Parallel Runtime Systems and an Introduction to TaskTorrent

Léopold Cambier

March 2020

From sequential to parallel

Classical Cholesky algorithm

```
for i = 0 ... n
    A[i,i] = sqrt(A[i,i])
    for j = i+1 ... n
        A[j,i] /= A[i,i]
    end
    for j = i+1 ... n
        A[j,j] -= A[j,i] * A[j,i]ᵀ
    end
    for j = i + 1 ... n
        for k = j + 1 ... n
            A[k,j] -= A[k,i] * A[j,i]ᵀ
        end
    end
end
```

From sequential to parallel

Blocked Cholesky algorithm

```
for i = 0 ... n
    A[i,i] = potrf(A[i,i])
    for j = i+1 ... n
        trsm(A[i,i],A[j,i])
    end
    for j = i+1 ... n
        syrk(A[j,i],A[j,j])
    end
    for j = i + 1 ... n
        for k = j + 1 ... n
            gemm(A[k,i],A[j,i],A[k,j])
        end
    end
end
```

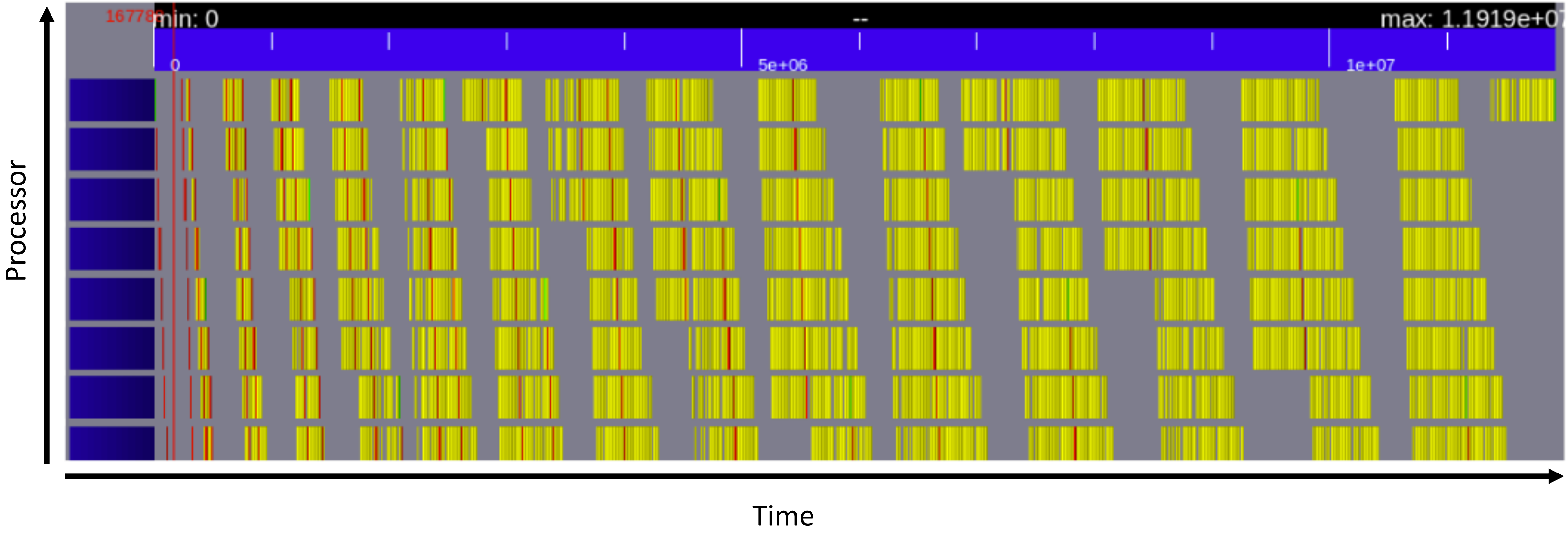
From sequential to parallel

SPMD parallel algorithm

```
for i = 0 ... n
    A[i,i] = potrf(A[i,i])
    parallel for j = i+1 ... n
        trsm(A[i,i],A[j,i])
    end
    parallel for j = i+1 ... n
        syrk(A[j,i],A[j,j])
    end
    parallel for j = i + 1 ... n
        parallel for k = j + 1 ... n
            gemm(A[k,i],A[j,i],A[k,j])
        end
    end
end
```

From sequential to parallel

SPMD with MPI



Graph from https://icl.utk.edu/projectsfiles/parsec/pubs/parsec-tutorial-1.pdf

Task Based Runtime systems to the rescue

```
for i = 0 ... n
    A[i,i] = potrf(A[i,i])
    for j = i+1 ... n
        trsm(A[i,i],A[j,i])
    end
    for j = i+1 ... n
        syrk(A[i,j],A[j,j])
    end
    for j = i + 1 ... n
        for k = j + 1 ... n
            gemm(A[k,i],A[j,i],A[k,j])
        end
    end
end
```

