# stuff done once per exucution--not depending on r

function InitWRE!(o::StrBootTest{T}) where T

  o.Repl.kZ>1 && (o.numer\_b = Vector{T}(undef,nrows(o.Repl.RRpar)))

  iszero(o.granular) && (o.Repl.Zperp = o.DGP.Zperp = Matrix{T}(undef,0,0))  # drop this potentially large array

  o.liml && o.Repl.kZ==1 && o.Nw==1 && (o.As = o.β̈s = zeros(1, o.B+1))

  o.S✻ZperpU              = [Matrix{T}(undef, o.Repl.kZperp, o.N✻) for \_ ∈ 0:o.Repl.kZ]

  o.invZperpZperpS✻ZperpU = [Matrix{T}(undef, o.Repl.kZperp, o.N✻) for \_ ∈ 0:o.Repl.kZ]

  o.S✻YU                  = [Vector{T}(undef, o.N✻) for \_ ∈ 0:o.Repl.kZ, \_ ∈ 0:o.Repl.kZ]

  o.S✻XU                  = [Matrix{T}(undef, o.DGP.kX, o.N✻) for \_ ∈ 0:o.Repl.kZ]

  o.invXXS✻XU             = [Matrix{T}(undef, o.DGP.kX, o.N✻) for \_ ∈ 0:o.Repl.kZ]

  o.S✻UU                  = [Vector{T}(undef, o.N✻) for \_ ∈ 0:o.Repl.kZ, \_ ∈ 0:o.Repl.kZ]

  o.S✻⋂XU₂      = Array{T,3}(undef, o.Repl.kX, o.N✻⋂, o.kY₂)

  o.S✻⋂XU₂RparY = Array{T,3}(undef, o.Repl.kX, o.N✻⋂, o.Repl.kZ)

  o.invXXS✻XU₂  = Array{T,3}(undef, o.Repl.kX, o.N✻ , o.kY₂)

  o.invXXS✻XU₂RparY = Array{T,3}(undef, o.Repl.kX, o.N✻, o.Repl.kZ)

  o.S✻XU₂ = Array{T,3}(undef, o.Repl.kX, o.N✻, o.kY₂)

  o.S✻XU₂RparY = Array{T,3}(undef, o.Repl.kX, o.N✻, o.Repl.kZ)

  o.S✻ZperpU₂ = Array{T,3}(undef, o.Repl.kZperp, o.N✻, o.kY₂)

  o.S✻ZperpU₂RparY = Array{T,3}(undef, o.Repl.kZperp, o.N✻, o.Repl.kZ)

  o.invZperpZperpS✻ZperpU₂ = Array{T,3}(undef, o.Repl.kZperp, o.N✻, o.kY₂)

  o.invZperpZperpS✻ZperpU₂RparY = Array{T,3}(undef, o.Repl.kZperp, o.N✻, o.Repl.kZ)

  o.T1L = isone(o.Nw) ? [Matrix{T}(undef, o.Repl.kX, ncols(o.v))] :

                        [Matrix{T}(undef, o.Repl.kX, length(o.WeightGrp[1])), Matrix{T}(undef, o.Repl.kX, length(o.WeightGrp[end]))]

  o.T1R = deepcopy(o.T1L)

  o.T2 = Matrix{T}(undef, o.N✻, o.N✻)

  o.robust && o.bootstrapt && iszero(o.granular) &&

    (o.negS✻UMZperpX = [Array{T,3}(undef, o.Repl.kX, o.N⋂, o.N✻) for \_ in 0:o.Repl.kZ])

  if o.bootstrapt

    if o.robust && o.granular

      o.S✻UMZperp = [Matrix{T}(undef, o.Nobs, o.N✻) for \_ ∈ 0:o.Repl.kZ]

      o.S✻UPX     = [Matrix{T}(undef, o.Nobs, o.N✻) for \_ ∈ 0:o.Repl.kZ]

    end

    if o.liml || !o.robust

      o.YY✻\_b   = zeros(o.Repl.kZ+1, o.Repl.kZ+1)

      o.YPXY✻\_b = zeros(o.Repl.kZ+1, o.Repl.kZ+1)

    end

    o.NFE>0 && (o.bootstrapt || !isone(o.κ) || o.liml) && (o.CTFEU = Vector{Matrix{T}}(undef, o.Repl.kZ+1))

  end

  o.S✻⋂XY₂      = o.Repl.S✻⋂XY₂     - o.Repl.S✻⋂XZperp     \* o.Repl.invZperpZperpZperpY₂  - o.Repl.invZperpZperpZperpX' \* (o.Repl.S✻⋂ZperpY₂  - o.Repl.S✻⋂ZperpZperp \* o.Repl.invZperpZperpZperpY₂ )

  o.S✻⋂XX       = o.Repl.S✻⋂XX      - o.Repl.S✻⋂XZperp     \* o.Repl.invZperpZperpZperpX   - o.Repl.invZperpZperpZperpX' \* (o.Repl.S✻⋂XZperp'  - o.Repl.S✻⋂ZperpZperp \* o.Repl.invZperpZperpZperpX  )

  o.S✻⋂XDGPZ    = o.DGP.S✻⋂XZpar    - o.Repl.S✻⋂XZperp     \* o.DGP.invZperpZperpZperpZpar - o.Repl.invZperpZperpZperpX' \* (o.DGP.S✻⋂ZperpZpar - o.Repl.S✻⋂ZperpZperp \* o.DGP.invZperpZperpZperpZpar)

  o.S✻⋂Xy₁      = o.Repl.S✻⋂Xy₁     - o.Repl.S✻⋂XZperp     \* o.Repl.invZperpZperpZperpy₁  - o.Repl.invZperpZperpZperpX' \* (o.Repl.S✻⋂Zperpy₁  - o.Repl.S✻⋂ZperpZperp \* o.Repl.invZperpZperpZperpy₁ )

    S✻⋂ZperpX   = o.Repl.S✻⋂XZperp' - o.Repl.S✻⋂ZperpZperp \* o.Repl.invZperpZperpZperpX

  o.DGP.restricted &&

    (o.S✻⋂X\_DGPZR₁ = o.DGP.S✻⋂XZR₁     - o.Repl.S✻⋂XZperp     \* o.DGP.invZperpZperpZperpZR₁  - o.Repl.invZperpZperpZperpX' \* (o.DGP.S✻⋂ZperpZR₁  - o.Repl.S✻⋂ZperpZperp \* o.DGP.invZperpZperpZperpZR₁ ))

  o.invXXS✻⋂XY₂     = o.Repl.invXX \* o.S✻⋂XY₂

  o.invXXS✻⋂XX      = o.Repl.invXX \* o.S✻⋂XX

  o.invXXS✻⋂XDGPZ   = o.Repl.invXX \* o.S✻⋂XDGPZ

  o.invXXS✻⋂Xy₁     = o.Repl.invXX \* o.S✻⋂Xy₁

  o.DGP.restricted &&

    (o.invXXS✻⋂XDGPZR₁ = o.Repl.invXX \* o.S✻⋂X\_DGPZR₁)

