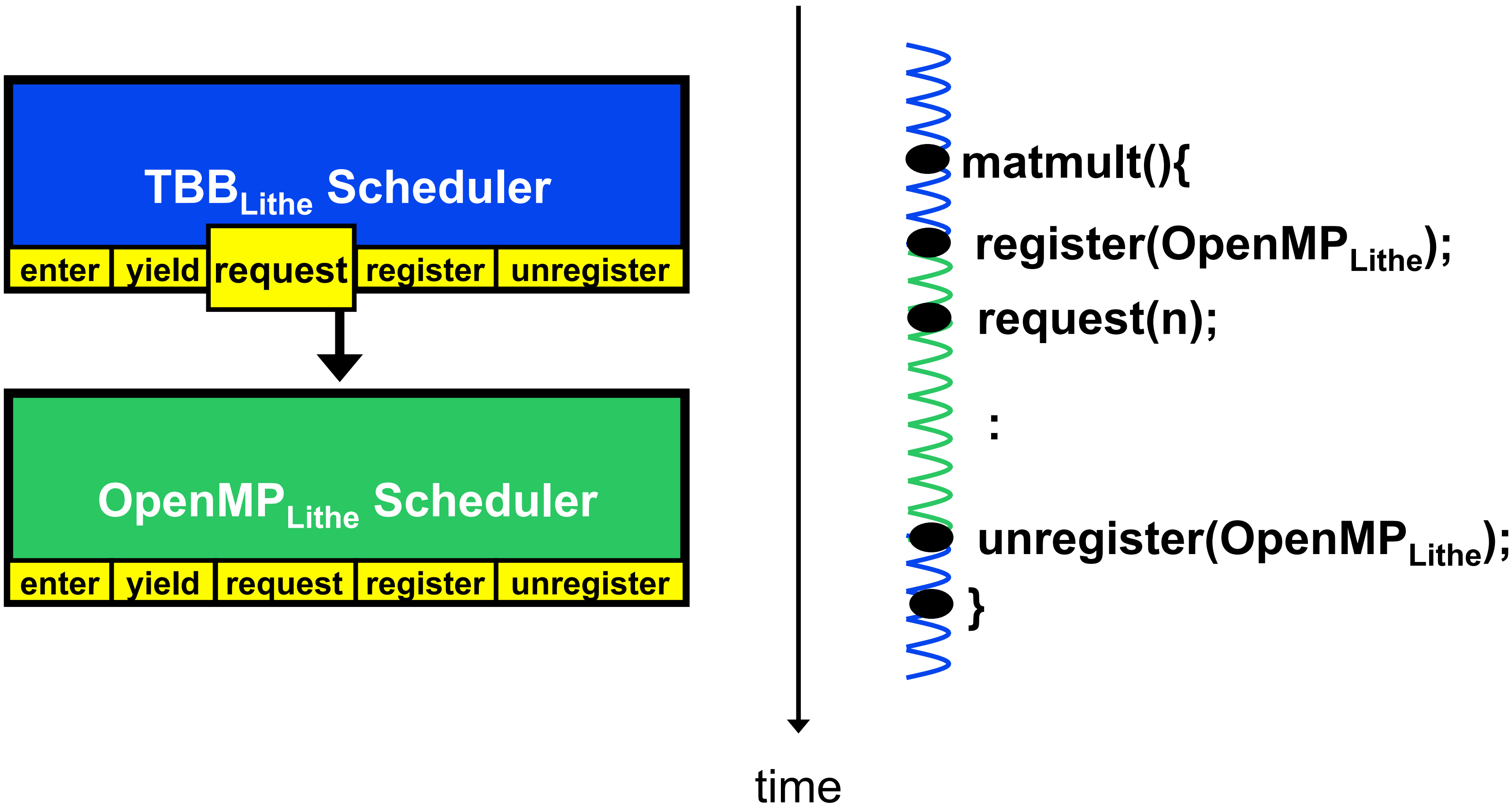


Register / Unregister



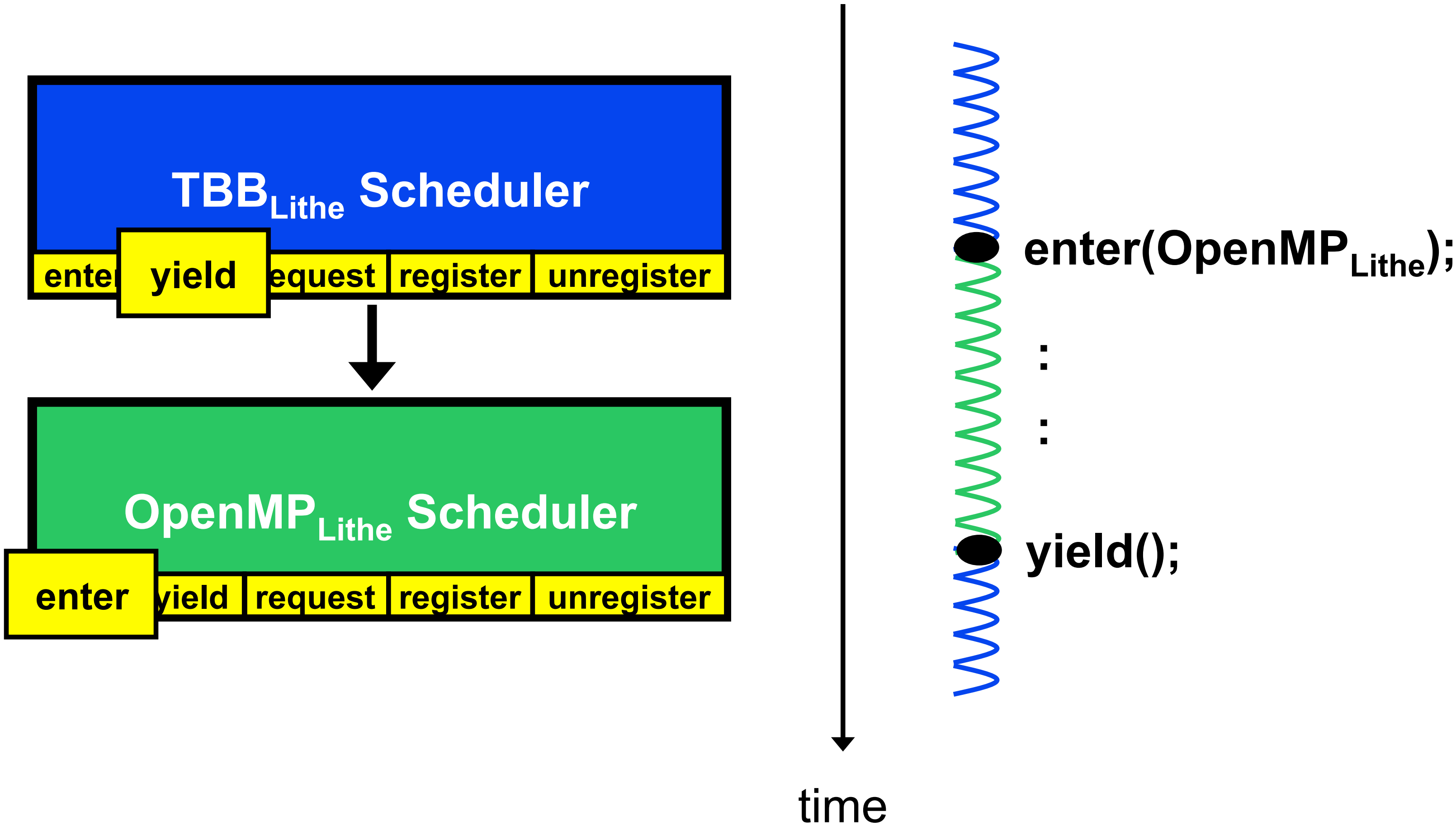
Register dynamically adds the new scheduler to the hierarchy.

Request



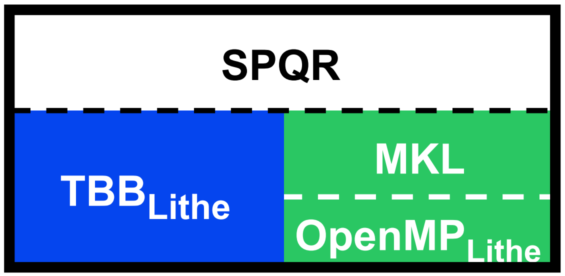
Request asks for more harts from the parent scheduler.

