From Data-Flow to Distributed DAG Scheduling

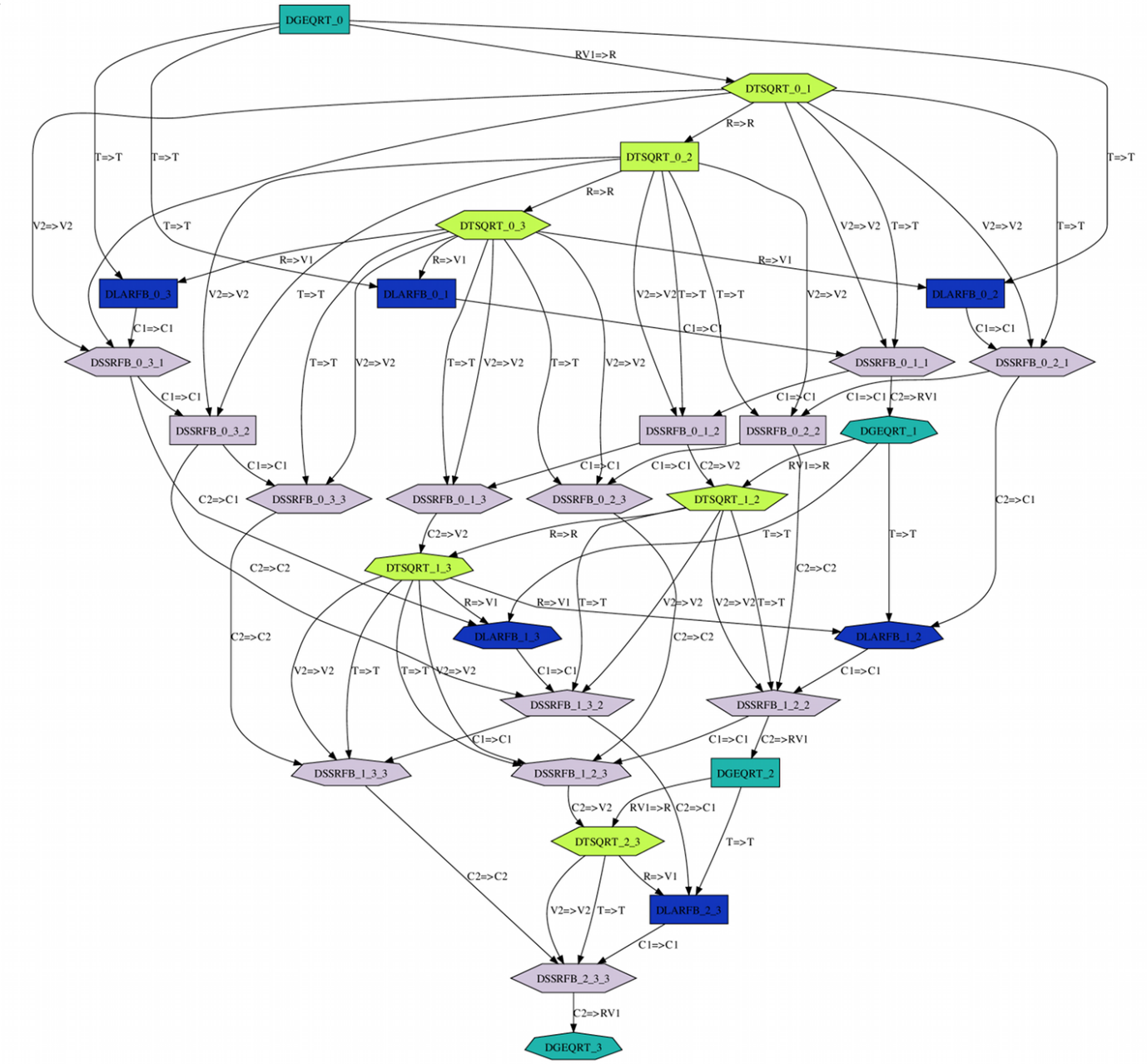
Anthony Danalis

UTK/ORNL

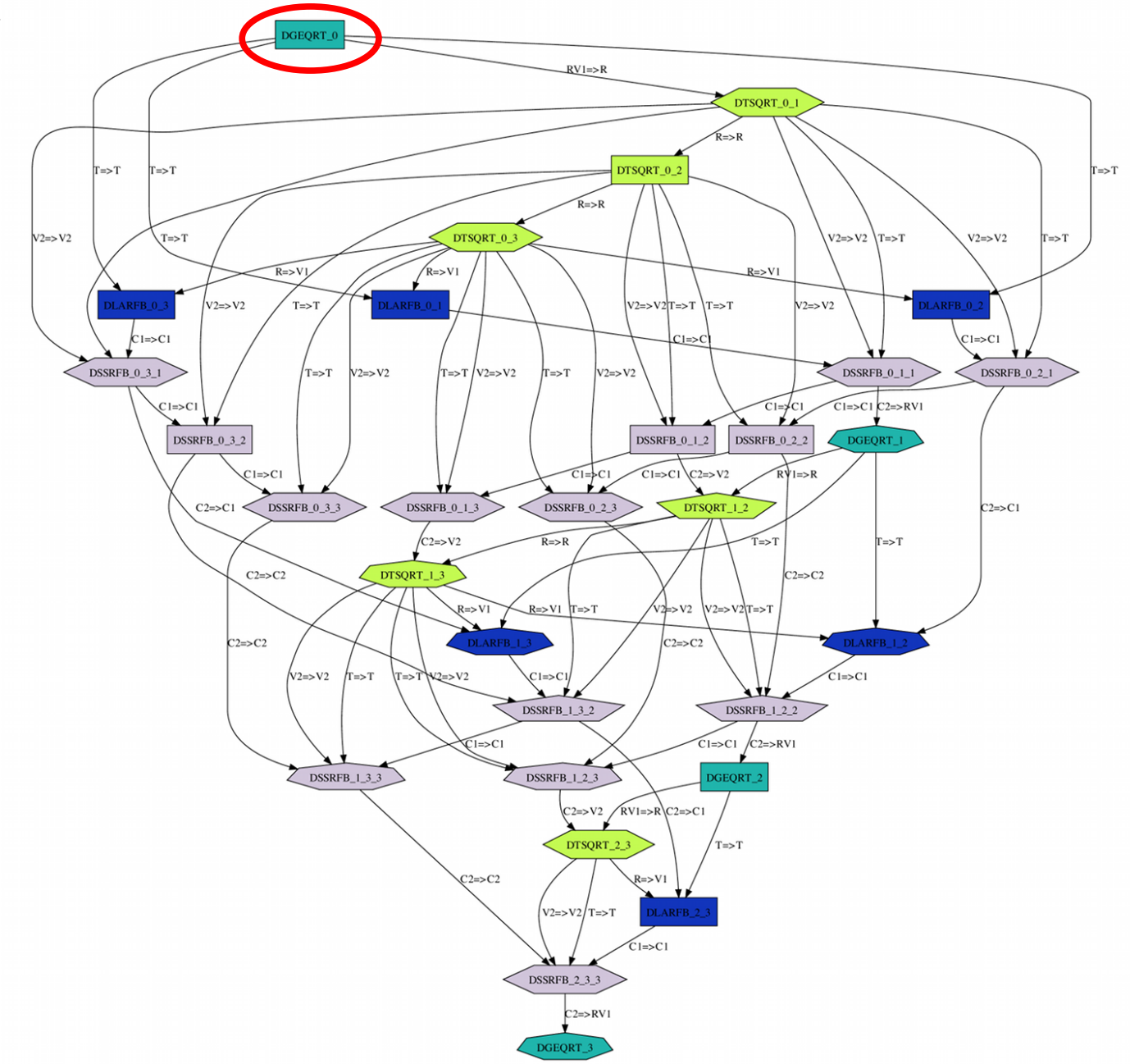
CCGSC 2010

