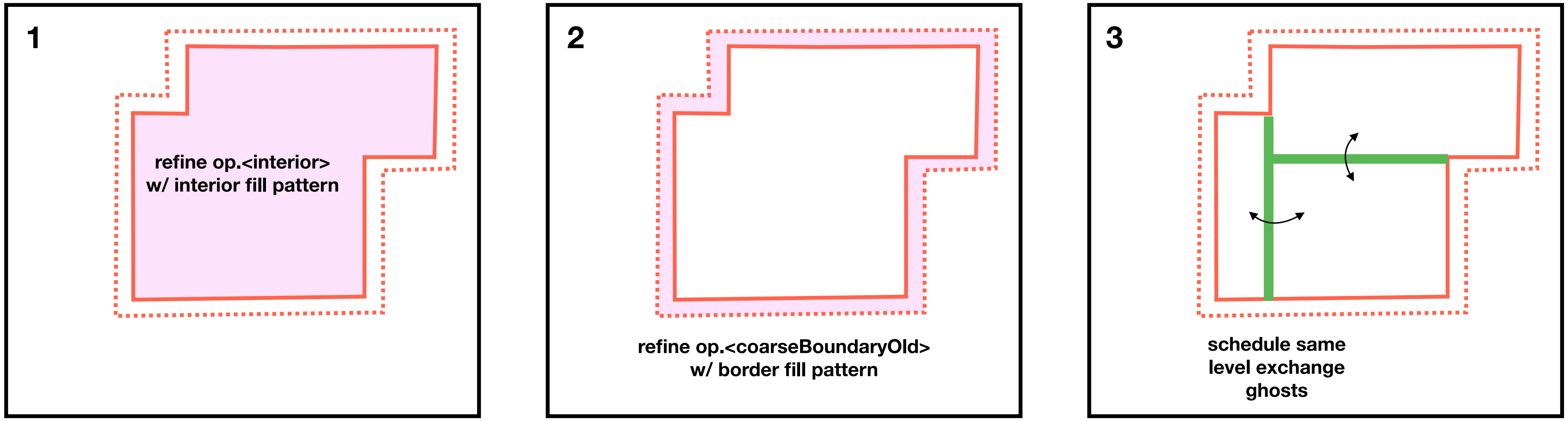
New level



1

refine op.<interior>
w/ interior fill pattern

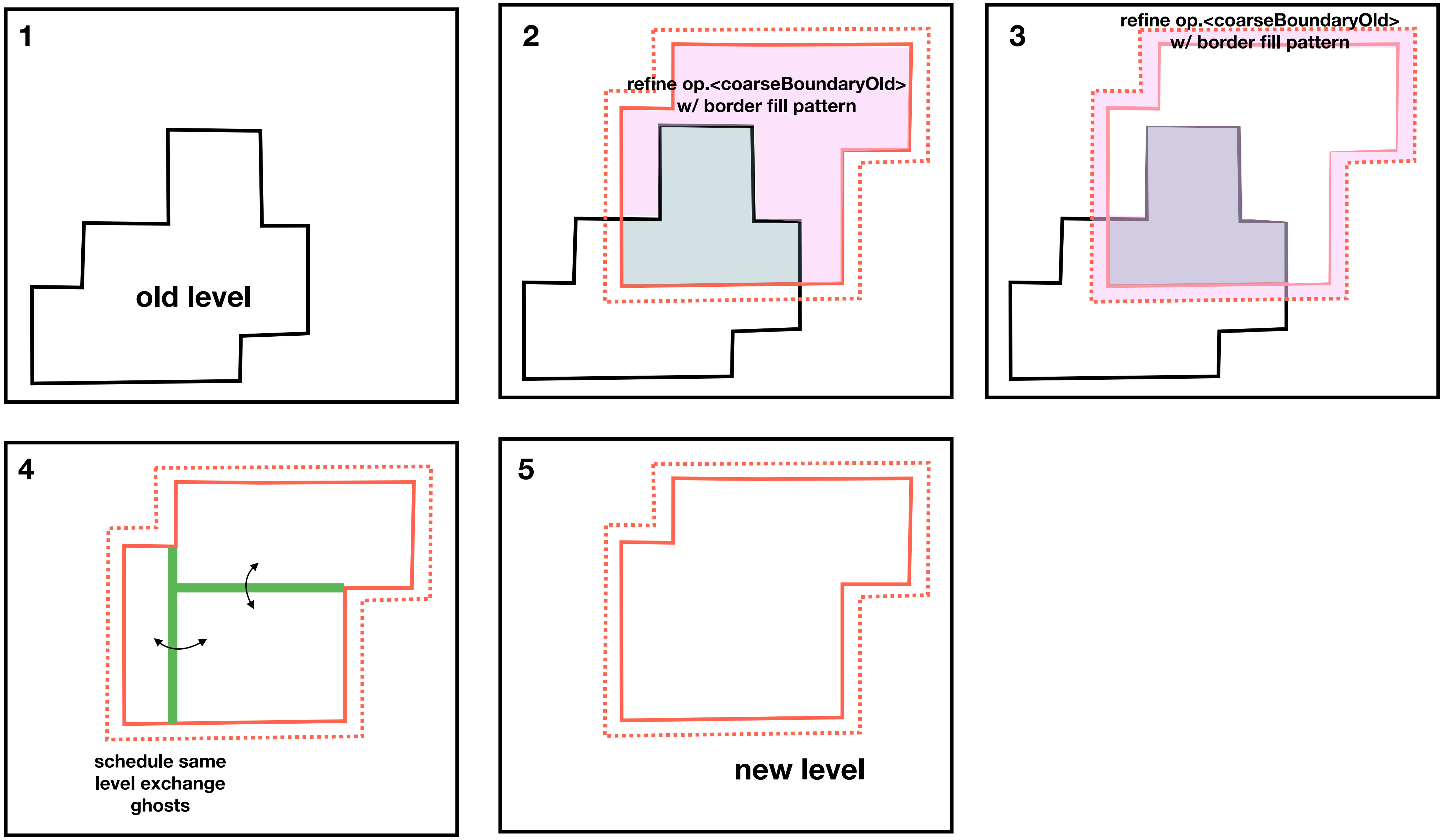
2

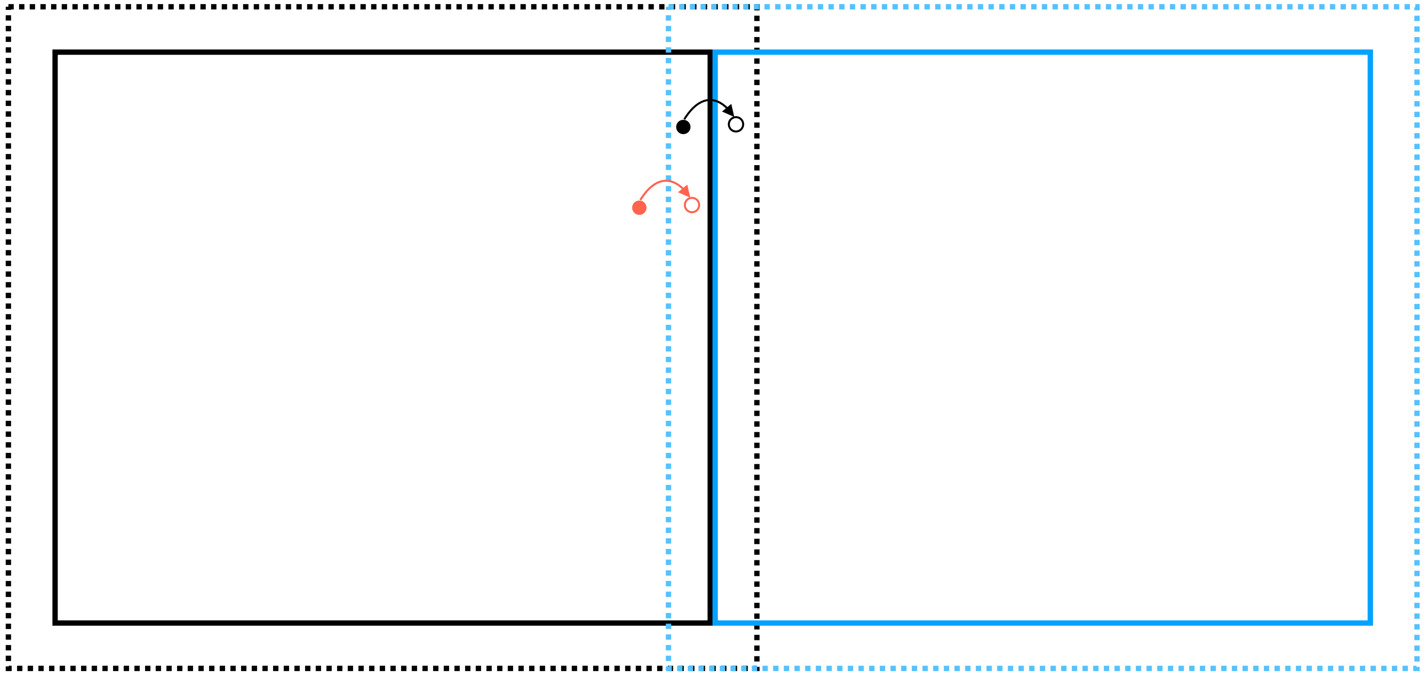
refine op.<coarseBoundaryOld>
w/ border fill pattern

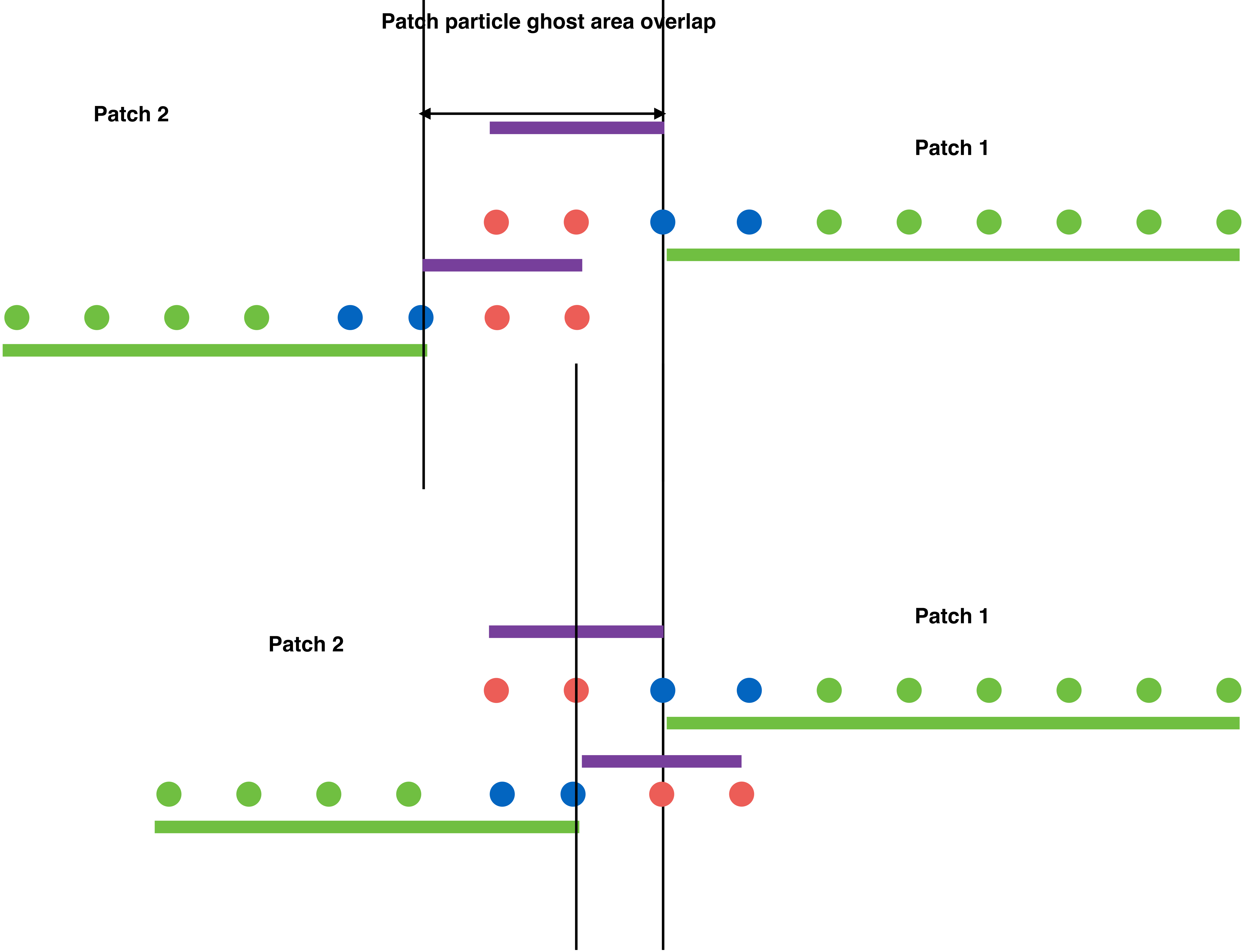
3

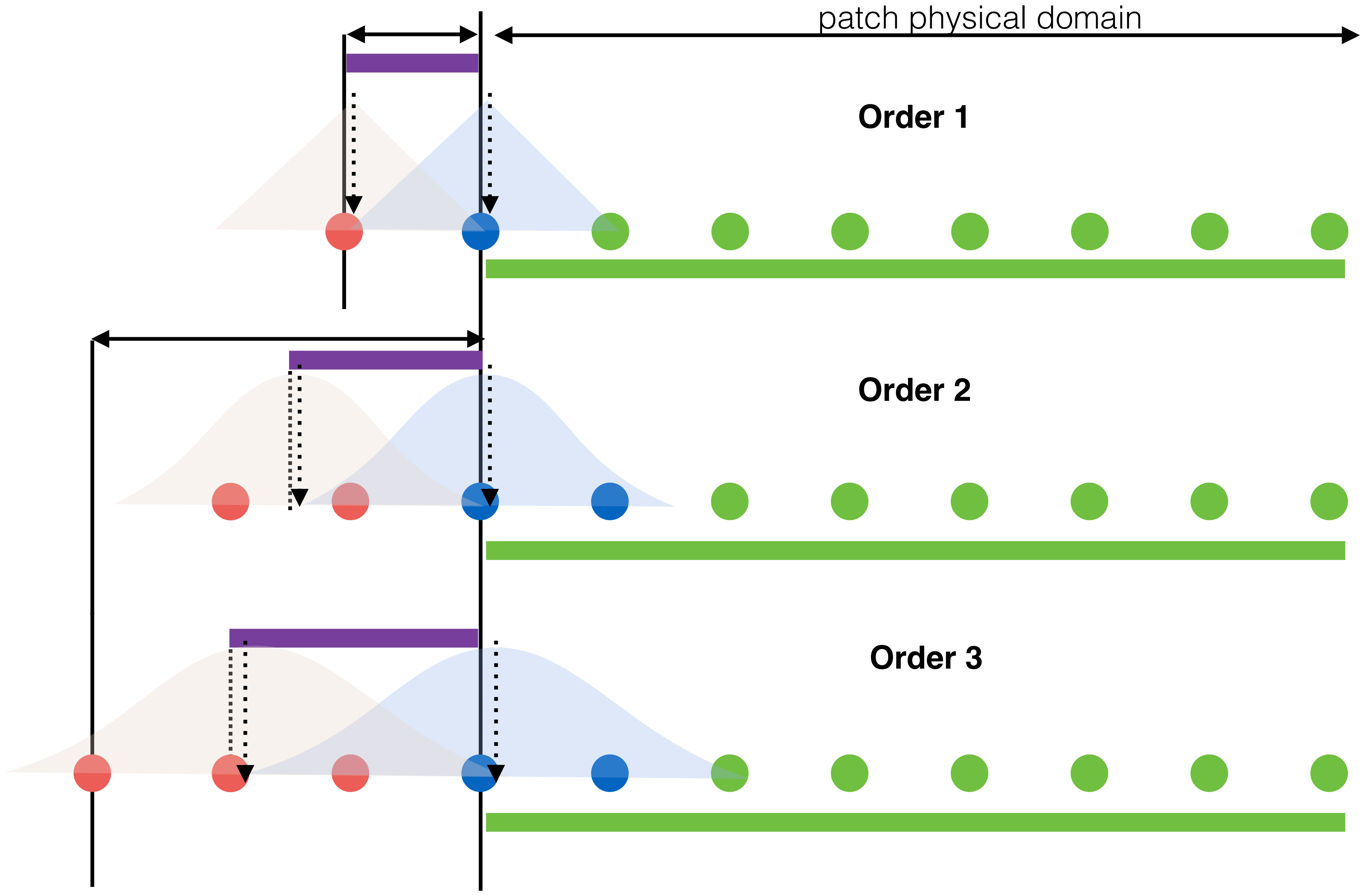
schedule same
level exchange
ghosts

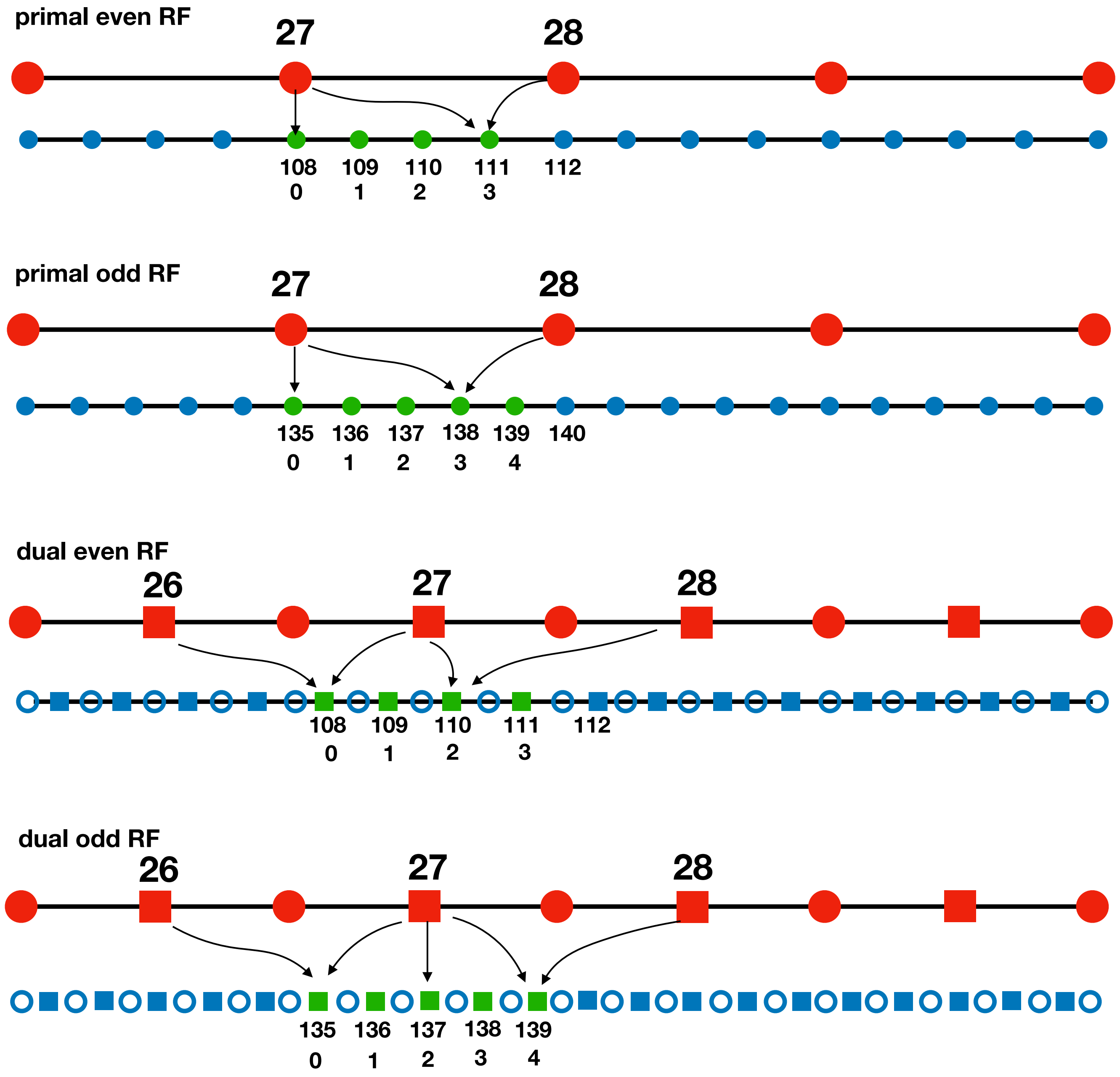
Regridding

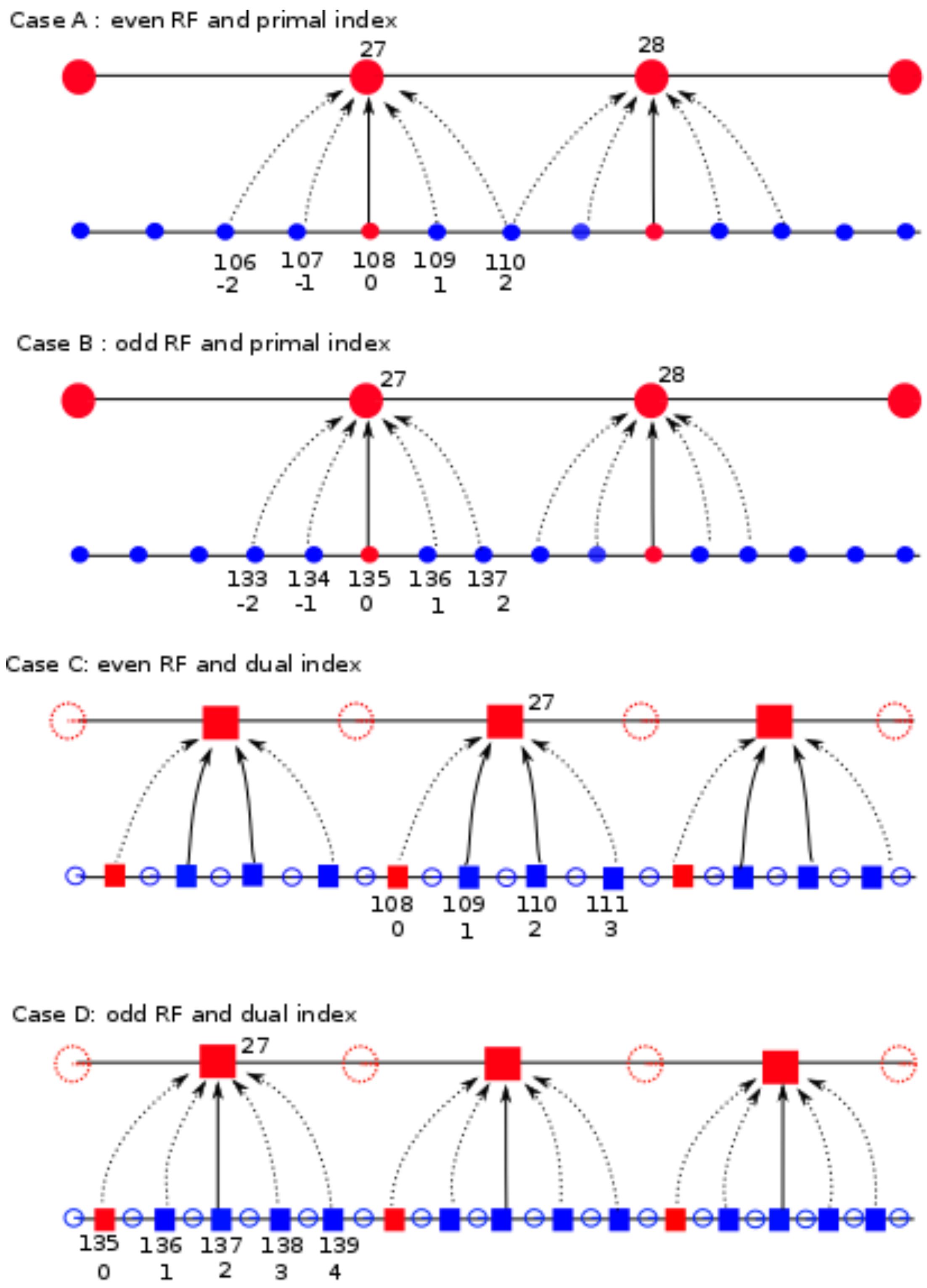

1
old level
2
refine op.<coarseBoundaryOld>
w/ border fill pattern
3
refine op.<coarseBoundaryOld>
w/ border fill pattern
4
schedule same
level exchange
ghosts
5
new level




