Kripke: OCCA Port

www.libocca.org

David S. Medina

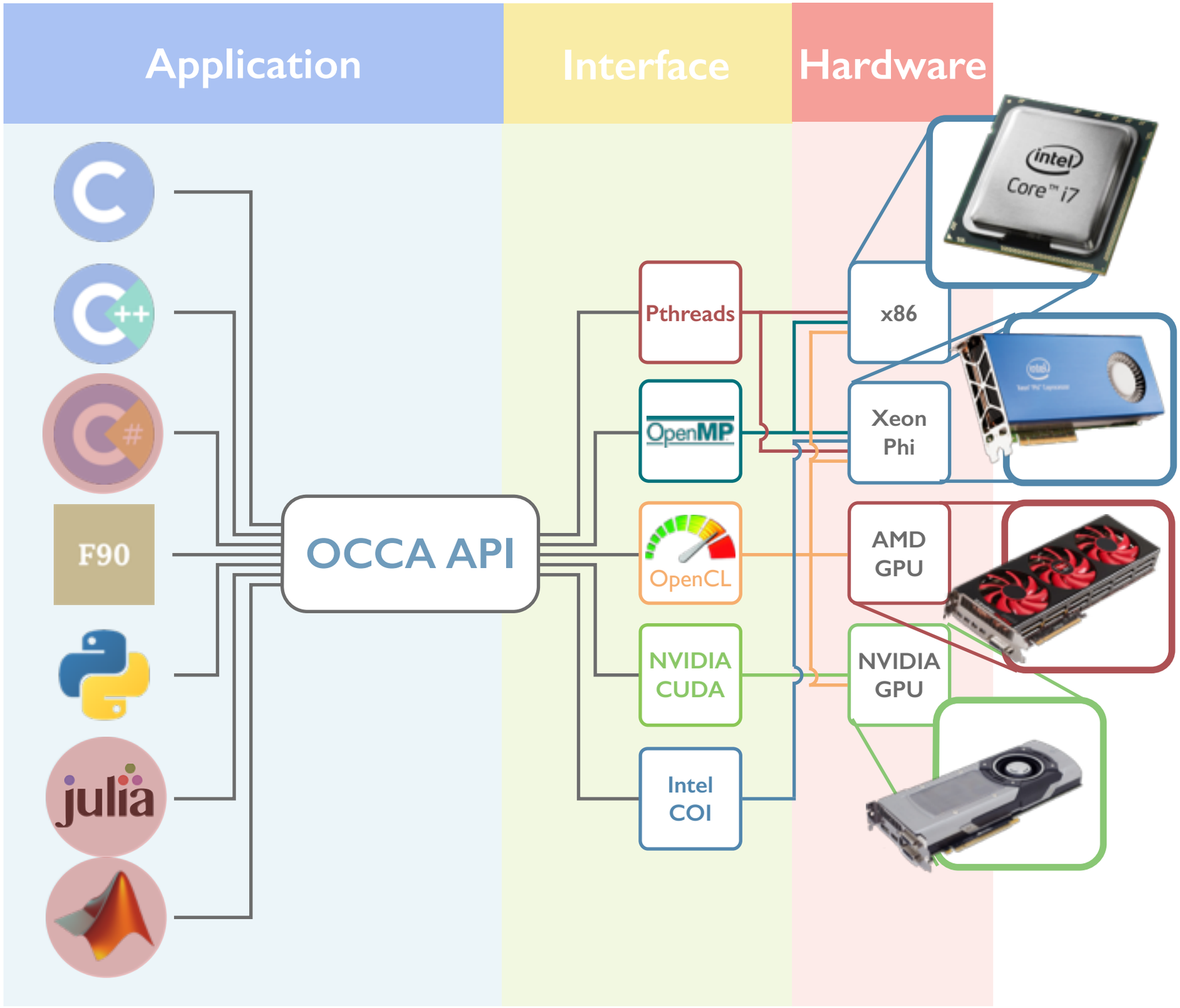
Advisor: Prof. Tim Warburton

LLNL Mentors: Jeff Keasler, Adam Kunen

This work was performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under Contract DE-AC52-07NA27344.
Lawrence Livermore National Security, LLC

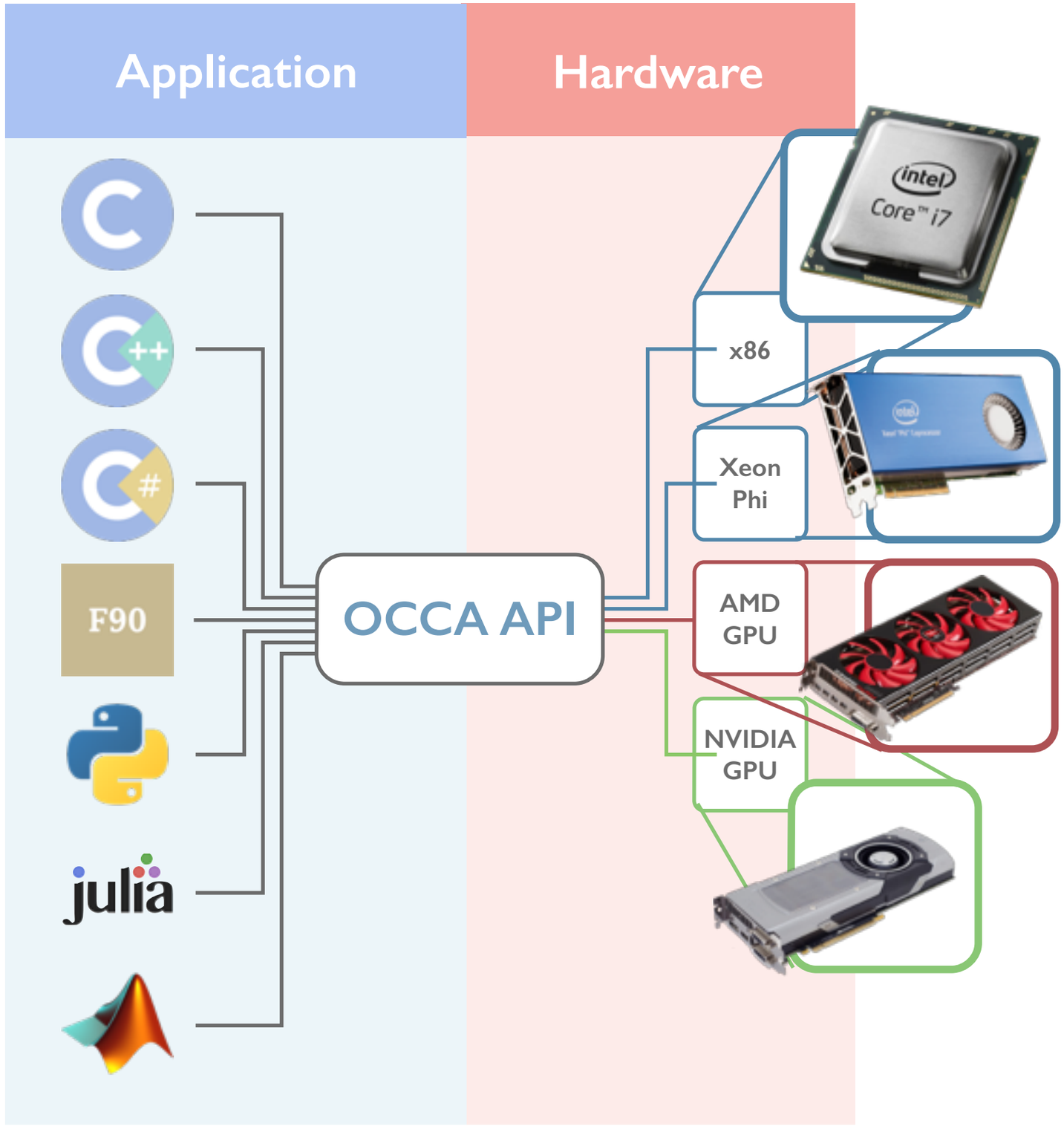
LLNL-PRES-675166

Summary



Many front-ends (more coming) together with multiple back-ends

Summary



Many front-ends (more coming) together with multiple back-ends

Summary: Approaches

Some Portability Approaches

API	Type	Front-ends	Kernel Language	Back-ends
Kokkos	ND Arrays	C++11	Templates	CUDA, OpenMP
OCCA	API + DSL + source-to-source	C, C++, Fortran, Python + others	Custom Kernel Language	Pthreads, OMP, OCL, CUDA, COI
RAJA	Lambdas	C++11	Loop-Macros	OMP, CUDA, ACC
CU2CL	Source-to-Source	Stand-alone	CUDA	OpenCL
Insieme	Intermediate Interpretation	C	OpenMP, Cilk, MPI, OpenCL	OpenCL, MPI, Insieme Runtime
OmpSs	Directives + Kernels	C, C++	Hybrid OpenMP, OpenCL, CUDA	OpenMP, OpenCL, CUDA
Ocelot	PTX Translator	CUDA	CUDA	CUDA, OpenCL, x86
OpenMP 4.0	#pragma	C++11	#pragma	OpenMP/CUDA
OpenACC	#pragma	C++11	#pragma	OpenMP/CUDA

API

Compiler

Summary: Application Snapshot

```
#include "occa.hpp"

int main(int argc, char
  float *a  = new float
  float *b  = new float
  float *ab = new float

  for(int i = 0; i < 5
    a[i]  = i;
    b[i]  = 1 - i;
    ab[i] = 0;
  }

  occa::device device;
  occa::kernel addVectol
  occa::memory o_a, o_b, o_ab;

  device.setup("mode = OpenCL, platformID = 0, deviceID = 0");

  o_a  = device.malloc(5*sizeof(float));
  o_b  = device.malloc(5*sizeof(float));
  o_ab = device.malloc(5*sizeof(float));

  o_a.copyFrom(a);
  o_b.copyFrom(b);

  addVectors = device.buildKernelFromSource("addVectors.okl",
                                            "addVectors");

  addVectors(5, o_a, o_b, o_ab);

  o_ab.copyTo(ab);

  for(int i = 0; i < 5; ++i)
    std::cout << i << ": " << ab[i] << '\n';
```

```
kernel void addVectors(const int entries,
                       const float *a,
                       const float *b,
                       float *ab){

  for(int i = 0; i < entries; ++i; tile(16)){
    if(i < entries)
      ab[i] = a[i] + b[i];
  }
}
```

OKL

https://github.com/libocca/occa/tree/master/examples/addVectors

Summary: Application Snapshot

```
#include "occa.hpp"

int main(int argc, char
  float *a  = new float
  float *b  = new float
  float *ab = new float

  for(int i = 0; i < 5
    a[i]  = i;
    b[i]  = 1 - i;
    ab[i] = 0;
  }

  occa::device device;
  occa::kernel addVector
  occa::memory o_a, o_b, o_ab;

  device.setup("mode = OpenCL, platformID = 0, deviceID = 0");

  o_a  = device.malloc(5*sizeof(float));
  o_b  = device.malloc(5*sizeof(float));
  o_ab = device.malloc(5*sizeof(float));

  o_a.copyFrom(a);
  o_b.copyFrom(b);

  addVectors = device.buildKernelFromSource("addVectors.okl",
                                            "addVectors");

  addVectors(5, o_a, o_b, o_ab);

  o_ab.copyTo(ab);

  for(int i = 0; i < 5; ++i)
    std::cout << i << ": " << ab[i] << '\n';
```

```
kernel void addVectors(const int entries,
                       const float *a,
                       const float *b,
                       float *ab){

  for(int i = 0; i < entries; ++i; tile(16)){
    if(i < entries)
      ab[i] = a[i] + b[i];
  }
}
```

OKL

https://github.com/libocca/occa/tree/master/examples/addVectors

Summary: Application Snapshot

```
#include "occa.hpp"

int main(int argc, char
  float *a  = new float
  float *b  = new float
  float *ab = new float

  for(int i = 0; i < 5
    a[i]  = i;
    b[i]  = 1 - i;
    ab[i] = 0;
  }

  occa::device device;
  occa::kernel addVector
  occa::memory o_a, o_b, o_ab;

  device.setup("mode = OpenCL, platformID = 0, deviceID = 0");

  o_a  = device.malloc(5*sizeof(float));
  o_b  = device.malloc(5*sizeof(float));
  o_ab = device.malloc(5*sizeof(float));

  o_a.copyFrom(a);
  o_b.copyFrom(b);

  addVectors = device.buildKernelFromSource("addVectors.okl",
                                            "addVectors");

  addVectors(5, o_a, o_b, o_ab);

  o_ab.copyTo(ab);

  for(int i = 0; i < 5; ++i)
    std::cout << i << ": " << ab[i] << '\n';
```

```
kernel void addVectors(const int entries,
                       const float *a,
                       const float *b,
                       float *ab){

  for(int i = 0; i < entries; ++i; tile(16)){
    if(i < entries)
      ab[i] = a[i] + b[i];
  }
}
```