EuroMPI 2011: Santorini, Greece

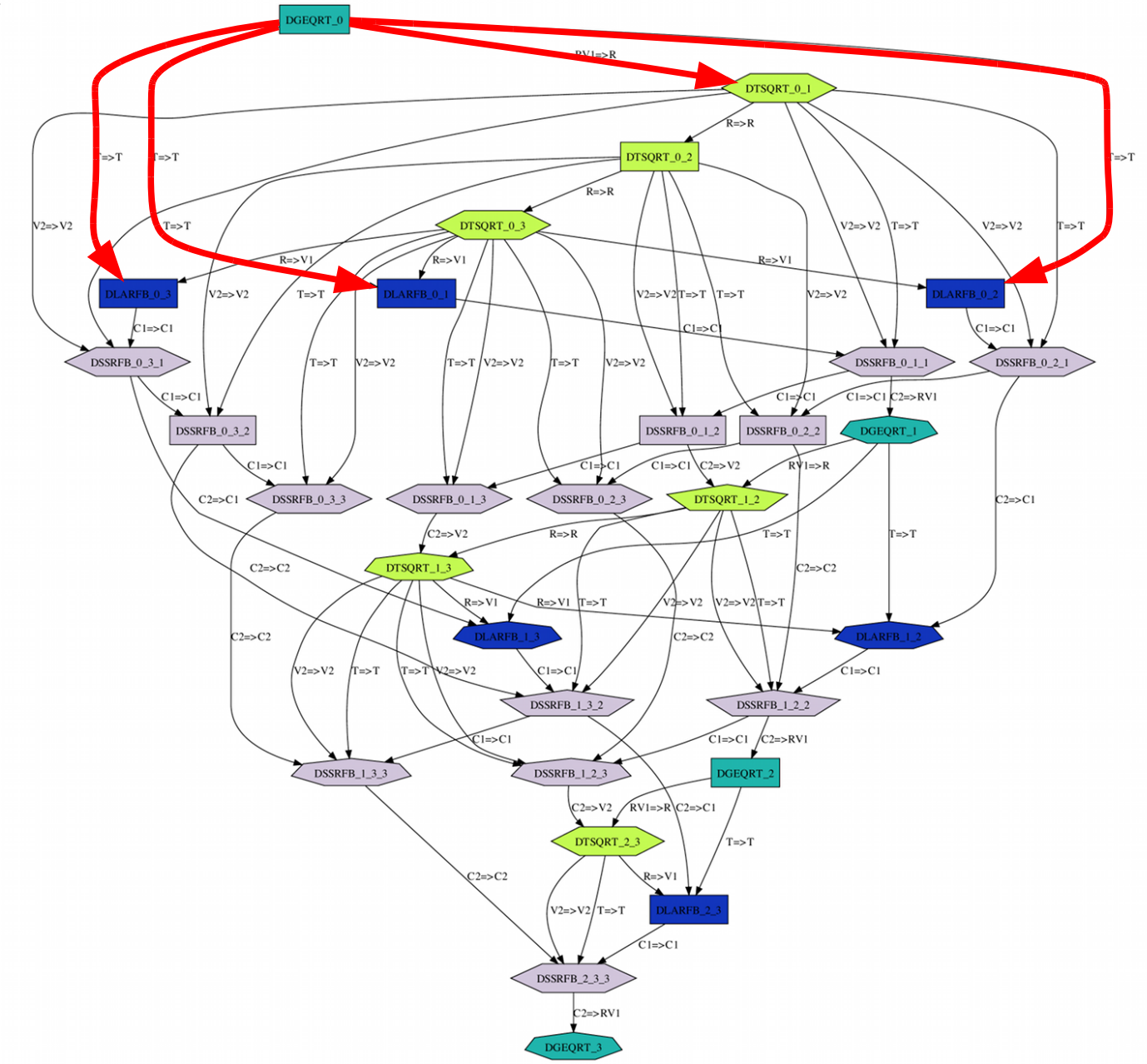
What is DAG Scheduling?



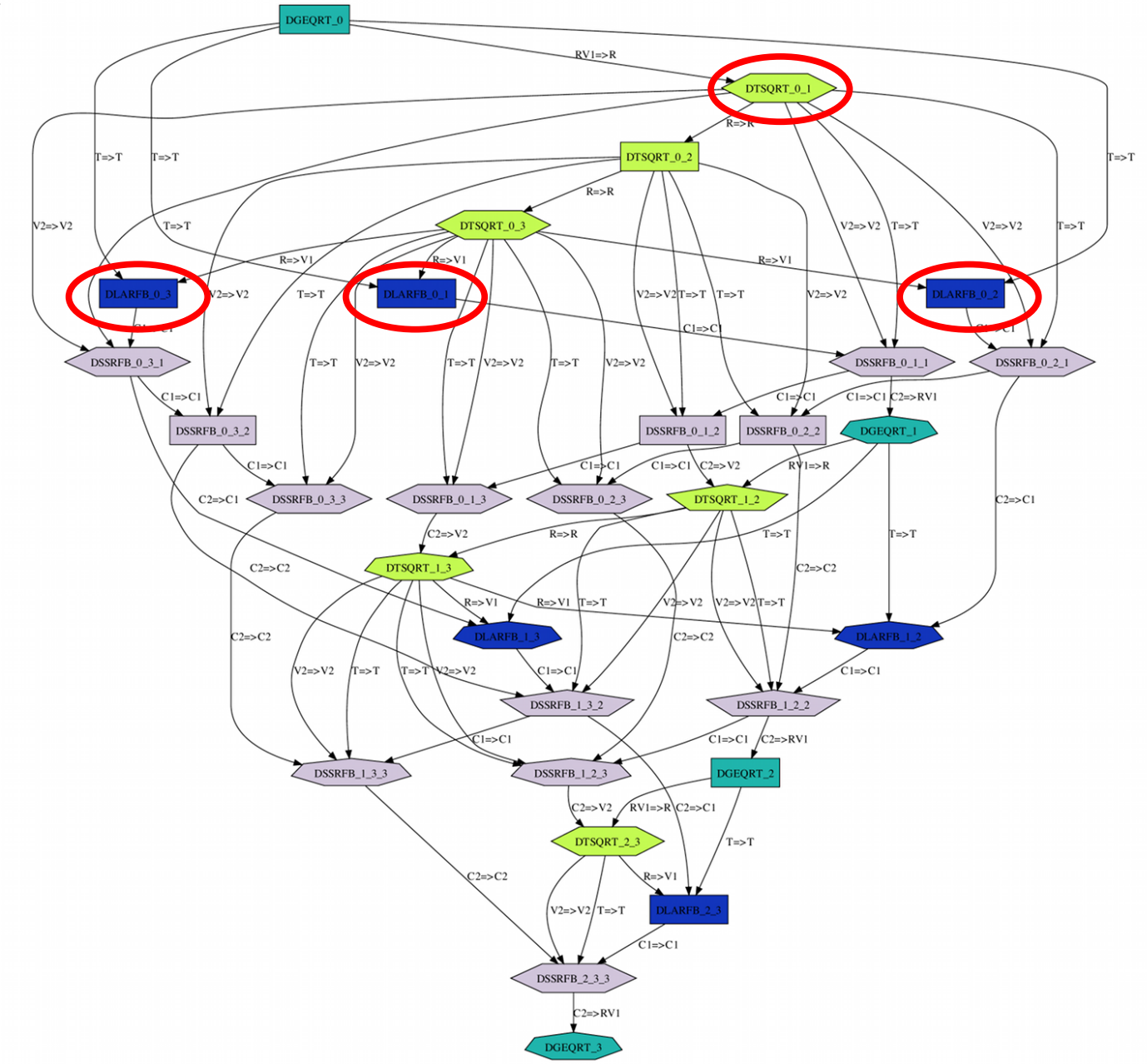
What is DAG Scheduling?