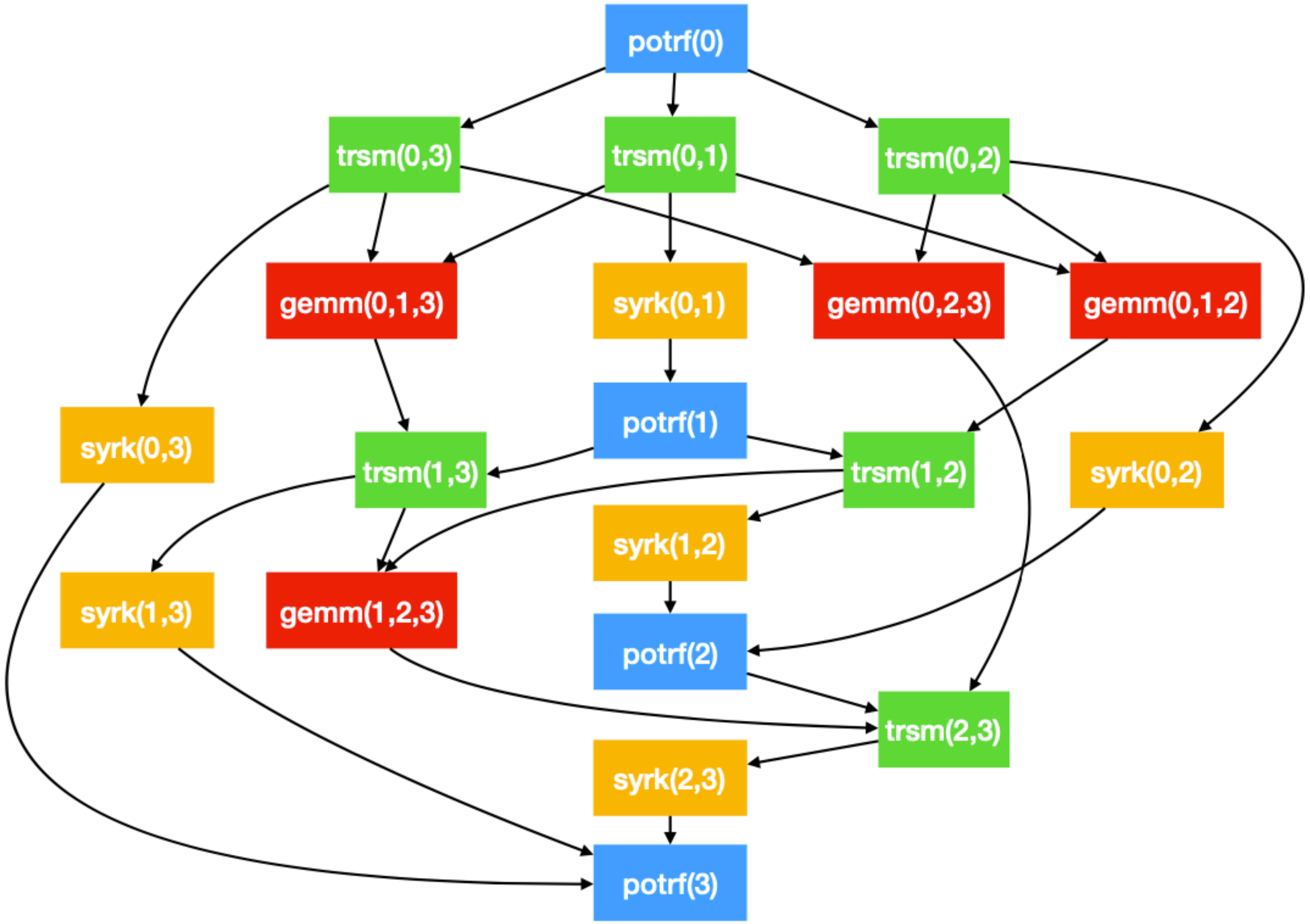
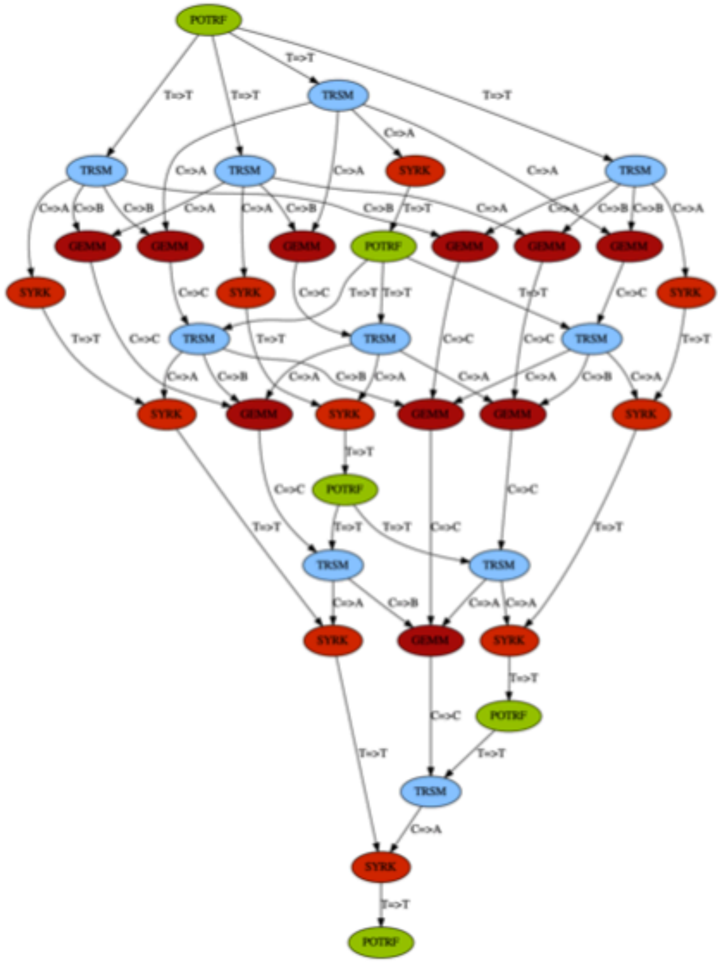
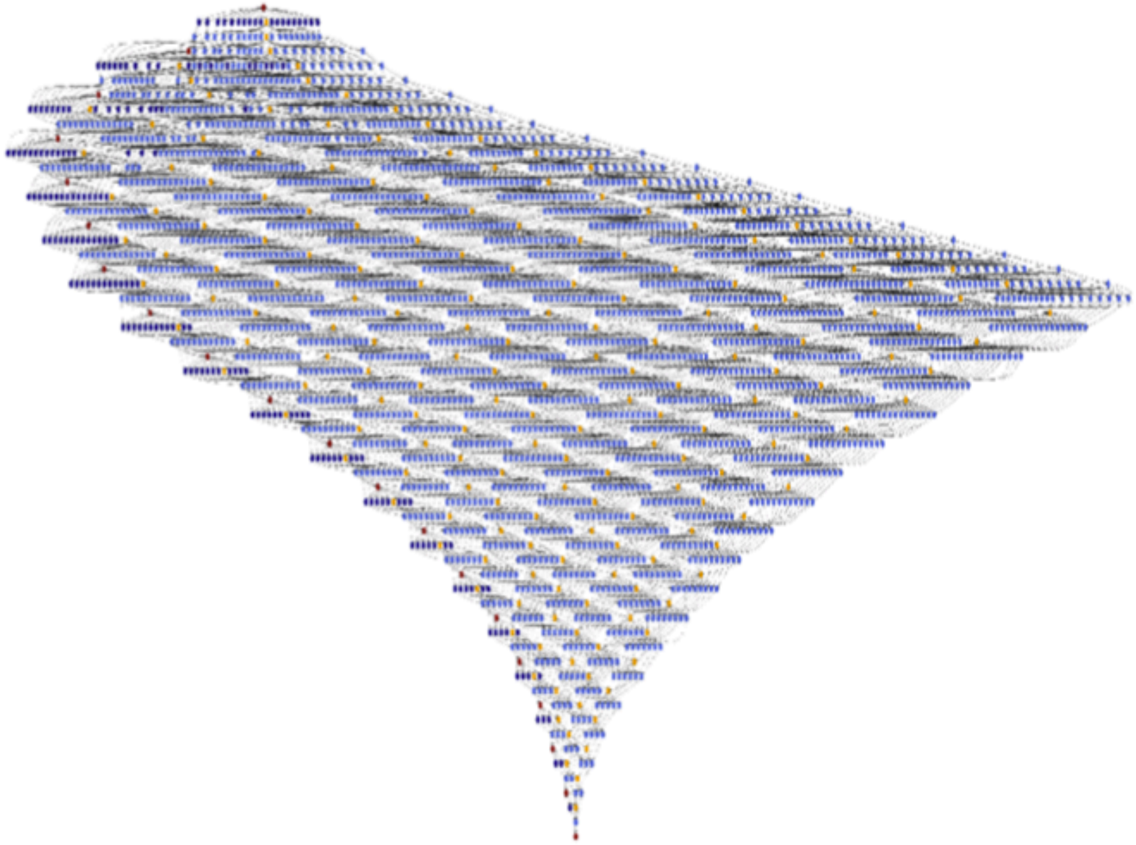


Figure courtesy of Yizhou Qian

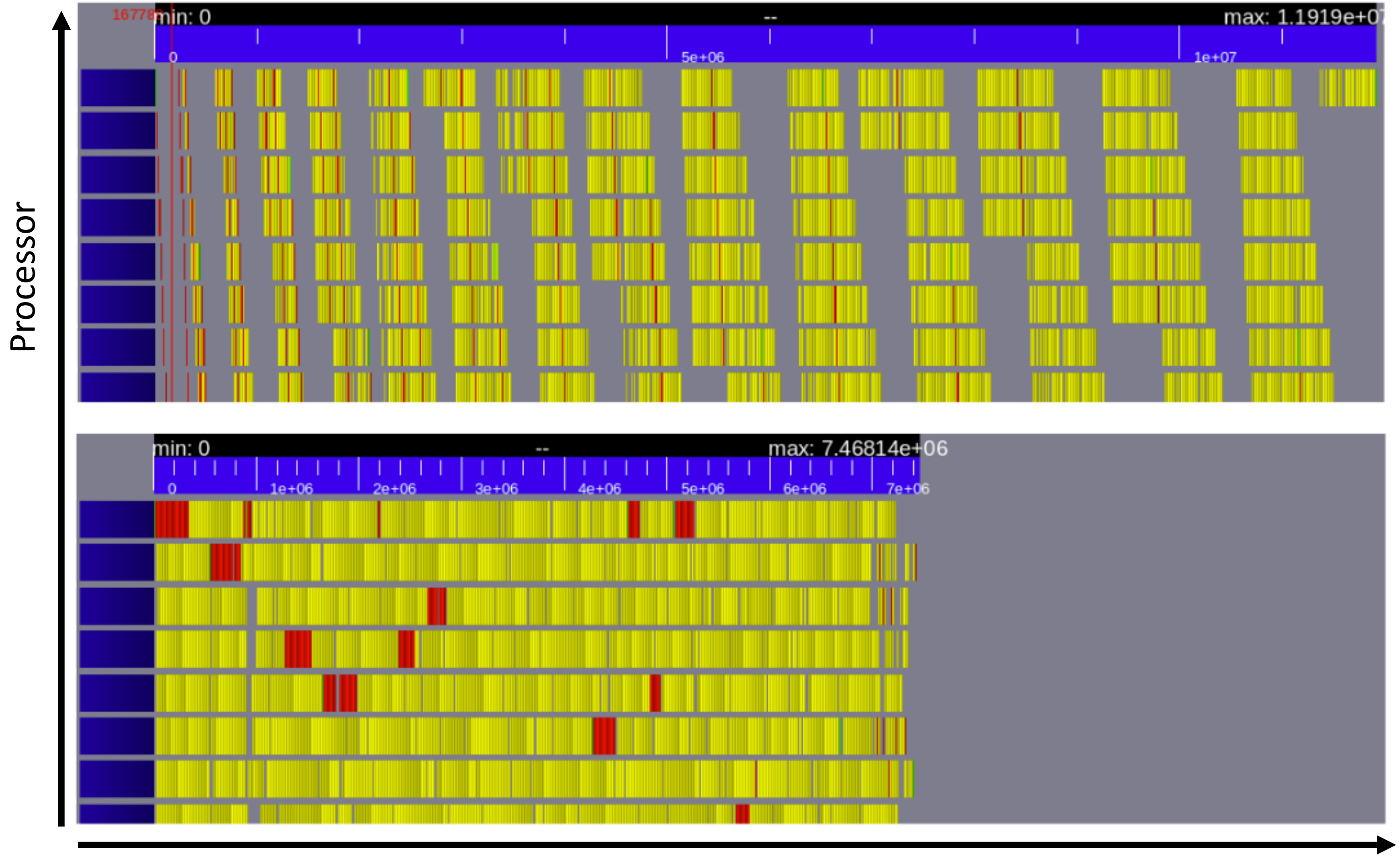
Task Based Runtime systems to the rescue





