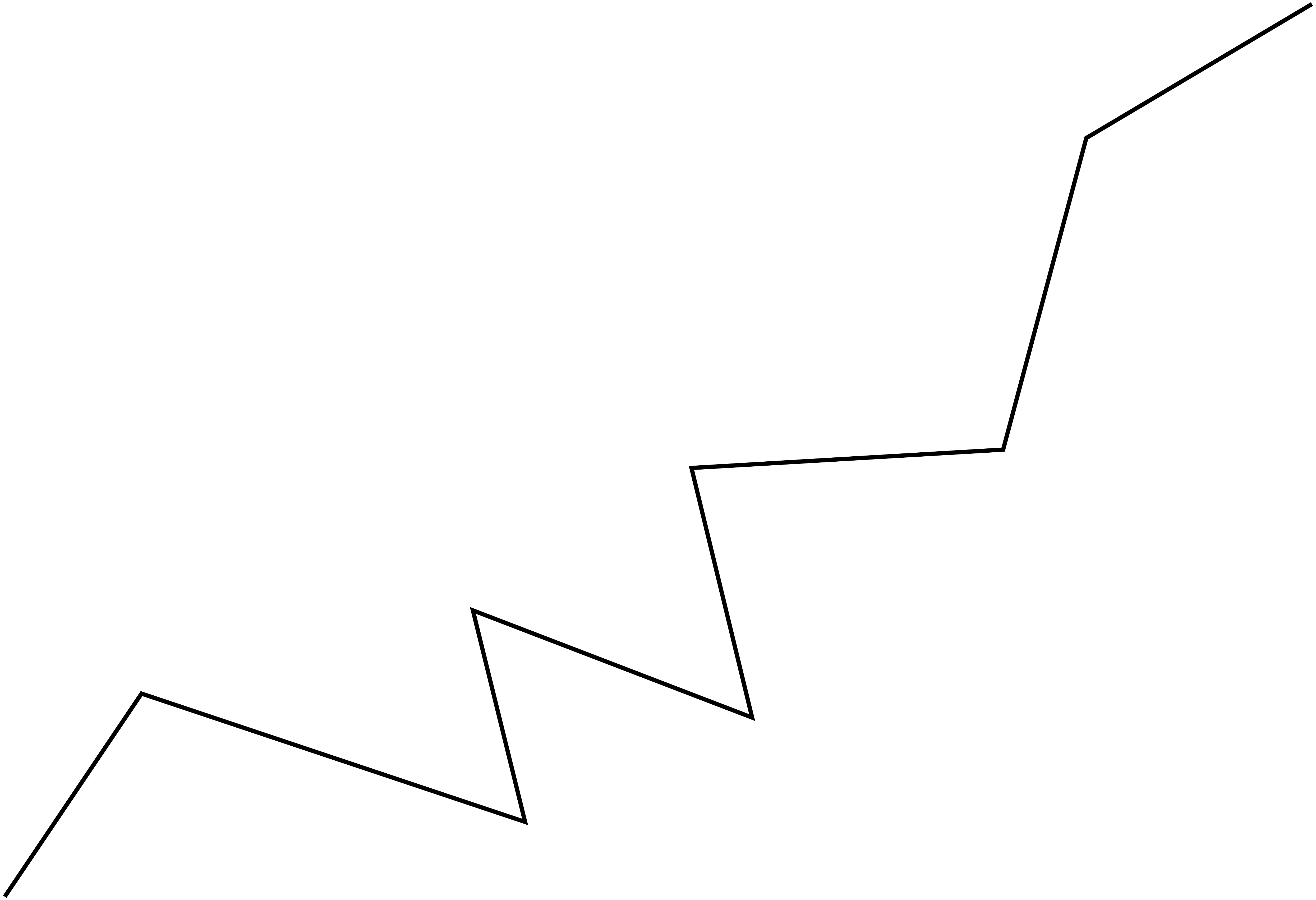
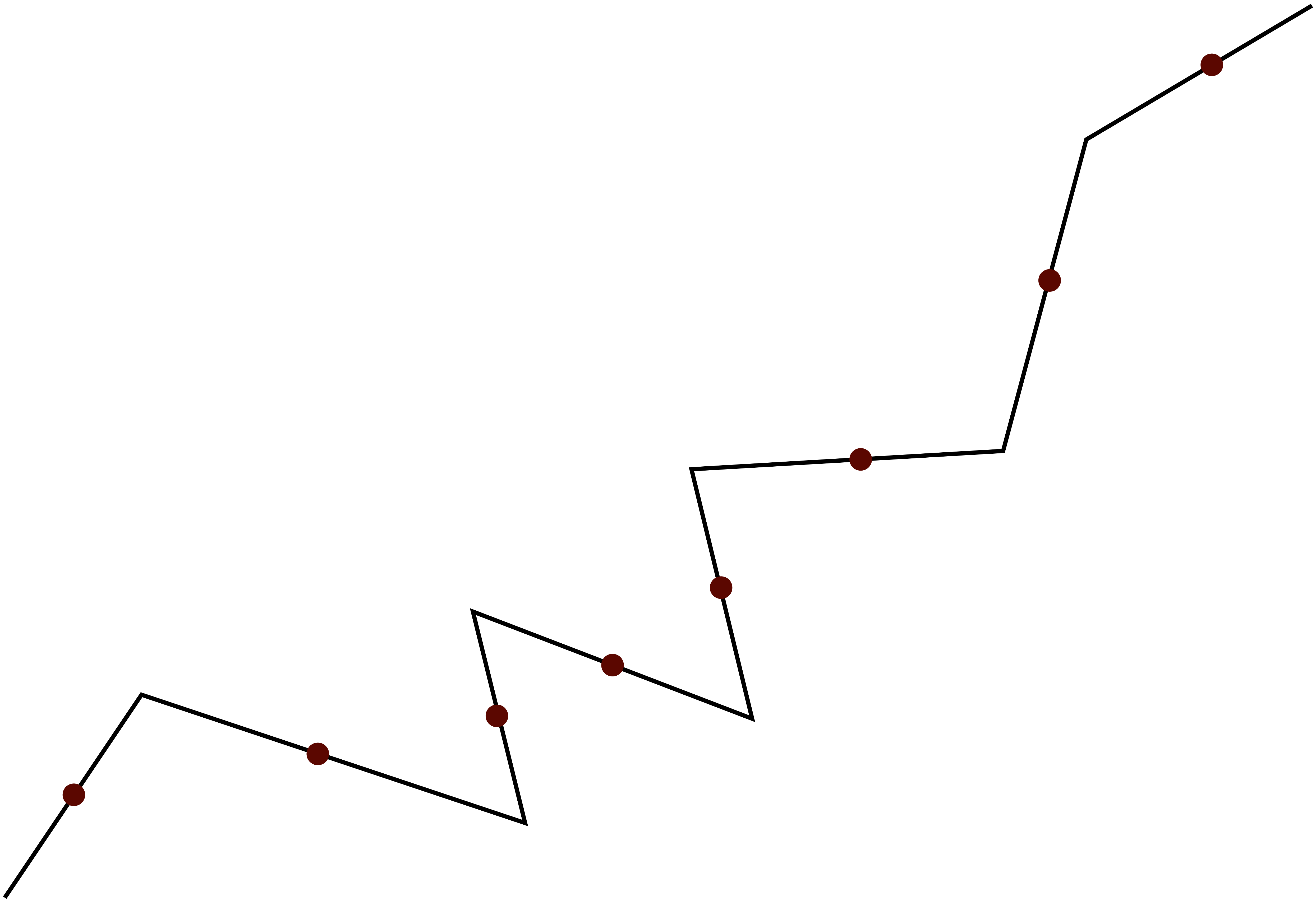
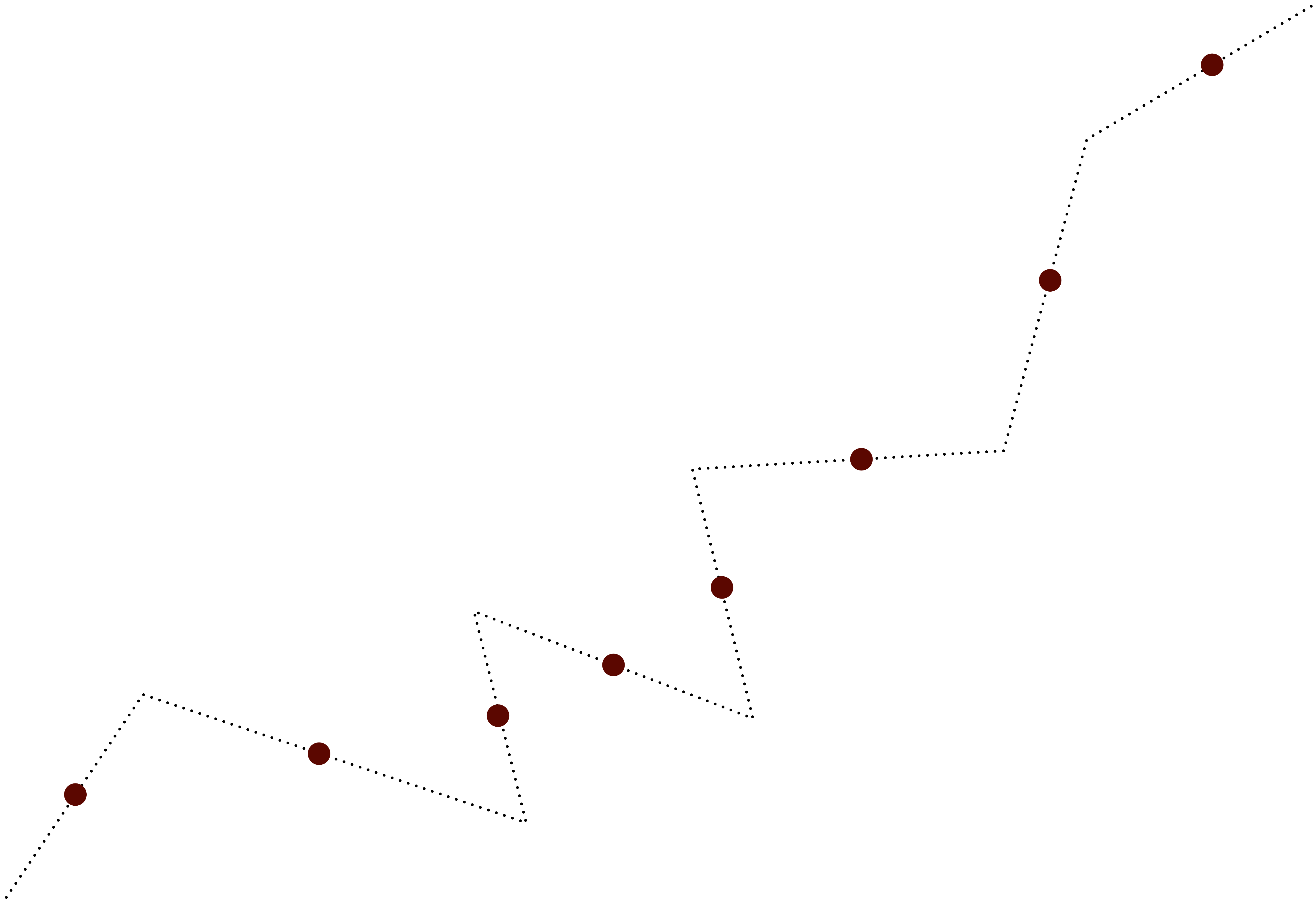
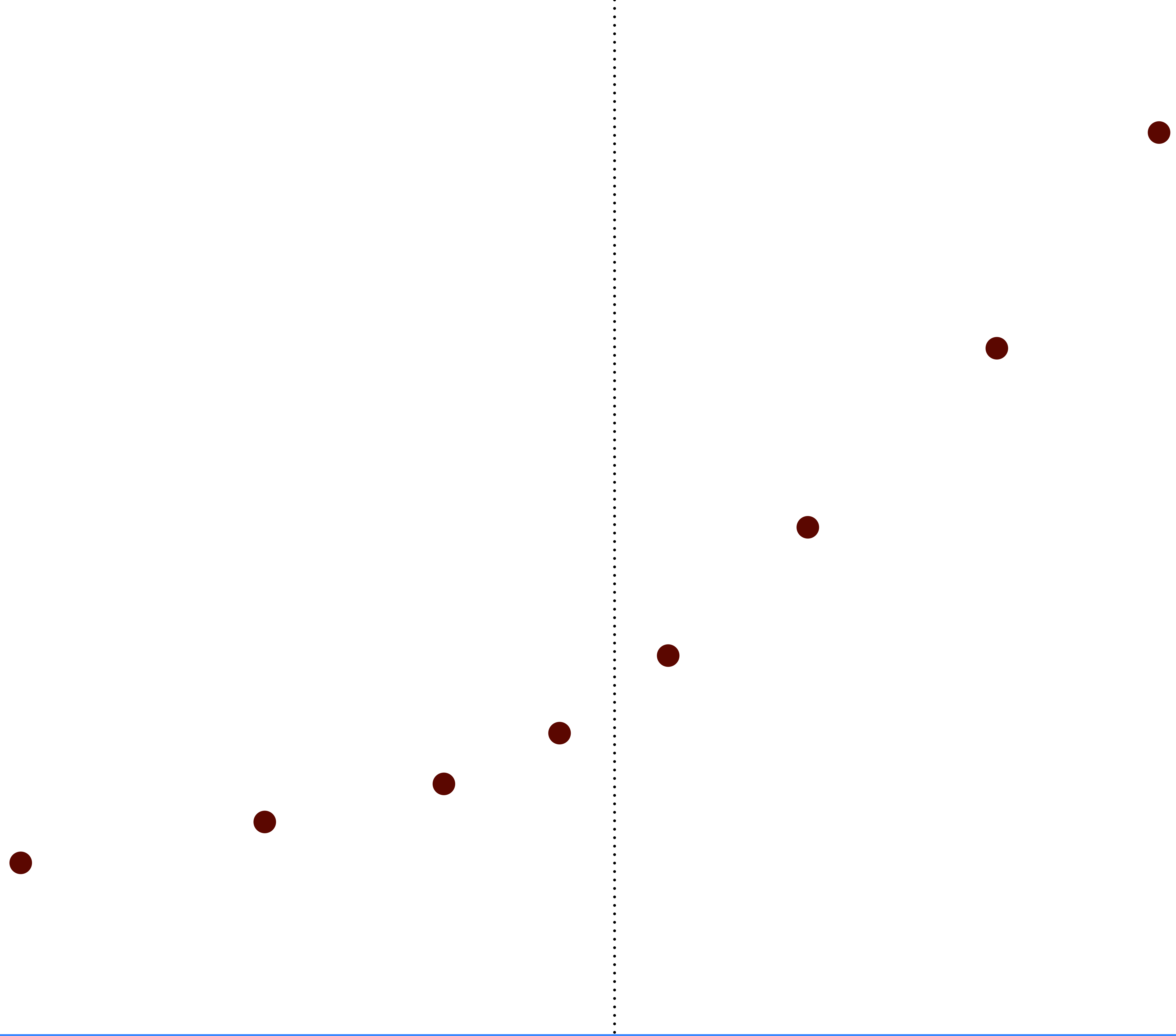
Collision detection (for simulated objects)