Patch particle ghost area overlap
Patch 2
Patch 1
Patch 2
Patch 1


patch physical domain
Order 1
Order 2
Order 3


primal even RF
27
28
108
0
109
1
110
2
111
3
112
primal odd RF
27
28
135
0
136
1
137
2
138
3
139
4
140
dual even RF
26
27
28
108
0
109
1
110
2
111
3
112
dual odd RF
26
27
28
135
0
136
1
137
2
138
3
139
4

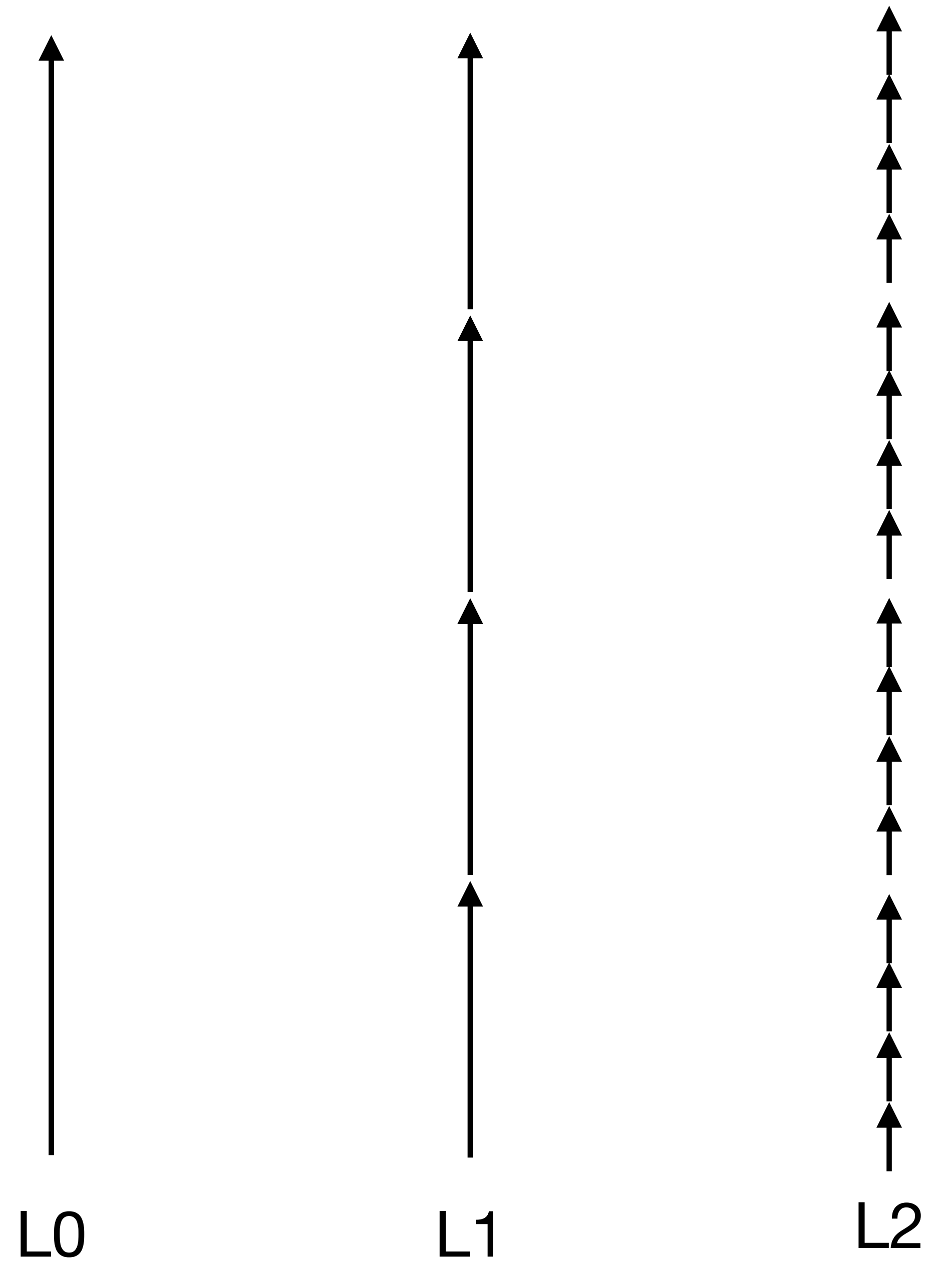

Case A : even RF and primal index

Case B : odd RF and primal index

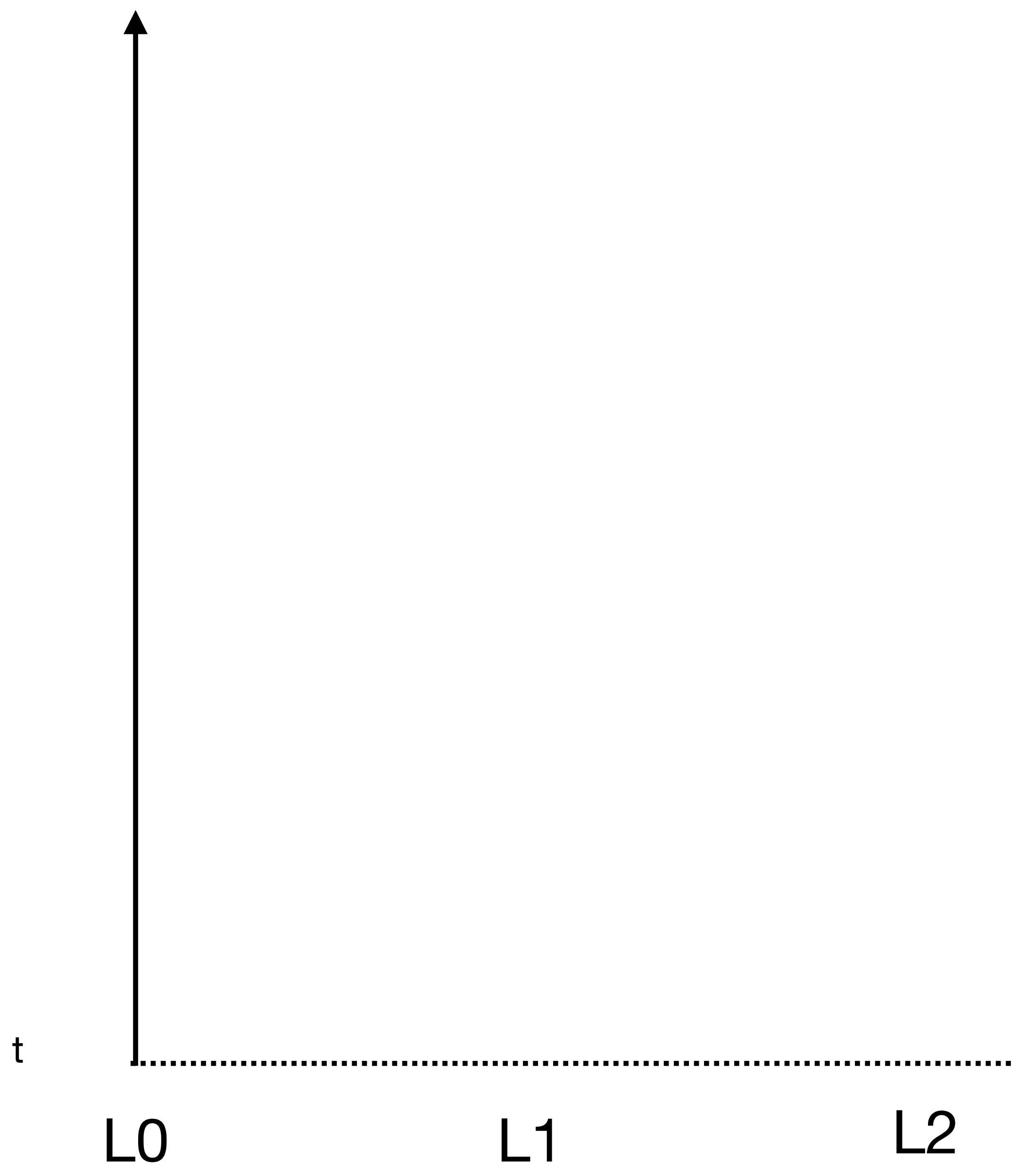
Case C: even RF and dual index

Case D: odd RF and dual index

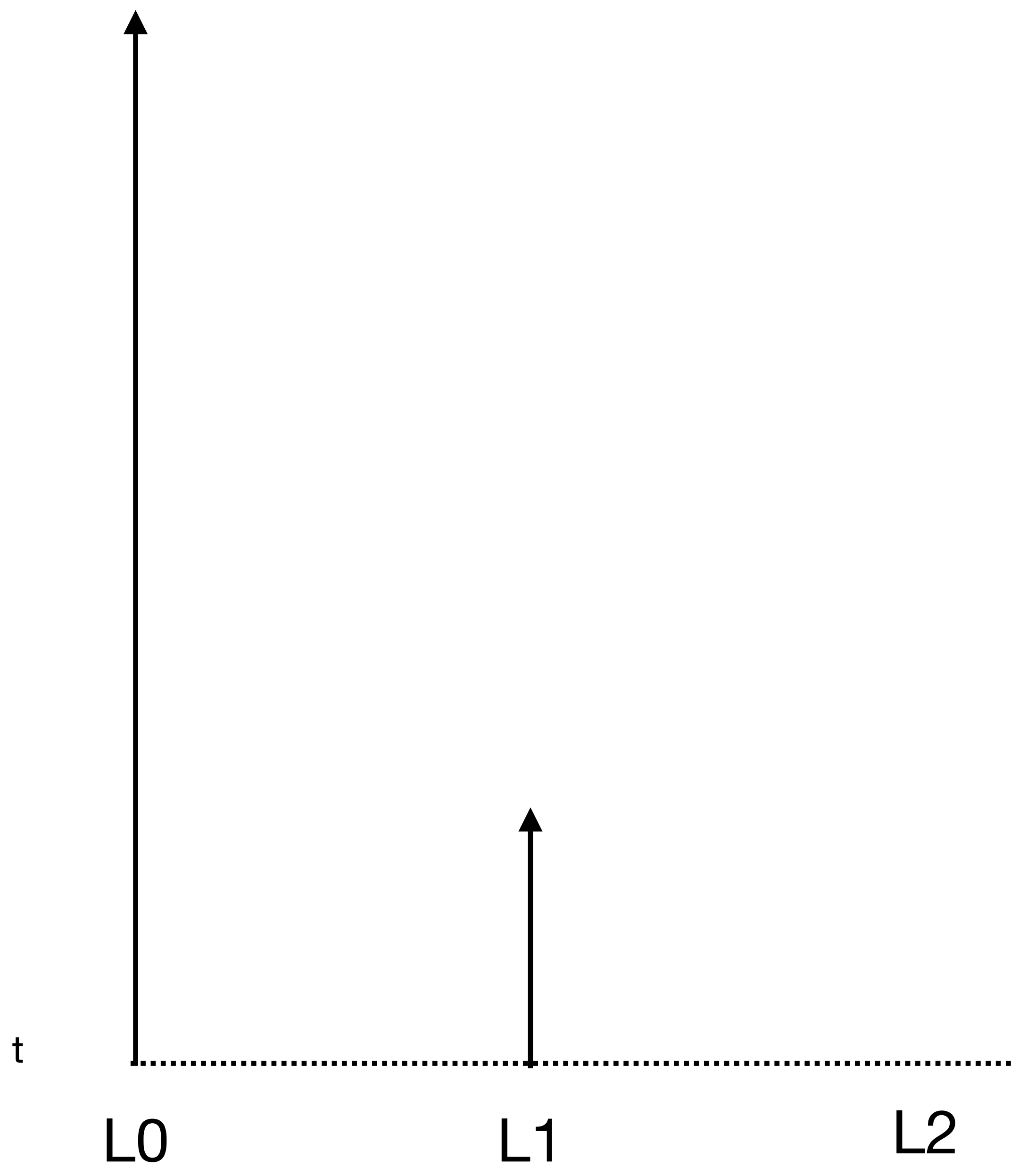
Recursive time advance by the TimeRefinementLevelIntegrator



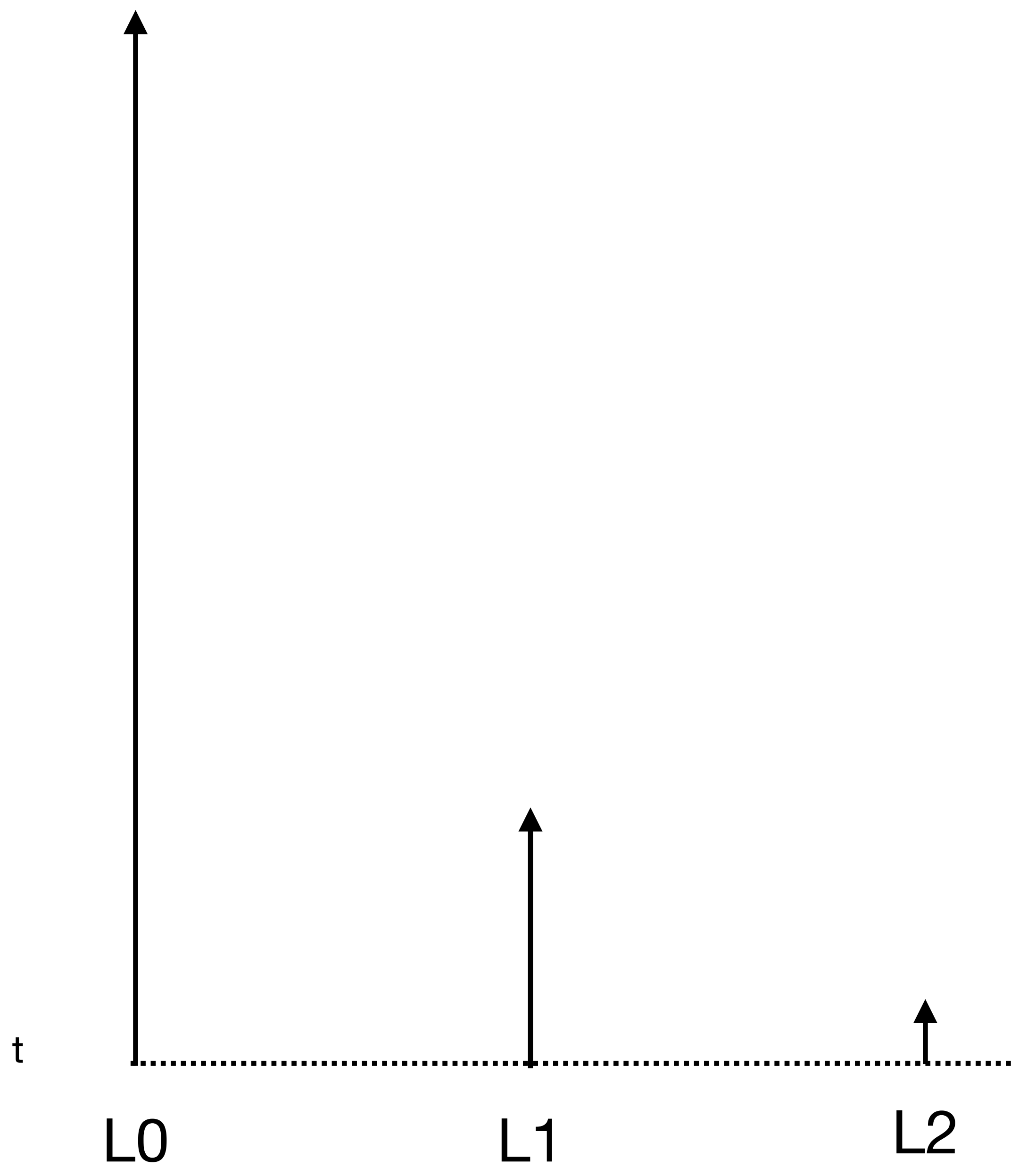
Recursive time advance by the TimeRefinementLevelIntegrator



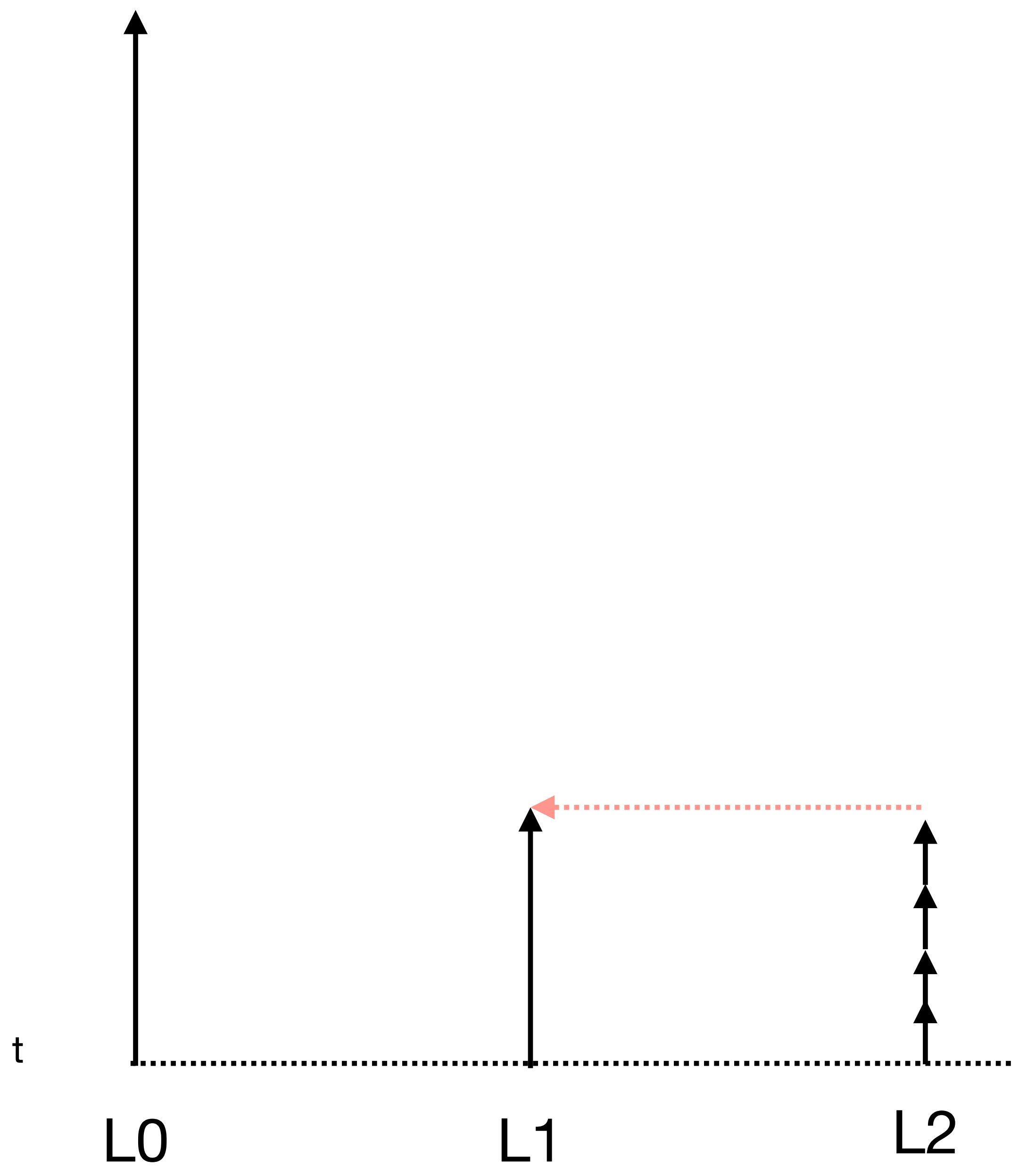
Recursive time advance by the TimeRefinementLevelIntegrator