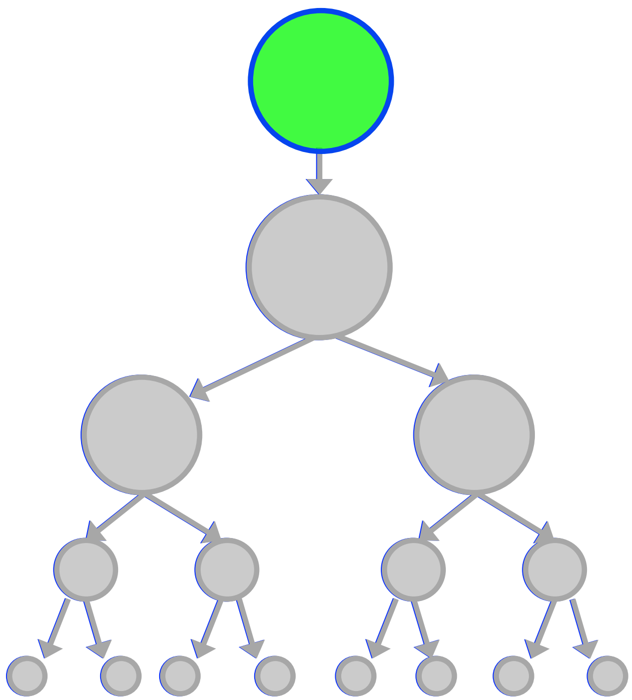
Enter / Yield

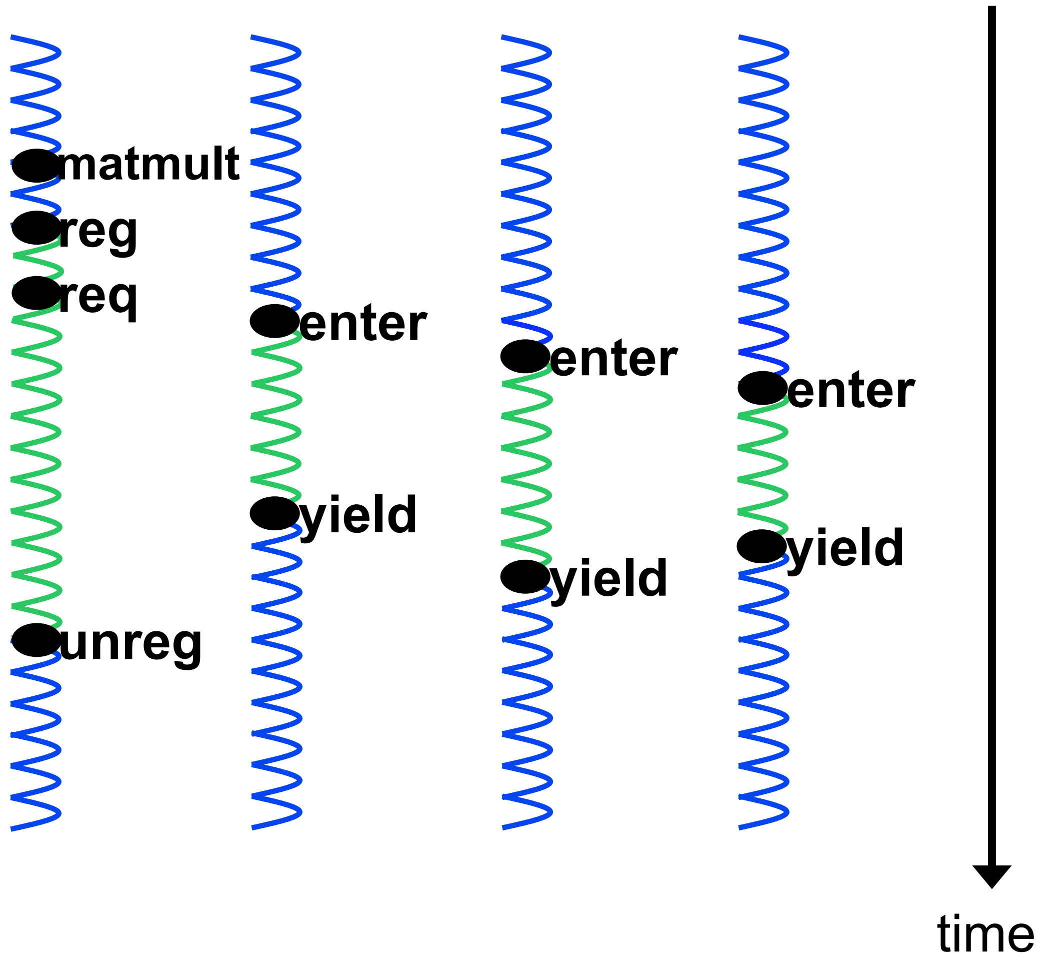


Enter/Yield transfers additional harts between the parent and child.