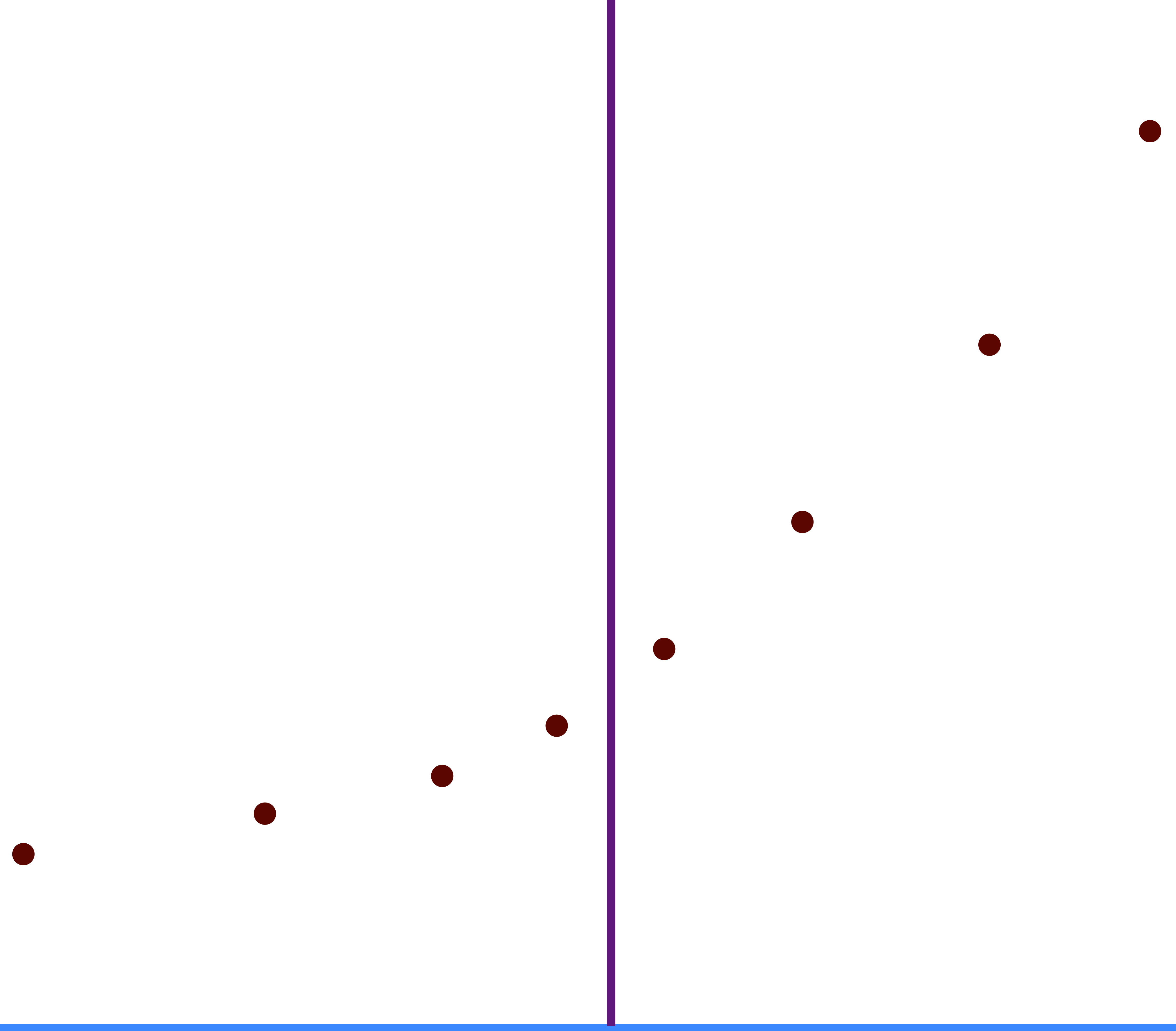
- Axis-aligned bounding box (AABB) query structure
  - Constructs a B-tree (or higher branching-order tree, depending on construction) with individual collision primitives at the leaves
  - Bounding boxes are aggregated, as we go from the leaves to the root of the tree
  - Prunes distant primitives when checking for collisions
    - Ideally, every tree level has O(1) potentially colliding nodes
    - Ideal cost of checking for collisions : O(logN) per query
    - O(N*logN) to check collisions between N primitives
  - Popular construction method : using k-d trees