Graphs from https://icl.utk.edu/projectsfiles/parsec/pubs/parsec-tutorial-1.pdf

Task Based Runtime systems to the rescue



Graphs from https://icl.utk.edu/projectsfiles/parsec/pubs/parsec-tutorial-1.pdf

Ecosystem

Some tools (*not* an exhaustive list)

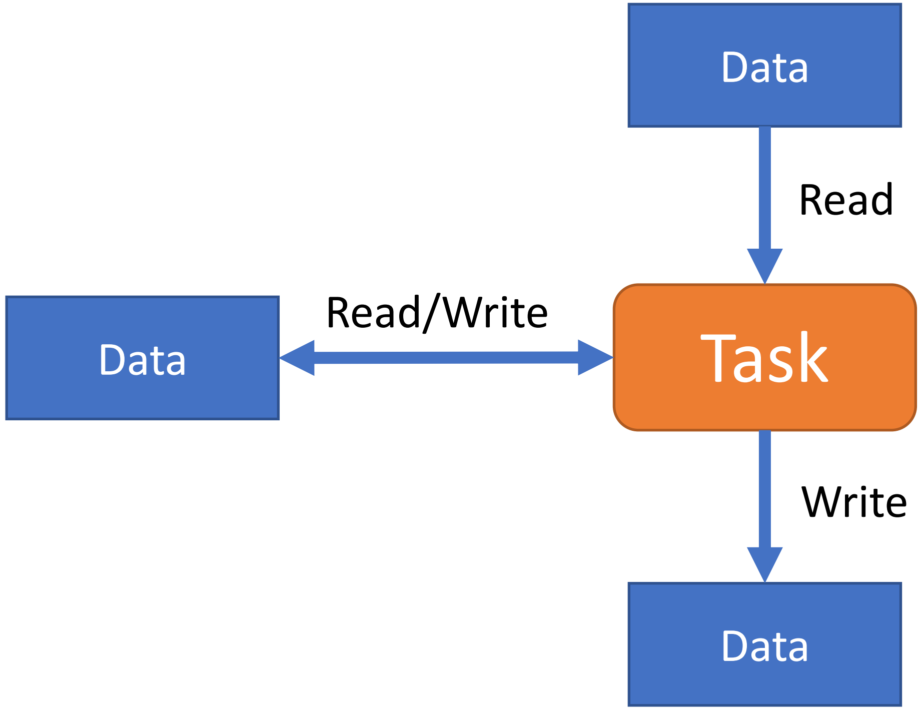
- Legion (Stanford)
- StarPU (Inria, France)
- Parsec (ICL at UTK)
- TaskTorrent (Us ☺)

Legion / Regent

- Developed at Stanford (→ CS315B)
- Language with sequential semantic (code looks “normal”)
- Logical regions (partitioned into subregions) and tasks
- Dependencies between tasks inferred from data (read, read/write, reduce)
- Distributed parallel scheduling
- Legion = C++ API ; Regent = high-level language

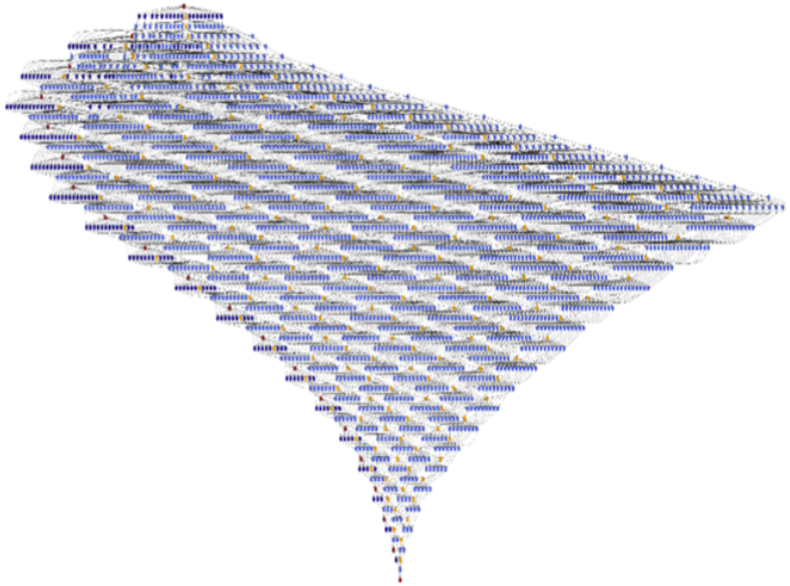
Bauer, Michael, et al. "Legion: Expressing locality and independence with logical regions."
SC'12: Proceedings of the International Conference on High Performance Computing, Networking, Storage and Analysis. IEEE, 2012.

Legion / Regent



+ Sequential-looking code & parallel scheduler

=



Right graph from https://icl.utk.edu/projectsfiles/parsec/pubs/parsec-tutorial-1.pdf

Cholesky in Regent

```
var bn = n / np
for x = 0, np do
    dpotrf(x, n, bn, pB[f2d { x = x, y = x }])
    for y = x + 1, np do
        dtrsm(x, y, n, bn, pB[f2d { x = x, y = y }], pB[f2d { x = x, y = x }])
    end
    for k = x + 1, np do
        dsyrk(x, k, n, bn, pB[f2d { x = k, y = k }], pB[f2d { x = x, y = k }])
    end
    for k = x + 1, np do
        for y = k + 1, np do
            dgemm(x, y, k, n, bn,
                  pB[f2d { x = k, y = y }],
                  pB[f2d { x = x, y = y }],
                  pB[f2d { x = x, y = k }])
        end
    end
end
```

Source: https://github.com/StanfordLegion/legion/blob/stable/language/examples/cholesky.rg

StarPU (*PU)

- Developed at Inria (France)
- Designed for hybrid architectures
- In “normal” C++

Augonnet, Cédric, et al. "StarPU: a unified platform for task scheduling on heterogeneous multicore architectures.”
Concurrency and Computation: Practice and Experience 23.2 (2011): 187-198.