OKL

https://github.com/libocca/occa/tree/master/examples/addVectors

Summary: Application Snapshot

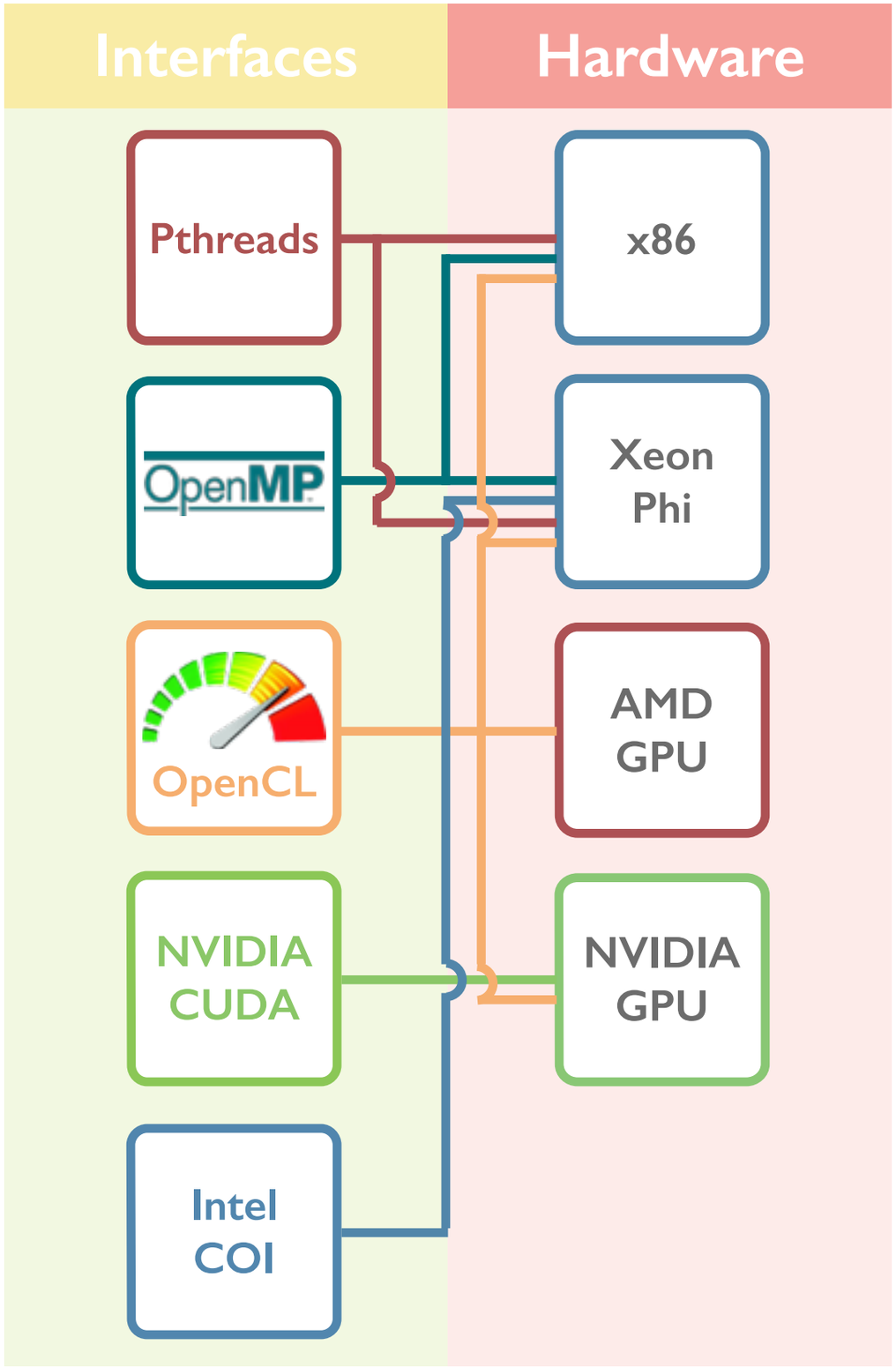
```
#include "occa.hpp"

int main(int argc, char
  float *a  = new float
  float *b  = new float
  float *ab = new float

  for(int i = 0; i < 5
    a[i]  = i;
    b[i]  = 1 - i;
    ab[i] = 0;
  }

  occa::device device;
  occa::kernel addVector
  occa::memory o_a, o_b, o_ab;

  device.setup("mode = OpenCL, platformID = 0, deviceID = 0");

  o_a  = device.malloc(5*sizeof(float));
  o_b  = device.malloc(5*sizeof(float));
  o_ab = device.malloc(5*sizeof(float));

  o_a.copyFrom(a);
  o_b.copyFrom(b);

  addVectors = device.buildKernelFromSource("addVectors.okl",
                                            "addVectors");

  addVectors(5, o_a, o_b, o_ab);

  o_ab.copyTo(ab);

  for(int i = 0; i < 5; ++i)
    std::cout << i << ": " << ab[i] << '\n';
```

```
kernel void addVectors(const int entries,
                       const float *a,
                       const float *b,
                       float *ab){

  for(int i = 0; i < entries; ++i; tile(16)){
    if(i < entries)
      ab[i] = a[i] + b[i];
  }
}
```