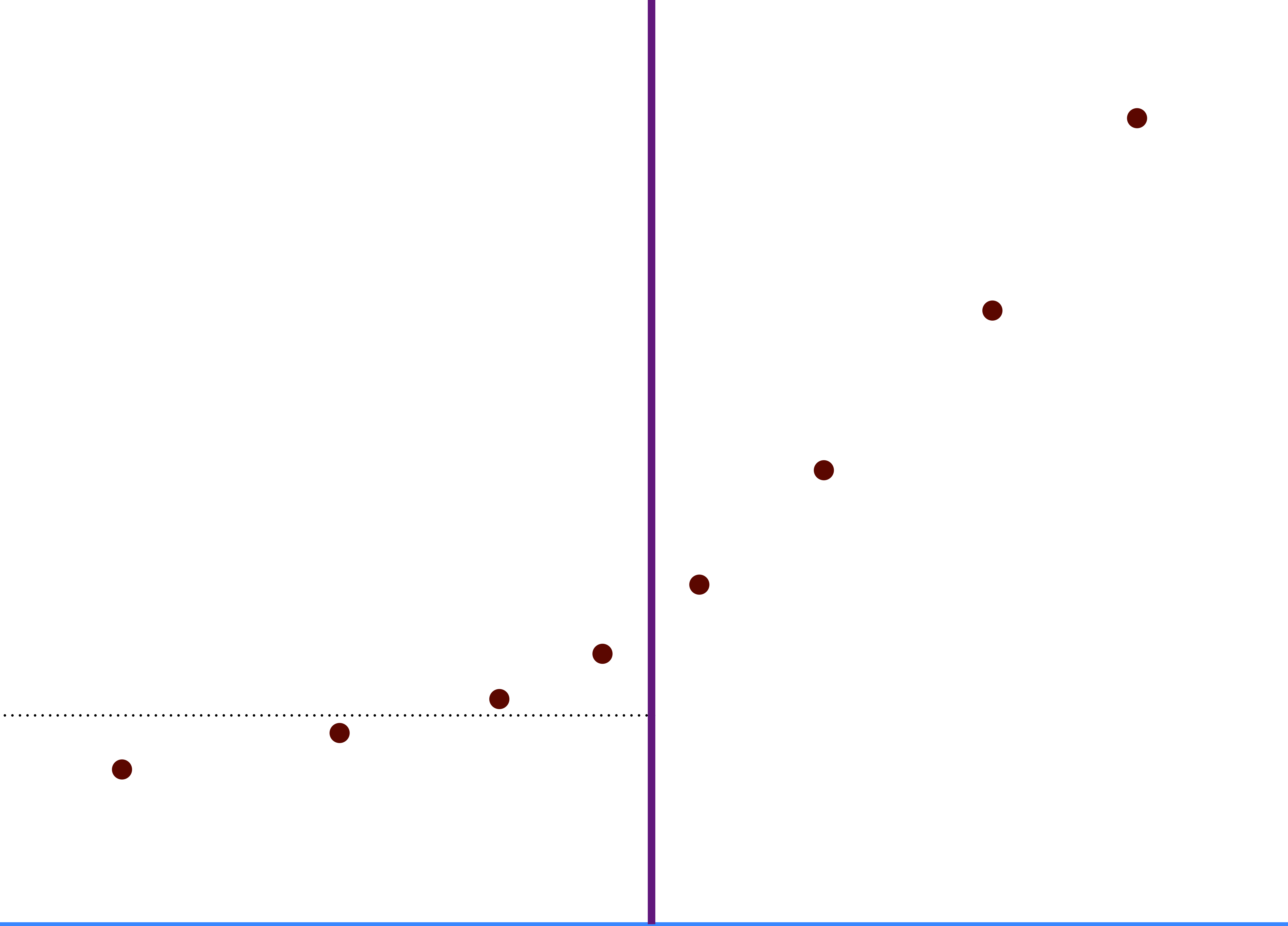


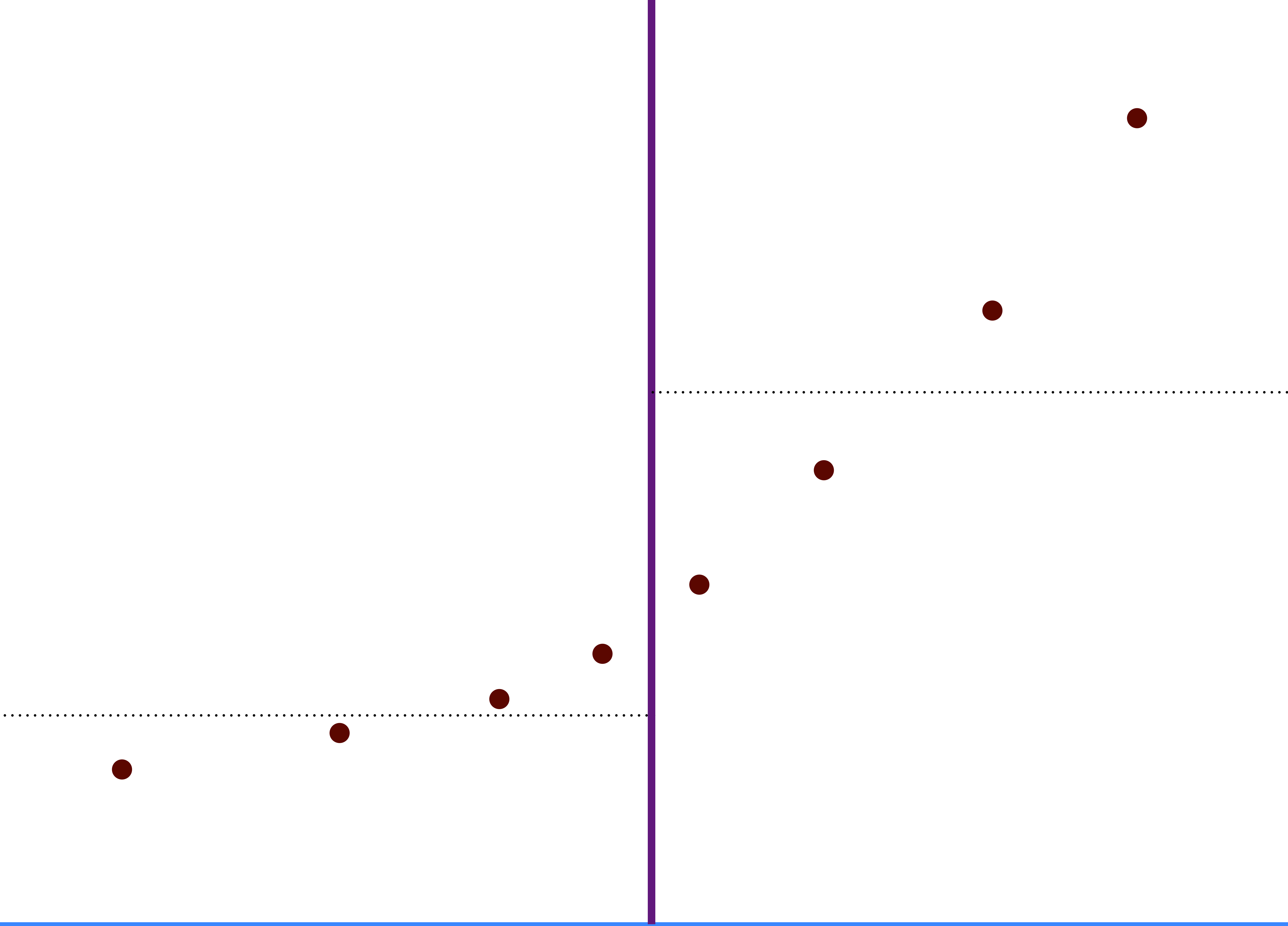


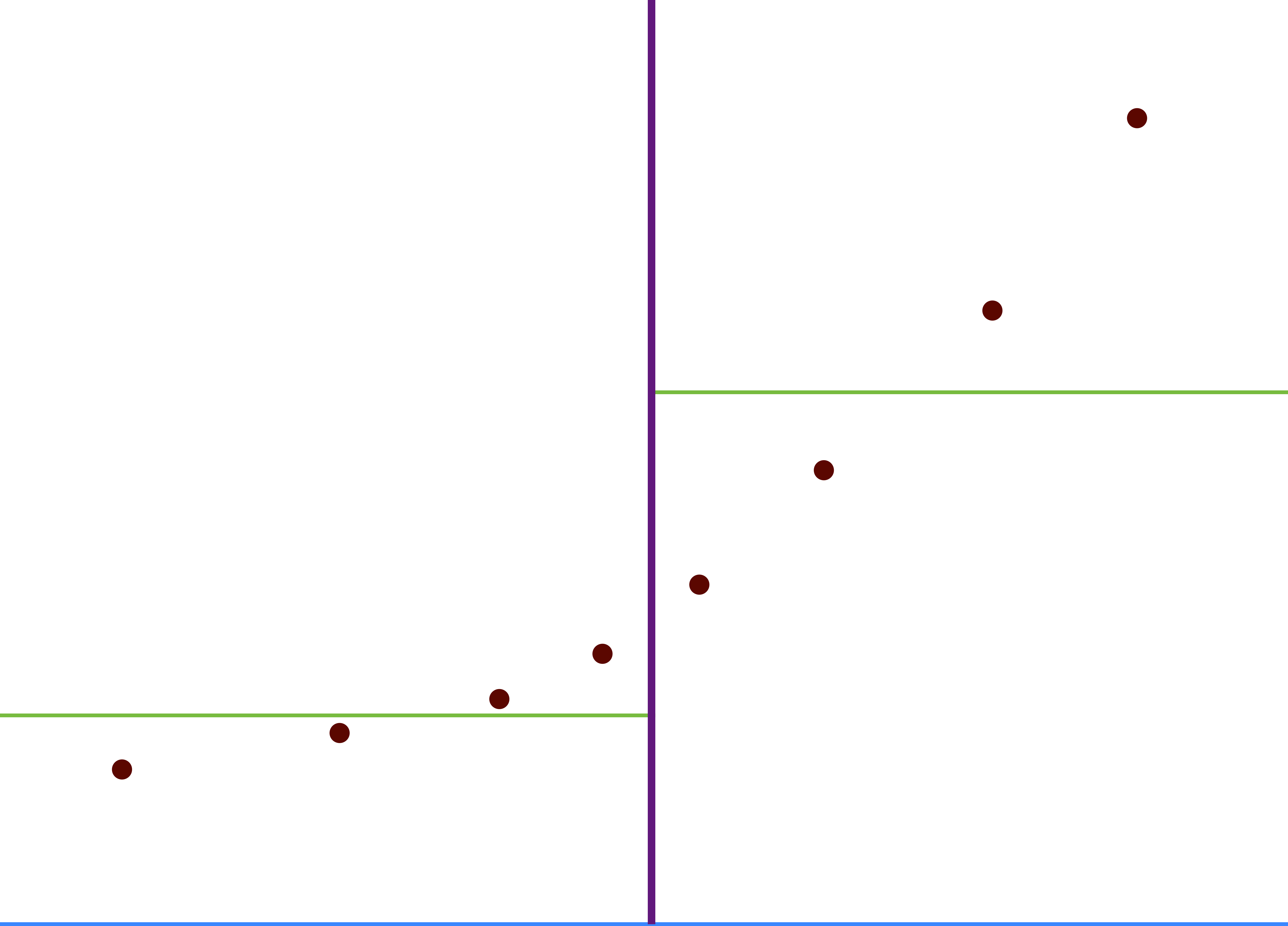


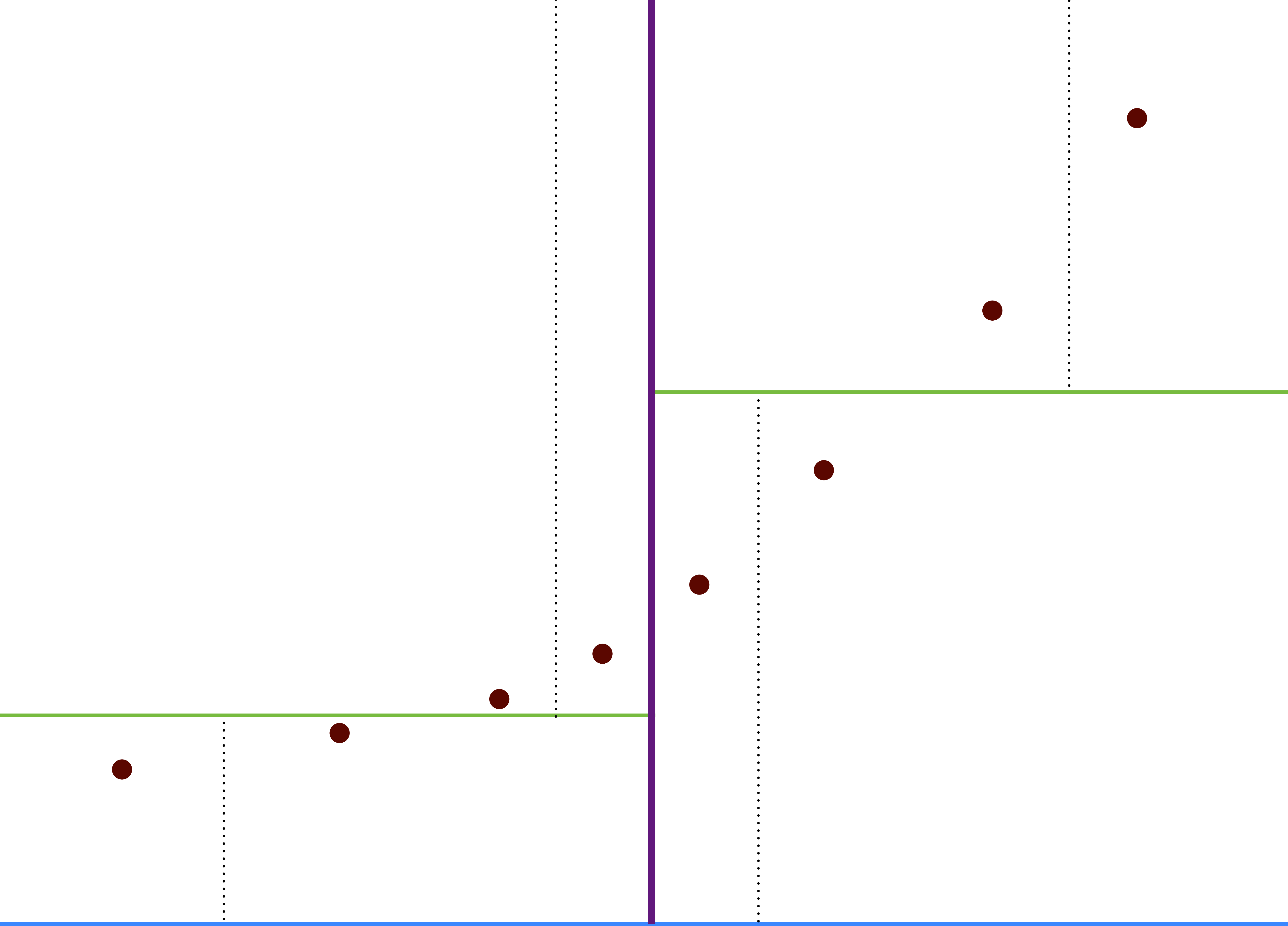


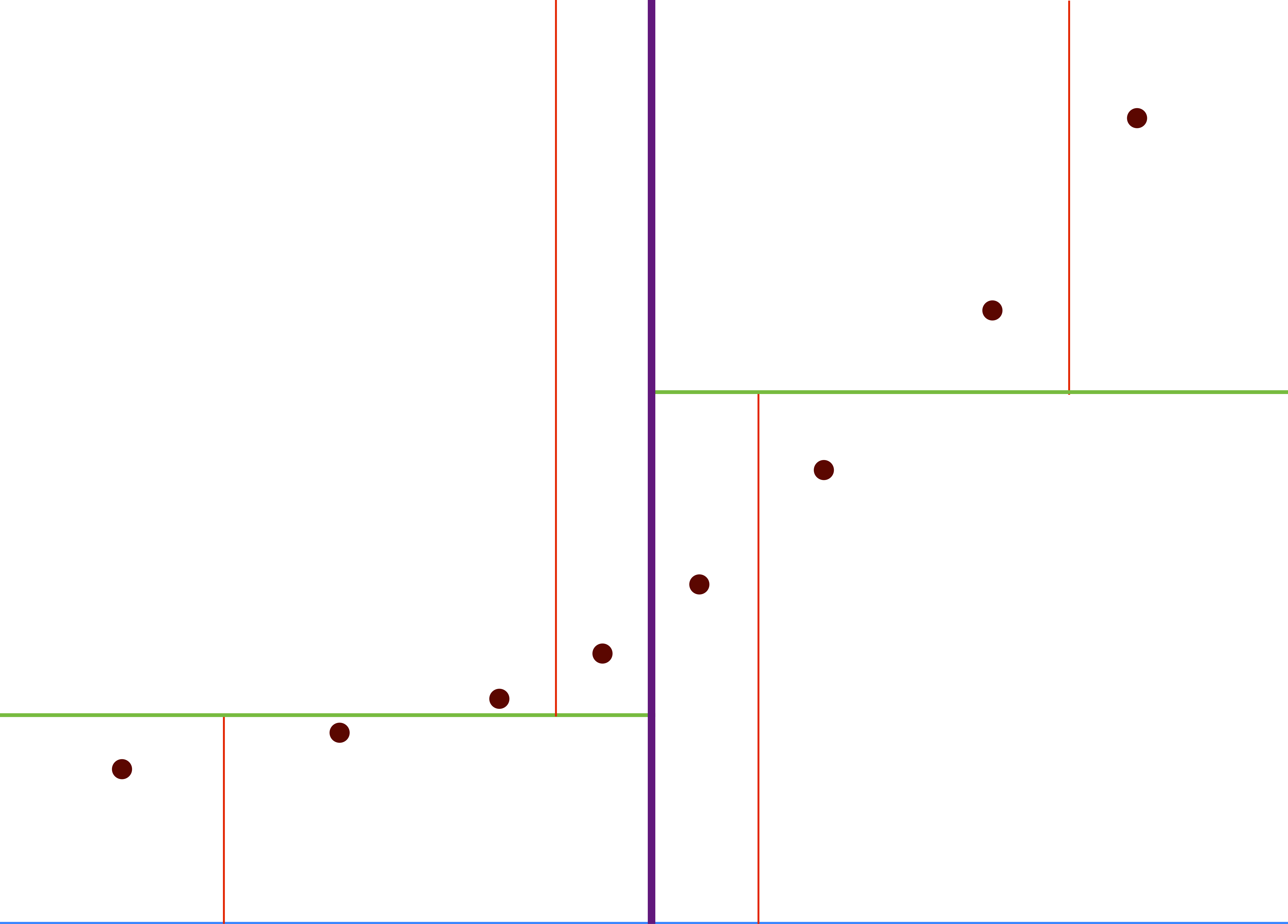


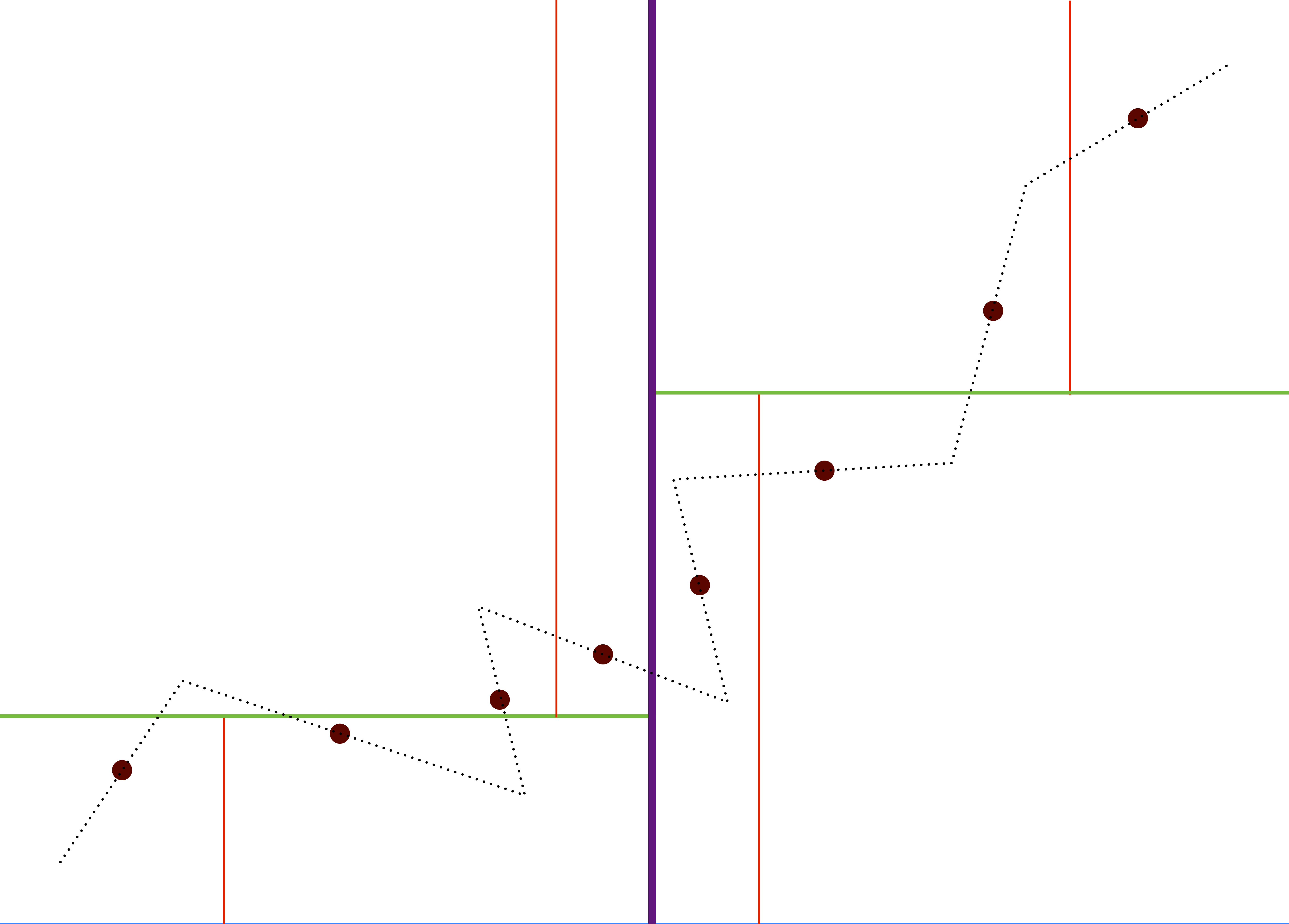


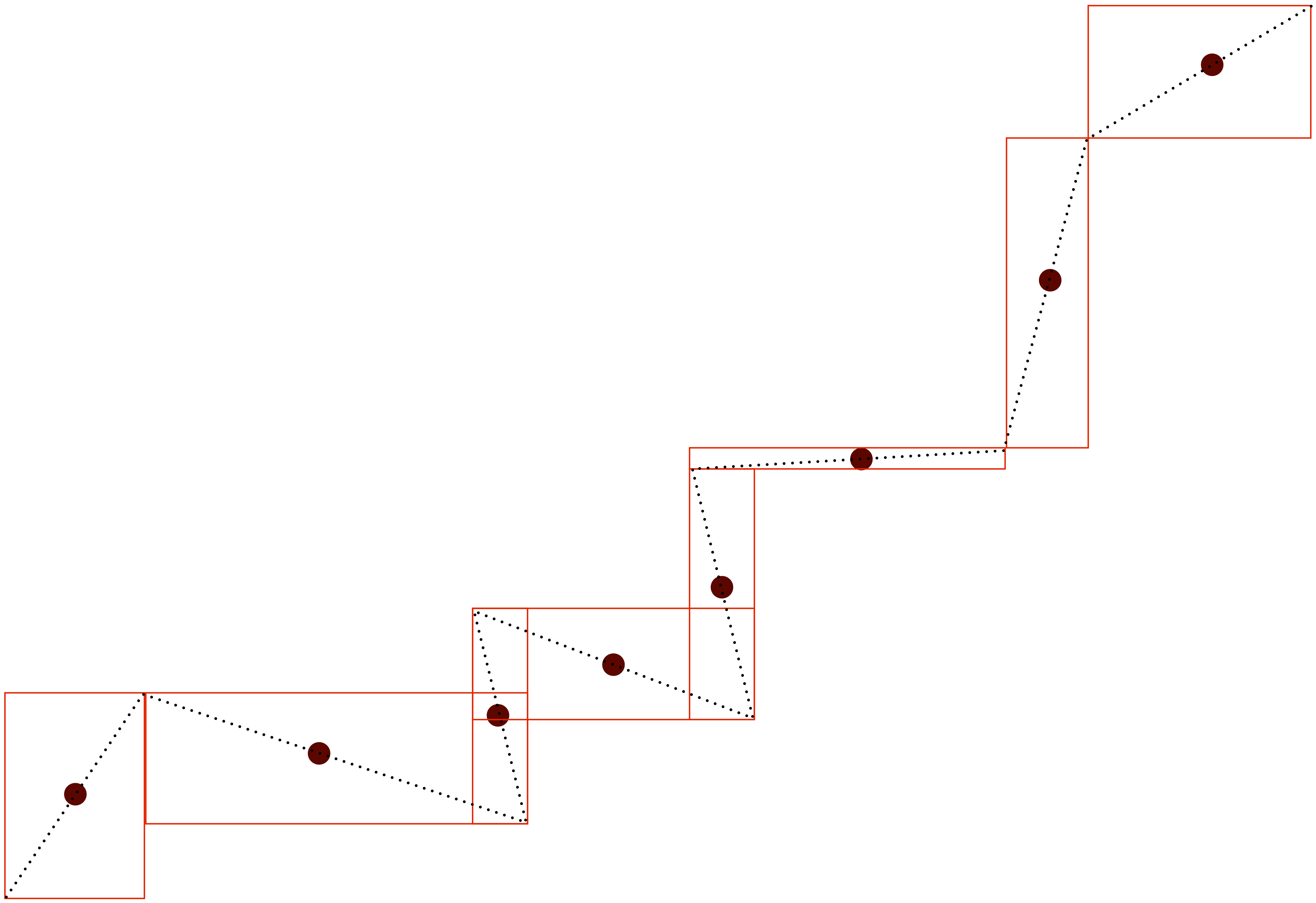


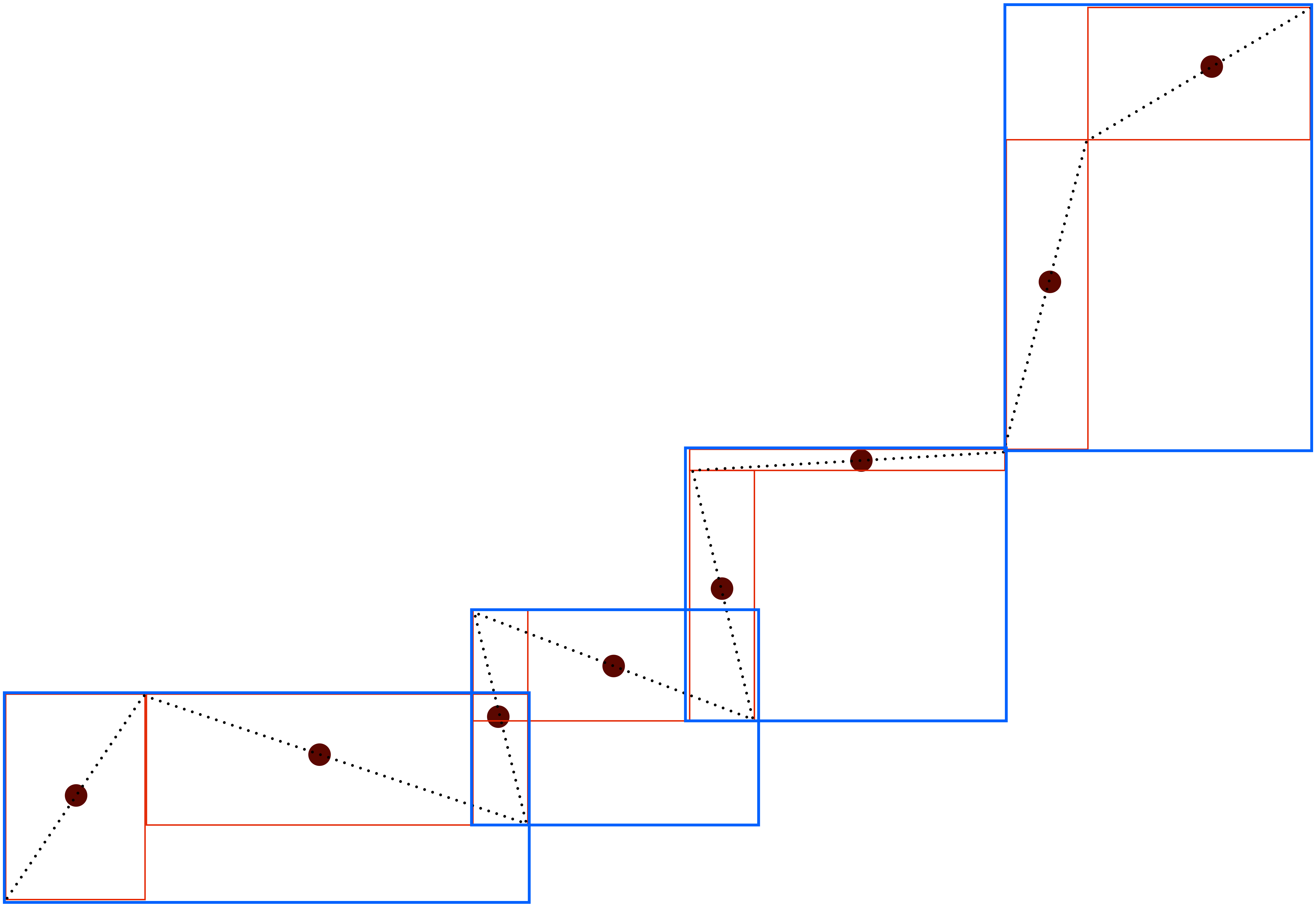


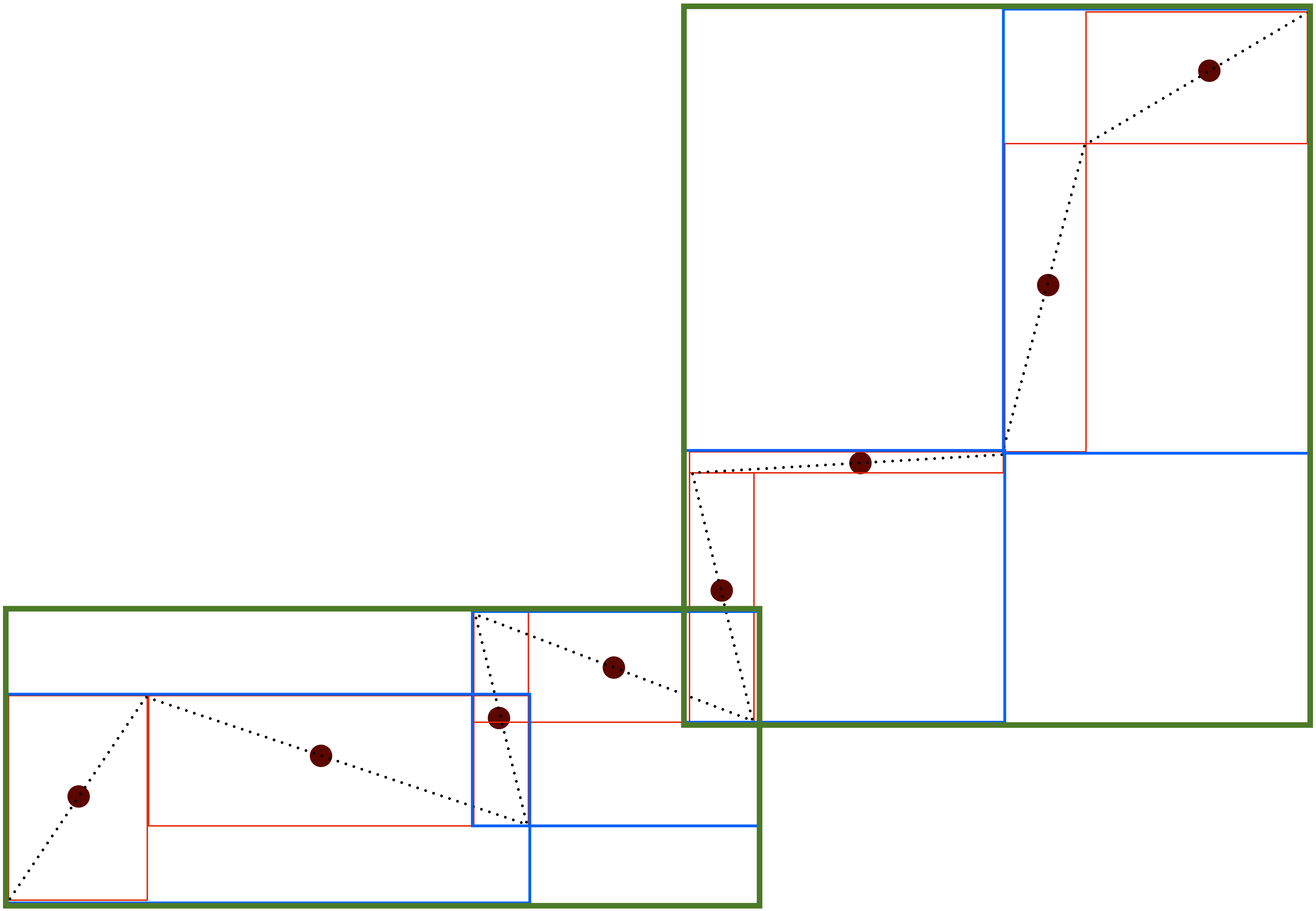


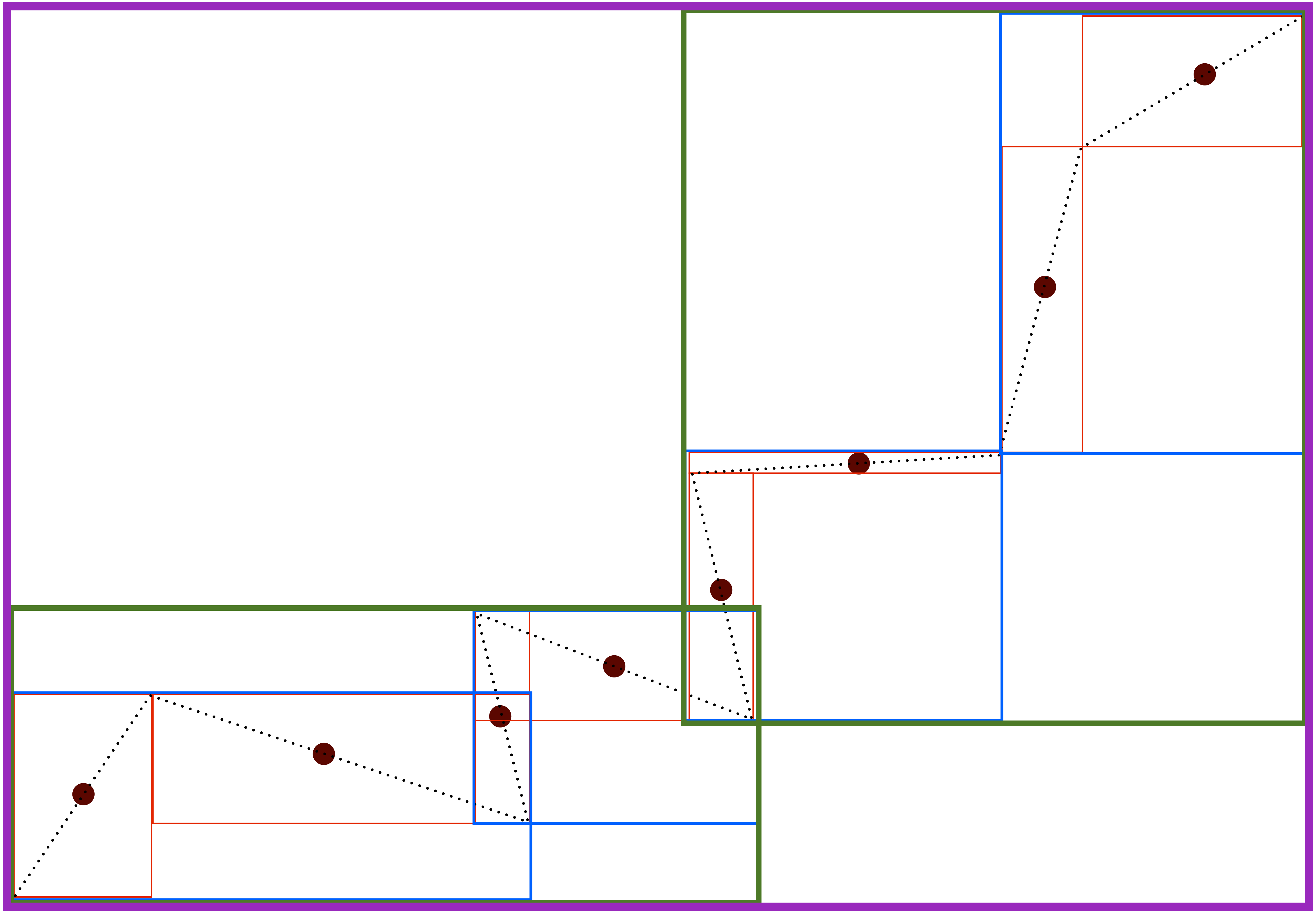


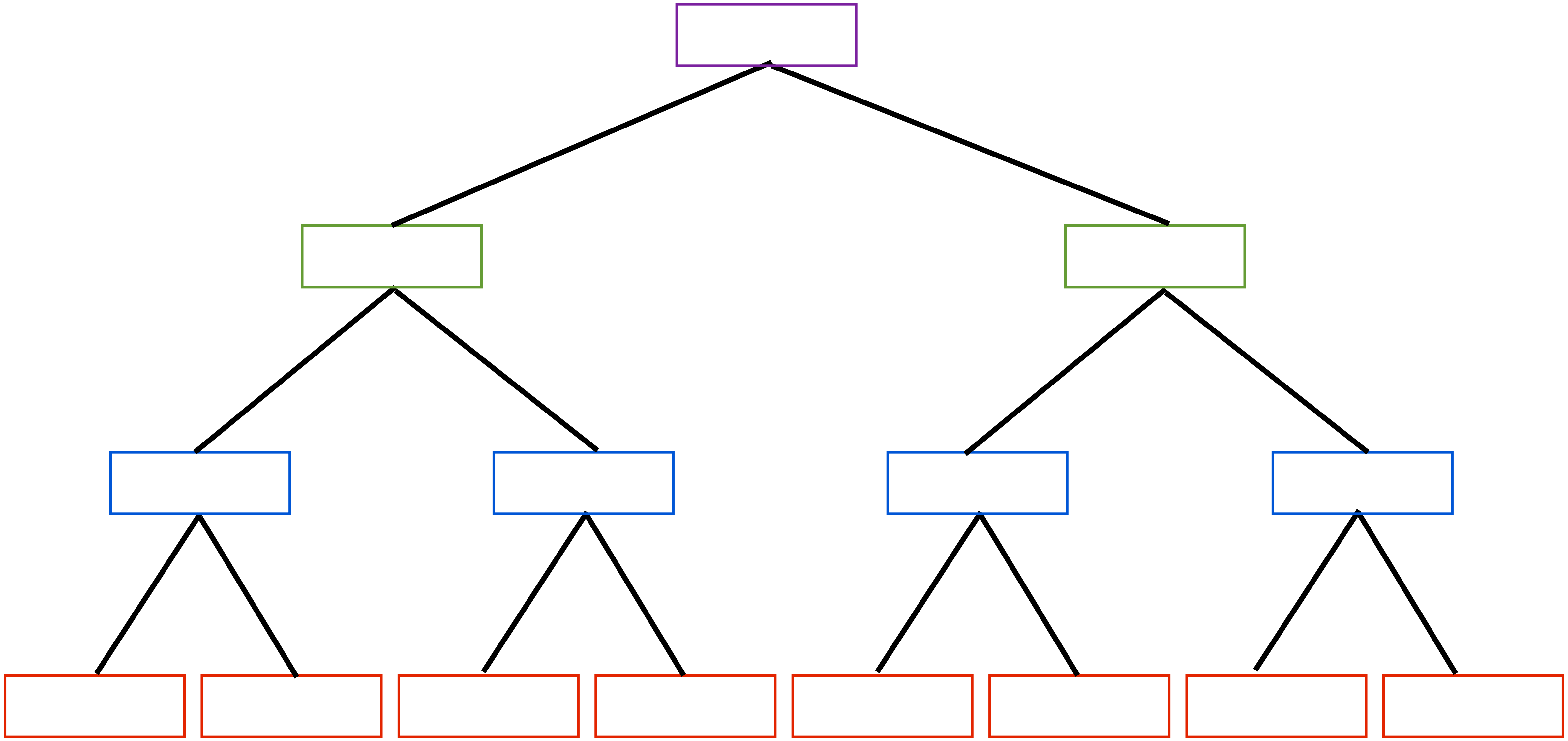












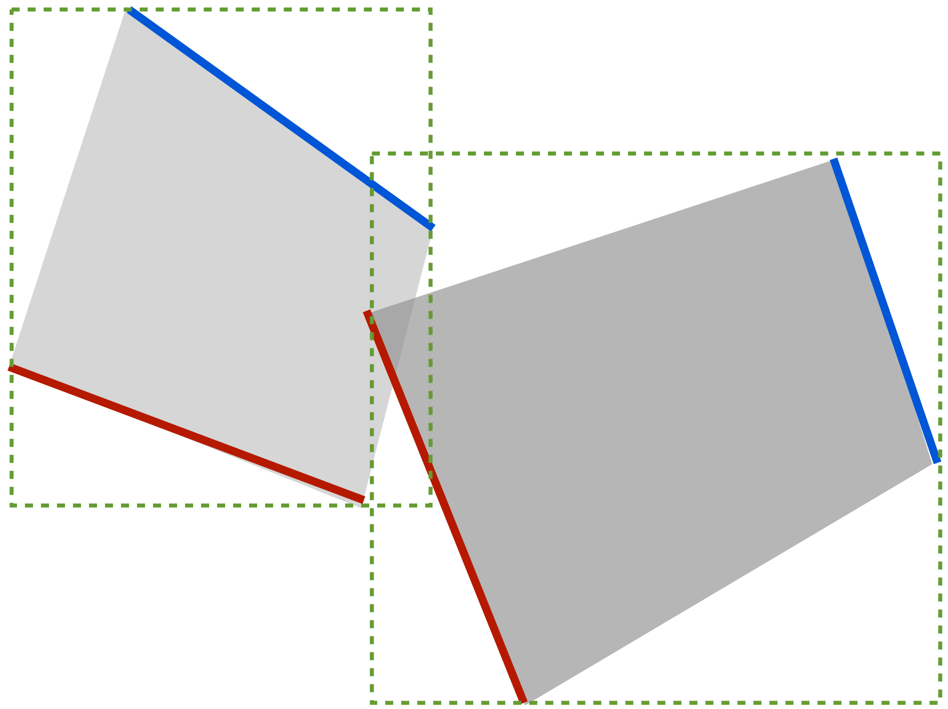
Collision detection (for simulated objects)

- Axis-aligned bounding box (AABB) query structure
  - When the simulated object moves, the AABB tree hierarchy needs to be updated
    - Bounding boxes are updated in a bottom-up fashion
  - When large motions occur the hierarchy efficiency may be compromised :
    - Bounding boxes appear which span large areas, yet contain only very few primitives