  \_S✻ZperpY₂      = @panelsum(o, o.Repl.S✻⋂ZperpY₂ , o.info✻\_✻⋂)  # moments of variables \_before\_ FWL processing

  \_S✻Zperpy₁      = dropdims(@panelsum(o, reshape(o.Repl.S✻⋂Zperpy₁,Val(3)), o.info✻\_✻⋂); dims=3)

  \_S✻ZperpDGPZpar = @panelsum(o, o.DGP.S✻⋂ZperpZpar, o.info✻\_✻⋂)

  o.DGP.restricted &&

    (\_S✻ZperpDGPZR₁  = @panelsum(o, o.DGP.S✻⋂ZperpZR₁, o.info✻\_✻⋂))

  S✻ZperpZperp    = @panelsum(o, o.Repl.S✻⋂ZperpZperp, o.info✻\_✻⋂)

  o.S✻XY₂         = @panelsum(o, o.S✻⋂XY₂   , o.info✻\_✻⋂)

  o.S✻XX          = @panelsum(o, o.S✻⋂XX    , o.info✻\_✻⋂)

  o.S✻XDGPZ       = @panelsum(o, o.S✻⋂XDGPZ, o.info✻\_✻⋂)

  o.S✻Xy₁         = dropdims(@panelsum(o, reshape(o.S✻⋂Xy₁,Val(3)), o.info✻\_✻⋂); dims=3)

  o.S✻ZperpX      = @panelsum(o, S✻⋂ZperpX, o.info✻\_✻⋂)

  o.S✻ZperpY₂     = \_S✻ZperpY₂ - S✻ZperpZperp \* o.Repl.invZperpZperpZperpY₂

  o.S✻ZperpDGPZ   = \_S✻ZperpDGPZpar - S✻ZperpZperp \* o.DGP.invZperpZperpZperpZpar

  o.S✻Zperpy₁     = \_S✻Zperpy₁ - S✻ZperpZperp \* o.Repl.invZperpZperpZperpy₁

  if o.DGP.restricted

    o.S✻XZR₁        = @panelsum(o, o.S✻⋂X\_DGPZR₁, o.info✻\_✻⋂)

    o.S✻ZperpDGPZR₁ = @panelsum(o, o.DGP.S✻⋂ZperpZR₁ , o.info✻\_✻⋂) - S✻ZperpZperp \* o.DGP.invZperpZperpZperpZR₁

  end

  if o.NFE>0 && (o.liml || !isone(o.κ) || o.bootstrapt)

      CT✻⋂FEX  = [crosstabFE(o, o.Repl.X₁, o.info✻⋂) crosstabFE(o, o.Repl.X₂, o.info✻⋂)]

    o.CT✻FEX   = @panelsum(o, CT✻⋂FEX, o.info✻\_✻⋂)

    o.CT✻FEY₂  = crosstabFE(o, o.DGP.Y₂, o.info✻)

    o.CT✻FEZ   = crosstabFE(o, o.DGP.Z, o.info✻)

    o.CT✻FEy₁  = crosstabFE(o, o.DGP.y₁, o.info✻)

    o.DGP.restricted &&

      (o.CT✻FEZR₁ = crosstabFE(o, o.DGP.ZR₁, o.info✻))

  end

  if ((o.robust && o.bootstrapt) || o.liml || !o.robust || !isone(o.κ))

    S✻⋂ReplZX = (o.Repl.S✻⋂XZpar - o.Repl.S✻⋂XZperp \* o.Repl.invZperpZperpZperpZpar - o.Repl.invZperpZperpZperpX' \* (o.Repl.S✻⋂ZperpZpar - o.Repl.S✻⋂ZperpZperp \* o.Repl.invZperpZperpZperpZpar))'

  end

  if o.bootstrapt & o.robust

    o.info⋂\_✻⋂ = panelsetup(o.ID✻⋂, o.subcluster+1:o.NClustVar)

    o.S⋂ReplZX = @panelsum(o, S✻⋂ReplZX, o.info⋂\_✻⋂)

    S⋂ZperpX  = @panelsum(o, S✻⋂ZperpX, o.info⋂\_✻⋂)

    o.S⋂Xy₁  = dropdims(@panelsum(o, reshape(o.S✻⋂Xy₁,Val(3)), o.info⋂\_✻⋂); dims=3)

    o.J⋂s = isone(o.Nw) ? [Array{T,3}(undef, o.N⋂, ncols(o.v), o.Repl.kZ)] :

                          [Array{T,3}(undef, o.N⋂, length(o.WeightGrp[1]), o.Repl.kZ), Array{T,3}(undef, o.N⋂, length(o.WeightGrp[end]), o.Repl.kZ)]

    if o.granular

      o.crosstab✻ind = o.Nobs==o.N✻ ? Vector(diagind(FakeArray(o.N✻,o.N✻))) : LinearIndices(FakeArray(o.Nobs,o.N✻))[CartesianIndex.(1:o.Nobs, o.ID✻)]

      o.XinvXX = X₁₂B(o, o.Repl.X₁, o.Repl.X₂, o.Repl.invXX)

      o.PXZ    = X₁₂B(o, o.Repl.X₁, o.Repl.X₂, o.Repl.invXXXZ)

    else

      inds = o.subcluster>0 ?

              [CartesianIndex(j,i) for (j,v) ∈ enumerate(o.info⋂\_✻⋂) for i ∈ v] :  # crosstab ∩,\* is wide

              o.NClustVar == o.NBootClustVar ?

                  [CartesianIndex(i,i) for i ∈ 1:o.N✻⋂] :  # crosstab \*,∩ is square

                  [CartesianIndex(i,j) for (j,v) ∈ enumerate(o.clust[o.BootClust].info) for i ∈ v]  # crosstab ∩,\* is tall

      inds = [CartesianIndex(k,I) for I ∈ inds for k ∈ 1:o.Repl.kX]

      o.crosstab⋂✻ind = LinearIndices(FakeArray(Tuple(max(inds...))...))[inds]

      o.S⋂XZperpinvZperpZperp = S⋂ZperpX' \* o.Repl.invZperpZperp

      o.Q    = Array{T,3}(undef, o.N✻, o.N⋂, o.N✻)

      o.NFE>0 && (o.CT⋂FEX = o.invFEwt .\* @panelsum(o, CT✻⋂FEX, o.info⋂\_✻⋂))

      o.β̈v = isone(o.Nw) ? [Matrix{T}(undef, o.N✻, ncols(o.v))] :

