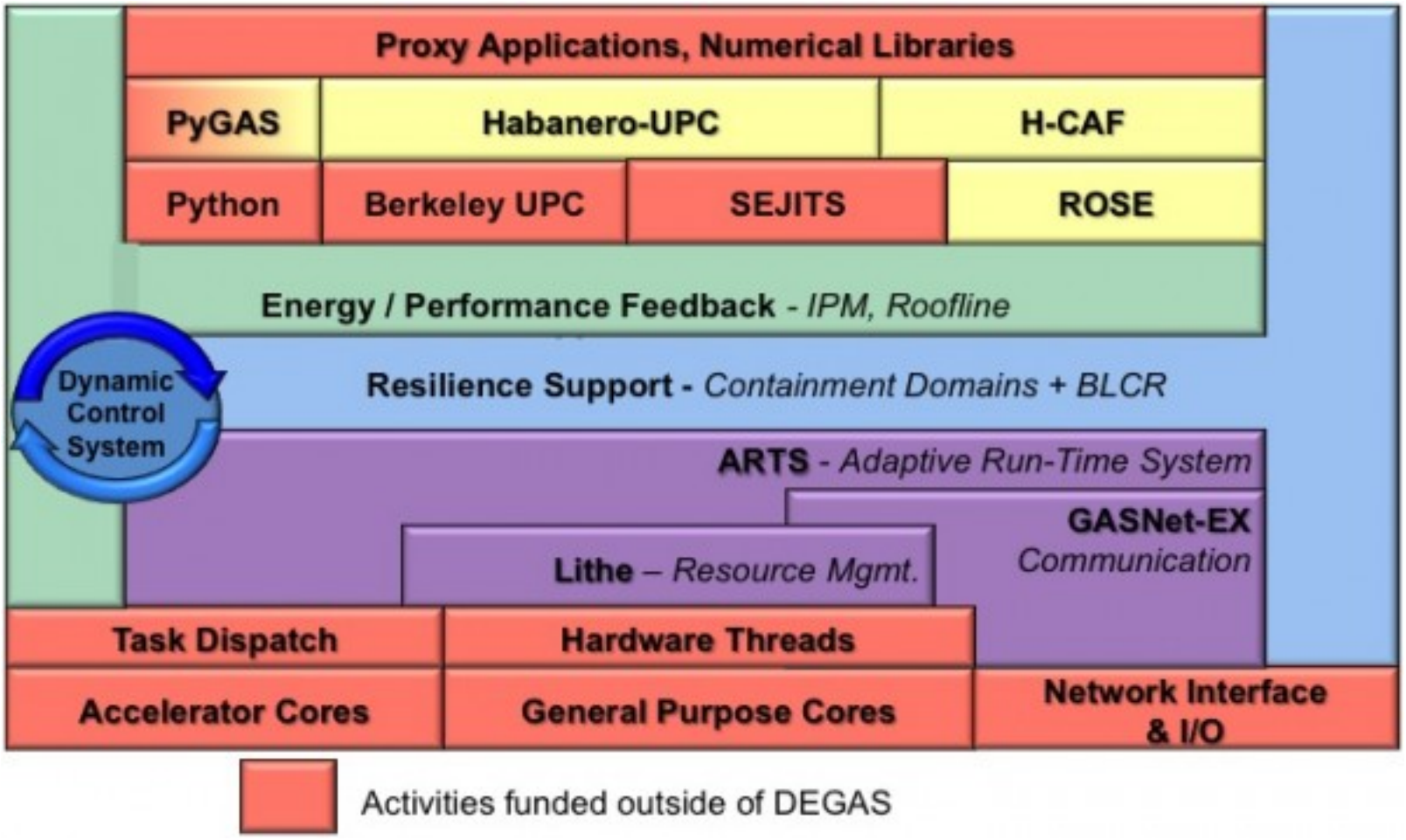
# LITHE

# *Composing Parallel Software Efficiently*

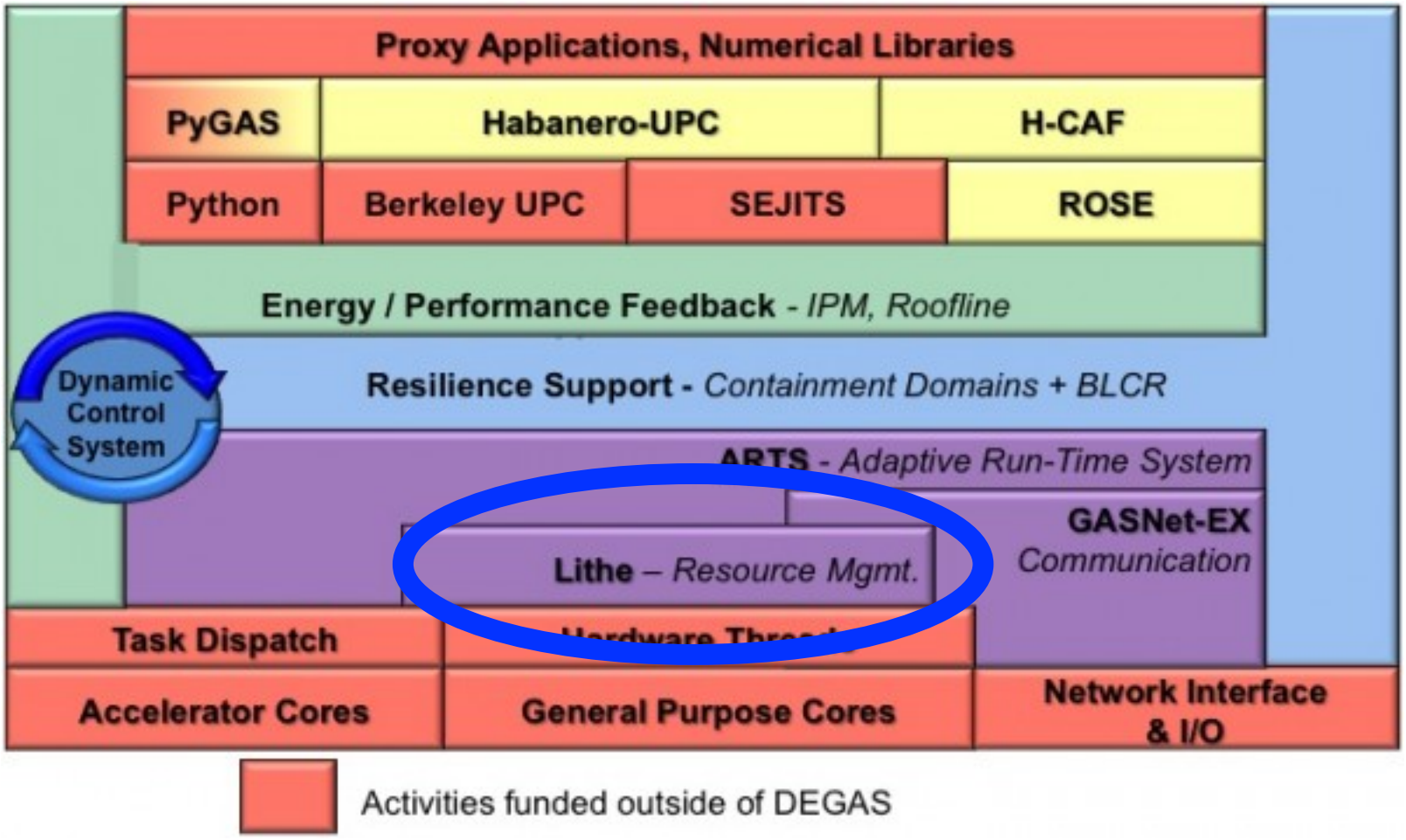
Kevin Klues
klueska@cs.berkeley.edu

DEGAS Summer Retreat
June 4th, 2013
http://lithe.eecs.berkeley.edu

# Where Does Lithe Fit?

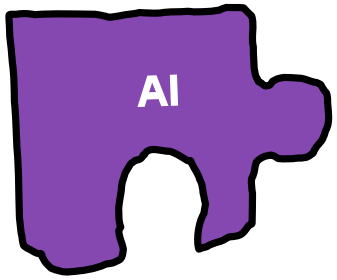


# Where Does Lithe Fit?



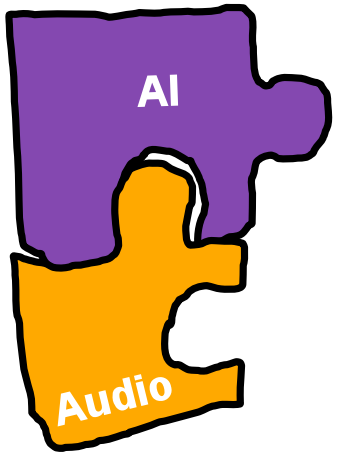
# Composition is King

```
game()   {
    forall frames:
      AI.compute()  ;




}
```



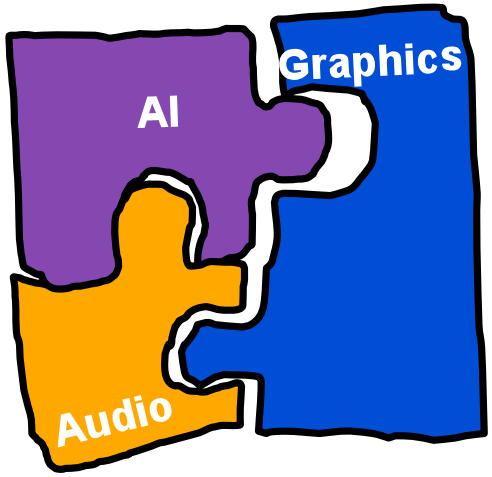
# Composition is King

```
game()   {
    forall frames:
      AI.compute() ;
      Audio.play() ;




}
```



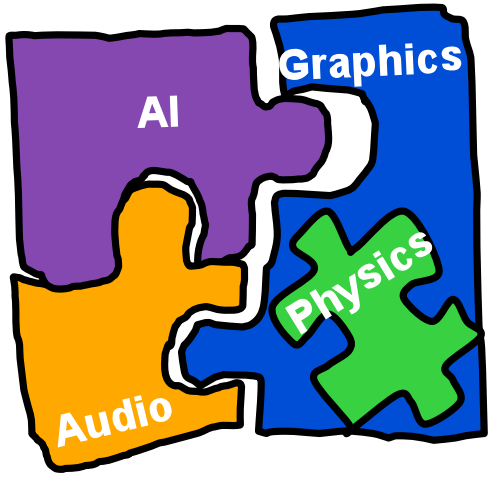
# Composition is King

```
game()  {
    forall frames:
      AI.compute() ;
      Audio.play() ;
      Graphics.render();


}
```



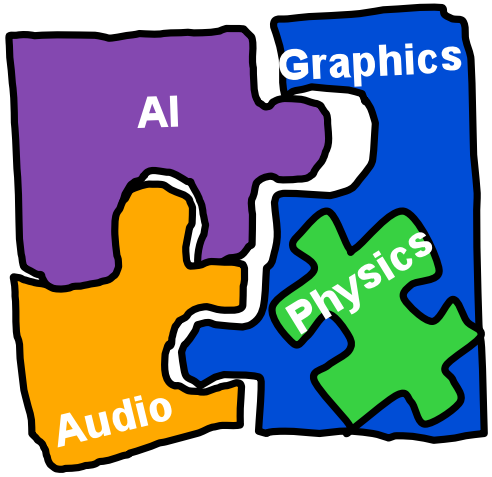
# Composition is King

```
game()  {
    forall frames:
      AI.compute() ;
      Audio.play() ;
      Graphics.render()  {
        Physics.calc ();
        :
      }
}
```



# Composition is King

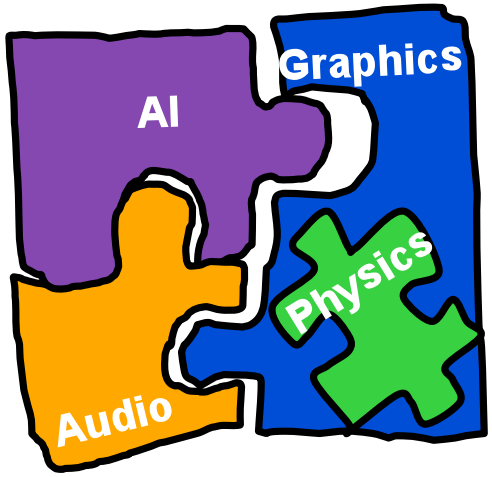
```
game()  {
    forall frames:
      AI.compute() ;
      Audio.play() ;
      Graphics.render()  {
        Physics.calc ();
        :
      }
}
```




- **Productivity:** Don't want to implement & understand everything.

# Composition is King

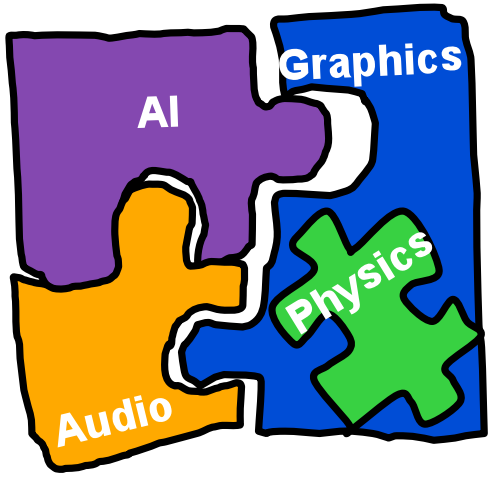
```
game()  {
    forall frames:
      AI.compute() ;
      Audio.play() ;
      Graphics.render()  {
        Physics.calc ();
        :
      }
}
```




- **Productivity:** Don't want to implement & understand everything.
- **Performance:** Leverage language & runtime optimizations within components.

# Composition is King

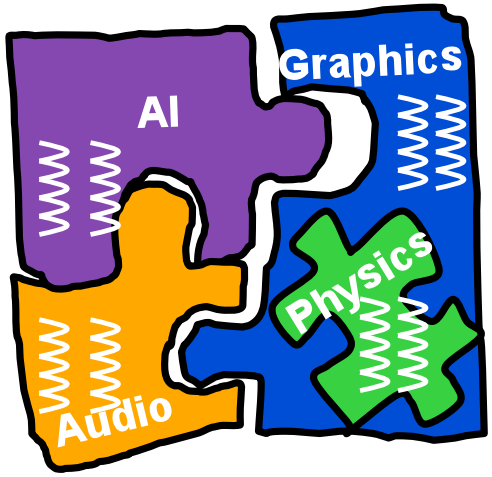
```
game()  {
    forall frames:
      AI.compute() ;
      Audio.play() ;
      Graphics.render()  {
        Physics.calc ();
        :
      }
}
```




- **Productivity:** Don't want to implement & understand everything.
- **Performance:** Leverage language & runtime optimizations within components.
- **Diversity:** Components may want to use different abstractions & languages.

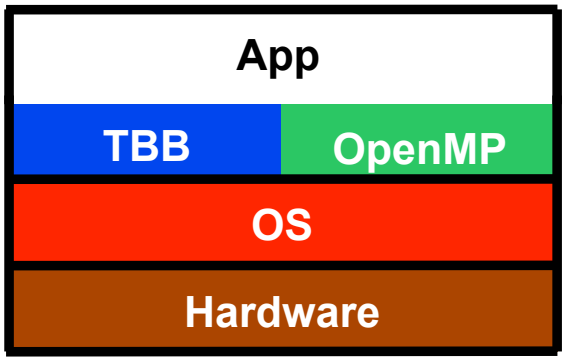
# Composition is King

```
game()  {
    forall frames:
      AI.compute()  ||
      Audio.play()  ||
      Graphics.render()  {
        Physics.calc ();
        :
      }
}
```

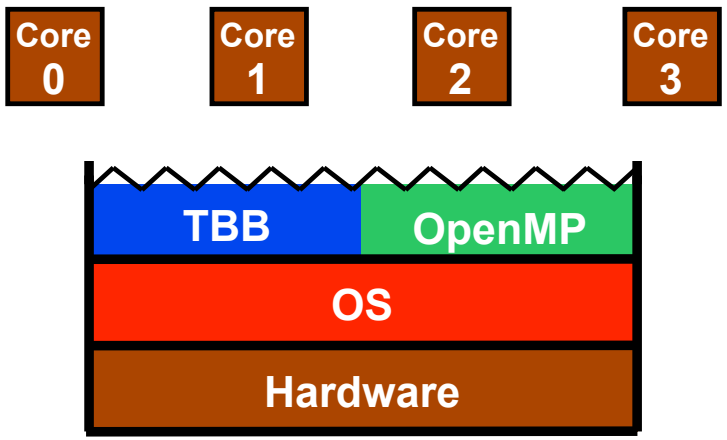



- **Productivity:** Don't want to implement & understand everything.
- **Performance:** Leverage language & runtime optimizations within components.
- **Diversity:** Components may want to use different abstractions & languages.

# Multiple Components Oversubscribe Resources

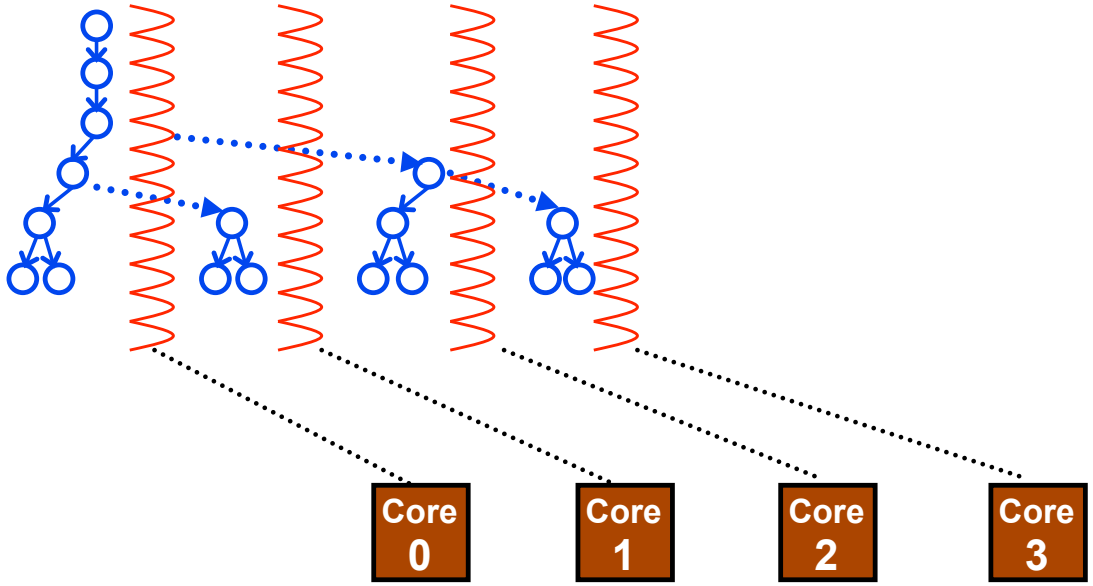


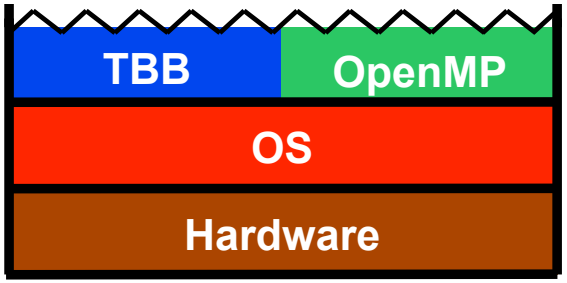
# Multiple Components Oversubscribe Resources



# Multiple Components Oversubscribe Resources

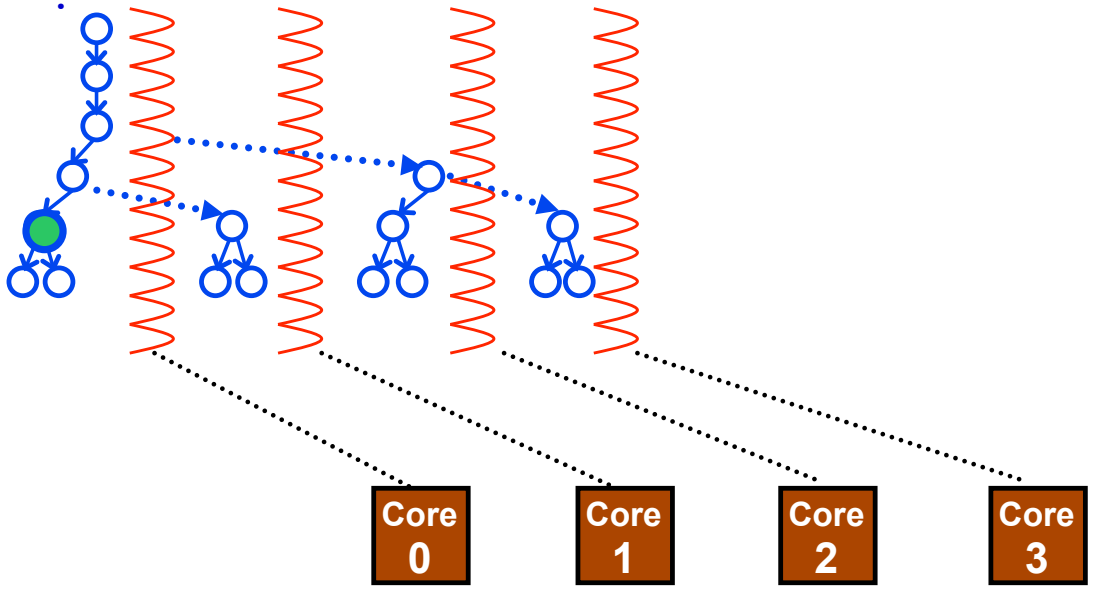
`tbb::task()`

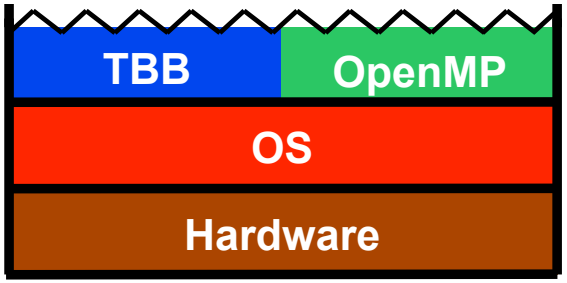


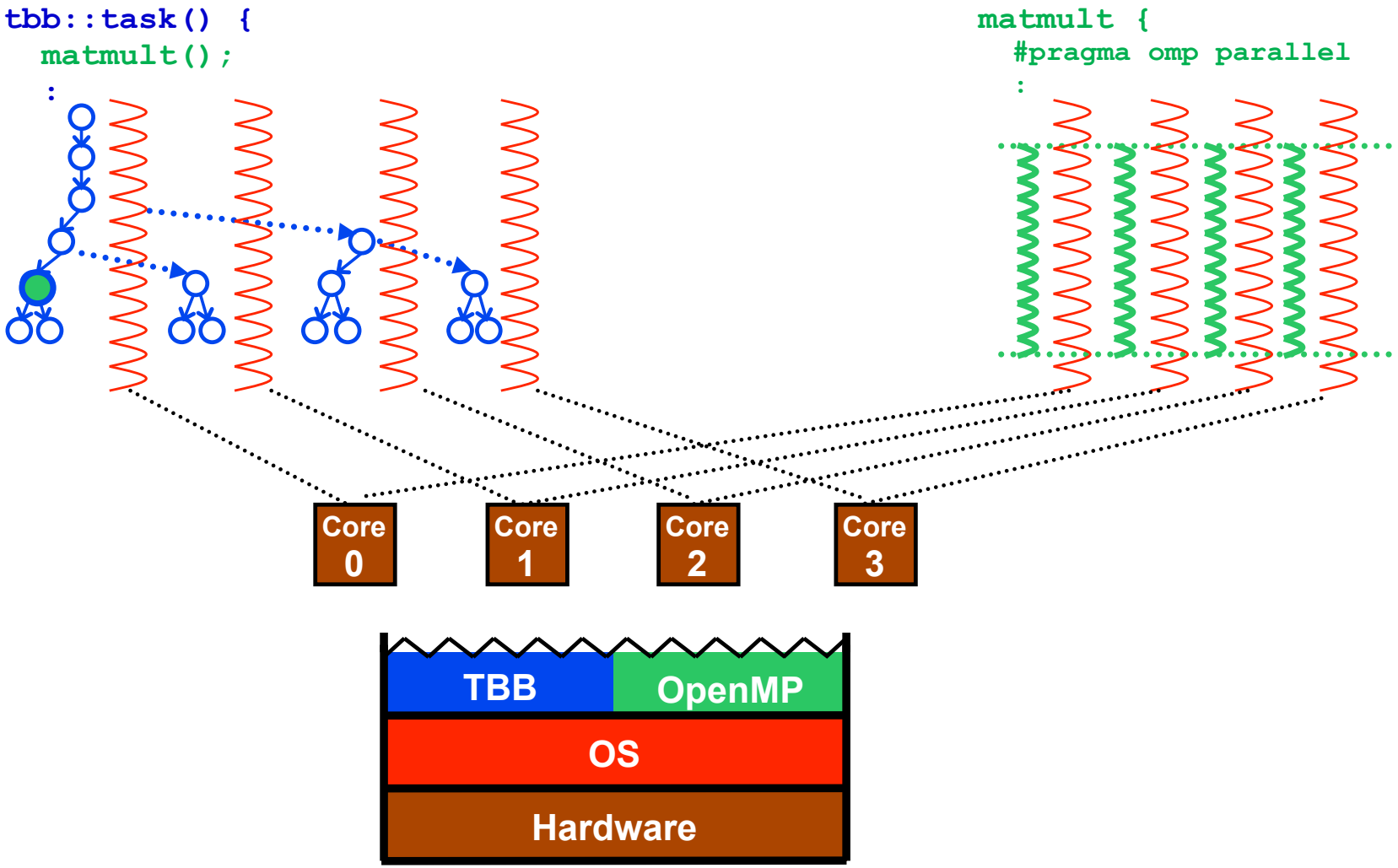
# Multiple Components Oversubscribe Resources

```
tbb::task() {
  matmult();
  :
```

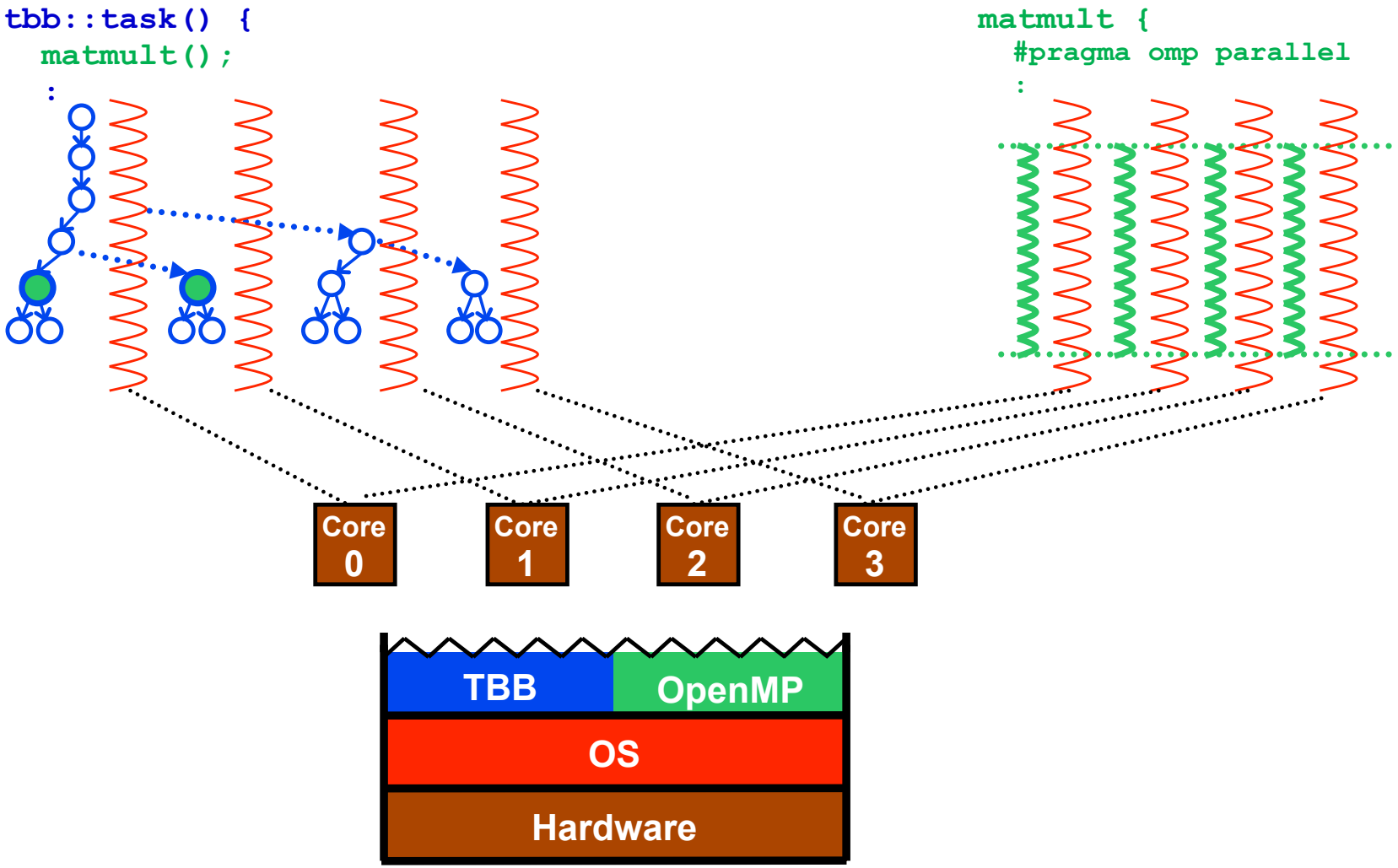


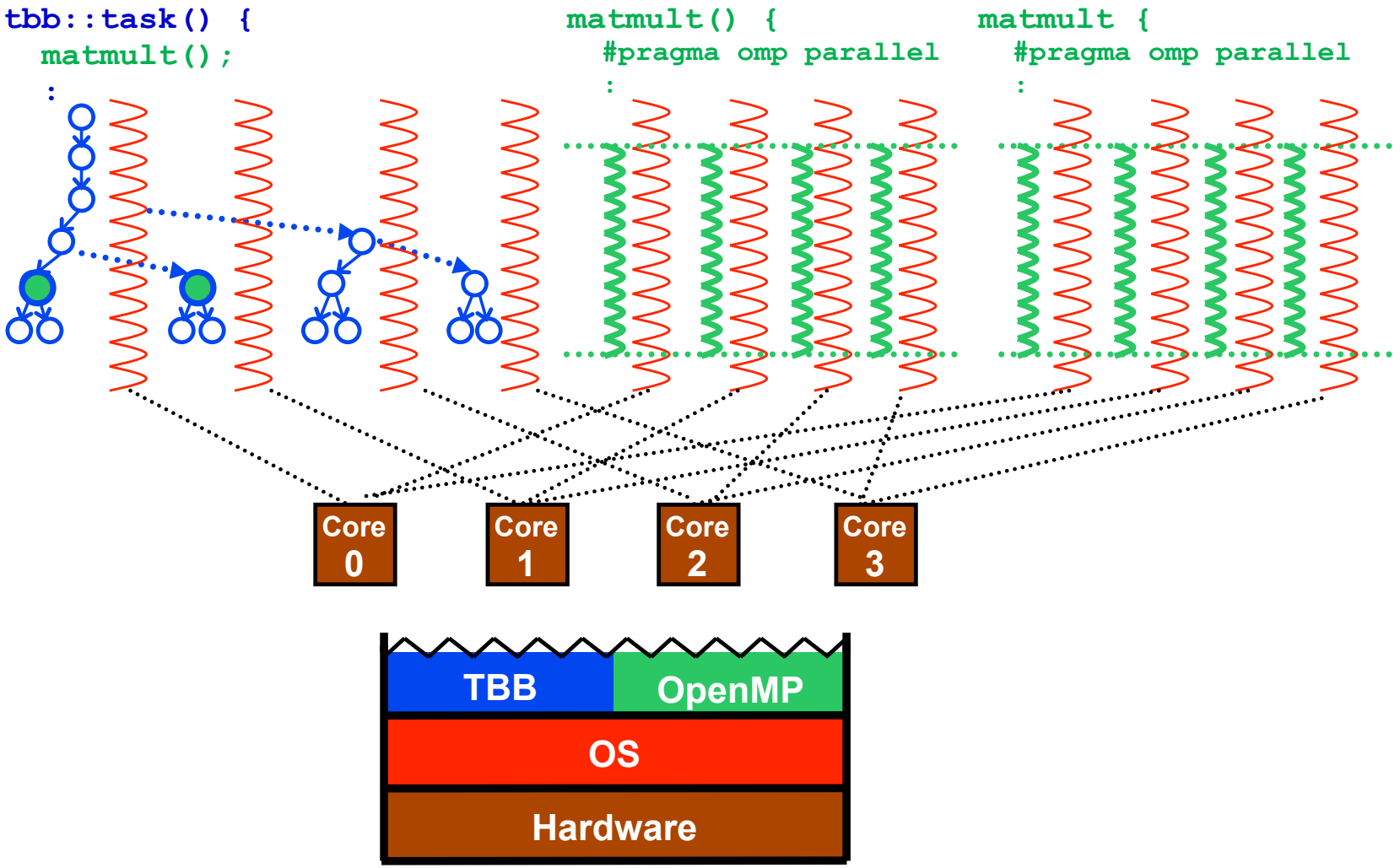
# Multiple Components Oversubscribe Resources



# Multiple Components Oversubscribe Resources



# Multiple Components Oversubscribe 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).**
One of the advantages of using the IntelMKL is that it is multithreaded using OpenMP*. OpenMP* requires buffers to perform some operations and allocates memory even for single-processor systems and single-thread applications. This memory allocation occurs once the first time the OpenMP software is encountered in the program. This memory allocation persists until the application terminates. In addition, the Windows* operating system will allocate a stack equal to the main stack for every additional thread created, so the amount of memory that is automatically allocated will depend on the main stack, the OpenMP allocations and the number of threads used.

**Using Threading with BLAS and LAPACK**
Intel MKL is threaded in a number of places: LAPACK (*GETRF, *POTRF, *GBTRF routines), Level 3 BLAS, DFTs, and FFTs. Intel MKL uses OpenMP* threading software. There are situations in which conflicts can exist that make the use of threads in Intel MKL problematic. We list them here with recommendations for dealing with these. First, a brief discussion of why the problem exists is appropriate.

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)**
The OpenMP* software responds to the environment variable OMP_NUM_THREADS:

- Windows*: Open the Environment panel of the System Properties box of the Control Panel on Microsoft* Windows NT*, or it can be set in the shell the program is running in with the command: set OMP_NUM_THREADS=<number of threads to use>.
- Linux*: To set and export the variableP "export OMP_NUM_THREADS=<number of threads to use>".

**Note:** Setting the variable when running on Microsoft* Windows* 98 or Windows* Me is meaningless, since multiprocessing is not supported.