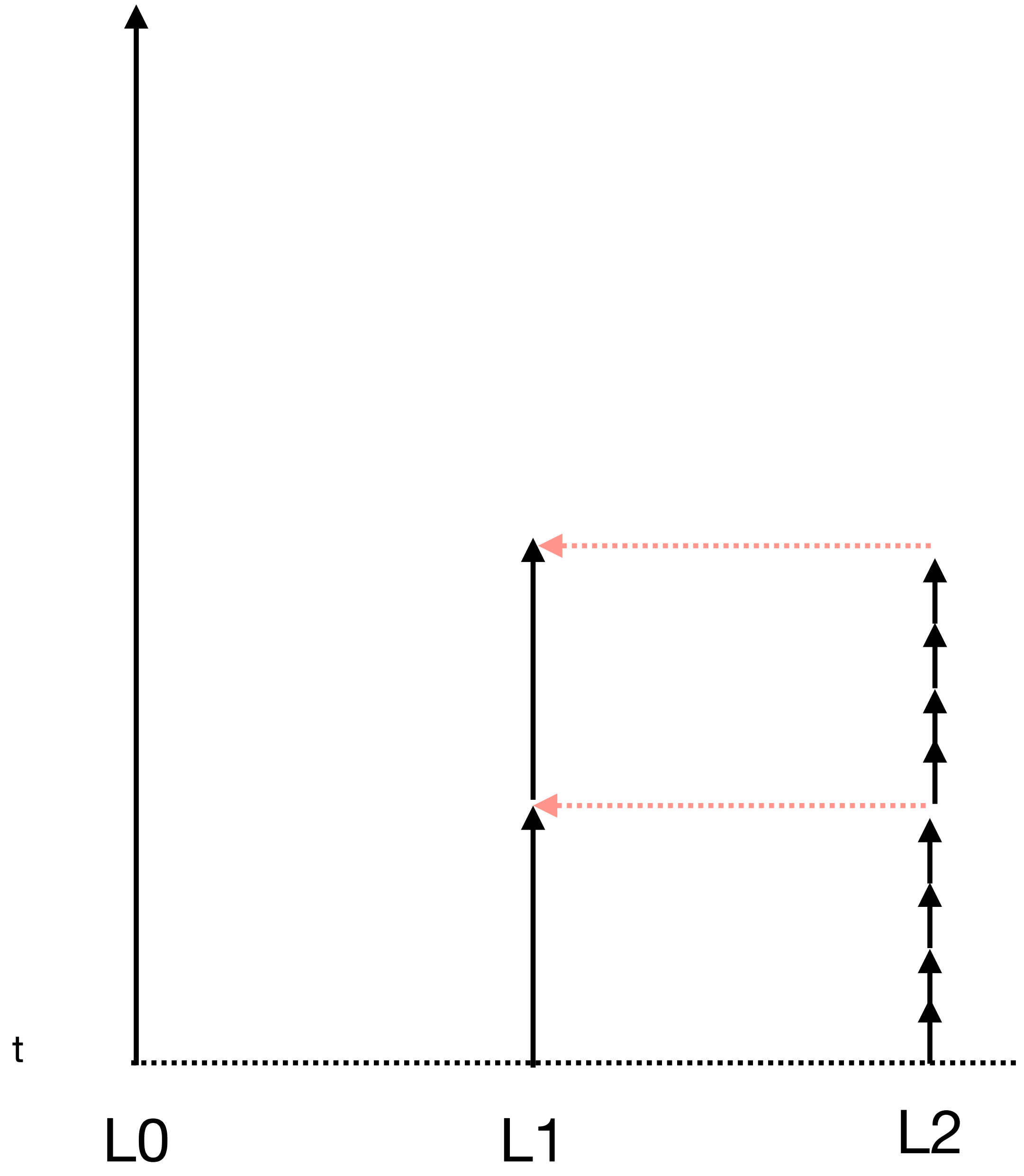
Recursive time advance by the TimeRefinementLevelIntegrator



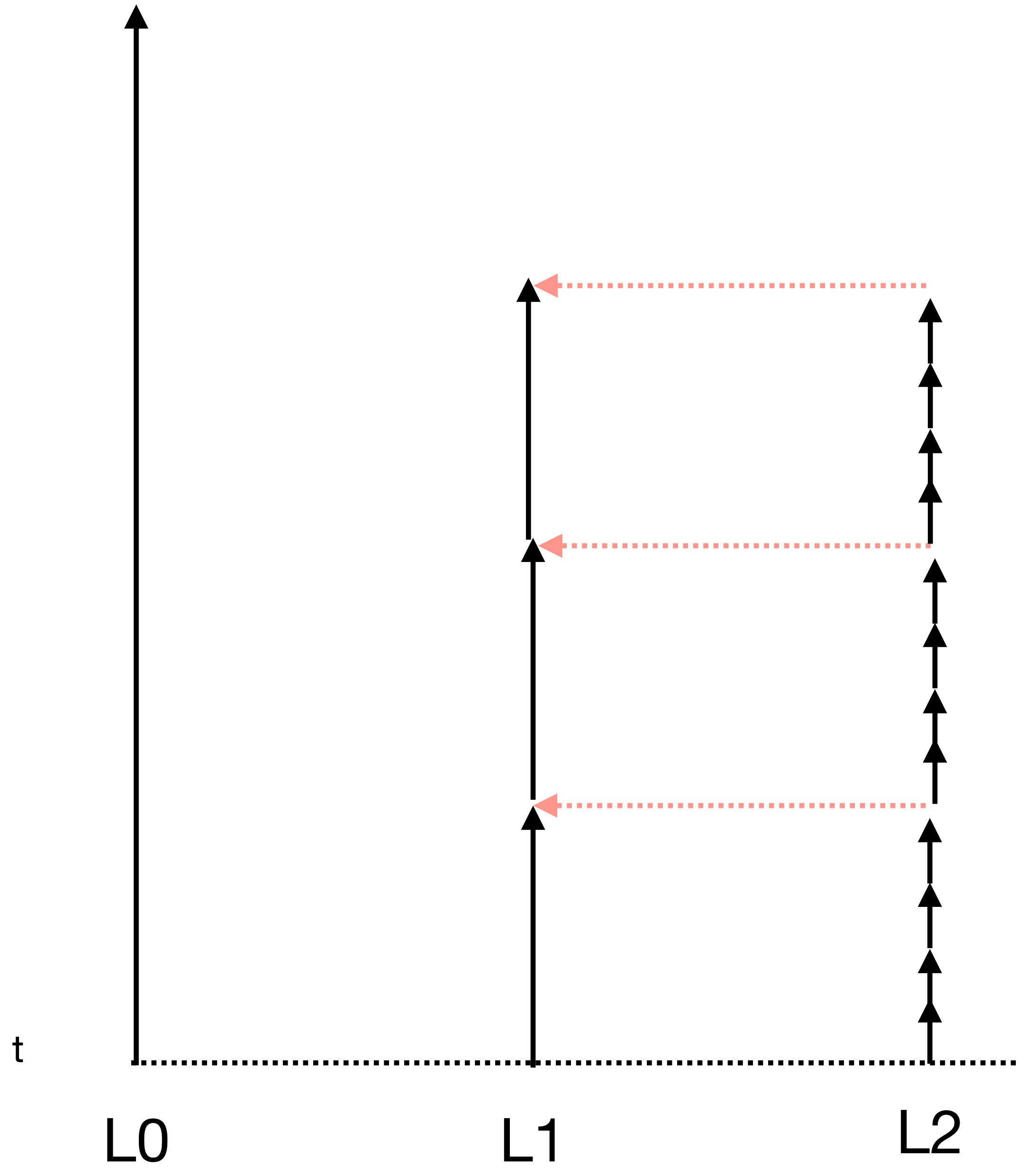
Recursive time advance by the TimeRefinementLevelIntegrator



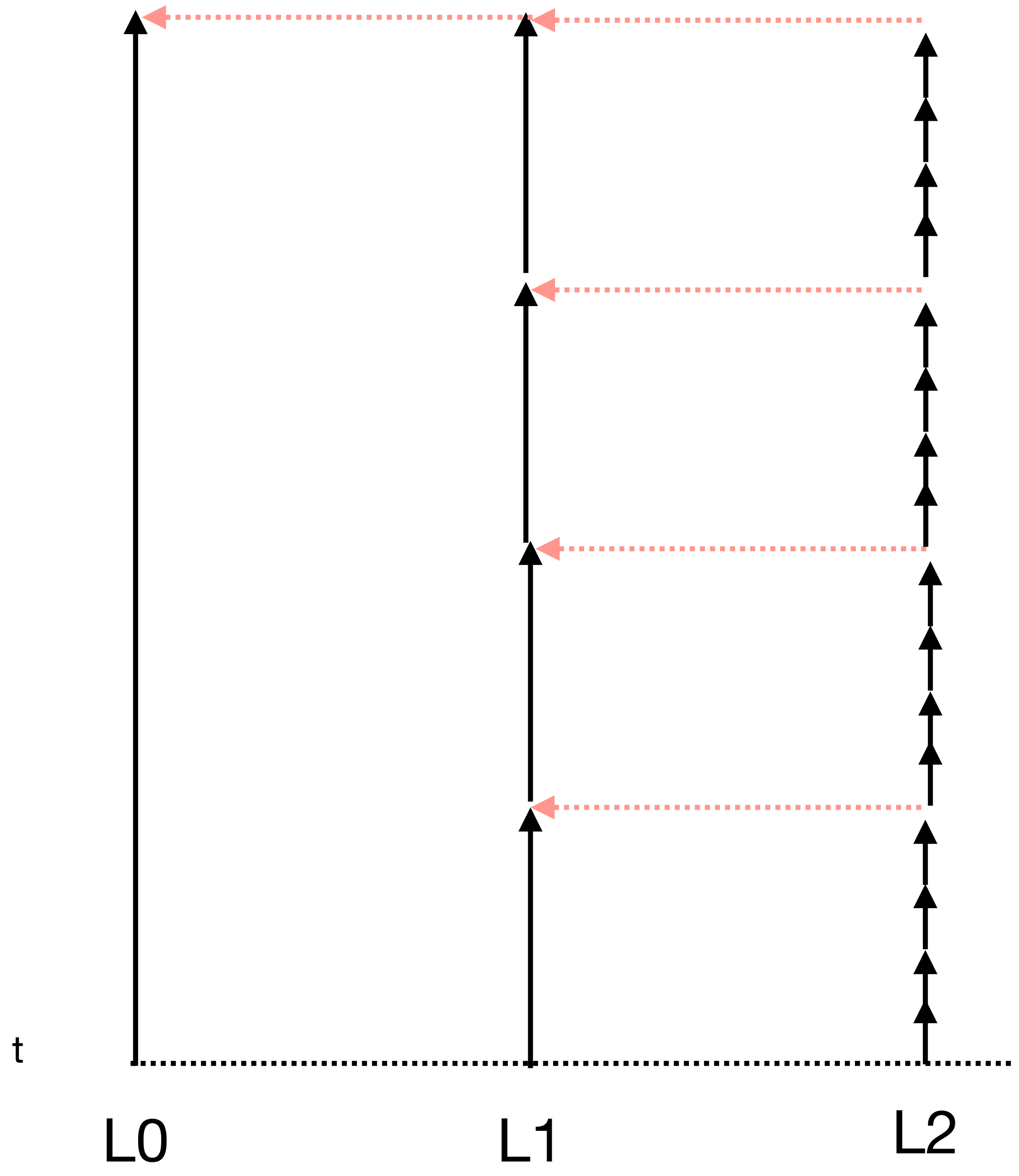
Recursive time advance by the TimeRefinementLevelIntegrator



Recursive time advance by the TimeRefinementLevelIntegrator



Recursive time advance by the TimeRefinementLevelIntegrator



InitializeLevelData()

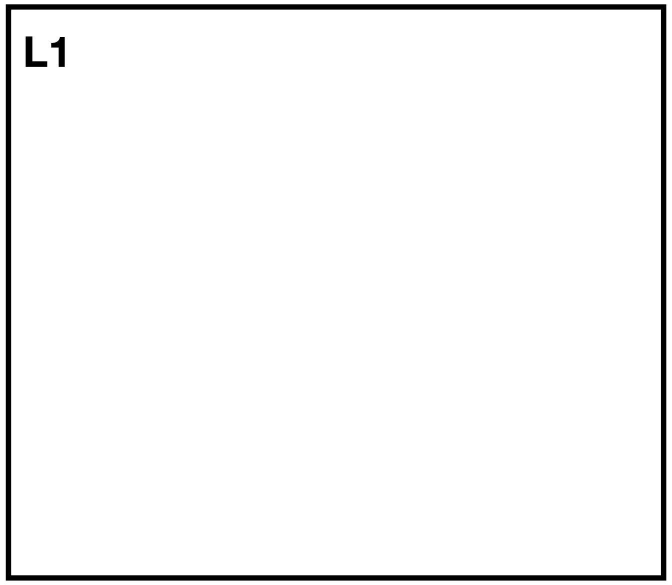
Initializing the root level

t+dt0
t
L0
L1
L2

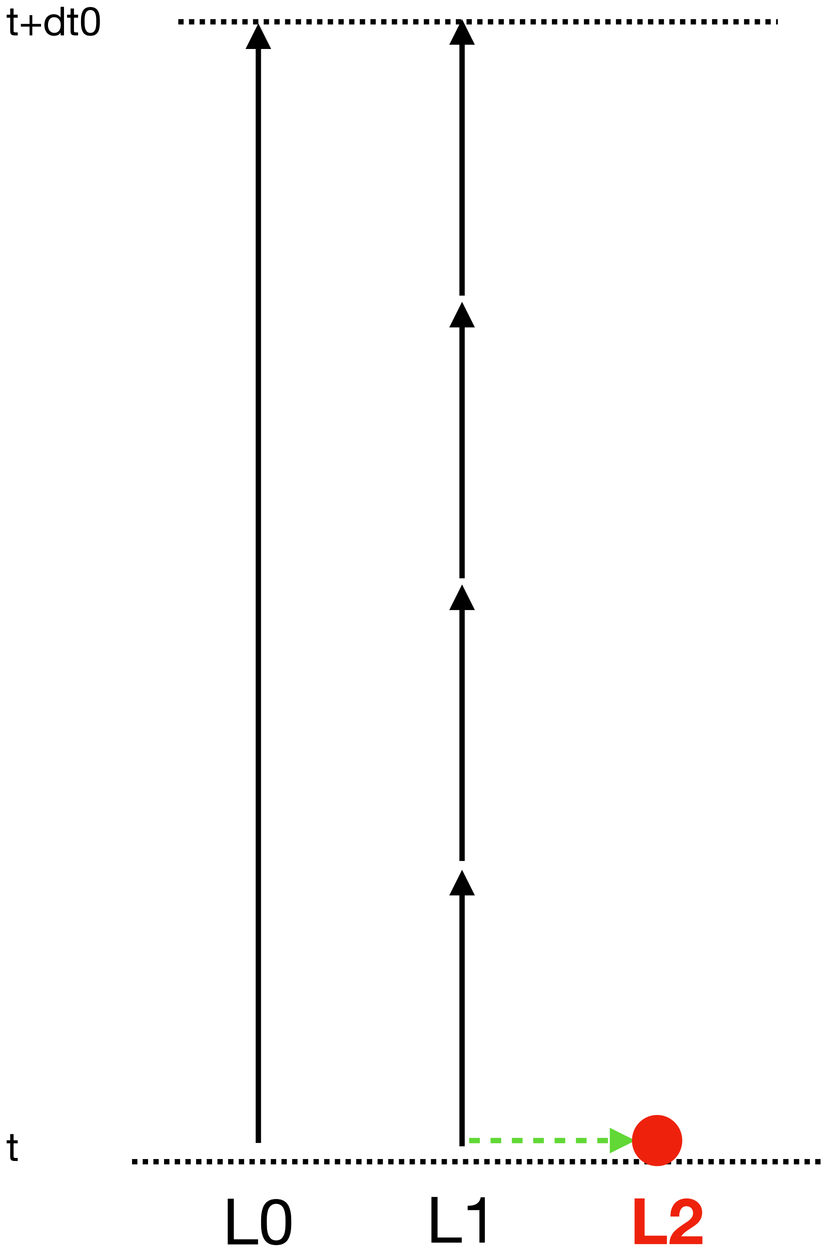
Two possibilities:

- initializing from a user input
- initializing from a checkpoint

Either way it is transparent for the MultiPhysicsIntegrator.
All we do is model.initialize(*patch).



New level creation

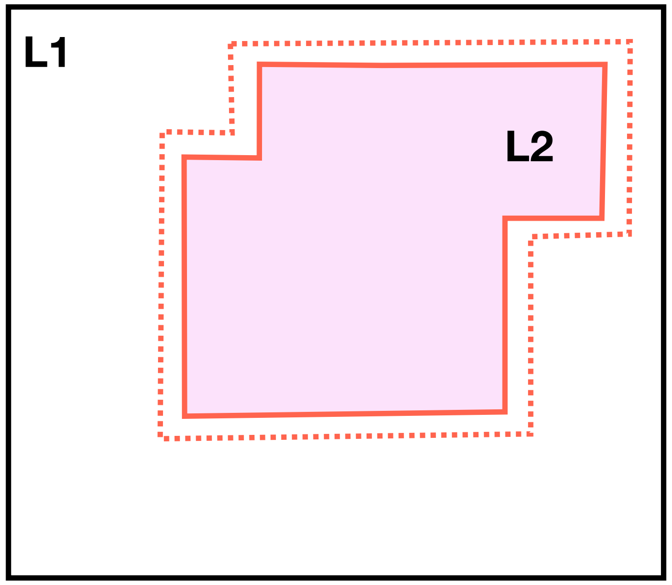


Here we create level 2. At this time, all levels are synchronized at the same time t.
SAMRAI calls `TimeRefinenementLevelStrategy::initializeLevelData()`, which is overriden by `MultiPhysicsIntegrator::initializeLevelData()`.

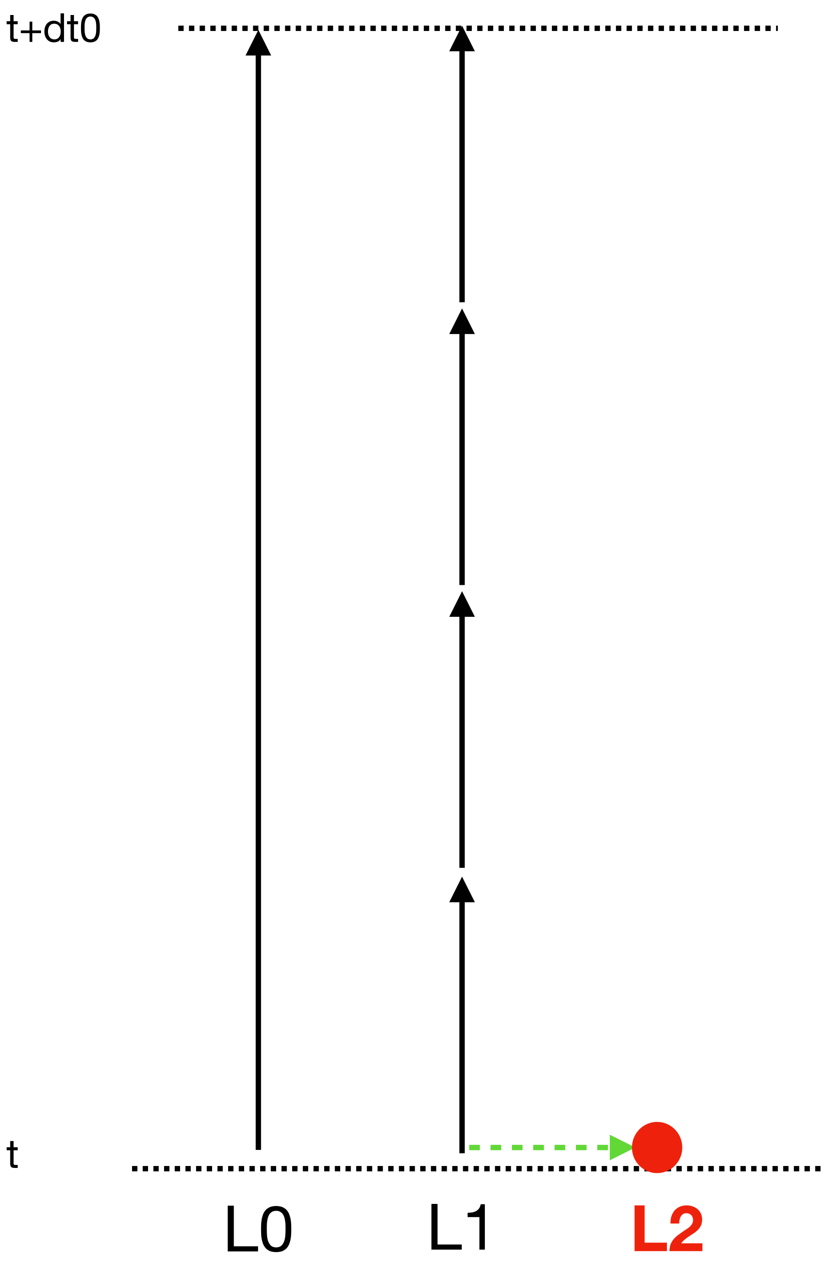
This routine needs to prepare L2 to be ready for the first advanceLevel().