Cholesky in StarPU

```
for (int kk = 0; kk < nb; ++kk) {
    starpu_mpi_task_insert(MPI_COMM_WORLD, &potrf_cl,STARPU_RW, dataA[kk+kk*nb], 0);
    for (int ii = kk+1; ii < nb; ++ii) {
        starpu_mpi_task_insert(MPI_COMM_WORLD, &trsm_cl,STARPU_R, dataA[kk+kk*nb], STARPU_RW, dataA[ii+kk*nb], 0);
        starpu_mpi_cache_flush(MPI_COMM_WORLD, dataA[kk+kk*nb]);
        for (int jj=kk+1; jj < nb; ++jj) {
            if (jj <= ii) { if (jj==ii) {
                starpu_mpi_task_insert(MPI_COMM_WORLD, &syrk_cl, STARPU_R,
                                       dataA[ii+kk*nb], STARPU_RW, dataA[ii+jj*nb], 0);
            } else {
                starpu_mpi_task_insert(MPI_COMM_WORLD, &gemm_cl,STARPU_R,
                                       dataA[ii+kk*nb], STARPU_R, dataA[jj+kk*nb], STARPU_RW, dataA[ii+jj*nb], 0);
        }
    }
}
```

➔ Sequential task flow

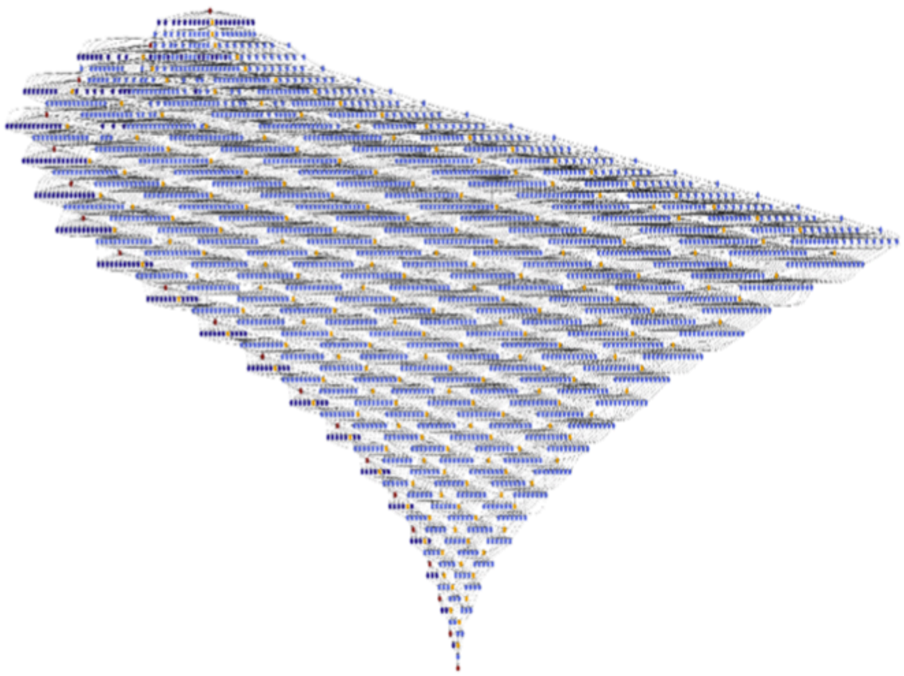
Source: StarPU source code, https://gforge.inria.fr/frs/?group_id=1570