    - Violates the efficiency property that only O(1) collisions are found per level of hierarchy (there are many primitives that show up in almost every query)
    - Remedy : Periodically (how often?) rebuild the query hierarchy

Collision detection (for simulated objects)

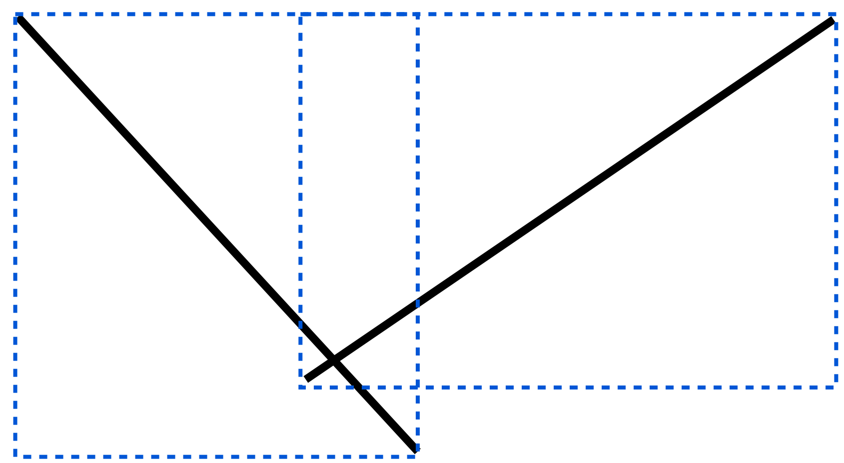
- Axis-aligned bounding box (AABB) query structure
  - Complexity review *(k = # of real collisions)*
    - Construct/rebuild hierarchy (using k-d trees) : O(NlogN)
    - Update after object motion (without rebuild) : O(N)
    - Ideal cost of a single query : O(logN + k)
    - Intersect N primitives with hierarchy : O(N(logN + k))
    - Intersect 2 hierarchies (or one with itself) : O(logN + k)
      - Using simultaneous pairwise traversal of 2 trees
  - Alternatives
    - Quadtrees/Octrees
    - Hashed grids, etc.

Collision detection (for simulated objects)

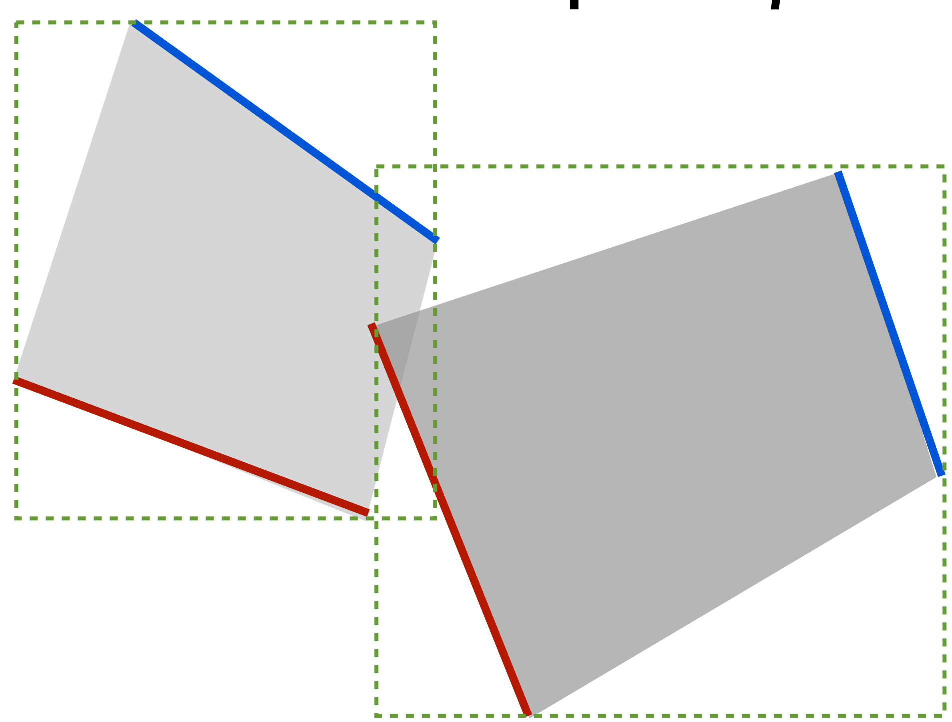