What is DAG Scheduling?



What is DAG Scheduling?



Distributed DAG Scheduling

DAGuE:

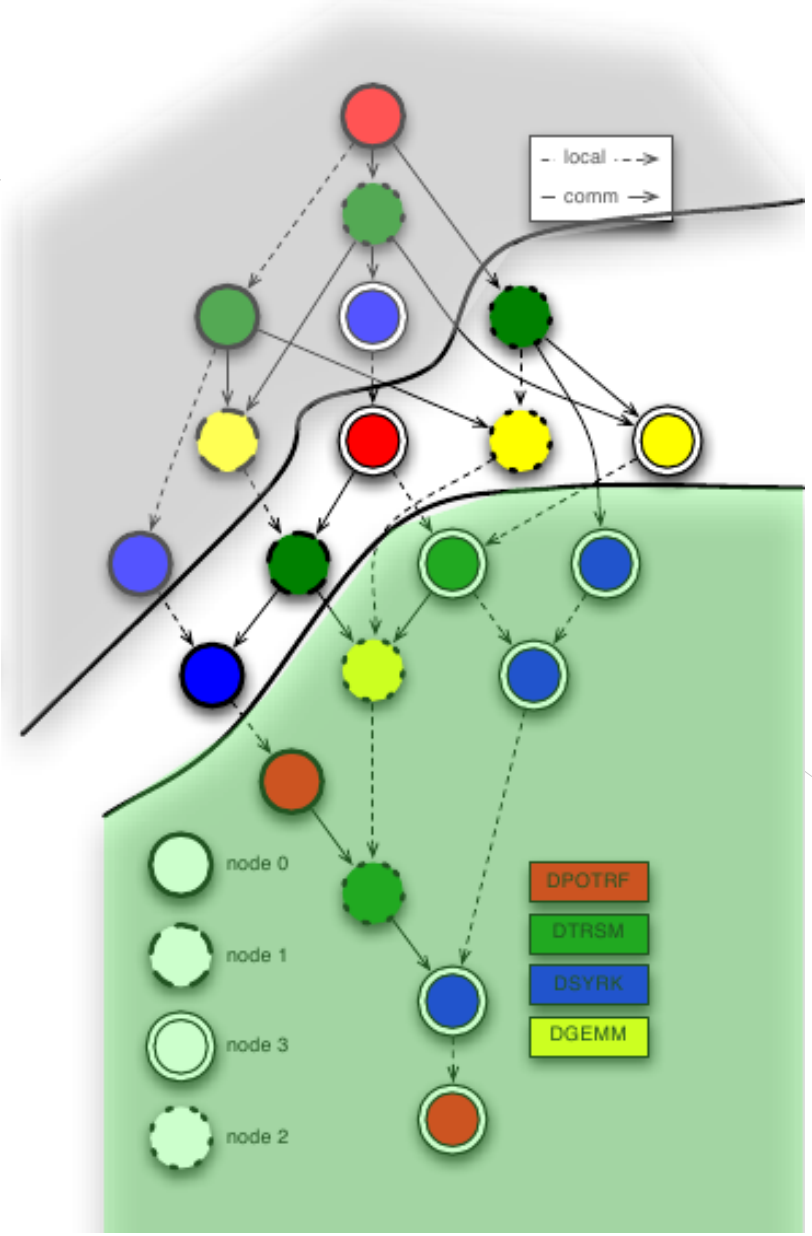
Fully Distributed Dynamic Micro-Tasks Scheduling for Multi-Core High Performance Computing

Distributed DAG Scheduling

DAGuE:

Fully Distributed Dynamic Micro-Tasks Scheduling for Multi-Core High Performance Computing

Distributed DAG Scheduling



Distributed Scheduling Approaches

- Centralized scheduling decisions
- Distributed fully unrolled DAG information
- Distributed concise representation of DAG

Data Flow to Symbolic DAG

```
for (k = 0 .. SIZE-1){
  A[k][k], T[k][k] ← DGEQRT(A[k][k])
  for (m = k+1 .. SIZE-1){
    A[k][k], A[m][k], T[m][k] ←
          DTSQRT(A[k][k], A[m][k], T[m][k])
  }
  for (n = k+1 .. SIZE-1){
    A[k][n] ← DORMQR(A[k][k], T[k][k], A[k][n])
    for (m = k+1 .. SIZE-1){
      A[k][n], A[m][n] ←
          DSSMQR(A[m][k], T[m][k], A[k][n],A[m][n])
    }
  }
}
```

Data Flow to Symbolic DAG

```
for (k = 0 .. SIZE-1){
  A[k][k], T[k][k] ← DGEQRT(A[k][k])
  for (m = k+1 .. SIZE-1){
    A[k][k], A[m][k], T[m][k] ←
          DTSQRT(A[k][k], A[m][k], T[m][k])
  }
  for (n = k+1 .. SIZE-1){
    A[k][n] ← DORMQR(A[k][k], T[k][k], A[k][n])
    for (m = k+1 .. SIZE-1){
      A[k][n], A[m][n] ←
          DSSMQR(A[m][k], T[m][k], A[k][n],A[m][n])
    }
  }
}
```

DGEQRT(k)

DTSQRT(k,m)

DORMQR(k,n)

DSSMQR(k,n,m)

Data Flow to Symbolic DAG

```
for (k = 0 .. SIZE-1){
  A[k][k], T[k][k] ← DGEQRT(A[k][k])
  for (m = k+1 .. SIZE-1){
    A[k][k], A[m][k], T[m][k] ←
          DTSQRT(A[k][k], A[m][k], T[m][k])
  }
  for (n = k+1 .. SIZE-1){
    A[k][n] ← DORMQR(A[k][k], T[k][k], A[k][n])
    for (m = k+1 .. SIZE-1){
      A[k][n], A[m][n] ←
          DSSMQR(A[m][k], T[m][k], A[k][n],A[m][n])
    }
  }
}
```