**Changing the Number of Processors for Threading During Runtime**
It is not possible to change the number of processors during runtime using the environment variable OMP_NUM_THREADS. You can call OpenMP API functions from your program to change the number of threads during runtime. The following sample code demonstrates changing the number of threads during runtime using the omp_set_num_threads() routine:

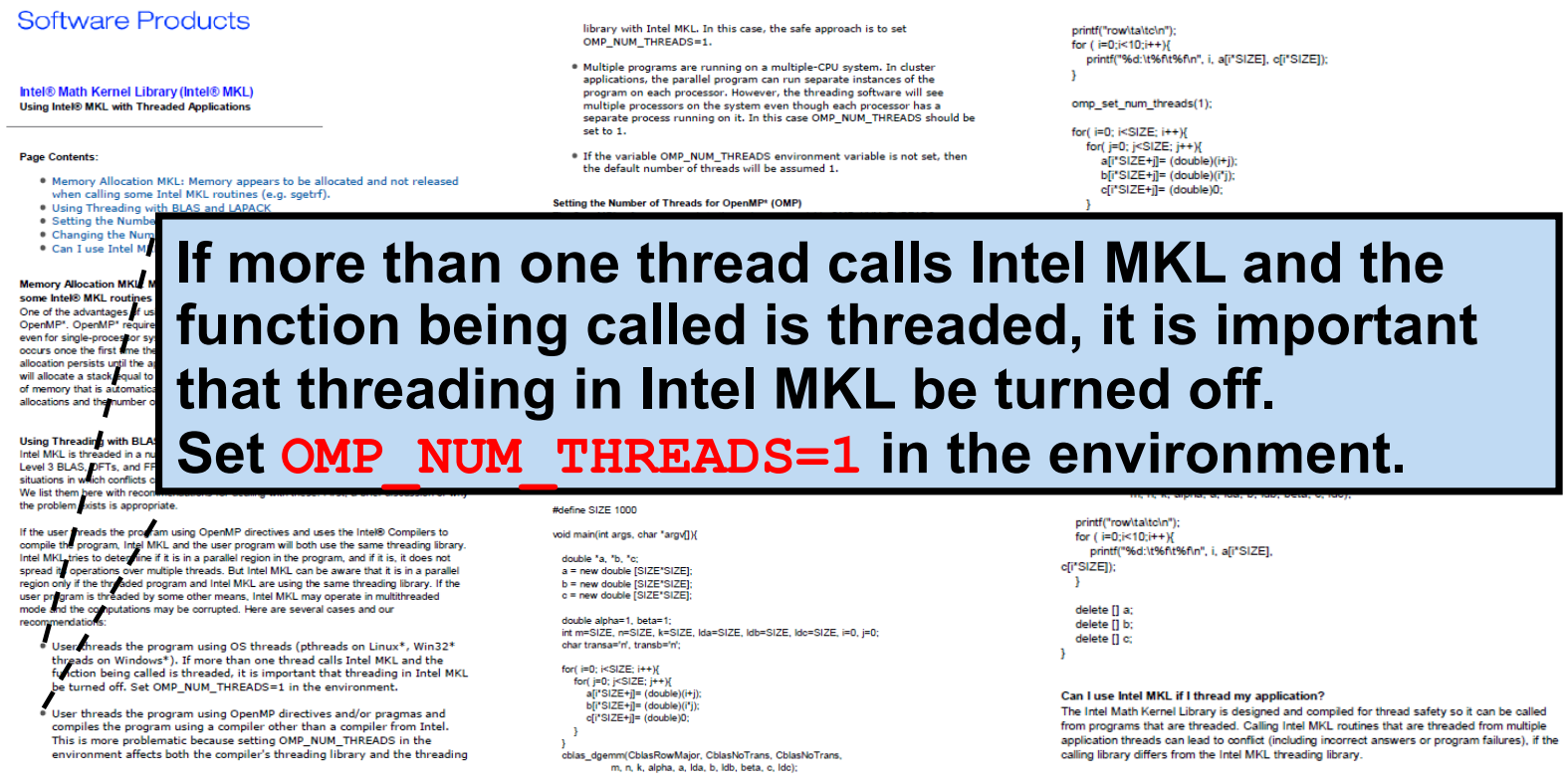
```
#include "omp.h"
#include "mkl.h"
#include <stdio.h>

#define SIZE 1000

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;
    }
  }
  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(2);
  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]);
  }

  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.

# 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 Number
- Changing the Num
- Can I use Intel M

Memory Allocation MKL
some Intel® MKL routines
One of the advantages of us
OpenMP*. OpenMP* require
even for single-processor sy
occurs once the first time the
allocation persists until the a
will allocate a stack equal to
of memory that is automatica
allocations and the number c

Using Threading with BLA
Intel MKL is threaded in a nu
Level 3 BLAS, DFTs, and FF
situations in which conflicts c
We list them here with recom
the problem exists is appropriate.

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)

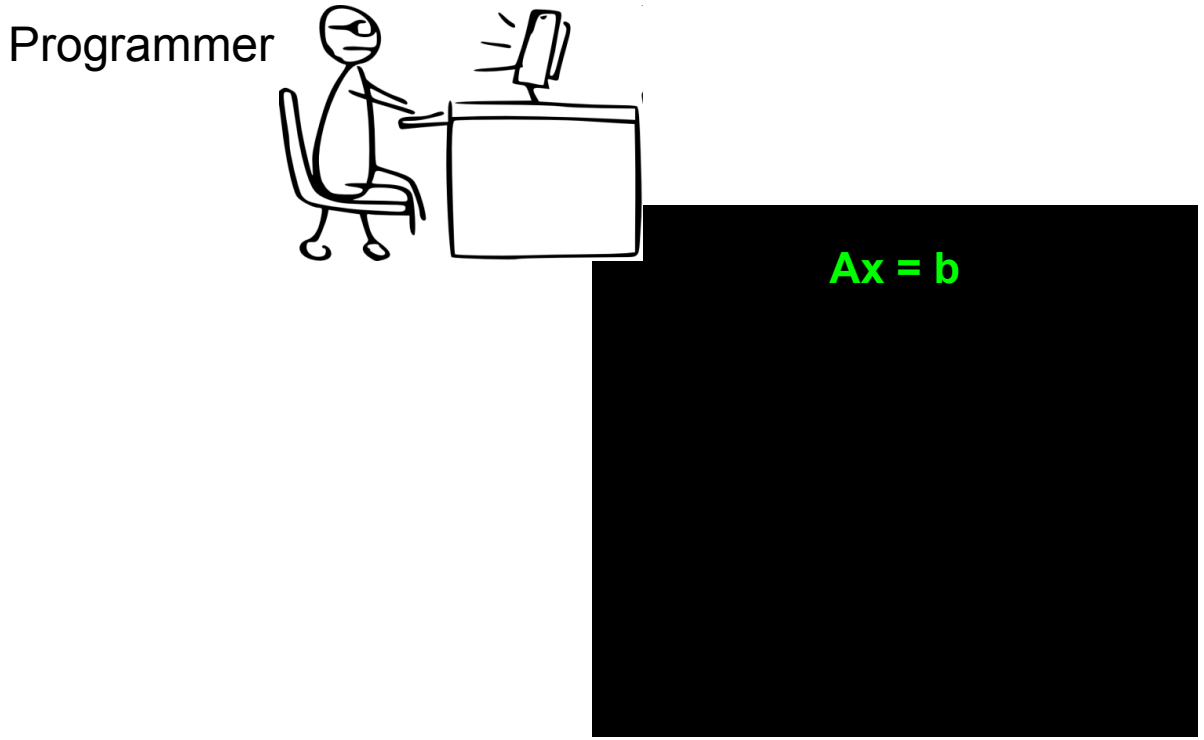
```
#define SIZE 1000

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;
}
```

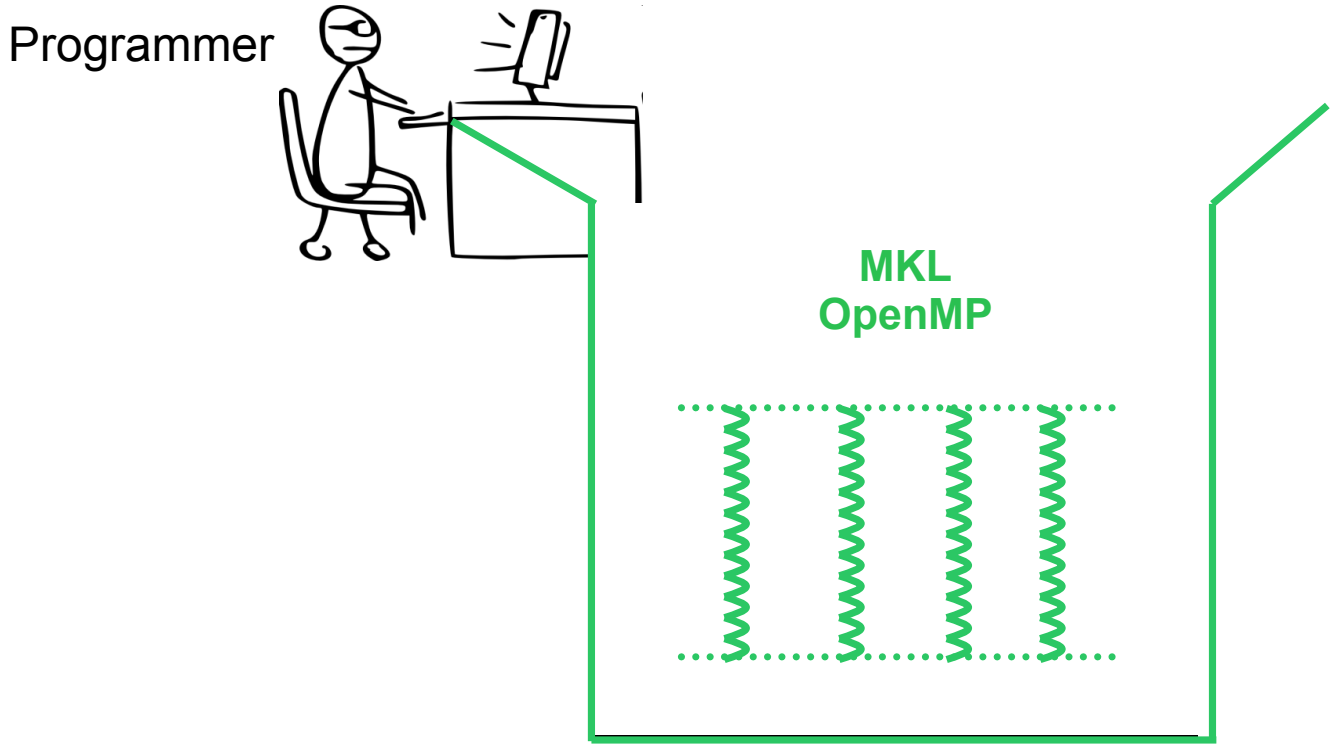
Can I use Intel MKL if I thread my application?
The Intel Math Kernel Library is designed and compiled for thread safety so it can be called from programs that are threaded. Calling Intel MKL routines that are threaded from multiple application threads can lead to conflict (including incorrect answers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

**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.**

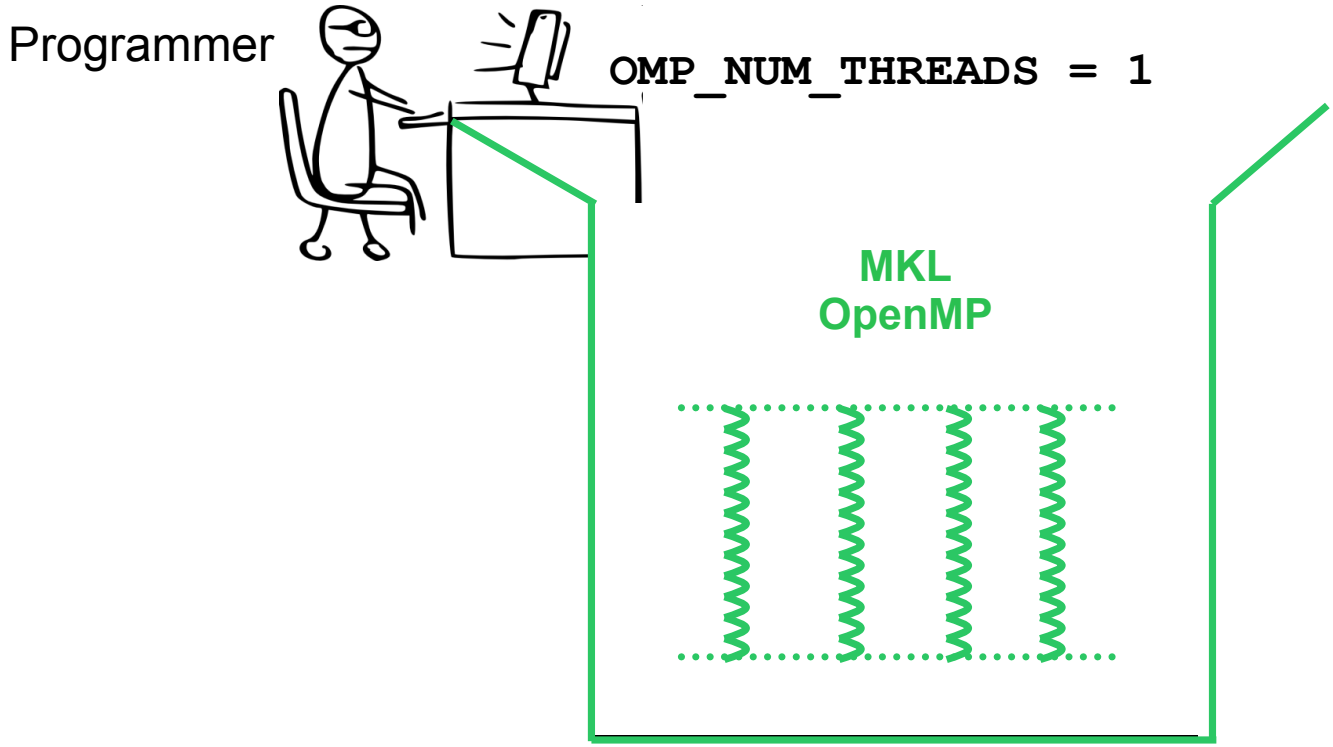
# Breaks Black Box Abstraction



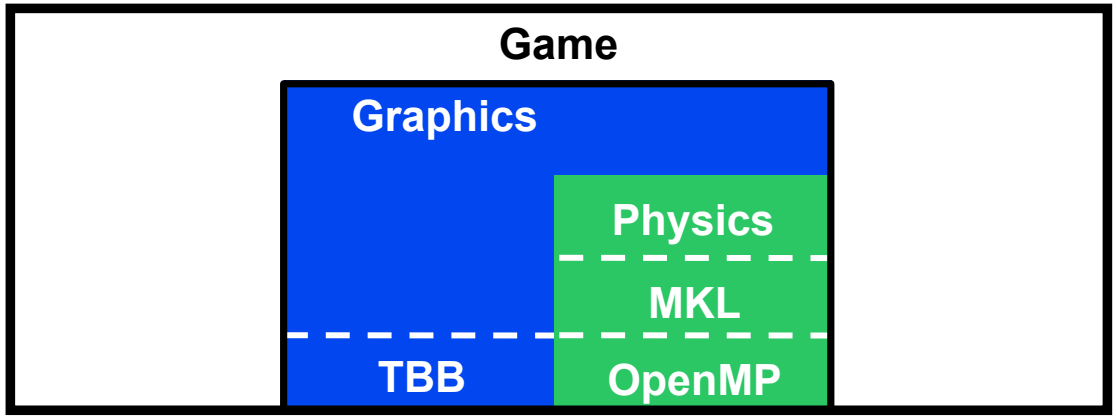
# Breaks Black Box Abstraction

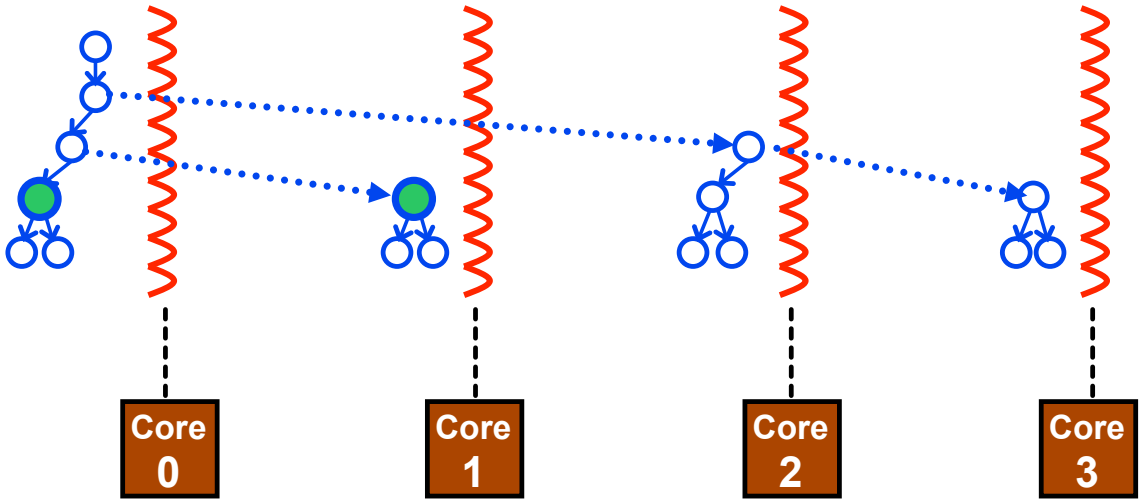


# Breaks Black Box Abstraction

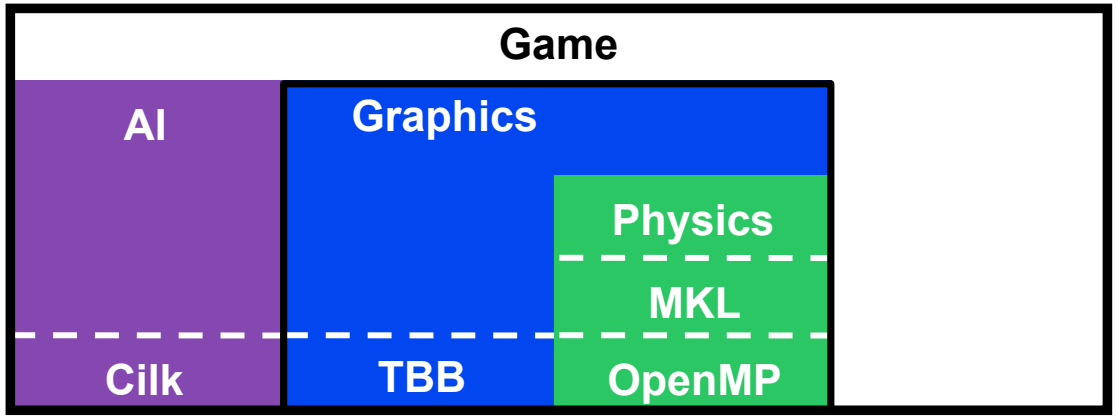


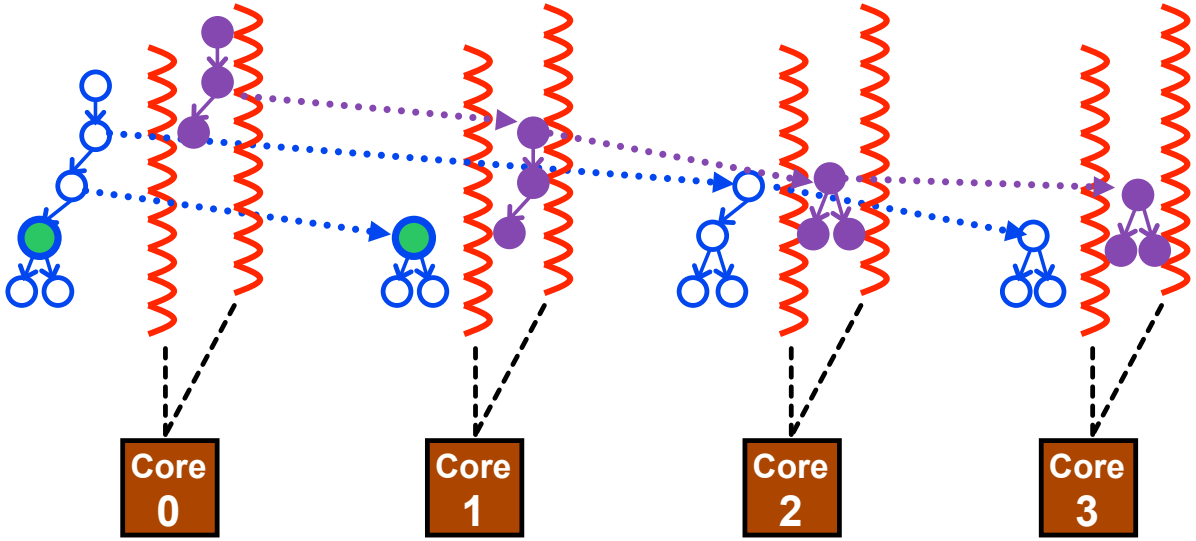
# Exports Problem to User



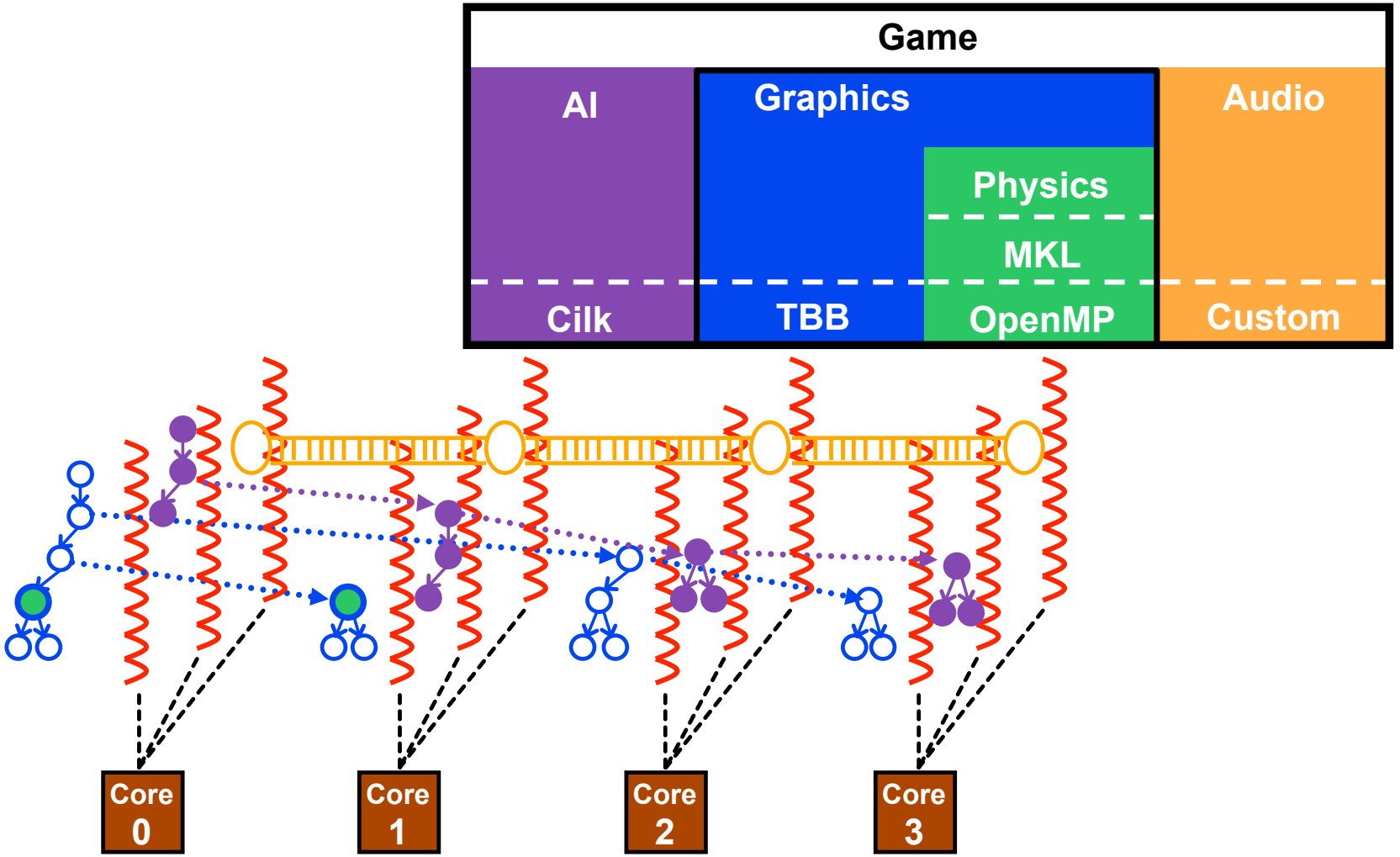



# Exports Problem to User

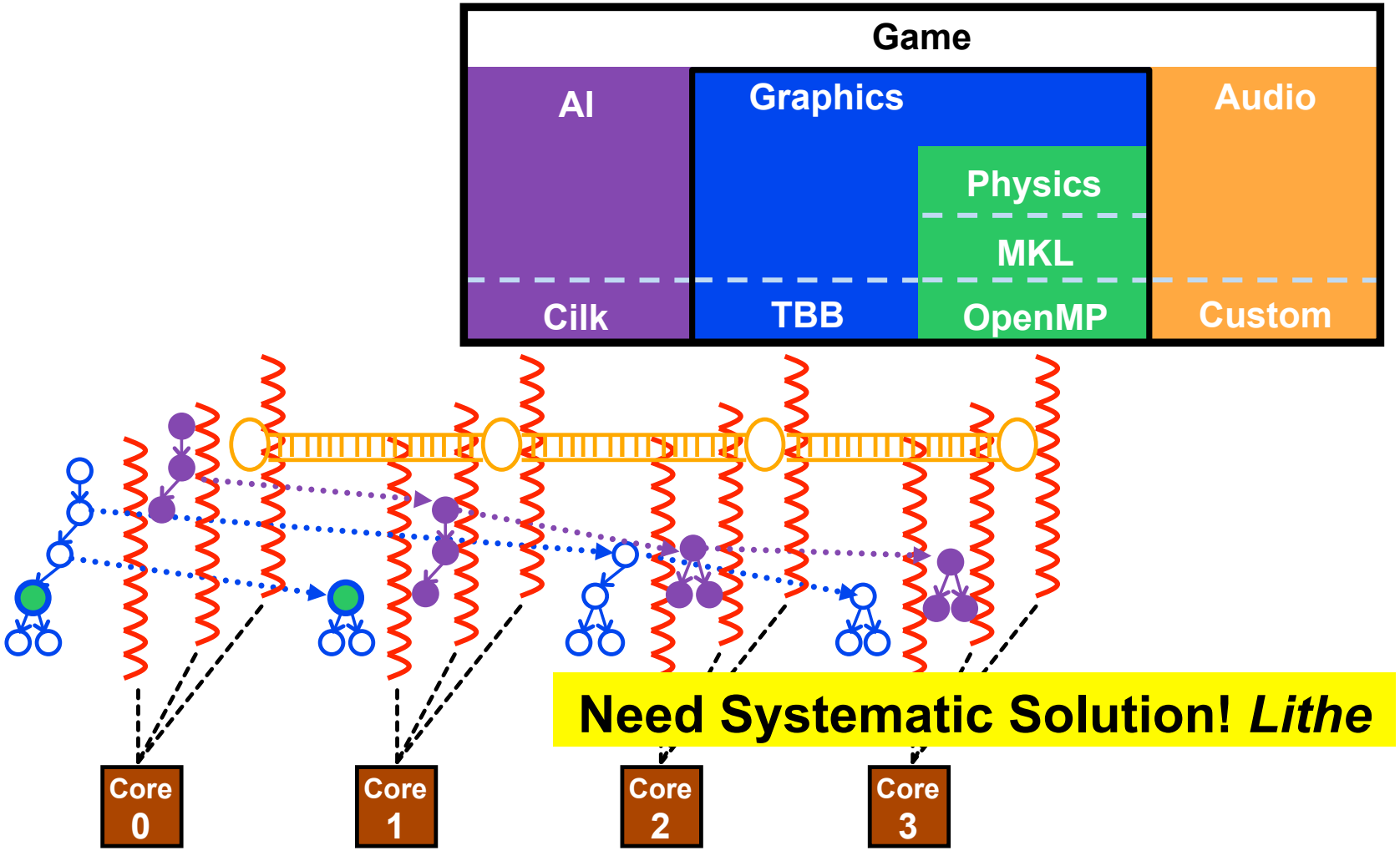


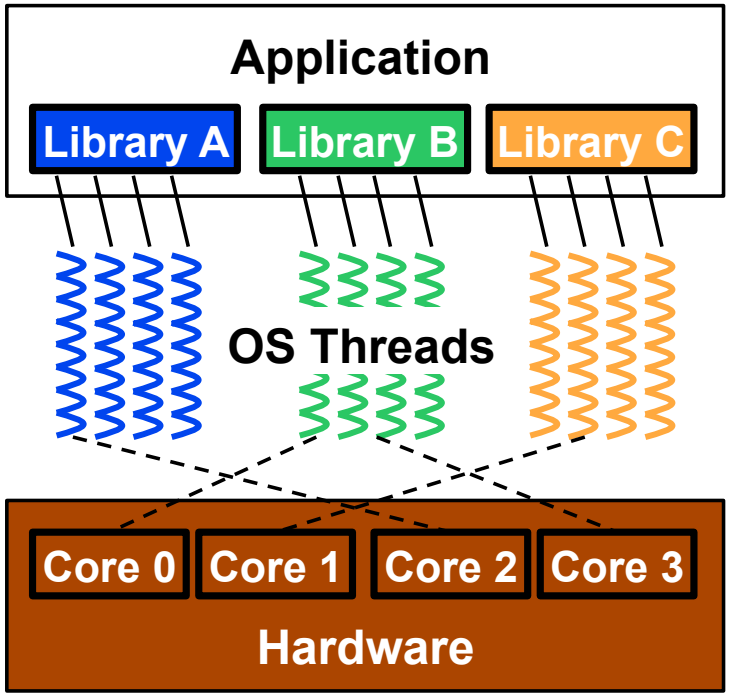
# Exports Problem to User



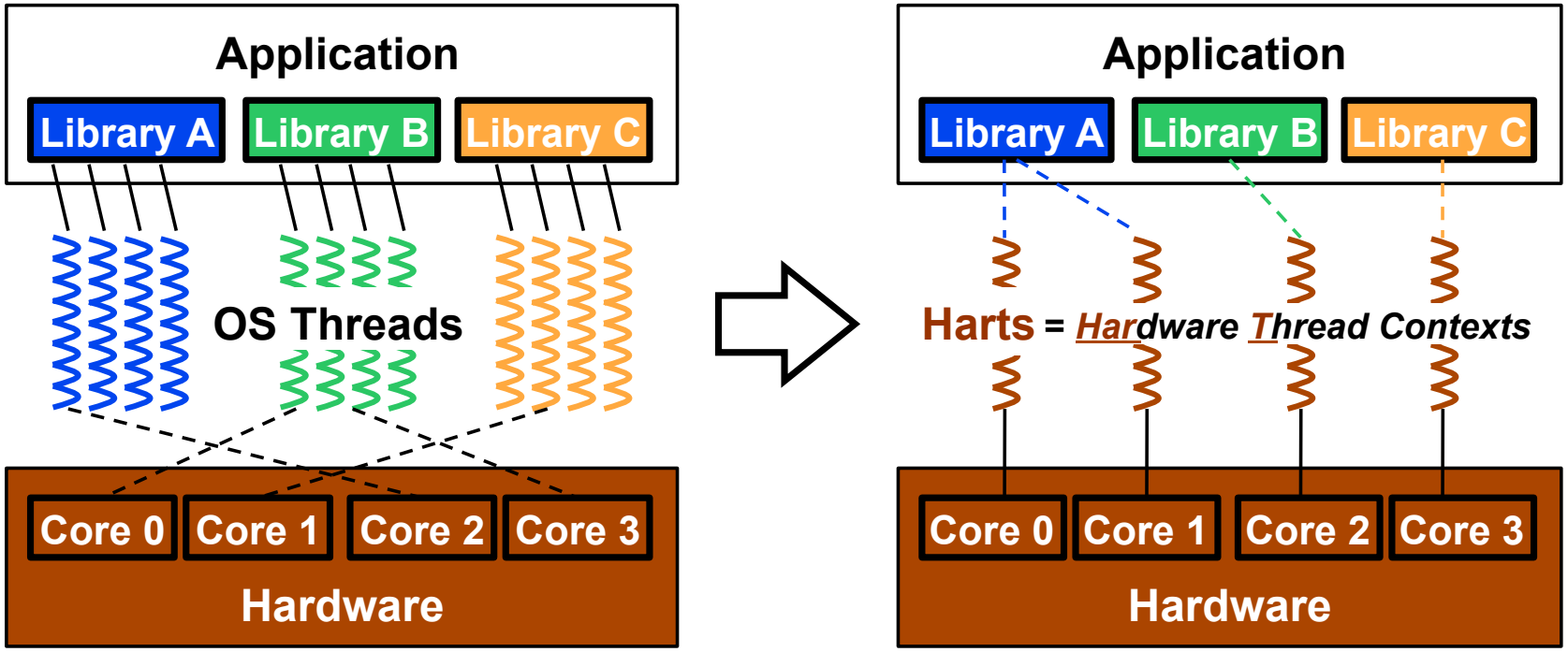
# Exports Problem to User



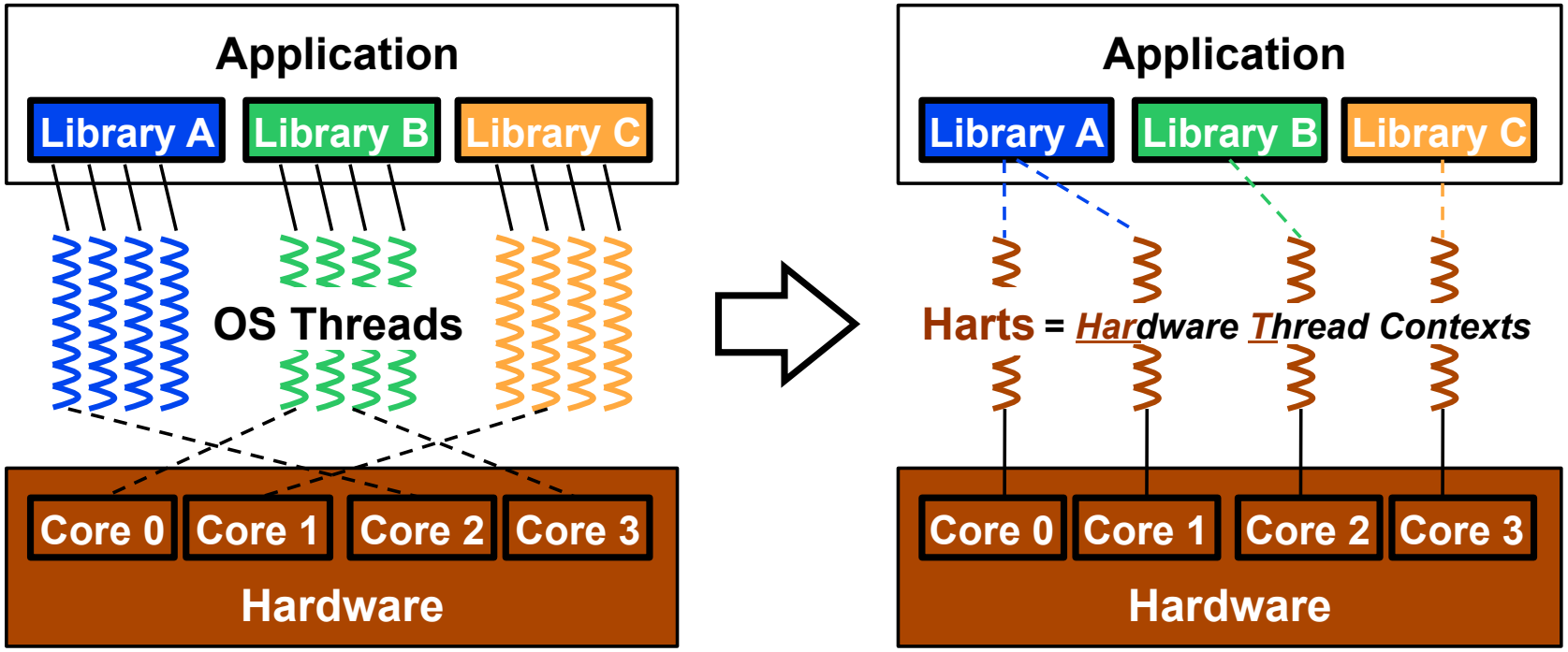
# Better Resource Abstraction: HARTS



# Better Resource Abstraction: HARTS

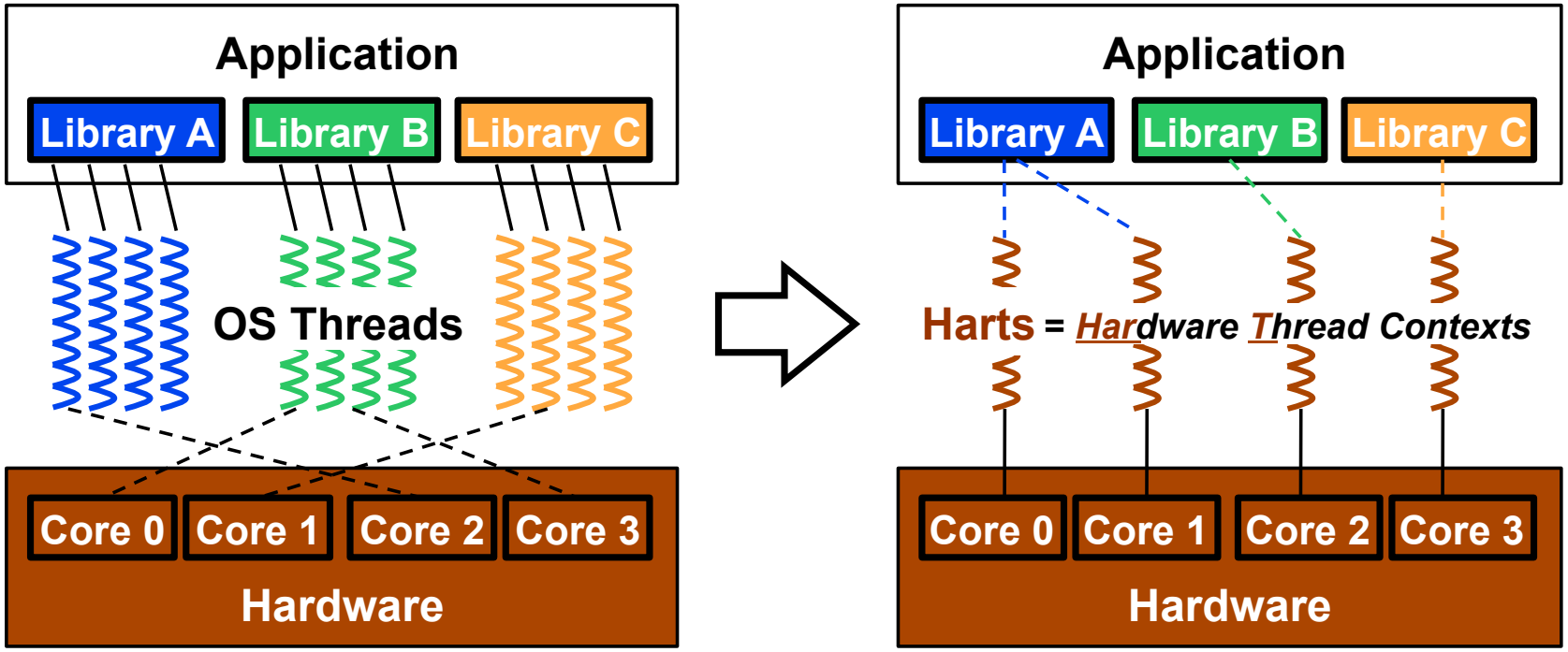


# Better Resource Abstraction: HARTS




- Create as many threads as wanted.
- Allocated a finite amount of harts.

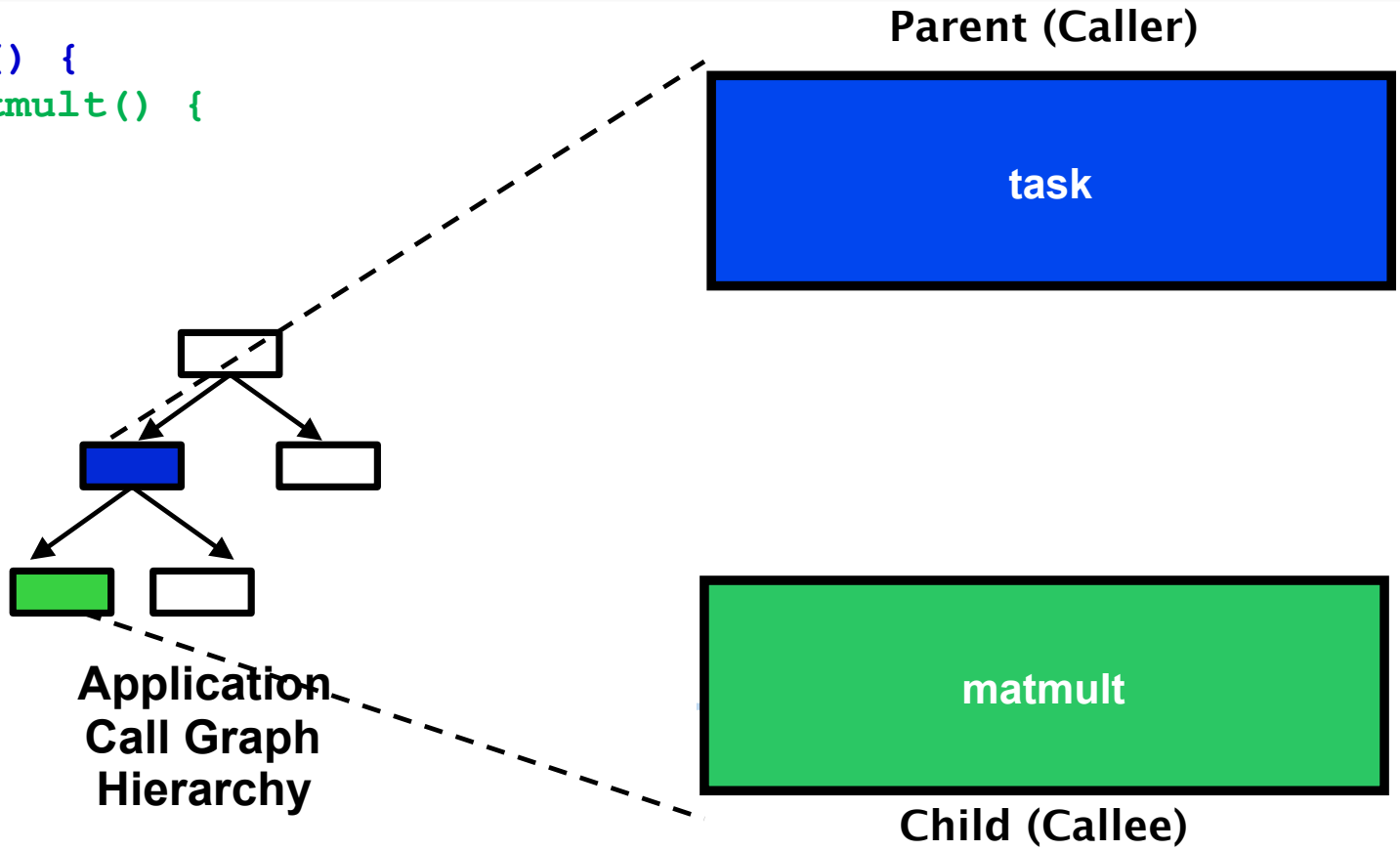
# Better Resource Abstraction: HARTS




- Create as many threads as wanted.
- Threads = Resource + Programming Abstraction
- Allocated a finite amount of harts.
- Harts = Resource Abstraction

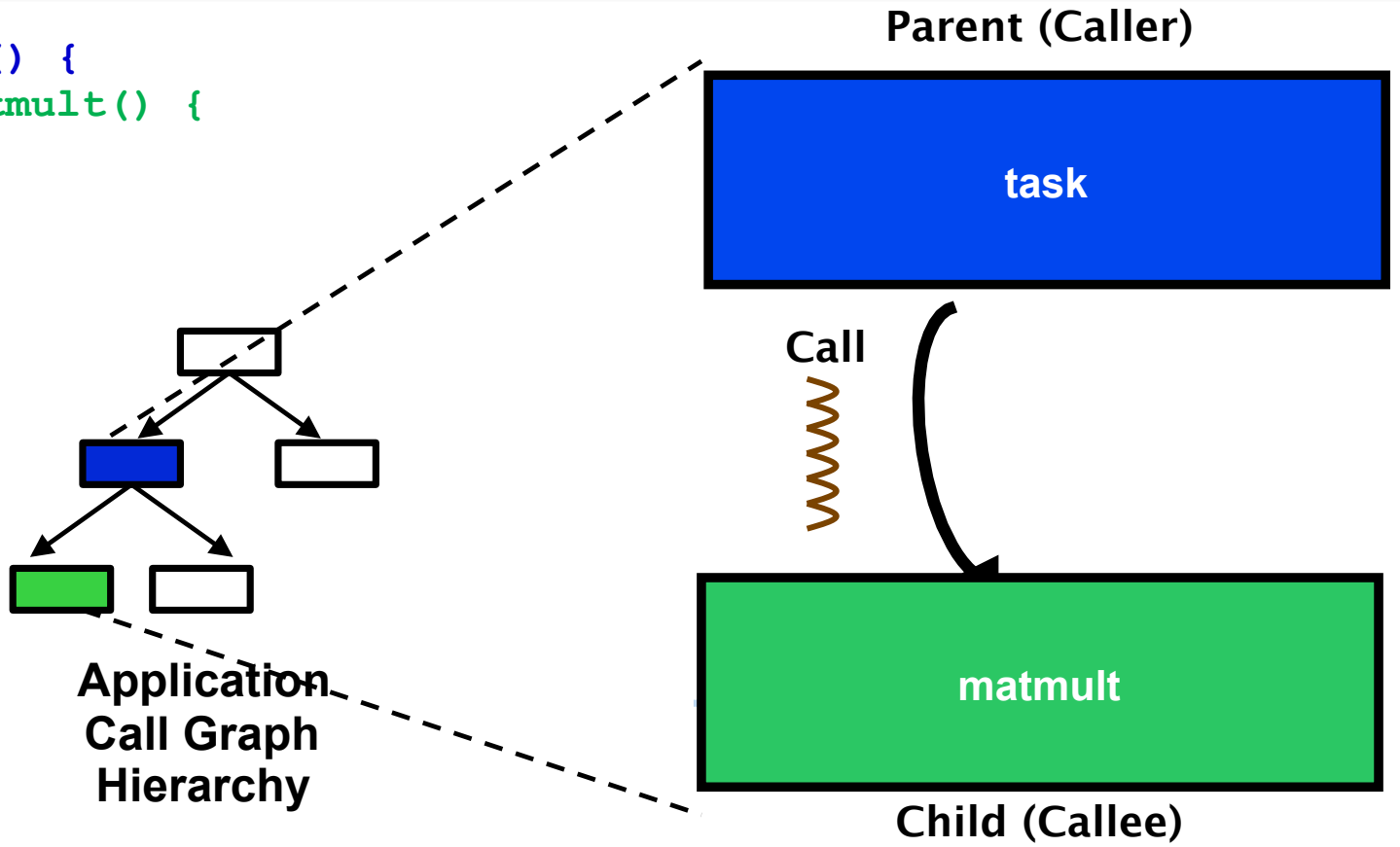
# Cooperative Hierarchical Resource Scheduling