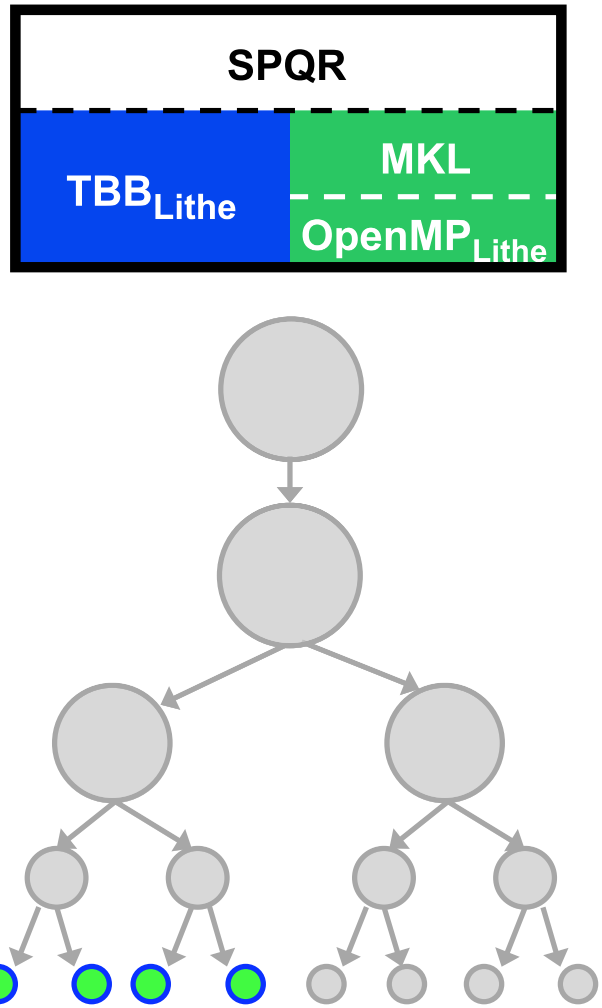
SPQR with Lithe

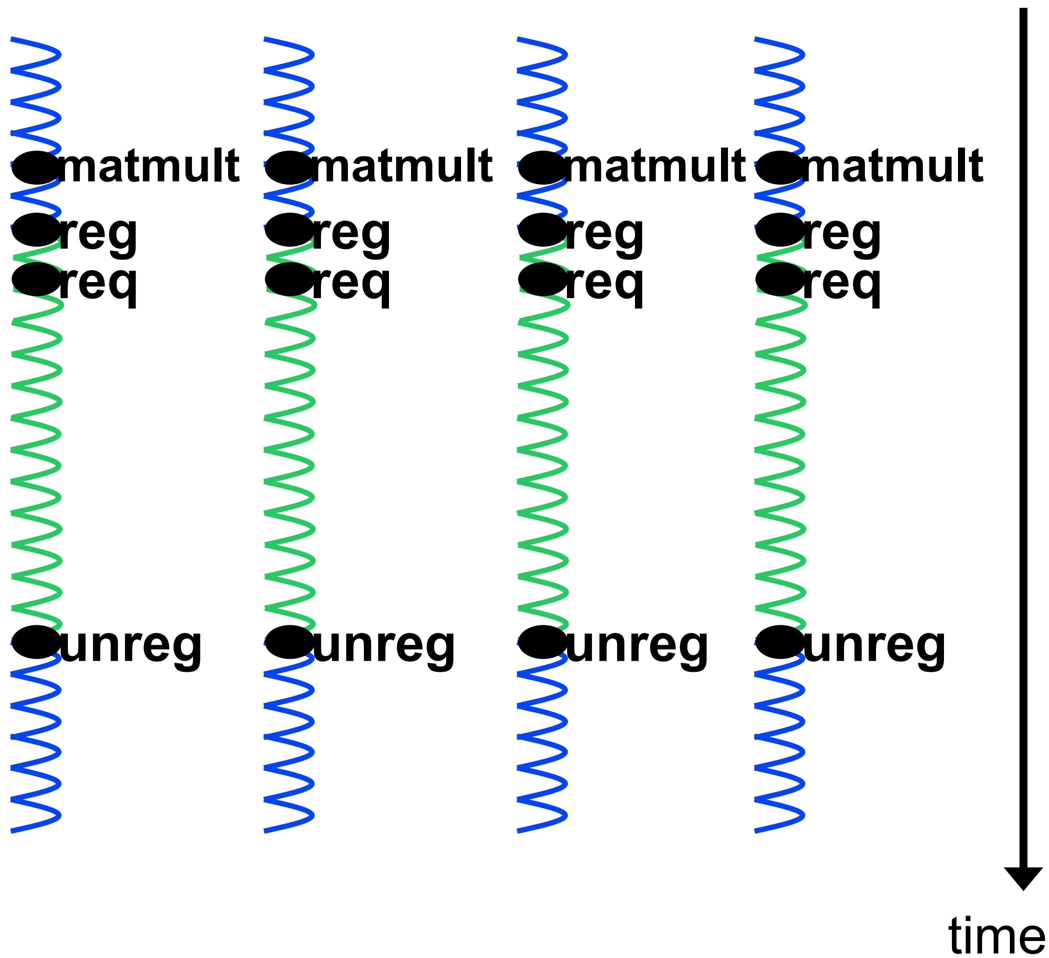






SPQR with Lithe



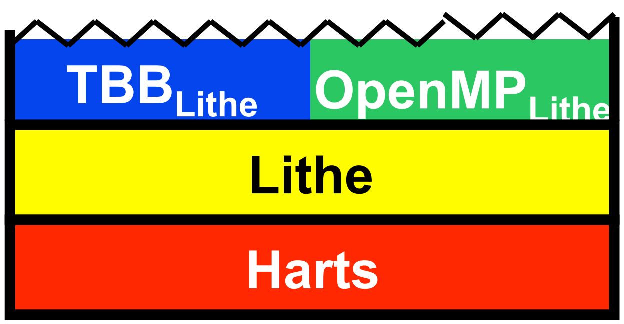


Talk Roadmap

- Problem: *Efficient* parallel composability is hard!
- Solution:
  - Harts
  - Lithe
- **Evaluation**

Implementation

- **Harts:** simulated using pinned Pthreads on x86-Linux
  ~600 lines of C & assembly
- **Lithe:** user-level library **(register, unregister, request, enter, yield, ...)**
  ~2000 lines of C, C++, assembly
- **$\text{TBB}_{\text{Lithe}}$**
  ~1500 / ~8000 relevant lines added/removed/modified
- **$\text{OpenMP}_{\text{Lithe}}$** (GCC4.4)
  ~1000 / ~6000 relevant lines added/removed/modified



No Lithe Overhead w/o Composing

All results on Linux 2.6.18, 8-core Intel Clovertown.