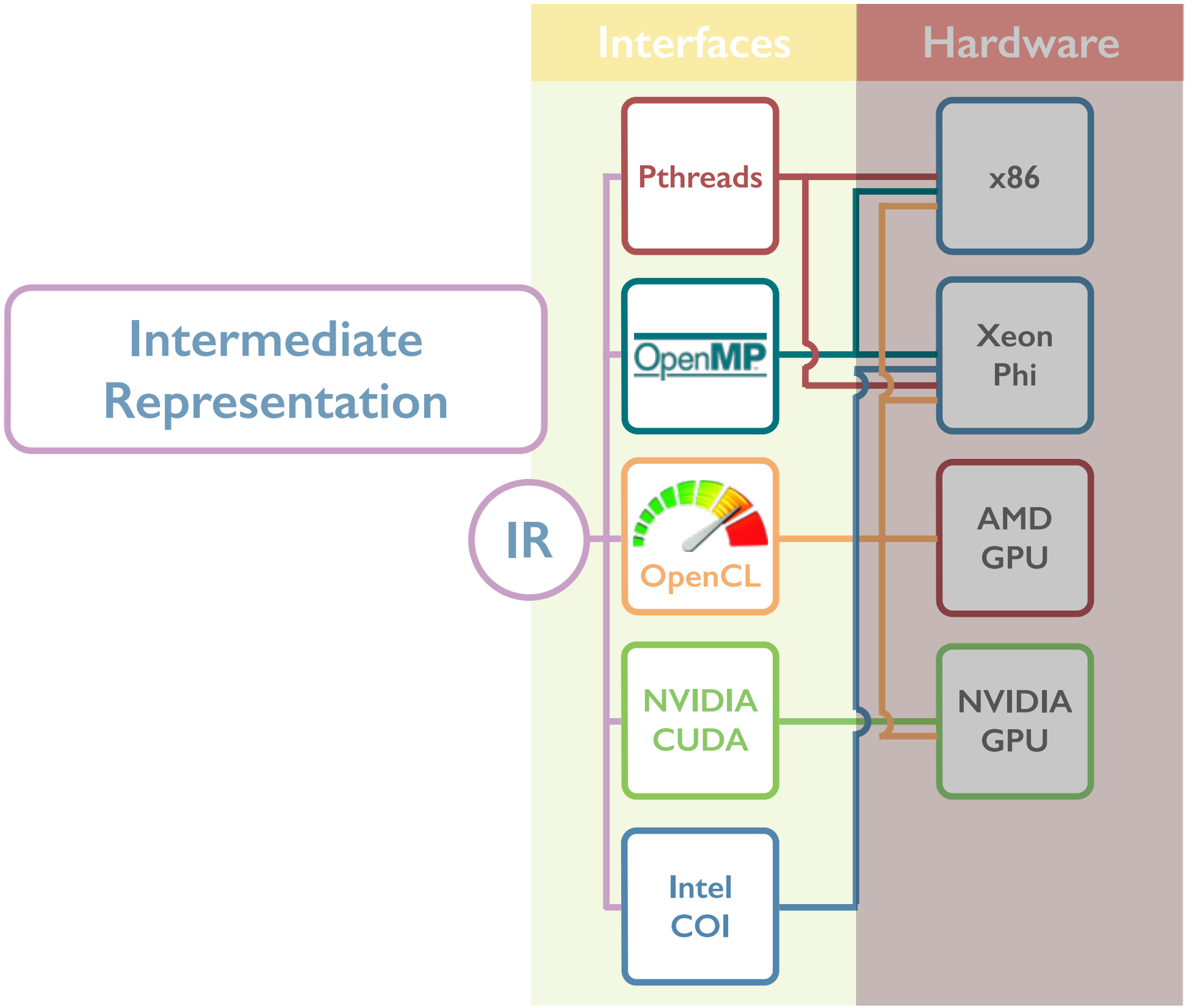
OKL

https://github.com/libocca/occa/tree/master/examples/addVectors

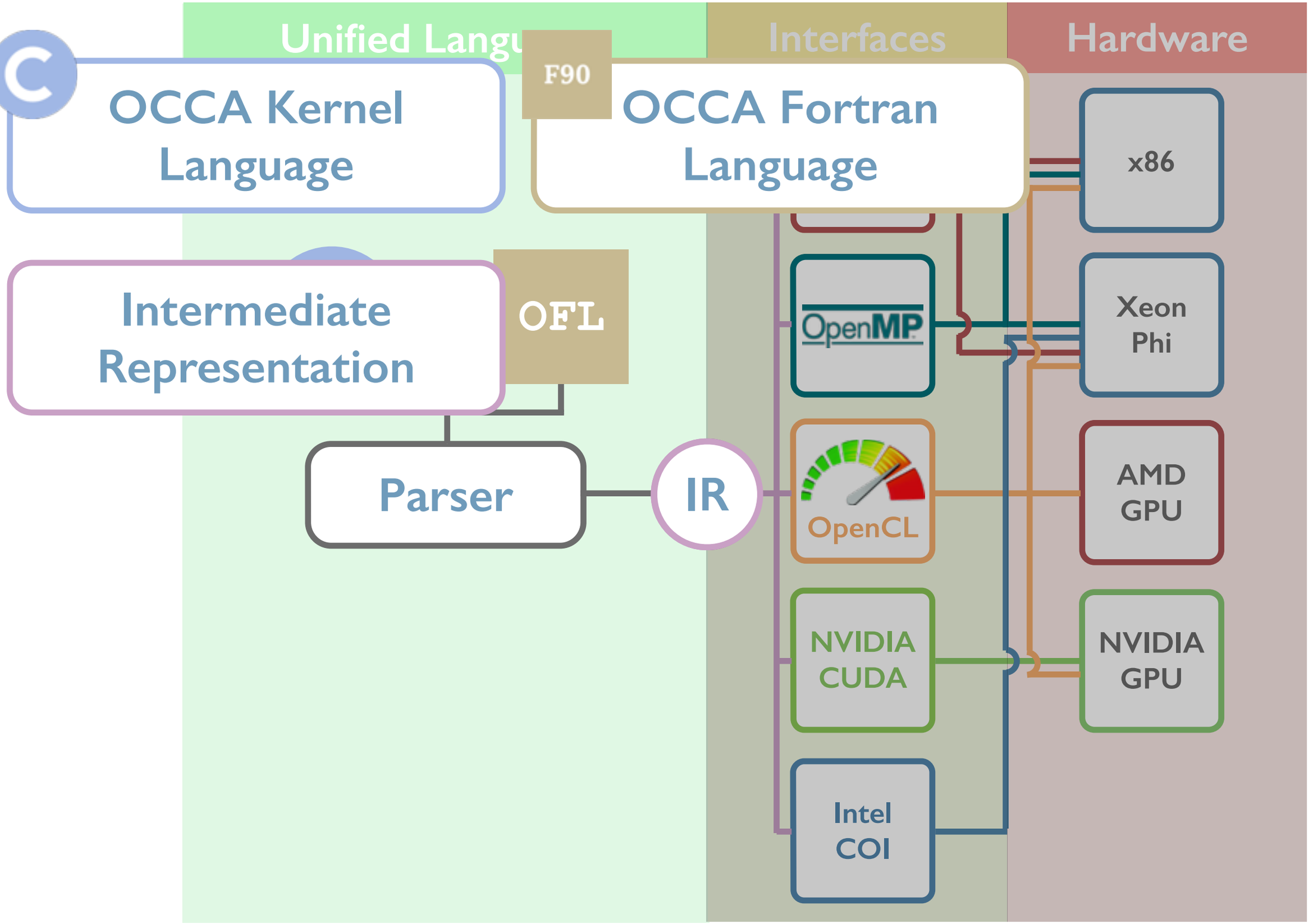
OCCA Overview



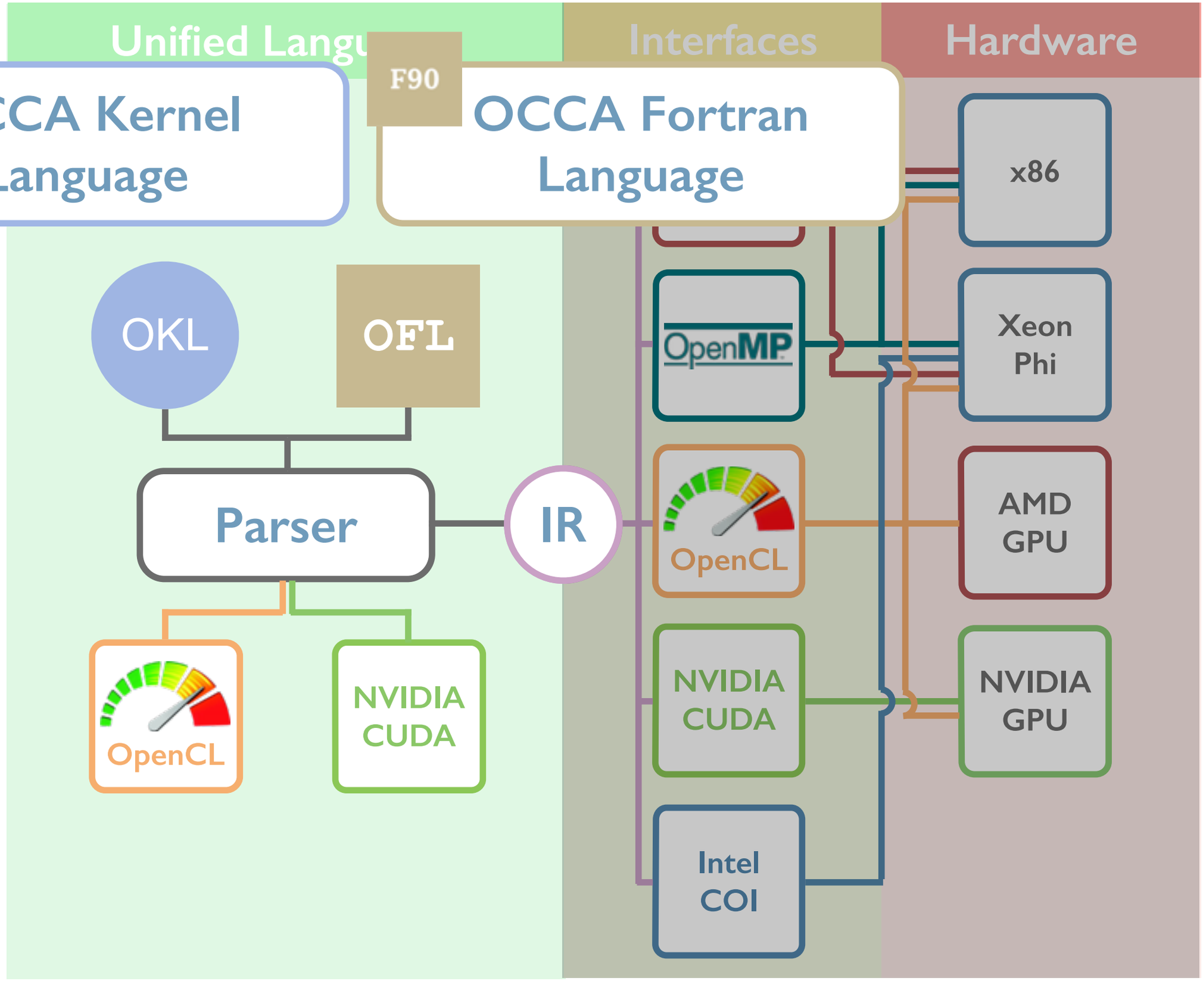
OCCA Overview



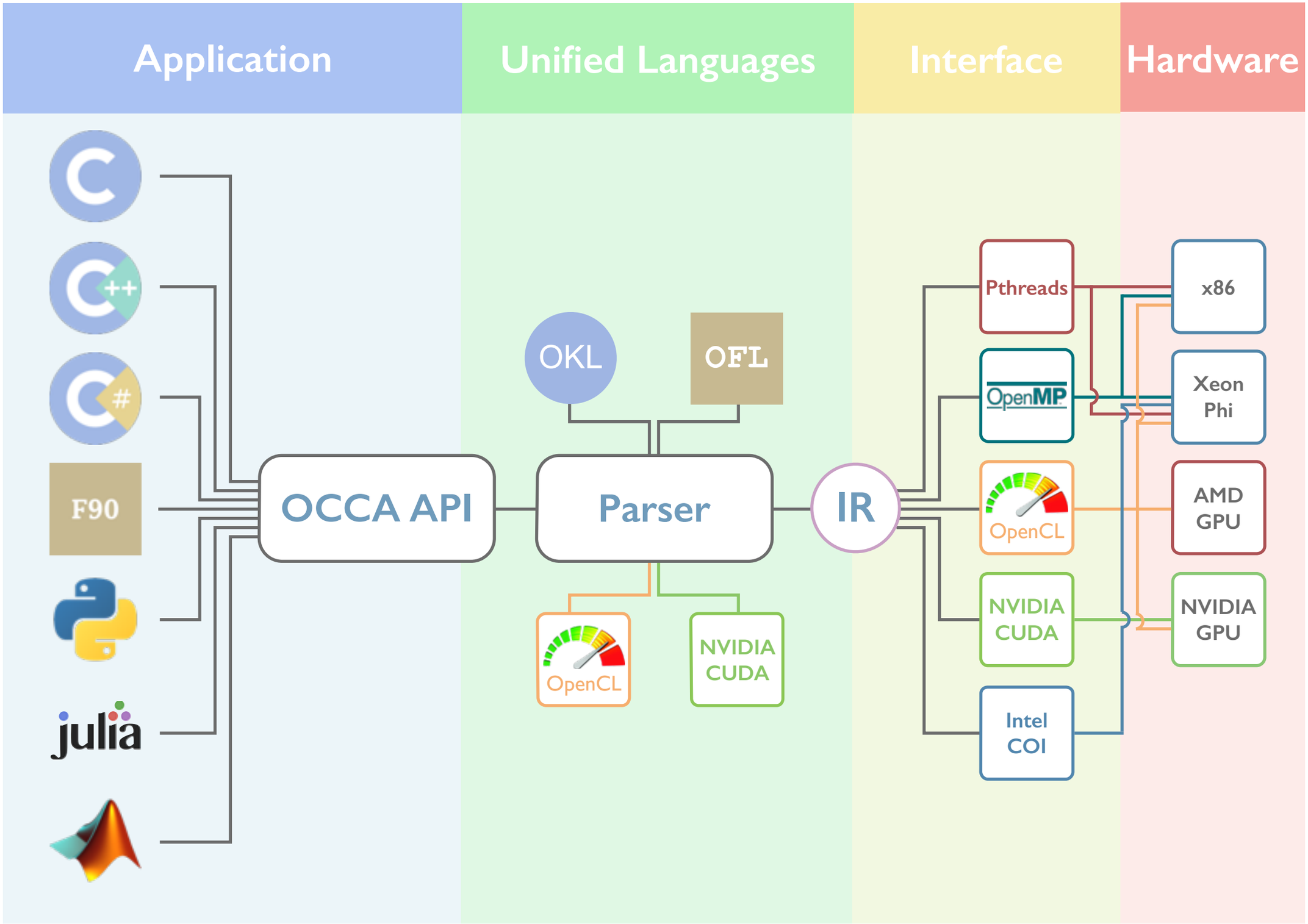
OCCA Overview



OCCA Overview

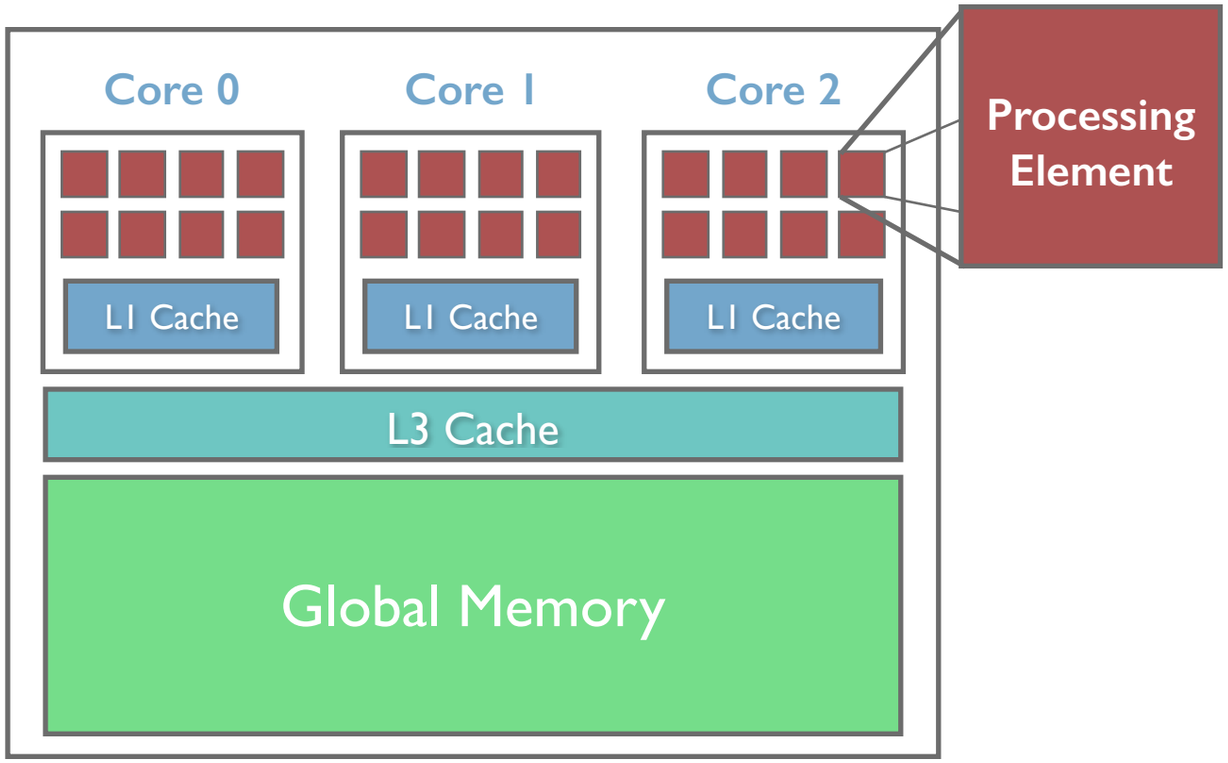


OCCA Overview

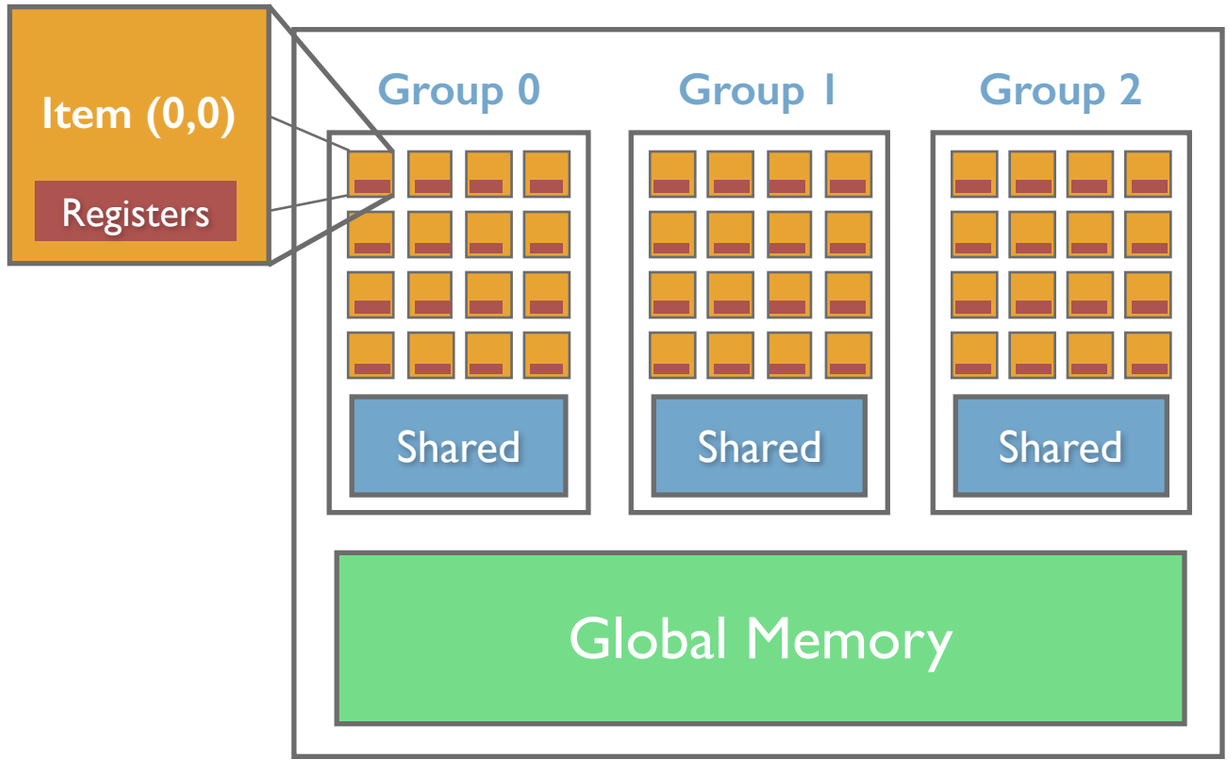


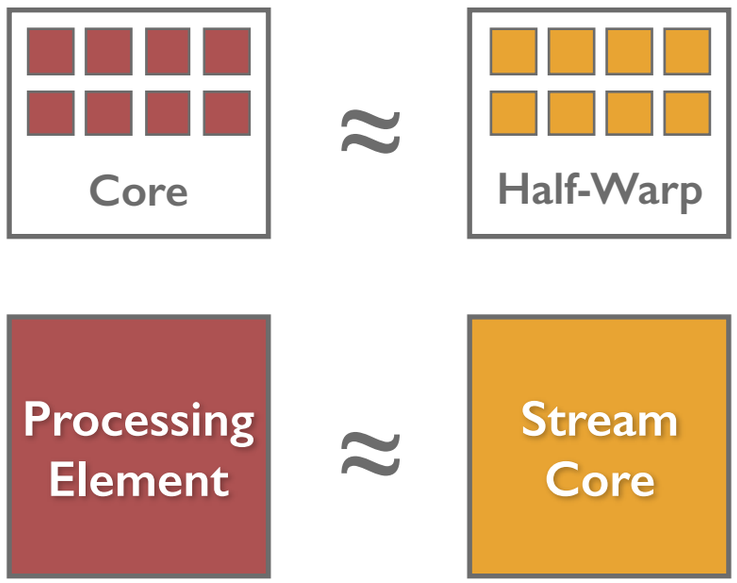
Parallelization Paradigm

CPU Architecture



GPU Architecture