```
task() {
  matmult() {
    :
  }
  :
}
```



# Cooperative Hierarchical Resource Scheduling

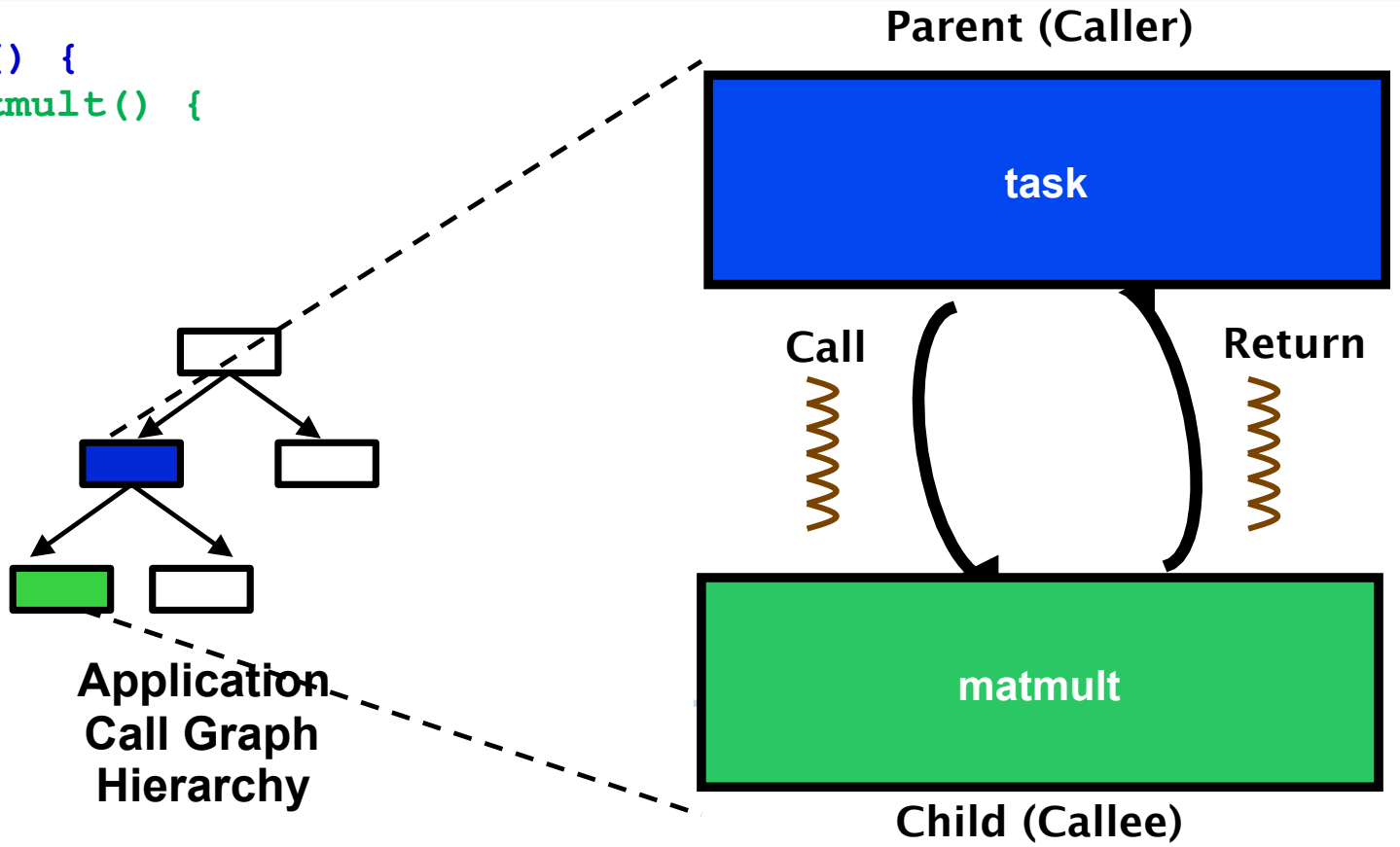
```
task() {
  matmult() {
    :
  }
  :
}
```




**Transfer of control coupled with transfer of resources.**

# Cooperative Hierarchical Resource Scheduling

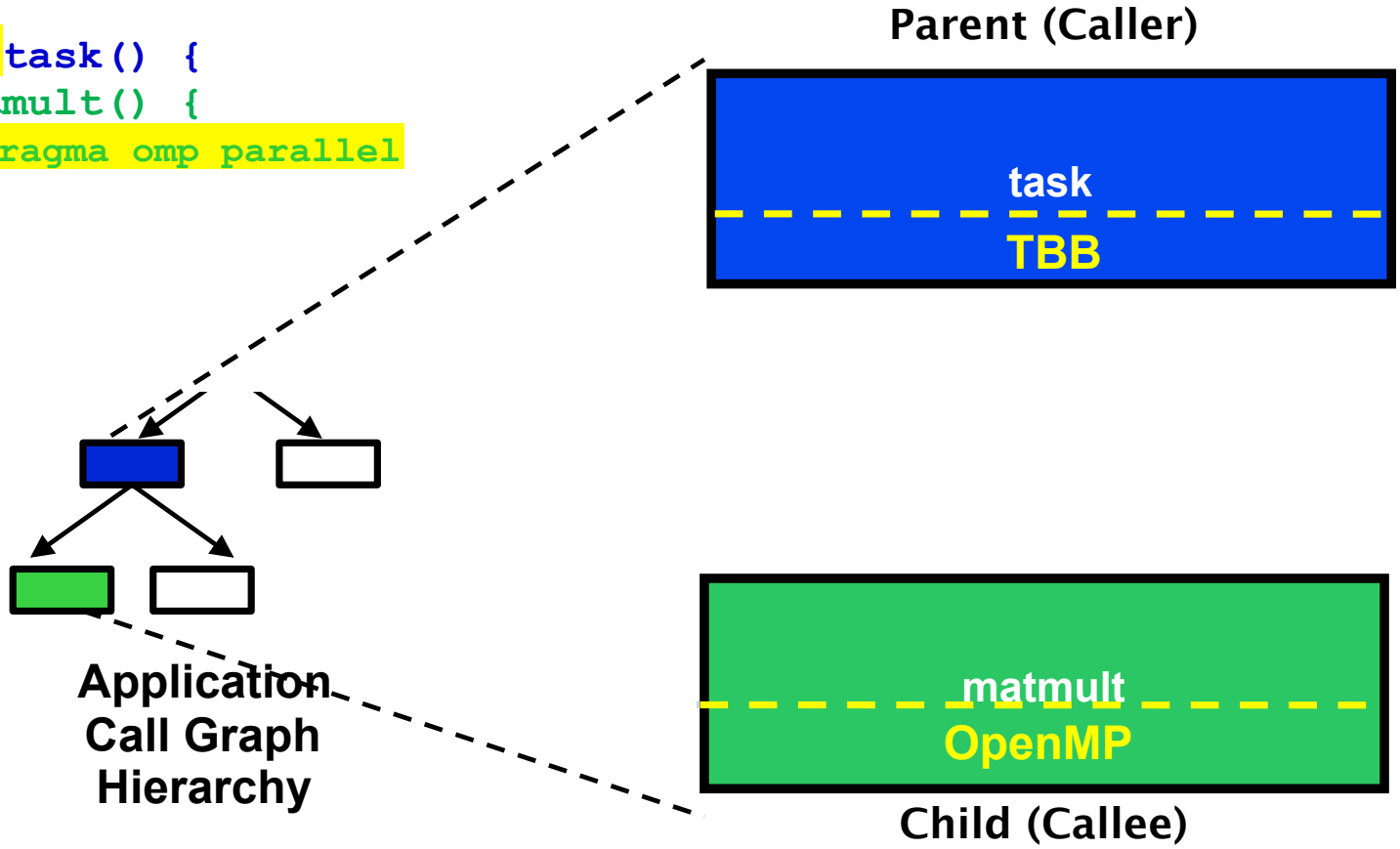
```
task() {
  matmult() {
    :
  }
  :
}
```




**Transfer of control coupled with transfer of resources.**

# Cooperative Hierarchical Resource Scheduling

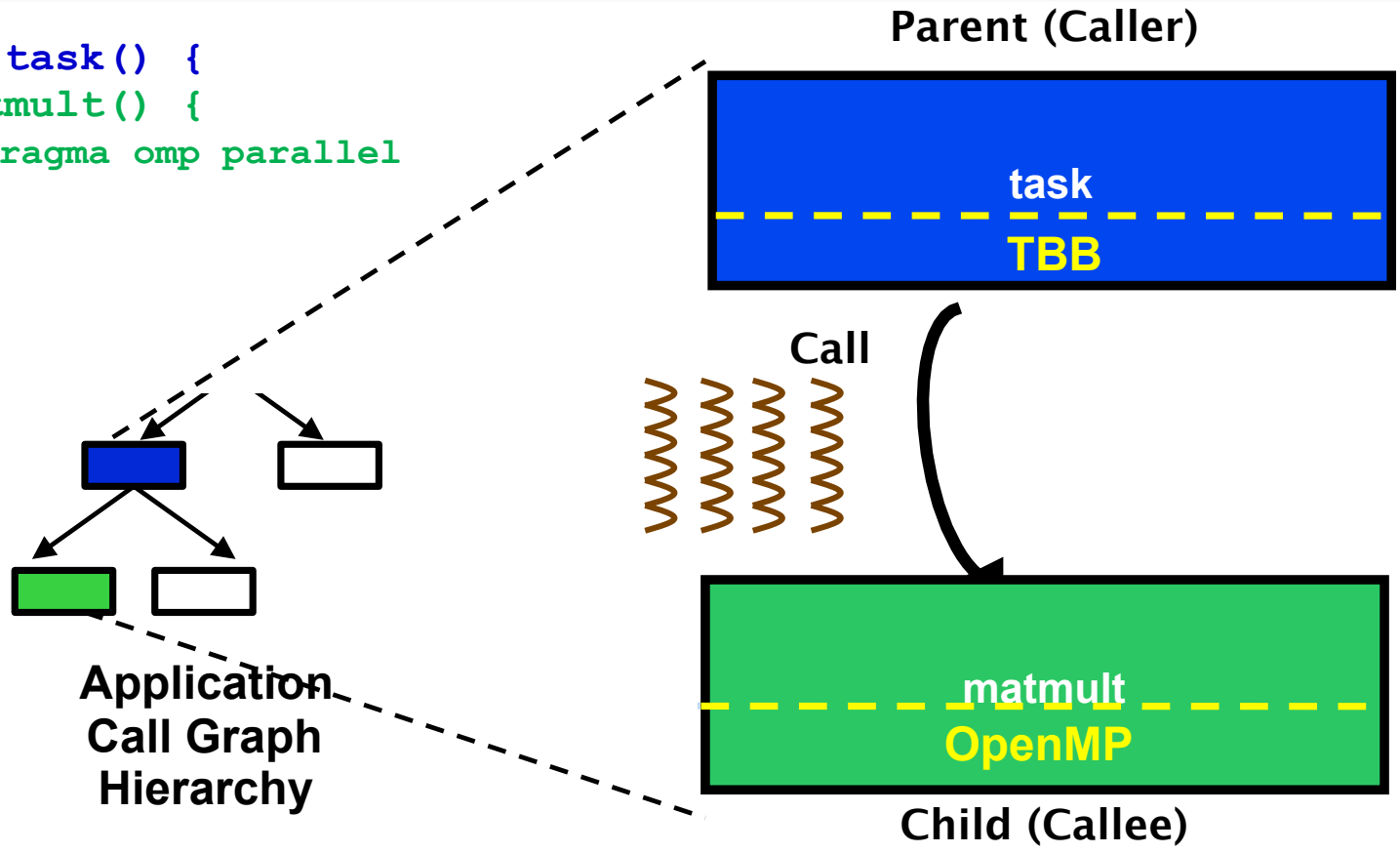
```
tbb::task() {
  matmult() {
    #pragma omp parallel
    :
  }
  :
}
```




**Transfer of control coupled with transfer of resources.**

# Cooperative Hierarchical Resource Scheduling

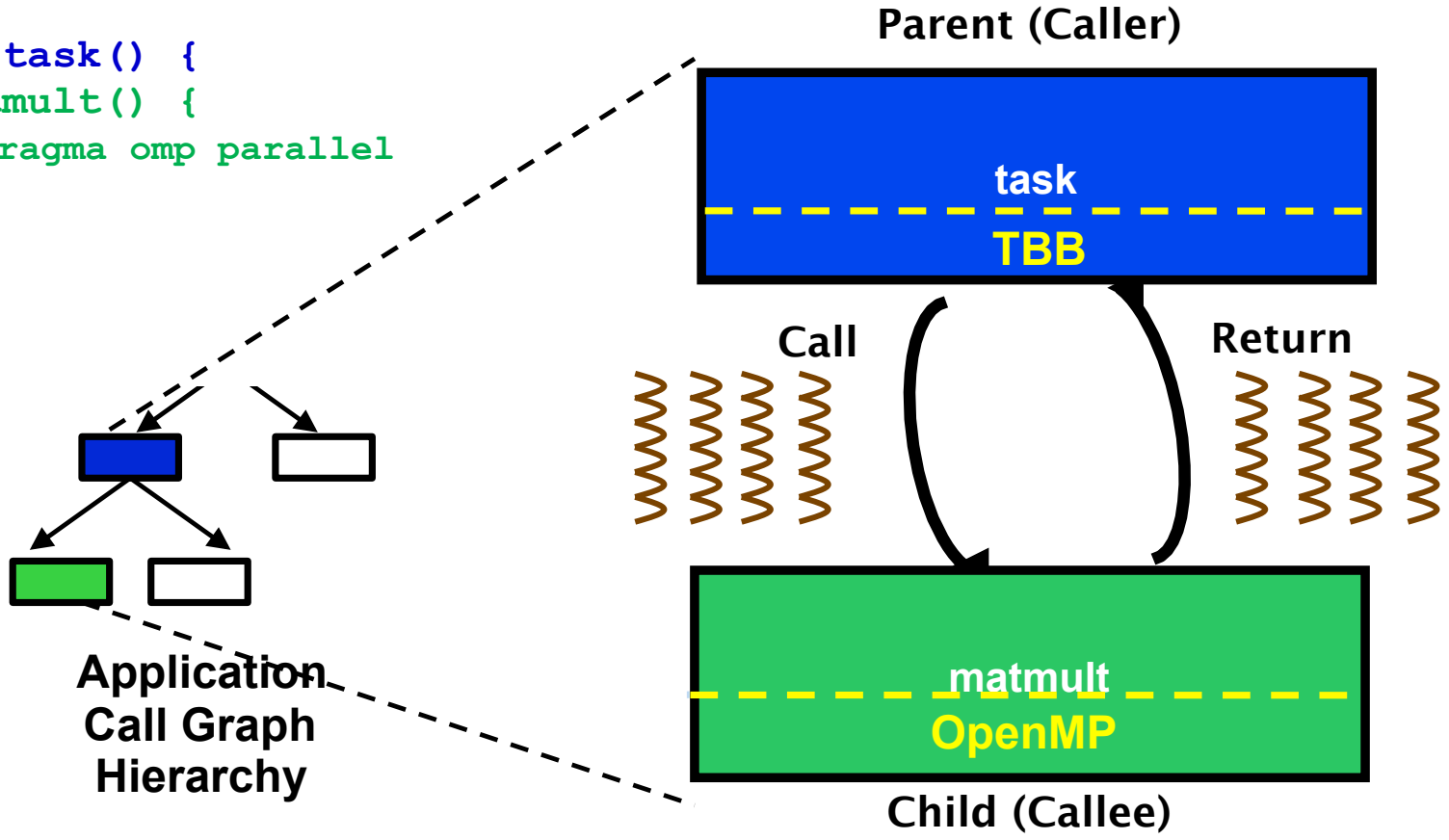
```
tbb::task() {
  matmult() {
   #pragma omp parallel
   :
  }
  :
}
```




**Transfer of control coupled with transfer of resources.**

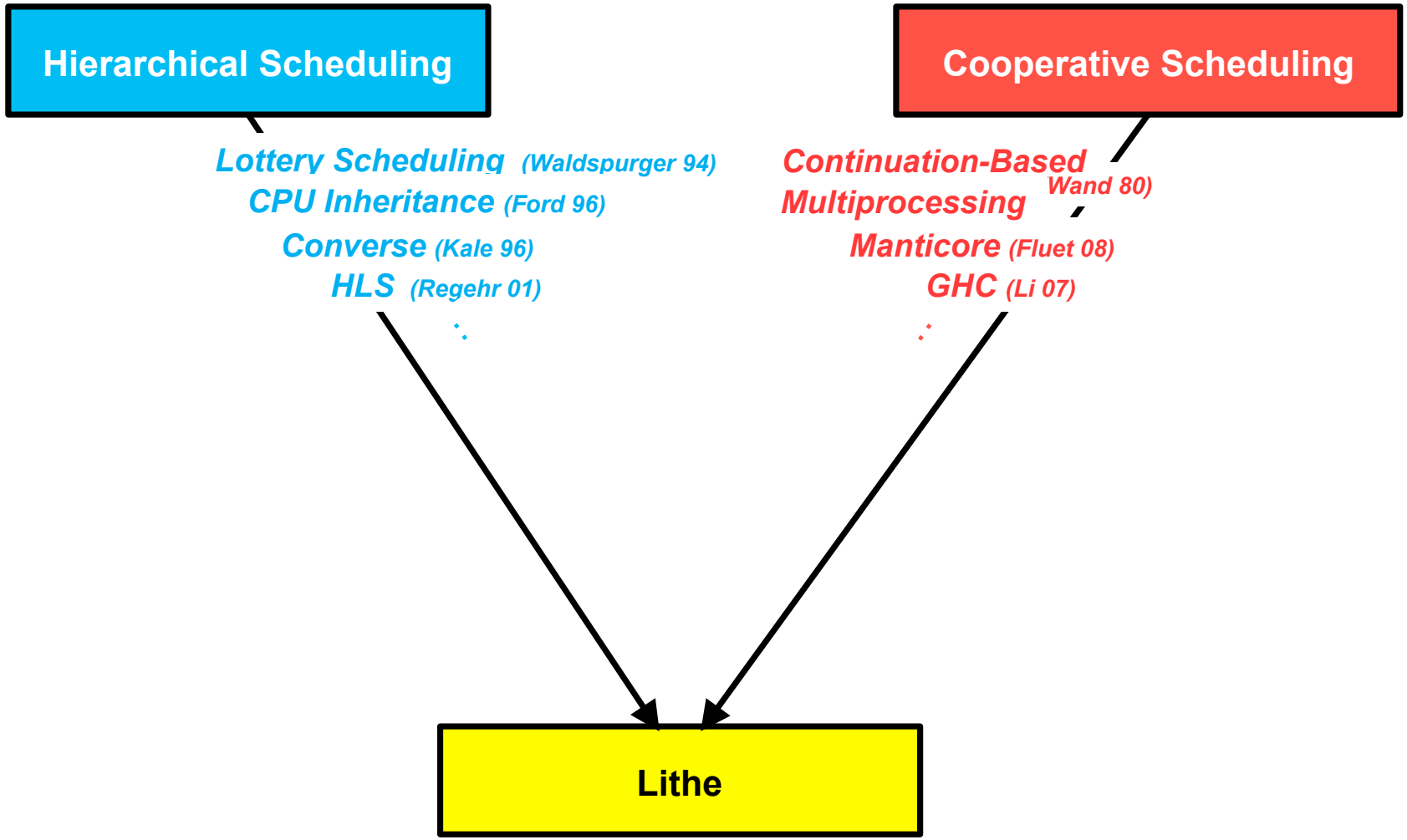
# Cooperative Hierarchical Resource Scheduling