                           [Matrix{T}(undef, o.N✻, length(o.WeightGrp[1])), Matrix{T}(undef, o.N✻, length(o.WeightGrp[end]))]

    end

  end

  if o.liml || !o.robust || !isone(o.κ)  # cluster-wise moments after FWL

    o.S✻Y₂Y₂     = o.Repl.S✻Y₂Y₂    - \_S✻ZperpY₂'      \* o.DGP.invZperpZperpZperpY₂   - o.DGP.invZperpZperpZperpY₂'   \* o.S✻ZperpY₂

    o.S✻DGPZDGPZ = o.DGP.S✻ZparZpar - \_S✻ZperpDGPZpar' \* o.DGP.invZperpZperpZperpZpar - o.DGP.invZperpZperpZperpZpar' \* o.S✻ZperpDGPZ

    o.S✻DGPZY₂   = o.DGP.S✻ZparY₂   - \_S✻ZperpDGPZpar' \* o.DGP.invZperpZperpZperpY₂   - o.DGP.invZperpZperpZperpZpar' \* o.S✻ZperpY₂

    o.S✻DGPZy₁   = o.DGP.S✻Zpary₁   - \_S✻ZperpDGPZpar' \* o.DGP.invZperpZperpZperpy₁   - o.DGP.invZperpZperpZperpZpar' \* o.S✻Zperpy₁

    o.S✻Y₂y₁     = o.Repl.S✻Y₂y₁    - \_S✻ZperpY₂'      \* o.DGP.invZperpZperpZperpy₁   - o.DGP.invZperpZperpZperpY₂'   \* o.S✻Zperpy₁

    o.S✻y₁y₁     = o.Repl.S✻y₁y₁    - \_S✻Zperpy₁'      \* o.DGP.invZperpZperpZperpy₁   - o.S✻Zperpy₁' \* o.DGP.invZperpZperpZperpy₁

    o.DGP.restricted &&

      (o.S✻DGPZR₁y₁ = o.DGP.S✻ZR₁y₁ - \_S✻ZperpDGPZR₁' \* o.DGP.invZperpZperpZperpy₁ - o.DGP.invZperpZperpZperpZR₁' \* o.S✻Zperpy₁)

    if o.Repl.restricted

      \_S✻ZperpReplZR₁ = @panelsum(o, o.Repl.S✻⋂ZperpZR₁, o.info✻\_✻⋂)

      \_S✻⋂XReplZR₁    = @panelsum(o, o.Repl.S✻⋂XZR₁    , o.info✻\_✻⋂)

      o.r₁S✻ReplZR₁Y₂     = o.r₁' \* (o.Repl.S✻ZR₁Y₂ - \_S✻ZperpReplZR₁' \* o.Repl.invZperpZperpZperpY₂ - o.Repl.invZperpZperpZperpZR₁' \* o.S✻ZperpY₂)

      o.r₁S✻ReplZR₁X      = o.r₁' \* (\_S✻⋂XReplZR₁'  - \_S✻ZperpReplZR₁' \* o.Repl.invZperpZperpZperpX  - o.Repl.invZperpZperpZperpZR₁' \* o.S✻ZperpX )

      o.r₁S✻ReplZR₁DGPZ   = o.r₁' \* panelcross(o.Repl.ZR₁, o.DGP.Z, o.info✻)

      o.r₁S✻ReplZR₁y₁     = o.r₁' \* (o.Repl.S✻ZR₁y₁ - \_S✻ZperpReplZR₁' \* o.Repl.invZperpZperpZperpy₁ - o.Repl.invZperpZperpZperpZR₁' \* o.S✻Zperpy₁)

      o.DGP.restricted &&

        (o.r₁S✻ReplZR₁DGPZR₁ = o.r₁' \* panelcross(o.Repl.ZR₁, o.DGP.ZR₁, o.info✻))

    end

    \_S✻ZperpReplZpar = @panelsum(o, o.Repl.S✻⋂ZperpZpar, o.info✻\_✻⋂)

    \_S✻ReplXZ        = @panelsum(o, o.Repl.S✻⋂XZpar    , o.info✻\_✻⋂)

    o.S✻ReplZY₂      = o.Repl.S✻ZparY₂ - \_S✻ZperpReplZpar' \* o.Repl.invZperpZperpZperpY₂ - o.Repl.invZperpZperpZperpZpar' \* o.S✻ZperpY₂

    o.S✻ReplZX       = \_S✻ReplXZ'      - \_S✻ZperpReplZpar' \* o.Repl.invZperpZperpZperpX  - o.Repl.invZperpZperpZperpZpar' \* o.S✻ZperpX

    o.S✻ReplZDGPZ    = panelcross(o.Repl.Z, o.DGP.Z, o.info✻)

    o.S✻ReplZy₁      = o.Repl.S✻Zpary₁ - \_S✻ZperpReplZpar' \* o.Repl.invZperpZperpZperpy₁ - o.Repl.invZperpZperpZperpZpar' \* o.S✻Zperpy₁

    if o.DGP.restricted

      \_S✻⋂XDGPZR₁ = @panelsum(o, o.DGP.S✻⋂XZR₁, o.info✻\_✻⋂)

      o.S✻ReplZDGPZR₁  = panelcross(o.Repl.Z, o.DGP.ZR₁, o.info✻)

      o.S✻DGPZR₁Y₂     = o.DGP.S✻ZR₁Y₂  - \_S✻ZperpDGPZR₁' \* o.Repl.invZperpZperpZperpY₂  - o.DGP.invZperpZperpZperpZR₁' \* o.S✻ZperpY₂

      o.S✻DGPZR₁DGPZR₁ = o.DGP.S✻ZR₁ZR₁ - \_S✻ZperpDGPZR₁' \* o.DGP.invZperpZperpZperpZR₁  - o.DGP.invZperpZperpZperpZR₁' \* o.S✻ZperpDGPZR₁

      o.S✻DGPZR₁DGPZ   = o.DGP.S✻ZR₁Z   - \_S✻ZperpDGPZR₁' \* o.DGP.invZperpZperpZperpZpar - o.DGP.invZperpZperpZperpZR₁' \* o.S✻ZperpDGPZ

      o.S✻DGPZR₁X      = \_S✻⋂XDGPZR₁'   - \_S✻ZperpDGPZR₁' \* o.Repl.invZperpZperpZperpX   - o.DGP.invZperpZperpZperpZR₁' \* o.S✻ZperpX

    end

  end

  o.invXXS✻XY₂   = @panelsum(o, o.invXXS✻⋂XY₂  , o.info✻\_✻⋂)

  o.invXXS✻XX    = @panelsum(o, o.invXXS✻⋂XX   , o.info✻\_✻⋂)

  o.invXXS✻XDGPZ = @panelsum(o, o.invXXS✻⋂XDGPZ, o.info✻\_✻⋂)