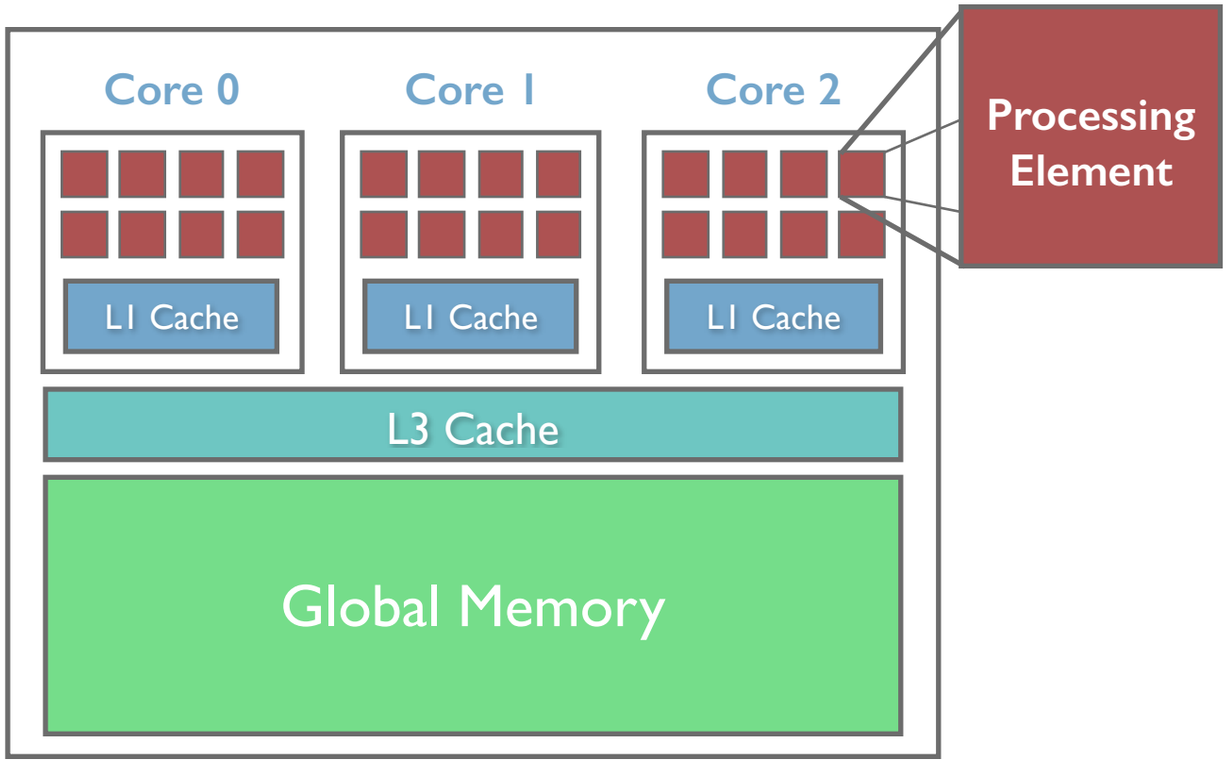




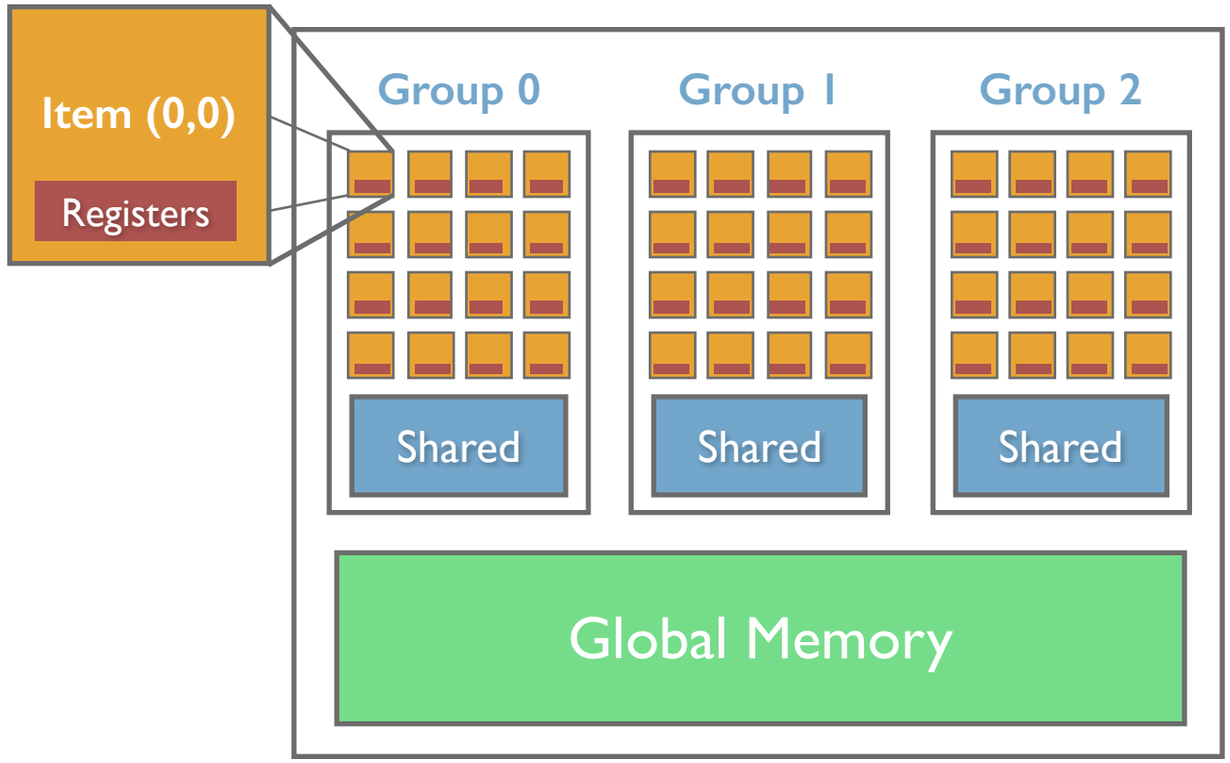
Computational hierarchies are similar

Parallelization Paradigm

CPU Architecture



GPU Architecture



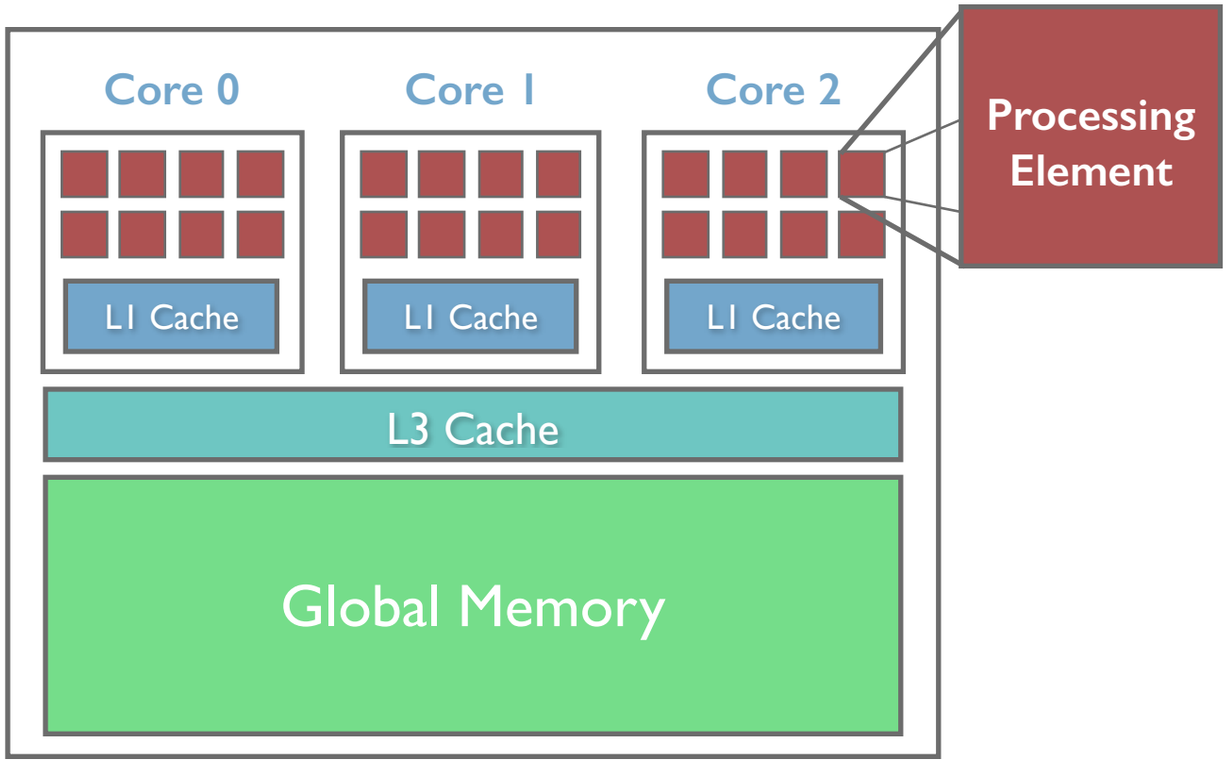

```
void cpuFunction(){
  #pragma omp parallel for
  for(int i = 0; i < work; ++i){

    Do [hopefully thread-independent] work

  }

}
```

```
__kernel void gpuFunction(){
// for each work-group {
//   for each work-item in group {

  Do [group-independent] work

//   }
// }
}
```