```
tbb::task() {
  matmult() {
   #pragma omp parallel
   :
  }
  :
}
```

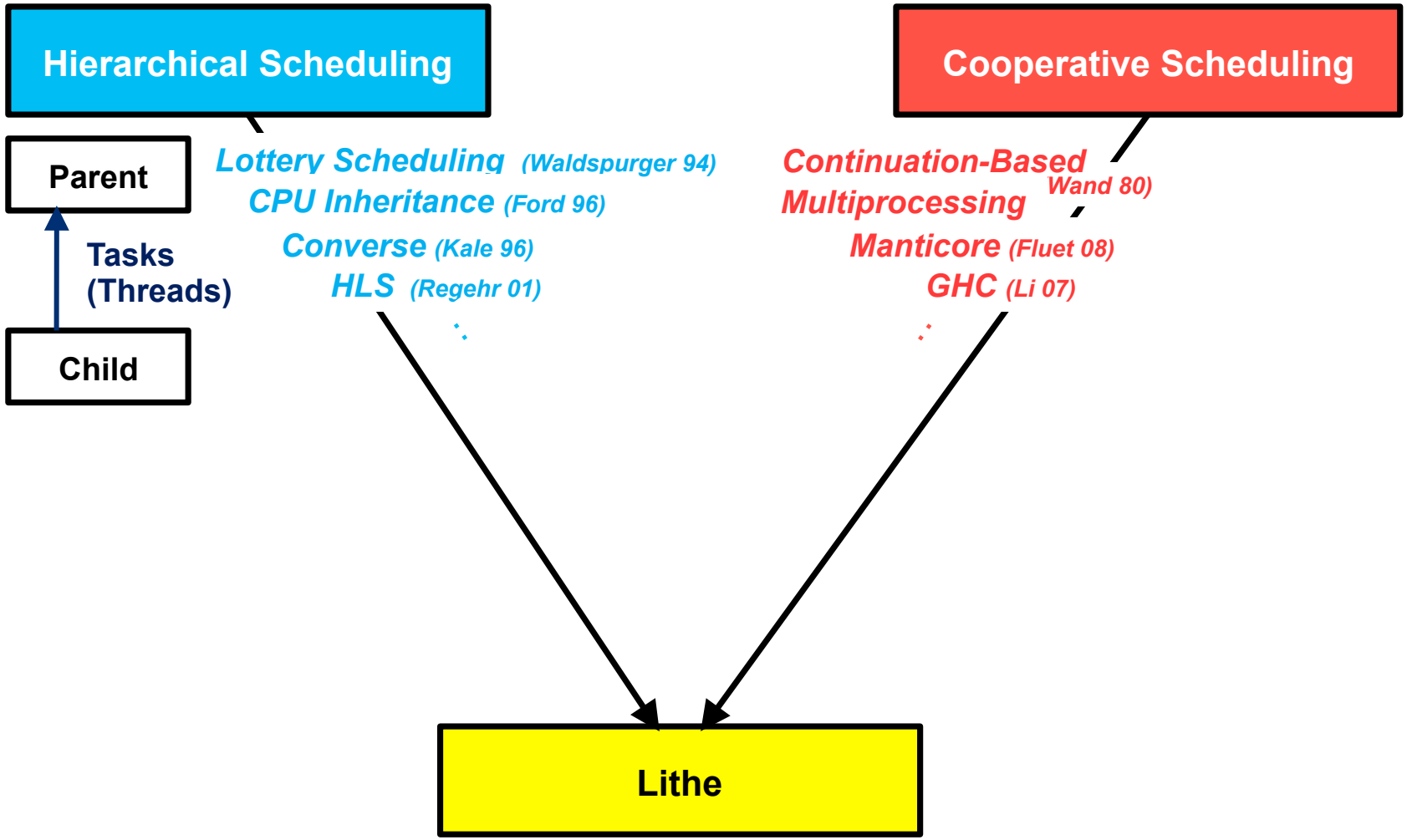



**Transfer of control coupled with transfer of resources.**

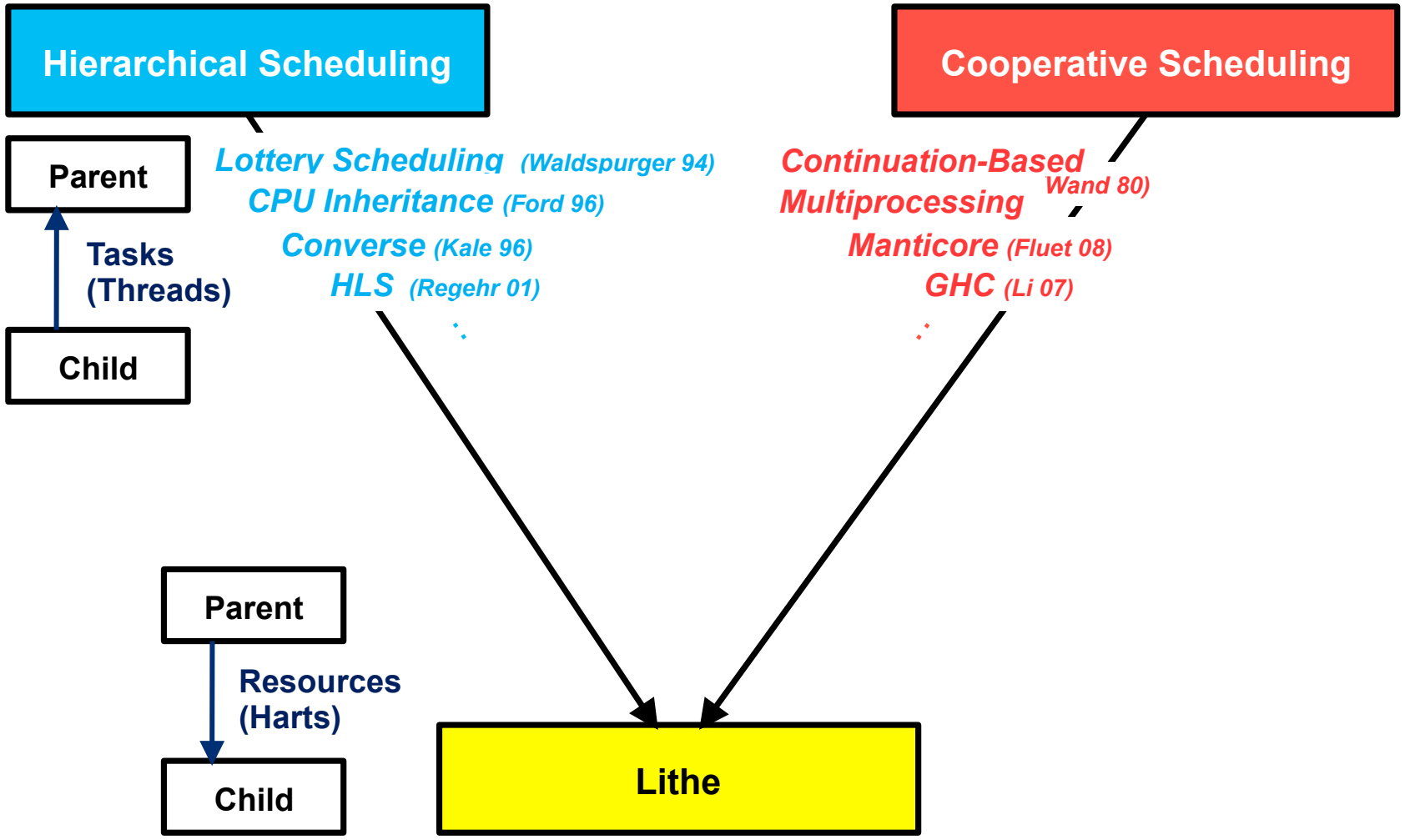
# Confluence of Related Work



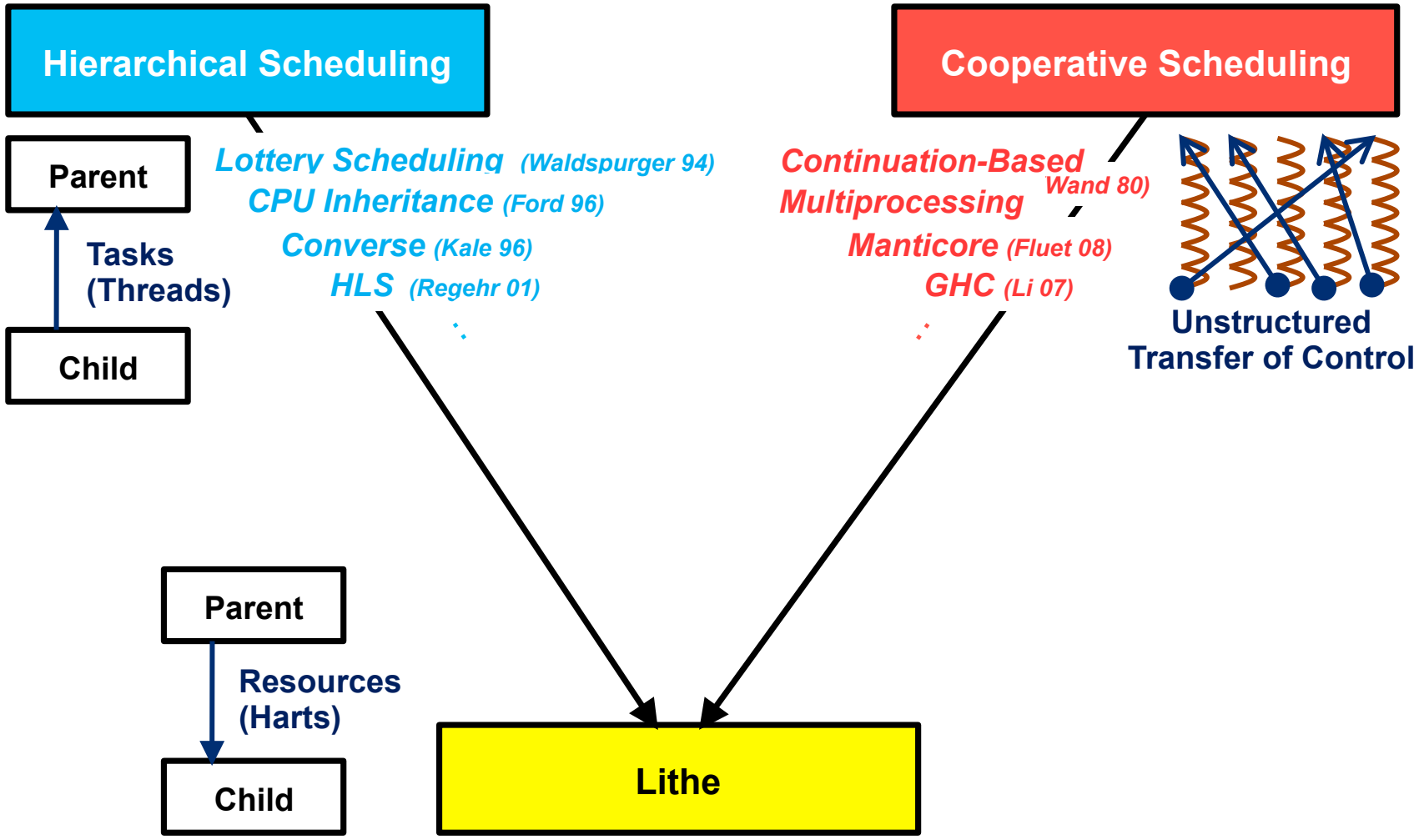
# Confluence of Related Work



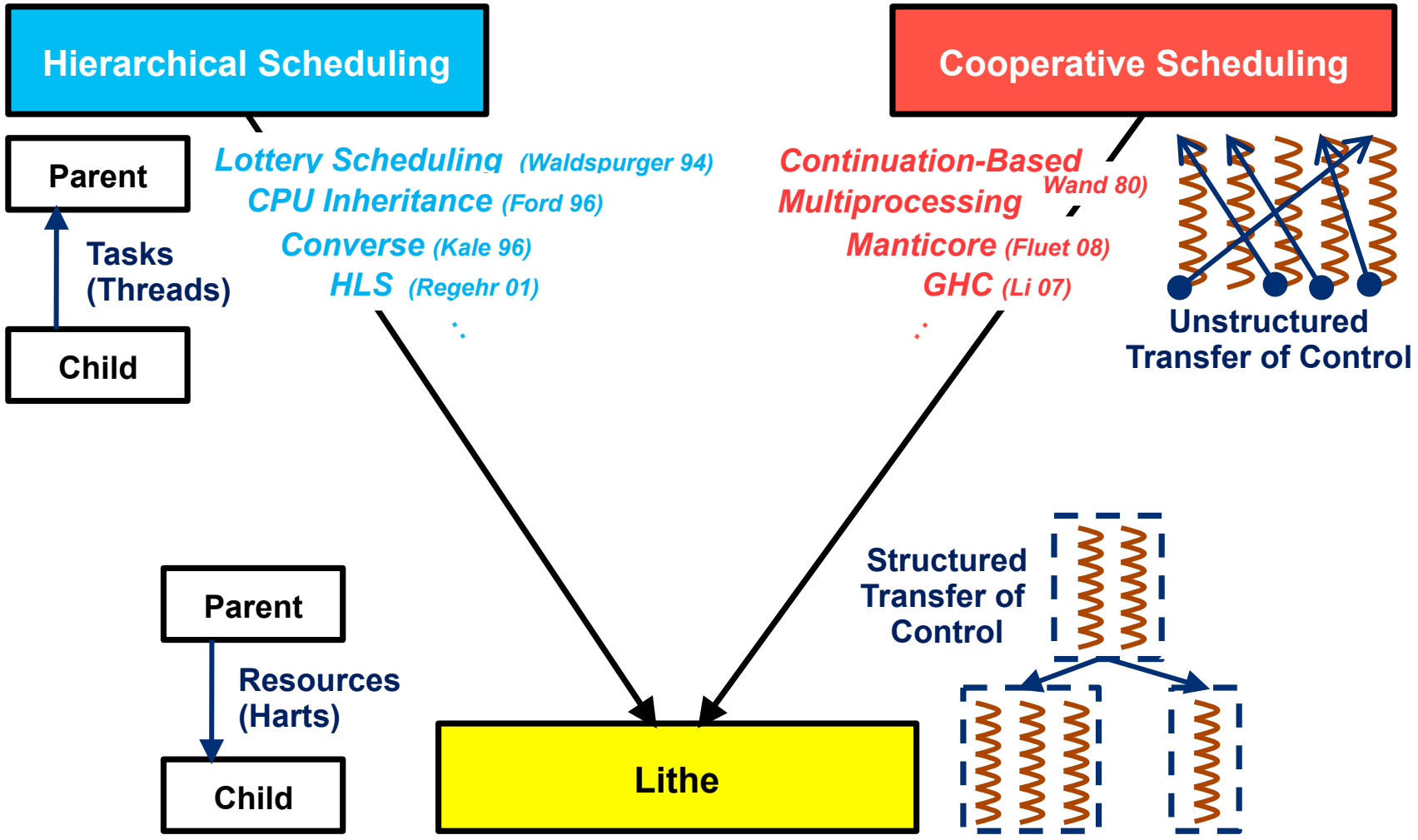
# Confluence of Related Work



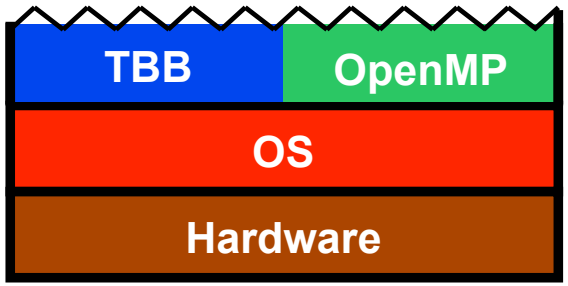
# Confluence of Related Work



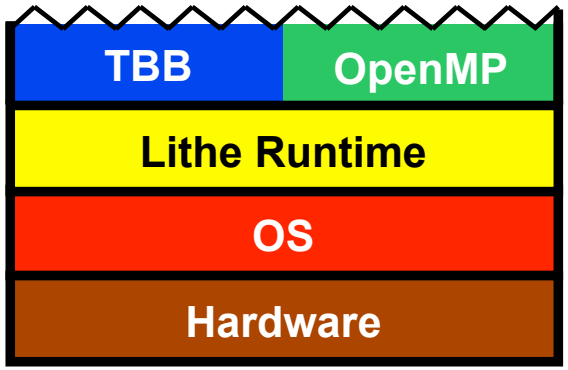
# Confluence of Related Work



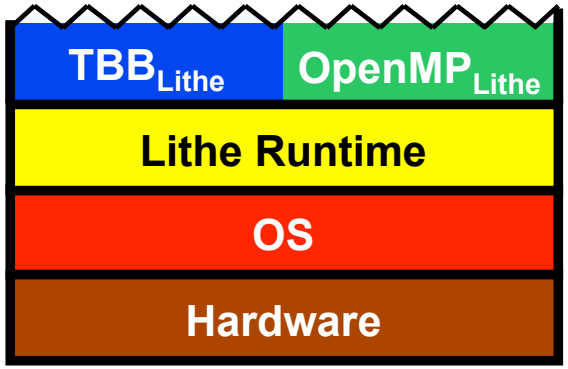
# The Lithe Runtime



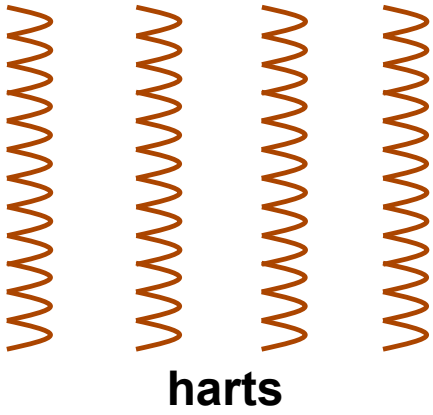
# The Lithe Runtime

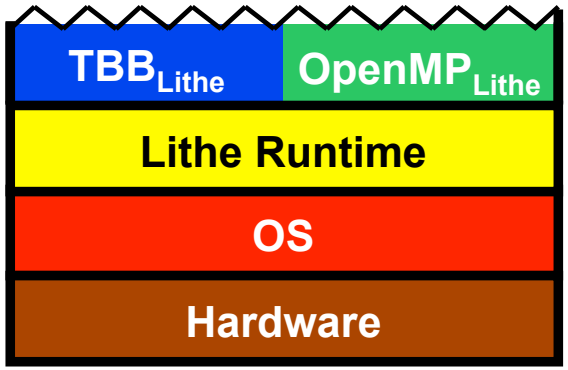


# The Lithe Runtime

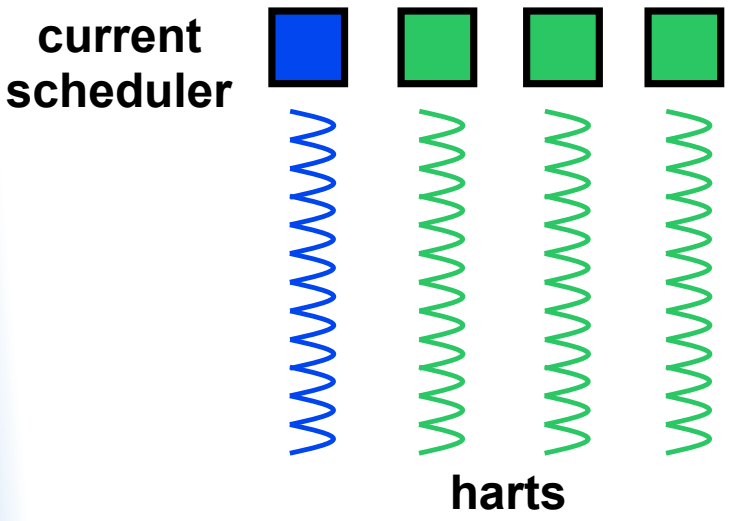


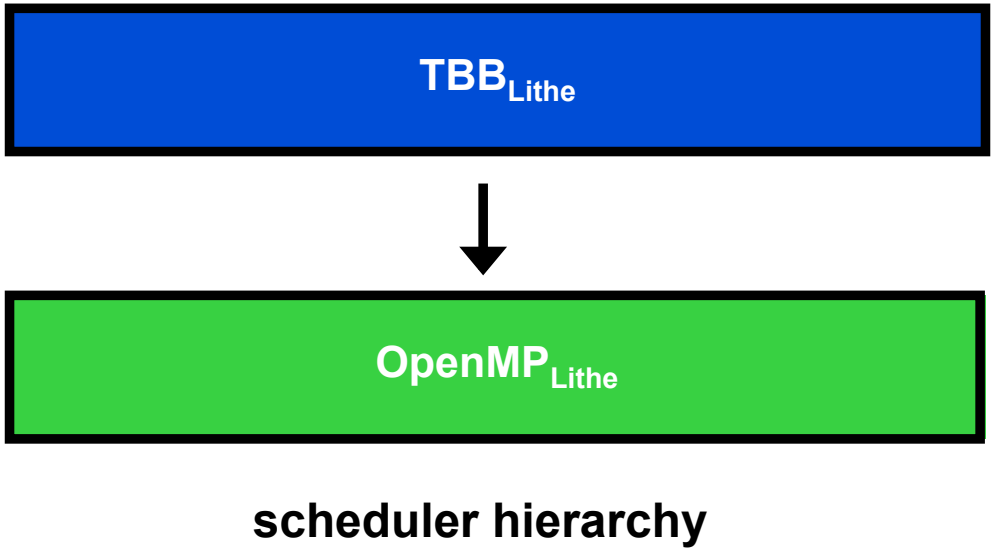
# The Lithe Runtime

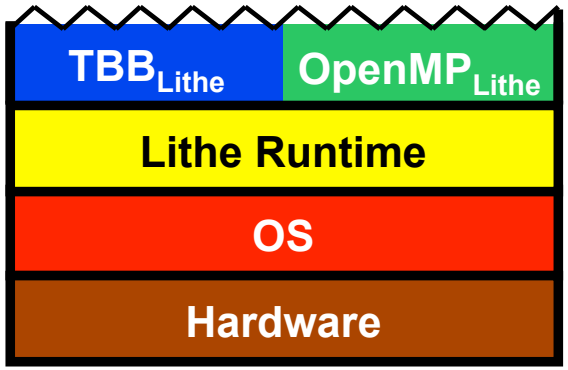


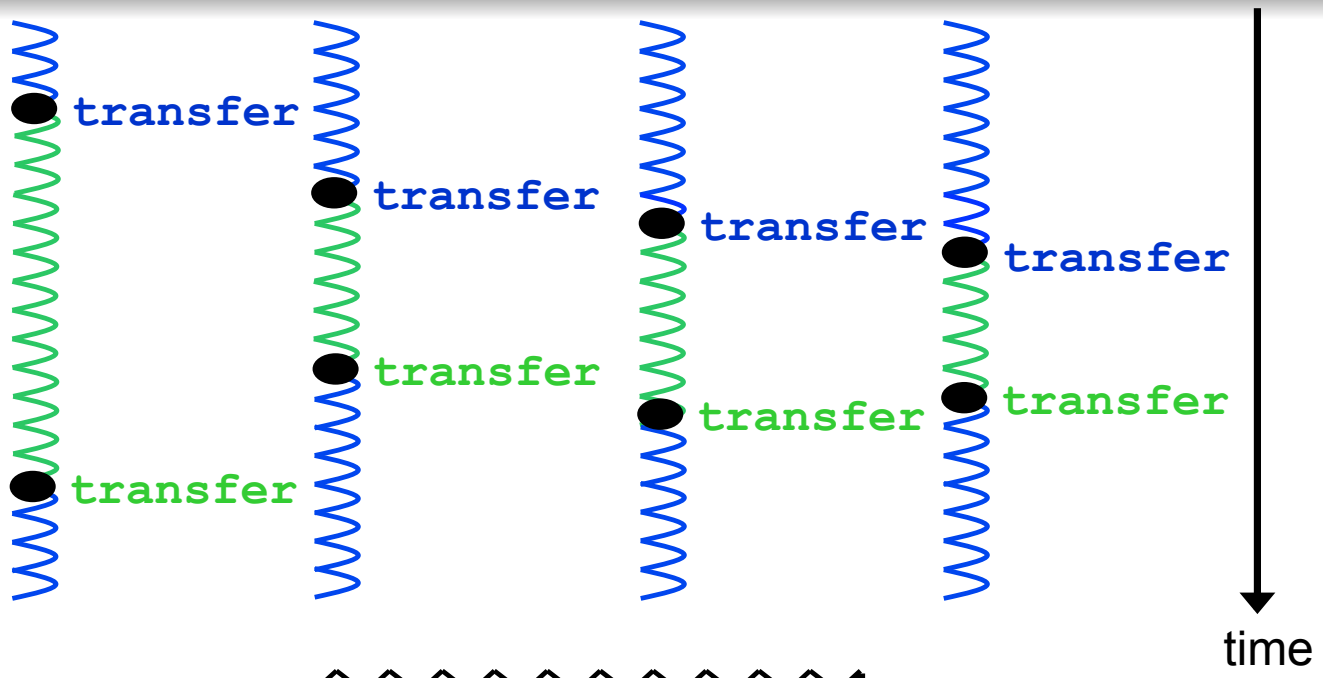
# The Lithe Runtime

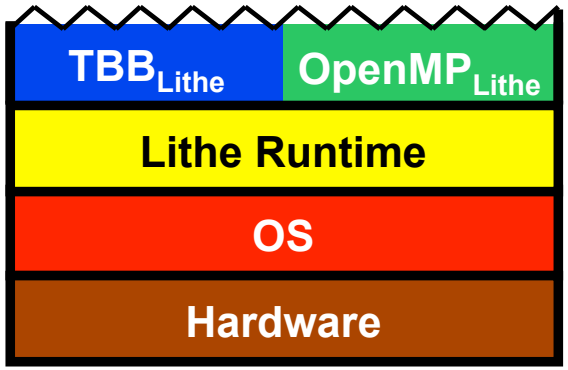


# The Lithe Runtime



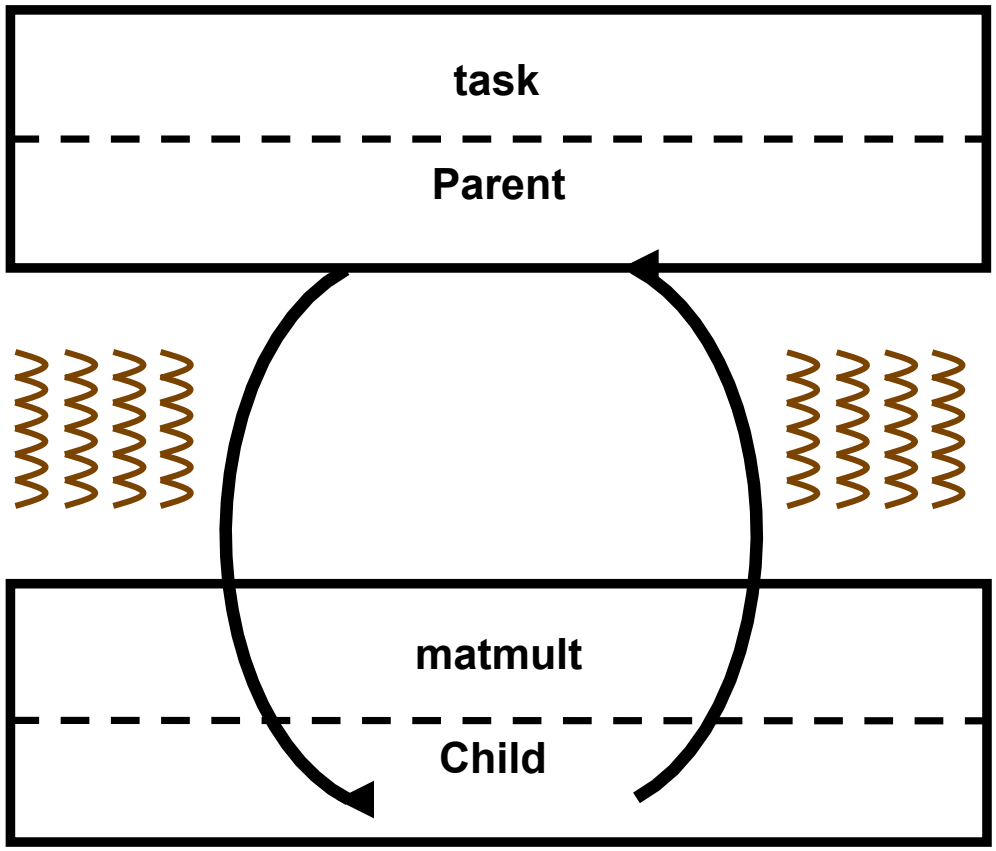






48

# Standard API/Callback Interface

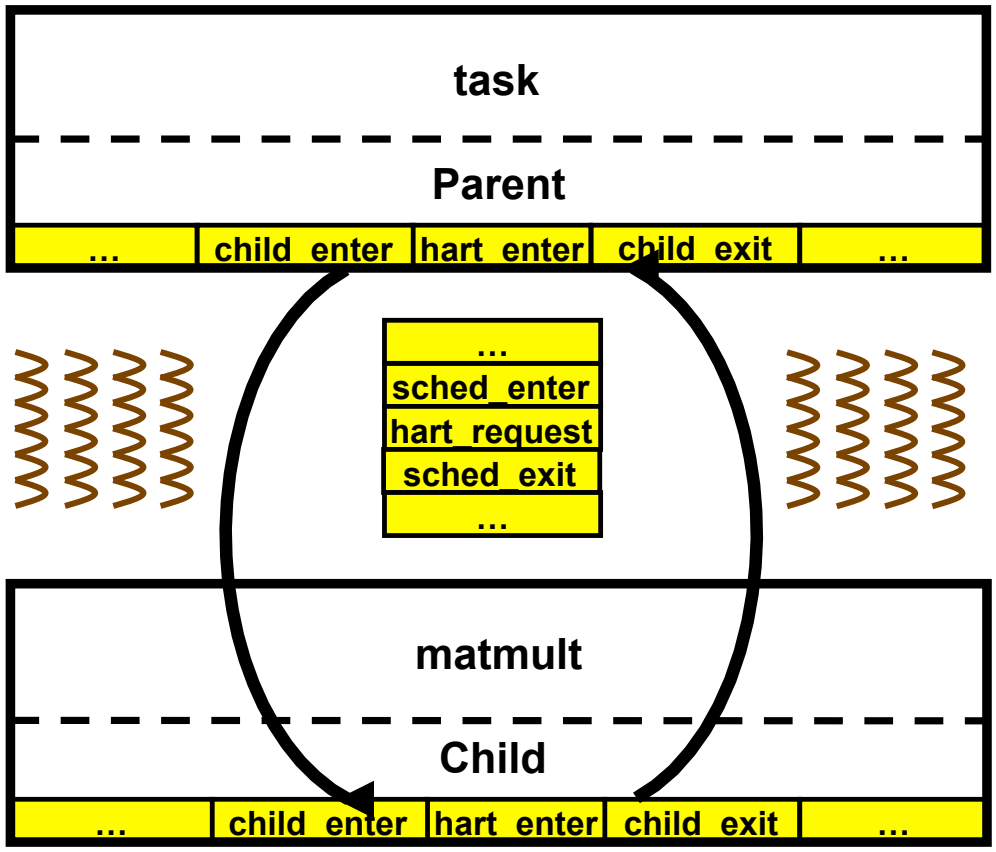
```
task() {
  matmult() {

    :
  }
  :
}
```




**Separation of Interface and Implementation**

# Standard API/Callback Interface

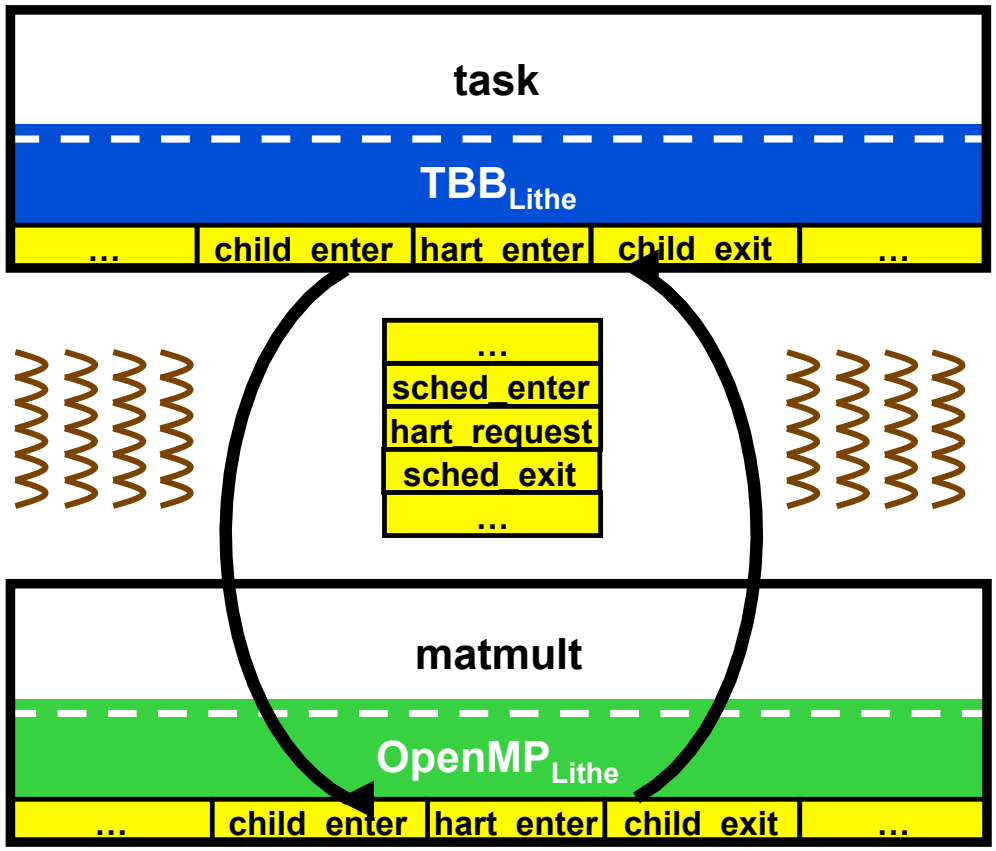
```
task() {
  matmult() {