This includes:

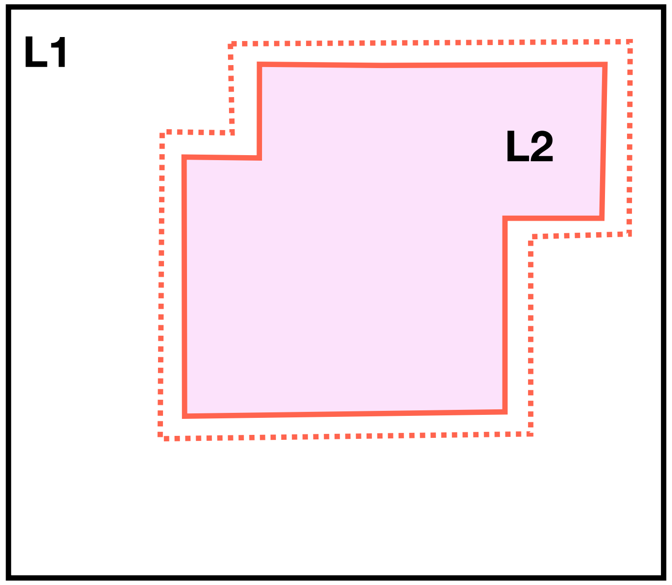
- setting E and B on interior and patch and level ghost nodes
- getting interior particles
- getting patch ghost particles
- getting level ghost particles into the ‘`levelGhostParticlesOld`’ buffer
- copying `levelGhostParticlesOld` into `levelGhostParticles`
- computing ion moments from:
    - interior particles
    - patch ghost particles
    - levelGhostParticles



New level creation



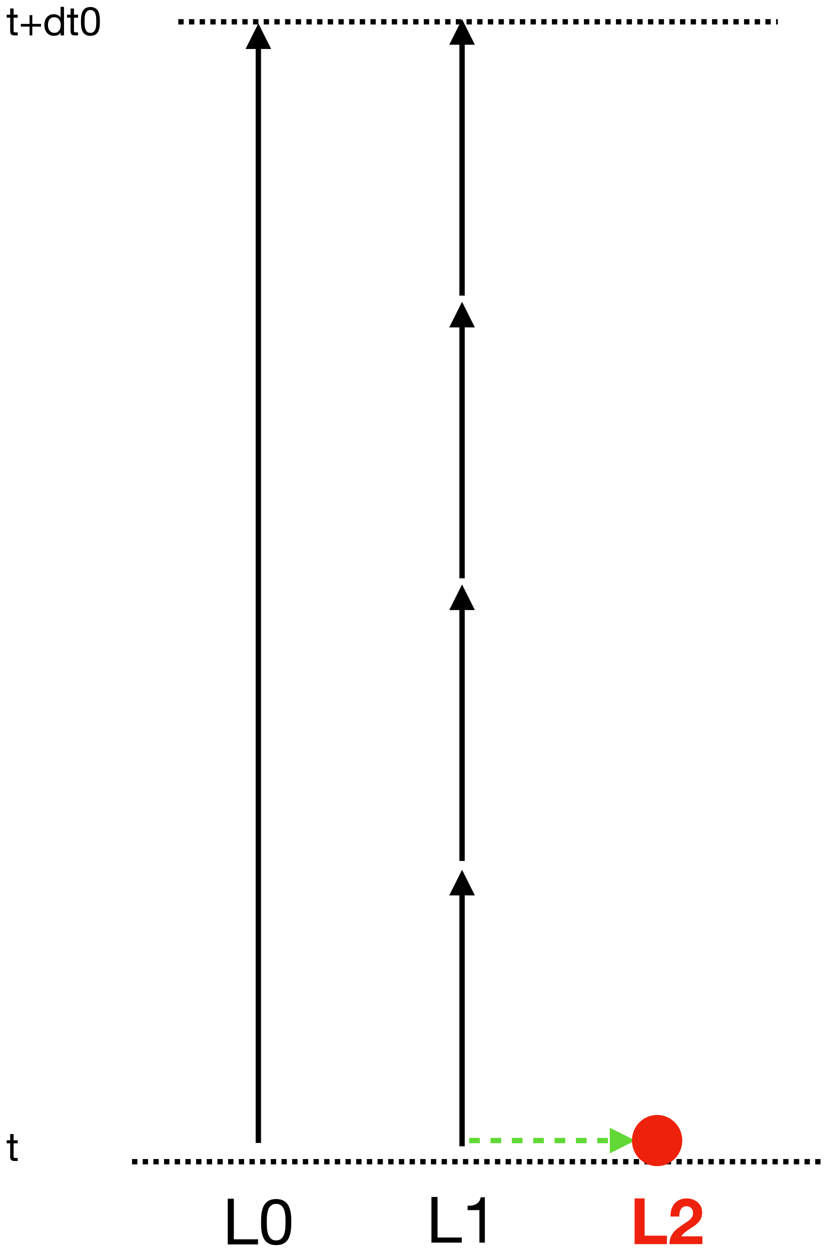
interior particles are obtained from a refinement from the coarser level.



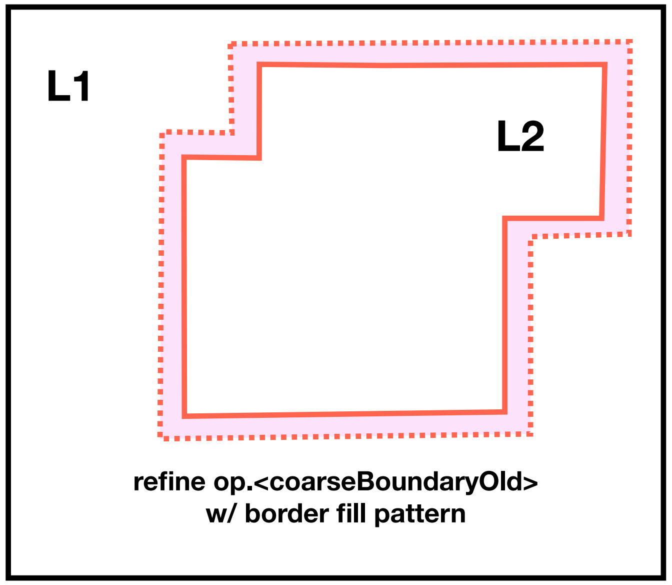
Refined particles are obtained from splitting. If the splitting algo is not deterministic then patch ghost particles of a processor will not be clones of interior particles of the neighbor proc as they should be.

Thus the schedule that refines particles into the level interior has an interior fill pattern to not fill the patch ghost cells. Patch ghost particles will be obtained in a next phase.

New level creation

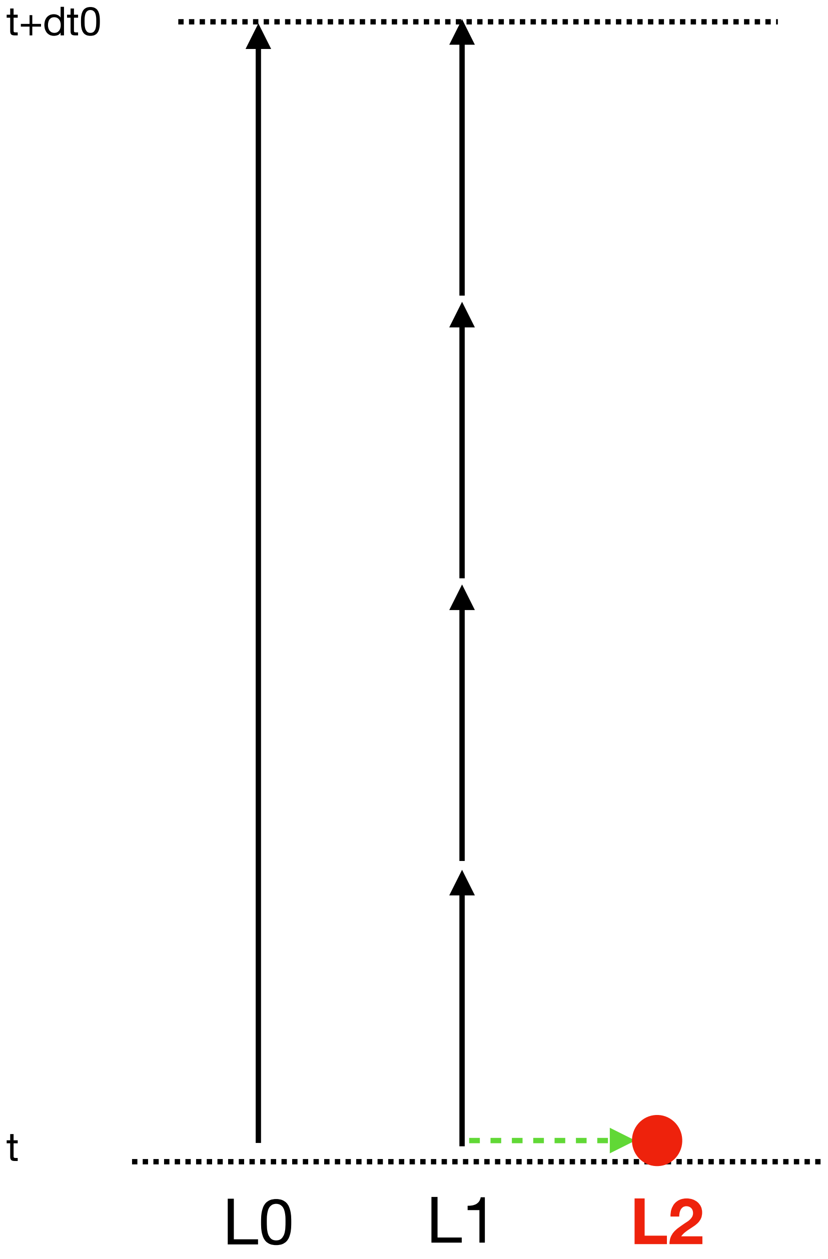


levelGhostParticles are obtained from a refinement from L1 with a restriction to the level border



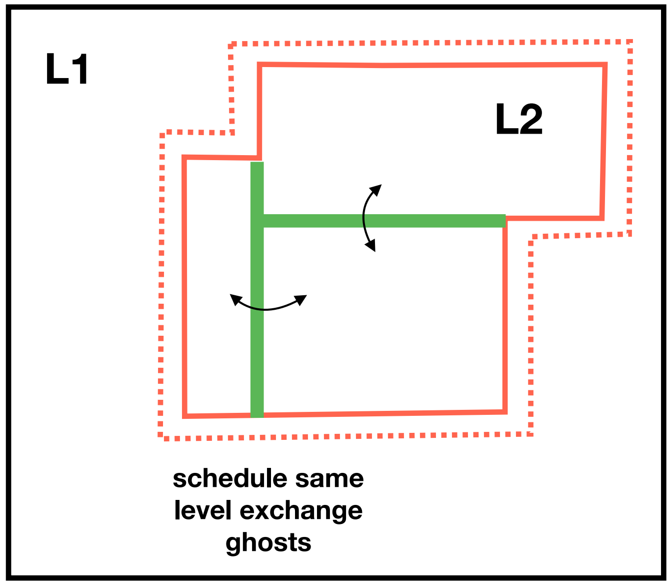
refine op.<coarseBoundaryOld>
w/ border fill pattern

New level creation



patch ghost particles are obtained from a refine schedule within L2.

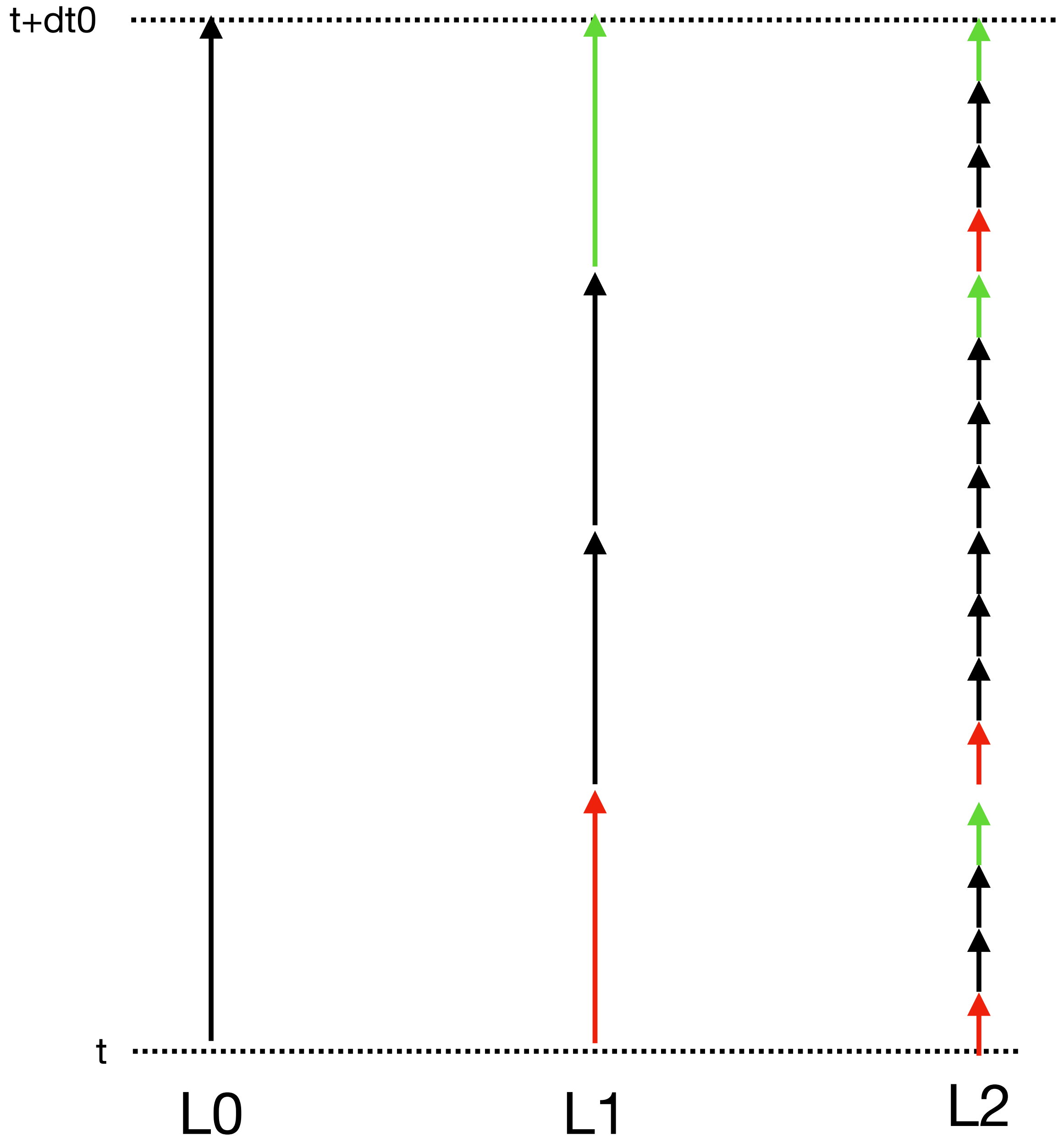
This way, they are clones of interior particles of neighbor processors.



AdvanceLevel()

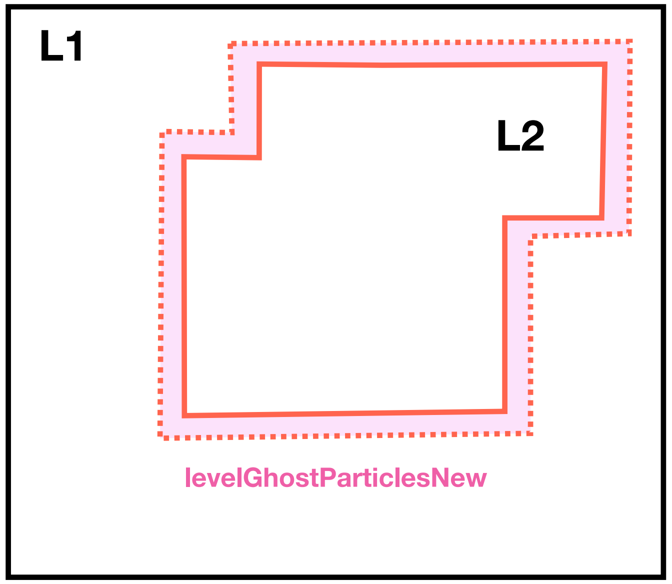
firstStep() and lastStep()

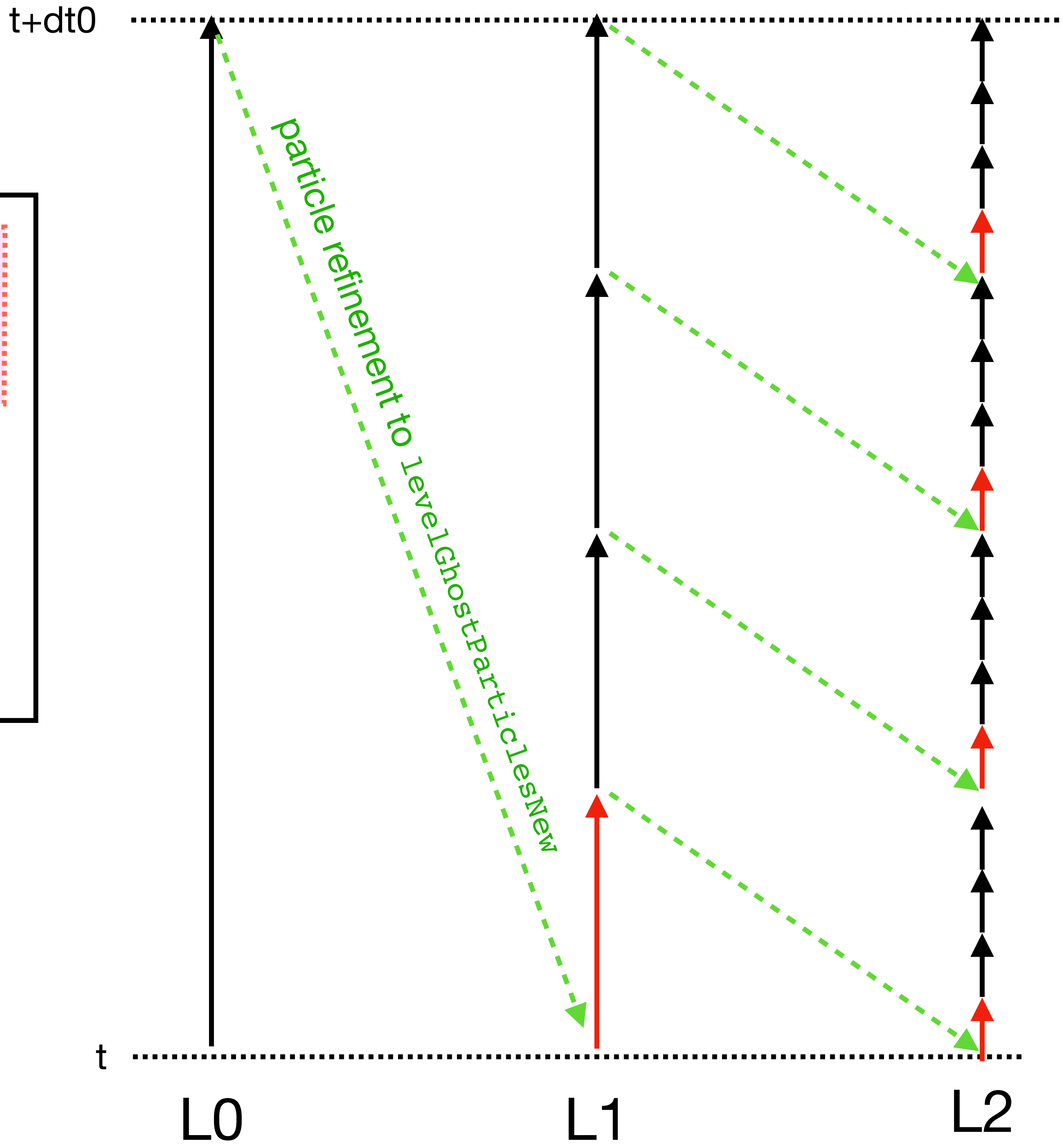
Special actions are to be performed at the first and last step of the subcycling



First step

Let’s consider the red firstStep of L1. At this step, L1 needs to get level ghost particles from L0 in `levelGhostParticlesNew`.
When L1 is at firstStep, L0 is already at t+dt0. These refined particles are thus defined at t+dt0 this is why they go into `levelGhostParticlesNew`.
Same applies for instance to the red firstStep of L2. This `levelGhostParticlesNew` buffer will be needed during `advanceLevel`() to get ion moments on level ghost nodes, using a time interpolaton between `levelGhostParticlesOld` and `levelGhostParticlesNew`.

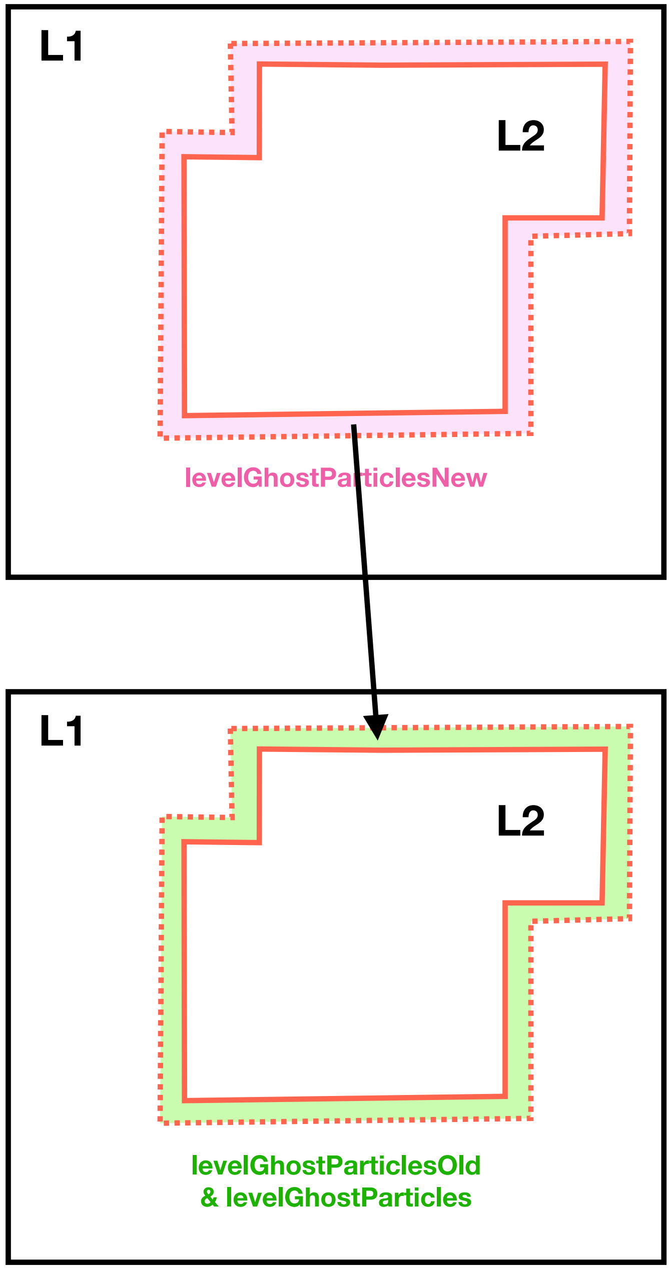


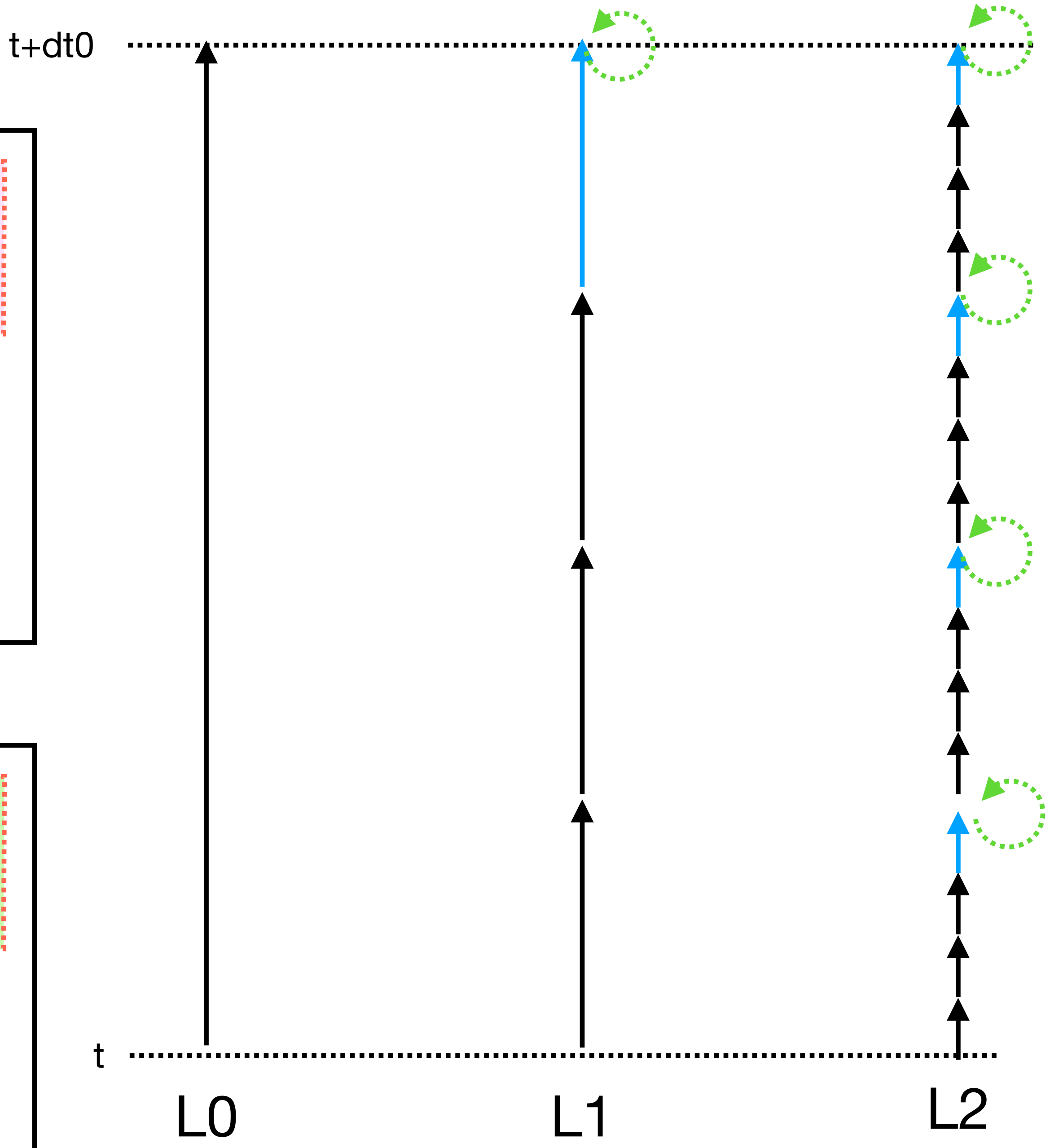


`levelGhostParticlesNew` are not changing during advance from first to last steps, like `levelGhostParticlesOld`.

