

From Data-Flow to Distributed DAG Scheduling

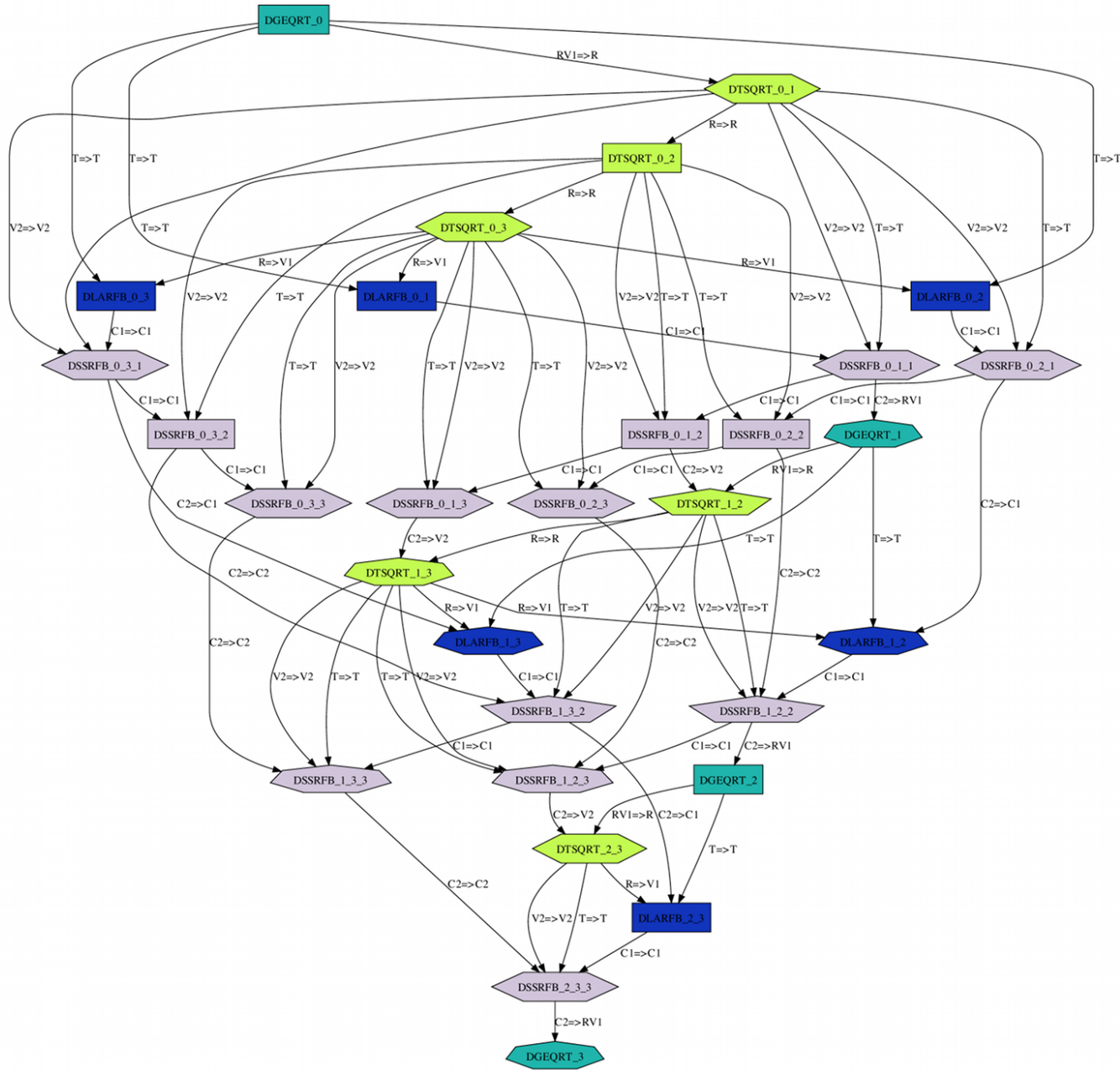