  o.invXXS✻Xy₁   = dropdims(@panelsum(o, reshape(o.invXXS✻⋂Xy₁,Val(3)), o.info✻\_✻⋂); dims=3)

  o.invZperpZperpS✻ZperpY₂   = o.Repl.invZperpZperp \* o.S✻ZperpY₂

  o.invZperpZperpS✻ZperpX    = o.Repl.invZperpZperp \* o.S✻ZperpX

  o.invZperpZperpS✻Zperpy₁   = o.Repl.invZperpZperp \* o.S✻Zperpy₁

  o.invZperpZperpS✻ZperpDGPZ = o.Repl.invZperpZperp \* o.S✻ZperpDGPZ

  if o.DGP.restricted

    o.invXXS✻XDGPZR₁ = @panelsum(o, o.invXXS✻⋂XDGPZR₁, o.info✻\_✻⋂)

    o.invZperpZperpS✻ZperpDGPZR₁ = o.Repl.invZperpZperp \* o.S✻ZperpDGPZR₁

  end

end

function PrepWRE!(o::StrBootTest{T}) where T

  r₁ = o.null ? [o.r₁ ; o.r] : o.r₁

  EstimateIV!(o.DGP, o, r₁)

  MakeResidualsIV!(o.DGP, o)

  o.robust && o.bootstrapt && o.granular && (Ü₂par = view(o.DGP.Ü₂ \* o.Repl.RparY,:,:))

  o.S✻⋂XU₂ .= o.S✻⋂XY₂ - o.S✻⋂XX \* o.DGP.Π̂

  o.S✻⋂XU₂RparY .= o.S✻⋂XU₂ \* o.Repl.RparY

  o.S✻XU₂ .= o.S✻XY₂ - o.S✻XX \* o.DGP.Π̂

  o.S✻XU₂RparY .= o.S✻XU₂ \* o.Repl.RparY

  o.S✻ZperpU₂ .= o.S✻ZperpY₂ - o.S✻ZperpX \* o.DGP.Π̂

  o.S✻ZperpU₂RparY .= o.S✻ZperpU₂ \* o.Repl.RparY

  o.invZperpZperpS✻ZperpU₂ .= o.invZperpZperpS✻ZperpY₂ - o.invZperpZperpS✻ZperpX \* o.DGP.Π̂

  o.invZperpZperpS✻ZperpU₂RparY .= o.invZperpZperpS✻ZperpU₂ \* o.Repl.RparY

  o.invXXS✻XU₂ .= o.Repl.invXX \* o.S✻XU₂

  o.invXXS✻XU₂RparY .= o.invXXS✻XU₂ \* o.Repl.RparY

  if o.liml || !o.robust || !isone(o.κ)

    S✻U₂y₁ = o.S✻Y₂y₁ - o.DGP.Π̂' \* o.S✻Xy₁

    S✻U₂RparYy₁ = o.Repl.RparY' \* S✻U₂y₁

    S✻ZU₂ = o.S✻ReplZY₂ - o.S✻ReplZX \* o.DGP.Π̂

    S✻ZU₂RparY = S✻ZU₂ \* o.Repl.RparY

    Π̂S✻XÜ₂γ̈ = o.DGP.Π̂' \* o.S✻XU₂ \* o.DGP.γ̈

    S✻Ü₂Y₂ = o.S✻Y₂Y₂ - o.DGP.Π̂' \* o.S✻XY₂

    S✻Y₂Ü₂γ̈ = S✻Ü₂Y₂' \* o.DGP.γ̈

    S✻Ü₂parÜ₂par = o.Repl.RparY' \* (S✻Ü₂Y₂ - o.S✻XU₂' \* o.DGP.Π̂) \* o.Repl.RparY

    S✻UUterm = o.S✻Y₂y₁ - o.S✻DGPZY₂' \* o.DGP.β̈ - o.DGP.Π̂' \* (o.S✻Xy₁  - o.S✻XDGPZ \* o.DGP.β̈)

    if o.Repl.restricted

      S✻ReplZR₁r₁U₂ = o.r₁S✻ReplZR₁Y₂ - o.r₁S✻ReplZR₁X \* o.DGP.Π̂

      S✻ReplZR₁r₁U₂RparY = S✻ReplZR₁r₁U₂ \* o.Repl.RparY

    end

    if o.DGP.restricted

      r₁S✻DGPZR₁y₁ = r₁' \* o.S✻DGPZR₁y₁

      S✻ReplZDGPZR₁r₁ = o.S✻ReplZDGPZR₁ \* r₁

      o.Repl.restricted && (S✻ReplZR₁r₁DGPZR₁r₁ = o.r₁S✻ReplZR₁DGPZR₁ \* r₁)

    end

  end

  if (o.liml || o.bootstrapt || !isone(o.κ)) && o.NFE>0

    CT✻FEU = o.CT✻FEY₂ - o.CT✻FEX \* o.DGP.Π̂

    CT✻FEURparY = CT✻FEU \* o.Repl.RparY

  end

  @inbounds for i ∈ 0:o.Repl.kZ  # precompute various clusterwise sums

    # S\_✻(u .\* X), S\_✻(u .\* Zperp) for residuals u for each endog var; store transposed

    if iszero(i)

      o.S✻XU[1]      .= o.S✻Xy₁      - o.S✻XDGPZ      \* o.DGP.β̈ + o.S✻XU₂      \* o.DGP.γ̈

      o.invXXS✻XU[1] .= o.invXXS✻Xy₁ - o.invXXS✻XDGPZ \* o.DGP.β̈ + o.invXXS✻XU₂ \* o.DGP.γ̈

      if o.DGP.restricted

        o.S✻XU[1]      .-= o.S✻XZR₁         \* r₁

        o.invXXS✻XU[1] .-= o.invXXS✻XDGPZR₁ \* r₁

      end

    else

      o.S✻XU[i+1]      .= view(o.S✻XU₂RparY,:,:,i)

      o.invXXS✻XU[i+1] .= view(o.invXXS✻XU₂RparY,:,:,i)

    end

    if o.liml || !isone(o.κ) || o.bootstrapt

      if iszero(i)

        o.S✻ZperpU[1]              .= o.S✻Zperpy₁              - o.S✻ZperpDGPZ              \* o.DGP.β̈ + o.S✻ZperpU₂              \* o.DGP.γ̈

        o.invZperpZperpS✻ZperpU[1] .= o.invZperpZperpS✻Zperpy₁ - o.invZperpZperpS✻ZperpDGPZ \* o.DGP.β̈ + o.invZperpZperpS✻ZperpU₂ \* o.DGP.γ̈

        if o.DGP.restricted

          o.S✻ZperpU[1]              .-= o.S✻ZperpDGPZR₁              \* r₁

          o.invZperpZperpS✻ZperpU[1] .-= o.invZperpZperpS✻ZperpDGPZR₁ \* r₁

        end

      else

        o.S✻ZperpU[i+1]              .= view(o.S✻ZperpU₂RparY,:,:,i)