Last step

At this step, we need to move `levelGhostParticlesNew` into `levelGhostParticlesOld` so that this buffer is ready for the next FirstStep.
Also, `levelGhostParticlesNew` buffer is emptied, so that at the next firstStep, it can be filled from the coarser level, as shown in previous slide.
`LevelGhostParticles` buffer is emptied and receives a copy of `levelGhostParticlesOld` particles. These two buffers need to be equal at the next firstStep so that levelGhostParticles can be emptied into the interior during the subcycling.

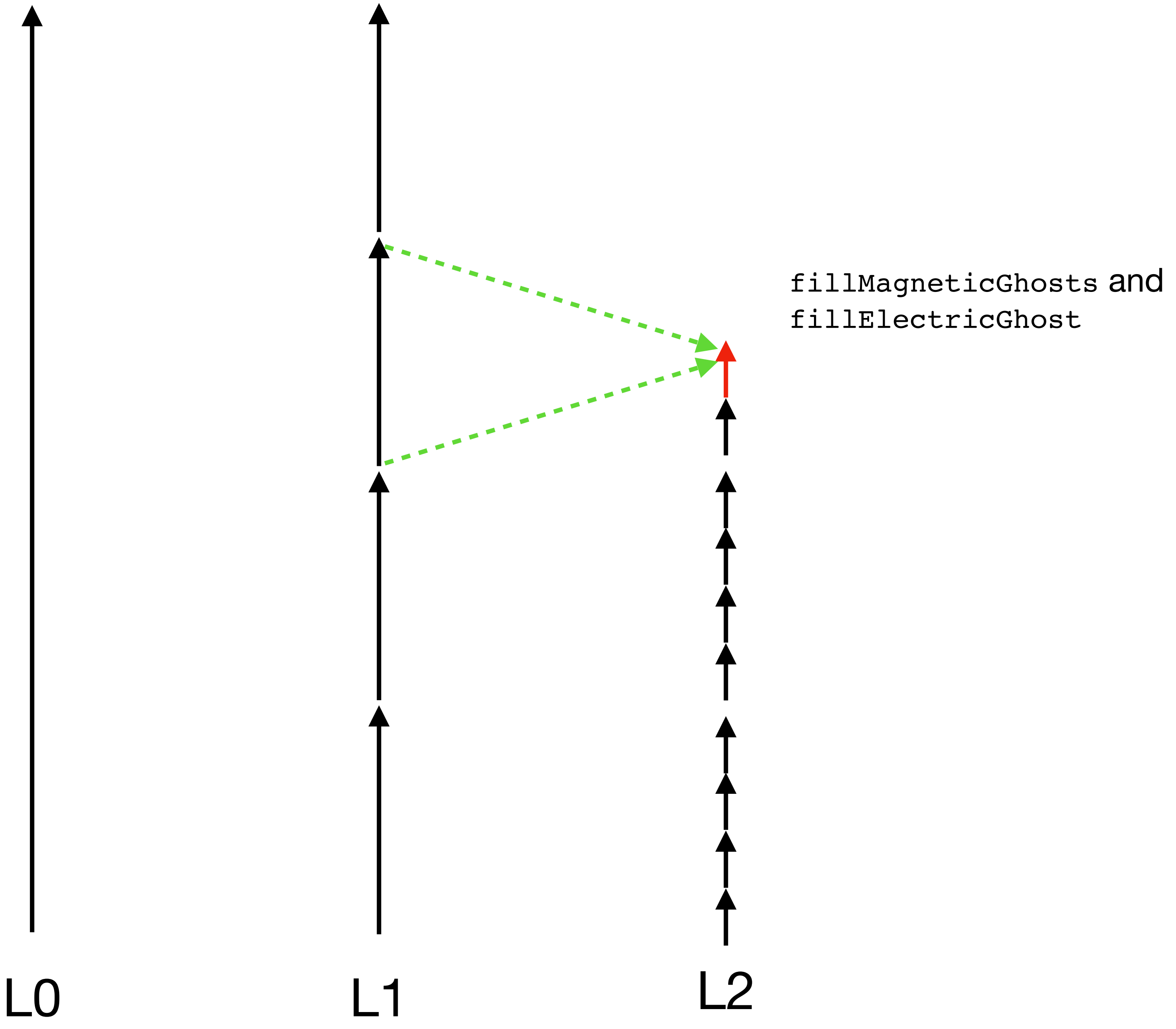




`levelGhostParticlesNew` moved into `levelGhostParticlesOld`.

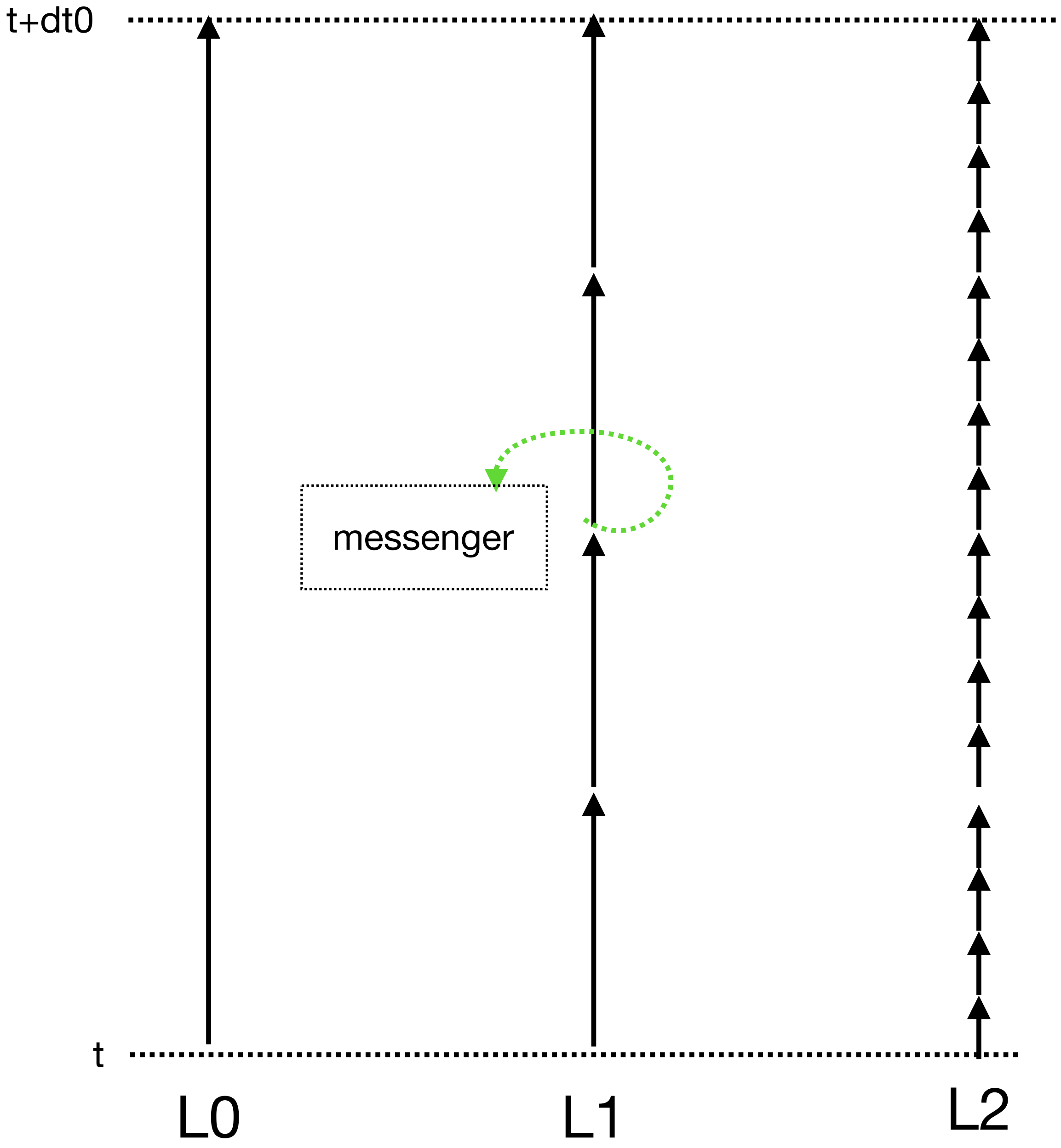
Fill electric and magnetic ghosts

communications between levels are needed to get quantities at ghost nodes.



PrepareStep

To fill the EM ghosts on level i+1, we need EM fields on level i at both time n and n+1. Because level i is at time n+1 when filling routines are called model fields are at n+1 there. We need, before advancing the model field to n+1, to save the t=n fields in the Messenger. That's what prepare step is doing. It copies EM_{t=n} into the messenger, before calling the solver::advanceLevel().



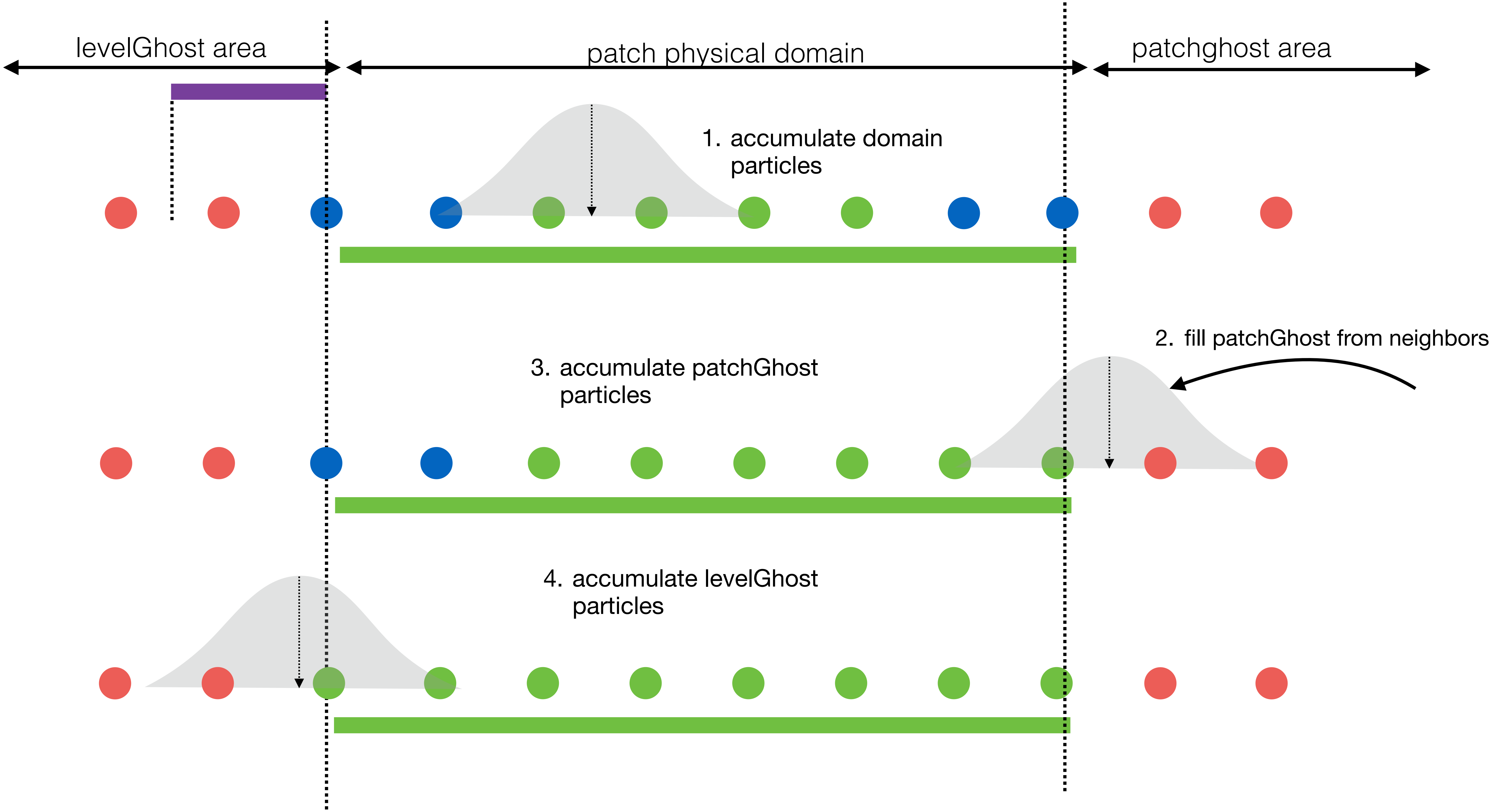
Moving and accumulating ions

When entering this function, we assume:

- domains particles are defined at time t=n
- patchGhost particles are defined at time t=n and clones of domain particles at the same time in overlaped neighbor cells
- levelGhost particles are defined at time t=n

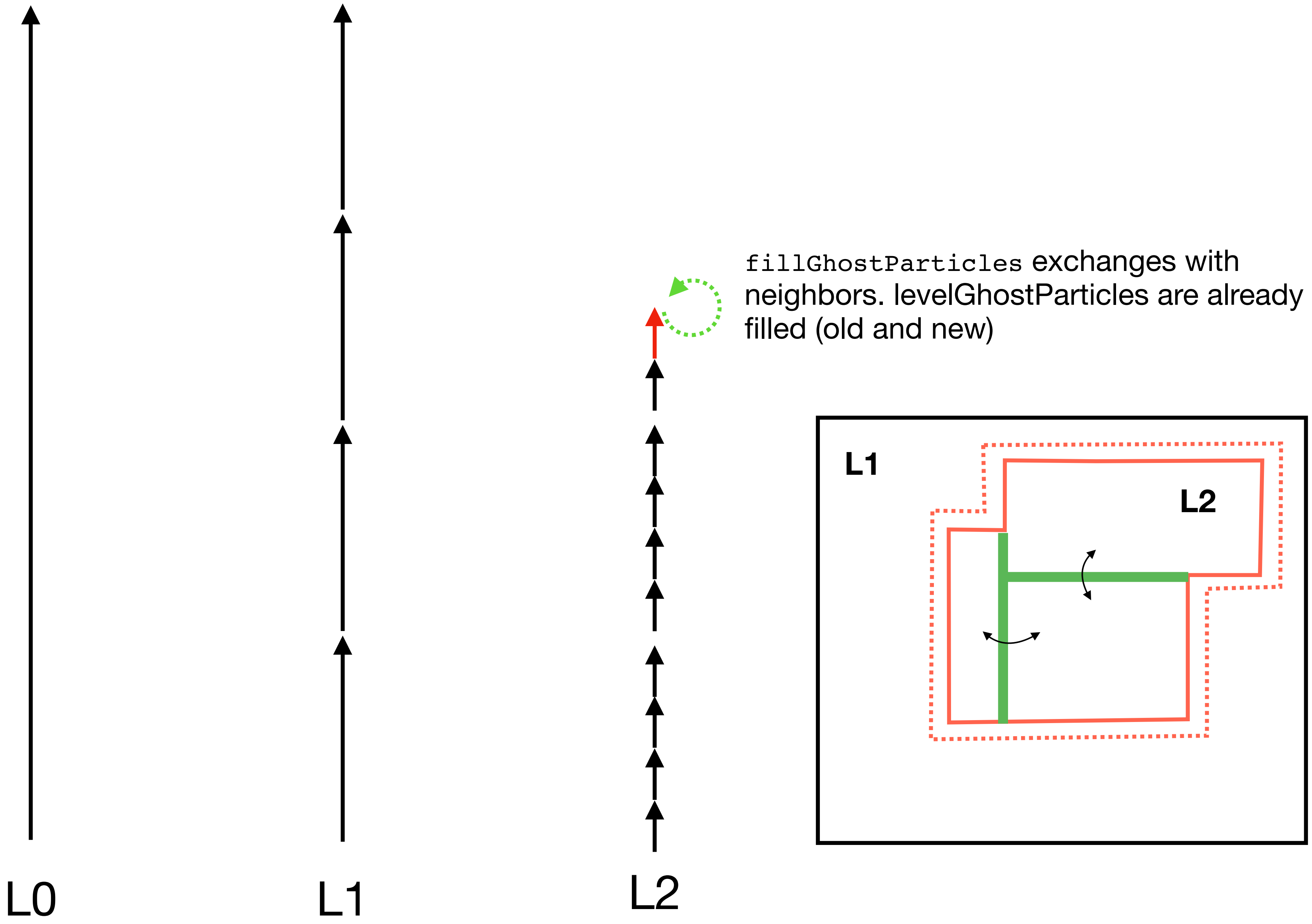
- Move domain particles and get them sorted by [first_in_domain, …., last_in_domain, first_out… last_out]
- Move patchGhost particles and have them sorted [first_in_domain, …., last_in_domain, first_out… last_out]
- Fill patchGhost areas with neighbor domain particles
- Move levelGhost particles and have them sorted [first_in_domain, …., last_in_domain, first_out… last_out]

- Accumulate all particles from first_in_domain to last_in_domain that are in:
    - domain


levelGhost area
patch physical domain
patchghost area
1. accumulate domain particles
2. fill patchGhost from neighbors
3. accumulate patchGhost particles
4. accumulate levelGhost particles

Filling patch ghost particles

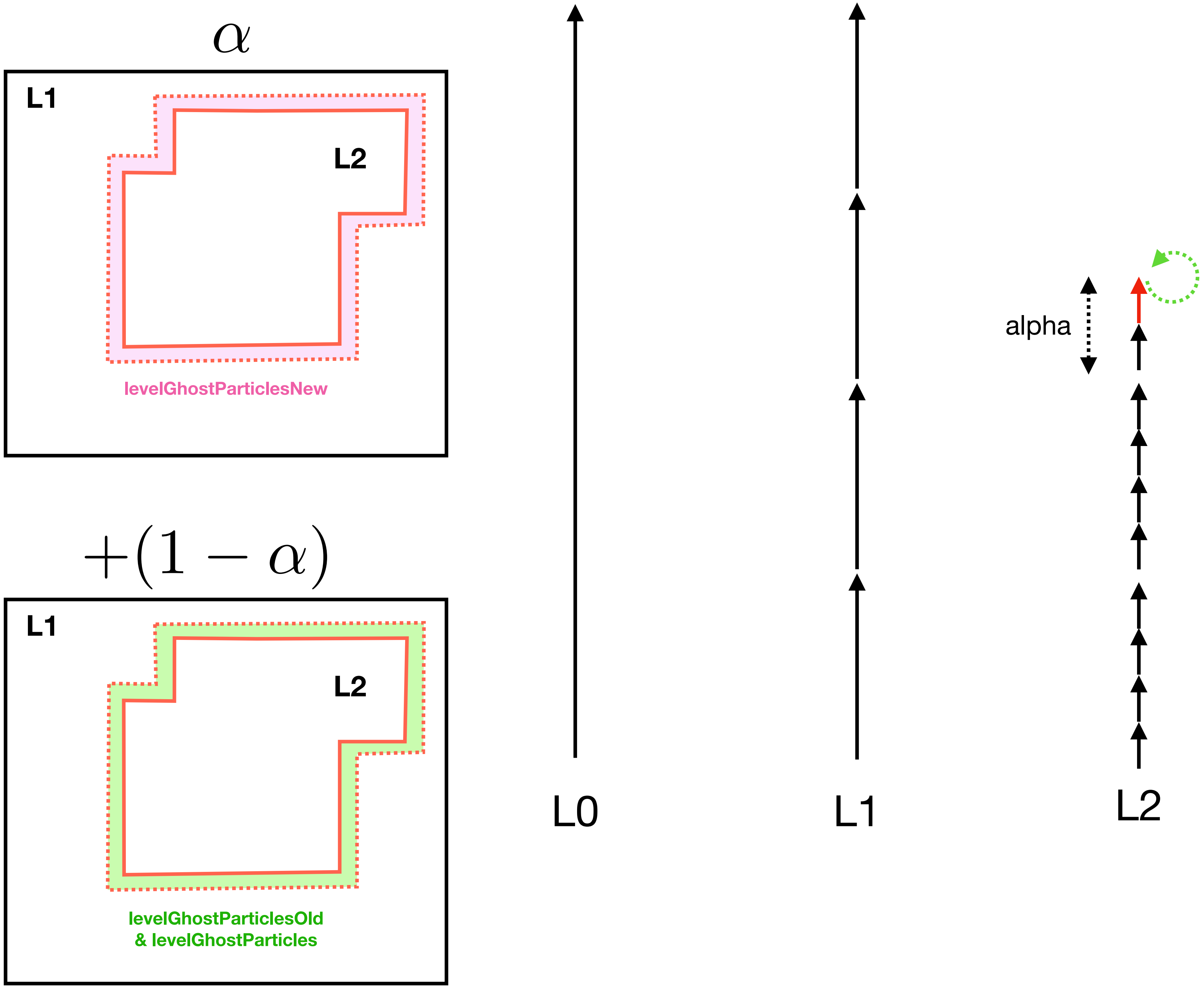
This is called by the solver after advancing particles



`fillGhostParticles` exchanges with neighbors. levelGhostParticles are already filled (old and new)

Fill Ion Moment Ghosts

This is filling the ghost nodes for ion moments.
There are two kinds : patch ghost nodes and level ghost nodes.
For level ghost nodes, this is different from filling EM ghost because here we are not making the time interpolation of the moments defined on the coarser level. We’re using the levelGhostParticles (new and old).