Anthony Danalis
UTK/ORNL

CCGSC 2010

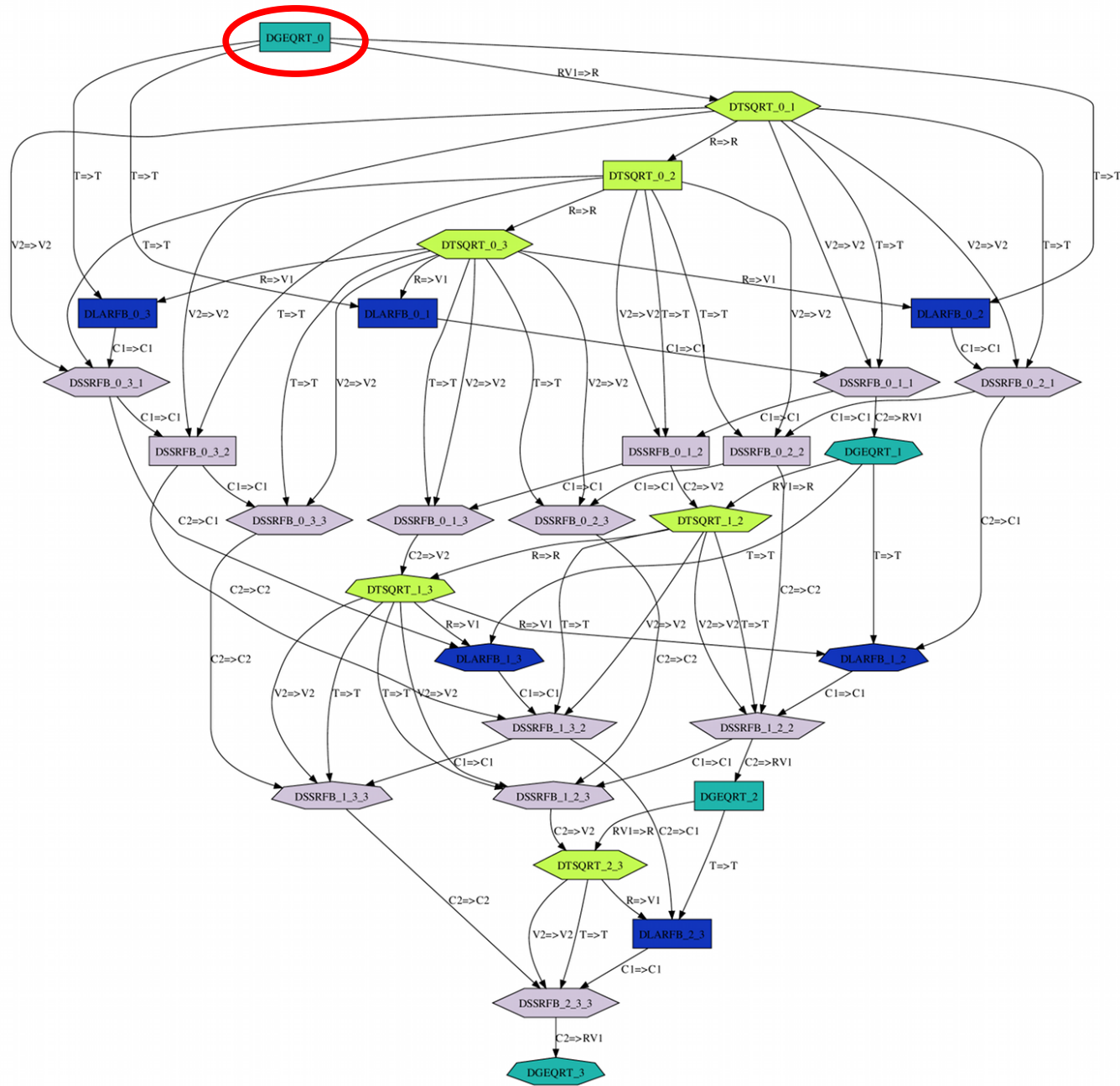