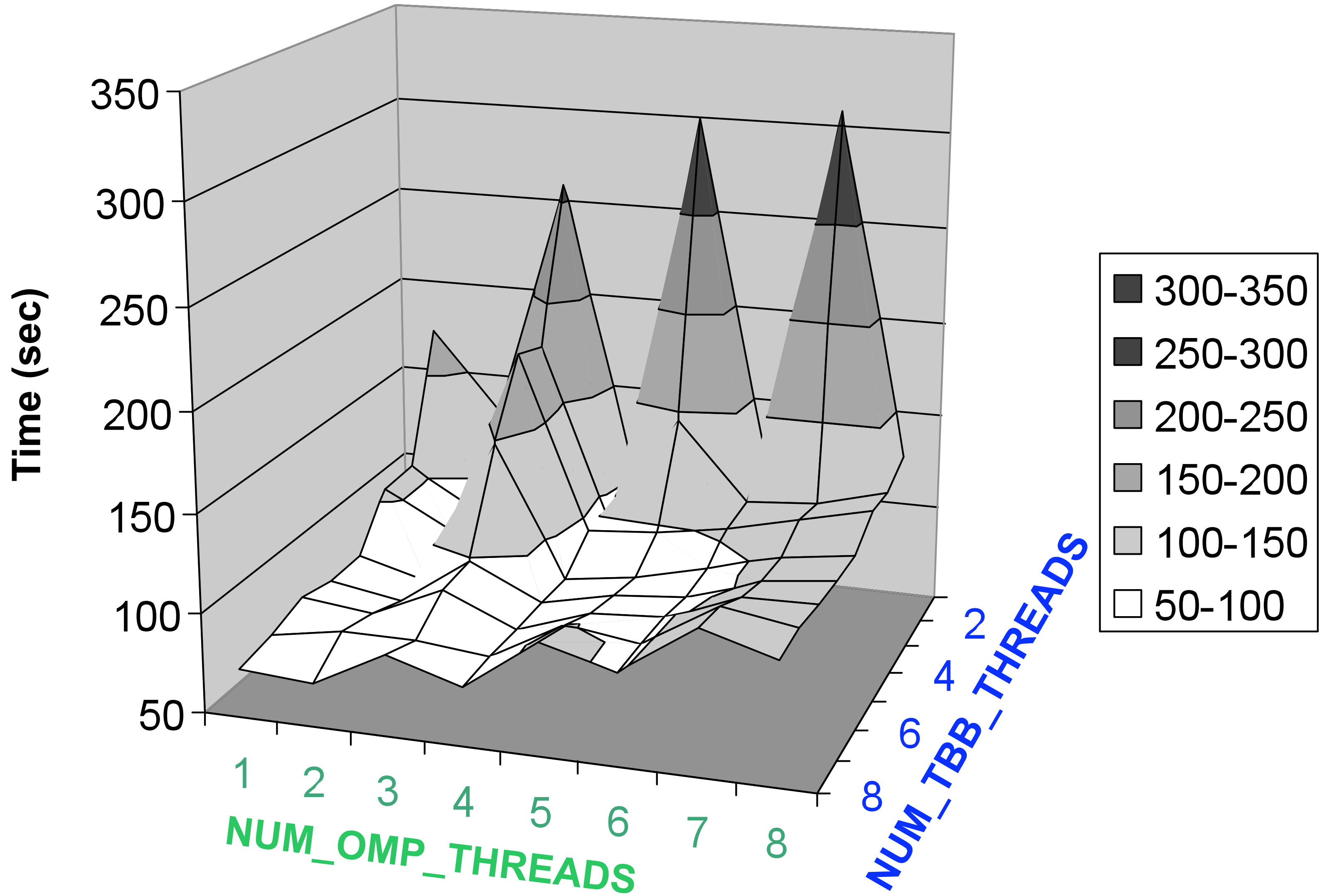
- **$\text{TBB}_{\text{Lithe}}$ Performance (µbench included with release)**

	tree sum	preorder	fibonacci
$\text{TBB}_{\text{Lithe}}$	54.80ms	228.20ms	8.42ms
TBB	54.80ms	242.51ms	8.72ms

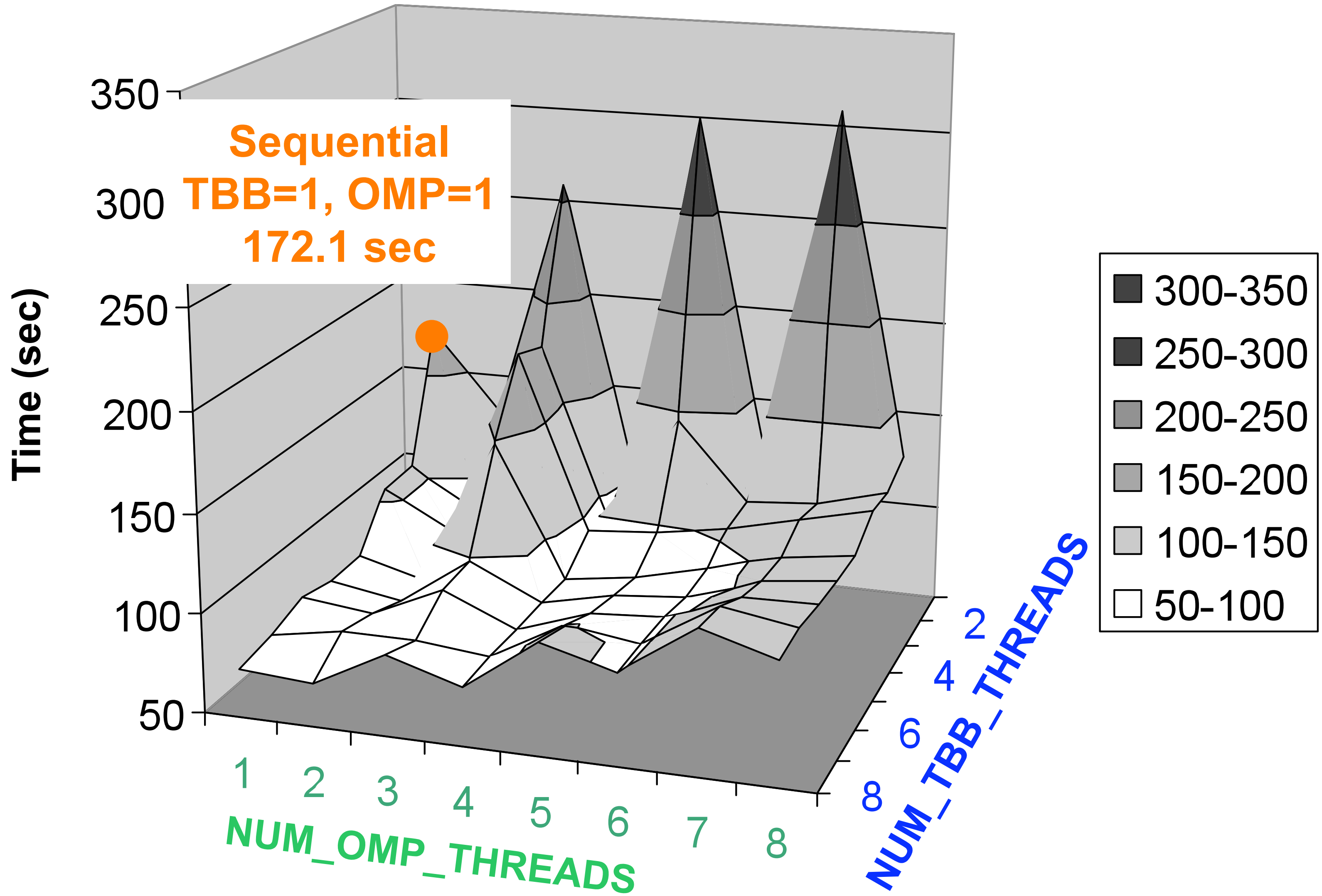
- **$\text{OpenMP}_{\text{Lithe}}$ Performance (NAS parallel benchmarks)**