    :
  }
  :
}
```




**Separation of Interface and Implementation**

# Standard API/Callback Interface

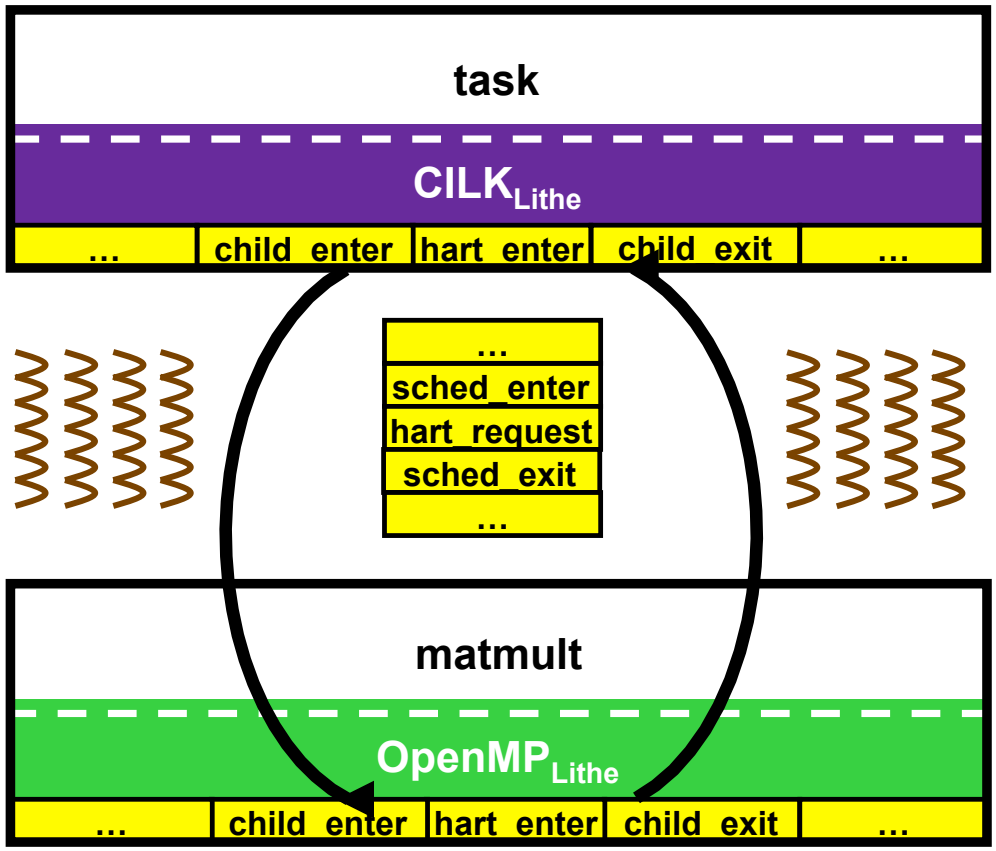
```
tbb::
task() {
  matmult() {
    #pragma OMP parallel
    :
  }
  :
}
```




**Separation of Interface and Implementation**

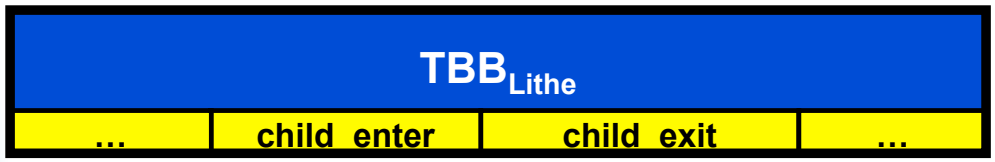
# Standard API/Callback Interface

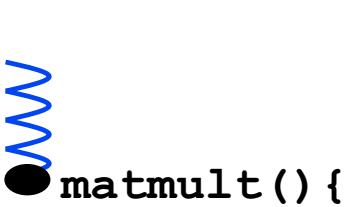
```
cilk
task() {
  matmult() {
    #pragma OMP parallel
    :
  }
  :
}
```



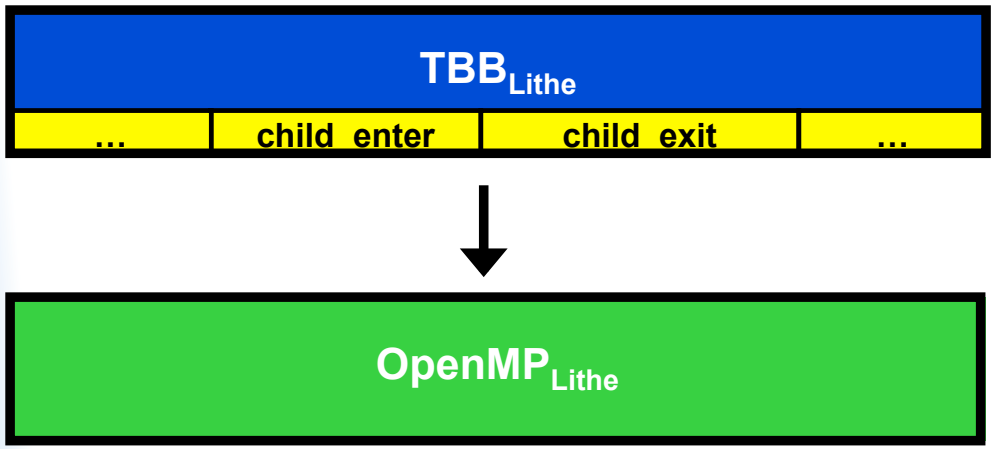

**Separation of Interface and Implementation**

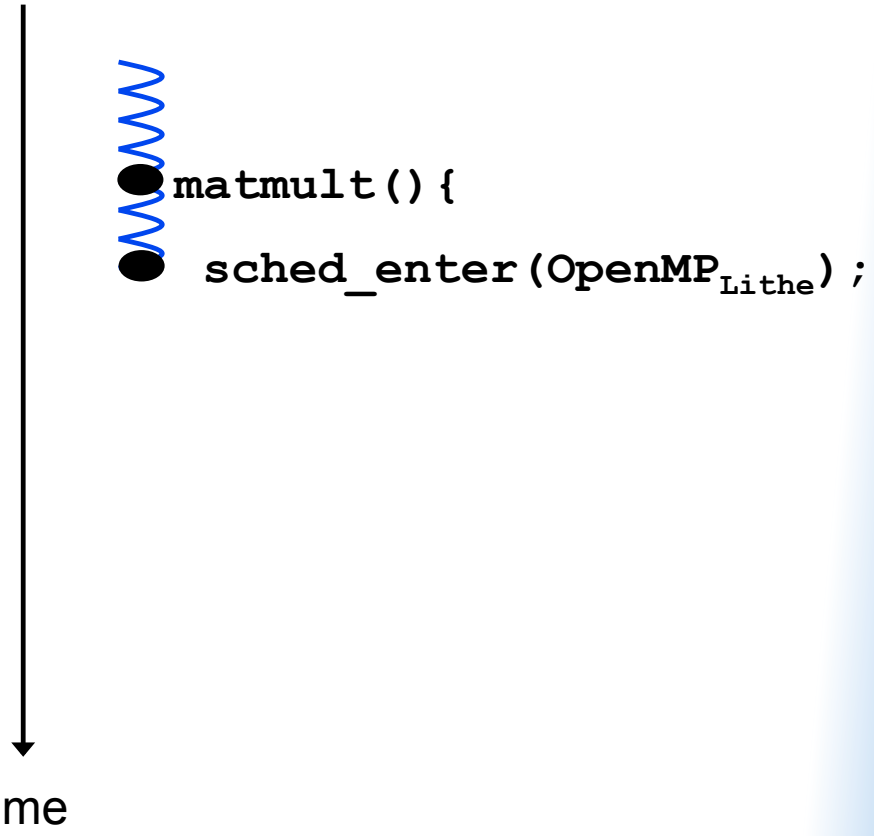
# Enter/Exit a Scheduler



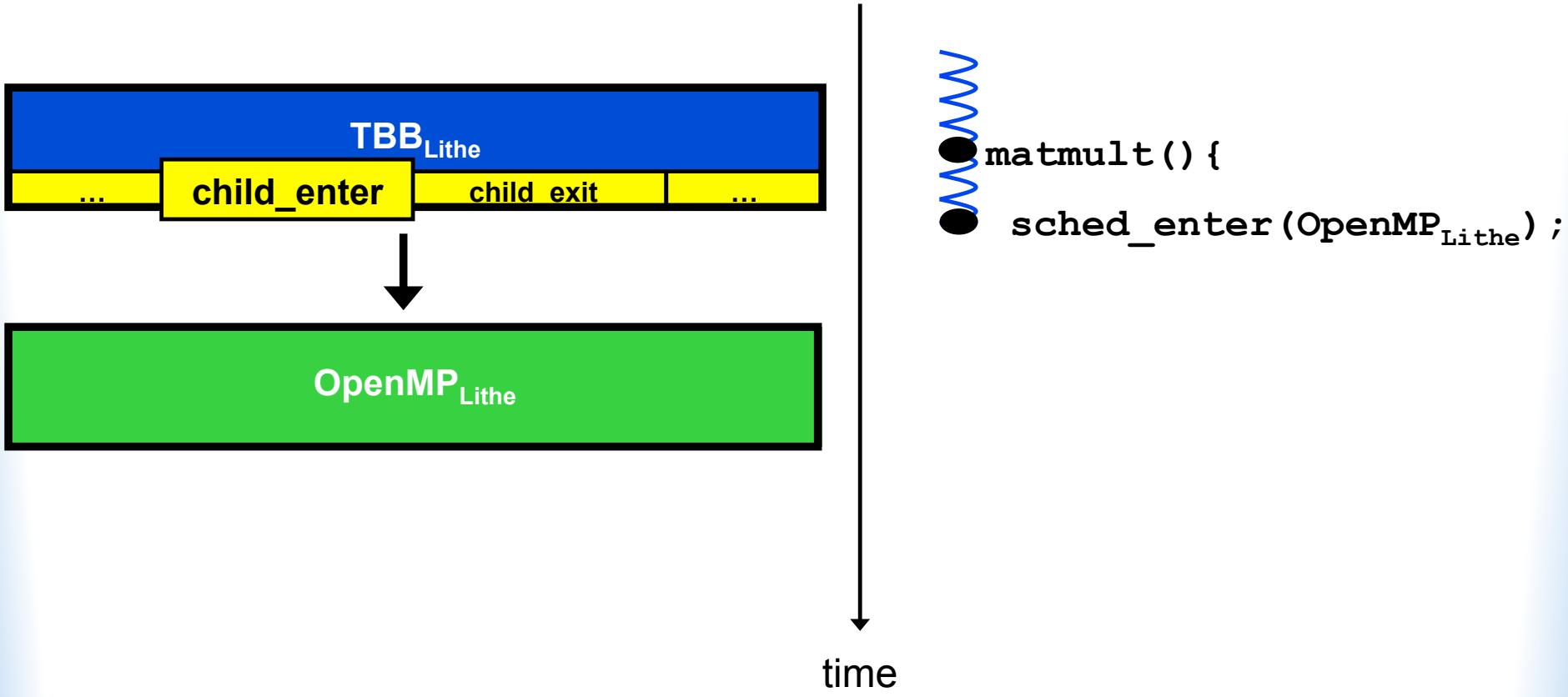




# Enter/Exit a Scheduler

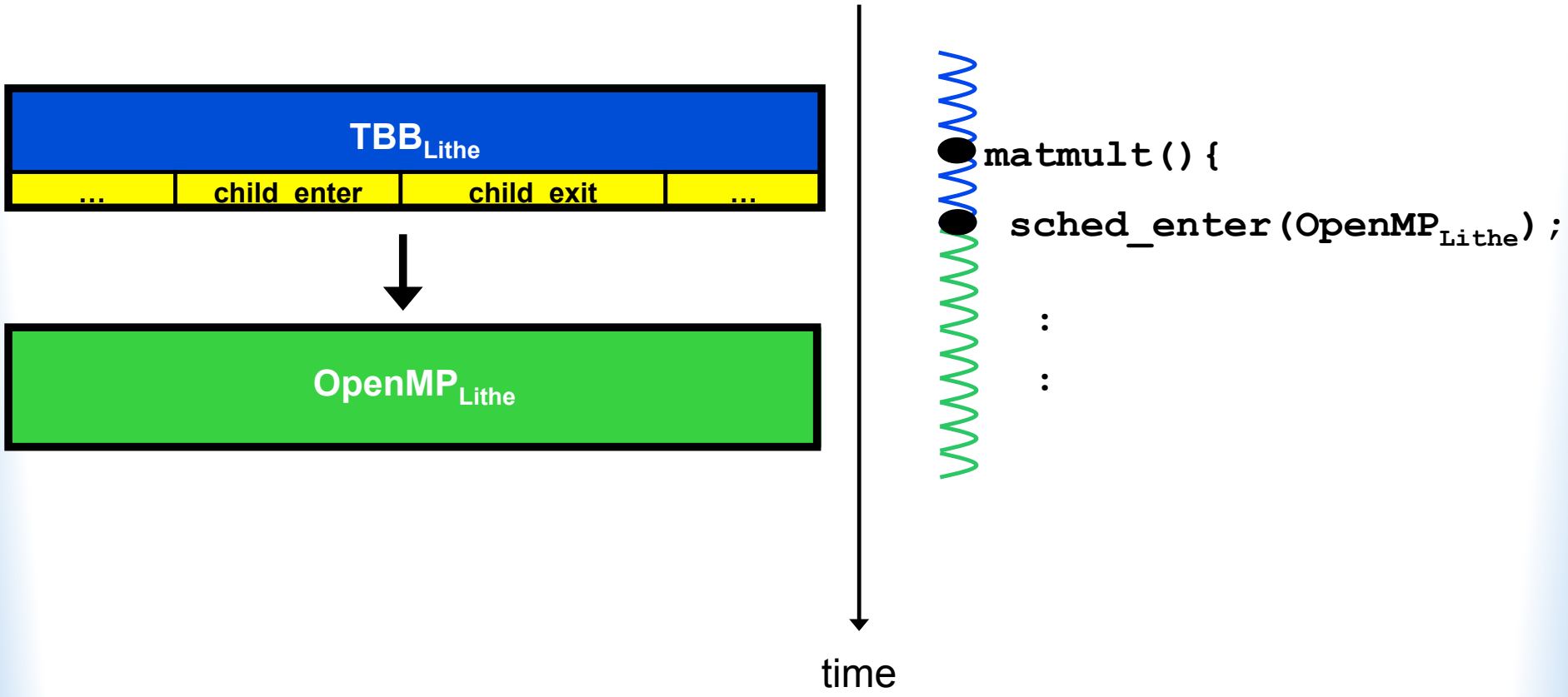


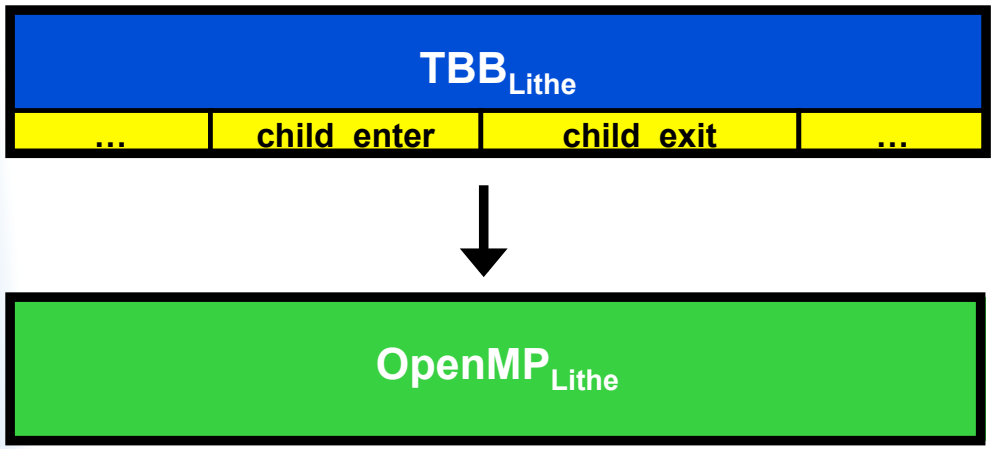
# Enter/Exit a Scheduler

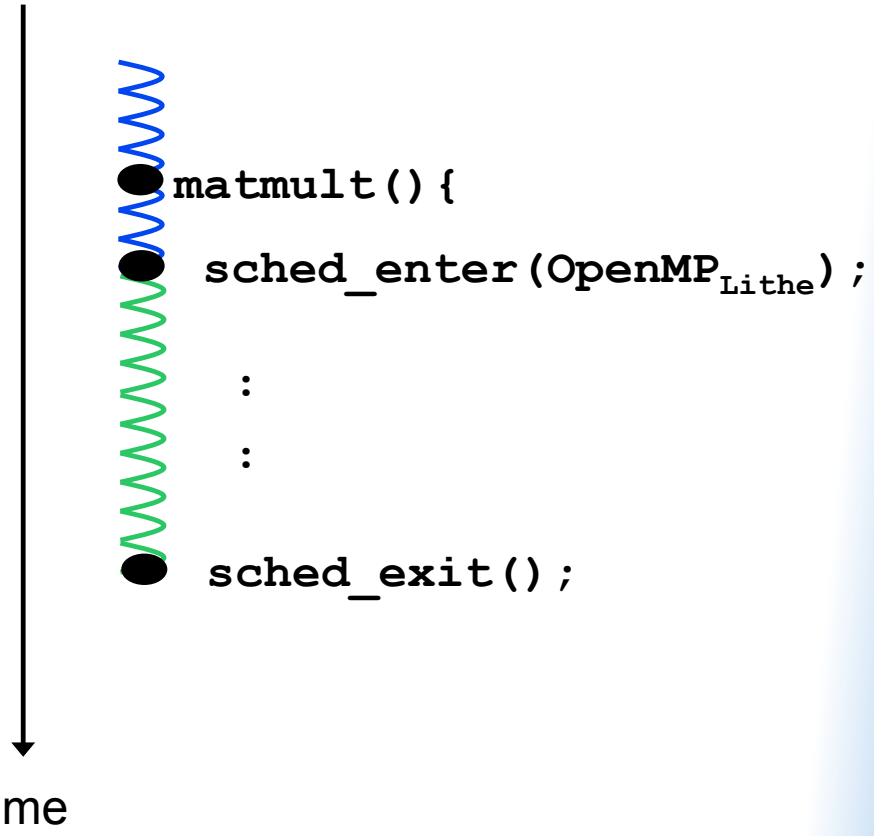


# Enter/Exit a Scheduler

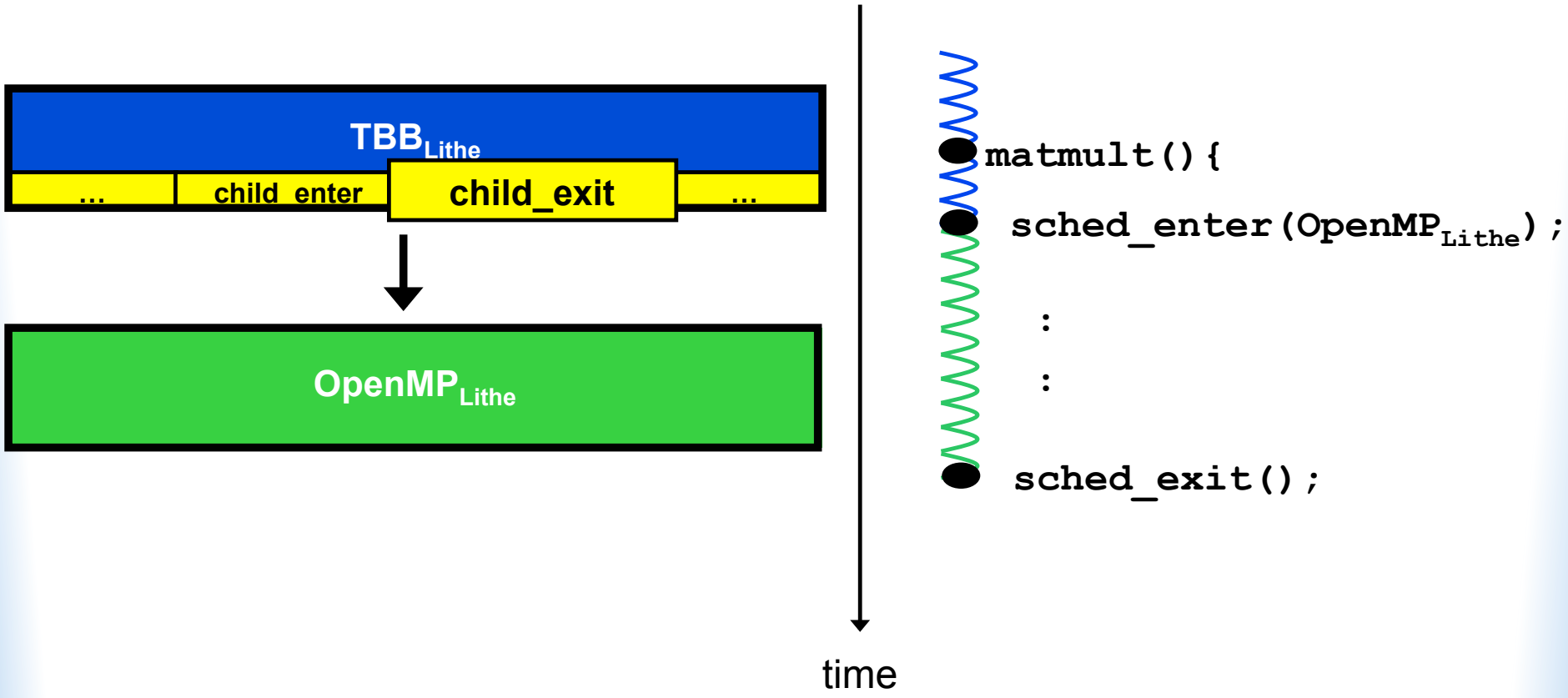


# Enter/Exit a Scheduler

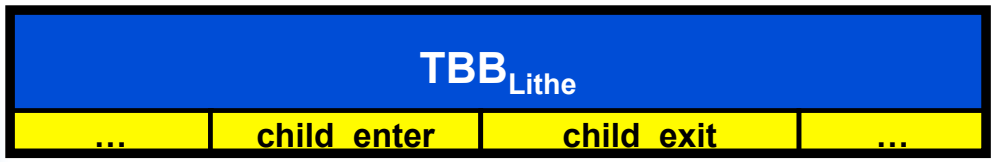


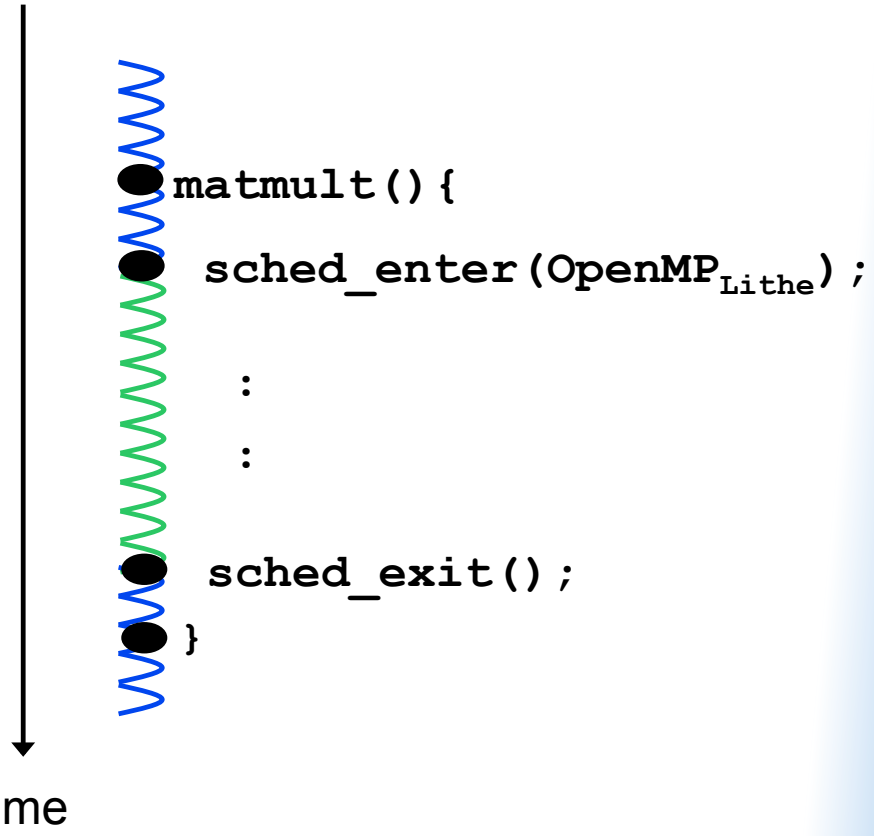



# Enter/Exit a Scheduler

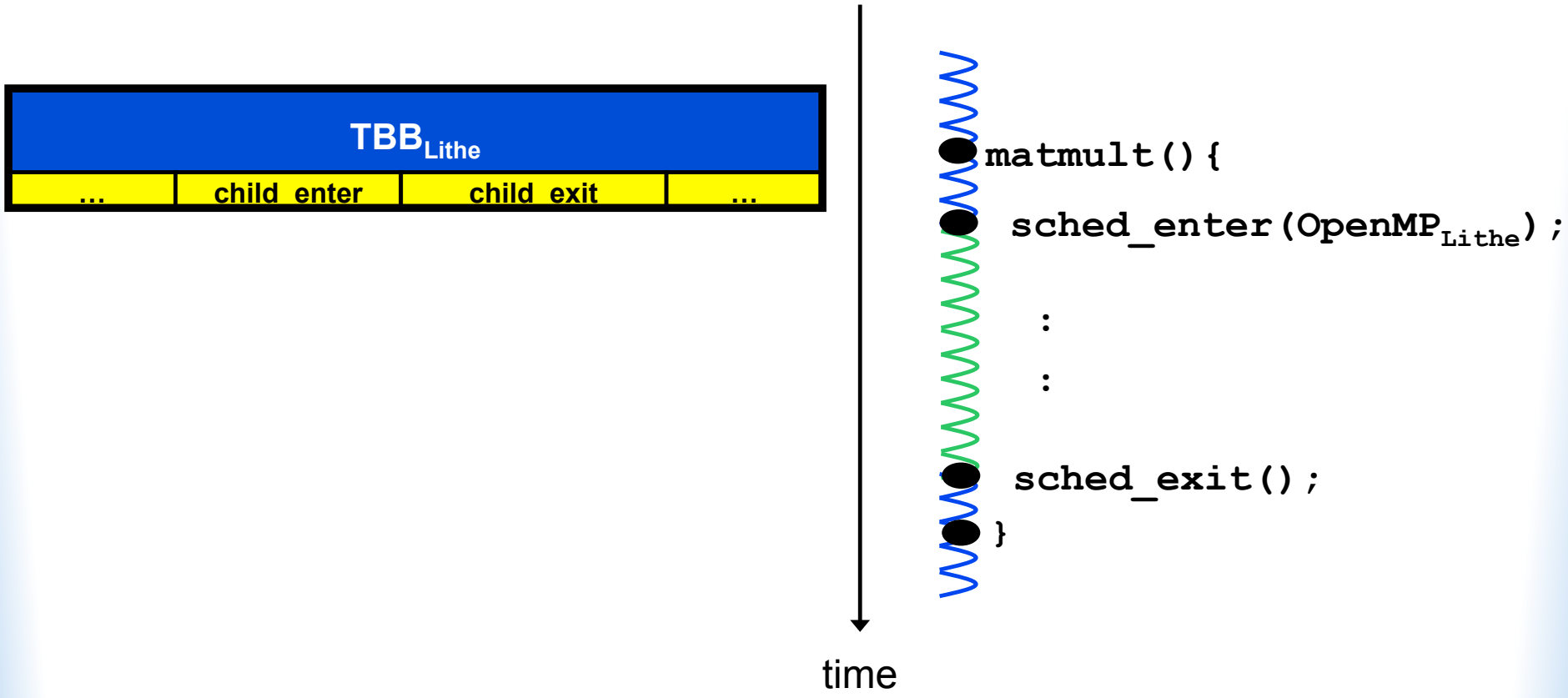


# Enter/Exit a Scheduler



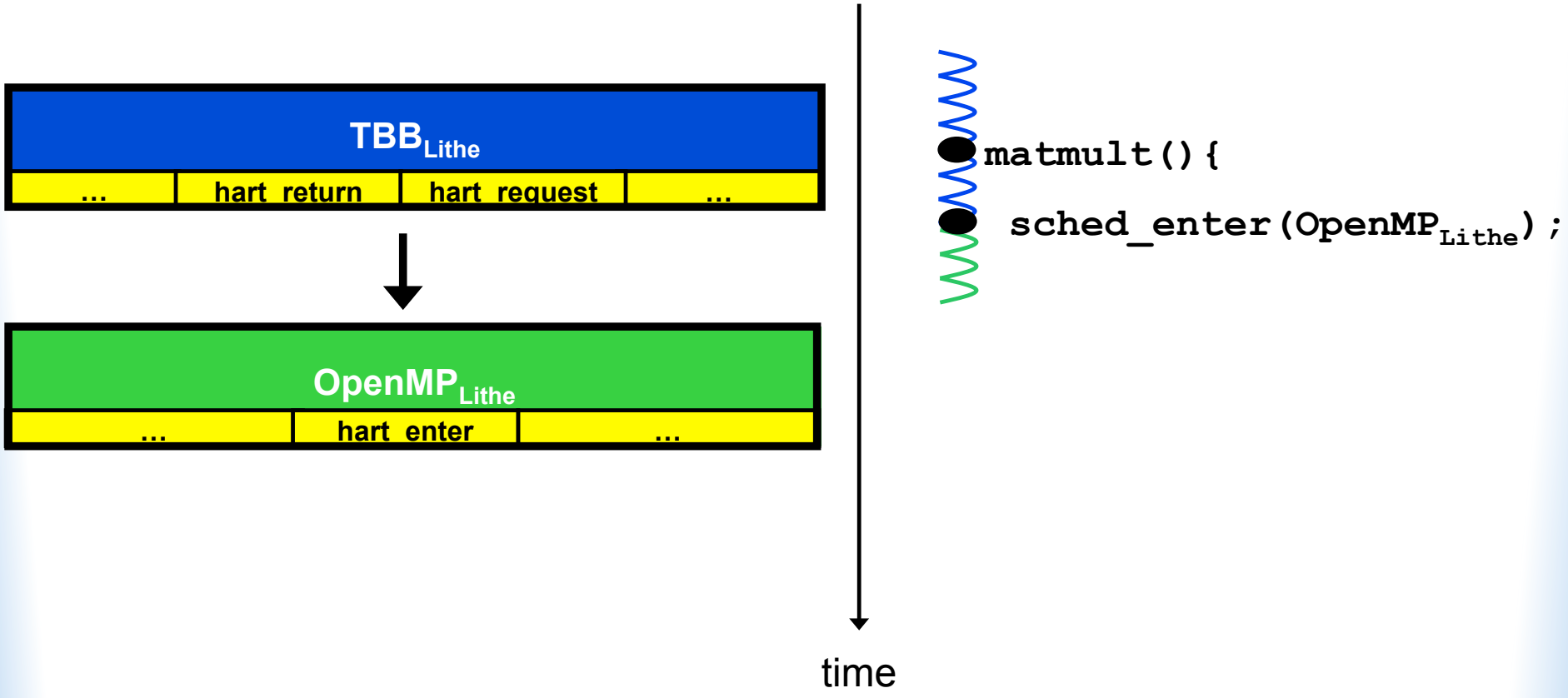



# Enter/Exit a Scheduler



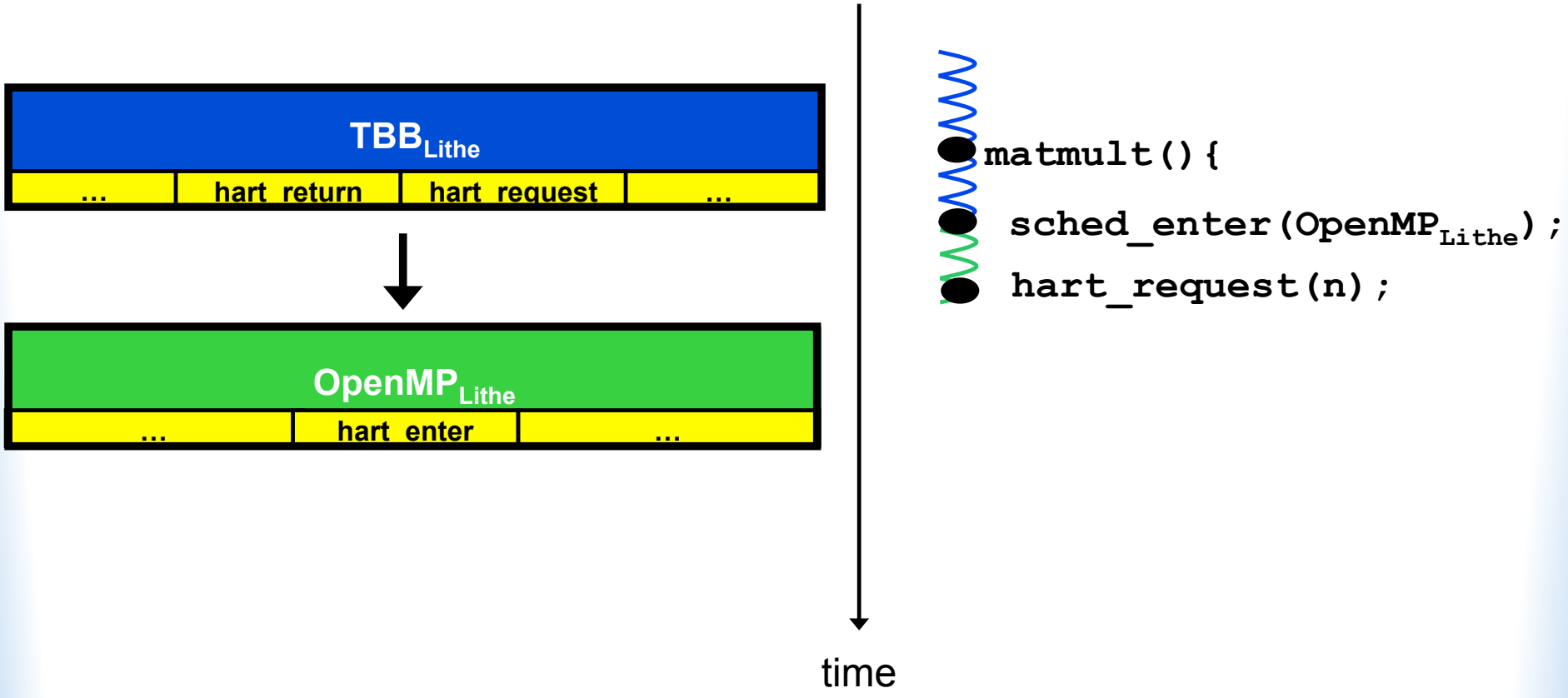

`sched_enter()` dynamically adds the new scheduler to the hierarchy.
`sched_exit()` dynamically removes a scheduler from the hierarchy.

# Request/Grant/Yield a Hart



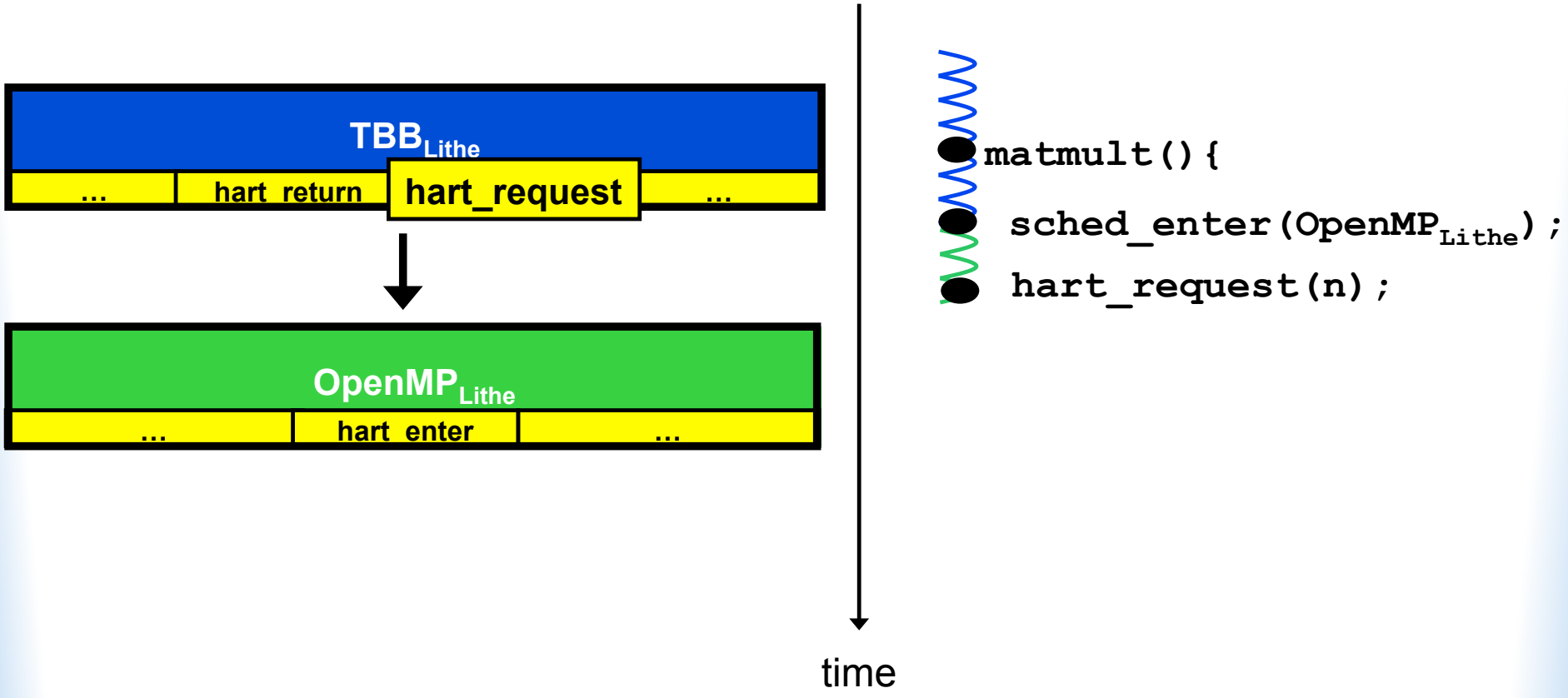

`hart_request()`,`hart_grant()`,`hart_yield()`
**transfer harts between a parent and child.**

# Request/Grant/Yield a Hart



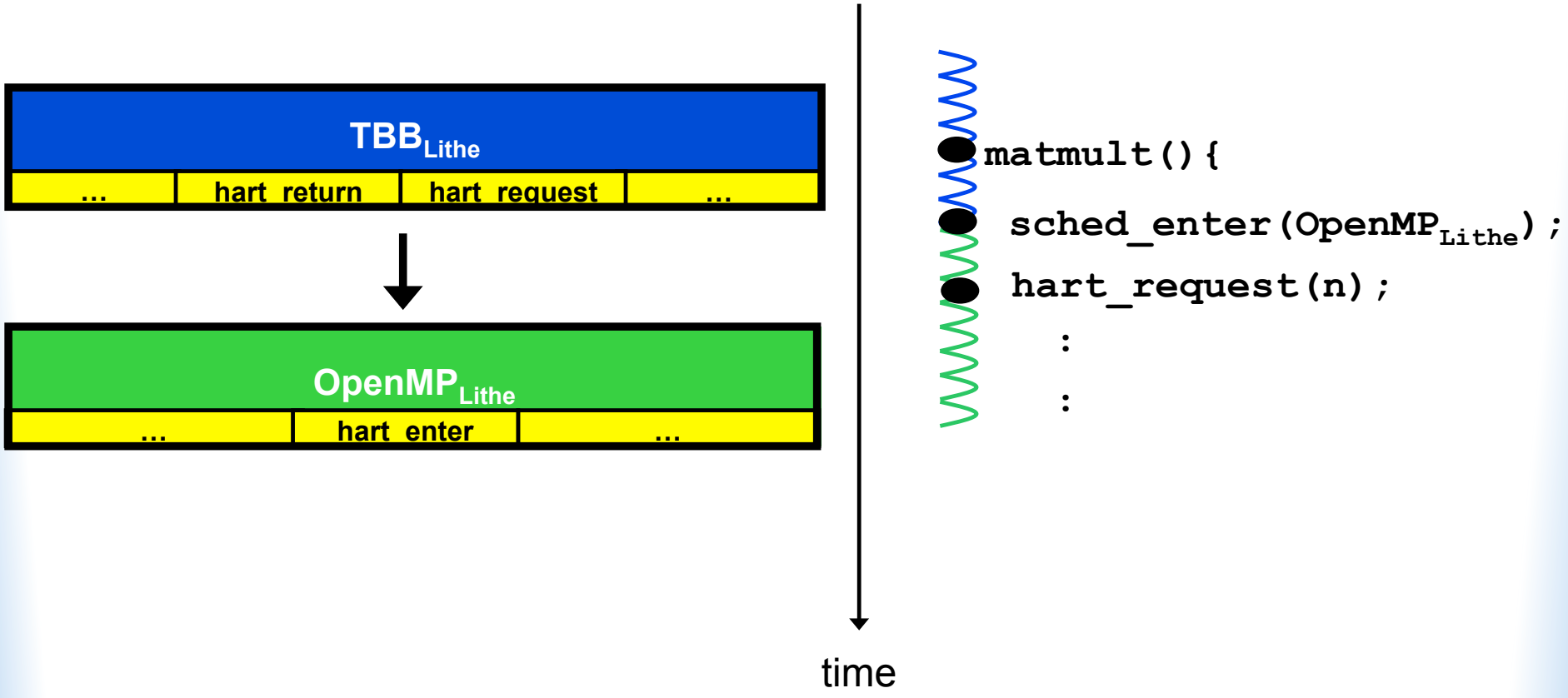

`hart_request()`,`hart_grant()`,`hart_yield()`
**transfer harts between a parent and child.**

# Request/Grant/Yield a Hart



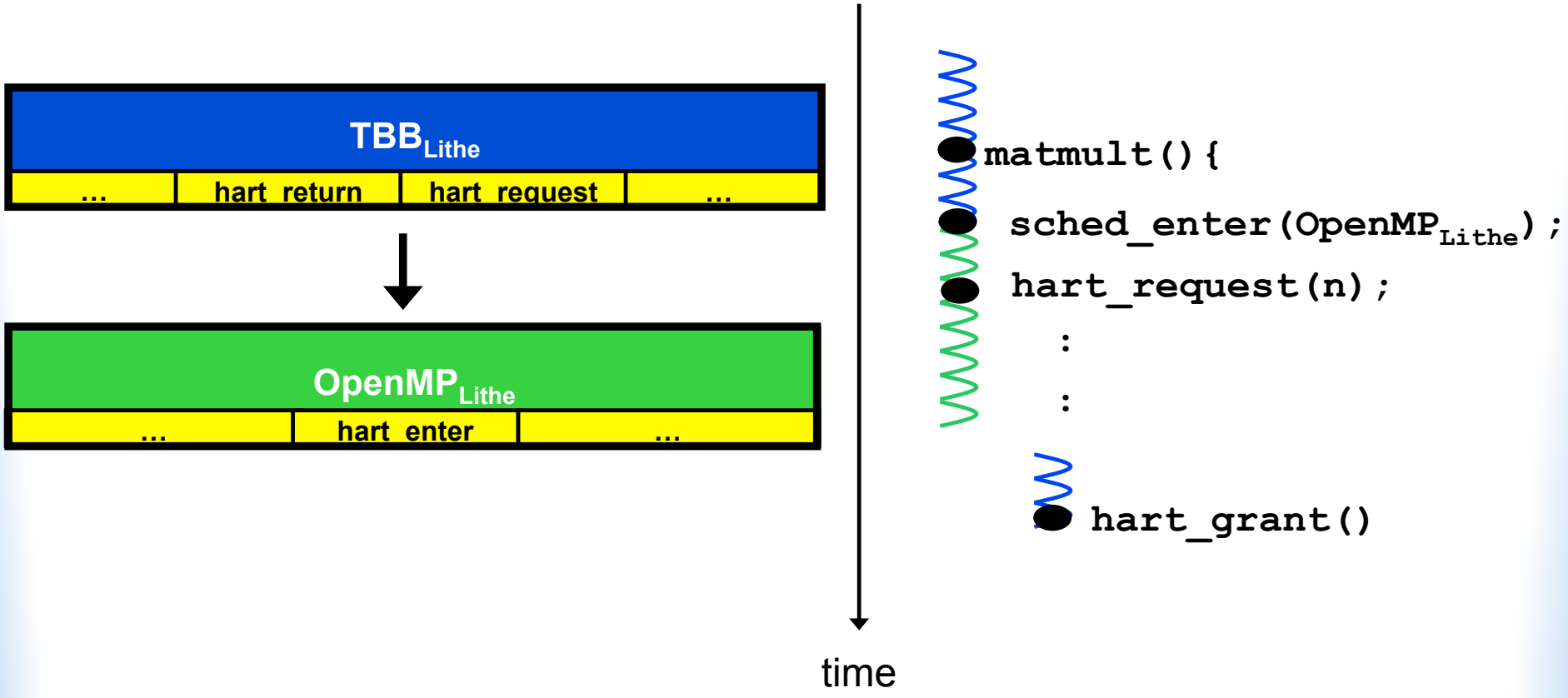

`hart_request()`,`hart_grant()`,`hart_yield()`
**transfer harts between a parent and child.**

# Request/Grant/Yield a Hart



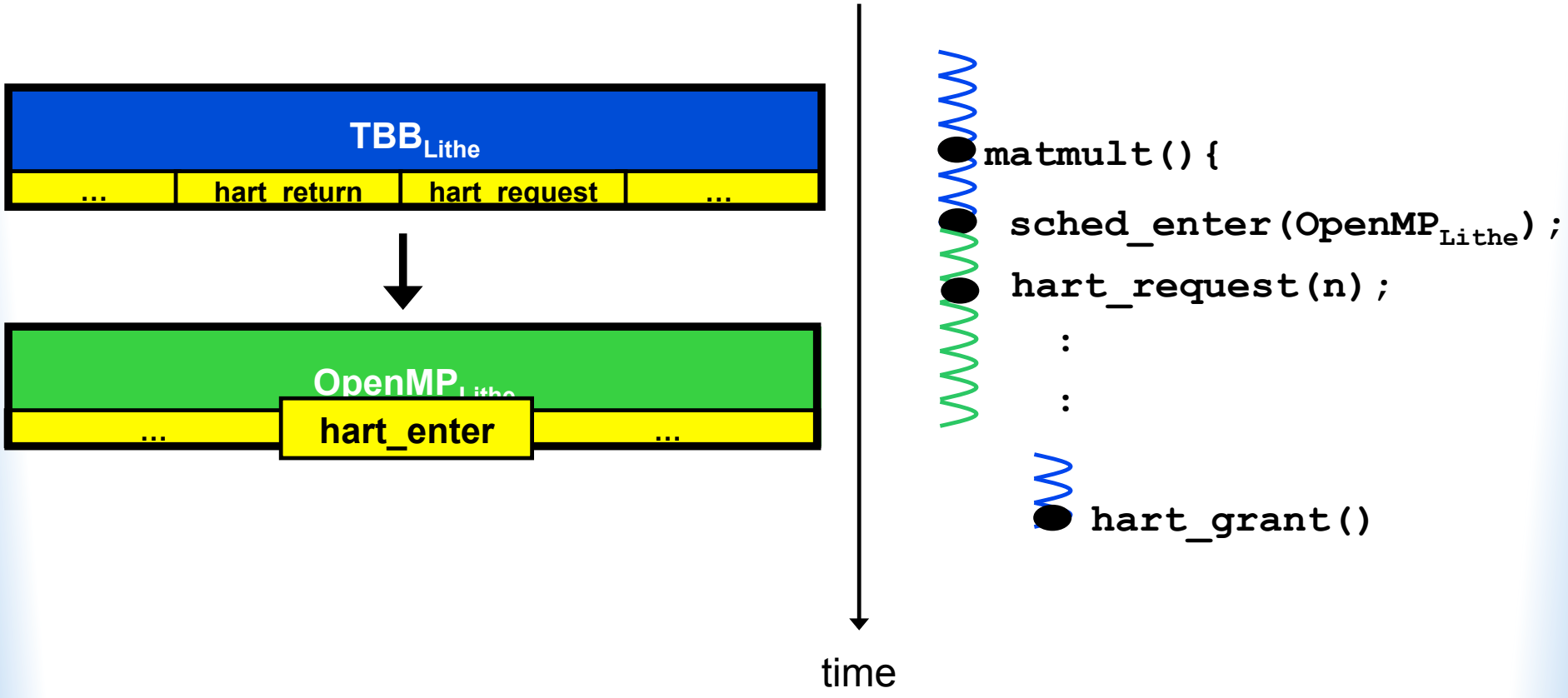

`hart_request()`,`hart_grant()`,`hart_yield()`
**transfer harts between a parent and child.**

# Request/Grant/Yield a Hart



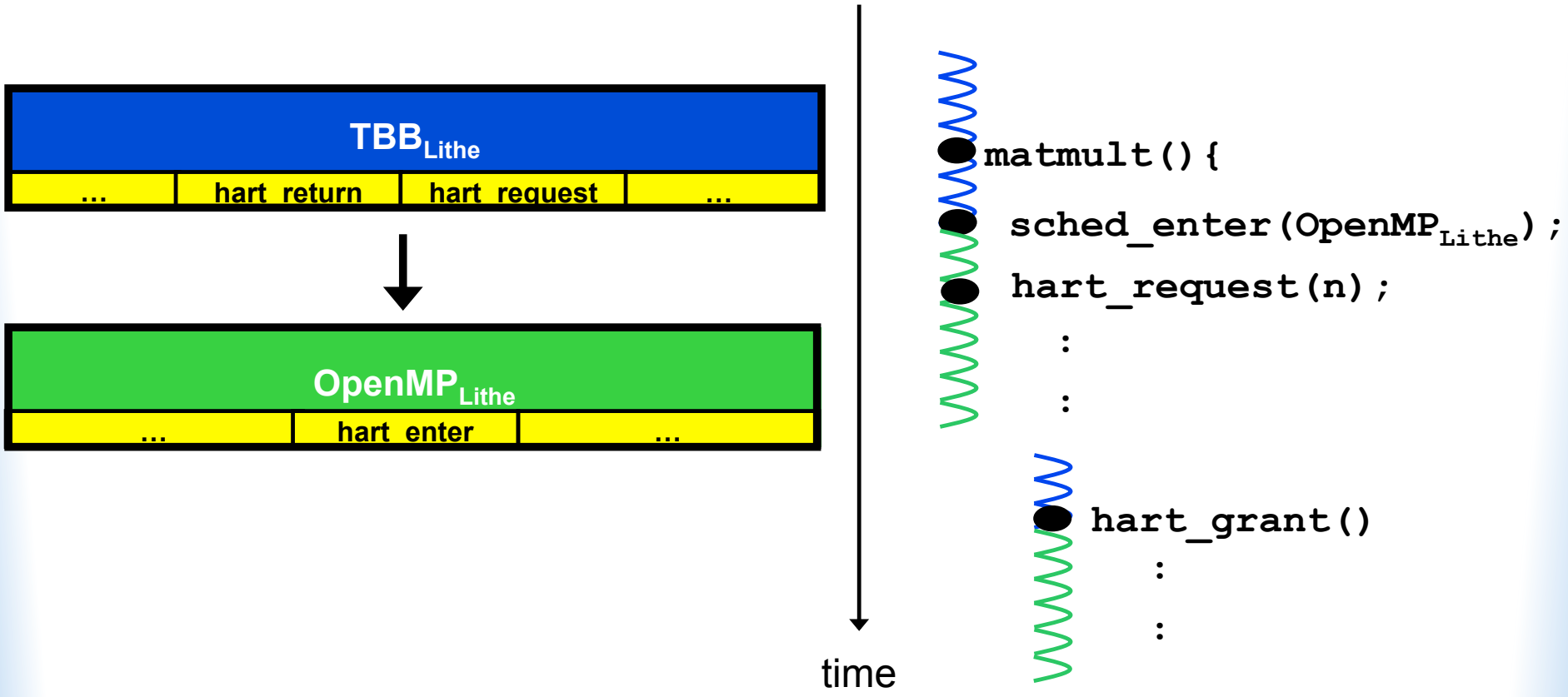

`hart_request()`,`hart_grant()`,`hart_yield()`
**transfer harts between a parent and child.**

# Request/Grant/Yield a Hart



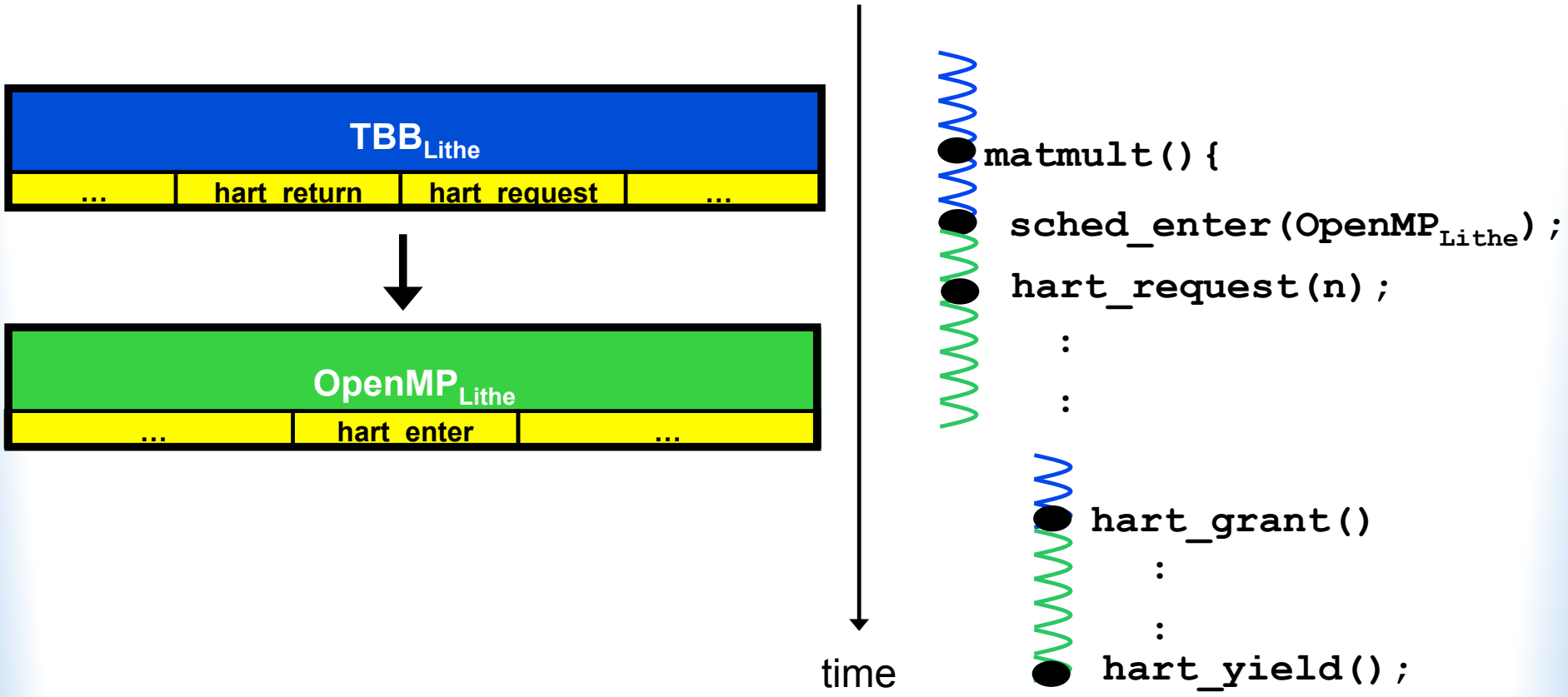

`hart_request()`,`hart_grant()`,`hart_yield()`
transfer harts between a parent and child.

# Request/Grant/Yield a Hart



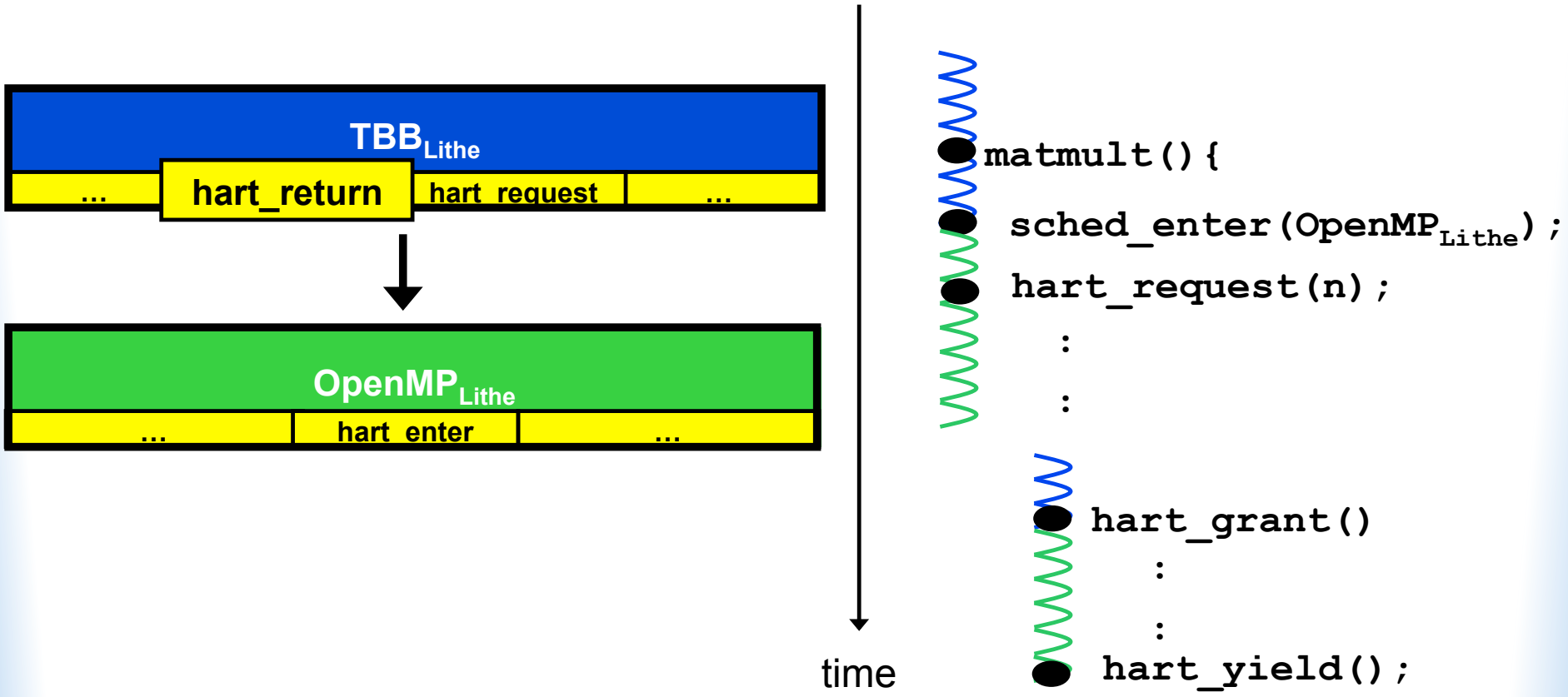

`hart_request()`,`hart_grant()`,`hart_yield()`
transfer harts between a parent and child.

# Request/Grant/Yield a Hart



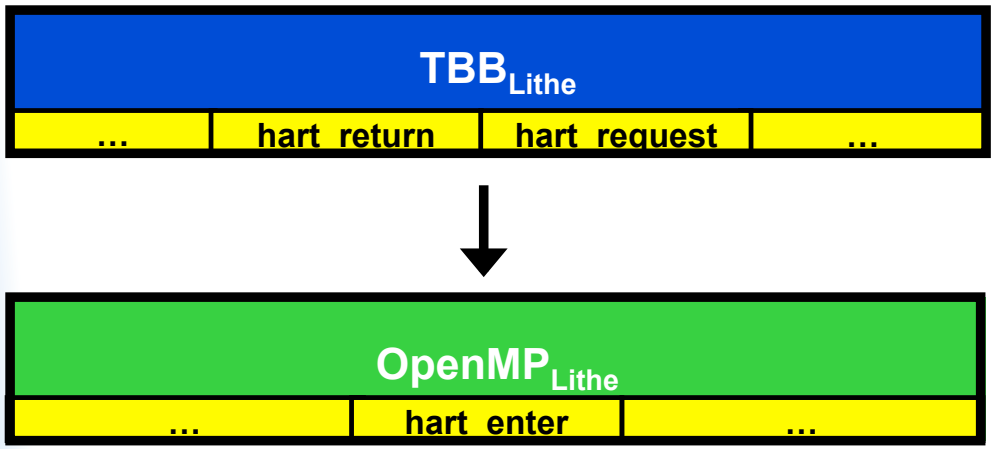

`hart_request()`,`hart_grant()`,`hart_yield()`
transfer harts between a parent and child.

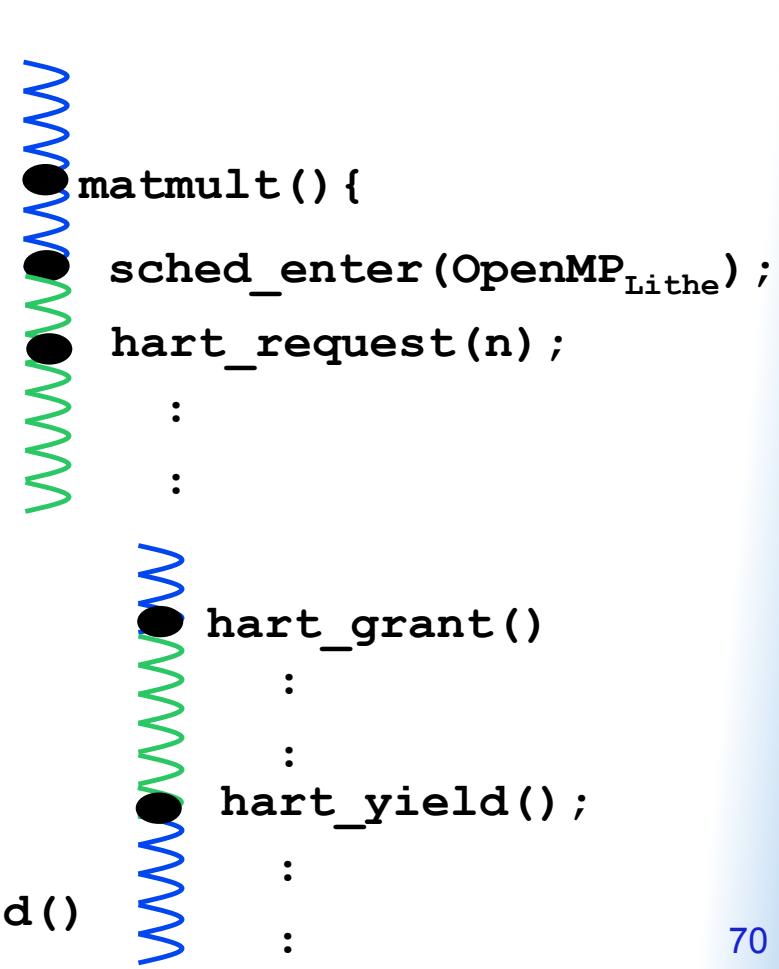
# Request/Grant/Yield a Hart




`hart_request()`,`hart_grant()`,`hart_yield()`
transfer harts between a parent and child.

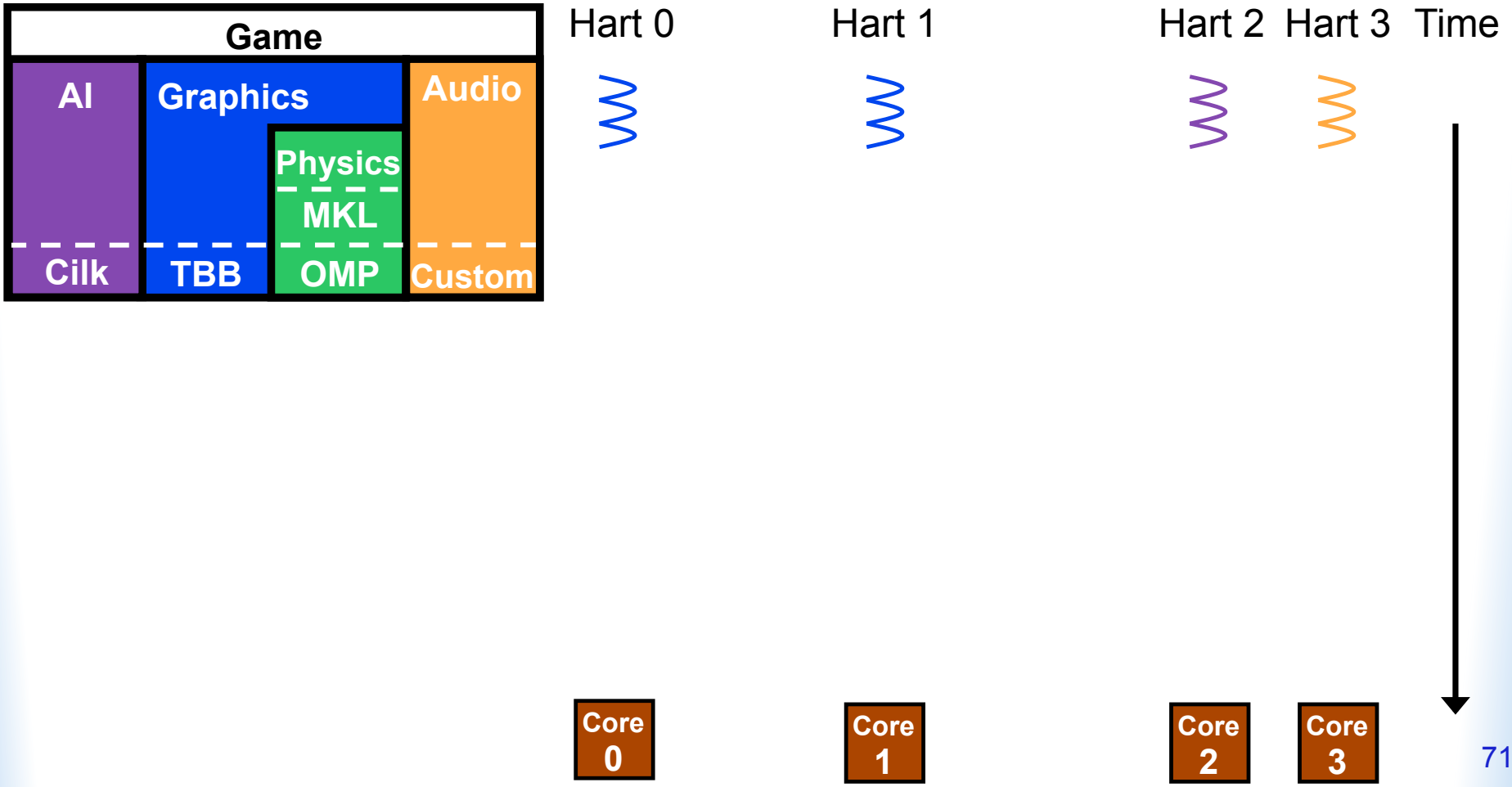
# Request/Grant/Yield a Hart



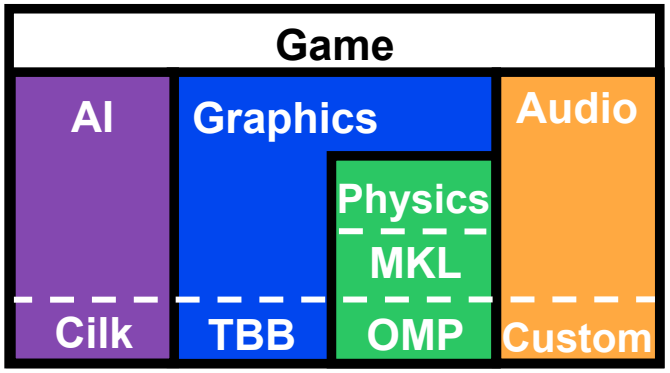



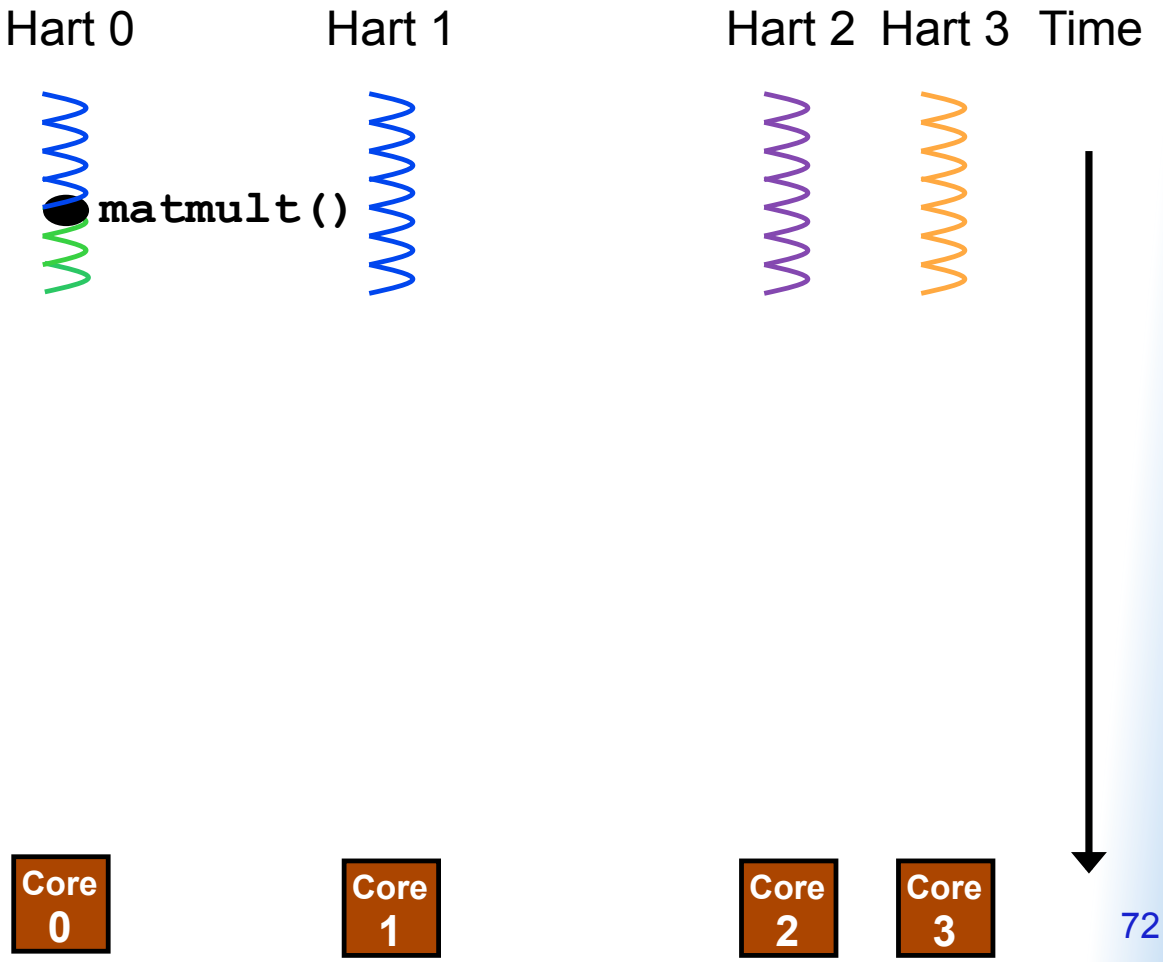

`hart_request()`,`hart_grant()`,`hart_yield()`
**transfer harts between a parent and child.**

# Putting it All Together



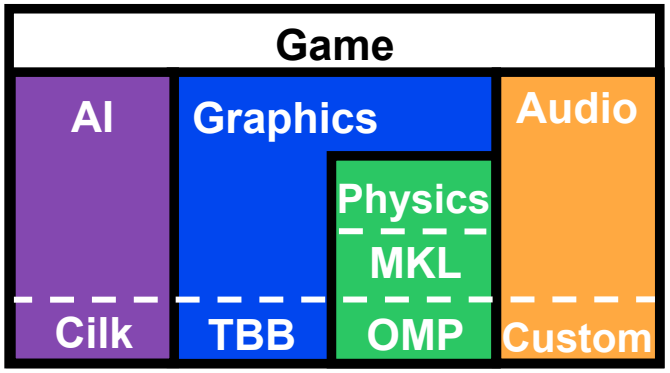
# Putting it All Together

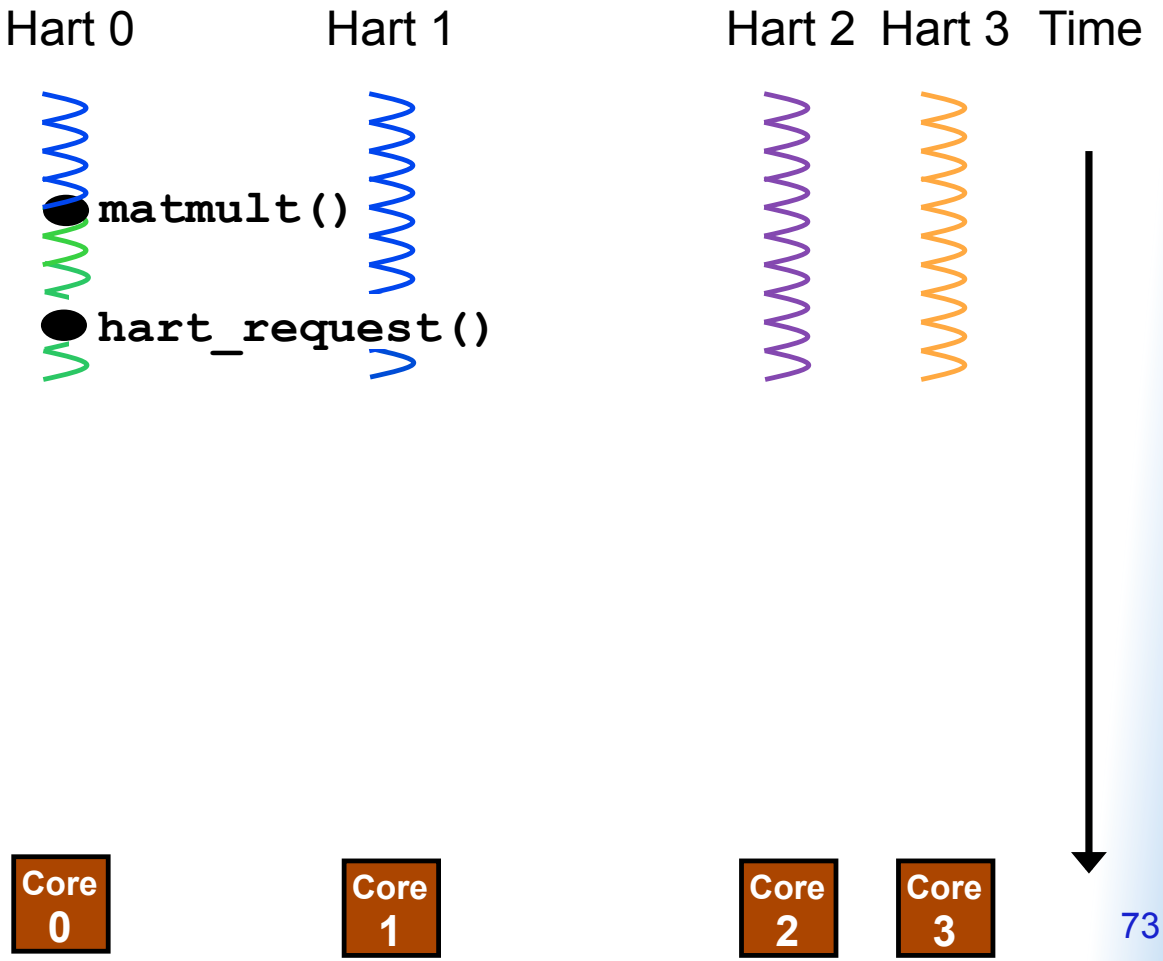