Computational hierarchies are similar

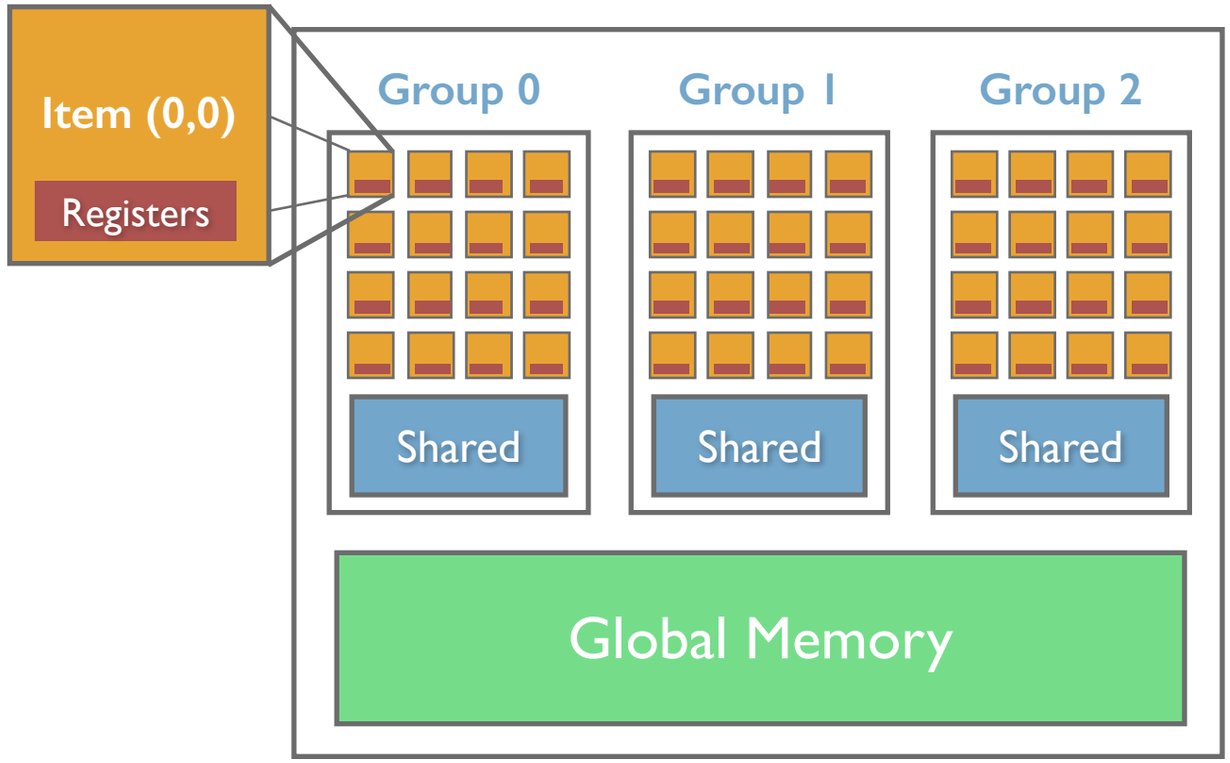
Parallelization Paradigm

CPU Architecture




```
void ompFunction(){
// for each thread {
  for(thread's work){

    Do [hopefully thread-independent] work

  }
// }
}
```

GPU Architecture




```
__kernel void gpuFunction(){
// for each work-group {
//   for each work-item in group {

  Do [group-independent] work

//   }
// }
}
```

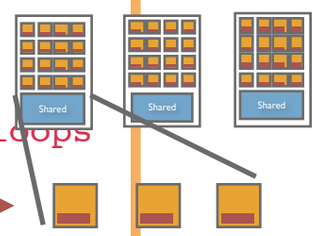

Computational hierarchies are similar

OKL: OCCA Kernel Language

Description

- Minimal extensions to C, familiar for regular programmers
- Translates to OCCA IR with code transformations
- Parallel loops are explicit through the fourth for-loop inner and outer labels

```
kernel void kernelName(...){
  ...

  for(int groupZ = 0; groupZ < zGroups; ++groupZ; outer2){
    for(int groupY = 0; groupY < yGroups; ++groupY; outer1){
      for(int groupX = 0; groupX < xGroups; ++groupX; outer0){     // Work-group implicit loops
  for(outer){
    for(inner){ for(int itemZ = 0; itemZ < zItems; ++itemZ; inner2){
    }      for(int itemY = 0; itemY < yItems; ++itemY; inner1){
  }          for(int itemX = 0; itemX < xItems; ++itemX; inner0){   // Work-item implicit loops
             // GPU Kernel Scope
         }}}
  }}}

  ...
}
```



NVIDIA CUDA

```
dim3 blockDim(xGroups,yGroups,zGroups);
dim3 threadDim(xItems,yItems,zItems);
kernelName<<< blockDim , threadDim >>>(…);
```

OKL

The concept of iterating over groups and items is simple

OKL: OCCA Kernel Language

Multiple outer-loops

- Sets of outer-loops are synonymous with CUDA and OpenCL kernels
- Extension: allow for multiple outer-loops per kernel

```
kernel void kernelName(...){
  for(outer){
    for(inner){
    }
  }

  for(outer){
    for(inner){
    }
  }

  for(outer){
    for(inner){
    }
  }
}
```

OKL

Execute a set of outer-loops is equivalent to launching a CUDA/OpenCL kernel

OKL: OCCA Kernel Language

Multiple outer-loops

- Sets of outer-loops are synonymous with CUDA and OpenCL kernels
- Extension: allow for multiple outer-loops per kernel

```
kernel void kernelName(...){
  ...

  for(outer){
    for(inner){
    }
  }

  ...
}
```

OKL

Execute a set of outer-loops is equivalent to launching a CUDA/OpenCL kernel

OKL: OCCA Kernel Language

Multiple outer-loops

- Sets of outer-loops are synonymous with CUDA and OpenCL kernels
- Extension: allow for multiple outer-loops per kernel

```
kernel void kernelName(…){

  if(expr){
    for(outer){
      for(inner){
      }
    }
  else{
    for(outer){
      for(inner){
      }
    }
  }

  while(expr){
    for(outer){
      for(inner){
      }
    }
  }

}
```

OKL

Execute a set of outer-loops is equivalent to launching a CUDA/OpenCL kernel

OKL: OCCA Kernel Language

Shared Memory

```
for(int groupX = 0; groupX < xGroups; ++groupX; outer0){  // Work-group implicit loops
  shared int sharedVar[16];

  for(int itemX = 0; itemX < 16; ++ itemX; inner0){  // Work-item implicit loops
    sharedVar[itemX] = itemX;
  }

  // Auto-insert [barrier(localMemFence);]

  for(int itemX = 0; itemX < 16; ++ itemX; inner0){  // Work-item implicit loops
    int i = (sharedVar[itemX] + sharedVar[(itemX + 1) % 16]);
  }
}
```

OKL

Register Memory (SIMD Variables)

```
for(int groupX = 0; groupX < xGroups; ++groupX; outer0){  // Work-group implicit loops
  exclusive int exclusiveVar, exclusiveArray[10];

  for(int itemX = 0; itemX < 16; ++ itemX; inner0){  // Work-item implicit loops
    exclusiveVar = itemX;  // Pre-fetch
  }

  // Auto-insert [barrier(localMemFence);]

  for(int itemX = 0; itemX < 16; ++ itemX; inner0){  // Work-item implicit loops
    int i = exclusiveVar;  // Use pre-fetched data
  }
}
```

OKL

Local barriers are auto-inserted (gives a warning)

OKL: OCCA Kernel Language

Shared Memory

```
for(int groupX = 0; groupX < xGroups; ++groupX; outer0){  // Work-group implicit loops
  shared int sharedVar[16];

   for(int itemX = 0; itemX < 16; ++ itemX; inner0){  // Work-item implicit loops
     sharedVar[itemX] = itemX;
   }

   // Auto-insert [barrier(localMemFence);]

   for(int itemX = 0; itemX < 16; ++ itemX; inner0){  // Work-item implicit loops
     int i = (sharedVar[itemX] + sharedVar[(itemX + 1) % 16]);
   }
}
```

OKL

Register Memory (SIMD Variables)

```
for(int groupX = 0; groupX < xGroups; ++groupX; outer0){  // Work-group implicit loops
  exclusive int exclusiveVar, exclusiveArray[10];

   for(int itemX = 0; itemX < 16; ++ itemX; inner0){  // Work-item implicit loops
     exclusiveVar = itemX;  // Pre-fetch
   }

   // Auto-insert [barrier(localMemFence);]

   for(int itemX = 0; itemX < 16; ++ itemX; inner0){  // Work-item implicit loops
     int i = exclusiveVar;  // Use pre-fetched data
   }
}
```

OKL

Local barriers are auto-inserted (gives a warning)

OFL: OCCA Fortran Language

Description

- Translates to OKL and then to OCCA IR with code transformations
- Parallel loops are explicit through the inner and outer DO-labels

```
kernel subroutine kernelName(...)
  ...

  DO groupY = 1, yGroups, outer1
   DO groupX = 1, xGroups, outer0 // Work-group implicit loops
    DO itemY = 1, yItems, inner1
     DO itemX = 1, xItems, inner0 // Work-item implicit loops
      // GPU Kernel Scope
     END DO
    END DO
   END DO
  END DO

  ...
end subroutine kernelName
```

OFL

Shared and Exclusive Memory

```
integer(4), shared    :: sharedVar(16,30)
integer(4), exclusive :: exclusiveVar, exclusiveArray(10)
```

OFL

Because [OFL -> OKL], all features added to OKL are inherintly added to OFL

OCCA API: Original

```
#include "occa.hpp"

int main(int argc, char **argv){
  occa::device device;
  occa::kernel addVectors;
  occa::memory o_a, o_b, o_ab;

  device.setup("mode = OpenCL, platformID = 0, deviceID = 0");

  float *a  = new float[5];
  float *b  = new float[5];
  float *ab = new float[5];

  for(int i = 0; i < 5; ++i){
    a[i]  = i;
    b[i]  = 1 - i;
    ab[i] = 0;
  }

  o_a  = device.malloc(5*sizeof(float));
  o_b  = device.malloc(5*sizeof(float));
  o_ab = device.malloc(5*sizeof(float));

  o_a.copyFrom(a);
  o_b.copyFrom(b);

  addVectors = device.buildKernel("addVectors.okl",
                                  "addVectors");

  addVectors(5, o_a, o_b, o_ab);

  o_ab.copyTo(ab);

  for(int i = 0; i < 5; ++i)
    std::cout << i << ": " << ab[i] << '\n';
```


https://github.com/libocca/occa/tree/master/examples/addVectors

OCCA API: Original + UVA + Managed Memory

```
#include "occa.hpp"

int main(int argc, char **argv){
  occa::device device;
  occa::kernel addVectors;
  occa::memory o_a, o_b, o_ab;

  device.setup("mode = OpenCL, platformID = 0, deviceID = 0");

  float *a  = (float*) device.managedAlloc(5 * sizeof(float));
  float *b  = (float*) device.managedAlloc(5 * sizeof(float));
  float *ab = (float*) device.managedAlloc(5 * sizeof(float));

  for(int i = 0; i < 5; ++i){
    a[i]  = i;
    b[i]  = 1 - i;
    ab[i] = 0;
  }

  o_a  = device.malloc(5*sizeof(float));
  o_b  = device.malloc(5*sizeof(float));
  o_ab = device.malloc(5*sizeof(float));

  o_a.copyFrom(a);
  o_b.copyFrom(b);

  addVectors = device.buildKernel("addVectors.okl",
                                  "addVectors");

  addVectors(5, a, b, ab);

  device::finish(); // occa::memcpy(ab, o_ab, 5*sizeof(float));

  for(int i = 0; i < 5; ++i)
    std::cout << i << ": " << ab[i] << '\n';
```


https://github.com/libocca/occa/tree/master/examples/uvaAddVectors

OCCA API: Original + Managed + BG Device

```
#include "occa.hpp"

int main(int argc, char **argv){
  occa::device device;
  occa::kernel addVectors;
  occa::memory o_a, o_b, o_ab;

  occa::setDevice("mode = OpenCL, platformID = 0, deviceID = 0");

  float *a  = (float*) occa::managedAlloc(5 * sizeof(float));
  float *b  = (float*) occa::managedAlloc(5 * sizeof(float));
  float *ab = (float*) occa::managedAlloc(5 * sizeof(float));

  for(int i = 0; i < 5; ++i){
    a[i]  = i;
    b[i]  = 1 - i;
    ab[i] = 0;
  }

  o_a  = (float*) device.uvaAlloc(5*sizeof(float));
  o_b  = (float*) device.uvaAlloc(5*sizeof(float));
  o_ab = (float*) device.uvaAlloc(5*sizeof(float));

  occa::memcpy(o_a, a, 5*sizeof(float));
  occa::memcpy(o_b, b, 5*sizeof(float));

  addVectors = occa::buildKernel("addVectors.okl",
                                 "addVectors");

  addVectors(5, a, b, ab);

  occa::finish() // occa::memcpy(ab, o_ab, 5*sizeof(float));

  for(int i = 0; i < 5; ++i)
    std::cout << i << ": " << ab[i] << '\n';
```


https://github.com/libocca/occa/tree/master/examples/uvaAddVectors

Kripke Port

LLNL-PRES-675166

Kripke: Goals

Loop Reordering (Run-Time)