Data Flow to Symbolic DAG

```
for (k = 0 .. SIZE-1){
  A[k][k], T[k][k] ← DGEQRT(A[k][k])
  for (m = k+1 .. SIZE-1){
    A[k][k], A[m][k], T[m][k] ←
          DTSQRT(A[k][k], A[m][k], T[m][k])
  }
  for (n = k+1 .. SIZE-1){
    A[k][n] ← DORMQR(A[k][k], T[k][k], A[k][n])
    for (m = k+1 .. SIZE-1){
      A[k][n], A[m][n] ←
          DSSMQR(A[m][k], T[m][k], A[k][n],A[m][n])
    }
  }
}
```

k == SIZE-1

Data Flow to Symbolic DAG

```
for (k = 0 .. SIZE-1){
  A[k][k], T[k][k] ← DGEQRT(A[k][k])
  for (m = k+1 .. SIZE-1){
    A[k][k], A[m][k], T[m][k] ←
          DTSQRT(A[k][k], A[m][k], T[m][k])
  }
  for (n = k+1 .. SIZE-1){
    A[k][n] ← DORMQR(A[k][k], T[k][k], A[k][n])
    for (m = k+1 .. SIZE-1){
      A[k][n], A[m][n] ←
          DSSMQR(A[m][k], T[m][k], A[k][n],A[m][n])
    }
  }
}
```

k == SIZE-1

Data Flow to Symbolic DAG

```
for (k = 0 .. SIZE-1){
  A[k][k], T[k][k] ← DGEQRT(A[k][k])
  for (m = k+1 .. SIZE-1){
    A[k][k], A[m][k], T[m][k] ←
        DTSQRT(A[k][k], A[m][k], T[m][k])
  }
  for (n = k+1 .. SIZE-1){
    A[k][n] ← DORMQR(A[k][k], T[k][k], A[k][n])
    for (m = k+1 .. SIZE-1){
      A[k][n], A[m][n] ←
          DSSMQR(A[m][k], T[m][k], A[k][n],A[m][n])
    }
  }
}
```

k == SIZE-1

Data Flow to Symbolic DAG

```
for (k = 0 .. SIZE-1){
  A[k][k], T[k][k] ← DGEQRT(A[k][k])
  for (m = k+1 .. SIZE-1){
    A[k][k], A[m][k], T[m][k] ←
          DTSQRT(A[k][k], A[m][k], T[m][k])
  }
```

DGEQRT(k):A[k][k] -> DTSQRT(k,m)

Data Flow to Symbolic DAG

```
for (k = 0 .. SIZE-1){
  A[k][k], T[k][k] ← DGEQRT(A[k][k])
  for (m = k+1 .. SIZE-1){
    A[k][k], A[m][k], T[m][k] ←
          DTSQRT(A[k][k], A[m][k], T[m][k])
  }
```

m > k+1

DGEQRT(k):A[k][k] -> DTSQRT(k,m)

Data Flow to Symbolic DAG

```
for (k = 0 .. SIZE-1){
  A[k][k], T[k][k] ← DGEQRT(A[k][k])
  for (m = k+1 .. SIZE-1){
    A[k][k], A[m][k], T[m][k] ←
          DTSQRT(A[k][k], A[m][k], T[m][k])
  }
```

m > k+1

m == k+1

DGEQRT(k):A[k][k] -> DTSQRT(k,m)

Data Flow to Symbolic DAG

```
for (k = 0 .. SIZE-1){
  A[k][k], T[k][k] ← DGEQRT(A[k][k])
  for (m = k+1 .. SIZE-1){
    A[k][k], A[m][k], T[m][k] ←
          DTSQRT(A[k][k], A[m][k], T[m][k])
  }
```

m > k+1

DGEQRT(k):A[k][k] -> DTSQRT(k,k+1)

Data Flow to Symbolic DAG

```
for (k = 0 .. SIZE-1){
  A[k][k], T[k][k] ← DGEQRT(A[k][k])
  for (m = k+1 .. SIZE-1){
    A[k][k], A[m][k], T[m][k] ←
          DTSQRT(A[k][k], A[m][k], T[m][k])
  }
```

m > k+1

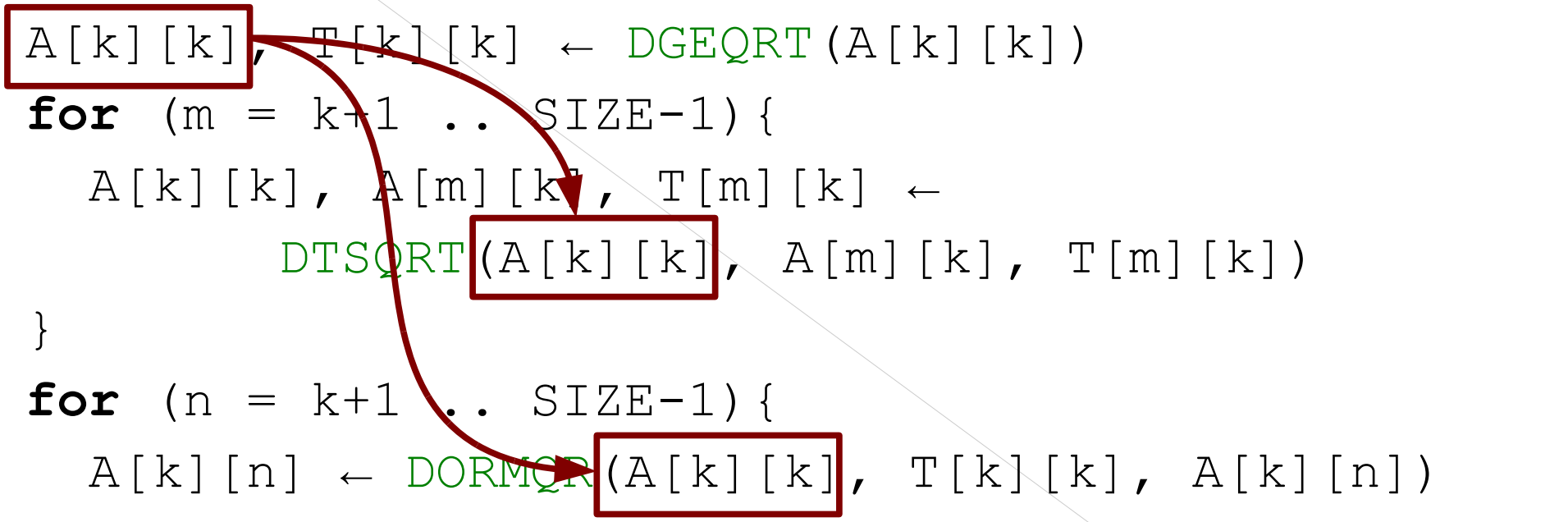
DGEQRT(k):

A[k][k] -> DTSQRT(k,k+1) (k<SIZE-1)