	conjugate gradient (cg)	LU solver (lu)	multigrid (mg)
$\text{OpenMP}_{\text{Lithe}}$	57.06s	122.15s	9.23s
OpenMP	57.00s	123.68s	9.54s

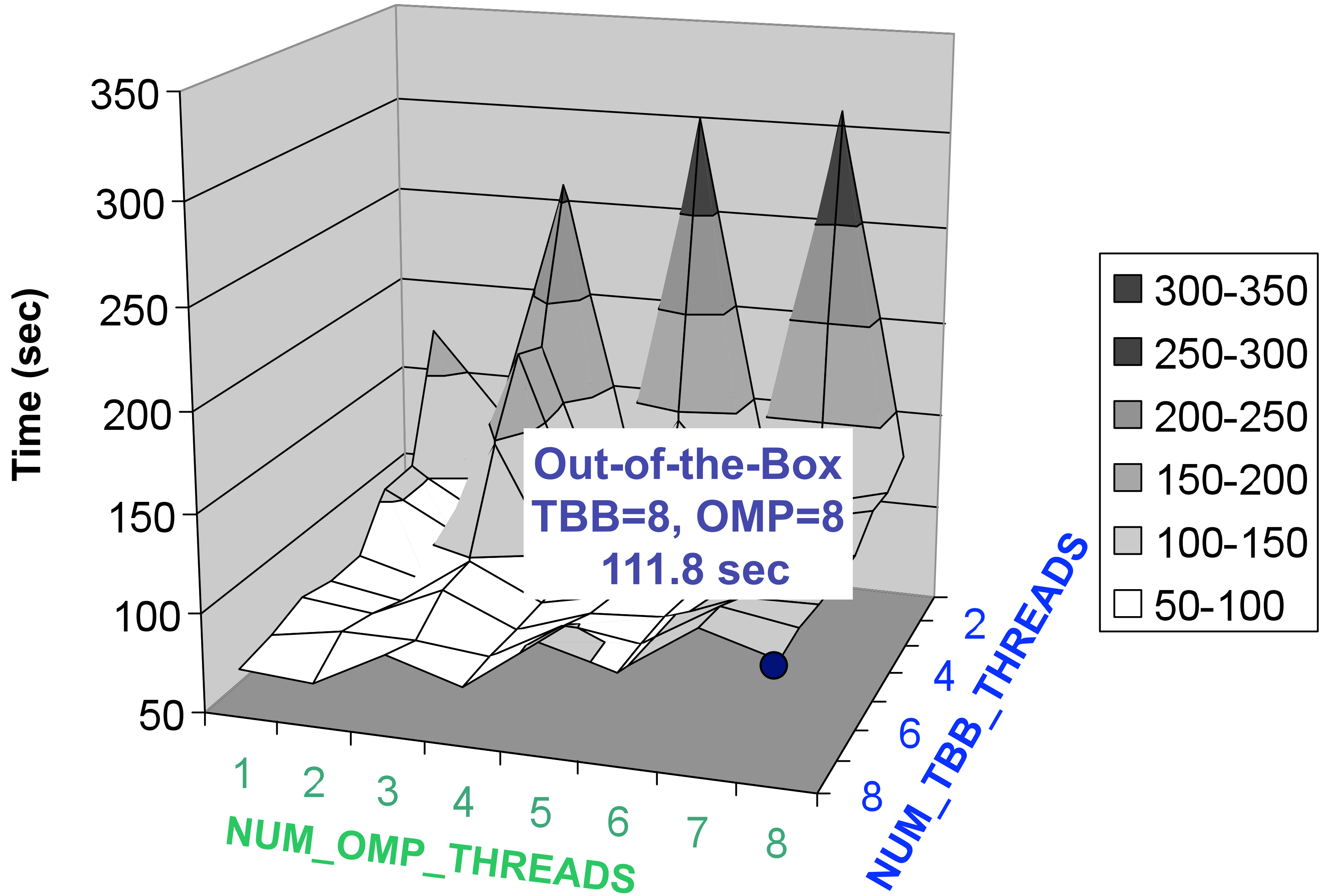
Performance Characteristics of SPQR (Input = ESOC)