Collision detection (for simulated objects)

- Static detection vs. moving detection
  - For static detection, wrap primitives in AABBs



  - For moving detection, wrap *swept volumes* in AABBs

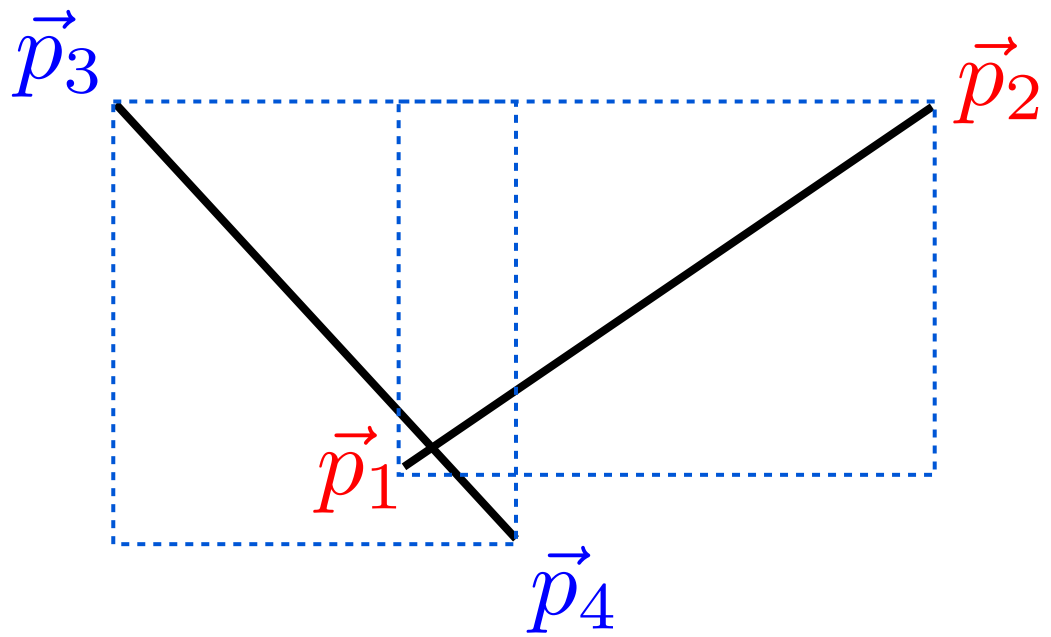


$\text{Blue}: t = t_*$

$\text{Red}: t = t_* + dt$

Collision detection (for simulated objects)

- From AABB collisions to exact collision tests
  - Static collision (e.g. segment-segment collision)



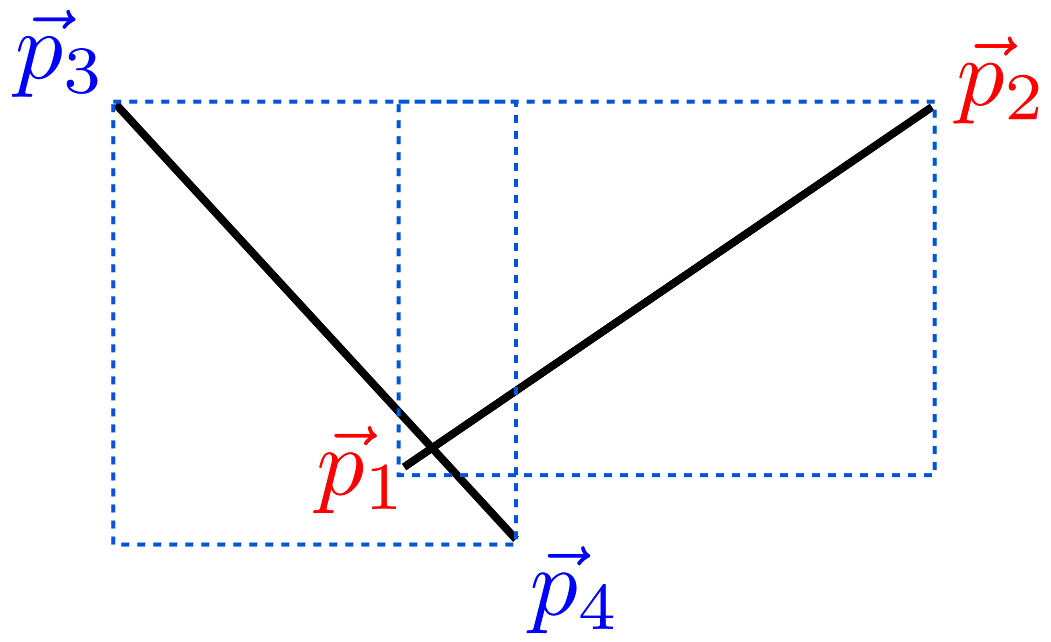
Line through $\vec{p}_1 = (x_1, y1),\ \vec{p}_2 = (x_2, y_2)$

$$\text{Given by : } f(x,y) = \begin{vmatrix} x & y & 1 \\ x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \end{vmatrix} = 0$$

$p_3 p_4$ intersects the line through $p_1 p_2$ iff $f(x_3, y_3) f(x_4, y_4) < 0$

Collision detection (for simulated objects)

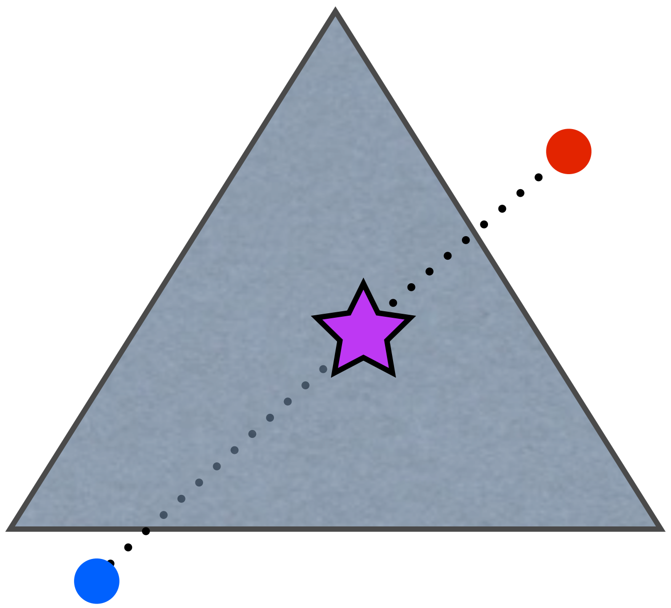
- From AABB collisions to exact collision tests
  - Static collision (e.g. segment-segment collision)