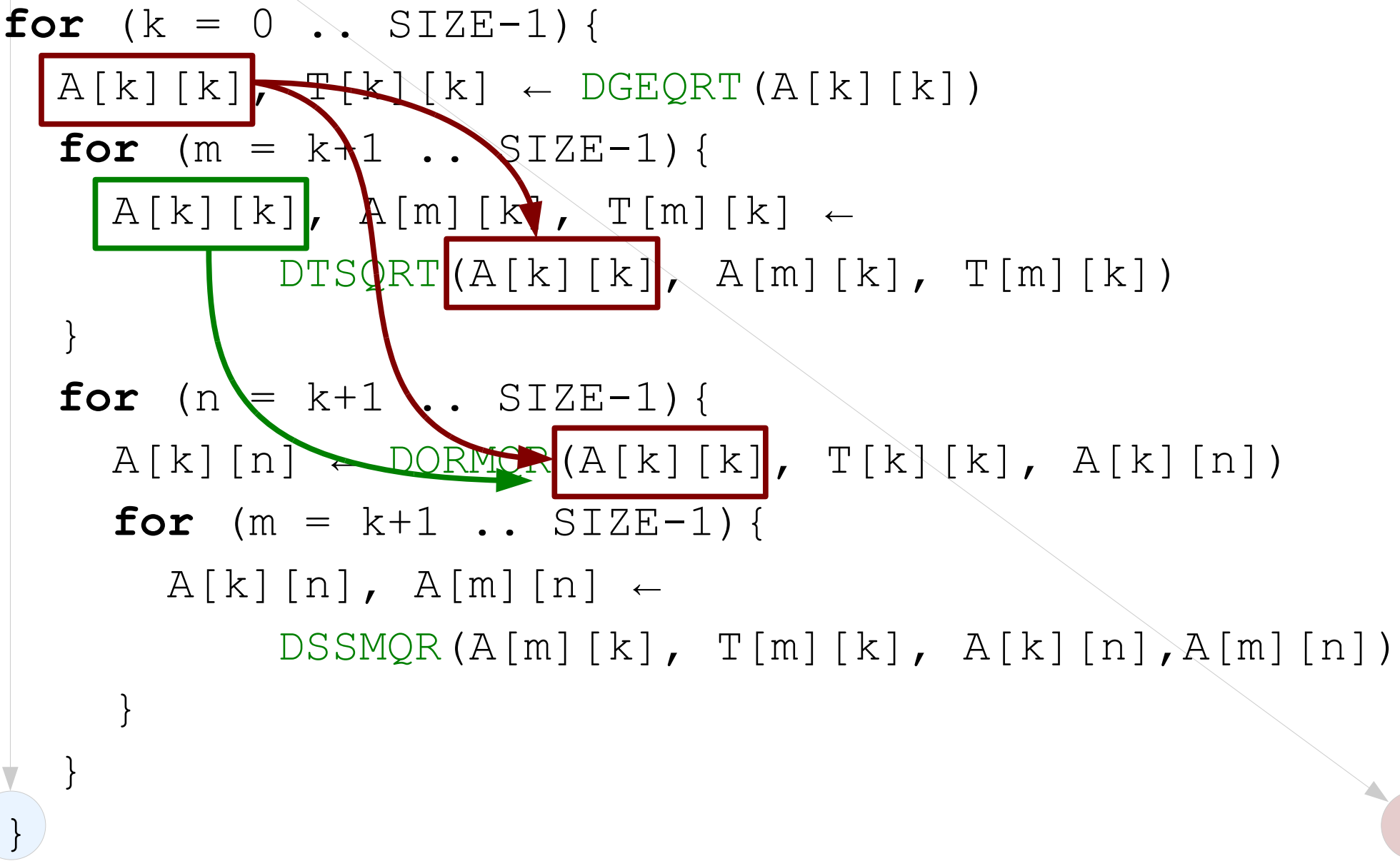
-> A[k][k] (k==SIZE-1)

Data Flow to Symbolic DAG

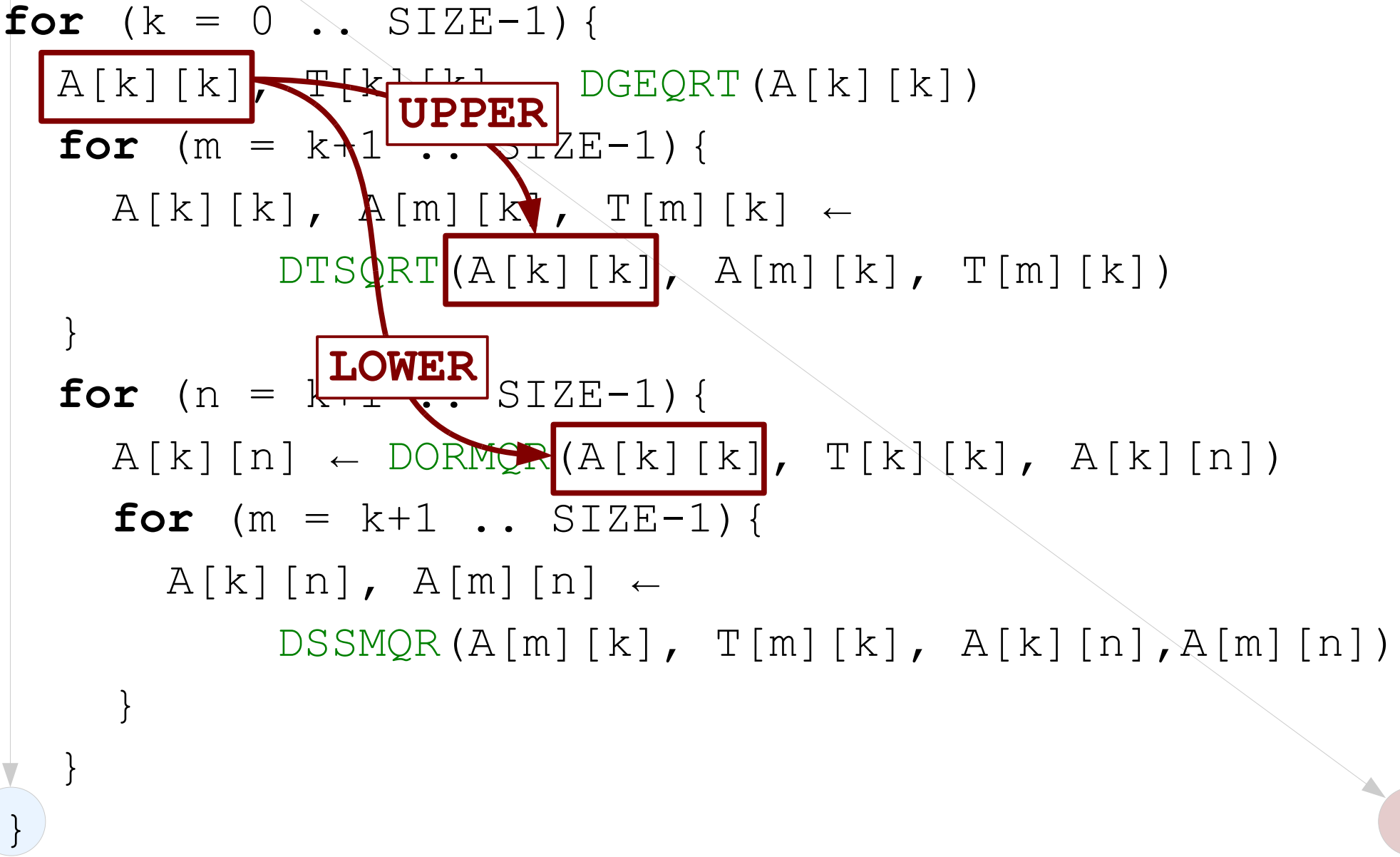
```
for (k = 0 .. SIZE-1){
  A[k][k], T[k][k] ← DGEQRT(A[k][k])
  for (m = k+1 .. SIZE-1){
    A[k][k], A[m][k], T[m][k] ←
          DTSQRT(A[k][k], A[m][k], T[m][k])
  }
  for (n = k+1 .. SIZE-1){
    A[k][n] ← DORMQR(A[k][k], T[k][k], A[k][n])
    for (m = k+1 .. SIZE-1){
      A[k][n], A[m][n] ←
          DSSMQR(A[m][k], T[m][k], A[k][n],A[m][n])
    }
  }
}
```