Sequential Task Flow

Complex to explore (in distributed way!) a very large task graph

Can lead to very large overhead (CPU/RAM) when tasks are small

→ Think local

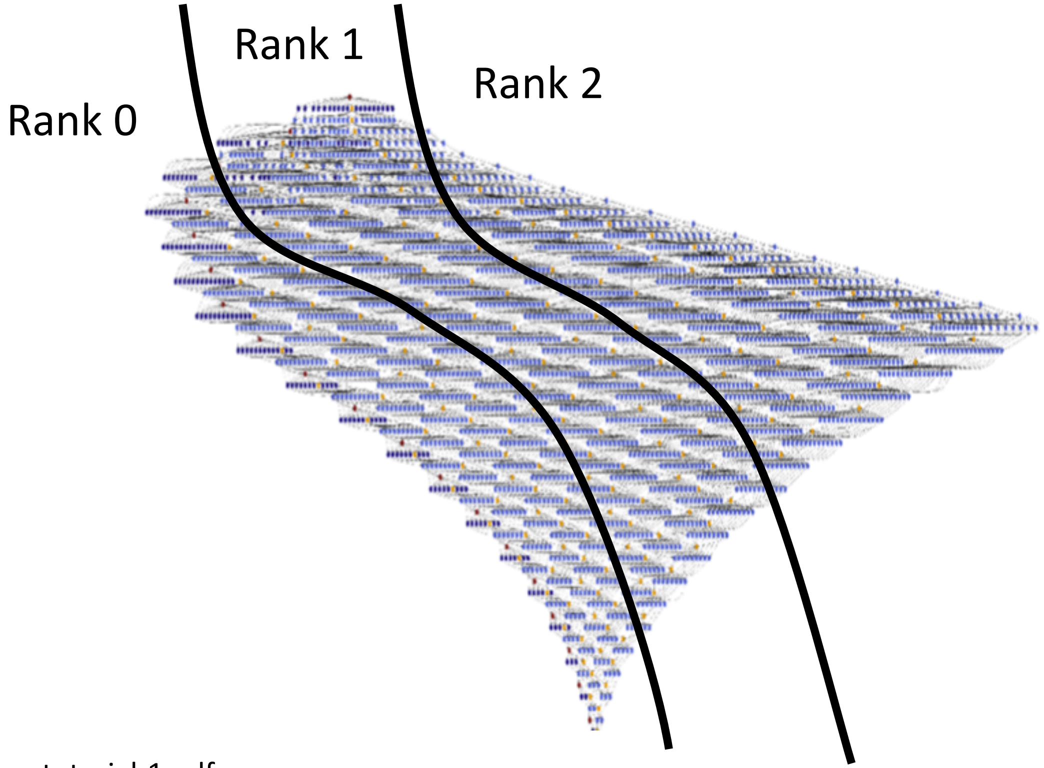


From https://icl.utk.edu/projectsfiles/parsec/pubs/parsec-tutorial-1.pdf

Parametrized Task Graph

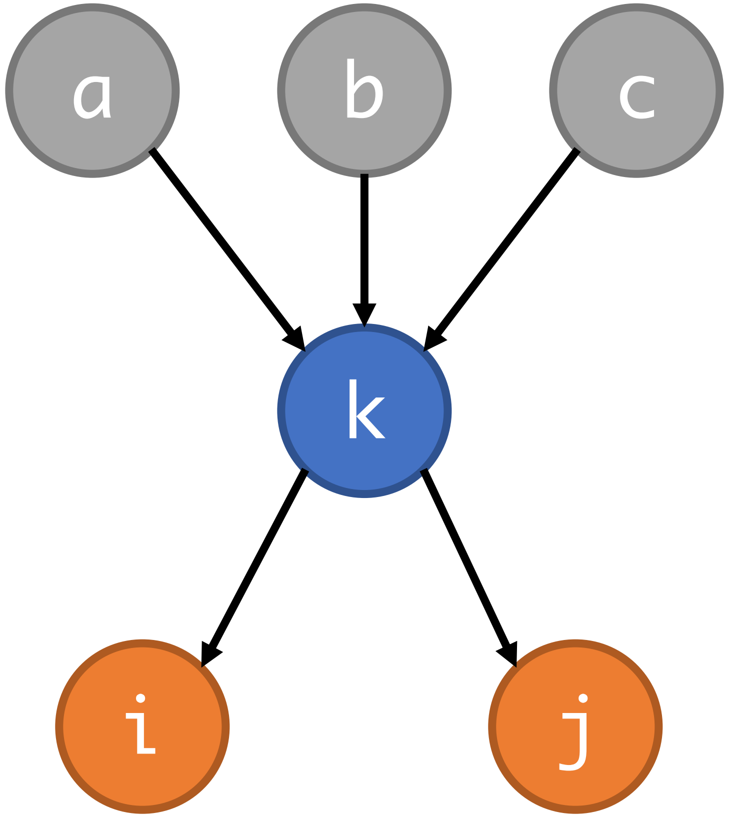
Solution: only express the local graph

PTG is one way to do that



From https://icl.utk.edu/projectsfiles/parsec/pubs/parsec-tutorial-1.pdf

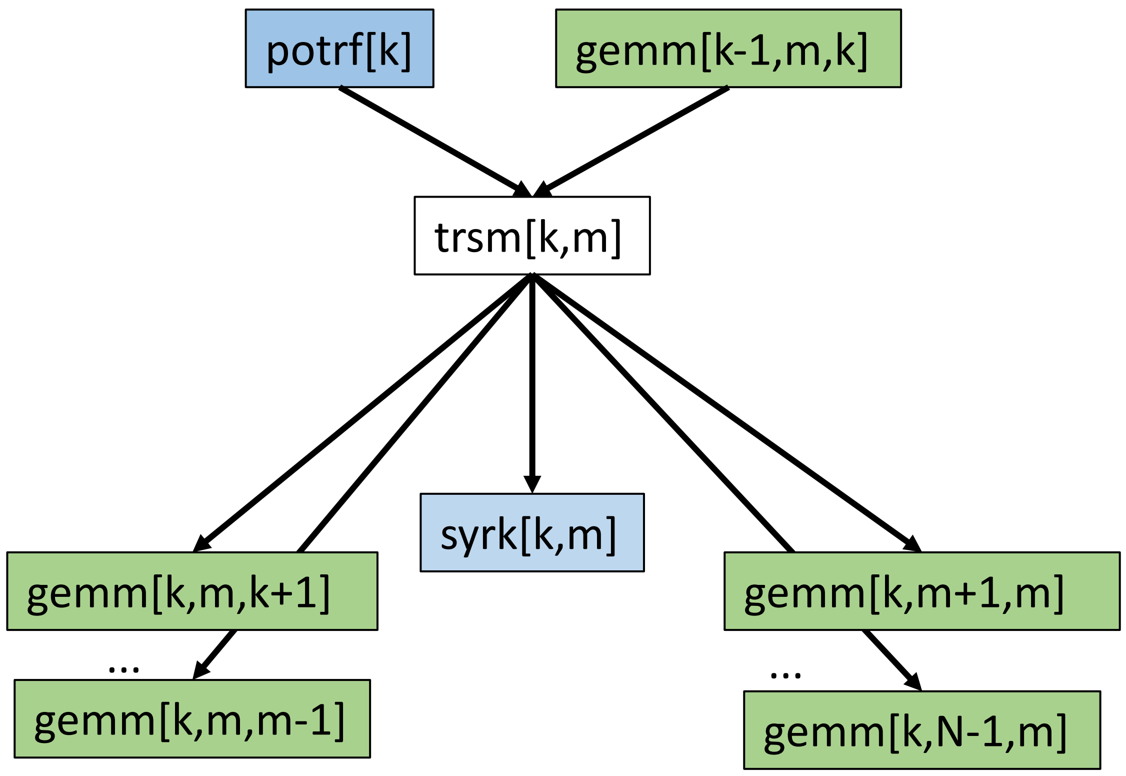
Parametrized Task Graph




```
incoming_deps(k) {
    // Describe incomings deps
}

task(k) {
    // Describe what to run
}

outgoing_deps(k) {
    // Describe outgoing deps
}
```

PaRSEC

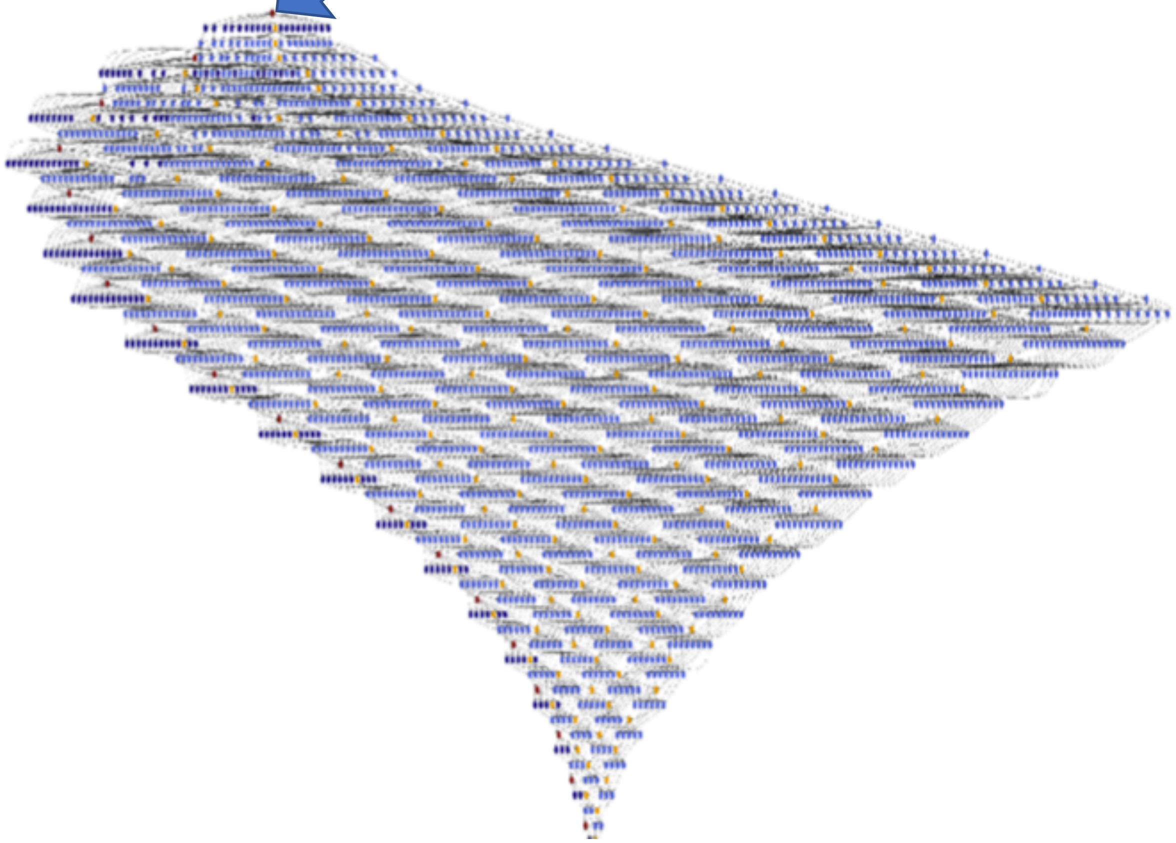
```
TRSM(k, m)

// Execution space
k = 0   .. NT-1
m = k+1 .. NT-1

// Partitioning
: dataA(m, k)

// Flows & their dependencies
READ  A <- A POTRF(k) [type = LOWER]
RW    C <- (k == 0) ? dataA(m, k)
        <- (k != 0) ? C GEMM(k-1, m, k)
        -> A SYRK(k, m)
        -> A GEMM(k, m, k+1..m-1)
        -> B GEMM(k, m+1..NT-1, m)
        -> dataA(m, k)

BODY
 trsm(A, C);
END
```



From https://icl.utk.edu/projectsfiles/parsec/pubs/parsec-tutorial-1.pdf

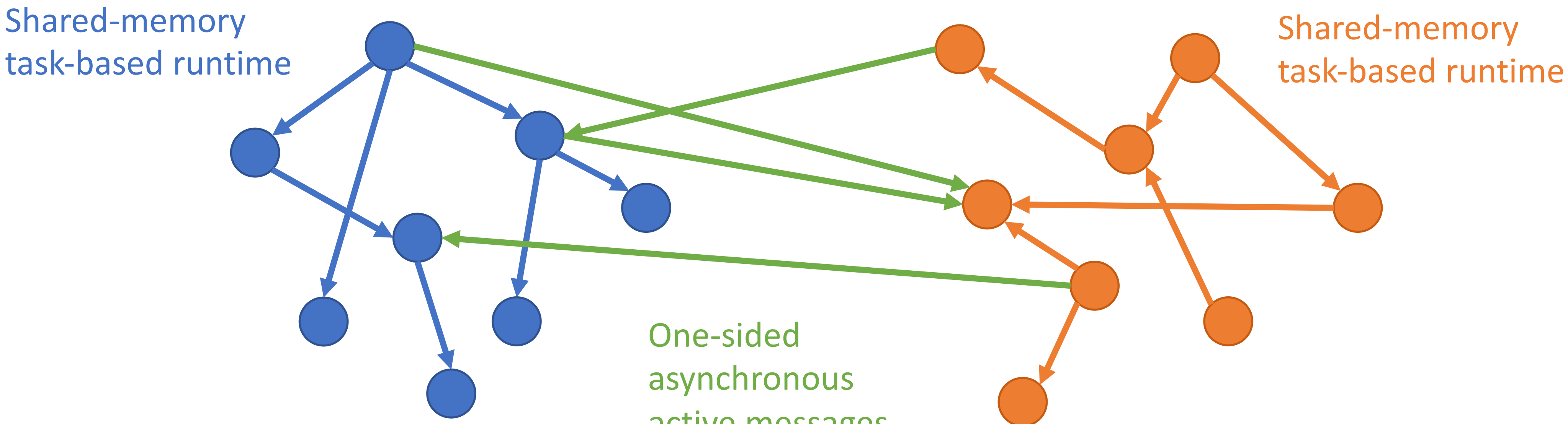
How to start ?

