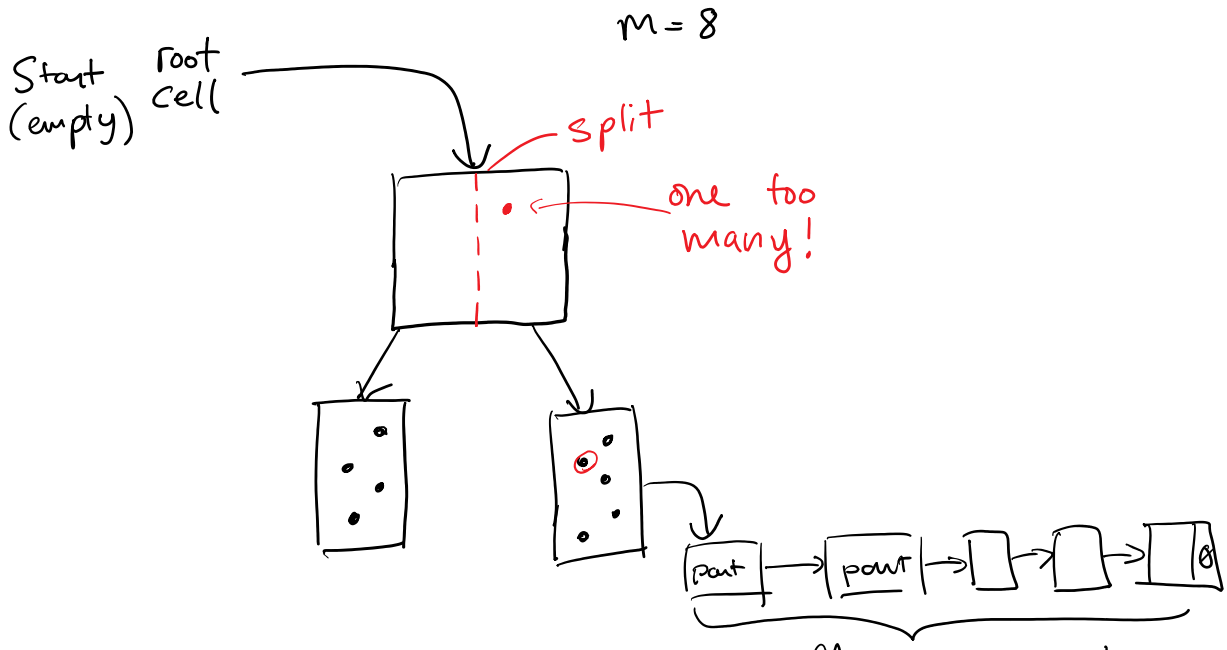
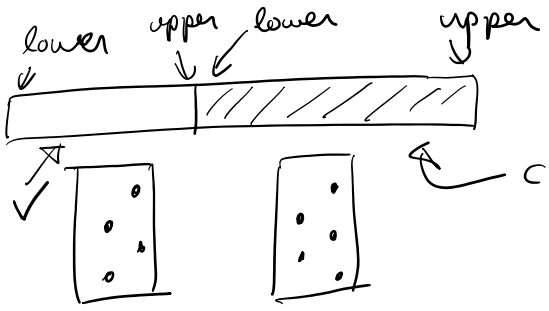
Another method to build a tree:



all are nearby in space (geometrically)

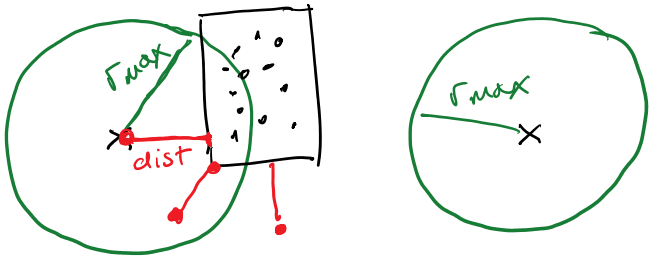
But not necessarily nearby in memory!

Partition function

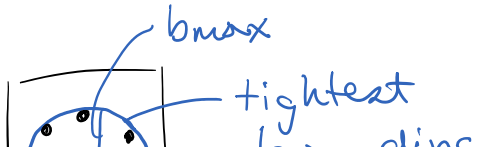


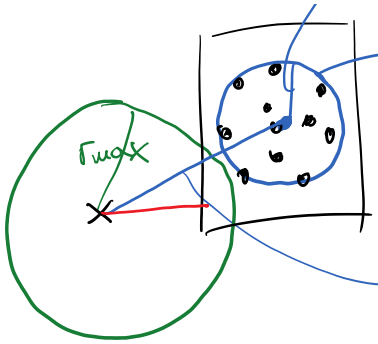
Geometrical locality ⟹ data locality

Nearest Neighbor Searching



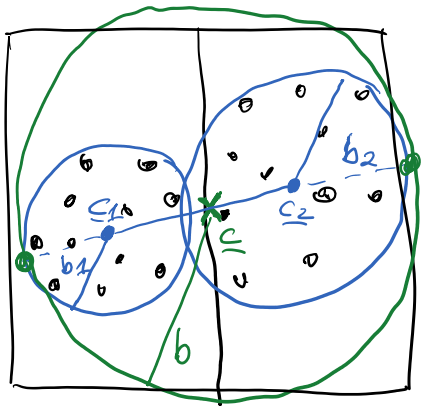
if dist2 < rmax2 then open the cell.
else ignore it.