Data Flow to Symbolic DAG



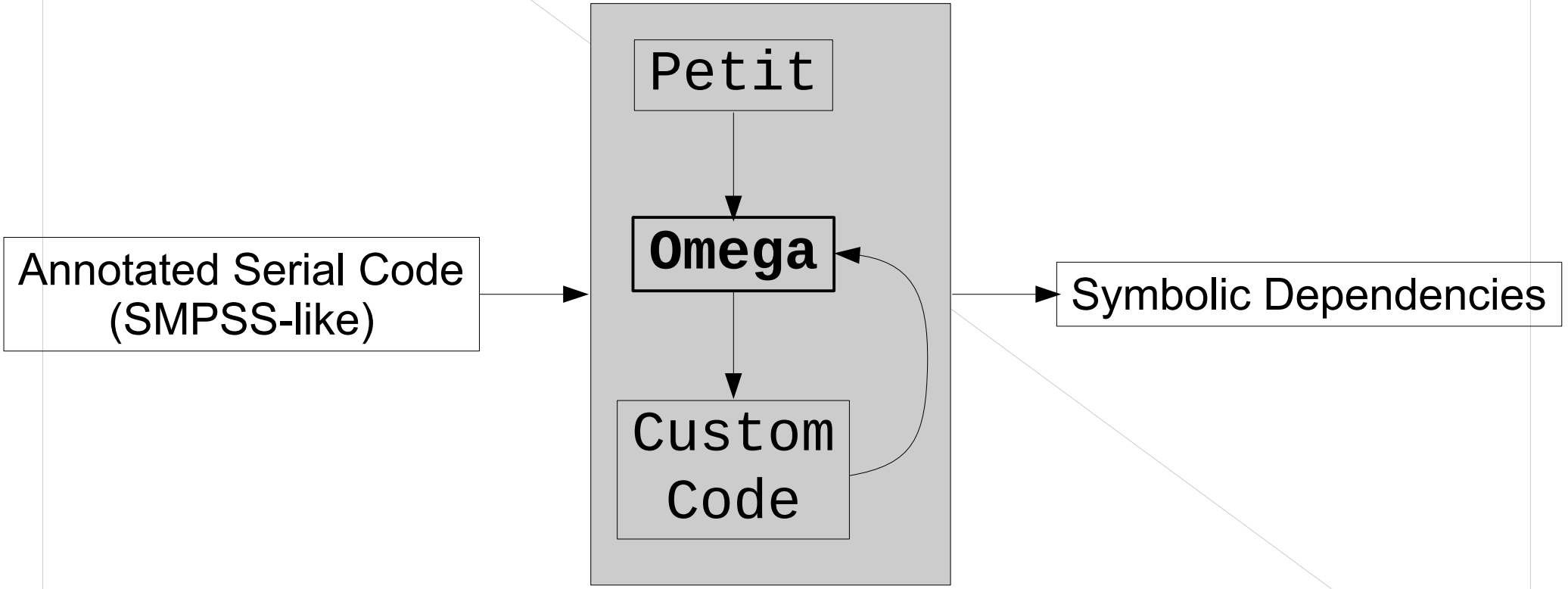
Data Flow to Symbolic DAG



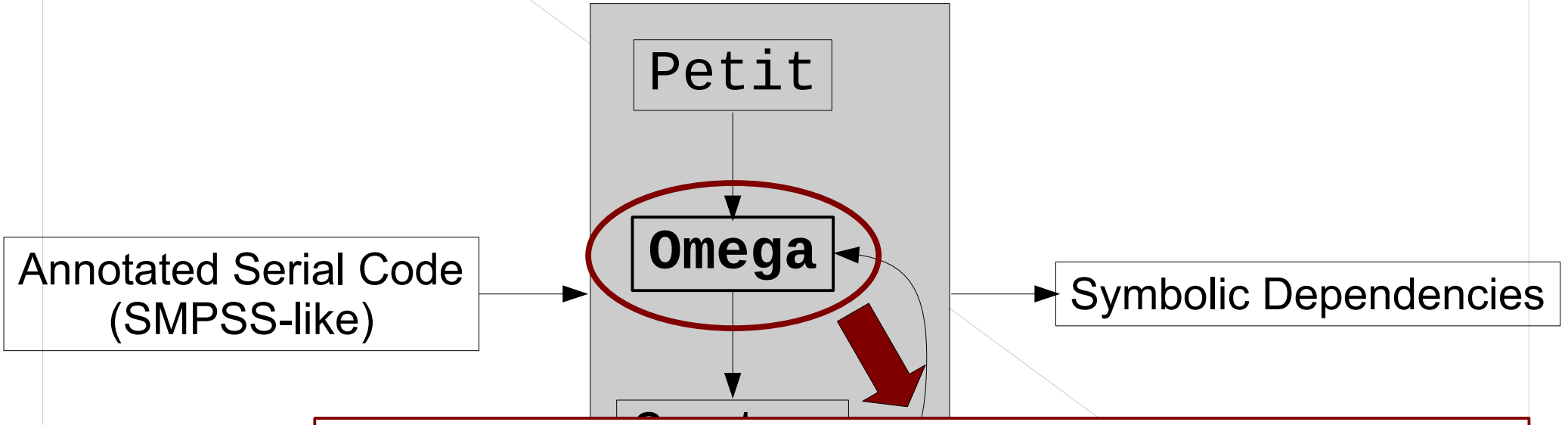
Job Data Flow (JDF)

```
DGEQRT(k) executes on A(k,k)
  k = 0..SIZE-1

  INOUT RV1 <- k==0          ?
                  A(0,0) : C2 DSSMQR(k-1,k,k)
            -> k==SIZE-1 ?
                  A(k,k) : R DTSQRT(k,k+1)
            -> k==SIZE-1 ?
                  A(k,k) : V1 DORMQR(k,k+1..SIZE-1)

  OUT T     -> T DORMQR(k,k+1..SIZE-1)
            -> T(k,k)
```