Start the first task
Everything flows from there



TaskTorrent

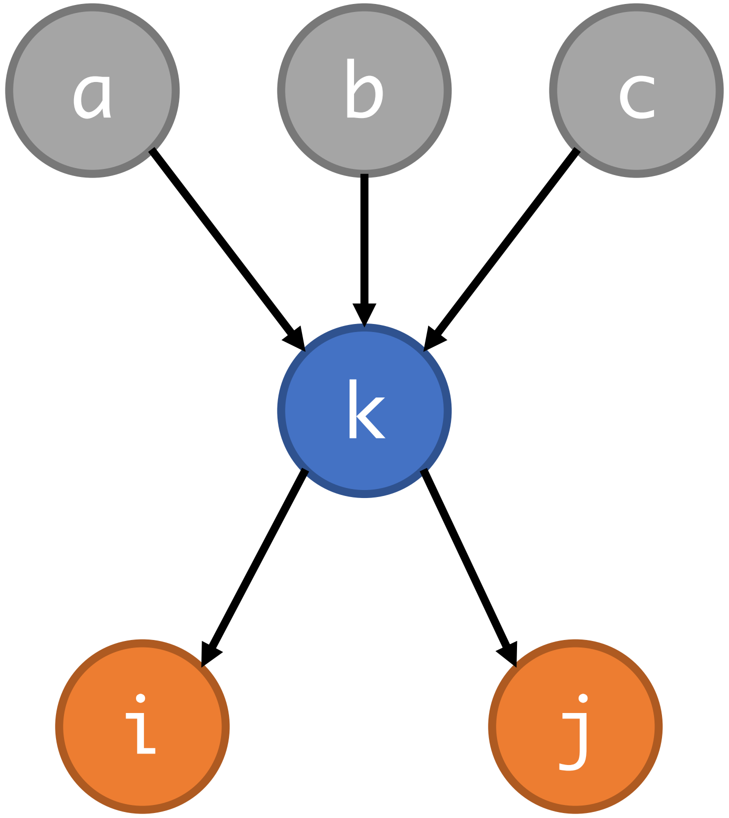
https://github.com/leopoldcambier/tasktorrent



TaskTorrent

- C++ library
- Parametrized task graph
- Lightweight (minimal scheduling overhead), designed for small tasks
- Dependencies and data are separated
- C++ threads and MPI communication backend
- Only dependency is MPI

Parametrized Task Graph in TaskTorrent




```
n_deps_in(int k) {
    return deps(k)
}

task(int k) {
    do_something(k)

    for(k_ in deps(k)) fulfill_promise(k_)
}
```

TaskTorrent

- Tasks defined over an index space k (`int`, `tuple<int,N>`, ...)

```
// Task flow definition
tf.set_indegree([&](int k) {
    // How many incoming dependencies
    return 1;
}).set_task([&](int k) {
    // What to run
    data[k] = run(k);
    // ...
```

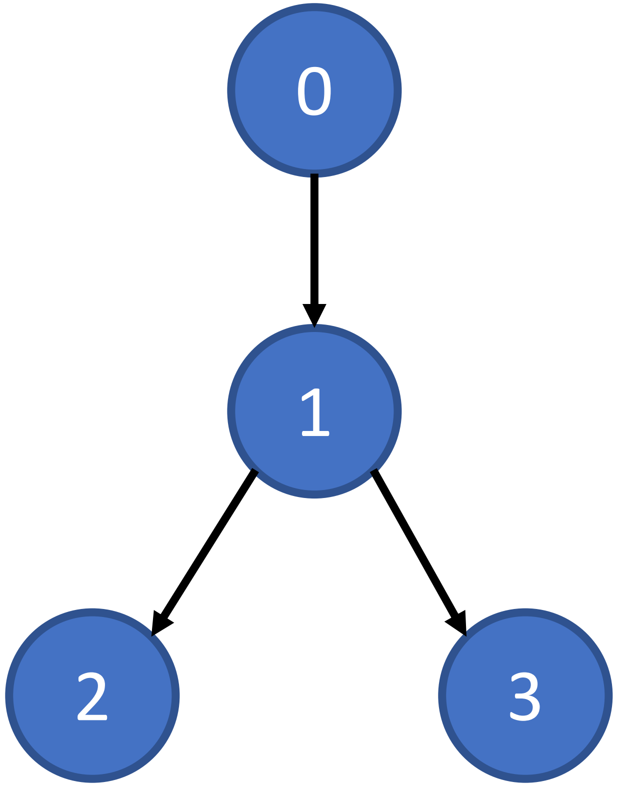
TaskTorrent

- Tasks trigger other tasks:
    - Immediately on the same rank
    - By sending an active message, typically with some data, to another rank

```
// What to trigger
for(auto k_: task_deps[k]) {
    int dest = task_to_rank(k_);
    if(dest == rank) {
        tf.fulfill_promise(k_);
    } else {
        am->send(dest, data[k], k_);
    }
 }
})
```

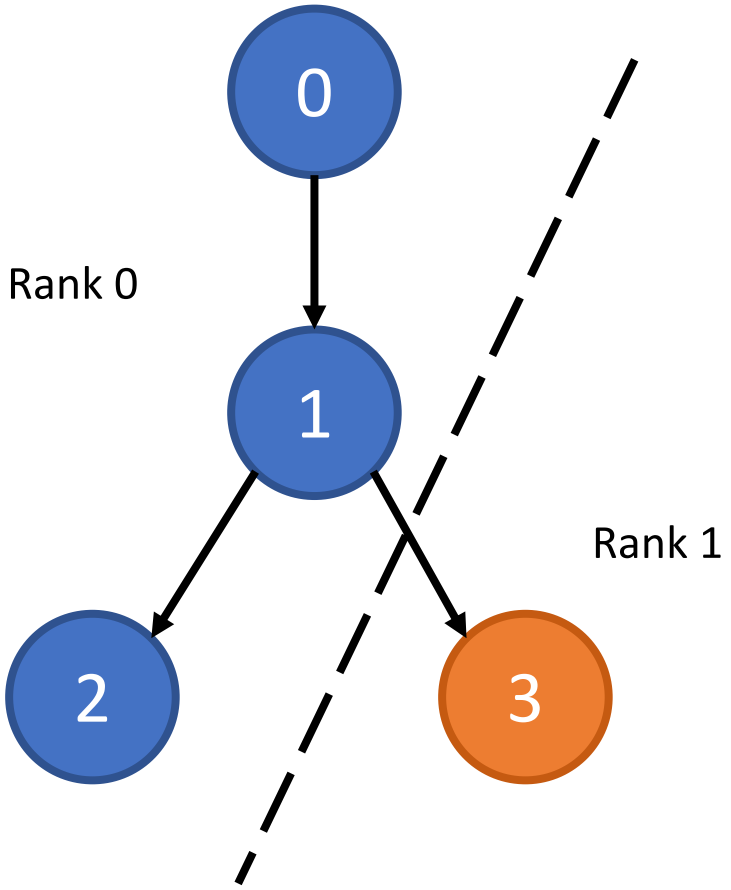
```
// Runs on a remote rank
// Copies data + triggers other
// tasks
auto am = comm.make_active_msg(
[&](int &data, int &k, int &k_) {
    my_data[k] = data;
    tf.fulfill_promise(k_);
});
```

Shared memory example




```
// Task flow tf
tf.set_indegree([&](int k) {
    // How many incoming dependencies
    return 1;
}).set_task([&](int k) {
    // What to do
    if(data_inputs[k] == -1) {
        task_data[k] = run(k, 0);
    } else {
        task_data[k] = run(k,task_data[data_inputs[k]]);
    }
    // Trigger dependencies
    for(auto k_: task_deps[k]) {
        tf.fulfill_promise(k_);
    }
// Where are tasks suggested to run (tasks can be stolen!)
}).set_mapping([&](int k) {
    return (k % n_threads);
});
```