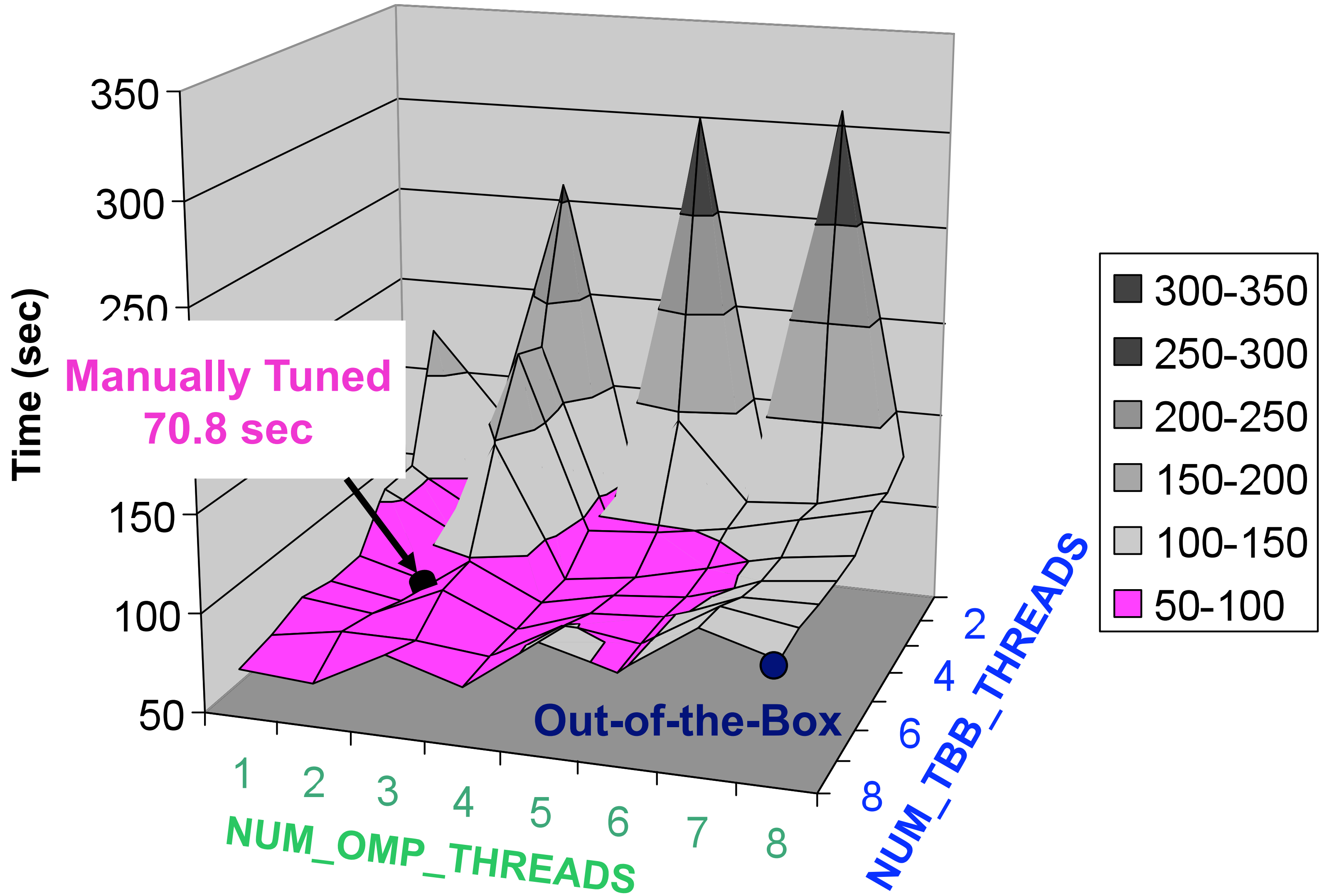
Performance Characteristics of SPQR (Input = ESOC)