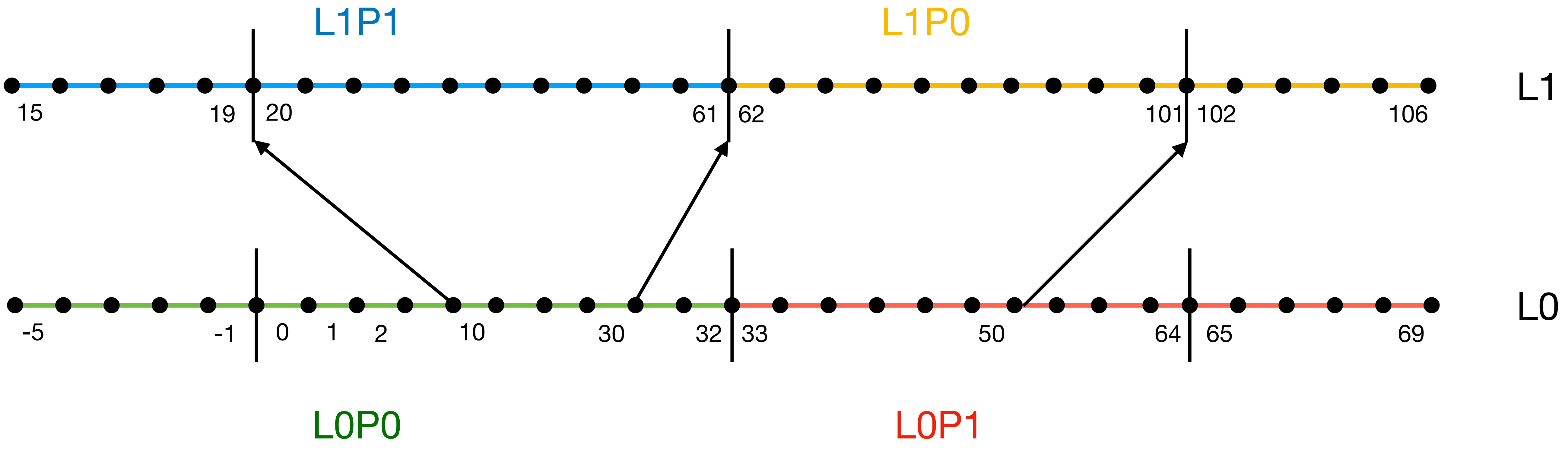


`fillIonMomentGhosts` takes :

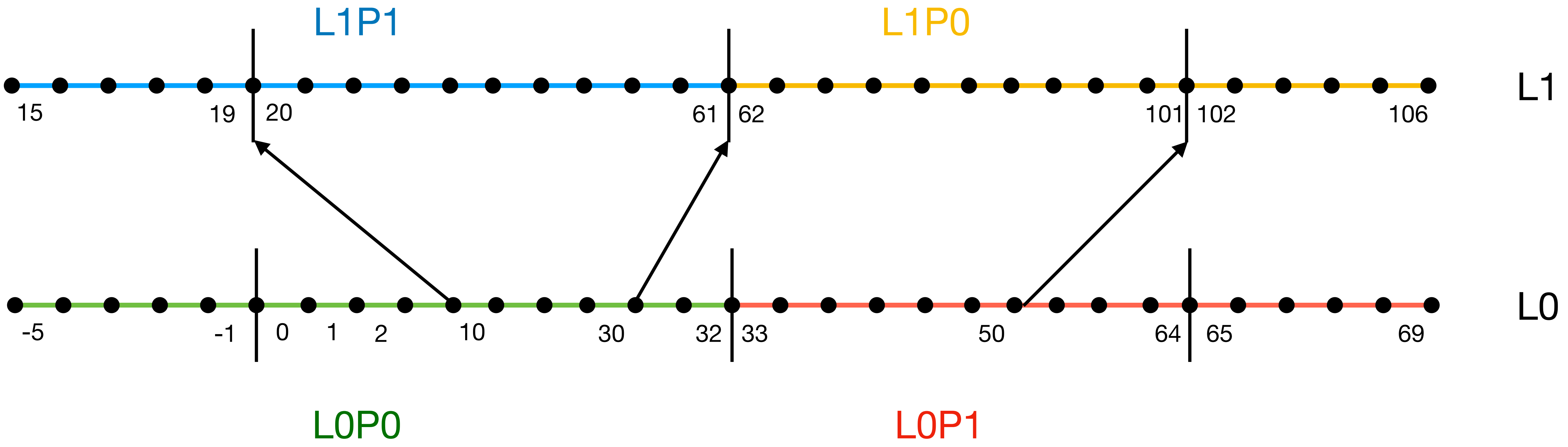
- patchGhostParticles and compute their moments on the grid
- levelGhostParticlesOld and levelGhostParticlesNew and compute their moments with a coef alpha and 1-alpha, alpha being the fraction of the total subcycling time period already performed.

GhostFilling()

Example


L1P1
L1P0
L1
15
19
20
61
62
101
102
106
L0
-5
-1
0
1
2
10
30
32
33
50
64
65
69
L0P0
L0P1

Example

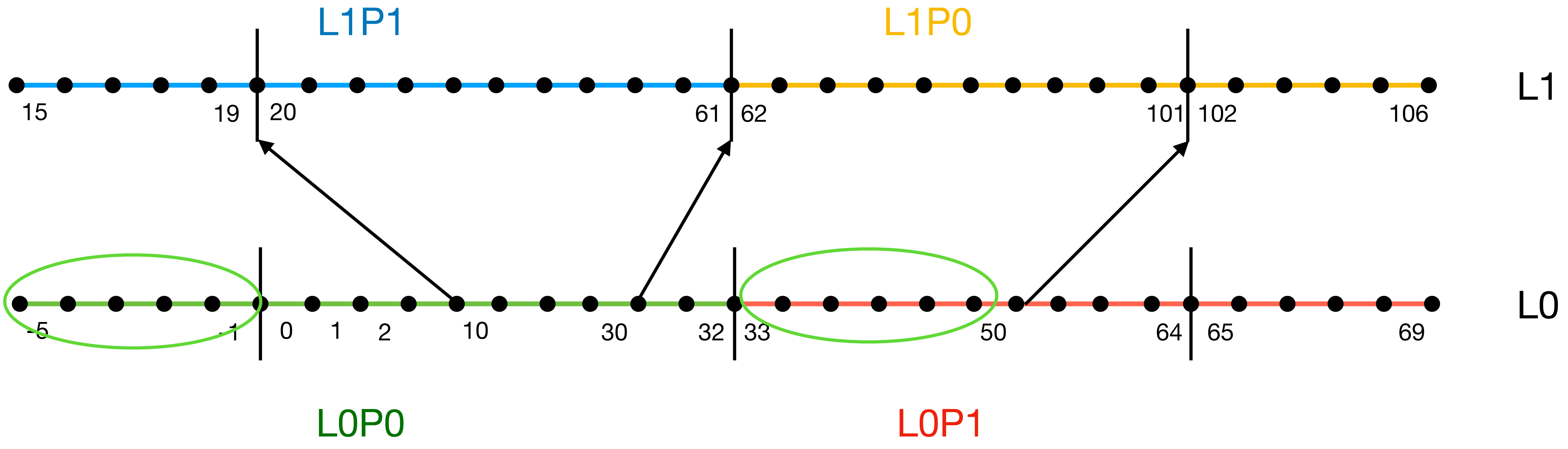


Level0 : only calls copy(), example for Bx :
copy : L0P0 (0,32) has source (33,64) with overlap (33,38) and transformation offset 0
copy : L0P0 (0,32) has source (33,64) with overlap (-5,0) and transformation offset -65
copy : L0P1 (33,64) has source (0,32) with overlap (28,33) and transformation offset 0
copy : L0P1 (33,64) has source (0,32) with overlap (65,70) and transformation offset 65

Level1 : calls copy and refine, example for Bx
copy : L1P1 (20,61) has source (62,101) with overlap (62,67)
copy : L1P0 (62,101) has source (20,61) with overlap (57,62)
refine : destination (20,61) L1P1, source (7,9) on L0P0, overlap (15,20)
refine : destination (62,101) L1P0, source (51,53) on L0P1, overlap (102,107)

First get L0P0 ghosts

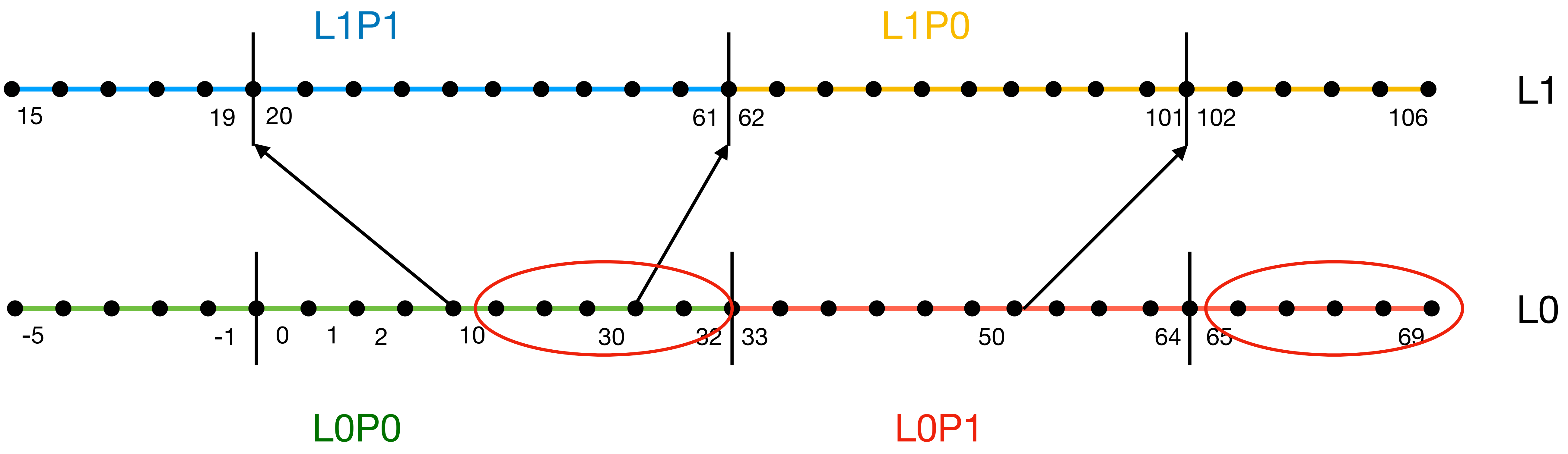
In serial, this involves two calls to FieldData::copy()



copy : L0P0 (0,32) has source (33,64) with overlap (33,38) and transformation offset 0
copy : L0P0 (0,32) has source (33,64) with overlap (-5,0) and transformation offset -65

then get L0P1 ghosts

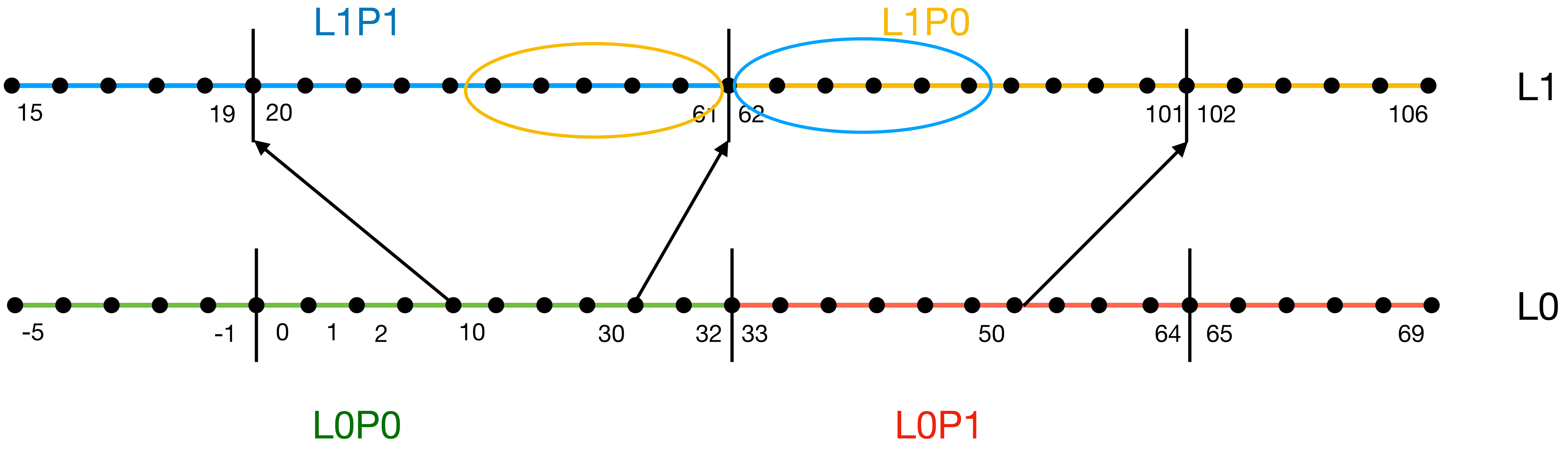
In serial, this involves two calls to FieldData::copy() per field



Level0 : only calls copy(), example for Bx :
copy : L0P1 (33,64) has source (0,32) with overlap (28,33) and transformation offset 0
copy : L0P1 (33,64) has source (0,32) with overlap (65,70) and transformation offset 65

then get L1P1 ghosts

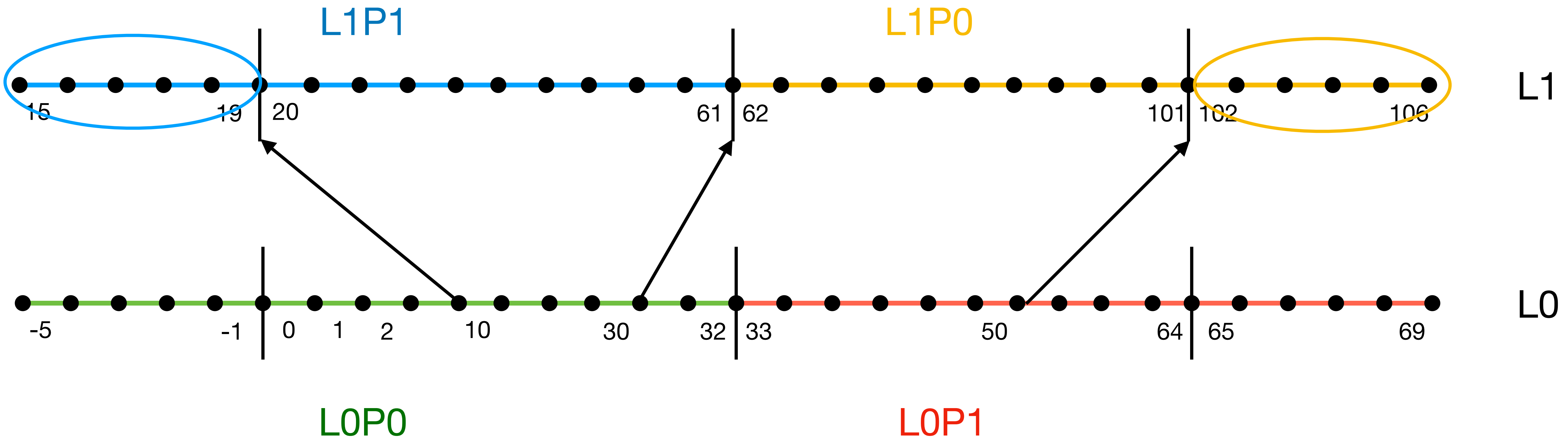
First get Patch ghosts
In serial, this involves two calls to FieldData::copy() per field for the patch ghosts



copy : L1P1 (20,61) has source (62,101) with overlap (62,67)
copy : L1P0 (62,101) has source (20,61) with overlap (57,62)

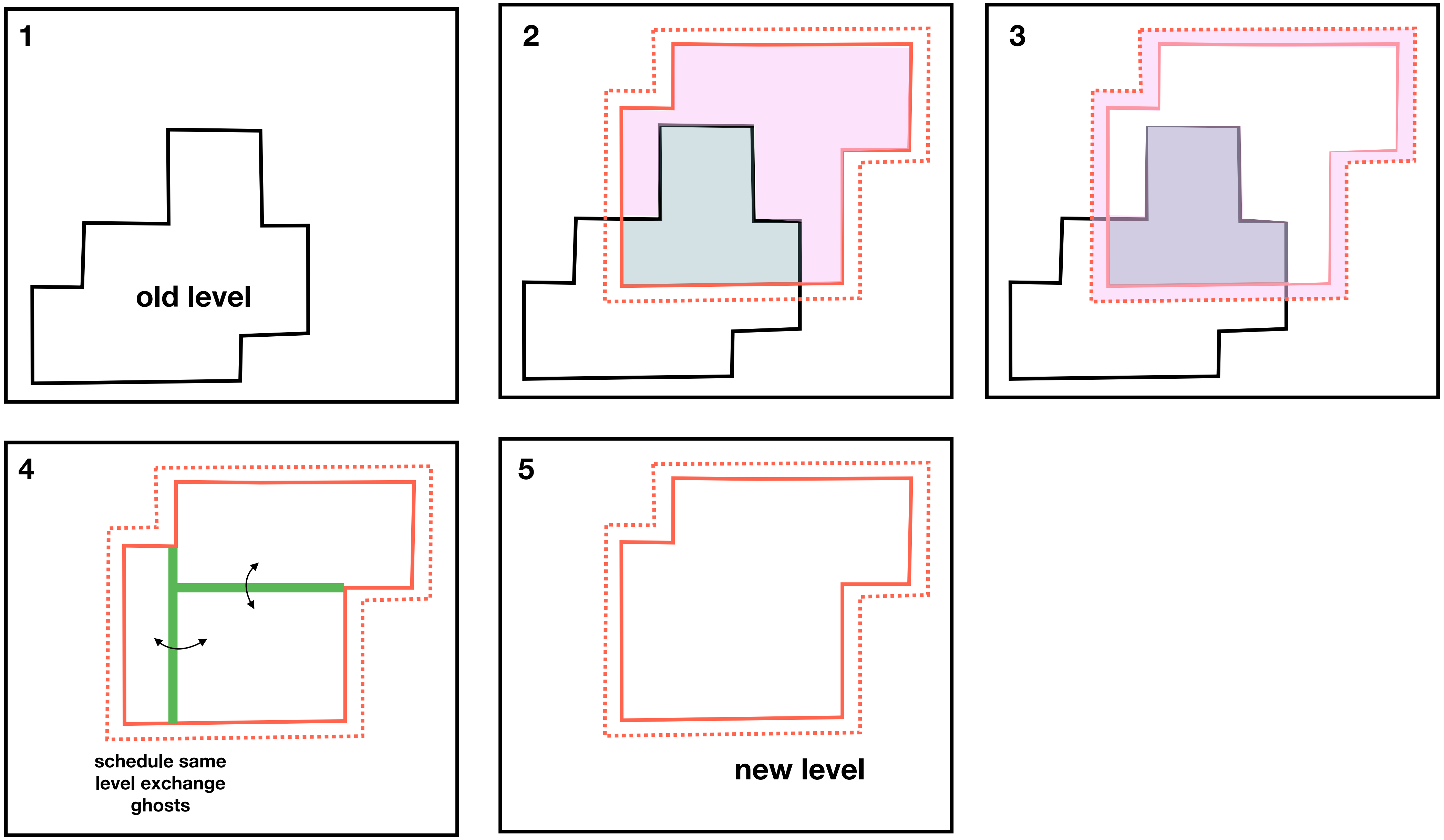
then get L1P1 ghosts

Then level ghosts
In serial, this involves two calls to FieldLinearTimeInterpolate::timeInterpolate() and two calls to FieldRefineOperator::refine() per component



refine : destination (20,61) L1P1, source (7,9) on L0P0, overlap (15,20)
refine : destination (62,101) L1P0, source (51,53) on L0P1, overlap (102,107)