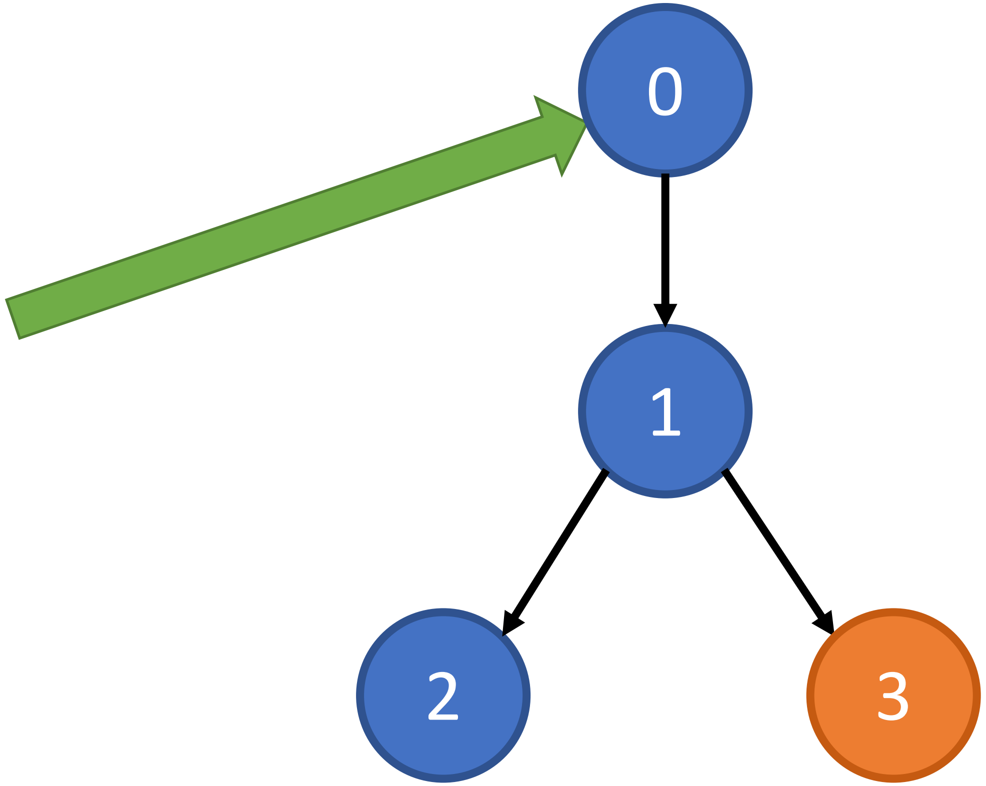
Distributed memory example




```
// Active message
auto am = comm.make_active_msg(
[&](int &data, int &k, int &k_) {
    task_data[k] = data;
    tf.fulfill_promise(k_);
});

// Task flow
tf.set_indegree([&](int k) {
    return 1;
}).set_task([&](int k) {
    if(data_inputs[k] == -1) {
        task_data[k] = run(k, 0);
    } else {
        task_data[k] = run(k, task_data[data_inputs[k]]);
    }
    for(auto k_: task_deps[k]) {
        int dest = task_to_rank(k_);
        if(dest == rank) {
            tf.fulfill_promise(k_);
        } else {
            am->send(dest, task_data[k], k, k_);
        }
    }
}).set_mapping([&](int k) {
    return (k % n_threads);
});
```

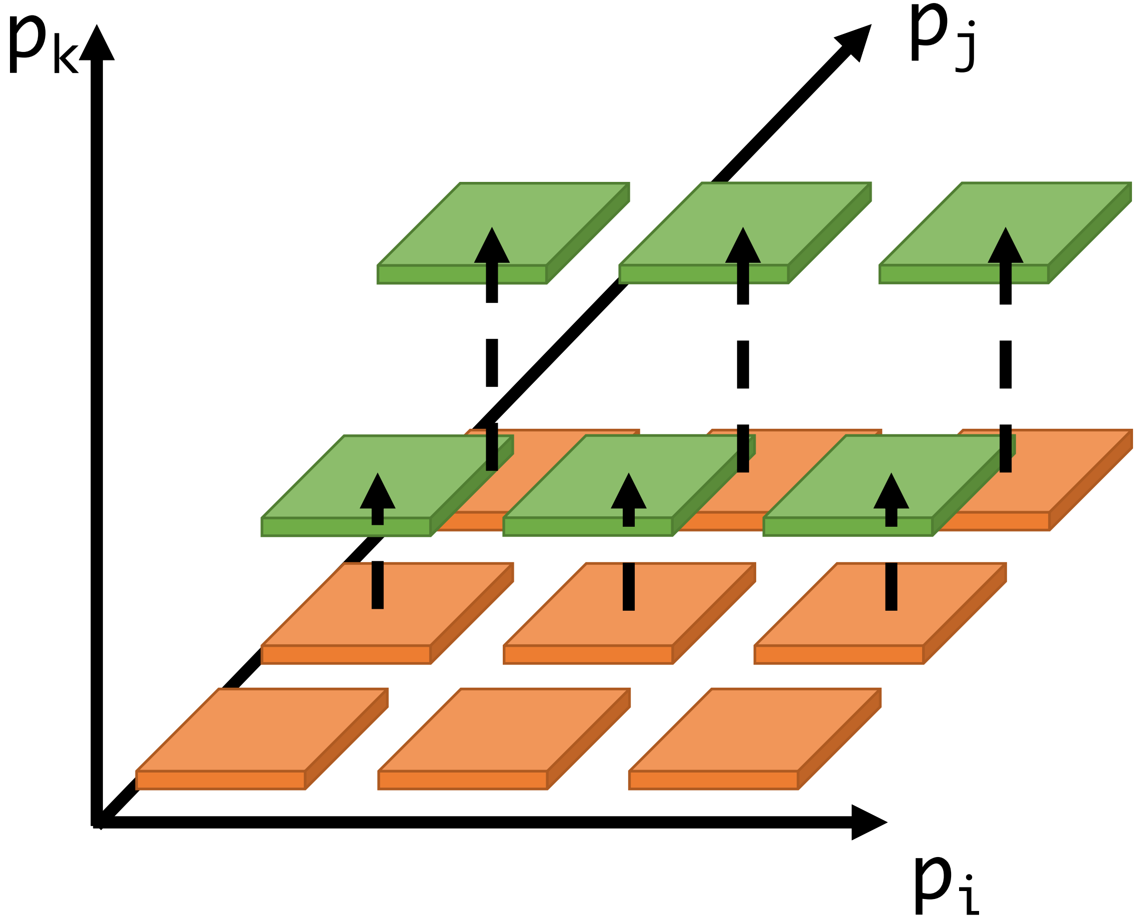
How to start ?

```
tf.fulfill_promise(0);
```