EuroMPI 2011: Santorini, Greece



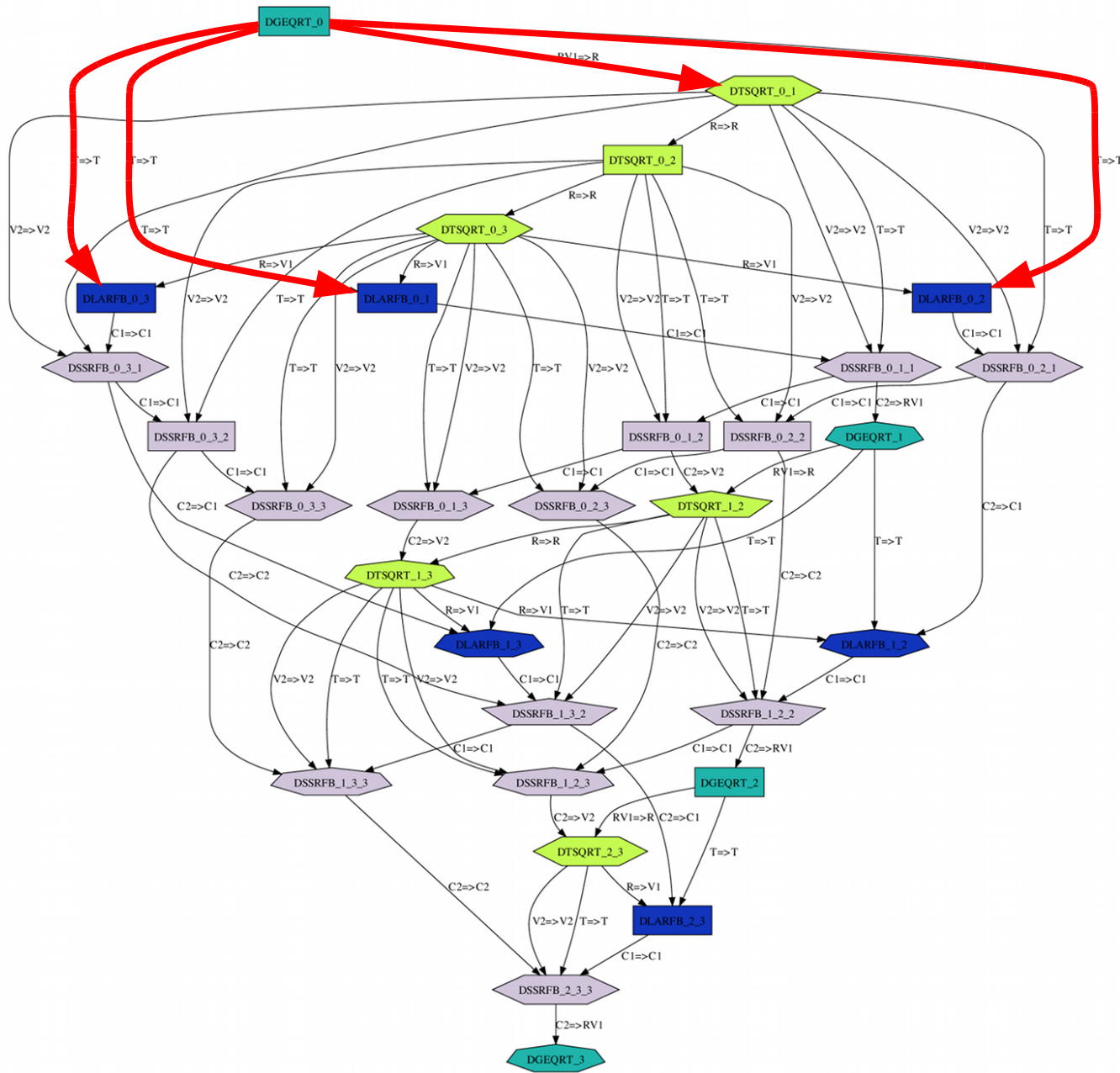
What is DAG Scheduling?