```
tbb::task() {
  matmult() {
   #pragma omp parallel
    :
  }
  :
}
```

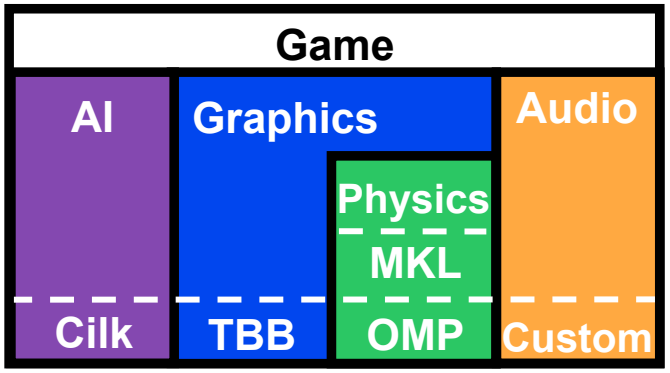
# Putting it All Together



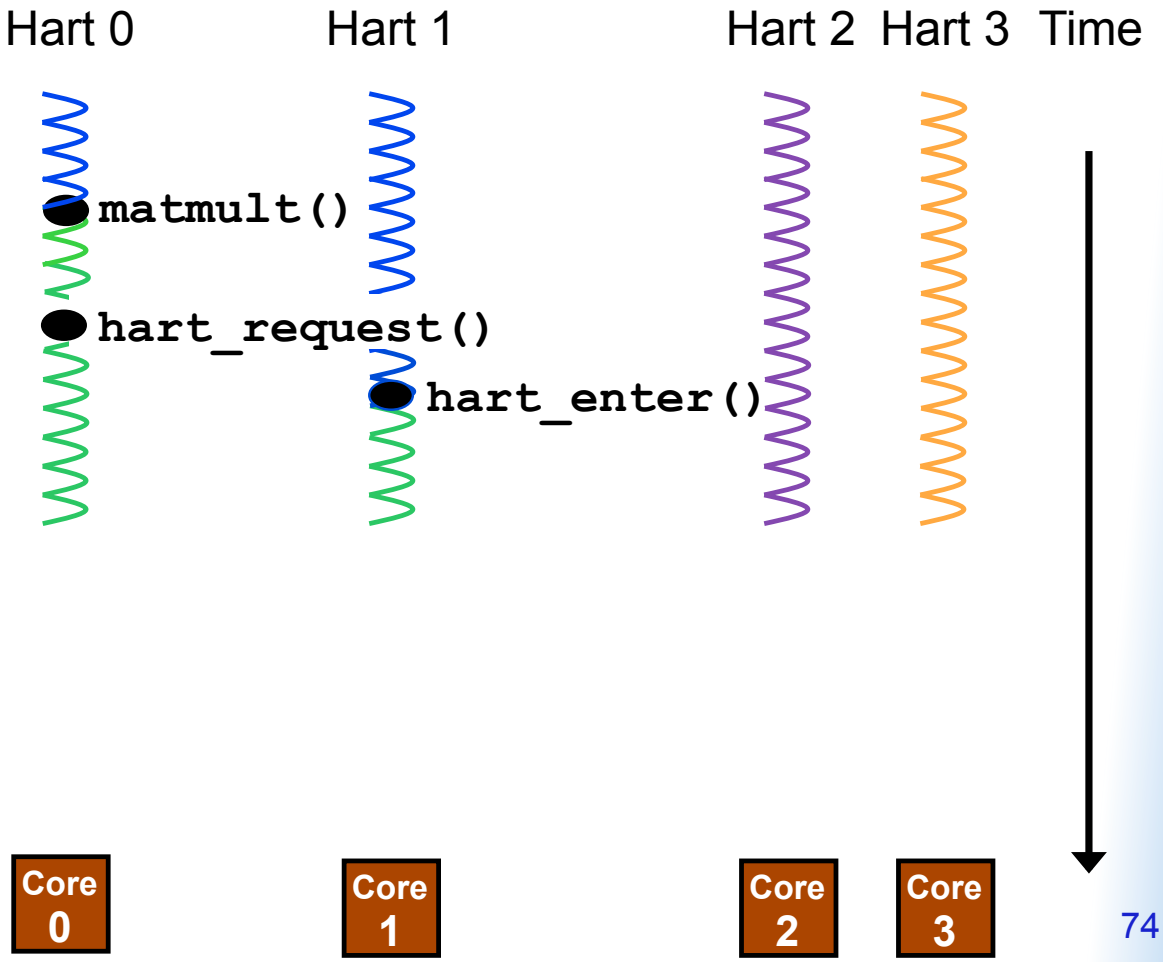