Regridding



1

old level

2

refine op.<InteriorParticleRefineOp>
w/ PatchLevelInteriorFillPattern

3

refine op.<coarseBoundaryOld>
w/ border fill pattern

4

schedule same
level exchange
ghosts

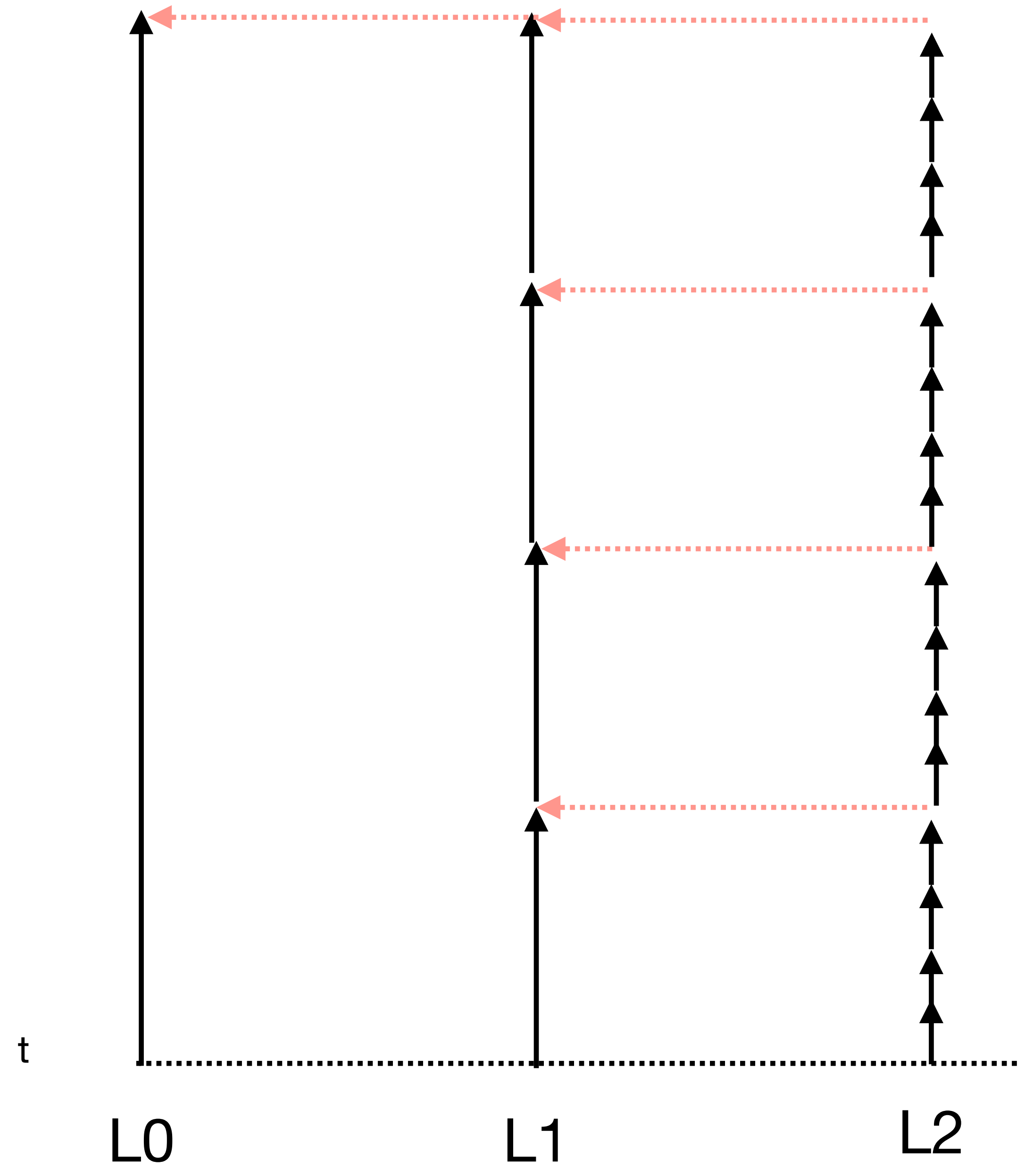
5

new level

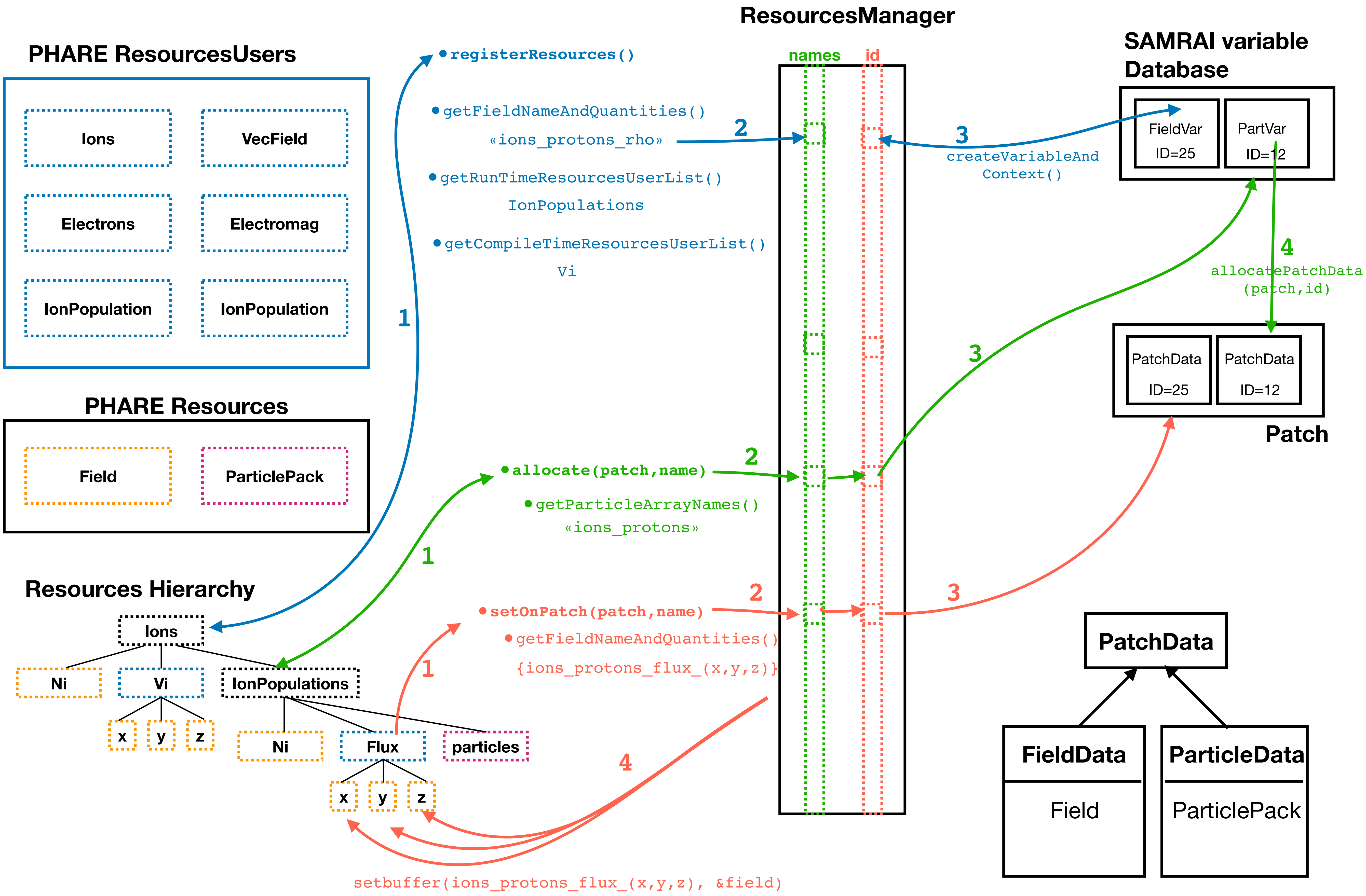
Level Synchronization

Synchronization

We want to synchronize: B, E, Moments from the refined levels onto the next coarser one



ResourcesManager Cheat Sheet



PHARE ResourcesUsers

Ions

VecField

Electrons

Electromag

IonPopulation

IonPopulation

PHARE Resources

Field

ParticlePack

Resources Hierarchy

Ions

Ni

Vi

x

y

z

IonPopulations

Ni

Flux

x

y

z

particles

ResourcesManager

names

id

SAMRAI variable Database

FieldVar ID=25

PartVar ID=12

Patch

PatchData ID=25

PatchData ID=12

- registerResources()
- getFieldNameAndQuantities()

  «ions_protons_rho»
- getRunTimeResourcesUserList()

  IonPopulations
- getCompileTimeResourcesUserList()

  Vi

3 createVariableAndContext()

4 allocatePatchData(patch,id)

- allocate(patch,name)
- getParticleArrayNames()

  «ions_protons»

- setOnPatch(patch,name)
- getFieldNameAndQuantities()

  {ions_protons_flux_(x,y,z)}

4 setbuffer(ions_protons_flux_(x,y,z), &field)

PatchData

FieldData

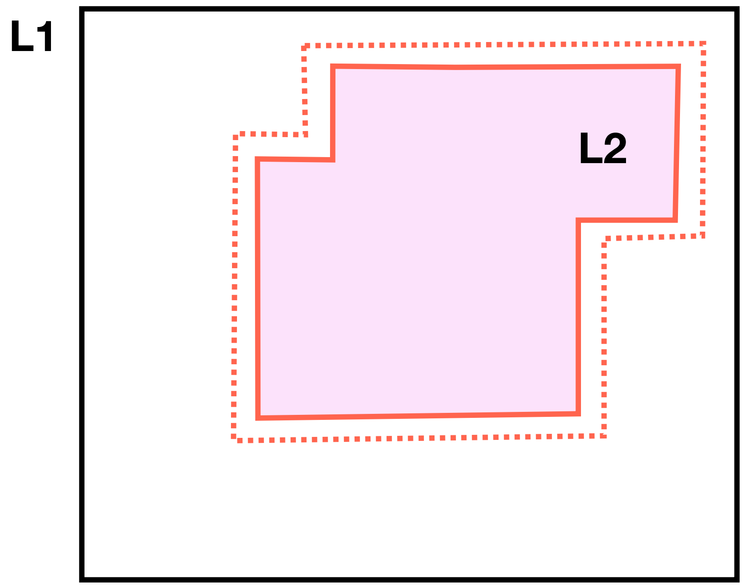
Field

ParticleData

ParticlePack

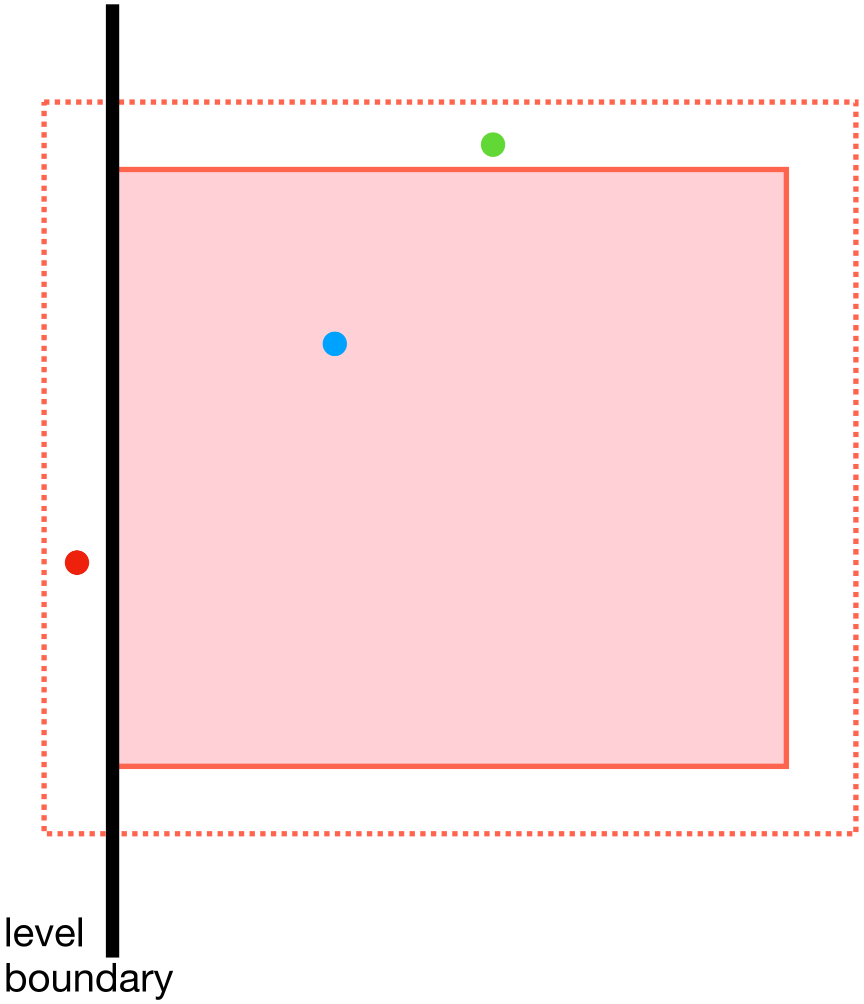
moveIons

The goal here is to move ions and accumulate density and flux on the mesh.



movelons

There are three kinds of particles: interior, patchGhostParticles, levelGhostParticles. Each of these need to be pushed



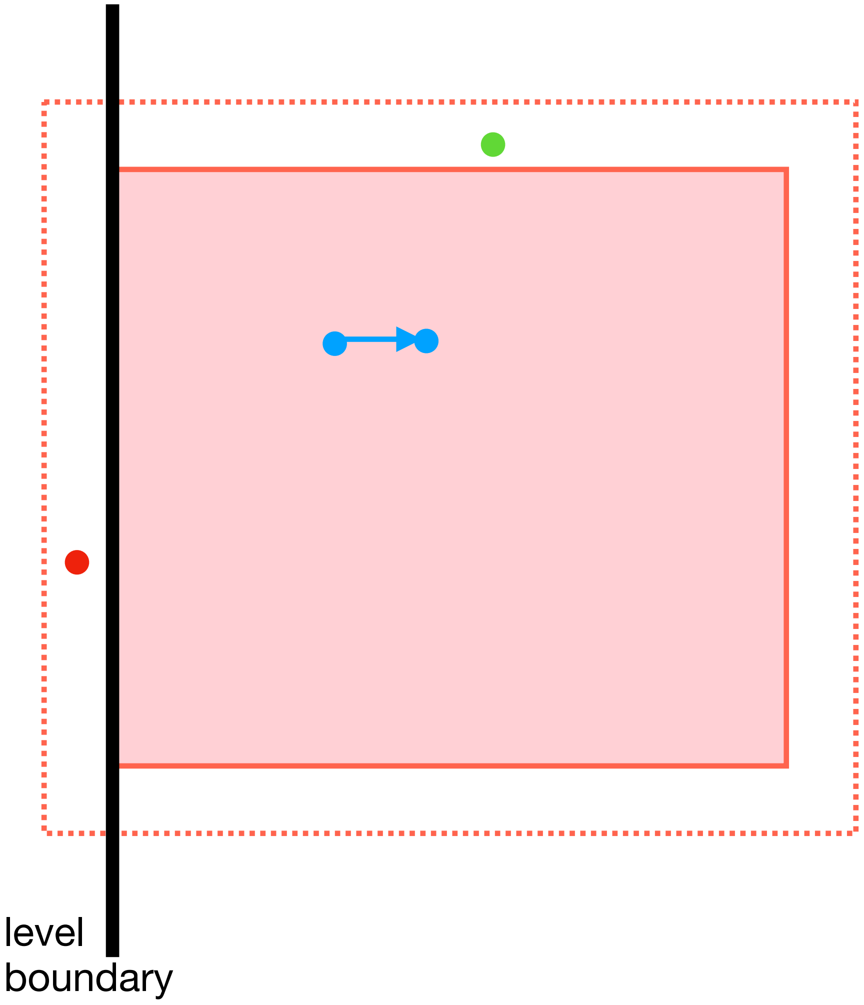
movelons

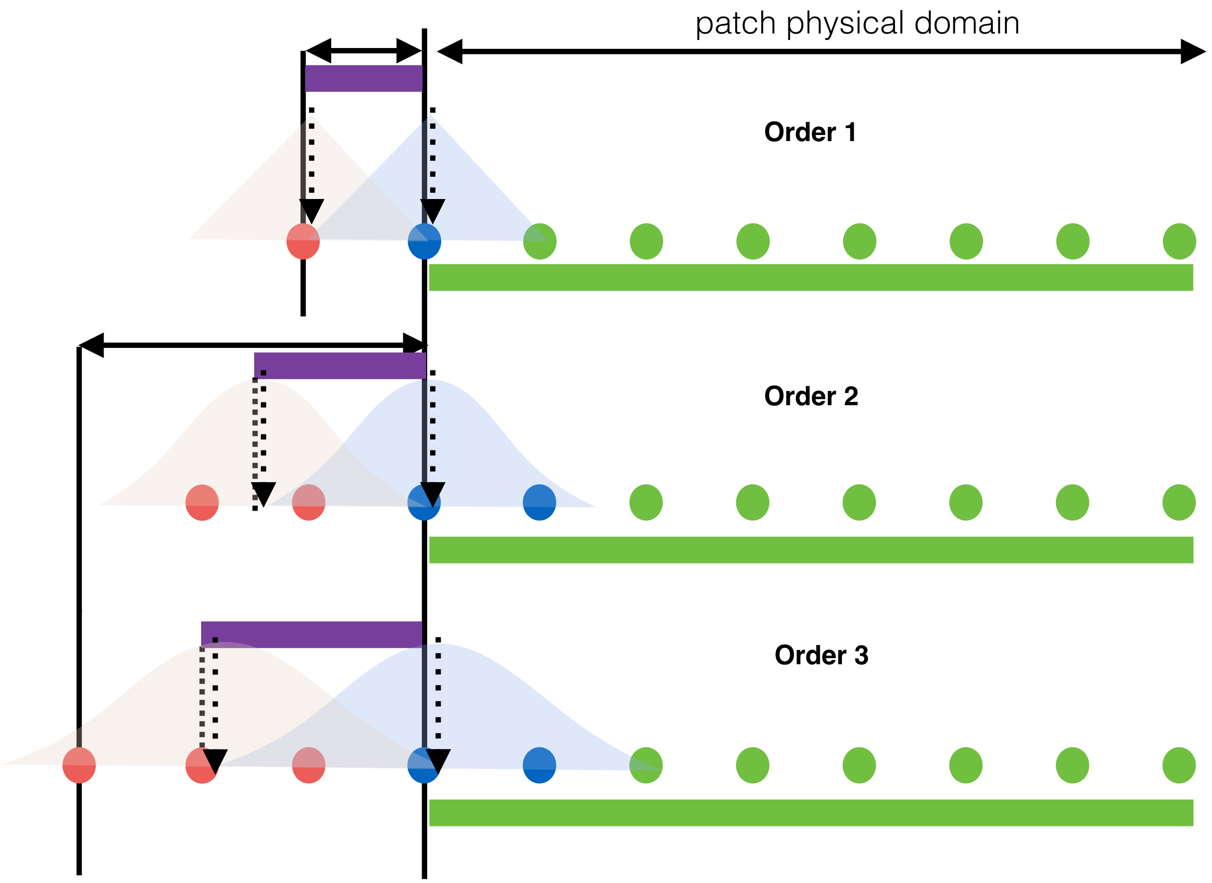
There are three kinds of particles: interior, patchGhostParticles, levelGhostParticles. Each of these need to be pushed

1- push interior particles and accumulate their density & flux.

interior particles can deposit on green, blue and some of the red nodes blow.

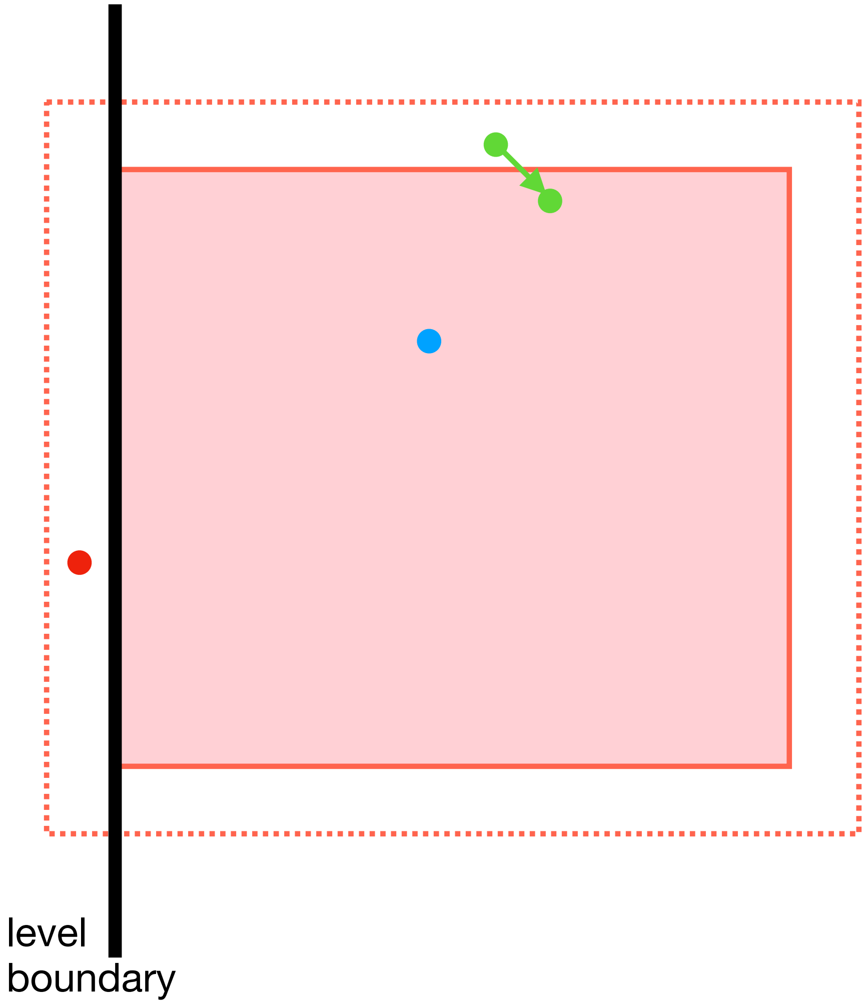
At this point, blue and red nodes are still incomplete