- Simplify loop re-ordering for testing
- Avoid re-writing code (less code, less bugs)

```
for(g){
  for(d){
    for(z){
      // Work ...
    }
  }
}
```

```
for(g){
  for(z){
    for(d){
      // Work ...
    }
  }
}
```

```
for(d){
  for(g){
    for(z){
      // Work ...
    }
  }
}
```

```
for(d){
  for(z){
    for(g){
      // Work ...
    }
  }
}
```

```
for(z){
  for(d){
    for(g){
      // Work ...
    }
  }
}
```

```
for(z){
  for(g){
    for(d){
      // Work ...
    }
  }
}
```

```
for(g;      loopOrder(lo_g)){
  for(d;    loopOrder(lo_d)){
    for(z;  loopOrder(lo_z)){
      // Work ...
    }
  }
}
```

Kripke: Goals

Index Ordering (Run-Time)

- Swapping loop-order changes access order
- Avoid re-writing code (less code, less bugs) ... again

```
double *psi @dim(groups, moments, zones);

psi(g,m,z) = 0;
```

```
double *psi;

psi[z + zones*(m + moments*g)] = 0;
```

Kripke: Goals

Index Ordering (Run-Time)

- Swapping loop-order changes access order
- Avoid re-writing code (less code, less bugs) ... again

```
double *psi @(dim(groups, moments, zones),
              idxOrder(0,1,2));

psi(g,m,z) = 0;
```

```
double *psi;

psi[g + groups*(m + moments*z)] = 0;
```

Kripke: Goals

Index Ordering (Run-Time)

- Swapping loop-order changes access order
- Avoid re-writing code (less code, less bugs) ... again

```
double * KRESTRICT phi     = sdom.phi->ptr(group0, 0, 0);
double * KRESTRICT psi_ptr = sdom.psi->ptr();
for(g){
  double * KRESTRICT ell_nm = sdom.ell->ptr();
  for(m){
    double * KRESTRICT psi = psi_ptr;
    for(d){
      double ell_nm_d = ell_nm[d];
      for(z){
        phi[z] += ell_nm_d * psi[z];
      }
      psi += num_zones;
    }
    ell_nm += num_local_directions;
    phi    += num_zones;
  }
  psi_ptr += num_zones * num_local_directions;
}
```

Kripke: Goals

Index Ordering (Run-Time)

- Swapping loop-order changes access order
- Avoid re-writing code (less code, less bugs) ... again

Kripke Kernels: 2004 lines
Average File : 334 lines

OCCA Kernels: 275 lines

```
kernel void LTimes(      double * restrict phi @(dim(PHI_DIM),
                                                 idxOrder(PHI_IDX)),
                   const double * restrict ell @(dim(ELL_DIM)
                                                 idxOrder(PHI_IDX)),
                   const double * restrict psi @(dim(PSI_DIM)
                                                 idxOrder(PHI_IDX))){

  for(g;      loopOrder(lo_g)){
    for(m;    loopOrder(lo_m)){
      for(d;  loopOrder(lo_d)){
        for(z; loopOrder(lo_z)){
          phi(group0 + g,m,z) += ell(d,m) * psi(g,d,z);
        }
      }
    }
  }
}
```

Kripke: Goals

Index Ordering (Run-Time)

- Swapping loop-order changes access order
- Avoid re-writing code (less code, less bugs) ... again

Kripke Kernels: 2004 lines
Average File : 334 lines

OCCA Kernels: 275 lines

```
kernel void LTimes(      double * restrict phi @(dim(PHI_DIM),
                                                 idxOrder(PHI_IDX)),
                   const double * restrict ell @(dim(ELL_DIM)
                                                 idxOrder(PHI_IDX)),
                   const double * restrict psi @(dim(PSI_DIM)
                                                 idxOrder(PHI_IDX))){

  for(g;      loopOrder(lo_g)){
    for(m;    loopOrder(lo_m)){
      for(d;  loopOrder(lo_d)){
        const double r_ell = ell(d,m);
        for(z; loopOrder(lo_z)){
          phi(group0 + g,m,z) += r_ell * psi(g,d,z);
        }
      }
    }
  }
}
```

Kripke: Goals

Command-Line Options

- Compatibility : --nest DGZ works to compare with the original Kripke code
- Loop Ordering : --lo GZDM,GZDM,ZMGP,XG,KJIGD
- Index Ordering : -io GMZ,GDZ,DM,DM,GCP,GZ
- Help : --help, --moreHelp

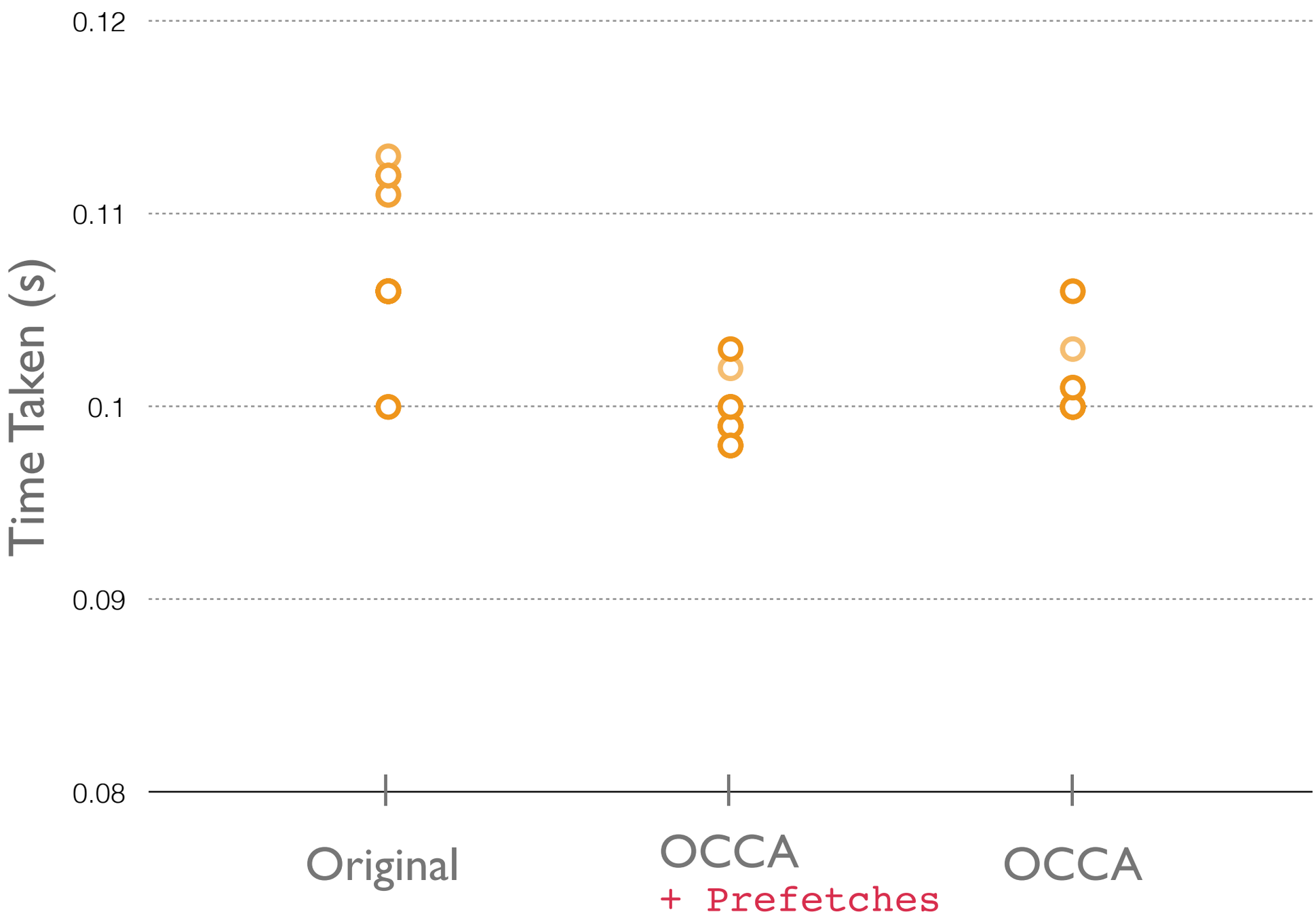
```
--LTimesOrder      GZDM
--LPlusTimesOrder  GZDM
--scatteringOrder  ZMGP
--sourceOrder      XG
--sweepOrder       KJIGD
```

```
--phiOrder      GMZ
--psiOrder      GDZ
--ellOrder      DM
--ellPlusOrder  DM
--sigsOrder     GCP
--sigtOrder     GZ
```

```
G: Group (From in scattering)
P: Group (To   in scattering)
Z: Zone
D: Directions
M: Moment
X: Mix
K: K spatial dimension
J: J spatial dimension
I: I spatial dimension
C: Coefficient
```

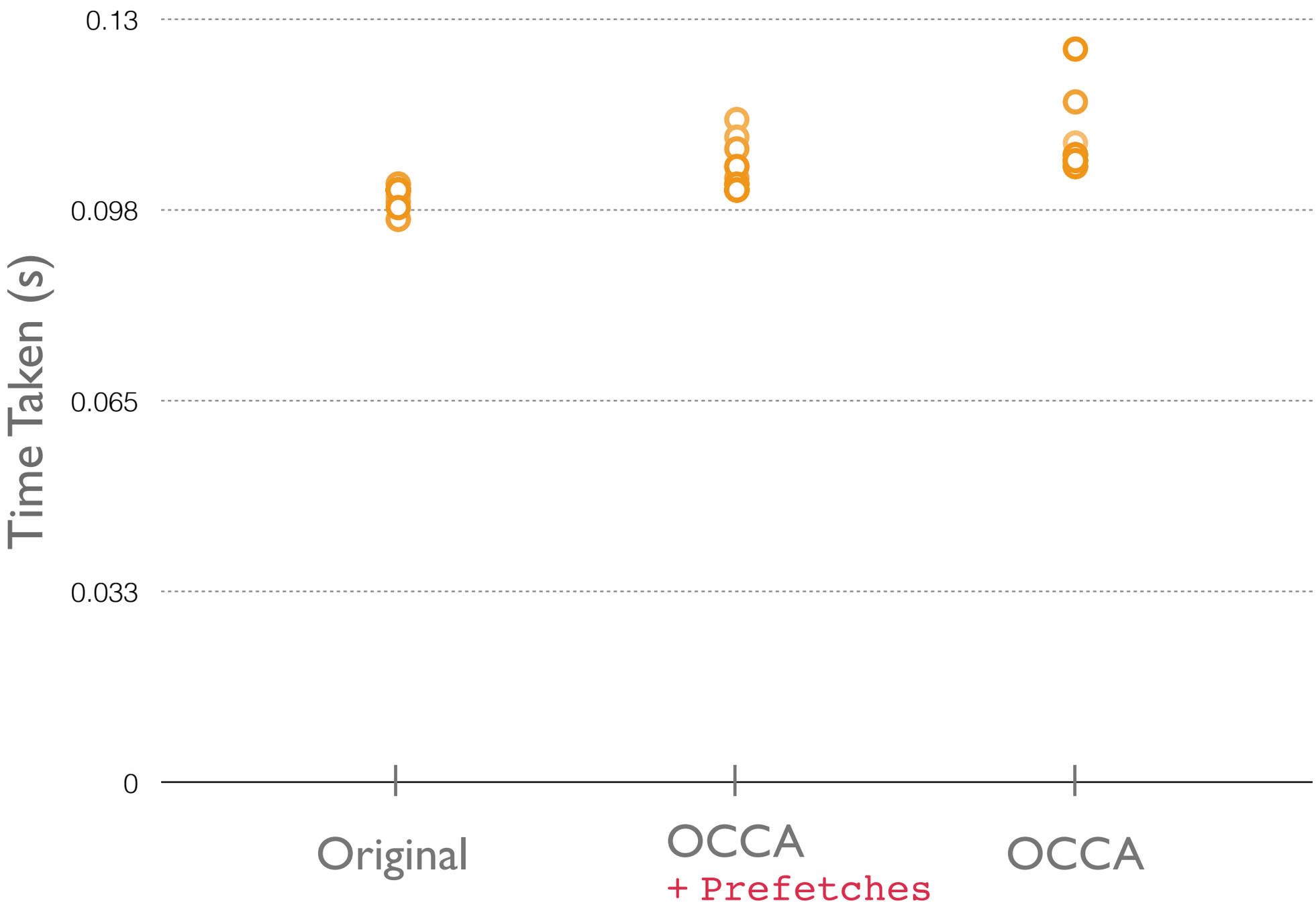
Kripke: Results

LTimes Kernel (DGZ on [cab])