        o.invZperpZperpS✻ZperpU[i+1] .= view(o.invZperpZperpS✻ZperpU₂RparY,:,:,i)

      end

      if o.NFE>0

        if iszero(i)

          o.CTFEU[1] = o.CT✻FEy₁ - o.CT✻FEZ \* o.DGP.β̈ + CT✻FEU \* o.DGP.γ̈

          o.DGP.restricted &&

            (o.CTFEU[1] .-= o.CT✻FEZR₁ \* r₁)

        else

          o.CTFEU[i+1] = @view CT✻FEURparY[:,:,i]

        end

      end

    end

    if o.liml || !isone(o.κ) || !o.robust

      if iszero(i)  # panelsum2(o, o.Repl.y₁par, o.Repl.Z, uwt, o.info✻)

        o.S✻YU[1,1] .= o.S✻y₁y₁ - o.S✻DGPZy₁'o.DGP.β̈ + S✻U₂y₁'o.DGP.γ̈

        o.DGP.restricted &&

          (o.S✻YU[1,1] .-= view(r₁S✻DGPZR₁y₁, 1,:))

        if o.Repl.restricted

          o.S✻YU[1,1] .-= view(o.r₁S✻ReplZR₁y₁ - o.r₁S✻ReplZR₁DGPZ \* o.DGP.β̈ + S✻ReplZR₁r₁U₂ \* o.DGP.γ̈, 1,:,1)

          o.DGP.restricted &&

            (o.S✻YU[1,1] .+= view(S✻ReplZR₁r₁DGPZR₁r₁, 1,:,1))

        end

        for j ∈ 1:o.Repl.kZ

          o.S✻YU[j+1,1] .= view(o.S✻ReplZy₁ - o.S✻ReplZDGPZ \* o.DGP.β̈ + S✻ZU₂ \* o.DGP.γ̈, j,:,1)

          o.DGP.restricted &&

            (o.S✻YU[j+1,1] .-= view(S✻ReplZDGPZR₁r₁, j,:,1))

        end

      else

        o.S✻YU[1,i+1] .= @view S✻U₂RparYy₁[i,:]

        o.Repl.restricted &&

          (o.S✻YU[1,i+1] .-= @view S✻ReplZR₁r₁U₂RparY[1,:,i])

        for j ∈ 1:o.Repl.kZ

          o.S✻YU[j+1,i+1] .= @view S✻ZU₂RparY[j,:,i]

        end

      end

      if iszero(i)

        o.S✻UU[1,i+1] .= o.S✻y₁y₁ - (2 \* o.S✻DGPZy₁ - o.S✻DGPZDGPZ \* o.DGP.β̈)'o.DGP.β̈ + (2 \* S✻UUterm - Π̂S✻XÜ₂γ̈ + S✻Y₂Ü₂γ̈)'o.DGP.γ̈

        o.DGP.restricted

          (o.S✻UU[1,i+1] .+= dropdims(-2 \* r₁S✻DGPZR₁y₁ + r₁' \* (o.S✻DGPZR₁DGPZR₁ \* r₁) + 2 \* r₁' \* (o.S✻DGPZR₁DGPZ \* o.DGP.β̈ + (o.S✻DGPZR₁X \* o.DGP.Π̂ - o.S✻DGPZR₁Y₂) \* o.DGP.γ̈); dims=1))  # XXX dropdims vs view...

      else

        o.S✻UU[1,i+1] .= view(o.Repl.RparY' \* (S✻UUterm + S✻Y₂Ü₂γ̈ - Π̂S✻XÜ₂γ̈), i,:)

        if o.DGP.restricted

          o.S✻UU[1,i+1] .-= view(r₁' \* (o.S✻DGPZR₁Y₂ - o.S✻DGPZR₁X \* o.DGP.Π̂) \* o.Repl.RparY, 1,:,i)

        end

      end

      for j ∈ 1:i

        o.S✻UU[j+1,i+1] .= view(S✻Ü₂parÜ₂par, j,:,i)

      end

    end

    if o.robust && o.bootstrapt && o.granular

      i>0 && (o.S✻UPX[i+1] .= o.XinvXX \* o.S✻XU[i+1])

      o.S✻UMZperp[i+1] .= o.Repl.Zperp \* o.invZperpZperpS✻ZperpU[i+1]

      if iszero(i)  # subtract crosstab of observation by ∩-group of u

        o.S✻UMZperp[   1][o.crosstab✻ind] .-= o.DGP.u⃛₁

      else

        o.S✻UMZperp[i+1][o.crosstab✻ind] .-= view(Ü₂par,:,i)

      end

      o.NFE>0 &&

        (o.S✻UMZperp[i+1] .-= view(o.invFEwt .\* o.CTFEU[i+1], o.\_FEID, :))  # CT\_(\*,FE) (U ̈\_(parj) ) (S\_FE S\_FE^' )^(-1) S\_FE

    end

  end

  if o.robust && o.bootstrapt && iszero(o.granular)

    @inbounds for j ∈ 0:o.Repl.kZ

      if o.Repl.Yendog[j+1]

        o.negS✻UMZperpX[j+1] = o.S⋂XZperpinvZperpZperp \* o.S✻ZperpU[j+1]  # S\_\* diag⁡(U ̈\_(∥j) ) Z\_⊥ (Z\_⊥^' Z\_⊥ )^(-1) Z\_(⊥g)^' X\_(∥g)

        if iszero(j)  # - S\_\*  diag⁡(U ̈\_(∥j) ) I\_g^' X\_(∥g)

          o.negS✻UMZperpX[j+1][o.crosstab⋂✻ind] .-= vec(o.S✻⋂Xy₁ - o.S✻⋂XDGPZ \* o.DGP.β̈ + o.S✻⋂XU₂ \* o.DGP.γ̈)

          o.DGP.restricted &&

            (o.negS✻UMZperpX[j+1][o.crosstab⋂✻ind] .+= vec(o.S✻⋂X\_DGPZR₁ \* r₁))

        else

          o.negS✻UMZperpX[j+1][o.crosstab⋂✻ind] .-= vec(view(o.S✻⋂XU₂RparY,:,:,j))

        end

        o.NFE>0 &&

          (o.negS✻UMZperpX[j+1] .+= o.CT⋂FEX'o.CTFEU[j+1])  # CT\_(\*,FE) (U ̈\_(∥j) ) (S\_FE S\_FE^' )^(-1) S\_FE

      end

    end

  end

  nothing

end

# For WRE, and with reference to Y = [y₁ Z], given 0-based columns indexes within it, i, j, return all bootstrap realizations of

