State variables

state variable: $\varphi = \varphi(ji, jj, jk, jt)$

where: $jt = j_{before},\ j_{now},\ j_{after}$ for leapfrog

$jt = j_{before},\ j_{RHS},\ j_{after}$ for RK3

Note that memory management is logically different in 3-level (leapfrog) and 2-level (RK) schemes:

Leapfrog: j_{after} is loaded first with the trend then with the after state variable. j_{before} and j_{now} are needed for the time step calculations.

RK3: j_{RHS} and j_{after} are updated incrementally for the successive update steps in the timestep while j_{before} remains the same.

Calls to TRA/DYN routines

Pass “out” or “in/out” variables to Level 2 routines:

```
stp:    call dyn_vor( kstp, j_now, uu(j_aft), vv(j_aft) )
        subroutine dyn_vor(kt, jt, pu_rhs, pv_rhs)
            call vor_een( kt, ncor, jt, pu_rhs, pv_rhs) )
            subroutine vor_een( kt, ncor, jt, pu_rhs, pv_rhs )
                pu_rhs = f( uu(jt), vv(jt) ) etc.
```

update: uu(j_aft) = uu(j_bef) + 2.Δt.uu(j_aft)

time level swap: j_bef = j_now ; j_now = j_aft

Transition code

Use of pointers allows us to run with some routines converted and some not.

uu = uu(ji, jj, jk, jt) ; where jt = j_bef, j_now, j_aft

real_wp, pointer, dimension(: , : , :) :: un

un => uu(:, :, :, j_now) *etc.*