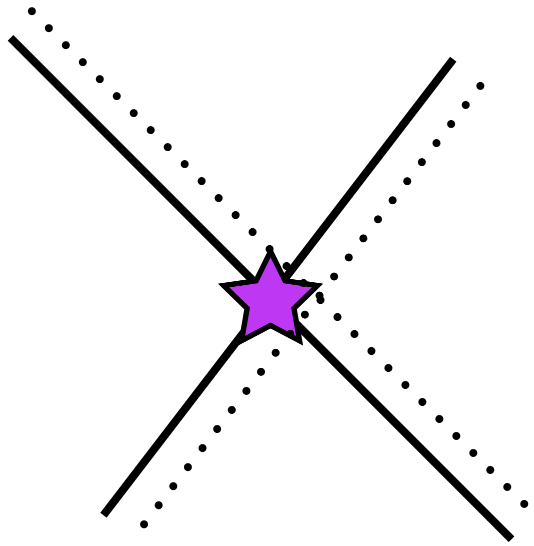


$$p_3p_4 \text{ intersects line through } p_1p_2 \text{ iff } \begin{cases} p_3p_4 \text{ intersects line passing through } p_1p_2 \\ \text{AND} \\ p_1p_2 \text{ intersects line passing through } p_3p_4 \end{cases}$$

Collision detection (for simulated objects)

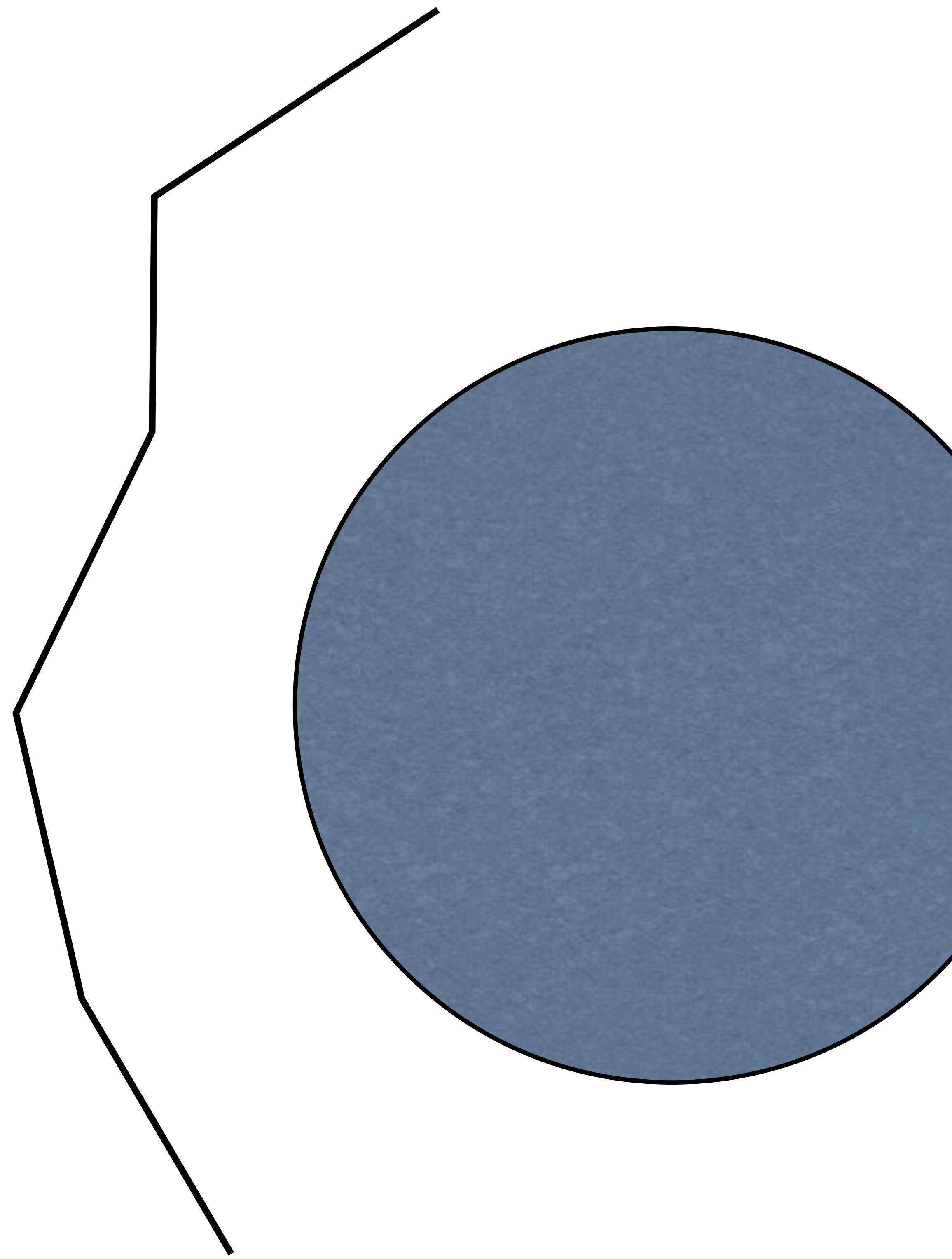
- From AABB collisions to exact collision tests
  - Dynamic collision (e.g. between cloth surfaces)
  - With either triangle-point or edge-segment collision, at the time of contact the four involved vertices become *coplanar*
    - Interpolate moving positions : $p_i(t) = p_i(t_*) + p_i(t_* + dt)\frac{t - t_*}{dt}$
    - Primitives collide when 4 moving points are *coplanar (cubic eqn)*





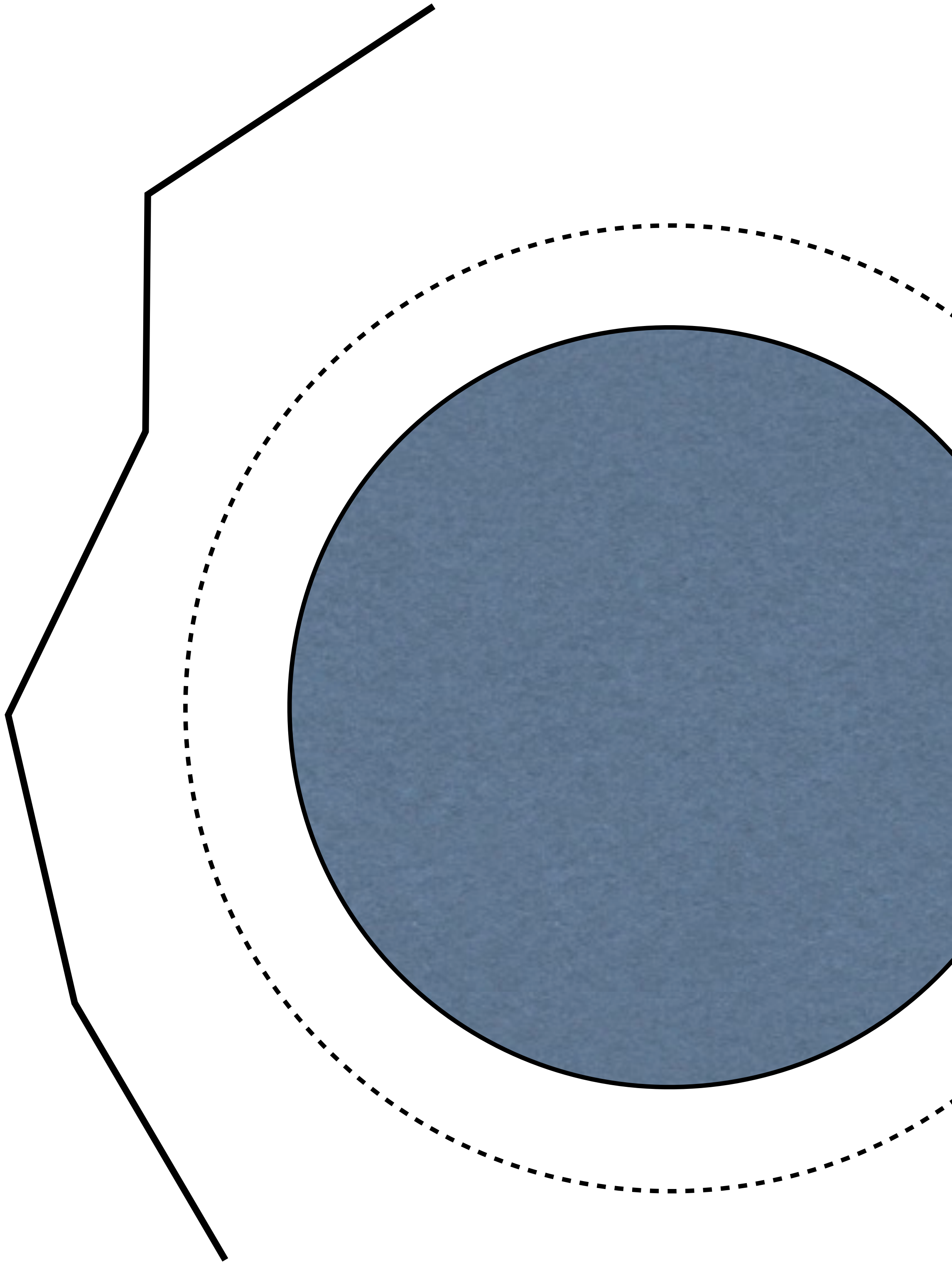
Collision response (general approaches)

- Penalty-based methods
  - Detect proximity to collision objects and apply a repulsive "*penalty*" force when the distance to the collision target is small
  - Increase strength of repulsion force as distance decreases (or as interpenetration starts to occur)



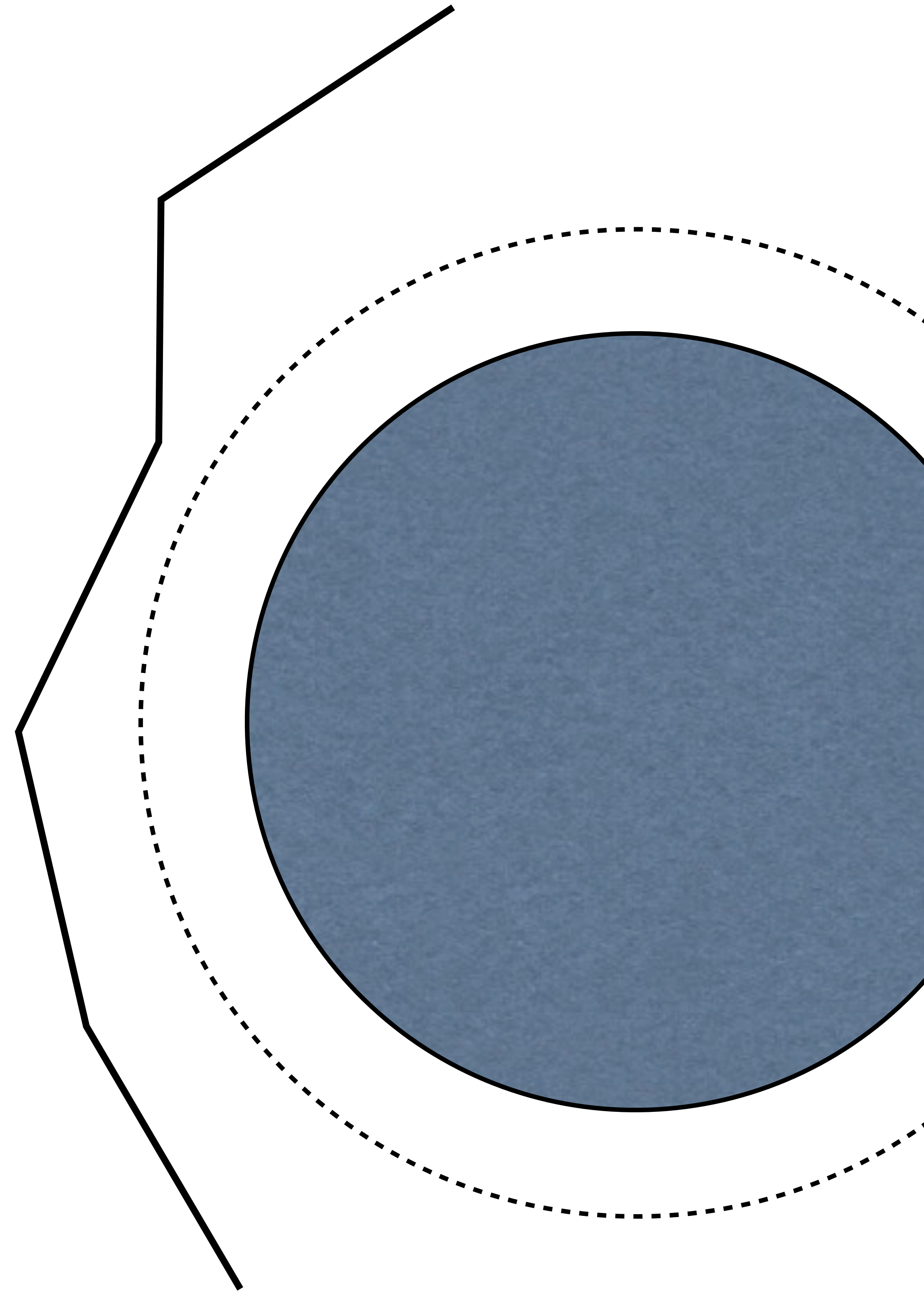
Collision response (general approaches)

- Penalty-based methods
  - Detect proximity to collision objects and apply a repulsive "*penalty*" force when the distance to the collision target is small
  - Increase strength of repulsion force as distance decreases (or as interpenetration starts to occur)