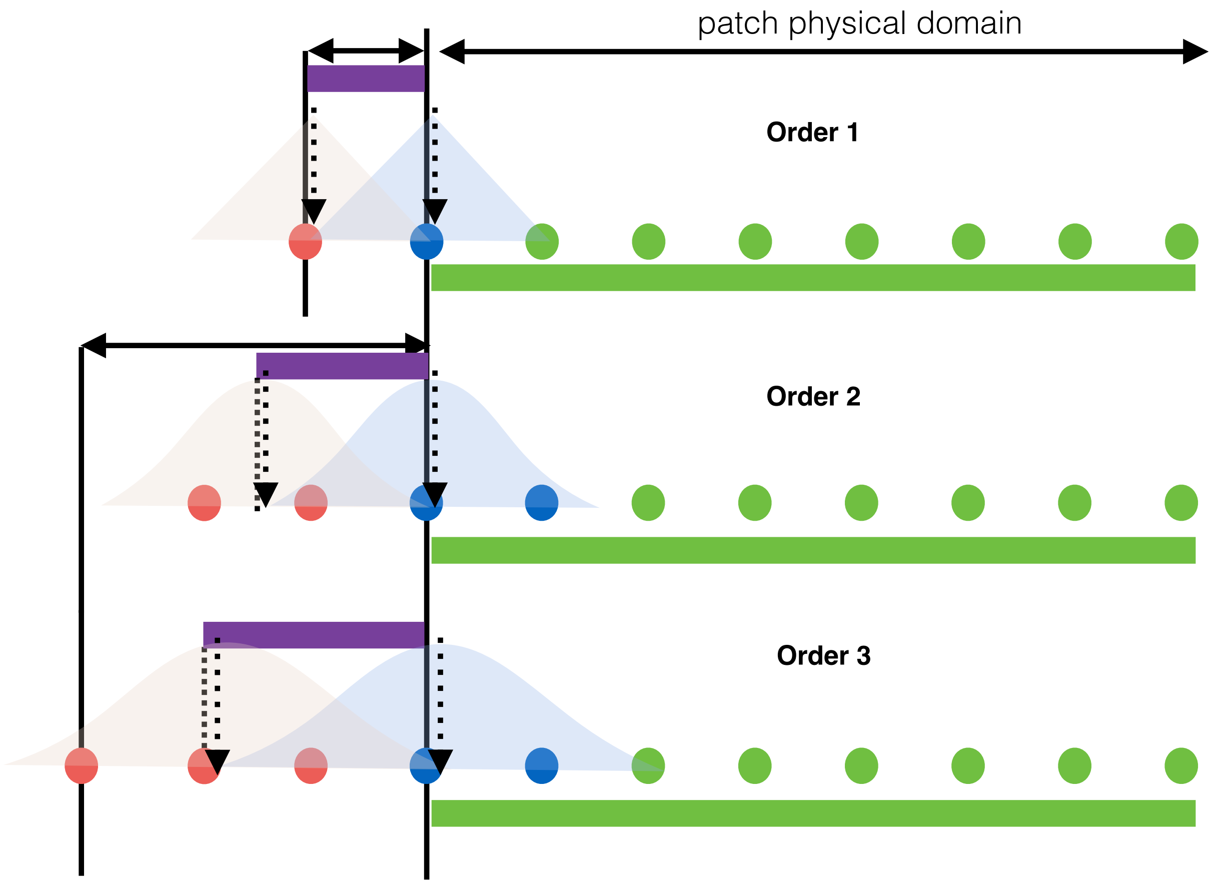
movelons

There are three kinds of particles: interior, patchGhostParticles, levelGhostParticles. Each of these need to be pushed



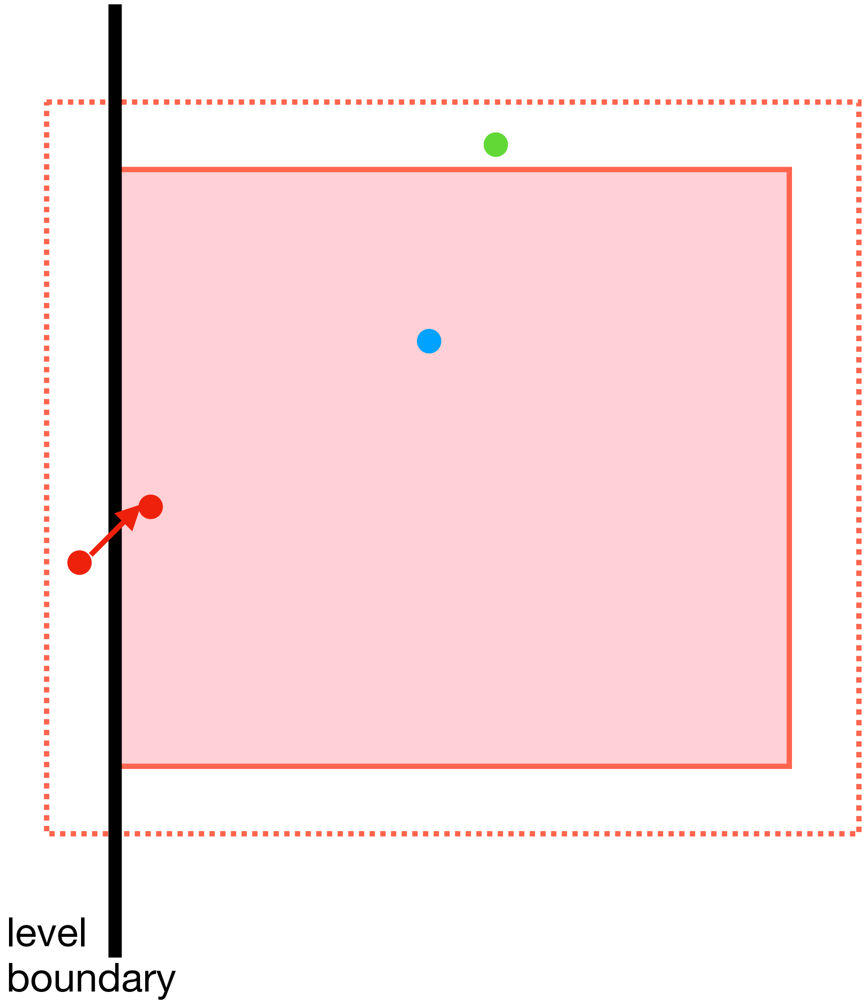
2- push patchGhostParticles and accumulate their density & flux. Only accumulate the ghost that have entered the domain. These project onto green and blue, like interior particles.

at this point, blue and red nodes are still incomplete on patch and level borders



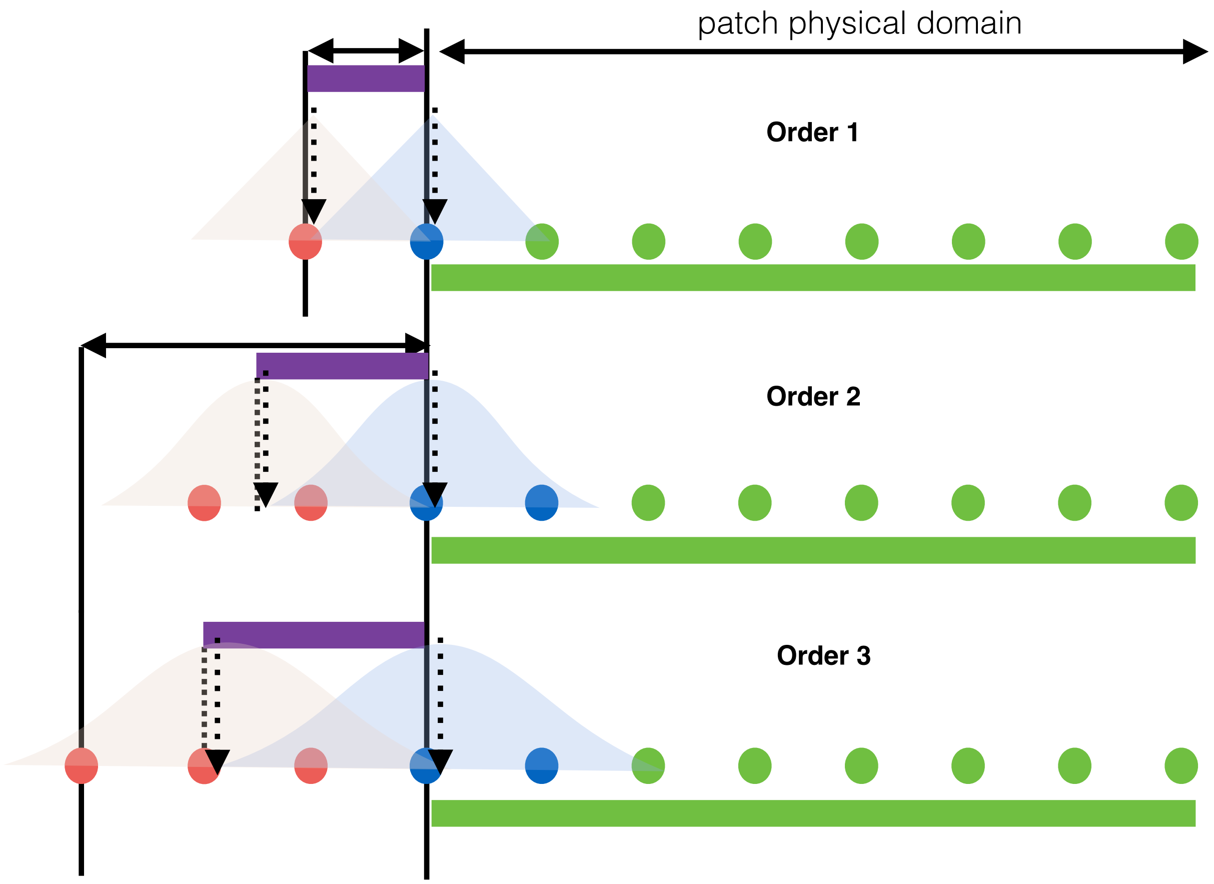
movelons

There are three kinds of particles: interior, patchGhostParticles, levelGhostParticles. Each of these need to be pushed



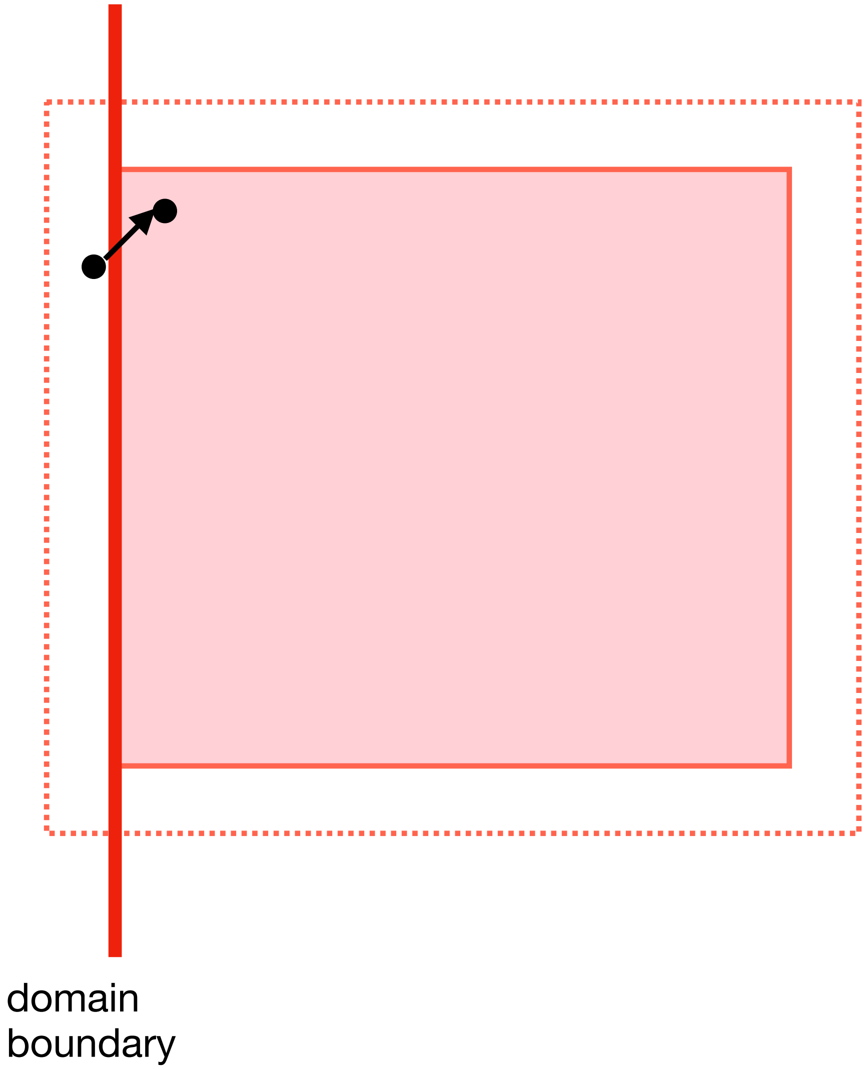
3- push levelGhostParticles and accumulate their density & flux. Only accumulate the ghost that have entered the domain. These project onto green and blue, like interior particles.

at this point, blue and red nodes are still incomplete on patch and level borders



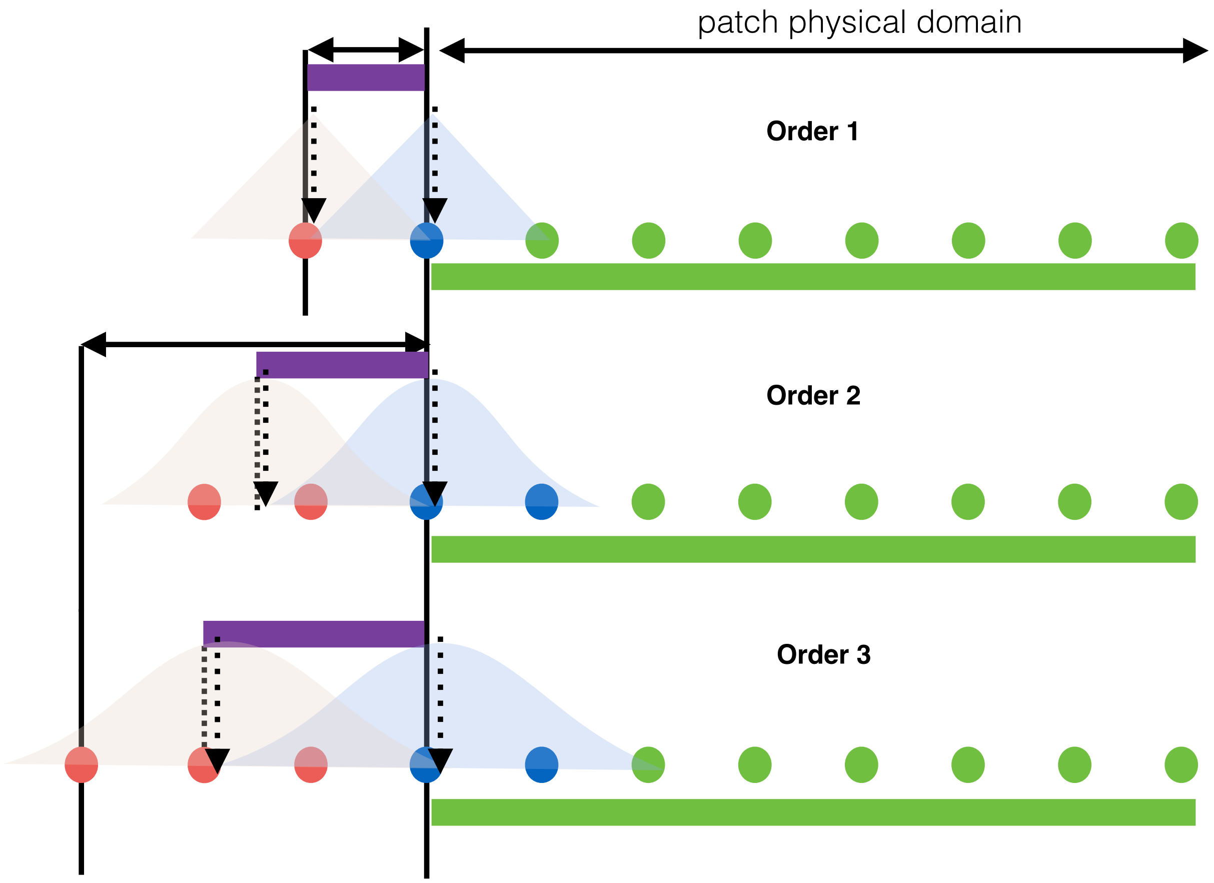
movelons

If the patch is touching the domain boundary, specific boundary conditions may inject particles. One need to inject them and accumulate their density and flux.



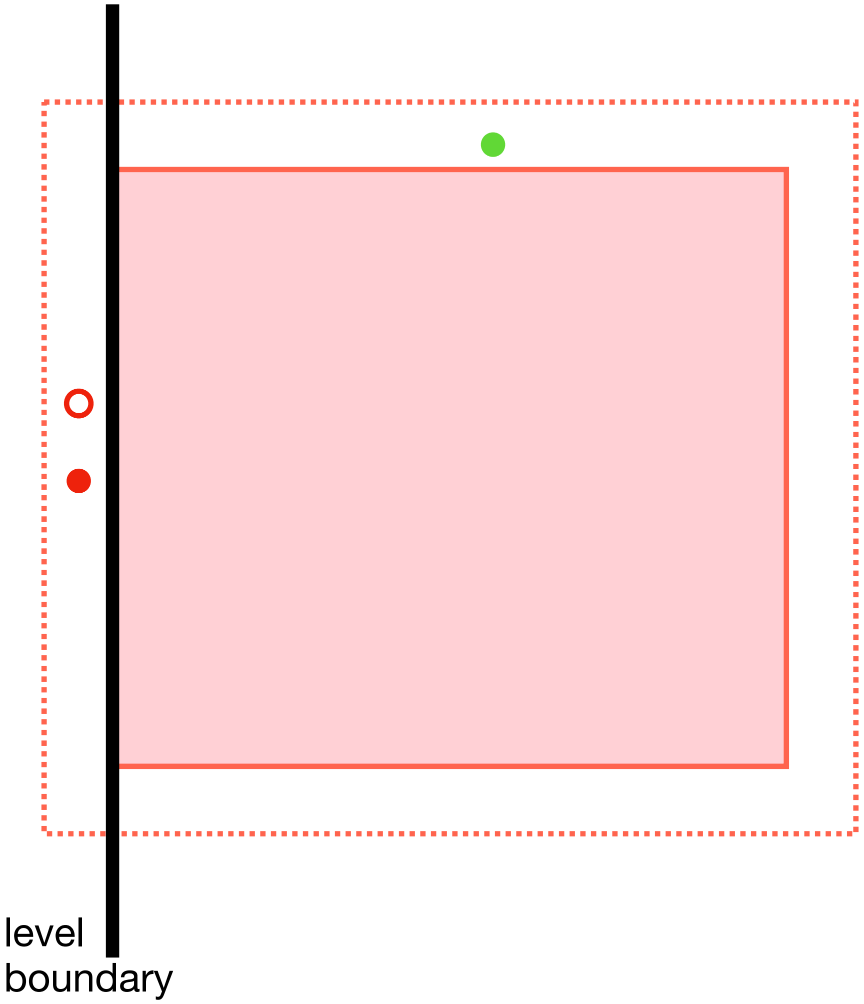
These project onto green and blue, like interior particles.

at this point, blue and red nodes are still incomplete on patch and level borders



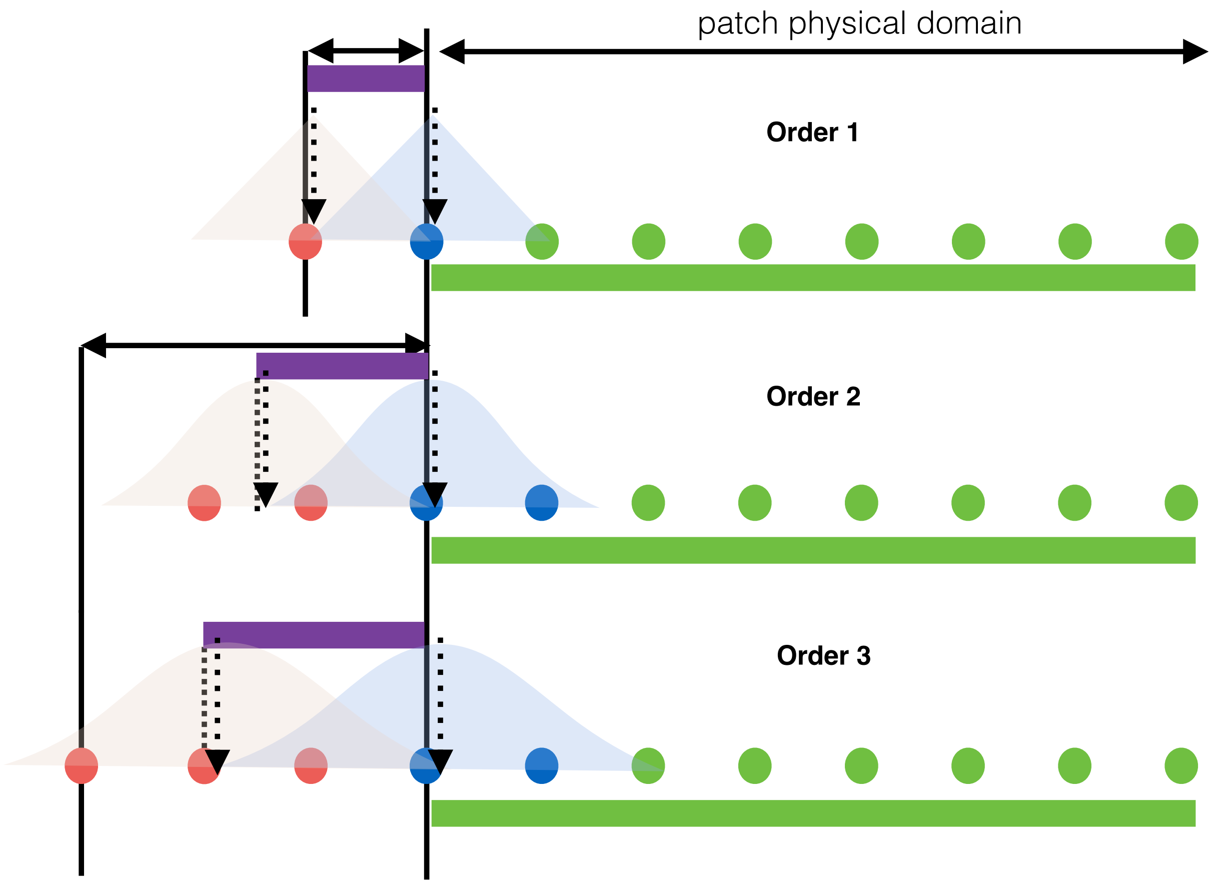
moveIons

now interior, patchGhostParticles, levelGhostParticles have been accumulated on to the mesh. Green nodes are complete. Blue and red nodes are not complete. We need to fill these nodes with ghost moments.



`fillIonMomentGhosts()` will :

- accumulate density and flux of `patchGhostParticles`. Those particles will put density on blue and needed red nodes.
- accumulate density and flux from `levelGhostParticlesOld` and `levelGhostParticlesNew`


patch physical domain
Order 1
Order 2
Order 3