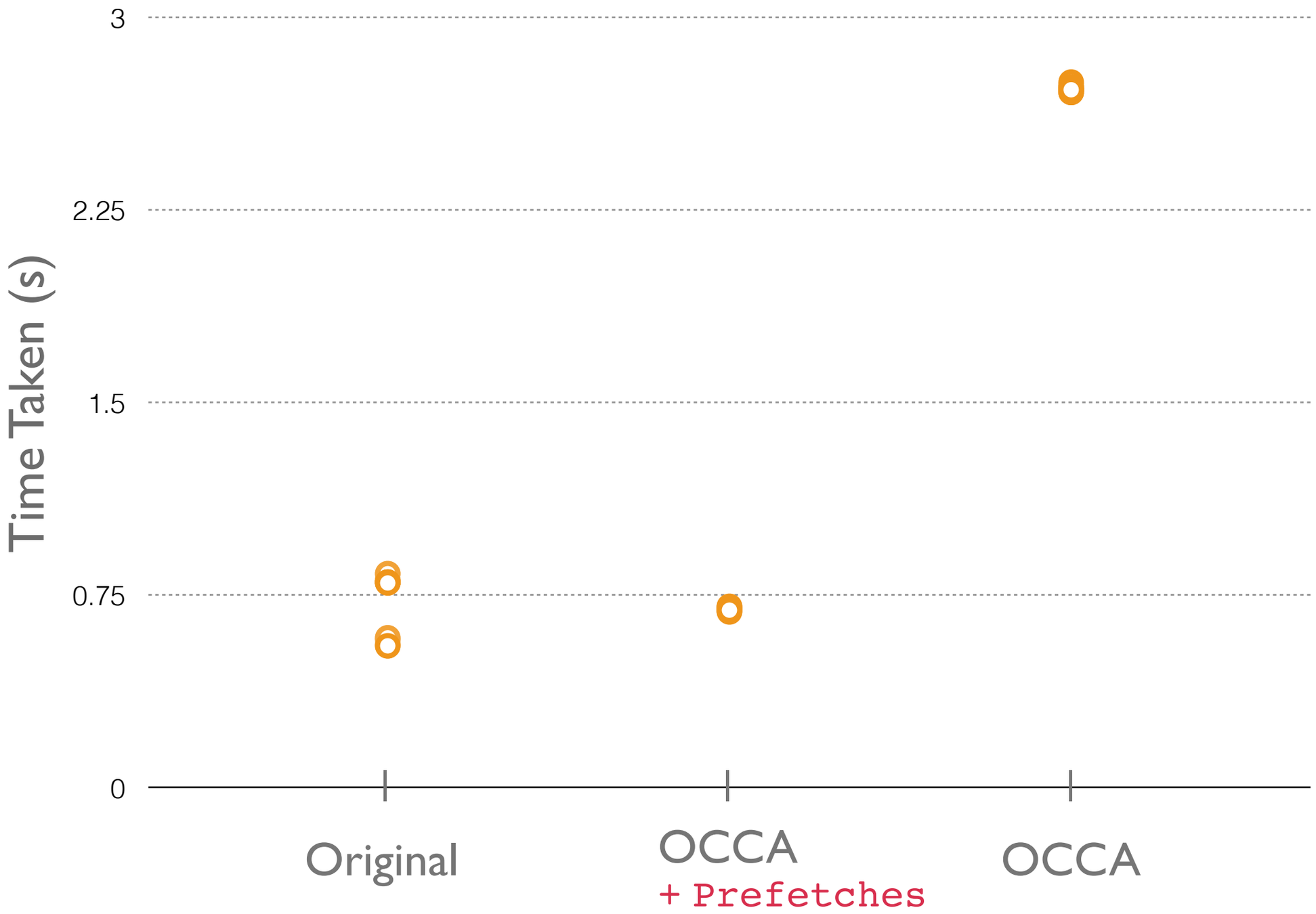
Kripke: Results

LPlusTimes Kernel (DGZ on [cab])



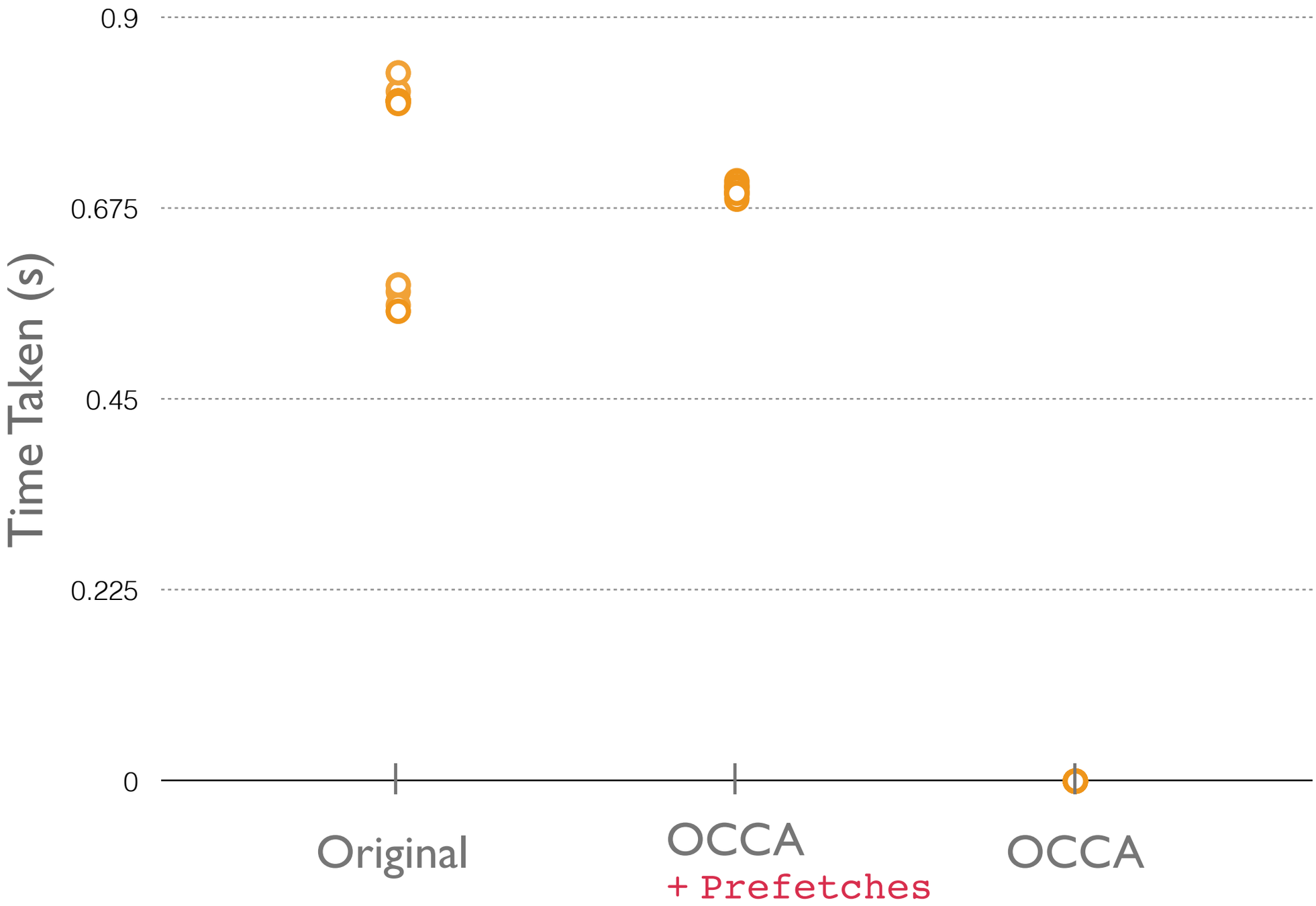
Kripke: Results

Scattering Kernel (DGZ on [cab])



Kripke: Results (Only with Prefetches)

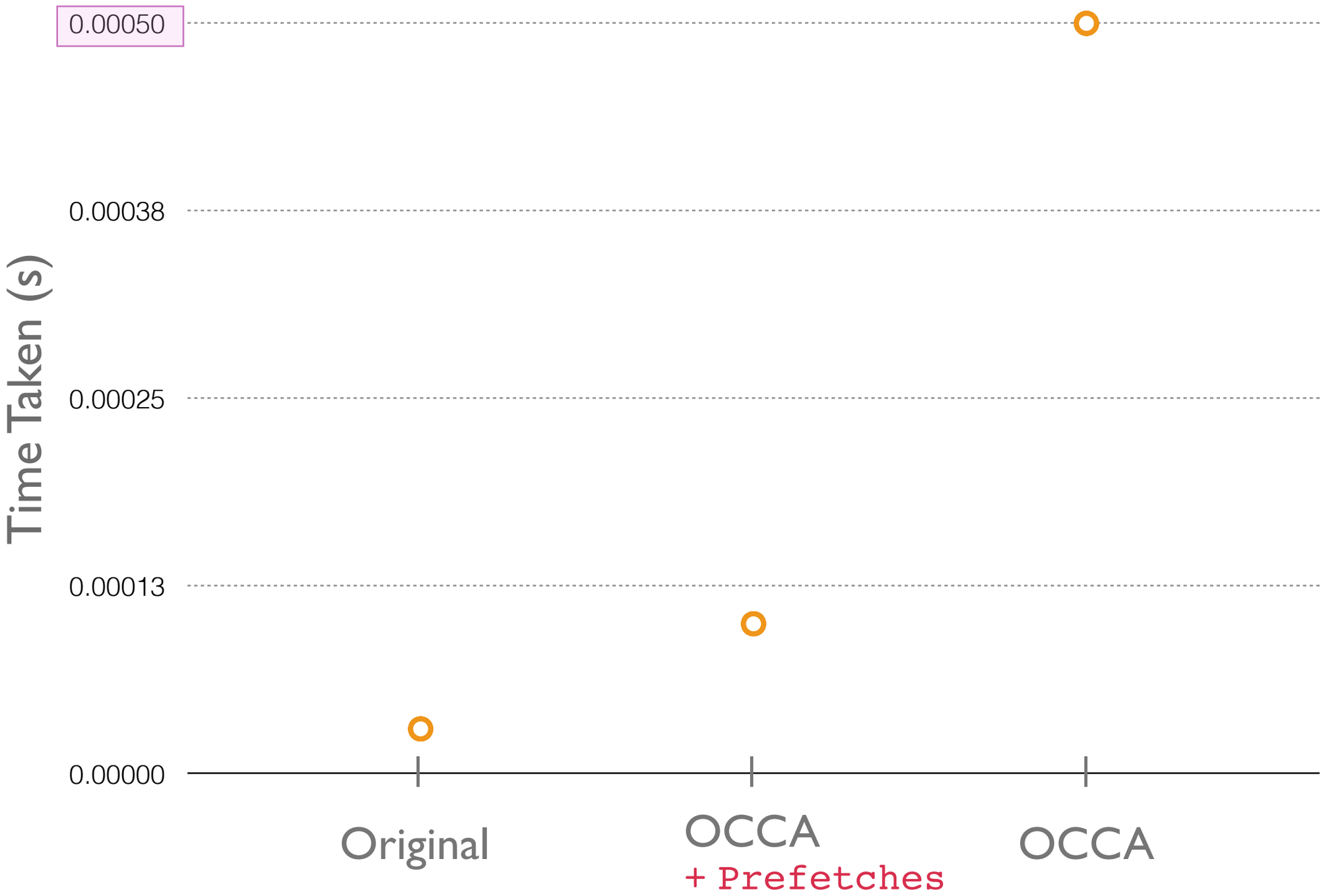
Scattering Kernel (DGZ on [cab])