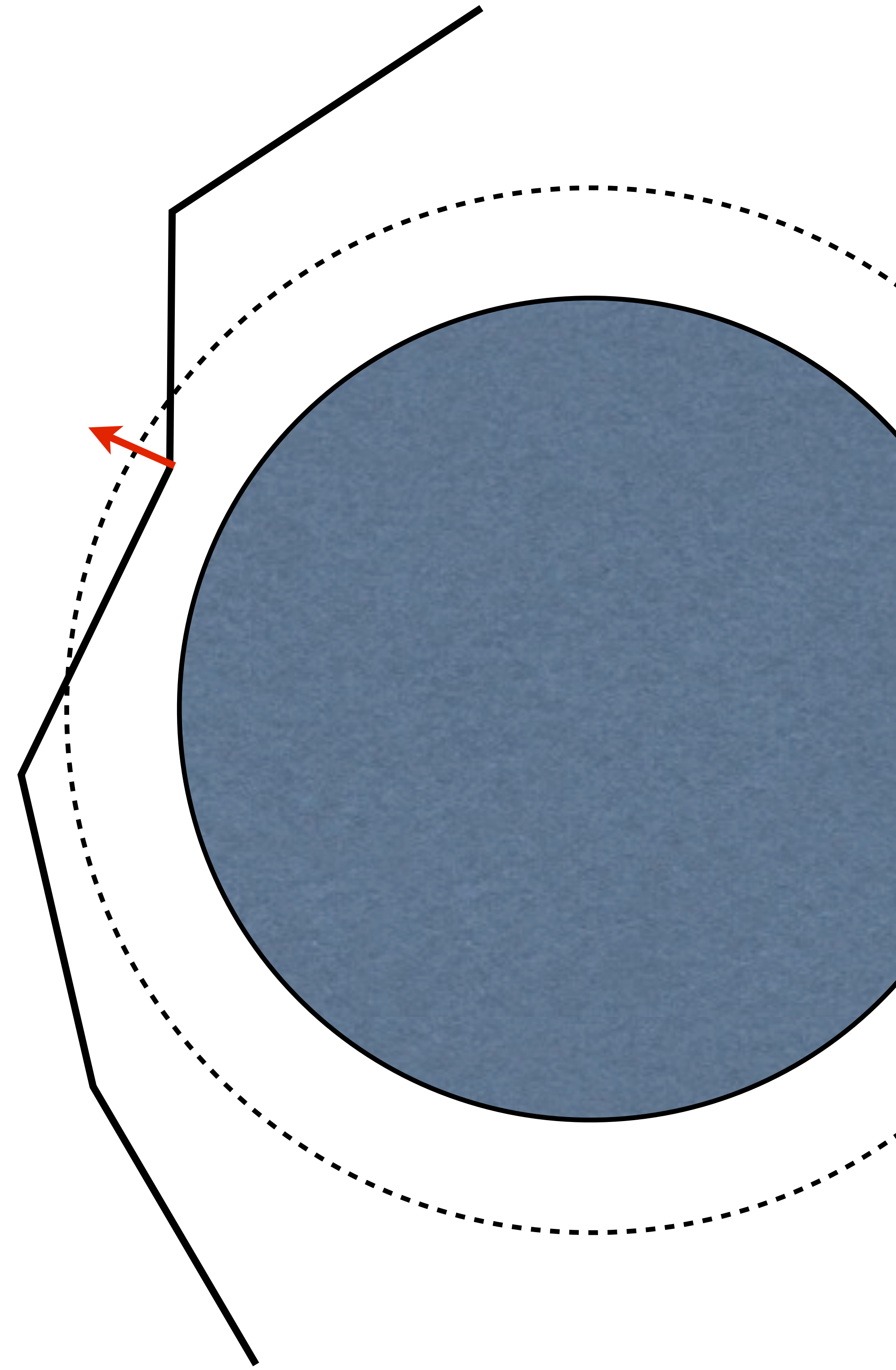
Collision response (general approaches)

- Penalty-based methods
  - Detect proximity to collision objects and apply a repulsive "*penalty*" force when the distance to the collision target is small
  - Increase strength of repulsion force as distance decreases (or as interpenetration starts to occur)



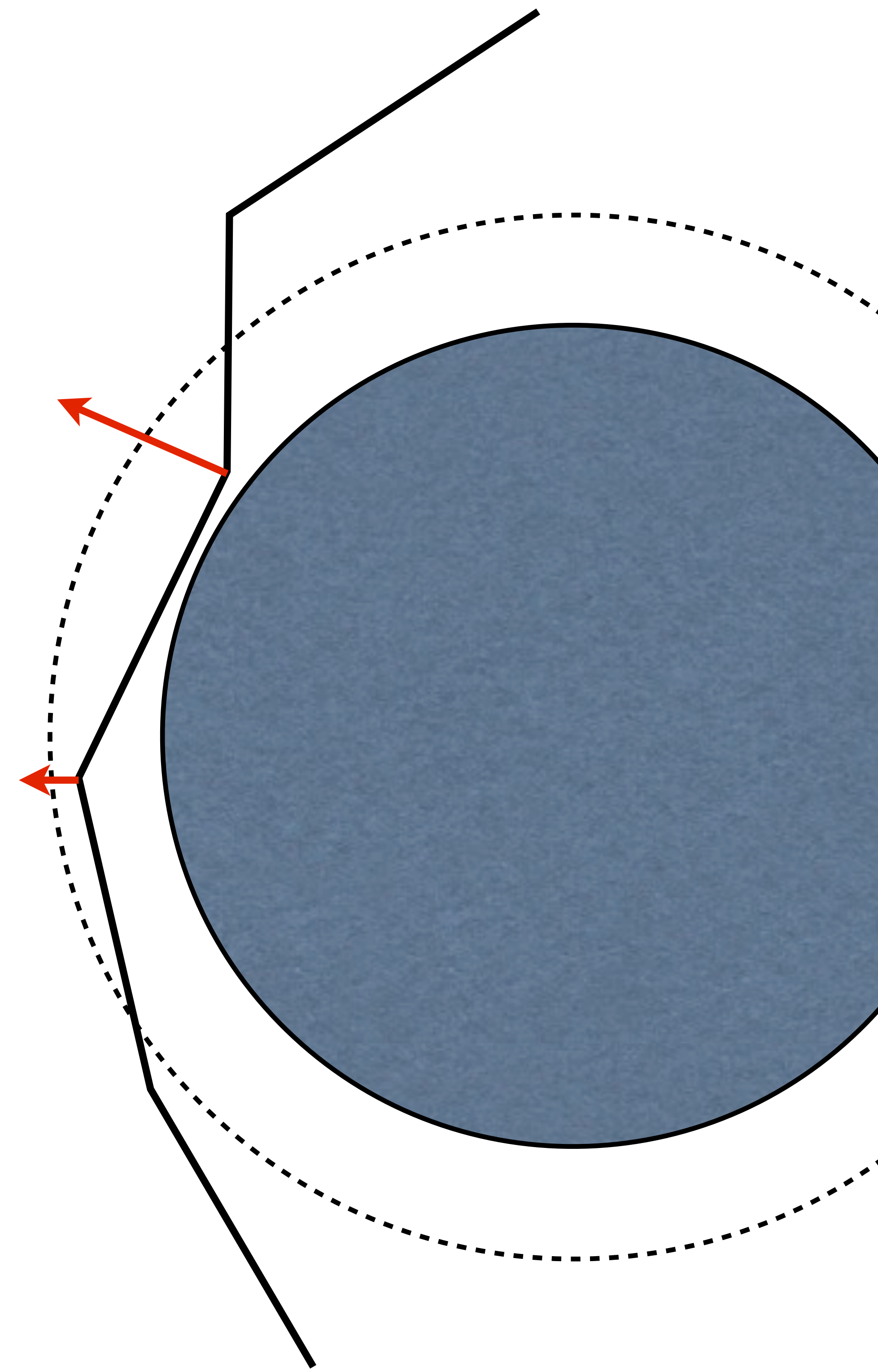
Collision response (general approaches)

- Penalty-based methods
  - Detect proximity to collision objects and apply a repulsive "*penalty*" force when the distance to the collision target is small
  - Increase strength of repulsion force as distance decreases (or as interpenetration starts to occur)



Collision response (general approaches)

- Penalty-based methods
  - Detect proximity to collision objects and apply a repulsive "*penalty*" force when the distance to the collision target is small
  - Increase strength of repulsion force as distance decreases (or as interpenetration starts to occur)



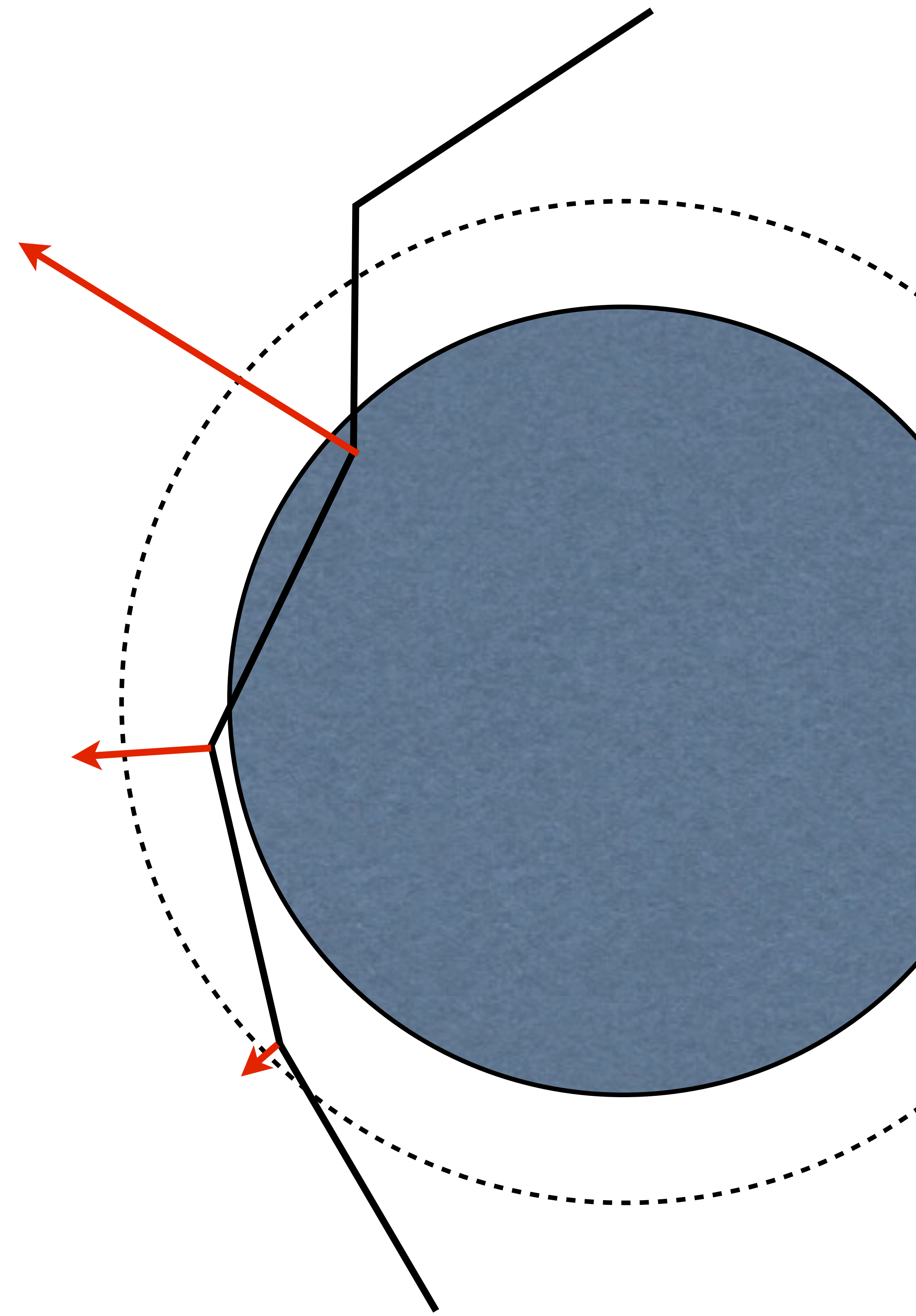
Collision response (general approaches)

- Penalty-based methods
  - Detect proximity to collision objects and apply a repulsive "*penalty*" force when the distance to the collision target is small
  - Increase strength of repulsion force as distance decreases (or as interpenetration starts to occur)

$$\vec{f} = -k \cdot (d - D)\vec{n}$$

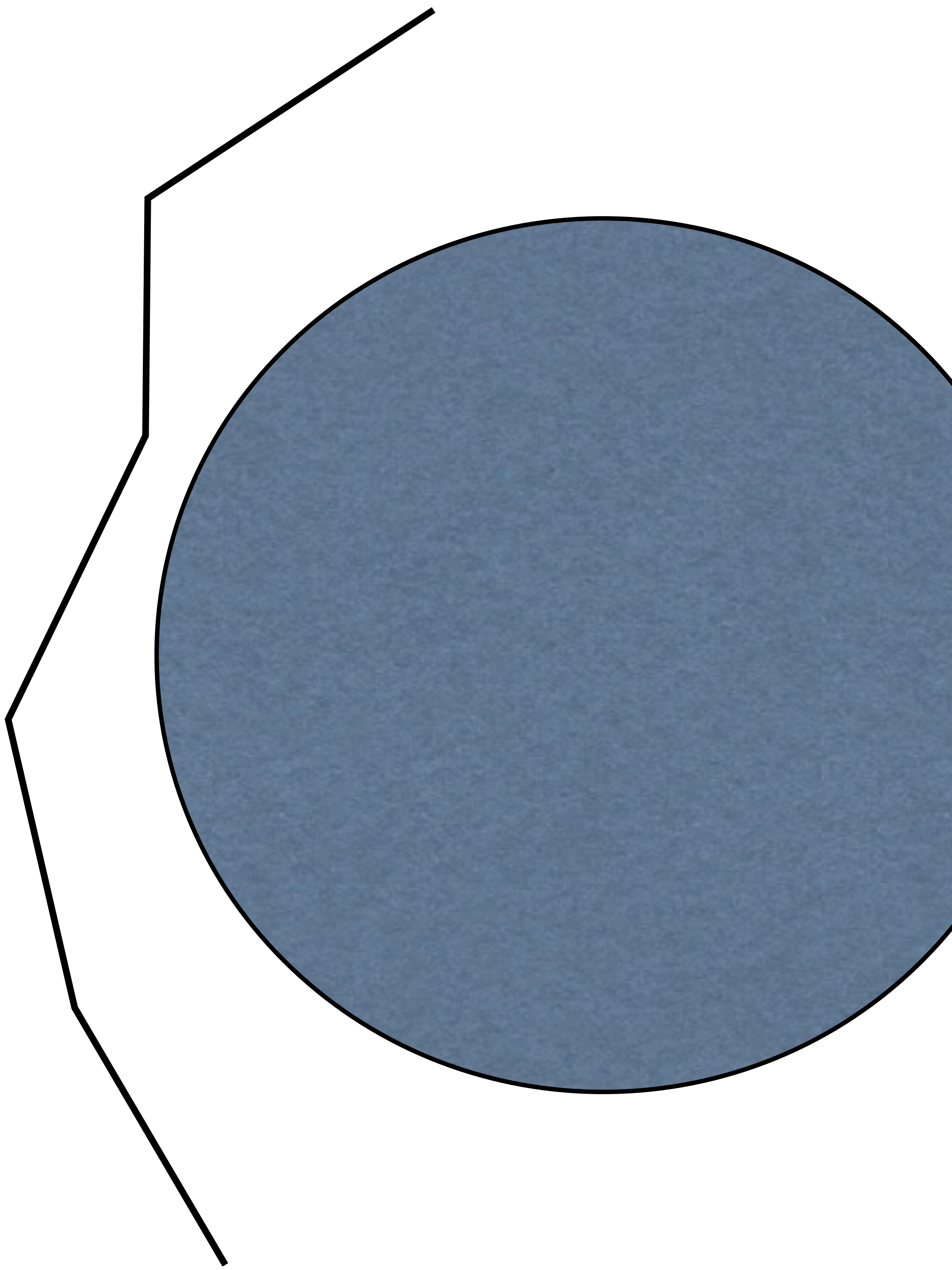
$$\vec{f} = -k \cdot (\phi(\vec{x}) - D)\nabla\phi(\vec{x})$$