JDF Generation Toolchain



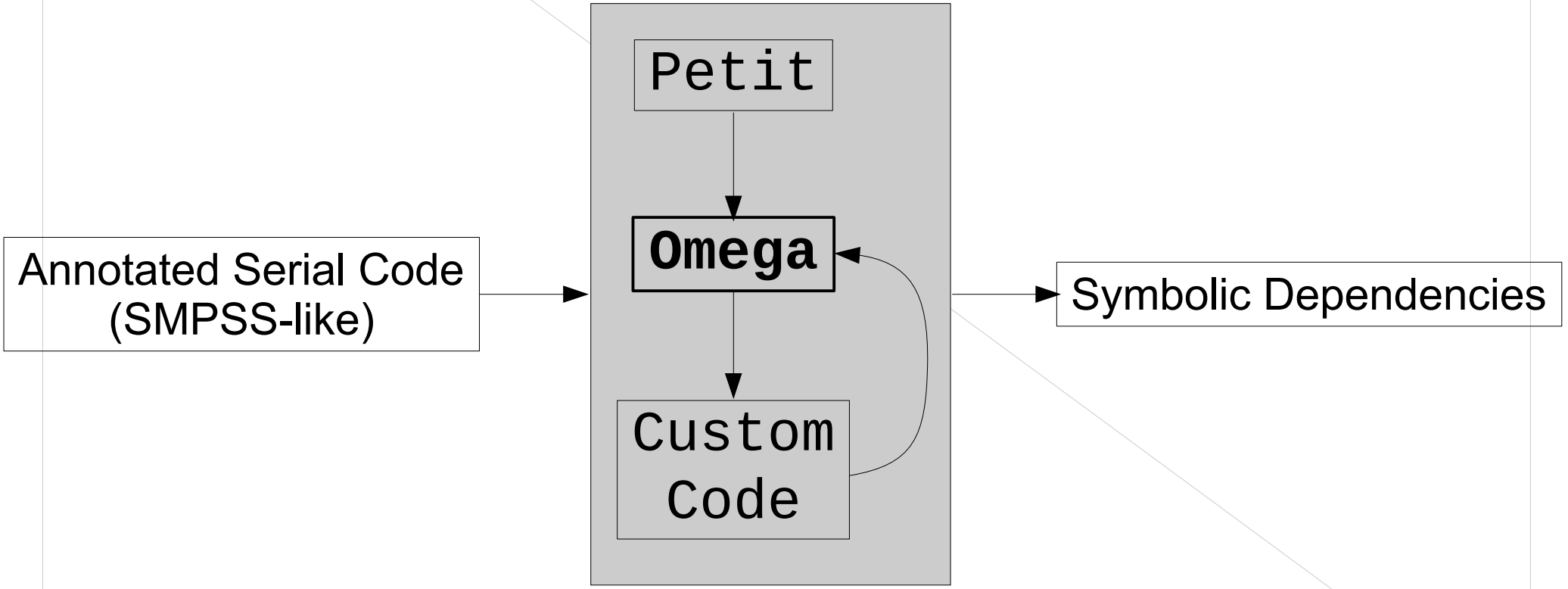
JDF Generation Toolchain



R1 := {[k] -> [k,m] : 0 <= k < BB && k+1 <= m <= BB-1};
{[k] -> [k,m] : 0 <= k < m < BB}

R2 := {[k] -> [k,m] : 0 <= k < BB && k+1< m < 0};
{[k] -> [k',m] : FALSE }

JDF Generation Toolchain



Toolchain limitations

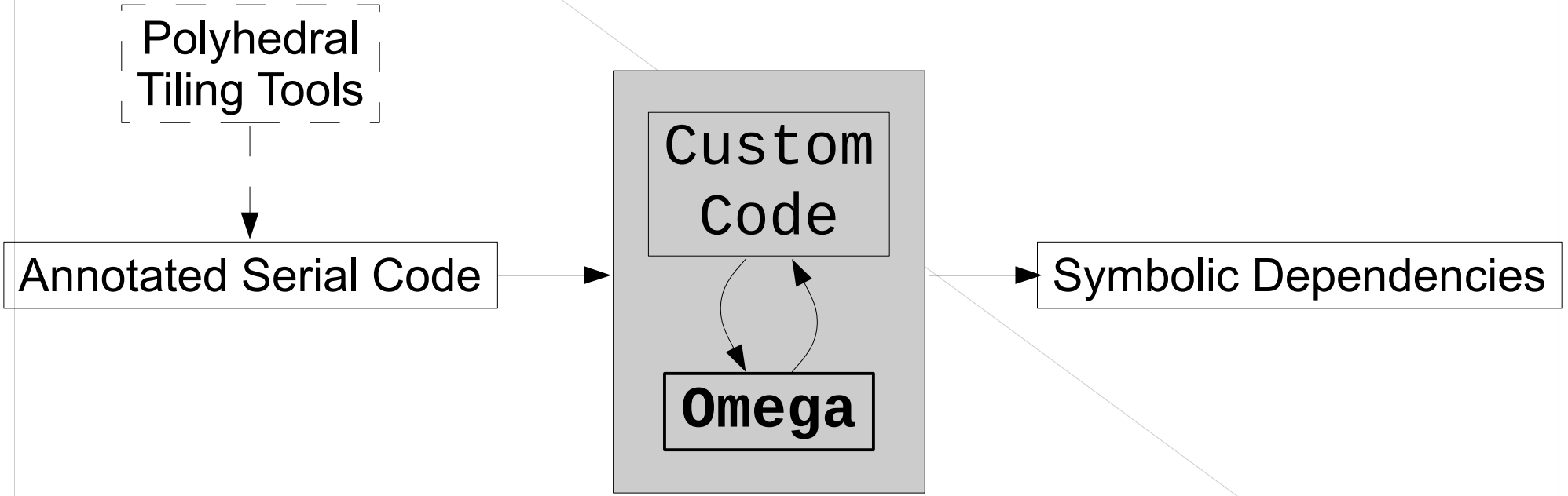
Petit limitations:

- Fortran like pseudo-language syntax
- No notion of Task

Omega limitations:

- Can handle only affine constraints

JDF Generation Toolchain (future)



Beyond Affine Constraints

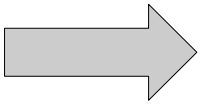
```
for (k = 0; k < BB; k++) {
   for (M = k; M < BB; M += BS) {
      CORE_sgeqrt( ... );

      for (m = M+1; m < M+BS && m < BB; m++) {
         CORE_stsqrt( ... );
      }

      for (n = k+1; n < BB; n++) {
         CORE_sormqr( ... );
      }
   }

   for (M = k; M < BB; M += BS) {
      for (n = k+1; n < BB; n++) {
         for (m = M+1; m < M+BS && m < BB; m++) {
            CORE_sssmqr( ... );
         }
      }
   }

   for (SS = BS; SS < BB-k; SS *= 2) {
      for (M = k; M+SS < BB; M+=2*SS) {
         CORE_sttqrt( ... );

         for (n = k+1; n < BB; n++) {
            CORE_sttmqr( ... );
         }
      }
   }
}
```