# Y[:,i]'((1-κ)\*M\_Zperp-κ\*M\_Xpar)\*Y[:,j] for κ constant across replications

# i can be a rowvector

# (only really the Hessian when we narrow Y to Z)

function HessianFixedkappa(o::StrBootTest{T}, is::Vector{S} where S<:Integer, j::Integer, κ::Number, w::Integer) where T

  dest = Matrix{T}(undef, length(is), ncols(o.v))

  @inbounds for i ∈ eachindex(is, axes(dest,1))

    \_HessianFixedkappa!(o, view(dest,i:i,:), is[i], j, κ, w)

  end

  dest

end

function \_HessianFixedkappa!(o::StrBootTest, dest::AbstractMatrix, i::Integer, j::Integer, κ::Number, w::Integer)

  if !(o.Repl.Yendog[i+1] || o.Repl.Yendog[j+1])  # if both vars exog, result = order-0 term only, same for all draws

    !iszero(κ) &&

      (dest .= dot(view(o.Repl.XZ,:,i), view(o.Repl.invXXXZ,:,j)))

    if !isone(κ)

      if iszero(κ)

        fill!(dest, o.Repl.YY[i+1,j+1])

      else

        dest .= κ .\* dest .+ (1 - κ) .\* o.Repl.YY[i+1,j+1]

      end

    end

  else

    if !iszero(κ)  # repetitiveness in this section to maintain type stability

      if o.Repl.Yendog[i+1]

        T1L = o.T1L[isone(o.Nw) || w<o.Nw ? 1 : 2]  # preallocated destinations

        mul!(T1L, o.S✻XU[i+1], o.v)

        if iszero(i)

          T1L .+= o.Repl.Xy₁par

        else

          T1L .+= view(o.Repl.XZ,:,i)

        end

        if o.Repl.Yendog[j+1]

          T1R = o.T1R[isone(o.Nw) || w<o.Nw ? 1 : 2]  # preallocated destinations

          mul!(T1R, o.invXXS✻XU[j+1], o.v)

          if iszero(j)

            T1R .+=  o.Repl.invXXXy₁par

          else

            T1R .+= view(o.Repl.invXXXZ,:,j)

          end

          coldot!(o, dest, T1L, T1R)

        else

          dest .= view(o.Repl.invXXXZ,:,j)'T1L  # coldot!(o, dest, T1L, view(o.Repl.invXXXZ,:,j))

        end

      else

        if o.Repl.Yendog[j+1]

          T1R = o.T1R[isone(o.Nw) || w<o.Nw ? 1 : 2]  # use preallocated destinations

          mul!(T1R, o.invXXS✻XU[j+1], o.v)

          if iszero(j)

            T1R .+=  o.Repl.invXXXy₁par

          else

            T1R .+= view(o.Repl.invXXXZ,:,j)

          end

          dest .= view(o.Repl.XZ,:,i)'T1R

        else

          dest .= dot(view(o.Repl.XZ,:,i), view(o.Repl.invXXXZ,:,j))

        end

      end

    end

    if !isone(κ)

      if o.Repl.Yendog[i+1]

        o.T2 .= o.invZperpZperpS✻ZperpU[i+1]'o.S✻ZperpU[j+1]  # quadratic term

        o.T2[diagind(o.T2)] .-= i ≤ j ? o.S✻UU[i+1, j+1] : o.S✻UU[j+1, i+1]  # minus diagonal crosstab

        o.NFE>0 &&

          (o.T2 .+= o.CTFEU[i+1]' \* (o.invFEwt .\* o.CTFEU[j+1]))

        if iszero(κ)

          dest .= o.Repl.YY[i+1,j+1] .+ colquadformminus!(o, (                            o.S✻YU[i+1,j+1] .+ o.S✻YU[j+1,i+1])'o.v, o.T2, o.v)

        else

          dest .= κ .\* dest .+ (1 - κ)     .\* colquadformminus!(o, (o.Repl.YY[i+1,j+1] .+ o.S✻YU[i+1,j+1] .+ o.S✻YU[j+1,i+1])'o.v, o.T2, o.v)

        end

      elseif iszero(κ)

        dest .= o.Repl.YY[i+1,j+1]

      else

        dest .= κ .\* dest .+ (1 - κ) .\* o.Repl.YY[i+1,j+1]

      end

    end

  end

  nothing

end

# put threaded loops in functions to prevent compiler-perceived type instability https://discourse.julialang.org/t/type-inference-with-threads/2004/3

function FillingLoop1!(o::StrBootTest{T}, dest::Matrix{T}, i::Integer, j::Integer, \_β̈::AbstractMatrix{T}, β̈v::AbstractMatrix{T}) where T

  Threads.@threads for g ∈ 1:o.N⋂

    PXY✻ = [o.PXZ[g,i]]

    o.Repl.Yendog[i+1] && (PXY✻ = PXY✻ .+ view(o.S✻UPX[i+1],g,:)'o.v)

    if iszero(j)

      dest[g,:]   = dropdims(colsum(PXY✻ .\* (o.Repl.y₁[g] .- view(o.S✻UMZperp[1],g,:)'o.v)); dims=1)

    elseif o.Repl.Yendog[j+1]

      dest[g,:] .-= dropdims(colsum(PXY✻ .\* (o.Repl.Z[g,j] \* \_β̈ .- view(o.S✻UMZperp[j+1],g,:)'β̈v)); dims=1)

    else

      dest[g,:] .-= dropdims(colsum(PXY✻ .\* (o.Repl.Z[g,j] \* \_β̈)); dims=1)

    end

  end

  nothing

end

function FillingLoop2!(o::StrBootTest{T}, dest::Matrix{T}, i::Integer, j::Integer, \_β̈::AbstractMatrix{T}, β̈v::AbstractMatrix{T}) where T

  Threads.@threads for g ∈ 1:o.N⋂

    S = o.info⋂[g]

    PXY✻ = o.Repl.Yendog[i+1] ? view(o.PXZ,S,i) .+ view(o.S✻UPX[i+1],S,:) \* o.v :

                         reshape(view(o.PXZ,S,i), :, 1)

    if iszero(j)

      dest[i,:]   = dropdims(colsum(PXY✻ .\* (o.Repl.y₁[S] .- view(o.S✻UMZperp[1],S,:) \* o.v)); dims=1)

    else

      dest[i,:] .-= dropdims(colsum(PXY✻ .\* (o.Repl.Yendog[j+1] ? o.Repl.Z[S,j] \* \_β̈ .- view(o.S✻UMZperp[j+1],S,:) \* β̈v :

                                                                   o.Repl.Z[S,j] \* \_β̈                                       )); dims=1)

    end

  end

  nothing

end

# Workhorse for WRE CRVE sandwich filling

# Given a zero-based column index, i>0, and a matrix β̈s of all the bootstrap estimates,

# return all bootstrap realizations of P\_X \* Z[:,i]\_g ' û₁g^\*b

# for all groups in the intersection of all error clusterings

# return value has one row per ⋂ cluster, one col per bootstrap replication

# that is, given i, β̈s = δ ̂\_CRκ^(\*), return, over all g, b (P\_(X\_∥ g) Z\_(∥i)^(\*b) )^' (M\_(Z\_⊥ ) y\_(1∥)^(\*b) )\_g-(P\_(X\_∥ g) Z\_(∥i)^(\*b) )^' (M\_(Z\_⊥ ) Z\_∥^(\*b) )\_g δ ̂\_CRκ^(\*b)

function Filling!(o::StrBootTest{T}, dest::AbstractMatrix{T}, i::Int64, β̈s::AbstractMatrix) where T

  if o.granular

    β̈v = \_β̈ = Matrix{T}(undef,0,0)

    if o.Nw == 1  # create or avoid NxB matrix?