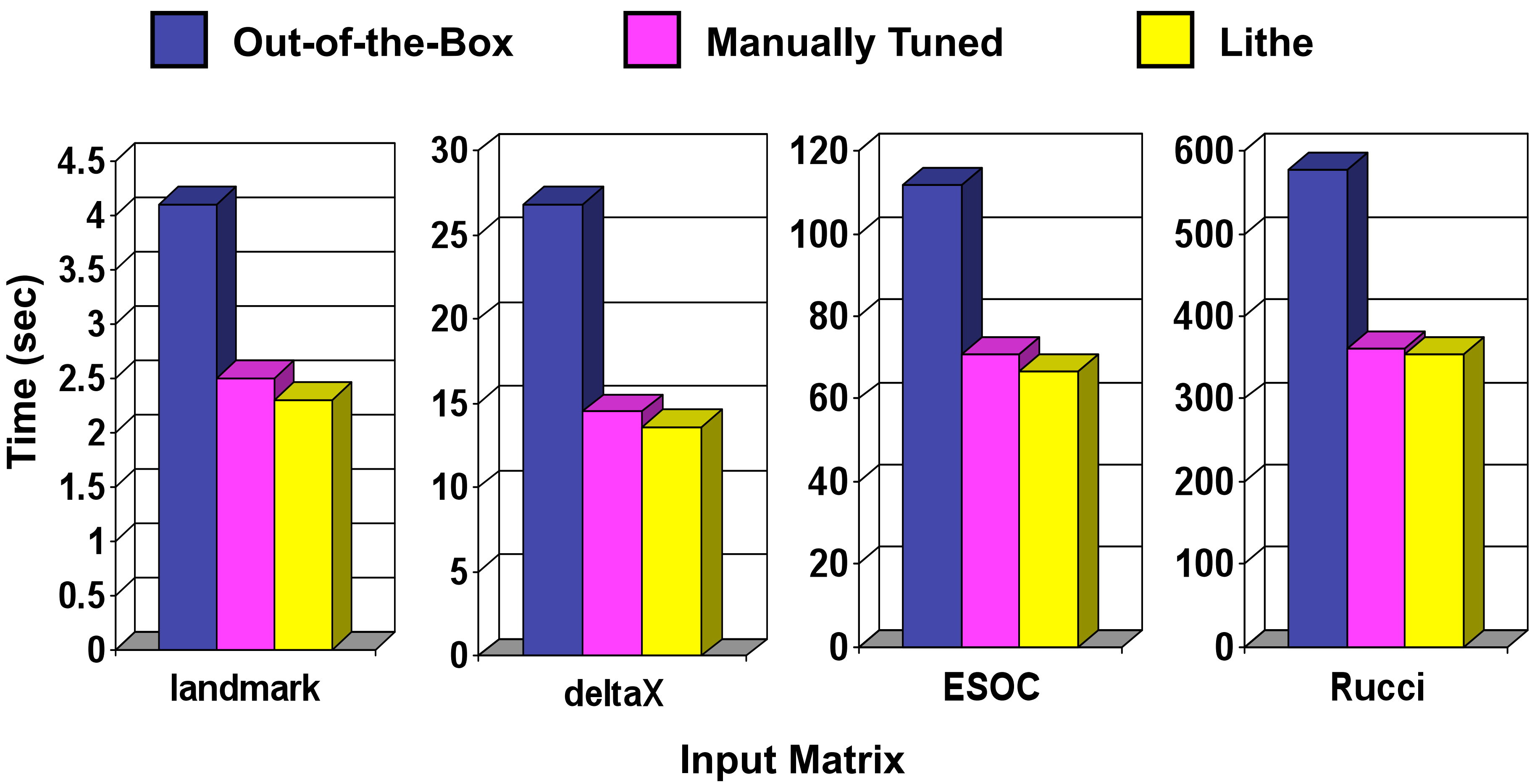
Performance Characteristics of SPQR (Input = ESOC)



Performance Characteristics of SPQR (Input = ESOC)



Performance of SPQR with Lithe

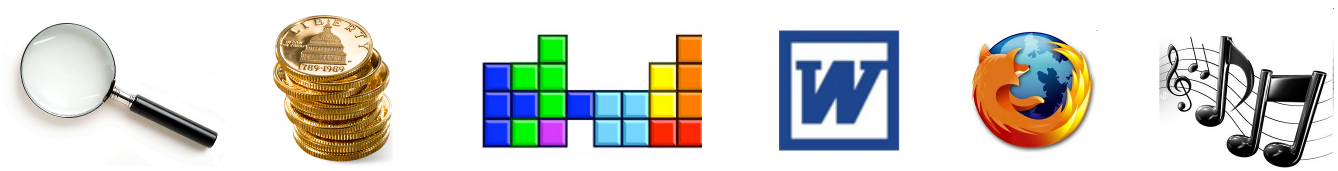


CSAIL

Future Work

BERKELEY PAR LAB

SPQR



$\text{TBB}_{\text{Lithe}}$				
enter	yield	req	req	unreq

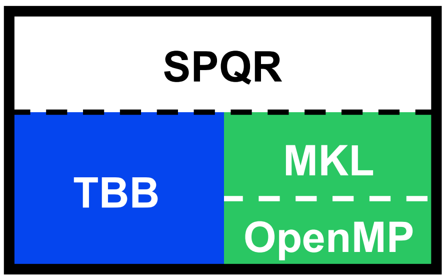
$\text{OpenMP}_{\text{Lithe}}$				
enter	yield	req	req	unreq

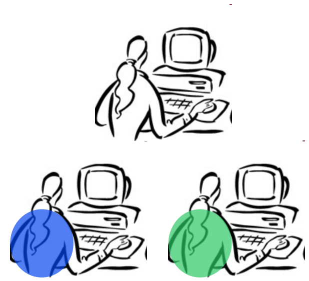
Ct_{Lithe}				
enter	yield	req	req	unreq

$\text{Cilk}_{\text{Lithe}}$				
enter	yield	req	req	unreq

Conclusion

- Composability essential for parallel programming to become widely adopted.

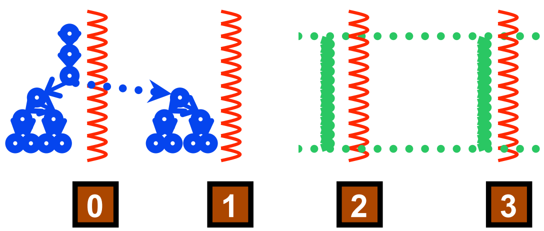




functionality

resource management

- Parallel libraries need to share resources cooperatively.



- Lithe project contributions
  - **Harts**: better resource model for parallel programming
  - **Lithe**: enables parallel codes to interoperate by standardizing the sharing of harts

CSAIL

Acknowledgements

BERKELEY PAR LAB

We would like to thank George Necula and the rest of Berkeley Par Lab for their feedback on this work.

Research supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227). This work has also been in part supported by a National Science Foundation Graduate Research Fellowship. Any opinions, findings, conclusions, or recommendations expressed in this publication are those of the authors and do not necessarily reflect the views of the National Science Foundation. The authors also acknowledge the support of the Gigascale Systems Research Focus Center, one of five research centers funded under the Focus Center Research Program, a Semiconductor Research Corporation program.