```
tbb::task() {
  matmult() {
   #pragma omp parallel
    :
   }
   :
}
```

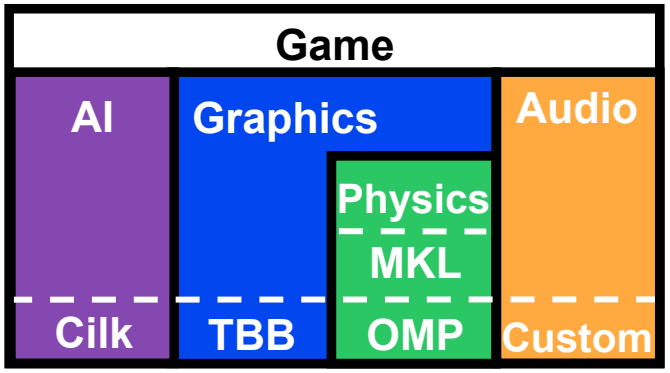
# Putting it All Together



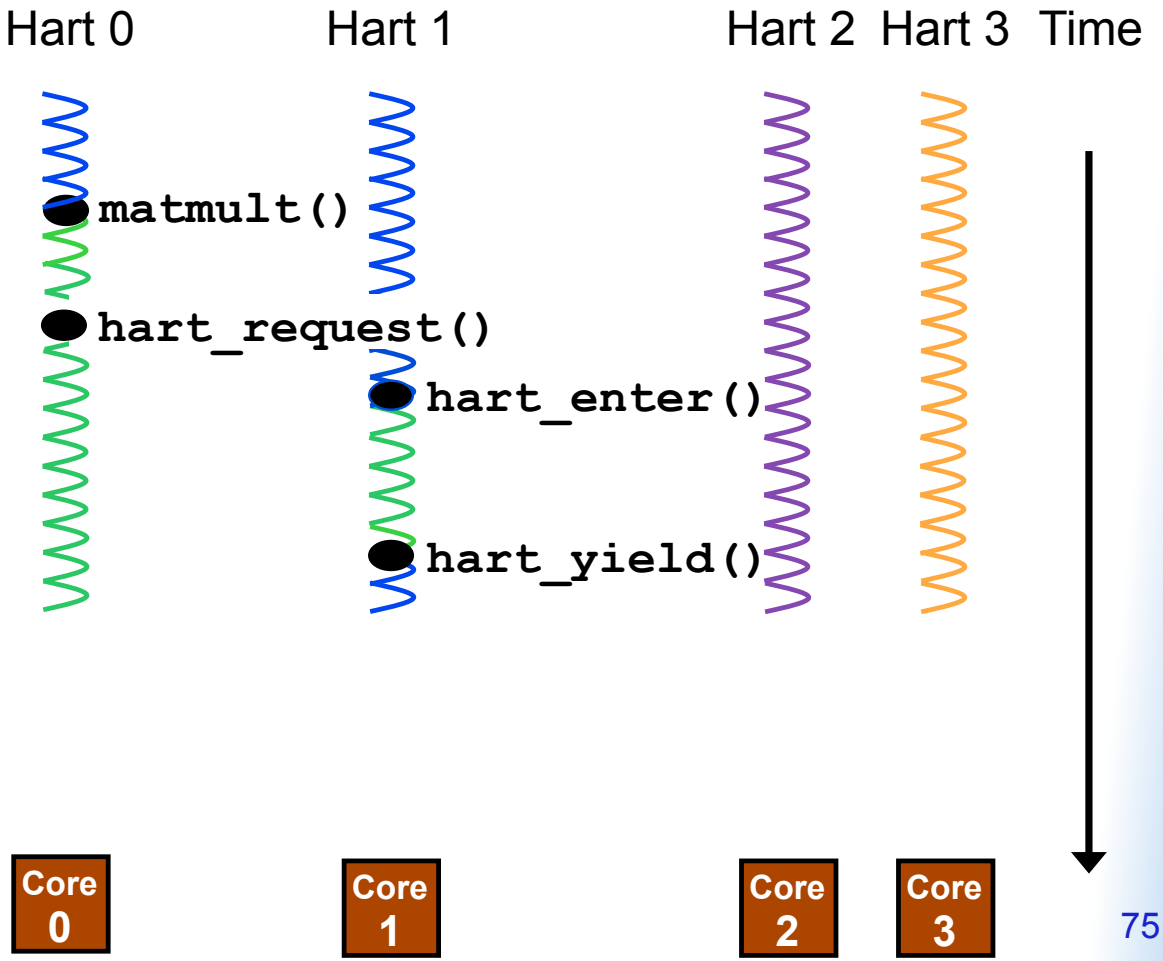

```
tbb::task() {
  matmult() {
   #pragma omp parallel
    :
   }
   :
}
```



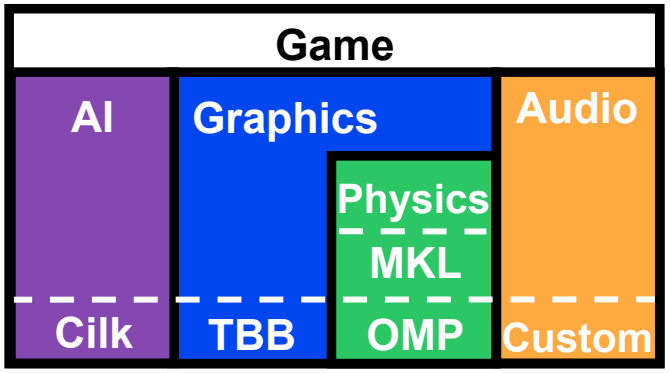
# Putting it All Together



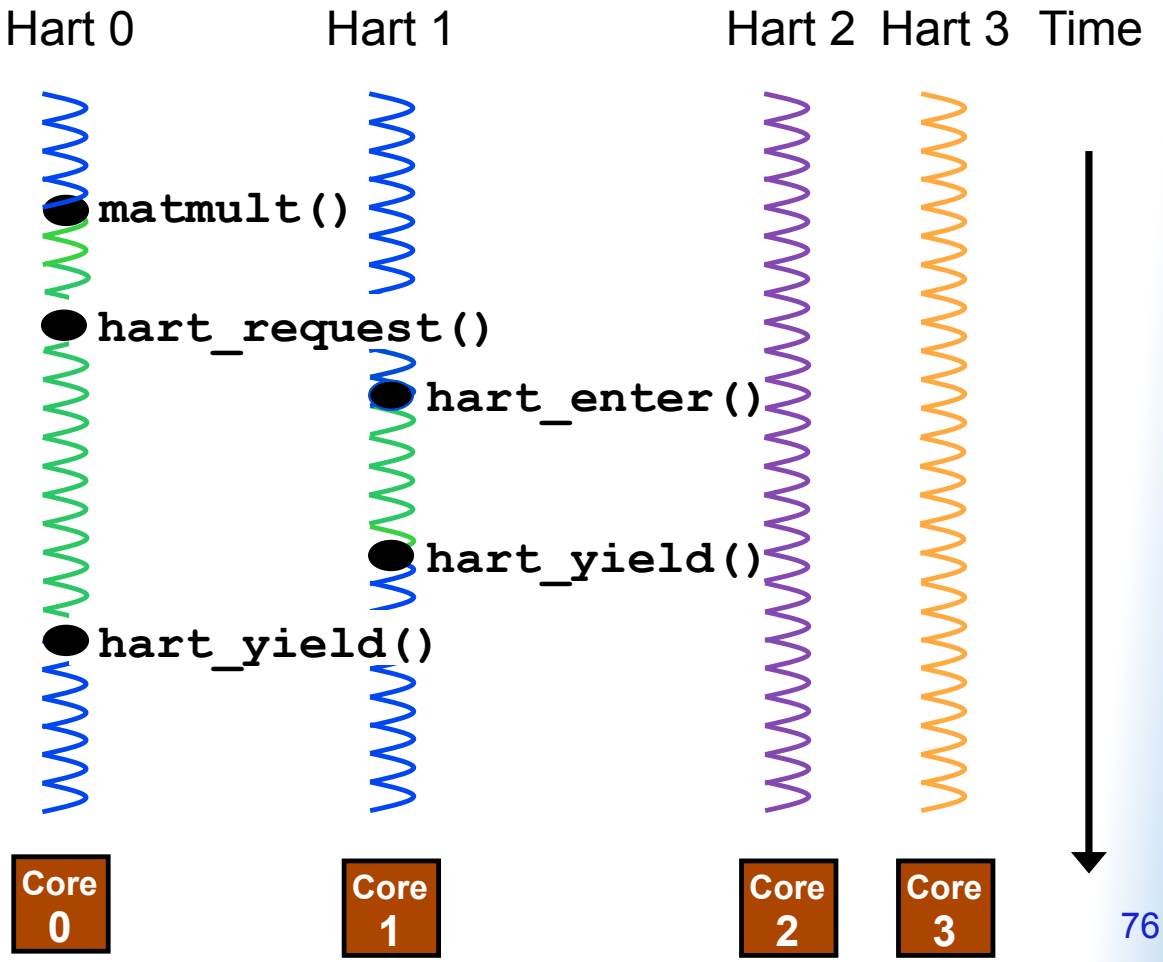

```
tbb::task() {
  matmult() {
    #pragma omp parallel
    :
   }
   :
}
```



# Putting it All Together




```
tbb::task() {
  matmult() {
    #pragma omp parallel
    :
   }
   :
}
```



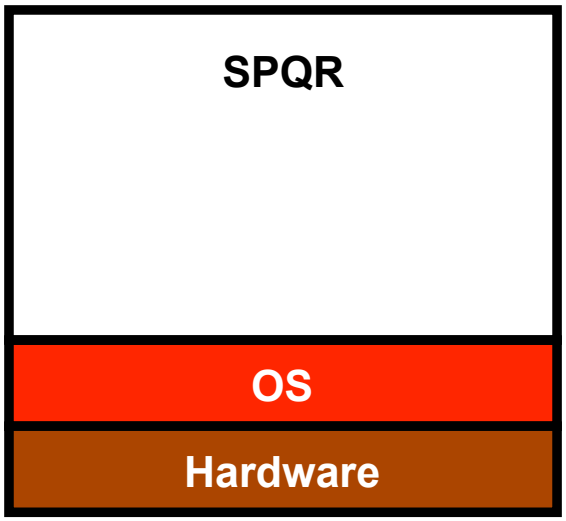
# Lithe Context Support

- Basic user-level threading API
  - Integrated directly to work well with harts
  - Can be extended by each library for custom scheduler support
  - Callbacks into scheduler similar to hart callbacks when operation performed