      PXY✻ = reshape(view(o.PXZ,:,i), :, 1)

      o.Repl.Yendog[i+1] && (PXY✻ = PXY✻ .+ o.S✻UPX[i+1] \* o.v)

      dest .= @panelsum(o, PXY✻ .\* (o.Repl.y₁ .- o.S✻UMZperp[1] \* o.v), o.info⋂)

      @inbounds for j ∈ 1:o.Repl.kZ

        \_β̈ = view(β̈s,j,:)'

        dest .-= @panelsum(o, PXY✻ .\* (o.Repl.Yendog[j+1] ? view(o.Repl.Z,:,j) \* \_β̈ .- o.S✻UMZperp[j+1] \* (o.v .\* \_β̈) :

                                                            (view(o.Repl.Z,:,j) \* \_β̈)                                    ), o.info⋂)

      end

    else  # create pieces of each N x B matrix one at a time rather than whole thing at once

      @inbounds for j ∈ 0:o.Repl.kZ

        j>0 && (β̈v = o.v .\* (\_β̈ = view(β̈s,j,:)'))

        (o.purerobust ? FillingLoop1! : FillingLoop2!)(o, dest, i, j, \_β̈, β̈v)

      end

    end

  else  # coarse error clustering

    # (P\_(X\_∥ g) Z\_∥^\* )^' (M\_(Z\_⊥ ) y\_(1∥)^\* )\_g

    F1₀ = view(o.Repl.invXXXZ,:,i)

    F1₁ = o.invXXS✻XU[i+1]

    F2₀ = o.S⋂Xy₁

    F2₁ = o.negS✻UMZperpX[1]

    if o.Repl.Yendog[i+1]  # add terms that are zero only if Zpar[i] is exogenous, i.e. if a null refers only to exogenous variables

      dest .= reshape(F1₀'F2₀,:,1) .- (dropdims(F1₀'F2₁; dims=1) - F2₀'F1₁) \* o.v  # 0th- & 1st-order terms

      o.Q .= F1₁'F2₁

      @inbounds for g ∈ 1:o.N⋂

        o.colquadformminus!(dest, g, o.v, o.Q[:,g,:], o.v)

      end

    else

      dest .= F2₀'F1₀ .- dropdims(F1₀'F2₁; dims=1) \* o.v  # 0th- & 1st-order terms

    end

    # -(P\_(X\_∥ g) Z\_∥^\* )^' (M\_(Z\_⊥ ) Z\_∥^(\*b) )\_g δ ̂\_CRκ^\*

    @inbounds for j ∈ 1:o.Repl.kZ

      F2₀ = view(o.S⋂ReplZX,j,:,:)'

      β̈v = o.β̈v[isone(o.Nw) || w<o.Nw ? 1 : 2]; β̈v .= o.v .\* (\_β̈ = -view(β̈s,j,:)')

      if o.Repl.Yendog[j+1]

        F2₁ = o.negS✻UMZperpX[j+1]

        dest .+= F2₀'F1₀ .\* \_β̈ .- (dropdims(F1₀'F2₁; dims=1) - F2₀'F1₁) \* β̈v  # "-" because S✻UMZperpX is stored negated as F2₁=negS✻UMZperpX[j+1]

        o.Q .= F1₁'F2₁

        for g ∈ 1:o.N⋂

          o.colquadformminus!(dest, g, o.v, o.Q[:,g,:], β̈v)

        end

      elseif o.Repl.Yendog[i+1]

        dest .+= reshape(F1₀'F2₀,:,1) .\* \_β̈  .+ F2₀'F1₁ \* β̈v

      else

        dest .+= reshape(F1₀'F2₀,:,1) .\* \_β̈

      end

    end

  end

  nothing

end

function MakeWREStats!(o::StrBootTest{T}, w::Integer) where T

  if isone(o.Repl.kZ)  # optimized code for 1 coefficient in bootstrap regression

    if o.liml

      YY₁₁   = HessianFixedkappa(o, [0], 0, zero(T), w)  # κ=0 => Y\*MZperp\*Y

      YY₁₂   = HessianFixedkappa(o, [0], 1, zero(T), w)

      YY₂₂   = HessianFixedkappa(o, [1], 1, zero(T), w)

      YPXY₁₁ = HessianFixedkappa(o, [0], 0, one(T) , w)  # κ=1 => Y\*PXpar\*Y

      YPXY₁₂ = HessianFixedkappa(o, [0], 1, one(T) , w)

      YPXY₂₂ = HessianFixedkappa(o, [1], 1, one(T) , w)

      YY₁₂YPXY₁₂ = YY₁₂ .\* YPXY₁₂

      x₁₁ = YY₂₂ .\* YPXY₁₁ .- YY₁₂YPXY₁₂      # elements of YY✻^-1 \* YPXY✻ up to factor of det(YY✻)

      x₁₂ = YY₂₂ .\* YPXY₁₂ .- YY₁₂ .\* YPXY₂₂

      x₂₁ = YY₁₁ .\* YPXY₁₂ .- YY₁₂ .\* YPXY₁₁

      x₂₂ = YY₁₁ .\* YPXY₂₂ .- YY₁₂YPXY₁₂

      κs = (x₁₁ .+ x₂₂)./2; κs .= 1 ./ (1 .- (κs .- sqrtNaN.(κs.^2 .- x₁₁ .\* x₂₂ .+ x₁₂ .\* x₂₁)) ./ (YY₁₁ .\* YY₂₂ .- YY₁₂ .\* YY₁₂))  # solve quadratic equation for smaller eignenvalue; last term is det(YY✻)