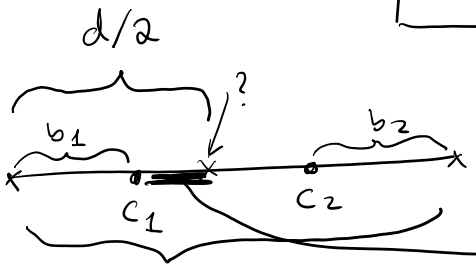


tightest bounding ball.

if $dist < (b_{max} + r_{max})$ then open the cell else ignore it.



You really only need the tightest bounding ball for leaf cells. Then you can get a good approximation to the tightest bounding ball of a parent cell using the 2 leaf cells!



$d = |\underline{c}_2 - \underline{c}_1| + b_1 + b_2$

$d/2 - b_1$

$$c = c_1 + \left(\frac{d}{2} - b_1\right)$$

But as a vector we need this to follow the unit vector along $\underline{c}_2 - \underline{c}_1$, so:

$$\underline{c} = \underline{c}_1 + \left(\frac{d}{2} - b_1\right)\frac{(\underline{c}_2 - \underline{c}_1)}{|\underline{c}_2 - \underline{c}_1|}$$

— unit vector

but $\boxed{\ell = |\underline{c}_2 - \underline{c}_1|}$

$$\underline{c} = \underline{c}_1 + \left(\frac{1}{2}\left(1 + \frac{b_1}{\ell} + \frac{b_2}{\ell}\right) - \frac{b_1}{\ell}\right)(\underline{c}_2 - \underline{c}_1)$$

$$\underline{c} = \underline{c}_1 + \left(\frac{1}{2} - \frac{1}{2}\frac{b_1}{\ell} + \frac{1}{2}\frac{b_2}{\ell}\right)(\underline{c}_2 - \underline{c}_1)$$

$$\boxed{x = \frac{b_2 - b_1}{\ell}}$$

$$\underline{c} = \left(1 - \frac{1}{2} + \frac{1}{2}x\right)\underline{c}_1 + \left(\frac{1}{2} + \frac{1}{2}x\right)\underline{c}_2$$

$$\boxed{\underline{c} = \frac{1}{2}\left((1+x)\,\underline{c}_1 + (1+x)\,\underline{c}_2\right)}$$

$$\boxed{b = d/2 = \frac{\ell}{2}(\ell + b_1 + b_2)}$$

This will be very close to the true bounding ball of the parent cell.

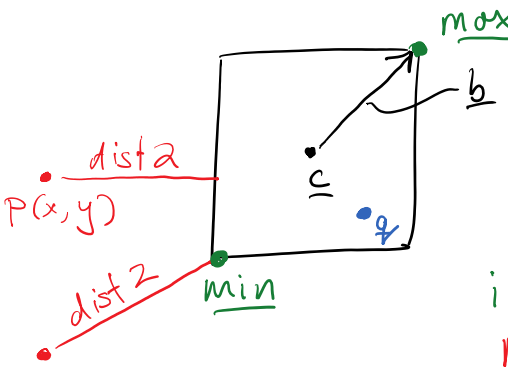
But, is the tightest bounding ball really better?

NO/YES

For leaf cells, probably yes, but for larger parent cells, (closer to the root of the tree), probably no. Due to the way we split particles along a cartesian dimension, we tend to get "boxy" distributions of particles higher up in the tree structure.

We will stick with using only bounding boxes for simplicity.

We need a Ball-Box intersection test:



$$\text{Min} \le q < \text{Max}$$

We can define the box via (max, min) or via (c, b).

Is there a difference in this choice? Yes. Due to roundoff errors in floating point calculations, the calculated min from (c, b) may not be the true min which bounds the particles.

However, we will choose to ignore this issue and continue using (c, b), but being aware of the potential problem is important!

We need to calculate the minimum dist2 between a point and the box.

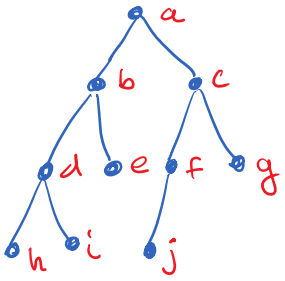
```
def boxdist(c, b, p):
    dist2 = 0
    for d in range(0,2):
        t = abs(c[d] - p[d]) - b[d]
```

```
for a ...
    t = abs(c[d] - p[d]) - b[d]
    if t>0:
        dist2 += t*t
return dist2
```

if dist2 < rmax² then ball intersects box!

Can you write boxdist(min, max, p) ?

Recursion was made for tree algorithms!




```
LNR(c)
    if (c != Null)
        LNR(c->left)
        print(c->data)
        LNR(c->right)
```

What does LNR(root) print out?

⇒ h d i b e a j f c g

what about:

```
RLN(c)
    if (c!=Null)
        RLN(c->right)
        RLN(c->left)
        print(c->data)
```

write out the output of RLN(root)!

```
def BALLWALK(c, Σ, rmax2):
    cnt = 0
    if c.isLeaf:
        for a in A[c.lower : c.upper+1]:
            if dist2(Σ, a.Σ) < rmax2:
                ++cnt
    else:
        if c.left.dist2(Σ) < rmax2:
            cnt += BALLWALK(c.left, Σ, rmax2)
        if c.right.dist2(Σ) < rmax2:
            cnt += BALLWALK(c.right, Σ, rmax2)
    return cnt
```

careful, both left and right subcells must exist! Make sure when you build the tree!