Beyond Affine Constraints

```
for (k = 0; k < BB; k++) {
   for (M = k; M < BB; M += BS) {
      CORE_sgeqrt( ... );

      for (m = M+1; m < M+BS && m < BB; m++) {
         CORE_stsqrt( ... );
      }

      for (n = k+1; n < BB; n++) {
         CORE_sormqr( ... );
      }
   }

   for (M = k; M < BB; M += BS) {
      for (n = k+1; n < BB; n++) {
         for (m = M+1; m < M+BS && m < BB; m++) {
            CORE_sssmqr( ... );
         }
      }
   }

   for (SS = BS; SS < BB-k; SS *= 2) {
      for (M = k; M+SS < BB; M+=2*SS) {
         CORE_sttqrt( ... );

         for (n = k+1; n < BB; n++) {
            CORE_sttmqr( ... );
         }
      }
   }
}
```

Beyond Affine Constraints

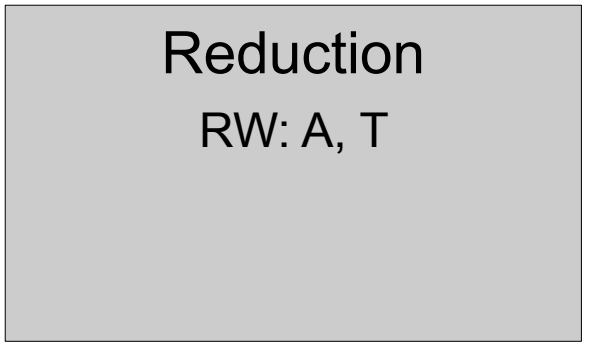
```
for (k = 0; k < BB; k++) {
   for (M = k; M < BB; M += BS) {
      CORE_sgeqrt( ... );

      for (m = M+1; m < M+BS && m < BB; m++) {
         CORE_stsqrt( ... );
      }

      for (n = k+1; n < BB; n++) {
         CORE_sormqr( ... );
      }
   }

   for (M = k; M < BB; M += BS) {
      for (n = k+1; n < BB; n++) {
         for (m = M+1; m < M+BS && m < BB; m++) {
            CORE_sssmqr( ... );
         }
      }
   }

   for (SS = BS; SS < BB-k; SS *= 2) {
      for (M = k; M+SS < BB; M+=2*SS) {
         CORE_sttqrt( ... );

         for (n = k+1; n < BB; n++) {
            CORE_sttmqr( ... );
         }
      }
   }
}
```



Reduction

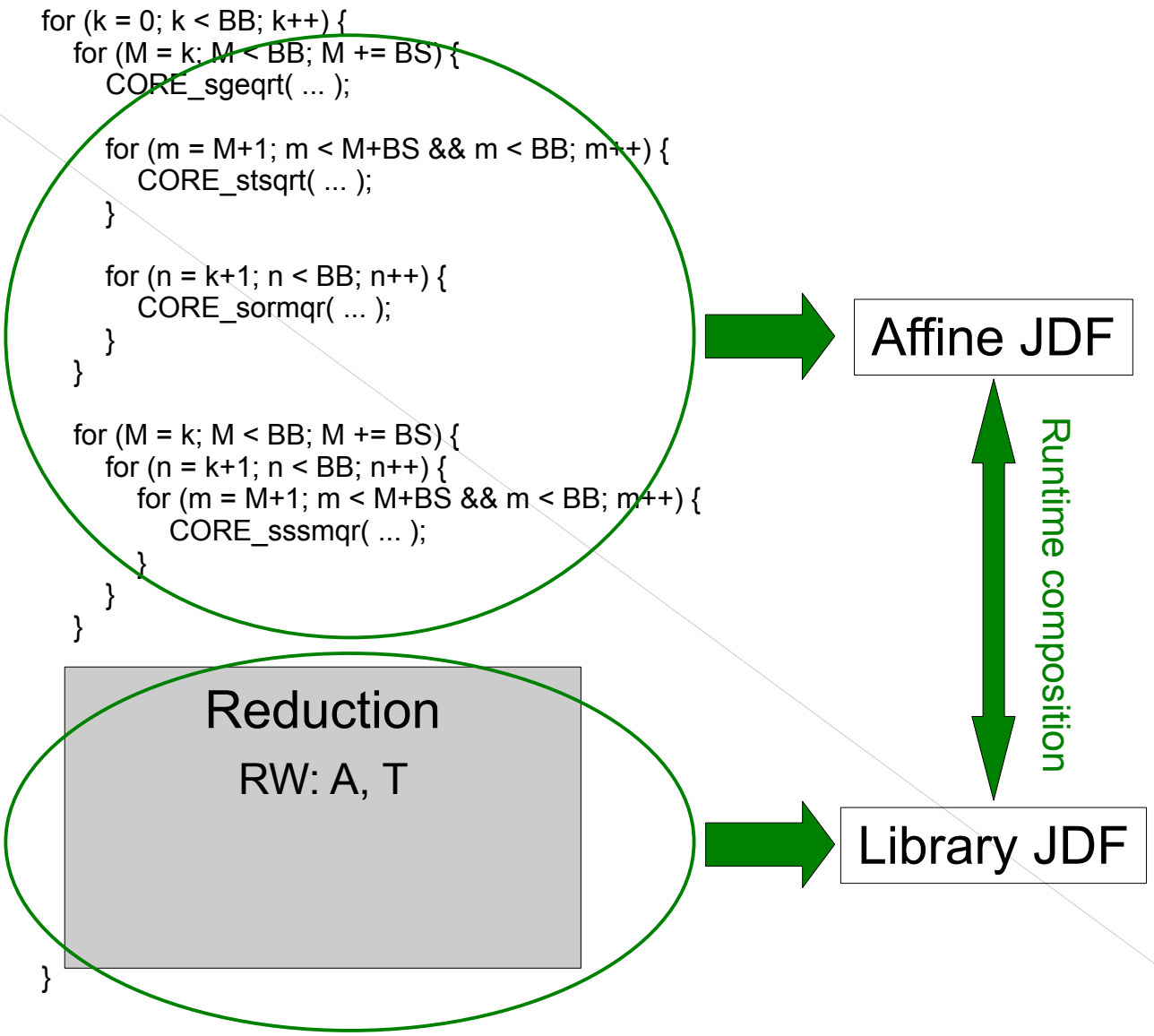
Beyond Affine Constraints

```
for (k = 0; k < BB; k++) {
   for (M = k; M < BB; M += BS) {
      CORE_sgeqrt( ... );

      for (m = M+1; m < M+BS && m < BB; m++) {
         CORE_stsqrt( ... );
      }

      for (n = k+1; n < BB; n++) {
         CORE_sormqr( ... );
      }
   }

   for (M = k; M < BB; M += BS) {
      for (n = k+1; n < BB; n++) {
         for (m = M+1; m < M+BS && m < BB; m++) {
            CORE_sssmqr( ... );
         }
      }
   }
```




```
}
```

Beyond Affine Constraints



Conclusions

- Distributed DAG Scheduling is promising
- DAGuE is a fully Distributed DAG Sc. Engine
- Symbolic Representation (JDF) from Data Flow
- Potential auto-parallelization extensions
- Traditional Data Flow limitations (maybe) surmountable