What is DAG Scheduling?



What is DAG Scheduling?





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 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])  
    }  
  }  
}
```

The diagram illustrates the data flow from the code to symbolic DAG nodes. A green circle at the top left is connected by a vertical line to a blue circle at the bottom left. A yellow circle at the top right is connected by a vertical line to a red circle at the bottom right. A diagonal line also connects the green circle to the red circle. The code is positioned between these circles, with the left side of the code aligned with the left vertical line and the right side aligned with the right vertical line.

Data Flow to Symbolic DAG

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

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])  
  }  
}
```

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

A[k][k]

A[k][k]

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])  
  }  
}
```

The diagram illustrates the data flow from the `DGEQRT` function to the `DTSQRT` function. A red arrow points from the `DGEQRT(A[k][k])` call to the first argument of `DTSQRT(A[k][k], A[m][k], T[m][k])`. Red boxes highlight the expression `m > k+1` in the inner loop header, the `A[k][k]` argument in the `DTSQRT` call, and the `A[m][k]` argument in the `DTSQRT` call.

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

```
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])  
    }  
  }  
}
```


Job Data Flow (JDF)

DGEQRT(k) **executes** on $A(k,k)$

$k = 0..SIZE-1$

INOUT RV1 \leftarrow $k==0$?

$A(0,0)$: C2 DSSMQR($k-1,k,k$)

\rightarrow $k==SIZE-1$?

$A(k,k)$: R DTSQRT($k,k+1$)

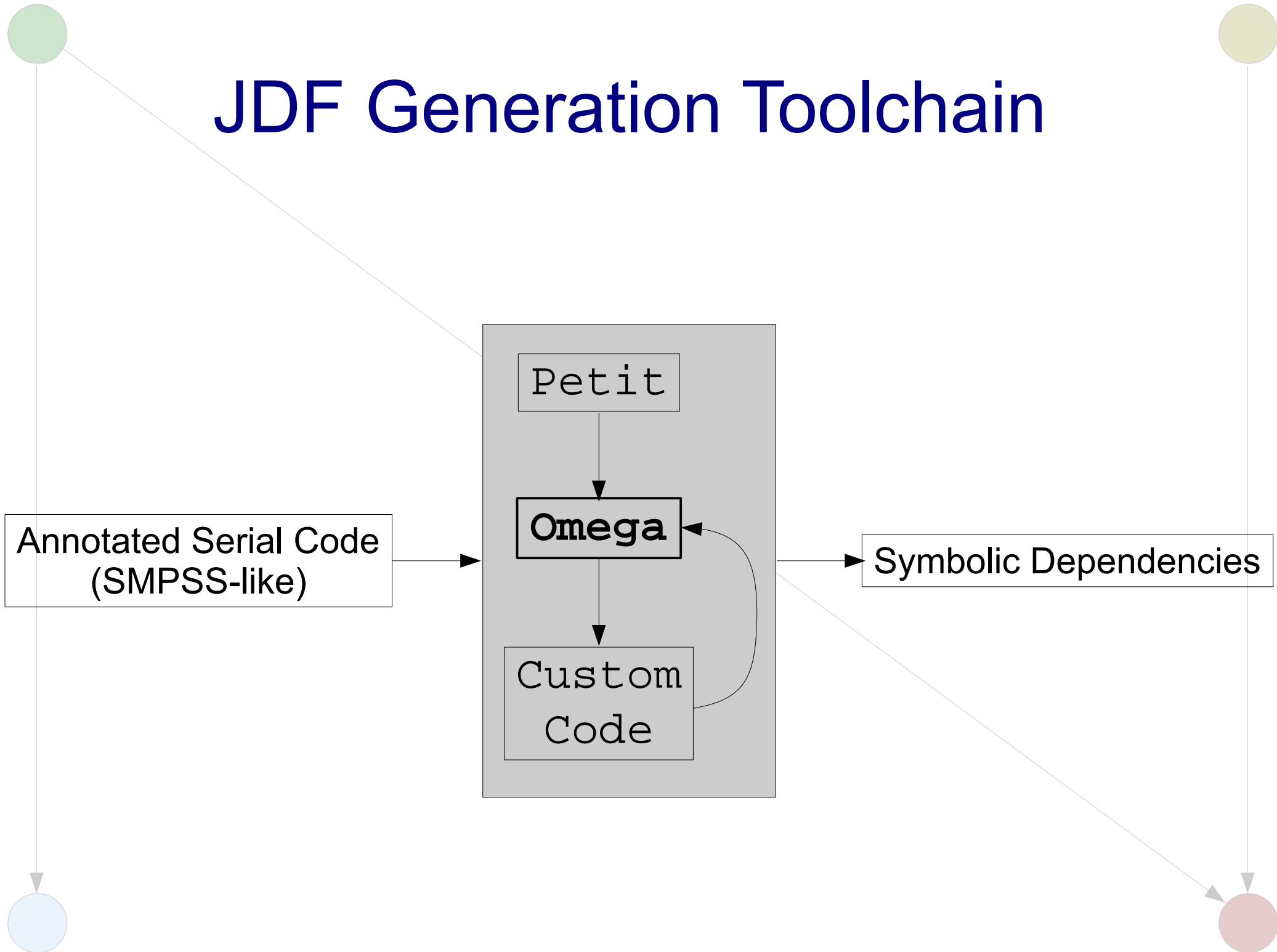
\rightarrow $k==SIZE-1$?

$A(k,k)$: V1 DORMQR($k,k+1..SIZE-1$)

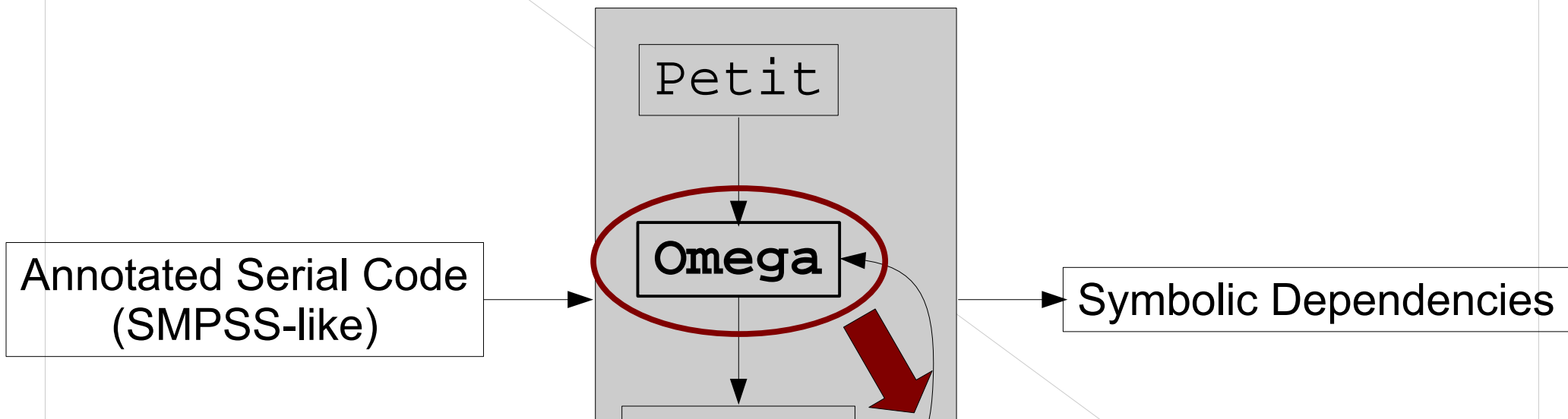
OUT T \rightarrow T DORMQR($k,k+1..SIZE-1$)

\rightarrow T(k,k)

JDF Generation Toolchain



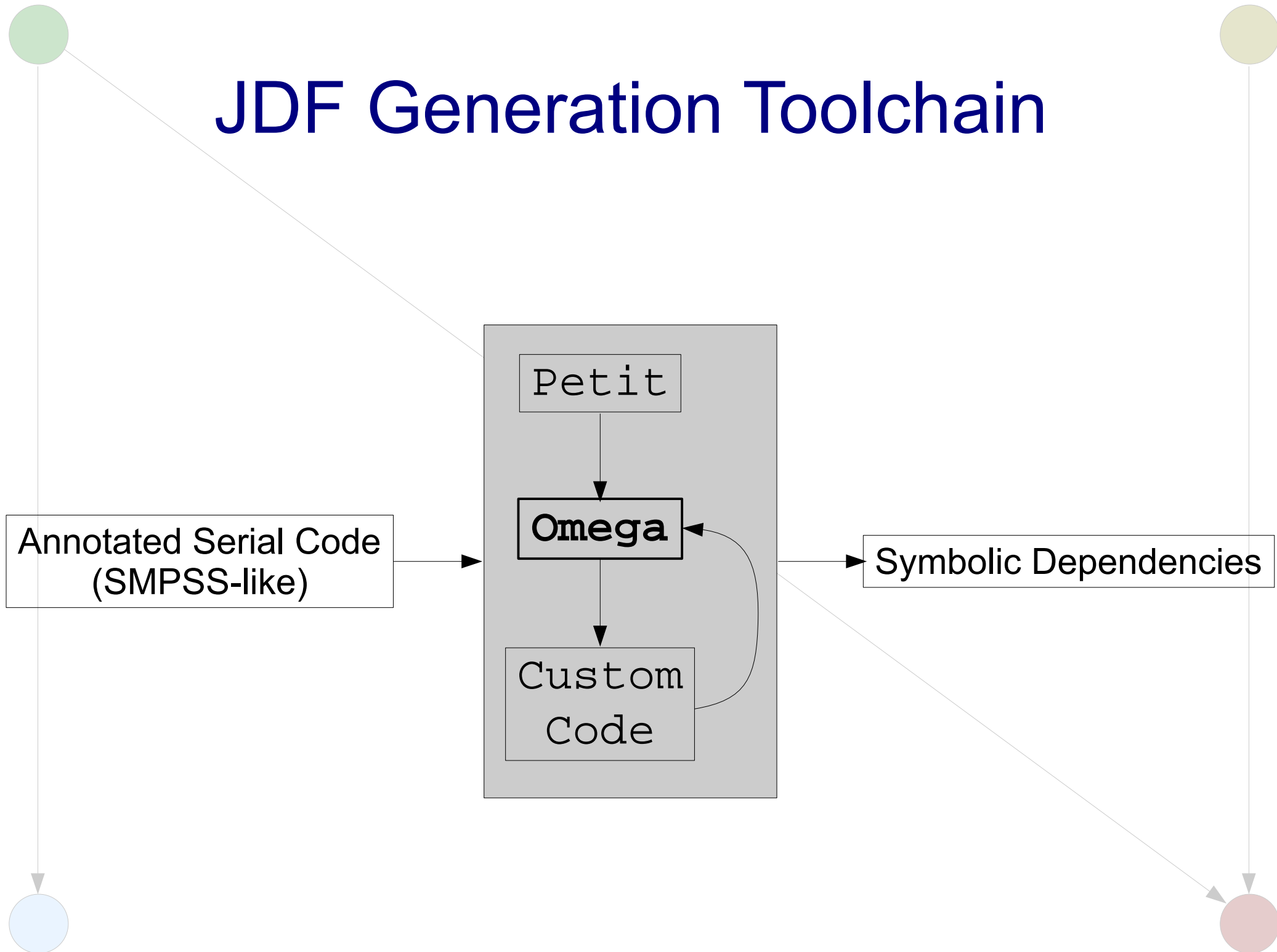
JDF Generation Toolchain



$R1 := \{[k] \rightarrow [k,m] : 0 \leq k < BB \ \&\& \ k+1 \leq m \leq BB-1\};$
 $\{[k] \rightarrow [k,m] : 0 \leq k < m < BB\}$

$R2 := \{[k] \rightarrow [k,m] : 0 \leq k < BB \ \&\& \ k+1 < m < 0\};$
 $\{[k] \rightarrow [k',m] : FALSE \}$

JDF Generation Toolchain





Toolchain limitations

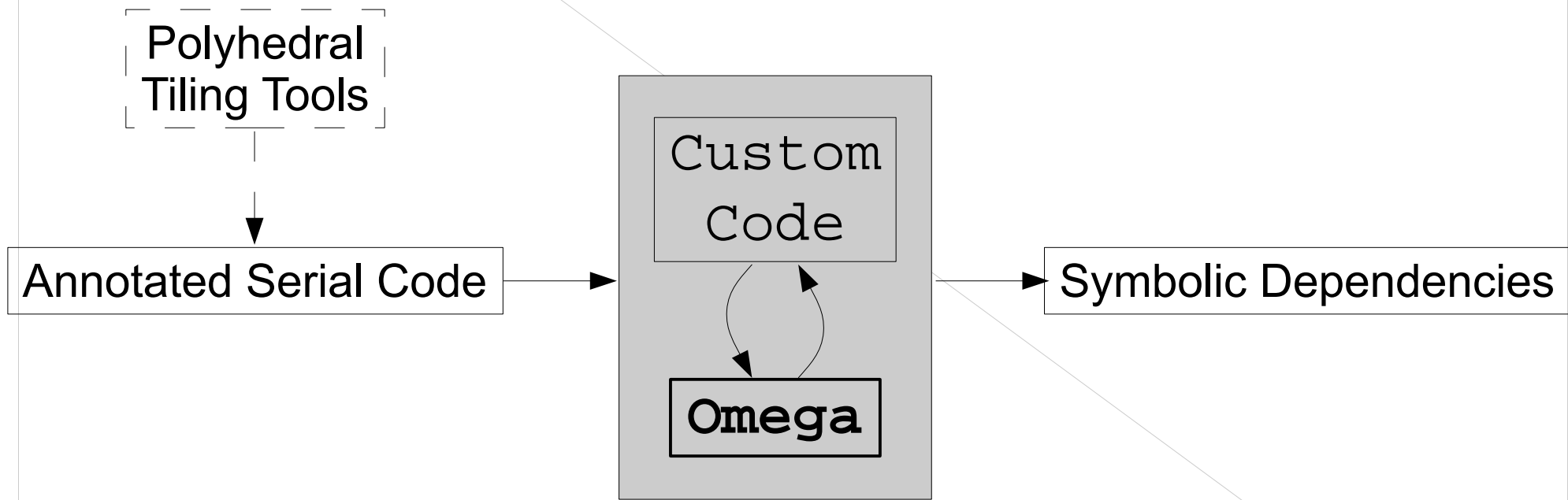
Petit limitations:

- 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( ... );
      }
    }
  }
}
```

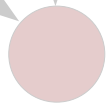
Beyond Affine Constraints

```
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_ssmqr( ... );  
      }  
    }  
  }  
}
```

```
for (SS = BS; SS < BB-k; SS *= 2) {  
  for (M = k; M+SS < BB; M+=2*SS) {  
    CORE_stqrt( ... );  
  
    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_ssmqr( ... );  
      }  
    }  
  }  
}
```

Reduction
RW: A, T

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_stsqr( ... );  
    }  
  
    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_ssmqr( ... );  
      }  
    }  
  }  
}
```

Reduction
RW: A, T

Affine JDF

Library JDF

Runtime composition

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