      !iszero(o.fuller) && (κs .-= o.fuller / (o.\_Nobs - o.kX))

      o.As = κs .\* (YPXY₂₂ .- YY₂₂) .+ YY₂₂

      o.β̈s = (κs .\* (YPXY₁₂ .- YY₁₂) .+ YY₁₂) ./ o.As

    else

      o.As = HessianFixedkappa(o, [1], 1, o.κ, w)

      o.β̈s = HessianFixedkappa(o, [1], 0, o.κ, w) ./ o.As

    end

    if o.null

      o.numerw = o.β̈s .+ (o.Repl.Rt₁ - o.r) / o.Repl.RRpar

    else

      o.numerw = o.β̈s .- o.DGP.β̈

      isone(w) && (o.numerw[1] = o.β̈s[1] + (o.Repl.Rt₁[1] - o.r[1]) / o.Repl.RRpar[1])

    end

    @storeWtGrpResults!(o.numer, o.numerw)

    if o.bootstrapt

      if o.robust

        J⋂s1 = dropdims(o.J⋂s[isone(o.Nw) || w<o.Nw ? 1 : 2]; dims=3)

        Filling!(o, J⋂s1, 1, o.β̈s);

        J⋂s1 ./= o.As

        @inbounds for c ∈ 1:o.NErrClustCombs  # sum sandwich over error clusterings

          nrows(o.clust[c].order)>0 &&

            (J⋂s1 .= J⋂s1[o.clust[c].order,:])

          @clustAccum!(denom, c, coldot(o, @panelsum(o, J⋂s1, o.clust[c].info)))

        end

      else

        denom = (HessianFixedkappa(o, [0], 0, zero(T), w) .- 2 .\* o.β̈s .\* HessianFixedkappa(o, [0], 1, zero(T), w) .+ o.β̈s.^2 .\* HessianFixedkappa(o, [1], 1, zero(T), w)) ./ o.\_Nobs ./ o.As  # classical error variance

      end

      @storeWtGrpResults!(o.dist, o.sqrt ? o.numerw ./ sqrtNaN.(denom) : o.numerw .^ 2 ./ denom)

      denom \*= o.Repl.RRpar[1]^2

    end

    w==1 && o.bootstrapt && (o.statDenom = fill(denom[1],1,1))  # original-sample denominator

  else  # WRE bootstrap for more than 1 retained coefficient in bootstrap regression

    β̈s = zeros(T, o.Repl.kZ, ncols(o.v))

    A = Vector{Matrix{T}}(undef, ncols(o.v))

    if o.liml

      YY✻   = [HessianFixedkappa(o, collect(0:i), i, zero(T), w) for i ∈ 0:o.Repl.kZ] # κ=0 => Y\*MZperp\*Y

      YPXY✻ = [HessianFixedkappa(o, collect(0:i), i,  one(T), w) for i ∈ 0:o.Repl.kZ] # κ=1 => Y\*PXpar\*Y

      @inbounds for b ∈ axes(o.v,2)

        for i ∈ 0:o.Repl.kZ

          o.YY✻\_b[1:i+1,i+1]   = YY✻[i+1][:,b]  # fill uppper triangles, which is all that invsym() looks at

          o.YPXY✻\_b[1:i+1,i+1] = YPXY✻[i+1][:,b]

        end

        o.κ = 1/(1 - real(eigvals(invsym(o.YY✻\_b) \* Symmetric(o.YPXY✻\_b))[1]))

        !iszero(o.fuller) && (o.κ -= o.fuller / (o.\_Nobs - o.kX))

        β̈s[:,b] = (A[b] = invsym(o.κ\*o.YPXY✻\_b[2:end,2:end] + (1-o.κ)\*o.YY✻\_b[2:end,2:end])) \* (o.κ\*o.YPXY✻\_b[1,2:end] + (1-o.κ)\*o.YY✻\_b[1,2:end])

      end

    else

      δnumer =  HessianFixedkappa(o, collect(1:o.Repl.kZ), 0, o.κ, w)

      δdenom = [HessianFixedkappa(o, collect(1:i), i, o.κ, w) for i ∈ 1:o.Repl.kZ]

      nt = Threads.nthreads()

      cs = [round(Int, size(o.v,2)/nt\*i) for i ∈ 0:nt]

      @inbounds Threads.@threads for t ∈ 1:nt

        δdenom\_b = zeros(T, o.Repl.kZ, o.Repl.kZ)  # thread-safe scratch pad

        @inbounds for b ∈ cs[t]+1:cs[t+1]

          for i ∈ 1:o.Repl.kZ

            δdenom\_b[1:i,i] = view(δdenom[i],:,b)  # fill uppper triangle

          end

          β̈s[:,b] = (A[b] = invsym(δdenom\_b)) \* view(δnumer,:,b)

        end

      end

    end

    if o.bootstrapt

      if o.robust

        J⋂s = o.J⋂s[isone(o.Nw) || w<o.Nw ? 1 : 2]

        @inbounds for i ∈ 1:o.Repl.kZ  # avoid list comprehension construction for compiler-perceived type stability

          Filling!(o, view(J⋂s,:,:,i), i, β̈s)

        end

      else

        YY✻ = [HessianFixedkappa(o, collect(i:o.Repl.kZ), i, zero(T), w) for i ∈ 0:o.Repl.kZ]  # κ=0 => Y\*MZperp\*Y

      end

    end

    @inbounds for b ∈ reverse(axes(o.v,2))

      o.numer\_b .= o.null || w==1 && b==1 ? o.Repl.RRpar \* view(β̈s,:,b) + o.Repl.Rt₁ - o.r : o.Repl.RRpar \* (view(β̈s,:,b) - o.DGP.β̈₀)

      if o.bootstrapt

        if o.robust  # Compute denominator for this WRE test stat

          J⋂ = view(J⋂s,:,b,:) \* (A[b] \* o.Repl.RRpar')

          for c ∈ 1:o.NErrClustCombs

            (!isone(o.NClustVar) && nrows(o.clust[c].order)>0) &&

              (J⋂ = J⋂[o.clust[c].order,:])

            J\_b = @panelsum(o, J⋂, o.clust[c].info)

            @clustAccum!(denom, c, J\_b'J\_b)

          end

        else  # non-robust

          for i ∈ 0:o.Repl.kZ

            o.YY✻\_b[i+1,i+1:o.Repl.kZ+1] = view(YY✻[i+1],:,b)  # fill upper triangle

          end

          denom = (o.Repl.RRpar \* A[b] \* o.Repl.RRpar') \* [-one(T) ; β̈s[:,b]]'Symmetric(o.YY✻\_b) \* [-one(T) ; β̈s[:,b]] / o.\_Nobs  # 2nd half is sig2 of errors

        end

        o.dist[b+first(o.WeightGrp[w])-1] = o.sqrt ? o.numer\_b[1] / sqrtNaN(denom[1]) : o.numer\_b'invsym(denom)\*o.numer\_b  # hand-code for 2-dimensional?

      end

      o.numer[:,b+first(o.WeightGrp[w])-1] = o.numer\_b  # slight inefficiency: in usual bootstrap-t case, only need to save numerators in numer if getdist("numer") is coming because of svmat(numer)

    end

    w==1 && o.bootstrapt && (o.statDenom = denom)  # original-sample denominator

  end

  nothing

end