```
lithe_context_init()
lithe_context_cleanup()
lithe_context_run()
lithe_context_block()
lithe_context_unblock()
lithe_context_yield()
lithe_context_exit()
```

# Real World Example

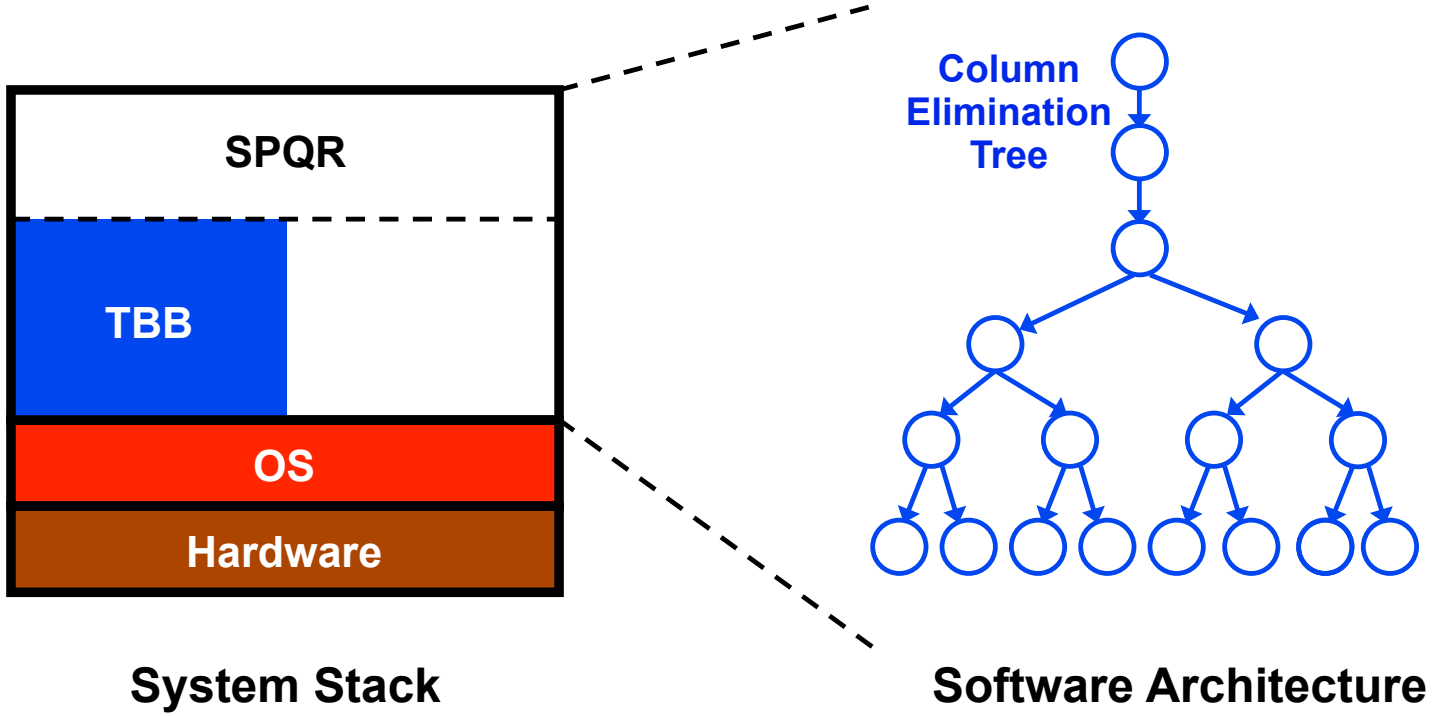
**Sparse QR Factorization**
(Tim Davis, Univ of Florida)




**System Stack**

# Real World Example

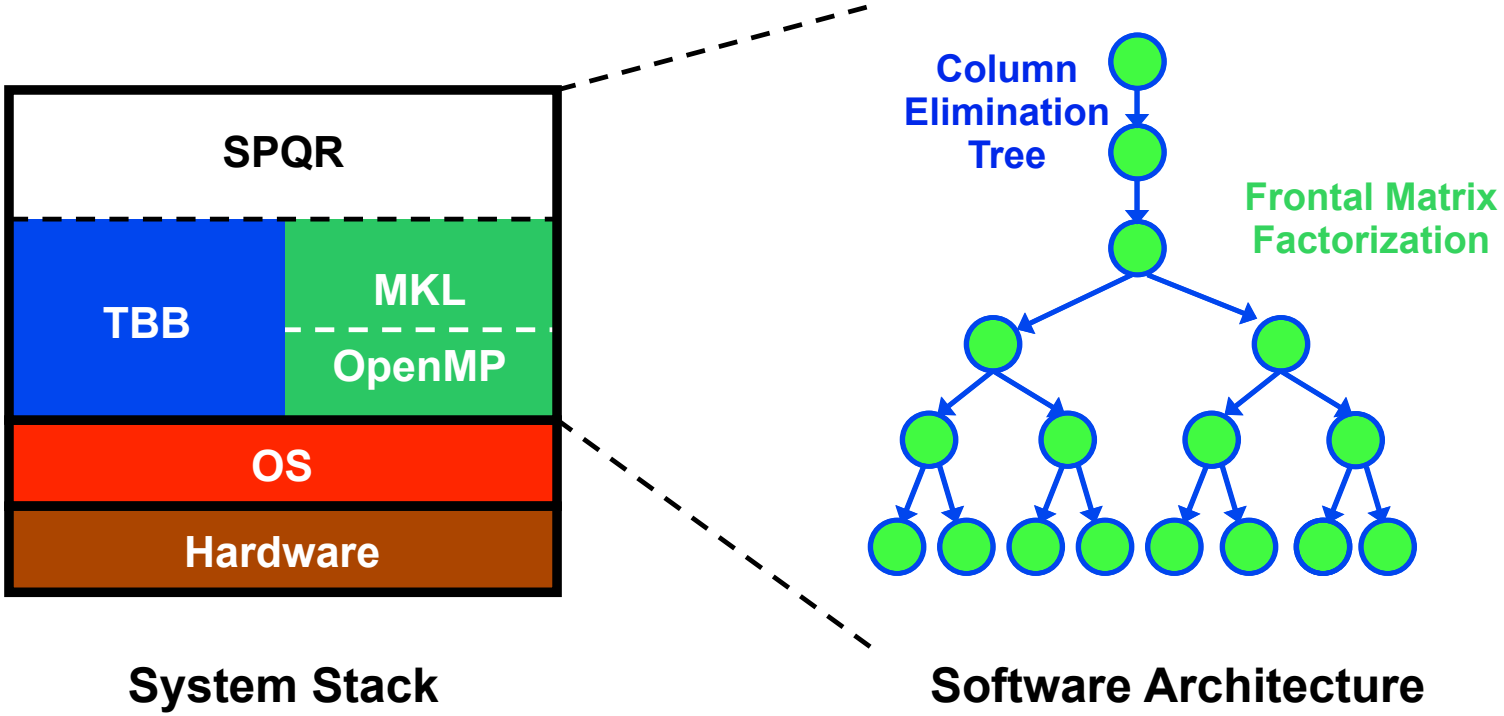
**Sparse QR Factorization**
(Tim Davis, Univ of Florida)




**System Stack** **Software Architecture**

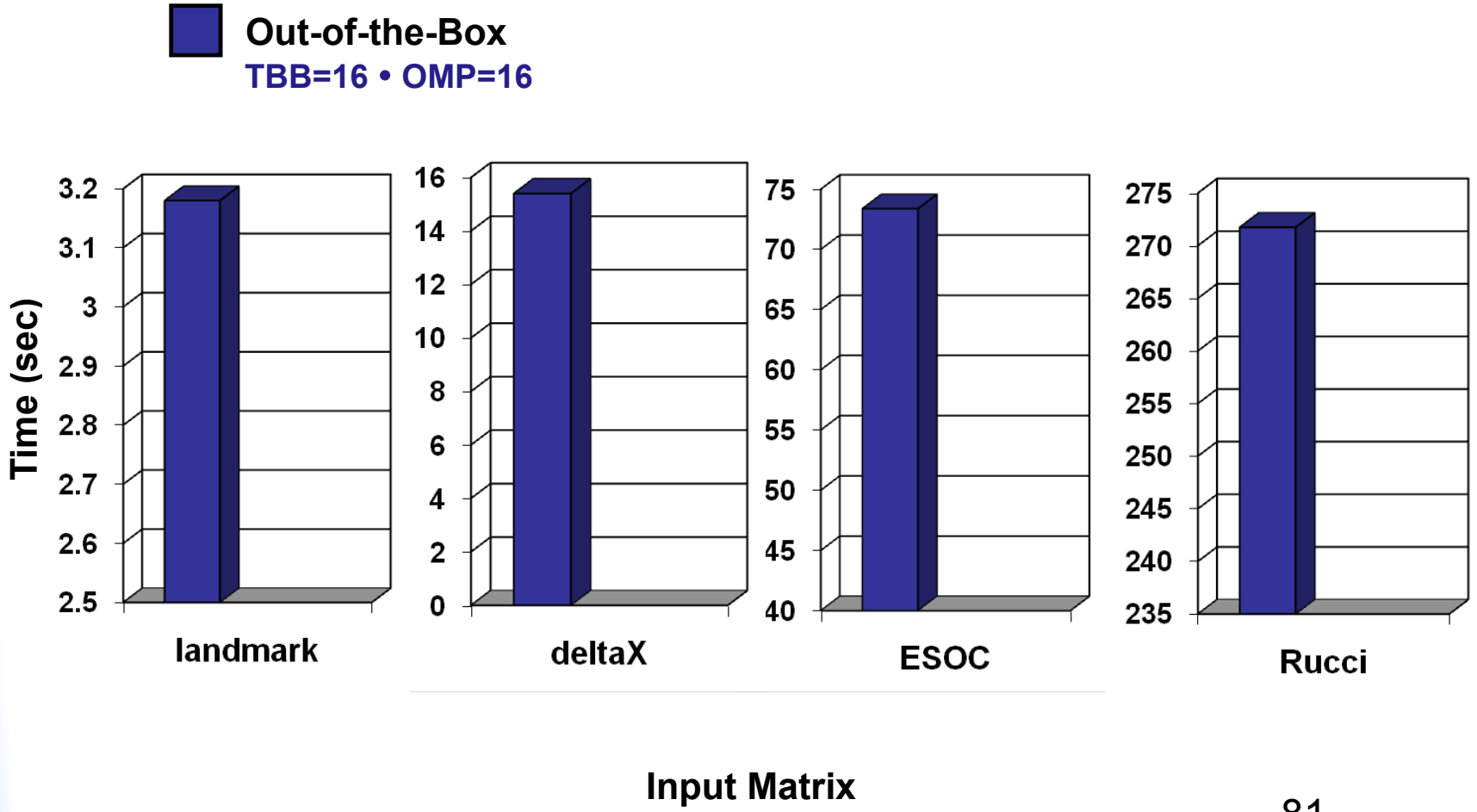
# Real World Example

**Sparse QR Factorization**
(Tim Davis, Univ of Florida)




**System Stack** **Software Architecture**

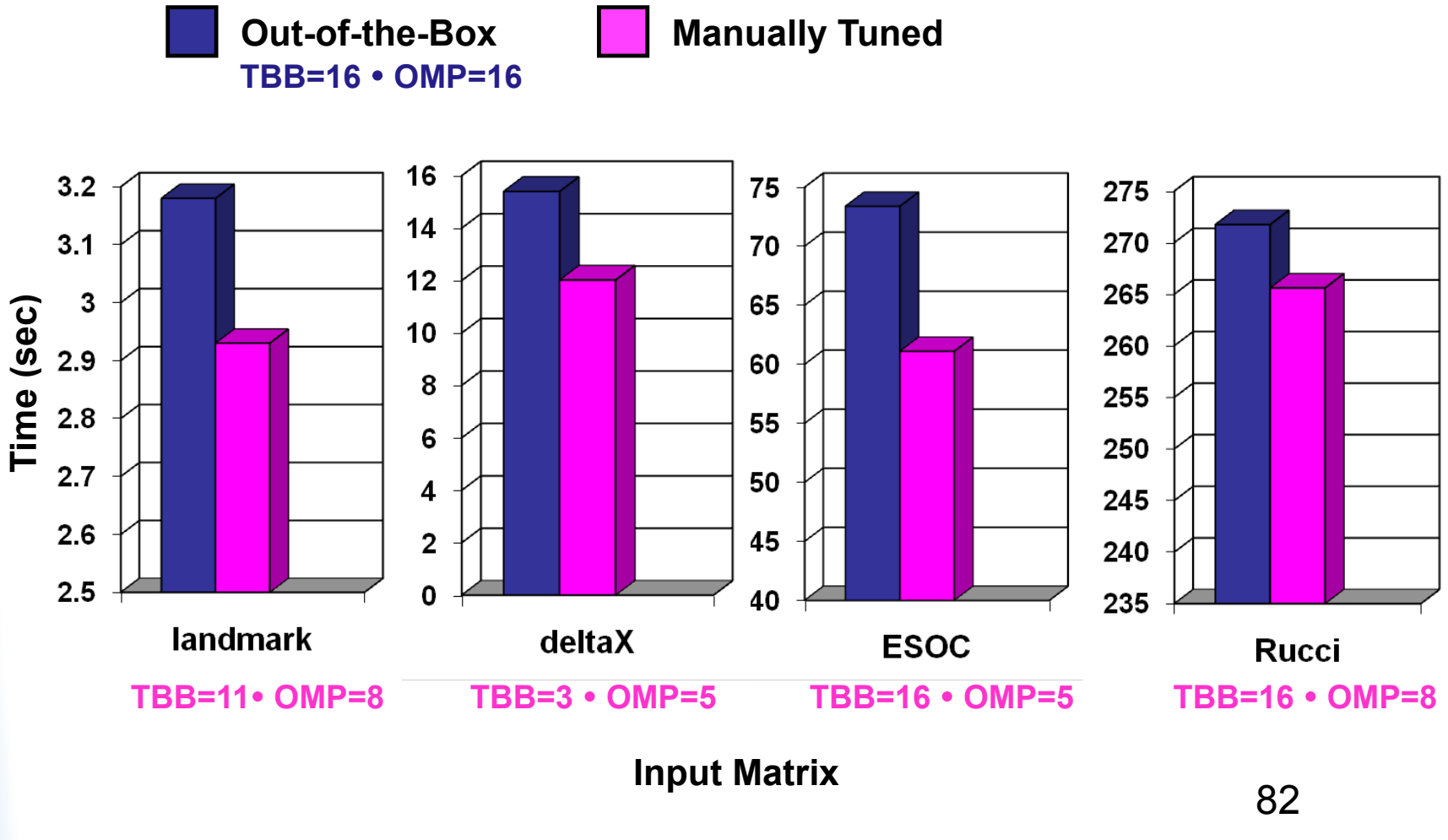
# Performance of SPQR on 16-Core machine






81

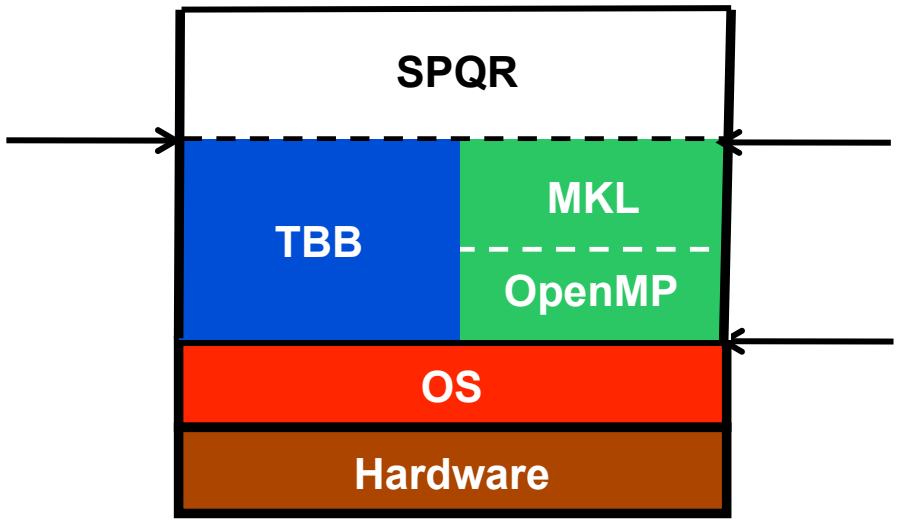
# Performance of SPQR on 16-Core machine





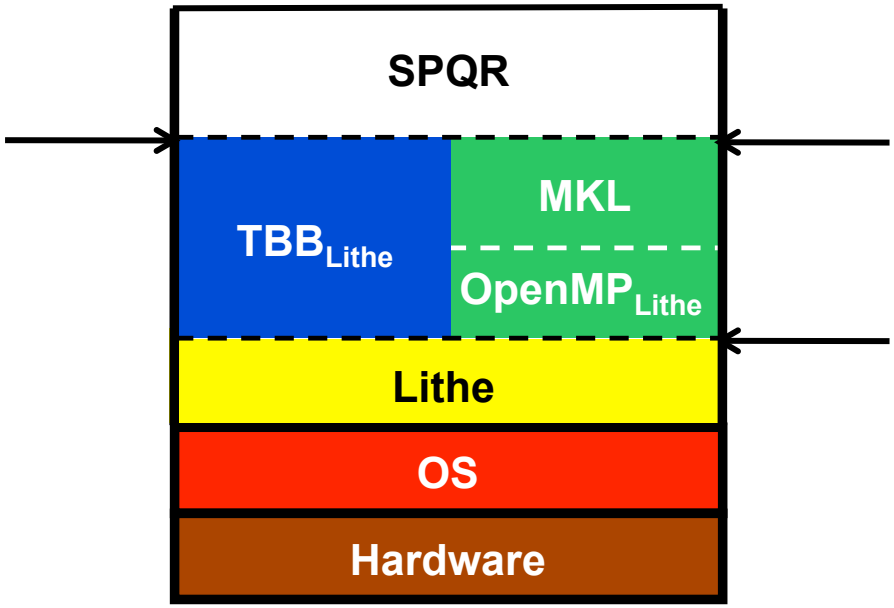

82

# SPQR with Lithe



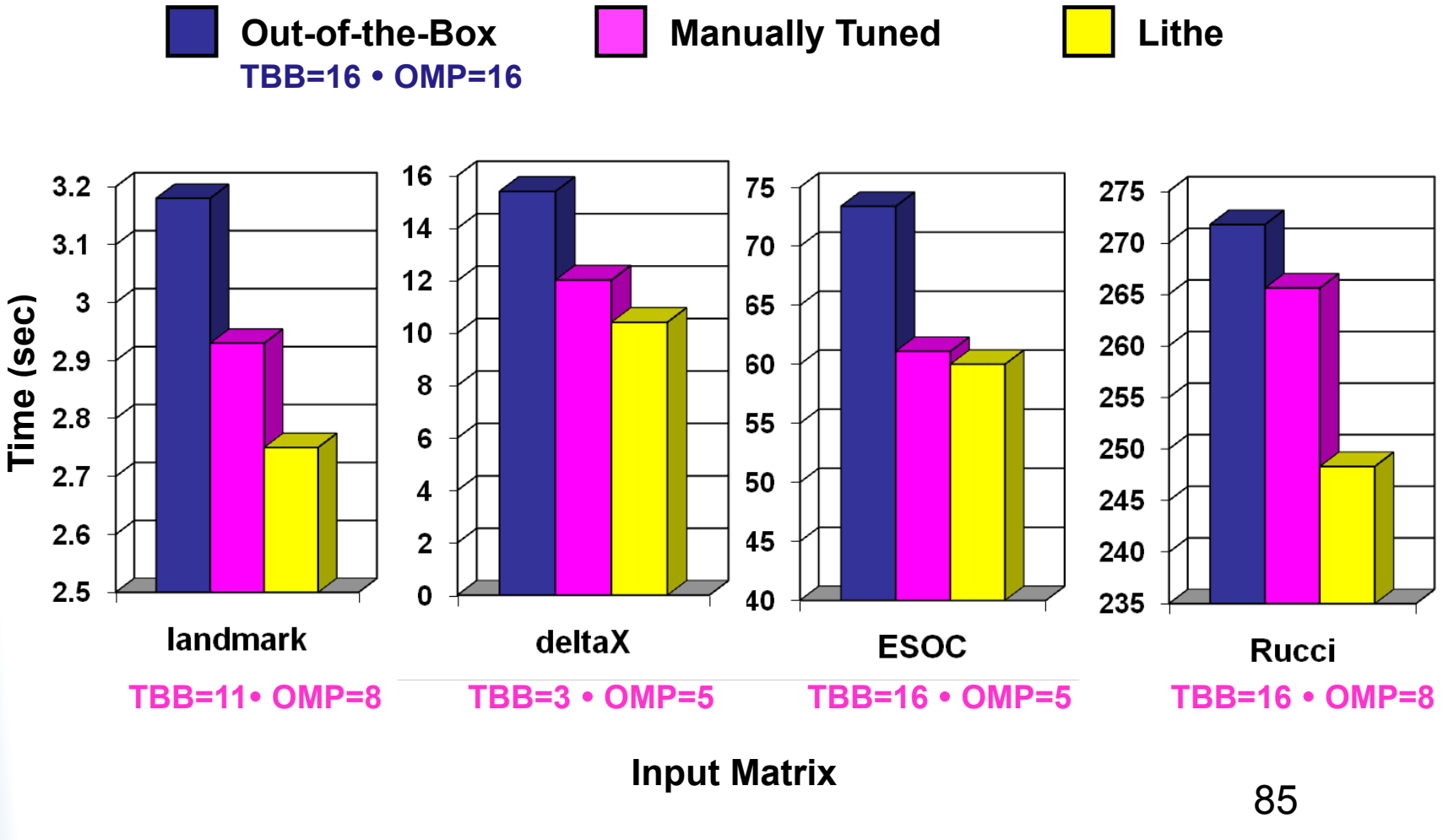

- Library interfaces remain the same
- Zero lines of high-level code changed (SPQR, MKL)

# SPQR with Lithe




- Library interfaces remain the same
- Zero lines of high-level code changed (SPQR, MKL)
- Just link in Lithe runtime + Lithe versions of libraries (TBB, OpenMP)

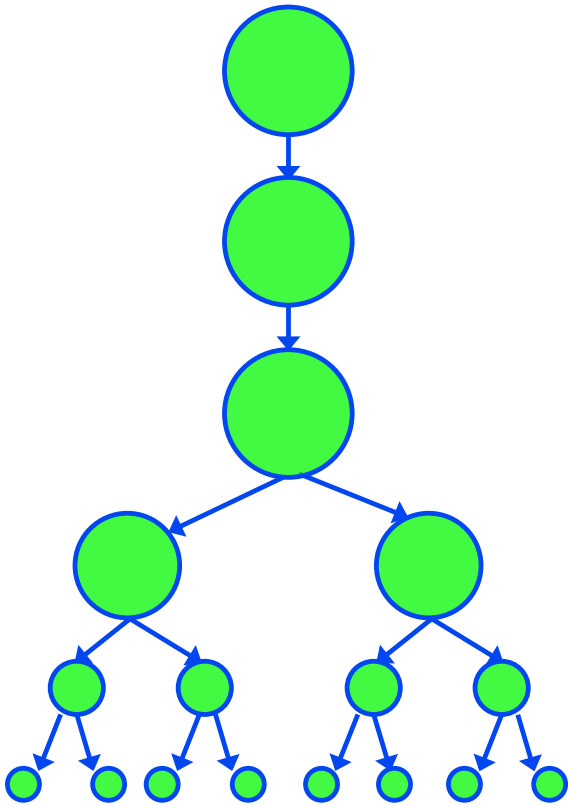
# Performance of SPQR with Lithe



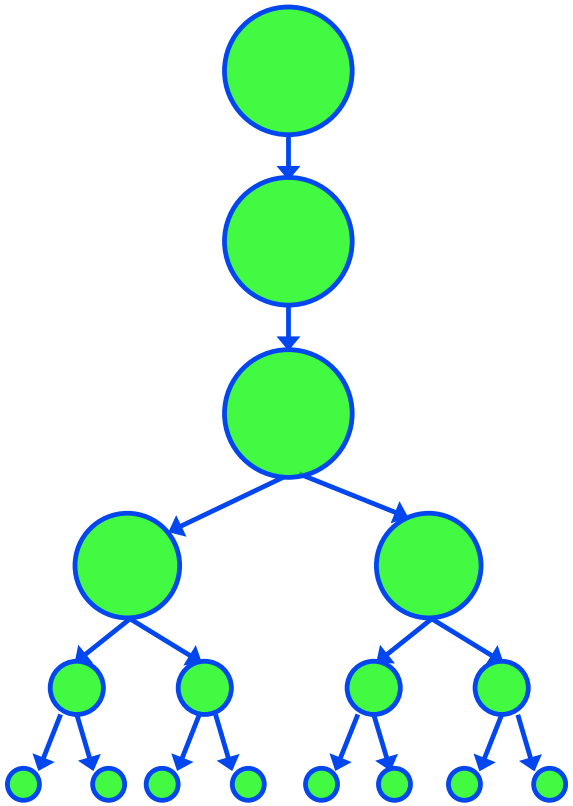



85

# Lithe Enables Flexible Sharing of Resources

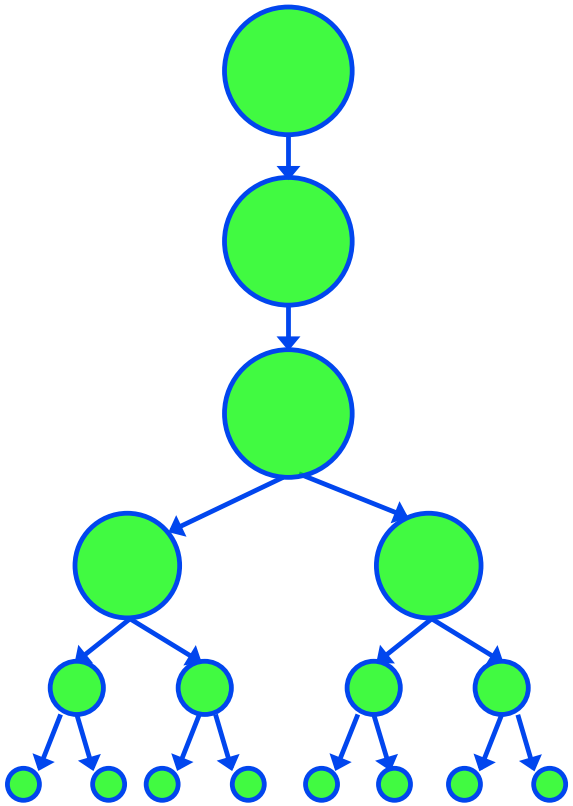


# Lithe Enables Flexible Sharing of Resources



Give resources to OpenMP

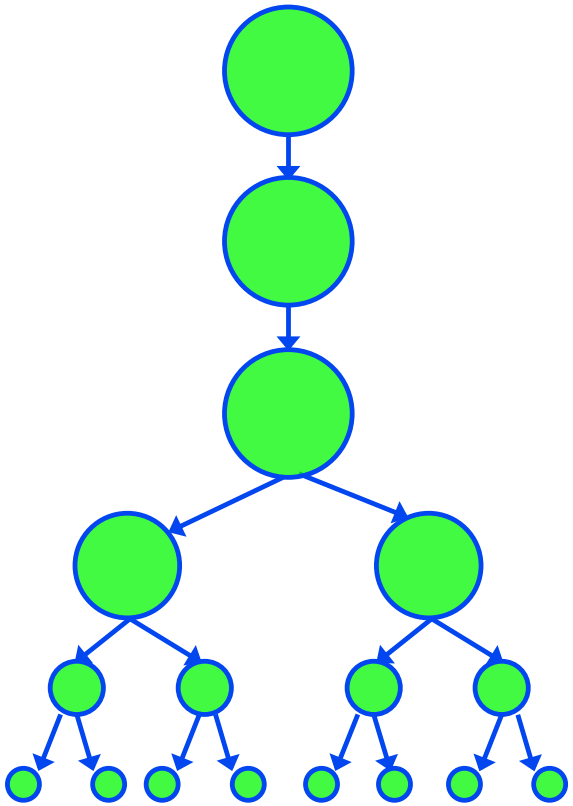
# Lithe Enables Flexible Sharing of Resources



Give resources to OpenMP

Give resources to TBB

# Lithe Enables Flexible Sharing of Resources

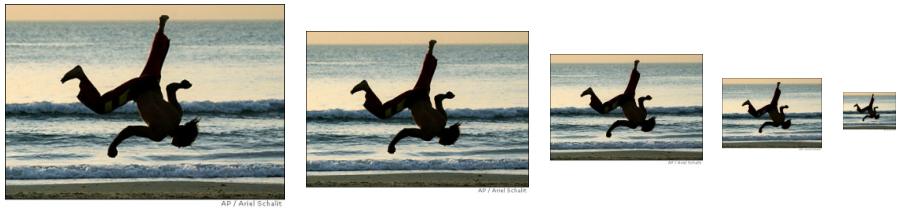


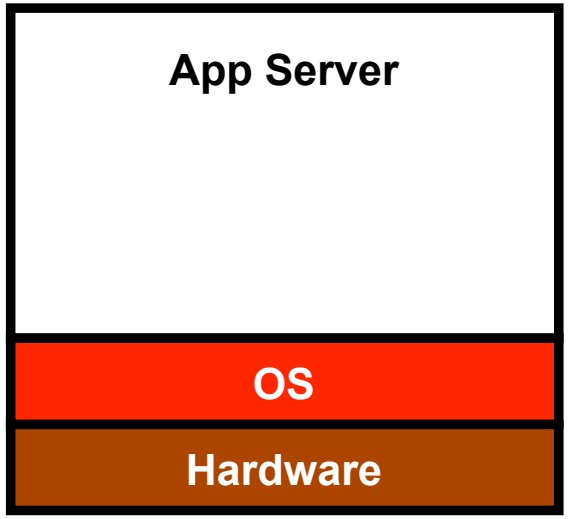
Give resources to OpenMP

Give resources to TBB

***Manual tuning is stuck with 1 TBB/OMP config throughout run.***

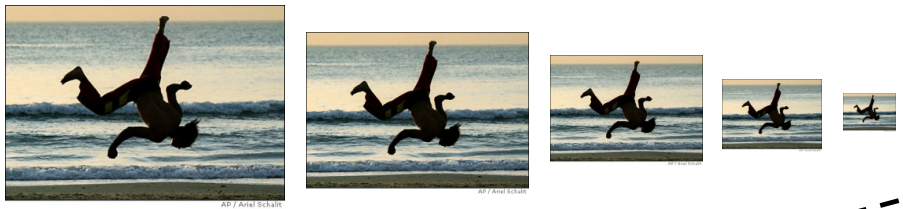
# Flickr-Like Image Processing App Server

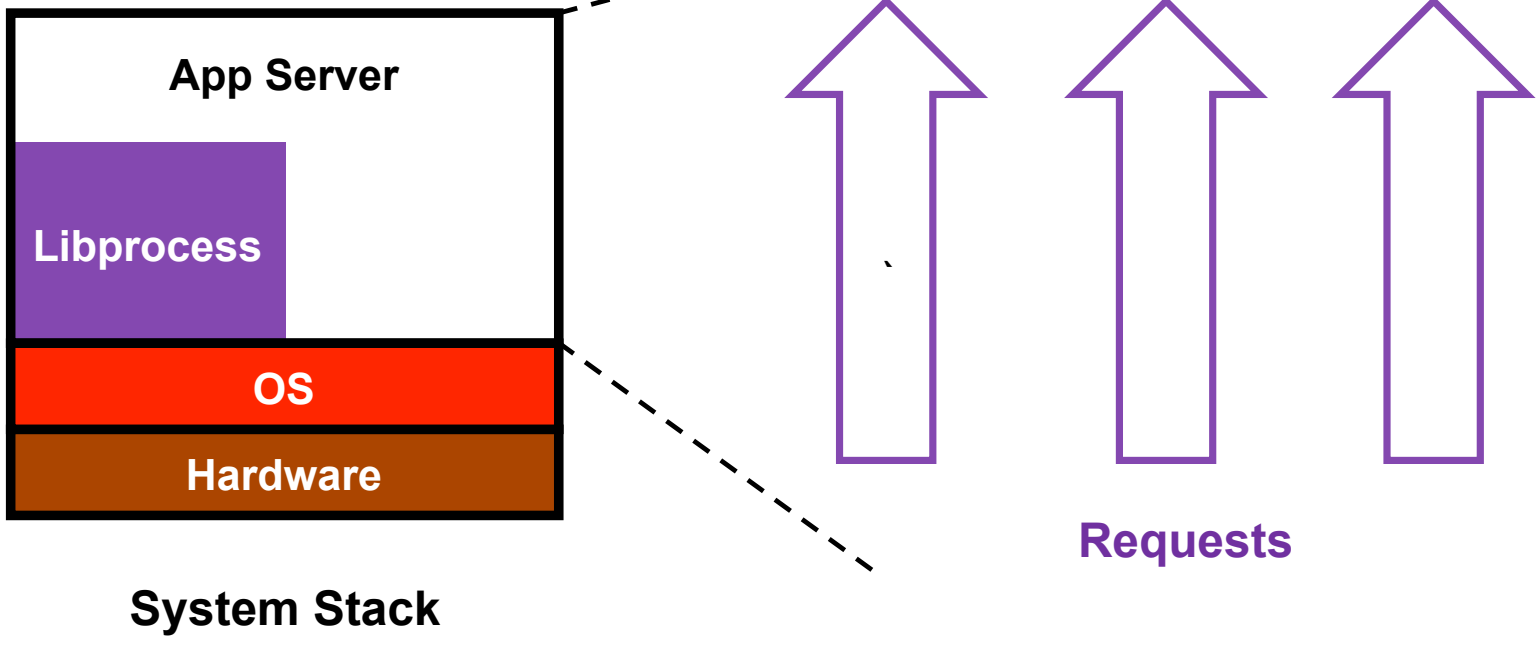




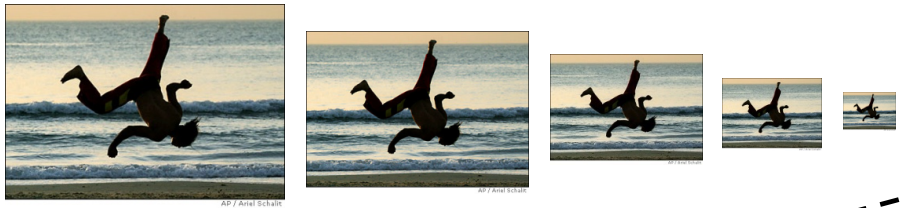

System Stack

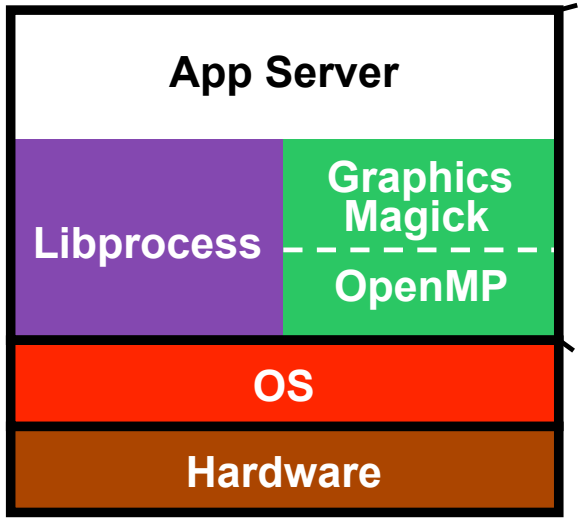
# Flickr-Like Image Processing App Server





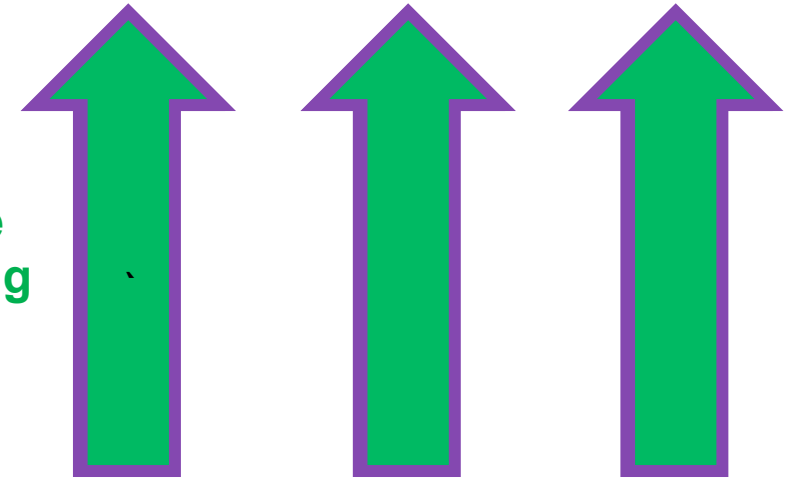
# Flickr-Like Image Processing App Server





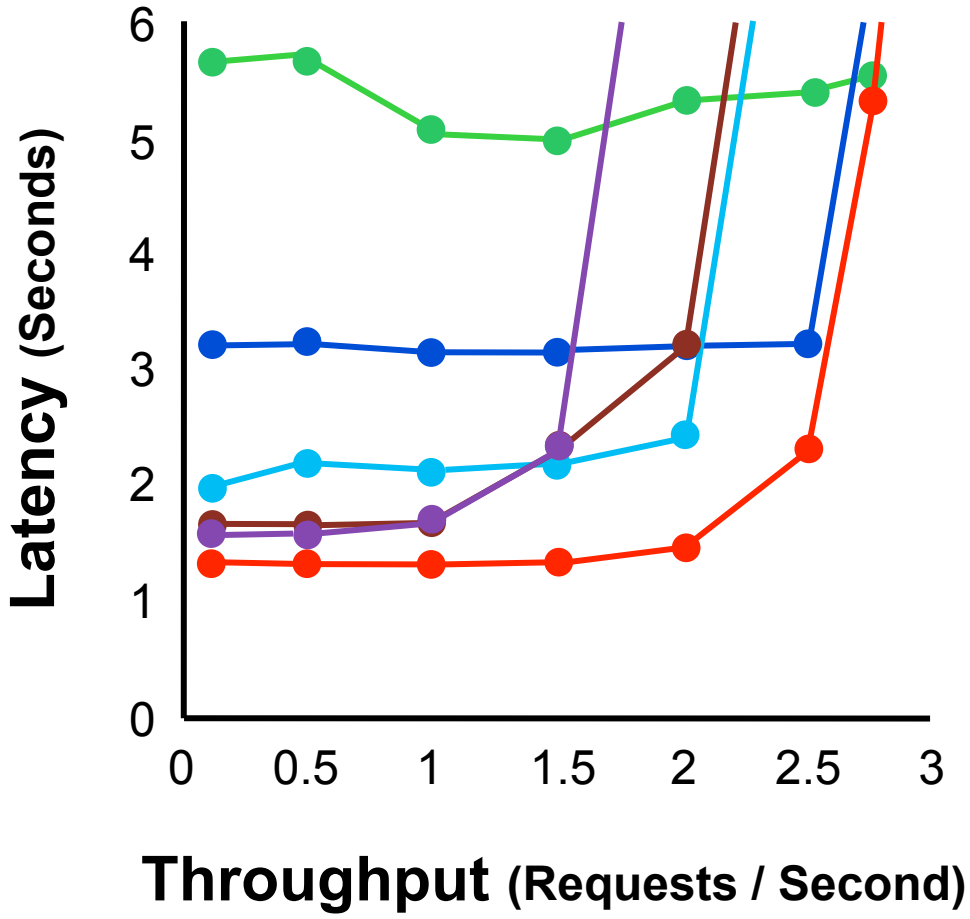

System Stack

Image Resizing



Requests

# Performance of App Server




(16-Core Machine)

# OMP Threads = 1

# OMP Threads = 2

# OMP Threads = 4

# OMP Threads = 8

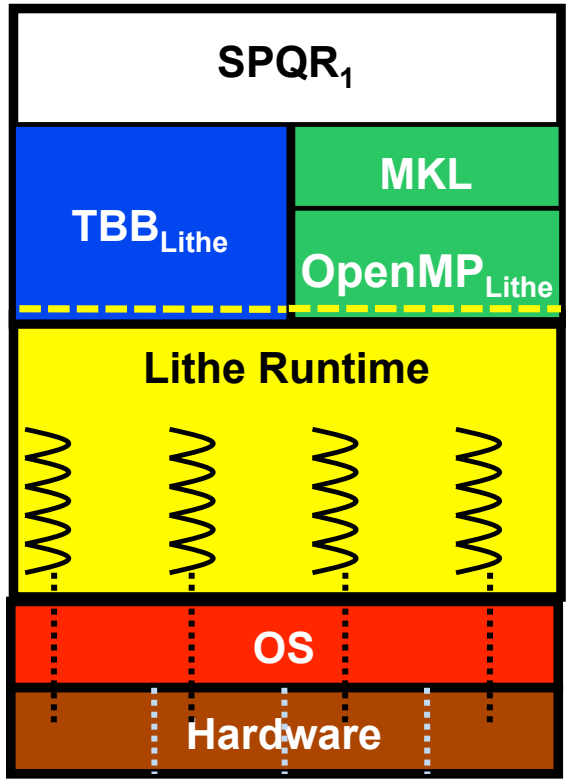
# OMP Threads = 16

Lithe

# Future Directions

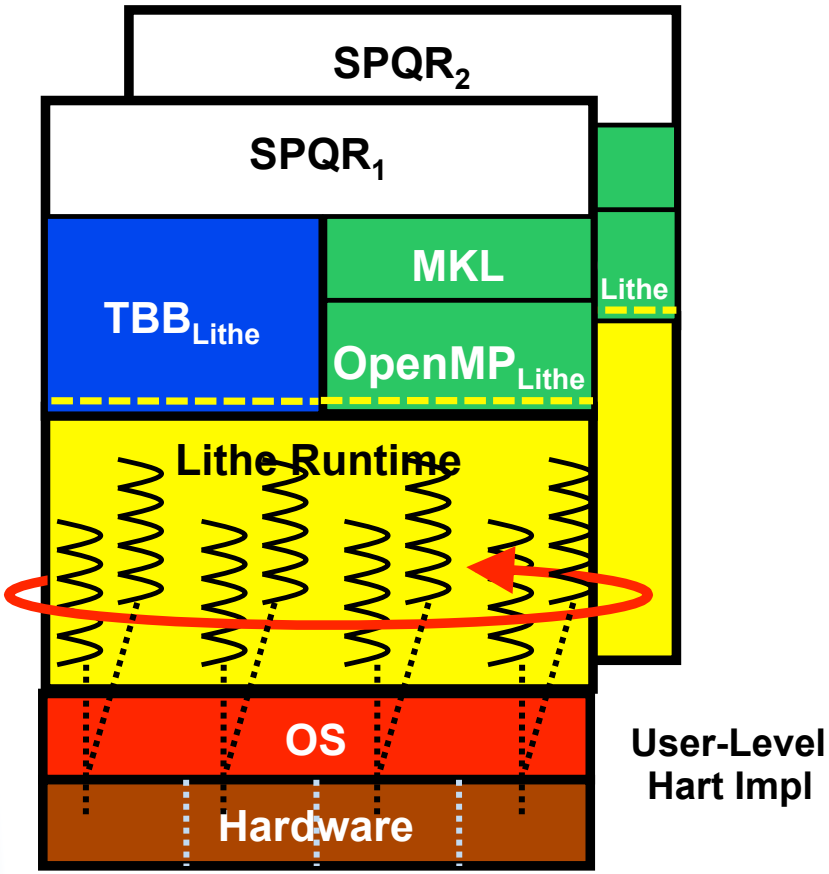
- OS Support for Lithe
  - Akaros, Tessellation
- Preemptive Version of Lithe
  - Direct support for MPI
  - Integrate with GASNet
    - Leverage lithe contexts
- Other ways to integrate with the DEGAS stack?

# OS Support for Lithe

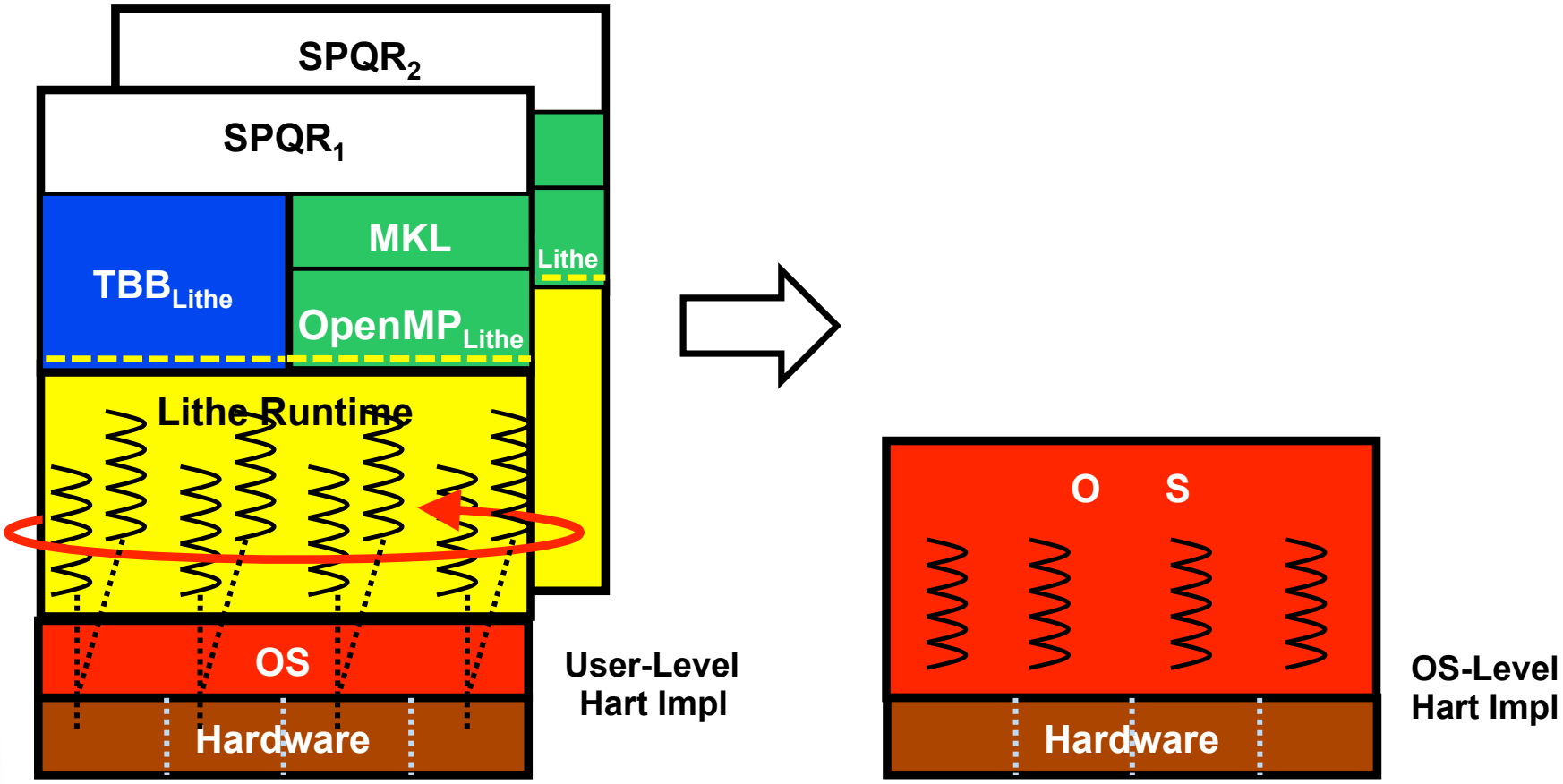



User-Level
Hart Impl

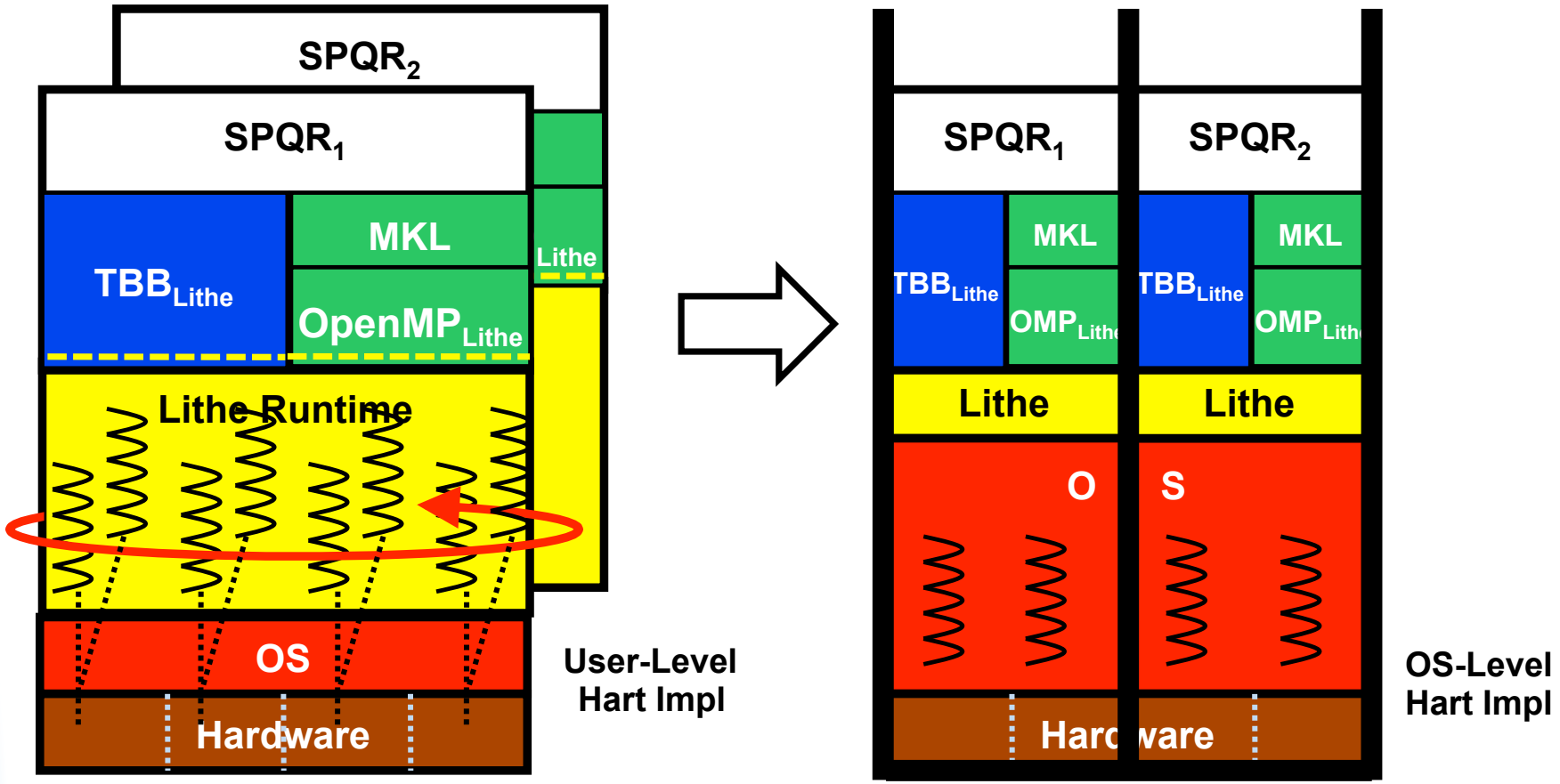
# OS Support for Lithe



# OS Support for Lithe

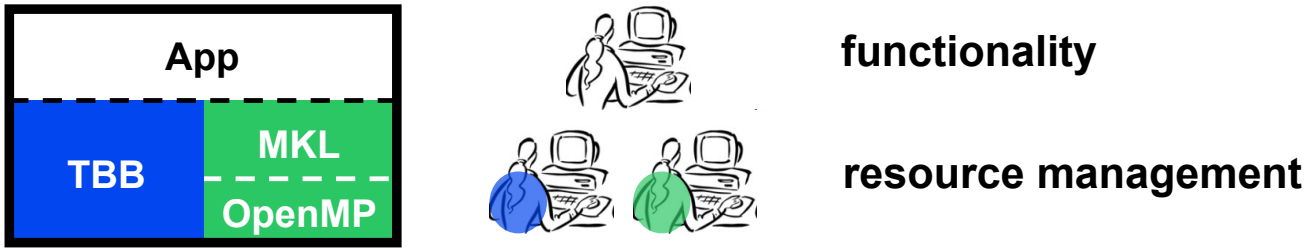


# OS Support for Lithe

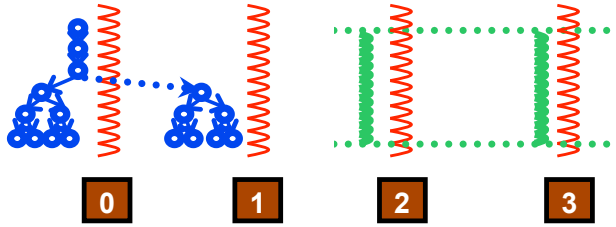


# Conclusion

- Composability essential for parallel programming to become widely adopted

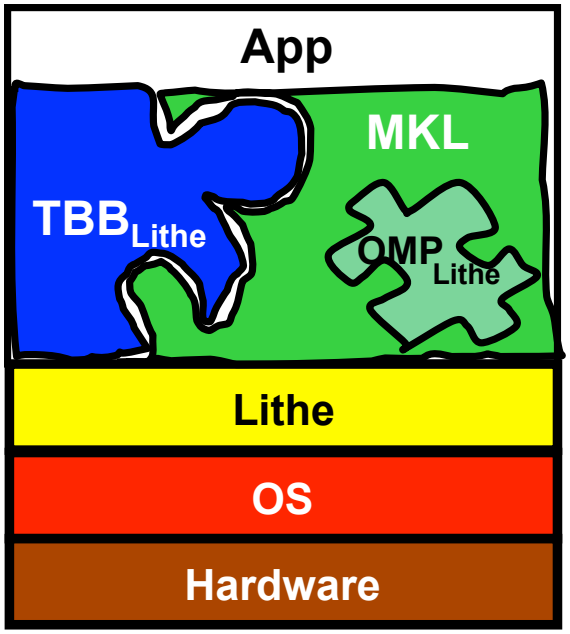



- Parallel libraries need to share resources cooperatively




- Main Contributions
  - **Harts:** better resource model for parallel programming
  - **Lithe:** framework for using and sharing harts

# Questions?




http://lithe.eecs.berkeley.edu

# Preemptive Lithe

- Three-level priority scheme
  - Same Priority → Cooperative
  - Higher priority can only preempt lower
  - Developer sets runtime priorities
  - Lithe runtime enforces priorities
- Current Lithe == Single Level