Kripke: Results

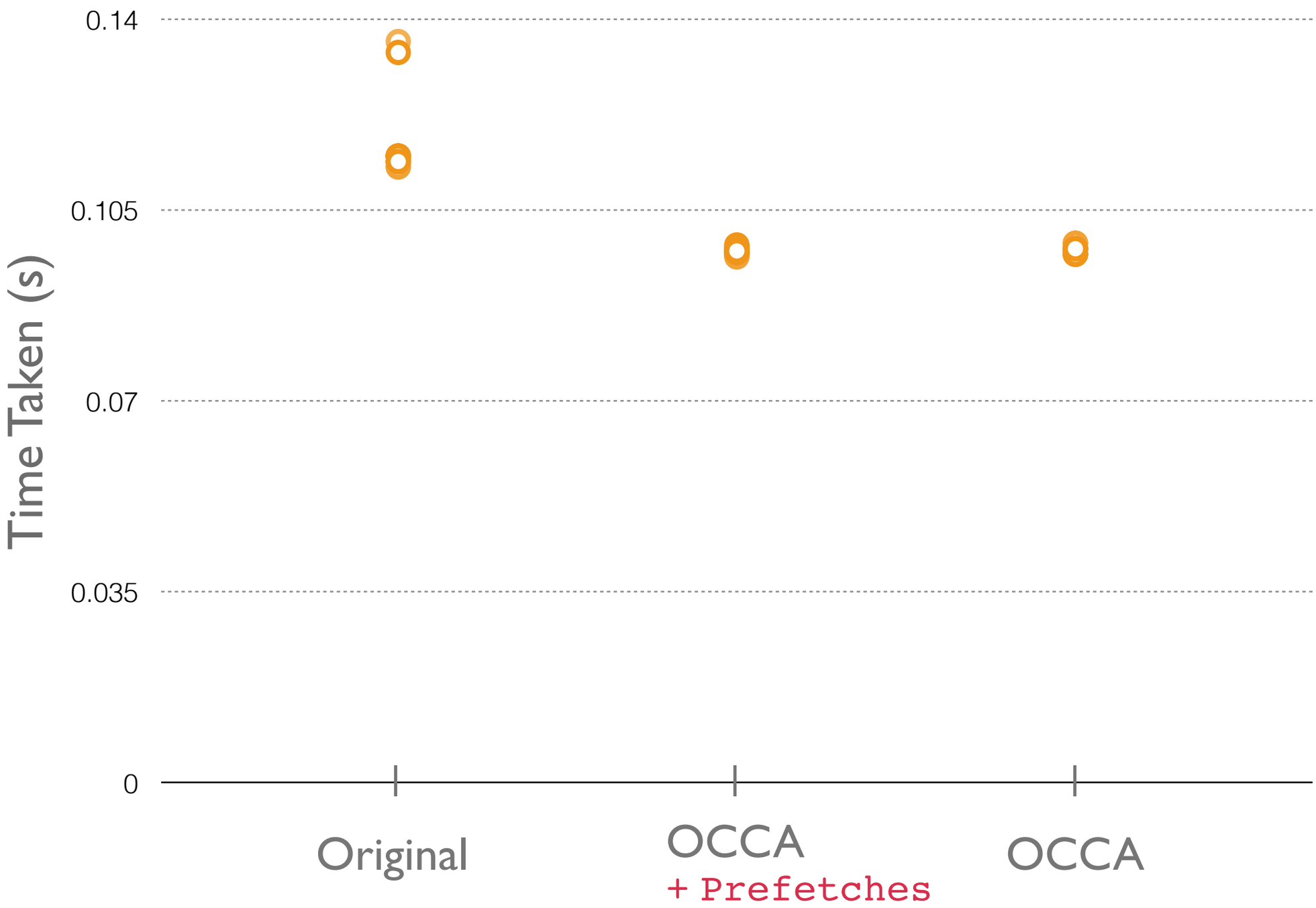
Source Kernel (DGZ on [cab])

Probably OCCA launch overhead



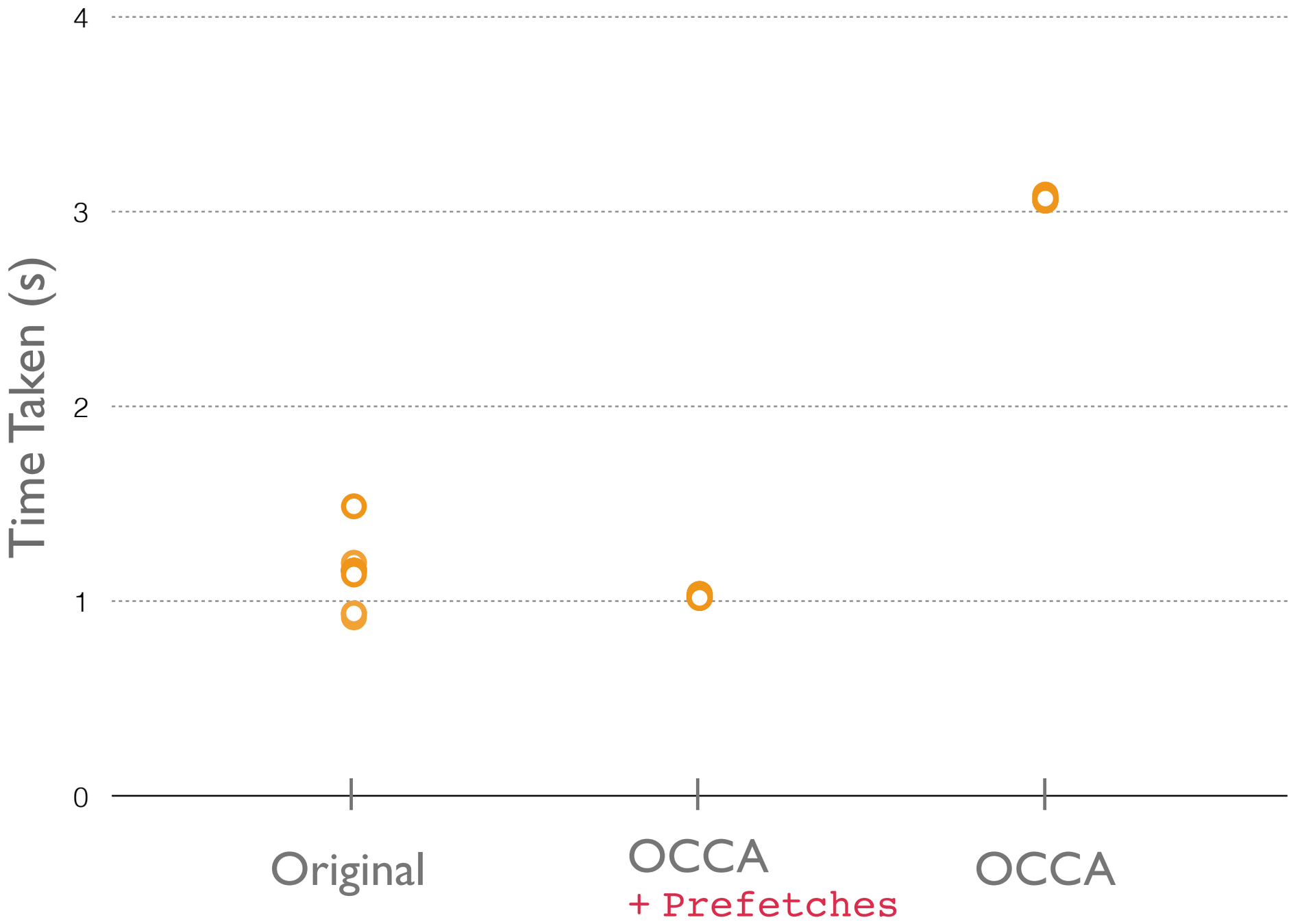
Kripke: Results

Sweep Kernel (DGZ on [cab])



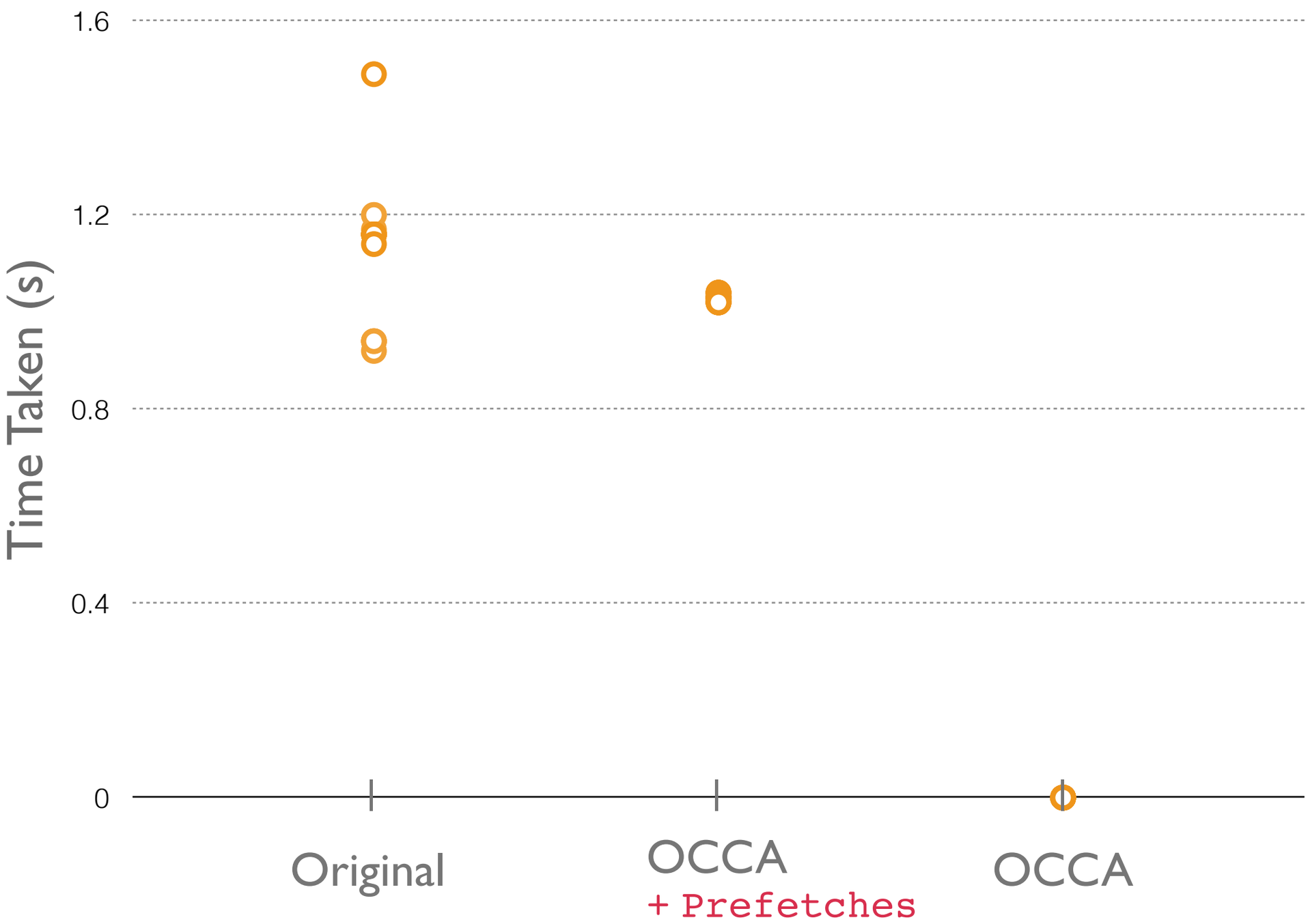
Kripke: Results

Solve Time (DGZ on [cab])



Kripke: Results (Only with Prefetches)

Solve Time (DGZ on [cab])



Live Demo

www.libocca.org

LLNL-PRES-675166