$$\vec{f} = k(e^{D-d} - 1)\vec{n}$$



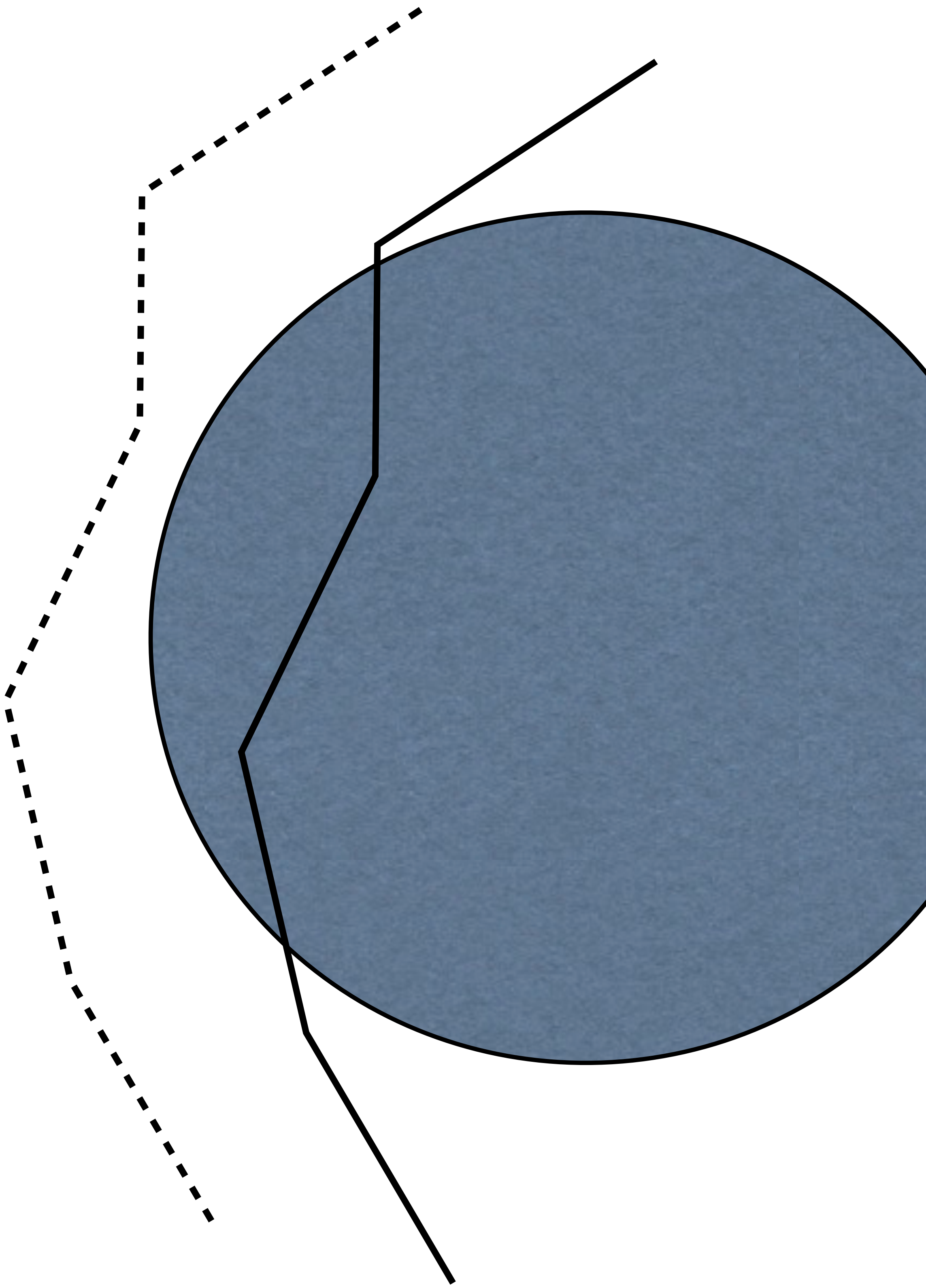
Collision response (general approaches)

- Impulse-based methods
  - Usually attempt to guarantee that *no collision* is produced or left untreated, at any time
  - Starting from a collision-free state at time t*, the system is advanced to time t*+dt
  - Collisions that occurred in the interval [t*,t*+dt] are localized (in space and time)
  - An *impulse* is applied to instantaneously correct the object trajectory and prevent (or fix) any collision events



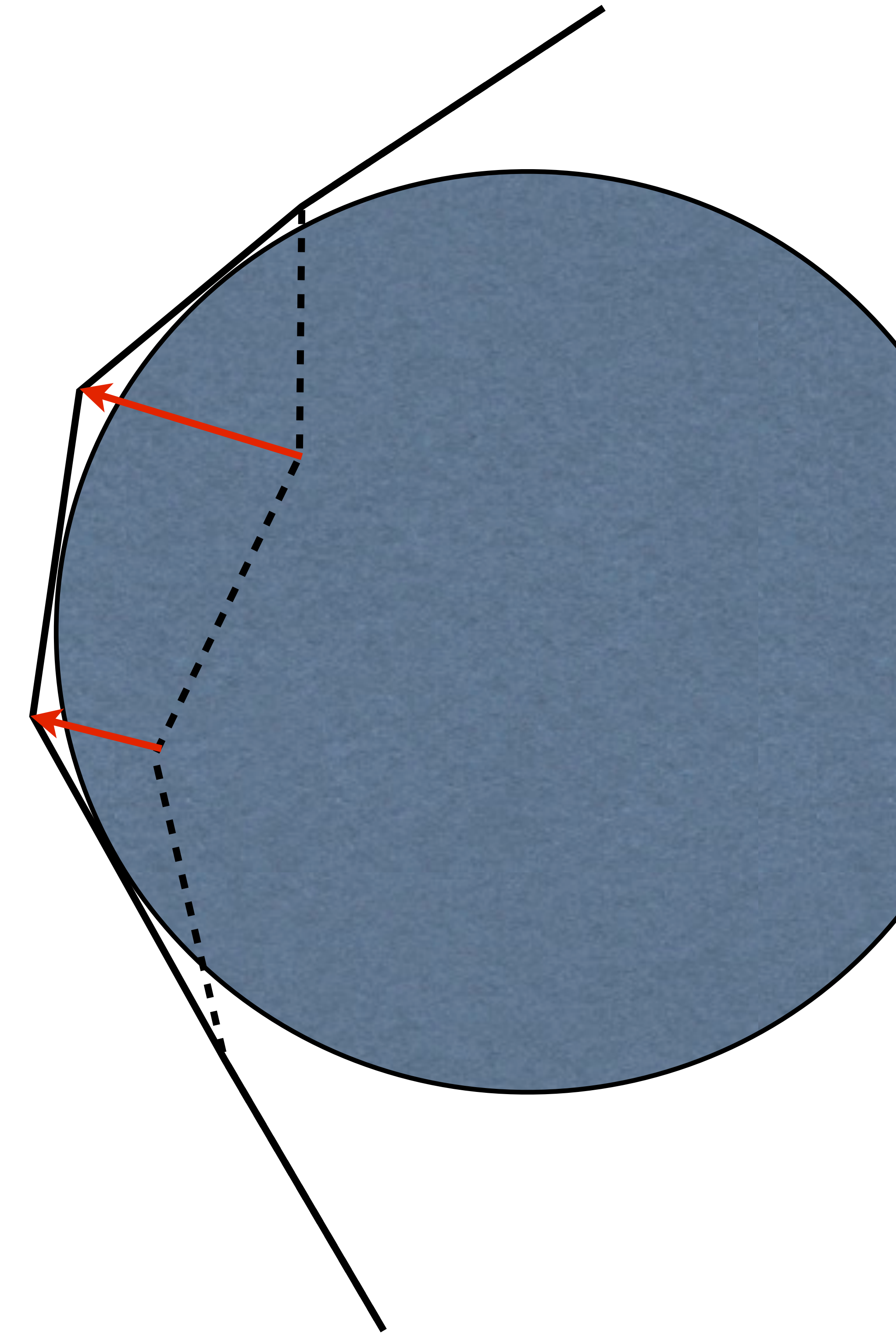
Collision response (general approaches)

- Impulse-based methods
  - Usually attempt to guarantee that *no collision* is produced or left untreated, at any time
  - Starting from a collision-free state at time t*, the system is advanced to time t*+dt
  - Collisions that occurred in the interval [t*,t*+dt] are localized (in space and time)
  - An *impulse* is applied to instantaneously correct the object trajectory and prevent (or fix) any collision events



Collision response (general approaches)

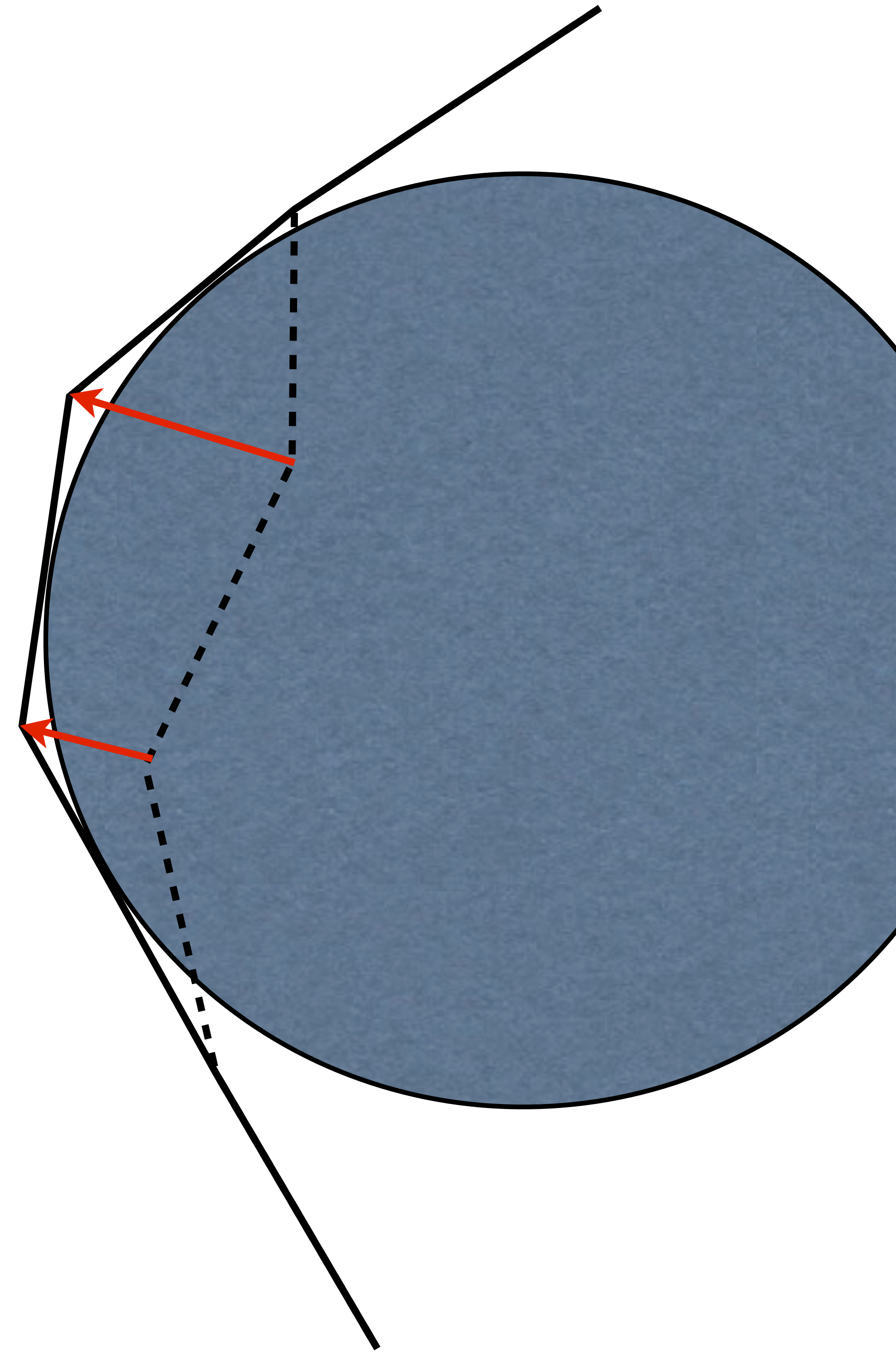
- Impulse-based methods
  - Use time integration scheme to generate *candidate* values for position & velocity at the end of the time step : $x_*^{n+1},\ v_*^{n+1}$
  - Apply an instantaneous correction to fix collision : $(x_*^{n+1},\ v_*^{n+1}) \Rightarrow (x^{n+1},\ v^{n+1})$
  - Check that collision was in fact resolved (in conflicting fixes were used), otherwise retry
  - If attempt at resolving collision was unsuccessful, repeat the time step with a smaller dt



Collision response (general approaches)

- Impulse-based methods
  - May incorporate additional effects (e.g. repulsions, friction) and elaborate time integration *(e.g. [Bridson et al 2002])*

- Select a collision time step size Δt and set $t^{n+1} = t^n + \Delta t$
- Advance to candidate positions $\bar{\mathbf{x}}^{n+1}$ and velocities $\bar{\mathbf{v}}^{n+1}$ at time t^{n+1} with the cloth internal dynamics
- Compute the average velocity $\bar{\mathbf{v}}^{n+1/2} = (\bar{\mathbf{x}}^{n+1} - \mathbf{x}^n)/\Delta t$
- Check $\mathbf{x}^n$ for proximity (section 6), then apply repulsion impulses (section 7.2) and friction (section 7.3) to the average velocity to get $\tilde{\mathbf{v}}^{n+1/2}$
- Check linear trajectories from $\mathbf{x}^n$ with $\tilde{\mathbf{v}}^{n+1/2}$ for collisions (section 6), resolving them with a final midstep velocity $\mathbf{v}^{n+1/2}$ (sections 7.4 and 7.5)
- Compute the final positions $\mathbf{x}^{n+1} = \mathbf{x}^n + \Delta t \mathbf{v}^{n+1/2}$
- If there were no repulsions or collisions, set $\mathbf{v}^{n+1} = \bar{\mathbf{v}}^{n+1}$
- Otherwise, advance the midstep velocity $\mathbf{v}^{n+1/2}$ to $\mathbf{v}^{n+1}$ (section 7.6)



Collision response (general approaches)

- Impulse-based methods
  - May incorporate additional effects (e.g. repulsions, friction) and elaborate time integration *(e.g. [Bridson et al 2002])*
  - For collisions with kinematic objects, impulses are defined such that the simulated is snapped to the surface of the colliding body
  - In the case of cloth collisions impulses are defined to mimic inelastic momentum exchange
  - Preventive repulsions modeled either as forces or impulses