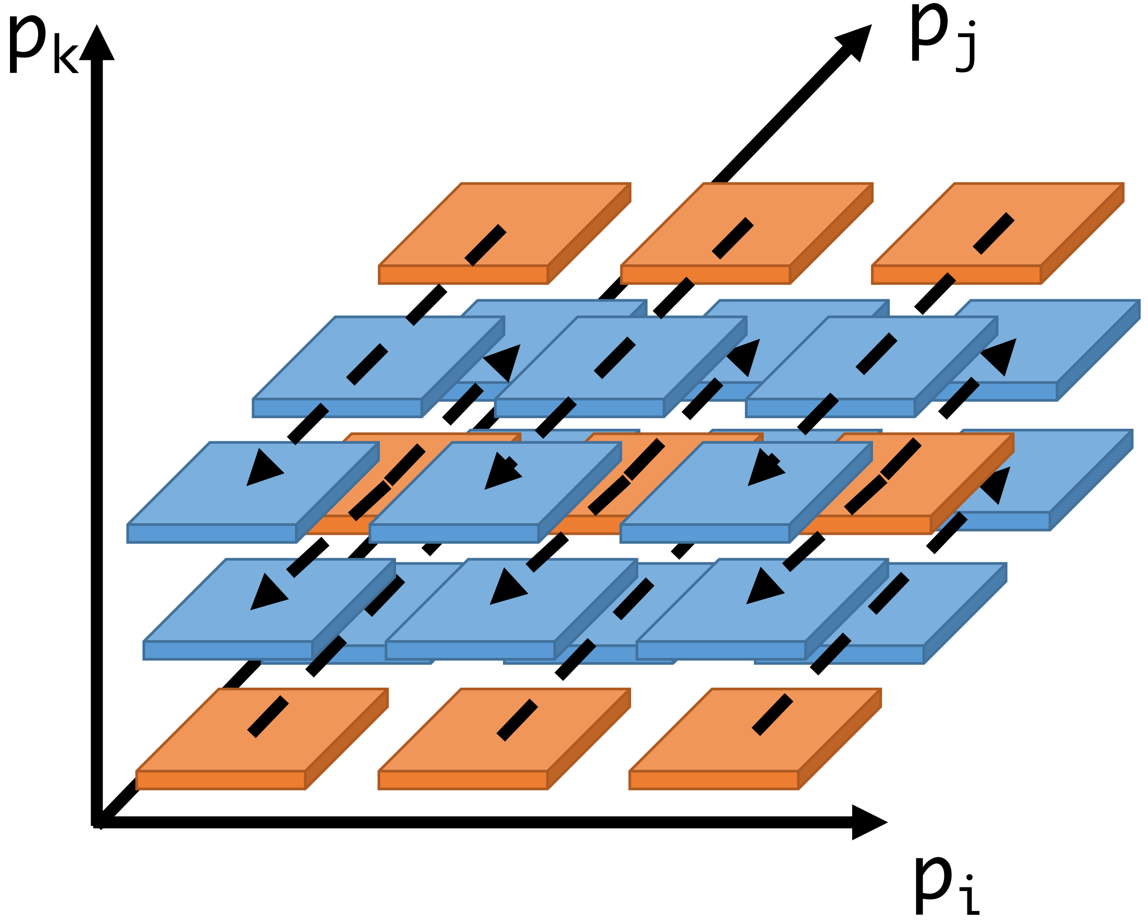
3D Gemm

(i,j,0) A[i,j] → (i,j,j)



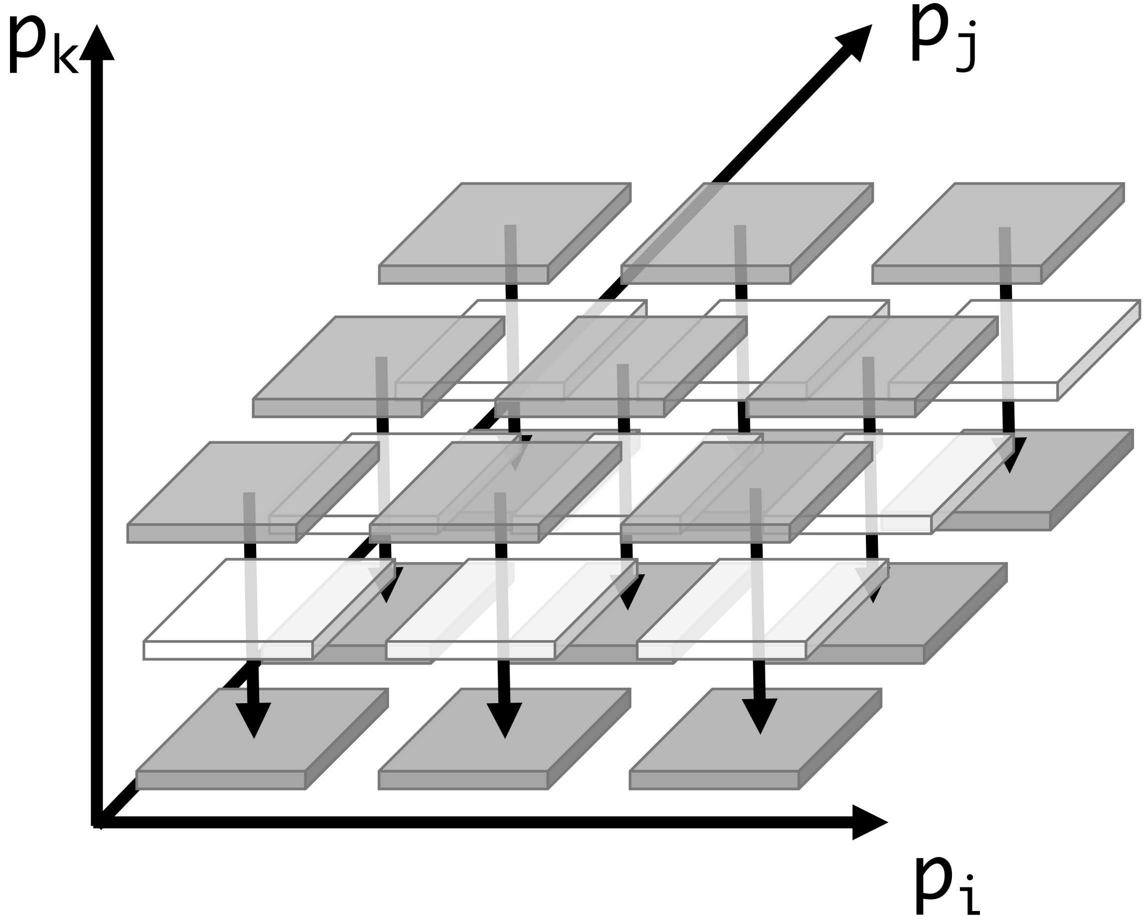
3D Gemm

(i,j,j) A[i,j] → (i,*,j)



3D Gemm

(i,j,k) A[i,k]*B[k,j] → (i,j,0)



Send: (i,j,0) A(i,j) → (i,j,j)

```
// Stores A_ij and triggers broadcast
auto send_Aij_am = comm.make_active_msg([&](ttor::view<double>& Aij) {
    copy_from_view(&A_ij, Aij);
    bcst_Aij.fulfill_promise(0);
});


// (i,j,0) sends A_ij to (i,j,j) for all i,j
send_Aij.set_task([&](int ijk){
    ttor::view<double> A_view = make_view(&A_ij);
    int dest = rank_ijk_to_rank(rank_i, rank_j, rank_j);
    if(dest != rank) {
        send_Aij_am->send(dest, A_view);
    } else {
        bcst_Aij.fulfill_promise(0);
    }
}).set_indegree([&](int ijk) {
   return 1;
})
```

Broadcast: (i,j,j) A(i,j) → (i,*,j)

```
// Store A_ij and triggers gemm
auto bcst_Aij_am = comm.make_active_msg([&](ttor::view<double>& Aij) {
    copy_from_view(&A_ij, Aij);
    gemm_Cijk.fulfill_promise(0);
});


// (i,j,j) sends A_ij along j to all (i,*,j) for all i,j
bcst_Aij.set_task([&](int ijk){
    ttor::view<double> A_view = make_view(&A_ij);
    for(int k = 0; k < n_ranks_1d; k++) {
        int dest = rank_ijk_to_rank(rank_i, k, rank_j);
        if(dest != rank) {
            bcst_Aij_am->send(dest, A_view);
        } else {
            gemm_Cijk.fulfill_promise(0);
        }
    }
}).set_indegree([&](int ij) {
return 1;
```

Gemm:
(i,j,k) A(i,k)*B(k,j) → (i,j,0)

```
// Stores C_ijk to be accumulated later
auto accu_Cijk_am = comm.make_active_msg([&](ttor::view<double>& Cijk, int& k) {
    copy_from_view(&C_ijks[k], Cijk);
    accu_Cijk.fulfill_promise(k);
});

// (i,j,k) computes C_ijk = A_ik * B_kj
gemm_Cijk.set_task([&](int ijk){
    C_ijk.noalias() += A_ij * B_ij;
    auto C_ijk_view = make_view(&C_ijk);
    int dest = rank_ijk_to_rank(rank_i, rank_j, 0);
    int k = rank_k;
    accu_Cijk_am->send(dest, C_ijk_view, k);
}).set_indegree([&](int ij) {
    return 2;
});
```

Accumulate:

```
(i,j,0) C(i,j)+=A(i,k)*B(k,j)
```

```
// Accumulate into C_ij; C_ij += Cijks[k]
accu_Cijk.set_task([&](int k){
    accumulate(&C_ij, &C_ijks[k]);
}).set_indegree([&](int ij) {
    return 1;
});
```

Trying out TaskTorrent

From laptop or your Google Cloud VM from HW5. You need git, mpicxx/mpirun and make

```
> git clone https://github.com/leopoldcambier/tasktorrent.git
> cd tasktorrent/tutorial
> make
> mpirun -n 2 ./tuto
```

```
Rank 0 hello from MyComputer
Rank 1 hello from MyComputer
Task 2 is now running on rank 1
Task 2 fulfilling local task 3 on rank 1
Task 0 is now running on rank 0
Task 0 fulfilling local task 1 on rank 0
Task 0 fulfilling 3 (remote)
Task 3 is now running on rank 1
Task 2 fulfilling 1 (remote)
Task 1 is now running on rank 0
```